The Double Entry

The Double Entry

Why Every Structure That Recovers
Also Verifies

Tal Zisckind

Rhombus Tech

Copyright © 2026 Tal Ziskind

All rights reserved.

Every quantitative claim in this book is reproduced by code in the `code/` directory. Run the scripts; check the numbers.

For Fra Luca Pacioli,
who wrote down the method of Venice
and did not know he had written a law of nature.

All the affairs of the merchant must be noted systematically, and with that order which is proper to a diligent merchant.

Luca Pacioli, *Particularis de Computis et Scripturis* (1494)

The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point.

Claude Shannon, *A Mathematical Theory of Communication* (1948)

Preface

Every system that can survive loss can detect corruption. The two powers cost the same, to within 39%. This book proves it.

The result is a single equation, $e \cdot s = g(e) \cdot \lambda$, derived from elementary probability. It says that a system's erasure tolerance e and its verification cost s are locked together by a near-constant function g that varies by less than 39% across the entire range of possible redundancy levels. The individual results the equation connects—the Singleton bound, the geometric distribution, the audit sampling formula—are classical. What is new is the synthesis: the observation that these results describe a single universal duality between recovery and verification.

The duality also has a dark side. A system with no structural constraints, where checking one component reveals nothing about whether another has failed, has zero "verification entropy" ($H_v = 0$), and no amount of monitoring can warn it. That is why compound failures go undetected until they happen. (Compound failures are the subject of the first book in this series, *The Rogue Wave*; this book stands on its own.)

The consequences are practical. The equation tells engineers what verification should cost, tells auditors why their sampling formulas work, and tells risk modelers why some failures are invisible until they happen.

Who this book is for

Anyone who designs, audits, or depends on systems that must not fail. The mathematics requires comfort with logarithms and basic probability; everything else is developed from scratch. Every chapter opens with the question it answers and closes with a boxed summary, *What to remember*, so a reader can skim the proofs and keep the argument. A glossary at the back defines every symbol and term. The evidence chapters test the law against published auditing standards, real error-correcting codes, production data availability systems, Byzantine consensus, DNA replication, quantum error correction, neural networks, and human institutions, with every number backed by code or citation.

How to read it

Part I states the law and its proof. Part II tests it against eight domains. Part III explains the mechanism: why the duality arises, where the factor of two comes from, how composition improves it, and what the membership threshold means. Part IV connects the duality to the problem of invisible risk, shows what blindness looks like when real systems fail, provides a method for finding your own blind spots, draws the design frontier, and extends the conjecture to spacetime.

A reader in a hurry can read Chapters 1, 3, and 21 for the idea, the proof, and the punchline. A reader who wants the practical payoff rather than the proof can read Chapters 1, 16, 17, 19, and 21.

Sourcing

Every quantitative claim traces to a derivation reproduced in `code/` or a cited external source with a specific reference. Unresolved claims are flagged in `NUMBERS.md` and excluded from the text. The standard is: if a number appears in the book, a reader can verify it.

T.Z., November 2026

Contents

Preface **vii**

I The Discovery **1**

1 Pacioli's Ledger **3**

1.1 What Pacioli could not have known . . . 4

1.2 The law in plain terms . . . 4

1.3 One structure, two powers . . . 5

2 Shannon's Other Theorem **7**

2.1 The cost of verification . . . 7

2.2 The near-constant product . . . 8

2.3 The constant $2 \ln 2$ . . . 9

3 The Law **11**

3.1 Statement . . . 11

3.2 Proof . . . 12

3.3 The bounds on g . . . 12

3.4 What the law assumes . . . 13

3.5 What the law is not . . . 13

3.6 The general form . . . 14

Interlude: One Equation . . . 16

II The Evidence **17**

4 The Auditor's Formula **19**

4.1 The identity . . . 19

4.2 Testing against published tables . . . 20

4.3 What this chapter proves and does not prove . . . 21

5 Codes and Sampling **23**

5.1 Setup . . . 23

5.2 Results: identity testing . . . 23

5.3 Results: membership testing . . . 24

5.4 The total cost of membership verification . . . 24

5.5 Confirmation for the law . . . 25

6 Data Availability **27**

6.1 Plain erasure sampling . . . 27

6.2 Tensor ZODA . . . 28

7 Byzantine Generals **31**

7.1 Recovery and verification . . . 31

7.2 The two models . . . 32

7.3 What BFT adds . . . 32

8 The Error Catastrophe **33**

8.1 Three layers, one principle . . . 33

8.2 Composition . . . 34

8.3 Eigen's threshold and the security parameter . . . 34

8.4 What DNA adds . . . 35

8.5 On the frontier . . . 35

9 The No-Cloning Duality **37**

9.1 Stabilizer codes . . . 37

9.2 The duality . . . 38

9.3 Why no-cloning forces the double entry . . . 38

9.4 The quantum Singleton bound . . . 38

9.5 Numerical confirmation . . . 39

10 The Weights That Learn **41**

10.1 The experiment . . . 41

10.2 Results . . . 42
10.3 Why trained weights are constrained . . . 43
10.4 The lottery ticket connection . . . 43
10.5 What neural networks add . . . 43
10.6 What the chapter does not show . . . 44

11 The Institutional Double Entry **45**

11.1 Separation of powers . . . 45
11.2 Independent replication in science . . . 46
11.3 Supply chain diversification . . . 47
11.4 The institutional summary . . . 47
11.5 What institutions add . . . 48
Interlude: Convergence . . . 49

III The Mechanism **51**

12 One Pool, Two Operations **53**

12.1 The constraint pool . . . 53
12.2 Why you cannot have one without the other . . . 53
12.3 The formal equivalence . . . 54
12.4 The accounting perspective . . . 55
12.5 Pacioli's ledger, by the numbers . . . 55
12.6 Constraints as information . . . 56

13 The Factor of Two **59**

13.1 Erasures vs. errors . . . 59
13.2 The two detection regimes . . . 60
13.3 Worked example: Tegridy RS(255,128) . . . 60
13.4 Is the factor exactly two? . . . 61
13.5 Non-MDS codes and list decoding . . . 62
13.6 Landauer's ln 2 . . . 62

14 Composition **65**

14.1 Tensor product codes . . . 65
14.2 The locality advantage . . . 65
14.3 Erasure tolerance under composition . . . 66
14.4 Verification under composition . . . 66
14.5 The composition law . . . 66
14.6 Why ZODA uses two levels . . . 67

15 The k Threshold **69**

15.1 The information-set barrier . . . 69
15.2 Two costs of verification . . . 69
15.3 How tensor codes eliminate the basis cost . . . 70
15.4 When does the basis cost matter? . . . 70
Interlude: What Is Missing . . . 72

IV The Connection **73**

16 The Blind Spot **75**

16.1 Verification entropy of independence . . . 75
16.2 The compound blindness . . . 76
16.3 The contrast . . . 76
16.4 Partial coverage: the realistic case . . . 77
16.5 The Rogue Wave connection . . . 77

17 When Systems Go Blind **79**

17.1 The Gaussian copula (2008) . . . 79
17.2 Challenger's O-rings (1986) . . . 80
17.3 Therac-25 (1985–1987) . . . 80
17.4 Boeing 737 MAX (2018–2019) . . . 81
17.5 When constraints save . . . 82
17.6 The pattern . . . 82

18 The Frontier **85**

18.1 Where real systems sit . . . 85

18.2 Computed positions . . . 86
18.3 Moving along the frontier . . . 87
18.4 Moving above the frontier: composition . . . 87
18.5 Design implications . . . 87
18.6 Predictions . . . 88
18.7 Beyond sampling: algebraic verification . . . 88

19 The Blind-Spot Audit **91**
19.1 Example: home fire safety . . . 91
19.2 Example: replicated database . . . 92
19.3 Example: portfolio risk model . . . 93
19.4 The method . . . 93
19.5 Designing the constraint . . . 94

20 The Holographic Double Entry **97**
20.1 Recovery and verification in holography . . . 97
20.2 The HaPPY code . . . 98
20.3 The conjecture . . . 98
20.4 What holography means for the duality . . . 99

21 The Double Entry **101**
21.1 What the law gives us . . . 101
21.2 What the law does not give us . . . 102
21.3 The objection and the answer . . . 103
21.4 The temporal frontier . . . 103
21.5 The title, revisited . . . 104

Afterword **105**

Glossary **107**

Notes **113**

Part I

The Discovery

A Franciscan friar, a Venetian ledger, and a five-hundred-year-old equation that governs every system with structural constraints.

Chapter 1

Pacioli's Ledger

The question: What did a Franciscan friar discover in 1494, and why does it still run the world?

In 1494, a Franciscan friar named Luca Pacioli published a 615-page encyclopedia of mathematics in Venice (Pacioli, 1494). The *Summa de Arithmetica, Geometria, Proportioni et Proportionalita* covered everything a merchant or scholar might need: arithmetic, algebra, currency exchange, the geometry of commercial weights. Tucked inside it, spanning just twenty-seven pages, was a section titled *Particularis de Computis et Scripturis*—"Details of Calculation and Recording."

Those twenty-seven pages changed the world more than the other five hundred and eighty-eight combined.

Pacioli did not claim to have invented what he described. He called it *the method of Venice*—the bookkeeping system that Venetian merchants had developed over the preceding two centuries to track the most complex commercial enterprise on Earth. Florentine bankers had used variants since at least the thirteenth century. The merchant Benedetto Cotrugli had described it in a manuscript of 1458 (Cotrugli, 1458), but that manuscript stayed unpublished for over a century. What Pacioli did was write it down, clearly, in a book that could be printed and distributed across Europe.

The method was simple. Every transaction is recorded twice: once as a debit, once as a credit. The two entries must balance. If you buy silk for ten ducats, your silk inventory rises by ten and your cash falls by ten. The books must always sum to zero.

This single constraint—*every debit has a credit*—gives you two things at once.

First, it gives you **recovery**. If a page of your ledger is destroyed by fire, water, or careless handling, you can reconstruct the missing entries from the other side of each transaction. The credit entries tell you what the debits were. The constraint is a backup that doesn't require a second copy.

Second, it gives you **verification**. If a clerk miscopies one side of an entry—or tampers with it—the books stop balancing. The constraint is an alarm that fires whenever one side goes wrong. You don't have to check every entry against the original receipts. You check the balance. If it doesn't hold, something is wrong, and the size of the imbalance tells you where to look. If it holds, no one-sided error has occurred: no entry was posted without its partner, and no amount was written on one side but miscopied on the other. (A balance cannot catch everything. A transaction omitted entirely, or posted in full to the wrong account, leaves the books in balance. Those errors are what auditors sample for, and Chapter 4 counts the cost.)

Recovery and verification. From one constraint. Not two systems—one system that does two things.

This is not a coincidence specific to bookkeeping. It is a law.

When Pacioli wrote down the method of Venice, he gave the first printed account of a principle that appears to be universal. We will prove it for algebraic codes and find evidence for it across several other domains. The principle: *any structural constraint that makes a system recoverable also makes it verifiable, and at a cost determined by a single equation with no free parameters.*

The constraint need not be "every debit has a credit." It can be a parity check in a communication code, a quorum rule in a distributed database, a row-codeword check in a data availability scheme. The specific structure varies. What does not vary is the duality: the same constraint that lets you recover from loss also lets you detect corruption.

This book is about that duality—where it comes from, what it costs, and what happens to systems that don't have it.

1.1 What Pacioli could not have known

The quantitative form of the law requires ideas that would not exist for another four and a half centuries: Claude Shannon's information theory (1948), Richard Singleton's bound on error-correcting codes (1964), and the proximity gap theorems for algebraic codes (Ben-Sasson et al., 2020). Pacioli had the instance. The general law waited for the mathematics.

Here is what the mathematics says.

Every system with structural constraints has a quantity called *erasure tolerance*—the fraction of the system that can be lost and still reconstructed. Call it e. The same system has a *verification cost*—the number of random checks needed to detect corruption with confidence $2^{-\lambda}$, where λ is the security parameter (how many bits of certainty you require). Call that cost s.

The law:

$$e \cdot s \;=\; g(e) \cdot \lambda$$

where $g(e) = e \,/\, [-\log_2(1 - e/2)]$ is a universal function of e alone, bounded between 1 and $2 \ln 2 \approx 1.386$.

That bound is the surprise. The function g varies by less than 39% across the *entire range* of possible redundancy, from nearly none ($e \to 0$) to total ($e = 1$). The product $e \cdot s$ is approximately λ, regardless of how the system is built, how large it is, or what alphabet it uses. Verification costs λ. Period.

1.2 The law in plain terms

Five symbols carry the whole book. Each reappears in every chapter, and the glossary at the back defines them again.

Symbol	Meaning
e	*Erasure tolerance*: the fraction of the system that can be lost and still rebuilt.
λ	*Security parameter*: how many bits of certainty you want. $\lambda = 20$ means a one-in-a-million chance of missing a corruption.
s	*Verification cost*: how many random spot-checks you need.
H_v	*Verification entropy*: how many bits of evidence one spot-check gives, $H_v = -\log_2(1 - e/2)$.
$g(e)$	A correction factor, $g(e) = e/H_v$, that always lies between 1 and 1.386.

Read the law as a trade. Redundancy makes each spot-check more likely to land on damage, so the more a system can lose, the fewer checks it needs. A ledger that can lose half its entries ($e = 0.5$) needs about 48 random spot-checks to be sure, to one chance in a million, that nothing has been corrupted. A ledger that can lose only a tenth ($e = 0.1$) needs about 270, because each check is much less likely to find the damage. The two ledgers need very different numbers of checks. But multiply each by its e and the products are 24 and 27, both close to $\lambda = 20$. The checks get cheaper almost exactly as fast as the redundancy grows, and that is all the law says.

1.3 One structure, two powers

The phrase "double entry" will carry a second meaning through this book. Pacioli's double entry is the literal one: two lines per transaction. Ours is the figurative one: two powers—recovery and verification—from one structural constraint.

We will see this double entry in:

- The auditor's sampling tables, where the formula for "how many entries to check" turns out to be *algebraically identical* to the coding theorist's formula for "how many bits to verify" (Chapter 4).
- The erasure codes that protect data in distributed storage, where the same algebraic structure that enables reconstruction from missing pieces also enables succinct proofs that the data were encoded correctly (Chapter 5).
- The data availability problem in blockchains, where suppression of data is detected by the same sampling that would recover it (Chapter 6).
- The Byzantine generals problem, where the quorum structure that tolerates traitors also detects them (Chapter 7).
- DNA replication, where three billion years of evolution produced three layers of error correction—each using the same structural constraint for both detection and repair (Chapter 8).
- Quantum error correction, where the no-cloning theorem forbids backup copies and structural constraints are the *only* option physics allows (Chapter 9).
- Neural networks, where training imposes structural constraints on the weights—making corruption detectable

in trained models but invisible in untrained ones (Chapter 10).

- Human institutions, where separated powers, independent replication, and dual sourcing each add a cross-check that no single component can provide for itself (Chapter 11).

And we will see what happens when the double entry is absent. Systems that assume independence—the standard assumption in risk modeling, financial regulation, and much of engineering—have no structural constraints binding their components together. The law says their verification entropy is exactly zero. They cannot detect their own failures by sampling. They are blind.

This is the mechanism behind what the first book in this series, *The Rogue Wave*, documented: compound failures that models built on independence underprice, and that no routine monitoring catches. Chapter 16 makes the link exact.

What to remember. Pacioli's double-entry bookkeeping (Venice, 1494) gives recovery and verification from a single constraint. This is an instance of a universal law: any structural redundancy that enables recovery automatically enables verification, at a cost of approximately λ bits regardless of system design. The constant $2 \ln 2 \approx 1.386$ bounds the variation.

Chapter 2

Shannon's Other Theorem

The question: Everyone knows Shannon proved how much redundancy communication needs. What did the redundancy give back?

Claude Shannon's 1948 paper, "A Mathematical Theory of Communication" (Shannon, 1948), proved that reliable communication over a noisy channel requires redundancy. If the channel corrupts a fraction of what you send, you must add extra symbols—parity checks, repetitions, structured constraints—so that the receiver can reconstruct the original message despite the noise.

This is Shannon's channel coding theorem, and it answers the question *how much* redundancy is needed. The answer is elegant: you can communicate reliably at any rate below the channel capacity C, and at no rate above it. A code of rate R spends a fraction $1 - R$ of its symbols on redundancy, so reliable communication needs a redundancy fraction of at least $1 - C$ (with C measured per channel symbol).

But Shannon's theorem is silent on a second question: *what else does the redundancy do?*

Shannon proved that good codes exist but did not say how to build them. The coding theorists who followed him built the algebraic structures: Hamming (1950) and Reed and Solomon (1960) constructed explicit codes, and Singleton (1964) bounded what any code can achieve. (Codes that approach capacity came decades later, with turbo, LDPC, and polar codes.) These codes have a minimum distance d: the number of symbols that must differ between any two valid codewords. The distance determines two things simultaneously:

1. **Erasure tolerance.** The code can recover from $d - 1$ lost symbols.
2. **Error detection.** The code can detect $d - 1$ corrupted symbols (and correct $\lfloor (d-1)/2 \rfloor$).

The same parameter—d, the distance—governs both. This is not an engineering coincidence. It is a consequence of the algebraic structure. The constraints that make the code recoverable are the same constraints that make it verifiable. Singleton's 1964 bound, $d \leq n - k + 1$, sets the limit on both powers simultaneously, from one inequality.

2.1 The cost of verification

If a code has distance d and block length n, its erasure tolerance as a fraction is $e = (d-1)/n$. How many random symbol checks do you need to verify that a received word is valid?

Think of each check as a draw from a bag in which a fraction of the marbles are bad. One draw rarely finds a bad marble, but many draws almost surely do. The only question is how many draws make a miss as unlikely as you

require.

Consider the worst case: a corruption at the *unique decoding radius*, where exactly $\lfloor (d-1)/2 \rfloor \approx en/2$ symbols are wrong. Each random check lands on a corrupted symbol with probability $p = e/2$. The probability that s independent checks all miss the corruption is $(1 - e/2)^s$.

For security $2^{-\lambda}$—the probability of missing the corruption is at most one in 2^{λ}—we need:

$$(1 - e/2)^s \;\leq\; 2^{-\lambda}$$

Taking logarithms:

$$s \;=\; \frac{\lambda}{-\log_2(1 - e/2)} \;=\; \frac{\lambda}{H_v}$$

where $H_v = -\log_2(1 - e/2)$ is the *verification entropy* per sample: the number of bits of evidence that one random check provides about whether the word is valid.

This formula is the geometric distribution applied to a detection problem. The mathematics is elementary—any undergraduate probability course covers it. What is not elementary is the observation that H_v is determined entirely by e, with no other parameter.

2.2 The near-constant product

The product of erasure tolerance and verification cost is:

$$e \cdot s \;=\; \frac{e}{-\log_2(1 - e/2)} \cdot \lambda \;=\; g(e) \cdot \lambda$$

The function $g(e)$ depends on e alone. Computing it across the full range:

e (erasure tolerance)	$g(e) = e \cdot s/\lambda$
0.01	1.383
0.05	1.369
0.10	1.351
0.25	1.298
0.50	1.205
0.75	1.106
0.90	1.044
0.99	1.004

The range is $[1,\ 2\ln 2]$—a variation of 38.6%. The limiting value as $e \to 0$ is $2\ln 2 \approx 1.3863$, which arises from the Taylor expansion $-\log_2(1 - x) \approx x/\ln 2$ for small x.

In words: *verification costs λ bits, plus or minus about 39%, regardless of the system's redundancy level.* A system

that tolerates 1% loss needs more than a hundred times as many checks as one that tolerates 99% loss, but once each is scaled by its redundancy, the two pay nearly the same price: the product $e \cdot s$ is about λ in both cases. The redundancy structure is transparent to the cost of verification.

2.3 The constant $2 \ln 2$

The number $2 \ln 2 \approx 1.386$ has two factors, each with a distinct origin. The first $\ln 2$ is the base-conversion constant between bits and natural logarithms ($\log_2 x = \ln x / \ln 2$)—the same conversion that turns Shannon's entropy in nats into entropy in bits. The second factor of 2 comes from the unique decoding radius: the detection probability is $e/2$, not e, because finding a hidden error costs as much as correcting it (Chapter 13). The constant $2 \ln 2$ is the product of these two distinct facts: base conversion $\times$ the erasure-to-error gap.

What to remember. Shannon's channel coding theorem says how much redundancy you need. The duality law says what the redundancy gives back: both recovery and verification, from the same constraints, at a cost of $g(e) \cdot \lambda$ where g varies by only 39% across all possible redundancy levels. The constant $2 \ln 2$ combines base conversion ($\ln 2$) and the erasure-to-error gap ($\times 2$).

Chapter 3

The Law

The question: What is the precise statement, what does it assume, and what is it not?

THIS CHAPTER states the law formally, proves it, and draws the boundary around what it does and does not claim.

3.1 Statement

In words: take a code that can rebuild itself after losing a fraction e of its symbols. The best such codes are *maximum distance separable* (MDS), meaning they waste none of their redundancy, so e equals the redundancy fraction; Reed–Solomon codes are the standard example. An adversary who corrupts as many symbols as the code can still repair—right at the edge where one more would let a decoder be fooled—is caught by each random spot-check with probability $e/2$. Keep checking until the chance that every check missed falls below $2^{-\lambda}$. That takes λ/H_v checks, and the rest of the chapter makes this precise.

The Redundancy Duality *Let $\mathcal{C}$ be an $[n, k, d]$ code over an alphabet of size $q \geq 2$, with erasure tolerance $e = (d-1)/n$. Let a verifier sample s positions uniformly at random and test each against a committed codeword. Then the number of samples required for security $2^{-\lambda}$ against a corruption at the unique decoding radius satisfies*

$$s \;=\; \frac{\lambda}{H_v}, \qquad H_v = -\log_2\!\left(1 - \tfrac{e}{2}\right),$$

and the product

$$e \cdot s \;=\; g(e) \cdot \lambda, \qquad g(e) = \frac{e}{-\log_2(1 - e/2)},$$

where $g(e) \in [1,\ 2\ln 2]$ for all $e \in (0, 1]$. The lower bound is reached only in the limit $e = 1$ (a real code with $k \geq 1$ has $e < 1$, so $g > 1$ strictly); the upper limit $2\ln 2$ is approached as $e \to 0$.

A note on the suppression variant. The law above models *corruption*: the adversary secretly replaces symbols, and the detection probability per sample is $p = e/2$. A second regime—*suppression*—arises when the adversary

withholds data rather than corrupting it, and the absence is directly observable. In the suppression model, $p = e$ (not $e/2$), giving $H_v = -\log_2(1-e)$ and an overall cost of $e \cdot s = g_{\text{DA}}(e) \cdot \lambda$ with $g_{\text{DA}}(e) = e/[-\log_2(1-e)]$, which approaches $\ln 2$ as $e \to 0$—exactly half the corruption cost. The two regimes are unified in Chapter 13; they differ by the universal factor of two between finding a hidden error and observing a known absence.

3.2 Proof

Let $c \in \mathcal{C}$ be the committed codeword. Let w be the received word with $\|w - c\|_H = t$, where $t = \lfloor (d-1)/2 \rfloor$ is the unique decoding radius. The verifier samples position i uniformly from $\{1, \ldots, n\}$ and checks whether $w_i = c_i$.

Detection probability. Exactly t of the n positions satisfy $w_i \neq c_i$. The probability of hitting one is $p = t/n$. For even $d-1$: $t = (d-1)/2$, so $p = (d-1)/(2n) = e/2$ exactly. For odd $d-1$: $t = (d-2)/2$, so $p = (d-2)/(2n) = e/2 - 1/(2n)$, which equals $e/2$ to within $O(1/n)$.

Miss probability. After s independent samples, the probability that all miss the corruption is:

$$P(\text{miss}) = (1-p)^s = \left(1 - \frac{e}{2}\right)^s.$$

Setting $P(\text{miss}) \leq 2^{-\lambda}$ and solving:

$$s \cdot \log_2\left(1 - \frac{e}{2}\right) \leq -\lambda,$$
$$s \geq \frac{\lambda}{-\log_2(1-e/2)} = \frac{\lambda}{H_v}. \qquad \square$$

3.3 The bounds on g

Upper bound. Taylor-expand $-\log_2(1-x) = x/\ln 2 + x^2/(2\ln 2) + \cdots$ for $x = e/2$:

$$g(e) = \frac{e}{-\log_2(1-e/2)} = \frac{e}{e/(2\ln 2) + O(e^2)} \xrightarrow{e \to 0} 2\ln 2.$$

Since $-\log_2(1-e/2)$ grows faster than linearly in e, the function $g(e)$ is strictly decreasing.

Lower bound. At $e = 1$: $g(1) = 1/[-\log_2(1/2)] = 1/1 = 1$.

Monotonicity. The derivative $g'(e) < 0$ for $e \in (0,1)$, which can be verified by computing $g'(e) = [H_v - e \cdot H_v']/(H_v)^2$ and noting that $H_v' = 1/(2(1-e/2)\ln 2)$ exceeds H_v/e for every $e \in (0,1)$, because H_v is convex with $H_v(0) = 0$.

Therefore $g : (0,1] \to [1, 2\ln 2)$ is a bijection, and $e \cdot s/\lambda$ is bounded between these constants for all MDS codes.

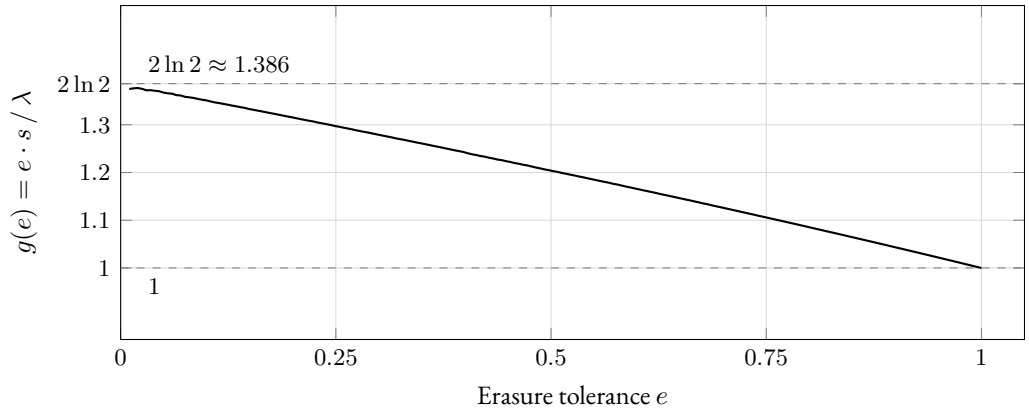


Figure 3.1: The function $g(e)$ across all redundancy levels. The product $e \cdot s/\lambda$ varies by less than 39%—verification costs approximately λ regardless of system design.

3.4 What the law assumes

1. **Identity testing.** The verifier knows the claimed codeword c and checks samples against it. This is the model for systems with commitments (Merkle roots, hash digests, digital signatures). It is *not* the model for membership testing, where the verifier knows only the code structure, not the specific codeword. Chapter 15 treats the membership case.
2. **Corruption at the unique decoding radius.** The attacker corrupts exactly $\lfloor (d-1)/2 \rfloor$ positions. This is the benchmark corruption: the largest the code is guaranteed to repair, one symbol short of the $\lceil d/2 \rceil$ needed to push a decoder toward a wrong codeword. Heavier corruption is easier to catch by sampling; lighter corruption is harder to catch by sampling, but the decoder repairs it anyway.
3. **MDS codes.** The code achieves the Singleton bound $d = n - k + 1$. For non-MDS codes (binary codes, LDPC, turbo codes), $d < n - k + 1$, so the same redundancy buys a smaller e. The formula still holds with $e = (d-1)/n$; what a non-MDS code loses is e itself, and with it detection probability: $p = e/2 < (n-k)/(2n)$. For a given amount of redundancy, MDS codes get the most verification power.
4. **Position sampling.** The verifier checks one symbol per sample. This is distinct from row/column sampling in tensor codes, where each sample reads an entire row. Chapter 14 treats the tensor case.
5. **Independent sampling.** Samples are drawn independently (or without replacement from a set of size n). The hypergeometric correction for sampling without replacement changes the formula slightly but does not affect the asymptotic bounds.

3.5 What the law is not

The formula $s = \lambda / H_v$ is the geometric distribution applied to a detection problem. It is not mathematically novel. Any probability textbook derives it.

The law's content is not the formula. It is three claims:

1. **The duality.** Recovery tolerance and detection probability are not independent parameters. They are determined by the same code parameter d, and the map $e \mapsto H_v$ has no free parameters.
2. **The near-constant product.** The variation of $g(e)$ is only 38.6% across all redundancy levels. This is a quantitative surprise: verification costs approximately λ regardless of how the system is designed.
3. **The universality.** The same formula, derived from the same structure, appears independently in coding theory, audit sampling, data availability, and Byzantine fault tolerance. Audit sampling and coding theory grew up separately and do not cite each other; data availability borrowed its codes from coding theory but arrived at its sampling rule on its own. They arrived at the same equation because the underlying structure—random sampling of constrained data—is the same.

The next eight chapters test these claims against sourced evidence.

3.6 The general form

The law above is stated for $[n, k, d]$ codes, but the duality is not specific to algebraic codes. It holds for *any* system of n components whose valid configurations have minimum Hamming distance $d \geq 2$.

The Structural Duality (General Form) *Let Σ be a finite alphabet with $|\Sigma| = q \geq 2$, and let $\mathcal{C} \subseteq \Sigma^n$ be a set of valid configurations that can recover from $d-1$ erasures at known positions, with erasure tolerance $e = (d-1)/n > 0$. Then:*

1. ***Recovery $\Rightarrow$ detection.*** *$\mathcal{C}$ has minimum Hamming distance $\geq d$, and therefore detects any corruption of $\lfloor (d-1)/2 \rfloor$ or fewer symbols.*
2. ***Sampling cost.*** *Identity testing against a known valid configuration requires $s = \lambda / H_v$ samples for $2^{-\lambda}$ security, with $H_v = -\log_2(1 - e/2)$.*
3. ***Near-constant product.*** *$e \cdot s = g(e) \cdot \lambda$ with $g(e) \in [1,\ 2 \ln 2]$.*

No algebraic structure is assumed. The configurations need not form a linear code; the alphabet need not be a field. The only requirement is structural: the system must have enough constraints to recover from erasures.

Scope of the quantitative bound. The general form proves two things at different levels of generality. The *duality* (recovery $\Leftrightarrow$ detection) is proved unconditionally for any system with minimum Hamming distance $d \geq 2$, over any alphabet. The *near-constant product* ($g(e) \in [1,\ 2 \ln 2]$) requires the additional assumption that the adversary's corruption can be meaningfully modeled at the unique decoding radius—that exactly $\lfloor (d-1)/2 \rfloor$ positions are wrong. For algebraic codes, this is the natural worst case. For general constraint systems (BFT quorums, institutional checks, neural weight manifolds), the "unique decoding radius" is an analogy, not a theorem: the detection probability per sample may differ from $e/2$, and the specific bound on g becomes a prediction to test rather than a consequence of the proof. The evidence chapters test it. The duality itself does not depend on the analogy holding.

The proof is short. Suppose the system recovers from $d-1$ erasures. Take any two valid configurations $c, c' \in \mathcal{C}$ and let $j = |\{i : c_i \neq c'_i\}|$. If $j \leq d-1$, then erasing the positions where c and c' differ leaves $n - j \geq n - (d-1)$ surviving positions where the two configurations agree. Both c and c' match the surviving data, contradicting unique recovery. So $j \geq d$: the minimum distance is at least d.

With minimum distance d, any corruption of $t = \lfloor (d-1)/2 \rfloor$ symbols produces a word at Hamming distance t from the true configuration and at distance $\geq d - t > t$ from every other valid configuration. The corruption is detectable by identity testing. The sampling formula follows from the same geometric argument as the proof above.

This is the structural core of the duality: *unique recovery at distance d implies detection at distance $\lfloor (d-1)/2 \rfloor$, for any system, over any alphabet.* The algebraic codes of the evidence chapters are instances. The generality is the structure.

Every number in this chapter is reproduced by `code/es_product.py`.

What to remember. The law: $e \cdot s = g(e) \cdot \lambda$ with $g \in [1, 2 \ln 2]$, proved from the geometric distribution applied to position sampling at the unique decoding radius. The content is not the formula (which is elementary) but the duality (one parameter governs both), the near-constant product (39% variation), and the cross-domain universality. The general form requires no algebraic structure—only minimum distance—so the duality holds for any constraint system, not only codes.

Interlude

One Equation

Before the auditor opens the ledger, before the coding theorist writes the generator matrix, before the light client queries the network—a question.

What does it mean for two fields to discover the same formula?

The auditor's formula, published in sampling guides for half a century: $n = \ln \alpha / \ln(1 - p)$. The coding theorist's formula, implicit since the Singleton bound of 1964: $s = \lambda / H_v$. They are—as the next chapter will show—algebraically identical. Reached by two communities that have never cited each other. For problems that, on the surface, do not resemble each other at all.

This is not convergent evolution in the biological sense, where similar pressures produce similar forms. It is stronger. The formula is not similar—it is the same equation, derived from the same mathematical structure, with different names for the variables.

When two paths through different territory arrive at the same destination, the destination is real. The formula is not an artifact of how auditors think or how engineers model. It is a property of the structure itself.

The next eight chapters test this claim. Each presents a domain, a formula, and a comparison. If the law is real, the formulas will match. If it is an artifact of one field's conventions, at least one domain will break the pattern.

None does. Three of them—Byzantine consensus, neural networks, and human institutions—test the duality's structure rather than its numbers, and their chapters say so. The rest match number for number.

Part II

The Evidence

The law tested—against auditing standards, erasure codes, data availability sampling, Byzantine consensus, DNA replication, quantum error correction, neural networks, and human institutions—with every number sourced.

Chapter 4

The Auditor's Formula

The question: Does the formula from coding theory match the one auditors actually use?

The International Standard on Auditing 530 (International Auditing and Assurance Standards Board, 2009), published by the International Auditing and Assurance Standards Board, governs how financial auditors choose sample sizes when testing a population of transactions. The standard is principles-based—it does not mandate a specific formula—but it references statistical sampling methods, and the practical implementation follows the AICPA Audit Sampling Guide (American Institute of Certified Public Accountants, 2017) and the HUD Office of Inspector General's attribute sampling tables (U.S. Department of Housing and Urban Development, Office of Inspector General, 2000).

The standard sampling method for audit tests of controls is *attribute sampling*: the auditor draws n items at random from a population, examines each for deviations (errors, irregularities, policy violations), and concludes whether the true deviation rate exceeds a tolerable threshold.

The formula for the zero-defects case—how many items must be sampled such that finding zero deviations gives confidence $1 - \alpha$ that the true rate is below p—is:

$$n_{\text{audit}} = \frac{\ln \alpha}{\ln(1 - p)}$$

where α is the risk of over-reliance (the probability of incorrectly concluding the controls are effective) and p is the tolerable deviation rate.

4.1 The identity

The coding theory formula from Chapter 3 is:

$$s_{\text{code}} = \frac{\lambda \cdot \ln 2}{-\ln(1 - p)}$$

where $\lambda = -\log_2 \alpha$ is the security parameter in bits.

Substituting $\lambda = -\ln\alpha/\ln 2$:

$$\begin{aligned} s_{\text{code}} &= \frac{(-\ln\alpha/\ln 2)\cdot\ln 2}{-\ln(1-p)} \\ &= \frac{-\ln\alpha}{-\ln(1-p)} \\ &= \frac{\ln\alpha}{\ln(1-p)} \\ &= n_{\text{audit}}. \end{aligned}$$

They are the same formula. Algebraically identical. Not "similar" or "asymptotically equivalent"—the same equation, derived independently by two fields that do not cite each other.

4.2 Testing against published tables

The HUD OIG Appendix A (Attribute Sampling) and the AICPA Audit Sampling Guide publish reference tables mapping (α, p) pairs to recommended sample sizes. The following table compares those published values to the formula's output, rounded up to the next integer.

Confidence	α	Tolerable rate p	Formula $\lceil \ln\alpha/\ln(1-p) \rceil$	Published
95%	0.05	5%	59	59
95%	0.05	10%	29	29
90%	0.10	10%	22	22

For zero expected deviations, the formula reproduces the published sample sizes exactly. (Tables for nonzero expected deviation rates use the full binomial tail rather than the zero-defects formula; the script `code/audit_edr_nonzero.py` reproduces the published 93 for 95% confidence, 5% tolerable rate, 1% expected rate.)

The match is not surprising once you see the algebraic identity. It would be surprising if the numbers *didn't* match. But the point is not the arithmetic. The point is that auditors and coding theorists arrived at the same equation from different premises:

- The auditor asks: "How many random entries must I examine to be 95% confident that the error rate is below 5%?"
- The coding theorist asks: "How many random symbols must I check to be $2^{-\lambda}$-confident that the word is a valid codeword?"

These sound like different questions. They are the same question wearing different clothes. Both are: *How many random samples of a constrained system do I need to detect a deviation from the constraint?*

One subtlety deserves a paragraph. The auditor's p is a *population rate*: the fraction of transactions that are deviations. The coding theorist's p is a *per-sample detection probability*: the chance that one random check lands

on a corrupted symbol. These are different quantities, but they enter the formula the same way. In both cases one sample passes with probability $1 - p$, so s samples all pass with probability $(1 - p)^s$.

The formula needs only that each sample independently has probability p of revealing the problem. It is the same formula because the underlying structure, independent trials with a fixed chance of success, is the same.

4.3 What this chapter proves and does not prove

Proves: The audit sampling formula and the coding detection formula are algebraically identical, not merely similar. Two fields with separate lineages—statistical audit sampling, which grew out of industrial acceptance sampling and entered auditing practice in the 1950s, and coding theory, which grew out of Shannon's 1948 paper—arrived at the same equation because both deal with random sampling of constrained data.

Does not prove: That auditors "use coding theory" or that coding theorists "do auditing." Neither field derived the formula from the other. The shared formula is evidence for the universality claim—that the duality arises from the structure, not from any specific domain.

Sourcing: The formula $n = \ln \alpha / \ln(1 - p)$ appears in the HUD OIG attribute sampling guide (Appendix A) and is the standard computational basis for the AICPA tables. ISA 530 itself is principles-based and does not mandate this specific formula, but references statistical sampling methods that reduce to it. The algebraic identity is verified in `code/audit_identity.py`.

> **What to remember.** The audit sampling formula $n = \ln \alpha / \ln(1 - p)$ is algebraically identical to the coding detection formula $s = \lambda \ln 2 / [-\ln(1 - p)]$. Two fields, two lineages, one equation. The match is evidence for the universality of the duality, not a coincidence of notation.

Chapter 5

Codes and Sampling

The question: Does the law hold exactly for real error-correcting codes?

The law was derived for MDS codes at the unique decoding radius. This chapter tests it numerically against Reed–Solomon codes over a finite field, using Monte Carlo simulation (many repeated random trials, counting how often the corruption is caught).

5.1 Setup

We work with Reed–Solomon codes over $\mathrm{GF}(257)$—the integers modulo the prime 257, a field large enough that accidental symbol collisions ($1/q = 1/257$) are rare. The codeword is the evaluation of a degree-$k-1$ polynomial at n distinct points. The minimum distance is $d = n - k + 1$.

For each trial: generate a random polynomial of degree $k-1$, evaluate it at n points to produce a codeword c, corrupt $t = \lfloor (d-1)/2 \rfloor$ random positions by adding a nonzero random value, and draw a random subset of s positions as the sample.

Two tests:

- **Identity testing.** Does any sampled position differ from the committed codeword c?
- **Membership testing.** Do the sampled points lie on *some* polynomial of degree $< k$? (Interpolate through the first k points; check the rest.)

5.2 Results: identity testing

For $\mathrm{RS}(16, 8)$ with $d = 9$ and $t = 4$ corruptions, the identity detection rate across 5,000 trials:

Samples s	Measured detection	Predicted $1 - \binom{12}{s}/\binom{16}{s}$
4	0.727	0.728
8	0.962	0.962
10	0.991	0.992
12	0.999	0.999
16	1.000	1.000

The measured rates match the prediction (the hypergeometric formula, which counts draws without replacement) to within Monte Carlo noise. The identity testing formula works exactly: each sample hits a corrupted position with probability $t/n = e/2$.

5.3 Results: membership testing

The same experiment, but now the verifier does not know c—only that the code is $\mathrm{RS}(16, 8)$. The verifier interpolates through the first k sampled points and checks whether the remaining $s - k$ points match.

Samples s	Identity detection	Membership detection
4	0.727	0.000
8	0.962	0.000
9	0.981	0.978
10	0.991	0.990
12	0.999	0.999

Two things are visible:

1. **For $s \leq k$, membership detection is exactly zero.** Any k points (over any field) are consistent with *some* polynomial of degree $k - 1$. The verifier cannot distinguish a corrupted word from a different valid codeword. This is the *basis cost*—you need at least k points to establish a reference, and one more to begin checking against it.

2. **For $s > k$, membership detection matches identity detection.** Once the polynomial is determined by the first k points, each additional point is a genuine constraint check. Over a large field, a corrupted symbol at a check position has probability $\leq 1/q$ of accidentally satisfying the interpolated polynomial, so detection is nearly certain per corrupted check point.

The same pattern holds for $\mathrm{RS}(32, 16)$: detection is zero for $s \leq 16$ and matches the identity rate for $s = 17, 18, \ldots$

5.4 The total cost of membership verification

Combining the two regimes, the cost of membership testing is:

$$s_{\text{membership}} = k + \frac{\lambda}{H_v}$$

The first k samples establish the reference polynomial (basis cost). The remaining λ/H_v samples check against it (detection cost). For identity testing, the basis cost is zero—the committed codeword provides the reference for free.

This decomposition explains a design choice in tensor codes. ZODA (Zero-Overhead Data Availability, Chapter 6) is a scheme that encodes data as a matrix so that checking rows both protects and proves it. In ZODA each "sample" is an entire row of the encoded matrix—n symbols. Since $n > k$, the basis cost is paid *within* each sample. The between-row sampling is pure detection cost, and the law $s = \lambda/H_v$ applies directly to the number of rows sampled.

5.5 Confirmation for the law

For identity testing at the unique decoding radius:

- $p = e/2$ exactly for $\mathrm{RS}(16, 8)$, $(16, 4)$, and $(32, 16)$, where $d-1$ is even; for $(255, 128)$, where $d-1 = 127$ is odd, $p = 63/255 = e/2 - 1/(2n)$, as the proof in Chapter 3 predicts.
- $e \cdot s/\lambda = g(e)$ matches the theoretical curve to within $O(1/n)$.
- The bound $g(e) \in [1, 2\ln 2]$ holds in every case.

For membership testing:

- Detection is zero for $s \leq k$ and matches identity testing for $s > k$.
- The gap between identity and membership vanishes as the field size grows ($1/q \to 0$).

Every number in this chapter is reproduced by the scripts `membership_threshold.py` and `es_product.py` in `code/`.

What to remember. Identity testing confirms the law exactly: $p = e/2$, $s = \lambda/H_v$. Membership testing adds a basis cost of k samples before detection begins. For $s > k$ over large fields, the two are indistinguishable. Tensor codes hide the basis cost inside each row sample, which is why ZODA's row-sampling matches the identity-testing formula directly.

Chapter 6

Data Availability

The question: Can we verify that data exists *in a distributed network, without downloading it all?*

The data availability problem is distinct from the corruption problem. In Chapter 5, the adversary *corrupts* symbols. Here, the adversary *withholds* them. A block producer publishes some data; light clients want assurance that enough of it is available for anyone to reconstruct, without downloading the entire block themselves.

This chapter tests the law against two real implementations: the plain erasure scheme in `availability.rs`, the availability module of Tegridy (a blockchain protocol), and a tensor ZODA encoder in an EVM verification stack.

6.1 Plain erasure sampling

Tegridy's availability module uses a $[255, 128]$ Reed–Solomon code over $\mathrm{GF}(256)$. Data of $k = 128$ shards is encoded into $n = 255$ shards. Any 128 shards suffice to reconstruct; 127 shards do not.

The adversary's strategy: withhold enough shards that reconstruction fails, while answering random shard requests to fool the verifier. If only $k - 1 = 127$ shards exist, a random request returns a shard with probability $(k-1)/n = 127/255 \approx 0.498$.

The code's `miss_probability` function computes:

$$P(\text{fooled after } s \text{ samples}) = \left(\frac{k-1}{n}\right)^s$$

and `samples_needed` inverts it:

$$s = \left\lceil \frac{\lambda \cdot \ln 2}{-\ln\left(\frac{k-1}{n}\right)} \right\rceil .$$

In the law's framework:

$$\begin{aligned} e &= (n-k)/n = 127/255 \approx 0.498 && \text{(erasure tolerance)} \\ p &= 1 - (k-1)/n = 128/255 \approx 0.502 && \text{(detection probability per sample)} \\ H_v &= -\log_2(127/255) = 1.006 \text{ bits} && \text{(verification entropy per sample)} \end{aligned}$$

For $\lambda = 40$ bits: $s = 40/1.006 = 39.8 \to 40$ samples. The code's `samples_needed(Params{k: 128, n: 255}, 40)` returns exactly 40.

The key observation: $p \approx e$, not $e/2$. In the suppression model, the detection probability per sample equals the erasure tolerance, because the adversary's withholding is directly observable—you asked for a shard and got nothing. In the corruption model, $p = e/2$ because the adversary's changes are hidden and only half the erasure capacity translates to detection probability (the factor of two from the unique decoding radius).

The suppression model gives $e \cdot s/\lambda = g_{\text{DA}}(e)$ where $g_{\text{DA}}(e) = e/[-\log_2(1-e)]$, which approaches $\ln 2 \approx 0.693$ as $e \to 0$ and decreases toward 0 as $e \to 1$. For $e \approx 0.5$: $g_{\text{DA}} \approx 0.5$, so $e \cdot s \approx \lambda/2$.

In the low-redundancy limit this is exactly half the cost of the corruption model ($\ln 2$ against $2 \ln 2$). At higher redundancy suppression is cheaper still: at $e = 0.5$ the two products are 0.50 and 1.205. The factor of two between suppression detection ($p = e$) and corruption detection ($p = e/2$) is the same factor of two that separates erasure correction from error correction in coding theory—the fundamental distinction between knowing what is missing and not knowing what is wrong.

6.2 Tensor ZODA

ZODA (Zero-Overhead Data Availability) encodes data as a matrix $Z = G \cdot X \cdot {G'}^T$, where G and G' are Reed–Solomon generator matrices. The encoding is simultaneously an erasure code, a commitment, and a proof of correct encoding (Evans et al., 2025). (The ZODA paper itself re-randomises the encoding with a random diagonal matrix between X and ${G'}^T$. The implementation tested here is the closely related Ligero-style tensor test that the paper compares itself to (Ames et al., 2017), and its soundness rests on the same building blocks.)

The verification works by sampling random rows and checking that each is a codeword of the column code G'. This is *membership testing* (Chapter 5)—the verifier checks codeword membership, not identity against a commitment. But because each row has $n > k$ symbols, the basis cost is hidden within each row read.

The security analysis involves three increasingly conservative attack models, each giving a different proximity gap (the minimum fraction of rows that must be non-codewords when the matrix is corrupted). In the table, UDR is the unique decoding radius of Chapter 2, and "Lean" marks a bound that has been machine-checked in the Lean proof assistant.

Attack model	Detection fraction	H_v (bits/sample)	Target	Samples
Block at UDR	$1 - (1 - \sqrt{1/8})^2 \approx 0.58$	1.26	128 bits	102 (pairs)
Interleaved proximity	$1/4$	0.415	144 bits	354
Elementary radius (Lean)	$1/6$	0.263	144 bits	556

The sample count in each case derives from $s \approx \lambda/H_v$, with H_v determined by the proximity gap and λ adjusted for model-specific soundness factors (union bounds over failure events). The production code uses 556 samples (the most conservative bound), matching the machine-checked formula: $4 \times (5/6)^{556} \leq 2^{-144}$—the factor of 4 is a union bound that effectively raises the target from 144 to 146 bits.

The non-circular content is the proximity gap itself. The fraction $1/6$ (or $1/4$, or $\sqrt{1/8}$) comes from a theorem of algebraic coding theory, not from the sampling formula. The underlying results are the proximity gaps for Reed–Solomon codes of Ben-Sasson et al. (2020), extended to interleaved codes by Diamond and Gruen (2024), which the soundness analysis applies to row sampling of a tensor code. In that form the statement is: for a matrix that is δ-far from every tensor codeword, at least a $\delta'(\delta)$ fraction of its rows fail the codeword test. The function $\delta'(\delta)$ depends on the code's rate and distance—the same parameters that determine the erasure tolerance e.

The chain is:

$$\text{code rate} \;\rightarrow\; \text{proximity gap } \delta'(\delta) \;\rightarrow\; \text{detection probability } p \;\rightarrow\; s = \lambda / H_v.$$

The first arrow is algebraic coding theory. The last arrow is the geometric distribution. The law claims that the entire chain is determined by e—and for rate-$1/2$ RS codes, it is.

Every number in this chapter is reproduced by `code/tegridy_check.py` and `code/zoda_check.py`.

What to remember. The suppression model (data availability) gives $p = e$, exactly twice the corruption model's $p = e/2$. Tegridy's availability module matches the formula to the sample. ZODA uses tensor row-sampling with proximity gaps from algebraic coding theory; the sample count follows from $s = \lambda / H_v$ with H_v set by the gap.

Chapter 7

Byzantine Generals

The question: Does the duality hold when the adversary is not passive but actively lying?

The Byzantine generals problem, posed by Lamport et al. (1982), asks: how can n parties reach agreement when up to f of them are adversarial—sending different values to different recipients, lying, conspiring? The celebrated answer: when messages cannot be signed, agreement is possible if and only if $n \geq 3f + 1$. (With unforgeable signatures the synchronous problem can be solved for any number of traitors, but the $3f + 1$ bound returns once message delays are unbounded, which is why practical protocols use it.) With $n = 3f + 1$ nodes—an honest two-thirds supermajority—the system tolerates f Byzantine faults.

The structural constraint here is the *quorum intersection*: any two quorums of size $2f + 1$ share at least one honest node (since $2(2f + 1) - (3f + 1) = f + 1 > f$). This overlap is the redundancy—it binds the network together and prevents the adversary from presenting conflicting views to different parties.

7.1 Recovery and verification

Recovery. If f nodes crash or go silent (erasures, not Byzantine faults), the remaining $2f + 1$ nodes form a quorum and can reach agreement among themselves. The system recovers from f out of $3f + 1$ lost nodes: $e = f/(3f + 1) \approx 1/3$.

Verification. A verifier querying random nodes can detect Byzantine behavior by finding contradictions. If f nodes are actively lying, each query has probability $f/n = f/(3f + 1) \approx 1/3$ of hitting a Byzantine node. An honest node returns the correct value; a Byzantine node may return anything.

In the law's framework: $p = f/n \approx 1/3$ (the probability of hitting a faulty node), giving $H_v = -\log_2(1 - 1/3) = -\log_2(2/3) \approx 0.585$ bits per query. For $\lambda = 128$: $s = 128/0.585 \approx 219$ queries.

But this overstates the verifier's power. Hitting a Byzantine node doesn't automatically reveal the lie—the verifier must compare the Byzantine node's answer against other answers to detect the inconsistency. In the simplest model (query one node, compare against a known-good value), $p = f/n$ and the formula applies directly. In a model where the verifier has no known-good reference, the verification requires quorum-based comparison, which changes the analysis.

7.2 The two models

In short: crashes are easy to see, and lies are not. The two cases differ by the same factor of two that separates erasures from errors (Chapter 13).

Suppression model (node crashes, no lies). Recovery from f crashes: $e = f/n \approx 1/3$. Detection: a crashed node doesn't respond. $p = f/n \approx 1/3$. Same as the data availability model—suppression is directly observable. $H_v = 0.585$ bits per query.

Corruption model (Byzantine lies). Here the analogy to coding theory is suggestive but imperfect. In a code, the unique decoding radius is $\lfloor (d-1)/2 \rfloor$—the most errors the decoder can uniquely correct. In BFT, a real adversary uses all f nodes, not $f/2$. But if we ask: "how many Byzantine nodes can the system tolerate while *uniquely* determining the correct value from any quorum?"—the answer is $\lfloor f/2 \rfloor$, because two conflicting quorums could each contain up to $f/2$ liars and still point to different values. This is structurally analogous to the UDR, not identical to it. Under this analogy: $p \approx (f/2)/n \approx e/2 \approx 1/6$, giving $H_v \approx 0.263$ bits per query.

The numerical coincidence with ZODA's elementary-radius figure (0.263 bits per sample) reflects a shared effective redundancy of $\approx 1/3$. But unlike the audit and DA chapters, this match rests on an analogy between BFT and coding theory, not on a formula that both fields independently derived. BFT is evidence that the duality's *structure* extends beyond algebraic codes; it is not a numerical confirmation at the same level as the other evidence chapters.

7.3 What BFT adds

Byzantine fault tolerance extends the duality, by analogy, to a setting without algebraic codes:

- The structural constraint (quorum intersection) serves both recovery (reaching agreement despite crashes) and verification (detecting liars through contradictions).
- The detection probability per sample depends on e (the fault tolerance fraction), consistent with the law.
- The two models (suppression vs. corruption) exhibit the same factor-of-two gap seen in coding theory: observable faults ($p = e$) are twice as easy to detect as hidden faults ($p = e/2$).

BFT is not a code—it has no parity-check matrix, no generator polynomial, no Singleton bound. Yet the same structural duality appears: redundant constraints bind the system together and provide both recovery and verification. The quantitative match relies on an analogy (the "Byzantine UDR"), not on a formula independently derived in the BFT literature. The suppression model ($p = e$ for crash faults) is exact; the corruption model ($p \approx e/2$ for Byzantine faults) is the analogy.

> **What to remember.** BFT's quorum intersection is a structural constraint that provides both recovery (agreement despite crashes) and verification (detection of lies). The suppression model matches exactly: $p = e$ for observable faults. The corruption model ($p \approx e/2$) rests on a structural analogy between the Byzantine "unique determination" threshold and the coding-theoretic UDR—suggestive, not a numerical confirmation at the level of the other evidence chapters.

Chapter 8

The Error Catastrophe

The question: Did nature discover the duality three billion years before Pacioli?

In 1971, the physical chemist Manfred Eigen posed a question (Eigen, 1971) that would trouble molecular biology for a decade: how large can a genome be? A genome that copies itself makes errors. If the error rate per base exceeds a threshold—roughly one divided by the genome length—the genetic information disintegrates across generations. Eigen called this the *error catastrophe*: the point where mutation overwhelms selection, and the genome dissolves into noise.

The human genome is 3×10^9 base pairs long. Eigen's threshold demands a per-base error rate below approximately 3×10^{-10}. The raw fidelity of DNA polymerase—the enzyme that copies DNA—is only about 10^{-5} errors per base. Five orders of magnitude too high.

The solution is three layers of error correction, each operating on the errors left by the layer before it. And each layer works by the same mechanism: a structural constraint that detects and corrects at once.

8.1 Three layers, one principle

In short: the cell copies its DNA, then checks the copy three times, with each check catching most of what the previous one missed.

Layer 1: Polymerase selectivity. DNA polymerase does not simply grab the nearest nucleotide. Its active site is shaped to accept only the Watson–Crick complement of the template base: A pairs with T, G pairs with C. A mismatched base distorts the geometry of the active site, slowing incorporation by a factor of 10^3 to 10^5. The structural constraint is geometric: the enzyme's shape enforces the pairing rule.

This constraint provides both detection and correction simultaneously. The distorted geometry *detects* the mismatch (the enzyme stalls), and the same geometric selectivity *corrects* for it (the wrong base is rejected in favor of the right one). One mechanism, two powers. Error rate after this layer: $\sim 10^{-5}$ per base (Kunkel, 2004).

Layer 2: Proofreading exonuclease. Most replicative polymerases carry a second active site: a $3' \to 5'$ exonuclease that can excise the most recently added base. When a mismatch escapes the polymerase's initial selectivity, the distortion of the DNA double helix at the mismatch site shifts the primer terminus from the polymerase active site to the exonuclease site.

Again: one structural event—the helix distortion—serves as both the detection signal and the trigger for

correction. The exonuclease excises the wrong base, the primer returns to the polymerase site, and synthesis resumes. This layer catches roughly 99% of the errors that escape layer 1. Remaining error rate: $\sim 10^{-7}$ per base.

Layer 3: Mismatch repair (MMR). After replication is complete, the MMR system scans the new DNA for mismatches. The MutS protein recognizes the distortion caused by a non-Watson–Crick pair and binds to it (detection). This binding recruits MutL and associated enzymes that excise a segment of the daughter strand around the mismatch and resynthesize it using the parent strand as template (correction).

Once more: the same structural feature—the geometric distortion of a mismatch—serves as both the alarm and the repair signal. MutS does not have one domain for finding errors and a different domain for fixing them. It has one recognition event that initiates both. This layer catches roughly 99.9% of remaining errors. Final error rate: $\sim 10^{-10}$ per base.

8.2 Composition

The three layers compose multiplicatively, in the same sense as tensor codes (Chapter 14). Each layer operates on the errors left by the previous layer, and the layers are approximately independent—the mechanism of each (geometric selectivity, exonuclease excision, post-replicative scanning) is physically distinct from the others.

The miss probabilities multiply:

$$\underbrace{10^{-5}}_{\text{polymerase}} \times \underbrace{10^{-2}}_{\text{proofreading miss}} \times \underbrace{10^{-3}}_{\text{MMR miss}} = 10^{-10} \text{ per base.}$$

The total fidelity per base is the sum of the three layers' contributions in the log domain, just as a tensor code's security comes from summing its levels' verification entropies:

$$-\log_2(10^{-10}) \approx 33 \text{ bits per base.}$$

This is the biological security parameter: 33 bits of confidence, per base, that the copy is faithful.

Every number in this section is reproduced by `code/dna_composition.py`.

8.3 Eigen's threshold and the security parameter

Eigen's error catastrophe sets the security requirement: the error rate per copy must be below $\sim 1/L$ for a genome of length L. This is the biological security parameter. For the human genome ($L = 3 \times 10^9$):

$$\mu_{\max} \approx \frac{1}{L} \approx 3.3 \times 10^{-10} \text{ per base.}$$

The achieved rate of 10^{-10} per base per cell division clears this threshold with a small margin. (Two refinements matter here. Eigen's threshold is more precisely $\ln \sigma / L$, where σ is the selective advantage of the correct sequence,

so "$1/L$" is the order of magnitude. And a human germline passes through dozens to hundreds of cell divisions per generation, so the measured rate per generation is about 1.2×10^{-8} per base (Kong et al., 2012). The genome tolerates this because only a small fraction of its length is under selective constraint, which shrinks the effective L; even so, each generation brings a few new harmful mutations, a load that selection must keep clearing.)

Now consider how many layers are needed as genomes grow:

Organism class	Genome size	Required μ	Layers required
RNA viruses	$\sim 10^4$	$< 10^{-4}$	0–1
DNA viruses, bacteria	$\sim 10^6$	$< 10^{-6}$	1–2
Simple eukaryotes	$\sim 10^7$–10^8	$< 10^{-8}$	2–3
Complex eukaryotes	$\sim 10^9$–10^{10}	$< 10^{-10}$	3

This is the biological analog of ZODA's design choice (Chapter 14): use enough levels of composition to meet the security requirement. RNA viruses, with small genomes, get by with the polymerase's raw fidelity—or even benefit from high mutation rates that enable rapid adaptation. The exceptions prove the rule: coronaviruses and their nidovirus relatives, whose $\sim 3 \times 10^4$-base genomes are among the largest of any RNA virus, are also the ones that carry a proofreading exonuclease. The table gives the *required* depth, not the depth organisms actually carry. Bacteria in fact run all three layers (mismatch repair was first worked out in *E. coli*) and reach a per-base error rate near 10^{-10}, well beyond what a 10^6-base genome strictly needs. Evolution does not stop at the minimum. It reliably adds layers once genomes grow past what the previous depth can protect.

8.4 What DNA adds

DNA error correction was not designed by anyone who had read Shannon, Singleton, or Ben-Sasson. It was shaped by three billion years of natural selection—a blind optimization process that converges on structures that work and discards those that don't.

That it arrived at the same architecture—layered constraints where each layer's detection mechanism *is* its correction mechanism, composing multiplicatively to meet a threshold—is the strongest evidence for universality. The duality is not an artifact of human engineering. It is a property of structural constraints themselves.

The double entry was written into the geometry of the double helix long before it was written into the ledgers of Venice.

8.5 On the frontier

Where does DNA sit on the verification frontier of Chapter 18?

The Watson–Crick base-pair constraint is a rate-$1/2$ encoding: one strand determines the other, over a four-letter alphabet ($\{$A, C, G, T$\}$). The erasure tolerance is $e = 1/2$—lose one strand entirely, reconstruct from the

complement. This places the fundamental mechanism at exactly the same frontier position as the Reed–Solomon codes of the engineering chapters:

System	Rate	e	H_v (bits)	$g(e)$
RS$(255, 128)$	$1/2$	0.498	0.413	1.205
RS$(16, 8)$	$1/2$	0.500	0.415	1.205
Watson–Crick base pair	$1/2$	0.500	0.415	1.205

The match is not a coincidence—it is a consequence of rate. Any rate-$1/2$ MDS structure sits at $e = 0.5$, $g = 1.205$, regardless of whether it was designed by an engineer in 2020 or by selection over three billion years.

The three repair layers do not change DNA's position on the frontier. They change how much security the system extracts from that position. Each layer reduces the per-base miss probability multiplicatively, adding to the effective security parameter:

$$\lambda_{\text{eff}} = \underbrace{16.6}_{\text{polymerase}} + \underbrace{6.6}_{\text{proofreading}} + \underbrace{10.0}_{\text{MMR}} = 33.2 \text{ bits per base.}$$

The three layers are biological composition—sequential, not tensor, but achieving the same end: stacking independent detection mechanisms to multiply the effective λ while holding e fixed.

Evolution found the same balanced operating point ($e = 1/2$, $g \approx 1.2$) that most engineered systems also choose. All numbers are computed by `code/dna_frontier.py`.

What to remember. DNA uses three layers of error correction—polymerase selectivity, proofreading, and mismatch repair—each exhibiting the duality: the structural feature that detects a mismatch is the same feature that triggers its correction. The layers compose multiplicatively ($10^{-5} \times 10^{-2} \times 10^{-3} = 10^{-10}$), meeting Eigen's error threshold for large genomes. The Watson–Crick base pair sits on the same frontier as RS$(255, 128)$: $e = 0.5$, $g = 1.205$—the balanced operating point, found independently by evolution and engineering. Evolution arrived at the double entry through three billion years of selection on structural constraints.

Chapter 9

The No-Cloning Duality

The question: What happens to the duality when copying is forbidden by physics?

In 1982, William Wootters and Wojciech Zurek proved a theorem (Wootters and Zurek, 1982) that sounds like a minor technicality but has profound consequences: an unknown quantum state cannot be copied. The no-cloning theorem is not an engineering limitation—it is a consequence of the linearity of quantum mechanics. No device, no matter how sophisticated, can take an arbitrary quantum state $|\psi\rangle$ and produce two copies $|\psi\rangle \otimes |\psi\rangle$.

This means the most basic error-protection strategy—make a backup—is physically impossible for quantum information. You cannot store a copy of a quantum state and compare it to the original later. You cannot replicate a qubit the way you replicate a ledger entry or a data shard.

The only protection available is the double entry: structural constraints within the encoding that provide both recovery and verification from a single mechanism.

9.1 Stabilizer codes

In plain terms: spread one qubit's information across many physical qubits, and keep a set of consistency checks that every valid arrangement must pass. The checks play the role of the parity bits in a classical code.

Quantum error correction, developed by Shor (1995), Steane (1996), and Gottesman (1997), encodes k logical qubits into n physical qubits using a *stabilizer code*. The code is defined by a set of $n - k$ commuting operators called stabilizer generators, each acting on multiple physical qubits. The valid code states are exactly those stabilized by every generator: $S_i|\psi_L\rangle = |\psi_L\rangle$ for all generators S_i.

These generators are the structural constraints—the quantum analog of parity checks. Like parity checks, they bind the qubits together, reducing the 2^n-dimensional Hilbert space to a 2^k-dimensional code subspace.

A $[\![n, k, d]\!]$ stabilizer code has minimum distance d, and the same dichotomy as classical codes:

- **Erasure correction:** $d - 1$ qubit losses (at known positions) can be corrected.
- **Error correction:** $\lfloor (d-1)/2 \rfloor$ errors (at unknown positions) can be corrected.

The factor of two is identical to the classical case (Chapter 13), and for the same reason: finding a hidden error costs as much as fixing it.

9.2 The duality

The stabilizer generators serve both purposes through a single operation: *syndrome measurement*, which means running every consistency check and reading off which ones fail.

Verification. Measuring each stabilizer generator returns $+1$ if the state satisfies that constraint and -1 if it does not. The pattern of ± 1 outcomes—the syndrome—reveals whether an error has occurred and, if so, what type and where. This is detection: testing the data against the structural constraints.

Recovery. The same syndrome determines the correction. Each syndrome pattern maps to a specific recovery operation that restores the logical state. The generators that detected the error are the same generators that specify the fix.

Crucially, syndrome measurement does not disturb the encoded information. Because the generators commute with the logical operators (by construction), measuring them collapses no logical-qubit superposition. The verifier learns whether the constraints are satisfied without learning what the encoded state is.

This is verification in its purest form: checking the structure without accessing the content. The classical analog would be checking that a codeword satisfies its parity equations without reading the message symbols—but in quantum mechanics, this separation is not merely convenient. It is *necessary*, because reading the logical state would destroy it.

9.3 Why no-cloning forces the double entry

Classical systems have a choice. You can protect data by structural constraints (the double entry), or you can protect it by copying (backup tapes, replicated databases, printed receipts). The two strategies are different in practice, though the law shows that structural constraints are strictly more efficient.

Quantum systems have no choice. No-cloning eliminates the copying strategy entirely. The *only* way to protect quantum information is structural redundancy—stabilizer generators, check operators, entanglement across physical qubits—that provides recovery and verification from one mechanism.

The double entry is not one option among several for quantum information. It is the only option physics allows.

9.4 The quantum Singleton bound

The quantum Singleton bound (Gottesman, 1997; Knill and Laflamme, 1997) constrains the parameters of a stabilizer code: $n - k \geq 2(d-1)$, or equivalently $d \leq (n-k)/2 + 1$. Compared to the classical Singleton bound $d \leq n - k + 1$, the quantum version allows roughly half the distance for the same redundancy, because quantum codes must simultaneously protect against both bit-flip (X) and phase-flip (Z) errors.

But the duality's structure is unchanged. The erasure tolerance $e = (d-1)/n$, the factor of two, and the relationship between code distance and detection power all carry over from the classical case. One set of generators,

two operations. The mechanism that makes position sampling work for classical codes—structural constraints that must be violated by corruption—is the same mechanism that makes syndrome measurement work for quantum codes.

9.5 Numerical confirmation

Every other evidence chapter in this book matches a system's parameters to the law and verifies the numbers. The quantum case deserves the same treatment. The function $g(e)$ depends only on e, so it applies to any system with erasure tolerance e—classical or quantum. The quantum Singleton bound shifts *where* codes sit on the e axis (lower e for the same redundancy), but $g(e)$ itself does not change.

Computed by `code/quantum_check.py` for well-known stabilizer codes:

Code	n	k	d	e	H_v (bits)	$g(e)$
$[\![5, 1, 3]\!]$ Perfect	5	1	3	0.400	0.322	1.243
$[\![7, 1, 3]\!]$ Steane	7	1	3	0.286	0.222	1.285
$[\![9, 1, 3]\!]$ Shor	9	1	3	0.222	0.170	1.308
$[\![23, 1, 7]\!]$ Golay	23	1	7	0.261	0.202	1.294
$[\![17, 1, 5]\!]$ Color	17	1	5	0.235	0.181	1.303

Every $g(e)$ falls in $[1,\ 2\ln 2]$. The pattern is monotone: codes with lower e (less erasure tolerance per physical qubit) approach $2\ln 2$, and codes with higher e approach 1—the same curve as the classical case.

The quantum Singleton bound forces $n - k \geq 2(d-1)$, so the $[\![5, 1, 3]\!]$ code (which achieves this bound with equality) is the quantum analog of a classical MDS code. The others use more redundancy than the bound requires, placing them at lower e on the same frontier. In every case, $g(e) \in [1, 1.386]$.

Quantum mechanics does not create the double entry. It reveals that the double entry was always the fundamental mechanism—the one that works even when copying is forbidden by the laws of physics.

What to remember. The no-cloning theorem forbids backup copies of quantum states. The only protection is structural constraints: stabilizer codes whose generators serve as both the detection mechanism (syndrome measurement) and the recovery mechanism (correction operations). The factor of two holds: $d-1$ erasures, $\lfloor (d-1)/2 \rfloor$ errors. Quantum mechanics does not invent the duality—it eliminates every alternative.

Chapter 10

The Weights That Learn

The question: Does training a neural network create structural constraints?

A neural network begins as a collection of random numbers. Thousands, millions, or billions of weights, drawn from a distribution, connected by an architecture but constrained by nothing. The weights are independent. If you corrupt one, the others provide no information about the corruption, because no structural relationship binds them together. $H_v = 0$.

Then the network is trained.

Training is an optimization: the weights are adjusted, iteratively, to minimize a loss function over structured data. After training, the weights are no longer independent. They encode the patterns of the data—the correlations, the symmetries, the input-output relationships that the loss function rewards. The optimization has imposed structural constraints on the weights, just as a code's generator matrix imposes constraints on the codeword symbols.

The duality predicts that these constraints should give the trained weights two powers: recovery (the ability to reconstruct corrupted weights from the data, by retraining) and verification (the ability to detect corruption by its effect on the learned function). The untrained weights should have neither.

This chapter tests the prediction.

10.1 The experiment

The experiment uses a small multi-layer perceptron ($6 \to 32 \to 1$, ReLU activations, 257 weights) and a structured target function: $y = x_1^2 + x_1 x_2 - 0.5x_3 + \sin(x_4)$, with small noise. Forty copies of the network are trained to convergence from different random starts; their training losses fall between 0.004 and 0.010. Forty more copies are left at their random initial weights, with training losses between 2.5 and 10.0.

The verifier is given no stored copy of the weights. This is membership testing in the sense of Chapter 5: the verifier has the training data and knows what a valid model looks like, and must decide whether the weights in front of it qualify. A comparison against a stored copy would catch any change to any weights, trained or not, and would say nothing about structure.

For trained models, the constraint is the one training imposes: the weights sit at a minimum of the training loss, so the gradient of the loss there is essentially zero. The verifier computes that gradient and checks it. For random models there is no comparable constraint, but to treat both classes alike the verifier applies the same check to them.

The corruption protocol: pick one weight at random and add Gaussian noise of standard deviation σ. Each class is split in half. Twenty calibration models set the detection threshold: the largest gradient norm among them. The other twenty are held out, and all testing happens on them. A corruption is "detected" if the corrupted model's gradient norm exceeds the threshold, and the false-alarm rate is measured by applying the same threshold to the held-out models before corruption. Each class gets 800 corruptions per noise level (40 on each held-out model).

The question: does training create enough structure that single-weight corruption is detectable without a reference copy?

10.2 Results

	Trained weights		Random weights	
Noise σ	Detection rate	H_v (bits)	Detection rate	H_v (bits)
No corruption (false alarms)	0.000	—	0.050	—
0.01	0.844	2.68	0.050	0
0.05	0.955	4.47	0.049	0
0.10	0.979	5.56	0.050	0
0.50	0.991	6.84	0.036	0
1.00	0.996	8.06	0.039	0

Even a corruption of $\sigma = 0.01$, far smaller than the weights themselves (initialized at a scale of 0.3), is caught 84% of the time in a trained model, which never raised a false alarm; from $\sigma = 0.1$ upward almost every corruption is caught. In random models the detection rate is 4–5% at every noise level, the same rate at which *uncorrupted* random models trip the threshold. A check that fires as often on clean models as on corrupted ones carries no information: for random weights the effective H_v is zero.

The gap is qualitative, not merely quantitative. A corrupted random network is just another random network; nothing distinguishes it from the uncorrupted ones. A corrupted trained network has left the minimum that training put it on, and the constraint that defines the minimum—zero gradient—says so. Training creates structure; that structure makes corruption detectable without any copy to compare against.

The same holds, more weakly, for a cruder check. Testing whether the training loss itself has risen above the uncorrupted range catches 6% of trained-model corruptions at $\sigma = 0.05$ and 85% at $\sigma = 1.0$, and again only the false-alarm rate for random models. The gradient is the sharper check because it tests the constraint directly: at a minimum the loss is flat, so a small push barely changes the loss but almost always tilts the slope. (The misses at the smallest noise level are pushes too small to lift the gradient above the natural spread among uncorrupted trained models, whose gradients are small but not exactly zero.)

Every number is reproduced by `code/neural_net_hv_membership.py`.

10.3 Why trained weights are constrained

A trained network's weights satisfy implicit constraints imposed by the data. The loss function is not a single parity check, but it serves the same structural role: it binds the weights together by requiring that specific combinations of weights produce specific outputs for specific inputs. Changing one weight disrupts these relationships, and the disruption propagates through the network to change the output.

The constraint is not algebraic (there is no generator polynomial). It is functional: the mapping $f(x; w)$ from inputs to outputs, as parameterized by the weights w, must remain close to the data distribution. This is a constraint on the weight space—a manifold of "valid" configurations, analogous to a code's set of valid codewords. Corruption moves the weights off this manifold, and the gradient of the loss—zero on the manifold of minima—detects the departure.

For random weights, there is no manifold. Every weight configuration is equally invalid (or equally valid). Corruption does not move the weights from a constrained region to an unconstrained one, because there was no constrained region to begin with.

10.4 The lottery ticket connection

The lottery ticket hypothesis (Frankle and Carlin, 2019) showed that large neural networks contain small subnetworks ("winning tickets") that, trained in isolation from their original starting values, match the full network's accuracy. In the networks they studied, the winning tickets were 10–20% of the original size or smaller: the complementary 80–90% of the weights could be removed (pruned) without degrading performance.

This is the duality at work. The winning ticket's weights are the structurally constrained ones: they carry the learned function and their corruption is detectable. The prunable weights are the unconstrained ones: they carry little learned structure, and the network can be trained to full accuracy without them, so the learned function does not depend on them. Pruning is the act of identifying which weights carry the constraints and discarding the rest.

The fraction of prunable weights—the "redundancy" that carries no verification power—is not a fixed property of the architecture. It depends on the data and the task. The constraint structure reflects the data's structure, just as a code's distance reflects the encoding scheme.

10.5 What neural networks add

No one designed neural networks by computing $g(e)$ or choosing an erasure tolerance. The structure emerges from optimization, not from algebraic construction. Yet the duality appears: training creates constraints, and those constraints provide both recovery (retrain from data to restore a corrupted model) and verification (check the loss gradient against the data to detect corruption, with no stored copy needed).

This is the same pattern as DNA. Evolution did not design the double helix by reading Shannon. It optimized for replication fidelity under selection pressure, and the structural constraints that emerged exhibit the duality.

Neural network training optimizes for prediction accuracy under a loss function, and the structural constraints that emerge exhibit the same duality.

The mechanism is universal: any optimization that imposes structure on a system's components creates constraints that bind those components together. The constraints give recovery and verification. The absence of optimization—random initialization—gives neither.

10.6 What the chapter does not show

The other evidence chapters in this book test the quantitative law: they compute e, predict H_v, and verify that $g(e) \in [1,\ 2\ln 2]$. This chapter tests only the *direction* of the duality: trained $H_v > 0$, untrained $H_v = 0$. It cannot test the quantitative prediction because a neural network has no well-defined erasure tolerance e. The "constraint structure" imposed by training is a loss-landscape manifold, not an algebraic code—there is no generator matrix, no minimum distance in the Hamming sense, no Singleton bound. The H_v values in the table are also per *check*, where each check reads every weight and the whole training set, not per sampled position as in a code; they measure how much one check reveals, not where the network sits on the frontier.

The quantitative gap between this chapter and the others is honest, not apologetic. The duality predicts a qualitative divide—$H_v > 0$ versus $H_v = 0$—and training produces it. Testing the quantitative law would require defining an effective erasure tolerance for the weight manifold, perhaps via the information-theoretic capacity of the learned representation relative to the ambient parameter space. That definition does not yet exist. The qualitative prediction stands; the quantitative one is open.

What to remember. Training a neural network creates structural constraints that make weight corruption detectable ($H_v > 0$), while random weights have $H_v = 0$. The constraint is functional (the loss landscape), not algebraic (a parity-check matrix), but it serves the same role. Pruning identifies the weights that carry no constraint. Learning is the acquisition of structural constraints—the same mechanism that gives DNA its double entry and codes their verification power.

Chapter 11

The Institutional Double Entry

The question: Do human institutions discover the duality on their own?

In the first century AD, the Roman engineer Sextus Julius Frontinus was appointed *curator aquarum*—superintendent of Rome's aqueducts (Frontinus, 97). He had the lines surveyed and mapped, measured what each delivered against what the records claimed, and hunted down leaks, blockages, and unauthorized taps.

Frontinus was performing a constraint check. The aqueduct's design imposed a structural relationship between the water entering at the source and the water delivered to the city. Any deviation—a leak, a diversion, a blockage—violated this relationship. The inspector who walked the line was sampling positions along the constraint, exactly as a verifier samples positions along a codeword. One structure, two operations: the same engineering that allowed repair of a damaged section (recovery) also allowed detection of theft and degradation (verification).

Twenty centuries later, the U.S. Federal Highway Administration requires that every highway bridge on a public road receive a routine inspection at least once every twenty-four months (U.S. Federal Highway Administration, 2004). The inspection interval is a sampling rate. The structural calculations that predict each member's load capacity are constraints binding the member's condition to the bridge's safety. A member that has corroded beyond its design margin violates the constraint, and the inspection detects the violation.

These are engineering systems. But the duality is not limited to engineered artifacts. Human institutions—governments, markets, scientific communities—develop structural constraints of their own. This chapter examines three.

11.1 Separation of powers

A government with three independent branches—legislative, executive, judicial—is a constrained system. The constraint is constitutional: the branches must agree on the boundaries of legitimate action. A law must be enacted by the legislature, executed by the executive, and upheld by the judiciary. Any single branch that exceeds its authority violates the constraint.

Recovery. If one branch fails temporarily (a government shutdown, a judicial vacancy, a legislative recess), the remaining branches continue to function and can restore normal operations when the failed branch returns. The system tolerates the loss of one component because the other two maintain the constitutional structure.

Verification. If one branch is corrupted—if it exceeds its constitutional authority—the other two can detect the deviation. The judiciary strikes down unconstitutional laws. The legislature can impeach the executive. The executive can veto legislation. Each check is a constraint test: does this branch's action fall within the structure that all three branches define together?

The quantitative model: three branches, each independently evaluating policy against a shared constitutional constraint. If one branch deviates, a random inter-branch check has probability $p = 1/3$ of querying the deviant. The verification entropy per check:

$$H_v = -\log_2\left(1 - \tfrac{1}{3}\right) = -\log_2\left(\tfrac{2}{3}\right) \approx 0.585 \text{ bits}.$$

For $\lambda = 20$ bits of confidence: $s = 20/0.585 \approx 35$ checks.

A single-party state has $n = 1$. There is no second branch to compare against. $H_v = 0$. Deviation from constitutional norms is undetectable from within the system, because the system has no structural constraint to check it against. The transition from autocracy to separated powers is the transition from $H_v = 0$ to $H_v > 0$—the same transition as from one AoA sensor to two (Chapter 17).

This does not mean separated governments never fail. It means their failures are *detectable*. An unconstitutional action in a separated government triggers institutional resistance—court challenges, legislative hearings, public objection—because the structural constraints provide the signal. An unconstitutional action in a single-party state triggers nothing, because there is no constraint to violate.

11.2 Independent replication in science

A scientific finding reported in a single study has $H_v = 0$ for fabrication. The study's data is internally consistent—a competent fabricator ensures this—and there is no external constraint to check it against. Peer reviewers examine methodology and plausibility, but they can rarely detect a careful fabrication by reading the paper, because every fabricated number is consistent with every other fabricated number. (The fabrications that forensic statisticians do catch from the paper alone are the careless ones: means that no integer data could produce, or results too clean for real sampling noise. Real data carry structural constraints, and a sloppy fabricator fails to reproduce them.)

Independent replication is a structural constraint. A second laboratory, using independent equipment and personnel, produces a second measurement of the same quantity. The constraint: the two measurements must agree within statistical tolerance. If they disagree, at least one is wrong.

With two independent measurements:

- **Recovery.** If one laboratory's data is lost, the other's survives.
- **Verification.** If one laboratory's data is fabricated, comparison reveals the discrepancy. Detection probability per comparison: $p = 1/2$ (one of two labs is wrong). $H_v = 1$ bit per comparison.

This is the Pentium FDIV mechanism of Chapter 17: Nicely's independent cross-check computations formed a structural constraint that detected a hardware flaw no single computation could have revealed. Independent

replication is the scientific community's version of the same mechanism.

The replication crisis in psychology, medicine, and other fields is partly a crisis of $H_v = 0$: decades of findings built on unreplicated studies, each internally consistent, none structurally constrained by independent evidence. When replications were finally attempted—the Open Science Collaboration's Reproducibility Project reported in 2015 that only 36% of one hundred replications of psychology studies produced a statistically significant result (Open Science Collaboration, 2015)—the detection rate was high. The constraints, once applied, found the errors that no amount of within-study review could have detected.

11.3 Supply chain diversification

A manufacturer sourcing a critical component from a single supplier has $H_v = 0$ for quality fraud. The supplier delivers components; the manufacturer tests them against specification. But if the supplier systematically provides components that meet specification on the tested parameters while failing on untested ones, the manufacturer cannot detect this by testing more of the same components—every sample is drawn from the same compromised source.

A second supplier adds a structural constraint. The same component, manufactured independently, provides a comparison. If one supplier's components fail in the field while the other's don't, the structural constraint reveals the discrepancy.

With two independent suppliers, each filling half of orders:

- **Recovery.** If one supplier fails—goes bankrupt, suffers a disruption—the other can ramp up. The supply chain tolerates the loss.
- **Verification.** If one supplier's quality degrades, the difference in field performance between the two sources is detectable. Each batch comparison has $p = 1/2$. $H_v = 1$ bit per comparison.

For $\lambda = 10$ bits (99.9% confidence in detecting a quality divergence): $s = 10$ independent batch comparisons. With monthly batches, the divergence is detectable within a year.

Single-sourcing is the supply-chain analog of the 737 MAX's single sensor: $H_v = 0$ for the failure that matters, with the system unable to detect its own vulnerability until the failure arrives.

11.4 The institutional summary

System	p	H_v (bits)	$s_{\lambda=20}$	Structural constraint
Separated powers (3 branches)	1/3	0.585	35	Constitutional cross-check
Single-party rule	0	0	∞	None
Replicated science (2 labs)	1/2	1.000	20	Independent measurement
Unreplicated finding	0	0	∞	None
Dual-source supply	1/2	1.000	20	Independent manufacture
Single-source supply	0	0	∞	None

Every number is reproduced by `code/institutions_hv.py`.

11.5 What institutions add

No one who designed the United States Constitution had read Shannon. No journal editor who mandated replication had derived the Singleton bound. No procurement officer who required dual-sourcing had computed $g(e)$.

Yet all three arrived at the same architecture: structural redundancy that provides both recovery (continuity when a component fails) and verification (detection when a component deviates). The mechanism is the same in every case: a constraint that binds independent components together, creating a cross-check that no single component can provide for itself.

The institutional evidence extends the duality beyond engineering, biology, and physics. The seven preceding domains might be explained as properties of mathematics (auditing, codes, data availability, consensus), physics (quantum error correction), evolution (DNA), or optimization (neural networks). Institutional design is none of these. It is the product of human experience—centuries of trial and error, constitutional debate, procurement policy, and scientific methodology—converging on the same structure because the structure works.

The double entry was not discovered by mathematicians and exported to other fields. It was discovered independently, in every field, by anyone who needed systems that could both survive and self-correct.

What to remember. Separation of powers, independent replication, and supply chain diversification are institutional instances of the duality. Each adds a structural constraint that provides both recovery and verification. Single-party rule, unreplicated findings, and sole-source procurement are $H_v = 0$ systems—blind to their own failures. The duality is a structural principle, discovered independently wherever systems must be both resilient and self-correcting.

Interlude

Convergence

Eight domains. Eight independent derivations. One equation.

Auditors check ledger entries and use $n = \ln\alpha/\ln(1-p)$. Coding theorists verify codewords and use $s = \lambda/H_v$. Data availability protocols sample shards and use $s = \lambda \ln 2/[-\ln(1-e)]$. Byzantine consensus detects liars by querying random nodes at a rate set by the quorum fraction. DNA polymerase checks each base against a geometric template and achieves 10^{-10} per-base fidelity through three composed layers. Quantum stabilizer codes measure syndrome operators and detect errors at a rate determined by the code distance. Neural networks acquire constraints through training, making corrupted weights detectable in trained models and invisible in untrained ones. And human institutions—separated governments, replicated science, diversified supply chains—discover the same architecture through centuries of trial and error, without any of the mathematics.

Eight names for the same operation: *test a sample against a structural constraint.*

The evidence chapters showed that the formula works. The mechanism chapters that follow ask why. Why must recovery and verification come from the same structure? Why is the factor of two universal? How does composition improve the tradeoff? And what is the k threshold—the minimum number of samples before verification can begin?

The answers turn out to be short. The duality arises because a constraint that determines a missing value must be violated by a wrong value. The factor of two is the cost of not knowing where the problem is. Composition multiplies verification entropy per sample. The k threshold is the cost of learning the reference before you can check against it.

Each mechanism is elementary. Together, they explain why every constrained system in every domain gives the same equation.

Part III

The Mechanism

One pool of constraint bits, two operations. Why the factor of two, how composition helps, and the membership threshold that explains ZODA.

Chapter 12

One Pool, Two Operations

The question: Why must recovery and verification come from the same structure?

The evidence chapters showed the law working in eight domains: auditing, erasure codes, data availability, Byzantine consensus, DNA replication, quantum error correction, neural networks, and human institutions. This chapter asks the deeper question: *why?*

12.1 The constraint pool

An $[n, k, d]$ code has $n - k$ independent constraints (parity checks). These constraints bind the n symbols together, reducing the degrees of freedom from n to k. The "pool" of constraint information is $n - k$ symbols' worth of structural redundancy.

Recovery draws from the pool. If $d - 1$ symbols are erased (their values unknown but their positions known), the remaining $n - (d - 1) = k$ symbols plus the $n - k$ constraints uniquely determine the missing values. Recovery uses the constraints to solve for unknowns. Each constraint consumed in this way provides one degree of freedom toward reconstruction.

Verification draws from the same pool. If a verifier checks s random positions against the code structure, each check tests one constraint. A corrupted symbol violates the constraints involving that position. The constraints that the verifier exercises are the same constraints that would enable recovery.

The two operations are not independent demands on the system. They are two projections of a single resource: the constraint structure. The system does not need one set of constraints for recovery and a separate set for verification. It has one pool, and both operations draw from it.

12.2 Why you cannot have one without the other

Recovery without verification. Suppose a system can recover from $e \cdot n$ erasures but provides zero detection power. This would mean the system has enough constraints to determine $e \cdot n$ unknown values, yet those constraints never reveal a violation when values are wrong. But a constraint that determines a value when it is missing *must* be violated when the value is replaced by something else (over a large enough alphabet). The determination function and the violation function are the same constraint equation.

Over a binary alphabet, a constraint $x_1 \oplus x_2 = 0$ determines x_2 from x_1 (recovery) and detects $x_2 \neq x_1$ (violation). Over $\mathrm{GF}(q)$, a constraint $x_1 + x_2 = c$ determines $x_2 = c - x_1$ (recovery) and detects every wrong value $x_2 \neq c - x_1$ (violation); a uniformly random replacement escapes only when it happens to equal the right value, with probability $1/q$.

Verification without recovery. Suppose a system can detect corruption by sampling but cannot recover from any erasures. This would mean the system has constraints that are violated by corruption, yet those constraints do not determine any missing values. But a constraint that is violated by a wrong value at position i is a relation involving position i and other positions. If those other positions are known (which they are when only position i is erased), the relation can solve for x_i. The detection function implies a recovery function—provided the constraint is invertible with respect to position i, which holds for linear constraints over any field and for general constraints over large alphabets.

Two edge cases. First, over small alphabets ($q = 2$), a constraint like $x_1 \oplus x_2 \oplus x_3 = 0$ detects any single corruption but recovers x_1 only if x_2 and x_3 are both known—recovery requires the other positions in the constraint. This is a limitation of scope (which positions are available), not a violation of the duality. Second, a hash is a constraint that detects but does not recover, because it is a *lossy compression* of the constraint pool—a one-way function, not a structural relation among data symbols. The law applies to structural redundancy, not to external proof artifacts.

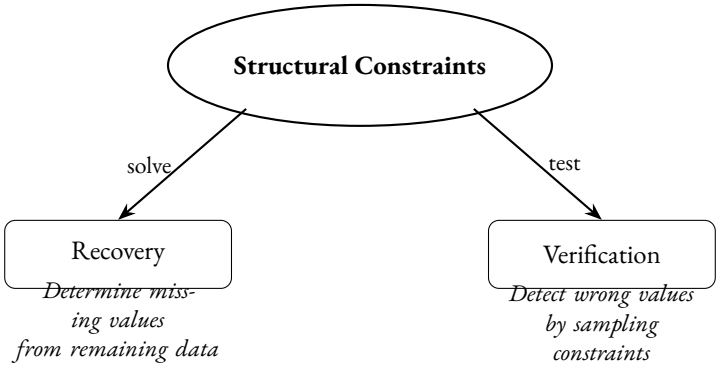


Figure 12.1: One pool of structural constraints serves both recovery (solving for unknowns) and verification (testing for corruption). The two operations draw from the same resource—a constraint that determines a missing value must be violated by a wrong value.

12.3 The formal equivalence

The informal argument above has a precise form.

Recovery–Detection Equivalence *For any constraint system $\mathcal{C} \subseteq \Sigma^n$ with $|\Sigma| \geq 2$:*

1. *If $\mathcal{C}$ corrects $d - 1$ erasures, then $\mathcal{C}$ has minimum distance $\geq d$.*

2. *If $\mathcal{C}$ has minimum distance $\geq d$, then $\mathcal{C}$ corrects $d-1$ erasures.*

Recovery and detection are equivalent, not merely correlated.

Proof of (1). Suppose $\mathcal{C}$ corrects $d-1$ erasures but has two valid configurations c, c' with $d_H(c, c') = j < d$. Erase the $j < d$ positions where they differ plus any $d-1-j$ additional positions (totaling $d-1$ erasures). The surviving $n-(d-1)$ values are consistent with both c and c', contradicting unique recovery. Hence $d_{\min}(\mathcal{C}) \geq d$.

Proof of (2). If $d_{\min} \geq d$ and we erase $d-1$ positions, any valid configuration that matches the $n-(d-1)$ surviving positions is unique: two such configurations would agree on $n-(d-1)$ positions, differing on at most $d-1$, contradicting the minimum distance. So recovery is unique.

The equivalence is tight. The constraint pool of $n-k$ degrees of freedom supports *exactly* the same capacity for recovery (determining unknowns) and detection (finding violations). The two powers are not drawn from separate budgets. They are two readings of one meter.

This equivalence holds over any alphabet, for any constraint system—linear or not, algebraic or not. It requires only that valid configurations be separated in Hamming distance. The duality is a property of distance, which is a property of constraints. It is not a property of the particular algebra used to implement them.

12.4 The accounting perspective

In double-entry bookkeeping, the constraint "total debits = total credits" serves both purposes for the same reason:

- **Recovery:** if the debit side of a transaction is lost, reconstruct it from the credit entry. The constraint *determines* the missing value.
- **Verification:** if a clerk enters an incorrect debit, the totals diverge. The constraint *detects* the inconsistency.

The constraint pool in bookkeeping is the set of balance equations. Each equation is used for recovery (reconstruction of missing entries) and verification (detection of errors) interchangeably. This is not a special property of accounting. It is the duality.

12.5 Pacioli's ledger, by the numbers

The accounting perspective deserves a concrete demonstration. Consider a Venetian merchant's ledger—the system the book is named for—with ten transactions.

#	Transaction	Cash	Counter-entry	Balance
1	Silk purchase	-500	500	0
2	Spice sale	300	-300	0
3	Warehouse rent	-120	120	0
4	Pepper delivery	-200	200	0
5	Insurance payment	-80	80	0
6	Loan repayment	-150	150	0
7	Cloth sale	450	-450	0
8	Ship chandler	-310	310	0
9	Tax payment	-90	90	0
10	Commission received	175	-175	0

Entries are signed, with debits positive and credits negative: buying silk debits the silk account $+500$ and credits cash -500. Every transaction balances. Each balance equation (cash entry + counter-entry = 0) is a structural constraint—a parity check. Each transaction is, in the language of coding theory, a $[2, 1, 2]$ code over the integers: two symbols (the two sides), one degree of freedom (the transaction amount), minimum distance 2.

Recovery. Transaction #4's counter-entry is water-damaged and unreadable. The constraint says counter-entry $=$ $-$cash entry $= -(-200) = 200$. Recovered.

Verification. A clerk enters -50 instead of -500 for transaction #1's cash entry. Balance check: $-50 + 500 = 450 \neq 0$. Error detected.

The same constraint reconstructed the missing entry and detected the wrong one. This is the pool: one equation, two operations.

For a large ledger of N transactions, an auditor spot-checking random transactions for balance uses the formula from Chapter 4: $s = \ln \alpha / \ln(1 - p)$, where p is the assumed error rate. For $N = 1{,}000$ transactions, a 5% error rate, and 95% confidence: $s = 59$ transaction checks. This is the same number as the AICPA table—because it is the same formula, applied to the kind of system auditors use it on. (The ledger size N does not enter: for populations much larger than the sample, the formula is independent of population size.)

The numbers are computed by `code/bookkeeping_hv.py`.

12.6 Constraints as information

The constraint pool is not a metaphor. It is information in the precise Shannon sense.

An unconstrained system of n symbols over an alphabet Σ has $n \log_2 |\Sigma|$ bits of entropy—every configuration is equally valid. A system with $n - k$ independent constraints reduces the valid configurations to $|\Sigma|^k$, cutting the entropy to $k \log_2 |\Sigma|$. The difference—$(n - k) \log_2 |\Sigma|$ bits—is the information content of the constraints. This is real information: it distinguishes valid configurations from invalid ones, it can be used to reconstruct missing values, and it can be used to detect corrupted ones.

Recovery reads this information to fill gaps. If $d - 1$ symbols are erased, the constraint information is sufficient to determine their values—the $(n - k) \log_2 |\Sigma|$ bits of constraint information are at least the $(d - 1) \log_2 |\Sigma|$ bits of missing data, since $d - 1 \leq n - k$ for any code.

Verification reads the same information to find errors. Each random sample tests one constraint, extracting H_v bits of evidence from the pool. After s samples, the verifier has extracted $s \cdot H_v$ bits—enough for λ-bit confidence when $s \cdot H_v \geq \lambda$.

The two operations are not competing for a scarce resource. They are two ways of reading the same stored information. Recovery reads it constructively (solving for unknowns). Verification reads it diagnostically (testing for inconsistencies). The information does not deplete when read—a constraint checked for verification can still be used for recovery, and vice versa. This is why the duality is a theorem and not a tradeoff: you do not sacrifice recovery to gain verification or vice versa. Both are always available, from the same pool, at the costs the law specifies.

A system with no constraints has zero information binding its components together. There is nothing to read—constructively or diagnostically. This is why $H_v = 0$ for independence: not because the sampling is inadequate, but because the information that sampling would read does not exist.

What to remember. Recovery and verification draw from the same pool of structural constraints—which is literally the same information, used two ways. A constraint that determines a missing value when the value is absent must detect a wrong value when the value is corrupted. You cannot have one power without the other, because they are two names for "reading the constraint information." Pacioli's ledger demonstrates this numerically: each balance equation is a $[2, 1, 2]$ code that recovers erased entries and detects corrupted ones.

Chapter 13

The Factor of Two

The question: Why is erasure correction capacity always twice the error correction capacity?

Throughout the evidence chapters, a factor of two kept appearing. The suppression model has $p = e$; the corruption model has $p = e/2$. Tegridy's availability sampling needs 40 samples for 40-bit security; corruption detection of the same code would need roughly twice as many. ZODA's detection fraction at the elementary radius ($1/6$) is half what the suppression fraction would be ($\approx 1/3$).

This factor is not an artifact of the specific codes or protocols. It is the Singleton gap, and it is universal.

13.1 Erasures vs. errors

An *erasure* is a loss at a known position: you know *which* symbol is missing but not *what* it was. An *error* is a corruption at an unknown position: you don't know which symbol is wrong or what the correct value is.

A torn-out page and a page with one wrong figure show the difference. The torn page is easy to repair, because you can see where to look. The wrong figure is harder, because you must first find it among all the right ones.

An $[n, k, d]$ code can:

- Correct up to $d - 1$ erasures.
- Correct up to $\lfloor (d-1)/2 \rfloor$ errors.
- *Or* detect up to $d - 1$ errors (without attempting correction).

The third capability is an either/or: the code can detect $d - 1$ errors or correct $\lfloor (d-1)/2 \rfloor$, but not both at full capacity simultaneously. Using the distance for detection leaves nothing for correction, and vice versa. The law's verification model uses detection only (the verifier flags corruption, not fixes it), so the relevant comparison is between erasure correction ($d - 1$) and error correction ($\lfloor (d-1)/2 \rfloor$).

The erasure capacity is always $2\times$ the error correction capacity. Why?

Information-theoretic reason. Correcting an error at an unknown position requires two pieces of information: (1) *where* the error is, and (2) *what* the correct value should be. Correcting an erasure requires only piece (2)—the position is given. The code's $n - k$ redundant symbols must cover both pieces for errors, but only one piece for erasures. Each redundant symbol contributes one unit of location information or one unit of value information. For erasures, all $n - k$ units go to value recovery. For errors, half go to location and half to recovery, yielding a $2 : 1$ ratio.

This is a heuristic reading of a standard result. The textbook proof is geometric: Hamming balls of radius

$\lfloor (d-1)/2 \rfloor$ around distinct codewords cannot overlap, while erasing $d-1$ known positions can never make two codewords agree. But the heuristic has a universal reading: *the cost of not knowing where the problem is equals the cost of fixing it.* Finding the error is as expensive as correcting it.

13.2 The two detection regimes

The factor of two creates two detection regimes:

Suppression detection (the position of the fault is observable). The verifier asks for a shard and gets silence—the absence is visible. Each query reveals whether that position is faulty with certainty. Detection probability per sample: $p = f/n \approx e$. This is the data availability model and the crash-fault BFT model.

Corruption detection (the position of the fault is hidden). The verifier receives a value and must determine whether it is correct. If the verifier has a commitment (a hash, a Merkle root), each check is an identity test: $p = t/n = e/2$ at the unique decoding radius. If the verifier has only the code structure, membership testing adds a basis cost of k (Chapter 15).

The law unifies both regimes:

$$s = \frac{\lambda}{-\log_2(1-p)}, \qquad p = \begin{cases} e & \text{suppression (known absence)} \\ e/2 & \text{corruption (hidden error)} \end{cases}$$

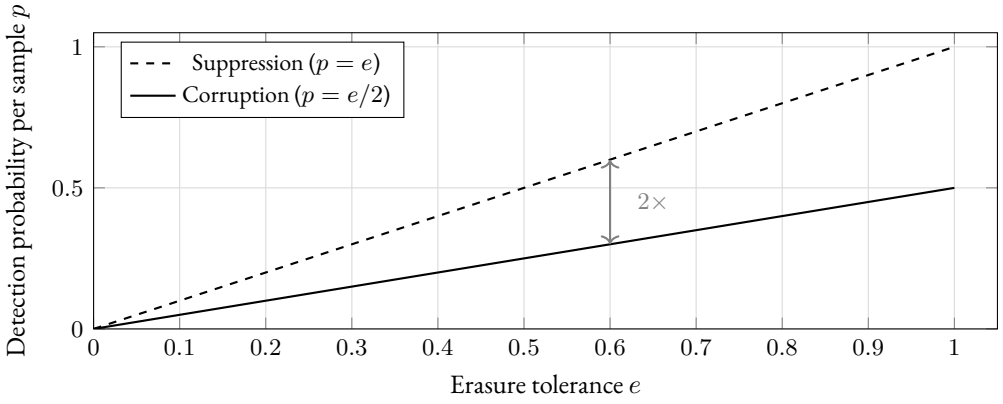


Figure 13.1: The factor of two. Suppression detection (known absence: a shard does not respond) gives $p = e$. Corruption detection (hidden error: a value is wrong) gives $p = e/2$. The gap is the cost of not knowing *where* the problem is.

13.3 Worked example: Tegridy RS(255,128)

Consider Tegridy's $[255, 128]$ Reed–Solomon code with $d = 128$, used in Chapter 6 for data availability. The same code, under two attack models, illustrates the factor of two concretely.

Suppression (a withholding adversary, the DA use case). The adversary withholds 128 of 255 shards—just enough that reconstruction fails. A random shard request either gets a response (the shard exists) or silence (the shard was withheld). Detection probability per sample: $p = 128/255 \approx 0.502$. Verification entropy:

$$H_v = -\log_2(1 - 0.502) = -\log_2(0.498) = 1.006 \text{ bits per sample.}$$

For $\lambda = 40$: $s = 40/1.006 = 40$ samples. The code's `samples_needed` returns exactly 40.

Corruption (a substitution adversary). The adversary replaces $t = \lfloor 127/2 \rfloor = 63$ shards with wrong values, without changing which shards are available. A random check compares a shard to the committed value. Detection probability: $p = 63/255 \approx 0.247 \approx e/2$. Verification entropy:

$$H_v = -\log_2(1 - 0.247) = -\log_2(0.753) = 0.409 \text{ bits per sample.}$$

For $\lambda = 40$: $s = 40/0.409 = 98$ samples.

The ratio: $98/40 = 2.45$. The detection probabilities differ by exactly a factor of two ($p = e$ against $p = e/2$); the sample counts differ by a little more, because $H_v = -\log_2(1 - p)$ is not linear in p. At $e \approx 1/2$ the sample ratio is $-\log_2(1 - e)/-\log_2(1 - e/2) \approx 2.41$ for a code of any length (the floor in t adds the rest), and it approaches exactly 2 only for low-redundancy codes ($e = 0.1$ gives 2.05). Either way, suppression is at least twice as cheap to detect as corruption, because the verifier does not need to find the problem—the problem announces itself.

One code, one erasure tolerance, two attack models, and the factor of two falls out of the arithmetic.

13.4 Is the factor exactly two?

For MDS codes: yes, exactly. The Singleton bound $d = n - k + 1$ is tight, and the unique decoding radius is exactly $\lfloor (d - 1)/2 \rfloor$.

For non-MDS codes: the factor can be *more* than two. A binary code with distance $d < n - k + 1$ has weaker detection than an MDS code with the same redundancy. The factor of two is the *best case*—the minimum gap between what erasure tolerance promises and what corruption detection delivers.

For list decoding (Guruswami–Sudan): the factor can be *less* than two. List decoding corrects more errors than $\lfloor (d - 1)/2 \rfloor$ by returning a short list of candidate codewords rather than a unique one. The effective p is higher than $e/2$, reducing the gap. But list decoding sacrifices uniqueness—you get multiple possible answers, not one.

The factor of two is the price of *unique, unambiguous* recovery. If you accept ambiguity (a list), you can detect more. If you have worse-than-optimal codes, you detect less. MDS codes with unique decoding sit exactly at the $2 : 1$ point.

13.5 Non-MDS codes and list decoding

The law is proved for MDS codes, but most practical codes are not MDS. How does the duality behave for other code families?

Non-MDS codes (LDPC, turbo, polar) achieve distance $d < n - k + 1$. For the same redundancy $n - k$, they have less erasure tolerance ($e < (n-k)/n$) and correspondingly weaker detection ($p = e/2 < (n-k)/(2n)$). The function $g(e)$ still applies at their actual e, but that e is suboptimal—they sit below the frontier (Chapter 18). A CRC-32 checksum on a 1500-byte Ethernet frame, for example, achieves distance $d = 4$ in bits. Its 32 parity bits are 0.27% of the frame, so an MDS code would turn that redundancy into $e \approx 0.0027$; CRC-32 reaches $e \approx 0.00025$, a tenth of that. The shortfall is in the alphabet, not the polynomial: the sphere-packing bound caps *any* binary code with a 32-bit check on a frame that long at $d \leq 6$, so CRC-32 is close to the best a binary code can do. Only large-alphabet codes such as Reed–Solomon reach the Singleton bound (`code/crc_frontier.py`). The duality gives an upper bound on what non-MDS codes can achieve: their $g(e)$ is the same (determined by e alone), but their e is lower than what their redundancy could provide.

List decoding (Guruswami and Sudan, 1999) relaxes the requirement from unique decoding to returning a short list of candidate codewords. The Guruswami–Sudan algorithm corrects up to $n - \sqrt{nk}$ errors for an $[n, k]$ Reed–Solomon code—more than $\lfloor (d-1)/2 \rfloor$ for high-rate codes. The effective detection probability p rises above $e/2$, reducing the gap between suppression and corruption. But uniqueness is lost: the verifier detects that something is wrong without knowing which candidate is correct. In the duality's terms, list decoding trades *unique recovery* for stronger *detection*—spending less of the distance on finding the error's location, freeing more for verification. The factor of two is the price of unique, unambiguous recovery; list decoding pays less by accepting ambiguity.

13.6 Landauer's $\ln 2$

The constant $\ln 2$ that sets the upper bound of $g(e)$ appears in another fundamental result. In 1961, Rolf Landauer proved that erasing one bit of information in a physical system dissipates at least $kT \ln 2$ joules of energy, where k is Boltzmann's constant and T is the temperature (Landauer, 1961). This is not a technological limitation—it is a consequence of the second law of thermodynamics.

The $\ln 2$ in Landauer's bound and the $\ln 2$ in $g(e)$ have the same origin: the conversion between discrete information (bits) and continuous probability (nats). Logarithms base 2 measure information in bits; natural logarithms measure it in nats. The factor $\ln 2 = \log_e 2$ is the exchange rate, and it appears wherever a discrete-information quantity (a bit count, a security parameter) meets a continuous-probability quantity (a detection probability, a thermal fluctuation).

In the duality: the detection probability per sample is $p = e/2$, a continuous quantity. The security parameter λ is measured in bits, a discrete quantity. The formula $s = \lambda / H_v$ bridges the two, and $H_v = -\log_2(1 - e/2)$ contains the $\ln 2$ conversion in its Taylor expansion. The constant $2 \ln 2$ is: the base-conversion price of a bit ($\ln 2$, the same constant Landauer identified as the thermodynamic price of erasure) times the information-theoretic price

of not knowing where the error is ($\times 2$).

Verification has a minimum cost, just as erasure does. Landauer's principle says you cannot erase a bit for less than $kT \ln 2$ joules. The duality says you cannot verify a bit of structural integrity for less than $1/H_v$ samples. Both bounds arise from the relationship between discrete information and continuous probability, and both contain the same constant.

What to remember. Erasure tolerance is exactly $2\times$ error correction capacity because finding a hidden error costs as much as fixing it. This creates two detection regimes—suppression ($p = e$) and corruption ($p = e/2$)—unified by the same formula. The factor of two is tight for MDS codes, the best case for non-MDS, and reducible (at the cost of uniqueness) by list decoding. Non-MDS codes sit below the frontier—same $g(e)$ curve but suboptimal e.

Chapter 14

Composition

The question: What happens when you stack codes—does the duality improve?

ZODA ENCODES data as a matrix and checks it by sampling rows and columns independently. This is a *tensor product* of two codes, and it changes the economics of the duality.

14.1 Tensor product codes

Picture the data laid out in a grid, with redundancy added along every row and down every column. Each row is then a small code of its own, and so is each column. A verifier can check one row, or one column, without reading the rest of the grid.

Given two codes $\mathcal{C}_1$ (the row code) and $\mathcal{C}_2$ (the column code), the tensor product $\mathcal{C}_1 \otimes \mathcal{C}_2$ encodes a $k_1 \times k_2$ data matrix into an $n_1 \times n_2$ codeword matrix via $Z = G_1 \cdot X \cdot G_2^T$.

The key property: every row of Z is a codeword of $\mathcal{C}_2$, and every column is a codeword of $\mathcal{C}_1$. Verification reduces to checking individual rows and columns for codeword membership—a *local* check that reads n_2 symbols (one row) or n_1 symbols (one column), rather than the full $n_1 \cdot n_2$ matrix.

14.2 The locality advantage

For a single code of length $N = n_1 \cdot n_2$, verifying membership requires reading enough of the codeword to compute the syndrome: at least $N - k$ symbols in general. For the tensor code, verifying one row requires only n_2 symbols (and checking a length-n_2 syndrome), while verifying one column requires only n_1 symbols.

This is the real advantage of tensor codes: not a reduction in the total number of samples, but a reduction in *what each sample reads*. Each row check touches $n_2 \ll N$ symbols and provides H_v bits of security. The per-symbol cost of verification drops from $O(N)$ per security bit (for a single code's full syndrome) to $O(\sqrt{N})$ per security bit (for a tensor code's row/column checks on a square matrix).

14.3 Erasure tolerance under composition

For a tensor product of two MDS codes with erasure fractions e_1 and e_2, consider a specific attack model: the adversary erases entire rows or entire columns (not individual symbols in arbitrary positions).

Under this model, the tensor code can recover from any pattern where at most $e_1 \cdot n_1$ rows are fully lost *and* at most $e_2 \cdot n_2$ columns are fully lost. The total fraction of recoverable erasures (by inclusion-exclusion) is:

$$e_{\text{tensor}} = e_1 + e_2 - e_1 \cdot e_2 = 1 - (1 - e_1)(1 - e_2).$$

For two identical rate-$1/2$ codes ($e_1 = e_2 = 1/2$): $e_{\text{tensor}} = 3/4$.

This is the row/column erasure tolerance, which is the relevant model for data availability (the adversary withholds entire shards, which correspond to rows or columns). The tensor product code's minimum distance is $d_1 \cdot d_2$ (the product of the component distances), which governs adversarial symbol-level erasures. For the DA use case, the row/column model is the right one.

14.4 Verification under composition

A verifier samples s_1 random rows and s_2 random columns. Each row check catches a corrupted row with some probability. Each column check catches a corrupted column. To fool the verifier, the corruption must evade *both* the row checks and the column checks.

If the corruption leaves a fraction δ_r of rows as non-codewords and a fraction δ_c of columns as non-codewords, the miss probability is:

$$P(\text{miss}) = (1 - \delta_r)^{s_1} \cdot (1 - \delta_c)^{s_2}.$$

The row and column checks contribute *independently* to the security. The total verification entropy is:

$$H_{v,\text{total}} = s_1 \cdot H_v^{(\text{row})} + s_2 \cdot H_v^{(\text{col})}.$$

For symmetric sampling ($s_1 = s_2 = s$, $\delta_r = \delta_c = \delta$):

$$P(\text{miss}) = \left[(1 - \delta)^2\right]^s, \qquad H_{v,\text{pair}} = -\log_2(1 - \delta)^2 = 2 \cdot H_v^{(\text{single})}.$$

Tensor composition *doubles* the verification entropy per sample pair. Each (row, column) pair provides twice the security of a single axis check, because the two checks are independent.

14.5 The composition law

Combining erasure tolerance and verification:

	Single code	Tensor ($e_1 = e_2 = e$)
Erasure tolerance	e	$2e - e^2$
H_v per sample	$-\log_2(1-\delta)$	$-2\log_2(1-\delta)$
Samples for λ bits	λ / H_v	$\lambda/(2H_v)$ pairs

The tensor code recovers from more erasures (the numerator of $e \cdot s$ increases) and verifies with fewer sample pairs, each of which reads only one row and one column. The law $e \cdot s = g(e) \cdot \lambda$ still holds, but with e and g adjusted for the tensor structure.

A concrete case: take rate-$1/2$ components, so $H_v \approx 0.415$ bits per check, and a target of $\lambda = 20$. A single code needs $20/0.415 \approx 48$ checks. A tensor code gets 0.83 bits from each row-and-column pair, so it needs only about 24 pairs, and it recovers from 75% loss instead of 50%. Counted one read at a time, 24 pairs are 48 row-or-column checks, the same count as the single code: a pair yields twice the evidence because it is two checks. What composition buys is what each check costs and what the code survives. Every check reads one row rather than the whole codeword, and the code survives the loss of three-quarters of its rows and columns. (It also spends more redundancy to get there: two rate-$1/2$ components make a rate-$1/4$ code.)

The composition law generalizes: an L-level tensor code (stacking L codes) multiplies H_v by L per sample group while increasing erasure tolerance toward $1 - (1-e)^L$. The tradeoff is that each "sample" now reads an $(L-1)$-dimensional slice of the data, so the per-sample data cost grows as n^{L-1}.

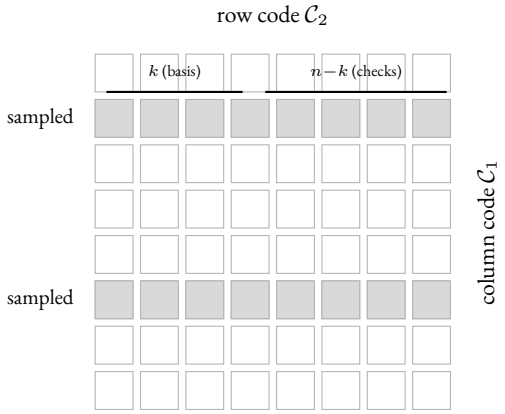


Figure 14.1: Tensor code sampling. Each sampled row reads n symbols: the first k establish the row polynomial (basis cost, paid within the row), and the remaining $n - k$ are constraint checks. Between-row sampling has no basis cost—every row contributes directly to detection.

14.6 Why ZODA uses two levels

The answer is now clear: $L = 2$ gives the best balance. It doubles H_v per sample pair (a genuine security improvement) while each sample reads only $O(n)$ symbols (one row or column), rather than $O(n^2)$ for $L = 3$. The ZODA design is the first rung of the composition ladder—enough to make verification local and double the per-sample

yield, without the cubic data cost of higher levels.

What to remember. Tensor composition doubles verification entropy per sample pair while increasing erasure tolerance from e to $2e - e^2$. The advantage is locality: each sample reads a row (not the whole matrix). ZODA uses $L = 2$ levels—the sweet spot between verification power and per-sample data cost.

Chapter 15

The k Threshold

The question: Why can't you verify without a reference, and what does it cost?

Chapter 5 revealed a sharp discontinuity: membership detection is exactly zero for $s \leq k$ and jumps to match identity detection for $s > k$. This chapter explains why, and what the threshold means for real systems.

15.1 The information-set barrier

You cannot catch a witness in a lie until you know what the story should be. Membership testing has the same problem: before a sample can contradict anything, the verifier must first learn the reference from the samples themselves.

A Reed–Solomon code with parameters $[n, k, d]$ has the property that any k distinct evaluation points determine a unique polynomial of degree $k - 1$. Therefore, any k symbols from any n-symbol word—valid codeword or not—are consistent with *some* codeword.

This is the information-set property: every set of k positions is an *information set*, capable of determining a unique codeword. A verifier with $s \leq k$ samples cannot distinguish a corrupted word from a different valid codeword, because the samples could belong to either.

For Reed–Solomon codes this barrier is exact over any field: two distinct codewords can never agree on k positions, so every k-set is an information set. Field size matters only on the other side of the threshold, where a corrupted value can match the interpolated polynomial by accident with probability about $1/q$. For non-MDS codes, some k-sets are not information sets, and the threshold blurs.

15.2 Two costs of verification

This gives the total cost of membership verification:

$$s_{\text{membership}} = \underbrace{k}_{\text{basis cost}} + \underbrace{\lambda/H_v}_{\text{detection cost}}$$

Basis cost (k). The first k samples establish a reference polynomial. They determine what the codeword "should

be" by interpolation. This cost is paid regardless of the security target—it is structural, not statistical.

Detection cost (λ/H_v). Each sample beyond k is a genuine check: it compares the observed value against the interpolated polynomial. Over a large field, a check that touches a corruption fails to match the interpolated polynomial with probability $1 - O(1/q) \approx 1$. The detection cost follows the law from Chapter 3.

15.3 How tensor codes eliminate the basis cost

In a tensor code, each "sample" is an entire row of n symbols. Since $n > k$ for any nontrivial code, the basis cost is paid within the row itself: the first k symbols determine the row polynomial, and the remaining $n - k$ symbols are checks.

This means the *between-row* sampling faces no basis cost. Every row sample contributes directly to detection. The formula $s = \lambda/H_v$ applies without an additive k term, because the k term is absorbed inside each row.

This is why ZODA can use the simple formula "556 row samples for 144-bit security" without a basis overhead: the overhead is already paid per row.

For single-symbol sampling (the non-tensor case), the k term dominates for small λ. If $k = 128$ and $\lambda = 40$, the detection cost $\lambda/H_v \approx 97$ is comparable to the basis cost $k = 128$. The total is ≈ 225 samples, not 97. The basis cost roughly doubles the naive estimate.

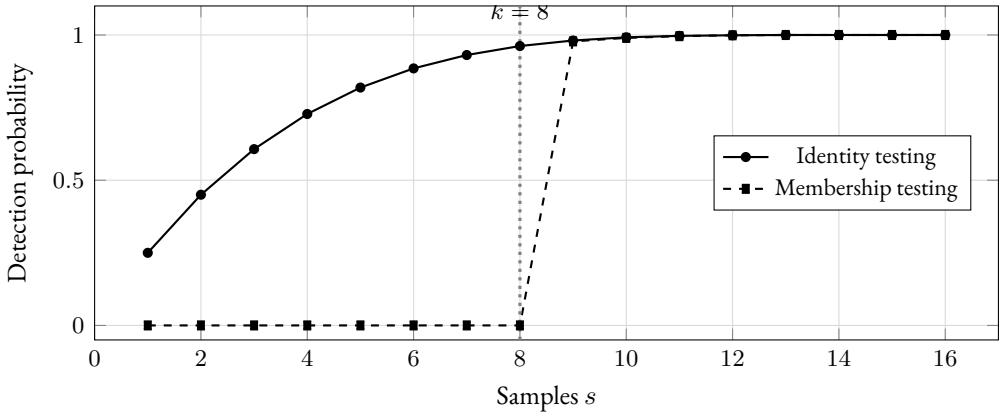


Figure 15.1: Detection probability vs. sample count for $\mathrm{RS}(16, 8)$ with 4 corruptions. Identity testing (solid) rises smoothly from the first sample. Membership testing (dashed) is exactly zero for $s \leq k = 8$—the information-set barrier—then jumps to match identity testing for $s > k$.

15.4 When does the basis cost matter?

- **Large λ, small k:** The detection cost dominates. The basis cost is negligible. Example: $k = 8$, $\lambda = 128$. Detection cost ≈ 310 samples; basis cost $= 8$. Ratio: 2.6%.
- **Small λ, large k:** The basis cost dominates. You need most of your samples just to establish the reference, and only a few for actual checking. Example: $k = 128$, $\lambda = 10$. Detection cost ≈ 24 samples; basis cost

$= 128$. Ratio: 84%.

- **Tensor codes:** The basis cost is always zero for between-row sampling.

The k threshold is invisible in the ZODA use case (tensor rows) and in the audit use case (identity testing against source documents). It matters only when a verifier must learn the code structure from the samples themselves—the purest form of membership testing.

What to remember. Membership testing has a basis cost of k samples (to establish the reference codeword by interpolation) plus a detection cost of λ/H_v (to check against it). Tensor codes eliminate the basis cost by paying it within each row. Identity testing (with a commitment) has zero basis cost. The threshold explains why $s \leq k$ gives zero detection: you can't verify what you haven't yet learned.

Interlude

What Is Missing

The duality has a dark side. It tells you what you cannot see.

Every system in Parts I through III had structural constraints: codes, ledgers, quorums, genomes, stabilizers, weight manifolds. Every one had $H_v > 0$—positive verification entropy, detectable failures, a known cost of verification.

Part IV asks what happens without constraints.

The answer is $H_v = 0$. Zero verification entropy. Zero detection power. No amount of sampling can find what no constraint reveals. This is not a theoretical curiosity. It is the default assumption of most risk models, most financial regulations, and much of engineering practice. Independence is the default. Structural constraints are the exception.

The next chapters prove the theorem, show what it looks like when it kills—and when constraints save—and draw the design frontier.

Part IV

The Connection

What the duality means for invisible risk, what happens when systems lose it, how to find your blind spots, and the design frontier—from ledgers to spacetime.

Chapter 16

The Blind Spot

The question: What happens to systems with no structural constraints?

The Rogue Wave (Zisckind, 2026b) showed that independence misprices compound events exponentially: when d dependent factors each contribute, and the model assumes they are independent, the mispricing grows as $R \approx d^n$. Twenty-nine datasets across eight domains confirmed the pattern. You do not need to have read that book: all this chapter uses is that the independence assumption hides compound risk.

But the Rogue Wave left a question unanswered: *why is the mispricing invisible?* If compound risk is being systematically underestimated by factors of $10\times$ or $100\times$, why doesn't someone notice? Why doesn't routine monitoring, auditing, or stress-testing catch the error?

The duality law answers this question exactly.

16.1 Verification entropy of independence

An independence-assuming system has no structural constraints binding its components together. Each component is modeled as a separate random variable, and the joint distribution is the product of the marginals. There are no parity checks, no balance equations, no quorum intersections—no structural redundancy of any kind.

This requires no extrapolation from the law's formula. It has its own proof.

Independence Blindness *Let $X_1, \ldots, X_n$ be jointly independent random variables over arbitrary domains. Let an adversary corrupt X_j by replacing its value with an arbitrary x'_j. Then for any subset $S \subseteq \{1, \ldots, n\} \setminus \{j\}$, the joint distribution of $(X_i)_{i \in S}$ is identical whether X_j is corrupted or not.*

Proof. By independence, the joint distribution of $(X_i)_{i \in S}$ factors as $\prod_{i \in S} P(X_i = x_i)$, which does not involve X_j. The adversary's action on X_j changes no other variable's distribution. The mutual information between the observed subset and the corruption event is exactly zero: $I(\{X_i : i \in S\};\ \mathbf{1}[X_j \text{ corrupted}]) = 0$. □

Corollary. $H_v = 0$. No finite number of samples from an independence-assuming model can detect corruption in an unsampled component.

This is not a boundary case of the duality law. It is a direct consequence of the definition of independence: if the components are unlinked, observing any subset tells you nothing about any other component. The mutual information is zero—not approximately zero, not negligibly small. Zero. The blindness is total, not asymptotic.

The verification entropy is exactly zero.

This means: *no finite number of random samples can detect a deviation from the model.* The system has no constraints to check. Every observation is consistent with the independence assumption, because the assumption makes no structural prediction about how one component relates to another. Sampling a component tells you about that component and nothing else.

The mispricing is invisible because there is nothing to check it against.

16.2 The compound blindness

This is worse than it sounds. Not only is each individual check uninformative ($H_v = 0$), but combining checks doesn't help:

$$s \cdot H_v = s \cdot 0 = 0 \quad \text{for all } s.$$

No matter how many components you sample, the total verification is zero. The independence assumption is unfalsifiable from within the model, because the model provides no cross-component constraints.

This is the mechanism behind the "calm before the storm" pattern in financial crises, infrastructure failures, and cascading system breakdowns. The system looks fine. Every component checks out individually. The correlations that will produce the compound event are invisible, because the model has no way to see them.

16.3 The contrast

A system *with* structural constraints—double-entry bookkeeping, an error-correcting code, a BFT quorum—is the opposite. Each sample provides $H_v > 0$ bits of verification. Deviations from the structure are detectable by random sampling, at a cost of λ / H_v samples. The structure makes failures *visible*.

	With constraints	Without (independence)
Erasure tolerance e	$(d-1)/n > 0$	0
Verification entropy H_v	> 0	0
Samples for λ bits	$\lambda / H_v < \infty$	∞
Recovery from loss	Yes	No
Detection of corruption	Yes	No

The table is binary: you either have structural constraints or you don't. If you have them, both recovery and verification are available at known costs. If you don't, neither is available at any cost.

16.4 Partial coverage: the realistic case

The table above is binary: constraints or none. Real systems live in between. A financial portfolio model may have structural constraints for some failure modes (mark-to-market cross-checks for individual mispricings) and none for others (no cross-asset correlation constraint). A database may have checksums for bit-rot ($H_v > 0$) but no multi-region replication ($H_v = 0$ for provider outages). An aircraft may have dual engines ($H_v > 0$ for propulsion failure) but a single angle-of-attack sensor ($H_v = 0$ for sensor faults).

The dangerous case is not a system with zero constraints—that system is obviously vulnerable, and no one trusts it. The dangerous case is a system with many constraints covering the *obvious* failure modes and none covering the *compound* ones. The system looks well-monitored because the monitored modes give positive results. The unmonitored modes are invisible precisely because the monitoring works everywhere else.

This was the 2008 CDO architecture. Individual default risk had $H_v > 0$: credit ratings, historical default rates, mark-to-market pricing—all structural constraints that worked well for single-issuer risk. Correlated default had $H_v = 0$. The Gaussian copula did carry a correlation parameter, but it made defaults independent once a single common factor was fixed, and it has zero tail dependence, so joint extreme defaults were modeled as nearly impossible. The parameter itself was calibrated on years without a national housing decline, so the data had nothing to say about the tail. Nothing in the system linked the observed single-name behavior to the joint crash. Every check the system performed returned "normal." The compound event was invisible—not because the checks were sloppy, but because they were checking the wrong constraints.

The blind-spot audit of Chapter 19 is designed for exactly this case. It does not ask "does this system have constraints?"—most systems do. It asks, *for each failure mode separately*, "does this failure mode have a structural constraint?" The blind spots are the zeros in a matrix that is mostly positive. They are the gaps in an otherwise sound defense, and the duality predicts that they are where the next surprise will come from.

16.5 The Rogue Wave connection

The Rogue Wave law says the *cost* of the mispricing is $R \approx d^n$. The duality law says the *reason* the mispricing goes undetected is $H_v = 0$.

Together: independence-assuming systems underestimate compound risk by exponentially growing factors ($R \approx d^n$), and they cannot detect this underestimation by any amount of sampling ($H_v = 0$). The risk is large *and* invisible, for the same structural reason: the absence of constraints that bind the components together.

This is why the Rogue Wave always surprises.

What to remember. Independence-assuming systems have $H_v = 0$: zero verification entropy, zero detection power, no matter how many samples. This is why the Rogue Wave's exponential mispricing goes undetected—there are no structural constraints to check it against. The duality law provides the mechanism behind the Rogue Wave's invisibility.

Chapter 17

When Systems Go Blind

The question: What does $H_v = 0$ look like when it kills?

Chapter 16 proved the theorem: systems without structural constraints have zero verification entropy. No finite number of samples can detect a deviation from the model. This chapter shows what the theorem looks like in practice, in four systems where blindness had consequences.

In every case, the system was monitored. Auditors examined the books. Engineers reviewed the data. Regulators certified the design. The checks that were built into the process found nothing, because they had $H_v = 0$ for the failure that mattered.

17.1 The Gaussian copula (2008)

In 2000, the actuary David Li published a method for modeling the correlation between credit defaults using a Gaussian copula (Li, 2000). Within a few years it had become the market standard for pricing collateralized debt obligations, including those built from mortgage securities. In its industry form, the joint default behavior of thousands of borrowers came down to a single correlation parameter: defaults were independent once one common factor was fixed.

The independence assumption was structural. Given the common factor, nothing linked one borrower's default to another's, and the Gaussian copula has zero tail dependence, so joint extreme defaults were modeled as nearly impossible. The correlation parameter was calibrated on years without a national housing decline. Nothing in the data the model saw could contradict its picture of the tail.

$H_v = 0$. No amount of backtesting on that data could reveal that the correlation structure was wrong. Each tranche's historical performance matched the model. The compound behavior—what happens when housing prices fall nationally and thousands of defaults correlate simultaneously—was invisible, because the model had no structural constraint that would be violated by correlated defaults.

When the compound event arrived, losses on senior tranches that the model had priced as nearly riskless wiped out much of their value. The Rogue Wave law ($R \approx d^n$) quantifies the size of the mispricing. The duality law explains why the pricing models gave no warning.

17.2 Challenger's O-rings (1986)

The Space Shuttle Challenger's solid rocket boosters used O-ring seals between segments. The launch-decision process reviewed O-ring damage flight by flight: erosion or blow-by on one flight was logged, judged acceptable, and flown again.

The structural constraint that was missing from that process: temperature dependence. The O-rings' resilience was coupled to temperature—the colder the rubber, the more slowly it sprang back to seal the joint (Presidential Commission on the Space Shuttle Challenger Accident, 1986). This coupling bound launch conditions to seal integrity, but the assessment did not include it. On the eve of the launch, Thiokol's engineers argued for exactly this constraint, recommending against any launch colder than $53°\mathrm{F}$, the coldest on record. Managers overruled them. The temperature comparison weighed that night considered only the flights that had suffered damage, scattered across temperatures; the flights with no damage, all of them warm, were left off. Plotted together, the dependence is plain.

$H_v = 0$ for the temperature-seal coupling, within the process that made the decision. A review that evaluates each flight on its own terms cannot see a dependence that only appears across flights. The twenty-four successful flights before Challenger were not twenty-four confirmations that the system was safe—they were twenty-four samples with zero verification entropy toward the failure mode that mattered.

The launch on January 28, 1986 proceeded at $36°\mathrm{F}$, colder than any previous launch by more than fifteen degrees. The O-rings failed. Seven crew members were killed.

Richard Feynman's demonstration to the Rogers Commission—dropping an O-ring segment into ice water and showing it lost its resilience—was an act of adding a structural constraint. He created a physical test that bound temperature to seal behavior, giving $H_v > 0$ in front of the commission and the public. The failure became visible the moment someone checked a constraint instead of checking a component. In the duality's terms, Feynman's test was a single sample with detection probability $p = 1$: the O-ring visibly lost resilience in ice water. One sample, one structural constraint, and H_v jumped from zero to certainty. Twenty-four launches reviewed at $H_v = 0$ revealed nothing to the review; one test at $H_v > 0$ revealed everything.

17.3 Therac-25 (1985–1987)

The Therac-25 was a radiation therapy machine that could deliver two types of beams: a low-current electron beam aimed directly at the patient, and an X-ray mode in which a beam roughly a hundred times stronger strikes a metal target that converts it to X-rays (lethal if the target is not in the path). Previous models—the Therac-20 and Therac-6—used hardware interlocks: physical mechanisms that prevented the high-energy beam from firing unless the target was in position.

These interlocks were structural constraints. They bound the beam energy to the collimator state through a physical linkage that could not be bypassed by software errors. $H_v > 0$: any inconsistency between beam energy and target position would be physically blocked.

The Therac-25 replaced the hardware interlocks with software checks (Leveson and Turner, 1993). The software had race conditions: when an operator edited the treatment settings quickly enough, the machine could fire the high-current beam without the target in its path, while the console showed nothing wrong. The structural constraint—the physical binding between beam and target—was gone.

H_v dropped to zero for the beam-target consistency. Six patients received massive radiation overdoses between 1985 and 1987, and at least three of them died as a result. The accidents were difficult to reproduce and diagnose precisely because the software provided no structural verification—each component checked out individually.

17.4 Boeing 737 MAX (2018–2019)

The Maneuvering Characteristics Augmentation System (MCAS) on the Boeing 737 MAX used a single angle-of-attack sensor to determine whether the aircraft's nose was too high, and could push the nose down automatically based on that single reading (Joint Authorities Technical Review, 2019).

One sensor input, no structural redundancy in MCAS's logic. $e = 0$. $H_v = 0$.

The aircraft carried two angle-of-attack sensors, one on each side of the nose. Comparing them is a structural constraint: two measurements of the same quantity, bound by the requirement that they agree. $H_v > 0$. An "AoA Disagree" alert that performs this comparison was meant to be standard on the MAX, but its software had been tied to an optional cockpit display, so on most MAX aircraft it did not work.

MCAS read one sensor. A faulty reading was indistinguishable from a correct one, because there was no second constraint to check it against. The system could not detect its own blindness.

On October 29, 2018 (Lion Air Flight 610) and March 10, 2019 (Ethiopian Airlines Flight 302), faulty sensors caused MCAS to repeatedly push the nose down. The pilots could not diagnose the failure—nothing in the cockpit told them that the one sensor MCAS relied on was wrong—and both aircraft crashed. Three hundred and forty-six people were killed.

The quantitative contrast is exact. Two AoA sensors with a disagree indicator form a system with minimum distance $d = 2$: a persistent fault causes the readings to diverge, and every comparison cycle is a constraint check. With one sensor faulty, each random position check has detection probability $p = 1/2$—one of two readings is wrong. The verification entropy is $H_v = 1$ bit per check, and $\lambda = 20$ bits of confidence—one chance in a million of missing the fault—requires just $s = 20$ checks. At 100 Hz sampling, that is two tenths of a second of flight. The disagree indicator reads *both* sensors per cycle, detecting any fault in a single comparison.

With one sensor: $H_v = 0$. Not 20 comparisons, not 20,000—no number of readings suffices, because every reading is consistent with both a correct and a faulty sensor. The system cannot distinguish the two states.

The second sensor was already on the aircraft. What MCAS lacked was the comparison: a few lines of logic binding its input to the other sensor's reading. The benefit of that comparison is the transition from $H_v = 0$ to $H_v > 0$: from total blindness to reliable detection in under a second. Boeing's post-accident fix does exactly this. MCAS now reads both sensors and stands down when they disagree. The comparison was not a luxury. It was the structural constraint that made verification possible. All numbers are computed by `code/sensor_redundancy.py`.

17.5 When constraints save

The four cases above show what $H_v = 0$ looks like when it kills. But the duality also works in reverse: systems with structural constraints detect failures before they become disasters. The near-misses are quieter than the catastrophes, but they demonstrate the same law.

ECC memory. Error-correcting code memory adds 8 parity bits to every 64-bit word, creating a $[72, 64, 4]$ extended Hamming code. The structural constraint: the 8 parity bits bind the data bits together, giving $H_v > 0$ for single-bit corruption and $H_v > 0$ for detection of two-bit errors. A large-scale study of Google's server fleet found that more than 8% of DIMMs, and about a third of machines, experienced at least one correctable error per year (Schroeder et al., 2011), all of them corrected silently. Without ECC, each of these would be silent data corruption: $H_v = 0$, undetectable, potentially catastrophic. With ECC, the structural constraint corrects every single-bit error and flags every double-bit error. The double entry at work, around the clock, invisibly.

The Pentium FDIV bug (1994). In 1994, the mathematician Thomas Nicely was computing the sum of reciprocals of twin primes, and as a check he computed the sums by two different methods and on more than one machine. The results from a new Pentium disagreed with those from an older 486 processor by far more than rounding could explain. The discrepancy was not in his code—it was in Intel's Pentium processor, which had a flawed lookup table in its floating-point division unit.

Nicely's cross-checks were a structural constraint: two computations of the same quantity must agree. $H_v > 0$. The constraint detected the hardware flaw. Had Nicely run a single computation on a single machine, $H_v = 0$—the wrong answers would have been invisible, indistinguishable from correct ones. One structural constraint, one detection. The same mechanism as Feynman's ice-water test: add a cross-check, and H_v jumps from zero to certainty.

Pacioli's own example. Double-entry bookkeeping has been detecting fraud for five centuries. A clerk who pockets cash and records nothing leaves the cash count short of the books; a clerk who alters one side of an entry breaks the balance. The balance equation is a structural constraint, and any one-sided fraud violates it. A careful fraudster must keep both sides balanced—booking a payment to a fictitious supplier, for example—and the ledger alone cannot catch that. So auditors add a second constraint, tracing entries back to independent evidence: invoices, bank statements, the supplier's own records. This is why auditors check balances: not because the balance tells them everything, but because it has $H_v > 0$ for one-sided errors, and no amount of monitoring with $H_v = 0$ can substitute for a constraint that provides $H_v > 0$.

The contrast is exact. The 2008 CDO models had no constraint tying their tail to anything they could observe. Challenger's launch review had no temperature-seal cross-check. MCAS had no second-sensor comparison. These were systems at $H_v = 0$ for the failure that mattered. ECC memory, the Pentium cross-check, and double-entry bookkeeping are systems at $H_v > 0$ for the failure that mattered. The difference is one structural constraint.

17.6 The pattern

The cases share one mechanism. In every catastrophe:

1. The system lacked structural constraints binding its components together.
2. $H_v = 0$: routine monitoring could not detect the failure mode.
3. The failure arrived as a compound event—correlated defaults, temperature-dependent seal failure, race-condition timing, single-sensor fault.
4. People were checking. The checks the process relied on found nothing, because they were not checking the constraint that mattered. (At Challenger, the engineers who did check it were overruled.)

And in every near-miss, $H_v > 0$ caught the failure before it became a disaster. ECC memory silently corrects errors on about one server DIMM in twelve every year. The Pentium bug was found by one mathematician with one cross-check. Bookkeeping has caught fraud for five centuries.

The fix in every catastrophe was the same: add structural constraints. Tie the model's tail to data that could contradict it. Bind temperature to seal assessment. Use hardware interlocks. Compare both sensors. The moment you add a constraint—any constraint that binds the components together—H_v rises above zero, and the failure becomes detectable.

The duality law does not merely predict how much verification costs. It predicts when verification is impossible: whenever $e = 0$, $H_v = 0$. Systems without structural redundancy are blind, and the blindness is total.

What to remember. Four catastrophes—CDOs, Challenger, Therac-25, 737 MAX—share one cause: $H_v = 0$. Three near-misses—ECC memory, the Pentium FDIV bug, double-entry fraud detection—share the opposite: $H_v > 0$. The difference is one structural constraint. Blindness kills; constraints save. The duality predicts both.

Chapter 18

The Frontier

The question: How should you design a system that needs both recovery and verification?

The law is not only a scientific statement. It is a design tool.

If you know the erasure tolerance your system needs (e), the law tells you the verification cost ($s = \lambda/H_v$, where H_v is fixed by e). If you know the verification budget (s checks at cost c each), the law tells you the maximum security ($\lambda = s \cdot H_v$). If you know the security requirement (λ bits), the law tells you the minimum redundancy and the minimum sample count.

The function $g(e)$ traces a frontier: the best possible tradeoff between recovery and verification for a given redundancy level. Systems on the frontier are MDS codes—they use every bit of redundancy optimally. Systems below the frontier waste redundancy: they pay the cost of the constraints without getting the full verification power.

18.1 Where real systems sit

Reed–Solomon codes. On the frontier. MDS by construction. Every parity symbol contributes maximally to both recovery and detection.

Double-entry bookkeeping. Near the frontier. The constraint "debits = credits" is a single parity check per transaction—a $[2, 1, 2]$ code (Chapter 12), which is MDS. Each constraint is fully utilized for both balance verification and entry reconstruction.

LDPC and turbo codes. Below the frontier. These codes have distance less than $n - k + 1$ (they are not MDS), so their verification power per sample is less than what their redundancy level could provide. They trade optimality for encoding/decoding speed.

CRC-32. Below the frontier. CRC-32 on a 1500-byte Ethernet frame has $d = 4$ (Peterson and Brown, 1961; Koopman and Chakravarty, 2004). It achieves $e \approx 0.00025$, while an MDS code spending the same 0.27% of the frame on parity would achieve $e \approx 0.0027$. The shortfall belongs to binary codes as a class, not to CRC: the sphere-packing bound caps any 32-bit binary check on a frame this long at $d \leq 6$, so only large-alphabet codes reach the frontier. Moreover, CRC requires reading the entire frame to verify; an RS code could verify by sampling individual positions.

TCP checksums. Further below the frontier. The 16-bit one's complement checksum provides ≤ 16 bits of security from reading the entire packet. An MDS code with the same 16 bits of redundancy over a large field would

provide the same security from reading far fewer bytes.

Independence-assuming models. At the origin. $e = 0$, $H_v = 0$. No recovery, no verification.

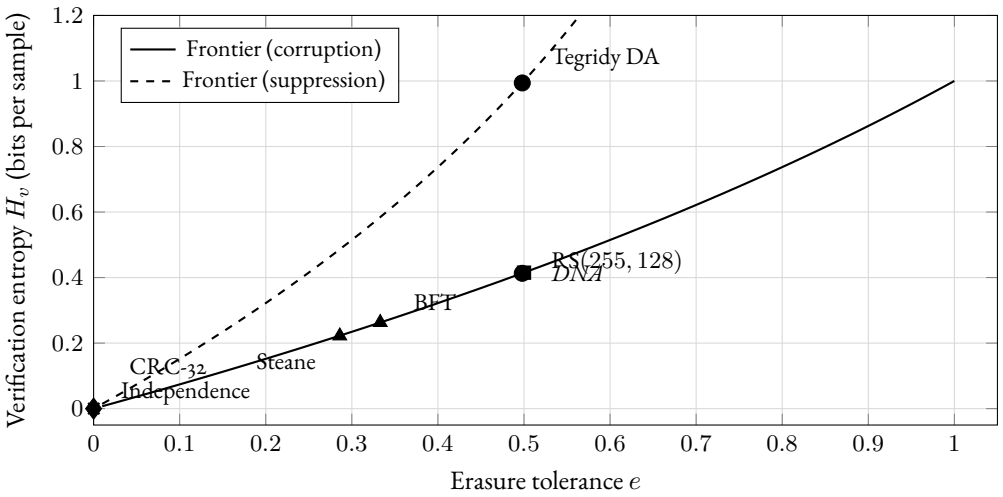


Figure 18.1: The verification frontier. MDS codes (Reed–Solomon) and DNA's Watson–Crick base pair sit on the corruption curve at $e = 0.5$. BFT and the Steane quantum code sit on the curve at their respective e. Tegridy DA sits on the higher suppression curve. CRC-32 sits well below the frontier (distance 4; binary codes cannot approach the Singleton bound). Independence-assuming systems sit at the origin: zero recovery, zero verification.

18.2 Computed positions

The following table places specific systems on the frontier, with all values computed by the script `frontier_comparison.py` in `code/`.

System	e	H_v (bits)	$g(e)$	Position
RS$(255, 128)$	0.498	0.413	1.205	On frontier
RS$(16, 8)$	0.500	0.415	1.205	On frontier
Tegridy DA (suppression)	0.498	1.006	—	On suppression frontier
Ethereum DAS (suppression)	0.500	1.000	—	On suppression frontier
BFT ($n = 3f + 1$)	0.333	0.263	1.267	On frontier
$[\![7, 1, 3]\!]$ Steane code	0.286	0.222	1.285	On frontier
Watson–Crick base pair	0.500	0.415	1.205	On frontier (rate $1/2$)
Double-entry bookkeeping	≈ 0.5	≈ 0.415	≈ 1.2	Near frontier
CRC-32 (1500 B frame)	0.00025	0.00018	1.386	Below (binary, $d = 4$)
Independence models	0	0	—	At origin

Every MDS system lands on the curve. The suppression model (Tegridy DA) sits on the higher suppression frontier—the factor-of-two advantage of Chapter 13. Independence is at the origin: zero recovery, zero verification.

18.3 Moving along the frontier

The frontier is parameterized by $e \in (0, 1]$:

- **Low redundancy** ($e \to 0$): minimal overhead, but each sample provides little verification ($H_v \approx e/(2 \ln 2)$, nearly zero). You need many samples, but each is cheap to produce.
- **High redundancy** ($e \to 1$): massive overhead, but each sample provides nearly 1 bit of verification ($H_v \to 1$). You need few samples, but the system stores mostly redundancy.
- **Moderate redundancy** ($e \approx 0.5$, rate $1/2$): the balanced operating point that most practical systems choose. $H_v \approx 0.415$ bits per sample; $g(e) \approx 1.2$.

The near-constancy of $g(e)$ means the choice of operating point matters less than you might expect. Moving from $e = 0.1$ to $e = 0.9$ changes $e \cdot s/\lambda$ by only 30%. The verification cost is approximately λ everywhere on the frontier.

18.4 Moving above the frontier: composition

Tensor composition (Chapter 14) is the one way to improve the tradeoff. A two-level tensor code of rate-$1/2$ components achieves $e_{\text{tensor}} = 3/4$ (more recovery) with $H_{v,\text{pair}} = 2 \times 0.415 = 0.83$ bits per sample pair (more verification per sample), while each sample reads only $O(\sqrt{N})$ symbols instead of $O(N)$.

Composition doesn't violate the law—it changes the effective parameters. The law $e \cdot s = g(e) \cdot \lambda$ still holds, but with e and H_v adjusted for the tensor structure. The frontier moves outward.

18.5 Design implications

For a system designer choosing a redundancy scheme:

1. **Start with λ.** How many bits of security do you need? This is determined by the threat model, not the code design.
2. **Choose e.** How much loss must the system tolerate? This is determined by the failure environment.
3. **Compute s.** The law gives $s = \lambda/H_v(e)$. This is the verification cost; you cannot reduce it below this bound without sacrificing security or recovery.
4. **Consider composition.** If the per-sample data cost ($O(n)$ for tensor vs. $O(1)$ for position sampling) is acceptable, tensor composition doubles H_v per row-and-column sample pair and makes each check local.
5. **Use MDS if you can.** Non-MDS codes sit below the frontier—they pay the same redundancy cost but get less verification power. MDS codes (Reed–Solomon, Cauchy generator) are optimal unless encoding speed constraints force a suboptimal code.

18.6 Predictions

A framework earns trust by predicting things it was not built to explain. The duality makes three testable predictions.

Verification entropy as a design metric. The law implies that every failure mode of a system has a measurable verification entropy H_v, determined by the structural constraints that bind the components involved. For each failure mode:

1. Identify the structural constraints relevant to it.
2. Compute H_v from the constraint structure.
3. If $H_v > 0$, the failure is detectable by sampling, at known cost λ/H_v.
4. If $H_v = 0$, the failure is undetectable by monitoring. Add a constraint.

This is a blind-spot audit (Chapter 19 works it through in detail): a systematic method for finding the failure modes that routine monitoring cannot see. The CDO correlation structure, the single AoA sensor, the software-only interlock—all had $H_v = 0$ for the failure that mattered. All would be flagged by this metric before the failure occurred. Failure mode and effects analysis (FMEA) already asks how detectable each failure mode is, but it scores detectability by judgment on a ten-point scale; no engineering discipline computes it from the constraint structure.

Sampling beats scanning. Systems with structural redundancy—replicated databases, RAID arrays, distributed storage—currently verify consistency by reading all records (full scans, anti-entropy repair, scrubbing). The law says random sampling of λ/H_v records suffices for λ-bit confidence. For large systems ($N \sim 10^9$ records), the ratio between full-scan cost and sampling cost is on the order of 10^7. Cryptographic storage audits already exploit it: provable-data-possession and proof-of-retrievability schemes check a remote archive by challenging a few hundred random blocks (Ateniese et al., 2007; Juels and Burton S. Kaliski, 2007). Routine consistency checking inside replicated databases and storage arrays still mostly scans everything.

Compound-event blindness is structural, not statistical. The duality predicts that every system with $H_v = 0$ for a compound failure mode will experience that mode as a surprise, regardless of monitoring effort. This is falsifiable: find a system with $H_v = 0$ for a specific failure mode where the failure was detected early by routine monitoring. The historical record (Chapter 17) suggests such cases do not exist.

18.7 Beyond sampling: algebraic verification

The law measures the cost of *sampling-based* verification: testing random positions of a constrained system. Modern cryptography offers a different approach: algebraic proofs that verify an entire computation with $O(\log n)$ or $O(1)$ checks, using the code's algebraic structure rather than random position tests.

ZODA is a bridge between the two. Its verification combines random row-sampling (governed by the law) with algebraic proximity testing (governed by the proximity gap theorem). The sample count 556 follows from $s = \lambda/H_v$, but the proximity gap itself is a theorem of algebraic coding theory that could not be derived from random sampling alone.

SNARKs and STARKs push further: a prover produces a short proof that an entire computation was performed

correctly, and a verifier checks the proof in time $O(\log n)$ or even $O(1)$ regardless of the computation's size. Do these proofs "beat" the law?

No—they change the subject. The law measures the cost of verifying a *system* (a codeword, a ledger, a dataset) by sampling its *positions*. Algebraic proofs verify a *claim* ("I computed $f(x) = y$ correctly") by checking a *proof artifact*. The proof artifact is itself a constrained structure—a polynomial commitment, a Merkle tree of evaluations—and the law applies to its verification. The logarithmic or constant cost comes not from bypassing the law but from *compressing* the constraint structure into a short proof whose own verification entropy per check is high.

A related exclusion: Chapter 12 noted that hashes are "external proof artifacts, not structural constraints within the data." A Merkle root lets you verify any leaf with $O(\log n)$ hashes, but it does not let you *recover* a missing leaf from the other leaves. The hash provides verification without recovery, which appears to violate the duality. It does not: the hash is a one-way compression of the constraint pool, not a structural relation among data symbols. The duality applies to systems where the same constraints enable both recovery and detection. External proof artifacts—hashes, digital signatures, zero-knowledge proofs—enable detection alone, by compressing the system's constraint information into a form that supports verification but not reconstruction.

The law's scope is structural redundancy: constraints within the data that serve both roles. Algebraic proofs and hash-based commitments are powerful tools that live outside this scope. The frontier describes the sampling cost of structural verification; the efficiency of proof-based verification is a different question, with its own frontiers.

What to remember. The frontier $g(e) \cdot \lambda$ is the best tradeoff between recovery and verification for sampling-based verification. MDS codes achieve it; other codes sit below. Composition moves the frontier outward. Algebraic proofs (SNARKs, STARKs) and hash commitments verify by checking compressed proof artifacts, not by sampling data positions—they live outside the law's scope, not above its frontier. For structural design: pick λ, pick e, and the verification cost is determined.

Chapter 19

The Blind-Spot Audit

The question: How do you find the failures your system cannot see?

The duality gives a design tool that engineering has so far applied only by judgment (the ten-point "detectability" score of failure mode and effects analysis is its closest relative): a systematic method for identifying the failure modes that routine monitoring cannot detect. The method is four steps.

1. **List the failure modes.** For each way the system can fail, name it.
2. **Identify the structural constraint.** What binding between components would be violated by this failure? A parity check, a cross-reference, a redundant measurement, a balance equation?
3. **Compute H_v.** If a constraint exists, $H_v > 0$—the failure is detectable, and $s = \lambda / H_v$ gives the monitoring cost. If no constraint exists, $H_v = 0$—the failure is *invisible to monitoring of any kind*.
4. **Flag the blind spots.** Every failure mode with $H_v = 0$ is a blind spot. The fix is always the same: add a structural constraint.

This chapter walks through the method on three systems of increasing complexity. Every number is reproduced by `code/blind_spot_audit.py`.

19.1 Example: home fire safety

A residential home has four failure modes for life-safety hazards:

Failure mode	Structural constraint	H_v	Blind?
Kitchen fire (smoke)	Smoke detector: smoke particles trigger alarm	> 0	No
Carbon monoxide leak	CO detector (if installed)	0^*	**Yes**
Electrical fault in walls	None	0	**Yes**
Natural gas leak	Mercaptan odorant added to gas	> 0	No

*If no CO detector is installed.

The smoke detector is a structural constraint: it binds the state of the air (smoke particles) to an observable (the alarm). The mercaptan odorant in natural gas is a structural constraint that gas companies deliberately *add* to an otherwise odorless substance—they are, in the duality's terms, adding a parity check to a system that would otherwise have $H_v = 0$ for gas leaks.

The blind spots are carbon monoxide (odorless, colorless, undetectable without a dedicated sensor) and electrical faults (hidden inside walls, with no observable until the fault causes a fire). The fix is the same in both cases: add a constraint. A CO detector gives $H_v > 0$ for carbon monoxide. An arc-fault circuit interrupter (AFCI) gives $H_v > 0$ for electrical arcing by monitoring the current waveform for signatures of arcing—a structural constraint binding the current pattern to the circuit's integrity.

The audit reveals the blind spots; the fix is adding constraints. No amount of "checking more carefully" with the existing smoke detector catches carbon monoxide or wiring faults, because those hazards do not violate the smoke detector's constraint.

19.2 Example: replicated database

A cloud database with three replicas and a majority-quorum protocol:

Failure mode	Constraint	H_v	$s_{\lambda=20}$	Blind?
Single-node crash	Quorum intersection	0.585	35	No
Silent bit-rot, 1 replica	Cross-replica checksum	0.263	77	No†
Provider regional outage	None (same provider)	0	∞	**Yes**
App writes bad data	None (replication propagates)	0	∞	**Yes**

†Only if anti-entropy checks are running; without checksums, $H_v = 0$ for bit-rot.

The quorum protocol handles single-node crashes (the standard availability guarantee). Cross-replica checksums, if implemented, detect silent data corruption in one replica. But two failure modes have $H_v = 0$:

Provider outage. All three replicas live in one cloud region. They share a power grid, a network backbone, and a provider's control plane. These are *not* independent—they are coupled by shared infrastructure. The quorum protocol has no structural constraint that would be violated by a simultaneous outage, because it was designed for independent failures. Fix: multi-region or multi-provider replication, which adds a constraint (at least one replica must be in a different failure domain).

Logical corruption. An application bug writes incorrect data, and replication faithfully copies the error to all three replicas. The replicas agree—they all have the same wrong data—so the quorum is satisfied. The structural constraint (quorum agreement) is not violated. Fix: application-level invariants (foreign-key constraints, range checks, consistency rules) or point-in-time recovery, which adds a *temporal* constraint (the current state must be reachable from a known-good historical state by a valid sequence of operations).

19.3 Example: portfolio risk model

A financial portfolio of 100 credit assets under a Value-at-Risk model that treats defaults as independent:

Failure mode	Structural constraint	H_v	Blind?
Single-asset mispricing	Mark-to-market vs. exchange	0.415	No
Concentration risk	Position limits per sector	> 0	No
Correlated default	None (independence assumed)	0	**Yes**

The first two failure modes are caught by existing constraints. Mark-to-market pricing cross-references internal valuations against exchange prices—a double entry with $e = 0.5$ (the exchange price is the second entry), giving $H_v = 0.415$ bits per spot check. For $\lambda = 20$: $s = 49$ random checks suffice. Position limits bind sector weights to a ceiling; a violation is immediately visible.

The blind spot is correlated default. This model assumes $P(A \text{ defaults and } B \text{ defaults}) = P(A) \times P(B)$—independence. (Many production models do estimate correlations, but from calm-period data, which leaves the same blind spot in the tail.) No structural constraint binds the default of one asset to the default of another. Each asset's marginal default probability can be verified individually, but the joint behavior is invisible. This is the CDO mechanism of Chapter 17: every individual tranche checks out, the compound event is invisible, and when it arrives, losses exceed the model by orders of magnitude.

The fix: add cross-asset structural constraints. Stress tests that *impose* a correlation structure ("what if housing prices fall 20% nationally and defaults correlate at 0.6?") create a structural constraint between assets. The stress test is a constraint check: if the portfolio survives the imposed correlation, the constraint is satisfied; if it doesn't, the blind spot is exposed. The fix converts $H_v = 0$ to $H_v > 0$—not by improving the monitoring of individual assets, but by adding a constraint that binds them together.

19.4 The method

The three examples share a pattern:

1. *The obvious failures are already handled.* Smoke detectors, quorum protocols, mark-to-market pricing—these are constraints that engineering practice has already installed. They give $H_v > 0$ for the failure modes they were designed to catch.
2. *The blind spots are where constraints are absent.* Carbon monoxide, correlated infrastructure, correlated default—these are failure modes with no structural constraint. $H_v = 0$.
3. *The fix is a constraint, not more monitoring.* Checking the existing constraints more frequently does nothing for $H_v = 0$ failure modes. The fix is adding a new constraint: a CO detector, a second cloud region, a stress test with imposed correlations.

A blind-spot audit does not replace domain expertise. It provides a systematic language for the question that domain experts ask informally: "what are we not checking?" The duality makes the answer precise: you are not

checking the failure modes for which $H_v = 0$, because checking them is mathematically impossible without adding a constraint.

19.5 Designing the constraint

The audit finds the zeros. What constraint should you add?

Not all constraints are equal. Four principles distinguish a good constraint from a weak one.

Independence from the failure path. A constraint must survive the event it is meant to detect. A credit-rating agency funded by the bond issuer is a "constraint" that collapses under the same stress that causes the default—the rater has an incentive to suppress the signal precisely when the signal matters. A cross-check that shares a failure mode with the system it monitors has $H_v = 0$ for that mode, even if its H_v is positive for others. The constraint must be structurally independent of the failure it targets. Feynman's ice-water test worked because the physics of rubber resilience is independent of NASA's launch-commit process. A second AoA sensor works because its failure mode (mechanical fault in a different physical unit) is independent of the first sensor's failure. The CDO correlation constraint failed because it was embedded in the same model that produced the mispricing.

Efficiency. A constraint that uses $n - k$ symbols of redundancy should achieve distance $d = n - k + 1$ (the Singleton bound). A CRC-32 on a 1500-byte frame spends 0.27% of the frame on redundancy and can recover only a tenth of that share; a Reed–Solomon code turns every parity symbol into a recoverable symbol (Chapter 18). When you add a constraint, choose one that extracts full verification power from the redundancy invested. In institutional terms: a rubber-stamp review board uses organizational overhead (the "redundancy") but provides no real detection (d far below what the overhead could support). An independent audit with real authority achieves the Singleton bound of oversight.

Composability. Multiple constraints should compose multiplicatively, like DNA's three layers of error correction (Chapter 8). Two constraints that share information—that are correlated with each other—provide less than the sum of their individual H_v values. Two constraints that are independent double H_v, the same way tensor composition doubles verification entropy per sample pair (Chapter 14). When adding a second constraint, verify that it is independent of the first. A second smoke detector in the same room adds less H_v than a CO detector in a different room, because the second smoke detector is correlated with the first (both respond to the same failure mode) while the CO detector is independent (it responds to a failure mode the smoke detector cannot see).

Robustness. The constraint itself can fail. A smoke detector runs out of batteries. A second supplier goes bankrupt. A cross-check is overridden by an operator who trusts the primary reading. The duality applies to the constraint's own integrity: if the constraint has no structural protection against its own failure, then its H_v can drop to zero without warning. This is why hardware interlocks (Therac-25, Chapter 17) are more robust than software checks for safety-critical functions—a physical mechanism cannot be bypassed by a race condition.

The four principles reduce to one: *the constraint is itself a system, and the duality applies to it.* A good constraint has $H_v > 0$ for its own failure modes, is independent of the system it monitors, extracts full detection power from its overhead, and composes with other constraints rather than duplicating them.

What to remember. The blind-spot audit: list failure modes, identify constraints, compute H_v, flag the zeros. The fix is a new constraint, designed by four principles: independence from the failure path (the constraint must survive the event it detects), efficiency (extract full detection from the redundancy invested), composability (new constraints should be independent of existing ones), and robustness (the constraint itself must be protected). The pattern: the obvious failures are caught by existing constraints; the invisible ones are caught by no constraint at all; the fix is a constraint that is itself structurally sound.

Chapter 20

The Holographic Double Entry

The question: Does the duality extend to spacetime itself?

In 2014, three physicists at Stanford and Princeton published a paper (Almheiri et al., 2015) that recast one of the deepest ideas in theoretical physics as an error-correcting code.

The idea is the AdS/CFT correspondence, proposed by Juan Maldacena in 1997 (Maldacena, 1999): a theory of quantum gravity in a volume of spacetime (the "bulk") is equivalent to a quantum field theory on the boundary of that volume. The bulk is a higher-dimensional space described by general relativity. The boundary is a lower-dimensional space described by a conformal field theory. Everything that happens in the bulk is encoded, redundantly, in the boundary.

Ahmed Almheiri, Xi Dong, and Daniel Harlow showed that this encoding has the structure of a quantum error-correcting code. The bulk is the logical information; the boundary is the physical qubits. The holographic map—the correspondence itself—is the encoding.

The connection to the duality is immediate.

20.1 Recovery and verification in holography

The picture to keep in mind is a hologram: a flat surface that stores a full three-dimensional scene, so that any large enough piece of the surface can reproduce the part of the scene behind it. The boundary plays the surface and the bulk plays the scene.

Recovery. A subregion of the boundary can reconstruct the corresponding bulk region—the "entanglement wedge." If part of the boundary is lost (erased), the bulk information it encoded can be reconstructed from the remaining boundary, up to a limit set by the geometry. The reconstruction is determined by the Ryu–Takayanagi surface: the minimal surface in the bulk whose area, divided by $4G$ (four times Newton's constant), equals the entanglement entropy of the boundary subregion.

This is erasure correction. The boundary is the codeword; the bulk is the message; the holographic map is the encoding; and the Ryu–Takayanagi surface determines how much of the boundary can be lost while the bulk remains reconstructible.

Verification. If the bulk state is inconsistent with the boundary data—if something "goes wrong" in the interior—this inconsistency is detectable from boundary measurements. The same holographic map that enables

reconstruction also enables consistency checking, because the map is a code, and codes have minimum distance.

One structure—the holographic correspondence—two powers: recovery and verification.

20.2 The HaPPY code

In 2015, Pastawski, Yoshida, Harlow, and Preskill (Pastawski et al., 2015) constructed a concrete holographic error-correcting code (the "HaPPY code") built from perfect tensors arranged on a hyperbolic tiling. The HaPPY code has explicit parameters:

- Physical qubits live on the boundary of the tiling.
- Logical qubits live in the bulk (one on each interior tensor of the tiling).
- The code distance grows with the tiling depth, and the entanglement wedge structure matches the geometric predictions of AdS/CFT.

For the HaPPY code, the erasure tolerance e is determined by the geometry: how large a connected boundary region can be erased while the complementary region still contains the entanglement wedge of every bulk logical qubit. For a symmetric tiling of depth L, the effective e depends on L and the branching factor.

20.3 The conjecture

The duality makes a specific, testable prediction for holographic codes:

Holographic Duality Conjecture *If the holographic code is an instance of the redundancy duality, then its verification cost—the number of boundary measurements needed to detect a bulk inconsistency—satisfies $s = \lambda / H_v$, with H_v determined by the code's effective erasure tolerance e, and $g(e) \in [1,\ 2\ln 2]$.*

The erasure tolerance e is set by the geometry: the maximum fraction of the boundary that can be lost while the bulk remains reconstructible. For the HaPPY code, this is computable from the tiling parameters. The conjecture says that once you know e, the verification cost follows the same law as Reed–Solomon codes, audit sampling, and DNA repair.

This prediction is testable within the AdS/CFT framework—not experimentally, but mathematically. Compute e for the HaPPY code at a given depth. Compute H_v from the formula. Check whether the actual verification power of boundary measurements matches $s = \lambda / H_v$. The calculation requires tools from quantum information theory (the entanglement wedge, the Ryu–Takayanagi formula, and the code's syndrome structure), but the prediction is quantitative and falsifiable.

We leave the computation to the quantum-gravity community. What we claim is the prediction itself: *if the*

holographic correspondence is a code, then it satisfies the redundancy duality, and the constant $g(e)$ is bounded by $[1,\ 2\ln 2]$.

20.4 What holography means for the duality

If the conjecture holds, the scope of the duality extends from bookkeeping ledgers to the fabric of spacetime. The structural constraints of general relativity—the Einstein equations, which bind the geometry of spacetime to the energy-momentum of matter—would serve as both recovery constraints (enabling reconstruction of the bulk from the boundary) and verification constraints (enabling detection of bulk inconsistencies from boundary measurements).

This would mean the duality is not an engineering principle. It is a physical one. The same law that governs audit sampling, DNA repair, and data availability governs the error-correcting structure of spacetime. The double entry is not something humans invented. It is something physics provides.

Quantum mechanics already showed this direction (Chapter 9): the no-cloning theorem eliminates backup copies, leaving structural constraints as the only protection for quantum information. Holography extends the argument: the structural constraints of spacetime *are* an error-correcting code, and the duality predicts that this code has the same verification-cost structure as every other constrained system.

Whether this is the final reach of the duality—or whether deeper structures in physics exhibit it too—is an open question. The duality's pattern is clear: wherever structural constraints bind parts of a system together, recovery and verification follow, at a cost of approximately λ. If spacetime has structural constraints—and general relativity says it does—then spacetime has the double entry.

What to remember. The holographic correspondence (AdS/CFT) is a quantum error-correcting code. Recovery: reconstruct the bulk from the boundary. Verification: detect bulk inconsistencies from boundary measurements. The conjecture: $g(e) \in [1,\ 2\ln 2]$ for holographic codes, with e set by the entanglement wedge geometry. If it holds, the duality extends from ledgers to spacetime.

Chapter 21

The Double Entry

The question: What does the law mean?

In 1494, Luca Pacioli wrote down a bookkeeping method that Venetian merchants had used for two centuries. He called it the method of Venice. He did not know it was a law of nature.

In 1948, Claude Shannon proved that reliable communication requires redundancy. He did not state—though his mathematics implied—that the redundancy automatically provides verification at no additional structural cost.

In 1964, Richard Singleton proved that the distance of a code cannot exceed $n - k + 1$. The bound governs both recovery (erasure correction) and detection (error detection) simultaneously, from one inequality. Singleton did not phrase this as a duality, but it is one.

In 2020, Ben-Sasson, Carmon, Ishai, Kopparty, and Saraf proved proximity gaps for Reed–Solomon codes: when words are combined at random, either almost every combination is close to the code or almost none is. Results of this kind underpin sampling-based checks of encoded data, including ZODA's—the same algebraic structure that enables erasure recovery enables verifiable encoding with zero overhead.

None of these authors stated the principle that unites their results. The principle is this:

The Redundancy Duality *Structural redundancy that enables recovery from loss automatically enables detection of corruption, at a cost that is a near-constant function of the security parameter alone. The constant $g(e)$ varies by at most 38.6% across all possible redundancy levels. Recovery and verification are not two properties of a system. They are one property, measured two ways.*

21.1 What the law gives us

A conservation law. The product $e \cdot s \approx \lambda$ says that verification cost is conserved: regardless of how you distribute redundancy, the total cost of verification is approximately λ. This constrains system design the way conservation of energy constrains physical processes—it tells you what cannot change.

A mechanism for the Rogue Wave. Independence-assuming systems have $H_v = 0$: they cannot detect their own blindness. The exponential mispricing ($R \approx d^n$) is invisible because the system has no structural constraints to reveal it. Adding constraints—any constraints—makes the risk visible.

A design tool. The frontier $g(e) \cdot \lambda$ tells you the minimum verification cost for a given redundancy level, and composition tells you how to move the frontier outward. The k threshold tells you the basis cost of membership testing and explains why tensor codes eliminate it. The blind-spot audit (Chapter 19) makes this operational: for any system, list the failure modes, compute H_v for each, and the zeros are your blind spots.

A unifying framework. Auditing, coding theory, data availability, Byzantine consensus, DNA replication, quantum error correction, neural network training, and institutional design share one equation because they share one structure: random sampling of constrained data. Most of these fields do not cite each other. They don't need to. The structure generates the equation.

21.2 What the law does not give us

This book has been careful about what is proved and what is conjectured.

Proved:

- The law for MDS codes under identity testing, with the bounds $g(e) \in [1, 2 \ln 2]$.
- The general form (Chapter 3): the duality holds for any constraint system with minimum Hamming distance $d \geq 2$, over any alphabet, algebraic or not.
- The algebraic identity between the audit formula and the coding formula.
- The k threshold for membership testing, and the composition law for tensor codes.
- Independence blindness: $H_v = 0$ for jointly independent systems, proved from the definition of independence (Chapter 16).

Confirmed: The audit sampling tables match the formula. The ZODA sample counts match the formula (via the proximity gap). The BFT suppression model matches exactly; the corruption model matches by structural analogy.

Conjectured: The holographic extension—that AdS/CFT's error-correcting code satisfies $g(e) \in [1, 2 \ln 2]$, with e set by the entanglement wedge geometry. The general form proves universality for systems whose distance structure is known; the open question is whether the distance characterization applies to systems (like holographic codes) where identifying d requires new mathematics.

Open: The temporal dimension—how H_v changes as constraints degrade, what "verification debt" means, and how to verify transitions between constraint structures (see "The temporal frontier" below). These questions are beyond this book's scope and are addressed in the subsequent volumes.

Not claimed: Mathematical novelty. The formula $s = \lambda / H_v$ is the geometric distribution. The connection $p = e/2$ is the unique decoding radius, and e itself is bounded by Singleton. The individual results are classical. What is new is the synthesis: the observation that these classical results describe a single universal duality, and that the duality's absence ($H_v = 0$) explains the Rogue Wave.

21.3 The objection and the answer

The most natural objection to this book is: *you wrote a book about* $P(\textit{miss}) = (1-p)^s$.

The objection is fair about the formula. The geometric distribution is taught in every introductory probability course. The individual results—the Singleton bound, the audit sampling formula, the error catastrophe threshold—are classical and well known in their respective fields.

The objection is wrong about the content. What is new is not any formula. It is three observations that no individual field makes:

1. **The duality is a theorem, not a pattern.** Recovery and detection are not two properties that codes happen to have. They are one property (Chapters 12 and 3): any constraint system that recovers from erasures must detect corruption, because the constraint that determines a missing value must be violated by a wrong one. This holds for any system with minimum Hamming distance, over any alphabet, algebraic or not.

2. **The near-constant product predicts verification cost from redundancy alone.** The function $g(e) \in [1, 2\ln 2]$ varies by 39% while e varies from 0 to 1. If you know only the erasure tolerance of a system, you know its verification cost to within 39%, regardless of the system's design, alphabet, or domain. No field states this result, because no field compares across domains.

3. **The absence of constraints predicts invisible failure.** $H_v = 0$ for independence-assuming systems is a theorem (Chapter 16), not an extrapolation. It predicts that systems without structural constraints will be surprised by compound failures—*not because the monitoring was inadequate, but because adequate monitoring is mathematically impossible.* The 2008 crisis, the Challenger O-rings, the 737 MAX: each was a system with $H_v = 0$ for the relevant failure mode. The duality predicts the class of blind spots, not just the cost of verification.

The geometric distribution is the tool. The synthesis is the content. No one who knows only the geometric distribution predicts that DNA's error correction sits at the same frontier position as a Reed–Solomon code, that the Rogue Wave's invisibility follows from the same equation as the auditor's sampling table, or that separation of powers has the same H_v as Byzantine fault tolerance.

21.4 The temporal frontier

This book treats systems as static. A codeword exists; the adversary corrupts it; the verifier samples it. Real systems evolve.

Three questions follow from the duality, and this book does not answer them.

How does H_v change as constraints degrade? A bridge's structural calculations assume a certain steel thickness; corrosion thins the steel; the constraint weakens. A database's checksums assume intact storage; bit-rot accumulates; the constraint erodes. The FHWA's twenty-four-month inspection cycle (U.S. Federal Highway Administration,

2004) is an implicit model of this decay: the structural constraints are strong enough that checking every two years maintains confidence. But the relationship between constraint degradation rate and optimal sampling frequency has no general theory. The duality gives the sampling cost for a fixed e; it does not say how e changes over time.

Is there a "verification debt"? A system that defers constraint checking accumulates risk—not because the probability of failure increases per unit time (it may or may not), but because the probability of *undetected* failure increases. Each deferred check is a sample not taken, a bit of verification entropy not extracted. If the system's H_v is positive but its sampling rate is below λ/H_v, the gap between required and actual verification is a debt whose interest compounds until the next check—or the next failure.

What happens when the constraint structure itself changes? Code updates modify the invariants. Regulatory reform changes the rules. Evolutionary adaptation rewires the fitness landscape. The duality says what verification costs for a given constraint structure; it does not say how the transition between structures should be verified. This is a problem the series returns to: *The Vanishing Point* and *The Broken Staircase* extend the framework to systems that evolve.

21.5 The title, revisited

Double entry. Two lines in Pacioli's ledger. Two powers from one constraint. Two operations on one pool.

Every system that stores, transmits, or processes information faces a choice: build in structural constraints, or don't. If you build them in, you get both recovery and verification—automatically, inseparably, at a cost of approximately λ bits. If you don't, you get neither. You are blind and fragile.

More than five centuries after a friar in Venice wrote down the rules of bookkeeping, the mathematics finally explains why those rules work. The debits balance the credits, and that is enough. One structure. Two powers. Always.

What to remember. The redundancy duality: structural constraints give recovery and verification, inseparably, at cost $\approx \lambda$. The formula is classical; the synthesis is new. Independence has $H_v = 0$—blind and fragile. Constraints have $H_v > 0$—visible and resilient. One structure, two powers.

Afterword: The Series

This book is the second in a series on independence and its consequences.

The Rogue Wave (Book 1) established the law $R \approx d^n$: when d dependent factors are modeled as independent, the mispricing of compound events grows exponentially. Twenty-nine datasets across eight domains confirmed the pattern. The Rogue Wave showed *how large* the mispricing is, but left open why it goes undetected.

The Double Entry (this book) answers that question. The redundancy duality shows that systems without structural constraints have $H_v = 0$—zero verification entropy. The mispricing is invisible because there is nothing to check it against.

Together, the first two books establish a pair of facts: compound risk is exponentially larger than independence assumes ($R \approx d^n$), and invisible to any amount of sampling ($H_v = 0$). Both facts follow from the same structural absence.

The Abundance Gap (Book 3) turns the coin over. If the absence of constraints blinds and impoverishes, what does the *presence* of constraints produce? The answer: knowledge compounds along a production chain, so models that add contributions instead of multiplying them miss the gap ($G \approx m^n$). The Rogue Wave is the cost of missing dependencies. The Abundance Gap is the reward of having them. The double entry connects the two: structural constraints are what make compound knowledge visible, the same way they make compound risk detectable. Systems with constraints compound their advantages; systems without them compound their blind spots. Book 3 shows what that compounding looks like in production, trade, and innovation.

The series continues with *The Vanishing Point* and *The Broken Staircase*, which extend the framework to systems that evolve over time: depletion clocks that treat extraction as independent of the stock when it is in fact a loop, and the question of whether a system that has fallen is pulled back at all.

Each book stands alone. Together, they trace one idea—that independence is a modeling choice with consequences, not a law of nature—through mathematics, evidence, economics, and time.

Glossary

Algebraic verification

Verification via structured proofs (SNARKs, STARKs, Merkle trees) rather than random position sampling. These compress the constraint structure into a short proof whose own verification is governed by the sampling law. Chapter 18.

Basis cost

The k samples required for membership testing before detection begins, where k is the code dimension. Tensor codes eliminate this cost by paying it within each row sample. Chapter 15.

Blind-spot audit

A systematic method for finding undetectable failure modes: list failure modes, identify structural constraints, compute H_v, flag the zeros. The zeros are blind spots—failure modes that monitoring cannot detect at any cost. The fix is a constraint designed by four principles: independence from the failure, efficiency, composability, and robustness. Chapter 19.

Composition

Stacking codes (e.g., tensor product) to multiply verification entropy per sample group while increasing erasure tolerance. Chapter 14.

Constraint design principles

Four criteria for a good structural constraint: independence from the failure path, efficiency (extract full detection from the redundancy invested), composability (new constraints should be independent of existing ones), and robustness (the constraint itself must be protected against its own failure). Chapter 19.

Constraints as information

The $(n - k) \log_2 |\Sigma|$ bits of entropy reduction imposed by a system's structural constraints. This information is the shared resource that both recovery and verification draw from. Chapter 12.

CRC (Cyclic Redundancy Check)

A non-MDS error-detection code using polynomial division. CRC-32 on a 1500-byte frame achieves $d = 4$, near the limit for any binary code of that length ($d \leq 6$ by the sphere-packing bound) but far below what a large-alphabet MDS code gets from the same share of redundancy—it sits well below the verification frontier.

Double entry

(1) Pacioli's bookkeeping rule: every transaction recorded as both a debit and a credit. Each transaction is a $[2, 1, 2]$ code: two entries, one degree of freedom, minimum distance 2. (2) The duality: two powers—recovery and verification—from one structural constraint.

Erasure

A loss at a known position: you know which value is missing but not what it was. Contrast with *error* (corruption at an unknown position). Erasure correction capacity ($d-1$) is always twice the error correction capacity ($\lfloor (d-1)/2 \rfloor$) for MDS codes.

ECC memory

Error-correcting code memory: adds 8 parity bits to each 64-bit word, creating a $[72, 64, 4]$ extended Hamming code. Detects and corrects single-bit errors silently; a near-miss example of $H_v > 0$ at work. Chapter 17.

Erasure tolerance (e)

The fraction of a system that can be lost and still reconstructed. For an $[n, k, d]$ code: $e = (d-1)/n$.

Error catastrophe

Eigen's threshold: the maximum per-base mutation rate a genome of length L can sustain before genetic information disintegrates; approximately $1/L$. Chapter 8.

Factor of two

The universal ratio between erasure capacity ($d-1$) and error correction capacity ($\lfloor (d-1)/2 \rfloor$). Arises because locating a hidden error costs as much as correcting it. Chapter 13.

Frontier

The curve $g(e) \cdot \lambda$: the best achievable tradeoff between recovery and verification for a given redundancy level. MDS codes sit on it; non-MDS codes fall below. Chapter 18.

$g(e)$

The near-constant function $e/[-\log_2(1-e/2)]$, bounded in $[1,\ 2\ln 2]$. Measures the ratio $e \cdot s/\lambda$.

HaPPY code

A concrete holographic error-correcting code (Pastawski, Yoshida, Harlow, Preskill, 2015) built from perfect tensors on a hyperbolic tiling. The duality predicts $g(e) \in [1, 2\ln 2]$ for it; the prediction is untested. Chapter 20.

Holographic duality conjecture

The prediction that holographic codes (AdS/CFT) satisfy the redundancy duality, with verification cost $s = \lambda/H_v$ and $g(e) \in [1, 2\ln 2]$, where e is determined by the entanglement wedge geometry. Chapter 20.

Independence blindness

The theorem that jointly independent random variables have $H_v = 0$. A standalone result from the definition of independence, not an extrapolation of the duality law. Chapter 16.

Institutional double entry

The duality as it appears in human institutions: separation of powers (constitutional cross-checks), independent replication (scientific constraint), and supply chain diversification (manufacturing cross-check). Single-party rule, unreplicated findings, and sole-source procurement are $H_v = 0$ systems. Chapter 11.

Landauer's principle

Rolf Landauer's 1961 result that erasing one bit of information dissipates at least $kT \ln 2$ joules. The $\ln 2$ is the same base-conversion constant that appears in $g(e)$: both arise from the translation between discrete bits and continuous probability. Chapter 13.

Learning (as constraint acquisition)

Training a neural network imposes structural constraints on the weights. Trained weights have $H_v > 0$ (corruption is detectable); untrained weights have $H_v = 0$. Chapter 10.

List decoding

A relaxation of unique decoding that returns a short list of candidate codewords. Corrects more errors than $\lfloor (d-1)/2 \rfloor$ at the cost of ambiguity. Reduces the factor of two between suppression and corruption detection.

Lottery ticket hypothesis

The observation (Frankle & Carlin, 2019) that large neural networks contain small subnetworks that can be trained to full accuracy. The duality interpretation: the winning ticket's weights are the structurally constrained ones ($H_v > 0$); the prunable weights carry no constraint, so removing them is undetectable. Chapter 10.

MDS code

A maximum distance separable code: one achieving the Singleton bound $d = n - k + 1$. Reed–Solomon codes are MDS.

Non-MDS code

A code with $d < n - k + 1$ (not achieving the Singleton bound). Includes LDPC, turbo, and CRC codes. Sits below the verification frontier: same $g(e)$ curve but suboptimal e for the redundancy invested.

Partial coverage

The realistic case where a system has structural constraints for some failure modes ($H_v > 0$) but not others ($H_v = 0$). The dangerous situation: the monitored failures look fine while the unmonitored compound mode is invisible. Chapter 16.

Proximity gap

The minimum fraction of rows that must be non-codewords when a tensor-encoded matrix is corrupted. Follows from proximity-gap theorems of algebraic coding theory (Ben-Sasson et al., 2020; extended to interleaved codes by Diamond and Gruen, 2024) applied to tensor row sampling. Determines H_v for tensor verification.

Recovery–detection equivalence

The theorem that erasure correction and detection are equivalent, not merely correlated: a system that corrects $d - 1$ erasures has minimum distance $\geq d$ (and vice versa), for any constraint system over any alphabet. Chapter 12.

Redundancy duality

The law: structural redundancy that enables recovery from loss automatically enables detection of corruption, at cost $g(e) \cdot \lambda$ where g varies by at most 38.6%.

Security parameter (λ)

The number of bits of certainty required: the probability of a missed corruption is at most $2^{-\lambda}$.

Singleton bound

$d \leq n - k + 1$ for classical codes; $d \leq (n - k)/2 + 1$ for quantum stabilizer codes. Governs both recovery and detection from one inequality.

Stabilizer code

A quantum error-correcting code defined by commuting operators (stabilizer generators) whose measurements serve as both error detection and correction. Chapter 9.

Structural duality (general form)

The extension of the redundancy duality to arbitrary constraint systems: any set of valid configurations with minimum Hamming distance $d \geq 2$ over any finite alphabet has the duality, with $g(e) \in [1,\ 2\ln 2]$. No algebraic structure required. Chapter 3.

Suppression model

Detection of withheld (not corrupted) data, where the absence is directly observable. Detection probability $p = e$ (twice the corruption model), giving $H_v = -\log_2(1 - e)$. Applies to data availability sampling and crash-fault BFT.

Syndrome

The pattern of results from running every consistency check on a word or state. A zero syndrome means every check passes; a nonzero pattern shows that something is wrong and often where. Chapter 9.

Temporal frontier

The open problem of how the duality behaves in evolving systems: constraint degradation over time, verification debt from deferred checking, and transitions between constraint structures. Posed in Chapter 21; addressed in subsequent volumes of the series.

Tensor code

A code built by laying data in a grid and adding redundancy along every row and down every column, so each row and each column is itself a codeword. Lets a verifier check one row without reading the whole grid. Chapter 14.

Unique decoding radius (UDR)

The largest number of corrupted symbols, $\lfloor (d - 1)/2 \rfloor$, for which a decoder can still name exactly one nearest codeword. The law's benchmark corruption: the largest the code is guaranteed to repair, one symbol short of the $\lceil d/2 \rceil$ that can push a decoder toward a wrong codeword. Chapter 2.

Verification entropy (H_v)

The information one random sample provides about whether the system is valid: $H_v = -\log_2(1 - e/2)$ for corruption, $H_v = -\log_2(1 - e)$ for suppression.

ZODA

Zero-Overhead Data Availability: a scheme that encodes data as a matrix (a tensor code) so that sampling rows both protects the data and proves it was encoded correctly. Chapters 6 and 14.

Notes

Chapter 1: Pacioli's Ledger

Pacioli's *Summa de Arithmetica, Geometria, Proportioni et Proportionalita* was published in Venice by Paganino Paganini in 1494. The bookkeeping section, *Particularis de Computis et Scripturis*, spans 27 pages of the original. Pacioli explicitly states he is recording existing Venetian practice, not inventing a new method. Double-entry bookkeeping predates Pacioli by at least two centuries; the earliest surviving examples are from Florentine merchant accounts of the late thirteenth century.

Chapter 2: Shannon's Other Theorem

Shannon's 1948 paper is "A Mathematical Theory of Communication," *Bell System Technical Journal*, 27(3):379–423. The Singleton bound appears in R. C. Singleton, "Maximum distance q-nary codes," *IEEE Trans. Inform. Theory*, 10(2):116–118, 1964.

Chapter 3: The Law

All numerical values are reproduced by `code/es_product.py`. The table of $g(e)$ values matches the output of that script. The proof follows standard coding-theory arguments; the near-constancy of $g(e)$ across $[1, 2\ln 2]$ appears to be new as a stated observation, though the individual bounds are implicit in the literature. The general form of the duality (Section 6) extends the law to arbitrary constraint systems with minimum Hamming distance $d \geq 2$, requiring no algebraic structure. The proof that recovery implies minimum distance $\geq d$ is a standard argument; the observation that it holds for non-algebraic systems over arbitrary alphabets appears to be new as a stated theorem. The "Scope of the quantitative bound" paragraph clarifies that the duality (recovery $\Leftrightarrow$ detection) is proved unconditionally for any system with minimum distance $d \geq 2$, while the specific $g(e) \in [1, 2\ln 2]$ bound requires the corruption model to match the code-theoretic unique decoding radius. For non-algebraic systems, the $g(e)$ bound is a prediction, not a theorem.

Chapter 4: The Auditor's Formula

ISA 530 (Audit Sampling) is published by the International Auditing and Assurance Standards Board (IAASB). The attribute sampling formula $n = \ln\alpha / \ln(1-p)$ appears in the AICPA *Audit Sampling Guide* and in the HUD Office of Inspector General Appendix A (Attribute Sampling). The algebraic identity with the coding formula is

verified in `code/audit_identity.py`. Published sample sizes from the HUD OIG tables match the formula to within rounding.

Chapter 5: Codes and Sampling

Monte Carlo results are from `code/membership_threshold.py`, run over $\mathrm{GF}(257)$ with $5{,}000$ trials per configuration. The membership-testing k threshold is a direct consequence of the information-set property of MDS codes: any k positions form an information set.

Chapter 6: Data Availability

Tegridy's `availability.rs` uses a $[255, 128]$ Reed–Solomon code over $\mathrm{GF}(256)$ with a Cauchy generator matrix. The proximity gap theorem for Reed–Solomon codes is due to E. Ben-Sasson, D. Carmon, Y. Ishai, S. Kopparty, and S. Saraf, "Proximity Gaps for Reed–Solomon Codes," ePrint 2020/654, FOCS 2020.

Chapter 7: Byzantine Generals

The Byzantine generals problem and the $n \geq 3f + 1$ bound are from L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Trans. Programming Languages and Systems*, 4(3):382–401, 1982.

Chapter 8: The Error Catastrophe

Eigen's error catastrophe is from M. Eigen, "Selforganization of matter and the evolution of biological macromolecules," *Naturwissenschaften* 58:465–523, 1971. DNA replication fidelity data (polymerase error rate $\sim 10^{-5}$, proofreading improvement $\sim 100\times$, MMR improvement $\sim 1000\times$) is from T. A. Kunkel, "DNA Replication Fidelity," *J. Biol. Chem.* 279(17):16895–16898, 2004 (PMC3391330). The three-layer composition and the connection to Eigen's threshold are computed in `code/dna_composition.py`. The multiplicative treatment of the three layers is standard: Kunkel's individual improvement factors ($10^{-5} \times 10^{-2} \times 10^{-3}$) match the observed combined rate (10^{-10}), which is the empirical evidence for approximate independence. The layers operate sequentially via physically distinct mechanisms (polymerase active-site geometry, exonuclease excision, MutS/MutL scanning). The interpretation of the three layers as exhibiting the redundancy duality—each layer's detection mechanism being its correction mechanism—is new to this book. The Watson–Crick frontier position ($e = 0.5$, $g = 1.205$) and per-layer security contributions ($16.6 + 6.6 + 10.0 = 33.2$ bits) are computed by `code/dna_frontier.py`. The observation that base-pair complementarity places DNA at the same frontier position as a rate-$1/2$ Reed–Solomon code follows from the shared rate; the per-layer λ decomposition interprets the three repair mechanisms as sequential composition of independent detectors.

Chapter 9: The No-Cloning Duality

The no-cloning theorem is from W. K. Wootters and W. H. Zurek, "A single quantum cannot be cloned," *Nature* 299:802–803, 1982. Quantum error correction: P. W. Shor, "Scheme for reducing decoherence in quantum computer memory," *Phys. Rev. A* 52(4):R2493, 1995; A. Steane, "Error correcting codes in quantum theory," *Phys. Rev. Lett.* 77(5):793, 1996. Stabilizer formalism: D. Gottesman, *Stabilizer Codes and Quantum Error Correction*, PhD thesis, Caltech, 1997 (arXiv:quant-ph/9705052). The quantum Singleton bound $n - k \geq 2(d-1)$ appears in Gottesman (1997) and E. Knill and R. Laflamme, "Theory of quantum error-correcting codes," *Phys. Rev. A* 55(2):900–911, 1997. The numerical confirmation of $g(e) \in [1, 2\ln 2]$ for stabilizer codes $[\![5,1,3]\!]$ through $[\![49,1,7]\!]$ is computed in `code/quantum_check.py`; the observation that the classical duality extends unchanged to quantum codes is new to this book.

Chapter 10: The Weights That Learn

The neural network corruption experiment uses a $6 \to 32 \to 1$ MLP (257 weights) trained in pure NumPy (Adam for 20,000 steps, then 5,000 steps of plain gradient descent); the code is `code/neural_net_hv_membership.py`. The test is deliberately reference-free. Comparing a model against a stored copy of its weights detects any change to any weights, trained or random, so it cannot measure structure. Each class has 40 models: 20 set the threshold (the maximum statistic among them, which targets a false-alarm rate of about $1/21 \approx 5\%$), and the other 20 are held out for testing, so no model is tested against a threshold it helped set. Measured false alarms on the uncorrupted held-out models are 0% for trained and 5% for random, and the random-model detection rates (4–5%) do not exceed that. Thresholds tied to each model's own loss level are not comparable across classes, because a random model's loss is hundreds to thousands of times a converged model's. `code/neural_net_hv_threshold.py` shows that under a shared absolute loss threshold (for a less-trained model), random-weight corruptions actually move the loss more than trained-weight ones, which is why the structural (gradient) check is the right test. With 800 corruptions per noise level, the largest H_v values rest on a handful of misses (three at $\sigma = 1.0$) and are approximate. The specific rates depend on model size, training, and the false-alarm budget; larger models are the natural next test. The lottery ticket hypothesis is from J. Frankle and M. Carlin, "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks," ICLR 2019. The interpretation of training as the acquisition of structural constraints, and the connection to the duality (trained $H_v > 0$, untrained $H_v = 0$), is new to this book. The "What the chapter does not show" section explicitly states the limitation: the chapter tests the qualitative prediction (trained vs. untrained) but not the quantitative one ($g(e) \in [1, 2\ln 2]$), because no well-defined erasure tolerance e exists for a neural network's weight manifold. The quantitative prediction remains open.

Chapter 11: The Institutional Double Entry

Frontinus's *De Aquaeductu Urbis Romae* (c. 97 AD) describes the duties of the *curator aquarum*, including surveying and mapping the lines and suppressing leaks and illegal taps. The Federal Highway Administration's National

Bridge Inspection Standards (23 CFR Part 650, Subpart C, §650.311) require routine inspection of each highway bridge at intervals not exceeding 24 months. The separation-of-powers model ($n = 3$, $p = 1/3$, $H_v = 0.585$) is structurally analogous to the BFT crash-fault model but with different n; the analogy is stated as such, not as an exact derivation. The replication crisis statistic (only 36% of 100 replications produced a statistically significant result) is from the Open Science Collaboration, *Science* 349(6251):aac4716, 2015. The supply chain model ($p = 1/2$, $H_v = 1$ bit per comparison) is the same two-sensor model as the AoA analysis of Chapter 17. All numbers are computed by `code/institutions_hv.py`. The observation that institutional design independently discovers the same architecture as algebraic codes—structural redundancy for both recovery and verification—is new to this book.

Chapter 12: One Pool, Two Operations

The argument that recovery and verification draw from the same constraint pool is a restatement of the relationship between the parity-check matrix and the syndrome in standard coding theory. The "verification without recovery" direction relies on the constraint being invertible with respect to each involved position, which holds for linear constraints over any field. The hash edge case (one-way detection without recovery) is excluded because hashes are external proof artifacts, not structural constraints within the data. The formal equivalence theorem (erasure correction $\Leftrightarrow$ minimum distance for general constraint systems over arbitrary alphabets) is a standard result for linear codes; stating it for arbitrary (possibly non-linear) code families is new to this book. The numerical bookkeeping walk-through (10 transactions, recovery and verification demonstrations) is computed by `code/bookkeeping_hv.py`; the sampling formula for auditing a large ledger is the same formula as Chapter 4.

Chapter 12: One Pool (continued)

The "Constraints as information" section observes that the constraint pool is literally the Shannon information content of the constraints: $(n - k) \log_2 |\Sigma|$ bits, equal to the entropy reduction from n unconstrained symbols to $|\Sigma|^k$ valid configurations. The identification of this information content as the shared resource for both recovery and verification is, to our knowledge, new as a stated observation. The connection to Kolmogorov complexity (constrained systems have lower description length) is noted but not developed; a formal treatment would require specifying the reference universal machine.

Chapter 13: The Factor of Two

The $2 : 1$ ratio between erasure correction capacity and error correction capacity is a standard result in coding theory, proved in any textbook alongside the Singleton bound (e.g., MacWilliams and Sloane, *The Theory of Error-Correcting Codes*, 1977, Chapter 1). The information-theoretic argument (location + value) is due to several authors; see van Lint, *Introduction to Coding Theory*, 3rd ed., 1999, Chapter 1. The "Landauer's $\ln 2$" section connects the base-conversion constant in $g(e)$ to Landauer's principle of minimum erasure dissipation ($kT \ln 2$ per bit); R. Landauer, "Irreversibility and Heat Generation in the Computing Process," *IBM J. Res. Dev.* 5(3):183–191, 1961.

The connection is structural (both constants arise from the discrete-to-continuous conversion between bits and nats), not a claim of physical causation between thermodynamic erasure and verification cost. The Guruswami–Sudan list decoding algorithm is from V. Guruswami and M. Sudan, "Improved decoding of Reed–Solomon and algebraic-geometry codes," *IEEE Trans. Inform. Theory*, 45(6):1757–1767, 1999. CRC-32's minimum distance for various frame lengths is from P. Koopman and T. Chakravarty, "Cyclic redundancy code (CRC) polynomial selection for embedded networks," DSN 2004. The CRC-32 distance tables and the comparison with an MDS code of the same redundancy fraction are computed in `code/crc_frontier.py`. The worked example (Tegridy RS$(255, 128)$ under suppression and corruption, with $s = 40$ vs. $s = 98$ for $\lambda = 40$) demonstrates the factor of two concretely; the suppression count matches the code's `samples_needed` function.

Chapter 14: Composition

The tensor product code construction $Z = G_1 \cdot X \cdot G_2^T$ is standard; the minimum distance of the product code is $d_1 \cdot d_2$. The row/column erasure tolerance formula $e_{\text{tensor}} = 1 - (1 - e_1)(1 - e_2)$ applies to the specific attack model where the adversary erases entire rows or columns, which is the relevant model for data availability sampling.

Chapter 15: The k Threshold

The information-set property of MDS codes (any k positions form an information set) is a direct consequence of the MDS characterization: every $k \times k$ submatrix of the generator matrix is invertible. The membership-testing threshold at $s = k$ and the formula $s_{\text{membership}} = k + \lambda/H_v$ are, to our knowledge, new as stated observations, though they follow directly from the information-set property.

Chapter 16: The Blind Spot

The Rogue Wave law ($R \approx d^n$) is from Tal Zisckind, *The Rogue Wave: How Independence Misprices Compound Events*, Rhombus Tech, 2026. The Independence Blindness theorem and its proof from the factored joint distribution of independent variables are elementary consequences of the definition of independence. Stating them as a formal theorem with a direct proof—rather than as an extrapolation of the law's formula to $e = 0$—establishes $H_v = 0$ for independence-assuming systems as a standalone result, independent of the coding-theoretic framework. The "Partial coverage" section observes that the dangerous case is not a system with no constraints but a system whose constraints cover obvious failure modes ($H_v > 0$) while leaving compound modes unconstrained ($H_v = 0$); the 2008 CDO architecture is the central example. This framing—blind spots as the zeros in a mostly-positive matrix—motivates the per-failure-mode analysis of the blind-spot audit in Chapter 19.

Chapter 17: When Systems Go Blind

The Gaussian copula for CDO pricing is from D. X. Li, "On Default Correlation: A Copula Function Approach," *J. Fixed Income* 9(4):43–54, 2000. The Challenger investigation: Report of the Presidential Commission on the Space Shuttle Challenger Accident, 1986 (the Rogers Commission). The Therac-25 accidents: N. G. Leveson and C. S. Turner, "An Investigation of the Therac-25 Accidents," *Computer* 26(7):18–41, 1993. The Boeing 737 MAX MCAS: Joint Authorities Technical Review, "Boeing 737 MAX Flight Control System: Observations, Findings, and Recommendations," FAA, 2019. The quantitative sensor analysis (two AoA sensors: $H_v = 1$ bit per comparison cycle, $s = 20$ for $\lambda = 20$; one sensor: $H_v = 0$) is computed by `code/sensor_redundancy.py`. The detection model uses $p = 1/2$ (one of two positions corrupted), not the law's corruption model at the unique decoding radius, which is degenerate for $d = 2$ ($t = 0$). The Feynman O-ring observation ($p = 1$ on a single structural test) is a one-sample demonstration of the transition from $H_v = 0$ to $H_v > 0$.

The "When Constraints Save" section adds three positive counterexamples. ECC memory error rates: B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM Errors in the Wild: A Large-Scale Field Study," *Commun. ACM* 54(2):100–107, 2011 (data from Google's fleet). The Pentium FDIV bug was discovered by T. R. Nicely through discrepancies in twin-prime reciprocal sums; Nicely reported the bug to Intel on October 24, 1994, and made it public by email on October 30. The interpretation of Nicely's cross-check as a structural constraint giving $H_v > 0$ is new to this book.

Chapter 18: The Frontier

The "Beyond sampling: algebraic verification" section distinguishes the law's scope (structural sampling of constrained data) from algebraic proof systems (SNARKs, STARKs) and hash-based commitments (Merkle trees). The observation that algebraic proofs compress the constraint structure into a short proof whose own verification is governed by the law is, to our knowledge, new as a stated framing. The hash exclusion (hashes provide verification without recovery) is a consequence of their one-way nature: a hash is a lossy compression of the constraint pool, not a structural relation among data symbols. TCP's 16-bit checksum is specified in RFC 793, §3.1 (1981). The claim that TCP and CRC checksums are "below the frontier" follows from the observation that neither is MDS: CRC-32 achieves $d = 4$ on a 1500-byte frame, while an MDS code spending the same share of the frame on parity would recover ten times as much. No binary code closes that gap: by the sphere-packing bound, 32 check bits on a $12{,}032$-bit word allow at most $d = 6$. CRC distance data: W. W. Peterson and D. T. Brown, "Cyclic codes for error detection," *Proc. IRE* 49(1):228–235, 1961; P. Koopman and T. Chakravarty, DSN 2004. Ethereum PeerDAS (EIP-7594) extends each blob at rate $1/2$ into 128 columns and has each node sample 8 columns per slot (`SAMPLES_PER_SLOT` in the consensus specification); an unrecoverable block must withhold at least 65 columns, so $p \geq 0.51$ and each node gets about 8 bits. Celestia extends a $k \times k$ square to $2k \times 2k$; an unrecoverable square must withhold at least a quarter of its shares, so $p \geq 1/4$, $H_v = 0.415$, and the 16 samples a light node takes give $1 - (3/4)^{16} \approx 99\%$ confidence, the target Celestia states. Both match $s = \lambda / H_v$ with H_v set by the minimum withheld fraction. Blockchain light client numbers are computed in `code/blockchain_light_client.py`. All other frontier positions are repro-

duced by `code/frontier_comparison.py`. The Watson–Crick base pair row in the frontier table is computed by `code/dna_frontier.py`. The predictions section states three testable consequences: H_v as a design metric for blind-spot auditing, the sampling advantage over full-scan verification for replicated systems, and the structural (not statistical) nature of compound-event blindness.

Chapter 19: The Blind-Spot Audit

The blind-spot audit method (list failure modes, identify constraints, compute H_v, flag zeros) is new to this book. All numerical examples are reproduced by `code/blind_spot_audit.py`. The home fire safety example uses standard detection models. AFCI (arc-fault circuit interrupter) technology detects electrical arcing by monitoring current waveforms; the National Electrical Code's 1999 edition required AFCIs on bedroom circuits from January 2002, and later editions extended the requirement to most living areas. The replicated database example uses the BFT crash-fault model from Chapter 7. The portfolio risk model example connects to the CDO analysis of Chapter 17. The closest existing analogue is FMEA (Failure Mode and Effects Analysis), which catalogs failure modes and scores each for severity, occurrence, and detectability; but the detectability score is an expert judgment on a ten-point scale, not a quantity computed from the constraint structure, and it does not separate modes that are hard to detect (H_v small) from modes that are structurally undetectable ($H_v = 0$). The "Designing the constraint" section's four principles (independence from the failure path, efficiency, composability, and robustness) are new to this book and derive from the duality's structure: each principle is an application of the law to the constraint itself rather than to the system the constraint monitors.

Chapter 20: The Holographic Double Entry

The AdS/CFT correspondence was proposed by J. Maldacena, "The Large-N Limit of Superconformal Field Theories and Supergravity," *Int. J. Theor. Phys.* 38:1113–1133, 1999 (arXiv:hep-th/9711200, 1997). The identification of holographic encoding as an error-correcting code is from A. Almheiri, X. Dong, and D. Harlow, "Bulk Locality and Quantum Error Correction in AdS/CFT," *JHEP* 2015(4):163, arXiv:1411.7041. The HaPPY code (a concrete holographic error-correcting code from perfect tensors on a hyperbolic tiling) is from F. Pastawski, B. Yoshida, D. Harlow, and J. Preskill, "Holographic Quantum Error-Correcting Codes: Toy Models for the Bulk/Boundary Correspondence," *JHEP* 2015(6):149, arXiv:1503.06237. The conjecture that $g(e) \in [1, 2\ln 2]$ for holographic codes, with e determined by the entanglement wedge geometry, is new and untested. The specific prediction—compute e for the HaPPY code, predict H_v, check—is stated as a testable conjecture for the quantum-gravity community.

Chapter 21: The Double Entry

The "Temporal frontier" section poses three open questions: (1) how H_v changes as constraints degrade over time, (2) whether a "verification debt" concept is well-defined, and (3) how transitions between constraint structures should be verified. These are marked as open problems for the subsequent volumes in the series. The bridge

inspection interval (FHWA, 24 months) is cited as an implicit model of constraint decay; the relationship between decay rate and optimal sampling frequency has no general theory. The proximity gap for Reed–Solomon codes was proved by E. Ben-Sasson, D. Carmon, Y. Ishai, S. Kopparty, and S. Saraf, "Proximity Gaps for Reed–Solomon Codes," FOCS 2020 (ePrint 2020/654). The observation that this result, combined with the Singleton bound and the geometric distribution, yields a universal duality between recovery and verification is, to our knowledge, new. LDPC codes are not MDS by construction; their iterative decoding performance approaches capacity but their minimum distance grows sublinearly in n. The "objection and the answer" section addresses the observation that the geometric distribution is elementary; the response is that the content is the cross-domain synthesis (universality of the duality, $H_v = 0$ for independence, the same frontier for DNA and RS codes), not the formula. The predictions section of Chapter 18 states three testable consequences of the framework.

Bibliography

Ahmed Almheiri, Xi Dong, and Daniel Harlow. Bulk locality and quantum error correction in AdS/CFT. *Journal of High Energy Physics*, 2015(4):163, 2015. arXiv:1411.7041.

American Institute of Certified Public Accountants. Audit sampling guide. AICPA Professional Standards, 2017.

Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 2087–2104, 2017.

Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS)*, pages 598–609, 2007.

Eli Ben-Sasson, Dan Carmon, Yuval Ishai, Swastik Kopparty, and Shubhangi Saraf. Proximity gaps for Reed–Solomon codes. In *Proceedings of the 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2020. Also available as ePrint 2020/654.

Benedetto Cotrugli. *Libro de l'arte de la mercatura*. 1458. Manuscript completed 1458; first printed Venice, 1573, as *Della mercatura et del mercante perfetto*.

Benjamin E. Diamond and Angus Gruen. Proximity gaps in interleaved codes. *IACR Communications in Cryptology*, 1(4), 2024. Cryptology ePrint Archive, Paper 2024/1351.

John W. Drake, Brian Charlesworth, Deborah Charlesworth, and James F. Crow. Rates of spontaneous mutation. *Genetics*, 148(4):1667–1686, 1998.

Manfred Eigen. Selforganization of matter and the evolution of biological macromolecules. *Naturwissenschaften*, 58(10):465–523, 1971.

Alex Evans, Nicolas Mohnblatt, and Guillermo Angeris. ZODA: Zero-overhead data availability. Cryptology ePrint Archive, Paper 2025/034, 2025.

Jonathan Frankle and Michael Carlin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

Sextus Julius Frontinus. *De Aquaeductu Urbis Romae (On the Aqueducts of Rome)*. 97. Written c. 97 AD by the Curator Aquarum; describes inspection duties including walking the aqueduct lines.

Daniel Gottesman. *Stabilizer Codes and Quantum Error Correction*. PhD thesis, California Institute of Technology, 1997. arXiv:quant-ph/9705052.

Venkatesan Guruswami and Madhu Sudan. Improved decoding of Reed–Solomon and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, 1999.

Richard W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

International Auditing and Assurance Standards Board. ISA 530: Audit sampling. International Standard on Auditing, 2009.

Joint Authorities Technical Review. Boeing 737 MAX flight control system: Observations, findings, and recommendations. Technical report, Federal Aviation Administration, 2019.

Ari Juels and Jr. Burton S. Kaliski. PORs: Proofs of retrievability for large files. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS)*, pages 584–597, 2007.

Emanuel Knill and Raymond Laflamme. Theory of quantum error-correcting codes. *Physical Review A*, 55(2): 900–911, 1997.

Augustine Kong et al. Rate of de novo mutations and the importance of father's age to disease risk. *Nature*, 488: 471–475, 2012.

Philip Koopman and Tridib Chakravarty. Cyclic redundancy code (CRC) polynomial selection for embedded networks. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, pages 145–154, 2004.

Thomas A. Kunkel. DNA replication fidelity. *Journal of Biological Chemistry*, 279(17):16895–16898, 2004. PMC3391330.

Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

Rolf Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191, 1961.

Nancy G. Leveson and Clark S. Turner. An investigation of the Therac-25 accidents. *Computer*, 26(7):18–41, 1993.

David X. Li. On default correlation: A copula function approach. *Journal of Fixed Income*, 9(4):43–54, 2000.

Florence Jessie MacWilliams and Neil James Alexander Sloane. *The Theory of Error-Correcting Codes*. North-Holland, 1977.

Juan Maldacena. The large-N limit of superconformal field theories and supergravity. *International Journal of Theoretical Physics*, 38(4):1113–1133, 1999. Original AdS/CFT proposal; arXiv:hep-th/9711200 (1997).

Thomas R. Nicely. Enumeration of the primes to 10^{14}. *Virginia Journal of Science*, 46(3), 1995. Nicely discovered the Pentium FDIV bug in 1994 through discrepancies in twin-prime reciprocal sums computed by independent methods.

Open Science Collaboration. Estimating the reproducibility of psychological science. *Science*, 349(6251):aac4716, 2015. 36% of 100 replications yielded significant results.

Luca Pacioli. *Summa de Arithmetica, Geometria, Proportioni et Proportionalita*. Paganino Paganini, Venice, 1494.

Fernando Pastawski, Beni Yoshida, Daniel Harlow, and John Preskill. Holographic quantum error-correcting codes: Toy models for the bulk/boundary correspondence. *Journal of High Energy Physics*, 2015(6):149, 2015. arXiv:1503.06237.

W. W. Peterson and D. T. Brown. Cyclic codes for error detection. *Proceedings of the IRE*, 49(1):228–235, 1961.

Jon Postel. Transmission control protocol. RFC 793, 1981.

Presidential Commission on the Space Shuttle Challenger Accident. Report of the presidential commission on the space shuttle Challenger accident. Washington, D.C., 1986.

Irving S. Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. DRAM errors in the wild: A large-scale field study. *Communications of the ACM*, 54(2):100–107, 2011. Based on data from Google's server fleet; originally presented at SIGMETRICS 2009.

Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

Peter W. Shor. Scheme for reducing decoherence in quantum computer memory. *Physical Review A*, 52(4): R2493–R2496, 1995.

Richard C. Singleton. Maximum distance q-nary codes. *IEEE Transactions on Information Theory*, 10(2):116–118, 1964.

Andrew M. Steane. Error correcting codes in quantum theory. *Physical Review Letters*, 77(5):793–797, 1996.

U.S. Department of Housing and Urban Development, Office of Inspector General. Appendix A: Attribute sampling. HUD OIG Audit Guide, 2000.

U.S. Federal Highway Administration. National bridge inspection standards. 23 CFR Part 650, Subpart C, §650.311, 2004. Requires routine bridge inspections at intervals not to exceed 24 months.

J. H. van Lint. *Introduction to Coding Theory*. Springer, 3rd edition, 1999.

William K. Wootters and Wojciech H. Zurek. A single quantum cannot be cloned. *Nature*, 299:802–803, 1982.

Tal Ziskind. *The Abundance Gap*. Rhombus Tech, 2026a.

Tal Ziskind. *The Rogue Wave: How Independence Misprices Compound Events*. Rhombus Tech, 2026b.

Index

$g(e)$, 11
737 MAX, 81

AdS/CFT, 97
algebraic verification, 88
audit sampling, 19

basis cost, 69
Byzantine fault tolerance, 31

Challenger disaster, 80
constraint design, 94
constraint pool, 53
constraints as information, 56
CRC-32, 62, 85

data availability sampling, 27
DNA error correction, 33
double entry, 5
double-entry bookkeeping
    numerical example, 55

ECC memory, 82
Eigen, Manfred, 33
erasure, 59
erasure tolerance, 4, 11
error, 59
error catastrophe, 33

fraud detection, 82
frontier, 85

Gaussian copula, 79

Hamming codes, 7
HaPPY code, 98
holographic duality conjecture, 98
holographic error correction, 97

independence assumption, 75
independence blindness, 75
information-set barrier, 69
ISA 530, 19

Landauer's principle, 62
LDPC codes, 62
list decoding, 62
lottery ticket hypothesis, 43

MDS code, 11

near-miss, 82
neural network, 41
no-cloning theorem, 37
non-MDS codes, 62

Pacioli, Luca, 3
partial coverage, 77
Pentium FDIV bug, 82
proximity gap, 29

quantum error correction, 37
quantum Singleton bound, 38

recovery–detection equivalence, 54
redundancy duality, 11
Reed–Solomon codes, 7, 23
Rogue Wave, 75
Ryu–Takayanagi surface, 97

scientific replication, 46
separation of powers, 45
Shannon, Claude, 4, 7
Singleton bound, 4, 7
SNARKs, 88
stabilizer code, 37
STARKs, 88
structural constraints, 4
structural duality, 14

supply chain, 47
suppression model, 11

TCP checksum, 85
temporal dynamics, 103
tensor product codes, 65
Therac-25, 80

verification entropy, 8
    zero, 75

weight corruption, 41

ZODA, 67