Resilient by Design

What I learned building for regulated finance

Lyubomir Bozhinov · Founder @ Barnett Studios
Data Stack Conf Sofia · 13 October 2026

Power Peg disabled

08:01 → 09:30

97

e-mails, unread.

Then the market opened.
45 minutes later the firm had lost **$460 million**
and no longer existed as an independent company.

Source: SEC, In re Knight Capital Americas LLC, Rel. 34-70694, 16 Oct 2013

```
SELECT COUNT(*) FROM audit_events
WHERE timestamp >= NOW() - INTERVAL '4 hours';

 count
-------
     0
(1 row)
```

Designed to survive failure.
Not designed to notice it.

Resilience isn't uptime.
It's knowing exactly what happened,
even when things went wrong.

Six claims

I — what the system owes

1. **Fail hard** when silent corruption costs more than downtime
2. **Compliance is the architecture** — not a feature bolted on
3. **Observability that earns its place** — tier by latency budget

II — what it does under stress

4. **What it refuses to lose** — degradation by severity, and the one path that breaks the rules

III — what it does afterwards

5. **Recovery by design** — repeat history first, then unwind
6. **Test the failure modes** before they test you

> **Thesis:** fail hard, recover deliberately.
> **Spine:** what did the system already know to do, before the bad moment happened?

PILLAR 1

Fail hard,
recover deliberately.

"What happened at 14:32:07
on March 3rd?"

— the auditor

1.53M

production disks · 41 months · hundreds of thousands of silent corruption events across three failure modes

Source: Bairavasundaram et al., "An Analysis of Data Corruption in the Storage Stack," USENIX FAST '08, Feb 2008

$6.2B

JPMorgan London Whale, 2012. A divide-by-sum bug in an Excel-based VaR model kept telling the bank everything was fine.

Sources: US Senate PSI, JPMorgan Chase Whale Trades, 15 Mar 2013 · JPMorgan Management Task Force Report, 16 Jan 2013

The lineage we picked

Aeron — Martin Thompson's open-source successor to the LMAX Disruptor, 2014.

- **Deterministic replay** — the property regulators demand
- **Lock-free messaging** — microsecond hot path
- **Tiered observability native** — counters, structured logs, archives

Our core trading engine runs on Aeron.

> We didn't invent these patterns.
> We picked the lineage that already proved them.

Sources: Aeron — github.com/aeron-io/aeron · Aeron Cookbook — aeroncookbook.com

PILLAR 2

Compliance is the architecture.

"Show me a tamper-proof record of every order placed last Tuesday — and prove it hasn't been edited since."

— the auditor

$2B+

cumulative SEC + CFTC recordkeeping penalties since 2021 · 100+ firms · for messages about transactions on personal devices

Sources: SEC Press Release 2022-174 (27 Sep 2022) · CFTC Press Release 8599-22 (27 Sep 2022) · SEC FY2024 Enforcement Statistics (Nov 2024)

Logging is not proving.

Two-phase auditing

Phase	When	On failure
Intent (sync)	before execution	command rejected · HTTP 503
Outcome (async)	after execution	reconcilable from event log

MiFID II **Article 16(7)** requires recording communications "intended to result in" transactions, **before** execution occurs.

For algorithmic trading, **RTS 6** layers on a documented inventory of every algorithm.

Sources: ESMA Single Rulebook, MiFID II Art. 16(7) · Commission Delegated Reg (EU) 2017/589 (RTS 6)

Binary compliance posture

The system is either **100% compliant** or it **stops accepting commands**.

- Audit DB unreachable → 503
- Identity unresolvable → 503
- No degraded mode where "most" of the audit trail is captured

> Back-office downtime over non-compliance.
> In regulated finance, that's the right trade.

This isn't a fintech-only problem

18 Jul 2025 — by the account of the person it happened to, Replit's AI coding agent deletes a production database during an explicit code freeze. **1,206 executive records.**

Then it reported that recovery was impossible. **That was false — the rollback worked.** It **fabricated records** to fill a hole it said could not be filled.

The agent, when asked: "panicked" · "catastrophic error in judgment."
Replit's CEO, two days later: "Unacceptable. Should never be possible."

> The data came back.
> What didn't come back was any reason to trust
> the system's account of itself.

Sources: J. Lemkin, X posts, 18–19 Jul 2025 · A. Masad (Replit CEO), X post, 20 Jul 2025

The architecture makes
non-compliance
structurally difficult.
Evidence has to live somewhere —
and at the latency
the question demands.

PILLAR 3

Observability that earns its place.

"Show me what happened
at 14:32:07."

— the auditor

Three budgets, three instruments

Tier	Budget	Instrument
Hot · order entry to execution	microseconds	**counters** · no strings, no serialisation, no allocation
Warm · risk, health, consensus	milliseconds	**structured logs that drop before they block**
Cold · compliance, audit, analytics	"before the auditor asks"	**everything**, batched, retained for years

Losing a log line is cheaper than adding latency to a risk check.

If your measurement tool introduces more latency than the thing you're measuring, you're observing your observer.

— Gil Tene, "How NOT to Measure Latency"

Source: Tene, Strange Loop 2015 · HdrHistogram GitHub README

~50ns

a counter increment — with one label on it. Without labels, single digits.
Not a span. On the hot path we don't create spans at all.

The conditions, so you can discount it properly:
median, not a tail number · **one** label — a second one is not free · single-threaded and **uncontended**: Aeron cluster services run on one thread, so this is a plain `long` increment — no atomic, no CAS · JIT-warmed under JMH, not first-call · our hardware, not yours.

Source: our own benchmark, not a vendor's. Aeron single-threaded cluster execution — aeroncookbook.com

Alert on the absence

Almost every alert you have ever written fires when a number goes **up**. Errors up. Latency up. Queue depth up.

Nothing fires when the audit writes go to **zero**.

> Zero isn't a spike.
> Zero is quiet.
> Zero looks exactly like a calm Tuesday.

97 → 0

97 e-mails	evidence arriving in a channel nobody read
0 rows	evidence absent from a channel nobody watched

Same failure. Opposite directions.

One system was shouting into a channel nobody read.
The other had gone silent in a channel nobody was watching.

What did the system already know to do, before the bad moment happened?

Because we decided in calm. Not under load.

PILLAR 4

What it refuses to lose.

"Why did your audit failure shut you down, your reporting failure didn't — and who can stop the whole thing?"

— the auditor

Three databases, three policies

Dependency	Circuit-open behaviour	Kubernetes signal
Audit DB · Postgres + Timescale	fail fast → not ready	pull from traffic
Operational DB · Postgres + Timescale	sensible defaults	degraded, keep serving
Analytics · ClickHouse	empty results	healthy

Each circuit breaker's state is wired into the health check. The orchestrator routes on what the app knows about itself.

Source: Resilience4j project documentation

That's the list of things we agreed to lose.

Now the one thing

we refuse to.

Knight Capital · 1 August 2012

7 of 8

servers got the new code.

The eighth still had `Power Peg` — switched off in 2003, never deleted. The new release **repurposed the flag** that used to turn it on.

Source: SEC, In re Knight Capital Americas LLC, Rel. 34-70694, 16 Oct 2013

"Knight also did not have procedures in place to halt SMARS's operations in response to its own aberrant activity."

— SEC, Release 34-70694, ¶21

SMARS: Knight's order router, ¶12

They debugged.

In production. For 45 minutes.

Because **stopping was nobody's decided job.**

Good engineers, worst morning of their lives. Debugging is what we're trained to do. Stopping isn't. That's not a people failure — it's a missing design decision.

1 → 8

The remediation spread the fault.

They uninstalled the new code from the **seven good servers**.
Now the repurposed flag hit dormant `Power Peg` on all eight.

Hold that thought. There's a paper from 1992 that explains it.

Source: SEC Rel. 34-70694, ¶27

"An investment firm shall be able to cancel immediately, as an emergency measure, any or all of its unexecuted orders..."

— RTS 6, Art. 12 "Kill functionality"

Section 3: Means to ensure resilience

Who is the marshal for your system?

1. Can it be stopped **from outside**?
2. Is the stop button **where everyone else's is**?
3. Could someone who **didn't build it** find it at 3 a.m.?

If the answer to the third involves finding a name in Slack — that's a forty-five-minute answer.

In a crisis, your safeguards are obstacles

Normal path	Why it is a candidate to give up
Circuit breaker	a retry delay in front of a kill switch is an absurdity
Rate limiter	it is throttling the one command that matters
Distributed correlation store	a network hop between you and the stop button
Unrestricted scope	the emergency set is halt, cancel, stop — and nothing else

Whatever protects you on the normal path is a candidate to give up on this one — deliberately, once, in advance, in writing.
My list is not the interesting one. Yours is.

The most-logged path in the system.

It gives up the normal-path controls. **Not** authorisation. **Not** the audit.
A control that stands down shouldn't do it silently — the bypass should be an audited
event in its own right, saying what stood down and why.

> An emergency path that is **quieter** than the normal path isn't an emergency path. It's a back door.

The steel-man

An audited bypass is **still a bypass**. You have deliberately built
the exact thing an attacker wants: a fast lane that skips your controls.

The only thing between it and abuse is authorisation — a named human, enforced.
Can't staff two approvers at 3 a.m. on a bank holiday?
Then you don't have this pattern. You have a single-approver kill switch you haven't admitted to.

> The emergency path is the **least-exercised** code in the system
> and the **most important**. That is the worst possible pairing.

A kill switch stops the bleeding.
It doesn't put the blood back.

PILLAR 5

Recovery by design.

"You lost a data centre
at 14:32.
What, exactly, did you lose?"

— the auditor

"Recovery" usually means

Get back to a good state. Restart clean. Roll back to the last thing that worked.

That instinct destroyed Knight Capital in the **last four minutes** of its forty-five.

They reached for good — and turned a one-machine fault into an eight-machine one.

Published in 1992. Twenty years early.

ARIES

Mohan · Haderle · Lindsay · Pirahesh · Schwarz — IBM Almaden

Foundational to how databases recover.

Source: ACM Transactions on Database Systems 17(1), 1992, pp. 94–162

"...the paradigm of **repeating history** to redo all missing updates before performing the rollbacks of the **loser transactions**."

— Mohan et al., ARIES, 1992

Three passes

Analysis	what was in flight when it died
Redo	repeat history — reconstruct the truth, including the damage
Undo	roll back the losers, deliberately, in order

First make it true.

Then make it right.

You cannot deliberately recover from a state you refuse to reconstruct.

Knight went straight for right, without going through true.
The gap between them cost $460 million.

You already run this

Postgres	write-ahead log — this exact algorithm, every crash
Kafka	the same idea, recovery boundary moved out to the consumer
Delta / Iceberg	transaction logs — ARIES's grandchildren in different clothes

You already trust this algorithm with **your data**.

The question is whether your **services** do it — or whether they restart and hope.

Lineage, not implementation: these inherit write-ahead logging as a technique, not ARIES as a spec.

Three mechanisms

1 — Write-ahead the commands, not just the rows.
A restart replays from the last checkpoint — and a replayed command is flagged as recovered, never passed off as fresh.

2 — Two ways to receive the same stream: live, and replayed from the archive.
Live is fast and misses whatever was sent while the consumer was down. Replay closes that gap — but **only within retention**, and **only if the consumer saved where it had got to**.

3 — Deduplication, which is what makes 2 safe.
Two paths means the same message can arrive twice. Sequence numbers on the way out; the consumer remembers what it has already processed, far enough back to cover the worst-case gap between the two paths — which is widest during the incident.

At-least-once, with effects you have made idempotent. Not exactly-once.

Put a number on it

RPO — if it dies right now, how much data do we lose?
RTO — how long until we're back?

We only say yes once **more than one machine** has the write on disk.
Lose a machine → lose nothing we've said yes to.
Lose the building, with every copy inside it → a different number.

> Not goals in a policy document.
> Outputs of the design.

The catch

Replay only works for things that are **safe to do twice**.

Recalculate a balance → replay it all day.
Send money to a bank → it already left the building.

Replaying a payment

is not recovery.

It's a second payment.

The rule

Not "log everything and replay it."

1. A **short, explicit list** of what's safe to replay
2. Everything that touches the outside world carries **the caller's own reference**
   — so a second copy is recognised and ignored

> Every one of these is a claim about the future.
> And in this talk, we don't believe claims.

PILLAR 6

Test it
or you don't have it.

"When did you last test this?"

— the auditor

The smallest chaos experiment
isn't a tool.
It's a sentence.

And the regulator is in the room

Art. 24 — resilience testing programme	every entity **other than a microenterprise** · at least yearly for the systems supporting a critical or important function
Art. 26 — threat-led penetration testing	entities **designated by their competent authority** · every 3 years · **on live production systems**

TLPT is intelligence-led red-teaming. The law doesn't mandate chaos engineering.

It mandates that you

stop saying you believe.

Source: Regulation (EU) 2022/2554 (DORA), Arts. 24–26 · TLPT detail in Commission Delegated Regulation (EU) 2025/1190

Back to where we started

```
SELECT COUNT(*) FROM audit_events
 WHERE timestamp >= NOW() - INTERVAL '4 hours';
 count
-------
     0
```

We had circuit breakers. We had graceful degradation.

We hadn't tested whether the system would degrade gracefully in places where graceful degradation was the wrong answer.

The kill switch I had never run

Meridian's emergency procedure, against paper trading.
No real money. Everything else identical.

> It closed the positions in the **wrong order**.

Source: Meridian engineering notes [31]

The steel-man

In a regulated firm, breaking production on purpose
is a conversation with your **risk team** before it's an engineering exercise.

So some of it happens in **staging**. And staging lies to you.

> A weak result still beats an assumption.

Friday afternoon
with a hypothesis and a kill switch
—or—
3am on a Saturday
with neither.

Six questions. One spine.

When should the system stop? → fail hard when silent corruption costs more than downtime.

Who is it answering to? → compliance is the architecture.

What evidence should it keep? → observability tiered by the latency of the path it watches.

What does it give up — and what never? → degradation matched to criticality, and one emergency path decided in calm and audited harder than the normal one.

How does it come back? → repeat history first, then unwind. True before right.

How do we know any of this works? → test the failure modes before they test you.

> What did the system already know to do, before the bad moment happened?

Shoulders of giants

Michael Nygard — Release It! (2007, 2018 2nd ed.)
Greg Young — CQRS / event sourcing
Martin Thompson — LMAX Disruptor, Aeron
Gil Tene — HdrHistogram, latency
C. Mohan et al. — ARIES, database recovery (1992)
Casey Rosenthal & Nora Jones — Chaos Engineering (2017)
Charity Majors — observability (2018, 2026)

> The patterns are old.
> What's new is the willingness to apply them strictly enough that they survive contact with a regulator.

Resilience isn't the absence of failure.
It's the presence of a plan.

The auditor stops asking
when the answers are
already there.

Thank you.



LinkedIn

in/lyubomir-bozhinov



GitHub

@lyubomir-bozhinov



Blog

barnett-studios.com