Software engineer: same job, only more interesting 🚀

Pierre Zemb, FinistDevs

La Nuit des Communautés Bretonne #3

17 September 2026 🌙

$ whoami 👋

- 🛠️ Staff engineer at Clever Cloud, working on **distributed systems**
  - 📟 Building, contributing, debugging, **getting paged**
- 🦀 Maintaining foundationdb-rs and a few other Rust libraries
- 🏸 Squash player
- 🤝 Co-organizing **FinistDevs**

FinistDevs

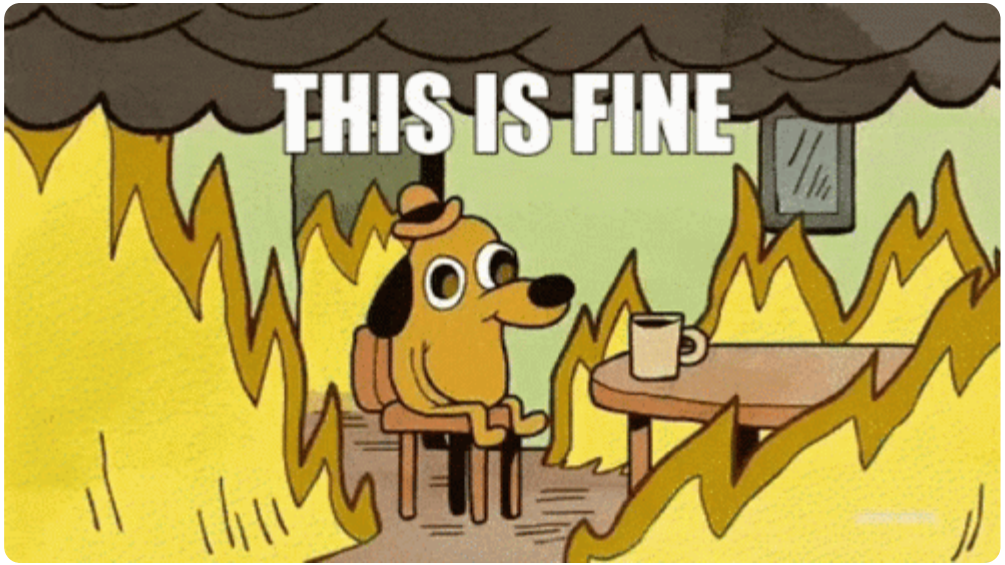
- 🎂 Born as **FinistJUG** in December 2011
- 🌙 **130+ evenings** since, on Java, cloud, DevOps, web, and whatever a member wants to share
- 👥 Three organizers: Horacio, Stéphanie, and me
- 🎤 **We are looking for speakers.** Never given a talk? Perfect, we help you prepare

📱 meetup.com/finistdevs
🔗 finistdevs.org

2010: I coded for the puzzle ☕

- 🎓 Engineering school in Brest: I came for electronics, I found **C**, and many other things
- ☕ I found out I could turn **coffee into software**, and I never stopped
- 🏗️ Part-time internships, always in **infrastructure teams**
  - 🌐 That is where I met **distributed systems**: hard, everywhere, so much to learn
  - 📖 The bible: *Designing Data-Intensive Applications*, Martin Kleppmann

2017: then I got a pager 📟



The world is worse than your tests 📚

You do not have to believe me (or your SRE colleague). Believe the literature:

You believe…	The research says otherwise
🌐 "The network is reliable"	Network-Partitioning Failures, OSDI '18
💾 "My data is safe on disk"	Data Corruption in the Storage Stack, FAST '08
💾 "fsync means it is saved"	Can Applications Recover from fsync Failures?, ATC '20
🤝 "Consensus recovers from crashes"	Protocol-Aware Recovery, FAST '18
🛡️ "3 replicas means I am safe"	Redundancy Does Not Imply Fault Tolerance, FAST '17
⚠️ "We handle all our errors"	Simple Testing Can Prevent Most Critical Failures, OSDI '14
🧵 "We have no concurrency bugs"	TaxDC, ASPLOS '16
🔄 "We just retry on failure"	Metastable Failures in the Wild, OSDI '22

What the pager taught me 📟

- 👀 Nothing teaches you a system like **watching it misbehave in production**
  - 🤔 How do you fix a system you do not understand?
- 💥 Software **will fail**: sometimes loudly, often **silently**
- 🤦 The failures that hurt are the ones **nobody imagined** while writing the code
- 🔭 You cannot fix what you cannot see: **observability first**

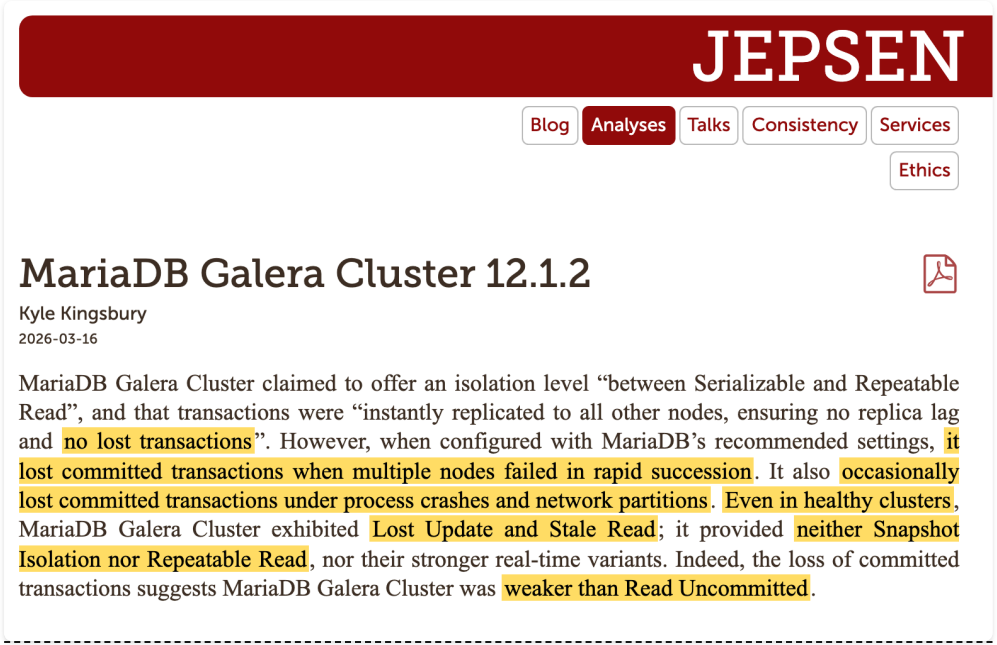
Développer des applications observables pour la production, Devoxx France

Down the rabbit hole of correctness 

- ✈️ Some software cannot afford the pager: aerospace, health, **distributed databases**, …
- 🛡️ So those teams built stronger harnesses: guardrails, fault injection, whole testing strategies
- 📚 Sixty years of techniques, refined in a niche: **types, property-based testing, fuzzing, model checking, deterministic simulation testing (DST), proofs**

Distributed systems live down that hole. **That is where I went.**

Jepsen breaks databases for a living 🔨



JEPSEN

Blog | Analyses | Talks | Consistency | Services | Ethics

MariaDB Galera Cluster 12.1.2

Kyle Kingsbury

2026-03-16

MariaDB Galera Cluster claimed to offer an isolation level "between Serializable and Repeatable Read", and that transactions were "instantly replicated to all other nodes, ensuring no replica lag and no lost transactions". However, when configured with MariaDB's recommended settings, it lost committed transactions when multiple nodes failed in rapid succession. It also occasionally lost committed transactions under process crashes and network partitions. Even in healthy clusters, MariaDB Galera Cluster exhibited Lost Update and Stale Read; it provided neither Snapshot Isolation nor Repeatable Read, nor their stronger real-time variants. Indeed, the loss of committed transactions suggests MariaDB Galera Cluster was weaker than Read Uncommitted.

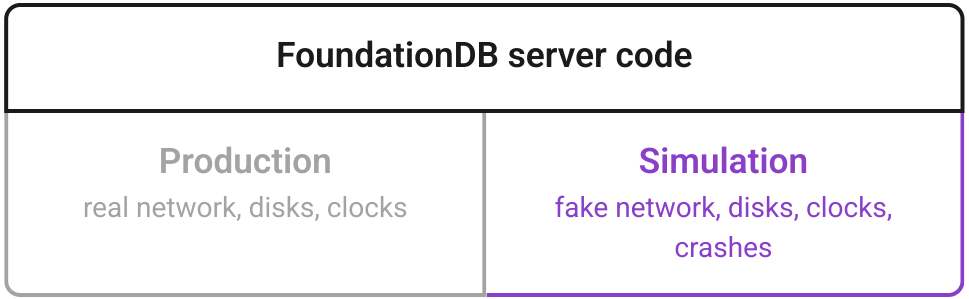
Since 2013: MongoDB, Redis, Cassandra, Kafka, etcd, CockroachDB... **almost none came out clean**

The one database Jepsen never bothered with 🍎

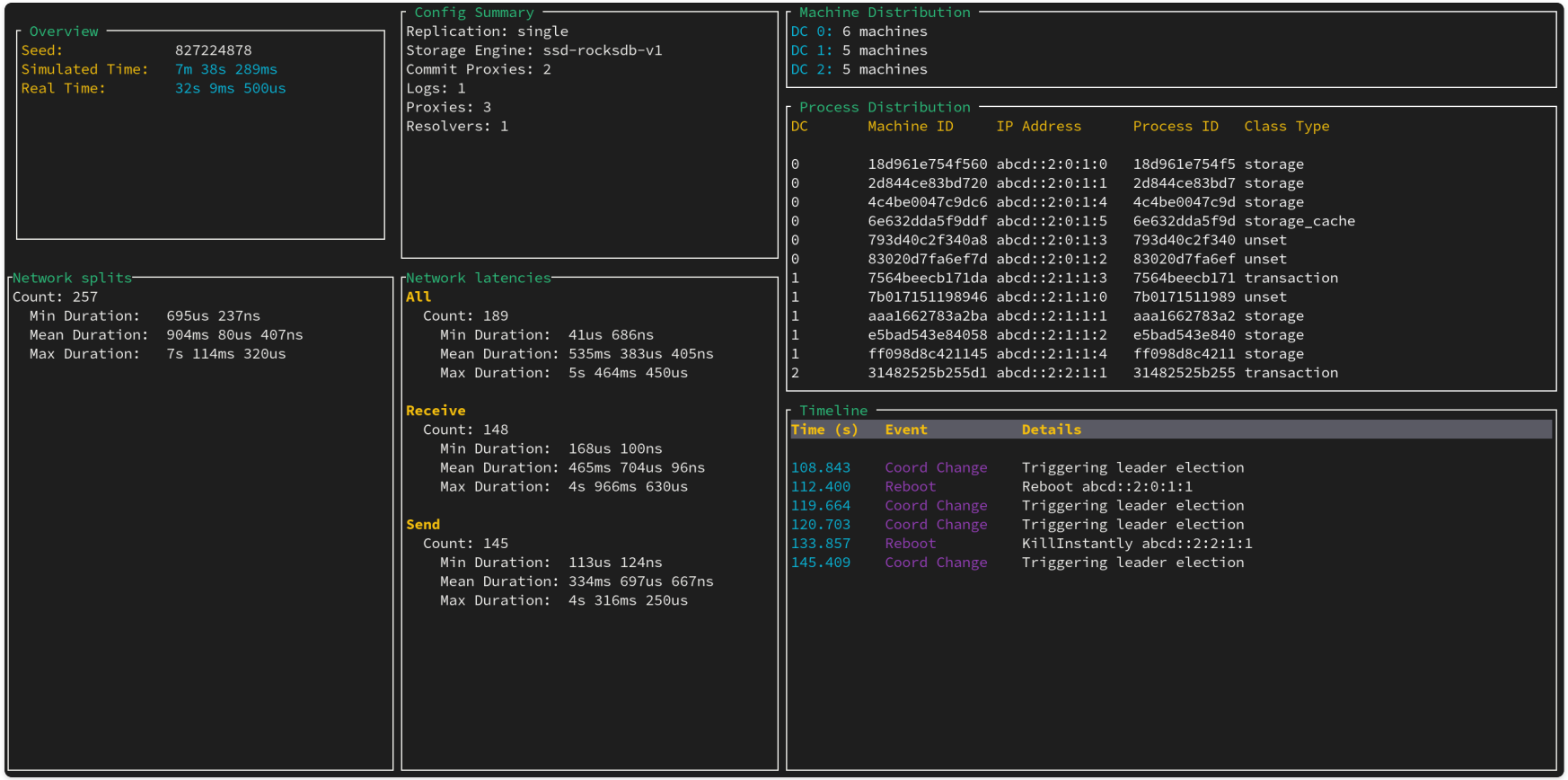
> "haven't tested foundation[DB] in part because their testing appears to be waaaay more rigorous than mine"

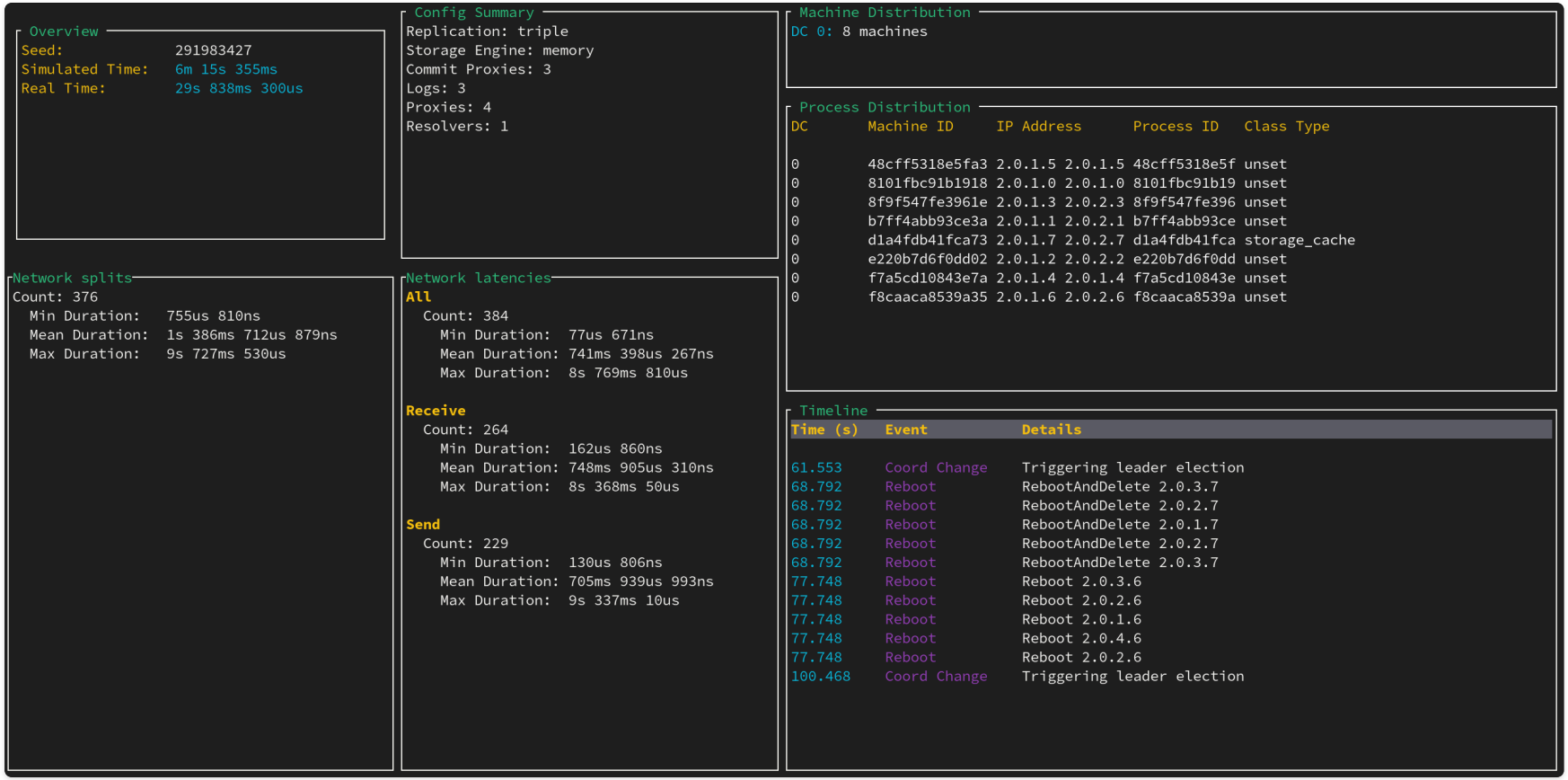
Kyle Kingsbury, aphyr, 2013

- 🎲 FoundationDB wrote its **simulator before the database**



- 🍎 **4,000 years** of simulated failures a year, at Apple
- 🧰 Years later, in production: **the sanest distributed system I have operated**

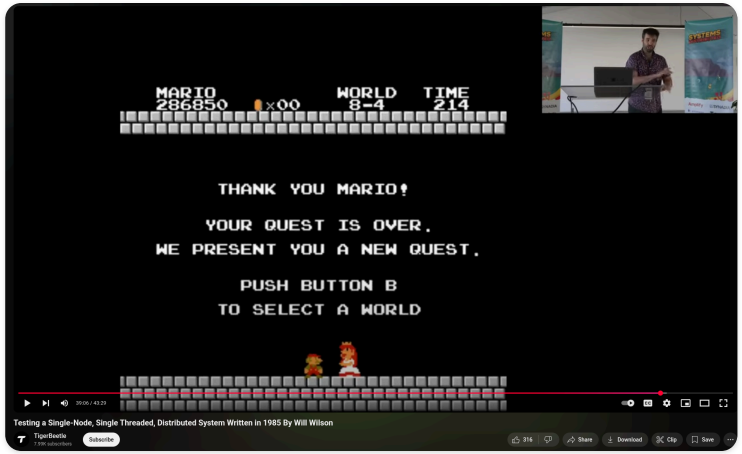

Overview
Seed: 827224878
Simulated Time: 7m 38s 289ms
Real Time: 32s 9ms 500us
Config Summary
Replication: single
Storage Engine: ssd-rocksdb-v1
Commit Proxies: 2
Logs: 1
Proxies: 3
Resolvers: 1
Machine Distribution
DC 0: 6 machines
DC 1: 5 machines
DC 2: 5 machines
Process Distribution
DC Machine ID IP Address Process ID Class Type
0 18d961e754f560 abcd::2:0:1:0 18d961e754f5 storage
0 2d844ce83bd720 abcd::2:0:1:1 2d844ce83bd7 storage
0 4c4be0047c9dc6 abcd::2:0:1:4 4c4be0047c9d storage
0 6e632dda5f9ddf abcd::2:0:1:5 6e632dda5f9d storage_cache
0 793d40c2f340a8 abcd::2:0:1:3 793d40c2f340 unset
0 83020d7fa6ef7d abcd::2:0:1:2 83020d7fa6ef unset
1 7564beecb171da abcd::2:1:1:3 7564beecb171 transaction
1 7b017151198946 abcd::2:1:1:0 7b0171511989 unset
1 aaa1662783a2ba abcd::2:1:1:1 aaa1662783a2 storage
1 e5bad543e84058 abcd::2:1:1:2 e5bad543e840 storage
1 ff098d8c421145 abcd::2:1:1:4 ff098d8c4211 storage
2 31482525b255d1 abcd::2:2:1:1 31482525b255 transaction
Network splits
Count: 257
Min Duration: 695us 237ns
Mean Duration: 904ms 80us 407ns
Max Duration: 7s 114ms 320us
Network latencies
All
Count: 189
Min Duration: 41us 686ns
Mean Duration: 535ms 383us 405ns
Max Duration: 5s 464ms 450us
Receive
Count: 148
Min Duration: 168us 100ns
Mean Duration: 465ms 704us 96ns
Max Duration: 4s 966ms 630us
Send
Count: 145
Min Duration: 113us 124ns
Mean Duration: 334ms 697us 667ns
Max Duration: 4s 316ms 250us
Timeline
Time (s) Event Details
108.843 Coord Change Triggering leader election
112.400 Reboot Reboot abcd::2:0:1:1
119.664 Coord Change Triggering leader election
120.703 Coord Change Triggering leader election
133.857 Reboot KillInstantly abcd::2:2:1:1
145.409 Coord Change Triggering leader election


Overview
Seed: 291983427
Simulated Time: 6m 15s 355ms
Real Time: 29s 838ms 300us
Config Summary
Replication: triple
Storage Engine: memory
Commit Proxies: 3
Logs: 3
Proxies: 4
Resolvers: 1
Machine Distribution
DC 0: 8 machines
Process Distribution
DC Machine ID IP Address Process ID Class Type
0 48cff5318e5fa3 2.0.1.5 2.0.1.5 48cff5318e5f unset
0 8101fbc91b1918 2.0.1.0 2.0.1.0 8101fbc91b19 unset
0 8f9f547fe3961e 2.0.1.3 2.0.2.3 8f9f547fe396 unset
0 b7ff4abb93ce3a 2.0.1.1 2.0.2.1 b7ff4abb93ce unset
0 d1a4fdb41fca73 2.0.1.7 2.0.2.7 d1a4fdb41fca storage_cache
0 e220b7d6f0dd02 2.0.1.2 2.0.2.2 e220b7d6f0dd unset
0 f7a5cd10843e7a 2.0.1.4 2.0.1.4 f7a5cd10843e unset
0 f8caaca8539a35 2.0.1.6 2.0.2.6 f8caaca8539a unset
Network splits
Count: 376
Min Duration: 755us 810ns
Mean Duration: 1s 386ms 712us 879ns
Max Duration: 9s 727ms 530us
Network latencies
All
Count: 384
Min Duration: 77us 671ns
Mean Duration: 741ms 398us 267ns
Max Duration: 8s 769ms 810us
Receive
Count: 264
Min Duration: 162us 860ns
Mean Duration: 748ms 905us 310ns
Max Duration: 8s 368ms 50us
Send
Count: 229
Min Duration: 130us 806ns
Mean Duration: 705ms 939us 993ns
Max Duration: 9s 337ms 10us
Timeline
Time (s) Event Details
61.553 Coord Change Triggering leader election
68.792 Reboot RebootAndDelete 2.0.3.7
68.792 Reboot RebootAndDelete 2.0.2.7
68.792 Reboot RebootAndDelete 2.0.1.7
68.792 Reboot RebootAndDelete 2.0.2.7
68.792 Reboot RebootAndDelete 2.0.3.7
77.748 Reboot Reboot 2.0.3.6
77.748 Reboot Reboot 2.0.2.6
77.748 Reboot Reboot 2.0.1.6
77.748 Reboot Reboot 2.0.4.6
77.748 Reboot Reboot 2.0.2.6
100.468 Coord Change Triggering leader election

DST for building distributed systems 📺

Et si on faisait du simulation-driven development ?, Devoxx France

Yes, DST can even beat Mario 🎮



- 🎲 Antithesis beat Super Mario Bros with **random inputs only**. No human, no script
- 🎯 Point that machine at your software and it finds **what you never thought to test**

Testing a Single-Node, Single Threaded, Distributed System Written in 1985, Will Wilson

2024: what was I actually paid for? 🤔

- ☕ Turning coffee into software: **less of the week than you would think**
- 📖 Reading **other people's code**, papers and logs
- 📟 Getting paged, then finding out **what I had missed**
- 🧗 The hard part: **understanding the system** well enough to change it

The job was never typing. It was **finding out what I missed before the users did.**

Then the LLMs showed up 🤖

Before LLMs, we trusted code because we:

- 🧠 understood it
- ✍️ wrote it
- 🧪 tried it
- 📟 got paged by it

AI broke the first three. The only feedback left is the **worst** one: the pager.

"We no longer pay developers to write code" 📺

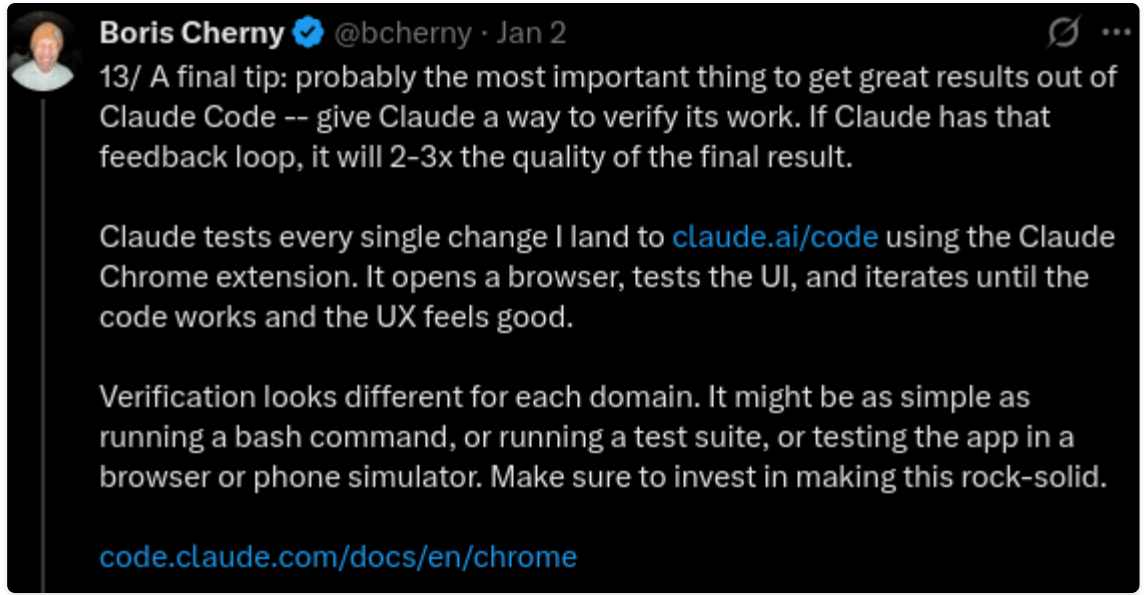
That is **my own CEO**, Quentin Adam.

On ne paie plus les développeurs pour écrire du code, Underscore_

Same model, not the same code 😬

- 🏢 Clever Cloud: **~100 employees, 70 developers**
- 📉 The quality of the generated code varied **wildly** from one project to the next
- ✅ Three kinds of projects where AI **just worked**:
  - 🦀 written in **Rust**
  - 🧪 with **lots of tests**
  - 🎲 or **stronger tests**: property-based testing, **DST**

DST: the ultimate LLM feedback loop 🤖🔁



● Perfect! Now I can see exactly what's happening. The logs show that:

1. **4 out of 5 seeds pass** (Success Rate: 80.00%)
2. **Only seed 9495001370864752853 fails**
3. The report shows **"Faulty seeds: [9495001370864752853]"** - great!

Now I need to run just the failing seed to see the specific ordering error. Let me modify the test to run only the problematic seed and get more detailed logs:

Update(moonpool-simulation/tests/ping_pong_tests.rs)
└ Updated **moonpool-simulation/tests/ping_pong_tests.rs** with **1** addition and **1** removal

```
30                .register_workload("ping_pong_server", ping_pong_server)
31                .register_workload("ping_pong_client", ping_pong_client)
32                .set_iteration_control(IterationControl::FixedCount(iteration_count))
33 -              .set_debug_seeds(vec![42, 9495001370864752853, 123456]) // Test multiple seeds
33 +              .set_debug_seeds(vec![9495001370864752853]) // Focus on the failing seed
34                .run()
35                .await;
36
```

🤖 LLM writes code → 🧪 Simulation finds bug → 🔍 LLM reads failing seed → 🔧 LLM fixes code 🔁

So, what is changing? ⏳

Few of us ever had time for **software quality**. Now that code costs nothing, **the time is back**.

- 🎯 I am the **architect, not the typist**
- 🔁 I give the agent a **fast feedback loop**
- 🛡️ Tests, simulation, property-based testing: **the correctness work now outweighs the code**
- 📜 Contracts, specifications and invariants are **the capital**

```
# Gherkin: requirements the agent can run
Scenario: a guest cannot pay with a saved card
  Given a guest user with a saved card
  When they check out with that card
  Then the payment is refused
```

I used to craft everything by hand 🏭

We can automate! 🏸

Do we automate all code? 🤔

No. What is left: specs, success criteria, feedback loops, QA, code review.

Simon Willison, Vibe engineering, 2025

- 🔪 Code like a surgeon: the agent preps, **I keep the scalpel**

Same job, only more interesting 🚀

- ☕ I still turn coffee into software, **a bit faster, with way more quality**
- 📖 Reading code and papers has **never been easier**
- 📟 I still get paged, but first I **torture software in conditions worse than production**
- 🧗 **The hard part is all that is left**
- 🔥 More **fun** than ever: never about writing code, always about **building** and **learning**

Don't fall into the anti-AI hype, antirez

Make correctness your goal 🎯

- 📏 Production code written by an agent needs a **higher bar** than a human's
- 🎲 DST pays off when you **rebuild from scratch**, like we did at Clever Cloud
  - 🧰 The rest of the toolbox is **less invasive**
- 📈 Every technique was niche. **It is now the standard for anyone running agents**
  - 🔮 New tools will appear!
- 🧭 While everyone is talking about speed, **talk about correctness and quality**

Thank you! 🙏

any questions?

🤝 Come talk to us at **FinistDevs**

🔗 Slides and stories on pierrezemb.fr

🌊 moonpool, a simulator for distributed systems in Rust

🏝️ paros, Paxos built inside it, 100% by agents