What if we embraced simulation-driven development? 🎲

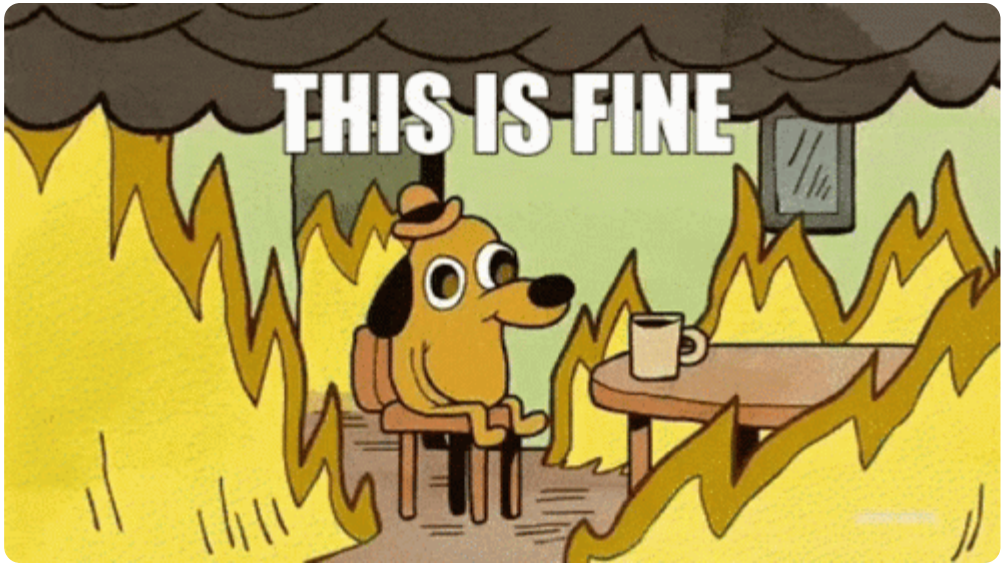
Pierre Zemb, Staff Engineer @ Clever Cloud

Sunny Tech 2026 🦩

$ whoami 👋

- 🛠️ Staff engineer, working around **distributed systems**
  - 🐞 Building, contributing, debugging, **paging**…
- 🦀 Maintaining several OSS libraries in Rust (foundationdb-rs)
- 🤝 Involved in local communities (**FinistDevs** / JUG)
- 🎾 Squash player

Also me 🔥



One of my most "wait what" bugs 🤯

- 🌐 A tough network partition hit a **70+ node Apache Hadoop cluster**
- 🐢 The cluster could not restart
- ☕ A `NullPointerException` **at startup**, caused by its faulty state
- 🩹 Patched in a newer HDFS: we **backported** the patch and redeployed the jar 😱
- ⏰ We hit it at the **worst moment**, during recovery
- 🔥 Rolling out an untested patch on a distributed system under fire is an **unpleasant** experience

So, what went wrong? 🔍

- 🚗 **Production environments ≠ Development environments**
  - 📝 Like the knowledge test vs the driving test
- 🌍 The code didn’t account for real-world complexity, like our "not-common" NPE
  - 👤 Users will do weird things
  - 🔧 Systems will be under pressure
  - 💥 And things will **break** if we’re not ready

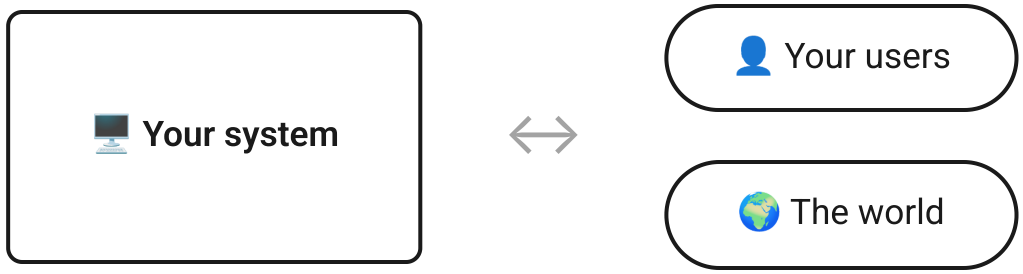
It’s 2026, and AI broke "good enough" 🤖

Before LLMs, we trusted **correctness** because we:

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

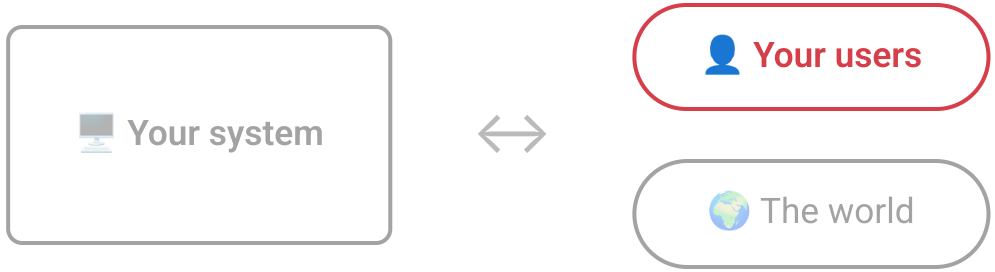
AI broke the first three. The only feedback left is the **worst** one: the pager.

Your code meets two things 🌍



Two sources of chaos. Let’s start with the **users**.

Can we test our users? 🛒



One checkout, so many paths 🛒

Dimension	Options	Count
👤 User type	Guest, Logged-in, Premium	3
💳 Payment	Card, PayPal, Apple Pay, Gift Card	4
🚚 Delivery	Standard, Express, Pickup	3
🏷️ Promotion	Yes, No	2
📦 Inventory	In stock, Low, Out	3
💱 Currency	USD, EUR, GBP	3

`3 x 4 x 3 x 2 x 3 x 3 = 648 happy-path test cases` 😱

Add a few features… 😱

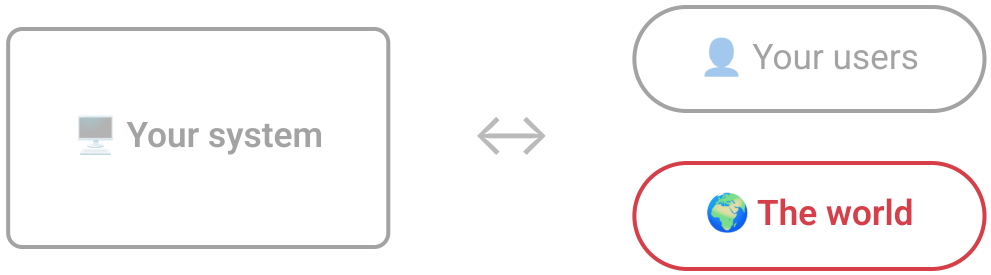
Dimension	Options	Count
👤 User type	Guest, Logged-in, Premium, **Business**	4
💳 Payment	Card, PayPal, Apple Pay, Gift Card, **Bank Transfer**	5
🚚 Delivery	Standard, Express, Pickup, **Same-Day**	4
🏷️ Promotion	Yes, No, **Expired promo**	3
📦 Inventory	In stock, Low, Out, **Preorder**	4
💱 Currency	USD, EUR, GBP, **JPY**	4

`4 x 5 x 4 x 3 x 4 x 4 = 3,840 test cases` 💀

You can’t test what you don’t know 🙈

- 🧠 Example-based tests only cover the cases you **already imagined**
- 🪤 The nasty bugs hide in the **combination you never tried**
- 🛡️ Tests are a **regression net, not a proof** that bugs are absent

Can our code handle the world? 🌍



Now the other side: **the world** 😈

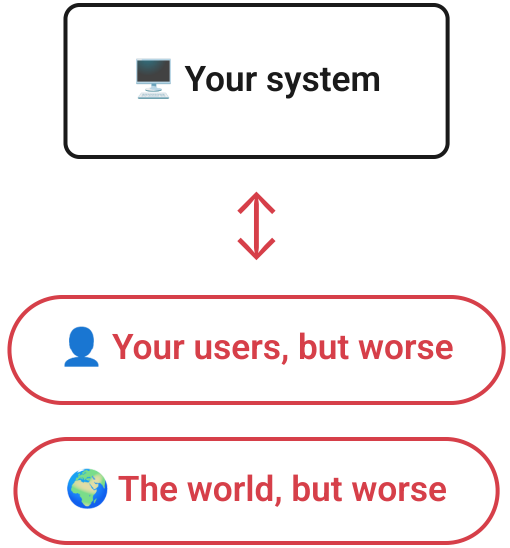
The world is worse than your tests 📚

You do not have to believe me. 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
📖 "Our documentation must be right"	Jepsen: MariaDB Galera

Let’s test the worst 😈

- ✅ To ensure software **behaves correctly**
- 🔍 To surface **hidden edge cases**
- 🕳️ To uncover the **unknown unknowns**



Two ways to be the worst 😈

👤 The worst users

↓

🎰 Random workloads

🌍 The worst world

↓

🎭 Fakes

Don’t write tests, write a generator 🎰

```
enum UserType { GUEST, LOGGED_IN, PREMIUM, BUSINESS }
enum PaymentMethod { CARD, PAYPAL, APPLE_PAY,
                     GIFT_CARD, BANK_TRANSFER }

Random rand = new Random(seed);

UserType user = pickRandom(rand, UserType.values());
PaymentMethod payment = pickRandom(rand, PaymentMethod.values());
// add a feature? add an enum value, not 648 tests
```

Iterate enough seeds and you **cover the cardinality**, for free.

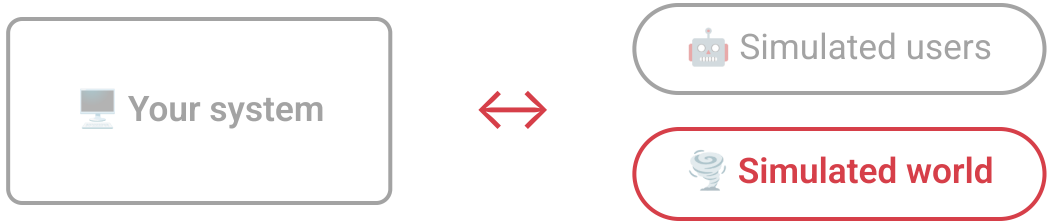
Don’t write assertions, write properties ✅

```
// from a hardcoded test case:
assertFalse(new User(GUEST).canUse(SAVED_CARD));

// to a property that holds for ANY generated input:
assertEquals(user.isAuthenticated(), user.canUse(SAVED_CARD));
```

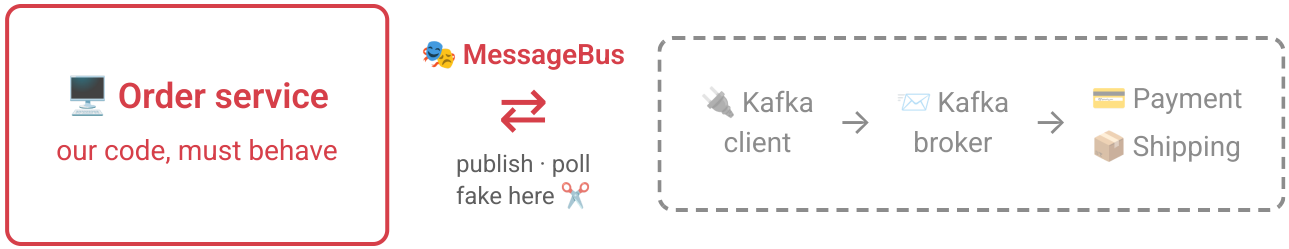
- A property reads like a **spec**, not a test case. It holds for **any** generated input
  - (Yes it’s like Property-based testing, but applied to integration tests)

The worst world → fakes 🌪️



- 💥 The world breaks time, network, disk, infra, dependencies: to test the worst, you must **control all of it**
- 🎭 The lightest way in: a **fake**, an in-memory stand-in you can make as cruel as you want

An app on a bus we don’t own 📨



We own the service. We don’t control the **driver** or the **broker** past it. The seam between them is where our **fake** lives, an autonomous in-memory stand-in for the bus.

A fake bus is just a list of messages 🎭

```
interface MessageBus {
    Future<Void> publish(String topic, Message msg);
    List<Message> poll(String topic);
}

// production: talks to Kafka
class KafkaBus implements MessageBus { /* driver */ }

// simulation: a fake bus is just a list of messages
// held by a singleton
class FakeBus implements MessageBus {
    Map<String, List<Message>> queues = new HashMap<>();
}
```

Same interface, two implementations: your code **can't tell the difference**. Keep the fake honest with one **contract test** run against both.

Make the fake fight back 😈

```
class FakeBus implements MessageBus {
    Map<String, List<Message>> queues = new HashMap<>();
    Random rng;

    public Future<Void> publish(String topic, Message msg) {
        sleep(rng.nextInt(500));                          // random lag
        float r = rng.nextFloat();
        if (r < 0.15) return failed(new ConnectionLost());  // never happened
        queues.get(topic).add(msg);                       // committed …
        if (r < 0.30) return failed(new ConnectionLost());  //  … ack lost!
        return completed();
    }
}
```

Your system should still behave correctly!

Be worse than production 😈

- 🩺 Jepsen's MariaDB Galera Cluster 12.1.2 report

The doc says:

> "Data is consistent across all nodes at all times"

The audit reveals:

> Even in healthy clusters, MariaDB Galera Cluster exhibited Lost Update and Stale Reads

- 😈 So your fake should be **worse**: 50% stale reads, not 0.1%
- 🛡️ Survive the fake, and you **survive production**

Bundle it: that's DST 🎲

Drive both from one **seed** on a deterministic loop, and every failure **replays exactly**.

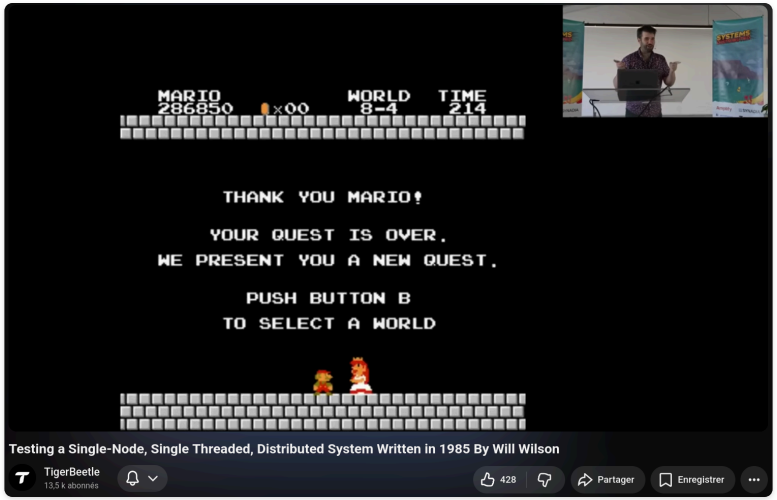
That bundle is **Deterministic Simulation Testing**. The whole point is to **discover what you don't know**.

Who actually does this? 🧬

- 🧬 A growing club, mostly **distributed databases**:
  - Clever Cloud, Antithesis, TigerBeetle
  - Turso, RisingWave, Dropbox
  - sled, Kafka KRaft, AWS

How far can DST go? 🍄

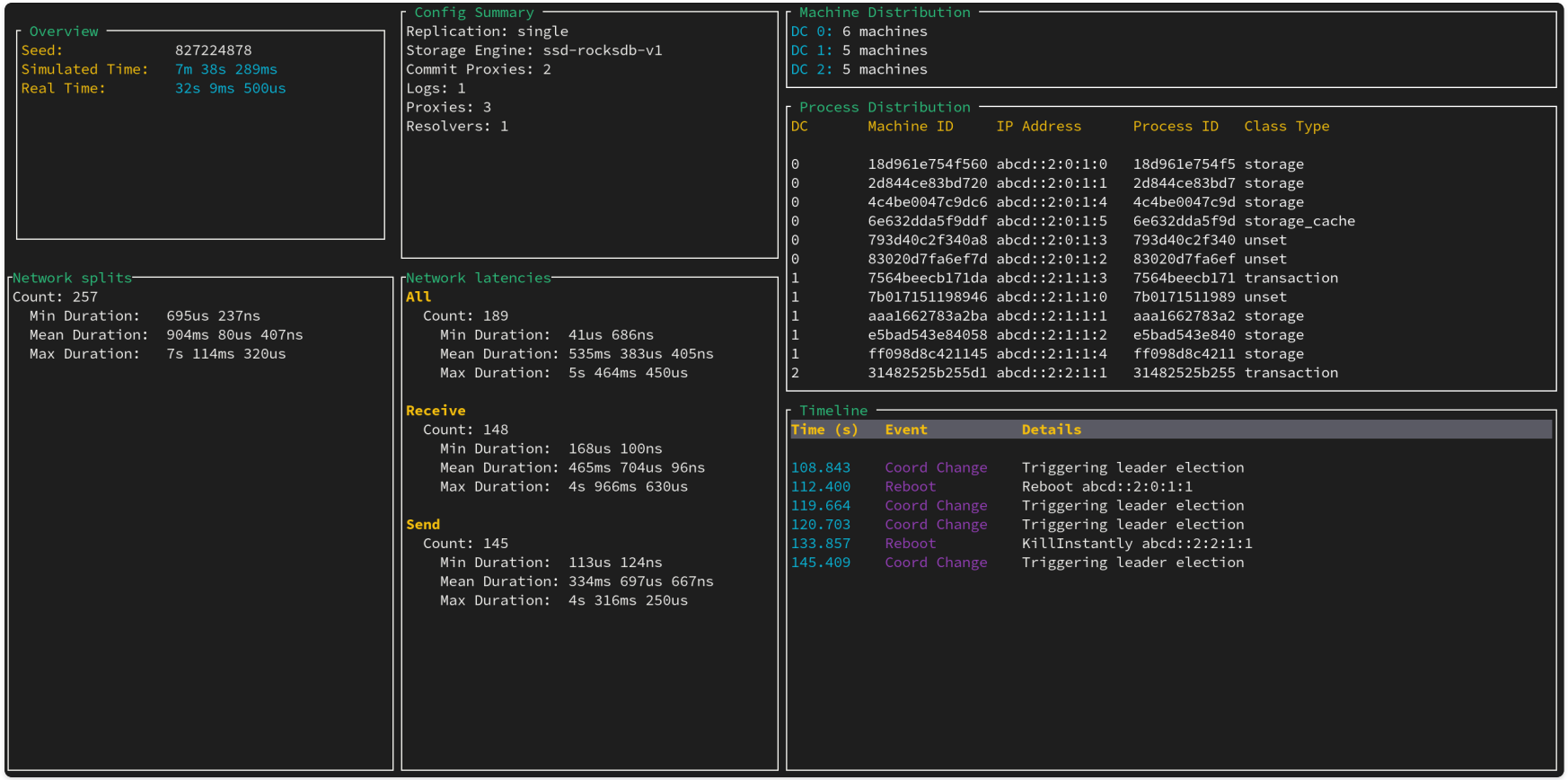
- 🤖 **Antithesis** does autonomous testing: it hunts bugs for you
- 🍄 From pure random input, it **beats Super Mario Bros (1985)**
- 🧱 …and finds a bug where **Mario clips through a wall**

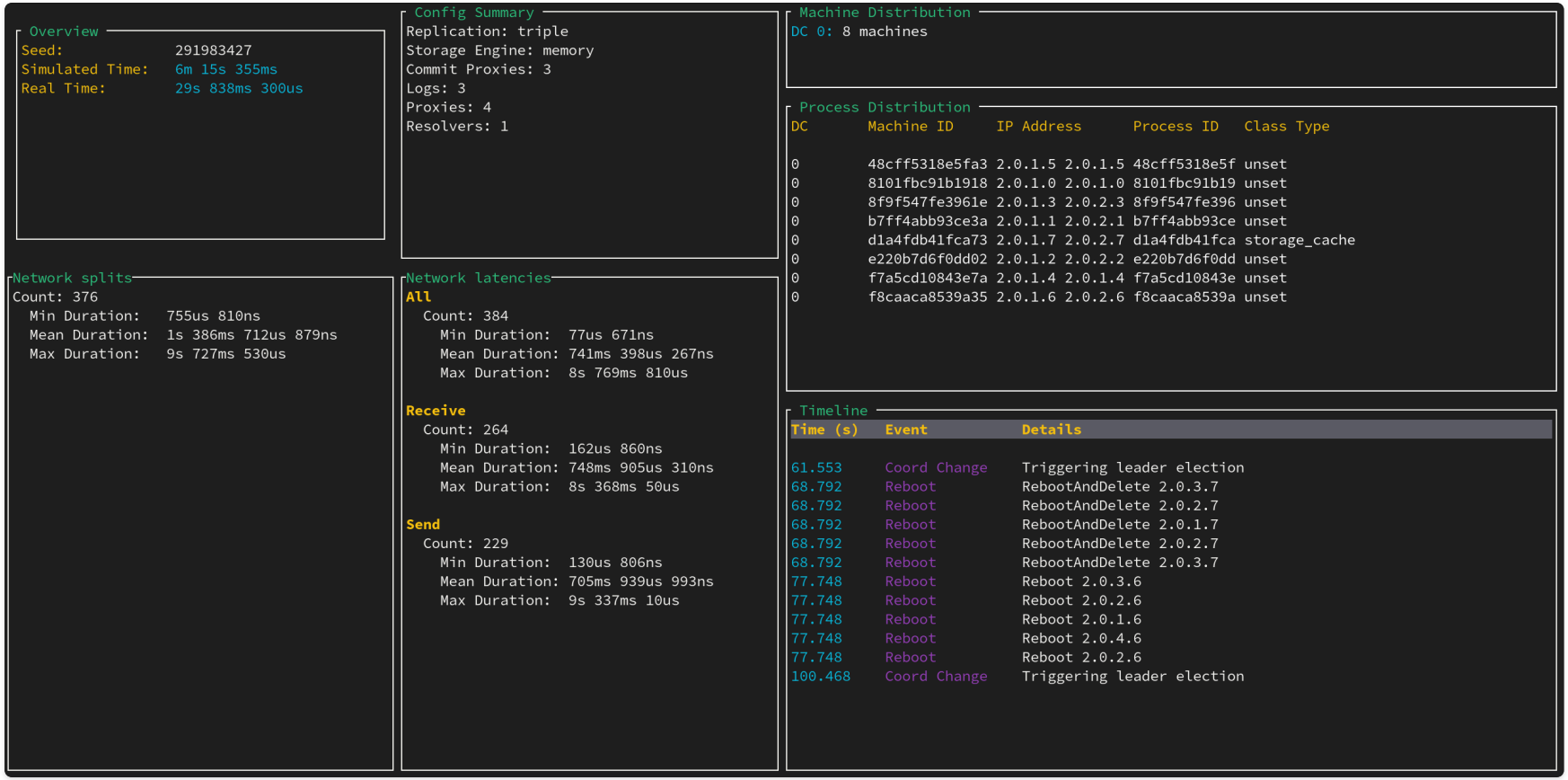


Testing a Single-Node, Single Threaded, Distributed System Written in 1985 — *Will Wilson*

So we're building a database 🗄️

- 🏢 At **Clever Cloud** we build **Materia**, a multi-tenant, multi-model distributed database
  - KV, ETCD, KMS, workflows, leader election…
- 📽️ Built on FoundationDB, which has a simulation framework that we hacked:
  - Borrowing FDB's simulator, BugBash 2026
  - Distributed DBs with FDB & Rust, Sunny Tech 2024
  - FoundationDB intro, Sunny Tech 2023


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

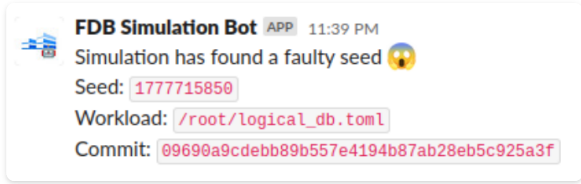

Overview
Seed: 291983427
Simulated Time: 6m 15s 355ms
Real Time: 29s 838ms 300us
Config Summary
Replication: triple
Storage Engine: memory
Commit Proxies: 3
Logs: 3
Proxies: 4
Resolvers: 1
Machine Distribution
DC 0: 8 machines
Process Distribution
DC Machine ID IP Address Process ID Class Type
0 48cff5318e5fa3 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

It found bugs everywhere 🔥

- 📊 query execution returning **wrong data**
- 🗺️ query planner picking the **wrong index**
- 💥 **corruption** during reindexing
- 👑 **dual leader election** under clock skew
- 🗜️ ETCD compaction **deleting live data**
- …

Simulation-driven development 🔁

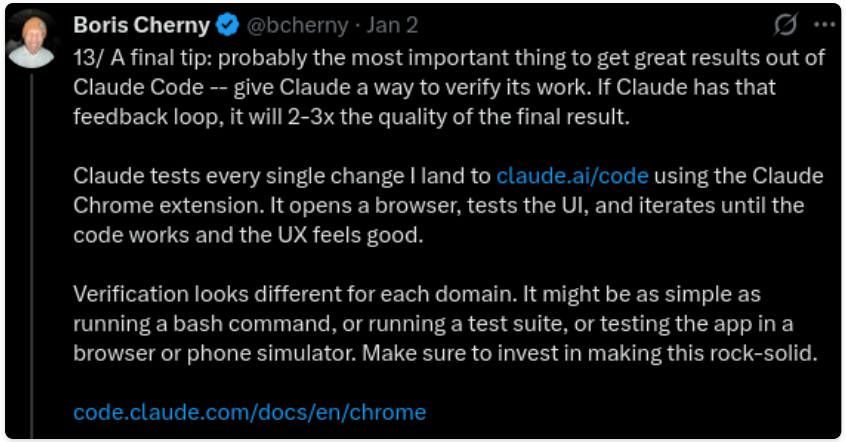
- 💡 Simulation surfaces what you **don't know to look for**
- 🏗️ Now every layer we build is **simulation-first**: we default to **discovery**
  - 🔁 Run it in a **loop**: in CI, and on a cloud fleet
  - ☁️ **30 min of sim is about 24h** of chaos testing
  - 🐞 A faulty seed **replays locally**, deterministically
- 🦸 It feels like super-powers, because we **trust our software**

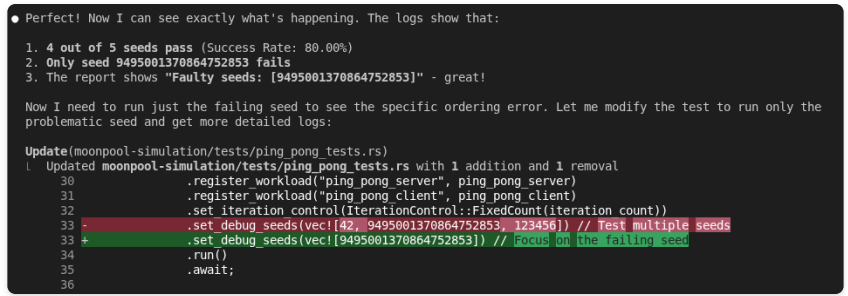


Not a silver bullet 🙅

- 📈 **Performance is invisible** in sim: you still need a perf farm
- 🕳️ **Verified fakes**: you must ensure your fake matches the real thing
- ⏳ **Bug-finding has latency**: a bug can hide in a seed for months

You don’t trust Claude, you trust the simulator 🔁

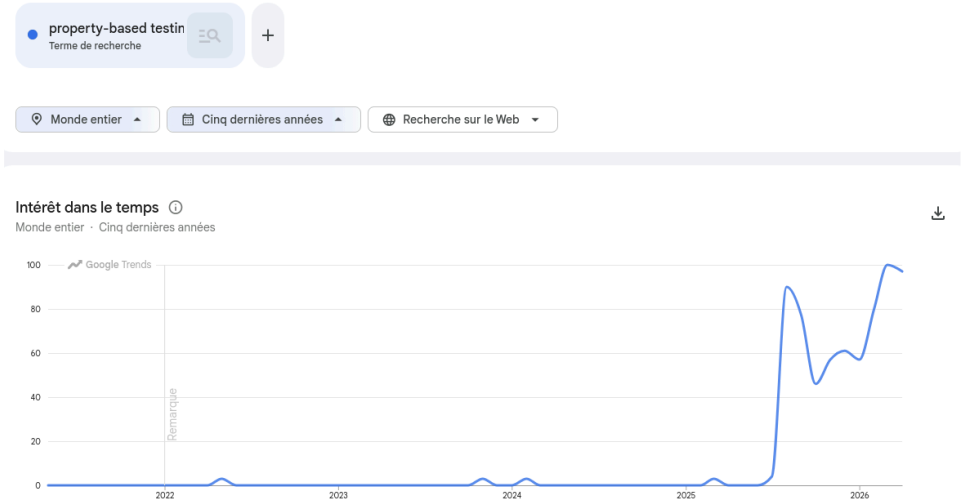




The age of correctness has started 🔬

Expressing correctness was a **niche craft** for decades: types, contracts, property-based testing, fuzzing, formal methods, simulation.

AI broke "good enough", so **all of them are about to grow**. Property-based testing already is:



Invest in correctness, now 🌅

- 🔭 Simulation is **one tool** in that rising tide, pick whichever fits you
- 🛡️ We have a **duty**: ship correct software, not crap
- ⏰ Don’t wait for the 3am page, **invest in correctness today**

How to adopt DST 📈

Start anywhere. Each level adds value.

Level	What to do	What you get
1 🎰	Random workload generation	Test unusual combinations
2 ✅	Property-based testing	Flush out your system spec
3 🎭	Fakes	Fast, deterministic tests
4 😈	Fault-injectable fakes	Discover edge cases
5 🎲	Seed-driven DST	Reproducible bugs, autonomous discovery

Thank you! 🙏

any questions?

📝 Rate this talk

🎈 Come say hi at the **Clever Cloud booth**

🔗 Everything is on pierrezemb.fr

🦀 My own DST in Rust