taiko

High-Assurance ZK

Building foundations you can trust

Mamy Ratsimbazafy

ZK Engineering @ Taiko

mratsim

m_ratsim

A quick technical presentation of Taiko

- **Type 1 zkEVM (Ethereum-equivalent)** → no changes to hashes, state trees, transaction trees, precompiles or any other in-consensus logic. Full implementation of the Ethereum (execution layer) yellow paper specifications
- **Based rollup** → block sequencing driven by L1 validators, thereby inheriting Ethereum's level of decentralization
- **Permissionless** → proposers & provers can join or leave the network at any time, thereby maximizing censorship resistance
- **Deterministic block execution** → finality is achieved immediately after a block is proposed, i.e. all block properties are immutable from that point on. Thereby Taiko L2 transactions are finalized after only a single L1 confirmation

High-Assurance software

High-Assurance?

ISO 15408 - Common Criteria for Information Technology Security Evaluation (1999)

EAL: Evaluation Assurance Level

- EAL1: Functionally Tested
- EAL2: Structurally Tested
- EAL3: Methodically Tested and Checked
- EAL4: Methodically Designed, Tested and Reviewed
- EAL5: Semiformally Designed and Tested
- EAL6: Semiformally Verified Design and Tested
- EAL7: Formally Verified Design and Tested

High-Assurance?

Software that you can trust to build
a multi-billion dollars ecosystem on

Software failures

What do Ariane 5, Boeing 787 and Gangnam Style have in common?

https://www.bbc.com/future/article/20150505-the-numbers-that-lead-to-disaster

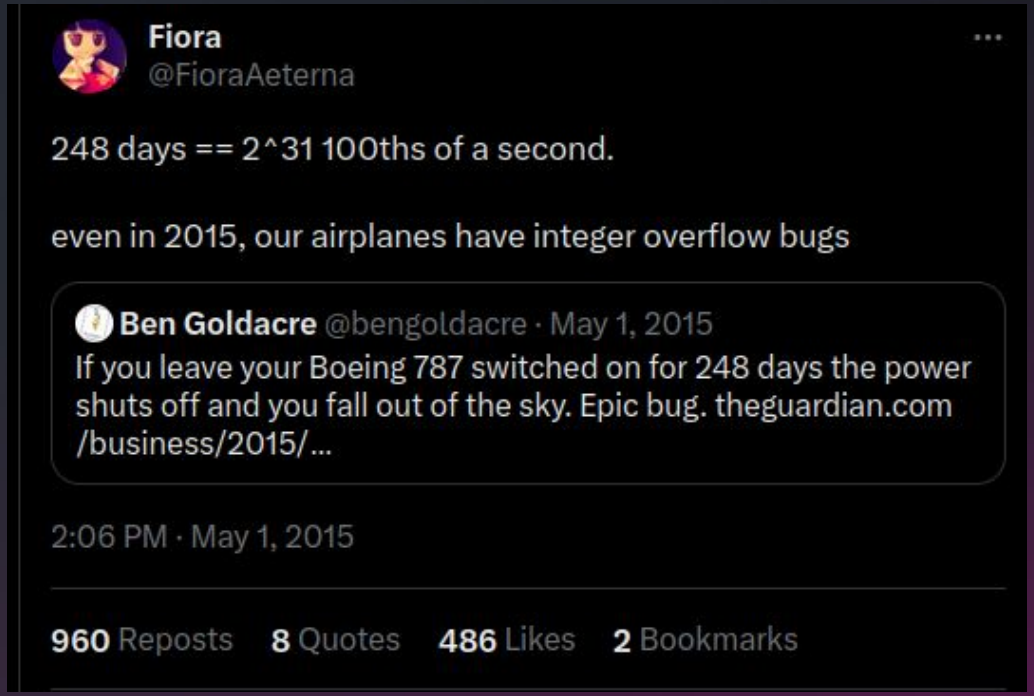
What do Ariane 5, Boeing 787 and Gangnam Style have in common?

Ariane 5: https://www-users.cse.umn.edu/~arnold/disasters/ariane5rep.html

“The internal SRI [Inertial Reference System] software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value.”

What do Ariane 5, Boeing 787 and Gangnam Style have in common?

Boeing 787



What do Ariane 5, Boeing 787 and Gangnam Style have in common?

Psy - Gangnam Style

In 2014, Gangnam Style exceeded Youtube view limits of 2,147,483,647

What do Ariane 5, Boeing 787 and Gangnam Style have in common?

Psy - Gangnam Style

In 2014, Gangnam Style exceeded Youtube view limits of 2,147,483,647

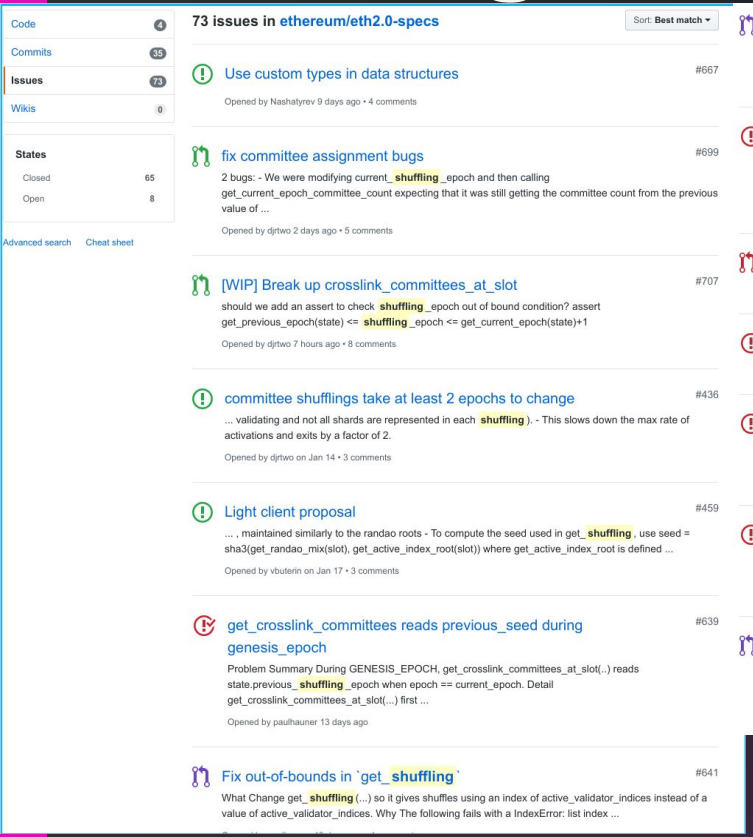
Radiation therapy gone wrong

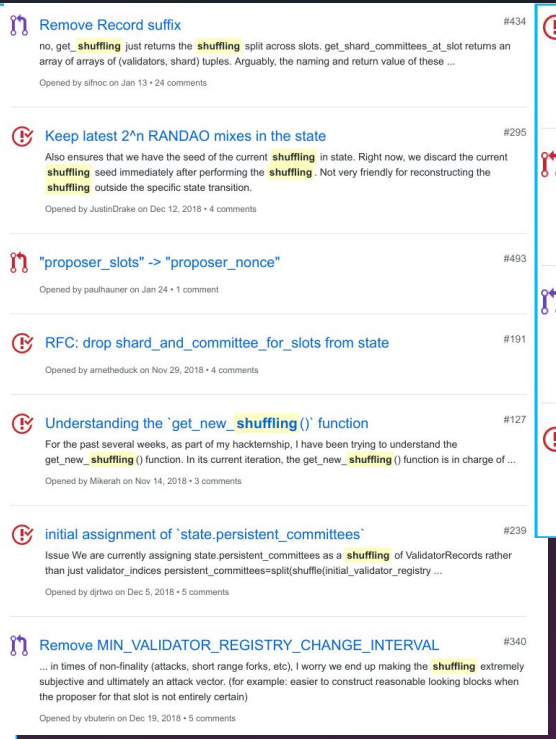
The Therac-25:
https://hackaday.com/2015/10/26/killed-by-a-machine-the-therac-25/

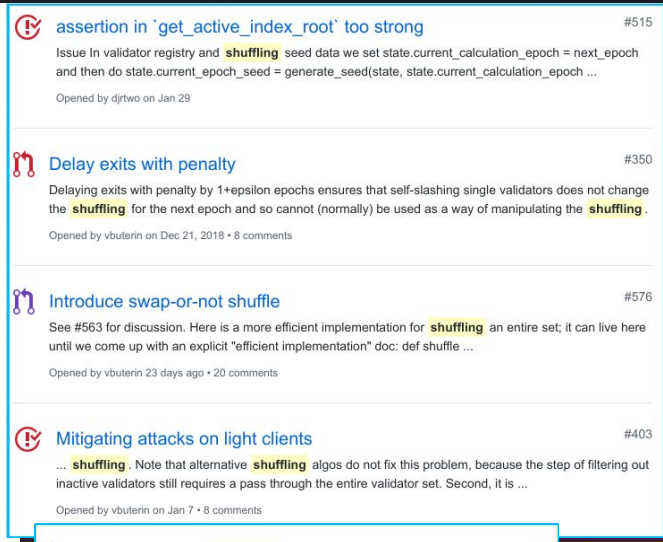
Between 1983 and 1987, 6 patients died from radiation overdose.

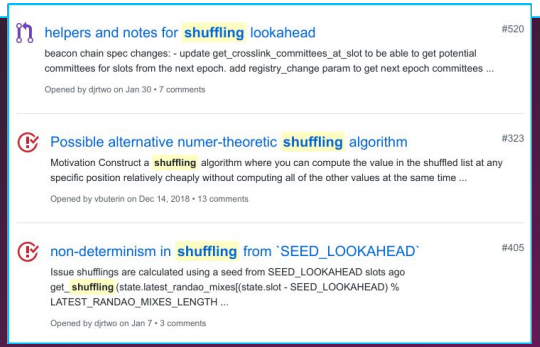
“With the technician running the machine, the two were able to pinpoint the issue. The VT-100 console used to enter Therac-25 prescriptions allowed cursor movement via cursor up and down keys. If the user selected X-ray mode, the machine would begin setting up the machine for high-powered X-rays. This process took about 8 seconds. If the user switched to Electron mode within those 8 seconds, the turntable would not switch over to the correct position, leaving the turntable in an unknown state.”

Shuffling Ethereum validators









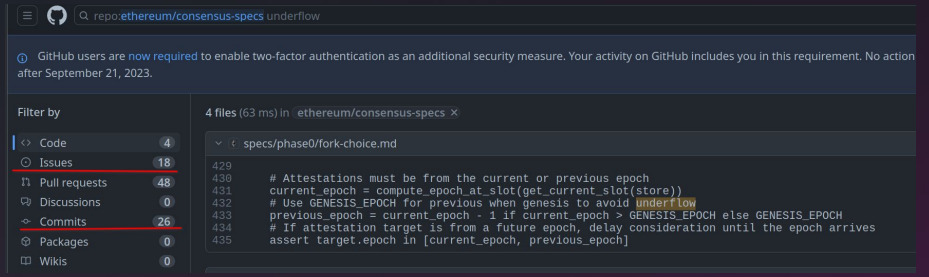
What could go wrong?

A Ethereum 2 spec bug that plagued us, enforcing the invariant:

assert x <= y - z

Often x and y are epoch, x represents a justified, finalized or RANDAO epoch, y is the current epoch. z is often 1.

And on Genesis (epoch 0) the assertion which should have caught a bug (signed 0 <= -1) doesn't
because of unsigned underflow (0 <= 2^64)



What could go wrong?

We have seen:

- Boundary / edge conditions (overflows and underflows)
- State machine bugs
- Design bugs

There are many more classes of bugs
And smart-contracts introduce even more (https://rekt.news/)

The stakes

What’s at stake?

- Your reputation
- Your (hopefully) billion-dollar TVL
- Your future revenue (from fees)

...

Your sleep

Catching the bugs

Testing

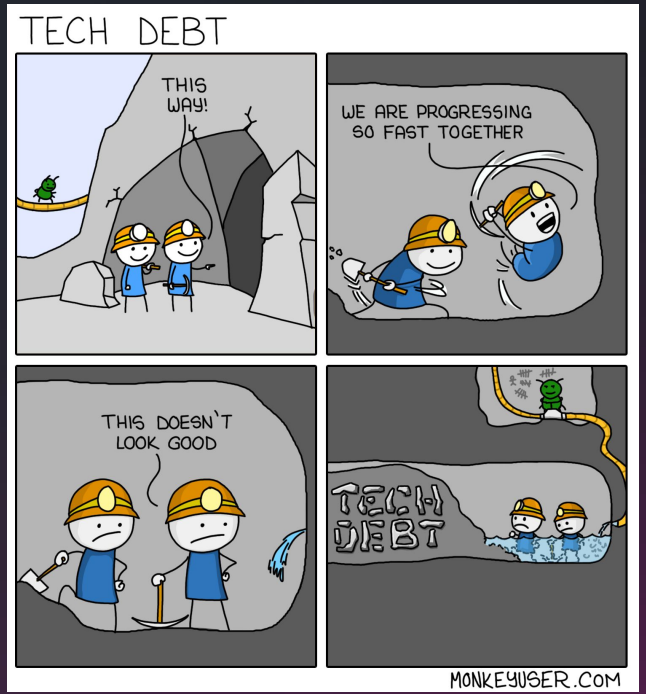
Are there projects still skipping on tests?

Wealth of resources on testing:

- Popularized by test-driven development
- Test coverage tools (beware “You become what you measure”)
- Unit testing and integration testing
- Negative and positive tests
- Property-based testing
- Anti-regression tests
- Continuous Integration

Testing

Are there projects still skipping on tests?



Auditing

Book your auditors early

Fuzzing

Charles Darwin meets software engineering

- Random population of inputs
- Can your software accommodate them?
- Mutate the population
- Can your software accommodate them?
- ....
- Crash your software with unlikely very specialized inputs

Fuzzing

- Blackbox fuzzing
- Whitebox fuzzing
- Greybox fuzzing / coverage-guided

Symbolic Execution

“In computer science, symbolic execution (also symbolic evaluation or symbex) is a means of analyzing a program to determine what inputs cause each part of a program to execute. An interpreter follows the program, assuming symbolic values for inputs rather than obtaining actual inputs as normal execution of the program would.
It thus arrives at expressions in terms of those symbols for expressions and variables in the program, and constraints in terms of those symbols for the possible outcomes of each conditional branch.
Finally, the possible inputs that trigger a branch can be determined by solving the constraints.”

Consider the program below, which reads in a value and fails if the input is 6.

```
1  int f() {
2    ...
3    y = read();
4    z = y * 2;
5    if (z == 12) {
6      fail();
7    } else {
8      printf("OK");
9    }
10 }
```

Formal verification

Model-checking

- Model checking is brute-forcing all possible states of your are of interest and ensuring that all pre-conditions, post-conditions and invariants are maintained.
- Necessary even for supposedly “safe” language, example https://github.com/tokio-rs/loom
  To ensure that concurrent data structures have no race conditions (which are different from data races that the Rust borrow checker addresses)
- Can spot design bugs

Formal verification

Deductive verification

- Armed with a proof assistant, you provide proofs, for example of your input ranges or array bounds, when the assistant cannot do the proof itself.

Formal verification

Correct-by-construction

- Using a formally proven compiler, you generate your code from a formally proven design.
  You have no bugs.

Example:
- https://github.com/mit-plv/fiat-crypto
- https://github.com/hacl-star/hacl-star

What’s next for ZK?

- Fuzzing (we’re working on zkEVM fuzzing at Taiko!)
- Using symbolic execution, SAT or SMT solver to narrow down on edge cases that humans (including auditors) can miss.

Pro Tip: Auditors always start with fuzzing if it’s possible. Saves a lot of manpower to find buggy areas.

taiko

High-Assurance ZK

Building foundations you can trust

Mamy Ratsimbazafy

ZK Engineering @ Taiko

mratsim

m_ratsim