# Fuzzing zkVMs

A report on progress toward secure scaling of Ethereum

Cody Gunton - Ethereum Foundation

# Outline

- The basic challenges of fuzzing
- SNARK pipelines and where we might fuzz them
- First focus: program execution and ISA compliance
- Findings for security and observations on tooling
- Future directions

# Fuzzing?

The art of throwing random inputs at a program to try to break it.

Factors for success:

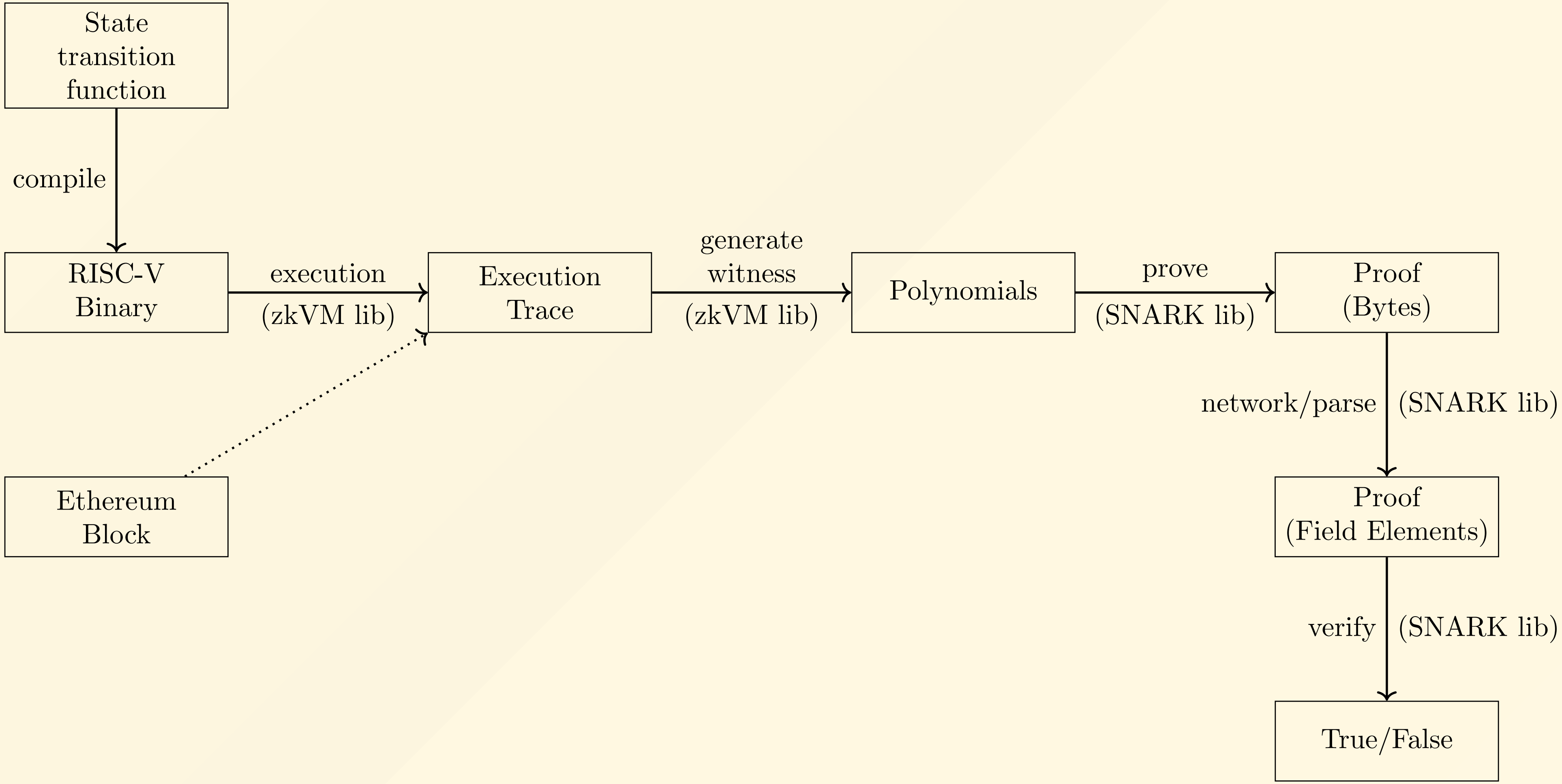
- Quality of generated inputs
  - Do the inputs "go deep"?
  - What is the "throughput" of novel inputs?
- Speed of input generation

# Fuzzing is not formal verification

Formal verification : "prove" that your code conforms to a spec.

- A major ongoing initiative [https://verified-zkevm.org/](https://verified-zkevm.org/)
- Formal verification is not foolproof
- High degree of assurance has a high startup cost
  - Want tools that are easier to implement
  - Want to find and fix bugs before formal verification

# Generic zkVM pipeline



# RISC-V Primer

RISC-V is a family of ISAs rv{XLEN}{exts} where

- XLEN is 32 or 64 (the register size)
- exts is a list of letters describing the available structures
  - Popular in SNARK settings: rv32im
  - Ideal for Go, Java,... compiled to SNARKs: rv64gc
    - Want this for client diversity!

# First Investigation: Execution

How can we gain confidence that a RISC-V implementation is correct?

Naive differential fuzzing:

- Generate a random RISC-V program
- Execute on a trusted reference (Sail model), recording checkpoints
- Execute on target, recording the same
- Compare

# Testing

- Unit tests: https://github.com/riscv-software-src/riscv-tests
  - Issue #368: doesn't work directly for rv32im
  - The testing framework assume old version of base set (rv32i)
- Architecture tests https://github.com/riscv-non-isa/riscv-arch-test
  - Reasonable framework for differential fuzzing
  - Implemented for some; results mostly good; should keep looking
  - Found a circuit bug

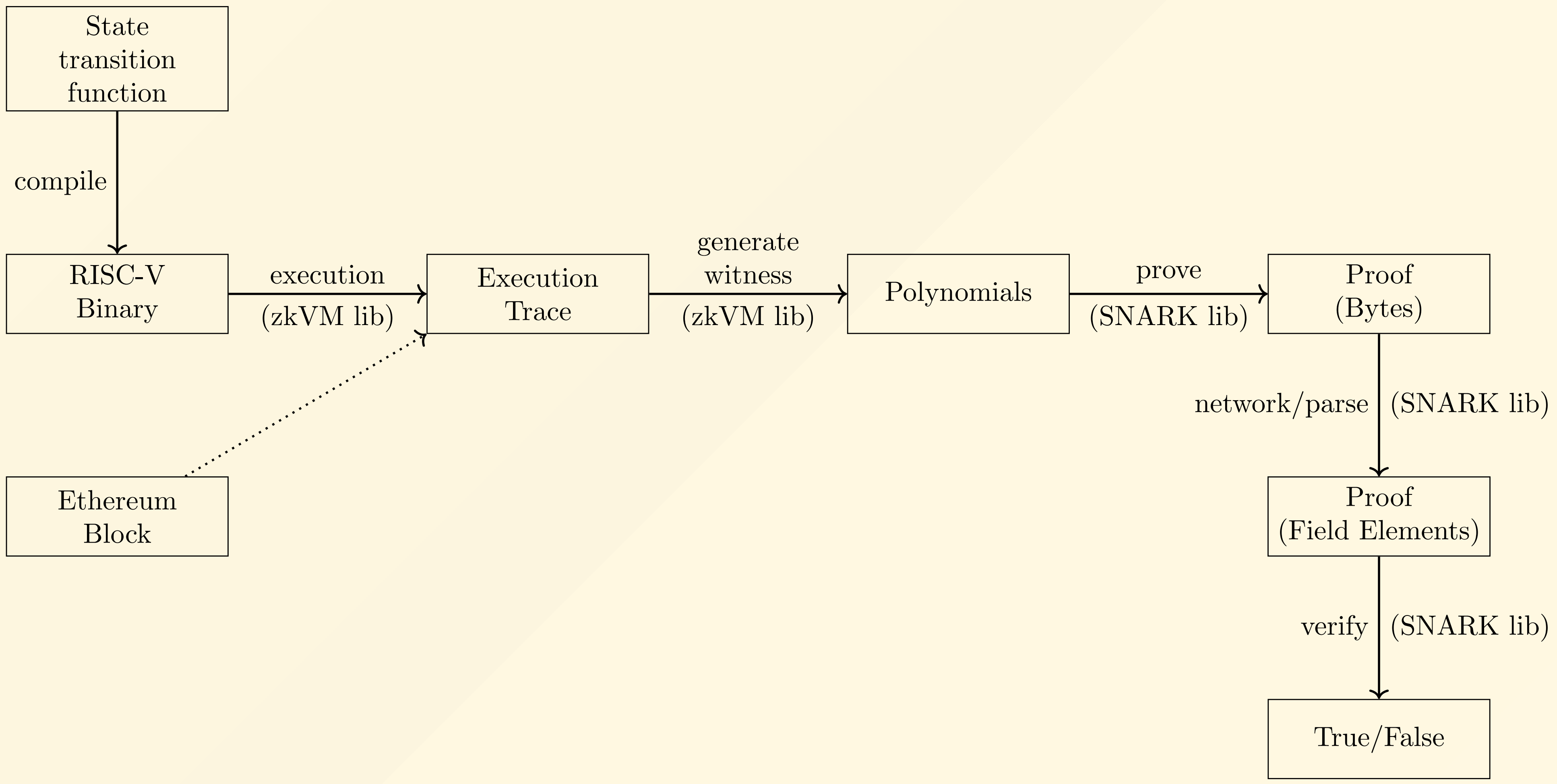
# Test Binary Generation

- Generate then compile (e.g. CSmith for C programs)
- Generate RISC-V directly with risc-dv
  - Assumes you have SystemVerilog and want to simulate
  - Issue 979: not compatible with Verilator ($\approx$ only free option)
- Cascade: looks good; claims to solve throughput issue well

# Smarter Input Generation

- Described approach is robust but coarse
    - How do you know what you're testing?
- Introspective approachs more work but very powerful
    - Coverage guidance: track stats about which lines were executed
    - Need to code template for fuzzer to find interesting cases
    - Can using existing tools (e.g. Honggfuzz); but FFIs?

# The Pipeline Again



# Circuit Correctness

- A huge concern!
- Security firms have many sophisticated tools in development.
- We care about completeness bugs because we care about liveness (need to be able to prove valid transactions are valid)
- Fuzzing can also help us find soundness bugs (cf Hochrainer, Isychev, Wüstholz, Christakis "Fuzzy Processing Pipelines...")

# Prover Correctness

- Can focus on commonly used libraries
- Can extend circuit generation pipeline
    - Will be slow though
- Can also focus on components (sumcheck? PCS?)?

# Verifier Correctness

- Seems easier to fuzz the verifier
  - Semantically simpler space of inputs
  - Weaker performance requirements $\Rightarrow$ simpler tech stack (?)
  - Verification is very fast $\Rightarrow$ can cover many inputs

# Thanks for your attention!

## References


- Solt, M., Lees, P., Koens, T., & Gerlitz, O., *Cascade: CPU Fuzzing via Intricate Program Generation*, USENIX Security '24, 2024.
- Hochrainer, C., Isychev, A., Wüstholz, V., & Christakis, M., *Fuzzing Processing Pipelines for Zero-Knowledge Circuits*, arXiv:2411.02077 (Nov 2024).