A Practical Introduction to zkVMs

OSCW 2025

01

What is a zkVM?

TL;DR

A virtual machine implemented using algebraic circuits, rather than silicon.

Provable execution of any program compiled to the target ISA (e.g. RISC-V, WASM)

Key properties

Knowledge soundness

It is computationally infeasible to produce a proof without knowledge of a valid execution trace.

Zero-knowledge (privacy)

The proof reveals no information, beyond the public claim.

Succinctness

Verifying a proof is asymptotically more efficient than re-executing the computation.

Misconception

Most “zero-knowledge VMs” don’t formally provide zero-knowledge.

*Some do provide weaker notions of privacy

02

How is it built?

Execution trace

Basically **a table**, with **each row** containing **a "cycle"** of the processor, the state of the **hardware registers at a point in "time"**

Constraints within and between row **act as the wiring and gates** that implement the state.

Input 1	24
Input 2	30

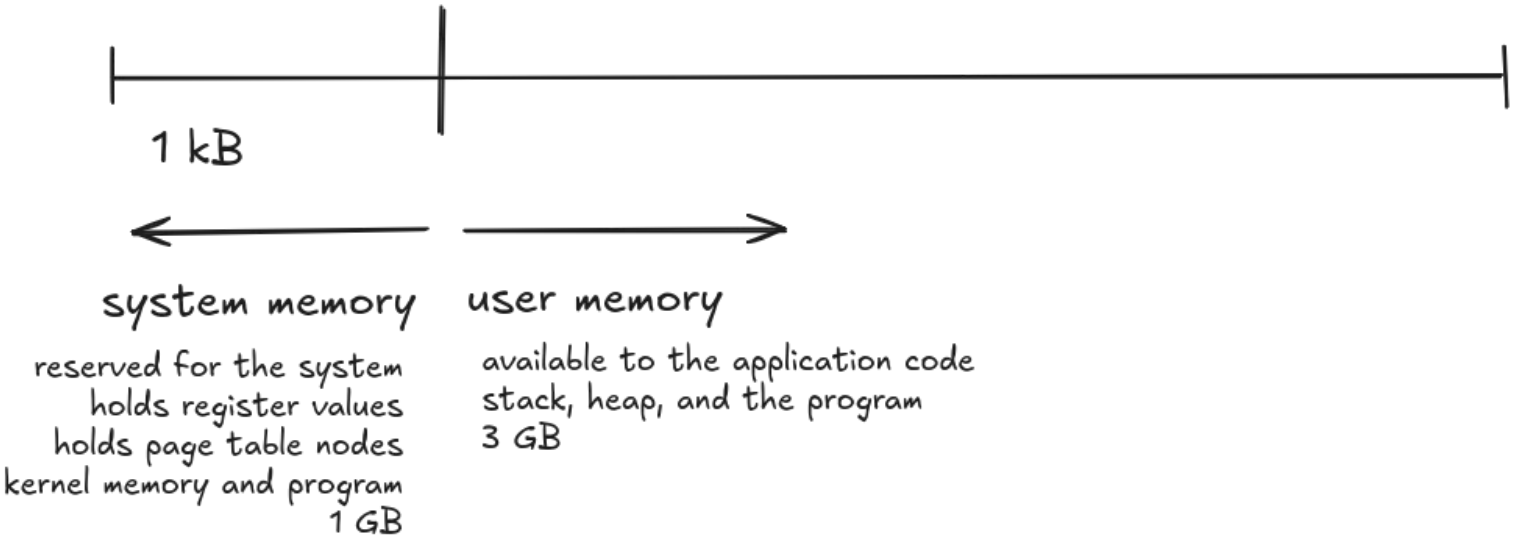
Modulo	97

Asserted Output	28

	Clock Cycle	Data Column 1	Data Column 2	Data Column 3	Control Column - Initialization	Control Column - Transition	Control Column - Termination
Execution Trace - Initialization	0	24	30	54	1	0	0
Execution Trace - Transition	1	30	54	84	0	1	0
Execution Trace - Transition	2	54	84	41	0	1	0
Execution Trace - Termination	3	84	41	28	0	1	1

Memory

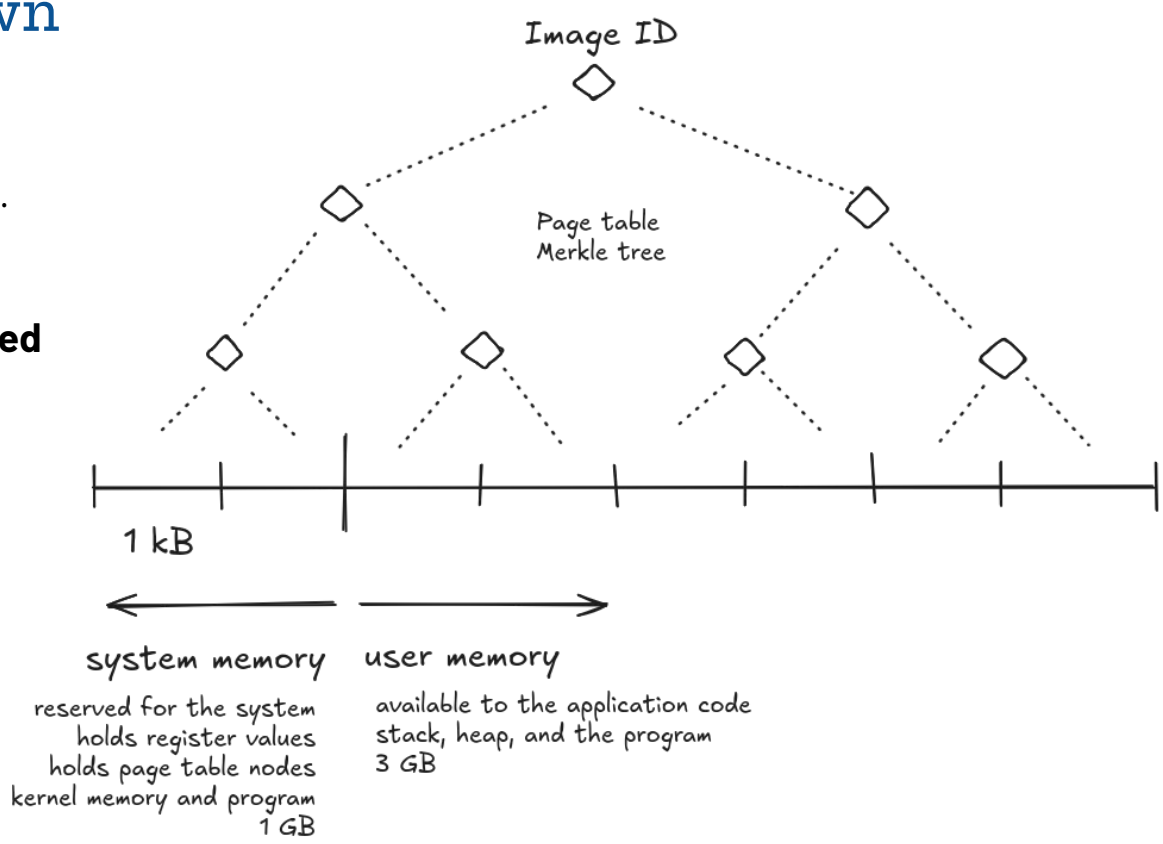
Random access memory (**RAM**) is provided by a **permutation argument**, which uses a multiset equivalency check to establish that the **values read equal the values written**.



Startup & shutdown

At the **start of execution**,
initial **memory** state is
loaded from a memory image.

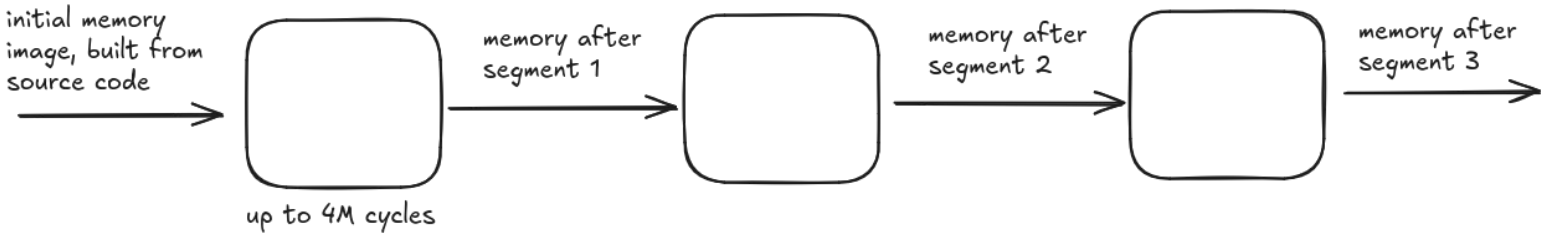
At the **end of execution**,
the final **memory state is stored**
and any **output is committed**.



Segments

Proving can only handle bounded size execution traces (e.g. 1M cycles).

Long executions are broken into segments, linked by the initial and final memory images.



Compute Compression

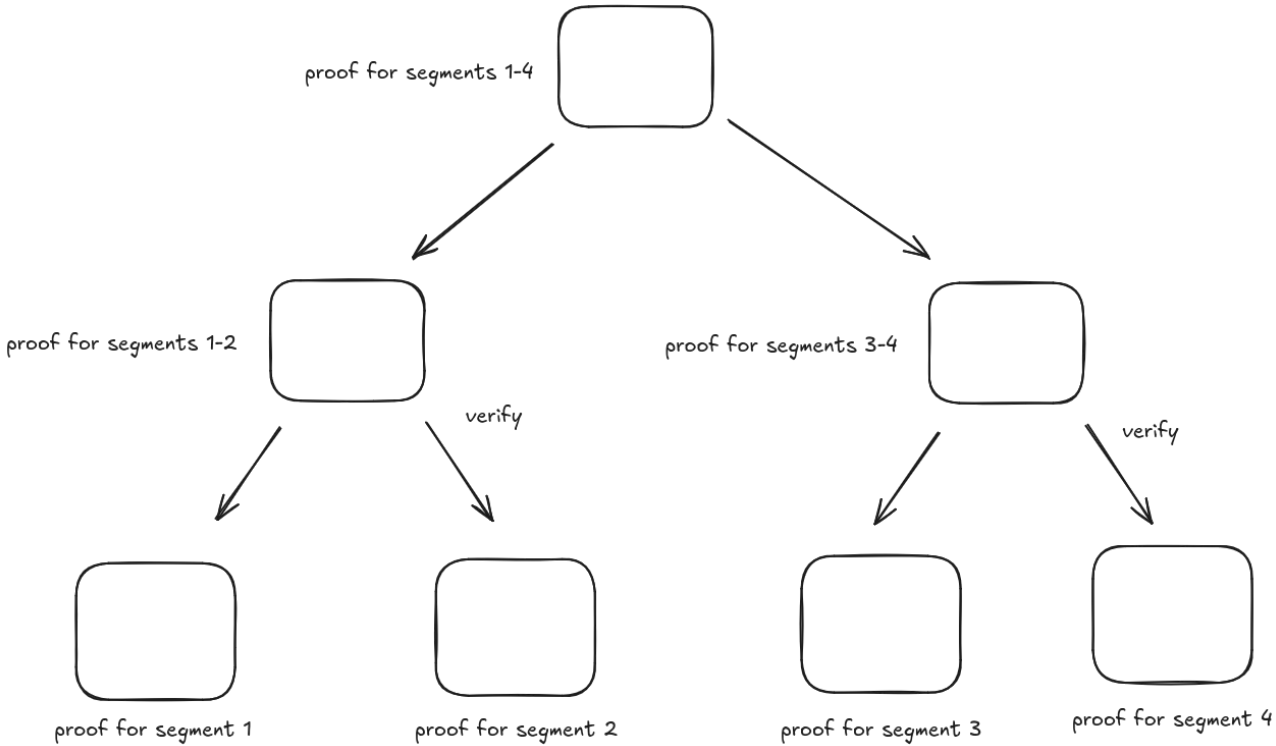
Each segment produces a separate proof. Total proof size grows linearly.

A 1 million cycle segment has a proof of ~270 kB.
1 billion cycles, split into 1000 segments of 1 million cycles produces ~270 MB

Proofs are compressed via recursive verification of 2 proofs to produce 1 proof.

Can be further compressed with recursive verification via Groth16 or KZG-Plonk.

Compute Compression



03

Battleship

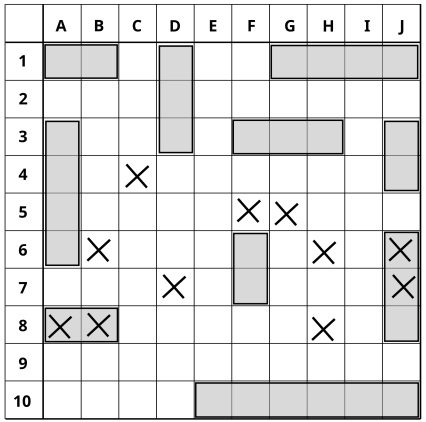
Battleship

Battleship

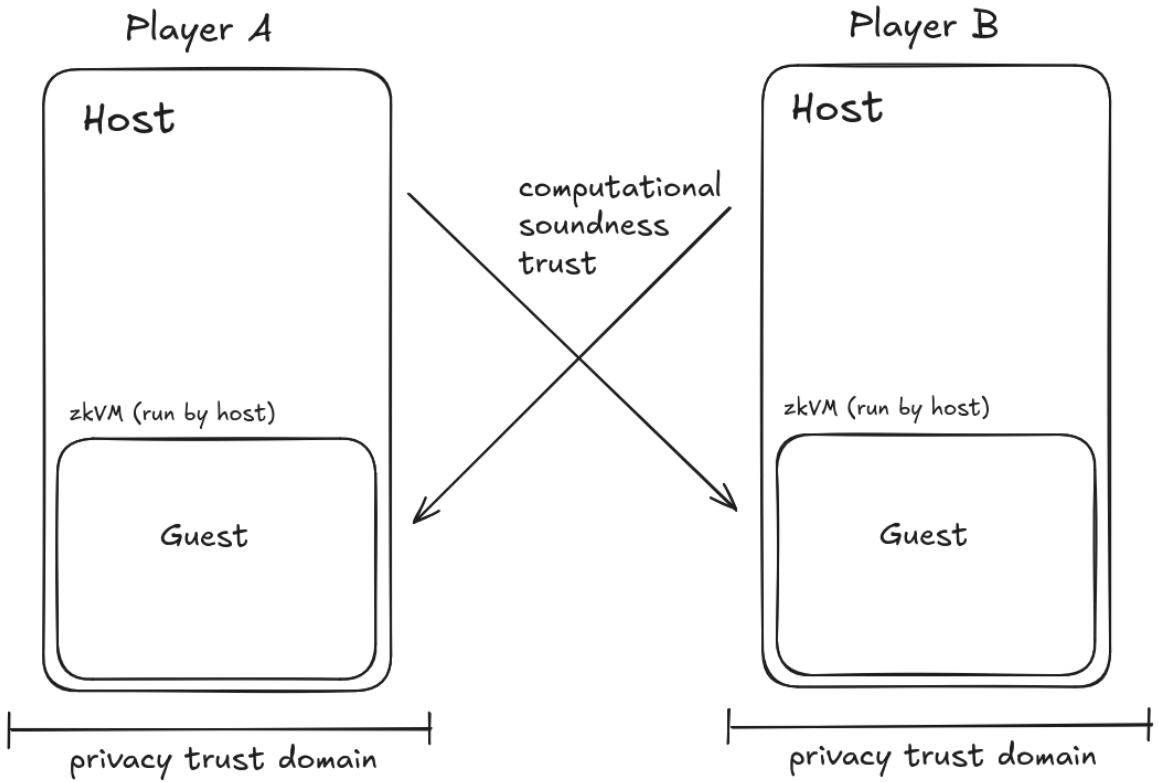
Game starts by choosing an arrangement of ships on their board.

Players take turns taking shots at opponent.

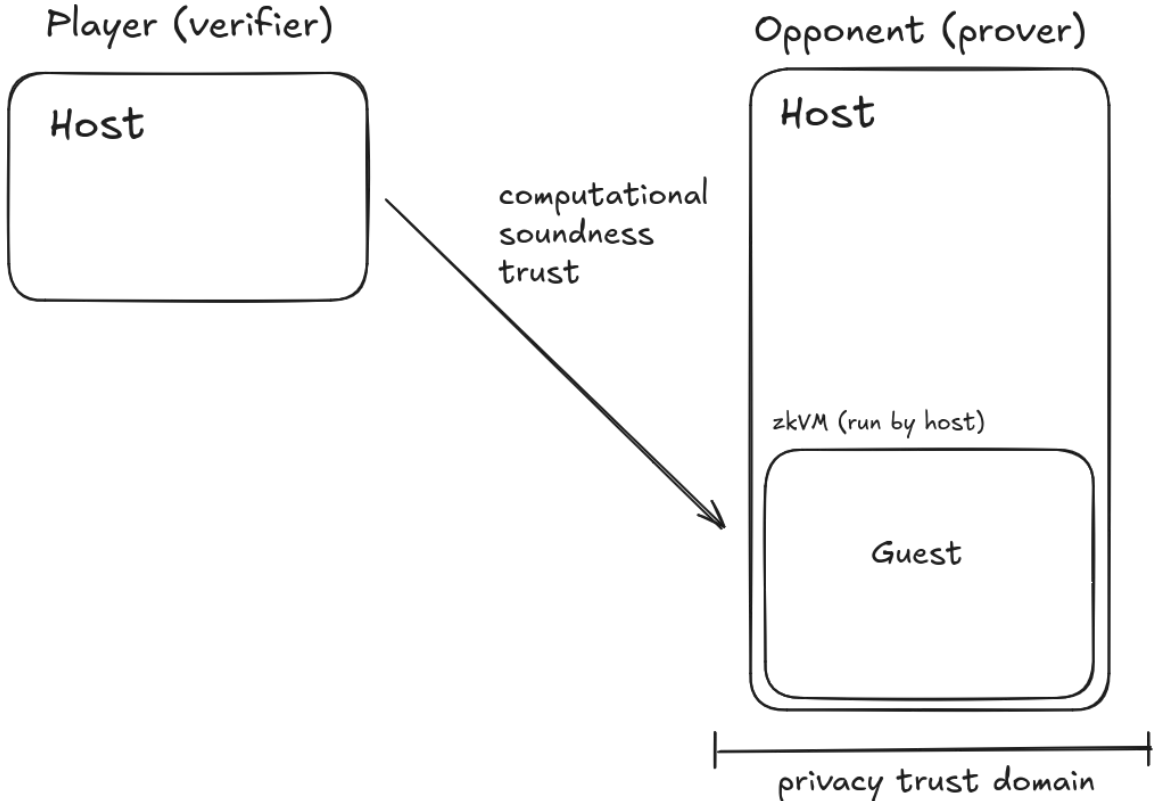
Game ends when a player's ships are sunk.

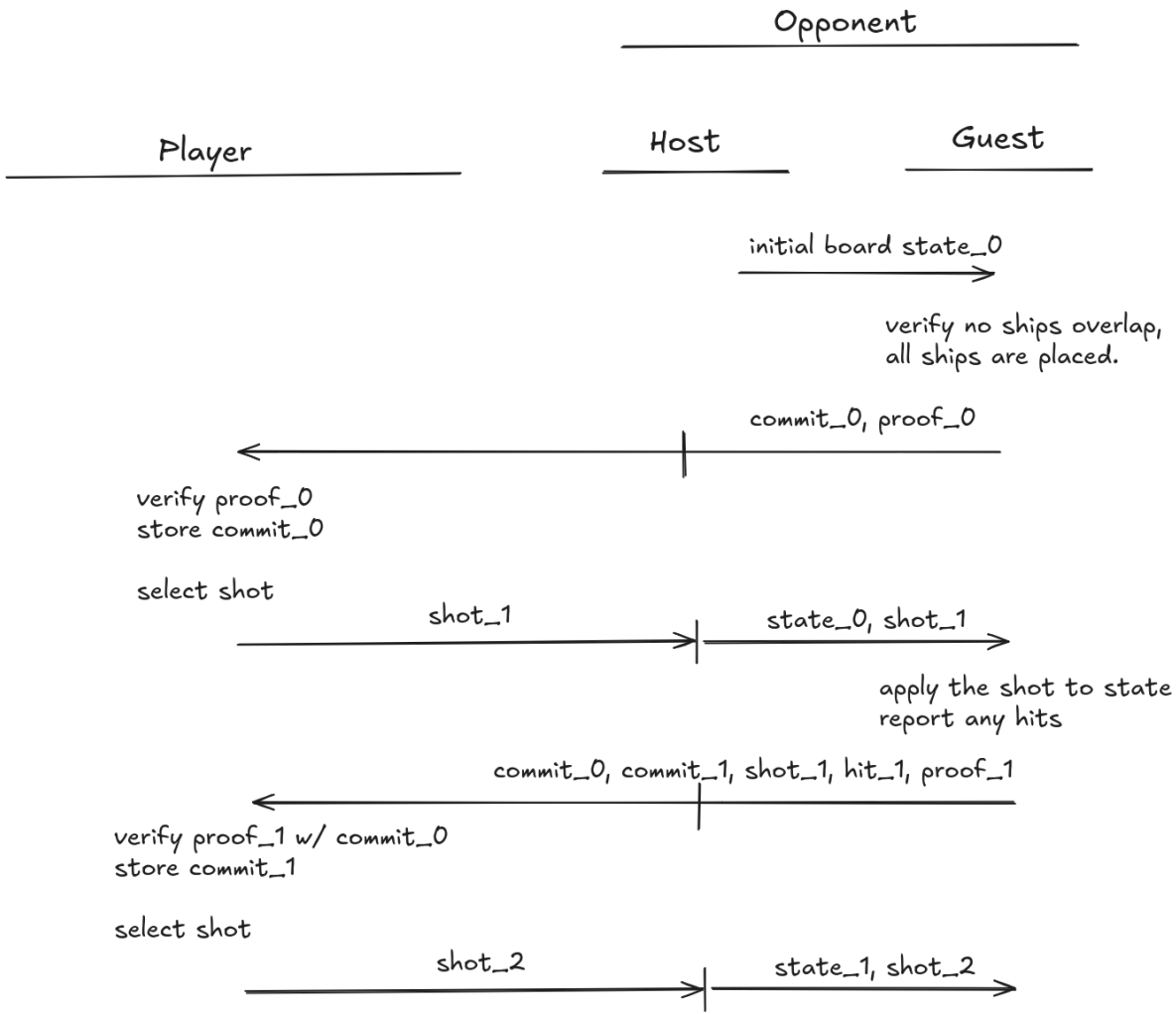


(Logical) Parties



(Logical) Parties




Opponent
Player
Host
Guest
initial board state_0
verify no ships overlap,
all ships are placed.
commit_0, proof_0
verify proof_0
store commit_0
select shot
shot_1
state_0, shot_1
apply the shot to state
report any hits
commit_0, commit_1, shot_1, hit_1, proof_1
verify proof_1 w/ commit_0
store commit_1
select shot
shot_2
state_1, shot_2

github.com/nategraf/oscw-zkvms-battleship

04

Next steps

Deploying an application

Image IDs act as the **verification key** (aka a public key) for **for guest program.**

Like public keys, distributing this image ID securely to the verifier is critical.

Embedding the image ID as a constant in your program is the most common solution.

Making this image ID recoverable from the source code increases transparency. RISC Zero and other zkVMs support reproducible builds (by running Cargo in a Docker container).

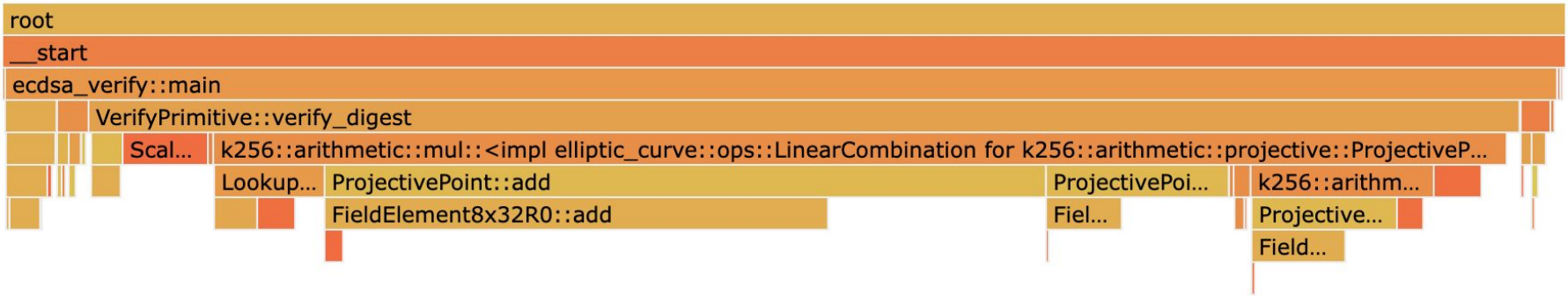
Optimization

Optimizing guest applications is often key to making proving practical.

Optimizing a guest program is 80% the same as optimizing any other program.

RISC Zero supports generation of flame graphs.

```
RISC0_PPROF_OUT=profile.pb RISC0_DEV_MODE=1 cargo run
```



Cryptography acceleration in the guest

Many guests need to run cryptography such as signature verification or hashing.

Using RISC-V to implement cryptography can take a large amount of execution cycles, making proving slow and expensive.

RISC Zero and other zkVMs support “accelerators” or “precompiles” which utilize additional algebraic circuits to act as a “coprocessor” to the CPU.

More questions?

victor@risczero.com

Examples

	ISA	Proof system
RISC Zero	RISC-V	RISC Zero (STARK)
SP1	RISC-V	Plonky3 (STARK)
Jolt	RISC-V	Jolt
OpenVM	RISC-V	Plonky3 (STARK)
Polygon Miden	Custom	Miden (STARK)
Cairo	Custom	Stone & Stwo (STARK)
Polygon zkEVM	EVM	STARK
ZKSync Era	EVM	Boojum (STARK)