Using Ideas from Hardware to Accelerate Zero-Knowledge Virtual Machines

Eric Archerman, Celine Zhang
Mentor: Simon Langowski

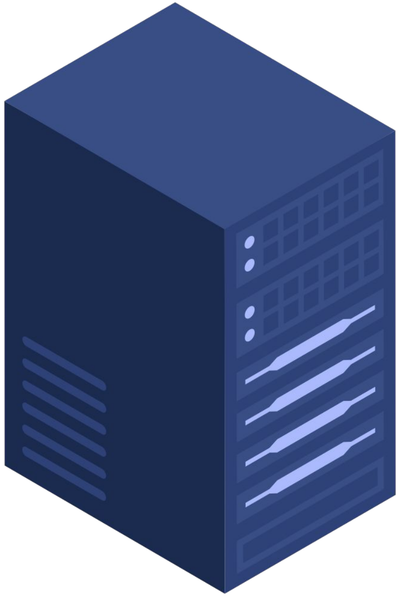
MIT PRIMES October Conference
October 12, 2024

Agenda

1. Background
   a. Verifiable computation
   b. SNARKs and Zero-Knowledge
2. Zero-Knowledge Virtual Machines
3. Hardware Optimizations
   a. Caching
   b. Batching + Multiple in-flight instructions
4. Our work + Implementation plan

Background

Motivating problem: how can we verify computation?



I ran program **P** and
got result **y**.

Is this true? I
need proof…

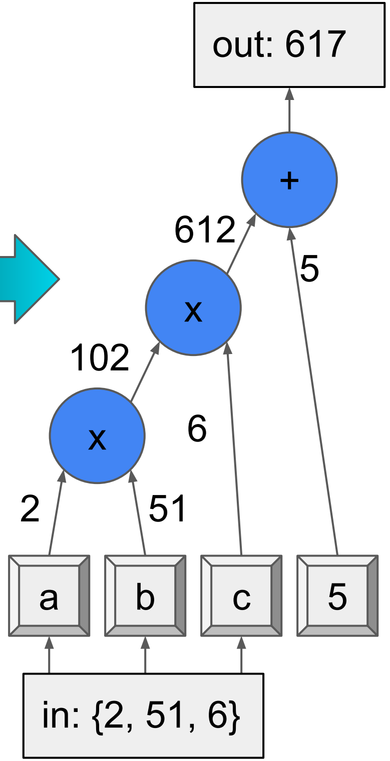
Trivially: Sam runs **P** himself, but…
Can we do better?

Traditional SNARK Design

program **P** in DSL

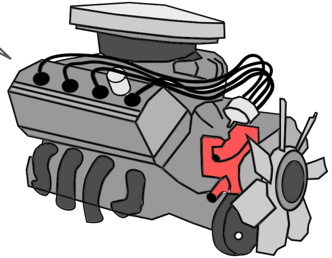
```
signal input a, b, c;
sum <== a + b;
prod <== sum * c;
res <== prod + 5;
signal output res;
```

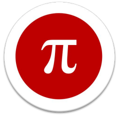
‘mathier’ representation (arithmetic circuit)



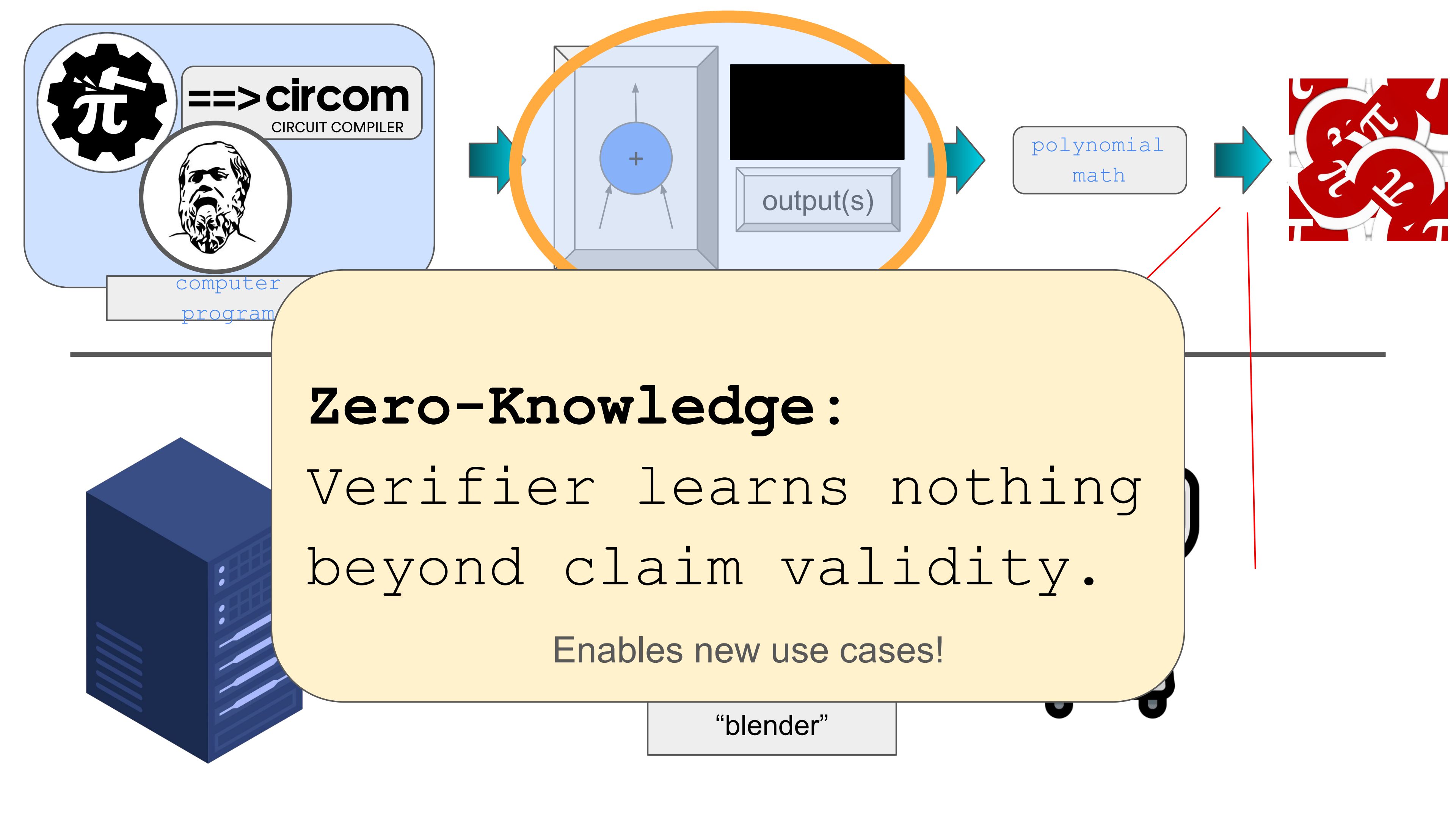
heavy
(polynomial)
computation

SNARK backend

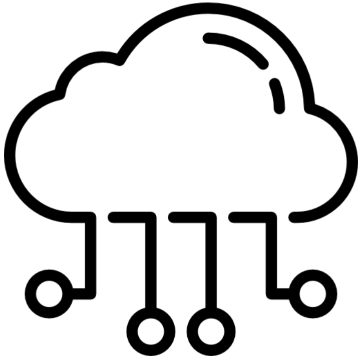




proof!


==>circom
CIRCUIT COMPILER
computer
program
+
output(s)
polynomial
math
Zero-Knowledge:
Verifier learns nothing
beyond claim validity.
Enables new use cases!
"blender"

Applications of Verifiable Computation

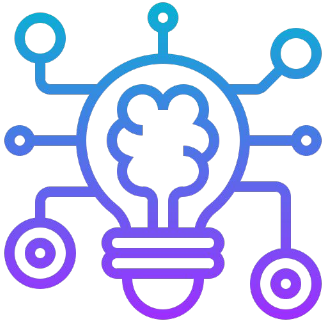


cloud computing

Offloads compute

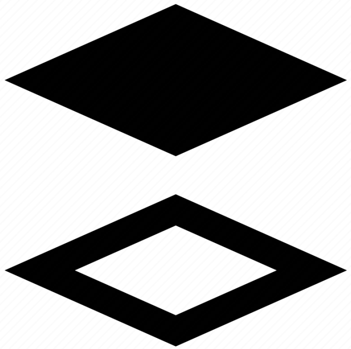
databases

Offloads storage



ML

Verifiable training and evaluation



blockchain scaling

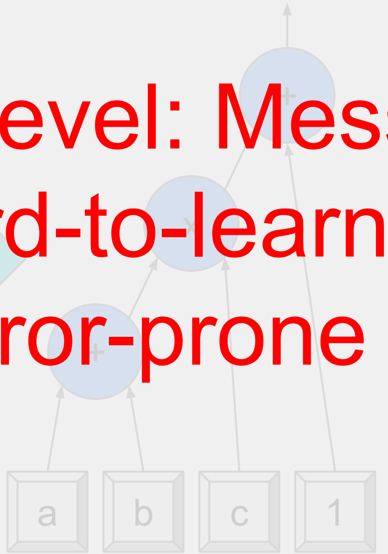
Reduces node work

Great, but…

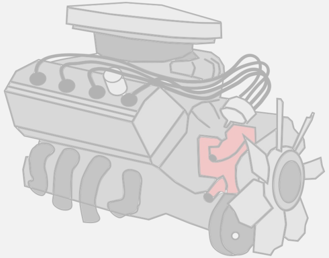
program **P** in DSL

```
signal input a, b, c;
sum <== a + b;
prod <== sum * c;
res <== prod + 1;
signal output res;
```

Intermediate rep



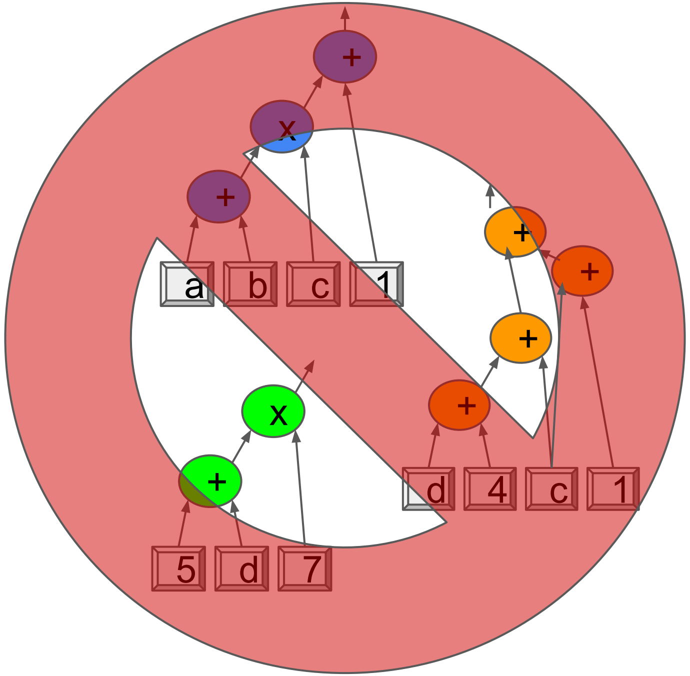
SNARK backend



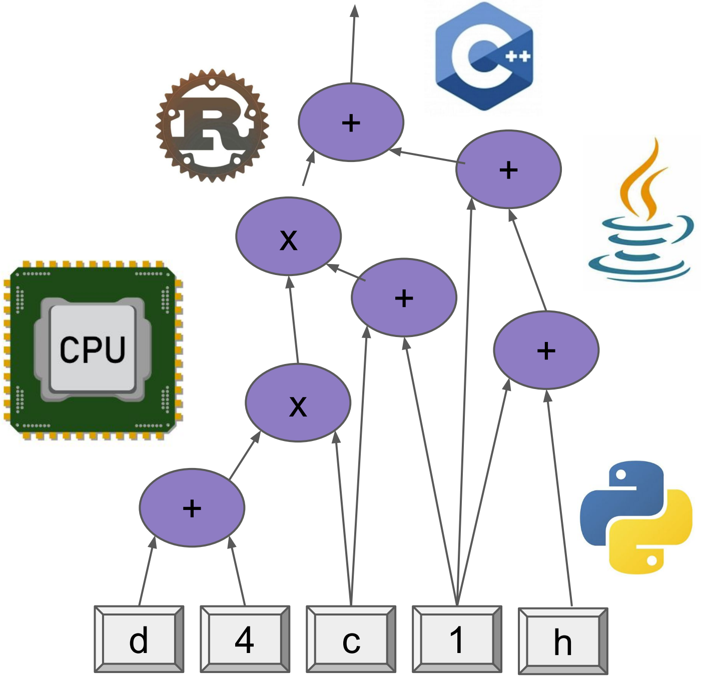
Low-level: Messy,
Hard-to-learn,
Error-prone

Towards Usability:
Zero-Knowledge Virtual Machines

The Big Idea of zkVMs

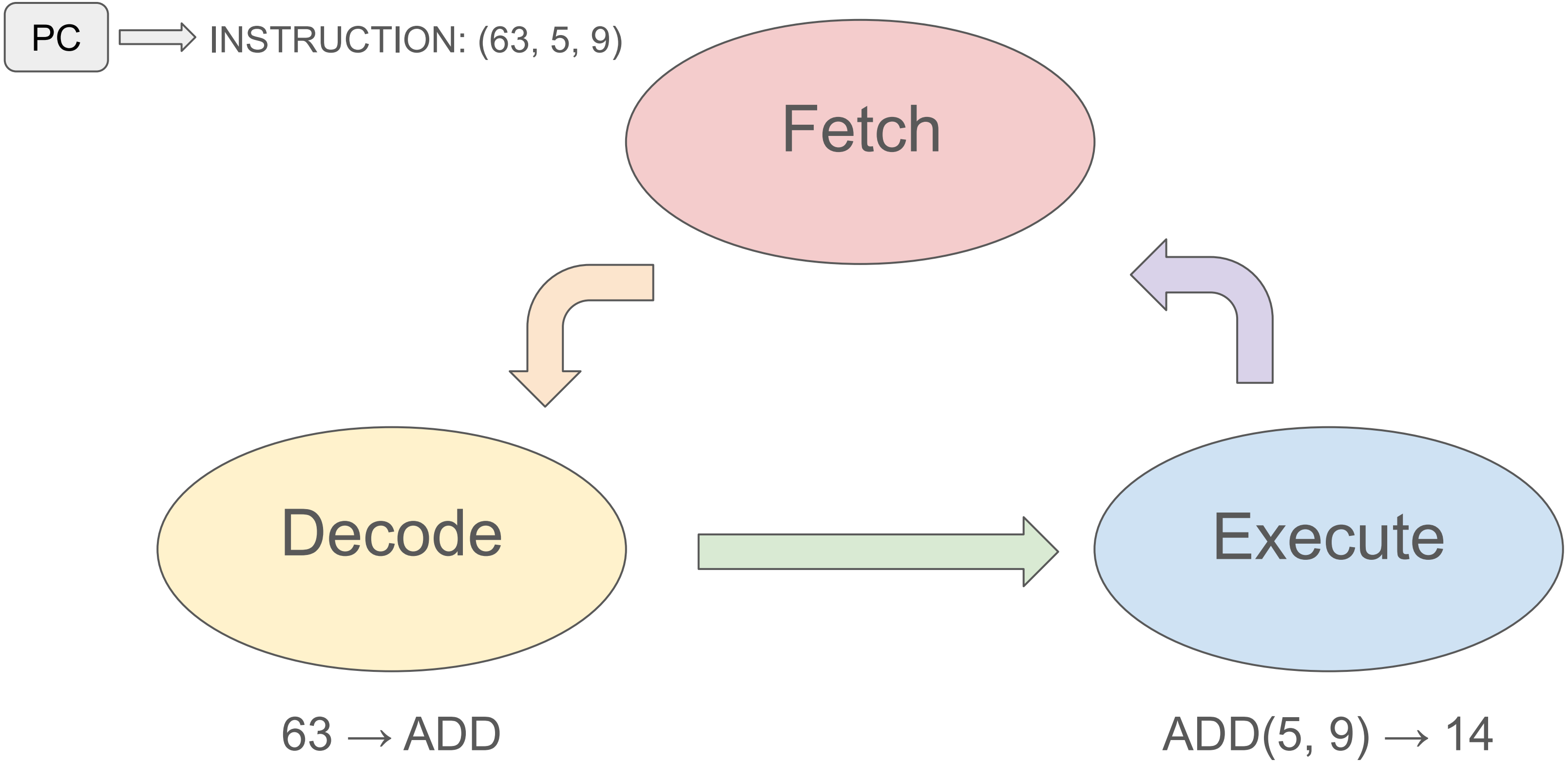


Traditional SNARKs: a different circuit for each program

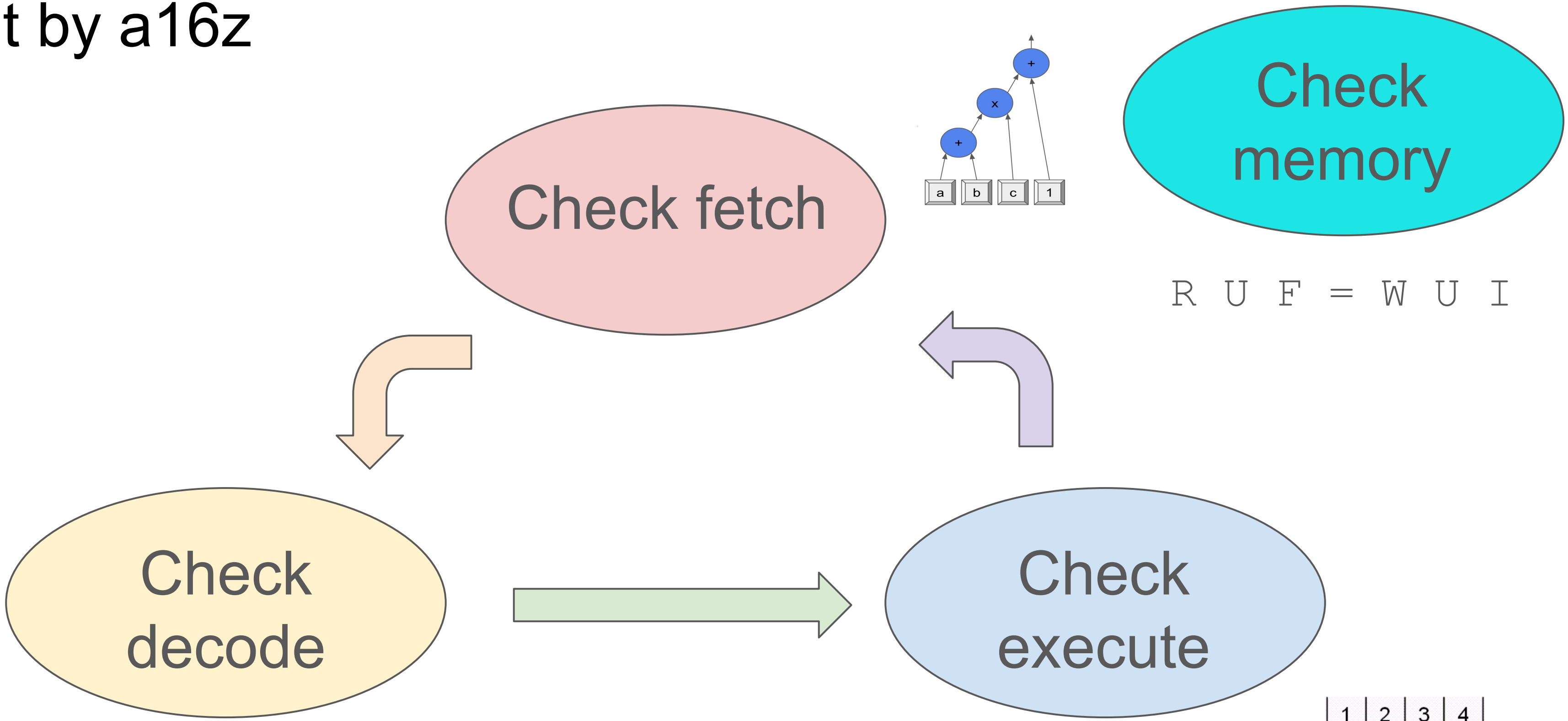


zkVMs: a single circuit that verifies CPU actions

How CPUs Process Instructions



Jolt by a16z




```
while n != 1 {
    if n % 2 == 0 {
        n /= 2;
    } else {
        n += (n << 1) + 1;
    }
```

```
BEQ(6, 0)
DIV(5, 8)
ADD(5, 4)
MUL(9, 7)
```

	1	2	3	4
1	f(1,1)	f(1,2)	f(1,3)	f(1,4)
2	f(2,1)	f(2,2)	f(2,3)	f(2,4)
3	f(3,1)	f(3,2)	f(3,3)	f(3,4)

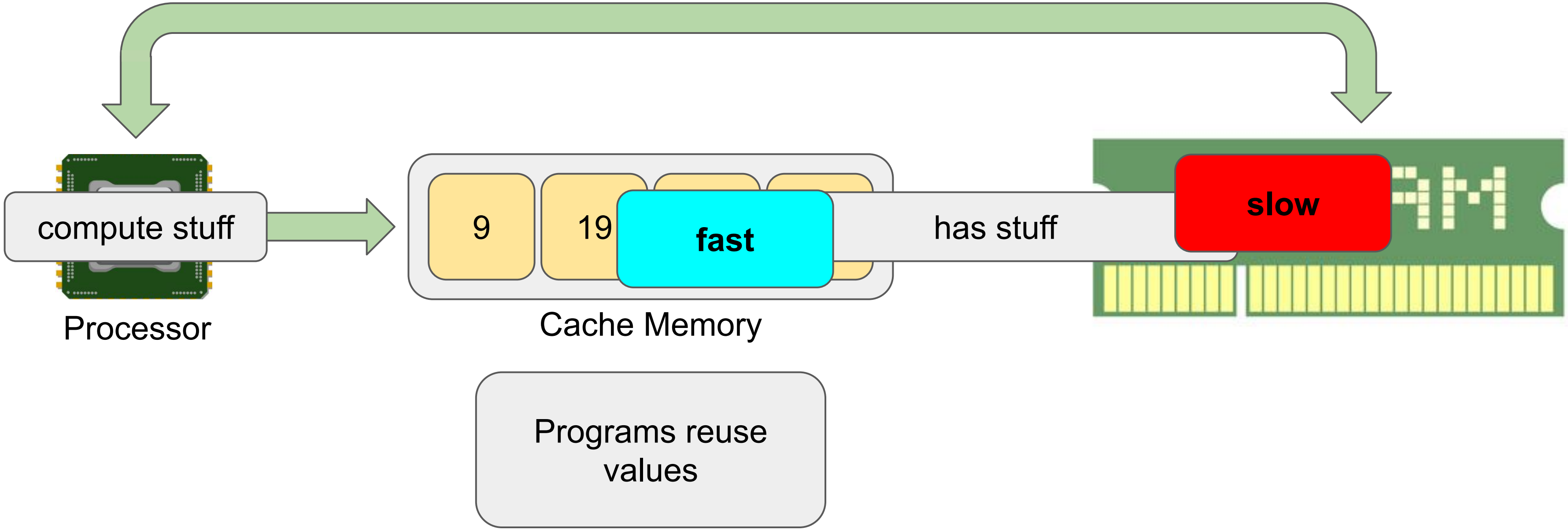
Optimizing zkVMs

- zkVMs are too slow for many practical applications
    - Overhead is introduced at every step
- We have made lots of optimizations to physical CPUs over the last 50 years, and we would like to make similar optimizations to zkVMs
    - zkVMs are a very new technology, so not much of this kind of optimization has been made

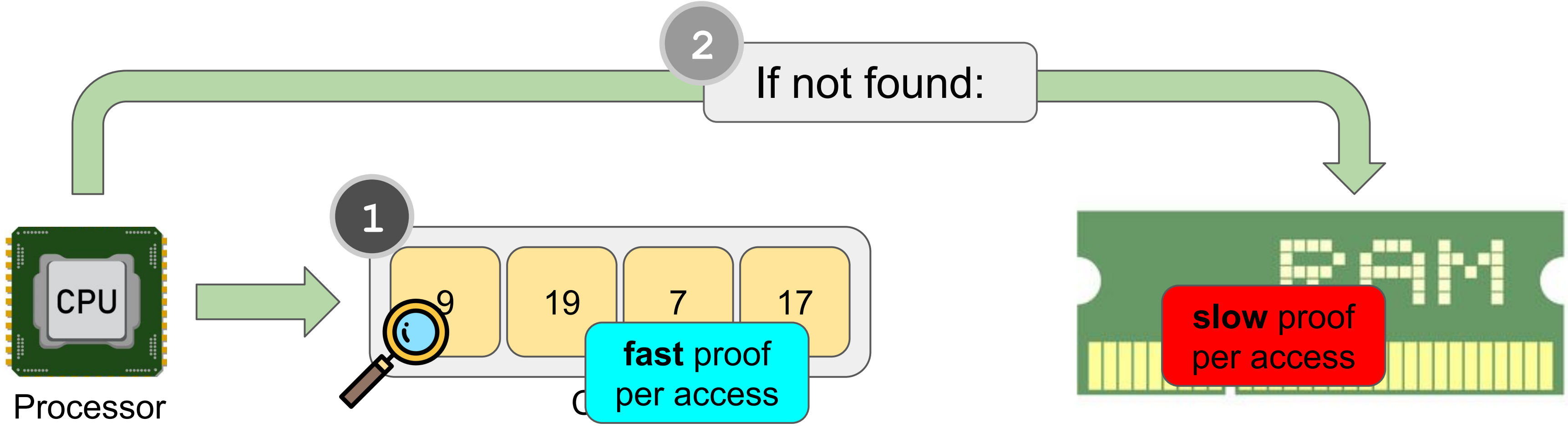
Ideas from Hardware

- Caching
- Batching
- Multiple in-flight instructions

Cache Memory



Caching for zkVMs

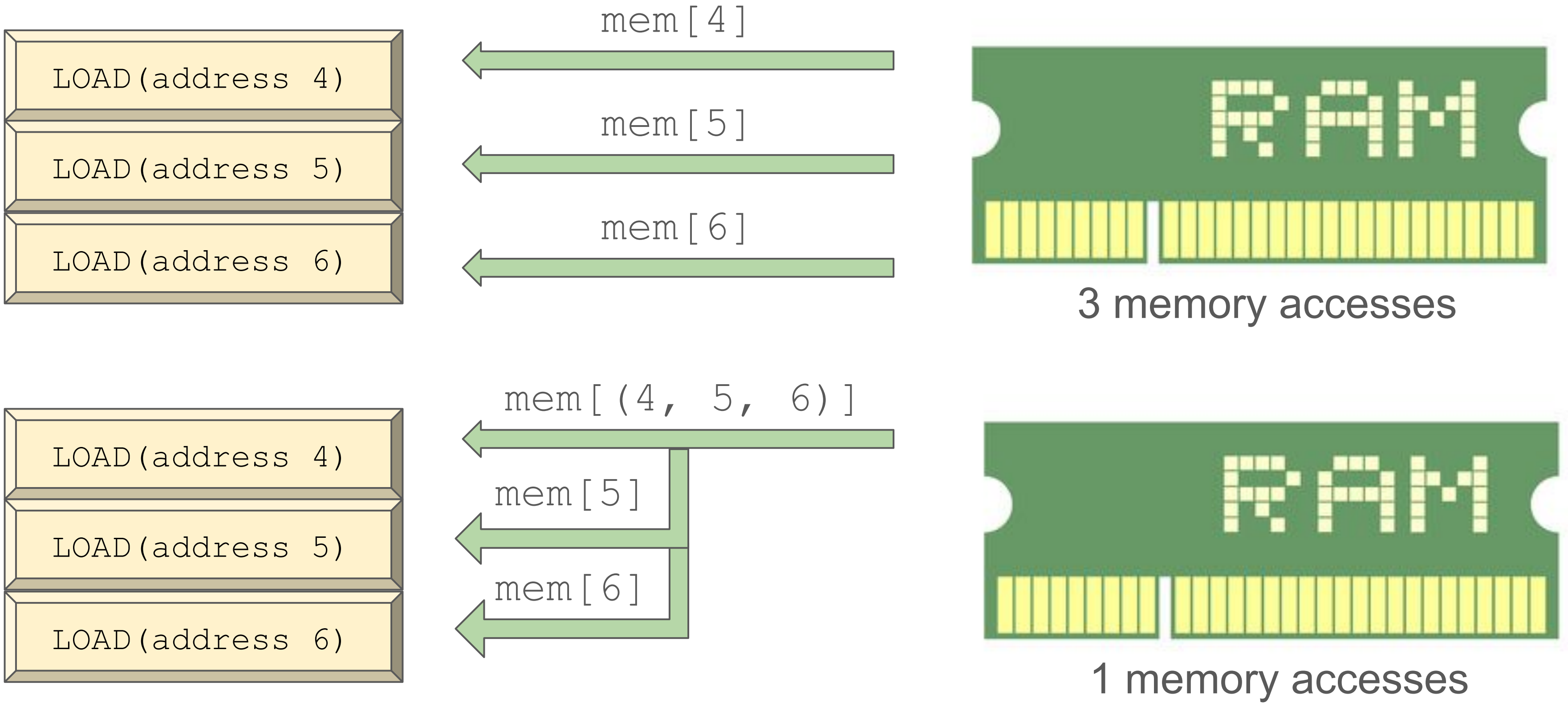


Costs:

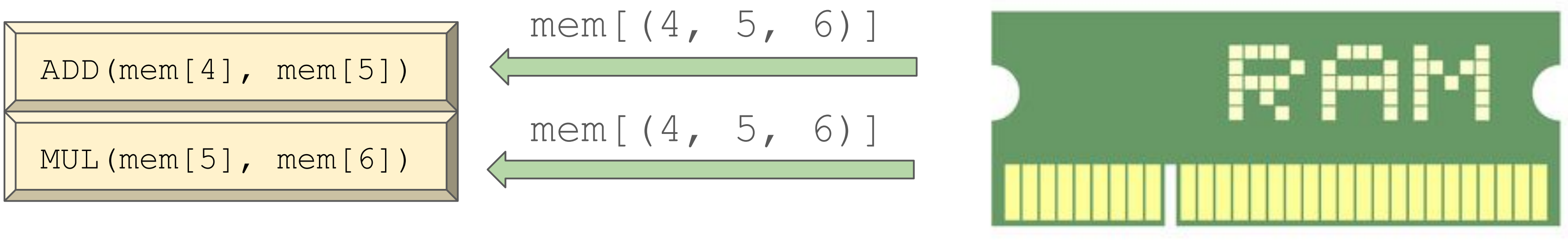
++ emulation overhead

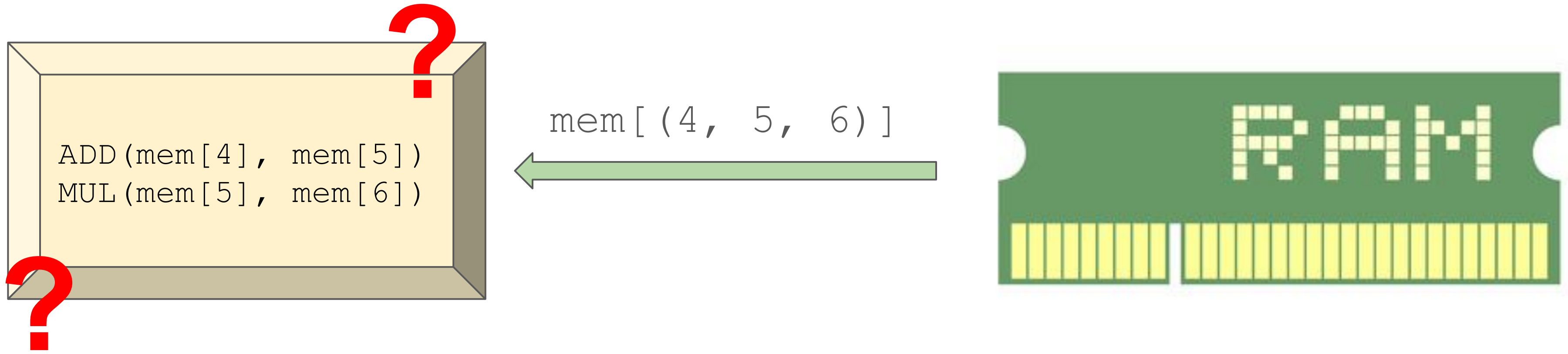
++ checks for cache security

Memory Batching



Multiple In-Flight Instructions

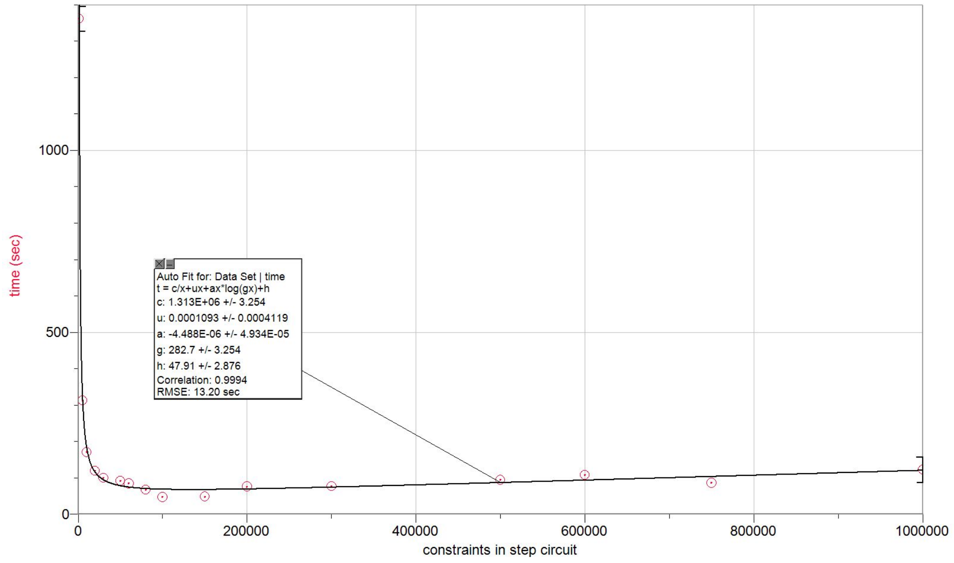




Not all groups of instructions can be verified this way!

Next Steps - The Implementation

Our Work



We have modelled the tradeoff between number of VM steps and work done in each step for a similar work, and the data indicates that there is an optimal step size

- This should correspond to the optimal number of instructions per VM step

Implementation Plan

- Caching:
    - Redesign constraint system to account for caching
        - Ensure security of cache operations to prevent prover cheating
    - Optimize caching algorithm/cache size
- Memory batching/multiple in-flight instructions:
    - Redesign memory checking to handle larger memory chunks
    - Redesign constraint system to resolve potential conflicts between multiple in-flight instructions
    - Find the optimal batch size and number of in-flight instructions
    - Fine-tune both optimizations to reap the most benefit out of both

Acknowledgements

We would like to thank everyone who made this talk possible, including:

- Our mentor Simon Langowski
- Our parents and friends, for moral support
- Uber, for driving Celine here
- Prof. Srini Devadas and Dr. Slava Gerovitch
- The MIT PRIMES program for giving us this wonderful opportunity

References

- Arasu Arun, Srinath Setty, and Justin Thaler. Jolt: SNARKs for Virtual Machines via Lookups.
- Srinath Setty, Justin Thaler, Riad Wahby. Unlocking the lookup singularity with Lasso.
- Srinath Setty, Sebastian Angel, Trinabh Gupta, and Jonathan Lee. Proving the correct execution of concurrent services in zero-knowledge.
- Xixun Yu and Zheng Yan. A survey of verifiable computation.