Verifiable ASICs

Aarhus Workshop on Secure Multiparty Computation
1 June 2016

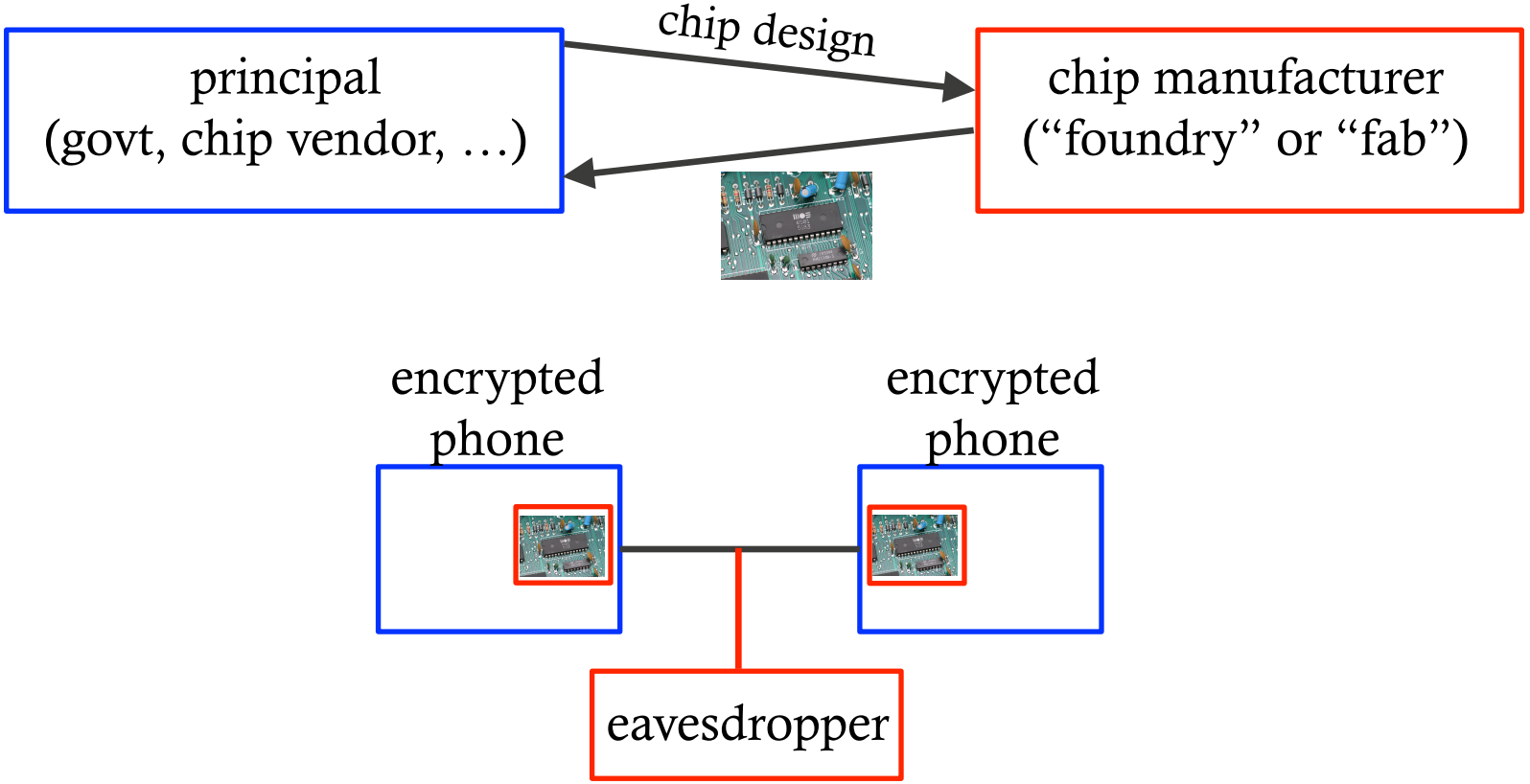
Michael Walfish
Dept. of Computer Science, Courant Institute, NYU

This is joint work with:

Riad S. Wahby (Stanford), Max Howald (Cooper Union and NYU), Siddharth Garg (NYU), abhi shelat (U. of Virginia)

Riad recently presented this work at IEEE S&P (Oakland).

Problem: the manufacturer ("foundry" or "fab") of a custom chip ("ASIC") can undermine the chip's execution.



Response: control the manufacturing chain with a trusted foundry

Trusted fabrication is the only solution with strong guarantees.

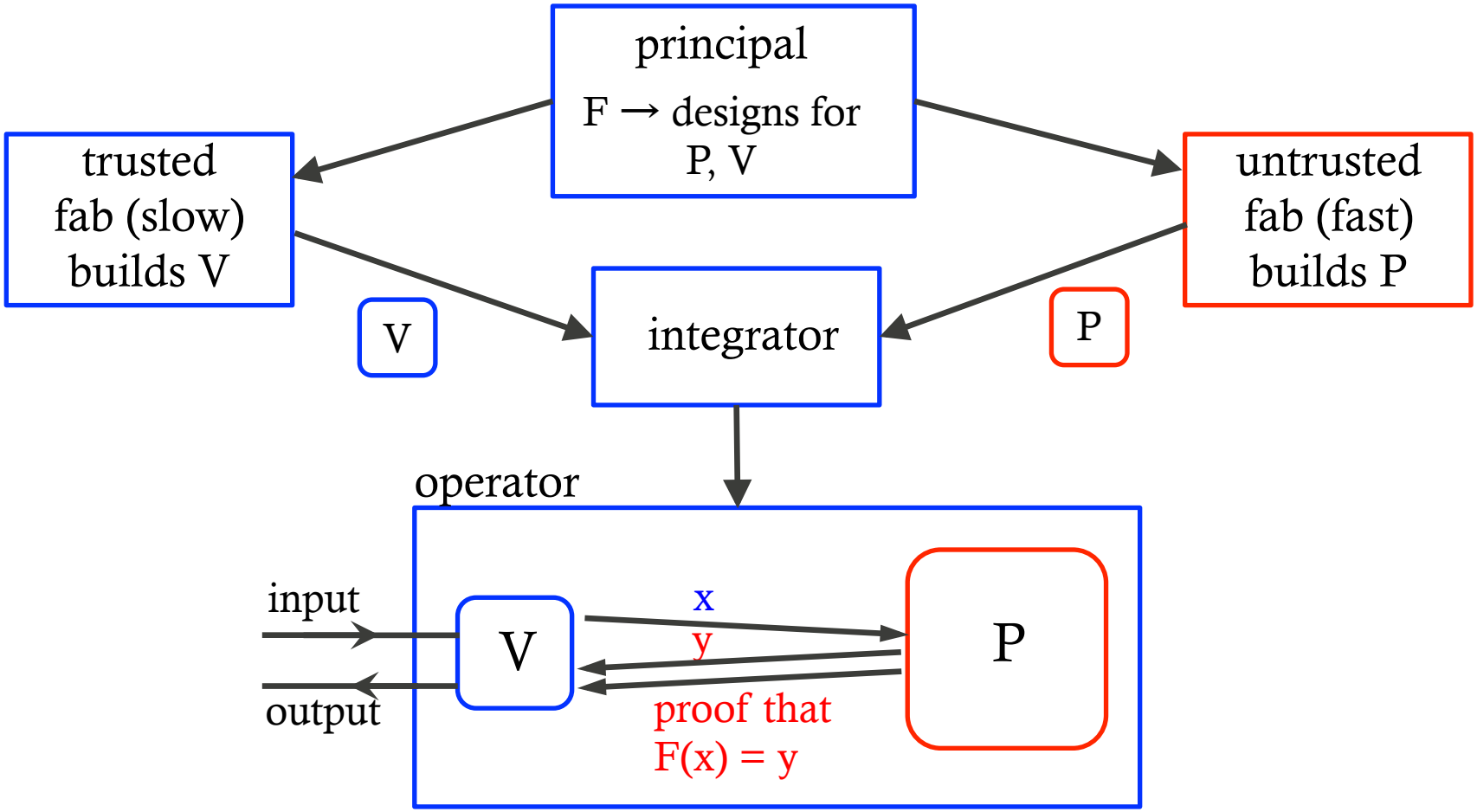
- For example, post-fab detection can be thwarted
  [A2: Analog Malicious Hardware. Yang et al., IEEE S&P 2016]

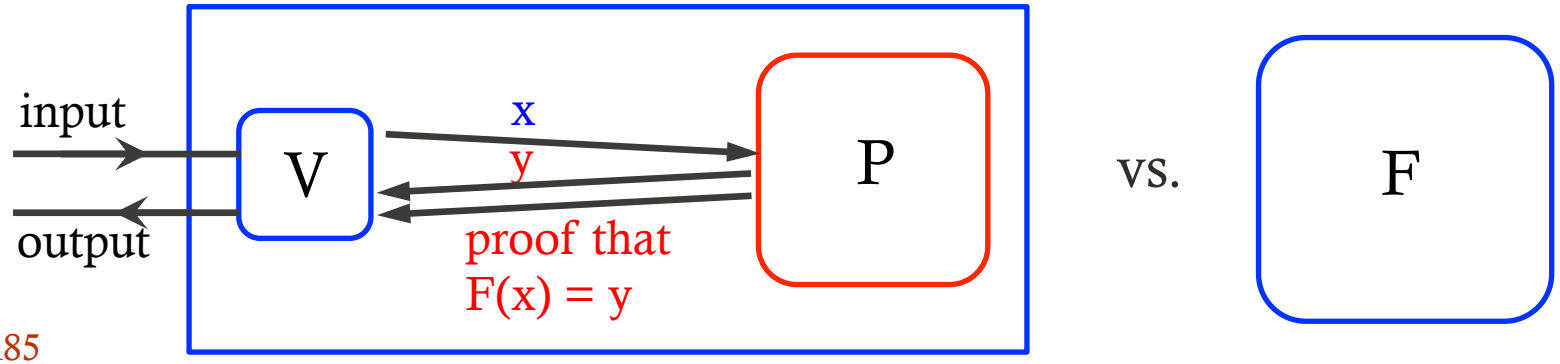
But trusted fabrication is not a panacea:

- Only 5 countries have cutting-edge fabs on shore
- Building a new fab takes $billions and years of R&D
- With semiconductor technology, area and energy reduce with square and cube of transistor dimension
- So: old fabs means enormous penalty. Example of India: $10^8\times$.

We thought: probabilistic proofs might let us get trust more cheaply!

An alternative: Verifiable ASICs





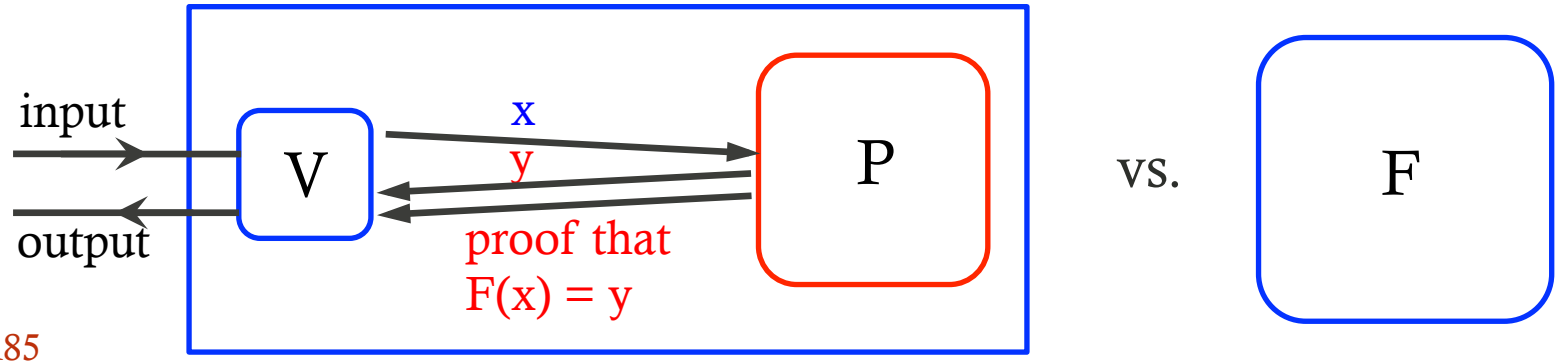
Makes sense if V + P cheaper than trusted F

Reasons for hope:

- Running time of V < F (asymptotically)
- Implementations exist, and …
- … though their costs for P are absurd, advanced fab might make P cheaper than F (!)

GMR85
Babai85
BCC86
BFLS91
FGLSS91
Kilian92
ALMSS92
AS92
Micali94
BG02
GOS06
IKO07
GKR08
KR09
GGP10
Groth10
GLR11
Lipmaa11
BCCT12
GGPR12
BCCT13
KRR14
…

SBW11
CMT12
SMBW12
TRMP12
SVPBBW12
SBVBPW13
VSBW13
PGHR13
Thaler13
BCGTV13
BFRSBW13
BFR13
DFKP13
BCTV14a
BCTV14b
BCGGMTV14
FL14
KPPSST14
FGP14
WSRHBW15
BBFR15
CFHKKNPZ15
CTV15
KZMQCPPSS15



Makes sense if V + P cheaper than trusted F

Reasons for ~~hope~~ caution:

- The theory is silent about feasibility (and the onus here is heavier than in prior work)
- Costs must reflect hardware: energy, area, ….
- We need physically realizable designs and plausible computation sizes

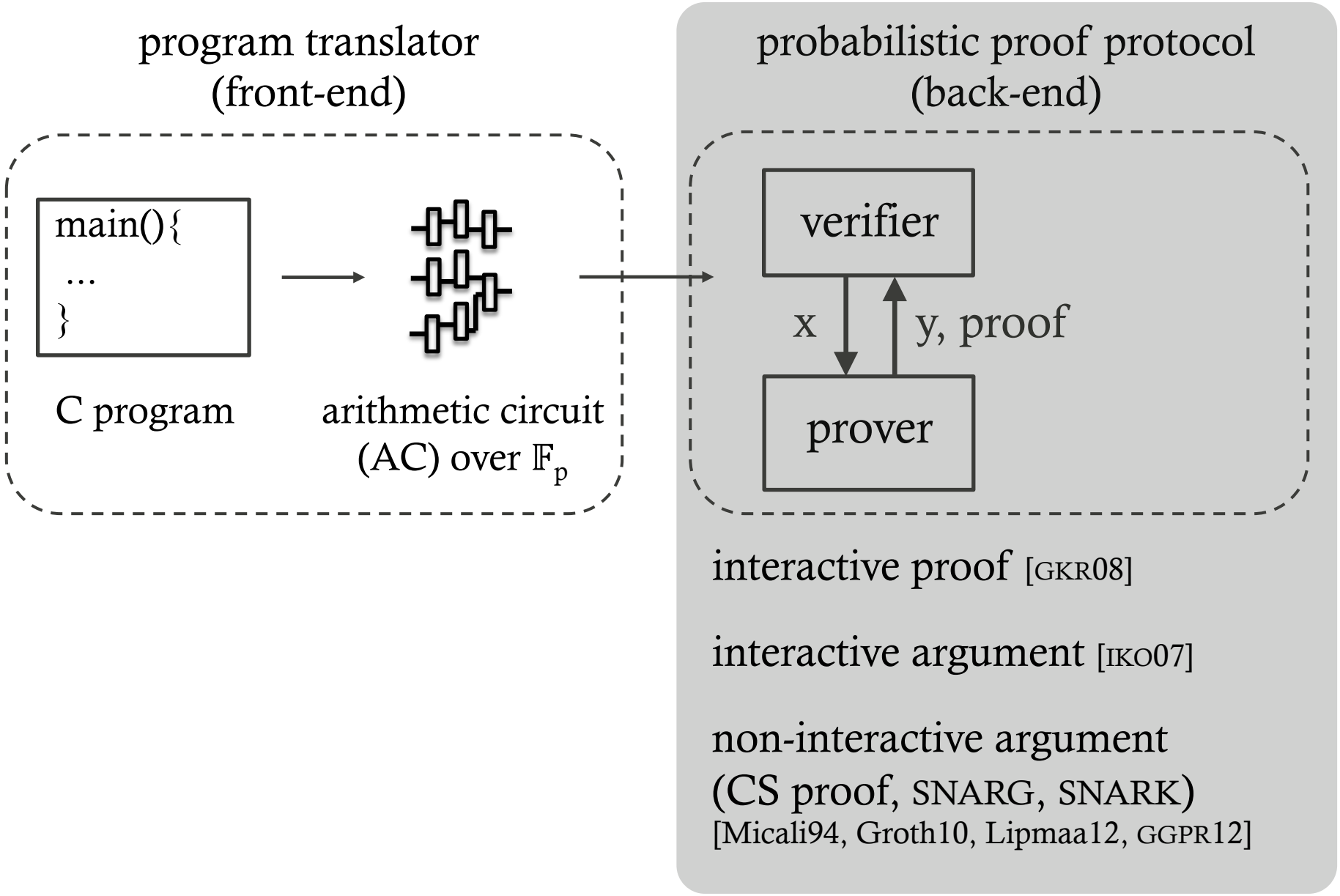
GMR85
Babai85
BCC86
BFLS91
FGLSS91
Kilian92
ALMSS92
AS92
Micali94
BG02
GOS06
IKO07
GKR08
KR09
GGP10
Groth10
GLR11
Lipmaa11
BCCT12
GGPR12
BCCT13
KRR14
…

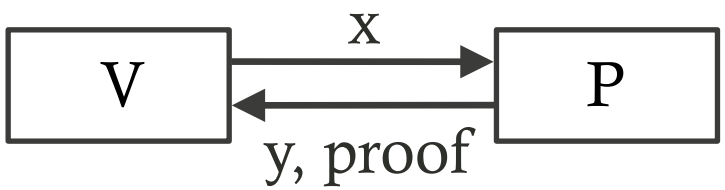
SBW11
CMT12
SMBW12
TRMP12
SVPBBW12
SBVBPW13
VSBW13
PGHR13
Thaler13
BCGTV13
BFRSBW13
BFR13
DFKP13
BCTV14a
BCTV14b
BCGGMTV14
FL14
KPPSST14
FGP14
WSRHBW15
BBFR15
CFHKKNPZ15
CTV15
KZMQCPPSS15

(1) Zebra: a system that saves costs

(2) … sometimes

Implementations of probabilistic proofs:





arguments
(interactive, SNARK, CS proof, etc.)
[GGPR12, PGHR13, SBVBPW13, BCTV14]

- non-deterministic ACs
- arbitrary AC geometry
- 1-round, 2-round protocols

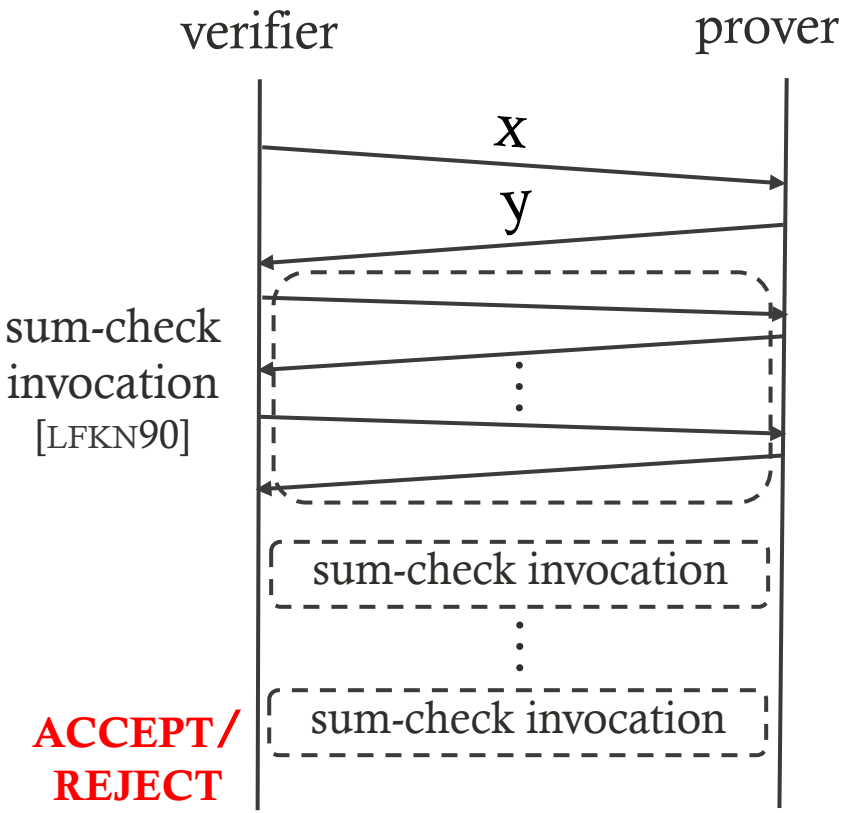
unsuited to hardware

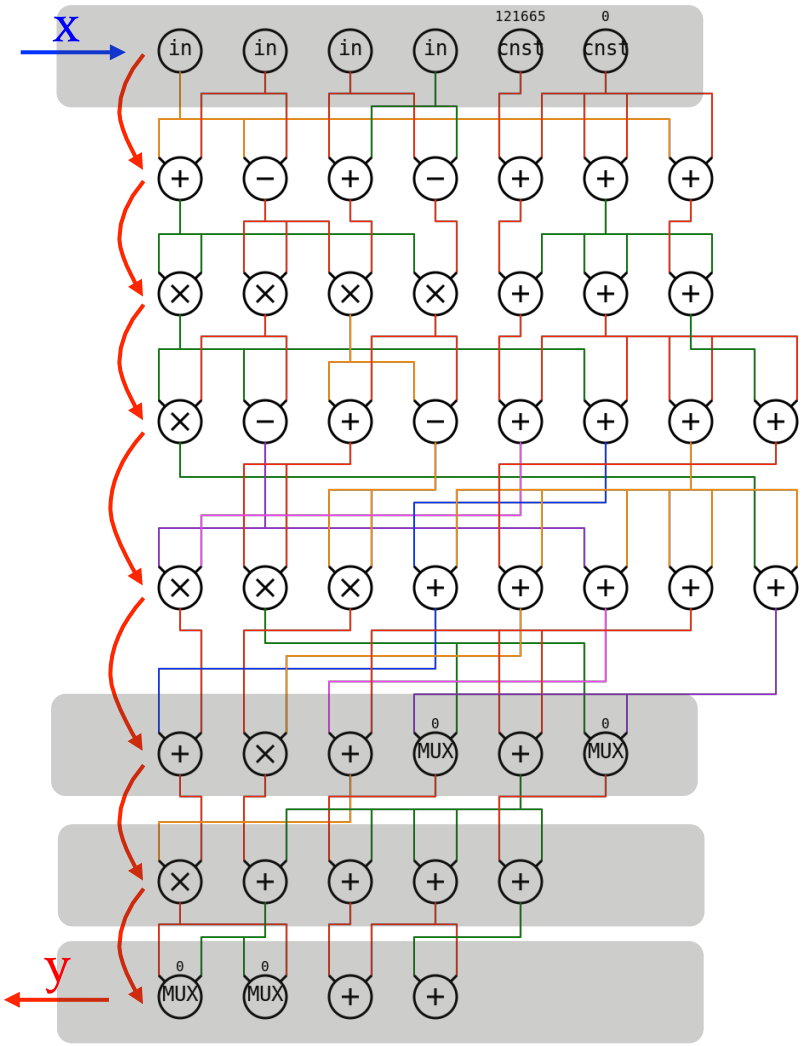
interactive proofs
[GKR08, CMT12, VSBW13]

- deterministic ACs only
- layered, low-depth ACs
- lots of rounds, communication

suited to hardware

Zebra builds on the GKR interactive proof [GKR08, CMT12, VSBW13]; computations are expressed as layered arithmetic circuits over $\mathbb{F}_p$



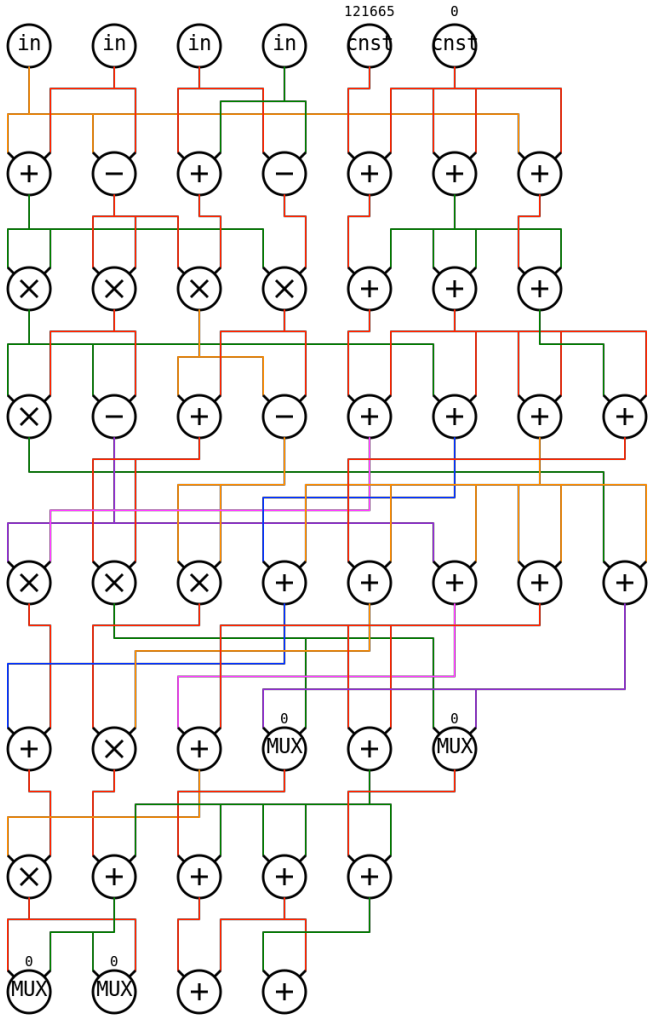


Zebra builds on the GKR interactive proof [GKR08, CMT12, VSBW13]; computations are expressed as layered arithmetic circuits over $\mathbb{F}_p$

Soundness error:
miniscule for large p

Cost to execute F directly:
O(depth · width)

V's sequential running time:
O(depth · log width + |x| + |y|),
assuming precomputation of queries



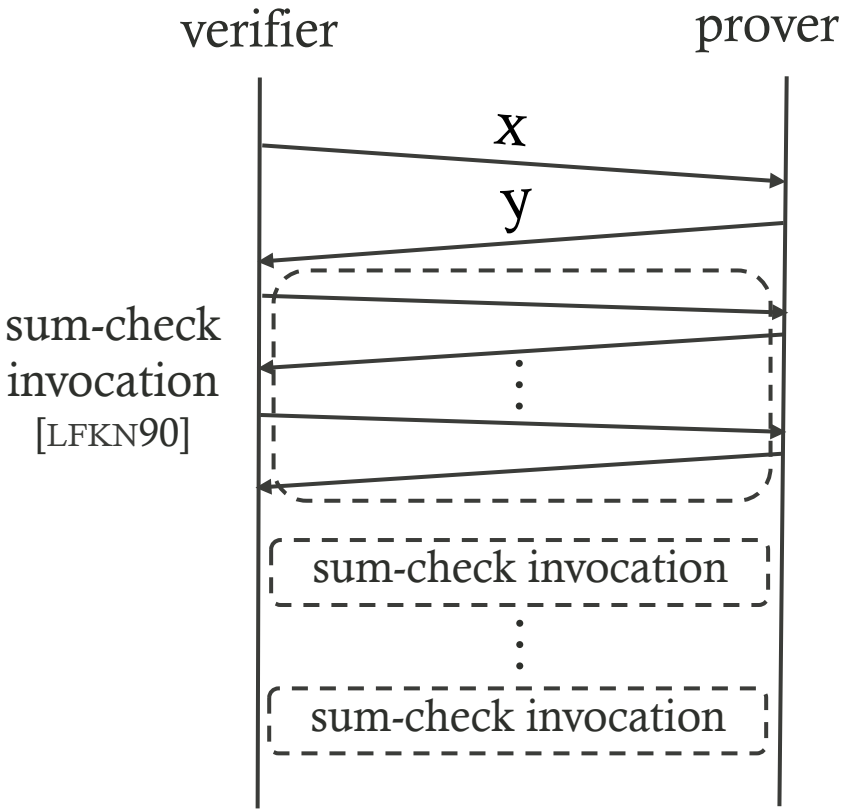
Zebra builds on the GKR interactive proof [GKR08, CMT12, VSBW13]; computations are expressed as layered arithmetic circuits over $\mathbb{F}_p$

Soundness error:
miniscule for large p

Cost to execute F directly:
O(depth · width)

V's sequential running time:
O(depth · log width + |x| + |y|),
assuming precomputation of queries

P's sequential running time:
O(depth · width · log width)

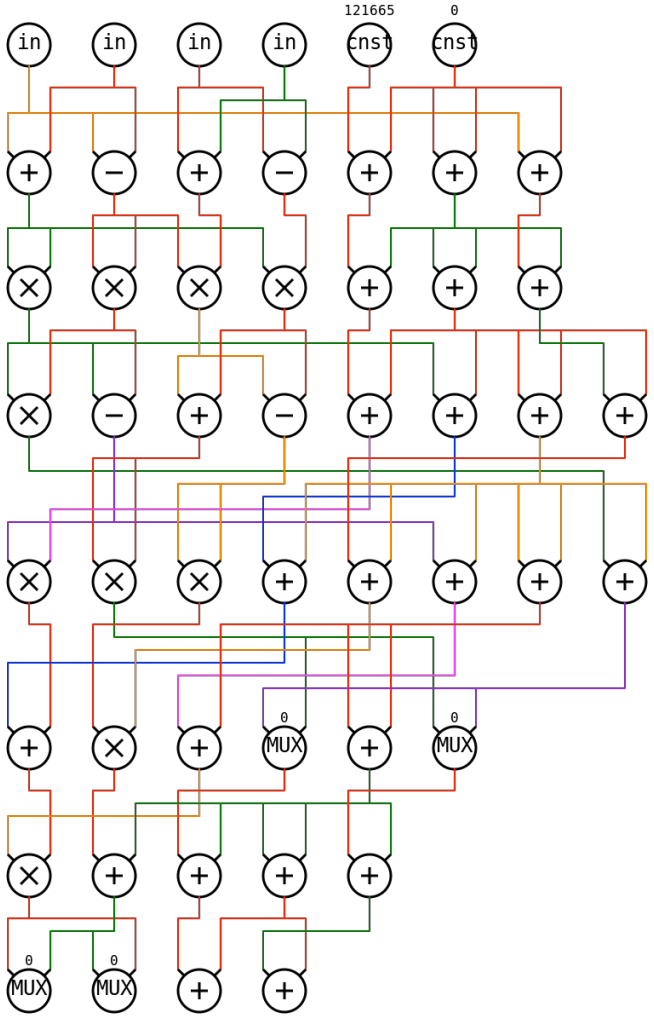


Zebra extracts parallelism

Execution step: layers are sequential, but gates can be executed in parallel.

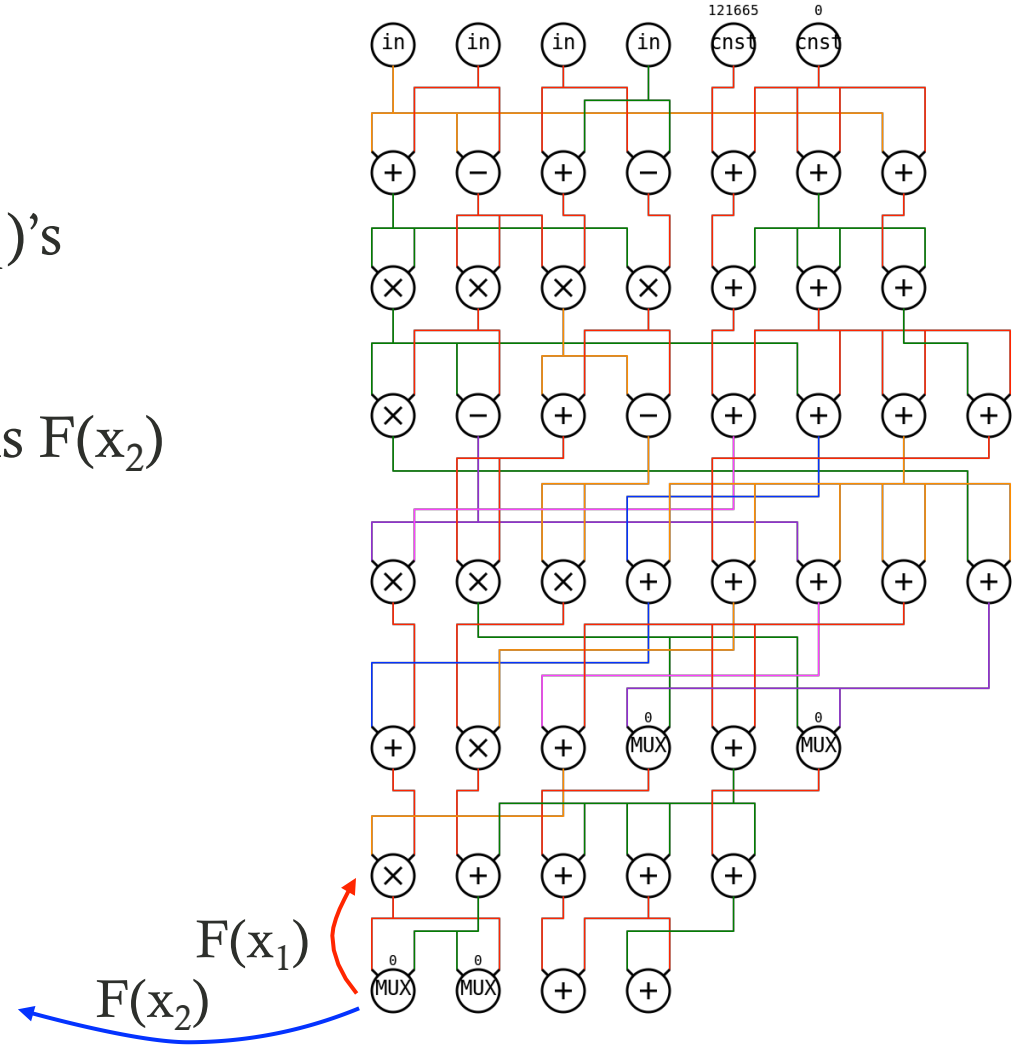
Proving step: can P and V parallelize the interaction?

- No. V must ask questions in order
- But. Parallelism is still available



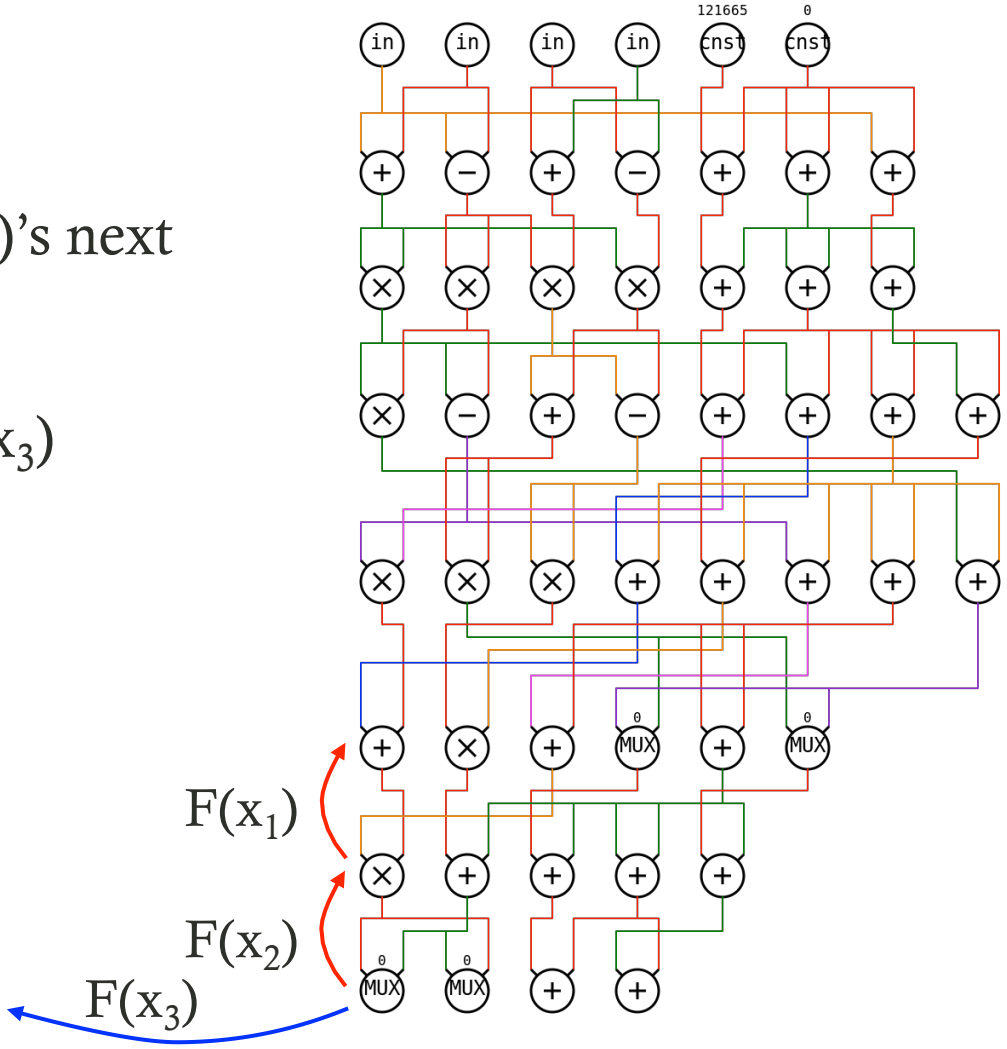
V questions P about $F(x_1)$'s output layer

Simultaneously, P returns $F(x_2)$

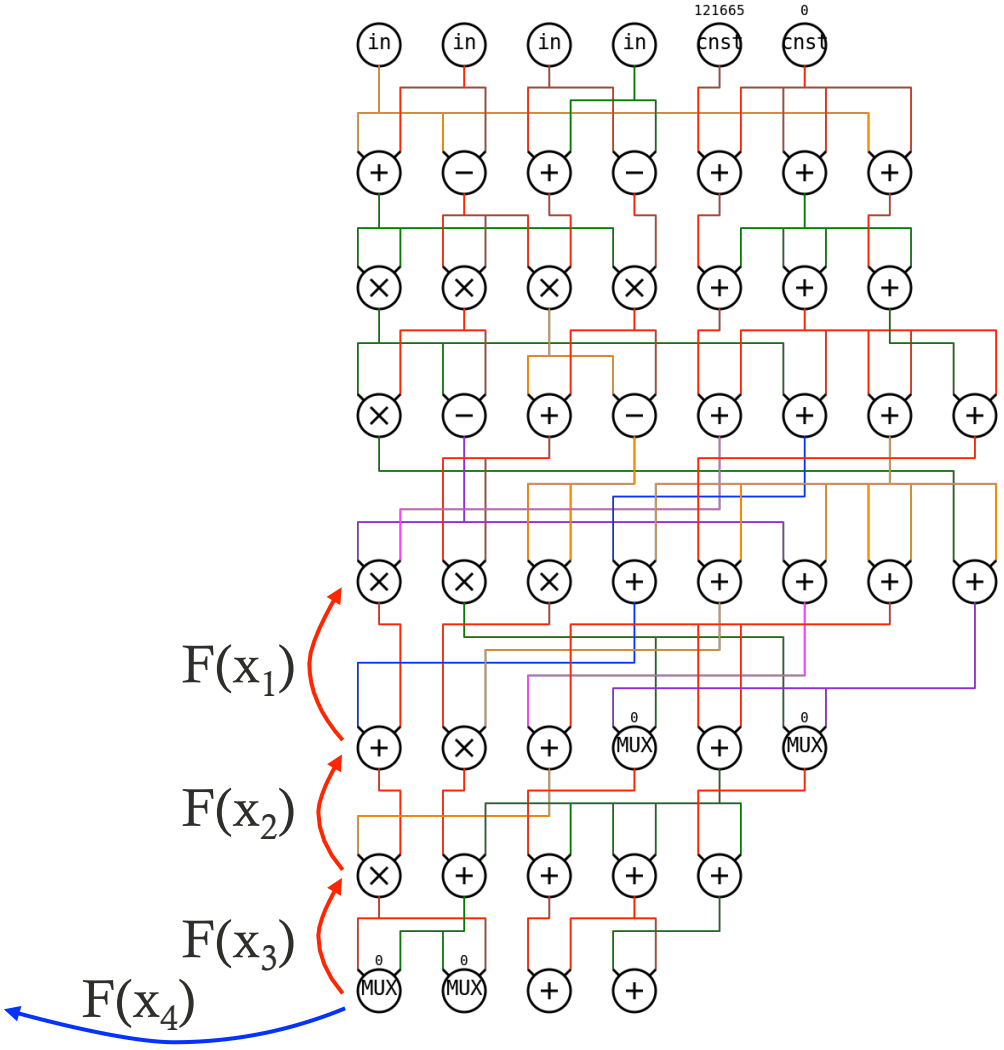


V questions P about $F(x_1)$'s next layer

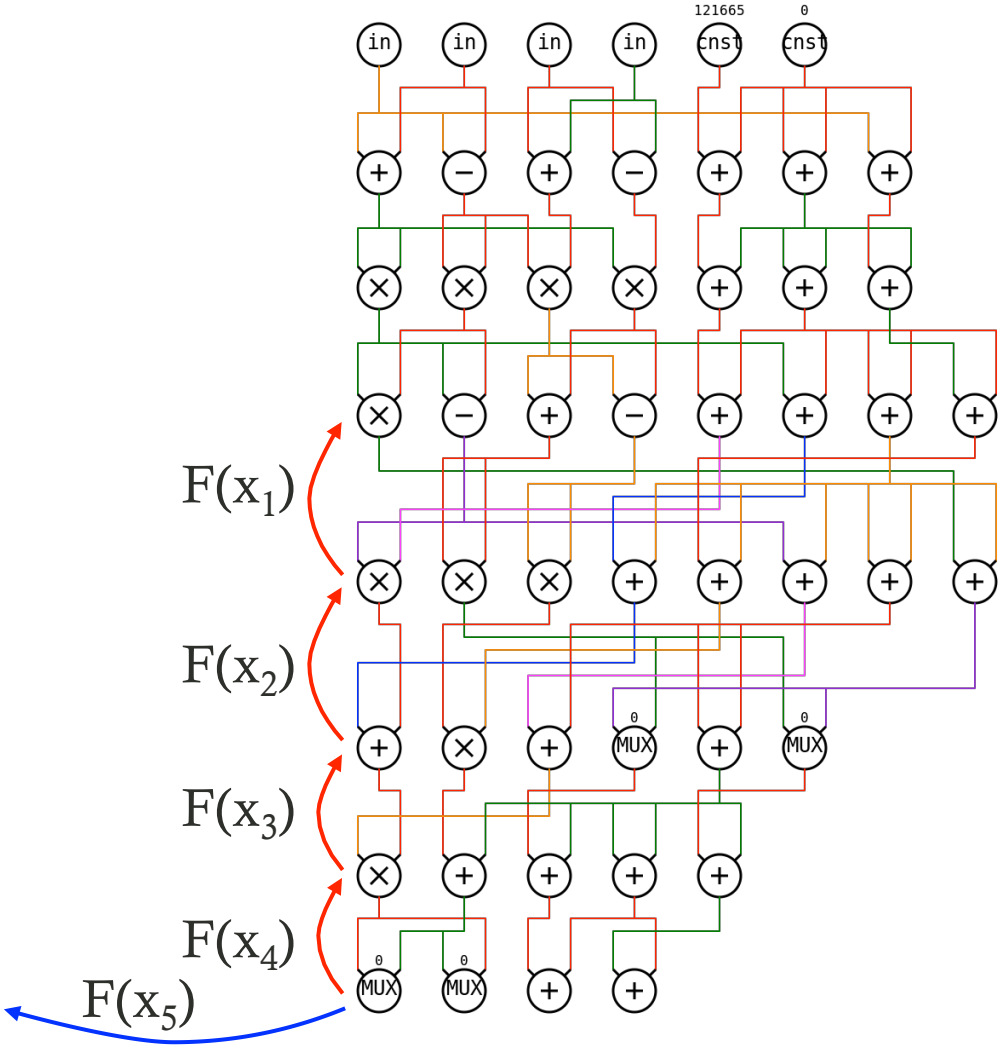
Meanwhile, P returns $F(x_3)$



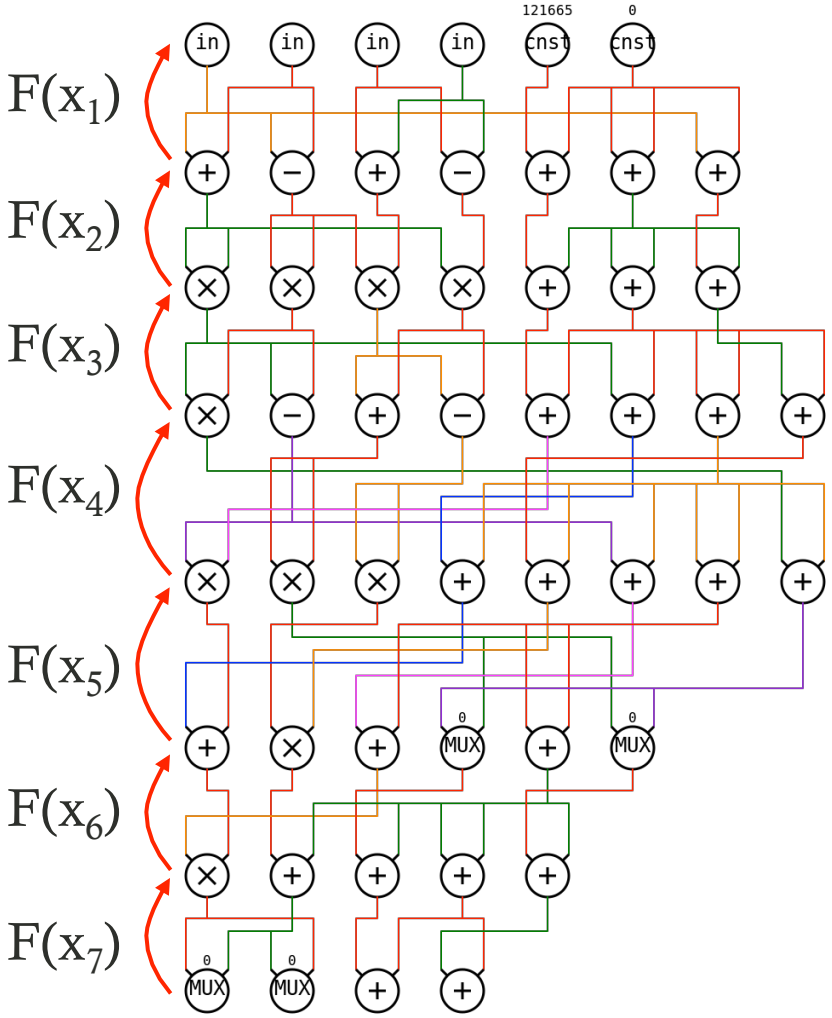
This process continues

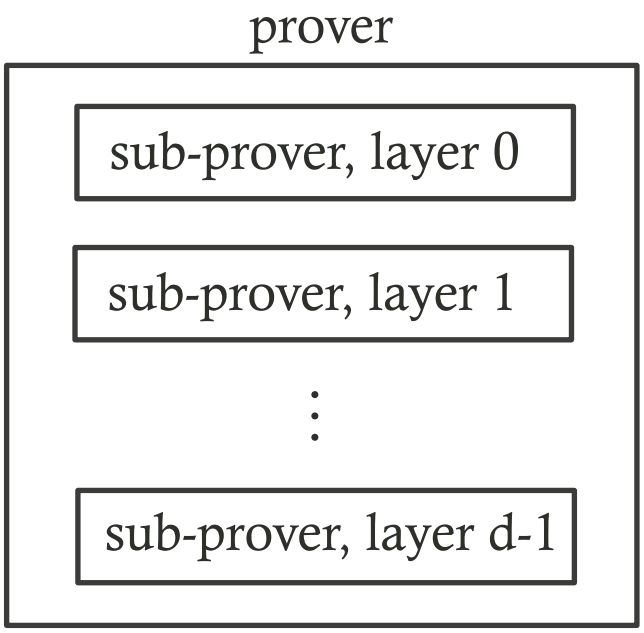


This process continues



This process continues until V and P are completing one proof in each time step.





This is nothing other than pipelining, a classic hardware technique.

It applies because layering organizes the work into stages.

There are other opportunities along these lines.

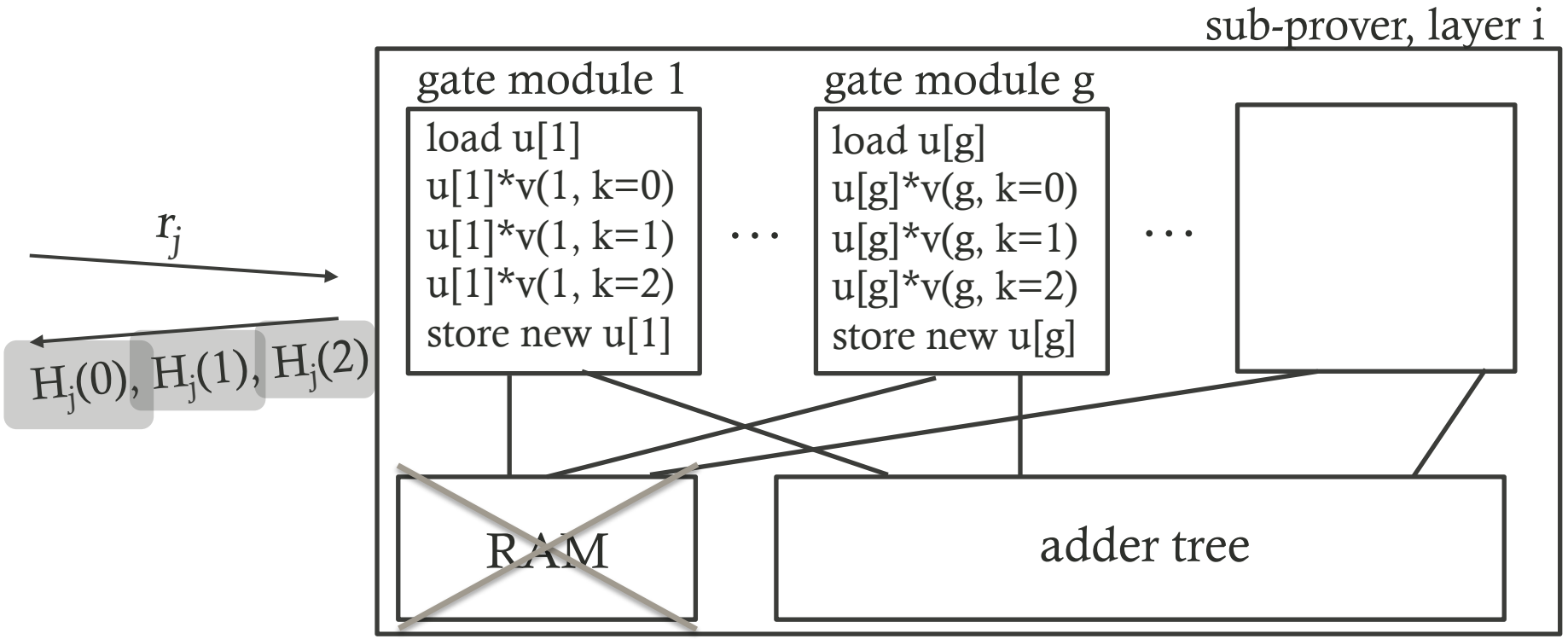
Sub-prover's obligation in round j of sum-check invocation: return $H_j(0)$, $H_j(1)$, $H_j(2)$, where

$$H_j(k) = \sum_{\text{gates } g} u_j(g) \cdot v_j(g, k)$$

$$u_{j+1}(g) = u_j(g) \cdot v_j(g, r_j)$$

```
for k in {0,1,2}:
  H[k] ← 0
  for all gates g:
    H[k] ← H[k] + u[g]*v(g,k)

for all gates g:
  u[g] ← u[g]*v(g,rj)
```



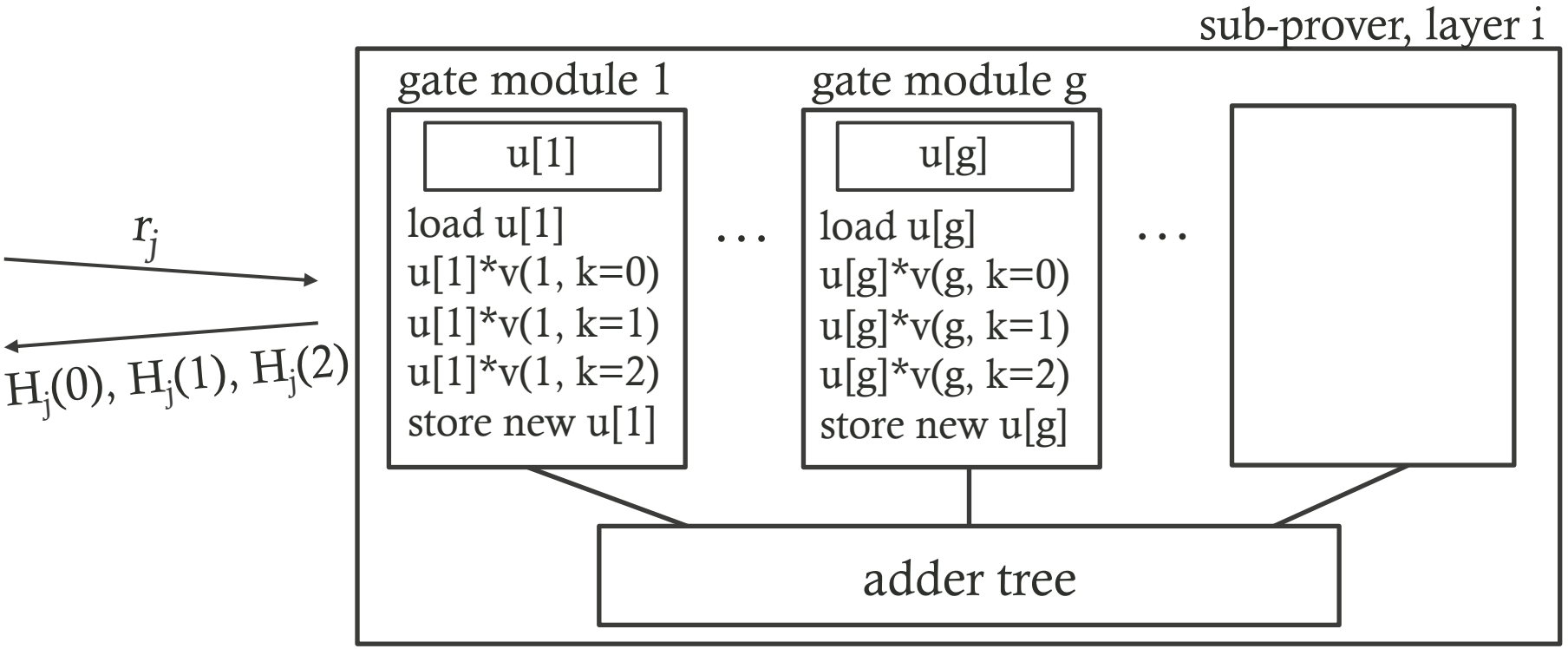
Sub-prover's obligation in round j of sum-check invocation: return $H_j(0)$, $H_j(1)$, $H_j(2)$, where

$$H_j(k) = \sum_{\text{gates } g} u_j(g) \cdot v_j(g, k)$$

$$u_{j+1}(g) = u_j(g) \cdot v_j(g, r_j)$$

```
for k in {0,1,2}:
  H[k] ← 0
  for all gates g:
    H[k] ← H[k] + u[g]*v(g,k)

for all gates g:
  u[g] ← u[g]*v(g,rj)
```



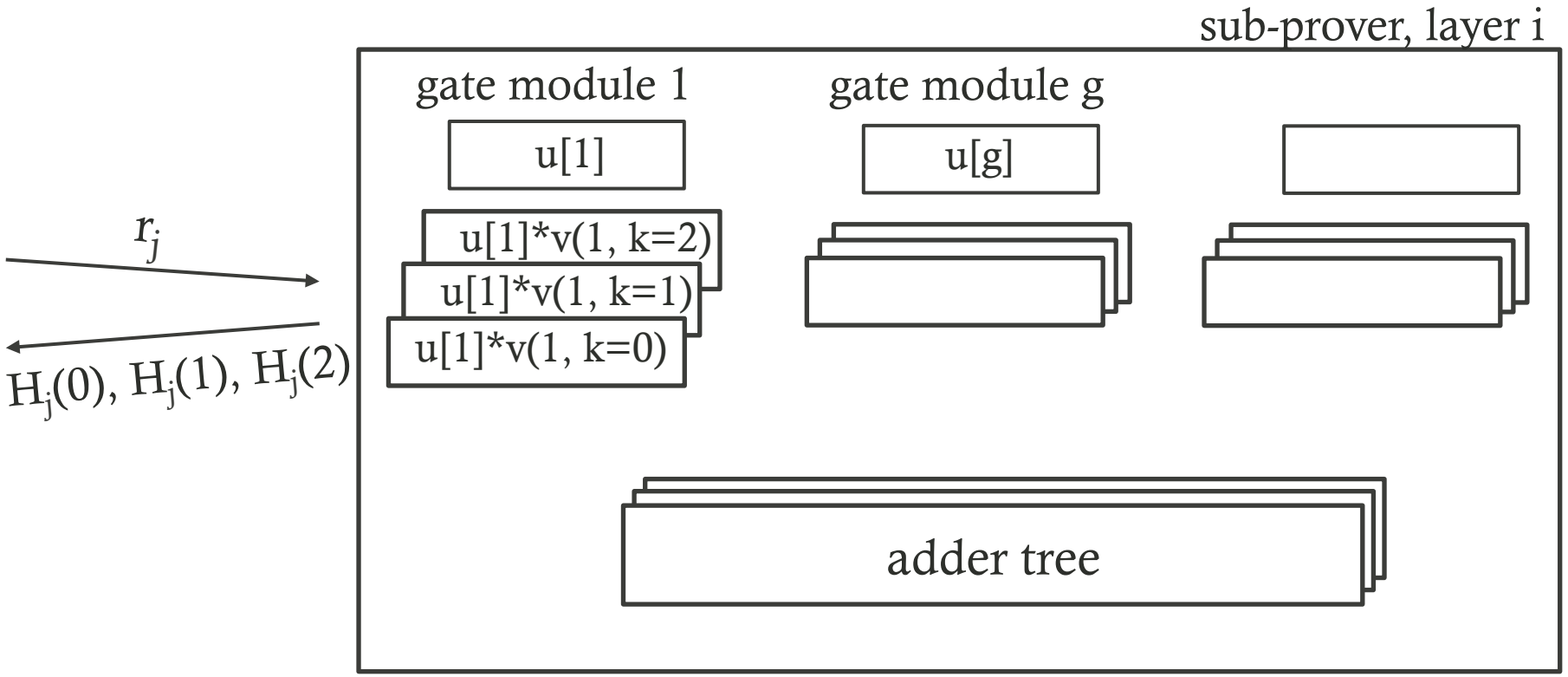
Sub-prover's obligation in round j of sum-check invocation: return $H_j(0)$, $H_j(1)$, $H_j(2)$, where

$$H_j(k) = \sum_{\text{gates } g} u_j(g) \cdot v_j(g, k)$$

$$u_{j+1}(g) = u_j(g) \cdot v_j(g, r_j)$$

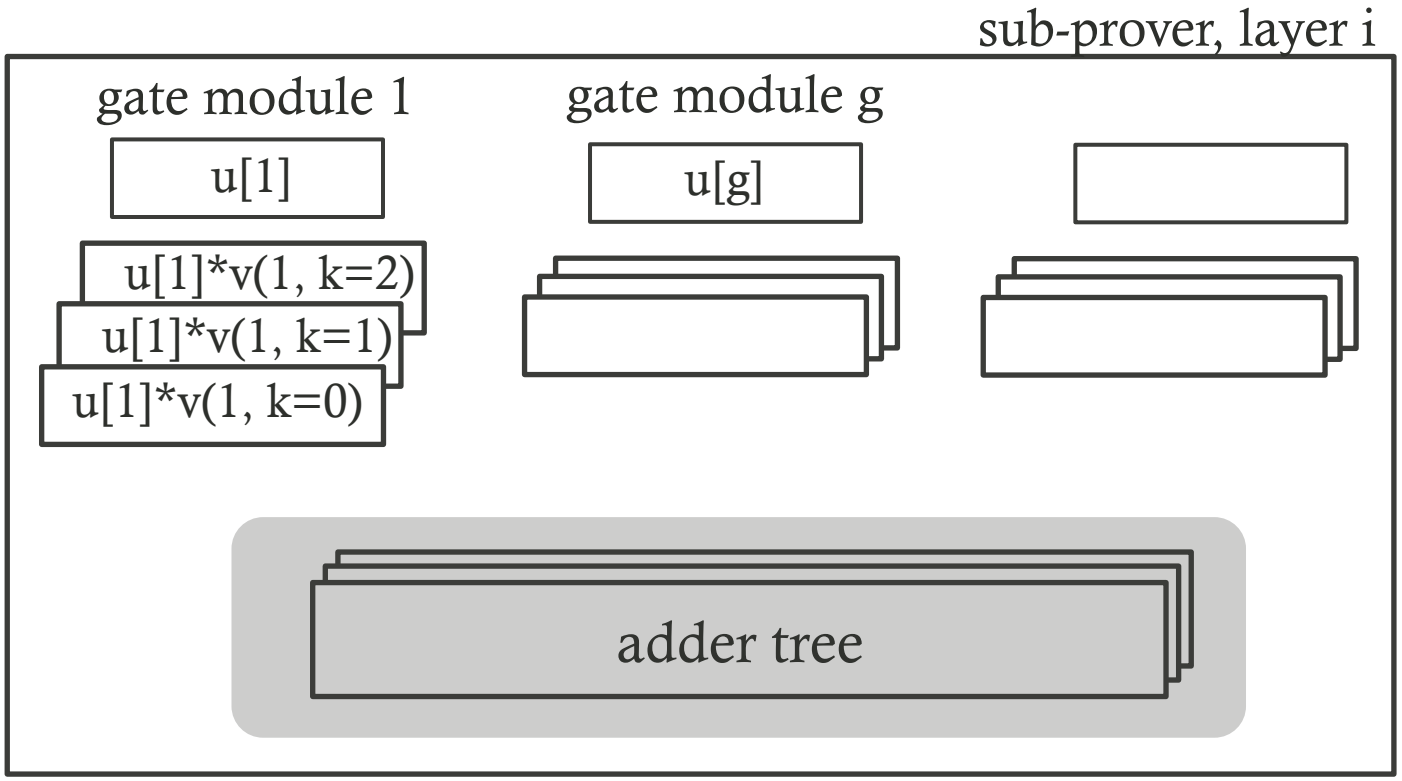
```
for k in {0,1,2}:
  H[k] ← 0
  for all gates g:
    H[k] ← H[k] + u[g]*v(g,k)

for all gates g:
  u[g] ← u[g]*v(g,rj)
```



Summary of Zebra's design approach:

- Extract parallelism
  - Pipelined proving, adder tree, gate proving, etc.
- Exploit locality: distribute state and control
  - Custom registers (no RAM): "data" wires are few and short
  - Latency-insensitive design: few "control" wires
- Reduce and reuse

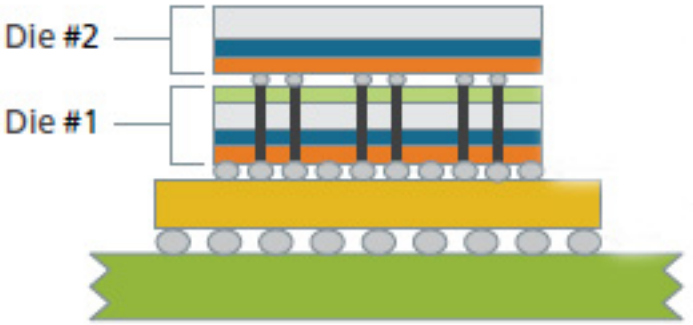

sub-prover, layer i
gate module 1
gate module g
u[1]
u[g]
u[1]*v(1, k=2)
u[1]*v(1, k=1)
u[1]*v(1, k=0)
adder tree

Summary of Zebra's design approach:

- Extract parallelism
  - Pipelined proving, adder tree, gate proving, etc.
- Exploit locality: distribute state and control
  - Custom registers (no RAM): "data" wires are few and short
  - Latency-insensitive design: few "control" wires
- Reuse and recycle
  - Recycle hardware circuitry for different tasks
  - Save energy by adding memoization to P
  - Reuse block designs; optimizations thus have high pay-off

Architectural and operational challenges for Zebra

1. Communication between V and P is high bandwidth
   - V and P on circuit board? Too much energy, circuit area
   - Zebra's response: use 3D packaging



2. Protocol requires input-independent precomputation
   - Zebra's response: amortize precomputations over many V-P pairs

3. Trusted storage would be prohibitive
   - Zebra's response: use untrusted storage, with auth-encryption

The implementation of Zebra includes:

- An arithmetic circuit to synthesizable Verilog compiler for P
  - Composes with existing C to arithmetic circuit compilers

- Two V implementations:
  - hardware (Verilog)
  - software (C++)

- Library to generate V's precomputations

- Verilog simulator extensions to model software or hardware V's interactions with P and with storage

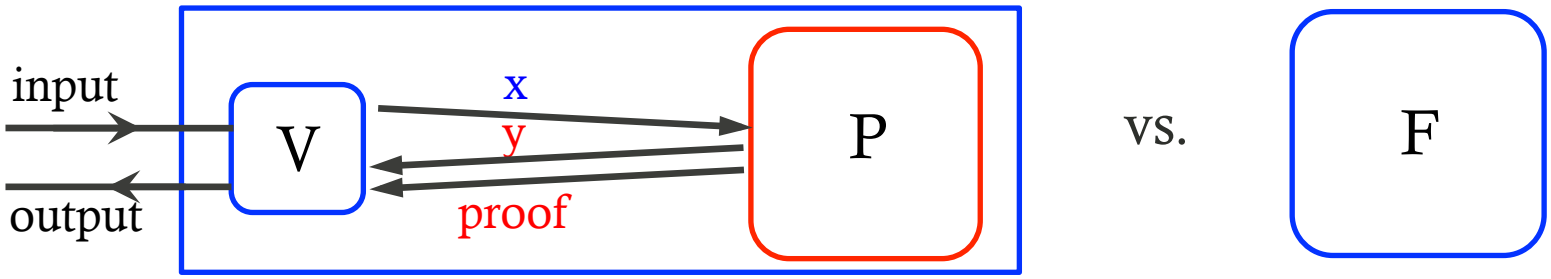
This implementation seemed to work great.

Existing implementations: 10 seconds per proof, at least

Zebra: 10^4 or 10^5 proofs per second

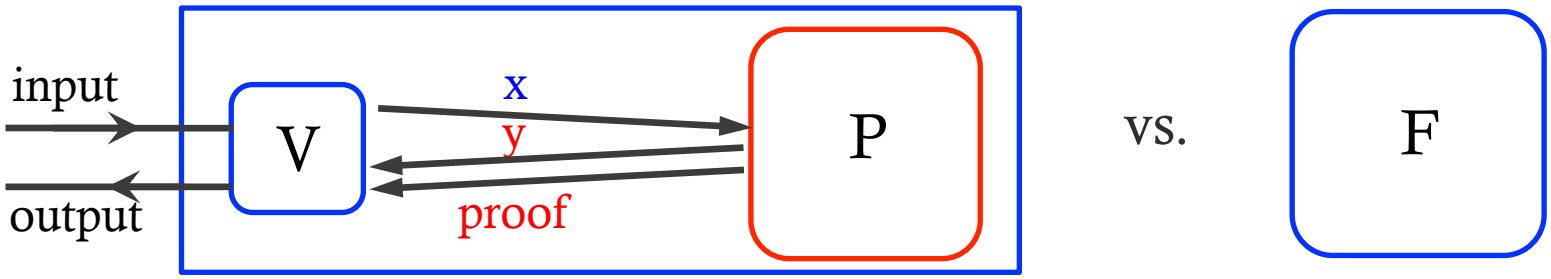
But that isn't a serious evaluation …

Evaluation method



- Baseline: direct implementation of F in same technology as V
- Metrics: energy, chip size per throughput (in paper)
- Assessed with circuit synthesis and simulation, published chip designs, and CMOS scaling models
  - Charge for V, P, communication; retrieving and decrypting precomputations; PRNG; operator communicating with V
- Constraints: trusted fab = 350 nm; untrusted fab = 7 nm; max chip area = 200 mm^2; max total power = 150 W

Evaluation method



- Baseline: direct implementation of F in same technology as V
- Metrics: energy, chip size per throughput (in paper)
- Assessed with circuit synthe
  designs, and CMOS scaling
  - Charge for V, P, communi
    precomputations; PRNG; operator communicating with V

1997	2017
350 nm	7 nm [TSMC]

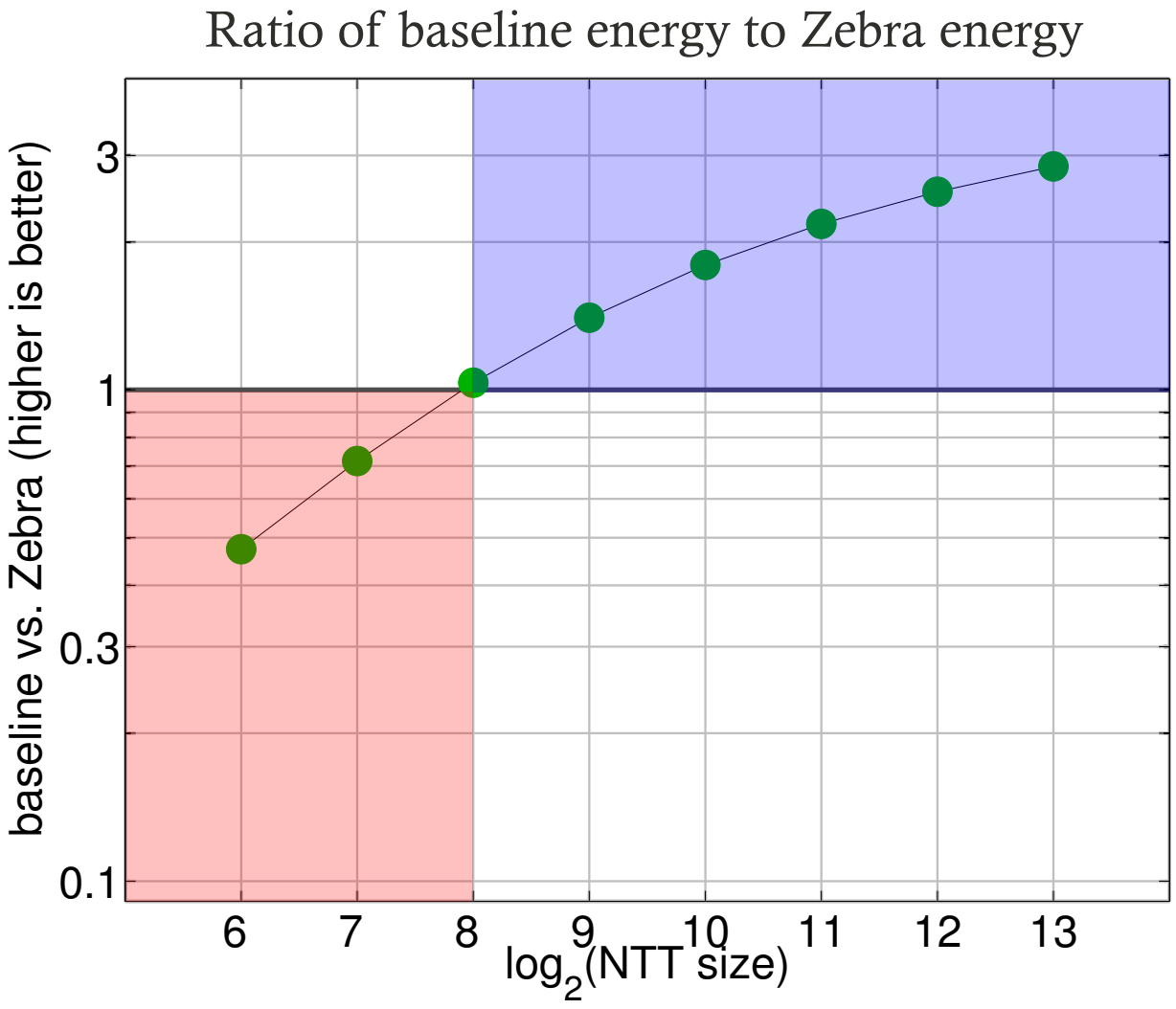
- Constraints: trusted fab = 350 nm; untrusted fab = 7 nm; max chip area = 200 mm^2; max total power = 150 W

Application #1: number theoretic transform

NTT: a Fourier transform over $\mathbb{F}_p$

Used in signal processing, computer algebra, etc.

Application #1: number theoretic transform

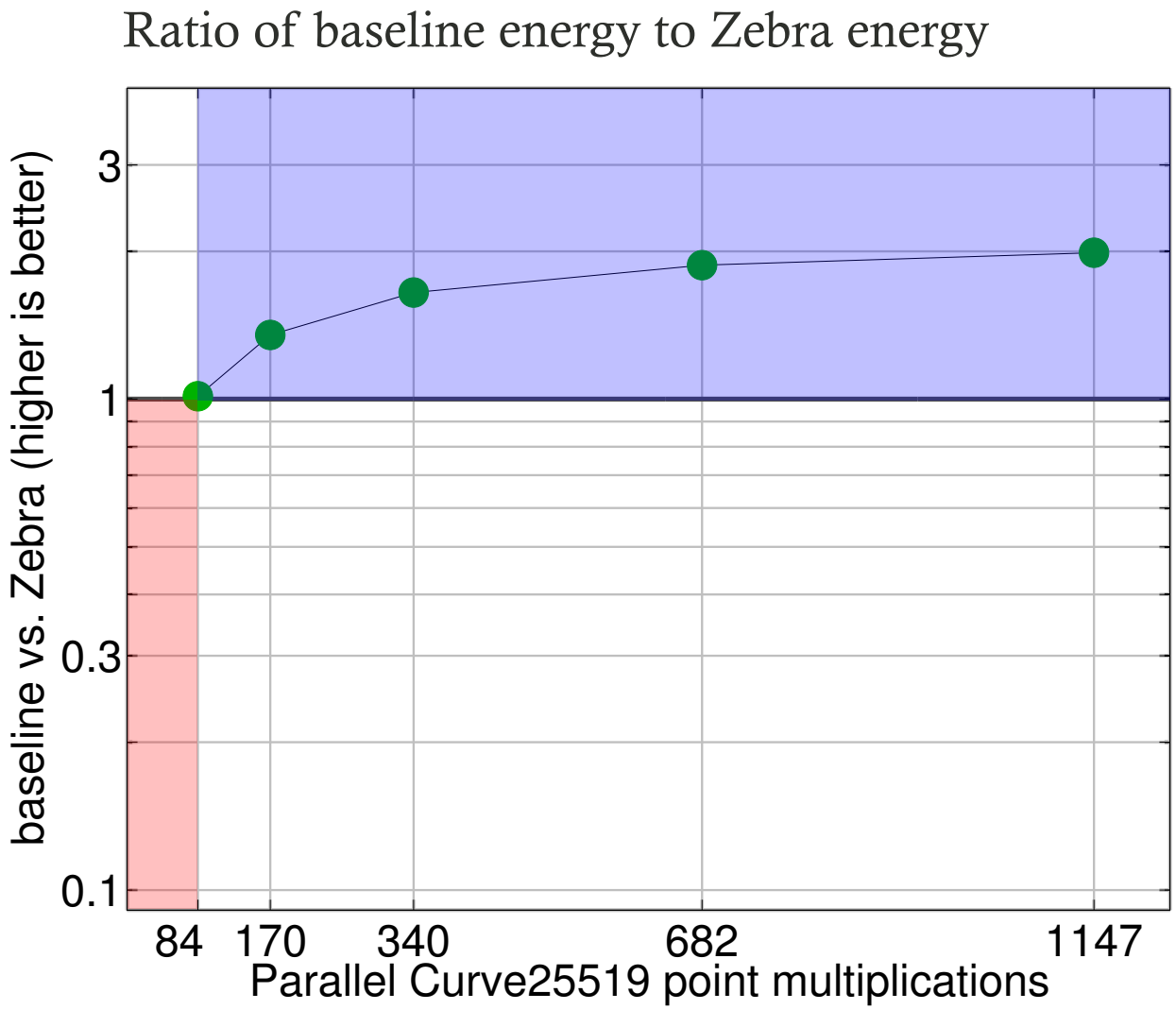


Application #2: Curve25519 point multiplication

Curve25519: a commonly-used elliptic curve

Point multiplication: primitive used for ECDH

Application #2: Curve25519 point multiplication



(1) Zebra: a system that saves costs

(2) … sometimes

Summary of Zebra's applicability:

restriction of the interactive proof (IP) setup

1. Computation F must have a layered, shallow, deterministic AC
2. Need wide gap between (fast) fab for P and (trusted) fab for V
3. Computation F must be relatively large for V to save work
4. Computation F must be efficient as an arithmetic circuit (AC)
5. Must amortize precomputations over many chips

Why did we build Zebra atop IPs instead of arguments?

Because argument protocols seem unfriendly to hardware:

Design principle	**interactive proofs** [GKR08, CMT12, VSBW13]	**arguments** (interactive, SNARK, CS proof, etc.) [GGPR12, PGHR13, SBVBPW13, BCTV14]
Extract parallelism	✓	✓
Exploit locality	✓	✗
Reduce and reuse	✓	✗

In arguments, P computes over entire AC at once ⟶ need RAM

P does crypto for every gate in AC ⟶ special crypto circuits needed

We hope these issues are surmountable!

Reality check on the restrictions:

applies to interactive proofs (IPs) but not arguments

1. Computation F must have a layered, shallow, deterministic AC

2. Need wide gap between (fast) fab for P and (trusted) fab for V

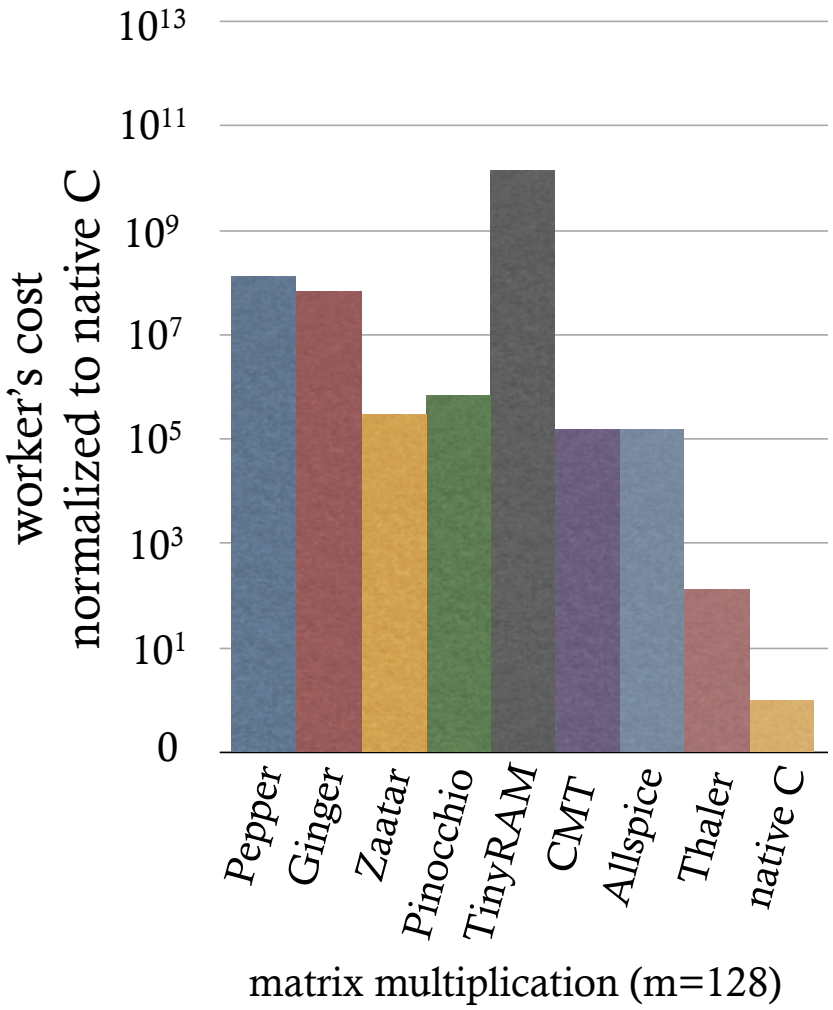
3. Computation F must be relatively large for V to save work

4. Computation F must be efficient as an arithmetic circuit (AC)

5. Must amortize precomputations over many chips

common to all implementations of probabilistic proofs

A limitation that is endemic to the area:
Need wide gap between (fast) fab for P and (trusted) fab for V



Limitations that are endemic to the area:

Computation F must be relatively large for V to save work

Computation F must be efficient as an arithmetic circuit

- Example: libsnark's [BCTV14] optimized implementation of GGPR/Pinocchio [GGPR12, PGHR13]. Great work, but:
- Verification time: 6 ms + (|x| + |y|)·3 μs on 2.7 Ghz CPU
- That time is >16 million CPU ops, which is a break-even point
- libsnark handles ≤ 16 million gates (with 32 GB of RAM), so to break even, F also needs on average CPU_ops/AC_gate > 1.
  - Example: addition over $\mathbb{F}_p$ instead of over fixed-width integers

Built probabilistic proof protocols amortize precomputations*

System	amortize precomputation over	size of advice
Zebra	multiple V-P pairs	short
Allspice [VSBW13]	over a batch of instances of a given F	short
Bootstrapped SNARKs [BCTV14a, CTV15]	over all computations	long
BCTV [BCTV14b]	over all computations of the same length	long
Pinocchio [PGHR13]	over all future uses of a given F	long
Pepper [SMBW12], Ginger [SVPBBW12], Zaatar [SBVBPW13]	over a batch of instances of a given F	long

*Exception: CMT [CMT12] applied to highly regular arithmetic circuits

Lessons (re)learned:

- Do careful feasibility studies first!
- Hardware is a powerful tool for acceleration …
  - … but only if data flows are amenable
  - Theory of computation versus application of physics
- General-purpose verifiable computation and succinct arguments are still far from practical

Summary and take-aways

- Verifiable ASICs: a new approach to building trustworthy hardware under a strong threat model
- First hardware design for a probabilistic proof protocol; first work to capture cost of prover, verifier together
- Improves performance compared to trusted baseline
- Improvement compared to baseline is modest
- Applicability is limited
  - Amortization, arithmetic circuits, "big enough" computations, large gap between trusted and untrusted technology, etc.
- Zebra is plausibly deployable (!), but work remains for this area

`http://www.pepper-project.org`