Verifiable ASICs: trustworthy hardware with untrusted components

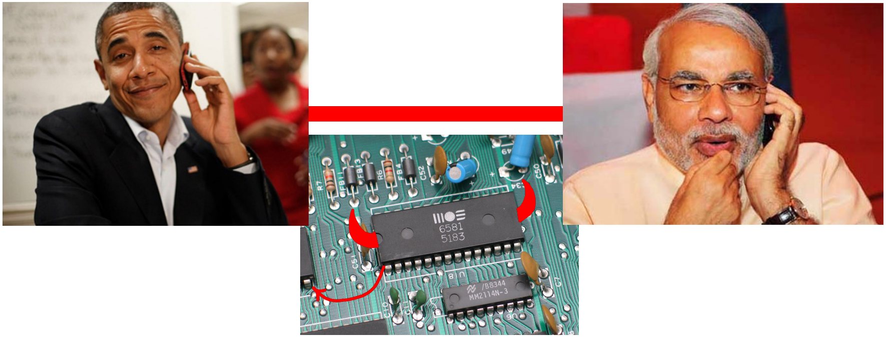
Riad S. Wahby[°][⋆], Max Howald[†][⋆],
Siddharth Garg[⋆], abhi shelat[‡], and Michael Walfish[⋆]

[°]Stanford University
[⋆]New York University
[†]The Cooper Union
[‡]The University of Virginia

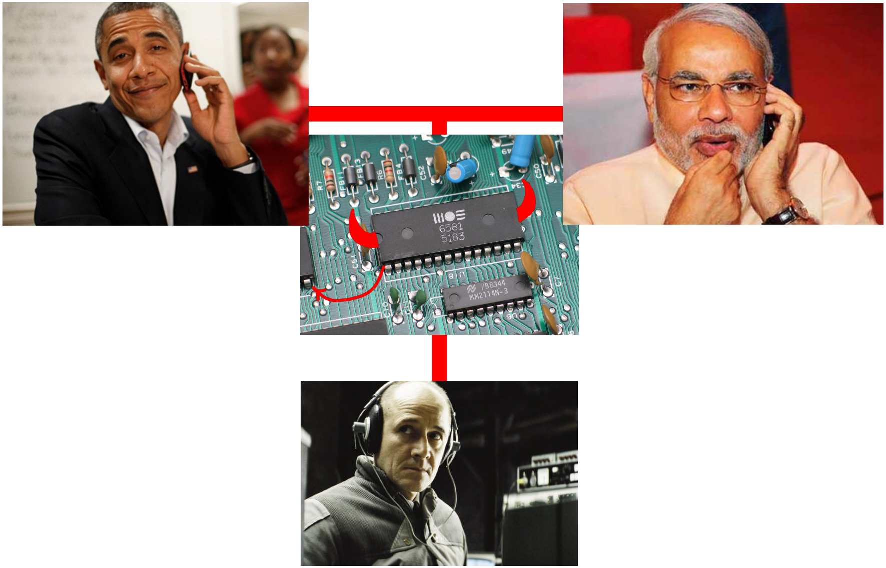
May 25th, 2016

Untrusted manufacturers can craft hardware Trojans

Untrusted manufacturers can craft hardware Trojans



Untrusted manufacturers can craft hardware Trojans



Untrusted manufacturers can craft hardware Trojans

Untrusted manufacturers can craft hardware Trojans

Trusted fabrication is not a panacea:

- ✗ Only 5 countries have cutting-edge fabs on-shore
- ✗ Building a new fab takes \$\$\$\$\$\$, years of R&D
- ✗ An old fab could mean $10^8\times$ performance hit accounting for speed, chip area, and energy

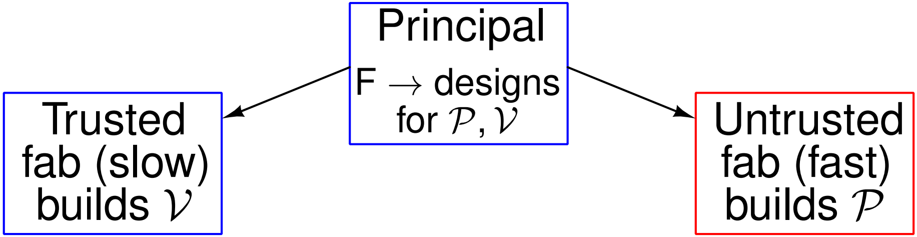
Can we get trust more cheaply?

Can we build Verifiable ASICs?

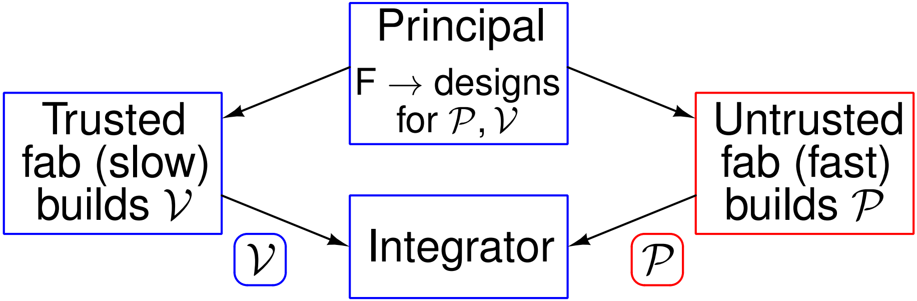
Principal

F $\rightarrow$ designs for $\mathcal{P}, \mathcal{V}$

Can we build Verifiable ASICs?

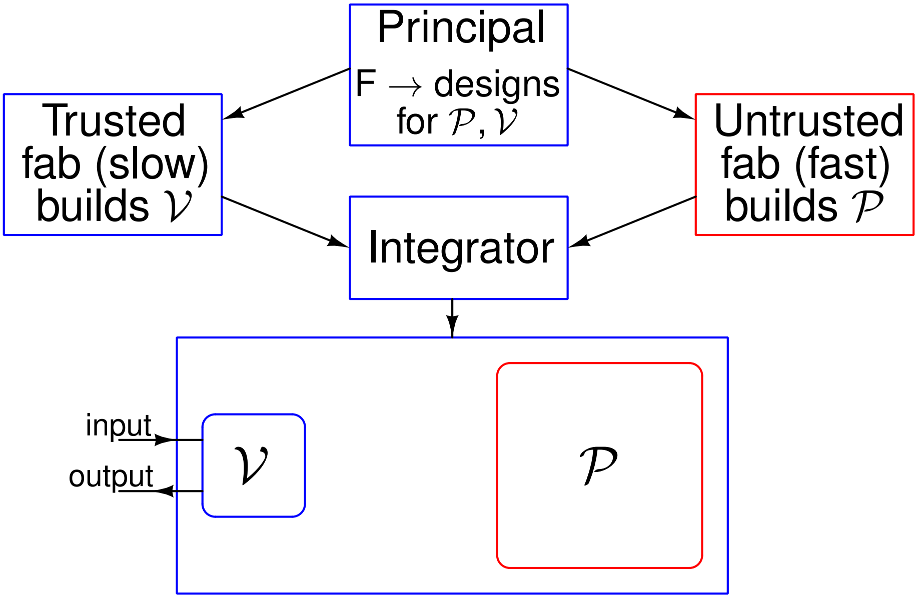

Principal
F $\rightarrow$ designs for $\mathcal{P}, \mathcal{V}$
Trusted fab (slow) builds $\mathcal{V}$
Untrusted fab (fast) builds $\mathcal{P}$

Can we build Verifiable ASICs?



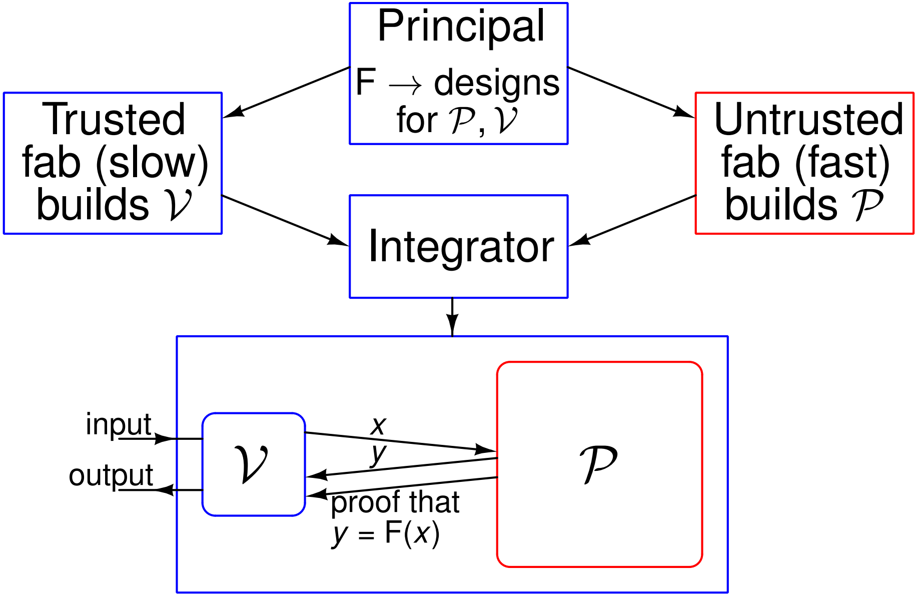
Principal
F $\rightarrow$ designs for $\mathcal{P}, \mathcal{V}$
Trusted fab (slow) builds $\mathcal{V}$
Untrusted fab (fast) builds $\mathcal{P}$
Integrator
$\mathcal{V}$
$\mathcal{P}$

Can we build Verifiable ASICs?



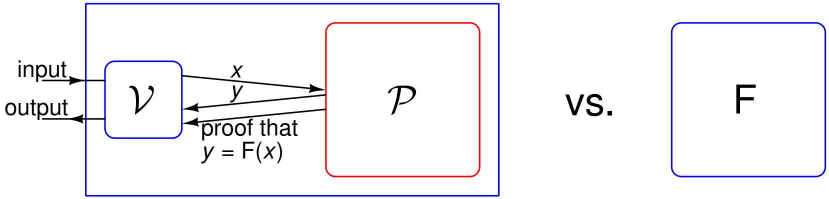
Principal
F $\rightarrow$ designs for $\mathcal{P}, \mathcal{V}$
Trusted fab (slow) builds $\mathcal{V}$
Untrusted fab (fast) builds $\mathcal{P}$
Integrator
input
output
$\mathcal{V}$
$\mathcal{P}$

Can we build Verifiable ASICs?



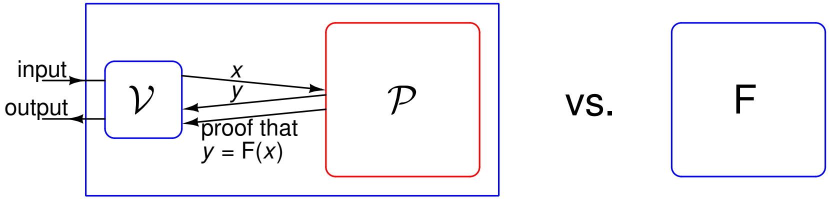
Principal
F $\rightarrow$ designs for $\mathcal{P}$, $\mathcal{V}$
Trusted fab (slow) builds $\mathcal{V}$
Untrusted fab (fast) builds $\mathcal{P}$
Integrator
input
output
$\mathcal{V}$
x
y
proof that $y = F(x)$
$\mathcal{P}$

Can we build Verifiable ASICs?



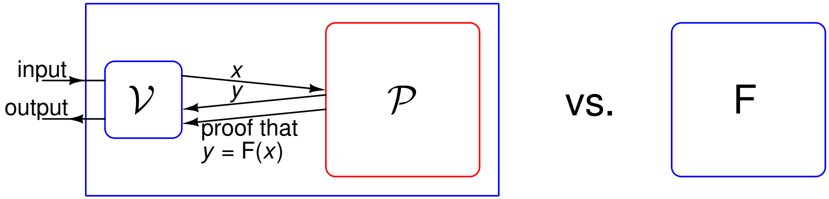
- Makes sense if $\mathcal{V} + \mathcal{P}$ are cheaper than trusted F

Can we build Verifiable ASICs?



- Makes sense if $\mathcal{V} + \mathcal{P}$ are cheaper than trusted F
- Reasons for hope:
  - running time of $\mathcal{V} <$ running time of F (asymptotically)
  - speed of cutting-edge fab might offset $\mathcal{P}$'s overheads

Can we build Verifiable ASICs?



- Makes sense if $\mathcal{V} + \mathcal{P}$ are cheaper than trusted F
- Reasons for hope:
  - running time of $\mathcal{V} <$ running time of F (asymptotically)
  - speed of cutting-edge fab might offset $\mathcal{P}$'s overheads
- Challenges remain:
  - Hardware issues: energy, chip area
  - Need physically realizable circuit design
  - $\mathcal{V}$ needs to save work at plausible computation sizes

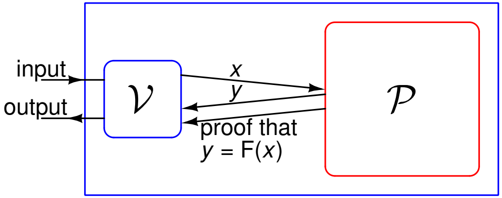
Zebra: a hardware design that saves costs

A **qualified** success

Zebra: a hardware design that saves costs. . .

. . . sometimes.

Probabilistic proof systems, briefly

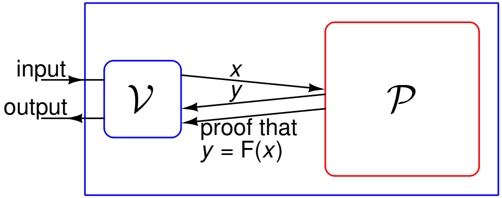


F must be expressed as an arithmetic circuit (AC)

generalized boolean circuit over $\mathbb{F}_p$

$\wedge \rightarrow \times \qquad \vee \rightarrow +$

Probabilistic proof systems, briefly

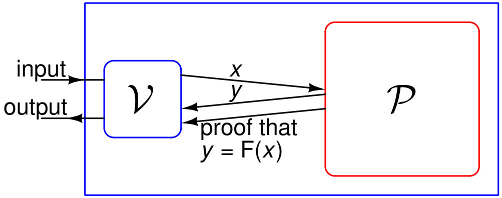


F must be expressed as an arithmetic circuit (AC)

AC satisfiable $\iff$ F was executed correctly

$\mathcal{P}$ convinces $\mathcal{V}$ that the AC is satisfiable

Probabilistic proof systems, briefly



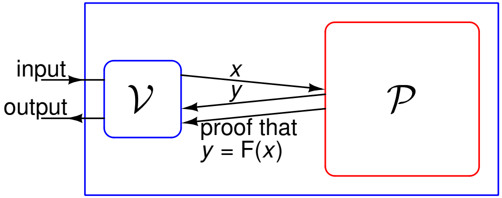
Arguments [GGPR13, SBVBPW13, PGHR13, BCTV14]

e.g., Zaatar, Pinocchio, libsnark

IPs [GKR08, CMT12, VSBW13]

e.g., Muggles, CMT, Allspice

Probabilistic proof systems, briefly



Arguments [GGPR13, SBVBPW13, PGHR13, BCTV14]

e.g., Zaatar, Pinocchio, libsnark

+ F with RAM, complex control flow
+ Little $\mathcal{V}$-$\mathcal{P}$ communication

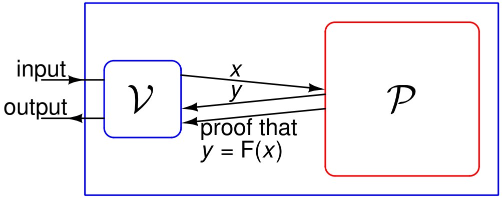
IPs [GKR08, CMT12, VSBW13]

e.g., Muggles, CMT, Allspice

– “Quasi–straight line” F

– Lots of $\mathcal{V}$-$\mathcal{P}$ communication

Probabilistic proof systems, briefly



Arguments [GGPR13, SBVBPW13, PGHR13, BCTV14]

e.g., Zaatar, Pinocchio, libsnark

+ F with RAM, complex control flow
+ Little $\mathcal{V}$-$\mathcal{P}$ communication

Unsuited to hardware implementation ✗

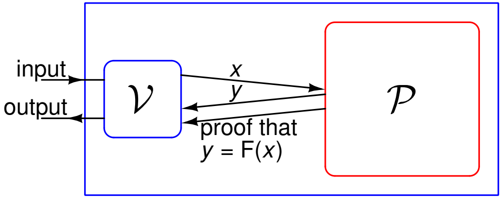
IPs [GKR08, CMT12, VSBW13]

e.g., Muggles, CMT, Allspice

– “Quasi–straight line” F

– Lots of $\mathcal{V}$-$\mathcal{P}$ communication

Probabilistic proof systems, briefly



Arguments [GGPR13, SBVBPW13, PGHR13, BCTV14]

e.g., Zaatar, Pinocchio, libsnark

+ F with RAM, complex control flow
+ Little $\mathcal{V}$-$\mathcal{P}$ communication

Unsuited to hardware implementation ✗

IPs [GKR08, CMT12, VSBW13]

e.g., Muggles, CMT, Allspice

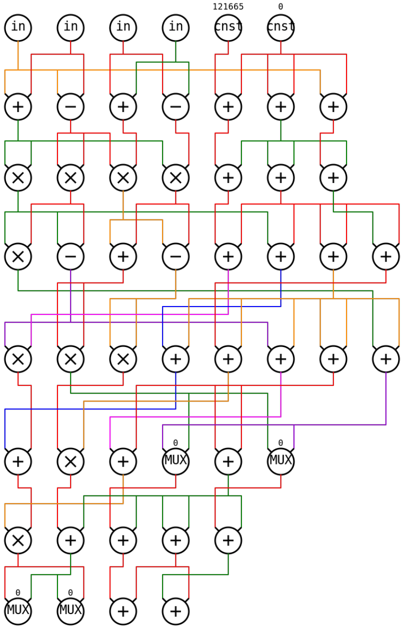
– "Quasi–straight line" F
– Lots of $\mathcal{V}$-$\mathcal{P}$ communication

Suited to hardware implementation ✓

Zebra builds on IPs of GKR [GKR08, CMT12, VSBW13]

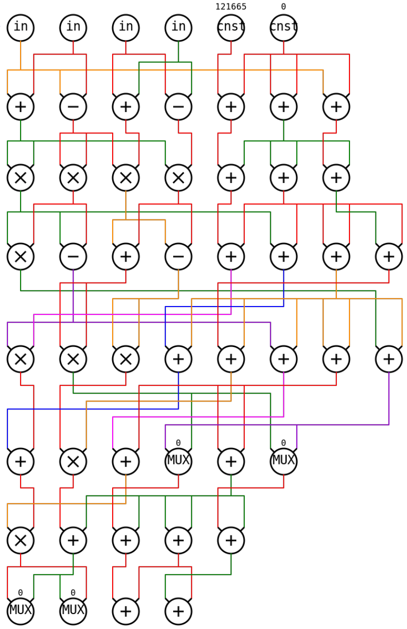
F must be expressed as a *layered* arithmetic circuit.

Note: this is an abstraction of F, *not* a physical circuit!



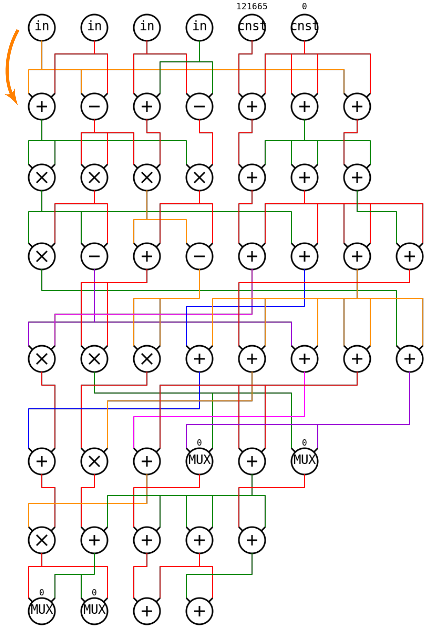
Zebra builds on IPs of GKR [GKR08, CMT12, VSBW13]

1. $\mathcal{V}$ sends inputs



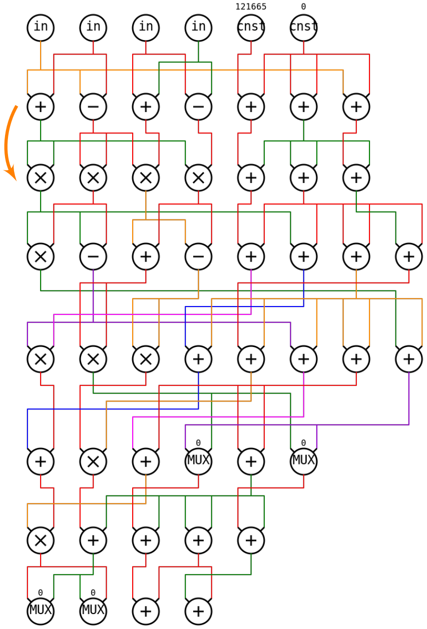
Zebra builds on IPs of GKR [GKR08, CMT12, VSBW13]

1. $\mathcal{V}$ sends inputs
2. $\mathcal{P}$ evaluates circuit



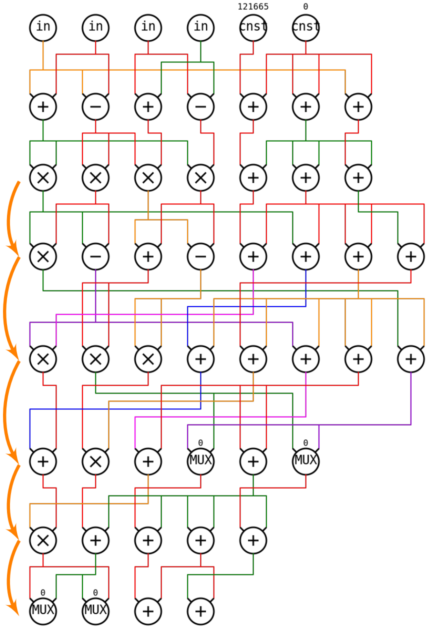
Zebra builds on IPs of GKR [GKR08, CMT12, VSBW13]

1. $\mathcal{V}$ sends inputs
2. $\mathcal{P}$ evaluates circuit



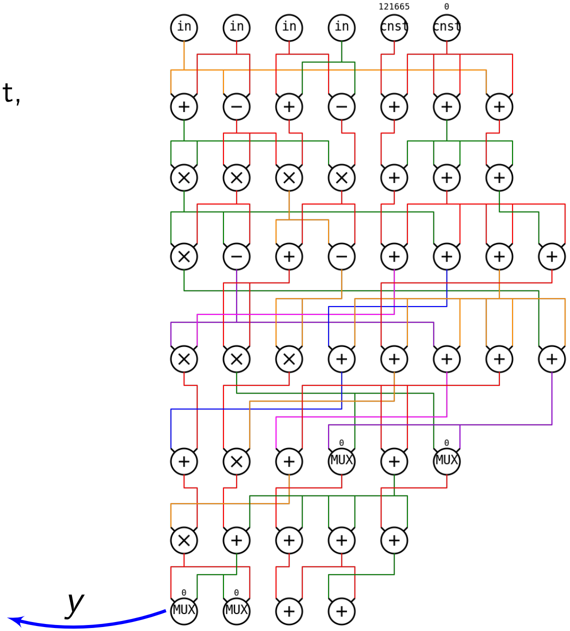
Zebra builds on IPs of GKR [GKR08, CMT12, VSBW13]

1. $\mathcal{V}$ sends inputs
2. $\mathcal{P}$ evaluates circuit



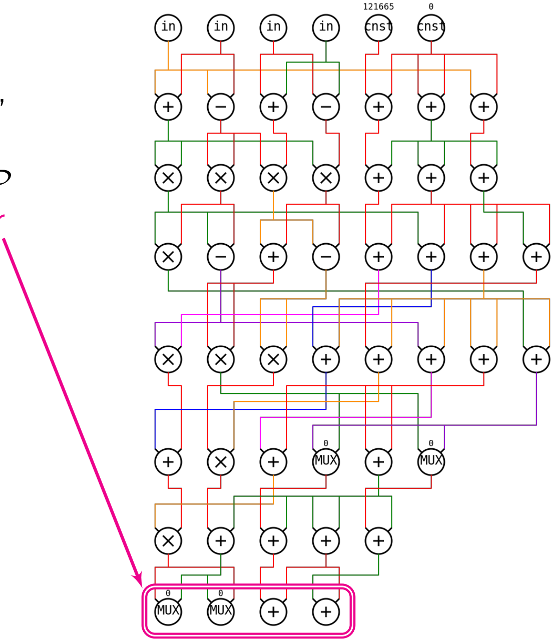
Zebra builds on IPs of GKR [GKR08, CMT12, VSBW13]

1. $\mathcal{V}$ sends inputs
2. $\mathcal{P}$ evaluates circuit, returns output y



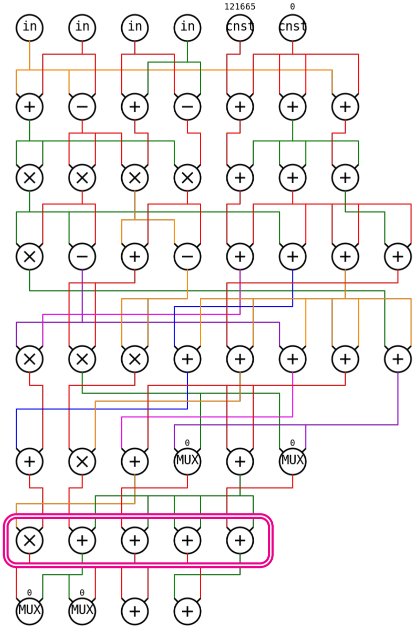
Zebra builds on IPs of GKR [GKR08, CMT12, VSBW13]

1. $\mathcal{V}$ sends inputs
2. $\mathcal{P}$ evaluates circuit, returns output y
3. $\mathcal{V}$ cross-examines $\mathcal{P}$ about the last layer



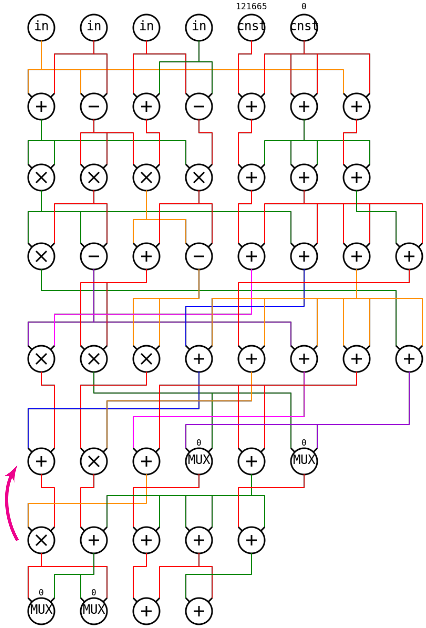
Zebra builds on IPs of GKR [GKR08, CMT12, VSBW13]

1. $\mathcal{V}$ sends inputs
2. $\mathcal{P}$ evaluates circuit, returns output y
3. $\mathcal{V}$ cross-examines $\mathcal{P}$ about the last layer, ends up with claim about second-last layer



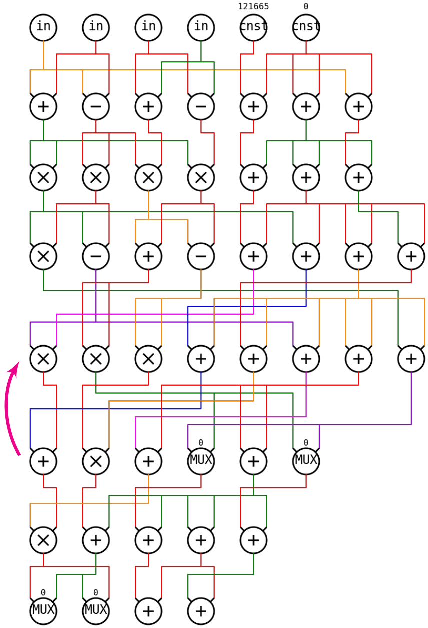
Zebra builds on IPs of GKR [GKR08, CMT12, VSBW13]

1. $\mathcal{V}$ sends inputs
2. $\mathcal{P}$ evaluates circuit, returns output y
3. $\mathcal{V}$ cross-examines $\mathcal{P}$ about the last layer, ends up with claim about second-last layer
4. $\mathcal{V}$ iterates



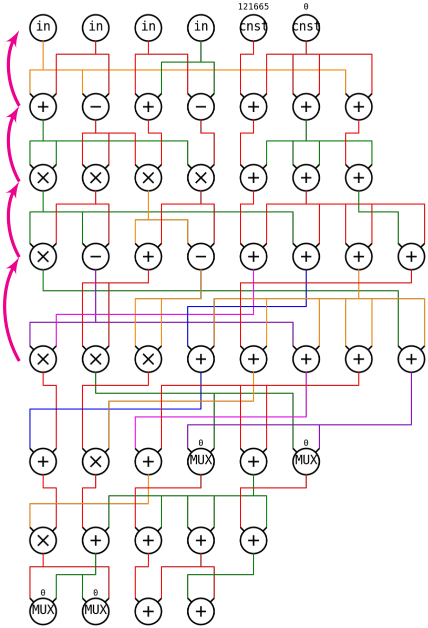
Zebra builds on IPs of GKR [GKR08, CMT12, VSBW13]

1. $\mathcal{V}$ sends inputs
2. $\mathcal{P}$ evaluates circuit, returns output y
3. $\mathcal{V}$ cross-examines $\mathcal{P}$ about the last layer, ends up with claim about second-last layer
4. $\mathcal{V}$ iterates



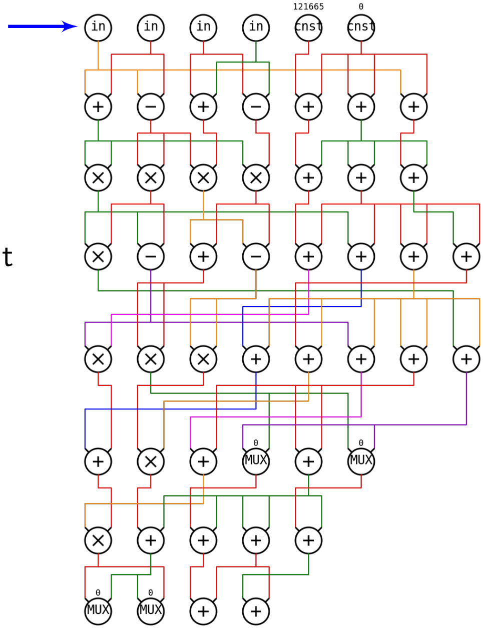
Zebra builds on IPs of GKR [GKR08, CMT12, VSBW13]

1. $\mathcal{V}$ sends inputs
2. $\mathcal{P}$ evaluates circuit, returns output y
3. $\mathcal{V}$ cross-examines $\mathcal{P}$ about the last layer, ends up with claim about second-last layer
4. $\mathcal{V}$ iterates



Zebra builds on IPs of GKR [GKR08, CMT12, VSBW13]

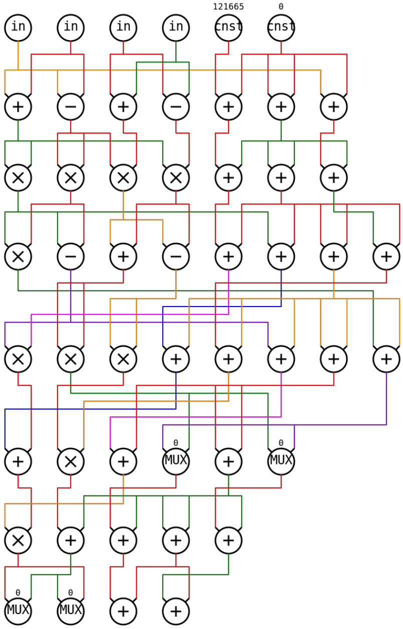
1. $\mathcal{V}$ sends inputs
2. $\mathcal{P}$ evaluates circuit, returns output y
3. $\mathcal{V}$ cross-examines $\mathcal{P}$ about the last layer, ends up with claim about second-last layer
4. $\mathcal{V}$ iterates, ends up with claim about inputs



Zebra builds on IPs of GKR [GKR08, CMT12, VSBW13]

1. $\mathcal{V}$ sends inputs
2. $\mathcal{P}$ evaluates circuit, returns output y
3. $\mathcal{V}$ cross-examines $\mathcal{P}$ about the last layer, ends up with claim about second-last layer
4. $\mathcal{V}$ iterates, ends up with claim about inputs
5. $\mathcal{V}$ checks consistency with the inputs

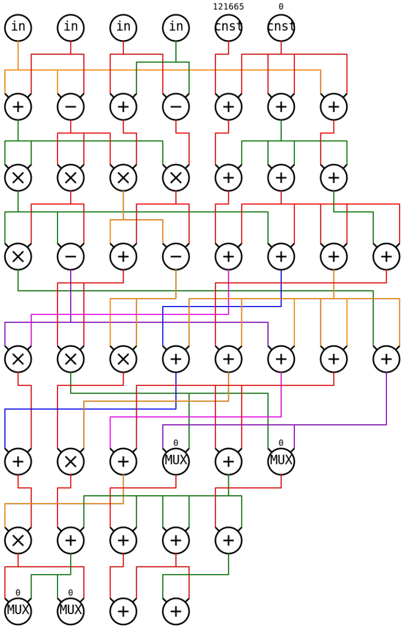
$\mathcal{V}$'s work $\approx O(\text{depth} \cdot \log \text{width})$, so it saves work when width $\gg$ depth



Can we parallelize this interaction?

Can $\mathcal{V}$ and $\mathcal{P}$ interact about all of F's layers at once?

No. $\mathcal{V}$ must ask questions in correct order or $\mathcal{P}$ can cheat!

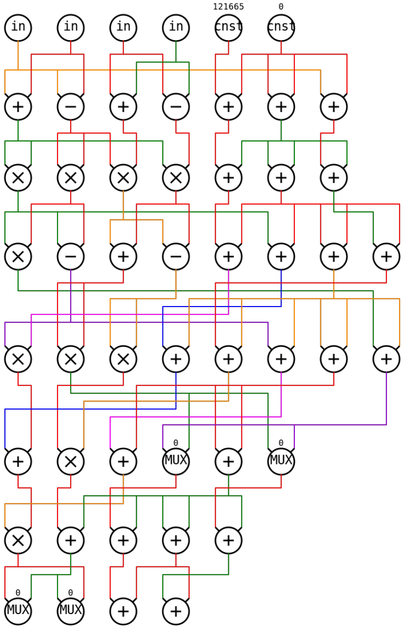


Can we parallelize this interaction?

Can $\mathcal{V}$ and $\mathcal{P}$ interact about all of F's layers at once?

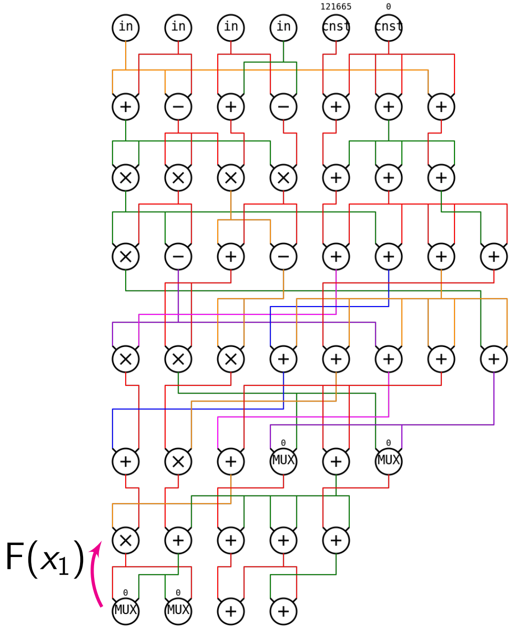
No. $\mathcal{V}$ must ask questions in correct order or $\mathcal{P}$ can cheat!

But: Zebra uses pipelining to parallelize several Fs.



Extracting parallelism through pipelining

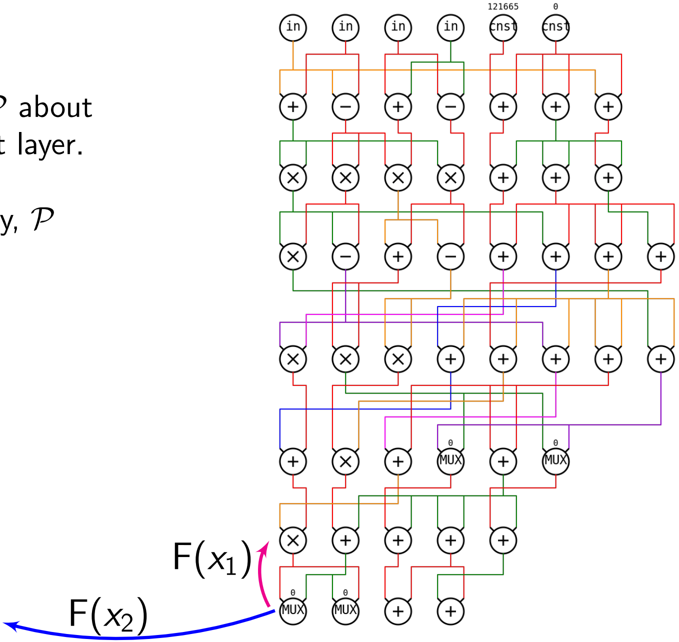
$\mathcal{V}$ questions $\mathcal{P}$ about $\mathrm{F}(x_1)$'s output layer.



Extracting parallelism through pipelining

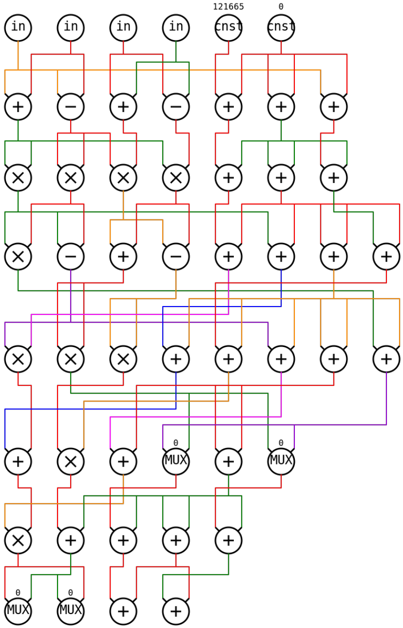
$\mathcal{V}$ questions $\mathcal{P}$ about $\mathsf{F}(x_1)$'s output layer.

Simultaneously, $\mathcal{P}$ returns $\mathsf{F}(x_2)$.



Extracting parallelism through pipelining

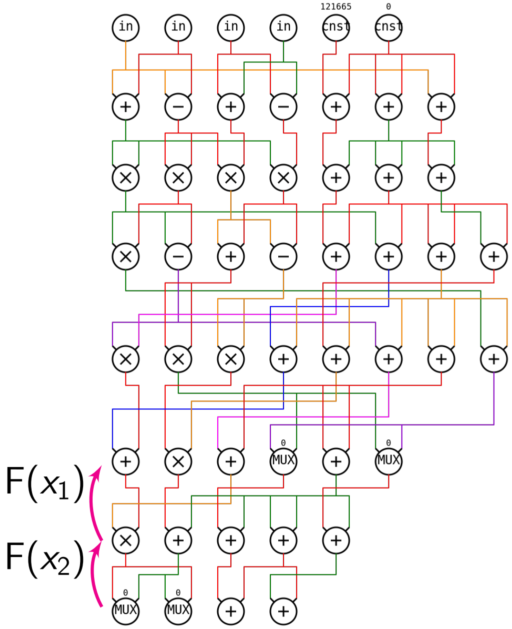
$\mathcal{V}$ questions $\mathcal{P}$ about $F(x_1)$'s next layer



$F(x_1)$

Extracting parallelism through pipelining

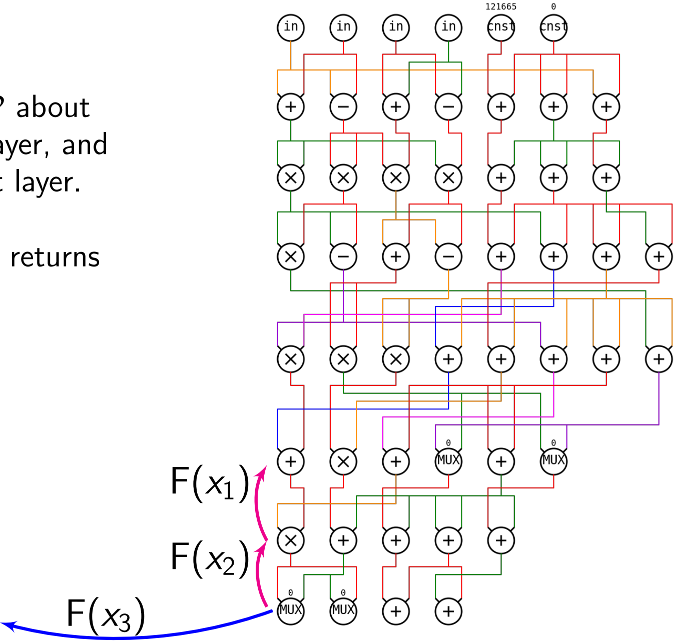
$\mathcal{V}$ questions $\mathcal{P}$ about F(x_1)'s next layer, and F(x_2)'s output layer.



Extracting parallelism through pipelining

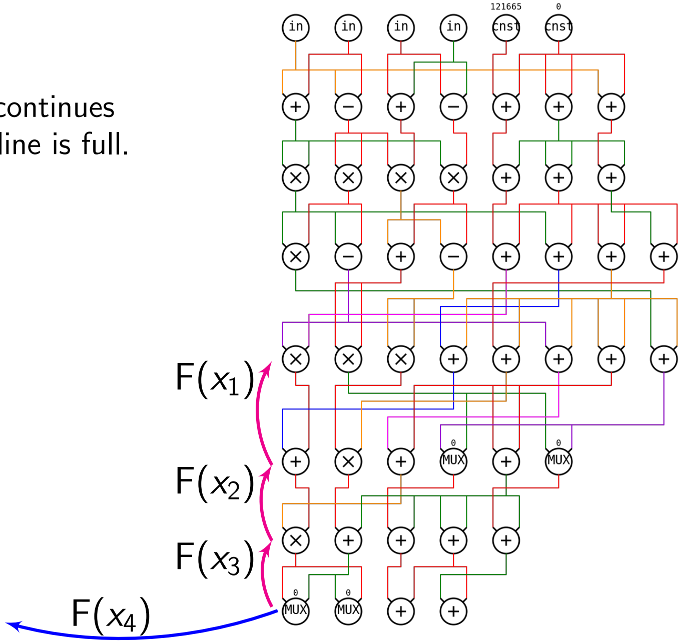
$\mathcal{V}$ questions $\mathcal{P}$ about $F(x_1)$'s next layer, and $F(x_2)$'s output layer.

Meanwhile, $\mathcal{P}$ returns $F(x_3)$.



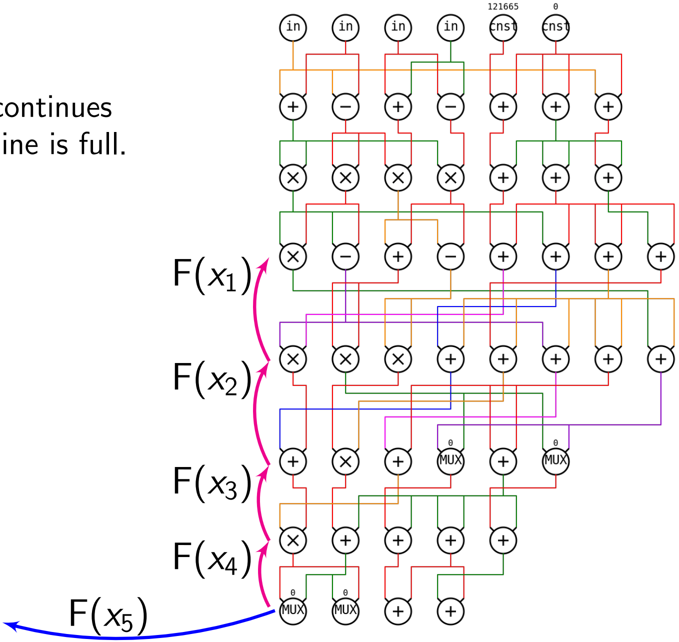
Extracting parallelism through pipelining

This process continues until the pipeline is full.



Extracting parallelism through pipelining

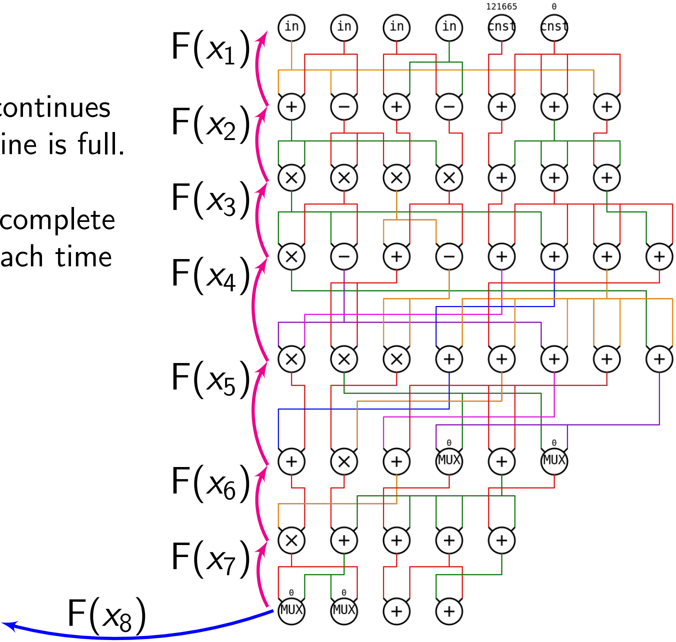
This process continues until the pipeline is full.



Extracting parallelism through pipelining

This process continues until the pipeline is full.

$\mathcal{V}$ and $\mathcal{P}$ can complete one proof in each time step.



Zebra's design approach

✓ Extract parallelism

e.g., pipelined proving

Zebra's design approach

✓ Extract parallelism

e.g., pipelined proving

✓ Exploit locality: distribute data and control

e.g., no RAM: data is kept close to places it is needed
e.g., *latency-insensitive* design: distributed state machine
avoids bottlenecks associated with central controller

Zebra's design approach

✓ Extract parallelism

e.g., pipelined proving

✓ Exploit locality: distribute data and control

e.g., no RAM: data is kept close to places it is needed
e.g., *latency-insensitive* design: distributed state machine
avoids bottlenecks associated with central controller

✓ Reduce, reuse, recycle

e.g., computation: save energy by adding memoization to $\mathcal{P}$
e.g., hardware: save chip area by reusing the same circuits

Architectural challenges

Interaction between $\mathcal{V}$ and $\mathcal{P}$ requires a lot of bandwidth

✗ $\mathcal{V}$ and $\mathcal{P}$ on circuit board? Too much energy, circuit area

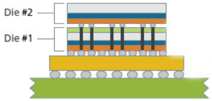
Protocol requires input-independent precomputation [Allspice13]

Architectural challenges

Interaction between $\mathcal{V}$ and $\mathcal{P}$ requires a lot of bandwidth

✗ $\mathcal{V}$ and $\mathcal{P}$ on circuit board? Too much energy, circuit area

✓ Zebra uses *3D integration*



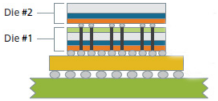
Protocol requires input-independent precomputation [Allspice13]

Architectural challenges

Interaction between $\mathcal{V}$ and $\mathcal{P}$ requires a lot of bandwidth

✗ $\mathcal{V}$ and $\mathcal{P}$ on circuit board? Too much energy, circuit area

✓ Zebra uses *3D integration*



Protocol requires input-independent precomputation [Allspice13]

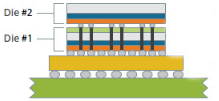
✓ Zebra amortizes precomputations over many $\mathcal{V}$-$\mathcal{P}$ pairs

Architectural challenges

Interaction between $\mathcal{V}$ and $\mathcal{P}$ requires a lot of bandwidth

✗ $\mathcal{V}$ and $\mathcal{P}$ on circuit board? Too much energy, circuit area

✓ Zebra uses *3D integration*



Protocol requires input-independent precomputation [Allspice13]

✓ Zebra amortizes precomputations over many $\mathcal{V}$-$\mathcal{P}$ pairs

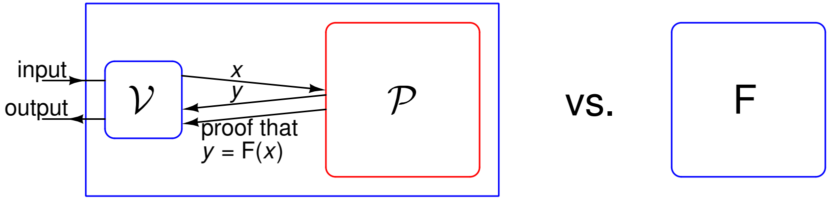
Several other details (see paper)

Implementation

Zebra's implementation includes

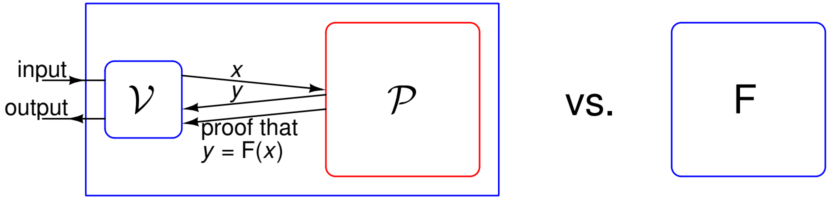
- a compiler that produces synthesizable Verilog for $\mathcal{P}$
- two $\mathcal{V}$ implementations
  - hardware (Verilog)
  - software (C++)
- library to generate $\mathcal{V}$'s precomputations
- Verilog simulator extensions to model software or hardware $\mathcal{V}$'s interactions with $\mathcal{P}$

Evaluation method



Baseline: direct implementation of F in same technology as $\mathcal{V}$

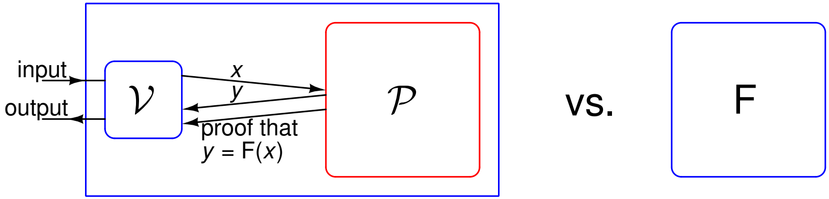
Evaluation method



Baseline: direct implementation of F in same technology as $\mathcal{V}$

Metrics: energy, chip size per throughput (see paper)

Evaluation method



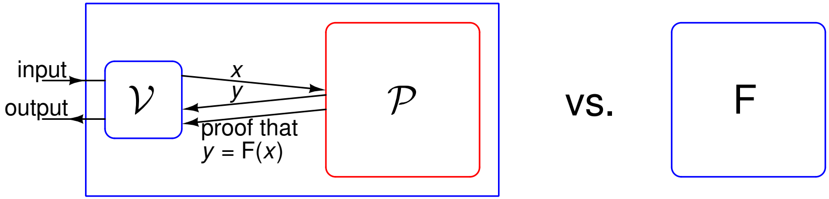
Baseline: direct implementation of F in same technology as $\mathcal{V}$

Metrics: energy, chip size per throughput (see paper)

Measurements: based on circuit synthesis and simulation, published chip designs, and CMOS scaling models

Charge for $\mathcal{V}$, $\mathcal{P}$, communication; retrieving and decrypting precomputations; PRNG; Operator communicating with $\mathcal{V}$

Evaluation method



Baseline: direct implementation of F in same technology as $\mathcal{V}$

Metrics: energy, chip size per tl

Measurements: based on circuit
published chip designs, and CM

Charge for $\mathcal{V}$, $\mathcal{P}$, communi
precomputations; PRNG; Operator communicating with

350 nm: 1997 (Pentium II)
7 nm: $\approx$ 2017 [TSMC]
$\approx$ 20 year gap between
trusted and untrusted fab

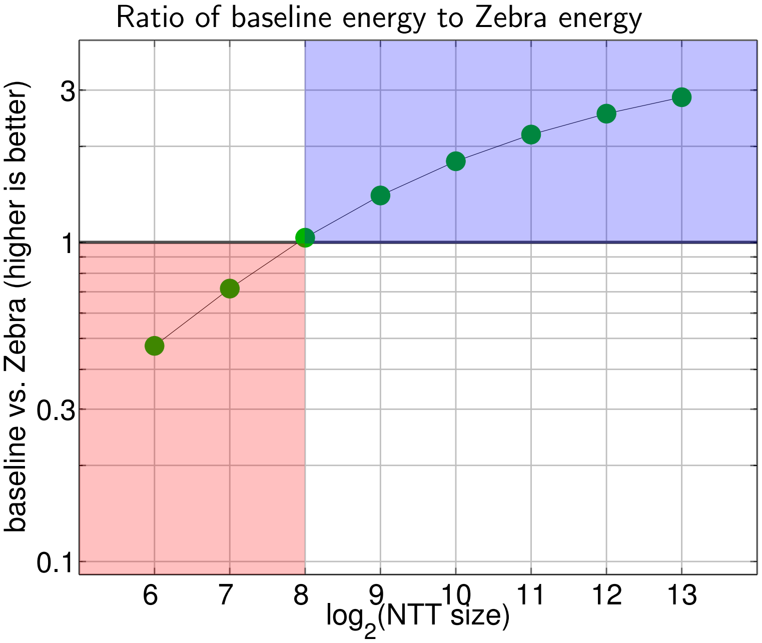
Constraints: trusted fab = 350 nm; untrusted fab = 7 nm
200 mm^2 max chip area; 150 W max total power

Application #1: number theoretic transform

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

Widely used, e.g., in computer algebra

Application #1: number theoretic transform

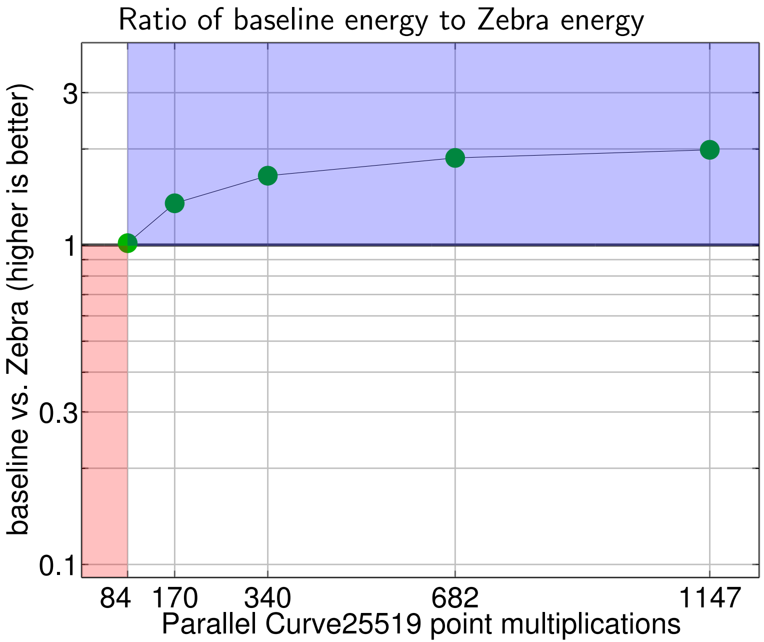


Application #2: Curve25519 point multiplication

Curve25519: a commonly-used elliptic curve

Point multiplication: primitive used for ECDH

Application #2: Curve25519 point multiplication



A **qualified** success

Zebra: a hardware design that saves costs...

...**sometimes**.

Summary of Zebra's applicability

1. Must have a wide gap between cutting-edge fab for $\mathcal{P}$ and trusted fab for $\mathcal{V}$
2. Must amortize precomputations over many instances
3. Computation F must be very large for $\mathcal{V}$ to save work
4. Computation F must be efficient as an arithmetic circuit
5. Computation F must have a layered, shallow, deterministic AC

Summary of Zebra's applicability

Common to essentially all built proof systems

1. Must have a wide gap between cutting-edge fab for $\mathcal{P}$ and trusted fab for $\mathcal{V}$
2. Must amortize precomputations over many instances
3. Computation F must be very large for $\mathcal{V}$ to save work
4. Computation F must be efficient as an arithmetic circuit

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

Summary of Zebra's applicability

Common to essentially all built proof systems

1. Must have a wide gap between cutting-edge fab for $\mathcal{P}$ and trusted fab for $\mathcal{V}$
2. Must amortize precomputations over many instances
3. Computation F must be very large for $\mathcal{V}$ to save work
4. Computation F must be efficient as an arithmetic circuit
5. Computation F must have a layered, shallow, deterministic AC

Applies to IPs, but not arguments

Arguments versus IPs, redux

Design principle	**IPs** [GKR08, CMT12, VSBW13]	**Arguments** [GGPR13, SBVBPW13, PGHR13, BCTV14]
Extract parallelism	✓	✓
Exploit locality	✓	
Reduce, reuse, recycle	✓	

Argument protocols seem friendly to hardware?

Arguments versus IPs, redux

Design principle	**IPs** [GKR08, CMT12, VSBW13]	**Arguments** [GGPR13, SBVBPW13, PGHR13, BCTV14]
Extract parallelism	✓	✓
Exploit locality	✓	✗
Reduce, reuse, recycle	✓	

Argument protocols seem unfriendly to hardware:

$\mathcal{P}$ computes over entire AC at once $\implies$ need RAM

Arguments versus IPs, redux

Design principle	**IPs** [GKR08, CMT12, VSBW13]	**Arguments** [GGPR13, SBVBPW13, PGHR13, BCTV14]
Extract parallelism	✓	✓
Exploit locality	✓	✗
Reduce, reuse, recycle	✓	✗

Argument protocols seem unfriendly to hardware:

$\mathcal{P}$ computes over entire AC at once $\Longrightarrow$ need RAM

$\mathcal{P}$ does crypto for every gate in AC $\Longrightarrow$ special crypto circuits

Arguments versus IPs, redux

Design principle	**IPs** [GKR08, CMT12, VSBW13]	**Arguments** [GGPR13, SBVBPW13, PGHR13, BCTV14]
Extract parallelism	✓	✓
Exploit locality	✓	✗
Reduce, reuse, recycle	✓	✗

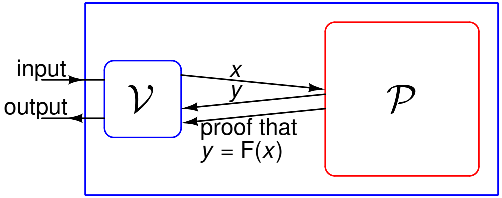
Argument protocols seem unfriendly to hardware:

$\mathcal{P}$ computes over entire AC at once $\implies$ need RAM

$\mathcal{P}$ does crypto for every gate in AC $\implies$ special crypto circuits

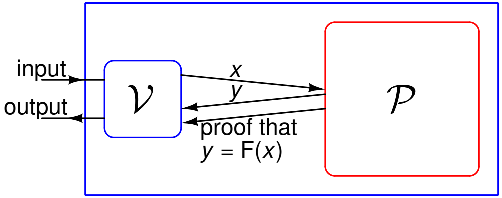
. . . but we hope these issues are surmountable!

Recap



- Verifiable ASICs: a new approach to building trustworthy hardware under a strong threat model
- First hardware design for a probabilistic proof protocol
- Improves performance compared to trusted baseline

Recap



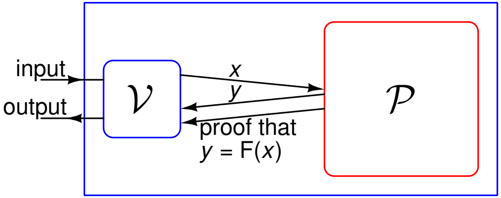
+ Verifiable ASICs: a new approach to building trustworthy hardware under a strong threat model
+ First hardware design for a probabilistic proof protocol
+ Improves performance compared to trusted baseline

– Improvement compared to the baseline is modest

– Applicability is limited:

  precomputations must be amortized
  computation needs to be "big enough"
  large gap between trusted and untrusted technology
  does not apply to all computations

Recap



+ Verifiable ASICs: a new approach to building trustworthy hardware under a strong threat model
+ First hardware design for a probabilistic proof protocol
+ Improves performance compared to trusted baseline

– Improvement compared to the baseline is modest
– Applicability is limited:
  - precomputations must be amortized
  - computation needs to be "big enough"
  - large gap between trusted and untrusted technology
  - does not apply to all computations

https://www.pepper-project.org/