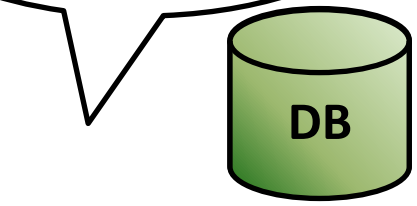
Succinct non-interactive zero knowledge for a von Neumann architecture

Eli Ben-Sasson
Alessandro Chiesa
Eran Tromer
Madars Virza

Technion Israel Institute of Technology
Massachusetts Institute of Technology
אוניברסיטת תל-אביב TEL AVIV UNIVERSITY
Massachusetts Institute of Technology

I'm a manager at a hospital

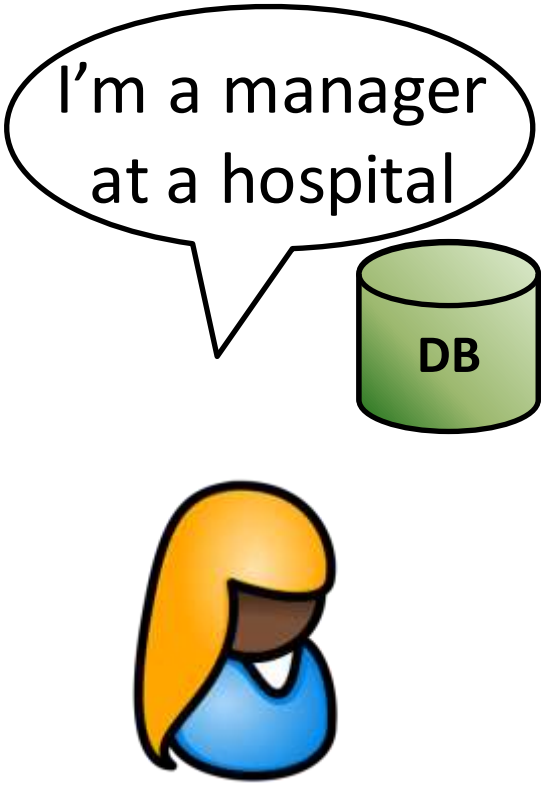


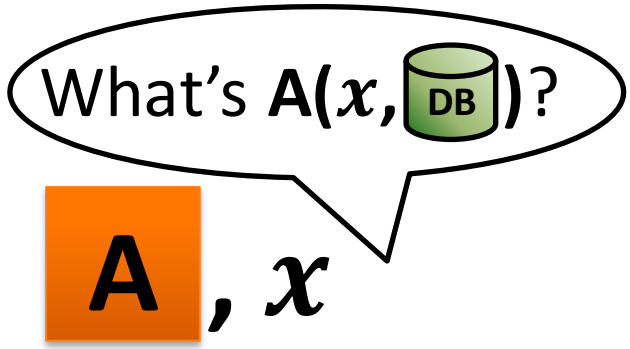


I'm a researcher.

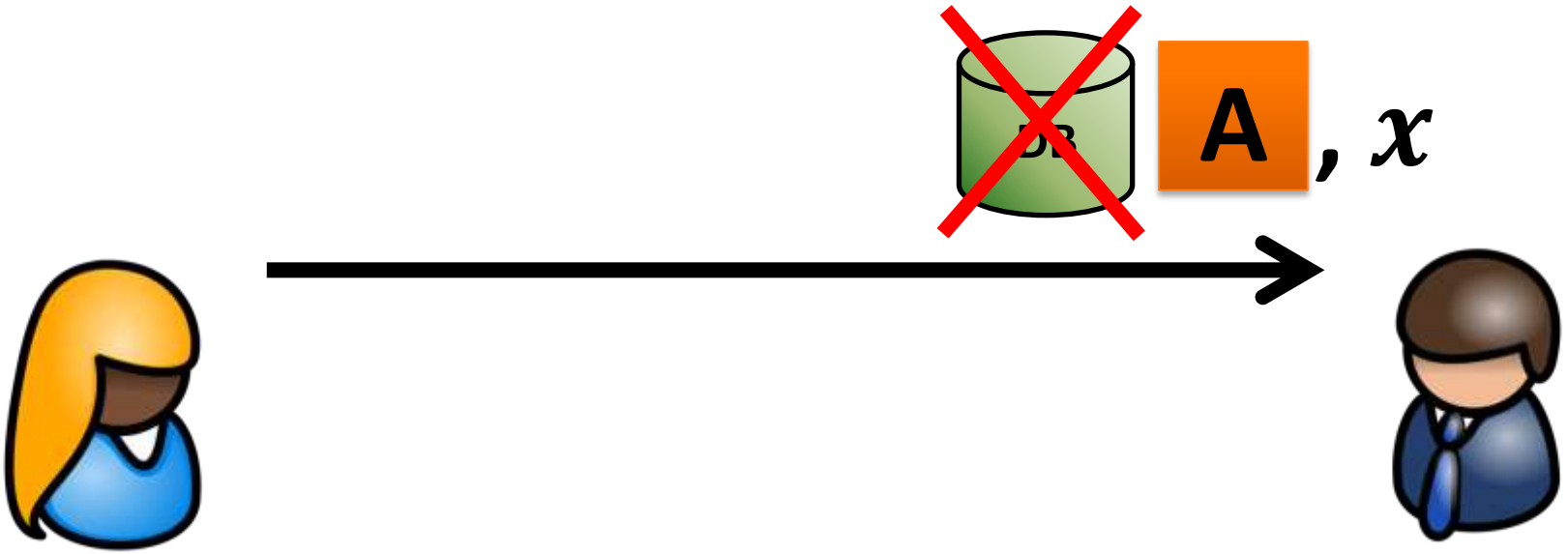
A, x




I'm a manager at a hospital
DB

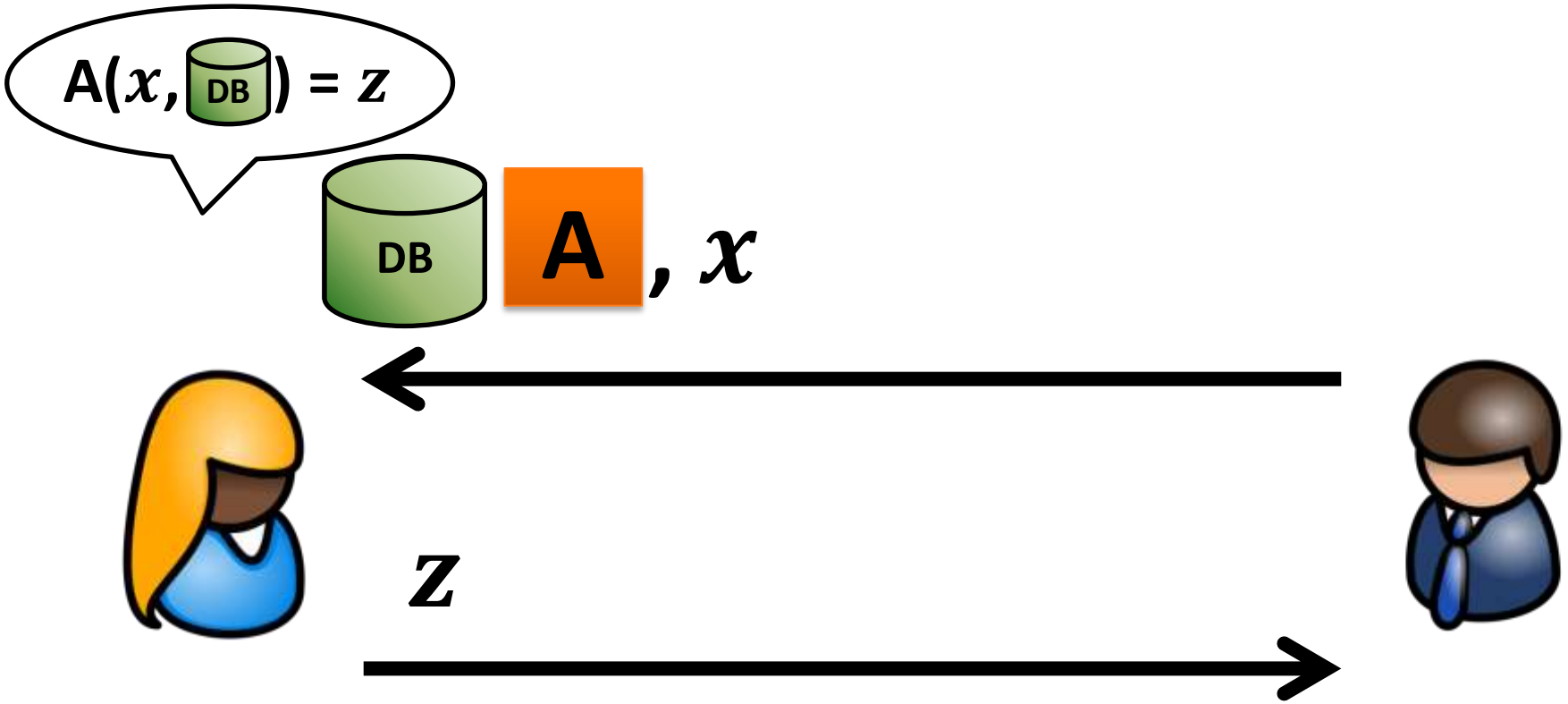

What's A(x, DB)?
A, x

Privacy.



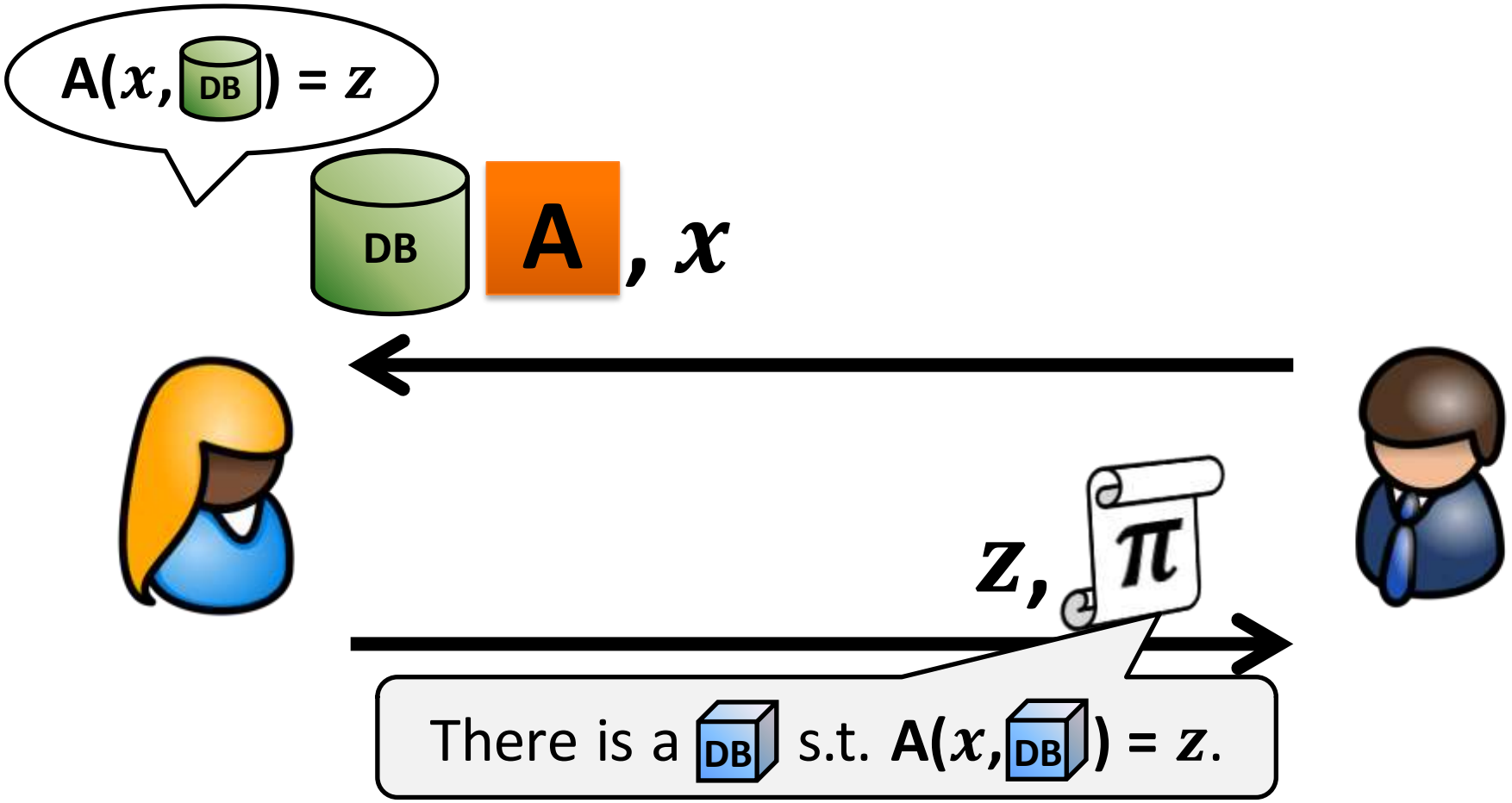
Privacy.

Verifiability.



Privacy.

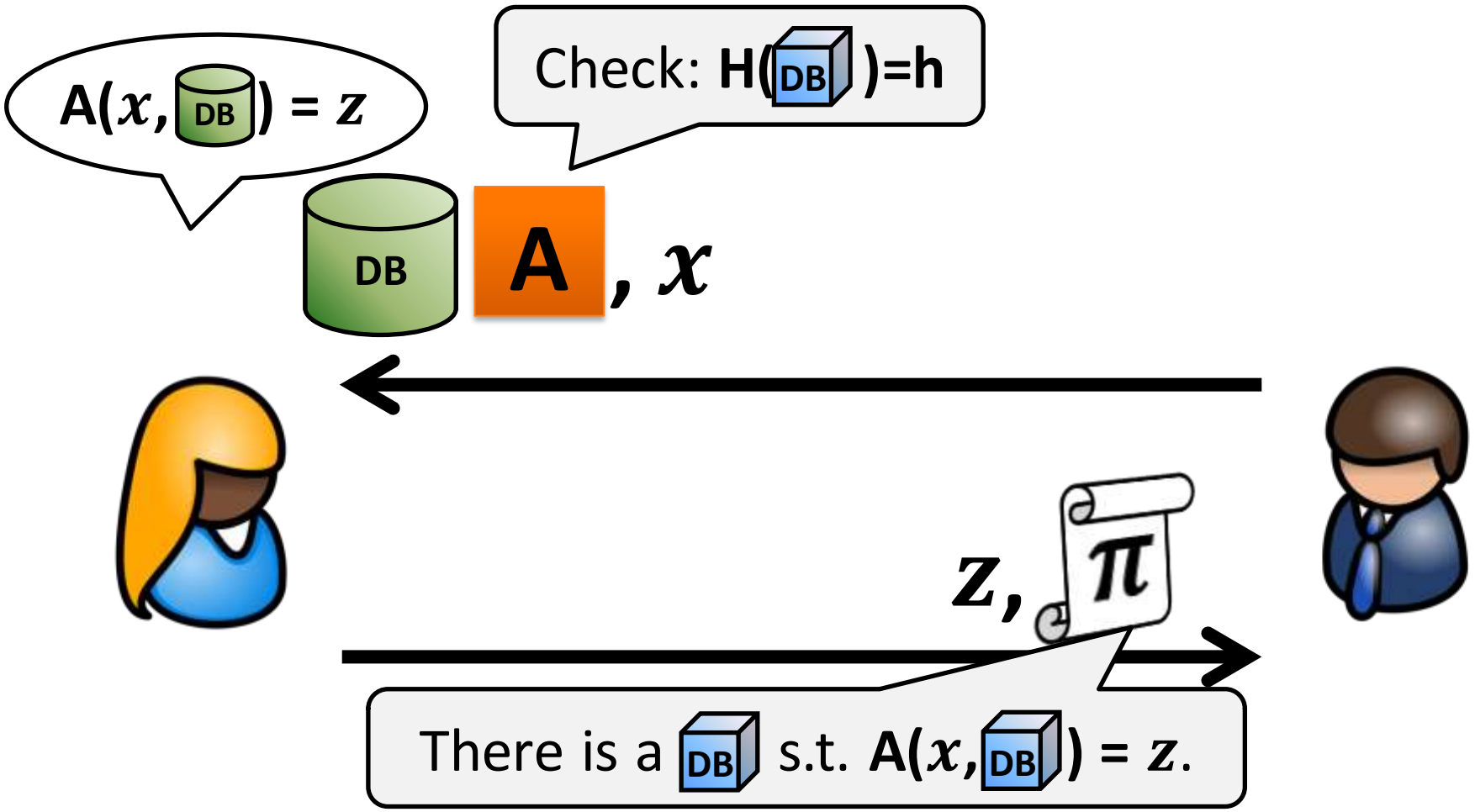
Verifiability.

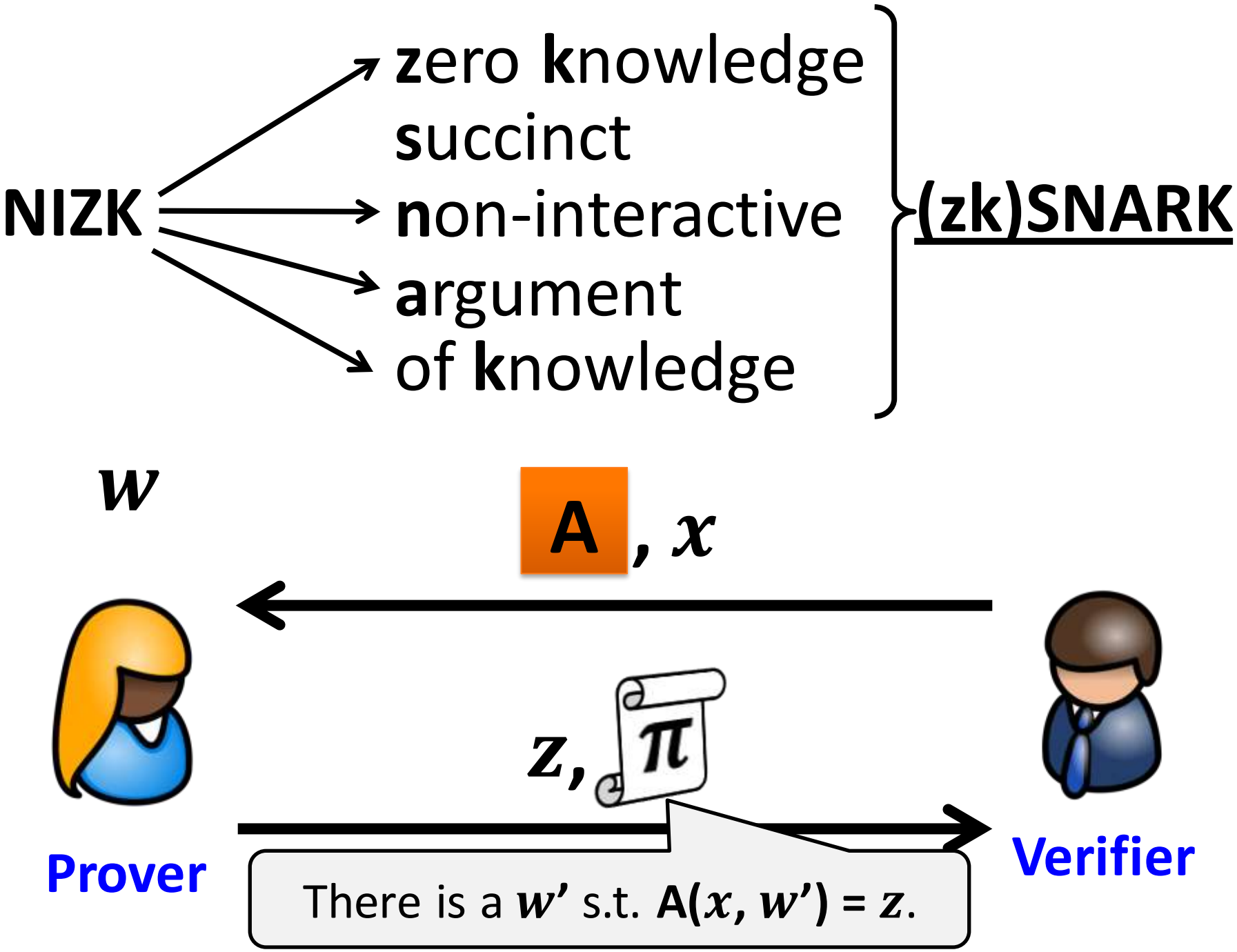


h=H(DB)

Privacy.

Verifiability.




zero knowledge
succinct
NIZK
non-interactive
argument
of knowledge
(zk)SNARK
w
A , x
z, π
Prover
Verifier
There is a w' s.t. A(x, w') = z.

SNARKs require setup



Generator

CRS | pk | vk

Once: initial trusted step

Many times

w

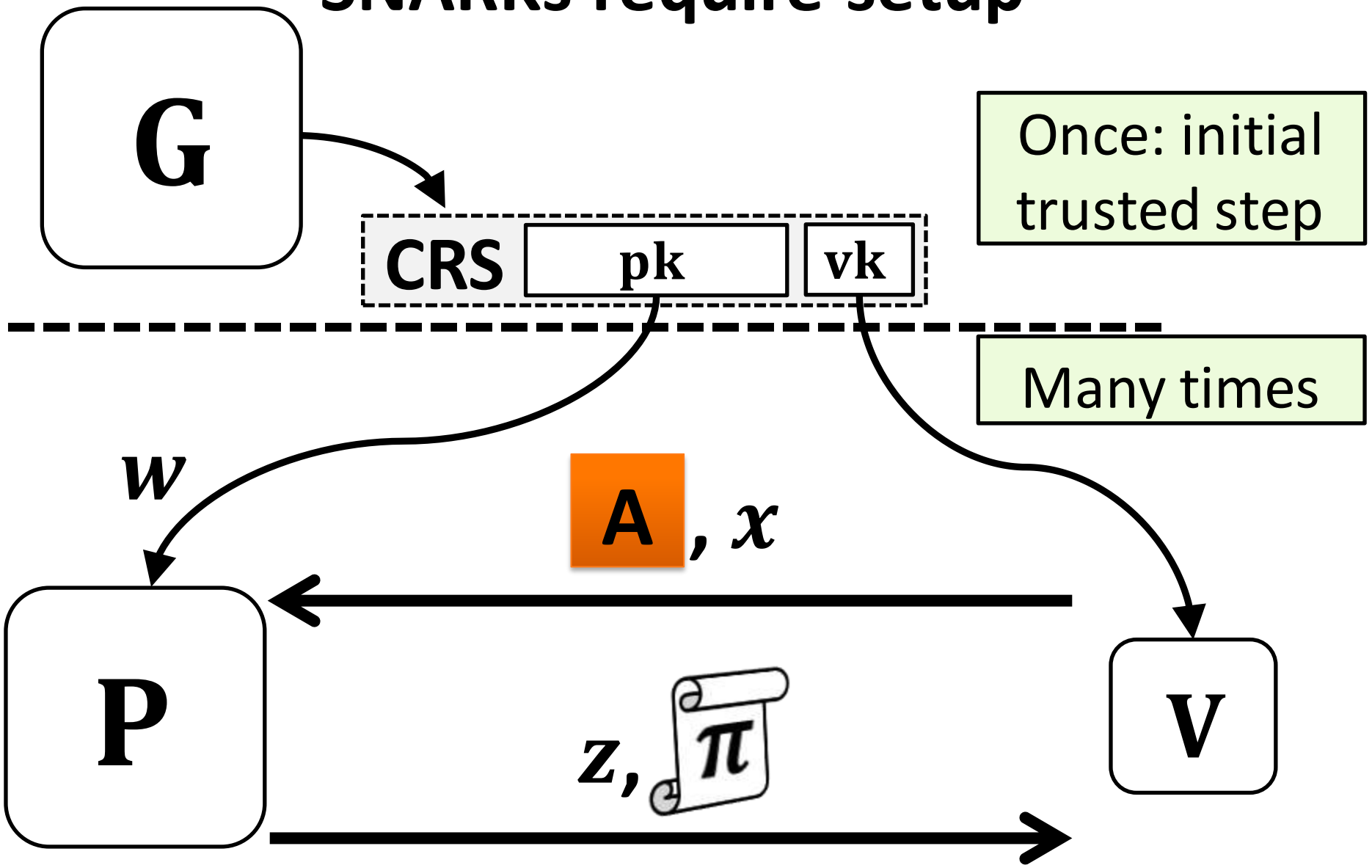
A, x

z, π

Prover

Verifier

SNARKs require setup



Prior work on SNARKs

(i) Theoretical constructions

[Kilian 92] [Micali 94] [Di Crescenzo Lipmaa 08] [Groth 10]
[Goldwasser Lin Rubinstein 11] [Lipmaa 12] [Bitansky Chiesa 12]
[Bitansky Canetti Chiesa Tromer 12] [Damgård Faust Hazay 12]
[Gennaro Gentry Parno Raykova 13] [Bitansky Chiesa Ishai Ostrovsky Paneth 13]

(ii) Working prototypes

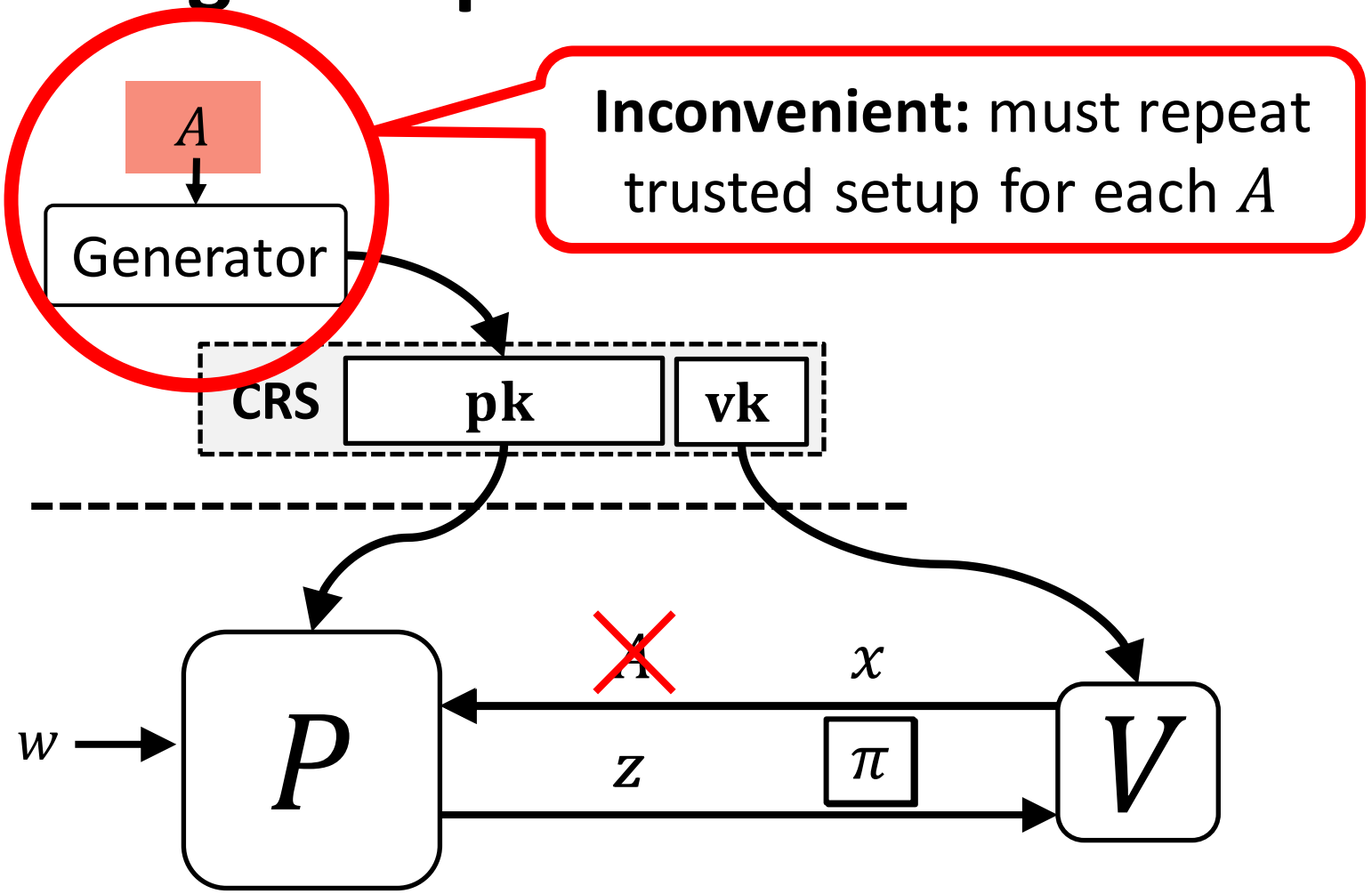
[Parno Gentry Howell Raykova 13]
[Ben-Sasson Chiesa Genkin Tromer Virza 13]
[Braun Feldman Ren Setty Blumberg Walfish 13]

SNARKs are practically feasible for certain applications!

Application example: **Zerocash** relies on a SNARK…
… for a tailored program that only has to check SHA256 computations and integer arithmetic.

[Ben-Sasson Chiesa Garman Green Miers Tromer Virza 2014]

Program-specific SNARKs

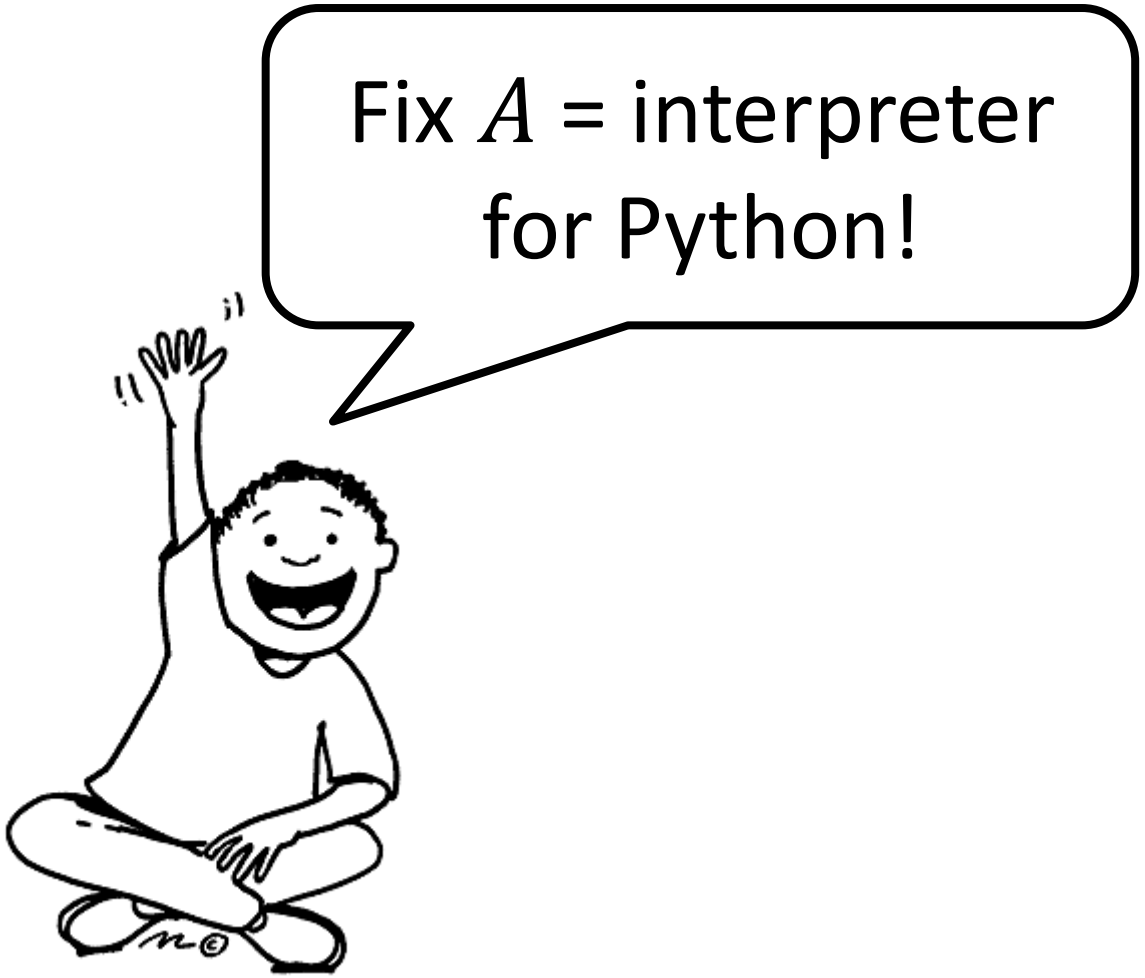


Currently: all prior implementations are program-specific

Ideally: one-time trusted setup that works for any A

Goal: SNARKs with universal trusted setup

Naive approach:



Goal: SNARKs with universal trusted setup

Naive approach:



Goal: SNARKs with universal trusted setup

Our results: Achieve the above goal.*

Performance for T-step programs:

Time(Prover)	Space(Prover)
$50T$ ms	$3T$ MB

We do so via two independent contributions:

1. Universal circuit generator

2. SNARK implementation for circuits

Both are significantly more efficient than prior work.

(* whenever the computation is suitably bounded)

The high-level picture

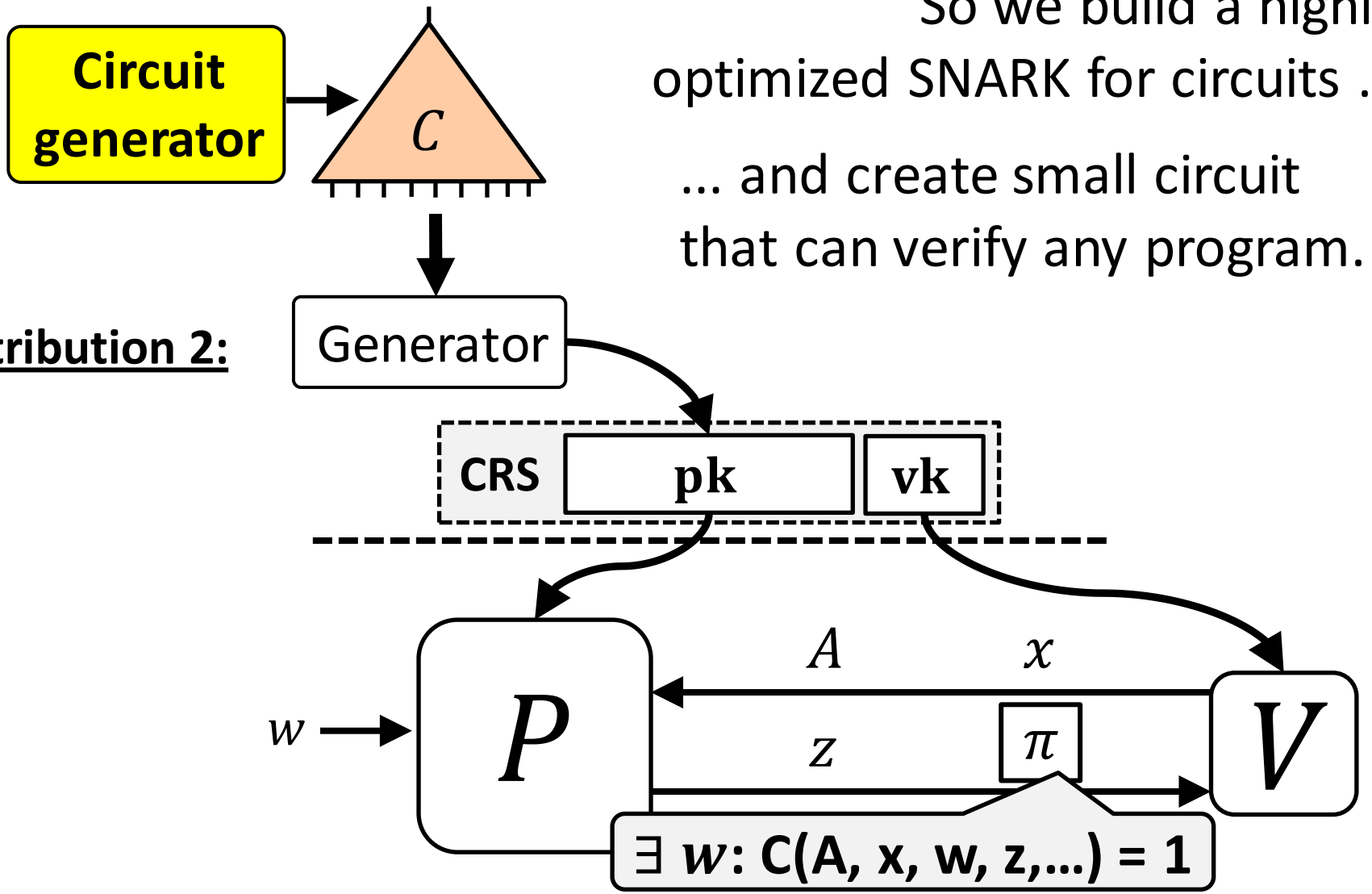
Best theoretical constructions achieve SNARKs for circuits.

Contribution 1:

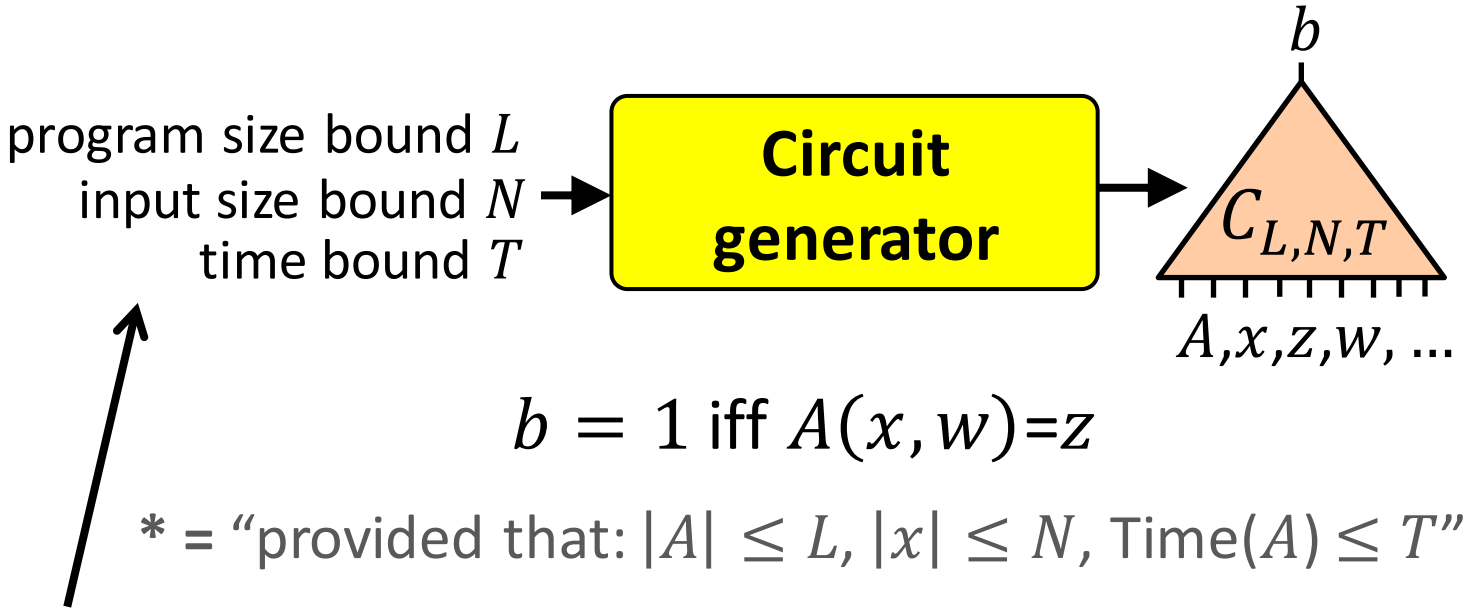
So we build a highly optimized SNARK for circuits …

… and create small circuit that can verify any program.

Contribution 2:



Contribution 1:
Universal circuit generator



$b = 1$ iff $A(x,w)=z$

* = "provided that: $|A| \leq L$, $|x| \leq N$, Time$(A) \leq T$"

- **Universal.**
- **Efficient.**

  Asymptotically: $|C_{L,N,T}| = O\big((L+N+T)\log(L+N+T)\big)$

  Concretely: $|C_{L,N,T}|/T \approx 2000$ for reasonable bounds

- **Expressive.**

  32-bit RISC von Neumann machine

  (can even support self-modifying code and JIT!)

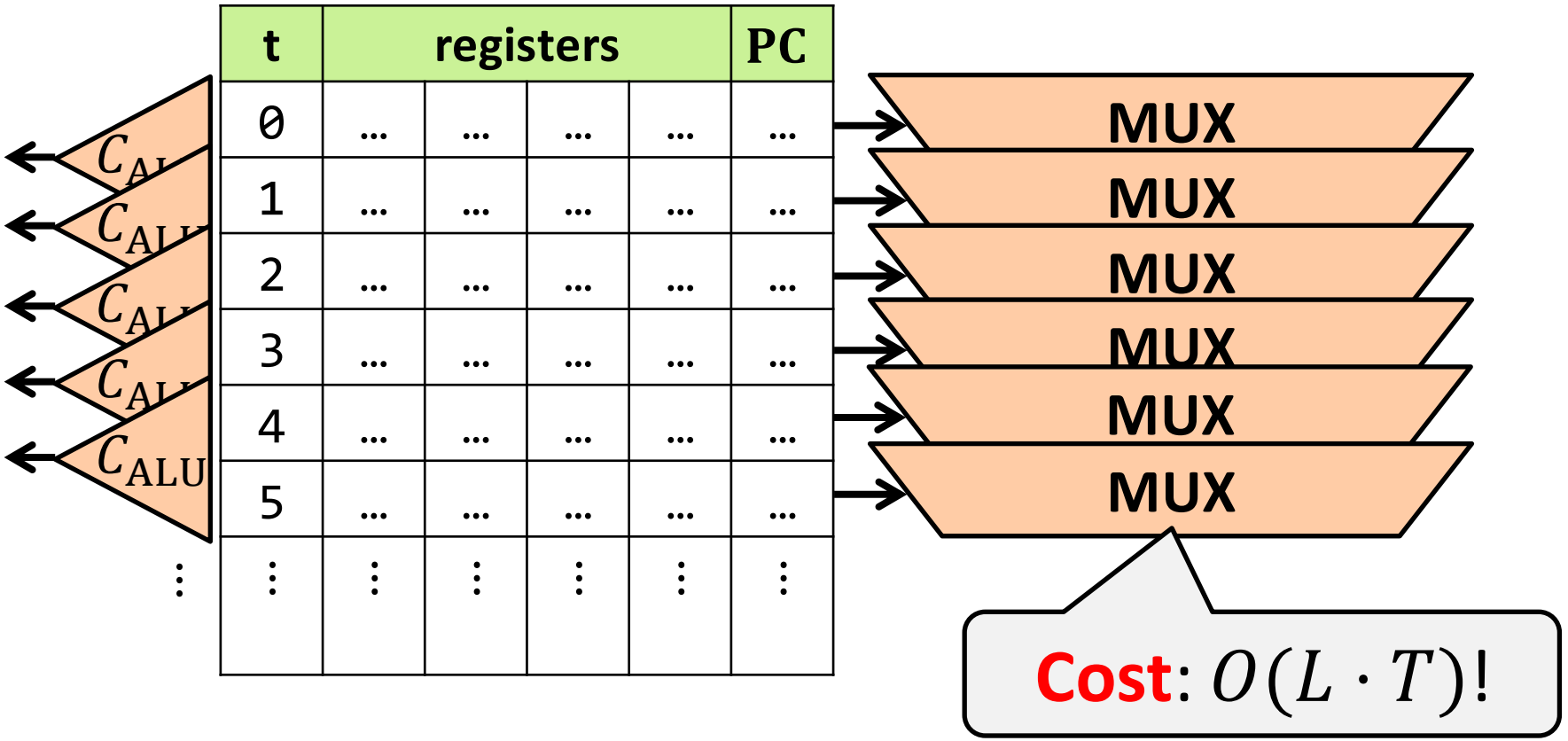
Our circuit generator

(based on [BCGTV13])

Circuit checks trace of a RAM computation, as follows:

① Check ALU instructions in the straightforward way

[BCGTV13]: use multiplexer



Our circuit generator

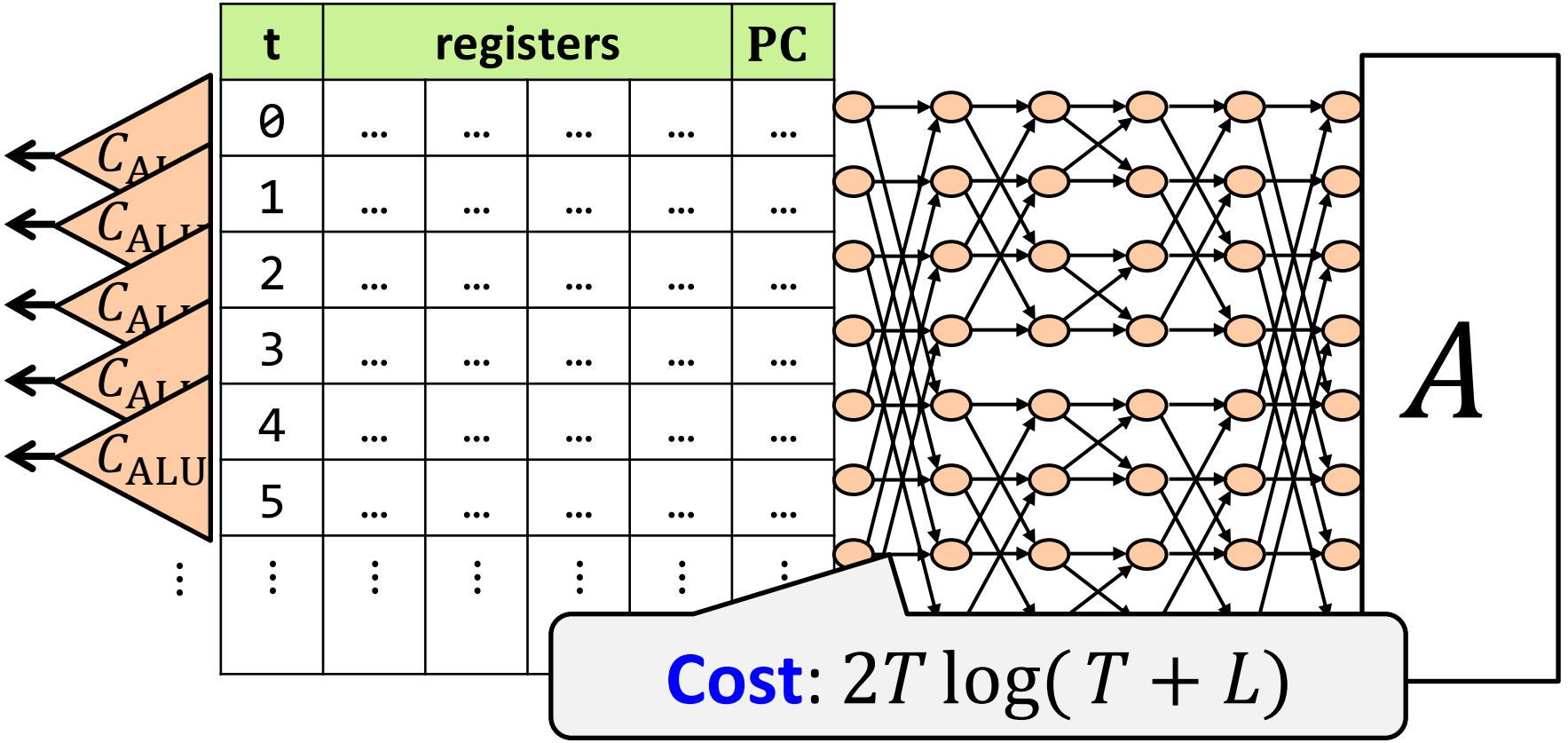
(based on [BCGTV13])

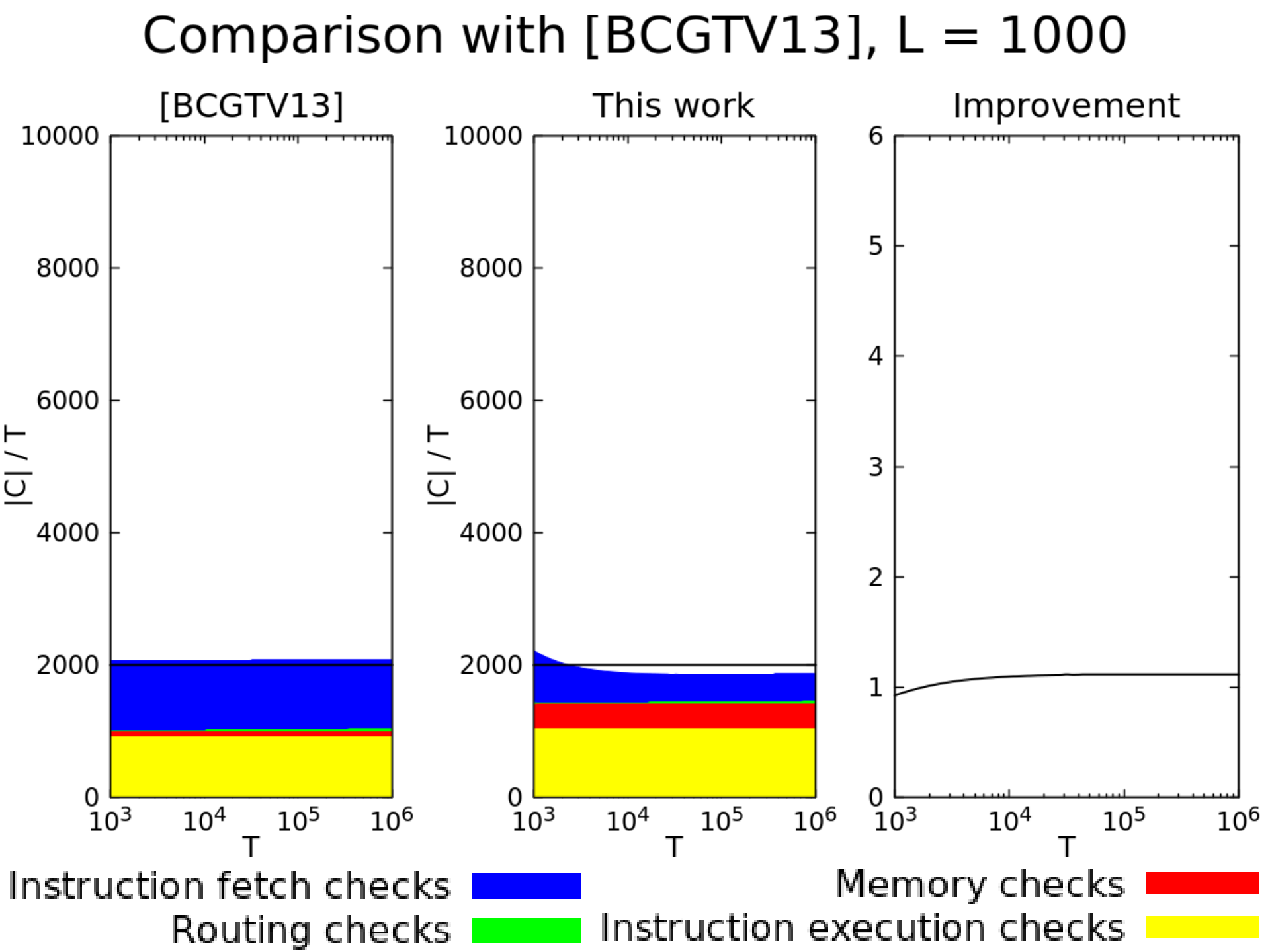
Circuit checks trace of a RAM computation, as follows:

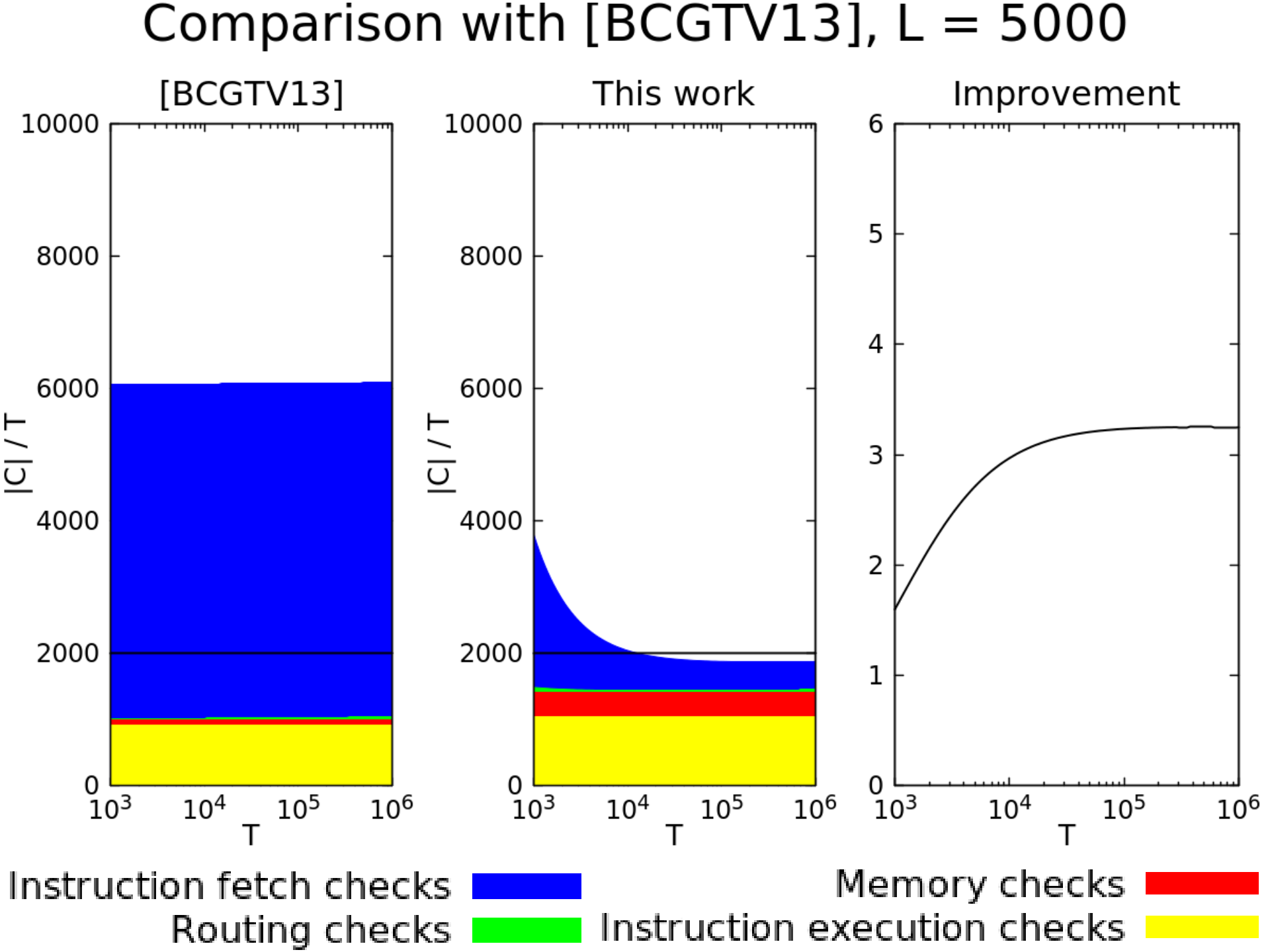
① Check ALU instructions in the straightforward way

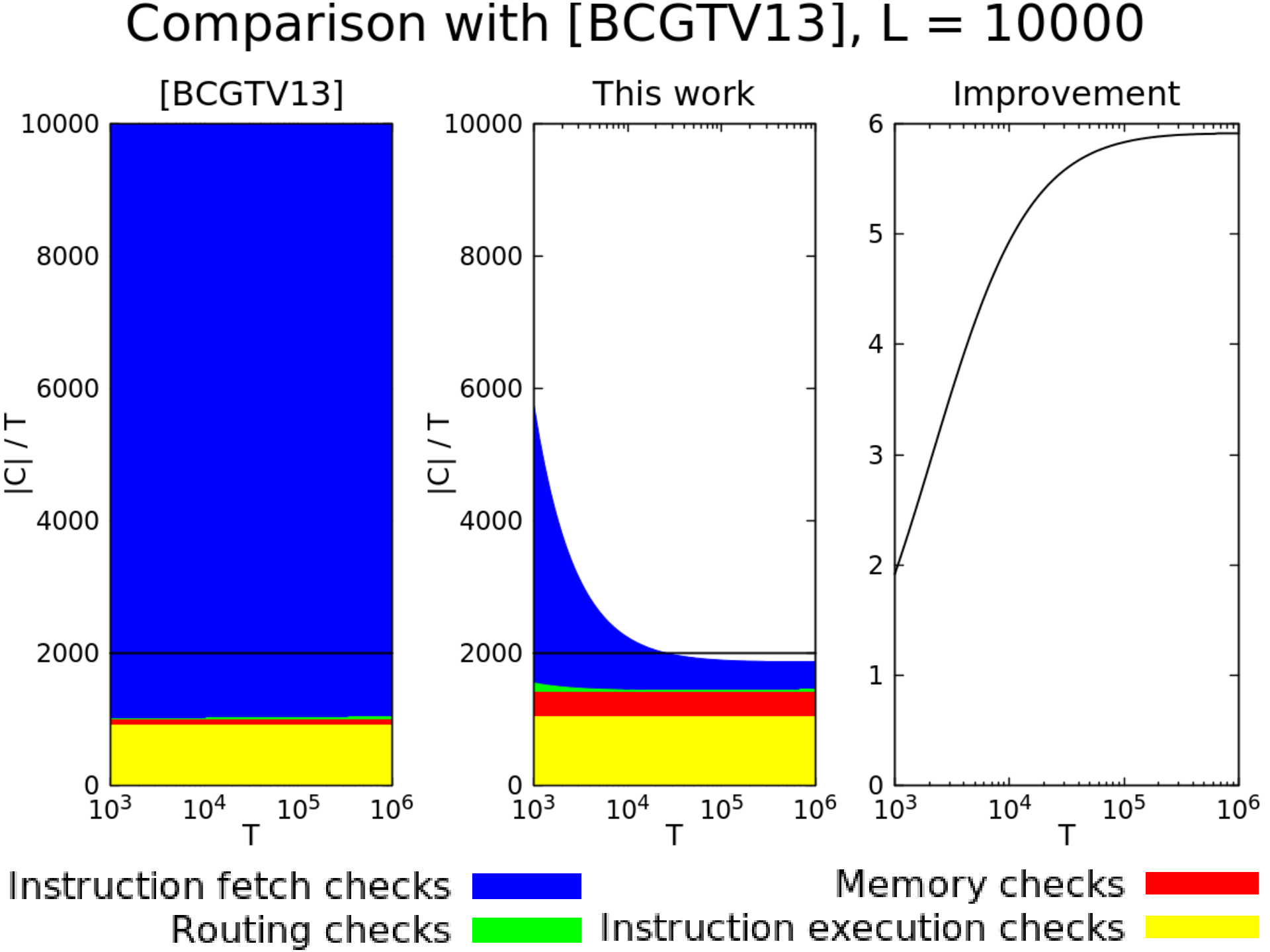
② Perform instruction fetch using non-deterministic routing

③ Check read/write instructions using same routing network




Comparison with [BCGTV13], L = 1000
[BCGTV13]
This work
Improvement
10000
8000
6000
4000
2000
0
|C| / T
6
5
4
3
2
1
10^3
10^4
10^5
10^6
T
Instruction fetch checks
Routing checks
Memory checks
Instruction execution checks


Comparison with [BCGTV13], L = 5000
[BCGTV13]
This work
Improvement
|C| / T
T
10000
8000
6000
4000
2000
0
6
5
4
3
2
1
10^3
10^4
10^5
10^6
Instruction fetch checks
Routing checks
Memory checks
Instruction execution checks


Comparison with [BCGTV13], L = 10000
[BCGTV13]
This work
Improvement
|C| / T
T
10000
8000
6000
4000
2000
0
6
5
4
3
2
1
10^3
10^4
10^5
10^6
Instruction fetch checks
Routing checks
Memory checks
Instruction execution checks

Comparison with program analysis

Most circuit generators use program analysis:

Fairplay	[Malkhi Nisan Pinkas Sella 04]
FairplayMP	[Ben-David Nisan Pinkas 08]
Ginger	[Setty Vu Panpalia Braun Blumberg Walfish 12]
Zaatar	[Setty Braun Vu Blumberg Parno Walfish 12]
Pinocchio	[Parno Gentry Howell Raykova 13]
Pantry	[Braun Feldman Ren Setty Blumberg Walfish 13]

(a) dissect program A into components (loops, branches, …)

(b) separately translate each component into a subcircuit

How to compare?

- Not universal (must know A in advance)
- Efficiency varies from A to A… hard to characterize.

  Roughly, very efficient for "circuit-like" programs; less efficient for memory-intensive programs.

Contribution 2:

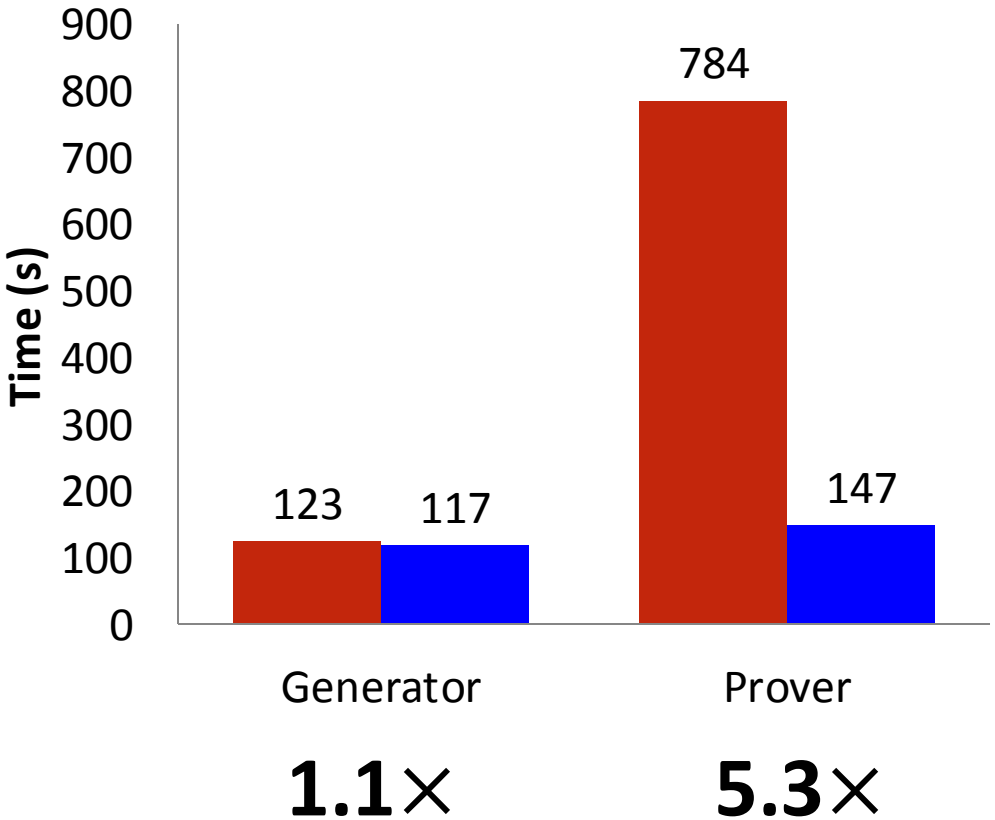
SNARK implementation for circuits

SNARK implementation for circuits

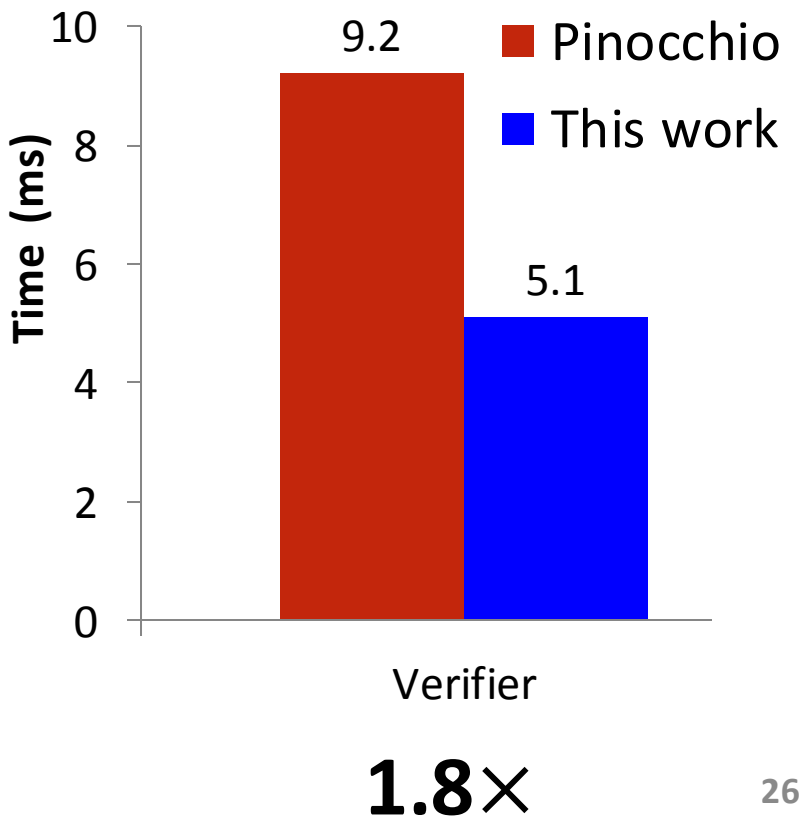
- Theoretical protocol based on [PGHR13]
- Tailored choices at every level of stack:

Finite fields, elliptic curves, pairings, multi-exponentiation, interpolation.

For 10^6-gate circuit and 1000-bit input:



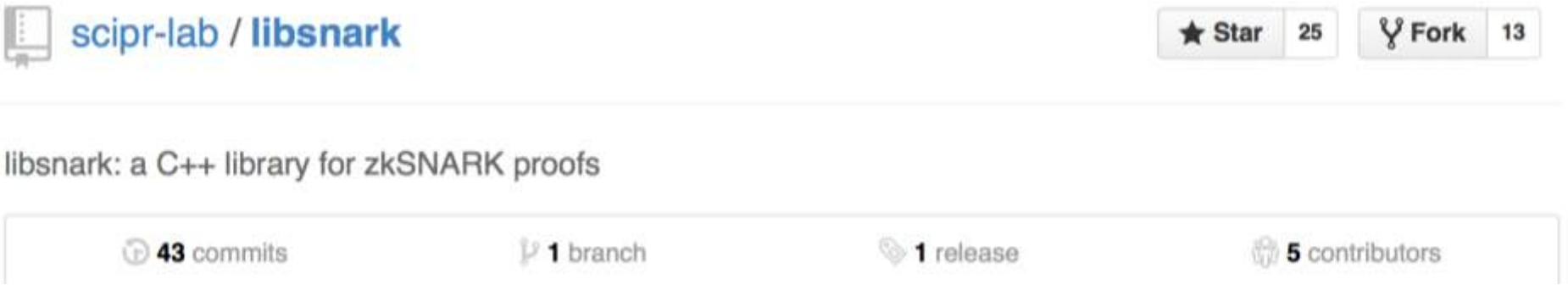
1.1× **5.3×**



1.8×

libsnark: our SNARK implementation

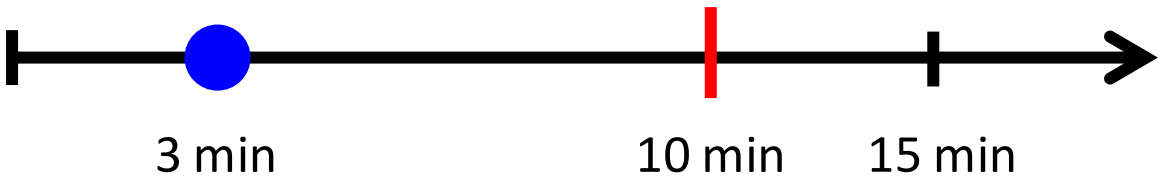
- **Open source**



`github.com/scipr-lab/libsnark`

- **Practical impact**

Time to create Zerocash transaction:



- `libsnark`
- block creation interval

Universality:

One-time trusted setup to **verify** **any** computation in **zero-knowledge**.

Efficiency:

Fastest zk-SNARK for circuits.

Applications:

Look at `libsnark` and come talk to us!

Thank you!

ePrint 2013/879

Research sponsored by:











Case study: memcpy with JIT compilation

`memcpy(void *dest, const void *src, size_t n)`

Trivial as a circuit only for **fixed** `src`, `dest` and `n`.

```
  ...
_copy:
  load.b  reg, [src]
  store.b [dest], reg
  add     src, src, 1
  add     dest, dest, 1
  sub     n, n, 1
  cnjmp    _copy
  ...
```

Can be eliminated when unrolling

Total cost: $\approx 6n$

With JIT:

- Use Newton's method to calculate $k = \sqrt{2n/7}$
- Unroll regular code k times
- Jump to the generated code

Total cost: $\approx 4n + 11.48\sqrt{n}$