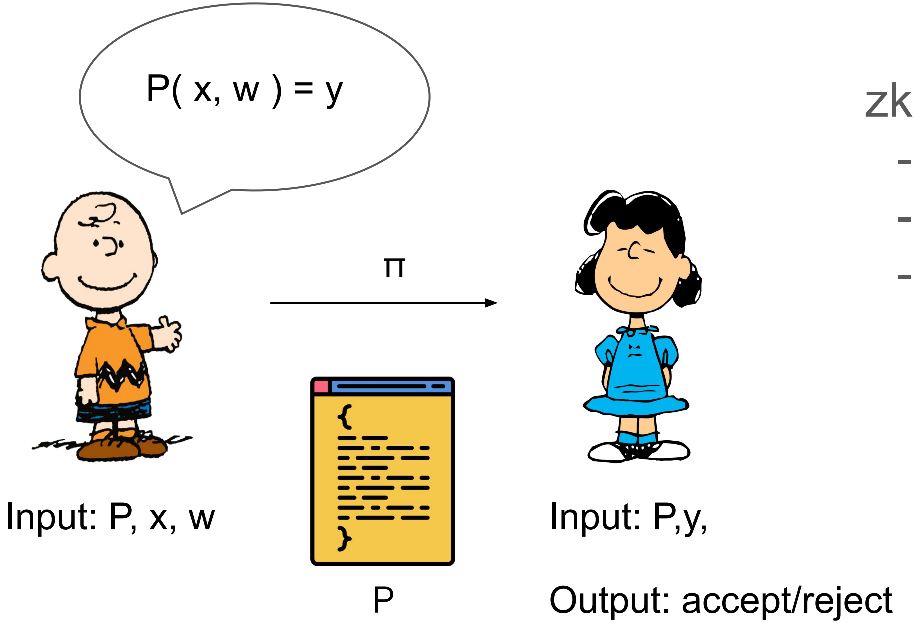
zkSNARKs for Virtual Machines are Non-Malleable

Matteo Campanelli, Antonio Faonio, Luigi Russo

EURECOM
Sophia Antipolis

Proofs of program execution



P(x, w) = y

π

Input: P, x, w

P

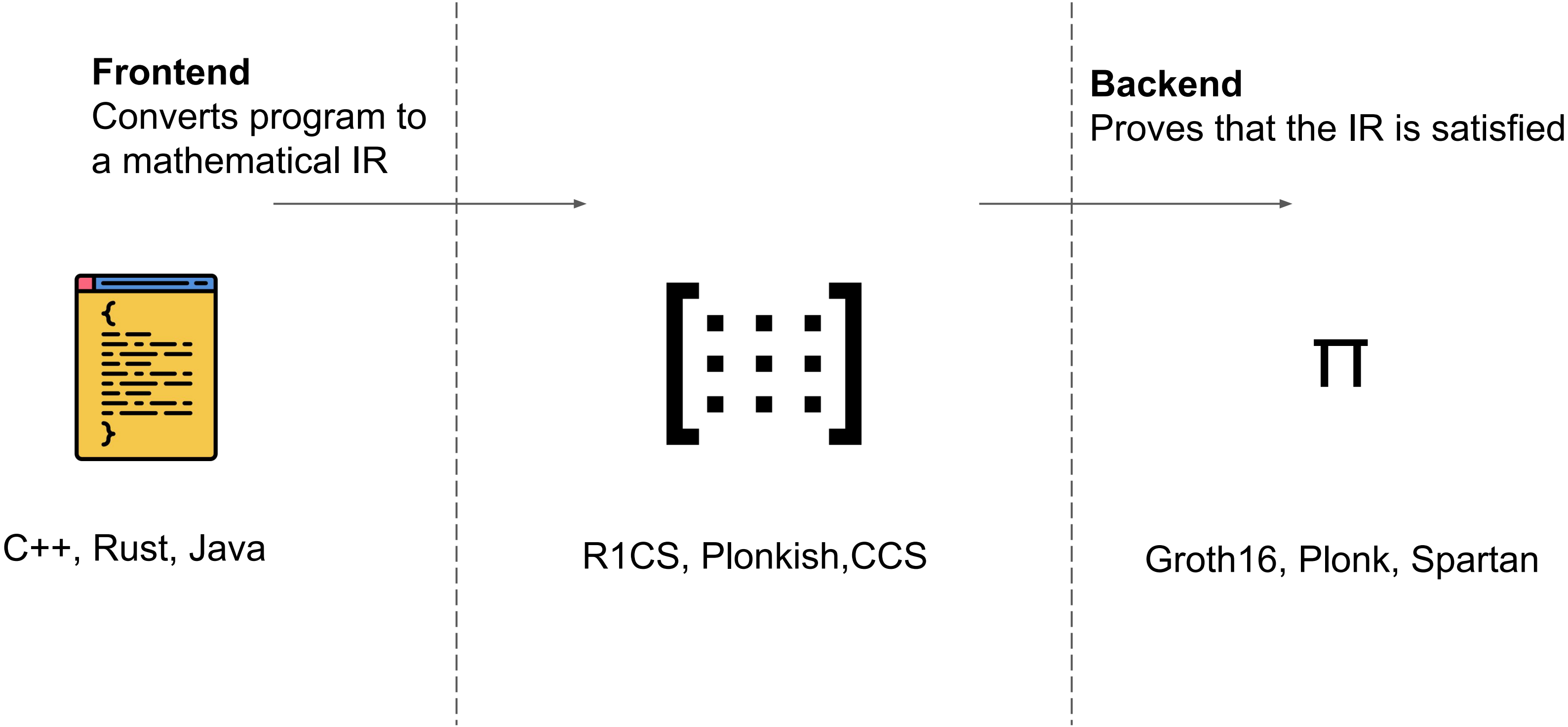
Input: P,y,

Output: accept/reject

zkSNARK

- Zero-knowledge
- Non-Interactive
- Succincts

zkSNARKs: frontends and backends



Frontend
Converts program to a mathematical IR

Backend
Proves that the IR is satisfied

C++, Rust, Java

R1CS, Plonkish,CCS

Groth16, Plonk, Spartan

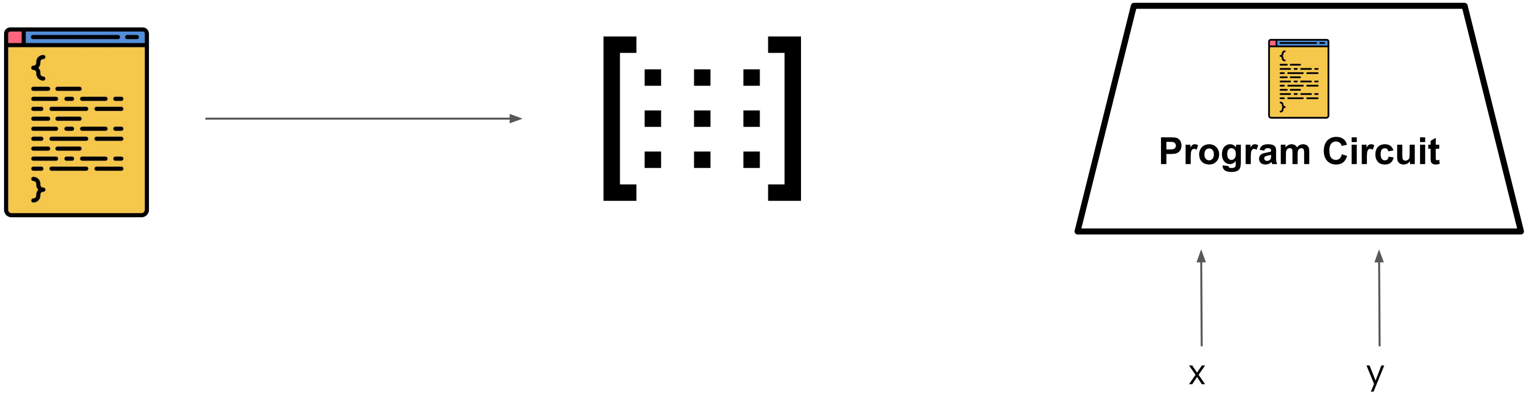
The classical approach

Per-Program compilation
compiles each program into a new “circuit”

R1CS, Plonkish,CCS

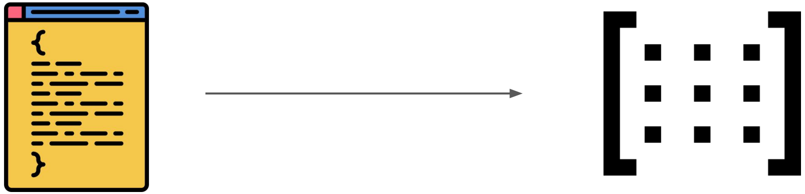
The classical approach

Per-Program compilation
compiles each program into a new “circuit”

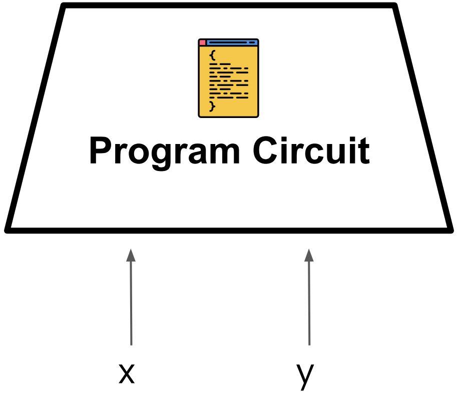


The classical approach

Per-Program compilation
compiles each program into a new “circuit”



- Need to perform auditing and formal verification
- Ad-hoc languages and tooling

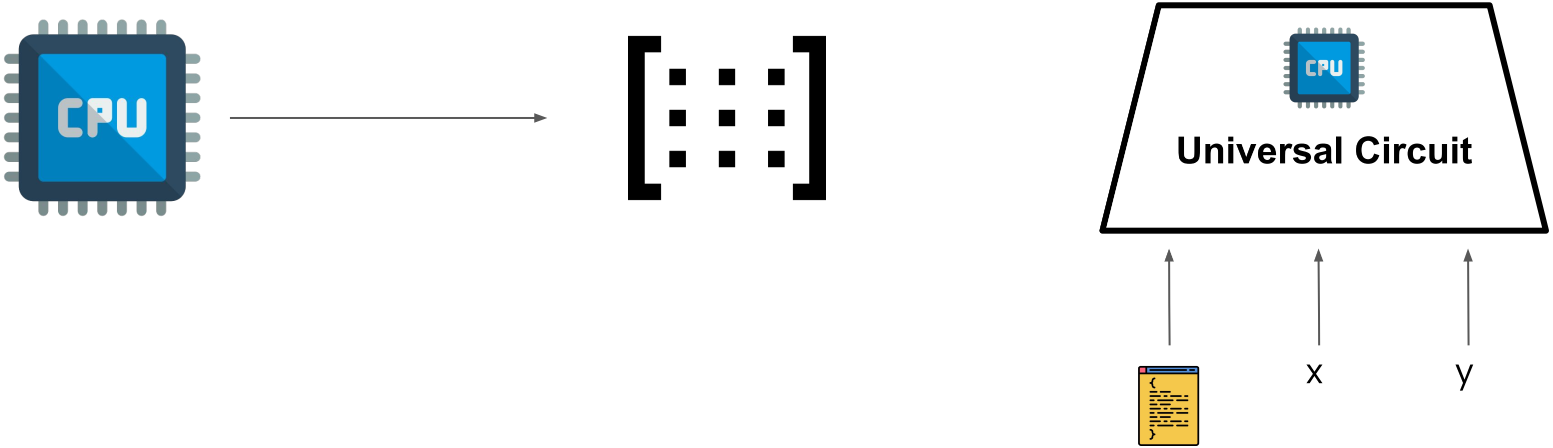


The zkVM approach

Per-Processor compilation
use a universal circuit that takes input (P,x) outputs y=P(x)

Frontend: a different approach

Per-Processor compilation
use a universal circuit that takes input (P,x) outputs y=P(x)



Universal Circuit

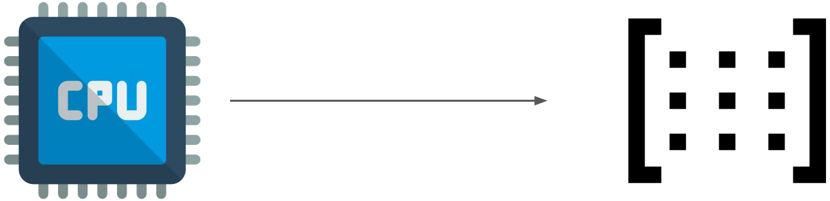
x

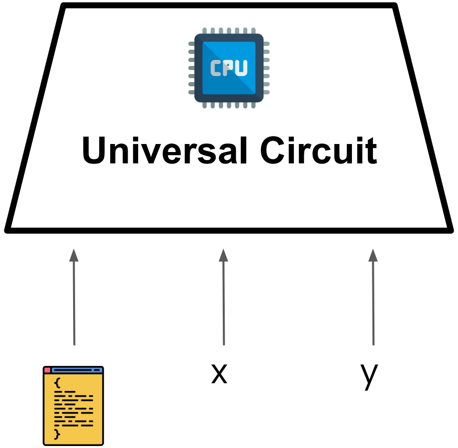
y

Frontend: a different approach

Per-Processor compilation

use a universal circuit that takes input (P,x) outputs y=P(x)



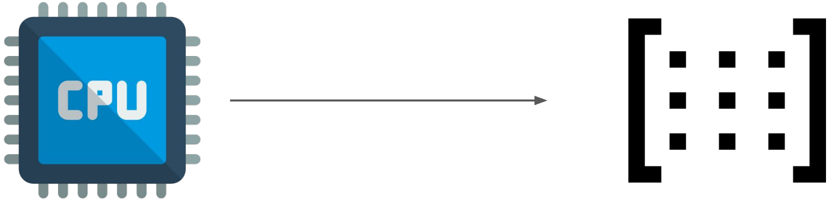


+ Auditing and formal verification on one circuit

+ Re-use existing languages and tooling

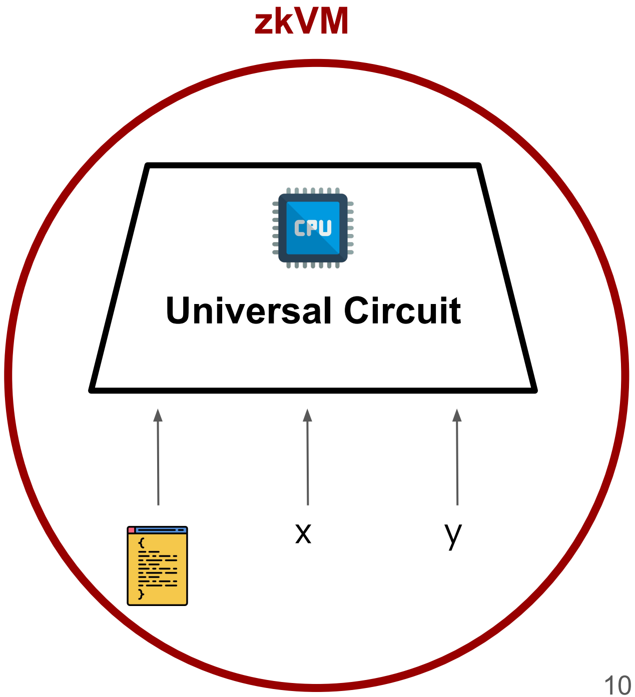
Frontend: a different approach

Per-Processor compilation
use a universal circuit that takes input (P,x) outputs y=P(x)



+ auditing and formal verification on one circuit

+ Re-use existing languages and tooling



Applications & Use Cases

Generic

- Proof of solvency
- Image provenance
- Content moderation
- Fancy T-Shirt with a succinct proof of Fermat's last theorem (OR .. OR ..)

Applications & Use Cases

Generic

- Proof of solvency

- Image provenance

- Content moderation

- Fancy T-Shirt with a succinct proof of Fermat's last theorem (OR .. OR ..)

Blockchain-related

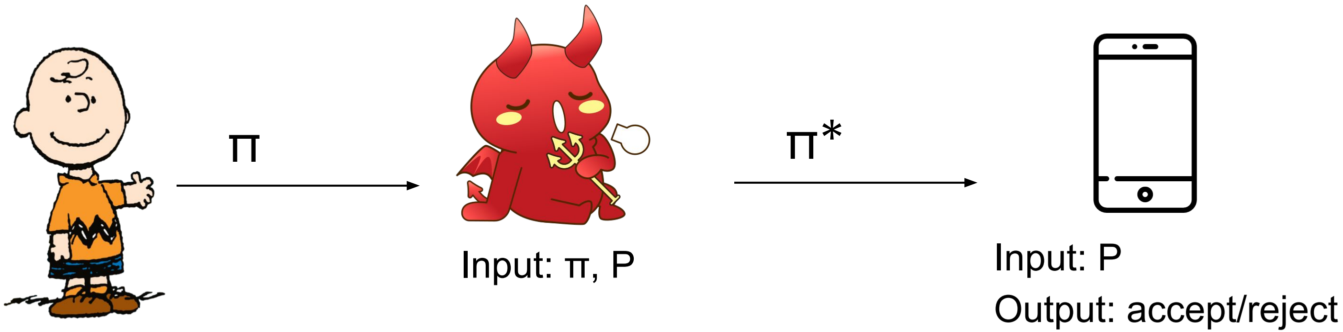
- Private transactions

- Private smart contracts

- ZK-Rollups

Can old proofs on-chain be useful to the adversary?

Security Question



π

π^*

Input: π, P

Input: P

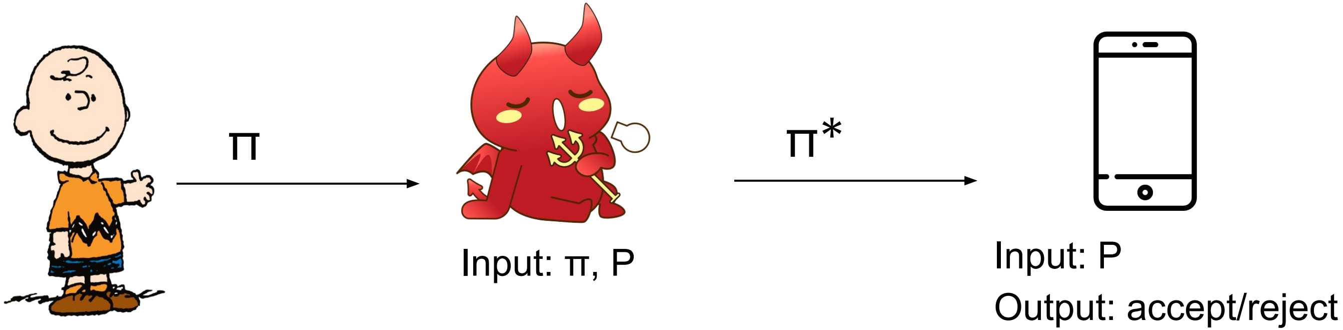
Output: accept/reject

Malleability attack

Modify an existing proof into a new proof

without knowing the witness

Security Question



π

π^*

Input: π, P

Input: P

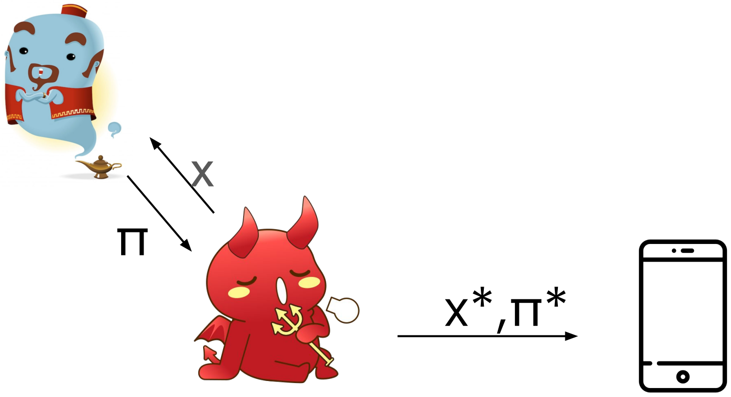
Output: accept/reject

Malleability attack

Modify an existing proof into a new proof without knowing the witness

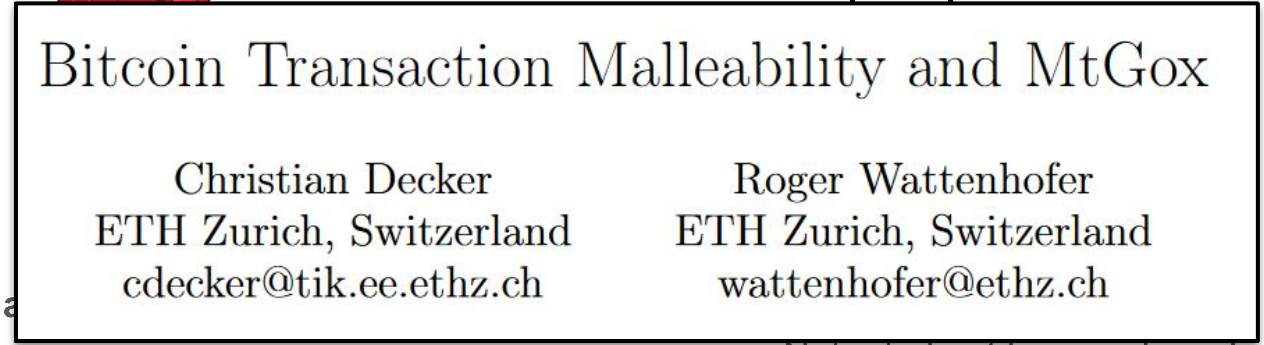
⚠ Not ruled out by zero-knowledge and knowledge soundness

Non-Malleability for zkSNARKS = Simulation-Extractability



- Adversary interacts with ZK simulator, and then forge a proof.
- We consider zkVMs that are indeed zero-knowledge 🙃.

Security Issues



Bitcoin Transaction Malleability and MtGox

Christian Decker
ETH Zurich, Switzerland
cdecker@tik.ee.ethz.ch

Roger Wattenhofer
ETH Zurich, Switzerland
wattenhofer@ethz.ch

Malleability a

Modify an existing proof into a new proof without knowing the witness

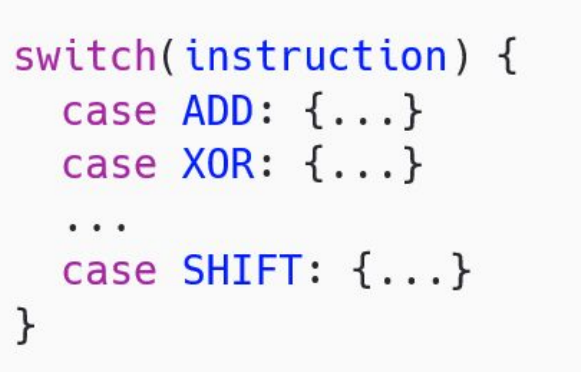
Not ruled out by zero-knowledge and knowledge soundness

Non-malleability of existing zkSNARKs

	[GOP+22]	[GKK+22]	[DG23]	[FFK+23]	[KPT23]	[Lib24]
Bulletproofs	✓		✓			
Spartan			✓			
Sonic		✓				
PLONK		✓		✓	✓	
Marlin		✓		✓	✓	
Lunar				✓	✓	
Basilisk				✓		
HyperPlonk						✓

The complexity of a zkVM

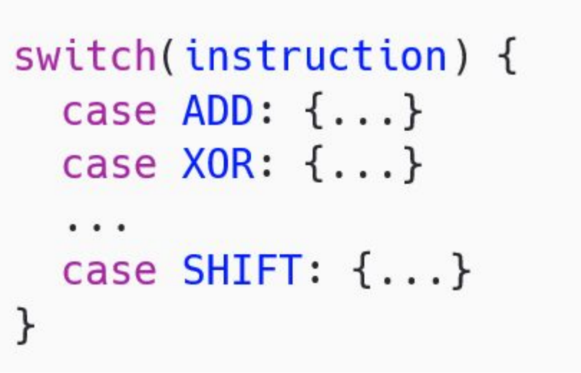
A universal circuit is large
It must be able to execute any operation at each step
e.g. RISC-V has 50 operations



```
switch(instruction) {
  case ADD: {...}
  case XOR: {...}
  ...
  case SHIFT: {...}
}
```

The complexity of a zkVM

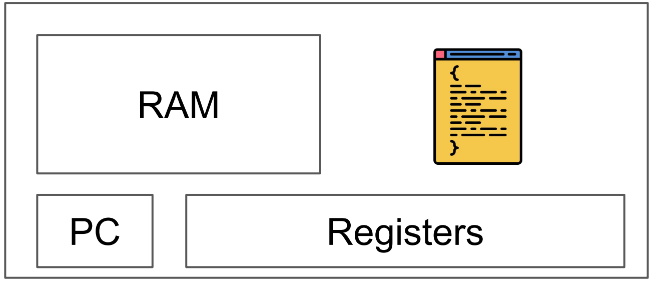
A universal circuit is large
It must be able to execute any operation at each step
e.g. RISC-V has 50 operations



```
switch(instruction) {
  case ADD: {...}
  case XOR: {...}
  ...
  case SHIFT: {...}
}
```

It's not only about instructions
The zkSNARK-Prover proves that:

- The memory is consistent throughout the entire computation
- The fetch and decode are correctly executed

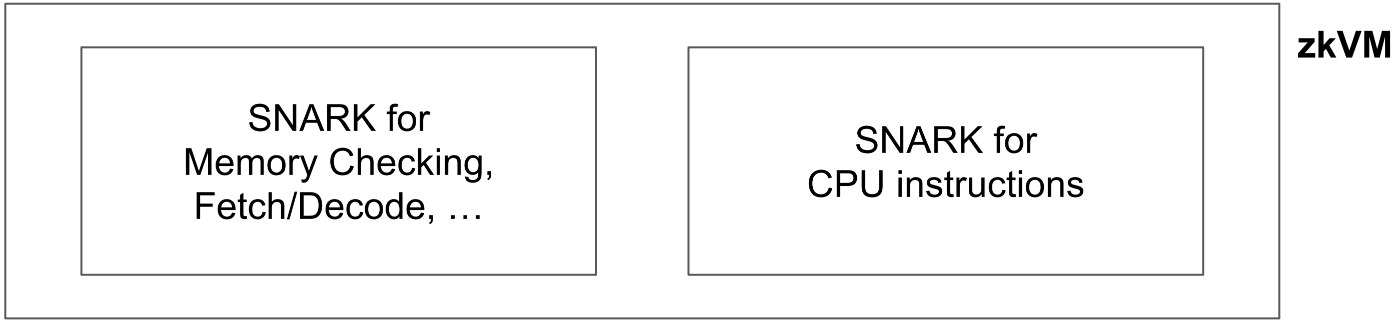


RAM

PC

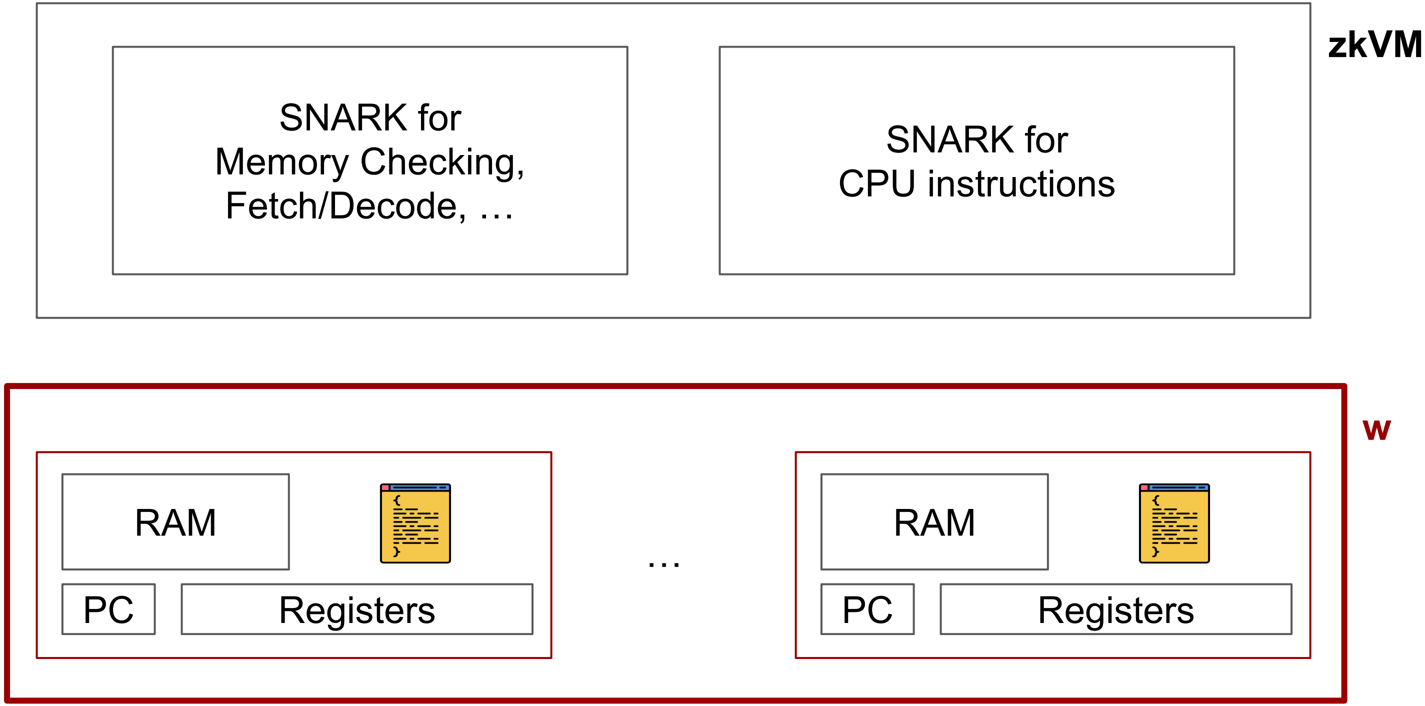
Registers

Joltish: a modular zkVM

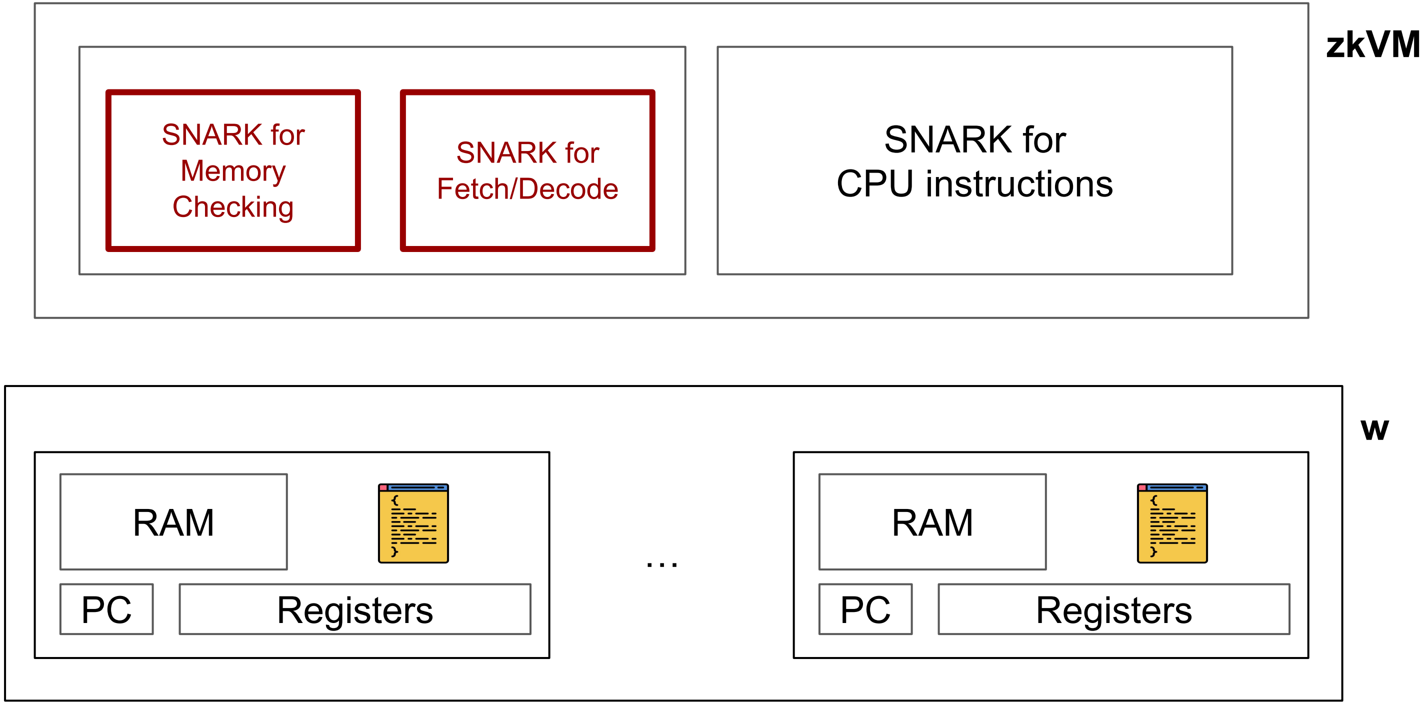


- Many zkVM's designs are modular (it's just natural)
- **Jolt** [AST'24] is the first zkVM based on *the lookup-singularity*
- Our Joltish: inspired by Jolt, but not quite Jolt

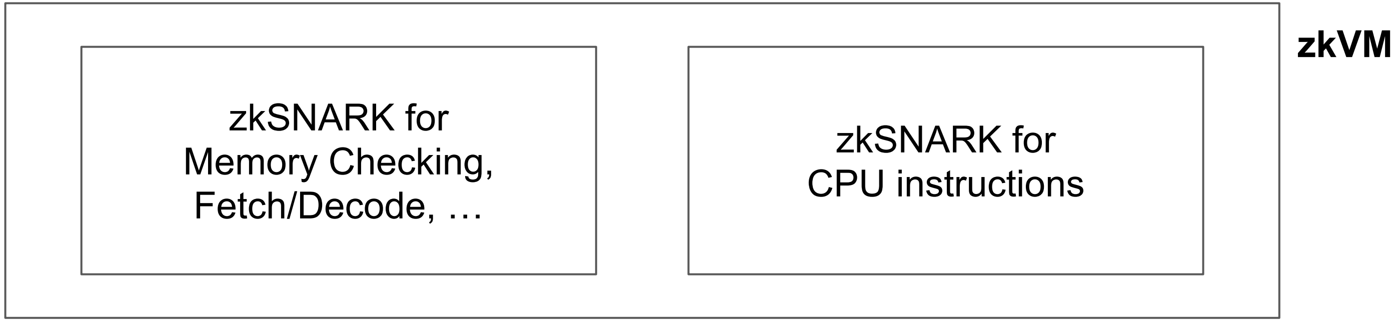
Joltish: a modular zkVM



Jolt~~ish~~: a modular zkVM



Joltish: a modular zkVM

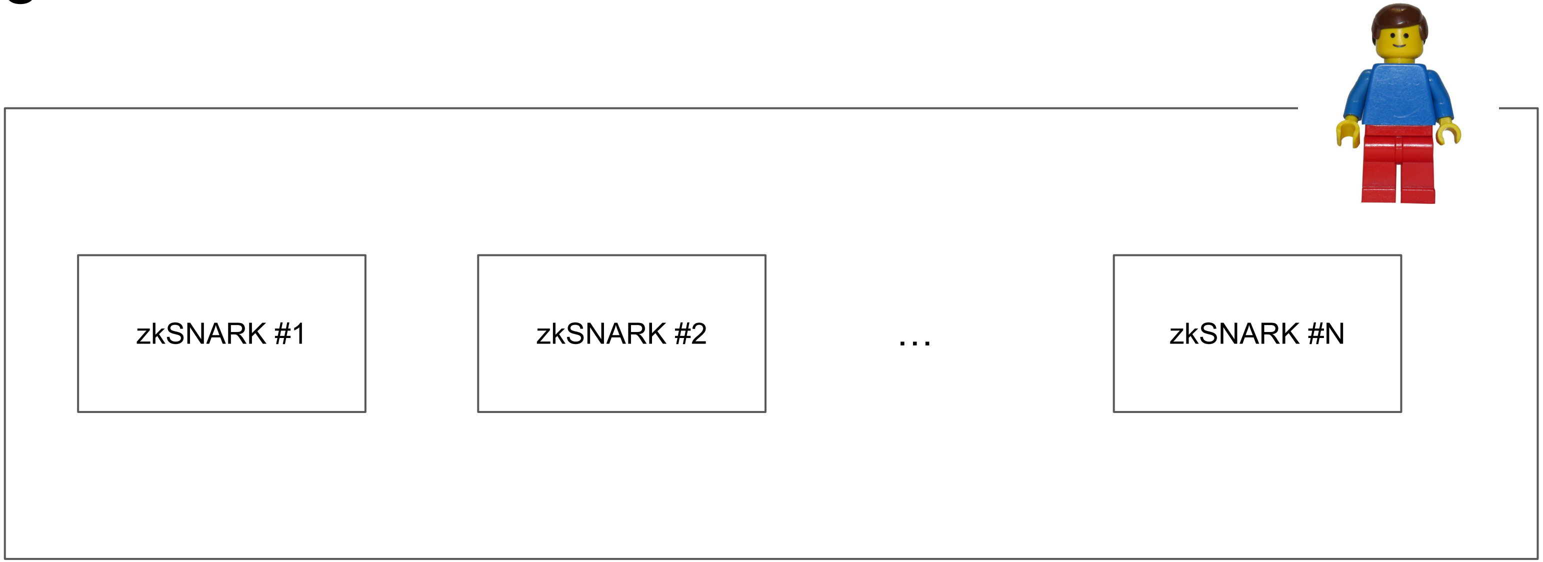

zkSNARK for Memory Checking, Fetch/Decode, …

zkSNARK for CPU instructions

zkVM

What are the conditions for the non-malleability of Joltish?

Lego-ish: a modular zkSNARK



zkSNARK #1

zkSNARK #2

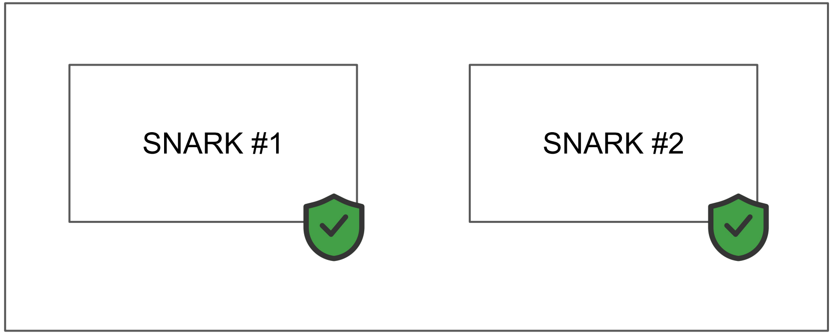
…

zkSNARK #N

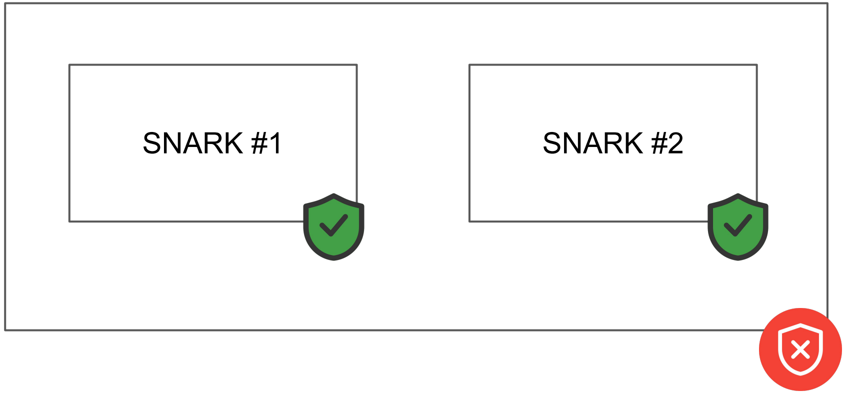
What are the conditions for the non-malleability of Joltish?

What are the conditions for the non-malleability of modular zkSNARKs?

Non-malleability challenges

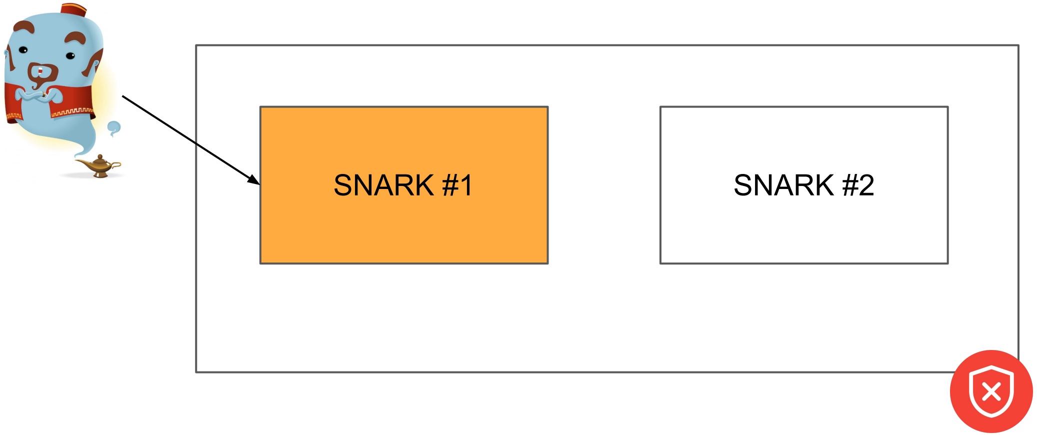


Non-malleability challenges



Copy & Paste attacks
Composition of non-malleable SNARK is not always secure

Non-malleability challenges



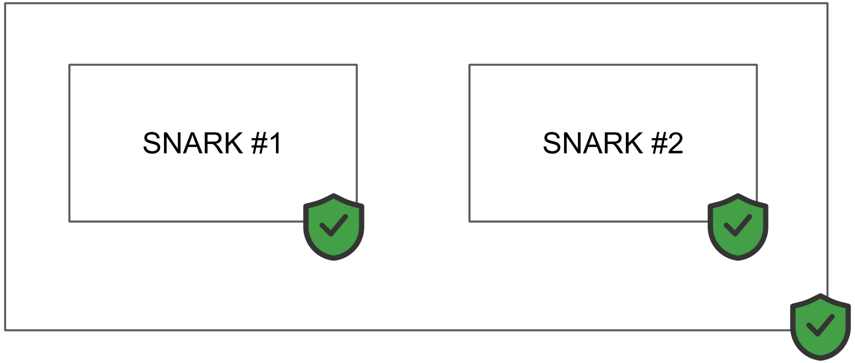
Copy & Paste attacks
Composition of non-malleable SNARK is not always safe!

Conjunction of Commit-and-Prove Relations

- A commit-and-prove relation (c, x ; w) in R iff
  - (1) P(x,w) = 1
  - (2) c = Commit(w)

- (Commit and Prove) Conjunction of R1 and R2 with shared witness
  - Instance (c, x1, x2)
  - Witness w
  - P1(x1, w) = 1 AND P2(x2, w) = 1 AND c = Commit(w)

Conjunction



R1(x1,w)=1 AND R2(x2,w)=1

Can we prove Non-Malleability for the conjunction with shared witness?

Yes! We give two (slightly different) Non-Malleable Compositions

Conjunction: First Case

Prover (x_1, x_2, w):

- commits w: as c = Commit(w)

- proves (c, x1; w) in R1 using zkSNARK #1

- proves that (c, x2; w) in R2 using zkSNARK #2

✓

Conjunction: First Construction

Prover (x_1, x_2, w):

- commits w: as c = Commit(w)

- proves (c, x1; w) in R1 using zkSNARK #1

- proves that (c, x2; w) in R2 using zkSNARK #2

zkSNARK #1

zkSNARK #2

Trapdoorless zero-knowledge, Sim-Extractable

Conjunction: Secon

- ZK-sim for Conjunction: compute honest π1, simulate π2
- Reduction to KS of SNARK#1 does not need of simulated proofs
- π1 could be malleable => SoK: any change on π1 needs a new signature π2′

P(x_1, x_2, w):

- - commits w: as c

- proves (c, x1; w) in R1 using zkSNARK #1

- proves that (c, x2; w) in R2 using zkSNARK #2

and π2 is also a Signature-of-Knowledge for π1

SNARK #1

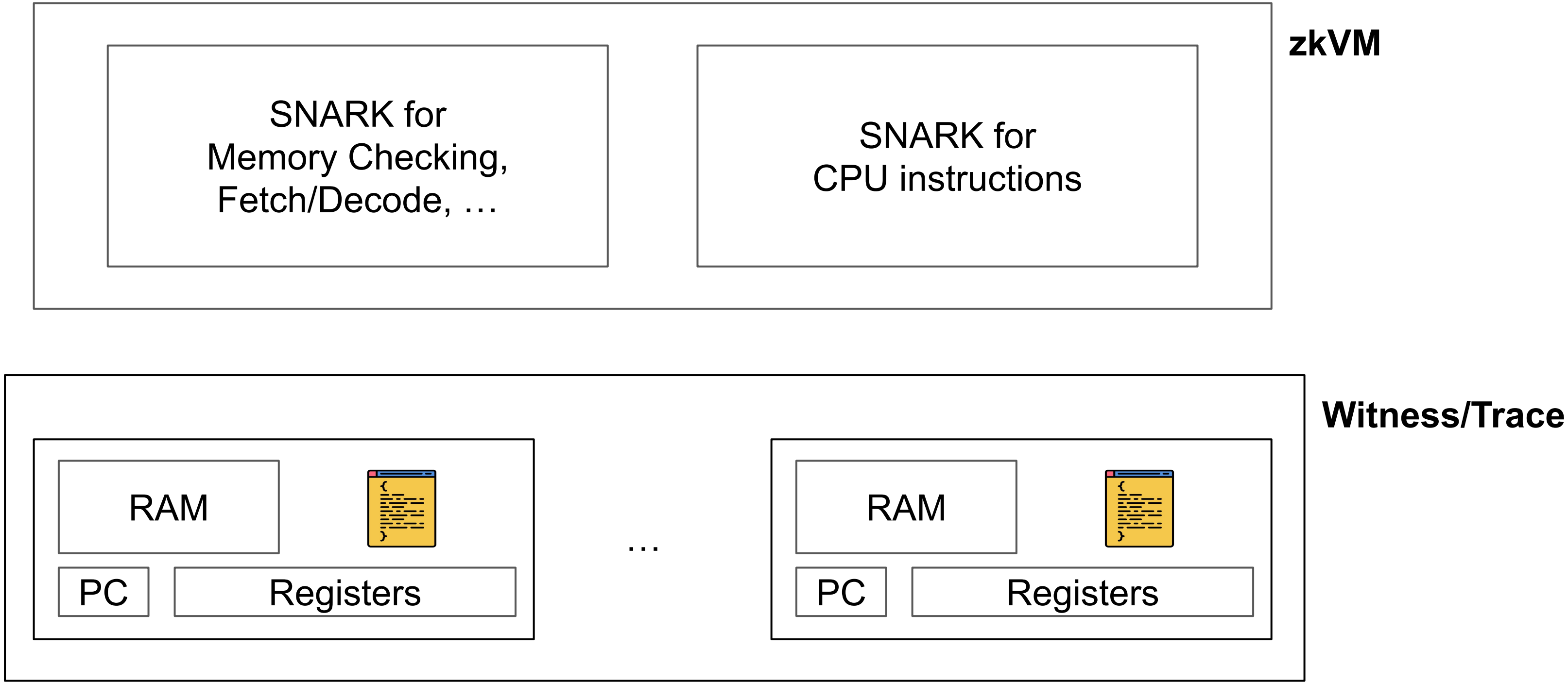
Witness-indistinguishability, knowledge-soundness,

efficient witness-computability *(easy to find w for x1)*

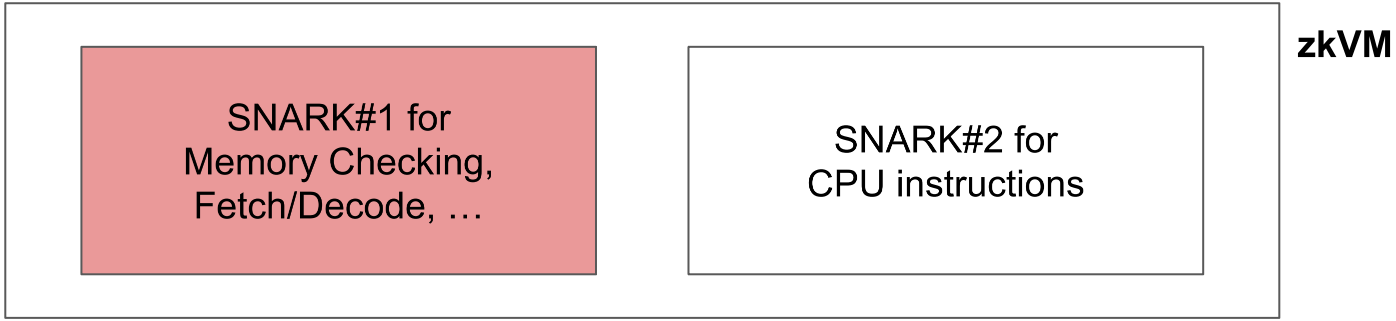
SNARK #2

Trapdoorless zero-knowledge, SoK (Sim-Extractable)

On the Non-Malleability of Joltish

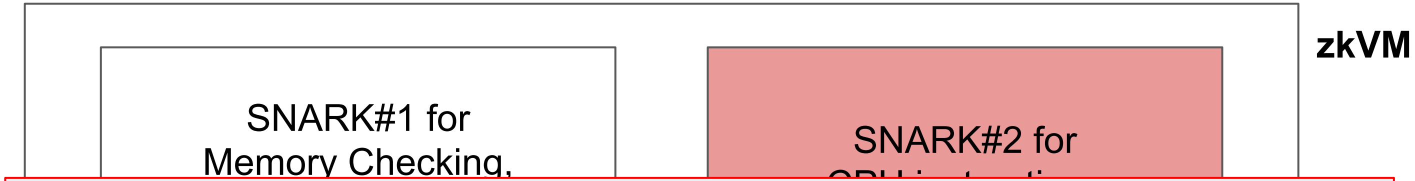


On the Non-Malleability of Joltish



- Knowledge-sound SNARK for Memory Checking, Fetch/Decode, …

On the Non-Malleability of Joltish

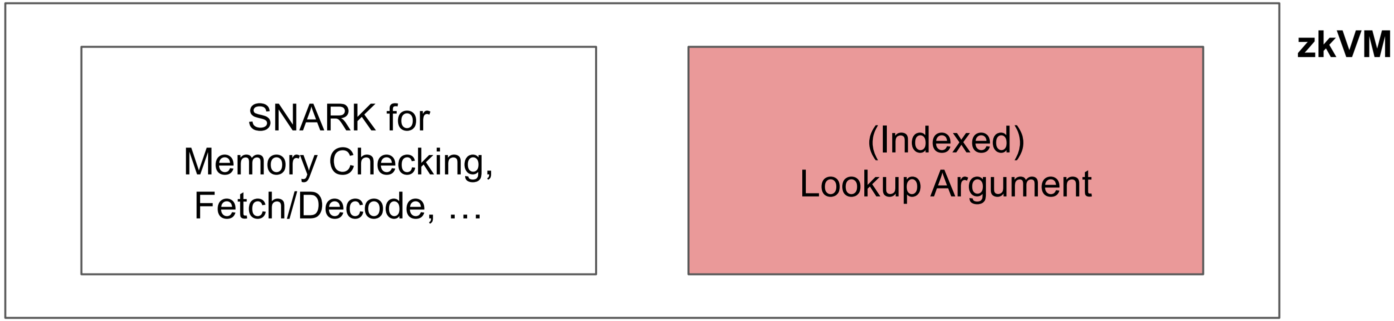


- Statement: (P,x,y) s.t. P(x)=y on RISC-V architecture.
- ZK-Simulator for (P,x,y) s.t. P(x)=y’≠ y on RISC-V architecture
  1) Hack the instructions set s.t. P(x)=y
  2) Run P(x)=y on hacked architecture to obtain program trace W
  3) W good trace for (P,x,y) for SNARK#1 => Run Prover#1
  4) W not good witness for P(x)=y for SNARK#2 => Simulated Proof

- K

- S

The Lookup Singularity based zkVM



- Knowledge-sound SNARK for Memory Checking, Fetch/Decode, …

- Sim-Ext ~~zkSNARK for CPU instructions~~ Lookup Argument

Lookup Arguments

- Lookup Arguments prove that committed vector F is sub-vector of big table T
  - |F| << |T|
  - prover complexity is proportional to |F|
- They can handle very big table:
  - as big as truth tables of all RISC-V instructions
- Lookup Argument in Jolt is Lasso [STW'24]

zk-Lasso

- Define a zero-knowledge version of Lasso
- Prove Sim-Extractability: based on the framework of [FaustKMV12]

  (trapdoorless ZK + Unique Response + Special Soundness => Sim Ext)

- We extend and improve over [Dao and Grubbs23]
  - (Lasso is based on Spartan)

Future Works

- Non-Malleability of ~~zk~~SNARKs: non-malleability w/o simulation extractability?
- Non-Malleability for composition of Reduction-of-Knowledge: very natural!
- Non-Malleability of other Lookup Arguments?

Thank you

Truth table and lookups

XOR(x,y)

x	**y**	**out**
00	00	00
00	01	01
00	10	10
00	11	11
	…	
11	10	01
11	11	00

Truth table and lookups

XOR(x,y)

x	**y**	**out**
00	00	00
00	01	01
00	10	10
00	11	11
	…	
11	10	01
11	11	00

Checking that out = XOR(x,y) can be reduced to

check that (x,y,out) is in the truth table of the XOR

Truth table and lookups

XOR(x,y)

x	**y**	**out**
00	00	00
00	01	01
00	10	10
00	11	11
	…	
11	10	01
11	11	00

Checking that out = XOR(x,y) can be reduced to check that (x,y,out) is in the truth table of the XOR

For 64-bits operands, this table has 2^128 entries!

Truth table and lookups

XOR(x,y)

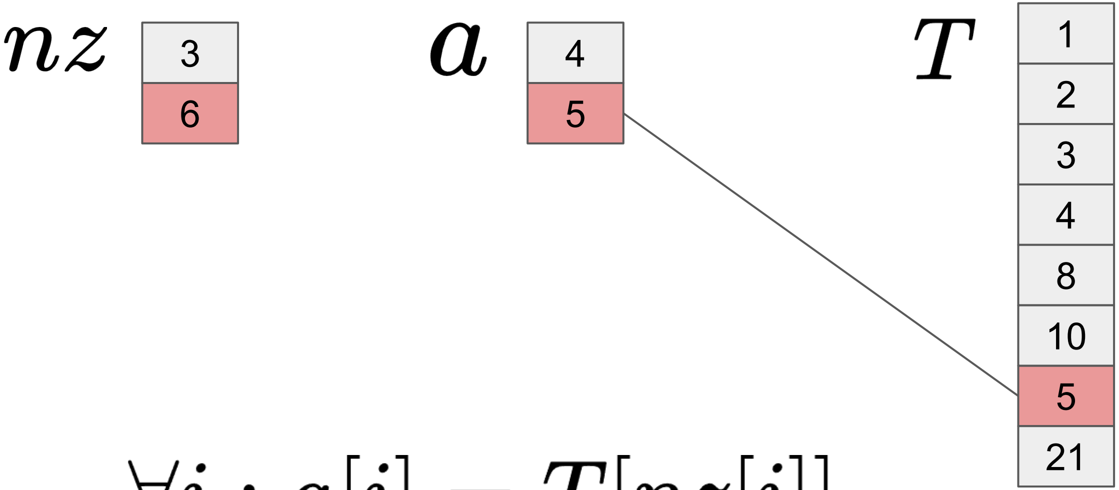
x	**y**	**out**
00	00	00
00	01	01
00	10	10
00	11	11
	…	
11	10	01
11	11	00

Checking that out = XOR(x,y) can be reduced to

check that (x,y,out) is in the truth table of the XOR

For 64-bits operands, this table has 2^128 entries!

Lasso in a nutshell



$$\forall i : a[i] = T[nz[i]]$$

Lasso in a nutshell

0	0	0	1	0	0	0	0
0	0	0	0	0	0	1	0

M

1
2
3
4
8
10
5
21

T

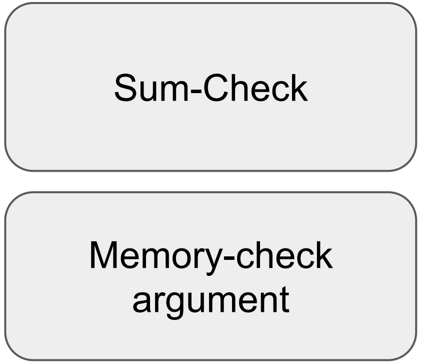
4
5

a

Lasso in a nutshell

+ Reduces the matrix-vector product to a Sum-Check
+ Performs a memory-checking argument
+ Supports gigantic tables as long as they are structured

- The scheme is not zero-knowledge



Sum-Check

Memory-check argument

Non-Malleability of Lasso

Reduce non-malleability to Special Soundness + k-ZK + k-UR

- The simulator can reprogram the RO only at the k-th round
- Proofs are unique after the k-th round
- Witness can be extracted from a sufficient number of proofs… or we can break dlog!
- Use rewinding to extract

References

[DG23] Quang Dao and Paul Grubbs. Spartan and bulletproofs are simulation-extractable (for free!). EUROCRYPT 2023

[FFK+23] Antonio Faonio, Dario Fiore, Markulf Kohlweiss, Luigi Russo, and Michal Zajac. From polynomial IOP and commitments to non-malleable zkSNARKs. TCC 2023

[GKK+22] Chaya Ganesh, Hamidreza Khoshakhlagh, Markulf Kohlweiss, Anca Nitulescu, and Michal Zajac. What makes fiat-shamir zksnarks (updatable SRS) simulation extractable? SCN 2022

[GOP+22] Chaya Ganesh, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. Fiat-shamir bulletproofs are non-malleable (in the algebraic group model). EUROCRYPT 2022

[KPT23] Markulf Kohlweiss, Mahak Pancholi, and Akira Takahashi. How to compile polynomial IOP into simulation-extractable SNARKs: A modular approach. TCC 2023

[Lib24] Benoit Libert. Simulation-Extractable KZG Polynomial Commitments and Applications to HyperPlonk. PKC 202

Additional notes

Our results on Legoish SNARKs

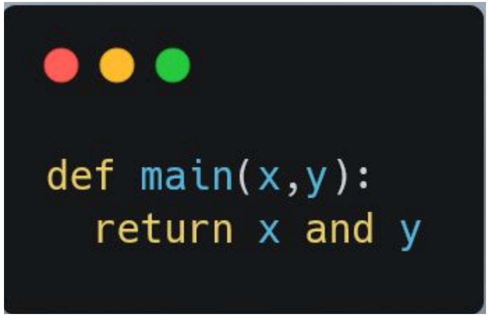
Conjunction (with shared witness)

- If two schemes are non-malleable then their composition is also non-malleable

- If the first scheme is knowledge-sound, witness-indistinguishable and witness-samplable, and the second one is a SoK, then their composition is non-malleable

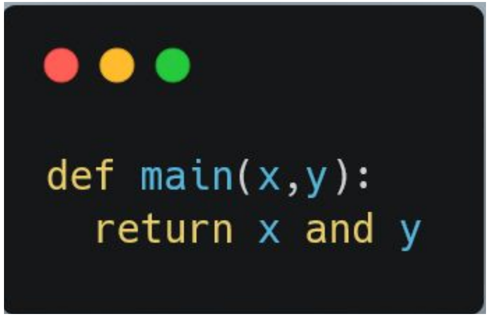
Similar results hold for **Functional composition**

From program to SNARK language



```
def main(x,y):
  return x and y
```

From program to SNARK language



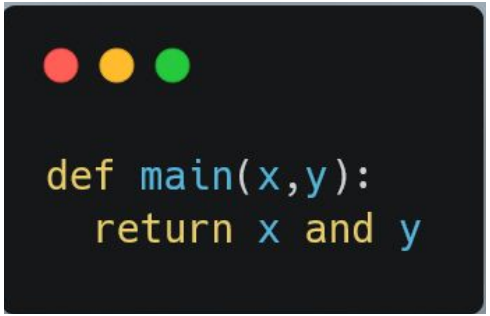
```
def main(x,y):
  return x and y
```

Polynomial Constraints

Check that:

- $x(x-1) = 0$
- $y(y-1) = 0$
- $x*y = out$

From program to SNARK language



```
def main(x,y):
  return x and y
```

Polynomial Constraints

Check that:

- **$x(x-1) = 0$**
- **$y(y-1) = 0$**
- **$x^*y = out$**

Lookup Constraint

Check that **(x,y,out) belongs to the AND table**

zkSNARKs based on Lookup Singularity

Lookup Singularity

Transform arbitrary computer program into “circuits” that only perform lookups

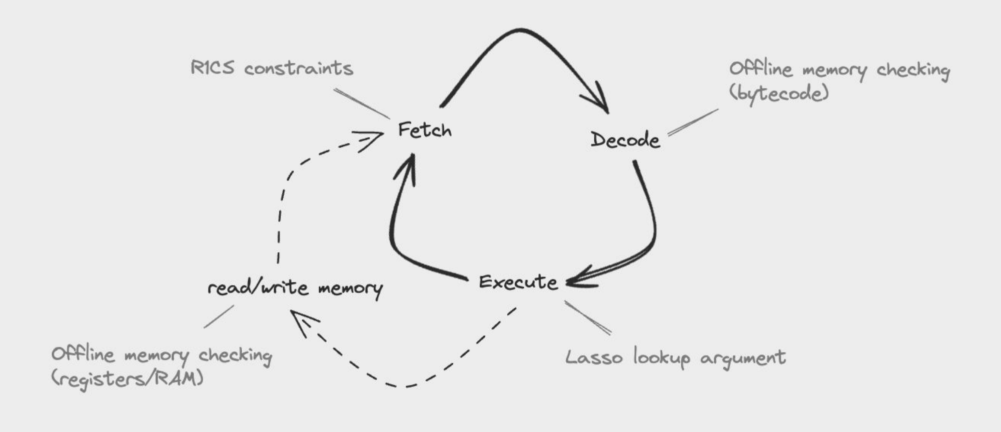
zkSNARKs based on Lookup Singularity

Lookup Singularity

Transform arbitrary computer program into “circuits” that only perform lookups

Jolt zkVM

- Repeatedly execute the fetch-decode-execute logic of its instruction set architecture (RISC-V)
- Perform reads and writes to RAM


R1CS constraints
Fetch
Decode
Offline memory checking (bytecode)
read/write memory
Execute
Offline memory checking (registers/RAM)
Lasso lookup argument

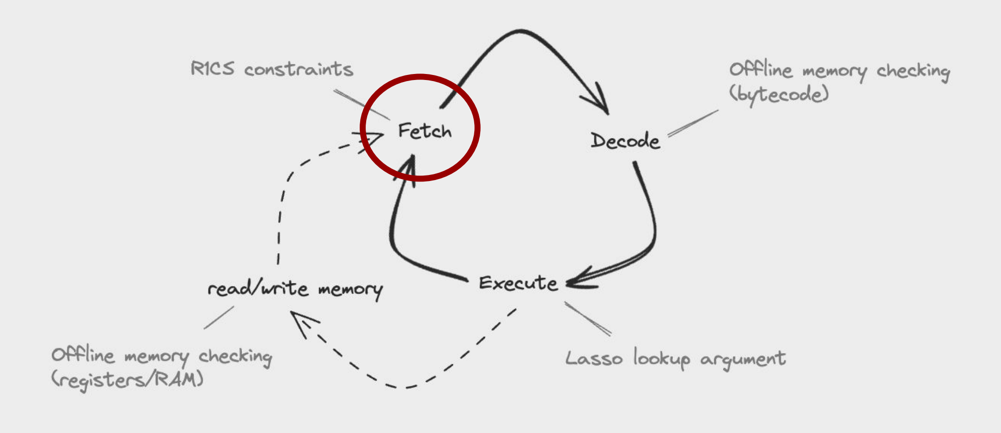
zkSNARKs based on Lookup Singularity

Lookup Singularity

Transform arbitrary computer program into “circuits” that only perform lookups

Jolt zkVM

- Repeatedly execute the fetch-decode-execute logic of its instruction set architecture (RISC-V)
- Perform reads and writes to RAM



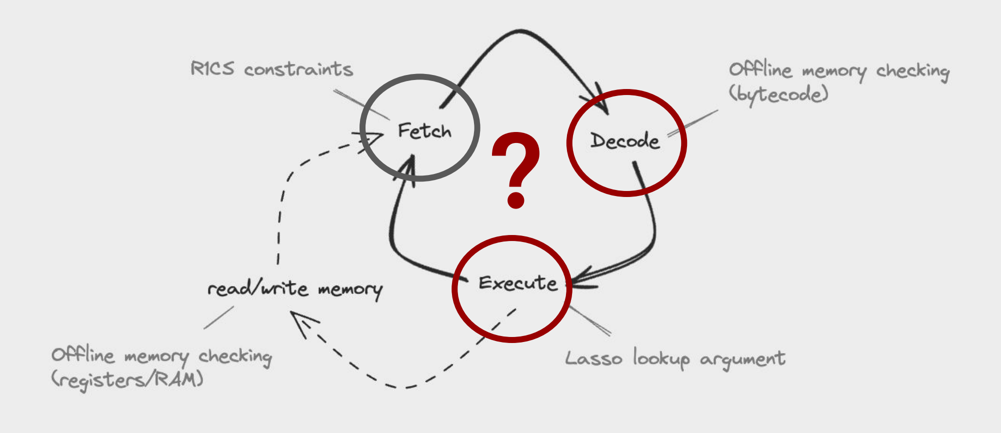
zkSNARKs based on Lookup Singularity

Lookup Singularity

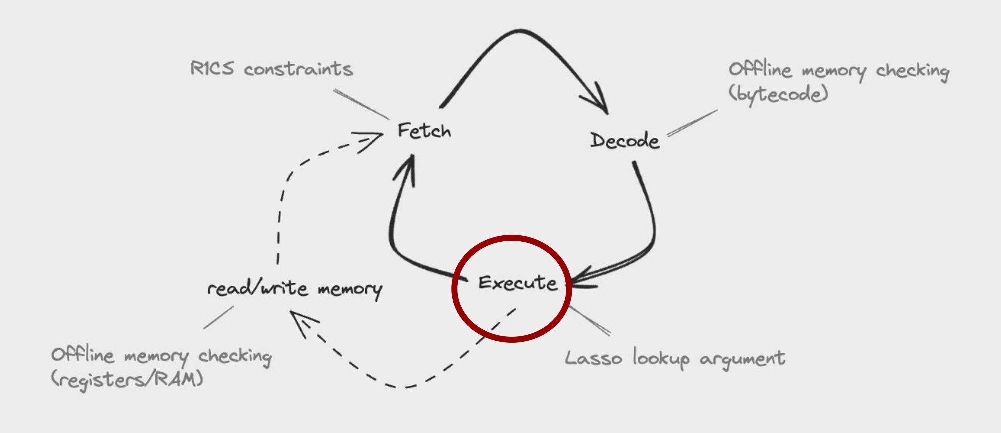
Transform arbitrary computer program into “circuits” that only perform lookups

Jolt zkVM

- Repeatedly execute the fetch-decode-execute logic of its instruction set architecture (RISC-V)
- Perform reads and writes to RAM


R1CS constraints
Fetch
Decode
Offline memory checking (bytecode)
?
Execute
read/write memory
Offline memory checking (registers/RAM)
Lasso lookup argument

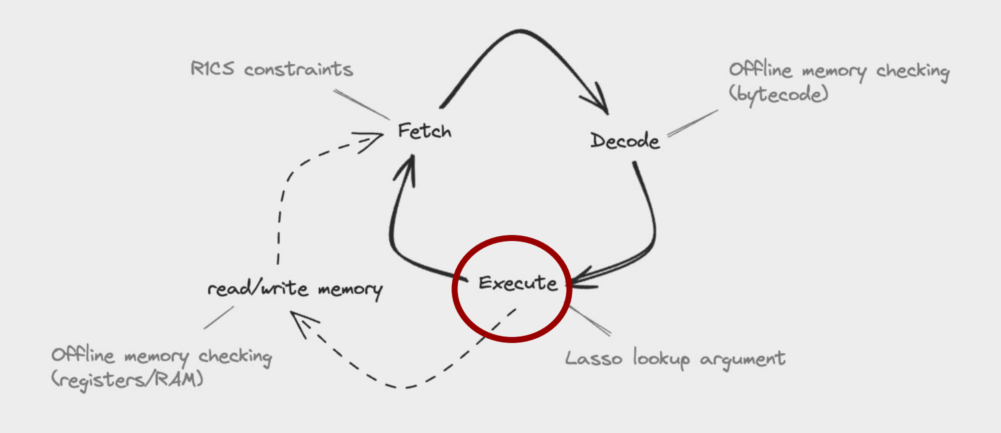
So what are our results?



So what are our results?

A zero-knowledge version of Lasso

- ○ Non-malleable under minimum assumptions (dlog+RO)
- ○ Comparably fast


R1CS constraints
Fetch
Decode
Offline memory checking (bytecode)
read/write memory
Execute
Offline memory checking (registers/RAM)
Lasso lookup argument

Credits

All images used in this presentation are being used for educational purposes in accordance with the principles of fair use. The copyright of these images remains with their respective owners. No infringement is intended.