Scroll & zkEVM & Proving system

Ye Zhang

Scroll

What is Scroll?

A scaling solution for Ethereum

What is Scroll?

An EVM-equivalent zk-Rollup

The technical workflow

Scroll

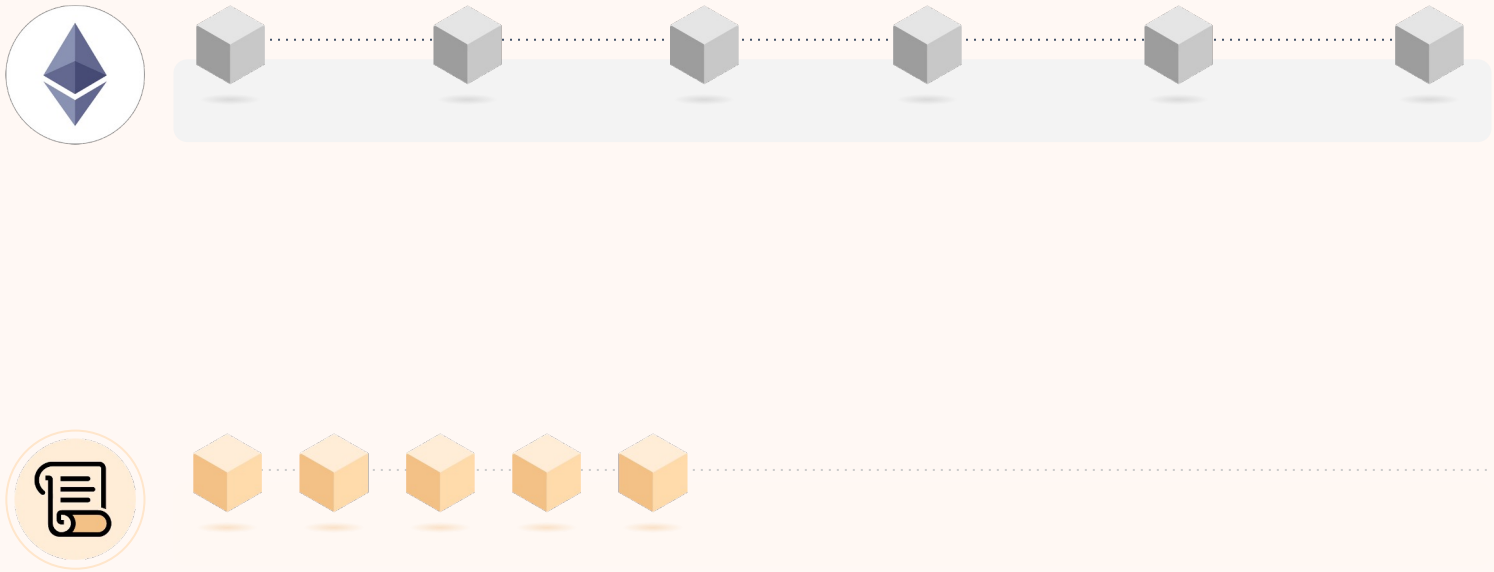
Pre-committed (1-2s)

Committed (minutes)

Finalized (10+minutes)

The technical workflow

Scroll



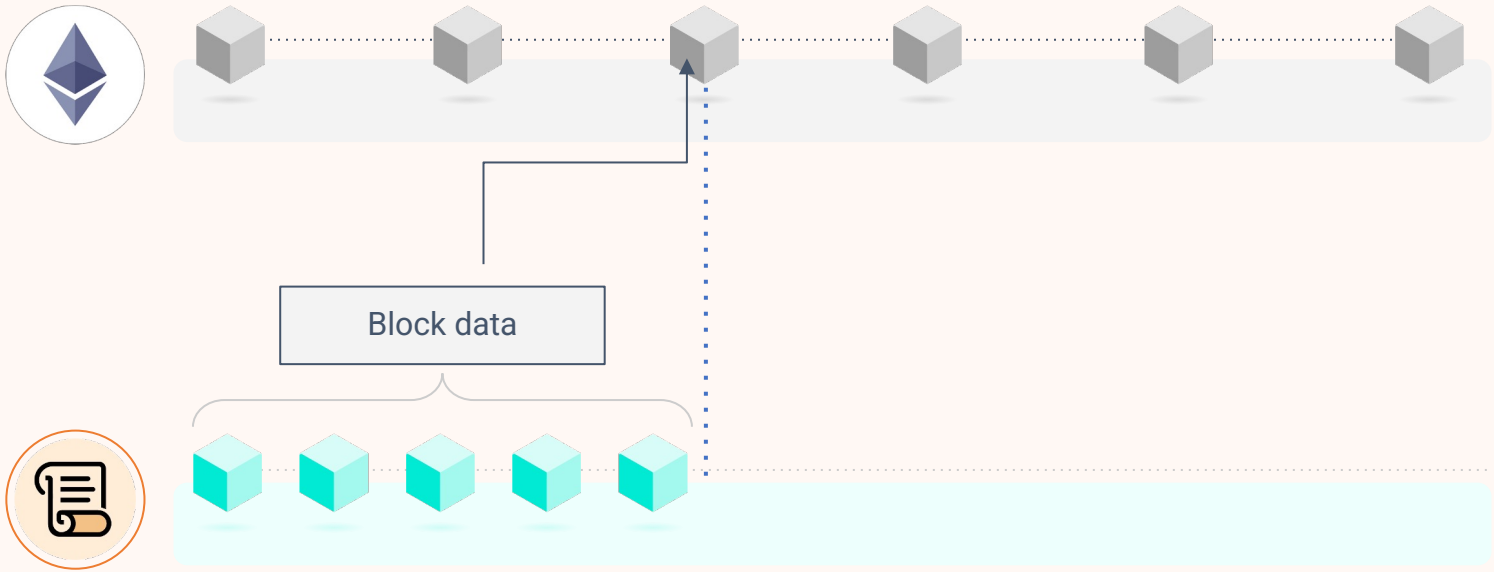
Pre-committed (1-2s)

Committed (minutes)

Finalized (10+minutes)

The technical workflow

Scroll



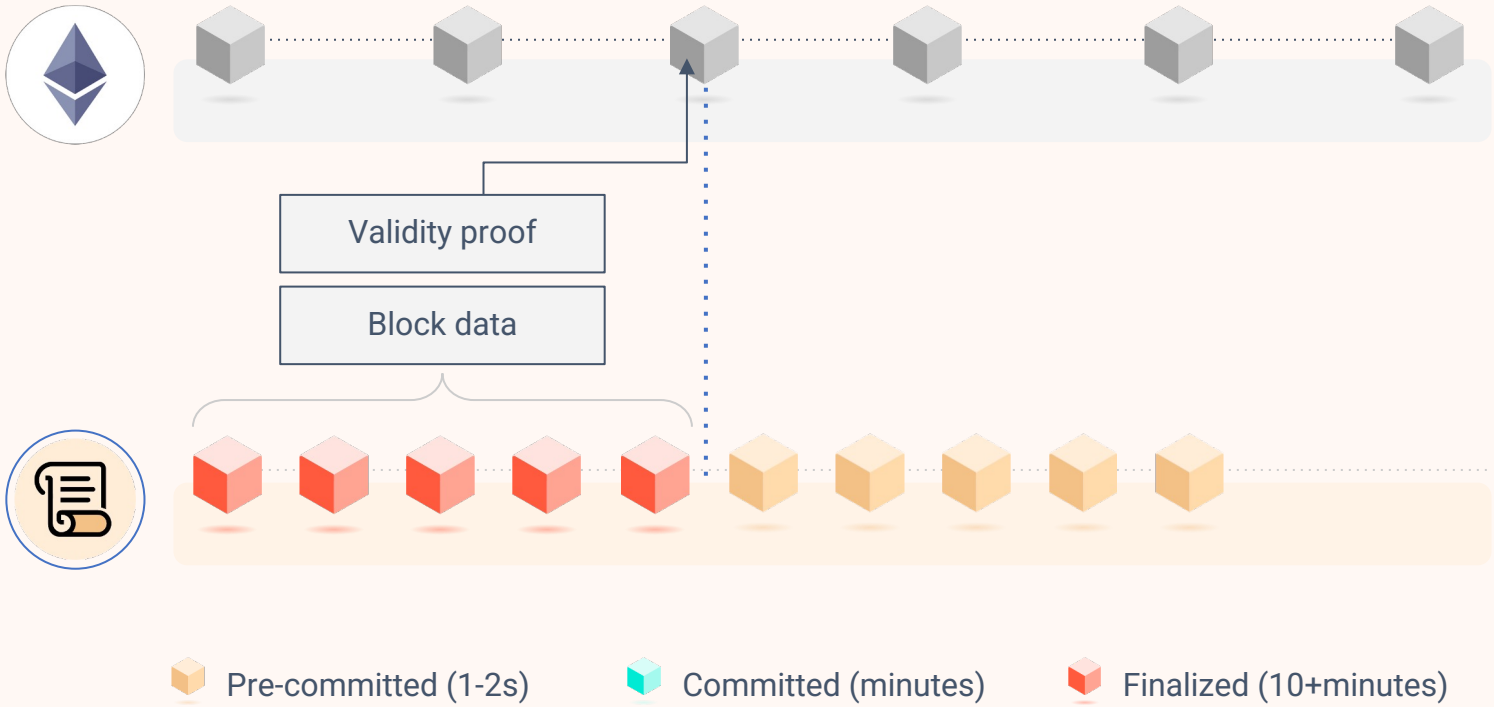
Pre-committed (1-2s)

Committed (minutes)

Finalized (10+minutes)

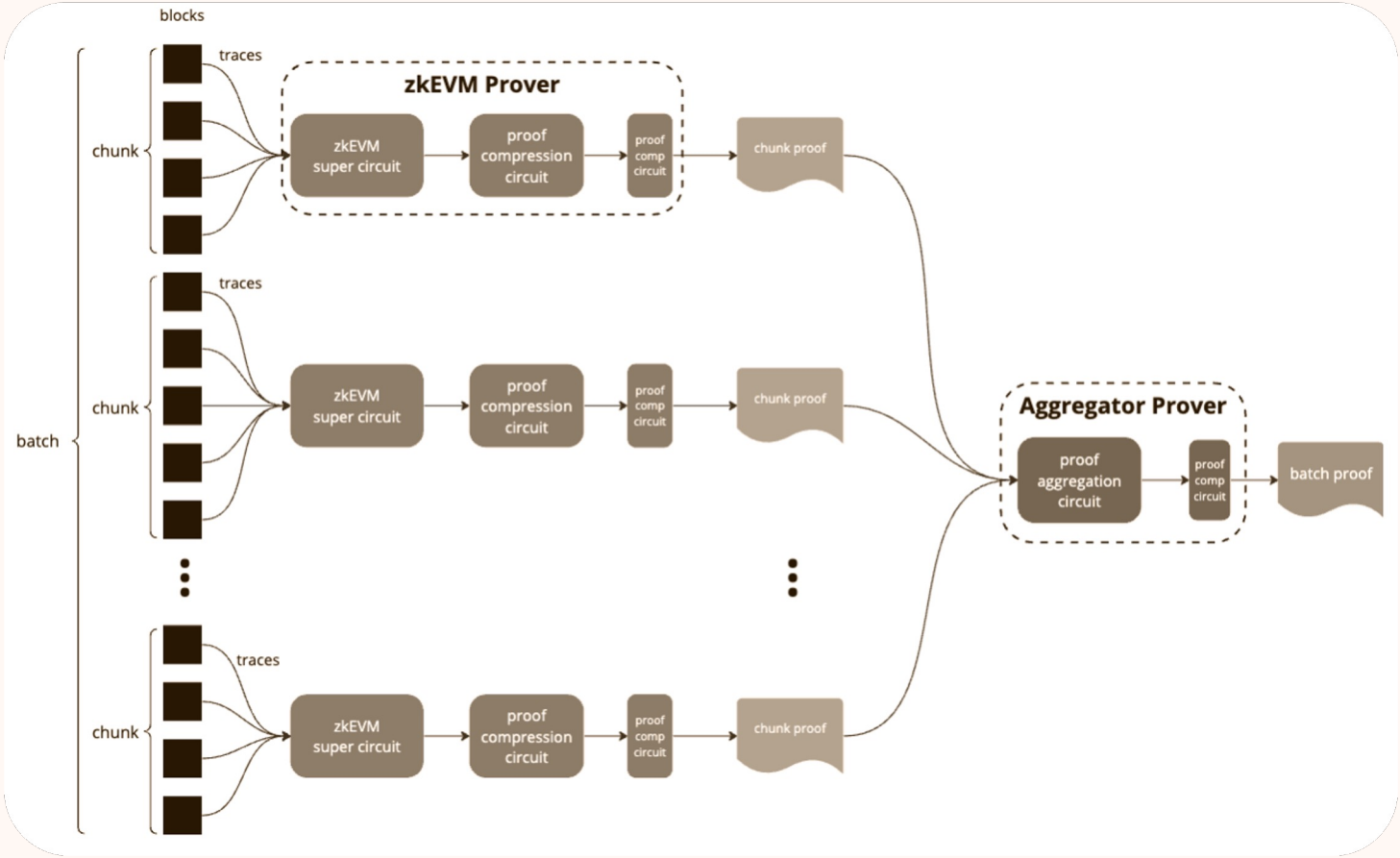
The technical workflow

Scroll



The technical workflow

Scroll



Build for users and developers

Scroll

For users and devs:

Scroll = Ethereum

But cheaper and faster

with a higher throughput

Copy solidity code from dApp

Paste code into Scroll interface

Unlock all the benefits of Scroll's EVM-equivalent zk-rollup

Build for users and developers

Scroll

For users and devs:

Scroll = Ethereum

But cheaper and faster

with a higher throughput

Hardhat

Foundry

Remix

EthersJS

Brownie

Truffle

The journey with community

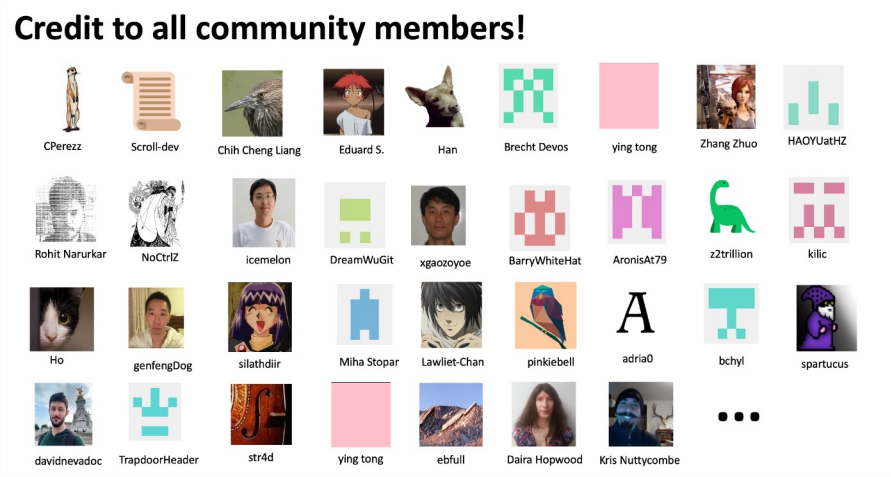
Scroll

Privacy & Scaling Explorations (PSE)
A zk working group within the Ethereum Foundation

The journey with community

Scroll

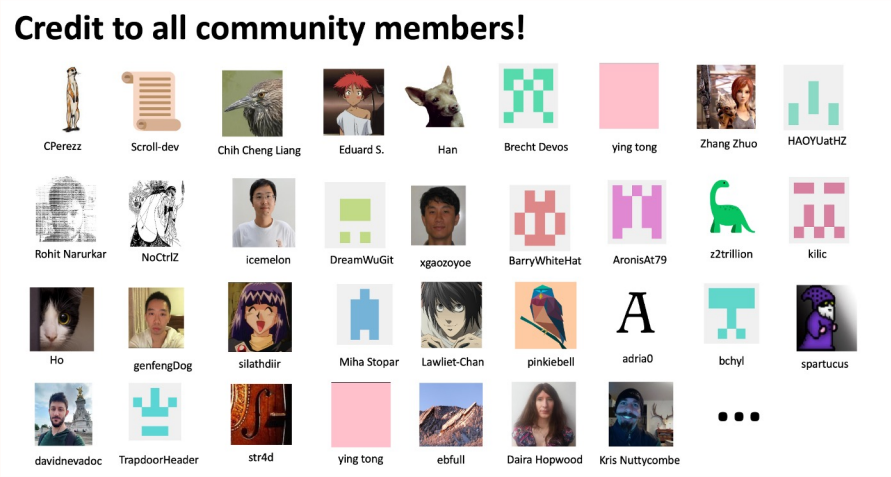
Privacy & Scaling Explorations (PSE)
A zk working group within the Ethereum Foundation



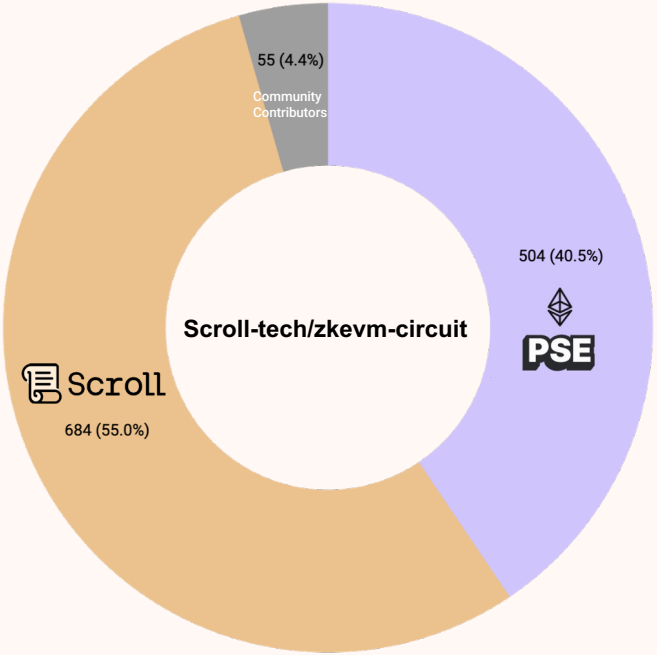
The journey with community

Scroll

Privacy & Scaling Explorations (PSE)
A zk working group within the Ethereum Foundation



Value: Openness and Community-driven



- Full transparency – anyone can review code
- The most open zkEVM from day 1
- The most community-driven zkEVM
- The only zkEVM that supports ecPairing

Security first

Scroll

Security Audit

Trail of Bits OpenZeppelin Zellic Kalos ABDK

Replay Txs from different networks

Standard EVM test vectors

Fuzzing for zkEVM opcodes

In-house Security Team

Red Team Blue Team

Reuse Geth

Fully open-source

Many more...

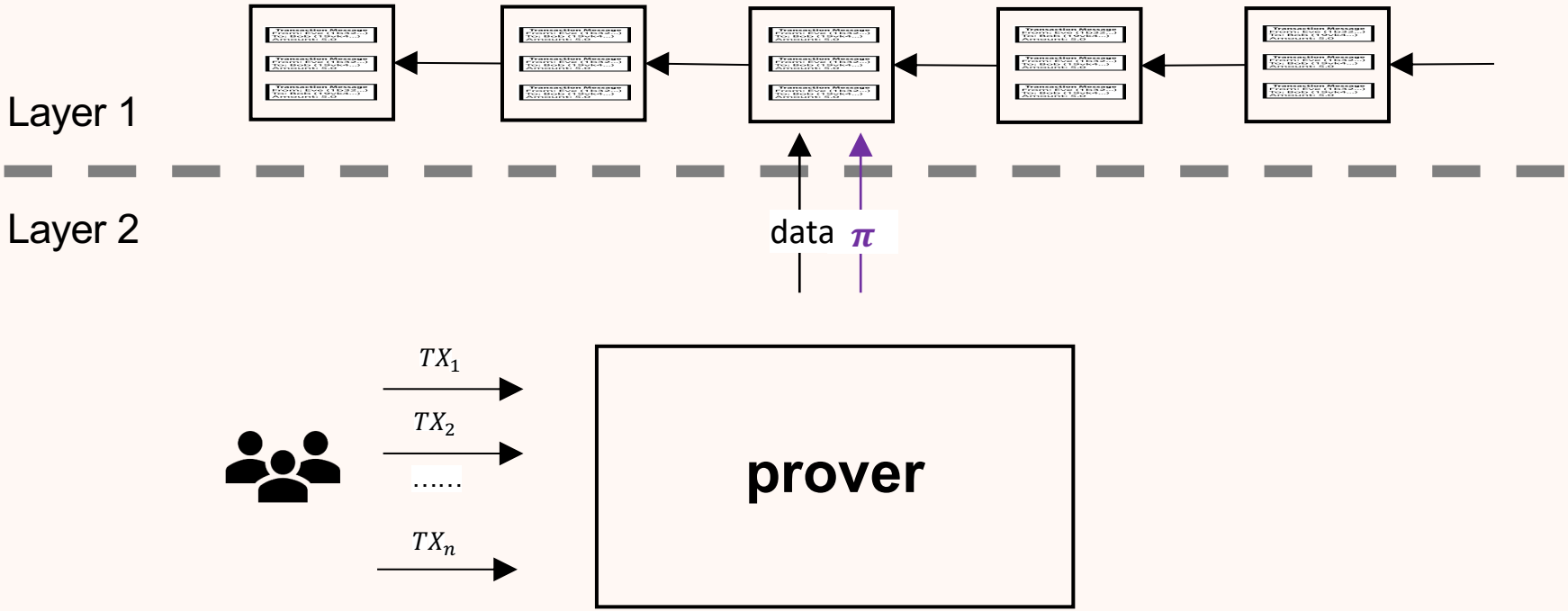
Outline

Scroll

- Background & motivation
- zkEVM circuit arithmetization
- zkEVM prover optimization
- Something interesting to share

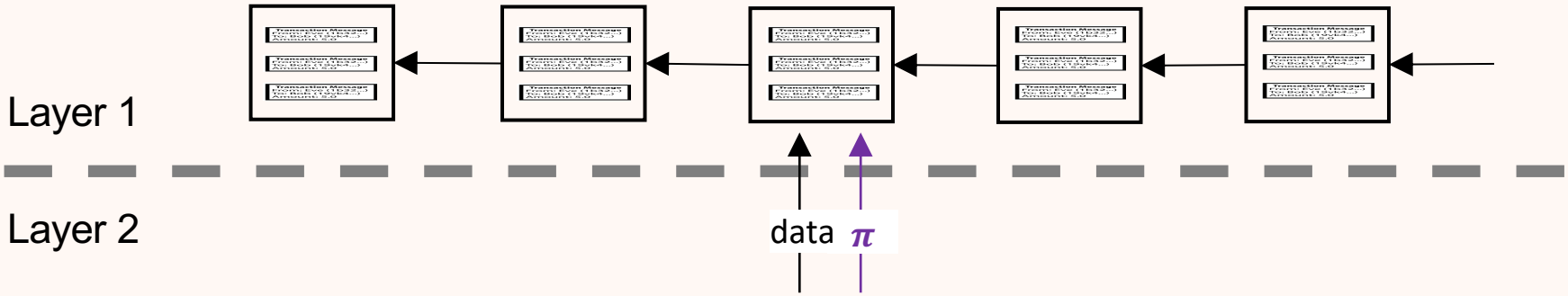
Zk-Rollup

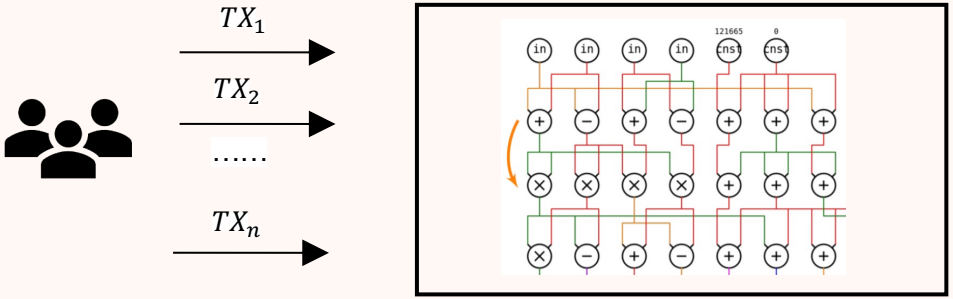
Scroll



However, …

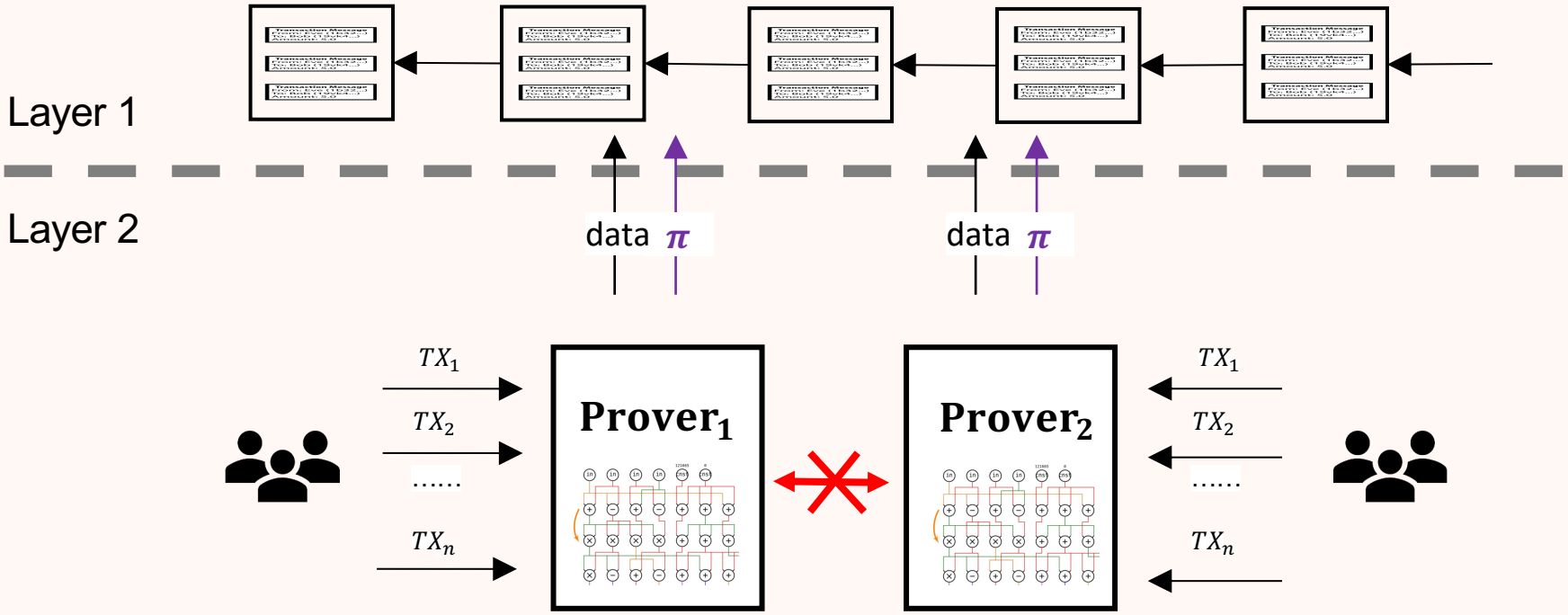
Scroll





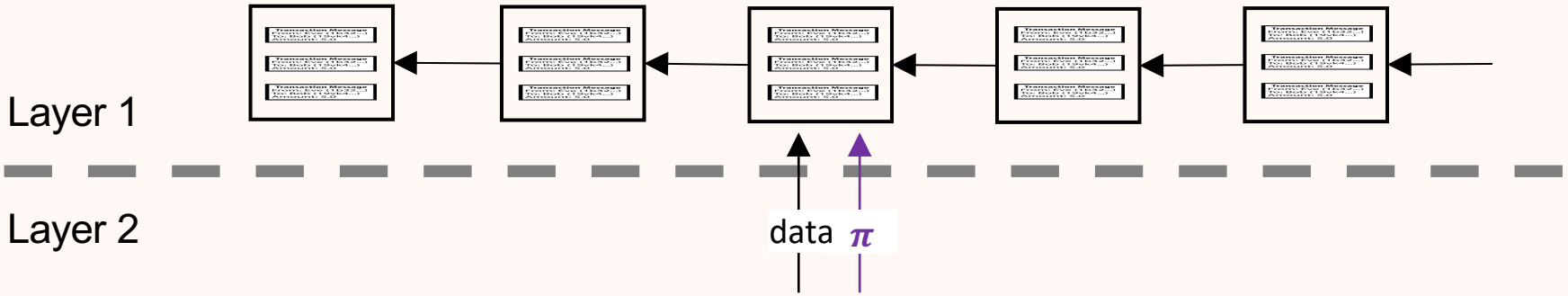
However, …

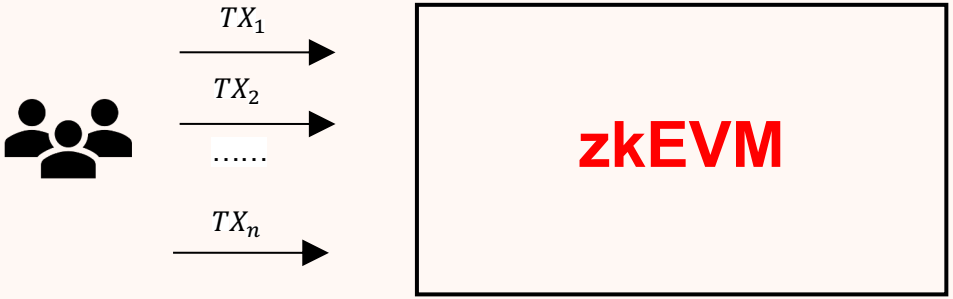
Scroll



Zk-Rollup

Scroll

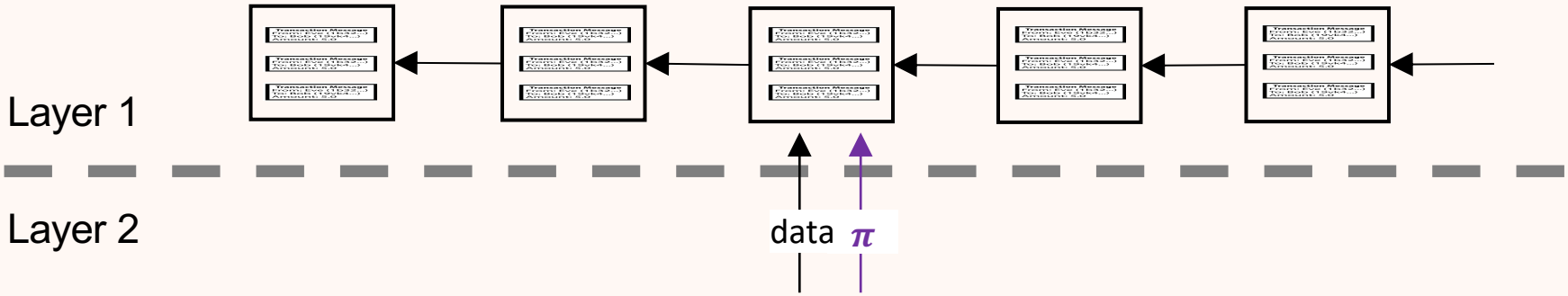


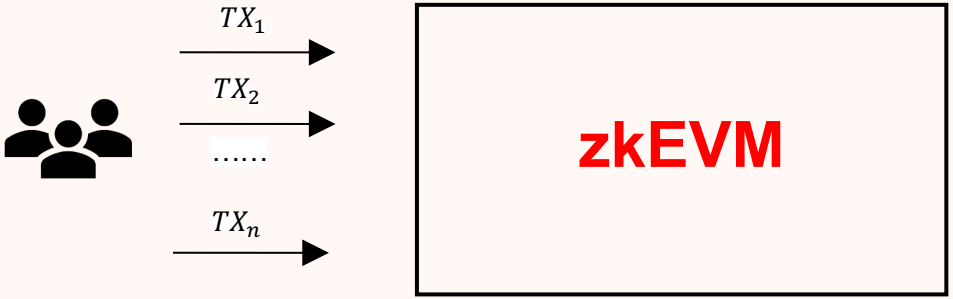


- Developer friendly
- Composability
- Hard to build ☹
- Large proving overhead ☹

Zk-Rollup

Scroll





- Polynomial commitment
- Lookup + Custom gate
- Hardware acceleration
- Recursive proof

zkEVM flavors (by Justin Drake)

Scroll

- **Language level**
  Transpile an EVM-friendly language (Solidity or Yul) to a SNARK-friendly VM which differs from the EVM. This is the approach of Matter Labs and Starkware.

- **Bytecode level**
  Interpret EVM bytecode directly, though potentially producing different state roots than the EVM, e.g. if certain implementation-level data structures are replaced with SNARK-friendly alternatives. This is the approach taken by Scroll, Hermez, and Consensys.

- **Consensus level**
  Target full equivalence with EVM as used by Ethereum L1 consensus. That is, it proves validity of L1 Ethereum state roots. This is part of the "zk-SNARK everything" roadmap for Ethereum.

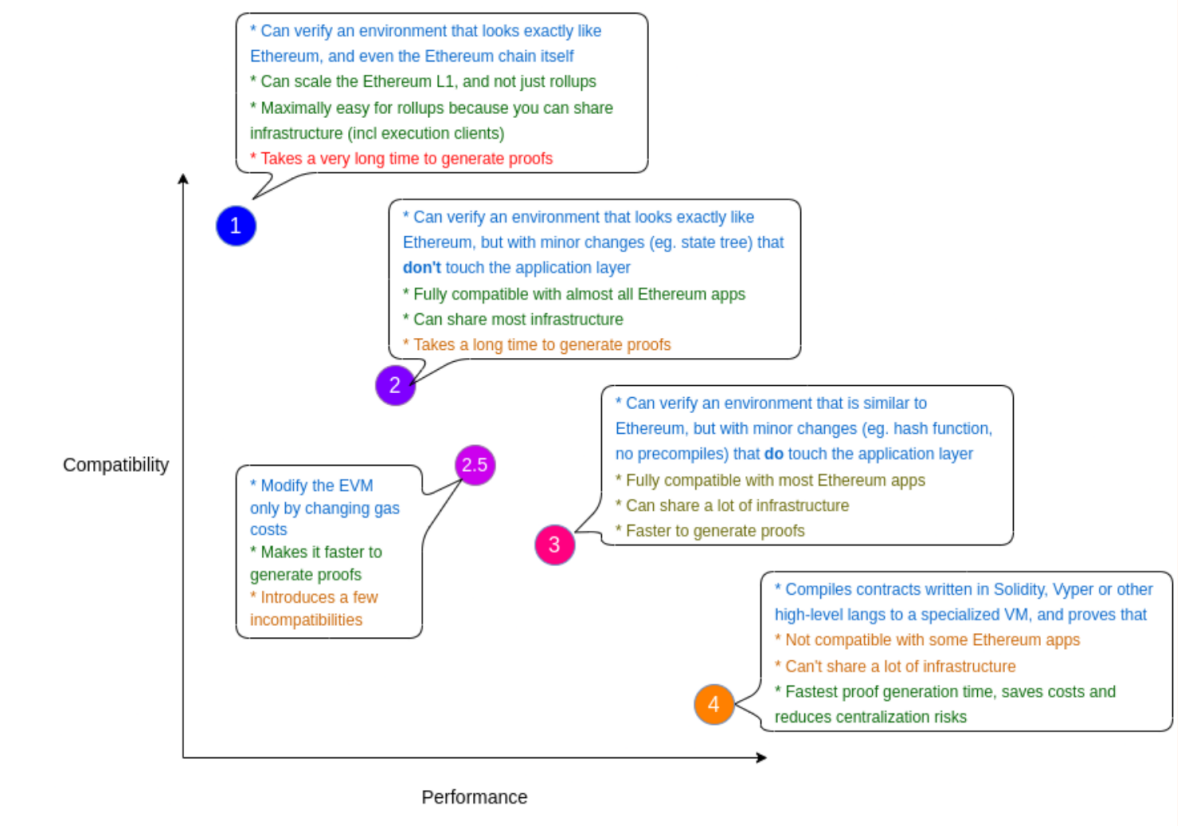
zkEVM flavors (by Justin Drake)

Scroll

- **Language level**
  Transpile an EVM-friendly language (Solidity or Yul) to a SNARK-friendly VM which differs from the EVM. This is the approach of Matter Labs and Starkware.
- **Bytecode level**
  Interpret EVM bytecode directly, though potentially producing different state roots than the EVM, e.g. if certain implementation-level data structures are replaced with SNARK-friendly alternatives. This is the approach taken by **Scroll**, Hermez, and Consensys.
- **Consensus level**
  Target full equivalence with EVM as used by Ethereum L1 consensus. That is, it proves validity of L1 Ethereum state roots. This is part of the "zk-SNARK everything" roadmap for Ethereum.

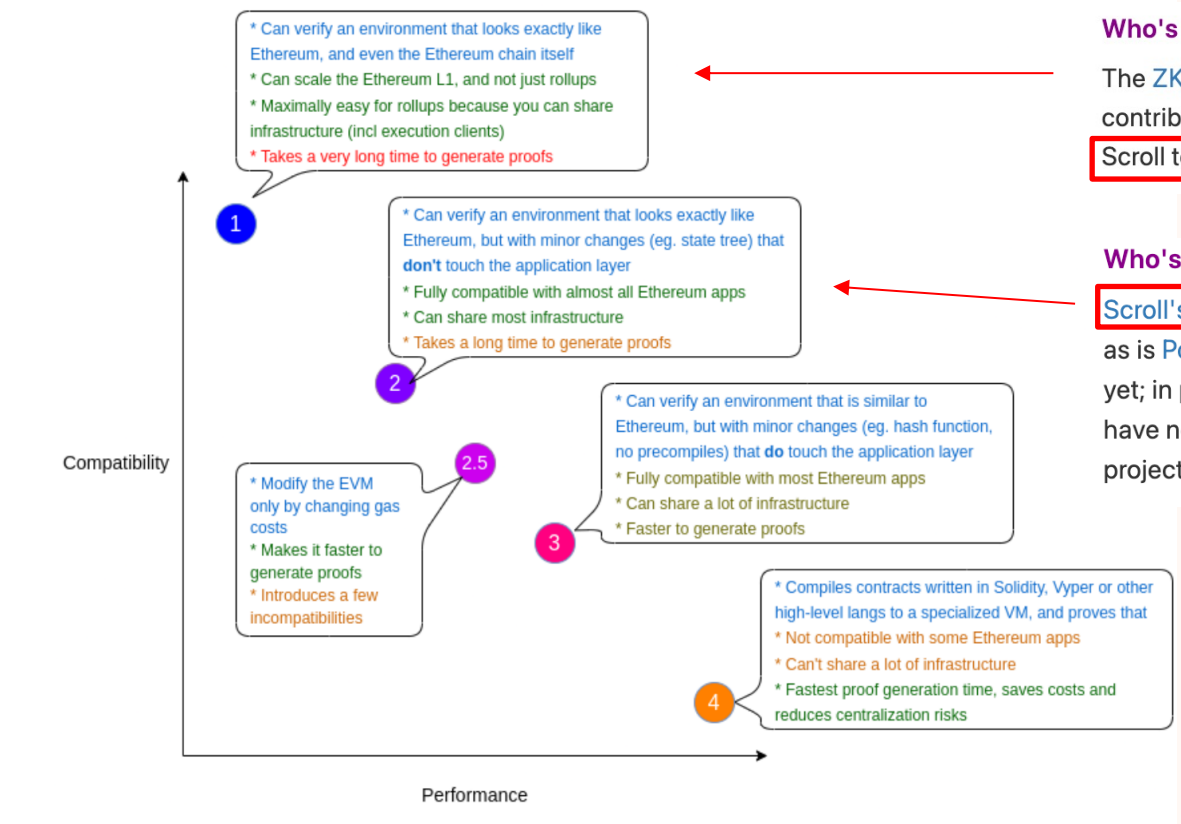
zkEVM flavors (by Vitalik)

Scroll



zkEVM flavors (by Vitalik)

Scroll



Who's building it?

The ZK-EVM Community Edition (bootstrapped by community contributors including Privacy and Scaling Explorations, the Scroll team, Taiko and others) is a Tier 1 ZK-EVM.

Who's building it?

Scroll's ZK-EVM project is building toward a Type 2 ZK-EVM, as is Polygon Hermez. That said, neither project is quite there yet; in particular, a lot of the more complicated precompiles have not yet been implemented. Hence, at the moment both projects are better considered Type 3.

Outline

Scroll

- Background & motivation
- zkEVM circuit arithmetization
- zkEVM prover optimization
- Something interesting to share

The workflow of zero-knowledge proof

Scroll

Program

```python
def hcf(x, y):
    if x > y:
        smaller = y
    else:
        smaller = x

    for i in range(1,smaller + 1):
        if((x % i == 0) and (y % i == 0)):
            hcf = i

    return hcf
```

R1CS
Plonkish
AIR

Constraints

```
x * x == var1
var1 * x == y
(y+x) * 1 == var2
(var2+5) * 1 == out
```

Polynomial IOP
+
PCS

Proof

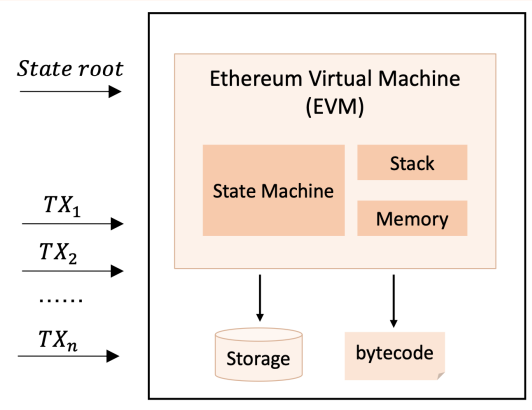
The workflow of zero-knowledge proof

Scroll

Program

Constraints

Proof



R1CS
Plonkish
AIR

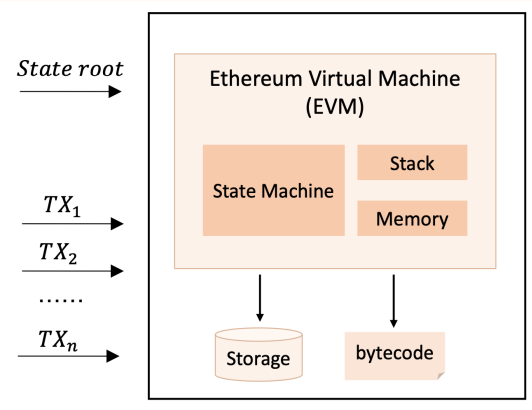
```
x * x == var1
var1 * x == y
(y+x) * 1 == var2
(var2+5) * 1 == out
```

Polynomial IOP
+
PCS

The workflow of zero-knowledge proof

Scroll

Program



Constraints

R1CS
Plonkish
AIR

```
x * x == var1
var1 * x == y
(y+x) * 1 == var2
(var2+5) * 1 == out
```

Proof

Plonk IOP
+
KZG

Plonkish Arithmetization

Scroll

a_0	a_1	a_2	a_3	a_4	T_0	T_1	T_2	T_3	T_4

witness **Table 1** **Table 2**

Plonkish Arithmetization

Scroll

a_0	a_1	a_2	a_3	a_4	T_0	T_1	T_2	T_3	T_4
$input_0$	$input_1$	$input_2$		$output$					
va_1	vb_1	vc_1		vd_1					
va_2	vb_2	vc_2		vd_2					
va_3	vb_3	vc_3		vd_3					
va_4	vb_4	vc_4	……	vd_4					
va_5	vb_5	vc_5		vd_5					
va_6	vb_6	vc_6		vd_6					
va_7	vb_7	vc_7		vd_7					
va_6	vb_6	vc_6		vd_6					
va_7	vb_7	vc_7		vd_7					

witness **Table 1** **Table 2**

Plonkish Arithmetization – Custom gate

Scroll

a_0	a_1	a_2	a_3	a_4	T_0	T_1	T_2	T_3	T_4
$input_0$	$input_1$	$input_2$		$output$					
va_1	vb_1	vc_1		vd_1					
va_2	vb_2	vc_2		vd_2					
va_3	vb_3	vc_3		vd_3					
va_4	vb_4	vc_4	……	vd_4					
va_5	vb_5	vc_5		vd_5					
va_6	vb_6	vc_6		vd_6					
va_7	vb_7	vc_7		vd_7					
va_6	vb_6	vc_6		vd_6					
va_7	vb_7	vc_7		vd_7					

witness | Table 1 | Table 2

$$va_3 * vb_3 * vc_3 - vb_4 = 0$$

Plonkish Arithmetization – Custom gate

Scroll

a_0	a_1	a_2	a_3	a_4	T_0	T_1	T_2	T_3	T_4
$input_0$	$input_1$	$input_2$		$output$					
va_1	vb_1	vc_1		vd_1					
va_2	vb_2	vc_2		vd_2					
va_3	vb_3	vc_3		vd_3					
va_4	vb_4	vc_4	……	vd_4					
va_5	vb_5	vc_5		vd_5					
va_6	vb_6	vc_6		vd_6					
va_7	vb_7	vc_7		vd_7					
va_6	vb_6	vc_6		vd_6					
va_7	vb_7	vc_7		vd_7					

witness **Table 1** **Table 2**

$$va_3 * vb_3 * vc_3 - vb_4 = 0$$

- High degree
- More customized

Plonkish Arithmetization – Custom gate

Scroll

a_0	a_1	a_2	a_3	a_4	T_0	T_1	T_2	T_3	T_4
$input_0$	$input_1$	$input_2$		$output$					
va_1	vb_1	vc_1		vd_1					
va_2	vb_2	vc_2		vd_2					
va_3	vb_3	vc_3		vd_3					
va_4	vb_4	vc_4	……	vd_4					
va_5	vb_5	vc_5		vd_5					
va_6	vb_6	vc_6		vd_6					
va_7	vb_7	vc_7		vd_7					
va_6	vb_6	vc_6		vd_6					
va_7	vb_7	vc_7		vd_7					

witness **Table 1** **Table 2**

$$vb_1 * vc_1 + vc_2 - vc_3 = 0$$

- High degree
- More customized

Plonkish Arithmetization – Custom gate

Scroll

a_0	a_1	a_2	a_3	a_4	T_0	T_1	T_2	T_3	T_4
$input_0$	$input_1$	$input_2$		$output$					
va_1	vb_1	vc_1		vd_1					
va_2	vb_2	vc_2		vd_2					
va_3	vb_3	vc_3		vd_3					
va_4	vb_4	vc_4	……	vd_4					
va_5	vb_5	vc_5		vd_5					
va_6	vb_6	vc_6		vd_6					
va_7	vb_7	vc_7		vd_7					
va_6	vb_6	vc_6		vd_6					
va_7	vb_7	vc_7		vd_7					

witness **Table 1** **Table 2**

$$vc_1 + va_2 * vb_4 - vc_4 = 0$$

- High degree
- More customized

Plonkish Arithmetization – Permutation

Scroll

a_0	a_1	a_2	a_3	a_4	T_0	T_1	T_2	T_3	T_4
$input_0$	$input_1$	$input_2$		$output$					
va_1	vb_1	vc_1		vd_1					
va_2	vb_2	vc_2		vd_2					
va_3	vb_3	vc_3		vd_3					
va_4	$\boxed{vb_4}$	vc_4	……	vd_4					
va_5	vb_5	vc_5		vd_5					
va_6	vb_6	$\boxed{vc_6}$		vd_6					
va_7	vb_7	vc_7		vd_7					
$\boxed{va_6}$	$\boxed{vb_6}$	vc_6		vd_6					
va_7	vb_7	vc_7		vd_7					

witness | Table 1 | Table 2

$$vb_4 = vc_6 = vb_6 = va_6$$

Plonkish Arithmetization – Lookup argument

Scroll

a_0	a_1	a_2	a_3	a_4	T_0	T_1	T_2	T_3	T_4
$input_0$	$input_1$	$input_2$		$output$					
va_1	vb_1	vc_1		vd_1					
va_2	vb_2	vc_2		vd_2					
va_3	vb_3	vc_3		vd_3					
va_4	vb_4	vc_4	……	vd_4					
va_5	vb_5	vc_5		vd_5					
va_6	vb_6	vc_6		vd_6					
va_7	vb_7	vc_7							
va_6	vb_6	vc_6		vd_6					
va_7	vb_7	vc_7		vd_7					

Lookup

witness | Table 1 | Table 2

$$(va_7, vb_7, vc_7) \in (T_0, T_1, T_2)$$

Plonkish Arithmetization – Lookup argument

Scroll

a_0	a_1	a_2	a_3	a_4	T_0	T_1	T_2	T_3	T_4
$input_0$	$input_1$	$input_2$		$output$	0000				
va_1	vb_1	vc_1		vd_1	0001				
va_2	vb_2	vc_2		vd_2	0010				
va_3	vb_3	vc_3		vd_3	0011				
va_4	vb_4	vc_4	……	vd_4	0100				
va_5	vb_5	vc_5		vd_5	0101				
va_6	vb_6	vc_6		vd_6	……				
va_7	vb_7	$\boxed{vc_7}$	Lookup		1101				
va_6	vb_6	vc_6		vd_6	1110				
va_7	vb_7	vc_7		vd_7	1111				

witness **Table 1** **Table 2**

$$vc_7 \in [0, 15]$$

Plonkish Arithmetization – Lookup argument

Scroll

a_0	a_1	a_2	a_3	a_4	T_0	T_1	T_2	T_3	T_4
$input_0$	$input_1$	$input_2$		$output$	0000	0000	0000		
va_1	vb_1	vc_1		vd_1	0000	0001	0001		
va_2	vb_2	vc_2		vd_2	0000	0010	0010		
va_3	vb_3	vc_3		vd_3	0000	0011	0011		
va_4	vb_4	vc_4	……	vd_4	0000	0100	0100		
va_5	vb_5	vc_5		vd_5	0000	0101	0101		
va_6	vb_6	vc_6		vd_6	……	……	……		
va_7	vb_7	vc_7			1111	1101	0010		
va_6	vb_6	vc_6		vd_6	1111	1110	0001		
va_7	vb_7	vc_7		vd_7	1111	1111	0000		

Lookup

witness | Table 1 | Table 2

$$vc_7 \in [0, 15]$$

$$va_7 \oplus vb_7 = vc_7$$

Plonkish Arithmetization – Lookup argument

Scroll

a_0	a_1	a_2	a_3	a_4	T_0	T_1	T_2	T_3	T_4
$input_0$	$input_1$	$input_2$		$output$	0000	0000	0000		
va_1	vb_1	vc_1		vd_1	0000	0001	0001		
va_2	vb_2	vc_2		vd_2	0000	0010	0010		
va_3	vb_3	vc_3		vd_3	0000	0011	0011		
va_4	vb_4	vc_4	……	vd_4	0000	0100	0100		
va_5	vb_5	vc_5		vd_5	0000	0101	0101		
va_6	vb_6	vc_6		vd_6	……	……	……		
va_7	vb_7	vc_7			1111	1101	0010		
va_6	vb_6	vc_6		vd_6	1111	1110	0001		
va_7	vb_7	vc_7		vd_7	1111	1111	0000		

Lookup

witness | Table 1 | Table 2

$$vc_7 \in [0, 15]$$

$$va_7 \oplus vb_7 = vc_7$$

RAM operation

Plonkish Arithmetization – Constraints

Scroll

a_0	a_1	a_2	a_3	a_4	T_0	T_1	T_2	T_3	T_4
$input_0$	$input_1$	$input_2$		$output$					
va_1	vb_1	vc_1		vd_1					
va_2	vb_2	vc_2		vd_2					
va_3	vb_3	vc_3		vd_3					
va_4	vb_4	vc_4	……	vd_4					
va_5	vb_5	vc_5		vd_5					
va_6	vb_6	vc_6		vd_6					
va_7	vb_7	vc_7		vd_7					
va_6	vb_6	vc_6		vd_6					
va_7	vb_7	vc_7		vd_7					

witness **Table 1** **Table 2**

$$vb_1 * vc_1 + vc_2 - vc_3 = 0$$
$$va_3 * vb_3 * vc_3 - vb_4 = 0$$
$$vb_4 + vc_6 * vb_6 - va_6 = 0$$

……

$$vb_4 = vc_6 = vb_6 = va_6$$

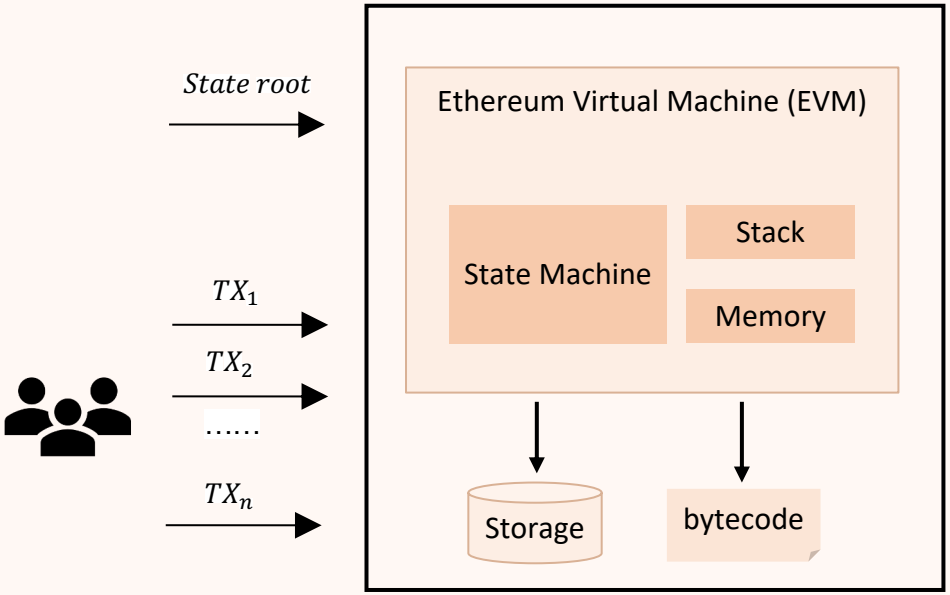
……

$$(va_7, vb_7, vc_7) \in (T_0, T_1, T_2)$$

How should we choose “front-end”?

Scroll

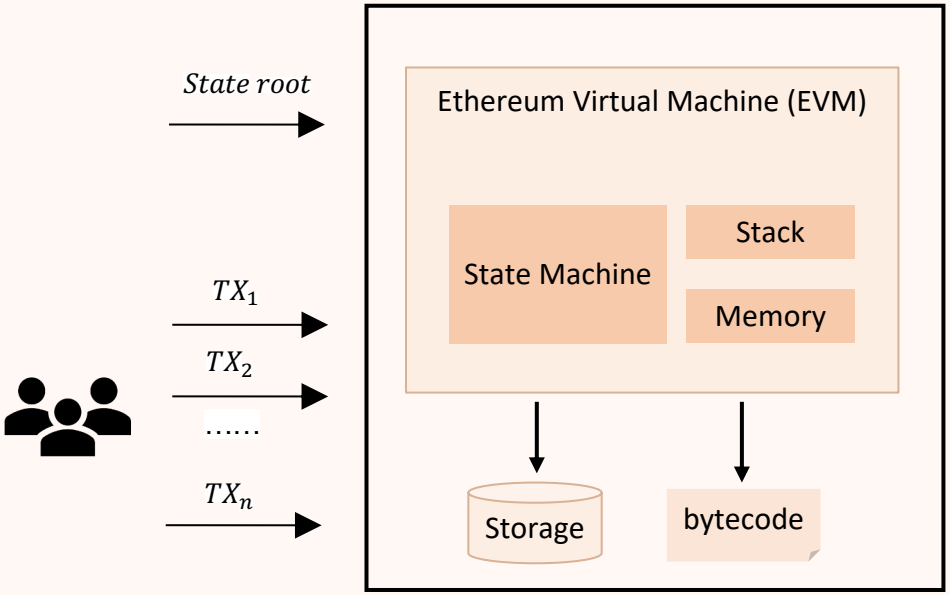
Computation



Why Plonkish arithmetization?

Scroll

Computation

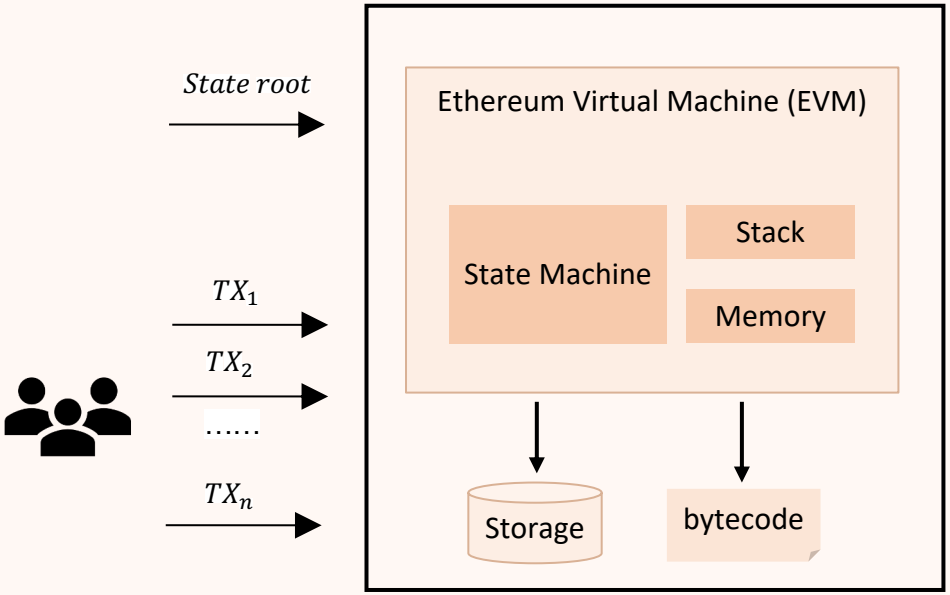


- EVM word size is 256bit
  - Efficient range proof
- EVM has zk-unfriendly opcodes
  - Efficient way to connect circuits
- Read & Write consistency
  - Efficient mapping
- EVM has a dynamic execution trace
  - Efficient on/off selectors

Why Plonkish arithmetization?

Scroll

Computation

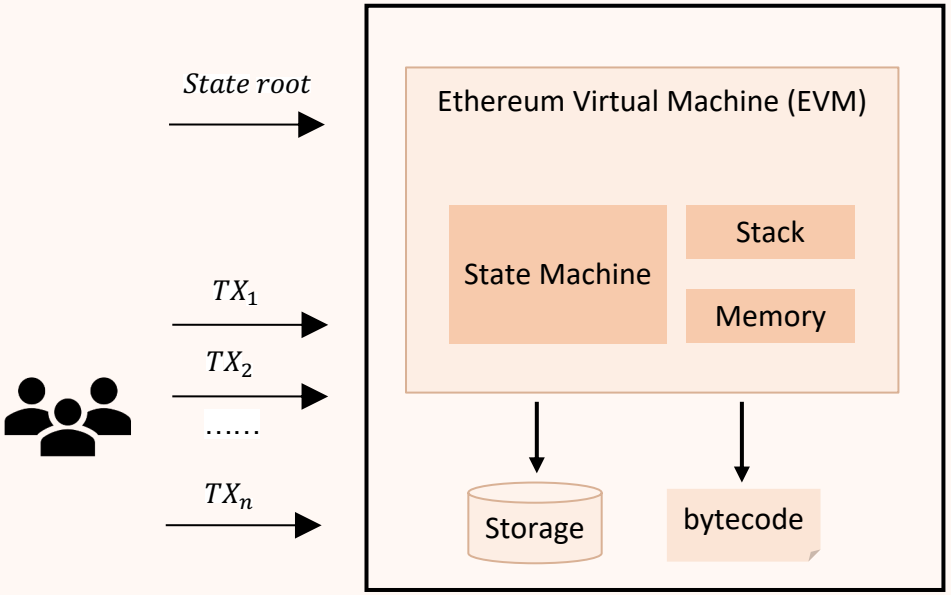


- EVM word size is 256bit
  - Efficient range proof
- EVM has zk-unfriendly opcodes
  - Efficient way to connect circuits
- Read & Write consistency
  - Efficient mapping
- EVM has a dynamic execution trace
  - Efficient on/off selectors

Why Plonkish arithmetization?

Scroll

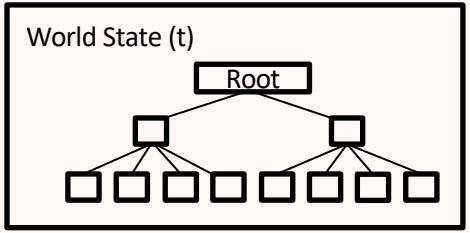
Computation



- EVM word size is 256bit
  - Efficient range proof
- EVM has zk-unfriendly opcodes
  - Efficient way to connect circuits
- Read & Write consistency
  - Efficient mapping
- EVM has a dynamic execution trace
  - Efficient on/off selectors

What you need to prove

Scroll

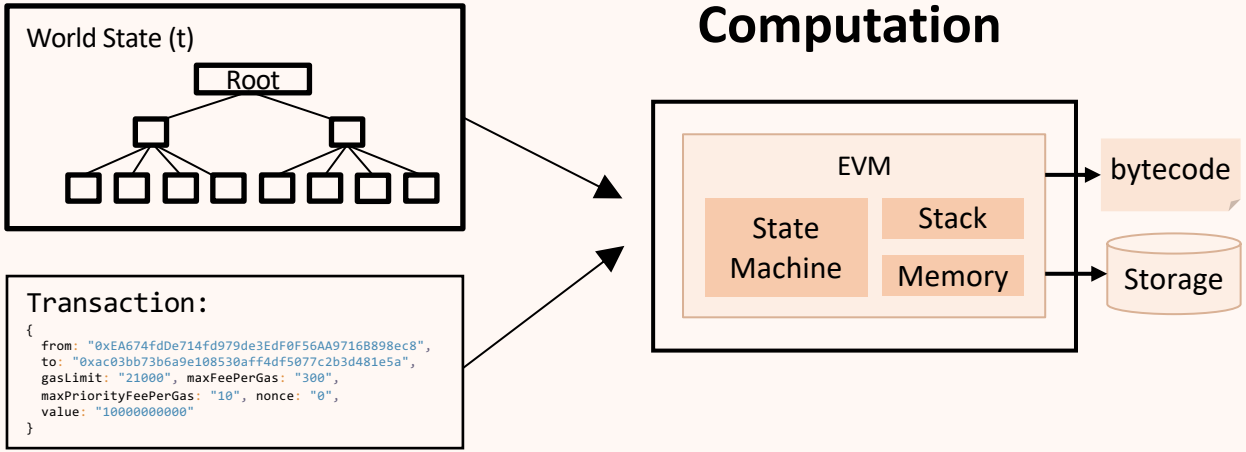


Transaction:

```
{
  from: "0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8",
  to: "0xac03bb73b6a9e108530aff4df5077c2b3d481e5a",
  gasLimit: "21000", maxFeePerGas: "300",
  maxPriorityFeePerGas: "10", nonce: "0",
  value: "10000000000"
}
```

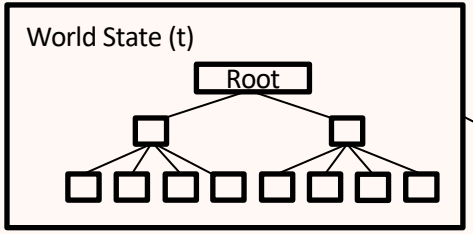
What you need to prove

Scroll



What you need to prove

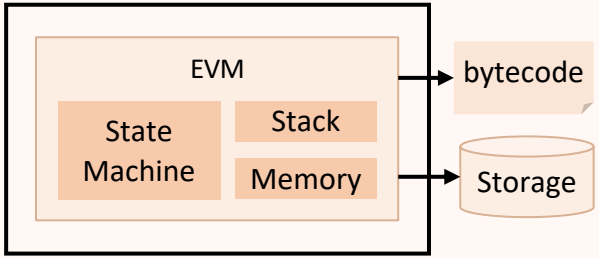
Scroll



Transaction:

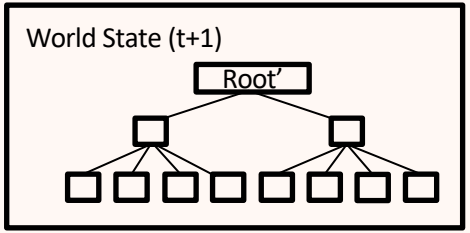
```
{
  from: "0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8",
  to: "0xac03bb73b6a9e108530aff4df5077c2b3d481e5a",
  gasLimit: "21000", maxFeePerGas: "300",
  maxPriorityFeePerGas: "10", nonce: "0",
  value: "10000000000"
}
```

Computation



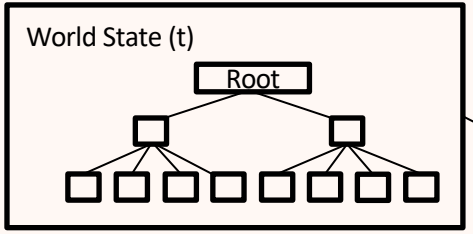
Execution trace

Step	Opcode
0	PUSH1 80
1	PUSH1 40
2	MSTORE
3	CALLVALUE
...	...
n	RETURN



What you need to prove

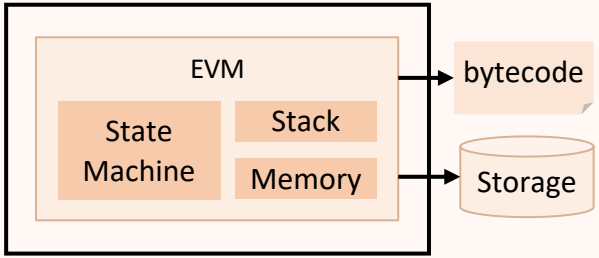
Scroll



Transaction:

```
{
  from: "0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8",
  to: "0xac03bb73b6a9e108530aff4df5077c2b3d481e5a",
  gasLimit: "21000", maxFeePerGas: "300",
  maxPriorityFeePerGas: "10", nonce: "0",
  value: "10000000000"
}
```

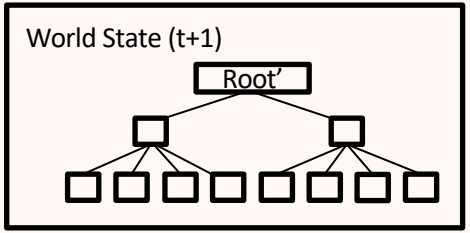
Computation



Execution trace

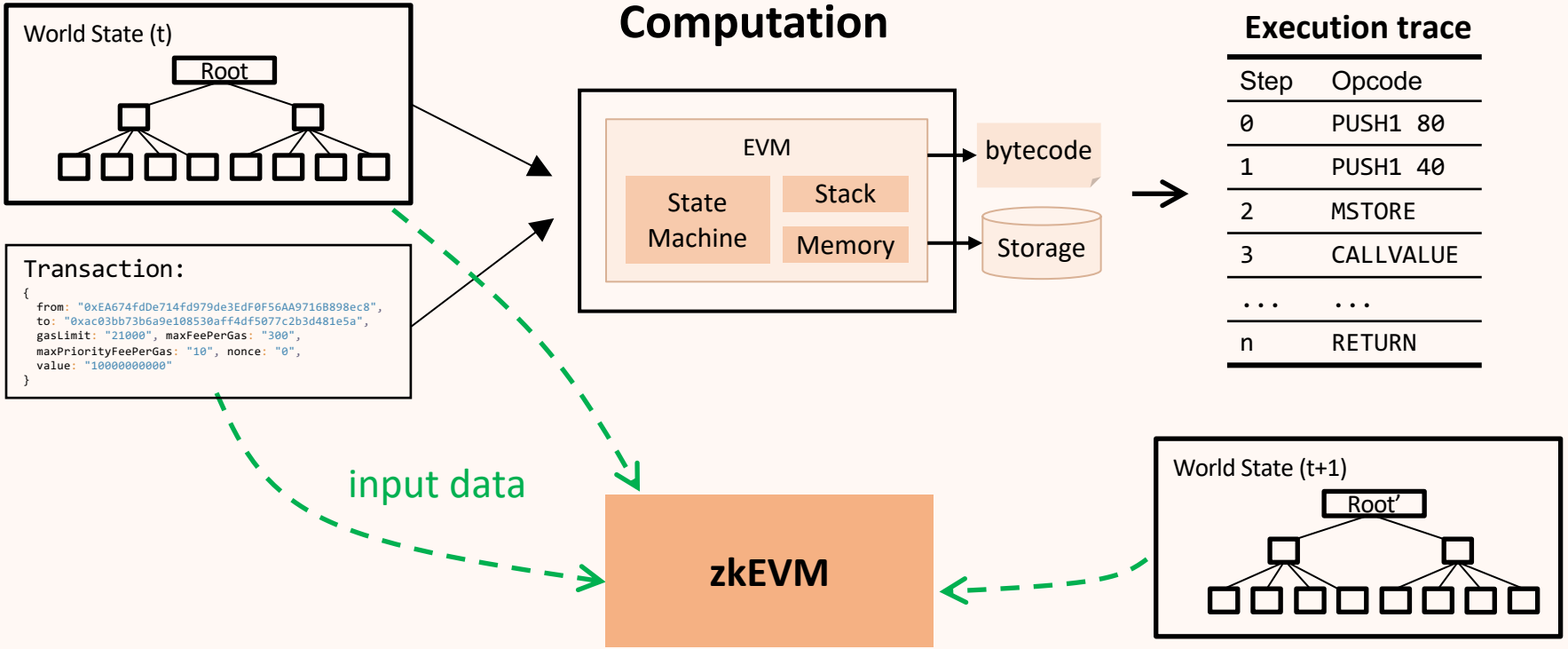
Step	Opcode
0	PUSH1 80
1	PUSH1 40
2	MSTORE
3	CALLVALUE
...	...
n	RETURN

zkEVM



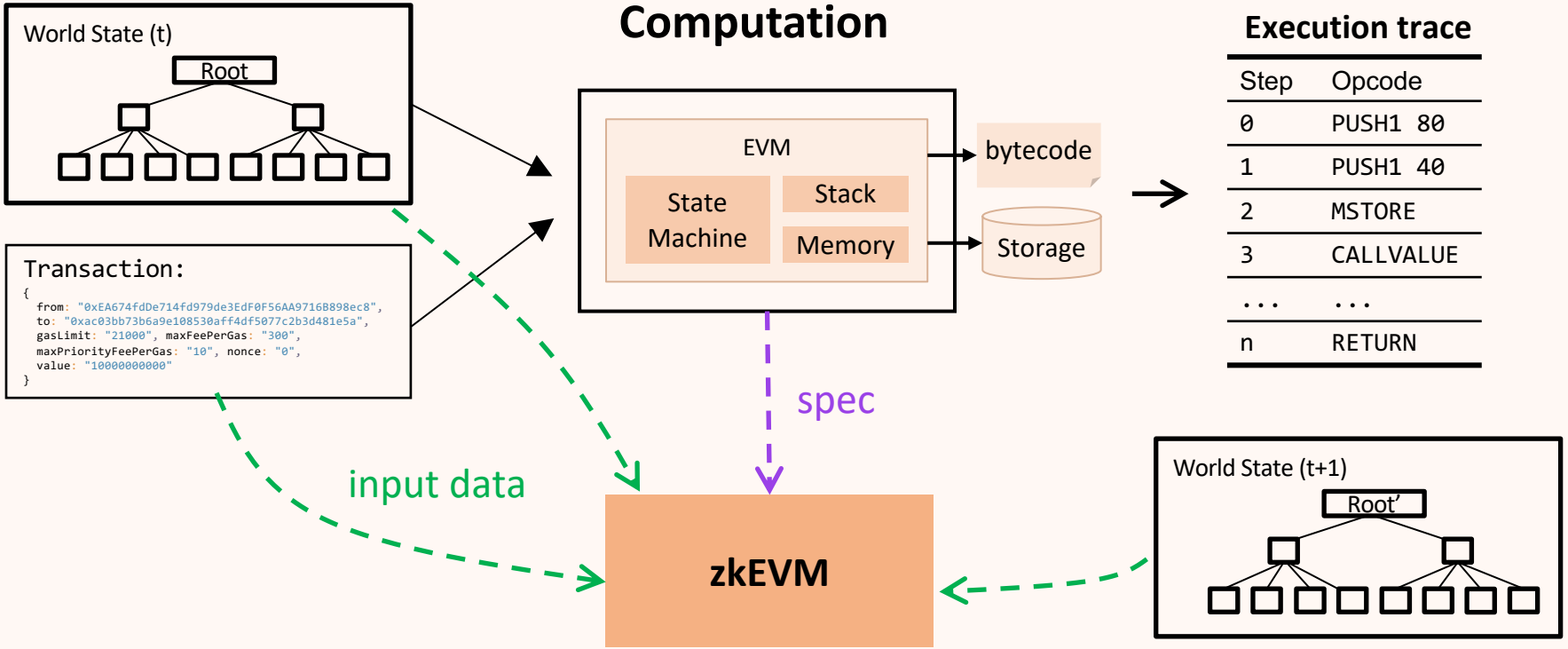
What you need to prove

Scroll



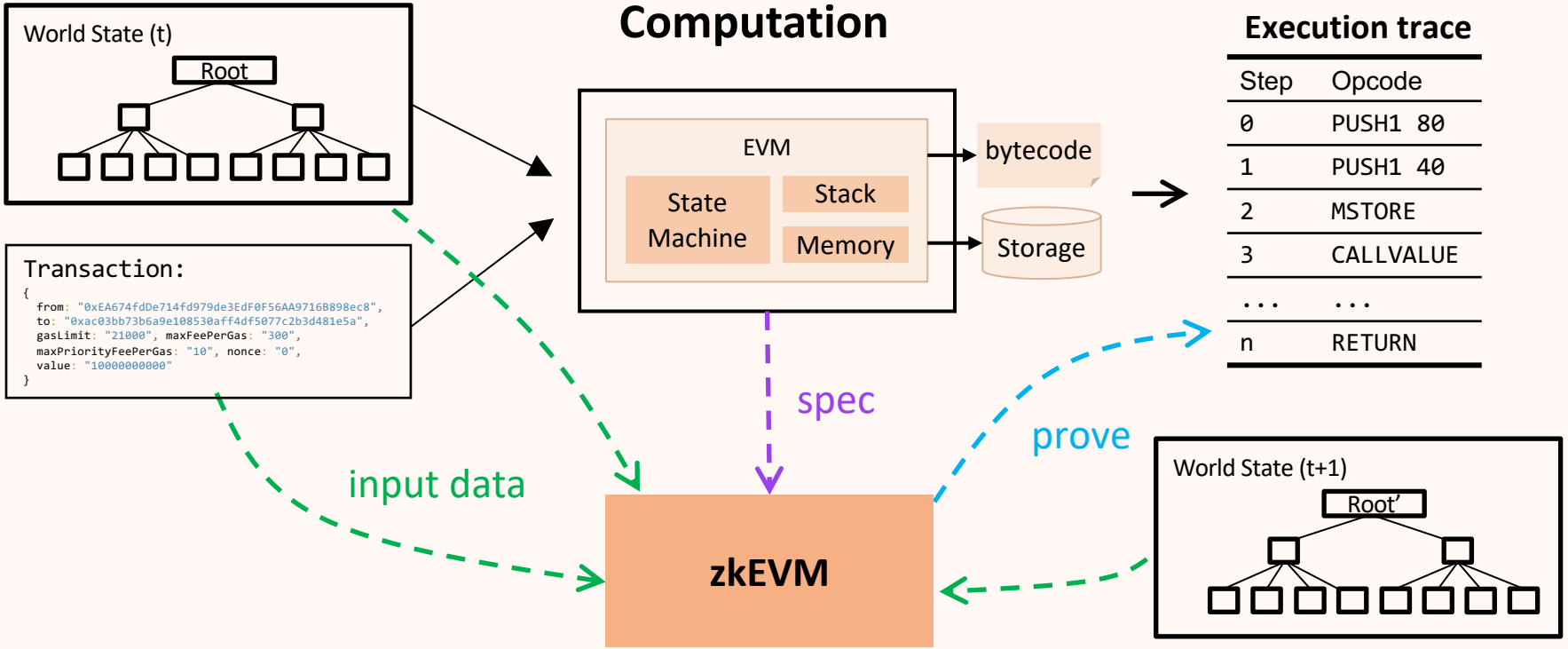
What you need to prove

Scroll



What you need to prove

Scroll



EVM circuit

Scroll

Execution trace

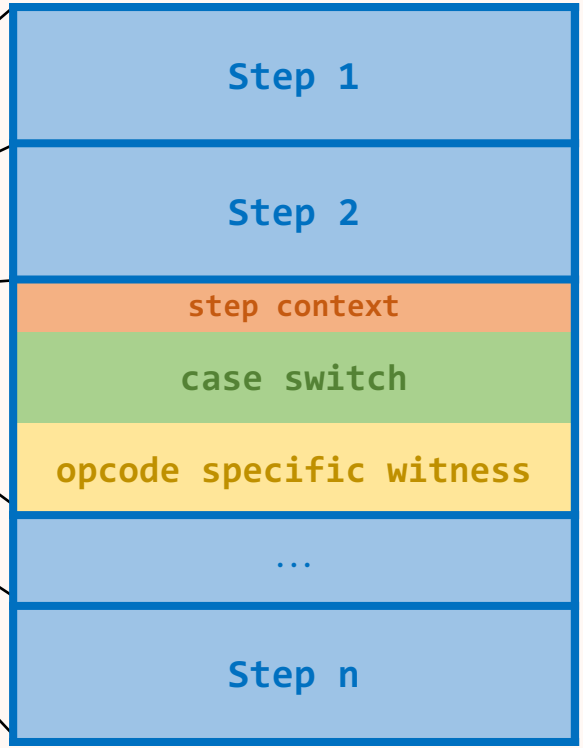
step	opcode
1	PUSH1 80
2	PUSH1 40
3	ADD
...	...
...	...
n	RETURN

EVM circuit

Scroll

Execution trace

step	opcode
1	PUSH1 80
2	PUSH1 40
3	ADD
...	...
...	...
n	RETURN

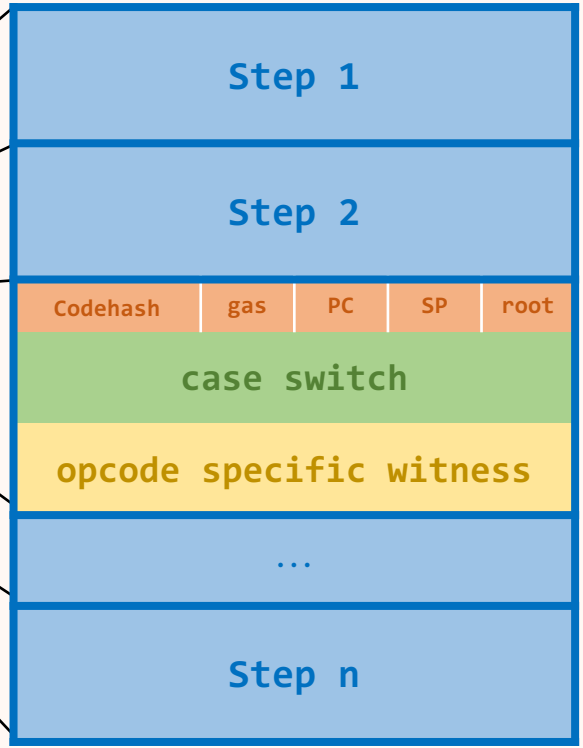


EVM circuit

Scroll

Execution trace

step	opcode
1	PUSH1 80
2	PUSH1 40
3	ADD
...	...
...	...
n	RETURN



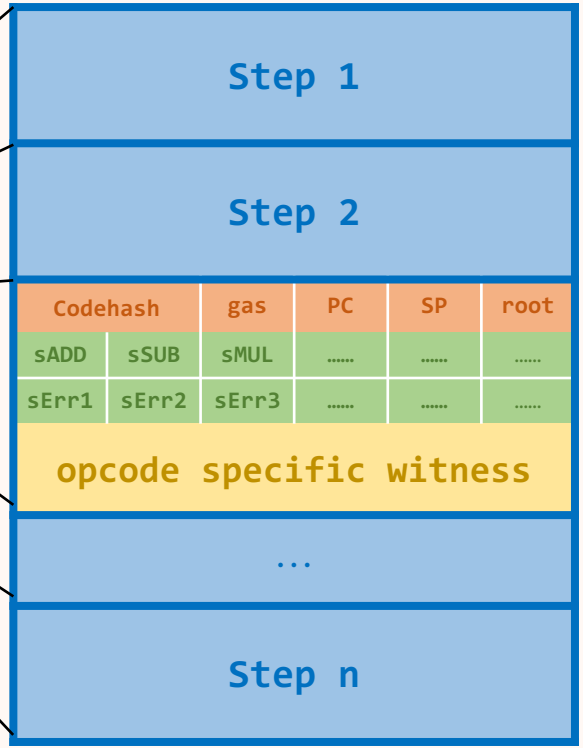
- **Step context**
  - Codehash
  - Gas left
  - Program counter, Stack pointer

EVM circuit

Scroll

Execution trace

step	opcode
1	PUSH1 80
2	PUSH1 40
3	ADD
...	...
...	...
n	RETURN



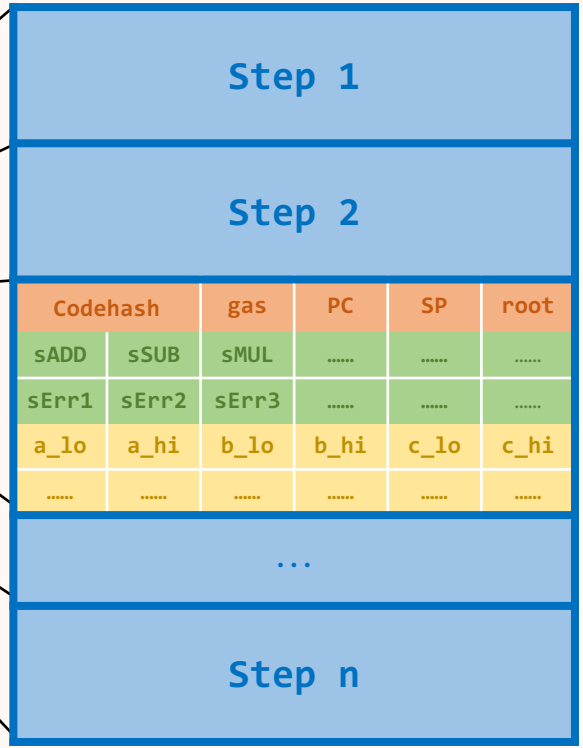
- **Step context**
  - Codehash
  - Gas left
  - Program counter, Stack pointer
- **Case switch**
  - Select opcodes & error cases
  - Exactly one is switched on

EVM circuit

Scroll

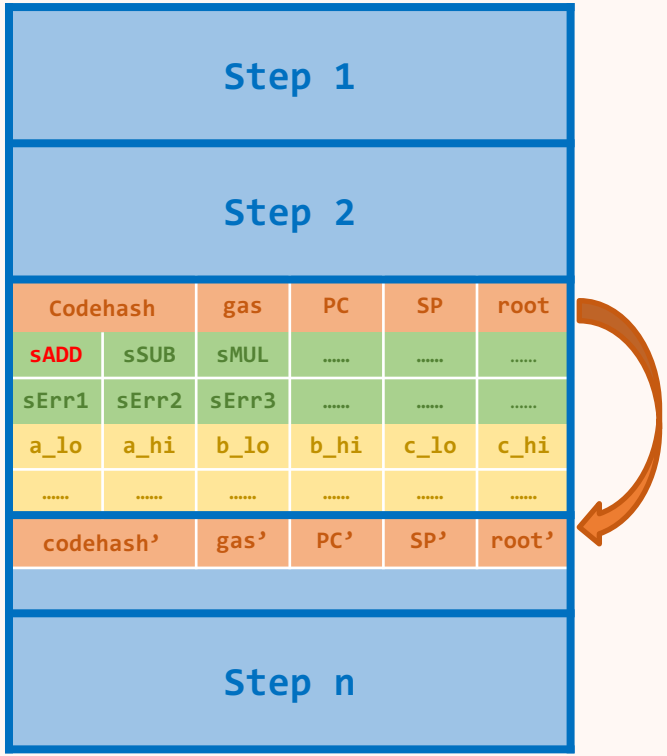
Execution trace

step	opcode
1	PUSH1 80
2	PUSH1 40
3	ADD
...	...
...	...
n	RETURN



- **Step context**
  - Codehash
  - Gas left
  - Program counter, Stack pointer
- **Case switch**
  - Select opcodes & error cases
  - Exactly one is switched on
- **Opcode specific witness**
  - Extra witness used for opcodes
  - i.e. operands, carry, limbs, …

EVM circuit - ADD



- **Step context**

```
sADD * (pc' – pc – 1) == 0
sADD * (sp' – sp – 1) == 0
sADD * (gas' – gas – 3) == 0
```

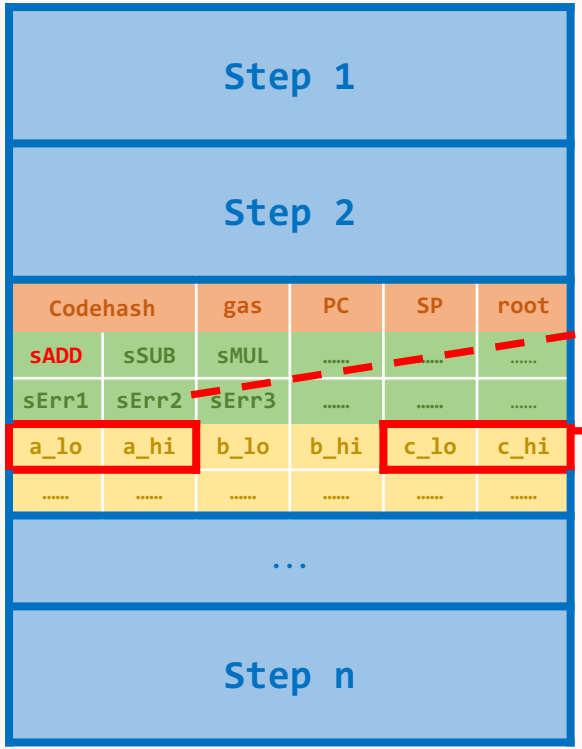
- **Case switch**

```
sADD * (1 – sADD) == 0
sMUL * (1 – sMUL) == 0
        ...
sADD + sMUL + ... + sERRk == 1
```

- **Opcode specific witness**

```
sADD*(a_lo+b_lo-c_lo – carry0* 2^128)== 0
sADD*(a_hi+b_hi+carry0-c_hi – carry1*2^128)== 0
```

EVM circuit - ADD

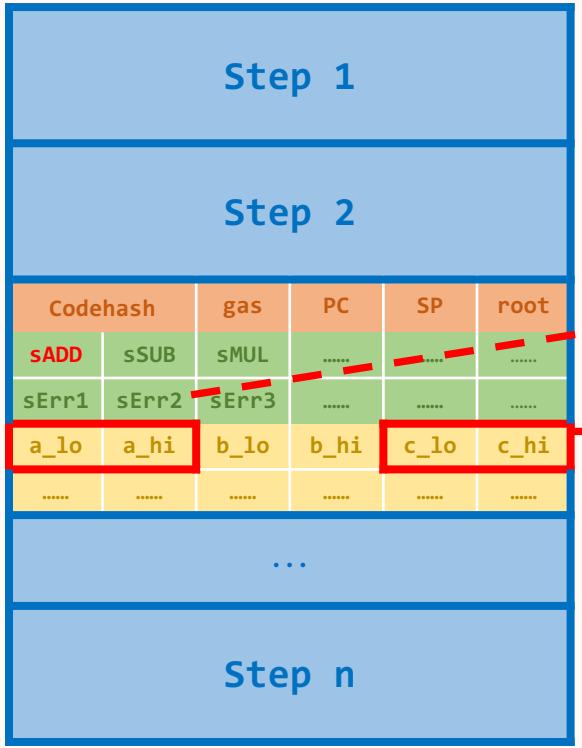


- **Opcode specific witness**

idx	tag	addr	R/W	value
1	STACK	1023	1	...
5	STACK	1022	0	word_a
6	STACK	1023	0	word_b
7	STACK	1023	1	word_c
...	STACK	...	...	...
...	MEMORY	0x40	1	...
...	MEMORY	...	...	...
...	STORAGE	...	...	...

EVM circuit - ADD

Scroll



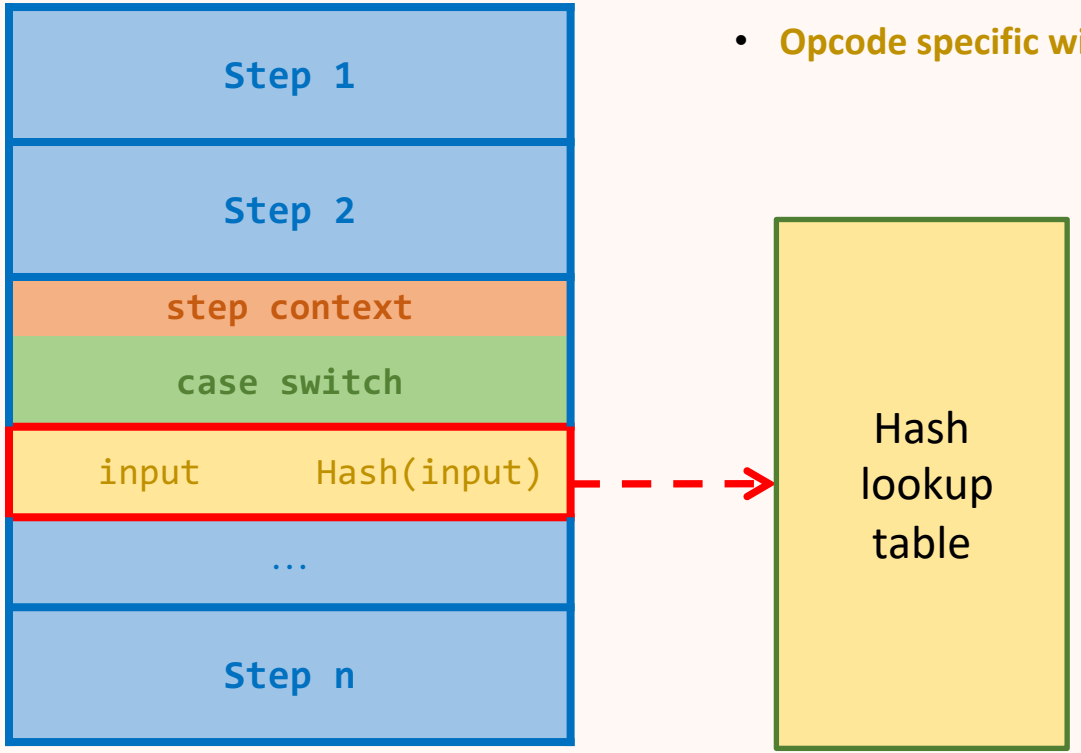
- **Opcode specific witness**

idx	tag	addr	R/W	value
1	STACK	1023	1	...
5	STACK	1022	0	word_a
6	STACK	1023	0	word_b
7	STACK	1023	1	word_c
...	STACK	...	...	...
...	MEMORY	0x40	1	...
...	MEMORY	...	...	...
...	STORAGE	...	...	...

RAM circuit

EVM circuit - Hash

Scroll

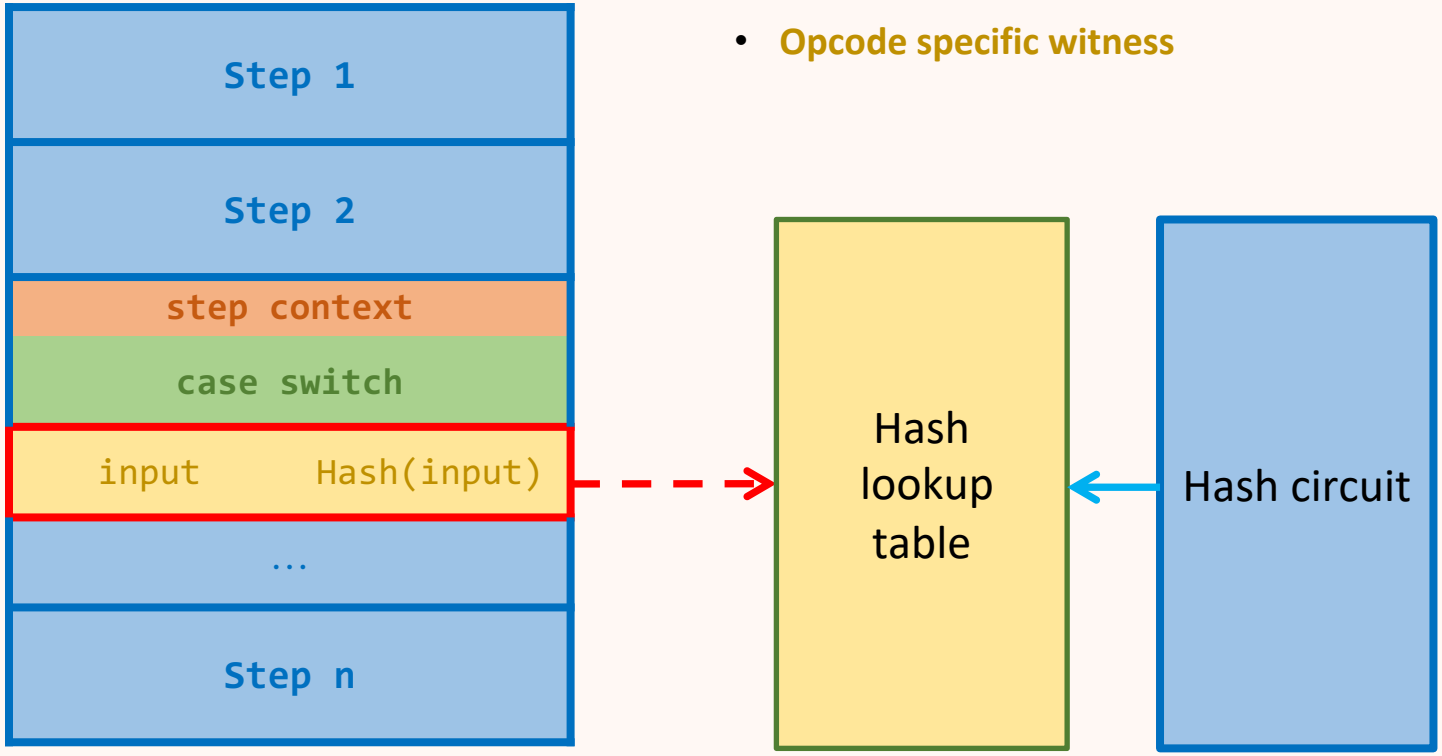


- Opcode specific witness

EVM circuit - Hash

Scroll

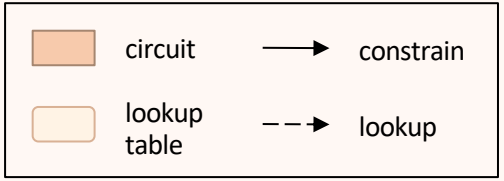
- Opcode specific witness



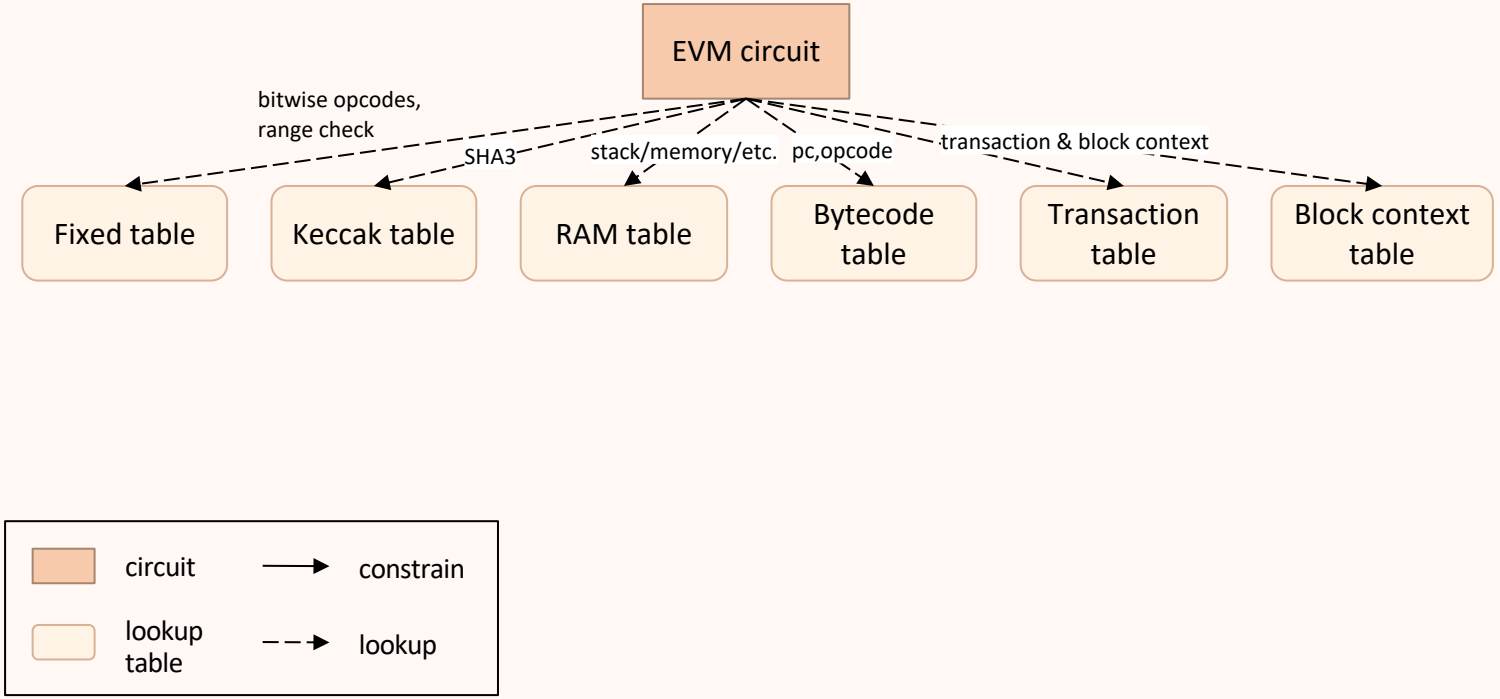
The architecture of zkEVM circuits

Scroll

EVM circuit

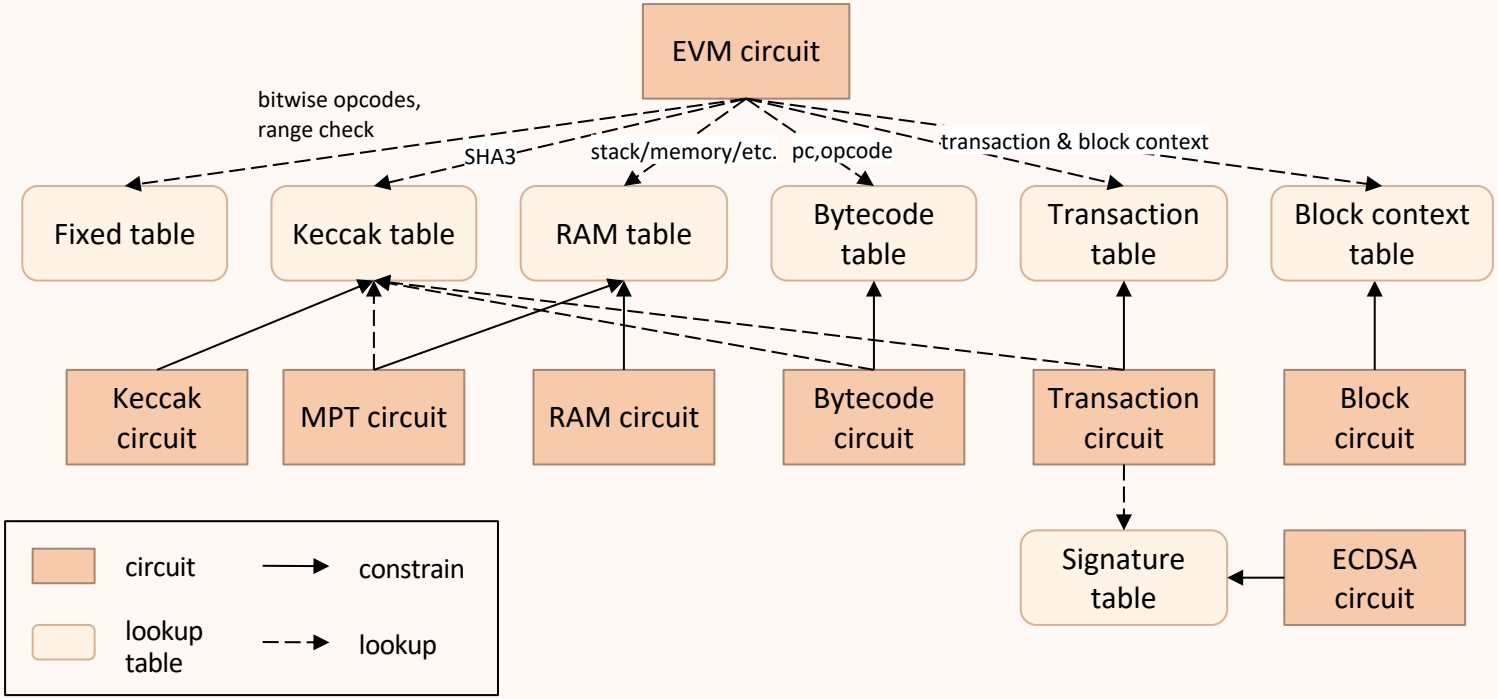


The architecture of zkEVM circuits



The architecture of zkEVM circuits

Scroll



Different zkEVM comparsion

Scroll

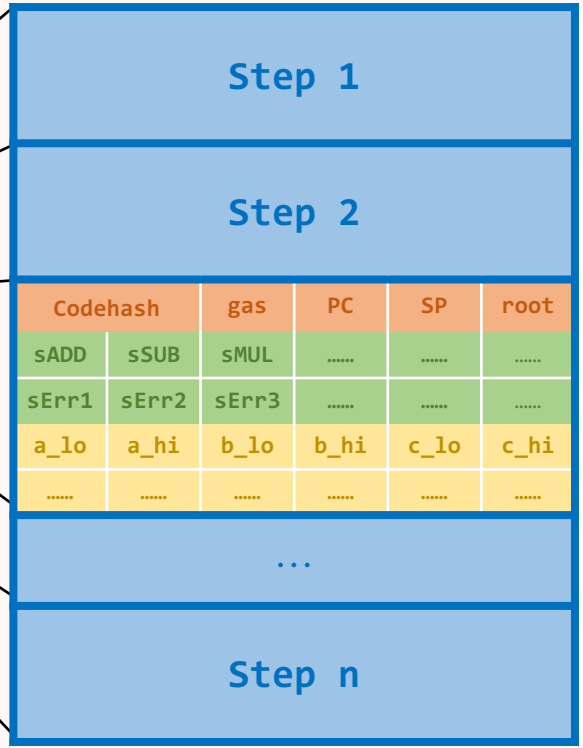
Will walk through without slides :(

EVM circuit

Scroll

Execution trace

step	opcode
1	PUSH1 80
2	PUSH1 40
3	ADD
...	...
...	...
n	RETURN



- **Step context**
  - Codehash
  - Gas left
  - Program counter, Stack pointer
- **Case switch**
  - Select opcodes & error cases
  - Exactly one is switched on
- **Opcode specific witness**
  - Extra witness used for opcodes
  - i.e. operands, carry, limbs, …

Outline

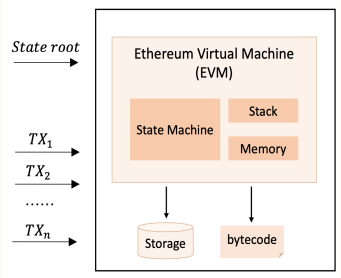
Scroll

- Background & motivation
- zkEVM circuit arithmetization
- zkEVM prover optimization
- Something interesting to share

The workflow of zero-knowledge proof

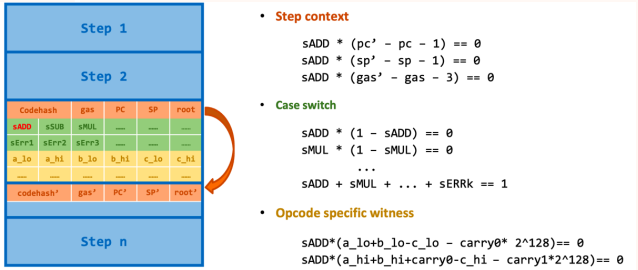
Scroll

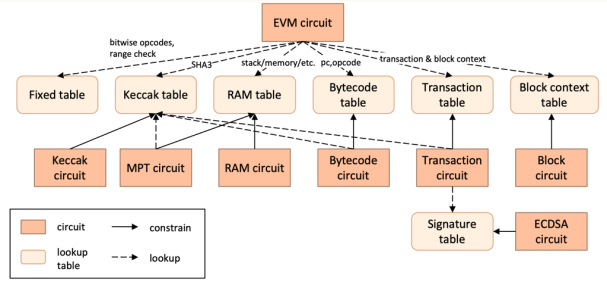
Program



R1CS
Plonkish
AIR

Constraints



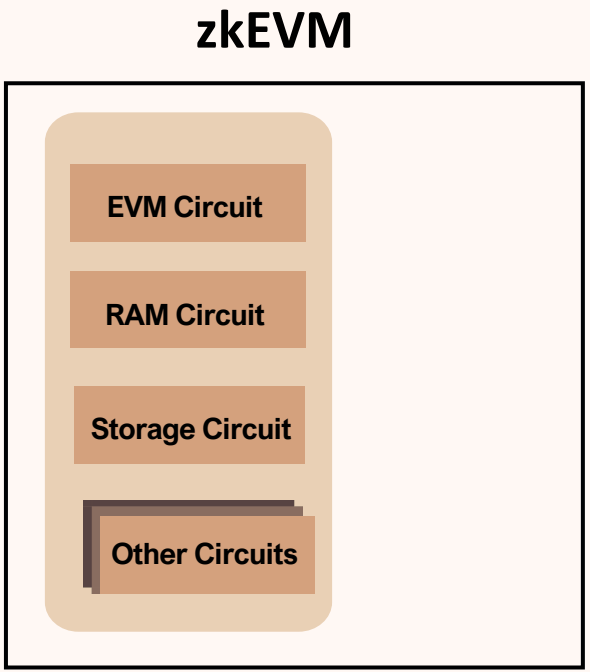


Proof

Plonk IOP
+
KZG

The proof system for zkEVM

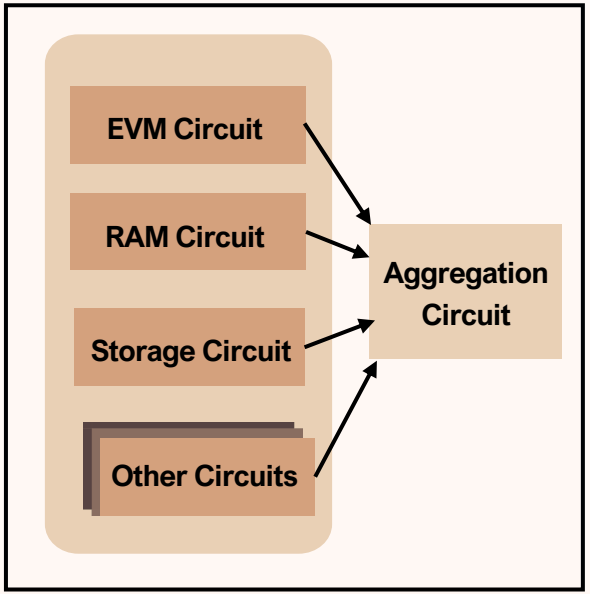
Scroll



The proof system for zkEVM

Scroll

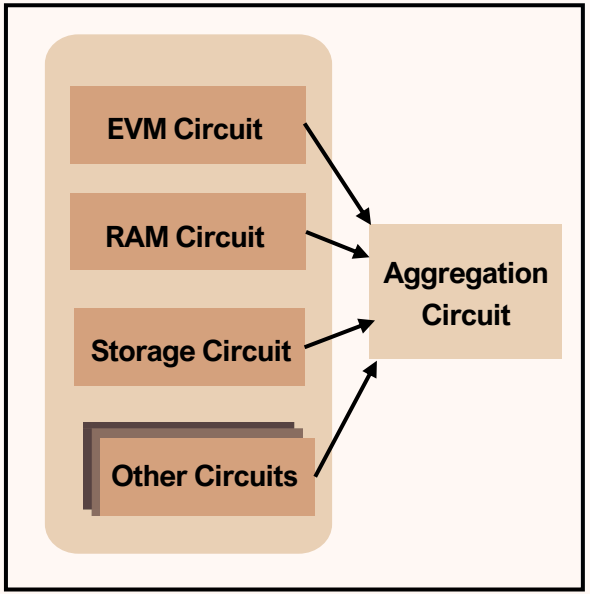
zkEVM



Two-layer architecture

Scroll

zkEVM

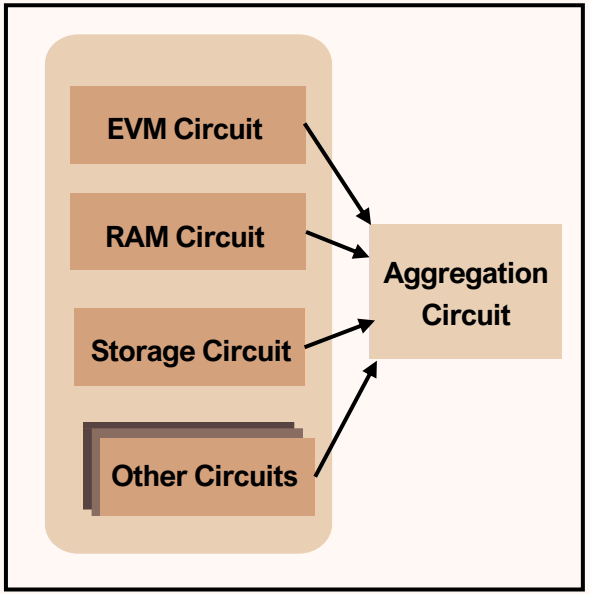


- **The first layer needs to handle large computation**
  - Custom gate, Lookup support (“expressive”, customized)
  - Hardware friendly prover (parallelizable, low peak memory)
  - The verification circuit is small
  - Transparent or Universal trusted setup
- Some promising candidates
  - Plonky2/Starky /eSTARK
  - Halo2/Halo2-KZG
  - New IOP without FFTs (i.e. HyperPlonk, Plonk without FFT)
  - If Spartan/Virgo/… (sumcheck based) or Nova can support Plonkish

Two-layer architecture

Scroll

zkEVM

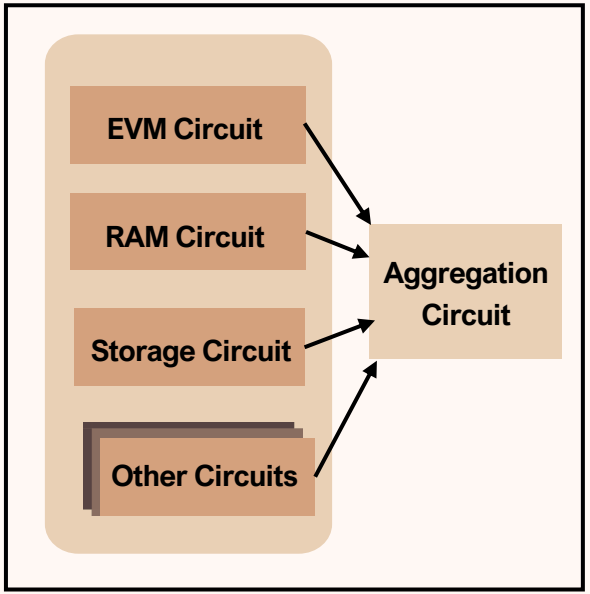


- **The second layer needs to be verifier efficient (in EVM)**
  - Proof is efficiently verifiable on EVM (small proof, low gas cost)
  - Prove the verification circuit of the former layer efficiently
  - Ideally, hardware friendly prover
  - Ideally, transparent or universal trusted setup
- Some promising candidates
  - Groth16
  - Plonk with very few columns
    - KZG/Fflonk/Keccak FRI (larger code rate)

Two-layer architecture

Scroll

zkEVM

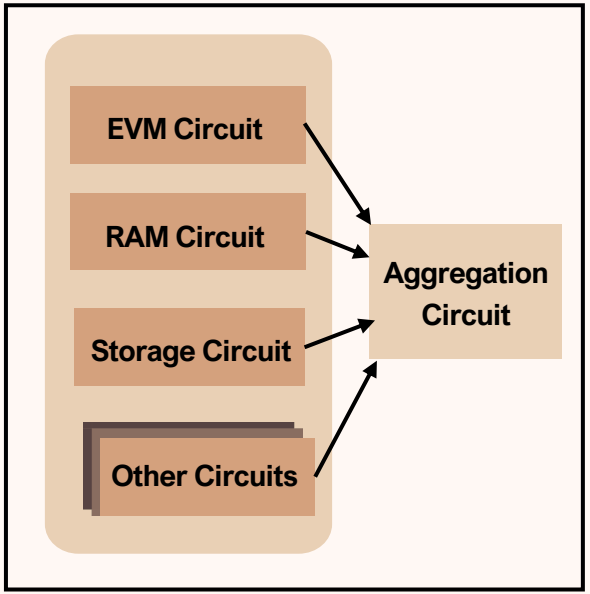


- **The first layer is Halo2-KZG** (Poseidon hash transcript)
  - Custom gate, Lookup support
  - Good enough prover performance (GPU prover)
  - The verification circuit is “small”
  - Universal trusted setup
- **The second layer is Halo2-KZG** (Keccak hash transcript)
  - Custom gate, Lookup support (express non-native efficiently)
  - Good enough prover performance (GPU prover)
  - The final verification cost can be configured to be really small

The performance

Scroll

zkEVM

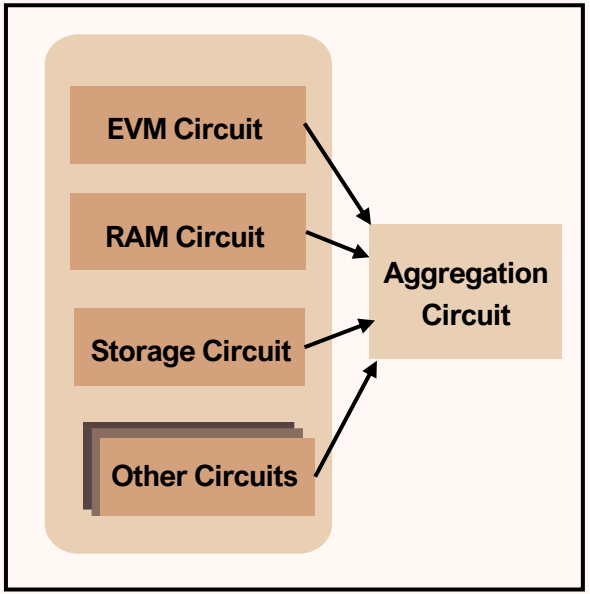


- The first layer needs to be “expressive”
  - EVM circuit has **143 columns, 5700 custom gates, 58 lookups**
  - Highest custom gate degree: 9
  - For 1M gas, EVM circuit needs **2^18 rows** (more gas, more rows)
- The second layer needs to aggregate proofs into one proof
  - Aggregation circuit has **23 columns, 1 custom gate, 7 lookups**
  - Highest custom gate degree: 5
  - For aggregating EVM, RAM, Storage circuits, it needs **2^25 rows**

The performance

Scroll

zkEVM



- Our GPU prover optimization
  - MSM, NTT and quotient kernel
  - Pipeline and overlap CPU and GPU computation
  - Multi-card implementation, memory optimization
- The Performance
  - For EVM circuit
    - CPU prover takes 270.5s, GPU prover takes **30s (9x speedup!)**
  - For Aggregation circuit
    - CPU prover takes 2265s, GPU prover takes **149s (15x speedup!)**
  - For 5M gas, first layer takes 2 minutes, second layer takes 3 minutes

Outline

Scroll

- Background & motivation
- zkEVM circuit arithmetization
- zkEVM prover optimization
- Something interesting to share

An overview of modularized proof system

Scroll

Frontend

R1CS / Plonkish/ AIR / CCS

Backend

Polynomial IOP + PCS

An overview of modularized proof system

Scroll

Frontend

R1CS / Plonkish/ AIR / CCS

Backend

Polynomial IOP + PCS

- **Different front-end is suitable for different applications**
  - R1CS if good for linear combination and general
  - Plonkish/AIR is more uniform and customized
  - CCS has smaller witness, but sparse matrix opening is expensive
- **The PCS influences concrete properties a lot**
  - Trusted setup, Security assumption
  - Prover efficiency, Proof size, Verifier efficiency

An overview of modularized proof system

Scroll

Frontend

R1CS / Plonkish/ AIR / CCS

Backend

Polynomial IOP + PCS

- **KZG**
  - Universal trusted setup, DLOG
  - Prover is doing MSM, Verifier is doing pairing, small proof
- **FRI-based**
  - No trusted setup, hash collision
  - Prover is doing hashes & FFTs, Verifier is doing hashes, large proof
- **IPA**
  - No trusted setup, DLOG
  - Prover is doing MSM, Verifier is doing MSM, medium proof
- **Multilinear PCS** (for sum-check based constructions)

An overview of modularized proof system

Scroll

Frontend

R1CS / Plonkish/ AIR / CCS

Backend

Polynomial IOP + PCS

- **Commonly used**
  - Halo2: Plonkish, Plonk IOP, IPA/KZG
  - Plonky2: Plonkish, Plonk IOP, FRI
  - STARK: AIR, STARK IOP, FRI
  - (Groth16 is based on linear PCP, not fall into this category)
- **New protocols based on multilinear PCS**
  - Spartan: R1CS, Spartan IOP, IPA derived
  - HyperPlonk: Plonkish, HyperPlonk IOP, KZG/FRI derived
  - Nova, SuperNova, HyperNova, Protostar

Some other considerations

Scroll

- **Ecosystem**
  - Compatibility with existing libraries
  - Existing projects and gadgets
- **Implementation**
  - Industrial vs academic implementation
  - Consider the best practice (GPUs)
  - License
  - Audit
- **Standardize**
  - Community
  - ZK ASICs

Some research we have been working on

Scroll

- Next generation zkEVM (faster, higher gas limit, etc)
- On-chain data compression, 4844 support
- Decentralized prover
- Decentralized sequencer
- Interoperability between zkRollups

Finally, …

Scroll

- **We are building cool things at Scroll!**
  - Scroll is a general purpose zkRollup scaling solution for Ethereum
  - Building a native zkEVM using very advanced circuit arithmetization + proof system
  - Building fast prover through hardware acceleration (GPU in production) + proof recursion
  - We are live on the mainnet with a production-level robust infrastructure

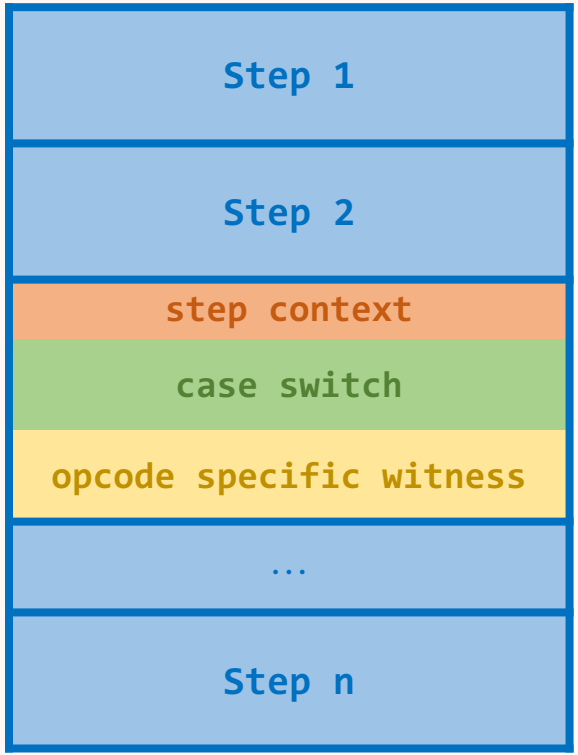
- **There are a bunch of interesting problems to be solved and we are hiring!**
  - Protocol design and mechanism design
  - Zk engineer & research for practical efficiency

Thank you!

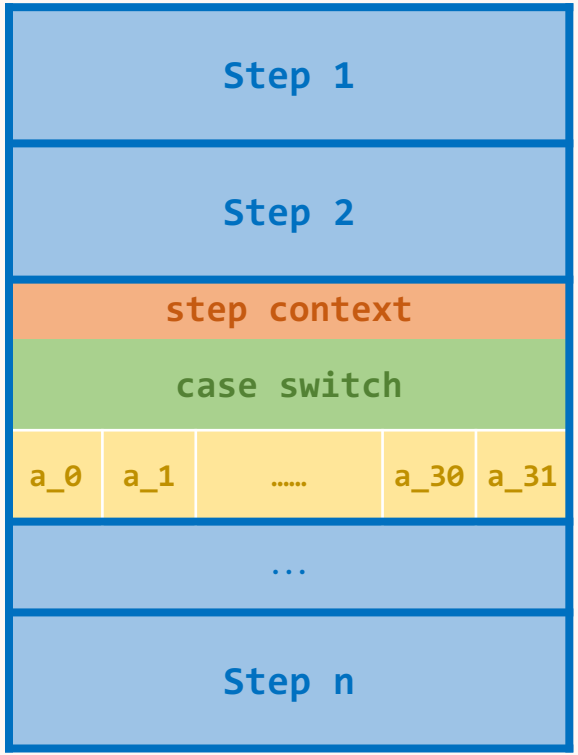
X / Twitter: @yezhang1998

Circuit - Randomness

Scroll



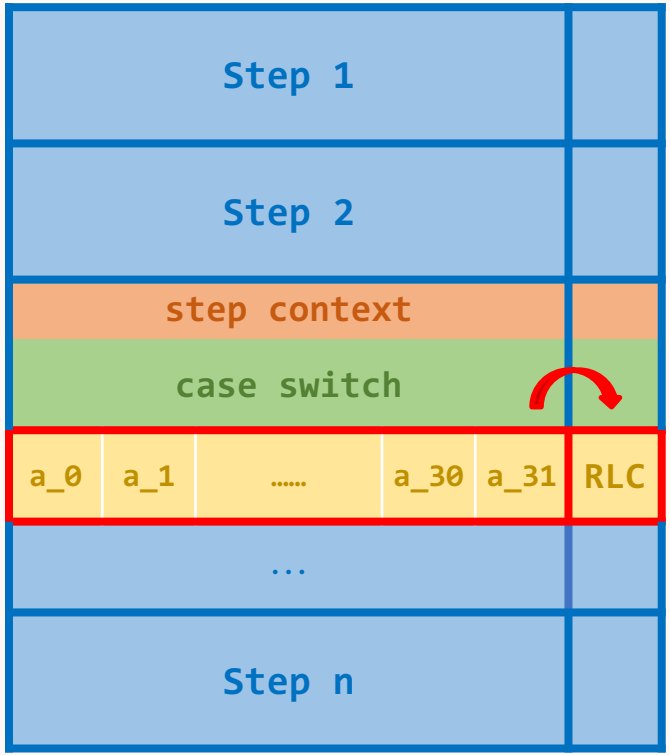
Circuit - Randomness



- Break down 256-bit word into 32 8-bit limbs.

$$A = a_0 + a_1 * 256 + a_2 * 256^2 + \cdots + a_{31} * 256^{31}$$

Circuit - Randomness



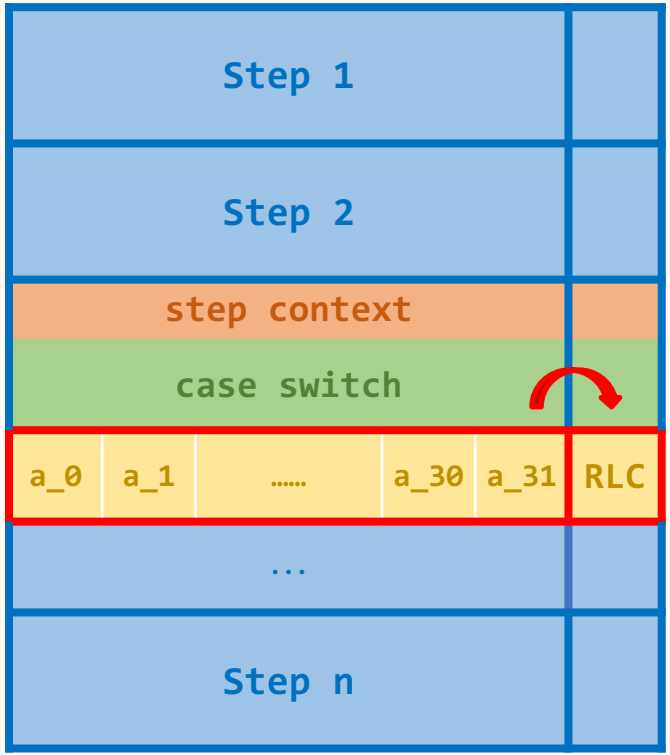
- Break down 256-bit word into 32 8-bit limbs.

$$A = a_0 + a_1 * 256 + a_2 * 256^2 + \cdots + a_{31} * 256^{31}$$

- Encode EVM word using RLC (Random Linear Combination)

$$A_{RLC} \equiv a_0 + a_1 * \theta + a_2 * \theta^2 + \cdots + a_{31} * \theta^{31} \pmod{F_p}$$

Circuit - Randomness



- Break down 256-bit word into 32 8-bit limbs.

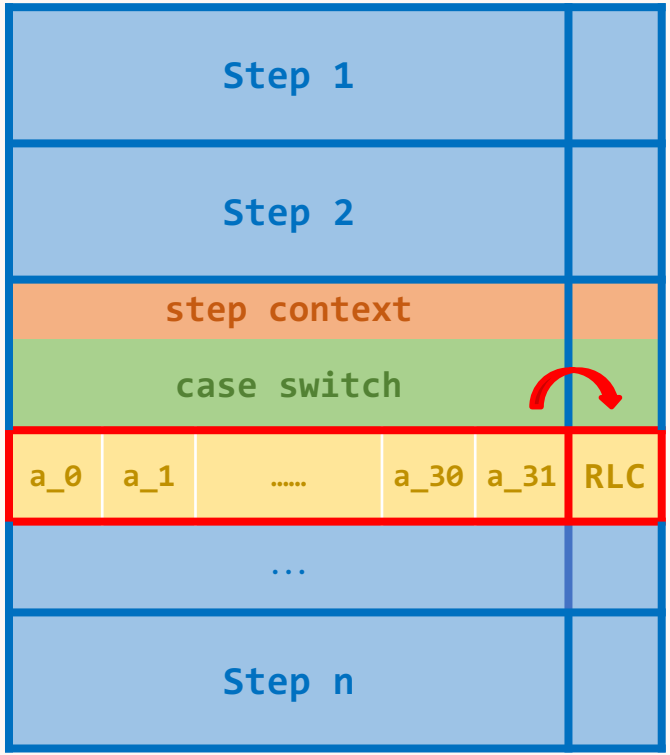
$$A = a_0 + a_1 * 256 + a_2 * 256^2 + \cdots + a_{31} * 256^{31}$$

- Encode EVM word using RLC (Random Linear Combination)

$$A_{RLC} \equiv a_0 + a_1 * \theta + a_2 * \theta^2 + \cdots + a_{31} * \theta^{31} \pmod{F_p}$$

- θ should be computed after $a_0, \ldots, a_{31}$ are fixed
  - Multi-phase prover: synthesis part of witness, derive witness

Circuit - Randomness



- Break down 256-bit word into 32 8-bit limbs.

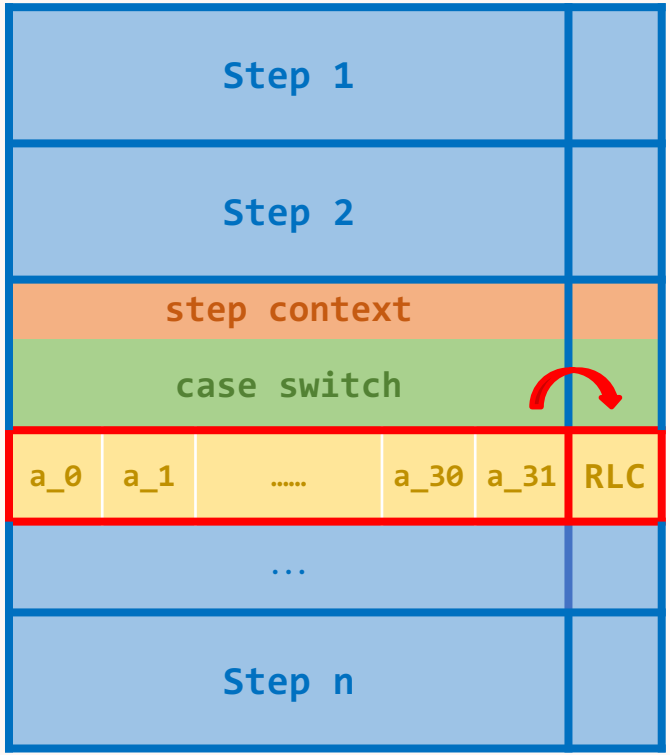
$$A = a_0 + a_1 * 256 + a_2 * 256^2 + \cdots + a_{31} * 256^{31}$$

- Encode EVM word using RLC (Random Linear Combination)

$$A_{RLC} \equiv a_0 + a_1 * \theta + a_2 * \theta^2 + \cdots + a_{31} * \theta^{31} \pmod{F_p}$$

- θ should be computed after $a_0, \ldots, a_{31}$ are fixed
  - Multi-phase prover: synthesis part of witness, derive witness
- RLC is useful in many places
  - Compress EVM word into one value
  - Encode dynamic length input
  - Lookup layout optimization

Circuit - Randomness



- Break down 256-bit word into 32 8-bit limbs.

$$A = a_0 + a_1 * 256 + a_2 * 256^2 + \cdots + a_{31} * 256^{31}$$

- Encode EVM word using RLC (Random Linear Combination)

$$A_{RLC} \equiv a_0 + a_1 * \theta + a_2 * \theta^2 + \cdots + a_{31} * \theta^{31} \pmod{F_p}$$

- θ should be computed after $a_0, \ldots, a_{31}$ are fixed
  - Multi-phase prover: synthesis part of witness, derive witness
- **RLC is useful in many places, remove it?**
  - Compress EVM word into one value → high, low for EVM word
  - Encode dynamic length input → fixed chunk, dynamic times
  - Lookup layout optimization

Circuit - Layout

Scroll

Execution trace

step	opcode
1	PUSH1 80
2	PUSH1 40
3	ADD
...	...
...	...
n	RETURN

Circuit - Layout

Scroll

Execution trace

step	opcode
1	PUSH1 80
2	PUSH1 40
3	ADD
...	...
...	...
n	RETURN

- The execution trace is dynamic
  - → enable different constraints
  - → permutation is not fixed
  - → hard to use standard gates

Circuit - Layout

Scroll

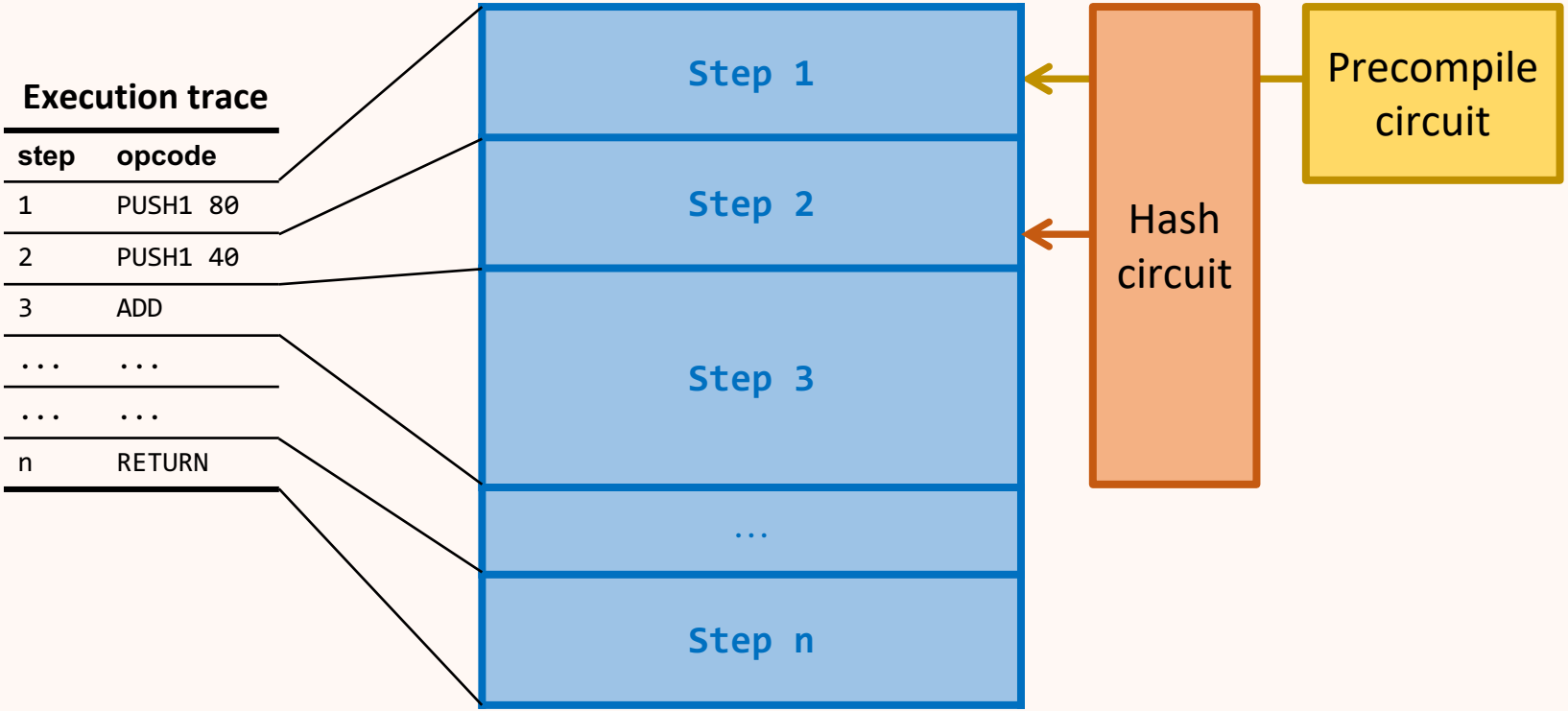
Execution trace

step	opcode
1	PUSH1 80
2	PUSH1 40
3	ADD
...	...
...	...
n	RETURN

- The execution trace is dynamic
  - → enable different constraints
  - → permutation is not fixed
  - → hard to use standard gates
- **Better way to layout?**
  - We have 2000+ custom gates
  - Different rotation to access cells

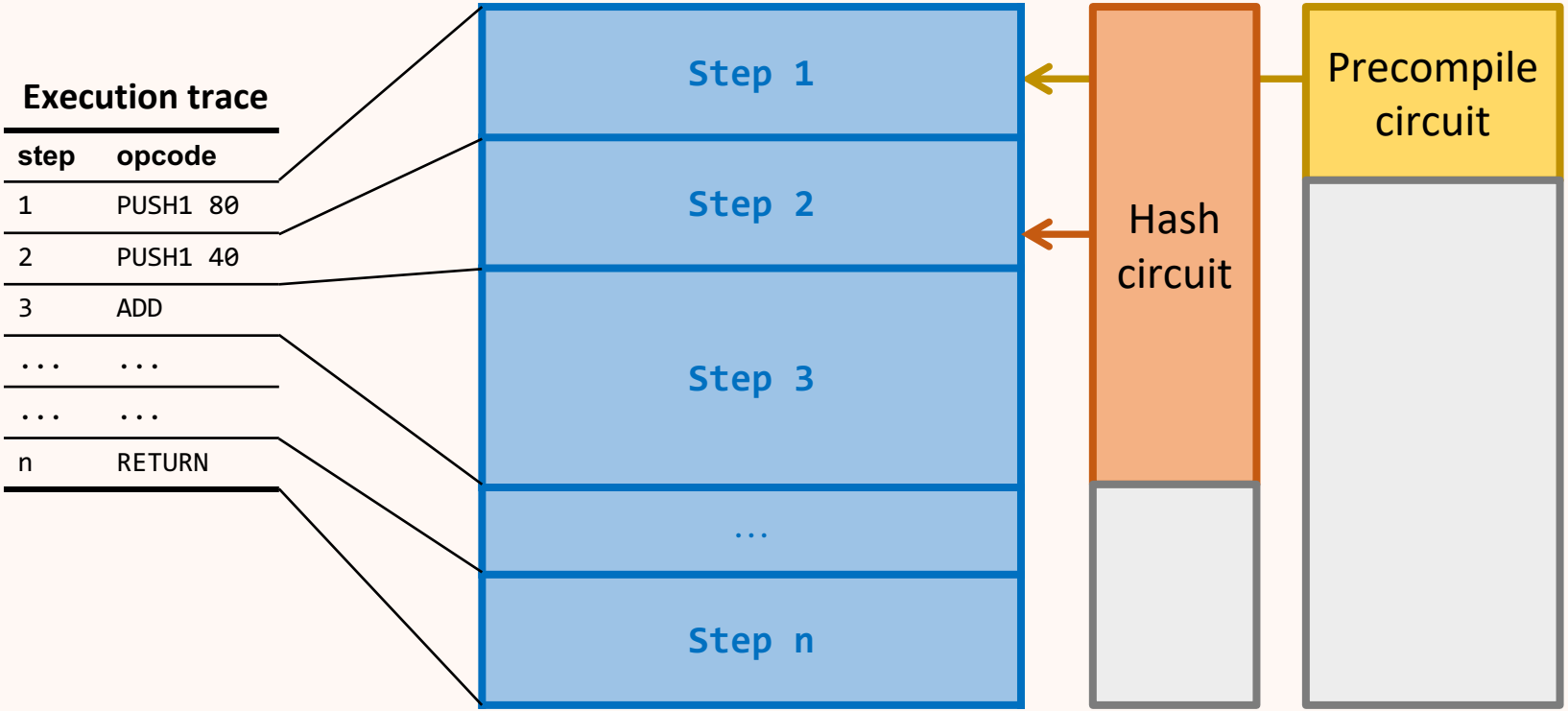
Circuit - Dynamic size

Scroll



Circuit - Dynamic size

Scroll



Circuit - Dynamic size

Scroll

Execution trace

step	opcode
1	PUSH1 80
2	PUSH1 40
3	ADD
...	...
...	...
n	RETURN

- Some bad influences

  i.e. Mload is more costly (more rows)

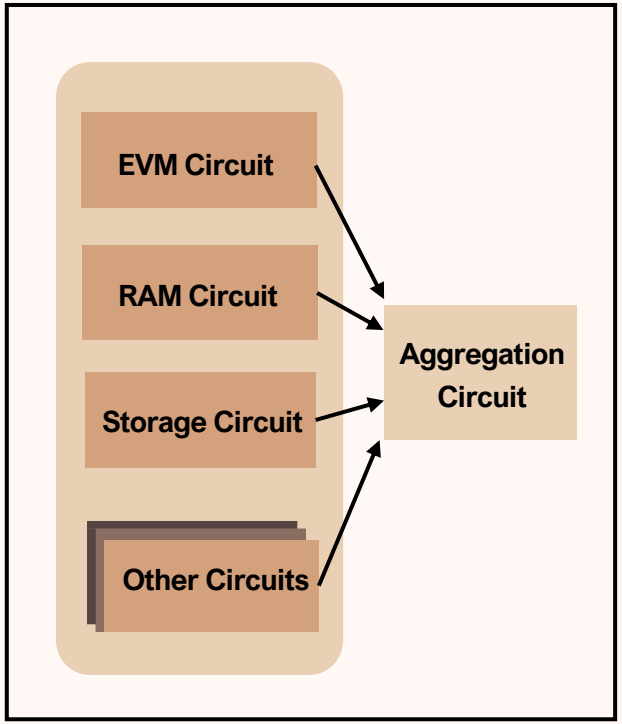
  i.e. Maximum number of Keccaks

  i.e. Pay larger proving cost for padding

- **Can we make zkEVM dynamic?**

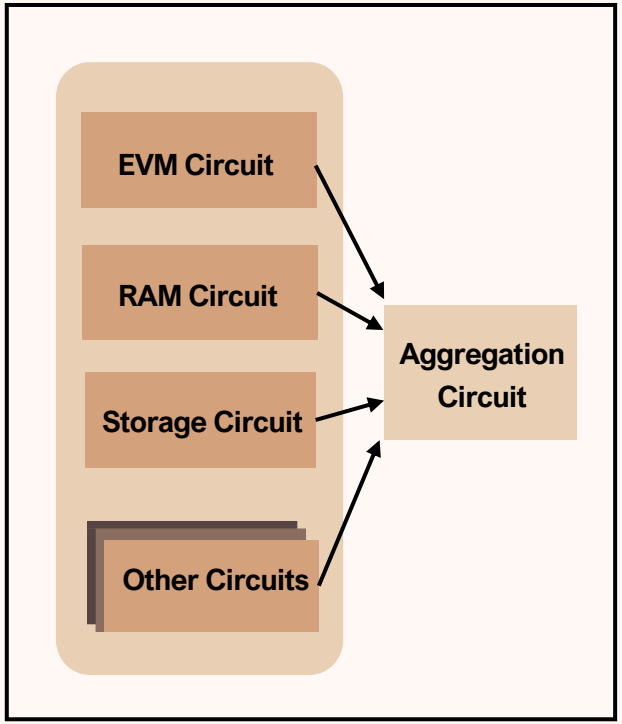
Prover – Hardware & Algorithm

Scroll



Prover – Hardware & Algorithm

Scroll

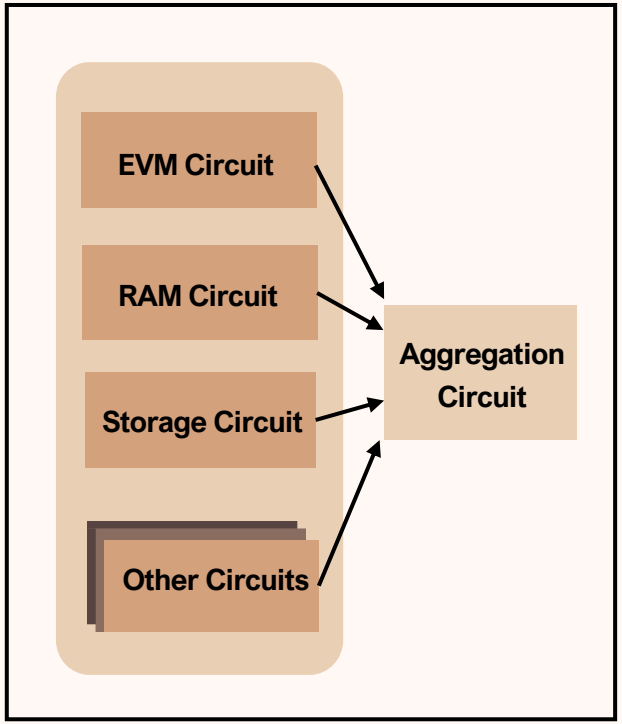


- Our prover
  - GPU can make MSM & NTT really fast

    Bottleneck moves to witness generation & data copy
  - Need large CPU memory (1TB -> 300GB+)
- **Hardware friendly prover?**
  - Parallelizable & Low peak memory
  - Don’t ignore the witness generation
  - Run on cheap machines, more decentralized

Prover – Hardware & Algorithm

Scroll



- **Best way to compose different proof system?**
  - The first layer needs to be "expressive"
  - The second layer needs to be verifier efficient (in EVM)
  - **Should we move to smaller field?**

    (Breakdown/FRI with Goldilocks, Mersenne prime)
  - **Should we stick to EC-based constructions?**

    (SuperNova, Cyclic elliptic curve with fast MSM)
  - More options waiting for you → Reach out to us!