LAYER TWO
AMSTERDAM

DEVCONNECT
AMS

L2BEAT

Ye Zhang

An overview of zkEVM

Design & Challenges

STARKWARE / ARBITRUM / OPTIMISM / BOBA NETWORK / METIS / FUEL LABS / NERVOS / ZKSYNC

What is Scroll?

We are building a native zkEVM solution

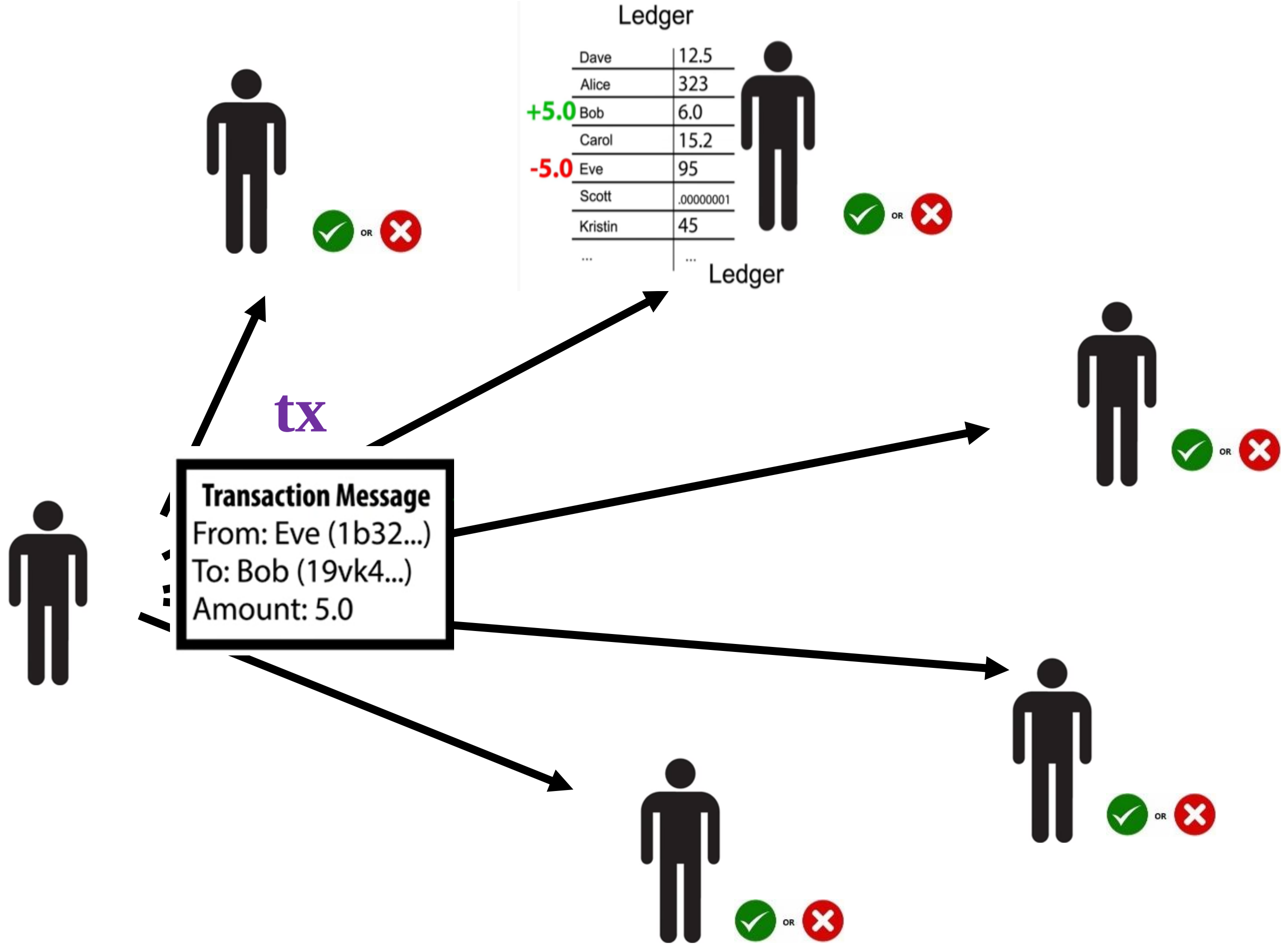
Our vision

Scroll

- EVM-equivalent zk-Rollup → Best user and developer experience
- Decentralization → Decentralized proving network
- Align with Ethereum → Open-source, co-build with community

The motivation

The problem of Layer 1



Transaction Message
From: Eve (1b32...)
To: Bob (19vk4...)
Amount: 5.0

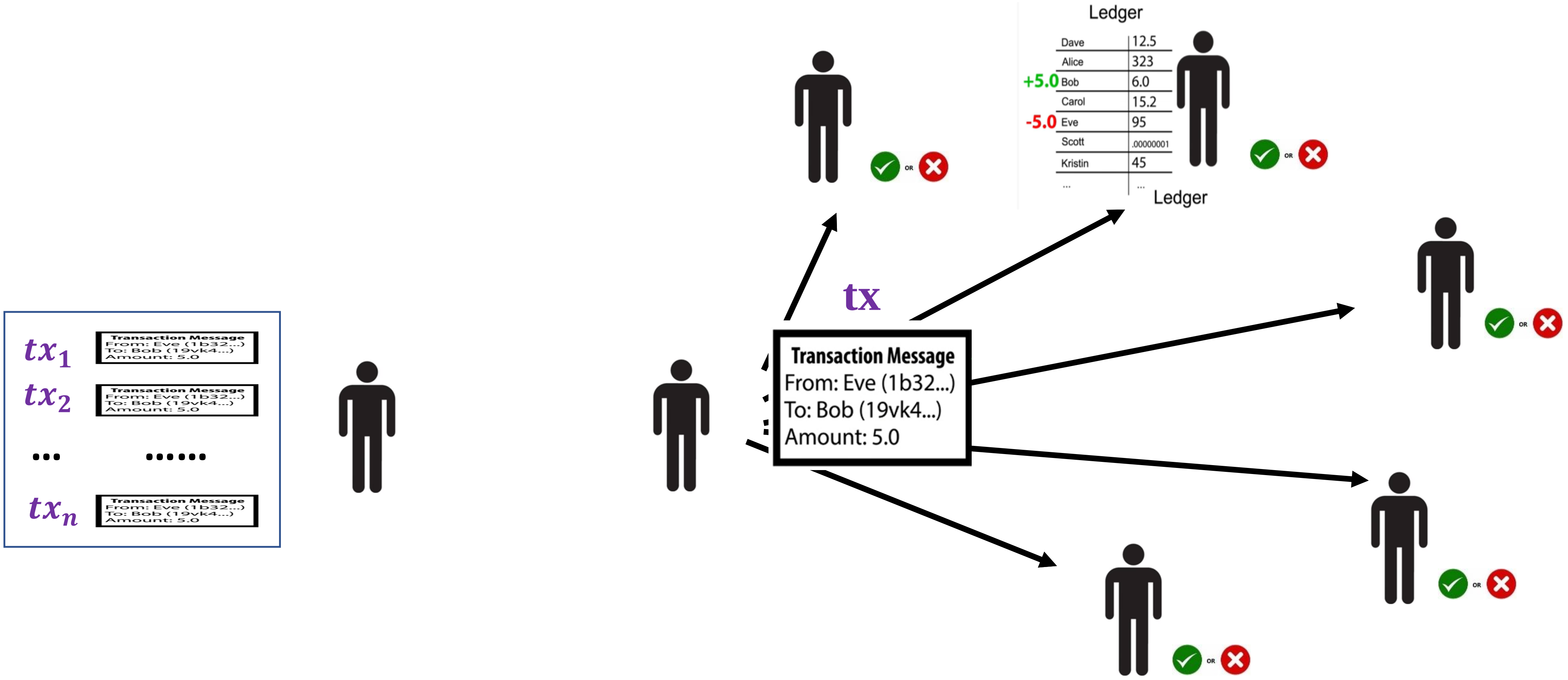
Ledger

	Dave	12.5
	Alice	323
+5.0	Bob	6.0
	Carol	15.2
-5.0	Eve	95
	Scott	.00000001
	Kristin	45
	...	...

Ledger

The idea behind zk-Rollup

Scroll



tx_1 Transaction Message From: Eve (1b32...) To: Bob (19vk4...) Amount: 5.0

tx_2 Transaction Message From: Eve (1b32...) To: Bob (19vk4...) Amount: 5.0

... .

tx_n Transaction Message From: Eve (1b32...) To: Bob (19vk4...) Amount: 5.0

tx

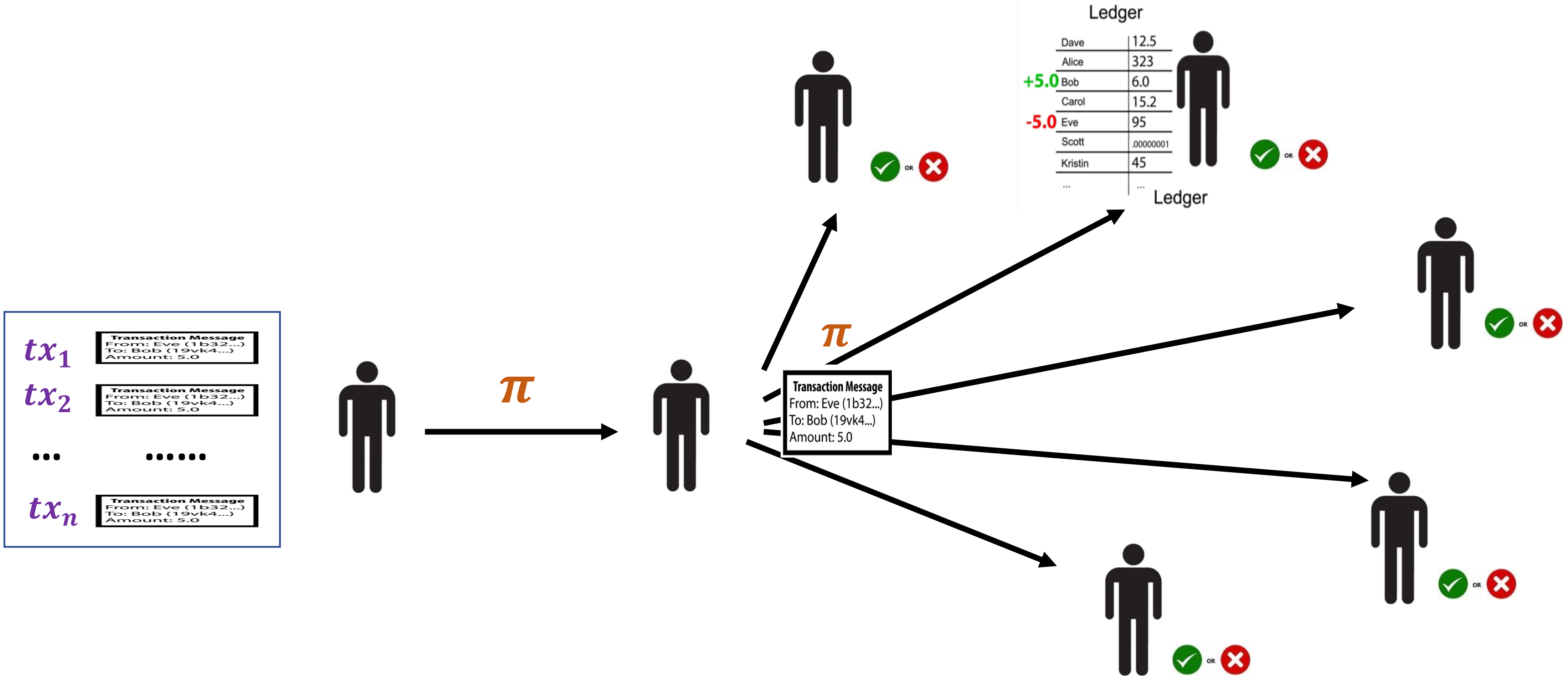
Transaction Message
From: Eve (1b32...)
To: Bob (19vk4...)
Amount: 5.0

Ledger

	Dave	12.5
	Alice	323
+5.0	Bob	6.0
	Carol	15.2
-5.0	Eve	95
	Scott	.00000001
	Kristin	45
	...	...

Ledger

The idea behind zk-Rollup



tx_1 Transaction Message From: Eve (1b32...) To: Bob (19vk4...) Amount: 5.0

tx_2 Transaction Message From: Eve (1b32...) To: Bob (19vk4...) Amount: 5.0

… ……

tx_n Transaction Message From: Eve (1b32...) To: Bob (19vk4...) Amount: 5.0

π

Transaction Message
From: Eve (1b32...)
To: Bob (19vk4...)
Amount: 5.0

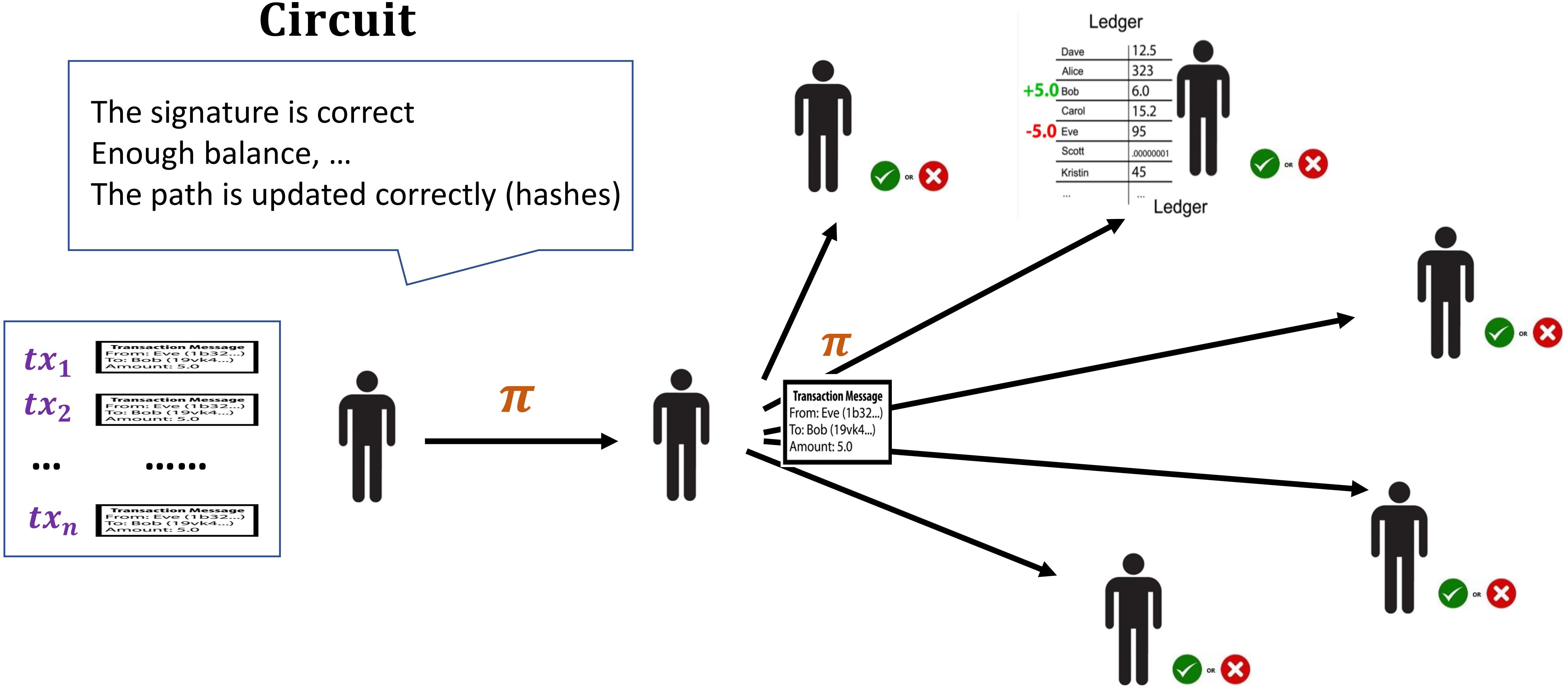
Ledger

	Dave	12.5
	Alice	323
+5.0	Bob	6.0
	Carol	15.2
-5.0	Eve	95
	Scott	.00000001
	Kristin	45
	…	…

Ledger

However…

Circuit



The signature is correct
Enough balance, …
The path is updated correctly (hashes)

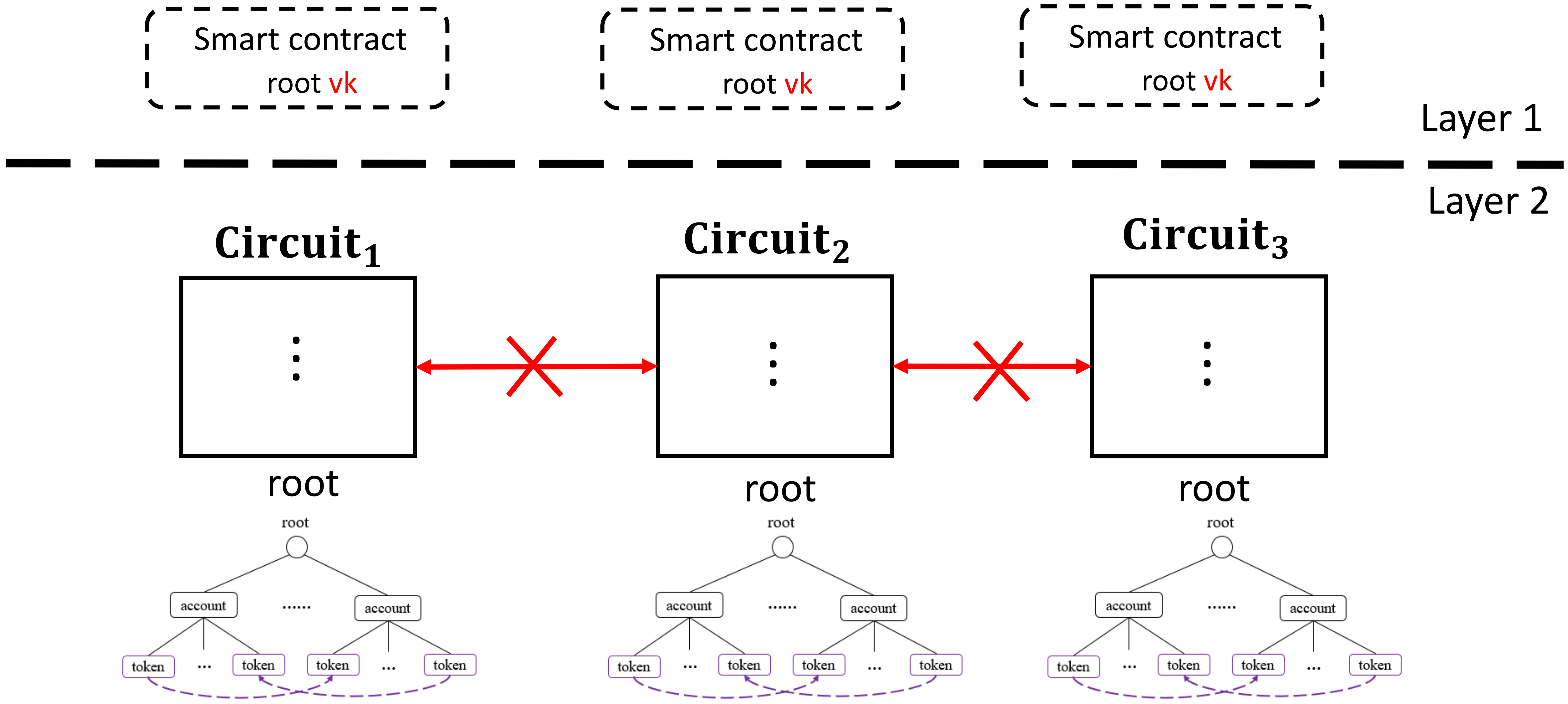
Transaction Message
From: Eve (1b32...)
To: Bob (19vk4...)
Amount: 5.0

Ledger

	Dave	12.5
	Alice	323
+5.0	Bob	6.0
	Carol	15.2
-5.0	Eve	95
	Scott	.00000001
	Kristin	45
	…	…

Ledger

Application specific zk-Rollups



Smart contract
root vk

Smart contract
root vk

Smart contract
root vk

Layer 1

Layer 2

$\mathbf{Circuit_1}$

$\mathbf{Circuit_2}$

$\mathbf{Circuit_3}$

root

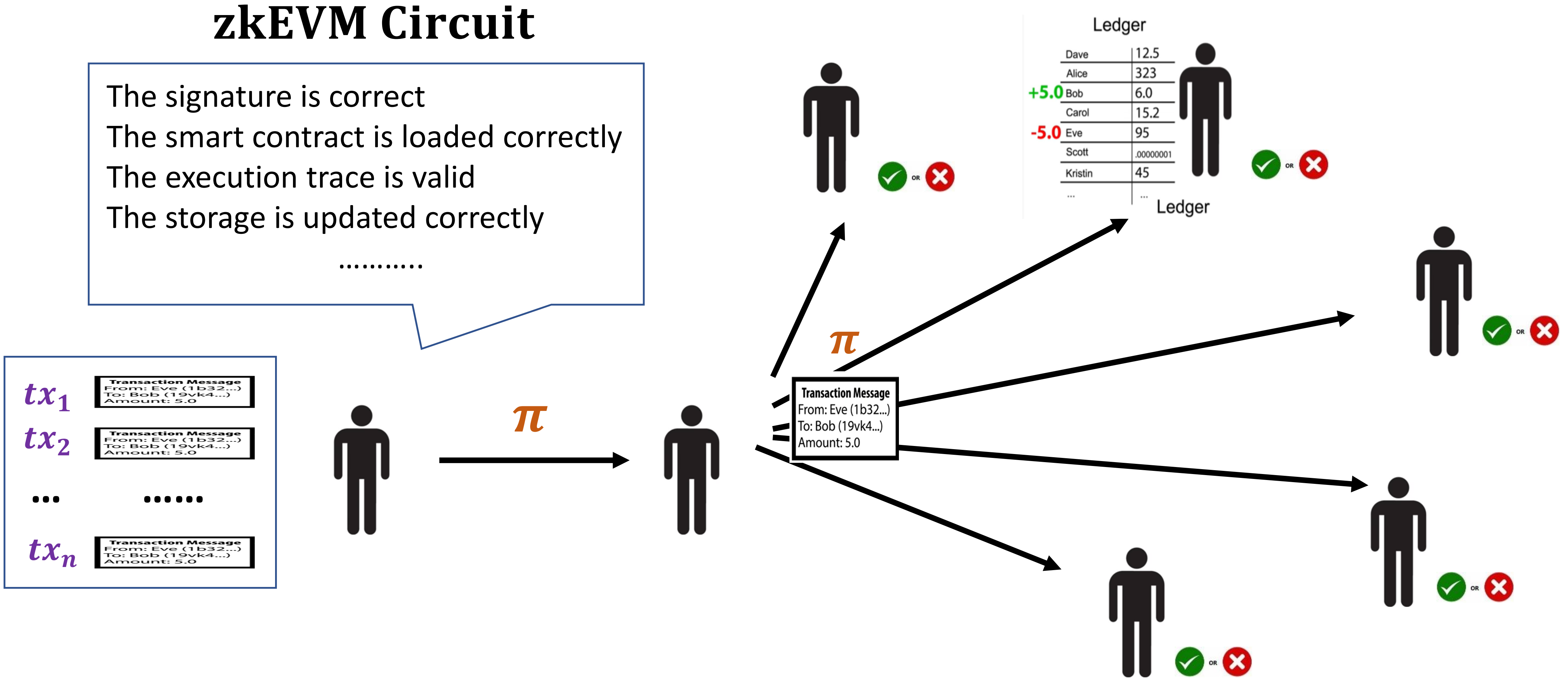
root

root

We simulate EVM in circuit

Scroll

zkEVM Circuit



The signature is correct
The smart contract is loaded correctly
The execution trace is valid
The storage is updated correctly
....

tx_1 Transaction Message From: Eve (1b32...) To: Bob (19vk4...) Amount: 5.0

tx_2 Transaction Message From: Eve (1b32...) To: Bob (19vk4...) Amount: 5.0

... .

tx_n Transaction Message From: Eve (1b32...) To: Bob (19vk4...) Amount: 5.0

Transaction Message
From: Eve (1b32...)
To: Bob (19vk4...)
Amount: 5.0

Ledger

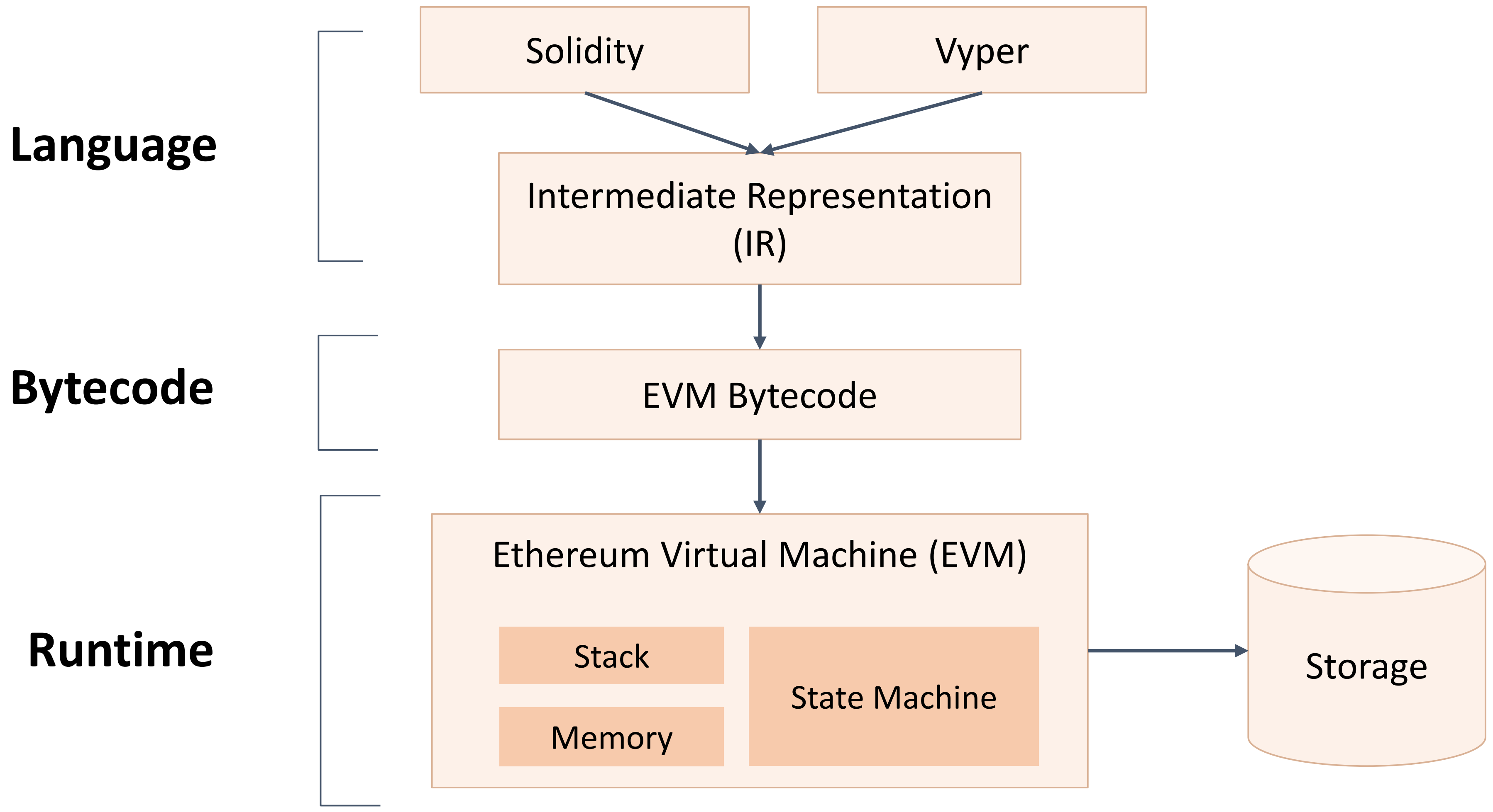
	Dave	12.5
	Alice	323
+5.0	Bob	6.0
	Carol	15.2
-5.0	Eve	95
	Scott	.00000001
	Kristin	45
	...	...

Ledger

The Comparison

How is a smart contract compiled and executed?

Scroll



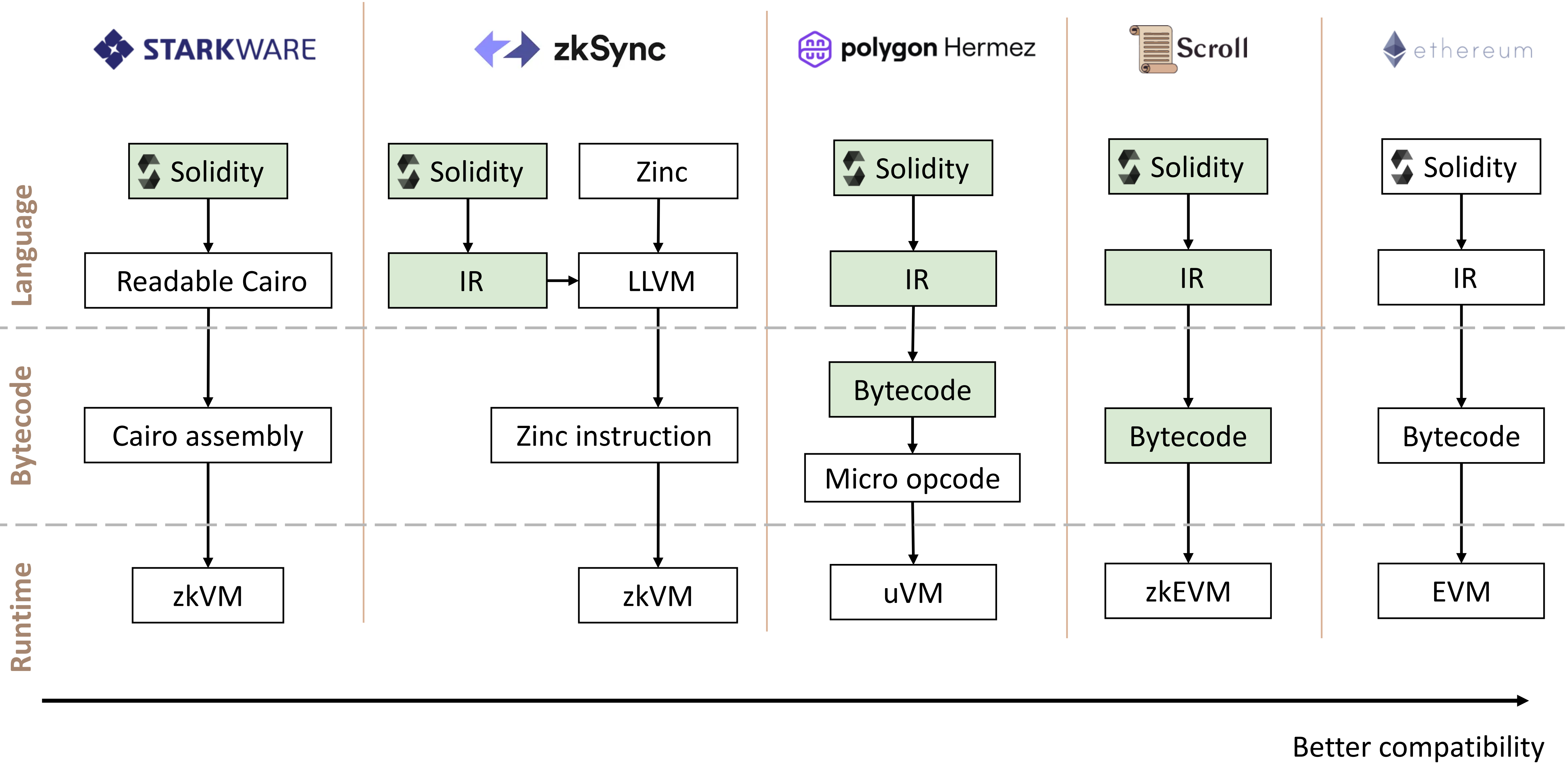
What is zkEVM? (according to Justin Drake)

Three flavours of zkEVM:

- **language-level**:
  Transpile an EVM-friendly language (Solidity or Yul) to a SNARK-friendly VM which may be different 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 the Consensys-led effort.

- **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.

STARKWARE
zkSync
polygon Hermez
Scroll
ethereum
Language
Bytecode
Runtime
Solidity
Readable Cairo
Cairo assembly
zkVM
Solidity
Zinc
IR
LLVM
Zinc instruction
zkVM
Solidity
IR
Bytecode
Micro opcode
uVM
Solidity
IR
Bytecode
zkEVM
Solidity
IR
Bytecode
EVM
Better compatibility

Why are we walking along this path?

Why are we walking along this path?

- **Security inherited from EVM**
  - This model has stood the test of time (i.e. gas)

Why are we walking along this path?

- **Security inherited from EVM**
  - This model has stood the test of time (i.e. gas)
- **Compatible with all infrastructure seamlessly**
  - All DApps, wallets, tools and interfaces
  - Re-use Geth as our node for execution

Why are we walking along this path?

- **Security inherited from EVM**
  - This model has stood the test of time (i.e. gas)
- **Compatible with all infrastructure seamlessly**
  - All DApps, wallets, tools and interfaces
  - Re-use Geth as our node for execution
- **High encapsulated complexity**
  - Hard for us to build, but it’s the best trade-off for users and developers
  - Doesn’t rely on external compiler

Why are we walking along this path?

- **Security inherited from EVM**
  - This model has stood the test of time (i.e. gas)
- **Compatible with all infrastructure seamlessly**
  - All DApps, wallets, tools and interfaces
  - Re-use Geth as our node for execution
- **High encapsulated complexity**
  - Hard for us to build, but it's the best trade-off for users and developers
  - Doesn't rely on external compiler
- **Efficiency problem solved through crypto and hardware**
  - Advances in ZK (i.e. custom gates, lookup arguments, recursive proof)
  - Hardware acceleration (i.e. ASIC/FPGA/GPU)

Why are we walking along this path?

Scroll

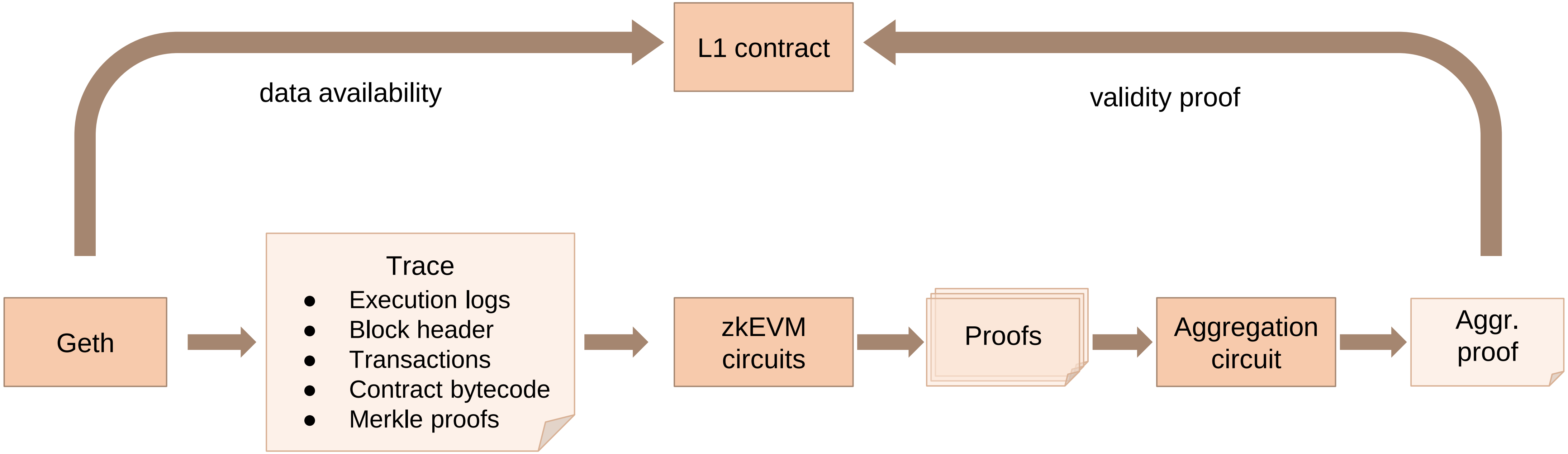
- **Security inherited from EVM**
  - This model has stood the test of time (i.e. gas)
- **Compatible with all infrastructure seamlessly**
  - All DApps, wallets, tools and interfaces
  - Re-use Geth as our node for execution
- **High encapsulated complexity**
  - Hard for us to build, but it’s the best trade-off for users and developers
  - Doesn’t rely on external compiler
- **Efficiency problem solved through crypto and hardware**
  - Advances in ZK (i.e. custom gates, lookup arguments, recursive proof)
  - Hardware acceleration (i.e. ASIC/FPGA/GPU)
- **Align with Ethereum**
  - A lot of credit goes to the community (EF appliedZKP team)
  - Push forward the end-goal of “zk-SNARK” everything

Scroll

The tech stack

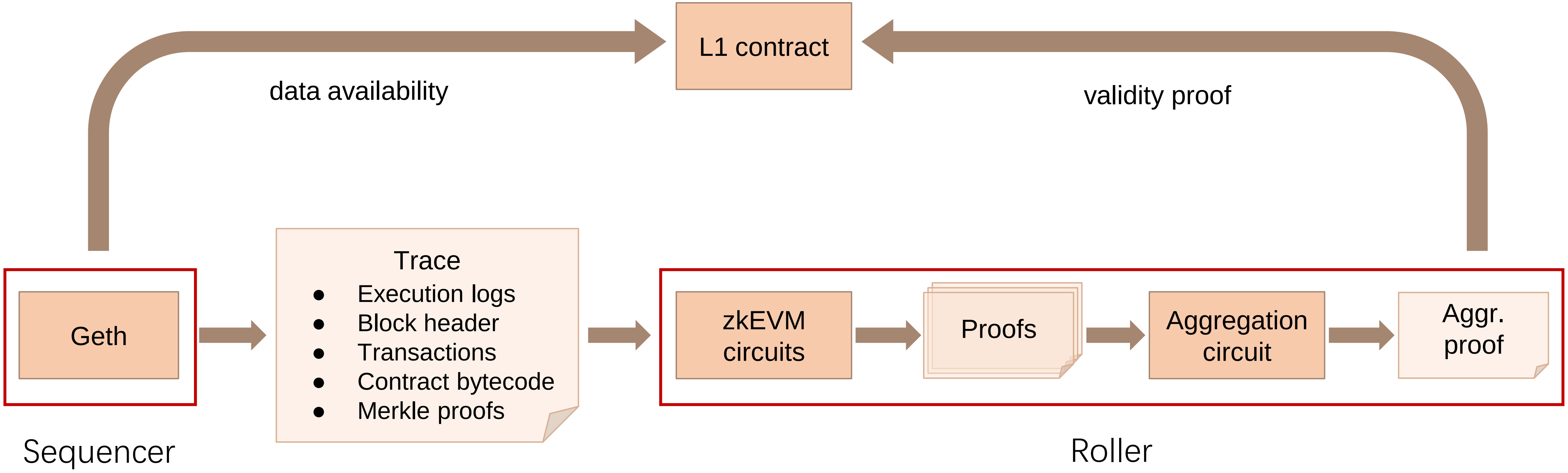
The workflow of zkEVM

Scroll



The workflow of zkEVM

Scroll

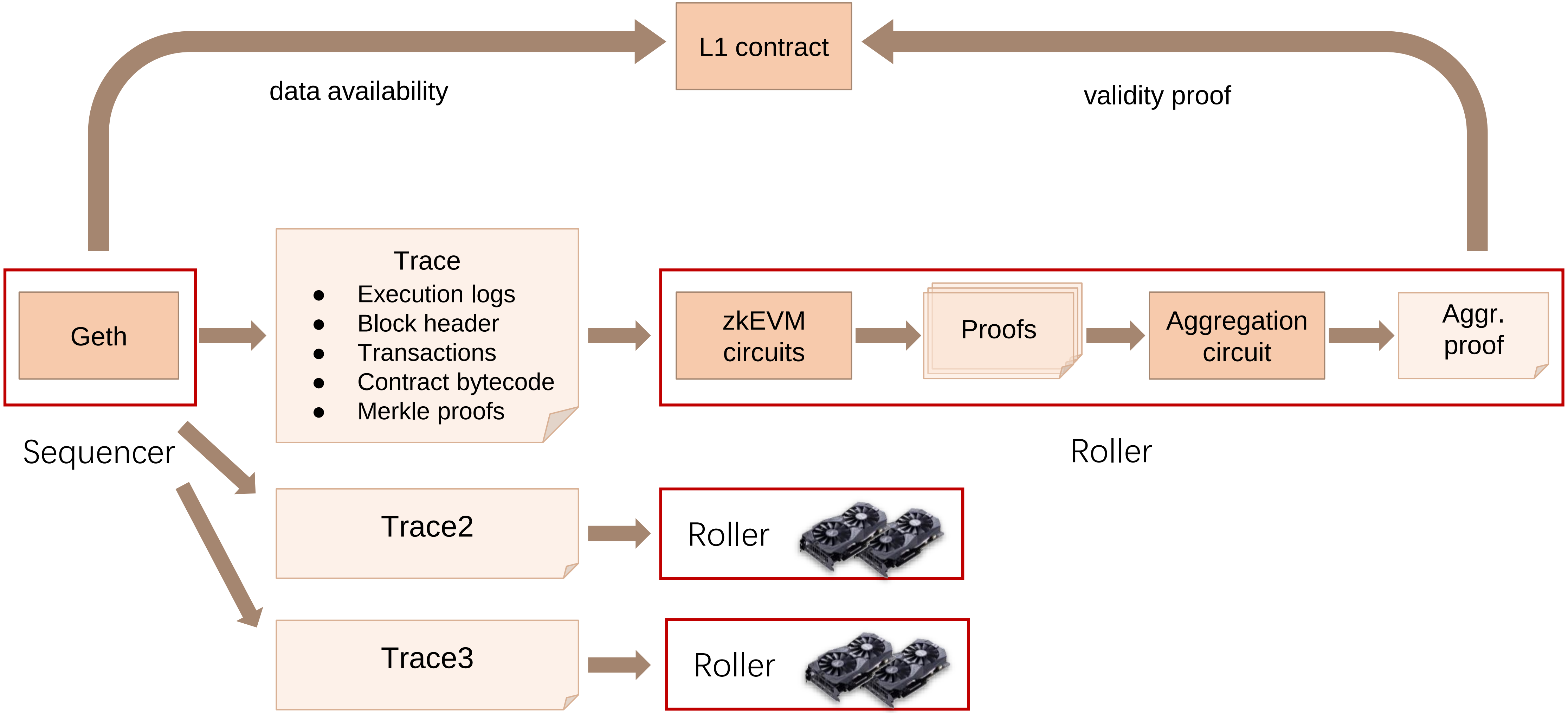


Trace

- Execution logs
- Block header
- Transactions
- Contract bytecode
- Merkle proofs

The workflow of zkEVM

Scroll



L1 contract

data availability

validity proof

Geth

Sequencer

Trace

- Execution logs
- Block header
- Transactions
- Contract bytecode
- Merkle proofs

zkEVM circuits

Proofs

Aggregation circuit

Aggr. proof

Roller

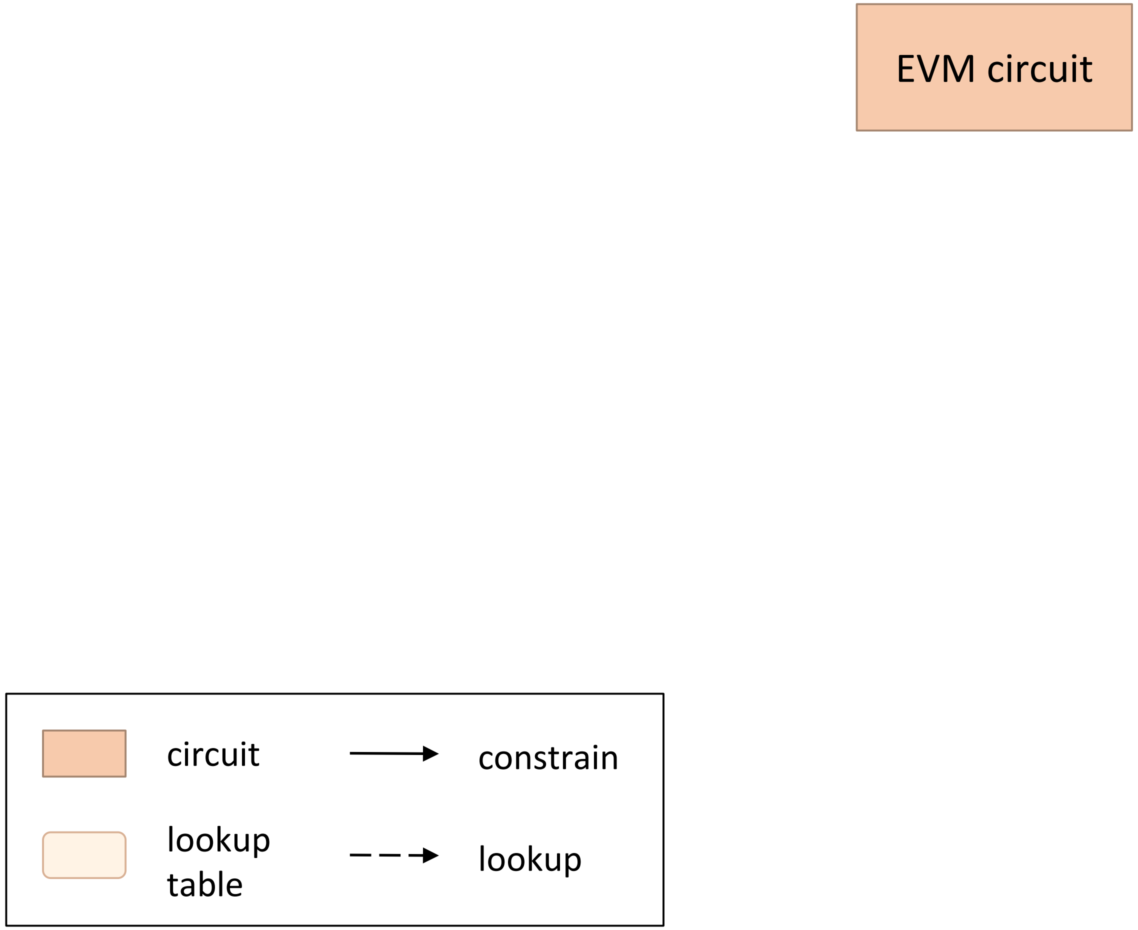
Trace2

Roller

Trace3

Roller

The architecture of zkEVM circuits

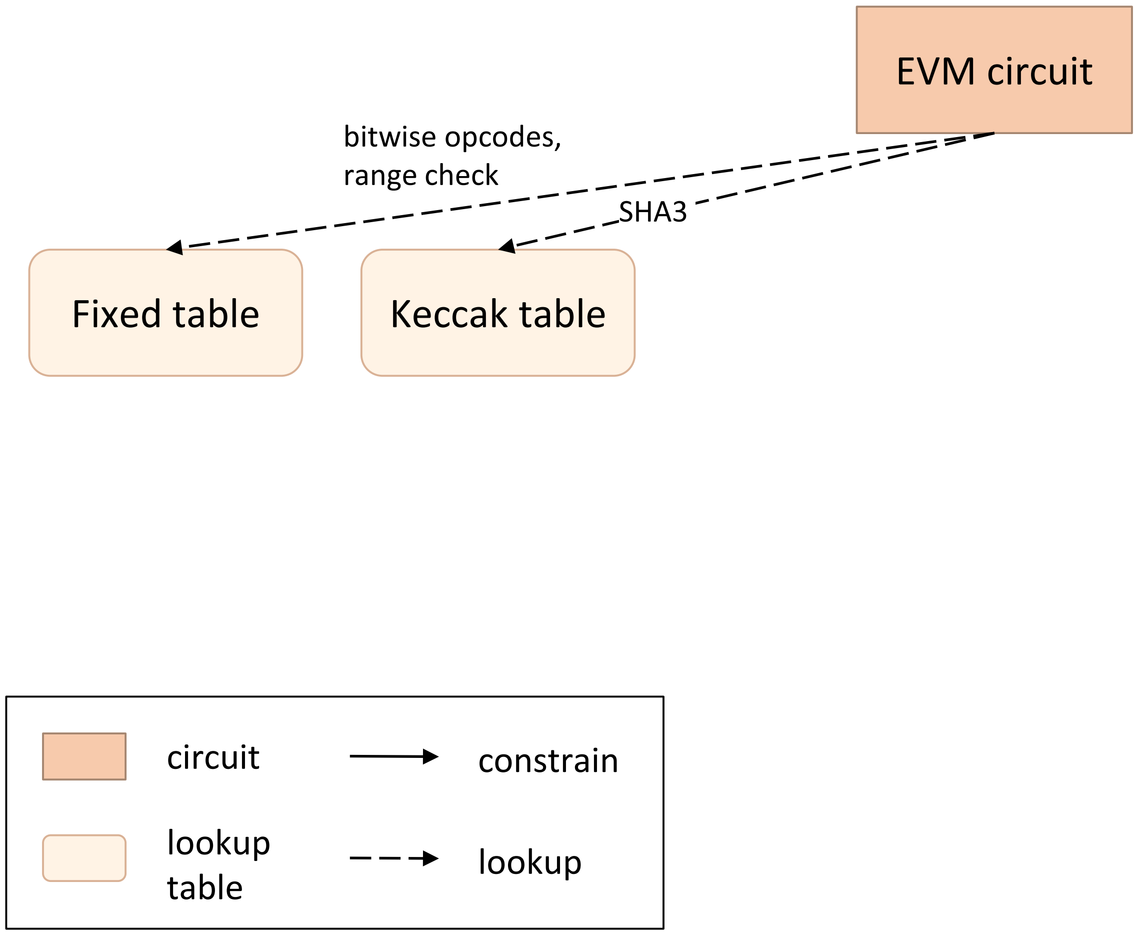


EVM circuit

Constrains the state machine

🟧	circuit	⟶	constrain
▢	lookup table	⇢	lookup

The architecture of zkEVM circuits



EVM circuit

bitwise opcodes,
range check

SHA3

Fixed table

Keccak table

circuit

lookup table

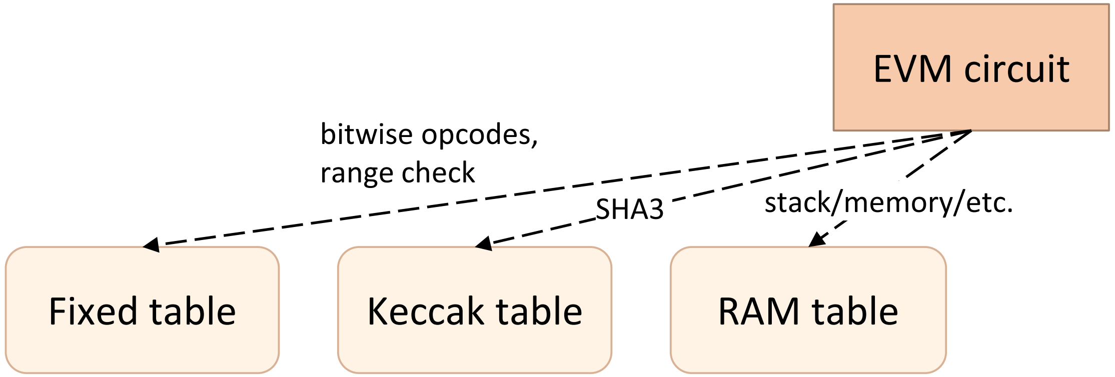
constrain

lookup

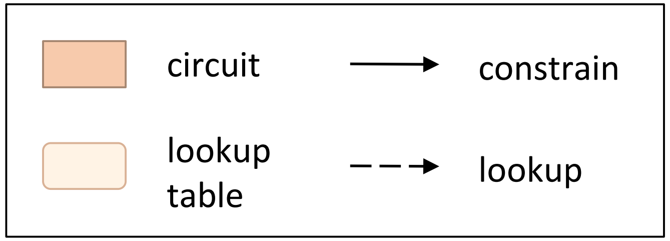
Challenge 1: zk-unfriendly opcodes

The architecture of zkEVM circuits

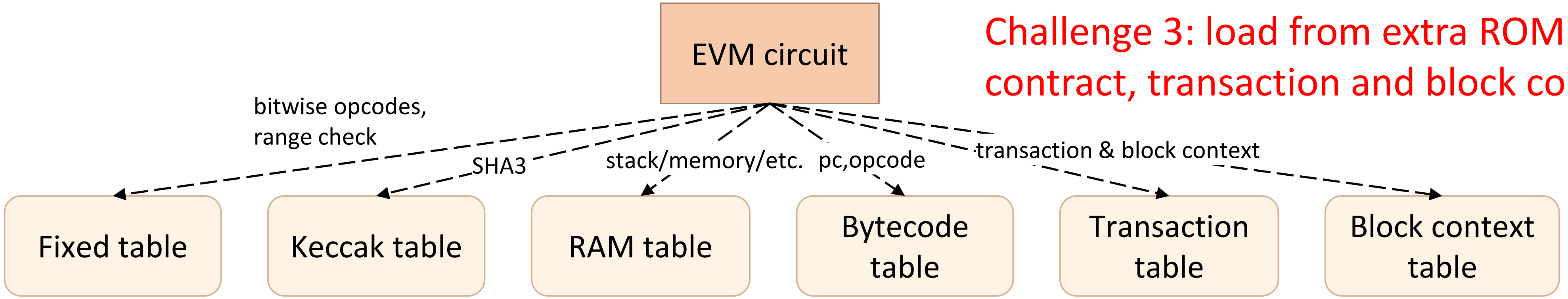
Scroll



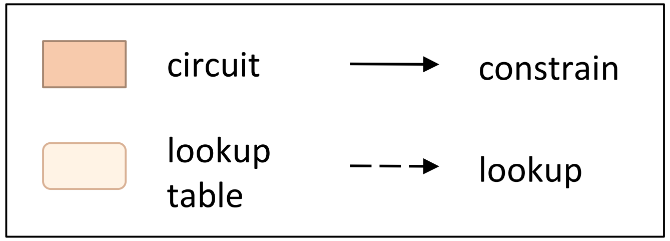
Challenge 2: verify stack/memory/storage operations



The architecture of zkEVM circuits



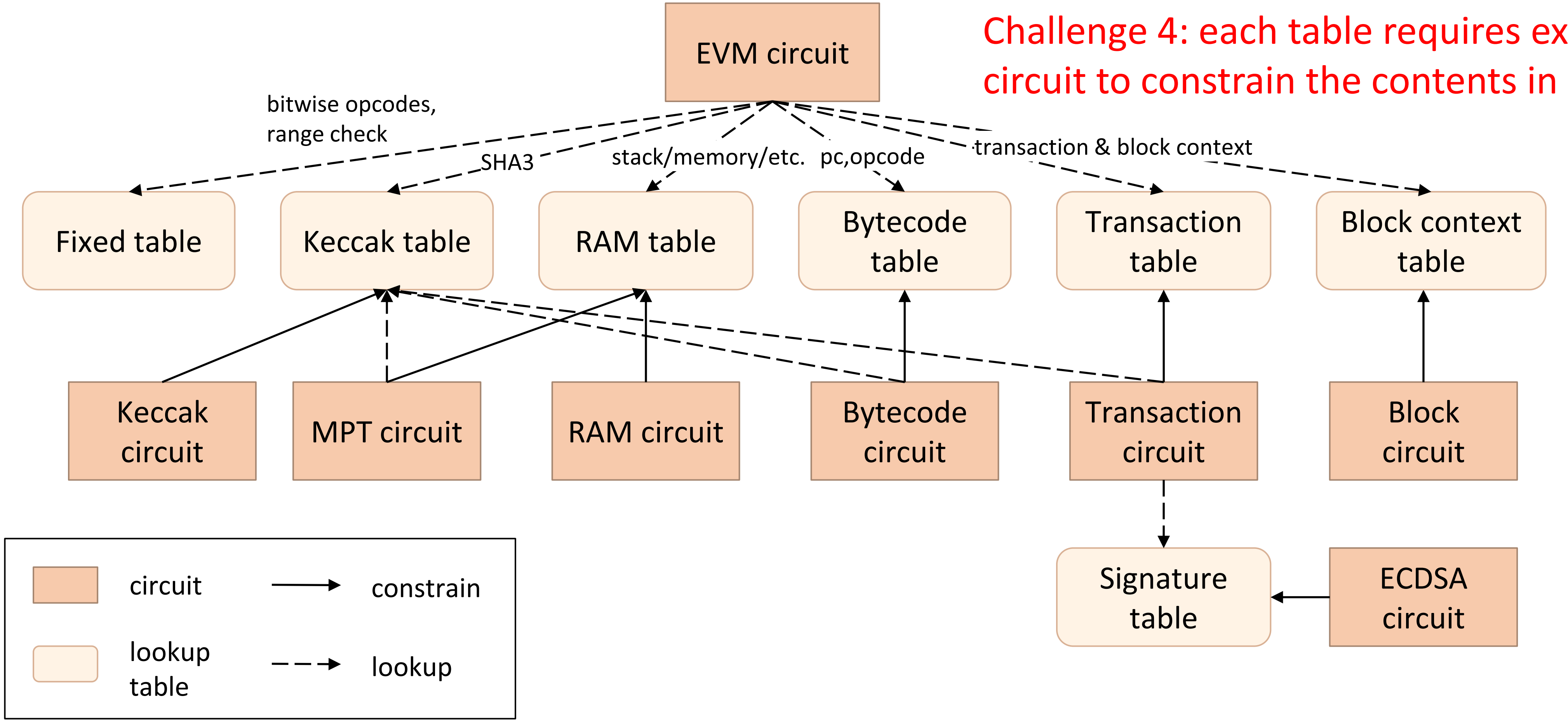
Challenge 3: load from extra ROM for contract, transaction and block contexts



The architecture of zkEVM circuits

Scroll

Challenge 4: each table requires extra circuit to constrain the contents in tables



More tech details

Plonkish Arithmetization

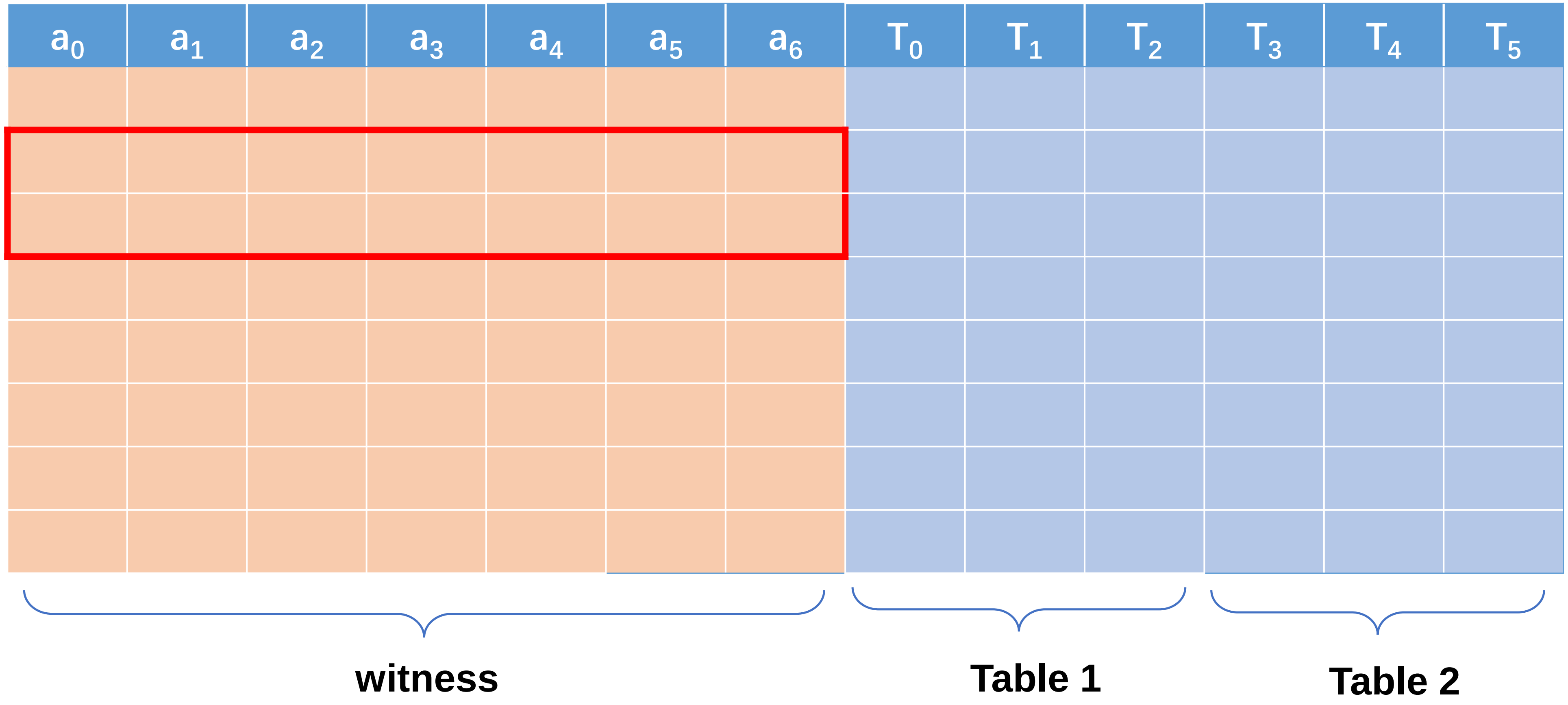
a_0	a_1	a_2	a_3	a_4	a_5	a_6	T_0	T_1	T_2	T_3	T_4	T_5

witness

Table 1

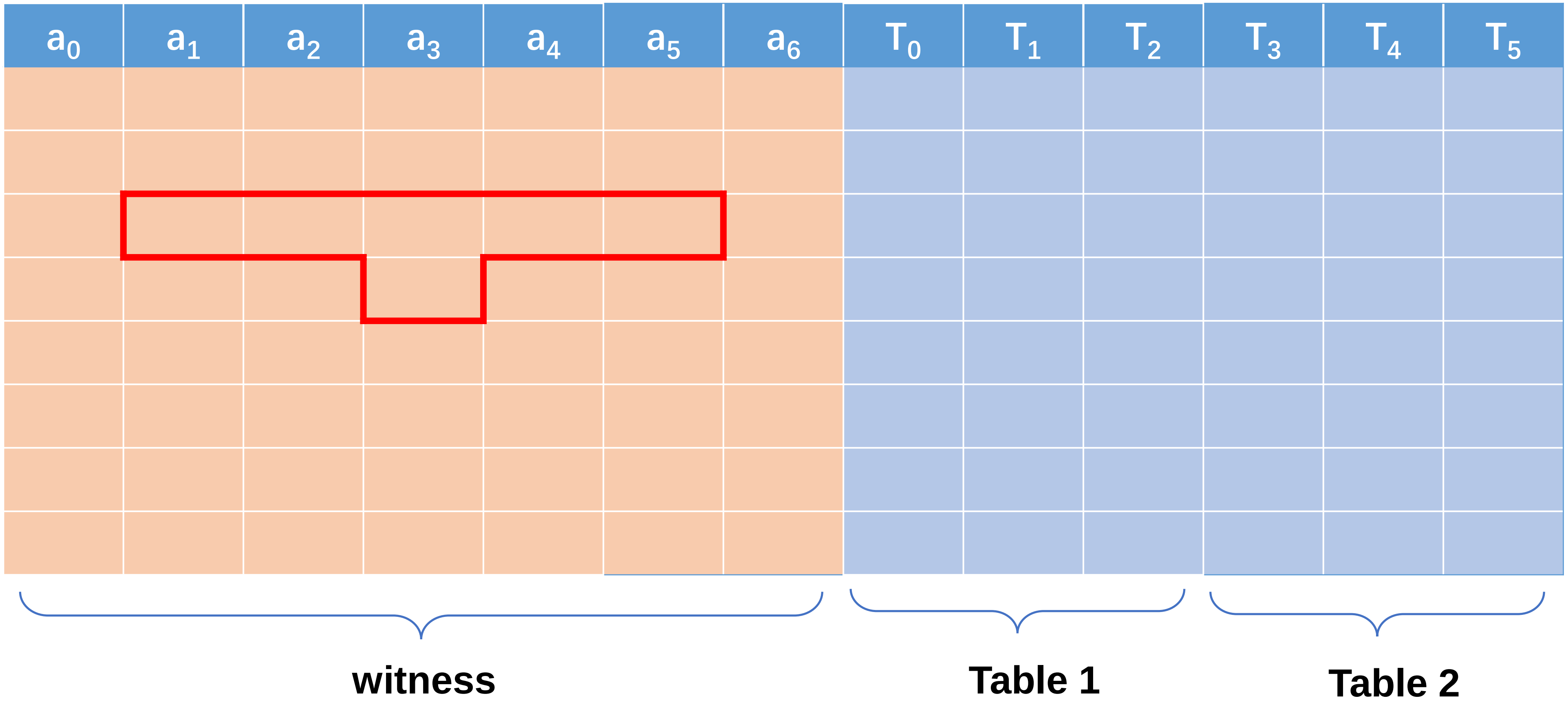
Table 2

Plonkish Arithmetization



a_0	a_1	a_2	a_3	a_4	a_5	a_6	T_0	T_1	T_2	T_3	T_4	T_5

Plonkish Arithmetization



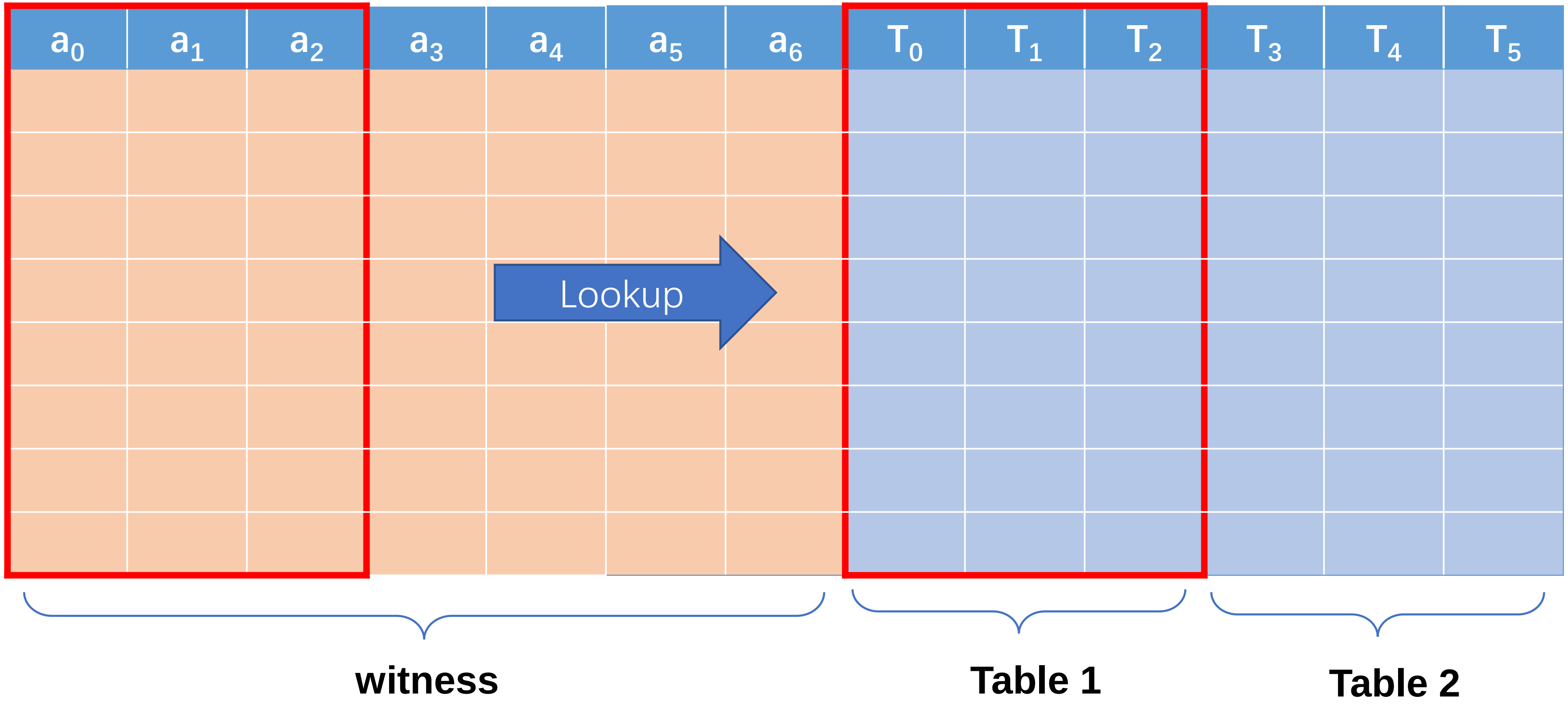
a_0 a_1 a_2 a_3 a_4 a_5 a_6 T_0 T_1 T_2 T_3 T_4 T_5

witness

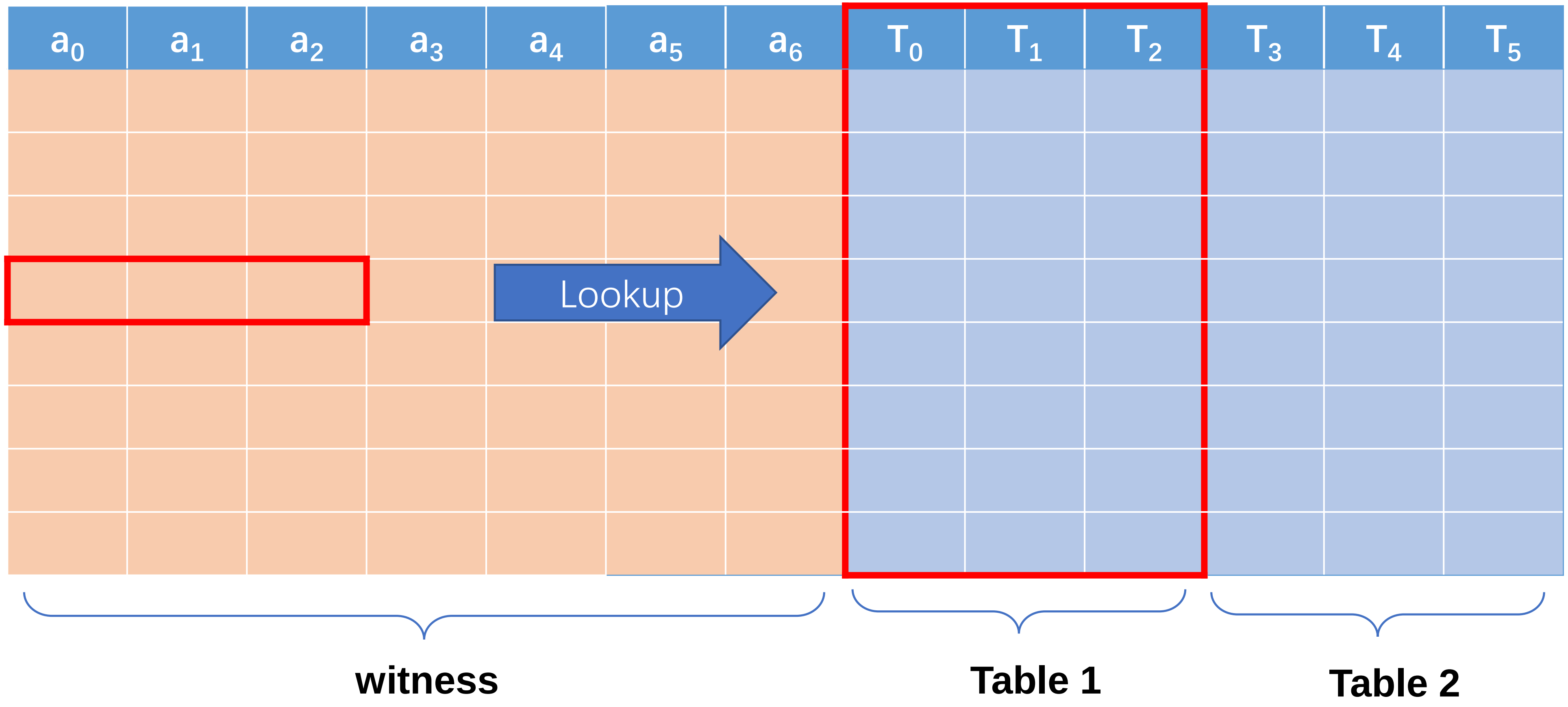
Table 1

Table 2

Plonkish Arithmetization



Plonkish Arithmetization



a_0 a_1 a_2 a_3 a_4 a_5 a_6 T_0 T_1 T_2 T_3 T_4 T_5

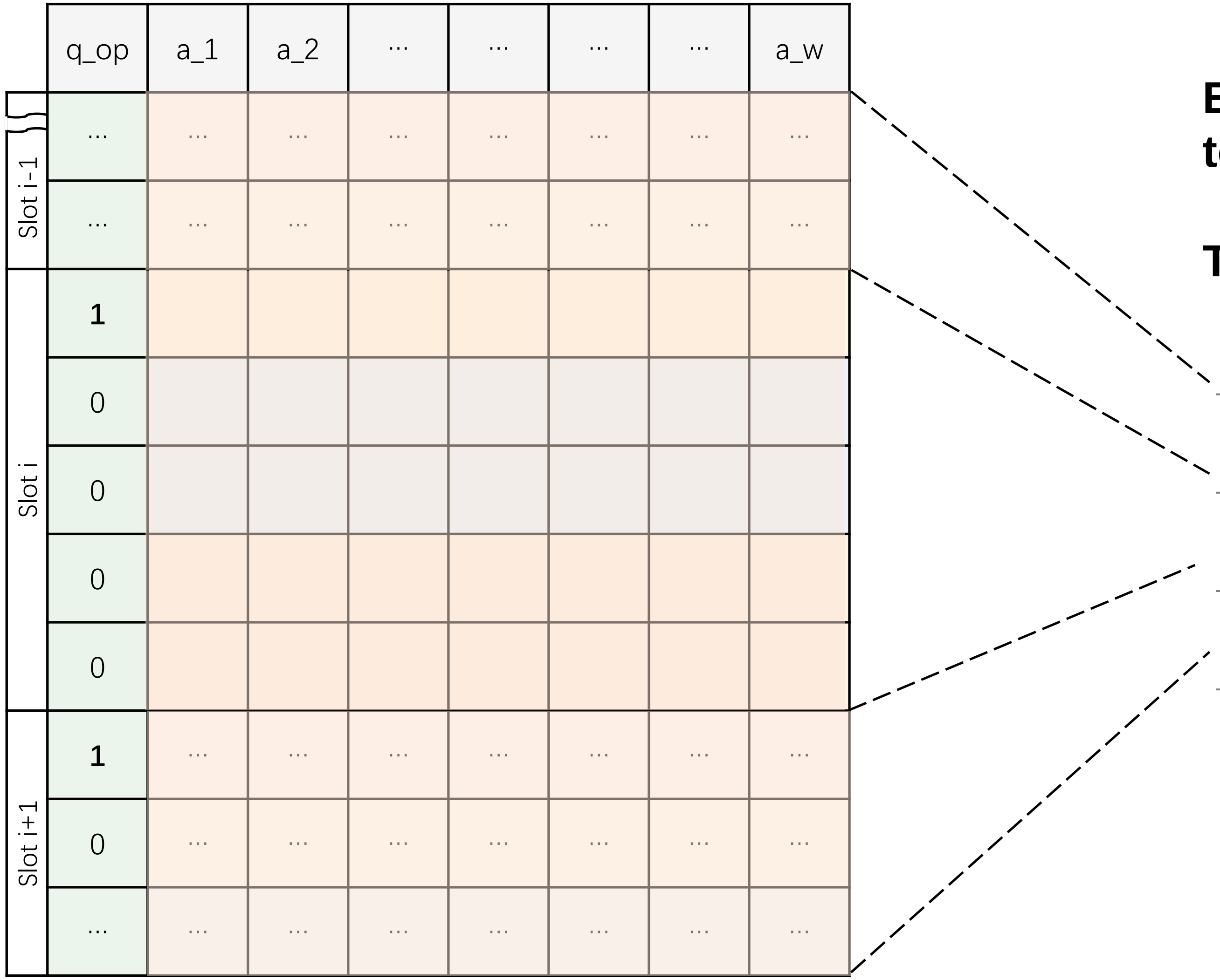
Lookup

witness

Table 1

Table 2

EVM Circuit



	q_op	a_1	a_2	...	...	...	...	a_w
Slot i-1	...	...	...	...	...	...	...	...
	...	...	...	...	...	...	...	...
Slot i	**1**							
	0							
	0							
	0							
	0							
Slot i+1	**1**	...	...	...	...	...	...	...
	0	...	...	...	...	...	...	...
	...	...	...	...	...	...	...	...

Each opcode in the trace is assigned to a slot in the EVM circuit

Trace

...	...
i-1:	PUSH
i :	ADD
i+1:	MUL
...	...

EVM Circuit

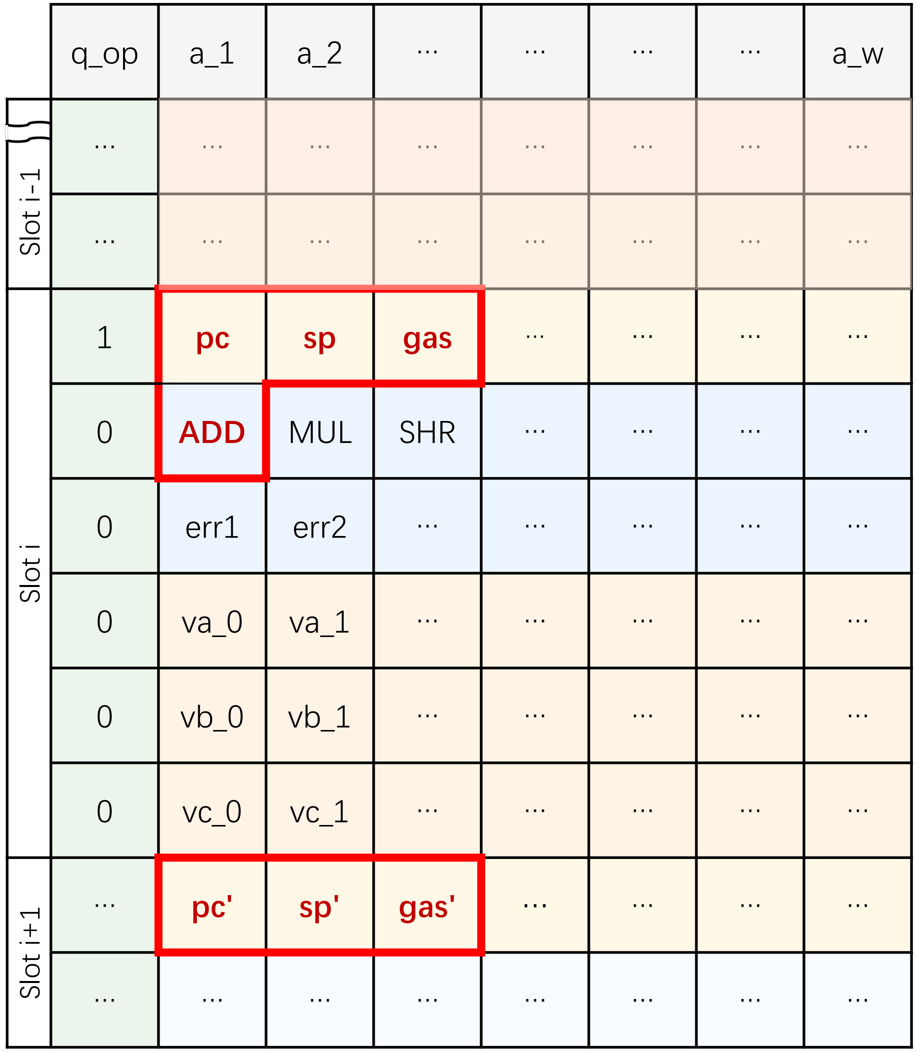
	q_op	a_1	a_2	…	…	…	…	a_w
Slot i-1	…	…	…	…	…	…	…	…
	…	…	…	…	…	…	…	…
Slot i	1	pc	sp	gas	…	…	…	…
	0	ADD	MUL	SHR	…	…	…	…
	0	err1	err2	…	…	…	…	…
	0	v_0	v_1	…	…	…	…	…
	0	…	…	…	…	…	…	v_n
Slot i+1	1	…	…	…	…	…	…	…
	0	…	…	…	…	…	…	…
	…	…	…	…	…	…	…	…

Split a slot to context, case switch, and operating values

context	Store the context of state machine
op and case switch	Switch the state between opcodes and error cases
Operating values	Operating values specific to each opcode

EVM Circuit

Scroll

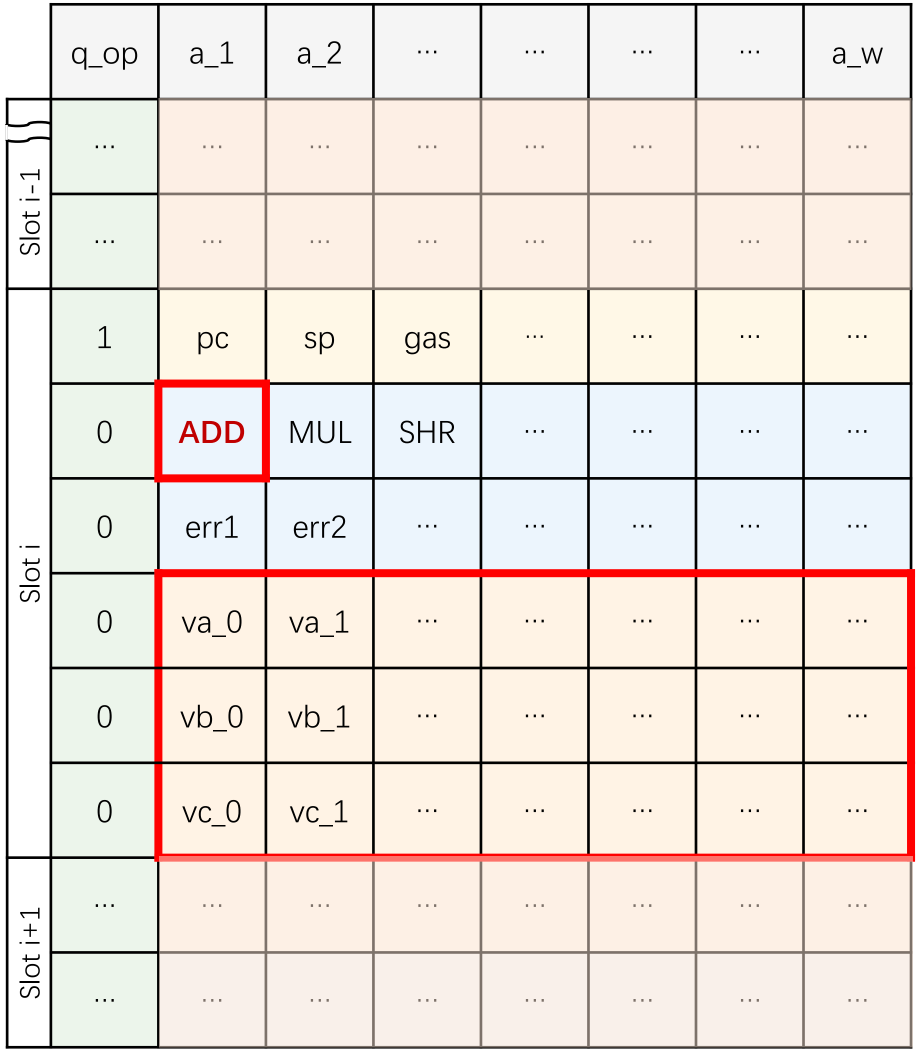


	q_op	a_1	a_2	…	…	…	…	a_w
Slot i-1	…	…	…	…	…	…	…	…
	…	…	…	…	…	…	…	…
Slot i	1	**pc**	**sp**	**gas**	…	…	…	…
	0	**ADD**	MUL	SHR	…	…	…	…
	0	err1	err2	…	…	…	…	…
	0	va_0	va_1	…	…	…	…	…
	0	vb_0	vb_1	…	…	…	…	…
	0	vc_0	vc_1	…	…	…	…	…
Slot i+1	…	**pc'**	**sp'**	**gas'**	…	…	…	…
	…	…	…	…	…	…	…	…

Constrain the context transition

```
if case == ADD:
  pc’ = pc + 1
  sp’ = sp – 1
  gas’= gas + 3
  ...
```

EVM Circuit

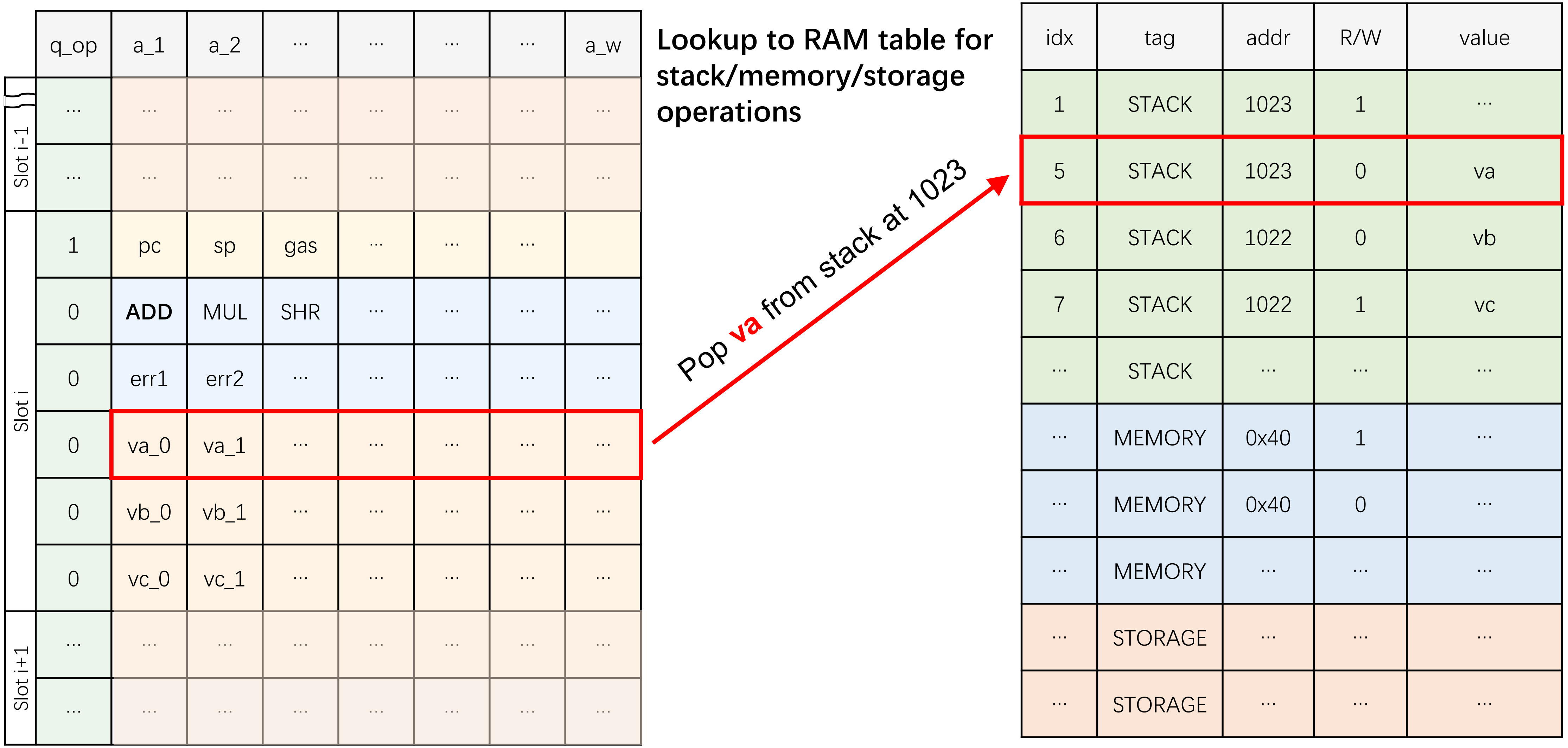


	q_op	a_1	a_2	…	…	…	…	a_w
Slot i-1	…	…	…	…	…	…	…	…
	…	…	…	…	…	…	…	…
Slot i	1	pc	sp	gas	…	…	…	…
	0	**ADD**	MUL	SHR	…	…	…	…
	0	err1	err2	…	…	…	…	…
	0	va_0	va_1	…	…	…	…	…
	0	vb_0	vb_1	…	…	…	…	…
	0	vc_0	vc_1	…	…	…	…	…
Slot i+1	…	…	…	…	…	…	…	…
	…	…	…	…	…	…	…	…

Write custom constraints for each opcode

```
if case == ADD:
  vc[0] + carry[0]*256 == va[0] + vb[0]
  vc[1] + carry[1]*256 == va[1] + vb[1] +
carry[0]
  ...
  vc[31] + carry[31]*256 == va[31] + vb[31] +
carry[30]
```

EVM Circuit



	q_op	a_1	a_2	…	…	…	…	a_w
Slot i-1	…	…	…	…	…	…	…	…
	…	…	…	…	…	…	…	…
Slot i	1	pc	sp	gas	…	…	…	
	0	**ADD**	MUL	SHR	…	…	…	…
	0	err1	err2	…	…	…	…	…
	0	va_0	va_1	…	…	…	…	…
	0	vb_0	vb_1	…	…	…	…	…
	0	vc_0	vc_1	…	…	…	…	…
Slot i+1	…	…	…	…	…	…	…	…
	…	…	…	…	…	…	…	…

Lookup to RAM table for stack/memory/storage operations

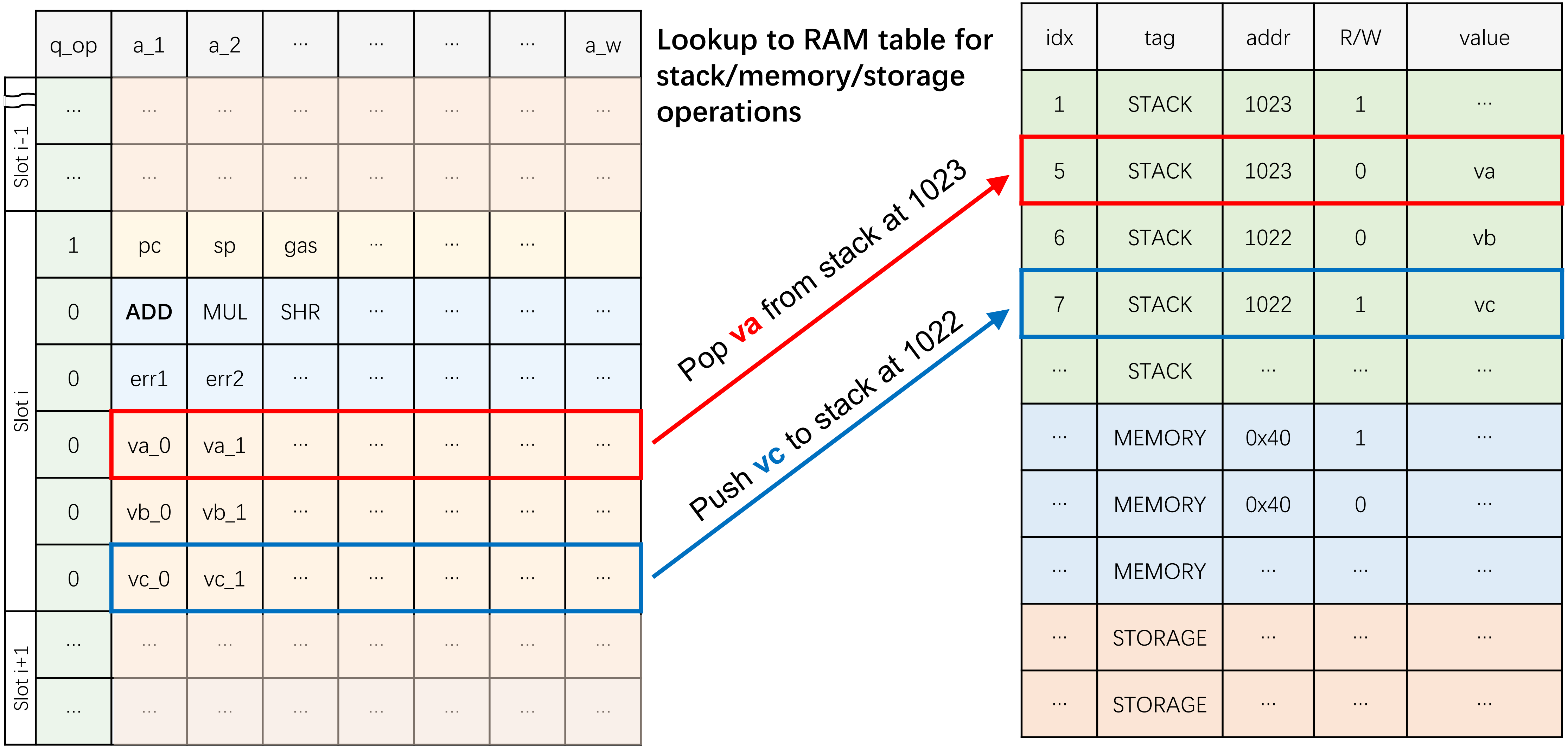
Pop **va** from stack at 1023

RAM table

idx	tag	addr	R/W	value
1	STACK	1023	1	…
5	STACK	1023	0	va
6	STACK	1022	0	vb
7	STACK	1022	1	vc
…	STACK	…	…	…
…	MEMORY	0x40	1	…
…	MEMORY	0x40	0	…
…	MEMORY	…	…	…
…	STORAGE	…	…	…
…	STORAGE	…	…	…

EVM Circuit

Scroll



Lookup to RAM table for stack/memory/storage operations

	q_op	a_1	a_2	…	…	…	…	a_w
Slot i-1	…	…	…	…	…	…	…	…
	…	…	…	…	…	…	…	…
Slot i	1	pc	sp	gas	…	…	…	
	0	**ADD**	MUL	SHR	…	…	…	…
	0	err1	err2	…	…	…	…	…
	0	va_0	va_1	…	…	…	…	…
	0	vb_0	vb_1	…	…	…	…	…
	0	vc_0	vc_1	…	…	…	…	…
Slot i+1	…	…	…	…	…	…	…	…
	…	…	…	…	…	…	…	…

RAM table

idx	tag	addr	R/W	value
1	STACK	1023	1	…
5	STACK	1023	0	va
6	STACK	1022	0	vb
7	STACK	1022	1	vc
…	STACK	…	…	…
…	MEMORY	0x40	1	…
…	MEMORY	0x40	0	…
…	MEMORY	…	…	…
…	STORAGE	…	…	…
…	STORAGE	…	…	…

More features in zkEVM circuits

Scroll

- Handle dynamic opcode (e.g., CALLDATACOPY, etc.)
- Handle error cases (e.g., out-of-gas error)
- Handle calls into other contracts
- Support EIPs such as warm storage access list, tx refund, etc.

Credit to all community members!

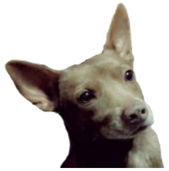
Scroll

CPerezz

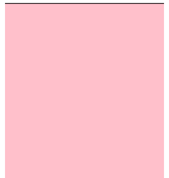
Scroll-dev

Chih Cheng Liang

Eduard S.


Han

Brecht Devos


ying tong

Zhang Zhuo

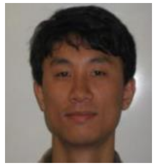
HAOYUatHZ

Rohit Narurkar

NoCtrlZ

icemelon

DreamWuGit


xgaozoyoe

BarryWhiteHat

AronisAt79

z2trillion

kilic

Ho

genfengDog

silathdiir

Miha Stopar

Lawliet-Chan

pinkiebell

adria0

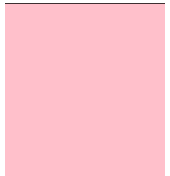
bchyl

spartucus

davidnevadoc

TrapdoorHeader

str4d


ying tong

ebfull

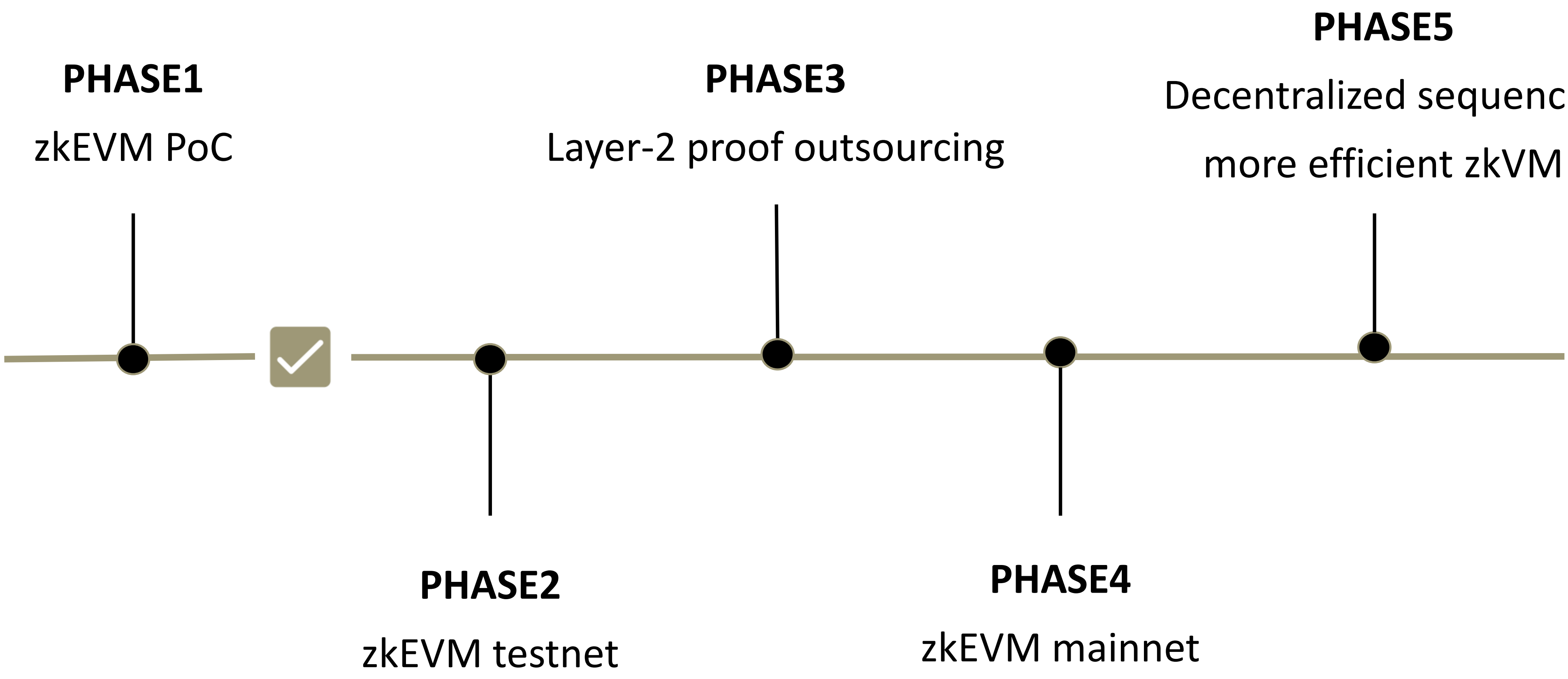
Daira Hopwood

Kris Nuttycombe

...

Roadmap

Scroll



PHASE1

zkEVM PoC

✓

PHASE2

zkEVM testnet

PHASE3

Layer-2 proof outsourcing

PHASE4

zkEVM mainnet

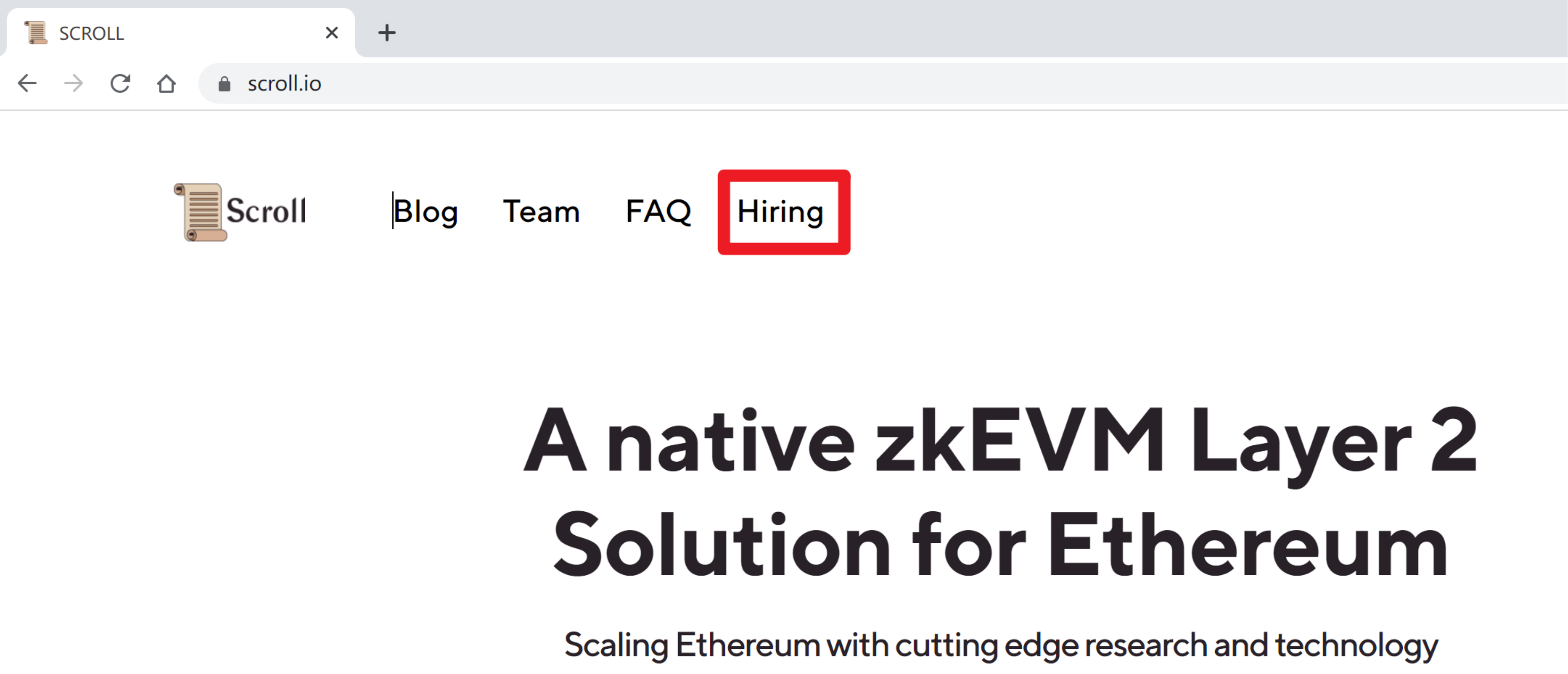
PHASE5

Decentralized sequencer

more efficient zkVM

We are hiring! Check out scroll.io

Scroll



SCROLL

scroll.io

Scroll

Blog Team FAQ Hiring

A native zkEVM Layer 2 Solution for Ethereum

Scaling Ethereum with cutting edge research and technology