State Minimized L2s and Why Ethereum > EVM

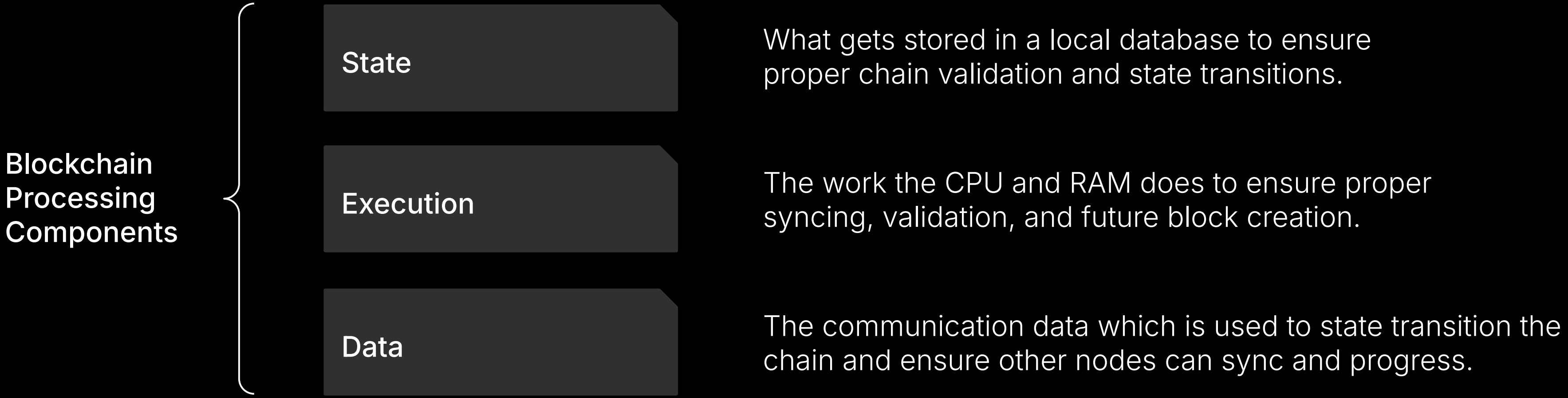
STATE GROWTH

Agenda

1. Components of a blockchain
2. Final boss: state growth
3. Solution
4. Fuel’s state philosophy
5. Closing

FUEL

Components of a blockchain?



Blockchain Processing Components

State

What gets stored in a local database to ensure proper chain validation and state transitions.

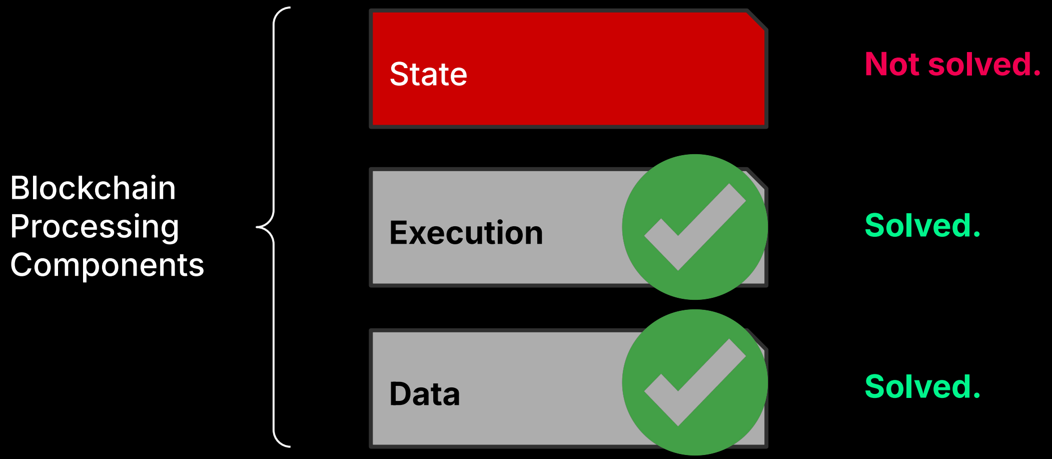
Execution

The work the CPU and RAM does to ensure proper syncing, validation, and future block creation.

Data

The communication data which is used to state transition the chain and ensure other nodes can sync and progress.

FUEL



Blockchain Processing Components

State — Not solved.

Execution ✓ — Solved.

Data ✓ — Solved.

FUEL

Execution:

- Not rollups: although they enable us to use other execution models
- **Parallel transaction execution**
  - Processing transactions in parallel using all cores of your CPU
- **More efficient virtual machine designs**
  - Stylus
  - SVM
  - FuelVM
- **Better pre-compiles**

Data:

- EIP-4844 → configuration setting changes → PeerDAS on the way
- Sharding designs
- External data-availability layers

FUEL

What about state?

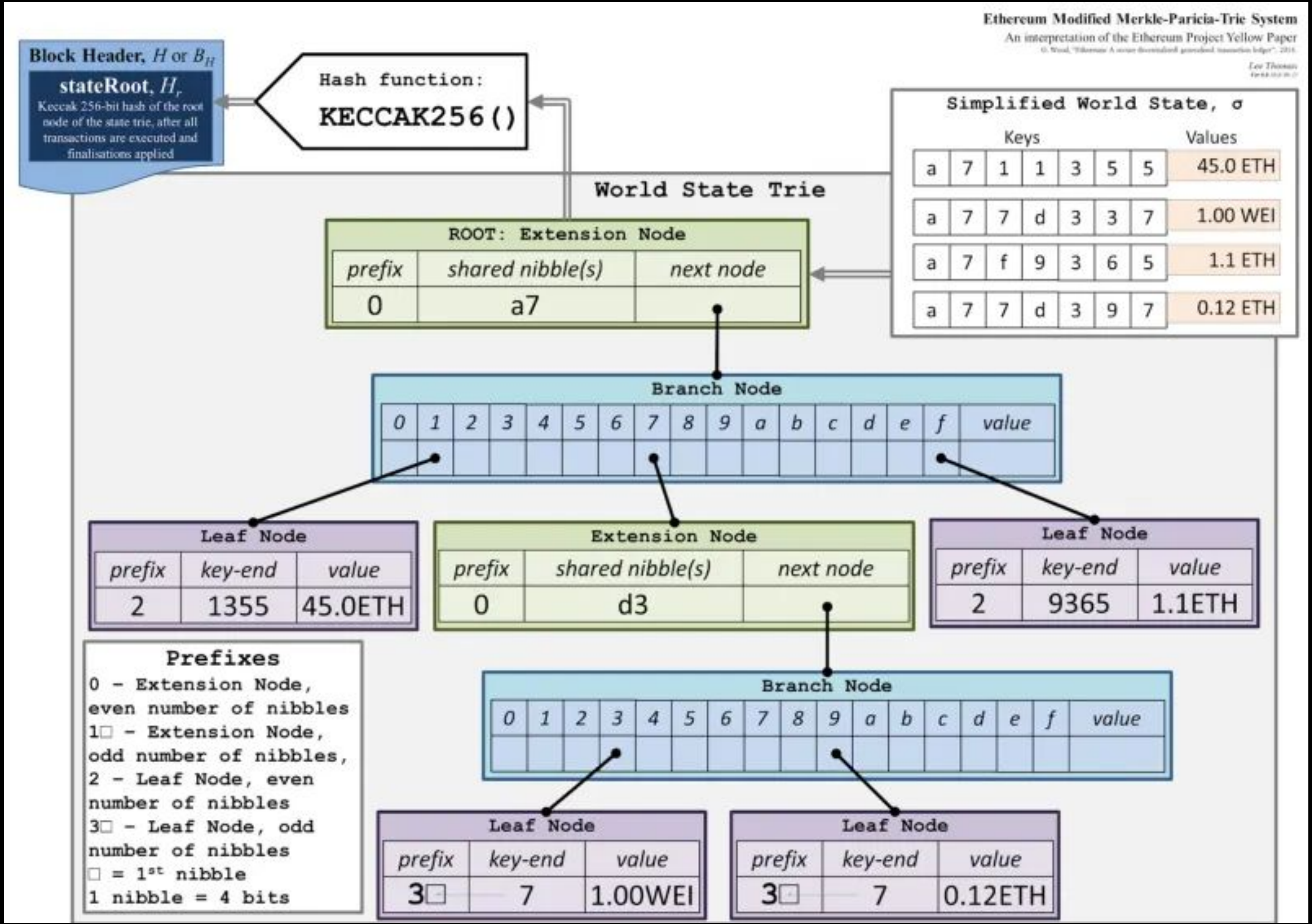
Comparing State Design

Bitcoin State

- UTXO Set

Ethereum State

- Account balances (ETH)
- Smart contract code
- Smart contract state
  - Token balances
  - Token approvals
  - All other data



Ethereum Modified Merkle-Paricia-Trie System

An interpretation of the Ethereum Project Yellow Paper

G. Wood, "Ethereum: A secure decentralised generalised transaction ledger", 2014.

Lee Thomas

Block Header, H or B_H

stateRoot, H_r

Keccak 256-bit hash of the root node of the state trie, after all transactions are executed and finalisations applied

Hash function:

KECCAK256()

Simplified World State, σ

Keys							Values
a	7	1	1	3	5	5	45.0 ETH
a	7	7	d	3	3	7	1.00 WEI
a	7	f	9	3	6	5	1.1 ETH
a	7	7	d	3	9	7	0.12 ETH

World State Trie

ROOT: Extension Node

prefix	shared nibble(s)	next node
0	a7	

Branch Node

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	value
	●						●								●	

Leaf Node

prefix	key-end	value
2	1355	45.0ETH

Extension Node

prefix	shared nibble(s)	next node
0	d3	

Leaf Node

prefix	key-end	value
2	9365	1.1ETH

Branch Node

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	value
			●						●							

Leaf Node

prefix	key-end	value
3□	7	1.00WEI

Leaf Node

prefix	key-end	value
3□	7	0.12ETH

Prefixes

0 – Extension Node, even number of nibbles

1□ – Extension Node, odd number of nibbles,

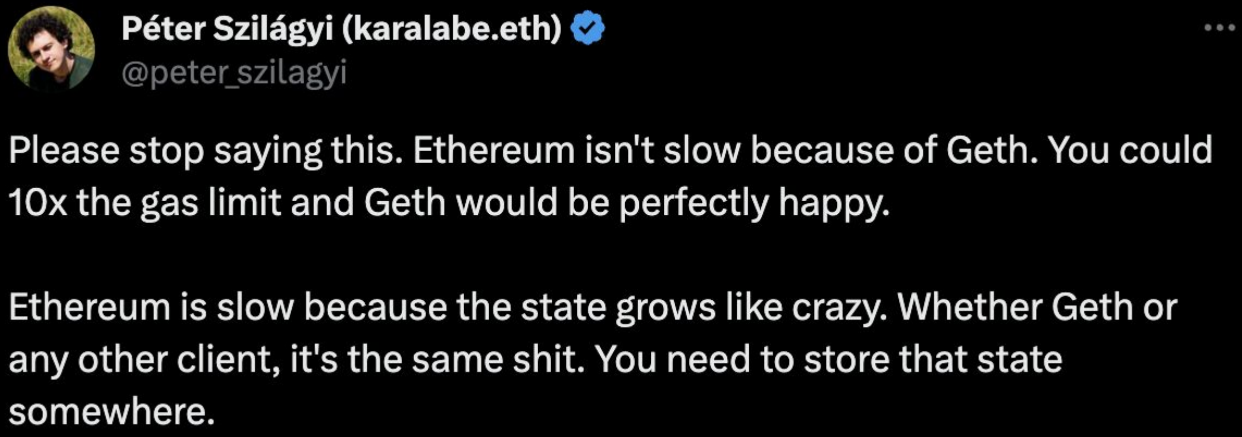
2 – Leaf Node, even number of nibbles

3□ – Leaf Node, odd number of nibbles

□ = 1st nibble

1 nibble = 4 bits

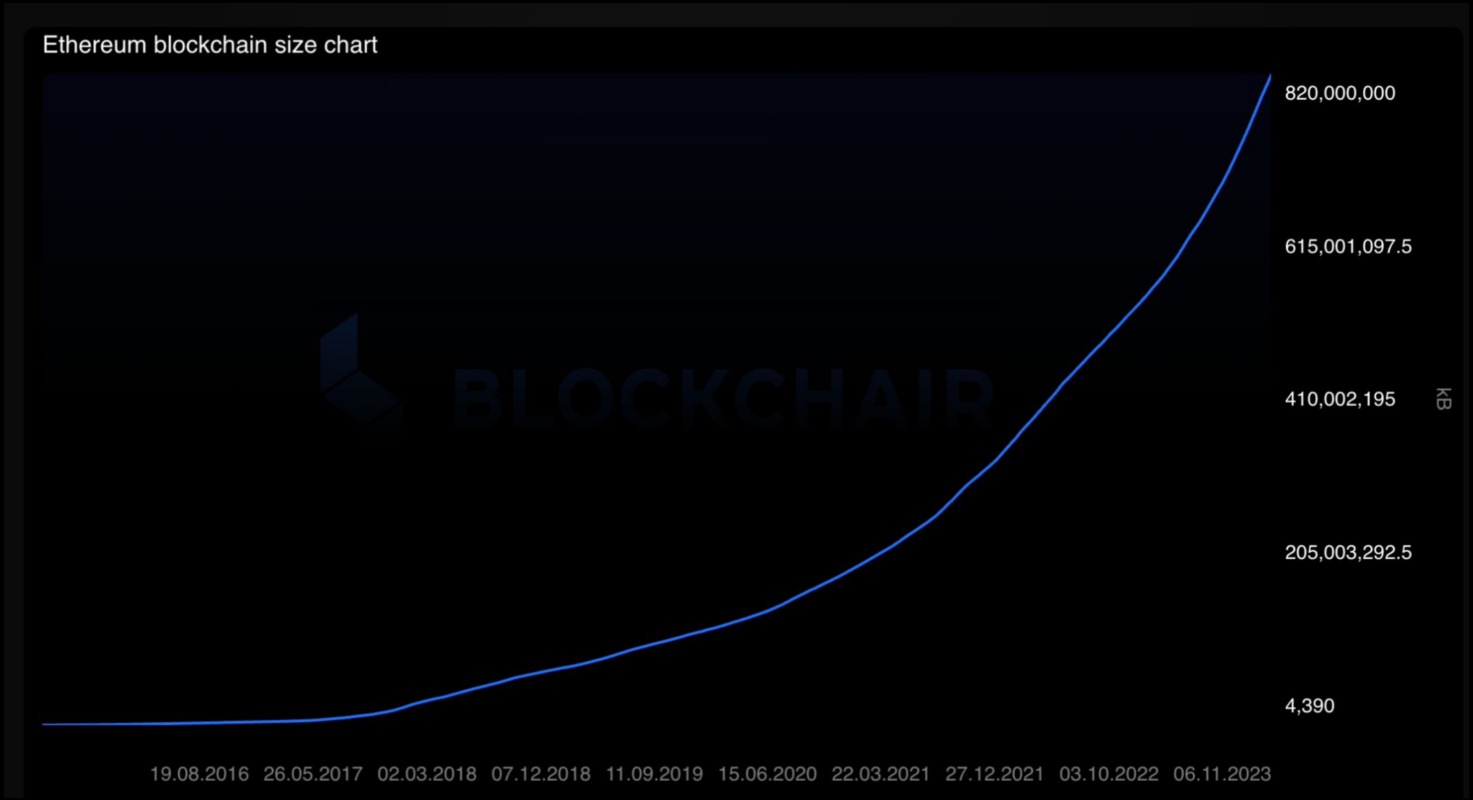
Ethereum clients to blame for scalability limits?



Péter Szilágyi (karalabe.eth)
@peter_szilagyi

Please stop saying this. Ethereum isn't slow because of Geth. You could 10x the gas limit and Geth would be perfectly happy.

Ethereum is slow because the state grows like crazy. Whether Geth or any other client, it's the same shit. You need to store that state somewhere.



Ethereum blockchain size chart

How can we address state growth?

FUEL

State growth options:

1. State rent
2. “Statelessness”
3. Un-merklaize the state
4. App level compression
5. Just “let the state grow”
6. Verkle Trees
7. ... bandwidth

Approach 1: State Rent

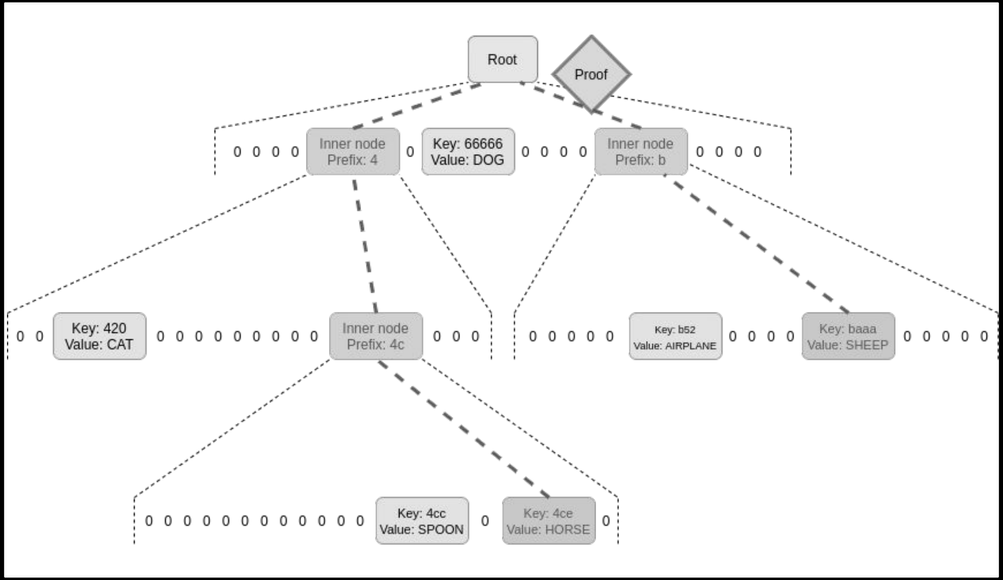
Charge “rent” to store state

- Keep the same UX/DevEx for most accounts
- No state is “lost”, it just needs to be “restored”

However...

- Tree rot can occur, among other issues

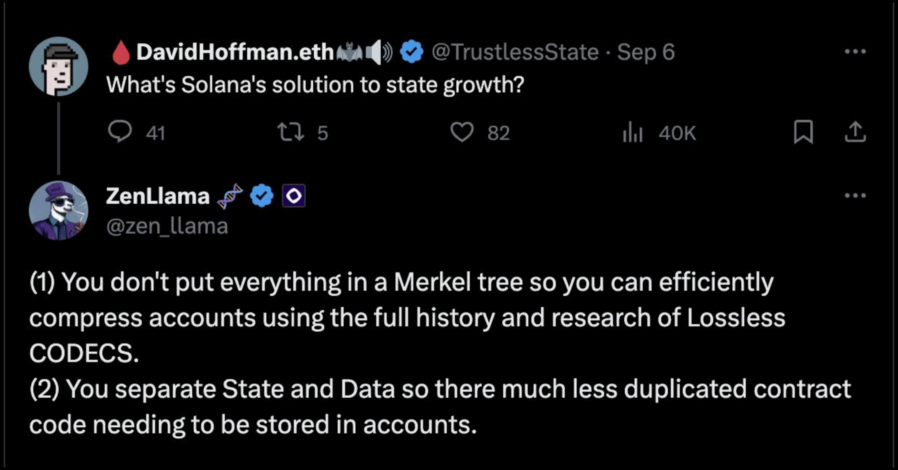
Approach 2: Statelessness



Vitalik.ca - Verkle Trees 2021

- Full nodes don’t need to store state.
- State proofs are included with transactions & blocks.

Approach 3: Un-Merkelize the State



DavidHoffman.eth @TrustlessState · Sep 6

What's Solana's solution to state growth?

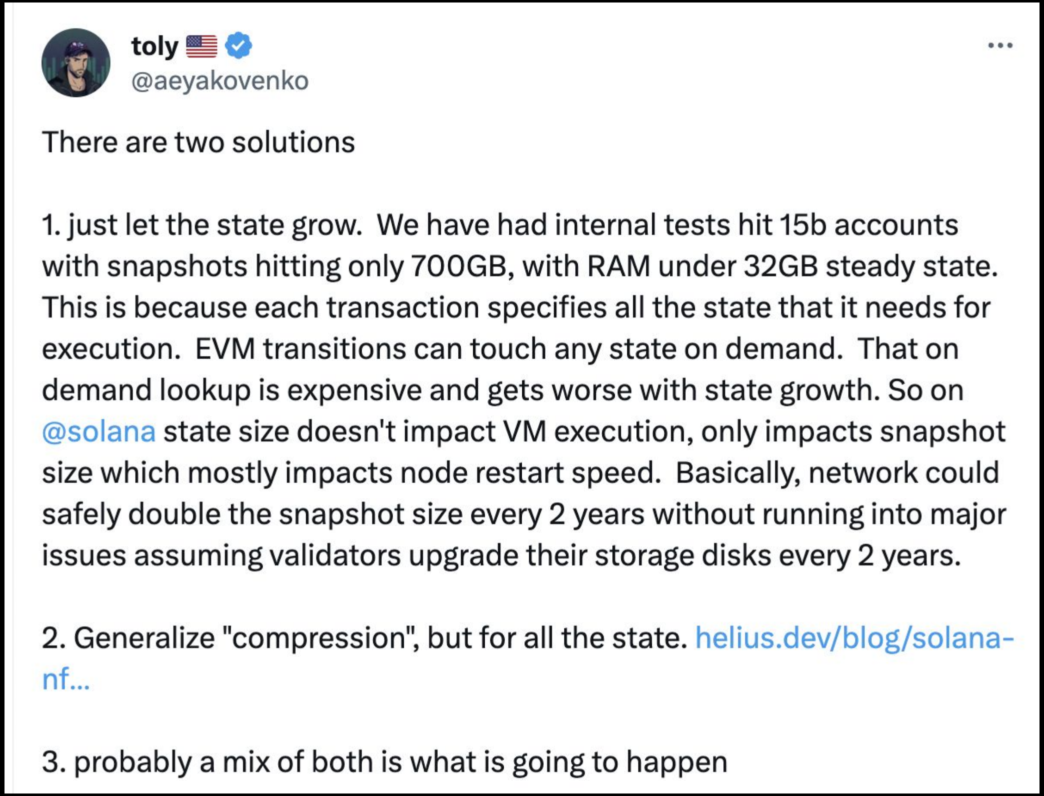
ZenLlama
@zen_llama

(1) You don't put everything in a Merkel tree so you can efficiently compress accounts using the full history and research of Lossless CODECS.
(2) You separate State and Data so there much less duplicated contract code needing to be stored in accounts.

Approach 4: Application-level State Compression

- Use techniques to leverage calldata over state
- Surface merkle proofs to prove ownership (Airdrops)
- Compressed application state

Approach 5: Let it Grow



toly 🇺🇸 ✓
@aeyakovenko

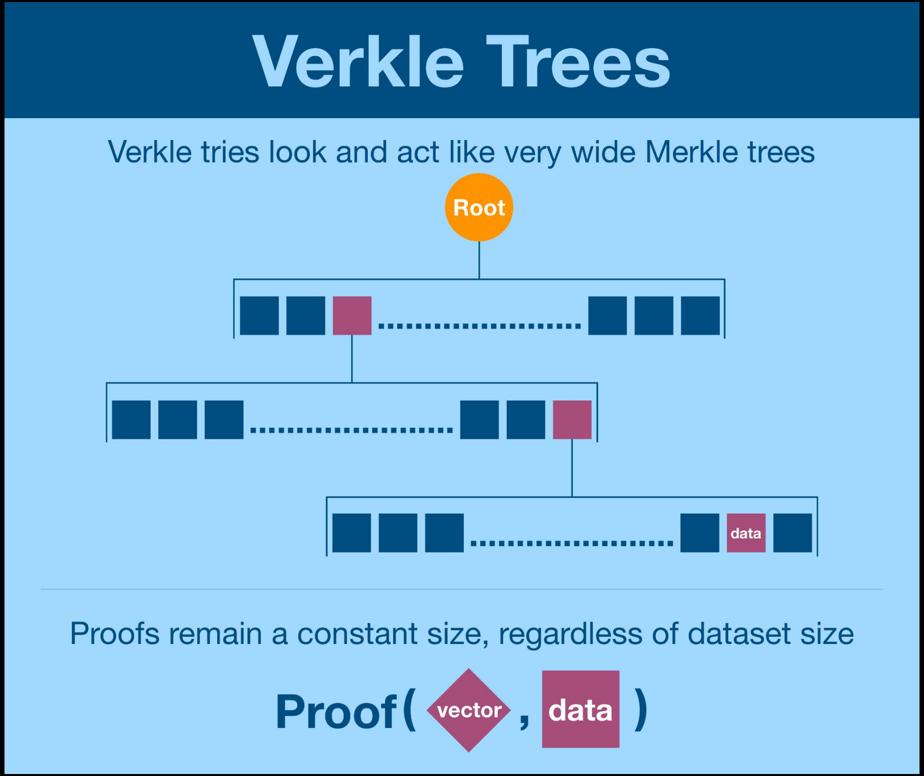
There are two solutions

1. just let the state grow. We have had internal tests hit 15b accounts with snapshots hitting only 700GB, with RAM under 32GB steady state. This is because each transaction specifies all the state that it needs for execution. EVM transitions can touch any state on demand. That on demand lookup is expensive and gets worse with state growth. So on @solana state size doesn't impact VM execution, only impacts snapshot size which mostly impacts node restart speed. Basically, network could safely double the snapshot size every 2 years without running into major issues assuming validators upgrade their storage disks every 2 years.

2. Generalize "compression", but for all the state. helius.dev/blog/solana-nf...

3. probably a mix of both is what is going to happen

Approach 6:



Verkle Trees

Verkle tries look and act like very wide Merkle trees

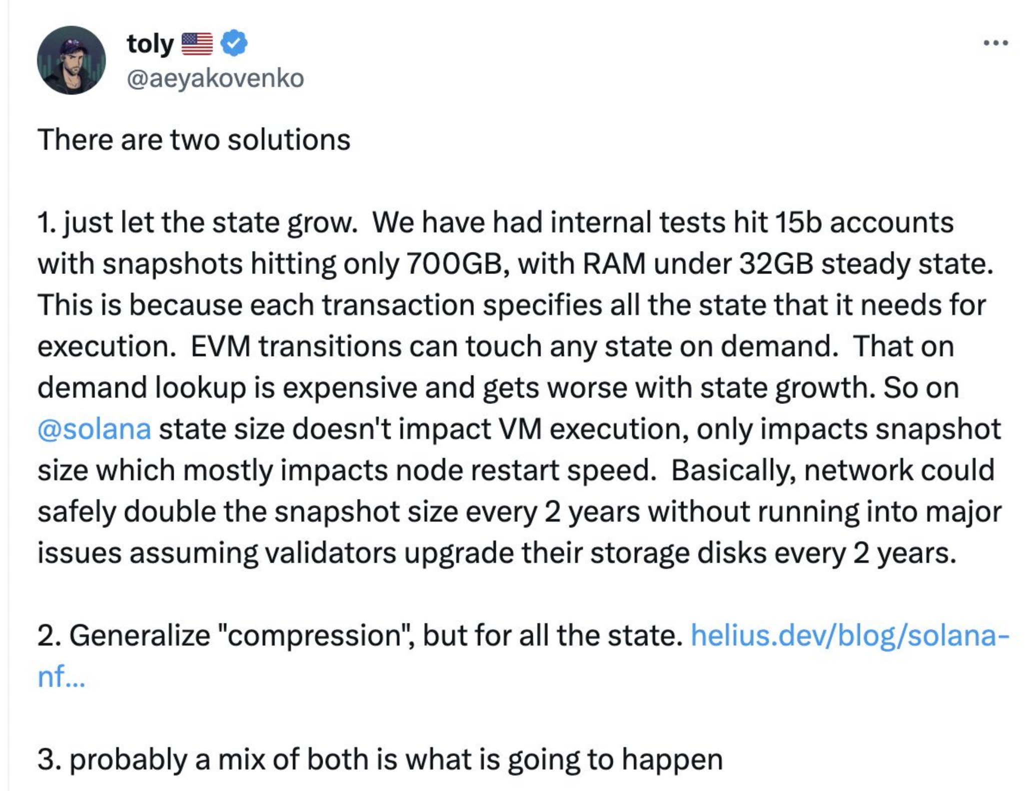
Proofs remain a constant size, regardless of dataset size

Proof(vector , data)

inevitableeth.com

How about AltVMs?

SVM?



toly 🇺🇸
@aeyakovenko

There are two solutions

1. just let the state grow. We have had internal tests hit 15b accounts with snapshots hitting only 700GB, with RAM under 32GB steady state. This is because each transaction specifies all the state that it needs for execution. EVM transitions can touch any state on demand. That on demand lookup is expensive and gets worse with state growth. So on @solana state size doesn't impact VM execution, only impacts snapshot size which mostly impacts node restart speed. Basically, network could safely double the snapshot size every 2 years without running into major issues assuming validators upgrade their storage disks every 2 years.

2. Generalize "compression", but for all the state. helius.dev/blog/solana-nf...

3. probably a mix of both is what is going to happen

Fuel State Philosophy

FUEL

Fuel State Philosophy: freebies

- No global state tree, only local state trees for each smart contract
- Native assets don’t need to be merkalized
- No useless approval state (from approve + transferFrom)

... while retaining **rich cryptographic light clients and verifiability** due to the UTXO model.

FUEL

Fuel State Philosophy:

Use more **bandwidth + execution** less **state**

FUEL

But how? Native state rehydration.

Conventional approach: contract state lookups (“use contracts for everything”)

New approach:

- Store root hashes / state changes only
- Present data over bandwidth to “rehydrate” state
- Provide many different tools for the developer to leverage this (scripts, native account abstraction etc.)

FUEL

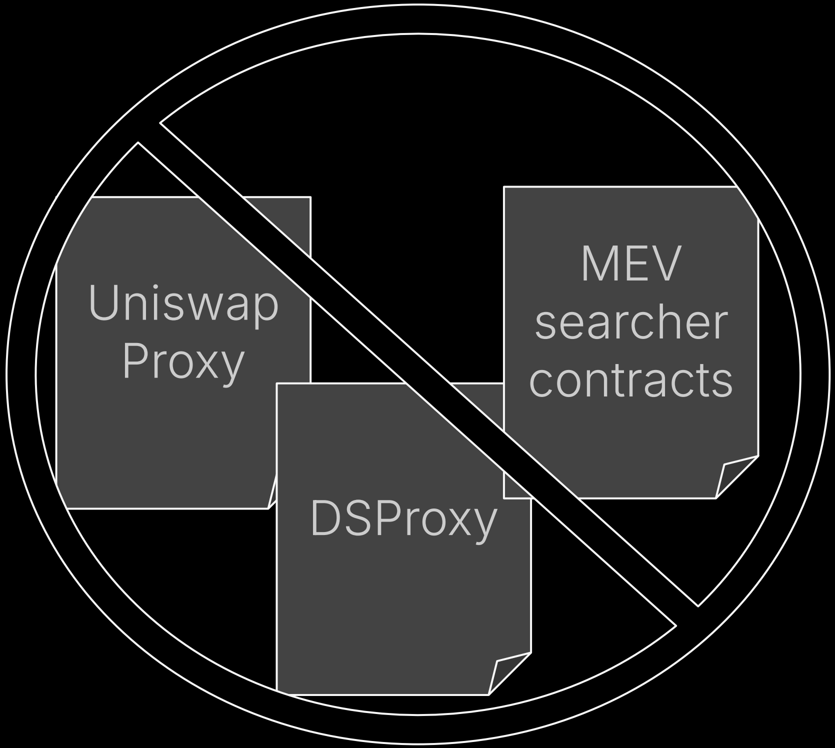
Fuel State Philosophy: State minimized mechanisms

We give the developer many ways to do things other than just smart contract storage

- **Scripts**
- **Predicates: lightweight, stateless contracts**
- **Native assets**
- **Flexible transaction model**

Scripts

Ephemeral logic is included in transactions, not stored in state



Native assets

All asset transfers only touch a single state element

Native assets can be used to represent non-value state (ex, and NFT to represent ownership)

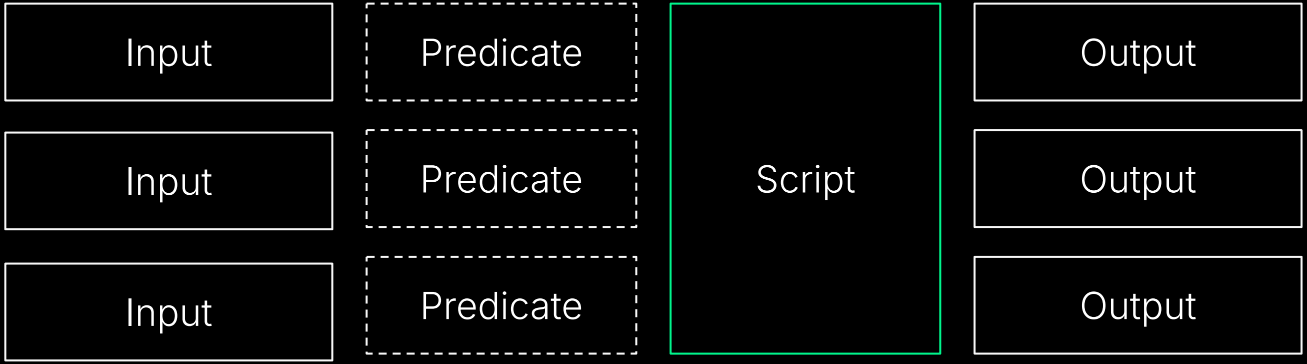
Predicates

Lightweight, **stateless code** for permissioning transfers & transactions

- Native account abstraction
- Stateless DeFi primitives (order books, etc)

New Transaction Model

More options to form multi-party complex transactions



FUEL

Ethereum > EVM