Rethinking Ethereum's Account Model

A forgotten mechanism to scale

Will Villanueva

Section 1

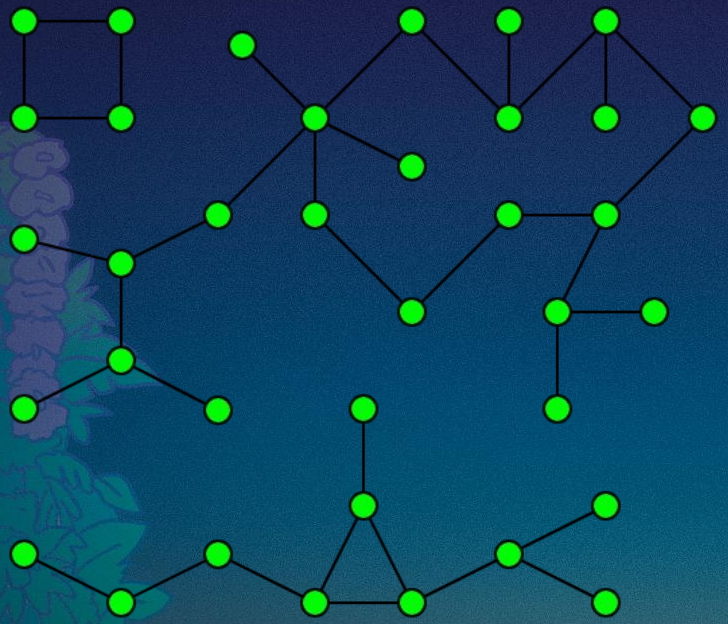
Deterministic State Access

Access Lists or Account List

- Knowing the state an account can touch ahead of time allow for concurrency
- Enforcement by the runtime, prevents any overrides
- If two swap transactions happen through the same protocol, but the same token, this can happen concurrently
    - It's a mechanism for multiplexing and spawning parallel processors
    - I/O is known ahead of time
- ERC20, is a one massive global state
- It puts a lock on this global state any time it is being touched

A form of sharding

- All pending transactions form a graph of nodes on their state dependencies
- Each isolated group of graphs can now be run completely concurrently
- Complimentary to multiple concurrent proposals and protocols on how to split the execution



Isolated state access, parallel processing

Section 2

Security

Approve Pattern on ERC20's is a disaster

- We all know it
- Wallets as a service
- Fingerprints ahead of time on what a transaction **can** do from a security perspective
- Who knows what contract could programmatically spend your approval
- In an account centric model, you have many deterministic addresses, generated from a seed/program that only gives access to that program

No way to know who has read/write access ahead of time

- No way to know that accounts may touch or alter state
- No form of isolation
- Reduced heuristics
- Memory/programmatic ownership patterns, hashes or clean and clear

Account Centric Models are More Secure

And they’re more fast, especially at the pre-processing level


Thank you!
Will Villanueva
william.j.villanueva@gmail.com
@wjvill