Catching up on the Internet Computer

May 2022

Yvonne-Anne Pignolet
Senior Research Manager, DFINITY Foundation

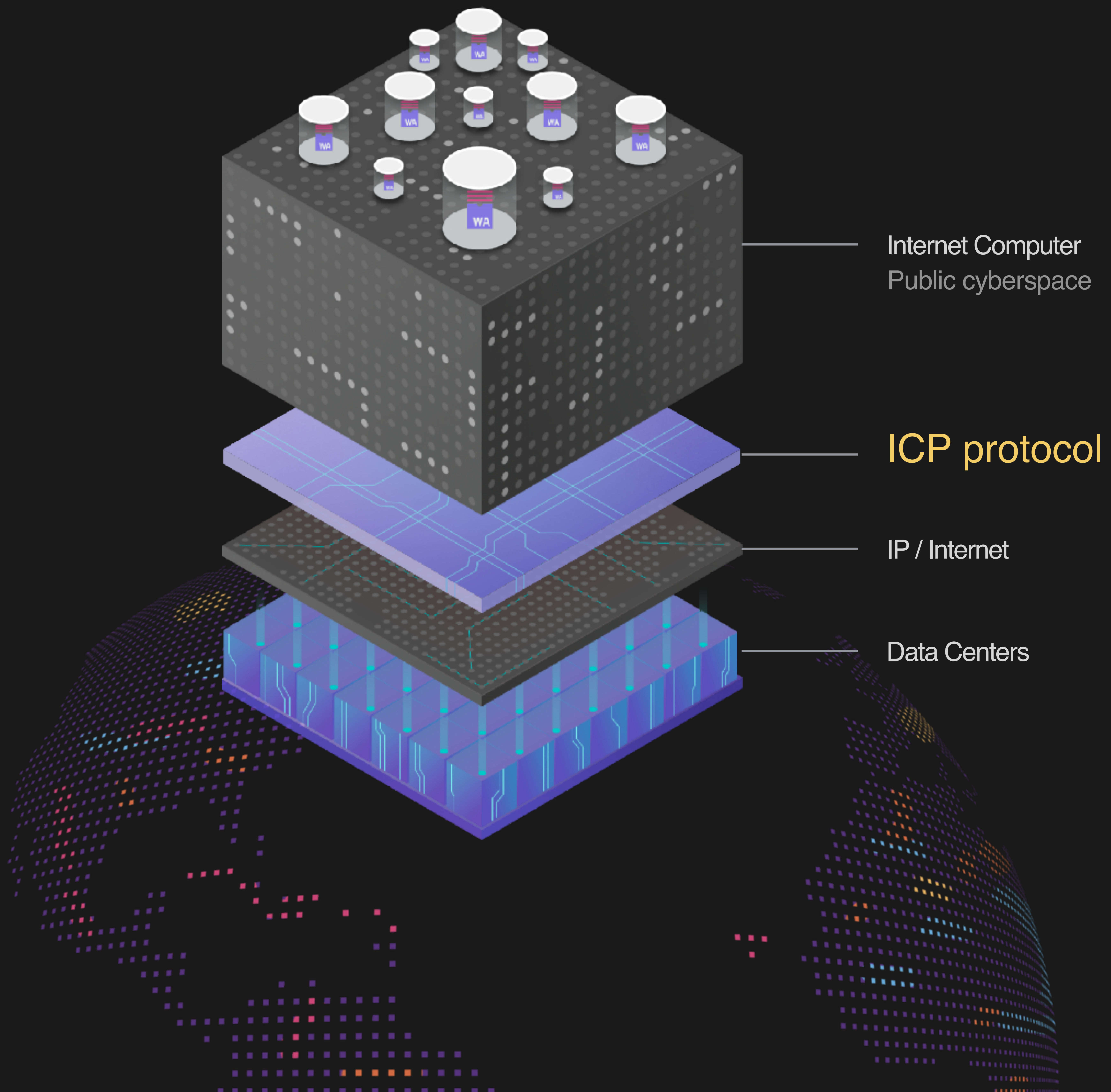
What is the Internet Computer

Platform to run **any computation** using blockchain technology for decentralisation and security

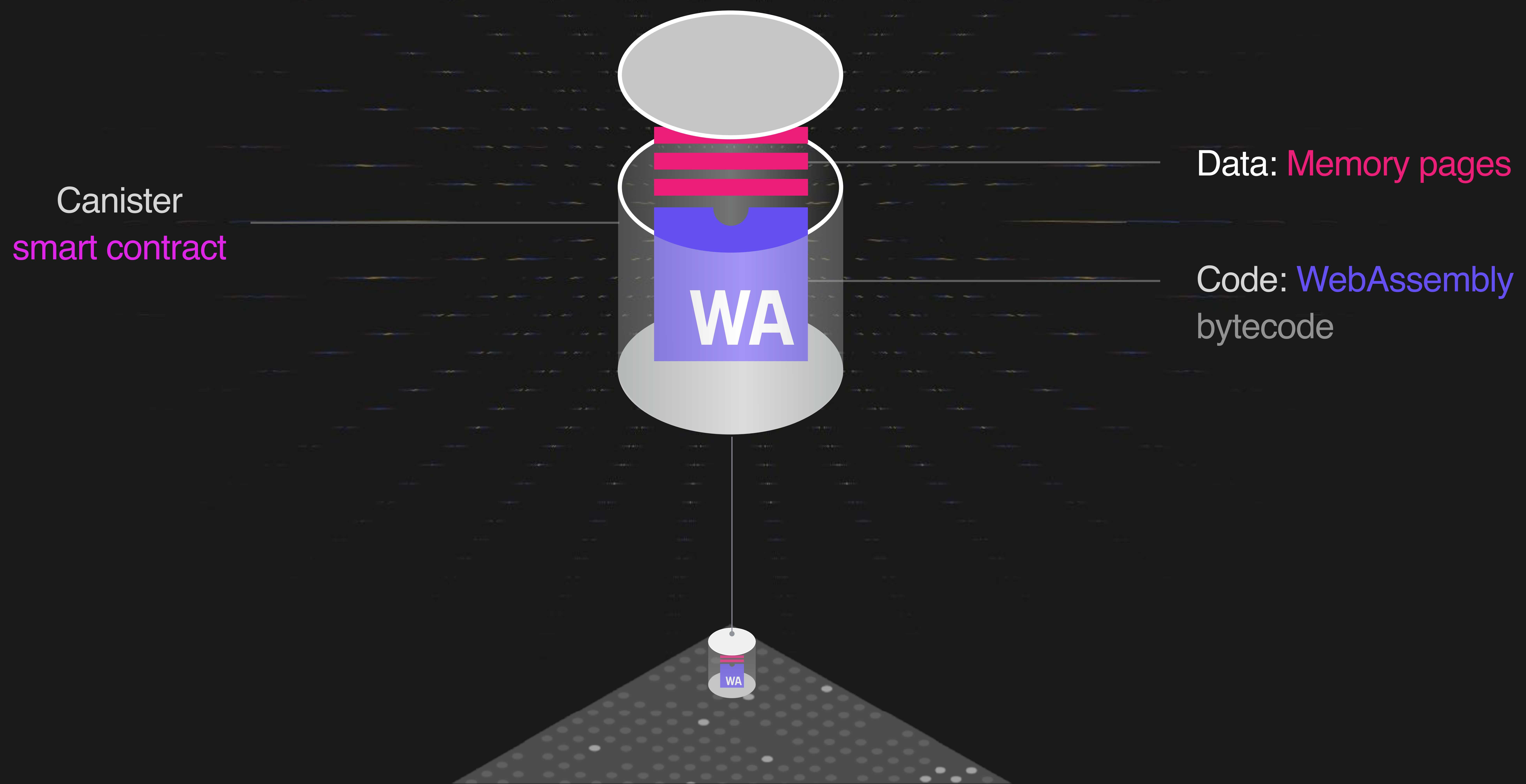
ICP | Internet Computer Protocol

Coordination of independent datacenters, jointly performing any computation for anyone

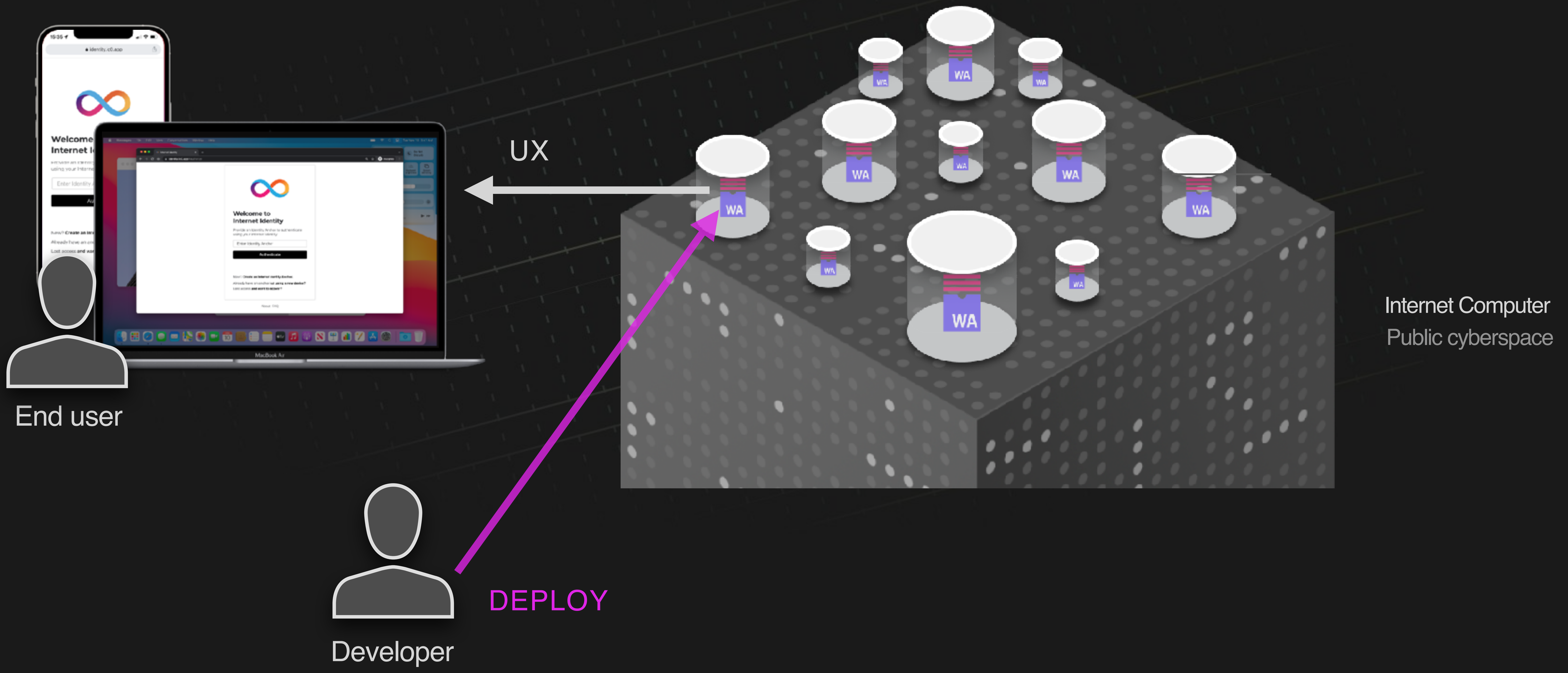
- Create Internet Computer blockchains.
- Ensures machines agree on sequence of computations carried out



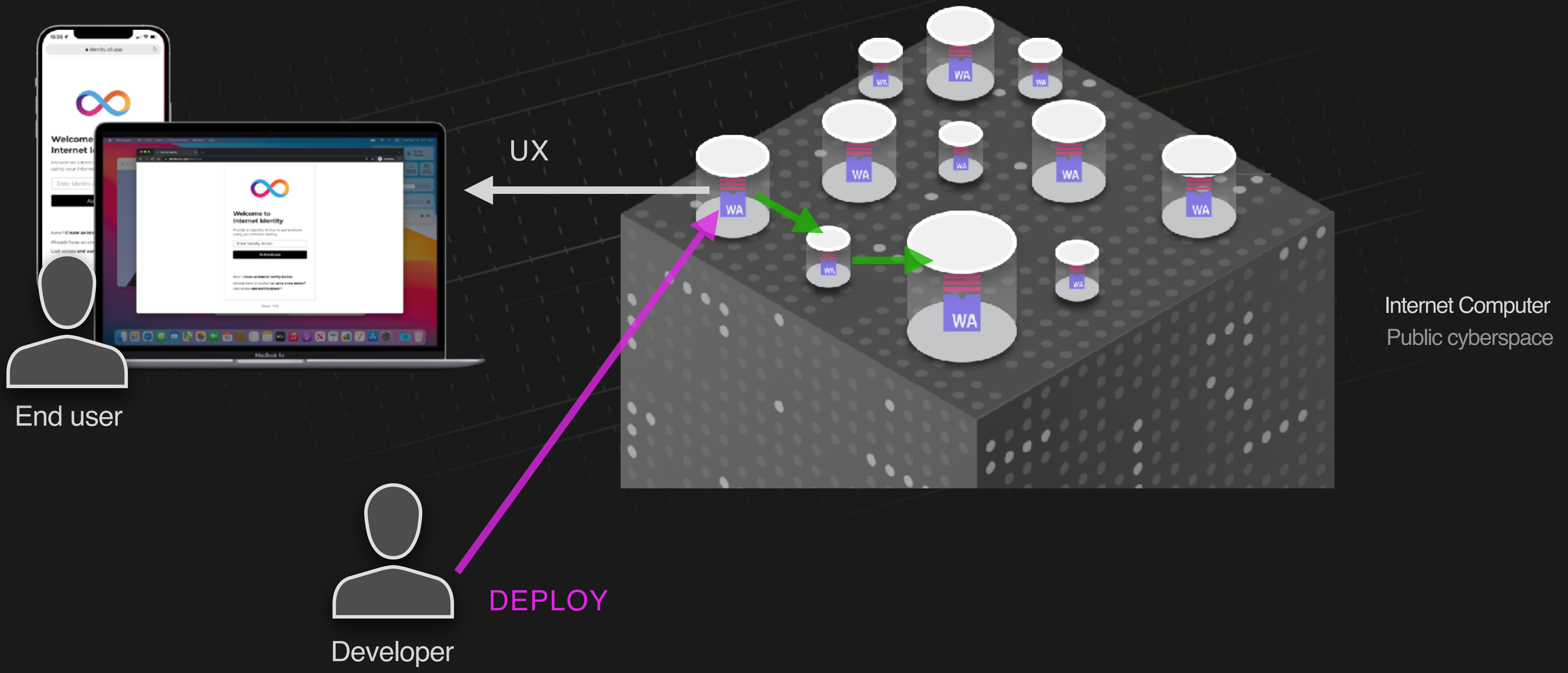
Canister smart contracts are fast, run in parallel, and scale...



Developers build dapps by uploading canisters to the IC.
No cloud computing necessary



Developers build dapps by uploading canisters to the IC.
No cloud computing necessary



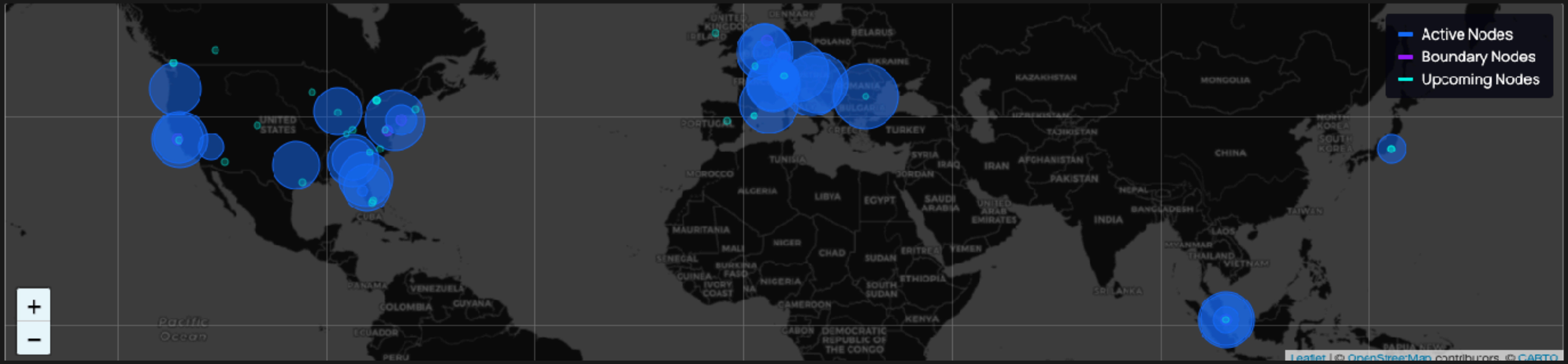
Launched May 2021. Growing more powerful daily…

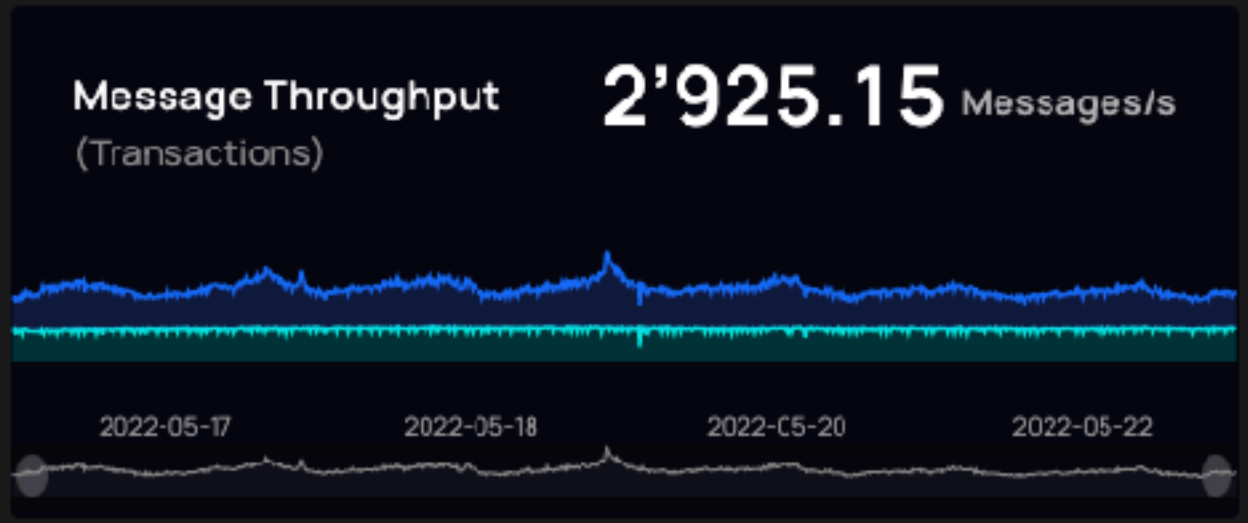
Blocks
842’849’251

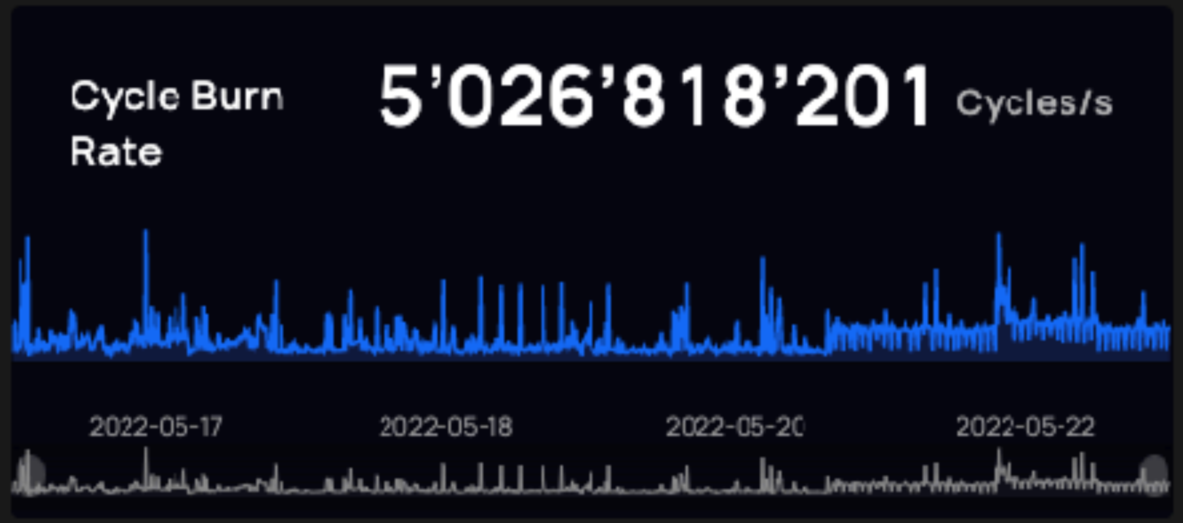
Flow
35.70 Blocks/s

Cycle Burn Rate
4’933’218’936 Cycles/s

NNS Community Fund
1’283’707 ICP







https://dashboard.internetcomputer.org/

Fast growing blockchain ecosystem

Over 1,000 developers now building

Comparison with other Blockchain Systems

coincodex

Layer-1 Performance Comparison

	Ethereum	Cardano	Solana	Avalanche	Algorand	Internet Computer
Transaction Speed	15-20 TPS	2 TPS	2,000-3,000 TPS	4,500 TPS	20 TPS	11,500 TPS 250,000 QPS
Transaction Finality	14 minutes	10-60 minutes	21-46 seconds	2-3 seconds	4-5 seconds	1 second
Scalability	Not very scalable	Not very scalable	Not very scalable	Not very scalable	More scalability	Indefinite scalability
Node Count	6,000 nodes	3,173 nodes	1,603 nodes	1,243 nodes	1,997 nodes	443 nodes
Storage Costs	$73,000,000 / GB	Inadequate data storage	$1,000,000 / GB	$988,000 / GB	IPFS off-chain storage	$5 / GB
Cloud Service Dependency	70% of nodes run on AWS	Unclear how many are cloud	Most nodes run on cloud	Unclear how many are cloud	Most nodes run on cloud	Independent data centers

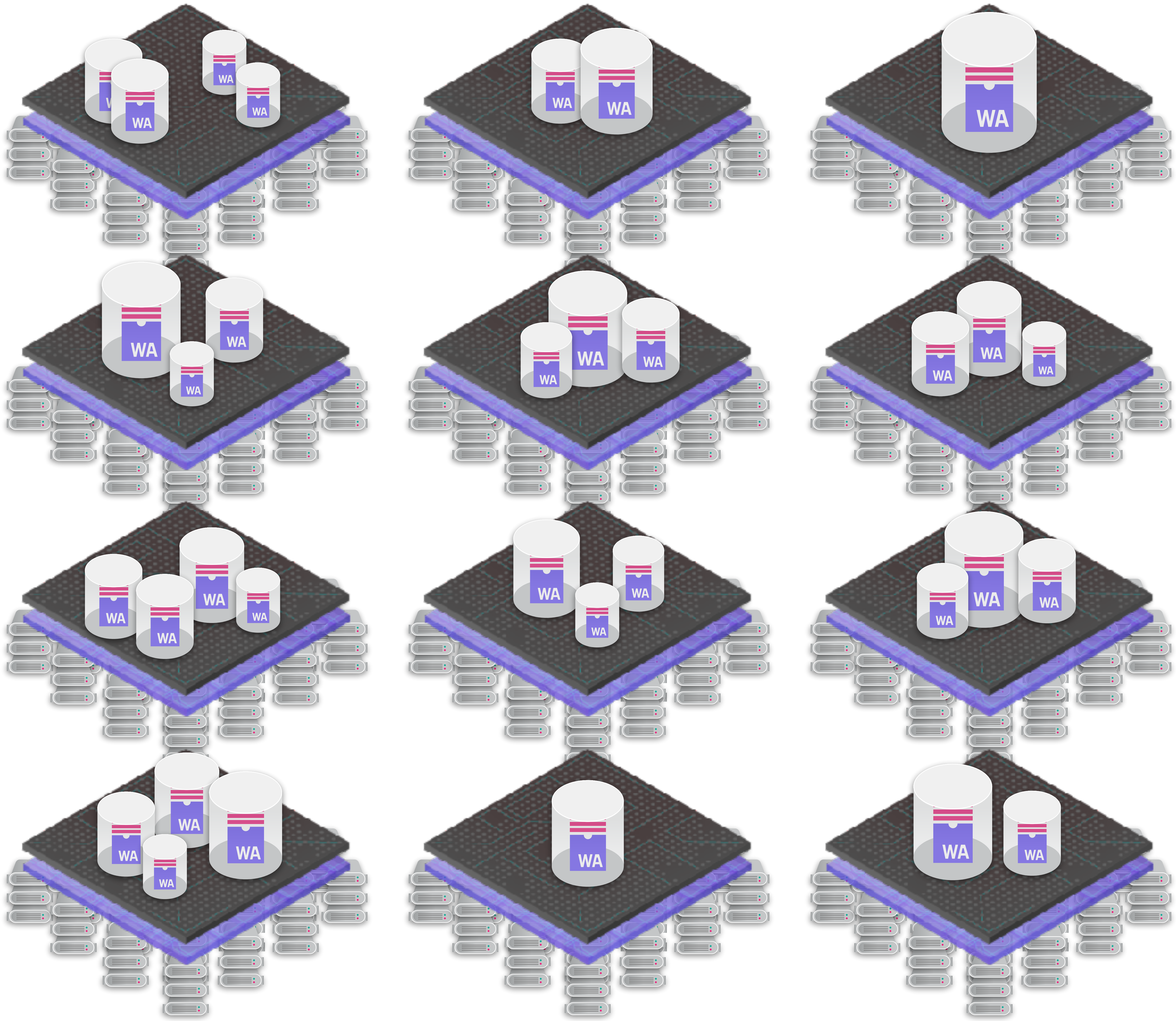
https://coincodex.com/article/14198/layer-1-performance-comparing-6-leading-blockchains/

Internet Computer Architecture

The Internet Computer is powered by a myriad of nodes

Nodes are partitioned into **subnets**.

Canister smart contracts are assigned to different subnets.



The Internet Computer is powered by a myriad of nodes

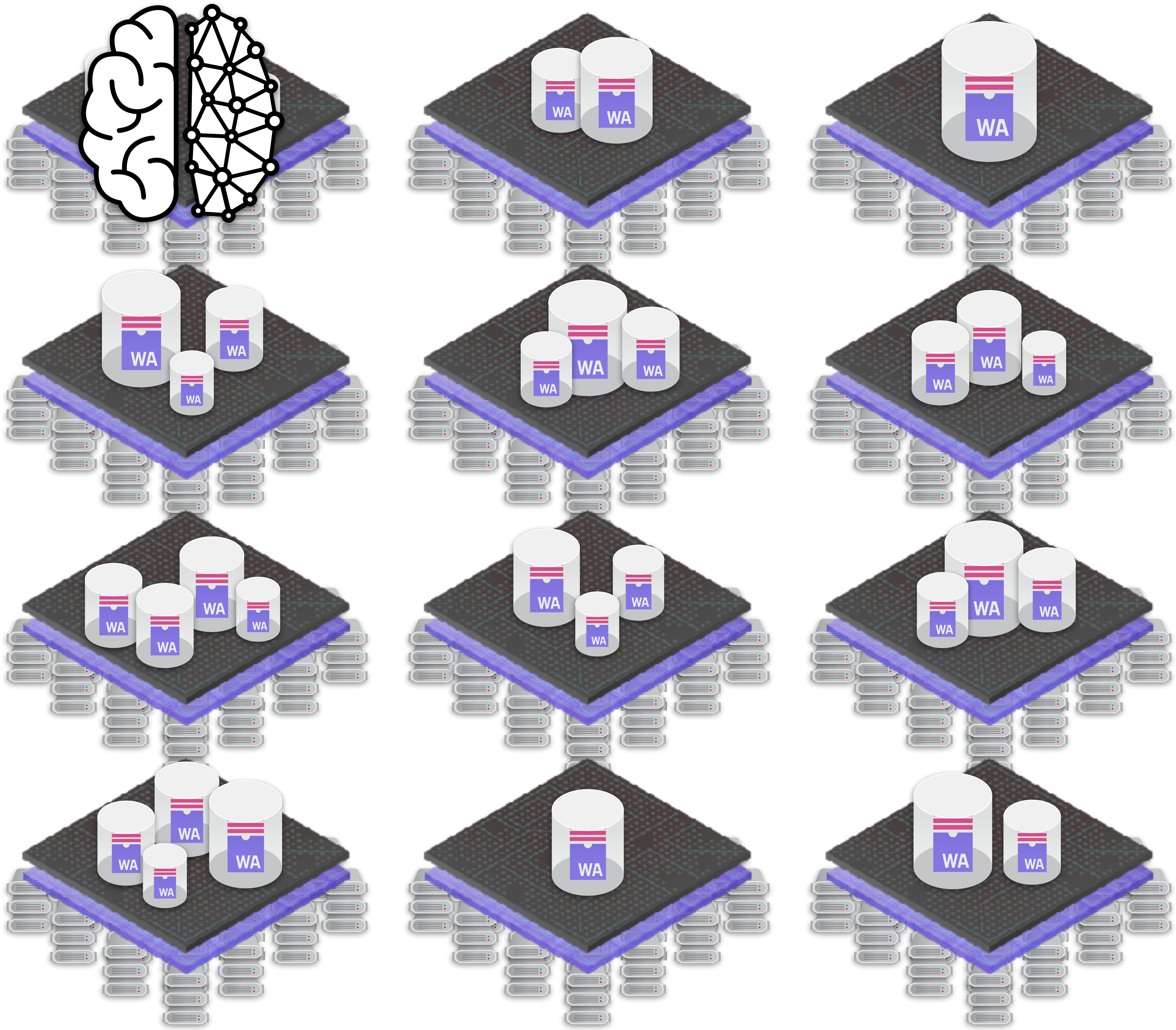
Nodes are partitioned into **subnets**.

Canister smart contracts are assigned to different subnets.

One subnet is special: it host the NNS canisters - the Network Nervous System that governs the IC

ICP token holders vote on

- Creation of new subnets
- Upgrades to new protocol version
- Replacement of nodes
- …



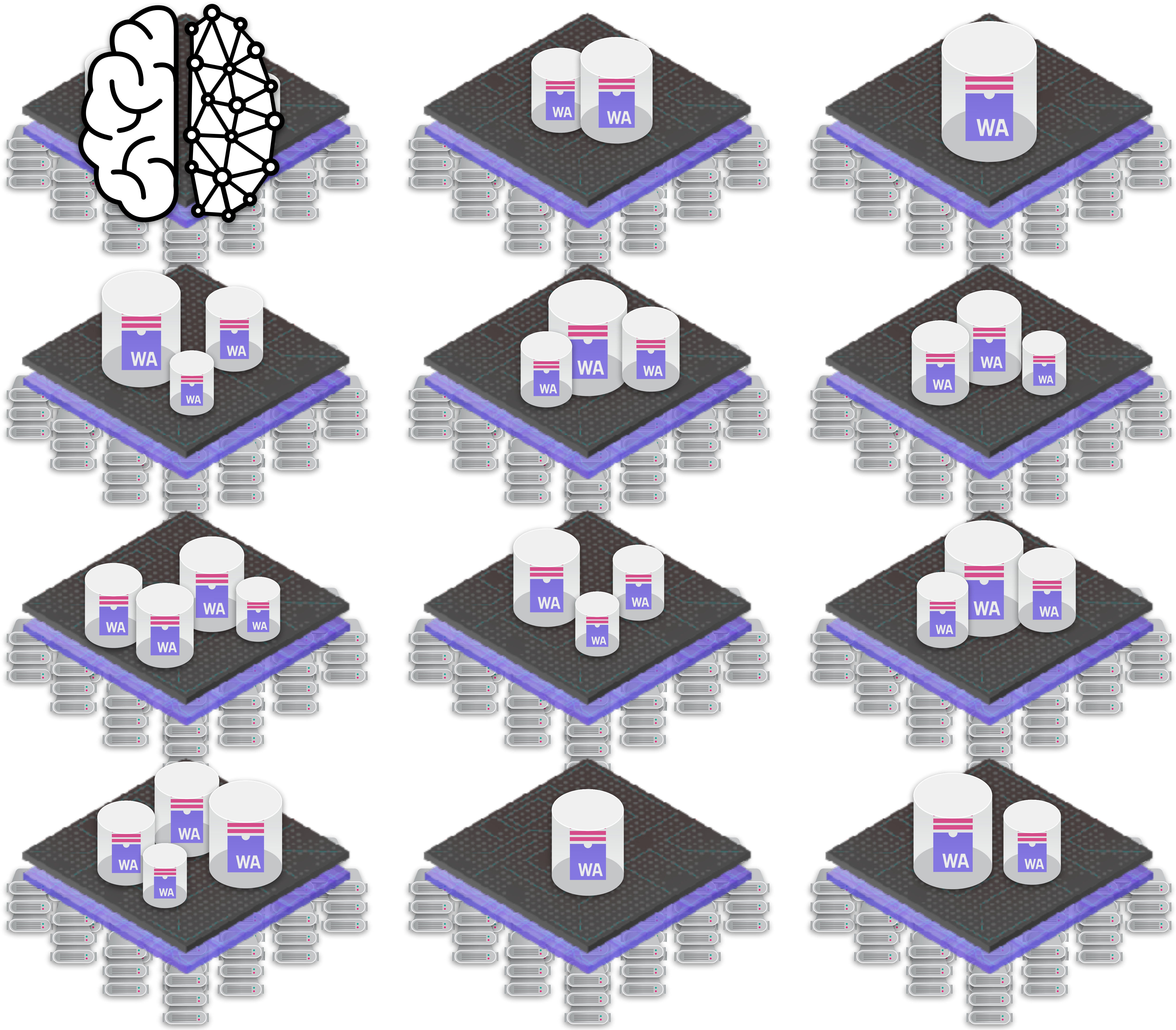
Each subnet is a replicated state machine

State:
- canisters and their queues

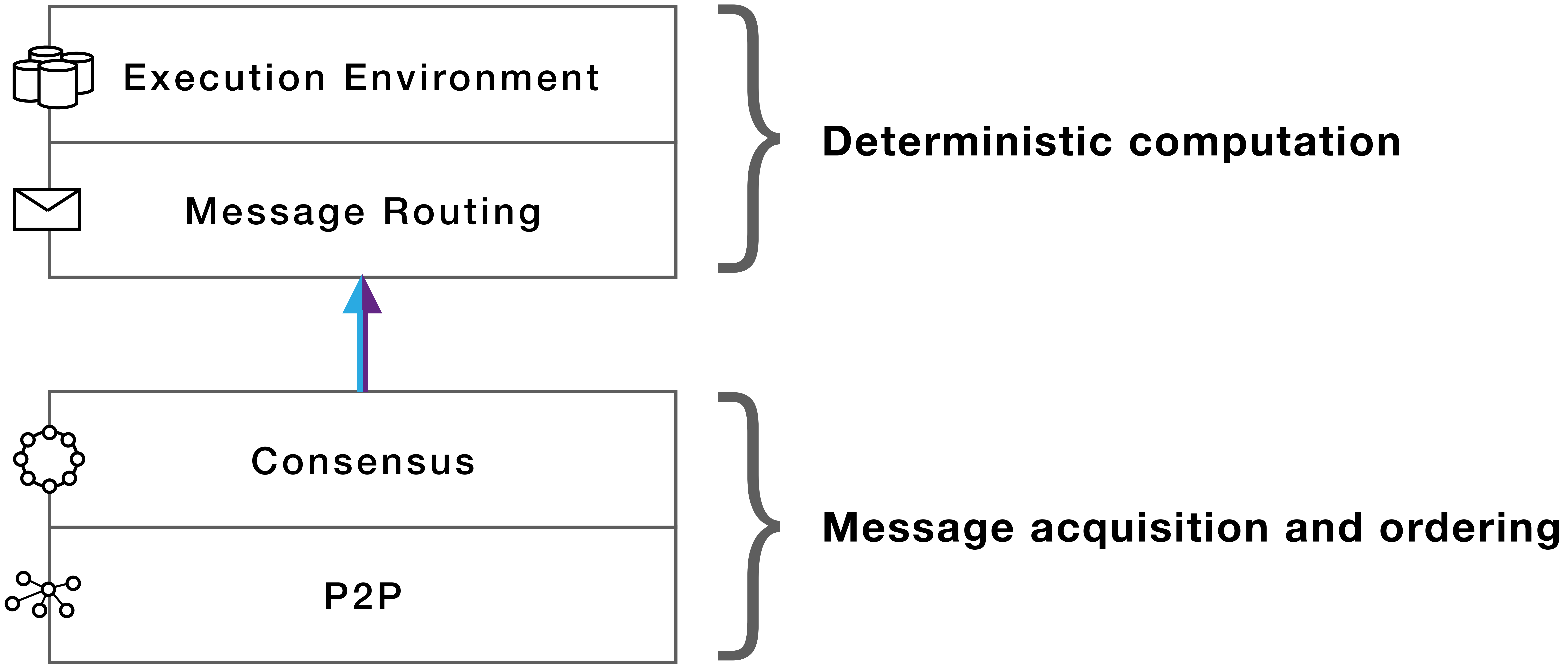
Inputs:
- new canisters to be installed,
- messages from users and other canisters

Outputs:
- responses to users and other canisters

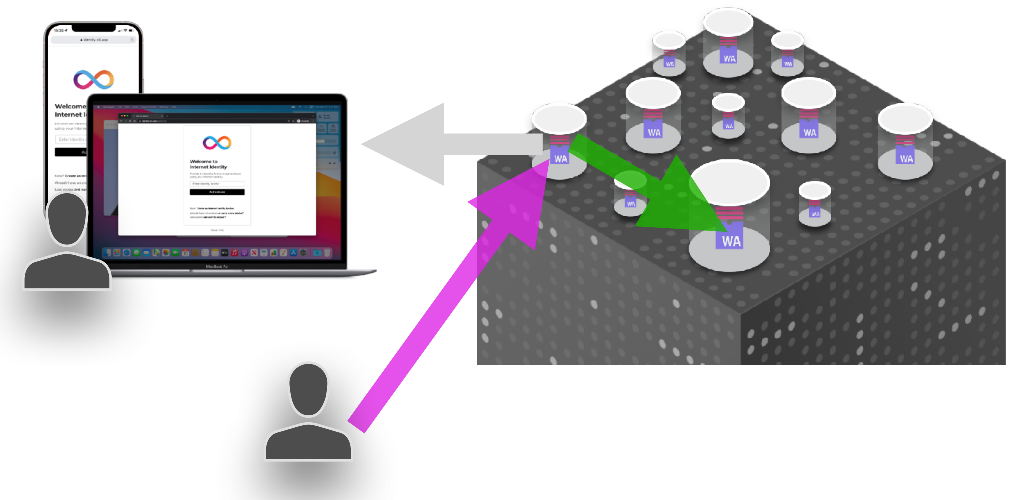
Transition function:
- message routing and scheduling
- canister code

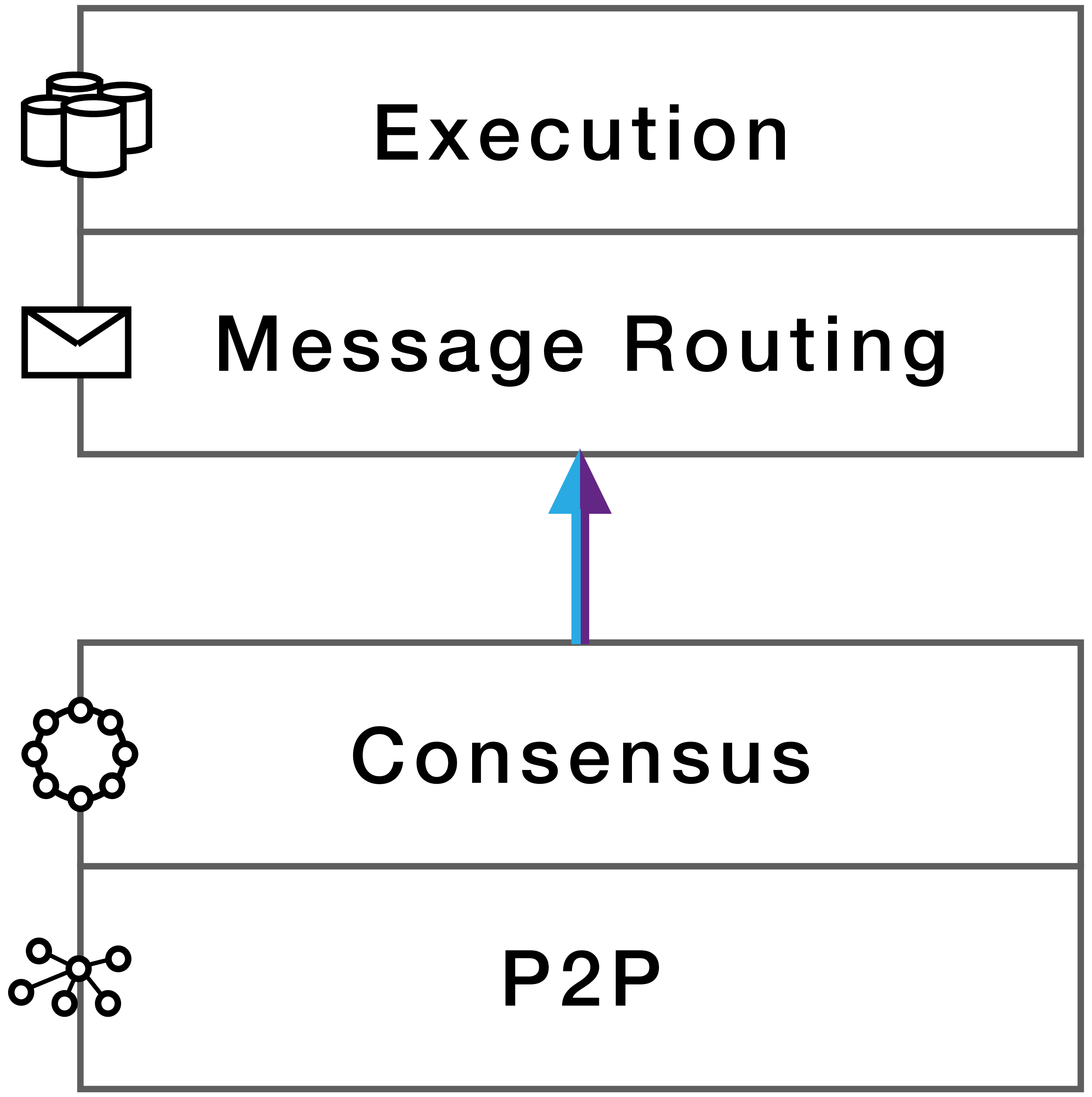


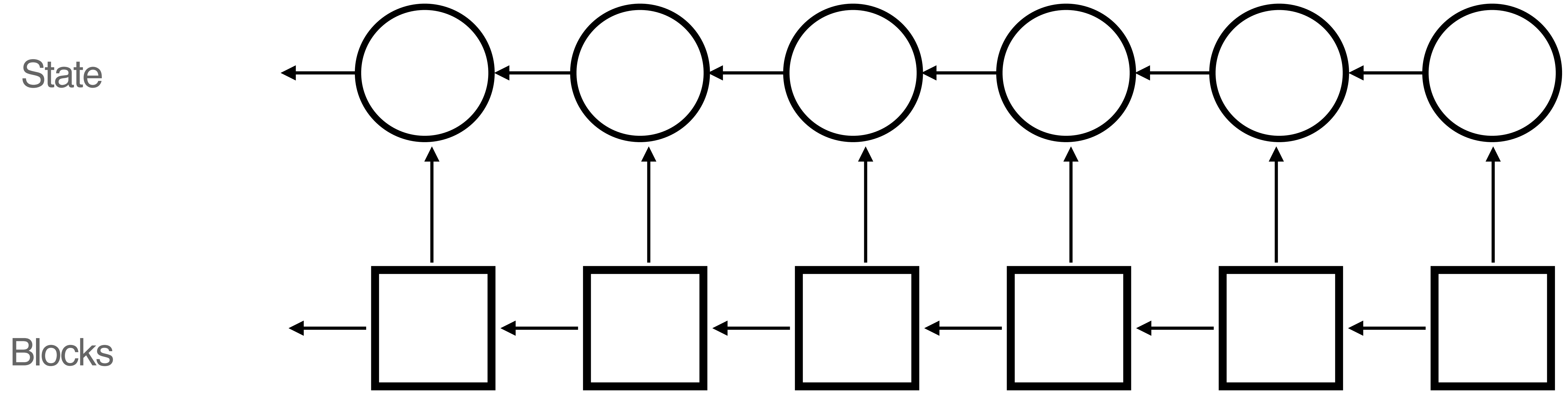
The layers of the Internet Computer Protocol



The layers of the Internet Computer Protocol

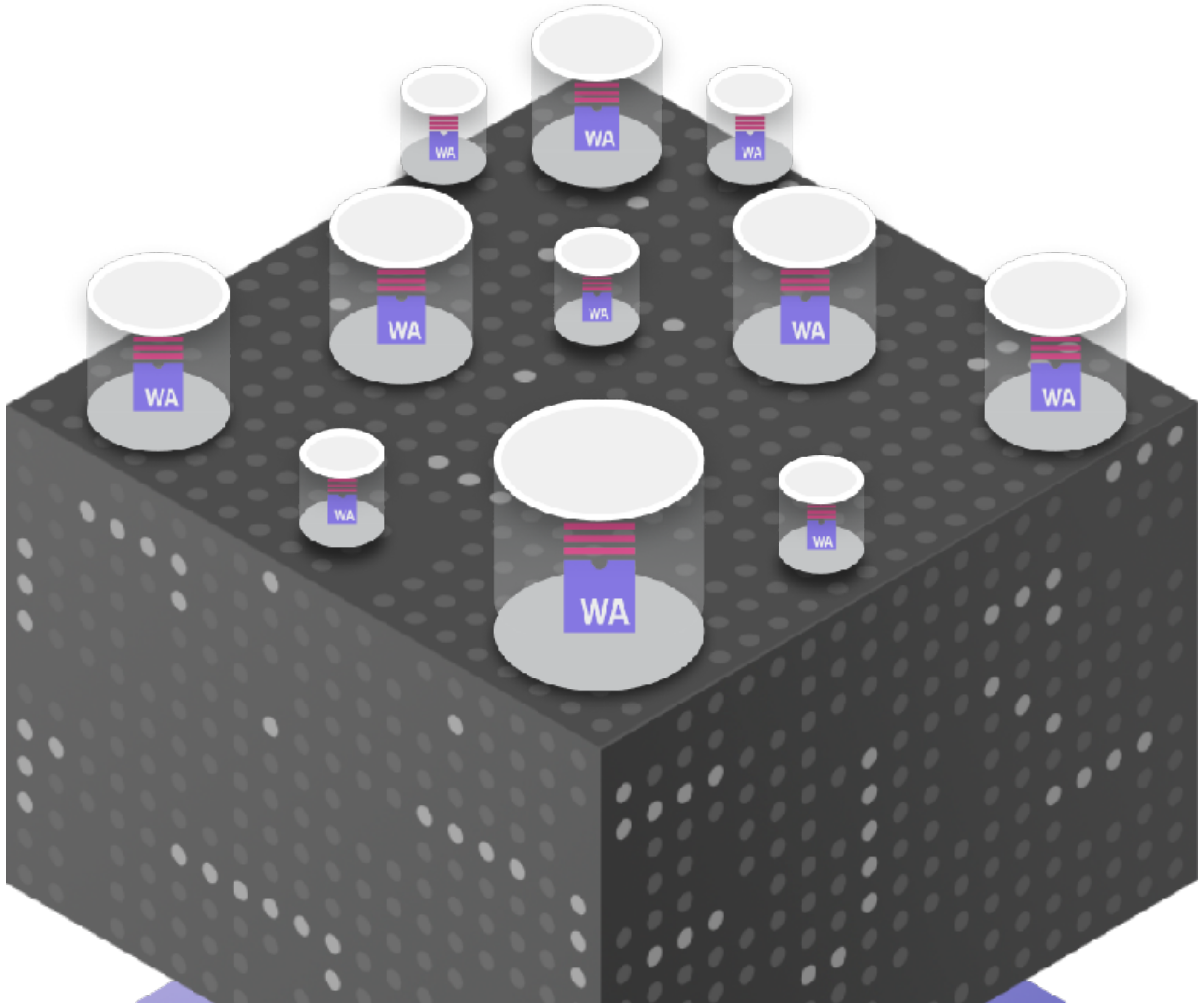






State tree

- Huge piece of tree-structured data.
- Size: up to several GiB per canister, 100s GiBs total.
- 64KiB memory pages for efficient storage on disk



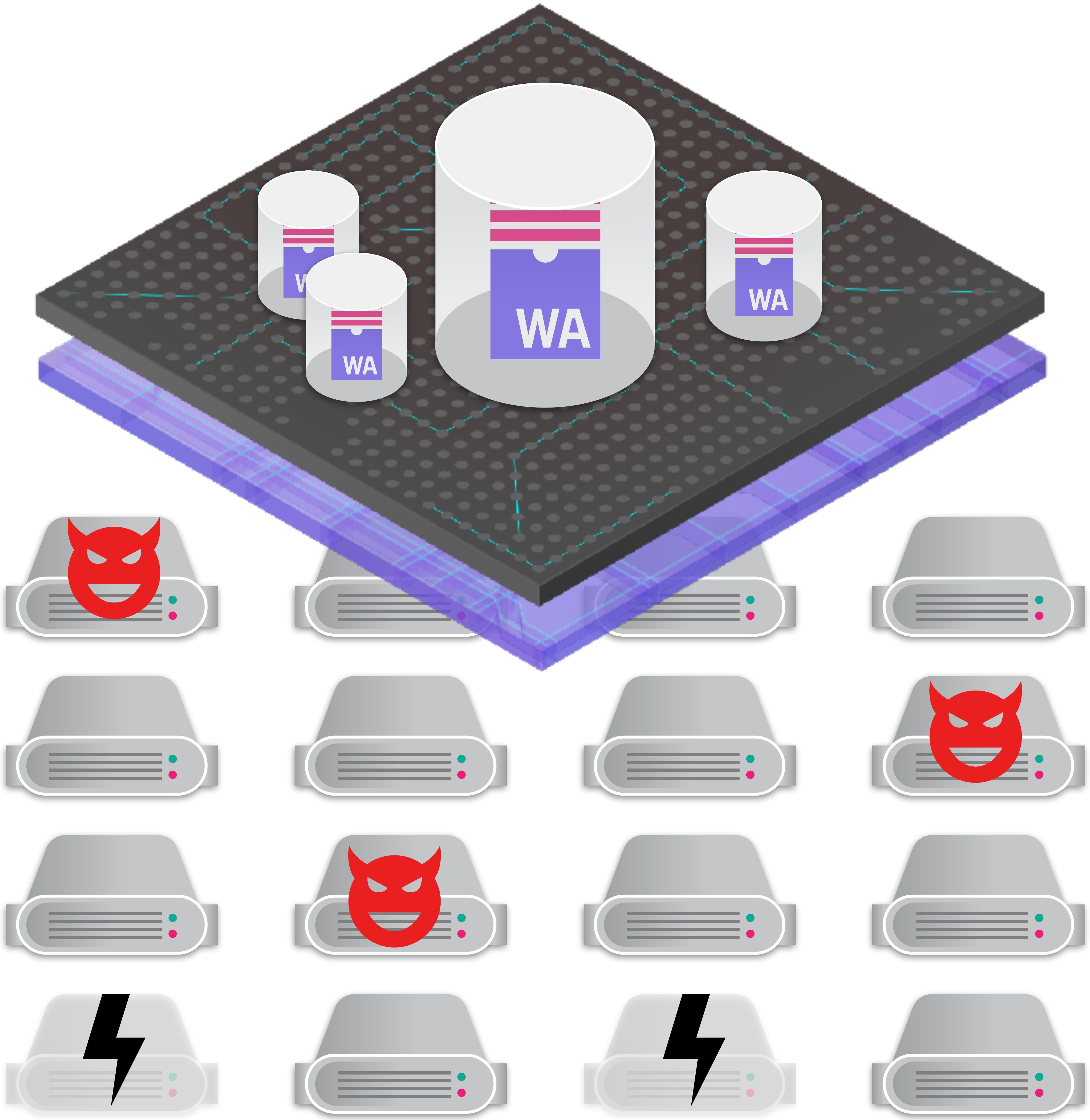
- state@5000
  - canisters
    - canister 1
      - input queues
      - output queues
      - memory0
        - 0xcf624a15dbf7f70…
        - 0x13d7232f4e05100…
        - …
    - canister 2
    - …
  - system metadata
    - ingress history
    - streams state

Fault-tolerance in a subnet

Honest replicas can fall behind

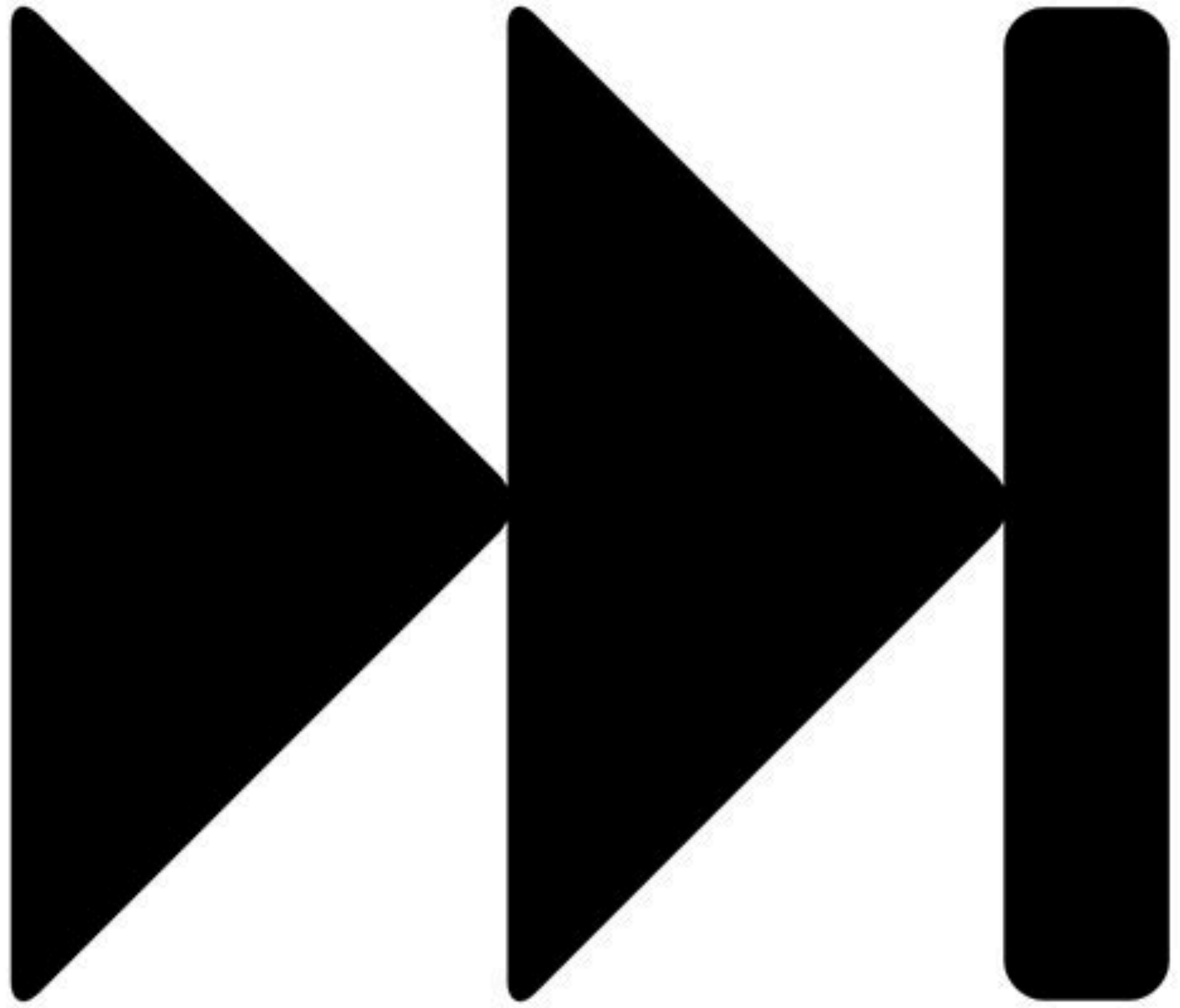
- Temporary network outage
- Power cycle
- Reboot after maintenance
- …

How do they catch up with the rest?



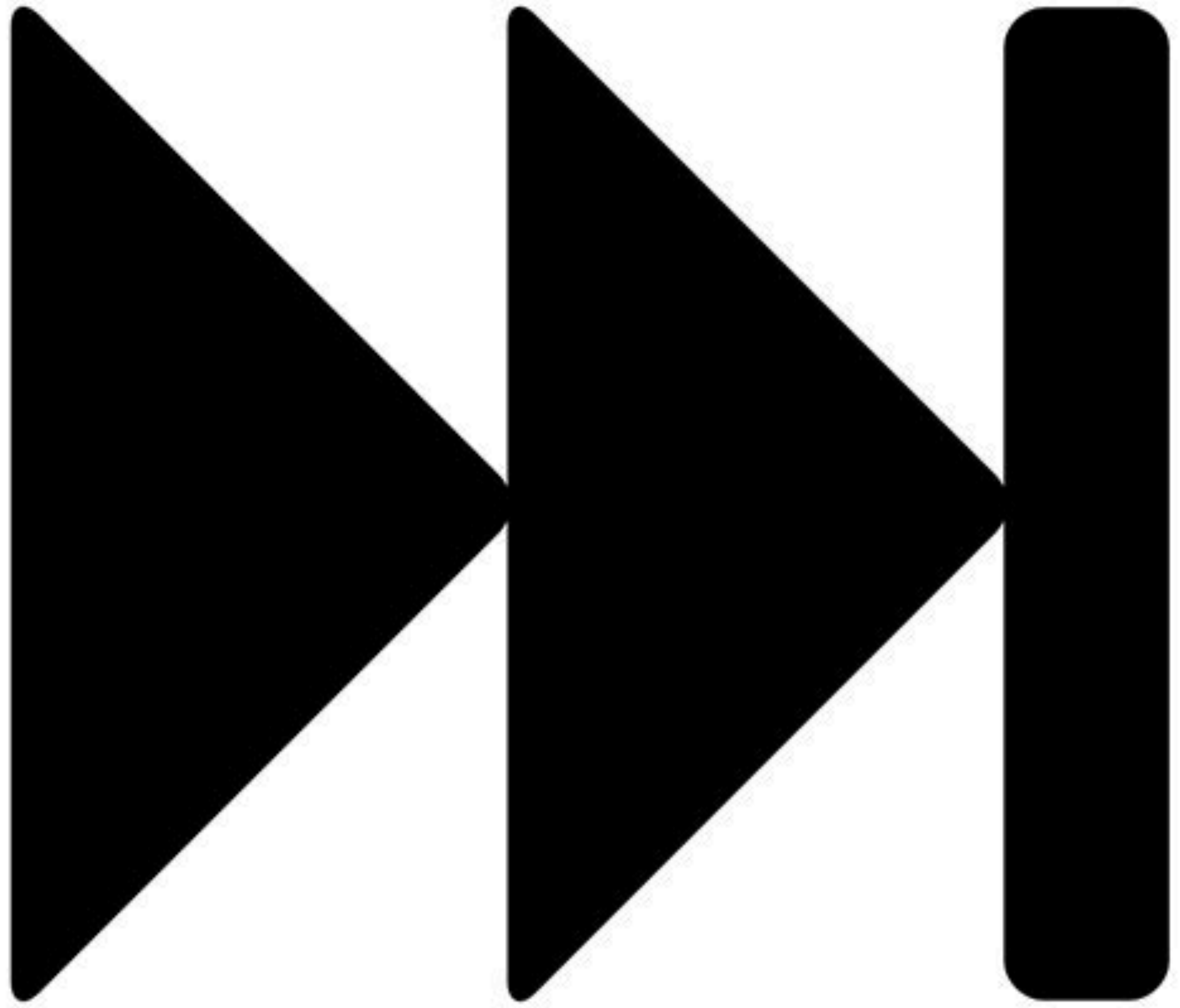
State Synchronization

Requirements



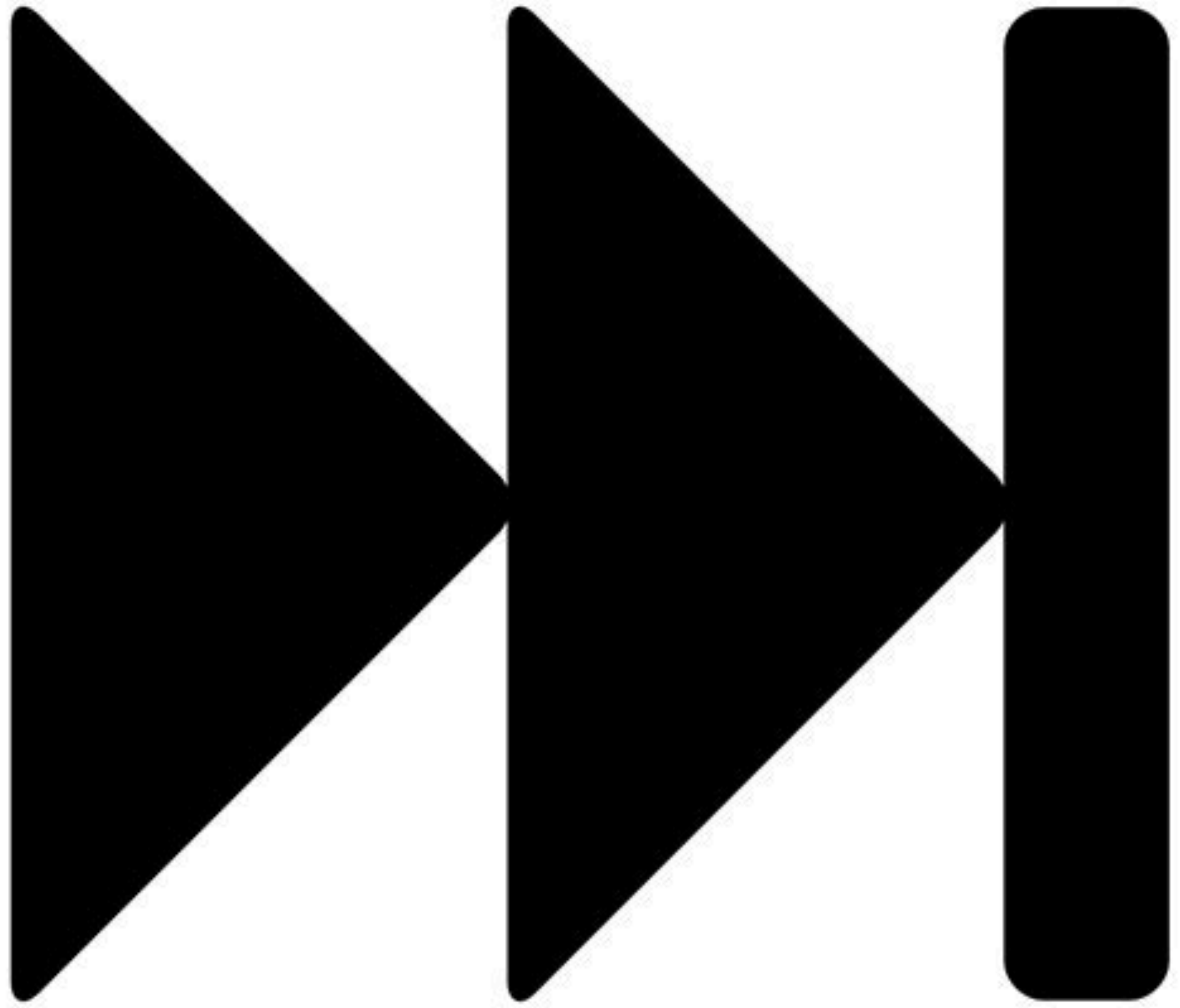
- Cope with Byzantine parties 

Requirements



- Cope with Byzantine parties 

- Bounded Memory and disk space

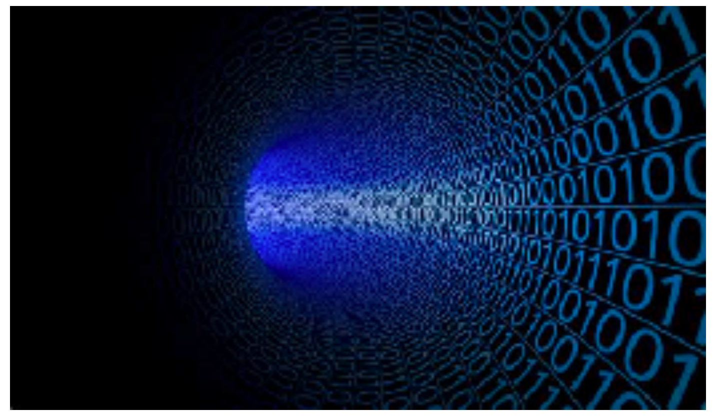


Requirements

- Cope with Byzantine parties

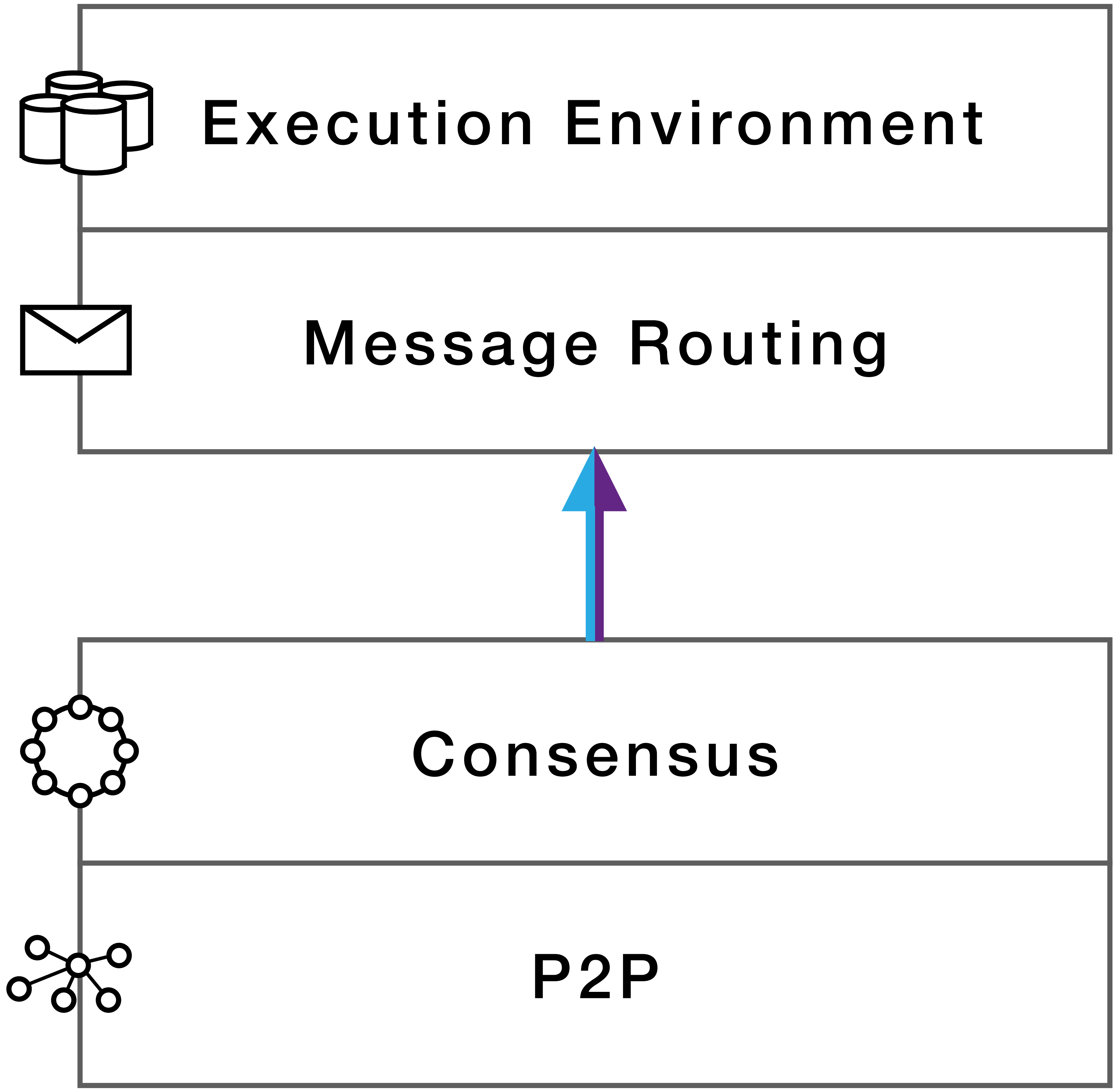
- Bounded Memory and disk space

- Minimize bandwidth and computation complexity



Catching Up

What a node needs to fully participate in the protocol



Canister state: to process messages

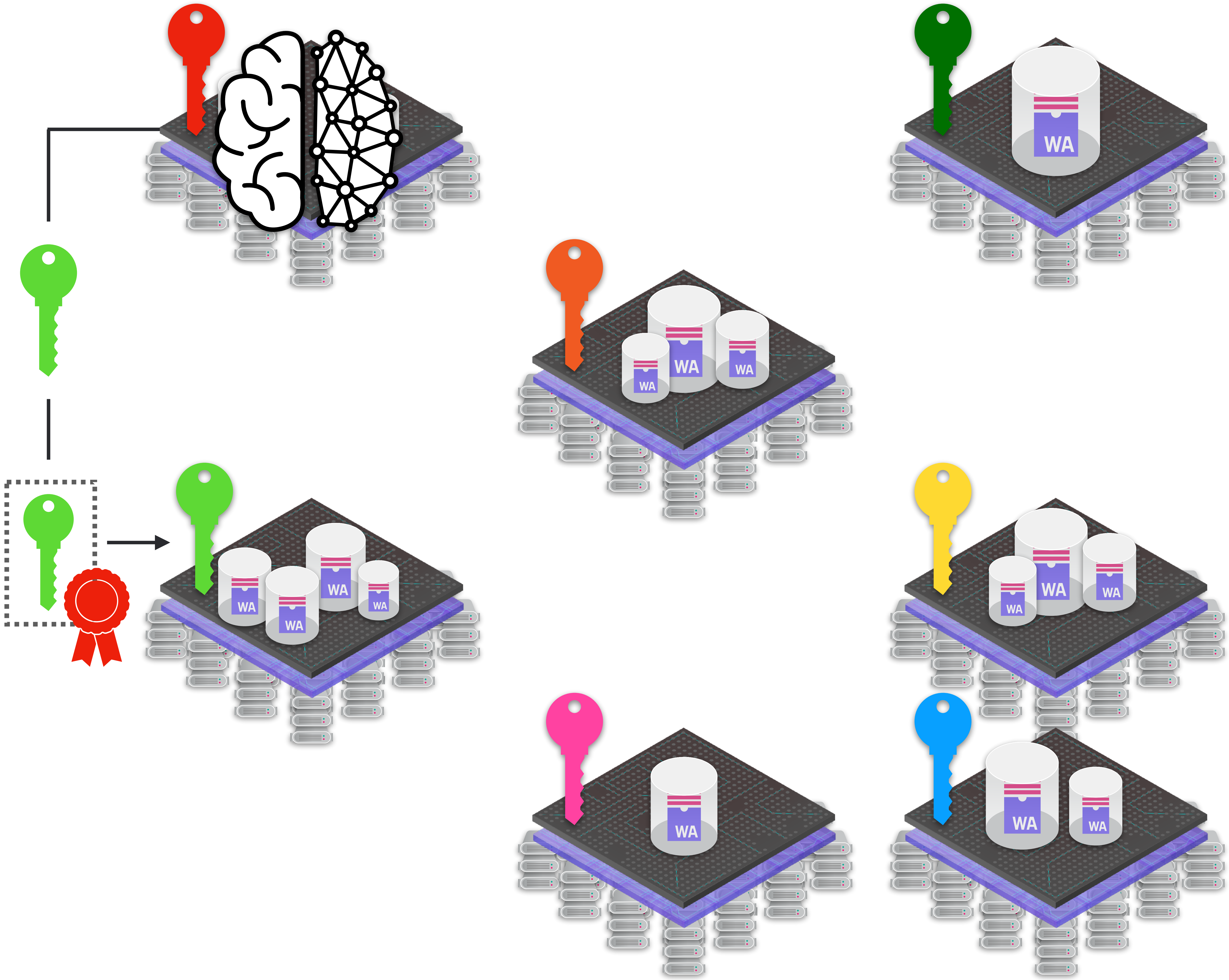
Queue state: to schedule and route messages

Key material: to sign and verify messages

Peers info: who to connect to and how

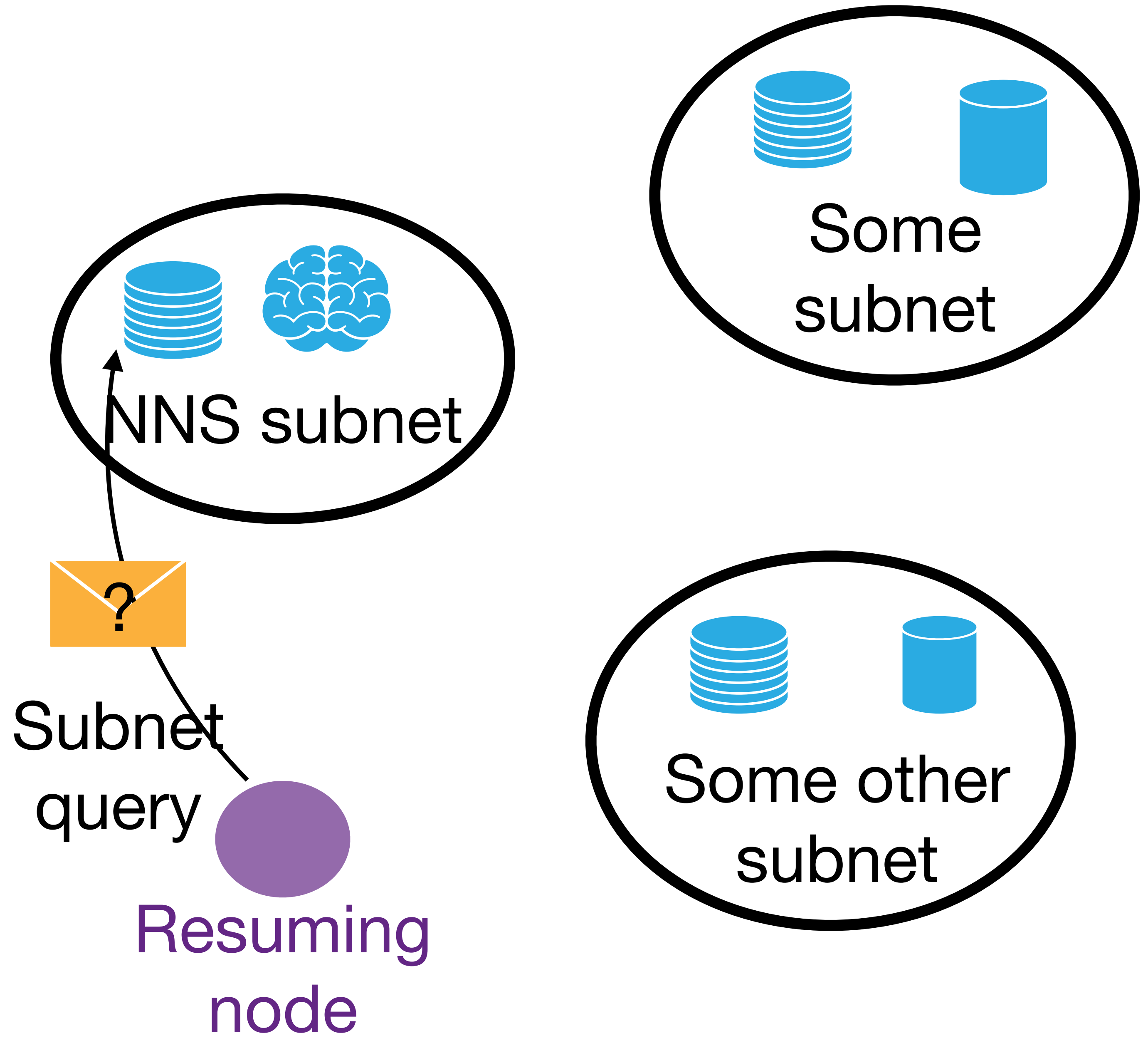
Chain Key Cryptography: Key management

NNS generates key of subnets and certifies them.



P2P Resumability

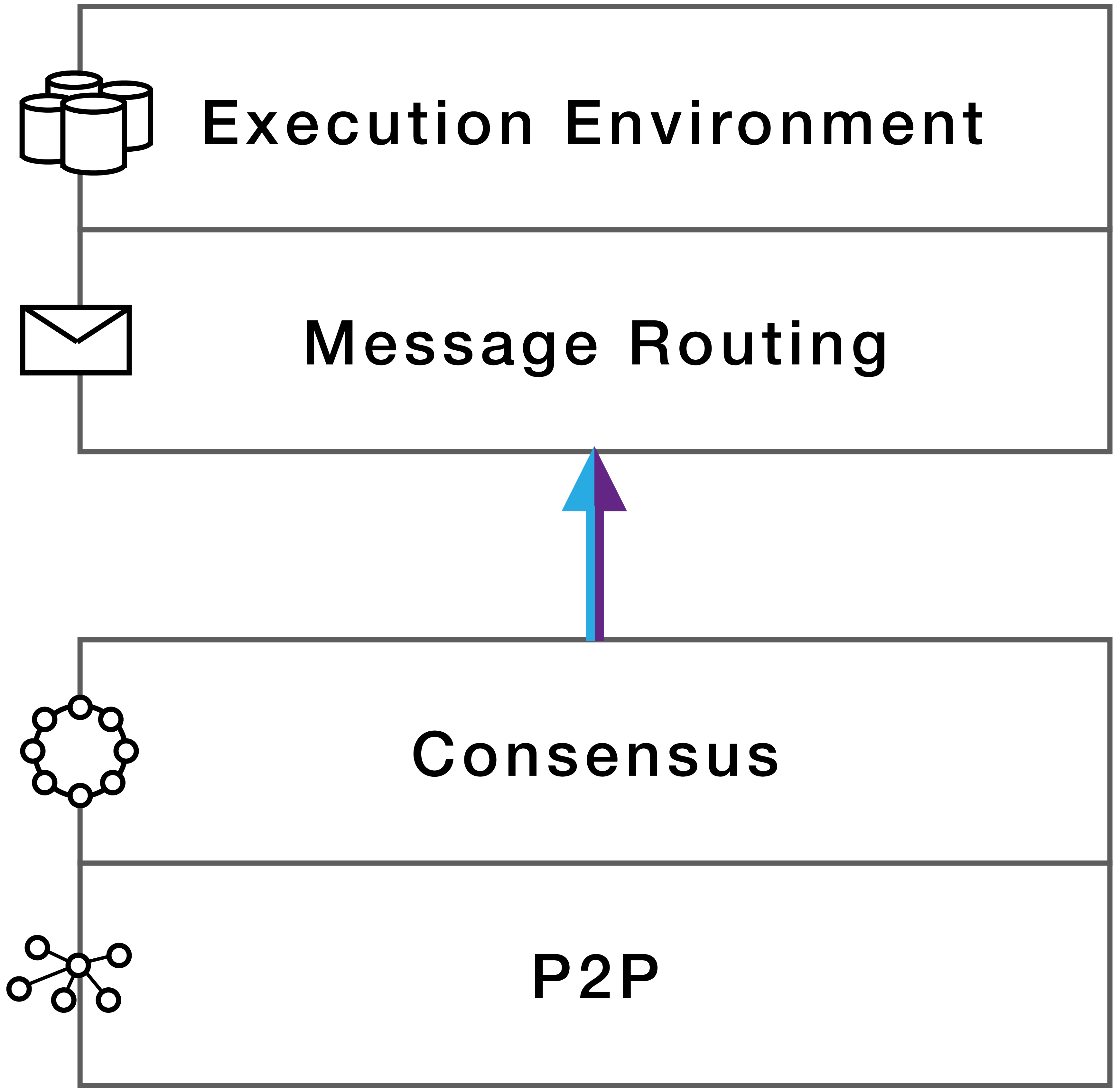
Basic premise: only connect to subnet overlay neighbors, at any time (to mitigate DOS attacks)



Resuming node *v*

- *v* is initialized with NNS public key
- Can verify NNS responses
- Repeatedly NNS subnet membership
- Determine other nodes in *v*'s subnet and subnet key

What a node needs to fully participate in the protocol



Canister state: to process messages

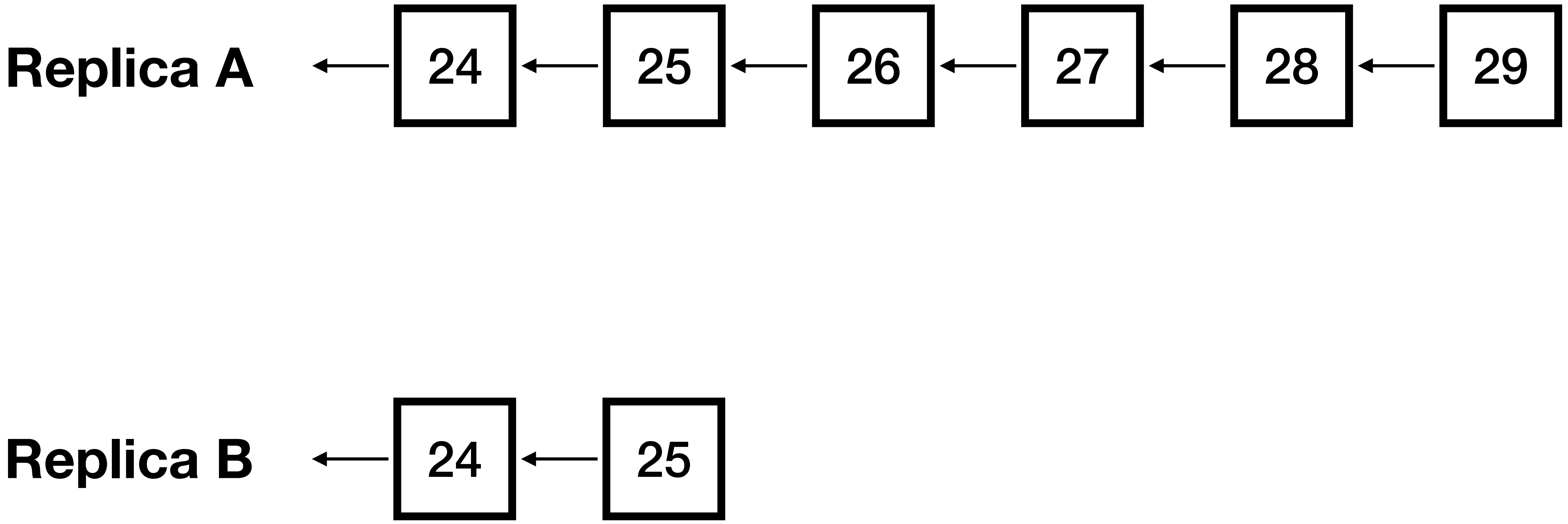
Queue state: to schedule and route messages

Key material: to sign and verify messages

Peers info: who to connect to and how

Consensus Resumability

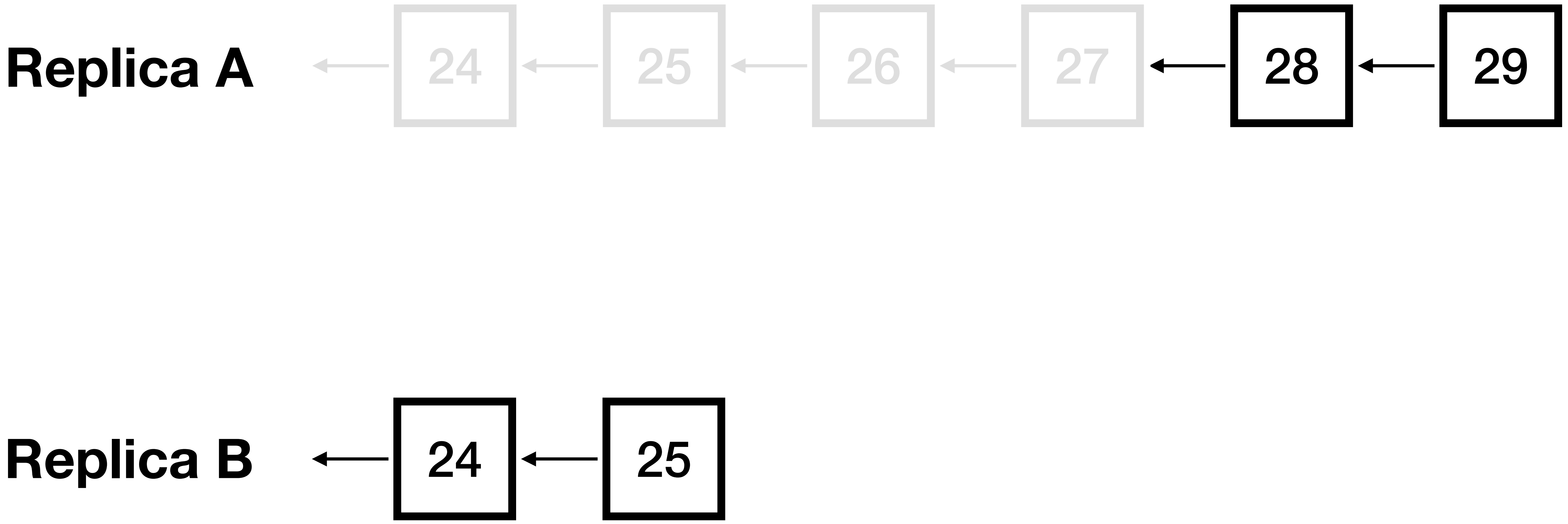
- Easy case: missing information still available from peers



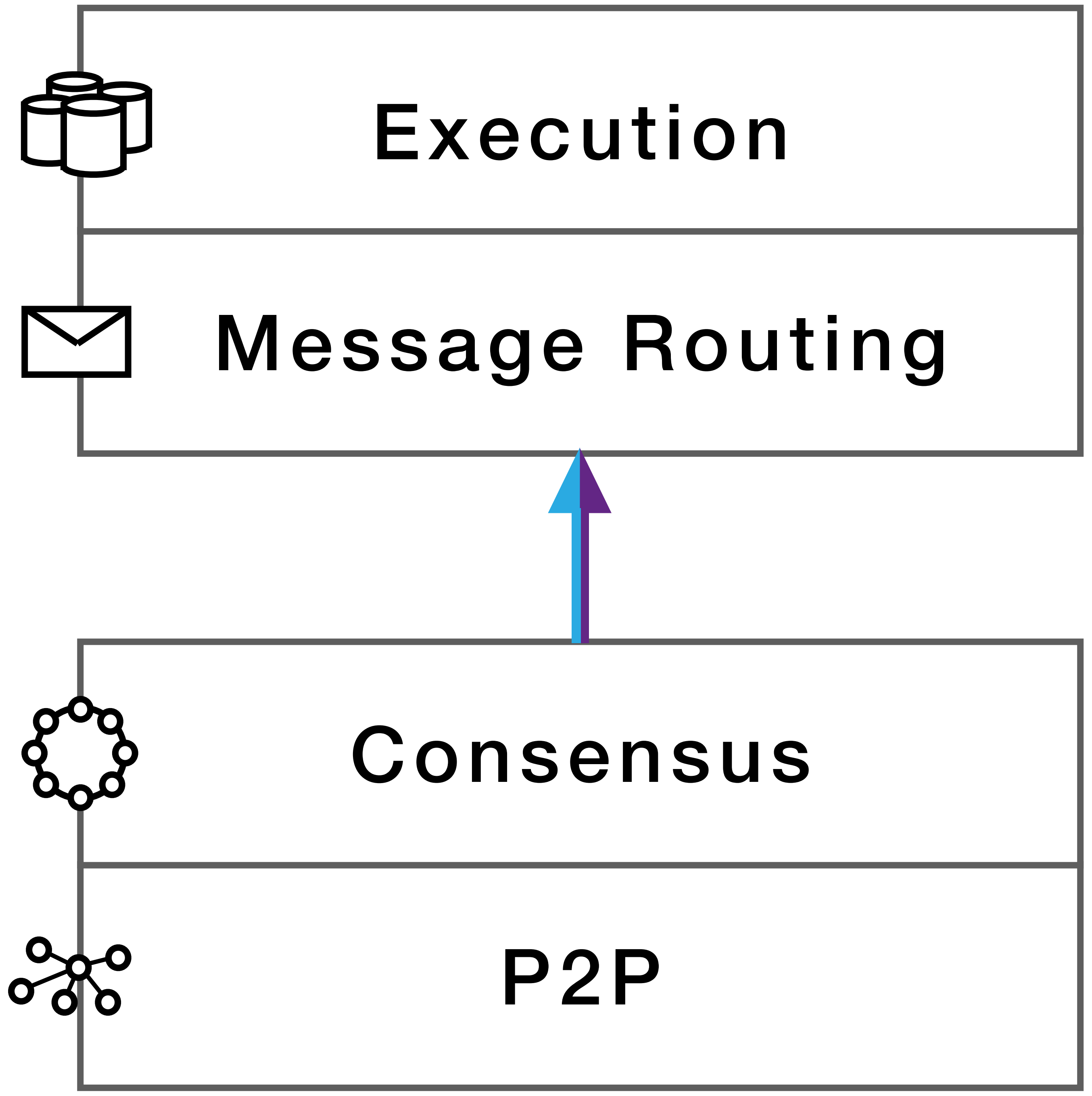
just fetch missing messages, construct blocks and execute the messages contained in them

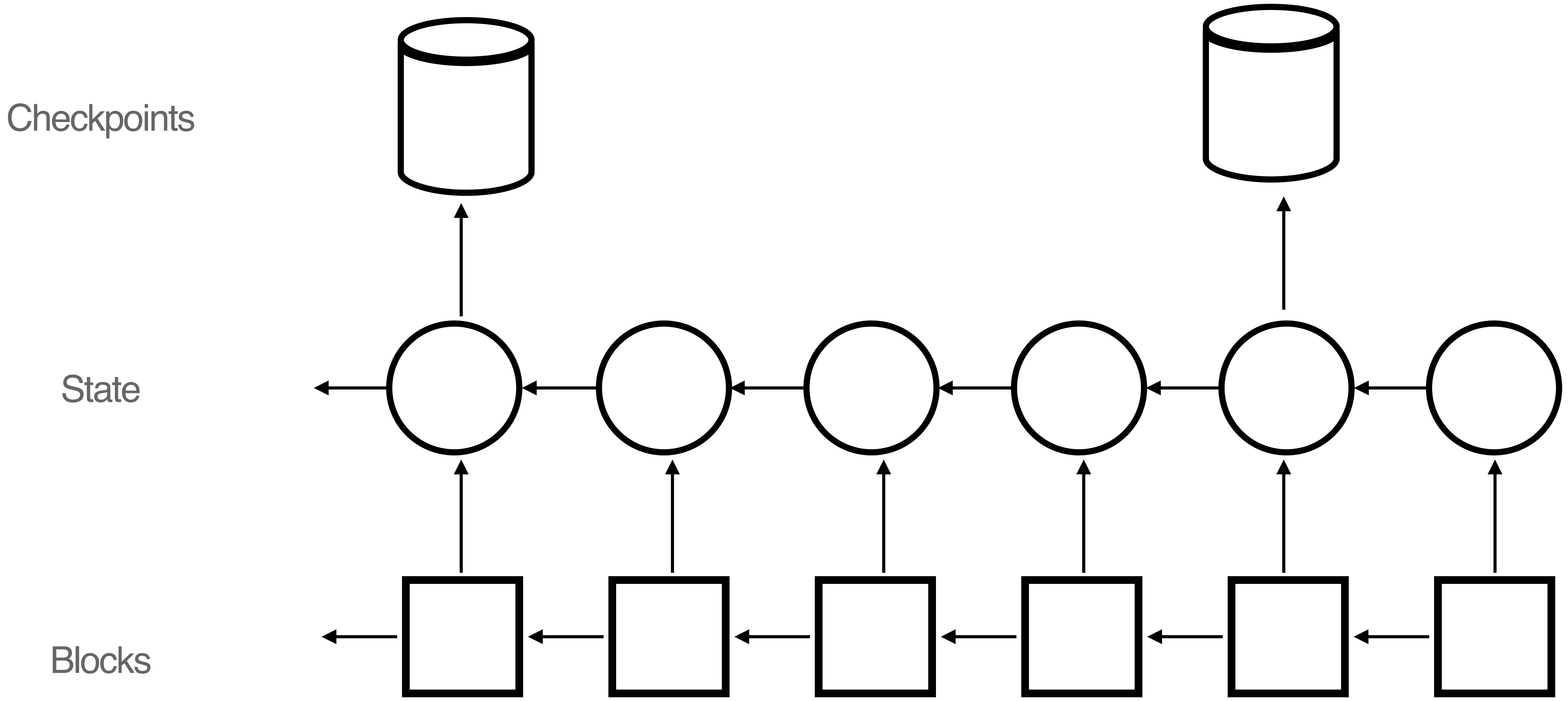
Consensus Resumability

- More difficult case: peers have purged missing information



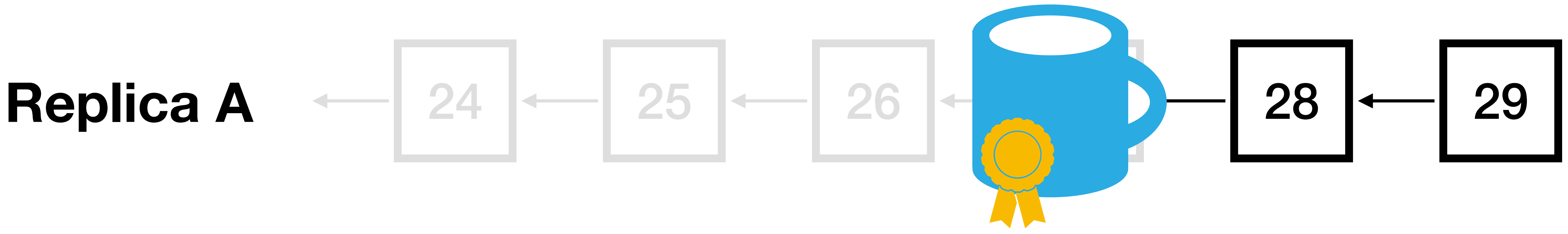
Checkpointing

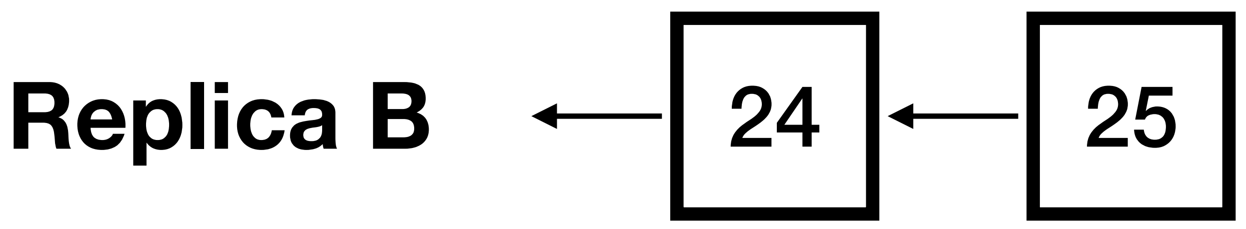




Consensus Resumability

- More difficult case: peers have purged missing information



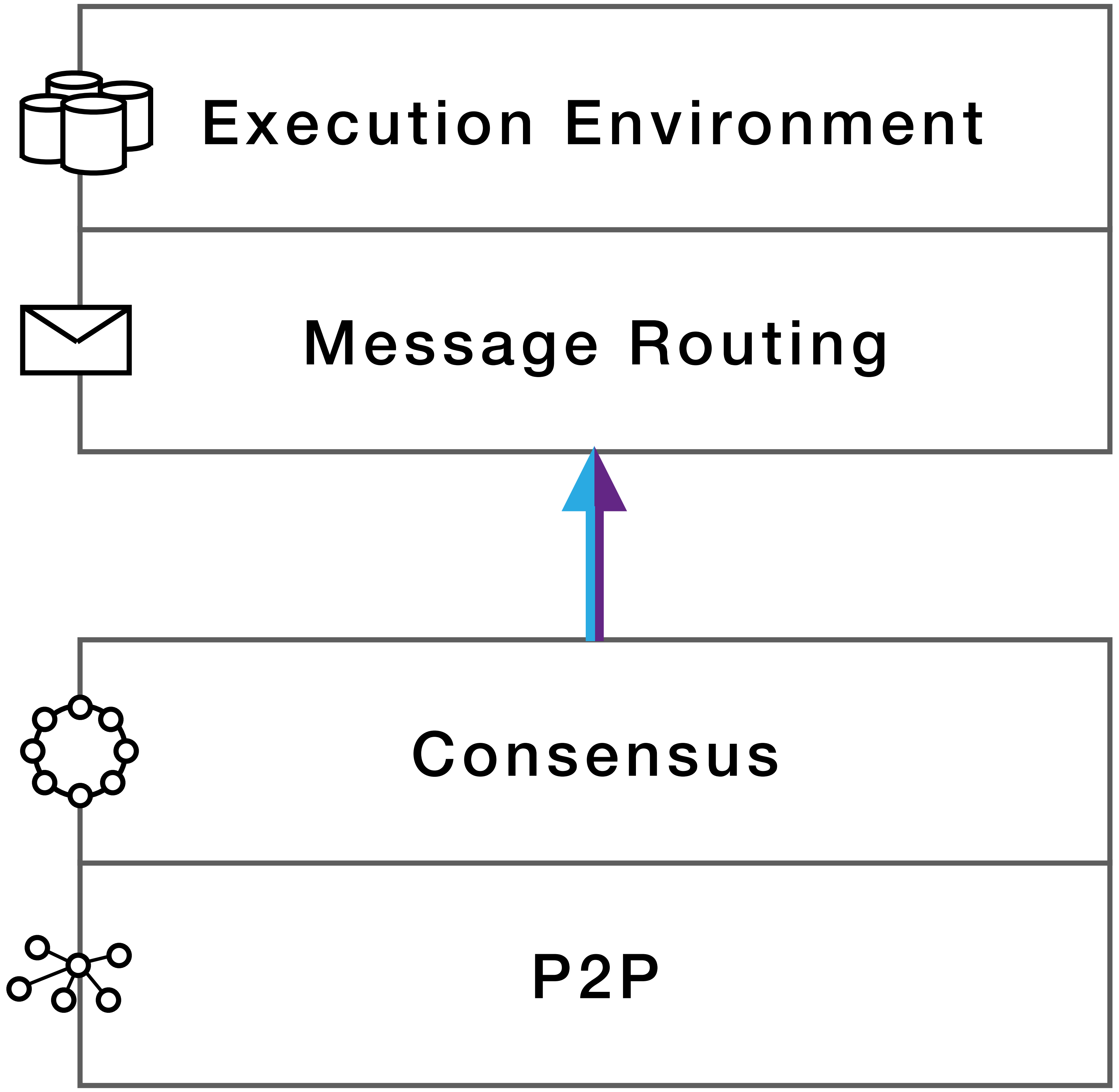


Catch-up package (CUP) containing

- Key material
- Consensus information
- Hash of checkpoint

Signed with subnet key

What a node needs to fully participate in the protocol



Canister state: to process messages

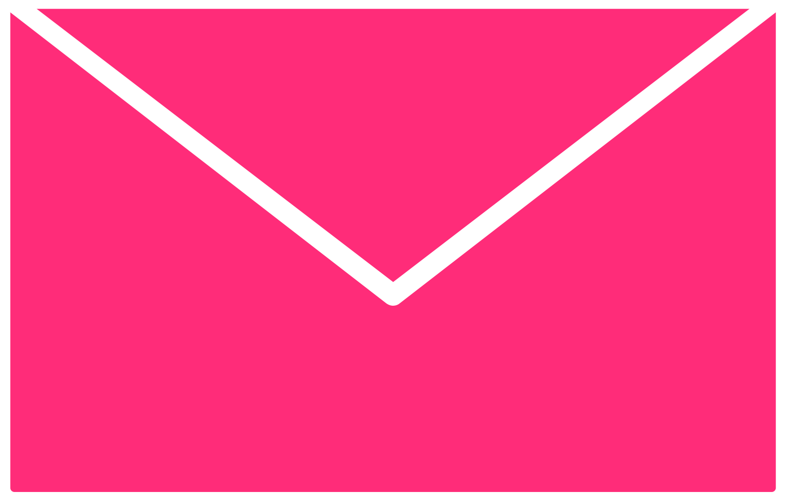
Queue state: to schedule and route messages

Key material: to sign and verify messages ✓

Peers info: who to connect to and how ✓

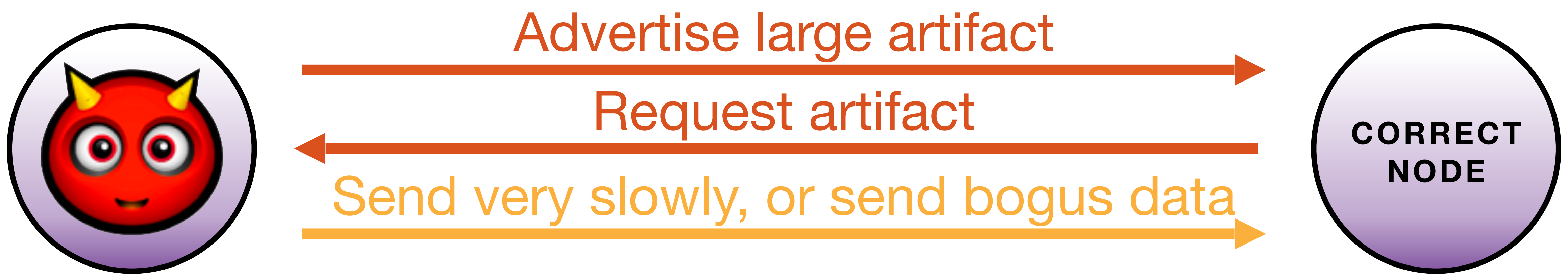
Chunking

Security Problem: Delivery Tampering Attacks



The problem with large artifacts

- need long download timeout
  e.g., 8 MB artifact over 1Gbps connection shared by 30 nodes in rack, 25 peers per node
  = 47s expected download time → timeout > 1m30s
- can be exploited by bad peer to prevent (timely) delivery



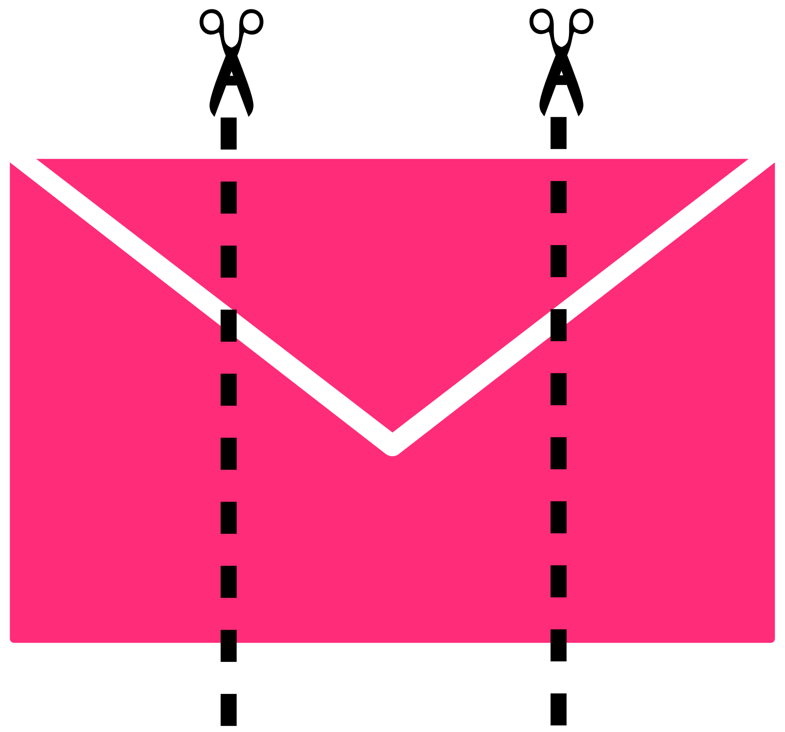
E.g., bad peer can block statesync by

- being first peer to advertize it (skipping checks)
- send bogus data until download times out
- repeat with other bad peers until lower-ranked block finalized

Solution: Chunking

Solution: split up in smaller chunks that can be

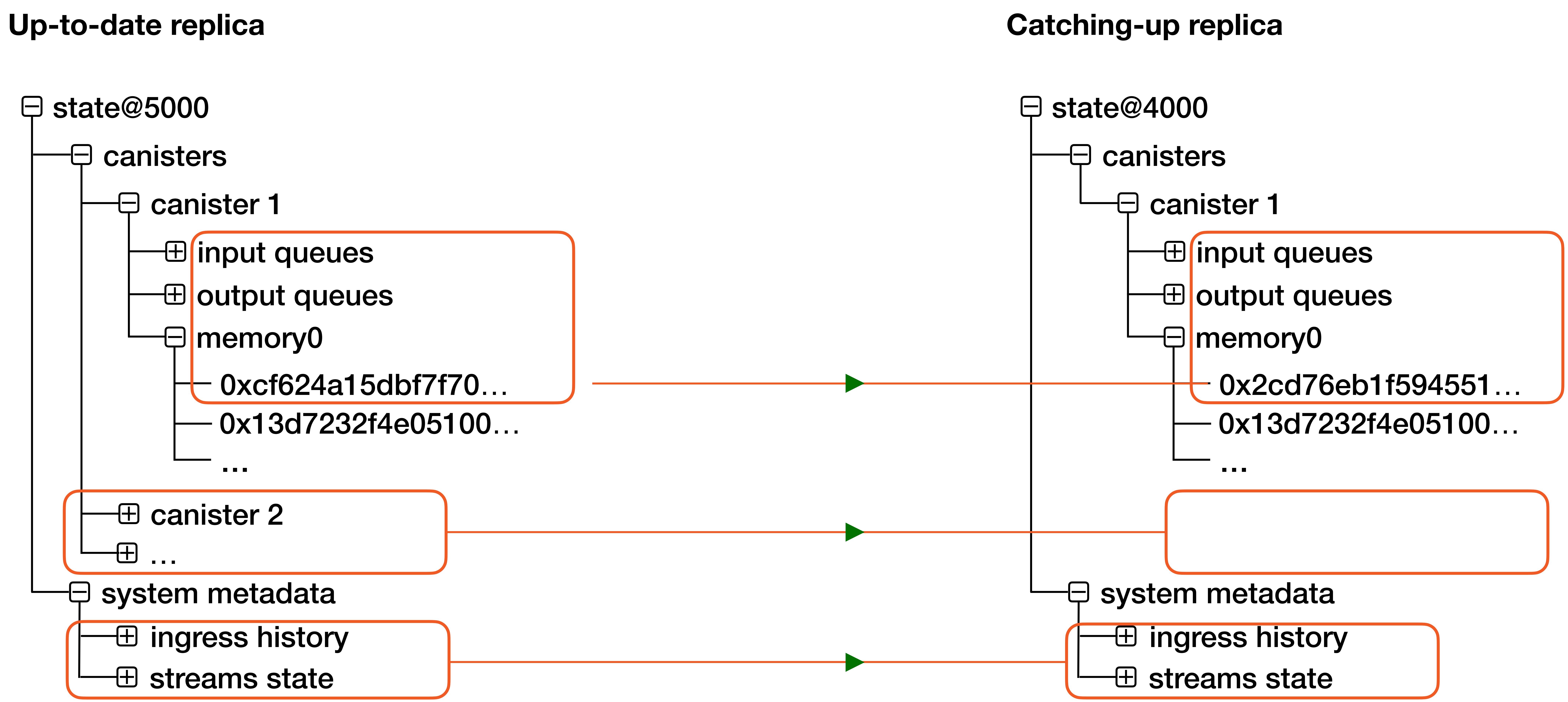
- requested separately
- downloaded in parallel



Advantages:

- shorter download timeouts
  → fail earlier
- parallelize download from multiple peers
  → lower latency
  → better bandwidth utilisation

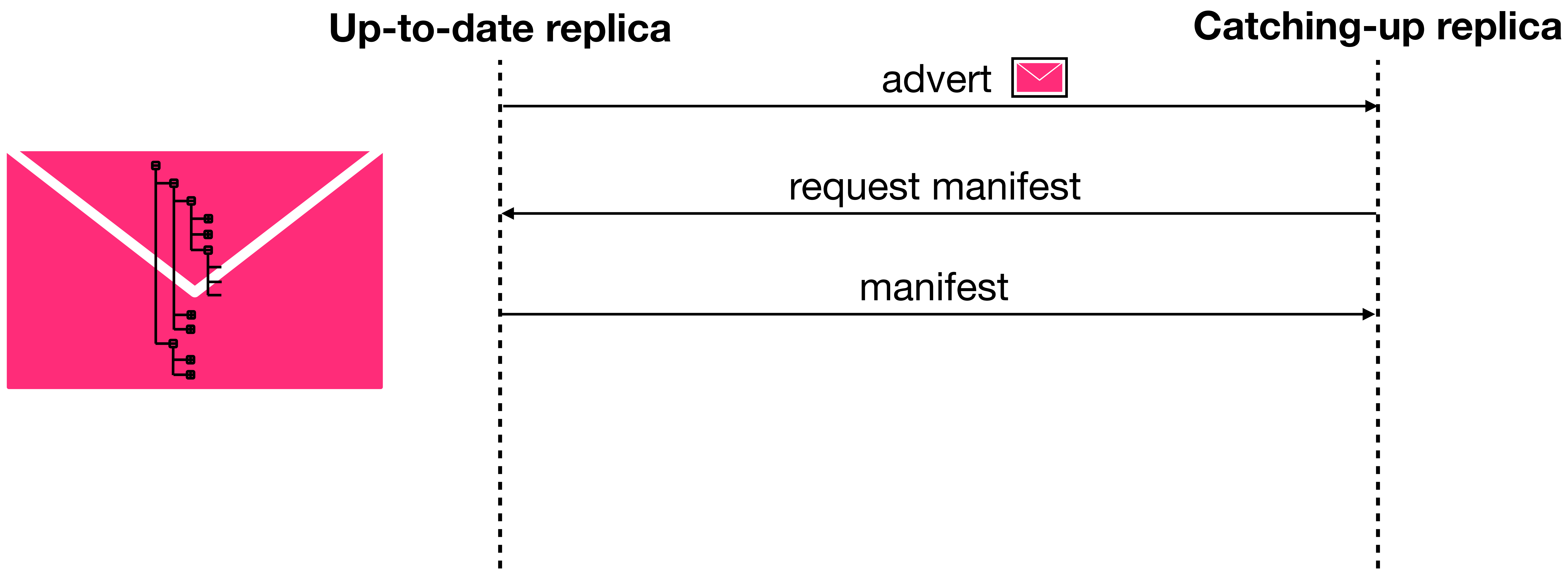
Tree structure of state



Design Overview

Announce state as one big artifact, use tree structure to request chunks

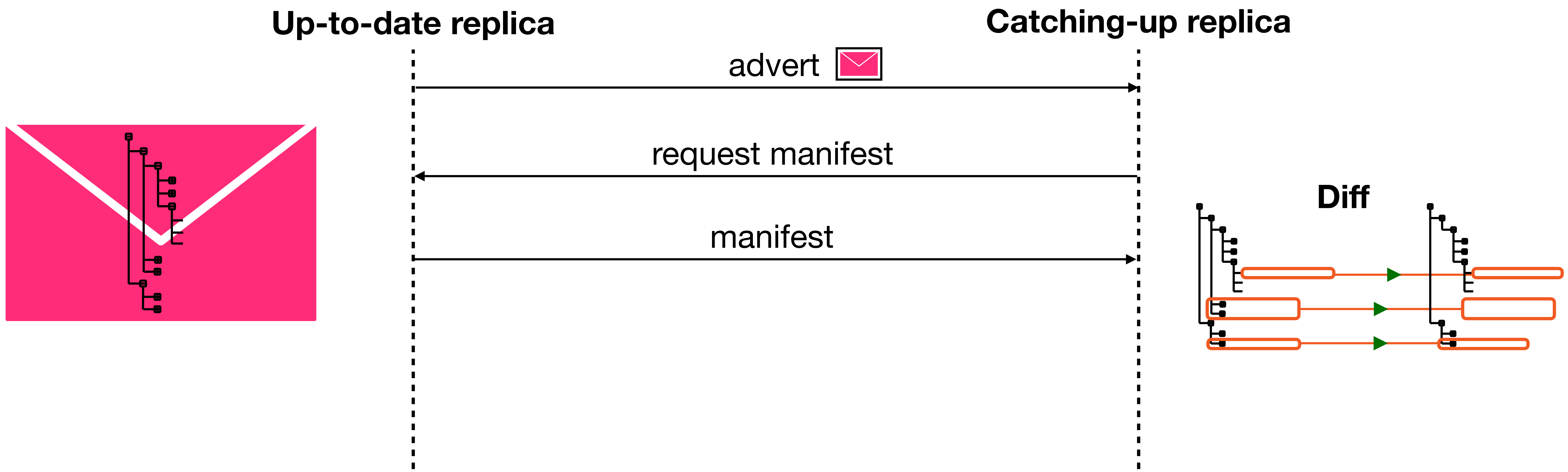
- Chunking mechanism
  - first request manifest with leaf/subtree hashes
  - then determine chunks to fetch (as opposed to always fetching all chunks)



Design Overview

Announce state as one big artifact, use tree structure to request chunks

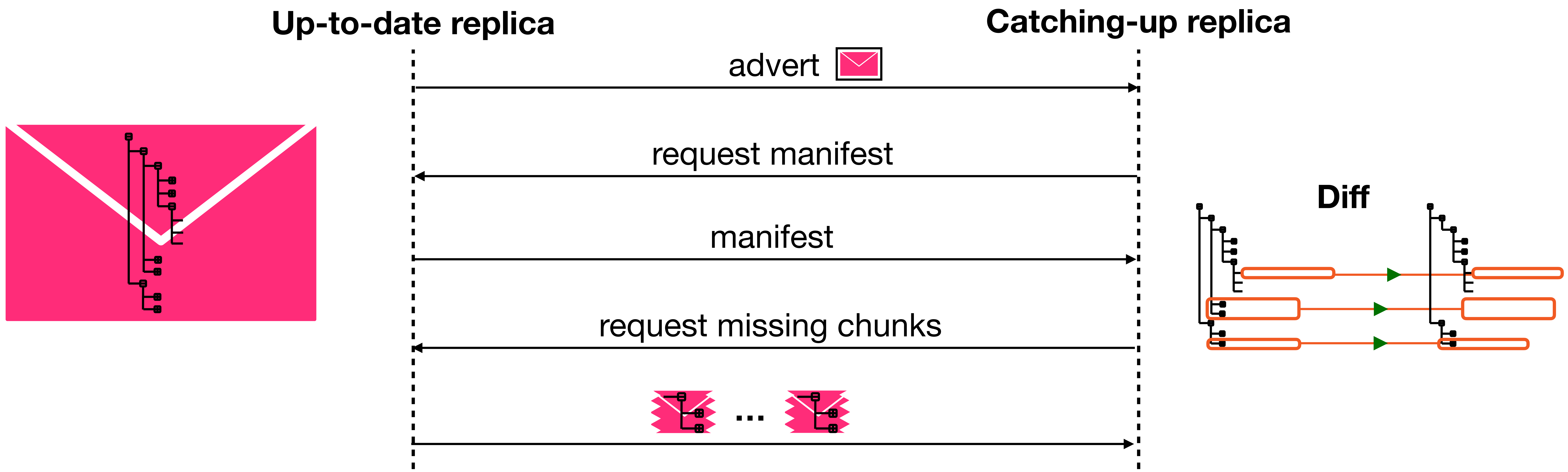
- Chunking mechanism
  - first request manifest with leaf/subtree hashes
  - then determine chunks to fetch (as opposed to always fetching all chunks)



Design Overview

Announce state as one big artifact, use tree structure to request chunks

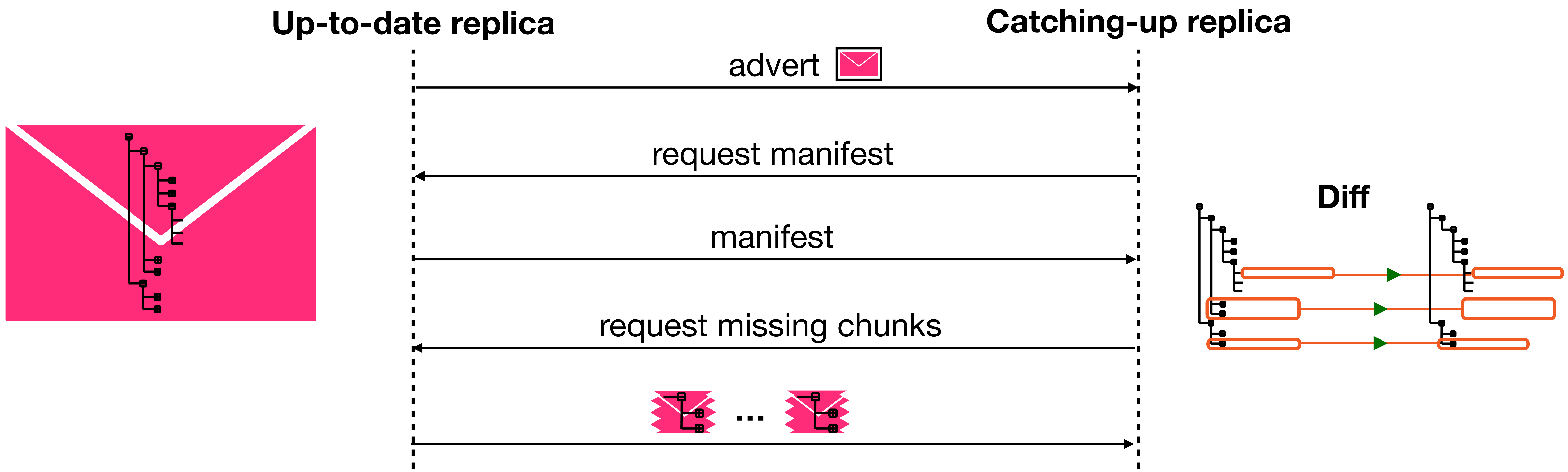
- Chunking mechanism
  - first request manifest with leaf/subtree hashes
  - then determine chunks to fetch (as opposed to always fetching all chunks)



Design Overview

Announce state as one big artifact, use tree structure to request chunks

- Chunking mechanism
  - first request manifest with leaf/subtree hashes
  - then determine chunks to fetch (as opposed to always fetching all chunks)
- Natural, efficient diff and de-duplication
  e.g., empty (all-zero) page transmitted at most once



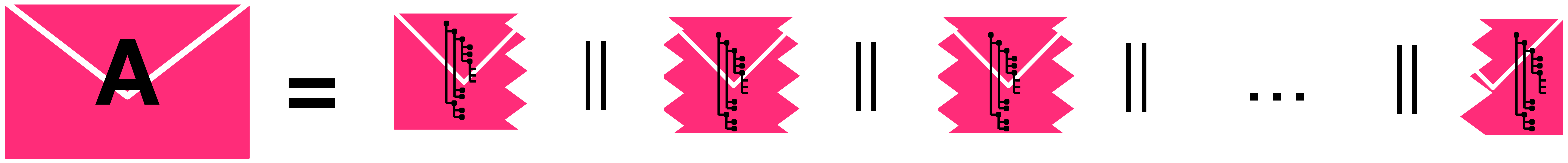
Summary

The Internet computer can

- **Run canister smart contracts**
- **Serve requests at web speed**
- **Despite byzantine nodes**

In particular, nodes can catch up quickly thanks to

- One public key per subnet, certified by NNS
- Catch Up Package containing a block with key info and checkpoint hash
- Chunking mechanism
  - first request manifest with leaf/subtree hashes
  - then determine chunks to fetch (as opposed to always fetching all chunks) from any peer



yvonneanne@dfinity.org