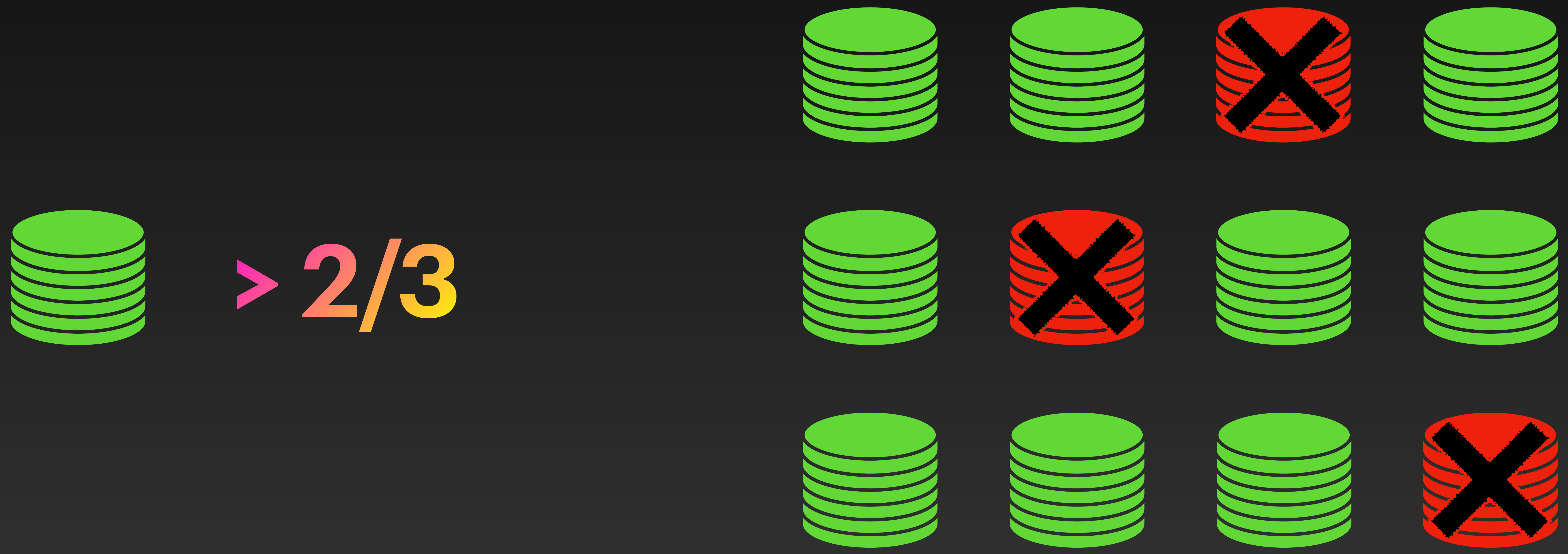
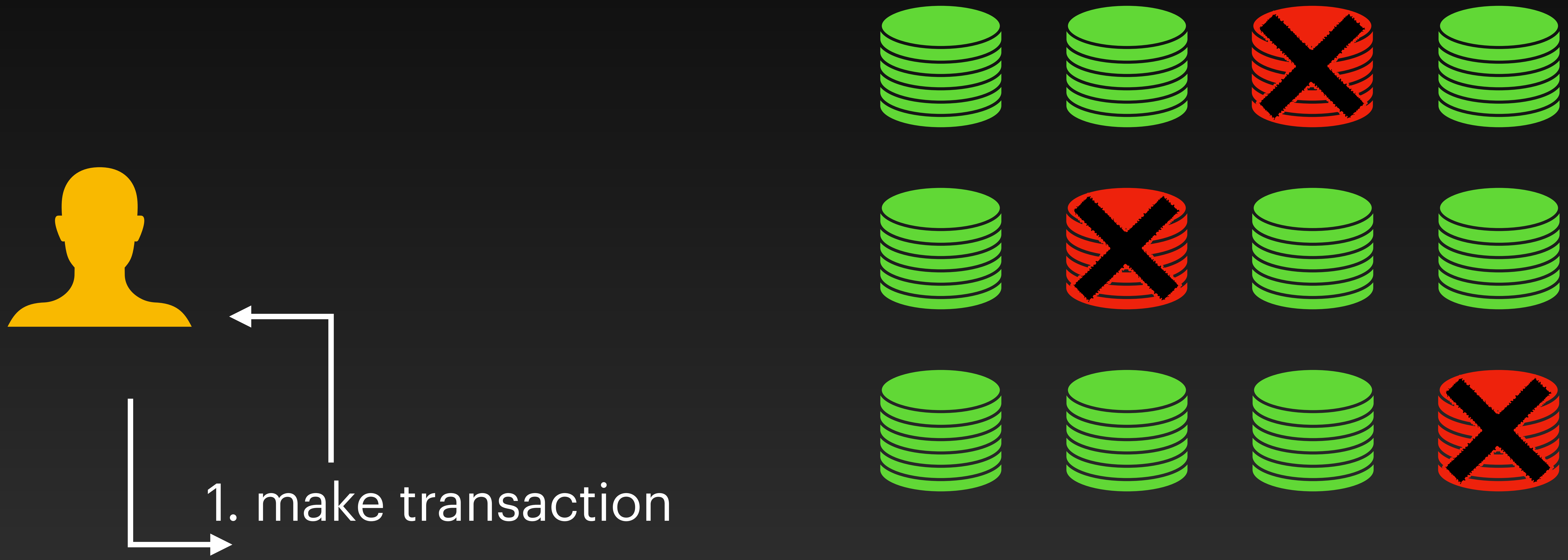
Modern Blockchains through the Lens of Network Security

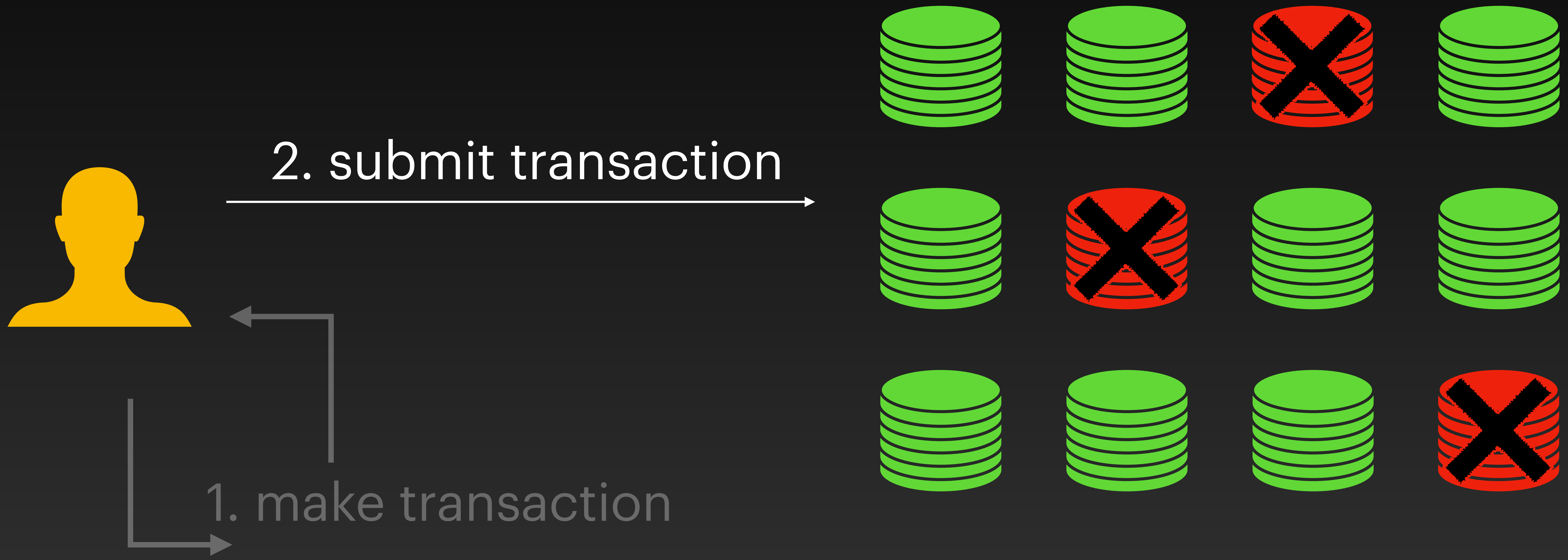
Alberto Sonnino

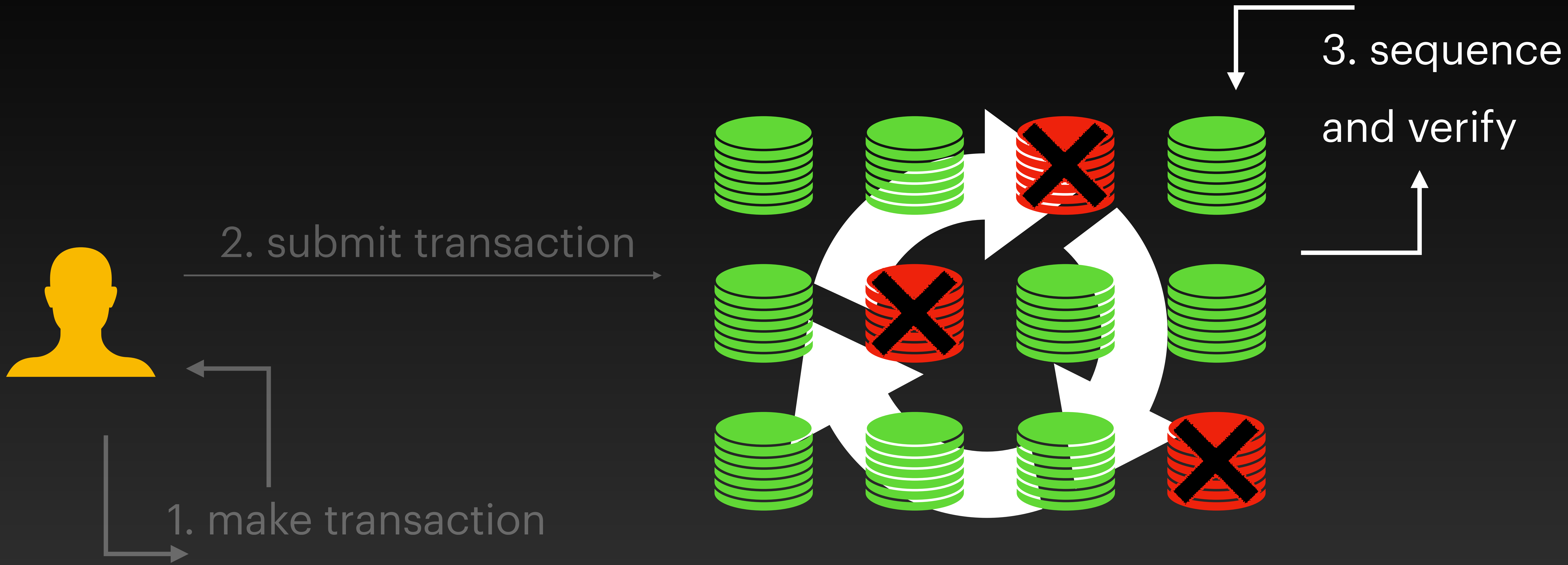
Byzantine Fault Tolerance

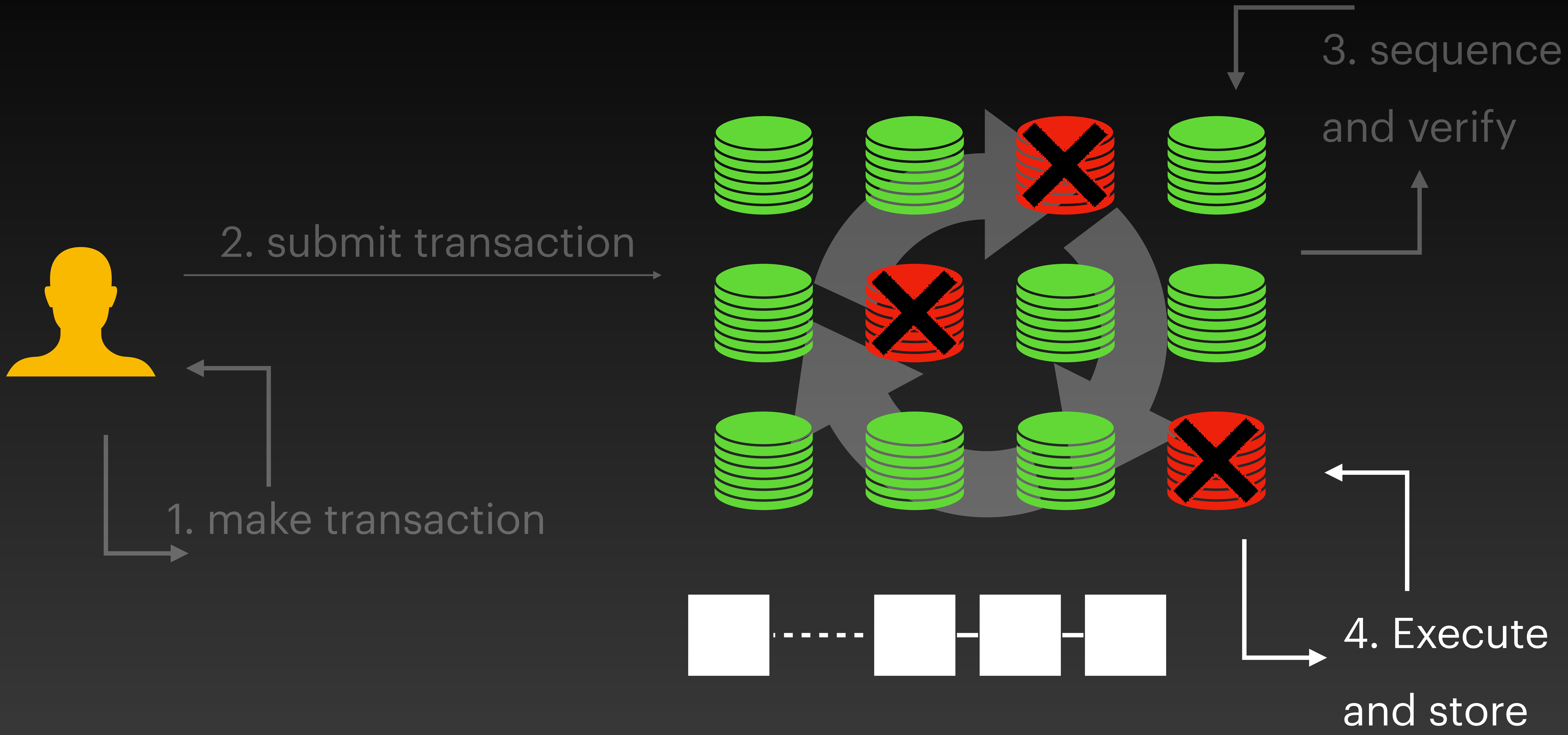
Byzantine Fault Tolerance




1. make transaction


2. submit transaction
1. make transaction


2. submit transaction
1. make transaction
3. sequence and verify


2. submit transaction
1. make transaction
3. sequence and verify
4. Execute and store

- **Distributed Systems**
  - But not like a DB running in my datacenter
  - Adversarial network and Byzantine adversaries
- **Systems Security**
  - Both network and systems security
  - Interaction between networked components
- **Programming Languages**
  - Execute the smart contract & ensure determinism
  - Solidity, Move
- **Cryptography**
  - Validators cannot use secrets to execute smart contracts
  - Anonymous credentials, ZK-proofs

- **Distributed Systems**
  - But not like a DB running in my datacenter
  - Adversarial network and Byzantine adversaries
- **Systems Security**
  - **Both network and systems security**
  - **Interaction between networked components**
- **Programming Languages**
  - Execute the smart contract & ensure determinism
  - Solidity, Move
- **Cryptography**
  - Validators cannot use secrets to execute smart contracts
  - Anonymous credentials, ZK-proofs

Security Properties

Safety

Undesirable things never happen

Liveness

Desirable things eventually happen

Adversary

#1 The Network: Worst possible schedule

Properties

- **Synchronous**: A message sent will be delivered before a maximum (known) delay.
- **Asynchronous**: A message sent will eventually be delivered at an arbitrary time before a maximum (unknown) delay.
- **Partial Synchronous**: the network is asynchronous but after some time it enters a period of synchrony.

Challenges

- Theoretical models: Need careful implementation to ensure we approximate them, e.g., retransmissions.
- Memory: Naive implementations use infinite buffers. Identify conditions after which retransmissions are not necessary and buffers can be freed.
- Asynchrony means the protocol should maintain properties for any re-ordering of message deliveries.
- Unknown delay means delay should be adaptive to ensure robustness.

Adversary

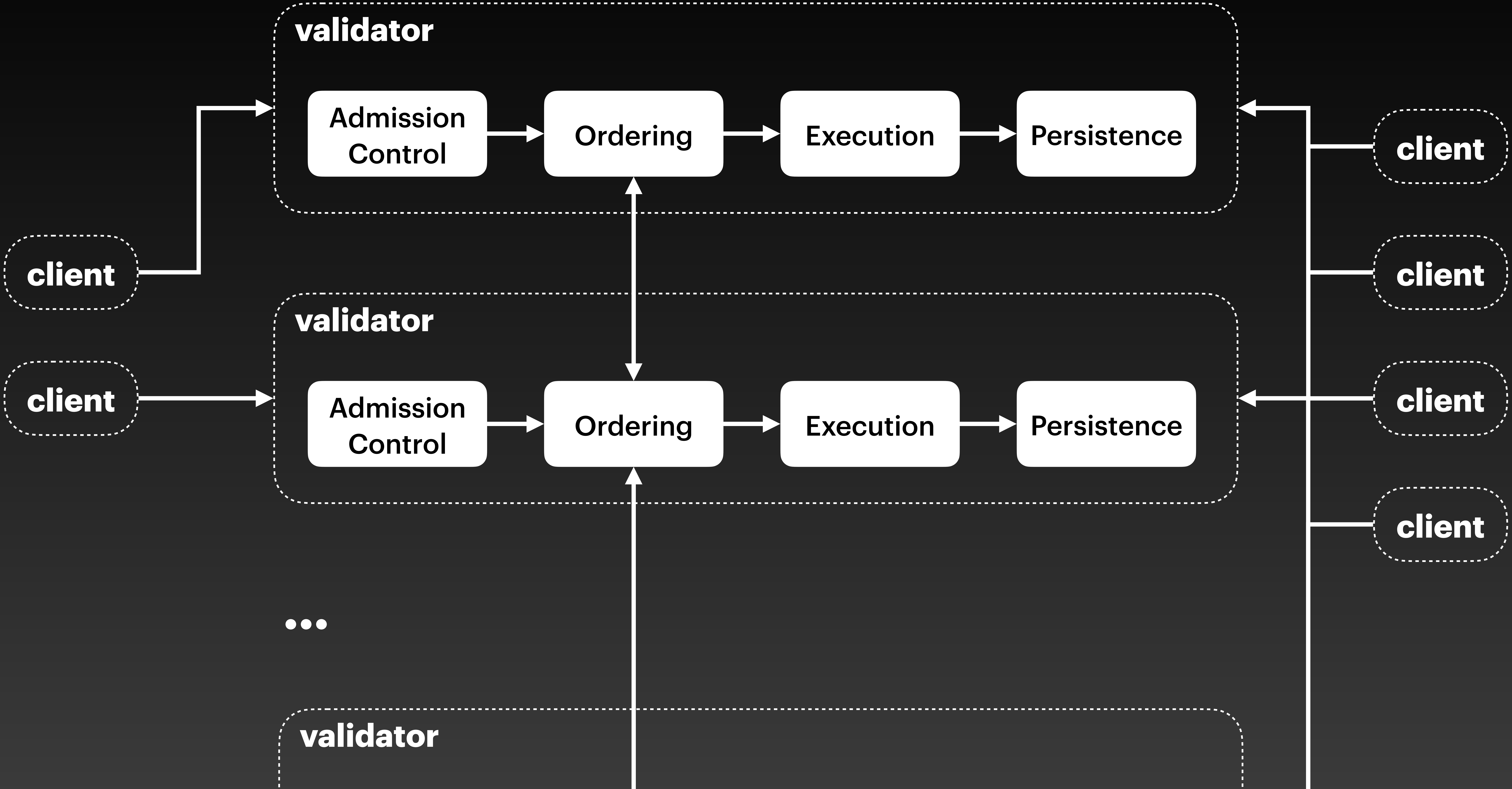
#2 Bad Nodes: Arbitrary behaviour

Properties

- **Correct / honest / good:** Will remain live and follow the protocol as specified by the designers of the system.
- **Byzantine:** will deviate arbitrarily from the protocol. May respond incorrectly or not at all.

Challenges

- **Crash & recover:** still a correct validators with very high latency. Need persistence to ensure this
- **Rational:** honest validators may have some discretion. They may use it to maximise profit


validator
Admission Control
Ordering
Execution
Persistence
client
client
validator
Admission Control
Ordering
Execution
Persistence
client
client
client
client
•••
validator

Network Security

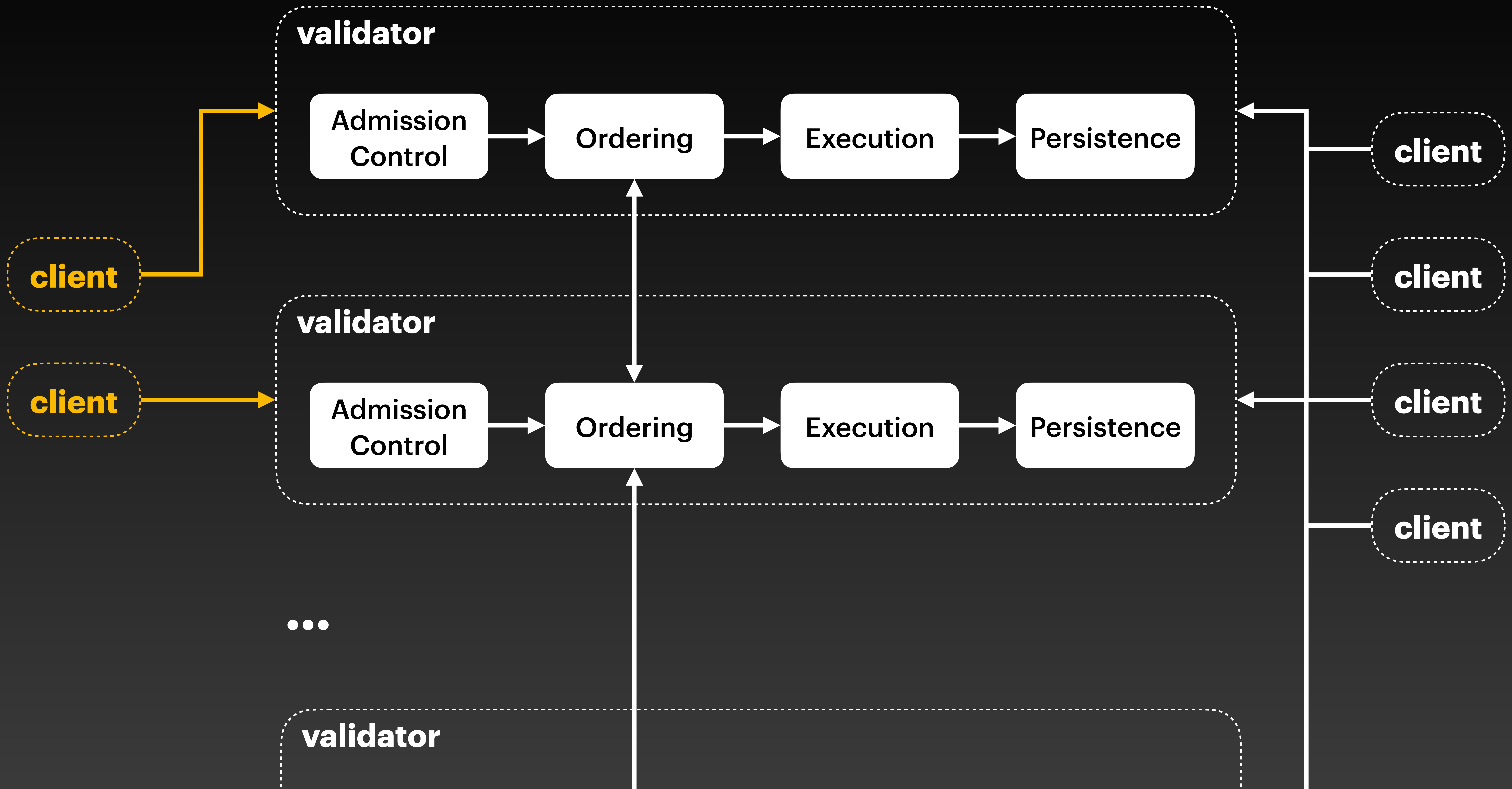
Challenge #1: Nodes

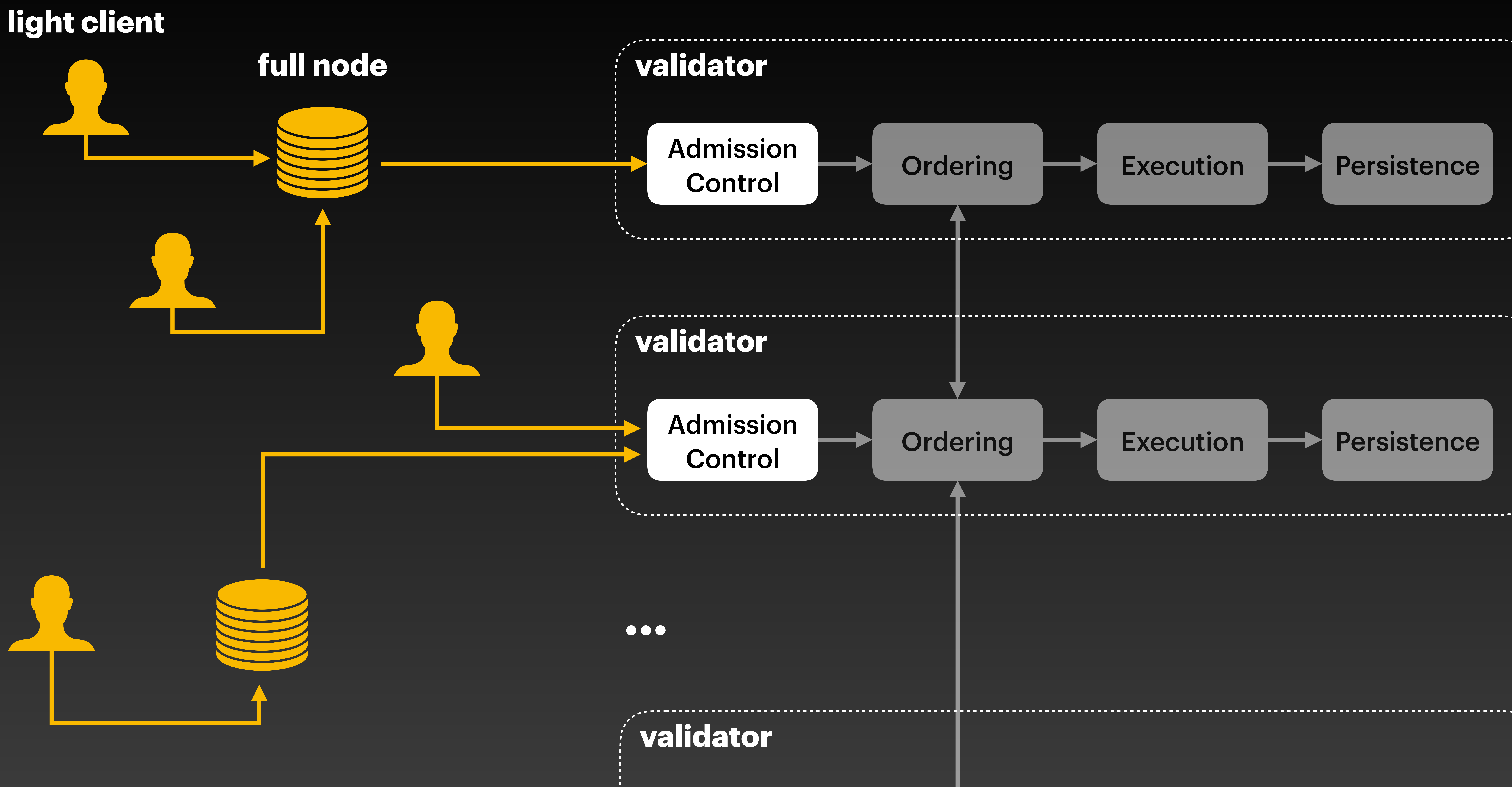
- **Validators are exposed (not in datacenter no on beefy machines)**

Network Security

Challenge #1: Nodes

- Validators are exposed (not in datacenter no on beefy machines)
- **Highly dynamic set of nodes**


validator
Admission Control
Ordering
Execution
Persistence
client
client
validator
Admission Control
Ordering
Execution
Persistence
•••
validator
client
client
client
client


light client
full node
validator
Admission Control
Ordering
Execution
Persistence
validator
Admission Control
Ordering
Execution
Persistence
...
validator

Network Security

Challenge #2: Clients

- **Different types of target links: clients-validator and validator-validator**

Network Security

Challenge #2: Clients

- Different types of target links: clients-validator and validator-validator
- **Highly dynamic clients**

Network Security

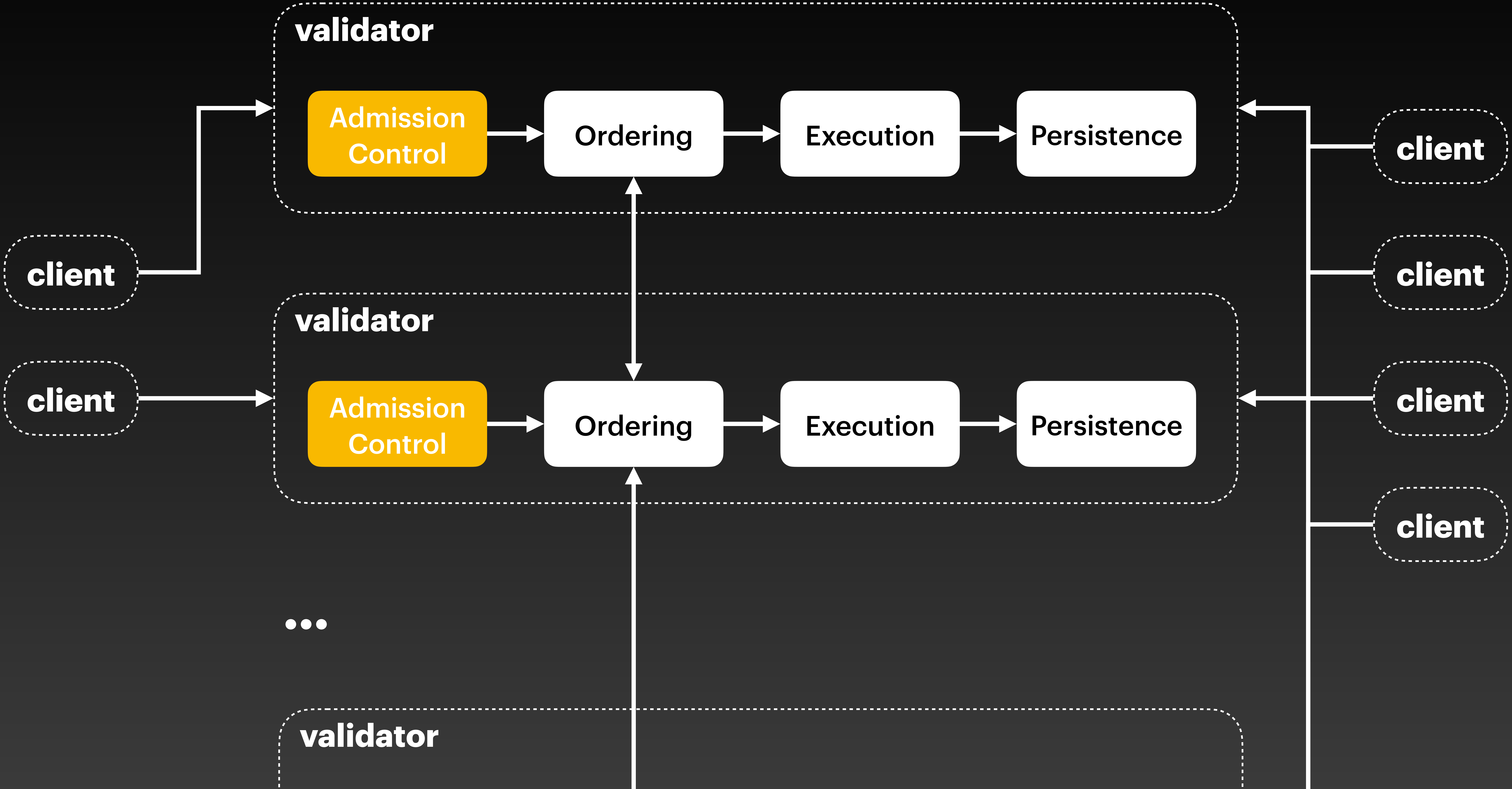
Challenge #2: Clients

- Different types of target links: clients-validator and validator-validator
- Highly dynamic clients
- **Clients have no fixed identity**

Network Security

Challenge #2: Clients

- Different types of target links: clients-validator and validator-validator
- Highly dynamic clients
- Clients have no fixed identity
- **Unclear validator selection algorithm**


validator
Admission Control
Ordering
Execution
Persistence
client
client
validator
Admission Control
Ordering
Execution
Persistence
client
client
client
client
•••
validator

Objects:

- Unique ID
- Version number
- Ownership Information
- Type

Transaction's content

Package, function	Coin::Send
Object Inputs	Alice's account
Arguments	Bob's account, Balance=5
Gas Information	0.001, max=0.005
Signature	

Example Transaction

T1

Inputs: O1, O2, O3

Output: Mutate O1, Transfer O2, Delete O3, Create O4

Example Transaction

T1

Inputs: O1, O2, O3

Output: Mutate O1, Transfer O2, Delete O3, Create O4

e.g., Mutate a coin to pay for gas

e.g., Delete a disease caught by my warrior

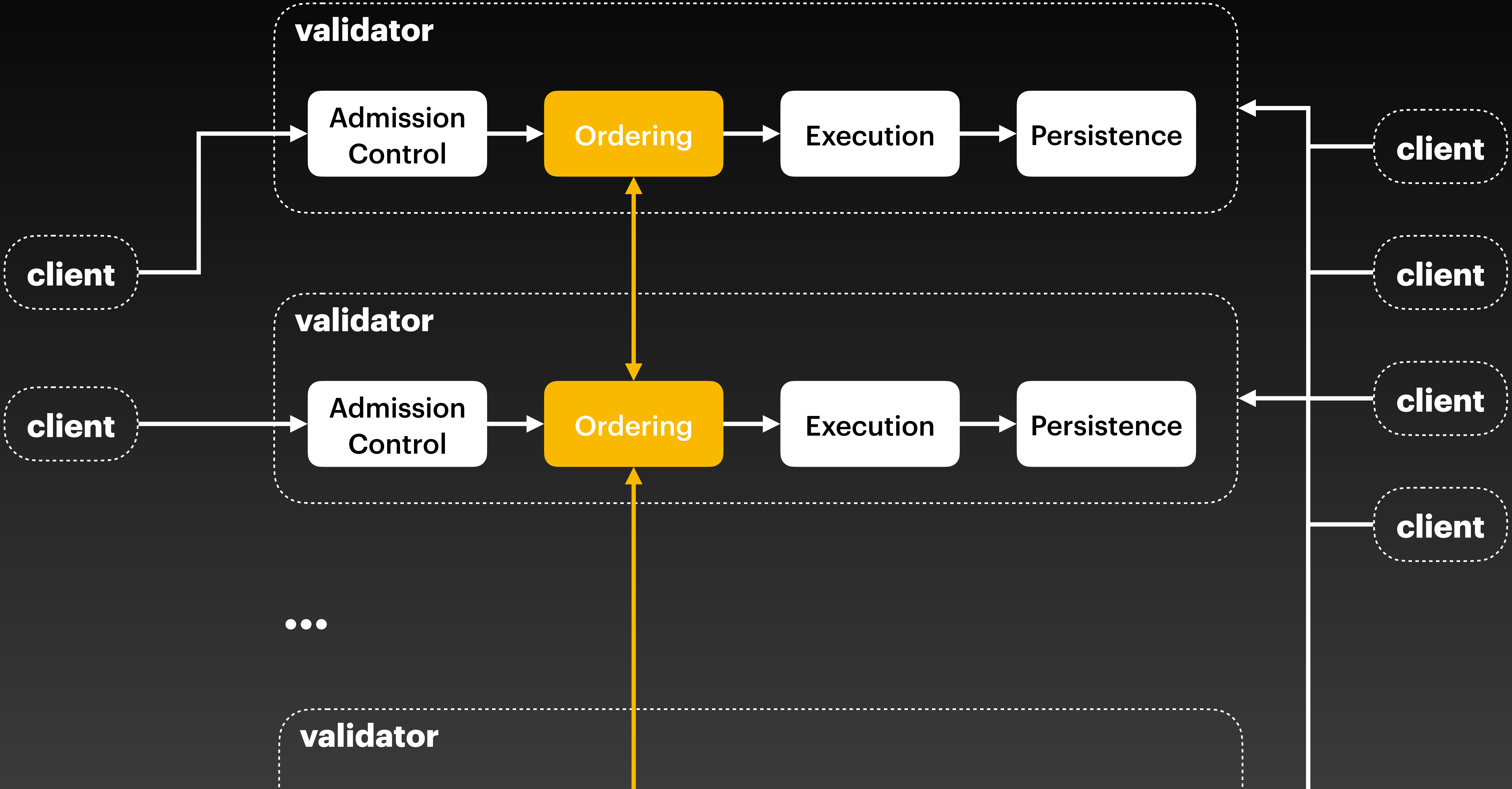
e.g., Transfer my warrior to friend

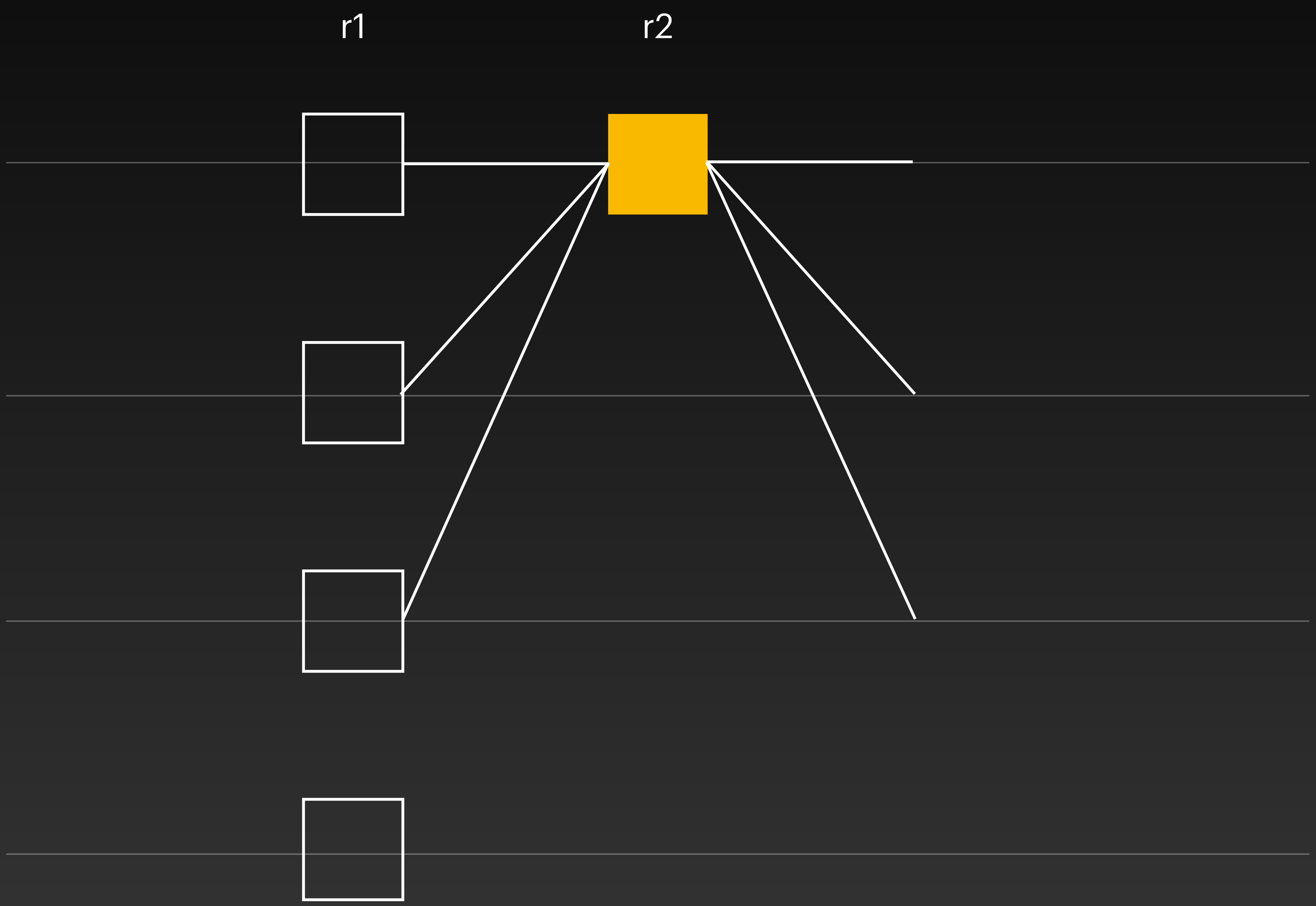
e.g., Be rewarded with a mystery gift

Network Security

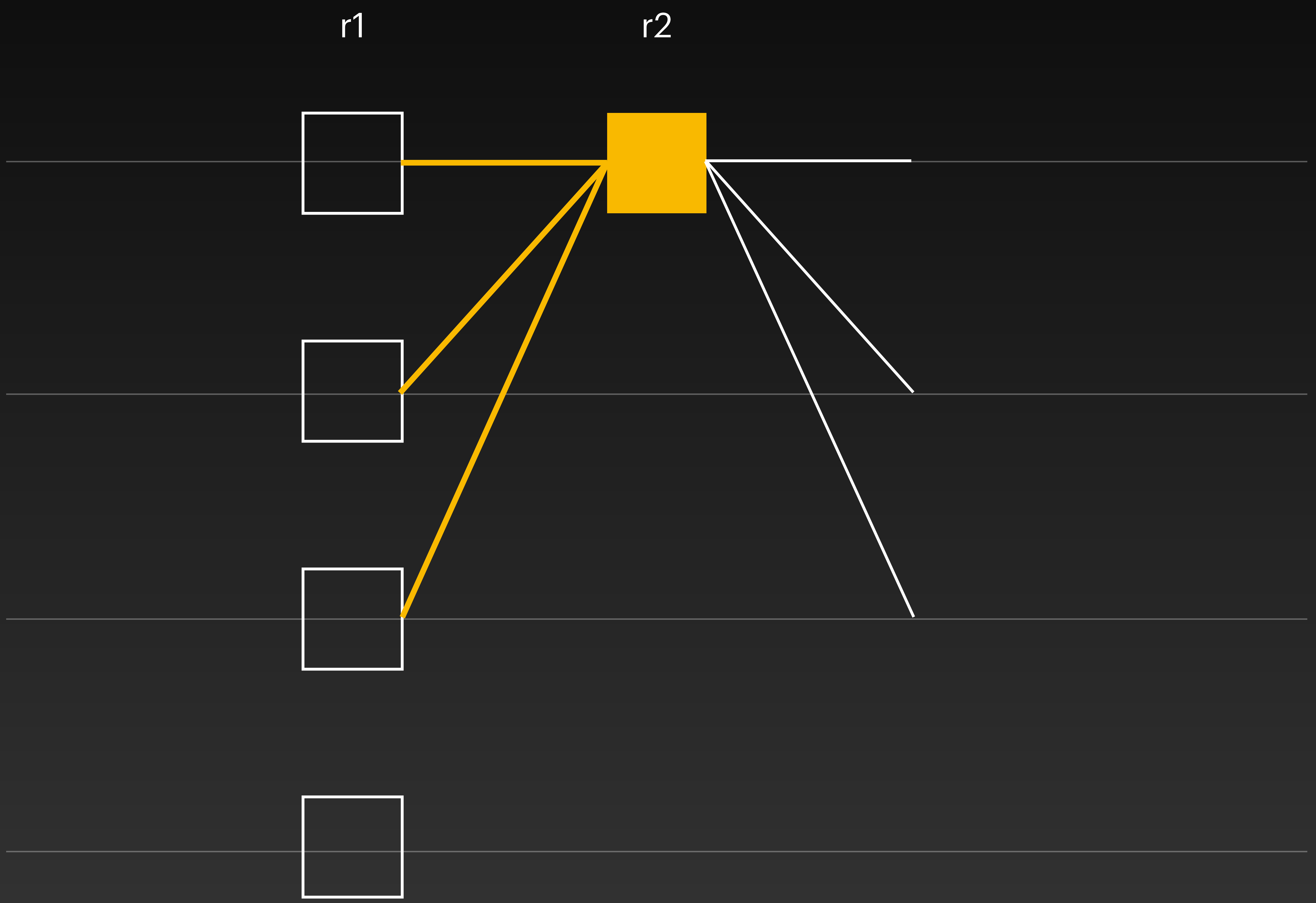
Challenge #3: Admission Control

- **No established way to run pre-checks on input transactions**

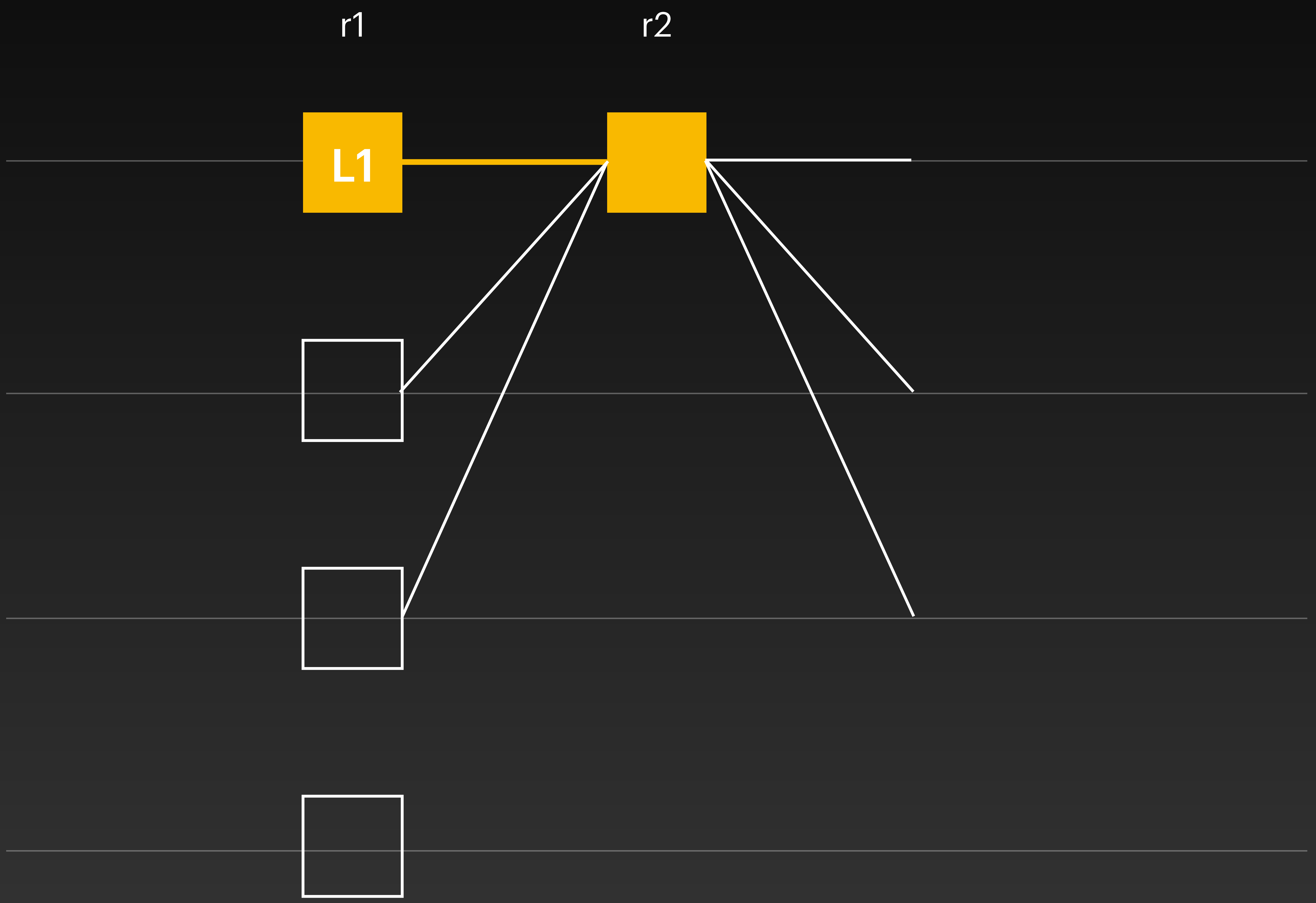

validator
Admission Control
Ordering
Execution
Persistence
client
client
client
client
client
validator
Admission Control
Ordering
Execution
Persistence
client
•••
validator



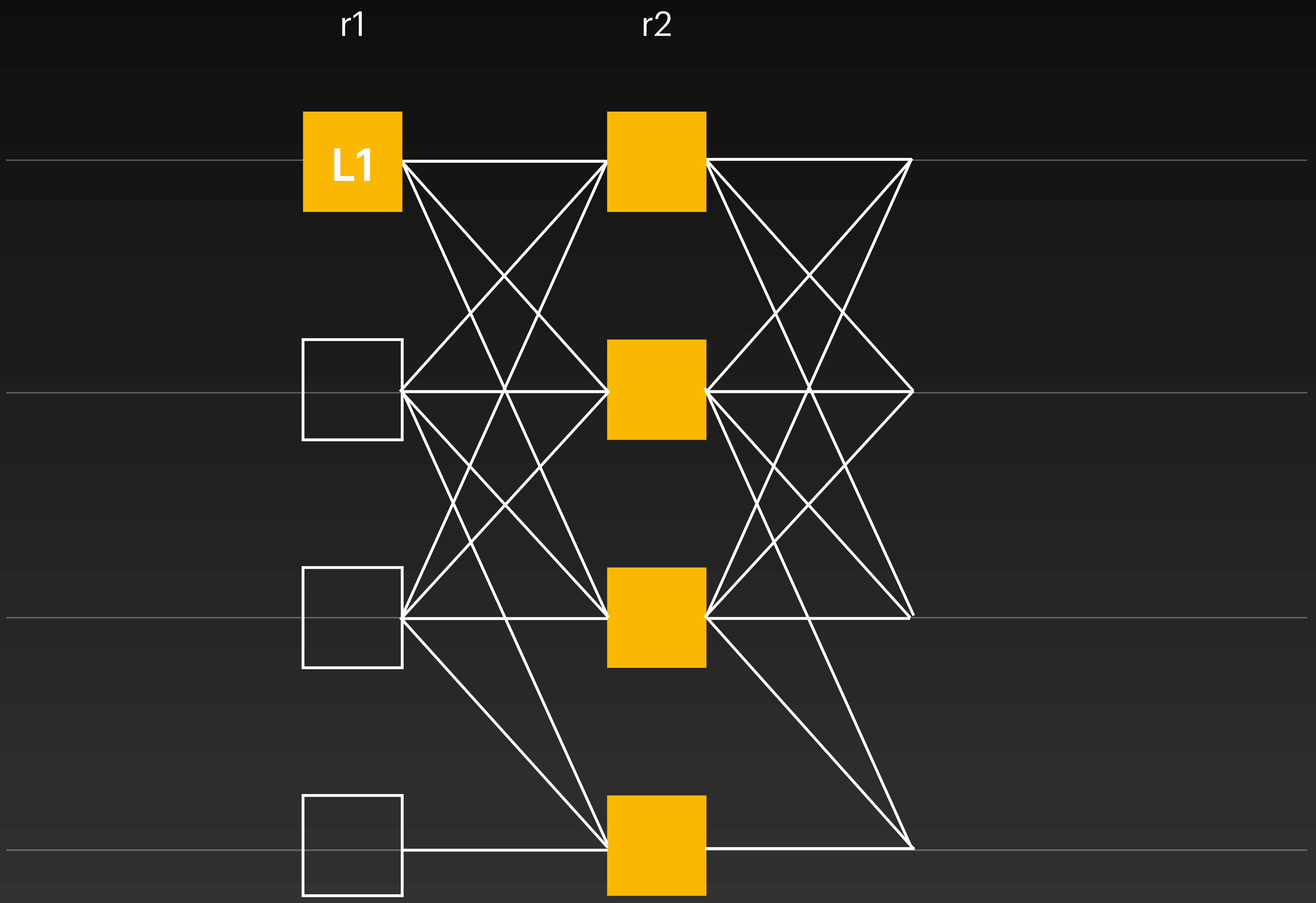
- Round number
- Author
- Payload (transactions)
- Signature



- Link to previous blocks



- Wait for the leader

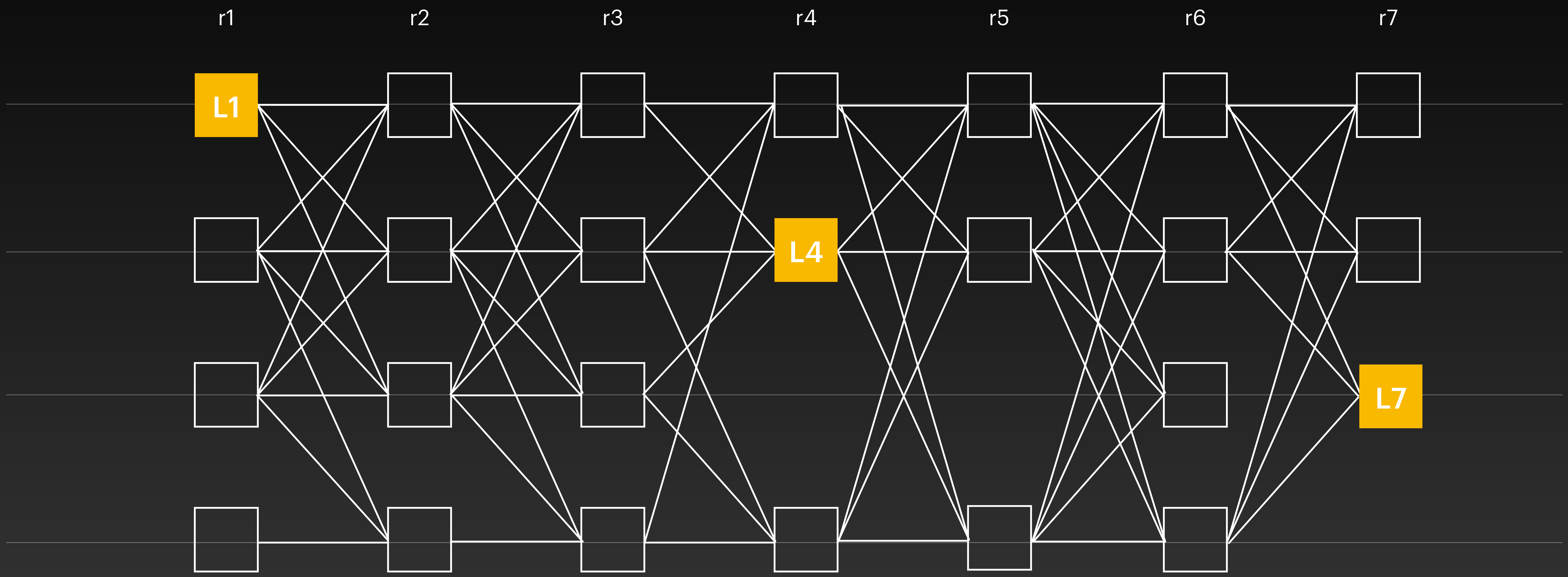


- All validators run in parallel

Network Security

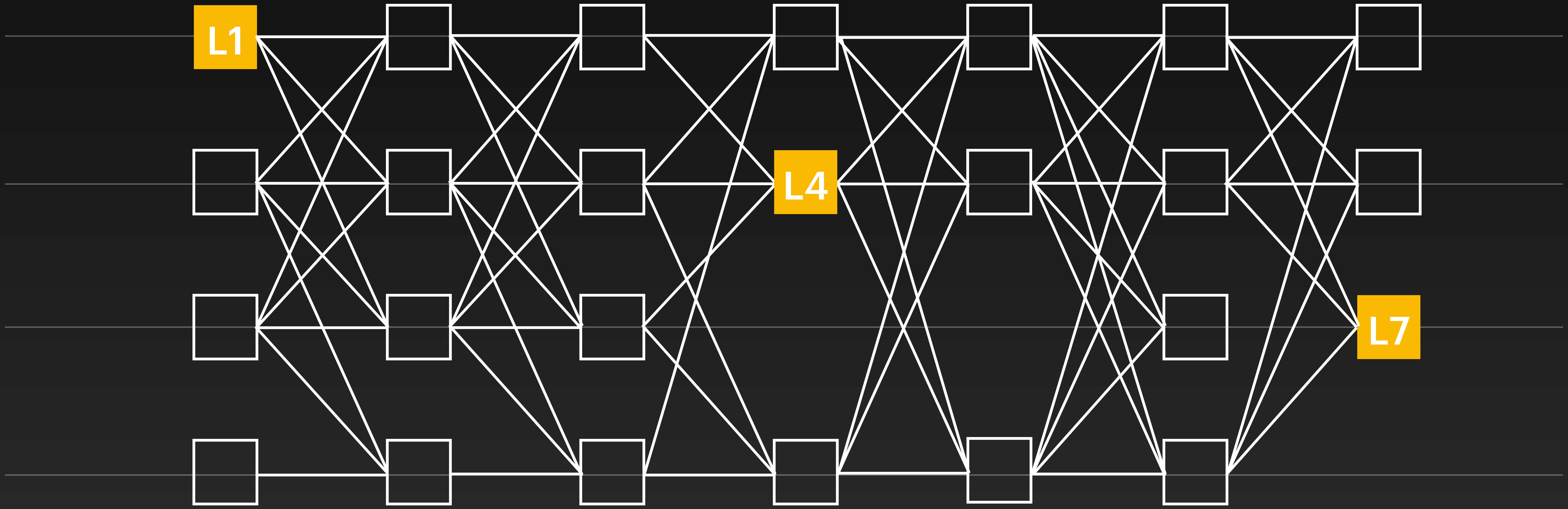
Challenge #4: Ordering

- **How to find the best path to send the block to another node?**


r1
r2
r3
r4
r5
r6
r7
L1
L4
L7

End Goal

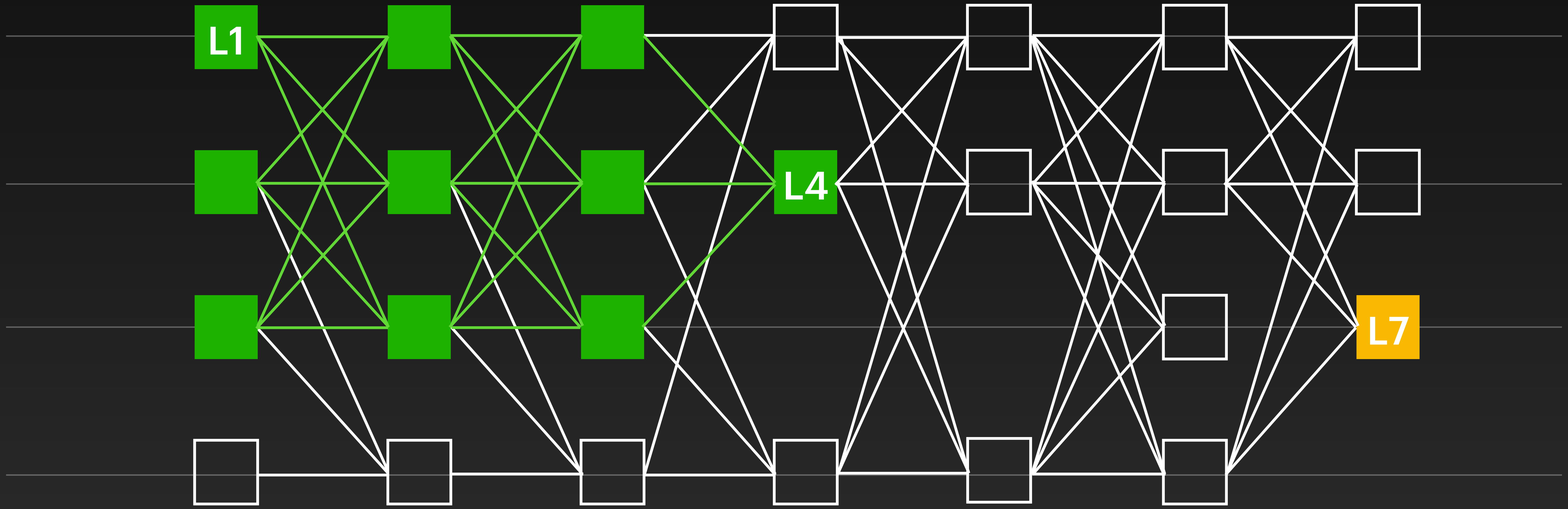
Ordering leaders



- We focus on ordering leaders: L1 L4 L7

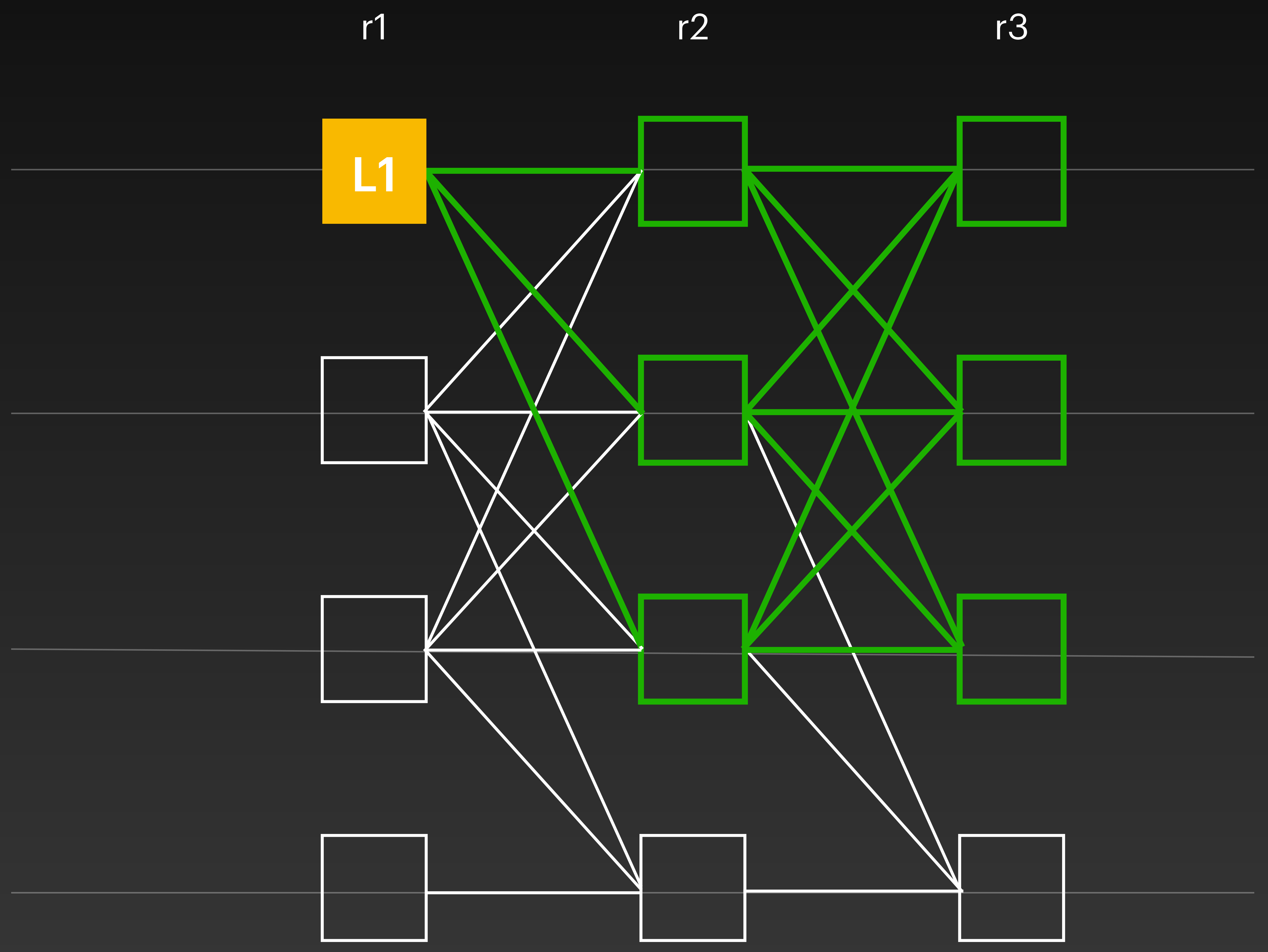
End Goal

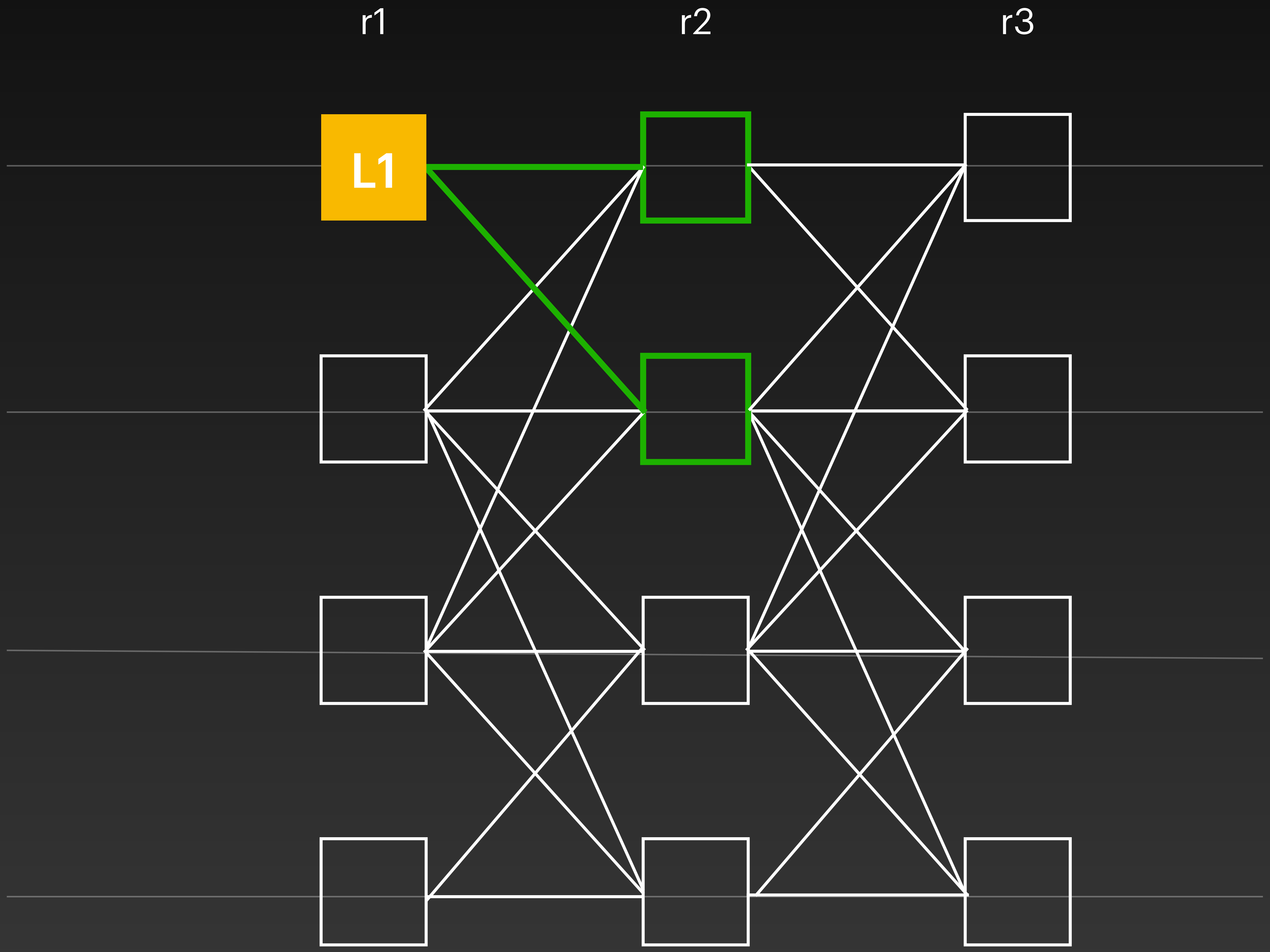
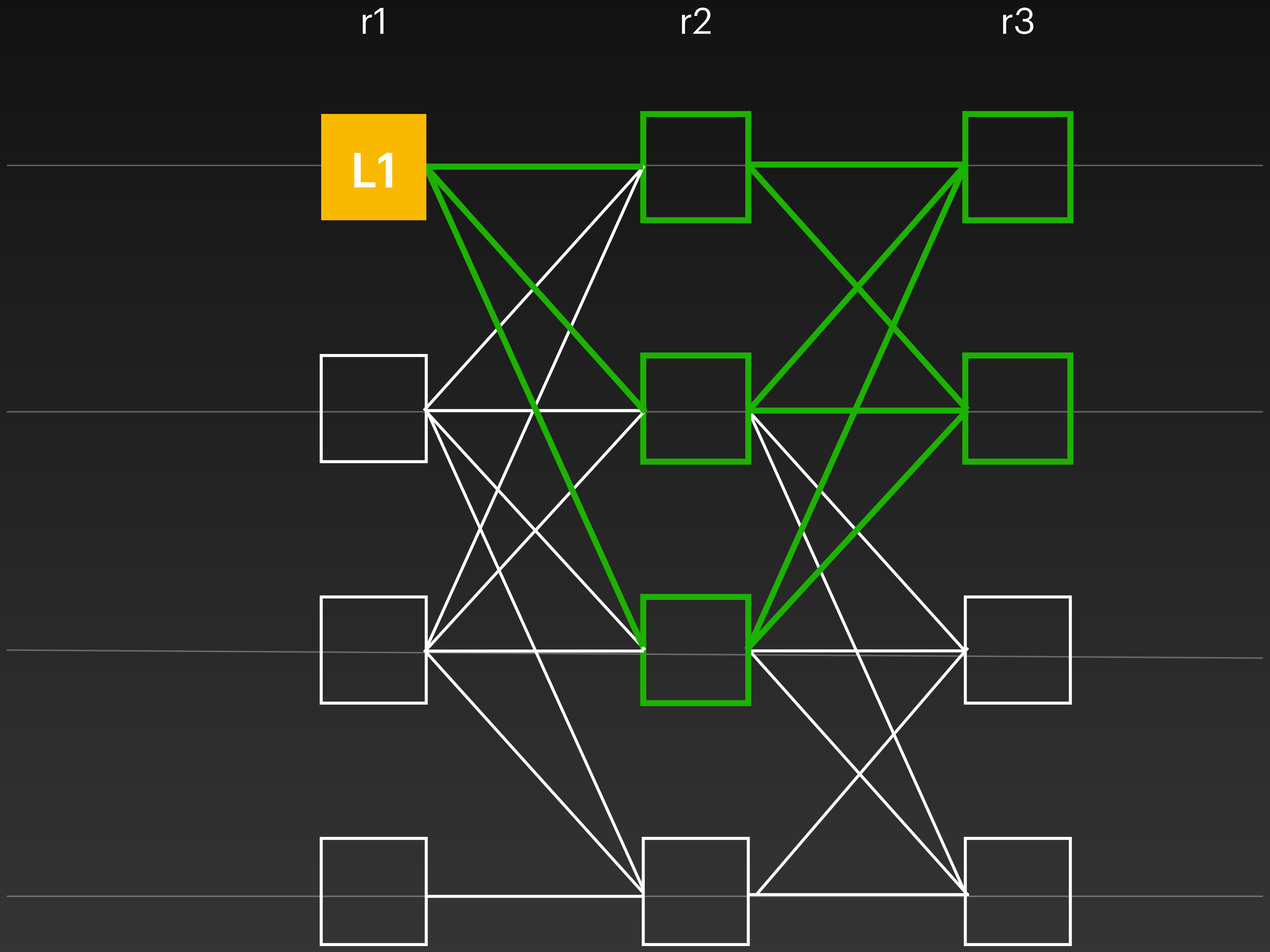
Ordering leaders



- We focus on ordering leaders: L1 L4 L7
- Linearising the sub-DAG is simple

How is it done?



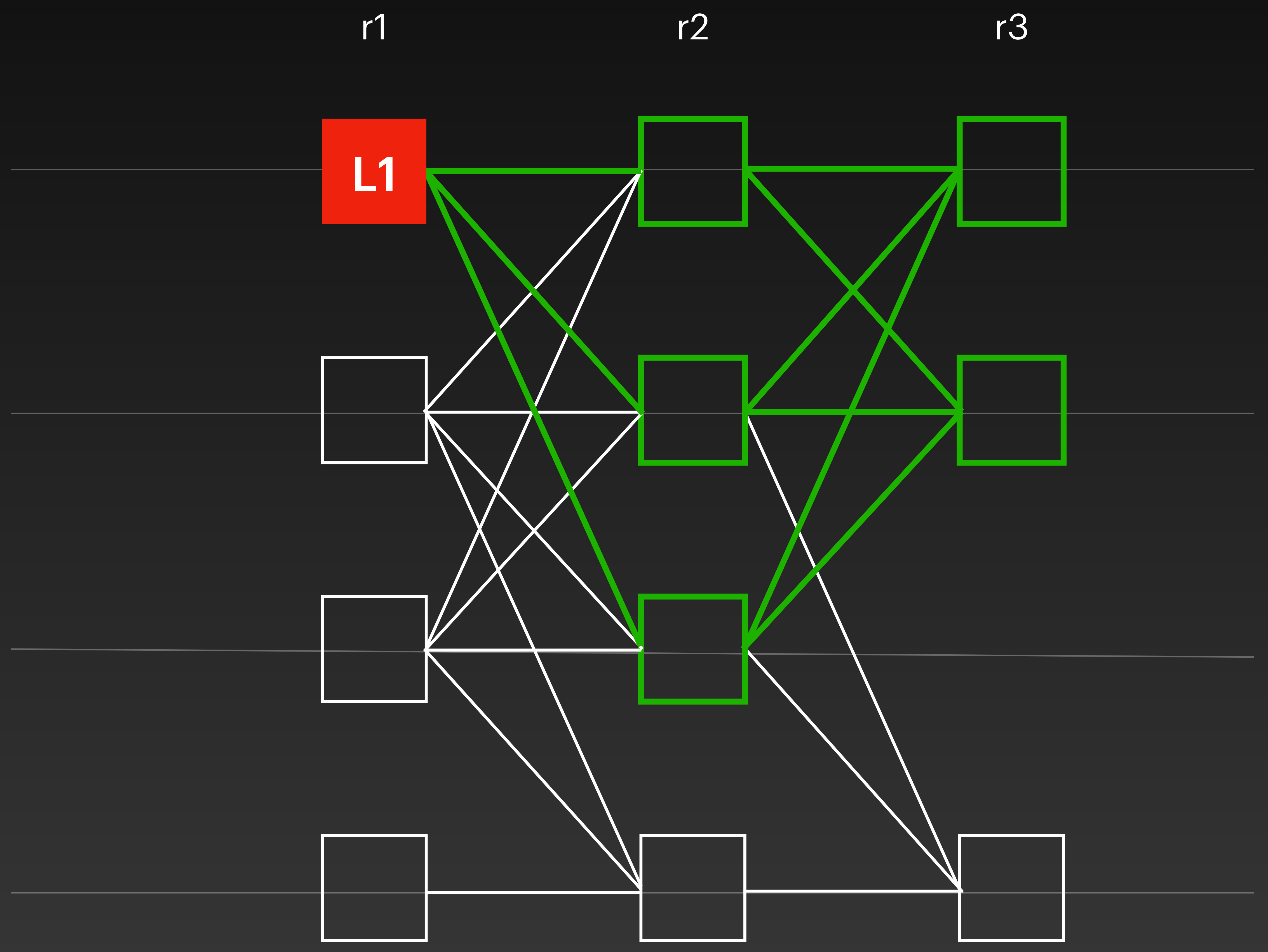

r1
r2
r3
L1

r1
r2
r3
L1

Network Security

Challenge #4: Ordering

- How to find the best path to send the block to another node?
- **DoS against the leader are particularly effective**

Message not received in order?



- Bad leader?
- Or bad network?

Network Security

Challenge #4: Ordering

- How to find the best path to send the block to another node?
- DoS against the leader are particularly effective
- **Reordering messages causes massive slowdowns**

Network Security

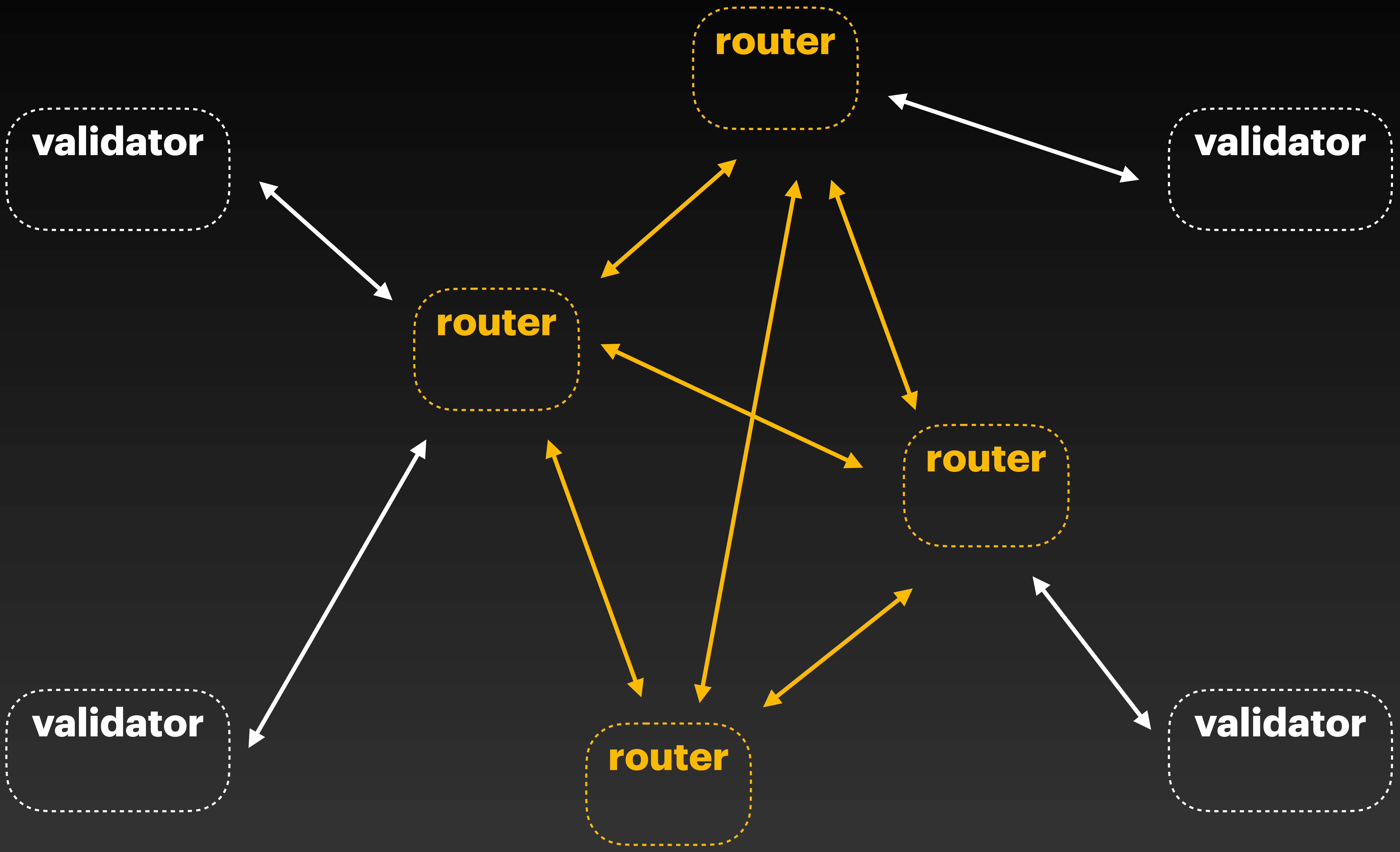
Challenge #4: Ordering

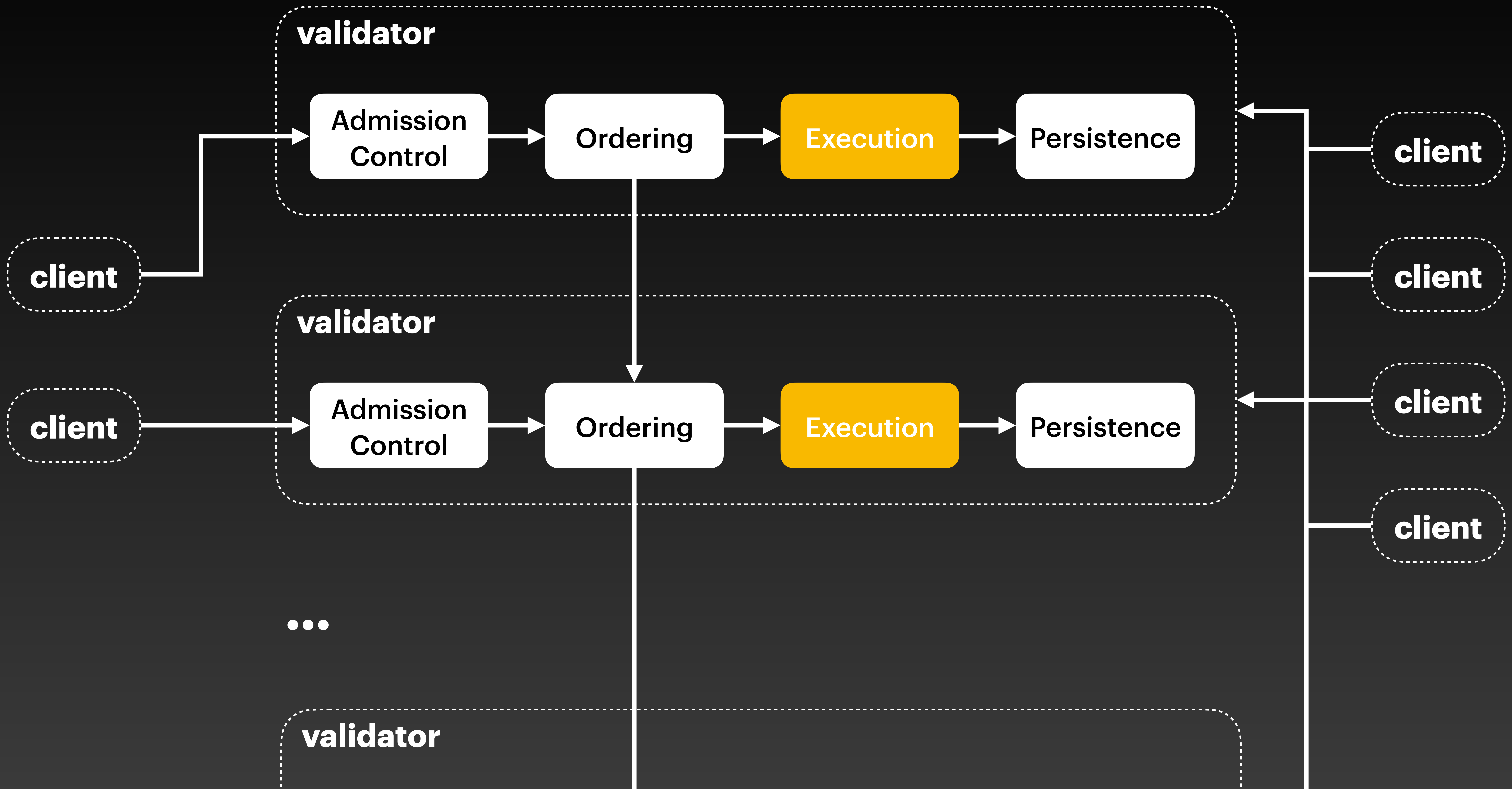
- How to find the best path to send the block to another node?
- DoS against the leader are particularly effective
- Reordering messages causes massive slowdowns
- **Nodes don’t know whether they are connected to a malicious node**

Network Security

Challenge #4: Ordering

- How to find the best path to send the block to another node?
- DoS against the leader are particularly effective
- Reordering messages causes massive slowdowns
- Nodes don’t know whether they are connected to a malicious node
- **Bad nodes have access to insider information (committee addresses)**


router
validator
validator
router
router
validator
router
validator


validator
Admission Control
Ordering
Execution
Persistence
client
client
validator
Admission Control
Ordering
Execution
Persistence
client
client
client
client
client
...
validator

Example Transaction

T1

Inputs: O1, O2, O3

Output: Mutate O1, Transfer O2, Delete O3, Create O4

Check transaction, assign locks

O1	Version = 10
	Owner = Alice

O2	Version = 27
	Owner = Alice

O3	Version = 1001
	Owner = Alice

Checks

Input objects exist

Function call details

Signature of Alice

Execute in parallel

O1

Version = 11

Owner = X

O2

Version = 28

Owner = Bob

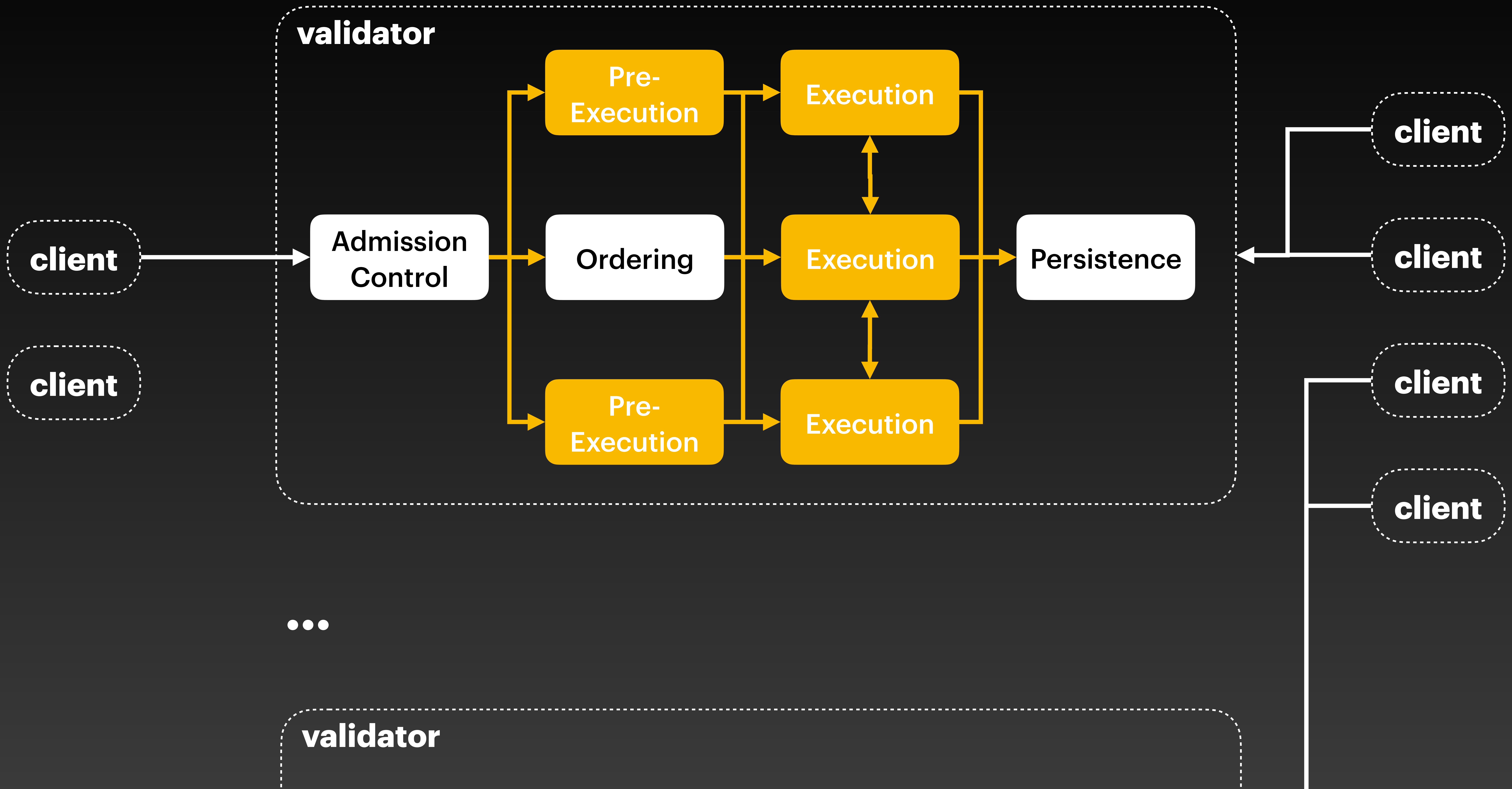
O4

Version = 1

Owner = Alice

Execute T1

- O1 mutated
- O2 transferred
- O3 deleted
- O4 created


validator
Pre-Execution
Execution
client
client
Admission Control
Ordering
Execution
Persistence
client
client
Pre-Execution
Execution
client
client
•••
validator

Network Security

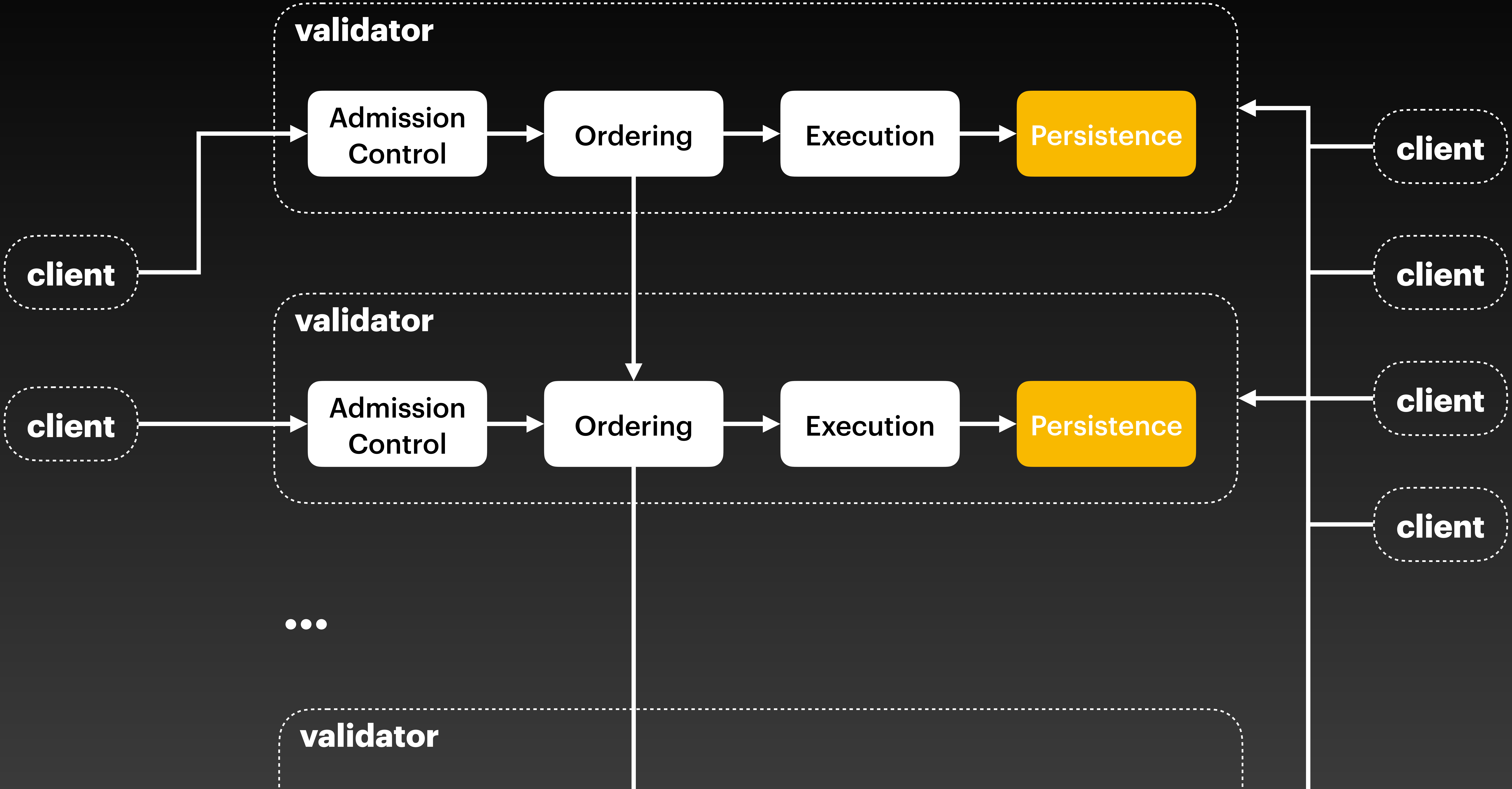
Challenge #5: Execution

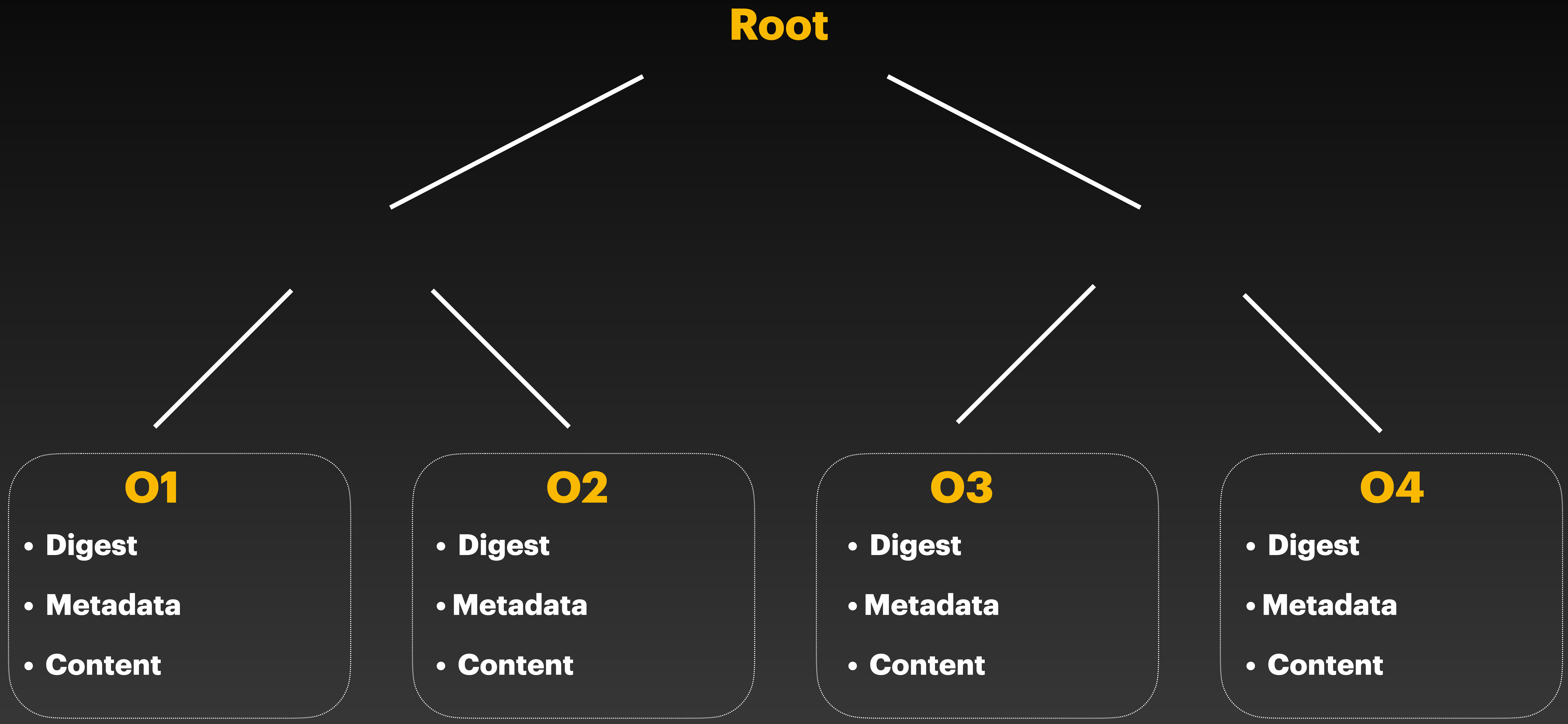
- **Intra-datacenter connections but on low power machines**

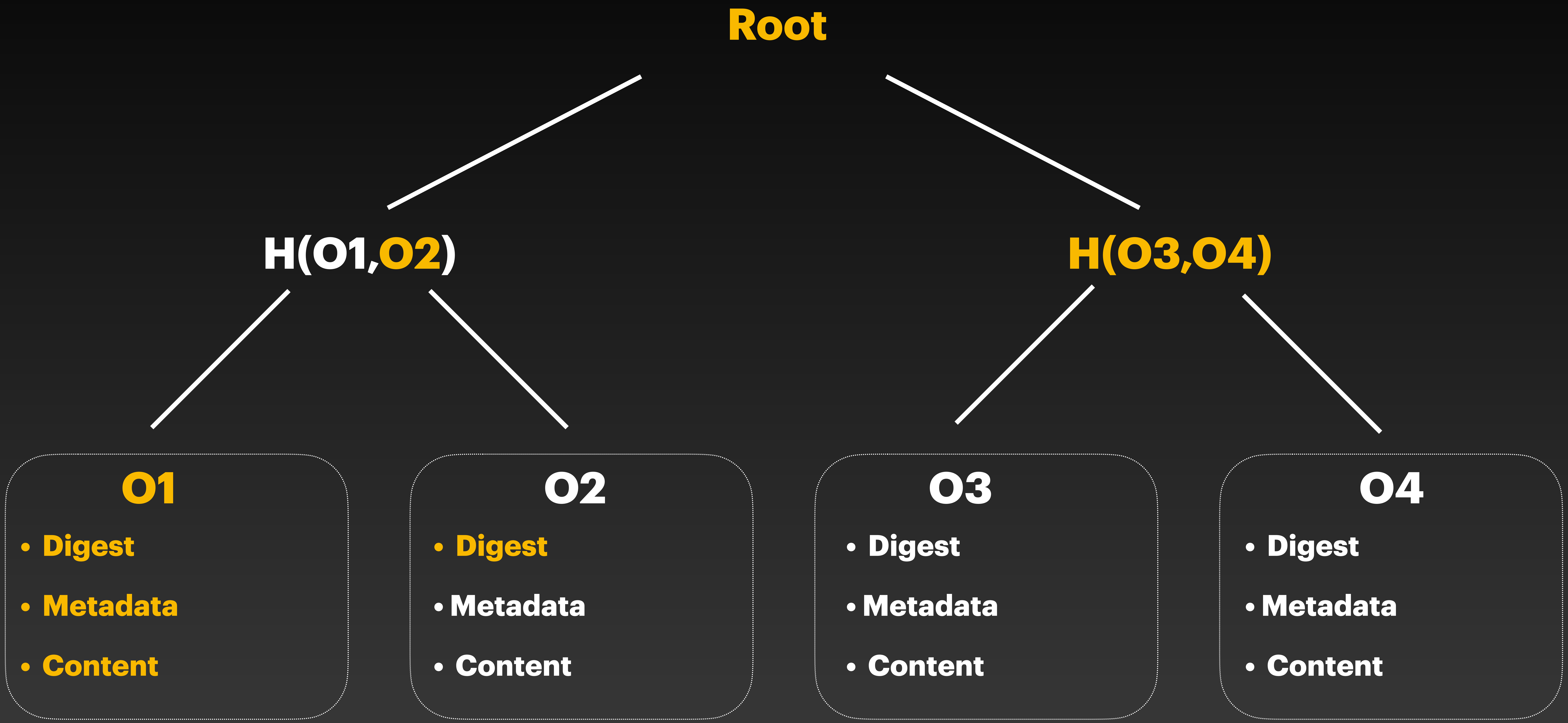
Network Security

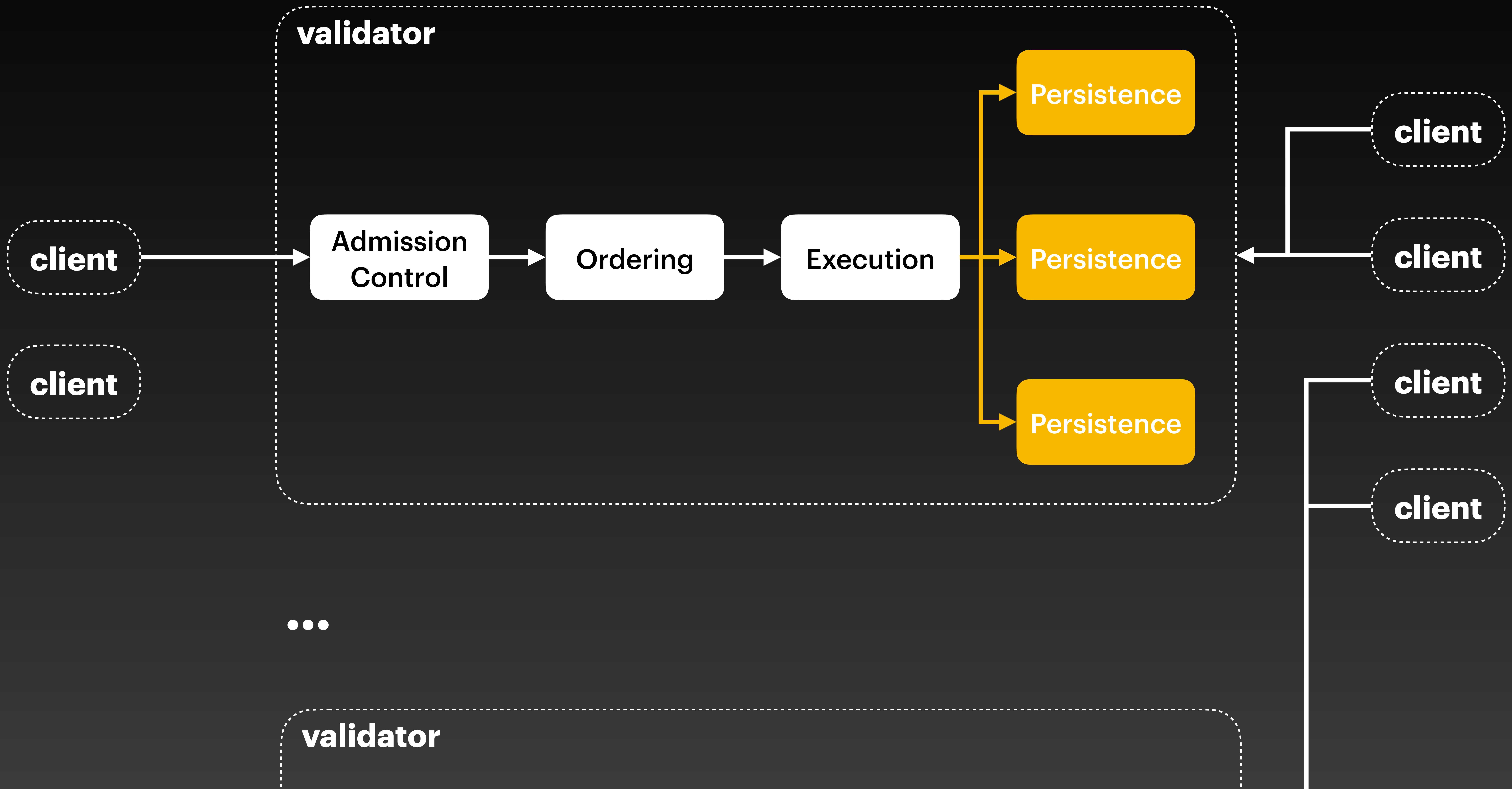
Challenge #5: Execution

- Intra-datacenter connections but on low power machines
- **Load drastically varies: need elasticity**


validator
Admission Control
Ordering
Execution
Persistence
client
validator
Admission Control
Ordering
Execution
Persistence
client
client
client
client
client
client
•••
validator


Root
01
• Digest
• Metadata
• Content
02
• Digest
• Metadata
• Content
03
• Digest
• Metadata
• Content
04
• Digest
• Metadata
• Content

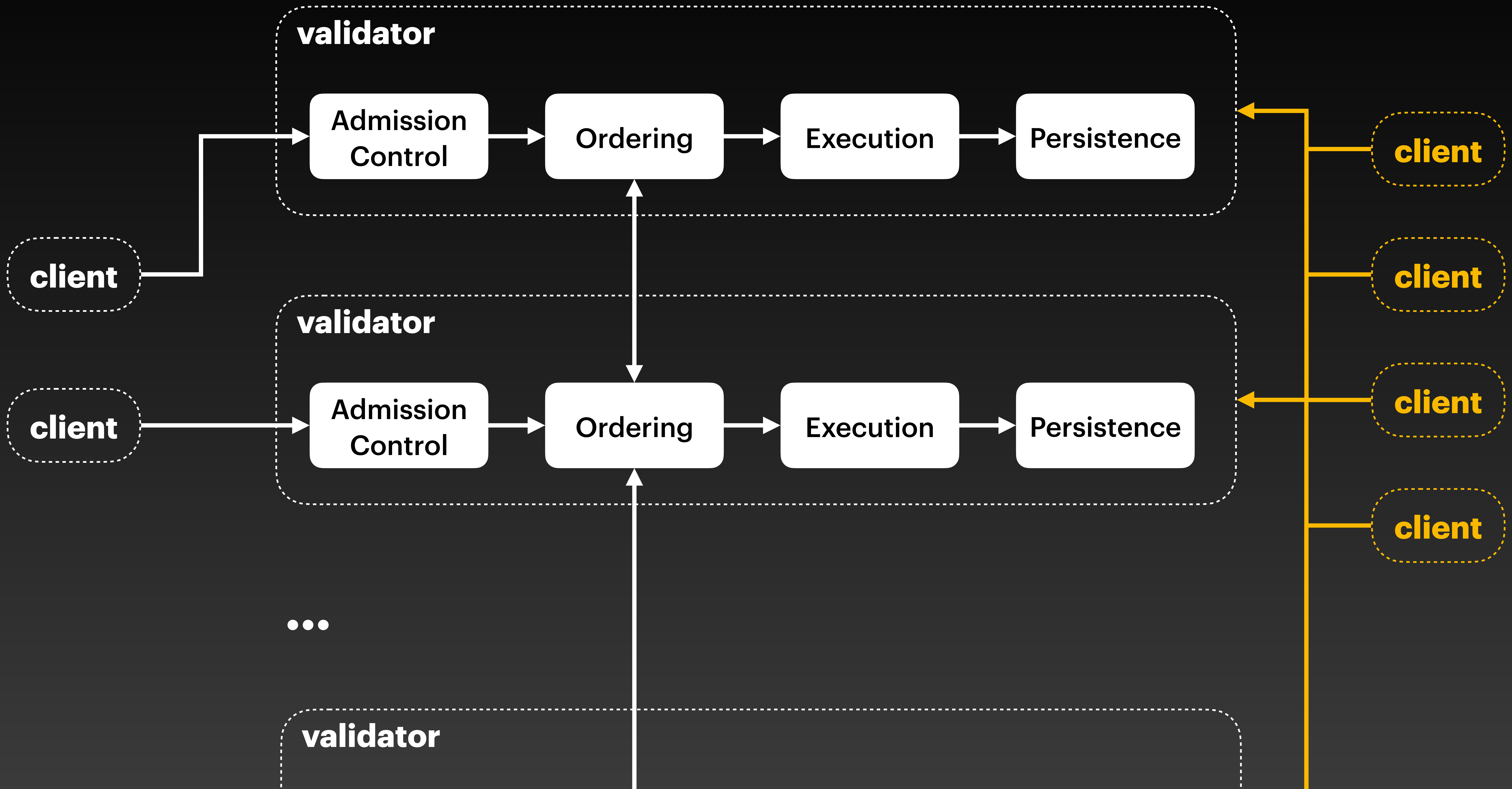

Root
H(O1,O2)
H(O3,O4)
O1
• Digest
• Metadata
• Content
O2
• Digest
• Metadata
• Content
O3
• Digest
• Metadata
• Content
O4
• Digest
• Metadata
• Content


validator
client
client
Admission Control
Ordering
Execution
Persistence
Persistence
Persistence
client
client
client
client
•••
validator

Network Security

Challenge #6: Persistence

- **Need low-latency networking to distribute the tree creation**


validator
Admission Control
Ordering
Execution
Persistence
client
client
validator
Admission Control
Ordering
Execution
Persistence
client
client
client
client
...
validator

validator
Admission Control
Ordering
Execution
Persistence
full node
validator
Admission Control
Ordering
Execution
Persistence
light client
•••
validator

Network Security

Challenge #7: Reads

- **Potentially very large number of readers (>400)**

Network Security

Challenge #7: Reads

- Potentially very large number of readers (>400)
- **Unpredictable, may read arbitrary data**

Network Security

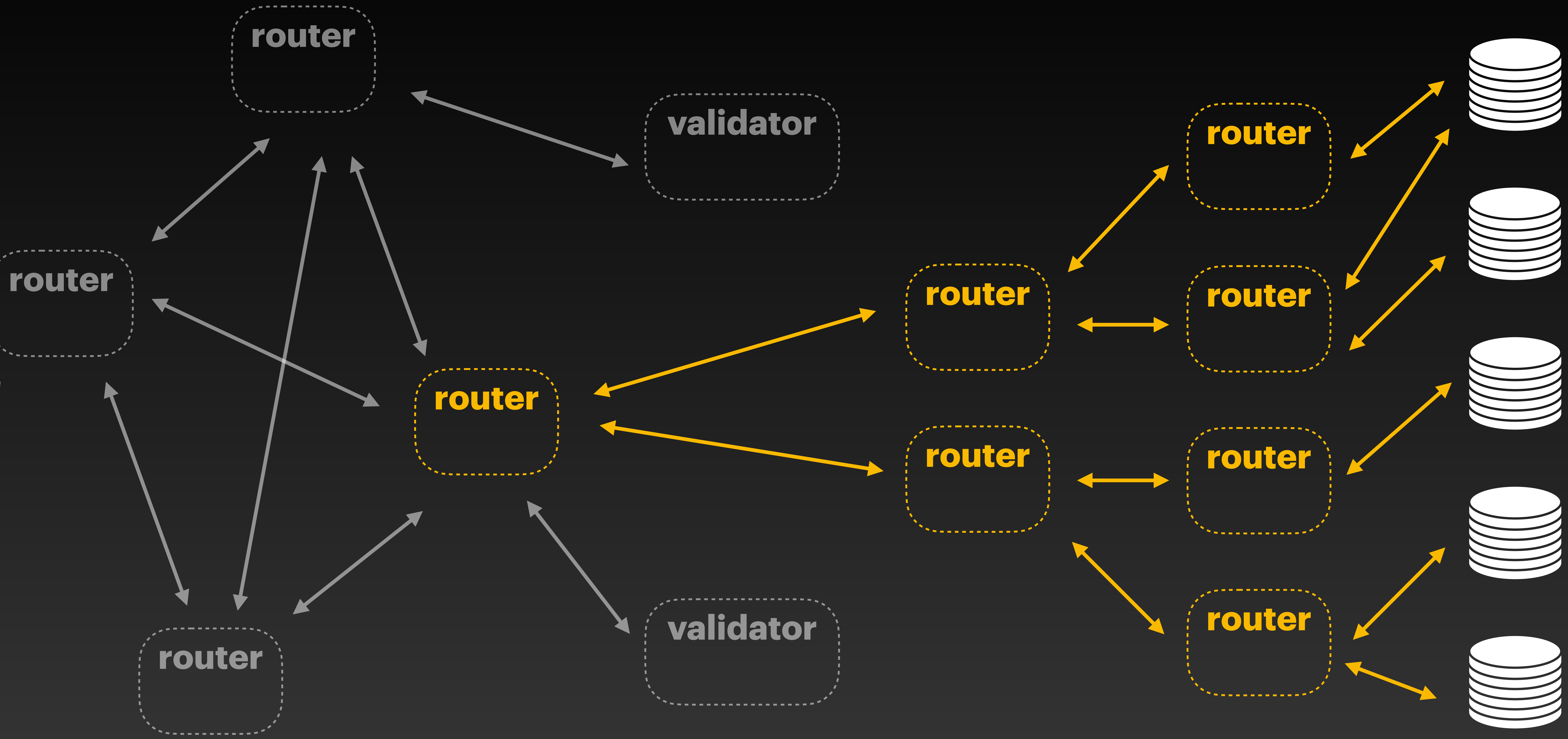
Challenge #7: Reads

- Potentially very large number of readers (>400)
- Unpredictable, may read arbitrary data
- **Sometimes require extreme performance**

Network Security

Challenge #7: Reads

- Potentially very large number of readers (>400)
- Unpredictable, may read arbitrary data
- Sometimes require extreme performance
- **Most reads must be free**


router
validator
router
router
router
validator
router
router
router
router
router
router
router

alberto@mystenlabs.com