Modern Blockchains

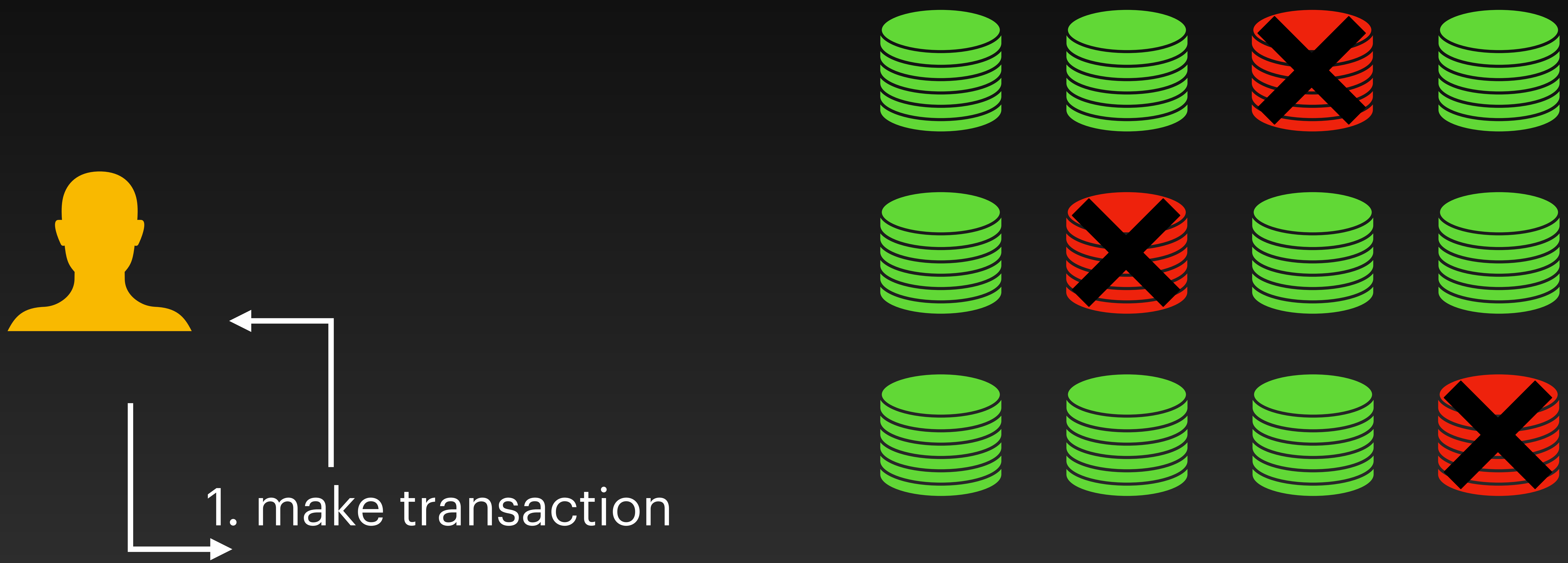
Broadcast and Execution

Alberto Sonnino

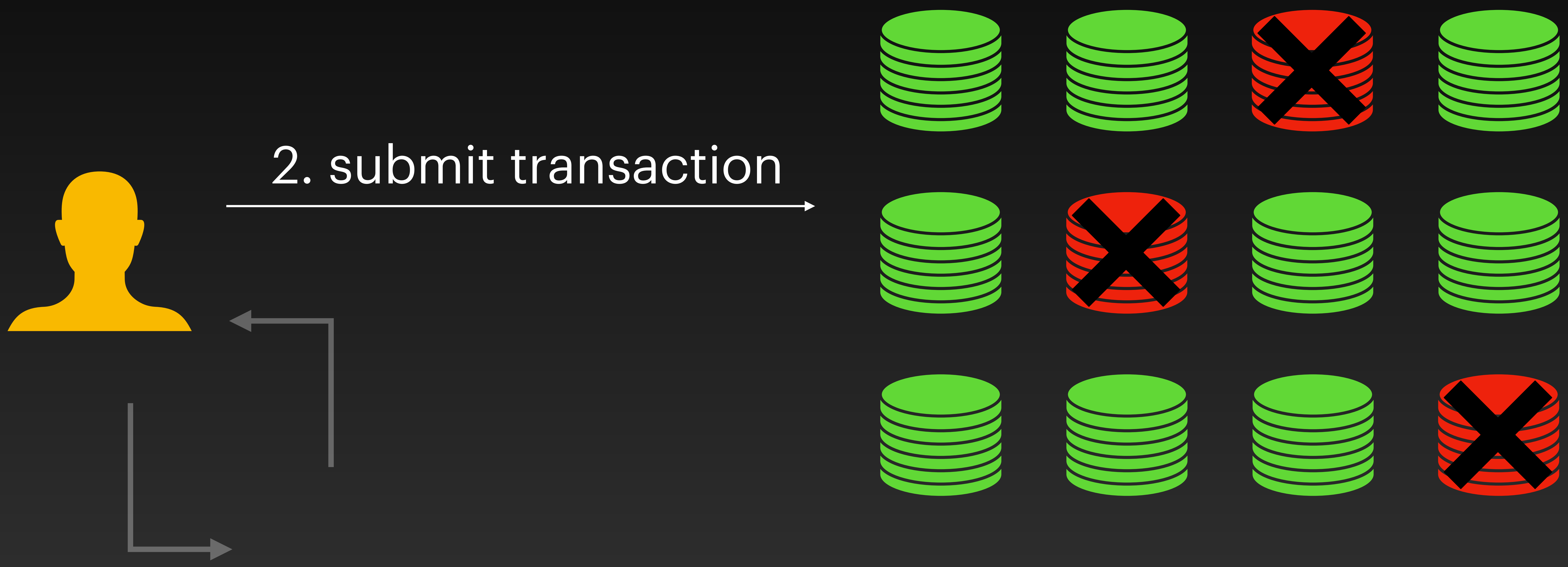
Byzantine Fault Tolerance

Byzantine Fault Tolerance

Blockchains

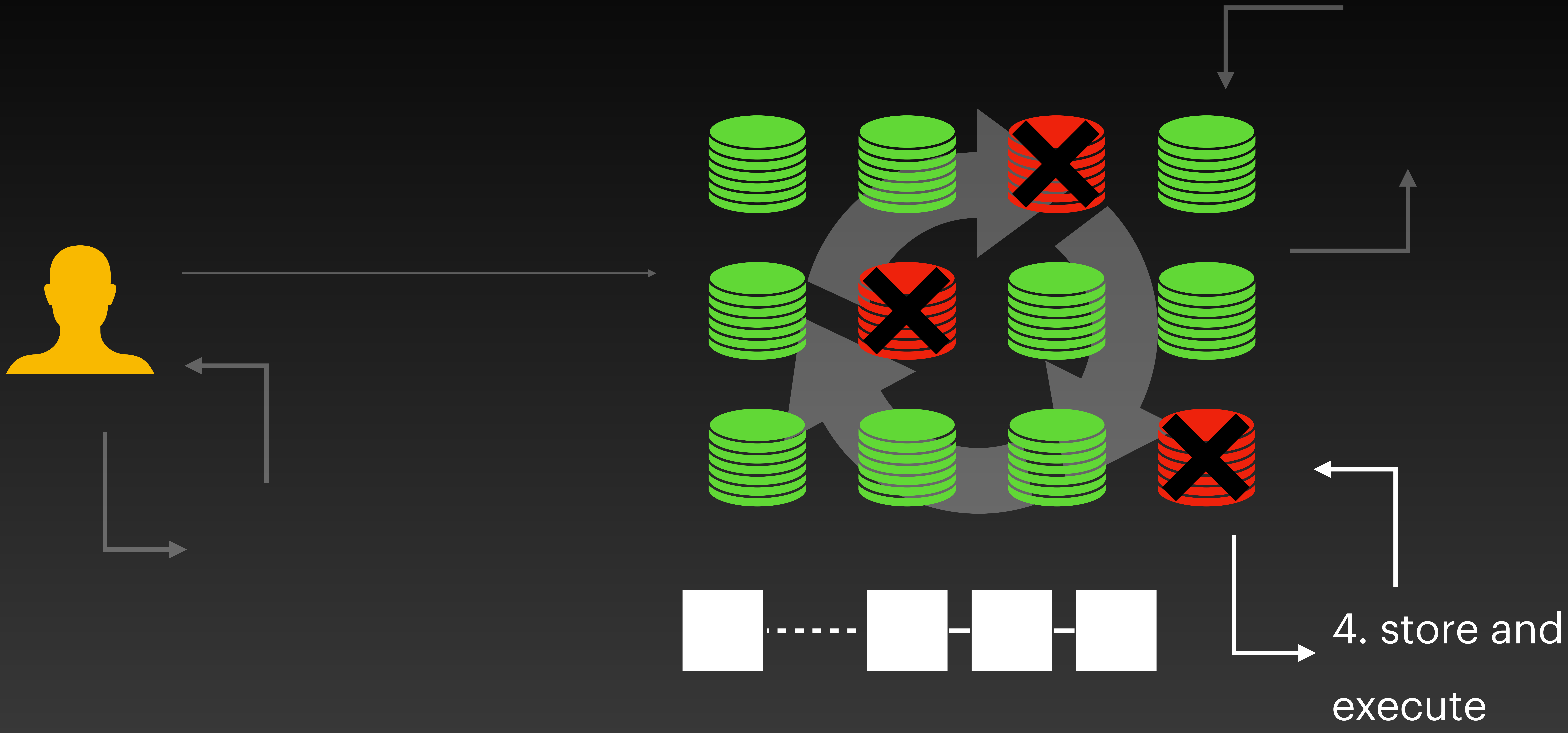


Blockchains

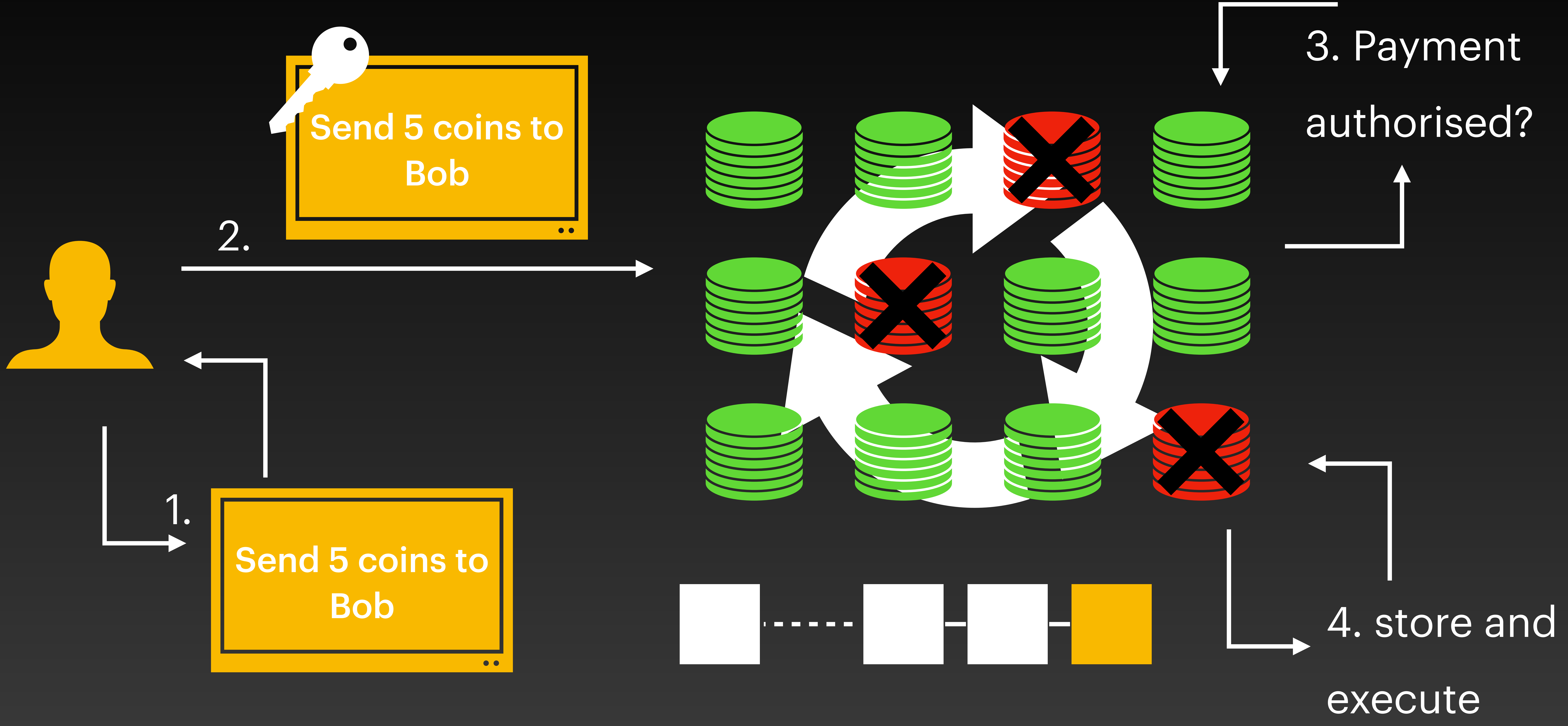


Blockchains

Blockchains



The Typical Example



Cross-Domain Discipline

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

Network Security

Challenge #1

Some node are not well-protected in datacenter;
we can't rely on beefy machines

Network Security

Challenge #2

Highly dynamic set of nodes

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

Network Security
Challenge #3

Some nodes are bad, you may be talking with someone lying and trying to DoS you

Network Security

Challenge #4

Bad nodes have access to all committee (insider) information

Typical Architecture

	Mempool / Initial Checks	Ordering	(Sequencial) Execution	DB Update & High-Integrity DS
TX →	P2P flood & Selection on fee	Sequence all transactions in blocks	Execute each transaction (global lock)	Update DB, indexes, crypto (Merkle trees)
	Overlay flooding slow and with significant redundancy	Seconds latency, traditionally low throughput	Single core does all computations. (eg EVM ~300 tps)	Added latency of store, blocks, and crypto computations

Typical Architecture

Sequence all transactions in blocks

Execute each transaction (global lock)

Ordering

(Sequencial) Execution

Single core does all computations. (eg EVM ~300 tps)

Seconds latency, traditionally low throughput

New Architecture

Consensus is not required

Coins, balances, and transfers

NFTs creation and transfers

Game logic allowing users to combine assets

Inventory management for games / metaverse

Auditable 3rd party services not trusted for safety

...

New Architecture

Consensus is required

- Increment a publicly-accessible counter
- Auctions
- Market places
- Collaborative in-game assets
- ...

New Architecture

The Sui System

Consensus only when you need to

New Architecture

Architecture

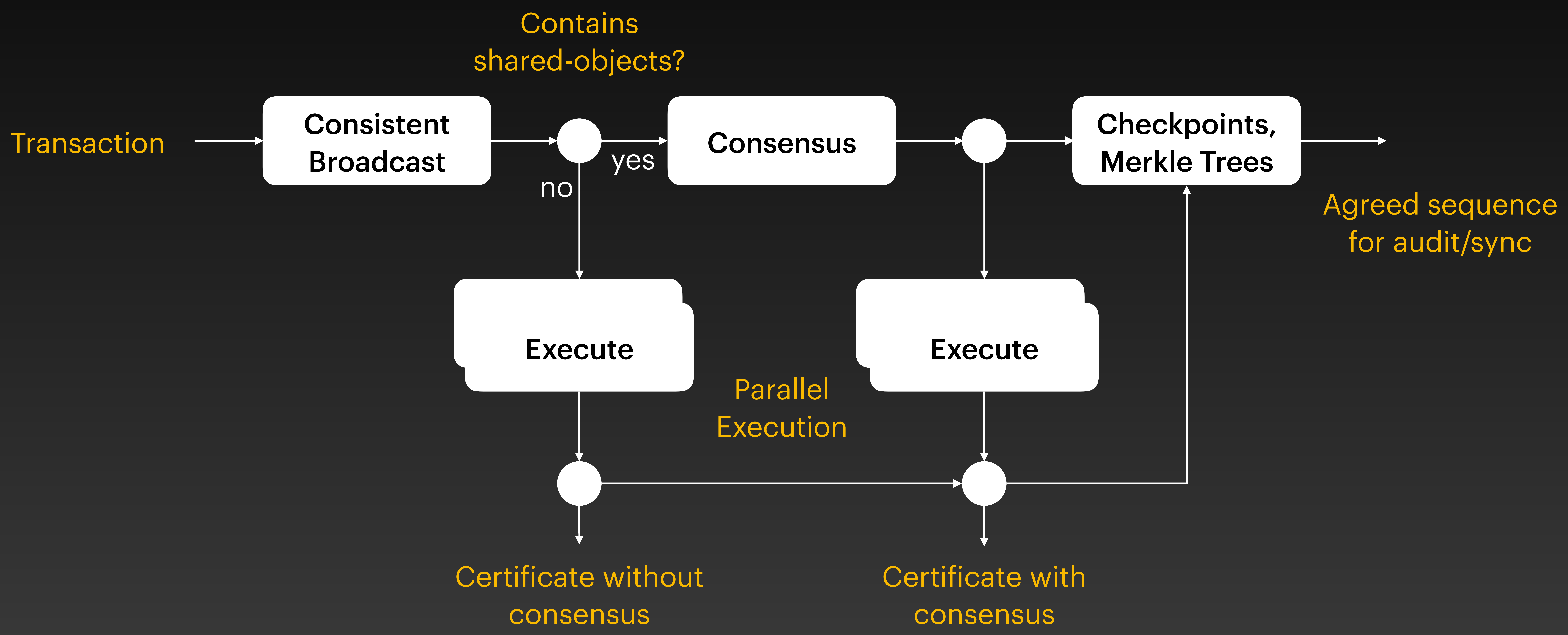
Owned Objects

- Objects that can be mutated by a single entity
- e.g., My bank account
- **Do not need consensus**

Shared Objects

- Objects that can be mutated my multiple entities
- e.g., A global counter
- **Need consensus**

The Sui System
Architecture



The Sui System
Transactions

Objects:

- Unique ID
- Version number
- Ownership Information
- Type (shared, owned)

The Sui System

Transactions

Objects:

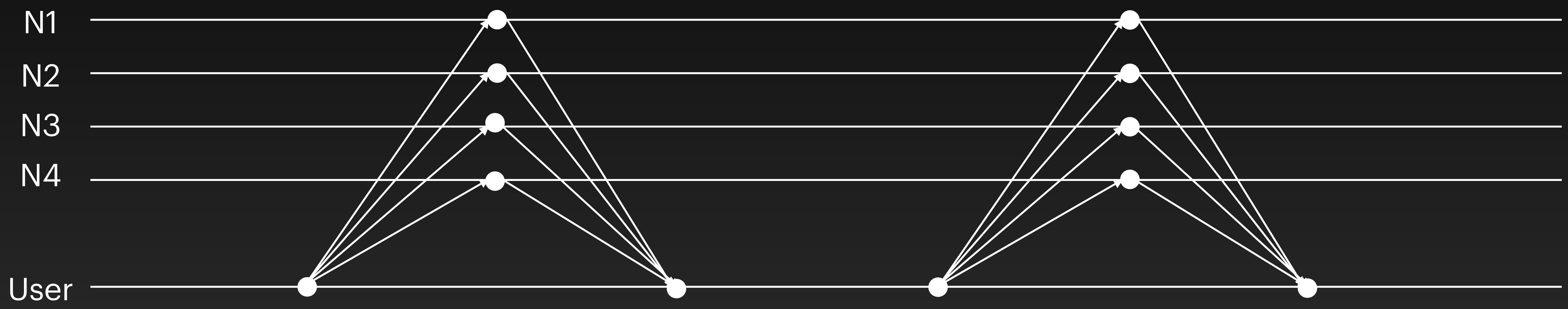
- Unique ID
- Version number
- Ownership Information
- Type (shared, owned)

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	

The Sui System

Consensus-less Path



Send T1:

Disseminate the transaction

Echo T1:

Nodes check and sign T1

Cert T1:

User gather >2/3 signatures into a certificate and disseminate it

Effect T1:

User gather >2/3 effect signatures for finality

Network Security

Challenge #5

Different types of target links: clients-nodes and nodes-nodes

The Sui System

Consensus-less Path

Example Transaction

T1

Inputs: O1, O2, O3

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

The Sui System

Consensus-less Path

Example Transaction

T1

Inputs: O1, O2, O3

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

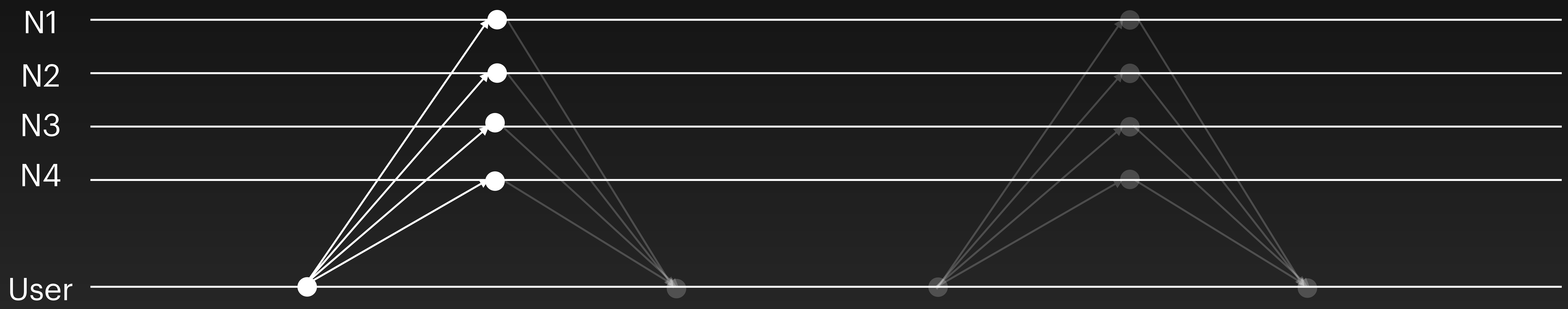
e.g., Mutate a coin to pay for gas

e.g., Transfer my warrior to friend

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

e.g., Be rewarded with a mystery gift

The Sui System
Consensus-less Path



Send T1:

Disseminate the transaction

The Sui System

Consensus-less Path

Step 1: Owned object locks & version exist at validator

O1

L1 = (O1, 10)

Sender=X : None

O2

L2 = (O2, 27)

Sender=X : None

O3

L3 = (O3, 1001)

Sender=X : None

We call these “locks”, and are initialised to None.

The Sui System
Consensus-less Path

Step 2: Validator V checks / signs transactions

O1

L1 = (O1, 10)

Sender=X : ~~None~~ T1

O2

L2 = (O2, 27)

Sender=X : ~~None~~ T1

O3

L3 = (O3, 1001)

Sender=X : ~~None~~ T1

Transaction: T1

Inputs: (O1, 10), (O2, 27), (O3, 1001)

Move call details

Signature of X

Checks T1 (Validity)

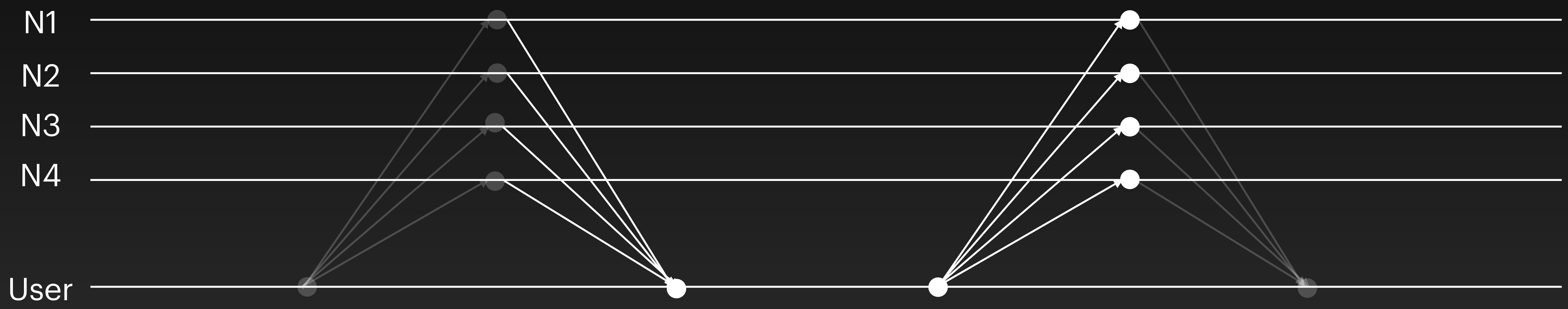
- Well-formed (syntactic)
- Valid Signature from X
- Valid execution function
- Version owned by X

Checks T1 (Broadcast)

- Object-version exist Lock was set to None

The Sui System

Consensus-less Path



Echo T1:

Nodes check and sign T1

Cert T1:

User gather >2/3 signatures into a certificate and disseminate it

The Sui System
Consensus-less Path

Step 3: Validator V process certificate

O1

L1 = (O1, 10)

Sender=X : ~~None~~ T1

O2

L2 = (O2, 27)

Sender=X : ~~None~~ T1

O3

L3 = (O3, 1001)

Sender=X : ~~None~~ T1

Transaction: T1

Inputs: (O1, 10), (O2, 27), (O3, 1001)

Move call details

Signature of X

Signature (V1, … V4)

Checks T1 (Validity)

- Again!

Checks T1 (Broadcast)

- Objects exist (with any lock)
- Certificate signed by quorum

The Sui System
Consensus-less Path

Step 4: Validator V executes / signs effect

O1

L1 = (O1, 11)

Sender=X : None

O2

L2 = (O2, 28)

Sender=Y : None

O4

L3 = (O4, 1)

Sender=X : None

Transaction: T1

Inputs: (O1, 10), (O2, 27), (O3, 1001)

Move call details

Signature of X

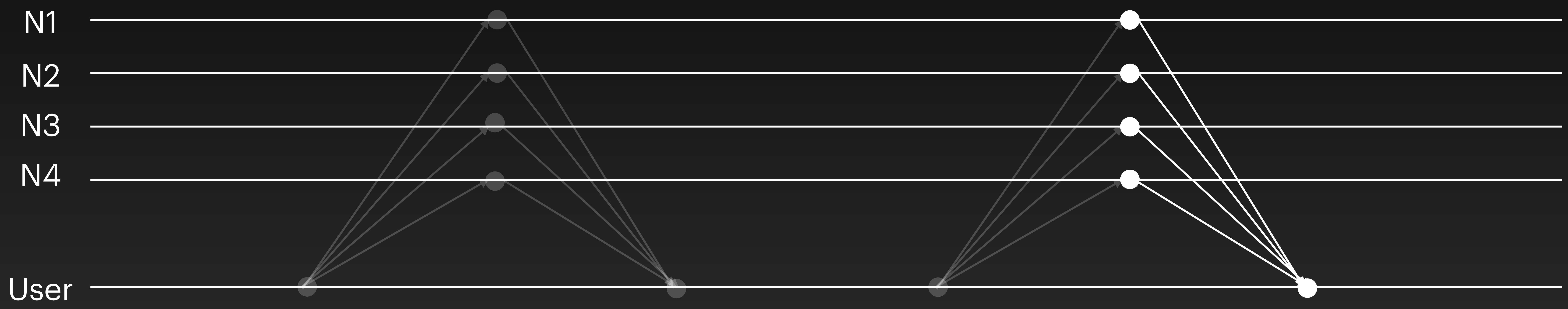
Signature (V1, ... V4)

Execute T1

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

The Sui System

Consensus-less Path



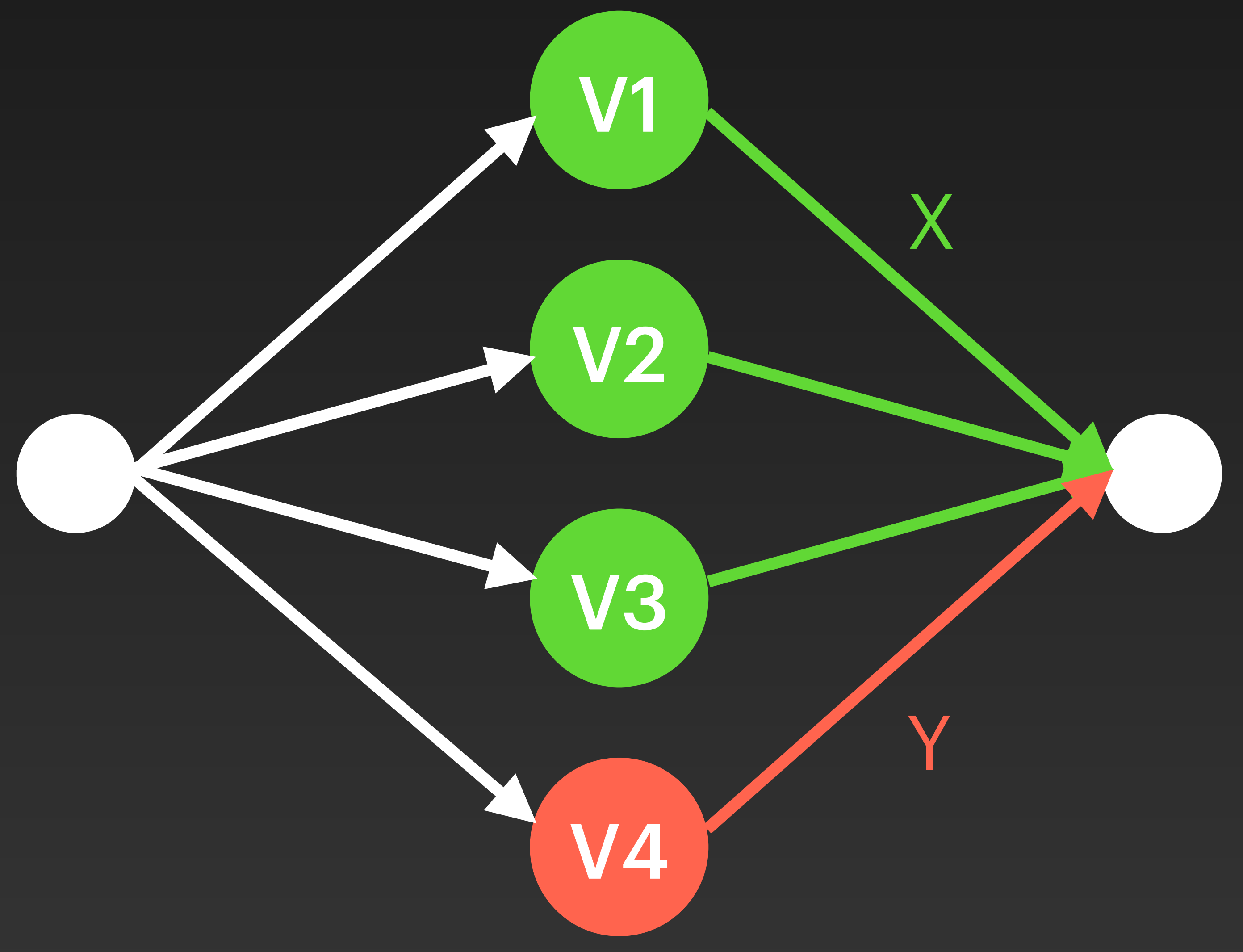
Effect T1:

User gather >2/3 effect signatures for finality

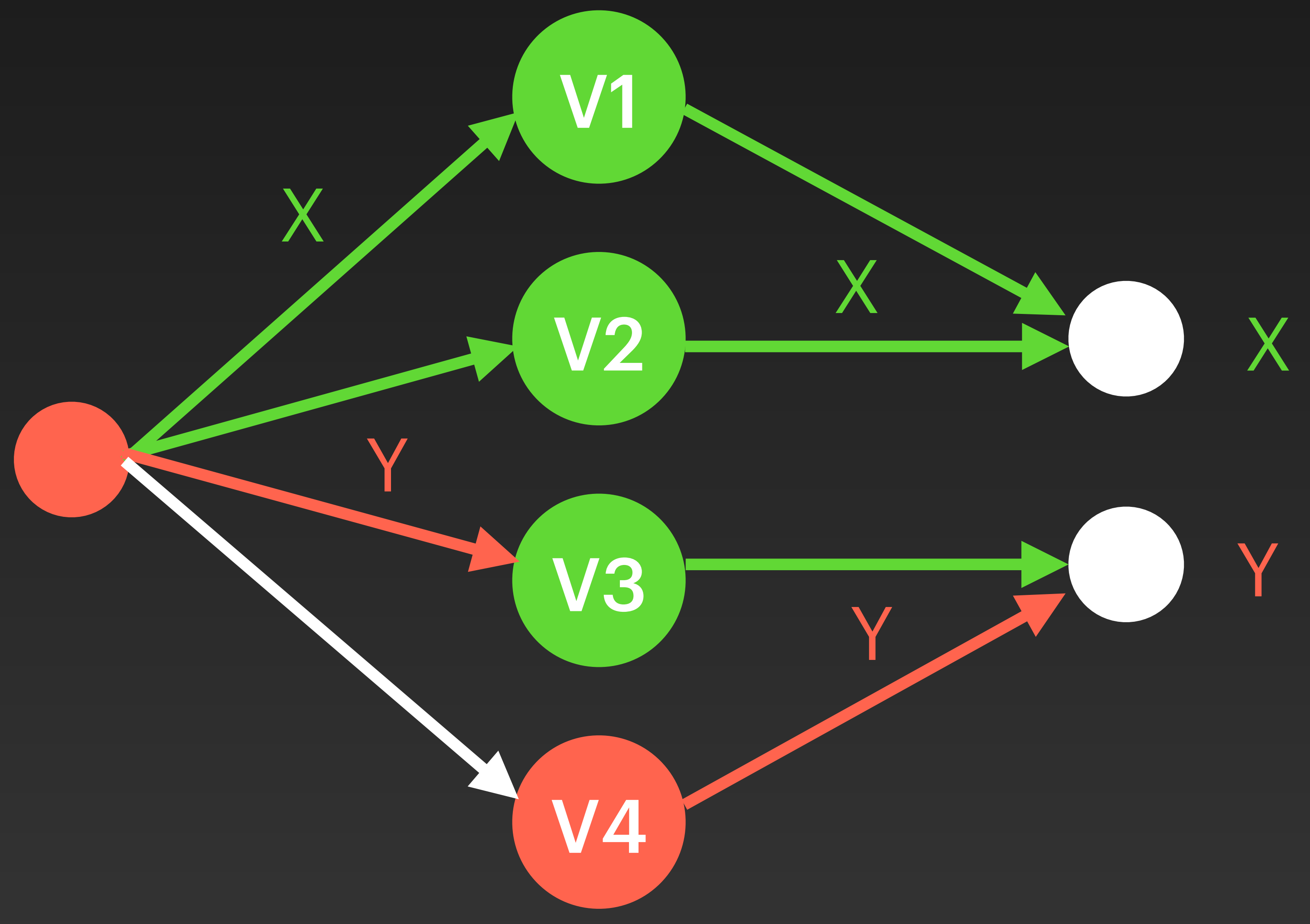
Quorum Intersection

Why do we need it?

With

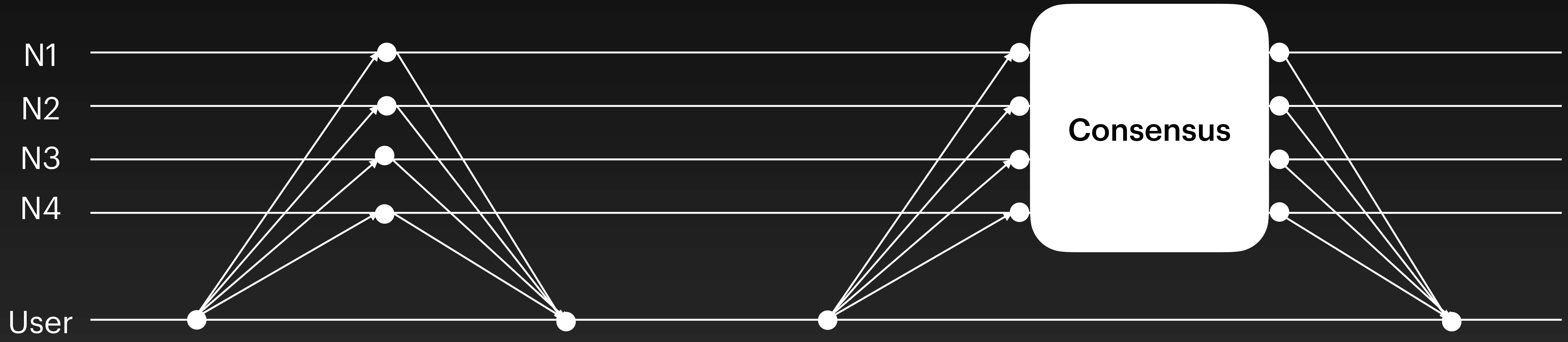


Without



The Sui System

Consensus Path



Send T1:

Disseminate the transaction

Echo T1:

Nodes check and sign T1

Cert T1:

User gather >2/3 signatures into a certificate and disseminate it

Effect T1:

User gather >2/3 effect signatures for finality

The Sui System
Consensus Path

Example Transaction

T2

Inputs: O1, S2

Output: Mutate O1, Mutate S2, Create O4

The Sui System

Consensus Path

Send T1:

Disseminate the transaction

The Sui System
Consensus Path

Step 1: Shared object locks exist at validator

O1

L1 = (O1, 10)

Sender=X : None

S2

L2 = (S2, *)

Sender=X

Do not check the version for shared objects

The Sui System
Consensus Path

Step 2: Validator V checks / signs transactions

O1

L1 = (O1, 10)

Sender=X : ~~None~~ T2

S2

L2 = (S2, *)

Sender=X

Transaction: T2

Inputs: (O1, 10), (S2, *)

Move call details

Signature of X

Checks T2 (Validity)

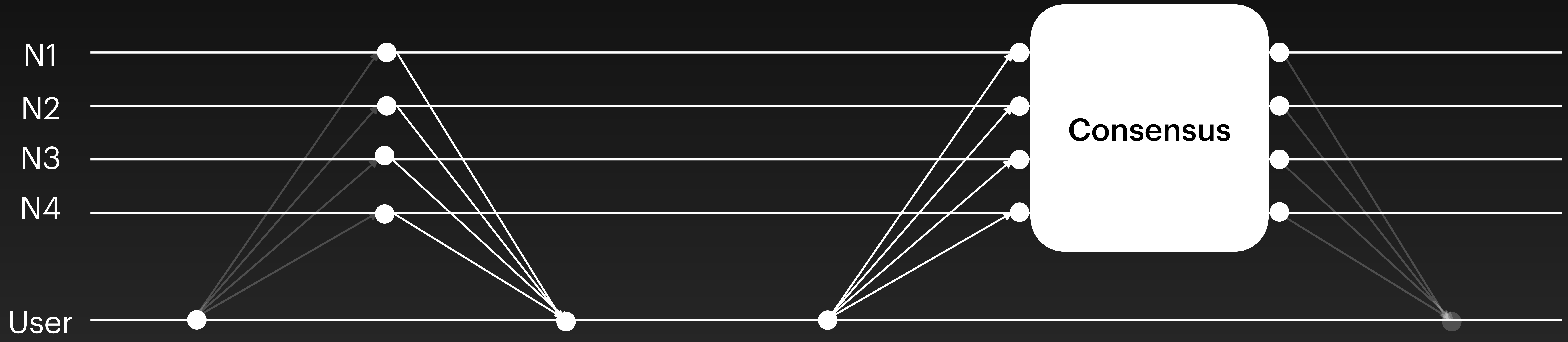
- Well-formed (syntactic)
- Valid Signature from X
- Valid execution function
- Version owned by X

Checks T2 (Broadcast)

- Object-version exist
- Lock is set to None

The Sui System

Consensus Path



Echo T1:

Nodes check and sign T2

Cert T1:

User gather >2/3 signatures into a certificate and disseminate it

The Sui System
Consensus Path

Step 3: After consensus, assign shared objects locks

O1

L1 = (O1, 10)

Sender=X : ~~None~~ T2

S2

L2 = (S2, 4)

Sender=X

Transaction: T2

Inputs: (O1, 10), (S2, *)

Move call details

Signature of X

Assign Shared Locks

- Every node sees the same sequence out of consensus
- So they can all assign the same shared object locks

The Sui System
Consensus-less Path

Same as before

Step 3: Validator V process certificate

O1

L1 = (O1, 10)

Sender=X : ~~None~~ T2

S2

L2 = (S2, 4)

Sender=X

Transaction: T2

Inputs: (O1, 10), (S2, *)

Move call details

Signature of X

Checks T2 (Validity)

- Again!

Checks T2 (Broadcast)

- Objects exist (with any lock)
- Certificate signed by quorum

The Sui System
Consensus-less Path

Same as before

Step 4: Validator V Applies / Signs Effect

O1

L1 = (O1, 11)

Sender=X : None

S2

L2 = (S2, 4)

Sender=X

O4

L3 = (O4, 1)

Sender=X : None

Transaction: T2

Inputs: (O1, 10), (S2, *)

Move call details

Signature of X

Execute T2

- O1 mutated
- O2 mutated
- O4 created

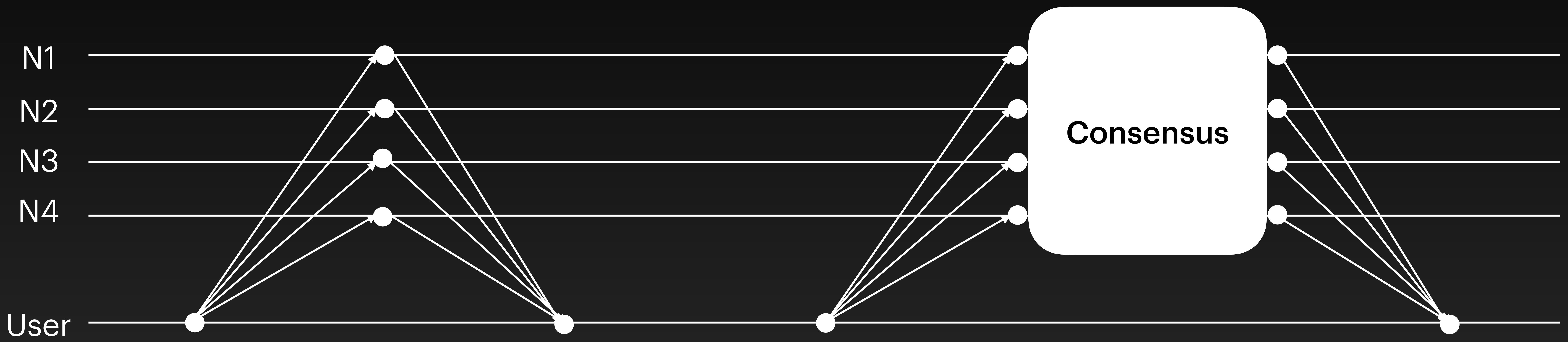
The Sui System

Consensus Path

Effect T1:

User gather >2/3 effect signatures for finality

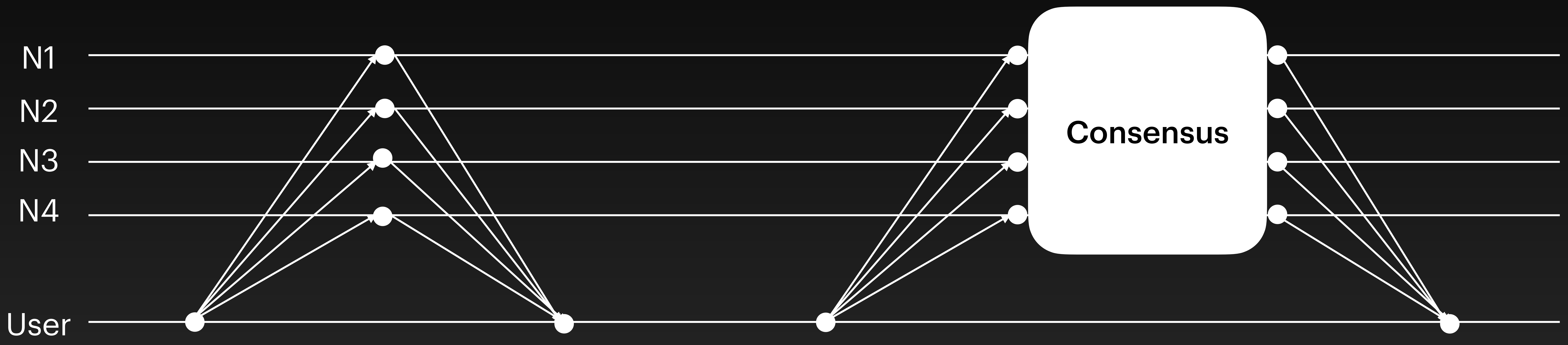
Why Consensus?



No single entity to assign version numbers:
the nodes need to choose it

Network Security

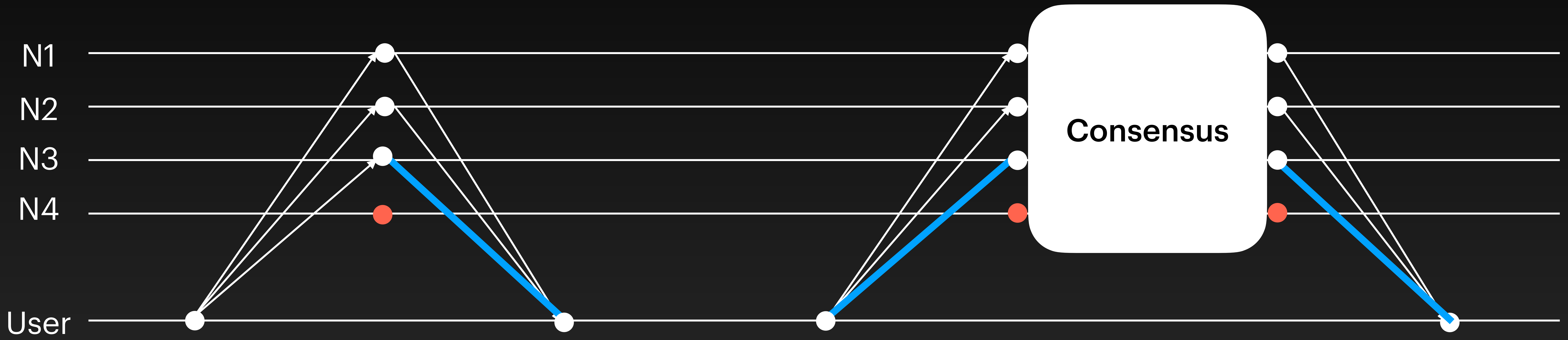
Challenge #6



If consensus is under DoS, all shared objects transactions are stalled

Network Security

Challenge #7



If any blue link is under DoS, the protocol stalls
(because we won't have a quorum)

The Sui System
Transaction Execution

- First, execute all precedent transactions
- Once there is a certificate, any validator can download Tx and execute

The Sui System
Transaction Execution

Owned-objects

Core 1

$\text{Ox, 10} \xrightarrow{\text{Tx}} \text{Ox, 11}$

Core 2

$\text{Oy, 65} \xrightarrow{\text{Ty}} \text{Oy, 66}$

Always executed in parallel

(once they inputs ID/version are known)

Shared-objects

Core 3

$\text{Ov, ?} \xrightarrow{\text{Tv}} \text{Ov, ?+1}$

Core 4

$\text{Ow, ?} \xrightarrow{\text{Tw}} \text{Ow, ?+1}$

Often executed in parallel

(Sequentially for each shared object)

Conclusion

The Sui System

- Separate owned and shared objects
- Only use consensus when you need to
- Execute in parallel whenever you can

- **Paper:** https://sui.io
- **Code:** https://github.com/mystenlabs/sui

alberto@mystenlabs.com