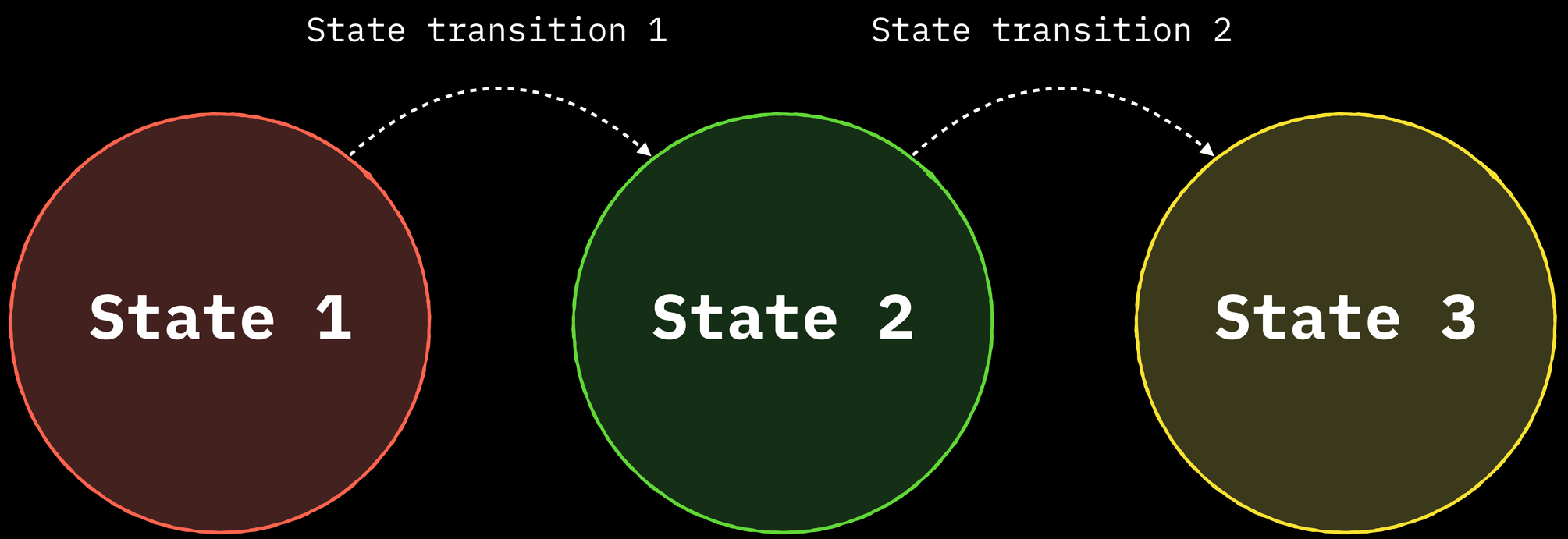
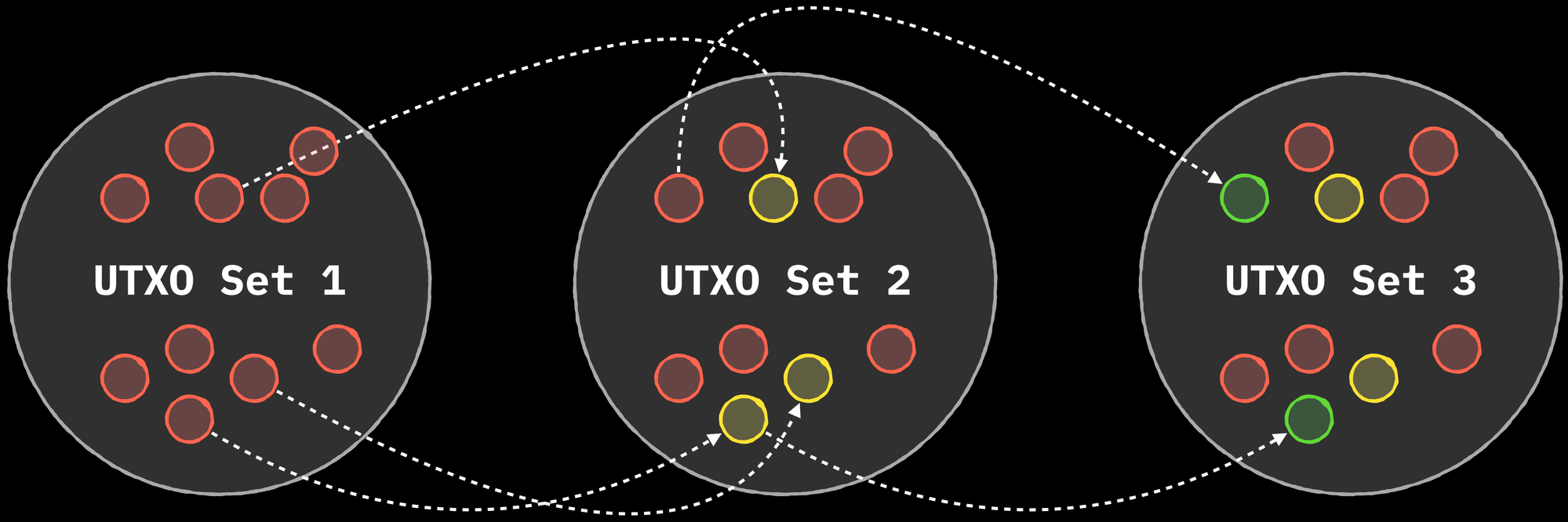
Client-side-validation

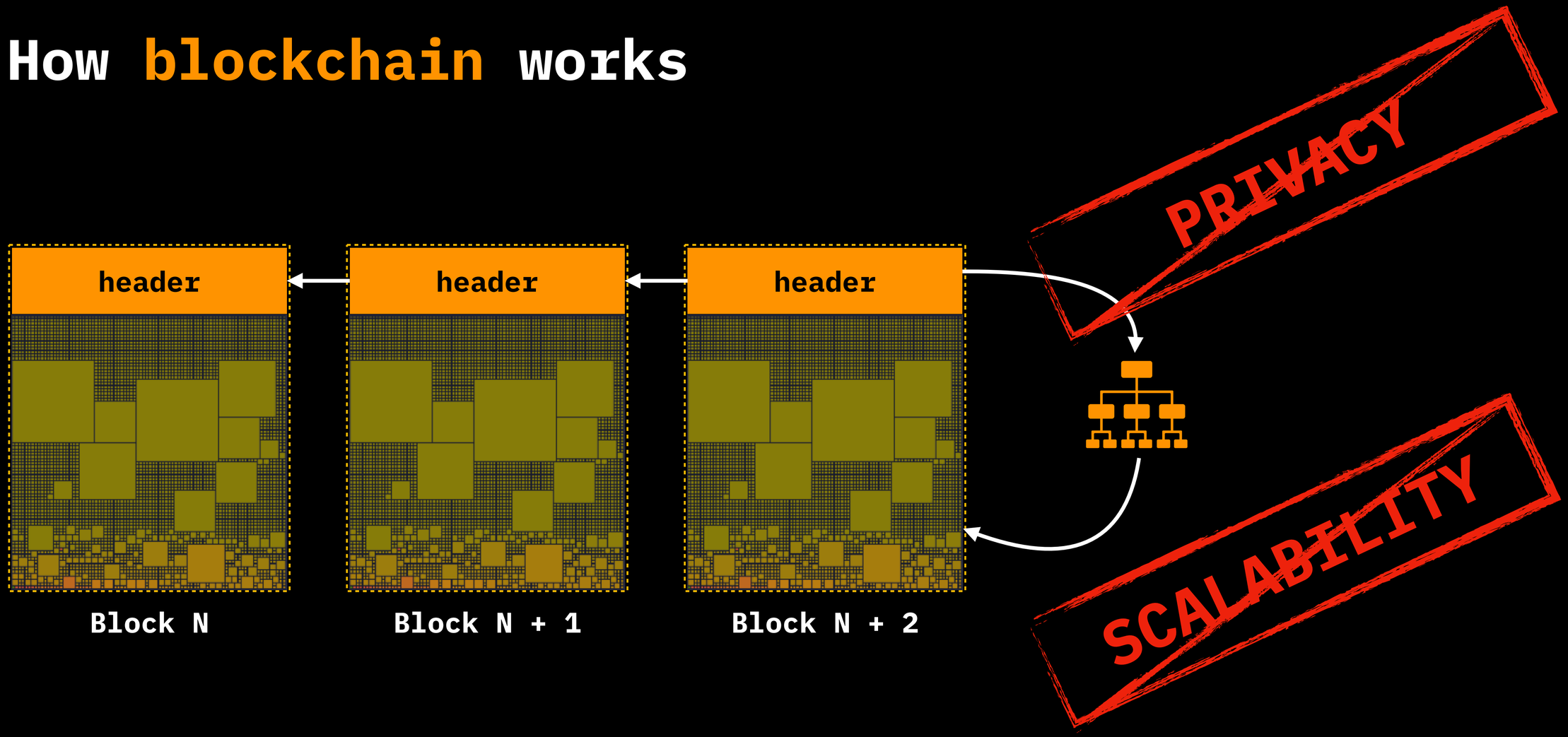
State and state machine



State and state machine

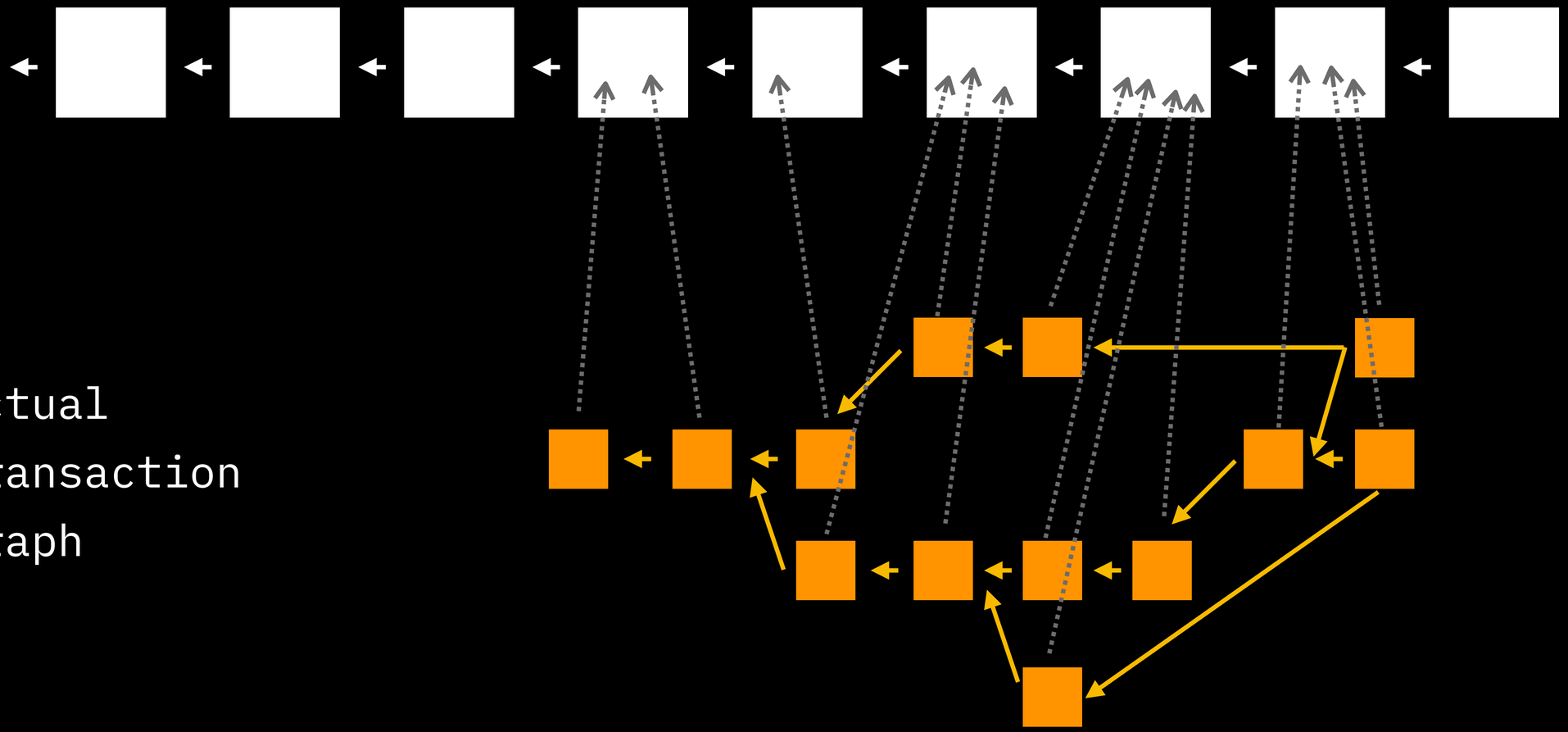


How blockchain works



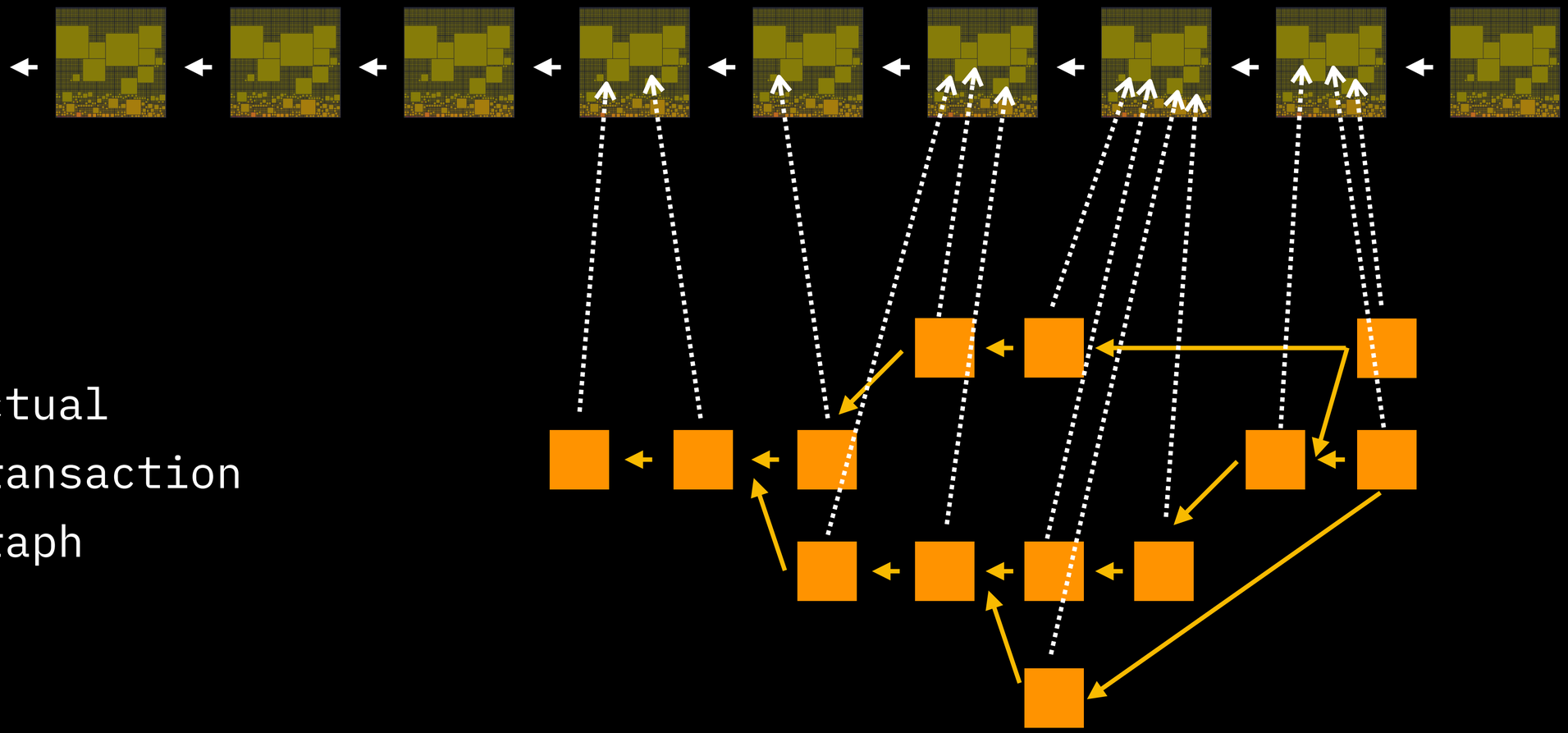
Everybody validates every transaction from everybody

Blockchain block packing



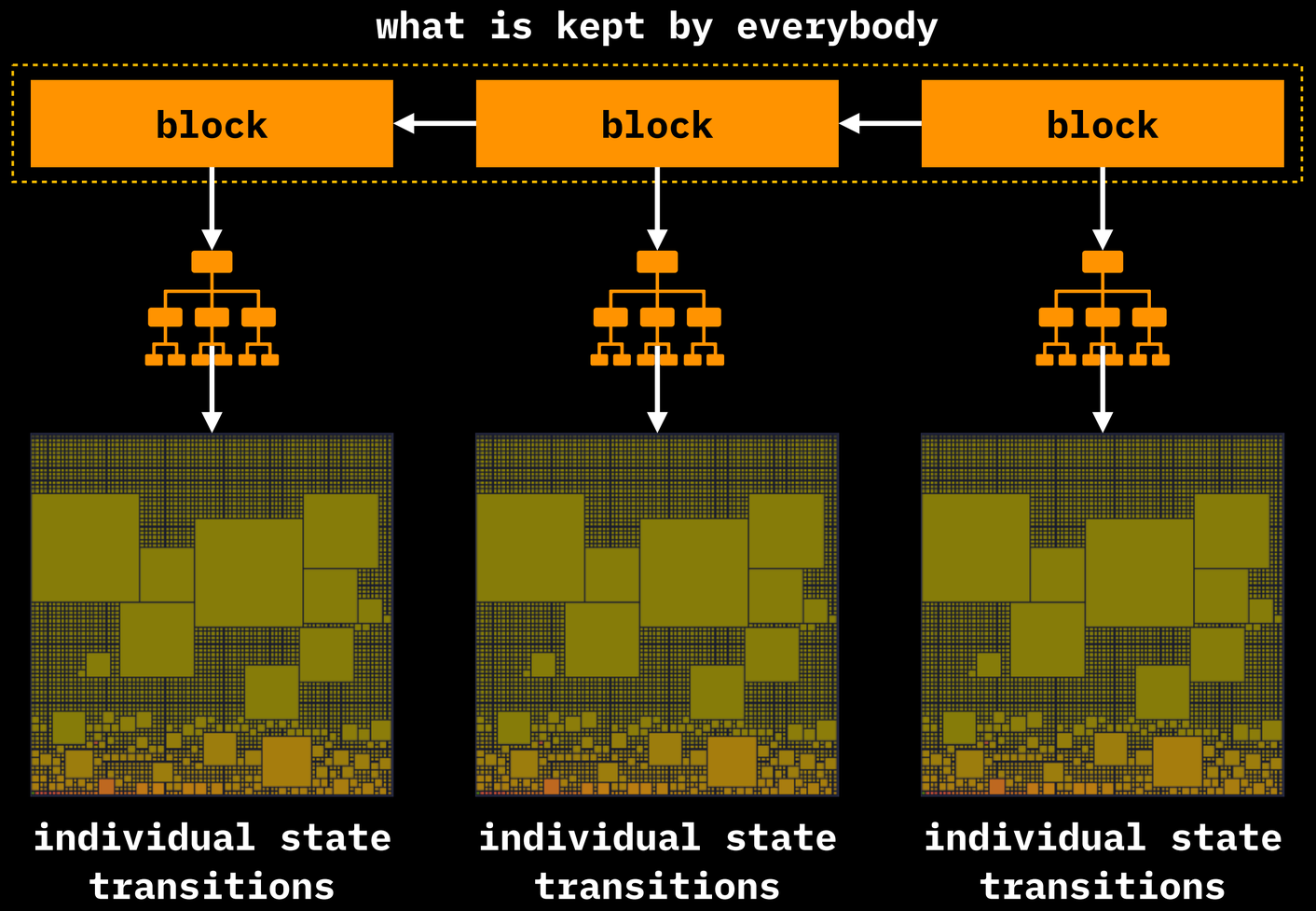
Actual transaction graph

Blockchain block packing



Actual transaction graph

How client-side validation works

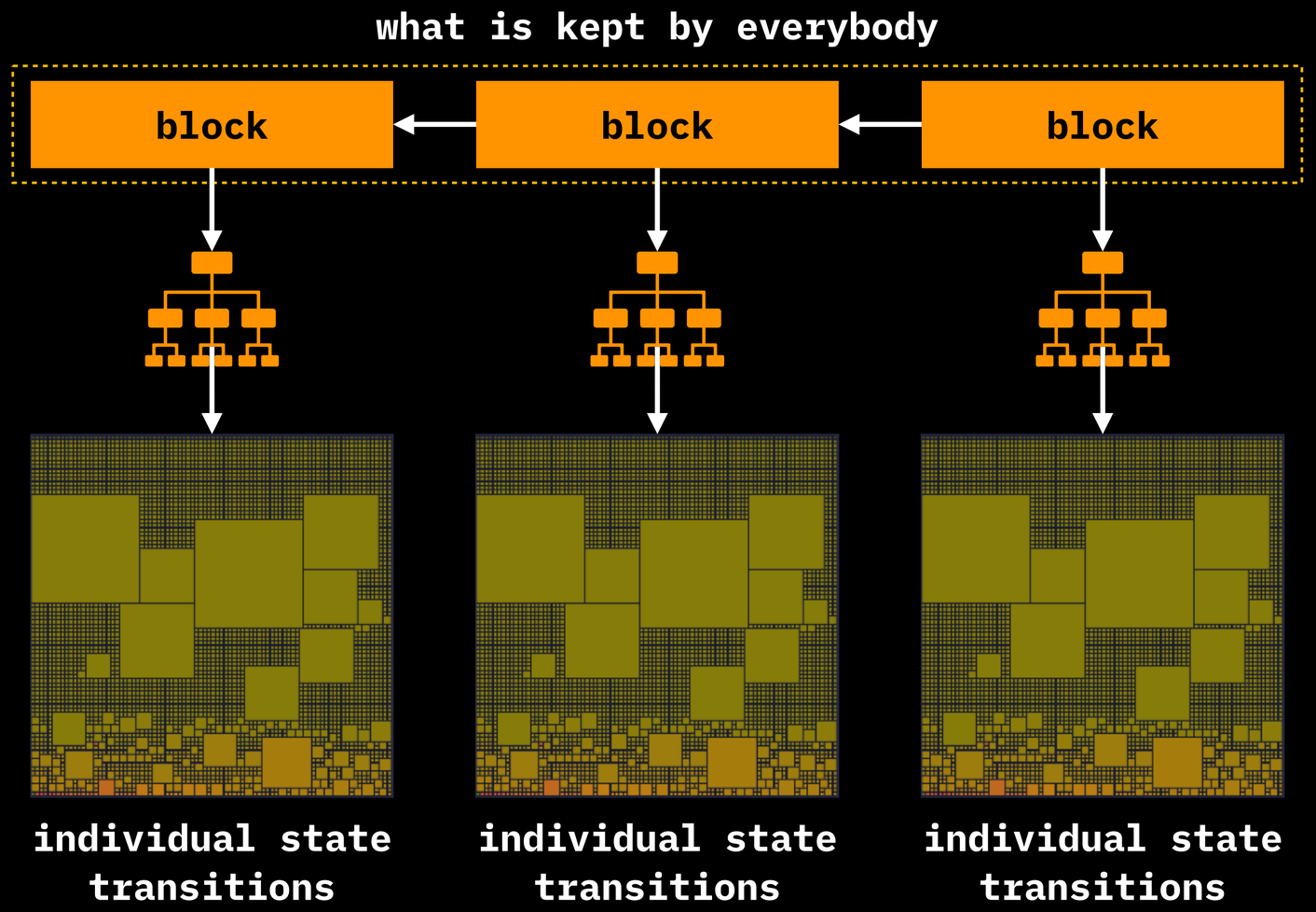


PRIVACY

SCALABILITY

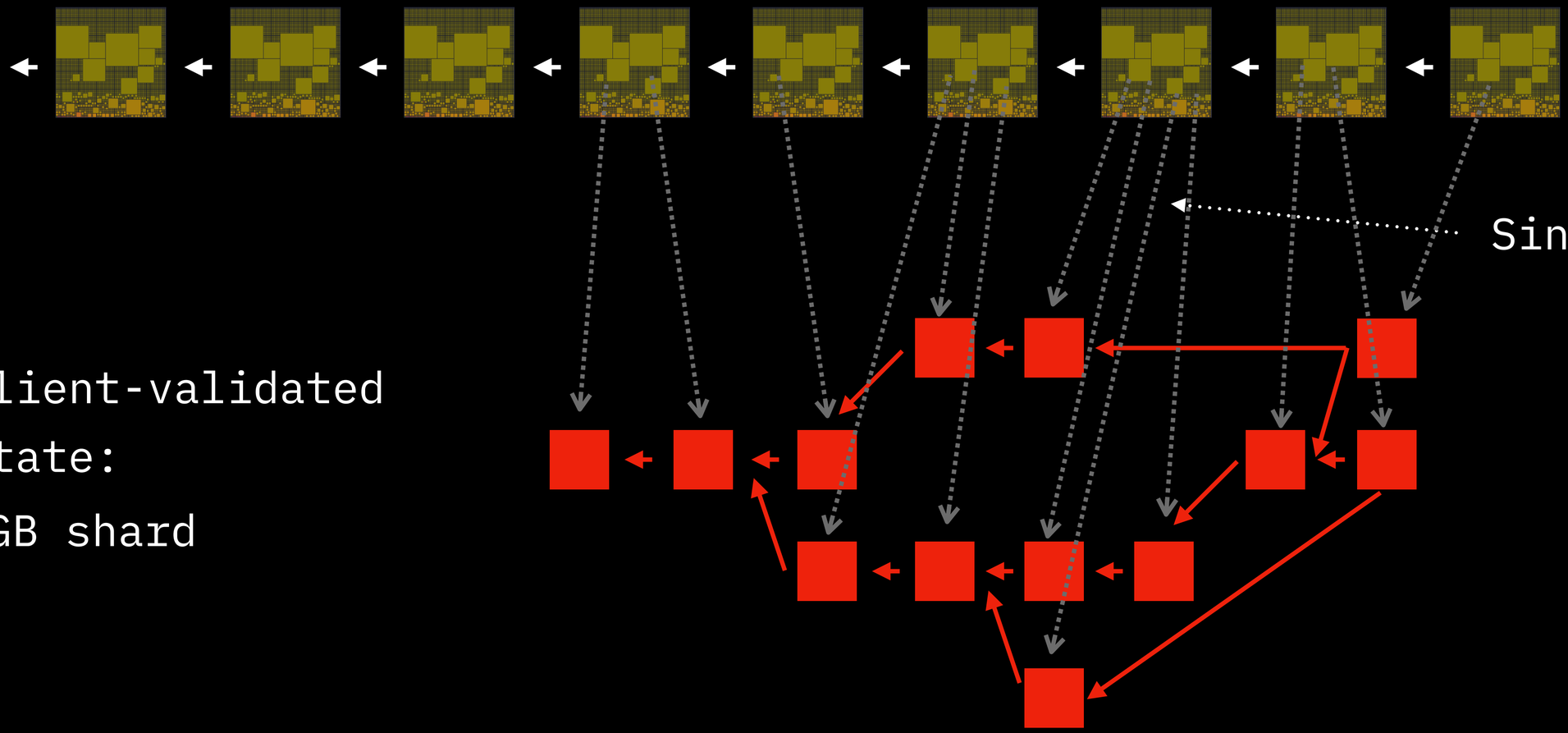
Each party validates just its own part of transaction history

How client-side validation works



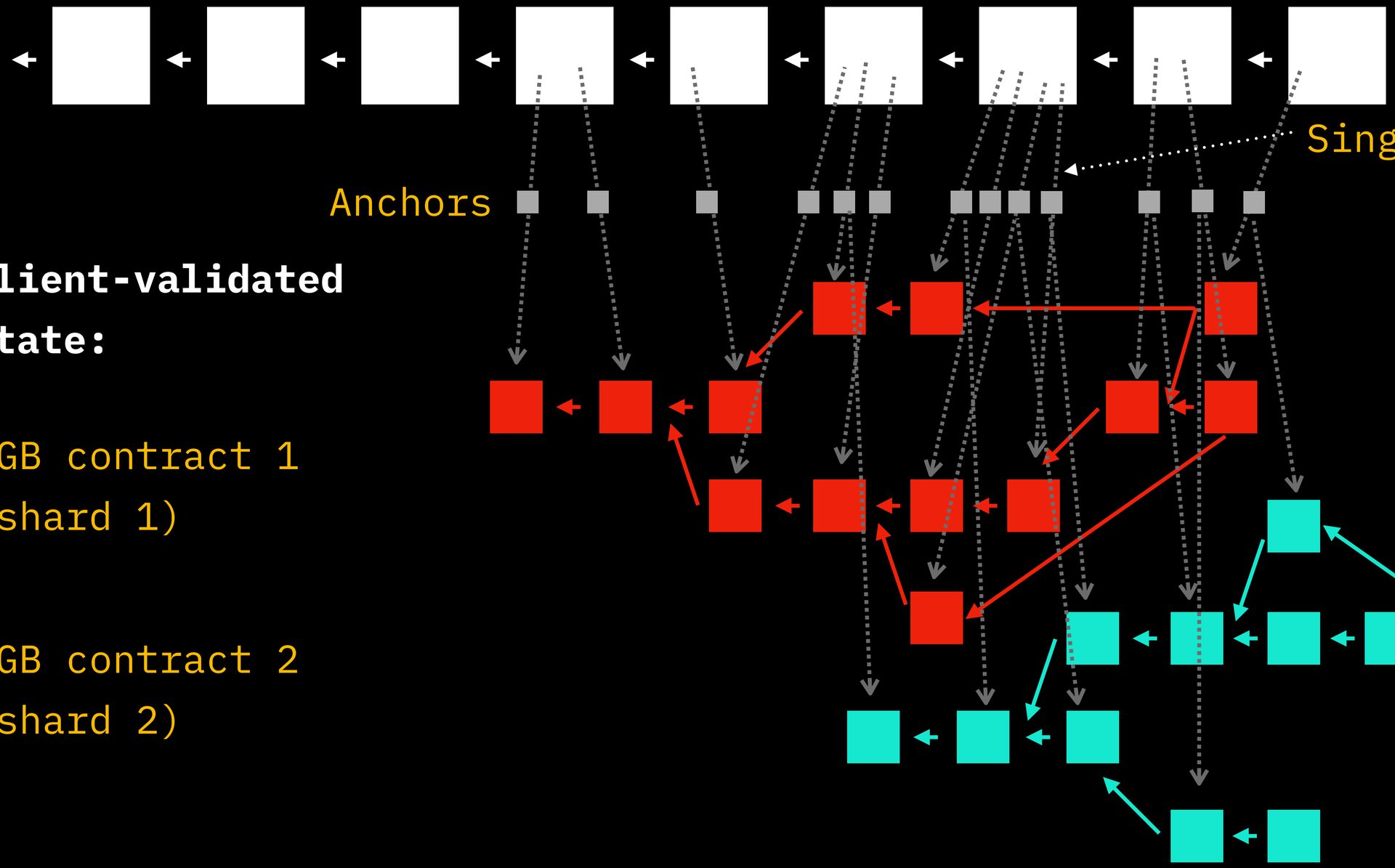
Each party validates just its own part of transaction history

Bitcoin blockchain:
state ownership



Client-validated
state:
RGB shard

Bitcoin blockchain:
state ownership



Client-validated state:

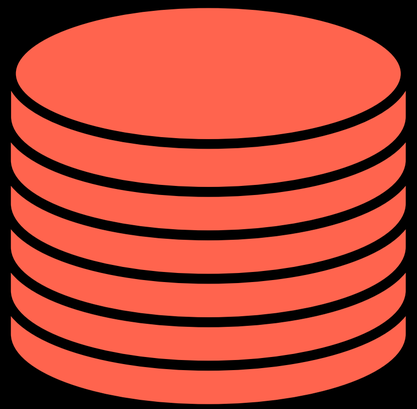
RGB contract 1
(shard 1)

RGB contract 2
(shard 2)

Channels

Blockchain

Stash

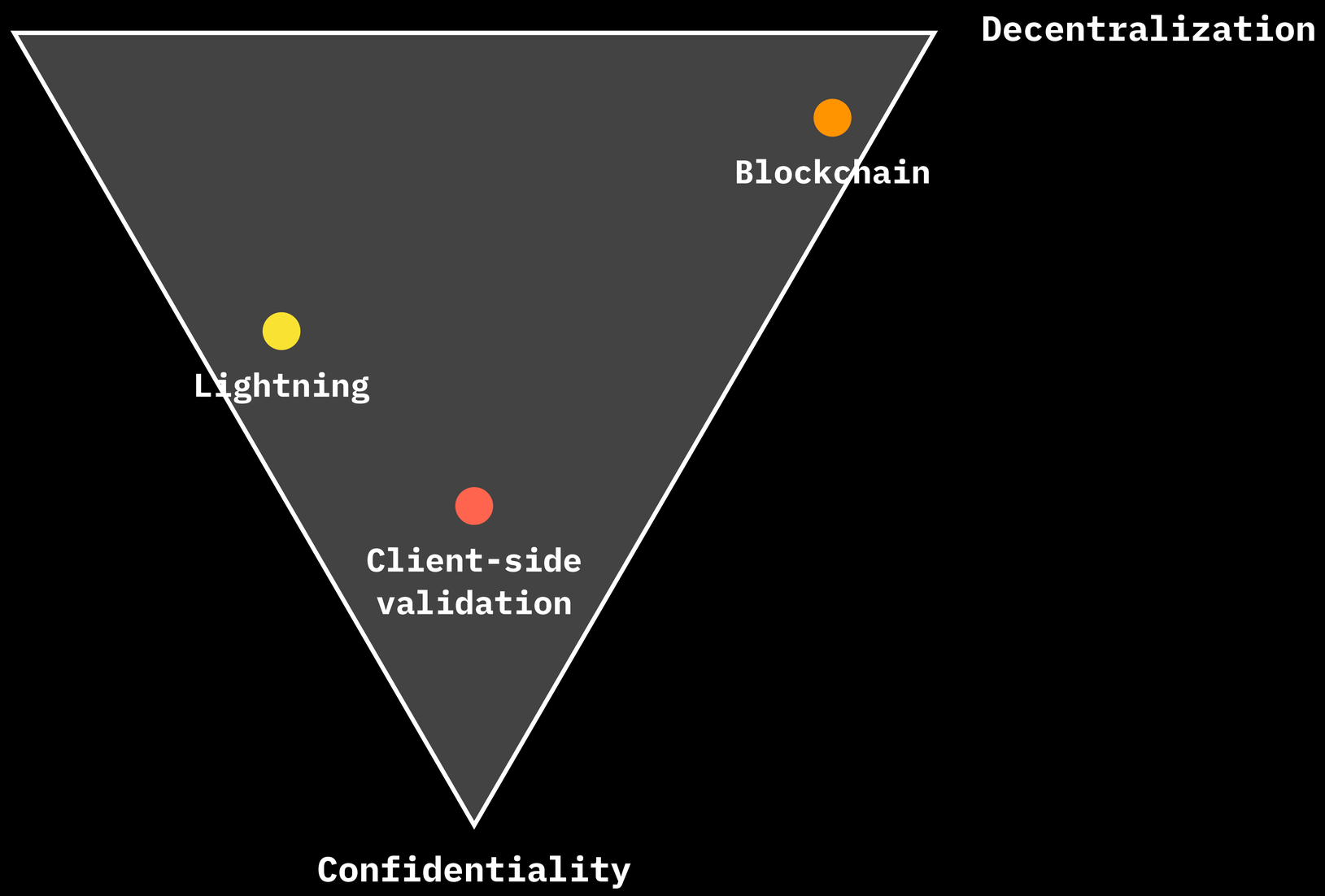


RGB client-side-validated data

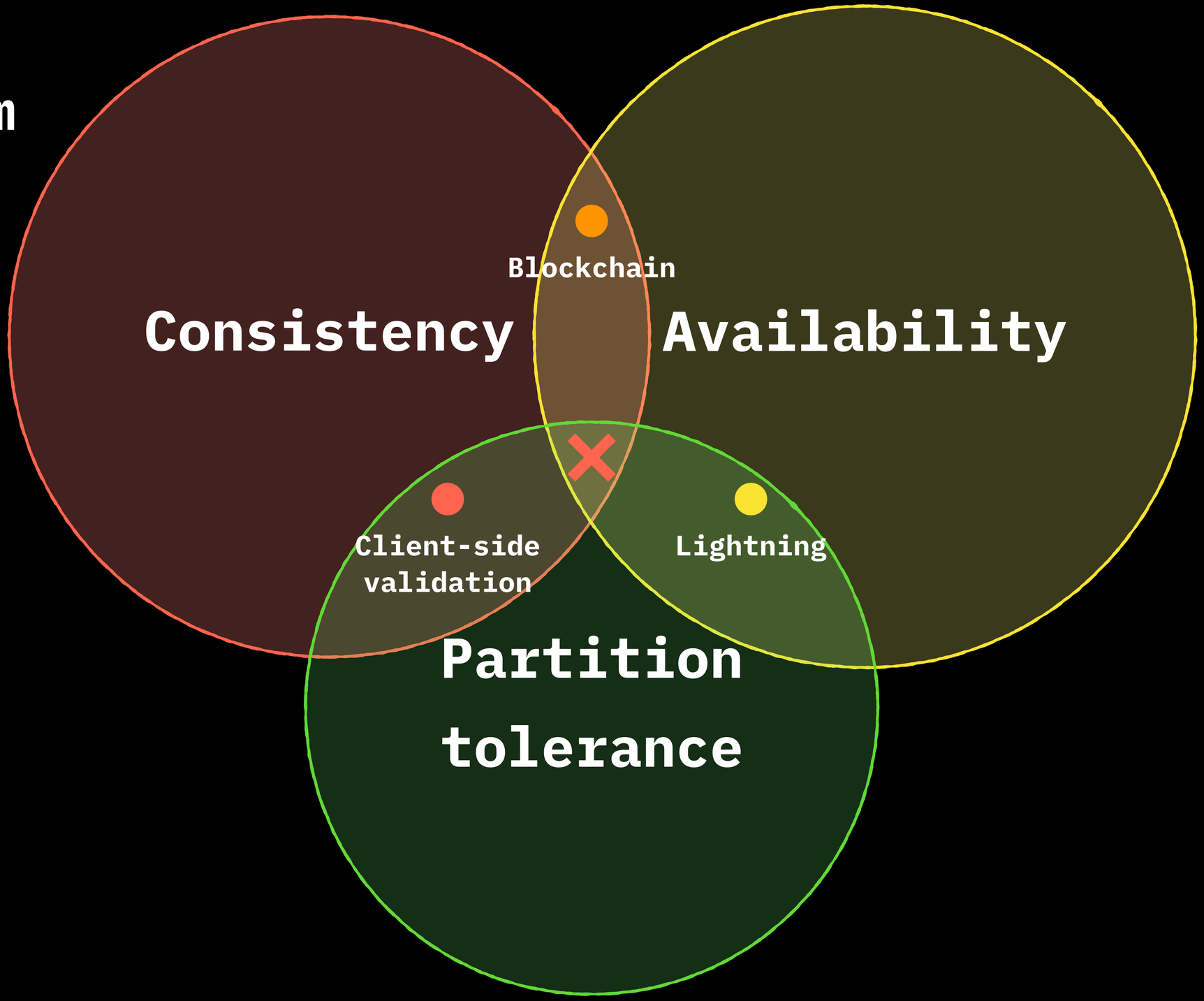
Channels

Blockchain

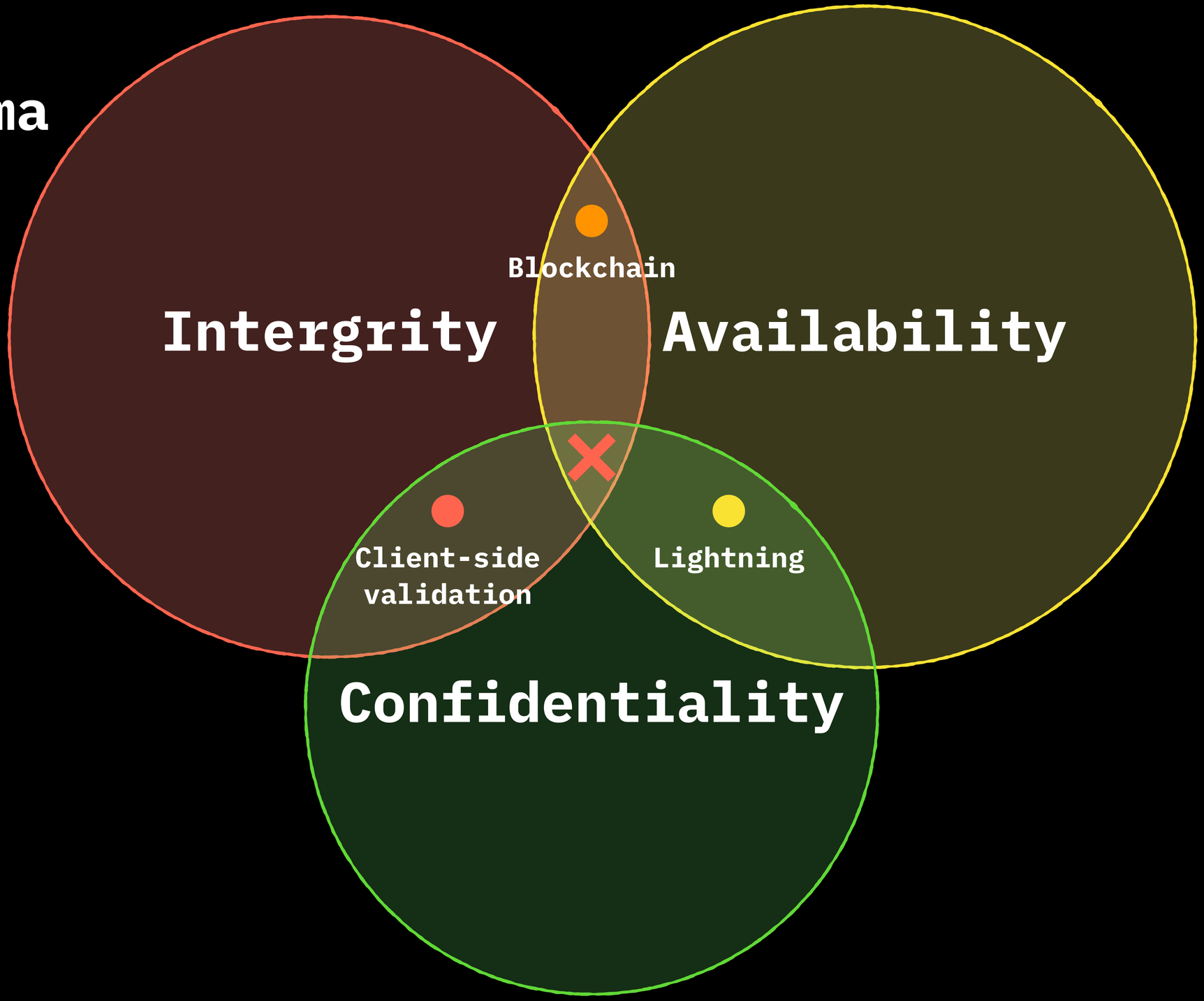
Trilemma


Scalability
Decentralization
Blockchain
Lightning
Client-side
validation
Confidentiality

CAP Theorem


Blockchain
Consistency
Availability
Client-side validation
Lightning
Partition tolerance

CIA Trilemma



Separating concerns

Asynchronous sharding	Client-side validated shard (PRISM)	Integrity & confidentiality
Synchronous sharding	State channels	Confidentiality & availability
Timestamping	Blockchain	Integrity & availability

Single-use seals

What is single-use-seal?

- Form of applied cryptographical commitment
- more advanced than
  - simple commitments
  - timestamps
- Requires “proof-of-publication medium”, i.e. medium preventing ability to “double spend” or “double commit” to certain data.
  This can be a medium with global consensus (like blockchain) but not necessarily decentralized (can be a centralized printed medium coming from a trusted party).
- Proposed as a cryptographic primitive by Peter Todd, following his development of timestamps

Proof of publication medium by Peter Todd

Proof-of-publication is what solves the double-spend/comit problem.

- Prove that every member *p* in of audience *P* has received message *m*.
  A real world analogy is a legal notice being published in a major newspaper - we can assume any subscriber received the message and had a chance to read it.

- Prove that message *m* has not been published.
  Extending the above real world analogy the court can easily determine that a legal notice was not published when it should have been by examining newspaper archives.

- Prove that some member *q* is in the audience *P*.
  Using newspaper analogy, because we know what today's date is, and we trust the newspaper never to publish two different editions with the same date we can be certain we have searched all possible issues where the legal notice may have been published.

Single-use-seal is
a commitment (formal promise) to commit
to a (yet) unknown message in the future,
once and only once,
such that commitment fact will be
provably known to all members of a certain audience

Single-use-seals vs other commitment schemes

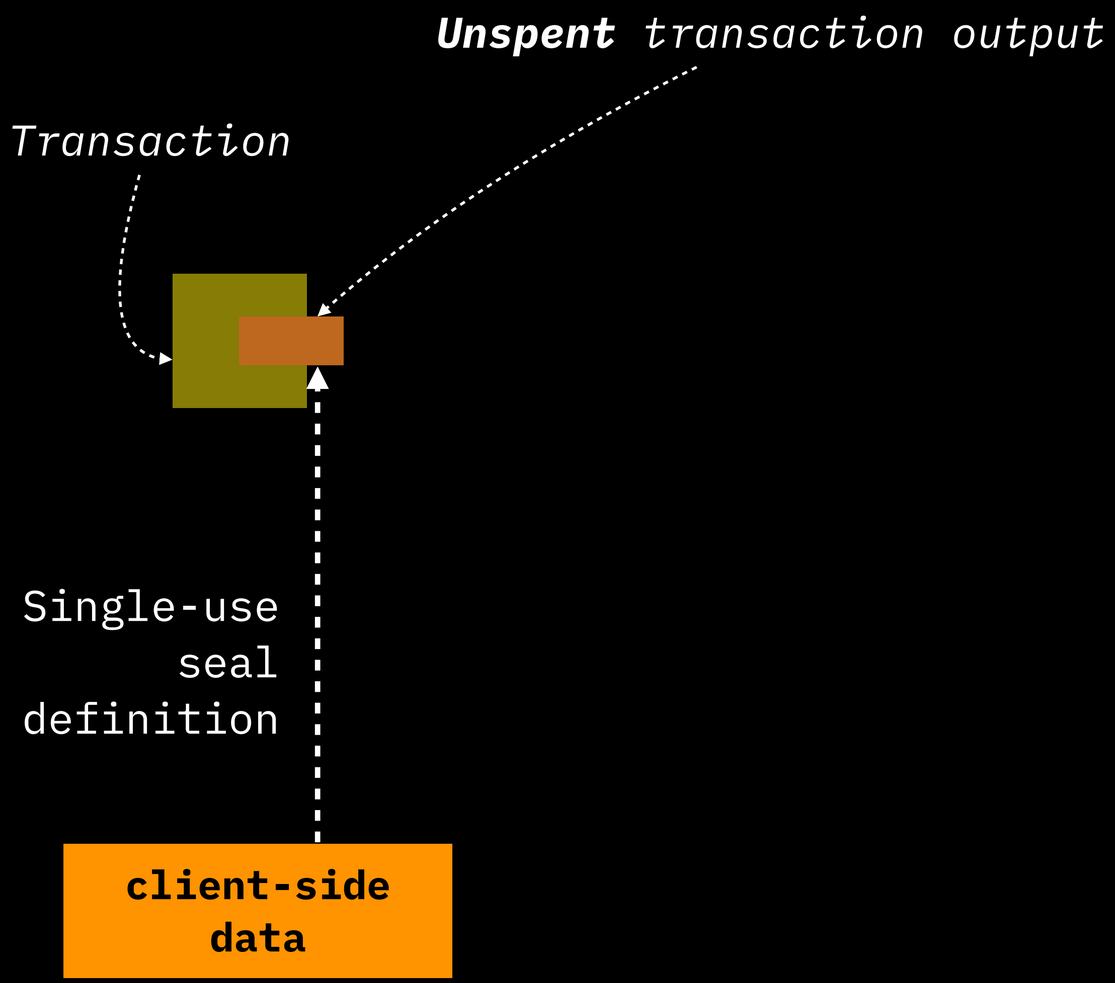
	Simple commitment (digest/hash)	Timestamps	Single-use-seals
Commitment publication does not reveal the message	Yes	Yes	Yes
Proof of the commitment time / message existence before certain date	Not possible	Possible	Possible
Prove that no alternative commitment can exist	Not possible	Not possible	Possible

Bitcoin single-use-seals

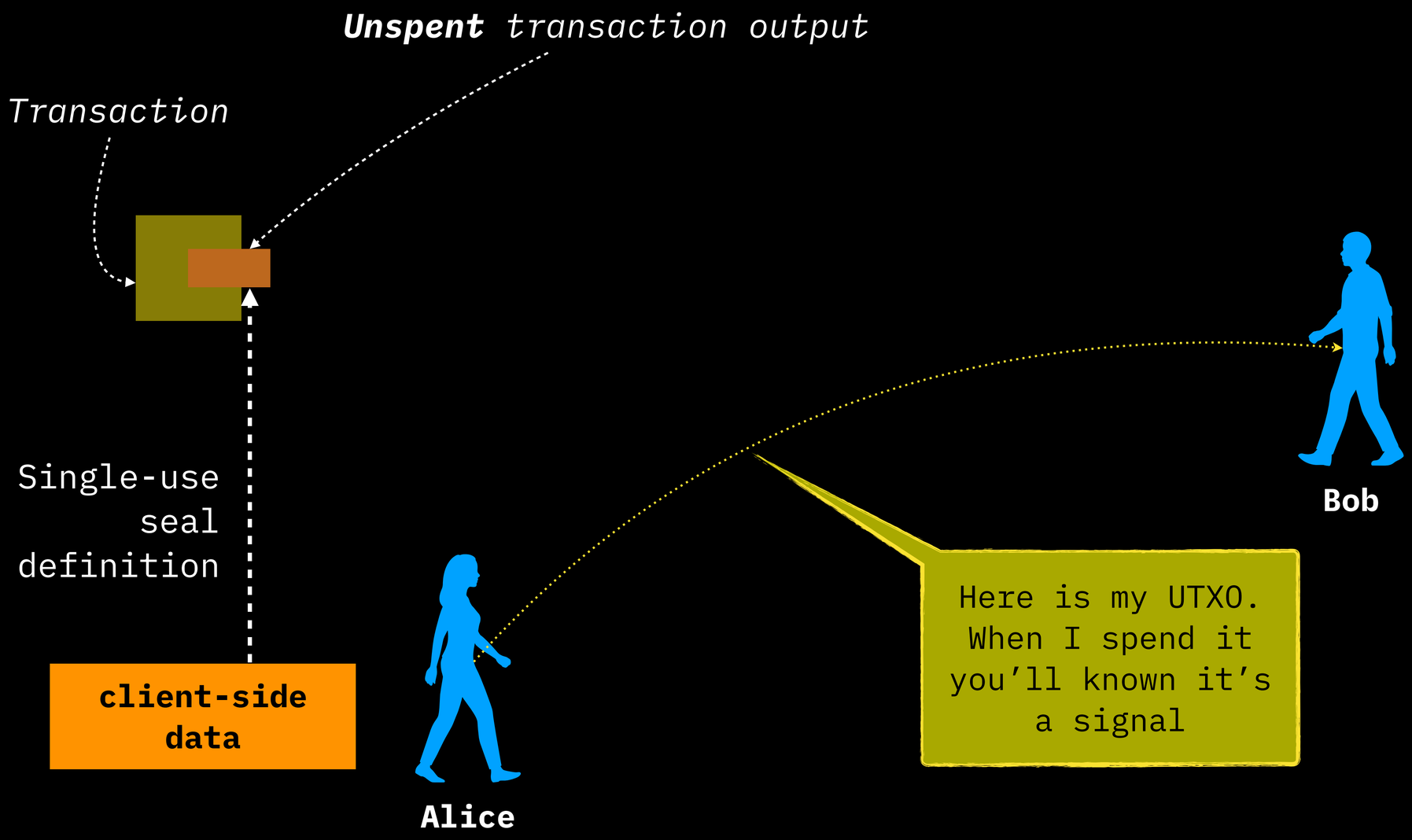
Bitcoin single-use-seal schemes

Scheme name	Seal definition	Seal closing	Additional requirements	Main application	Possible commitment schemes
PkO	Public key value	Transaction output	P2(W)PKH	none yet	Keytweak, taptweak, opret
TxO2	Transaction output		Requires deterministic bitcoin commitments	RGBv1 (universal)	
PkI	Public key value	Transaction input	Taproot-only & doesn’t work with legacy wallets	Bitcoin-based identities	Sigtweak, witweak
TxOI	Transaction output			none yet	

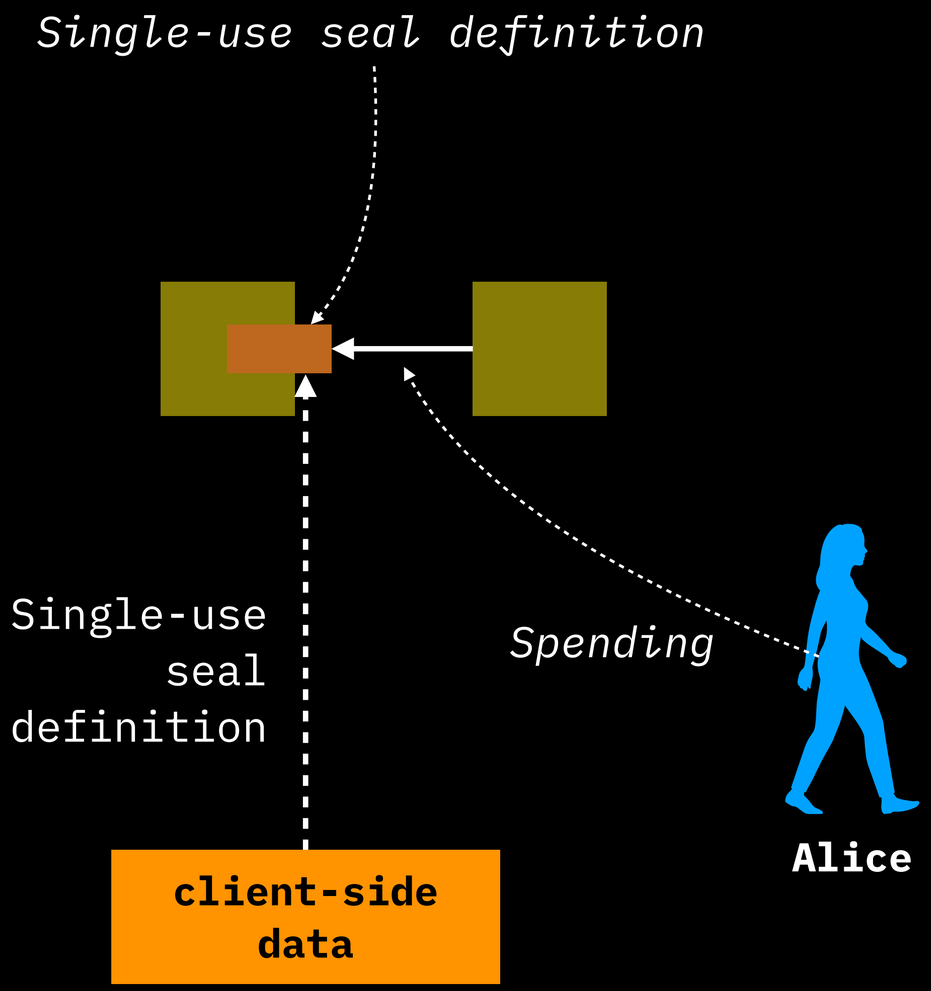
TxO-based single-use seals



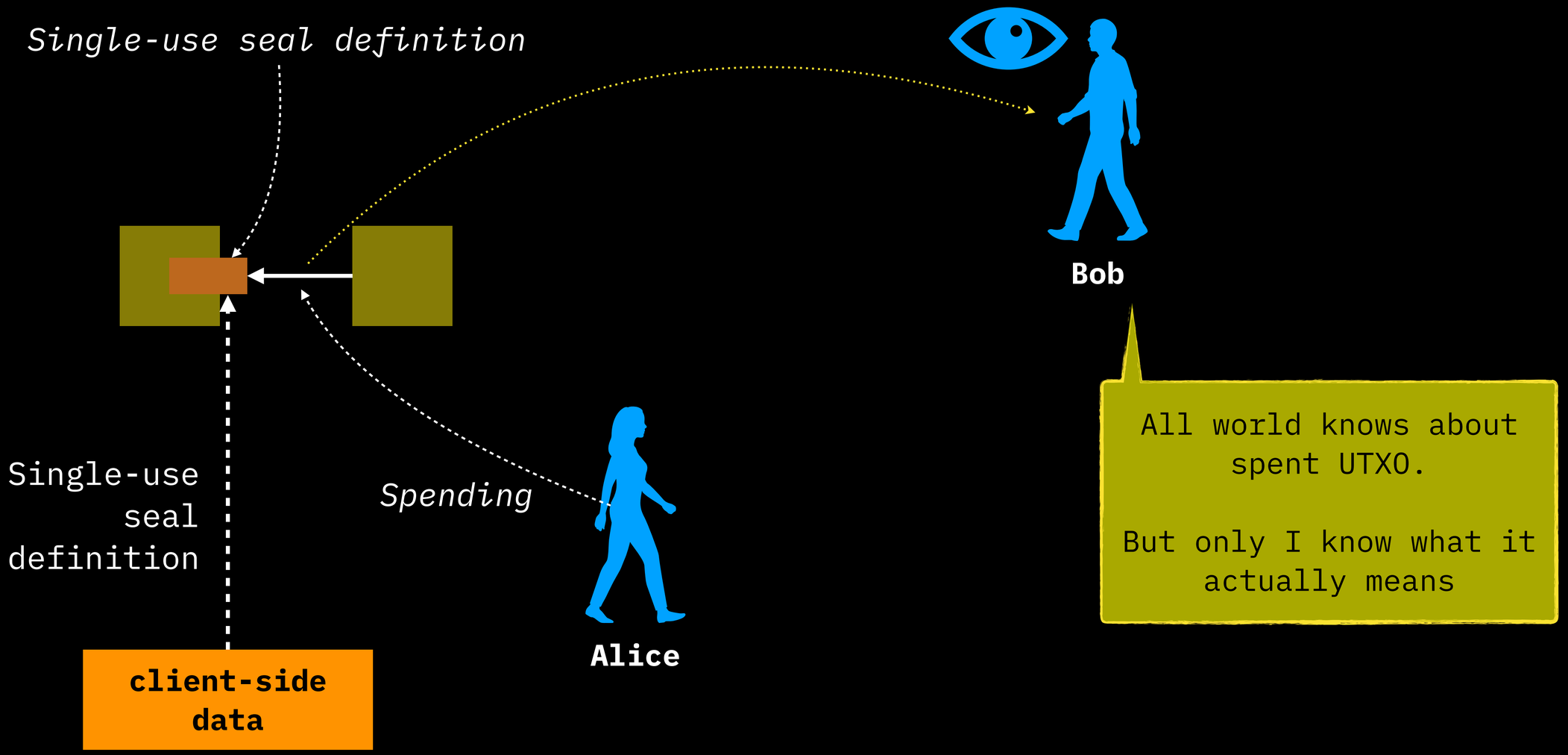
TxO-based single-use seals



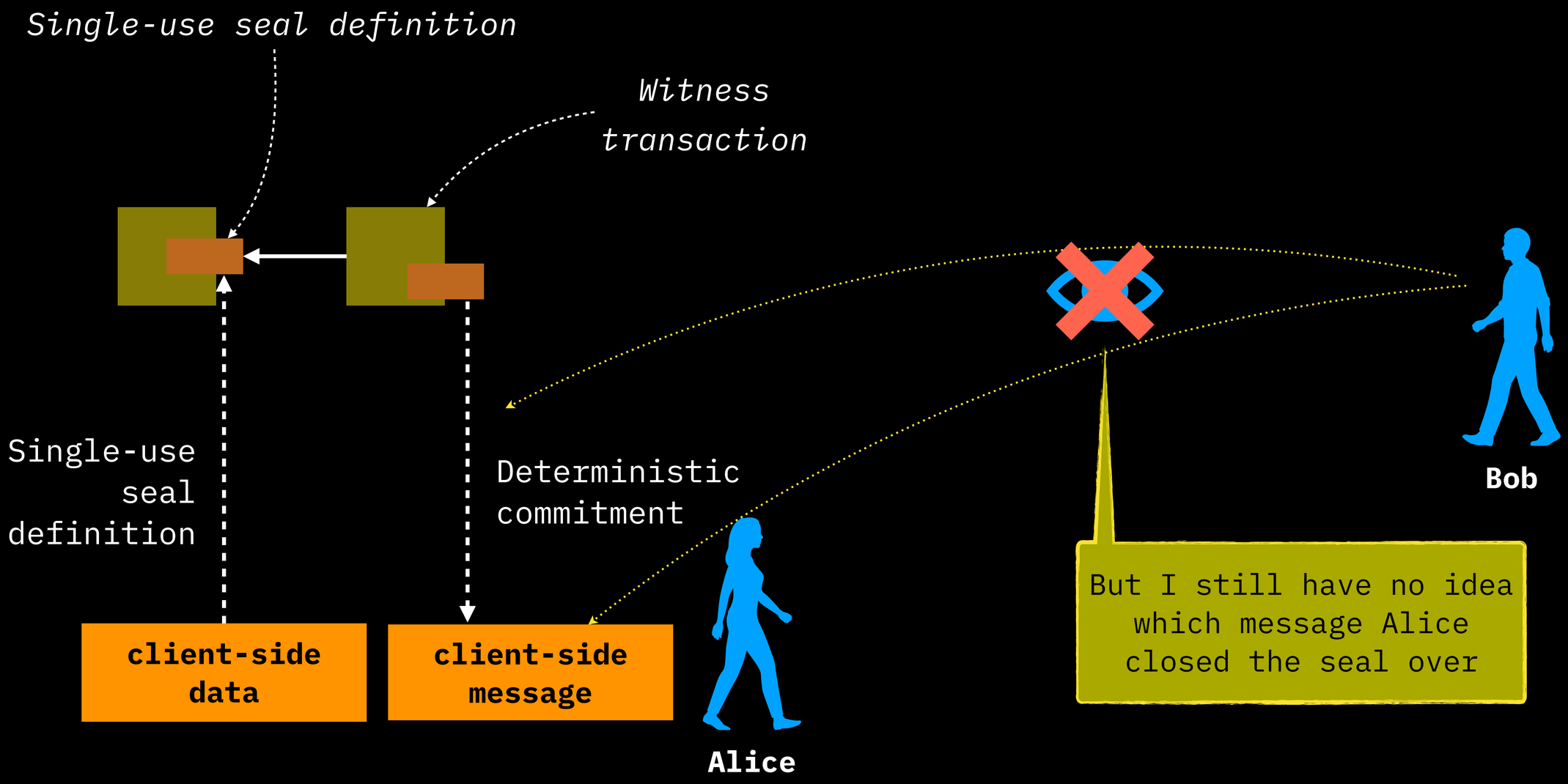
TxO-based single-use seals



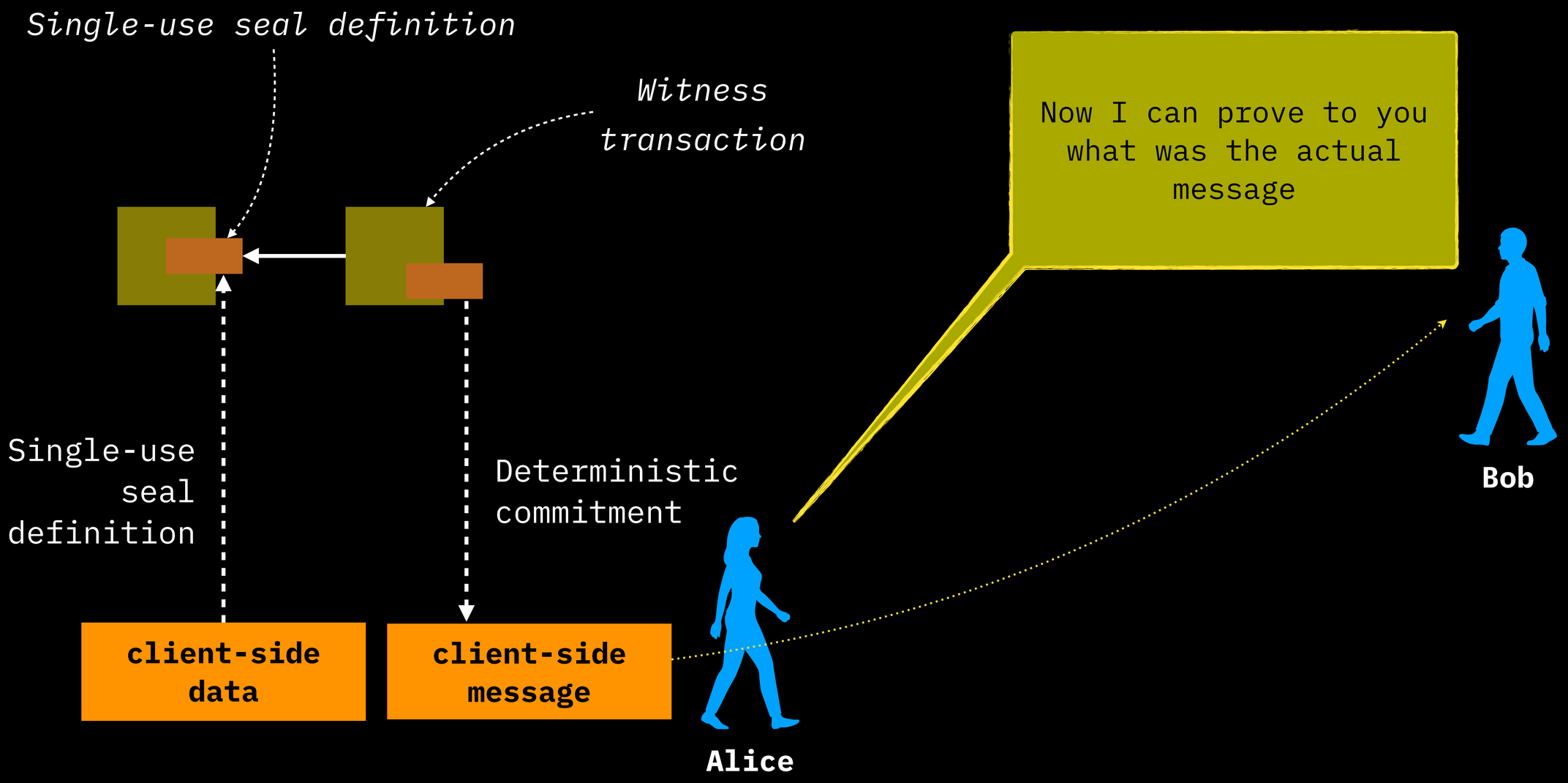
TxO-based single-use seals



TxO-based single-use seals



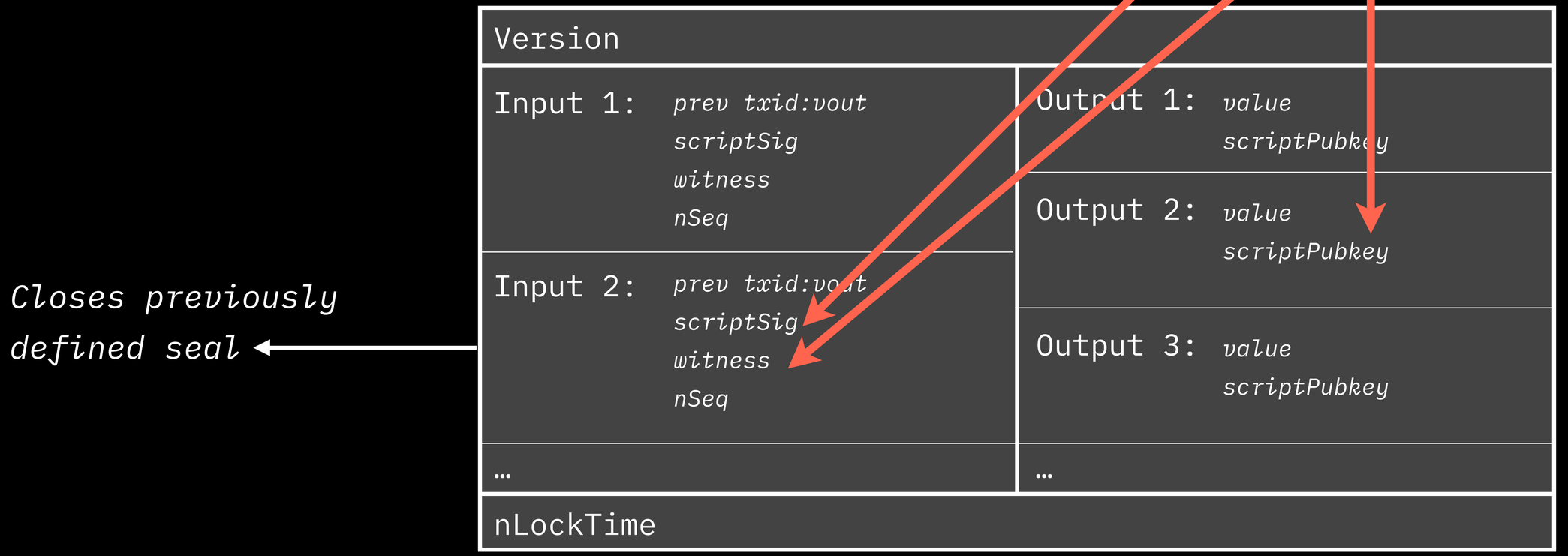
TxO-based single-use seals



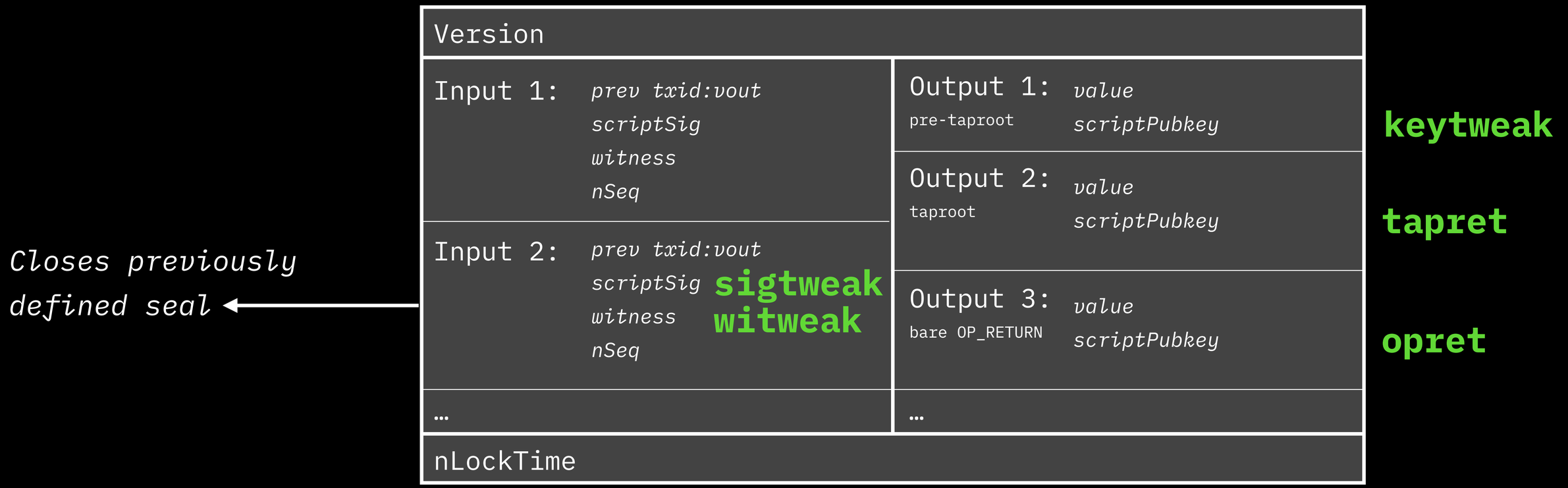
Determinism: witness transaction must provably contain only a single commitment

Closing seals with bitcoin transactions

Single-use-seal closing message commitment can go in here



Closing seals with bitcoin transactions



Tradeoff analysis

	Privacy & scalability	Inter-operability	Compatibility	Portability	Complexity
Keytweak (deterministic P2C)					
Sigtweak (deterministic S2C)					
Opret (OP_RETURN)					
Tapret algorithm: Top-left node					

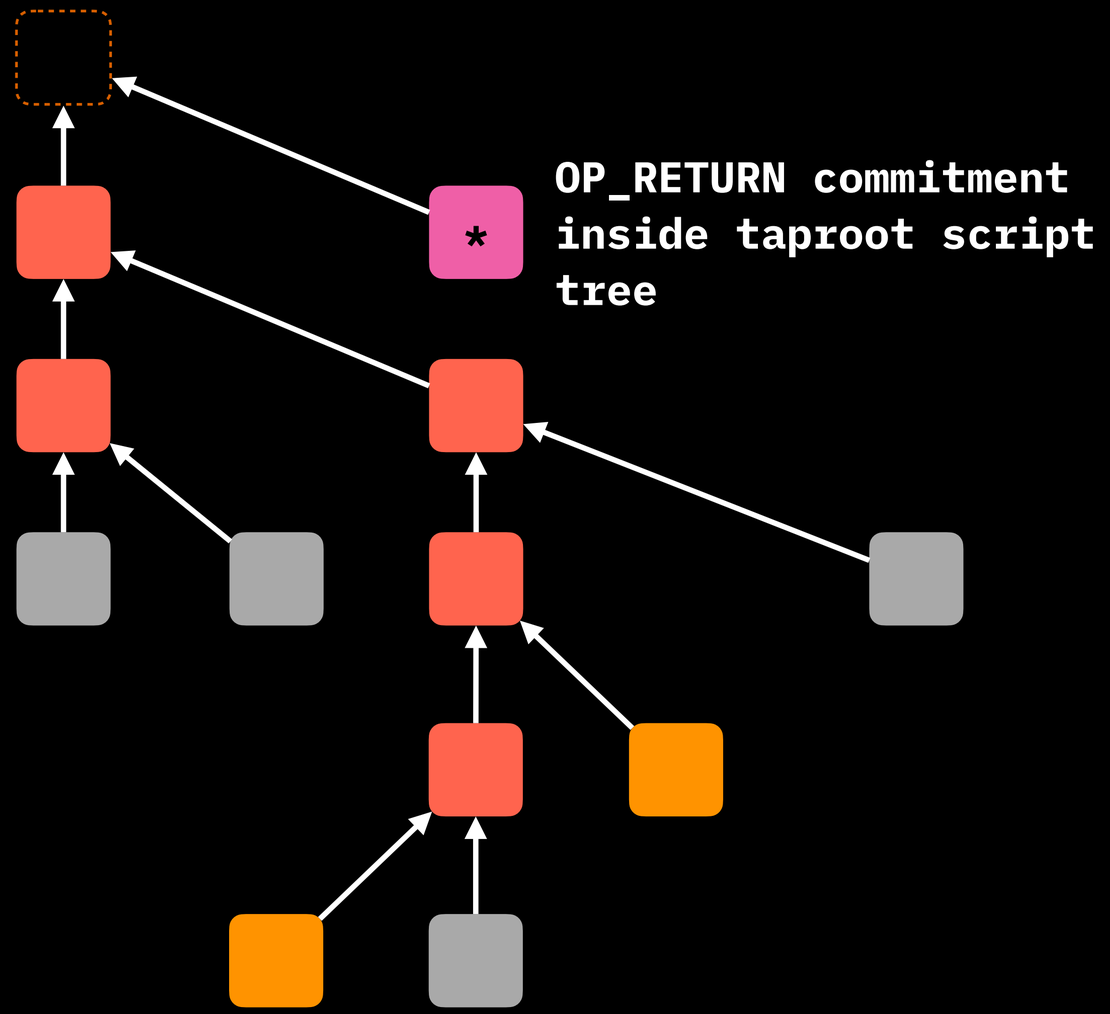
~~OP_RETURN~~

~~Pay-to-contract~~

~~Sign-to-contract~~

Tapret

Novel type of
private commitments
not using blockchain
space and affecting
keys/signatures

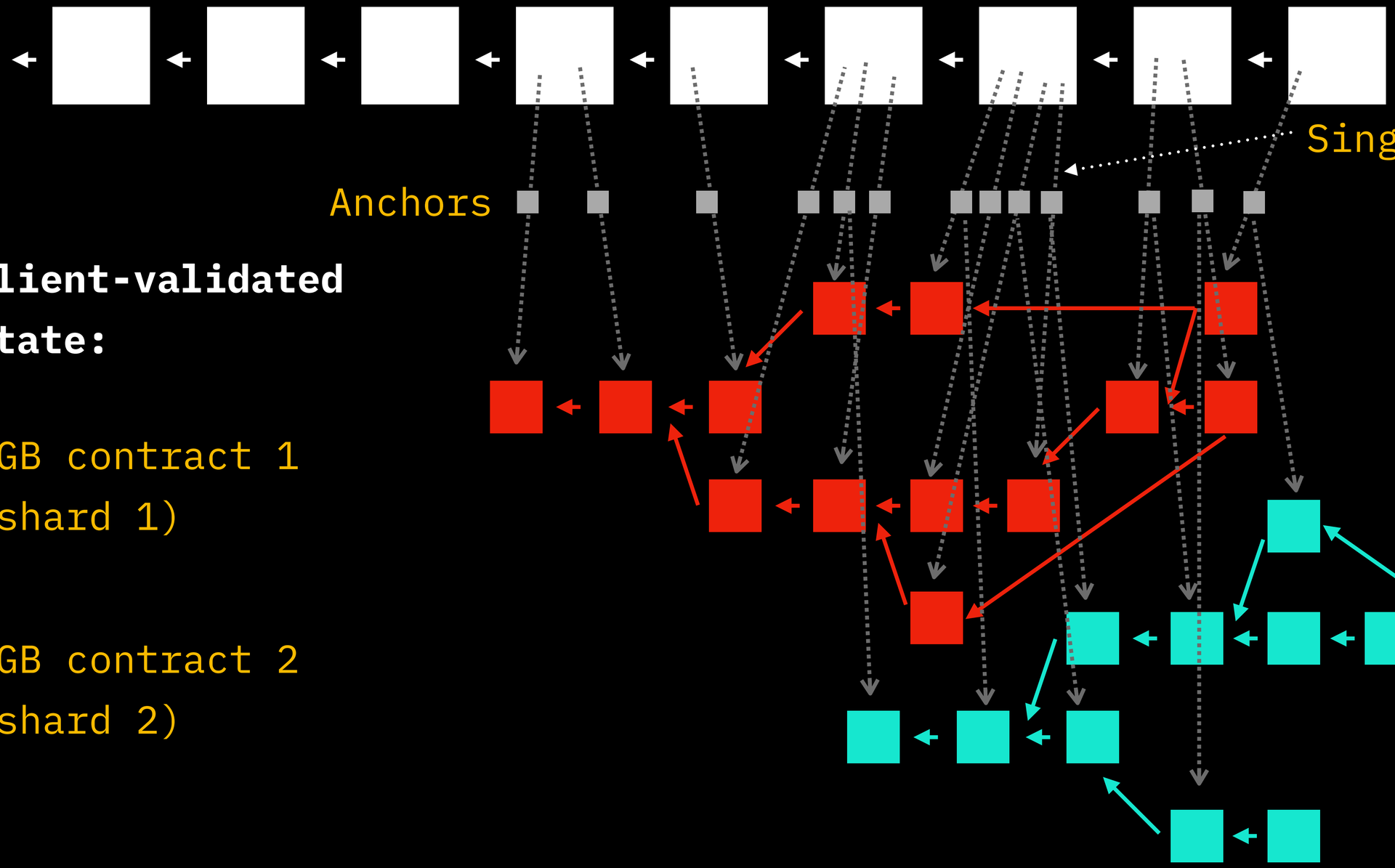


Multi-protocol commitments and anchors

Problem:

How to shard?

Bitcoin blockchain:
state ownership



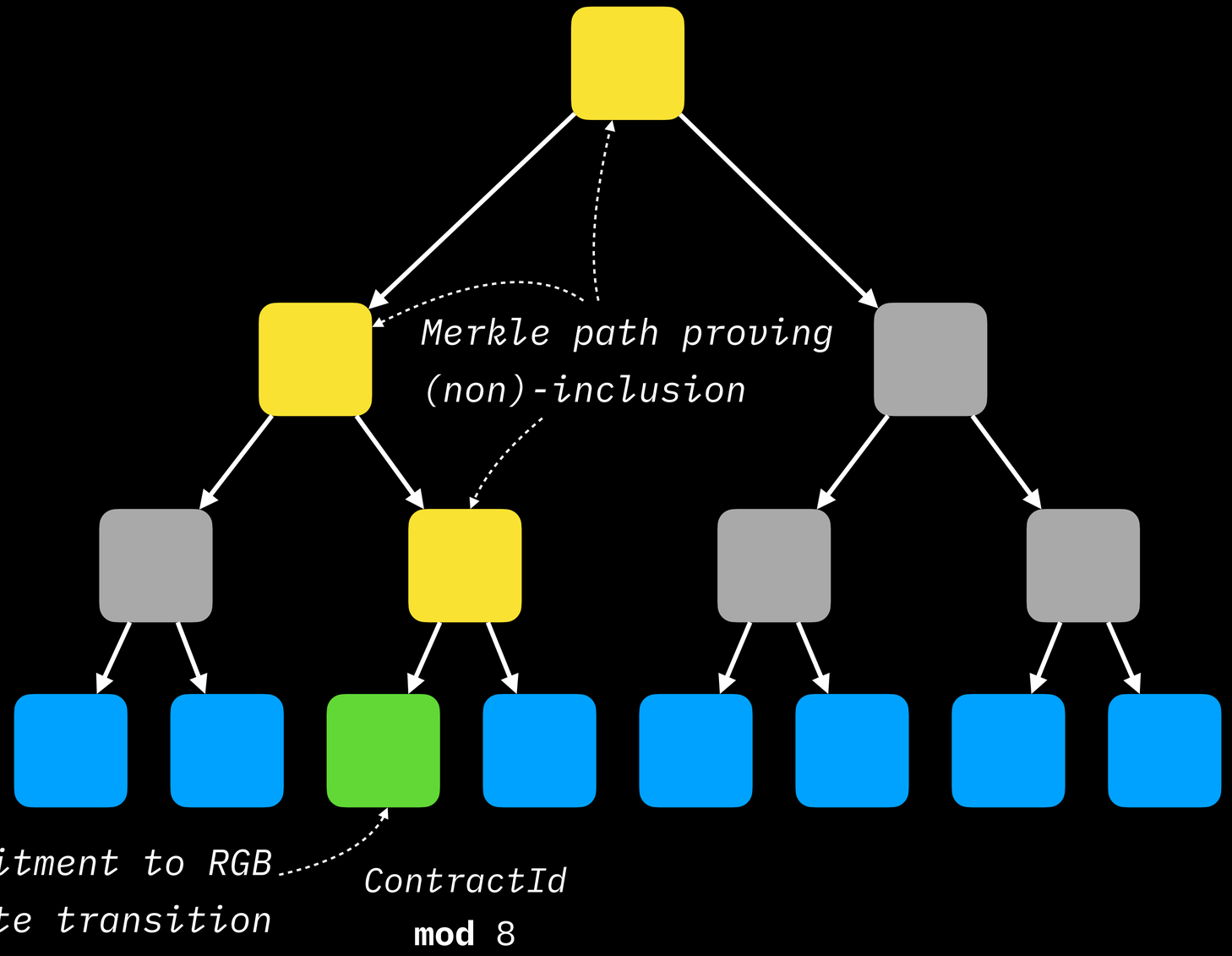
Client-validated state:

RGB contract 1
(shard 1)

RGB contract 2
(shard 2)

Multi-protocol commitments

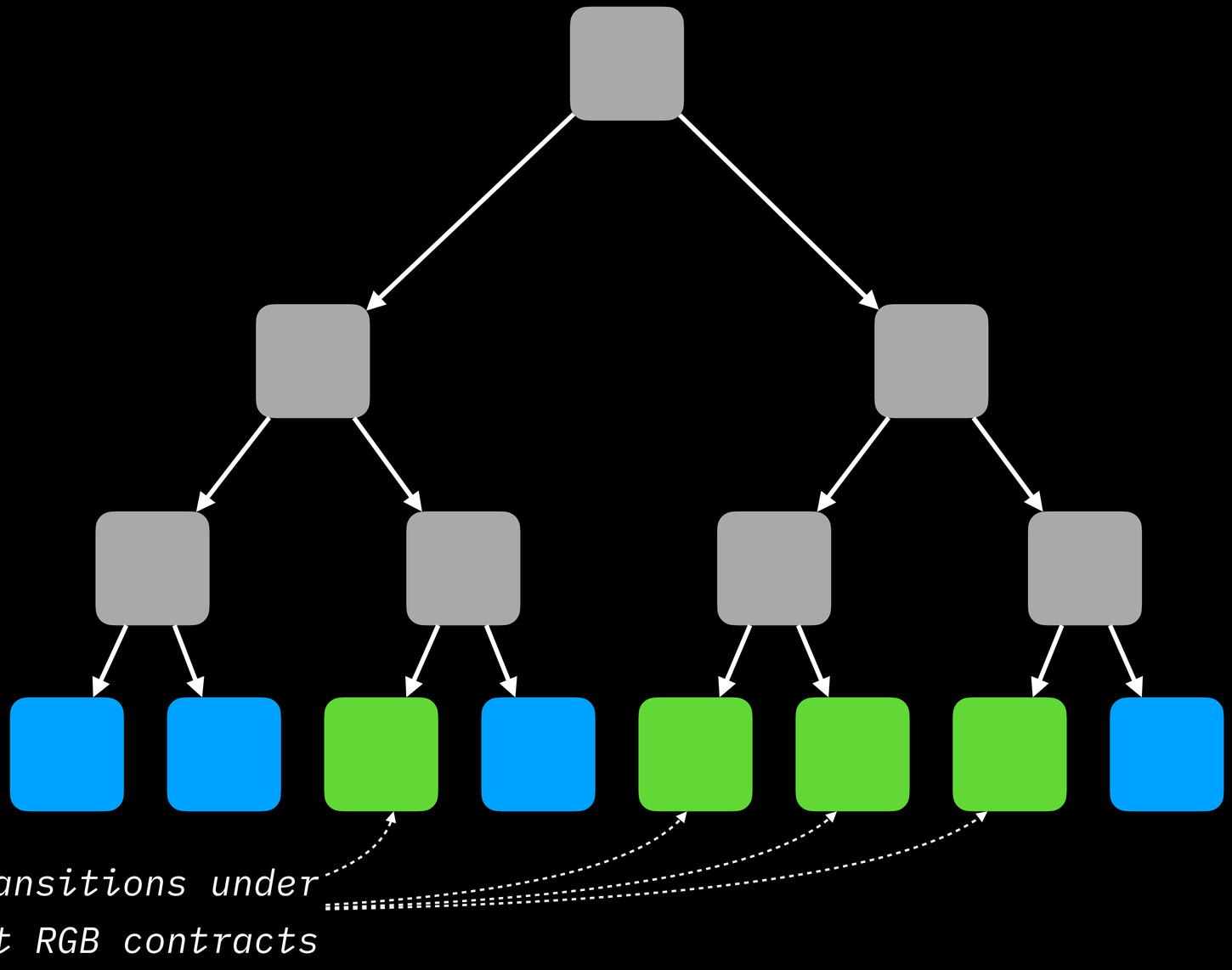
- Place commitments under different RGB contracts into a Merkle tree such that each RGB contract has a unique position



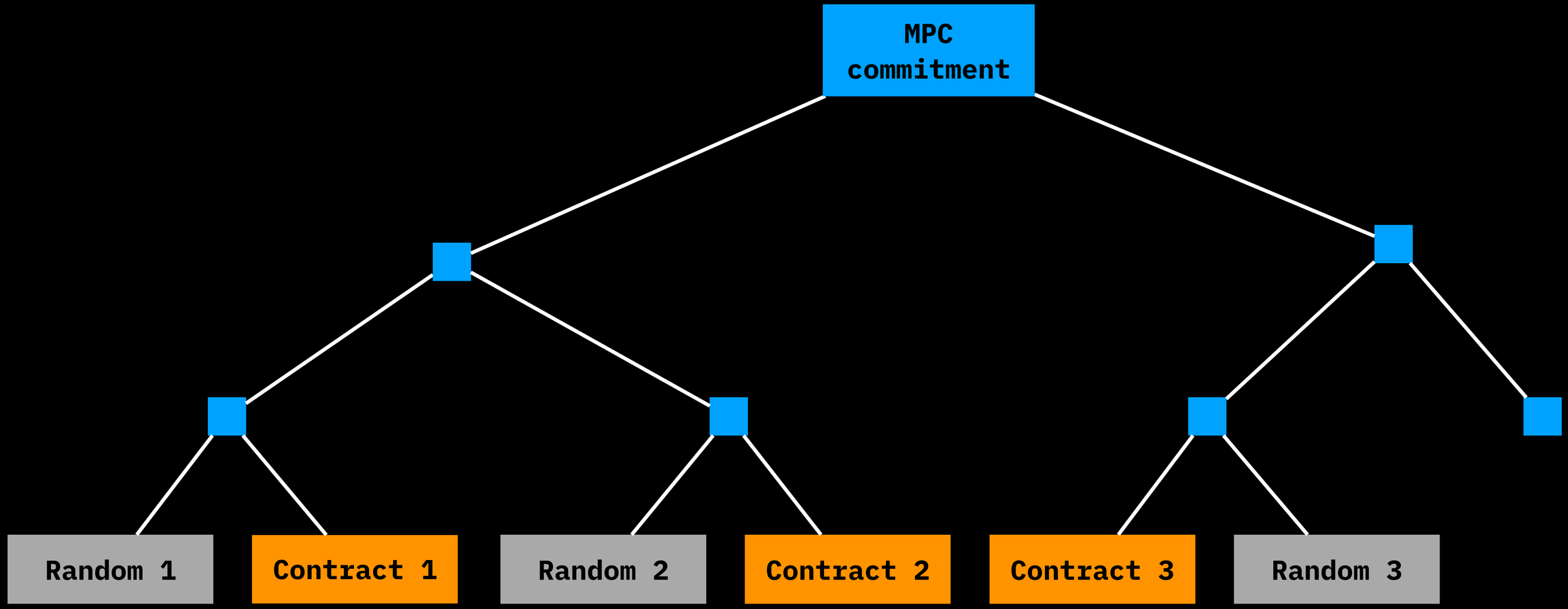
Producing MPCs

Find a width of a Merkle tree (w) such that for all RGB contract ids (c_i) all $w \bmod c_i$ values are different

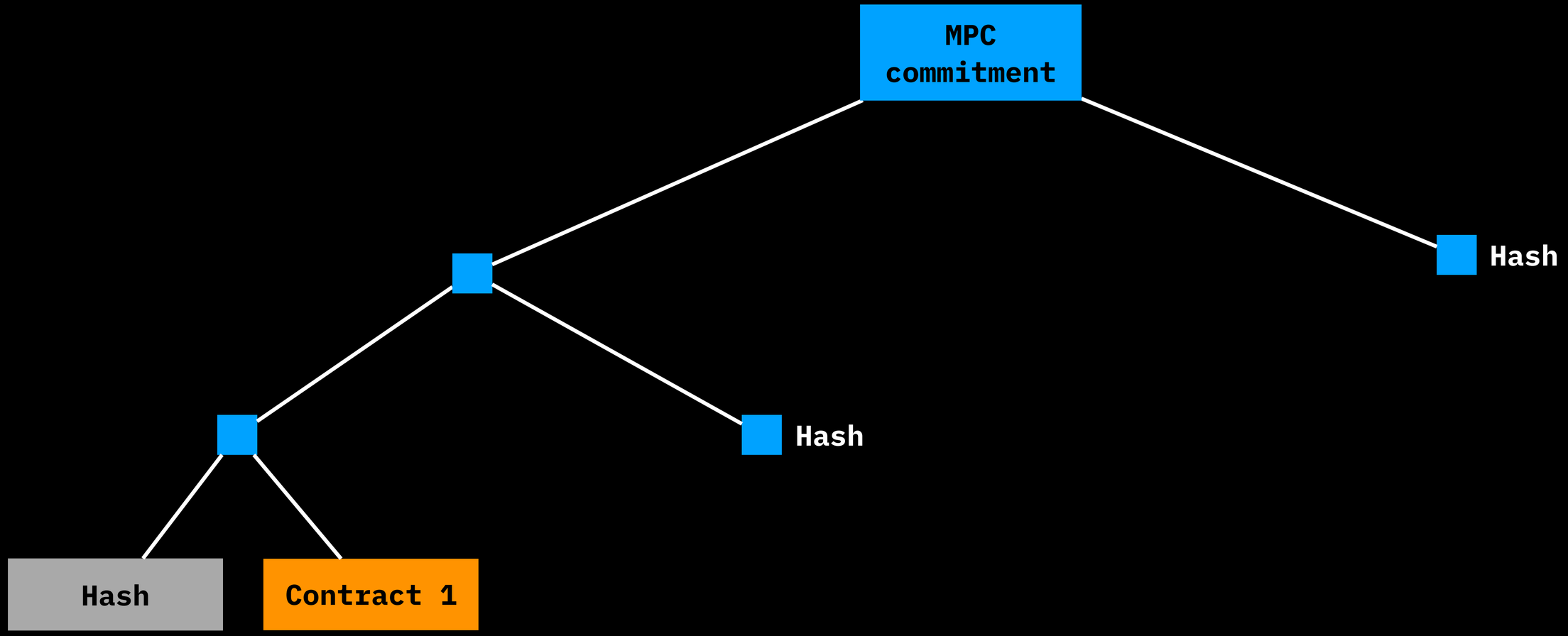
- Does a lot of SHA256 computations
- Can run on modern CPUs and GPUs
- CPU (Intel i9):
  ~10 mins for ~200k contracts



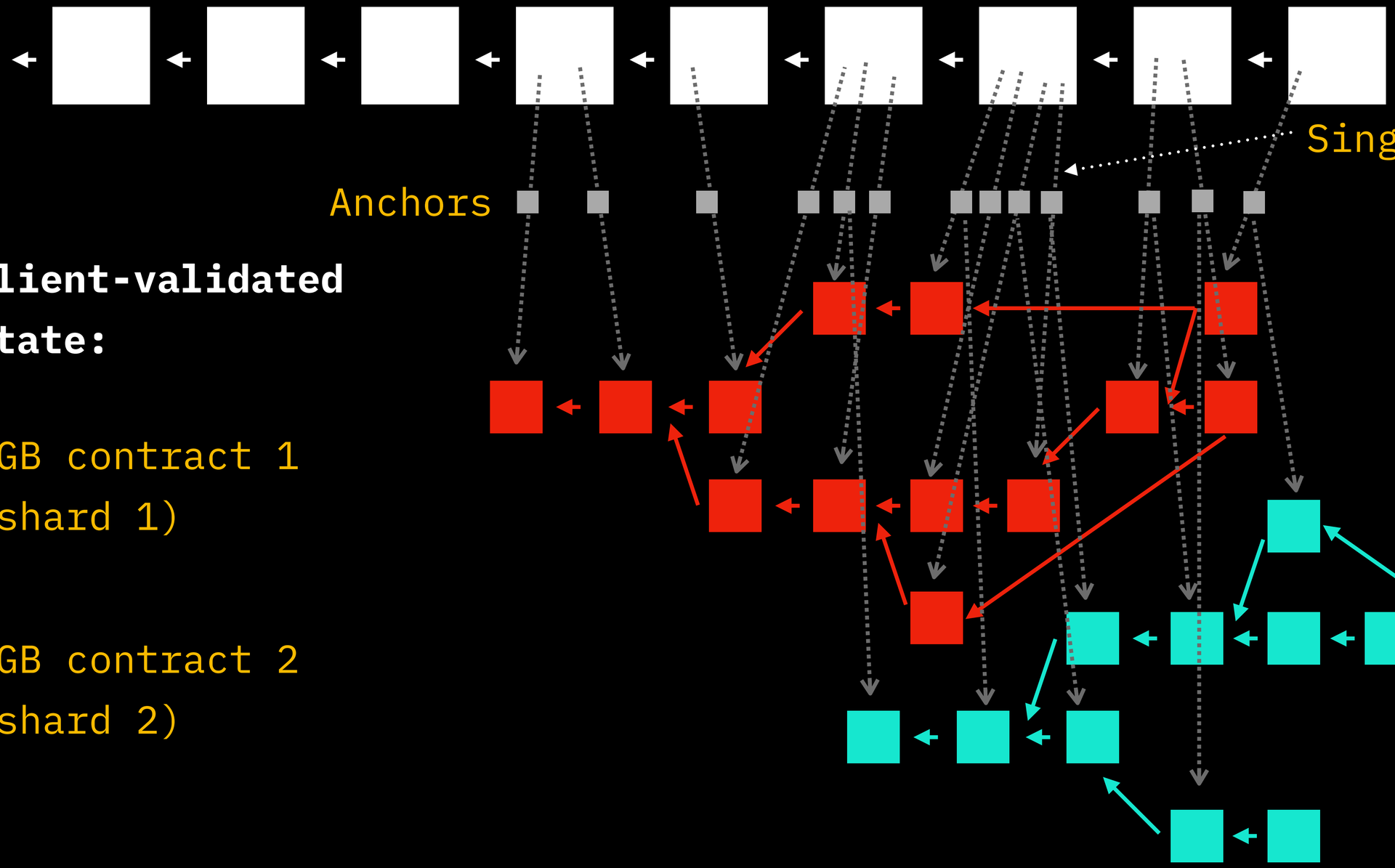
MPC Tree (as seen by creator)



MPC Proof (present to verifier)



Bitcoin blockchain:
state ownership



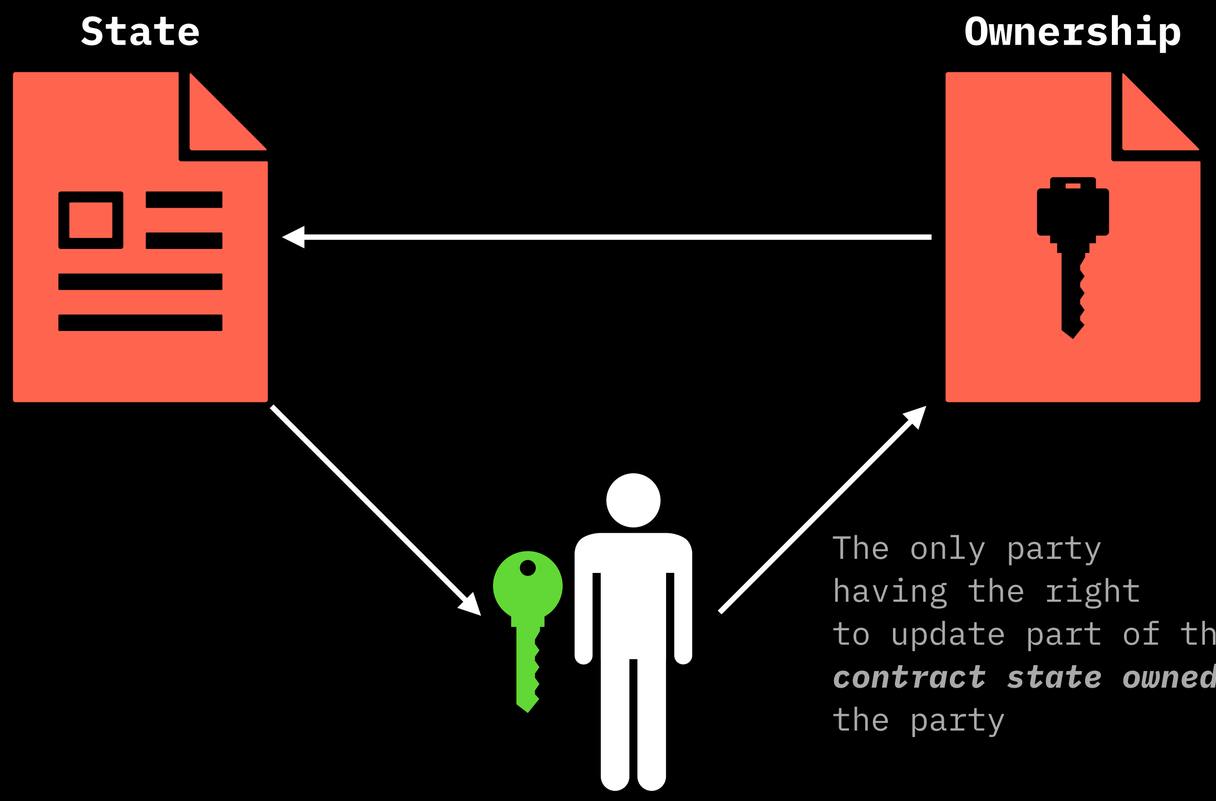
Client-validated state:

RGB contract 1
(shard 1)

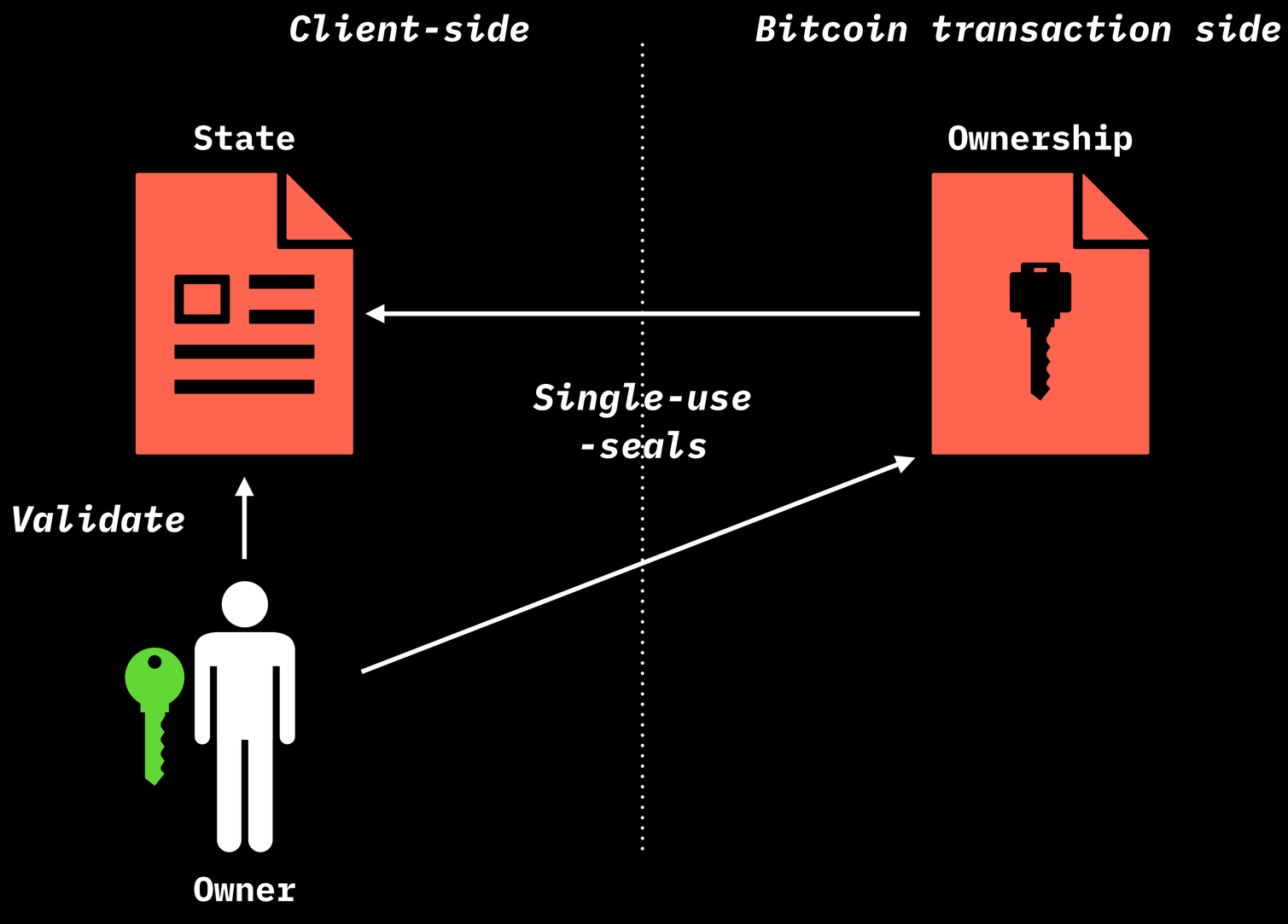
RGB contract 2
(shard 2)

Smart contracts

Separated state and ownership



The state exists client-side



Validation != computing

State ownership != state validation

- Ownership defines **WHO** can change the state: **Bitcoin**
- Client-side validation define **HOW** it may change: **RGB**

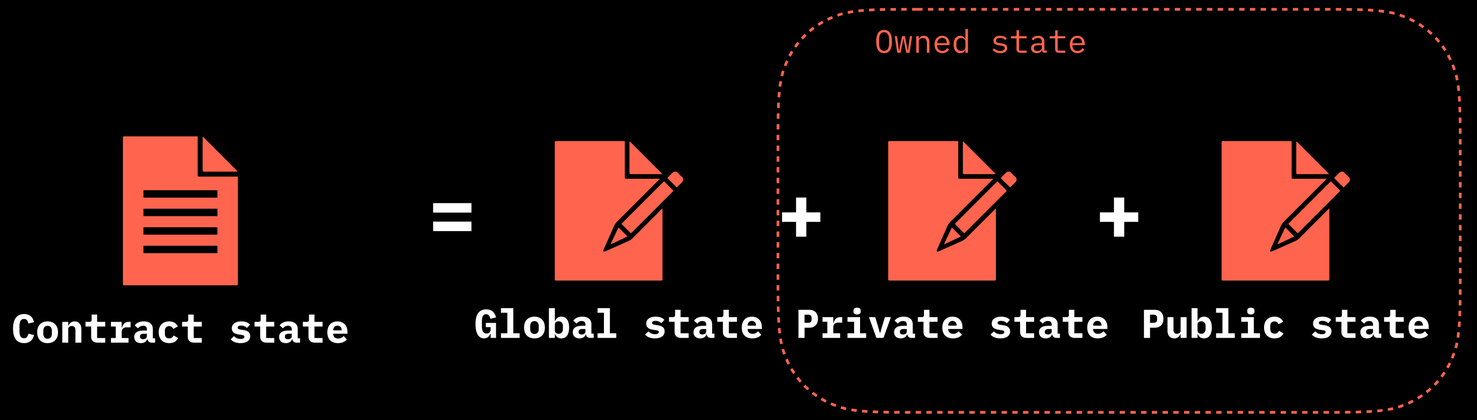
- Avoids mistakes by “blockchain smart contracts” (Ethereum etc): mixing Turing completeness into non-scalable blockchain
- Makes possible for smart contracts to go into Lightning Network

RGB validation:

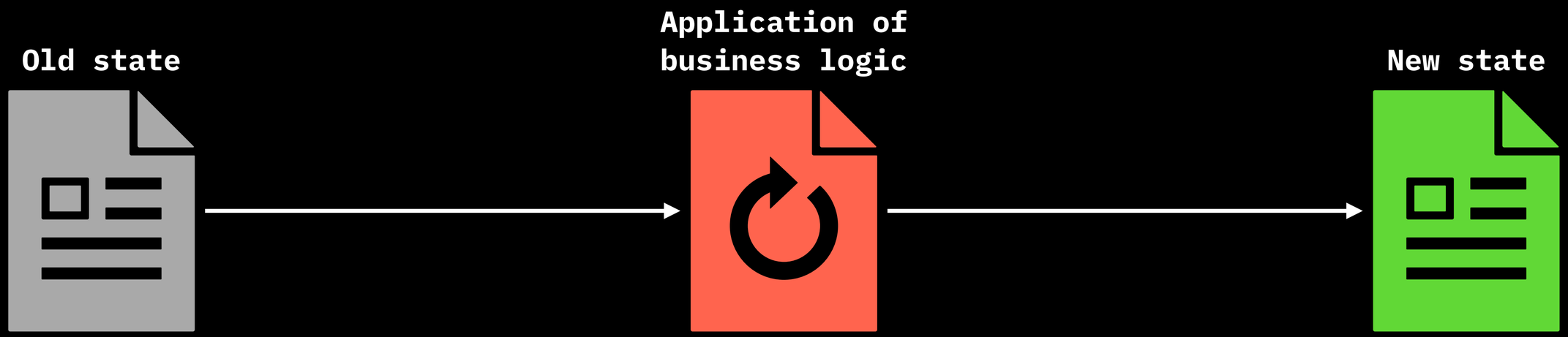
- Ownership & “double-spent” prevention: re-using **Bitcoin script** combined with **single-use-seal validation**
- Consistency & completeness: **Schema**
- Changes in state: **AluVM**
  you can learn more about AluVM and its use in RGB at
  - *https://github.com/LNP-BP/presentations/blob/master/Presentation%20slides/Single-use-seals.pdf*
  - *https://youtu.be/brfWta7XXFQ*

Public state vs private state

- **Owned state:** someone owns
  - **Public**: everybody knows (must always been included in transfers)
  - **Private**: nobody else knows (included only if it is a part of the history)
- **Global state:** nobody owns, everybody knows (always public)



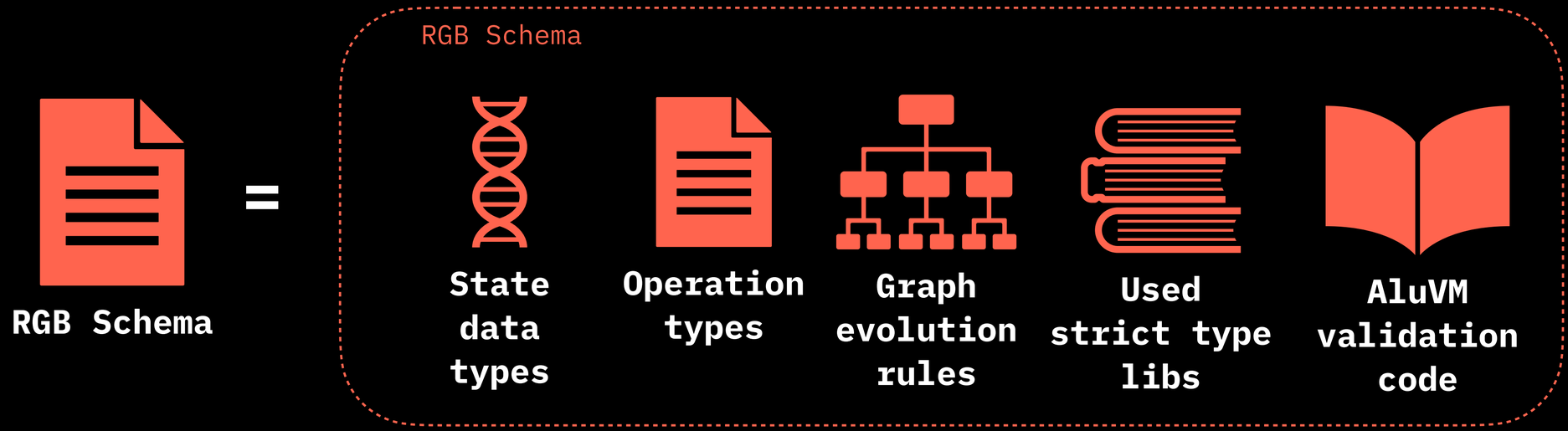
State transitions: nodes of contract evolution DAG



State transition

RGB contract schema defines:

- Which types of state can exist
- Data types used in state
- Which operations can be performed with the contract
- Validation rules and scripts



Contract definition

=

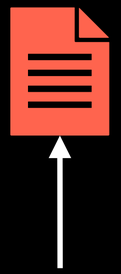
Schema

Requirements for the contract state + contract business logic

"Class" in terms of OOP

+

Interface



Interface implementation

Bindings to human & wallet-readable names from the interface

Implementation of an interface/trait for a class/struct

+

Genesis

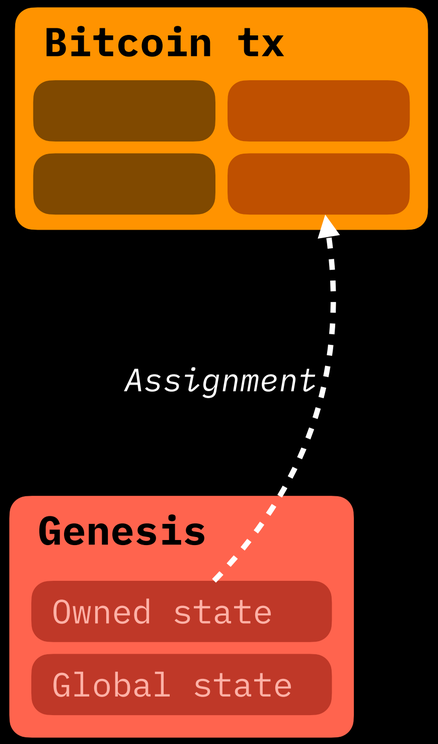
Initial setup of the contract state

Instance of a class created by the class constructor

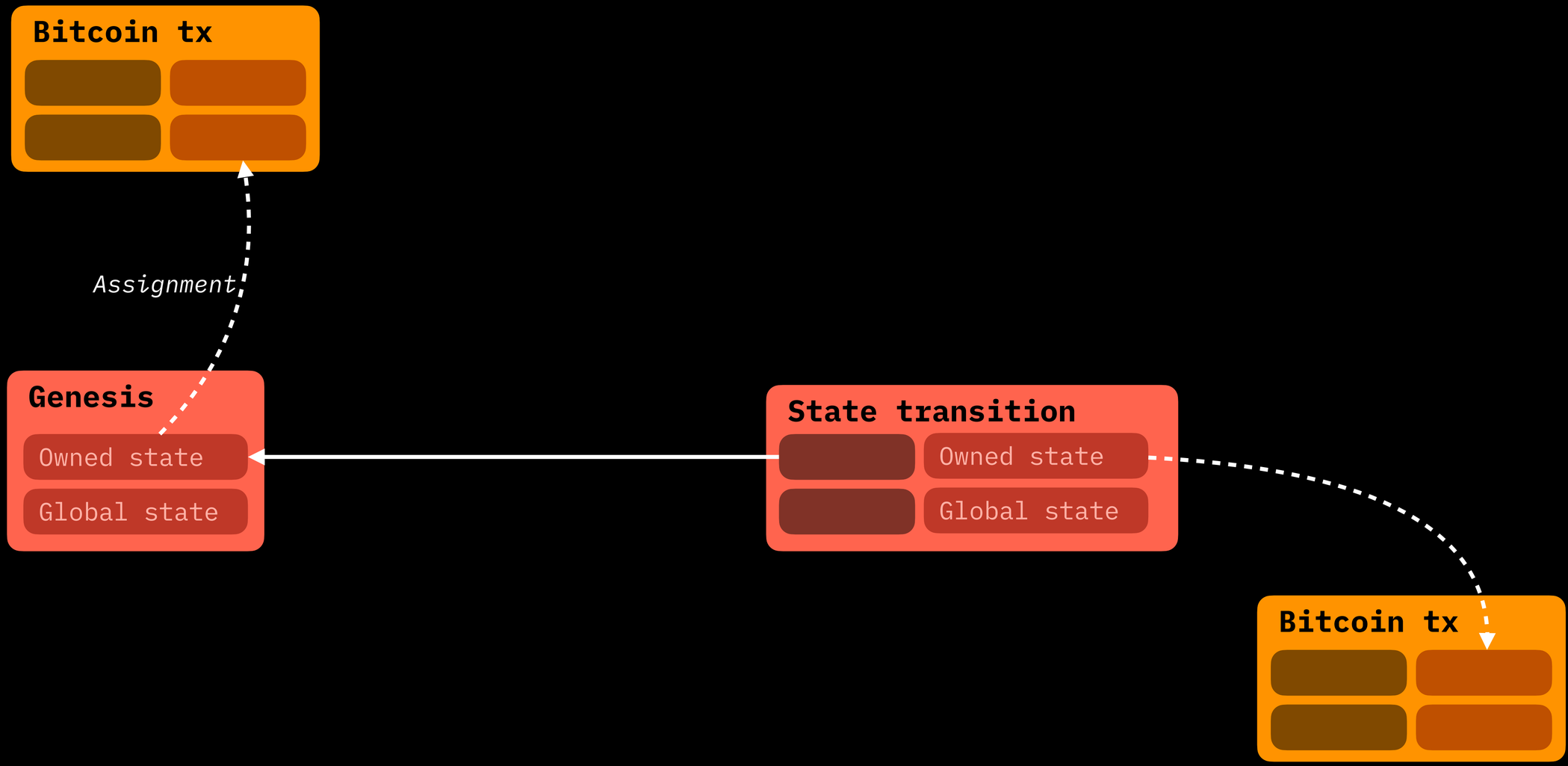
Contract components overview:

Contract components	Meaning	OOP terms	Ethereum terms
Interface	*Contract semantics*	Interface (Java), trait (Rust), protocol (Swift)	ERC* standards
Schema	*Contract business logic*	Class	Contract
Interface implementation	*Mapping semantics to business logic*	Impl (Rust), Implements (Java)	ABI
Genesis	*Initial contract state*	Class constructor	Contract constructor

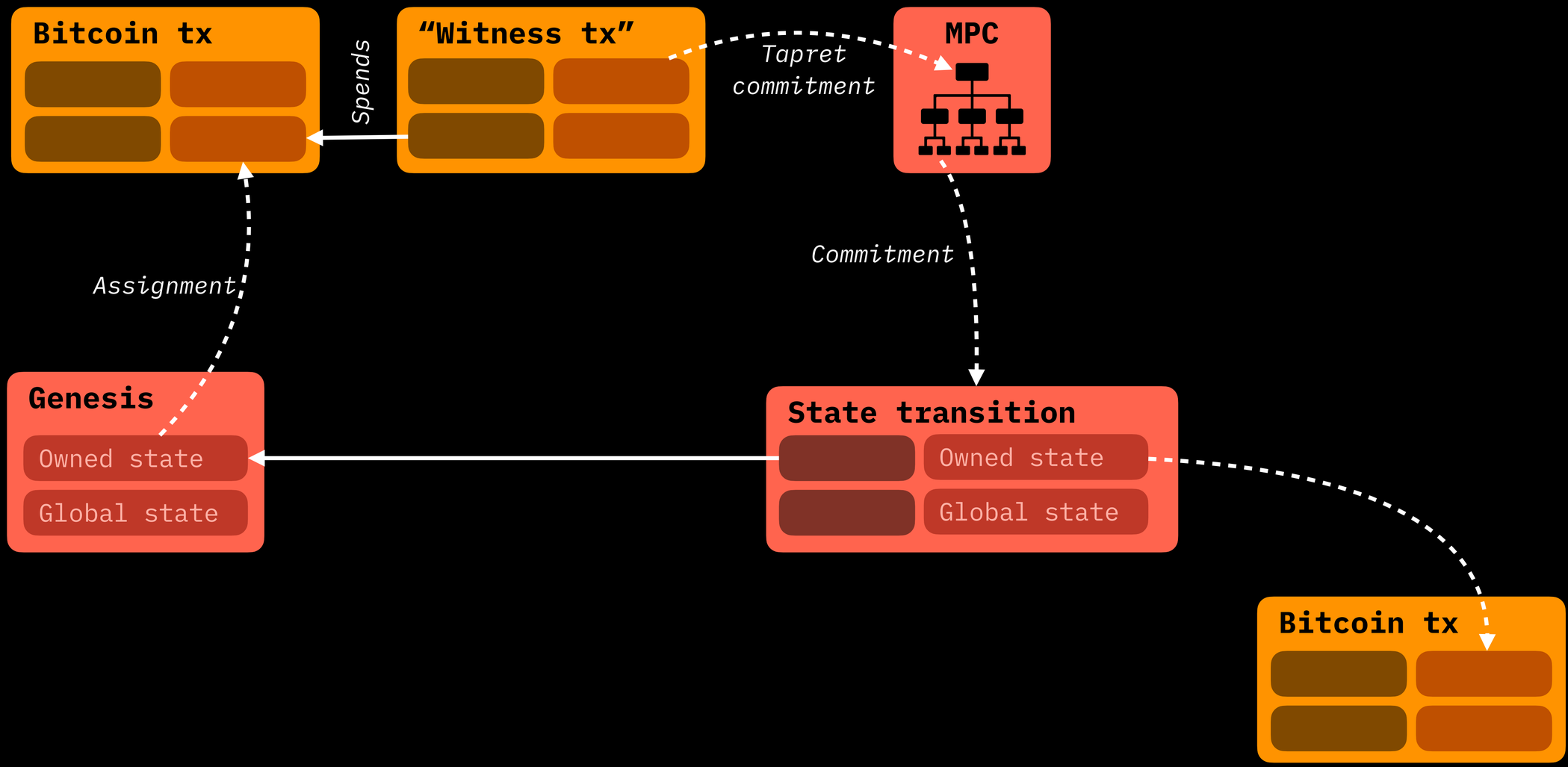
Anatomy of RGB operation



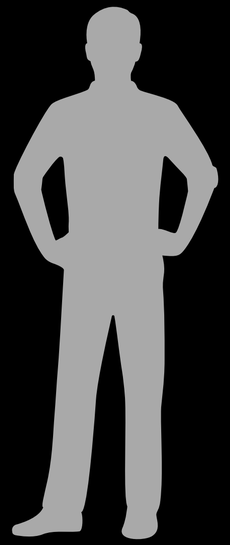
Anatomy of RGB operation



Anatomy of RGB operation



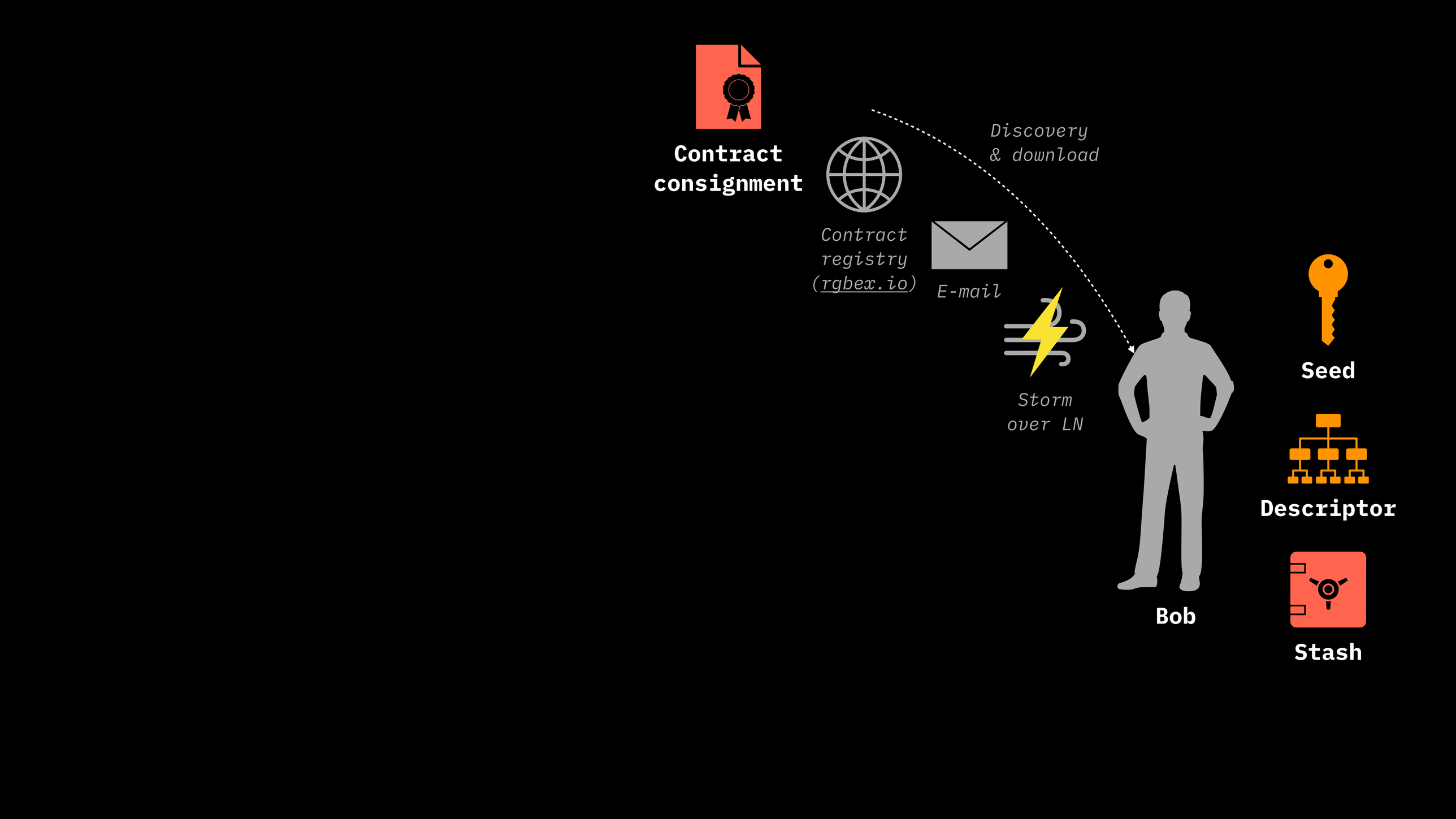
Transfers



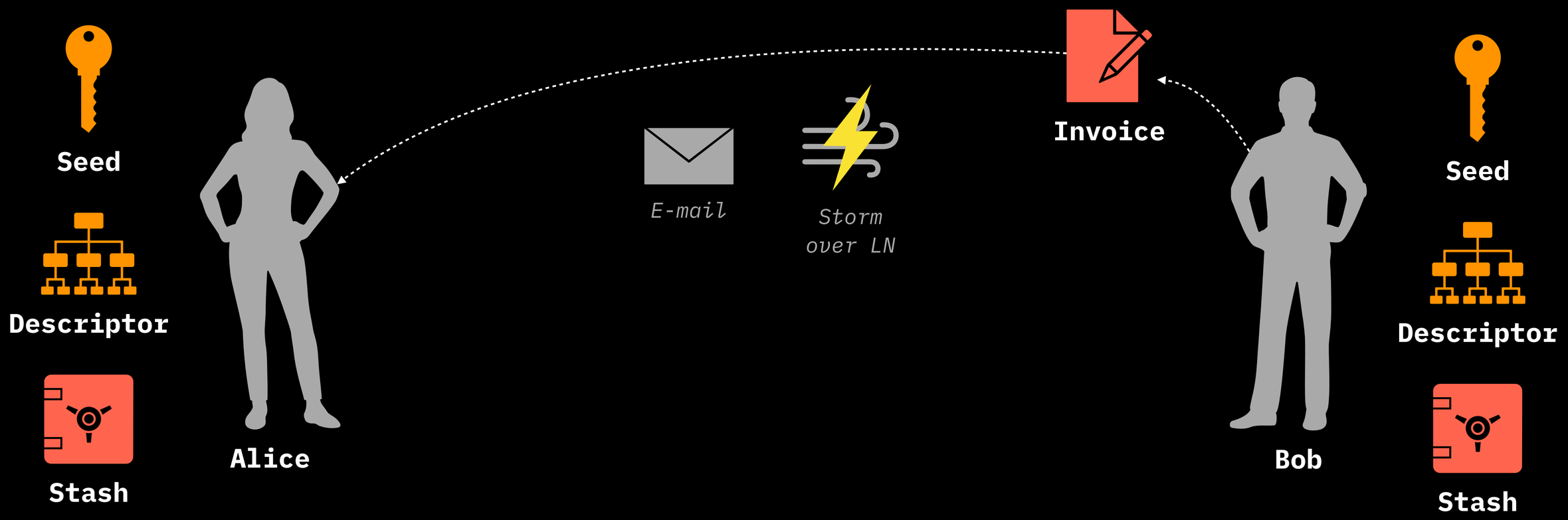
Bob

Seed

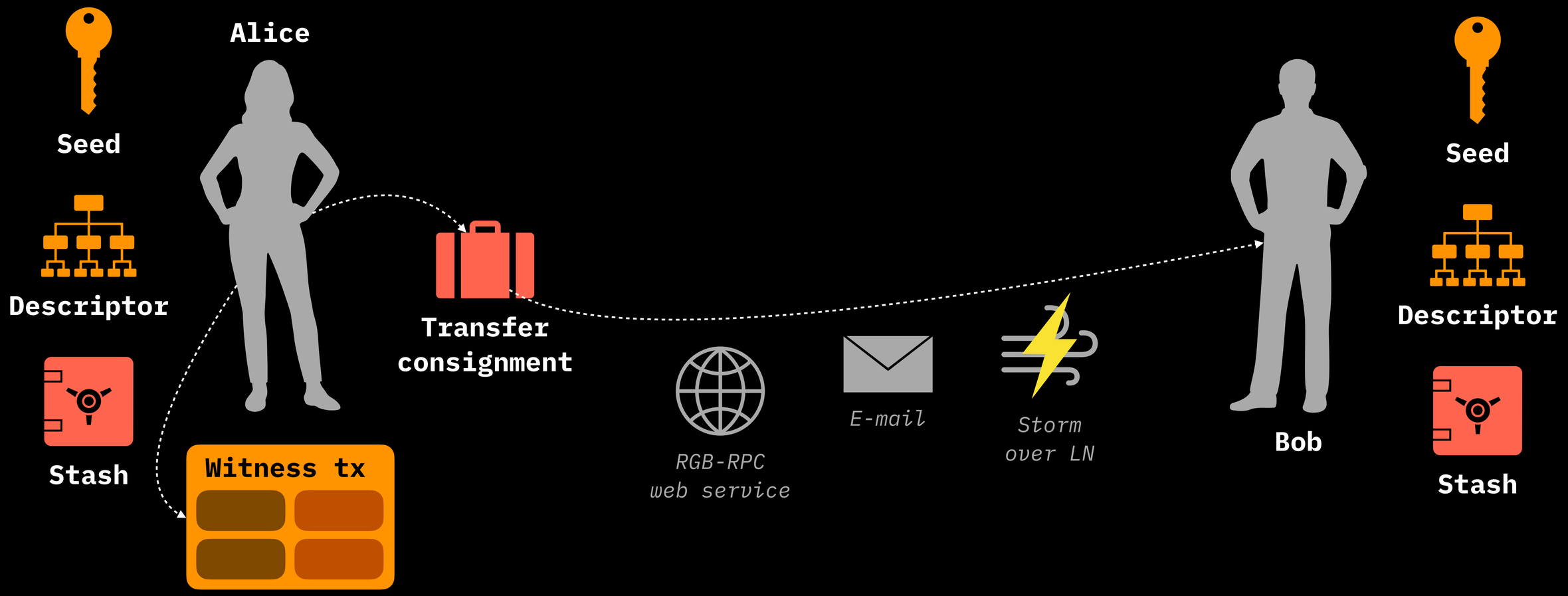
Descriptor


Contract consignment
Discovery & download
Contract registry (rgbex.io)
E-mail
Storm over LN
Bob
Seed
Descriptor
Stash

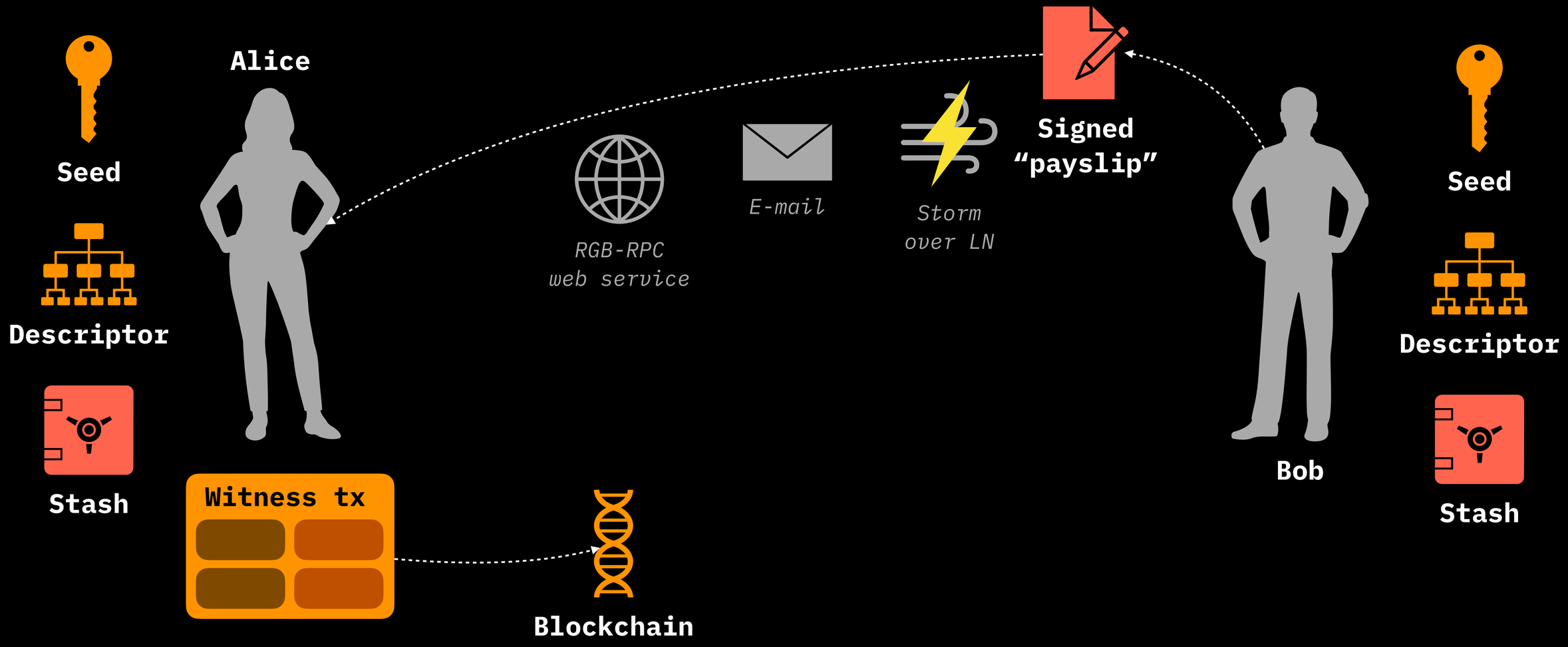
Payment round 1



Payment round 2



Payment round 3 (optional)



$ **rgb**

One command to rule all contracts

$ **git**

it should work just like git

- See what contracts do you have:
  $ rgb contracts
- Issue an asset:
  $ rgb issue rgb20 <schema> <genesis-file>
- Check contract state:
  $ rgb state <short-contract-name>
- Check contract state for a wallet descriptor:
  $ rgb state <short-contract-name> <descriptor>

```
$ cargo install rgb-contracts --all-features
```

How to install