LNP/BP Standards Association

RGB smart contracts

formal definition of RGB computational model (PRISM)

Dr Maxim Orlovsky,

Pandora Core AG

RGB is the first client-side-validated
smart contracting system

What is client-side-validation?

- Distributed computing systems operating on layer 2/3
  using layer 1 for state verification
  (any layer above 1 uses consensus from layer 1)

- Both an alternative (as layer 2) or an extension (as layer 3) to state channels (lightning channels etc)

- Coined by Peter Todd as a logical extension of his work on OpenTimeStamps

Not client-side-validation

Client-side data, but not client-side-validation:

- Bitcoin P2SH and P2TR requiring keeping script source on client side
- MuSig2 requiring client-side state
- State channels (LN, DLCs etc)

Client-side-validation is when you **validate** client-side data against certain rules, which **must include blockchain-based commitments** (timestamps or single-use-seals)

Wait, what are state channels?

- Lightning channels
  (organized into a payment network with payment routing)
- Discrete log contracts
  (oracle-based future contracts utilizing scriptless scripts)
- Storm
  (escrow-based trustless storage using zk-proofs)
- Prometheus
  (arbitration-based trustless distributed computing)

…and yes, you can do one state channels on top of other (DLCs over LN)

State channels

Client-side-validation

similarities and differences: neutral negative positive

- Works with blockchain and, sometimes, other state channels

- Works with both blockchain and any other state channels

State channels

- Works with blockchain and, sometimes, other state channels
- Synchronous

Client-side-validation

similarities and differences: neutral negative positive

- Works with both blockchain and any other state channels
- Asynchronous (network fault tolerant)

State channels

Client-side-validation

similarities and differences: neutral negative positive

- Works with blockchain and, sometimes, other state channels
- Synchronous
- May require routing (lightning channels)

- Works with both blockchain and any other state channels
- Asynchronous (network fault tolerant)
- May require storage providers

State channels

Client-side-validation

similarities and differences: neutral negative positive

- Works with blockchain and, sometimes, other state channels
- Synchronous
- May require routing (lightning channels)
- Uses client-side data (signatures or transactions)

- Works with both blockchain and any other state channels
- Asynchronous (network fault tolerant)
- May require storage providers
- Uses client-side data (any form fo complex state)

State channels

- Works with blockchain and, sometimes, other state channels
- Synchronous
- May require routing (lightning channels)
- Uses client-side data (signatures or transactions)
- Tiny size of client-side data

Client-side-validation

similarities and differences: neutral negative positive

- Works with both blockchain and any other state channels
- Asynchronous (network fault tolerant)
- May require storage providers
- Uses client-side data (any form fo complex state)
- Huge size of client-side-data

State channels

- Works with blockchain and, sometimes, other state channels
- Synchronous
- May require routing (lightning channels)
- Uses client-side data (signatures or transactions)
- Tiny size of client-side data
- Security requires watchtowers

Client-side-validation

similarities and differences: neutral negative positive

- Works with both blockchain and any other state channels
- Asynchronous (network fault tolerant)
- May require storage providers
- Uses client-side data (any form fo complex state)
- Huge size of client-side-data
- No watchtowers required (*when not on top of state channels)

State channels

- Works with blockchain and, sometimes, other state channels
- Synchronous
- May require routing (lightning channels)
- Uses client-side data (signatures or transactions)
- Tiny size of client-side data
- Security requires watchtowers

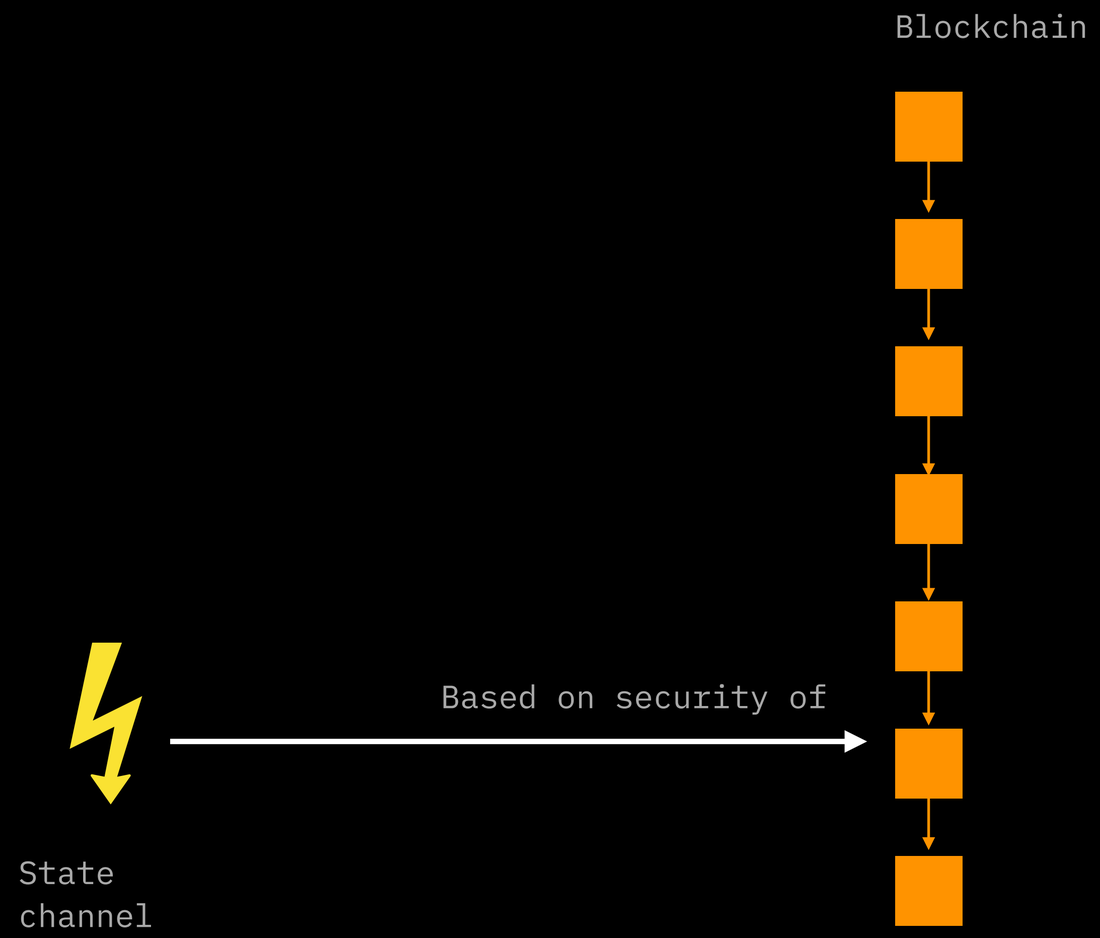
- No state validation (outside of blockchain-based mining & transaction validation scopes)

Client-side-validation

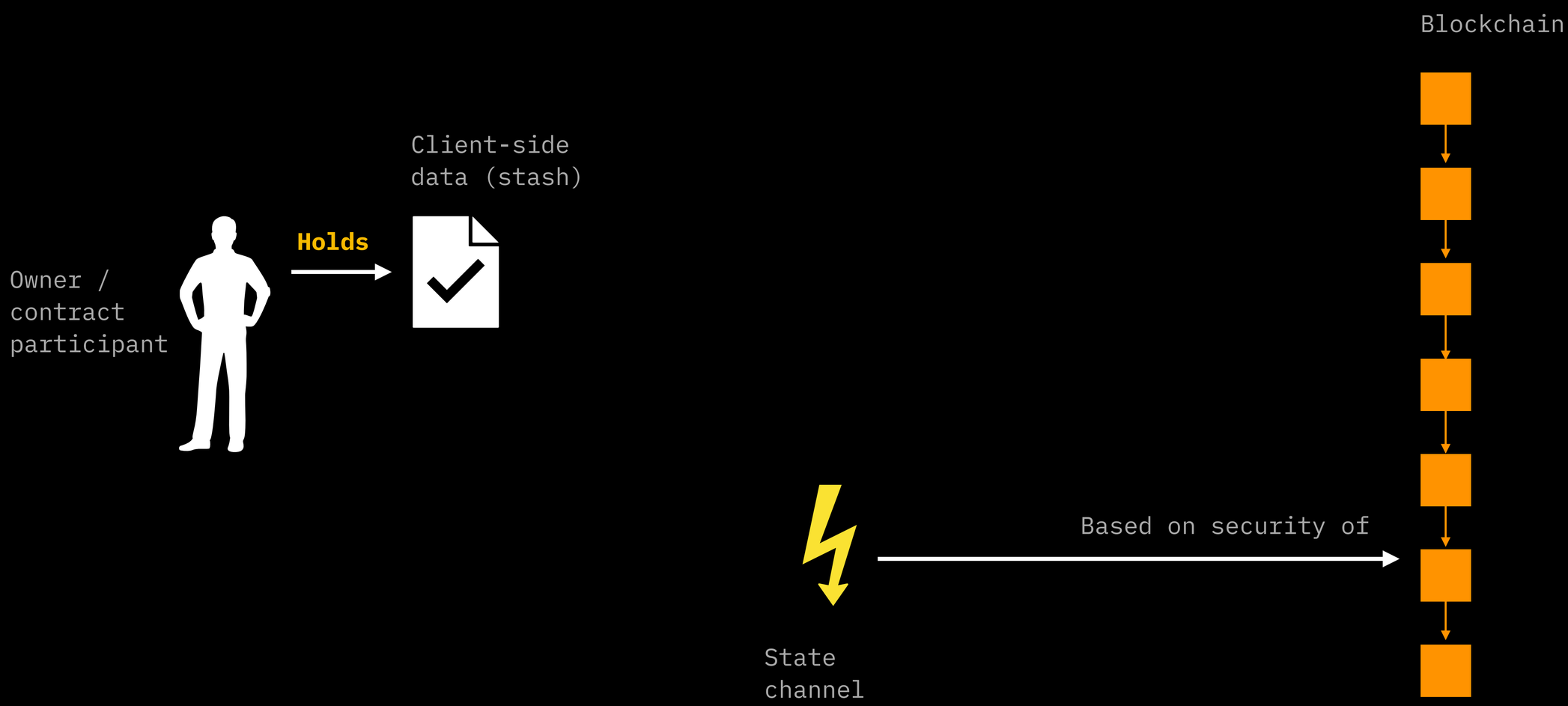
similarities and differences: neutral negative positive

- Works with both blockchain and any other state channels
- Asynchronous (network fault tolerant)
- May require storage providers
- Uses client-side data (any form fo complex state)
- Huge size of client-side-data
- No watchtowers required (*when not on top of state channels)
- Performs state validation additional to blockchain-based mining & transaction validation

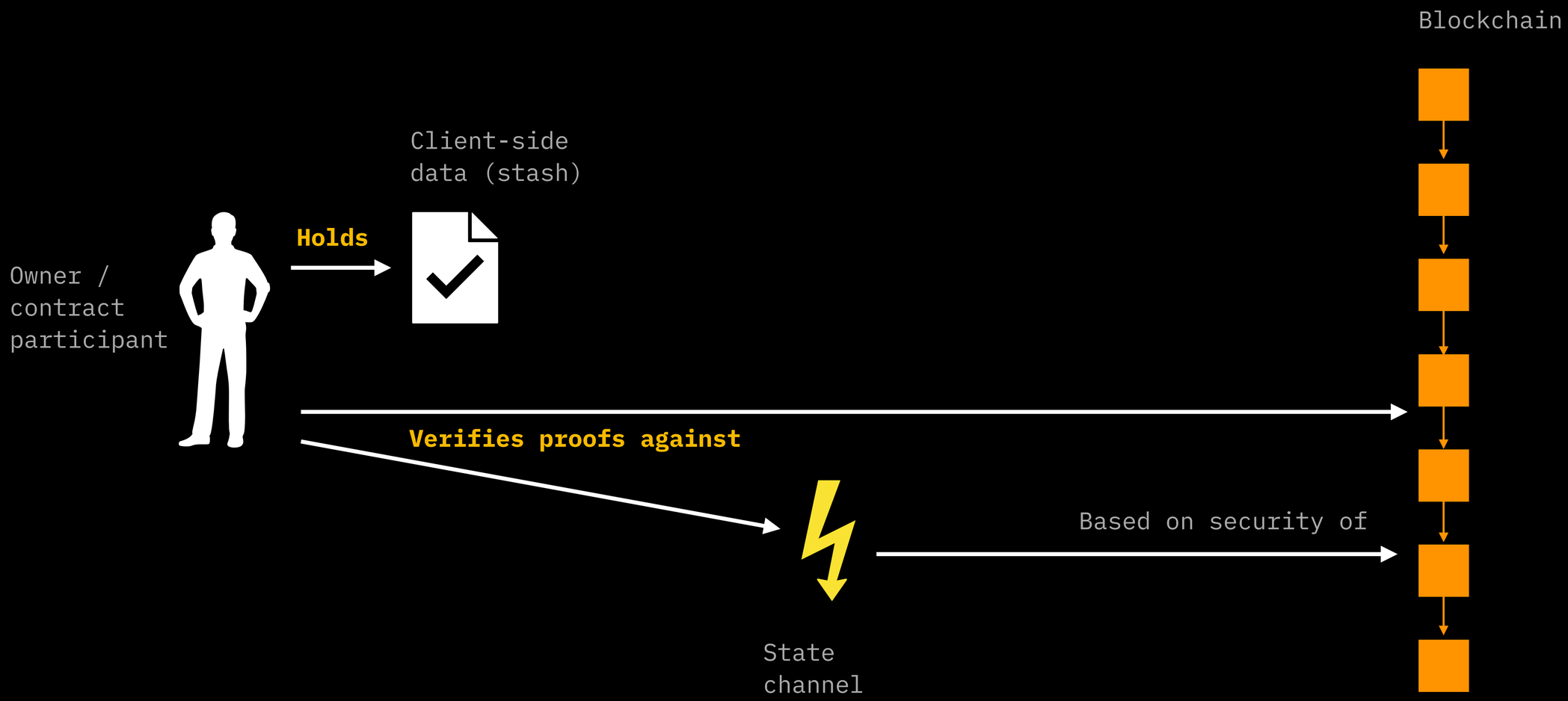
RGB: smart contracts using client-side validation



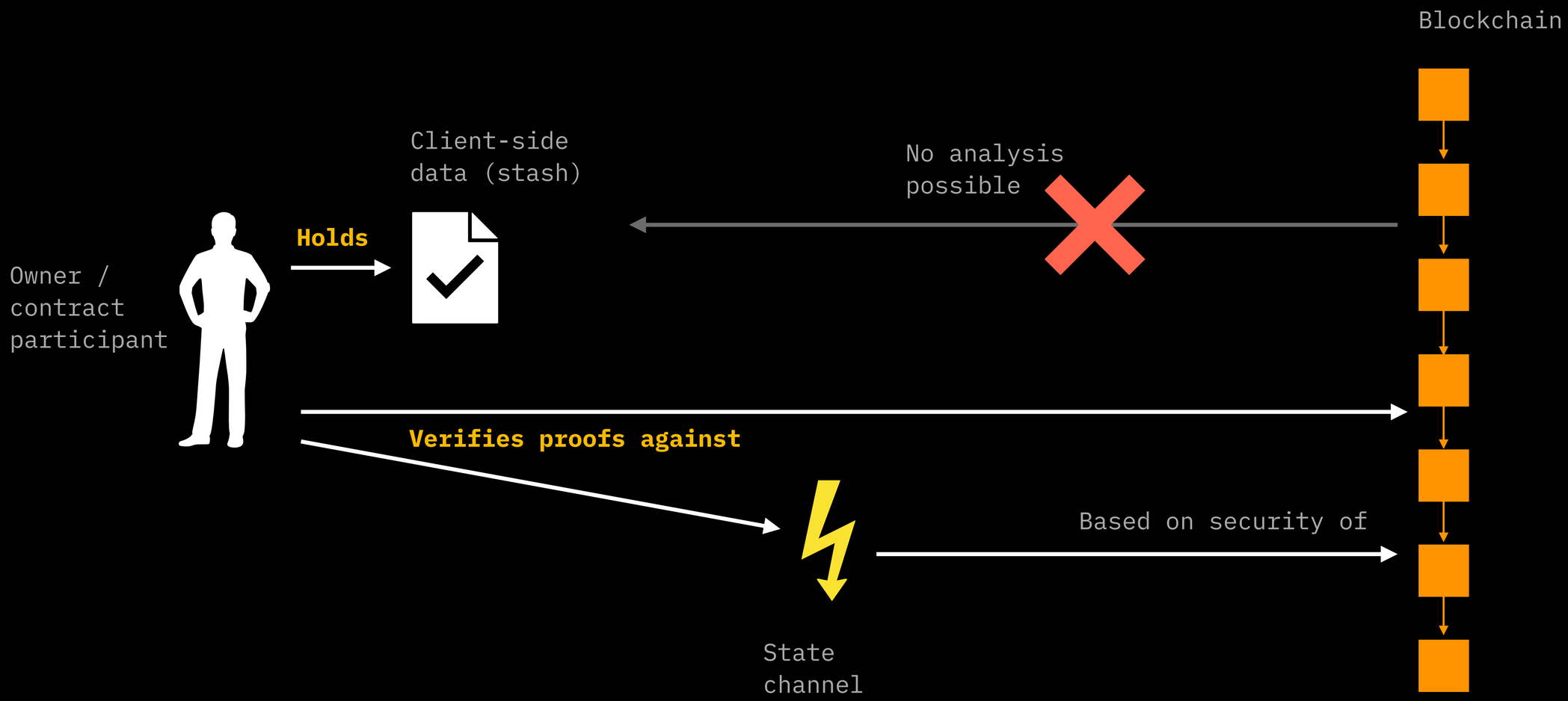
RGB: smart contracts using client-side validation



RGB: smart contracts using client-side validation



RGB: smart contracts using client-side validation



Why client-side-validation?

- More confidentiality *(than in blockchain)*
- More scalability *(than in blockchain)*
- More programmability *(than in both blockchain and state channels)*
- Richer state *(than in both blockchain and state channels)*

Client-side-validated systems

Timestamp based

- OpenTimeStamps

Single-use-seal based

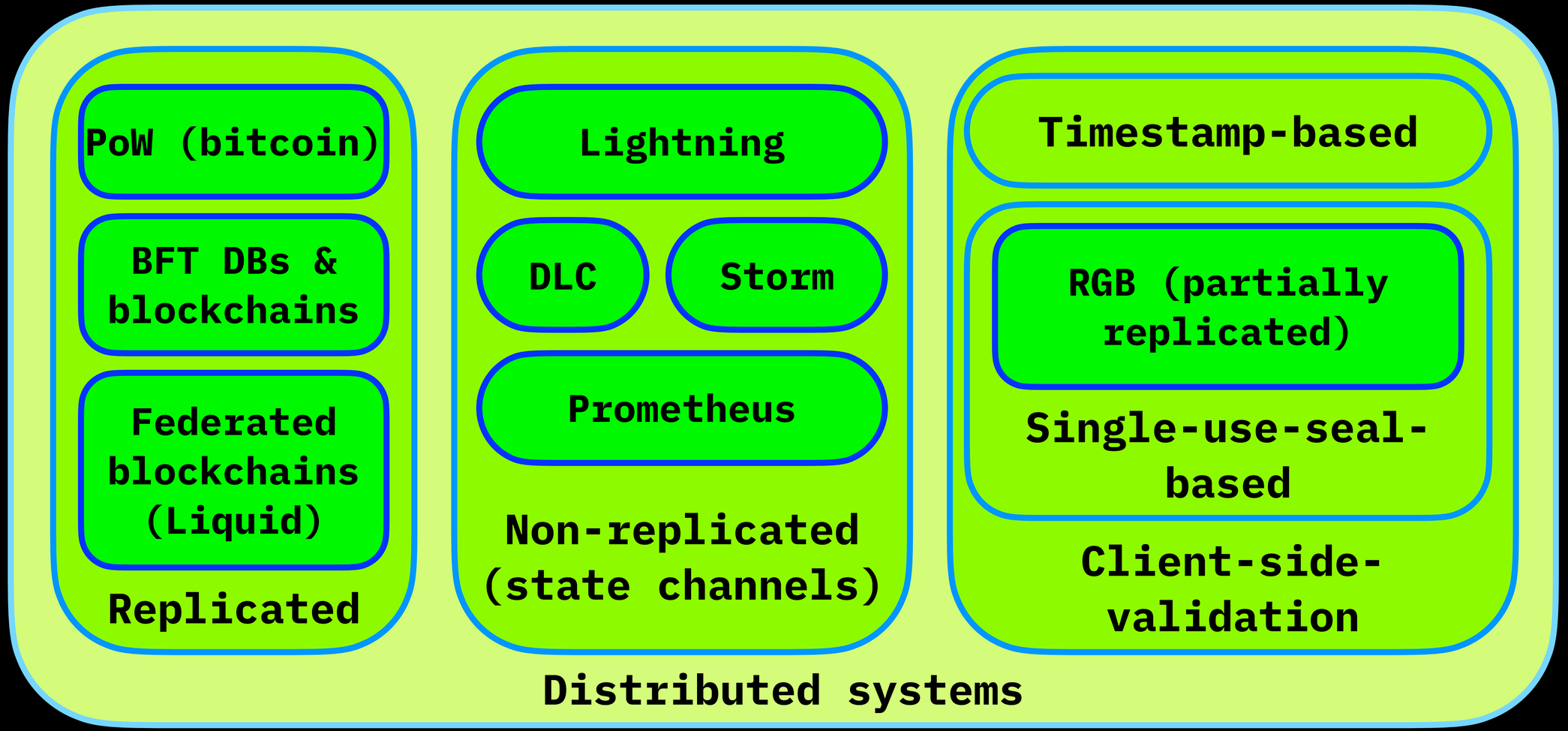
- RGB (smart contracts)
- BIP32/43-based standard for Schnorr signatures & decentralized identity
  https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2021-February/018381.html
- …more to come?

RGB client-side-validation is similar to timestamping, but more complex: it allows verification of the unique histories (via single-use-seals), not just a single state

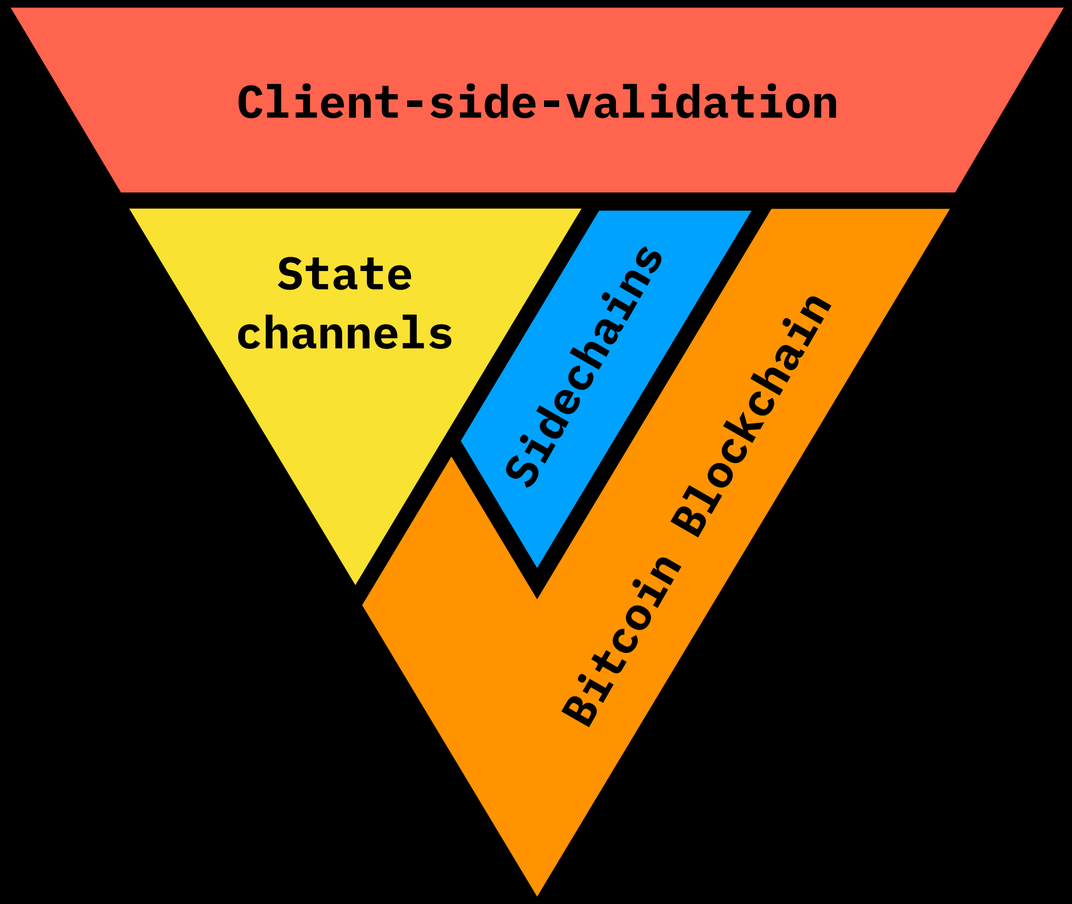
Single-use-seals

- Presentation: https://github.com/LNP-BP/presentations/blob/master/Presentation%20slides/Single-use-seals.pdf
- Video recording: https://www.youtube.com/watch?v=gGPLYfWOb_8

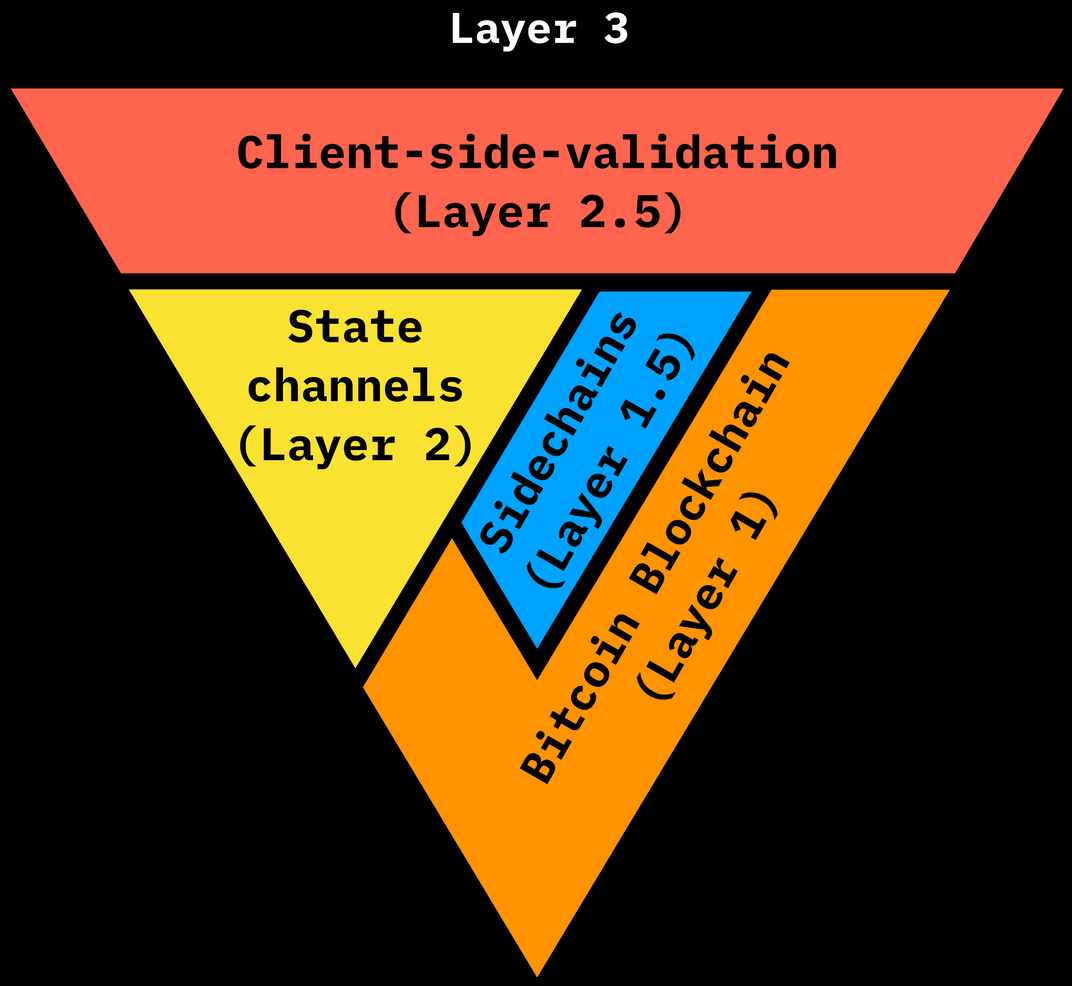
Putting it all together



Putting it all together: LNP/BP Stack



What is Layer 2 and Layer 3?



Composability: taking the best of all worlds

- You can put one state channels on top of others (DLCs-over-lightning)

  * *this is hard and not always possible*

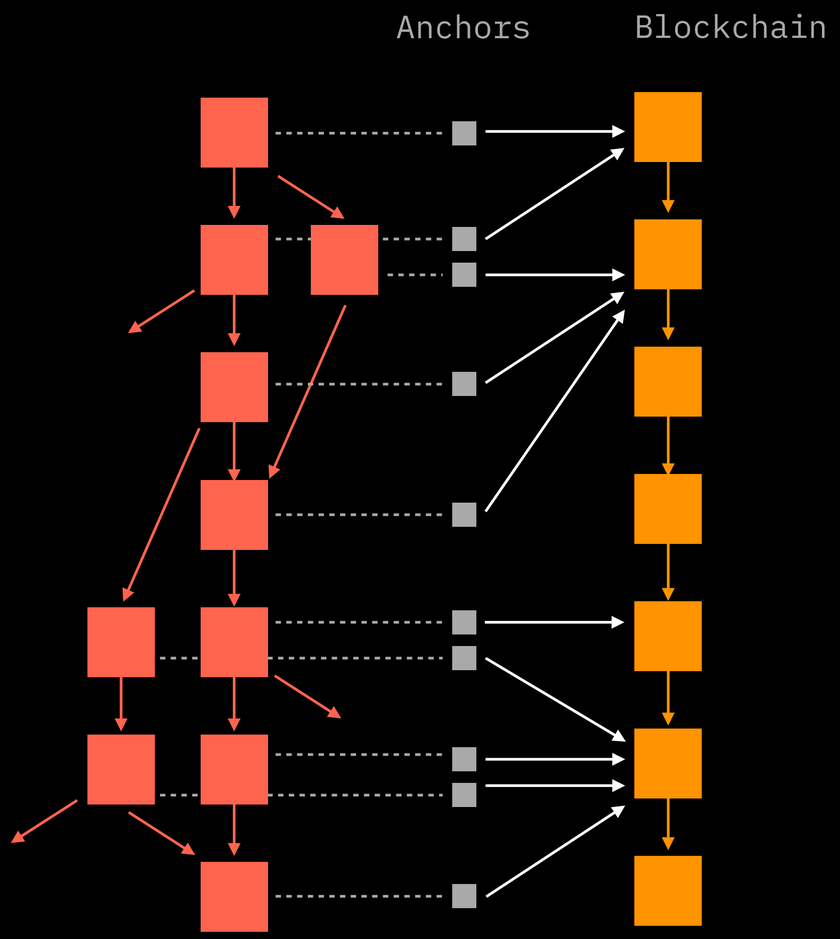
- But you can always put client-side-validation on top of state channels!

  * *may require customization of state channel protocols*

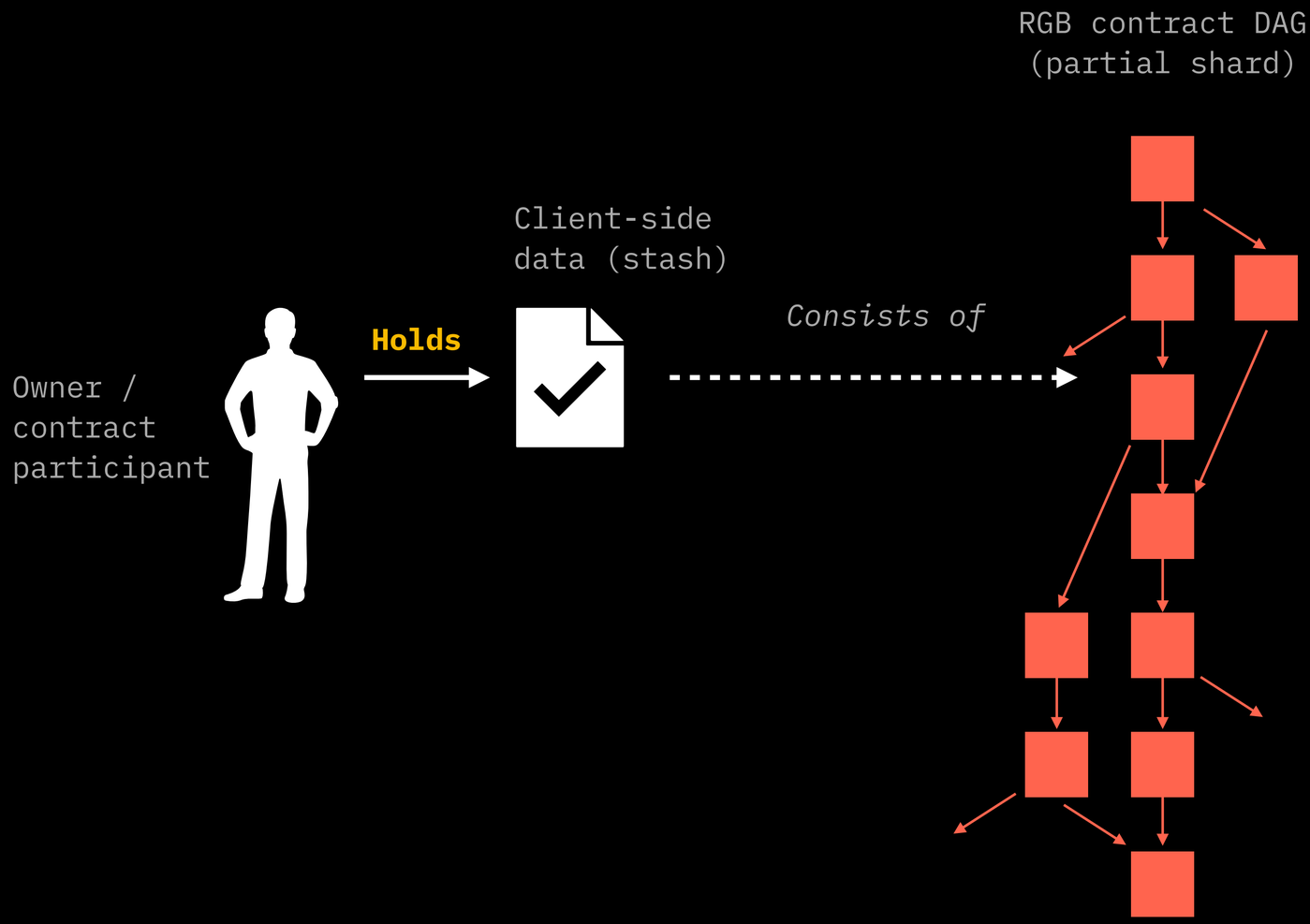
From client-side-validation to RGB

Smart contract state & history: DAG

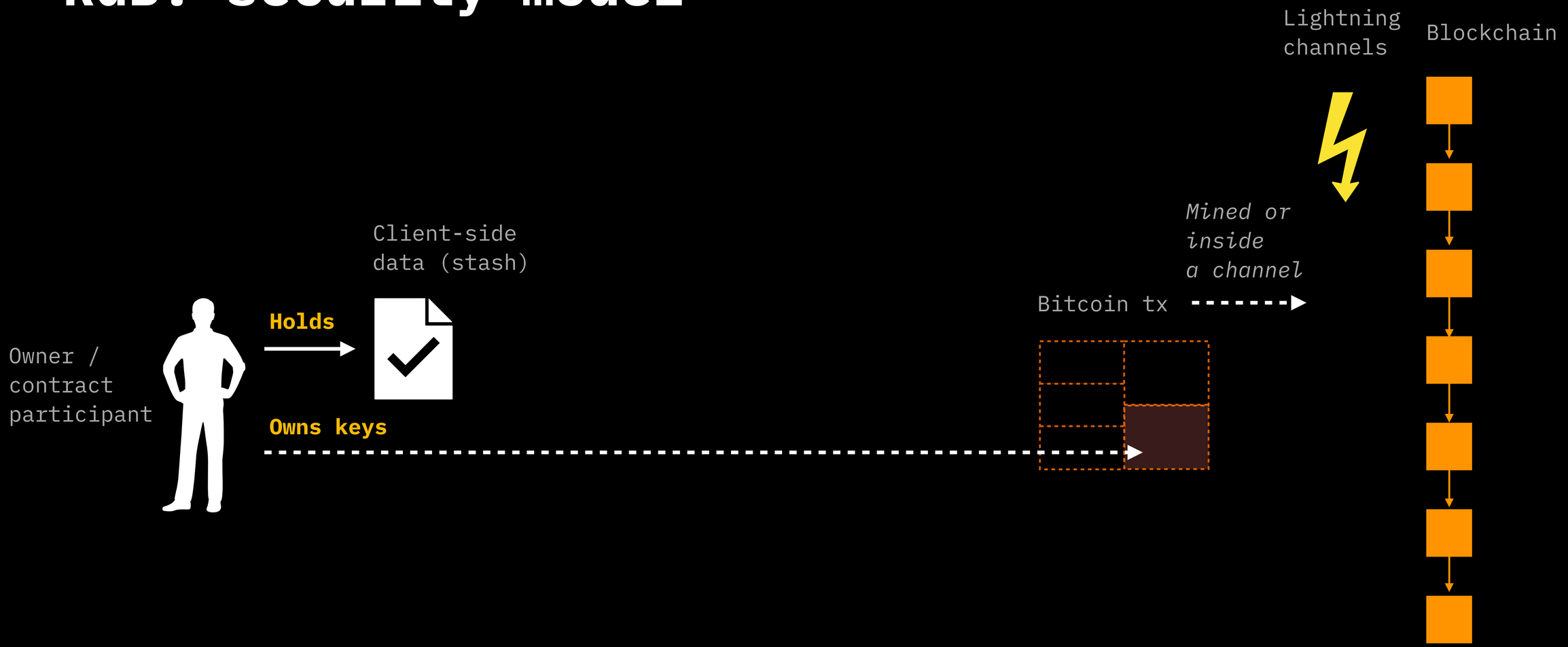
- Each RGB contract is an isolated DAG shard
- Only tips of the DAG contains active state; the rest is history
- A single view on contract by a node is “partial shard” covering history from the known state up to the state genesis in all branches
- Anchors may link state transitions from many shards (but always one-per-shard) to a single blockchain



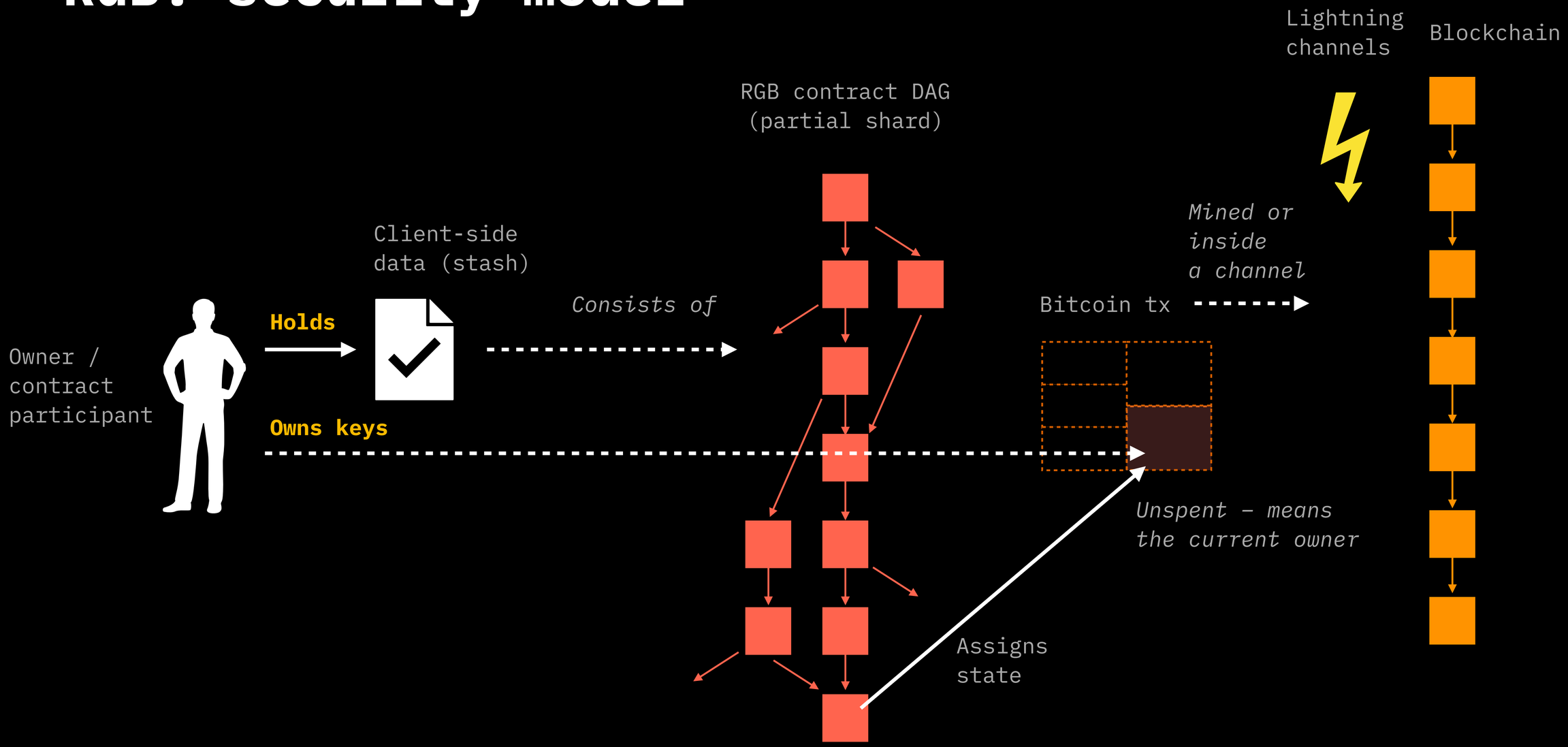
RGB: security model



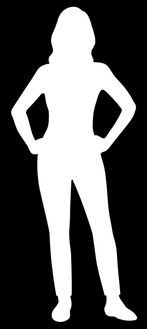
RGB: security model



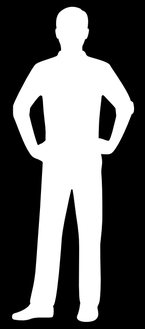
RGB: security model



RGB: how state transition works

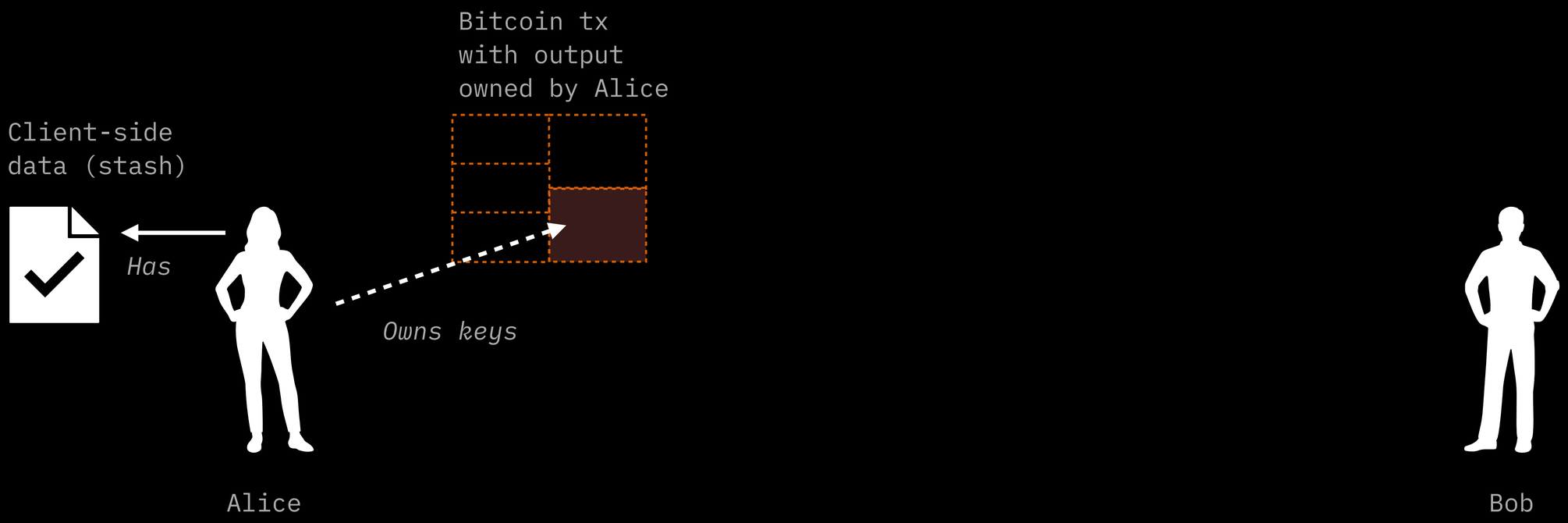


Alice

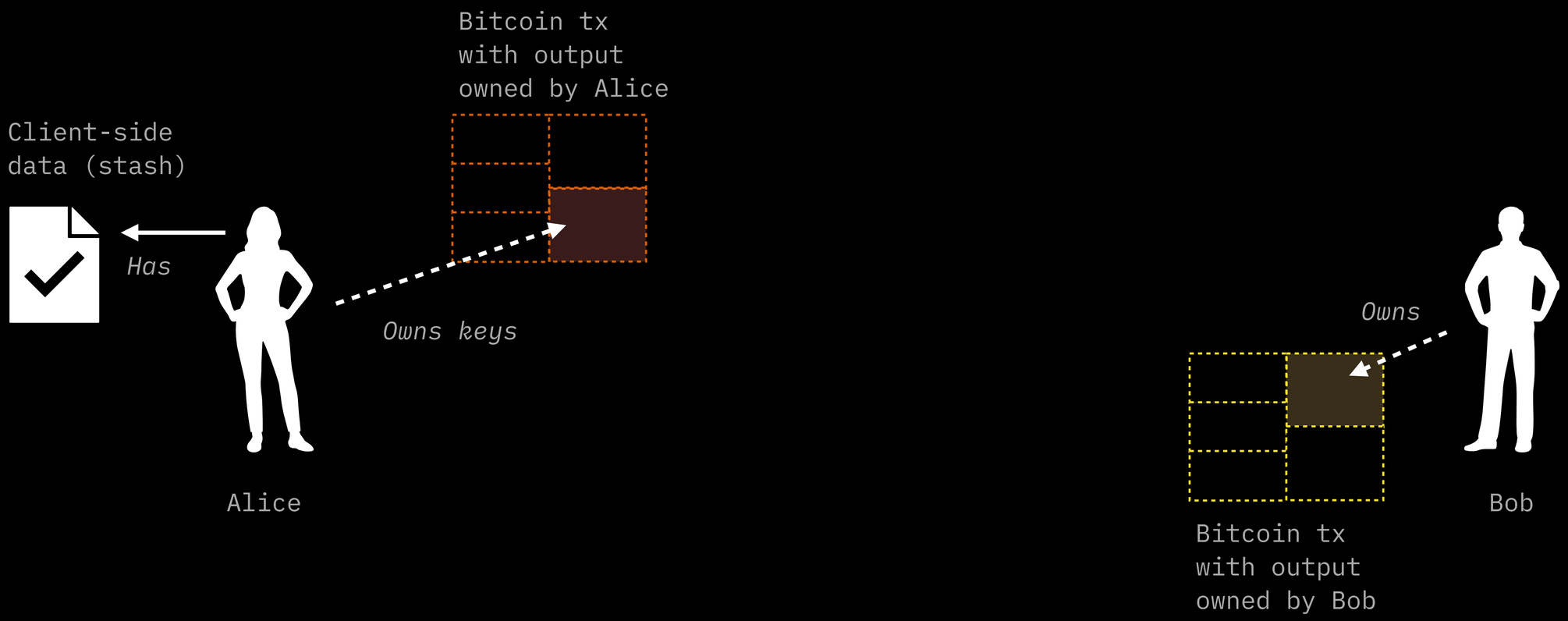


Bob

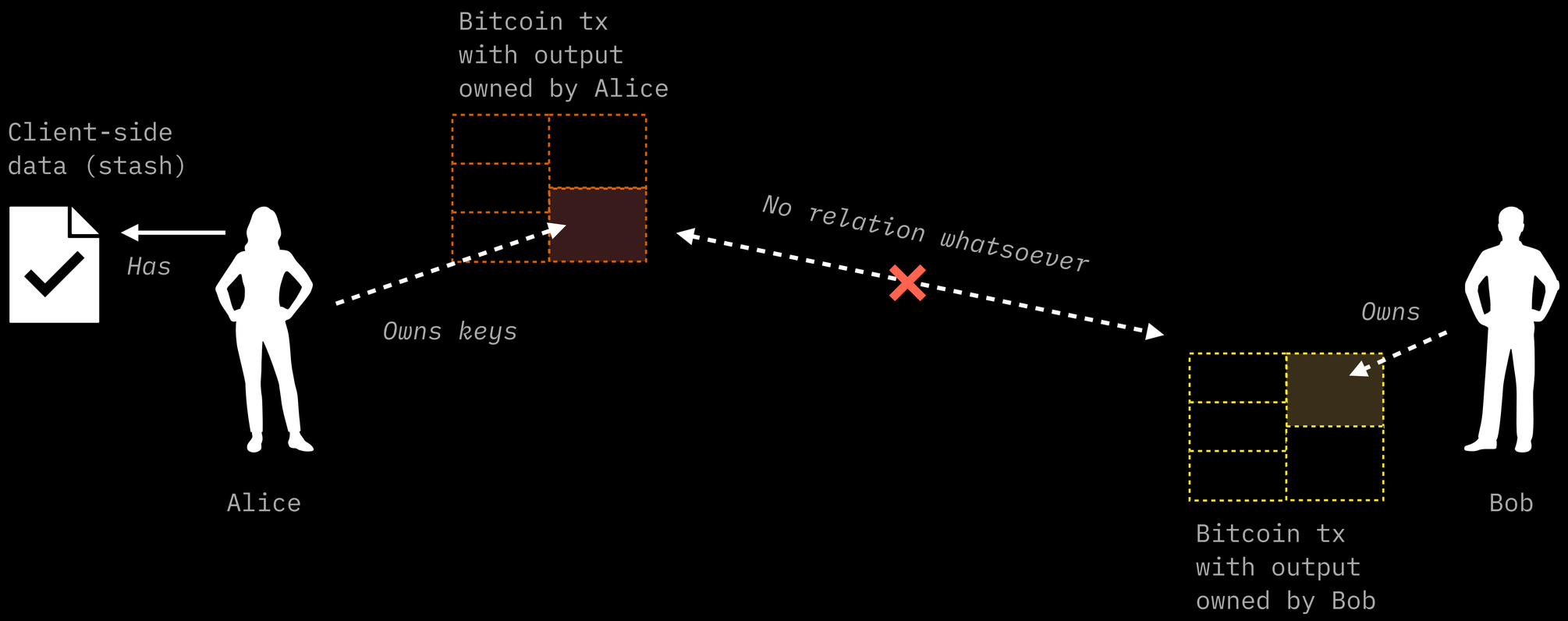
RGB: how state transition works



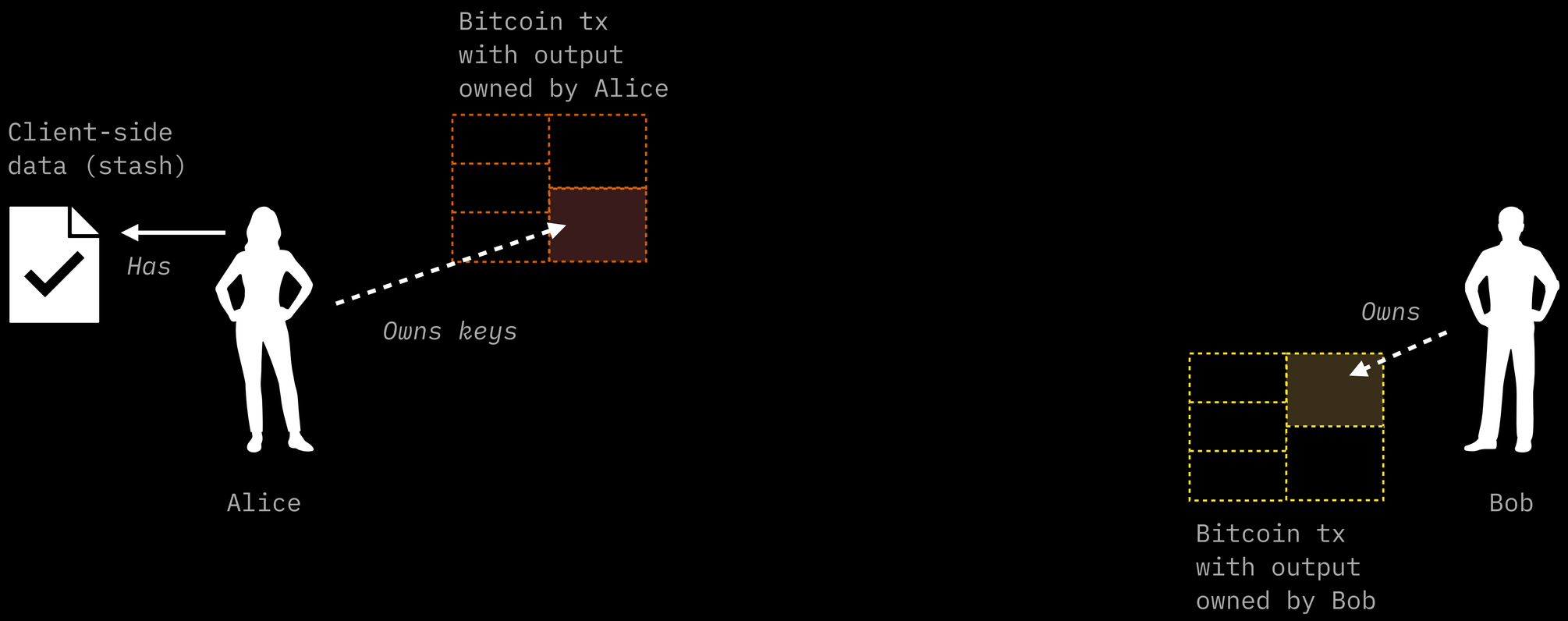
RGB: how state transition works



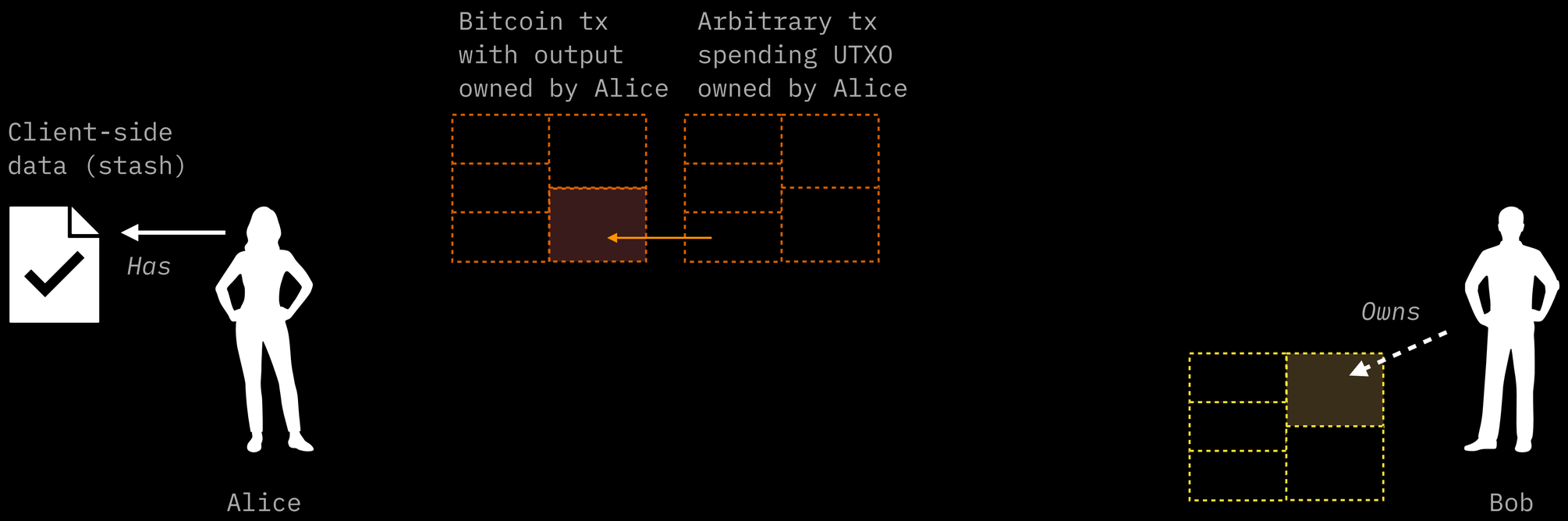
RGB: how state transition works



RGB: how state transition works

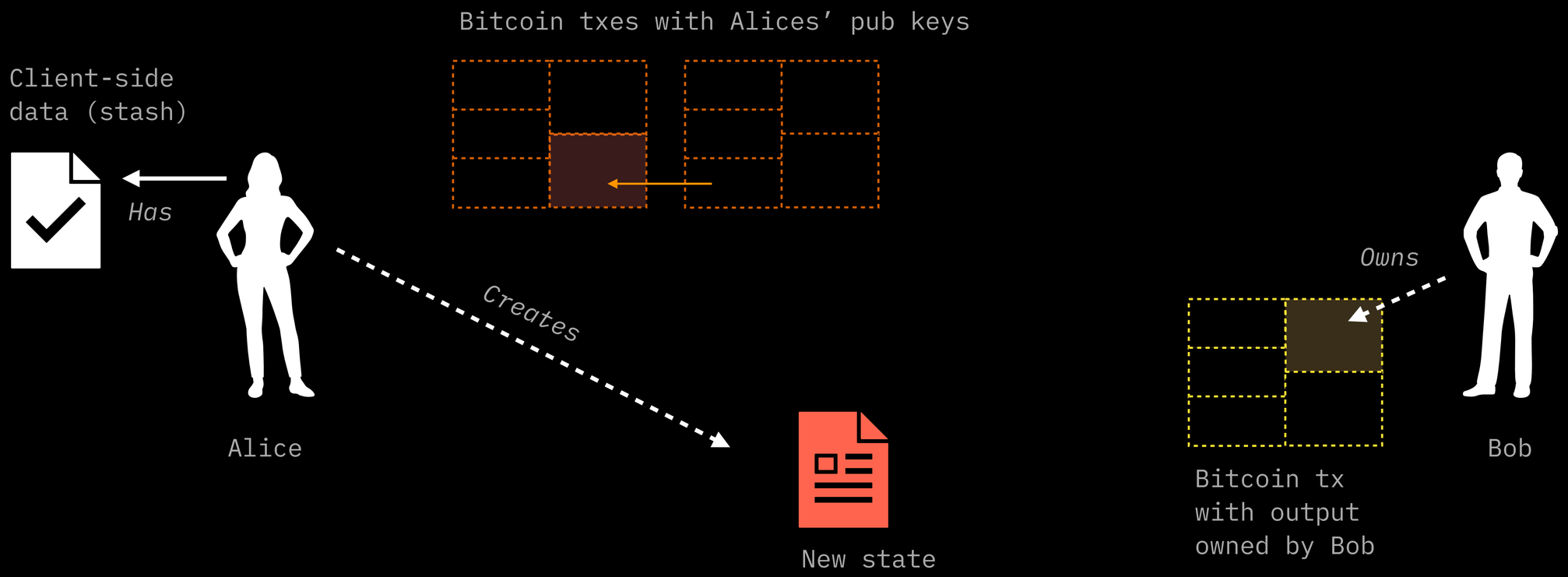


RGB: how state transition works

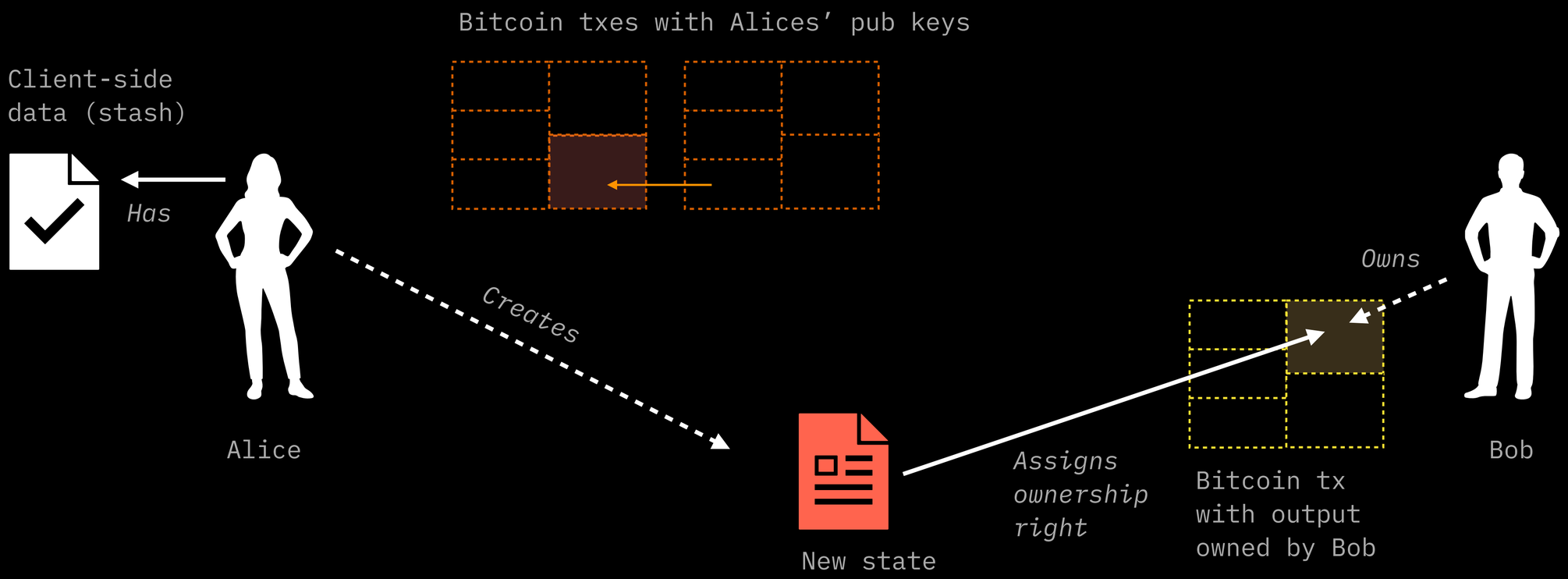


RGB: how state transition works

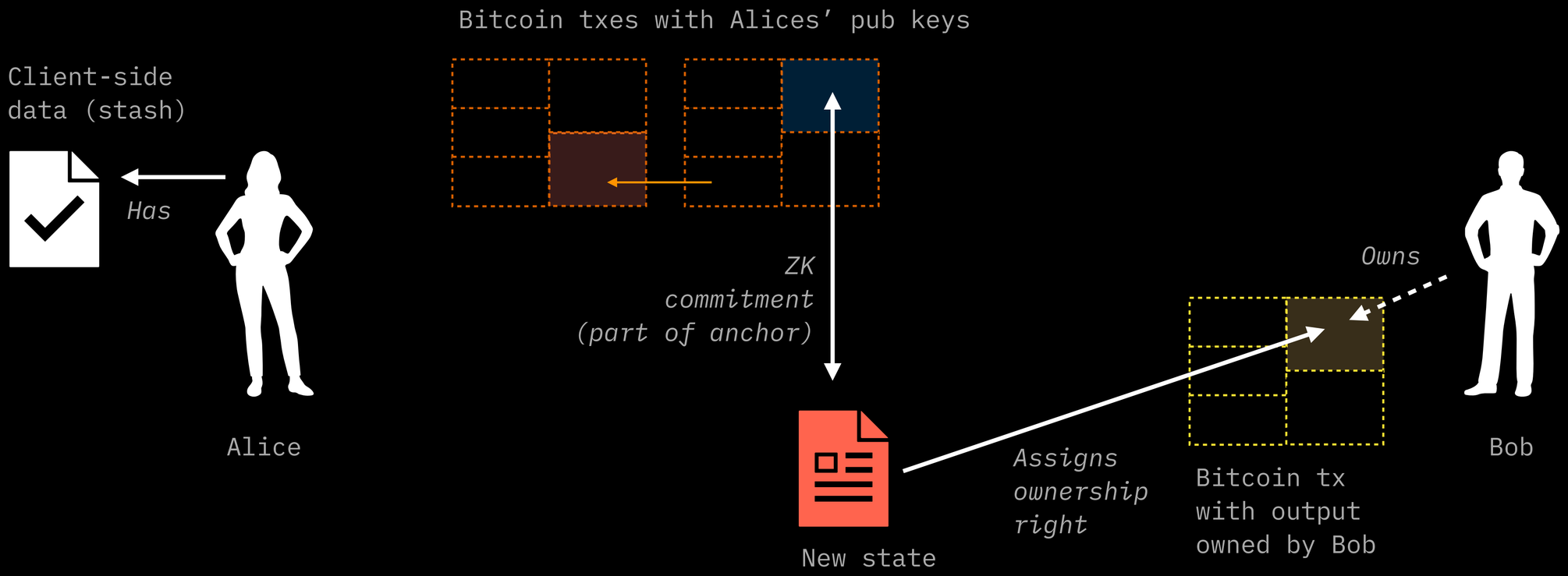
RGB: how state transition works



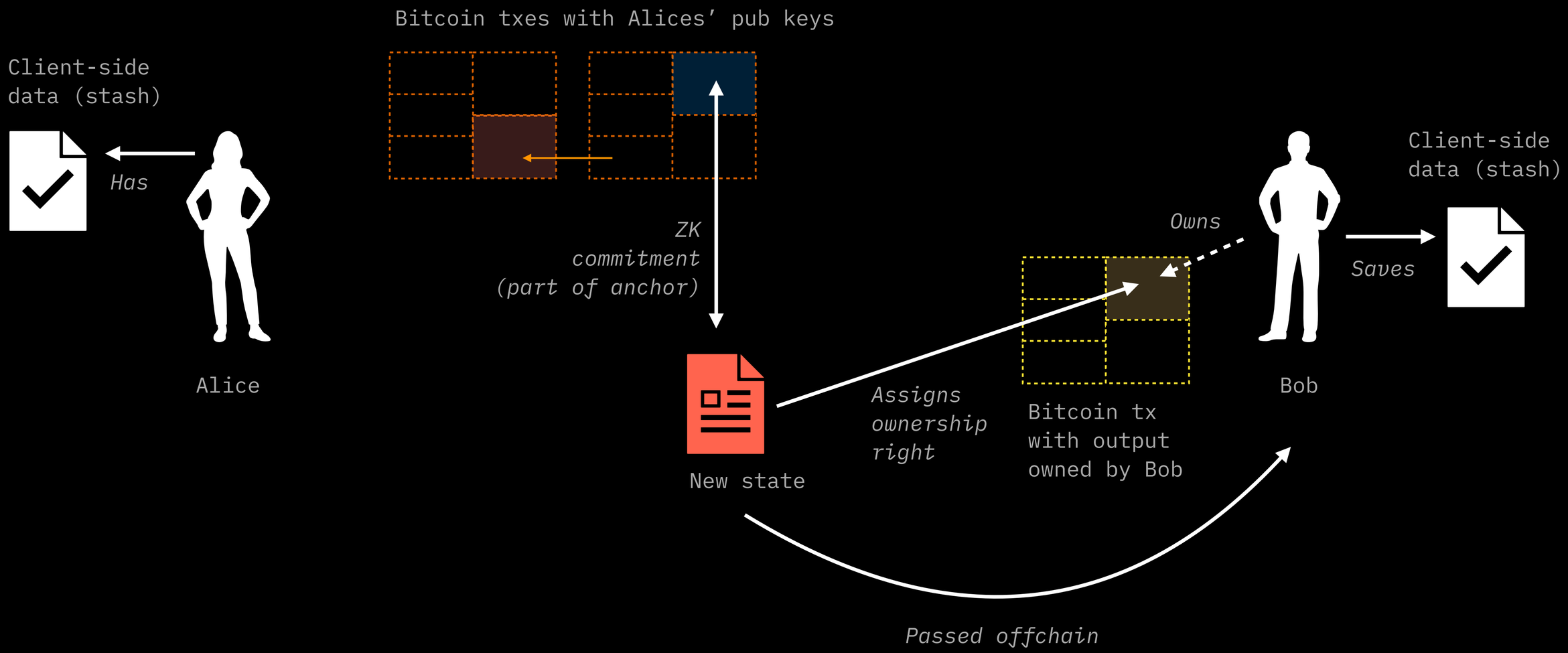
RGB: how state transition works



RGB: how state transition works



RGB: how state transition works



RGB: how state transition works

RGB: how state transition works

Client-side validation with RGB

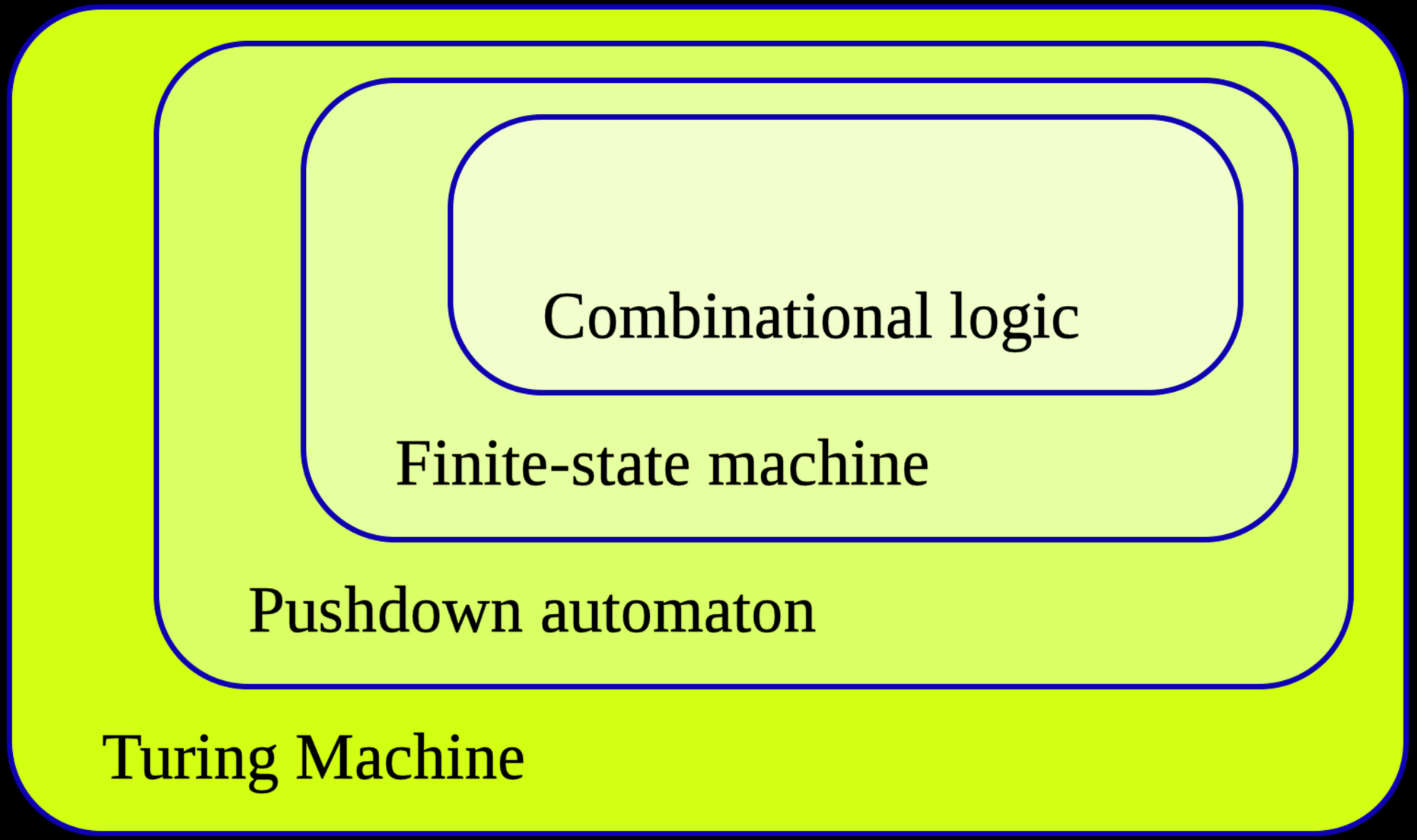
- Ownership & “double-spent” prevention: re-using **Bitcoin script** combined with **single-use-seal validation**
- Consistency & completeness: **Schema**
- Changes in state: **AluVM**
  you can learn more about AluVM and its use in RGB at
  - *https://github.com/LNP-BP/presentations/blob/master/Presentation%20slides/Single-use-seals.pdf*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

Client-side validation of RGB contract

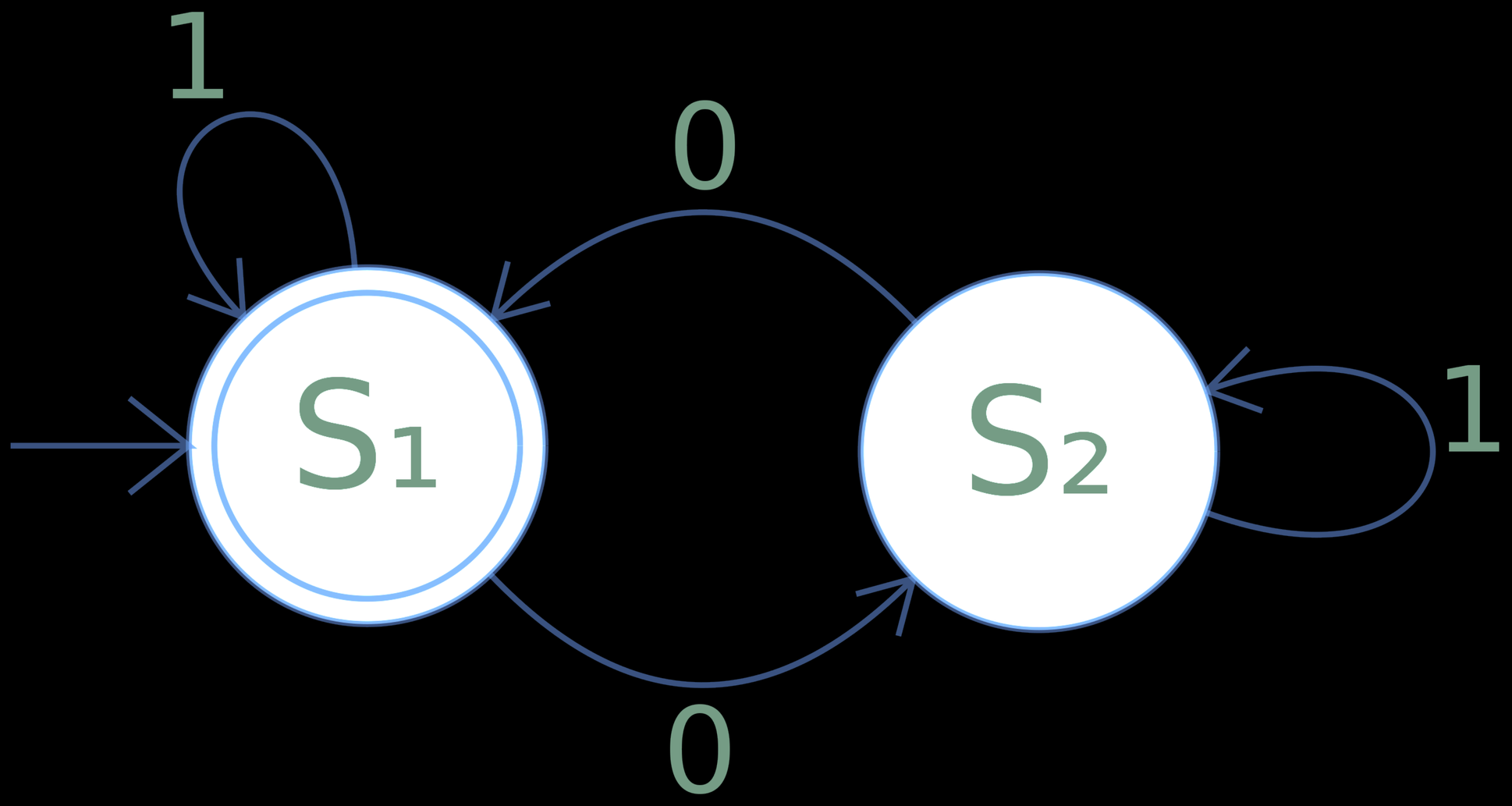
- **Validation of internal consistency**
  - Contract can be deserialized from a given byte representation
  - Contract container contains all required nodes and information
- **State verification**
  - Each contract node matches **schema** rules
  - Each change in state is valid under provided **AluVM** scripts
- **Anchors verification:** prevention of double-spends/double-transitions
  - Verification of all closed **single-use-seals** against blockchain (implies **bitcoin script** verification of all spent outputs by Bitcoin Core)
  - Verification of all required bitcoin transactions are mined in the blockchain or a part of the valid channel state

New paradigm (“client-side-validation”) required certain advancements in computing

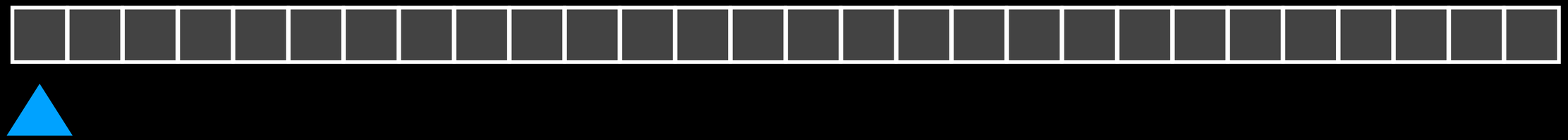
Classes of automata

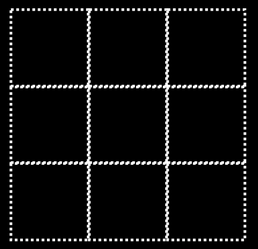


Finite state machines (include register-based VMs)



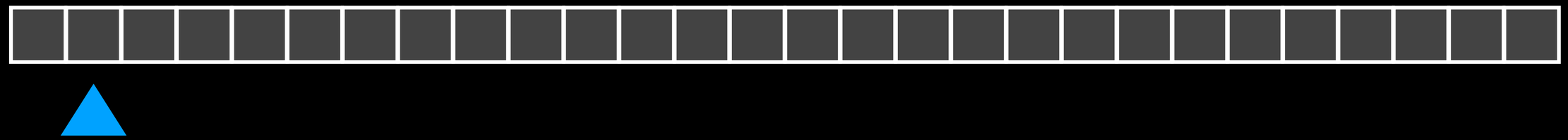
Register-based machines (x86_64, ARM, AluVM)

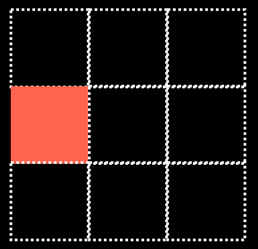




Registers

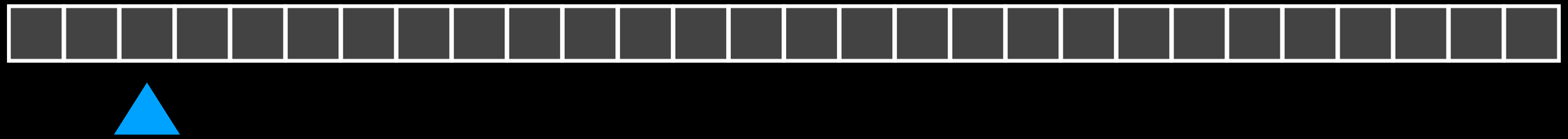
Register-based machines (x86_64, ARM, AluVM)

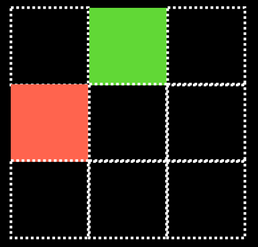




Registers

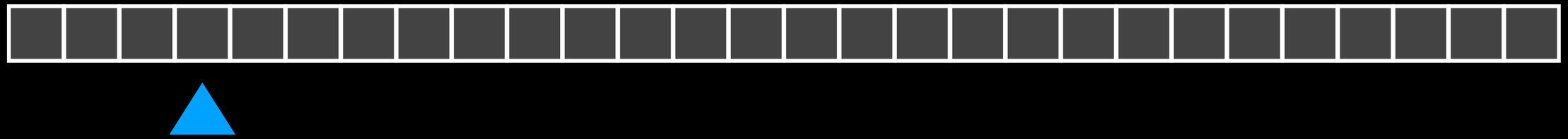
Register-based machines (x86_64, ARM, AluVM)

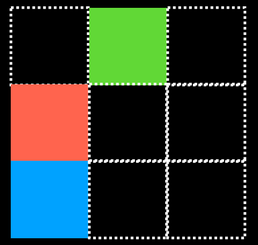




Registers

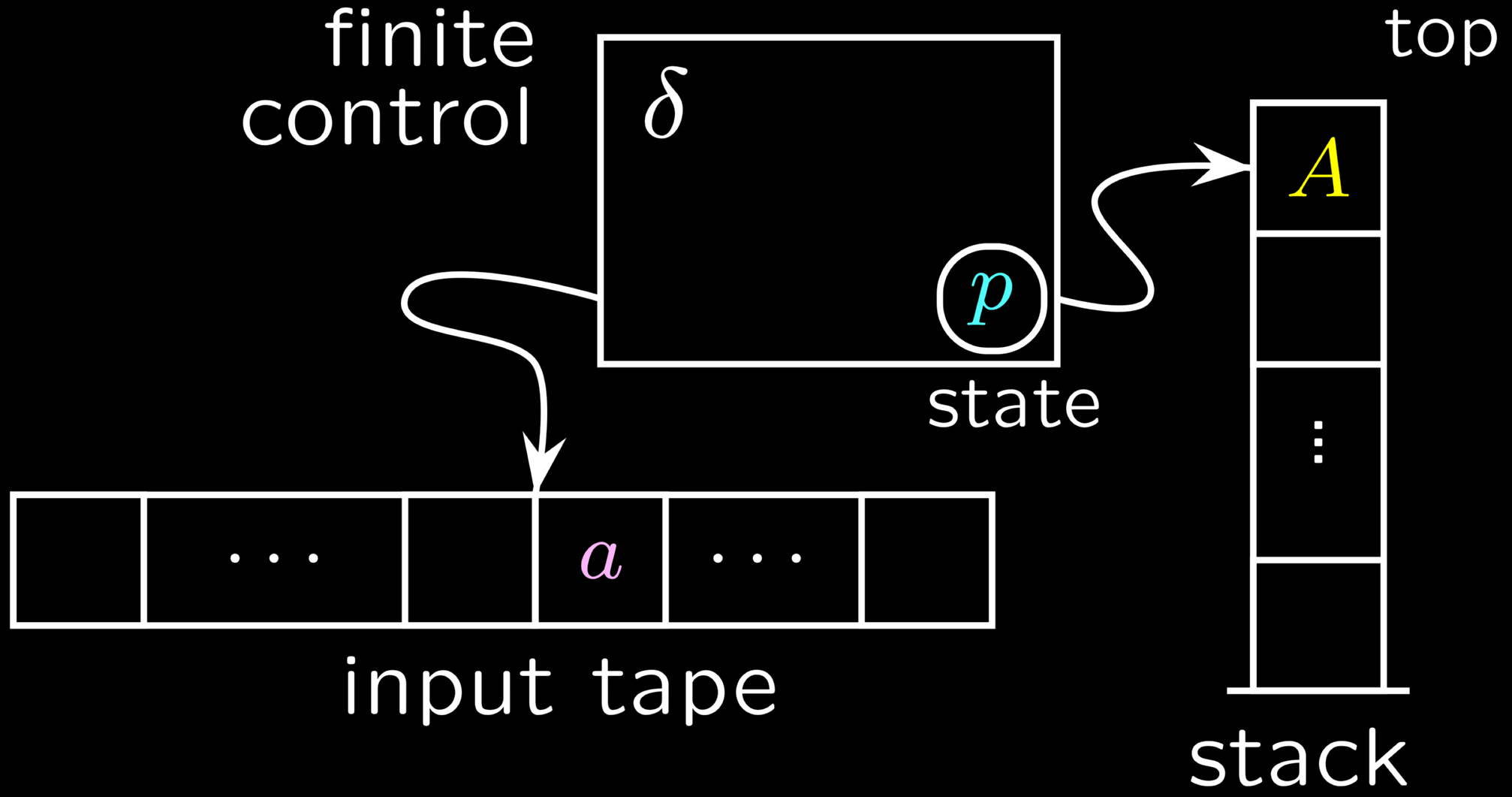
Register-based machines (x86_64, ARM, AluVM)



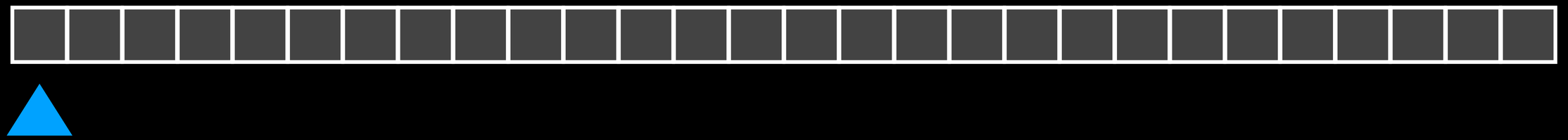


Registers

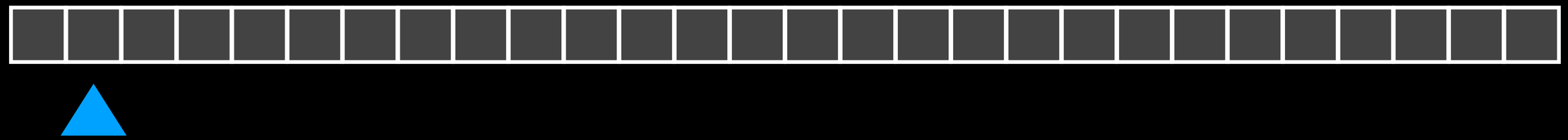
Pushdown automata: stack machines / VMs



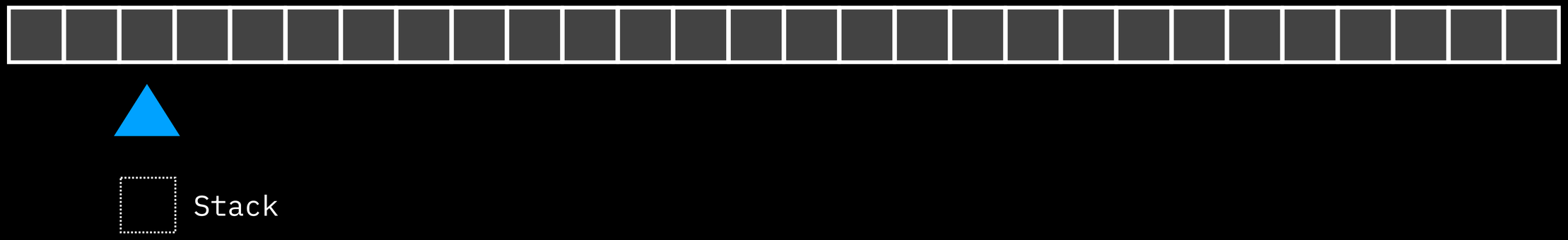
Stack-based VMs (Bitcoin Script, EVM, WASM)



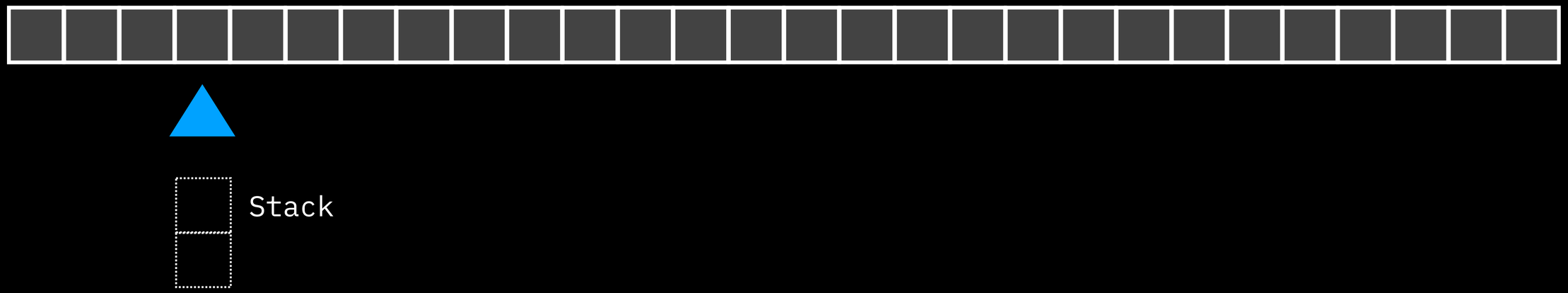
Stack-based VMs (Bitcoin Script, EVM, WASM)



Stack-based VMs (Bitcoin Script, EVM, WASM)



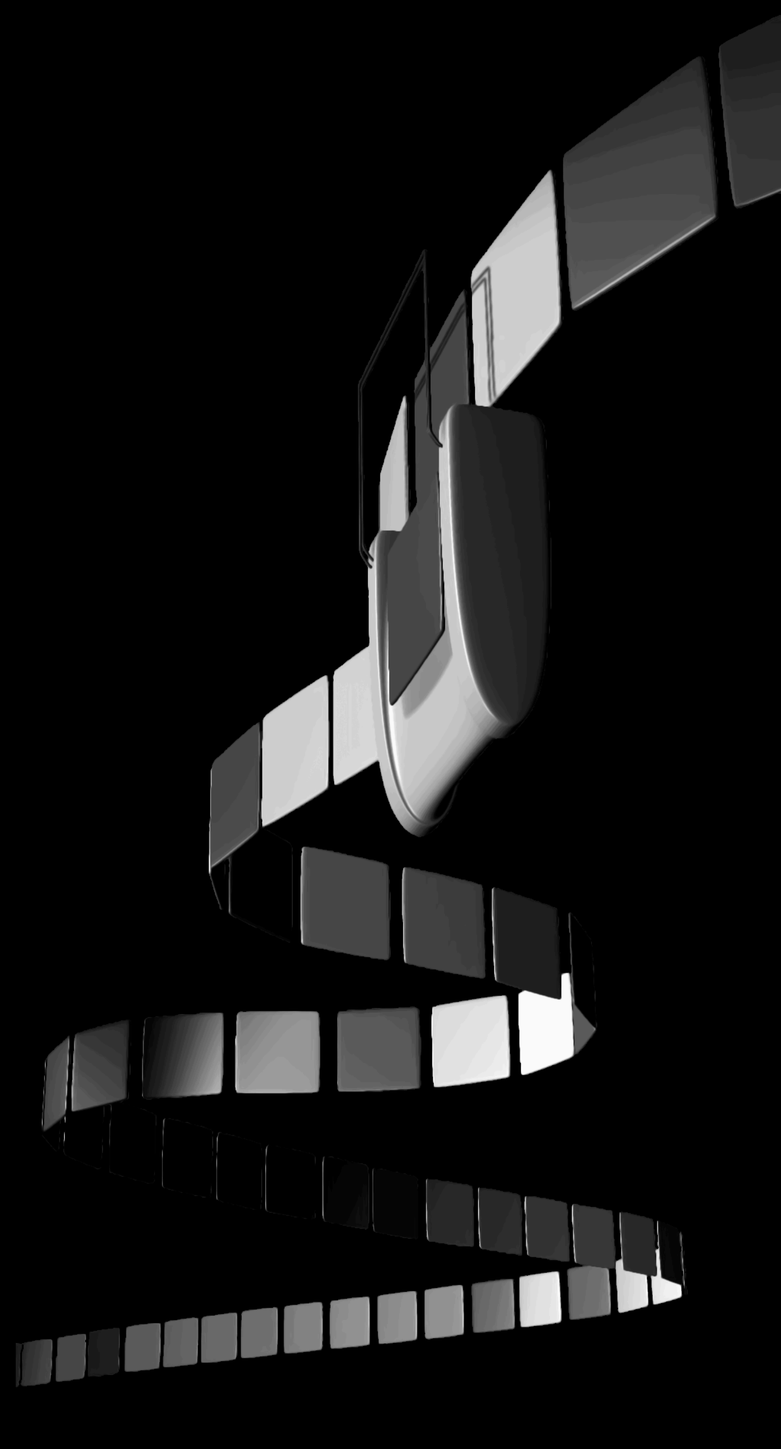
Stack-based VMs (Bitcoin Script, EVM, WASM)



Stack-based VMs (Bitcoin Script, EVM, WASM)

Turing machine

- Infinite “tape” with separate instructions from certain finite set
- “Head” reading single instruction at a time
- Has its own state modifiable by instructions
- Can write to the tape (basing on instructions)
- Can move throughout the tape (basing on instructions)



Turing machine design is inefficient in practical terms,
but being one of the first models of computing it became
abstract reference model of universal automation

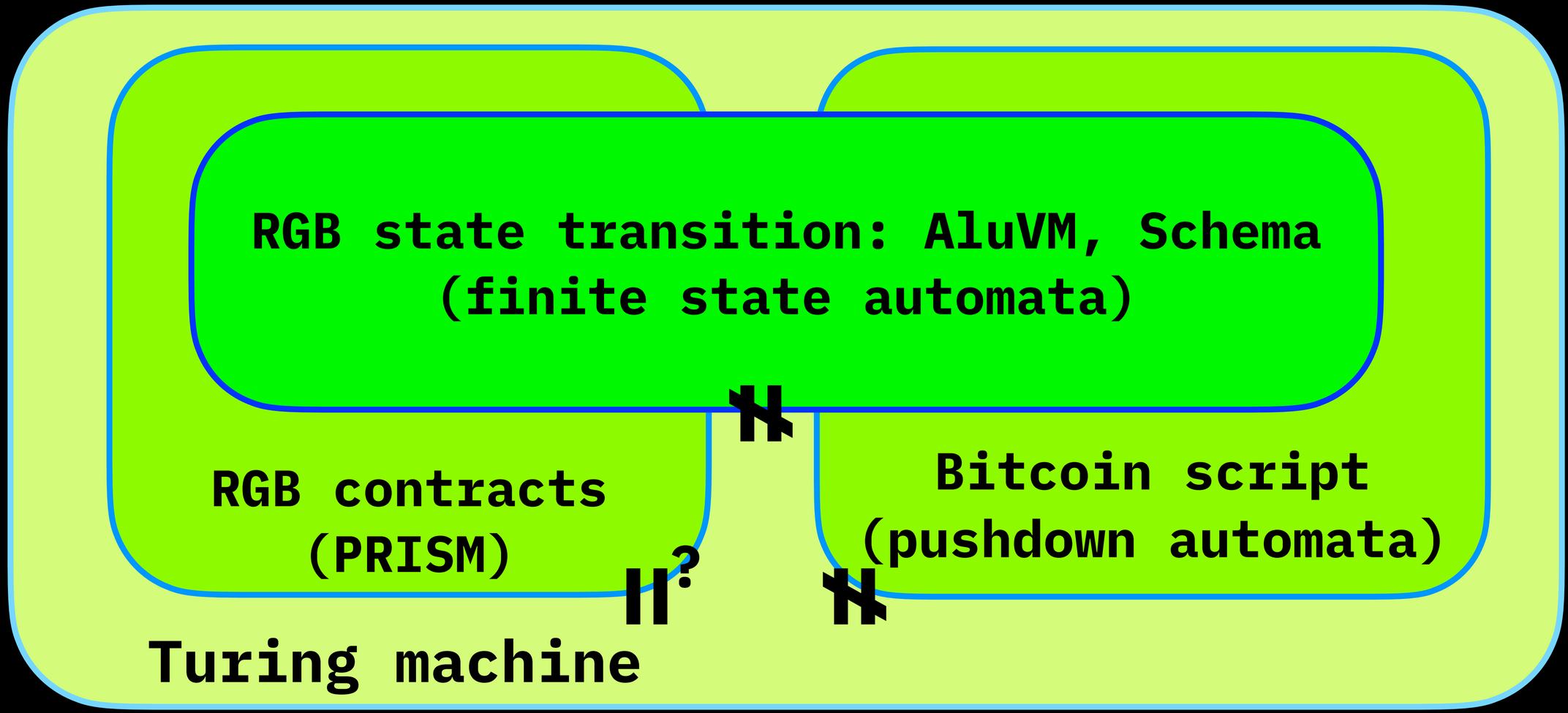
Any machine that may be transformed into a Turing machine
is called **universal** or **Turing equivalent**

RGB smart contract computation model (PRISM)

- **Distributed** (state is distributed across multiple actors)
  - *Fully asynchronous*
  - Network-*fault tolerant*
  - achieves *consensus* by utilizing special form of “finality proofs” (PoW-UTXO single-use-seal consensus)
- **Partially-replicated** – **PR**ism
  (each participant/actor has a partial view of smart contract state;
  the whole state is a DAG)
- **Indeterministic** (affected by participants’ decisions and PoW randomness) – pr**I**sm
- **State machine** (equivalent? to universal Turing machine) – pri**SM**

RGB smart contract computation model (PRISM)

- Smart contract operates within constraints immutably defined by schema and previous contract history
- Each state transition is
  - partial (may affect only part of the smart contract state)
  - equivalent to a finite state machine operation
- Whole contract may have countable, but infinite number of states, thus RGB contract can’t be represented by a finite state machine and is possibly equivalent to an universal Turing machine


RGB state transition: AluVM, Schema
(finite state automata)
RGB contracts
(PRISM)
Bitcoin script
(pushdown automata)
Turing machine

RGB smart contracts are

- Computationally-bounded (“finite state machine”) at each step (state transition)
- Should be equivalent to a universal Turing machine (“Turing-complete”) as a whole

Details of RGB contracts

Components of RGB contracts

- **Schema**: defines constraints for state machine
- **Nodes**: DAG of performed state transitions
- **Anchors**: proofs state transition validity, consisting of:
  - single-use-seal external witnesses (blockchain link)
  - commitments to state transition under multiple contracts (RGB link)
- **Bitcoin blockchain** (or **Lightning channel** backed by blockchain):
  - controls who can perform state transition
  - enables consensus over distributed state machine

Schema

- What kinds of ownership rights exist?
- What kinds of publicly-executable rights exists?
- What kinds of contract metadata can be defined by the contract?
- What types of state data may be assigned to owned rights?
- How this state data allowed to change with state transitions?
- What sequences of state transitions/ownership right operations are allowed?

Schema does that in a functional way

- Defining **types** for (declarative programming, run by RGB Core):
  - metadata
  - state
  - ownership rights
  - public rights
  - possible combinations of the above within state transition
- Providing state validation **functions** (run by **AluVM**):
  validate :: state -> bool
  for each of state types

Smart contract nodes

State generation and state transition events under single contract batched together by a specific finality proof (anchor)

- State generation:
  - **Genesis**
  - **State extension**
- **State transition**

Smart contract state inside contract node

- **Metadata** (not “ownable” part of state)
- **Assignments**, consisting of
  - Type of ownership rights (from schema)
  - Definition of single-use-seals owning that right
  - State data (optional)
- **Valencies**: public rights (from schema) with no state.
  Linking points for “state extensions”

Smart contract state

	Genesis	State extension	State transition
Metadata	+	+	+
Valencies	+	+	+
Assignments	+	+	+
Extinguished assignments	No	No	+
Used valencies	No	+	No

Packing of RGB contracts (RGB containers)

- **Stash**: all RGB contracts partial state owned by a user.
  May contain alternative histories
- **Consignment**: single RGB contract partial state from genesis and all off-chain data required for its validation
- **Disclosure**: fragments of data of multiple RGB contracts adding more information for a stash
- **Data container**: blobs of supplementary data (like NFT media) for a given RGB smart contract

Extinguishing assignments = updating state

- State is invalidated on a per-assignment basis
  by the fact of UTXO spending
  previously defined as a single-use-seal
- The single-use-seal must be closed over a message (anchor data)
  which contains a commitment to the newly defined state in a
  state transition