Invariant Safety for Distributed Applications

Sreeja Nair
Gustavo Petri
Marc Shapiro

Stateful Distributed Systems

We want:

$$\left.\begin{array}{l}\bullet\ \text{Scalability}\\ \bullet\ \text{Availability}\end{array}\right\}\begin{array}{c}\text{Replicated}\\ \text{State}\end{array}$$

- Programmability
  ≈ Strong Consistency

STATEFUL DISTRIBUTED SYSTEMS

WE WANT:

- Scalability
- Avai
- grammability
  ≈ Strong Consistency

} Replicated State

CAP* Theorem

* Consistency, Availability, Partition Tolerance [Gilbert&Lynch'02]

STATEFUL DISTRIBUTED SYSTEMS

WE WANT:

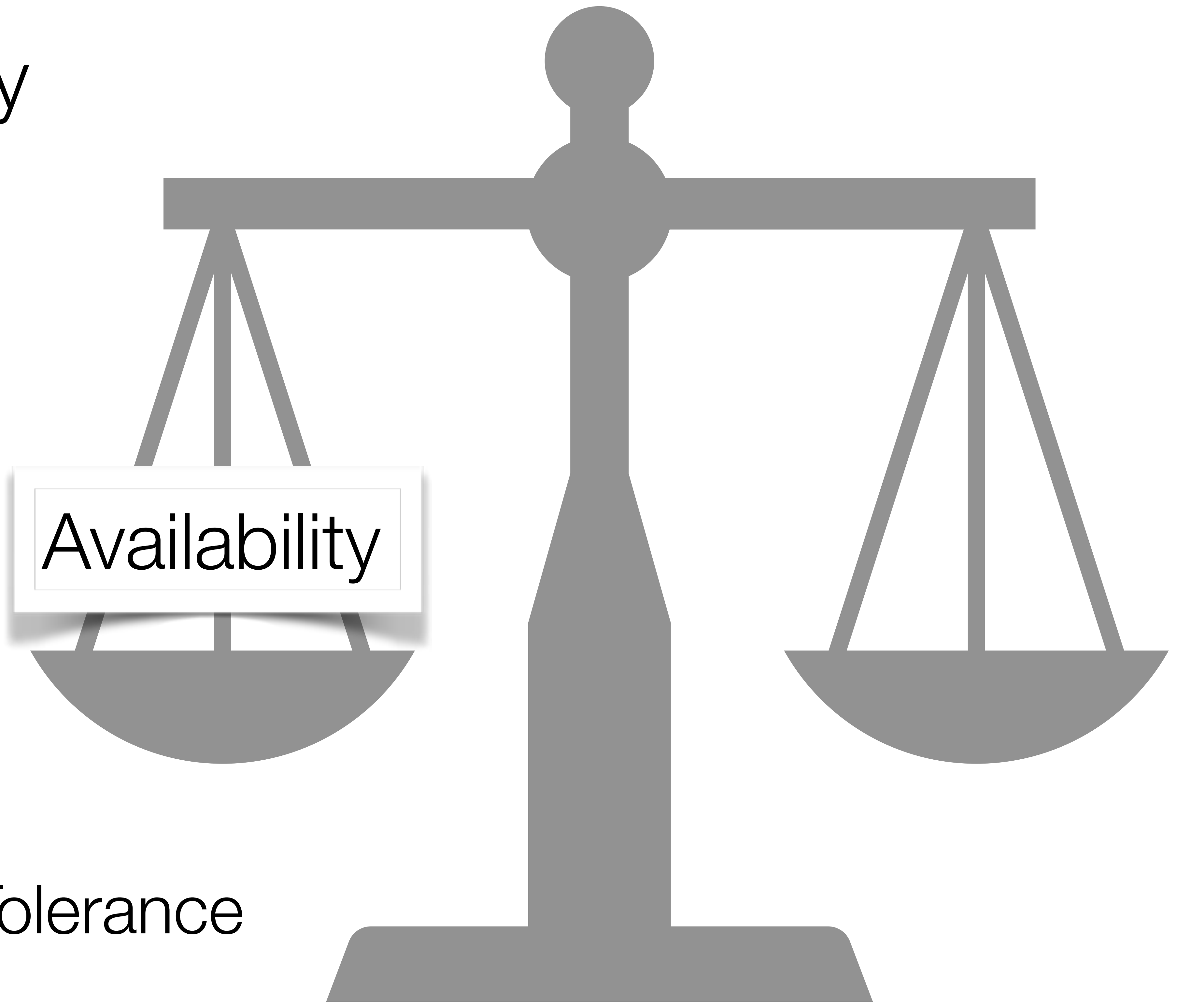
- Scalability
- Avail
- grammability
  ≈ Strong Consistency

} Replicated State

CAP* Theorem

WE GET:

- Availability



* Consistency, Availability, Partition Tolerance [Gilbert&Lynch'02]

Stateful Distributed Systems

We want:

- Scalability
- Avail
- grammability
  ≈ Strong Consistency

} Replicated State

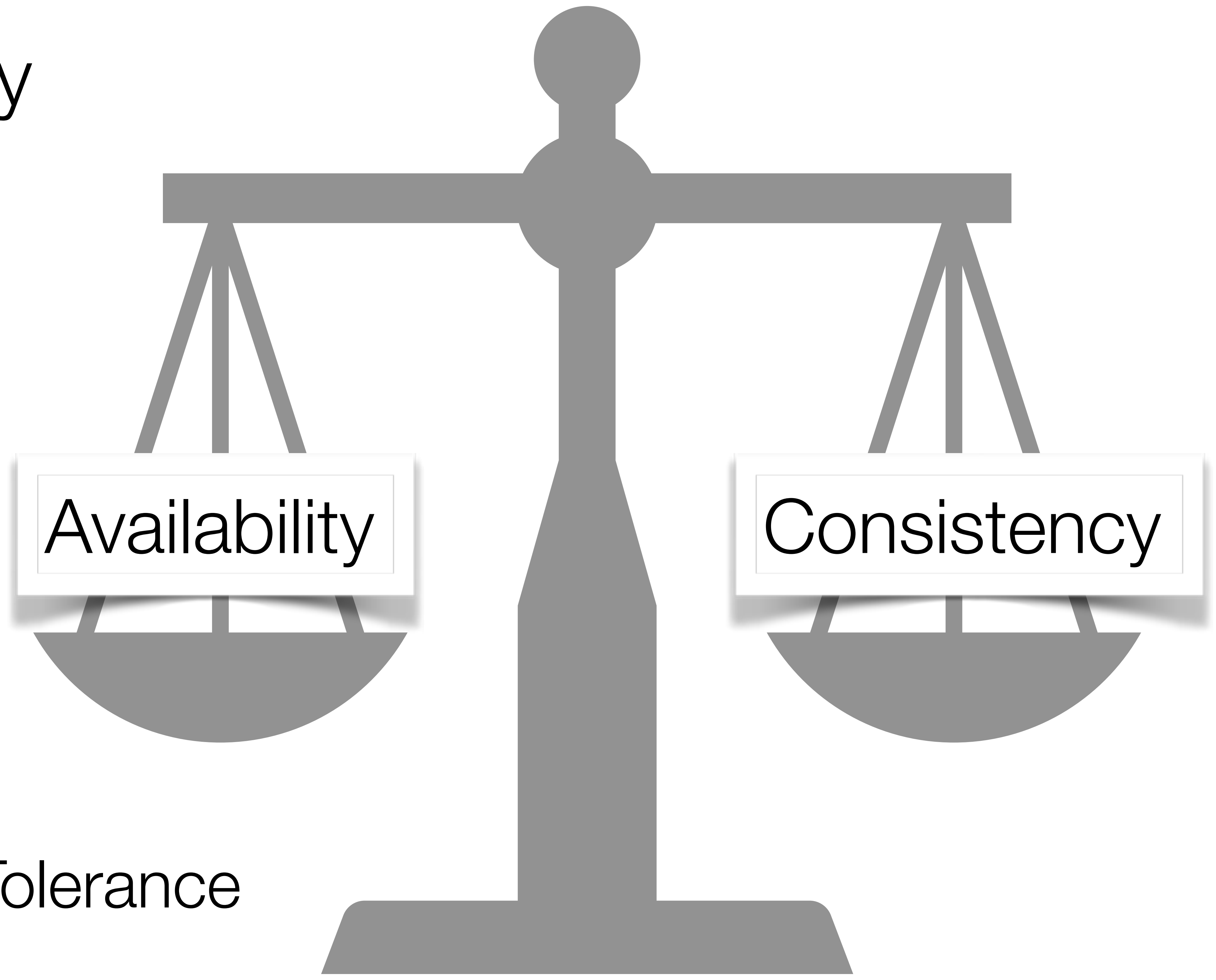
CAP* Theorem

We get:

- Availability

Or:

- Programmability



* Consistency, Availability, Partition Tolerance [Gilbert&Lynch'02]

Distributed State (CRDTs)

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A comprehensive study of Convergent and Commutative Replicated Data Types

Marc Shapiro, INRIA & LIP6, Paris, France
Nuno Preguiça, CITI, Universidade Nova de Lisboa, Portugal
Carlos Baquero, Universidade do Minho, Portugal
Marek Zawirski, INRIA & UPMC, Paris, France

N° 7506

Janvier 2011

Thème COM

Distributed State (CRDTs)

Conflict-Free Replicated Data Types

- Availability
  - Network Partition Tolerance
- (Strong) Eventual Consistency
- Distributed Data Type Abstractions
  - Deterministic Conflict Resolution
    $\Longrightarrow$ Eventual Convergence

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A comprehensive study of
Convergent and Commutative Replicated Data Types

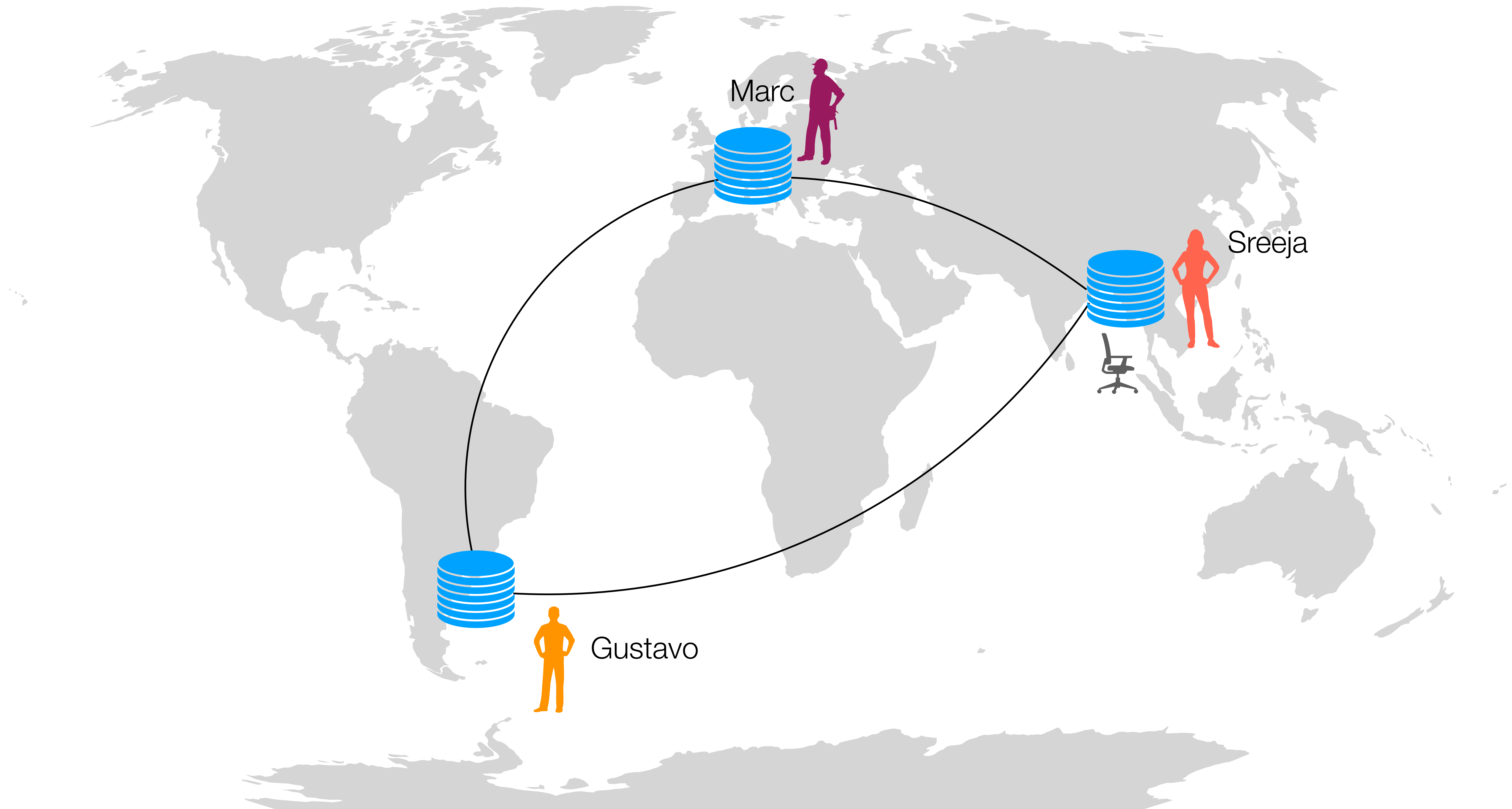
Marc Shapiro, INRIA & LIP6, Paris, France
Nuno Preguiça, CITI, Universidade Nova de Lisboa, Portugal
Carlos Baquero, Universidade do Minho, Portugal
Marek Zawirski, INRIA & UPMC, Paris, France

N° 7506

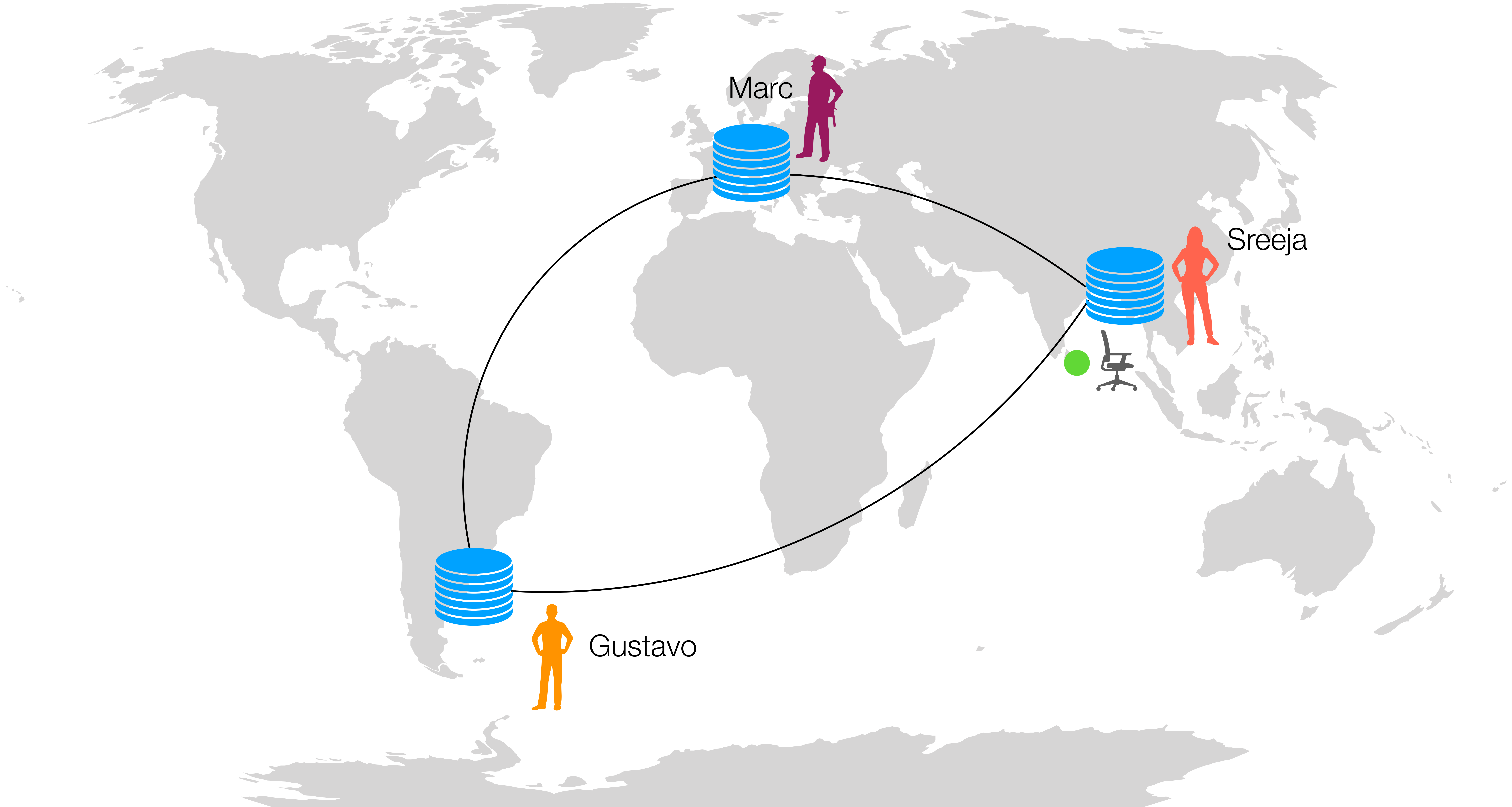
Janvier 2011

Thème COM

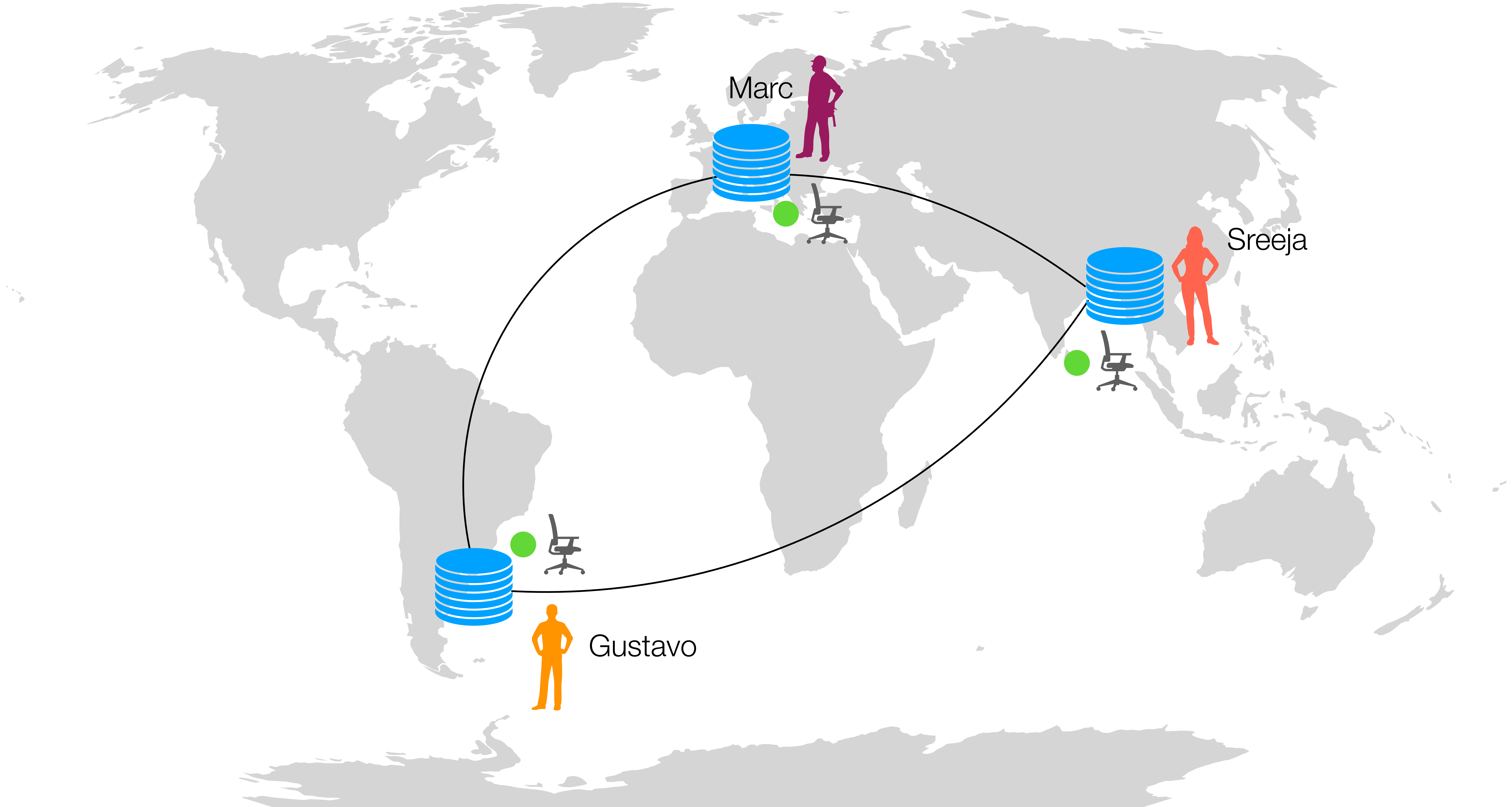
REPLICATED ONLINE AUCTION



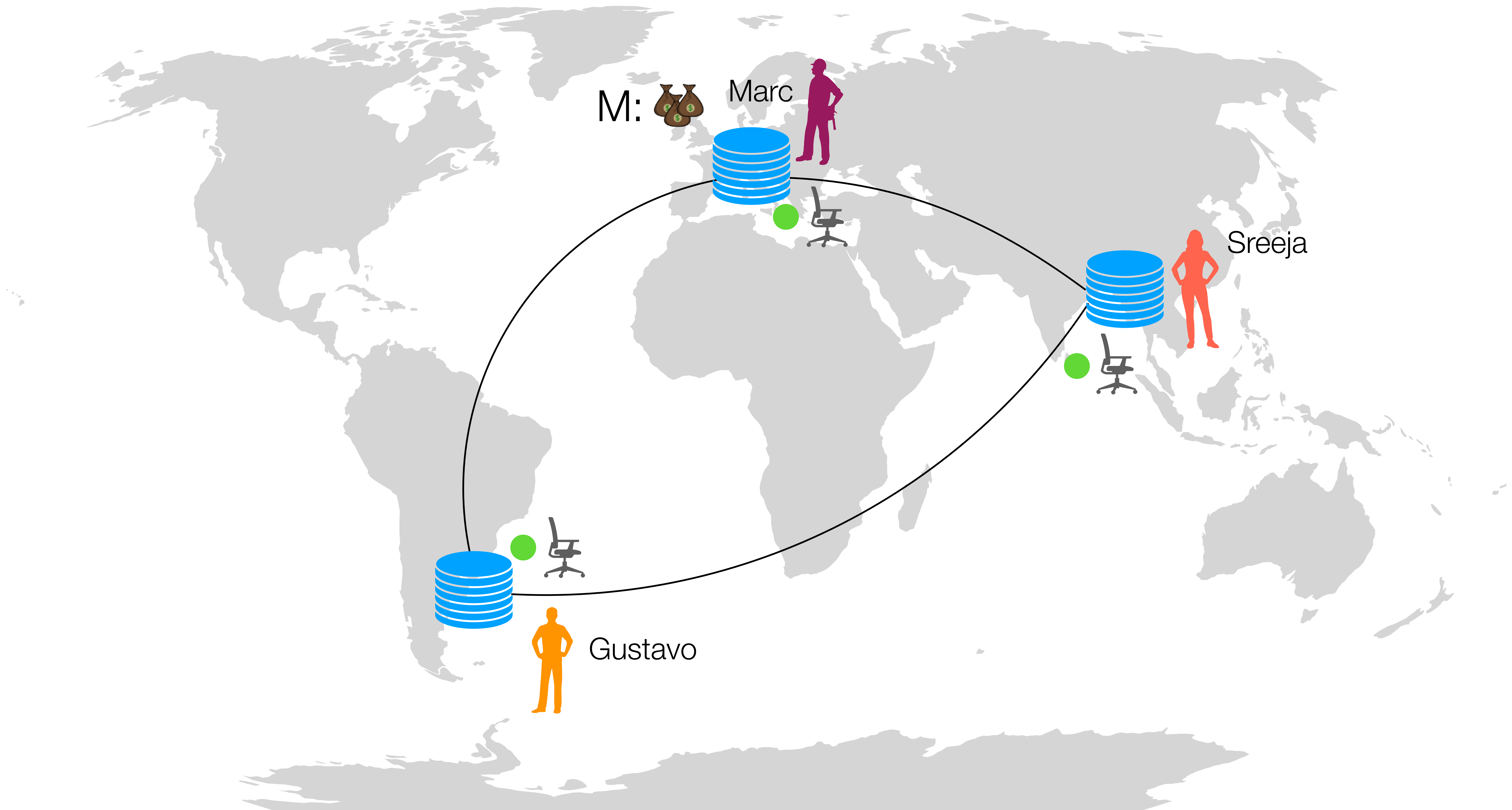
Replicated Online Auction



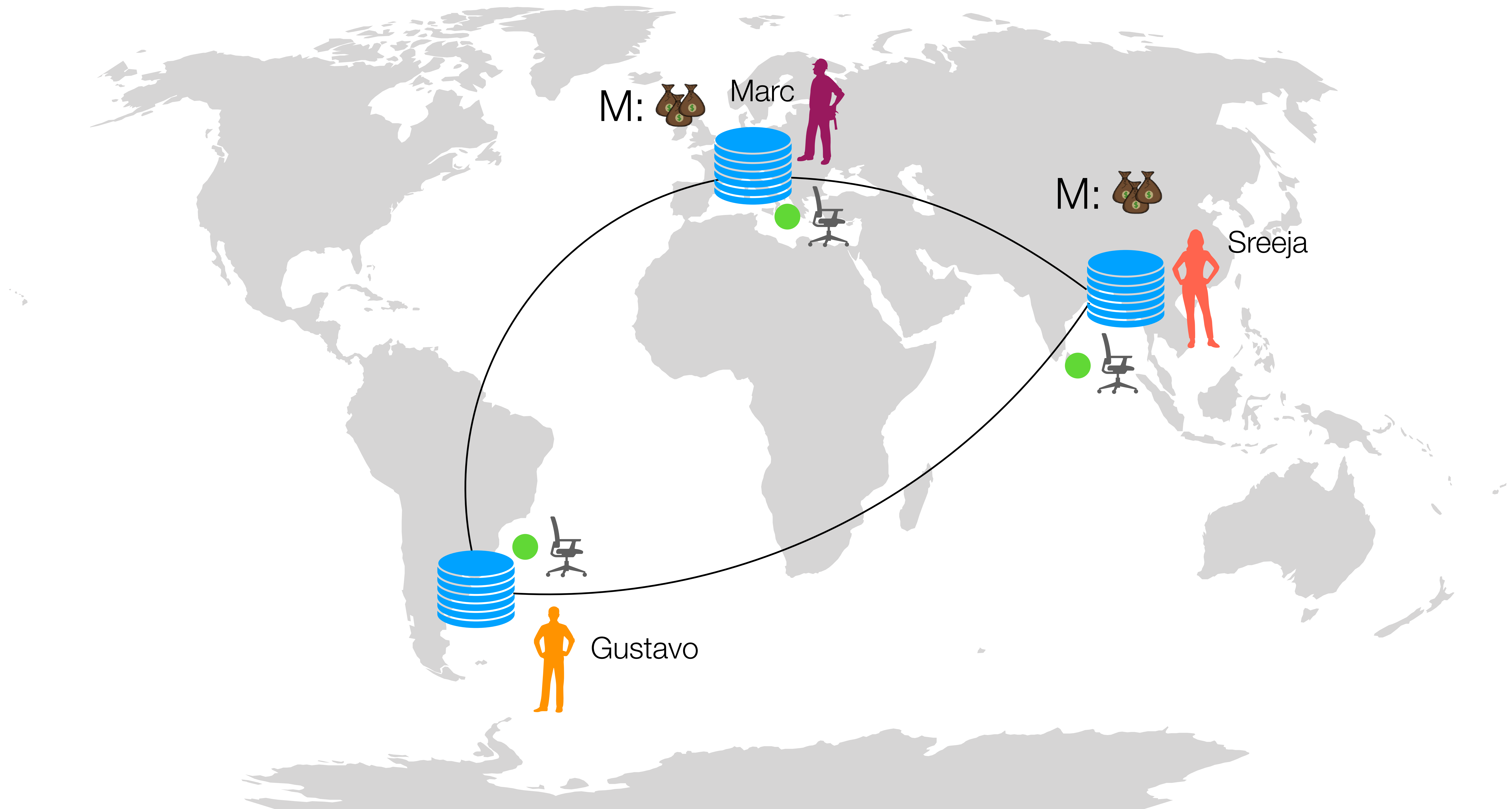
Replicated Online Auction



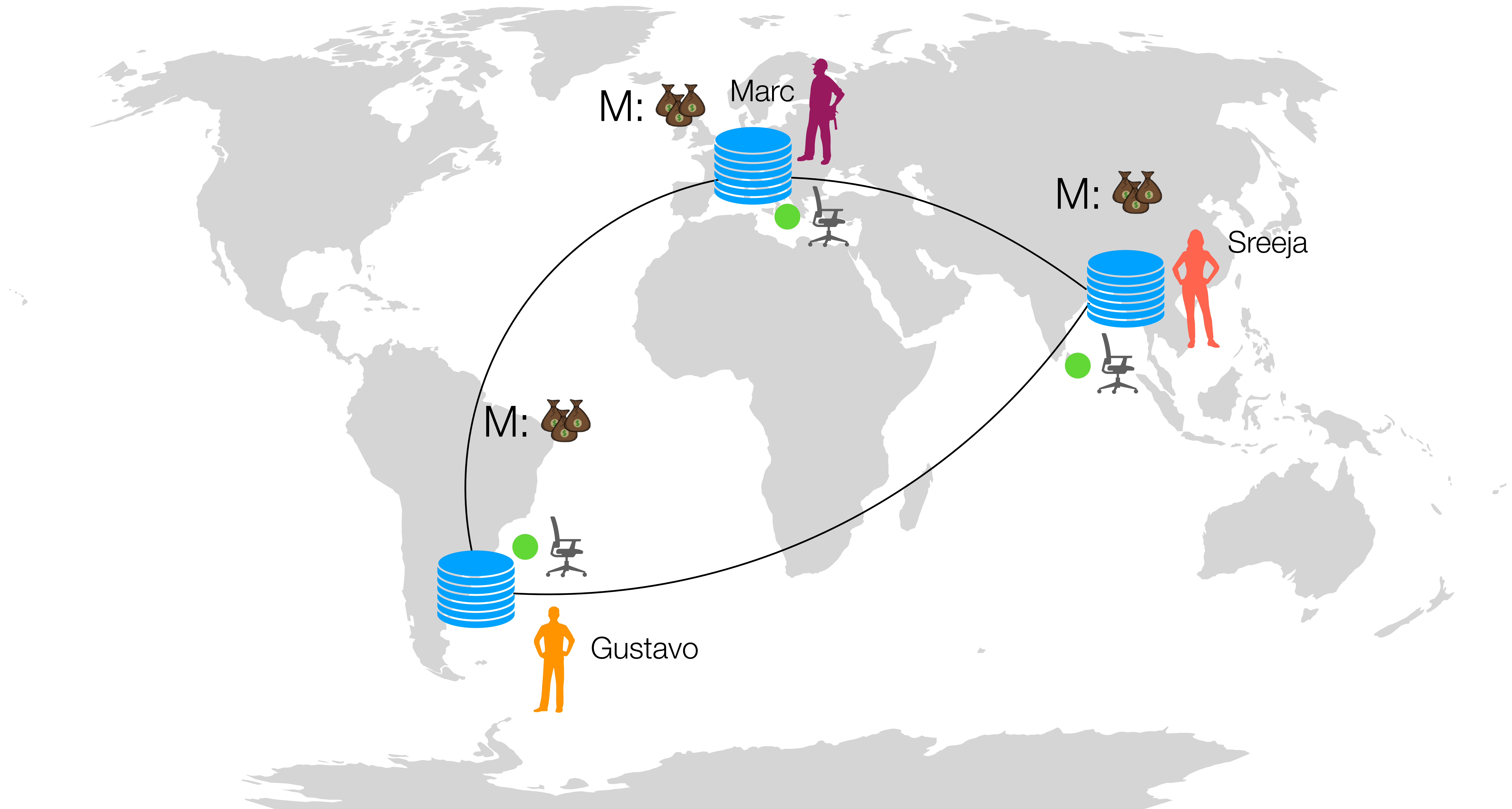
Replicated Online Auction



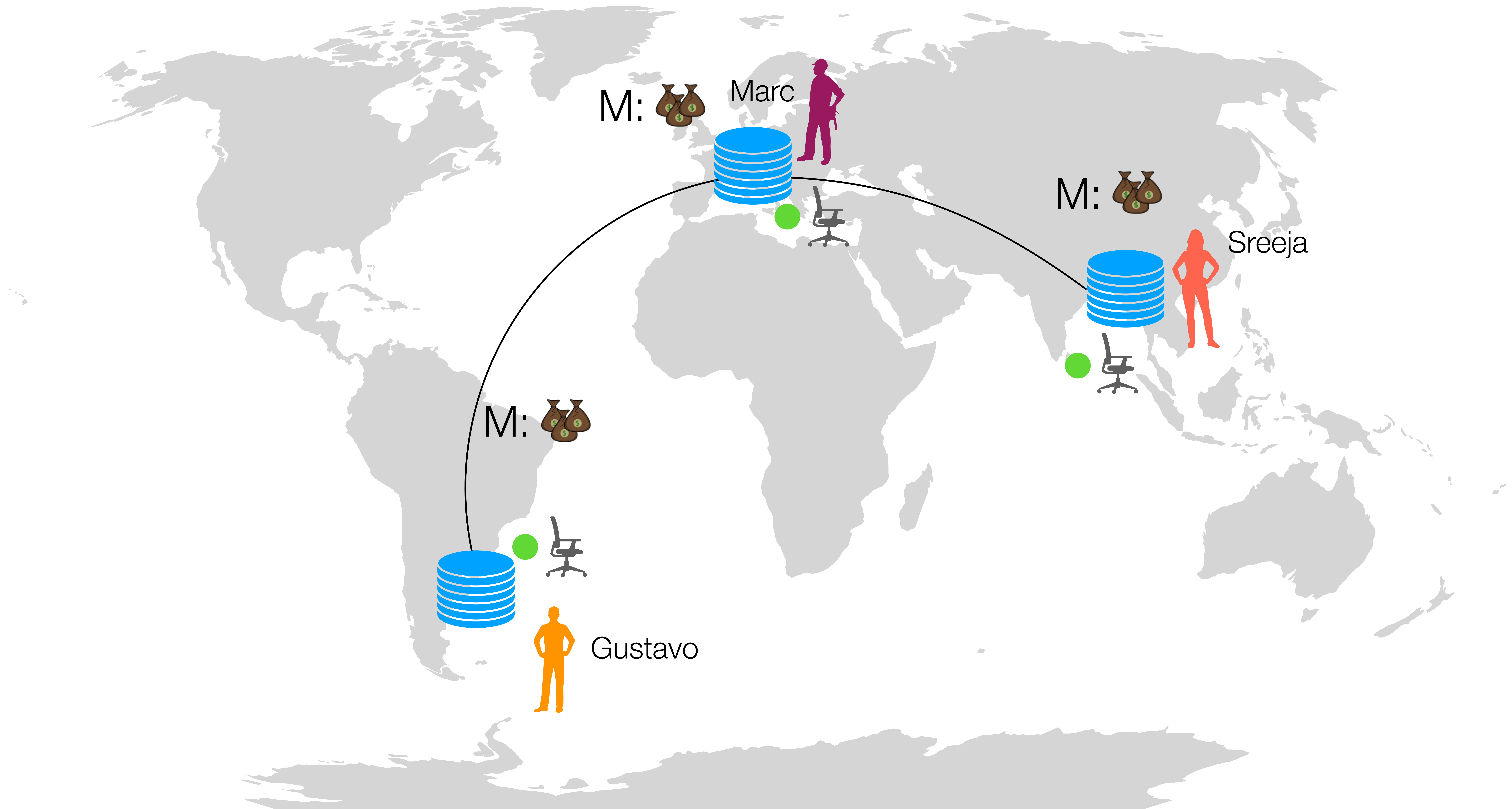
Replicated Online Auction



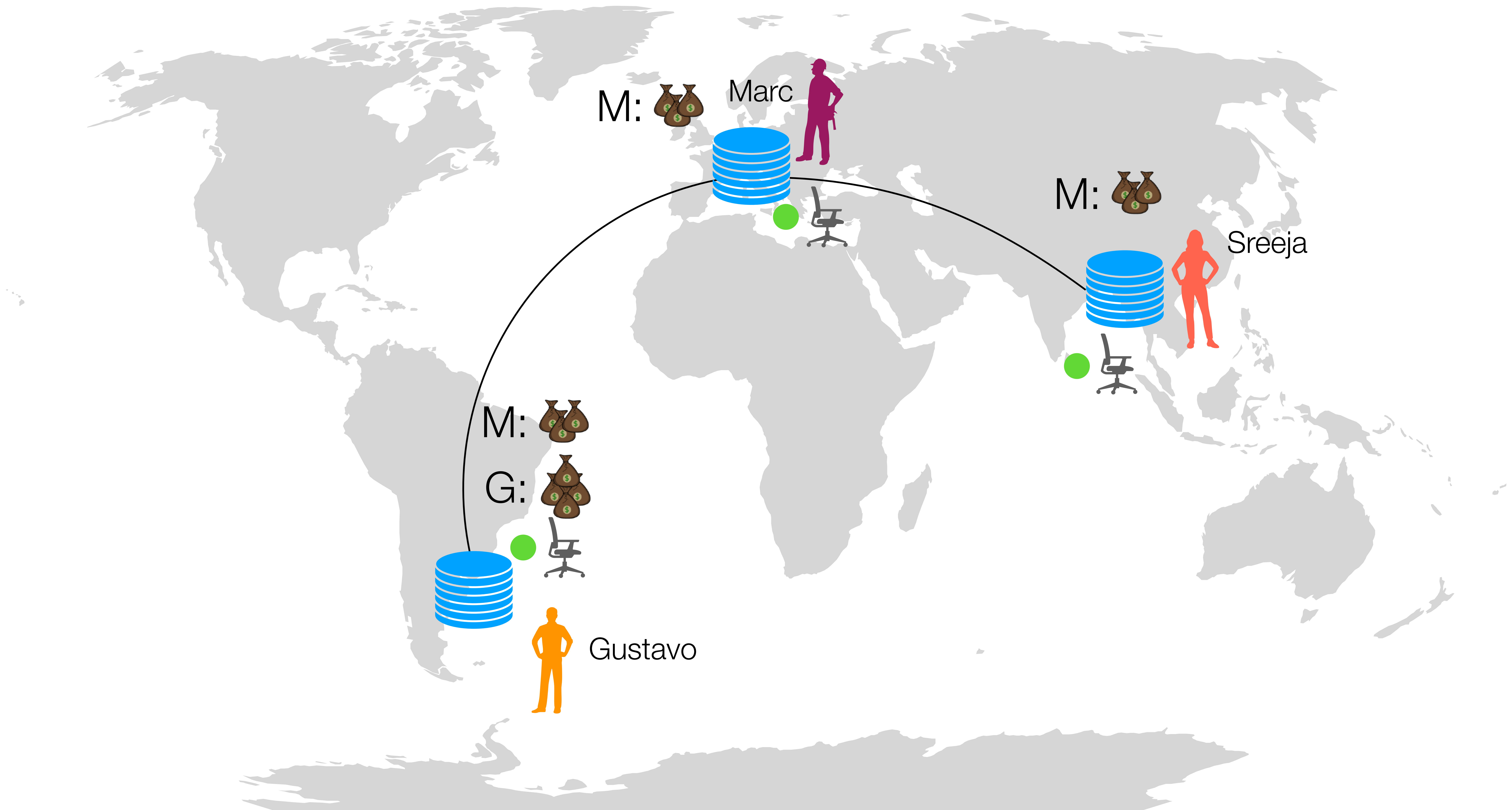
Replicated Online Auction



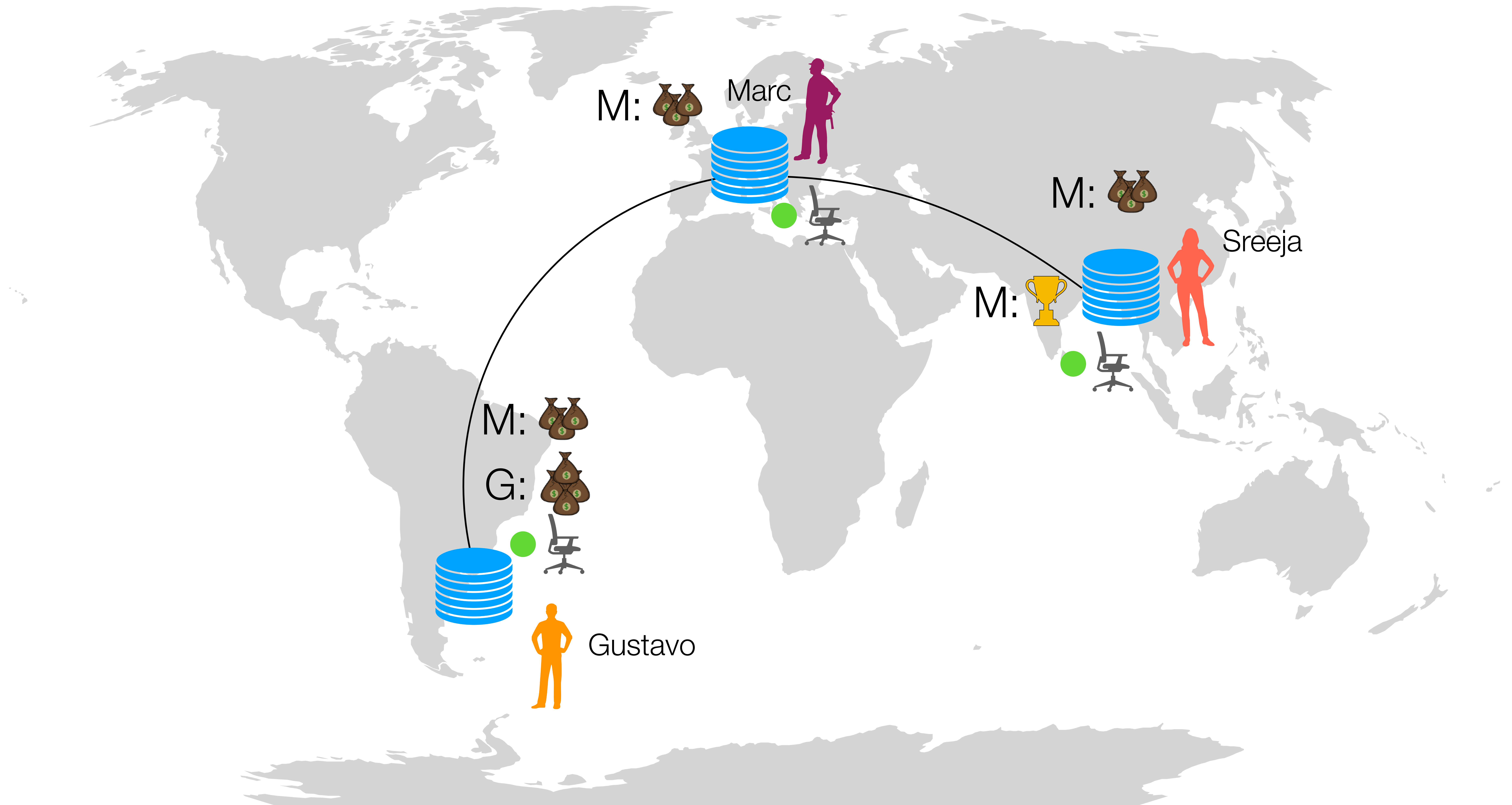
Replicated Online Auction



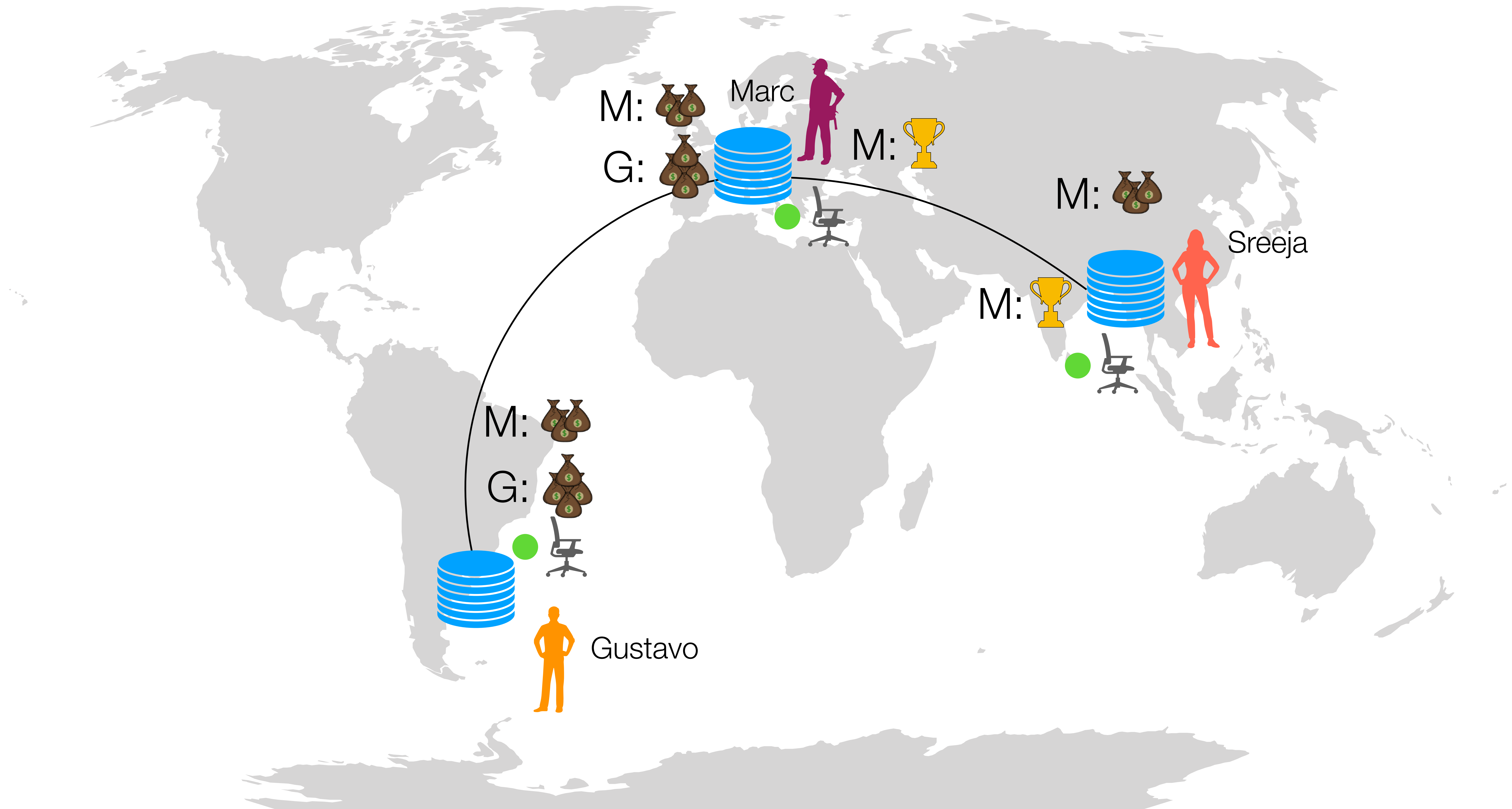
Replicated Online Auction



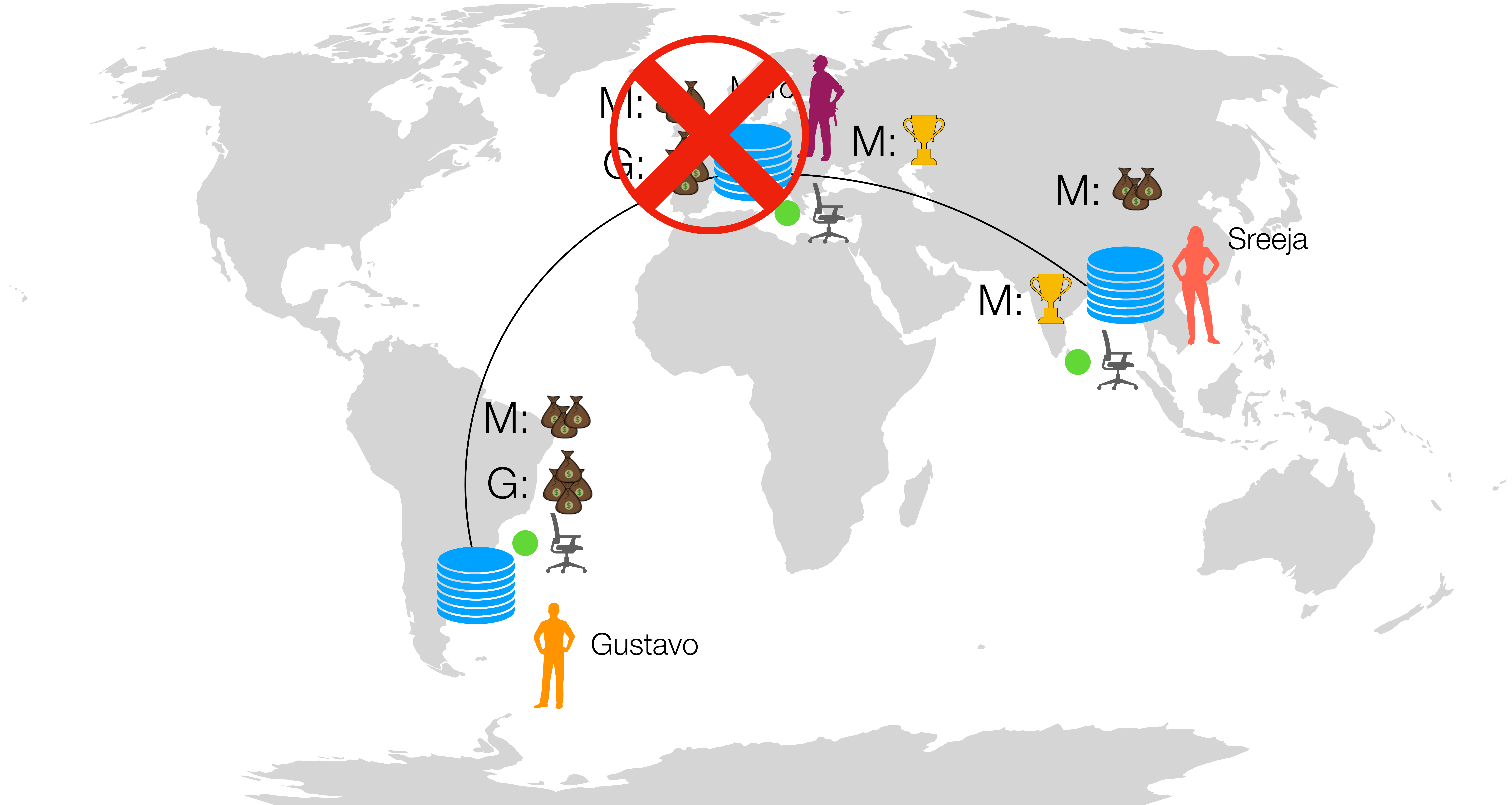
Replicated Online Auction



Replicated Online Auction



Replicated Online Auction



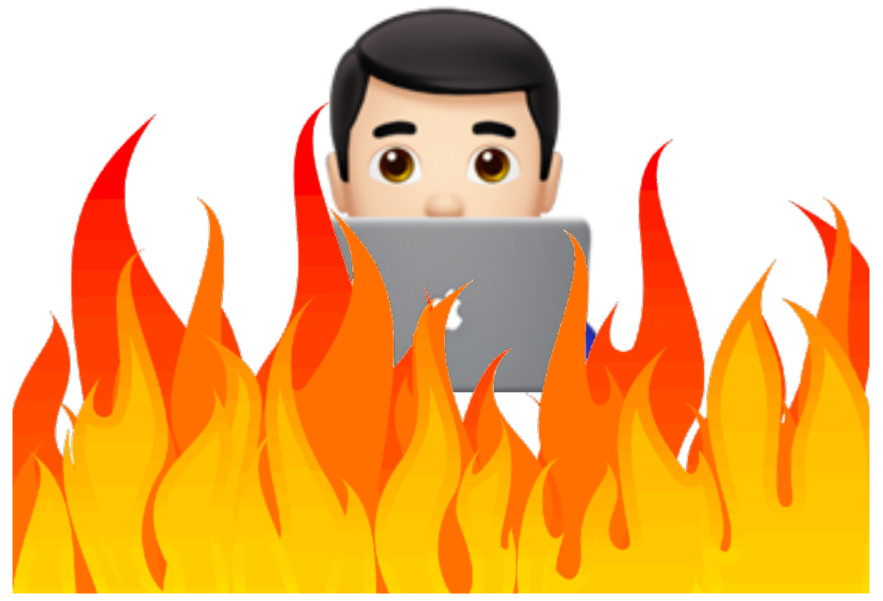
Safety for Distributed Applications

Safety for Distributed Applications

- High Availability
- Strong Consistency

Safety for Distributed Applications

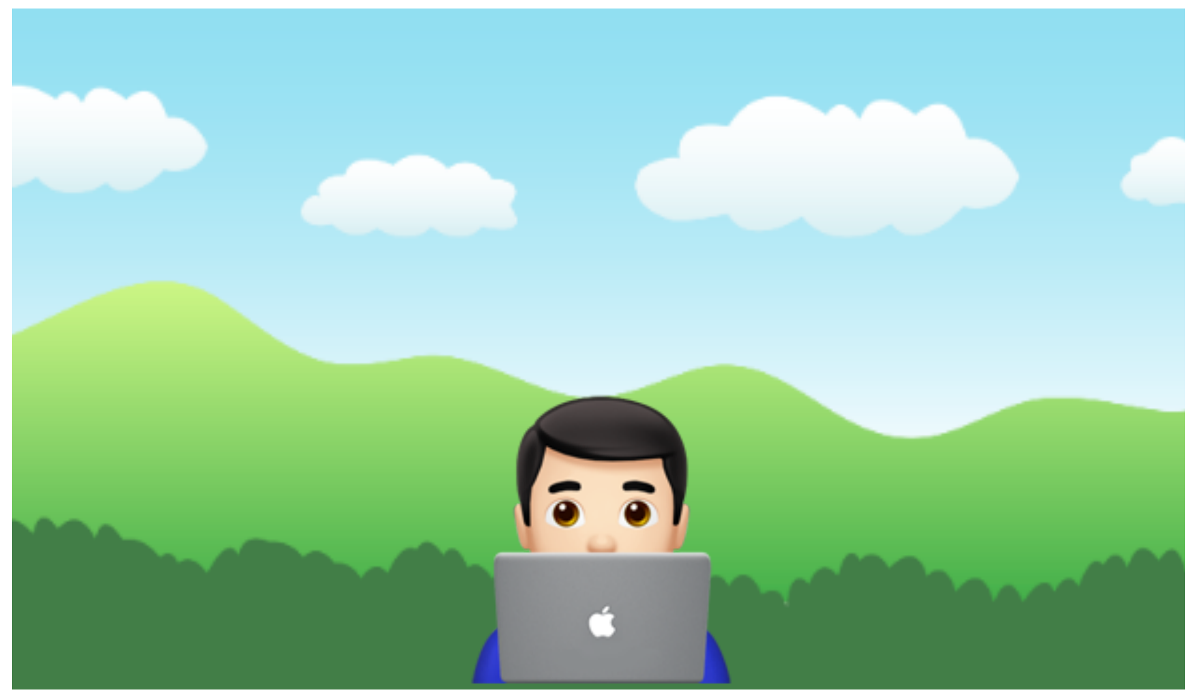
- High Availability
- Strong Consistency



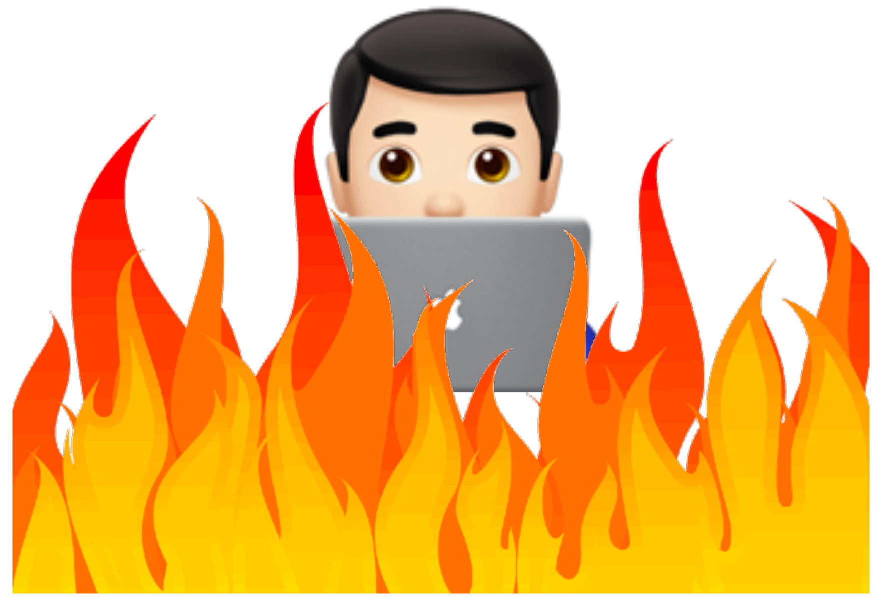
- High Availability
- Eventual Consistency

Safety for Distributed Applications

- High Availability
- Strong Consistency



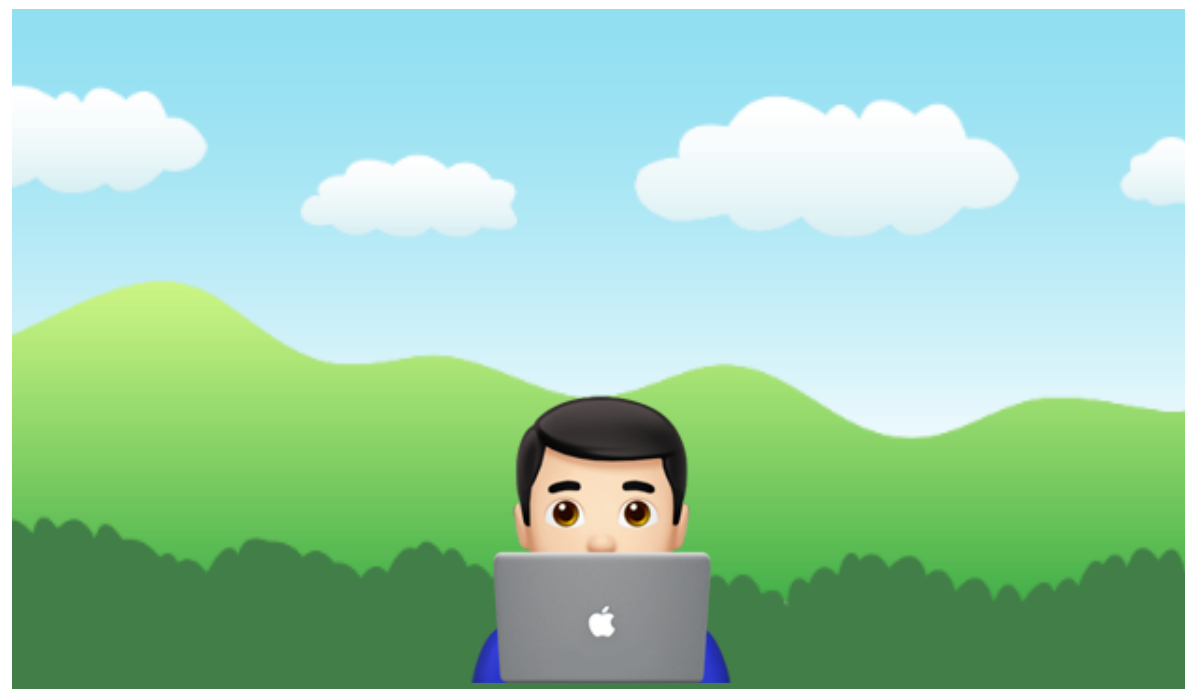
- High Availability
- Eventual Consistency
- Data Safety



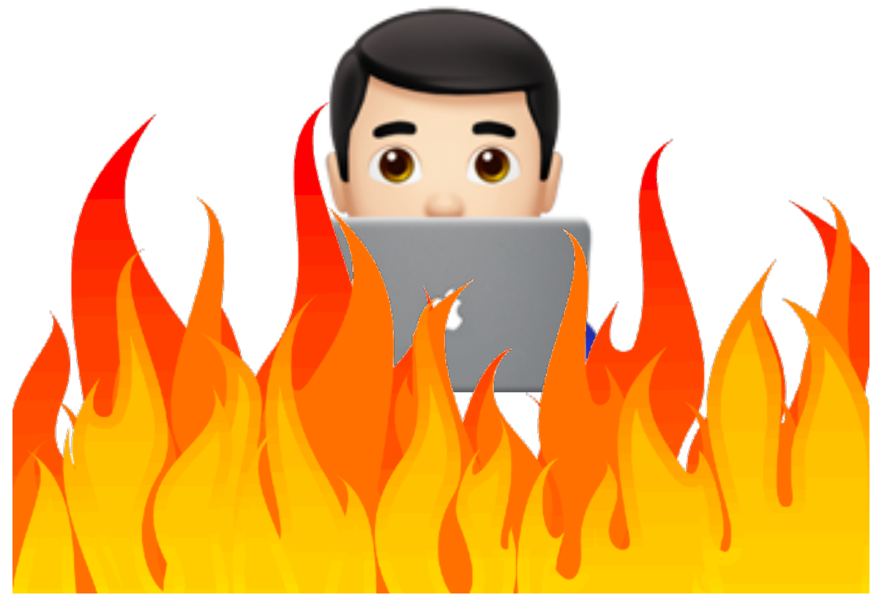
- High Availability
- Eventual Consistency

Safety for Distributed Applications

- High Availability
- Strong Consistency



- High Availability
- Eventual Consistency
- Data Safety



- High Availability
- Eventual Consistency

Proof rule for Stateful Distributed Application Safety

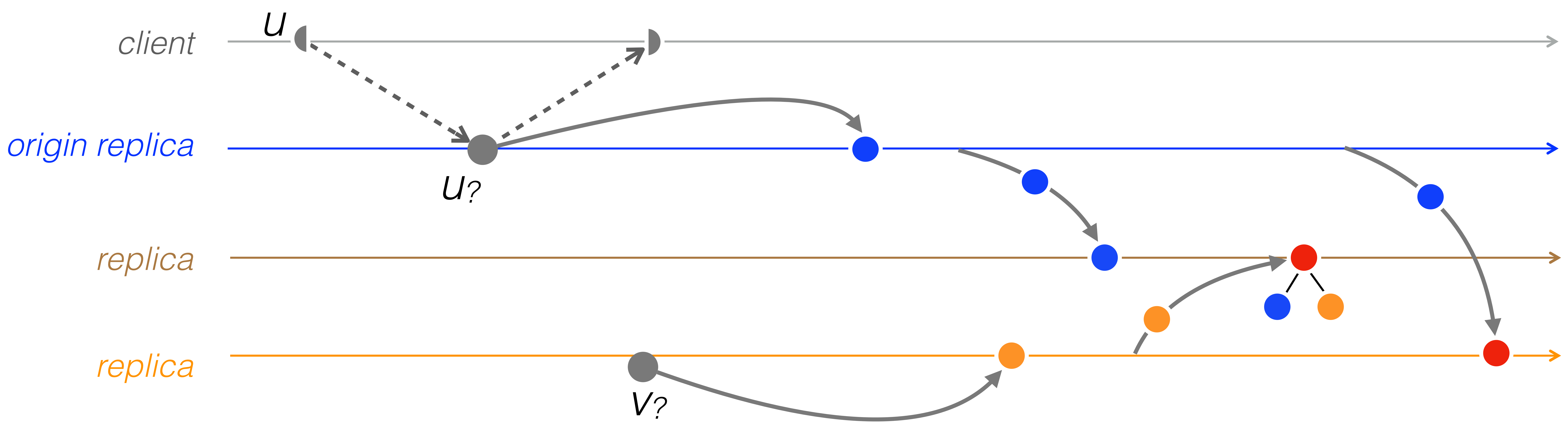
- Modular
- Automated verification

State-based CRDTs

- State-based CRDTs
  - Propagation of states (instead of operations)

State-based CRDTs

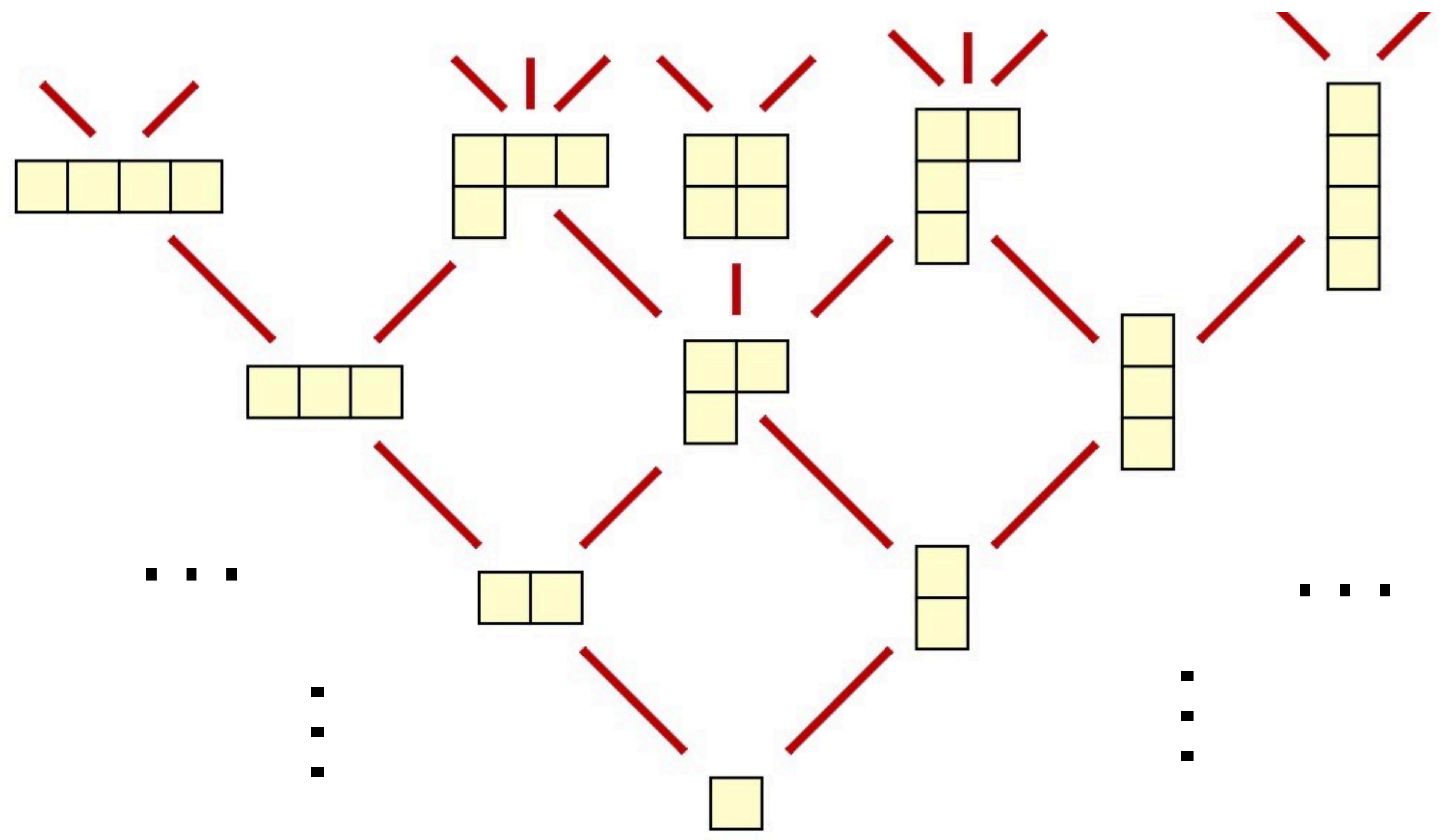
- State-based CRDTs
  - Propagation of states (instead of operations)



State-based CRDTs

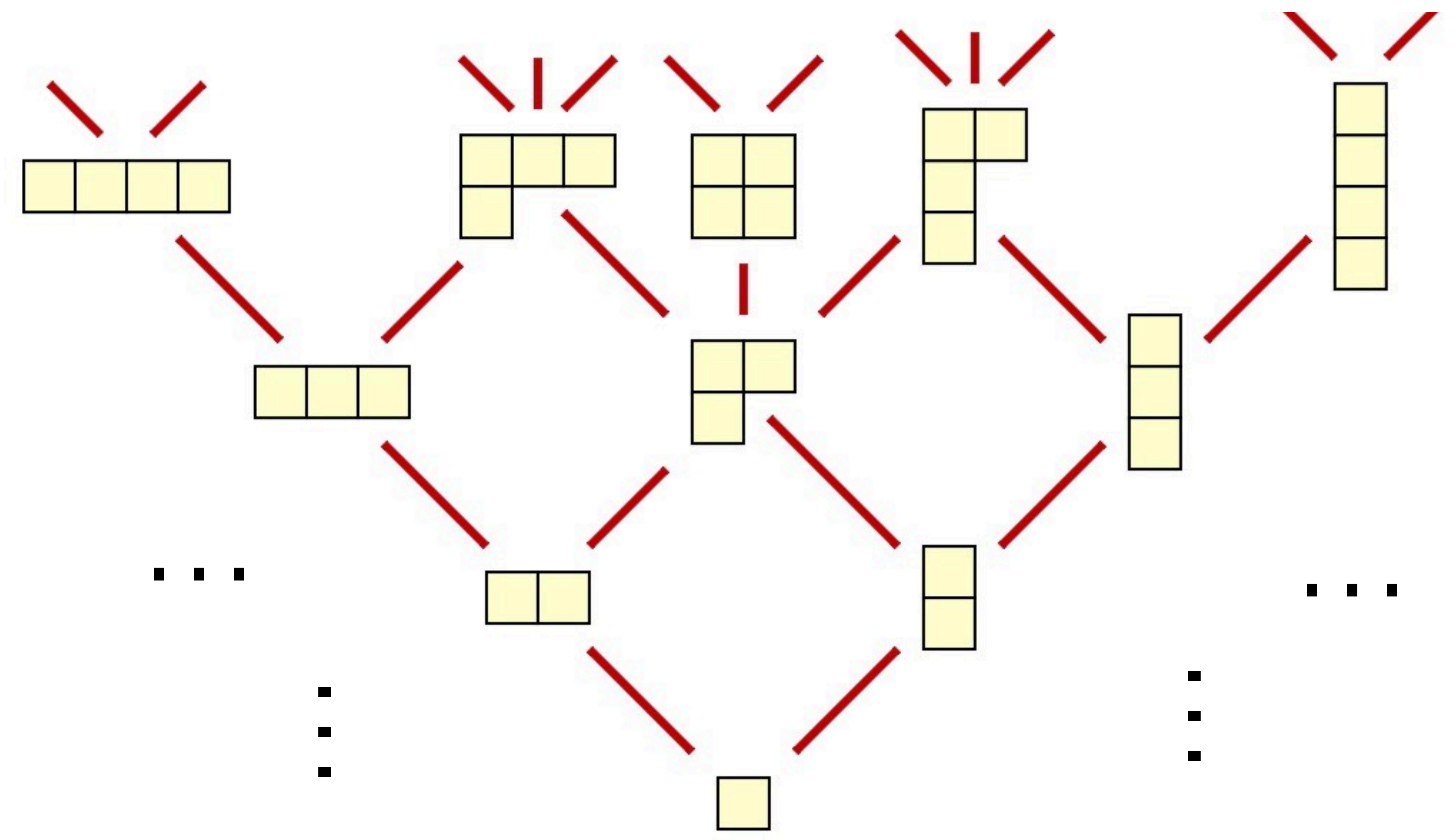
- State-based CRDTs
  - Propagation of states (instead of operations)
- States are **`merged`** on receive
  - Convergence: *concurrent conflicting* operations result *deterministically* on a unique state
  - No delivery assumptions

STATE-BASED CRDTS



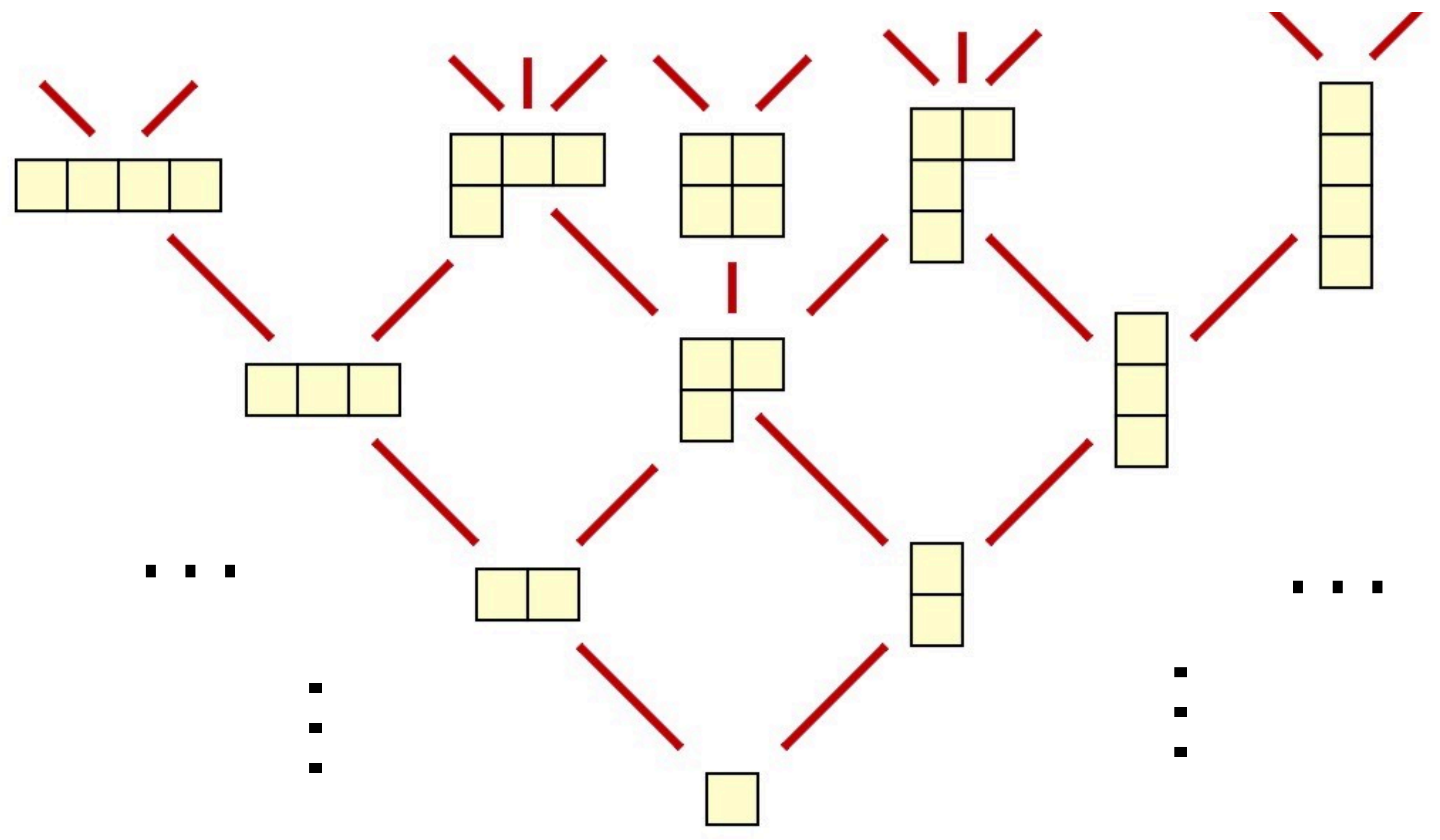
STATE-BASED CRDTS

- State is a (join semi-)Lattice



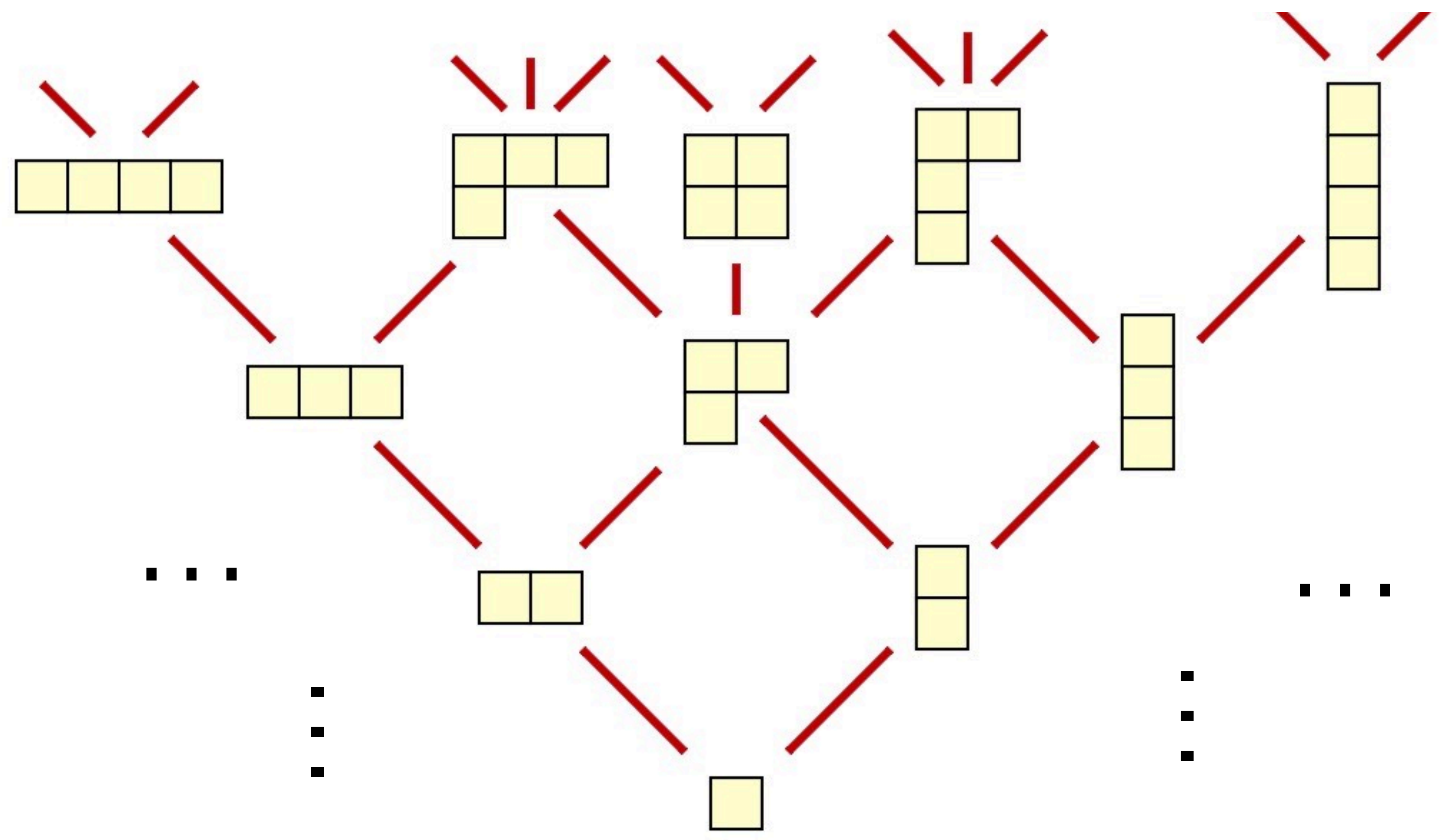
State-based CRDTs

- State is a (join semi-)Lattice
- Effectors send the state at the origin
  - Lazy update propagation



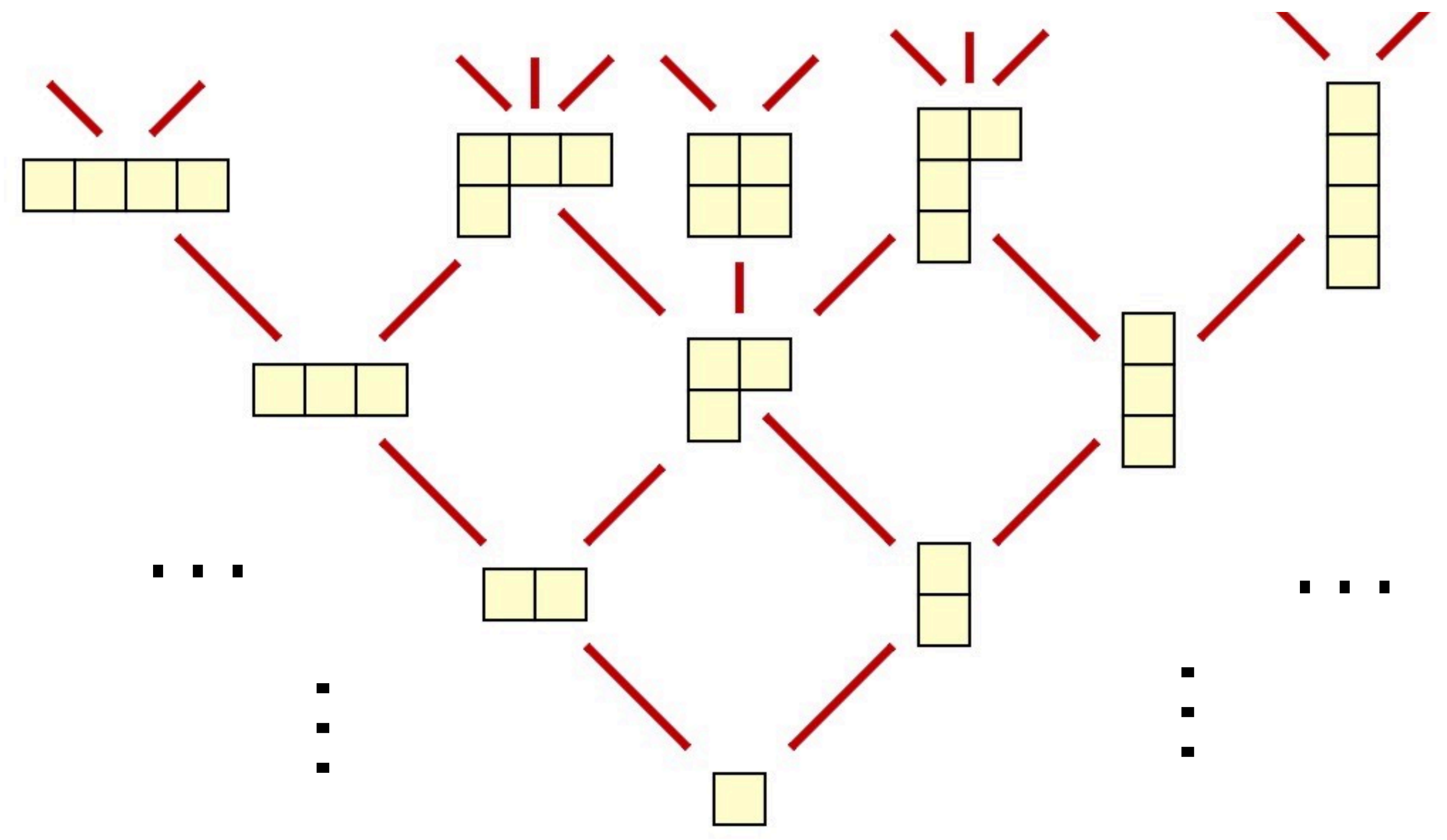
STATE-BASED CRDTs

- State is a (join semi-)Lattice
- Effectors send the state at the origin
  - Lazy update propagation
- Each operation is an inflation in the lattice



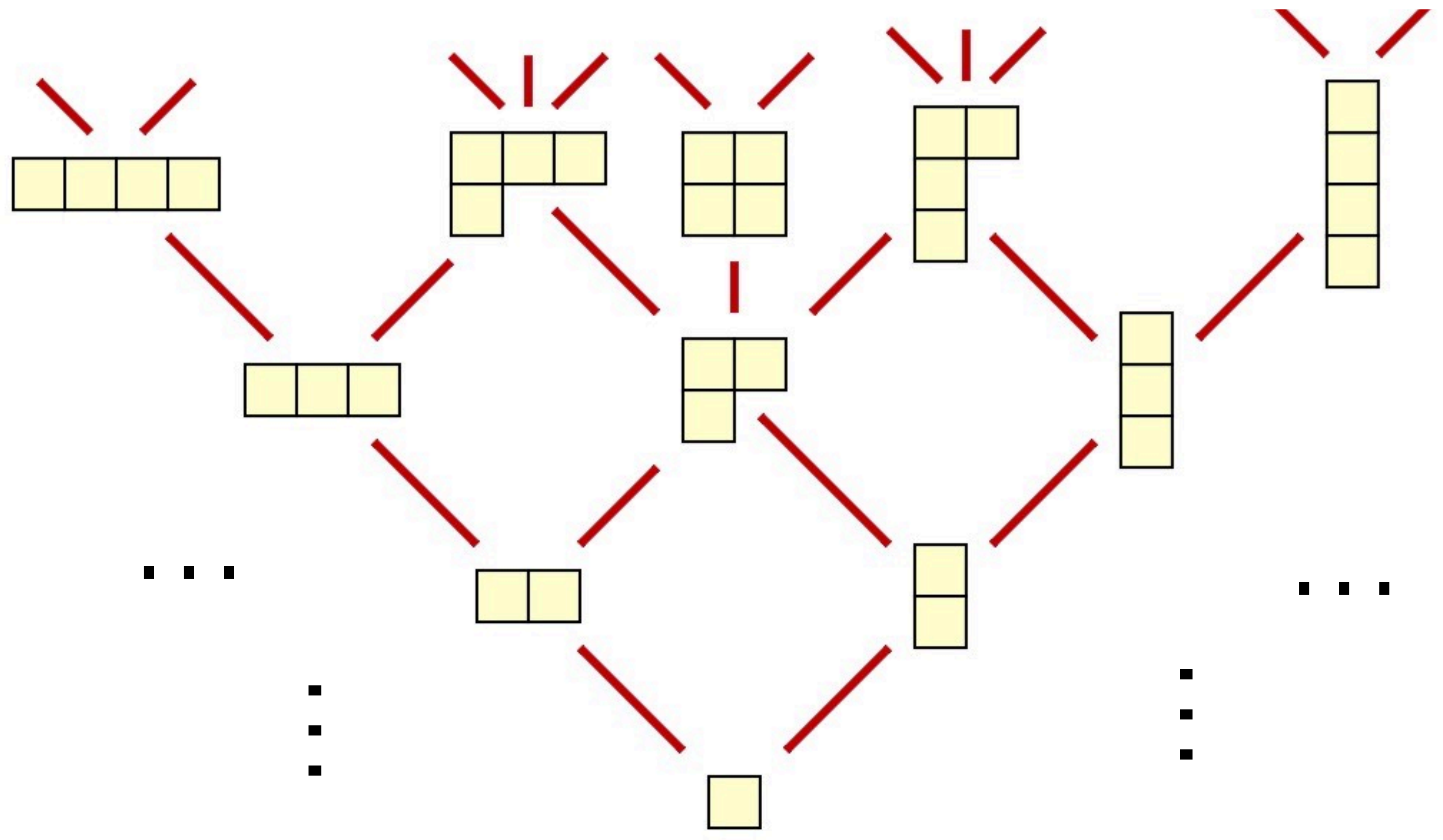
State-based CRDTs

- State is a (join semi-)Lattice
- Effectors send the state at the origin
  - Lazy update propagation
- Each operation is an inflation in the lattice
- **`merge`** function joins the state of two replicas
  - Join of the lattice

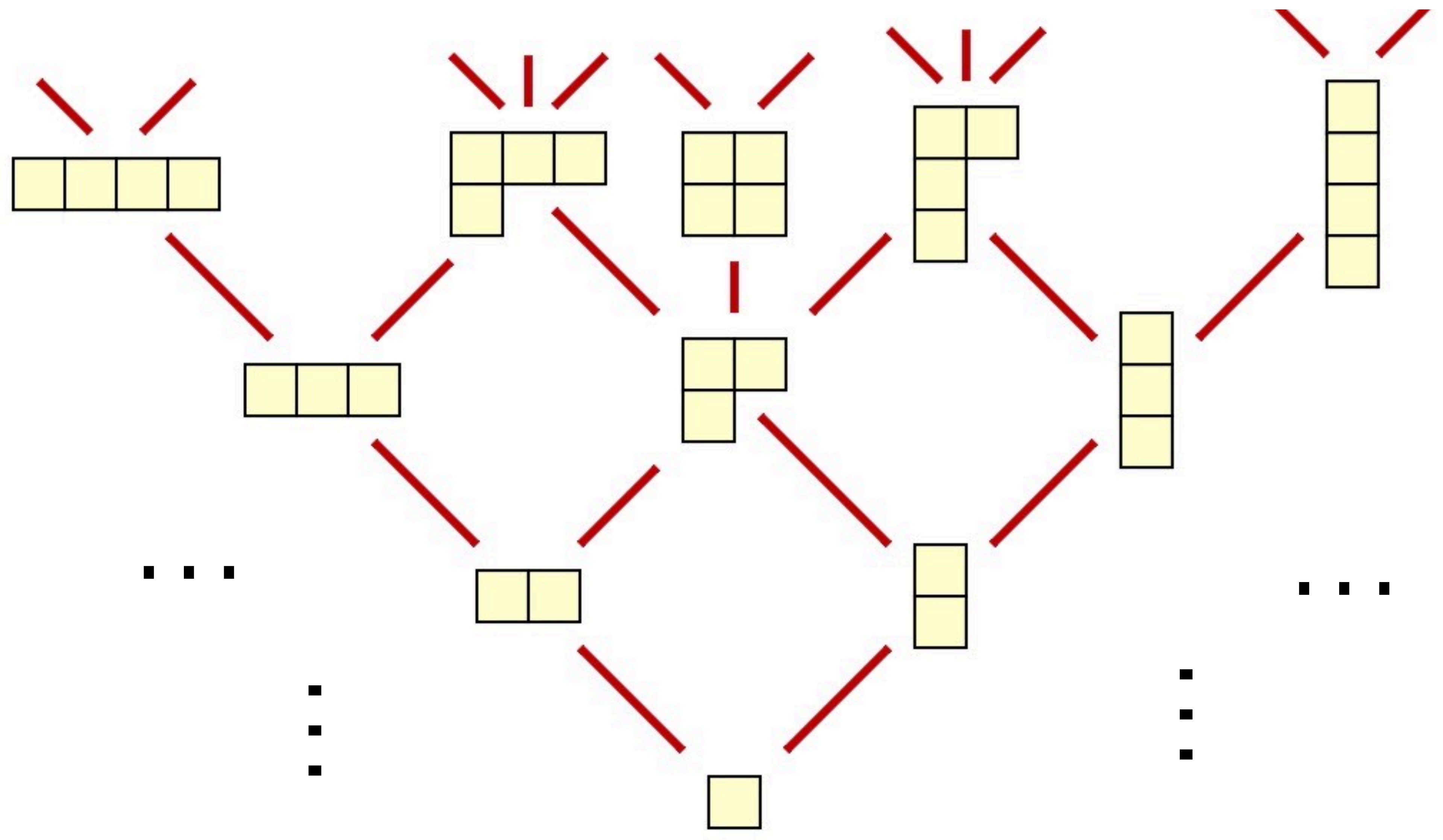


Invariants for SB-CRDTs

- CRDT (lattice) constraints



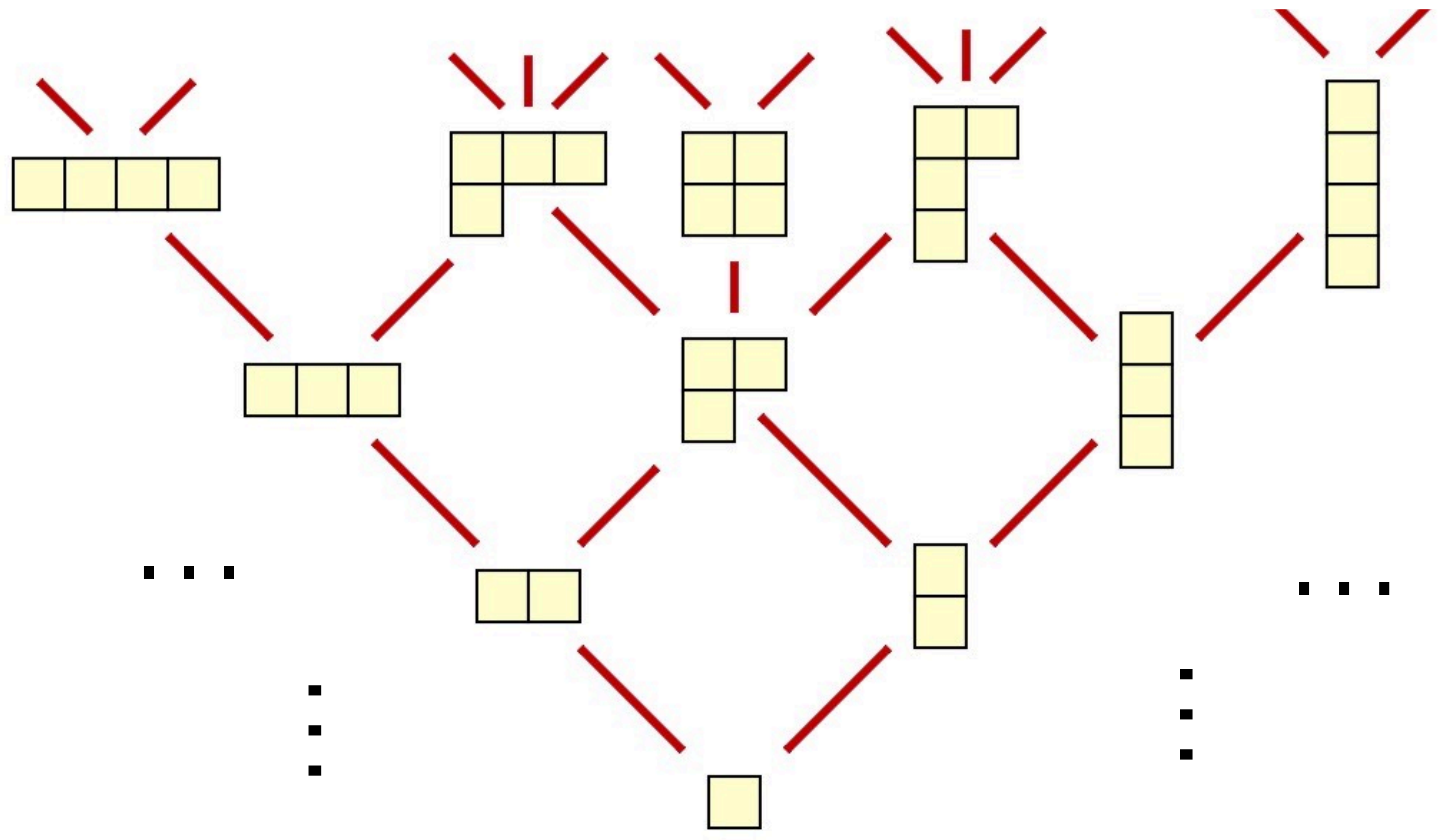
INVARIANTS FOR SB-CRDTS



- CRDT (lattice) constraints
  - Operations are inflations

$$\forall\ \mathsf{op}, \sigma, \sigma',\ \ \sigma \models \mathsf{Pre}_{\mathsf{op}}\ \wedge\ (\sigma, \sigma') \in [\![\mathsf{op}]\!]\ \Rightarrow \sigma \sqsubseteq \sigma'$$

Invariants for SB-CRDTs



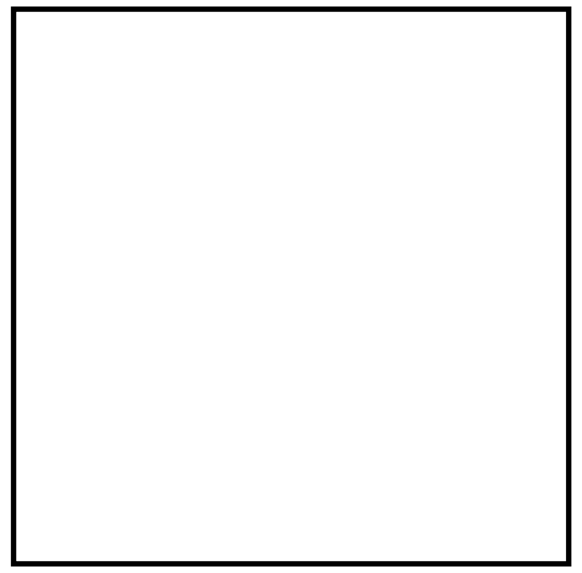
- CRDT (lattice) constraints
  - Operations are inflations

$$\forall\ \mathsf{op}, \sigma, \sigma',\ \ \sigma \models \mathsf{Pre}_{\mathsf{op}}\ \ \wedge\ \ (\sigma, \sigma') \in [\![\mathsf{op}]\!]\ \ \Rightarrow \sigma \sqsubseteq \sigma'$$

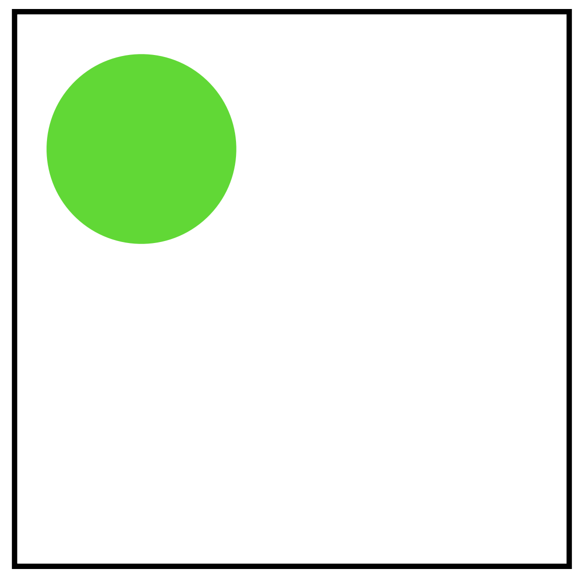
  - **`merge`** is join (LUB)

$$\forall \sigma, \sigma',\ \ \mathtt{merge}(\sigma, \sigma') = \sigma'' \Rightarrow \sigma'' = \mathsf{LUB}_{\sqsubseteq}(\sigma, \sigma')$$

Auction State Evolution



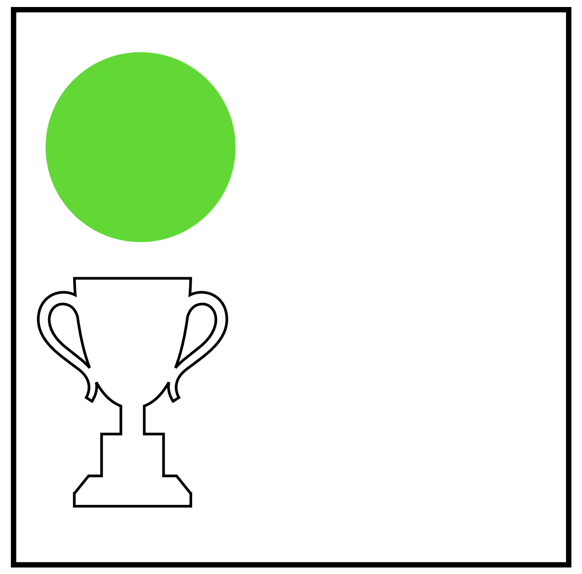
Auction State Evolution



Auction status:

🟢	Open
🔴	Closed

Auction State Evolution



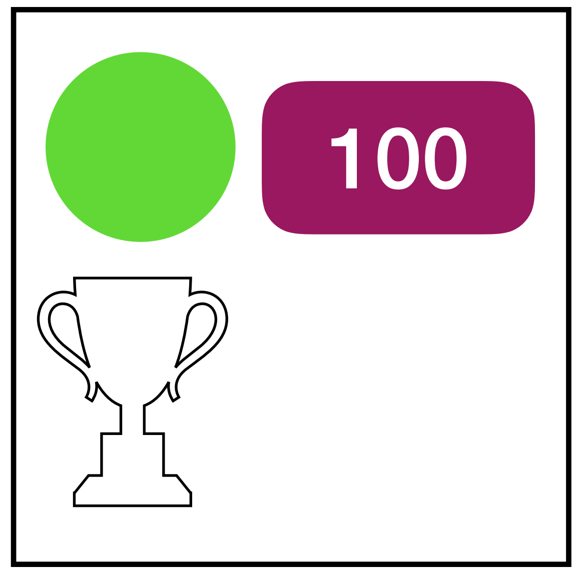
Auction status: 🟢 Open

🔴 Closed

Auction result: 🏆 No winner

🏆 Winner marker

Auction State Evolution



Auction status: 🟢 Open

🔴 Closed

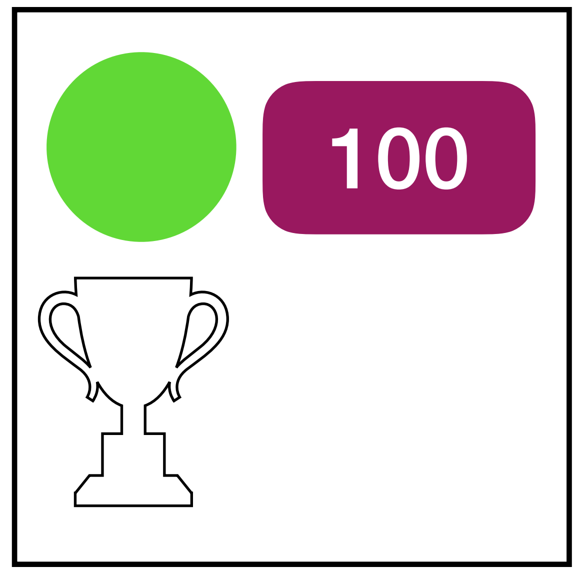
Auction Result: 🏆 No winner

🏆 Winner marker

Auction Bidders: 100

110

Auction State Evolution



Auction status: 🟢 Open
🔴 Closed

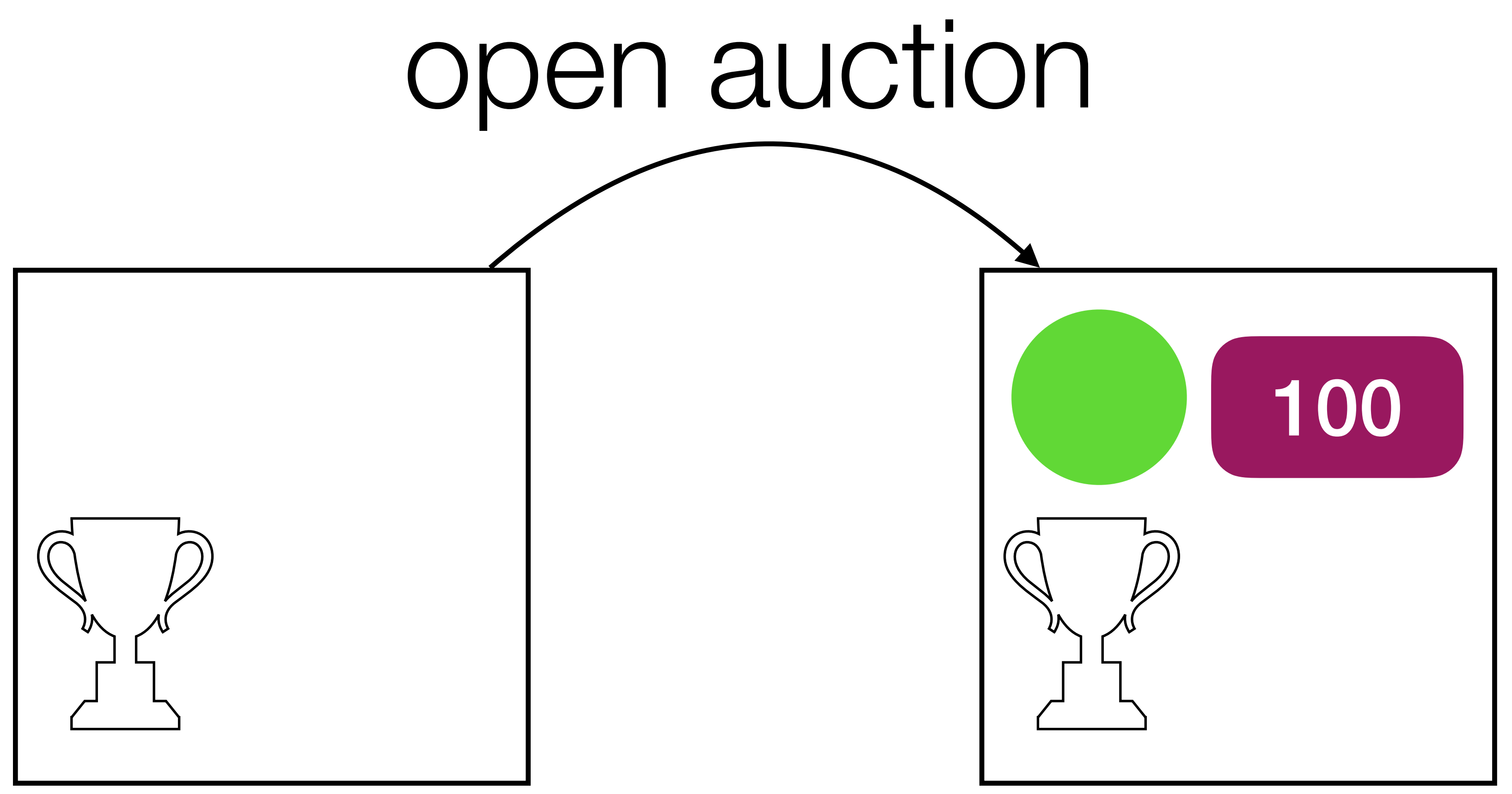
Auction Result: 🏆 No winner
🏆 Winner marker

Auction Bidders: 100 🧍
110 🧍

Auction Operations:

- Open auction
- Place bid
- Close auction
- Select winner

Auction State Evolution



Auction status: 🟢 Open
🔴 Closed

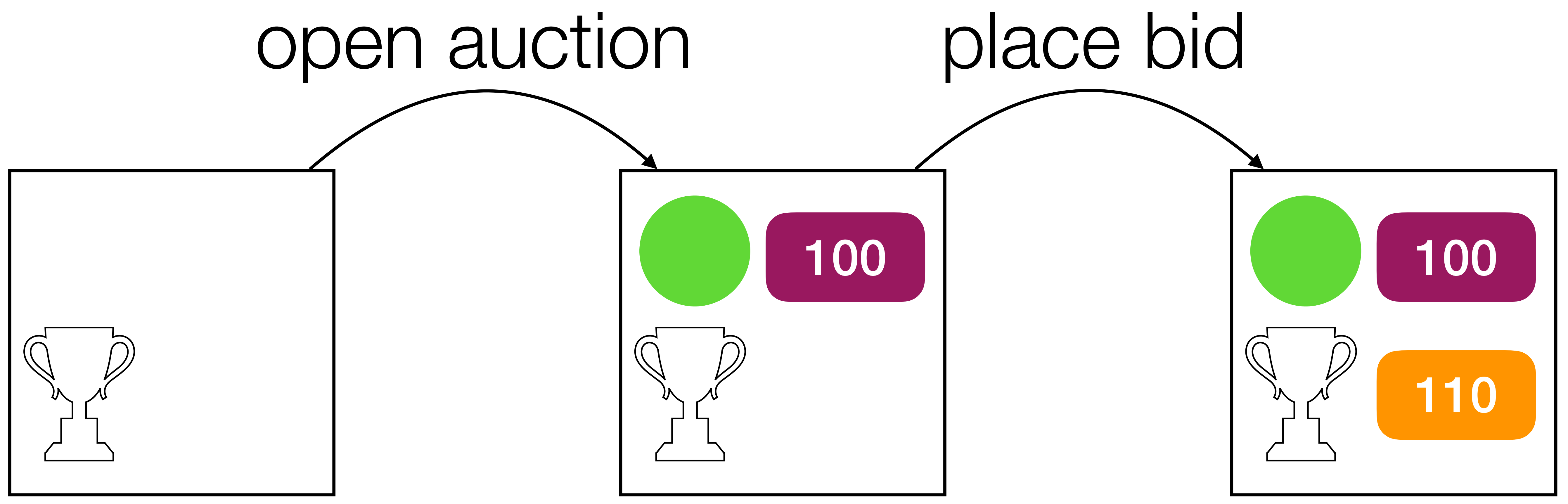
Auction result: 🏆 No winner
🏆 Winner marker

Auction bidders: 100 🟪
110 🟧

Auction operations:

- Open auction
- Place bid
- Close auction
- Select winner

Auction State Evolution



Auction status: 🟢 Open
🔴 Closed

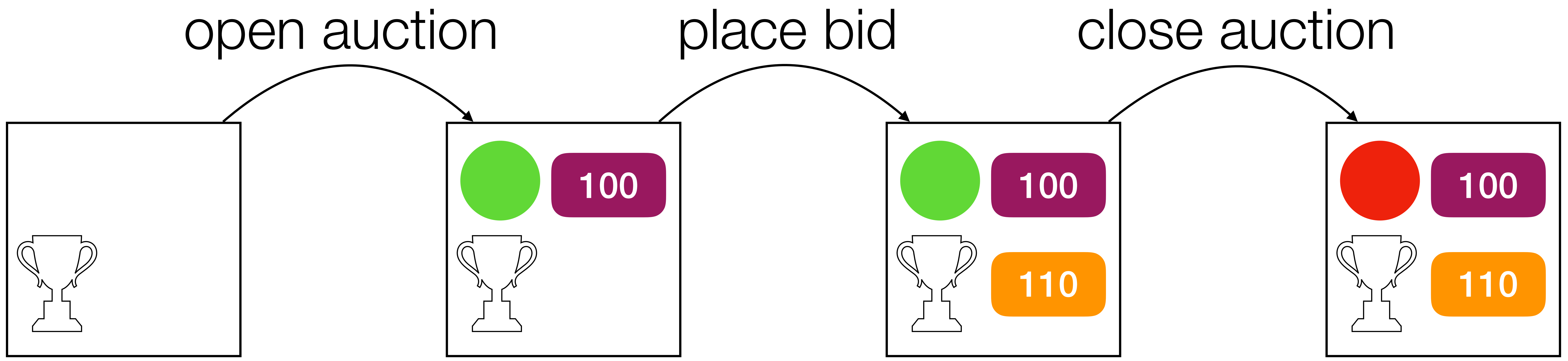
Auction Result: 🏆 No winner
🏆 Winner marker

Auction Bidders: 100
110

Auction Operations:

- Open auction
- Place bid
- Close auction
- Select winner

Auction State Evolution



Auction status: 🟢 Open
🔴 Closed

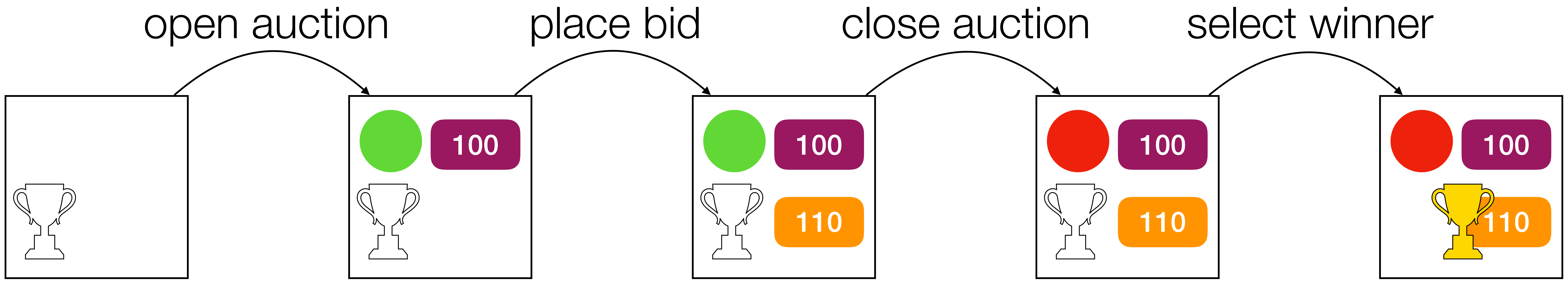
Auction bidders: 100
110

Auction result: 🏆 No winner
🏆 Winner marker

Auction operations:

- Open auction
- Place bid
- Close auction
- Select winner

Auction State Evolution



Auction status: 🟢 Open

🔴 Closed

Auction bidders: 100 🟪

110 🟧

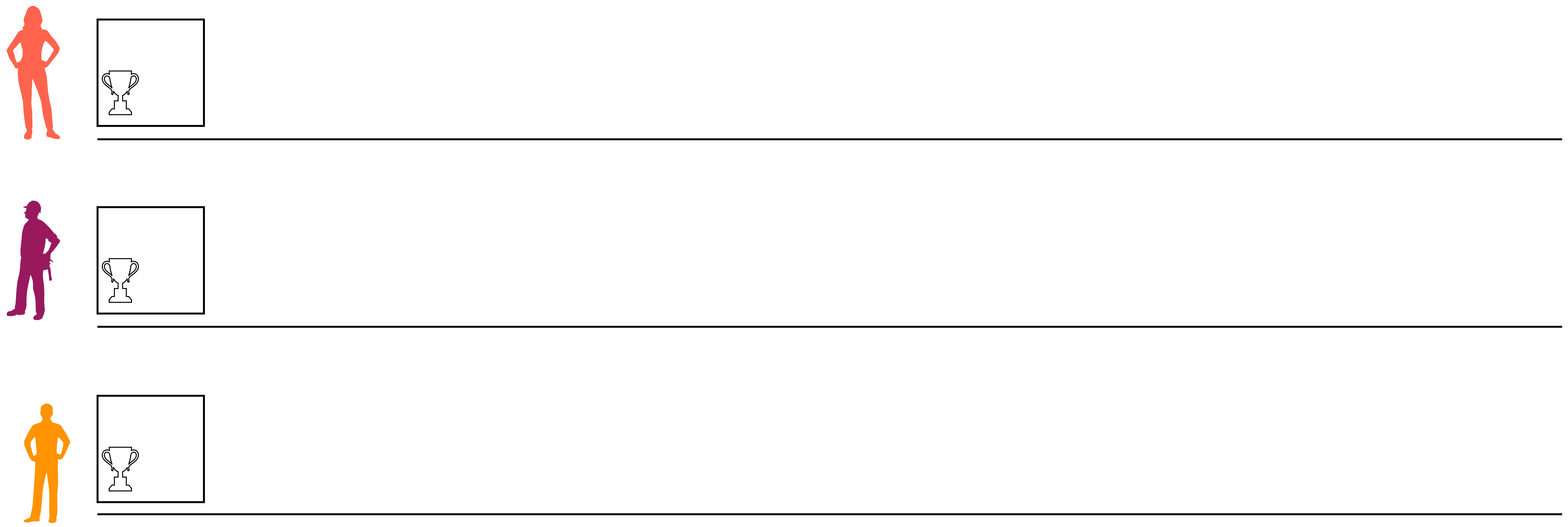
Auction result: 🏆 No winner

🏆 Winner marker

Auction operations:

- Open auction
- Place bid
- Close auction
- Select winner

Auction State Evolution

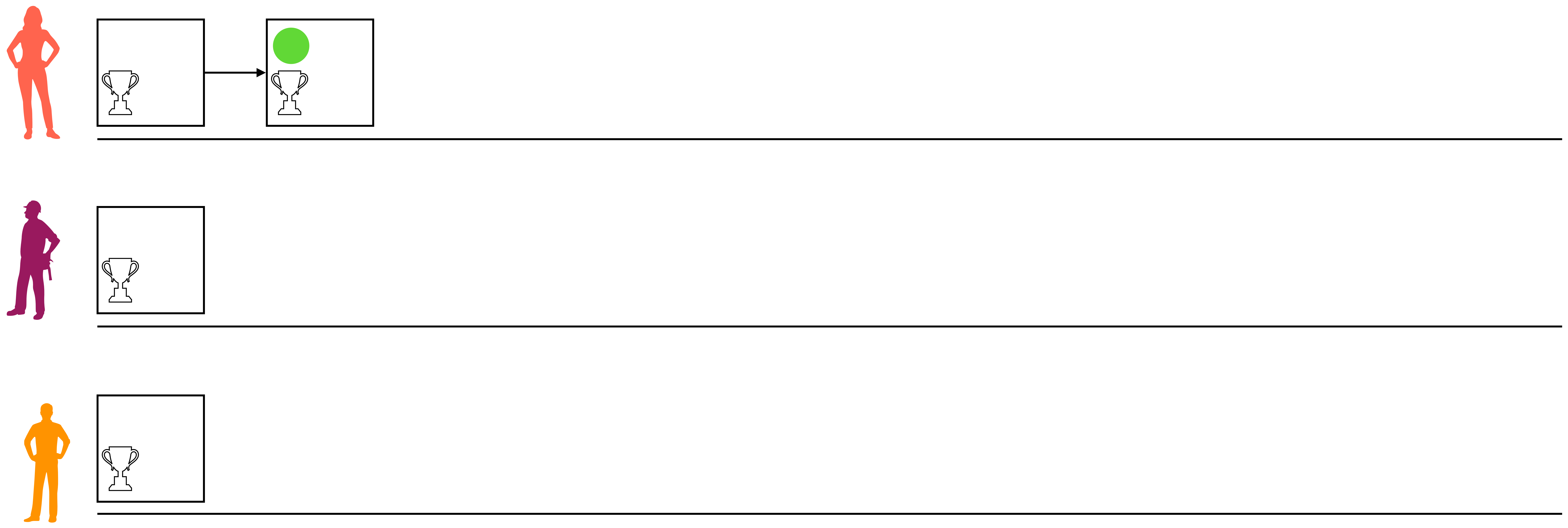


Auction status: ●

Auction Result: 🏆

Auction Bidders: 100

Auction State Evolution

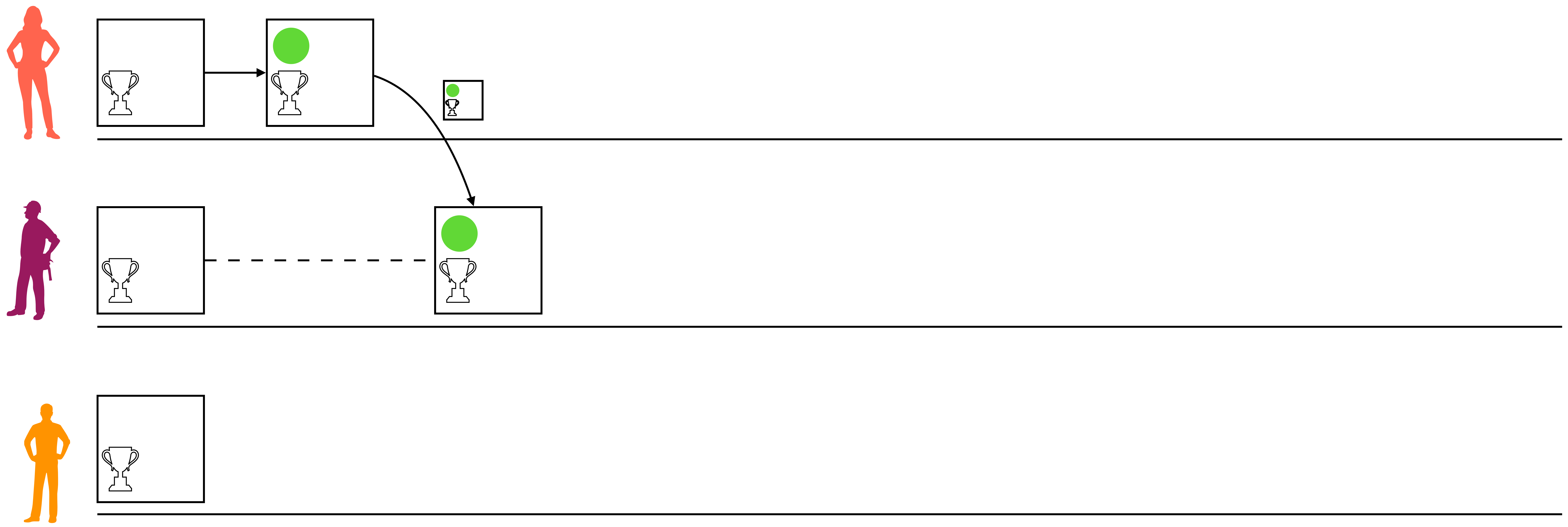


Auction status: ●

Auction Result: 🏆

Auction Bidders: 100

Auction State Evolution

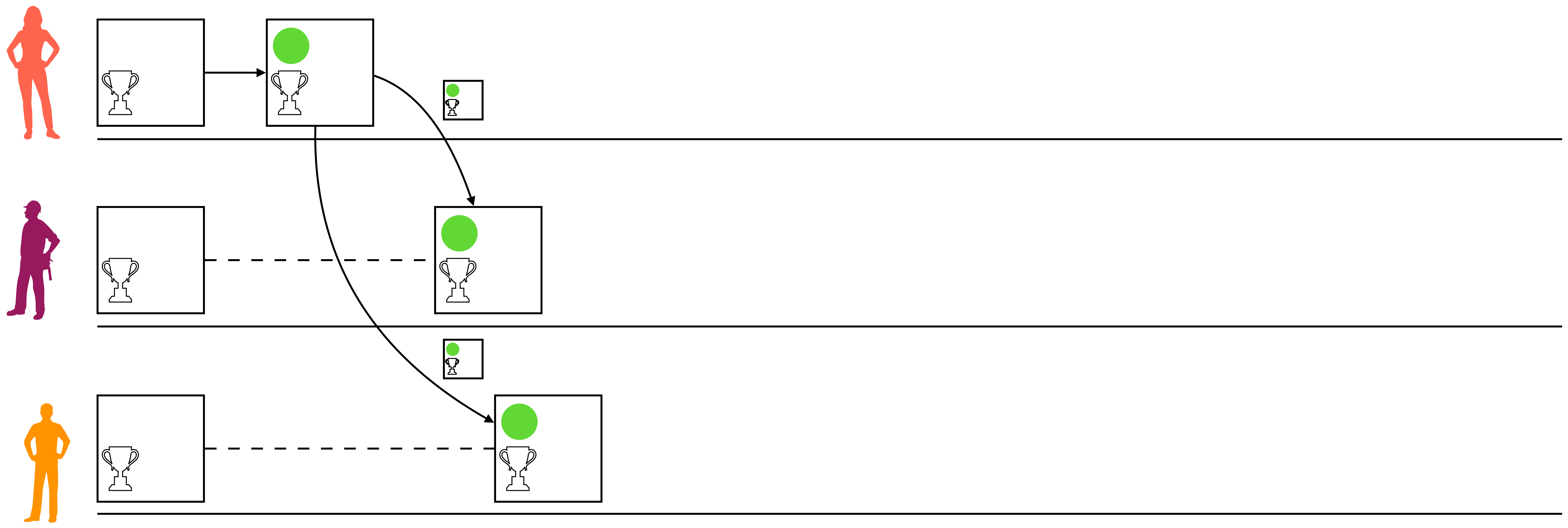


Auction status: ●

Auction Result: 🏆

Auction Bidders: 100

Auction State Evolution

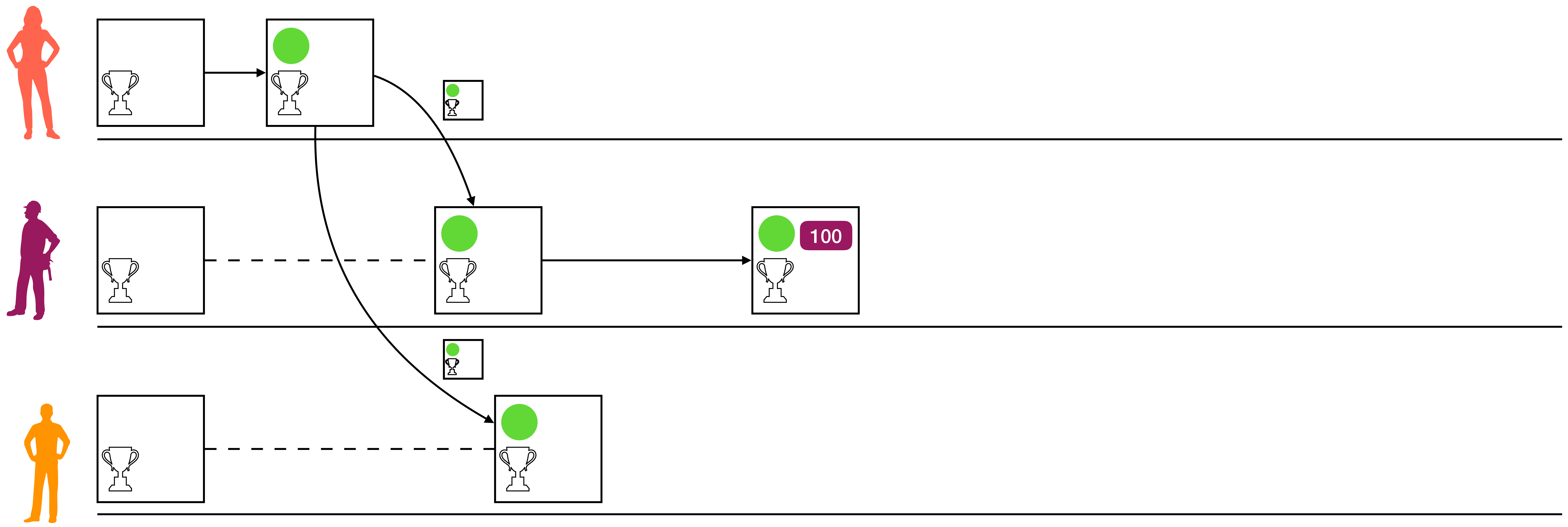


Auction status: ●

Auction Result: 🏆

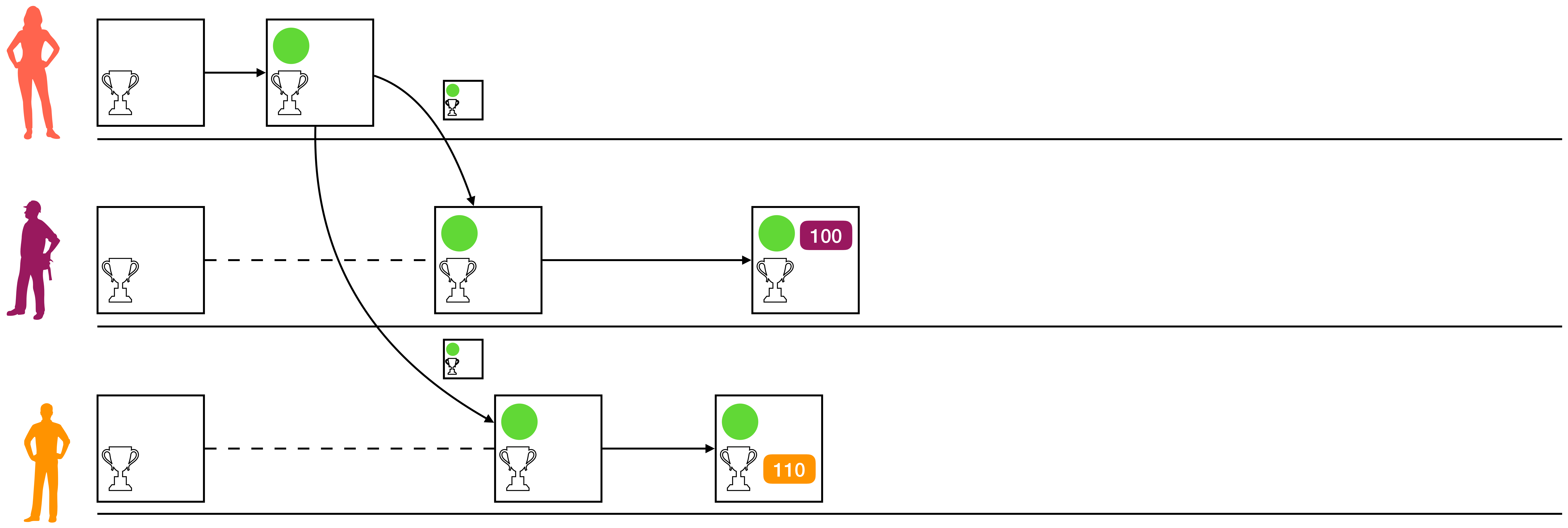
Auction Bidders: 100

Auction State Evolution



Auction status: ● Auction Result: 🏆 Auction Bidders: 100

Auction State Evolution

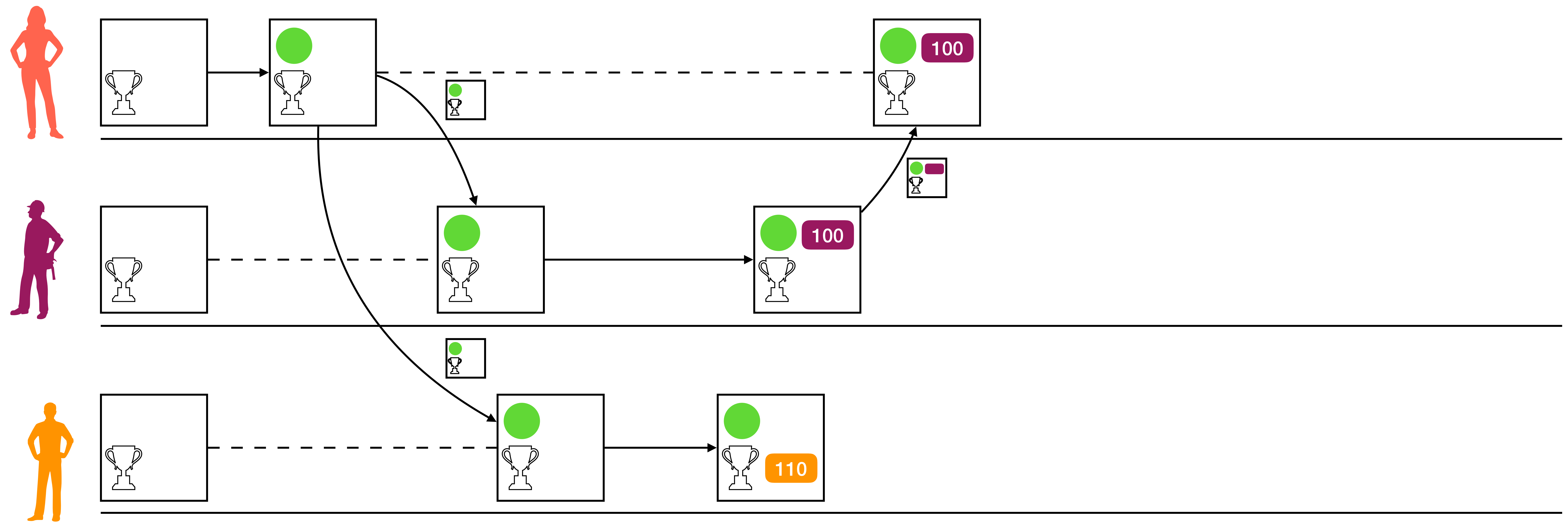


Auction status: ●

Auction Result: 🏆

Auction Bidders: 100

Auction State Evolution

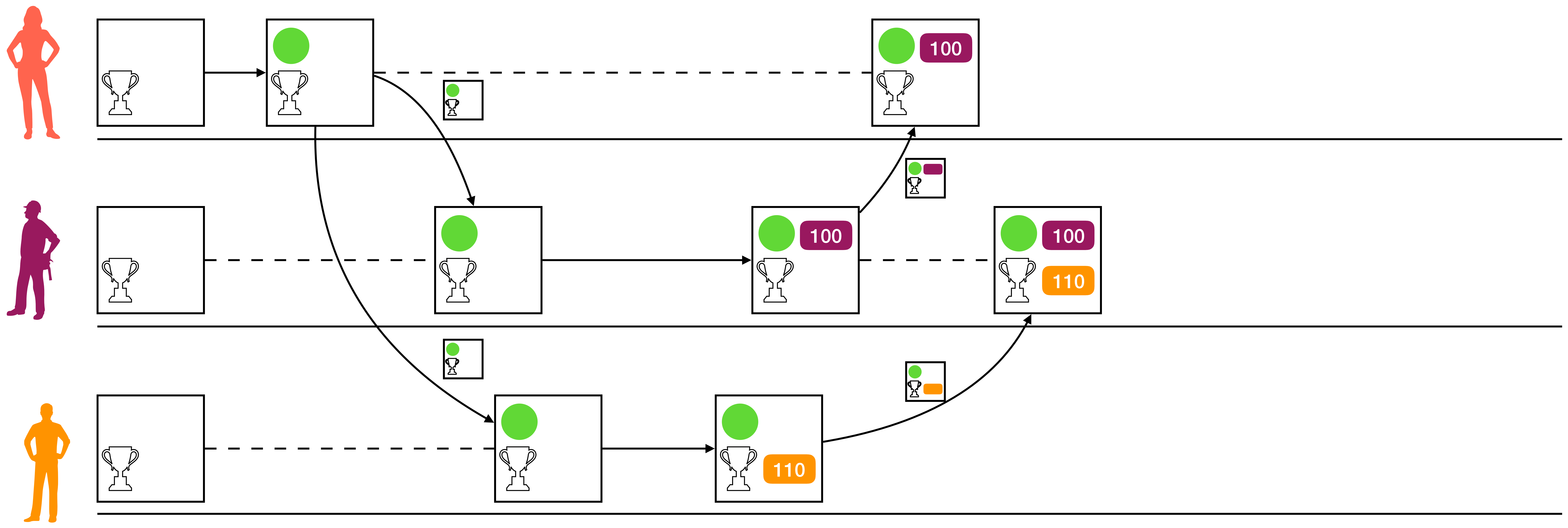


Auction status: ●

Auction Result: 🏆

Auction Bidders: 100

Auction State Evolution

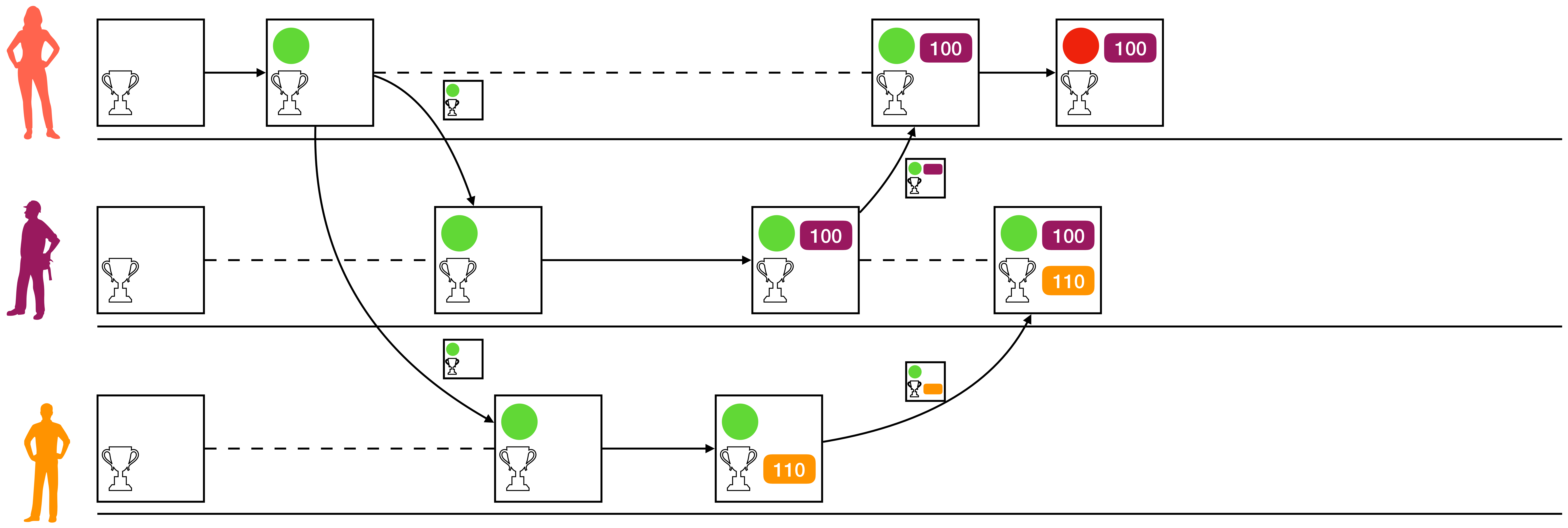


Auction status: ●

Auction Result: 🏆

Auction Bidders: 100

Auction State Evolution

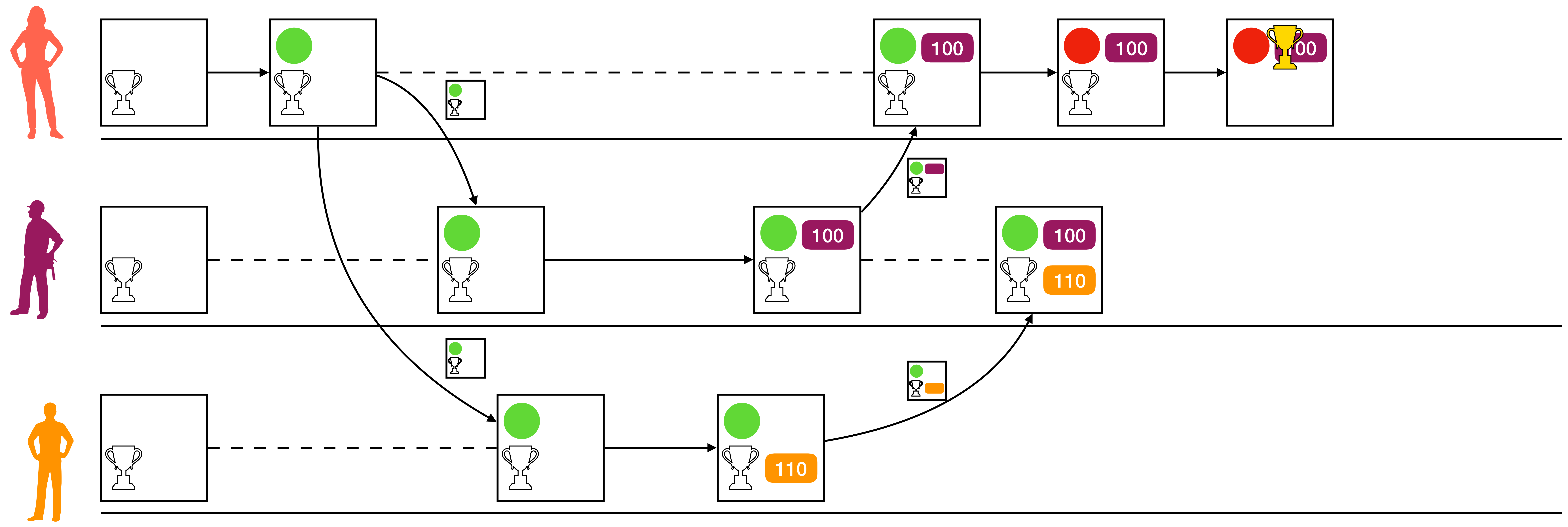


Auction status: ●

Auction Result: 🏆

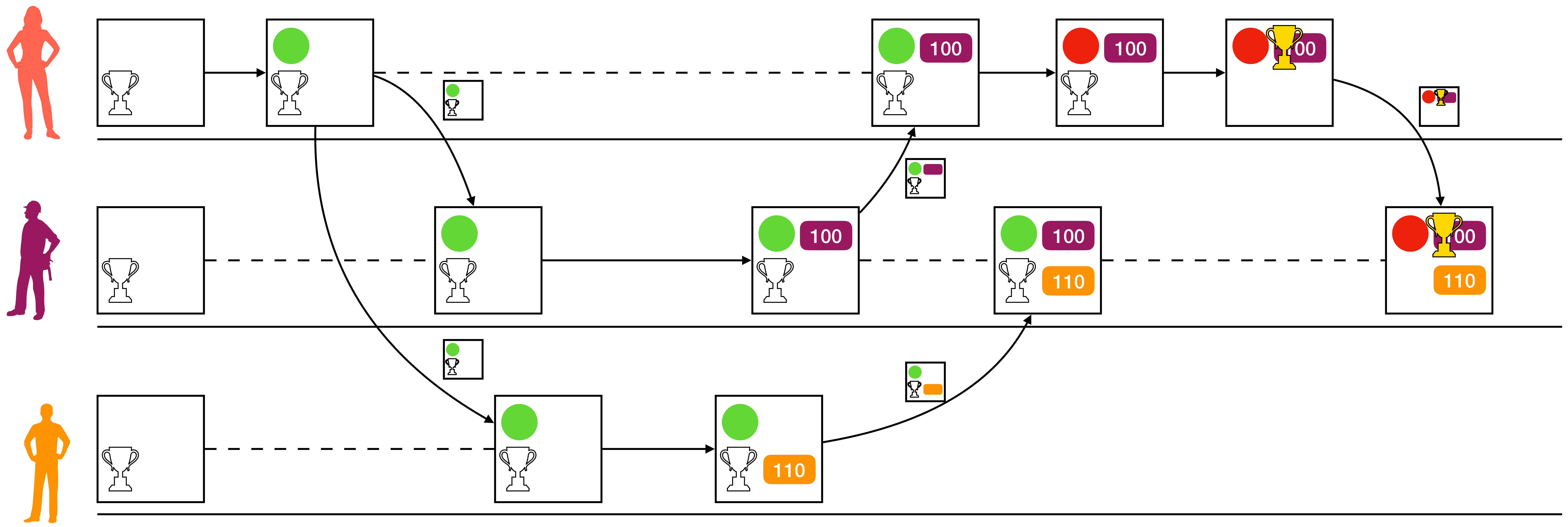
Auction Bidders: 100

Auction State Evolution



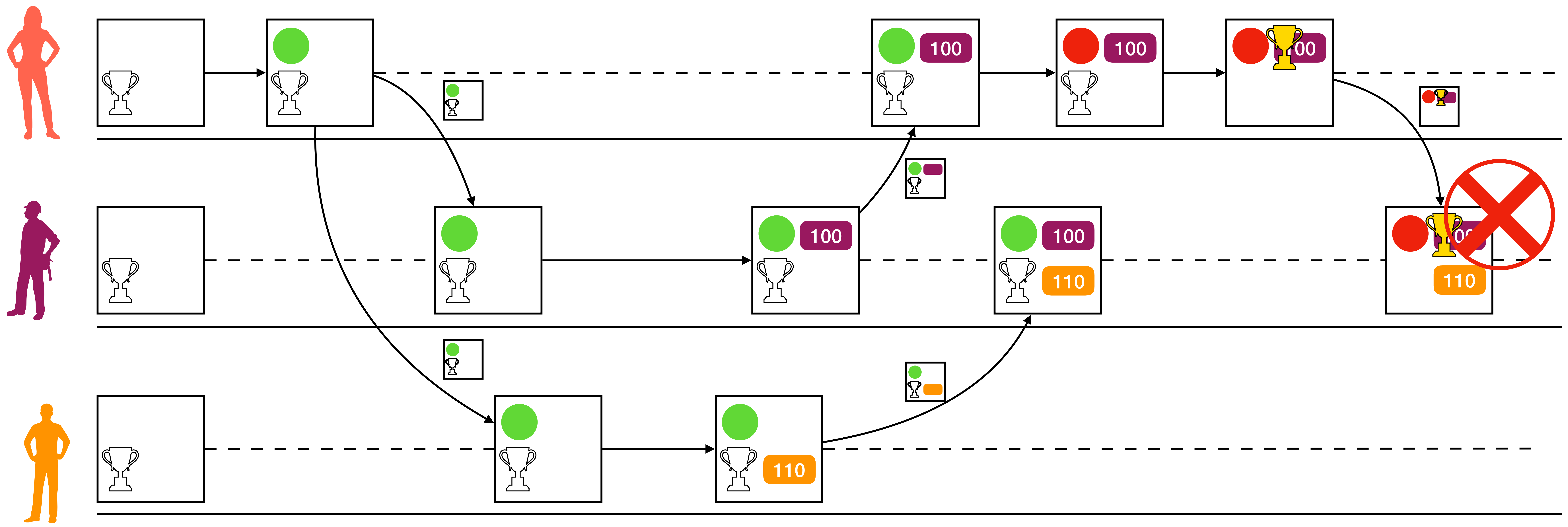
Auction status: ● Auction Result: 🏆 Auction Bidders: 100

Auction State Evolution



Auction status: ●   Auction Result: 🏆   Auction Bidders: 100

AUCTION STATE EVOLUTION

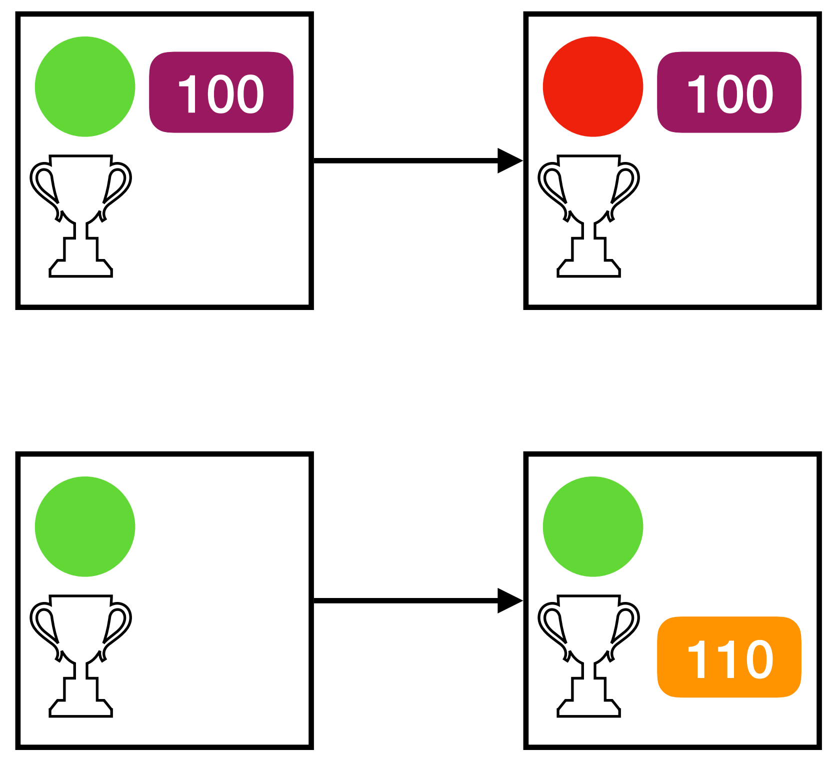


AUCTION STATUS: ●

AUCTION RESULT: 🏆

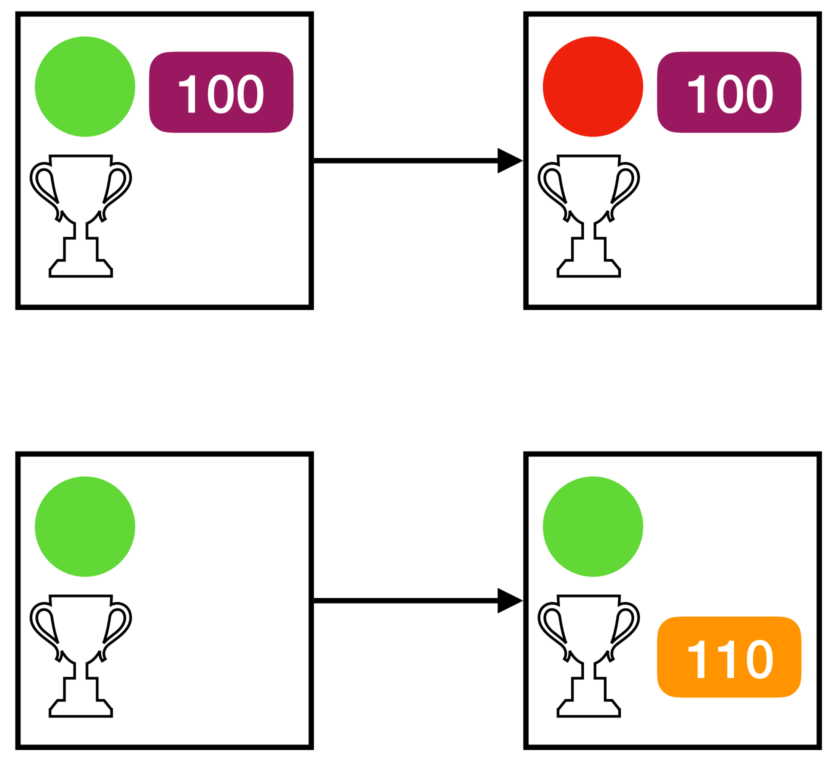
AUCTION BIDDERS: 100

Concurrency Control



Concurrency Control

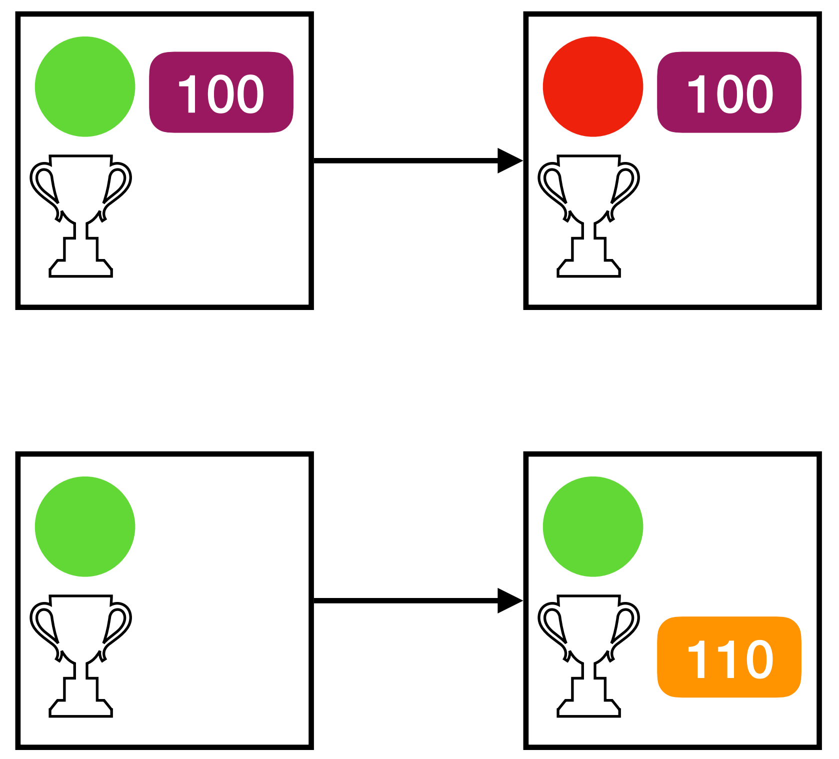
Invariants under Concurrency



CONCURRENCY CONTROL

INVARIANTS UNDER CONCURRENCY

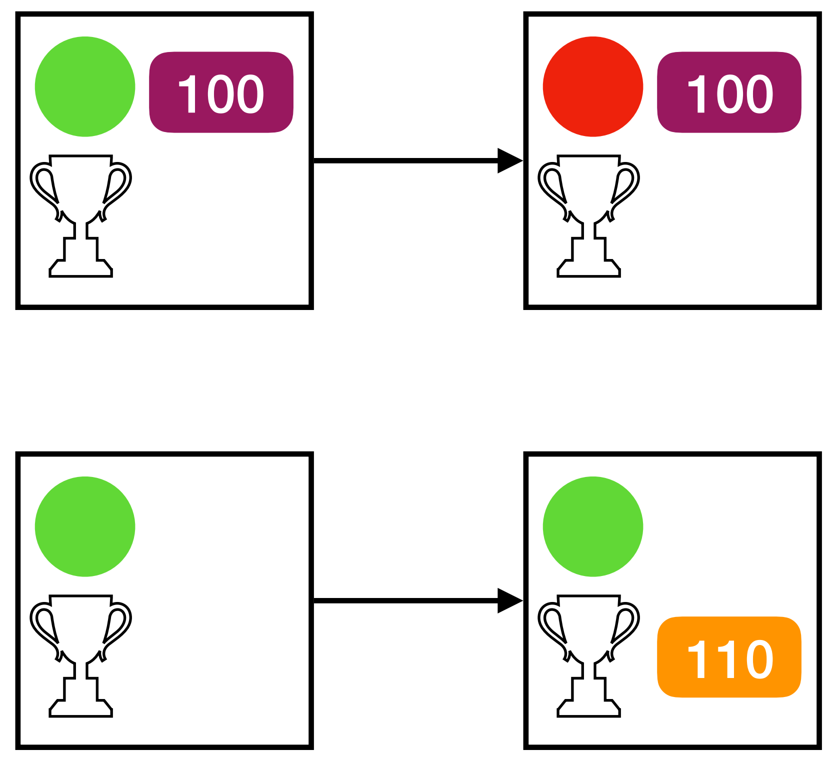
- Auction cannot be closed while bids are being placed



CONCURRENCY CONTROL

INVARIANTS UNDER CONCURRENCY

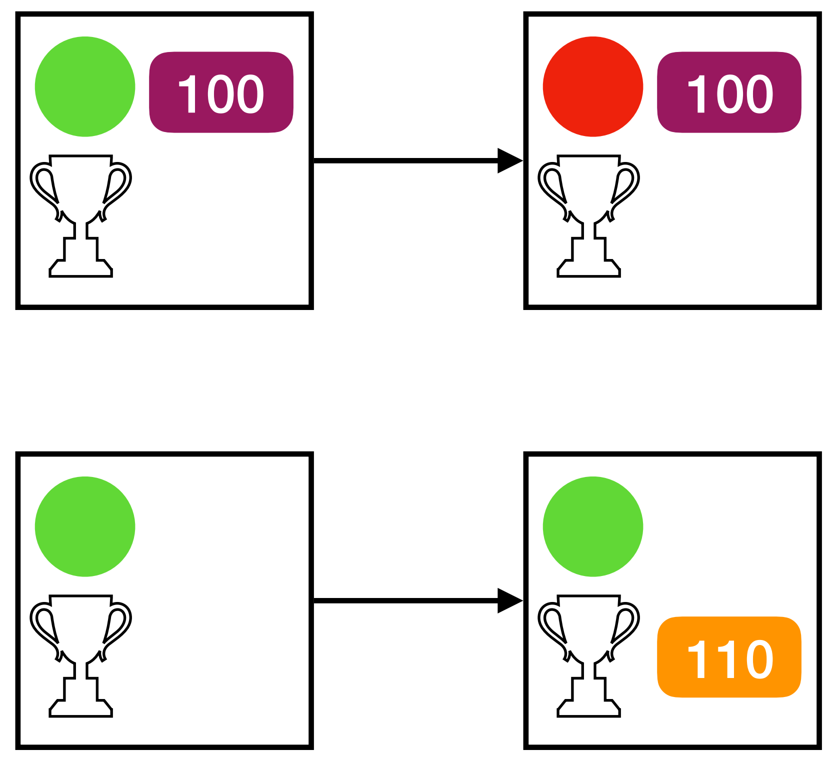
- Auction cannot be closed while bids are being placed
- Status can only go from: `inital` $\longmapsto$ `open` $\longmapsto$ `closed`



Concurrency Control

Invariants under Concurrency

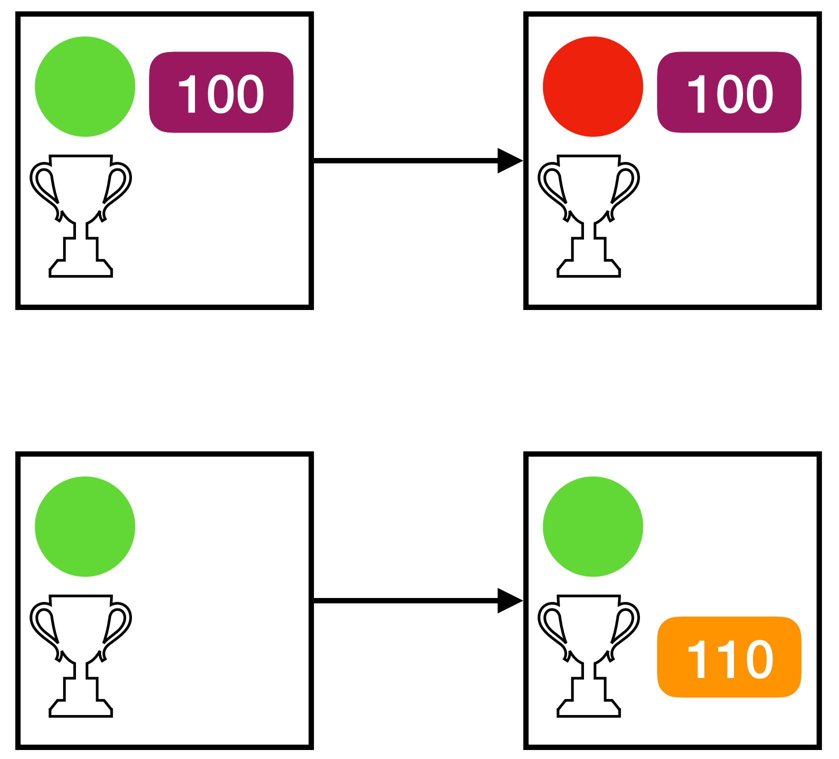
- Auction cannot be closed while bids are being placed
- Status can only go from: `inital` $\longmapsto$ `open` $\longmapsto$ `closed`
- Winner is highest bid when auction is closed



Concurrency Control

Invariants under Concurrency

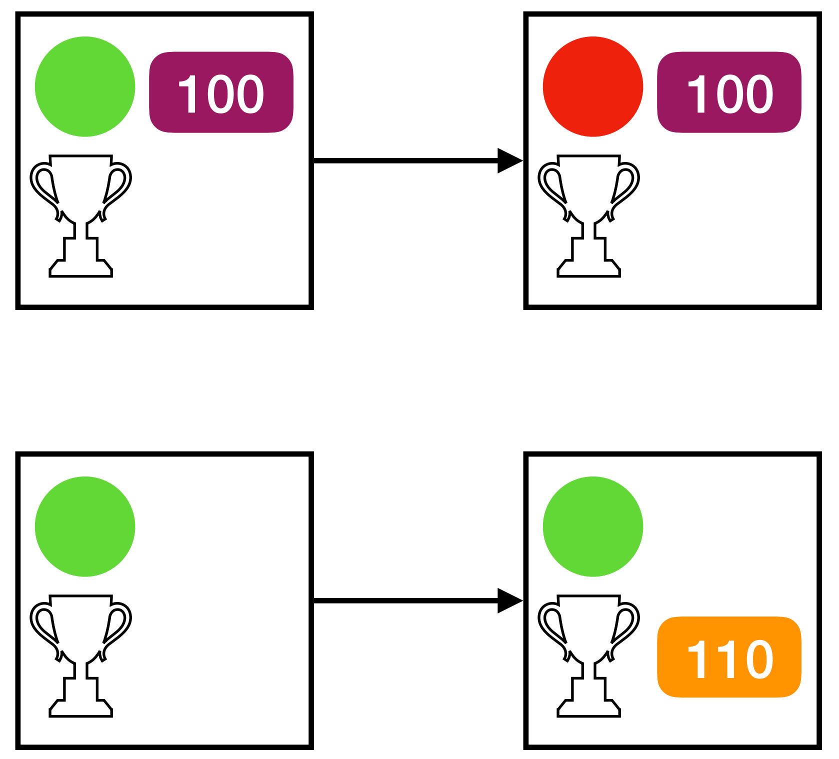
- Auction cannot be closed while bids are being placed
- Status can only go from: `inital` $\longmapsto$ `open` $\longmapsto$ `closed`
- Winner is highest bid when auction is closed
- Bids are strictly incremental



CONCURRENCY CONTROL

INVARIANTS UNDER CONCURRENCY

- Auction cannot be closed while bids are being placed
- Status can only go from: `inital` $\longmapsto$ `open` $\longmapsto$ `closed`
- Winner is highest bid when auction is closed
- Bids are strictly incremental

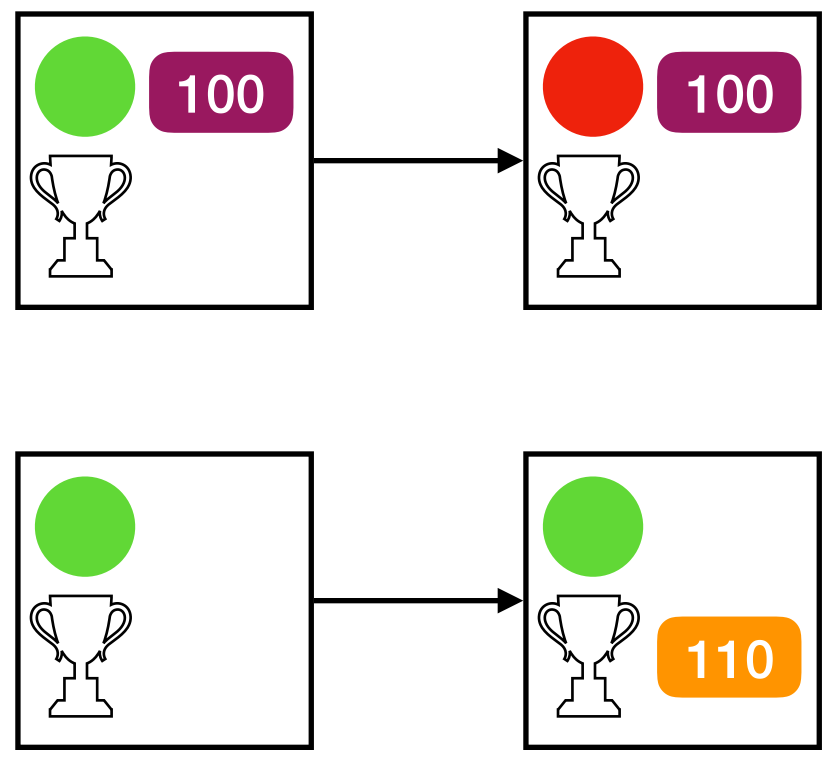


How do we enforce invariants?

Concurrency Control

Invariants under Concurrency

- Auction cannot be closed while bids are being placed
- Status can only go from: `inital` $\longmapsto$ `open` $\longmapsto$ `closed`
- Winner is highest bid when auction is closed
- Bids are strictly incremental

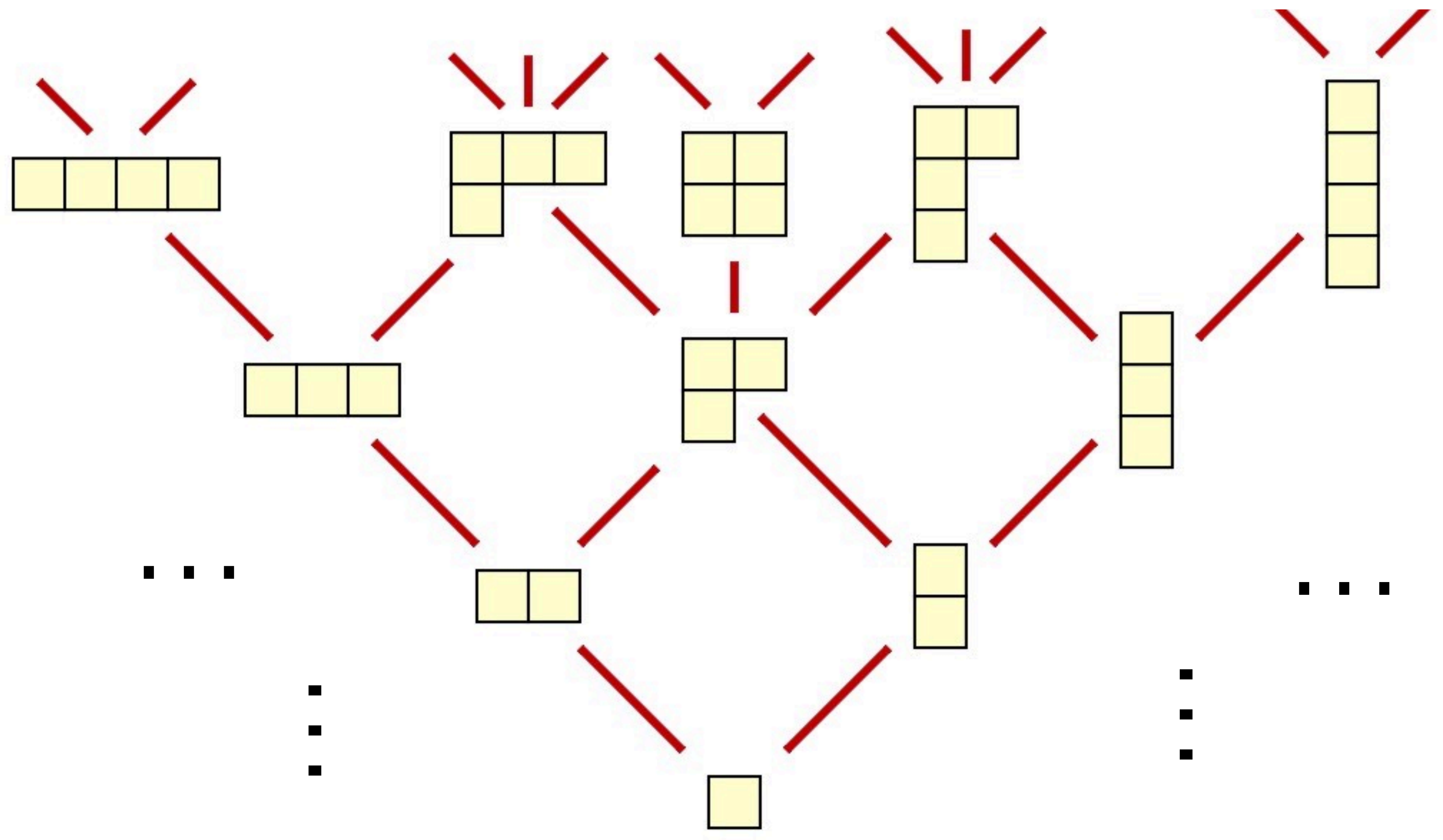


How do we enforce invariants?

How do we verify these invariants?

INVARIANTS FOR SB-CRDTS

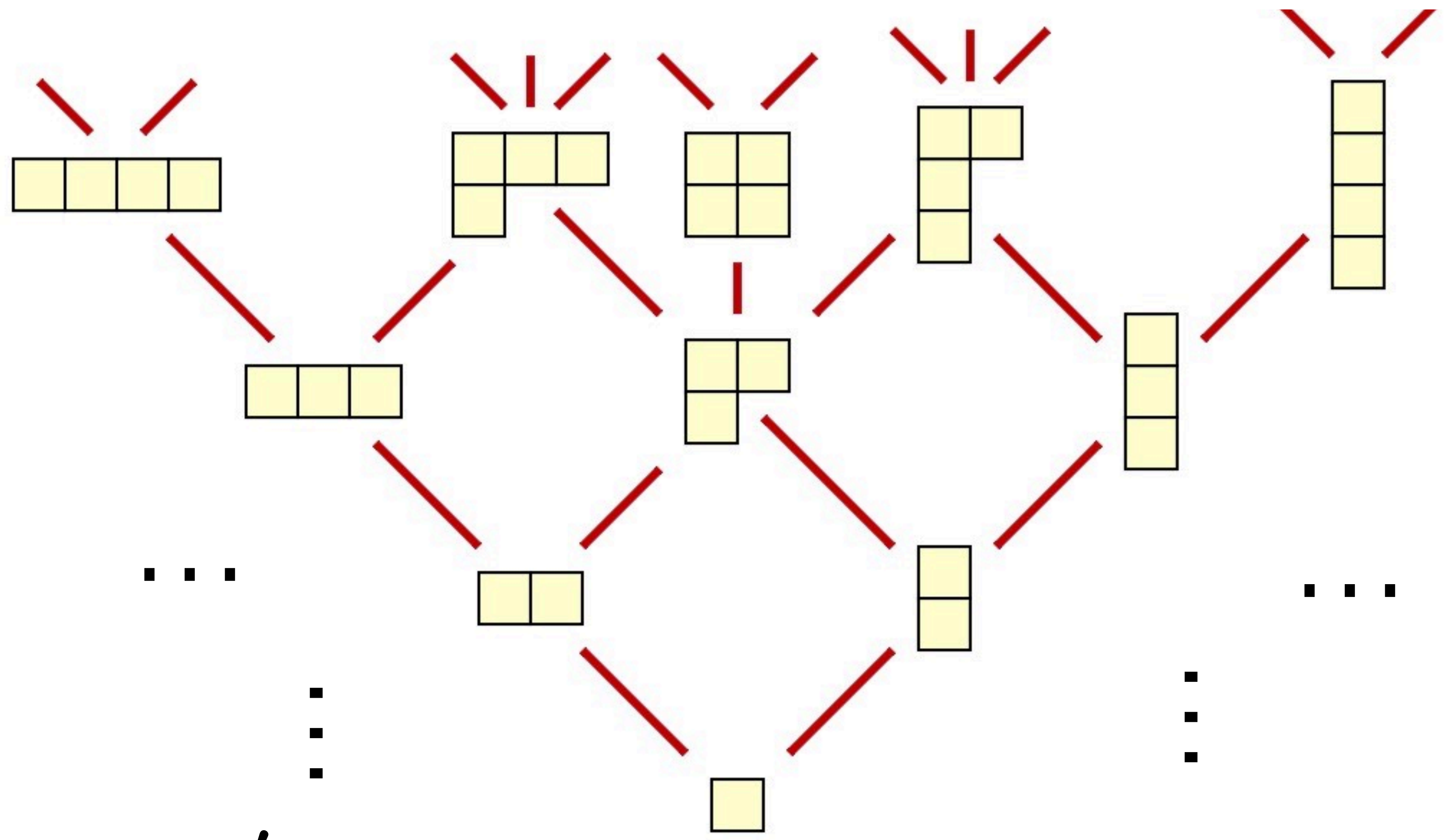
- Invariant constraints



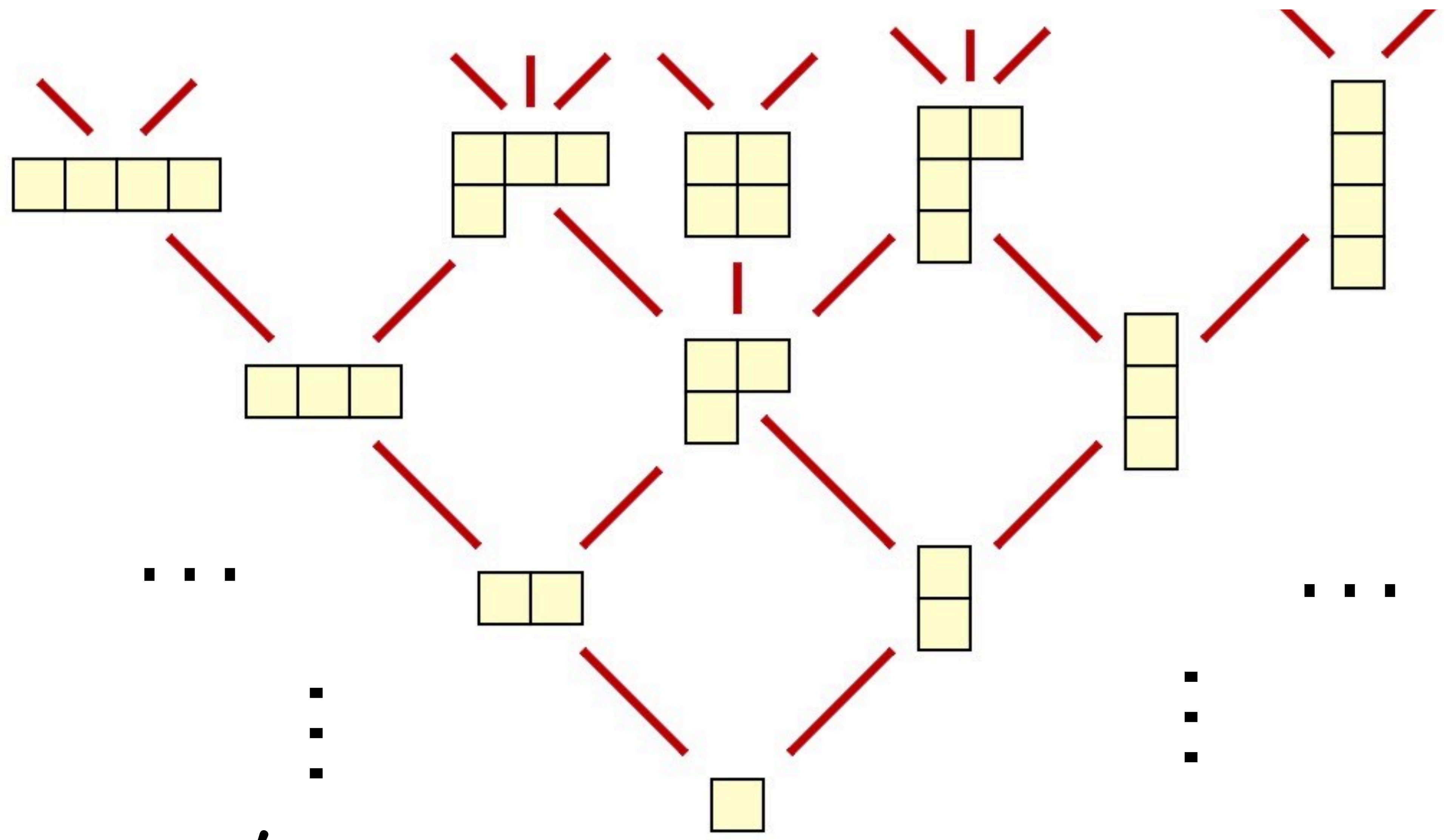
INVARIANTS FOR SB-CRDTS

- Invariant constraints
  - Operations preserve the invariant

$$\forall\ \mathsf{op}, \sigma, \sigma',\ \ \sigma \models \mathsf{Pre}_{\mathsf{op}}\ \wedge\ (\sigma, \sigma') \in [\![\mathsf{op}]\!]\ \Rightarrow \sigma'\ \models\ \mathsf{Inv}$$



INVARIANTS FOR SB-CRDTS



- Invariant constraints
  - Operations preserve the invariant

$$\forall\ \mathsf{op}, \sigma, \sigma',\ \ \sigma \vDash \mathsf{Pre}_{\mathsf{op}}\ \wedge\ (\sigma, \sigma') \in [\![\mathsf{op}]\!]\ \Rightarrow \sigma'\ \vDash\ \mathsf{Inv}$$

  - **`merge`** preserves the invariant

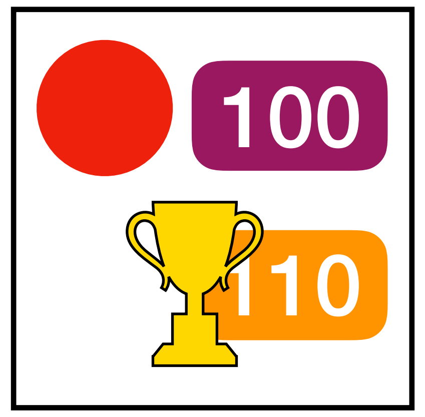
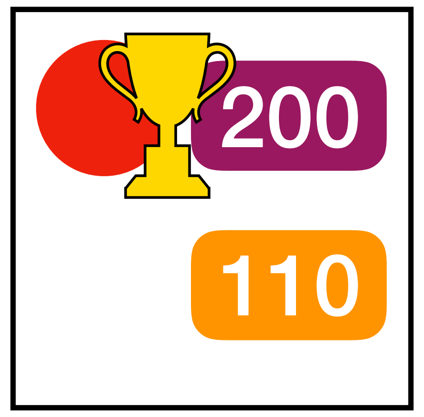
$$\forall\ \sigma, \sigma', \sigma'',\ \sigma\ \vDash\ \mathsf{Inv}\ \wedge \sigma''\ \vDash\ \mathsf{Inv}\ \wedge$$
$$\mathtt{merge}(\sigma, \sigma'') = \sigma'\ \Rightarrow \sigma'\ \vDash\ \mathsf{Inv}$$

Invariants for SB-CRDTs

$$\forall\, \sigma, \sigma', \sigma'',\ \sigma \ \models\ \mathsf{Inv}\ \wedge \sigma''\ \models\ \mathsf{Inv}\ \wedge$$
$$\texttt{merge}(\sigma, \sigma'') = \sigma'\ \Rightarrow \sigma'\ \models\ \mathsf{Inv}$$

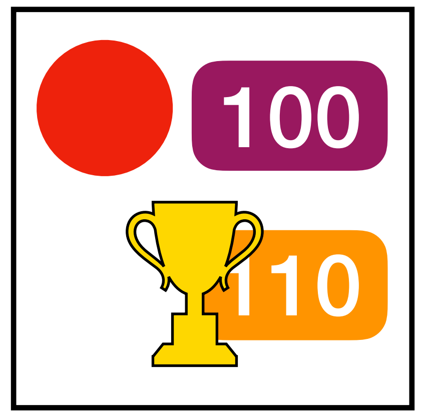
Invariants for SB-CRDTs

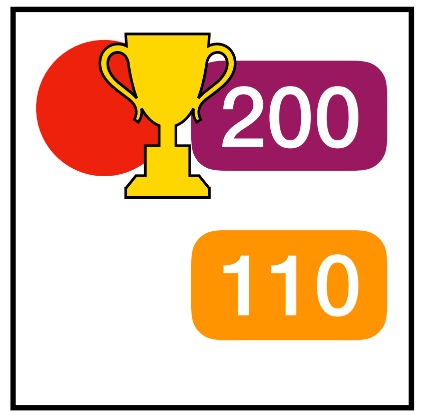
$$\forall\, \sigma, \sigma', \sigma'',\ \sigma \models \mathsf{Inv} \wedge \sigma'' \models \mathsf{Inv} \wedge \texttt{merge}(\sigma, \sigma'') = \sigma' \Rightarrow \sigma' \models \mathsf{Inv}$$

`merge(`  `,`  `)`

Invariants for SB-CRDTs

$$\forall\ \sigma, \sigma', \sigma'',\ \sigma \models \mathsf{Inv} \land \text{[illegible]} \models \mathsf{Inv} \land \texttt{merge}(\sigma, \text{[illegible]}) \text{[illegible]} \sigma' \Rightarrow \sigma' \models \mathsf{Inv}$$

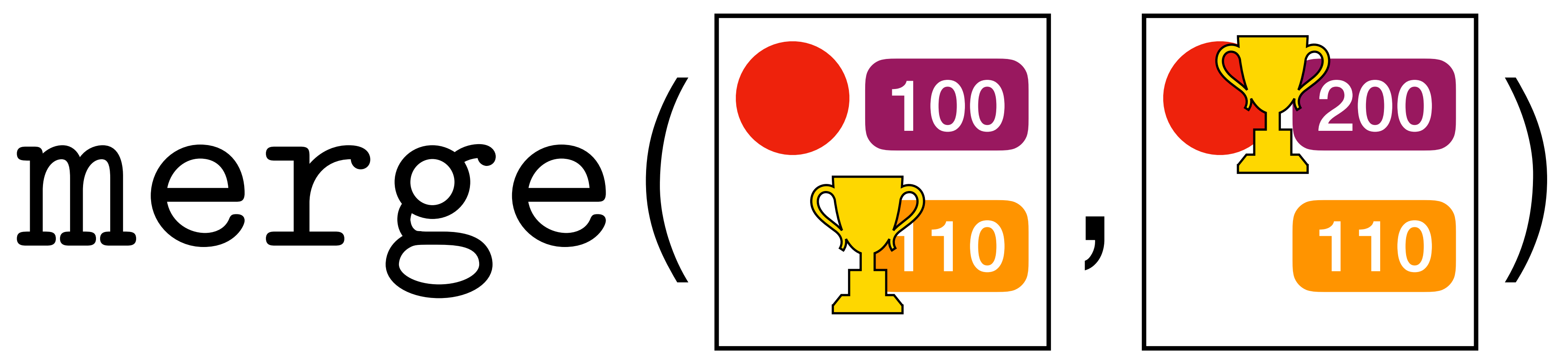
`merge(` 

`,` 

`)`

Invariants for SB-CRDTs

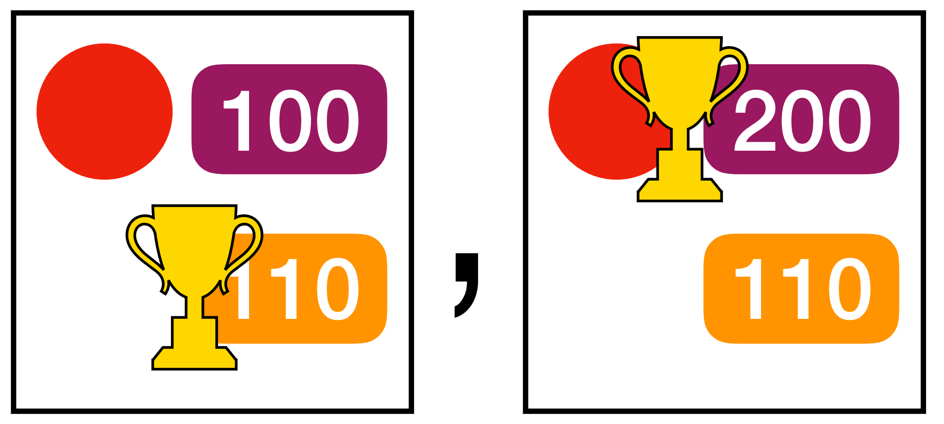
$$\cancel{\forall\, \sigma, \sigma', \sigma'',\ \sigma\ \vDash\ \mathsf{Inv}\ \wedge \ldots \vDash\ \mathsf{Inv}\ \wedge\ \texttt{merge}(\sigma, \ldots) \ldots \sigma'\ \Rightarrow \sigma'\ \vDash\ \mathsf{Inv}}$$



$$\forall\, \sigma, \sigma', \sigma'',\ \sigma\ \vDash\ \mathsf{Inv}\ \wedge \sigma''\ \vDash\ \mathsf{Inv}\ \wedge\ \mathsf{reachable}_{\sigma_i}(\sigma)\ \wedge\ \mathsf{reachable}_{\sigma_i}(\sigma'')\ \wedge$$
$$\texttt{merge}(\sigma, \sigma'') = \sigma'\ \Rightarrow \sigma'\ \vDash\ \mathsf{Inv}$$

Invariants for SB-CRDTs

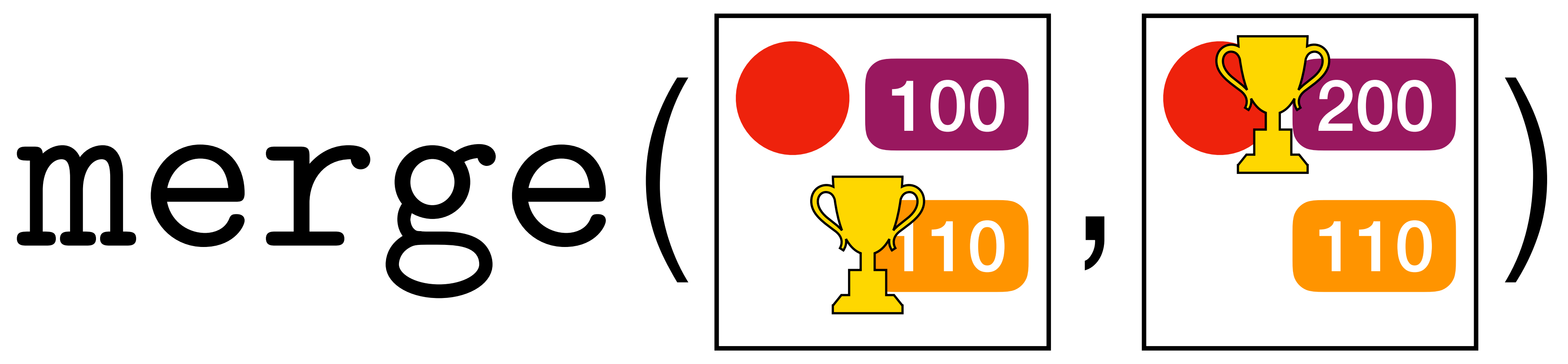
$$\forall\ \sigma, \sigma', \sigma'',\ \sigma\ \models\ \mathsf{Inv}\ \wedge \quad \models\ \mathsf{Inv}\ \wedge$$
$$\texttt{merge}(\sigma, \quad)\quad \sigma'\ \Rightarrow \sigma'\ \models\ \mathsf{Inv}$$

$\texttt{merge}($  $)$

$$\forall\ \sigma, \sigma', \sigma'',\ \sigma\ \models\ \mathsf{Inv}\ \wedge \sigma''\ \models\ \mathsf{Inv}\ \wedge\ \boxed{\mathsf{reachable}_{\sigma_i}(\sigma)\ \wedge\ \mathsf{reachable}_{\sigma_i}(\sigma'')}\ \wedge$$
$$\texttt{merge}(\sigma, \sigma'') = \sigma'\ \Rightarrow \sigma'\ \models\ \mathsf{Inv}$$

Invariants for SB-CRDTs

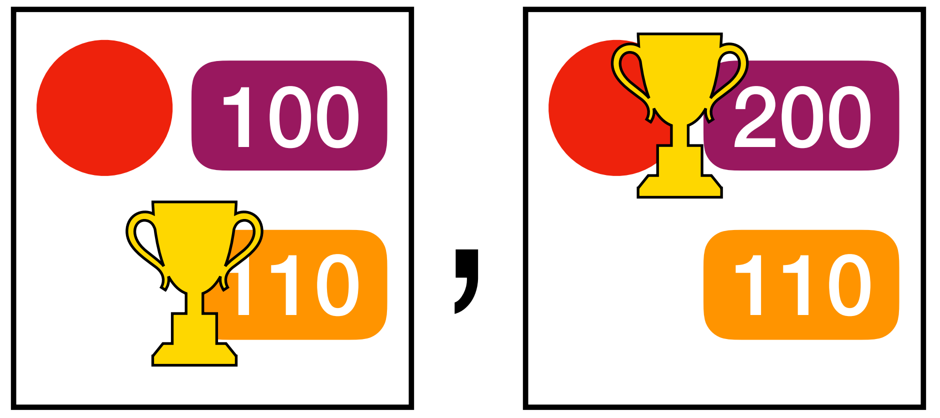
$\forall\ \sigma, \sigma', \sigma'',\ \sigma\ \vDash\ \mathsf{Inv}\ \wedge$ [illegible] $\vDash\ \mathsf{Inv}\ \wedge$
$\mathtt{merge}(\sigma,$ [illegible] $)$ [illegible] $\sigma'\ \Rightarrow \sigma'\ \vDash\ \mathsf{Inv}$



$\forall\ \sigma, \sigma', \sigma'',\ \sigma\ \vDash\ \mathsf{Inv}$ [illegible] $\vDash\ \mathsf{Inv}\ \wedge\ \mathsf{reachable}_{\sigma_i}(\sigma)\ \wedge\ \mathsf{reachable}_{\sigma_i}(\sigma'')\ \wedge$
$\mathtt{merge}(\sigma,$ [illegible] $= \sigma'\ \Rightarrow \sigma'\ \vDash\ \mathsf{Inv}$

INVARIANTS FOR SB-CRDTs

$\forall\ \sigma, \sigma', \sigma'',\ \sigma\ \models\ \mathsf{Inv}$ [illegible] $\models\ \mathsf{Inv}\ \wedge$
$\texttt{merge}(\sigma,$ [illegible] $\sigma'\ \Rightarrow \sigma'\ \models\ \mathsf{Inv}$

$\texttt{merge}($ 

$)$

$\forall\ \sigma, \sigma', \sigma'',\ \sigma\ \models\ \mathsf{Inv}$ [illegible] $\models\ \mathsf{Inv}\ \wedge\ \mathsf{reachable}_{\sigma_i}(\sigma)\ \wedge\ \mathsf{reachable}_{\sigma_i}(\sigma'')\ \wedge$
$\texttt{merge}(\sigma,$ [illegible] $\sigma'\ \Rightarrow \sigma'\ \models\ \mathsf{Inv}$

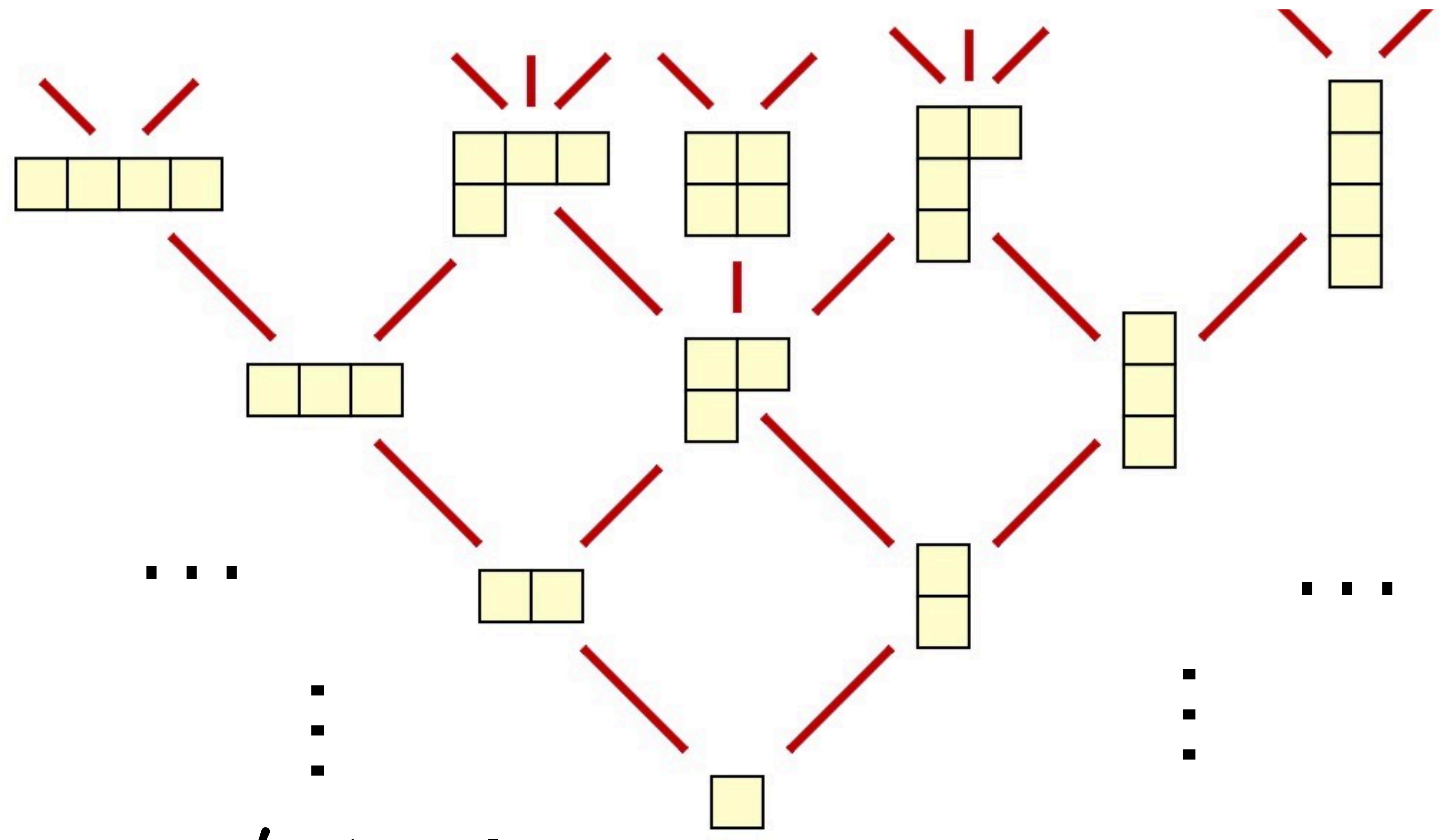
$$\forall\ \sigma, \sigma', \sigma'',\ (\sigma, \sigma'') \models \mathsf{Pre}_{\texttt{merge}}\ \wedge\ \texttt{merge}(\sigma, \sigma'') = \sigma'\ \Rightarrow \sigma'\ \models\ \mathsf{Inv}$$

$$\mathsf{Pre}_{\texttt{merge}} = \mathsf{wp}(\texttt{merge}(\sigma, \sigma''), \mathsf{Inv})$$

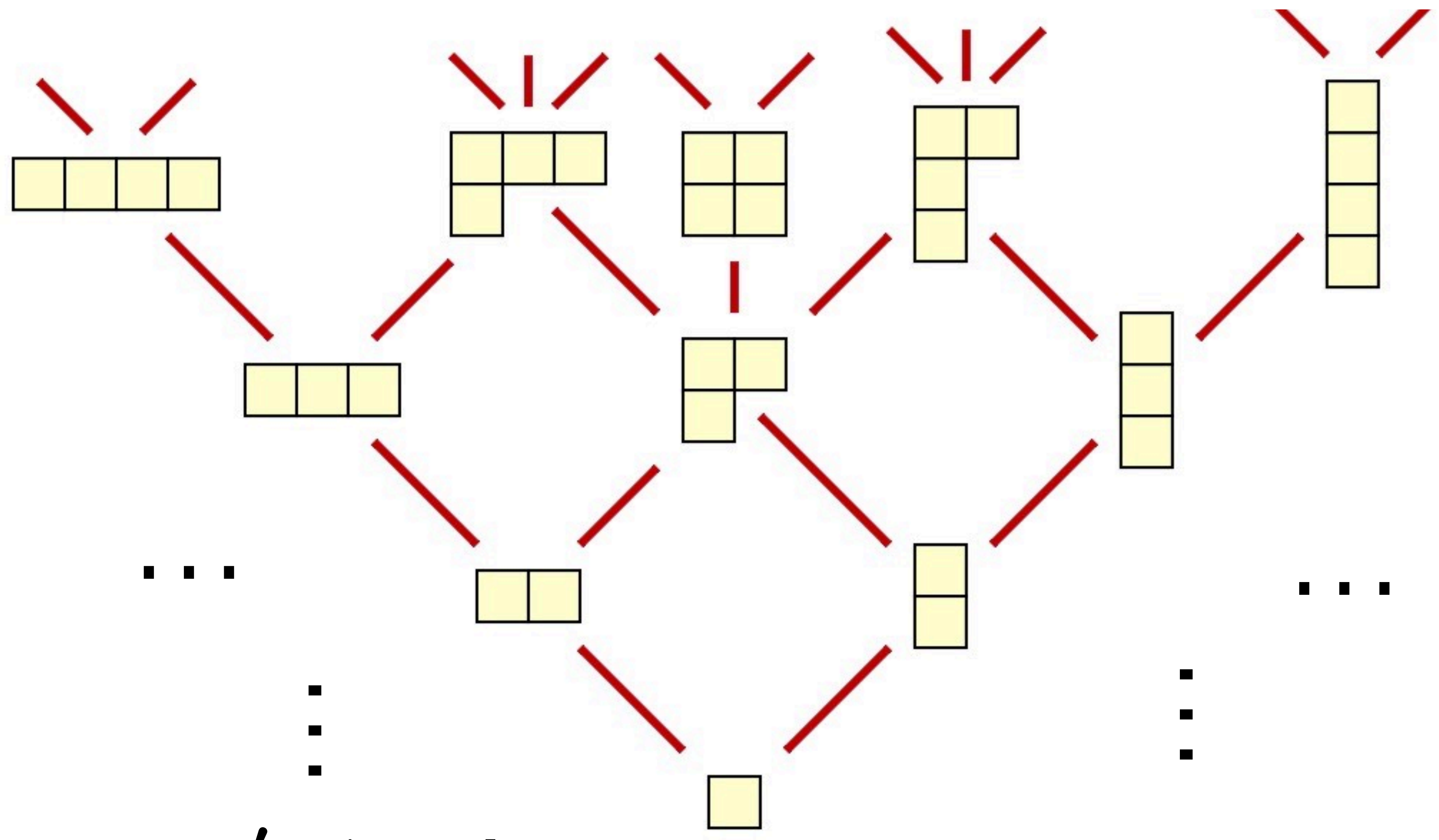
INVARIANTS FOR SB-CRDTS

- Invariant constraints
  - Operations preserve the invariant

$$\forall\ \mathsf{op}, \sigma, \sigma',\ \ \sigma \vDash \mathsf{Pre}_{\mathsf{op}}\ \wedge\ (\sigma, \sigma') \in [\![\mathsf{op}]\!]\ \Rightarrow \sigma'\ \vDash\ \mathsf{Inv}$$



INVARIANTS FOR SB-CRDTS



- Invariant constraints
  - Operations preserve the invariant

$$\forall\ \mathsf{op}, \sigma, \sigma',\ \ \sigma \models \mathsf{Pre}_{\mathsf{op}}\ \ \wedge\ \ (\sigma, \sigma') \in [\![\mathsf{op}]\!]\ \ \Rightarrow \sigma'\ \ \models\ \ \mathsf{Inv}$$

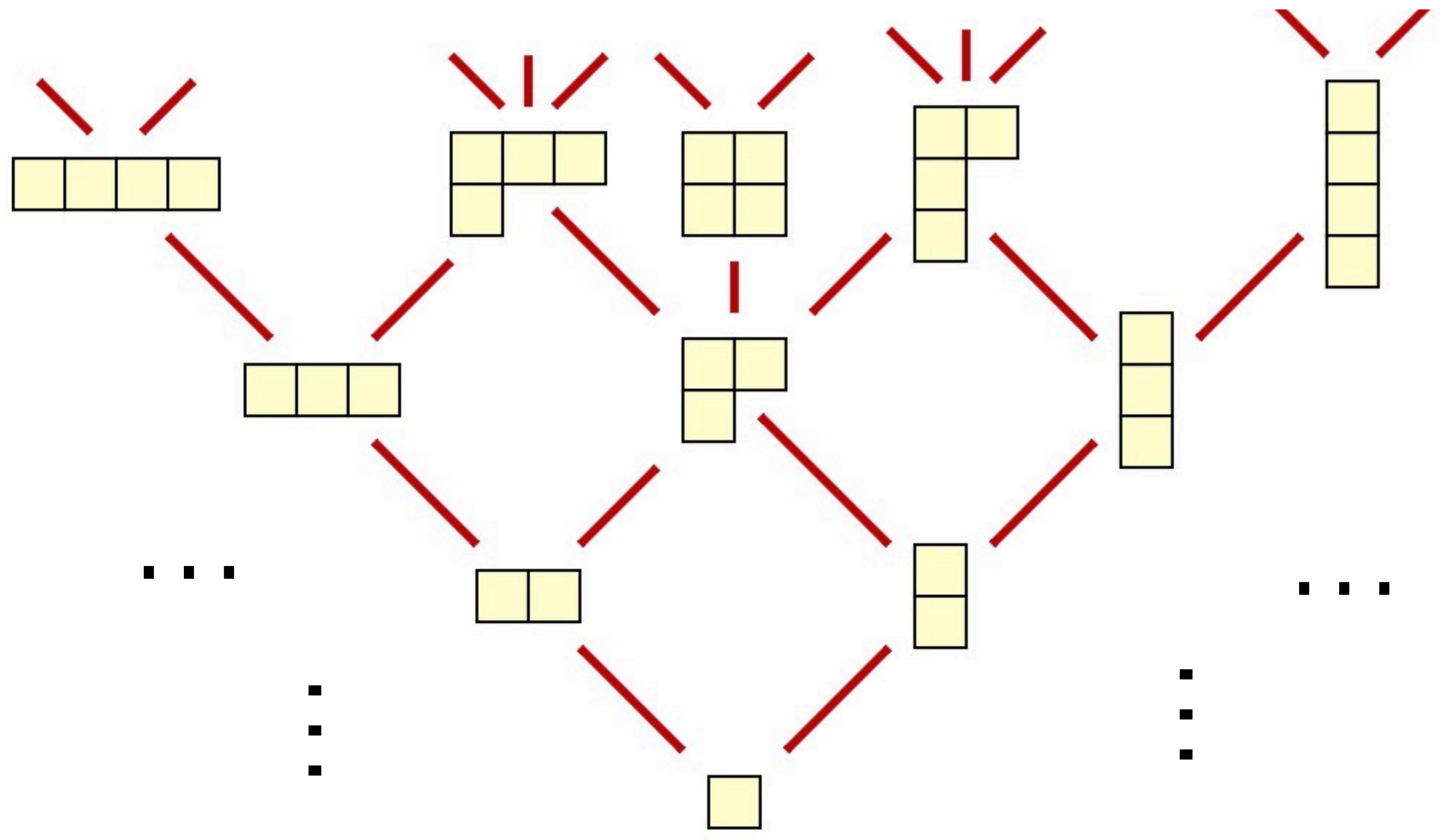
  - **`merge`** preserves the invariant

$$\forall\ \sigma, \sigma', \sigma'',\ \ (\sigma, \sigma'') \models \mathsf{Pre}_{\mathtt{merge}}\ \ \wedge\ \ \mathtt{merge}(\sigma, \sigma'') = \sigma'\ \ \Rightarrow \sigma'\ \ \models\ \ \mathsf{Inv}$$

Invariants for SB-CRDTs

- **`merge`** Pre constraints
  - Initial state satisfies **`merge`** Pre

$$\text{Pre}_{\texttt{merge}}(\sigma_i, \sigma_i)$$



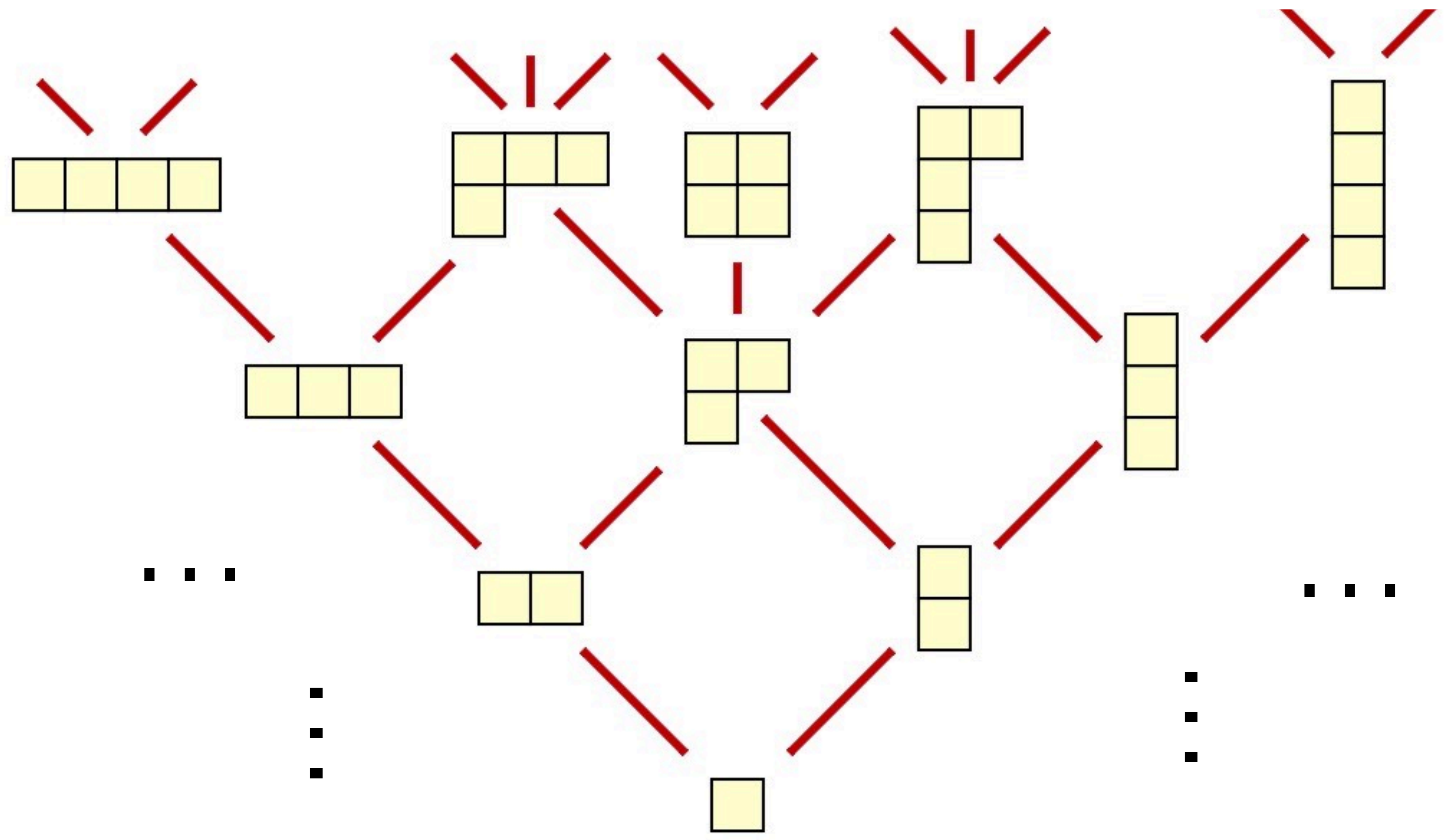
INVARIANTS FOR SB-CRDTS

- **`merge`** Pre constraints
  - Initial state satisfies **`merge`** Pre

    $$\text{Pre}_{\texttt{merge}}(\sigma_i, \sigma_i)$$

  - Operations preserve the **`merge`** Pre

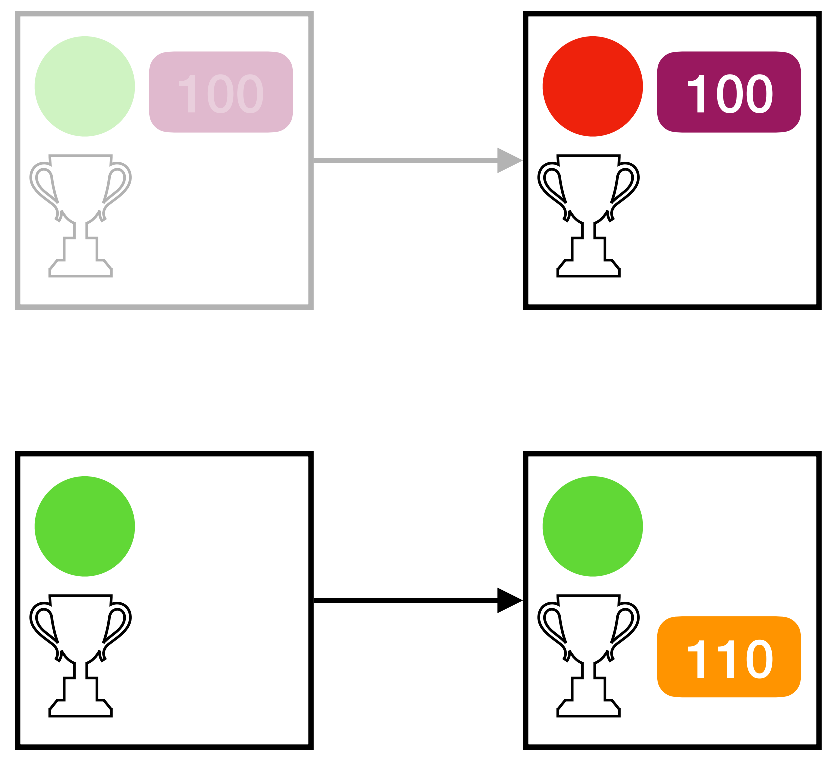
    $$\forall\ \text{op}, \sigma, \sigma', \sigma'',\ \begin{pmatrix} \sigma \vDash \text{Pre}_{\text{op}}\ \wedge \\ (\sigma, \sigma'') \vDash \text{Pre}_{\texttt{merge}}\ \wedge \\ (\sigma, \sigma') \in [\![\text{op}]\!] \end{pmatrix} \Rightarrow (\sigma', \sigma'')\ \vDash\ \text{Pre}_{\texttt{merge}}$$



Concurrency Control

Invariants to avoid Concurrency problems

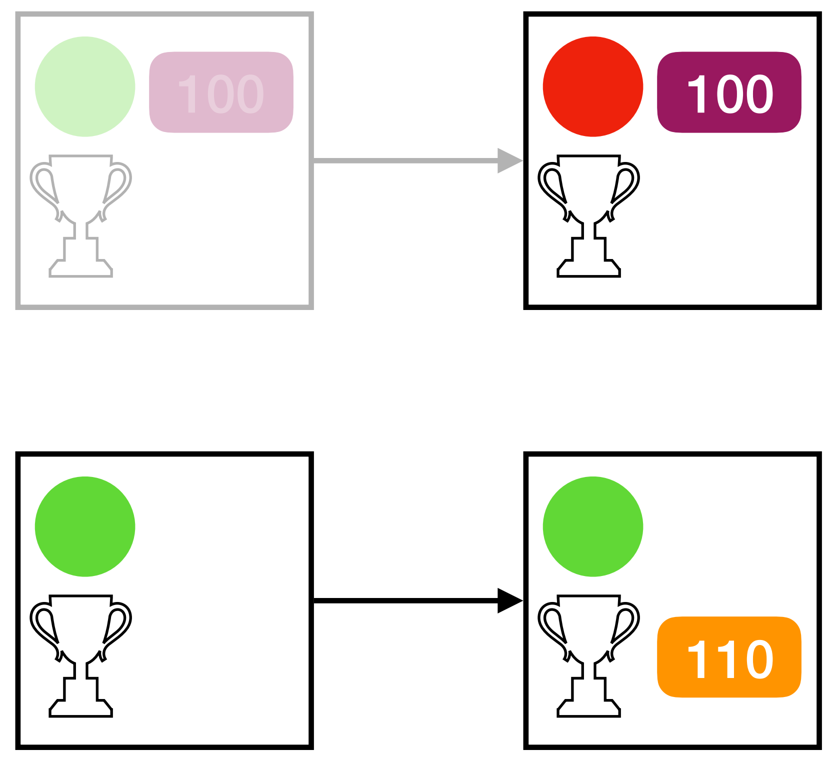
- Auction cannot be closed while bids are being placed
- Status can only go from: `inital` $\longmapsto$ `open` $\longmapsto$ `closed`
- Winner is highest bid when auction is closed
- Bids are strictly incremental



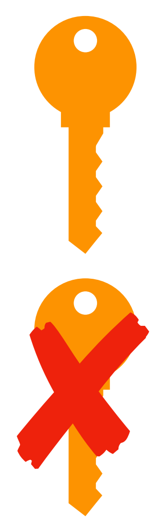
Concurrency Control

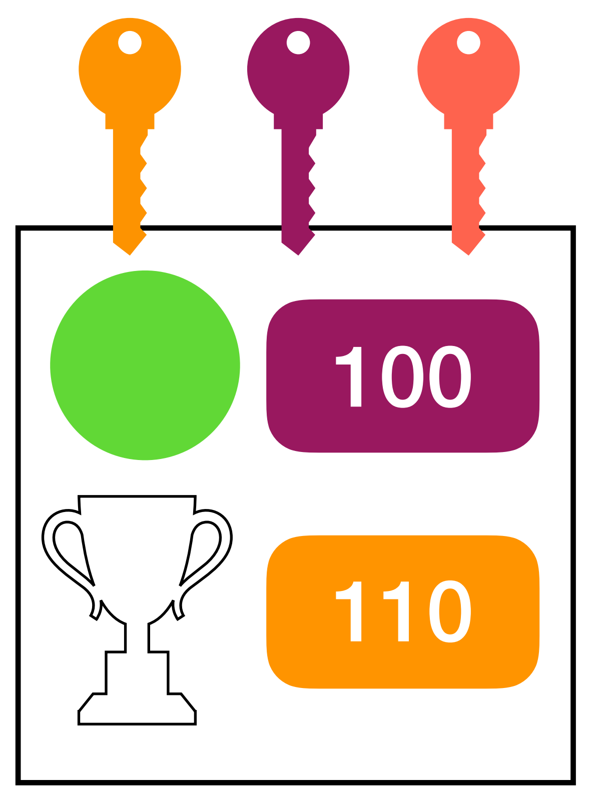
Invariants to avoid Concurrency problems

- Auction cannot be closed while bids are being placed
- Status can only go from: `inital` $\longmapsto$ `open` $\longmapsto$ `closed`
- Winner is highest bid when auction is closed
- Bids are strictly incremental



Tokens for CC:

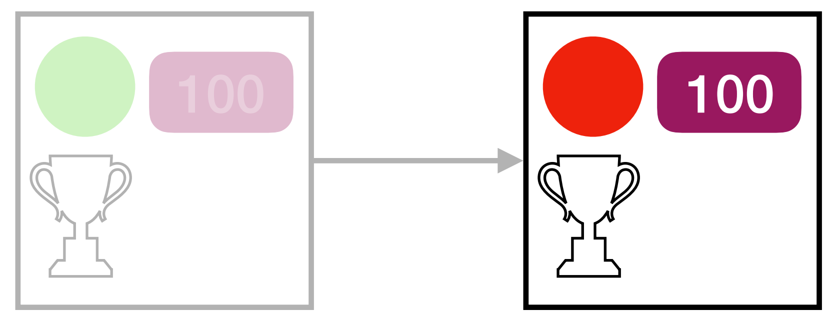


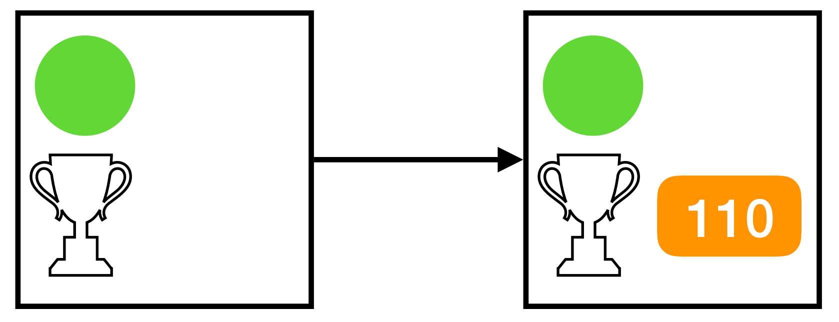


Concurrency Control

Invariants to avoid Concurrency problems

- Auction cannot be closed while bids are being placed
- Status can only go from: `inital` $\longmapsto$ `open` $\longmapsto$ `closed`
- Winner is highest bid when auction is closed
- Bids are strictly incremental

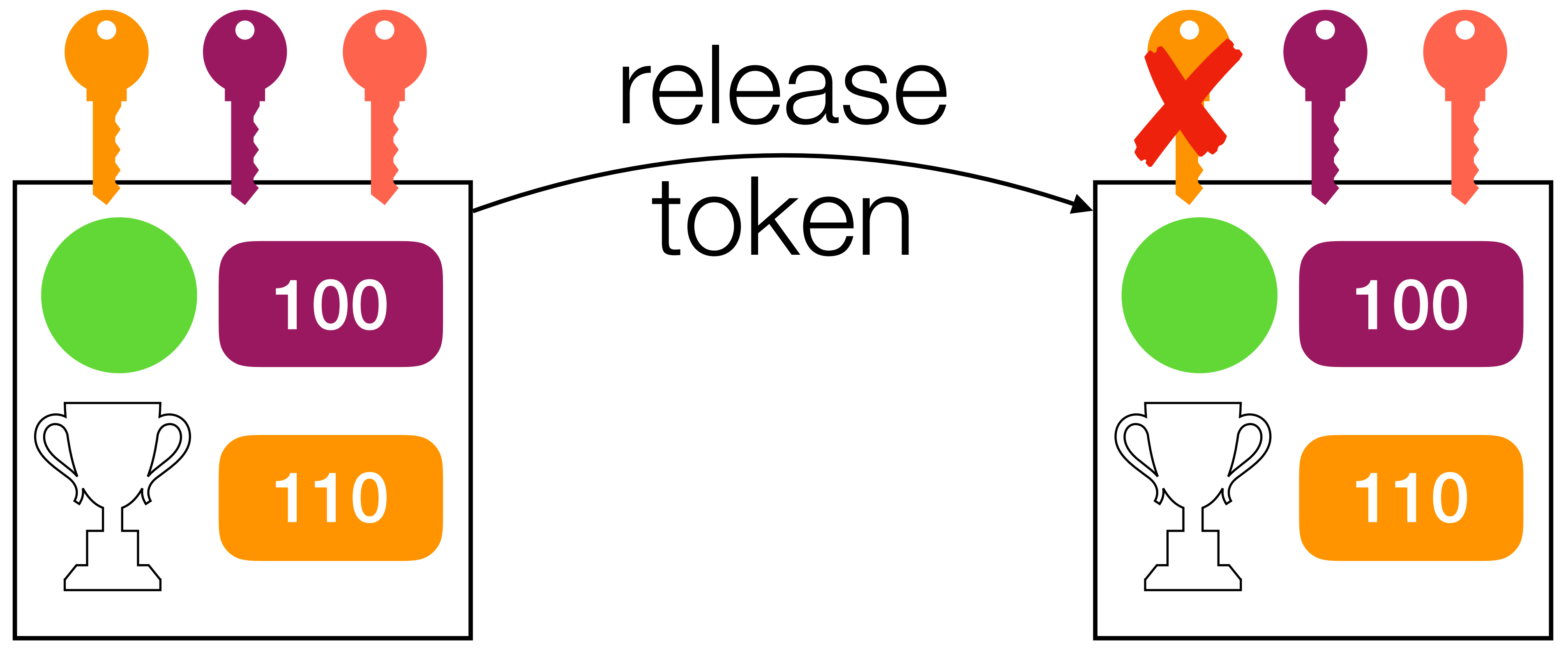




Tokens for CC:

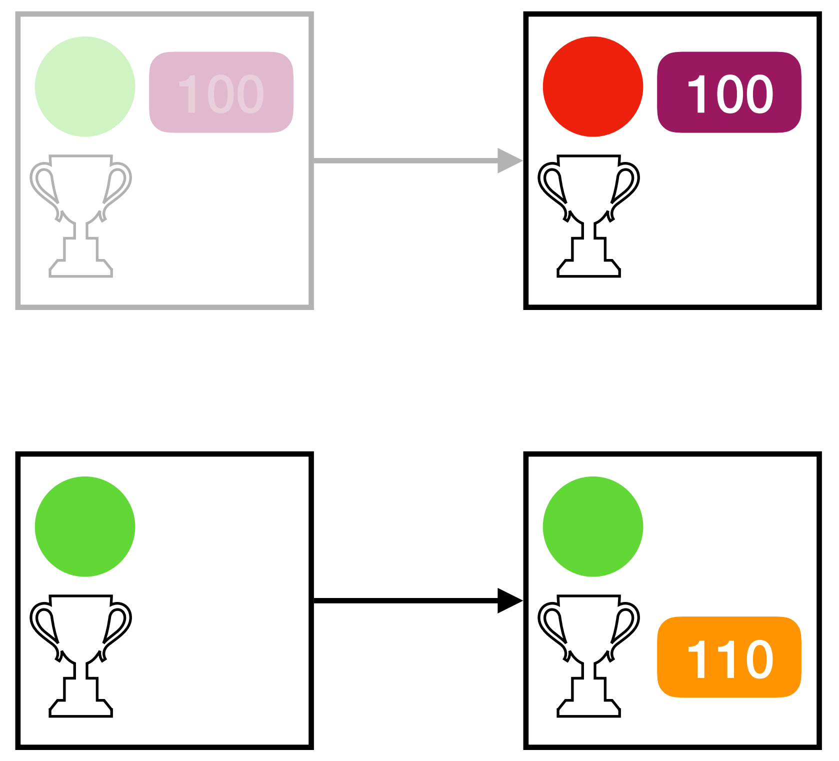
Taken

Released



Concurrency Control

Invariants to avoid Concurrency problems

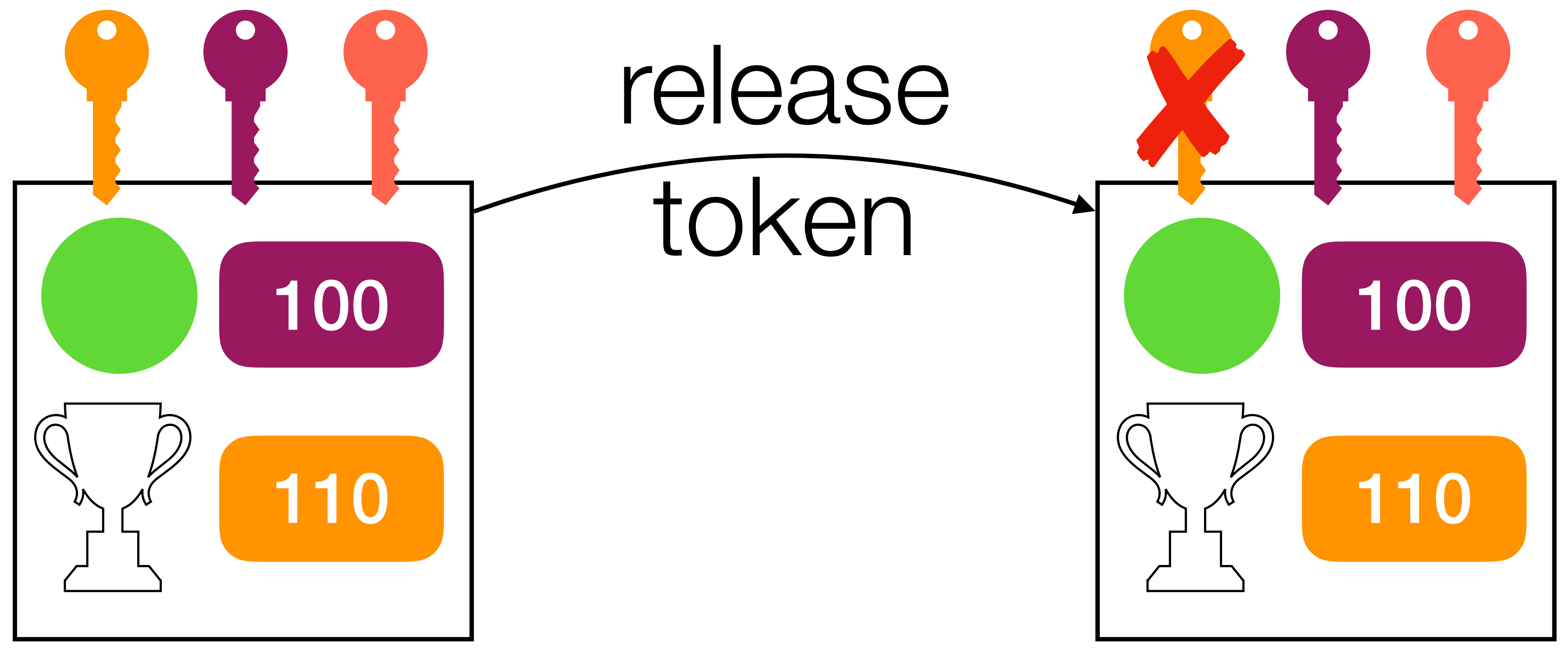


- Auction cannot be closed while bids are being placed
- Status can only go from: `inital` $\longmapsto$ `open` $\longmapsto$ `closed`
- Winner is highest bid when auction is closed
- Bids are strictly incremental
- Tokes go from `taken` $\longmapsto$ `released` (by owner)

Tokens for CC:

Taken

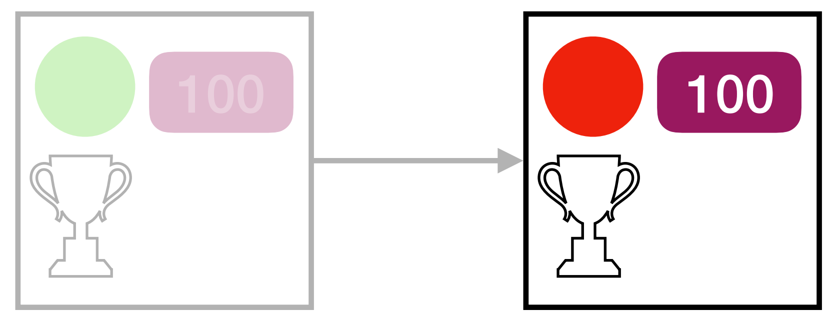
Released

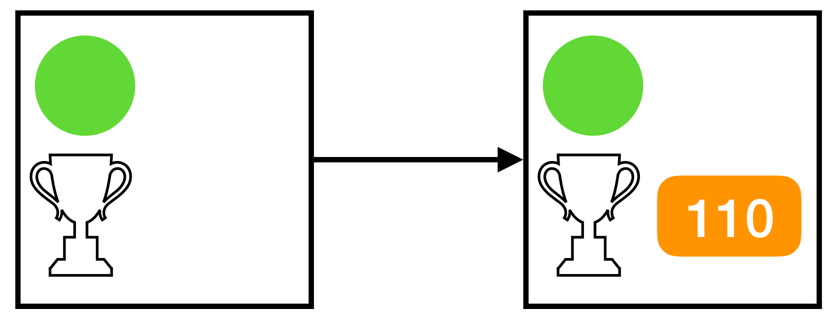


Concurrency Control

Invariants to avoid Concurrency problems

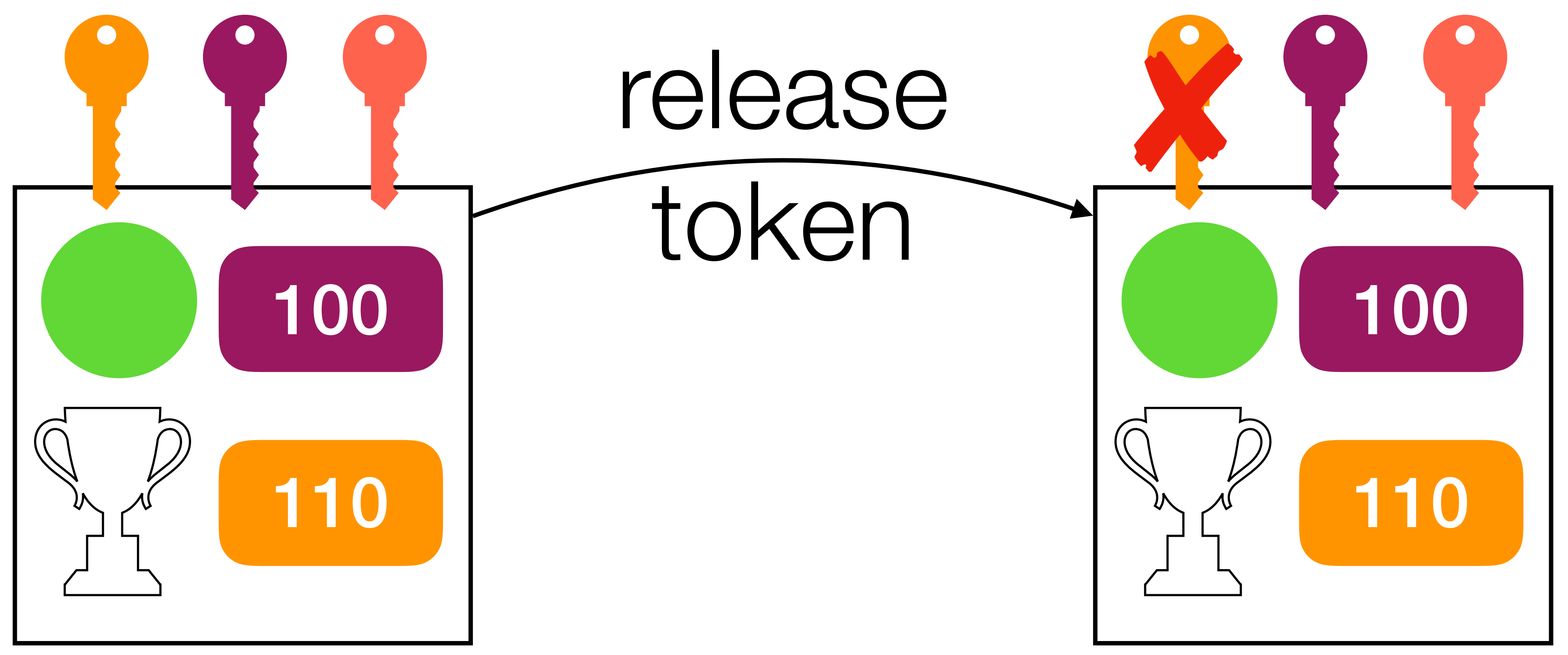
- Auction cannot be closed while bids are being placed
- Status can only go from: `inital` $\longmapsto$ `open` $\longmapsto$ `closed`
- Winner is highest bid when auction is closed
- Bids are strictly incremental
- Tokes go from `taken` $\longmapsto$ `released` (by owner)
- Close when all tokens are released



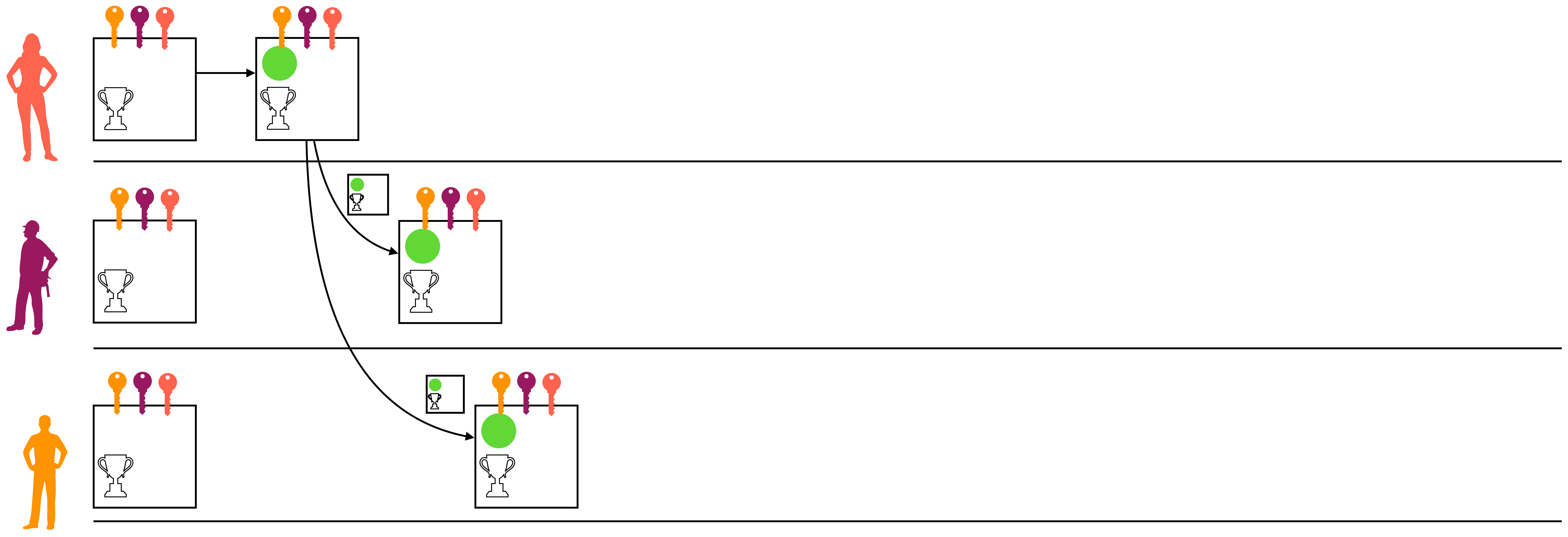


Tokens for CC: Taken

Released

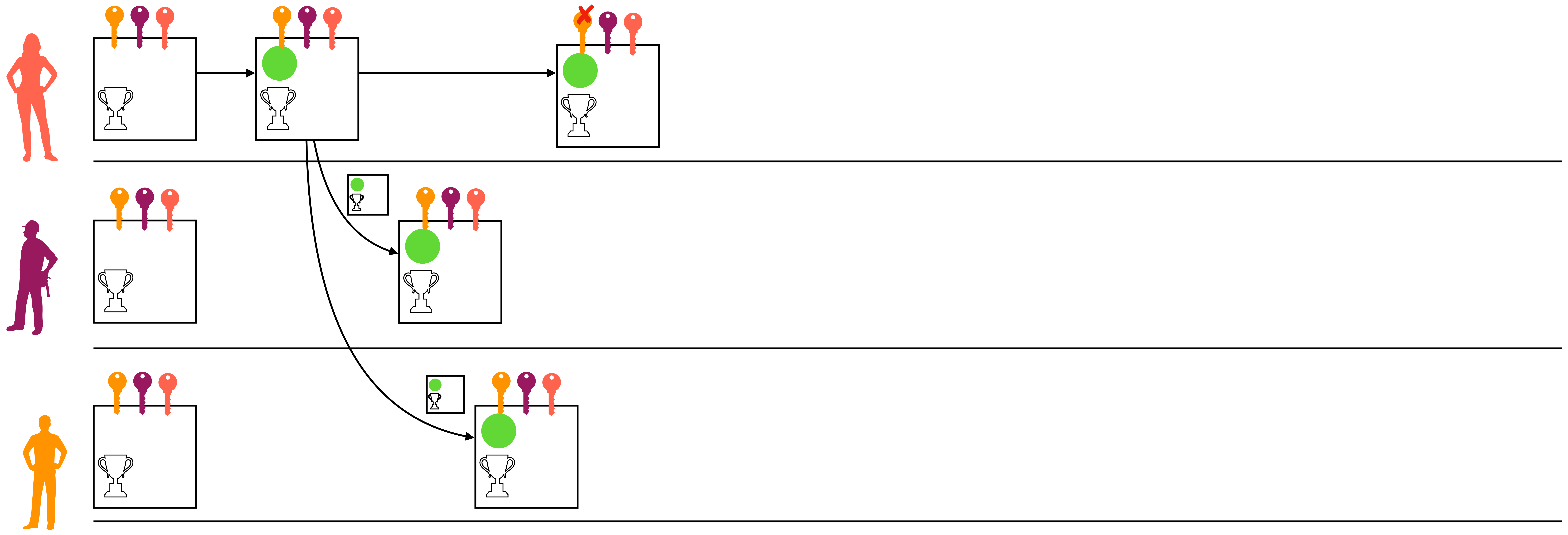


Auction + CC State Evolution



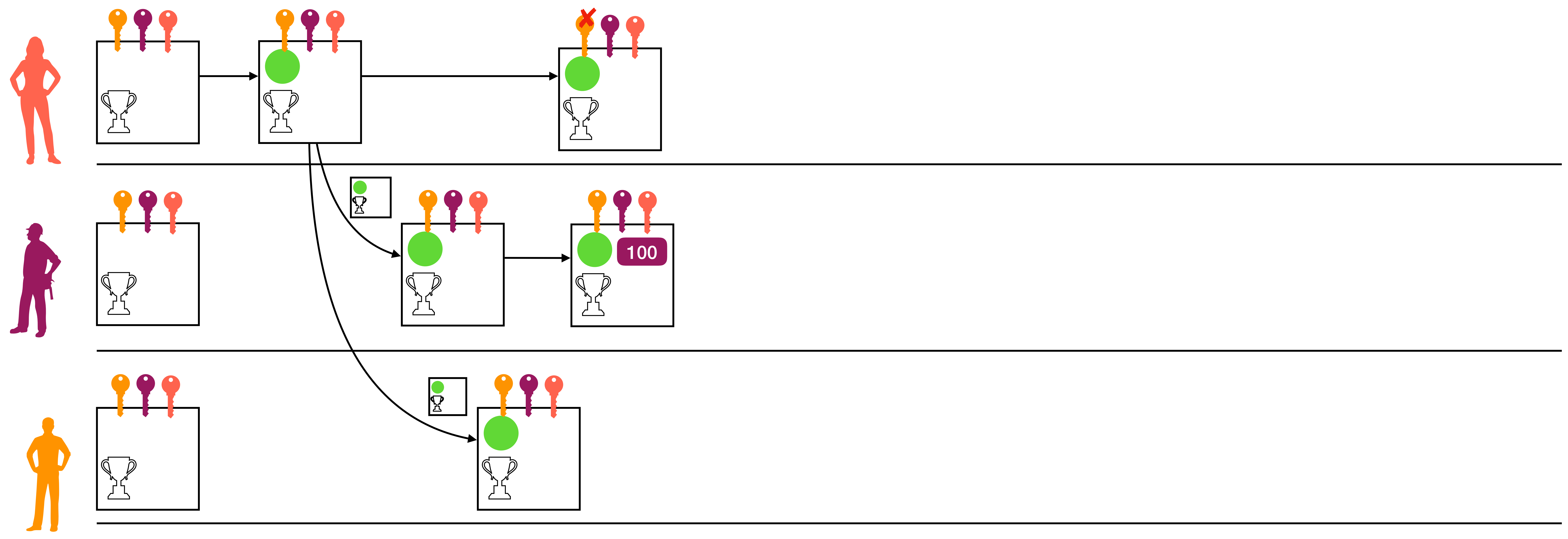
Tokens: 🔑 Auction status: ● Auction Result: 🏆 Auction Bidders: 100

Auction + CC State Evolution



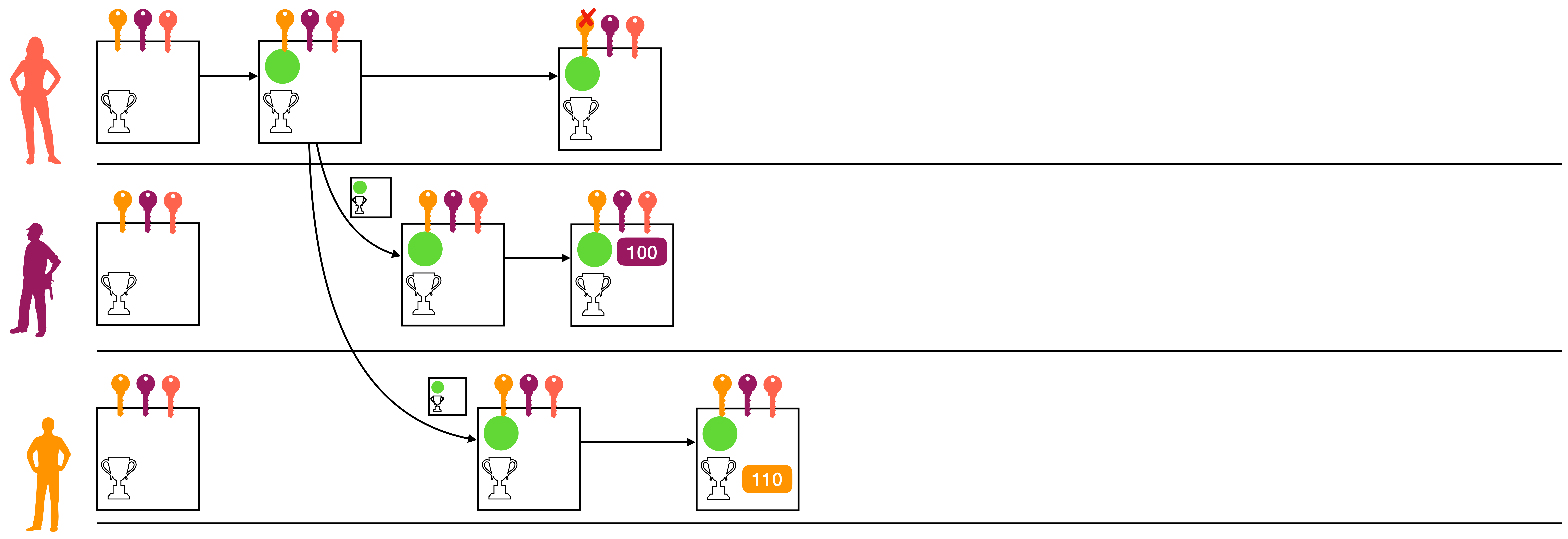
Tokens: 🔑 Auction status: ● Auction Result: 🏆 Auction Bidders: 100

Auction + CC State Evolution



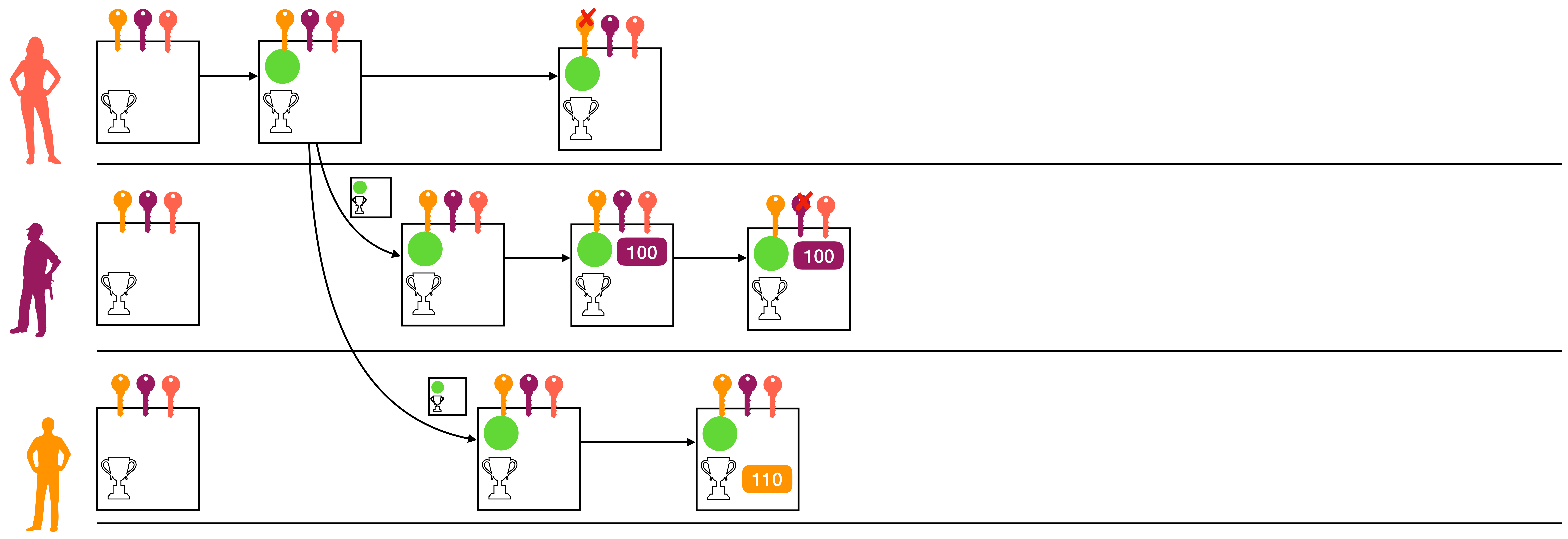
Tokens: 🔑 Auction status: ● Auction Result: 🏆 Auction Bidders: 100

Auction + CC State Evolution



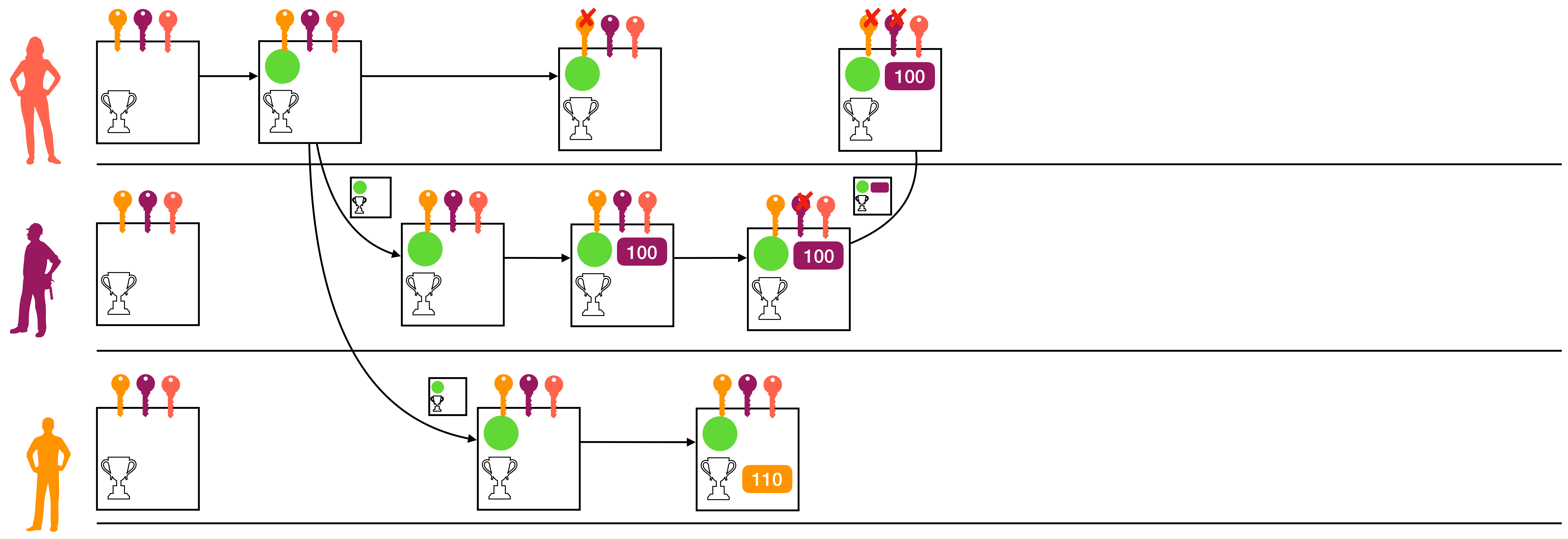
Tokens: 🔑 Auction status: ● Auction Result: 🏆 Auction Bidders: 100

Auction + CC State Evolution



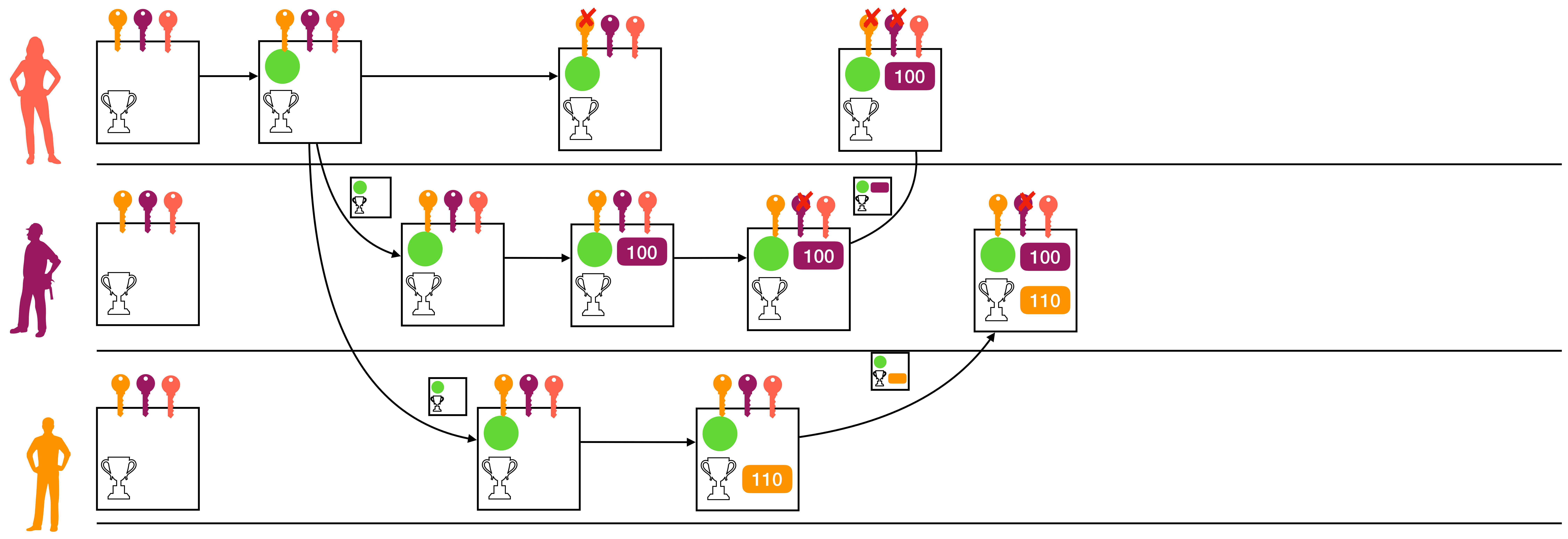
Tokens: Auction status: ● Auction Result: Auction Bidders: 100

Auction + CC State Evolution



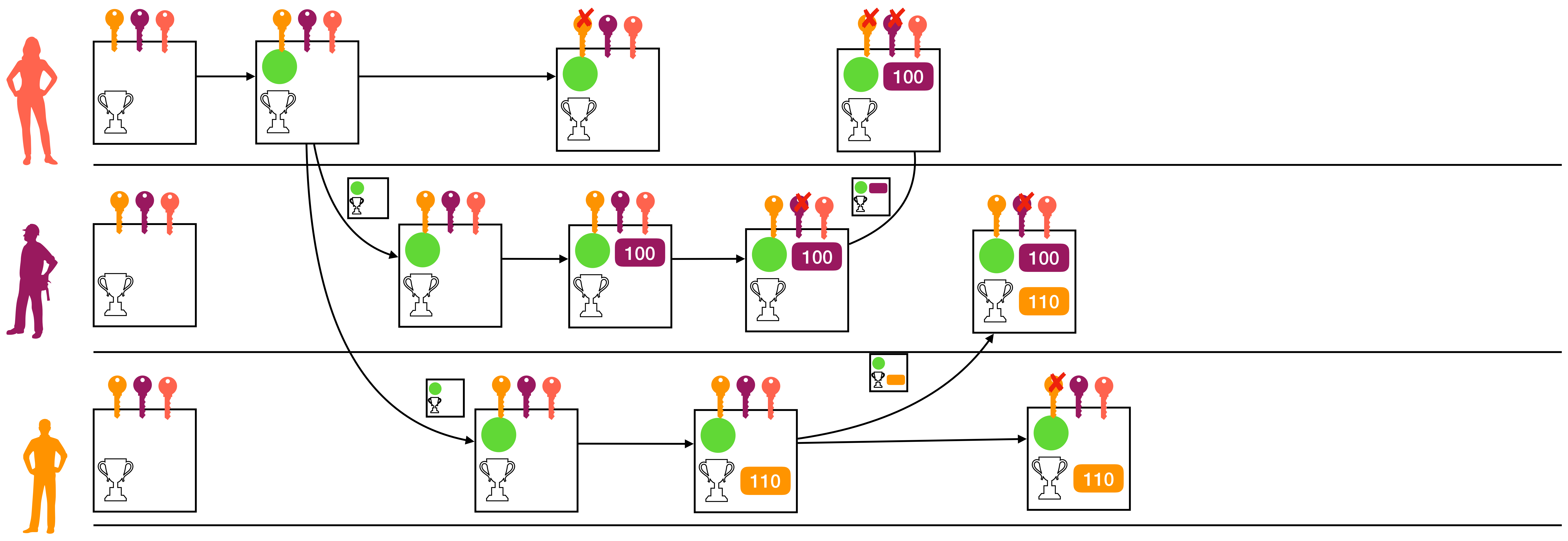
Tokens: Auction status: Auction Result: Auction Bidders: 100

Auction + CC State Evolution



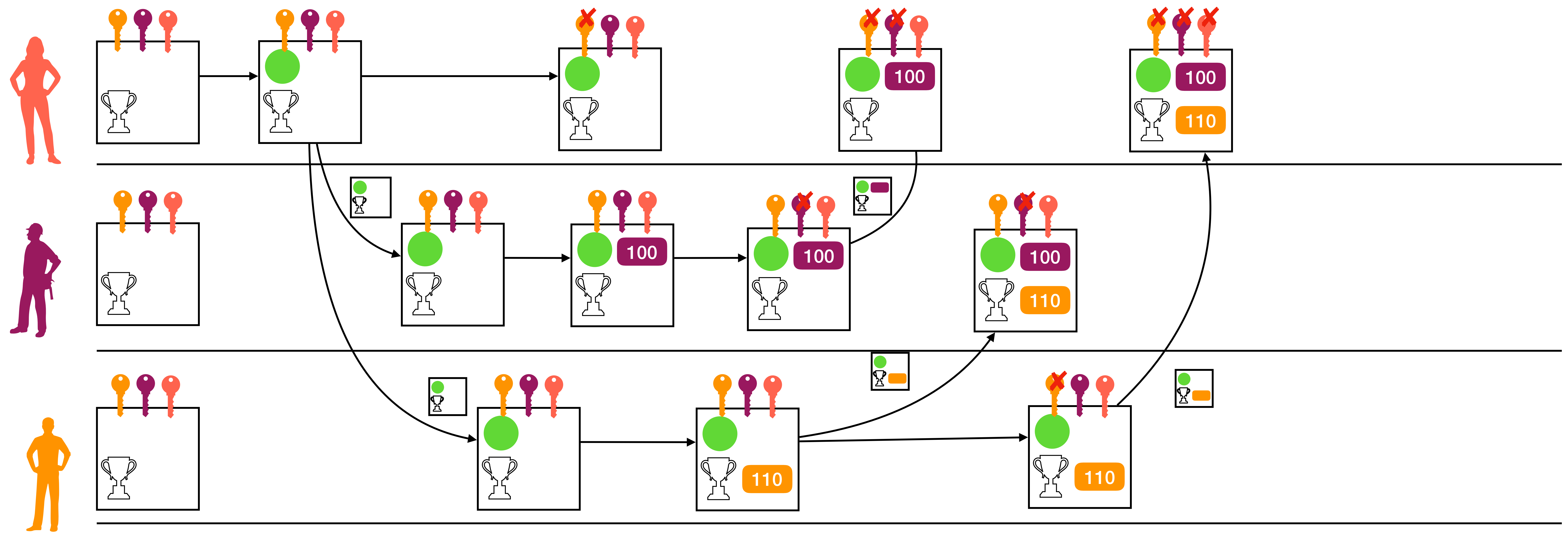
Tokens: ● Auction status: ● Auction Result: ● Auction Bidders: 100

Auction + CC State Evolution



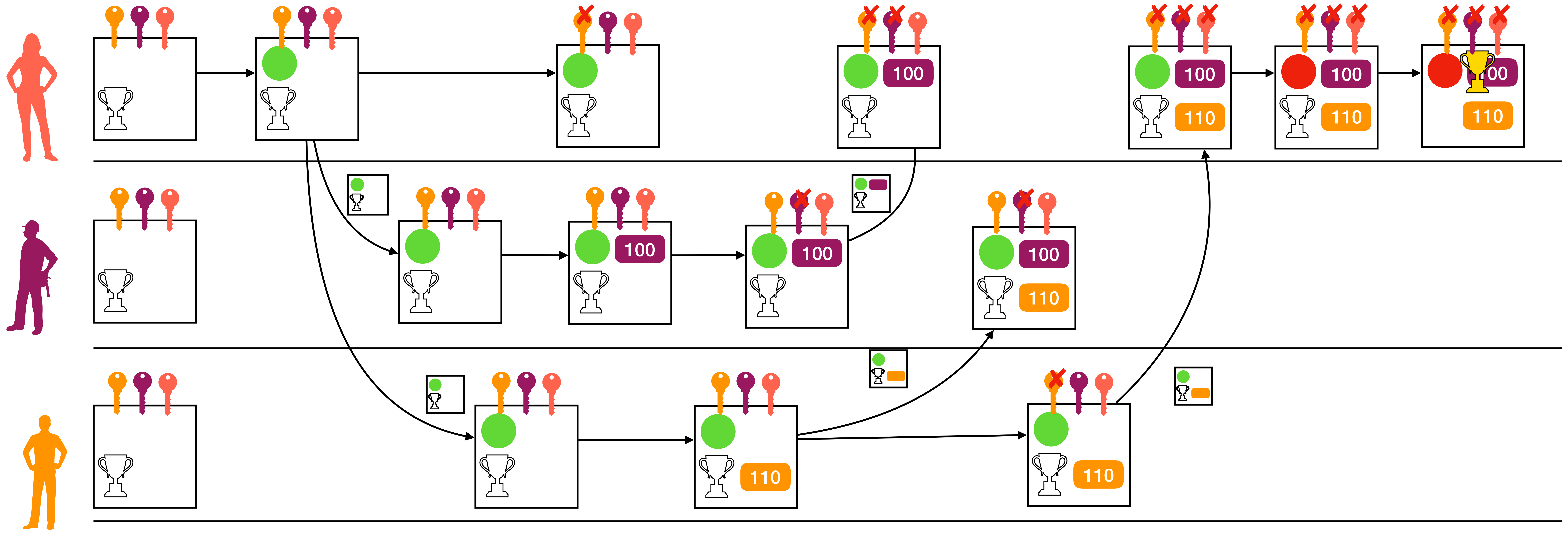
Tokens: 🔑 Auction status: ● Auction Result: 🏆 Auction Bidders: 100

Auction + CC State Evolution



Tokens: Auction status: ● Auction Result: Auction Bidders: 100

Auction + CC State Evolution



Tokens: Auction status: Auction Result: Auction Bidders: 100

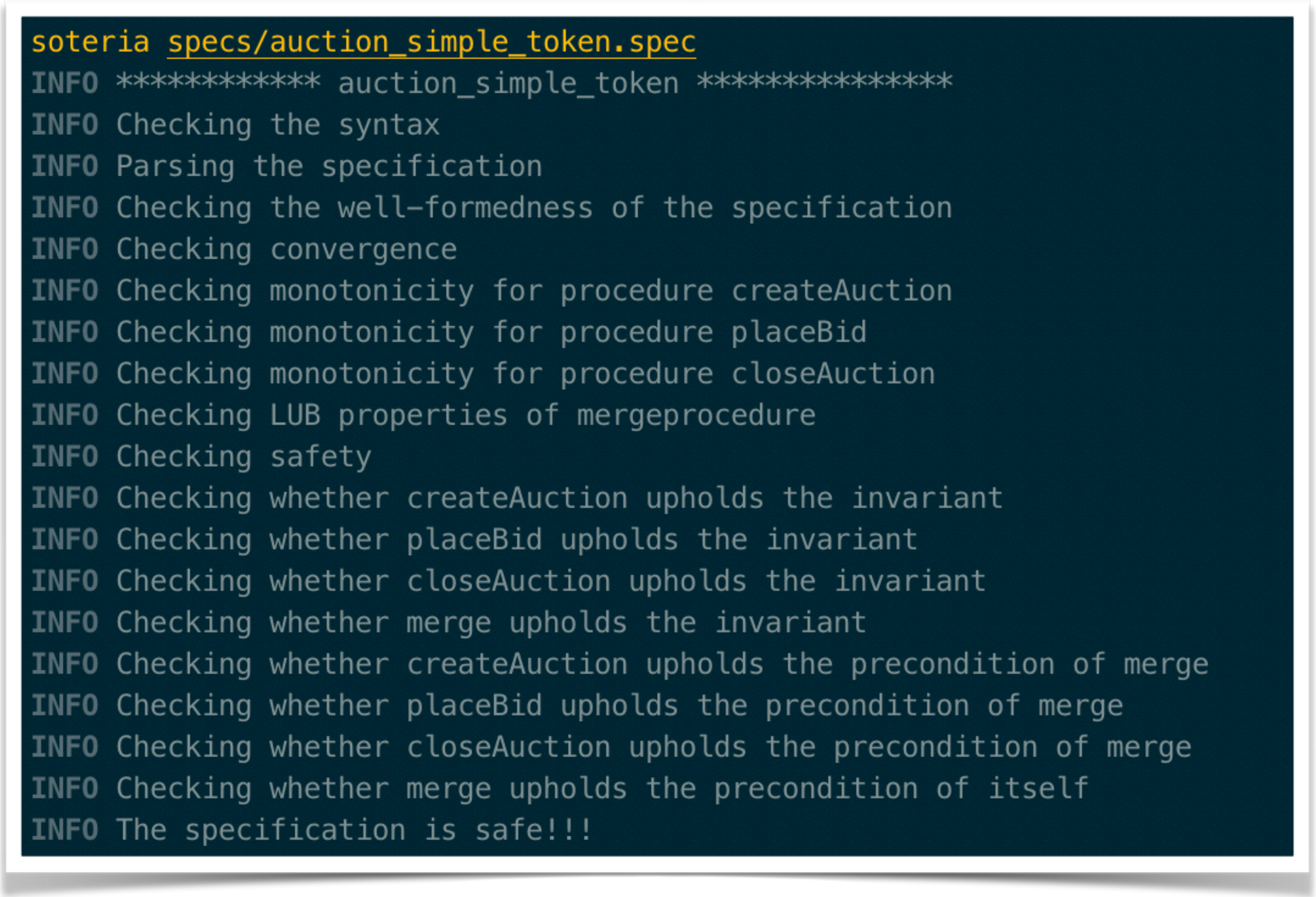
AUCTION IN BOOGIE

```
Initial state:              Invariant:
  status = INVALID            B.placed ⟹ status≥ACTIVE ∧ B.amount>0
  ∧ winner = ⊥                status≤ACTIVE ⟹ winner=⊥
  ∧ ∄ b∈B, b.placed           status=CLOSED ⟹ winner.placed ∧ is_highest(B, winner)
  ∧ ∀ t∈T, t                  status=CLOSED ⟹ ¬T
```

```
{Pre_merge:
  winner=winner_0 ∨ winner=⊥ ∨ winner_0=⊥
  ∧ B.amount = B_0.amount
  ∧ status=CLOSED ⟹ is_highest(B, winner) ∧ is_highest(B_0, winner)
  ∧ status_0=CLOSED ⟹ is_highest(B, winner_0) ∧ is_highest(B_0, winner_0)
  ∧ t.me ⟹ t_0.me
  ∧ (¬T ∧ ¬b.placed) ⟹ ¬b_0.placed
  ∧ ((∀ r, r≠me ∧ ¬t.r) ∧ ¬b.placed) ⟹ ¬b_0.placed
  ∧ ¬T ⟹ winner_0=winner ∨ winner_0=⊥
  ∧ T ⟹ winner=⊥ ∧ winner_0=⊥}
merge((status, winner, B, T), (status_0, winner_0, B_0, T_0)):
  status := max(status, status_0)
  winner := if winner_0≠⊥ then winner_0 else winner
  B.placed := B.placed∨B_0.placed
  B.amount := B.amount
  T := T∧T_0
```

TOOL SUPPORT

- Inputs:
  - Operations
  - Ordering relation ≤ for semi-lattice
  - Invariant **Inv**
- Derive **Pre**$_{\text{merge}}$ from **Inv**
- **Global invariants**: **Inv** and **Pre**$_{\text{merge}}$
- Check semi-lattice: *convergence*
- Proofs are **local** to each operation
  - Boogie for (sequential) verification
- https://github.com/sreeja/soteria_tool



```
soteria specs/auction_simple_token.spec
INFO ************ auction_simple_token **************
INFO Checking the syntax
INFO Parsing the specification
INFO Checking the well-formedness of the specification
INFO Checking convergence
INFO Checking monotonicity for procedure createAuction
INFO Checking monotonicity for procedure placeBid
INFO Checking monotonicity for procedure closeAuction
INFO Checking LUB properties of mergeprocedure
INFO Checking safety
INFO Checking whether createAuction upholds the invariant
INFO Checking whether placeBid upholds the invariant
INFO Checking whether closeAuction upholds the invariant
INFO Checking whether merge upholds the invariant
INFO Checking whether createAuction upholds the precondition of merge
INFO Checking whether placeBid upholds the precondition of merge
INFO Checking whether closeAuction upholds the precondition of merge
INFO Checking whether merge upholds the precondition of itself
INFO The specification is safe!!!
```

CONCLUSION

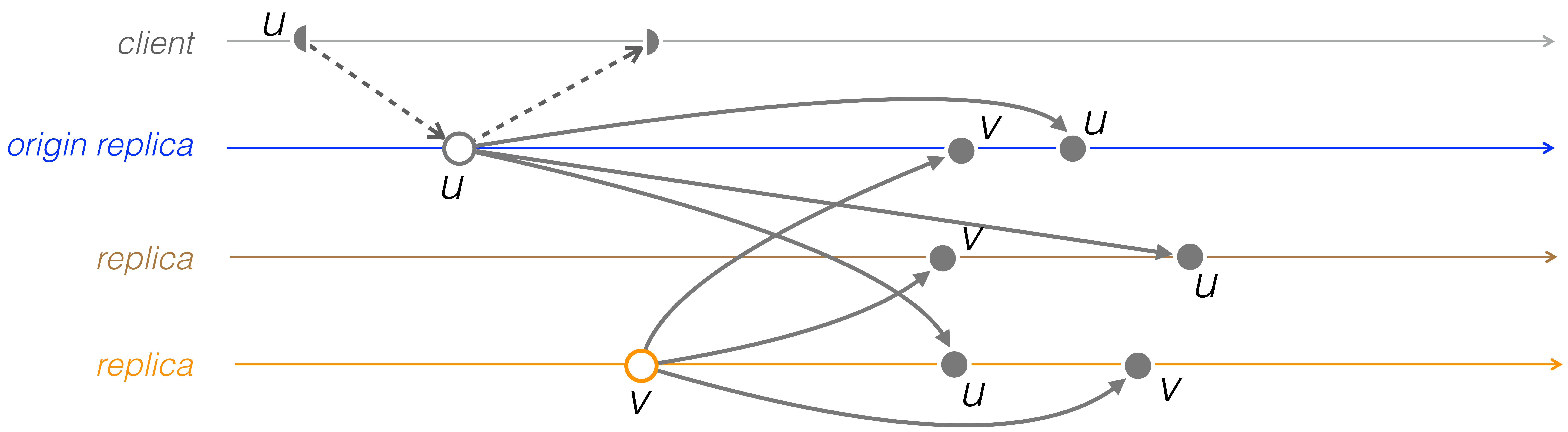
- Modular verification of State-based CRDT applications
- SOTERIA: Tool support based on Boogie
- WIP: Concurrency Control synthesis (recommendations)

Operation-based CRDTs

- Operation-based CRDTs
  - Each operation is delivered to each replica

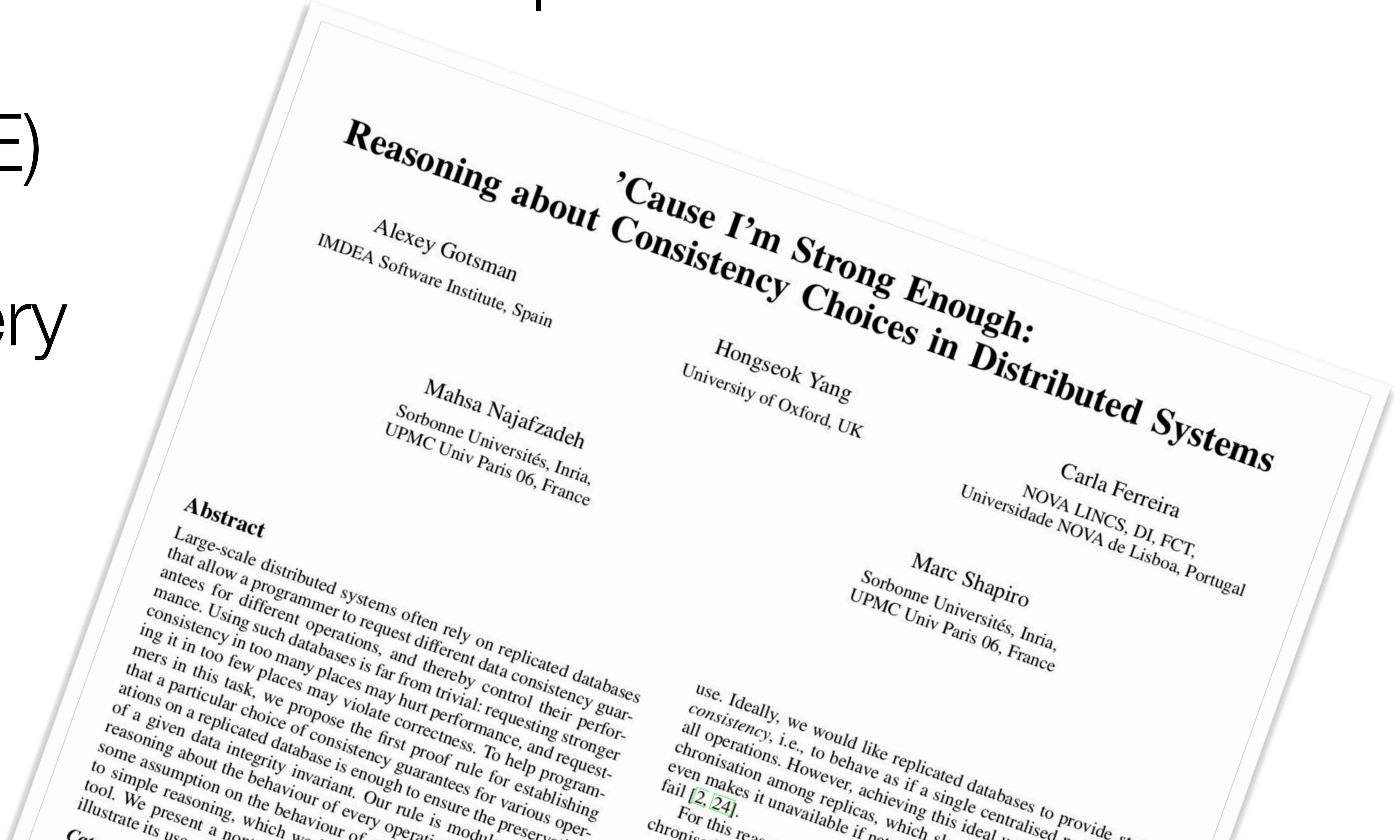
OPERATION-BASED CRDTS

- Operation-based CRDTs
  - Each operation is delivered to each replica



Operation-based CRDTs

- Operation-based CRDTs
  - Each operation is delivered to each replica
- Invariant Checking (CISE)
  - Requires causal delivery



'Cause I'm Strong Enough: Reasoning about Consistency Choices in Distributed Systems

Alexey Gotsman
IMDEA Software Institute, Spain

Hongseok Yang
University of Oxford, UK

Mahsa Najafzadeh
Sorbonne Universités, Inria,
UPMC Univ Paris 06, France

Carla Ferreira
NOVA LINCS, DI, FCT,
Universidade NOVA de Lisboa, Portugal

Marc Shapiro
Sorbonne Universités, Inria,
UPMC Univ Paris 06, France

Abstract

Large-scale distributed systems often rely on replicated databases that allow a programmer to request different data consistency guarantees for different operations, and thereby control their performance. Using such databases is far from trivial: requesting stronger consistency in too many places may hurt performance, and requesting it in too few places may violate correctness. To help programmers in this task, we propose the first proof rule for establishing that a particular choice of consistency guarantees for various operations on a replicated database is enough to ensure the preservatio of a given data integrity invariant. Our rule is modul reasoning about the behaviour of every operati some assumption on the behaviour of to simple reasoning, which w tool. We present a non illustrate its use

use. Ideally, we would like replicated databases to provide *consistency*, i.e., to behave as if a single centralised all operations. However, achieving this ideal chronisation among replicas, which even makes it unavailable if fail [2, 24].

For this reas chronis