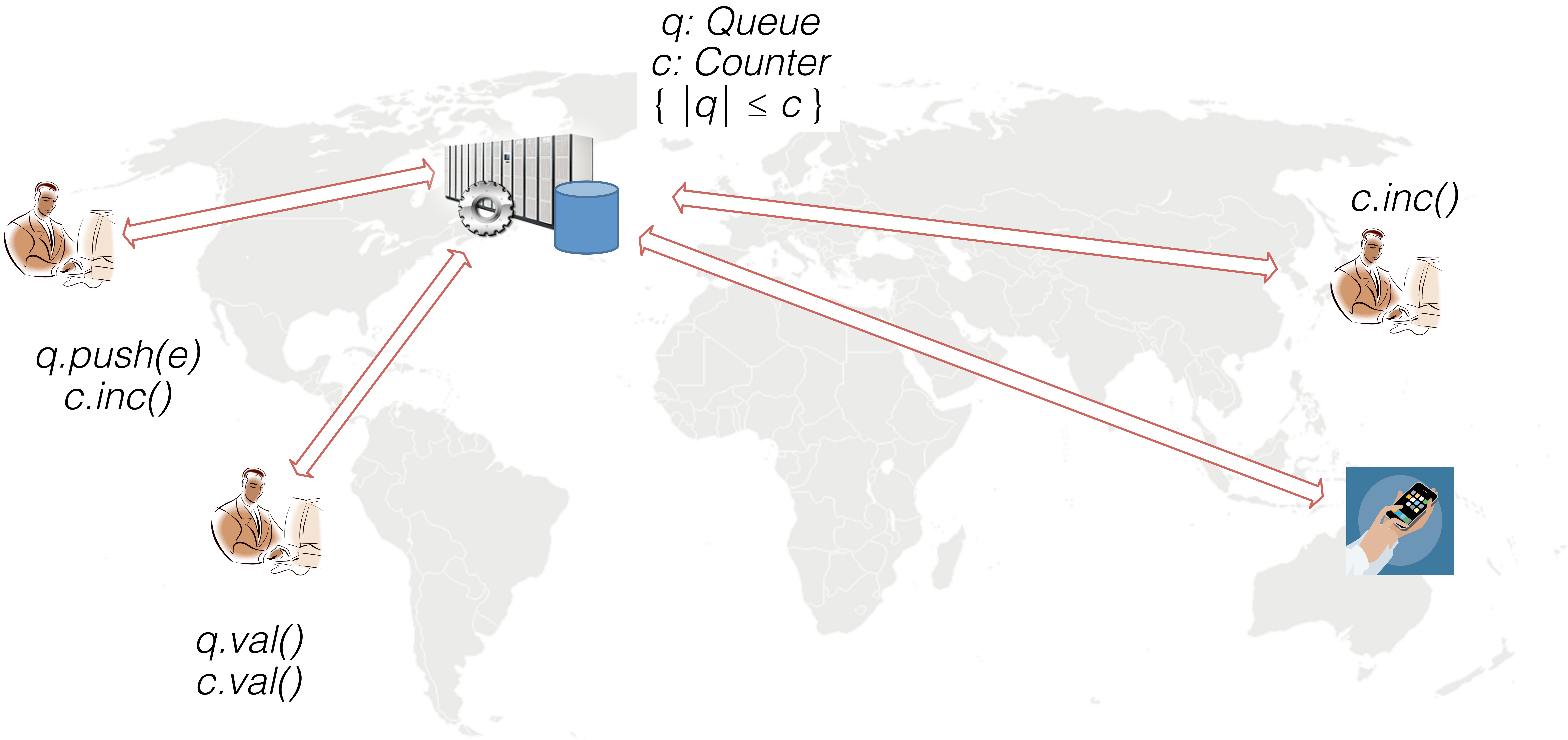
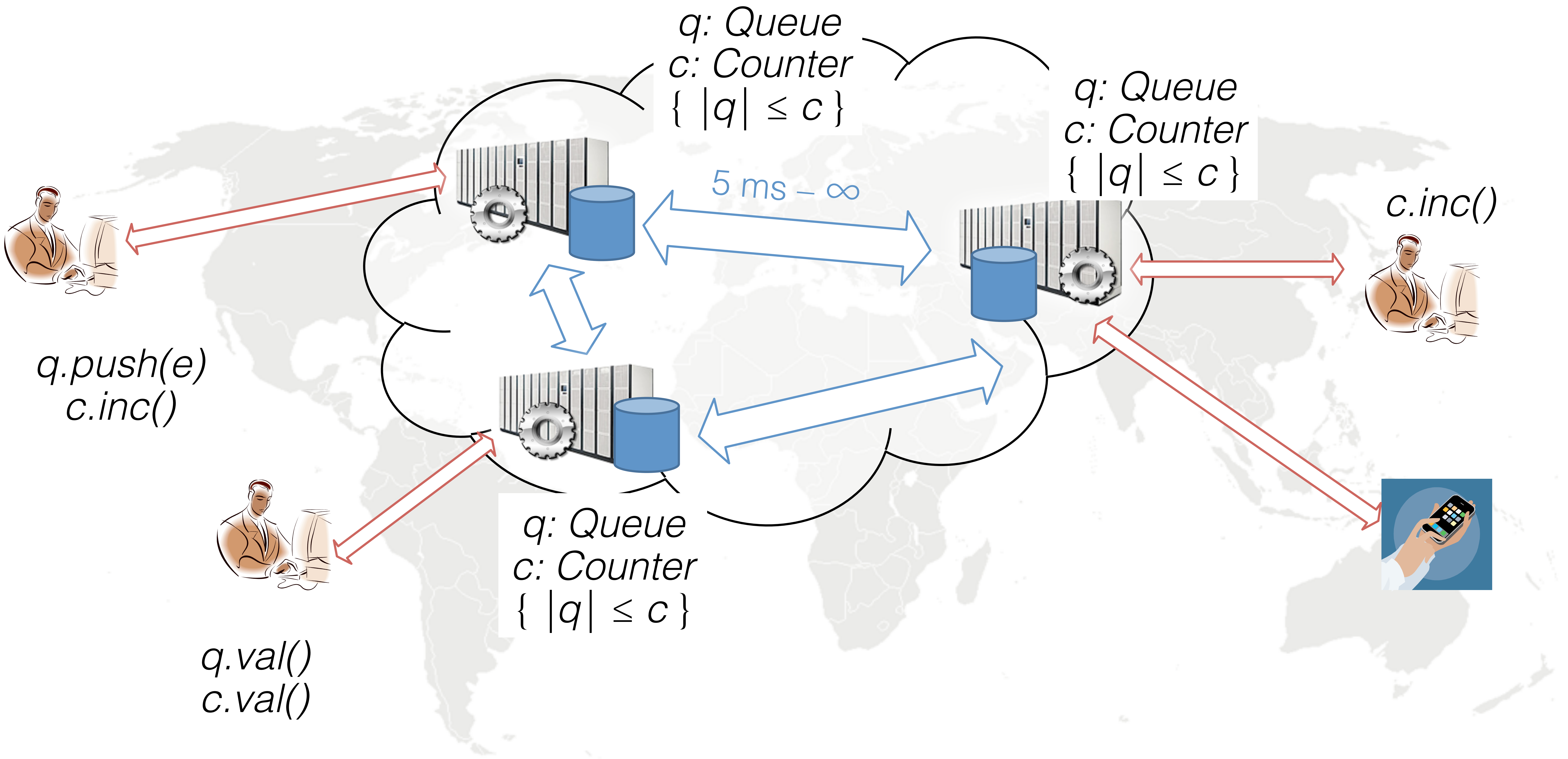
VERIFYING IMPLEMENTATIONS OF CRDTS

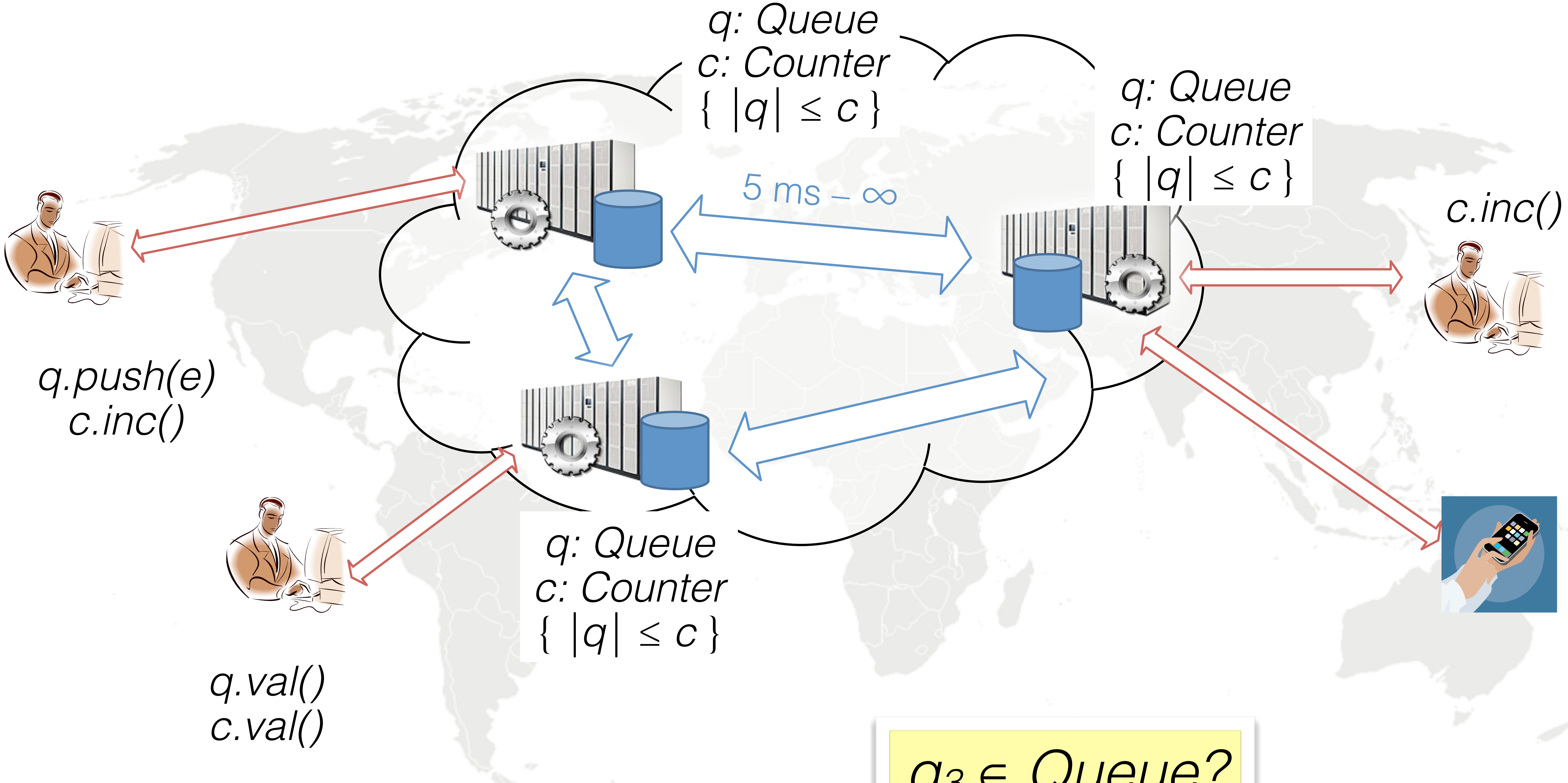
SHARED DATA TYPE



GEO-REPLICATED DATA TYPE



GEO-REPLICATED DATA TYPE

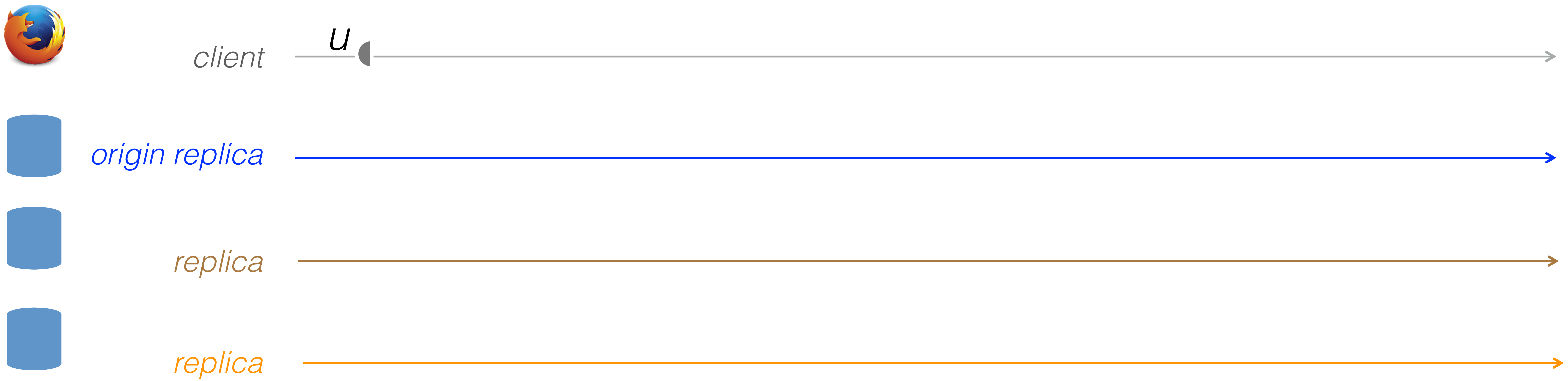


$q_3 \in$ Queue?

$q_1 = q_2$?

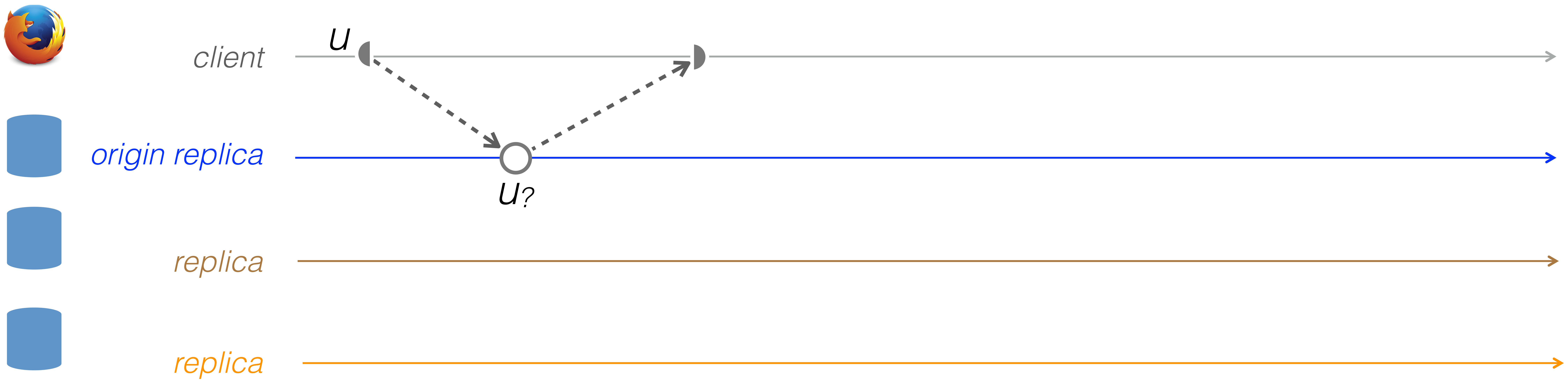
$|q_1| \leq c_4$?

Program Model (operation)



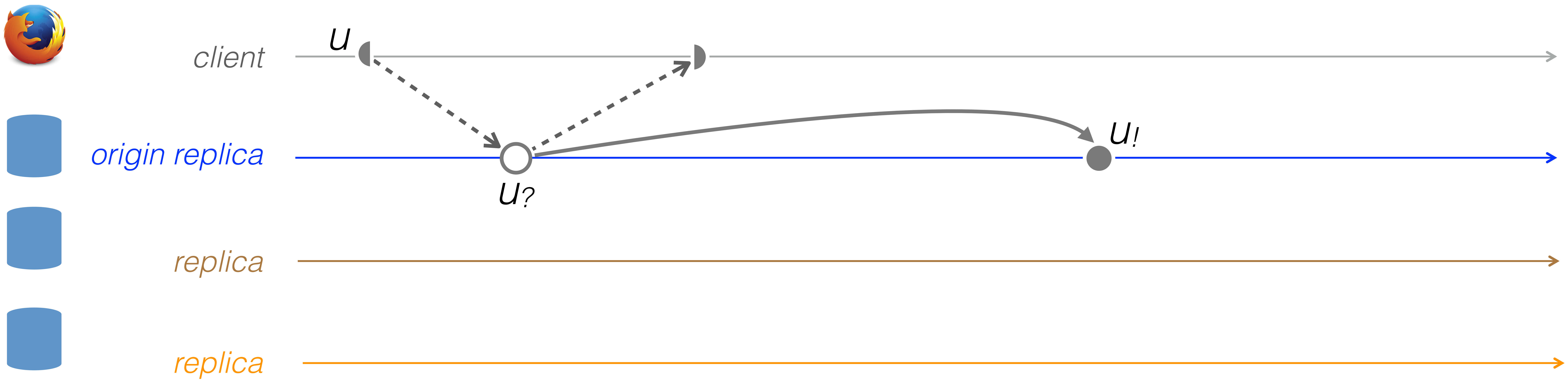
- $u: \mathit{state} \curvearrowright (\mathit{retval}, (\mathit{state} \curvearrowright \mathit{state}))$
- Prepare (@origin) $u_?$; deliver $u_!$
- Read One, Write All
- Deferred-Update Replication

Program Model (Operation)



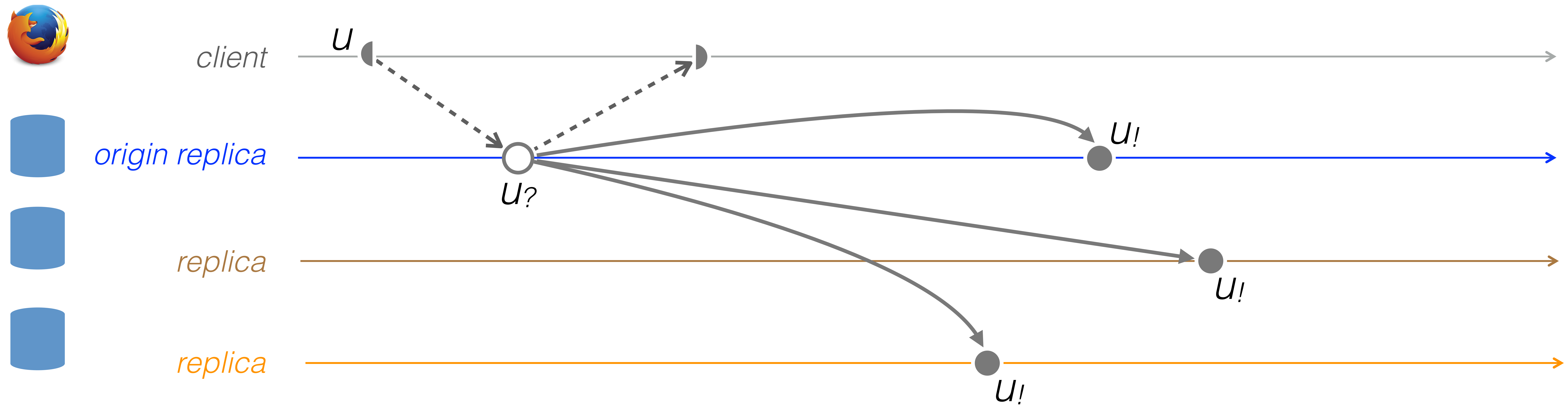
- $u: \mathit{state} \curvearrowright (\mathit{retval}, (\mathit{state} \curvearrowright \mathit{state}))$
- Prepare (@origin) $u_?$; deliver $u_!$
- Read One, Write All
- Deferred-Update Replication

Program Model (operation)



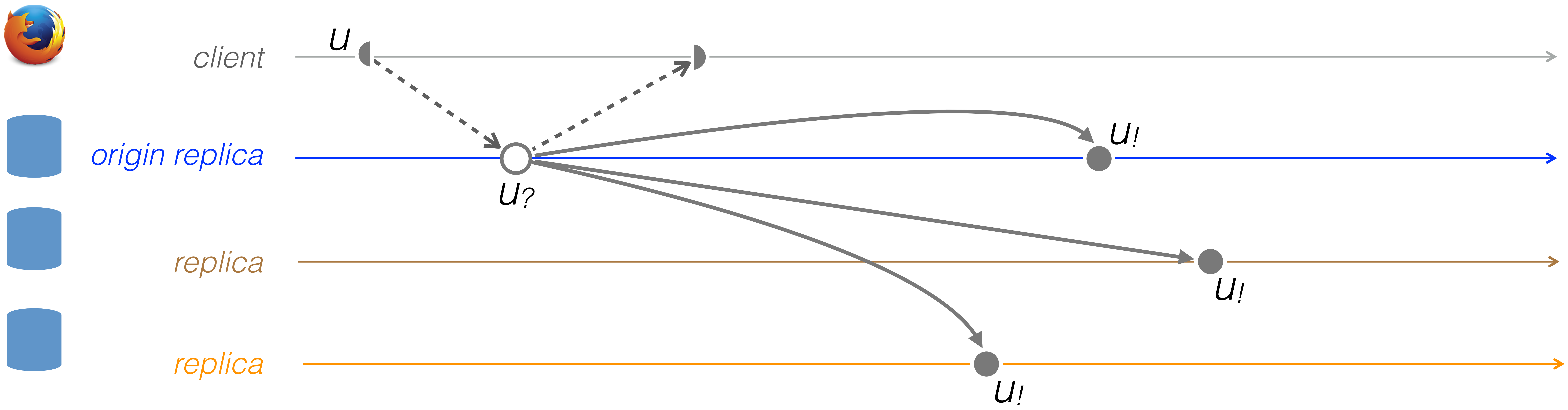
- $u: state \curvearrowright (retval, (state \curvearrowright state))$
- Prepare (@origin) $u_?$; deliver $u_!$
- Read One, Write All
- Deferred-Update Replication

PROGRAM MODEL (OPERATION)

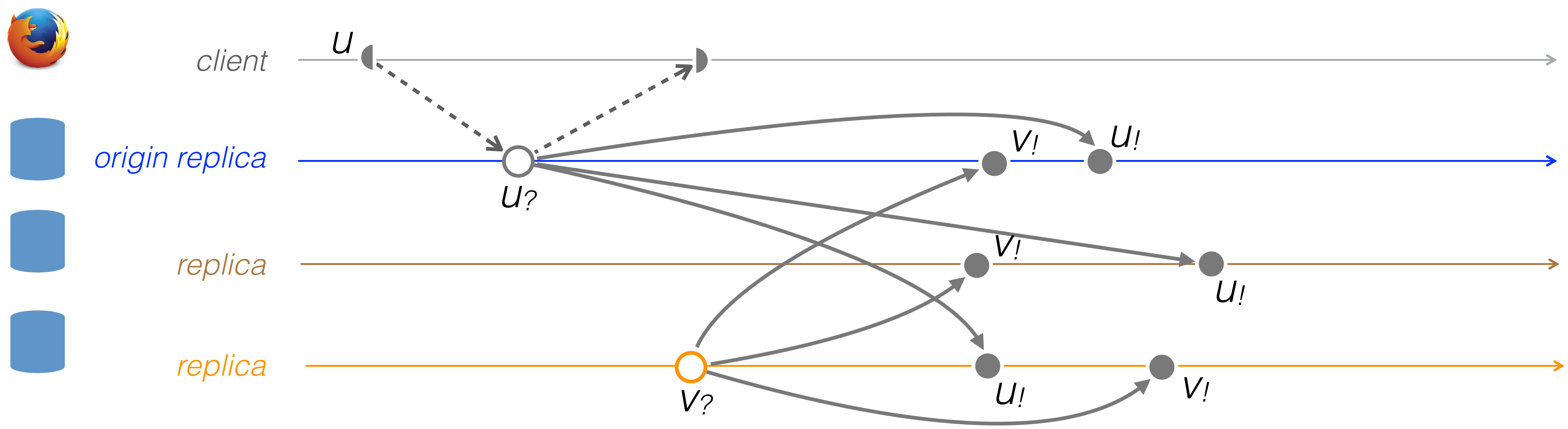


- *u: state ⤻ (retval, (state ⤻ state))*
- Prepare (@origin) *u?*; deliver *u!*
- Read One, Write All
- Deferred-Update Replication

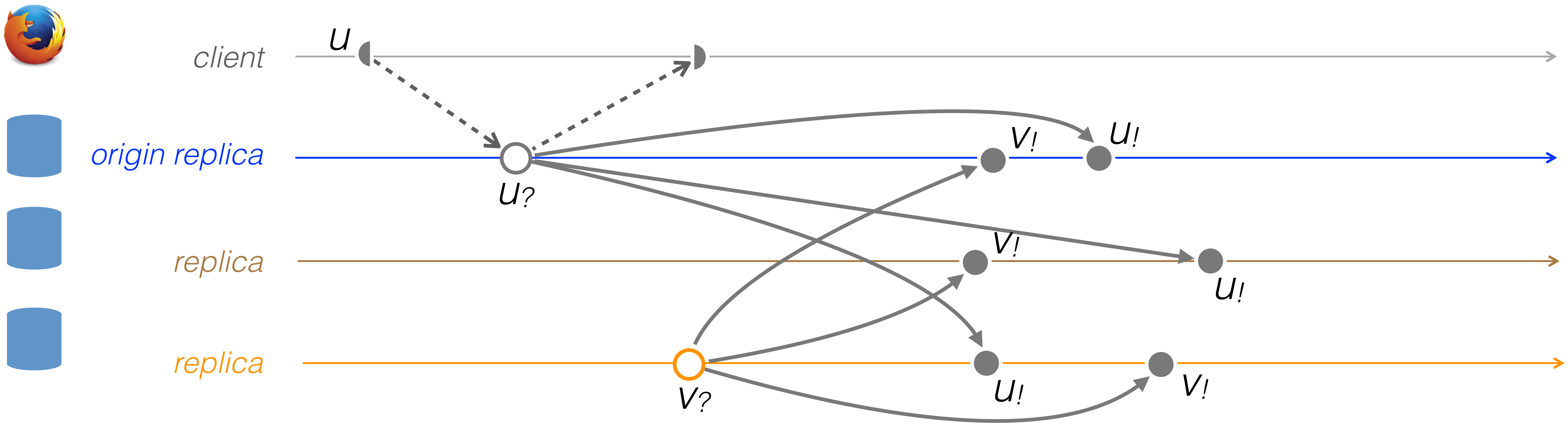
PROGRAM MODEL (SYSTEM)



PROGRAM MODEL (SYSTEM)

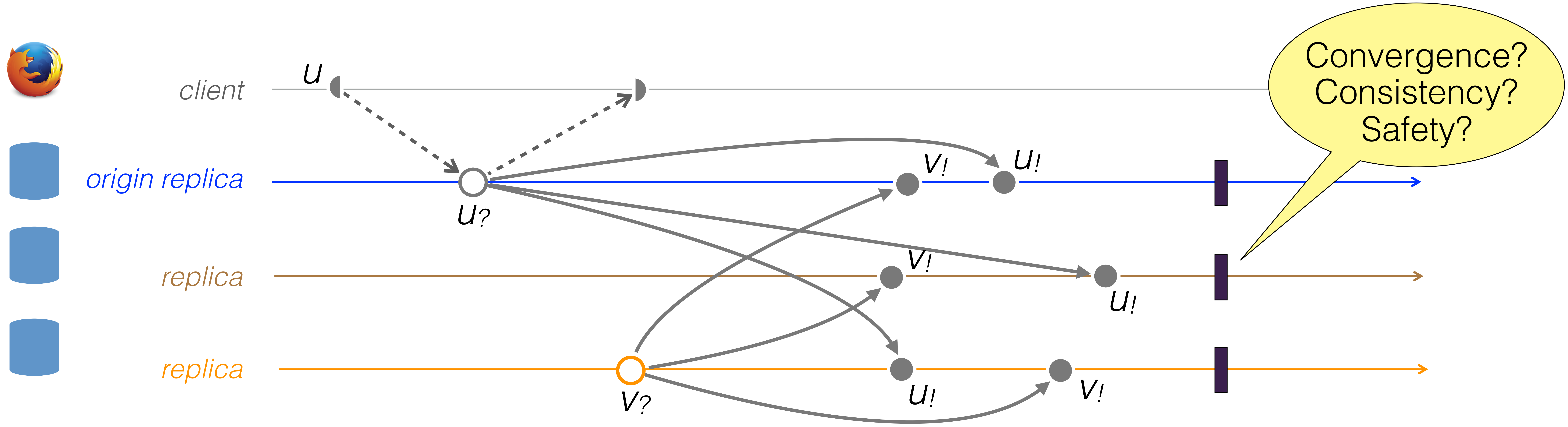


PROGRAM MODEL (SYSTEM)



- Concurrent, Multi-master
- Strong: total order, identical state
- Weak: concurrent, interleaving, no global state

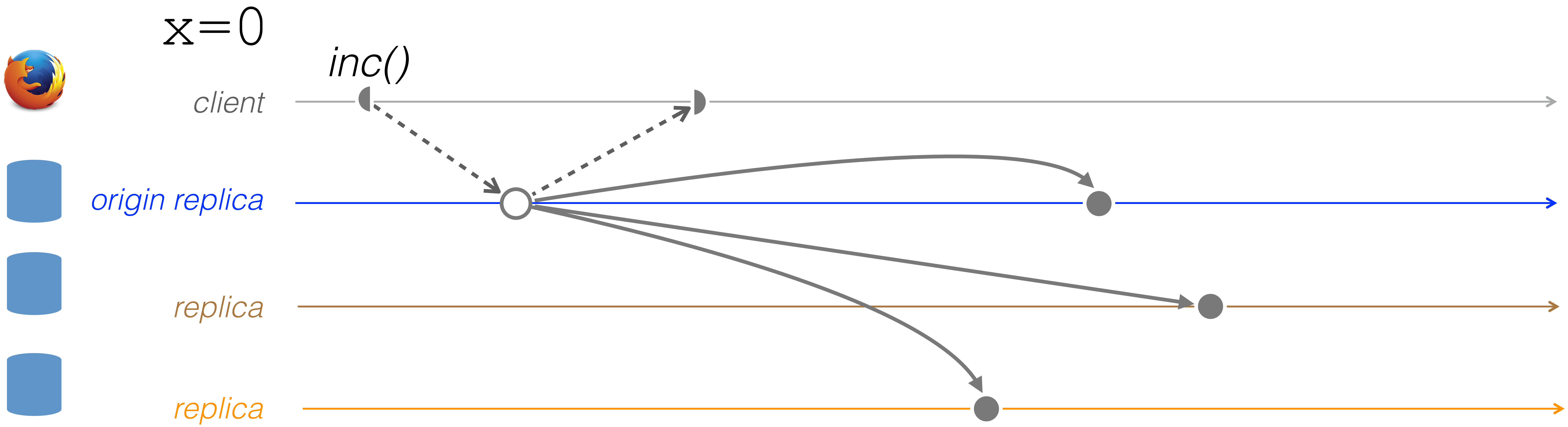
PROGRAM MODEL (SYSTEM)



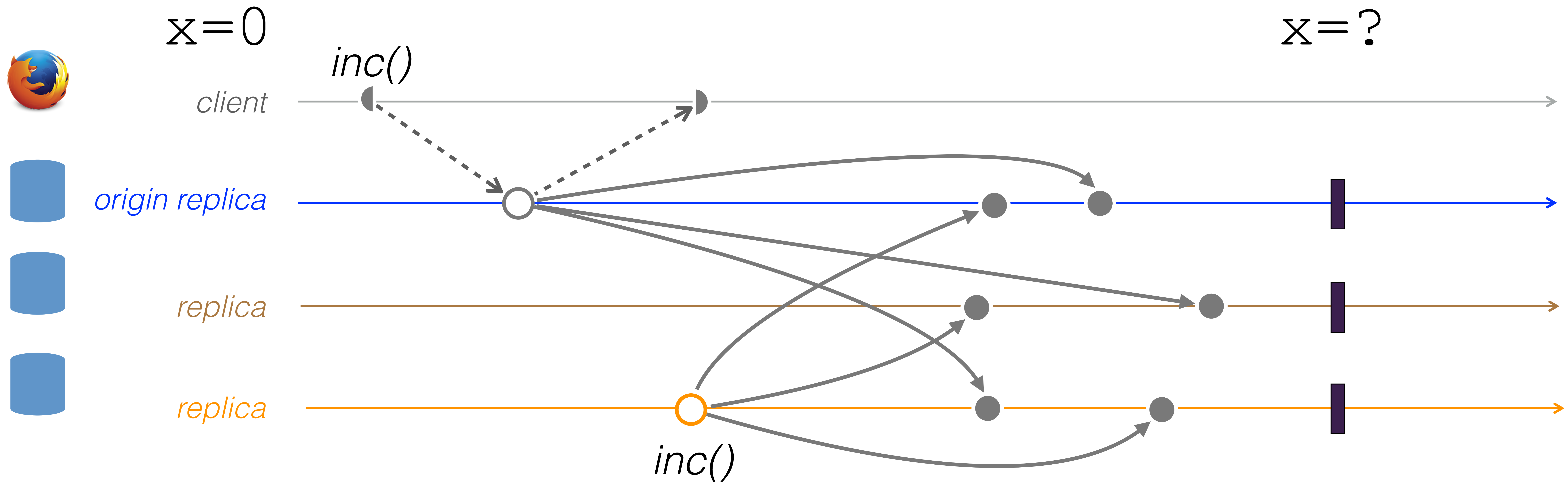
- Concurrent, Multi-master
- Strong: total order, identical state
- Weak: concurrent, interleaving, no global state

CRDT Examples

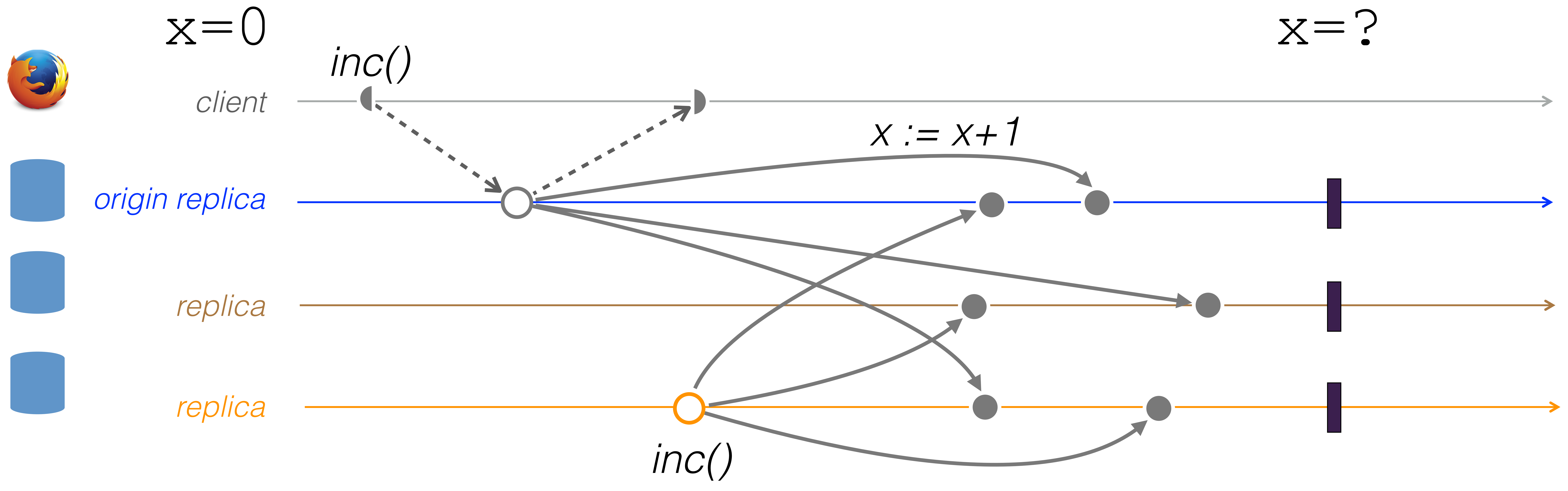
Example: Grow-Only Counter



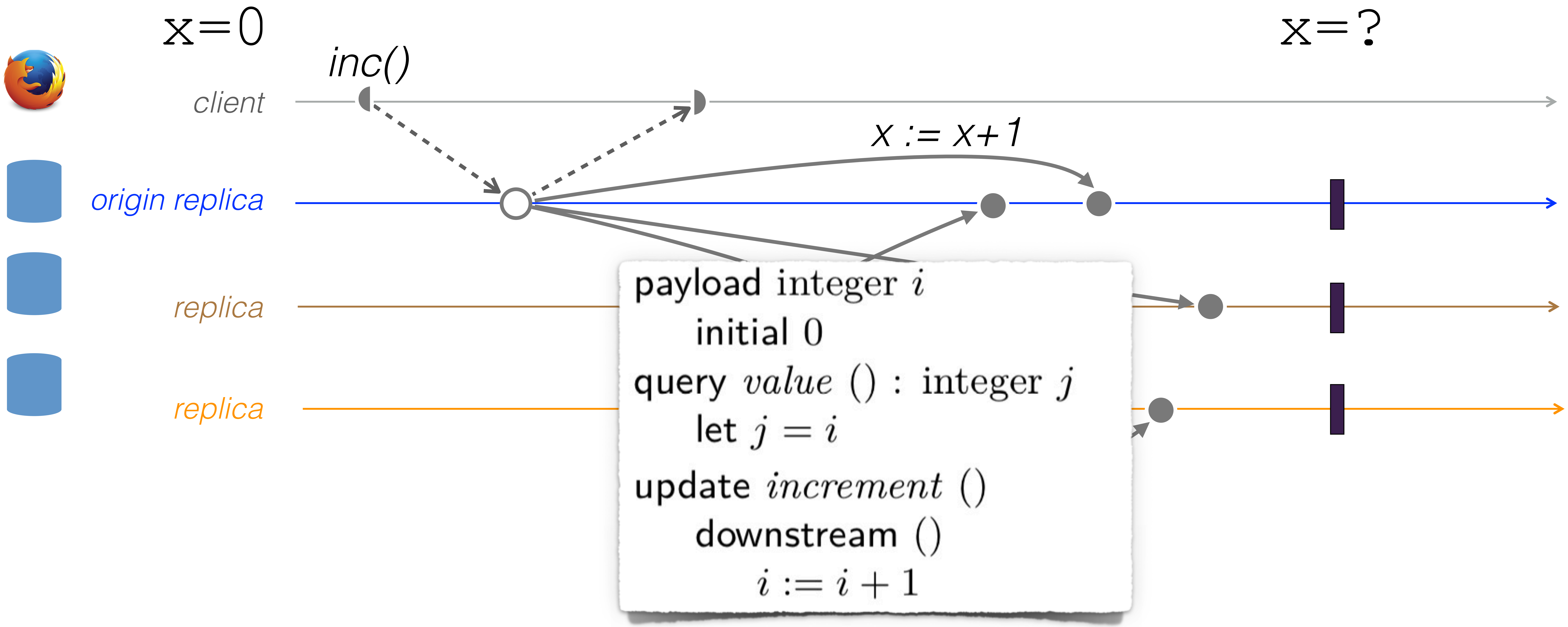
Example: Grow-Only Counter



Example: Grow-Only Counter

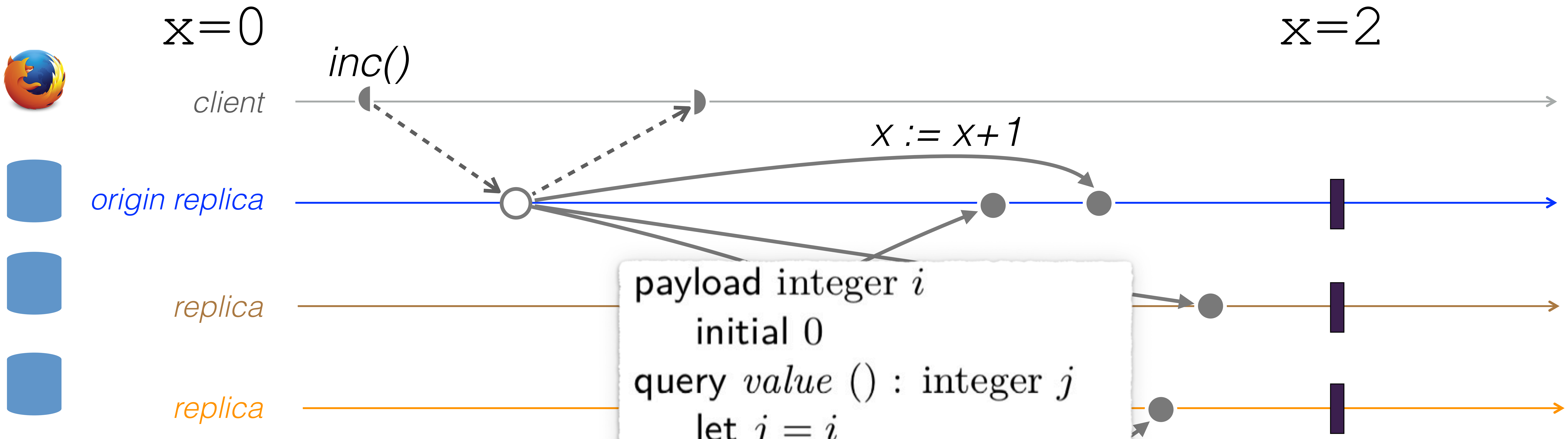


Example: Grow-Only Counter



[A Comprehensive Study of CRDT's (Shapiro et al. 2011)]

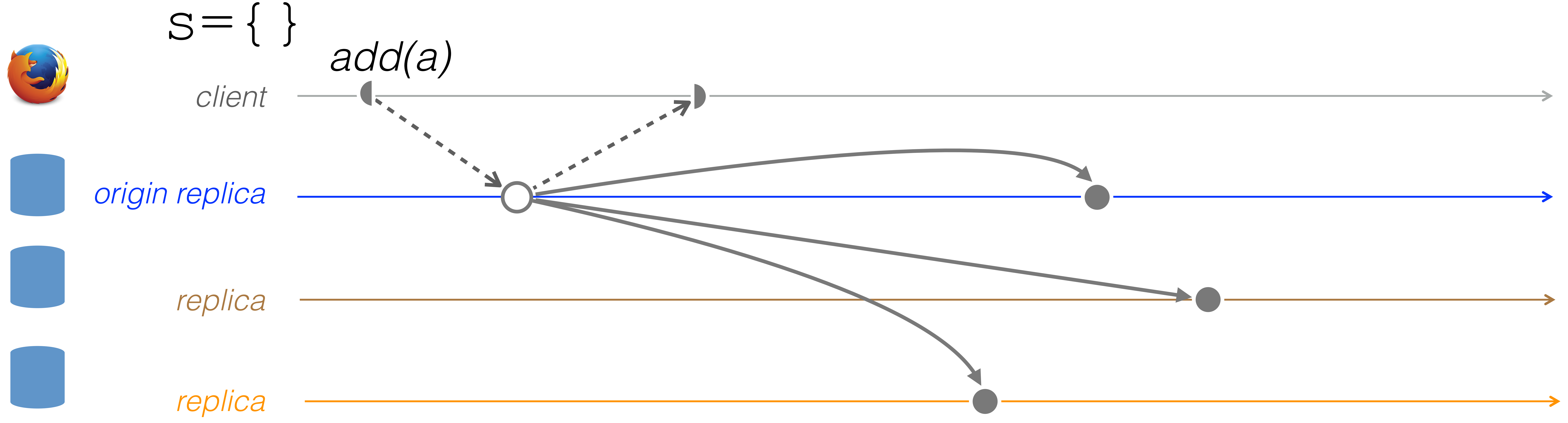
EXAMPLE: GROW-ONLY COUNTER



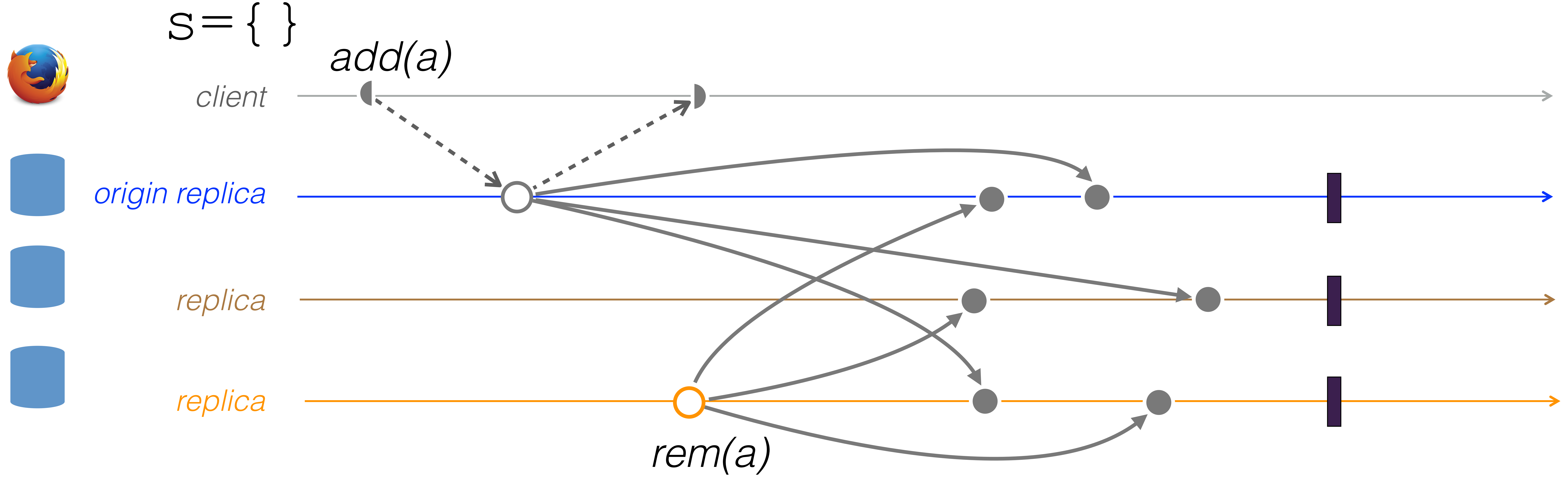
payload integer i
  initial 0
query *value* $()$: integer j
  let $j = i$
update *increment* $()$
  downstream $()$
    $i := i + 1$

[A Comprehensive Study of CRDT's (Shapiro et al. 2011)]

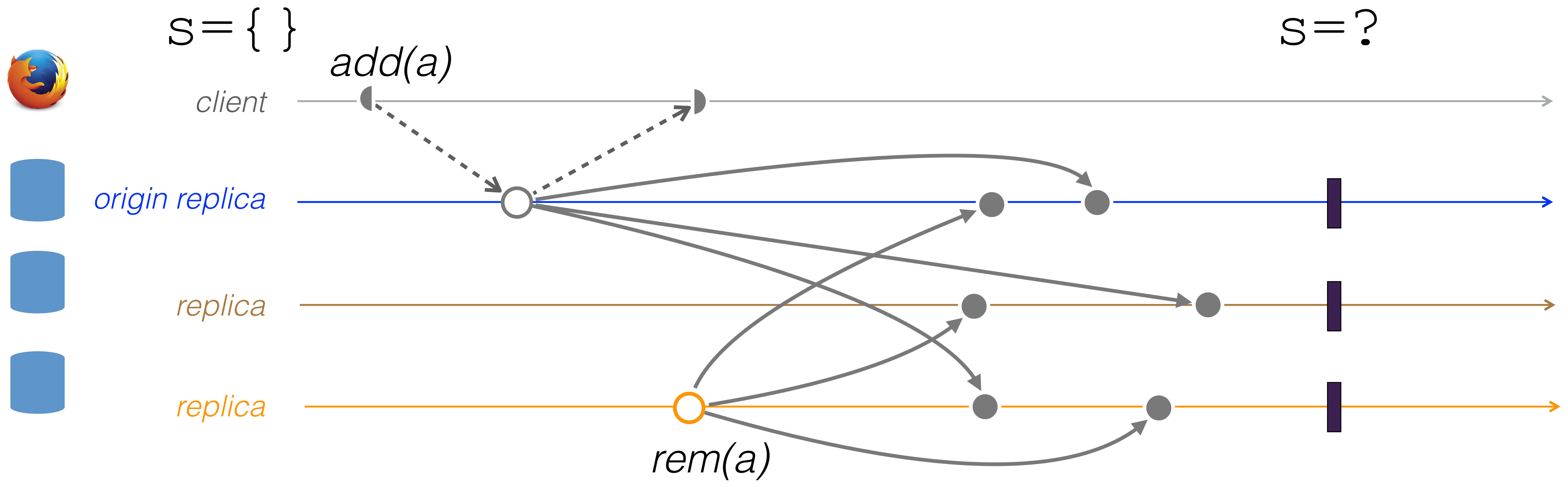
EXAMPLE: OBSERVED-REMOVE SET



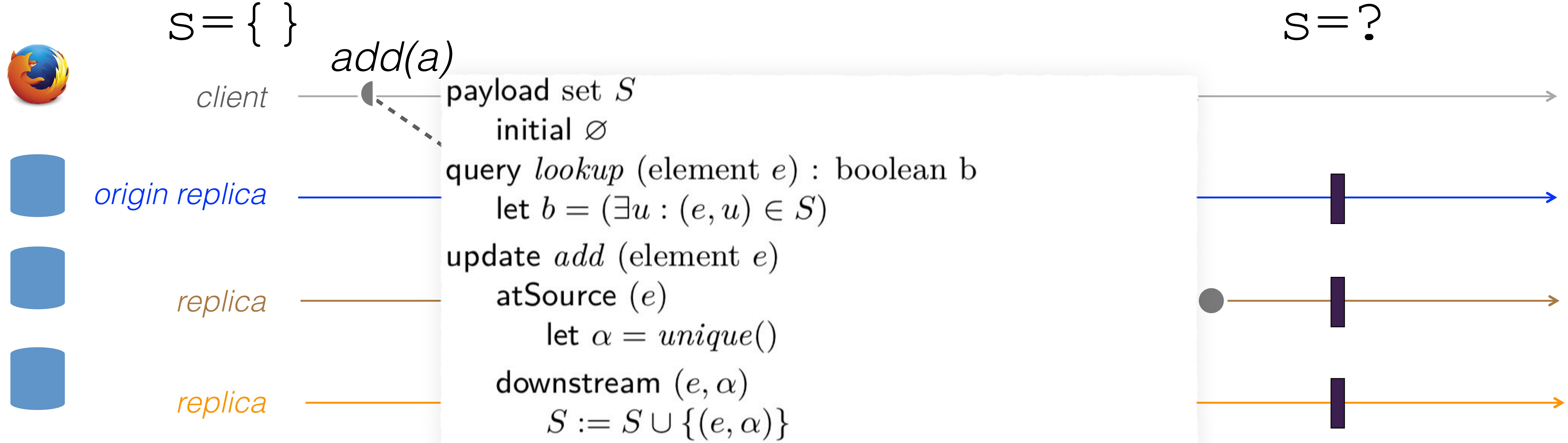
EXAMPLE: OBSERVED-REMOVE SET



EXAMPLE: OBSERVED-REMOVE SET



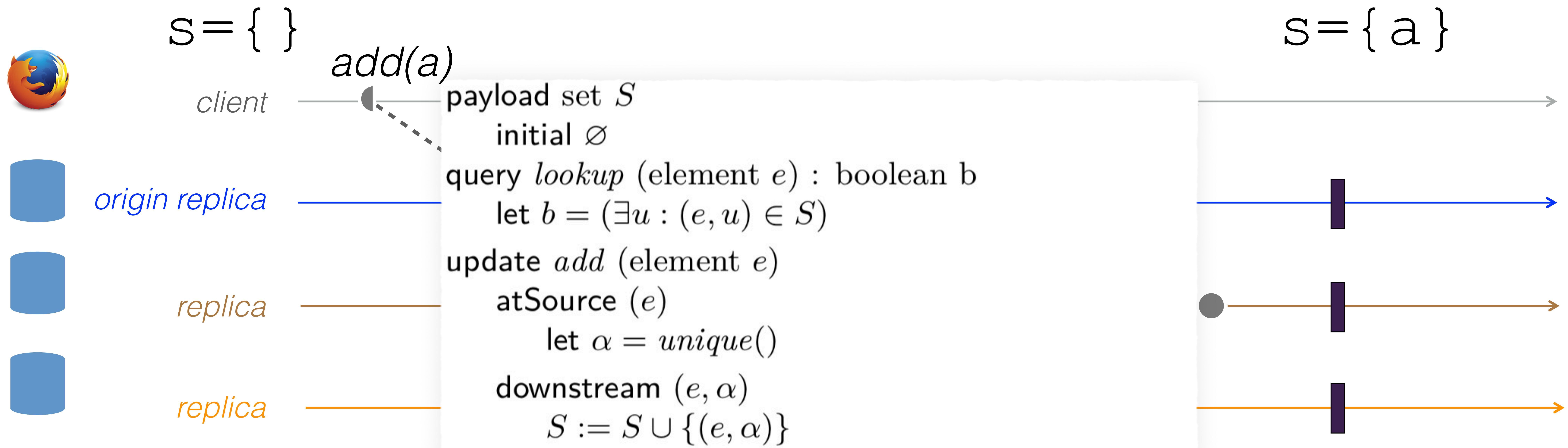
EXAMPLE: OBSERVED-REMOVE SET



payload set S
 initial $\varnothing$
query $lookup$ (element e) : boolean b
 let $b = (\exists u : (e, u) \in S)$
update add (element e)
 atSource (e)
  let $\alpha = unique()$
 downstream (e, α)
  $S := S \cup \{(e, \alpha)\}$
update $remove$ (element e)
 atSource (e)
  pre $lookup(e)$
  let $R = \{(e, u) | \exists u : (e, u) \in S\}$
 downstream (R)
  pre $\forall (e, u) \in R : add(e, u)$ has been delivered
  $S := S \setminus R$

[A Comprehensive Study of CRDT's (Shapiro et al. 2011)]

EXAMPLE: OBSERVED-REMOVE SET



payload set S
  initial $\varnothing$
query *lookup* (element e) : boolean b
  let $b = (\exists u : (e, u) \in S)$
update *add* (element e)
  atSource (e)
    let $\alpha = unique()$
  downstream (e, α)
    $S := S \cup \{(e, \alpha)\}$
update *remove* (element e)
  atSource (e)
    pre $lookup(e)$
    let $R = \{(e, u) | \exists u : (e, u) \in S\}$
  downstream (R)
    pre $\forall (e, u) \in R : add(e, u)$ has been delivered
    $S := S \setminus R$

[A Comprehensive Study of CRDT's (Shapiro et al. 2011)]

Anomalies of concurrent updates

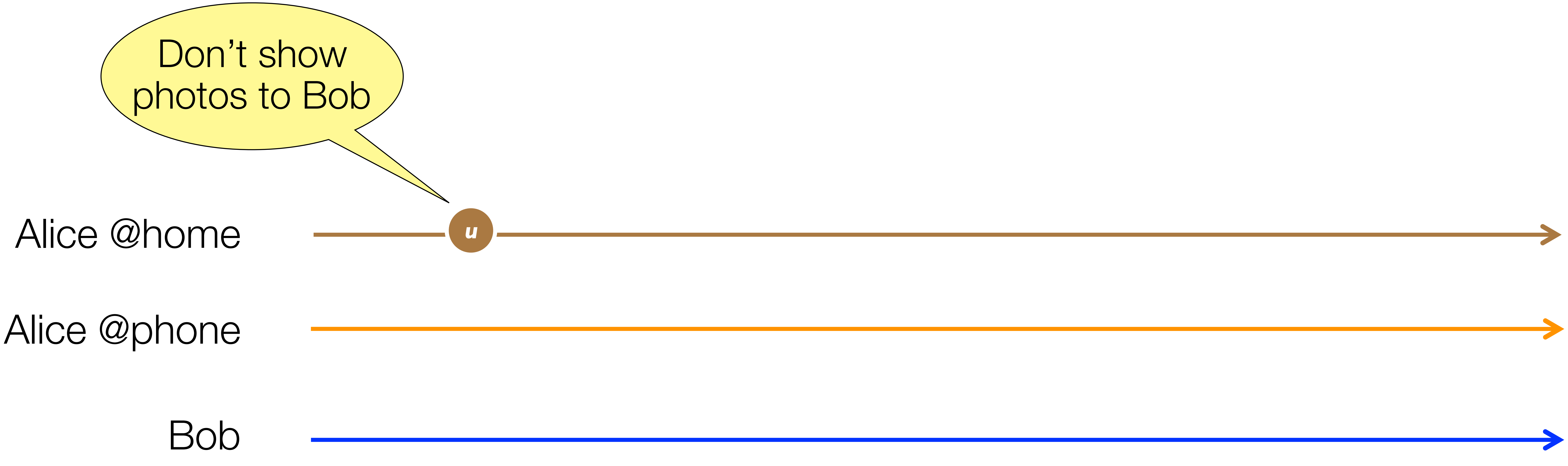
- Bank:
  - σ_{init} = 100€
  - Alice: *withdraw(20)* = { $\sigma \coloneqq 120$ }
  - Bob: *debit (60)* = { $\sigma \coloneqq 40$ }
  - σ = ???

Anomalies of concurrent updates

- Bank:
  - σ_{init} = 100€
  - Alice: *withdraw(20)* = { $\sigma \coloneqq$ 120 }
  - Bob: *debit (60)* = { $\sigma \coloneqq$ 40 }
  - σ = ???

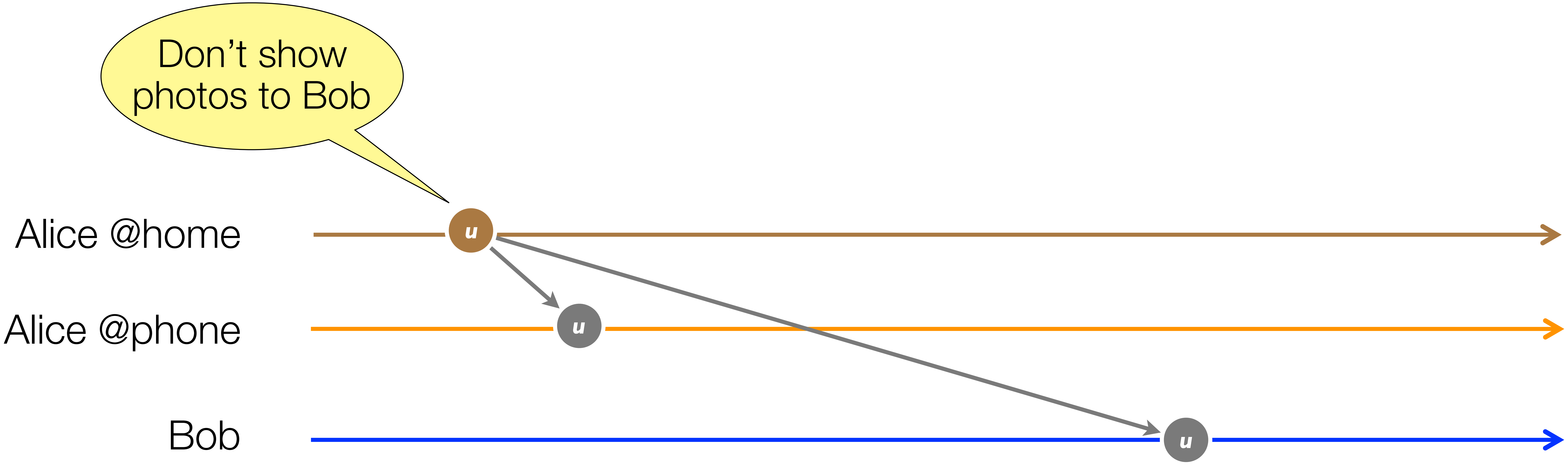
- File system:
  - σ_{init} = “/“
  - Alice: *mkdir (“/foo”); mkdir (“/foo/bar”)*
  - Bob: receives *mkdir (“/foo/bar”)*
  - σ = ???

Eventual Consistency



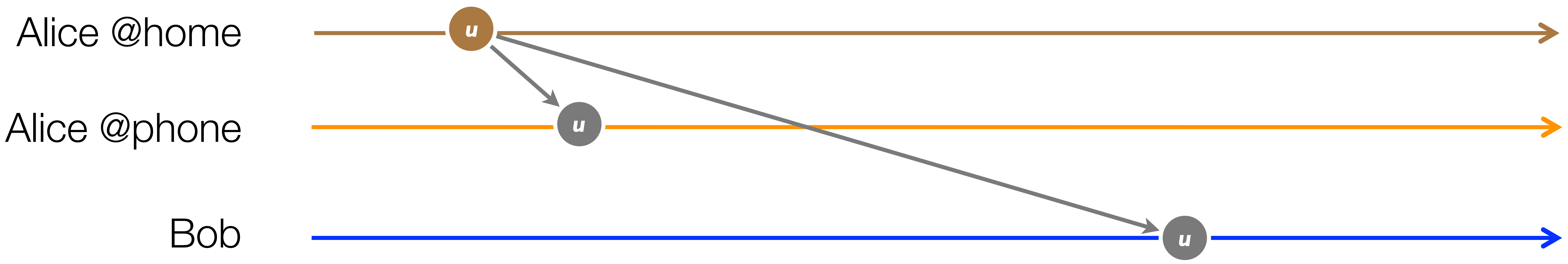
- access (Bob, photo) $\Longrightarrow$ ACL (Bob, photo)
- v observed effects of u $\Longrightarrow$ v should be delivered after u
- Available: doesn't slow down sender

Eventual Consistency



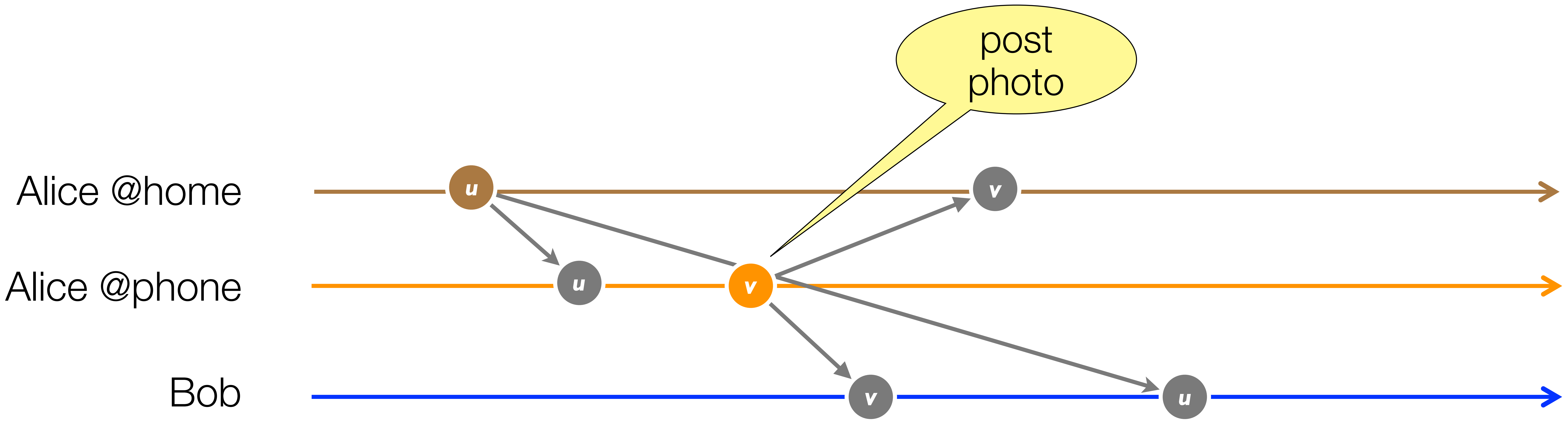
- access (Bob, photo) $\Longrightarrow$ ACL (Bob, photo)
- v observed effects of u $\Longrightarrow$ v should be delivered after u
- Available: doesn't slow down sender

Eventual Consistency



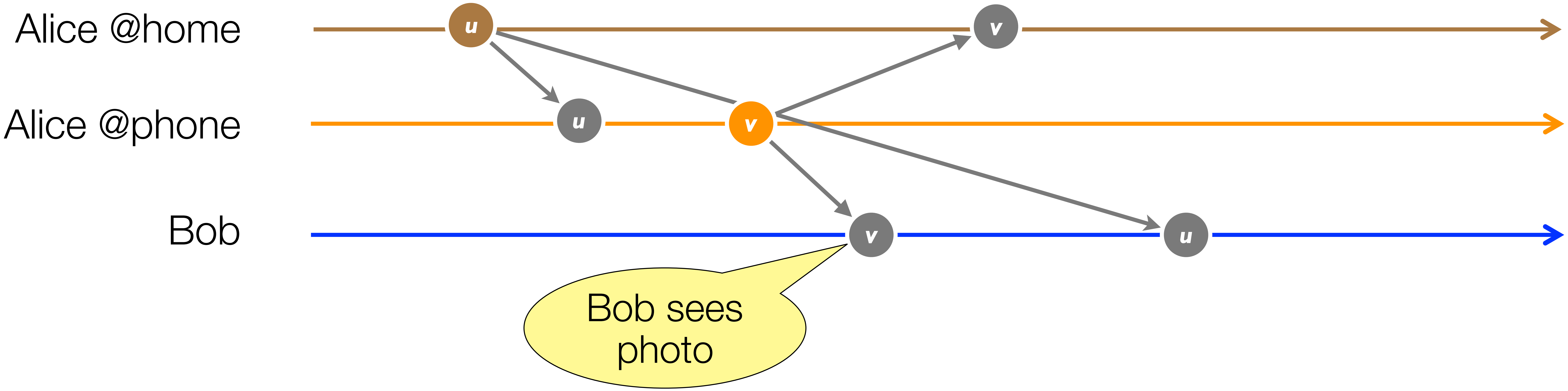
- access (Bob, photo) $\Longrightarrow$ ACL (Bob, photo)
- v observed effects of u $\Longrightarrow$ v should be delivered after u
- Available: doesn't slow down sender

Eventual Consistency



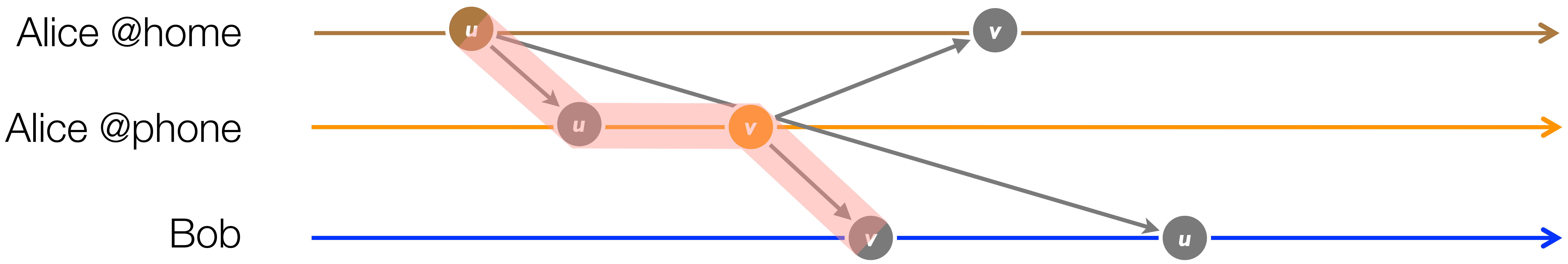
- access (Bob, photo) $\Longrightarrow$ ACL (Bob, photo)
- v observed effects of u $\Longrightarrow$ v should be delivered after u
- Available: doesn't slow down sender

Eventual Consistency



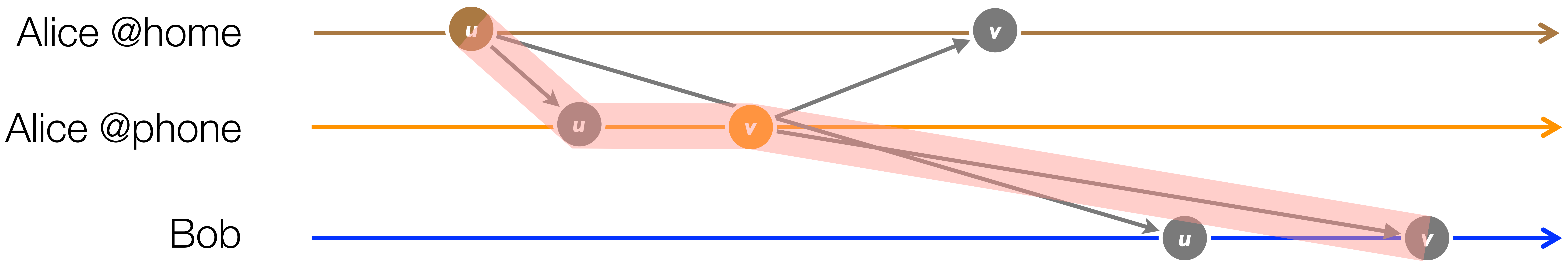
- access (Bob, photo) $\Longrightarrow$ ACL (Bob, photo)
- v observed effects of u $\Longrightarrow$ v should be delivered after u
- Available: doesn't slow down sender

Eventual Consistency



- access (Bob, photo) $\Longrightarrow$ ACL (Bob, photo)
- v observed effects of u $\Longrightarrow$ v should be delivered after u
- Available: doesn't slow down sender

Eventual Consistency



- access (Bob, photo) $\Longrightarrow$ ACL (Bob, photo)
- v observed effects of u $\Longrightarrow$ v should be delivered after u
- Available: doesn't slow down sender

Commutative Replicated Data Types

- Data type
  - Encapsulates state

Commutative Replicated Data Types

- Data type
  - Encapsulates state
- Replicated
  - At multiple nodes
  - Semantically a single object

COMMUTATIVE REPLICATED DATA TYPES

- Data type
  - Encapsulates state
- Replicated
  - At multiple nodes
  - Semantically a single object
- Available
  - Update origin replica without coordination
  - Convergence guaranteed by design
  - Decentralized

COMMUTATIVE REPLICATED DATA TYPES

- Data type
  - Encapsulates state
- Replicated
  - At multiple nodes
  - Semantically a single object
- Available
  - Update origin replica without coordination
  - Convergence guaranteed by design
  - Decentralized


INRIA
ROCQUENCOURT

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

Thème COM — Systèmes communicants
Projet Regal

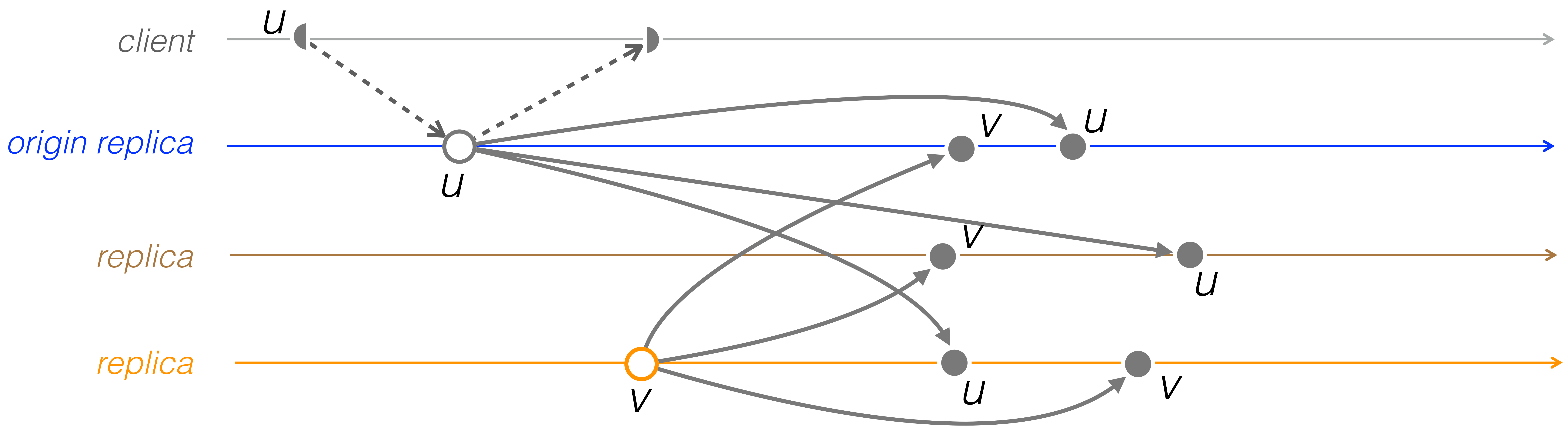
http://bit.ly/1PBC4zc

Operation-based CRDTs

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

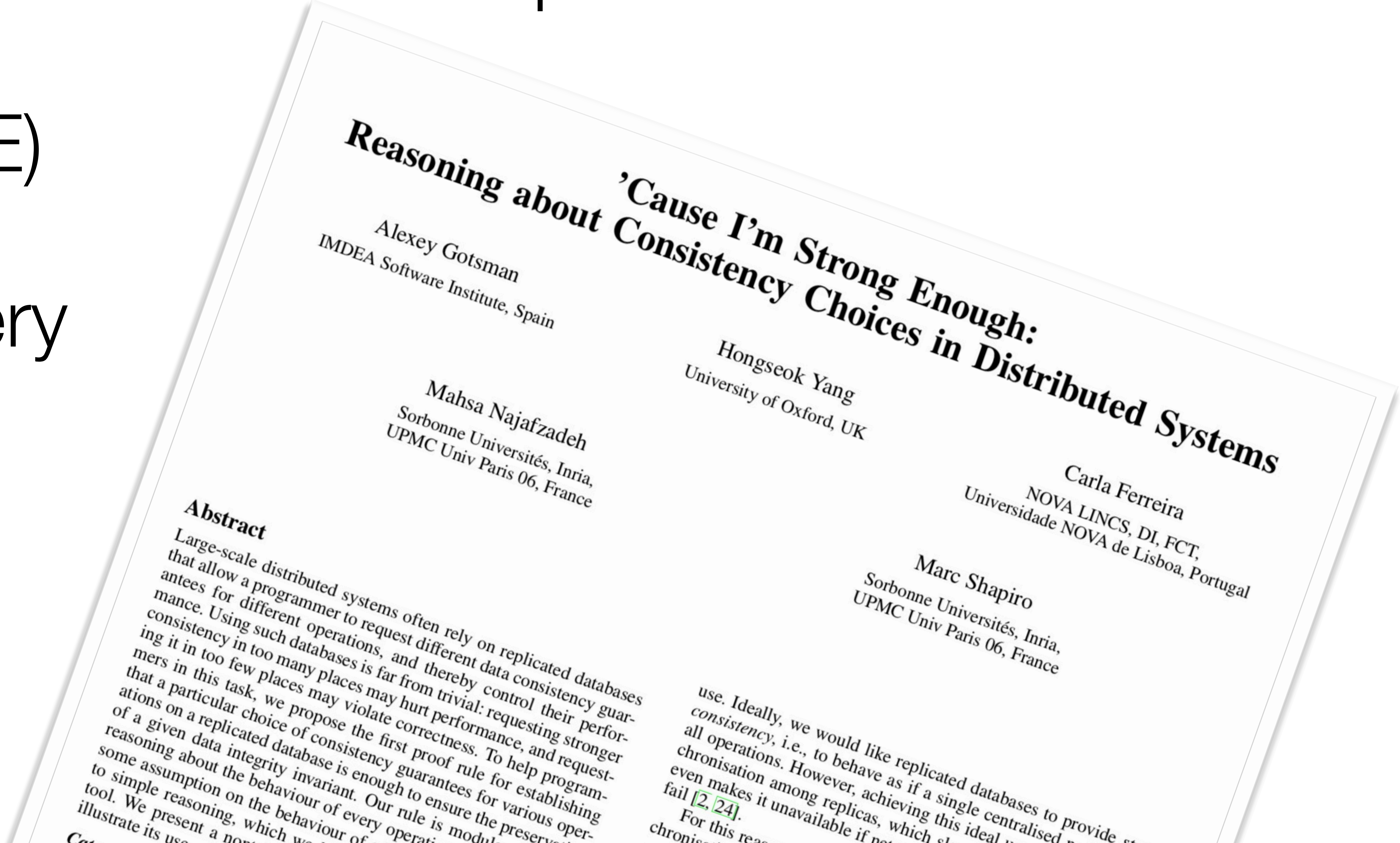
Operation-based CRDTs

- 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 preservation of a given data integrity invariant. Our rule is modular ... reasoning about the behaviour of every operation ... some assumption on the behaviour of ... to simple reasoning, which ... tool. We present a ... 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].

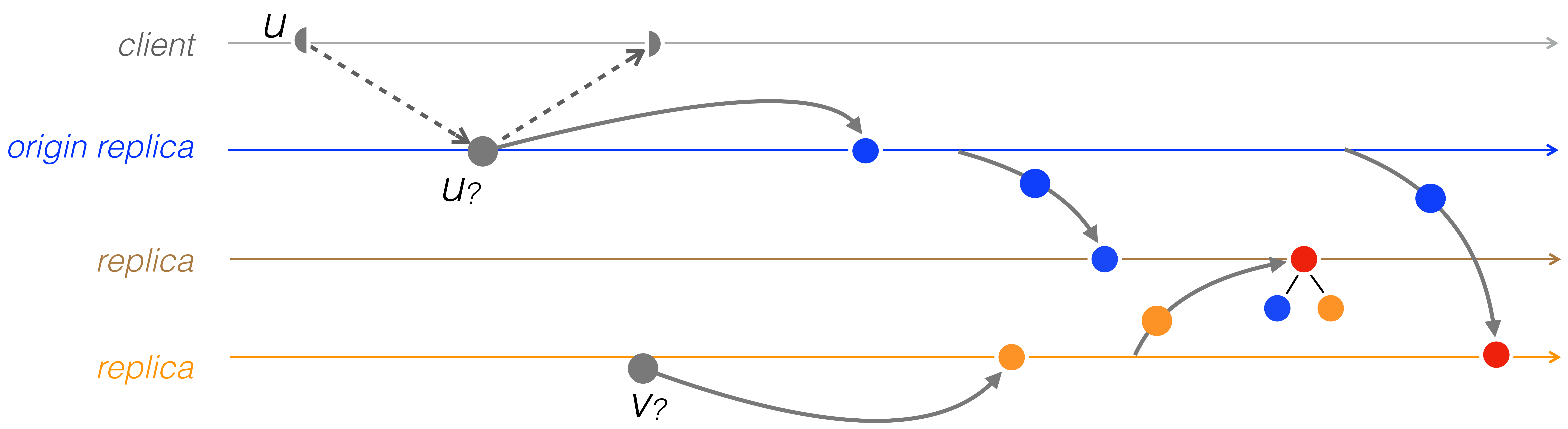
For this reas... chronis...

State-based CRDTs

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

State-based CRDTs

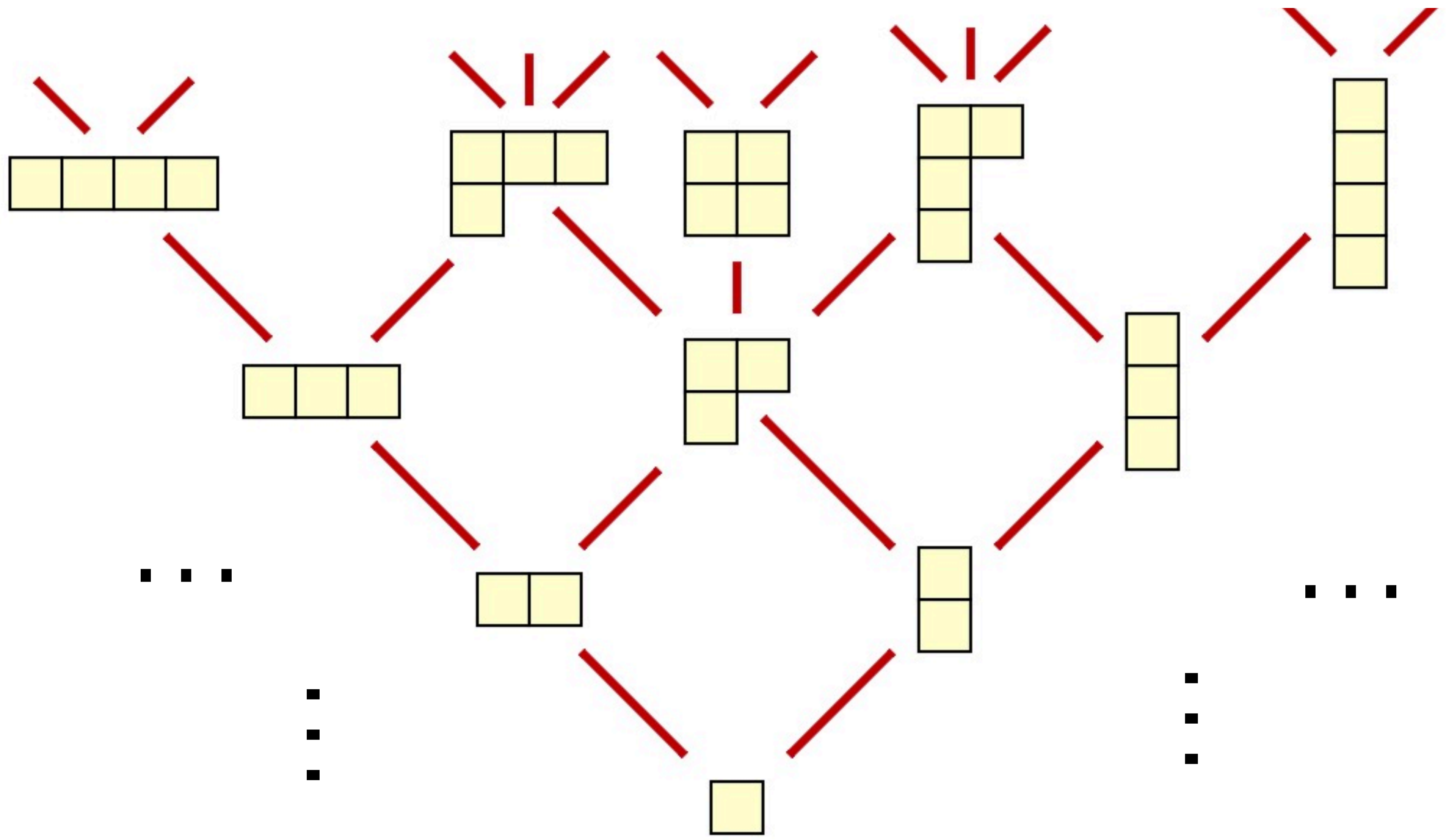
- State-based CRDTs
  - Propagation of states (instead of operations)



STATE-BASED CRDTS

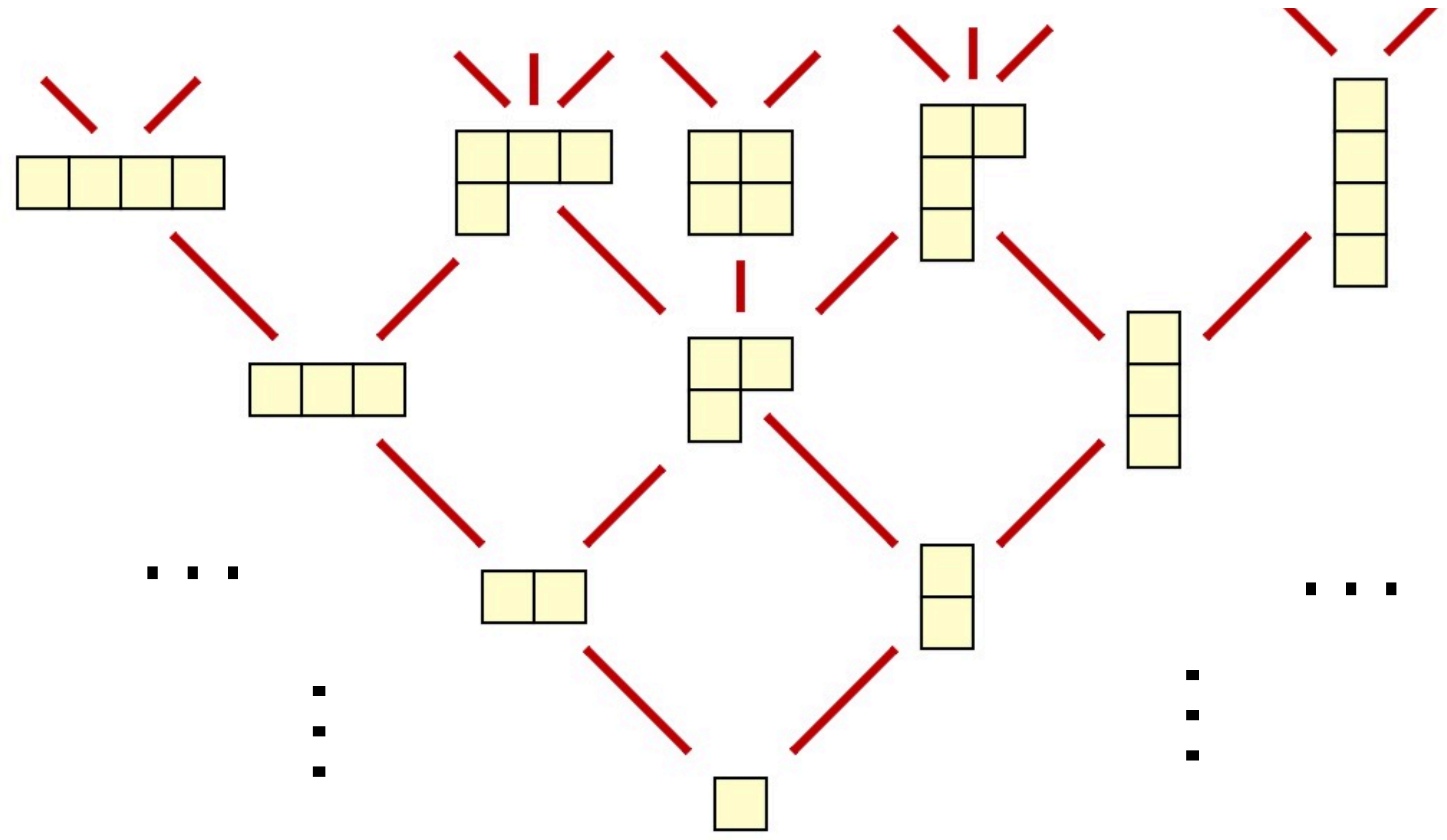
- State-based CRDTs
  - Propagation of states (instead of operations)
- States are **`merged`** on receive
  - Convergence: states resulting from *concurrent* operations result deterministically on a single state
  - No delivery assumptions

STATE-BASED CRDTS



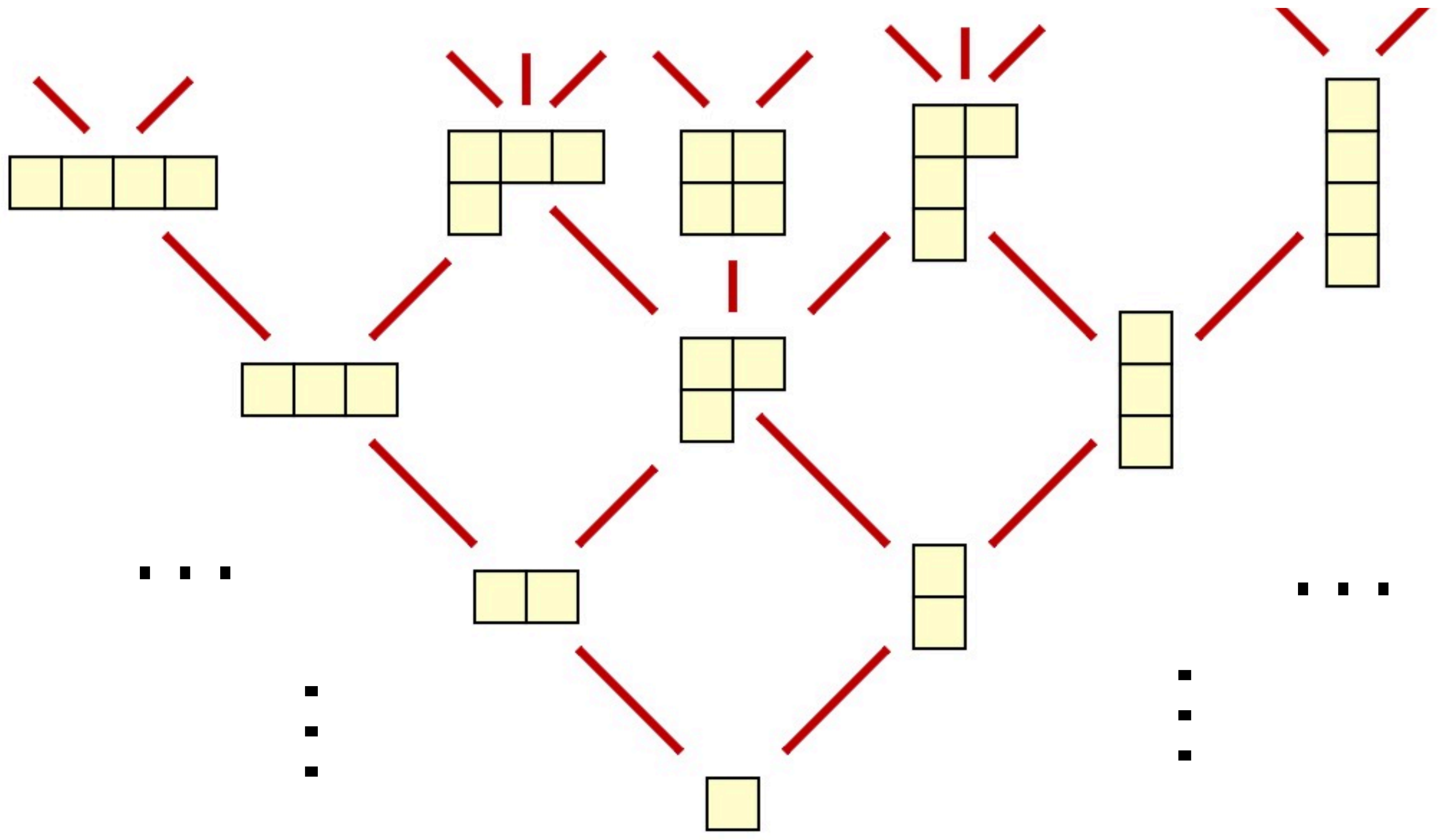
STATE-BASED CRDTS

- State is a (join semi-)Lattice

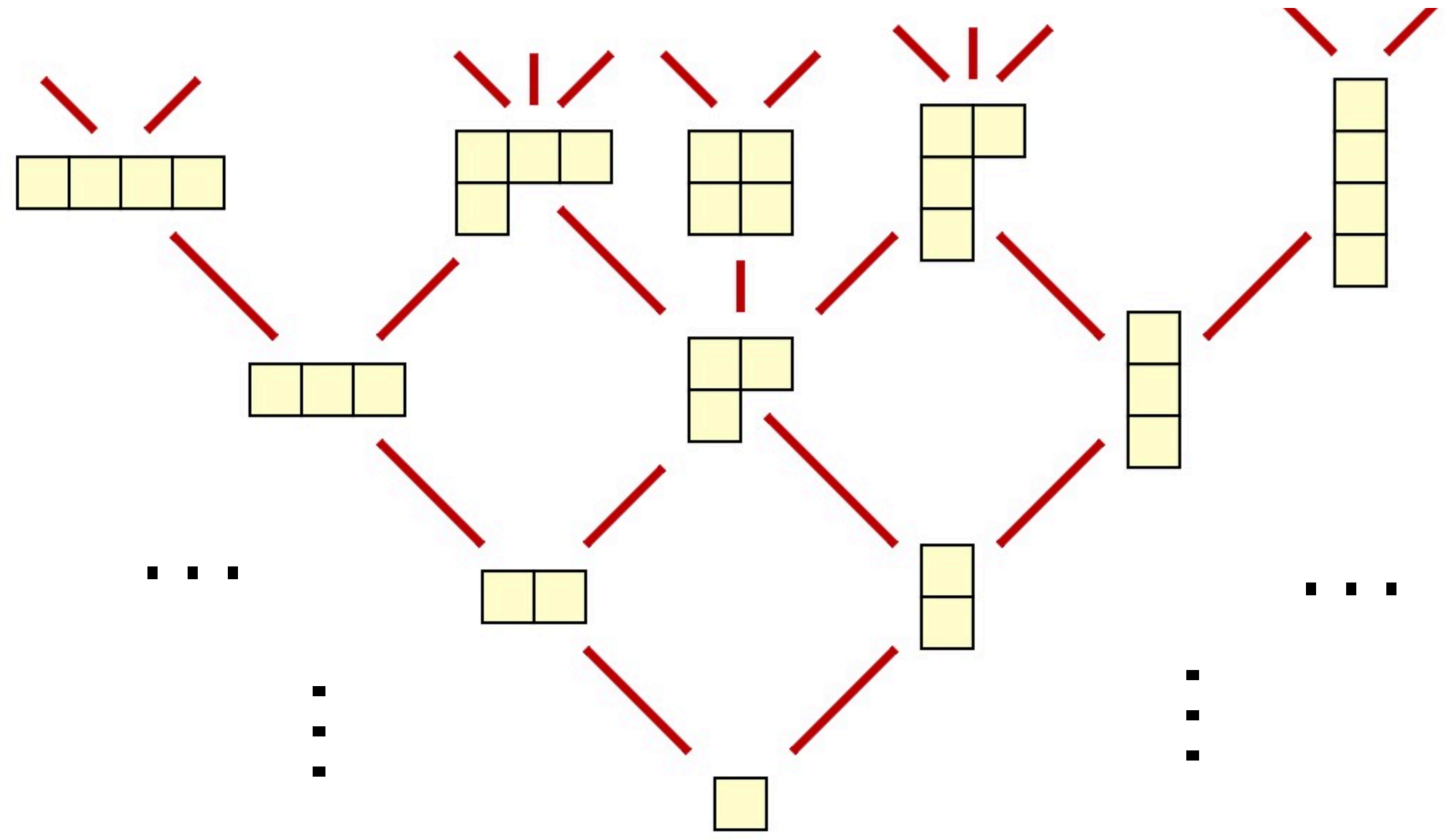


STATE-BASED CRDTS

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

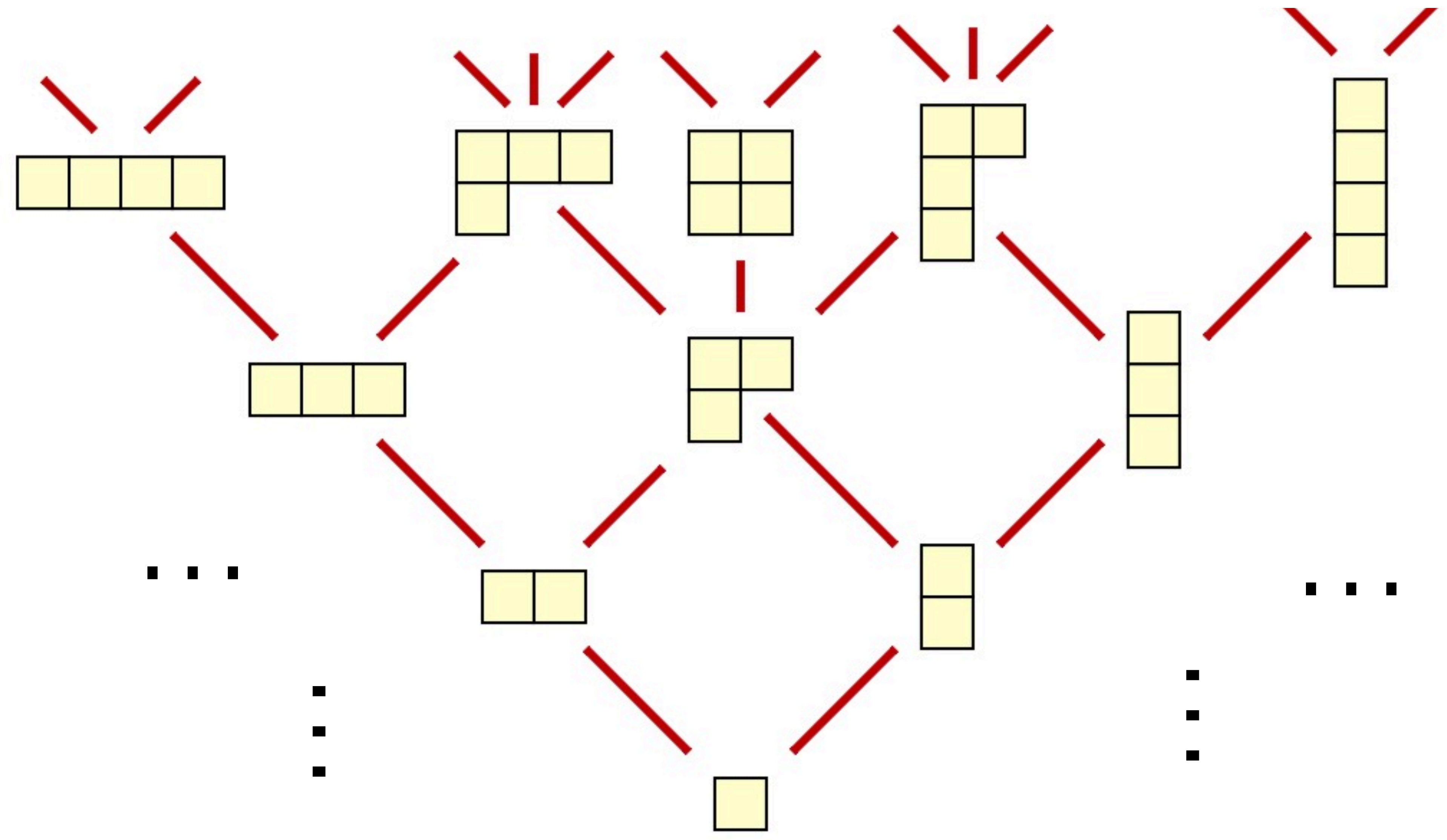


STATE-BASED CRDTS



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

State-based CRDTs



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

BOUNDED COUNTER

- N Replicas
- R matrix of positive counts
- U vector of negative counts

10	2	2	1
3	6	1	0
5	0	4	2
1	0	2	1

5
2
1
0

[Valter Balegas, 2016]

Bounded Counter

- N Replicas
- R matrix of positive counts
- U vector of negative counts

10	2	2	1
3	6	1	0
5	0	4	2
1	0	2	1

5
2
1
0

Total: $\sum_i R[i][i] - \sum_i U[i]$

[Valter Balegas, 2016]

BOUNDED COUNTER

- N Replicas
- R matrix of positive counts
- U vector of negative counts
- Invariant: 0 ≤ Total

10	2	2	1
3	6	1	0
5	0	4	2
1	0	2	1

5
2
1
0

Total: $\sum_i R[i][i] - \sum_i U[i]$

[Valter Balegas, 2016]

BOUNDED COUNTER

- N Replicas
- R matrix of positive counts
- U vector of negative counts
- Invariant: 0 ≤ Total
- **`increment`** @i

		i		j		
	10	2	2	1		5
i	3	6	1	0		2
	5	0	4	2		1
j	1	0	2	1		0

Total: $\sum_i R[i][i] - \sum_i U[i]$

[Valter Balegas, 2016]

Bounded Counter

- N Replicas
- R matrix of positive counts
- U vector of negative counts
- Invariant: $0 \leq$ Total
- **`increment`** @i
- **`decrement`** @i

		i		j		
	10	2	2	1		5
i	3	6	1	0		2
	5	0	4	2		1
j	1	0	2	1		0

Total: $\sum_i R[i][i] - \sum_i U[i]$

[Valter Balegas, 2016]

BOUNDED COUNTER

- N Replicas
- R matrix of positive counts
- U vector of negative counts
- Invariant: $0 \leq$ Total
- **`increment`** @i
- **`decrement`** @i

		i		j		
	10	2	2	1		5
i	3	6	1	0		2
	5	0	4	2		1
j	1	0	2	1		0

Total: $\sum_i R[i][i] - \sum_i U[i]$

Rights @i: $R[i][i] + \sum_j R[j][i] - \sum_{j \neq i} R[i][j] - U[i]$

[Valter Balegas, 2016]

BOUNDED COUNTER

- N Replicas
- R matrix of positive counts
- U vector of negative counts
- Invariant: 0 ≤ Total
- **`increment`** @i
- **`decrement`** @i

		i		j		U
	10	2	2	1		5
i	3	6	1	0		2
	5	0	4	2		1
j	1	0	2	1		0

Total: $\sum_i R[i][i] - \sum_i U[i]$ 13

Rights @i: $R[i][i] + \sum_j R[j][i] - \sum_{j \neq i} R[i][j] - U[i]$ @i 2

[Valter Balegas, 2016]

Bounded Counter

- N Replicas
- R matrix of positive counts
- U vector of negative counts
- Invariant: $0 \leq$ Total
- **`increment`** @i
- **`decrement`** @i
- **`transfer`** @i → j

		i		j		
	10	2	2	1		5
i	3	6	1	0		2
	5	0	4	2		1
j	1	0	2	1		0

Total: $\sum_i R[i][i] - \sum_i U[i]$ 13

Rights @i: $R[i][i] + \sum_j R[j][i] - \sum_{j \neq i} R[i][j] - U[i]$ @i 2

[Valter Balegas, 2016]

BOUNDED COUNTER

- N Replicas
- R matrix of positive counts
- U vector of negative counts
- Invariant: 0 ≤ Total
- **`increment`** @i
- **`decrement`** @i
- **`transfer`** @i → j

		i		j		
	10	2	2	1		5
i	3	6	1	0		2
	5	0	4	2		1
j	1	0	2	1		0

$$\texttt{merge}((M_0, V_0), (M_1, V_1)) = (\texttt{max}(M_0, M_1), \texttt{max}(V_0, V_1))$$

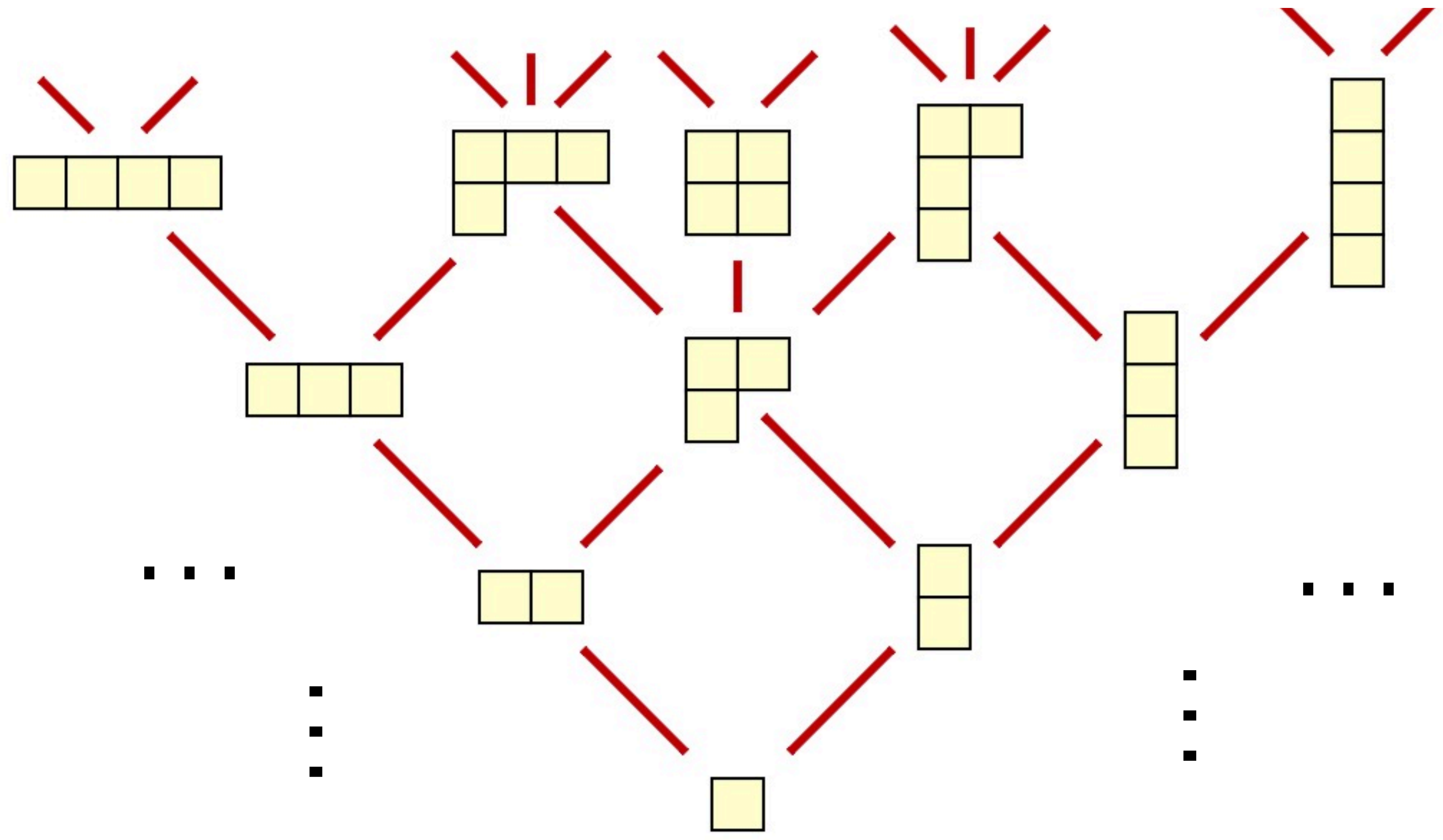
[Valter Balegas, 2016]

Checking Invariants

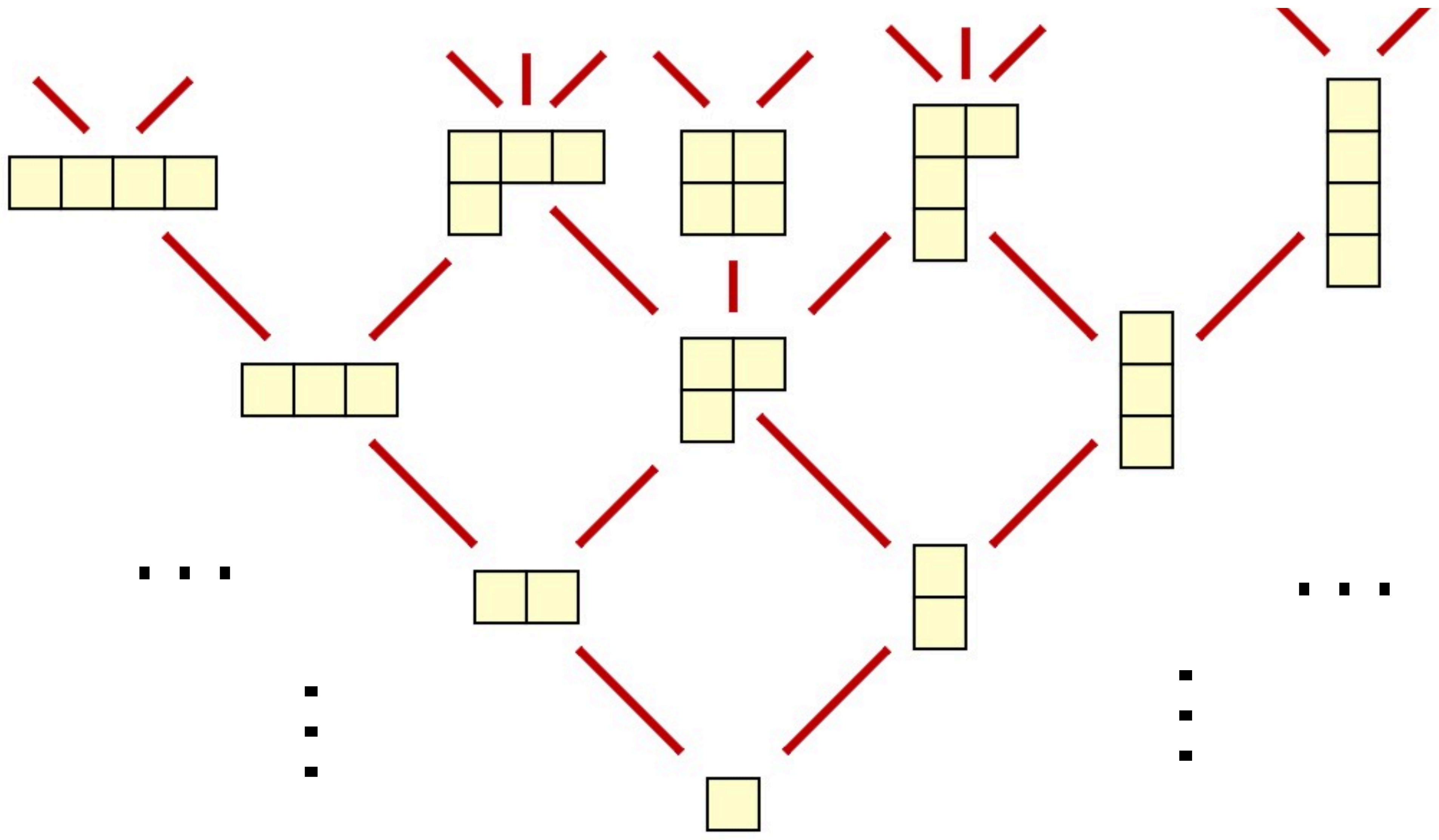
State-Based CRDTs

INVARIANTS FOR SB-CRDTS

- CRDT (lattice) constraints



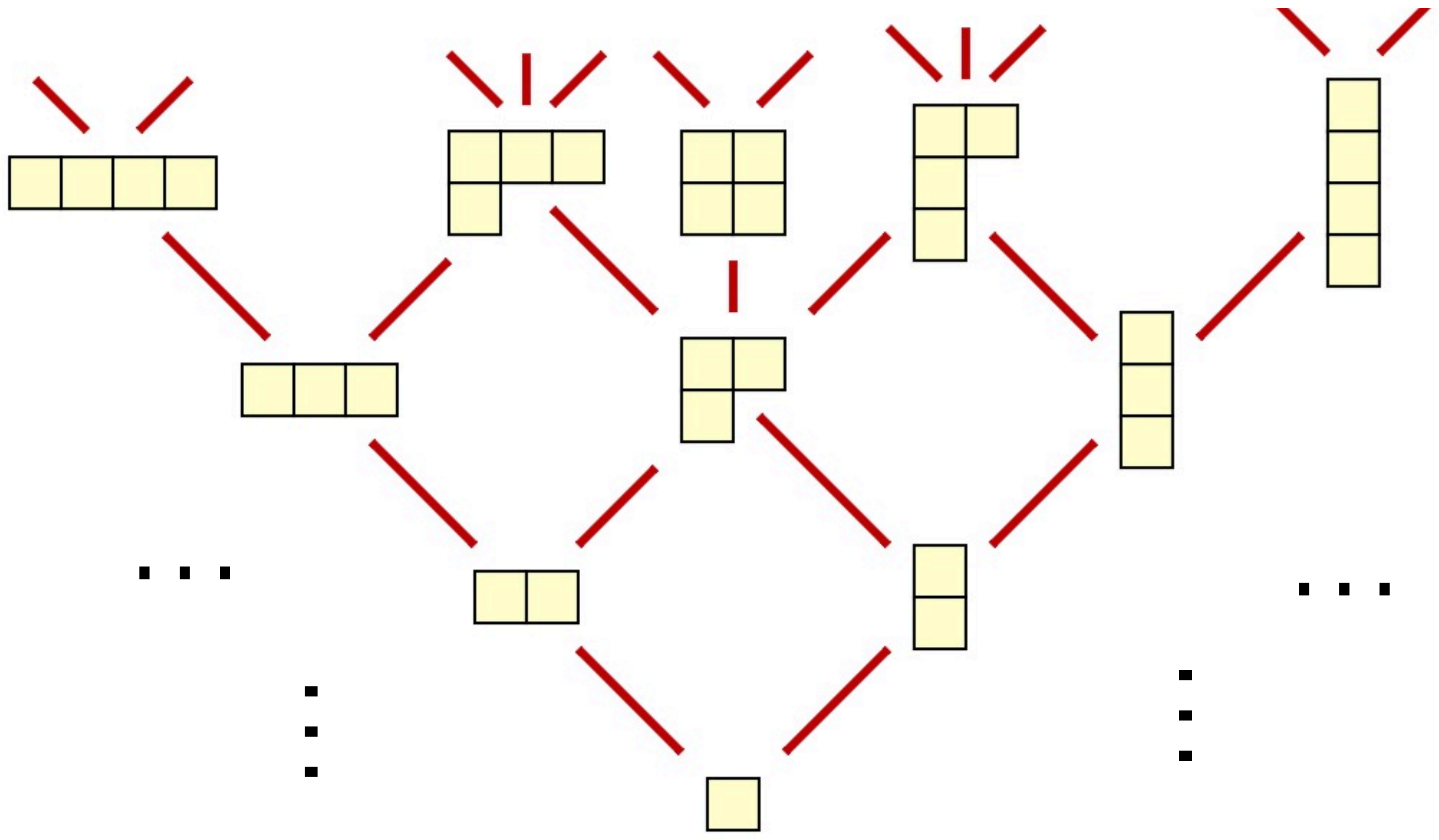
INVARIANTS FOR SB-CRDTS



- 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

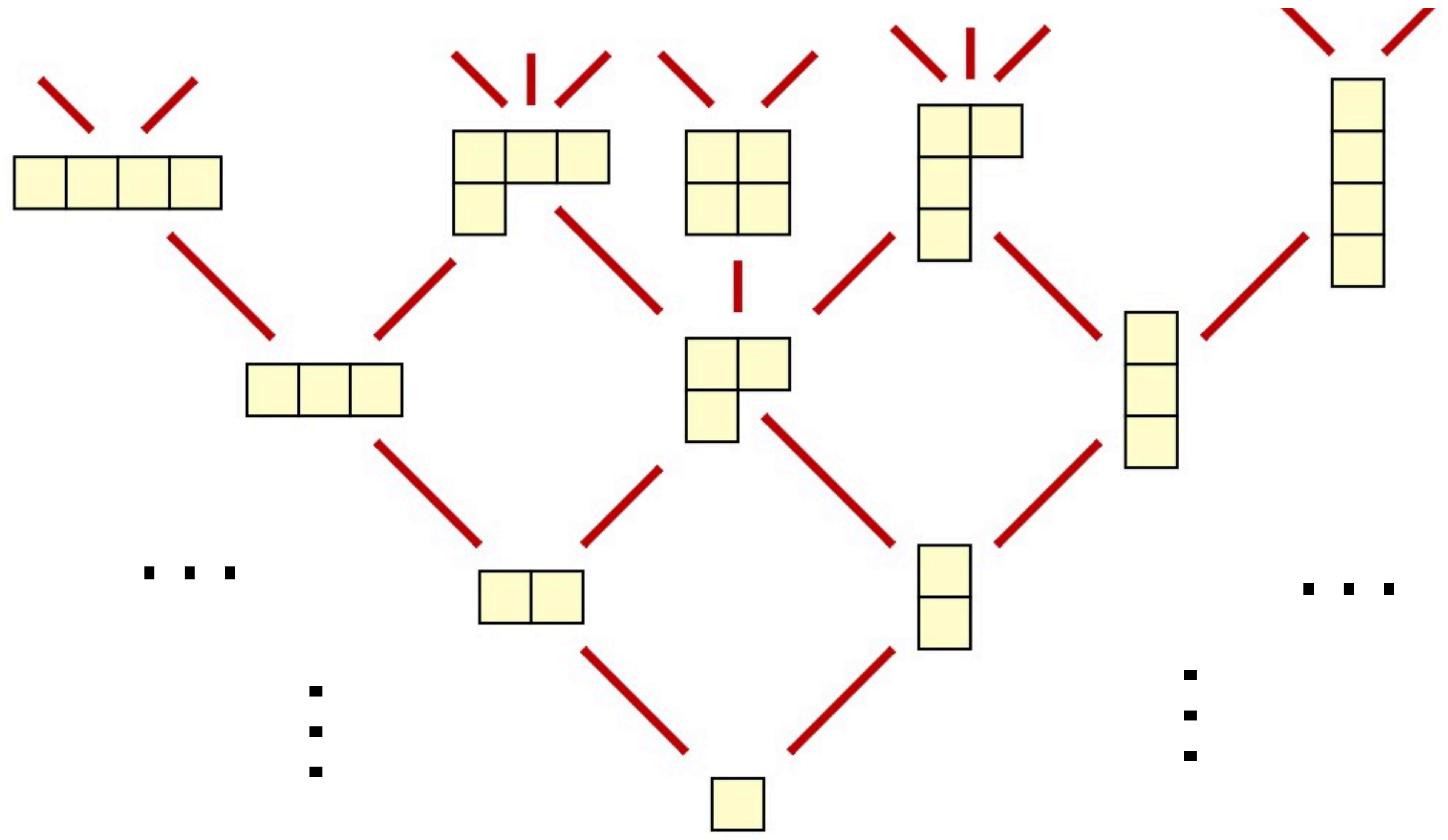
$$\forall\ \mathsf{op}, \sigma, \sigma',\ \ \sigma \vDash \mathsf{Pre}_{\mathsf{op}}\ \wedge\ (\sigma, \sigma') \in [\![\mathsf{op}]\!]\ \Rightarrow \sigma \sqsubseteq \sigma'$$

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

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

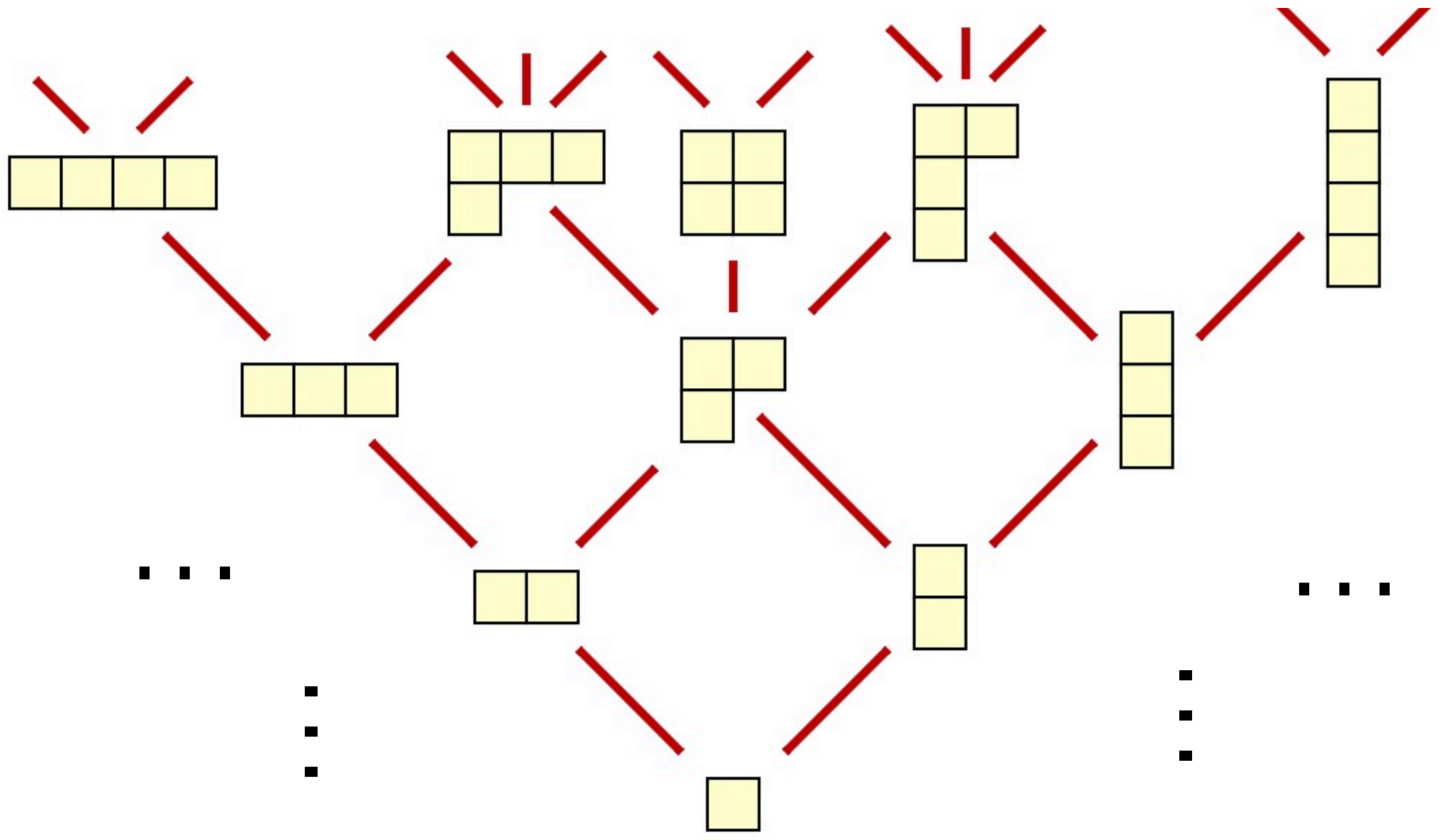
Invariants for SB-CRDTs

- Invariant constraints



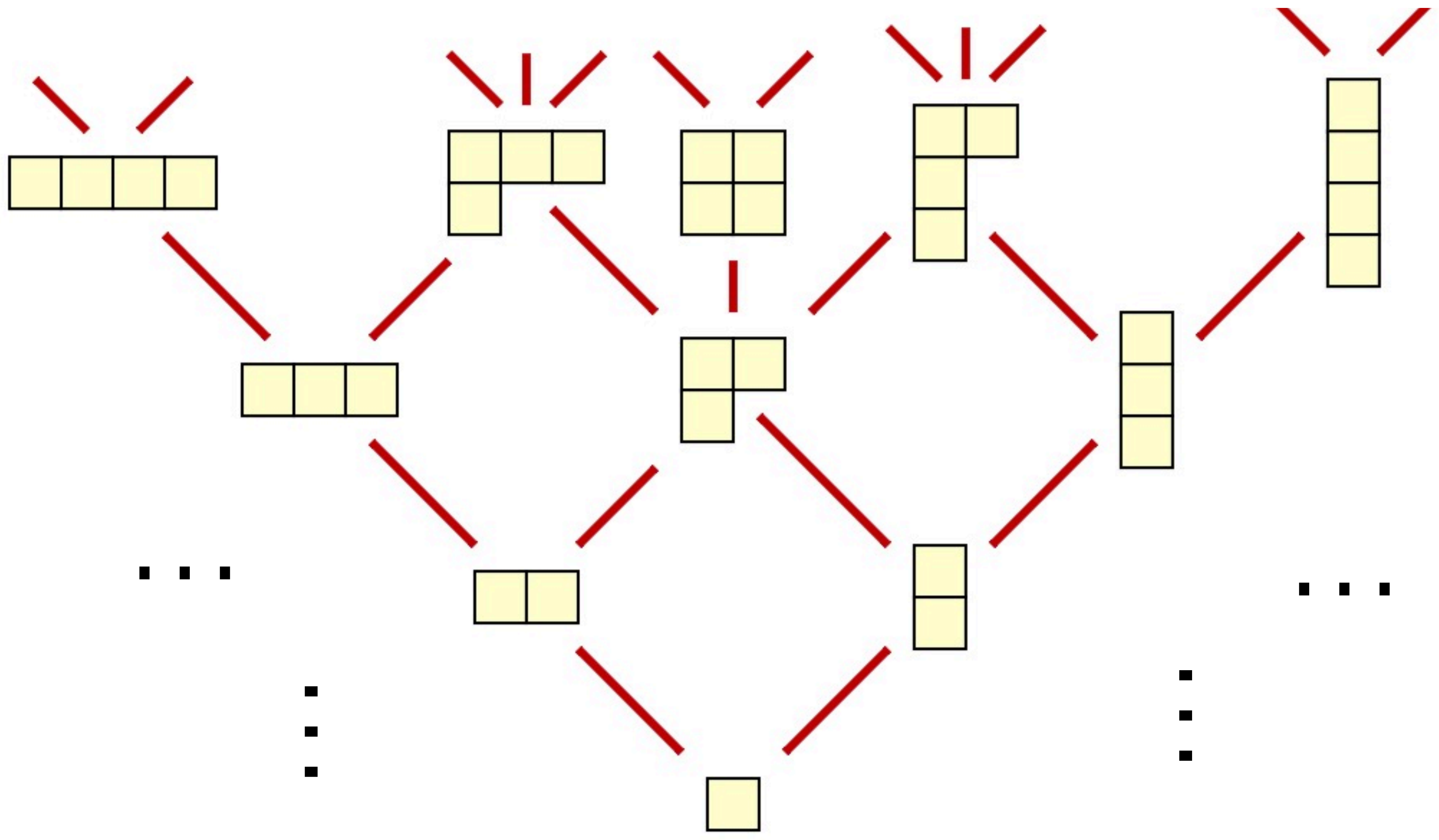
INVARIANTS FOR SB-CRDTS

- 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



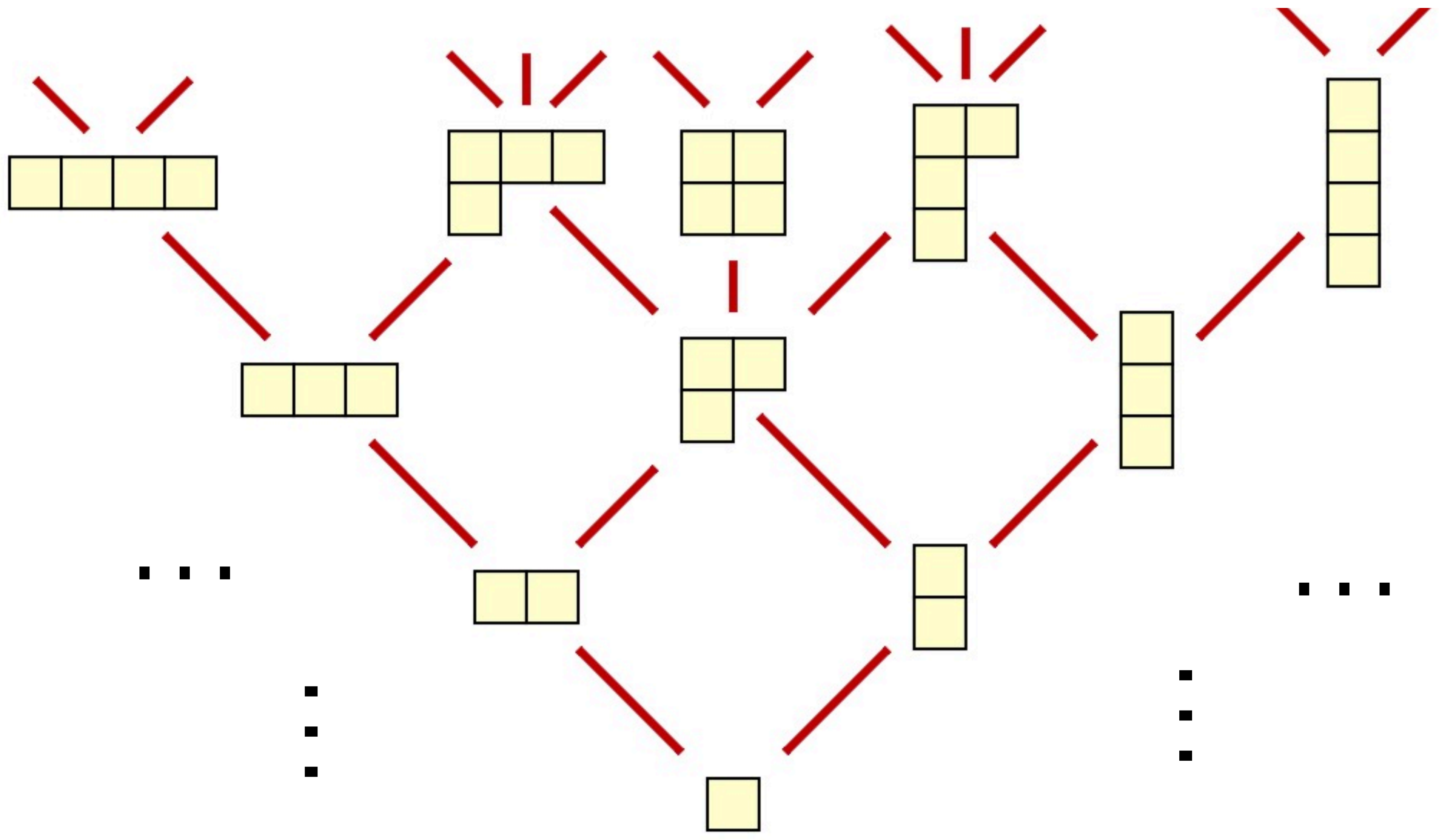
- Invariant constraints
  - Operations preserve the invariant

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

  - **`merge`** preserves the invariant

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

Invariants for SB-CRDTs



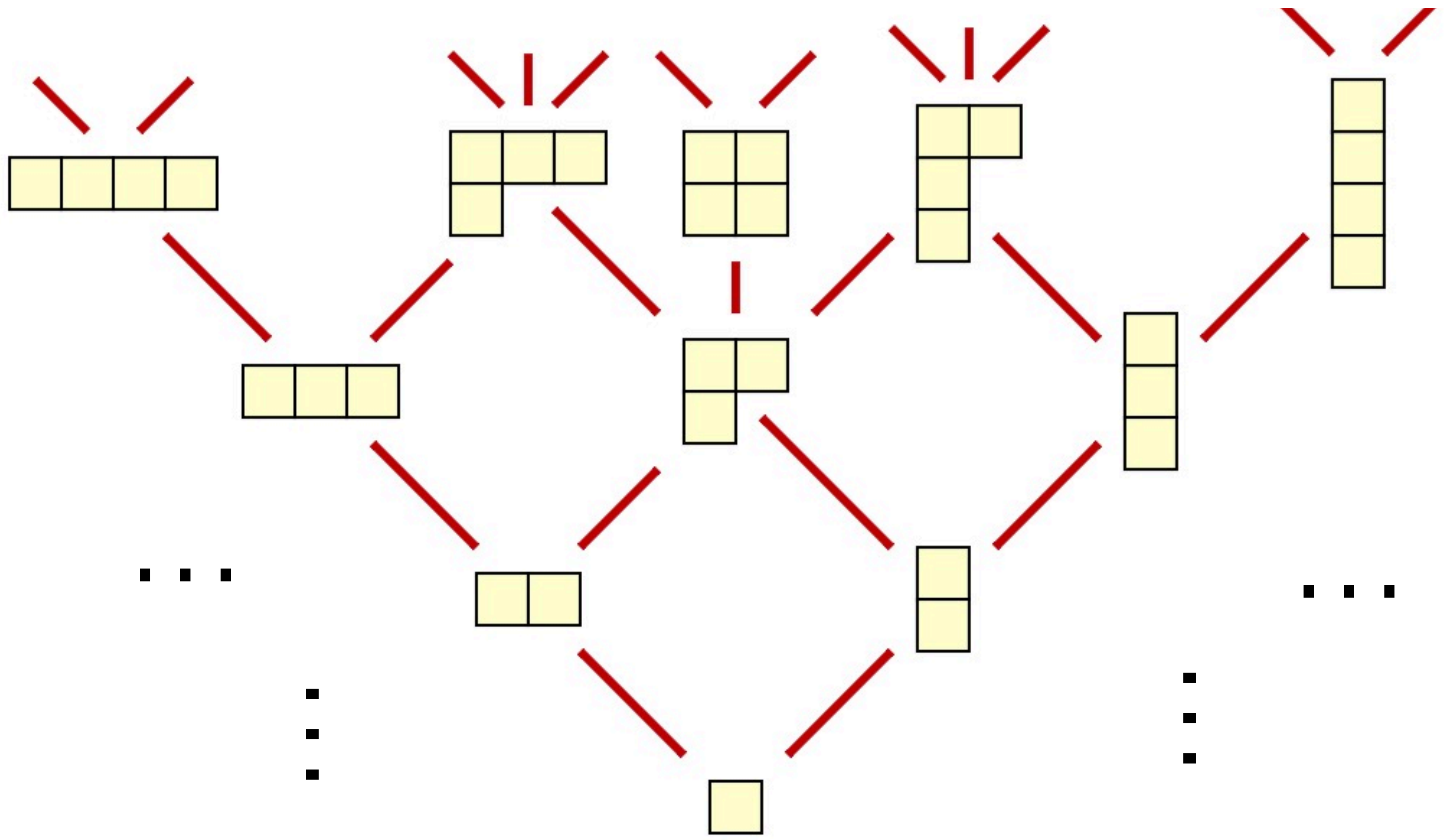
- Invariant constraints
  - Operations preserve the invariant

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

  - **`merge`** preserves the invariant

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

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

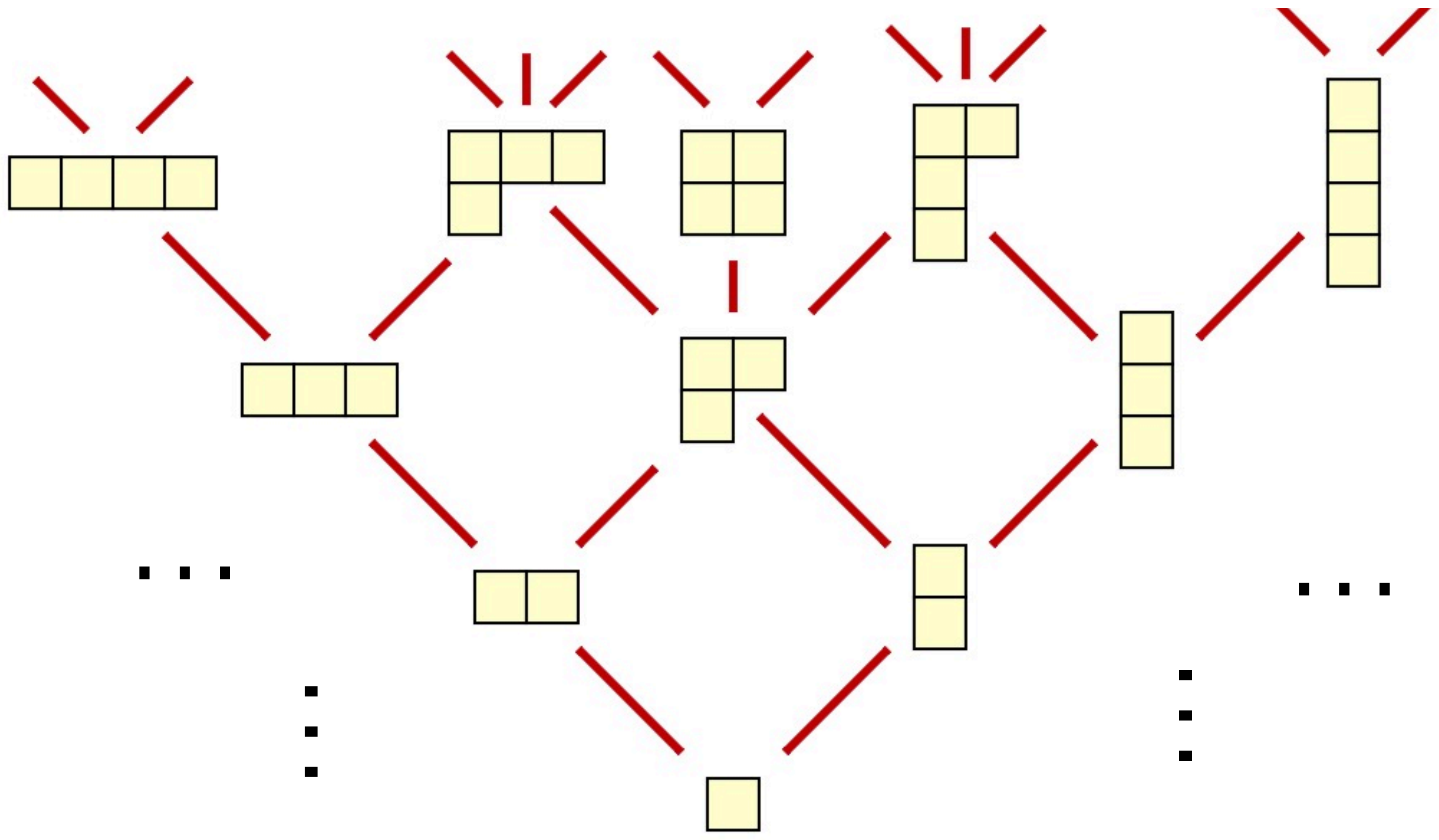
$$\forall\ \sigma, \sigma', \sigma'',\ (\sigma, \sigma'') \models \mathsf{Pre}_{\mathtt{merge}}\ \wedge\ \mathtt{merge}(\sigma, \sigma'') = \sigma'\ \Rightarrow \sigma'\ \models\ \mathsf{Inv}$$

Eg: Bounded Counter

$$\mathsf{Total}(\sigma) \geq 0 \wedge \mathsf{Total}(\sigma') \geq 0$$

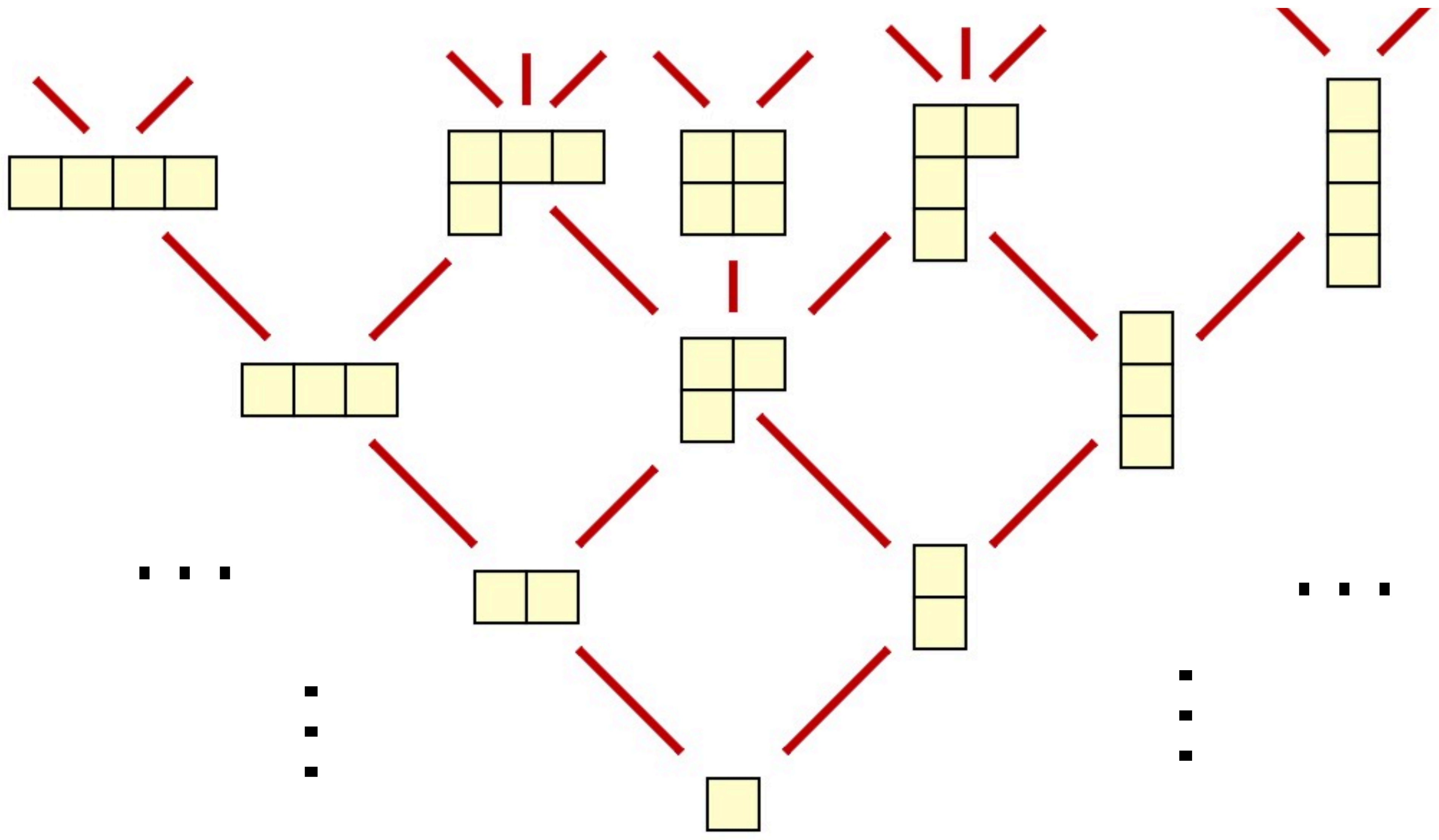
INVARIANTS FOR SB-CRDTS

- Invariant constraints



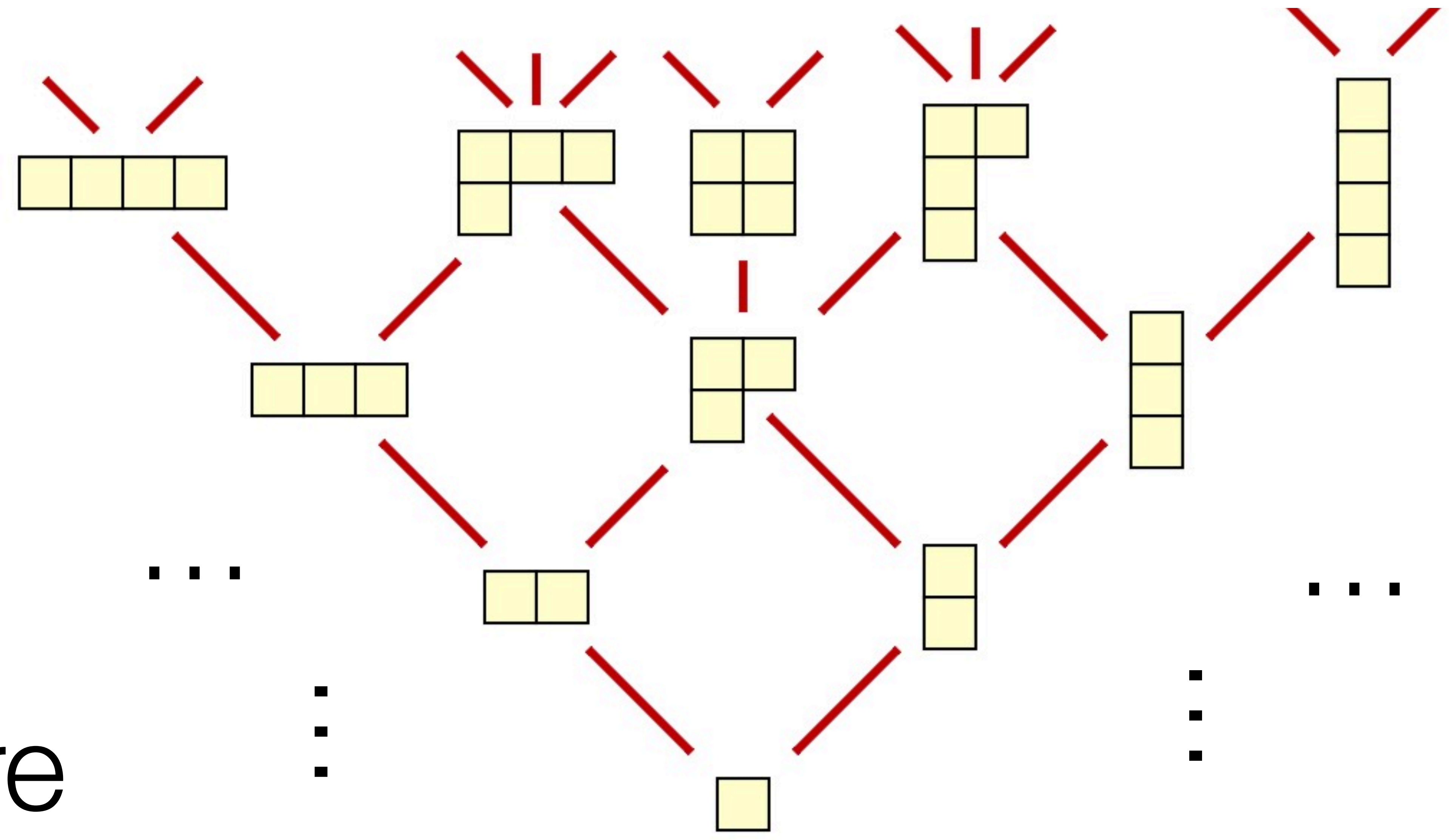
INVARIANTS FOR SB-CRDTS

- Invariant constraints
  - **`merge`** can execute at any time



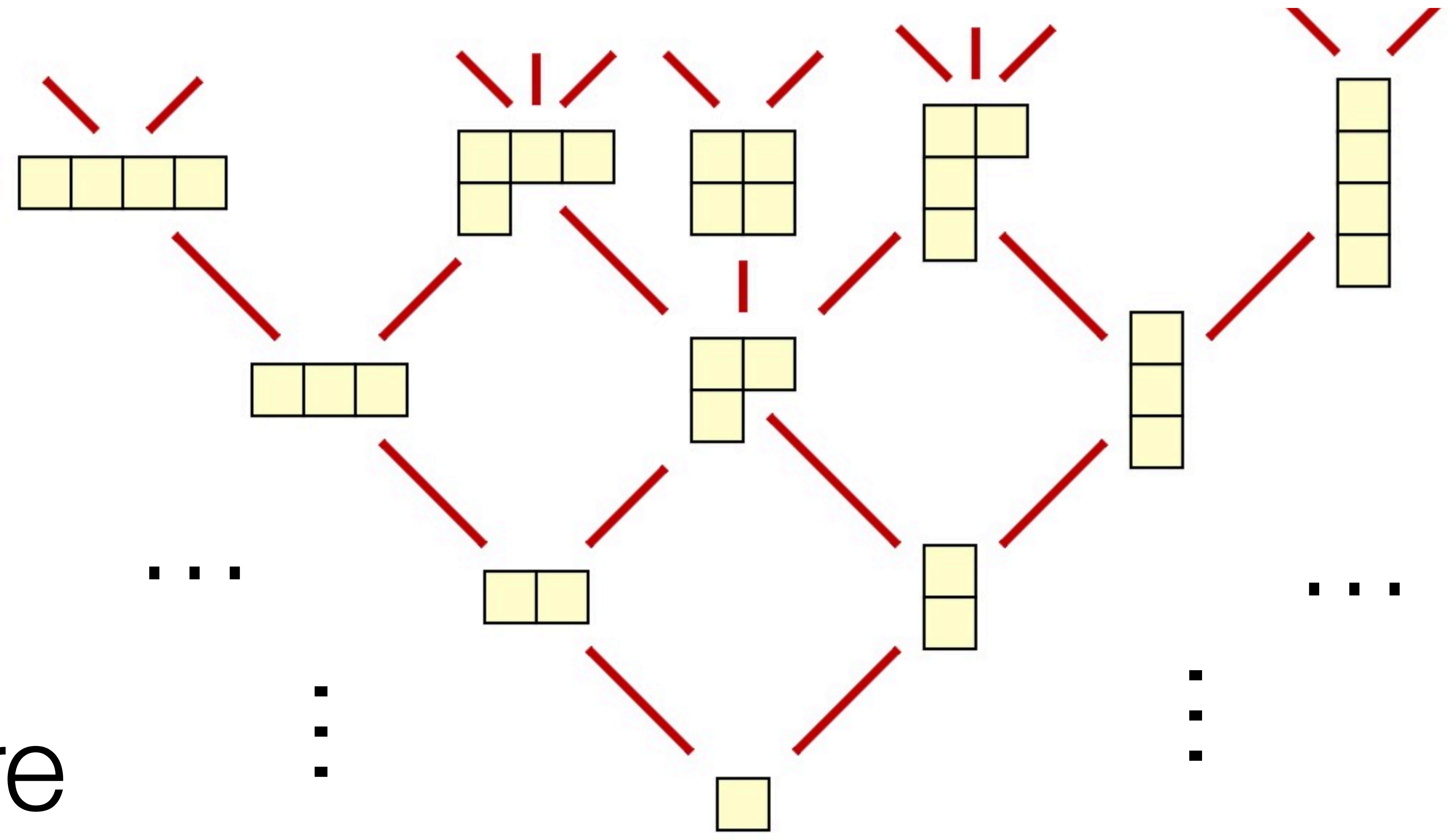
INVARIANTS FOR SB-CRDTS

- Invariant constraints
  - **`merge`** can execute at any time
  - Operations preserve the **`merge`** Pre



$$\forall\ \mathsf{op}, \sigma, \sigma', \sigma'',\ \left(\begin{array}{l} \sigma \vDash \mathsf{Pre}_{\mathsf{op}}\ \wedge \\ (\sigma, \sigma'') \vDash \mathsf{Pre}_{\mathtt{merge}}\ \wedge \\ (\sigma, \sigma') \in [\![\mathsf{op}]\!] \end{array}\right) \Rightarrow (\sigma', \sigma'')\ \vDash\ \mathsf{Pre}_{\mathtt{merge}}$$

INVARIANTS FOR SB-CRDTS



- Invariant constraints
  - **`merge`** can execute at any time
  - Operations preserve the **`merge`** Pre

$$\forall\ \mathsf{op}, \sigma, \sigma', \sigma'',\ \left(\begin{array}{l} \sigma \models \mathsf{Pre}_{\mathtt{op}}\ \wedge \\ (\sigma, \sigma'') \models \mathsf{Pre}_{\mathtt{merge}}\ \wedge \\ (\sigma, \sigma') \in [\![\mathsf{op}]\!] \end{array}\right) \Rightarrow (\sigma', \sigma'')\ \models\ \mathsf{Pre}_{\mathtt{merge}}$$

  - **`merge`** Pre is an invariant

Example: Token Implementation

(Eg.) Mutual Exclusion CRDT

(Eg.) Mutual Exclusion CRDT

- At most one replica has rights to the mutual exclusion token

(EG.) MUTUAL EXCLUSION CRDT

- At most one replica has rights to the mutual exclusion token
  - An array V[R] to indicate who has the “lock”

(Eg.) Mutual Exclusion CRDT

- At most one replica has rights to the mutual exclusion token
  - An array V[R] to indicate who has the "lock"
  - Escrow
    - Owner transfer the token to the next owner

(Eg.) Mutual Exclusion CRDT

- At most one replica has rights to the mutual exclusion token
  - An array V[R] to indicate who has the “lock”
  - Escrow
    - Owner transfer the token to the next owner
- Comparison function?

(Eg.) Mutual Exclusion CRDT

- At most one replica has rights to the mutual exclusion token
  - An array V[R] to indicate who has the “lock”
  - Escrow
    - Owner transfer the token to the next owner
- Comparison function?
  - Add a timestamp t

(Eg.) Mutual Exclusion CRDT

- At most one replica has rights to the mutual exclusion token
  - An array V[R] to indicate who has the "lock"
  - Escrow
    - Owner transfer the token to the next owner
- Comparison function?
  - Add a timestamp t

```
transfer((t,V),r_o):
  assert(V[r] = 1 ∧ (∀t_o ≠ t, t≥t_o))
  t = t+1
  V[r_s] = 0 # self
  V[r_o] = 1 # other
```

```
merge((t,V),(t_o,V_o)):
  t = max(t,t_o)
  v = (t_o<t)?V:V_o
```

(Eg.) Mutual Exclusion CRDT

```
transfer((t,V),r_o):
  assert(V[r] = 1 ∧ (∀t_o ≠ t, t≥t_o))
  t = t+1
  V[r_s] = 0 # self
  V[r_o] = 1 # other
```

```
merge((t,V),(t_o,V_o)):
  t = max(t,t_o)
  v = (t_o<t)?V:V_o
```

(EG.) MUTUAL EXCLUSION CRDT

```
transfer((t,V),rₒ):
  assert(V[r] = 1 ∧ (∀tₒ ≠ t, t≥tₒ))
  t = t+1
  V[rₛ] = 0 # self
  V[rₒ] = 1 # other
```

```
merge((t,V),(tₒ,Vₒ)):
  t = max(t,tₒ)
  v = (tₒ<t)?V:Vₒ
```

- Order:

(Eg.) Mutual Exclusion CRDT

```
transfer((t,V),r_o):
  assert(V[r] = 1 ∧ (∀t_o ≠ t, t≥t_o))
  t = t+1
  V[r_s] = 0 # self
  V[r_o] = 1 # other
```

```
merge((t,V),(t_o,V_o)):
  t = max(t,t_o)
  v = (t_o<t)?V:V_o
```

- Order: $(t_0, V_0) \leq (t_1, V_1) = t_0 \leq t_1 \wedge (t_0 \Rightarrow t_1 \wedge V_0 \leq V_1)$

(Eg.) Mutual Exclusion CRDT

```
transfer((t,V),r_o):
  assert(V[r] = 1 ∧ (∀t_o ≠ t, t≥t_o))
  t = t+1
  V[r_s] = 0 # self
  V[r_o] = 1 # other
```

```
merge((t,V),(t_o,V_o)):
  t = max(t,t_o)
  v = (t_o<t)?V:V_o
```

- Order: $$(t_0, V_0) \leq (t_1, V_1) = t_0 \leq t_1 \wedge (t_0 \Rightarrow t_1 \wedge V_0 \leq V_1)$$
- Invariant:

(Eg.) Mutual Exclusion CRDT

```
transfer((t,V),r_o):
  assert(V[r] = 1 ∧ (∀t_o ≠ t, t≥t_o))
  t = t+1
  V[r_s] = 0 # self
  V[r_o] = 1 # other
```

```
merge((t,V),(t_o,V_o)):
  t = max(t,t_o)
  v = (t_o<t)?V:V_o
```

- Order: $$(t_0, V_0) \leq (t_1, V_1) = t_0 \leq t_1 \wedge (t_0 \Rightarrow t_1 \wedge V_0 \leq V_1)$$
- Invariant: $$\mathsf{Inv}(\mathtt{t}, \mathtt{V}) \;=\; \sum_r \mathtt{V}[r] = 1$$

(Eg.) Mutual Exclusion CRDT

```
transfer((t,V),r_o):
  assert(V[r] = 1 ∧ (∀t_o ≠ t, t≥t_o))
  t = t+1
  V[r_s] = 0 # self
  V[r_o] = 1 # other
```

```
merge((t,V),(t_o,V_o)):
  t = max(t,t_o)
  v = (t_o<t)?V:V_o
```

- Order: $(t_0, V_0) \leq (t_1, V_1) = t_0 \leq t_1 \wedge (t_0 \Rightarrow t_1 \wedge V_0 \leq V_1)$
- Invariant: $\mathsf{Inv}(\mathtt{t}, \mathtt{V}) \;=\; \sum_{r} \mathtt{V}[r] = 1$
- Merge precondition:

(Eg.) Mutual Exclusion CRDT

```
transfer((t,V),r_o):
  assert(V[r] = 1 ∧ (∀t_o ≠ t, t≥t_o))
  t = t+1
  V[r_s] = 0 # self
  V[r_o] = 1 # other
```

```
merge((t,V),(t_o,V_o)):
  t = max(t,t_o)
  v = (t_o<t)?V:V_o
```

- Order: $(t_0, V_0) \leq (t_1, V_1) = t_0 \leq t_1 \wedge (t_0 \Rightarrow t_1 \wedge V_0 \leq V_1)$
- Invariant: $\mathsf{Inv}(\mathtt{t}, \mathtt{V}) = \sum_r \mathtt{V}[r] = 1$
- Merge precondition: $\mathsf{Pre}_{\mathtt{merge}}((\mathtt{t}_s, \mathtt{V}_s), (\mathtt{t}_o, \mathtt{V}_o)) = \mathsf{Inv}(\mathtt{t}_s, \mathtt{V}_s) \wedge \mathsf{Inv}(\mathtt{t}_o, \mathtt{V}_o) \wedge$
  $(\mathtt{t}_s = \mathtt{t}_o \Rightarrow \mathtt{V}_s = \mathtt{V}_o) \wedge$
  $(\mathtt{V}[\mathtt{r_s}] = 1 \Rightarrow \mathtt{t}_s \geq \mathtt{t}_o)$

Tool Support

- Input
  - Definition of Order
  - Definition of merging function
  - Invariant

Tool Support

- Input
  - Definition of Order
  - Definition of merging function
  - Invariant
- Soteria (sister to CISE/CEC)
  - Implemented on top of Boogie
  - Performs the lattice and invariant checks

Tool Support

- Input
  - Definition of Order
  - Definition of merging function
  - Invariant
- Soteria (sister to CISE/CEC)
  - Implemented on top of Boogie
  - Performs the lattice and invariant checks
- Work in progress

Tool Support

- Input

```
~/r/c/c/s/s/soteria (origin/rewriting±) ▶ python3 soteria.py specs/token.spec
INFO     : ************ token ***************
INFO     : Checking the syntax
INFO     : Parsing the specification
INFO     : Checking the well-formedness of the specification
INFO     : Checking convergence
INFO     : Checking safety
INFO     : The specification is safe!!!
```

  - Implemented on top of Boogie
  - Performs the lattice and invariant checks
- Work in progress

Research Opportunities

- Beyond Simple Invariants
  - Pre/Post conditions of client programs using (1+) CRDTs
- Transactions + CRDTs
- Consistency Models: Eventual, Causal, Strong, …
- Synchronization?
- …