Convergence is Not Enough

Automatically Verifying Replicated Data Types

KC Sivaramakrishnan
Joint work with Pranav Ramesh, Vimala Soundarapandian, Kartik Nagar

SERI 2026
16th Jul 2026

FP Launchpad

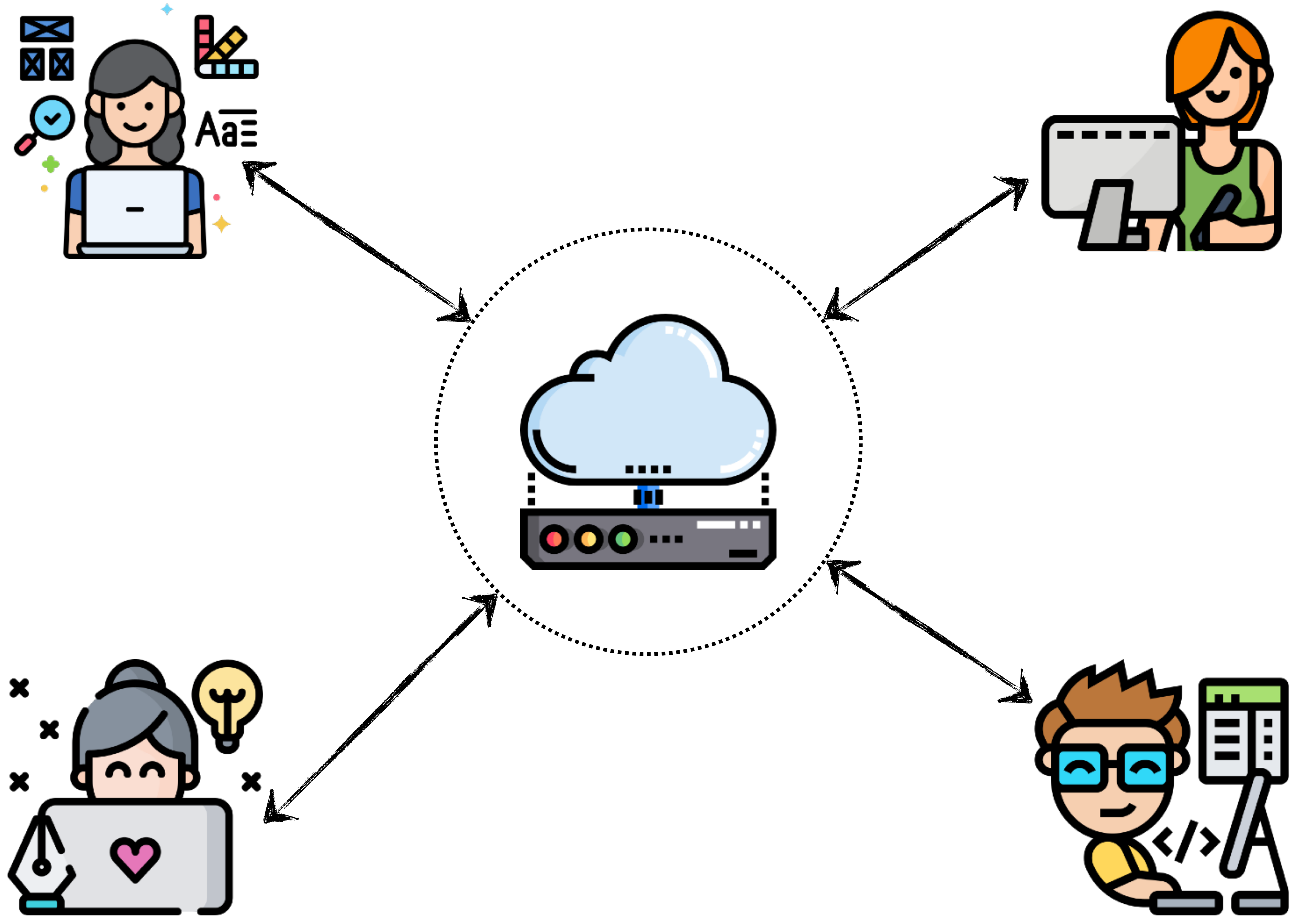
IIT MADRAS

Tarides

Collaborative Applications

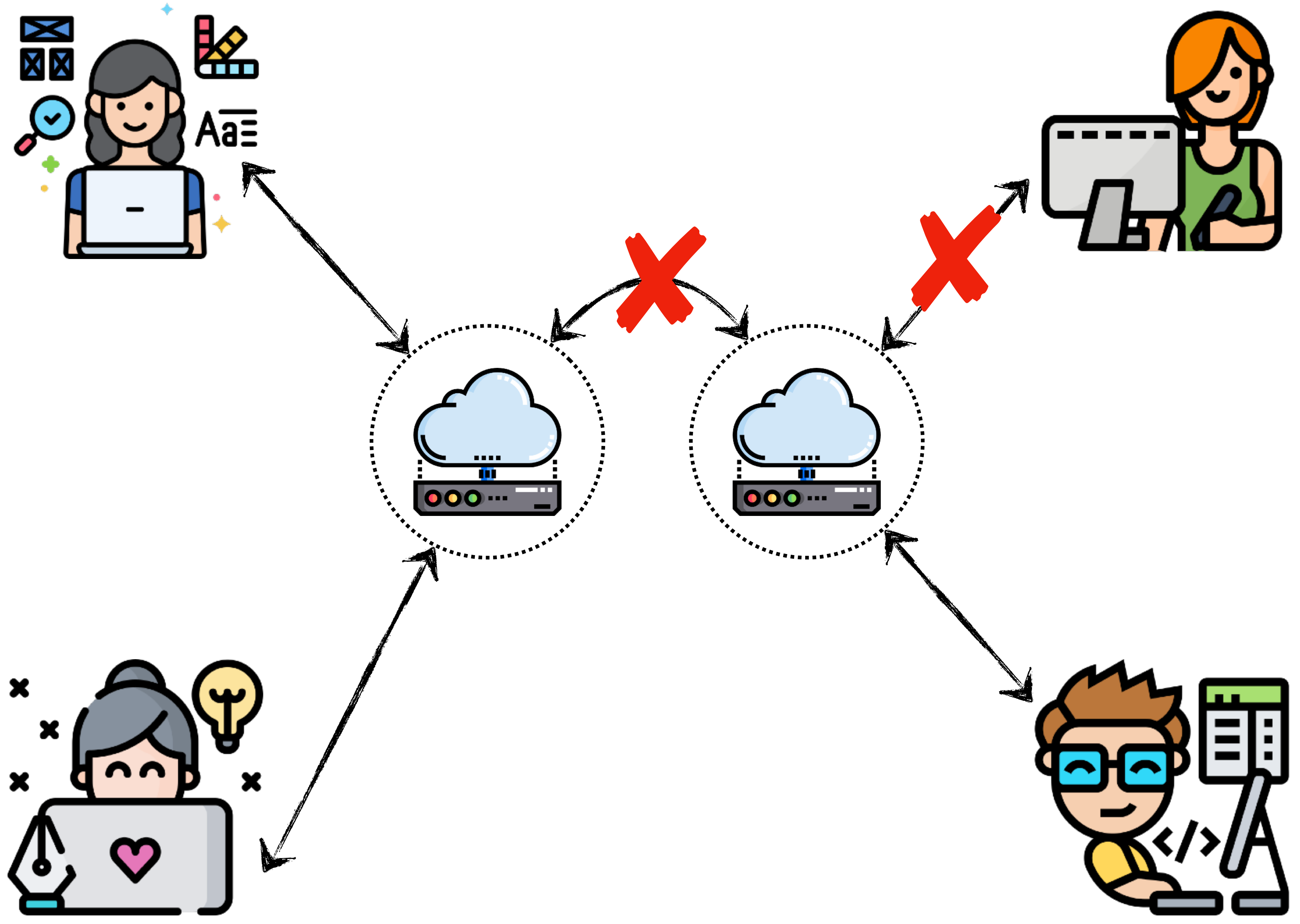
Collaborative Applications

Airtable Google Docs Figma Notion Overleaf



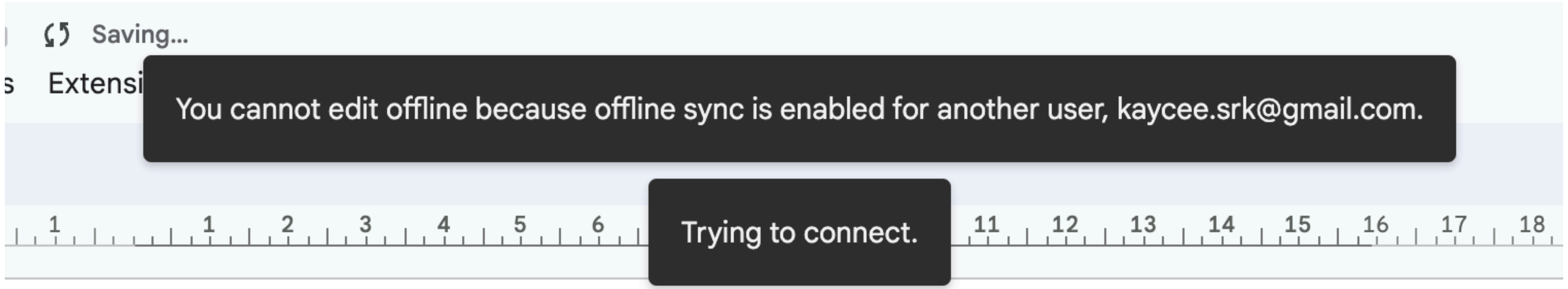
Collaborative Applications

Airtable Google Docs Figma Notion Overleaf



Network Partitions

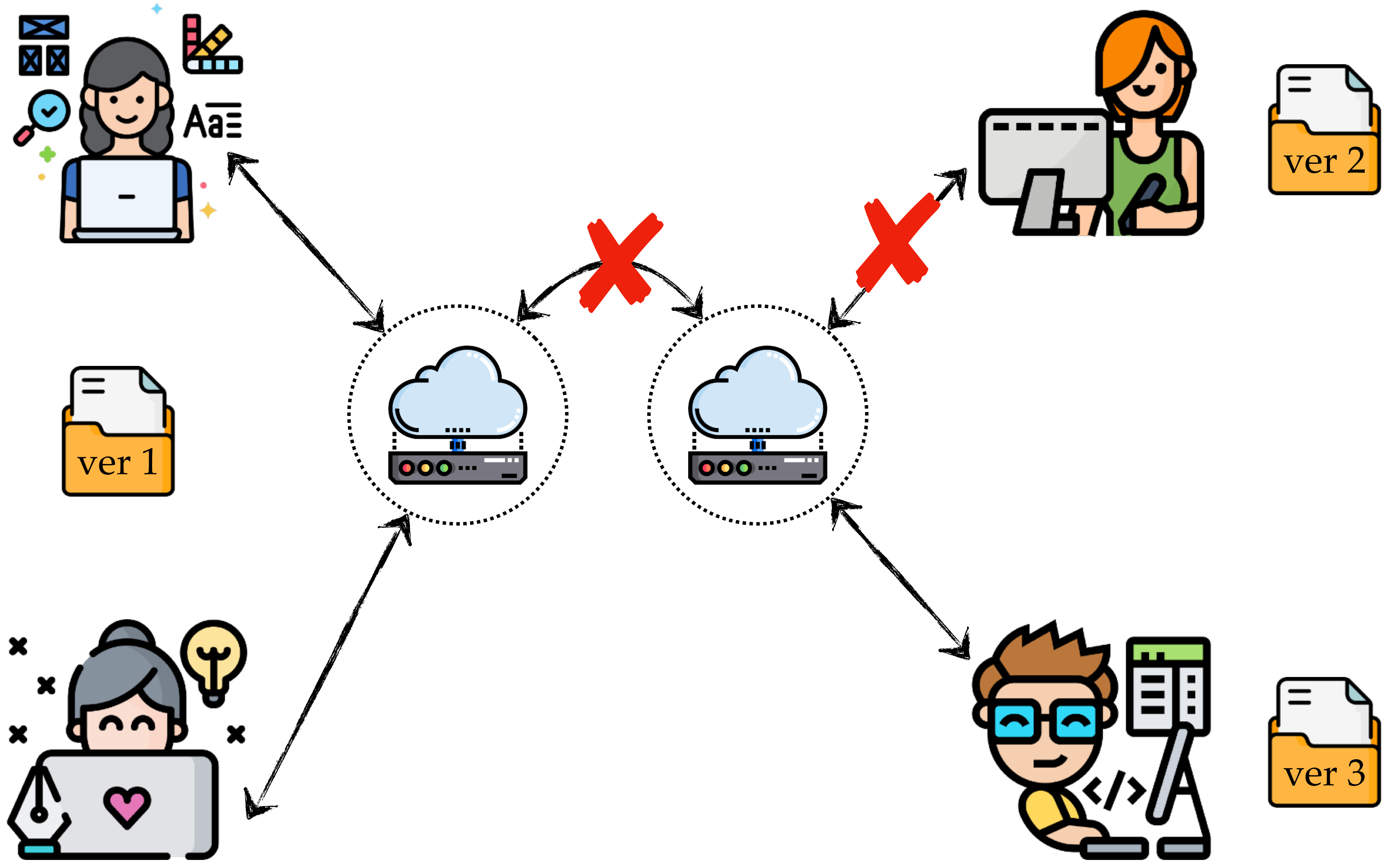
- Centralised Apps provide limited support for offline editing



Enabling offline sync for one account prevents other accounts from working offline

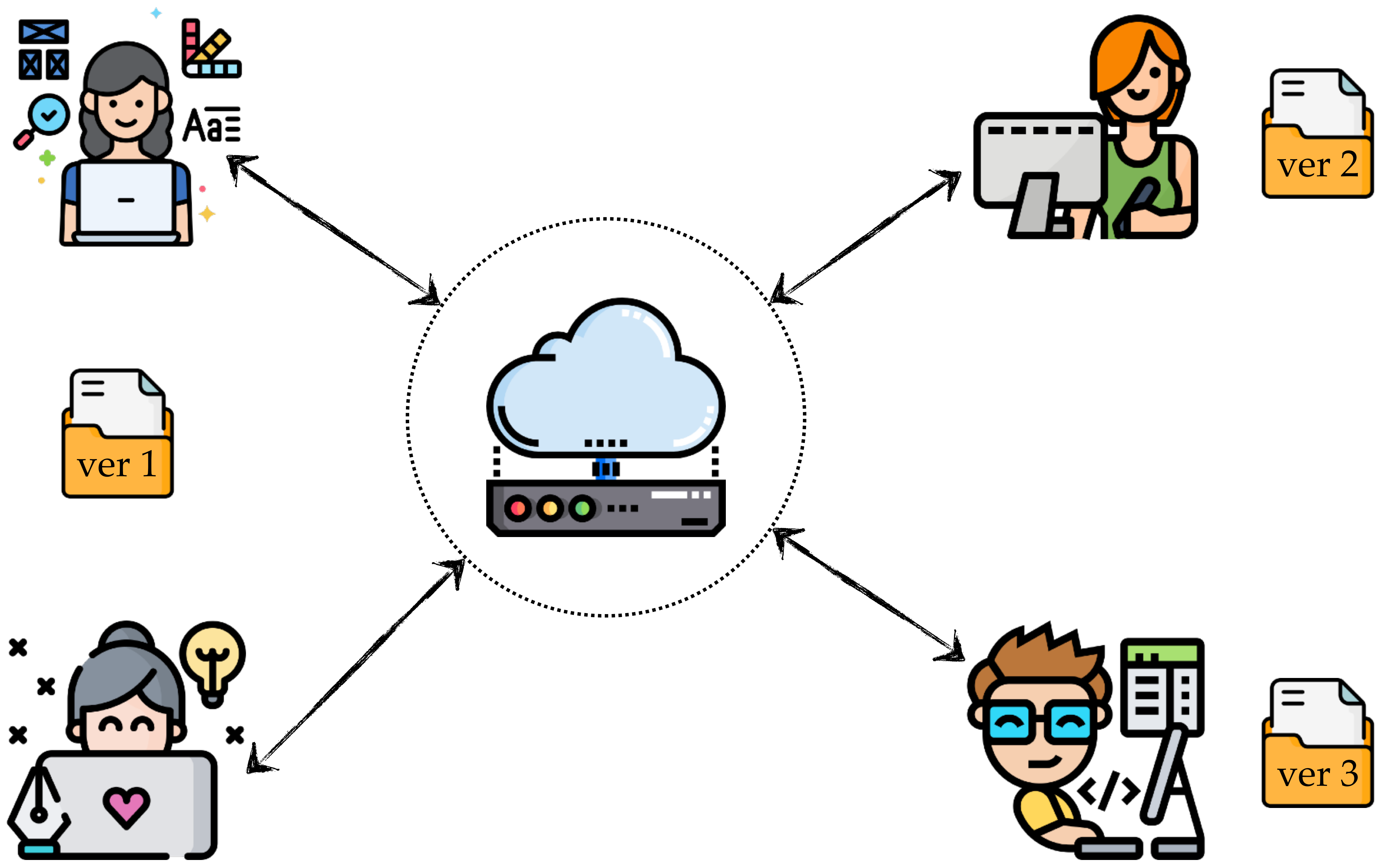
Local-first software

Airtable Google Docs Figma Notion Overleaf



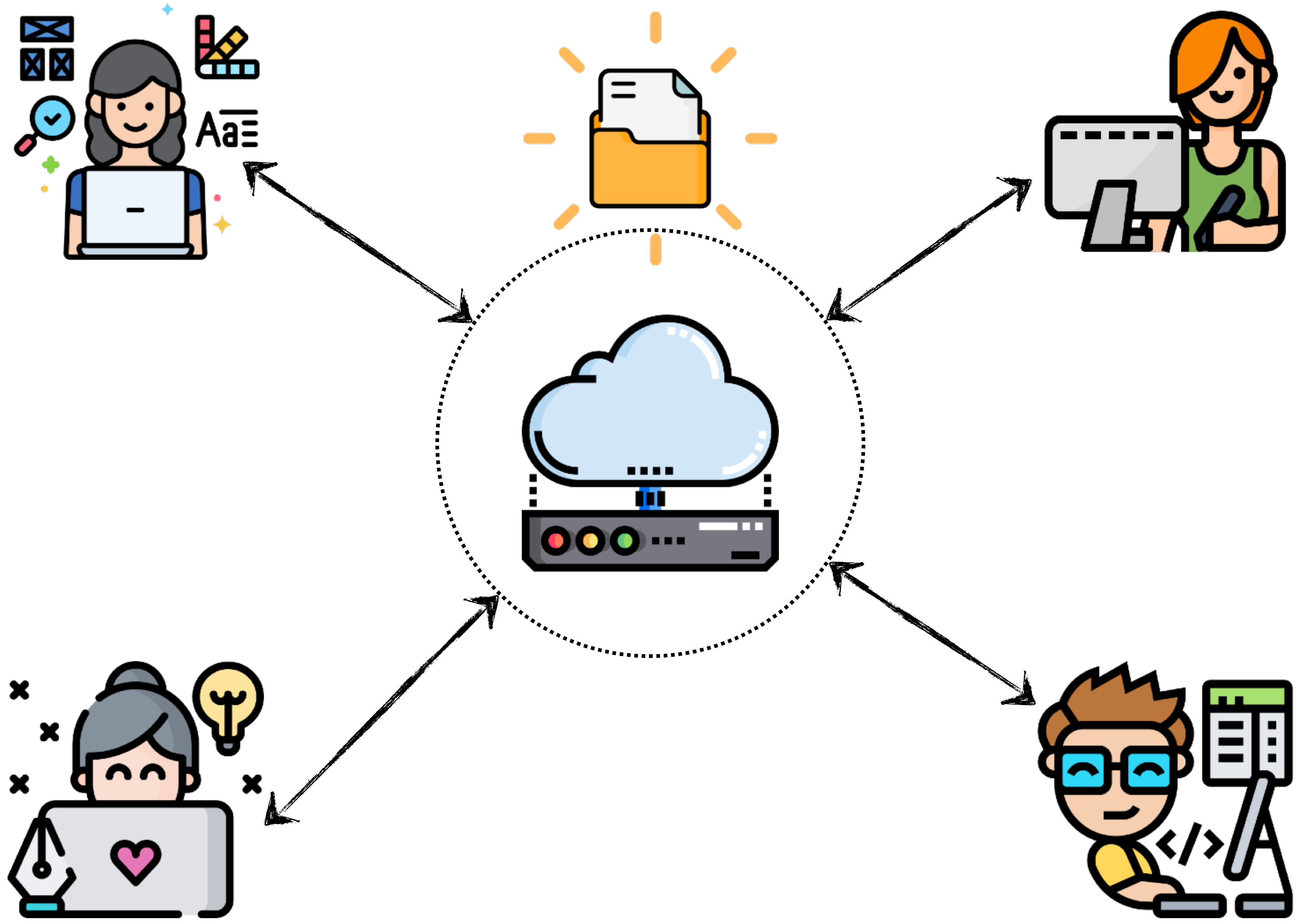
Local-first software

Airtable Google Docs Figma Notion Overleaf



Local-first software

Airtable Google Docs Figma Notion Overleaf



How do we build such applications?

How do we build such applications?

Embed the notion of replication into the data types

Convergent Replicated Data Types (CRDTs)

Convergent Replicated Data Types (CRDTs)

- Multiple replicas, concurrent evolution, no coordination, ***guaranteed convergence***

Convergent Replicated Data Types (CRDTs)

- Multiple replicas, concurrent evolution, no coordination, ***guaranteed convergence***
- Two flavours of CRDTs — Operation-based and state-based
    - Equivalent in expressive power

Convergent Replicated Data Types (CRDTs)

- Multiple replicas, concurrent evolution, no coordination, ***guaranteed convergence***
- Two flavours of CRDTs — Operation-based and state-based
  - Equivalent in expressive power
- Known RDTs
  - Counters, Sets, Maps, Sequences/Lists, Trees, Graphs, JSON

Convergent Replicated Data Types (CRDTs)

- Multiple replicas, concurrent evolution, no coordination, ***guaranteed convergence***
- Two flavours of CRDTs — Operation-based and state-based
    - Equivalent in expressive power
- Known RDTs
    - Counters, Sets, Maps, Sequences/Lists, Trees, Graphs, JSON
- In practice
    - Collaborative editing — YATA/Yjs, Automerge
    - Figma (CRDT-ish), Zed Editor, …

Mergeable Replicated Data Types (MRDTs)

- MRDTs = Sequential data types + 3-way merge function à la Git

Mergeable Replicated Data Types (MRDTs)

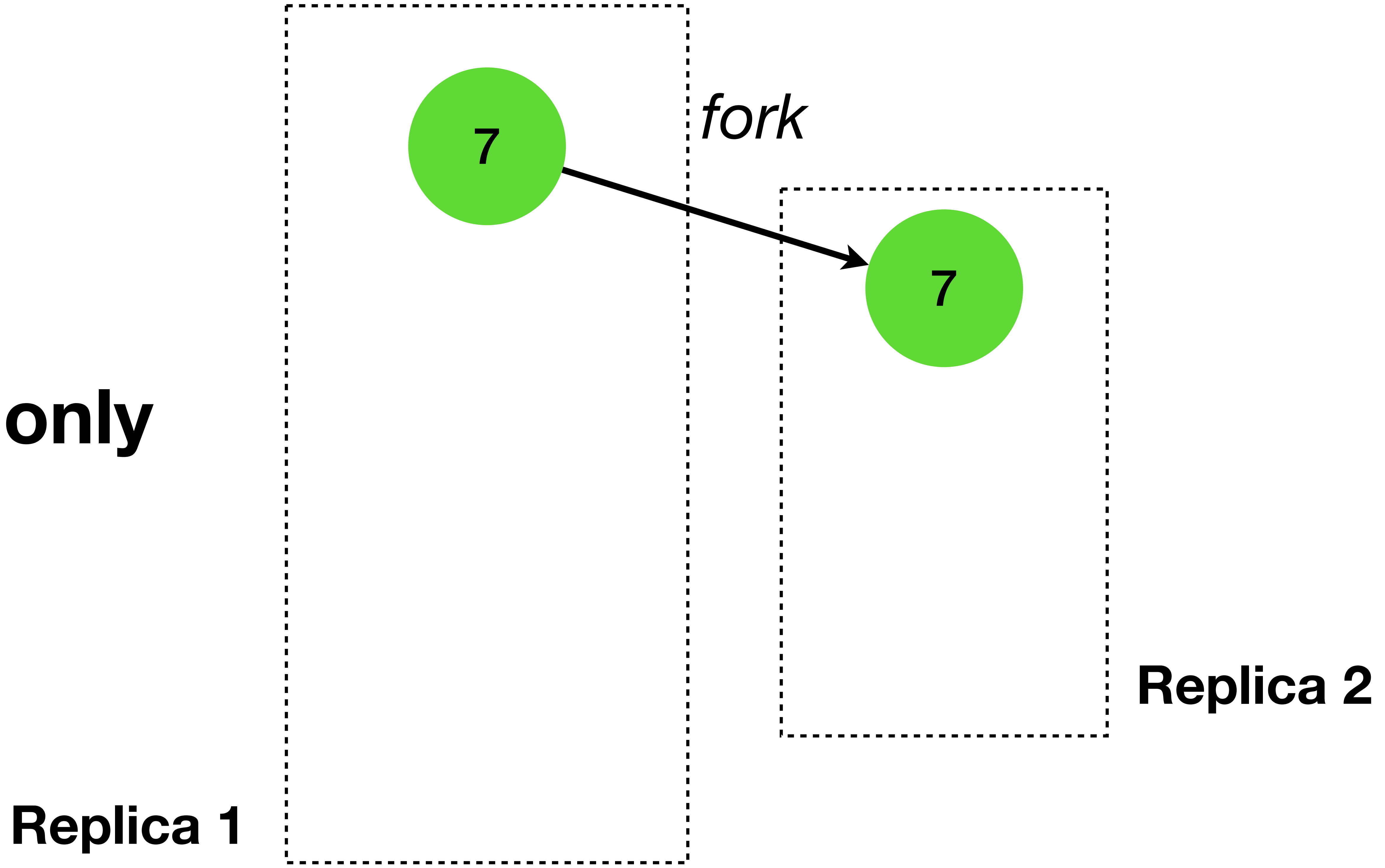
- MRDTs = Sequential data types + 3-way merge function à la Git

Increment-only counter

Mergeable Replicated Data Types (MRDTs)

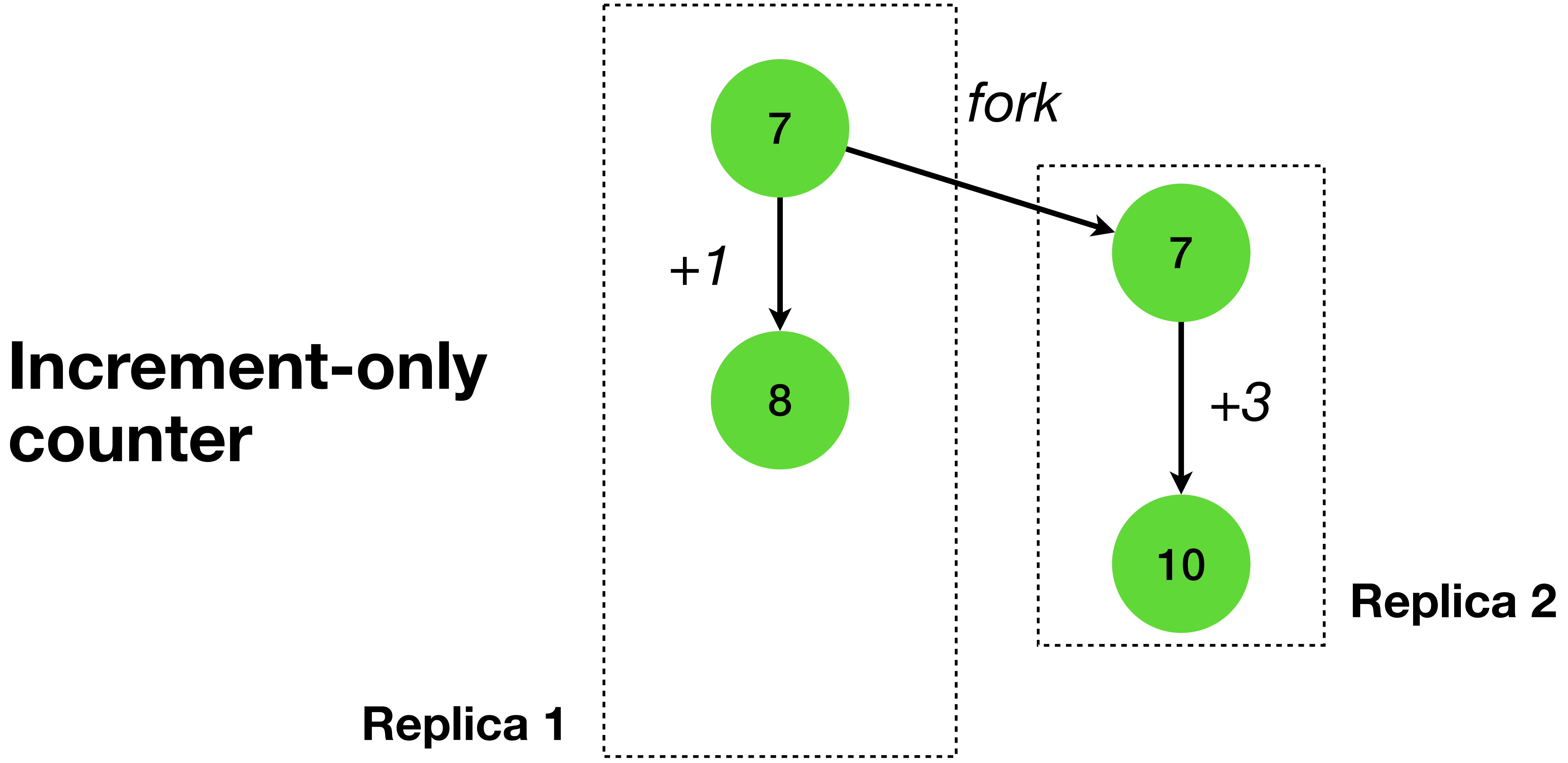
- MRDTs = Sequential data types + 3-way merge function à la Git

Increment-only counter



Mergeable Replicated Data Types (MRDTs)

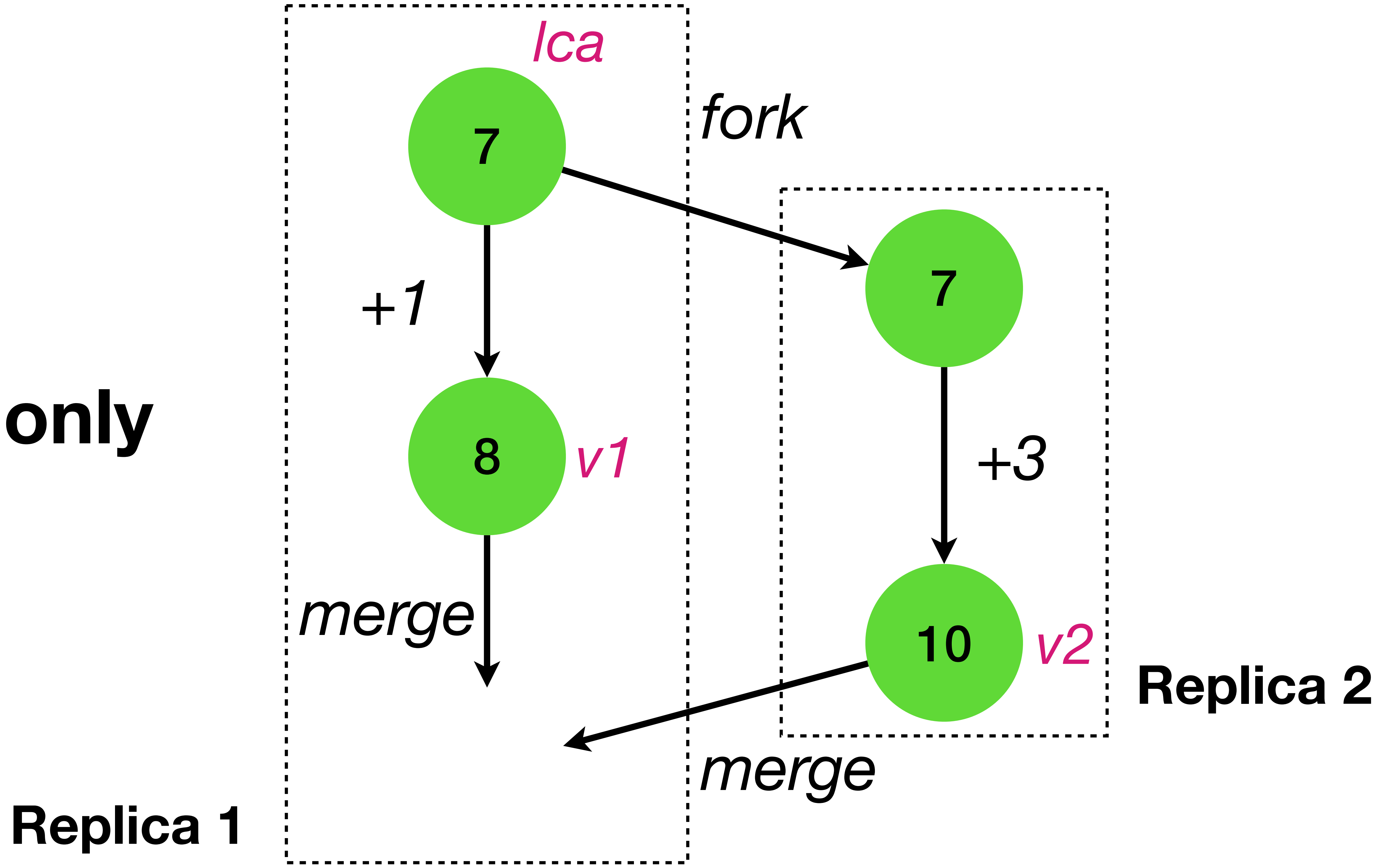
- MRDTs = Sequential data types + 3-way merge function à la Git



Mergeable Replicated Data Types (MRDTs)

- MRDTs = Sequential data types + 3-way merge function à la Git

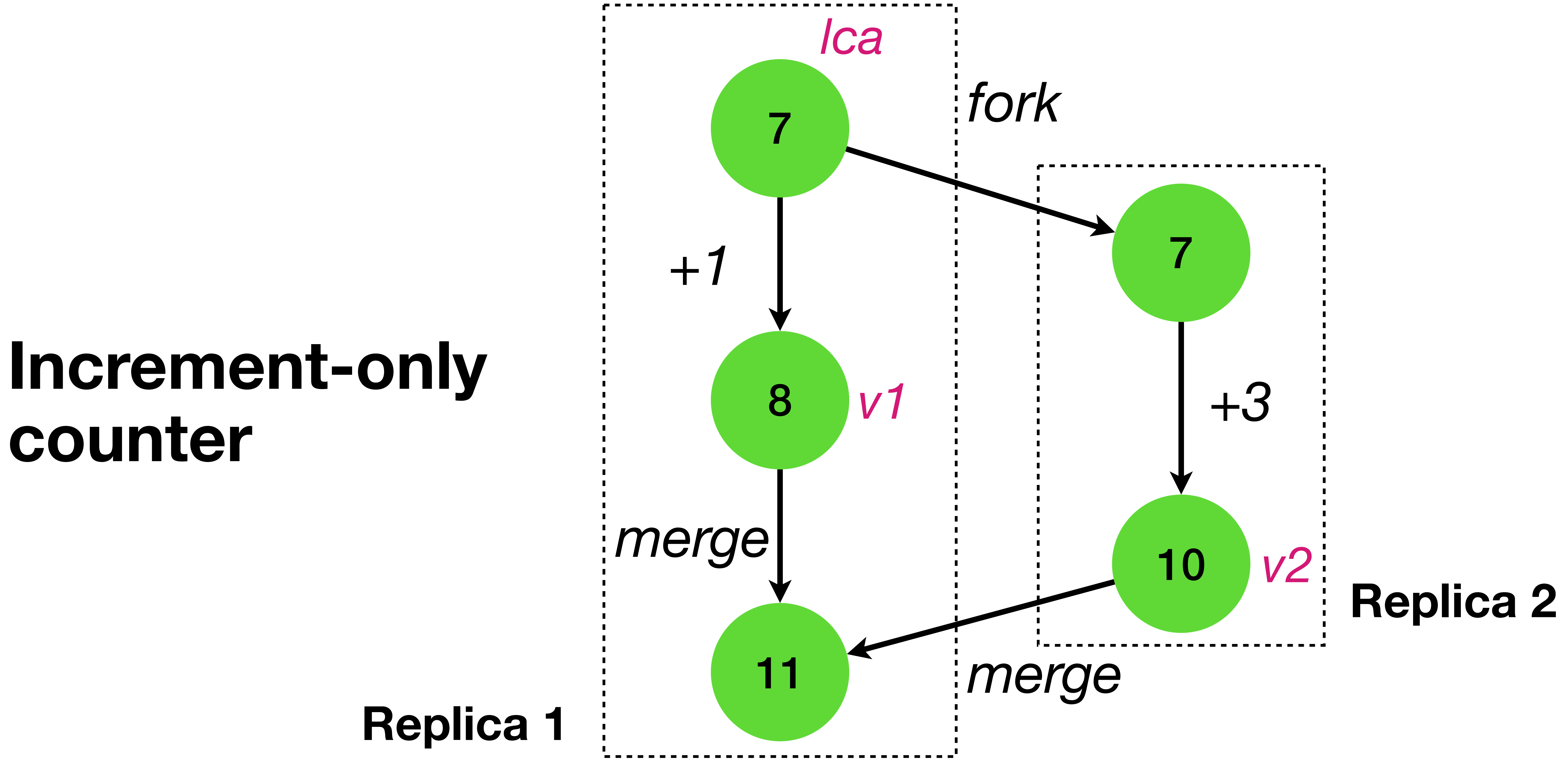
Increment-only counter




```
let merge lca v1 v2 =
  lca + (v1 - lca) + (v2 - lca)
```

Mergeable Replicated Data Types (MRDTs)

- MRDTs = Sequential data types + 3-way merge function à la Git

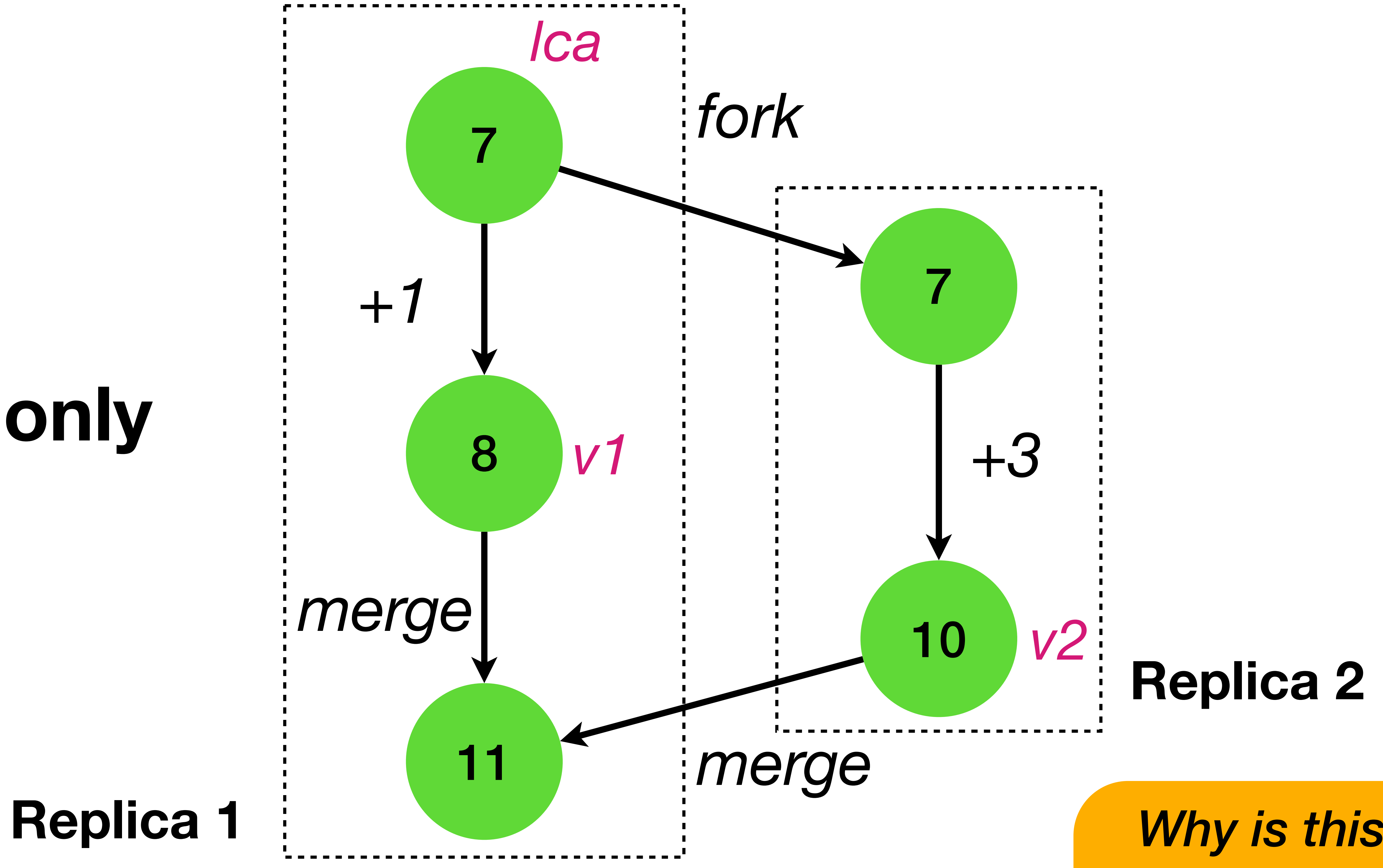



```
let merge lca v1 v2 =
  lca + (v1 − lca) + (v2 − lca)
```

Mergeable Replicated Data Types (MRDTs)

- MRDTs = Sequential data types + 3-way merge function à la Git

Increment-only counter




```
let merge lca v1 v2 =
  lca + (v1 - lca) + (v2 - lca)
```

Why is this correct?

How do we automatically verify it?

OR Set MRDT (aka Add-Wins Set)

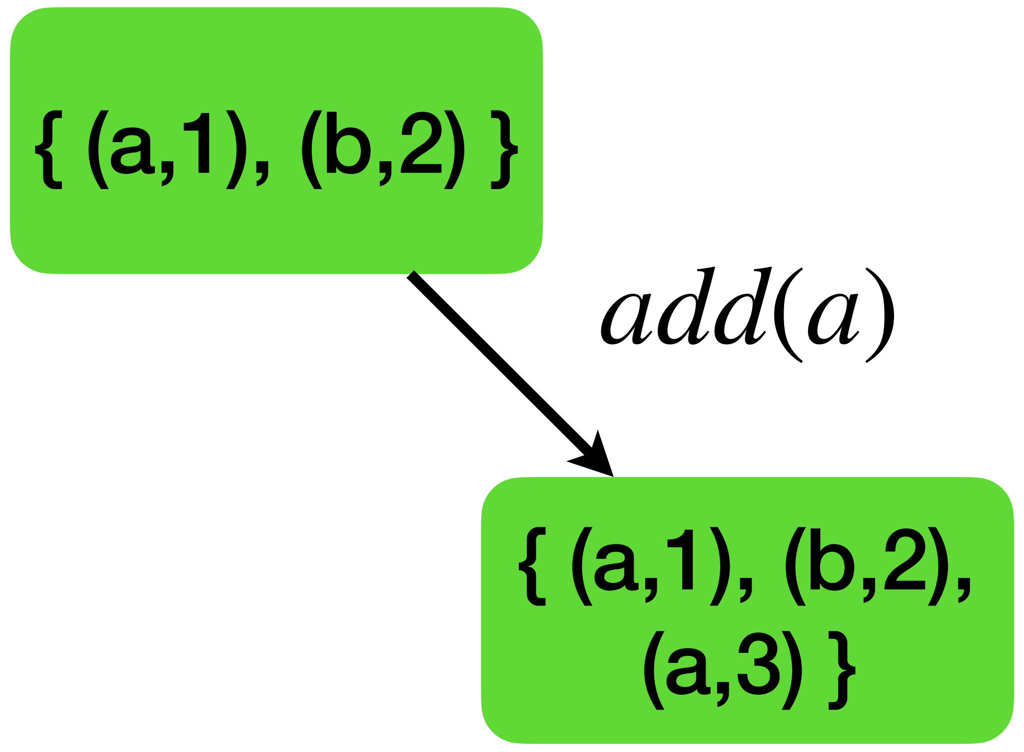
{ (a,1), (b,2) }

OR Set MRDT (aka Add-Wins Set)

{ (a,1), (b,2) }

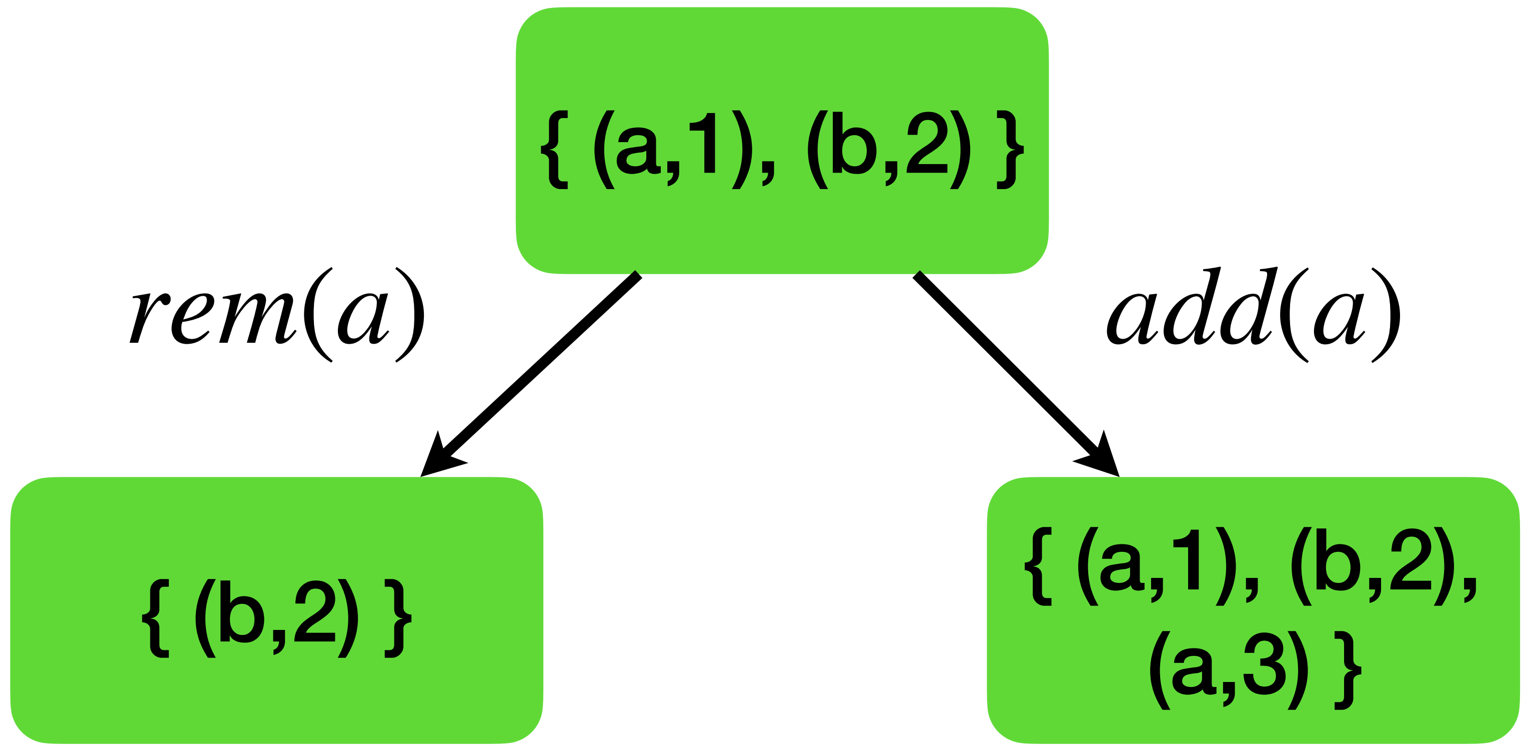
- Associate ***timestamp (id)*** with the element

OR Set MRDT (aka Add-Wins Set)



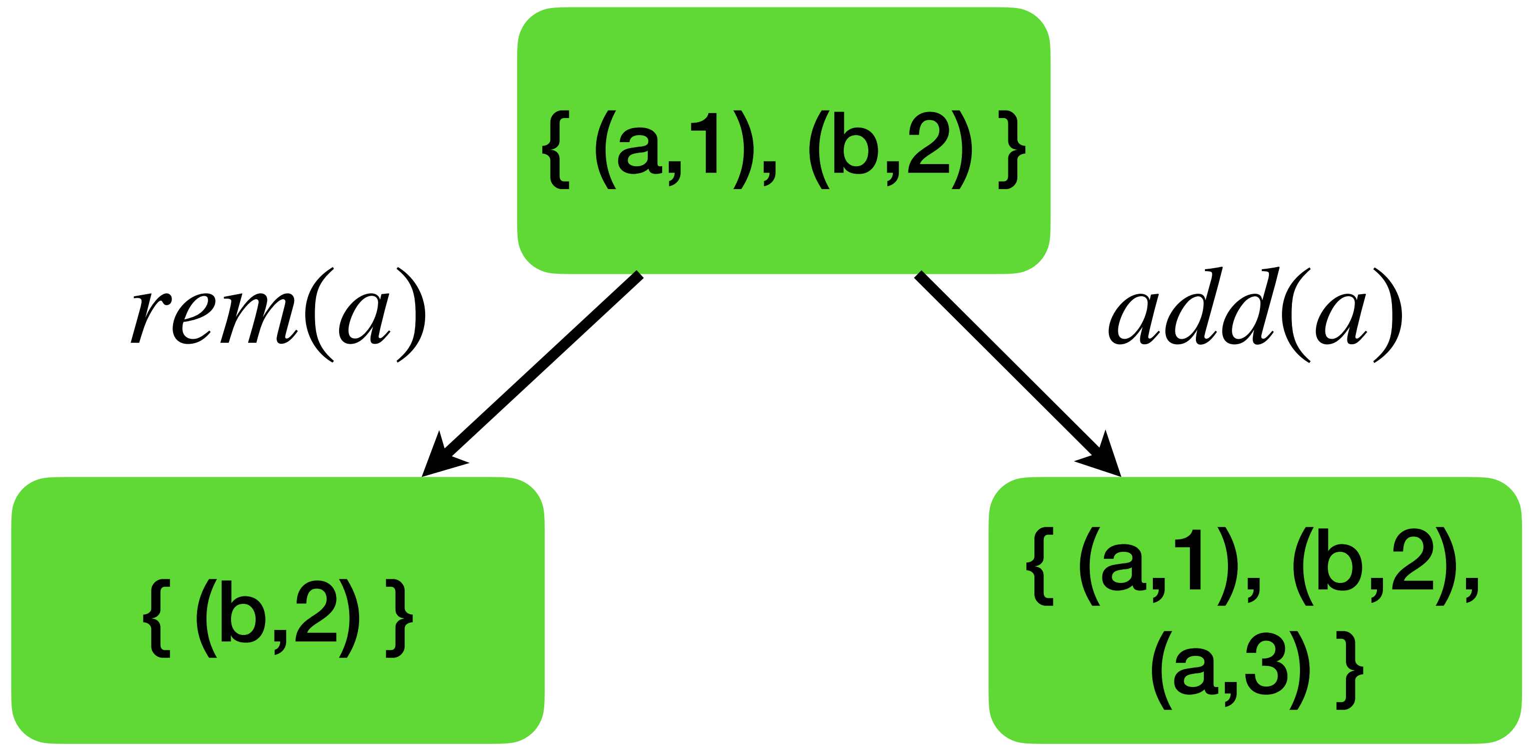
- Associate ***timestamp (id)*** with the element
- Add associates a ***fresh*** timestamp with the element

OR Set MRDT (aka Add-Wins Set)



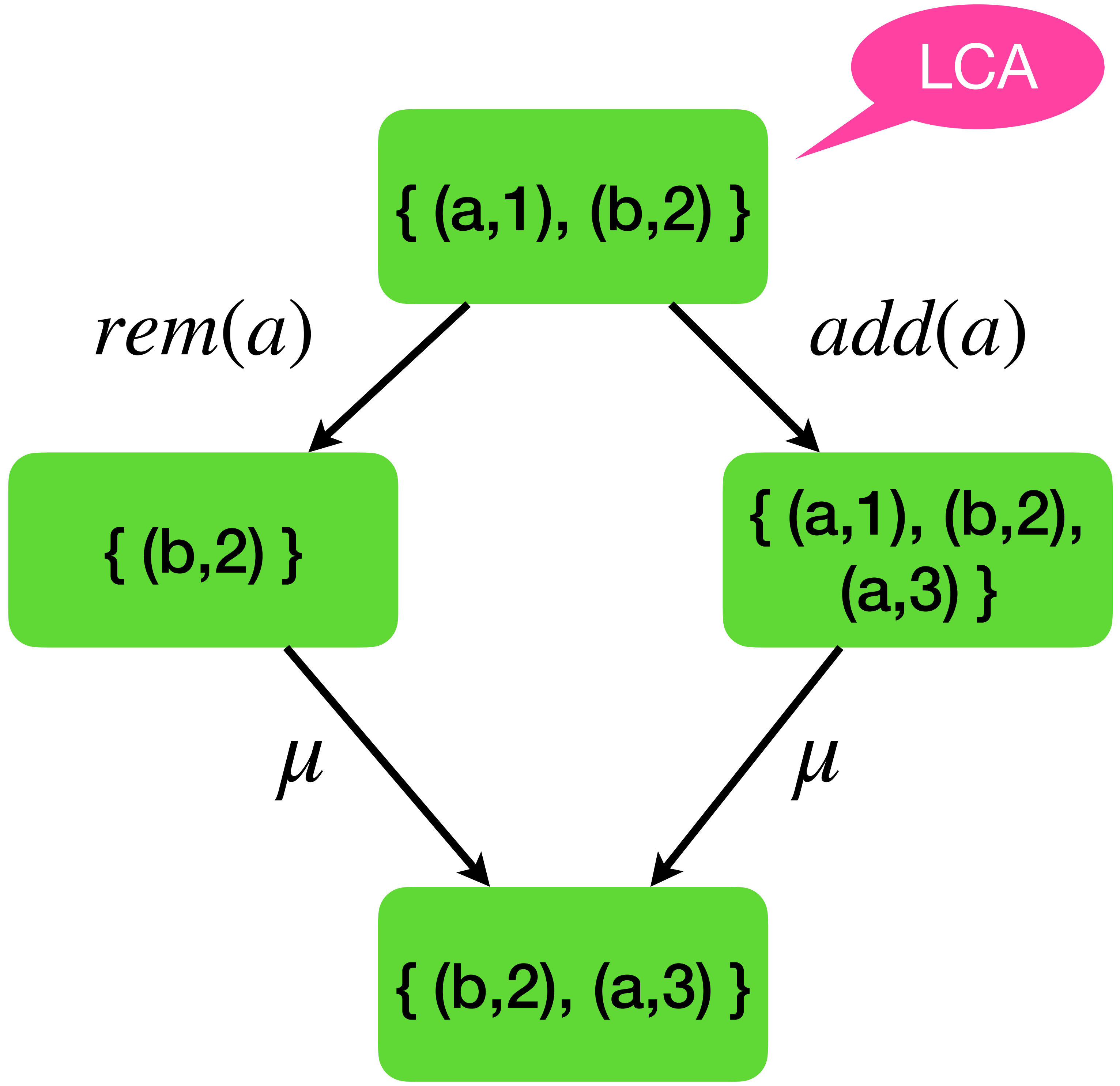
- Associate ***timestamp (id)*** with the element
- Add associates a ***fresh*** timestamp with the element
- Remove removes all matching elements with ***any id***
  - ***Removed from the concrete state!***

OR Set MRDT (aka Add-Wins Set)



- Associate ***timestamp (id)*** with the element
- Add associates a ***fresh*** timestamp with the element
- Remove removes all matching elements with ***any id***
  - ***Removed from the concrete state!***
- Read returns the set removing ids

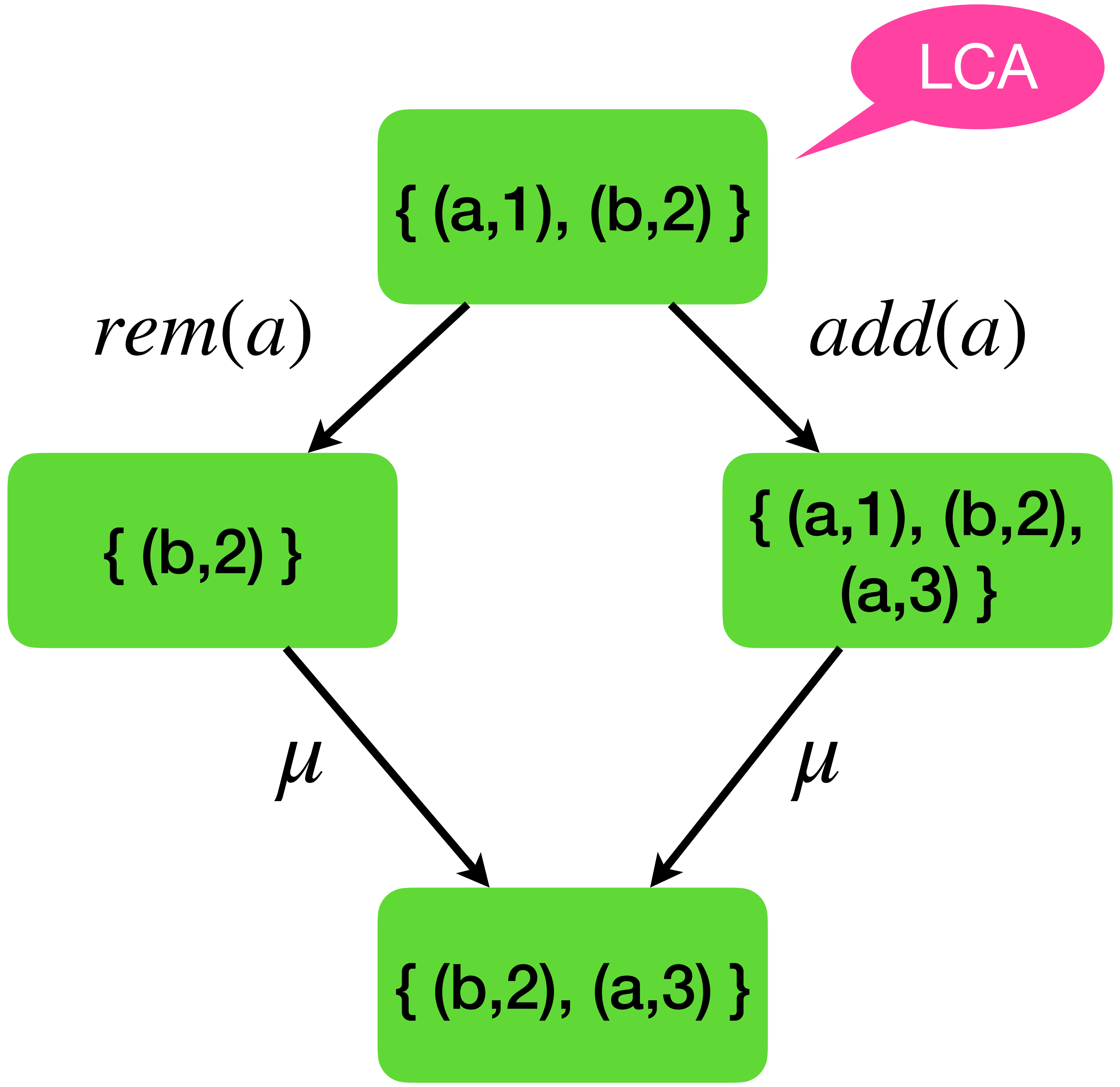
OR Set MRDT (aka Add-Wins Set)



- Associate ***timestamp (id)*** with the element
- Add associates a ***fresh*** timestamp with the element
- Remove removes all matching elements with ***any id***
  - ***Removed from the concrete state!***
- Read returns the set removing ids

```
let merge lca v1 v2 =
  (lca ∩ v1 ∩ v2) (* unchanged or removed elements *)
  ∪ (v1 \ lca) (* added elements on left *)
  ∪ (v2 \ lca) (* added elements on right *)
```

OR Set MRDT (aka Add-Wins Set)



- Associate ***timestamp (id)*** with the element
- Add associates a ***fresh*** timestamp with the element
- Remove removes all matching elements with ***any id***
  - ***Removed from the concrete state!***
- Read returns the set removing ids

```
let merge lca v1 v2 =
  (lca ∩ v1 ∩ v2) (* unchanged or removed elements *)
  ∪ (v1 \ lca) (* added elements on left *)
  ∪ (v2 \ lca) (* added elements on right *)
```

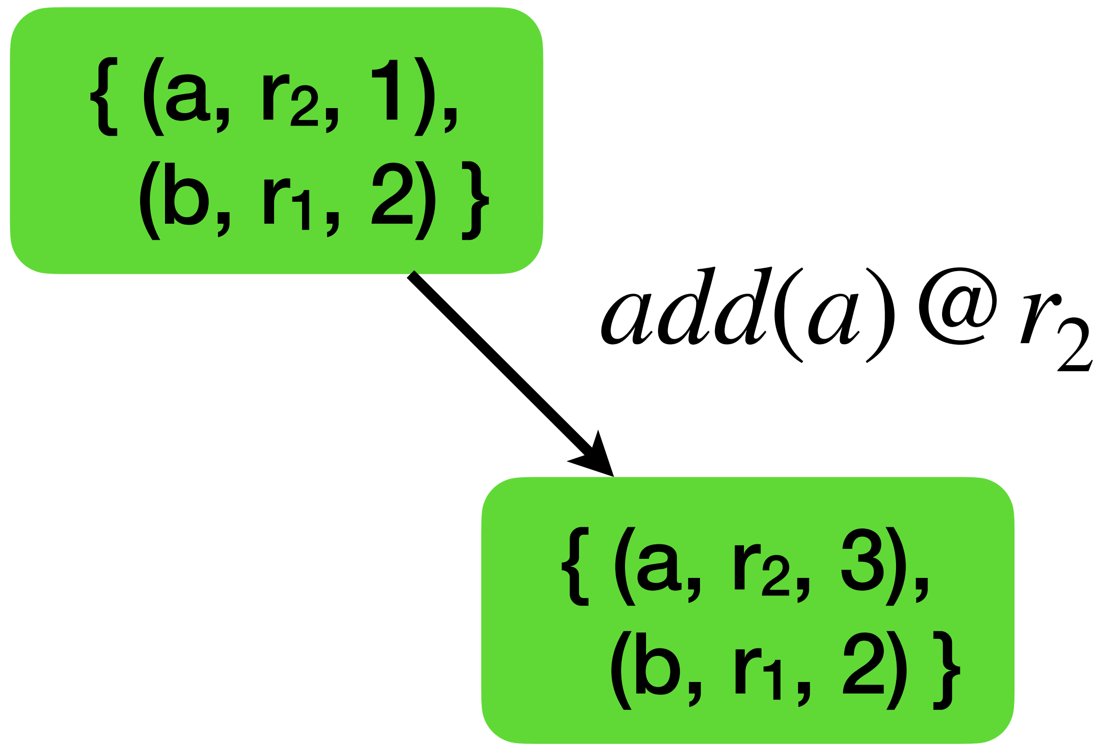
- Space usage $O(|\text{adds} \cap \text{live}|)$
  - Duplicates — same element with different ids

Efficient OR set MRDT

$\{ (a, r_2, 1),$
$(b, r_1, 2) \}$

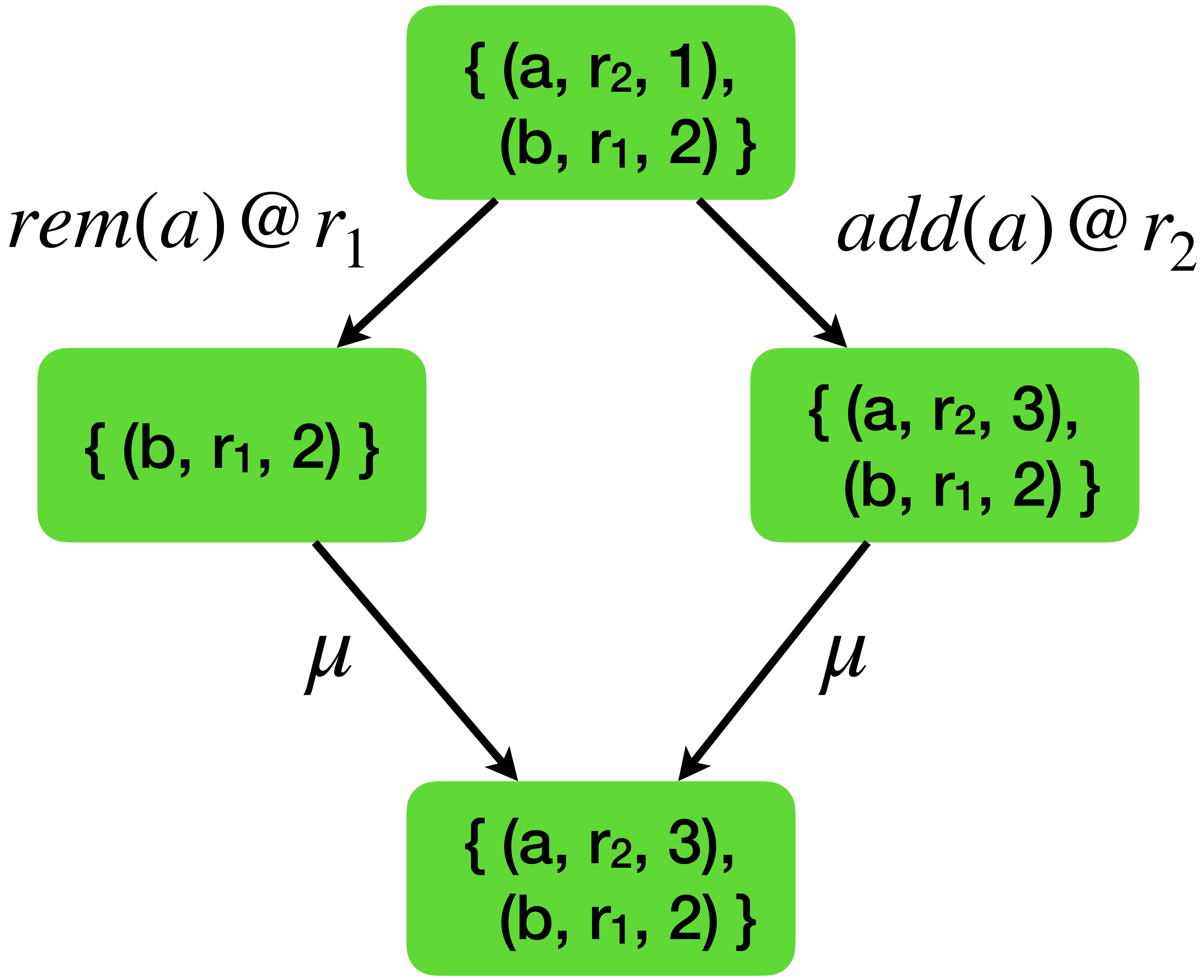
- Elements tagged with ***replica id*** and ***timestamp***

Efficient OR set MRDT



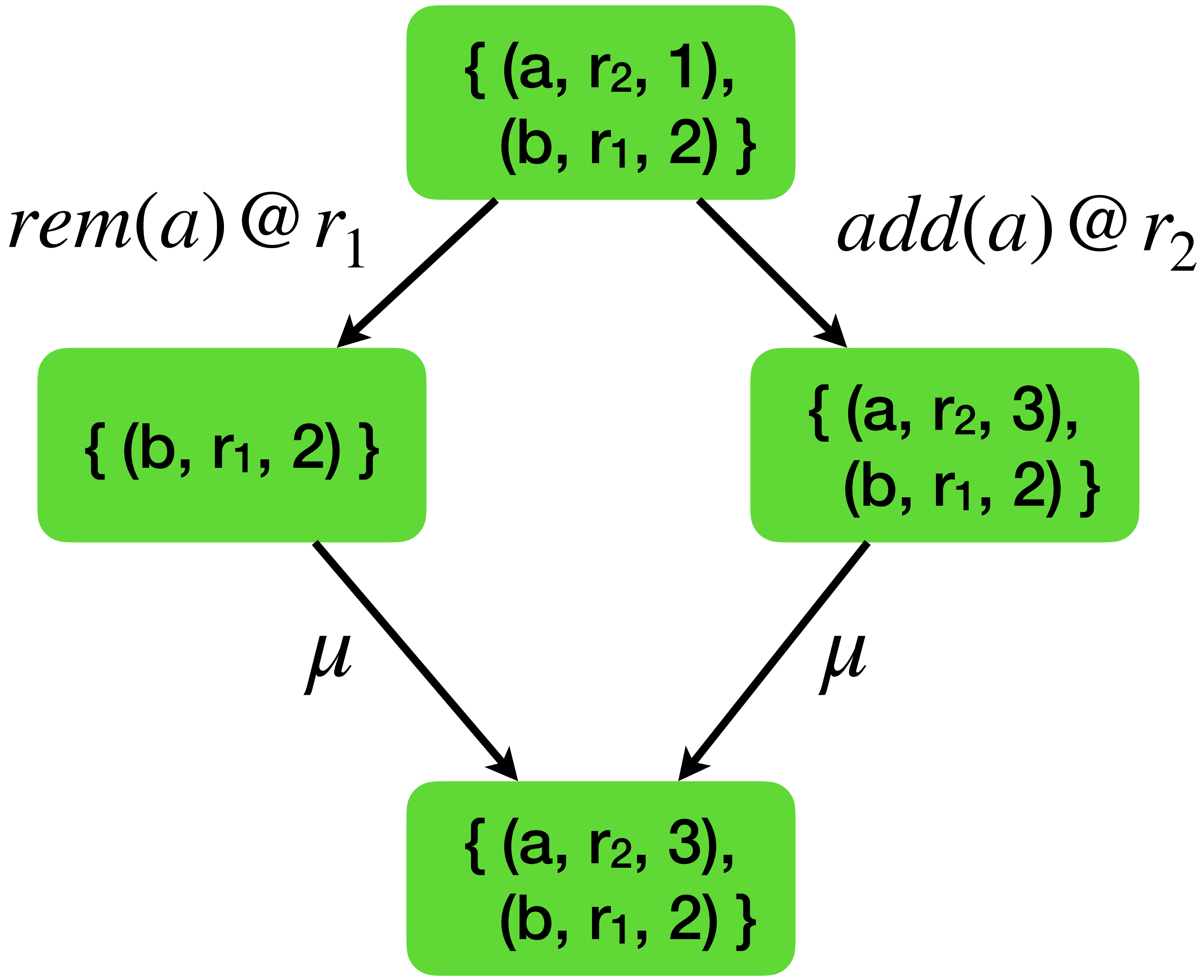
- Elements tagged with ***replica id*** and ***timestamp***
- **add(e)** @ replica r,
  - Remove any **(e, r, ts)** for any **ts**
  - Add **(e, r, ts')** for a fresh timestamp **ts'**

Efficient OR set MRDT



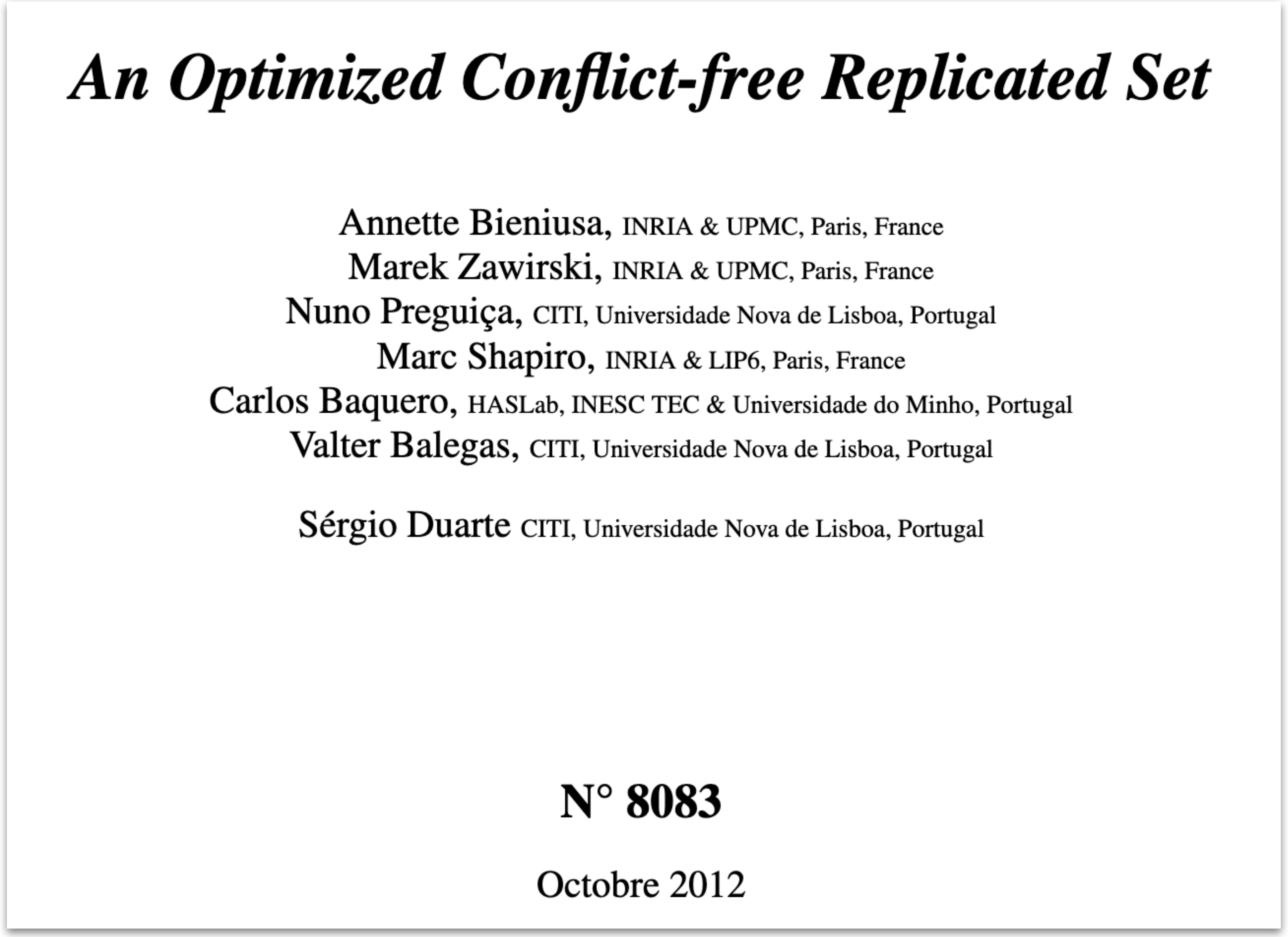
- Elements tagged with ***replica id*** and ***timestamp***
- **add(e)** @ replica r,
    - Remove any **(e, r, ts)** for any **ts**
    - Add **(e, r, ts')** for a fresh timestamp **ts'**
- Remove, read and merge remain similar.

Efficient OR set MRDT



- Elements tagged with ***replica id*** and ***timestamp***
- **add(e)** @ replica r,
    - Remove any **(e, r, ts)** for any **ts**
    - Add **(e, r, ts')** for a fresh timestamp **ts'**
- Remove, read and merge remain similar.
- Space usage $O(|\text{live}|.n)$
    - Duplicates — same element from different replicas

Not the first efficient OR set



An Optimized Conflict-free Replicated Set

Annette Bieniusa, INRIA & UPMC, Paris, France
Marek Zawirski, INRIA & UPMC, Paris, France
Nuno Preguiça, CITI, Universidade Nova de Lisboa, Portugal
Marc Shapiro, INRIA & LIP6, Paris, France
Carlos Baquero, HASLab, INESC TEC & Universidade do Minho, Portugal
Valter Balegas, CITI, Universidade Nova de Lisboa, Portugal

Sérgio Duarte CITI, Universidade Nova de Lisboa, Portugal

N° 8083

Octobre 2012

- Operation-based CRDT
- Maintains causality info in a version vector
- Space complexity $O(|\text{live}|.n + n)$

Not the first efficient OR set

RESEARCH-ARTICLE | FREE ACCESS

Riak DT map: a composable, convergent replicated dictionary

Authors: Russell Brown, Sean Cribbs, Christopher Meiklejohn, Sam Elliott | Authors Info & Claims

PaPEC '14: Proceedings of the First Workshop on Principles and Practice of Eventual Consistency • Article No.: 1, Page 1
https://doi.org/10.1145/2596631.2596633

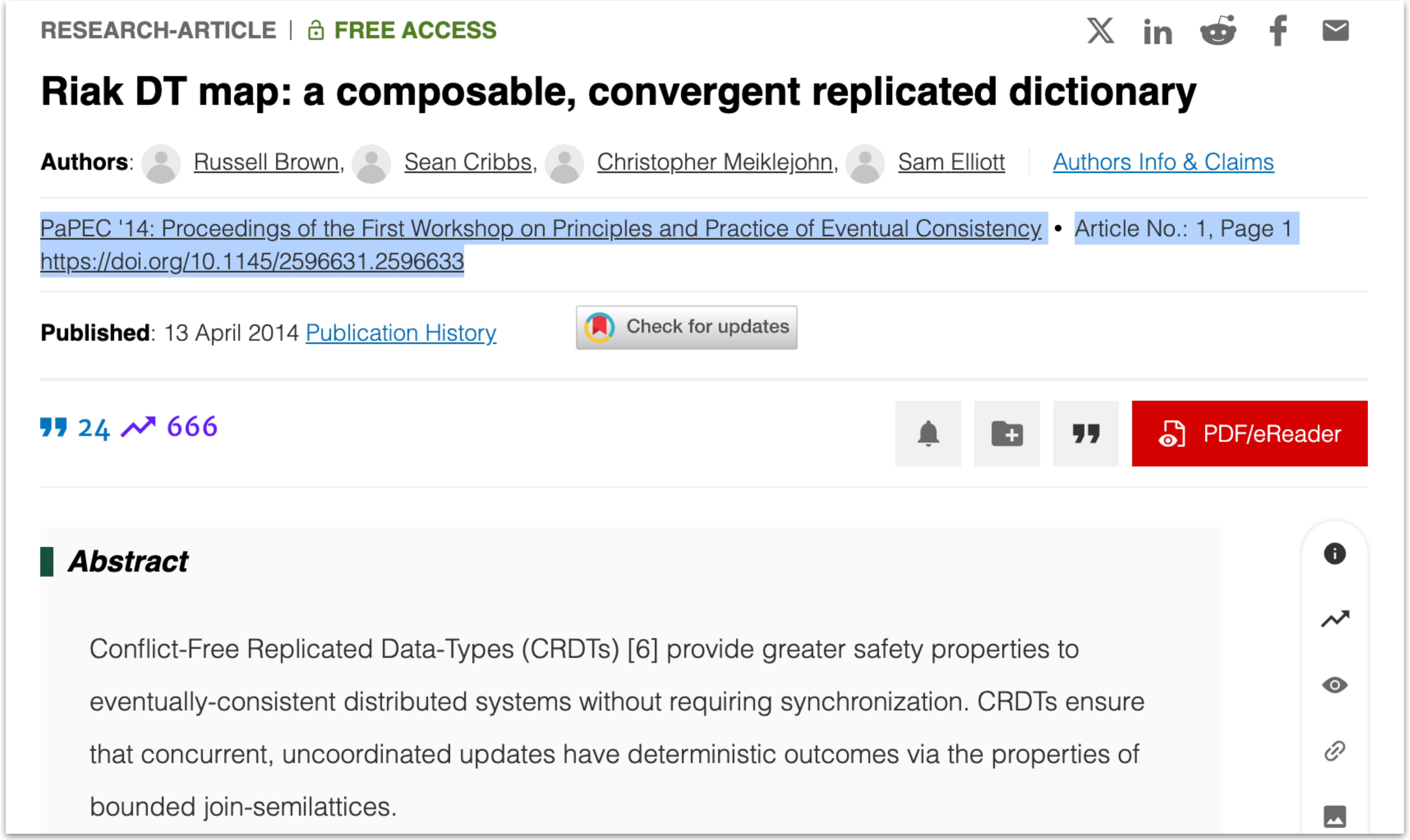
Published: 13 April 2014 Publication History

Abstract

Conflict-Free Replicated Data-Types (CRDTs) [6] provide greater safety properties to eventually-consistent distributed systems without requiring synchronization. CRDTs ensure that concurrent, uncoordinated updates have deterministic outcomes via the properties of bounded join-semilattices.

- State-based CRDT in Riak

Not the first efficient OR set


RESEARCH-ARTICLE | FREE ACCESS

Riak DT map: a composable, convergent replicated dictionary

Authors: Russell Brown, Sean Cribbs, Christopher Meiklejohn, Sam Elliott | Authors Info & Claims

PaPEC '14: Proceedings of the First Workshop on Principles and Practice of Eventual Consistency • Article No.: 1, Page 1
https://doi.org/10.1145/2596631.2596633

Published: 13 April 2014 Publication History

Check for updates

24 666

PDF/eReader

Abstract

Conflict-Free Replicated Data-Types (CRDTs) [6] provide greater safety properties to eventually-consistent distributed systems without requiring synchronization. CRDTs ensure that concurrent, uncoordinated updates have deterministic outcomes via the properties of bounded join-semilattices.

- State-based CRDT in Riak
- ***And buggy!***

Buggy Riak DT Map

Unsound Optimizations

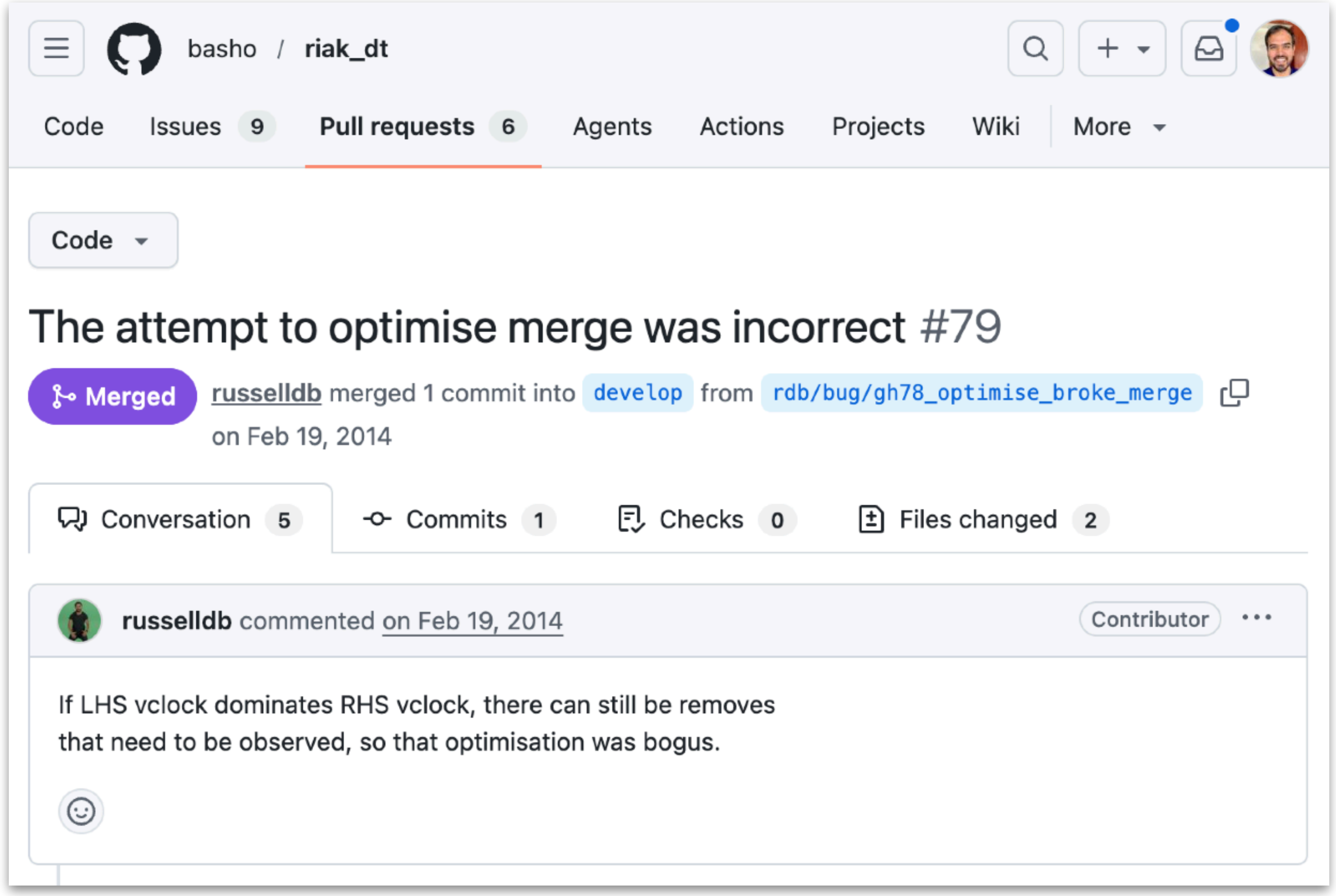
Implementing CRDTs can be quite challenging. Not only does the developer who is implementing CRDTs must ensure that updates to lattices are *inflationary* – ensuring the data structure is always moving up the lattice with respect to each change – but also that the merge function must be both *deterministic* and compute the *least-upper-bound.* We have even got this wrong a few times ourselves.

https://christophermeiklejohn.com/erlang/lasp/2019/03/08/monotonicity.html

Buggy Riak DT Map

> **Unsound Optimizations**
>
> Implementing CRDTs can be quite challenging. Not only does the developer who is implementing CRDTs must ensure that updates to lattices are *inflationary* – ensuring the data structure is always moving up the lattice with respect to each change – but also that the merge function must be both *deterministic* and compute the *least-upper-bound*. We have even got this wrong a few times ourselves.

https://christophermeiklejohn.com/erlang/lasp/2019/03/08/monotonicity.html



Not an isolated incident

Martin Kleppmann

Student Projects About/Contact Supporters

Interleaving anomalies in collaborative text editors

Martin Kleppmann, Victor B. F. Gomes, Dominic P. Mulligan, and Alastair R. Beresford

6th Workshop on Principles and Practice of Consistency for Distributed Data (PaPoC), Dresden, Germany, March 2019.

- Paper (open access)
- Paper (version of record)
- doi:10.1145/3301419.3323972
- Slides

Errata

A few years after publication, we learned about some serious flaws in this paper:

- The definition of non-interleaving in Section 2.1 cannot be guaranteed by any algorithm.
- The algorithm presented in Section 3.1 does not guarantee convergence.

Please see the Fugue paper for more details.

Verifying RDTs

- The goal is to ***verify*** the correctness of RDTs

Verifying RDTs

- The goal is to ***verify*** the correctness of RDTs
- What's common between **designing RDTs** and **pen-and-paper proofs**?

Verifying RDTs

- The goal is to ***verify*** the correctness of RDTs
- What's common between **designing RDTs** and **pen-and-paper proofs**?
  - I make mistakes in both!

Verifying RDTs

- The goal is to ***verify*** the correctness of RDTs
- What's common between **designing RDTs** and **pen-and-paper proofs**?
  - I make mistakes in both!
- Machine-checked proofs
  - Use SMT-solvers, proof-oriented PLs, proof-assistants

Verifying RDTs

- The goal is to ***verify*** the correctness of RDTs
- What's common between **designing RDTs** and **pen-and-paper proofs**?
    - I make mistakes in both!
- Machine-checked proofs
    - Use SMT-solvers, proof-oriented PLs, proof-assistants

RESEARCH-ARTICLE | OPEN ACCESS

Certified mergeable replicated data types

Peepul

Authors: Vimala Soundarapandian, Adharsh Kamath, Kartik Nagar, KC Sivaramakrishnan | Authors Info & Claims

PLDI 2022: Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation
Pages 332 - 347 • https://doi.org/10.1145/3519939.3523735

RESEARCH-ARTICLE | OPEN ACCESS |

Automatically Verifying Replication-Aware Linearizability

Authors: Vimala Soundarapandian, Kartik Nagar, Aseem Rastogi, KC Sivaramakrishnan | Authors Info & Claims

Proceedings of the ACM on Programming Languages, Volume 9, Issue OOPSLA1
Article No.: 111, Pages 871 - 897 • https://doi.org/10.1145/3720452

Published: 09 April 2025 Publication History | Check for updates

Neem

SAL: Multi-modal Verification of Replicated Data Types

Pranav Ramesh, Indian Institute of Technology, Madras, Chennai, India, cs22b015@smail.iitm.ac.in
Vimala Soundarapandian, Indian Institute of Technology, Madras, Chennai, India, cs19d750@smail.iitm.ac.in
KC Sivaramakrishnan, Indian Institute of Technology, Madras, Chennai, India, kcsrk@cse.iitm.ac.in

Abstract

Designing correct replicated data types (RDTs) is challenging because replicas evolve independently and must be merged while preserving application intent. A promising approach is correct-by-construction development in a proof-oriented programming language such as F*, Dafny and Lean, where

and Mergeable Replicated Data Types (MRDTs) [9] are widely used to implement such replicated state. However, designing correct RDTs is subtle. Even well-known designs such as Replicated Growable Arrays (RGA) [10] have had serious anomalies discovered after publication[1].

Replication-aware (RA) linearizability [21] provides a principled correctness condition for CRDTs, relating replica exe-

Convergence alone is insufficient

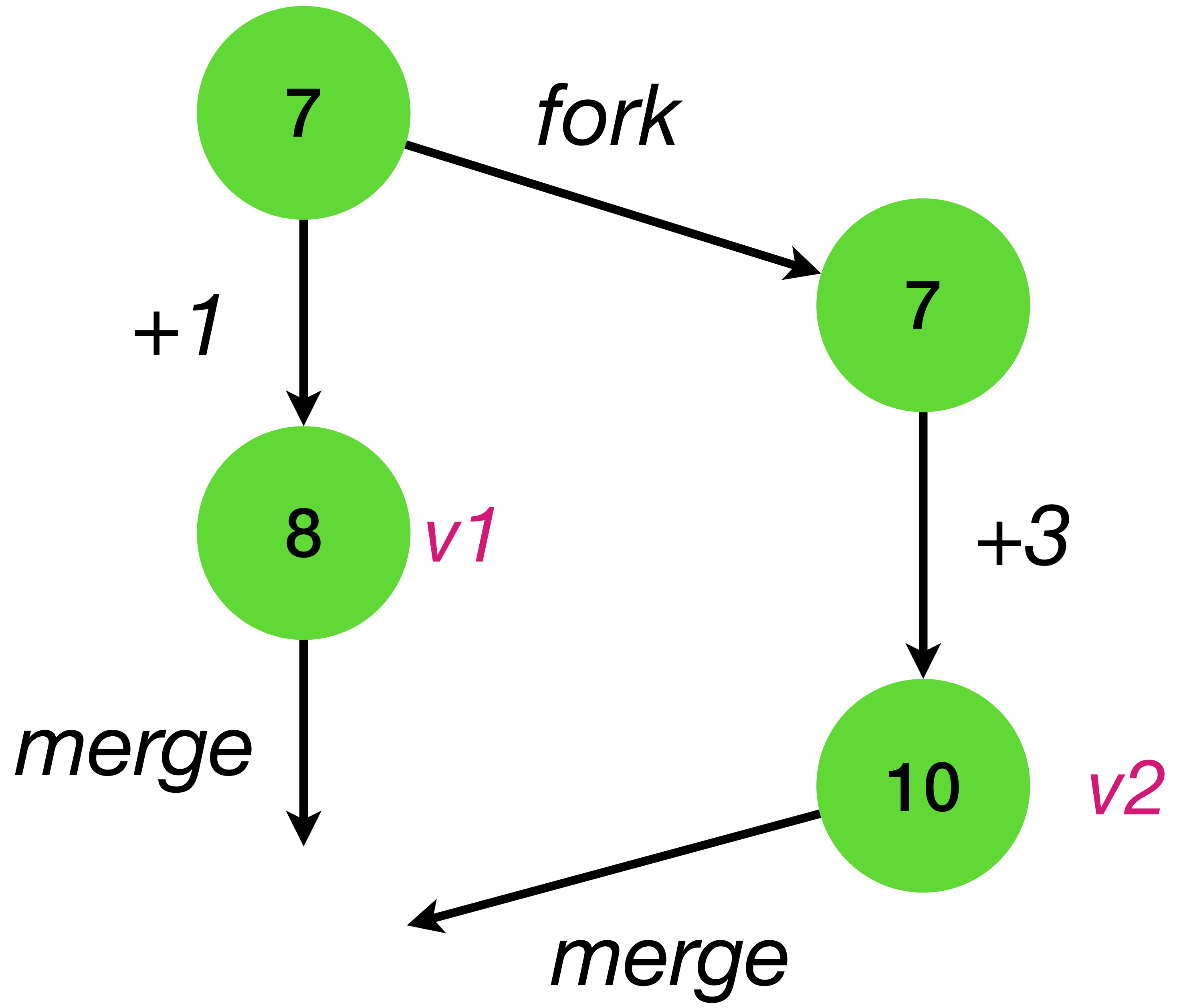
Merge function μ

$\mu(a, b) = \mu(b, a)$	Commutativity
$\mu(a, a) = a$	Idempotence
$\mu(\mu(a, b), c) = \mu(a, \mu(b, c))$	Associativity

Convergence alone is insufficient

Merge function μ

$\mu(a, b) = \mu(b, a)$	Commutativity
$\mu(a, a) = a$	Idempotence
$\mu(\mu(a, b), c) = \mu(a, \mu(b, c))$	Associativity



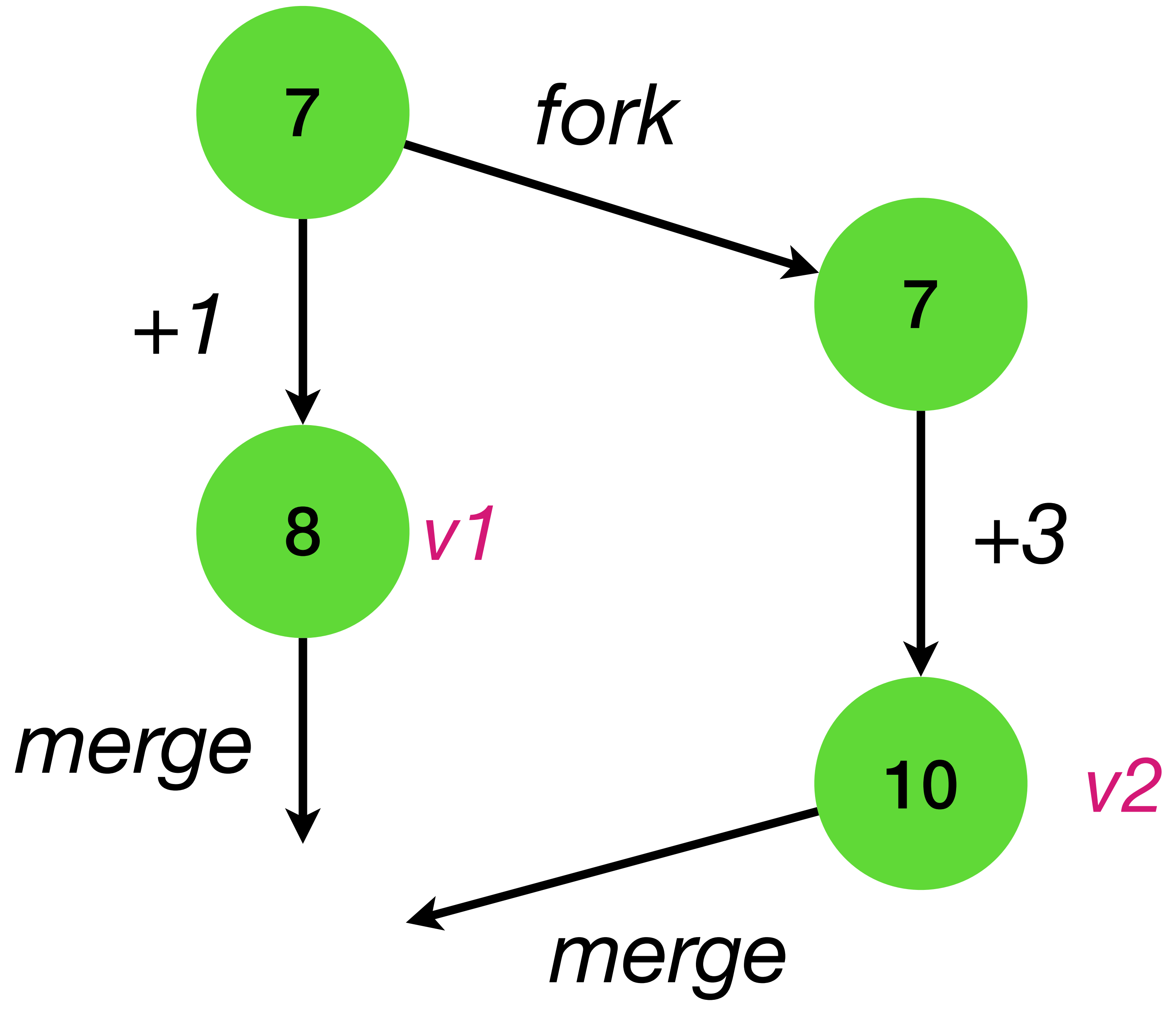
Convergence alone is insufficient

Merge function μ

$\mu(a,b) = \mu(b,a)$	Commutativity
$\mu(a,a) = a$	Idempotence
$\mu(\mu(a,b),c) = \mu(a,\mu(b,c))$	Associativity

Satisfies algebraic properties

```
let merge v1 v2 = max v1 v2
```



Convergence alone is insufficient

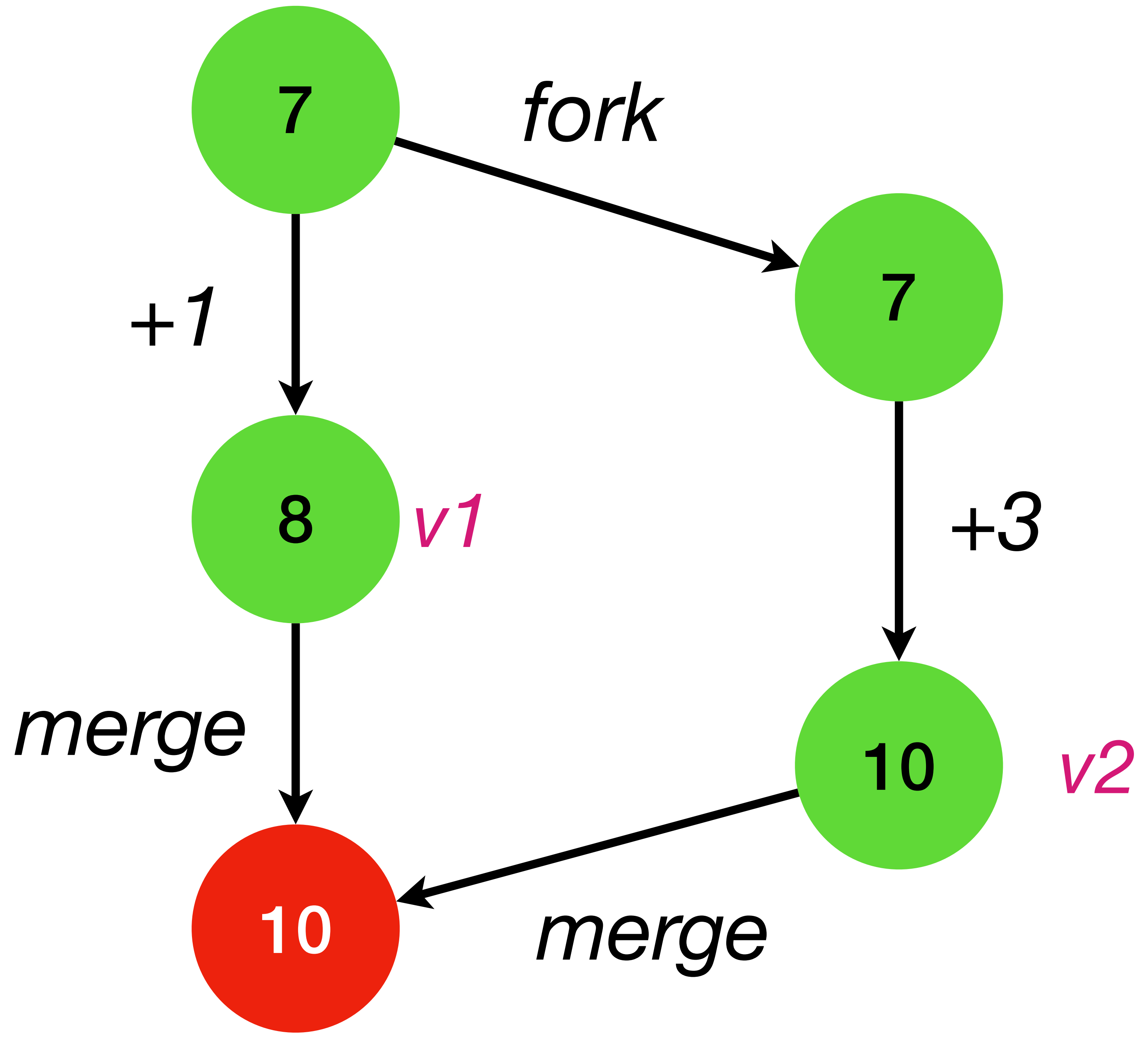
Merge function μ

$\mu(a,b) = \mu(b,a)$	Commutativity
$\mu(a,a) = a$	Idempotence
$\mu(\mu(a,b),c) = \mu(a,\mu(b,c))$	Associativity

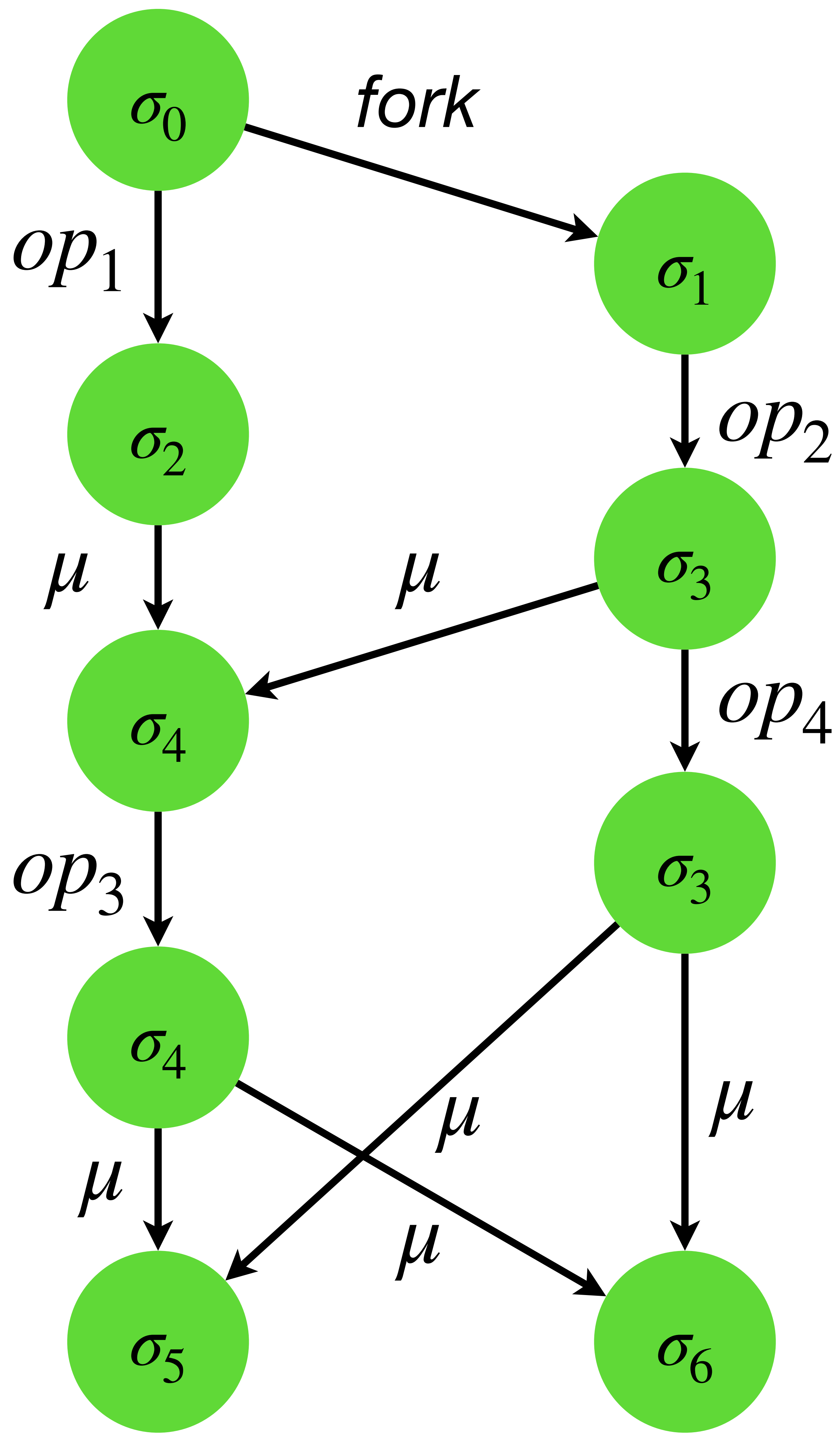
Satisfies algebraic properties

```
let merge v1 v2 = max v1 v2
```

Intent is not captured



Is there a better spec?



$$\sigma_5 = \sigma_6 = \text{linearization}(\{op_1, op_2, op_3, op_4\})\ \sigma_0$$

Replication-aware Linearizability

RESEARCH-ARTICLE

Replication-aware linearizability

Authors: Chao Wang, Constantin Enea, Suha Orhun Mutluergil, Gustavo Petri | Authors Info & Clai

PLDI 2019: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation
https://doi.org/10.1145/3314221.3314617

Replication-aware Linearizability

RESEARCH-ARTICLE

Replication-aware linearizability

Authors: Chao Wang, Constantin Enea, Suha Orhun Mutluergil, Gustavo Petri | Authors Info & Clai

PLDI 2019: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation
https://doi.org/10.1145/3314221.3314617

- Replica states should be a ***linearisation*** of observed ***update*** operations
  - Linearisation total order lo compatible with partially-ordered visibility relation vis
  - No real-time ordering requirement unlike traditional linearizability

Replication-aware Linearizability

RESEARCH-ARTICLE

Replication-aware linearizability

Authors: Chao Wang, Constantin Enea, Suha Orhun Mutluergil, Gustavo Petri | Authors Info & Clai

PLDI 2019: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation
https://doi.org/10.1145/3314221.3314617

- Replica states should be a ***linearisation*** of observed ***update*** operations
  - Linearisation total order lo compatible with partially-ordered visibility relation vis
  - No real-time ordering requirement unlike traditional linearizability
- Payoff
  - If a replicated object is RA-linearizable, reason about it using ***sequential semantics***

Using RA-linearizability for verification

`add(a);`	`add(a);`
`rem(a);`	
`X = read();`	`Y = read();`

$$a \in X \implies a \in Y$$

Using RA-linearizability for verification

```
add(a);          ||  add(a);
rem(a);          ||
X = read();      ||  Y = read();
```

$$a \in X \implies a \in Y$$

- Since Add-wins set is RA-linearizable, you can use ***totally ordered trace*** and the ***sequential reasoning***

Using RA-linearizability for verification

`add(a);`	`add(a);`
`rem(a);`	
`X = read();`	`Y = read();`

$$a \in X \implies a \in Y$$

- Since Add-wins set is RA-linearizable, you can use ***totally ordered trace*** and the ***sequential reasoning***

`add(a);`	`rem(a);`	`add(a);`	`X = read();`	`Y = read()`
{a}	{}	{a}	X = {a}	Y = {a}

Using RA-linearizability for verification

`add(a);`	`add(a);`
`rem(a);`	
`X = read();`	`Y = read();`

$$a \in X \implies a \in Y$$

- Let's prove by contradiction
  - Make $a \in X$ true and $a \in Y$ false

Using RA-linearizability for verification

```
add(a);        ||  add(a);
rem(a);        ||
X = read();    ||  Y = read();
```

$$a \in X \implies a \in Y$$

- Let’s prove by contradiction
  - Make $a \in X$ true and $a \in Y$ false

Using RA-linearizability for verification

```
add(a);            add(a);
rem(a);
X = read();        Y = read();
```

$$a \in X \implies a \in Y$$

- Let’s prove by contradiction
    - Make $a \in X$ true and $a \in Y$ false

Using RA-linearizability for verification

`add(a);`	`add(a);`
`rem(a);`	
`X = read();`	`Y = read();`

$$a \in X \implies a \in Y$$ ✓

- Let’s prove by contradiction
  - Make $a \in X$ true and $a \in Y$ false

Replication-aware Linearizability

RESEARCH-ARTICLE

Replication-aware linearizability

Authors: Chao Wang, Constantin Enea, Suha Orhun Mutluergil, Gustavo Petri | Authors Info & Clai

PLDI 2019: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation
https://doi.org/10.1145/3314221.3314617

- Presented a proof methodology to show that a CRDT is linearisable
- Not automated or mechanised

Automatic verification of RDTs

- Definition of RA-linearizability for MRDTs
- A novel induction scheme for MRDTs and state-based CRDTs to ***automatically*** verify RA-linearizability
- Implemented in F* and ***Lean***

Automatic verification of RDTs

- Definition of RA-linearizability for MRDTs
- A novel induction scheme for MRDTs and state-based CRDTs to ***automatically*** verify RA-linearizability
- Implemented in F* and ***Lean***

RESEARCH-ARTICLE | OPEN ACCESS |

Automatically Verifying Replication-Aware Linearizability

Authors: Vimala Soundarapandian, Kartik Nagar, Aseem Rastogi, KC Sivaramakrishnan | Authors Info & Claims

Proceedings of the ACM on Programming Languages, Volume 9, Issue OOPSLA1
Article No.: 111, Pages 871 - 897 • https://doi.org/10.1145/3720452

Published: 09 April 2025 Publication History Check for updates

Neem

SAL: Multi-modal Verification of Replicated Data Types

Pranav Ramesh
Indian Institute of Technology, Madras
Chennai, India
cs22b015@smail.iitm.ac.in

Vimala Soundarapandian
Indian Institute of Technology, Madras
Chennai, India
cs19d750@smail.iitm.ac.in

KC Sivaramakrishnan
Indian Institute of Technology, Madras
Chennai, India
kcsrk@cse.iitm.ac.in

Abstract

Designing correct replicated data types (RDTs) is challenging because replicas evolve independently and must be merged while preserving application intent. A promising approach is correct-by-construction development in a proof-oriented programming language such as F*, Dafny and Lean, where

and Mergeable Replicated Data Types (MRDTs) [9] are widely used to implement such replicated state. However, designing correct RDTs is subtle. Even well-known designs such as Replicated Growable Arrays (RGA) [10] have had serious anomalies discovered after publication[1].

Replication-aware (RA) linearizability [21] provides a principled correctness condition for CRDTs, relating replica exe-

Specifying RDTs in Neem

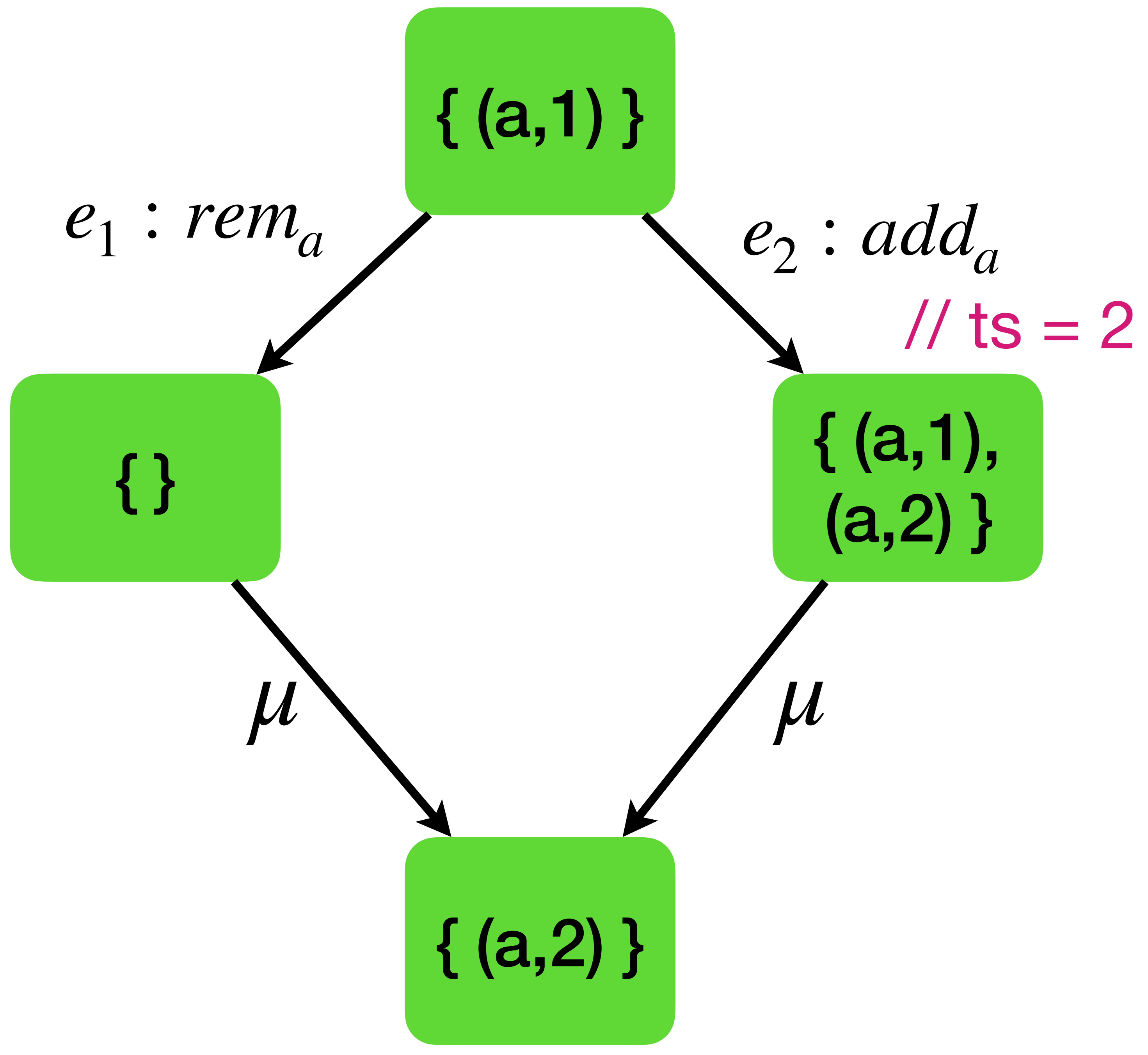
- Not all operations commute
  - **Add-wins set** — `add(a)` and `rem(a)` do not commute
  - Specify ordering using the **Conflict Resolution** relation $rc = \{(rem_a, add_a) \,|\, a \in \mathbb{E}\}$
  - Linearization order lo must be compatible with rc for concurrent events

Specifying RDTs in Neem

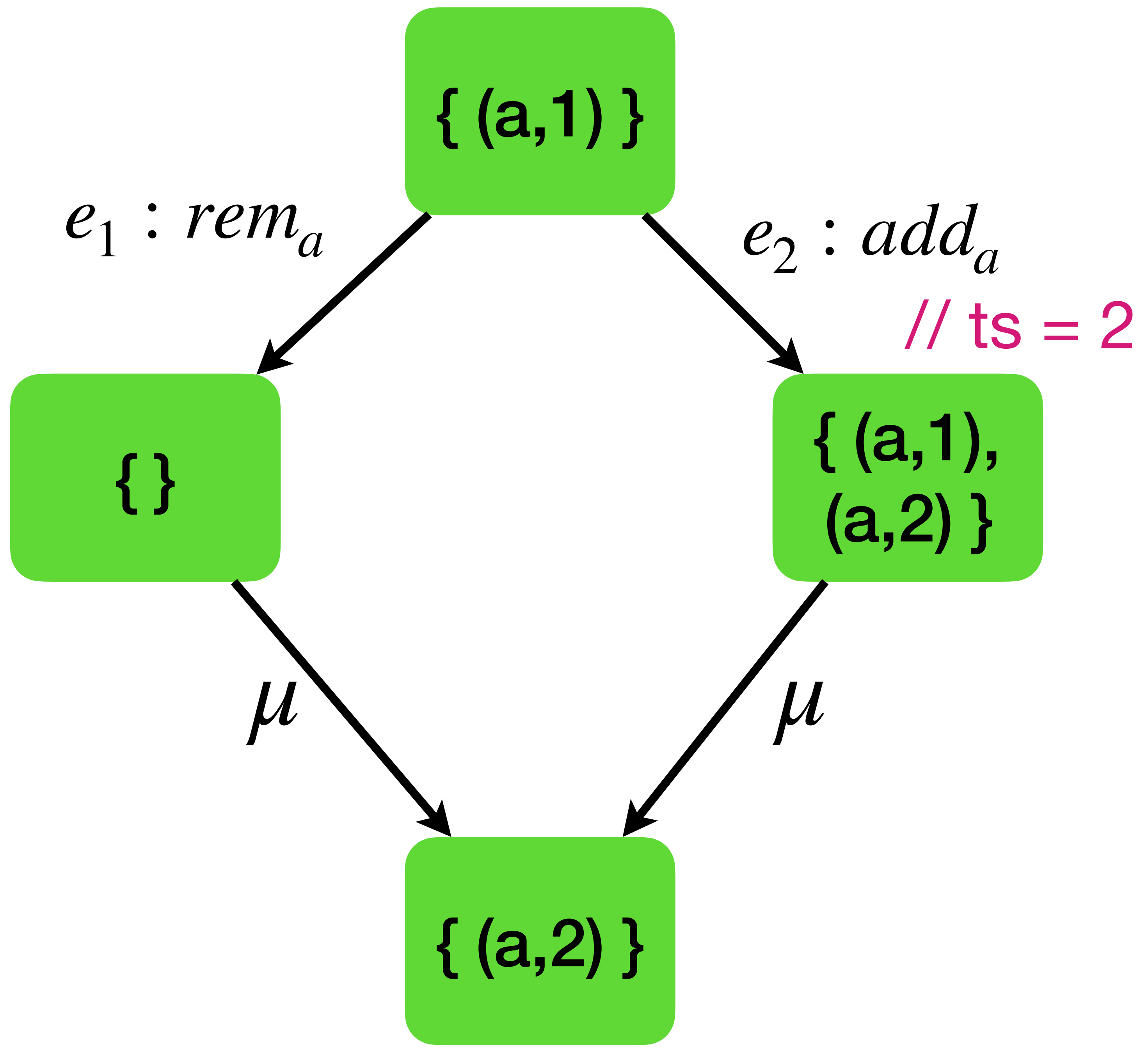
- Not all operations commute
  - **Add-wins set** — `add(a)` and `rem(a)` do not commute
  - Specify ordering using the **Conflict Resolution** relation $rc = \{(rem_a, add_a) \,|\, a \in \mathbb{E}\}$
  - Linearization order lo must be compatible with rc for concurrent events
- Neem developers provide
  - MRDT = Sequential Data Type + 3-way merge
  - Conflict Resolution rc relation

Add-wins Set

Add-wins Set

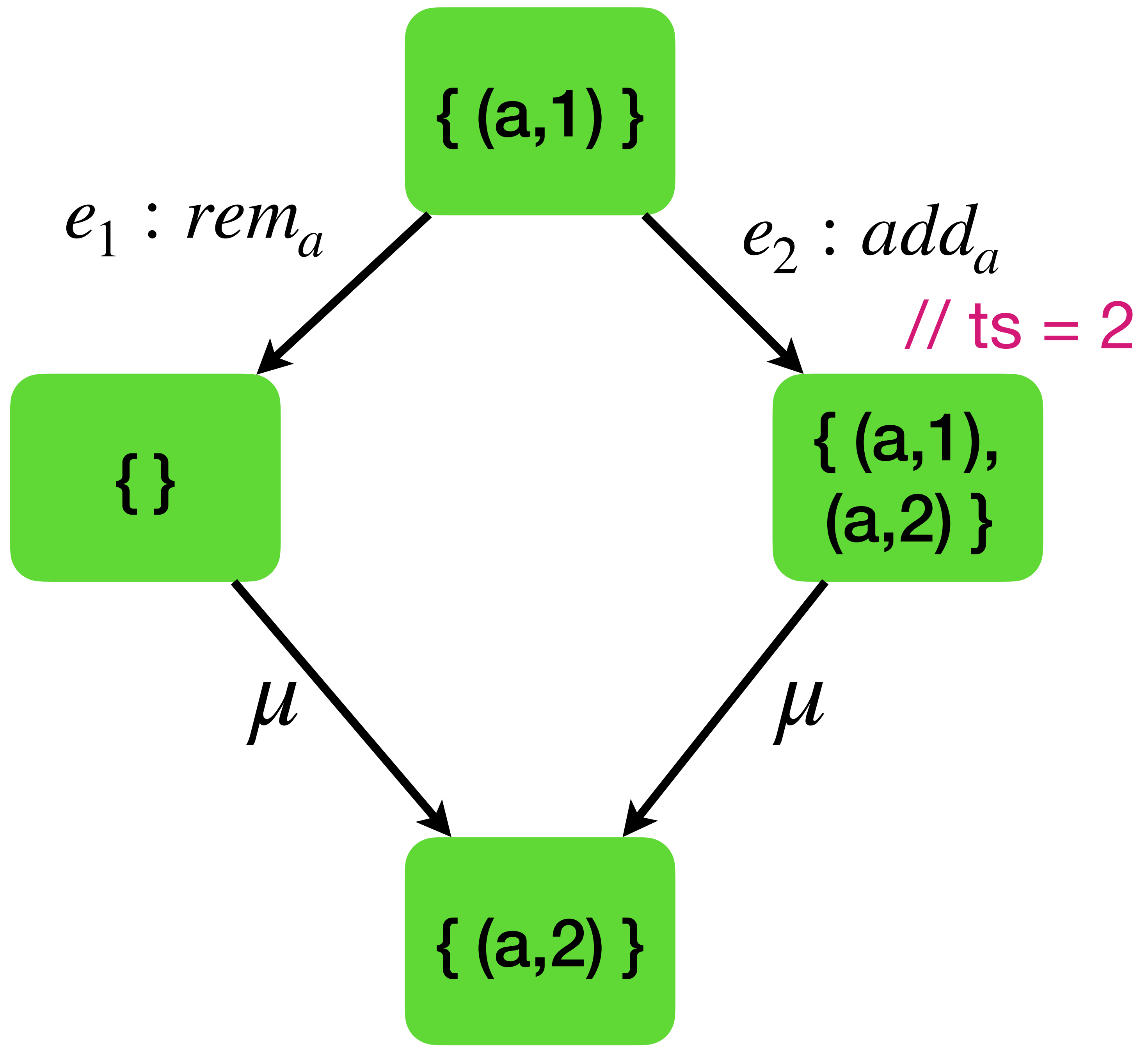


Add-wins Set



$$rc = \{(rem_a, add_a) \mid a \in \mathbb{E}\}$$

Add-wins Set



$$rc = \{(rem_a, add_a) \,|\, a \in \mathbb{E}\}$$

$$\{(a,2)\} = e_2(e_1\{(a,1)\}) = add_a(rem_a\{(a,1)\})$$

Linearization order

$$\forall e_1, e_2 \in \mathcal{E}_C.\ e_1 \xrightarrow{\mathsf{lo_C}} e_2 \Leftrightarrow (e_1 \xrightarrow{\mathsf{vis(C)}} e_2 \wedge \neg e_1 \rightleftarrows e_2)$$

$$\vee\ (e_1 \mid\mid_C e_2 \wedge e_1 \xrightarrow{\mathsf{rc}} e_2 \wedge \neg(\exists e_3 \in \mathcal{E}.\ e_2 \xrightarrow{\mathsf{vis(C)}} e_3 \wedge \neg e_2 \rightleftarrows e_3))$$

Linearization order

$$\forall e_1, e_2 \in \mathcal{E}_C.\ e_1 \xrightarrow{\text{lo}_\text{C}} e_2 \Leftrightarrow \left(e_1 \xrightarrow{\text{vis(C)}} e_2 \land \neg e_1 \rightleftarrows e_2\right)$$
$$\lor \left(e_1 \,||_C\, e_2 \land e_1 \xrightarrow{\text{rc}} e_2 \land \neg(\exists e_3 \in \mathcal{E}.\ e_2 \xrightarrow{\text{vis(C)}} e_3 \land \neg e_2 \rightleftarrows e_3)\right)$$

lo must preserve visibility unless the events commute

Linearization order

$$\forall e_1, e_2 \in \mathcal{E}_C.\ e_1 \xrightarrow{\mathsf{lo}_C} e_2 \Leftrightarrow \boxed{(e_1 \xrightarrow{\mathsf{vis}(C)} e_2 \wedge \neg e_1 \rightleftarrows e_2)}$$
$$\vee\ (e_1 \parallel_C e_2 \wedge e_1 \xrightarrow{\mathsf{rc}} e_2 \wedge \neg(\exists e_3 \in \mathcal{E}.\ e_2 \xrightarrow{\mathsf{vis}(C)} e_3 \wedge \neg e_2 \rightleftarrows e_3))$$

lo must preserve visibility unless the events commute

{ }

$e_1 : add_a$ (with $ts = 1$)

{ (a,1) }

$e_2 : add_b$ (with $ts = 2$)

{ (a,1), (b,2) }

$$e_1(e_2(\varnothing)) = e_2(e_1(\varnothing)) = \{(a,1), (b,2)\}$$

Linearization order

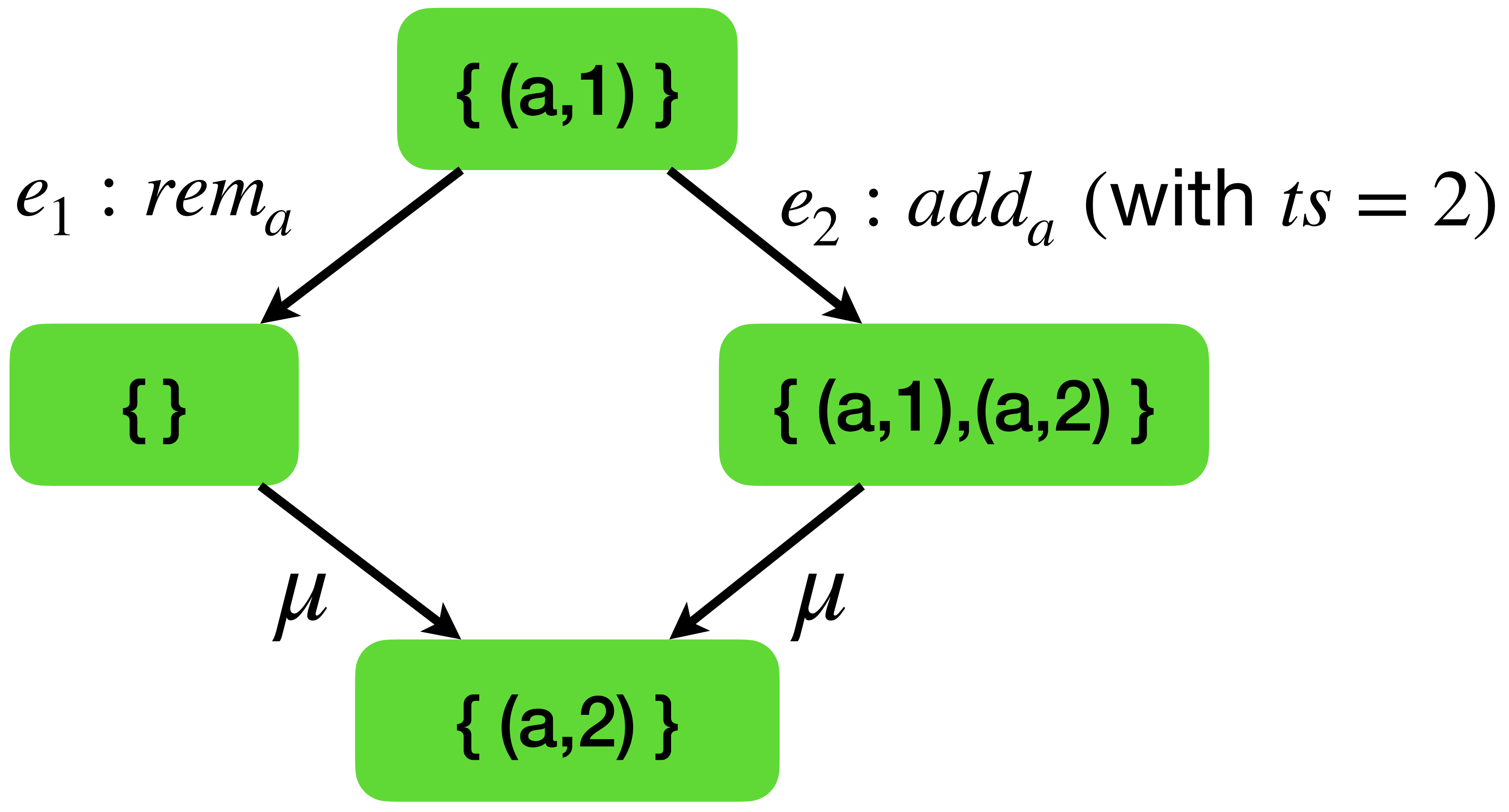
$$\forall e_1, e_2 \in \mathcal{E}_C.\ e_1 \xrightarrow{\mathsf{lo}_C} e_2 \Leftrightarrow (e_1 \xrightarrow{\mathsf{vis}(C)} e_2 \wedge \neg e_1 \rightleftarrows e_2)$$
$$\vee\ \boxed{(e_1 \,||_C\, e_2 \wedge e_1 \xrightarrow{\mathsf{rc}} e_2} \wedge \neg(\exists e_3 \in \mathcal{E}.\ e_2 \xrightarrow{\mathsf{vis}(C)} e_3 \wedge \neg e_2 \rightleftarrows e_3))$$

Concurrent event non-commuting events must be ordered according to rc

Linearization order

$$\forall e_1, e_2 \in \mathcal{E}_C.\ e_1 \xrightarrow{\mathsf{loc}_C} e_2 \Leftrightarrow (e_1 \xrightarrow{\mathsf{vis}(C)} e_2 \wedge \neg e_1 \rightleftarrows e_2)$$
$$\vee\ (e_1 \,||_C\, e_2 \wedge e_1 \xrightarrow{\mathsf{rc}} e_2 \wedge \neg(\exists e_3 \in \mathcal{E}.\ e_2 \xrightarrow{\mathsf{vis}(C)} e_3 \wedge \neg e_2 \rightleftarrows e_3))$$

Concurrent event non-commuting events must be ordered according to rc



$$\{(a,2)\} = e_2(e_1\{(a,1)\})$$

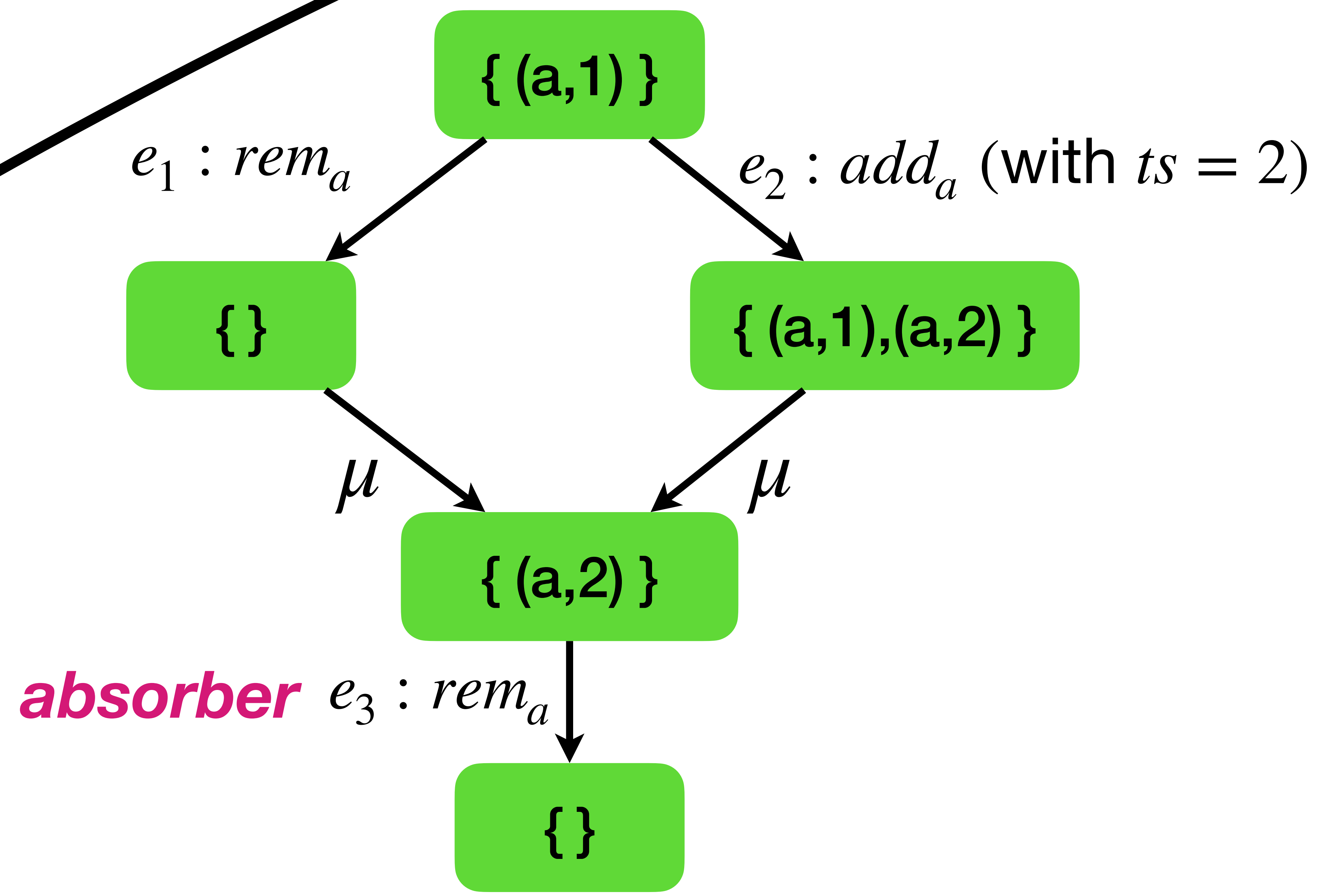
Linearization order

$$\forall e_1, e_2 \in \mathcal{E}_C.\ e_1 \xrightarrow{\mathsf{lo}_\mathsf{C}} e_2 \Leftrightarrow (e_1 \xrightarrow{\mathsf{vis(C)}} e_2 \wedge \neg e_1 \rightleftarrows e_2)$$

$$\vee\ (e_1 \,\|_C\, e_2 \wedge e_1 \xrightarrow{\mathsf{rc}} e_2 \wedge \neg(\exists e_3 \in \mathcal{E}.\ e_2 \xrightarrow{\mathsf{vis(C)}} e_3 \wedge \neg e_2 \rightleftarrows e_3))$$

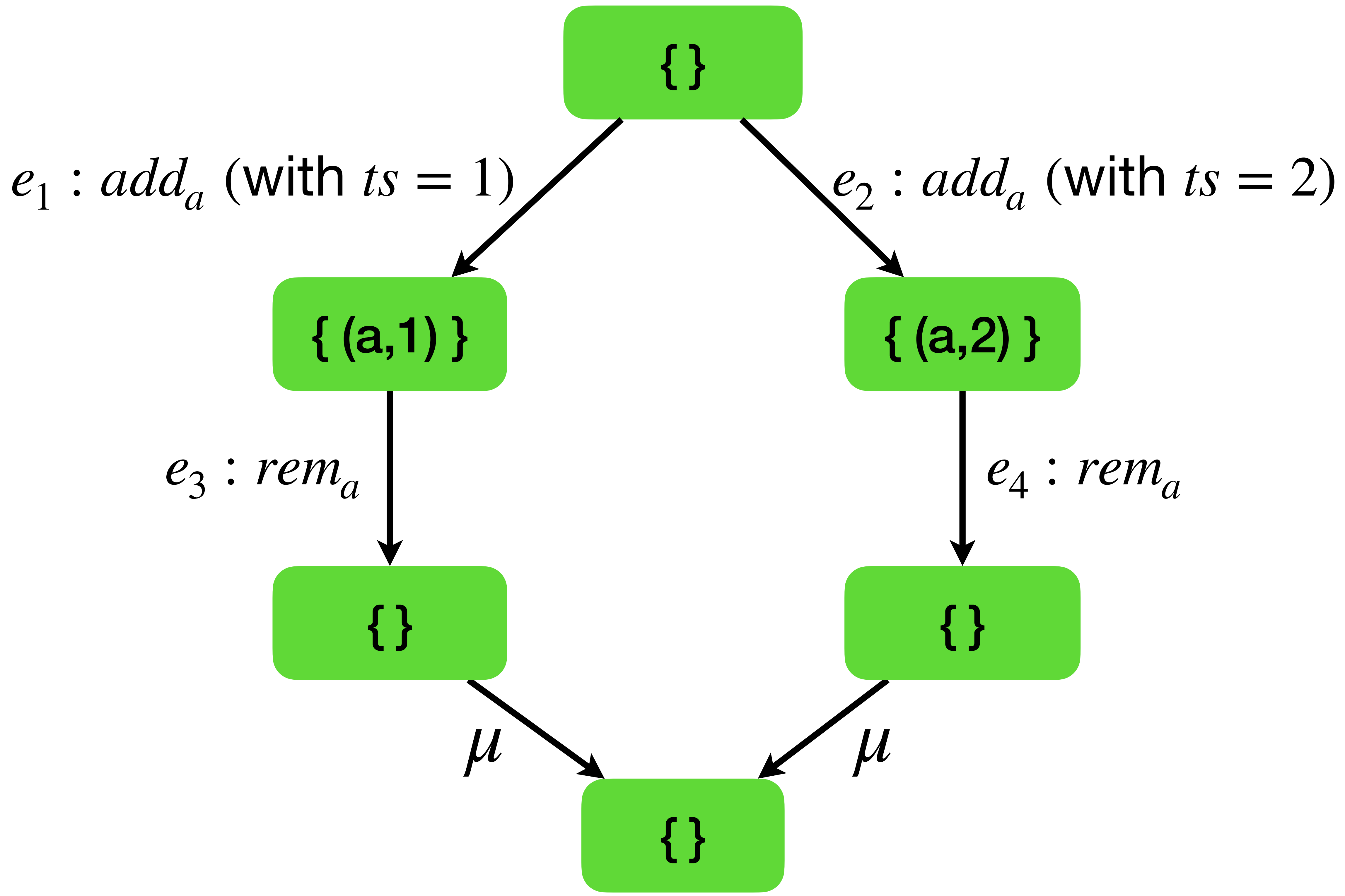
Concurrent event non-commuting events must be ordered according to rc

Unless there is a subsequent "absorber"

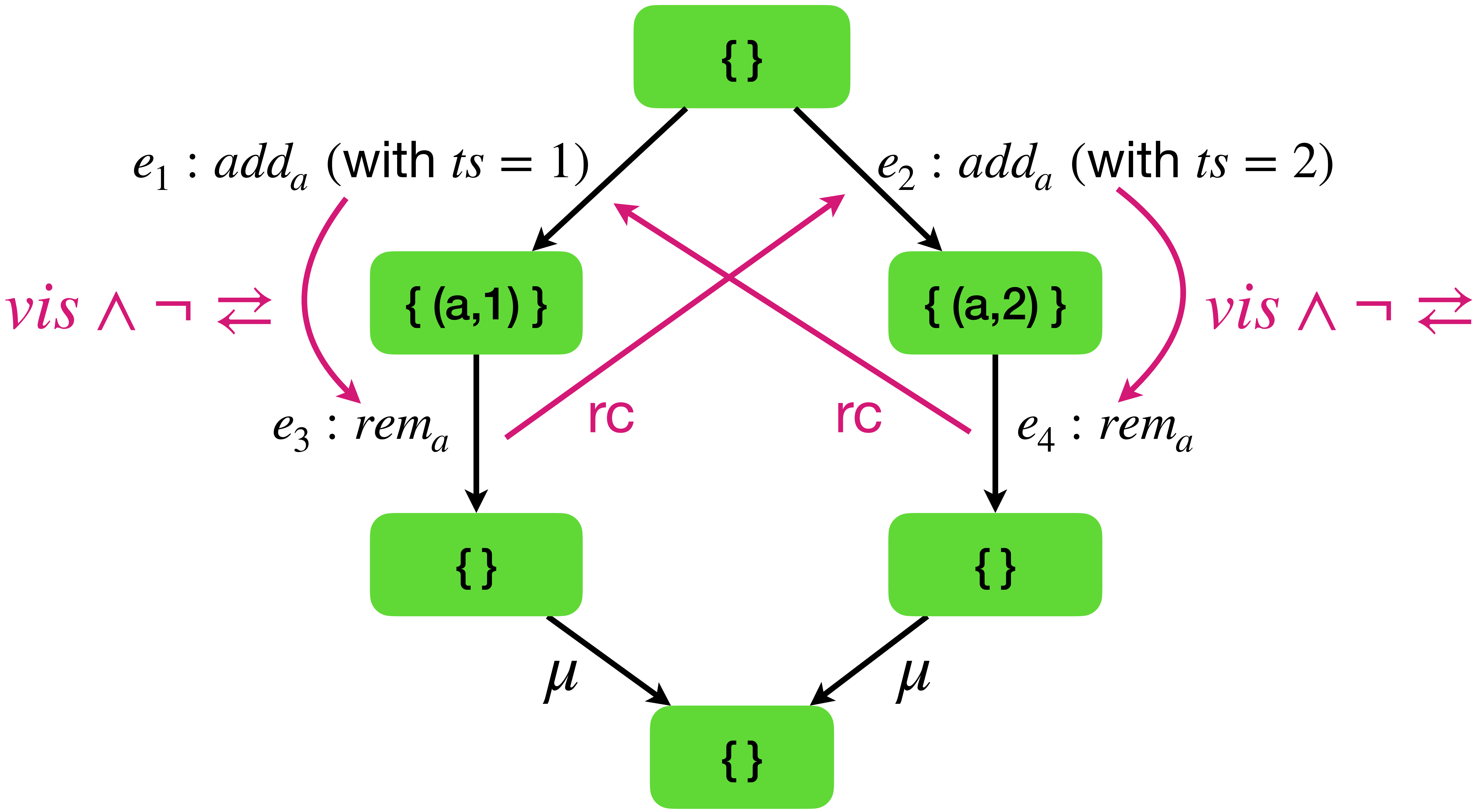


$$e_3(e_2(e_1\{(a,1)\}))) = e_3(e_2(e_1(\{(a,1)\}))) = \varnothing$$

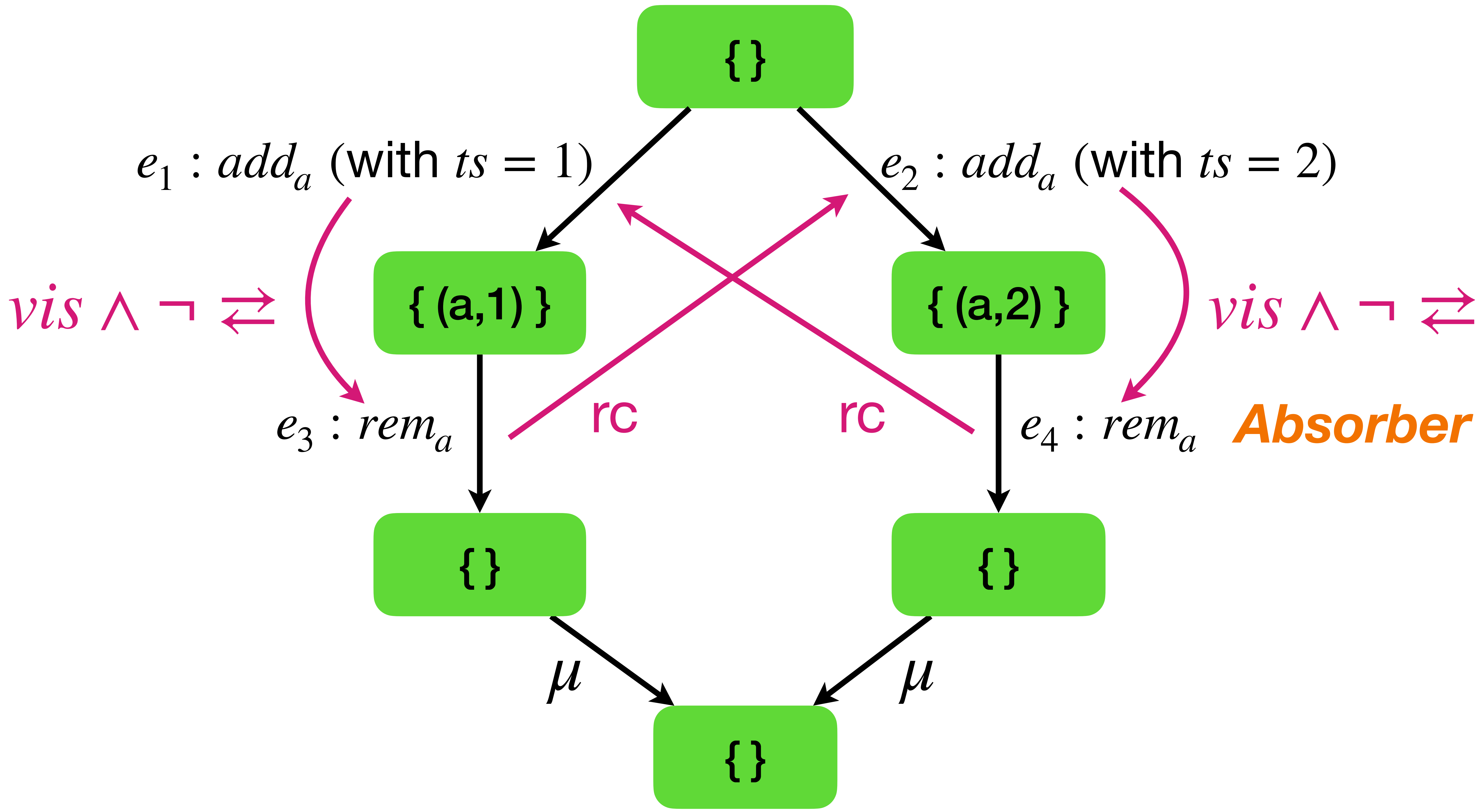
Where absorber helps



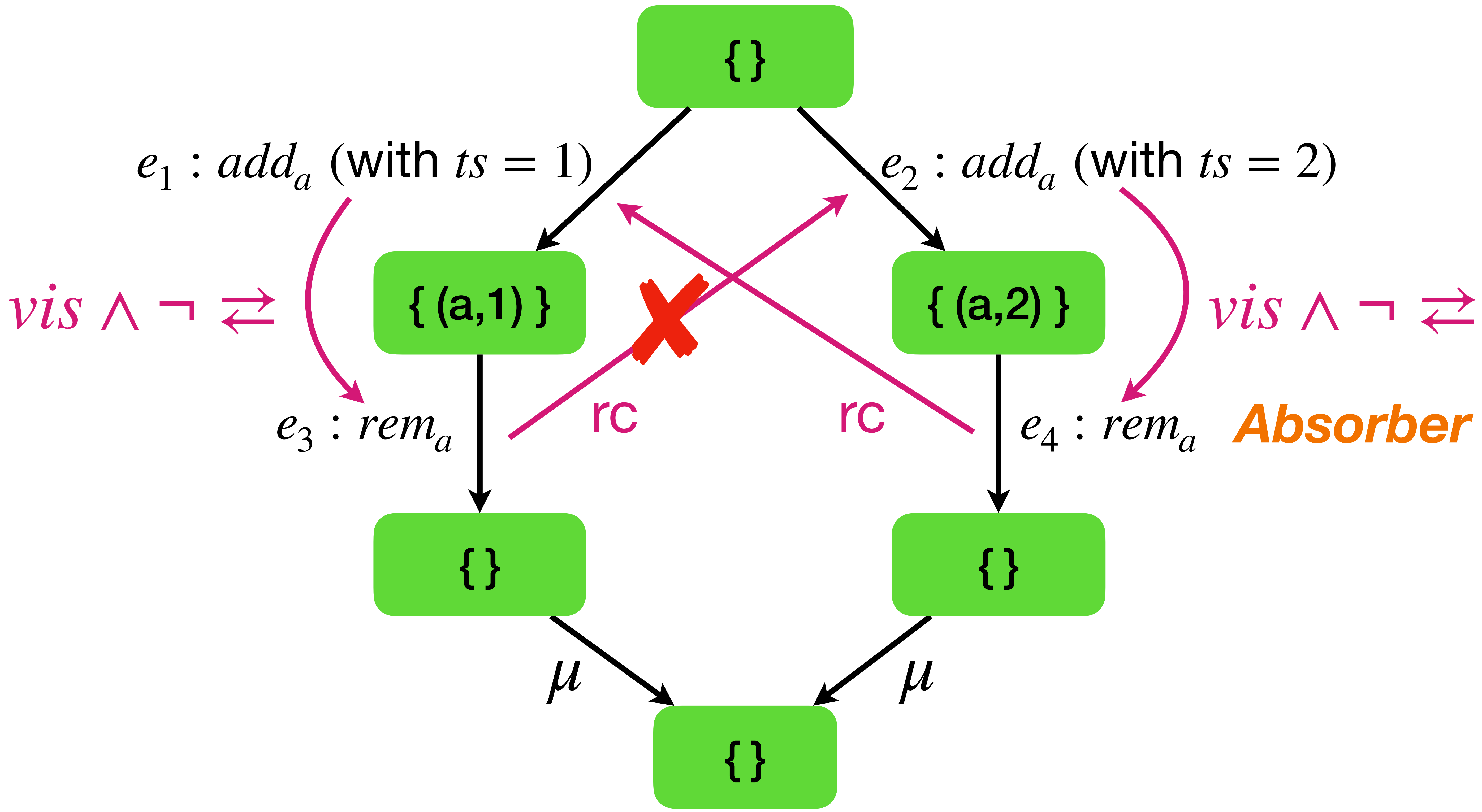
Where absorber helps



Where absorber helps



Where absorber helps



Proving RA-linearizability

Proving RA-linearizability

- Neem developers provide
  - MRDT = Sequential Data Type + 3-way merge + Conflict Resolution rc relation

Proving RA-linearizability

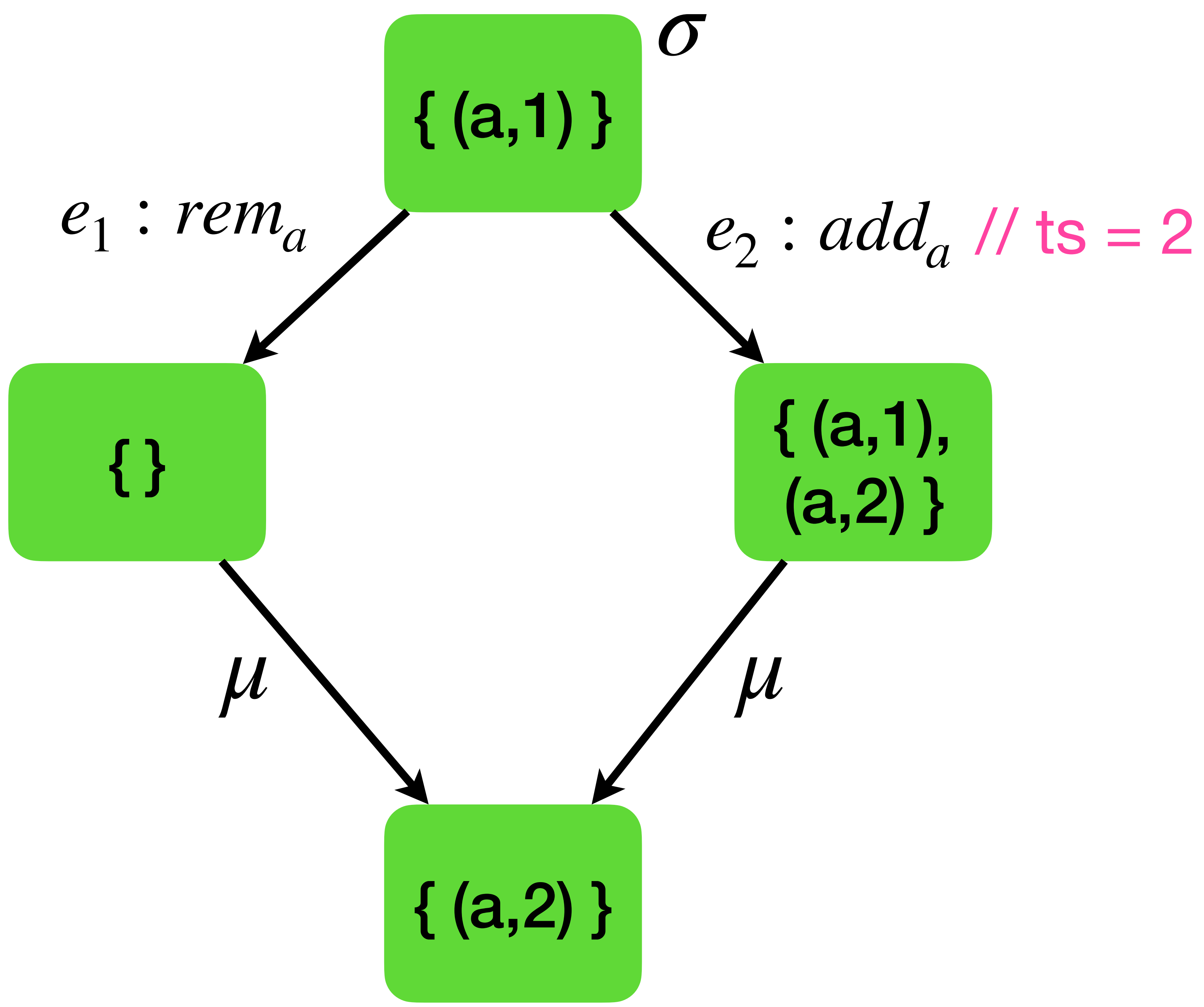
- Neem developers provide
  - MRDT = Sequential Data Type + 3-way merge + Conflict Resolution rc relation

THEOREM 4.7. *If an MRDT $\mathcal{D}$ satisfies the VCs ψ^**(BOTTOMUP-2-OP), *ψ^**(BOTTOMUP-1-OP), *ψ^**(BOTTOMUP-0-OP), MERGEIDEMPOTENCE *and* MERGECOMMUTATIVITY, *then $\mathcal{D}$ is linearizable.*

  - Meta theory — 24 VCs $\Longrightarrow$ RA-Linearizability
  - Automatically checked against 24 VCs using the SMT solver

Bottom up linearisation

$$rc = \{(rem_a, add_a) \mid a \in \mathbb{E}\}$$

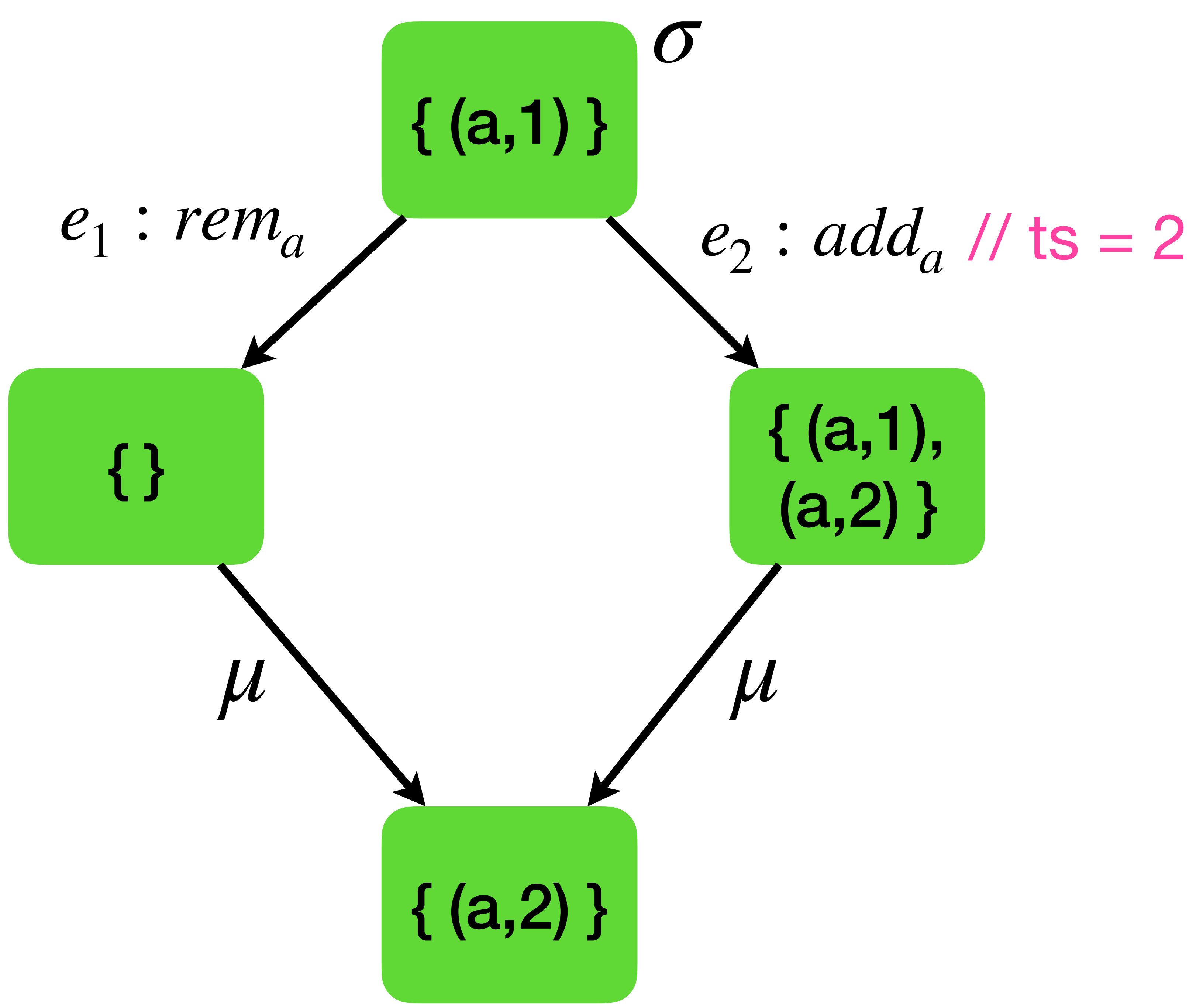


To show

$$\mu(\sigma, e_1(\sigma), e_2(\sigma)) = e_2(e_1(\sigma))$$

Bottom up linearisation

$$rc = \{(rem_a, add_a) \,|\, a \in \mathbb{E}\}$$



To show

$$\mu(\sigma, e_1(\sigma), e_2(\sigma)) = e_2(e_1(\sigma))$$

[BottomUp-2-OP]

$$\frac{e_1 \neq e_2 \qquad e_1 \xrightarrow{\mathsf{rc}} e_2 \vee e_1 \rightleftarrows e_2}{\mu(l, e_1(a), e_2(b)) = e_2(\mu(l, e_1(a), b))}$$

[BottomUp-1-OP]

$$\frac{(e_\top \neq \epsilon \wedge e_1 \neq e_\top) \vee (e_\top = \epsilon \wedge l = b)}{\mu(e_\top(l), e_1(a), e_\top(b)) = e_1(\mu(e_\top(l), a, e_\top(b)))}$$

[BottomUp-0-OP]

$$\mu(e_\top(l), e_\top(a), e_\top(b)) = e_\top(\mu(l, a, b))$$

[MergeIdempotence]

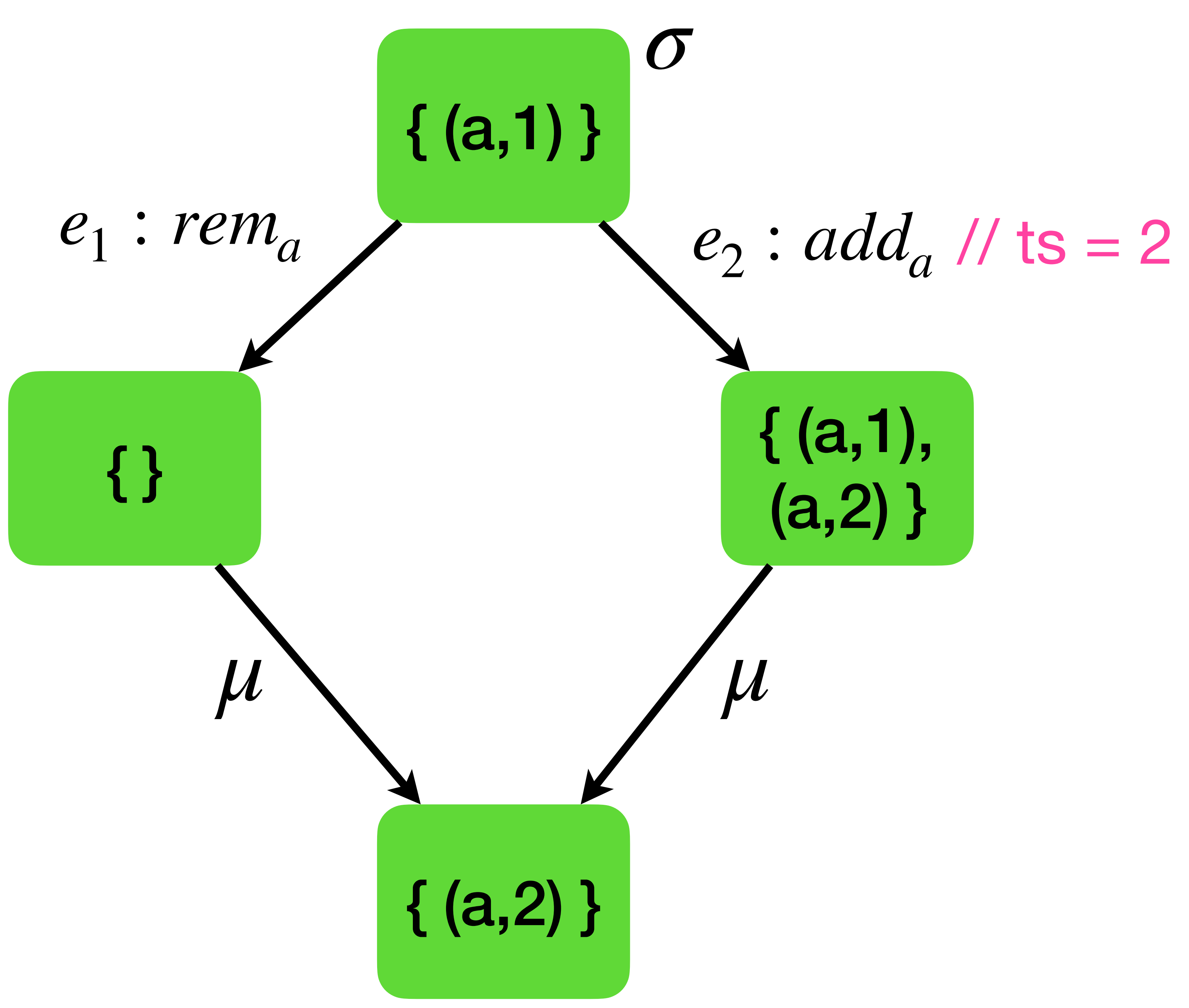
$$\mu(a, a, a) = a$$

[MergeCommutativity]

$$\mu(l, a, b) = \mu(l, b, a)$$

Bottom up linearisation

$$rc = \{(rem_a, add_a) \mid a \in \mathbb{E}\}$$



To show

$$\mu(\sigma, e_1(\sigma), e_2(\sigma)) = e_2(e_1(\sigma))$$

[BottomUp-2-OP]

$$\frac{e_1 \neq e_2 \quad e_1 \xrightarrow{\text{rc}} e_2 \lor e_1 \rightleftarrows e_2}{\mu(l, e_1(a), e_2(b)) = e_2(\mu(l, e_1(a), b))}$$

[BottomUp-1-OP]

$$\frac{(e_\top \neq \epsilon \land e_1 \neq e_\top) \lor (e_\top = \epsilon \land l = b)}{\mu(e_\top(l), e_1(a), e_\top(b)) = e_1(\mu(e_\top(l), a, e_\top(b)))}$$

[BottomUp-0-OP]

$$\mu(e_\top(l), e_\top(a), e_\top(b)) = e_\top(\mu(l, a, b))$$

[MergeIdempotence]

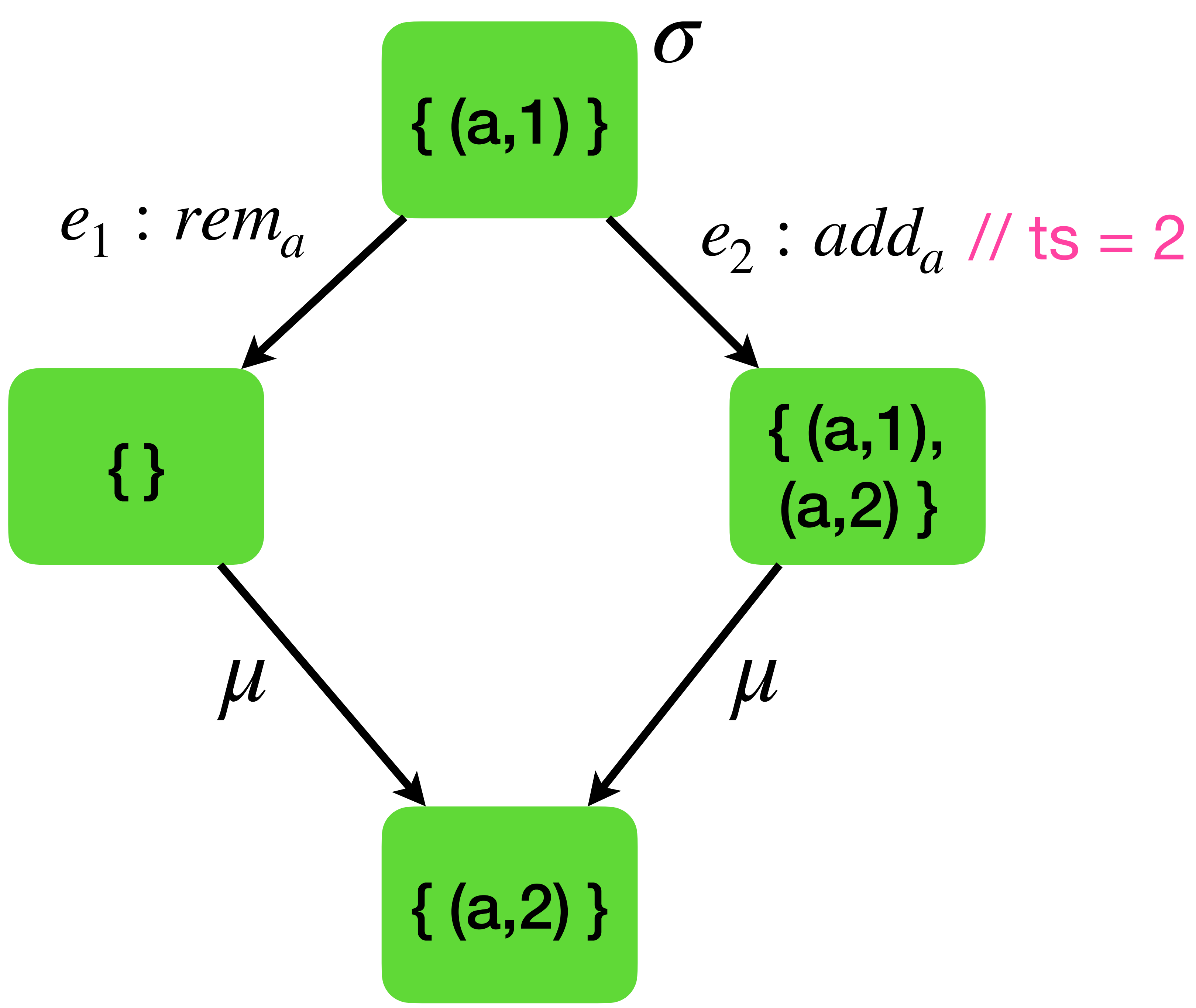
$$\mu(a, a, a) = a$$

[MergeCommutativity]

$$\mu(l, a, b) = \mu(l, b, a)$$

Bottom up linearisation

$$rc = \{(rem_a, add_a) \mid a \in \mathbb{E}\}$$



To show

$$e_2(\mu(\sigma, e_1(\sigma), \sigma)) = e_2(e_1(\sigma))$$

[BottomUp-2-OP]

$$\frac{e_1 \neq e_2 \quad e_1 \xrightarrow{\text{rc}} e_2 \lor e_1 \rightleftarrows e_2}{\mu(l, e_1(a), e_2(b)) = e_2(\mu(l, e_1(a), b))}$$

[BottomUp-1-OP]

$$\frac{(e_\top \neq \epsilon \land e_1 \neq e_\top) \lor (e_\top = \epsilon \land l = b)}{\mu(e_\top(l), e_1(a), e_\top(b)) = e_1(\mu(e_\top(l), a, e_\top(b)))}$$

[BottomUp-0-OP]

$$\mu(e_\top(l), e_\top(a), e_\top(b)) = e_\top(\mu(l, a, b))$$

[MergeIdempotence]

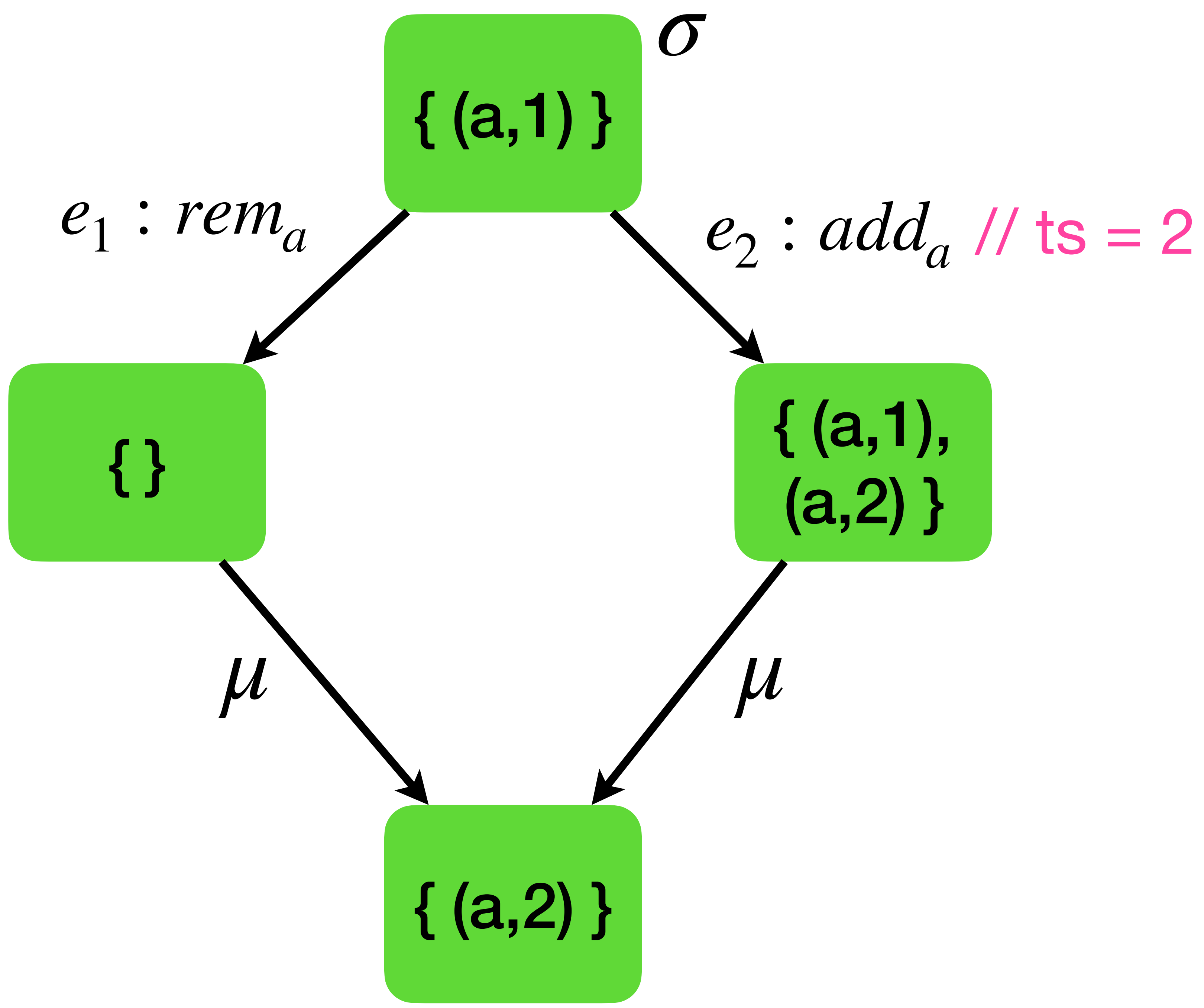
$$\mu(a, a, a) = a$$

[MergeCommutativity]

$$\mu(l, a, b) = \mu(l, b, a)$$

Bottom up linearisation

$$rc = \{(rem_a, add_a) \,|\, a \in \mathbb{E}\}$$



To show

$$e_2(\mu(\sigma, e_1(\sigma), \sigma)) = e_2(e_1(\sigma))$$

[BottomUp-2-OP]

$$\frac{e_1 \neq e_2 \qquad e_1 \xrightarrow{\mathsf{rc}} e_2 \vee e_1 \rightleftarrows e_2}{\mu(l, e_1(a), e_2(b)) = e_2(\mu(l, e_1(a), b))}$$

[BottomUp-1-OP]

$$\frac{(e_\top \neq \epsilon \wedge e_1 \neq e_\top) \vee (e_\top = \epsilon \wedge l = b)}{\mu(e_\top(l), e_1(a), e_\top(b)) = e_1(\mu(e_\top(l), a, e_\top(b)))}$$

[BottomUp-0-OP]

$$\mu(e_\top(l), e_\top(a), e_\top(b)) = e_\top(\mu(l, a, b))$$

[MergeIdempotence]

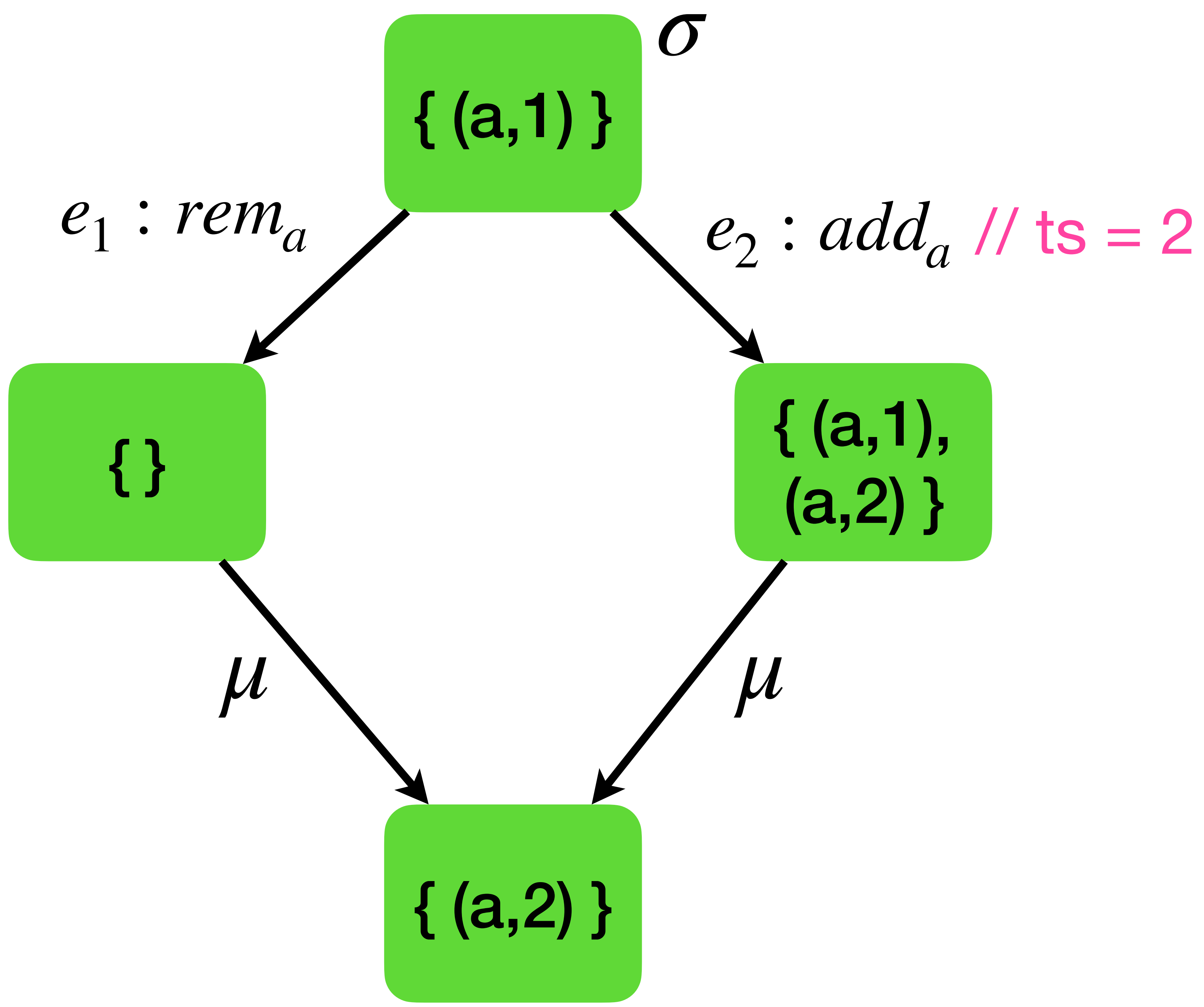
$$\mu(a, a, a) = a$$

[MergeCommutativity]

$$\mu(l, a, b) = \mu(l, b, a)$$

Bottom up linearisation

$$rc = \{(rem_a, add_a) \mid a \in \mathbb{E}\}$$



To show

$$e_2(e_1(\mu(\sigma, \sigma, \sigma))) = e_2(e_1(\sigma))$$

[BottomUp-2-OP]

$$\frac{e_1 \neq e_2 \quad e_1 \xrightarrow{\text{rc}} e_2 \vee e_1 \rightleftarrows e_2}{\mu(l, e_1(a), e_2(b)) = e_2(\mu(l, e_1(a), b))}$$

[BottomUp-1-OP]

$$\frac{(e_\top \neq \epsilon \wedge e_1 \neq e_\top) \vee (e_\top = \epsilon \wedge l = b)}{\mu(e_\top(l), e_1(a), e_\top(b)) = e_1(\mu(e_\top(l), a, e_\top(b)))}$$

[BottomUp-0-OP]

$$\mu(e_\top(l), e_\top(a), e_\top(b)) = e_\top(\mu(l, a, b))$$

[MergeIdempotence]

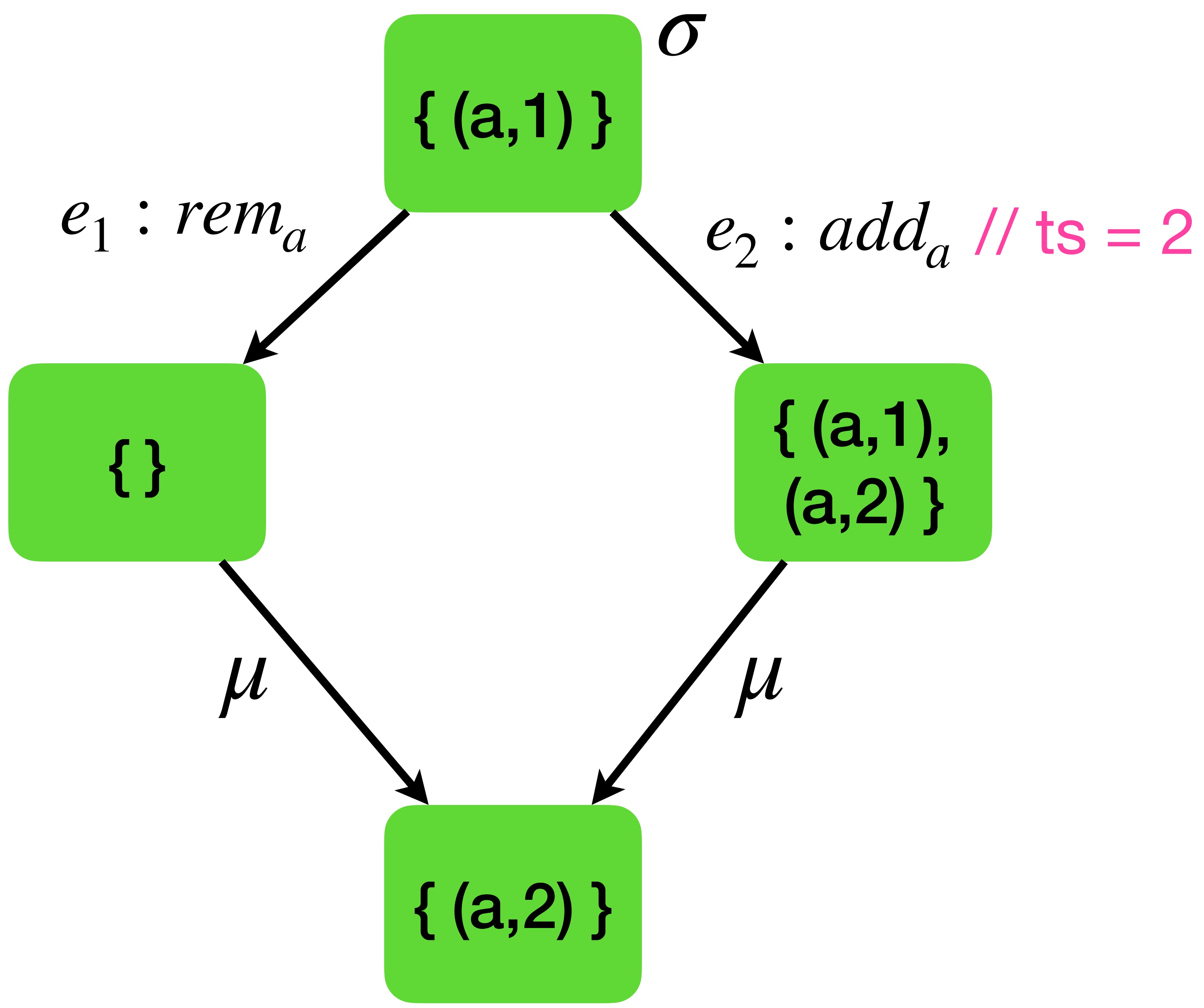
$$\mu(a, a, a) = a$$

[MergeCommutativity]

$$\mu(l, a, b) = \mu(l, b, a)$$

Bottom up linearisation

$$rc = \{(rem_a, add_a) \mid a \in \mathbb{E}\}$$



To show

$$e_2(e_1(\mu(\sigma, \sigma, \sigma))) = e_2(e_1(\sigma))$$

[BottomUp-2-OP]

$$\frac{e_1 \neq e_2 \qquad e_1 \xrightarrow{\text{rc}} e_2 \lor e_1 \rightleftarrows e_2}{\mu(l, e_1(a), e_2(b)) = e_2(\mu(l, e_1(a), b))}$$

[BottomUp-1-OP]

$$\frac{(e_\top \neq \epsilon \land e_1 \neq e_\top) \lor (e_\top = \epsilon \land l = b)}{\mu(e_\top(l), e_1(a), e_\top(b)) = e_1(\mu(e_\top(l), a, e_\top(b)))}$$

[BottomUp-0-OP]

$$\mu(e_\top(l), e_\top(a), e_\top(b)) = e_\top(\mu(l, a, b))$$

[MergeIdempotence]

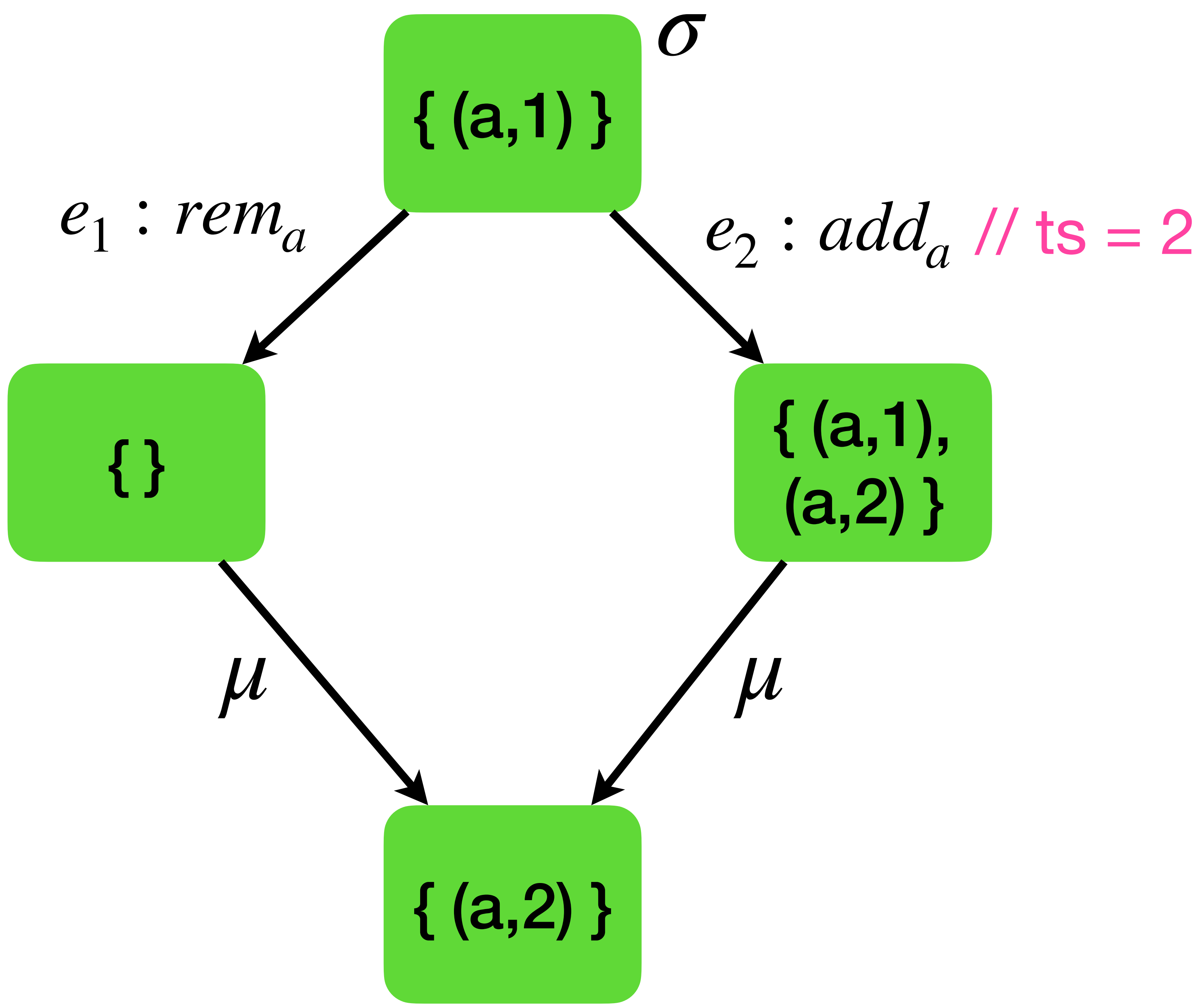
$$\mu(a, a, a) = a$$

[MergeCommutativity]

$$\mu(l, a, b) = \mu(l, b, a)$$

Bottom up linearisation

$$rc = \{(rem_a, add_a) \mid a \in \mathbb{E}\}$$



To show

$$e_2(e_1(\sigma)) = e_2(e_1(\sigma))$$

[BottomUp-2-OP]

$$\frac{e_1 \neq e_2 \qquad e_1 \xrightarrow{\text{rc}} e_2 \vee e_1 \rightleftarrows e_2}{\mu(l, e_1(a), e_2(b)) = e_2(\mu(l, e_1(a), b))}$$

[BottomUp-1-OP]

$$\frac{(e_\top \neq \epsilon \wedge e_1 \neq e_\top) \vee (e_\top = \epsilon \wedge l = b)}{\mu(e_\top(l), e_1(a), e_\top(b)) = e_1(\mu(e_\top(l), a, e_\top(b)))}$$

[BottomUp-0-OP]

$$\mu(e_\top(l), e_\top(a), e_\top(b)) = e_\top(\mu(l, a, b))$$

[MergeIdempotence]

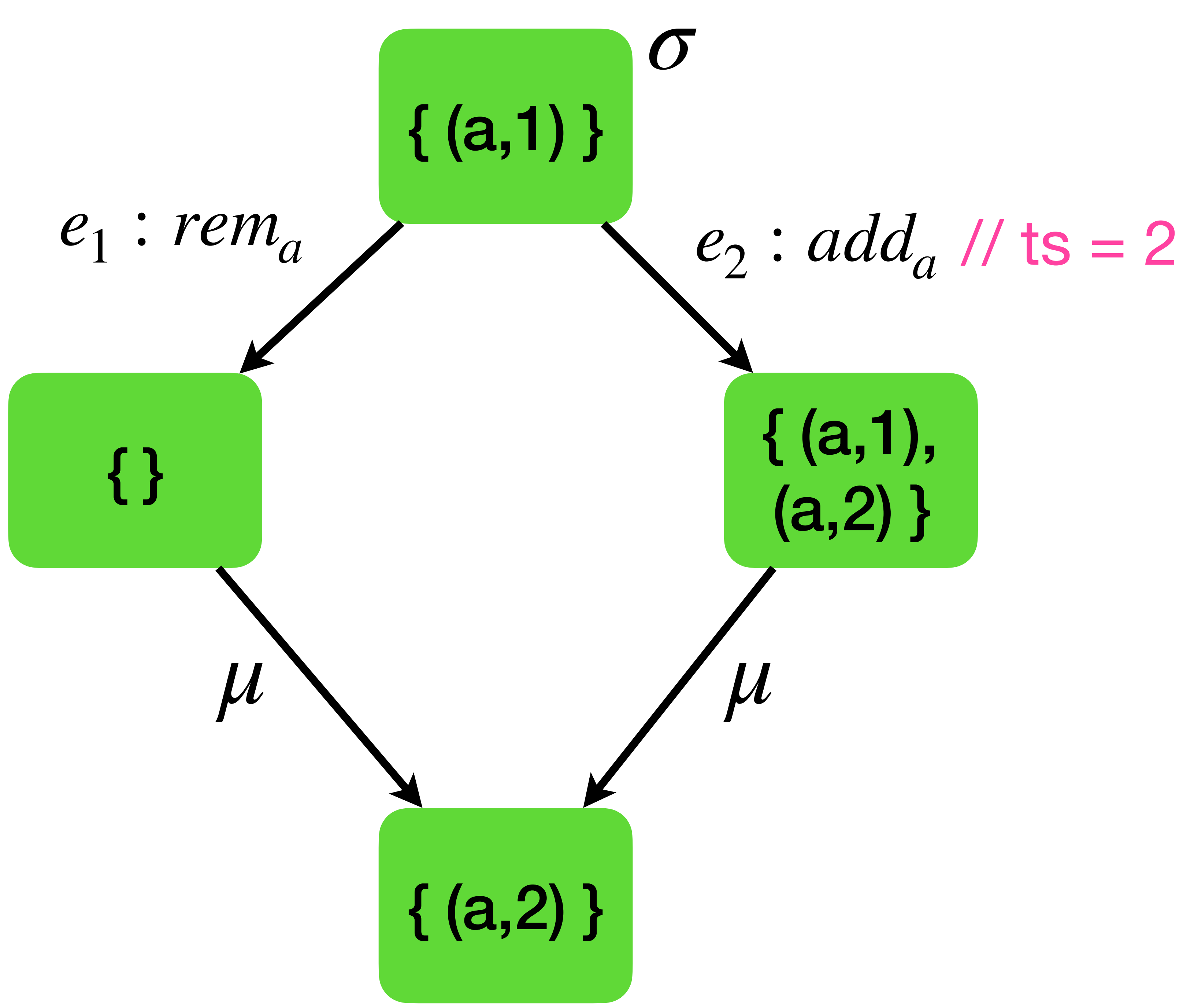
$$\mu(a, a, a) = a$$

[MergeCommutativity]

$$\mu(l, a, b) = \mu(l, b, a)$$

Bottom up linearisation

$$rc = \{(rem_a, add_a) \mid a \in \mathbb{E}\}$$



To show

$$e_2(e_1(\sigma)) = e_2(e_1(\sigma))$$

[BottomUp-2-OP]

$$\frac{e_1 \neq e_2 \qquad e_1 \xrightarrow{\mathsf{rc}} e_2 \vee e_1 \rightleftarrows e_2}{\mu(l, e_1(a), e_2(b)) = e_2(\mu(l, e_1(a), b))}$$

[BottomUp-1-OP]

$$\frac{(e_\top \neq \epsilon \wedge e_1 \neq e_\top) \vee (e_\top = \epsilon \wedge l = b)}{\mu(e_\top(l), e_1(a), e_\top(b)) = e_1(\mu(e_\top(l), a, e_\top(b)))}$$

[BottomUp-0-OP]

$$\mu(e_\top(l), e_\top(a), e_\top(b)) = e_\top(\mu(l, a, b))$$

[MergeIdempotence]

$$\mu(a, a, a) = a$$

[MergeCommutativity]

$$\mu(l, a, b) = \mu(l, b, a)$$

VCs are inductive versions over arbitrary length traces

Verified MRDTs

MRDT	rc **Policy**	**#LOC**	**Verification Time (s)**
Increment-only counter [12]	none	6	0.72
PN counter [23]	none	10	1.64
Enable-wins flag*	disable $\xrightarrow{\text{rc}}$ enable	30	29.80
Disable-wins flag*	enable $\xrightarrow{\text{rc}}$ disable	30	37.91
Grows-only set [12]	none	6	0.45
Grows-only map [23]	none	11	4.65
OR-set [23]	$\text{rem}_a \xrightarrow{\text{rc}} \text{add}_a$	20	4.53
OR-set (efficient)*	$\text{rem}_a \xrightarrow{\text{rc}} \text{add}_a$	34	660.00
Remove-wins set*	$\text{add}_a \xrightarrow{\text{rc}} \text{rem}_a$	22	9.60
Set-wins map*	$\text{del}_k \xrightarrow{\text{rc}} \text{set}_k$	20	5.06
Replicated Growable Array [1]	none	13	1.51
Optional register*	unset $\xrightarrow{\text{rc}}$ set	35	200.00
Multi-valued Register*	none	7	0.65
JSON-style MRDT*	Fig. 13	26	148.84

Neem also supports verification of RA-linearizability of state-based CRDTs

https://github.com/prismlab/neem

Challenges with Neem (F*)

- Automated verification in F* returns yes / no / ¯_(ツ)_/¯
  - Not pleasant for engineering
  - No counterexamples!

Challenges with Neem (F*)

- Automated verification in F* returns yes / no / ¯_(ツ)_/¯
  - Not pleasant for engineering
  - No counterexamples!
- TCB includes the SMT solver

Challenges with Neem (F*)

- Automated verification in F* returns yes / no / ¯_(ツ)_/¯
  - Not pleasant for engineering
  - No counterexamples!
- TCB includes the SMT solver
- F* SMT-aided proofs are brittle
  - Some lemmas need specific heuristics for fuel, unfolding, etc.
  - z3 upgrade breaks proofs!

Sal: Multi-modal verification of RDTs

- Move Neem from F* to Lean

Sal: Multi-modal verification of RDTs

- Move Neem from F* to Lean
- Advantages
  - Smaller TCB
  - Multi-modal verification
    - Property-based testing — counter-example generation and visualisation
    - SMT proofs, deductive verification

Sal: Multi-modal verification of RDTs

- Move Neem from F* to Lean
- Advantages
  - Smaller TCB
  - Multi-modal verification
    - Property-based testing — counter-example generation and visualisation
    - SMT proofs, deductive verification
- ***Agentic proof-oriented programming***
  - Fundamentally changing how we construct software
  - Coding Agents are great at Lean

Agentic Proof Oriented Programming in Sal

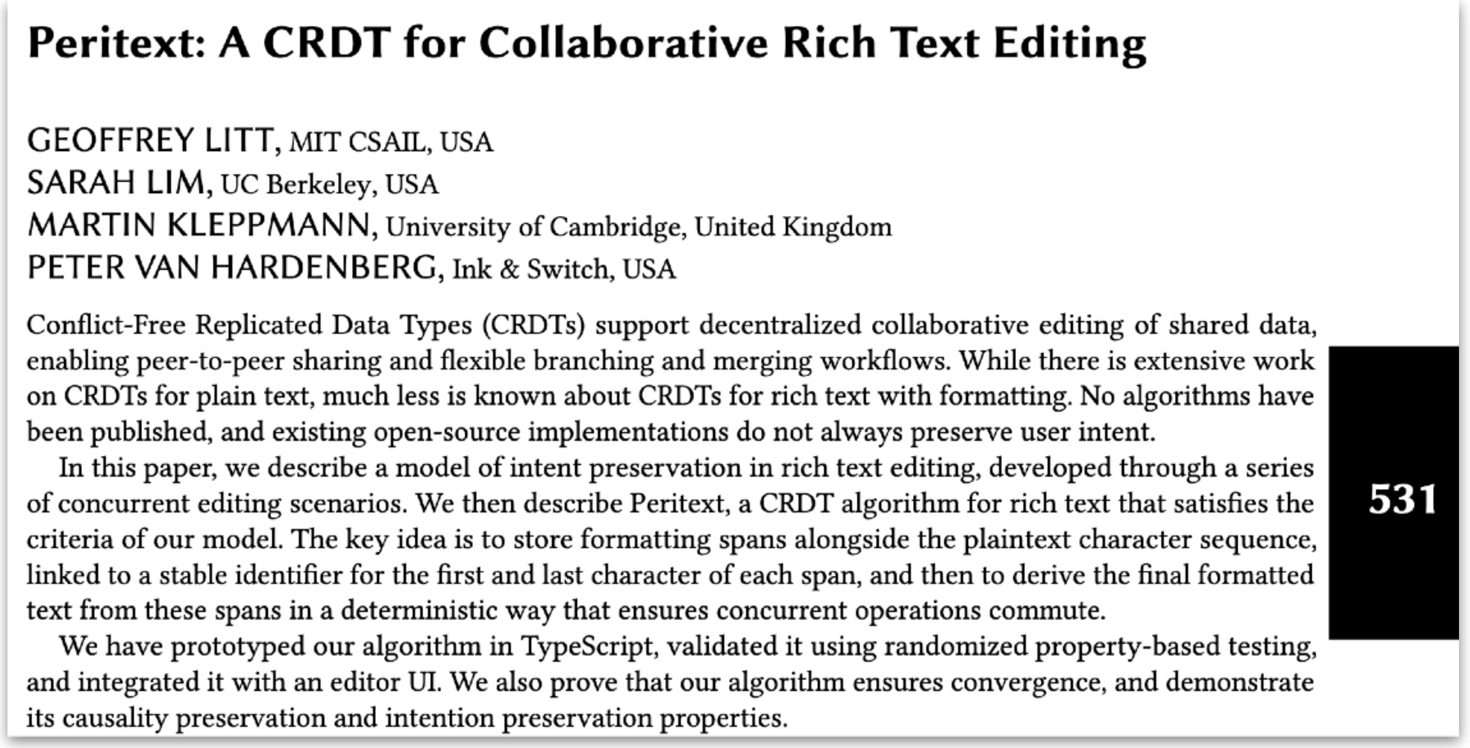
- Claude (Opus 4.7, Opus 4.8 and Fable) and Aristotle

Agentic Proof Oriented Programming in Sal

- Claude (Opus 4.7, Opus 4.8 and Fable) and Aristotle
- **13 new CRDTs and 4 new MRDTs** have been verified

Agentic Proof Oriented Programming in Sal

- Claude (Opus 4.7, Opus 4.8 and Fable) and Aristotle
- **13 new CRDTs and 4 new MRDTs** have been verified
- Including Peritext, Priority Queue, Bounded Counters



Peritext: A CRDT for Collaborative Rich Text Editing

GEOFFREY LITT, MIT CSAIL, USA
SARAH LIM, UC Berkeley, USA
MARTIN KLEPPMANN, University of Cambridge, United Kingdom
PETER VAN HARDENBERG, Ink & Switch, USA

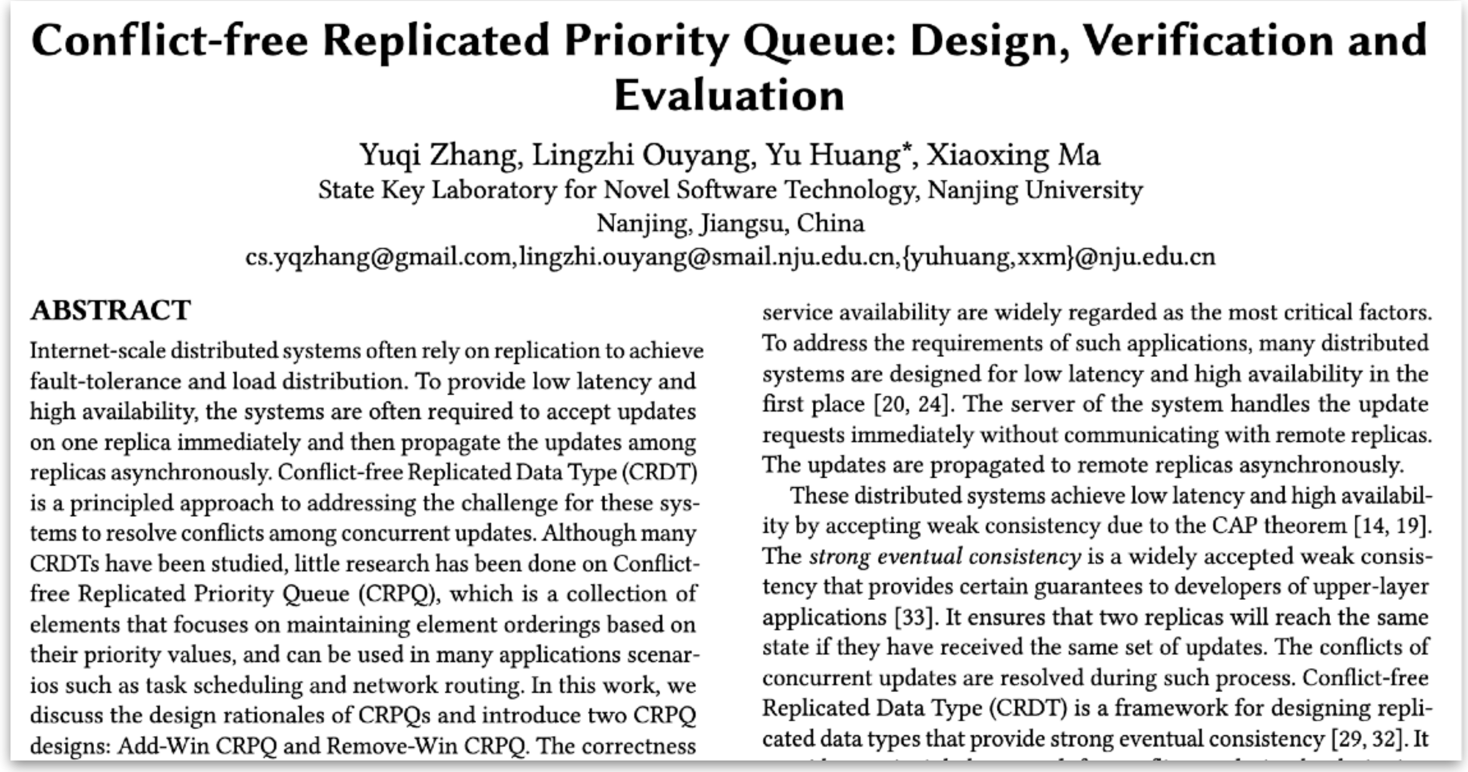
Conflict-Free Replicated Data Types (CRDTs) support decentralized collaborative editing of shared data, enabling peer-to-peer sharing and flexible branching and merging workflows. While there is extensive work on CRDTs for plain text, much less is known about CRDTs for rich text with formatting. No algorithms have been published, and existing open-source implementations do not always preserve user intent.

In this paper, we describe a model of intent preservation in rich text editing, developed through a series of concurrent editing scenarios. We then describe Peritext, a CRDT algorithm for rich text that satisfies the criteria of our model. The key idea is to store formatting spans alongside the plaintext character sequence, linked to a stable identifier for the first and last character of each span, and then to derive the final formatted text from these spans in a deterministic way that ensures concurrent operations commute.

We have prototyped our algorithm in TypeScript, validated it using randomized property-based testing, and integrated it with an editor UI. We also prove that our algorithm ensures convergence, and demonstrate its causality preservation and intention preservation properties.

531

Pen-and-paper proof + PBT



Conflict-free Replicated Priority Queue: Design, Verification and Evaluation

Yuqi Zhang, Lingzhi Ouyang, Yu Huang*, Xiaoxing Ma
State Key Laboratory for Novel Software Technology, Nanjing University
Nanjing, Jiangsu, China
cs.yqzhang@gmail.com,lingzhi.ouyang@smail.nju.edu.cn,{yuhuang,xxm}@nju.edu.cn

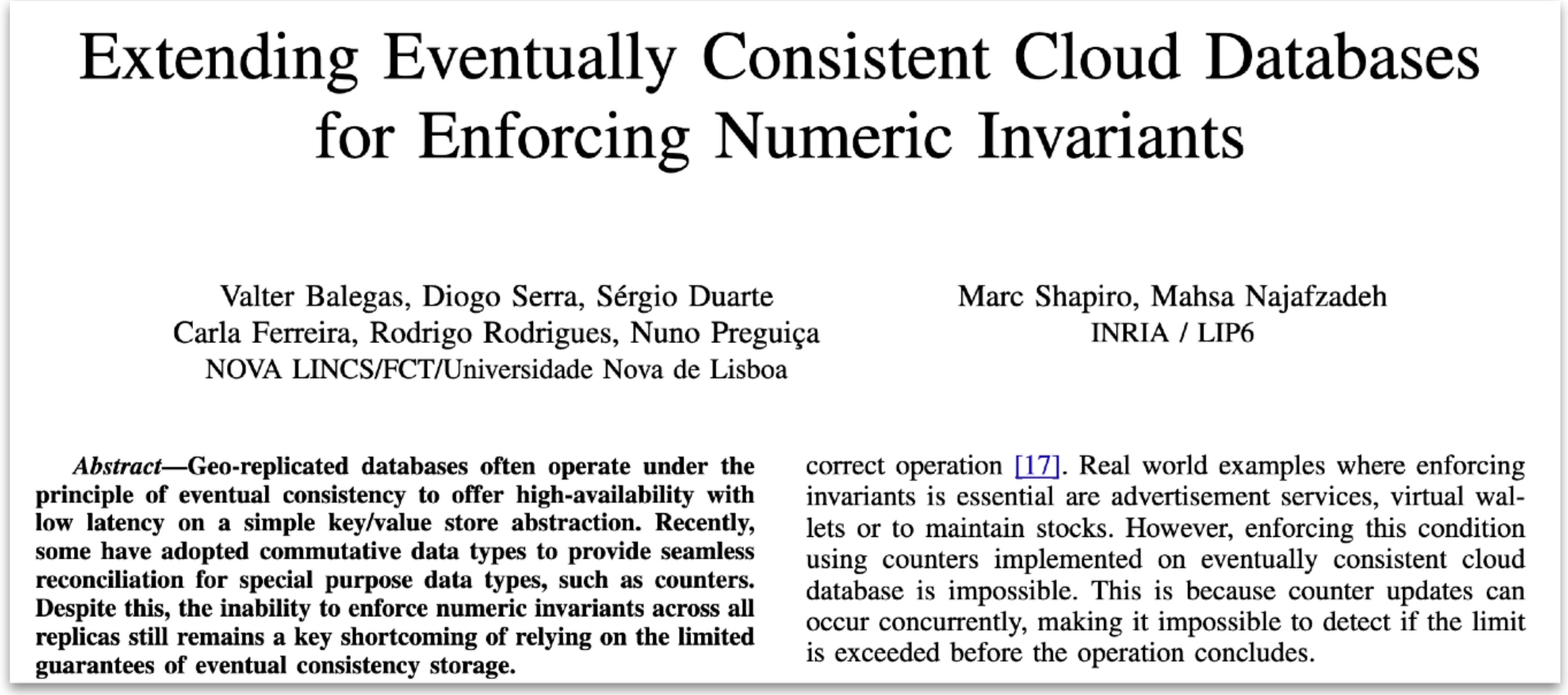
ABSTRACT

Internet-scale distributed systems often rely on replication to achieve fault-tolerance and load distribution. To provide low latency and high availability, the systems are often required to accept updates on one replica immediately and then propagate the updates among replicas asynchronously. Conflict-free Replicated Data Type (CRDT) is a principled approach to addressing the challenge for these systems to resolve conflicts among concurrent updates. Although many CRDTs have been studied, little research has been done on Conflict-free Replicated Priority Queue (CRPQ), which is a collection of elements that focuses on maintaining element orderings based on their priority values, and can be used in many applications scenarios such as task scheduling and network routing. In this work, we discuss the design rationales of CRPQs and introduce two CRPQ designs: Add-Win CRPQ and Remove-Win CRPQ. The correctness

service availability are widely regarded as the most critical factors. To address the requirements of such applications, many distributed systems are designed for low latency and high availability in the first place [20, 24]. The server of the system handles the update requests immediately without communicating with remote replicas. The updates are propagated to remote replicas asynchronously.

These distributed systems achieve low latency and high availability by accepting weak consistency due to the CAP theorem [14, 19]. The *strong eventual consistency* is a widely accepted weak consistency that provides certain guarantees to developers of upper-layer applications [33]. It ensures that two replicas will reach the same state if they have received the same set of updates. The conflicts of concurrent updates are resolved during such process. Conflict-free Replicated Data Type (CRDT) is a framework for designing replicated data types that provide strong eventual consistency [29, 32]. It

Verified in TLA+



Extending Eventually Consistent Cloud Databases for Enforcing Numeric Invariants

Valter Balegas, Diogo Serra, Sérgio Duarte
Carla Ferreira, Rodrigo Rodrigues, Nuno Preguiça
NOVA LINCS/FCT/Universidade Nova de Lisboa

Marc Shapiro, Mahsa Najafzadeh
INRIA / LIP6

***Abstract*—Geo-replicated databases often operate under the principle of eventual consistency to offer high-availability with low latency on a simple key/value store abstraction. Recently, some have adopted commutative data types to provide seamless reconciliation for special purpose data types, such as counters. Despite this, the inability to enforce numeric invariants across all replicas still remains a key shortcoming of relying on the limited guarantees of eventual consistency storage.**

correct operation [17]. Real world examples where enforcing invariants is essential are advertisement services, virtual wallets or to maintain stocks. However, enforcing this condition using counters implemented on eventually consistent cloud database is impossible. This is because counter updates can occur concurrently, making it impossible to detect if the limit is exceeded before the operation concludes.

Verified in TLA+

Agentic Proof Oriented Programming in Sal

- Claude (Opus 4.7, Opus 4.8 and Fable) and Aristotle
- **13 new CRDTs and 4 new MRDTs** have been verified
- Including Peritext, Priority Queue, Bounded Counters

Peritext: A CRDT for Collaborative Rich Text Editing

GEOFFREY LITT, MIT CSAIL, USA
SARAH LIM, UC Berkeley, USA
MARTIN KLEPPMANN, University of Cambridge, United Kingdom
PETER VAN HARDENBERG, Ink & Switch, USA

Conflict-Free Replicated Data Types (CRDTs) support decentralized collaborative editing of shared data, enabling peer-to-peer sharing and flexible branching and merging workflows. While there is extensive work on CRDTs for plain text, much less is known about CRDTs for rich text with formatting. No algorithms have been published, and existing open-source implementations do not always preserve user intent.

In this paper, we describe a model of intent preservation in rich text editing, developed through a series of concurrent editing scenarios. We then describe Peritext, a CRDT algorithm for rich text that satisfies the criteria of our model. The key idea is to store formatting spans alongside the plaintext character sequence, linked to a stable identifier for the first and last character of each span, and then to derive the final formatted text from these spans in a deterministic way that ensures concurrent operations commute.

We have prototyped our algorithm in TypeScript, validated it using randomized property-based testing, and integrated it with an editor UI. We also prove that our algorithm ensures convergence, and demonstrate its causality preservation and intention preservation properties.

531

Pen-and-paper proof + PBT

Conflict-free Replicated Priority Queue: Design, Verification and Evaluation

Yuqi Zhang, Lingzhi Ouyang, Yu Huang*, Xiaoxing Ma
State Key Laboratory for Novel Software Technology, Nanjing University
Nanjing, Jiangsu, China
cs.yqzhang@gmail.com,lingzhi.ouyang@smail.nju.edu.cn,{yuhuang,xxm}@nju.edu.cn

ABSTRACT

Internet-scale distributed systems often rely on replication to achieve fault-tolerance and load distribution. To provide low latency and high availability, the systems are often required to accept updates on one replica immediately and then propagate the updates among replicas asynchronously. Conflict-free Replicated Data Type (CRDT) is a principled approach to addressing the challenge for these systems to resolve conflicts among concurrent updates. Although many CRDTs have been studied, little research has been done on Conflict-free Replicated Priority Queue (CRPQ), which is a collection of elements that focuses on maintaining element orderings based on their priority values, and can be used in many applications scenarios such as task scheduling and network routing. In this work, we discuss the design rationales of CRPQs and introduce two CRPQ designs: Add-Win CRPQ and Remove-Win CRPQ. The correctness

service availability are widely regarded as the most critical factors. To address the requirements of such applications, many distributed systems are designed for low latency and high availability in the first place [20, 24]. The server of the system handles the update requests immediately without communicating with remote replicas. The updates are propagated to remote replicas asynchronously.

These distributed systems achieve low latency and high availability by accepting weak consistency due to the CAP theorem [14, 19]. The *strong eventual consistency* is a widely accepted weak consistency that provides certain guarantees to developers of upper-layer applications [33]. It ensures that two replicas will reach the same state if they have received the same set of updates. The conflicts of concurrent updates are resolved during such process. Conflict-free Replicated Data Type (CRDT) is a framework for designing replicated data types that provide strong eventual consistency [29, 32]. It

Verified in TLA+

Extending Eventually Consistent Cloud Databases for Enforcing Numeric Invariants

Valter Balegas, Diogo Serra, Sérgio Duarte
Carla Ferreira, Rodrigo Rodrigues, Nuno Preguiça
NOVA LINCS/FCT/Universidade Nova de Lisboa

Marc Shapiro, Mahsa Najafzadeh
INRIA / LIP6

***Abstract*—Geo-replicated databases often operate under the principle of eventual consistency to offer high-availability with low latency on a simple key/value store abstraction. Recently, some have adopted commutative data types to provide seamless reconciliation for special purpose data types, such as counters. Despite this, the inability to enforce numeric invariants across all replicas still remains a key shortcoming of relying on the limited guarantees of eventual consistency storage.**

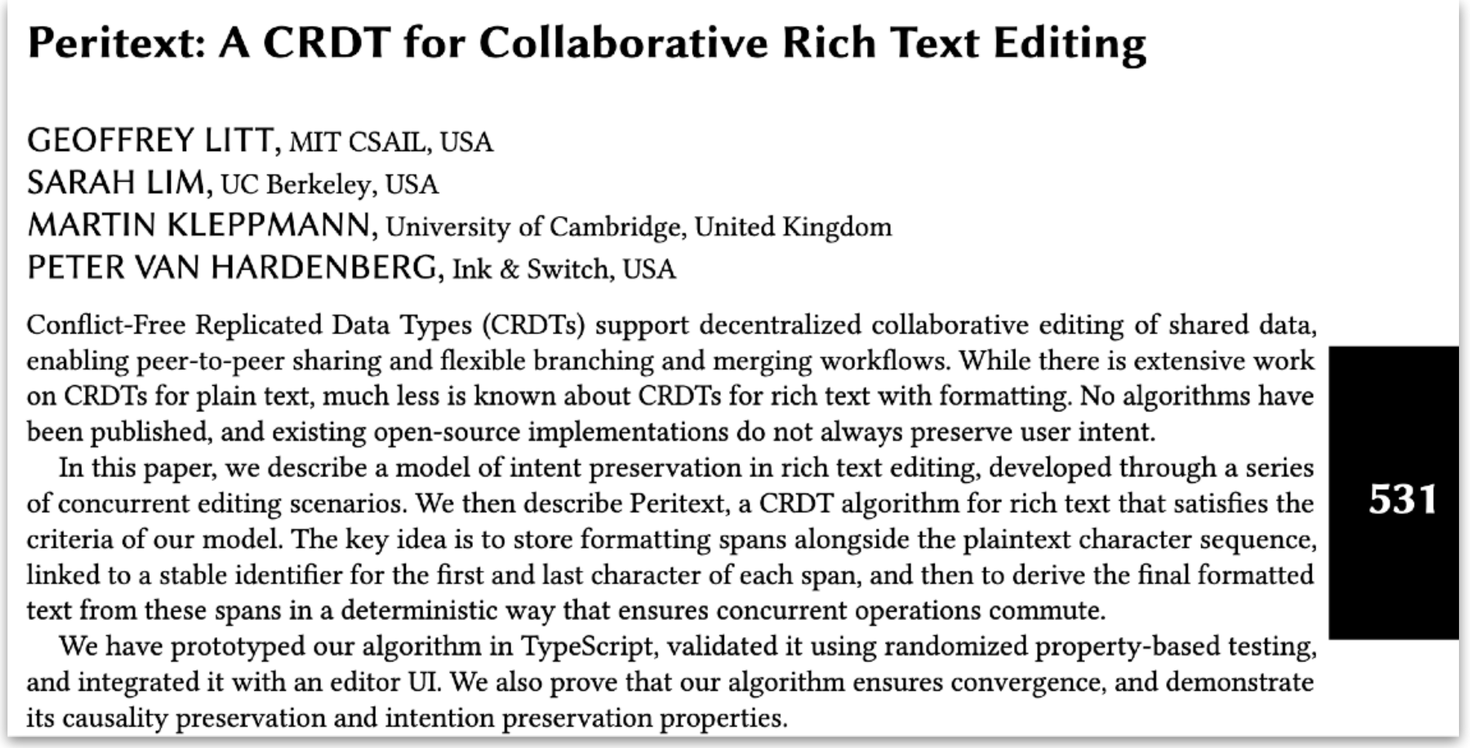
correct operation [17]. Real world examples where enforcing invariants is essential are advertisement services, virtual wallets or to maintain stocks. However, enforcing this condition using counters implemented on eventually consistent cloud database is impossible. This is because counter updates can occur concurrently, making it impossible to detect if the limit is exceeded before the operation concludes.

Verified in TLA+

- ***A novel tombstone-free collaborative text editing MRDT***

Agentic Proof Oriented Programming in Sal

- Claude (Opus 4.7, Opus 4.8 and Fable) and Aristotle
- **13 new CRDTs and 4 new MRDTs** have been verified
- Including Peritext, Priority Queue, Bounded Counters



Peritext: A CRDT for Collaborative Rich Text Editing

GEOFFREY LITT, MIT CSAIL, USA
SARAH LIM, UC Berkeley, USA
MARTIN KLEPPMANN, University of Cambridge, United Kingdom
PETER VAN HARDENBERG, Ink & Switch, USA

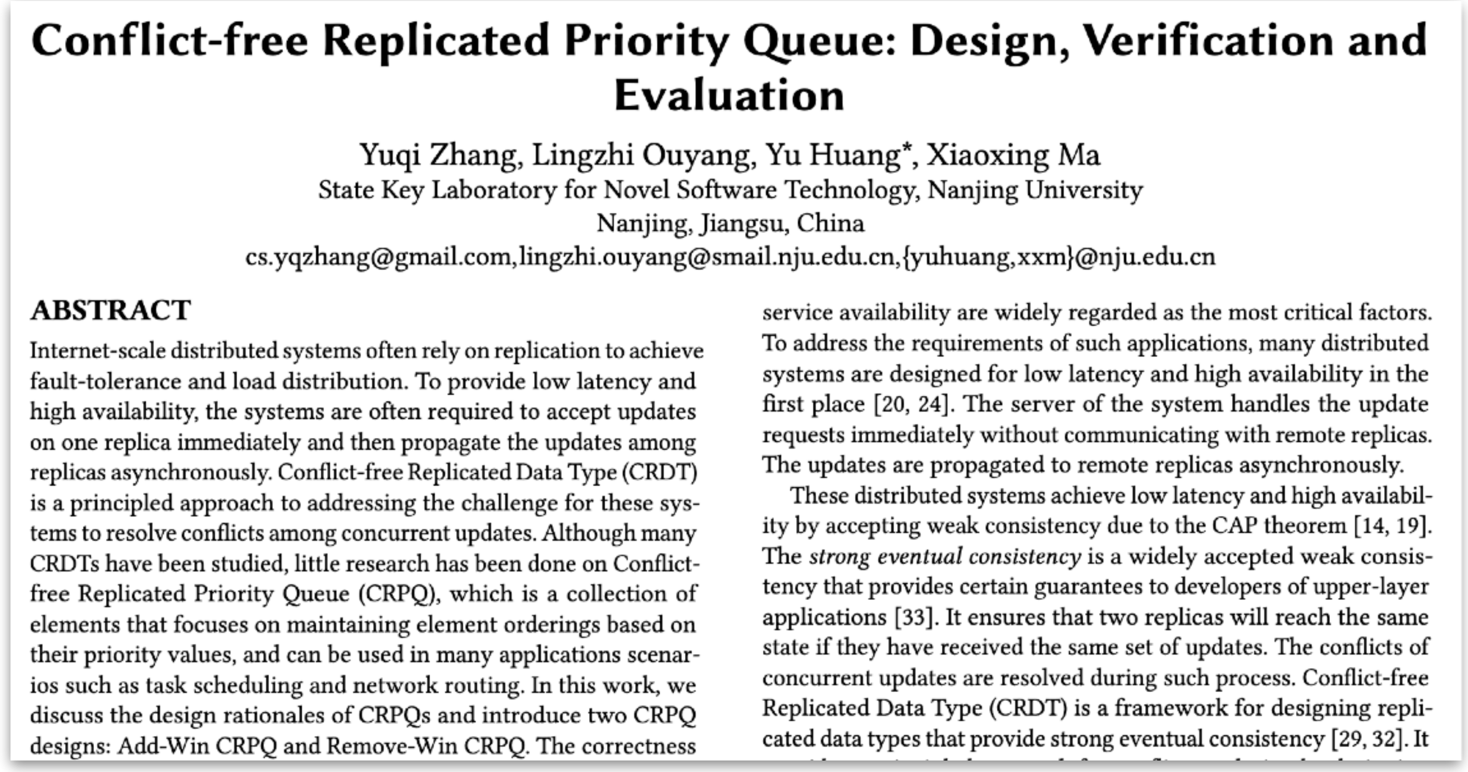
Conflict-Free Replicated Data Types (CRDTs) support decentralized collaborative editing of shared data, enabling peer-to-peer sharing and flexible branching and merging workflows. While there is extensive work on CRDTs for plain text, much less is known about CRDTs for rich text with formatting. No algorithms have been published, and existing open-source implementations do not always preserve user intent.

In this paper, we describe a model of intent preservation in rich text editing, developed through a series of concurrent editing scenarios. We then describe Peritext, a CRDT algorithm for rich text that satisfies the criteria of our model. The key idea is to store formatting spans alongside the plaintext character sequence, linked to a stable identifier for the first and last character of each span, and then to derive the final formatted text from these spans in a deterministic way that ensures concurrent operations commute.

We have prototyped our algorithm in TypeScript, validated it using randomized property-based testing, and integrated it with an editor UI. We also prove that our algorithm ensures convergence, and demonstrate its causality preservation and intention preservation properties.

531

Pen-and-paper proof + PBT



Conflict-free Replicated Priority Queue: Design, Verification and Evaluation

Yuqi Zhang, Lingzhi Ouyang, Yu Huang*, Xiaoxing Ma
State Key Laboratory for Novel Software Technology, Nanjing University
Nanjing, Jiangsu, China
cs.yqzhang@gmail.com,lingzhi.ouyang@smail.nju.edu.cn,{yuhuang,xxm}@nju.edu.cn

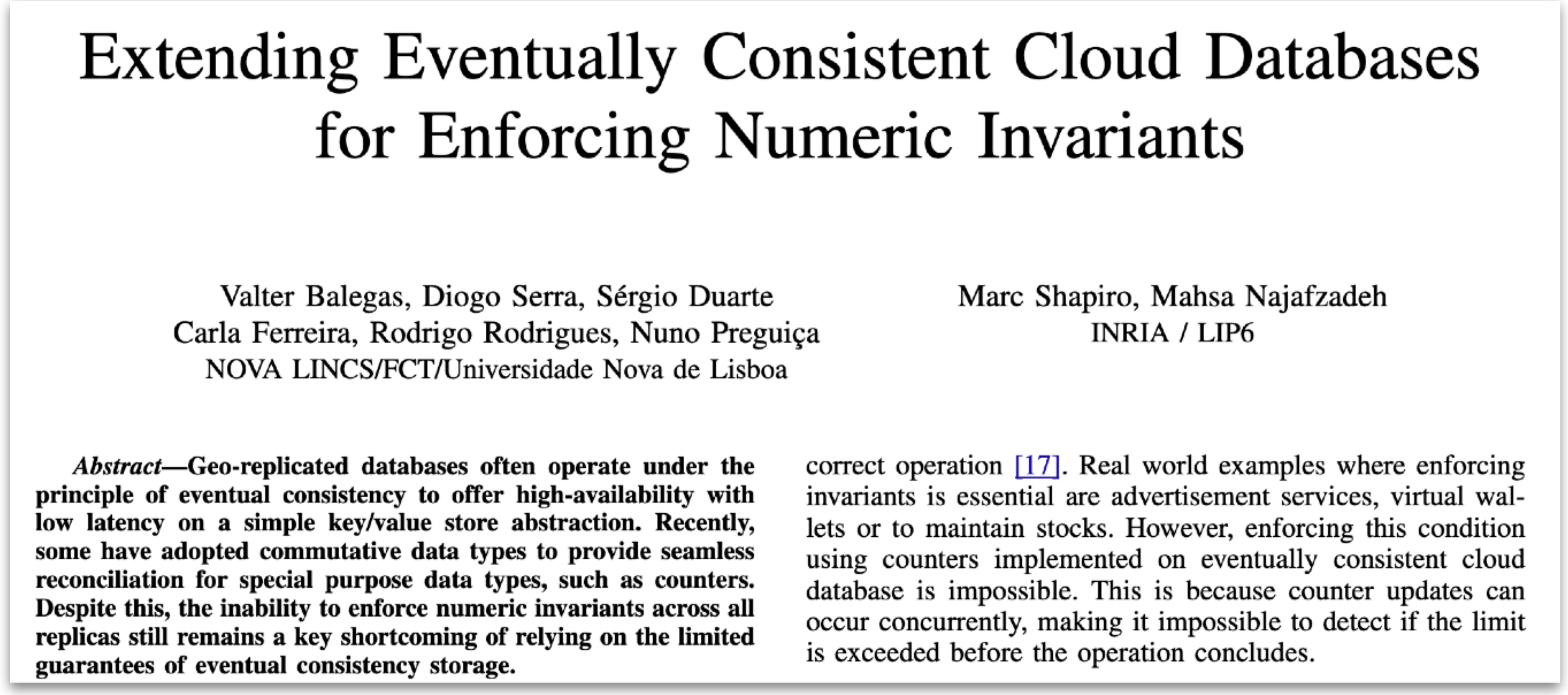
ABSTRACT

Internet-scale distributed systems often rely on replication to achieve fault-tolerance and load distribution. To provide low latency and high availability, the systems are often required to accept updates on one replica immediately and then propagate the updates among replicas asynchronously. Conflict-free Replicated Data Type (CRDT) is a principled approach to addressing the challenge for these systems to resolve conflicts among concurrent updates. Although many CRDTs have been studied, little research has been done on Conflict-free Replicated Priority Queue (CRPQ), which is a collection of elements that focuses on maintaining element orderings based on their priority values, and can be used in many applications scenarios such as task scheduling and network routing. In this work, we discuss the design rationales of CRPQs and introduce two CRPQ designs: Add-Win CRPQ and Remove-Win CRPQ. The correctness

service availability are widely regarded as the most critical factors. To address the requirements of such applications, many distributed systems are designed for low latency and high availability in the first place [20, 24]. The server of the system handles the update requests immediately without communicating with remote replicas. The updates are propagated to remote replicas asynchronously.

These distributed systems achieve low latency and high availability by accepting weak consistency due to the CAP theorem [14, 19]. The *strong eventual consistency* is a widely accepted weak consistency that provides certain guarantees to developers of upper-layer applications [33]. It ensures that two replicas will reach the same state if they have received the same set of updates. The conflicts of concurrent updates are resolved during such process. Conflict-free Replicated Data Type (CRDT) is a framework for designing replicated data types that provide strong eventual consistency [29, 32]. It

Verified in TLA+



Extending Eventually Consistent Cloud Databases for Enforcing Numeric Invariants

Valter Balegas, Diogo Serra, Sérgio Duarte
Carla Ferreira, Rodrigo Rodrigues, Nuno Preguiça
NOVA LINCS/FCT/Universidade Nova de Lisboa

Marc Shapiro, Mahsa Najafzadeh
INRIA / LIP6

***Abstract*—Geo-replicated databases often operate under the principle of eventual consistency to offer high-availability with low latency on a simple key/value store abstraction. Recently, some have adopted commutative data types to provide seamless reconciliation for special purpose data types, such as counters. Despite this, the inability to enforce numeric invariants across all replicas still remains a key shortcoming of relying on the limited guarantees of eventual consistency storage.**

correct operation [17]. Real world examples where enforcing invariants is essential are advertisement services, virtual wallets or to maintain stocks. However, enforcing this condition using counters implemented on eventually consistent cloud database is impossible. This is because counter updates can occur concurrently, making it impossible to detect if the limit is exceeded before the operation concludes.

Verified in TLA+

- ***A novel tombstone-free collaborative text editing MRDT***
- Meta-theory has been mechanised

Who checks the Specs?

- Agentic proof-oriented programming allows ***vibe proving***
  - ***How do you know that you have verified against the right specs?***

Who checks the Specs?

- Agentic proof-oriented programming allows ***vibe proving***
    - ***How do you know that you have verified against the right specs?***
- Small Proof-oriented Tests (SPOT)
    - Unit tests, lifted to proofs
    - RiSE MSR blog post: https://risemsr.github.io/blog/2026-04-16-spotting-specs/

Who checks the Specs?

- Agentic proof-oriented programming allows ***vibe proving***
  - ***How do you know that you have verified against the right specs?***
- Small Proof-oriented Tests (SPOT)
  - Unit tests, lifted to proofs
  - RiSE MSR blog post: https://risemsr.github.io/blog/2026-04-16-spotting-specs/
- For a correct sorting spec,
  - `sort [3,1,2] = [1,2,3]` ***holds***
  - `sort [3,1,2] = []` ***does not hold***
    - Holds for sort spec that only checks for sortedness and not permutation.

Who checks the Specs?

- Agentic proof-oriented programming allows ***vibe proving***
  - ***How do you know that you have verified against the right specs?***
- Small Proof-oriented Tests (SPOT)
  - Unit tests, lifted to proofs
  - RiSE MSR blog post: https://risemsr.github.io/blog/2026-04-16-spotting-specs/
- For a correct sorting spec,
  - `sort [3,1,2] = [1,2,3]` ***holds***
  - `sort [3,1,2] = []` ***does not hold***
    - Holds for sort spec that only checks for sortedness and not permutation.
- Use agents to write SPOTs and prove.

ReadSide Proofs — Peritext

Peritext: A CRDT for Collaborative Rich Text Editing

GEOFFREY LITT, MIT CSAIL, USA
SARAH LIM, UC Berkeley, USA
MARTIN KLEPPMANN, University of Cambridge, United Kingdom
PETER VAN HARDENBERG, Ink & Switch, USA

Conflict-Free Replicated Data Types (CRDTs) support decentralized collaborative editing of shared data, enabling peer-to-peer sharing and flexible branching and merging workflows. While there is extensive work on CRDTs for plain text, much less is known about CRDTs for rich text with formatting. No algorithms have been published, and existing open-source implementations do not always preserve user intent.

In this paper, we describe a model of intent preservation in rich text editing, developed through a series of concurrent editing scenarios. We then describe Peritext, a CRDT algorithm for rich text that satisfies the criteria of our model. The key idea is to store formatting spans alongside the plaintext character sequence, linked to a stable identifier for the first and last character of each span, and then to derive the final formatted text from these spans in a deterministic way that ensures concurrent operations commute.

We have prototyped our algorithm in TypeScript, validated it using randomized property-based testing, and integrated it with an editor UI. We also prove that our algorithm ensures convergence, and demonstrate its causality preservation and intention preservation properties.

PeriText — Intent preservation

Alice:

$\underbrace{\textbf{The fox}}_{\text{bold}}$ jumped.

Bob:

The $\underbrace{\textbf{fox jumped.}}_{\text{bold}}$

$\underbrace{\textbf{The fox jumped.}}_{\text{bold}}$

Concurrent overlapping marks

PeriText — Intent preservation

Alice:

$\underbrace{\textbf{The fox}}_{\text{bold}}$ jumped.

Bob:

The $\underbrace{\textbf{fox jumped.}}_{\text{bold}}$

$\underbrace{\textbf{The fox jumped.}}_{\text{bold}}$

Concurrent overlapping marks

The $\underbrace{\textbf{fox jumped}}_{\text{bold}}$.

The quick $\underbrace{\textbf{fox jumped over the dog}}_{\text{bold}}$.

Text insertion at span boundaries

ReadSide Proofs — Peritext

Paper claim	Sal artifact	Status / notes
§A.1 Convergence	24 RA-linearizability VCs in `Peritext_CRDT.lean` and `Peritext_MRDT.lean` (`merge_comm`, `merge_idem`, `base_2op`, `ind_lca_2op`, `inter_*`, `ind_*`, `lem_0op`)	✅ All closed via `by sal`, kernel-checked. Universally quantified over op interleavings.
§3.1 Causality preservation	Framework assumption (causal delivery); `afters` map structurally encodes insert causality; `causal_order_visible_lt` (RGA readside) lifts it to visible-order	✅ Same convention as paper §3.1.
§A.2 Intent preservation — convergence at the read	`readRichText_visible_convergent`	✅ Read-side analogue of merge convergence.
§A.2 Ex 1 — insertion within a span	`insert_within_span_in_span_visible` + propagation lemmas	✅
§A.2 Ex 2 — overlapping same-type Adds	`partial_overlap_all_adds_formatted_visible`	✅
§A.2 Ex 3 — different mark types coexist	`different_type_adds_coexist_visible`	✅
§A.2 Ex 4 — same-type, different values (colors)	not formalised	⚠️ Requires `markValue` field on `MarkOp` — state-shape change.
§A.2 Ex 5 — bold vs non-bold conflict	`add_wins_over_concurrent_remove_visible` + `no_add_cover_implies_unformatted_visible`	✅ LWW by opId (paper §4.4).
§A.2 Ex 6 — overlapping comments via distinct markType	follows from `different_type_adds_coexist_visible`	✅
§A.2 Ex 7 — bold-boundary insertion expands	`ex7_bold_older_sibling_in_span` + `bold_expand_in_span_visible` (via `bold_expand_reach`)	✅
§A.2 Ex 8 — link-boundary insertion does not expand	`ex8_link_descendant_visible_lt_endId` + `_not_in_span_visible_of_wf`	✅ Uses `wf_afters` acyclicity.
(Beyond paper) Anchors survive tombstones	`anchors_survive_tombstones_visible`	✅ Implicit in §4.4 paper discussion.

Take aways

Take aways

- Automated verification for RDTs is pragmatic and necessary

Take aways

- **Automated verification for RDTs is pragmatic and necessary**
- **Automated verification + Agentic proof engineering allows scaling to real-world RDTs**
  - Peritext was formalised in a couple of days
  - 50k+ lines of Lean proofs

Take aways

- **Automated verification for RDTs is pragmatic and necessary**
- **Automated verification + Agentic proof engineering allows scaling to real-world RDTs**
    - Peritext was formalised in a couple of days
    - 50k+ lines of Lean proofs
- **Automated verification VCs are guardrails for agentic proof-oriented programming**
    - Answers “How do we know the Spec is correct”?
    - John Regehr, “Zero-Degree-of-Freedom LLM Coding using Executable Oracles”, https://john.regehr.org/writing/zero_dof_programming.html

Take aways

- **Automated verification for RDTs is pragmatic and necessary**
- **Automated verification + Agentic proof engineering allows scaling to real-world RDTs**
    - Peritext was formalised in a couple of days
    - 50k+ lines of Lean proofs
- **Automated verification VCs are guardrails for agentic proof-oriented programming**
    - Answers “How do we know the Spec is correct”?
    - John Regehr, “Zero-Degree-of-Freedom LLM Coding using Executable Oracles”, https://john.regehr.org/writing/zero_dof_programming.html

https://github.com/fplaunchpad/sal