Automatically Verifying Replicated Data Types

KC Sivaramakrishnan

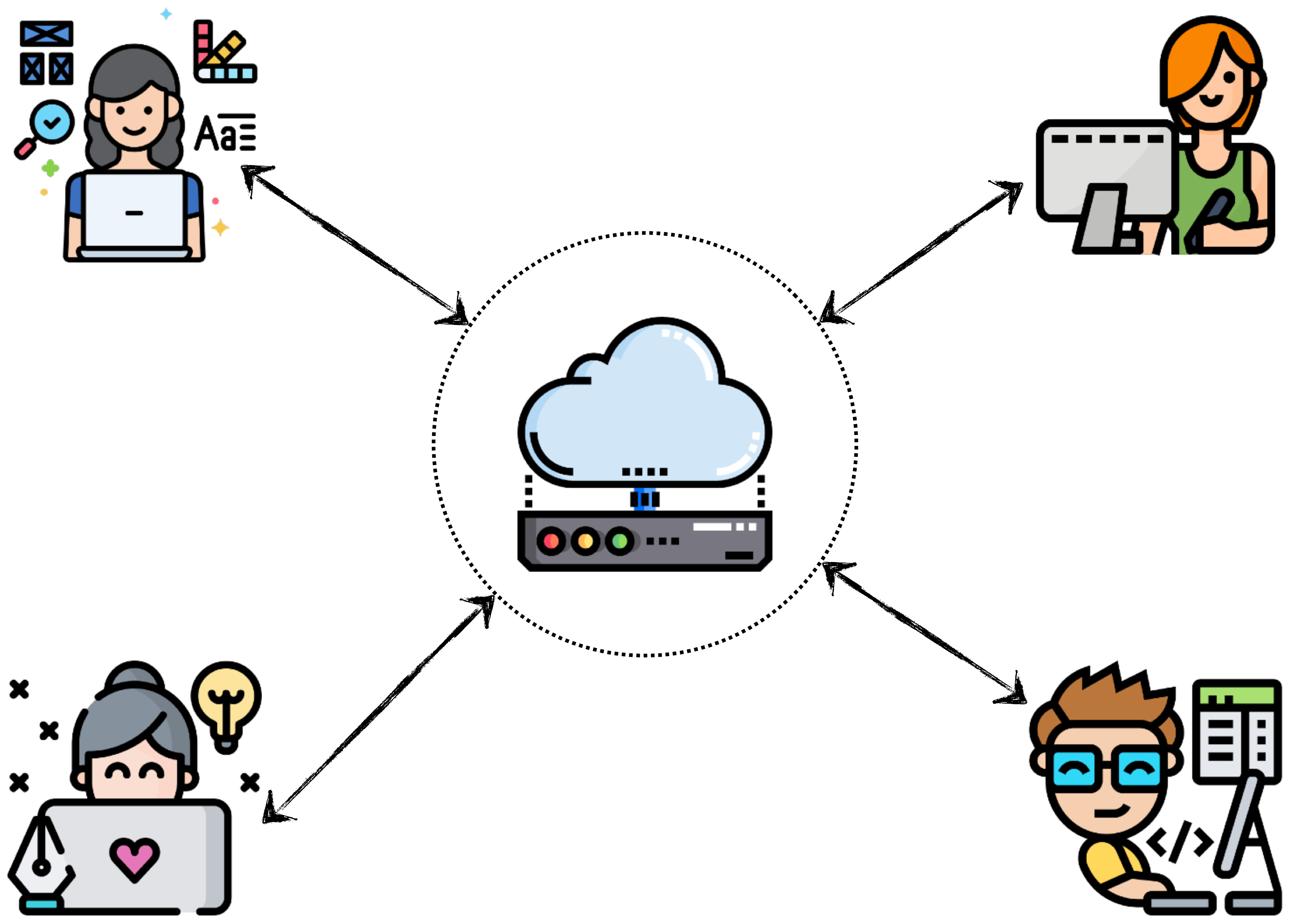
Joint work with Vimala Soundarapandian, Aseem Rastogi and Kartik Nagar

WG 2.1

Sep 8th to 12th, 2025

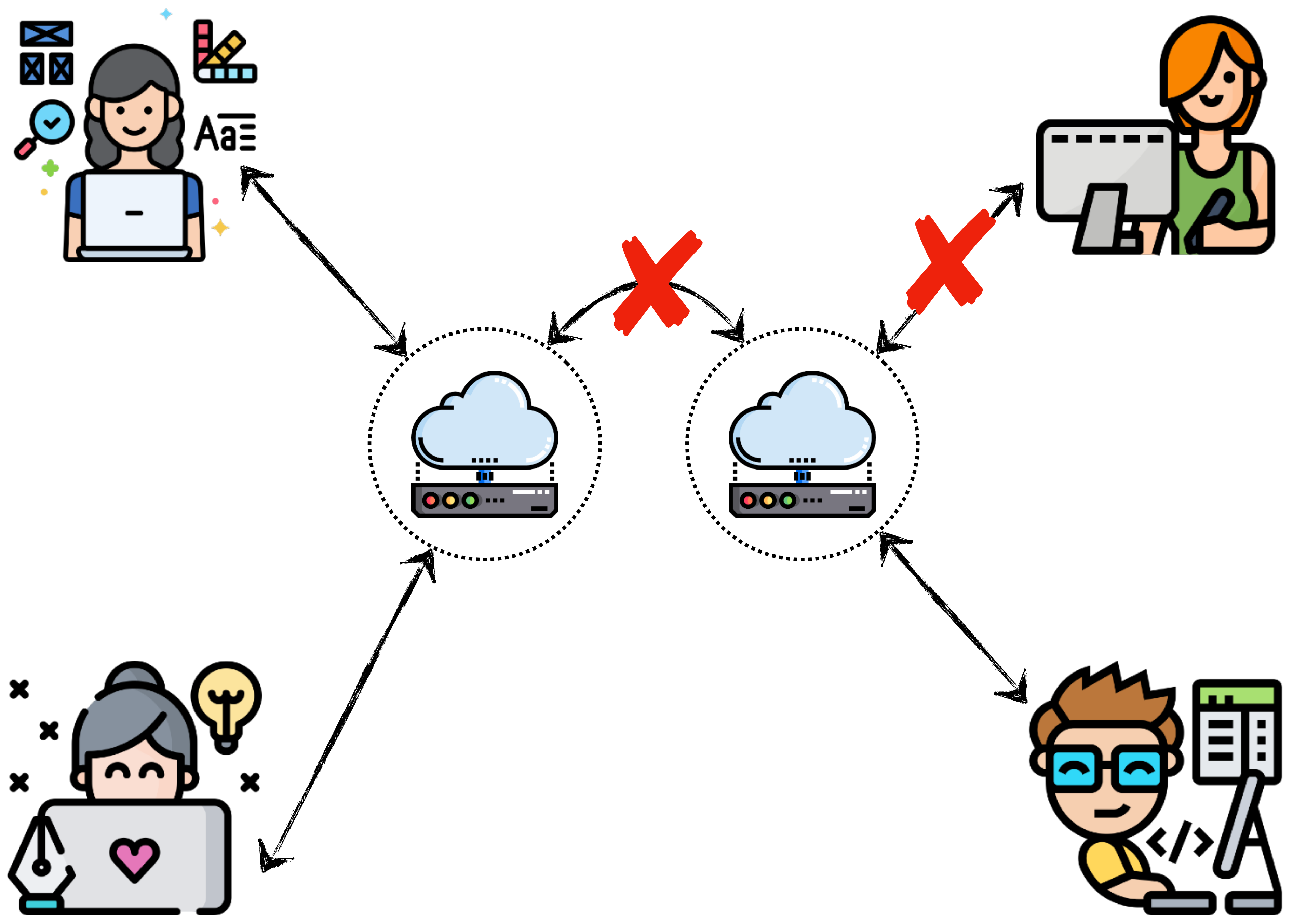
Collaborative Applications

Airtable Google Docs Figma Notion Overleaf



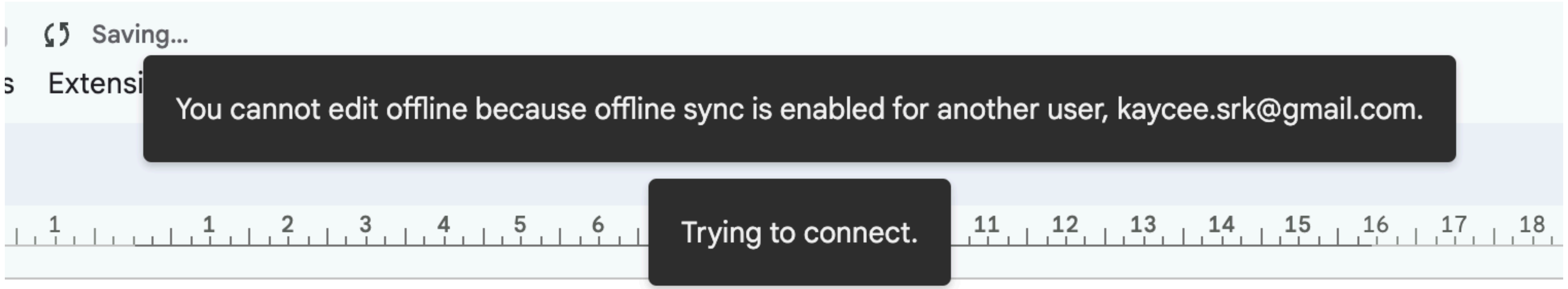
Collaborative Applications

Airtable Google Docs Figma Notion Overleaf



Network Partitions

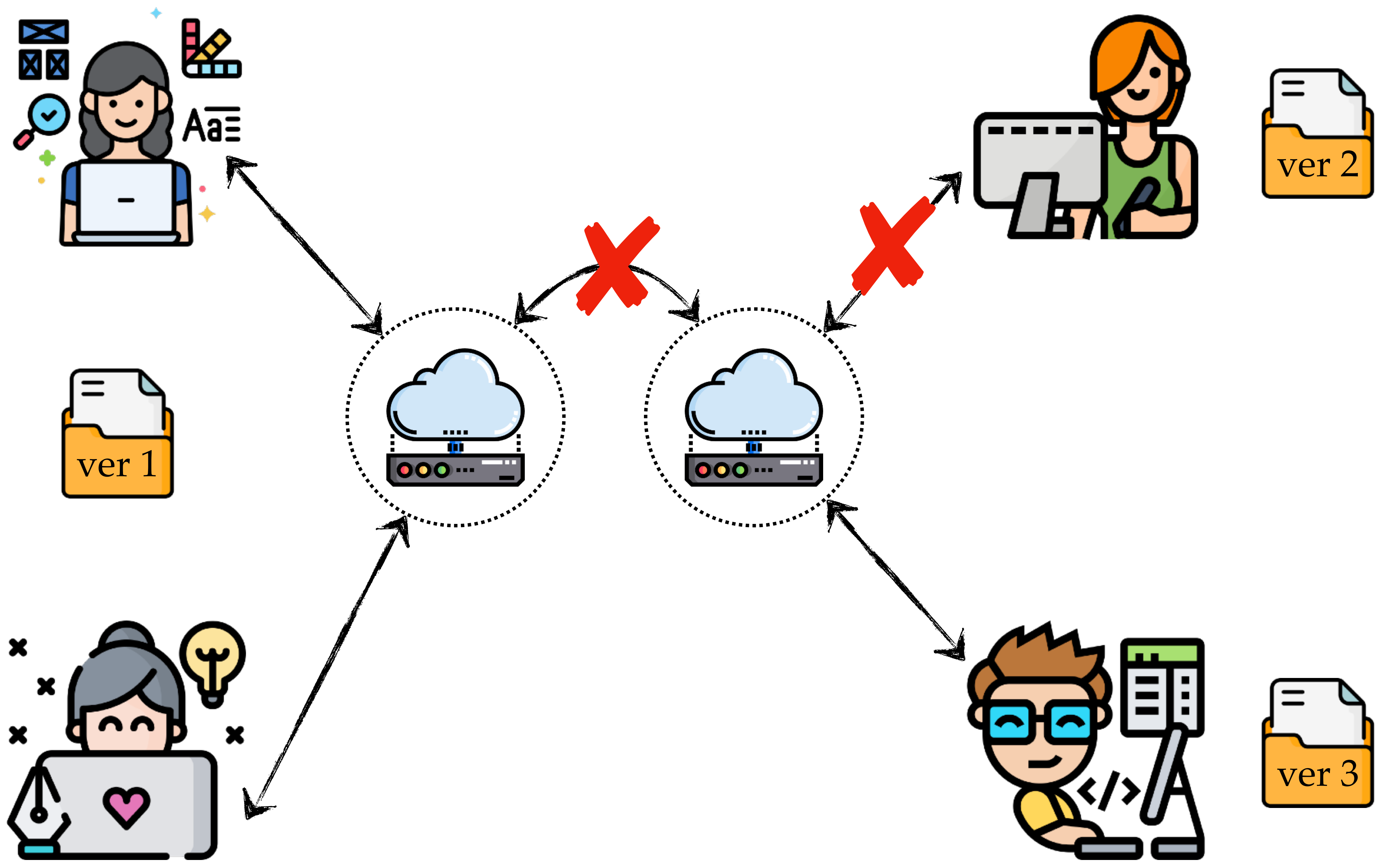
- Centralised Apps provide limited support for offline editing



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

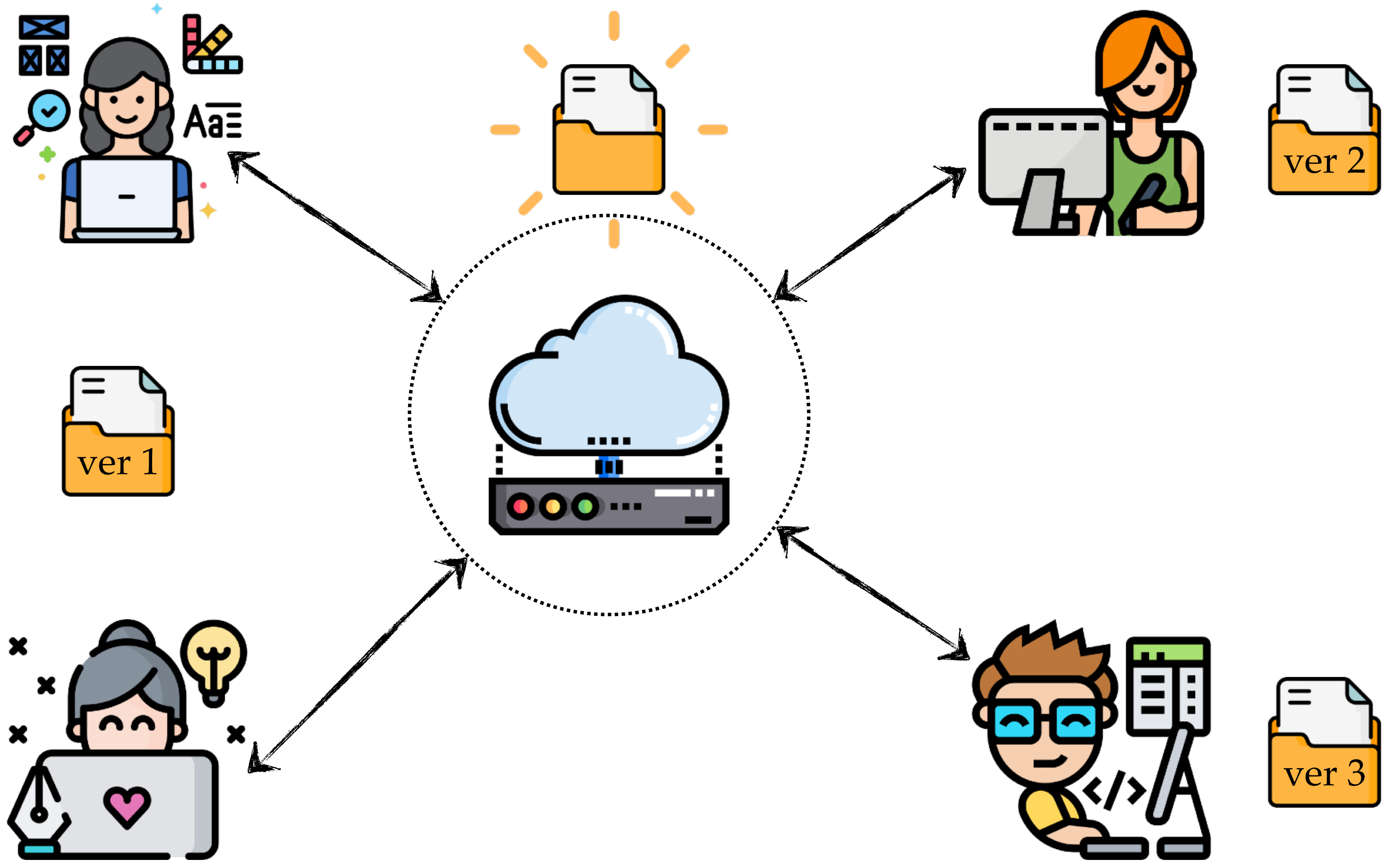
Local-first software

Airtable Google Docs Figma Notion Overleaf



Local-first software

Airtable Google Docs Figma Notion Overleaf



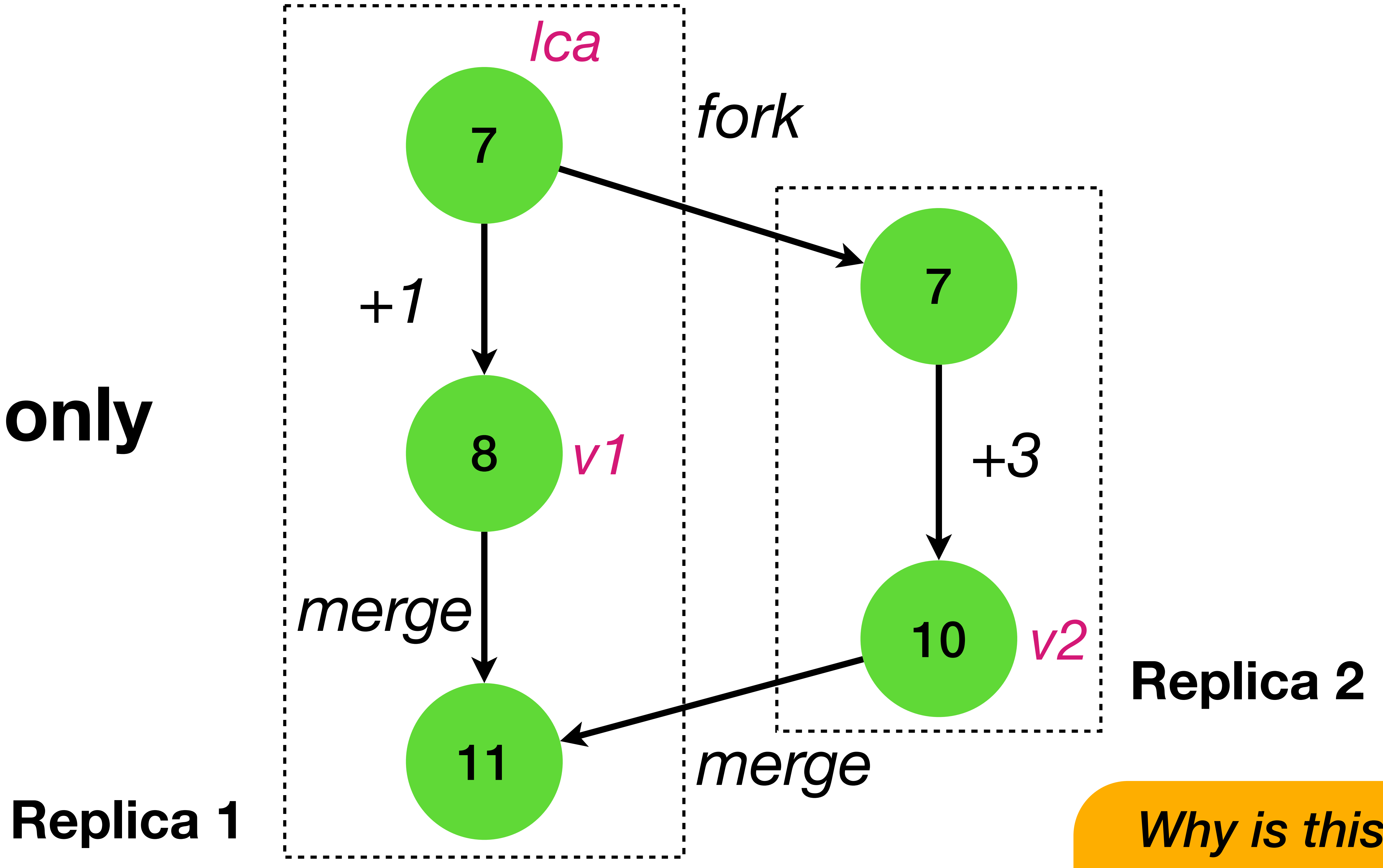
How do we build such applications?

Embed the notion of replication into the data types

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?

Verification using Algebraic Properties

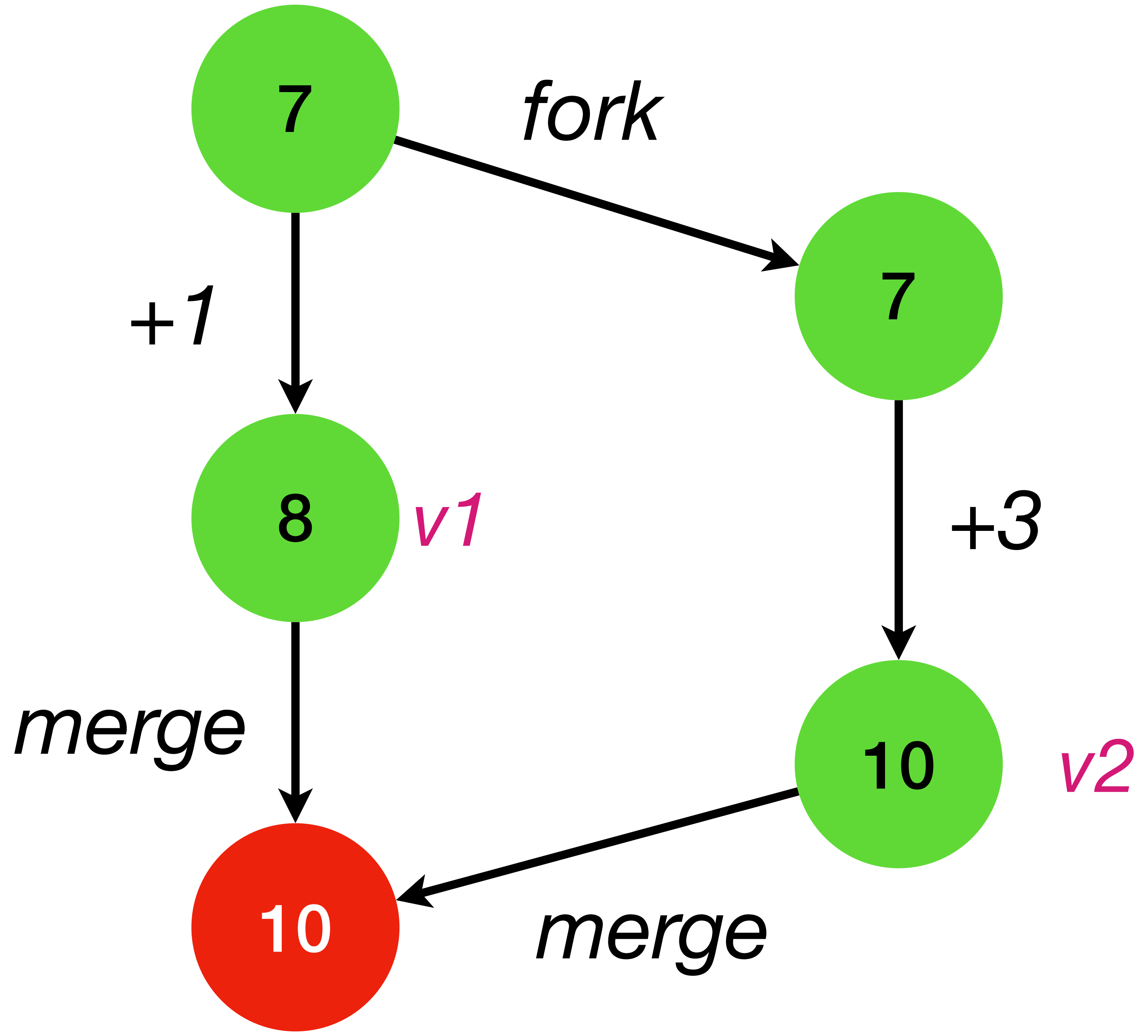
- State-based Convergent Replicated Data Types (CRDTs)
  - Merge is 2-way $-\mu(v_1, v_2)$
  - Verify algebraic properties of merge for *strong eventual consistency*

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



Prior work: Capturing Intent through Axiomatic Spec

RESEARCH-ARTICLE | OPEN ACCESS

Certified mergeable replicated data types

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

- **Execution graph** = events + visibility (partial order)
- **Operations** = folds over execution graphs

Specification

Simulation relation

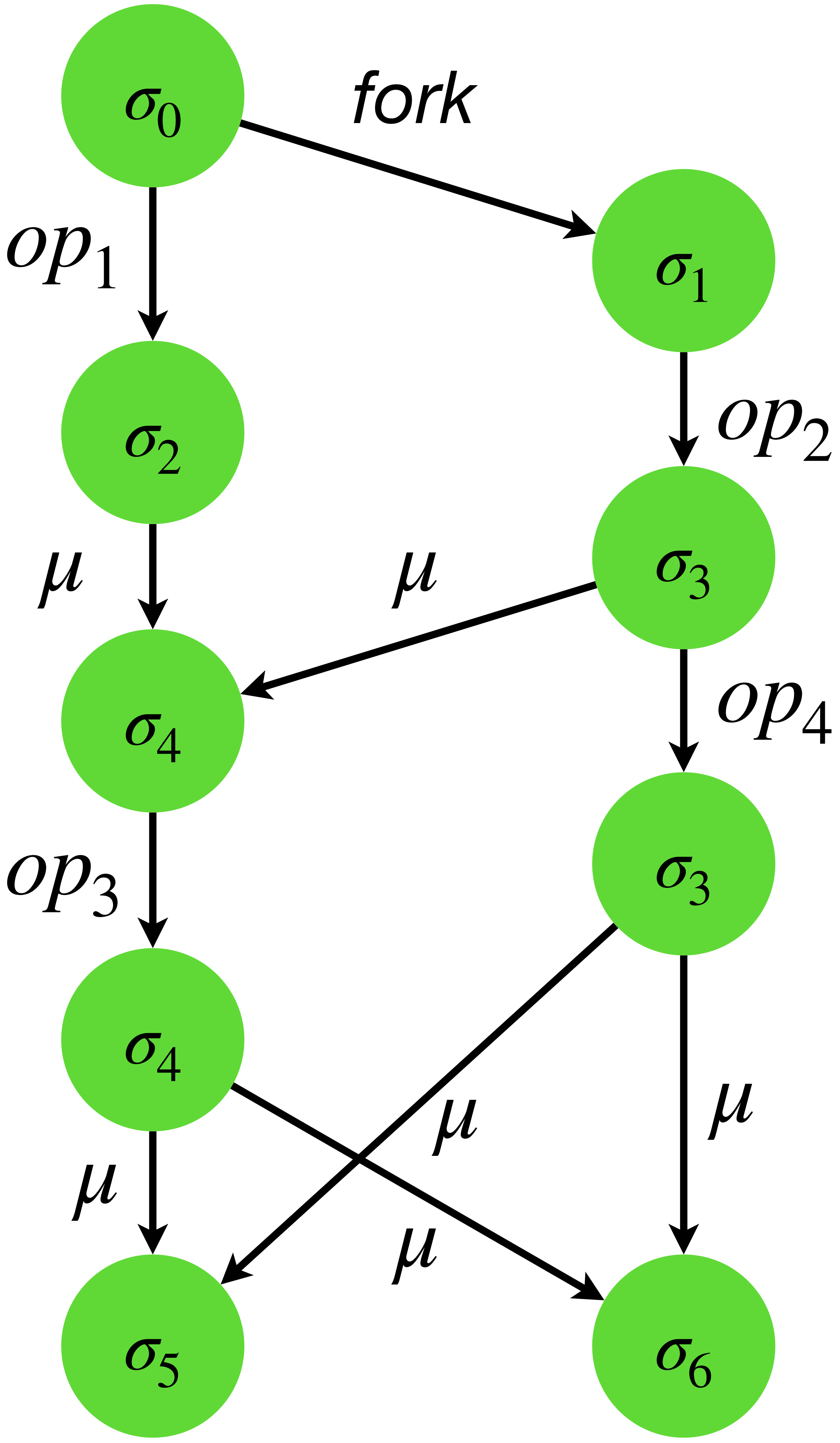
Sequential data type + operations + 3-way merge function

Implementation

Complex

Manual

Is there a more natural 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

- 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

Add-wins set CRDT

- **Add-wins set**
  - A concurrent set where add-wins in a concurrent add(e) and rem(e)

$$(\Sigma_a, \Sigma_r) \xrightarrow{\text{add}(a)} (\Sigma_a \cup \{(a, id)\}, \Sigma_r) \quad \text{where id is fresh}$$

$$(\Sigma_a, \Sigma_r) \xrightarrow{\text{rem}(a)} (\Sigma_a, \Sigma_r \cup \{\ (a, id) \mid (a, id) \in \Sigma_a\ \}$$

$$(\Sigma_a, \Sigma_r) \xrightarrow{\text{read}()} \{a \mid (a, id) \in \Sigma_a \backslash \Sigma_r\}$$

Add-wins set *sequential* specification

- States are asynchronously broadcast to other replicas

$$(\Sigma_a, \Sigma_r) \xrightarrow{\text{merge}(\Sigma_a', \Sigma_r')} (\Sigma_a \cup \Sigma_a', \Sigma_r \cup \Sigma_r')$$

Replication-aware Linearizability

A history $h = (E, vis)$, $E \subseteq \text{Queries} \uplus \text{Updates}$, is RA-linearizable w.r.t. a sequential specification Spec if there exists a total order seq on E (same events) such that:

(i) $vis \cup seq$ is acyclic;

(ii) $seq\downarrow_{Updates} \in \text{Spec}$;

(iii) $\forall \ell_{qr} \in E, \left(seq\downarrow_{\text{vis}^{-1}(\ell_{qr}) \cap Updates}\right) \cdot \ell_{qr} \in \text{Spec}$.

- A CRDT is said to be RA-linearizable if every history *h* is RA-linearizable
- Add-wins set is RA-linearizable
- *RA-linearizability makes program reasoning easier!*

Using RA-linearizability for verification

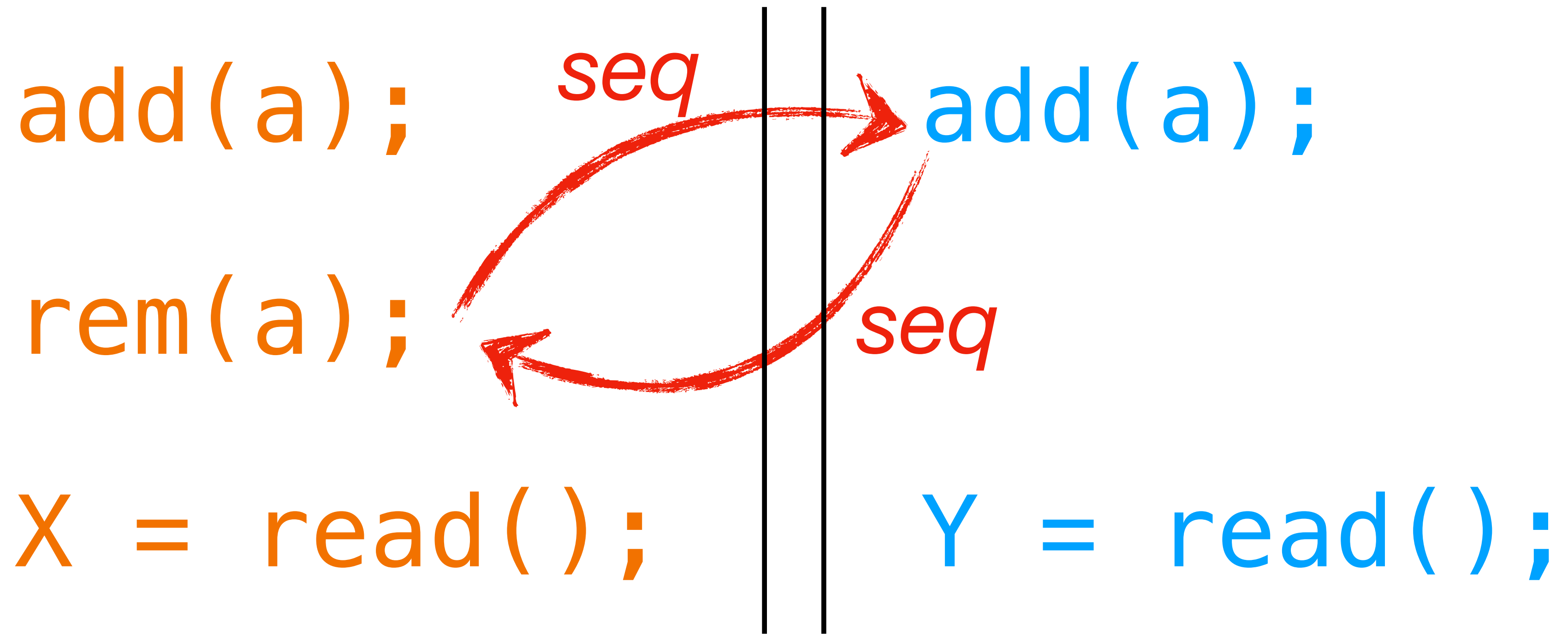
```
add(a);          ||  add(a);
rem(a);          ||
X = read();      ||  Y = read();
```

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

- Since Add-wins set is RA-linearizable, you can use *totally ordered trace* and the *sequential spec* to reason about correctness

`add(a);`	`rem(a);`	`add(a);`	`X = read();`	`Y = read()`
$(\{a_1\},\{\})$	$(\{a_1\},\{a_1\})$	$(\{a_1,a_2\},\{a_1\})$	$X = \{a\}$	$Y = \{a\}$

Using RA-linearizability for verification



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

- Let's try to make the statement false
    - Make $a \in X$ true and $a \in Y$ false

Replication-aware Linearizability

RESEARCH-ARTICLE

Replication-aware linearizability

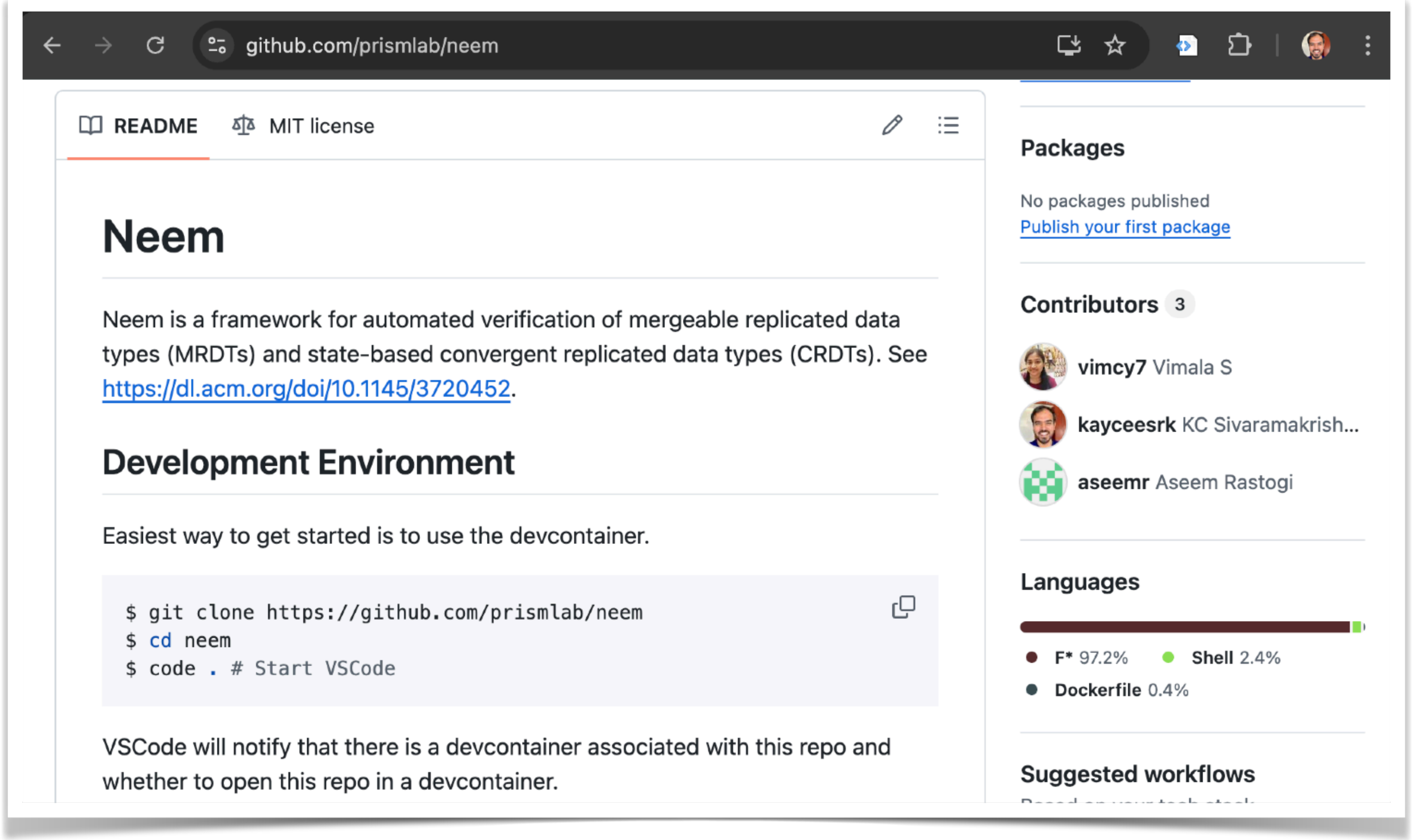
Authors: Chao Wang, Constantin Enea, Suha Orhun Mutluergil, Gustavo Petri | Authors Info & Clai

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

Neem — Automatic verification of RDTs

- **What's in the box?**
  - Definition of RA-linearizability for MRDTs
  - A novel induction scheme for MRDTs and state-based CRDTs to ***automatically*** verify RA-linearizability
  - Implemented in F*



Resolving conflicts

- Not all operations commute
  - **Add-wins set** — `add(a)` and `rem(a)` do not commute
  - Specify ordering using the **Conflict Resolution** relation $rc = \{(rem_a, add_a) \mid a \in \mathbb{E}\}$
  - 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

Increment-only Counter

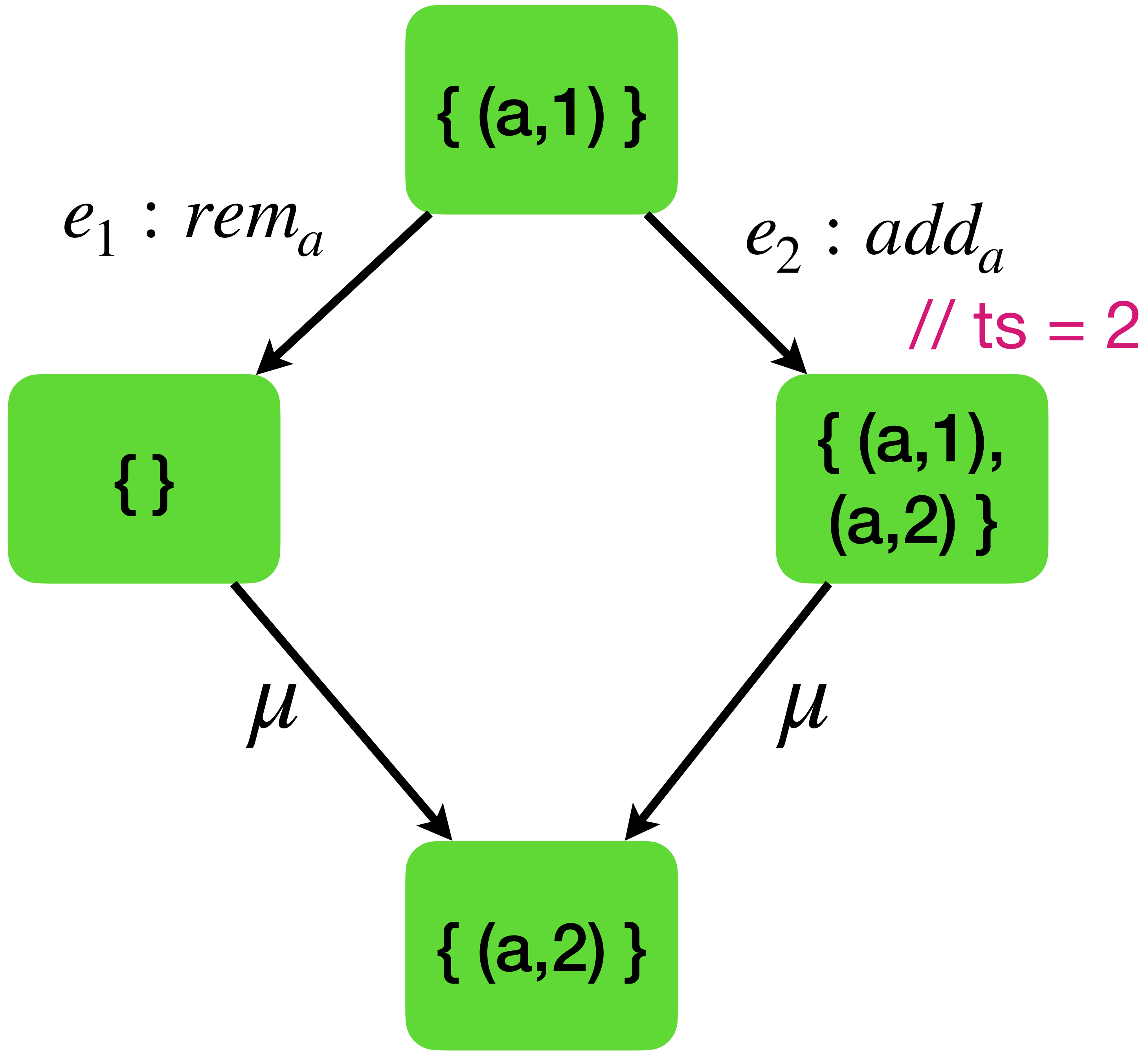
State	$\Sigma = \mathbb{N}$
Updates	$O = \{\mathrm{inc}\}$
Queries	$Q = \{\mathrm{rd}\}$
Init State	$\sigma_0 = 0$
Update behaviour	$\mathrm{do}(\sigma, \_, \_, \mathrm{inc}) = \sigma + 1$
Merge	$\mathrm{merge}(\sigma_\top, \sigma_1, \sigma_2) = \sigma_\top + (\sigma_1 - \sigma_\top) + (\sigma_2 - \sigma_\top)$
Query behaviour	$\mathrm{query}(\sigma, rd) = \sigma$
Resolve conflict	$\mathrm{rc} = \emptyset$

Unique timestamp

Replica ID

Add-wins Set

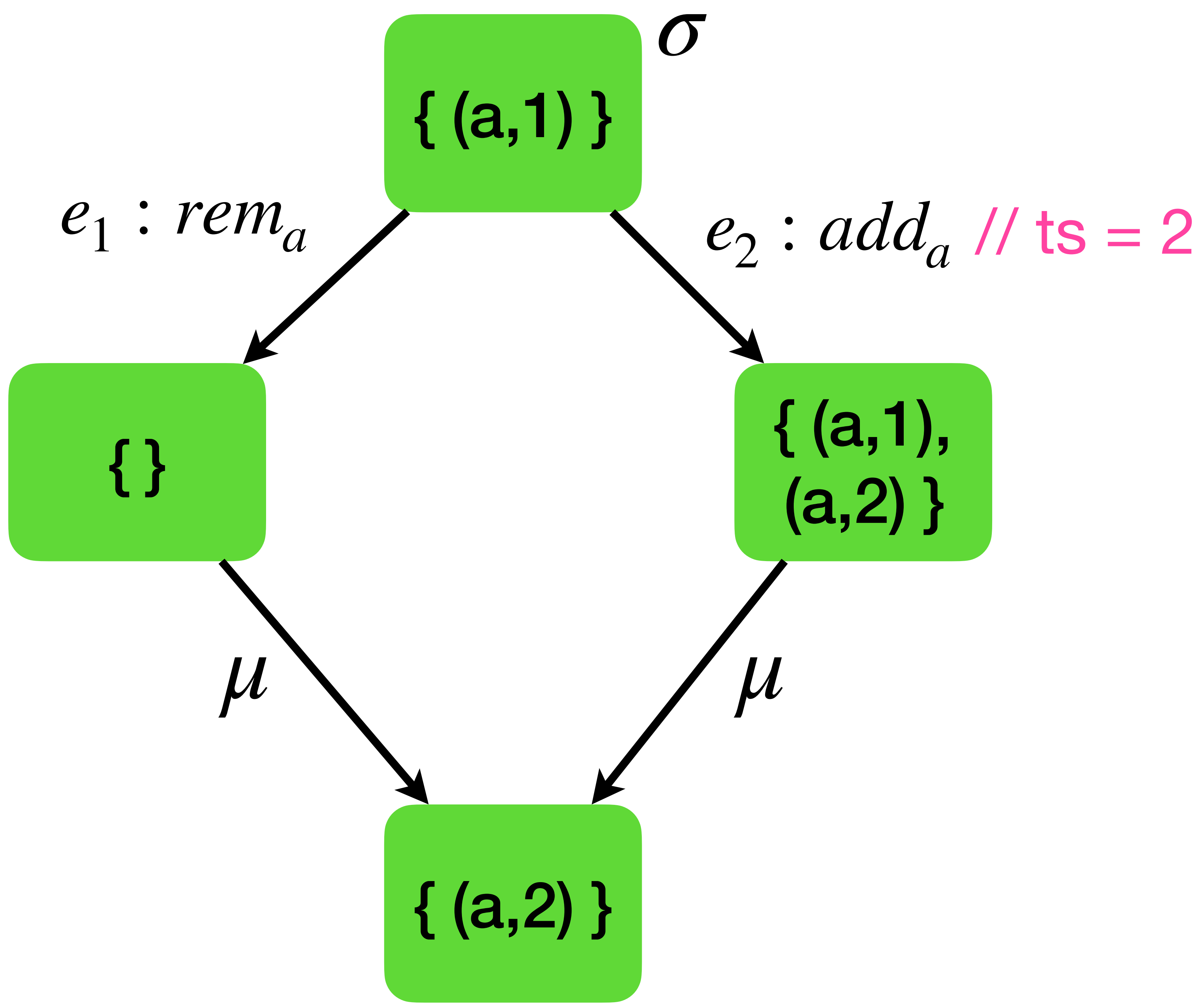
State	1: $\Sigma = \mathcal{P}(\mathbb{E} \times \mathcal{T})$
Updates	2: $O = \{\text{add}_a, \text{rem}_a \mid a \in \mathbb{E}\}$
Queries	3: $Q = \{\text{rd}\}$
Init State	4: $\sigma_0 = \{\}$
Update behaviour	5: $\text{do}(\sigma, t, \_, \text{add}_a) = \sigma \cup \{(a, t)\}$ 6: $\text{do}(\sigma, \_, \_, \text{rem}_a) = \sigma \setminus \{(a, i) \mid (a, i) \in \sigma\}$
Merge	7: $\text{merge}(\sigma_\top, \sigma_1, \sigma_2) = (\sigma_\top \cap \sigma_1 \cap \sigma_2) \cup (\sigma_1 \setminus \sigma_\top) \cup (\sigma_2 \setminus \sigma_\top)$
Query behaviour	8: $\text{query}(\sigma, rd) = \{a \mid (a, \_) \in \sigma\}$
Resolve conflict	9: $\text{rc} = \{(\text{rem}_a, \text{add}_a) \mid a \in \mathbb{E}\}$



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

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]

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

[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]

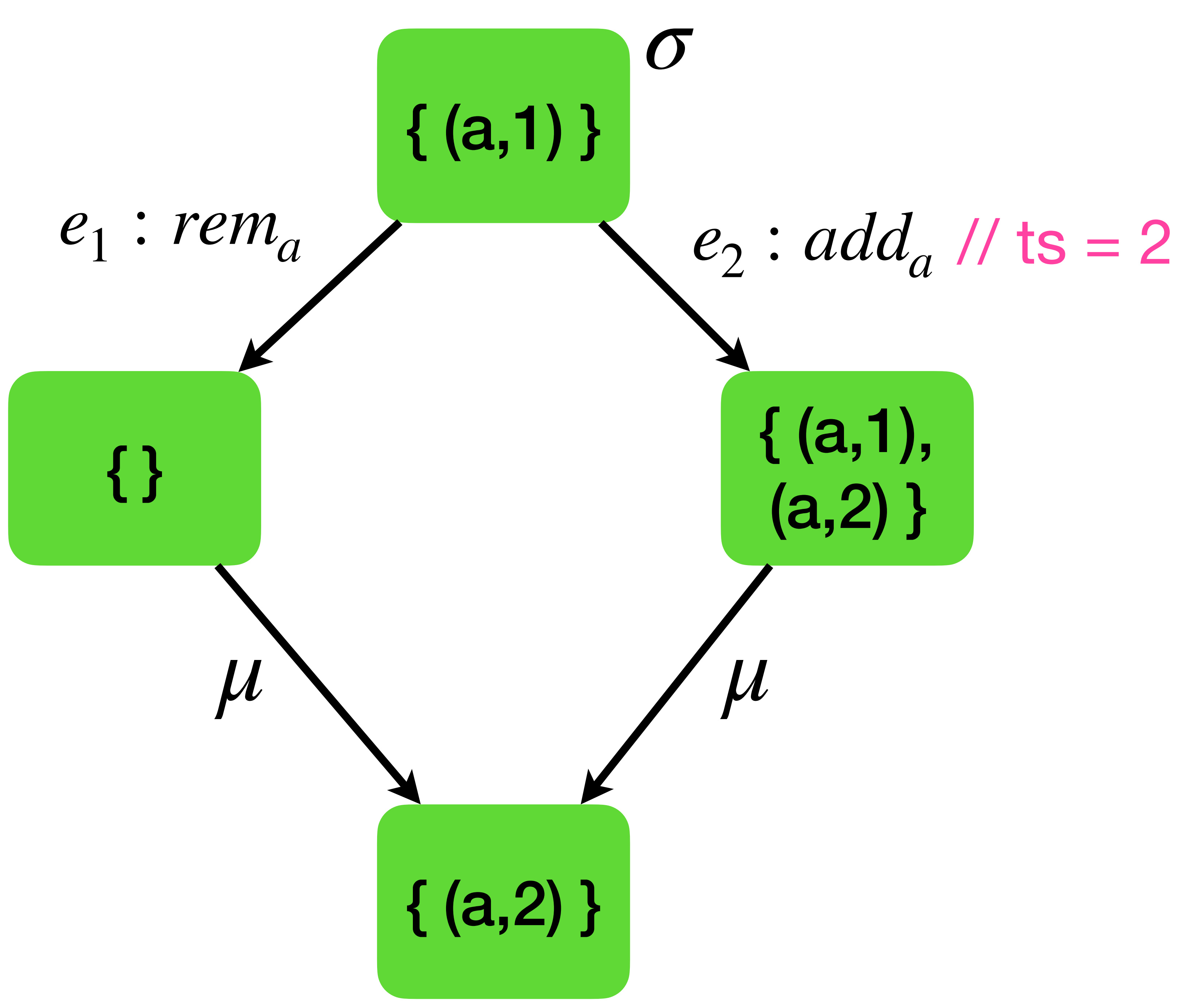
$$\mu(a, a, a) = a$$

[MergeCommutativity]

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

Bottom up linearisation

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



To show

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

[BottomUp-2-OP]

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

[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]

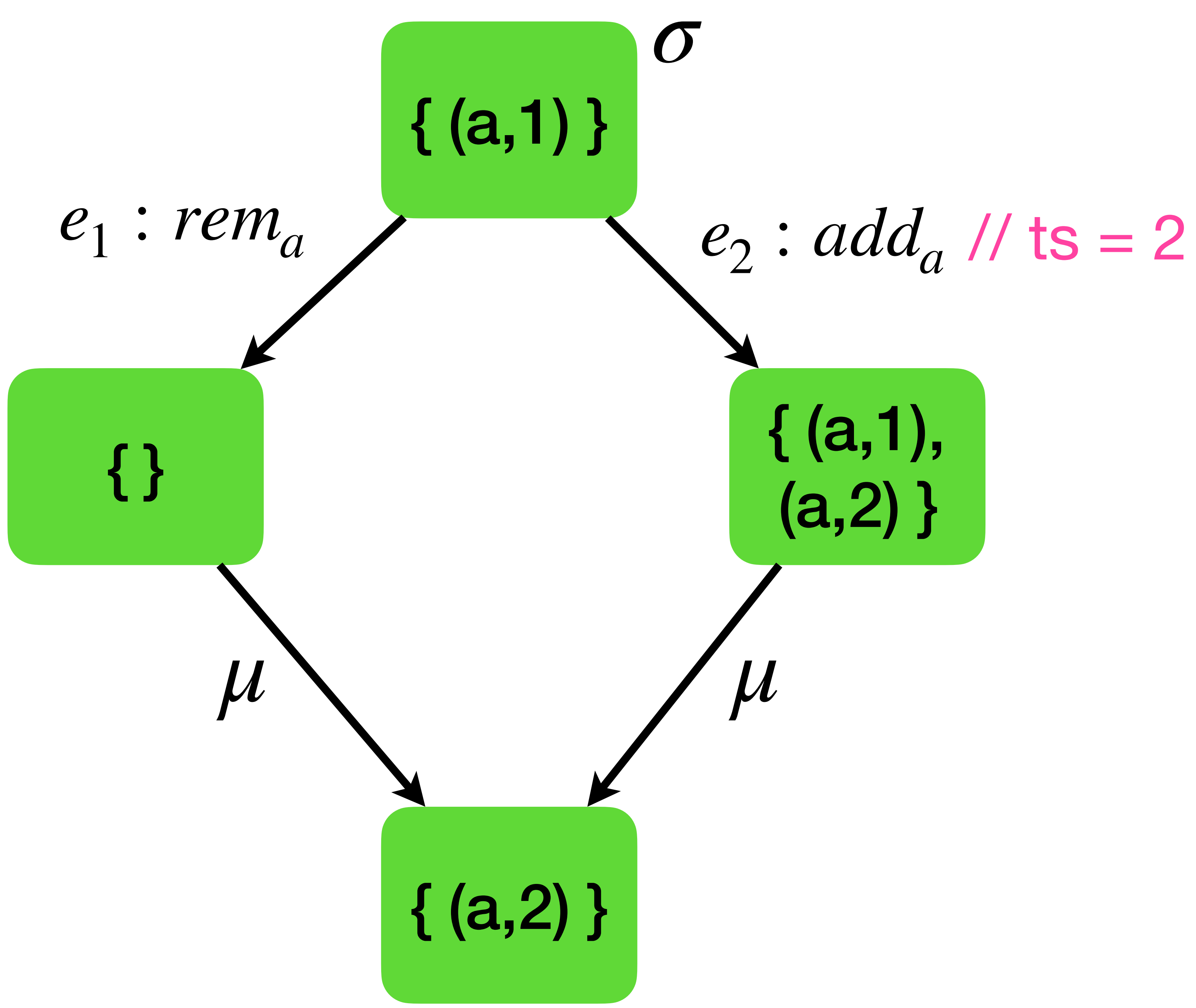
$$\mu(a, a, a) = a$$

[MergeCommutativity]

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

Bottom up linearisation

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



To show

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

[BottomUp-2-OP]

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

[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]

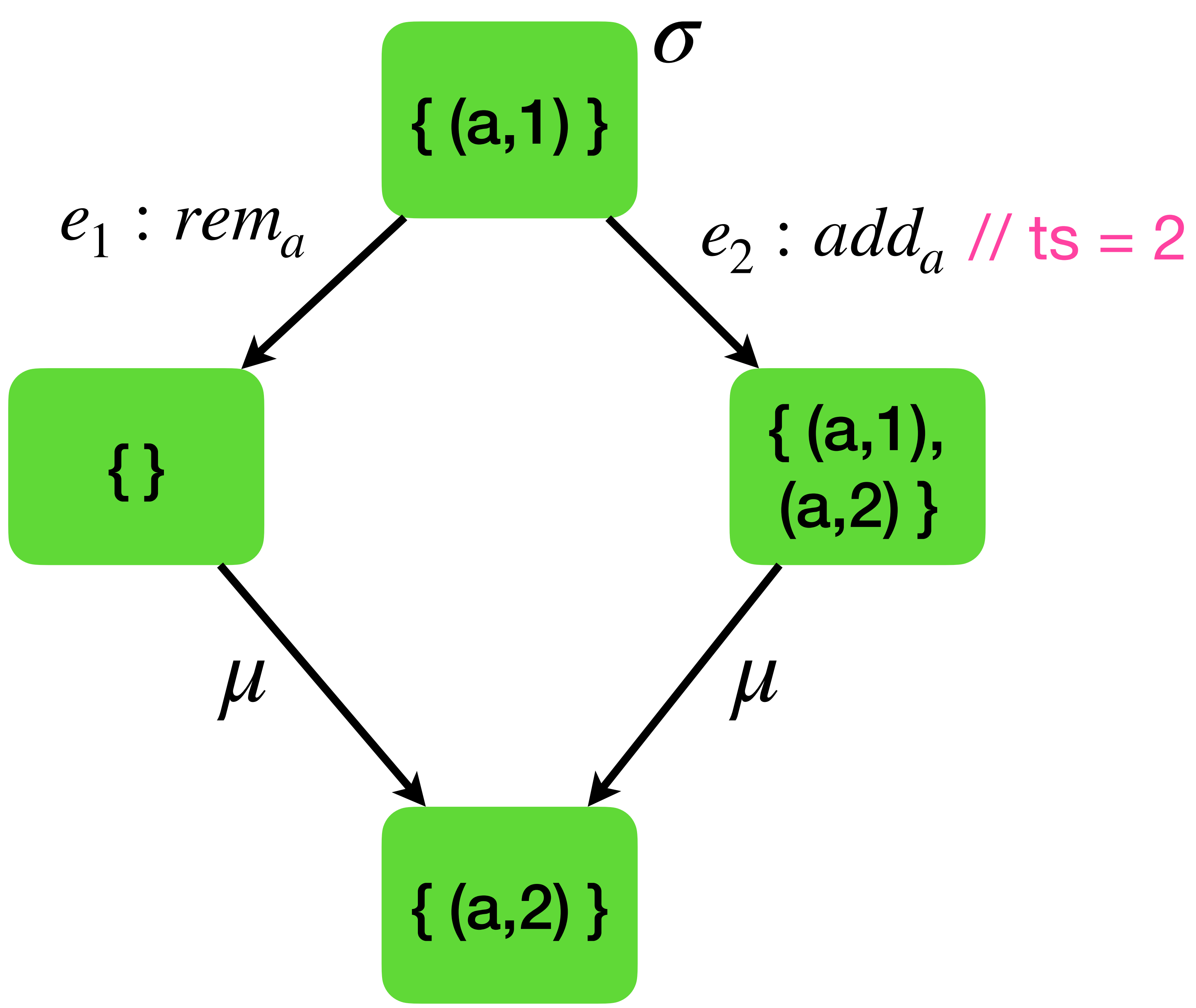
$$\mu(a, a, a) = a$$

[MergeCommutativity]

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

Bottom up linearisation

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



To show

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

[BottomUp-2-OP]

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

[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]

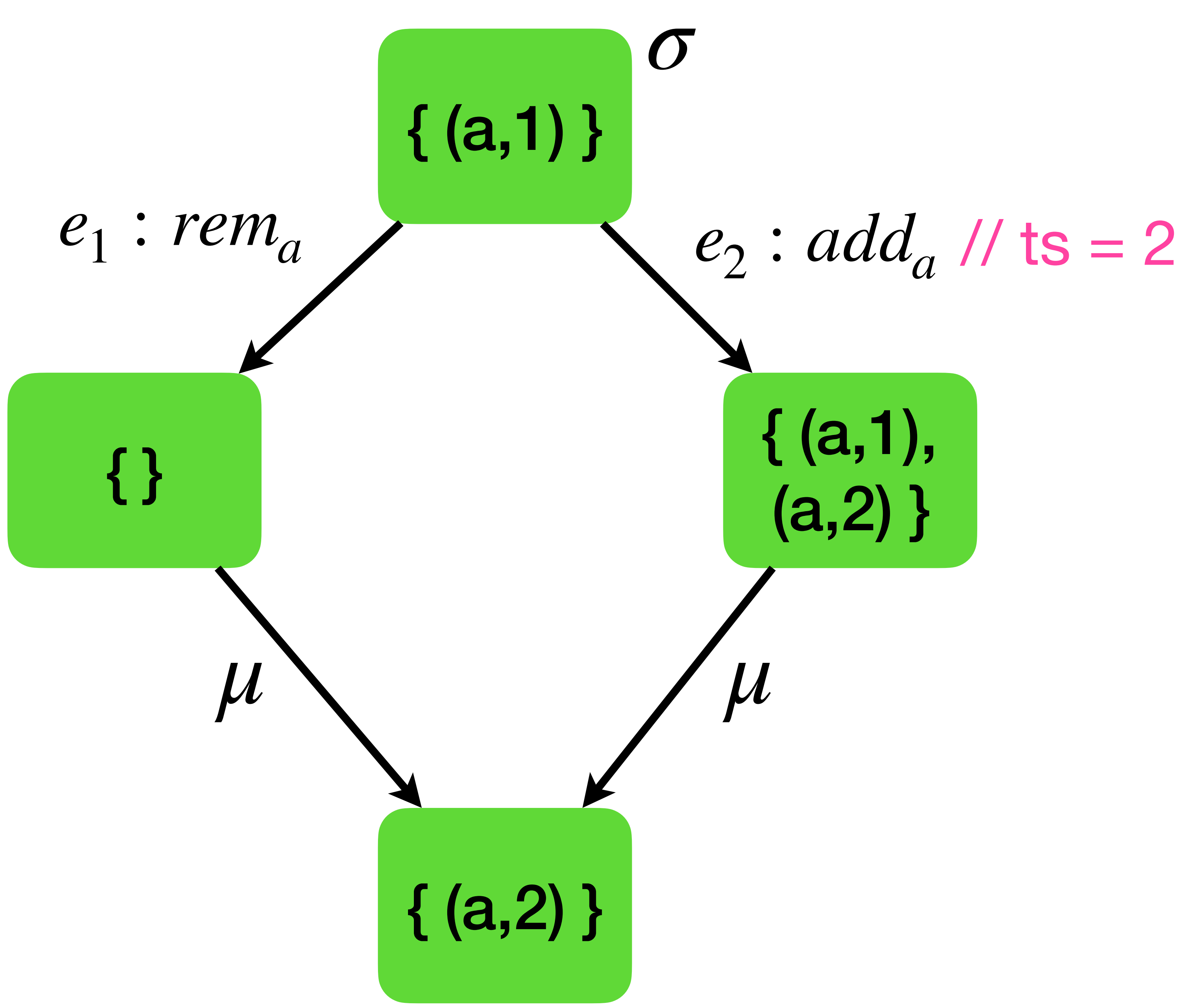
$$\mu(a, a, a) = a$$

[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]

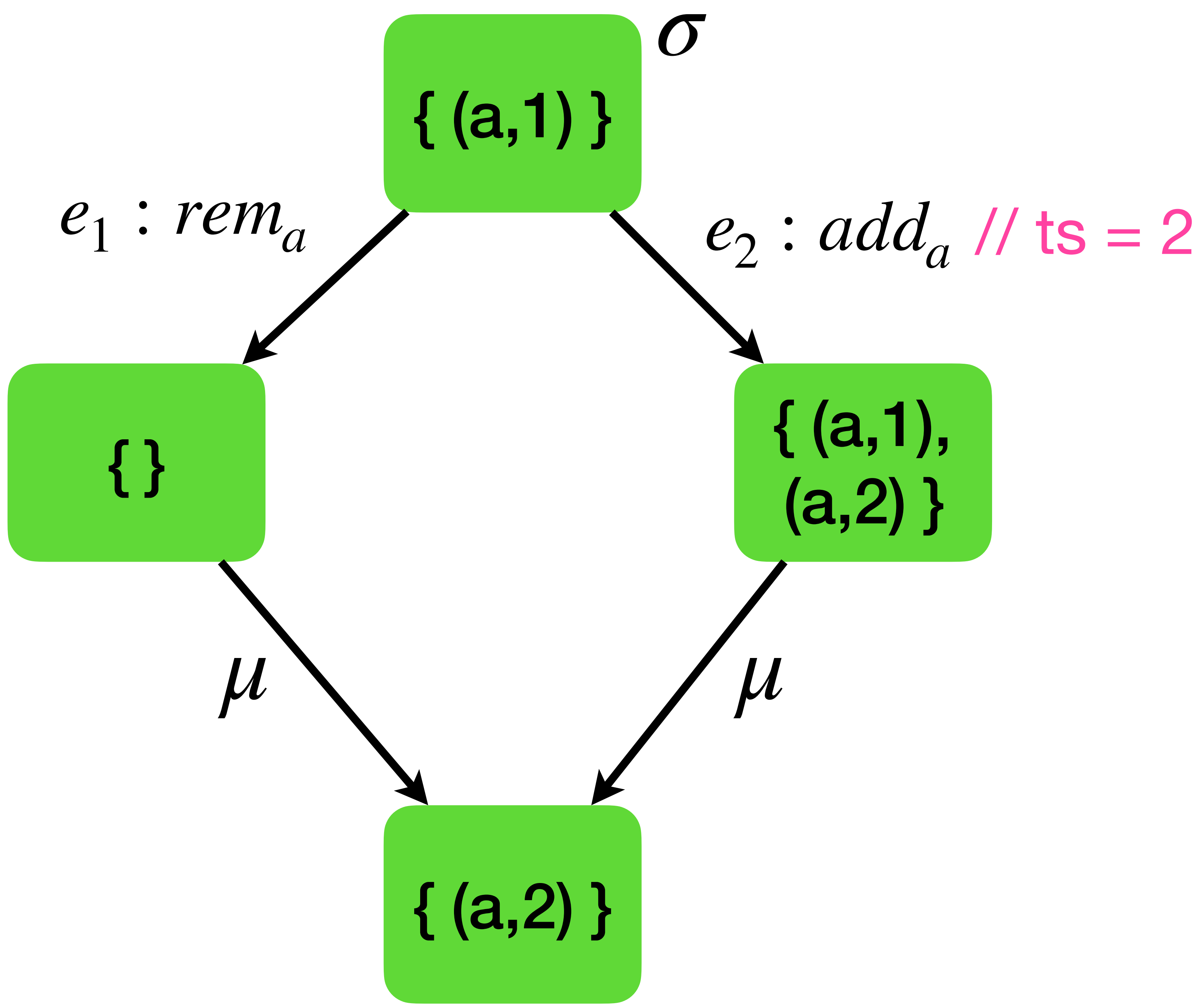
$$\mu(a, a, a) = a$$

[MergeCommutativity]

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

Bottom up linearisation

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



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]

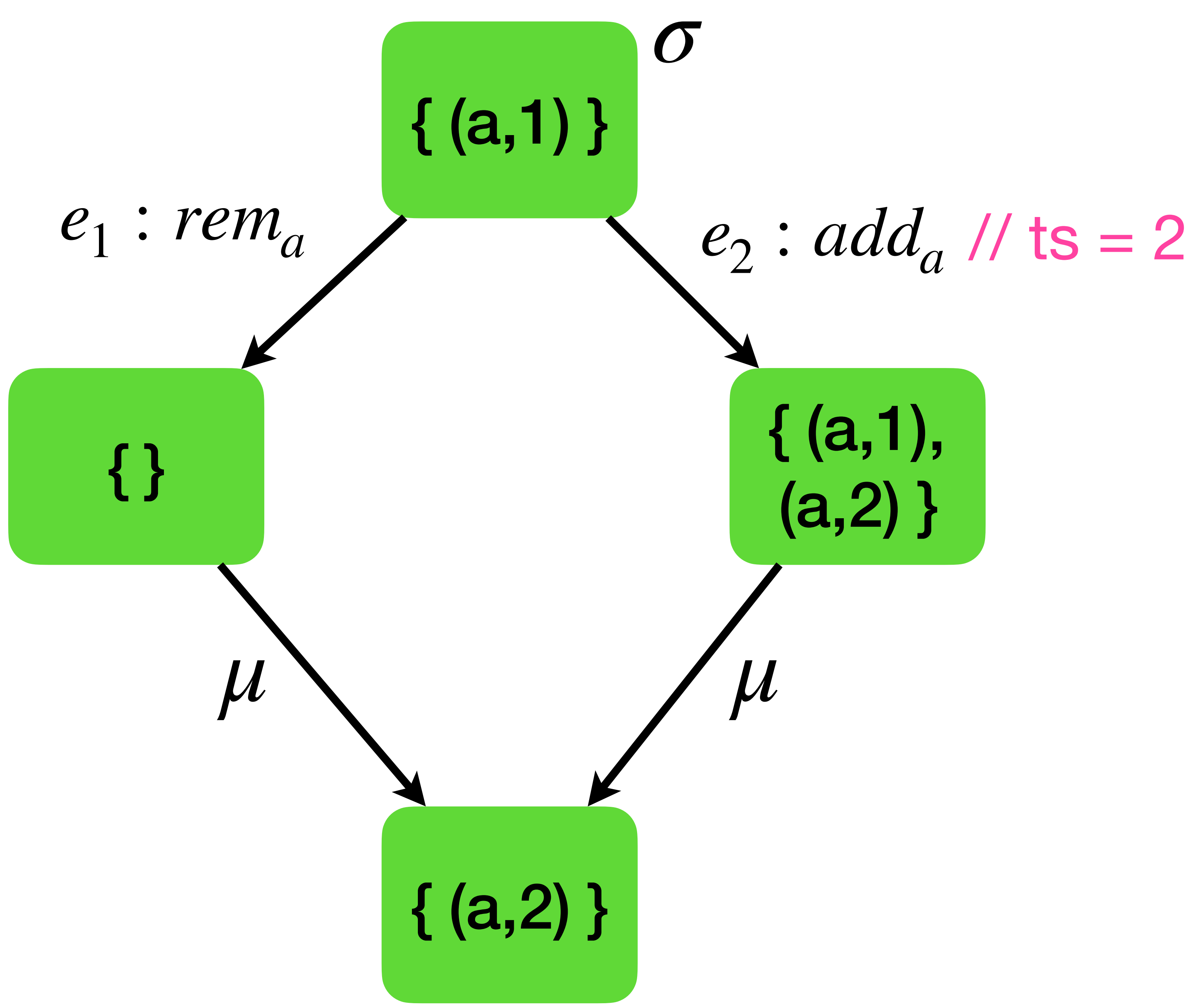
$$\mu(a, a, a) = a$$

[MergeCommutativity]

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

Bottom up linearisation

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



To show

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

[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]

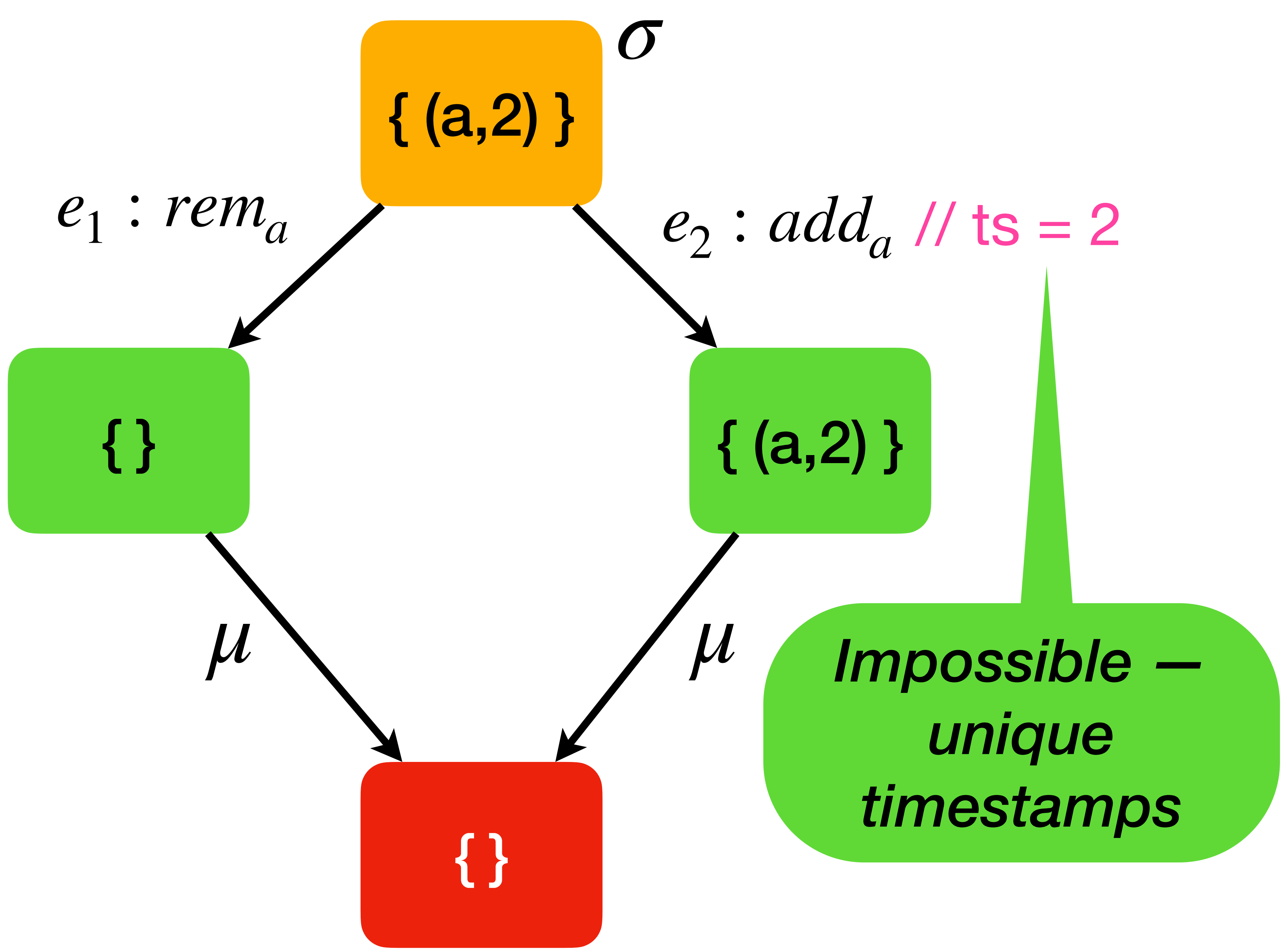
$$\mu(a, a, a) = a$$

[MergeCommutativity]

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

Making a good VC

$$\mathsf{rc} = \{(\mathsf{rem}_a, \mathsf{add}_a) \mid a \in \mathbb{E}\}$$



To show

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

$$\{\} \neq \{(a,2)\}$$

[BottomUp-2-OP]

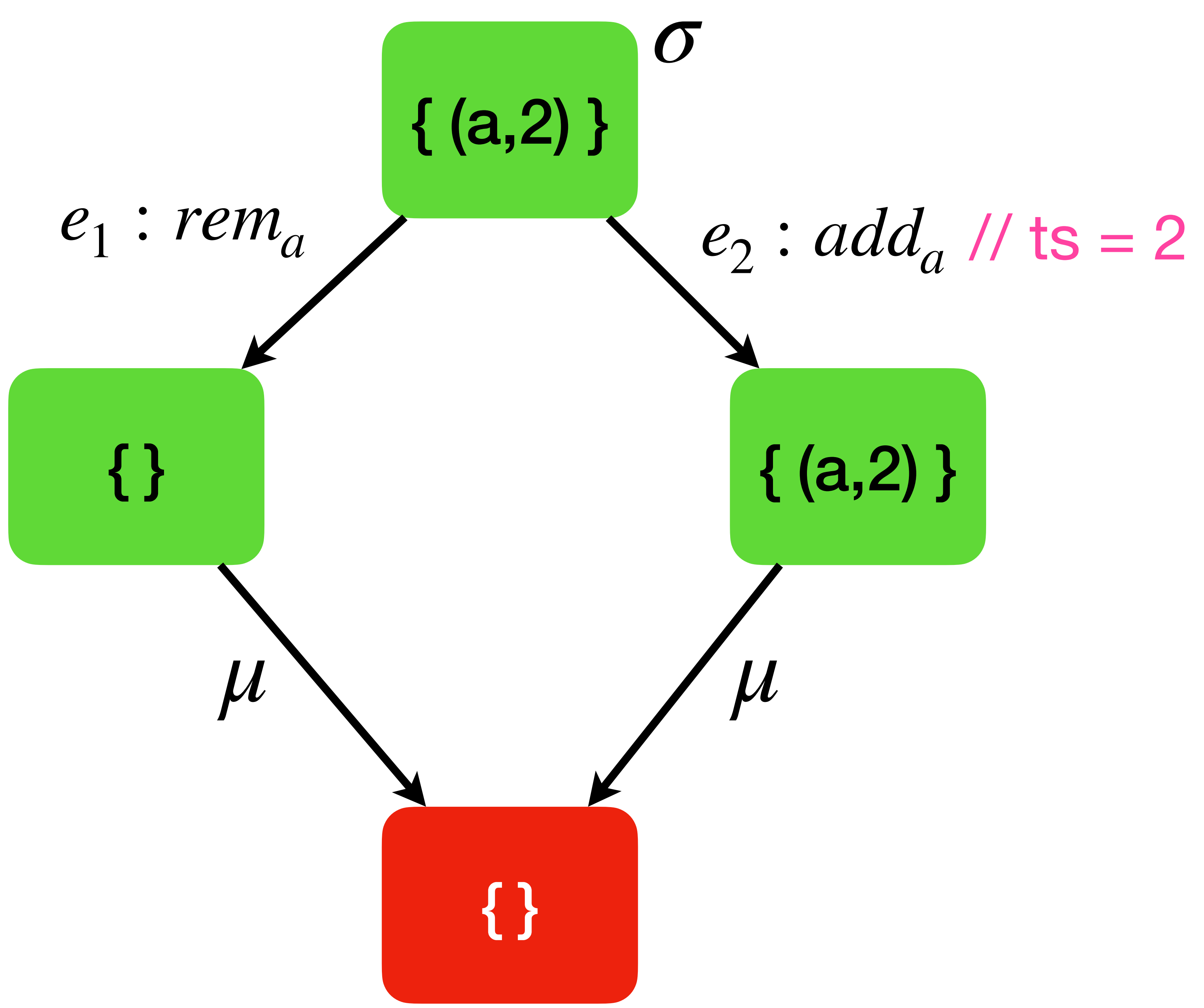
$$\frac{e_1 \neq e_2 \qquad e_1 \xrightarrow{\mathsf{rc}} e_2 \vee e_1 \rightleftarrows e_2}{\mu(l, e_1(a), e_2(b)) = e_2(\mu(l, e_1(a), b))}$$

Cannot prove for an arbitrary l

l must be a ***feasible state***, obtained by application of updates on the initial state

Induction over event sequences

$$\mathsf{rc} = \{(\mathsf{rem}_a, \mathsf{add}_a) \mid a \in \mathbb{E}\}$$



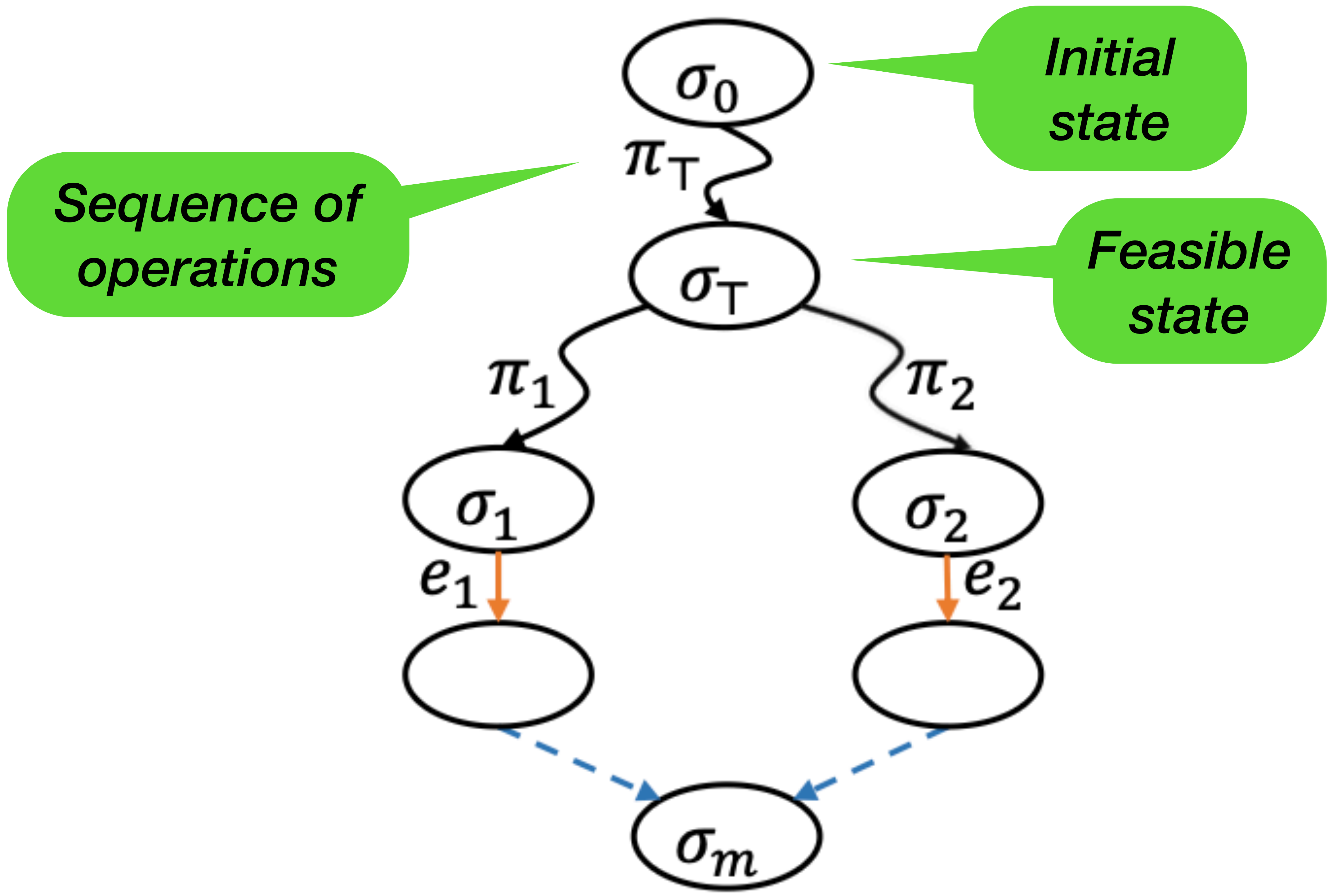
To show

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

$$\{\} \neq \{(a,2)\}$$

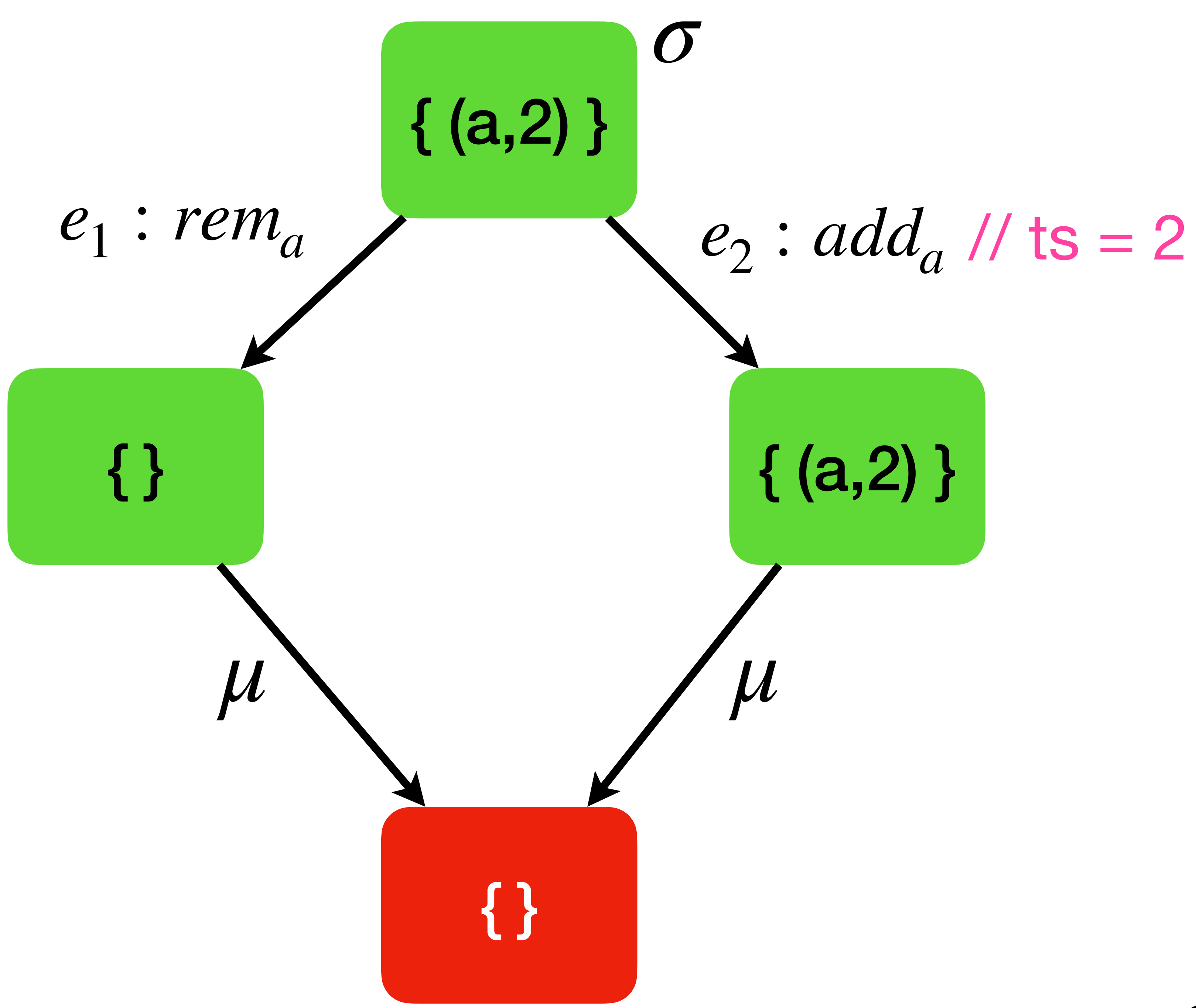
[BottomUp-2-OP]

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



Induction over event sequences

$$\mathsf{rc} = \{(\mathsf{rem}_a, \mathsf{add}_a) \mid a \in \mathbb{E}\}$$



To show

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

$$\{\} \neq \{(a,2)\}$$

[BottomUp-2-OP]

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

Induction on $\pi_\top$

Base case

$$\frac{< pre >}{\mu(\sigma_0, e_1(\sigma_0), e_2(\sigma_0)) = e_2(\mu(\sigma_0, e_1(\sigma_0), \sigma_0))}$$

$(a,2) \notin \sigma_0$

Inductive case

$$\frac{< pre > \qquad \mu(\sigma_t, e_1(\sigma_t), e_2(\sigma_t)) = e_2(\mu(\sigma_t, e_1(\sigma_t), \sigma_t)) \qquad \sigma'_t = e(\sigma_t)}{\mu(\sigma'_t, e_1(\sigma'_t), e_2(\sigma'_t)) = e_2(\mu(\sigma'_t, e_1(\sigma'_t), \sigma'_t))}$$

Timestamps are unique

Linearizable MRDTs

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.*

An MRDT that satisfies the algebraic properties is RA-linearizable

LEMMA 3.10. *If MRDT $\mathcal{D}$ is RA-linearizable, then for all executions $\tau \in [\![\mathcal{S}_{\mathcal{D}}]\!]$, for all transitions $C \xrightarrow{query(r,q,a)} C'$ in τ where $C = \langle N, H, L, G, vis \rangle$, there exists a sequence π consisting of all events in $L(H(r))$ such that $\mathsf{lo}(C)_{|L(H(r))} \subseteq \pi$ and $a = \mathsf{query}(\pi(\sigma_0), q)$.*

RA-linearizable MRDT query results match those obtained on the linearised updates applied to the initial state

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*	$\text{disable} \xrightarrow{\text{rc}} \text{enable}$	30	29.80
Disable-wins flag*	$\text{enable} \xrightarrow{\text{rc}} \text{disable}$	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*	$\text{unset} \xrightarrow{\text{rc}} \text{set}$	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

Limitations

- Automated verification returns yes / no / ¯_(ツ)_/¯
  - Not pleasant for engineering
  - No counterexamples!
- **Current work**
  - Optimal bounded model checking of MRDTs against RA-linearizability
    - Standard DPOR fails optimality
  - Moving to Lean — ITP with SMT backend, proof reconstruction, “Loom”, etc.

Neem — Automatic verification of RDTs

- **What's in the box?**
  - Definition of RA-linearizability for MRDTs
  - A novel induction scheme for MRDTs and state-based CRDTs to ***automatically*** verify RA-linearizability
  - Implemented in F*