Automatically Verifying Replicated Data Types

KC Sivaramakrishnan

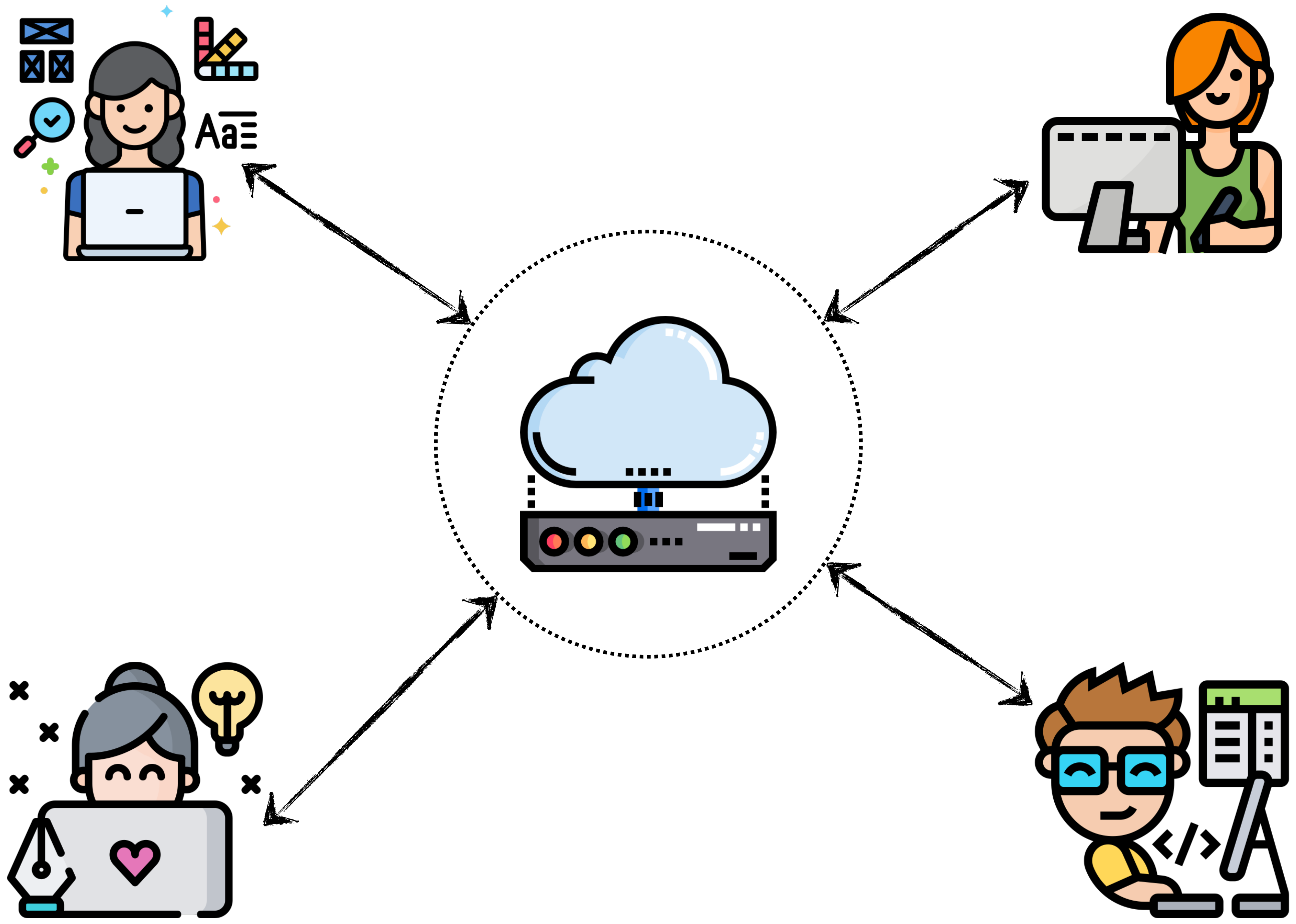
Joint work with Vimala Soundarapandian, Aseem Rastogi and Kartik Nagar

WG 2.8 12/05/2025

Will appear at OOPSLA 2025

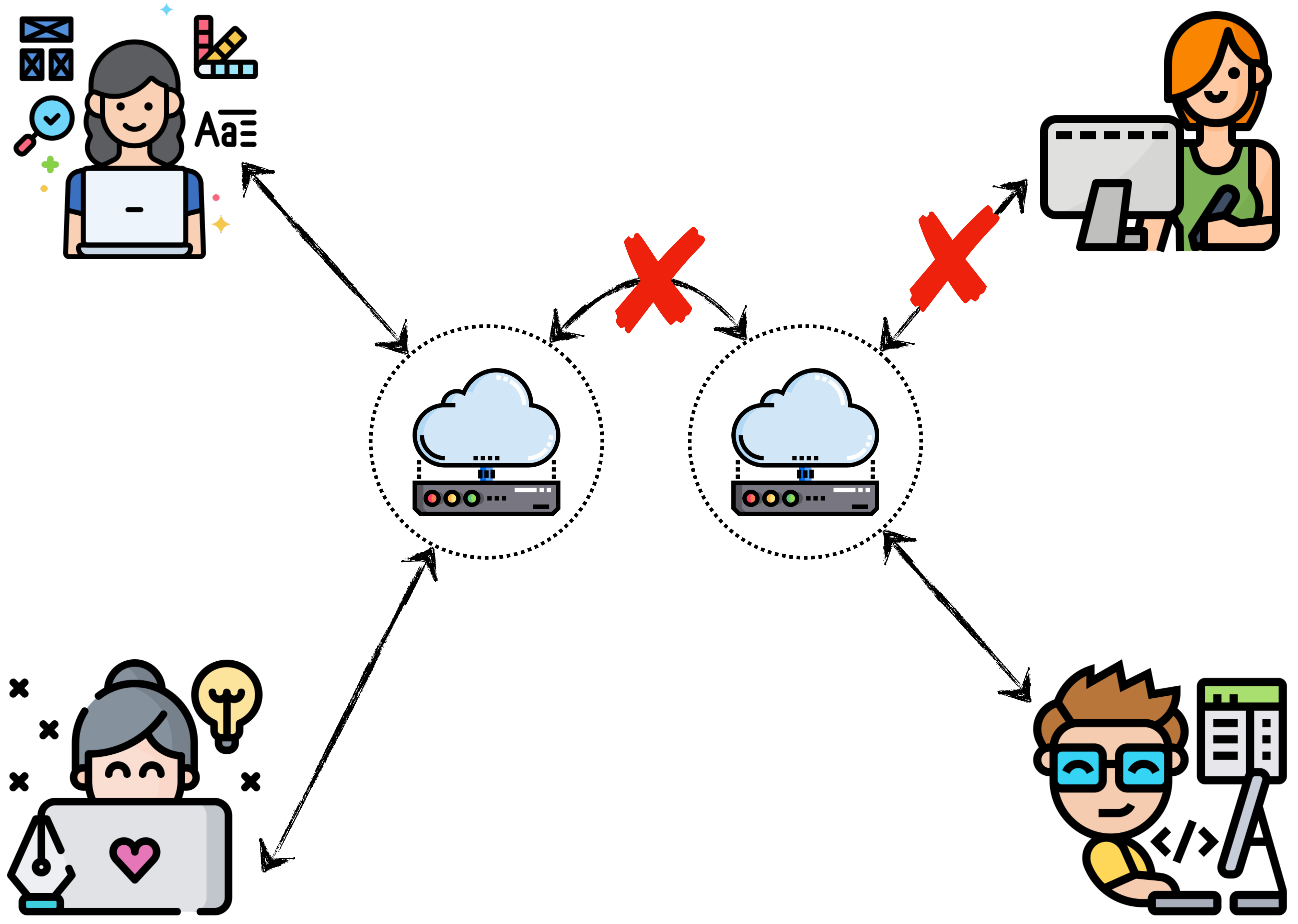
Collaborative Applications

Airtable Google Docs Figma Notion Overleaf



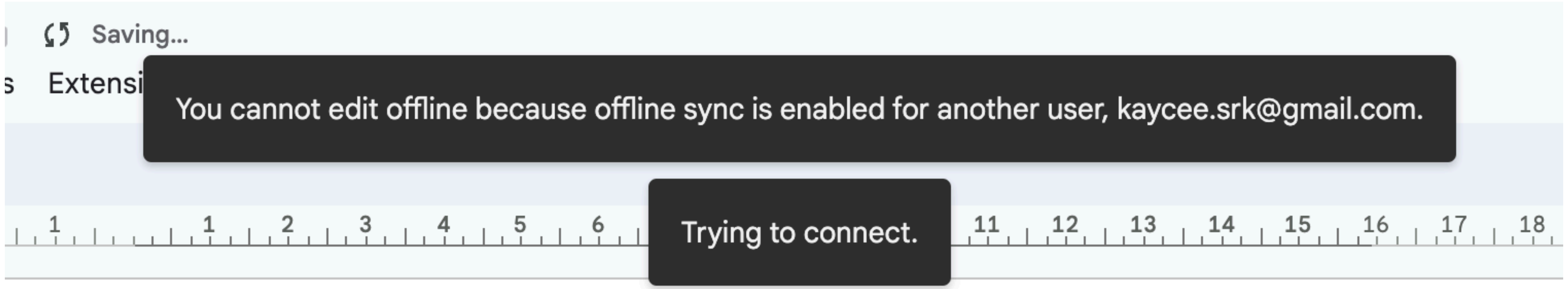
Collaborative Applications

Airtable Google Docs Figma Notion Overleaf



Network Partitions

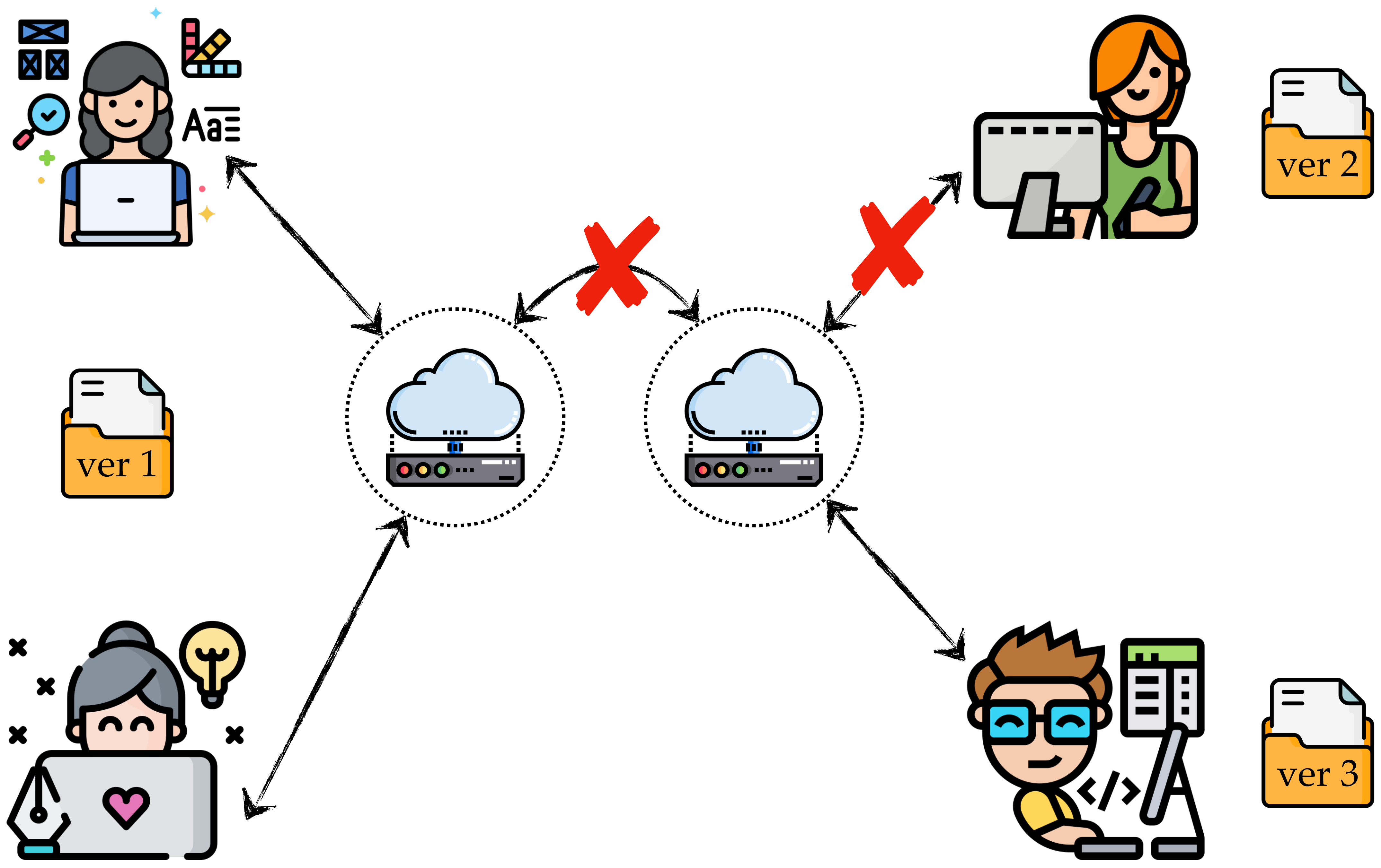
- Centralised Apps provide limited support for offline editing



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

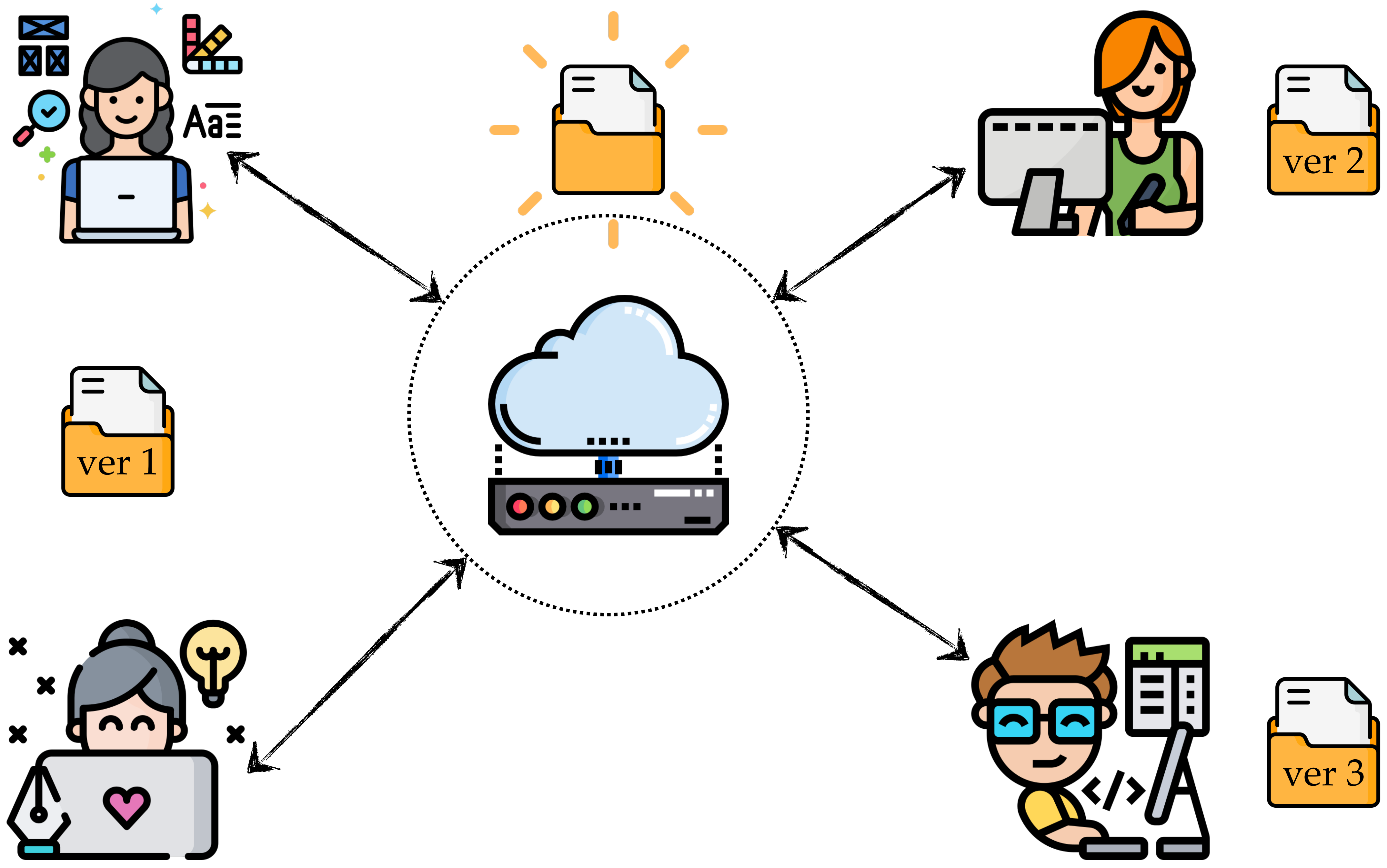
Local-first software

Airtable Google Docs Figma Notion Overleaf



Local-first software

Airtable Google Docs Figma Notion Overleaf



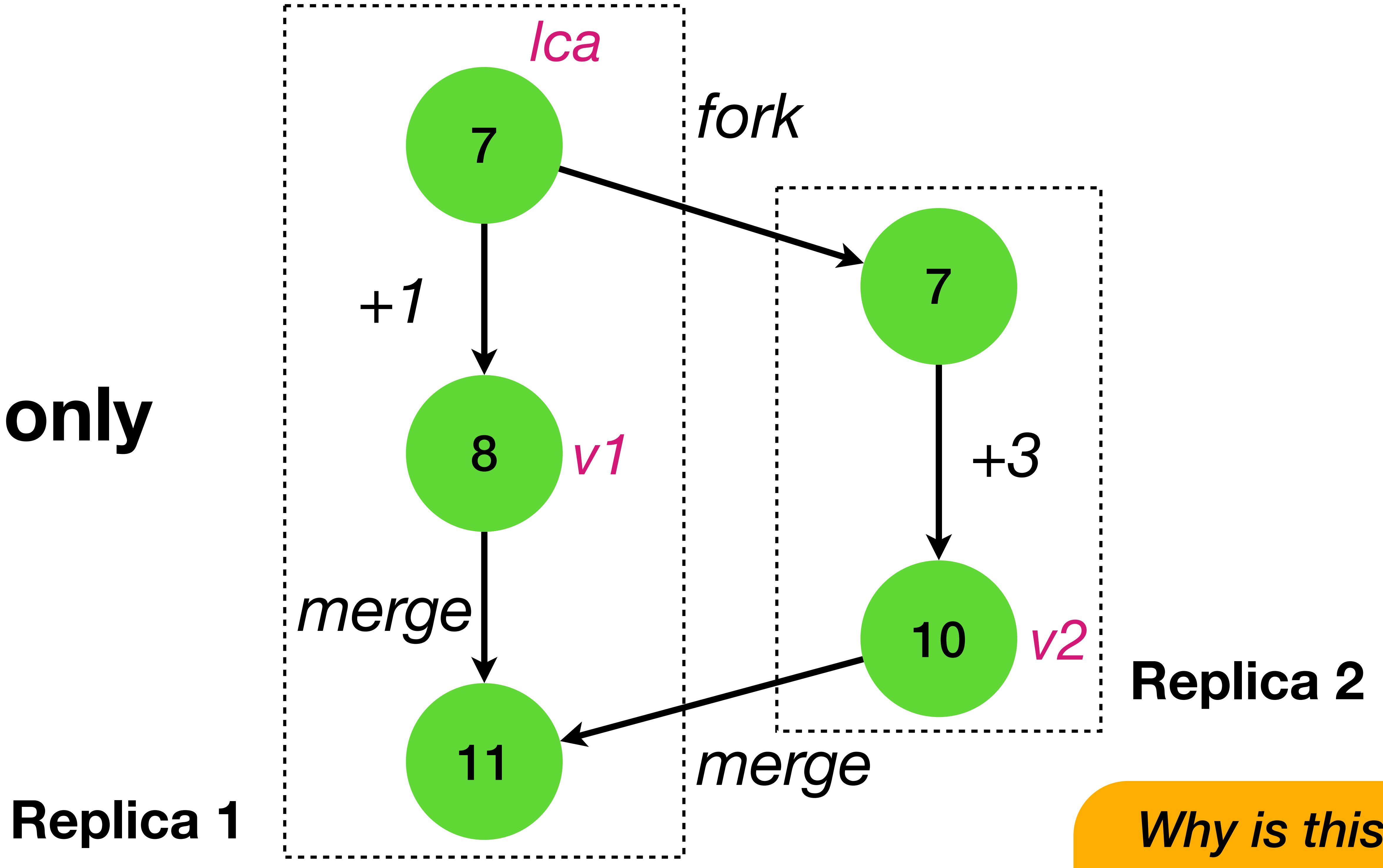
How do we build such applications?

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



Why is this correct?

How do we automatically verify it?

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

Verification using Algebraic Properties

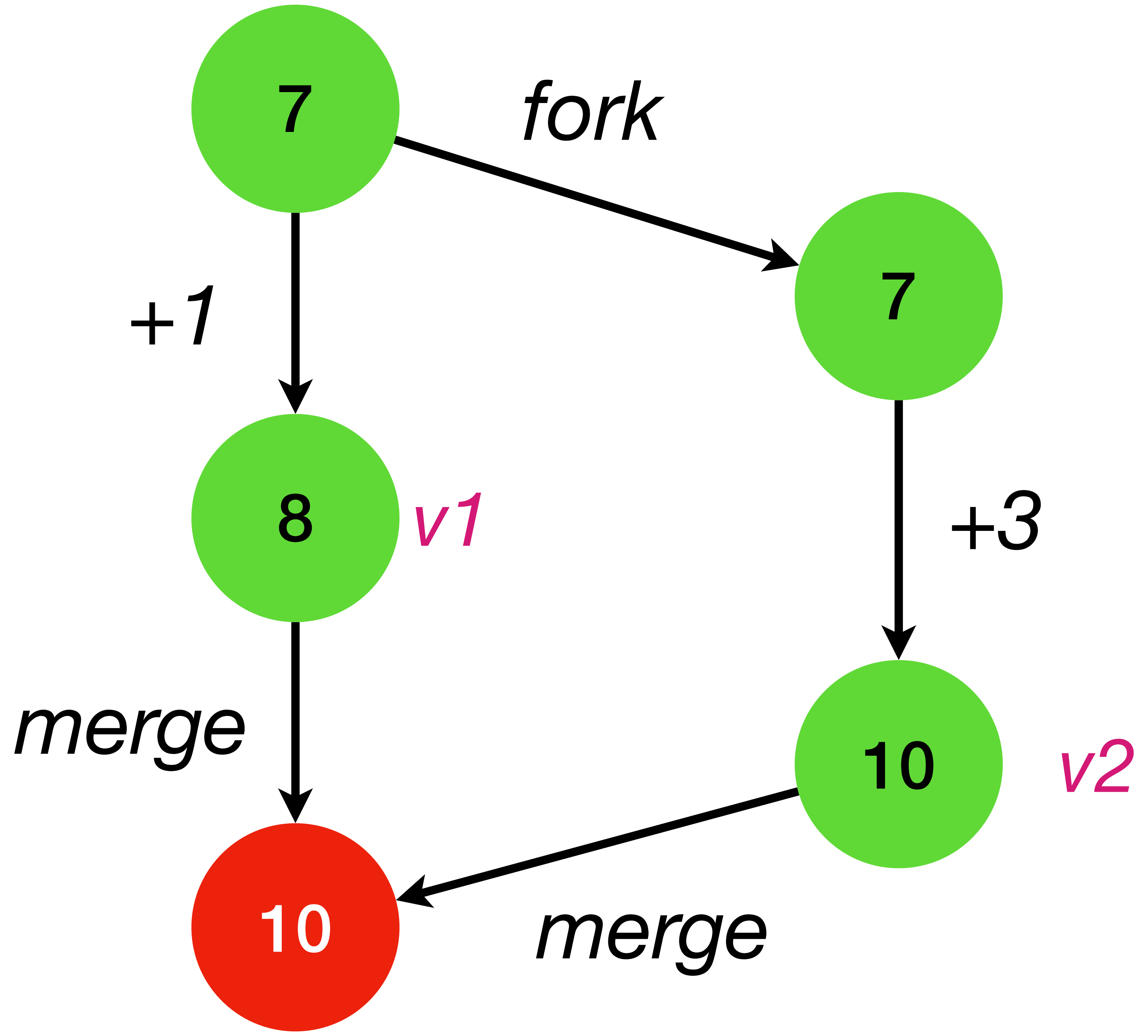
- State-based Convergent Replicated Data Types (CRDTs)
  - Merge is 2-way $- \mu(v_1, v_2)$
  - Verify algebraic properties of merge for *strong eventual consistency*

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



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

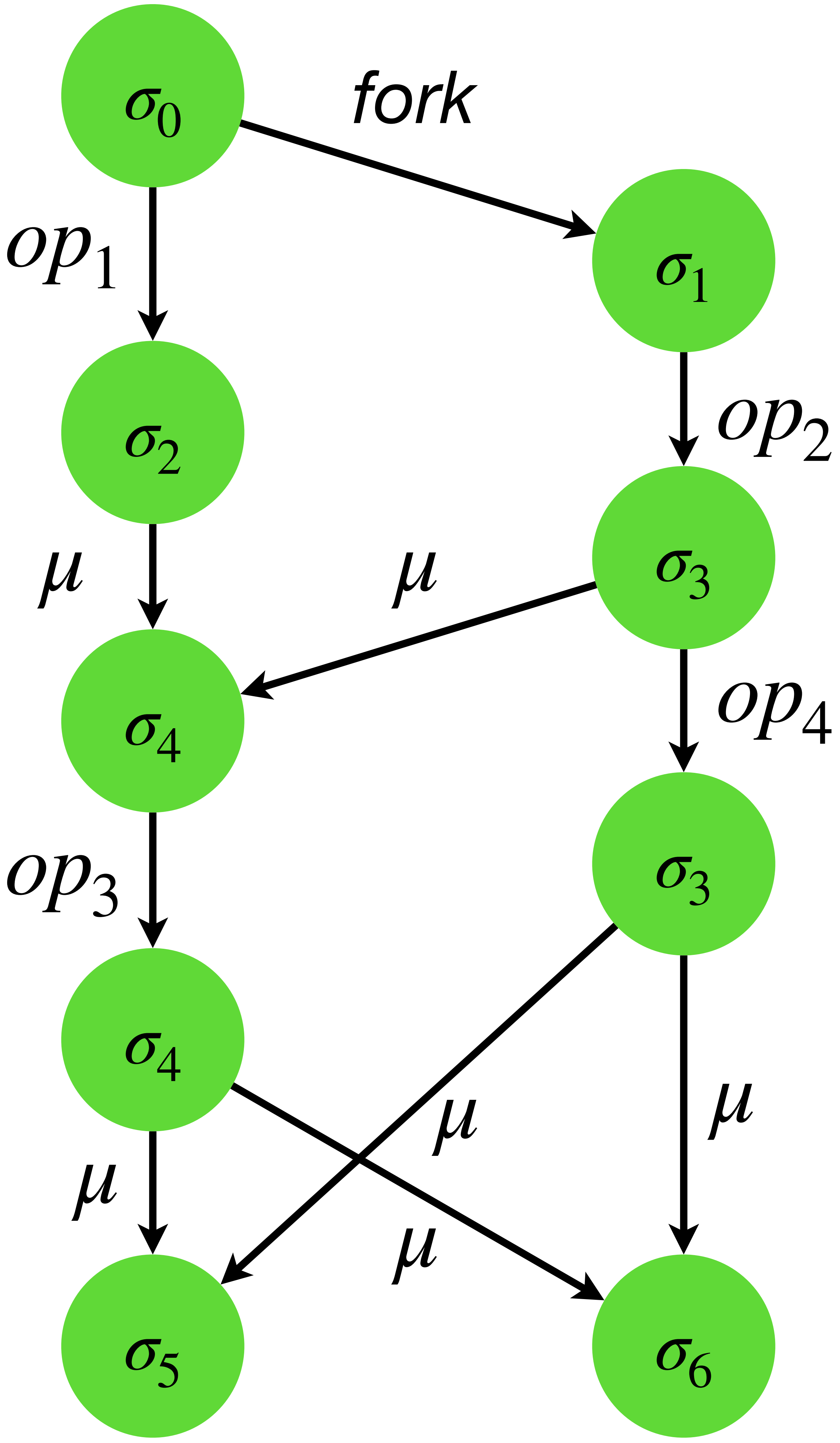
Sequential data type + operations + 3-way merge function

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

- Replica states should be a ***linearisation*** of observed ***update*** operations
    - Use commutativity and asynchrony → Replication-aware (RA) 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

- All replicas should ***converge*** to the same state — Strong Eventual Consistency

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 $rc = \{(rem_a, add_a) \,|\, a \in \mathbb{E}\}$
- Neem developers provide
  - MRDT = Sequential Data Type + 3-way merge
  - Conflict Resolution rc relation

Increment-only Counter

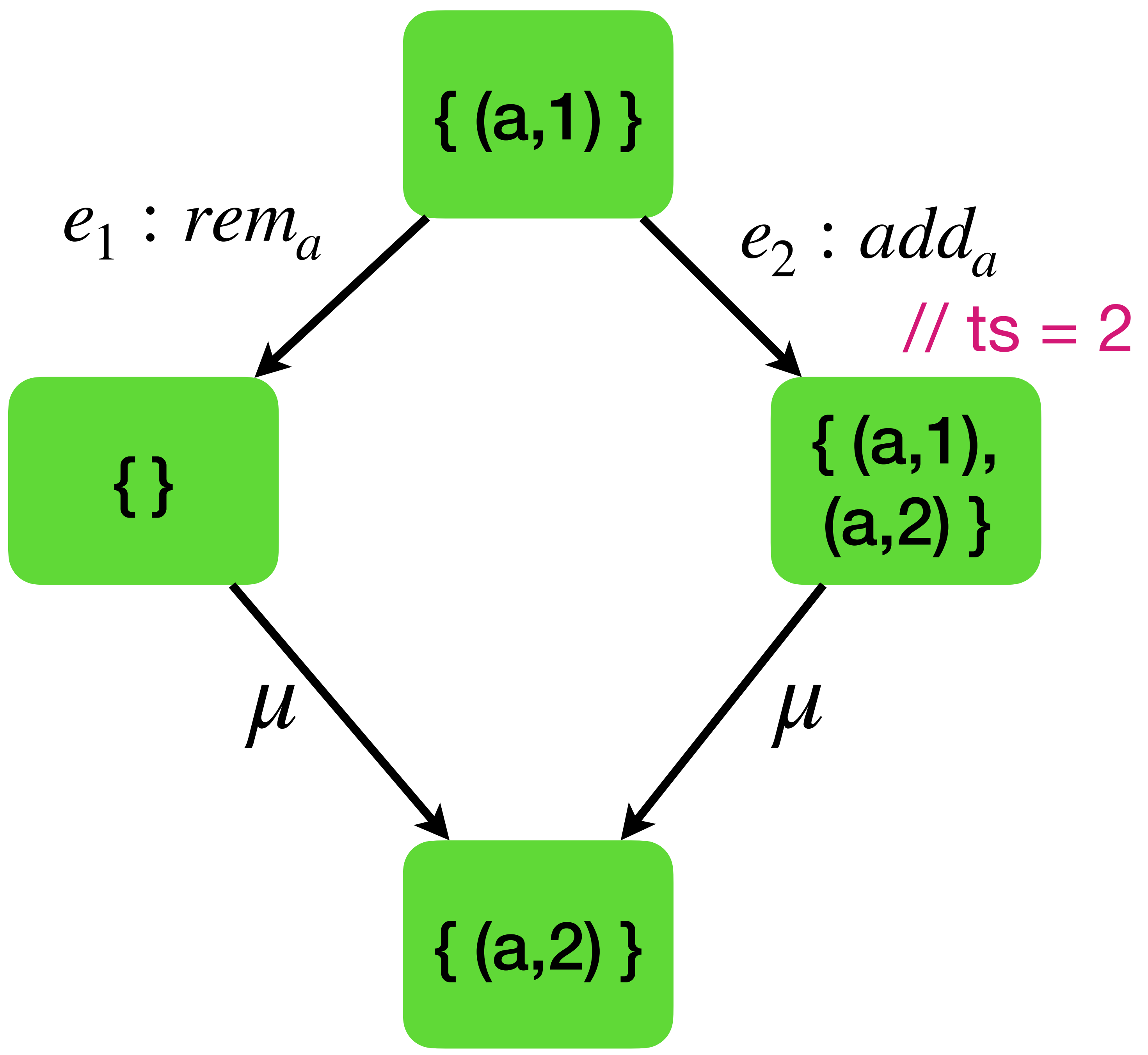
State	$1: \ \Sigma = \mathbb{N}$
Updates	$2: \ O = \{\mathrm{inc}\}$
Queries	$3: \ Q = \{\mathrm{rd}\}$
Init State	$4: \ \sigma_0 = 0$
Update behaviour	$5: \ \mathrm{do}(\sigma, \_, \_, \mathrm{inc}) = \sigma + 1$
Merge	$6: \ \mathrm{merge}(\sigma_\top, \sigma_1, \sigma_2) = \sigma_\top + (\sigma_1 - \sigma_\top) + (\sigma_2 - \sigma_\top)$
Query behaviour	$7: \ \mathrm{query}(\sigma, rd) = \sigma$
Resolve conflict	$8: \ \mathrm{rc} = \emptyset$

Unique timestamp

Replica ID

Add-wins Set

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



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

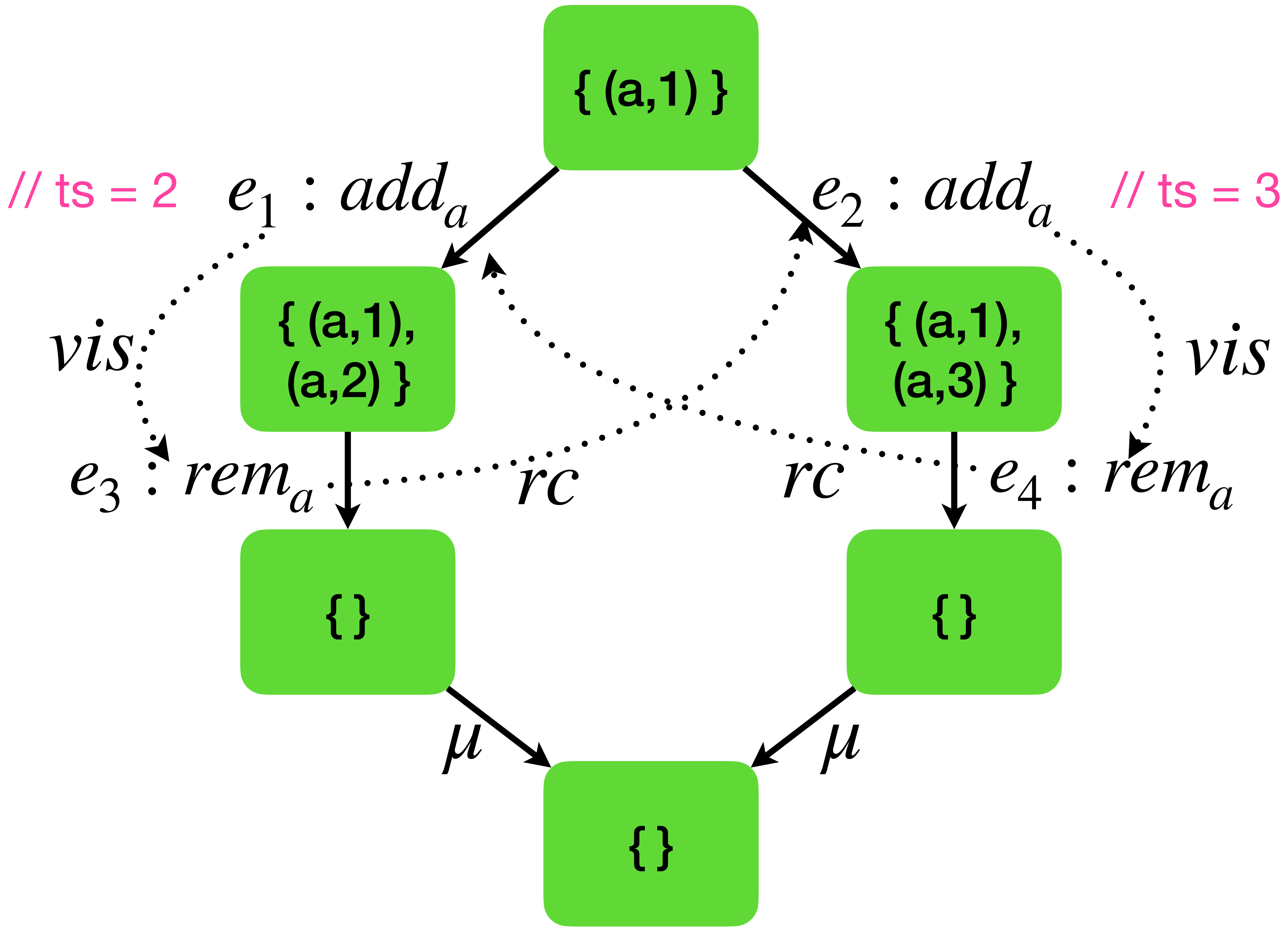
RA-Linearizability Challenge

- Should the linearisation ***total*** order be consistent with
  - Conflict Resolution ordering for concurrent events? $(rc \cap \|) \subseteq lo$
  - And, Visibility? $vis \subseteq lo$

$$e_1 \xrightarrow{vis} e_3$$
$$e3 \parallel e2 \wedge e3 \xrightarrow{rc} e2$$
$$e_2 \xrightarrow{vis} e_4$$
$$e4 \parallel e1 \wedge e4 \xrightarrow{rc} e1$$

lo cannot be total order since $(rc \cap \|) \cup vis$ is not irreflexive

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



RA-Linearizability Challenge

- Should the linearisation ***total*** order be consistent with
  - Conflict Resolution ordering for concurrent events? $(rc \cap \|) \subseteq lo$
  - And, Visibility? $vis \subseteq lo$

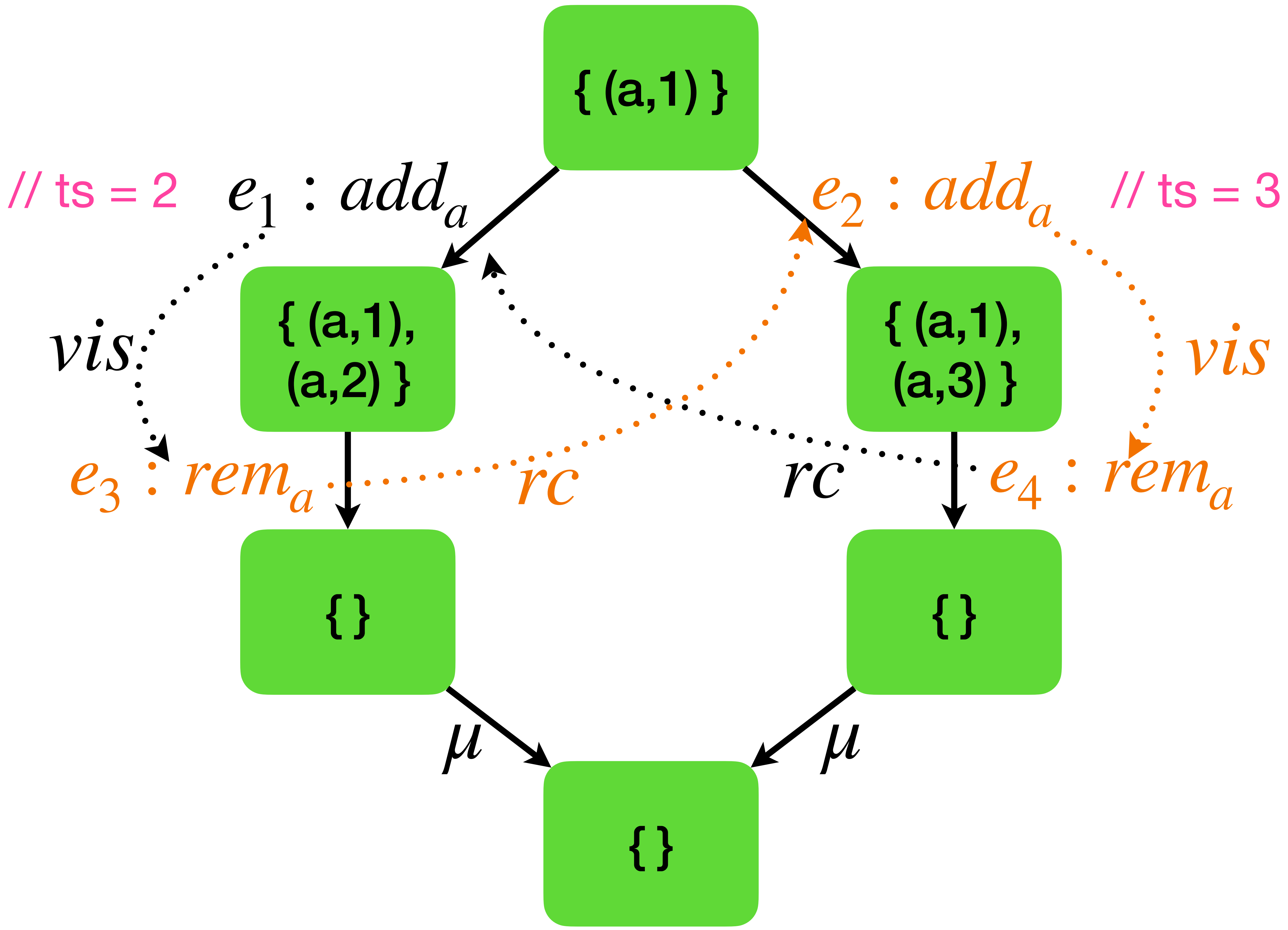
$$e_1 \xrightarrow{vis} e_3$$

$$e_2 \xrightarrow{vis} e_4$$

$$e4 \parallel e1 \wedge e4 \xrightarrow{rc} e1$$

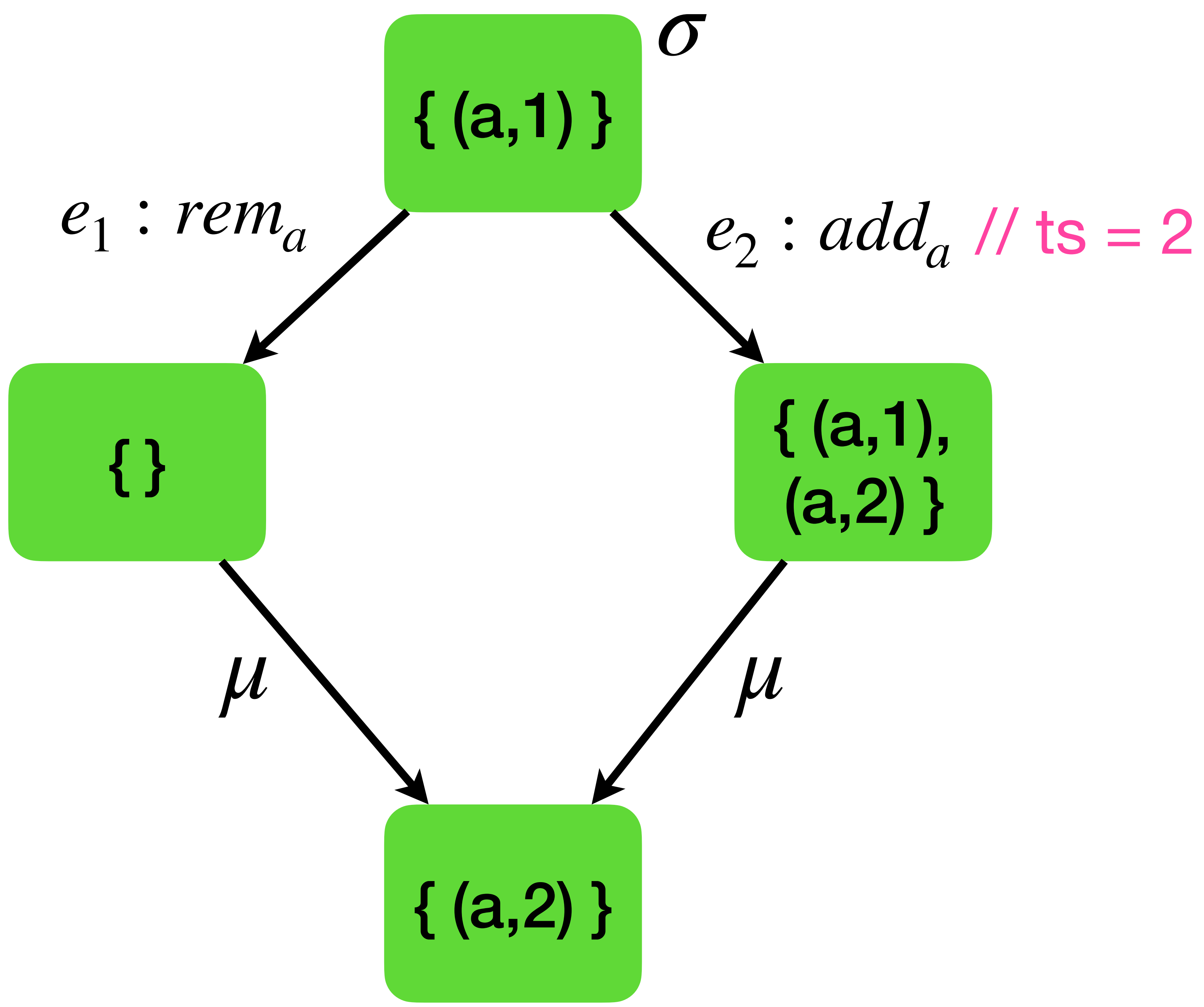
e_3 ***conditionally commutes*** wrt e_2 due to e_4

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



Bottom up linearisation

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



To show

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

[BottomUp-2-OP]

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

[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]

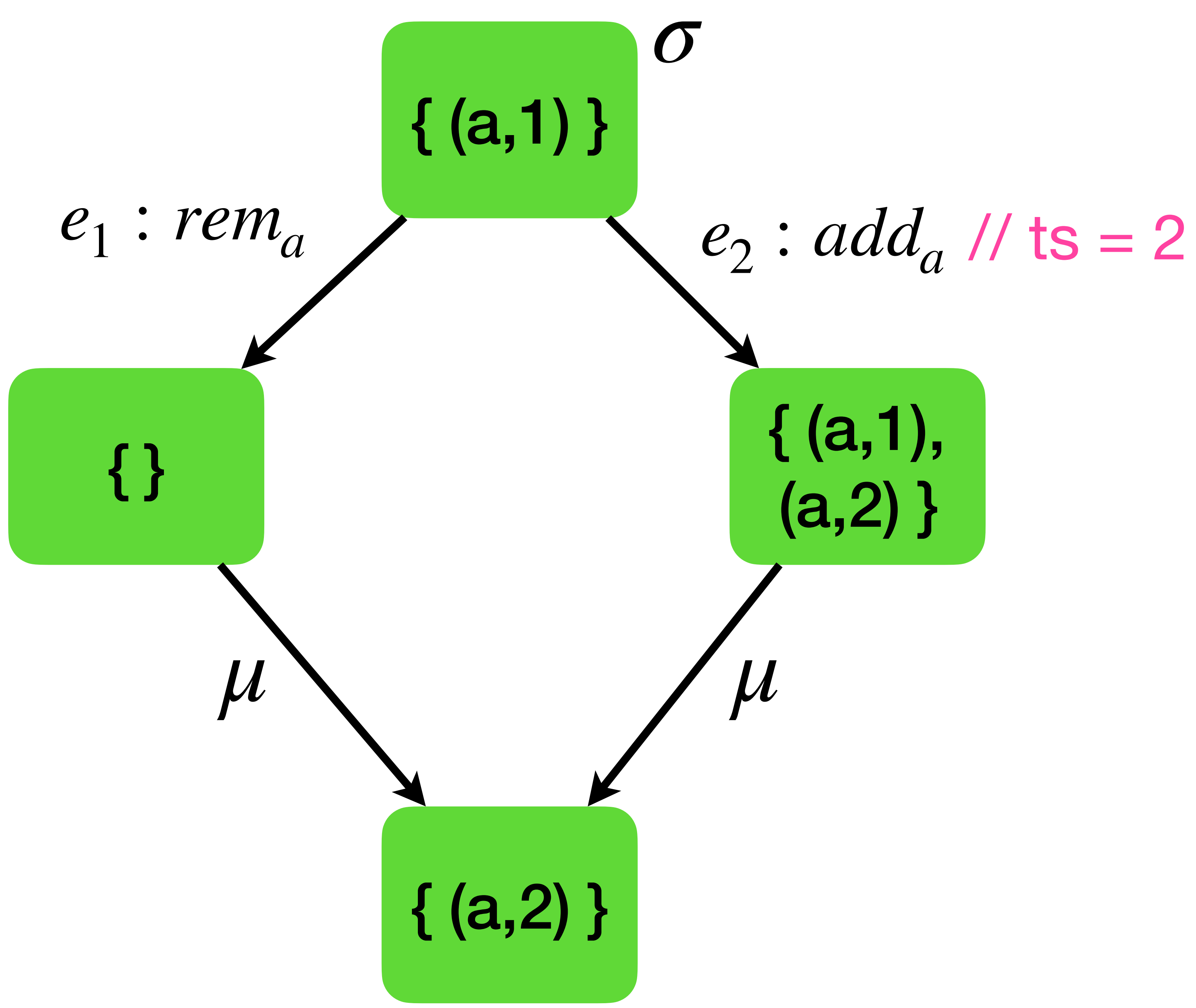
$$\mu(a, a, a) = a$$

[MergeCommutativity]

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

Bottom up linearisation

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



To show

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

[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]

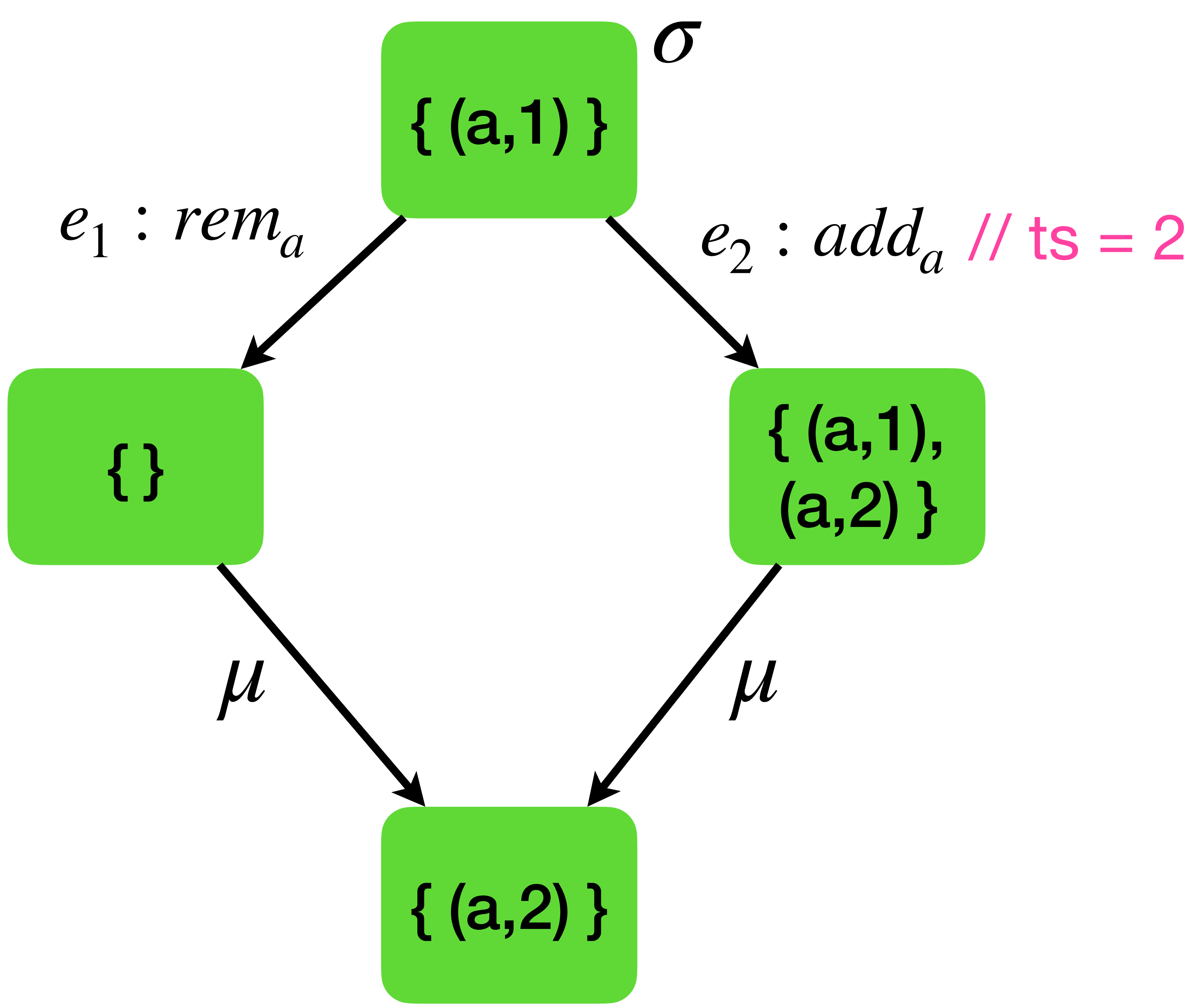
$$\mu(a, a, a) = a$$

[MergeCommutativity]

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

Bottom up linearisation

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



To show

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

[BottomUp-2-OP]

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

[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]

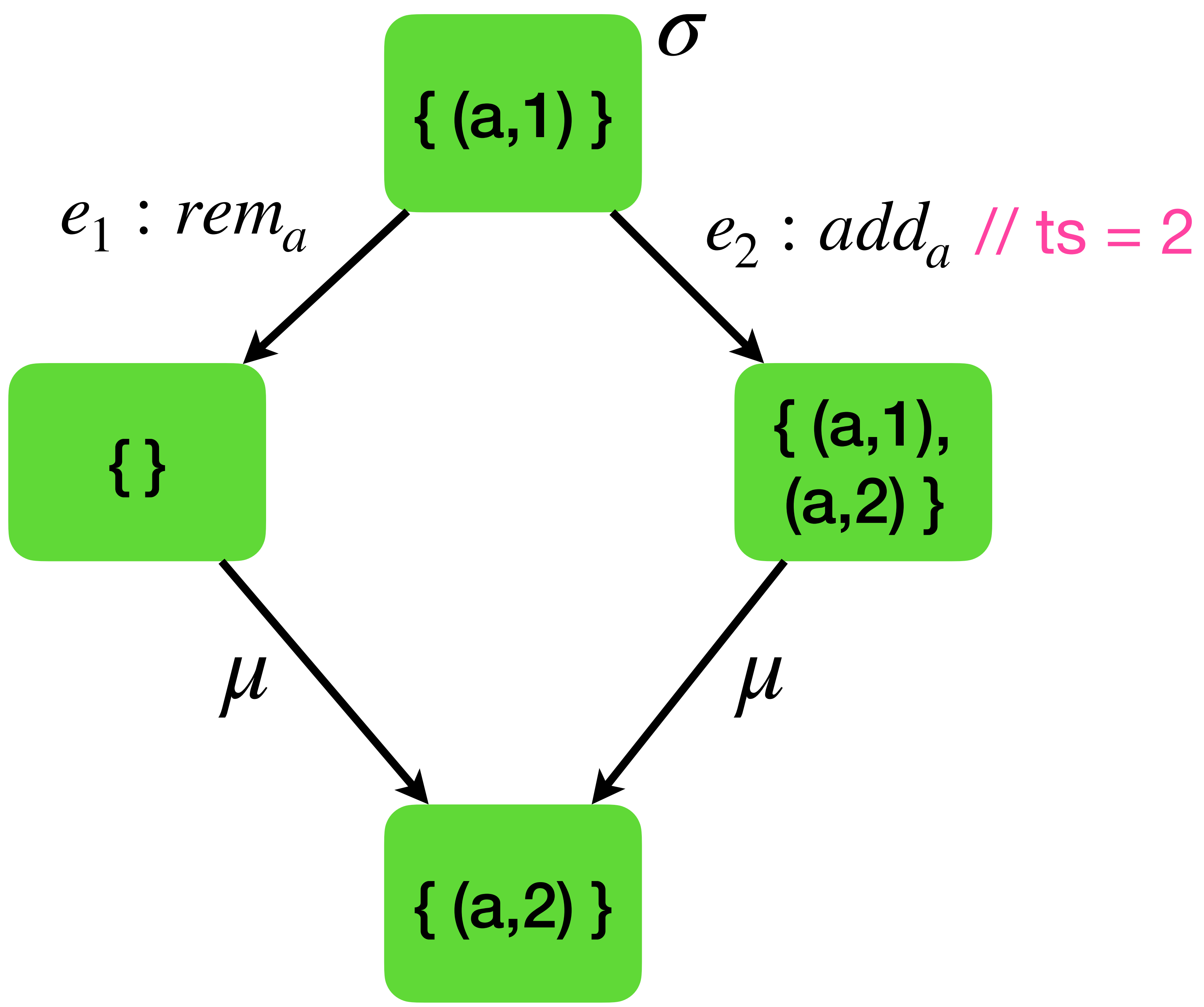
$$\mu(a, a, a) = a$$

[MergeCommutativity]

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

Bottom up linearisation

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



To show

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

[BottomUp-2-OP]

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

[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]

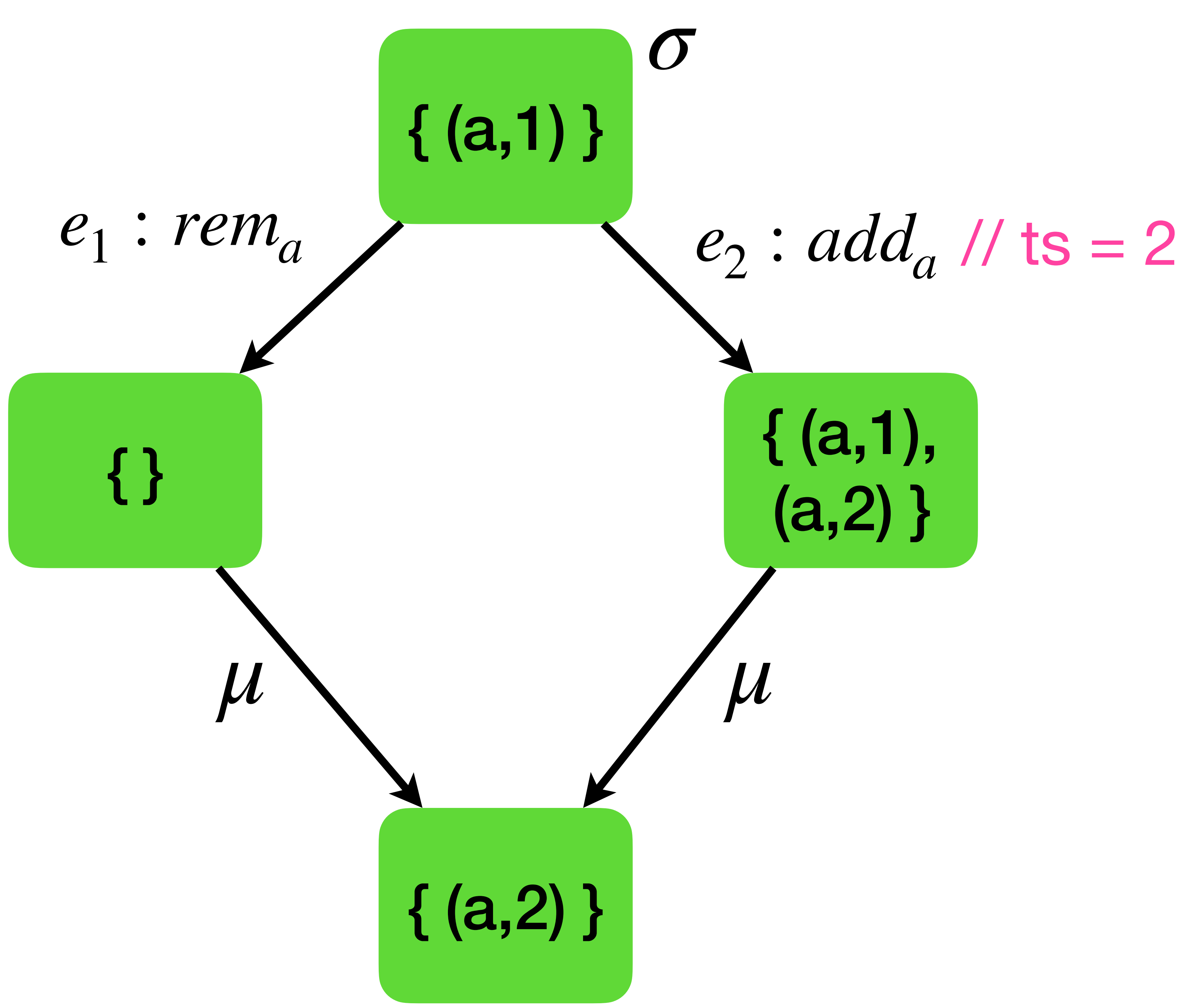
$$\mu(a, a, a) = a$$

[MergeCommutativity]

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

Bottom up linearisation

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



To show

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

[BottomUp-2-OP]

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

[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]

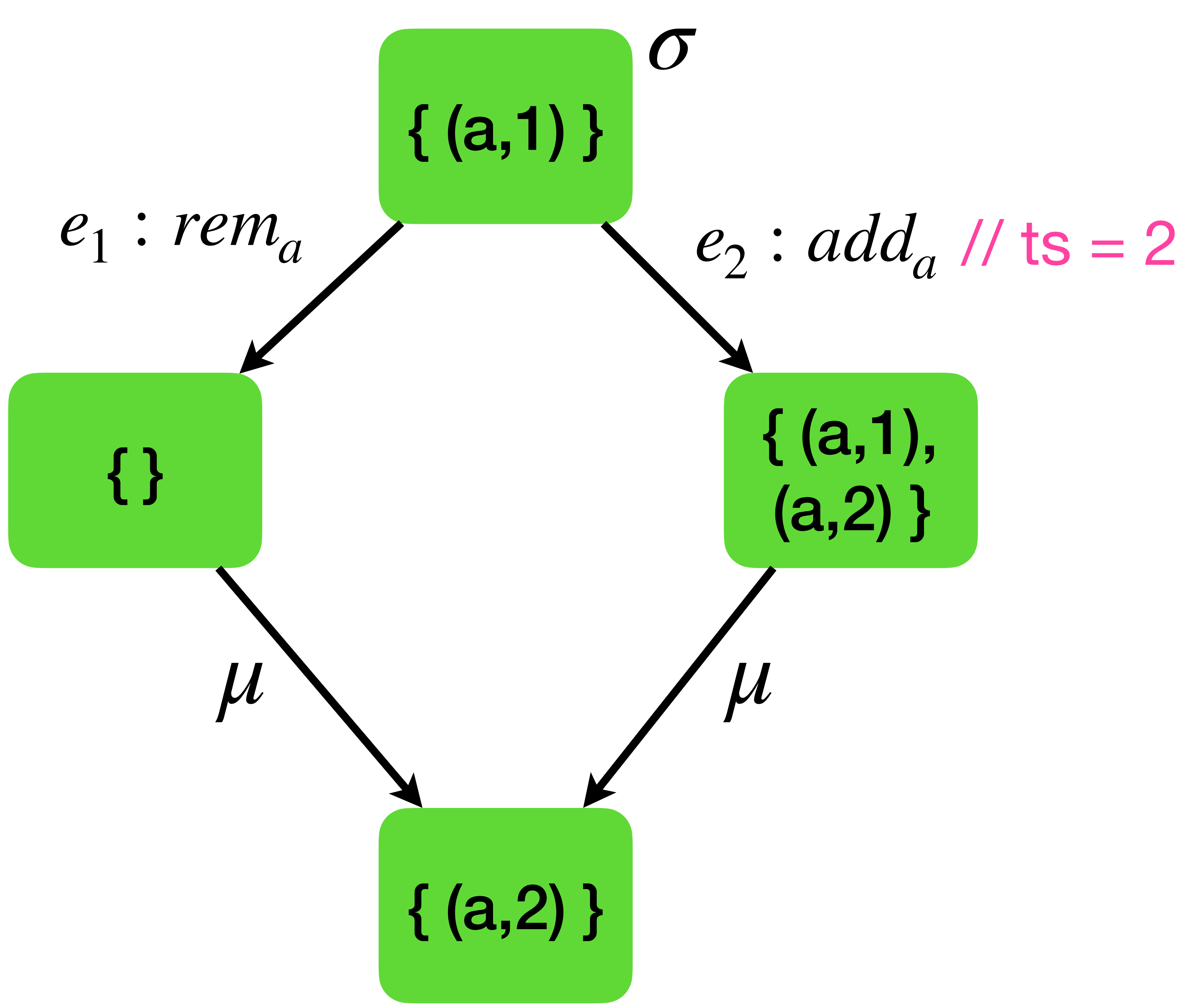
$$\mu(a, a, a) = a$$

[MergeCommutativity]

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

Bottom up linearisation

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



To show

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

[BottomUp-2-OP]

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

[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]

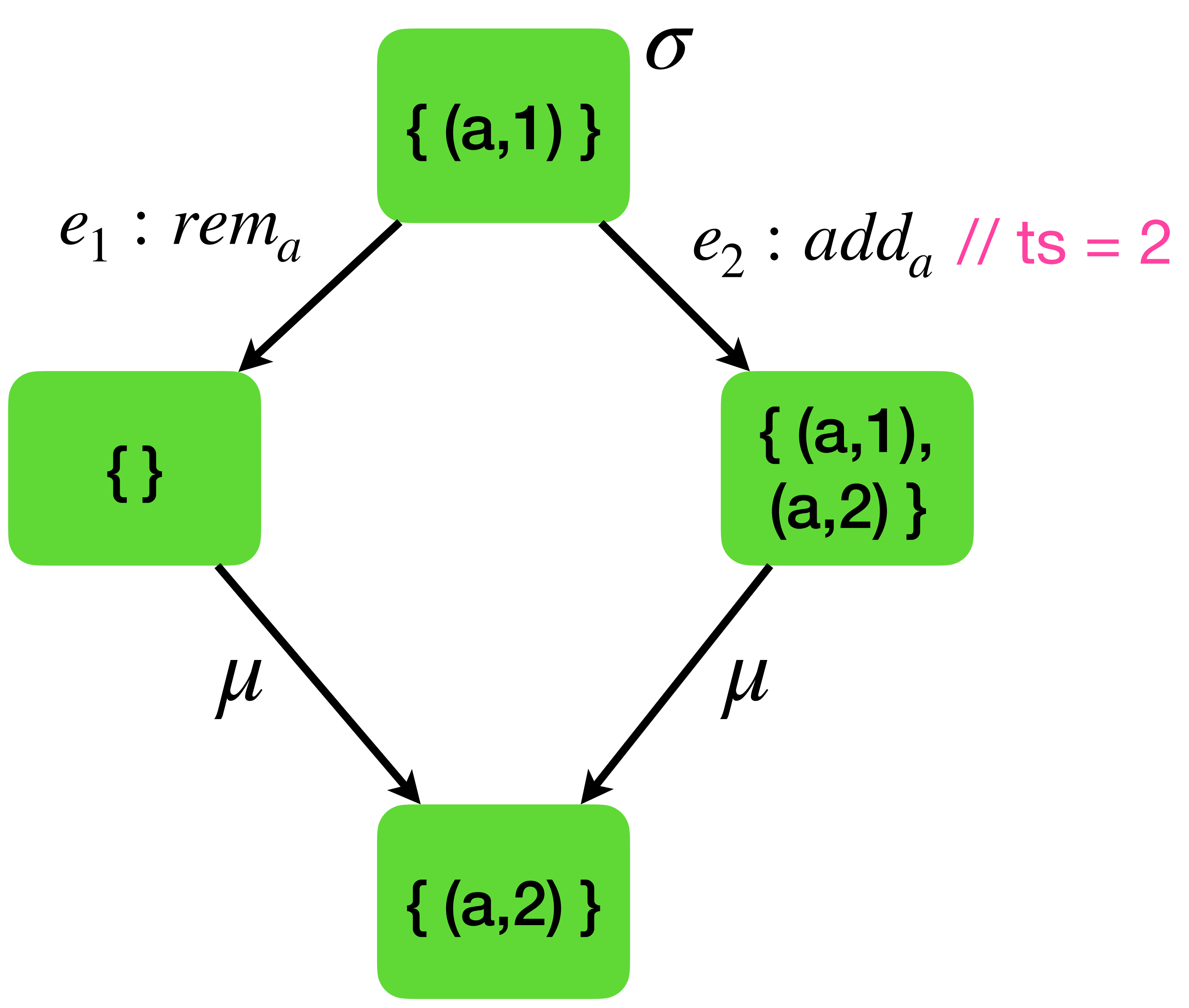
$$\mu(a, a, a) = a$$

[MergeCommutativity]

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

Bottom up linearisation

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



To show

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

[BottomUp-2-OP]

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

[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]

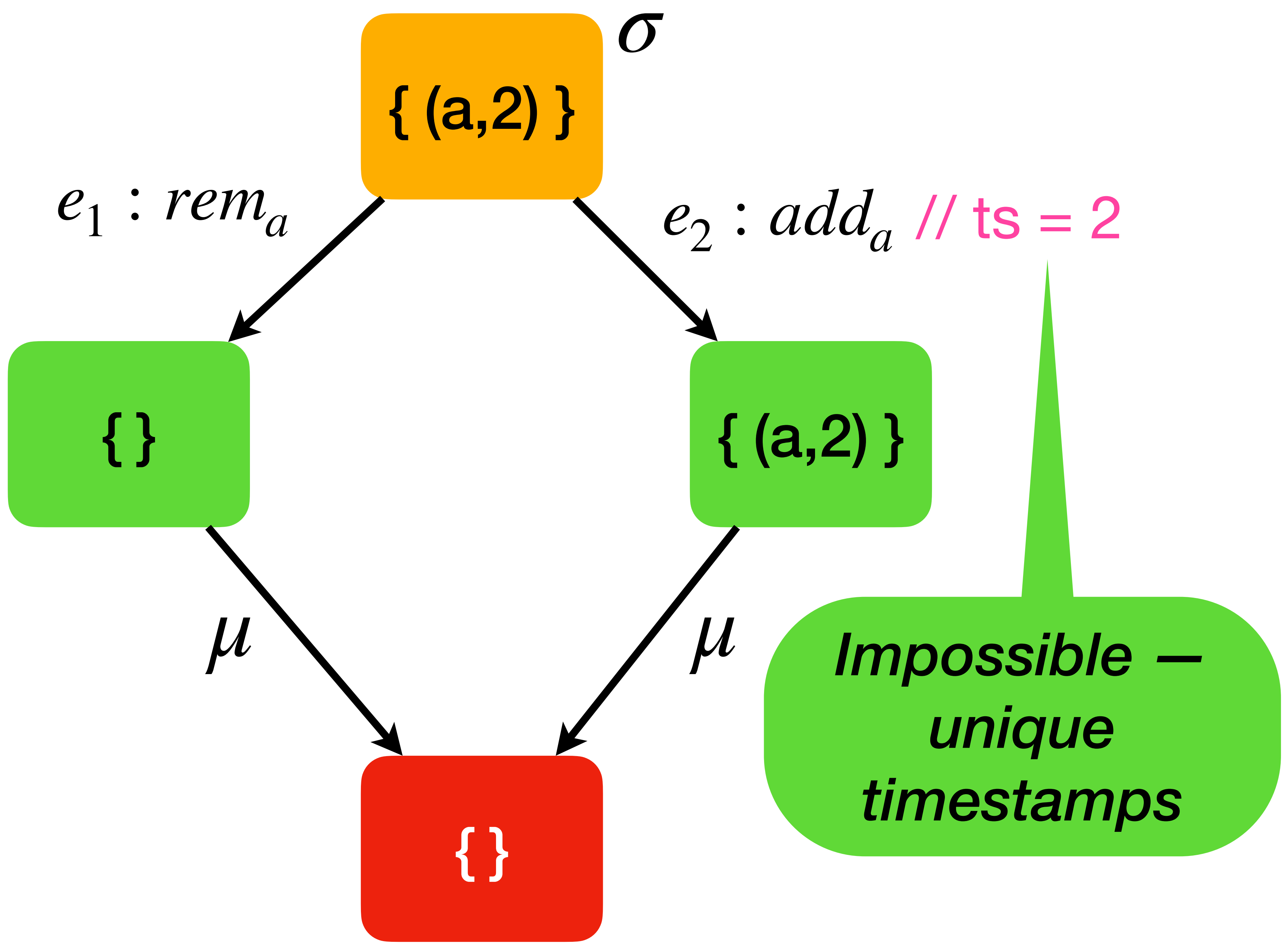
$$\mu(a, a, a) = a$$

[MergeCommutativity]

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

Making a good VC

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



To show

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

[BottomUp-2-OP]

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

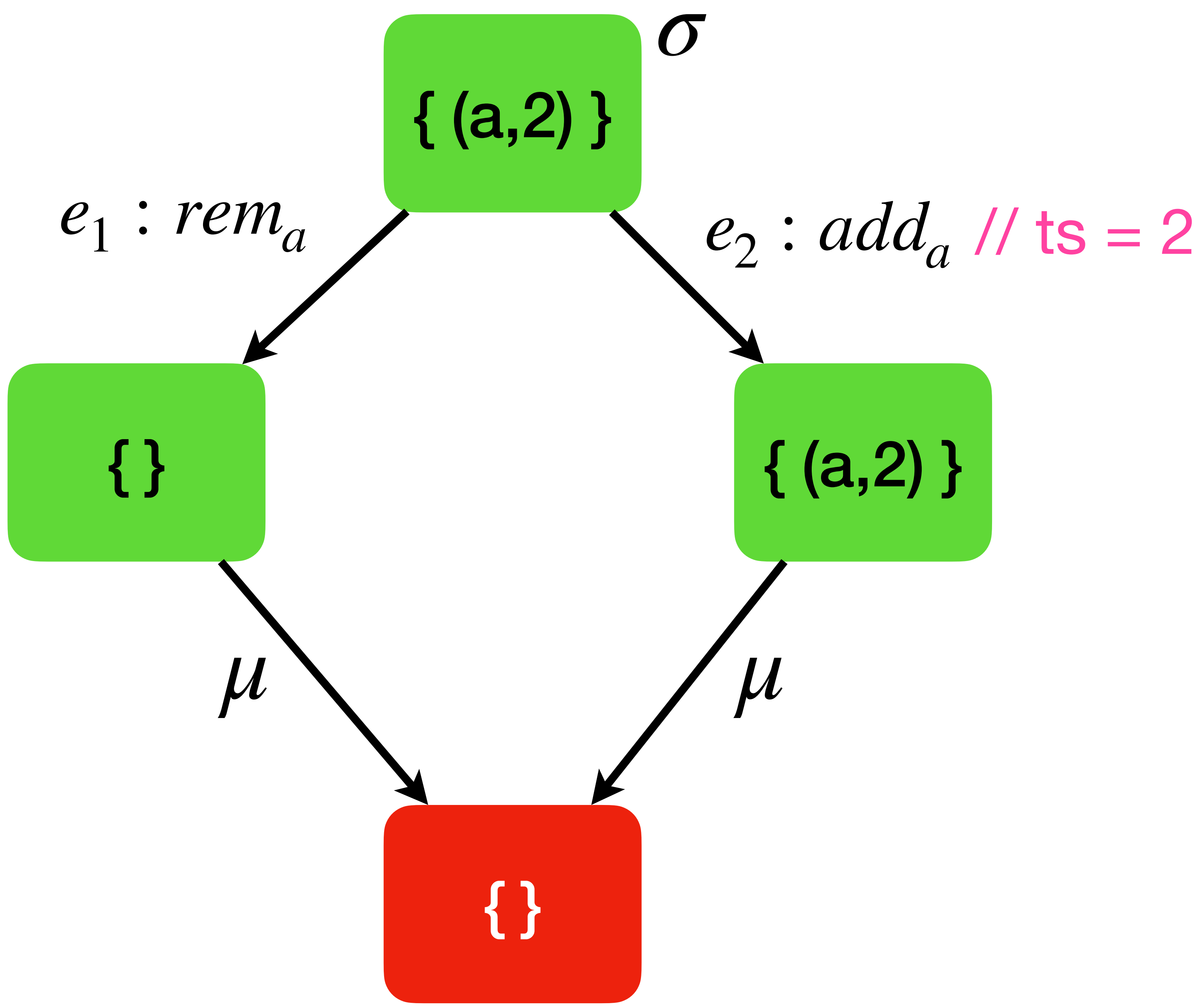
μ and e_2 may not commute in general

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

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

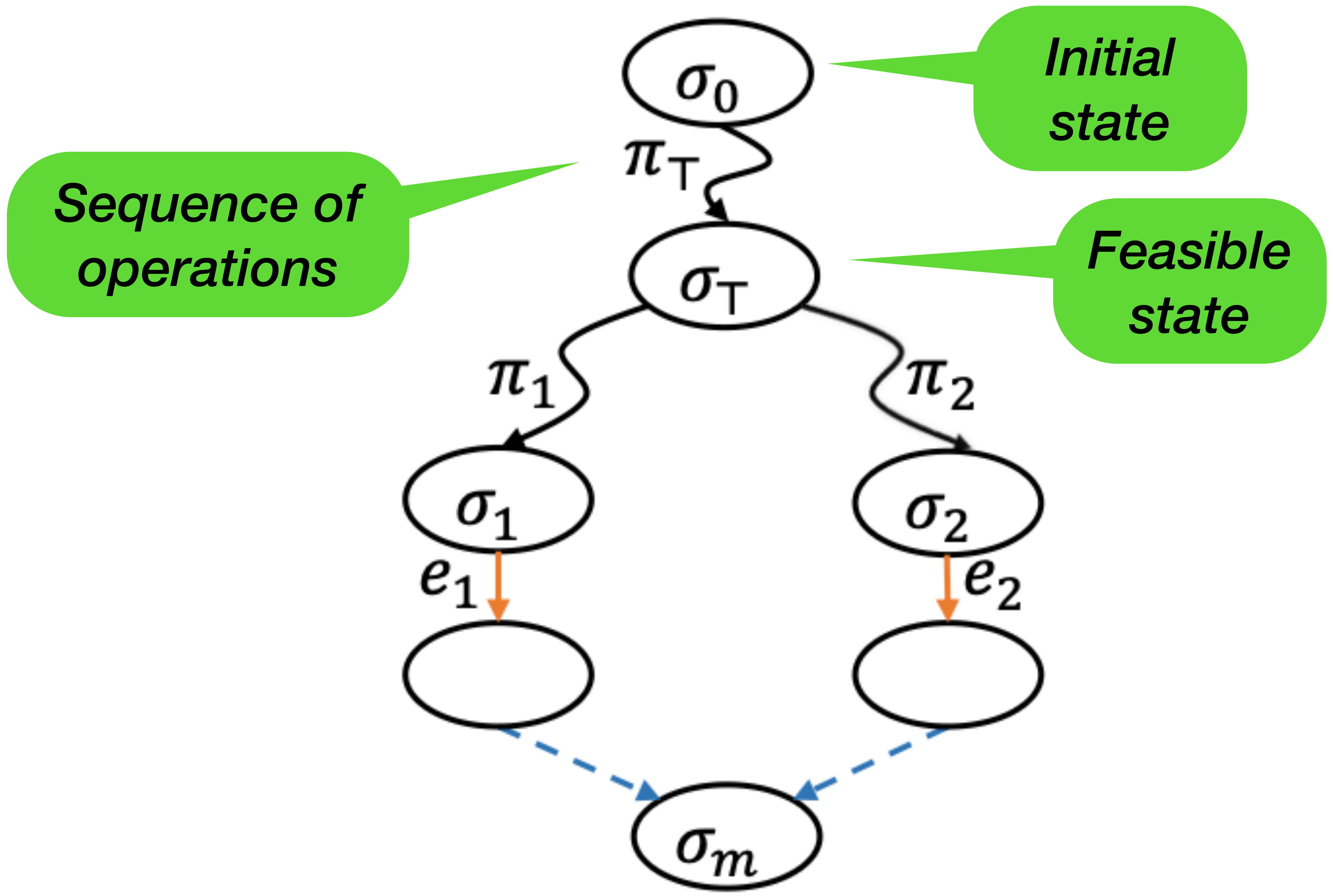


To show

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

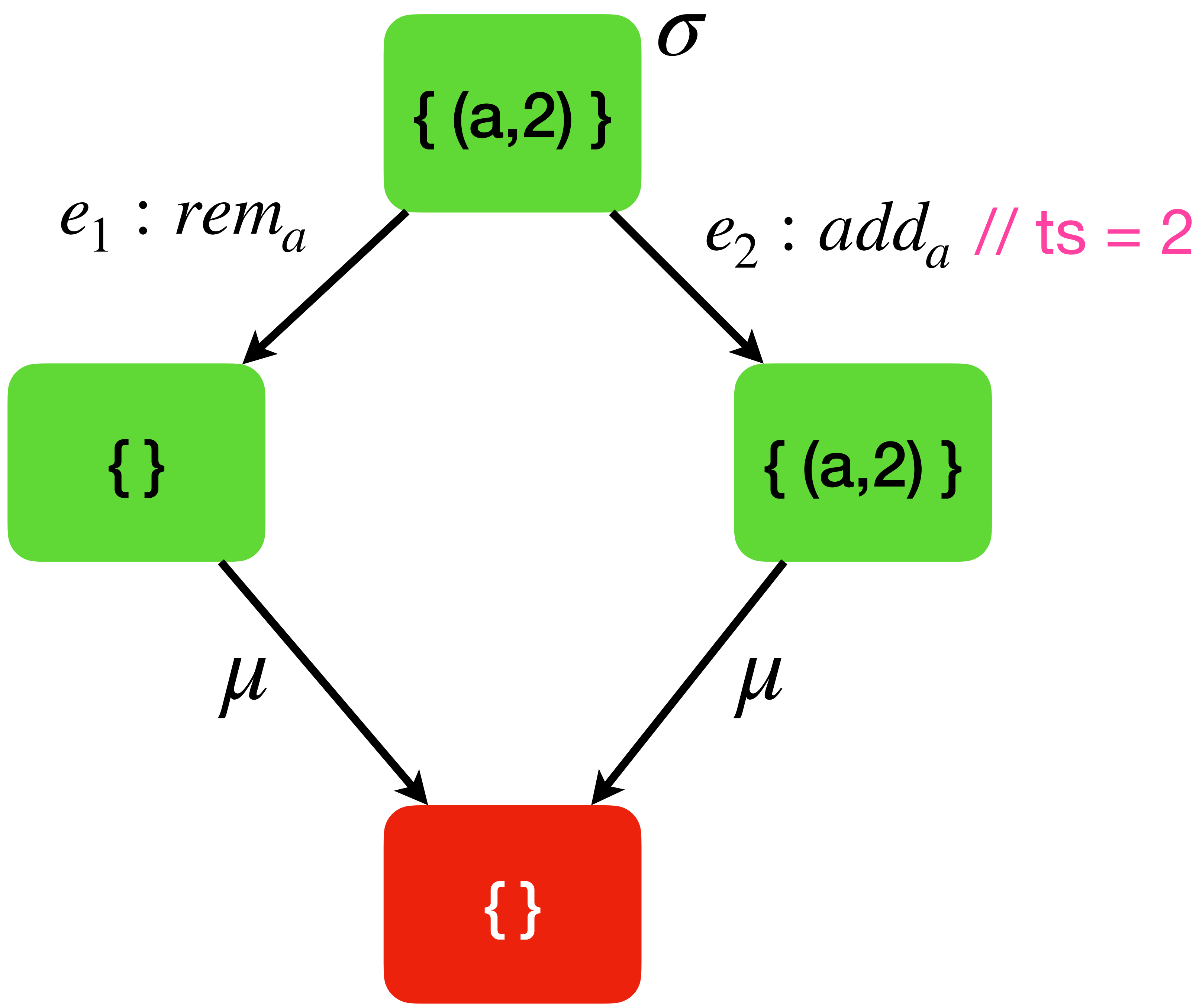
[BottomUp-2-OP]

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



Induction over event sequences

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



To show

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

[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 π_{T}

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

Base case

$(a,2) \notin \sigma_0$

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

Inductive case

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*	$\mathsf{disable} \xrightarrow{\mathsf{rc}} \mathsf{enable}$	30	29.80
Disable-wins flag*	$\mathsf{enable} \xrightarrow{\mathsf{rc}} \mathsf{disable}$	30	37.91
Grows-only set [12]	none	6	0.45
Grows-only map [23]	none	11	4.65
OR-set [23]	$\mathsf{rem}_a \xrightarrow{\mathsf{rc}} \mathsf{add}_a$	20	4.53
OR-set (efficient)*	$\mathsf{rem}_a \xrightarrow{\mathsf{rc}} \mathsf{add}_a$	34	660.00
Remove-wins set*	$\mathsf{add}_a \xrightarrow{\mathsf{rc}} \mathsf{rem}_a$	22	9.60
Set-wins map*	$\mathsf{del}_k \xrightarrow{\mathsf{rc}} \mathsf{set}_k$	20	5.06
Replicated Growable Array [1]	none	13	1.51
Optional register*	$\mathsf{unset} \xrightarrow{\mathsf{rc}} \mathsf{set}$	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

Future work (we could do better)

- Automated verification returns yes / no / ¯_(ツ)_/¯
  - Not pleasant for engineering
  - No counterexamples!
- Current work: model checking MRDTs against RA-linearizability
  - Fixed inputs & unrestricted concurrency
  - ***QuickCheck?***