Certified Mergeable Replicated Data Types

"KC" Sivaramakrishnan

joint work with

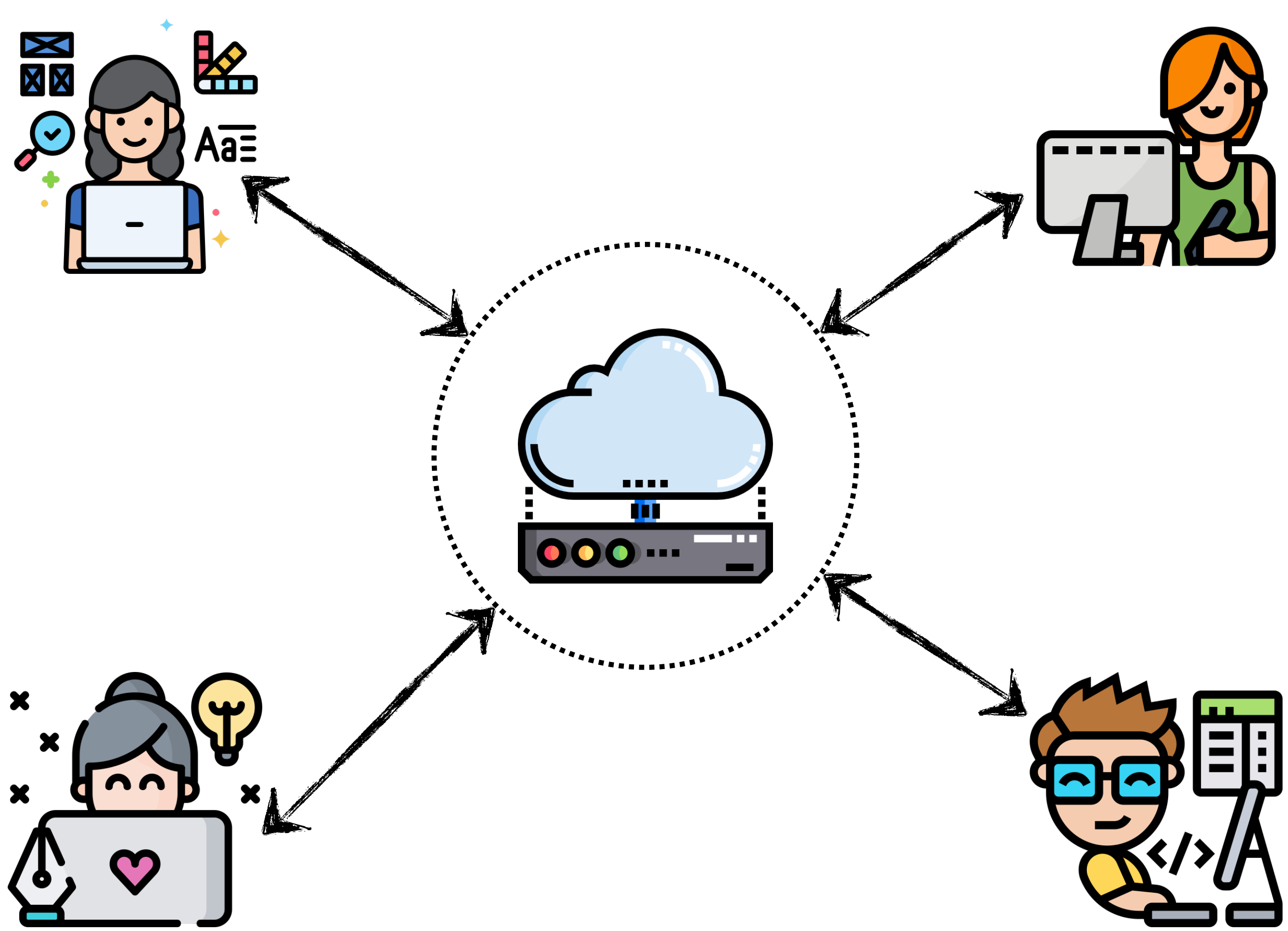
Vimala Soundarapandian, Adharsh Kamath and Kartik Nagar

Collaborative Apps

Airtable Google Docs Figma Notion Overleaf

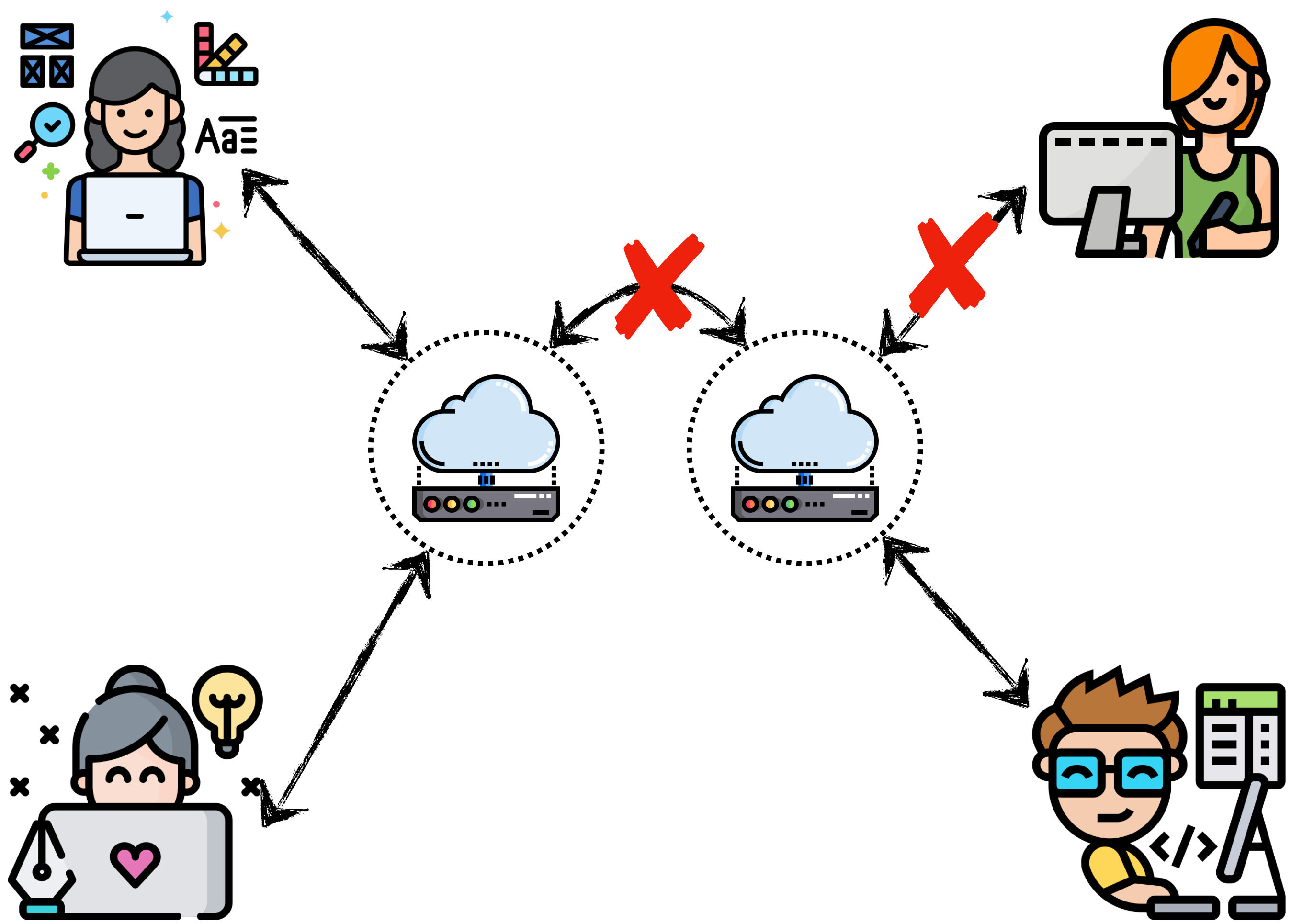
Collaborative Apps

Airtable Google Docs Figma Notion Overleaf



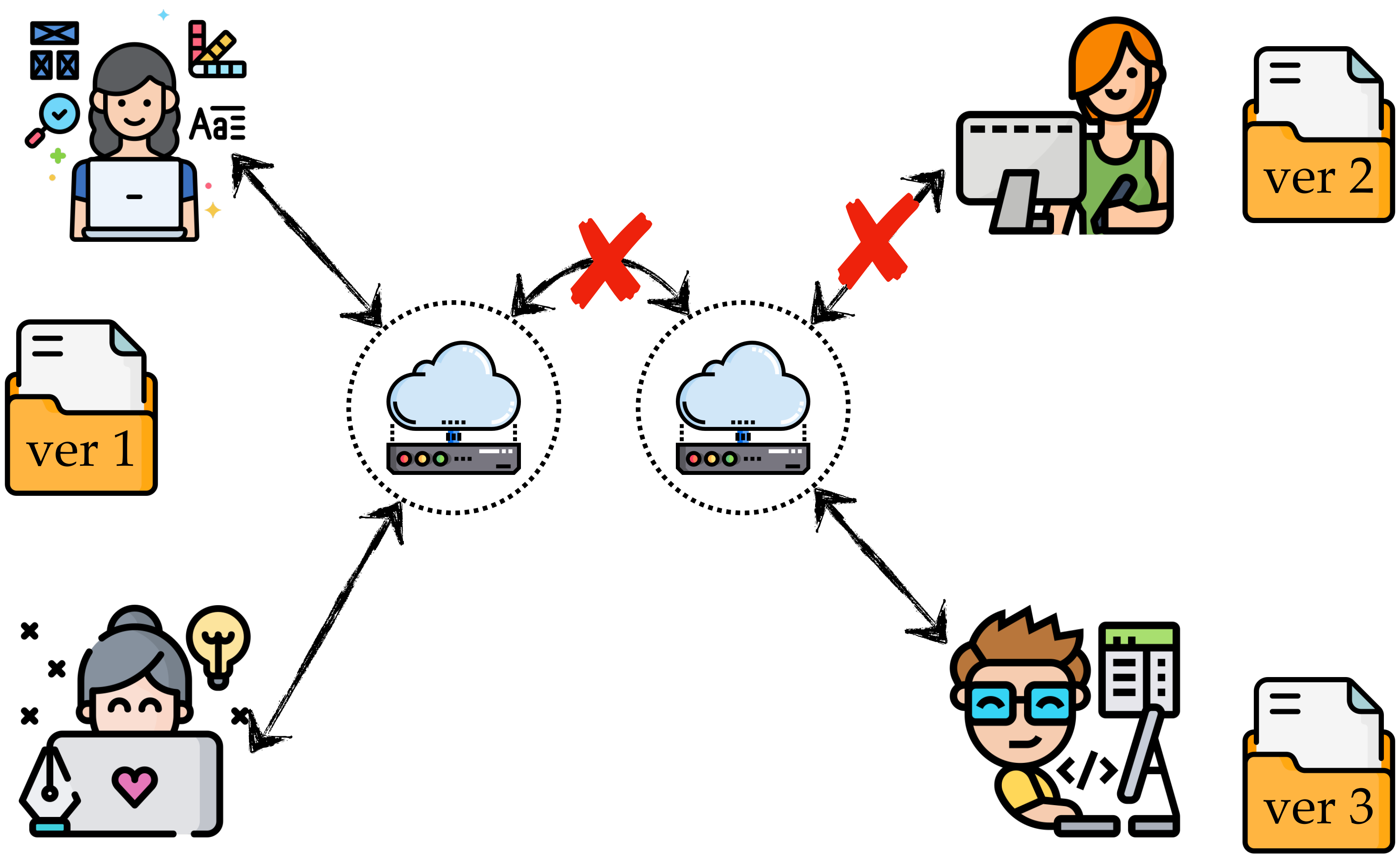
Network Partitions

Airtable Google Docs Figma Notion Overleaf



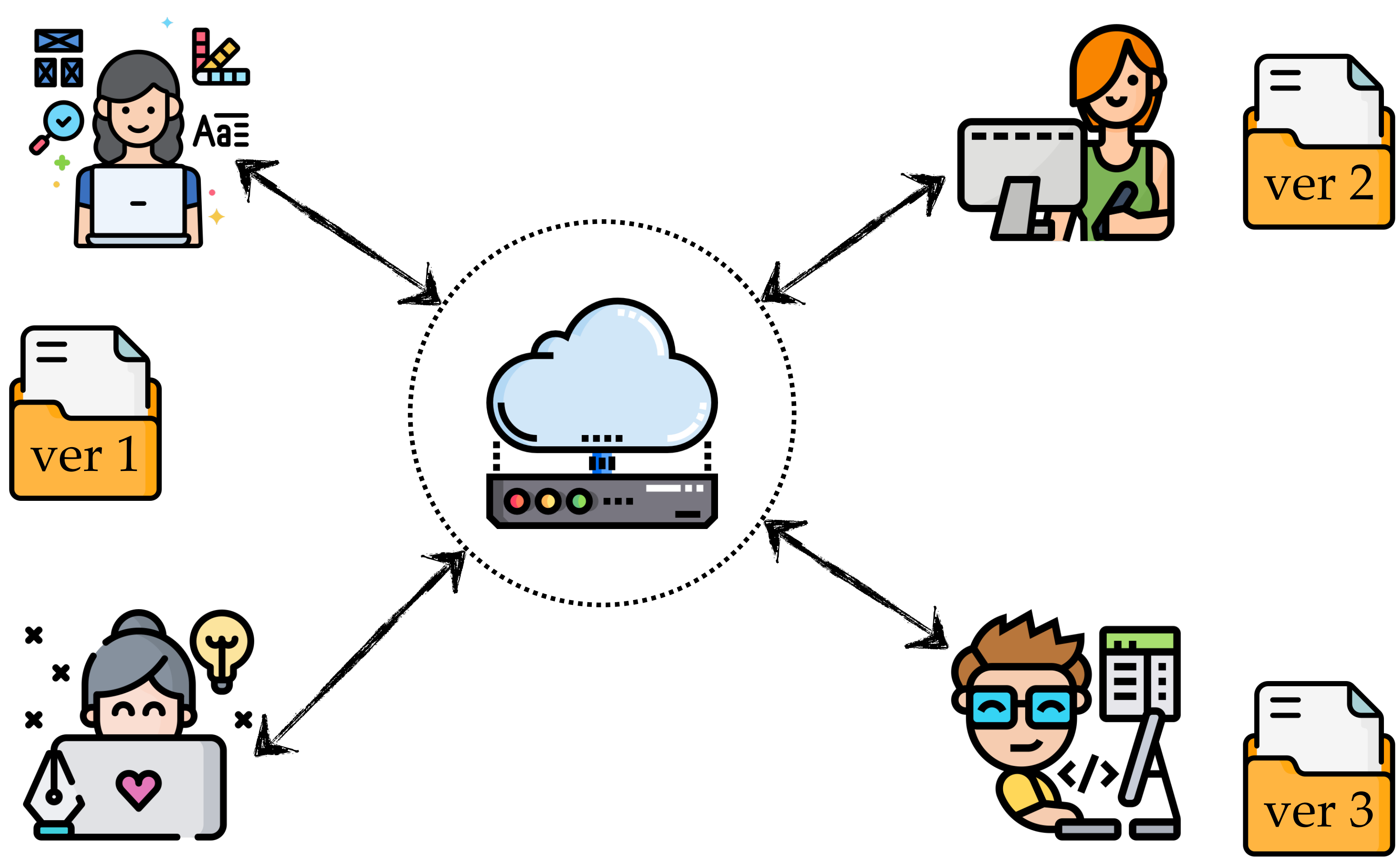
Local-first software

Airtable Google Docs Figma Notion Overleaf



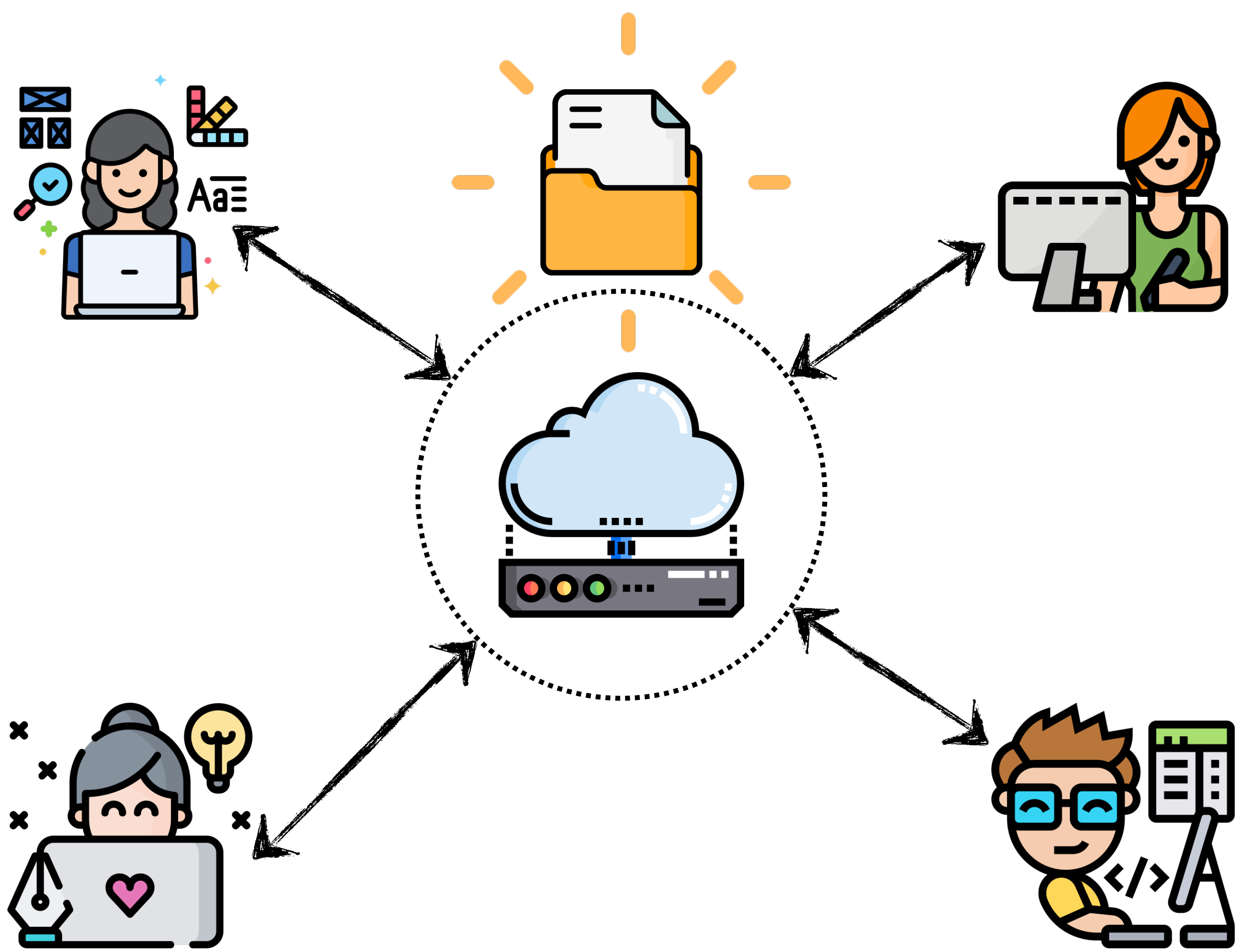
Local-first software

Airtable Google Docs Figma Notion Overleaf

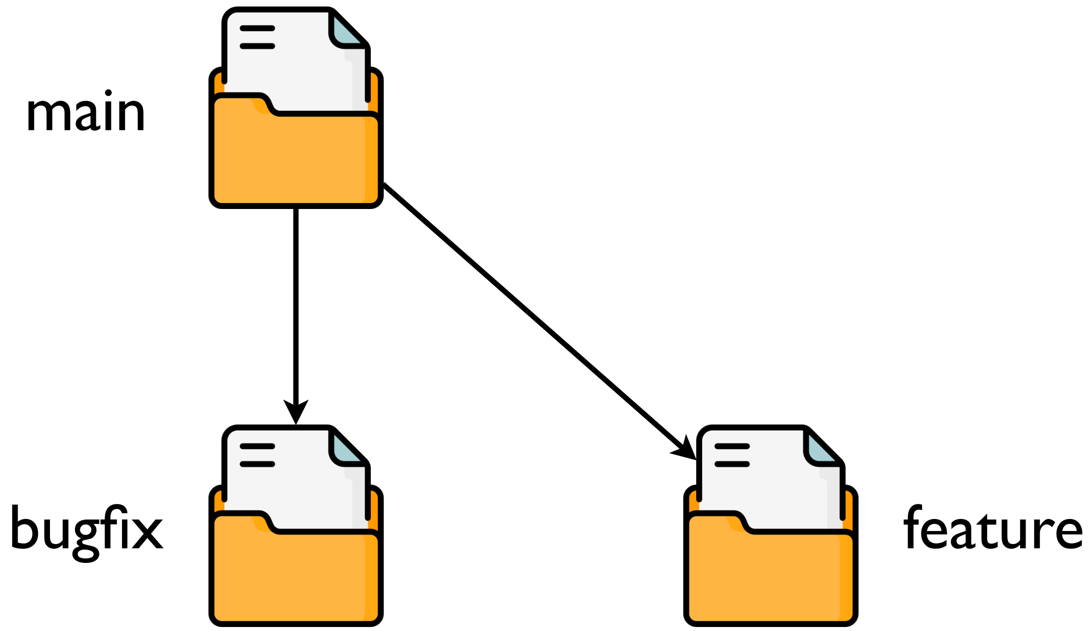


Local-first software

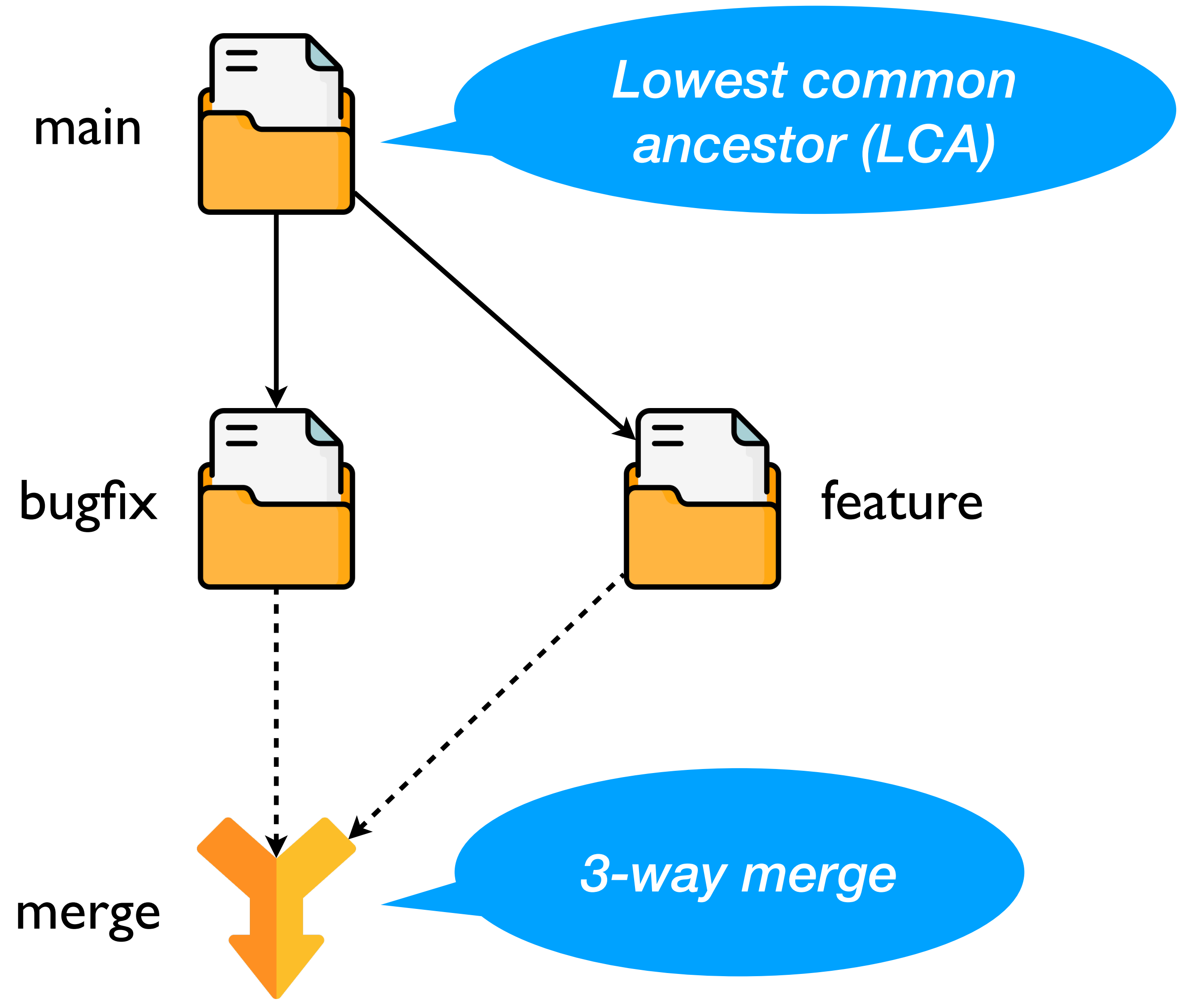
Airtable Google Docs Figma Notion Overleaf



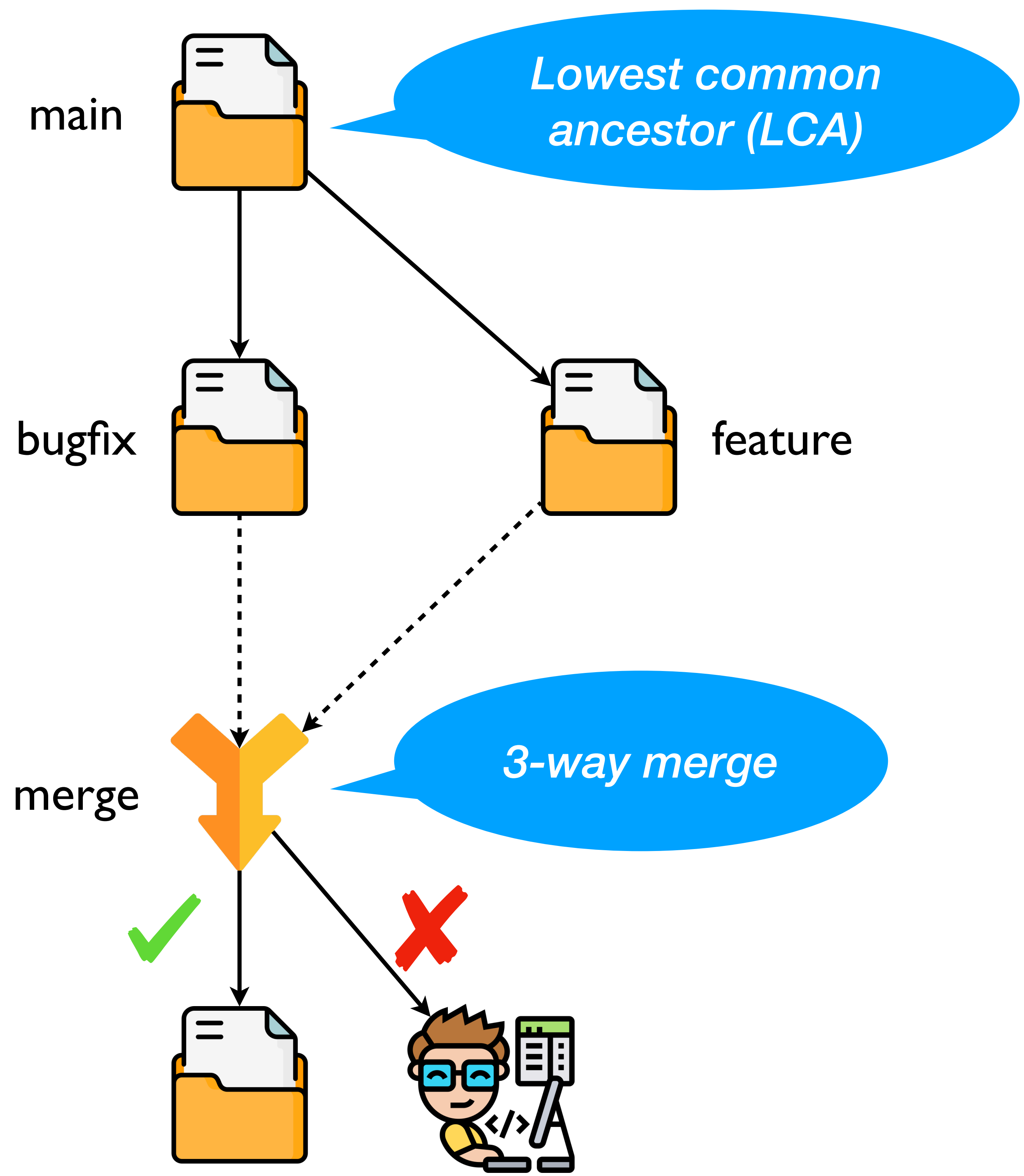
Distributed Version Control Systems



Distributed Version Control Systems



Distributed Version Control Systems



Mergeable Replicated Data Types

- MRDTs — DVCS for *data types* rather than just *text files*
- Sequential data types + 3-way merge = replicated data type!

Mergeable Replicated Data Types

- MRDTs — DVCS for *data types* rather than just *text files*
- Sequential data types + 3-way merge = replicated data type!

```
module Counter : sig
  type t
  val read : t -> int
  val add  : t -> int -> t
  val mult : t -> int -> t
  val merge : lca:t -> v1:t -> v2:t -> t
end = struct
  type t = int
  let read x = x
  let add x d = x + d
  let mult x n = x * n
  let merge ~lca ~v1 ~v2 =
    lca + (v1 - lca) + (v2 - lca)
end
```

Mergeable Replicated Data Types

- MRDTs — DVCS for *data types* rather than just *text files*
- Sequential data types + 3-way merge = replicated data type!

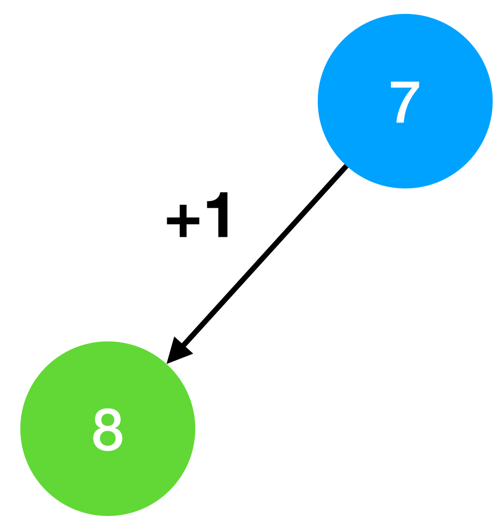
```
module Counter : sig
  type t
  val read : t -> int
  val add  : t -> int -> t
  val mult : t -> int -> t
  val merge : lca:t -> v1:t -> v2:t -> t
end = struct
  type t = int
  let read x = x
  let add x d = x + d
  let mult x n = x * n
  let merge ~lca ~v1 ~v2 =
    lca + (v1 - lca) + (v2 - lca)
end
```

7

Mergeable Replicated Data Types

- MRDTs — DVCS for *data types* rather than just *text files*
- Sequential data types + 3-way merge = replicated data type!

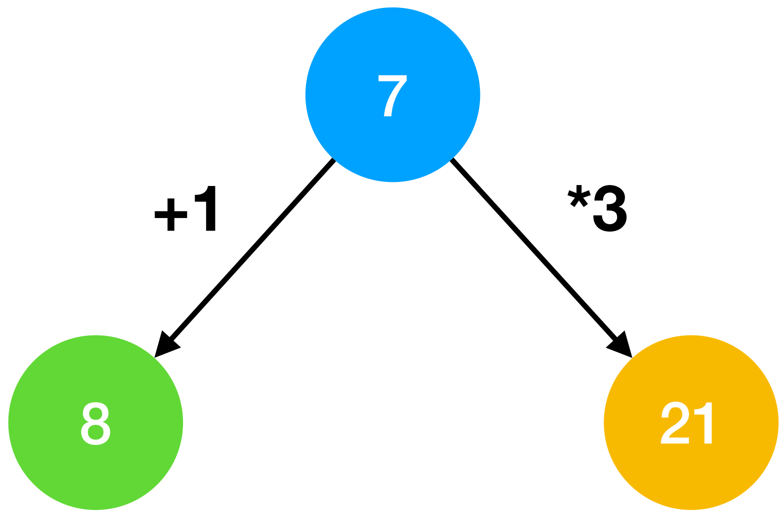
```
module Counter : sig
  type t
  val read : t -> int
  val add  : t -> int -> t
  val mult : t -> int -> t
  val merge : lca:t -> v1:t -> v2:t -> t
end = struct
  type t = int
  let read x = x
  let add x d = x + d
  let mult x n = x * n
  let merge ~lca ~v1 ~v2 =
    lca + (v1 - lca) + (v2 - lca)
end
```



Mergeable Replicated Data Types

- MRDTs — DVCS for *data types* rather than just *text files*
- Sequential data types + 3-way merge = replicated data type!

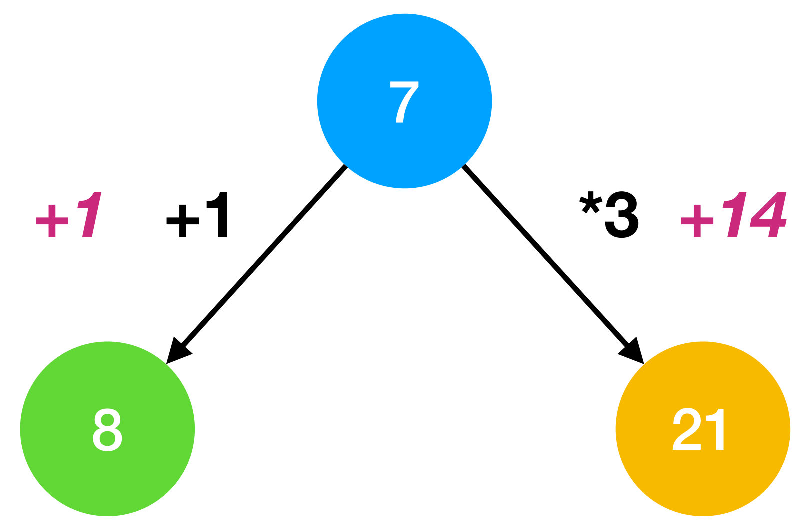
```
module Counter : sig
  type t
  val read : t -> int
  val add  : t -> int -> t
  val mult : t -> int -> t
  val merge : lca:t -> v1:t -> v2:t -> t
end = struct
  type t = int
  let read x = x
  let add x d = x + d
  let mult x n = x * n
  let merge ~lca ~v1 ~v2 =
    lca + (v1 - lca) + (v2 - lca)
end
```



Mergeable Replicated Data Types

- MRDTs — DVCS for *data types* rather than just *text files*
- Sequential data types + 3-way merge = replicated data type!

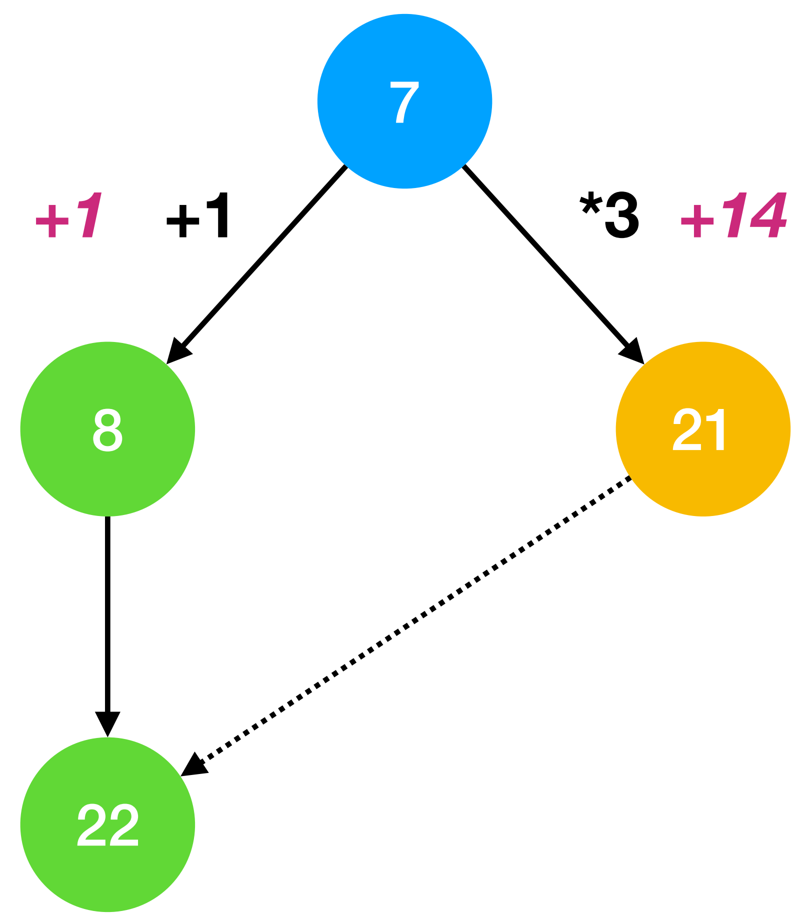
```
module Counter : sig
  type t
  val read : t -> int
  val add  : t -> int -> t
  val mult : t -> int -> t
  val merge : lca:t -> v1:t -> v2:t -> t
end = struct
  type t = int
  let read x = x
  let add x d = x + d
  let mult x n = x * n
  let merge ~lca ~v1 ~v2 =
    lca + (v1 - lca) + (v2 - lca)
end
```



Mergeable Replicated Data Types

- MRDTs — DVCS for *data types* rather than just *text files*
- Sequential data types + 3-way merge = replicated data type!

```
module Counter : sig
  type t
  val read : t -> int
  val add  : t -> int -> t
  val mult : t -> int -> t
  val merge : lca:t -> v1:t -> v2:t -> t
end = struct
  type t = int
  let read x = x
  let add x d = x + d
  let mult x n = x * n
  let merge ~lca ~v1 ~v2 =
    lca + (v1 - lca) + (v2 - lca)
end
```



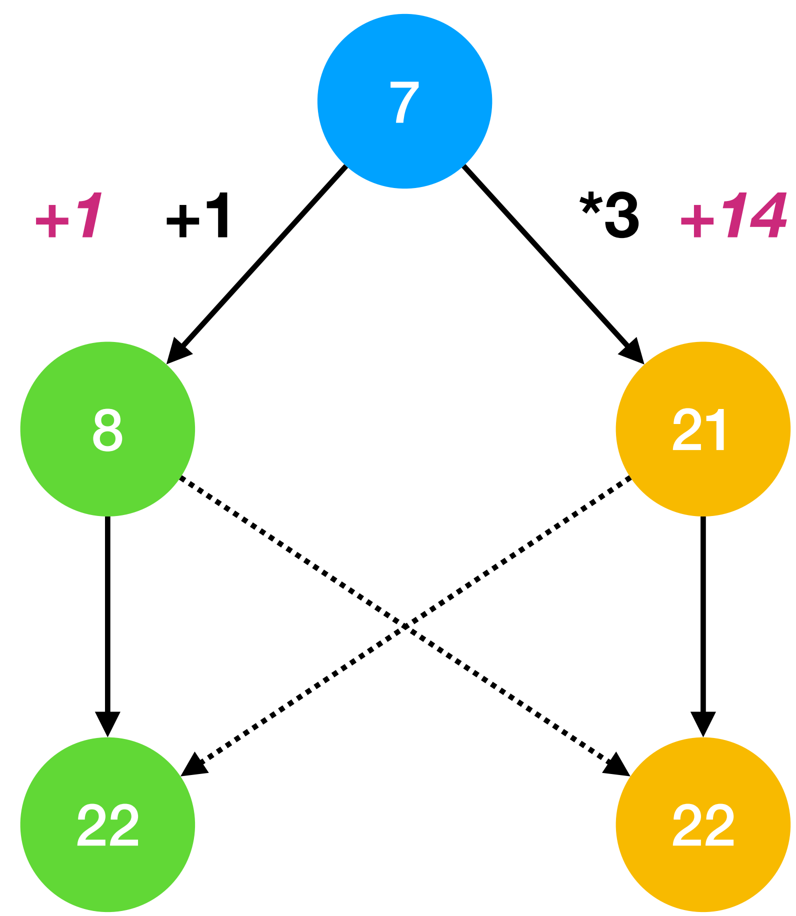
22 = 7 + (8-7) + (21 -7)

+1 +14

Mergeable Replicated Data Types

- MRDTs — DVCS for *data types* rather than just *text files*
- Sequential data types + 3-way merge = replicated data type!

```
module Counter : sig
  type t
  val read : t -> int
  val add  : t -> int -> t
  val mult : t -> int -> t
  val merge : lca:t -> v1:t -> v2:t -> t
end = struct
  type t = int
  let read x = x
  let add x d = x + d
  let mult x n = x * n
  let merge ~lca ~v1 ~v2 =
    lca + (v1 - lca) + (v2 - lca)
end
```



22 = 7 + (8-7) + (21 -7)

+1 +14

Does the 3-way merge idea generalise?

Does the 3-way merge idea generalise?

Sort of…

Observed-Removed Set

- OR-set — *add-wins* when there is a concurrent add and remove of the same element

Observed-Removed Set

- OR-set — *add-wins* when there is a concurrent add and remove of the same element

```
let merge ~lca ~v1 ~v2 =
  (lca ∩ v1 ∩ v2) (* unmodified elements *)
  ∪ (v1 - lca) (* added in v1 *)
  ∪ (v2 - lca) (* added in v2 *)
```

Kaki et al. "Mergeable Replicated Data Types", OOPSLA 2019

Observed-Removed Set

- OR-set — *add-wins* when there is a concurrent add and remove of the same element

```
let merge ~lca ~v1 ~v2 =
  (lca ∩ v1 ∩ v2) (* unmodified elements *)
  ∪ (v1 - lca) (* added in v1 *)
  ∪ (v2 - lca) (* added in v2 *)
```

{1}

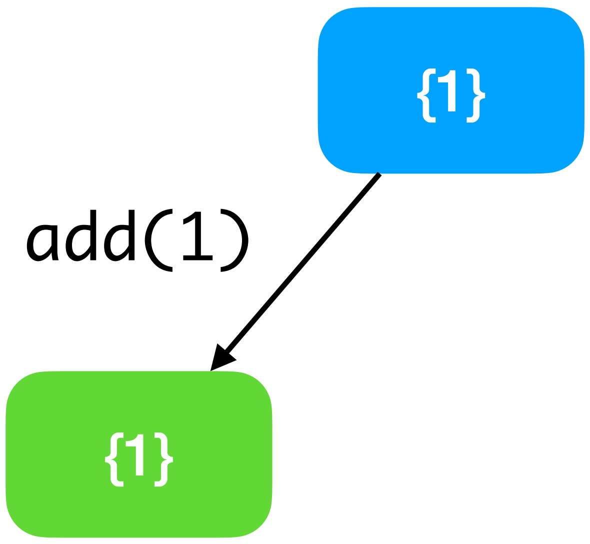
Kaki et al. “Mergeable Replicated Data Types”, OOPSLA 2019

Observed-Removed Set

- OR-set — *add-wins* when there is a concurrent add and remove of the same element

```
let merge ~lca ~v1 ~v2 =
  (lca ∩ v1 ∩ v2) (* unmodified elements *)
  ∪ (v1 - lca) (* added in v1 *)
  ∪ (v2 - lca) (* added in v2 *)
```

Kaki et al. "Mergeable Replicated Data Types", OOPSLA 2019

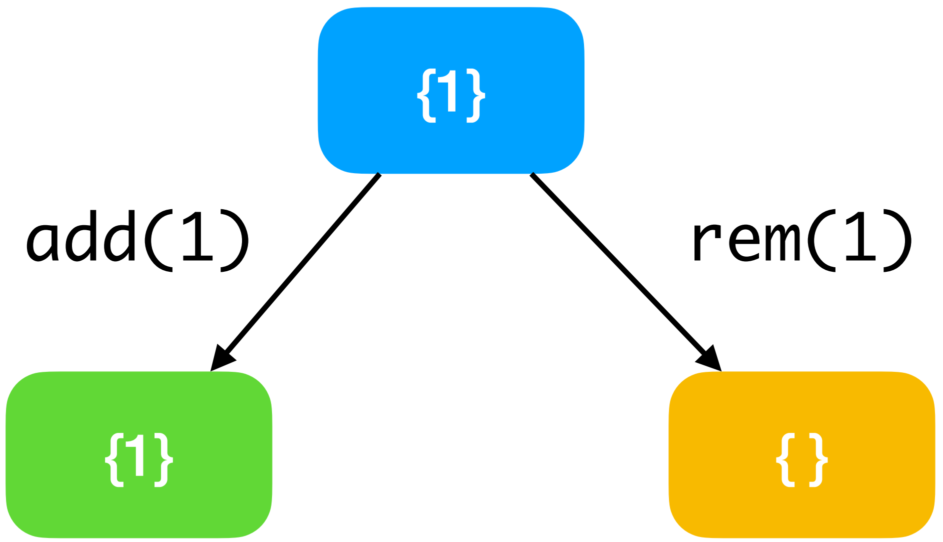


Observed-Removed Set

- OR-set — *add-wins* when there is a concurrent add and remove of the same element

```
let merge ~lca ~v1 ~v2 =
  (lca ∩ v1 ∩ v2) (* unmodified elements *)
  ∪ (v1 - lca) (* added in v1 *)
  ∪ (v2 - lca) (* added in v2 *)
```

Kaki et al. "Mergeable Replicated Data Types", OOPSLA 2019



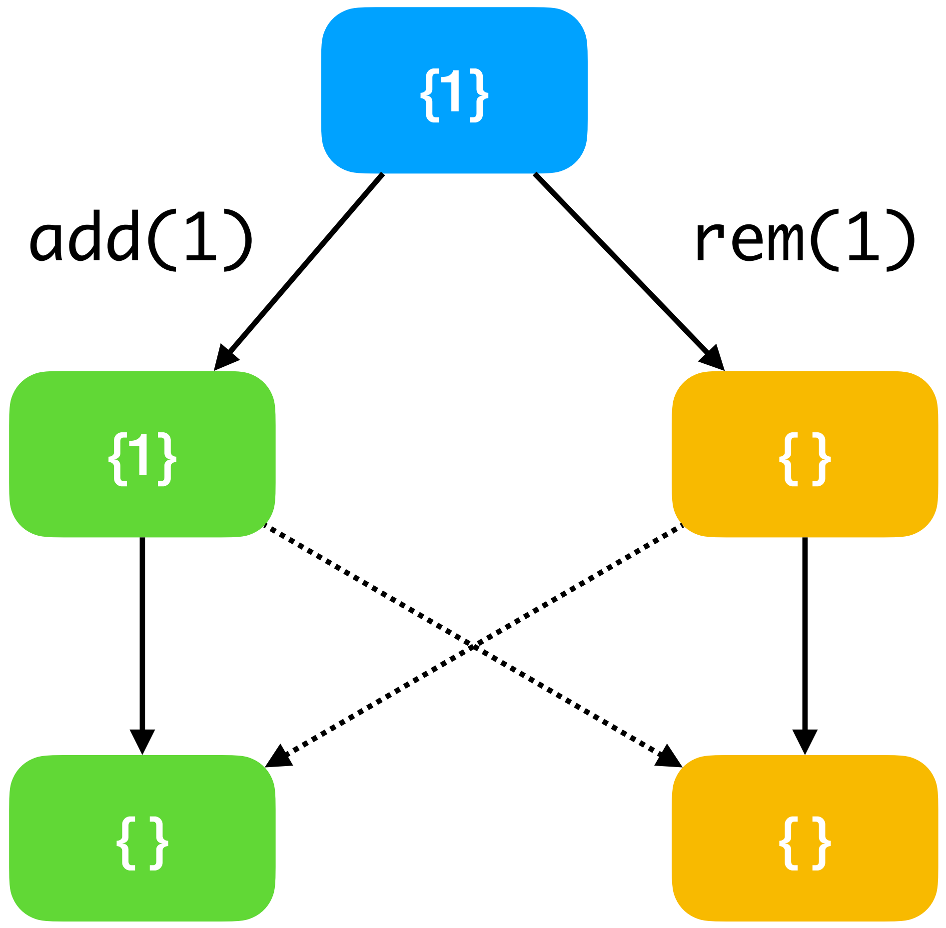
Observed-Removed Set

- OR-set — *add-wins* when there is a concurrent add and remove of the same element

```
let merge ~lca ~v1 ~v2 =
  (lca ∩ v1 ∩ v2) (* unmodified elements *)
  ∪ (v1 - lca) (* added in v1 *)
  ∪ (v2 - lca) (* added in v2 *)
```

Kaki et al. "Mergeable Replicated Data Types", OOPSLA 2019

$$\{\} \cup (\{1\} - \{1\}) \cup (\{\} - \{1\})$$
$$= \{\} \cup \{\} \cup \{\}$$
$$= \{\} \text{ (expected } \{1\})$$

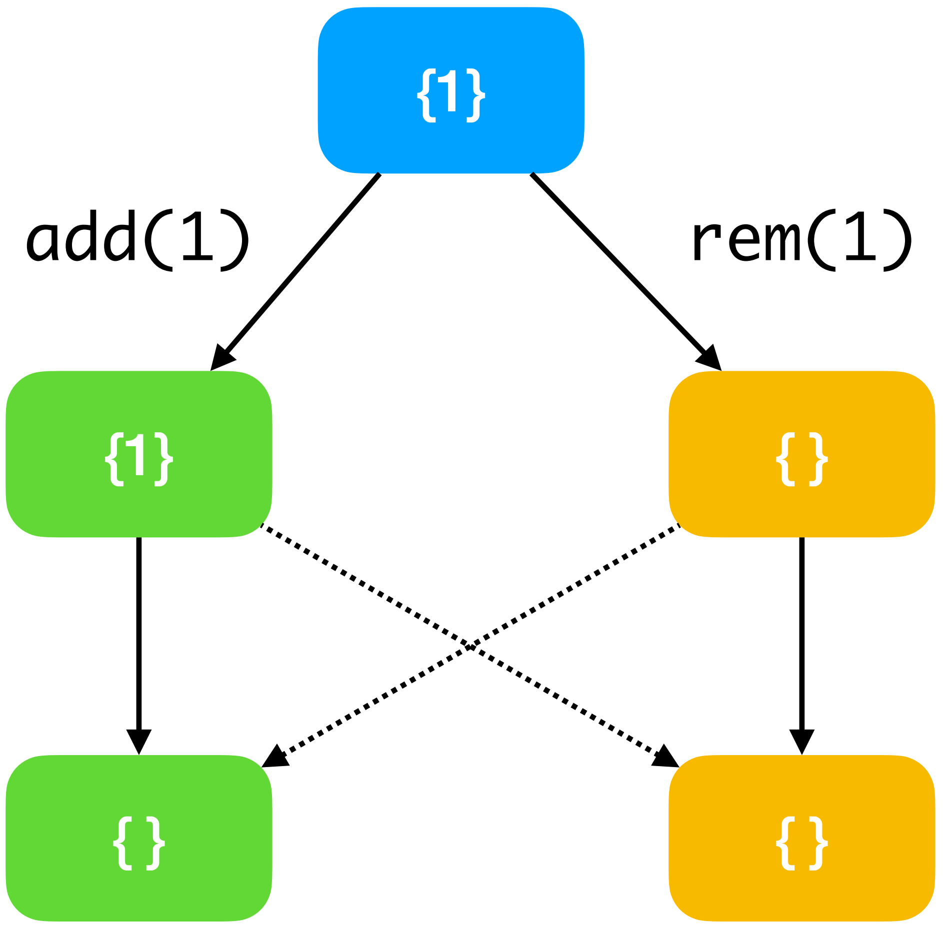


Observed-Removed Set

- OR-set — *add-wins* when there is a concurrent add and remove of the same element

```
let merge ~lca ~v1 ~v2 =
  (lca ∩ v1 ∩ v2) (* unmodified elements *)
  ∪ (v1 - lca) (* added in v1 *)
  ∪ (v2 - lca) (* added in v2 *)
```

Kaki et al. "Mergeable Replicated Data Types", OOPSLA 2019



$$
\begin{aligned}
& \{\,\} \cup (\{1\} - \{1\}) \cup (\{\,\} - \{1\}) \\
&= \{\,\} \cup \{\,\} \cup \{\,\} \\
&= \{\,\} \text{ \textit{(expected \{1\})}}
\end{aligned}
$$

- Convergence is not sufficient; *Intent* is not preserved

Concretising Intent

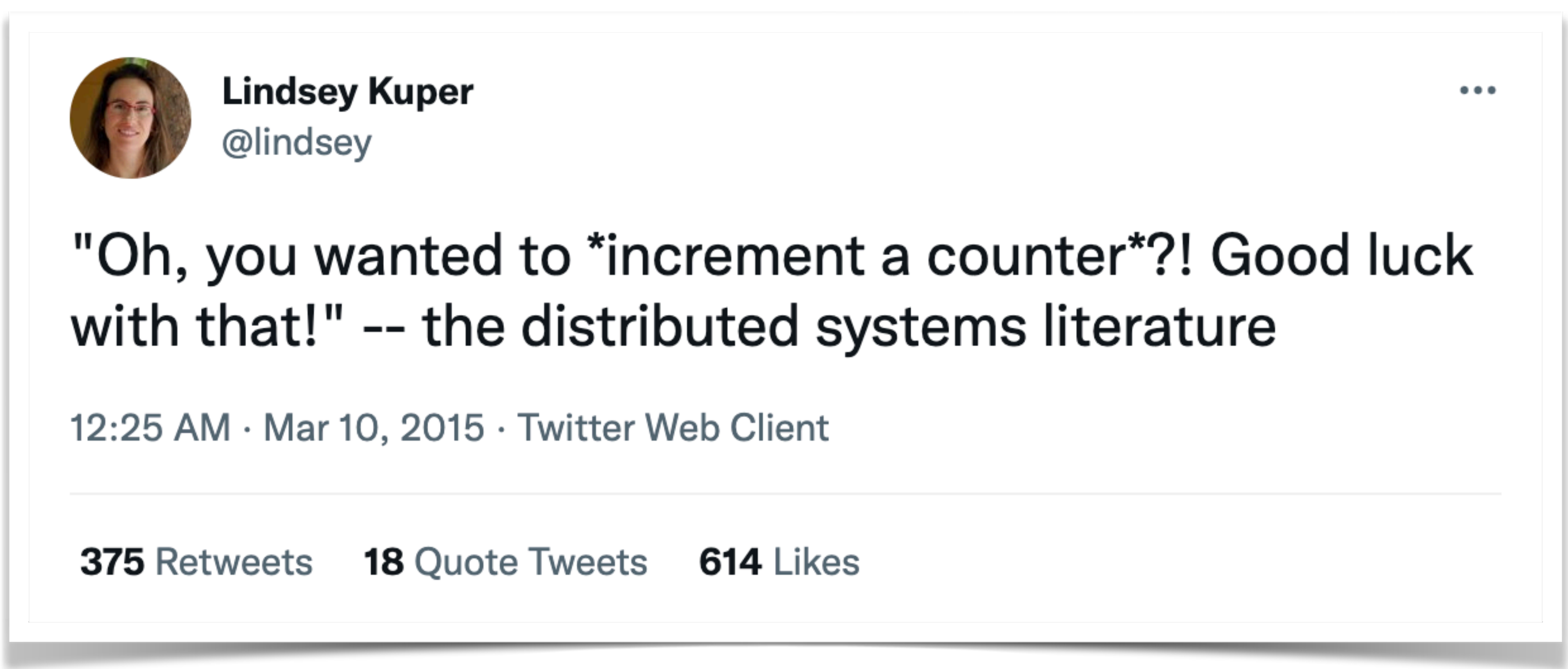
- A *formal specification language* to capture the *intent* of the MRDT
  - Must be rich enough to capture eventual consistency

Concretising Intent

- A *formal specification language* to capture the *intent* of the MRDT
    - Must be rich enough to capture eventual consistency
- Even *simple* data types attract enormous *complexity* when made *distributed*

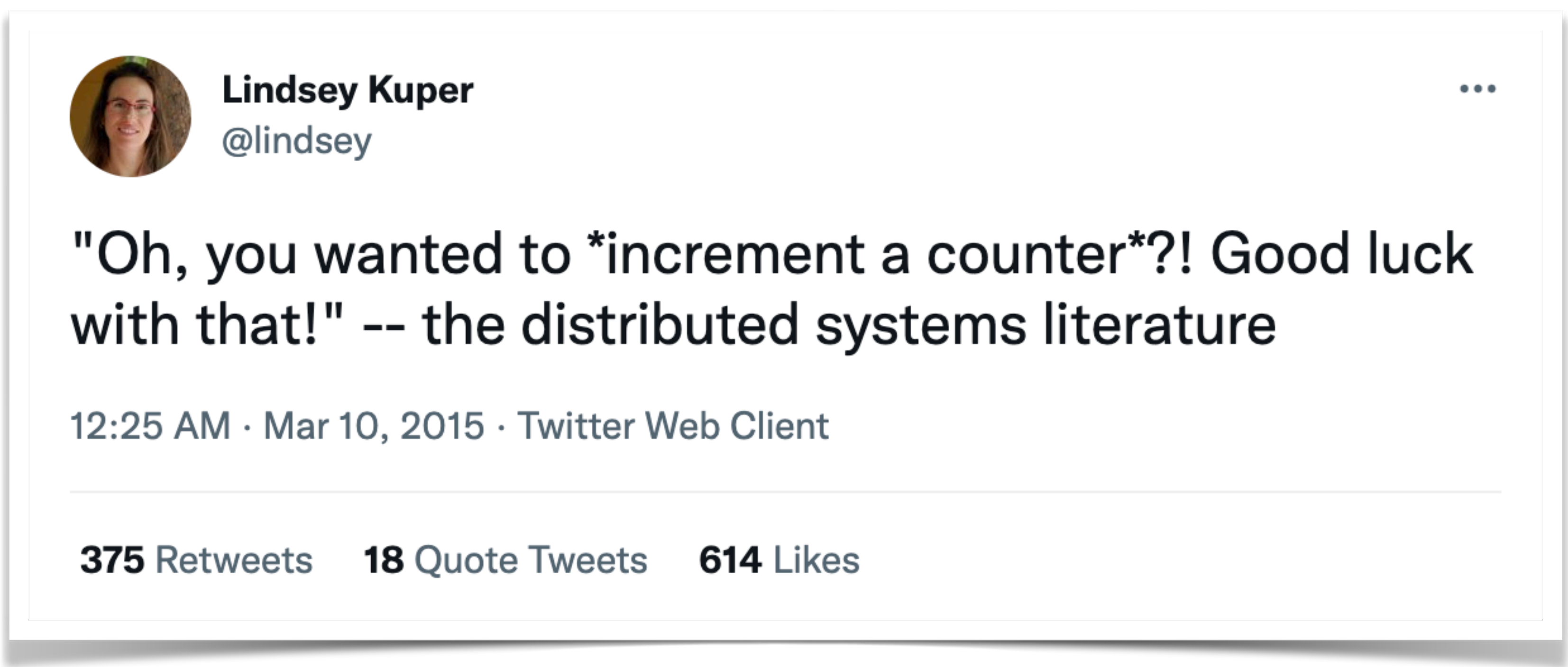
Concretising Intent

- A *formal specification language* to capture the *intent* of the MRDT
  - ✦ Must be rich enough to capture eventual consistency
- Even *simple* data types attract enormous *complexity* when made *distributed*



Concretising Intent

- A *formal specification language* to capture the *intent* of the MRDT
  - ✦ Must be rich enough to capture eventual consistency
- Even *simple* data types attract enormous *complexity* when made *distributed*



- *Mechanization* to bridge the gap between spec and impl

Peepul — Certified MRDTs

- An F* library implementing and proving MRDTs
  - ★ https://github.com/prismlab/peepul

Peepul — Certified MRDTs

- An F* library implementing and proving MRDTs
  - ★ https://github.com/prismlab/peepul
- Specification language is event-based
  - ★ Burckhardt et al. "Replicated Data Types: Specification, Verification and Optimality", POPL 2014

Peepul — Certified MRDTs

- An F* library implementing and proving MRDTs
  - ★ https://github.com/prismlab/peepul
- Specification language is event-based
  - ★ Burckhardt et al. "Replicated Data Types: Specification, Verification and Optimality", POPL 2014
- *Replication-aware simulation* to connect *specification* with *implementation*

Peepul — Certified MRDTs

- An F* library implementing and proving MRDTs
  - ★ https://github.com/prismlab/peepul
- Specification language is event-based
  - ★ Burckhardt et al. "Replicated Data Types: Specification, Verification and Optimality", POPL 2014
- *Replication-aware simulation* to connect *specification* with *implementation*
- *Space- and time-efficient* implementations
  - ★ 1st certified implementation of a O(1) replicated queue with O(n) merge.

Peepul — Certified MRDTs

- An F* library implementing and proving MRDTs
  - ★ https://github.com/prismlab/peepul
- Specification language is event-based
  - ★ Burckhardt et al. "Replicated Data Types: Specification, Verification and Optimality", POPL 2014
- *Replication-aware simulation* to connect *specification* with *implementation*
- *Space- and time-efficient* implementations
  - ★ 1st certified implementation of a O(1) replicated queue with O(n) merge.
- *Composition* of MRDTs and their proofs!

Peepul — Certified MRDTs

- An F* library implementing and proving MRDTs
  - ★ https://github.com/prismlab/peepul
- Specification language is event-based
  - ★ Burckhardt et al. "Replicated Data Types: Specification, Verification and Optimality", POPL 2014
- *Replication-aware simulation* to connect *specification* with *implementation*
- *Space- and time-efficient* implementations
  - ★ 1st certified implementation of a O(1) replicated queue with O(n) merge.
- *Composition* of MRDTs and their proofs!
- Extracted RDTs are compatible with *Irmin* — a Git-like distributed database

Fixing OR-Set

- Discriminate duplicate additions by associating a unique id

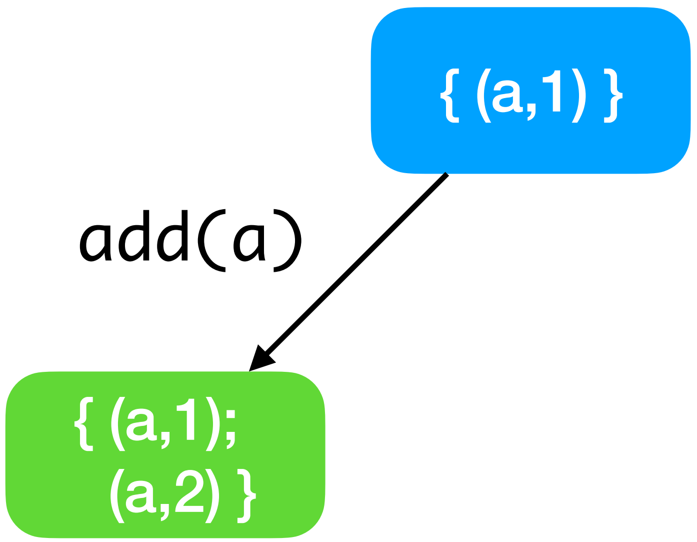
Fixing OR-Set

- Discriminate duplicate additions by associating a unique id

{ (a,1) }

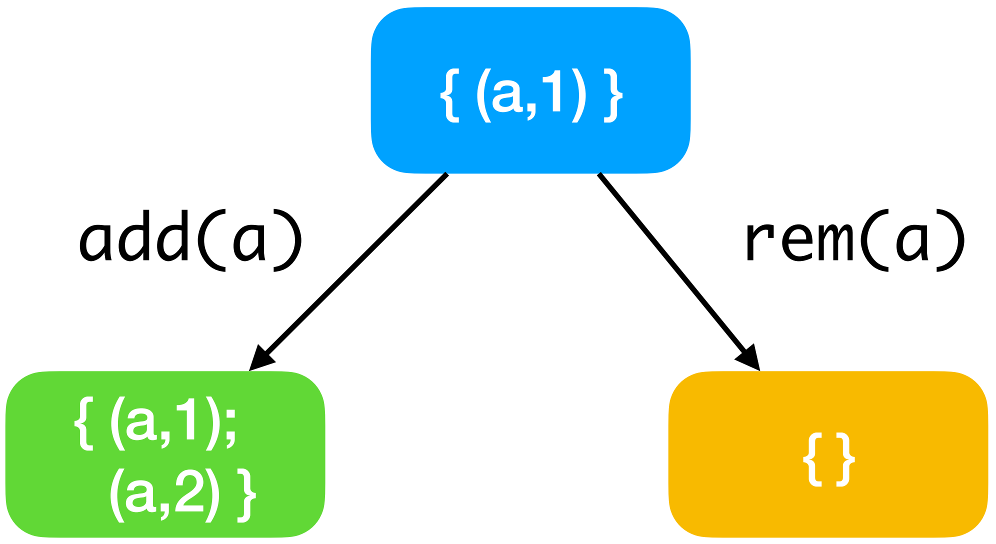
Fixing OR-Set

- Discriminate duplicate additions by associating a unique id



Fixing OR-Set

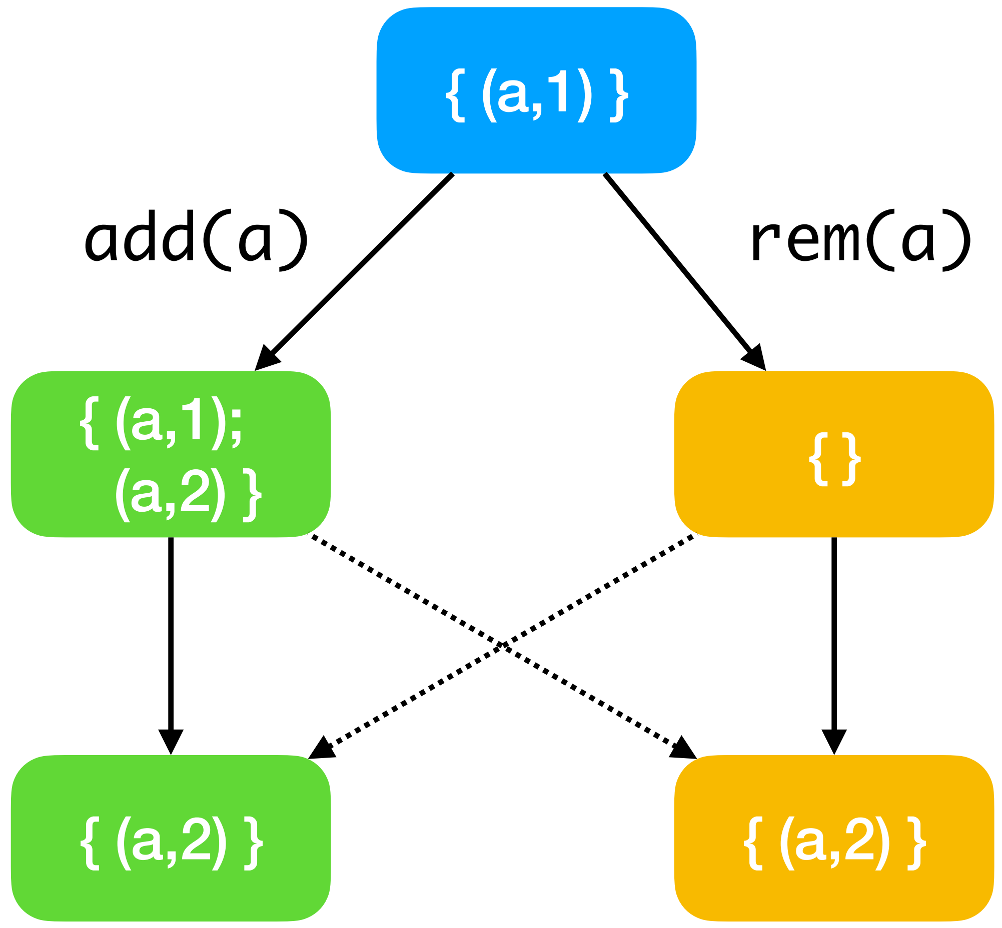
- Discriminate duplicate additions by associating a unique id



Fixing OR-Set

- Discriminate duplicate additions by associating a unique id

$$
\begin{aligned}
& \{\,\} \\
& \cup\ (\{\,(a,1);\ (a,2)\,\} - \{\,(a,1)\,\}) \\
& \cup\ (\{\,\} - \{\,(a,1)\,\}) \\
=\ & \{\,\} \cup \{\,(a,2)\,\} \cup \{\,\} \\
=\ & \{\,(a,2)\,\}
\end{aligned}
$$



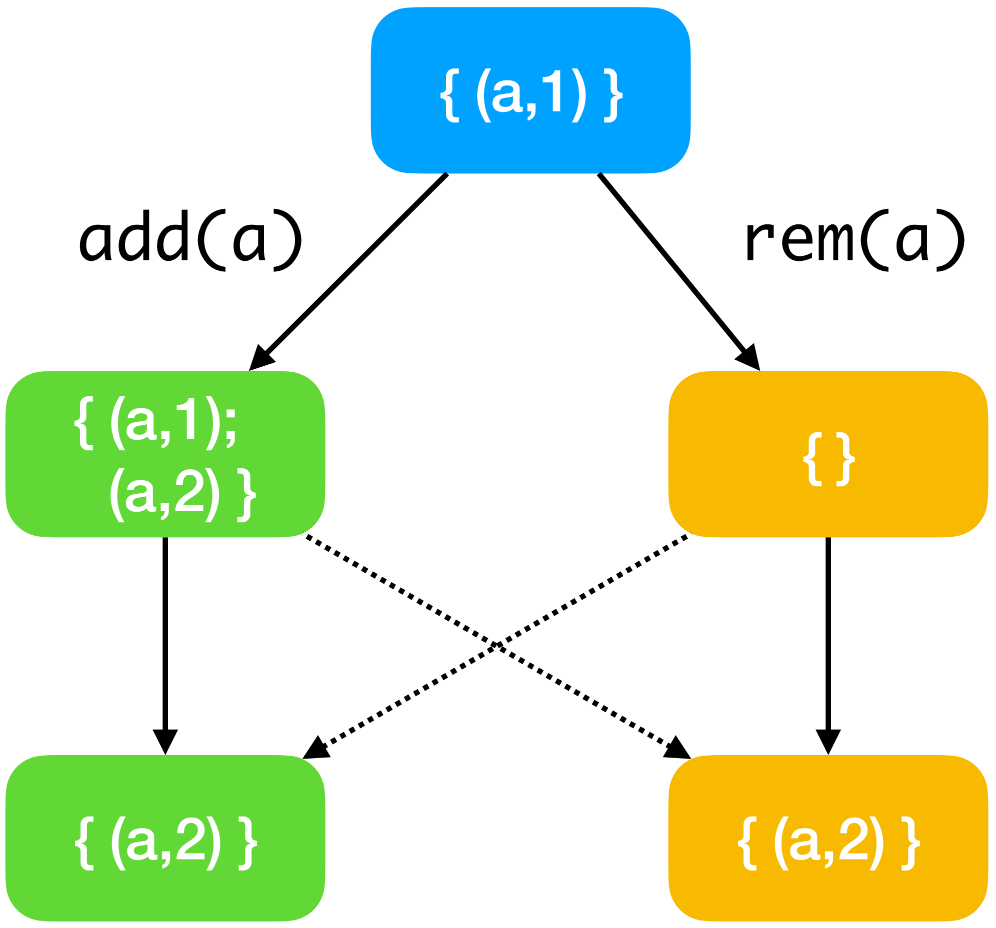
Fixing OR-Set

- Discriminate duplicate additions by associating a unique id

  { }
  ∪ ({ (a,1); (a,2) } - { (a,1) })
  ∪ ({ } - { (a,1) })
  = { } ∪ { (a,2) } ∪ { }
  = { (a,2) }

- MRDT implementation

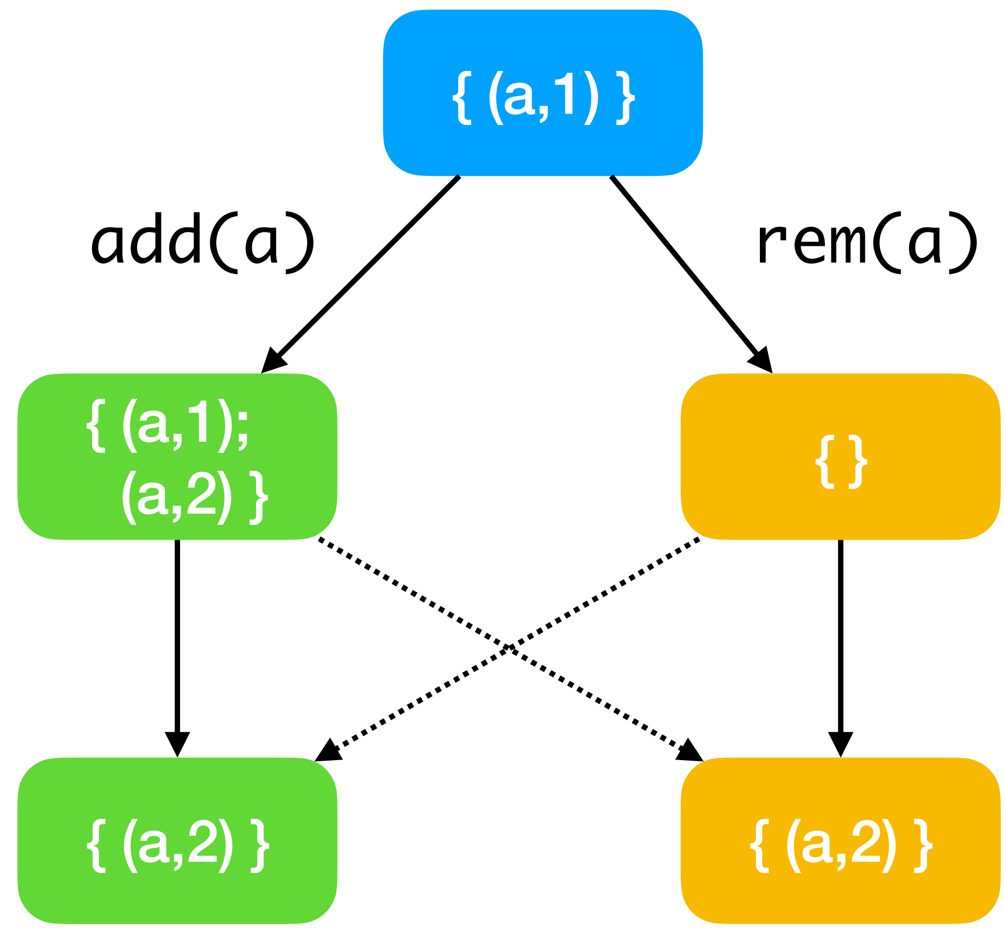
$$D_\tau = (\Sigma, \sigma_0, do, merge)$$



Fixing OR-Set

- Discriminate duplicate additions by associating a unique id

$$
\begin{aligned}
& \{\,\} \\
& \cup\ (\ \{\ (a,1);\ (a,2)\ \}\ -\ \{\ (a,1)\ \})\\
& \cup\ (\ \{\,\}\ -\ \{\ (a,1)\ \}\ )\\
= & \ \{\,\} \cup \{\ (a,2)\ \} \cup \{\,\}\\
= & \ \{\ (a,2)\ \}
\end{aligned}
$$



- MRDT implementation

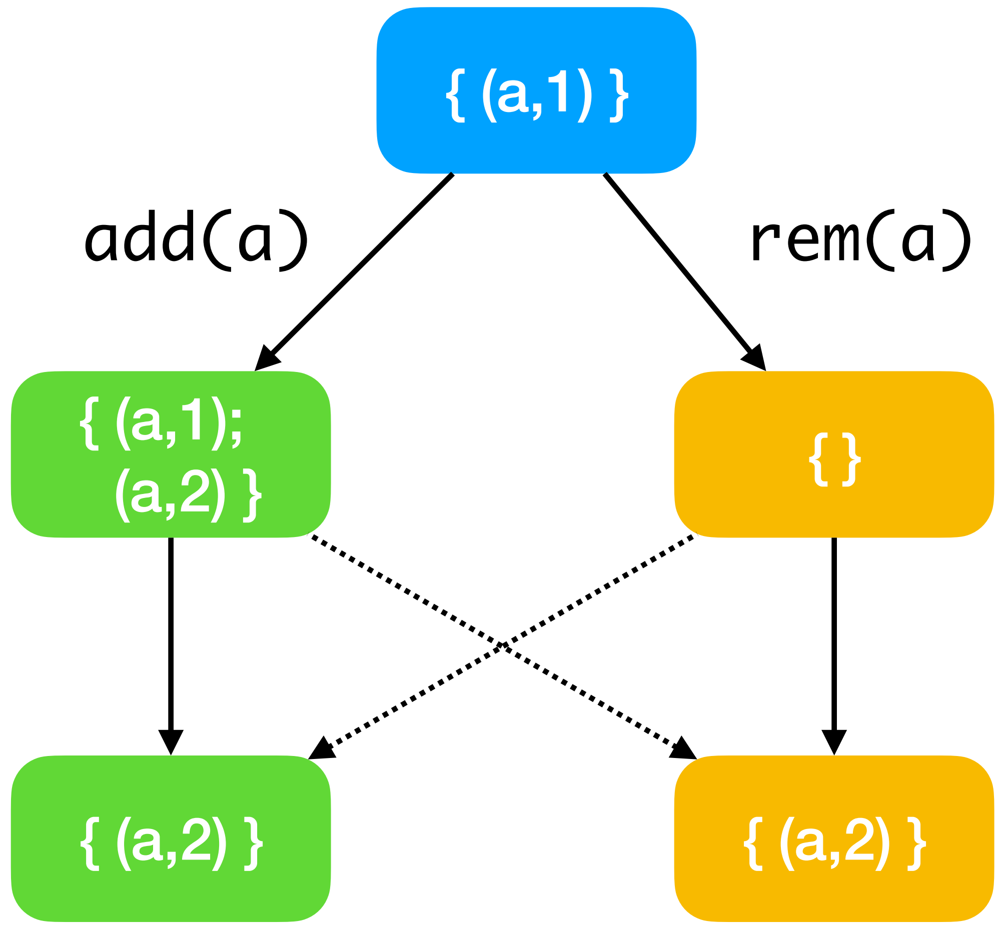
$$D_\tau = (\Sigma, \sigma_0, do, merge)$$

1: $\Sigma = \mathcal{P}(\mathbb{N} \times \mathbb{N})$
2: $\sigma_0 = \{\}$
3: $do(rd, \sigma, t) = (\sigma, \{a \mid (a, t) \in \sigma\})$
4: $do(add(a), \sigma, t) = (\sigma \cup \{(a, t)\}, \bot)$
5: $do(remove(a), \sigma, t) = (\{e \in \sigma \mid fst(e) \neq a\}, \bot)$
6: $merge(\sigma_{lca}, \sigma_a, \sigma_b) =$
$(\sigma_{lca} \cap \sigma_a \cap \sigma_b) \cup (\sigma_a - \sigma_{lca}) \cup (\sigma_b - \sigma_{lca})$

Fixing OR-Set

- Discriminate duplicate additions by associating a unique id

$$
\begin{aligned}
&\{\,\} \\
&\cup\ (\{ (a,1); (a,2) \} - \{ (a,1) \}) \\
&\cup\ (\{\,\} - \{ (a,1) \}) \\
=\ &\{\,\} \cup \{ (a,2) \} \cup \{\,\} \\
=\ &\{ (a,2) \}
\end{aligned}
$$



- MRDT implementation

$$D_\tau = (\Sigma, \sigma_0, do, merge)$$

Unique Lamport Timestamps

$\Sigma = \mathcal{P}(\mathbb{N} \times \mathbb{N})$

$\sigma_0 = \{\}$

$do(rd, \sigma, t) = (\sigma, \{a \mid (a, t) \in \sigma\})$

$do(add(a), \sigma, t) = (\sigma \cup \{(a, t)\}, \bot)$

$do(remove(a), \sigma, t) = (\{e \in \sigma \mid fst(e) \neq a\}, \bot)$

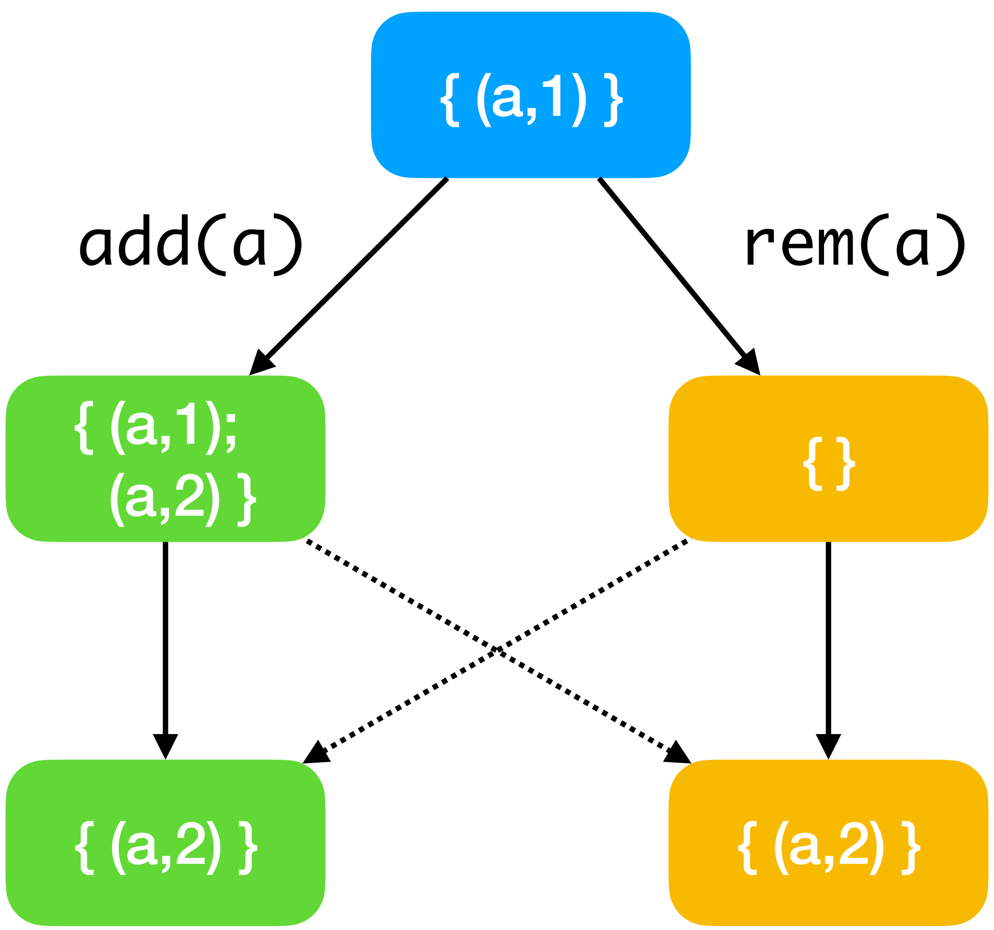
$merge(\sigma_{lca}, \sigma_a, \sigma_b) =$
$(\sigma_{lca} \cap \sigma_a \cap \sigma_b) \cup (\sigma_a - \sigma_{lca}) \cup (\sigma_b - \sigma_{lca})$

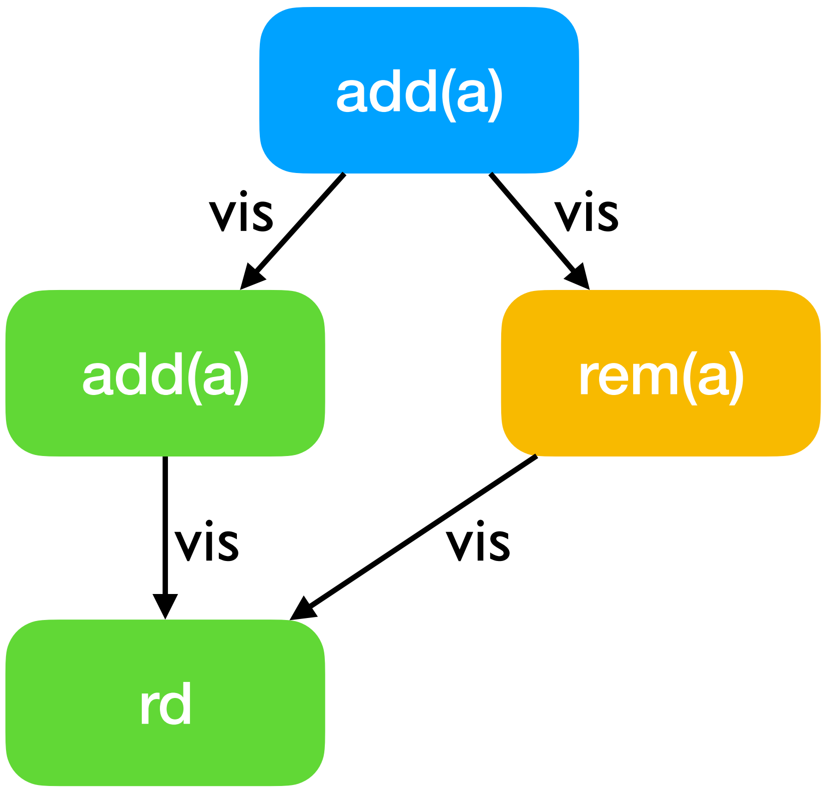
Specifying OR-Set

Abstract state $I = \langle E, oper, rval, time, vis \rangle$

Specifying OR-Set

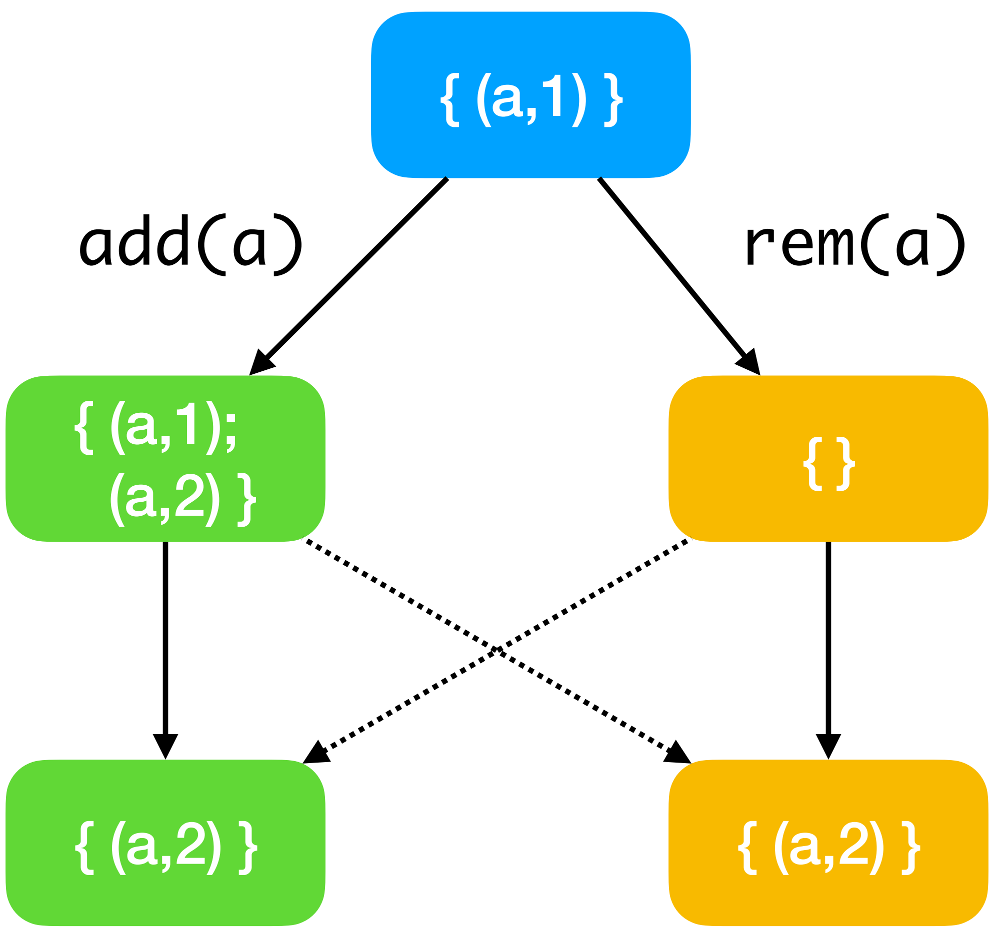
Abstract state $I = \langle E, oper, rval, time, vis \rangle$

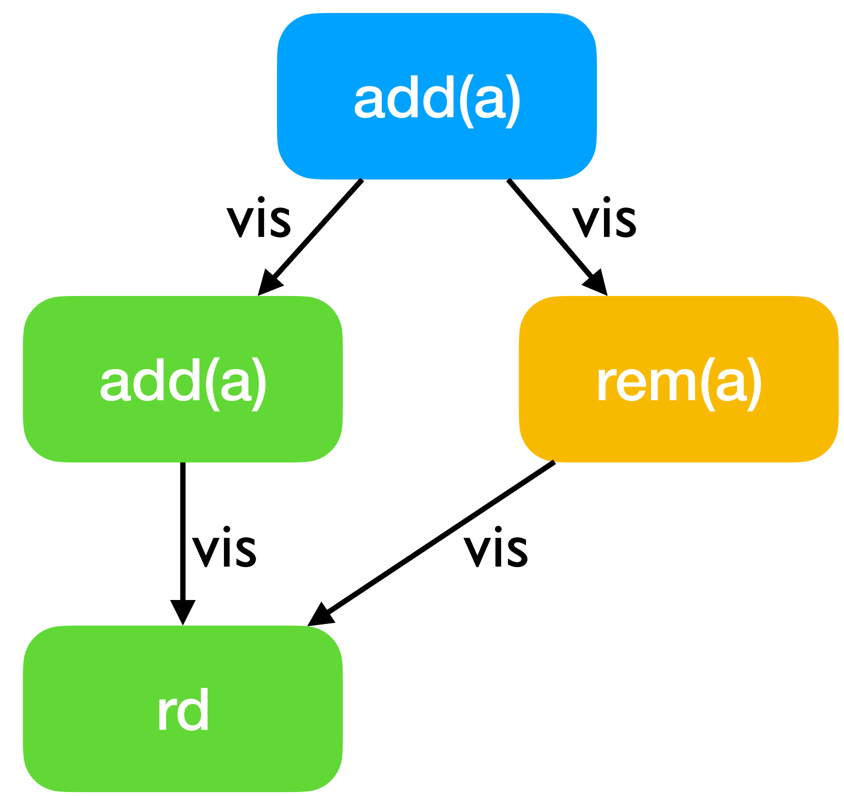




Specifying OR-Set

Abstract state $I = \langle E, oper, rval, time, vis \rangle$



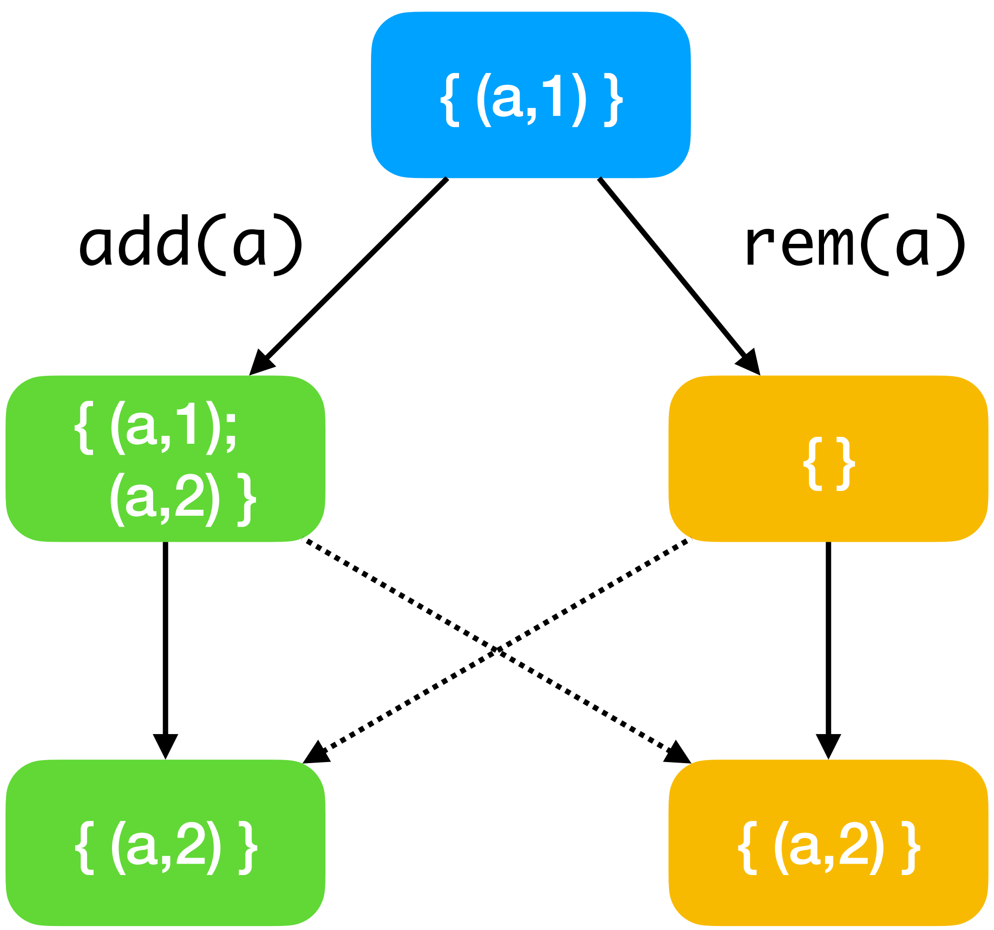


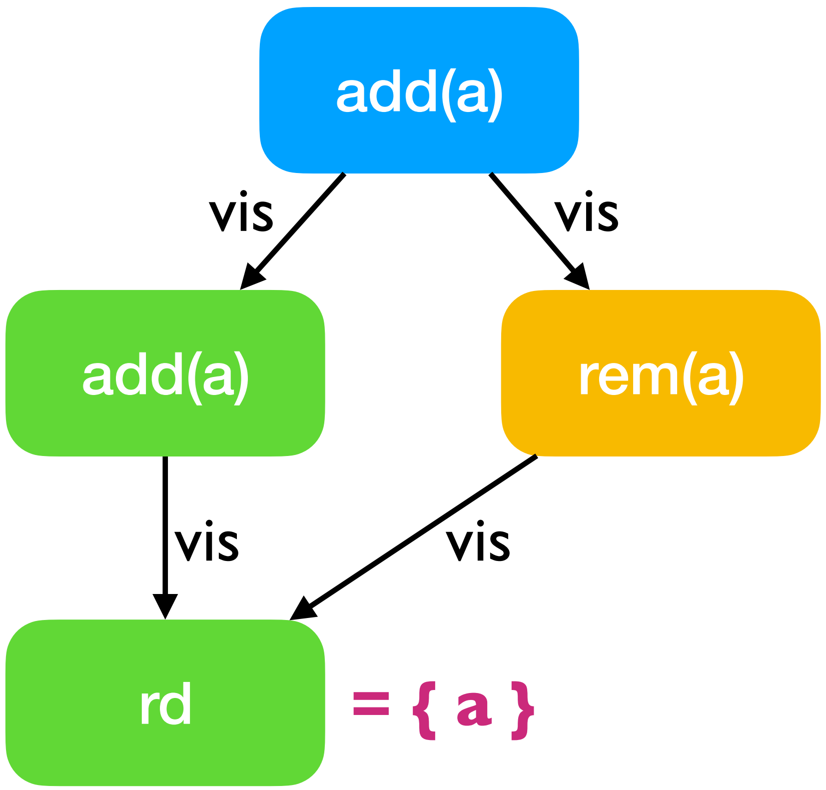
$$\mathcal{F}_{orset}(\mathrm{rd}, \langle E, oper, rval, time, vis \rangle) = \{a \mid \exists e \in E.\ oper(e)$$

$$= \mathrm{add}(a) \wedge \neg(\exists f \in E.\ oper(f) = \mathrm{remove}(a) \wedge e \xrightarrow{vis} f)\}$$

Specifying OR-Set

Abstract state $I = \langle E, oper, rval, time, vis \rangle$





$$\mathcal{F}_{orset}(\mathrm{rd}, \langle E, oper, rval, time, vis \rangle) = \{a \mid \exists e \in E.\ oper(e)$$
$$= \mathrm{add}(a) \wedge \neg(\exists f \in E.\ oper(f) = \mathrm{remove}(a) \wedge e \xrightarrow{vis} f)\}$$

Simulation Relation

Simulation Relation

- Connects the *abstract state* with the *concrete state*

Simulation Relation

- Connects the *abstract state* with the *concrete state*
- For the OR-set,

Simulation Relation

- Connects the *abstract state* with the *concrete state*
- For the OR-set,

$$\mathcal{R}_{sim}(I, \sigma) \iff (\forall (a, t) \in \sigma \iff$$
$$(\exists e \in I.E \land I.\,oper(e) = add(a) \land I.time(e) = t \land$$
$$\neg(\exists f \in I.E \land I.\,oper(f) = remove(a) \land e \xrightarrow{vis} f)))$$

Simulation Relation

- Connects the *abstract state* with the *concrete state*
- For the OR-set,

$$\mathcal{R}_{sim}(I,\sigma) \iff (\forall (a,t) \in \sigma \iff$$
$$(\exists e \in I.E \wedge I.\,oper(e) = add(a) \wedge I.time(e) = t \wedge$$
$$\neg(\exists f \in I.E \wedge I.\,oper(f) = remove(a) \wedge e \xrightarrow{vis} f)))$$

- The main verification effort is to show that the relation above is indeed a *simulation relation*
  - ★ Shown separately for *operations* and *merge function*
  - ★ Proof by induction on the execution trace

Verification effort

MRDTs verified	#Lines code	#Lines proof	#Lemmas	Verif. time (s)
Increment-only counter	6	43	2	3.494
PN counter	8	43	2	23.211
Enable-wins flag	20	58	3	1074
		81	6	171
		89	7	104
LWW register	5	44	1	4.21
G-set	10	23	0	4.71
		28	1	2.462
		33	2	1.993
G-map	48	26	0	26.089
Mergeable log	39	95	2	36.562
OR-set (§2.1.1)	30	36	0	43.85
		41	1	21.656
		46	2	8.829
OR-set-space (§2.1.2)	59	108	7	1716
OR-set-spacetime	97	266	7	1854
Queue	32	1123	75	4753

Verification effort

MRDTs verified	#Lines code	#Lines proof	#Lemmas	Verif. time (s)
Increment-only counter	6	43	2	3.494
PN counter	8	43	2	23.211
Enable-wins flag	20	58	3	1074
		81	6	171
		89	7	104
LWW register	5	44	1	4.21
G-set	10	23	0	4.71
		28	1	2.462
		33	2	1.993
G-map	48	26	0	26.089
Mergeable log	39	95	2	36.562
OR-set (§2.1.1)	30	36	0	43.85
		41	1	21.656
		46	2	8.829
OR-set-space (§2.1.2)	59	108	7	1716
OR-set-spacetime	97	266	7	1854
Queue	32	1123	75	4753

Composing RDTs is HARD!

Martin Kleppmann
@martinkl

Today in "distributed systems are hard": I wrote down a simple CRDT algorithm that I thought was "obviously correct" for a course I'm teaching. Only 10 lines or so long. Found a fatal bug only after spending hours trying to prove the algorithm correct. 😭

4:18 AM · Nov 13, 2020 · Tweetbot for iOS

41 Retweets **4** Quote Tweets **541** Likes

Martin Kleppmann @martinkl · Nov 13, 2020

The interesting thing about this bug is that it comes about only from the interaction of two features. A LWW map by itself is fine. A set in which you can insert and delete elements (but not update them) is fine. The problem arises only when delete and update interact.

1 16

Composing IRC-style chat

- Build IRC-style group chat
  - ★ Send and read messages in channels

Composing IRC-style chat

- Build IRC-style group chat
  - ★ Send and read messages in channels
- Represent application state as a map MRDT
  - ★ String (*channel name*) keys → mergeable-log MRDT values

Composing IRC-style chat

- Build IRC-style group chat
  - ★ Send and read messages in channels
- Represent application state as a map MRDT
  - ★ String (*channel name*) keys → mergeable-log MRDT values
- **Goal:**
  - ★ *map* and *log* proved correct separately
  - ★ Use the proof of underlying RDTs to prove chat application correctness

Generic Map MRDT

Implementation

$$\mathcal{D}_{\alpha-map} = (\Sigma, \sigma_0, do, merge_{\alpha-map}) \text{ where}$$

1: $\Sigma_{\alpha-map} = \mathcal{P}(string \times \Sigma_\alpha)$

2: $\sigma_0 = \{\}$

3: $\delta(\sigma, k) = \begin{cases} \sigma(k), & \text{if } k \in dom(\sigma) \\ \sigma_{0_\alpha}, & \text{otherwise} \end{cases}$

4: $do(set(k, o_\alpha), \sigma, t) =$
$\quad \text{let } (v, r) = do_\alpha(o_\alpha, \delta(\sigma, k), t) \text{ in } (\sigma[k \mapsto v], r)$

5: $do(get(k, o_\alpha), \sigma, t) =$
$\quad \text{let } (\_, r) = do_\alpha(o_\alpha, \delta(\sigma, k), t) \text{ in } (\sigma, r)$

6: $merge_{\alpha-map}(\sigma_{lca}, \sigma_a, \sigma_b) =$
$\quad \{(k, v) \mid (k \in dom(\sigma_{lca}) \cup dom(\sigma_a) \cup dom(\sigma_b)) \wedge$
$\qquad v = merge_\alpha(\delta(\sigma_{lca}, k), \delta(\sigma_a, k), \delta(\sigma_b, k))$

Simulation Relation

$$\mathcal{R}_{sim-\alpha-map}(I, \sigma) \iff \forall k.$$

1: $(k \in dom(\sigma) \iff \exists e \in I.E.\ oper(e) = set(k, \_)) \wedge$

2: $\quad \mathcal{R}_{sim-\alpha}\ (project(k, I),\ \delta(\sigma, k))$

Generic Map MRDT

Implementation

$$\mathcal{D}_{\alpha-map} = (\Sigma, \sigma_0, do, merge_{\alpha-map}) \text{ where}$$

$$\Sigma_{\alpha-map} = \mathcal{P}(string \times \Sigma_\alpha)$$

The values in the MRDT map are MRDTs

$$\sigma_0 = \{\}$$

$$\delta(\sigma, k) = \begin{cases} \sigma(k), & \text{if } k \in dom(\sigma) \\ \sigma_{0_\alpha}, & \text{otherwise} \end{cases}$$

$$do(set(k, o_\alpha), \sigma, t) = \\ \text{let } (v, r) = do_\alpha(o_\alpha, \delta(\sigma, k), t) \text{ in } (\sigma[k \mapsto v], r)$$

$$do(get(k, o_\alpha), \sigma, t) = \\ \text{let } (\_, r) = do_\alpha(o_\alpha, \delta(\sigma, k), t) \text{ in } (\sigma, r)$$

$$merge_{\alpha-map}(\sigma_{lca}, \sigma_a, \sigma_b) = \\ \{(k, v) \mid (k \in dom(\sigma_{lca}) \cup dom(\sigma_a) \cup dom(\sigma_b)) \land \\ v = merge_\alpha(\delta(\sigma_{lca}, k), \delta(\sigma_a, k), \delta(\sigma_b, k))$$

Simulation Relation

$$\mathcal{R}_{sim-\alpha-map}(I, \sigma) \iff \forall k.$$

$$(k \in dom(\sigma) \iff \exists e \in I.E.\ oper(e) = set(k, \_)) \land$$

$$\mathcal{R}_{sim-\alpha}\ (project(k, I),\ \delta(\sigma, k))$$

Generic Map MRDT

Implementation

$$\mathcal{D}_{\alpha-map} = (\Sigma, \sigma_0, do, merge_{\alpha-map}) \text{ where}$$

$$\Sigma_{\alpha-map} = \mathcal{P}(string \times \Sigma_\alpha)$$

The values in the MRDT map are MRDTs

$$\sigma_0 = \{\}$$

$$\delta(\sigma, k) = \begin{cases} \sigma(k), & \text{if } k \in dom(\sigma) \\ \sigma_{0_\alpha}, & \text{otherwise} \end{cases}$$

$$do(set(k, o_\alpha), \sigma, t) = \text{let } (v, r) = do_\alpha(o_\alpha, \delta(\sigma, k), t) \text{ in } (\sigma[k \mapsto v], r)$$

$$do(get(k, o_\alpha), \sigma, t) = \text{let } (\_, r) = do_\alpha(o_\alpha, \delta(\sigma, k), t) \text{ in } (\sigma, r)$$

$$merge_{\alpha-map}(\sigma_{lca}, \sigma_a, \sigma_b) = \{(k, v) \mid (k \in dom(\sigma_{lca}) \cup dom(\sigma_a) \cup dom(\sigma_b)) \wedge v = merge_\alpha(\delta(\sigma_{lca}, k), \delta(\sigma_a, k), \delta(\sigma_b, k))$$

Merge uses the merge of the underlying value type!

Simulation Relation

$$\mathcal{R}_{sim-\alpha-map}(I, \sigma) \iff \forall k.$$
$$(k \in dom(\sigma) \iff \exists e \in I.E.\ oper(e) = set(k, \_)) \wedge$$
$$\mathcal{R}_{sim-\alpha}\ (project(k, I),\ \delta(\sigma, k))$$

Generic Map MRDT

Implementation

$$\mathcal{D}_{\alpha-map} = (\Sigma, \sigma_0, do, merge_{\alpha-map}) \text{ where}$$

$$\Sigma_{\alpha-map} = \mathcal{P}(string \times \Sigma_\alpha)$$

The values in the MRDT map are MRDTs

$$\sigma_0 = \{\}$$

$$\delta(\sigma, k) = \begin{cases} \sigma(k), & \text{if } k \in dom(\sigma) \\ \sigma_{0_\alpha}, & \text{otherwise} \end{cases}$$

$$do(set(k, o_\alpha), \sigma, t) = \text{let } (v, r) = do_\alpha(o_\alpha, \delta(\sigma, k), t) \text{ in } (\sigma[k \mapsto v], r)$$

$$do(get(k, o_\alpha), \sigma, t) = \text{let } (\_, r) = do_\alpha(o_\alpha, \delta(\sigma, k), t) \text{ in } (\sigma, r)$$

$$merge_{\alpha-map}(\sigma_{lca}, \sigma_a, \sigma_b) = \{(k, v) \mid (k \in dom(\sigma_{lca}) \cup dom(\sigma_a) \cup dom(\sigma_b)) \wedge v = merge_\alpha(\delta(\sigma_{lca}, k), \delta(\sigma_a, k), \delta(\sigma_b, k))$$

Merge uses the merge of the underlying value type!

Simulation Relation

$$\mathcal{R}_{sim-\alpha-map}(I, \sigma) \iff \forall k.$$

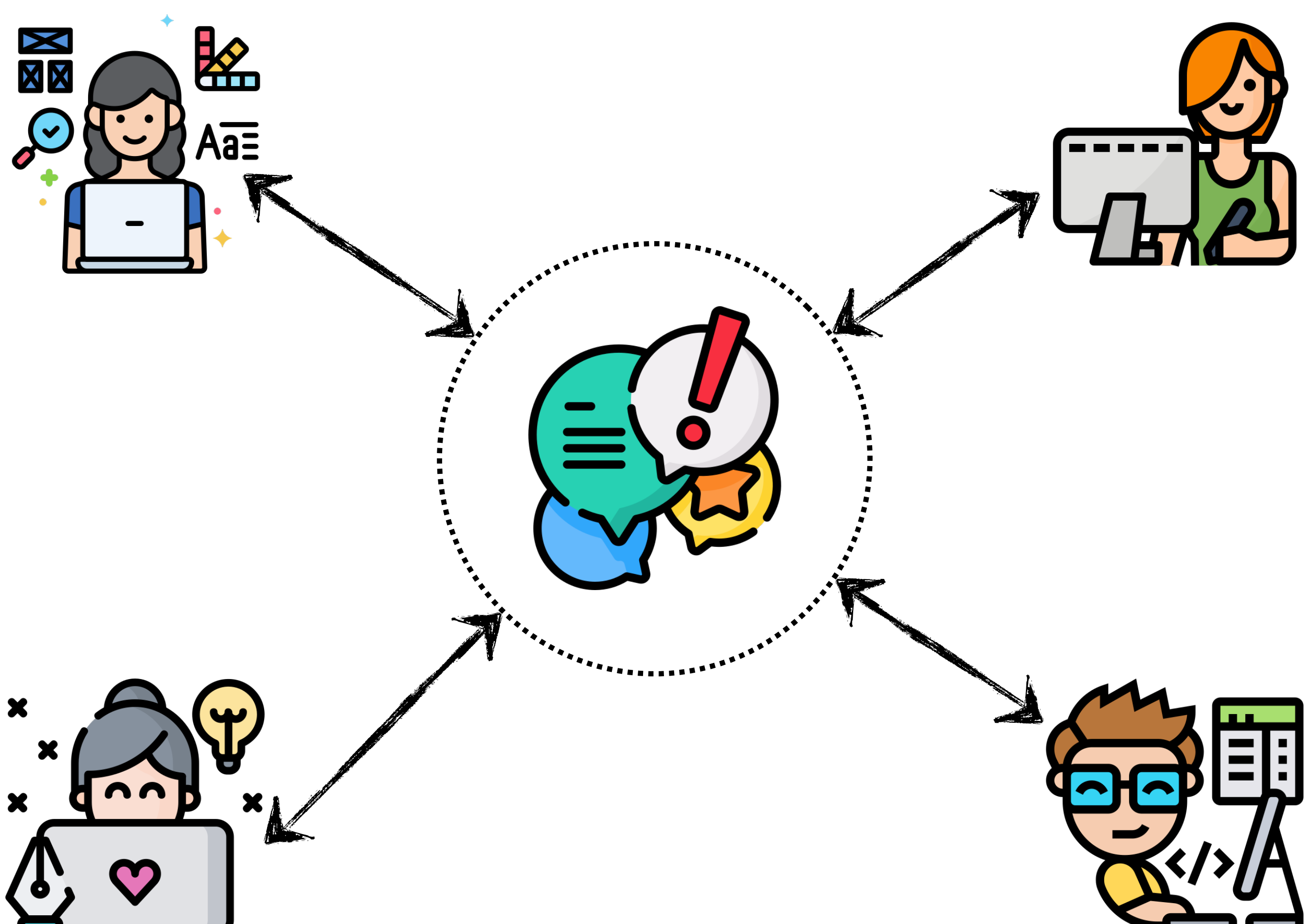
$$(k \in dom(\sigma) \iff \exists e \in I.E.\ oper(e) = set(k, \_)) \wedge$$

$$\mathcal{R}_{sim-\alpha}\ (project(k, I),\ \delta(\sigma, k))$$

Simulation relation appeals to the value type's simulation relation!

Composing IRC-style chat

- IRC app state is constructed by instantiating *generic map* with *mergeable log*
- The proof of correctness of the chat application directly follows from the composition.
  - ★ See paper for details!



Summary

- **Peepul**
  - ✦ An F* library implementing and proving MRDTs
  - ★ **https://github.com/prismlab/peepul**

Summary

- **Peepul**
  - An F* library implementing and proving MRDTs
  - **https://github.com/prismlab/peepul**
- *Space- and time-efficient* implementations
  - Certified implementation of a O(1) replicated queue with O(n) merge.

Summary

- **Peepul**
  - ✦ An F* library implementing and proving MRDTs
  - ★ **https://github.com/prismlab/peepul**
- *Space- and time-efficient* implementations
  - ★ Certified implementation of a O(1) replicated queue with O(n) merge.
- *Composition* of MRDTs and their proofs!

Summary

- **Peepul**
  - ✦ An F* library implementing and proving MRDTs
  - ★ **https://github.com/prismlab/peepul**
- *Space- and time-efficient* implementations
  - ★ Certified implementation of a O(1) replicated queue with O(n) merge.
- *Composition* of MRDTs and their proofs!
- See paper for
  - ✦ Formal description of the system + soundness proof
  - ✦ Case study on replicated queues
  - ✦ Performance results