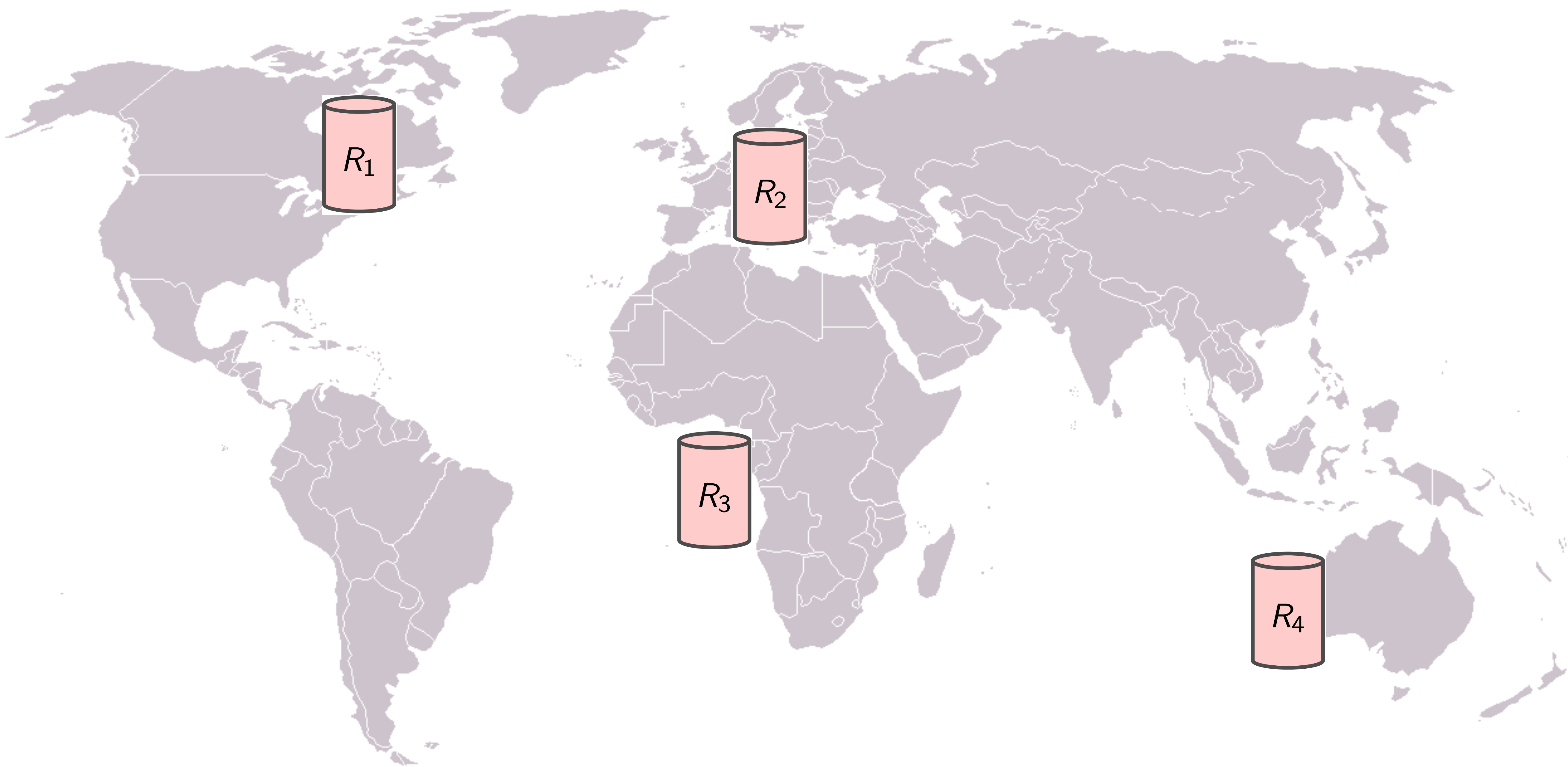
Specification And Implementation Of Replicated Data Types

Fabio Gadducci

joint work with Hernán Melgratti, Christian Roldán, and Matteo Sammartino

The setting: replicated data stores



Quickest background, 1

- Distributed systems replicate their state over different nodes in order to satisfy non-functional requirements.
- *Strong consistency* (every request receives the most recent update) of replicated data is in conflict with *availability* (every request is eventually executed) and tolerance to network *partitions* (the system operates even in the presence of failures that prevent communication among components).
- CAP theorem: it is impossible to simultaneously achieve strong Consistency, Availability and Partition tolerance.

CAP Theorem [Gilbert&Lynch,2002]

- It is impossible to simultaneously achieve
  - Consistency (read the latest written value):
    - Single system image (SSI)/linearizability
  - Availability (always-accessible)
    - Low latency
  - Partition-tolerance (partial failures)

We should cope with weaker notions of consistency

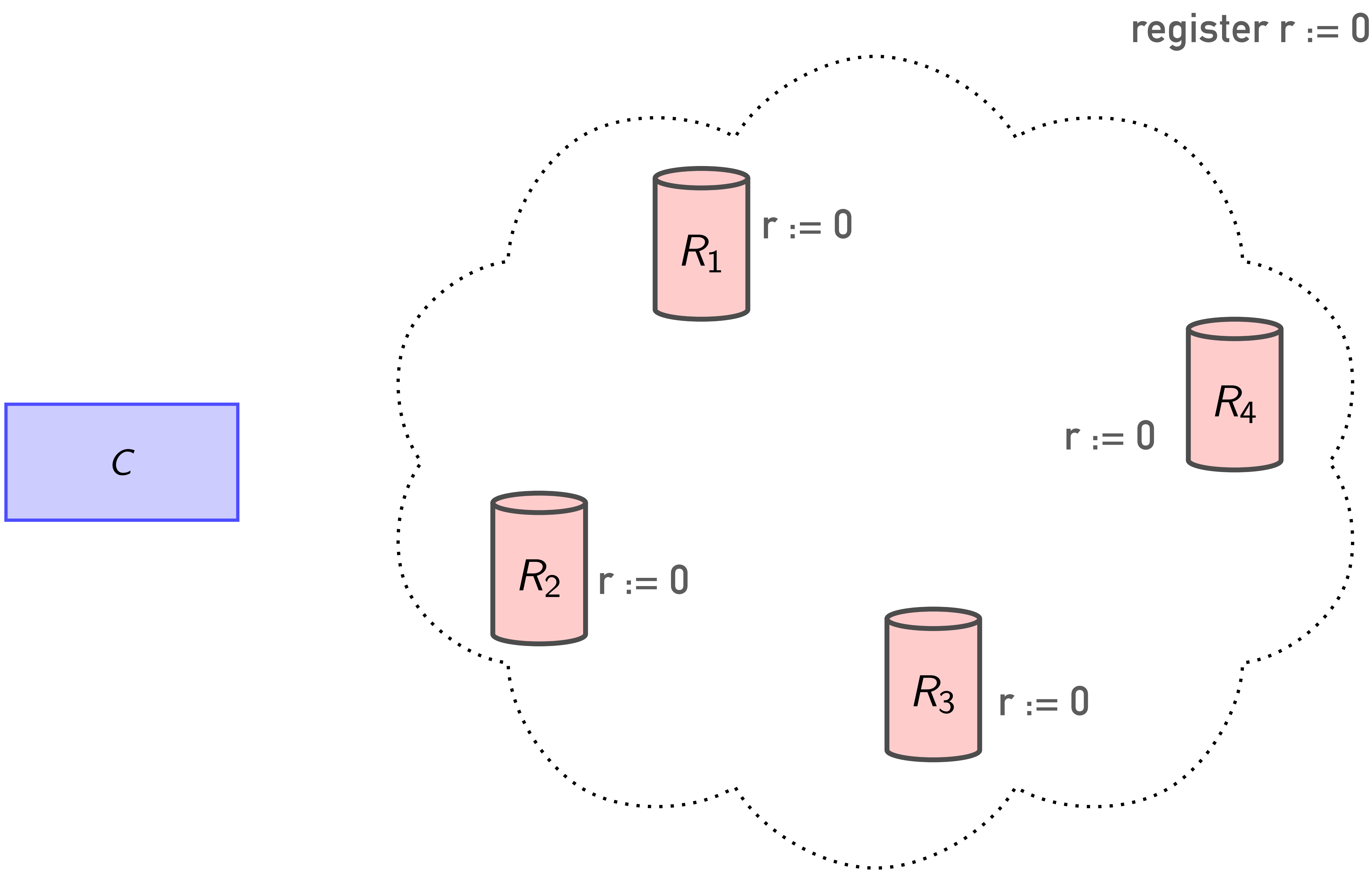
Quickest background, 2

- *Weak* consistency: replicas may (temporarily) exhibit discrepancies (every request receives a correct update).
- How are the data specified? States, state transitions and returned values should account for the different views that a data item may simultaneously have.
- In the end, consistency has to be *eventually* guaranteed (if no new updates are made to a data item, eventually all accesses to that item will return the most recent update).

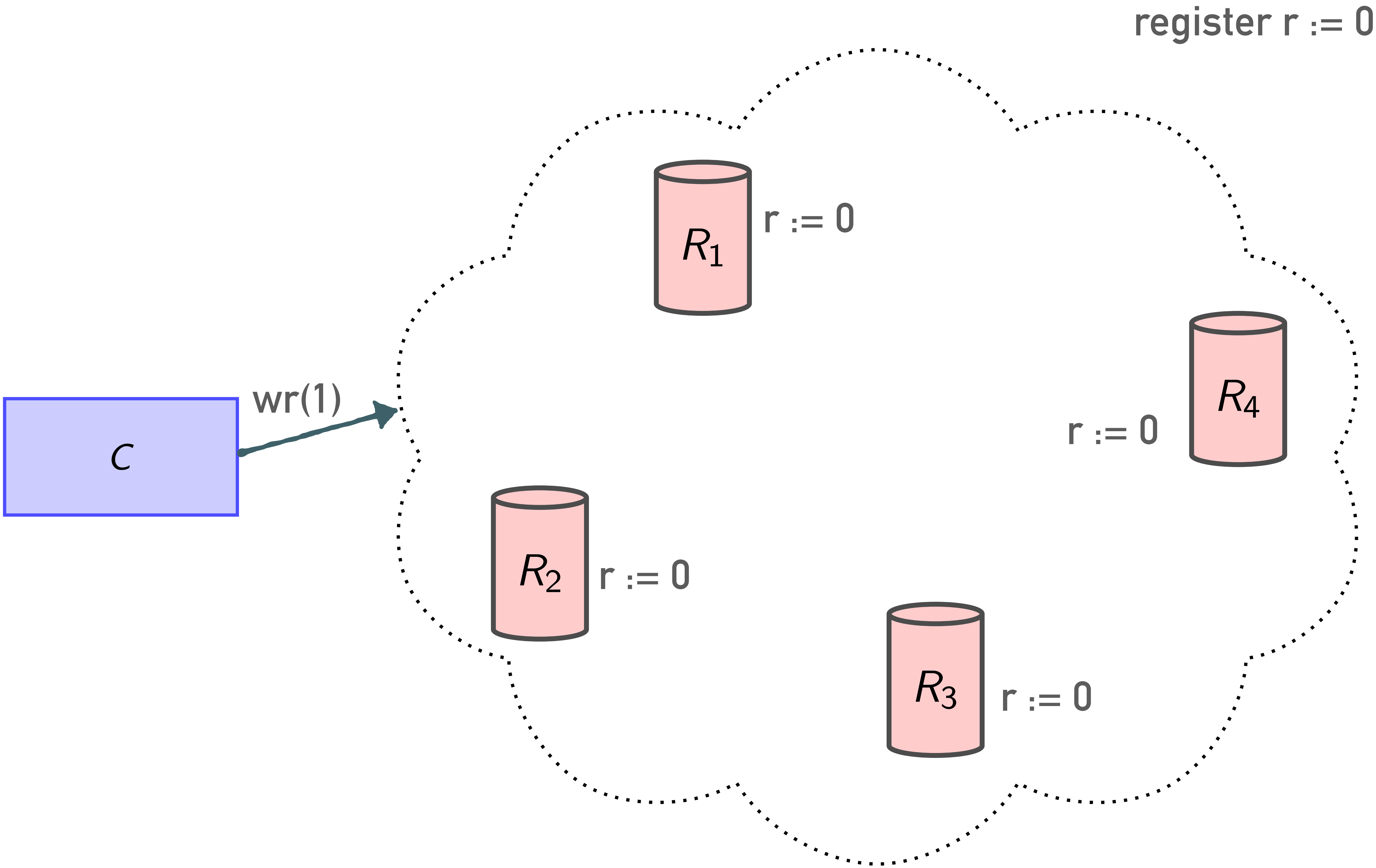
Replicated Data Types (RDTs)

- Suitable abstractions to deal with replication
- As customary, we are interested in the
  - *specification,*
  - *implementation, and*
  - *checking of implementation correctness*
- Goal: to frame these notions in an algebraic setting

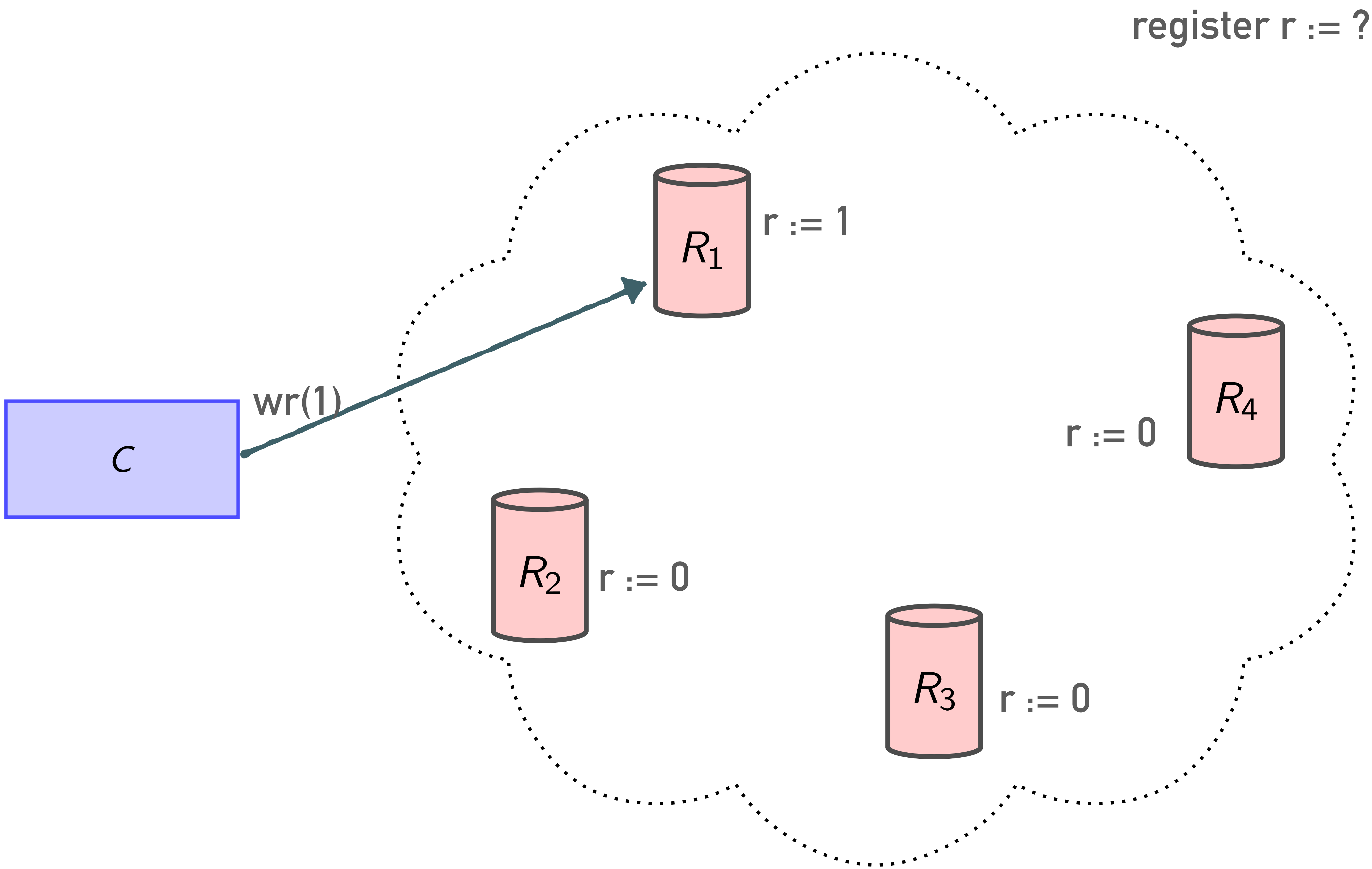
A Replicated Register



A Replicated Register

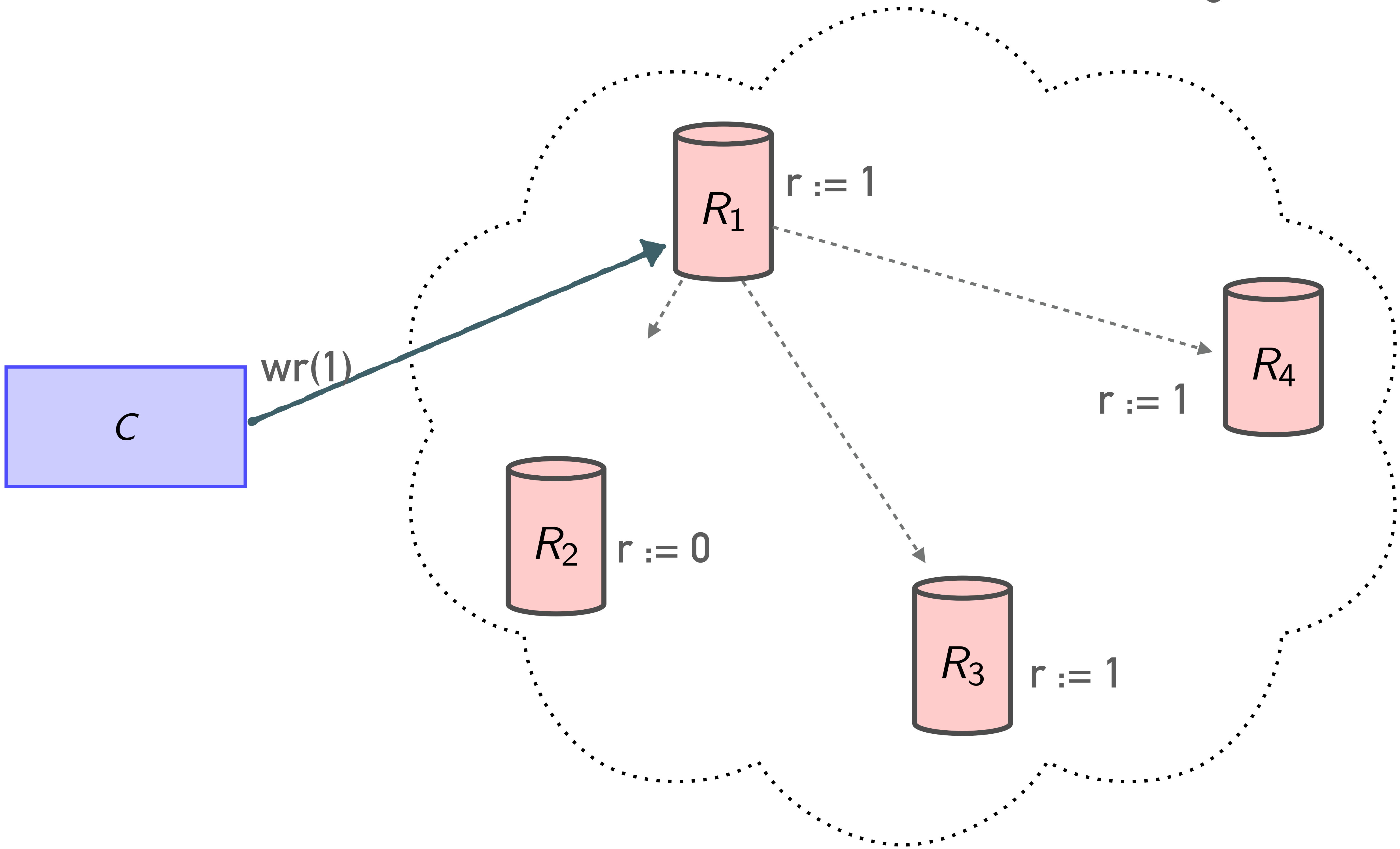


A Replicated Register

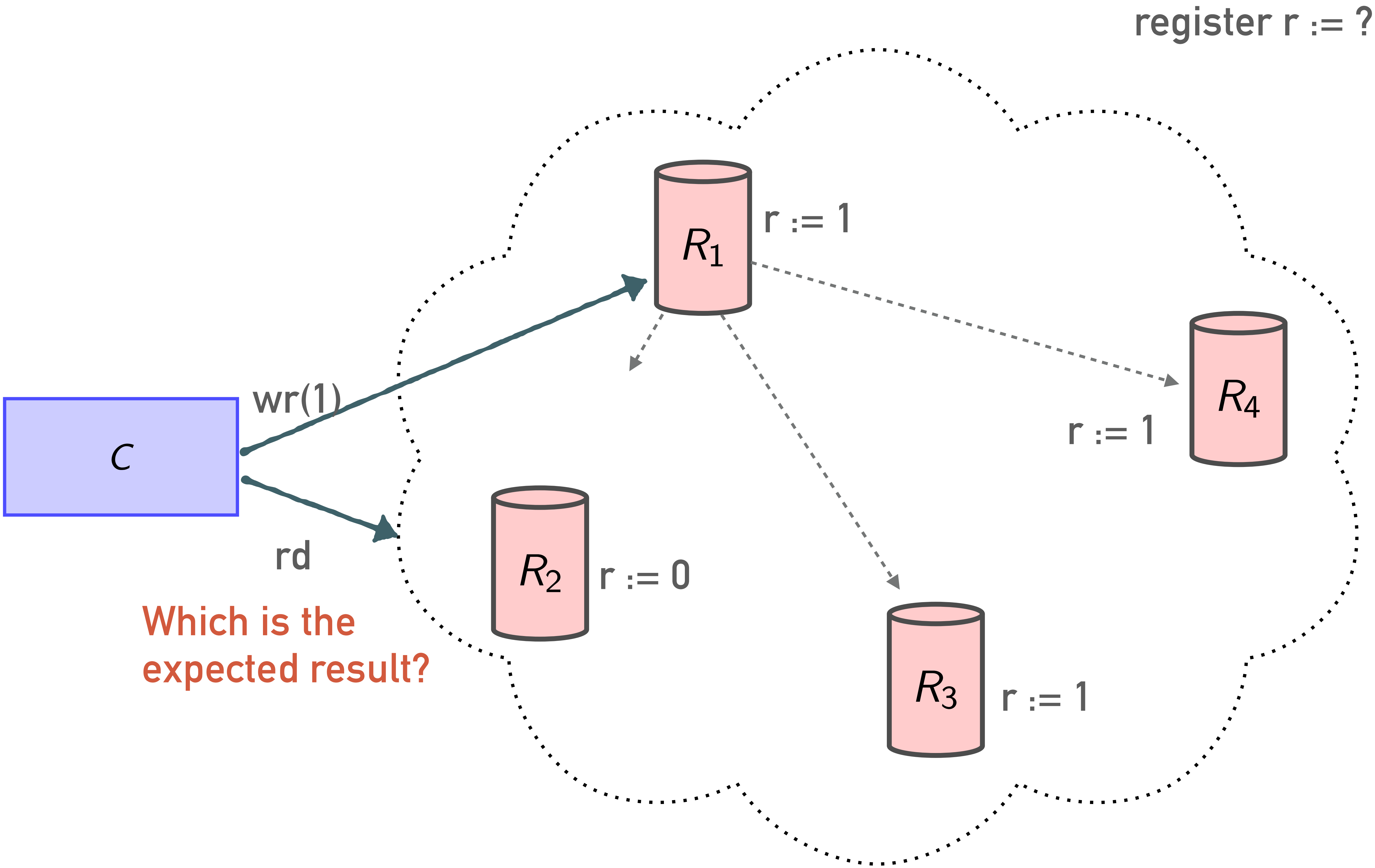


A Replicated Register

register r := ?



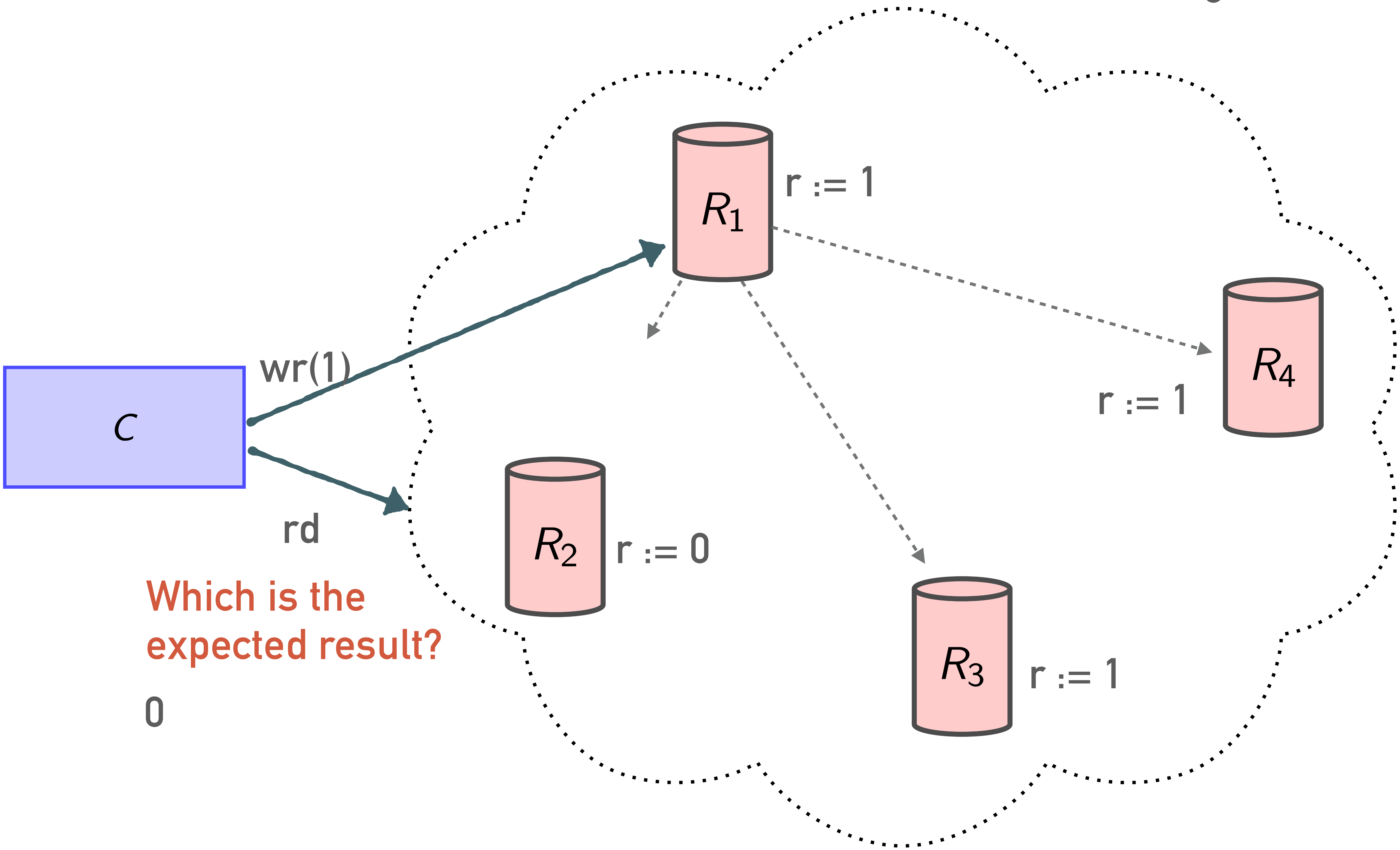
A Replicated Register



Which is the expected result?

A Replicated Register

register r := ?

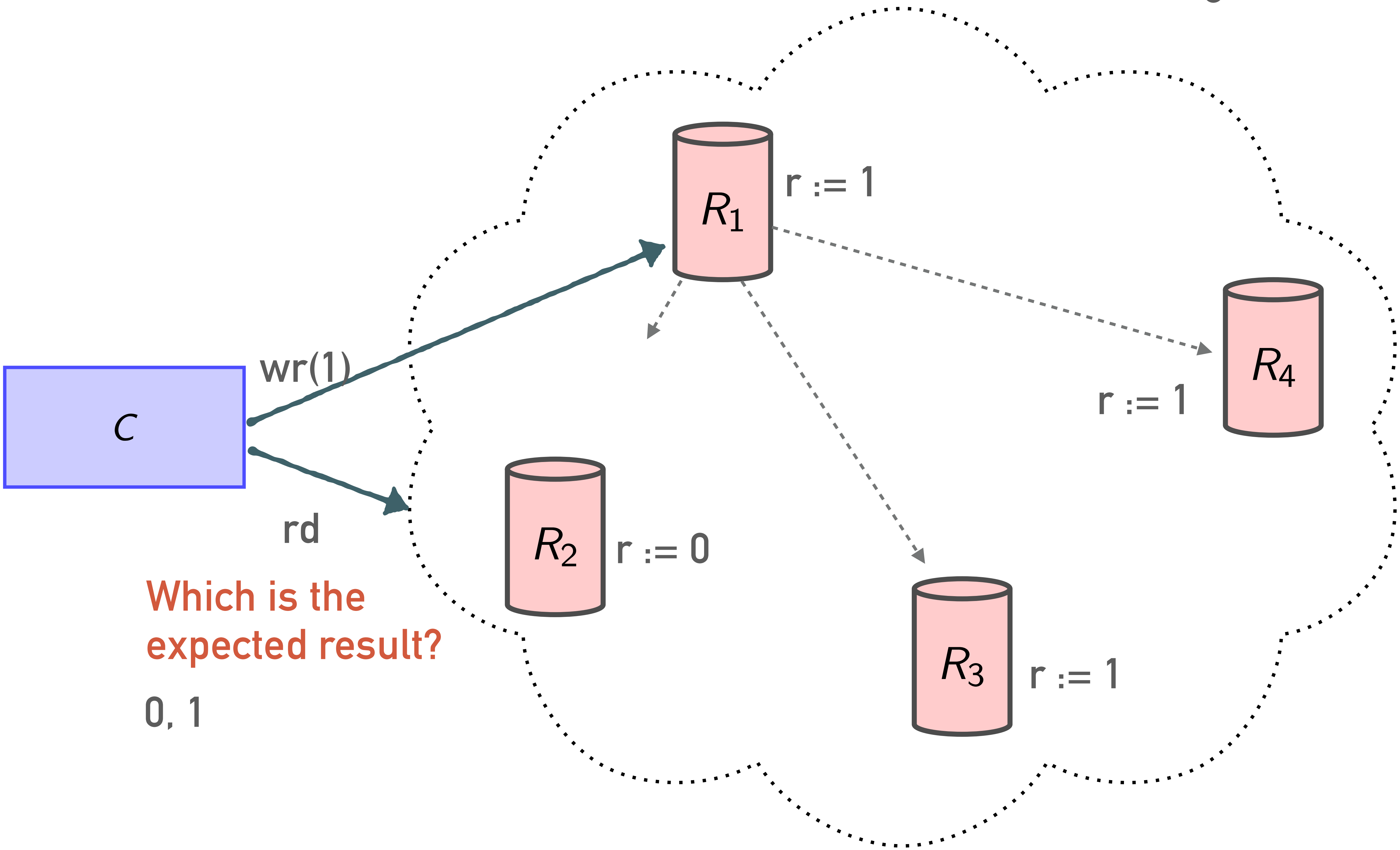


Which is the expected result?

0

A Replicated Register

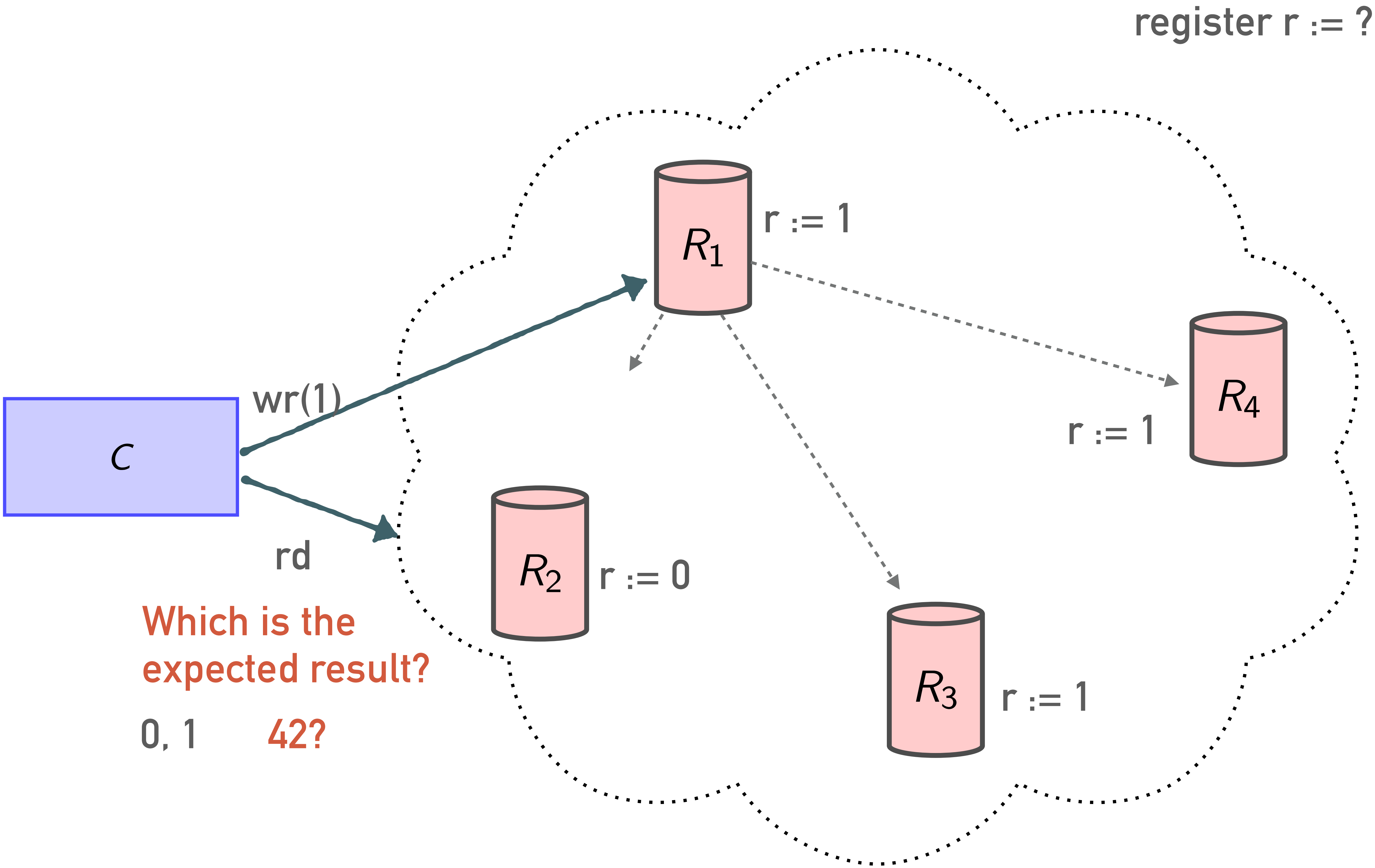
register r := ?



Which is the expected result?

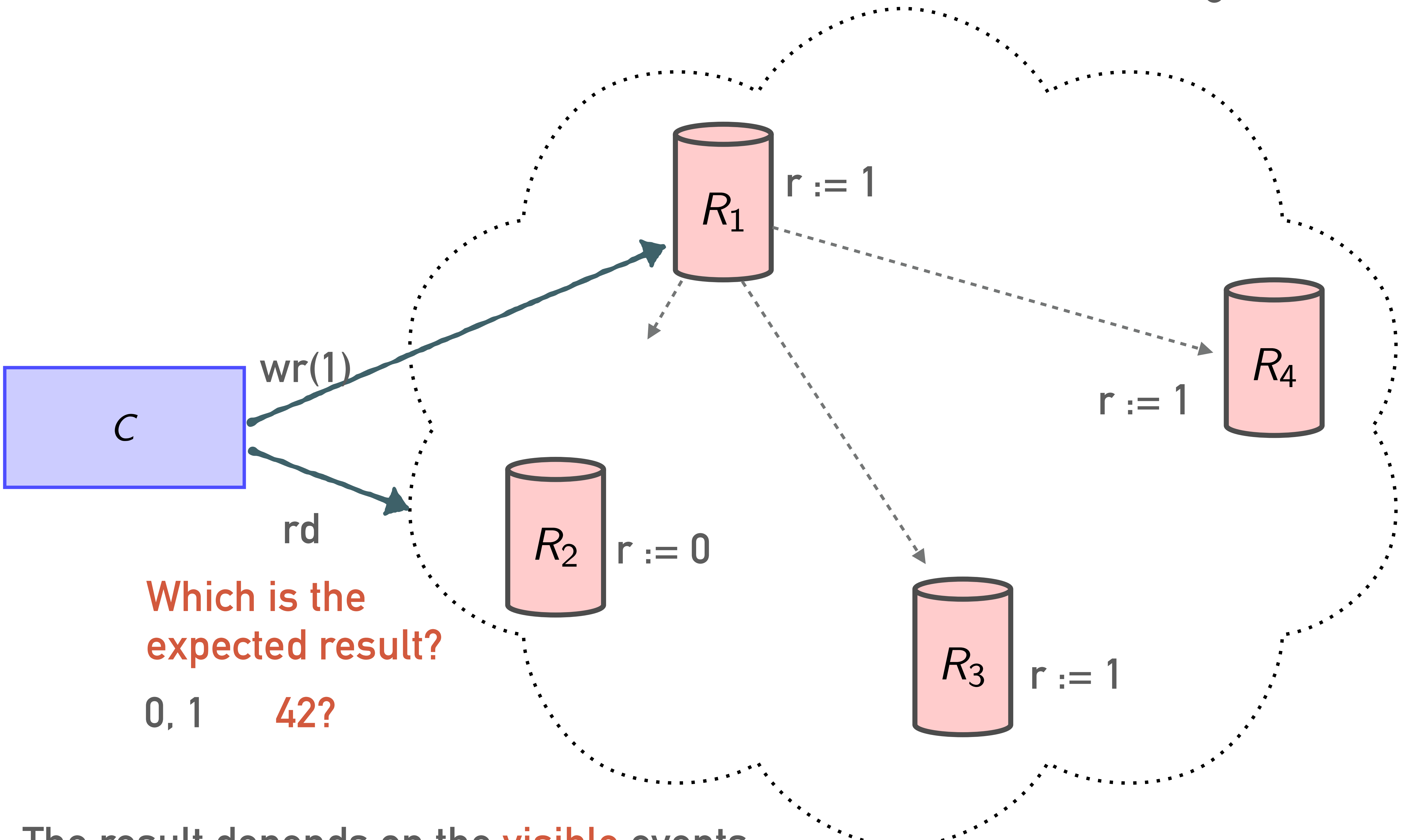
0, 1

A Replicated Register



A Replicated Register

register r := ?

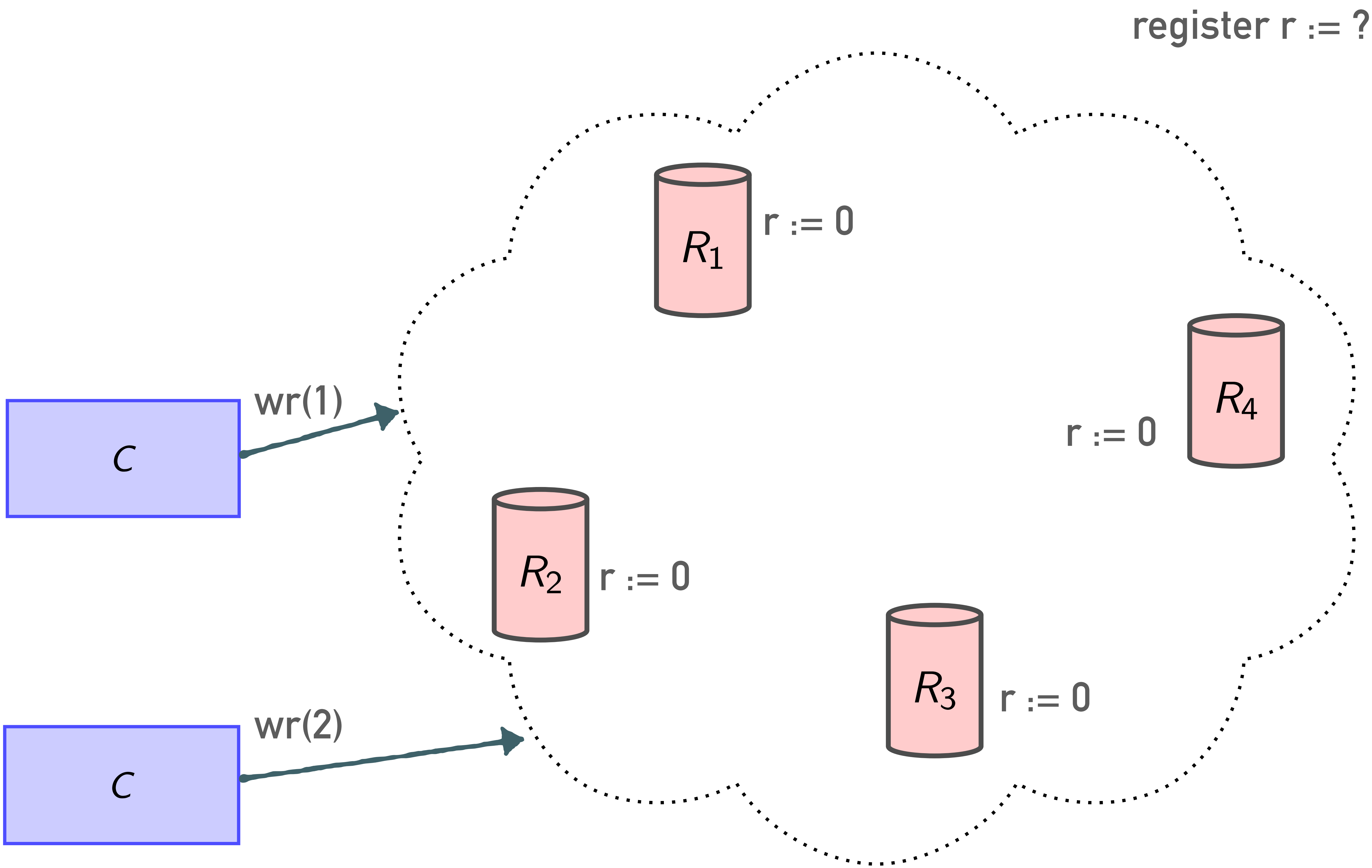


Which is the expected result?

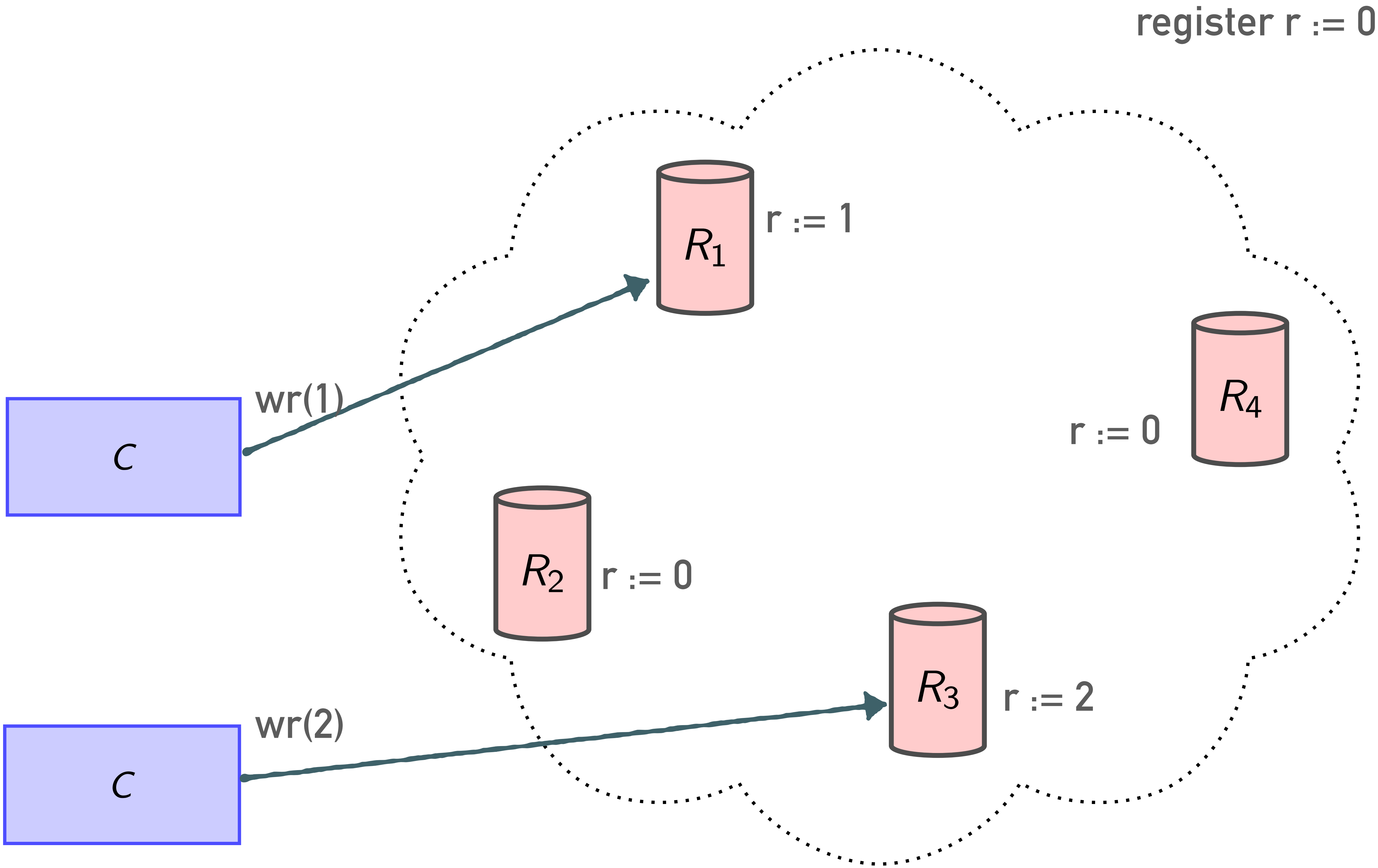
0, 1 42?

The result depends on the visible events

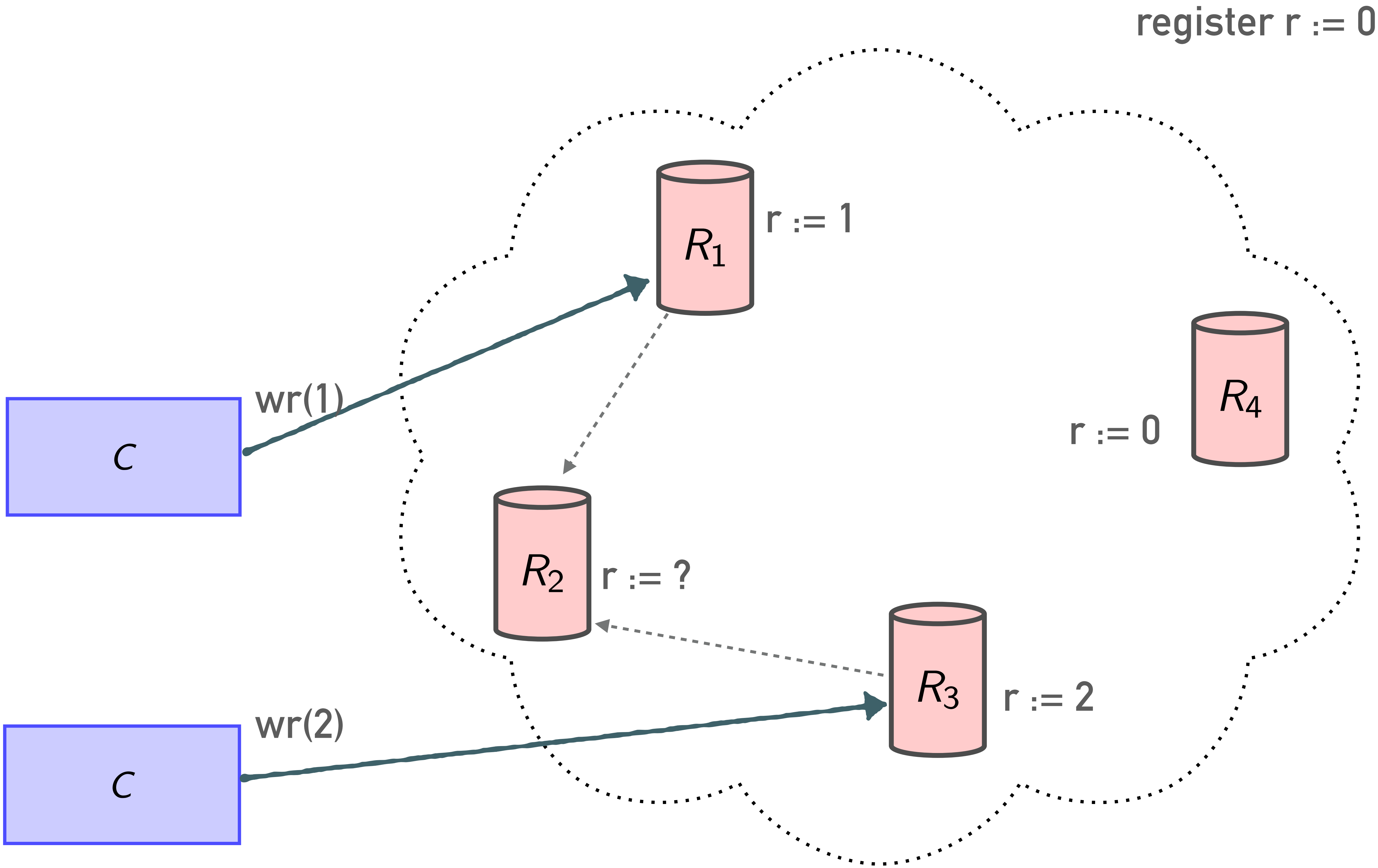
A Replicated Register



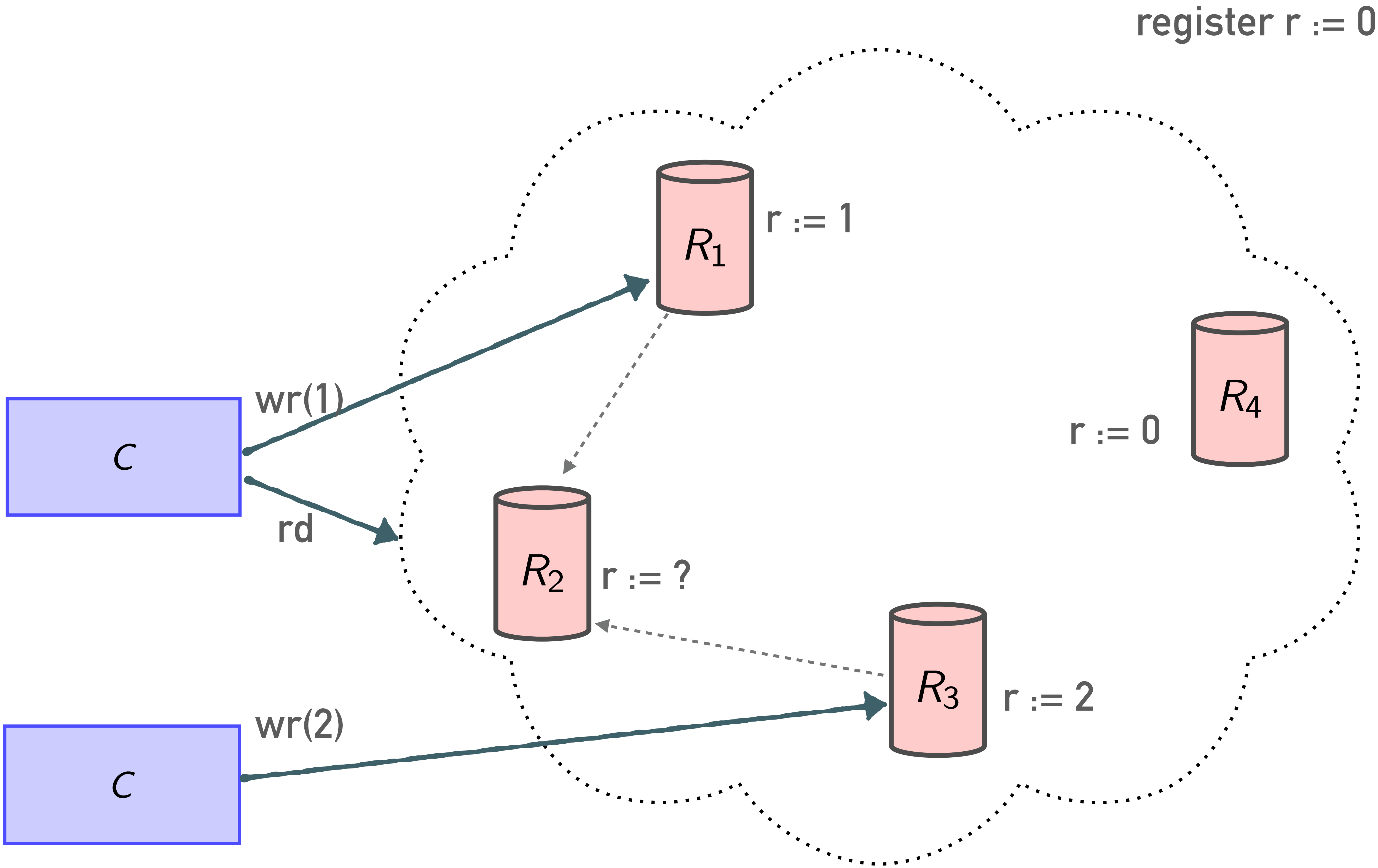
A Replicated Register



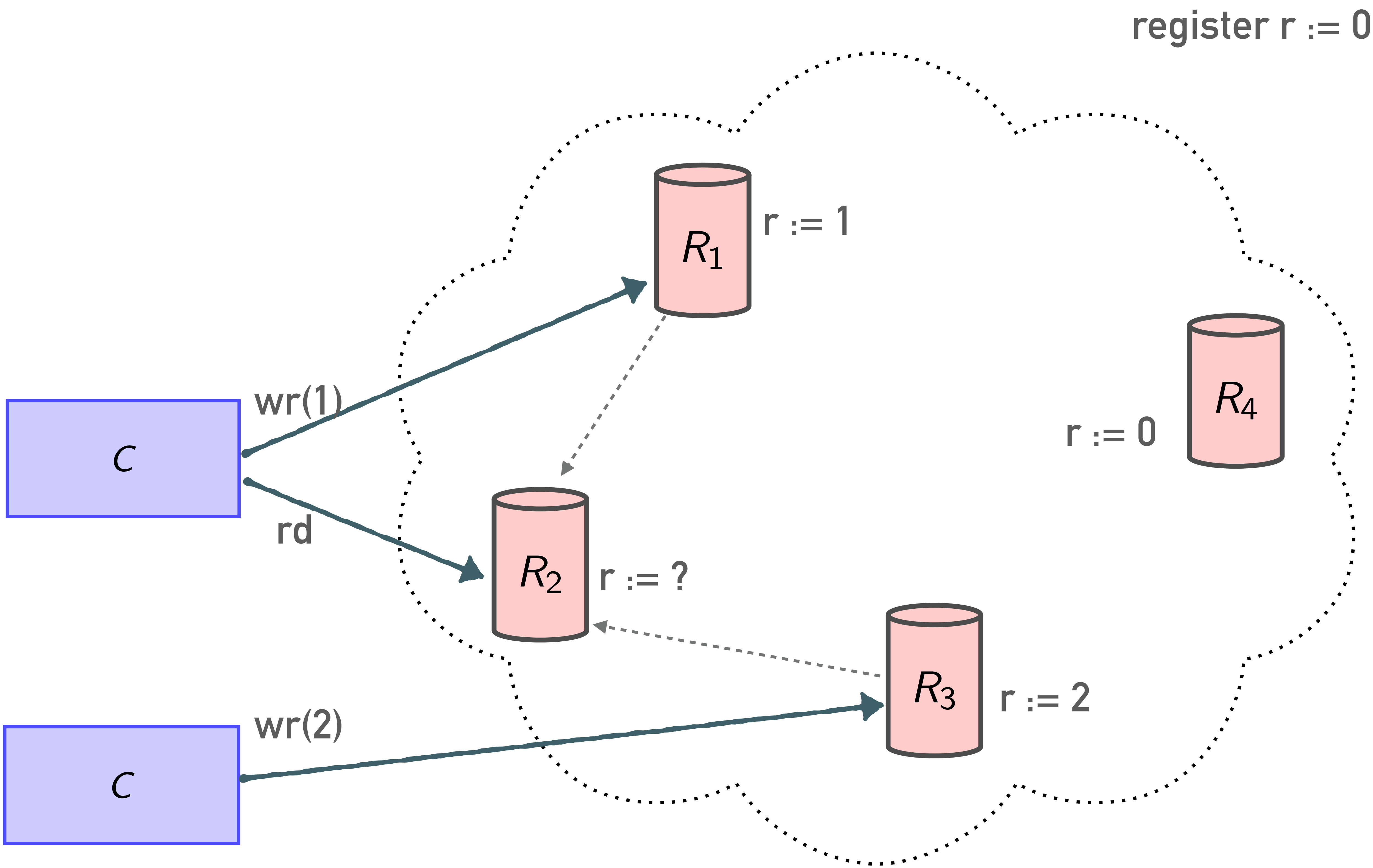
A Replicated Register



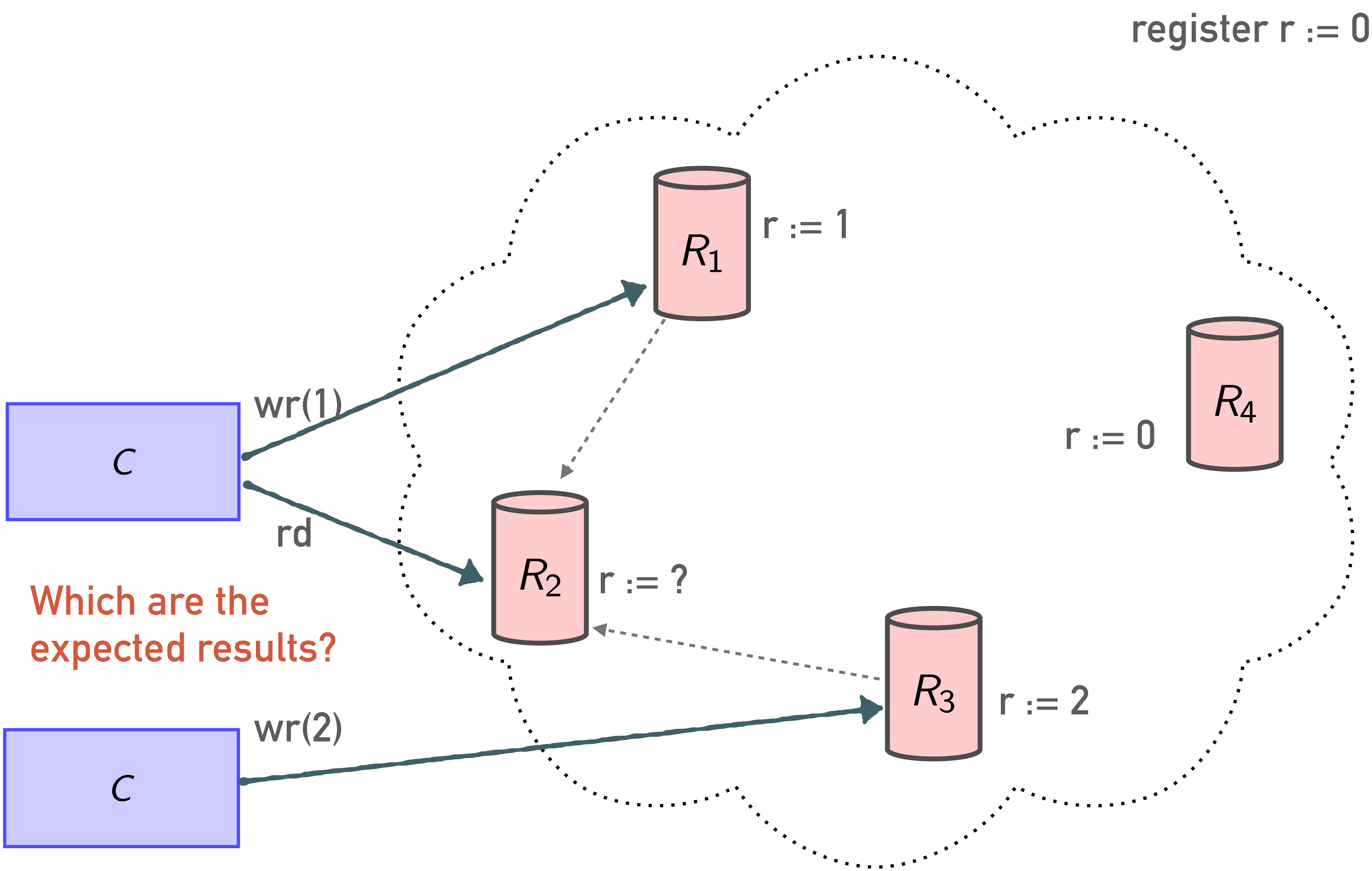
A Replicated Register



A Replicated Register



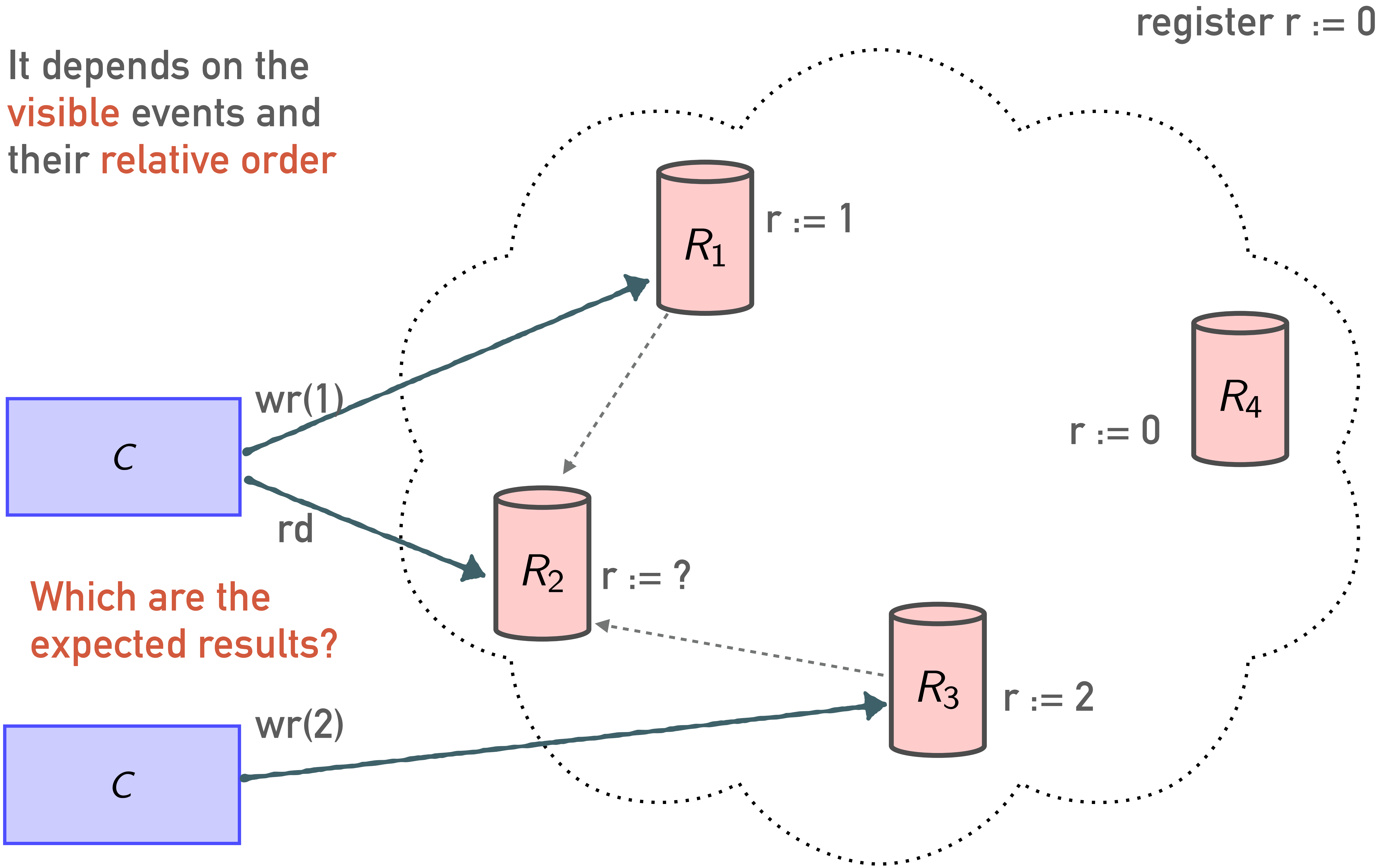
A Replicated Register



A Replicated Register

It depends on the visible events and their relative order

Which are the expected results?



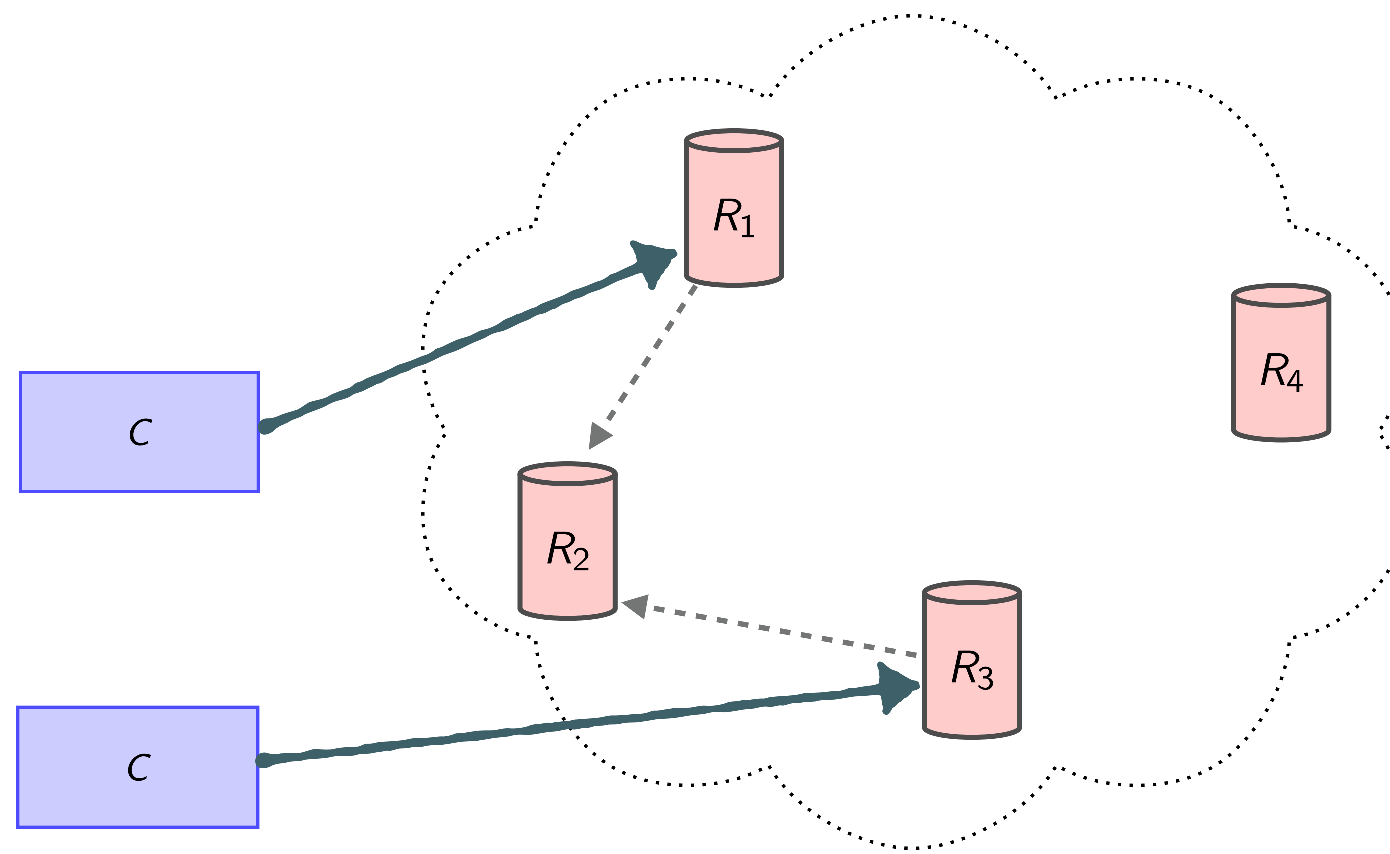
Specifying RDTs… classically

- op : VIS × ARB → RVAL
  - **VIS**ibility: A partial order of operations over a replica
  - **ARB**itration: A total order of such operations
  - **R**eturn **VAL**ue: The value returned by the last operation

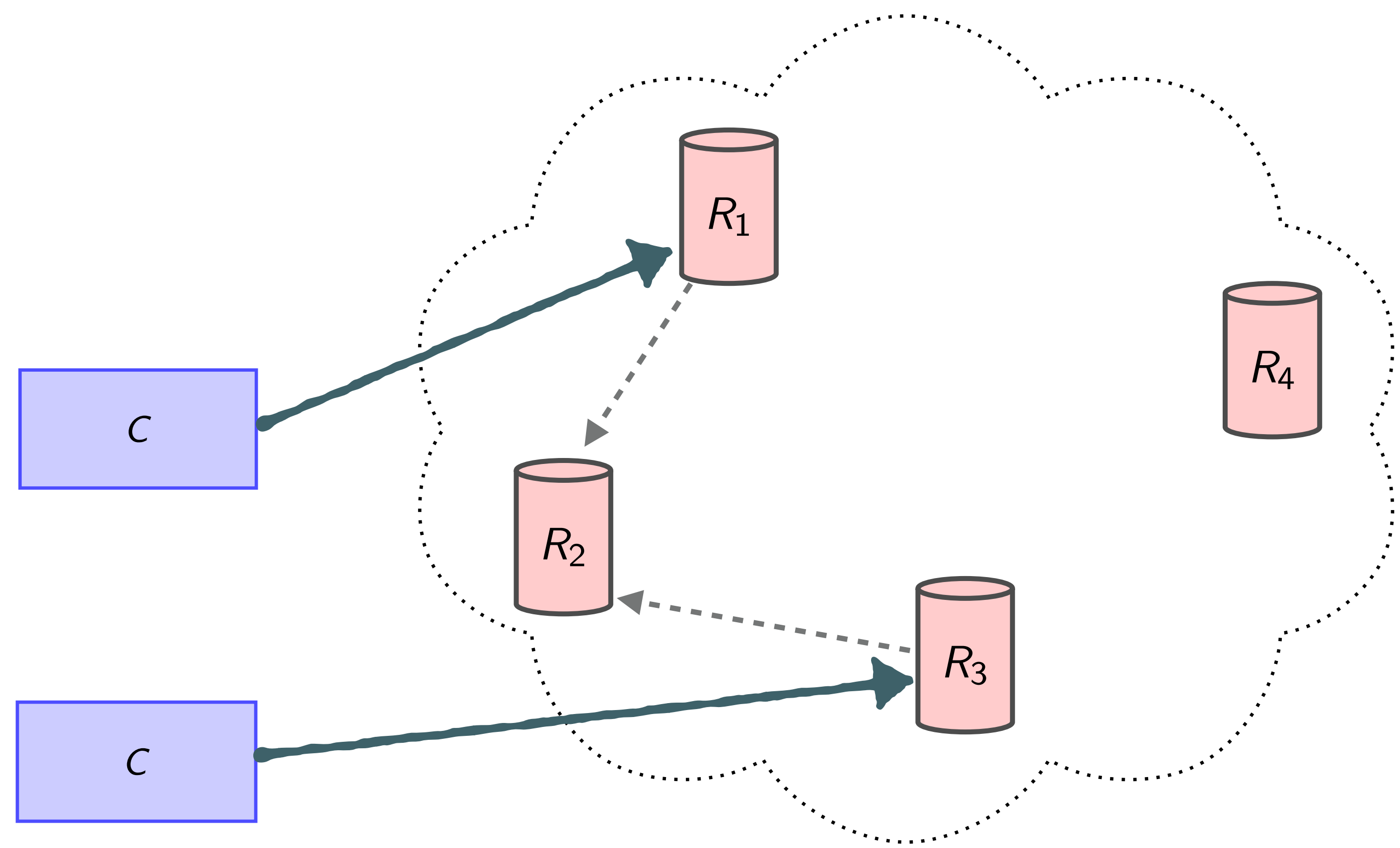
[BURCKHARDT, GOTSMAN, YANG, ZAWIRSKI 2015]

A replicated register

- Two operations
  - rd(_,_) = ?
  - wr(k)(_,_) = ok



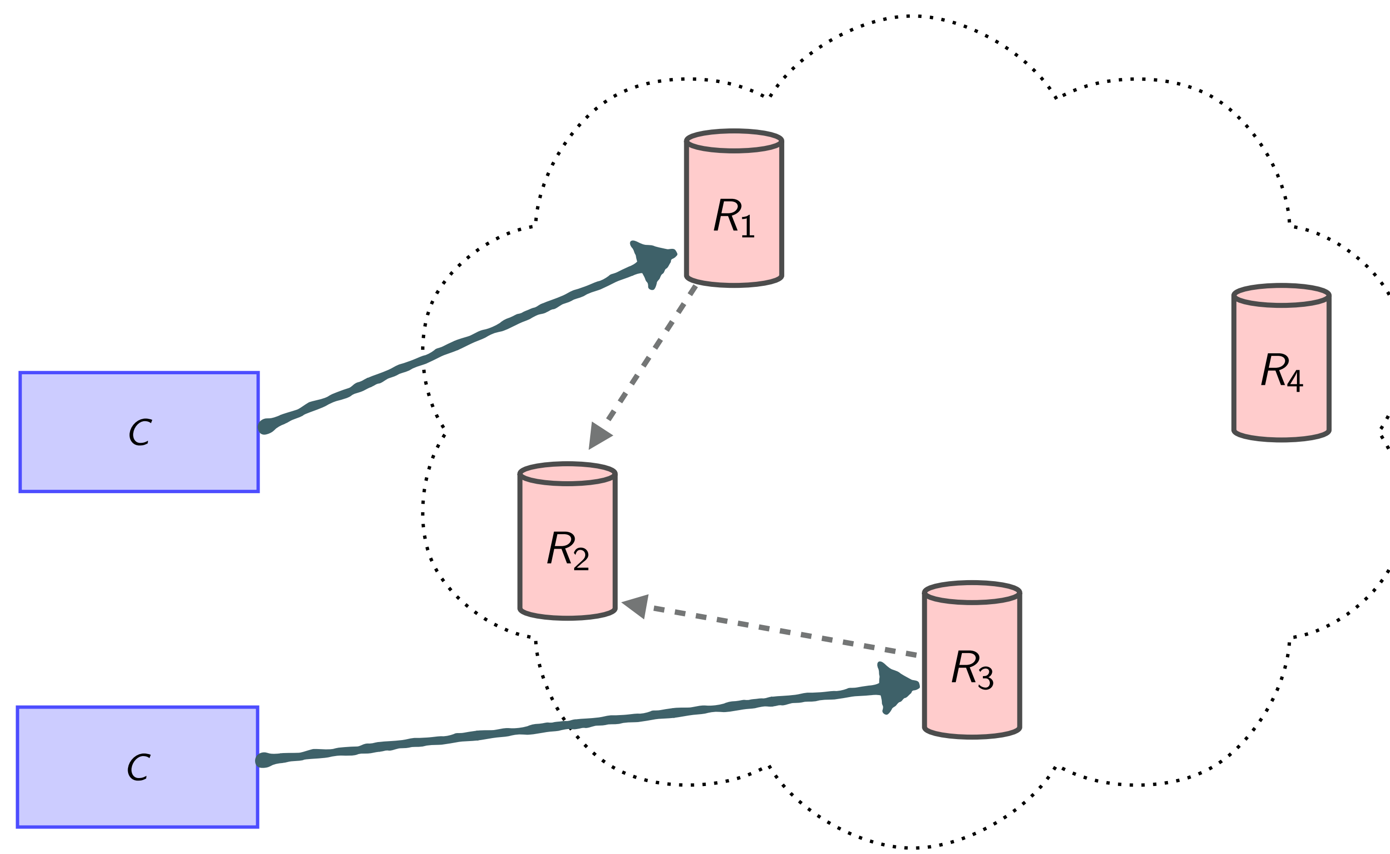
A replicated register



A replicated register

$\mathtt{wr}(1)$ $\mathtt{wr}(2)$

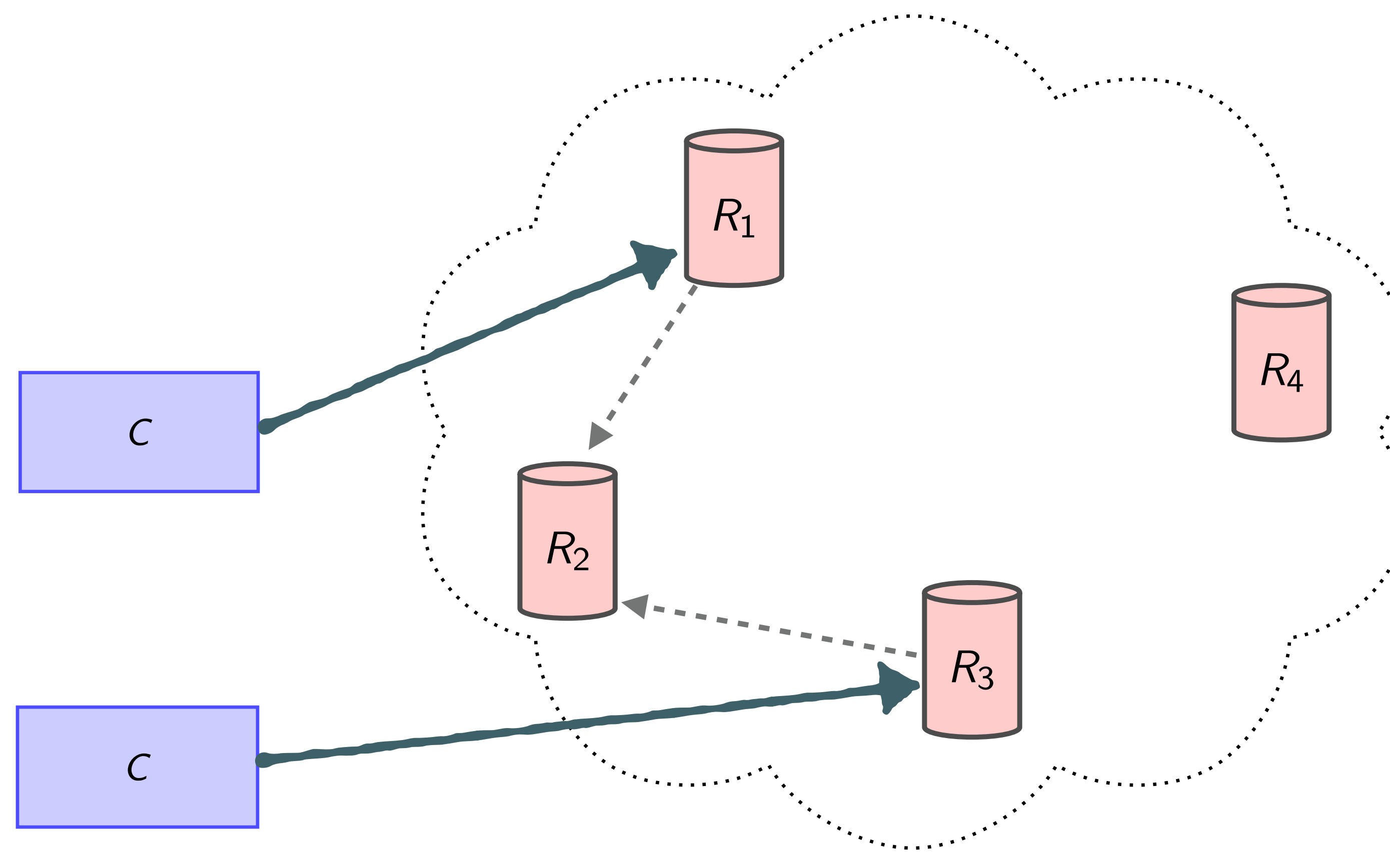
VISibility



A replicated register

wr(1) wr(2)

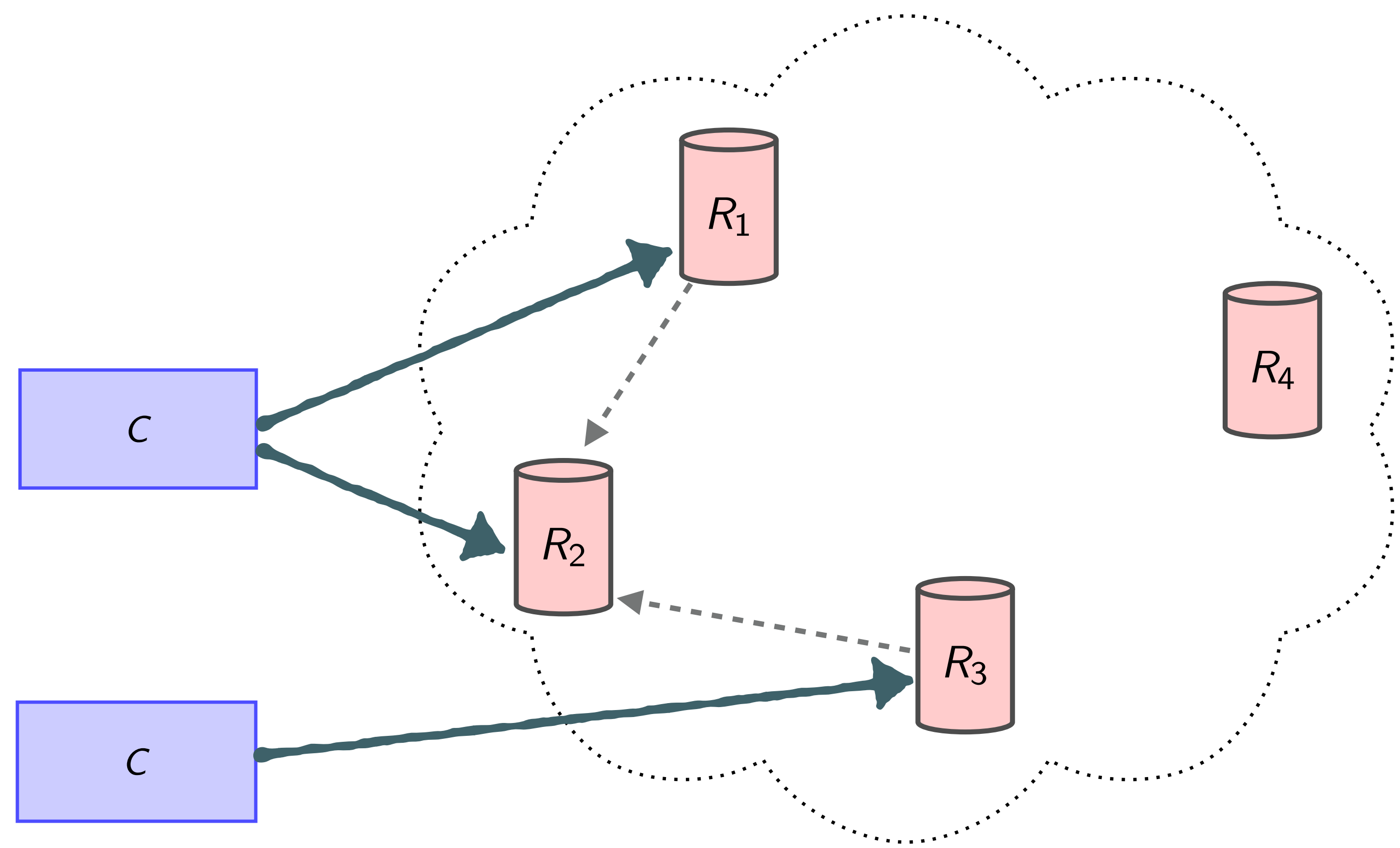
VISibility



wr(1)
|
wr(2)

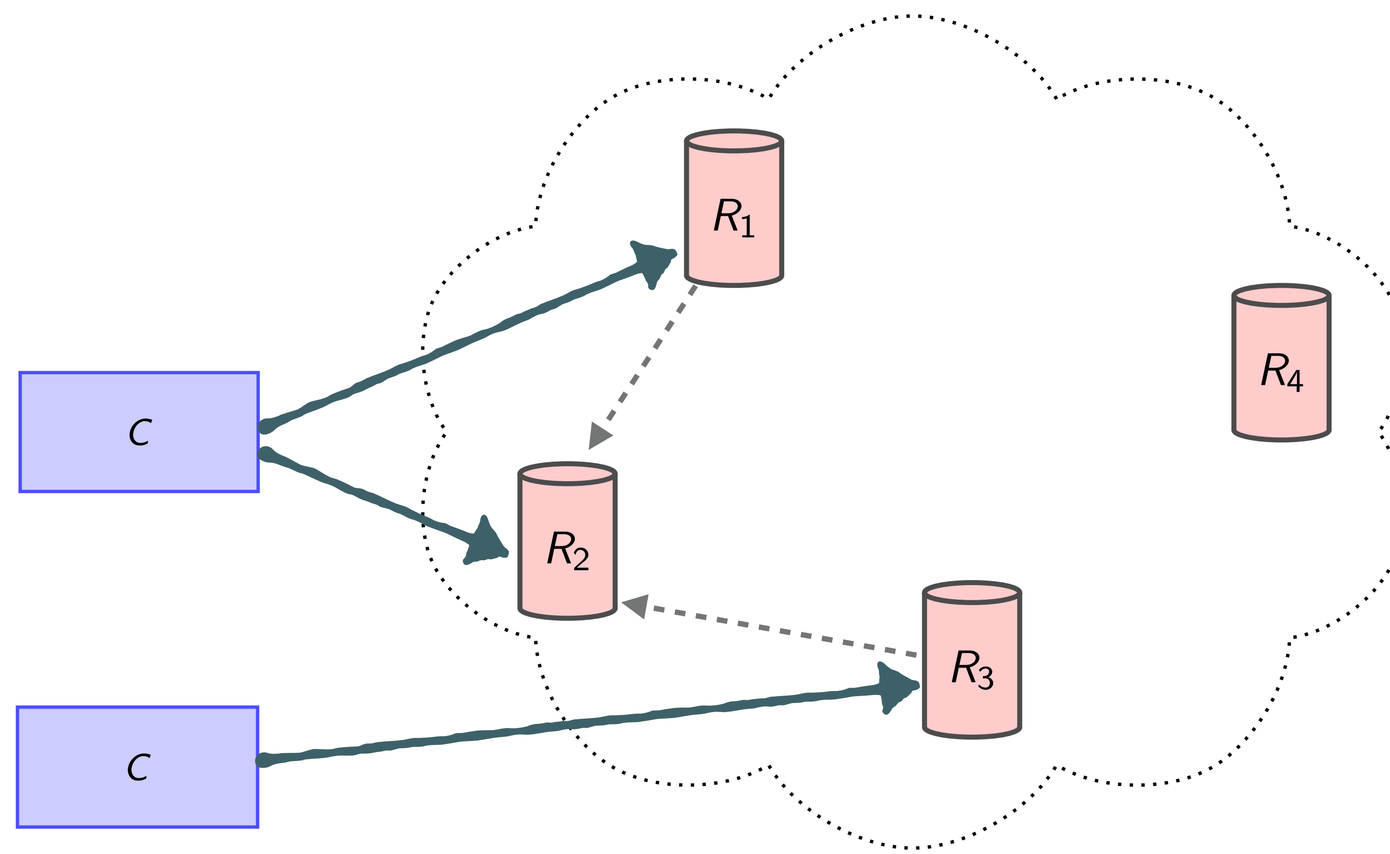
ARBitration

A replicated register



A replicated register

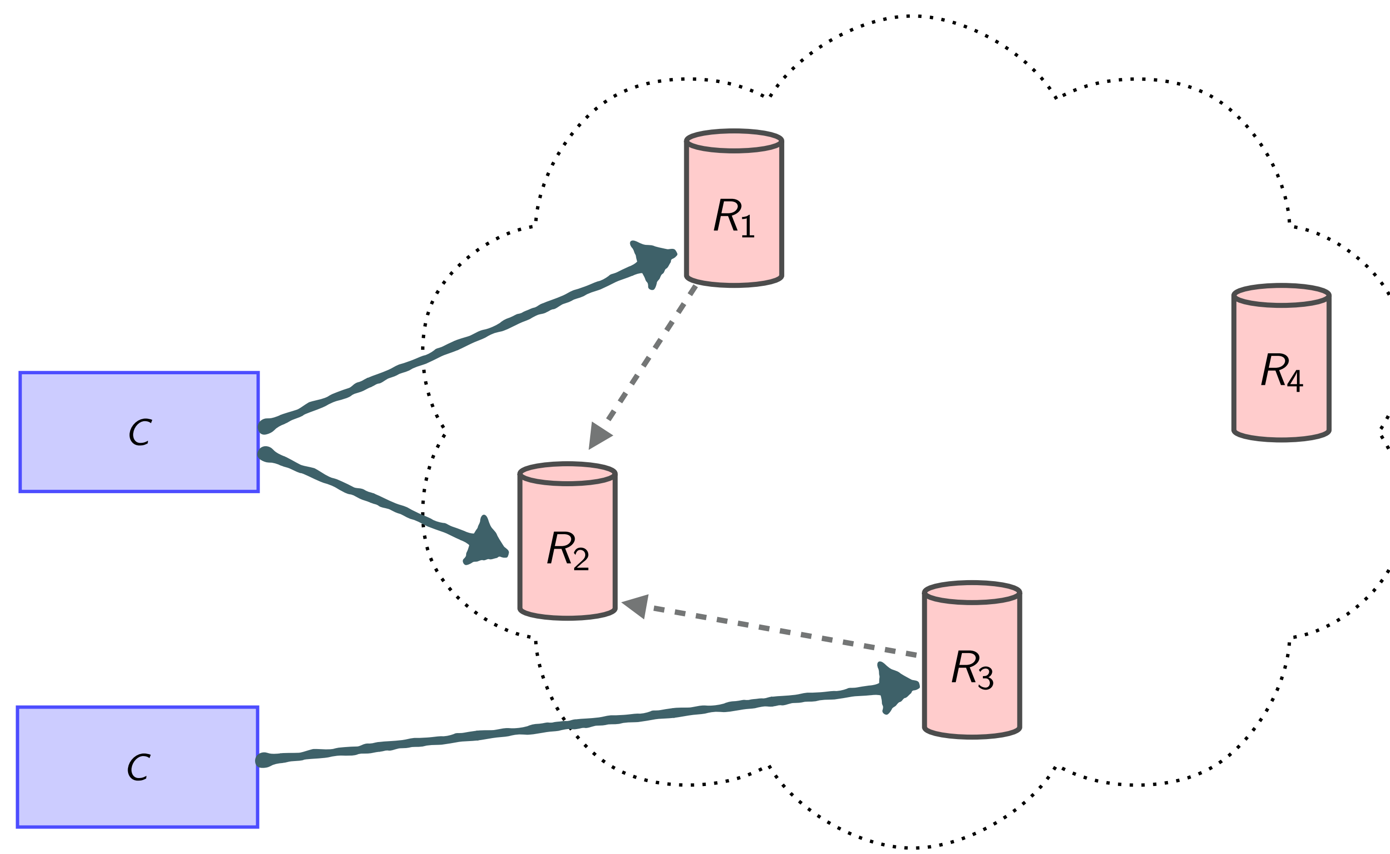
$$\mathrm{rd}\left(\mathtt{wr}(1) \quad \mathtt{wr}(2) \qquad \begin{array}{c}\mathtt{wr}(1)\\ | \\ \mathtt{wr}(2)\end{array}\right) = 2$$



A replicated register

$$\mathrm{rd}\left(\mathtt{wr}(1) \quad \mathtt{wr}(2) \qquad \begin{array}{c}\mathtt{wr}(1)\\ |\\ \mathtt{wr}(2)\end{array}\right) = 2$$

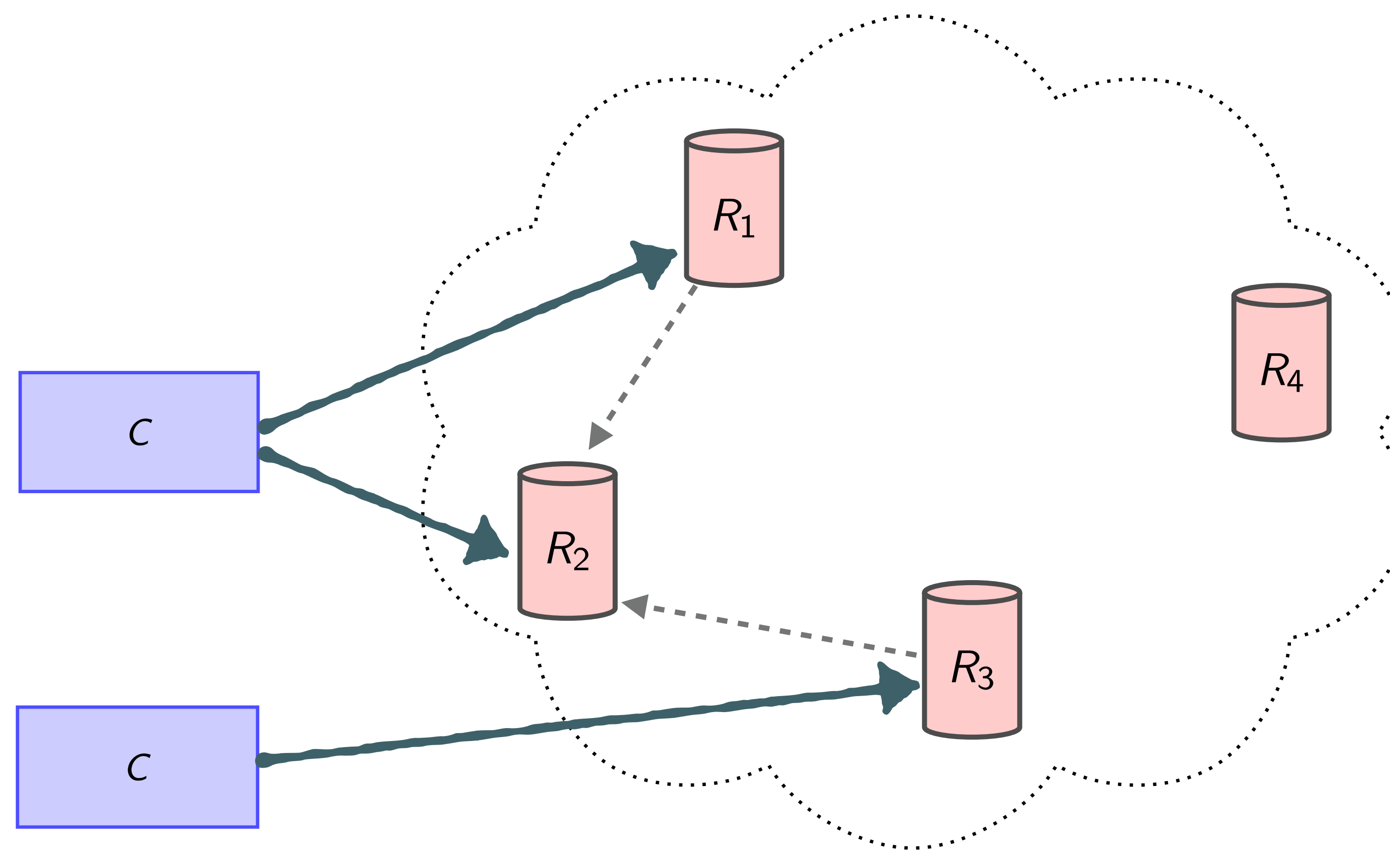
$$\mathrm{rd}\left(\mathtt{wr}(1) \quad \mathtt{wr}(2) \qquad \begin{array}{c}\mathtt{wr}(2)\\ |\\ \mathtt{wr}(1)\end{array}\right) = 1$$



A replicated register

$$\mathrm{rd}\left(\mathtt{wr}(1) \quad \mathtt{wr}(2) \qquad \begin{matrix} \mathtt{wr}(1) \\ | \\ \mathtt{wr}(2) \end{matrix} \right) = 2$$

$$\mathrm{rd}\left(\mathtt{wr}(1) \quad \mathtt{wr}(2) \qquad \begin{matrix} \mathtt{wr}(2) \\ | \\ \mathtt{wr}(1) \end{matrix} \right) = 1$$



Last-write-wins

Implementing RDTs

- Implementing RTDs means…
  - to provide an asynchronous communication mechanism among replicas
  - to ensure its compatibility wrt. the behaviour of the operations
  - to ensure global properties (e.g. eventual convergence of replicas) are preserved
- But first…
  - Is it possible to get an algebraic presentation of RTDs?
  - Is there any implicit assumption on the arbitrations?
  - Are RDTs compositional? That is, are arbitrations of larger visibility orders explained in terms of smaller ones?

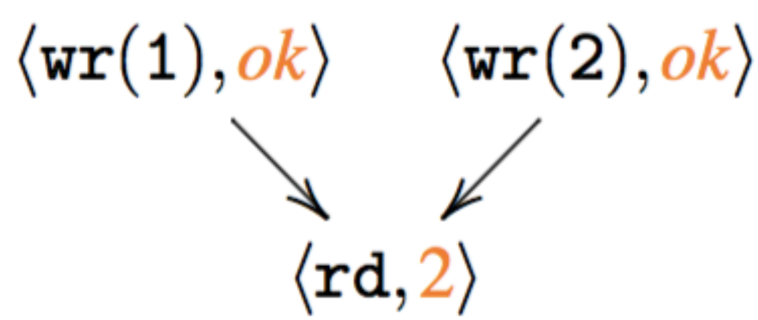
Internalising Values

$$\langle \texttt{wr}(1), ok \rangle \quad \langle \texttt{wr}(2), ok \rangle$$

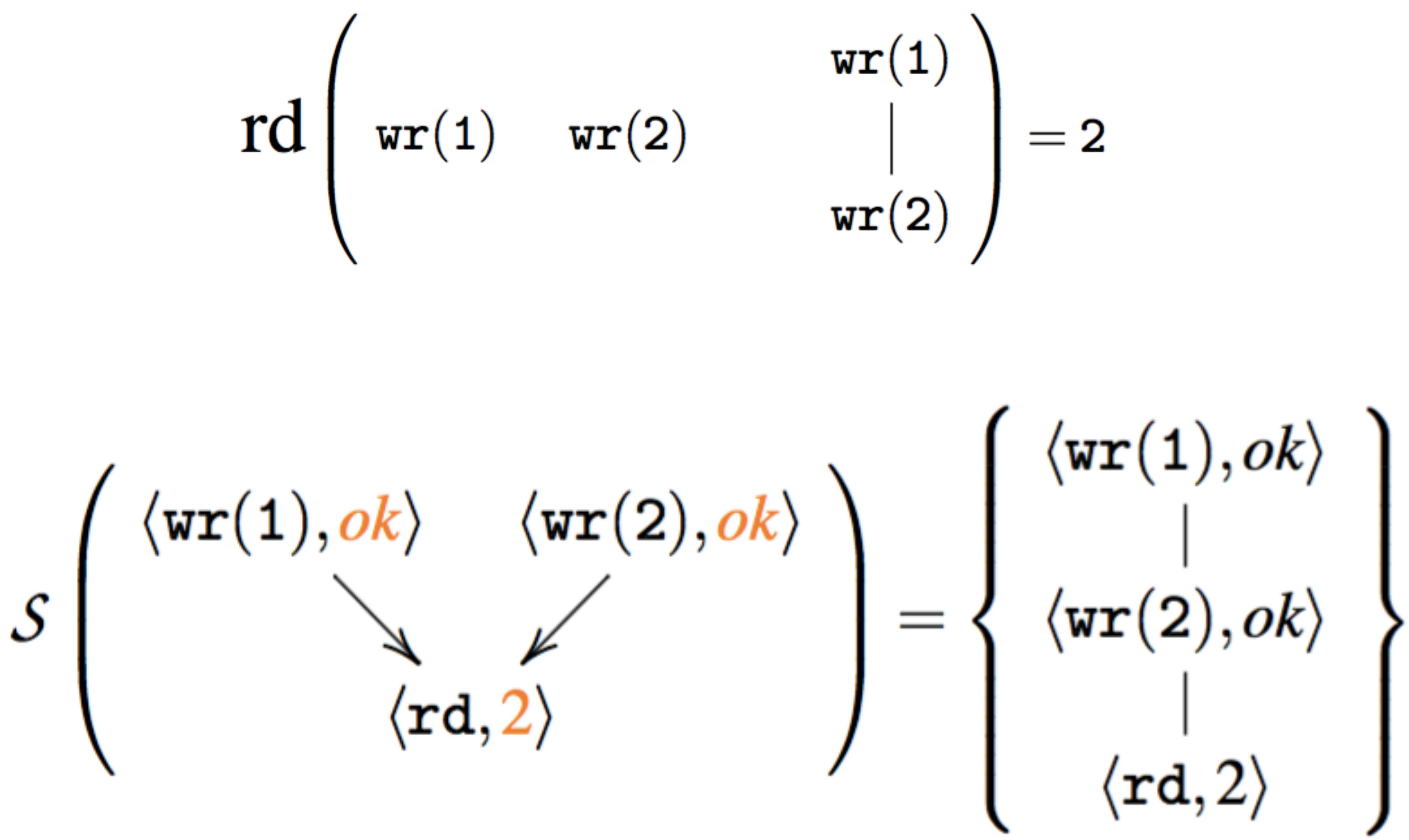
Internalising Values

$$\langle \texttt{wr}(1), ok \rangle \qquad \langle \texttt{wr}(2), ok \rangle$$

$$\mathrm{rd}\left(\texttt{wr}(1) \quad \texttt{wr}(2) \qquad \begin{array}{c} \texttt{wr}(1) \\ | \\ \texttt{wr}(2) \end{array} \right) = 2$$

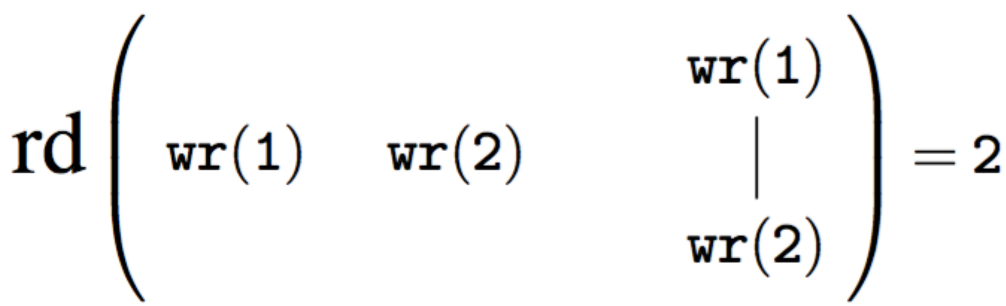


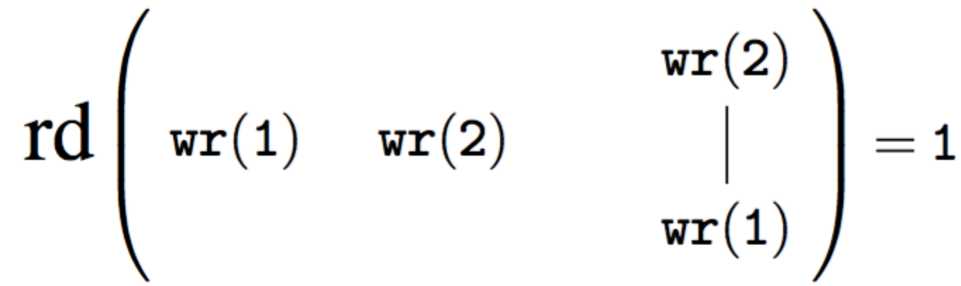
Internalising Values

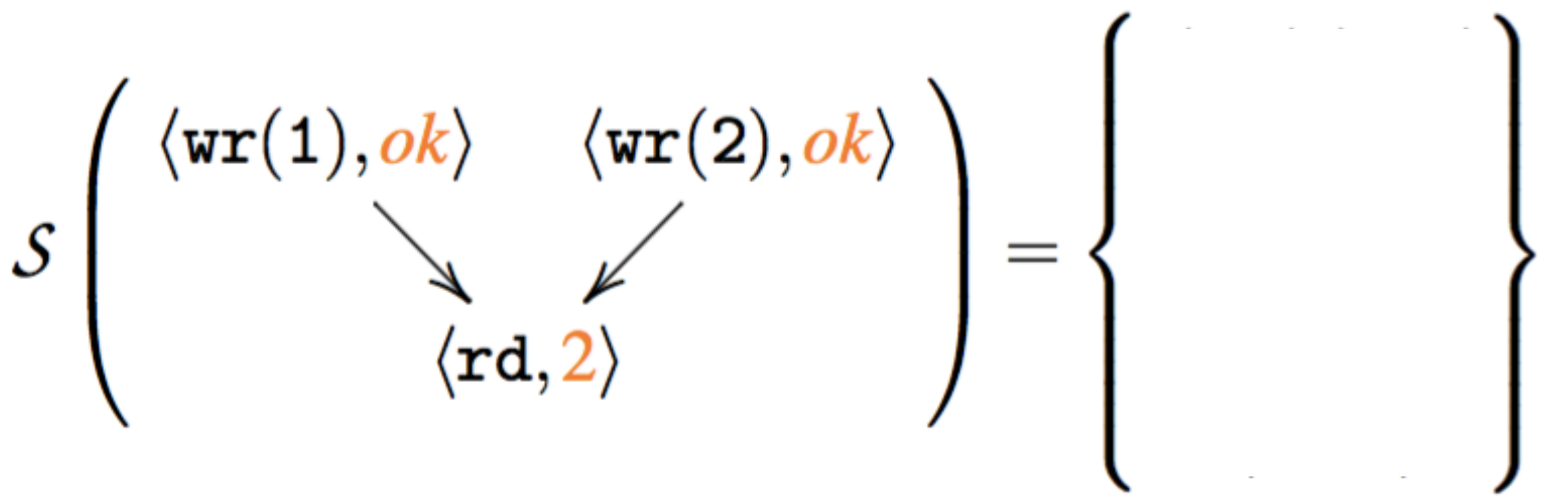


A specification goes from configurations to sets of arbitrations

Internalising Values







A specification goes from configurations to sets of arbitrations

Recovering Rtds: Saturation

$$\mathrm{rd}\left(\texttt{wr}(1) \quad \texttt{wr}(2) \qquad \begin{array}{c} \texttt{wr}(1) \\ | \\ \texttt{wr}(2) \end{array} \right) = 2$$

A specification goes from configurations to sets of arbitrations

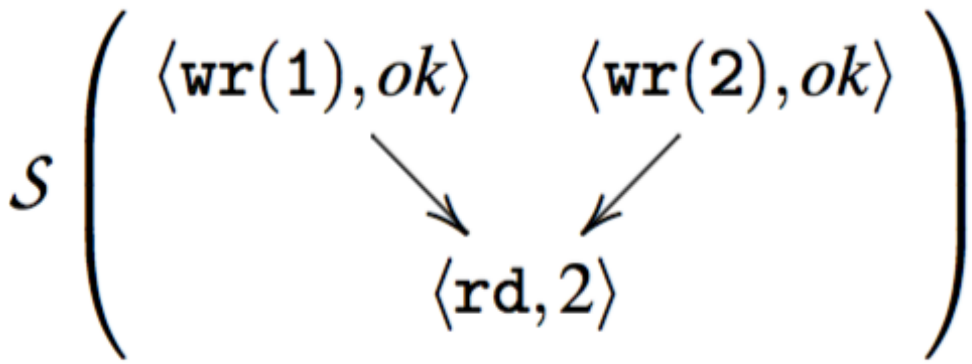
Recovering Rtds: Saturation

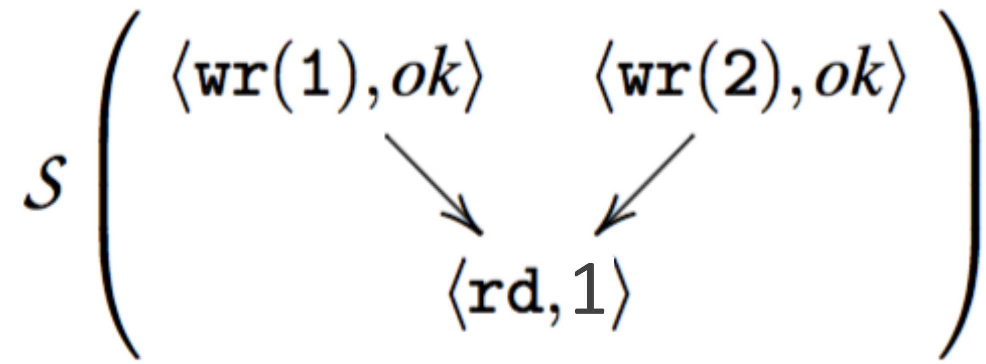
$$\mathrm{rd}\left(\begin{array}{ccc} & & \mathtt{wr}(1) \\ \mathtt{wr}(1) & \mathtt{wr}(2) & | \\ & & \mathtt{wr}(2) \end{array}\right) = 2$$

$$\mathcal{S}\left(\begin{array}{ccc} \langle \mathtt{wr}(1), ok\rangle & & \langle \mathtt{wr}(2), ok\rangle \\ & \searrow \quad \swarrow & \\ & \langle \mathtt{rd}, 2\rangle & \end{array}\right) = \left\{\begin{array}{ccc} \langle \mathtt{wr}(1), ok\rangle & \langle \mathtt{wr}(1), ok\rangle & \langle \mathtt{rd}, 2\rangle \\ | & | & | \\ \langle \mathtt{wr}(2), ok\rangle & \langle \mathtt{rd}, 2\rangle & \langle \mathtt{wr}(1), ok\rangle \\ | & | & | \\ \langle \mathtt{rd}, 2\rangle & \langle \mathtt{wr}(2), ok\rangle & \langle \mathtt{wr}(2), ok\rangle \end{array}\right\}$$

A specification goes from configurations to sets of arbitrations

Recovering Rtds: Determinism





value deterministic: empty intersection after removing last event

deterministic: empty intersection after forgetting also the value

(Classic) RTDs have chosen the second path
(thus e.g. forbidding write failures)

Recovering Rtds: Coherence

$$\begin{array}{c}\langle \mathtt{wr}(1),\mathtt{ok}\rangle\\ | \\ \langle \mathtt{wr}(2),\mathtt{ok}\rangle \\ | \\ \langle \mathtt{rd},2\rangle\end{array} \otimes \begin{array}{c}\langle \mathtt{wr}(2),\mathtt{ok}\rangle\\ | \\ \langle \mathtt{wr}(3),\mathtt{ok}\rangle\end{array} = \left\{ \begin{array}{cc}\langle \mathtt{wr}(1),\mathtt{ok}\rangle & \langle \mathtt{wr}(1),\mathtt{ok}\rangle\\ | & | \\ \langle \mathtt{wr}(2),\mathtt{ok}\rangle & \langle \mathtt{wr}(2),\mathtt{ok}\rangle\\ | & | \\ \langle \mathtt{rd},2\rangle & \langle \mathtt{wr}(3),\mathtt{ok}\rangle\\ | & | \\ \langle \mathtt{wr}(3),\mathtt{ok}\rangle & \langle \mathtt{rd},2\rangle\end{array}\right\}$$

Admissible arbitrations never increase when extending visibility

Recovering Rtds: Coherence

$$
\begin{array}{c}
\langle \mathtt{wr}(1), \mathtt{ok} \rangle \\
| \\
\langle \mathtt{wr}(2), \mathtt{ok} \rangle \\
| \\
\langle \mathtt{rd}, 2 \rangle
\end{array}
\otimes
\begin{array}{c}
\langle \mathtt{wr}(2), \mathtt{ok} \rangle \\
| \\
\langle \mathtt{wr}(3), \mathtt{ok} \rangle
\end{array}
=
\left\{
\begin{array}{cc}
\langle \mathtt{wr}(1), \mathtt{ok} \rangle & \langle \mathtt{wr}(1), \mathtt{ok} \rangle \\
| & | \\
\langle \mathtt{wr}(2), \mathtt{ok} \rangle & \langle \mathtt{wr}(2), \mathtt{ok} \rangle \\
| & | \\
\langle \mathtt{rd}, 2 \rangle & \langle \mathtt{wr}(3), \mathtt{ok} \rangle \\
| & | \\
\langle \mathtt{wr}(3), \mathtt{ok} \rangle & \langle \mathtt{rd}, 2 \rangle
\end{array}
\right\}
$$

$$\forall \mathtt{G}.\ \mathcal{S}(\mathtt{G}) = \bigotimes_{\mathtt{e} \in \mathcal{E}_{\mathtt{G}}} \mathcal{S}(\mathtt{G}|_{- \prec^* \mathtt{e}})$$

Admissible arbitrations never increase when extending visibility

Recovering Rtds: Main Theorem

- There is a one-to-one correspondence between RTDs and saturated, deterministic, and coherent specifications

Recovering Rtds: Main Theorem

- There is a one-to-one correspondence between RTDs and saturated, deterministic, and coherent specifications

....which is bad for RDTs!!

Recovering Rtds: Main Theorem

- There is a one-to-one correspondence between RTDs and saturated, deterministic, and coherent specifications

....which is bad for RDTs!!

saturation and especially determinism are bad!!

Recovering Rtds: Main Theorem

- There is a one-to-one correspondence between RTDs and saturated, deterministic, and coherent specifications

....which is bad for RDTs!!

saturation and especially determinism are bad!!

$$\mathcal{S}\begin{pmatrix}\langle \mathtt{inc}, \mathtt{ok}\rangle \\ \downarrow \\ \langle \mathtt{rd}, 1\rangle\end{pmatrix} = \left\{\begin{matrix}\langle \mathtt{inc}, \mathtt{ok}\rangle \\ | \\ \langle \mathtt{rd}, 1\rangle\end{matrix}\right\} \quad \mathcal{S}\begin{pmatrix}\langle \mathtt{inc}, \mathtt{fail}\rangle \\ \downarrow \\ \langle \mathtt{rd}, \bot\rangle\end{pmatrix} = \left\{\begin{matrix}\langle \mathtt{inc}, \mathtt{fail}\rangle \\ | \\ \langle \mathtt{rd}, \bot\rangle\end{matrix}\right\}$$

value-deterministic, yet not deterministic

From specifications to transitions systems

$\langle \mathtt{G}, \mathtt{P} \rangle$ $\quad$ $\mathtt{P} \in \mathcal{S}(\mathtt{G})$ $\quad$ states

From specifications to transitions systems

$\langle \mathtt{G}, \mathtt{P} \rangle \qquad \mathtt{P} \in \mathcal{S}(\mathtt{G})$ states

$\langle \mathtt{G}, \mathtt{P} \rangle \xrightarrow{\ell} \langle \mathtt{G}', \mathtt{P}' \rangle$ transitions

$\mathtt{G}' = \mathtt{G}^{\ell} \qquad \mathtt{P}'|_{\mathcal{E}_{\mathtt{G}}} = \mathtt{P}.$

From specifications to transitions systems

(COMP)

$$\frac{\langle \mathtt{G}_1\,,\, \mathtt{P}|_{\mathcal{E}_{\mathtt{G}_1}} \rangle \xrightarrow{\ell} \langle \mathtt{G}'_1, \mathtt{P}'_1 \rangle \qquad \mathtt{P}' \in \mathtt{P} \otimes \mathtt{P}'_1}{\langle \mathtt{G}_1 \sqcup \mathtt{G}_2, \mathtt{P} \rangle \xrightarrow{\ell} \langle \mathtt{G}'_1 \sqcup \mathtt{G}_2, \mathtt{P}' \rangle}$$

an abstract transition system against which to compare (by asynchronous simulation) those of actual implementations…

RDT specifications (functional style)

$$\mathbf{S} : \mathbf{G}(\mathcal{L}) \rightarrow 2^{\mathbf{P}(\mathcal{L})}$$

Abstract representation of the state

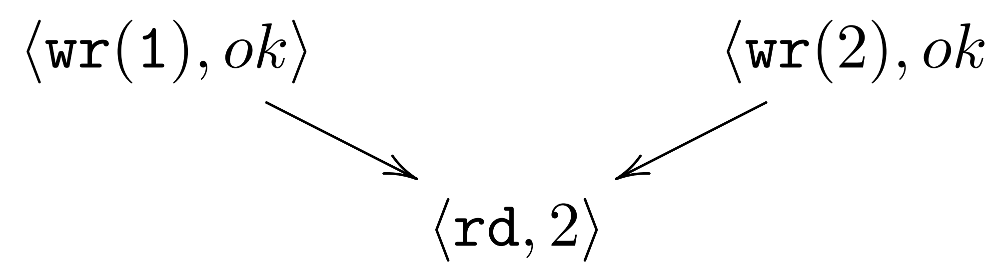
Sequence of operations that generate a state

Abstract representation of states

- A state is given as *labelled acyclic directed graph*
  - a **node** represents an executed operation
  - a label describes
    - the invoked operation, and
    - the return value
  - arcs stands for visible dependencies

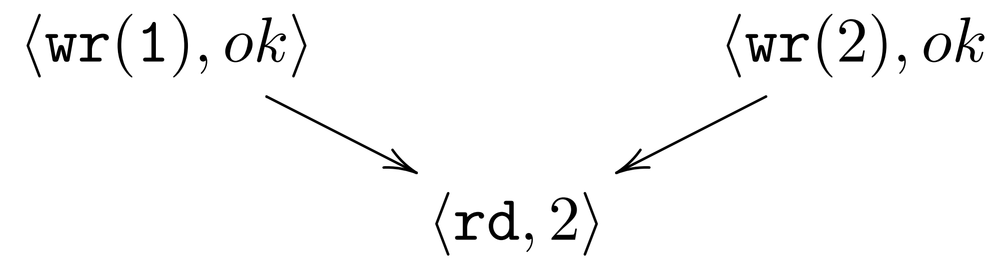
Abstract representation of states

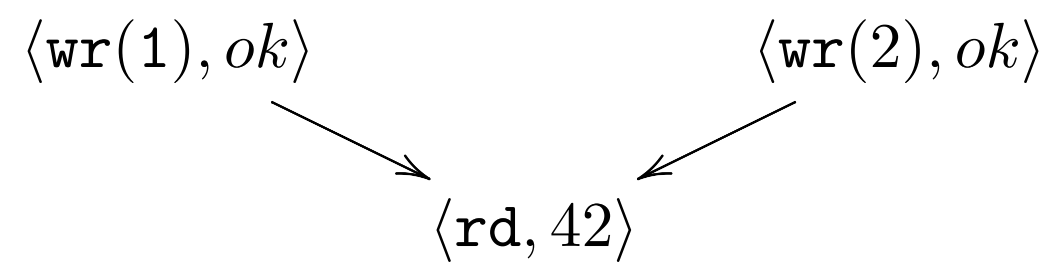
- A state is given as *labelled acyclic directed graph*
  - a **node** represents an executed operation
  - a label describes
    - the invoked operation, and
    - the return value
  - arcs stands for visible dependencies



Abstract representation of states

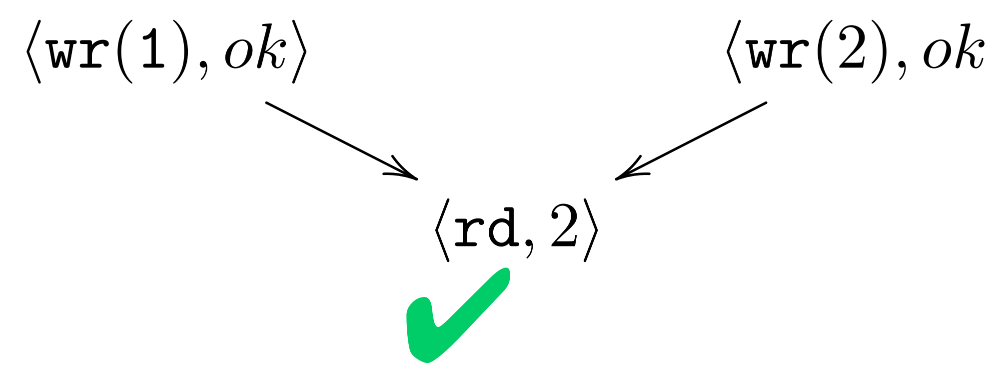
- A state is given as *labelled acyclic directed graph*
  - a **node** represents an executed operation
  - a label describes
    - the invoked operation, and
    - the return value
  - arcs stands for visible dependencies

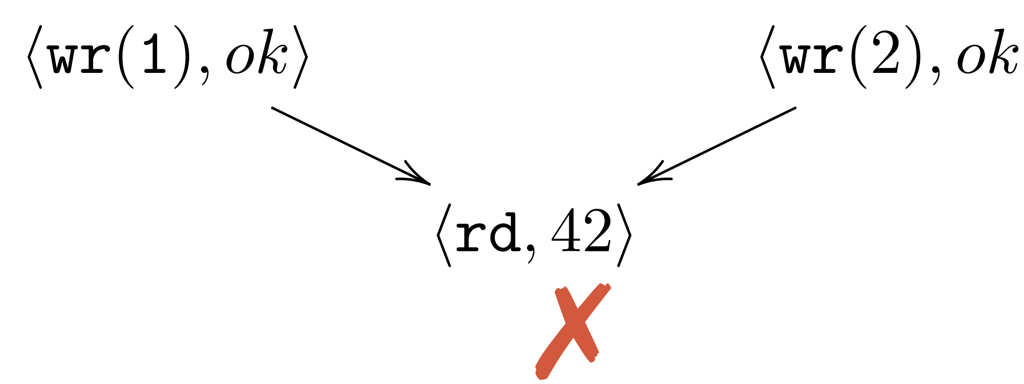




Abstract representation of states

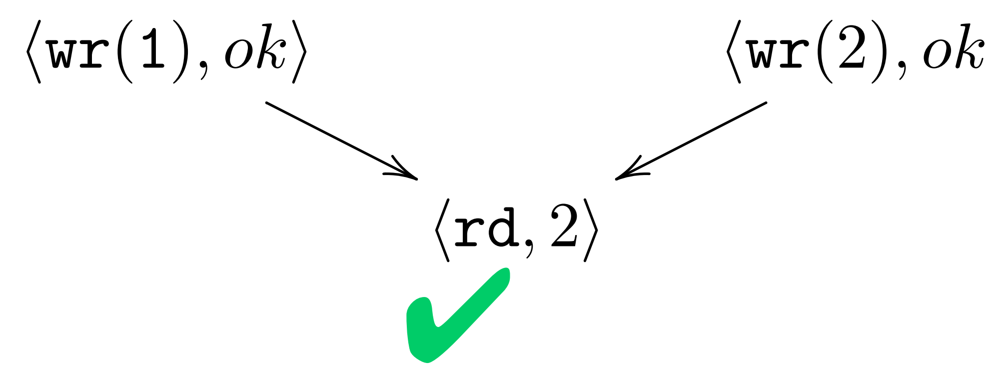
- A state is given as *labelled acyclic directed graph*
  - a **node** represents an executed operation
  - a label describes
    - the invoked operation, and
    - the return value
  - arcs stands for visible dependencies



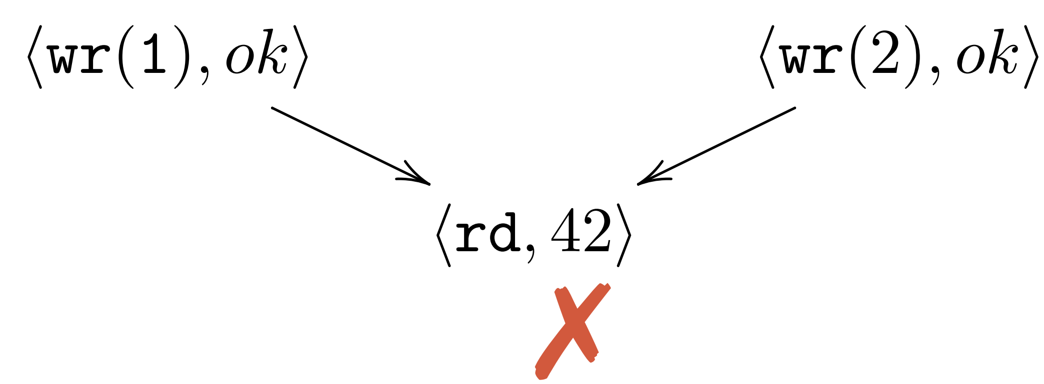


Abstract representation of states

- A state is given as *labelled acyclic directed graph*
  - a **node** represents an executed operation
  - a label describes
    - the invoked operation, and
    - the return value
  - arcs stands for visible dependencies



A specification allows us to make such a distinction



RDT specifications (functional style)

$$\mathbf{S} : \mathbf{G}(\mathcal{L}) \rightarrow 2^{\mathbf{P}(\mathcal{L})}$$

Acyclic graphs labelled over $\mathcal{L}$

Total orders (paths) labelled over $\mathcal{L}$

RDT specifications (functional style)

$$\mathbf{S} : \mathbf{G}(\mathcal{L}) \rightarrow 2^{\mathbf{P}(\mathcal{L})}$$

Acyclic graphs labelled over $\mathcal{L}$

Total orders (paths) labelled over $\mathcal{L}$

$$\mathbf{S}\left(\begin{array}{ccc} \langle \mathtt{wr}(1), ok \rangle & & \langle \mathtt{wr}(2), ok \rangle \\ & \searrow \quad \swarrow & \\ & \langle \mathtt{rd}, 2 \rangle & \end{array} \right) = \left\{ \begin{array}{c} \langle \mathtt{wr}(1), ok \rangle \\ | \\ \langle \mathtt{wr}(2), ok \rangle \\ | \\ \langle \mathtt{rd}, 2 \rangle \end{array} \right\}$$

A state

An ordering that generates that state

RDT specifications (functional style)

$$\mathbf{S} : \mathbf{G}(\mathcal{L}) \rightarrow 2^{\mathbf{P}(\mathcal{L})}$$

Acyclic graphs labelled over $\mathcal{L}$

Total orders (paths) labelled over $\mathcal{L}$

$$\mathbf{S}\left(\begin{array}{ccc} \langle \mathtt{wr}(1), ok\rangle & & \langle \mathtt{wr}(2), ok\rangle \\ & \searrow \quad \swarrow & \\ & \langle \mathtt{rd}, 1\rangle & \end{array}\right) = \left\{\begin{array}{c} \langle \mathtt{wr}(2), ok\rangle \\ | \\ \langle \mathtt{wr}(1), ok\rangle \\ | \\ \langle \mathtt{rd}, 1\rangle \end{array}\right\}$$

A state

An ordering that generates that state

RDT specifications (functional style)

$$\mathbf{S} : \mathbf{G}(\mathcal{L}) \rightarrow 2^{\mathbf{P}(\mathcal{L})}$$

Acyclic graphs labelled over $\mathcal{L}$

Total orders (paths) labelled over $\mathcal{L}$

$$\mathbf{S}\left(\begin{array}{ccc} \langle \mathtt{wr}(1), ok \rangle & & \langle \mathtt{wr}(2), ok \rangle \\ & \searrow \quad \swarrow & \\ & \langle \mathtt{rd}, 3 \rangle & \end{array}\right) = \emptyset$$

A state disallowed by the specification

RDTs, Algebraically: Roadmap

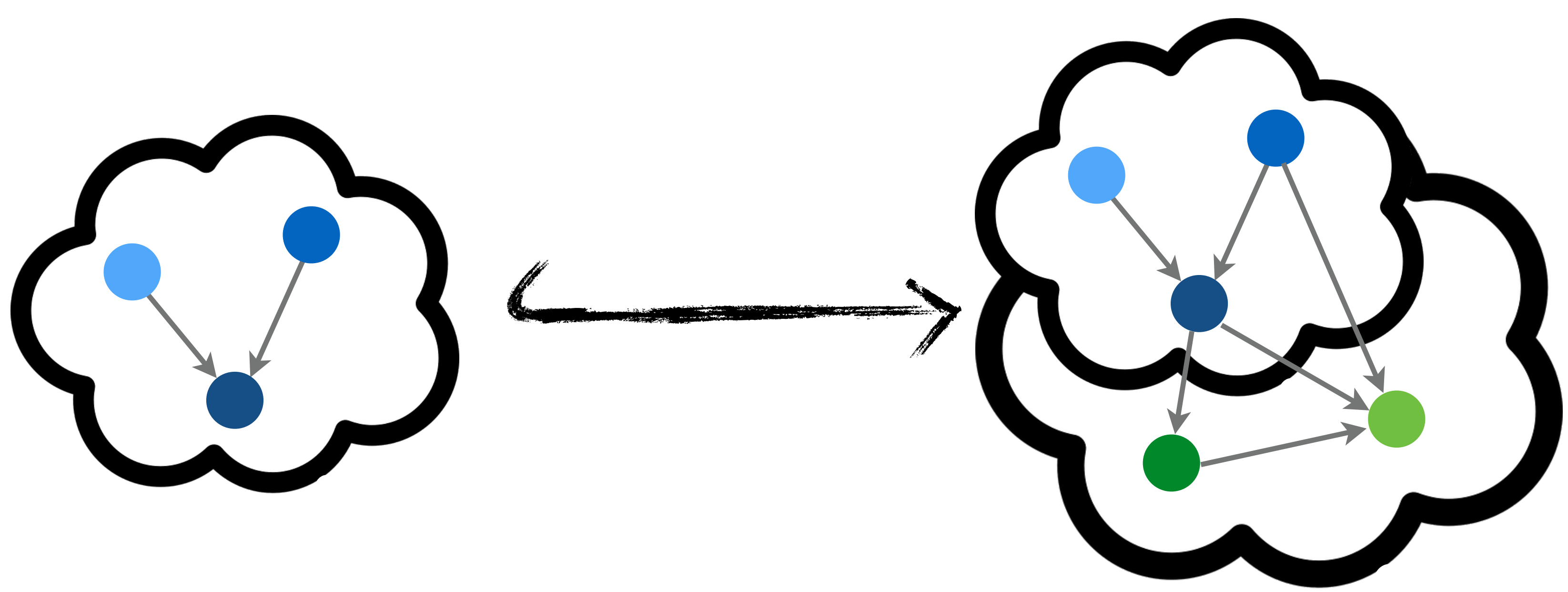
- Specifications are functors
- Implementations are functors
- LTSs are recovered from these functors
- Implementation correctness via simulation

RDT specification, algebraically

- A functor $\mathbf{S} : \mathbf{PIDag}(\mathcal{L}) \rightarrow \mathbf{SPaths}(\mathcal{L})$
  - from the category of states $(\mathbf{PIDag}(\mathcal{L}))$
  - to the category of sets of paths $(\mathbf{SPaths}(\mathcal{L}))$

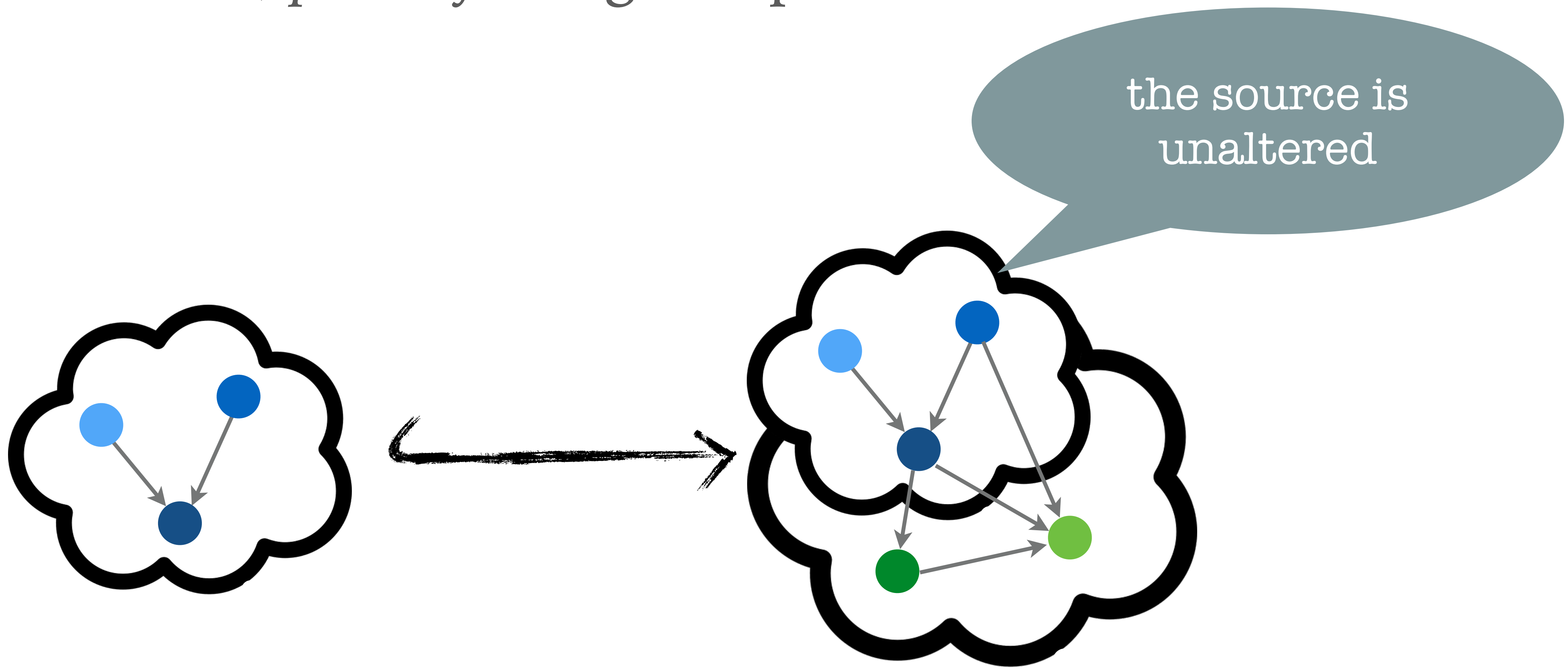
Category of States $\mathbf{PiDag}(\mathcal{L})$

- Objects: Acyclic directed graphs labelled over $\mathcal{L}$
- Arrows: monic, *past-reflecting* morphisms



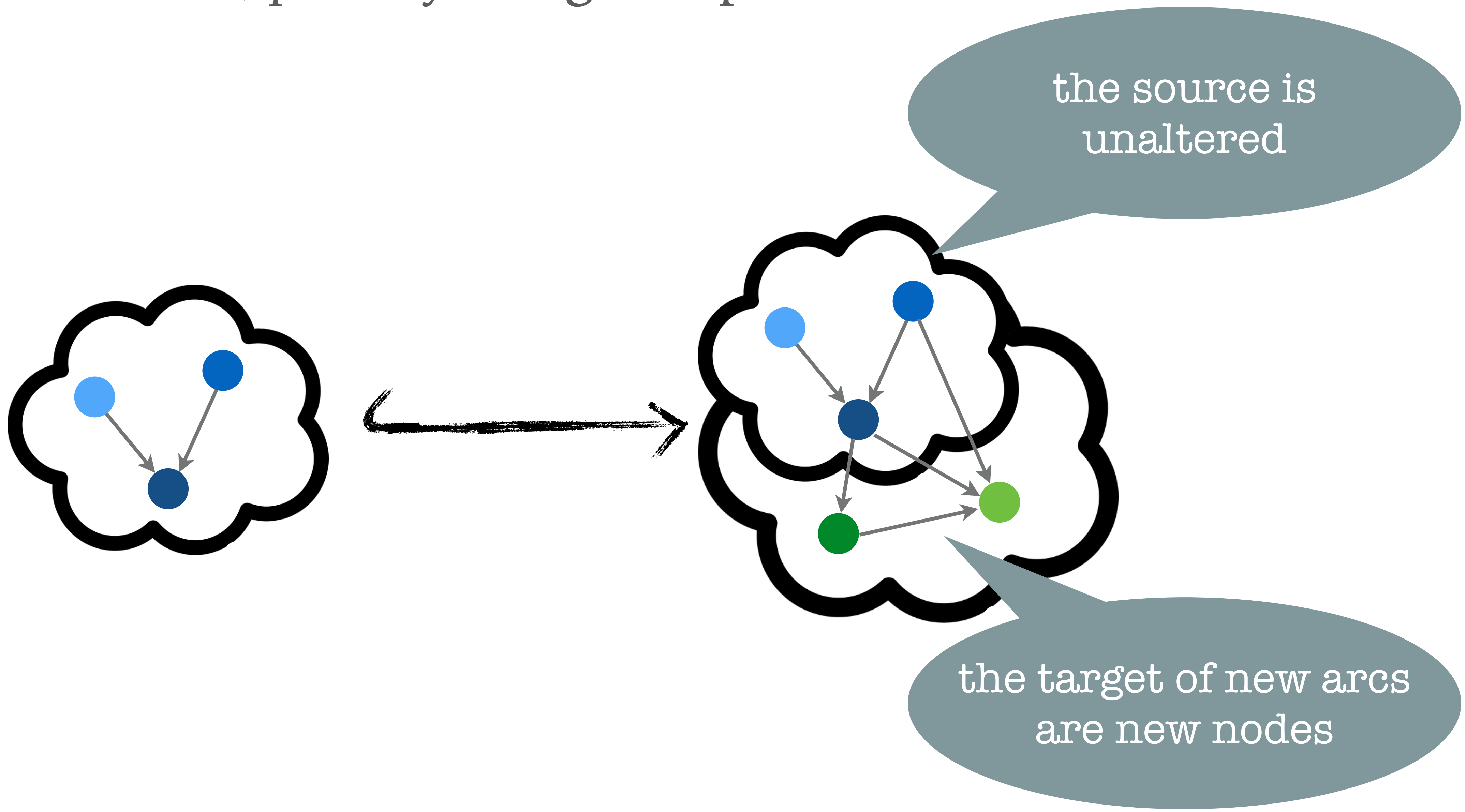
Category of States PiDag($\mathcal{L}$)

- Objects: Acyclic directed graphs labelled over $\mathcal{L}$
- Arrows: monic, *past-reflecting* morphisms



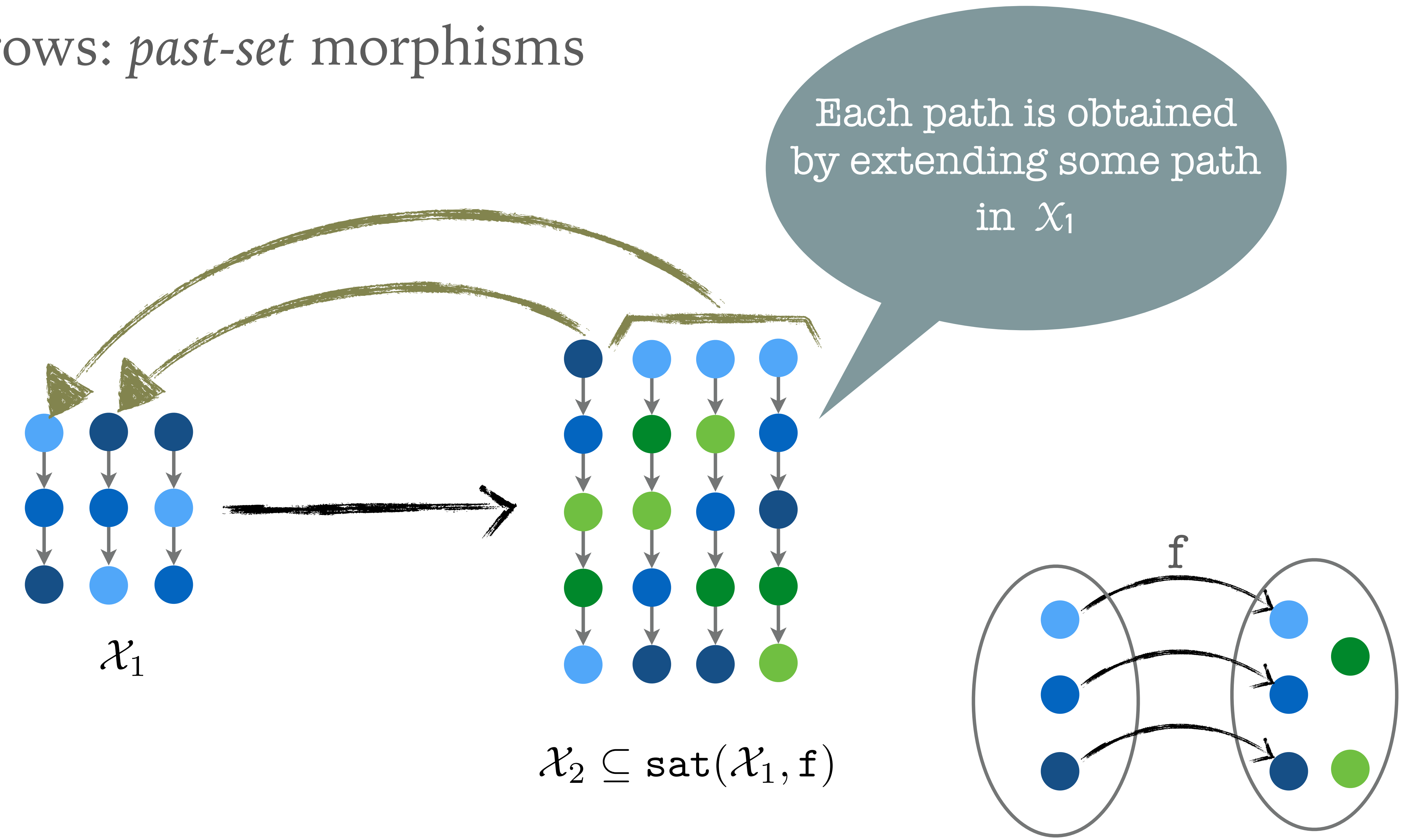
Category of States $\mathbf{PiDag}(\mathcal{L})$

- Objects: Acyclic directed graphs labelled over $\mathcal{L}$
- Arrows: monic, *past-reflecting* morphisms

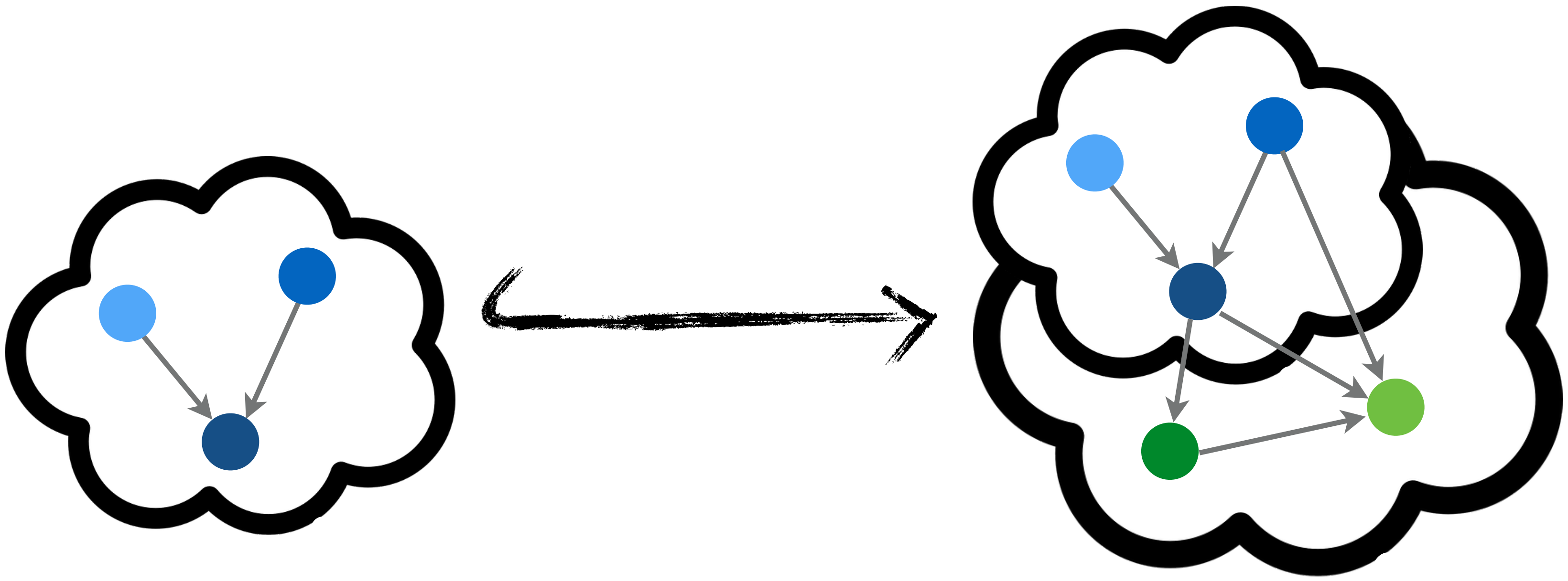


Category of Set of Paths $\mathrm{SPath}(\mathcal{L})$

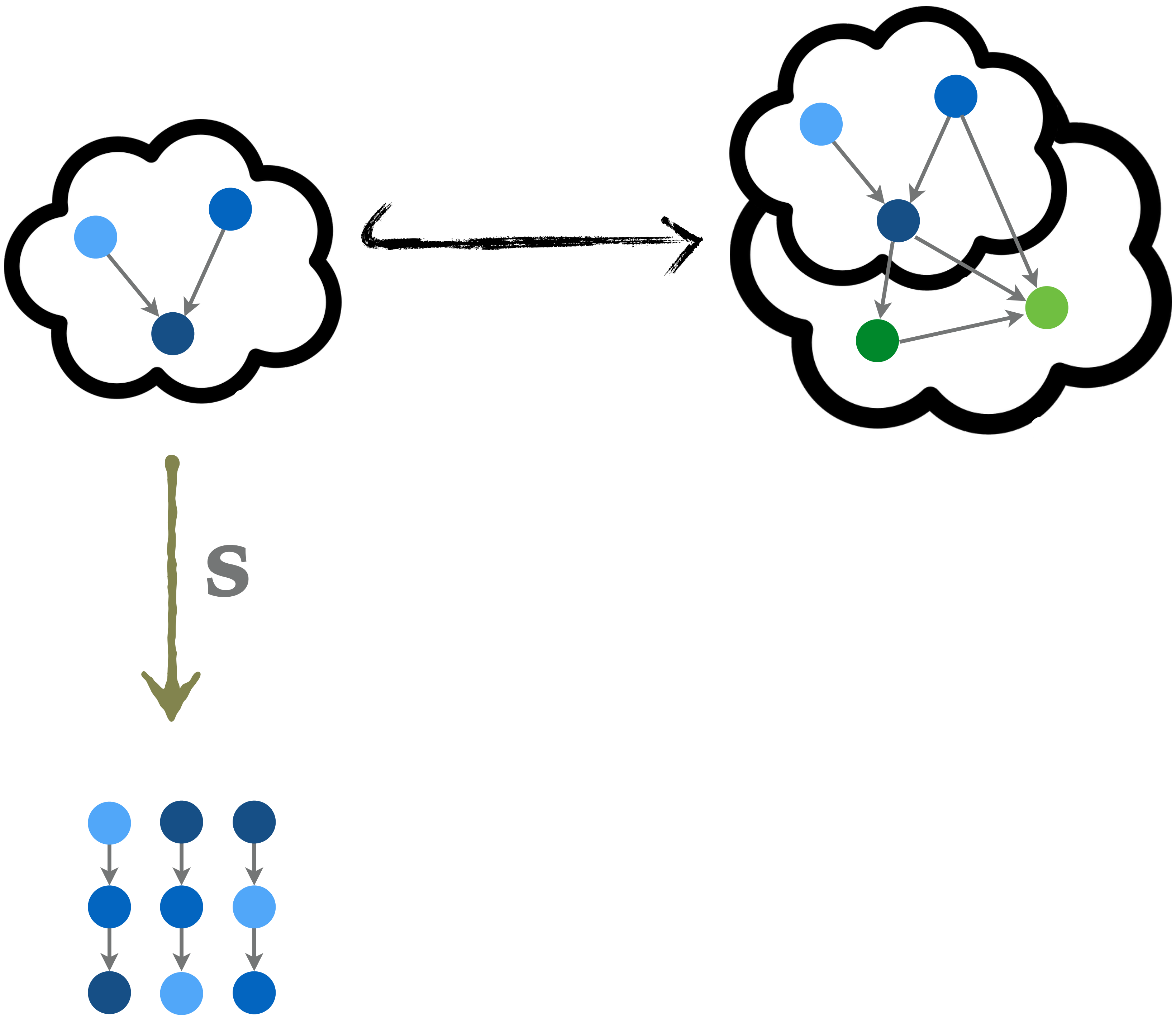
- Objects: set of paths labelled over $\mathcal{L}$
- Arrows: *past-set* morphisms



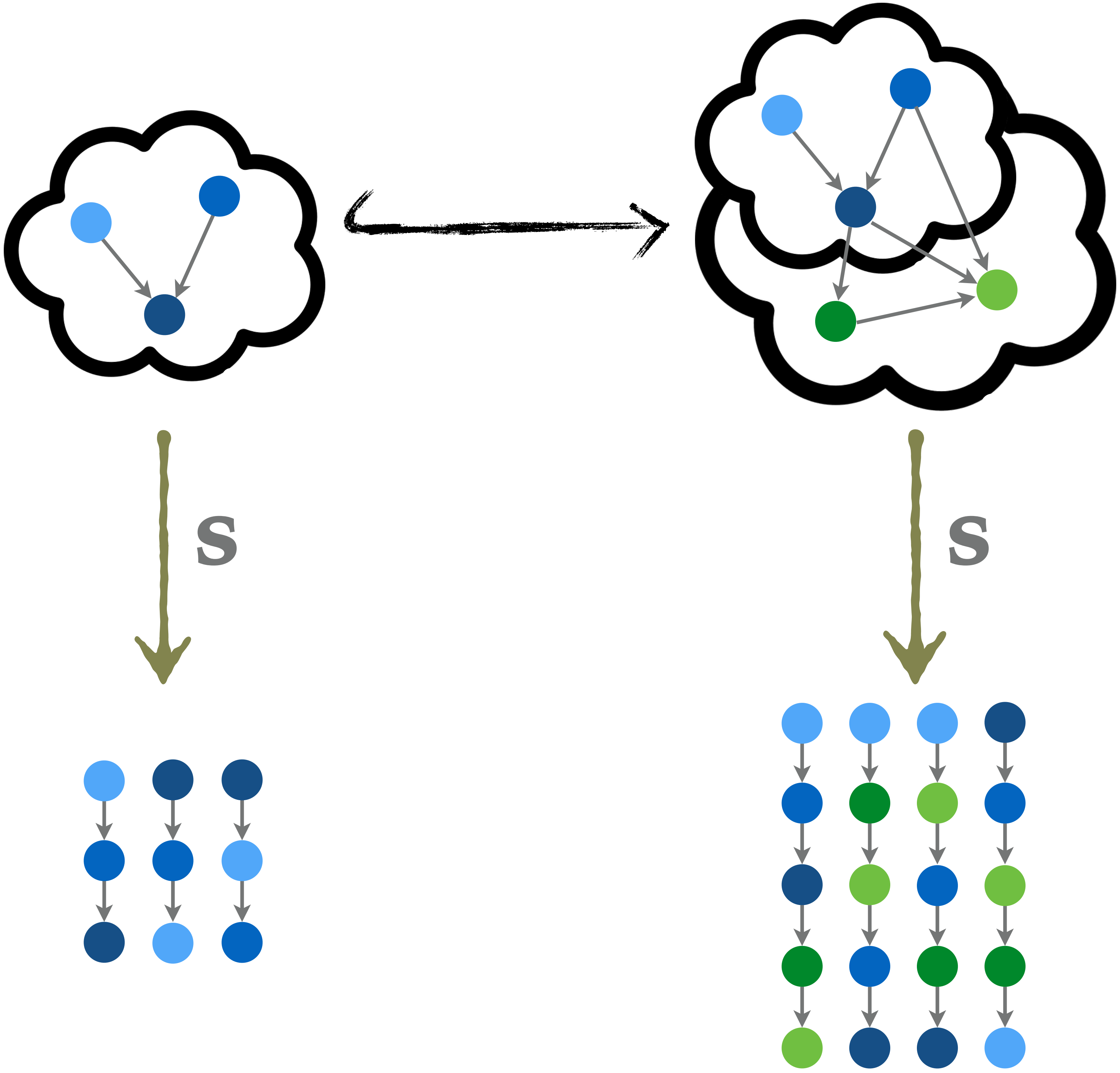
Specifications are functors from $\mathrm{PiDag}(\mathcal{L})$ to $\mathrm{SPath}(\mathcal{L})$



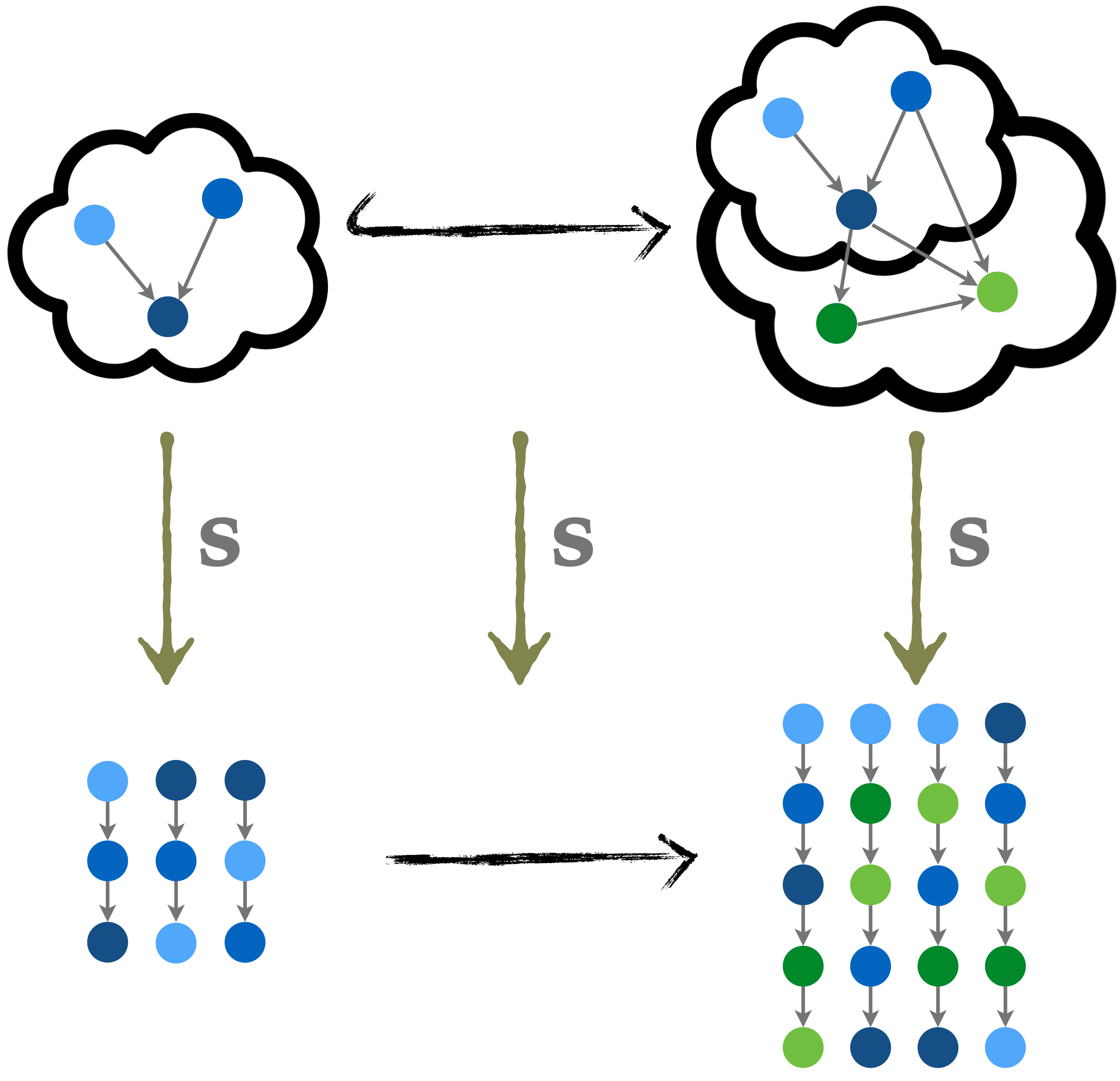
Specifications are functors from PiDag($\mathcal{L}$) to SPath($\mathcal{L}$)



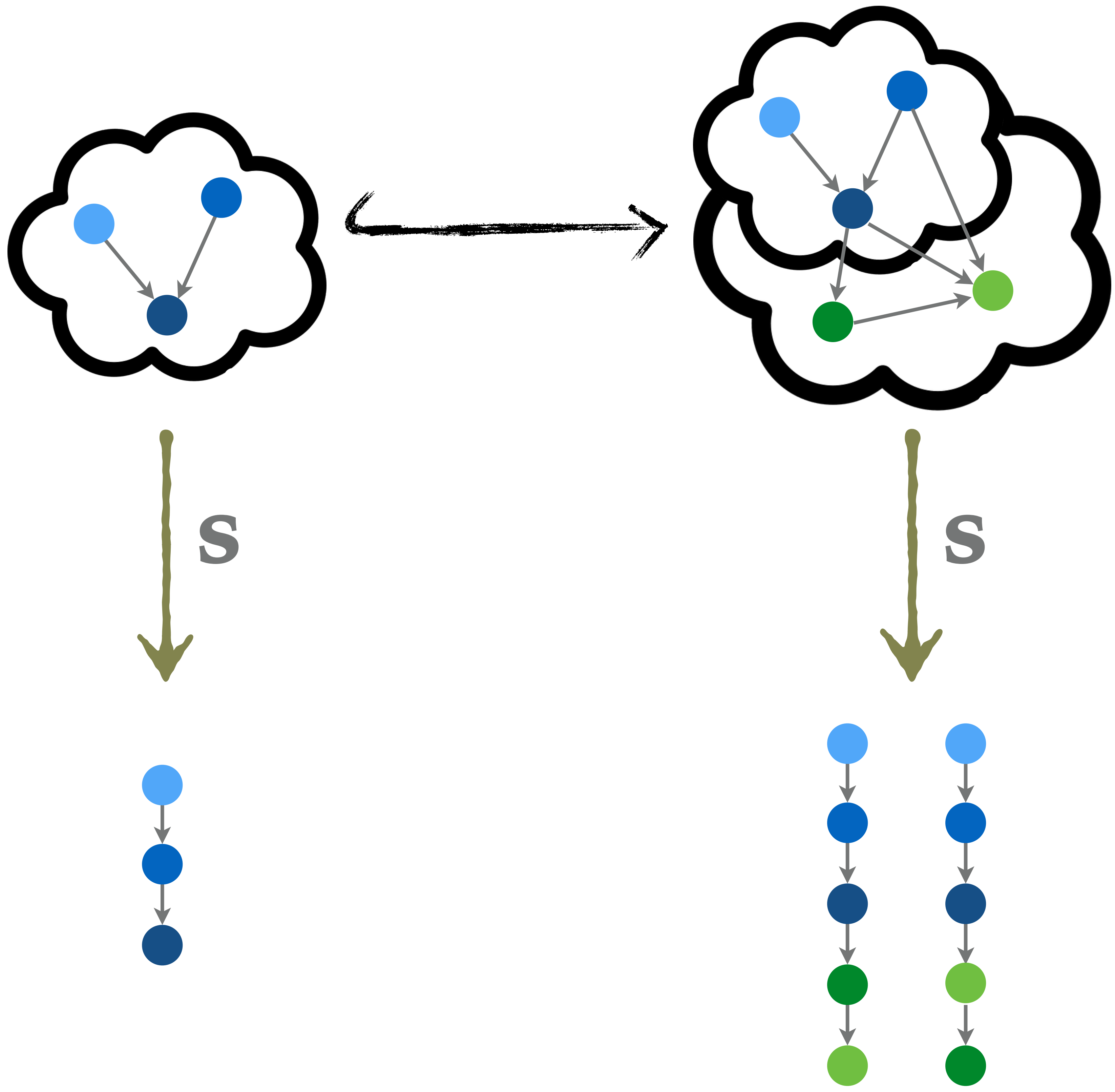
Specifications are functors from PiDag($\mathcal{L}$) to SPath($\mathcal{L}$)



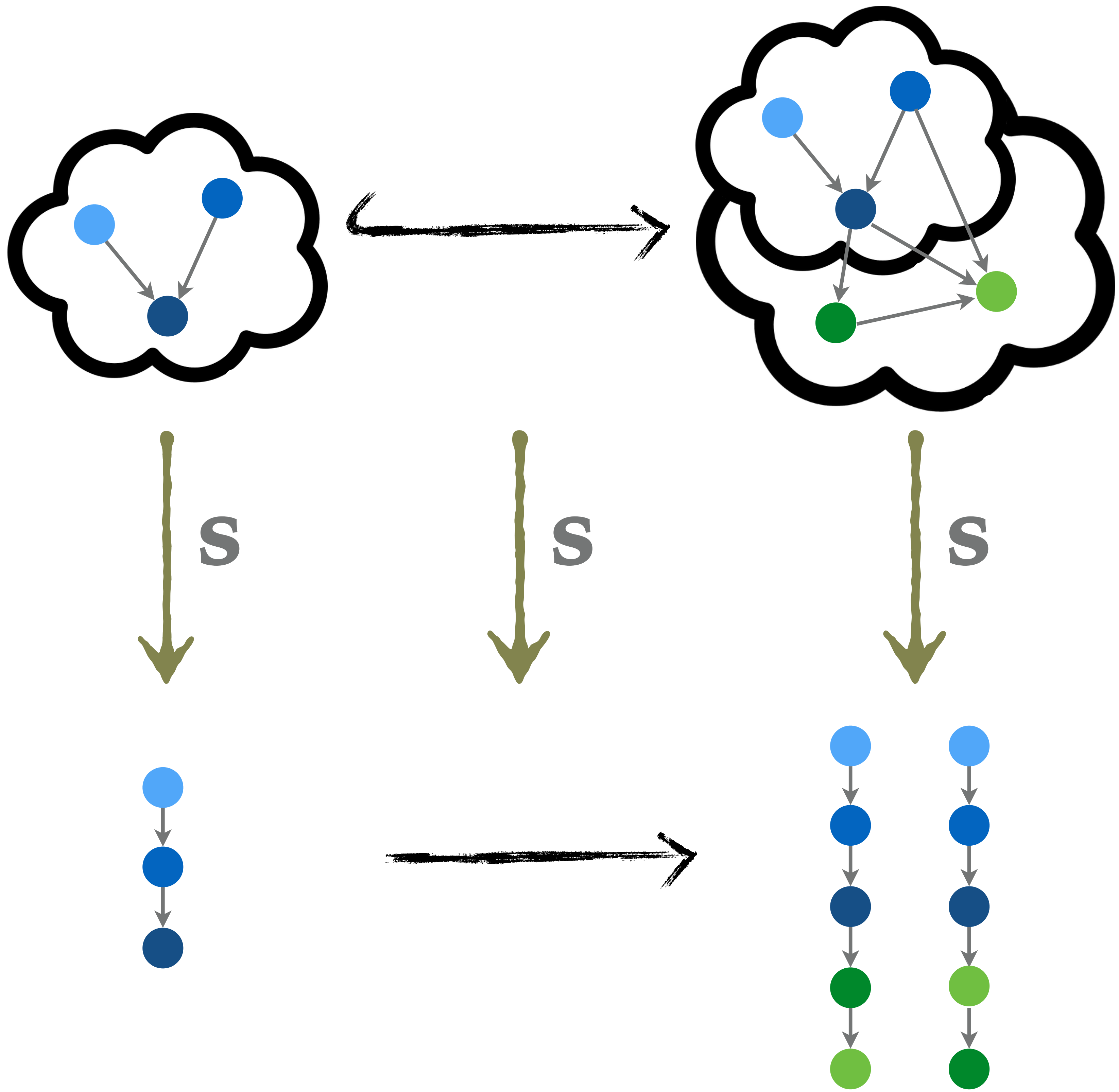
Specifications are functors from PiDag($\mathcal{L}$) to SPath($\mathcal{L}$)



Specifications may be topological...



Specifications may be topological...



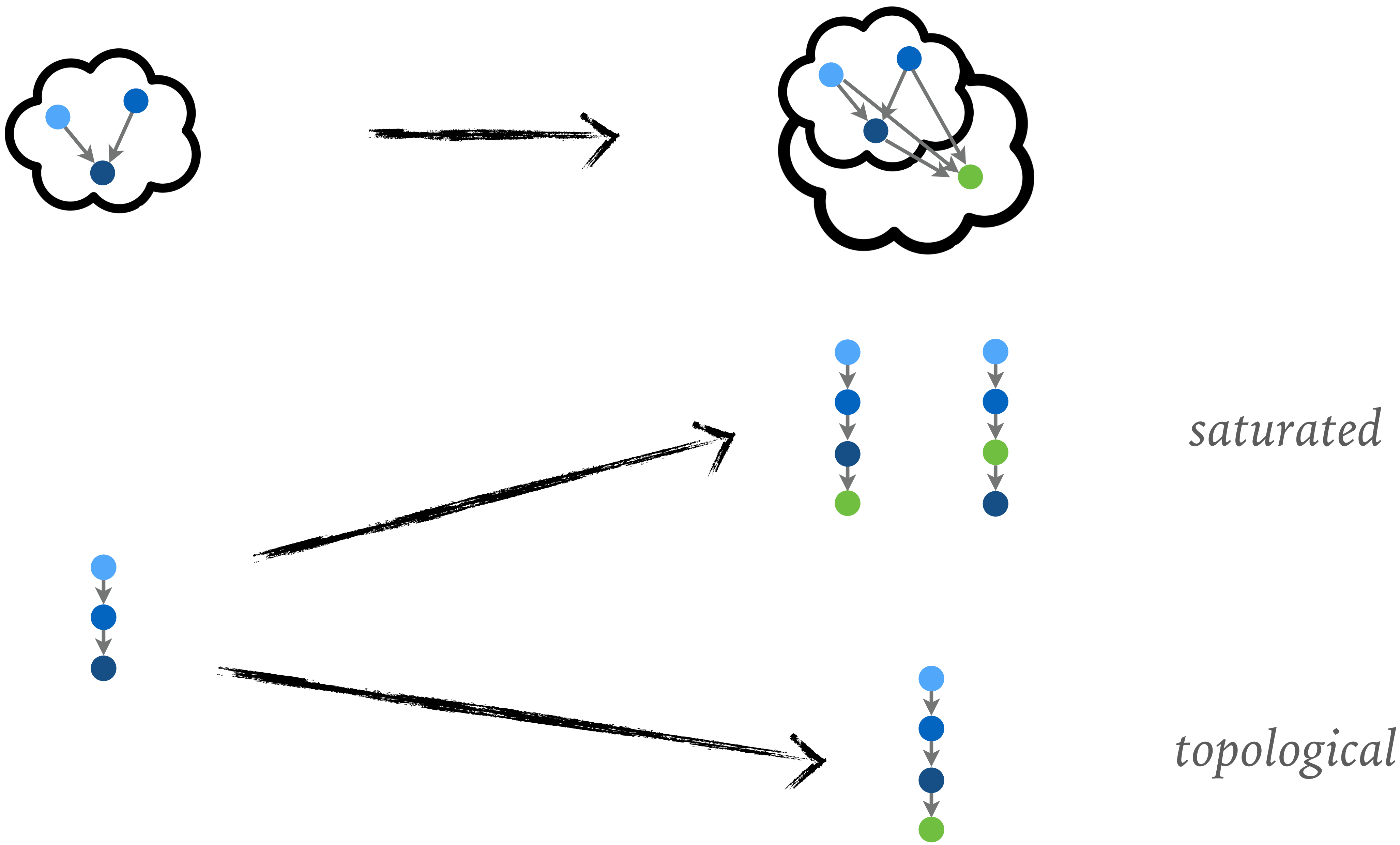
Specifications Are Functors “Preserving” Pushouts…

Specifications Are Functors "Preserving" Pushouts…

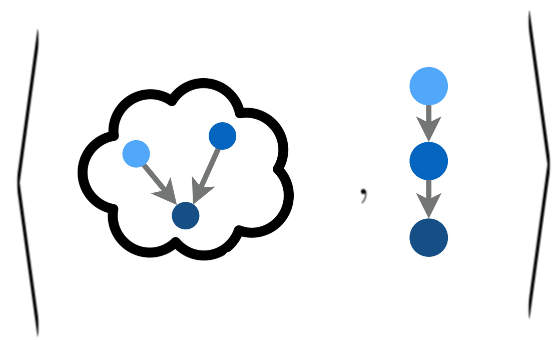
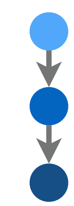
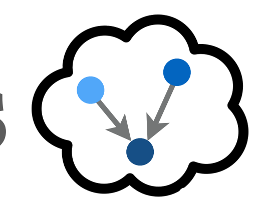
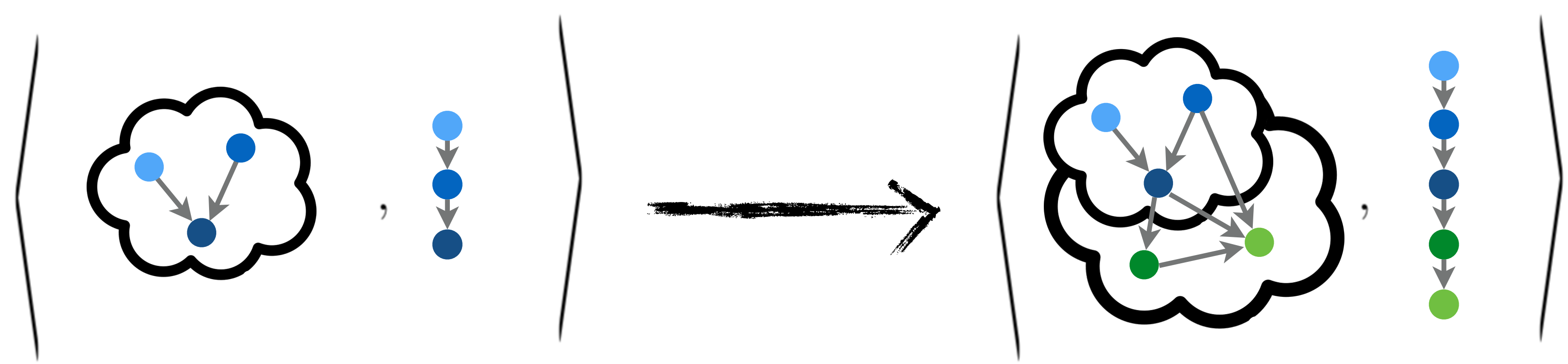
- (meaning pushouts corresponds to coherence)
- … plus suitably mapping root extensions

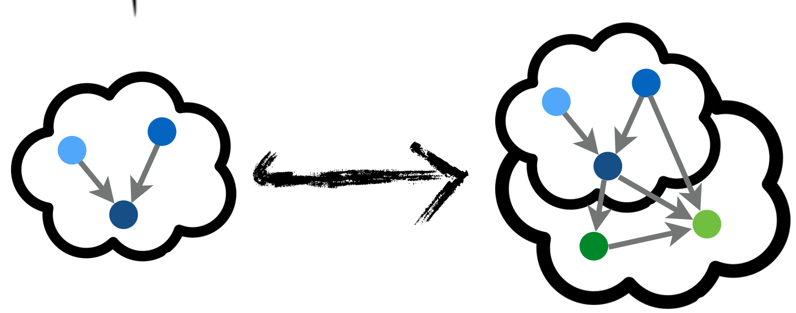
Specifications Are Functors "Preserving" Pushouts…

- (meaning pushouts corresponds to coherence)
- … plus suitably mapping root extensions



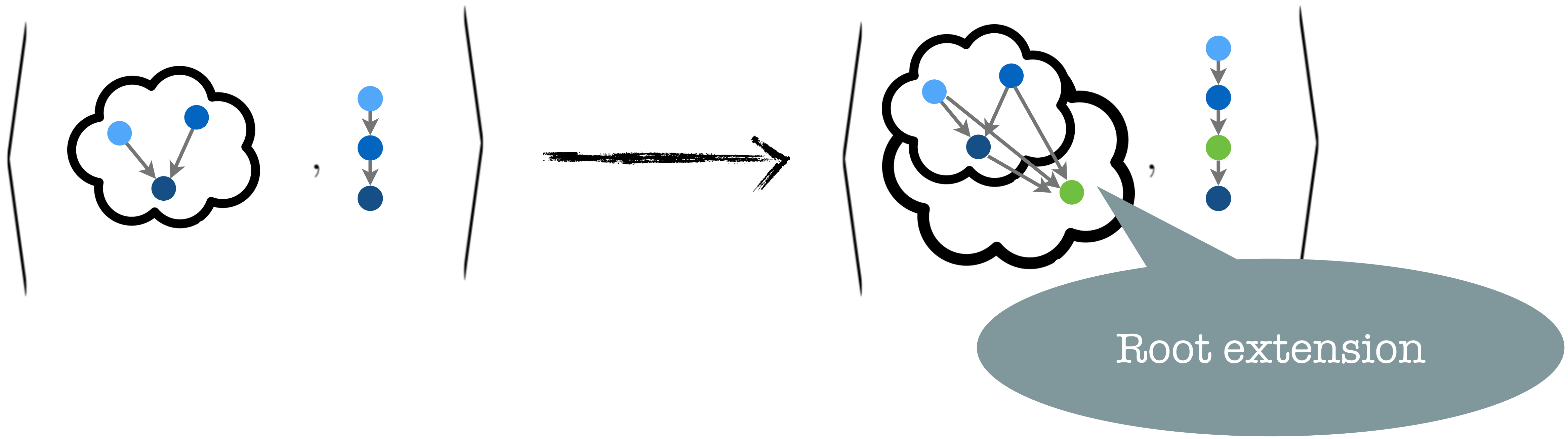
Operational interpretation of a specification s

- The category of elements $\mathscr{E}(\mathbf{S})$
  - Objects:  with  $\in$ **S** 
  - Arrows: 

  with  past-preserving

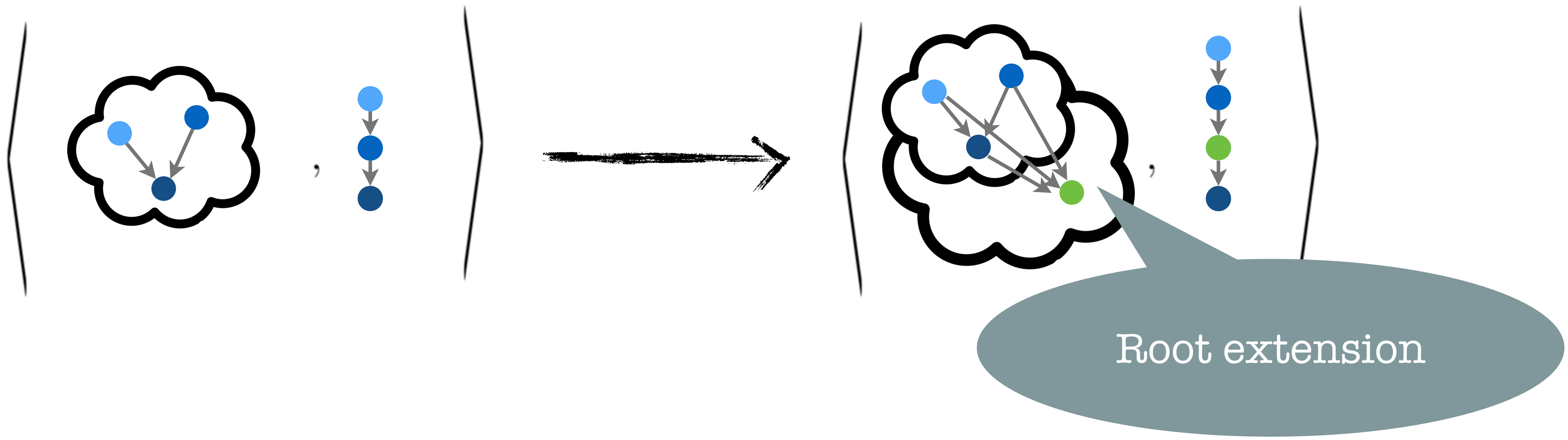
Operational interpretation of a specification $\mathscr{E}(\mathbf{S})$

- $\mathscr{E}_o(\mathbf{S})$: The behaviour of one replica

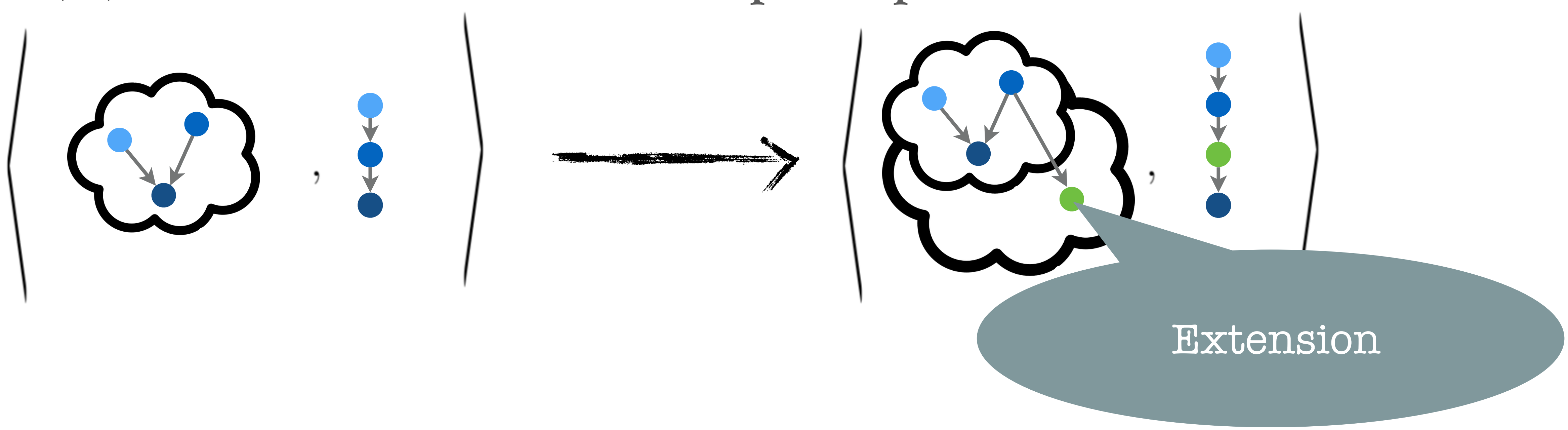


Operational interpretation of a specification $\mathcal{E}(\mathbf{S})$

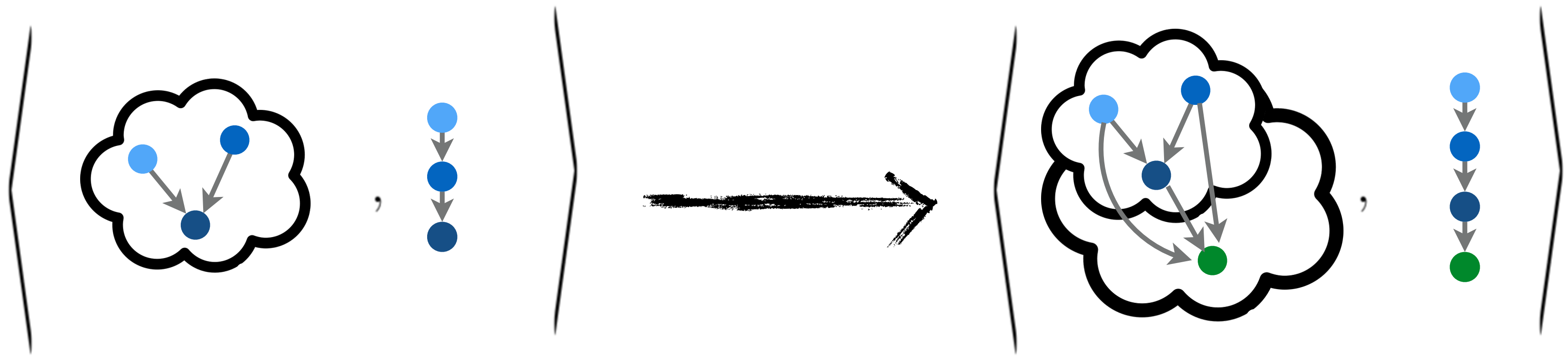
- $\mathcal{E}_o(\mathbf{S})$: The behaviour of one replica



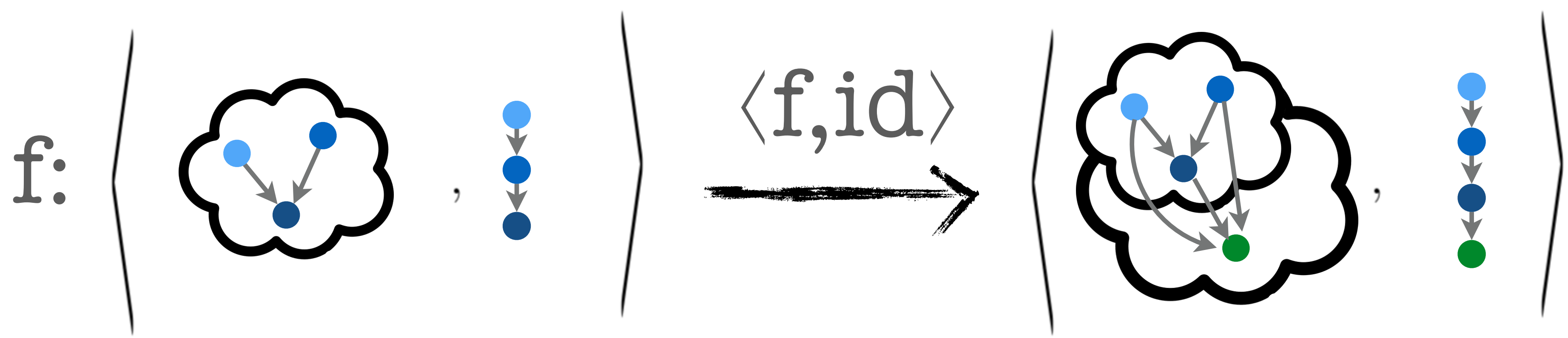
- $\mathcal{E}_m(\mathbf{S})$: The behaviour of multiple replicas



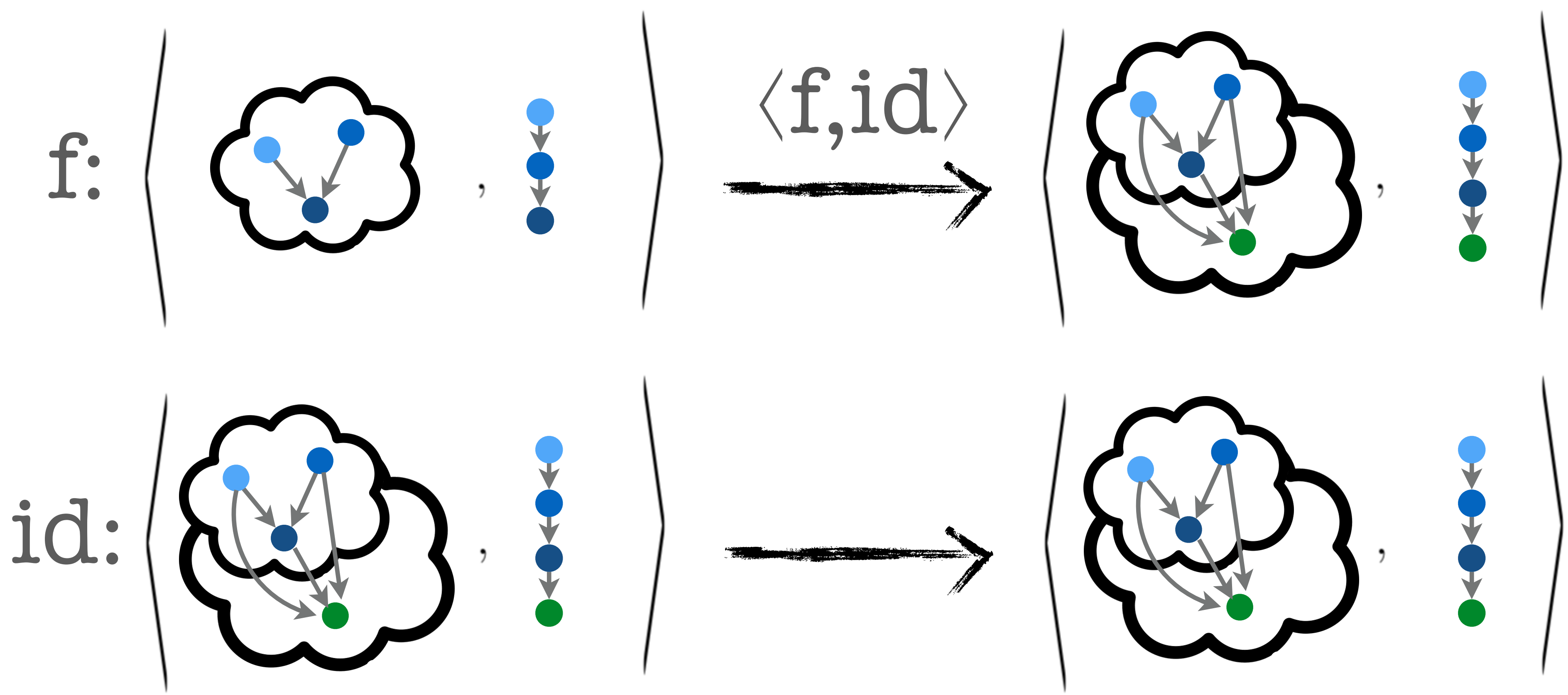
Recovering labels (Leifer-Milner approach)



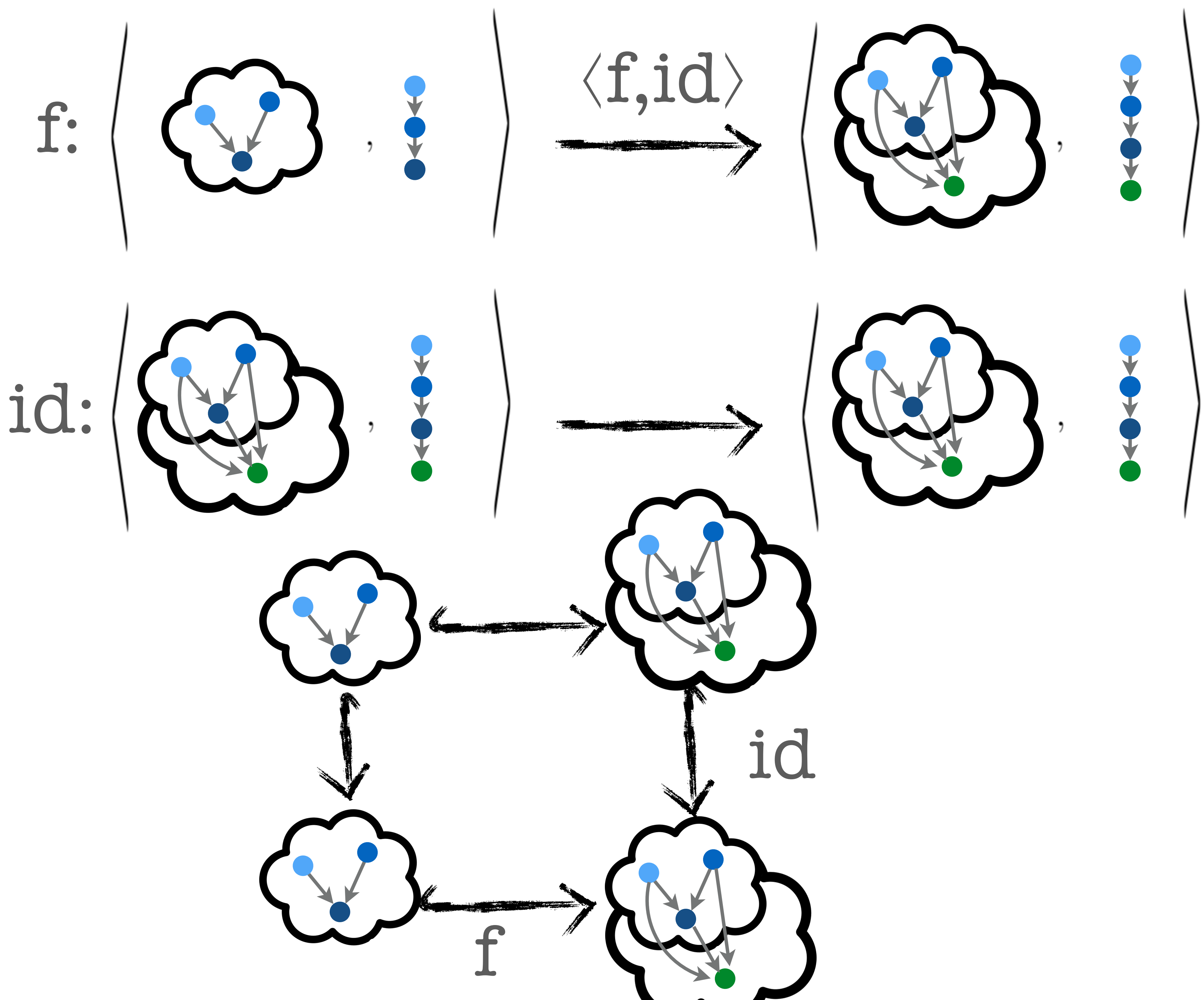
Recovering labels (Leifer-Milner approach)



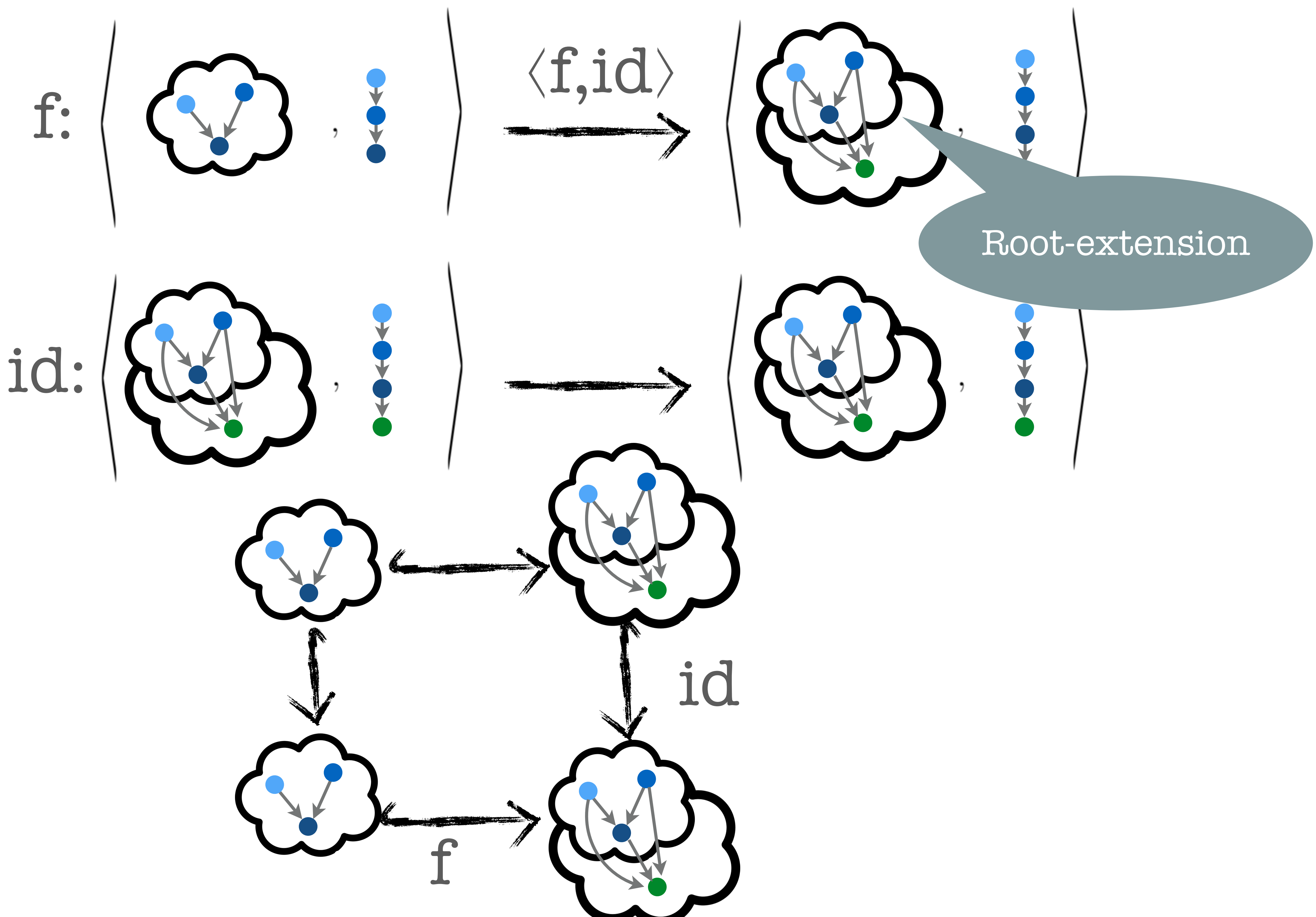
Recovering labels (Leifer-Milner approach)



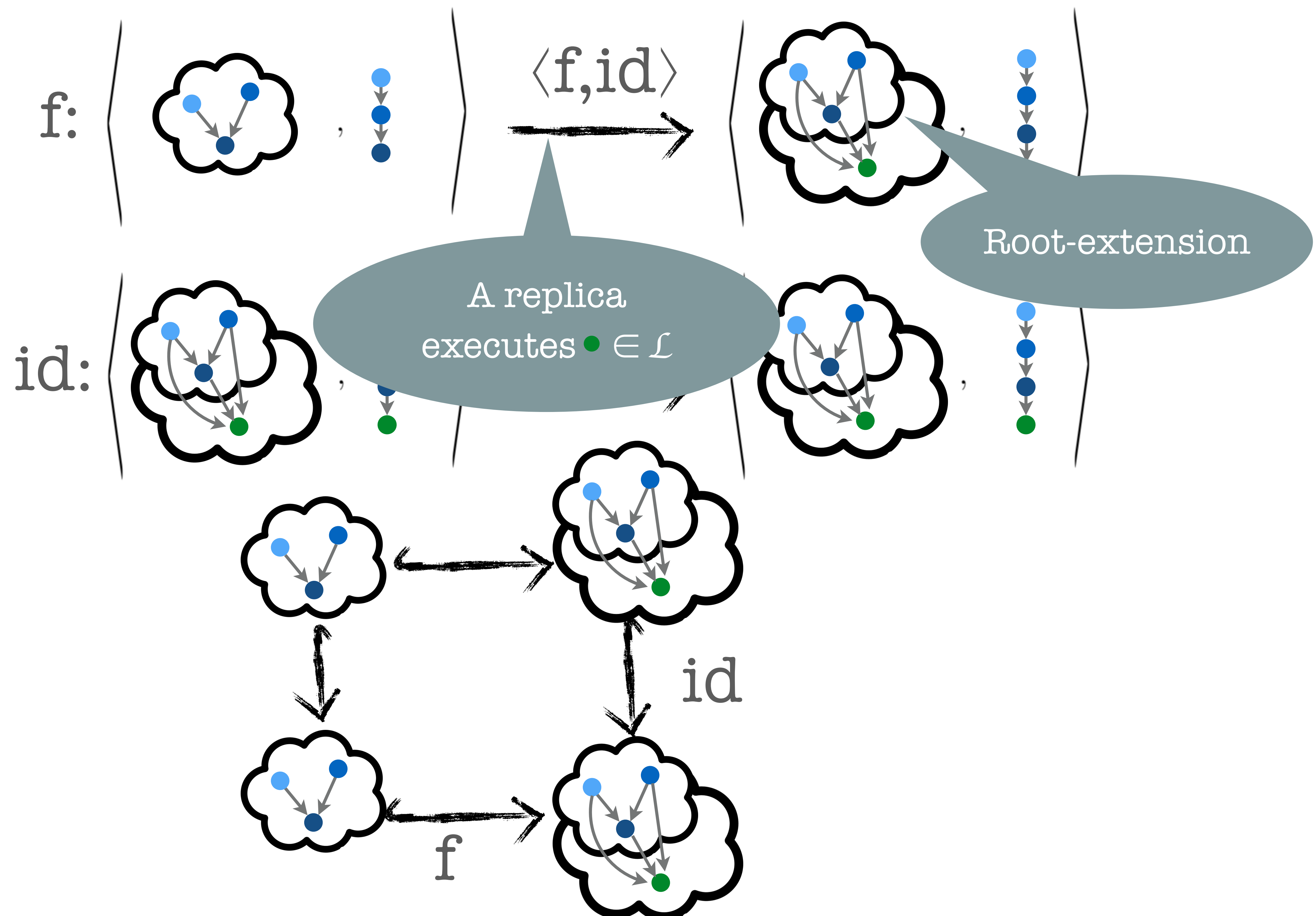
Recovering labels (Leifer-Milner approach)



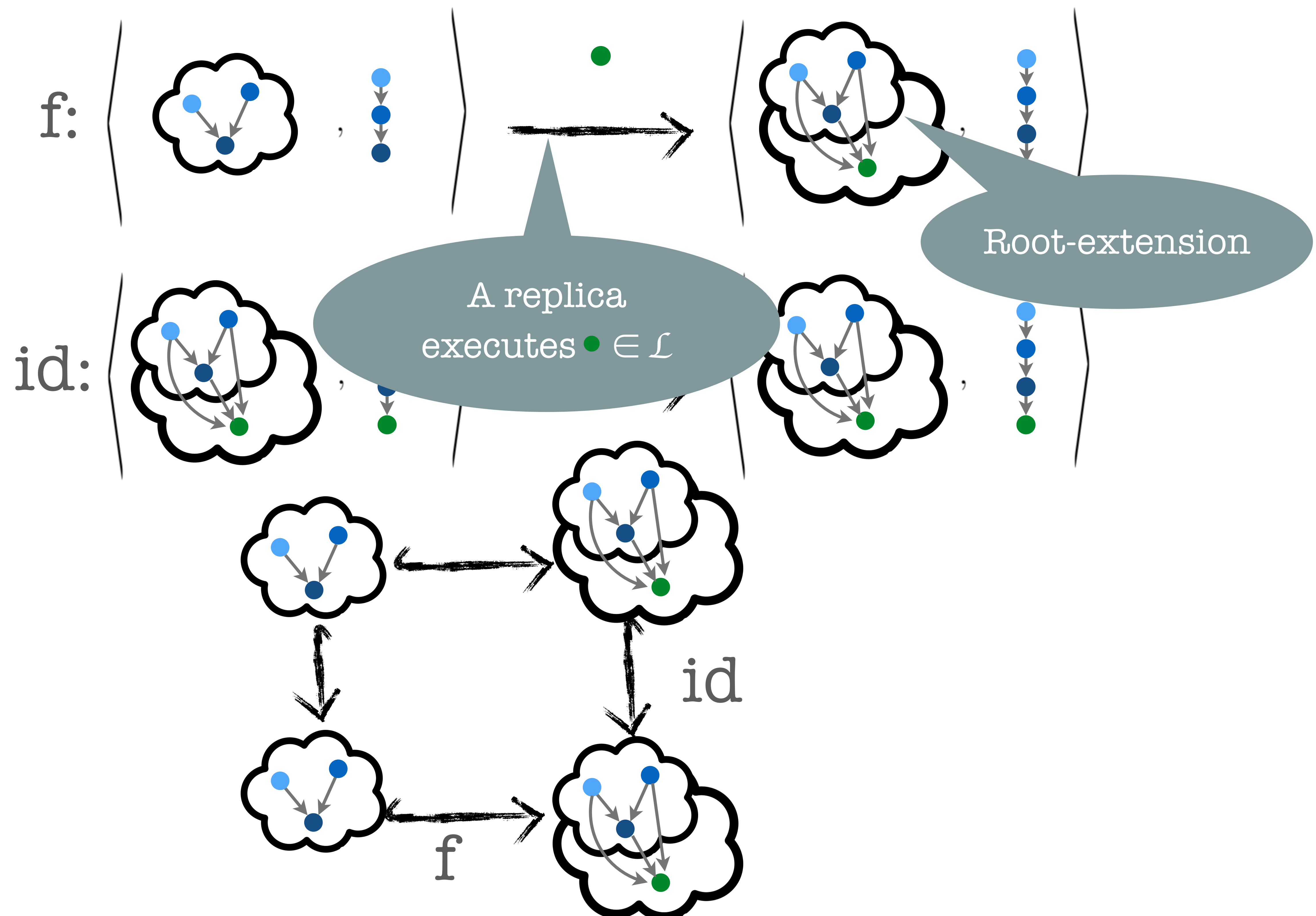
Recovering labels (Leifer-Milner approach)



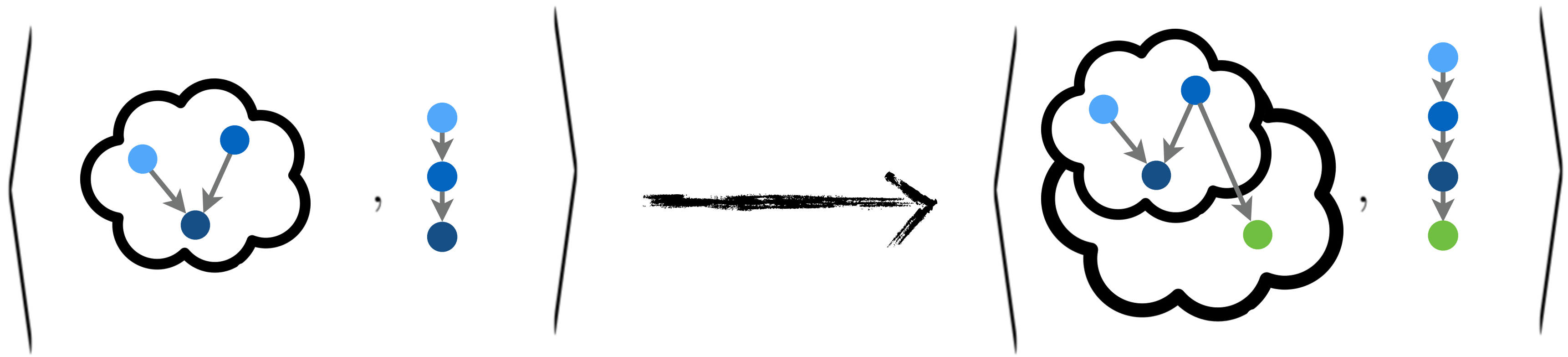
Recovering labels (Leifer-Milner approach)



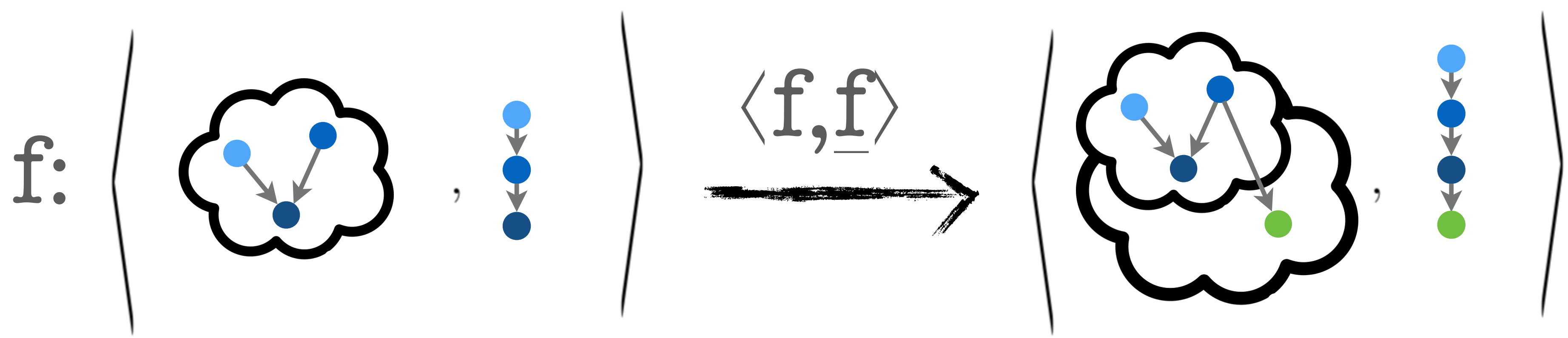
Recovering labels (Leifer-Milner approach)



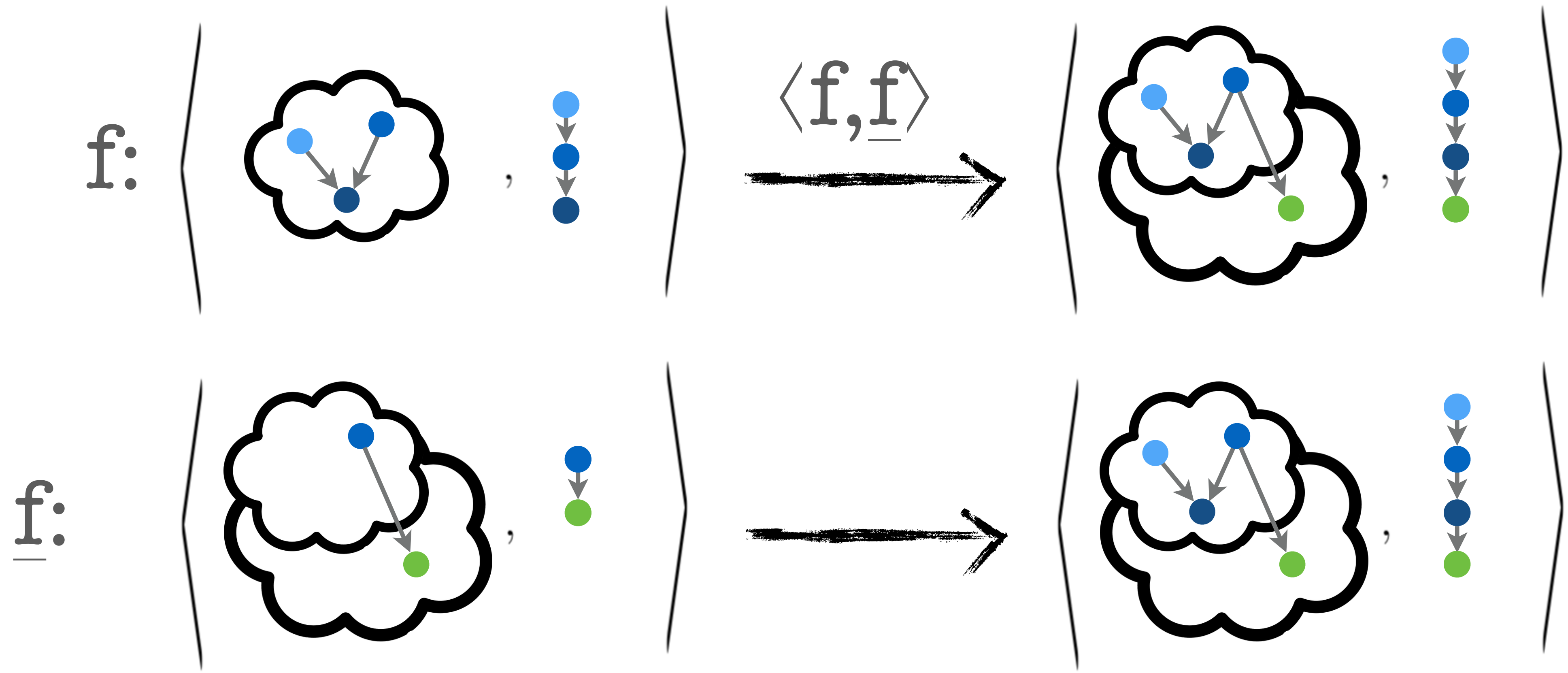
Recovering labels (Leifer-Milner approach)



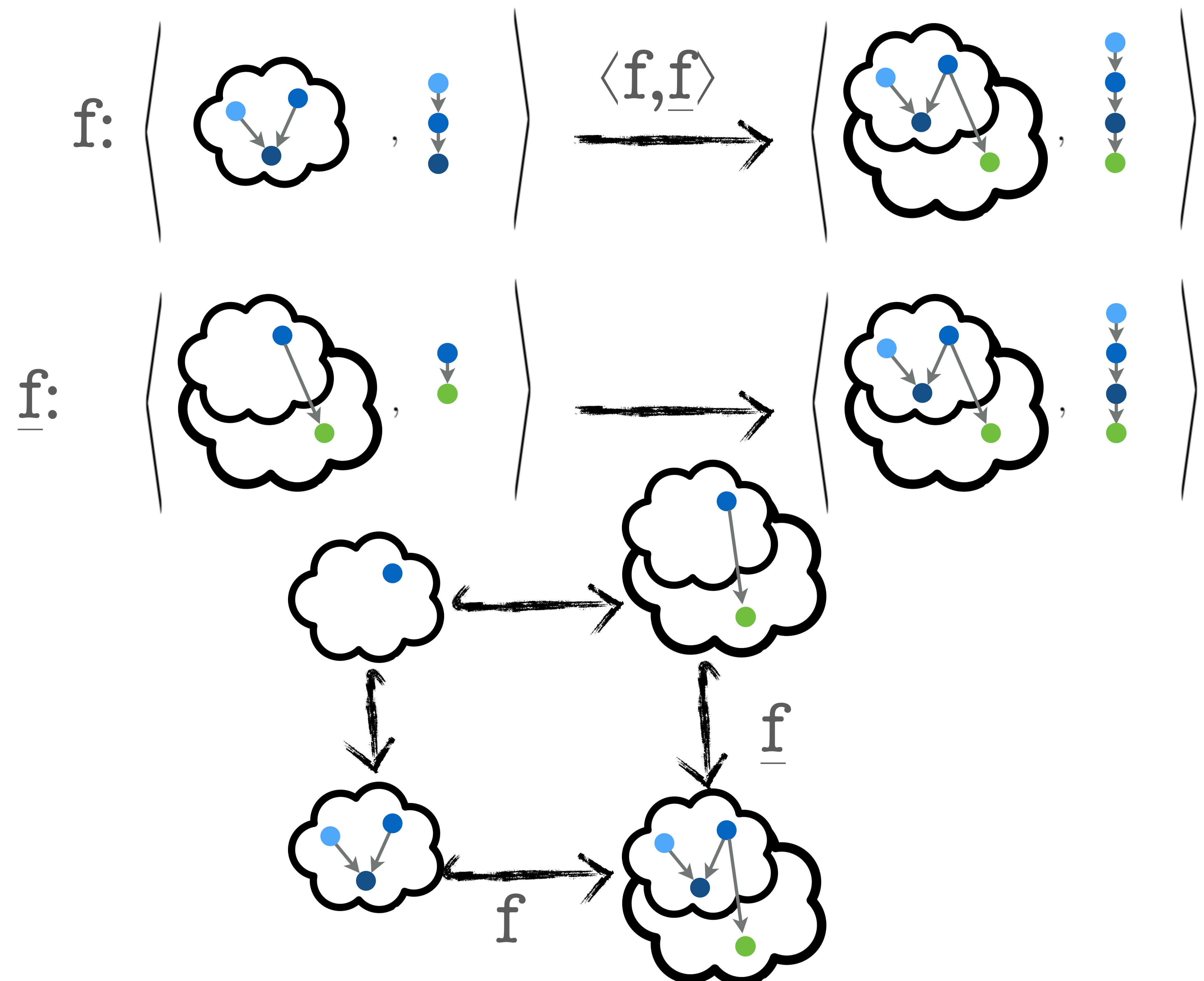
Recovering labels (Leifer-Milner approach)



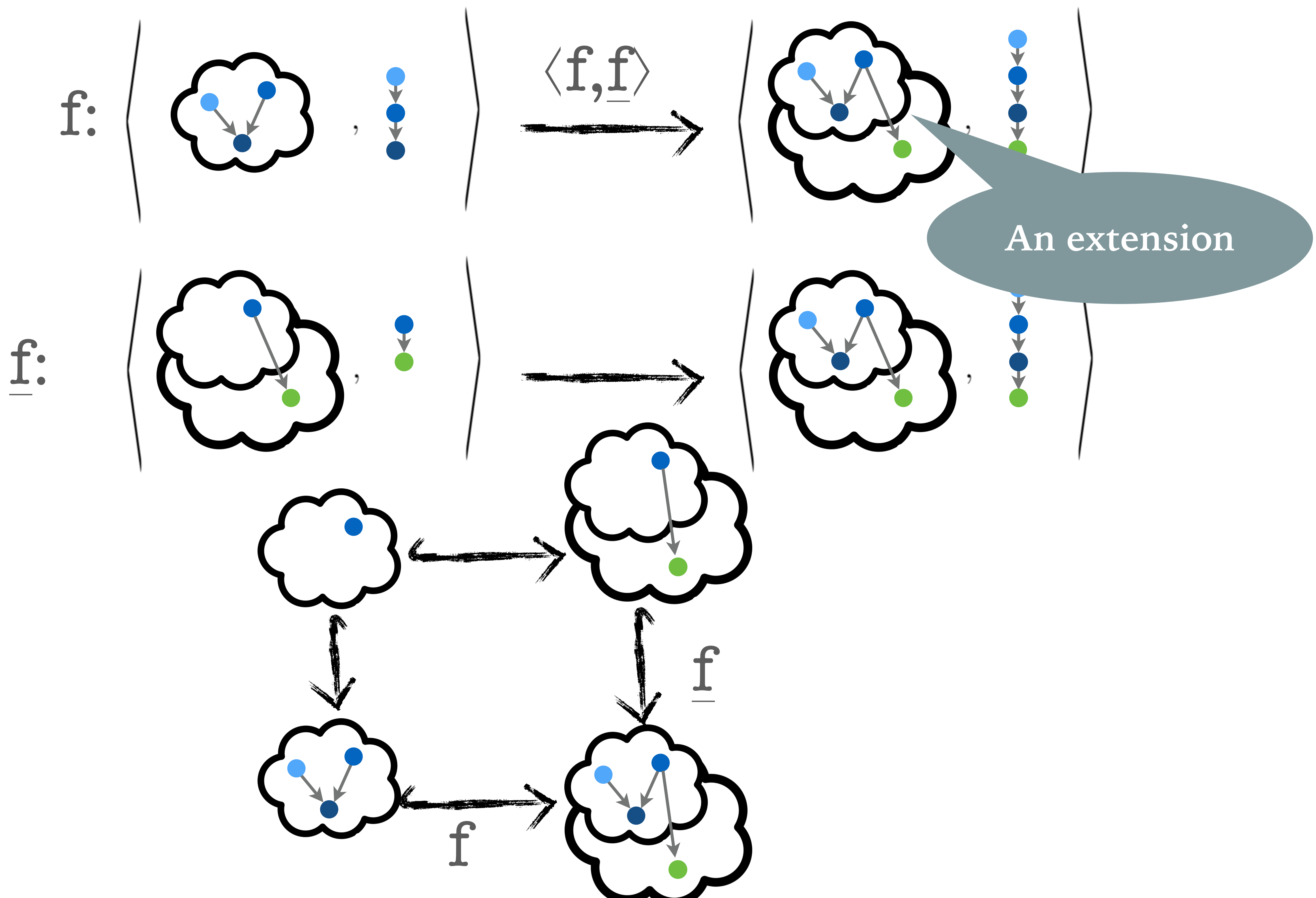
Recovering labels (Leifer-Milner approach)



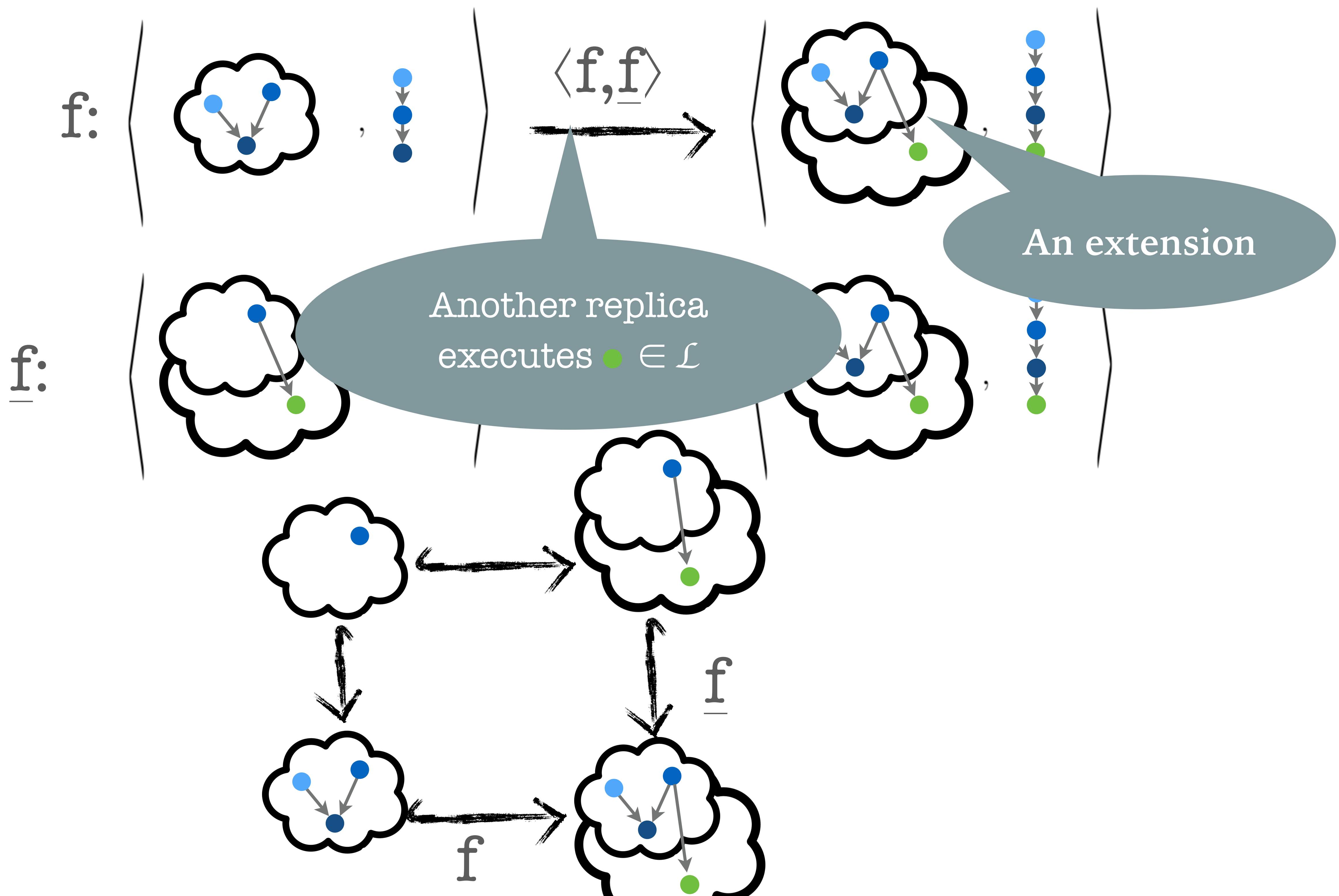
Recovering labels (Leifer-Milner approach)



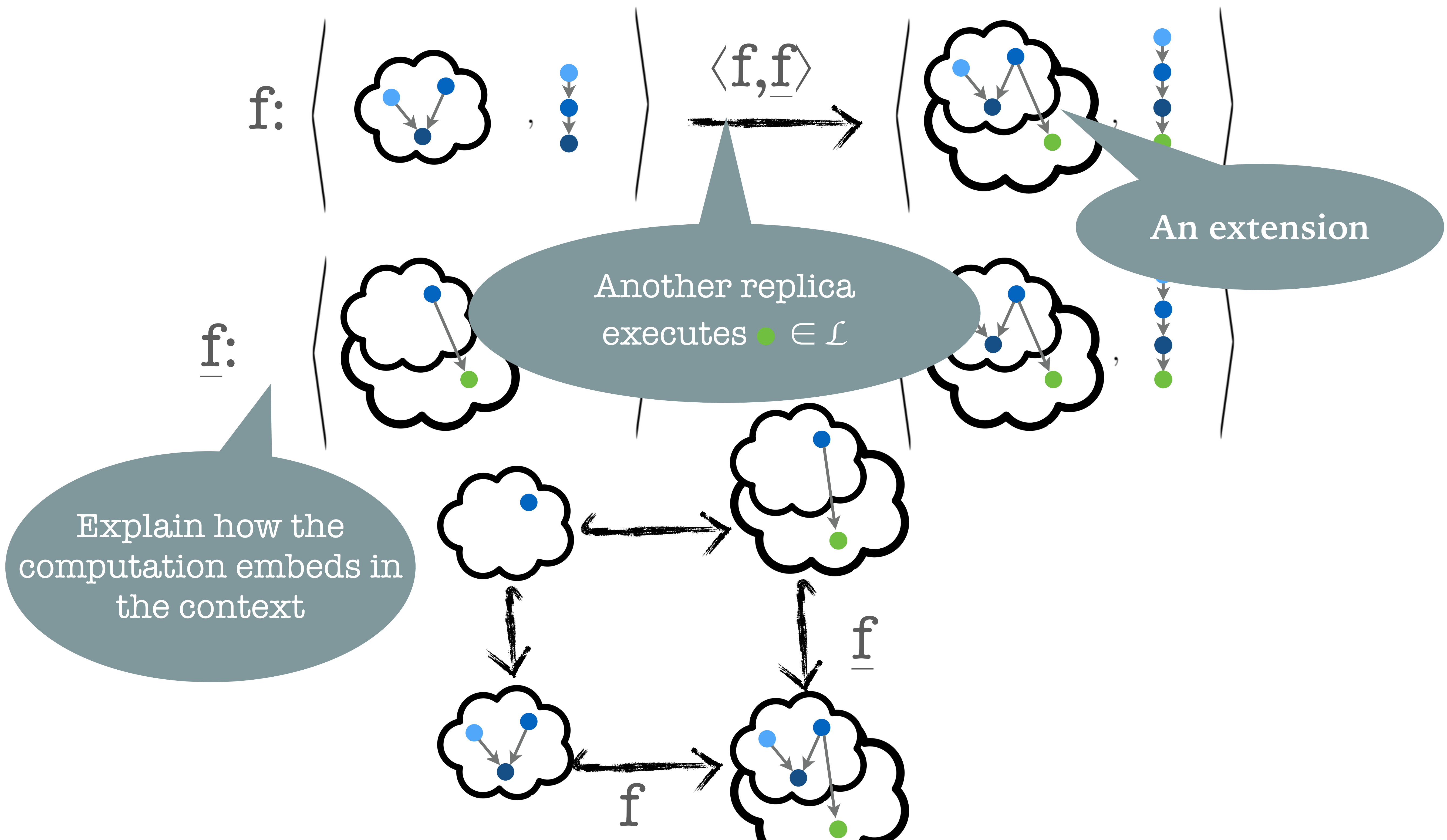
Recovering labels (Leifer-Milner approach)



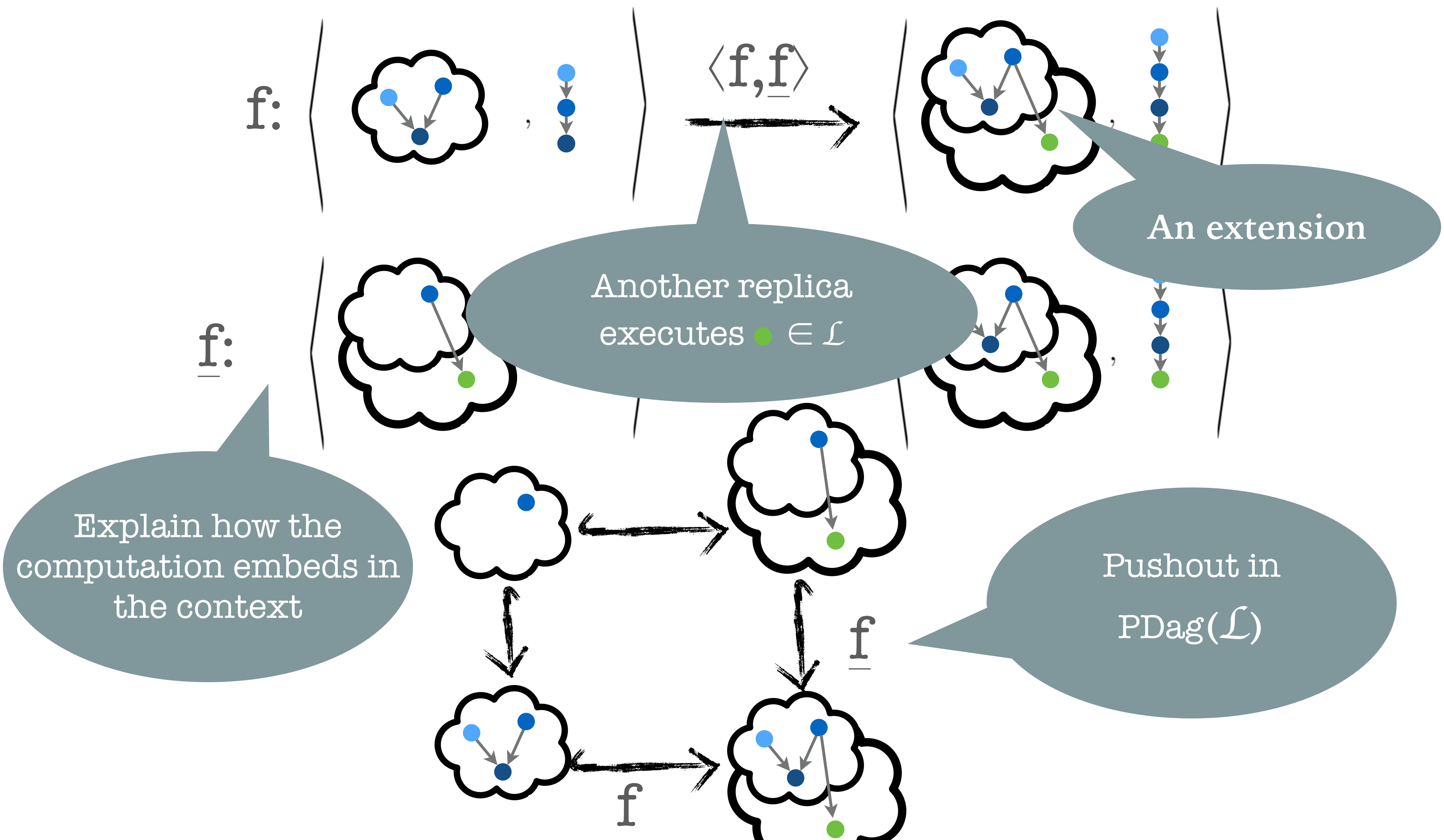
Recovering labels (Leifer-Milner approach)



Recovering labels (Leifer-Milner approach)



Recovering labels (Leifer-Milner approach)



Replica LTSs, requirement

$$\mathcal{Q} = \langle \Sigma, \oplus, \mathbf{1}, \rightarrow \rangle$$

Replica LTSs, requirement

$$\mathcal{Q} = \langle \Sigma, \oplus, \mathbf{1}, \rightarrow \rangle$$

$$\frac{}{\sigma \xrightarrow{l} \sigma'}$$

(set of) axioms

Replica LTSs, requirement

$$\mathcal{Q} = \langle \Sigma, \oplus, \mathbf{1}, \rightarrow \rangle$$

(set of) axioms

$$\frac{}{\sigma \xrightarrow{l} \sigma'}$$

closure

$$\frac{}{\sigma \xrightarrow{\sigma'} \sigma \oplus \sigma'}$$

$$\frac{\sigma_1 \xrightarrow{l} \sigma_1' \qquad \sigma_1 \xrightarrow{l} \sigma_2'}{\sigma_1 \oplus \sigma_2 \xrightarrow{l} \sigma_1' \oplus \sigma_2'}$$

Replica LTSs, requirement

$$\mathcal{Q} = \langle \Sigma, \oplus, \mathbf{1}, \rightarrow \rangle$$

(set of) axioms

$$\frac{}{\sigma \xrightarrow{l} \sigma'}$$

closure

$$\frac{}{\sigma \xrightarrow{\sigma'} \sigma \oplus \sigma'} \qquad \frac{\sigma_1 \xrightarrow{l} \sigma_1' \qquad \sigma_1 \xrightarrow{l} \sigma_2'}{\sigma_1 \oplus \sigma_2 \xrightarrow{l} \sigma_1' \oplus \sigma_2'}$$

plus a decomposition requirement for $\sigma_1 \oplus \sigma_2 \xrightarrow{l} \sigma'$

(Part of an) Implementation of a replicated counter

$$\mathcal{Q}_r = \langle \mathbb{N}^{\mathcal{R}}, \mathtt{max}, 0, \rightarrow_r \rangle$$

states are functions from replicas to the naturals

(Part of an) Implementation of a replicated counter

$$\mathcal{Q}_r = \langle \mathbb{N}^{\mathcal{R}}, \mathtt{max}, 0, \rightarrow_r \rangle$$

states are functions from replicas to the naturals

$$\frac{}{\sigma \xrightarrow{\langle inc,ok \rangle}_r \sigma[^{\sigma(r)+1}/_r]}$$

the axiom updates the value of its replica

(Part of an) Implementation of a replicated counter

$$\mathcal{Q}_r = \langle \mathbb{N}^{\mathcal{R}}, \mathtt{max}, 0, \rightarrow_r \rangle$$

states are functions from replicas to the naturals

$$\frac{}{\sigma \xrightarrow{\langle inc, ok \rangle}_r \sigma[{}^{\sigma(r)+1}/_r]}$$

the axiom updates the value of its replica

$$\frac{}{\sigma \xrightarrow{\sigma'}_r \mathtt{max}(\sigma, \sigma')}$$

point-wise updates of all replicas

(Part of an) Implementation of a replicated counter

$$\mathcal{Q}_r = \langle \mathbb{N}^{\mathcal{R}}, \texttt{max}, 0, \rightarrow_r \rangle$$

states are functions from replicas to the naturals

$$\frac{}{\sigma \xrightarrow{\langle inc, ok \rangle}_r \sigma[^{\sigma(r)+1}/_r]}$$

the axiom updates the value of its replica

$$\frac{}{\sigma \xrightarrow{\sigma'}_r \texttt{max}(\sigma, \sigma')}$$

point-wise updates of all replicas

plus a decomposition requirement for $\texttt{max}(\sigma_1, \sigma_2) \xrightarrow{\langle inc, ok \rangle}_r \sigma'$

Implementation correctness as simulation

An implementation relation R_S is a relation between states in I_S and C_S such that if $(\sigma, \langle G, P \rangle) \in R_S$ then

1. if $\sigma \xrightarrow{l} \sigma'$ then $\exists \langle G', P' \rangle$ such that $\langle G, P \rangle \xrightarrow{l} \langle G', P' \rangle$ and $(\sigma', \langle G', P' \rangle) \in R_S$
2. if $\sigma \xrightarrow{\sigma'} \sigma''$ then $\exists \langle G', P' \rangle, \langle G'', P'' \rangle$ such that $\langle G, P \rangle \xrightarrow{\langle G', P' \rangle} \langle G'', P'' \rangle$, $(\sigma', \langle G', P' \rangle) \in R_S$, and $(\sigma'', \langle G'', P'' \rangle) \in R_S$

RDT implementation, categorically

- A functor $\mathbf{I} : \mathbf{IR}(\mathcal{R}) \to \mathbf{P}(\mathcal{M}on)$
  - from the category of sequences of operations performed over replicas ($\mathbf{IR}(\mathcal{R})$)
  - to the category of implementation states $\mathbf{P}(\mathcal{M}on)$

Category of sequence of operations

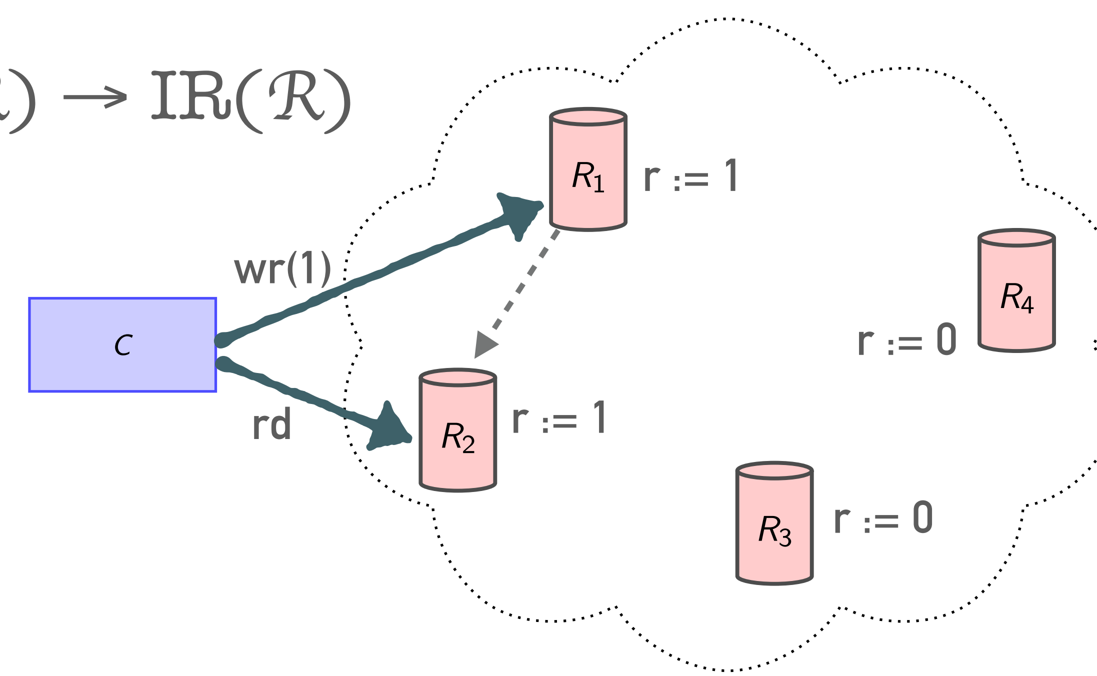
- One replica category **IR**:
  - One object
  - Words over $\mathcal{L}$ as arrows
- Multi replica category $\mathbf{IR}(\mathcal{R})$: $\#\mathcal{R}$ isomorphic copies of **IR**

Category of sequence of operations

- One replica category **IR**:
  - One object
  - Words over $\mathcal{L}$ as arrows
- Multi replica category $\mathbf{IR}(\mathcal{R})$: $\#\mathcal{R}$ isomorphic copies of **IR**

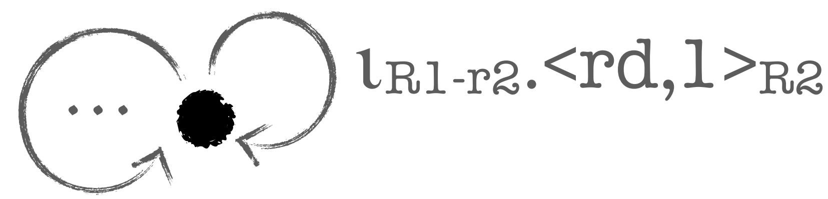
$\langle wr(1),ok\rangle_{R1}.\iota_{R1\text{-}R2}.\langle rd,1\rangle_{R2} : \mathbf{IR}(\mathcal{R}) \to \mathbf{IR}(\mathcal{R})$

a shorthand for
$\langle wr(1),ok\rangle : R_1 \to R_1$



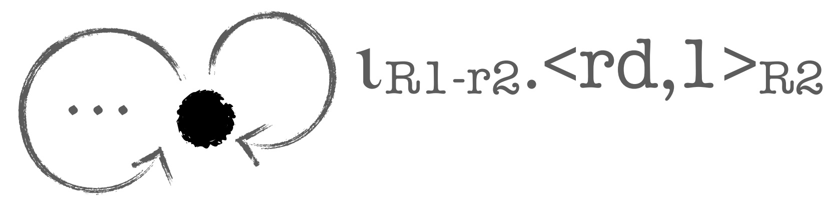
Implementation Functor

$\mathbf{IR}(\mathcal{R})$

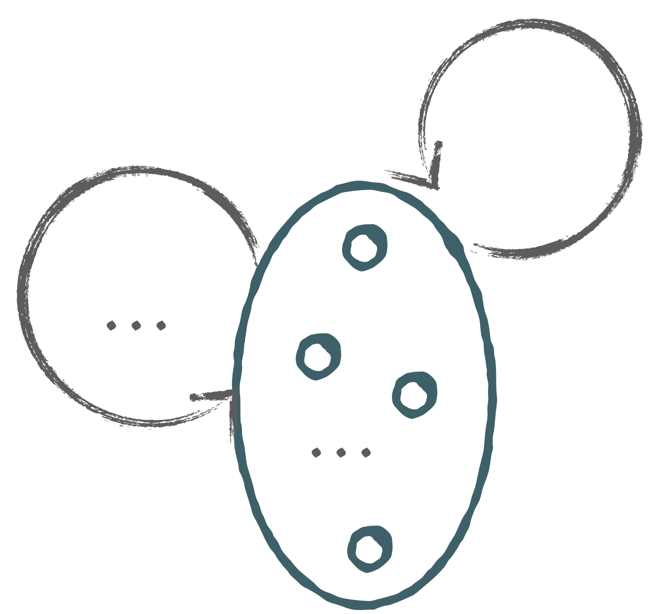


Implementation Functor

IR$(\mathcal{R})$

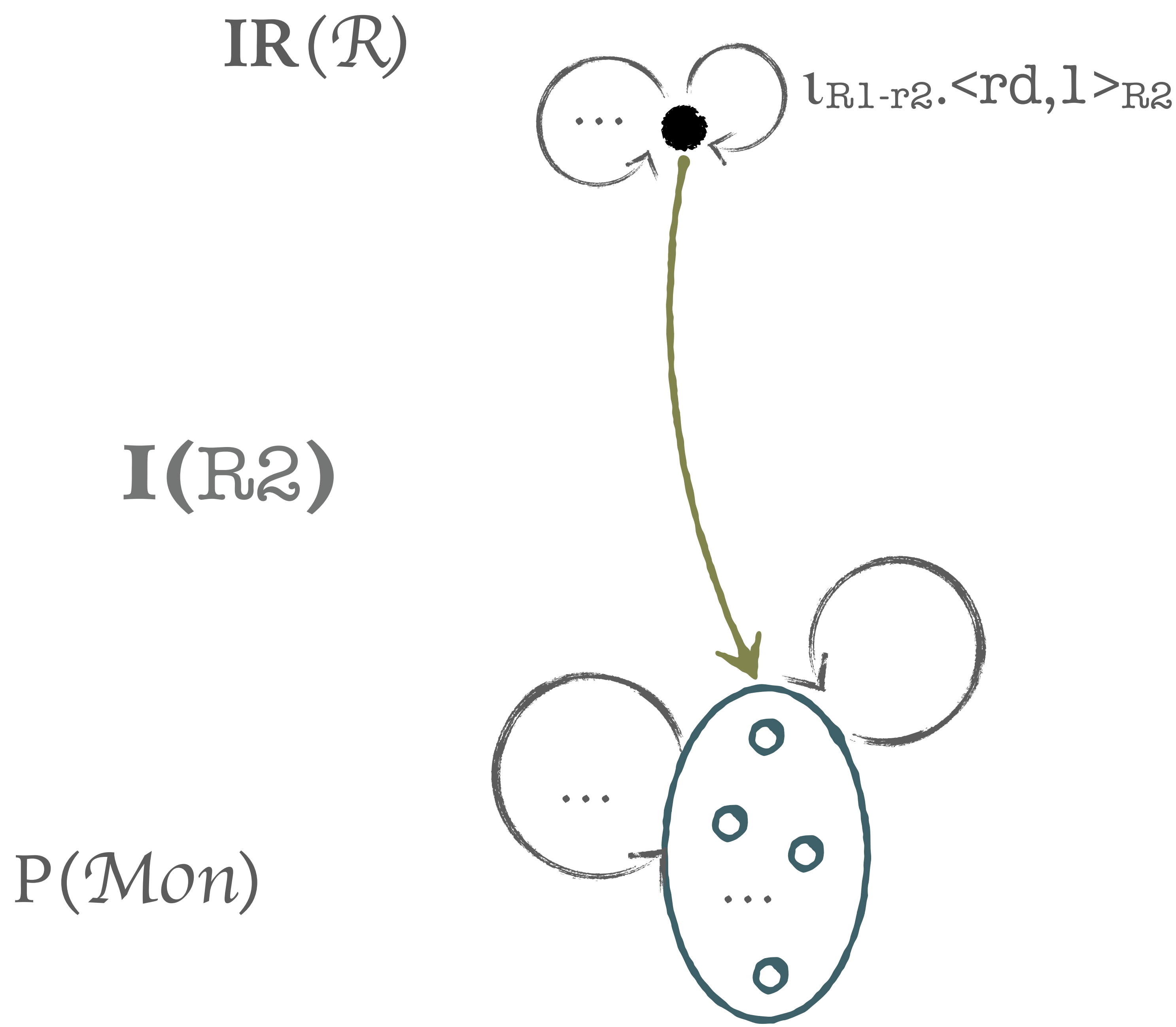


P$(\mathcal{M}on)$



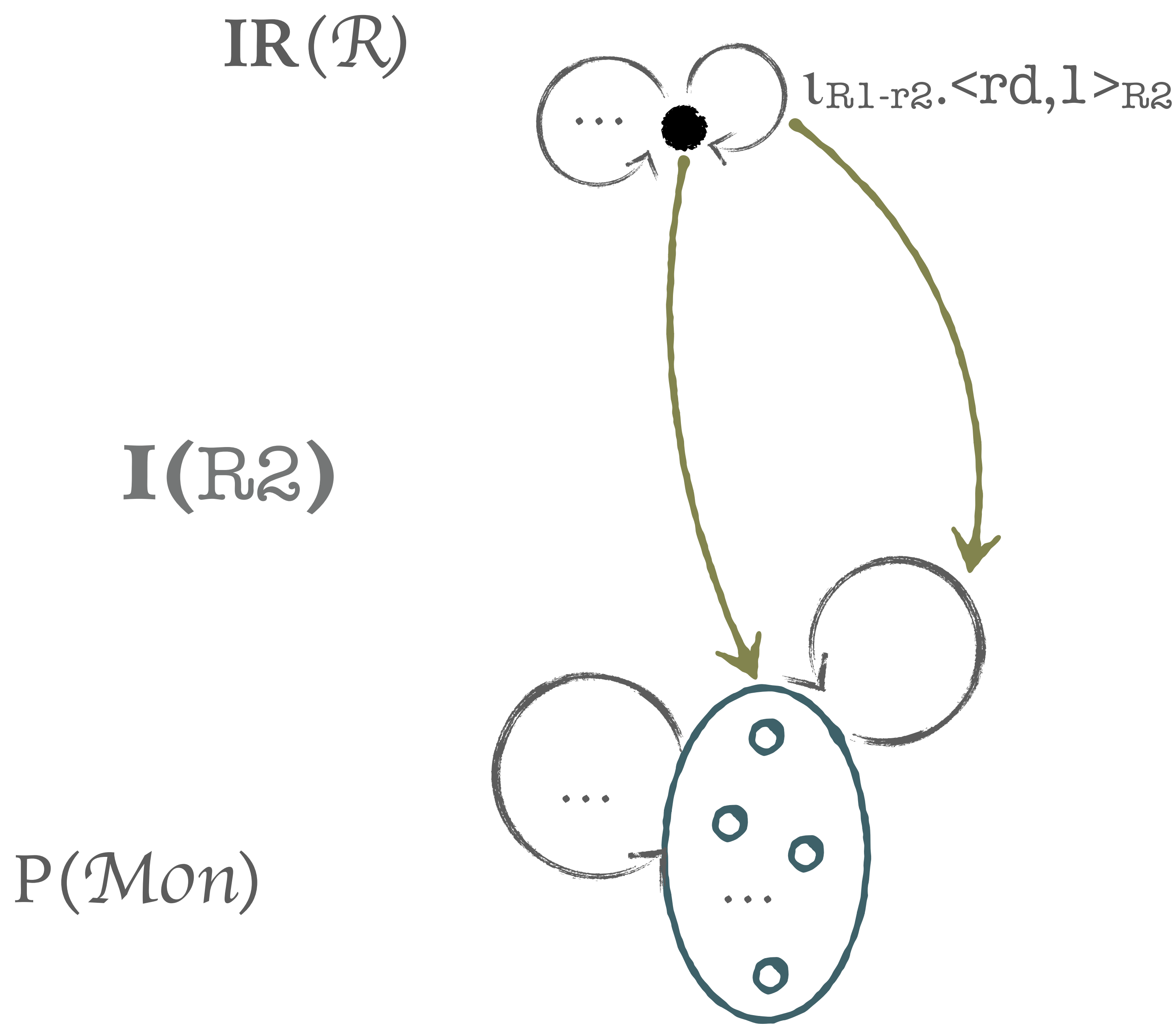
Set of all possible states of a replica

Implementation Functor



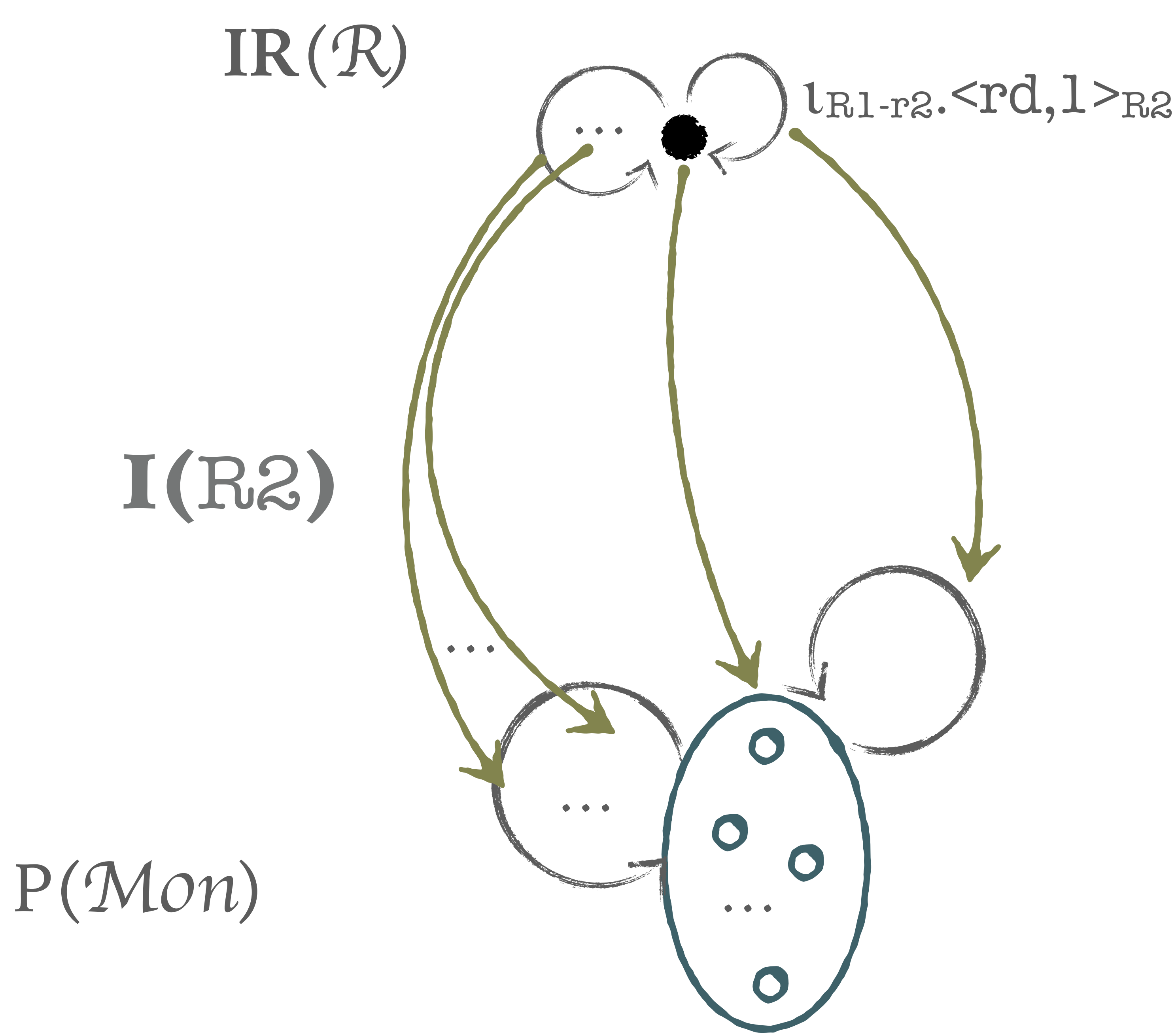
Set of all possible states of a replica

Implementation Functor



Set of all possible states of a replica

Implementation Functor



Set of all possible states of a replica

Final words

- We have provided
  - an algebraic characterisation of the specification and (state-based) implementation of RDTs
  - a notion of implementation correctness in terms of (higher-order) simulation
- The approach suffices to model various well-known RDTs
  - We did not consider labels for snd operations because most RDTs implementation communicate full copies of their state
- The approach does not cover operation-based implementations