Consistency Criteria for Distributed Data Structures

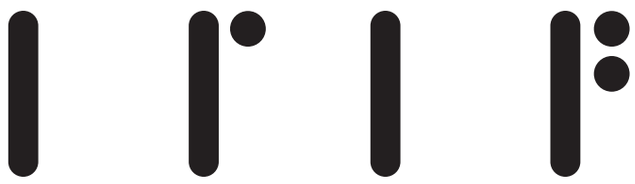
Constantin Enea

IRIF, University Paris Diderot

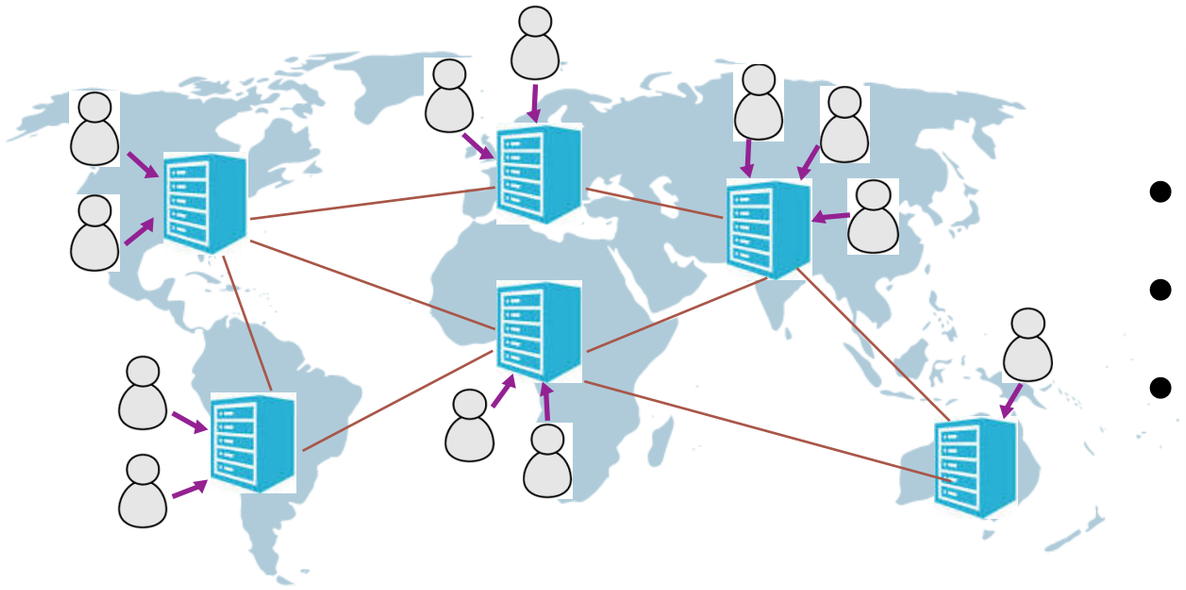
joint work with Ahmed Bouajjani, Rachid Guerraoui, Jad Hamza

European Research Council

Established by the European Commission



Geo-Distributed Software Systems

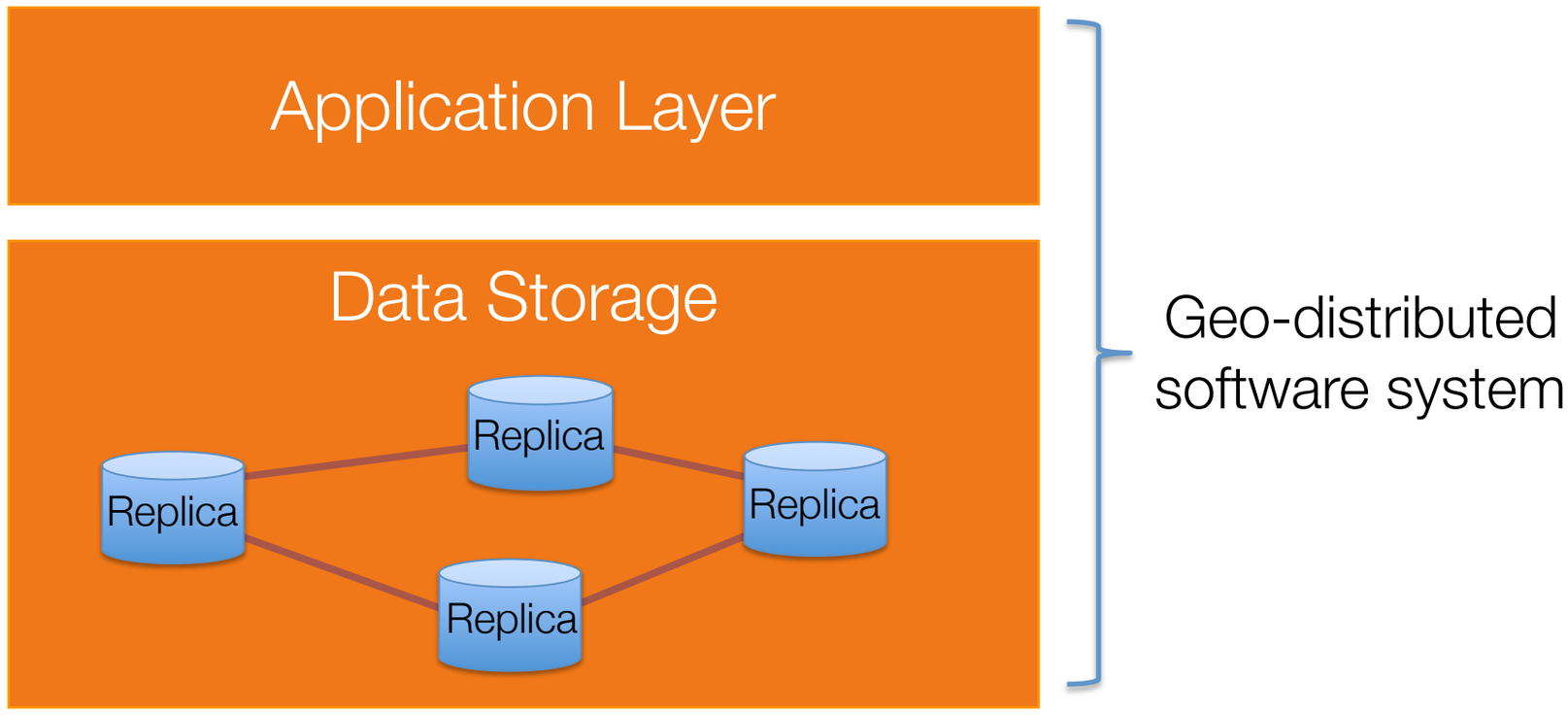


- large numbers of users
- distributed across wide geographical areas
- generate and access huge amounts of data

Examples:

- online markets (Amazon)
- government services (tax payment)
- enterprise customer services
- social networks (Facebook)
- managing connected devices and sensors (Internet of Things)

Distributed Data Storage (Distributed Data Structures)



- to support failures, data is **replicated**
- for availability, replicas may store different versions of data: **weak consistency**
- interface restricted to a **fixed set of methods**
  - key-value stores provide put(key,value) and get(key)
- weak consistency and restricted interface: different approach than relational databases

Current Status

Data storage layer

- informal and vague specifications
- lack of rigorous methodologies for validation
  - testing the behavior of the system under stress
- dormant bugs with potential severe consequences (millions of users)
  - used in critical systems: developing drugs, managing medical equipment, etc.

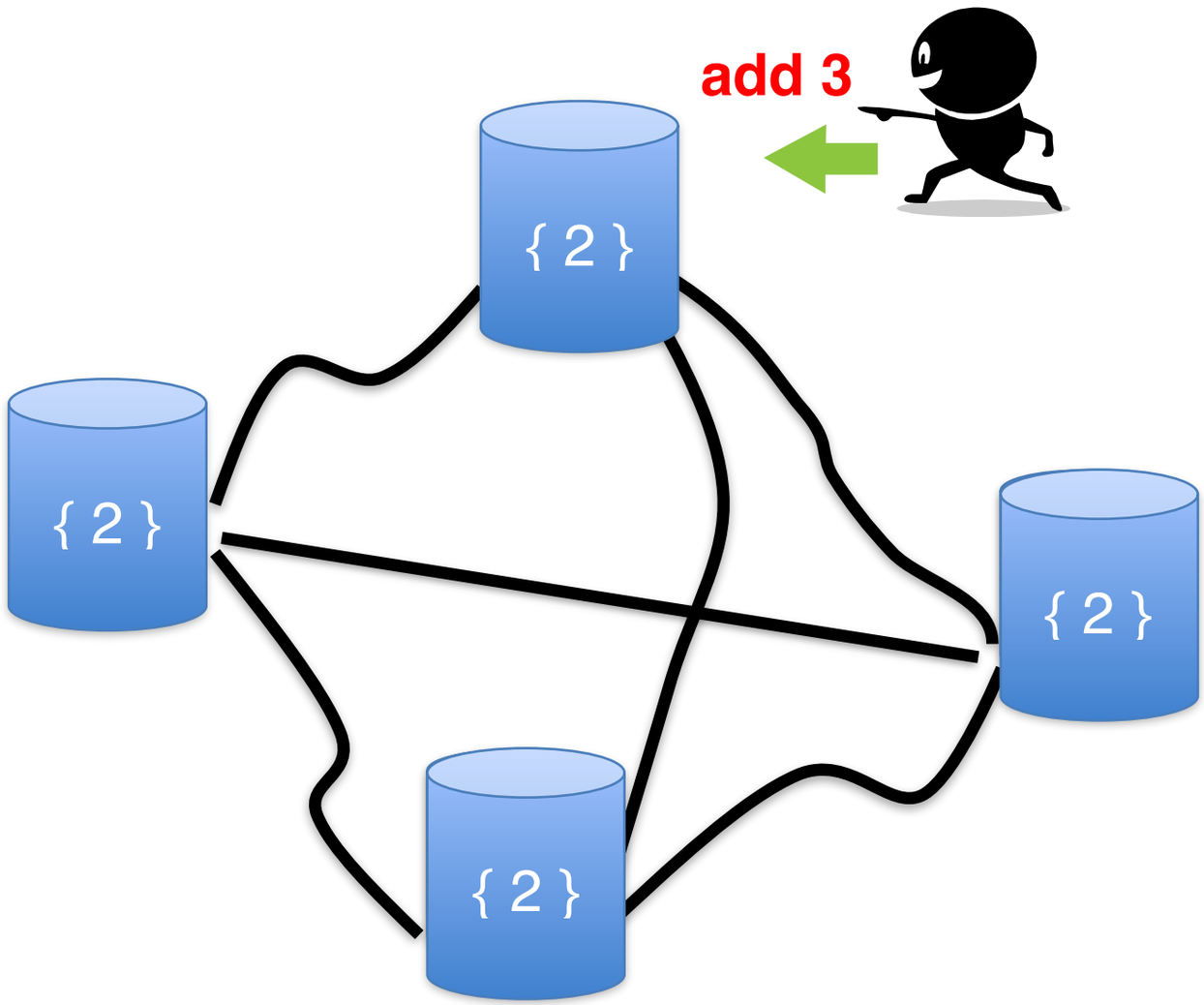
Application layer

- lack of programming abstractions of data storage while developing applications
  - no compositional reasoning
- complex and fragile software systems

Optimistic Data Replication

Optimistic replication: replicas are allowed to diverge

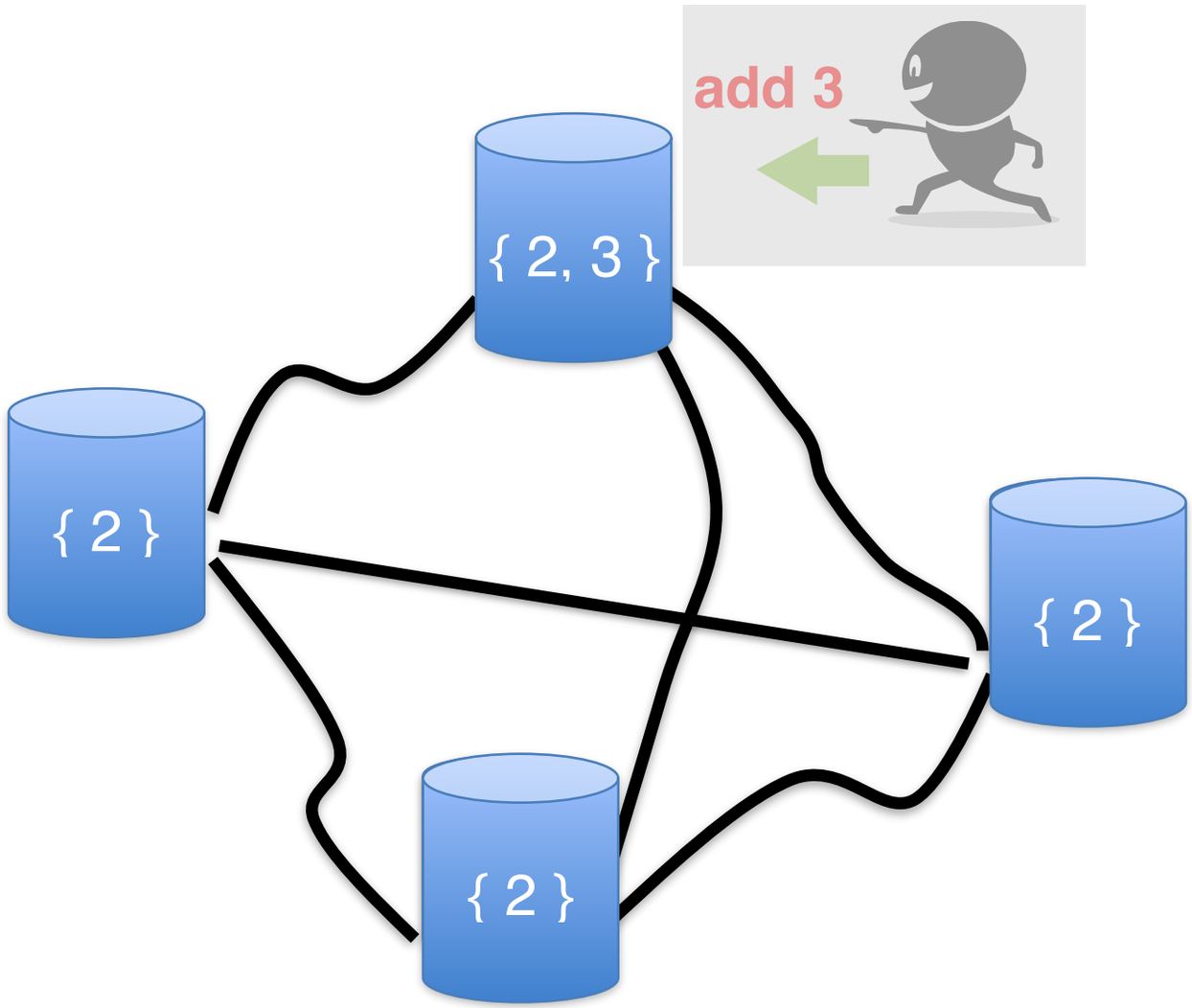
- operations are applied immediately at the submission site



Optimistic Data Replication

Optimistic replication: replicas are allowed to diverge

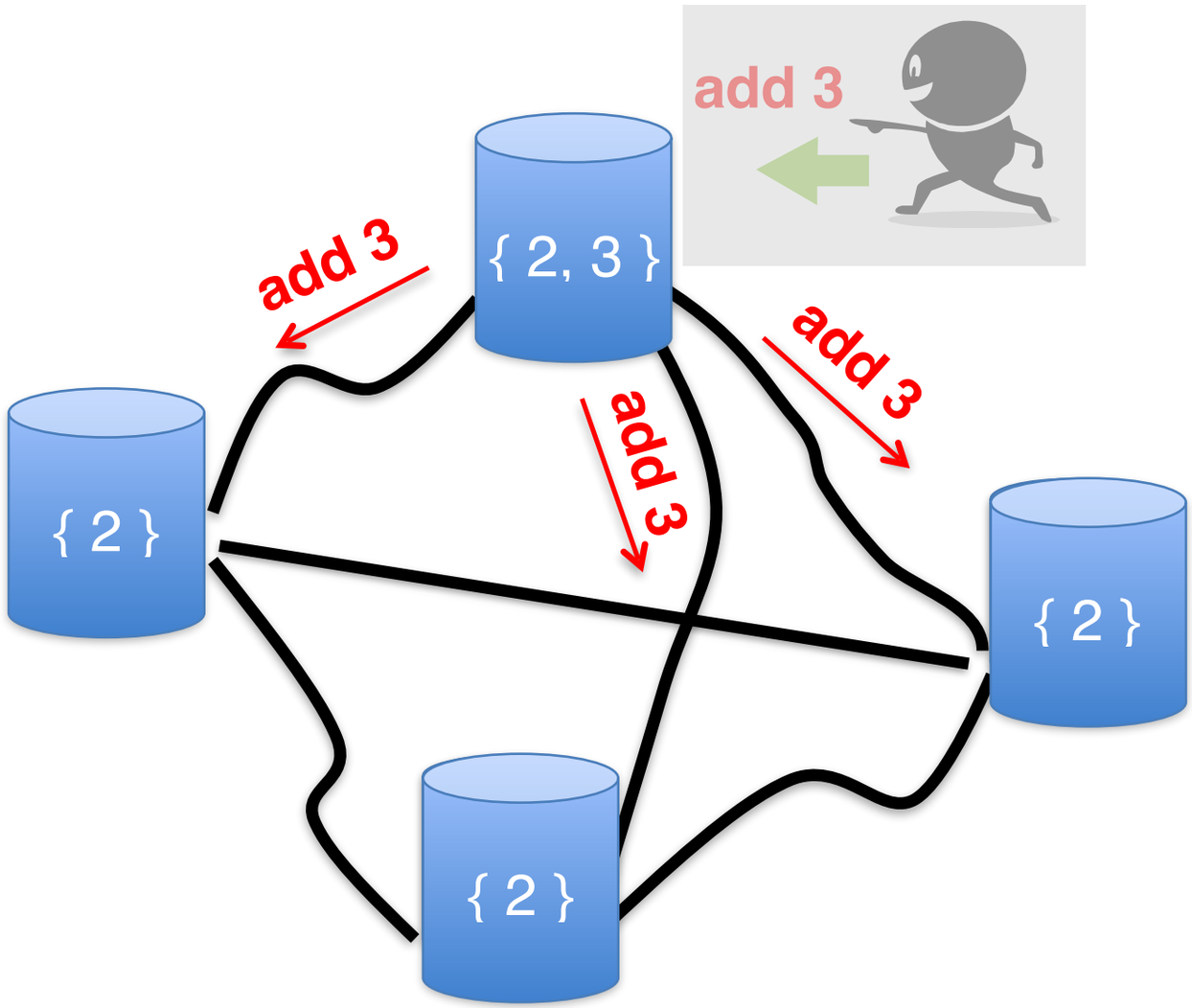
- operations are applied immediately at the submission site



Optimistic Data Replication

Optimistic replication: replicas are allowed to diverge

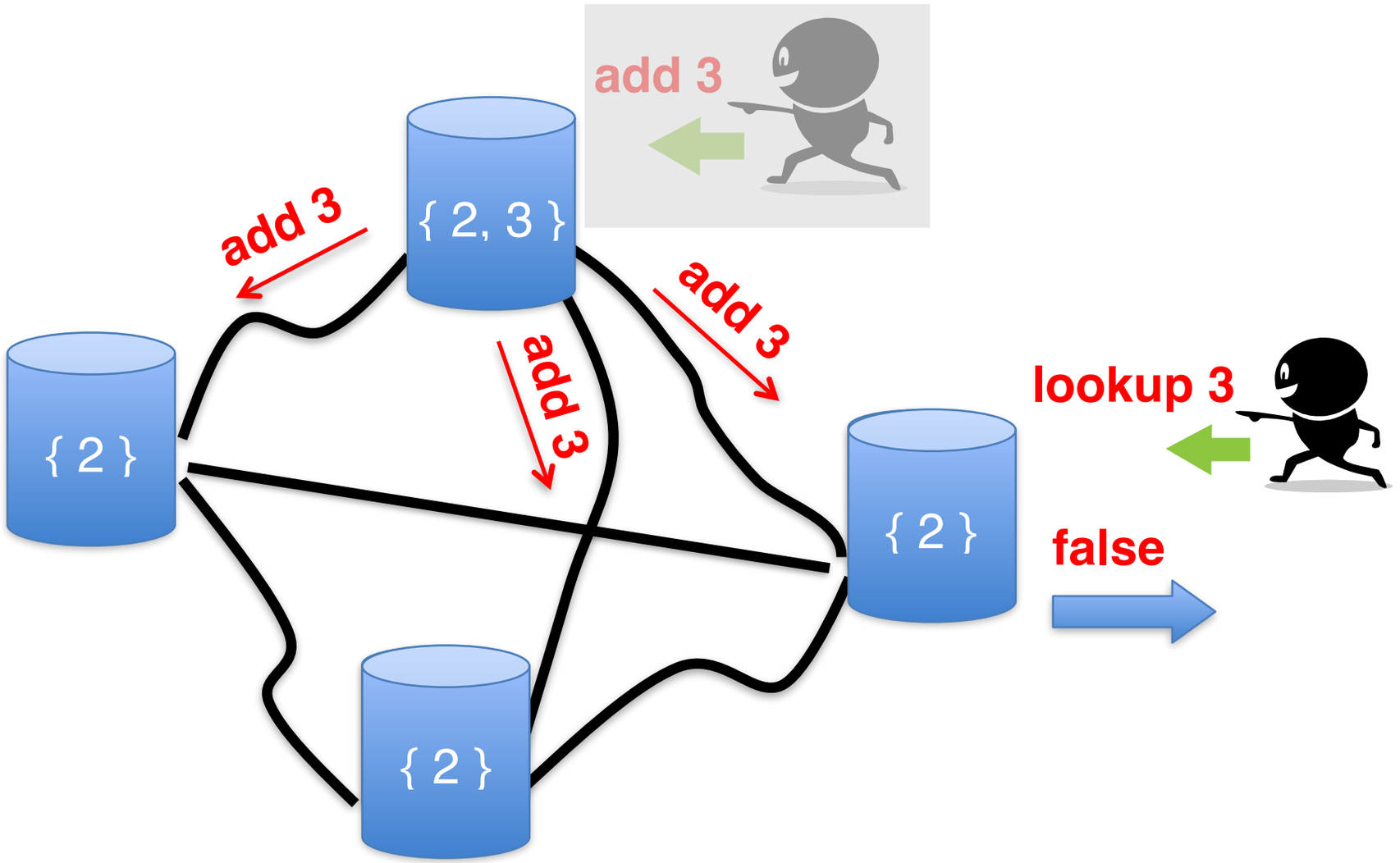
- operations are applied immediately at the submission site
- in the background, sites exchange and apply remote operations



Optimistic Data Replication

Optimistic replication: replicas are allowed to diverge

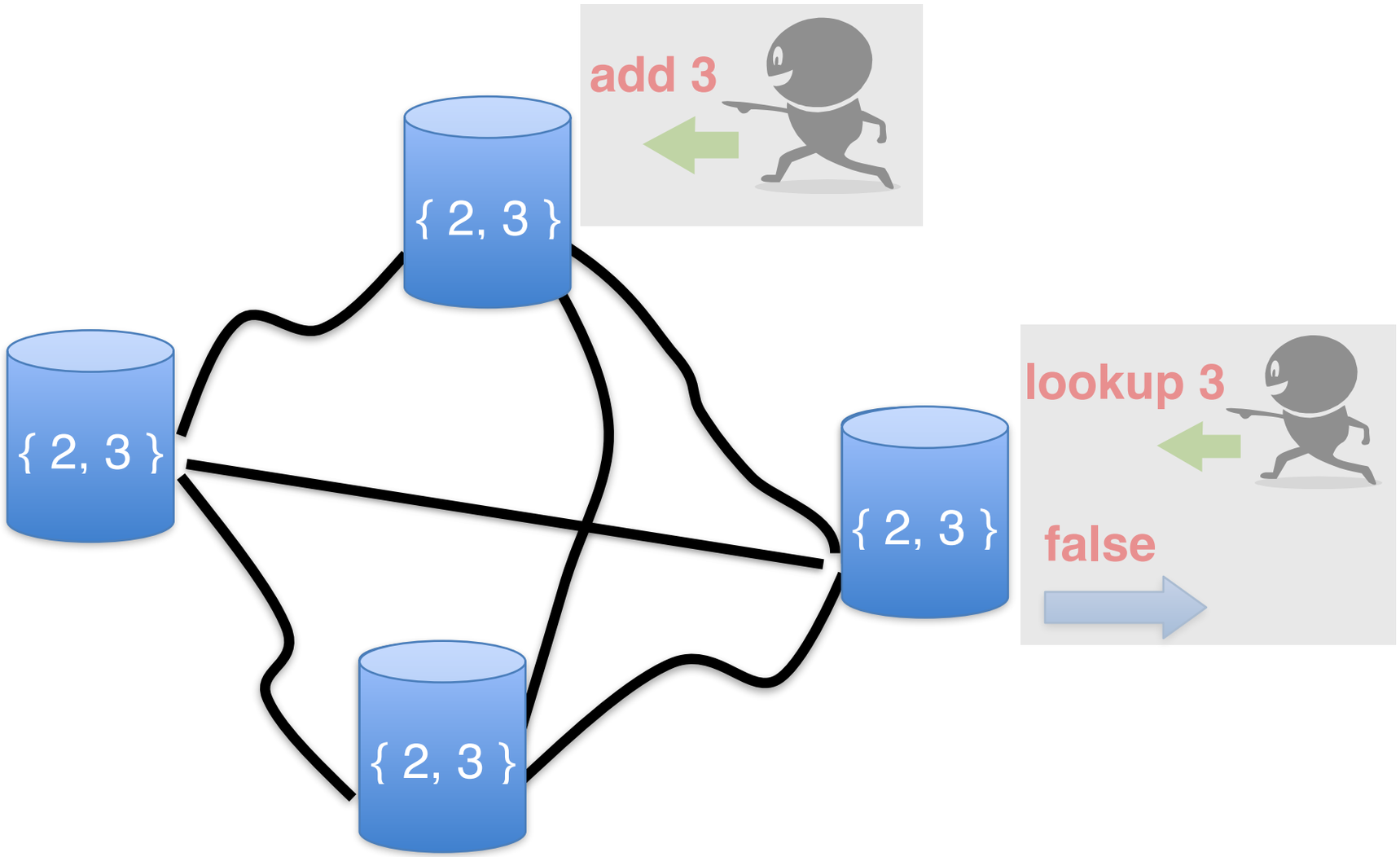
- operations are applied immediately at the submission site
- in the background, sites exchange and apply remote operations



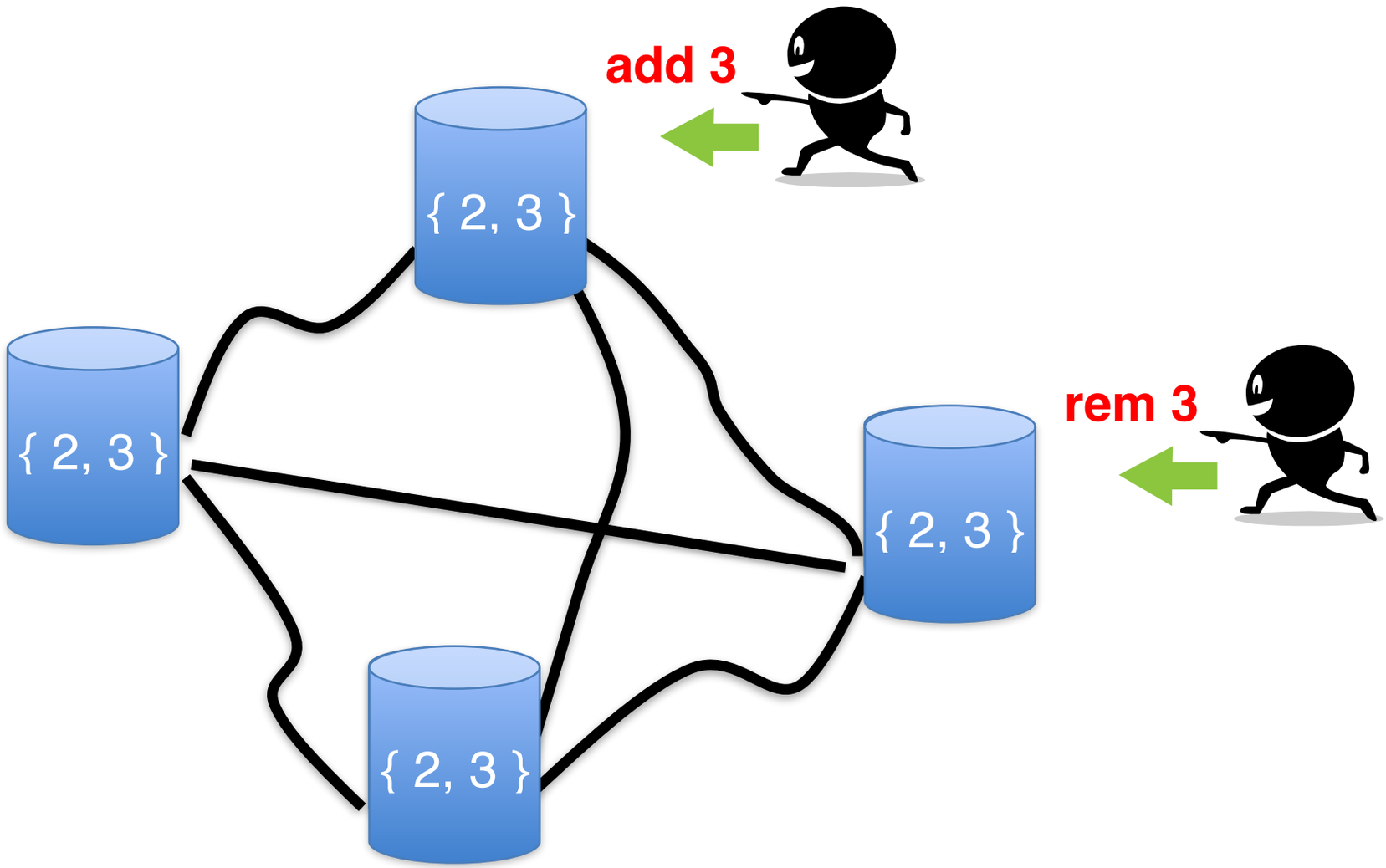
Optimistic Data Replication

Optimistic replication: replicas are allowed to diverge

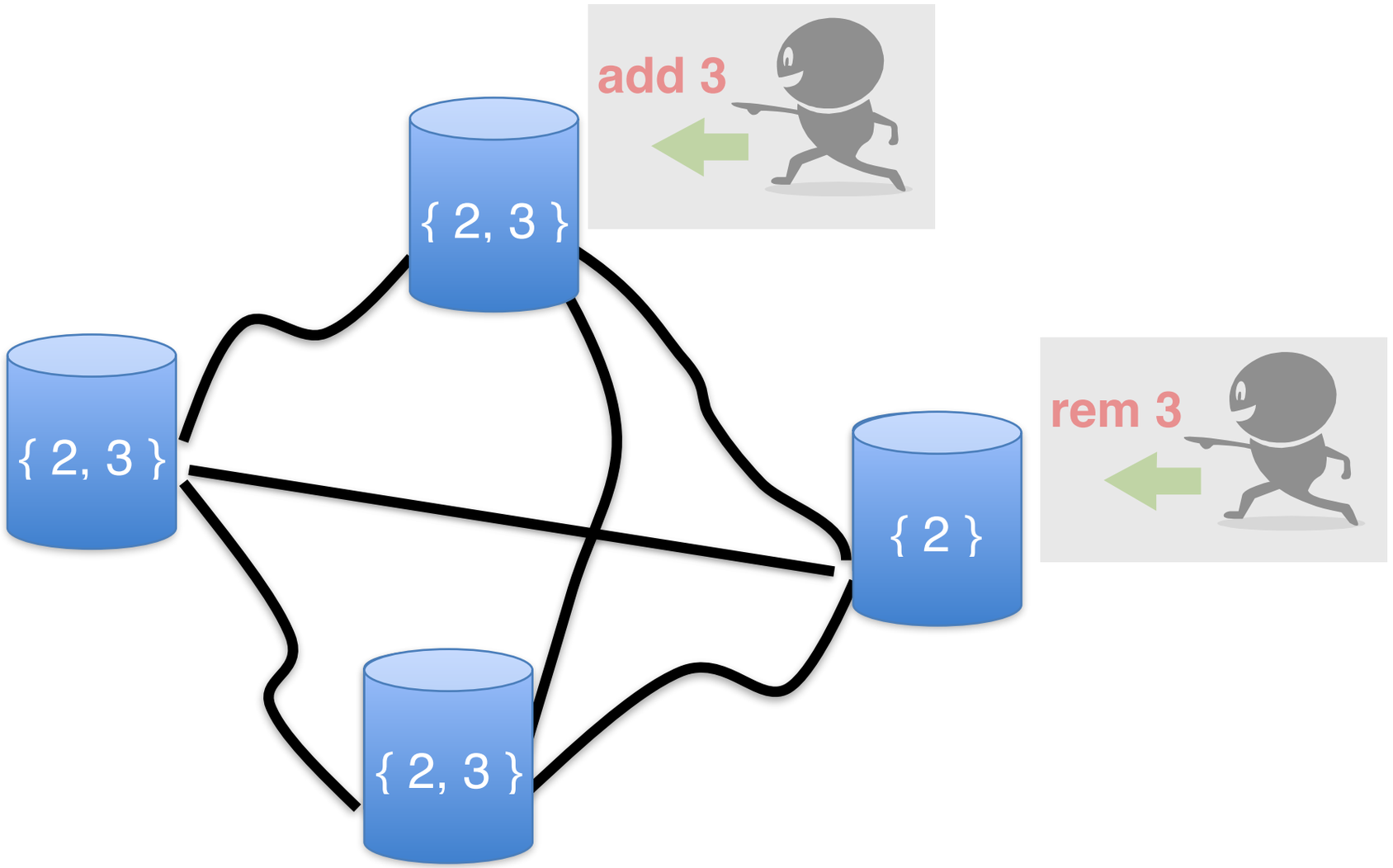
- operations are applied immediately at the submission site
- in the background, sites exchange and apply remote operations



Concurrent Operations

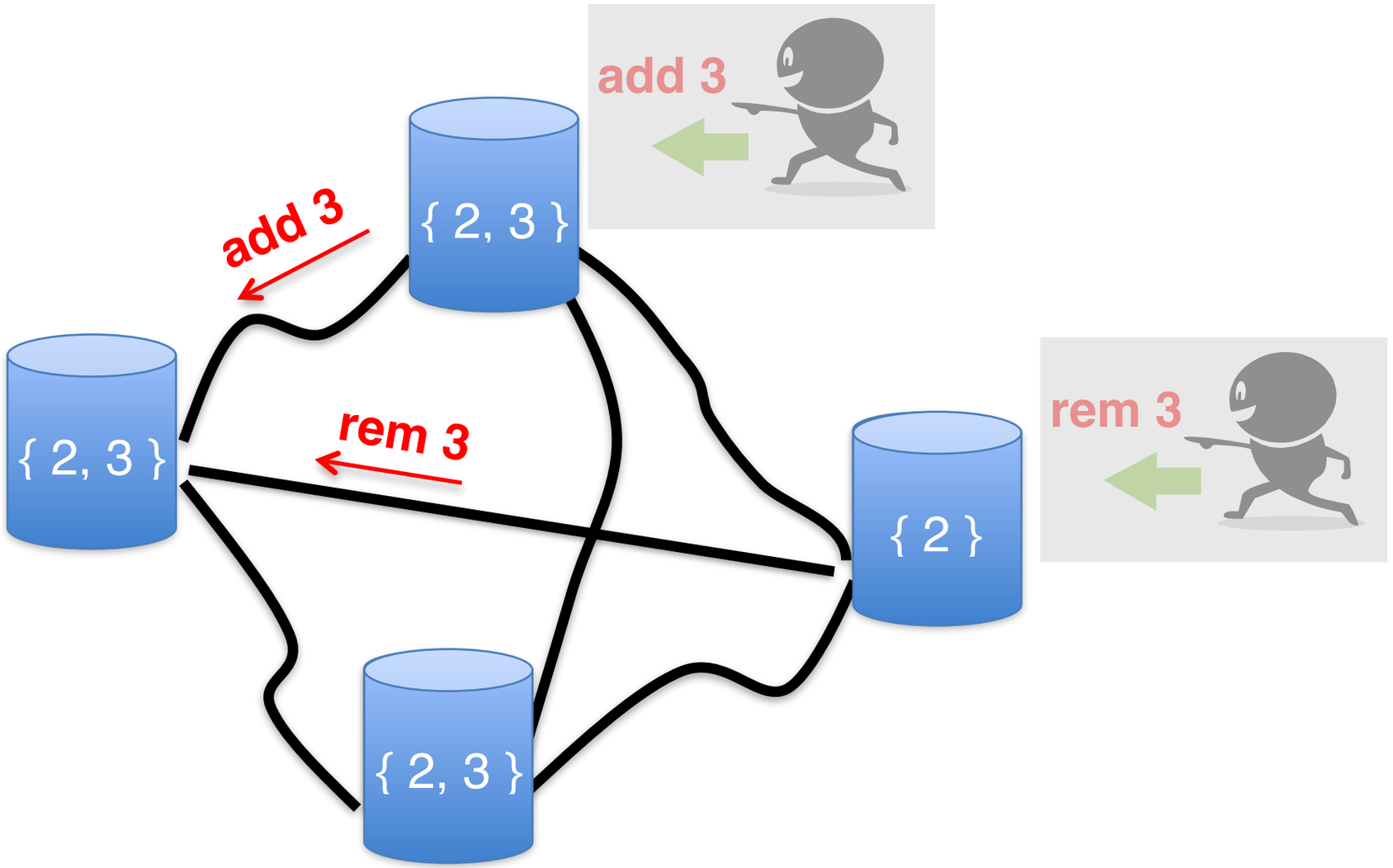


Concurrent Operations



Concurrent Operations

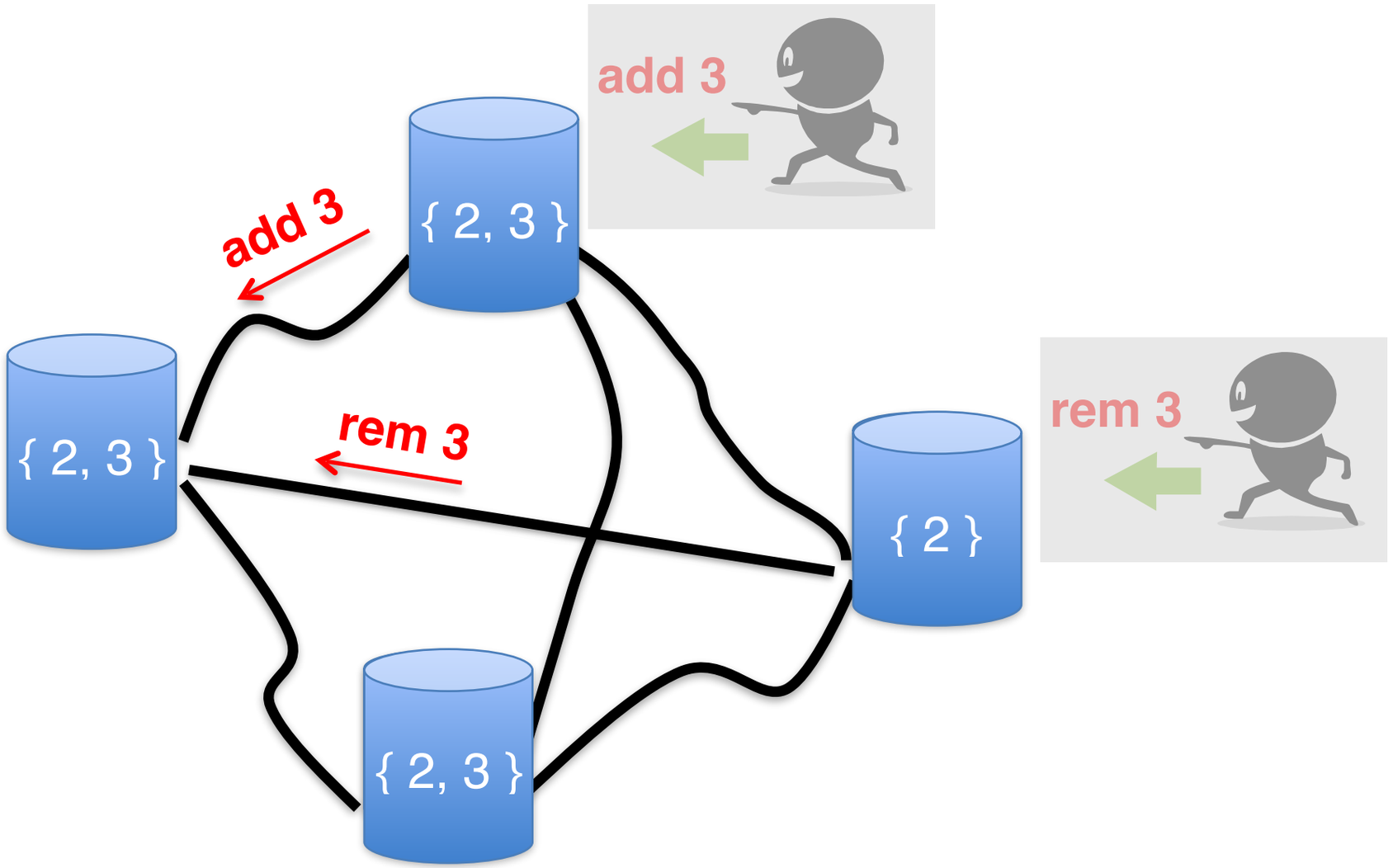
Solving conflicts between concurrent operations



Concurrent Operations

Solving conflicts between concurrent operations

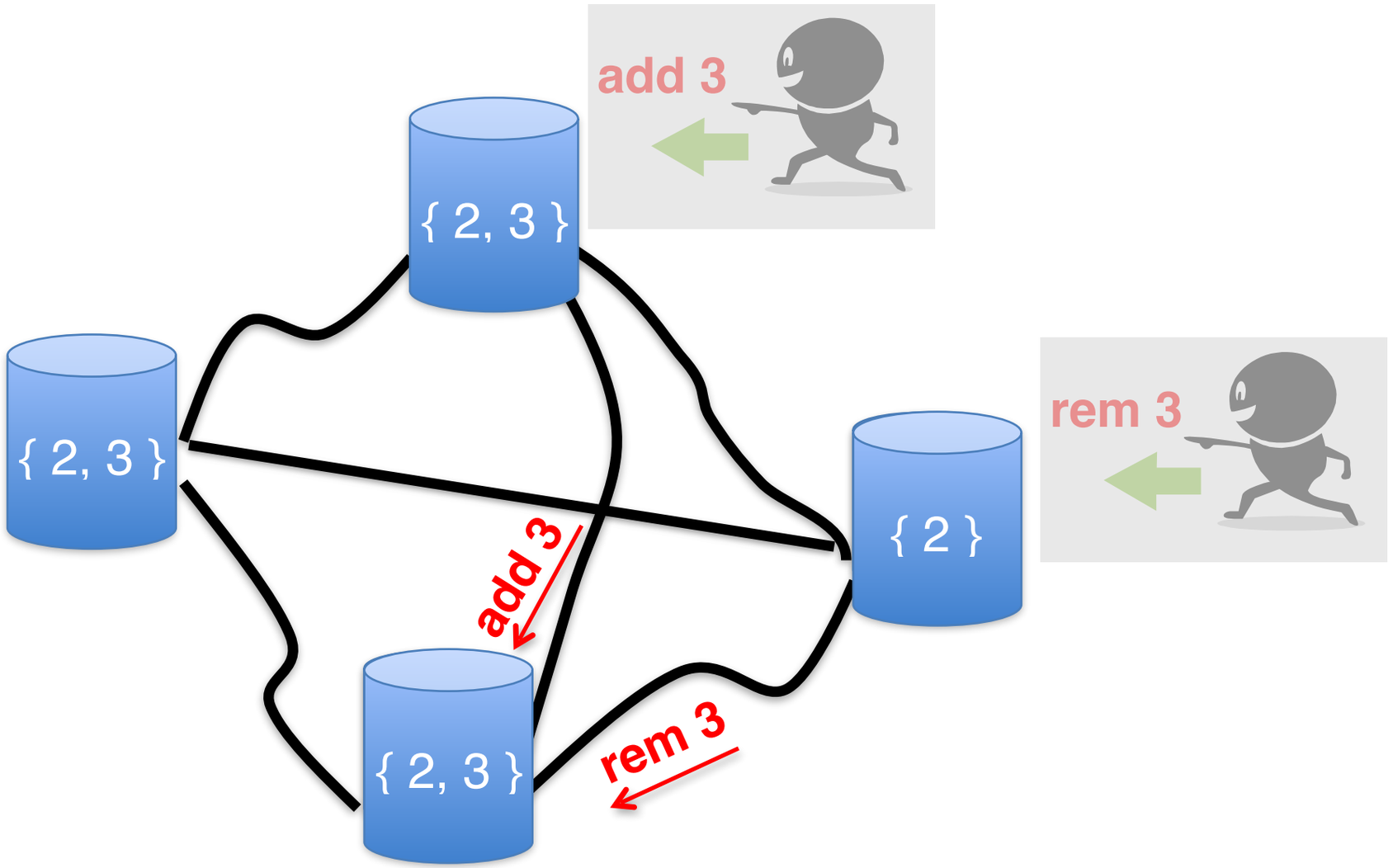
- **speculate and roll-back**, e.g., Google App Engine Datastore



Concurrent Operations

Solving conflicts between concurrent operations

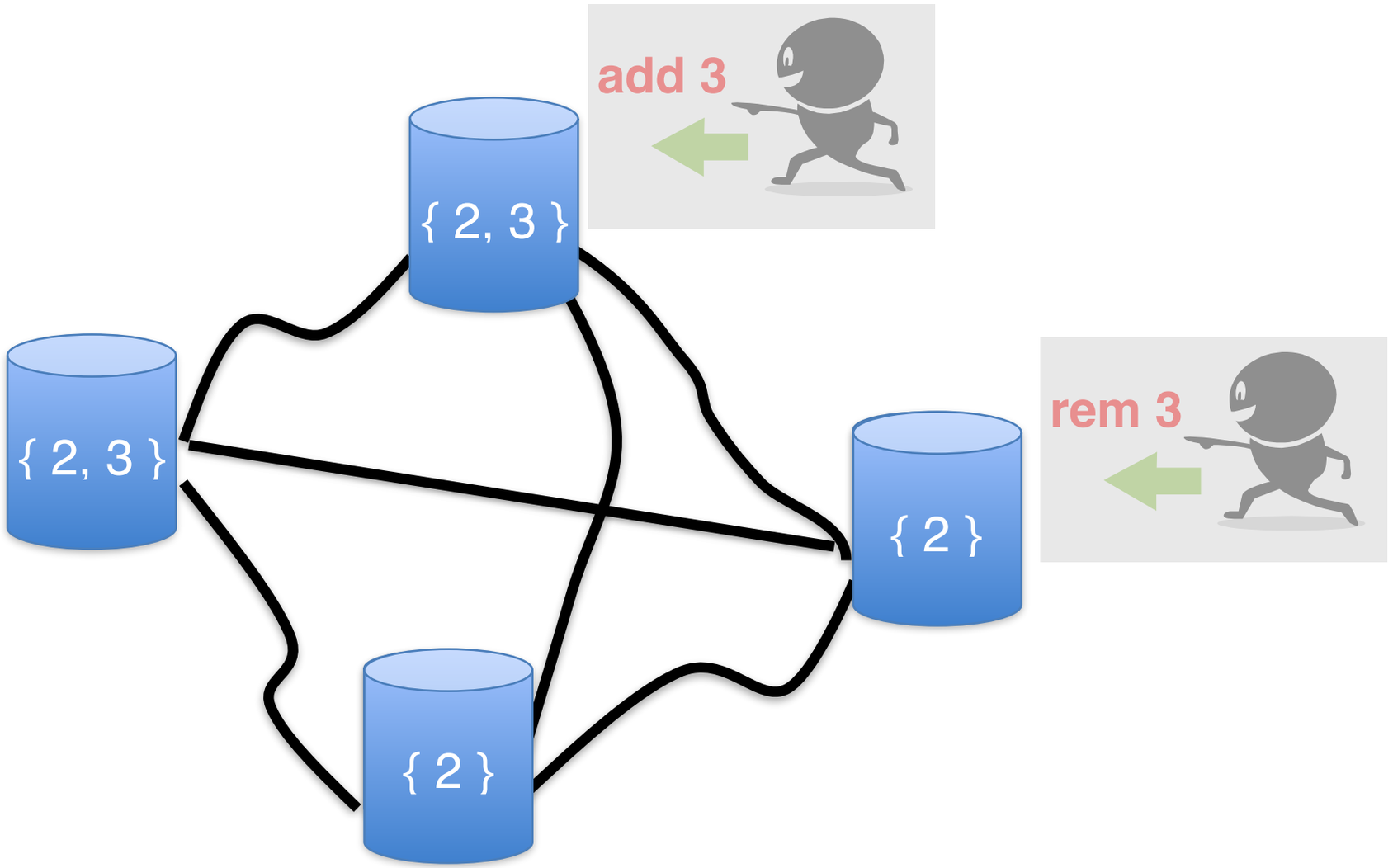
- **speculate and roll-back**, e.g., Google App Engine Datastore



Concurrent Operations

Solving conflicts between concurrent operations

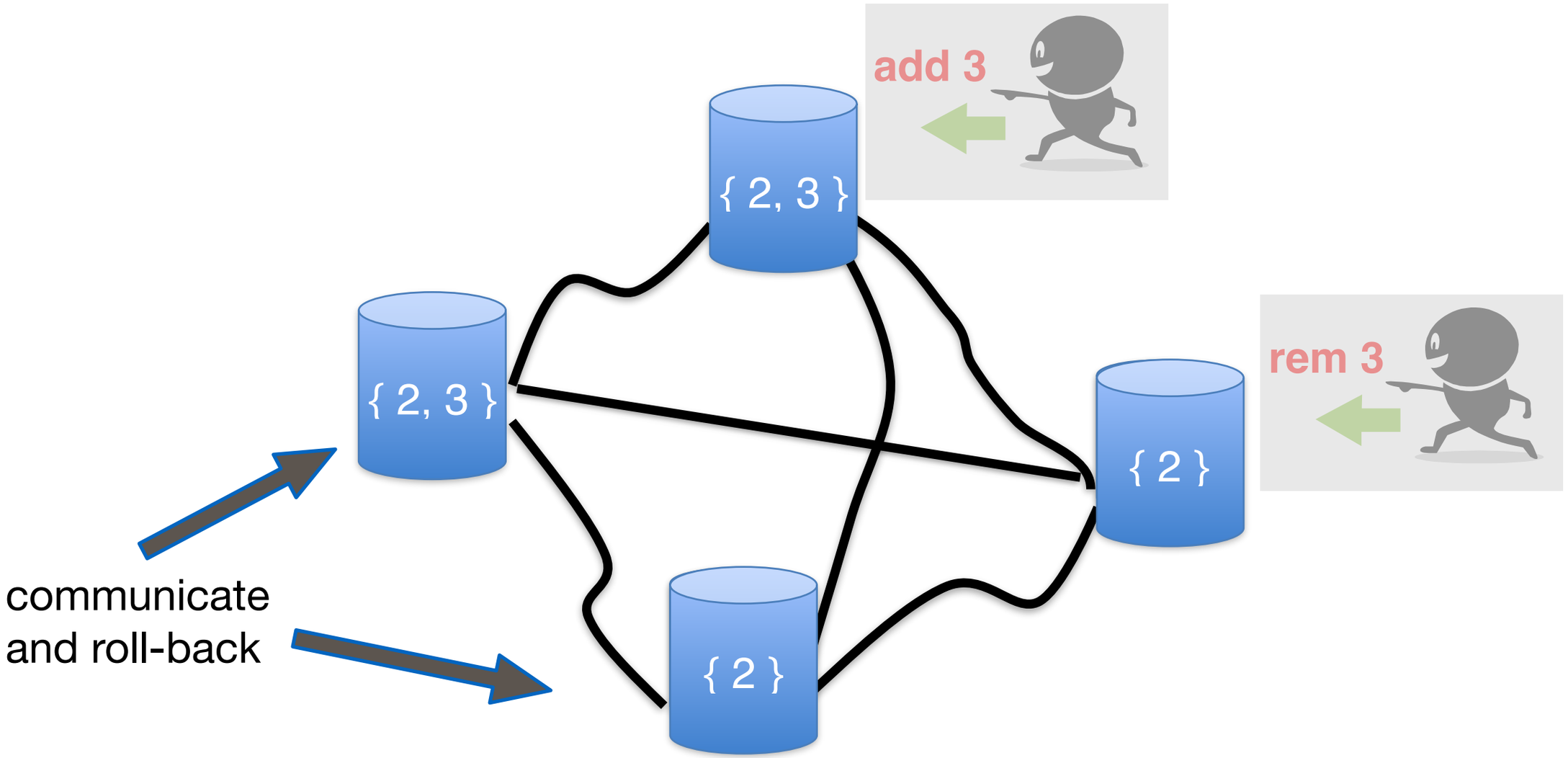
- **speculate and roll-back**, e.g., Google App Engine Datastore



Concurrent Operations

Solving conflicts between concurrent operations

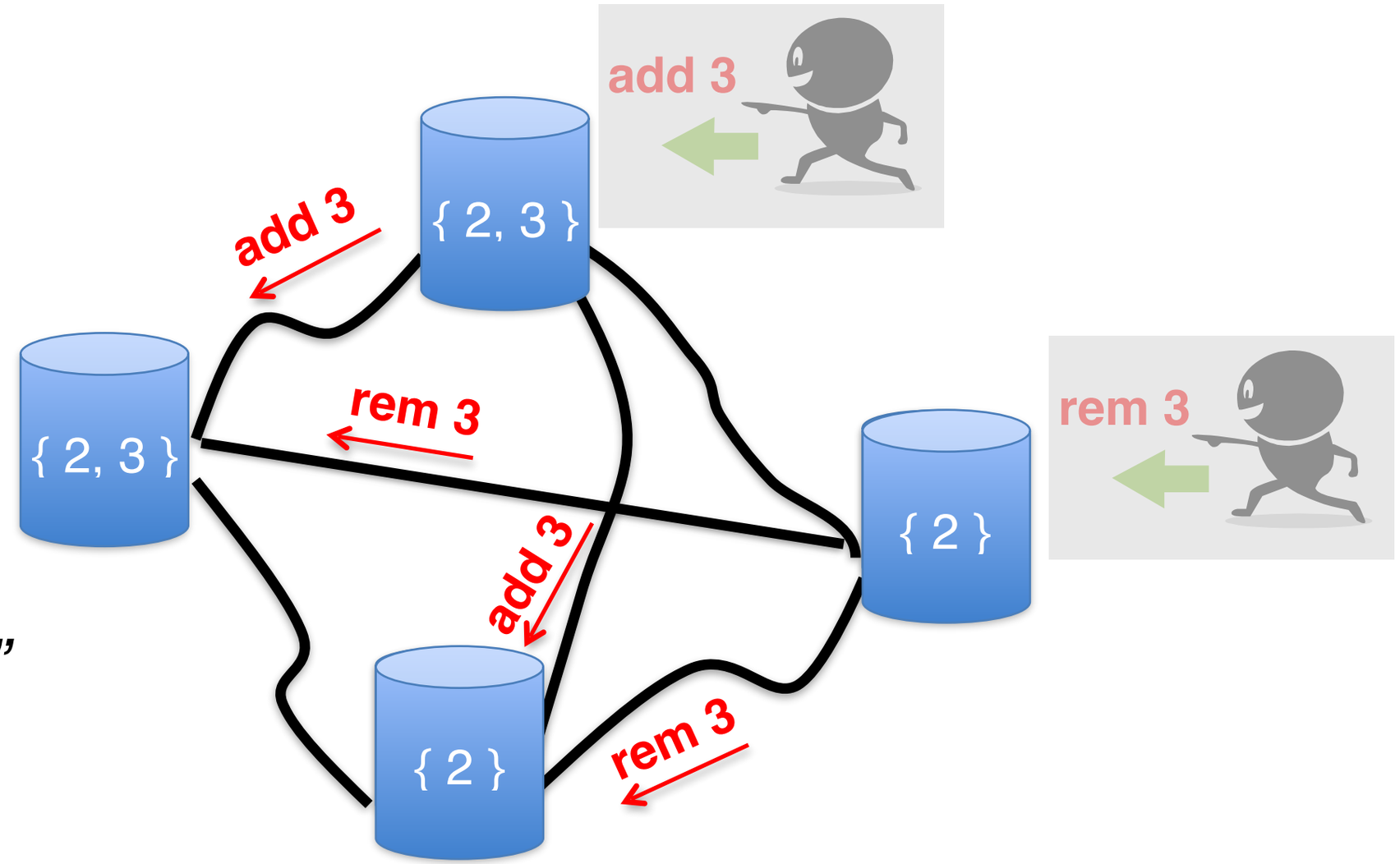
- **speculate and roll-back**, e.g., Google App Engine Datastore



Concurrent Operations

Solving conflicts between concurrent operations

- **speculate and roll-back**, e.g., Google App Engine Datastore
- **convergent conflict resolution**, e.g., CRDTs [Shapiro et al.'11]



"add wins"

Formal verification of DDSs

- Correct operations ? Allowed level of consistency between replicas ?
  - by CAP theorem [Gilbert et al.'02], achieving strong consistency (linearizability) is impossible
  - various correctness criteria: eventual consistency, causal consistency, etc.

- Developing algorithmic methods for the verification of these criteria

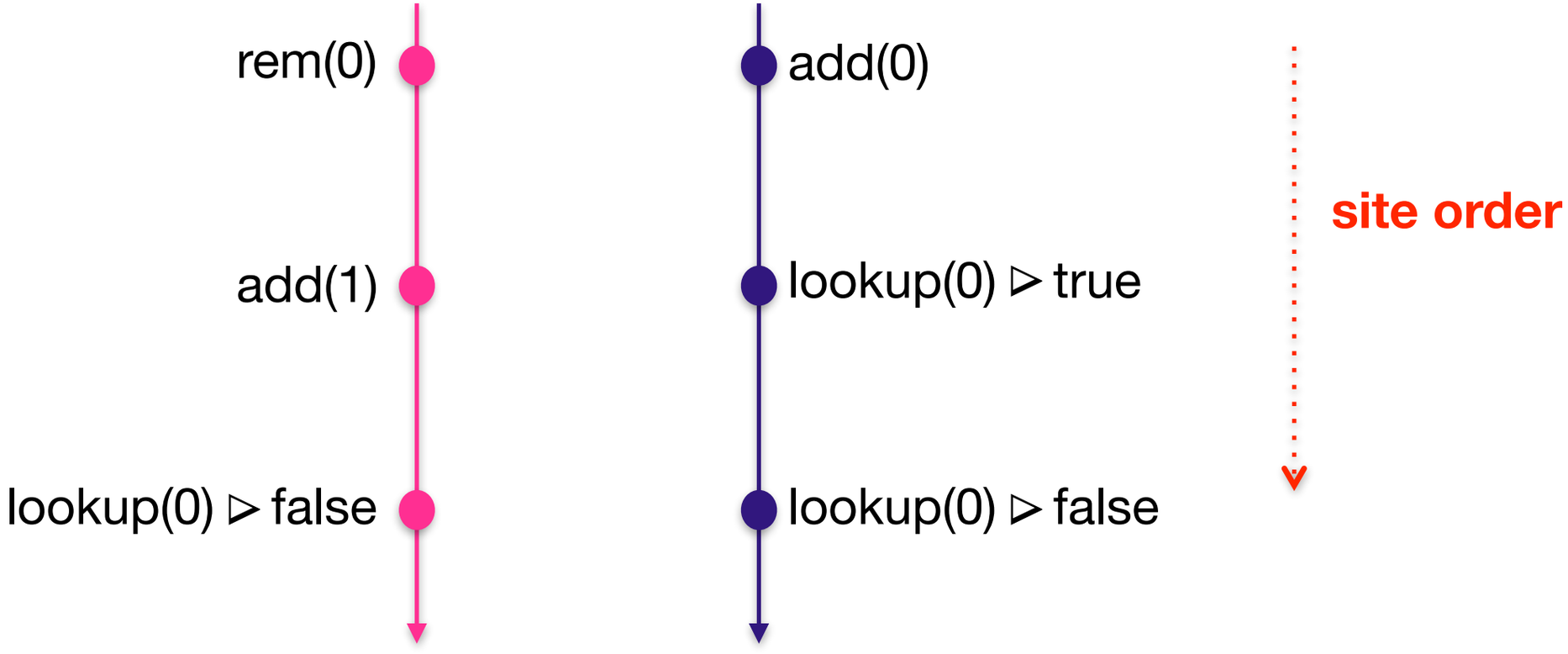
Plan

- Formal definitions of consistency criteria
  - safety: each operation is executed in the context of a local view, which must satisfy some specification
    - sites can have different local views
  - liveness: the local views converge toward a global view

- Automatic verification of consistency criteria
  - decidability/complexity results
  - general reductions to classical verification problems (reachability, model checking)

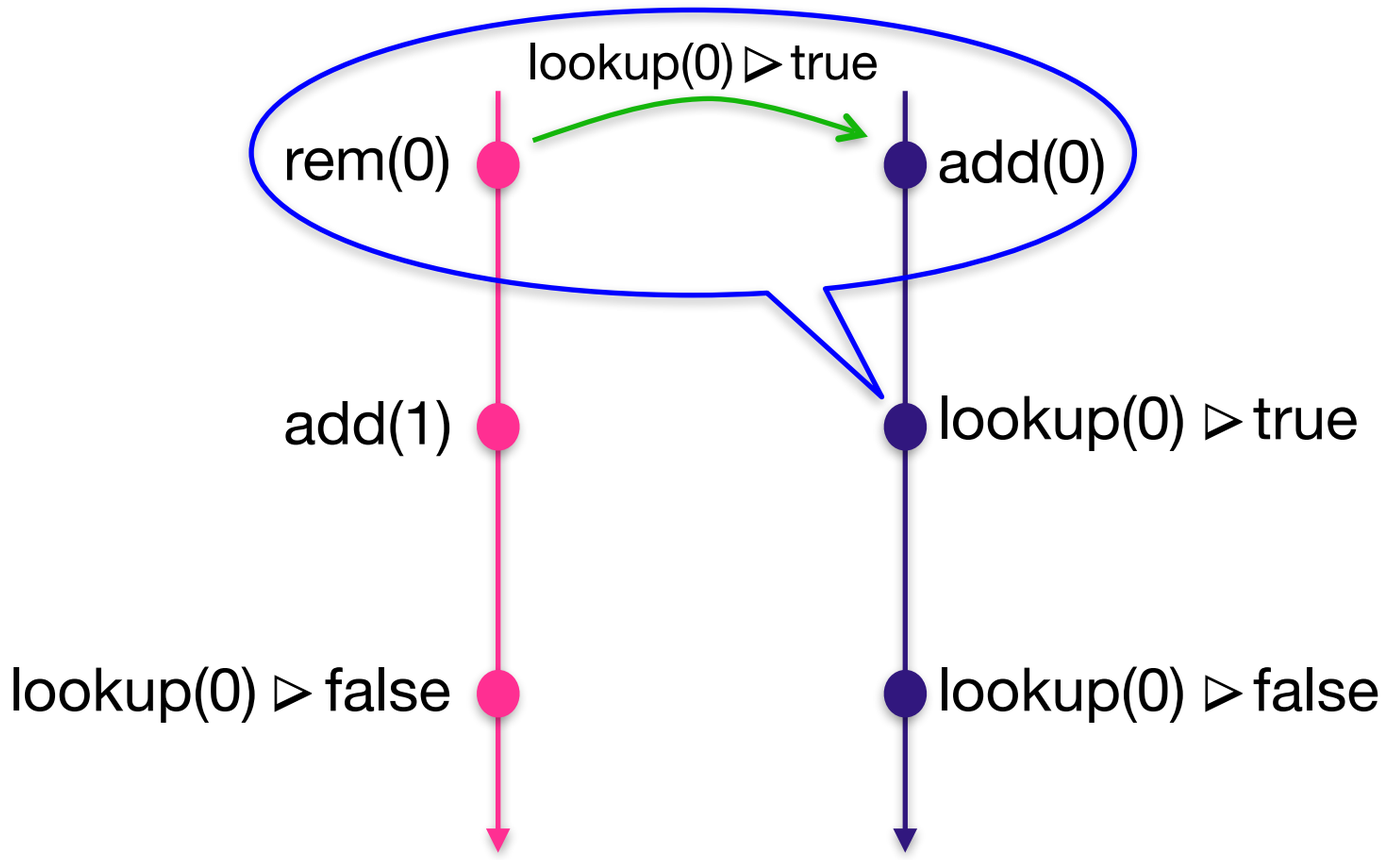
Modeling DDS behaviors with traces

- **operations** are instances of a set of methods (add, rem, lookup)
- **traces** record the submitted operations and their return values
  - trace = a partially-ordered set of operations
  - operations submitted to the same site are ordered



Specifying Safety

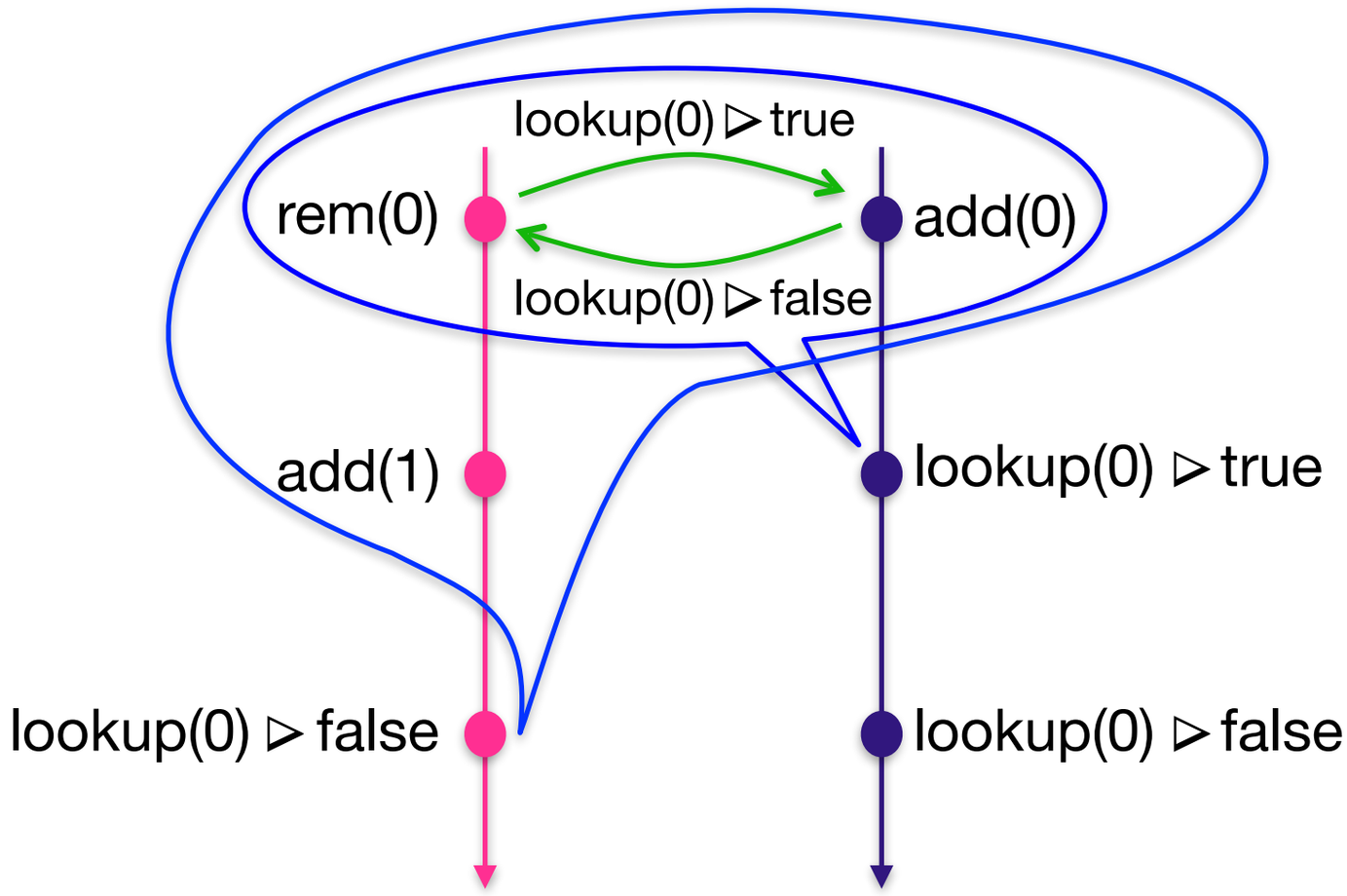
- Specifying local views using relations between operations



- **Vis(o',o)**: o' is visible when o is executed
- **Order$_o$(o_1,o_2)**: when o is executed, o1 has been executed before o2

Specifying Safety

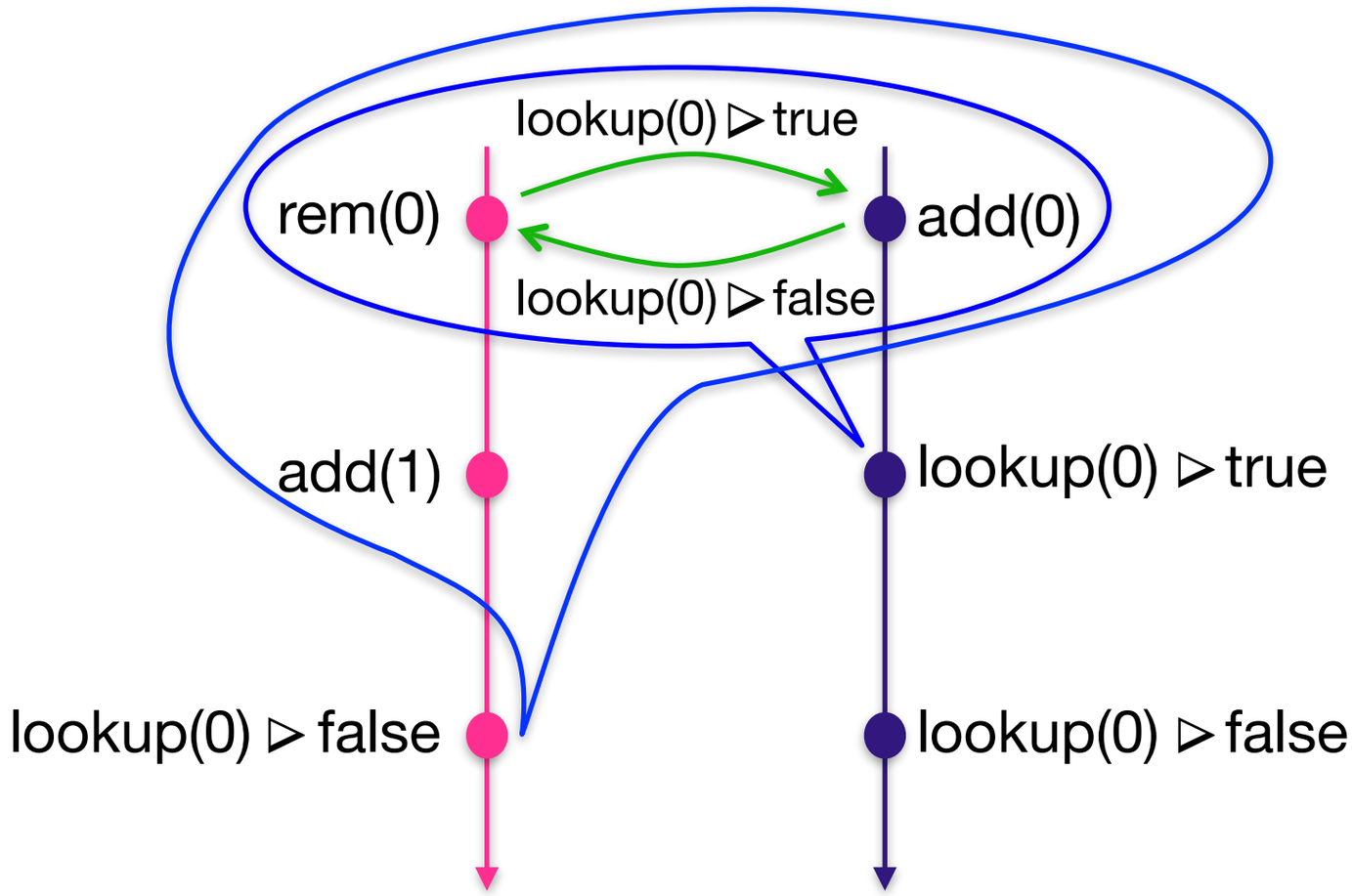
- Specifying local views using relations between operations



- **Vis(o',o)**: o' is visible when o is executed
- **Order$_o$(o_1,o_2)**: when o is executed, o1 has been executed before o2

Specifying Safety

- Specifying local views using relations between operations



- **Vis(o',o)**: o' is visible when o is executed
- **Order$_o$(o$_1$,o$_2$)**: when o is executed, o1 has been executed before o2
- **Safety:** ∃ **Vis** ∀o ∃ **Order$_o$ such that …**

Specifying Safety

- **Vis(o',o)**: o' is visible when o is executed
- **$Order_o(o_1,o_2)$**: when o is executed, o1 has been executed before o2
- **Safety:** ∃ **Vis** ∀o ∃ **$Order_o$ such that**
  - eventual consistency: **Vis** ∪ **Site_order** is acyclic

$Order_o$ satisfies the specification

Specifying Safety

- **Vis(o',o)**: o' is visible when o is executed
- **$Order_o(o_1,o_2)$**: when o is executed, o1 has been executed before o2
- **Safety:** ∃ **Vis** ∀o ∃ **$Order_o$ such that**
  - eventual consistency: **Vis** ∪ **Site_order** is acyclic

    $Order_o$ satisfies the specification
  - read-your-own-writes: **+ Site_order** ⊆ **Vis**

    if **$Vis(o_1,o)$, $Vis(o_2,o)$, and $Site\_order(o_1,o_2)$ then $Order_o(o_1,o_2)$**

Specifying Safety

- **Vis(o',o)**: o' is visible when o is executed
- **$\text{Order}_o(o_1,o_2)$**: when o is executed, o1 has been executed before o2
- **Safety:** ∃ **Vis** ∀o ∃ **Order_o such that**
  - eventual consistency: **Vis** ∪ **Site_order** is acyclic

    Order_o satisfies the specification
  - read-your-own-writes: **+ Site_order** ⊆ **Vis**

    if **$\text{Vis}(o_1,o)$, $\text{Vis}(o_2,o)$, and $\text{Site\_order}(o_1,o_2)$ then $\text{Order}_o(o_1,o_2)$**
  - causal consistency: **+ Site_order** ⊆ **Vis** and **Vis is transitive**

    if **$\text{Vis}(o_1,o)$, $\text{Vis}(o_2,o)$, and $\text{Vis}(o_1,o_2)$ then $\text{Order}_o(o_1,o_2)$**

Specifications

- **Sequential** world: an **operation** is considered **correct** depending on the sequence of previously executed operations
  - lookup(0) ▷ true is correct iff it executes after a sequence of operations where the projection on operations with input 0 ends in add(0)
    - e.g., add(2), rem(0), … , add(0), rem(2)

- **Distributed** world: sequences are replaced by partial orders
  - lookup(0) ▷ true is correct iff it executes after a poset of operations where the projection on operations with input 0 contains a maximal add(0)
    - e.g., add(2), rem(0), … , add(0), rem(2)

Specifying liveness (convergence)

- **Arb(o_1,o_2)**: eventually, all sites agree that o_1 should be executed before o_2

- Safety + Liveness: ∃ **Vis** ∃ **Arb** ∀o ∃ **Order$_o$** **such that … and**

∀ o_1,o_2: #o s.t. **Order$_o$(o_1,o_2)** ∧ **Arb(o_2,o_1)** is finite

Verification problems

- Given a **finite-state** implementation **Impl** and a **"regular"** specification **Spec**, is there a **"simple" centralized monitor** that

  outputs error iff **Impl** is not X-consistent w.r.t. **Spec**

  (X = eventually, causal, read-your-own-writes,…)

- Eventual consistency: Yes (counting + Presburger assertions)
- Read-your-own-writes, Causal consistency: No
- Linearizability: Yes (finite-state)

- Restricting specifications: for causal consistency, there exists such a monitor if the **Spec** is key-value store

Conclusions

- **Distributed data structures**, an alternative to classical databases

- Several **consistency criteria** which are poorly understood
  - **guarantees** for the application layer ?

- **Hard (undecidable)** verification problems
  - finding reasonable restrictions or approximations