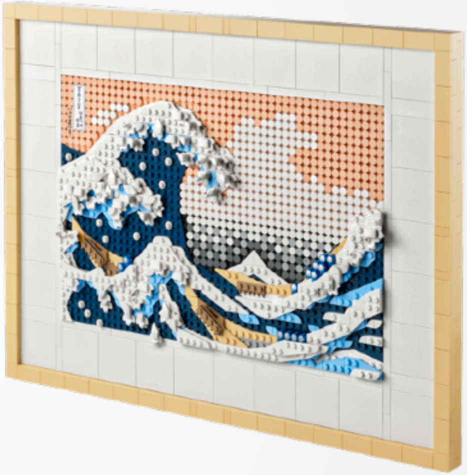
ADVANCED COURSE

Distributed Systems

Weaker Consistency Models & CRDTs



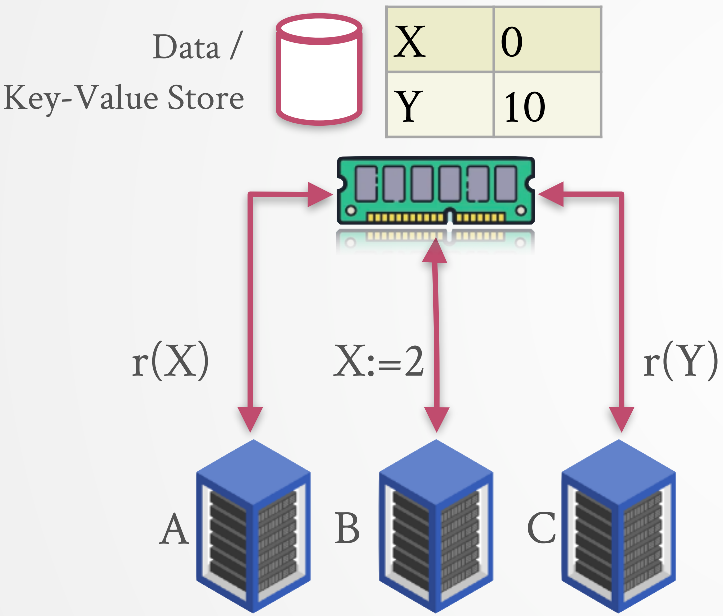
Paris Carbone

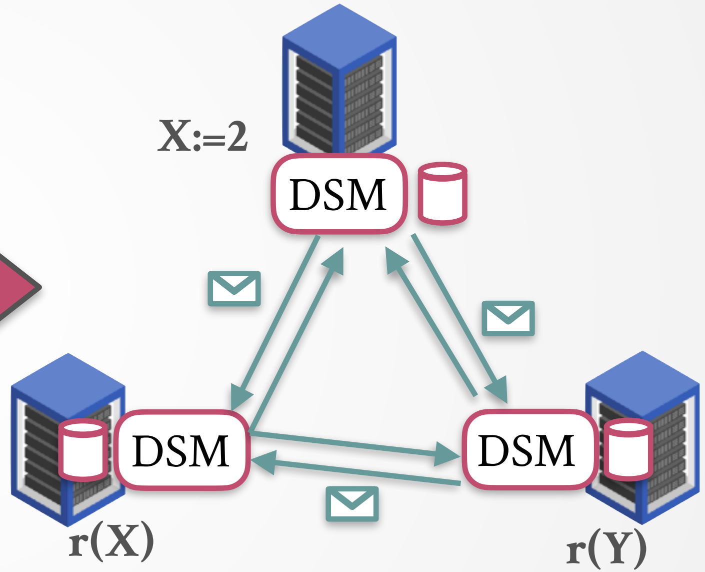
COURSE TOPICS

- Intro to Distributed Systems
- Fundamental Abstractions and Failure Detectors
- Reliable and Causal Order Broadcast
- Distributed Shared Memory-CRDTs
- Consensus (Paxos)
- Replicated State Machines (OmniPaxos, Raft, Zab etc.)
- Time Abstractions and Interval Clocks (Spanner etc.)
- Consistent Snapshotting (Stream Data Management)
- Distributed ACID Transactions (Cloud DBs)

Have we Achieved the Goal?

Registers / Key-Values

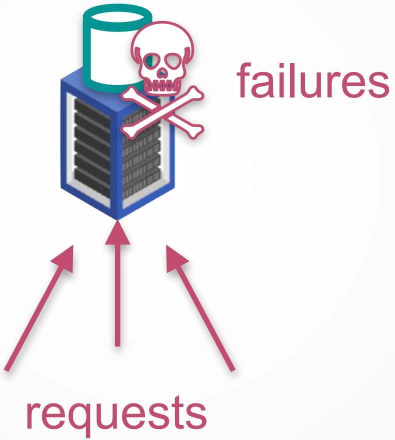




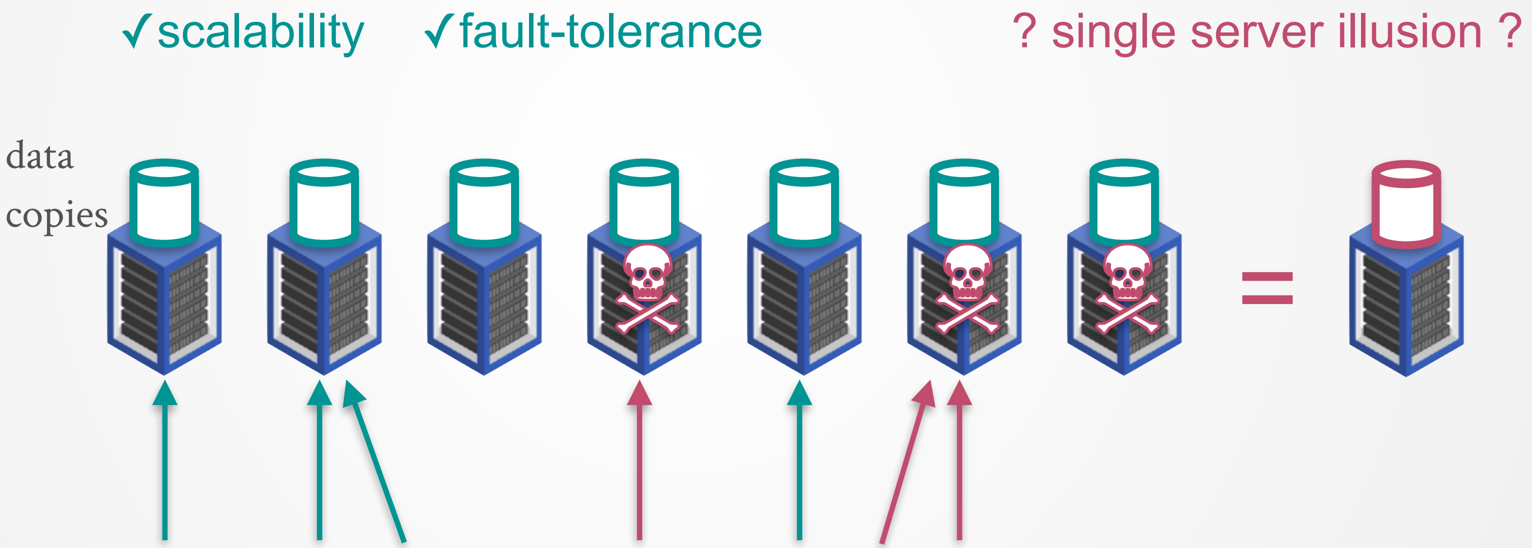
Shared Mem: Processes/Servers have direct memory access (no messages)

Distributed Shared Mem (DSM): Processes/Servers have indirect memory access (using messages)

REPLICATED DATA SERVICES



Replicated Data Services



PROPERTIES OF REPLICATED DATA SERVICES

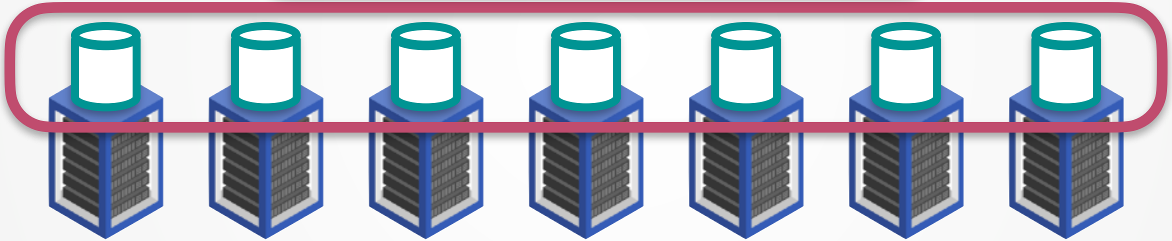
no coordination ++ #allowed states - - high coordination

(relaxed ordering of ops) (strict ordering of ops)

No Consistency = x n copies

Consistency

Strong Consistency = Linearizable,Atomic etc



"The degree at which data has a "Single-Copy View"

Properties of Replicated Data Services

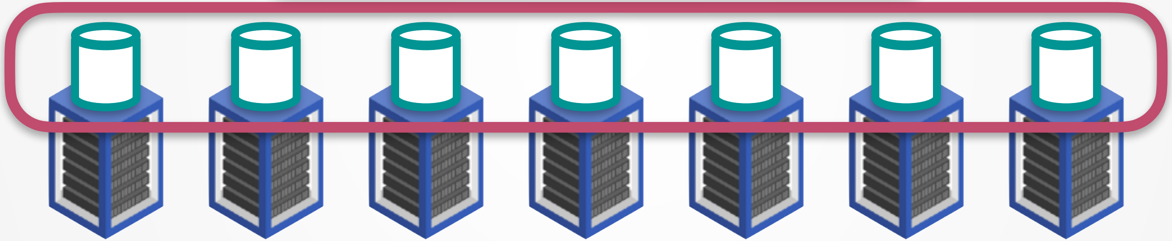
no coordination ++ #allowed states - - high coordination

(relaxed ordering of ops) (strict ordering of ops)

No Consistency = x n copies

Consistency

Strong Consistency = Linearizable,Atomic etc

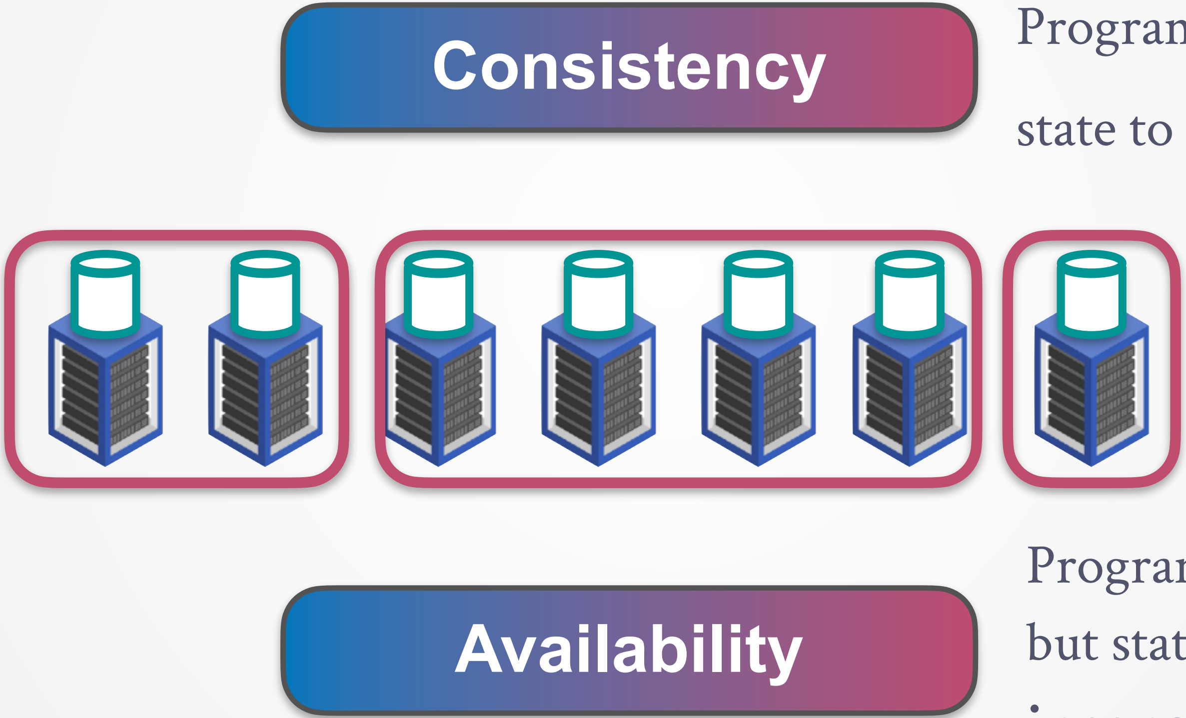


No Availability = System is unresponsive

Availability

Availability = Every correct node responds to requests

Network Partition = Sacrifice



Program **waits** for state to synchronize

Choose **one**

Program responds but state is **inconsistent**

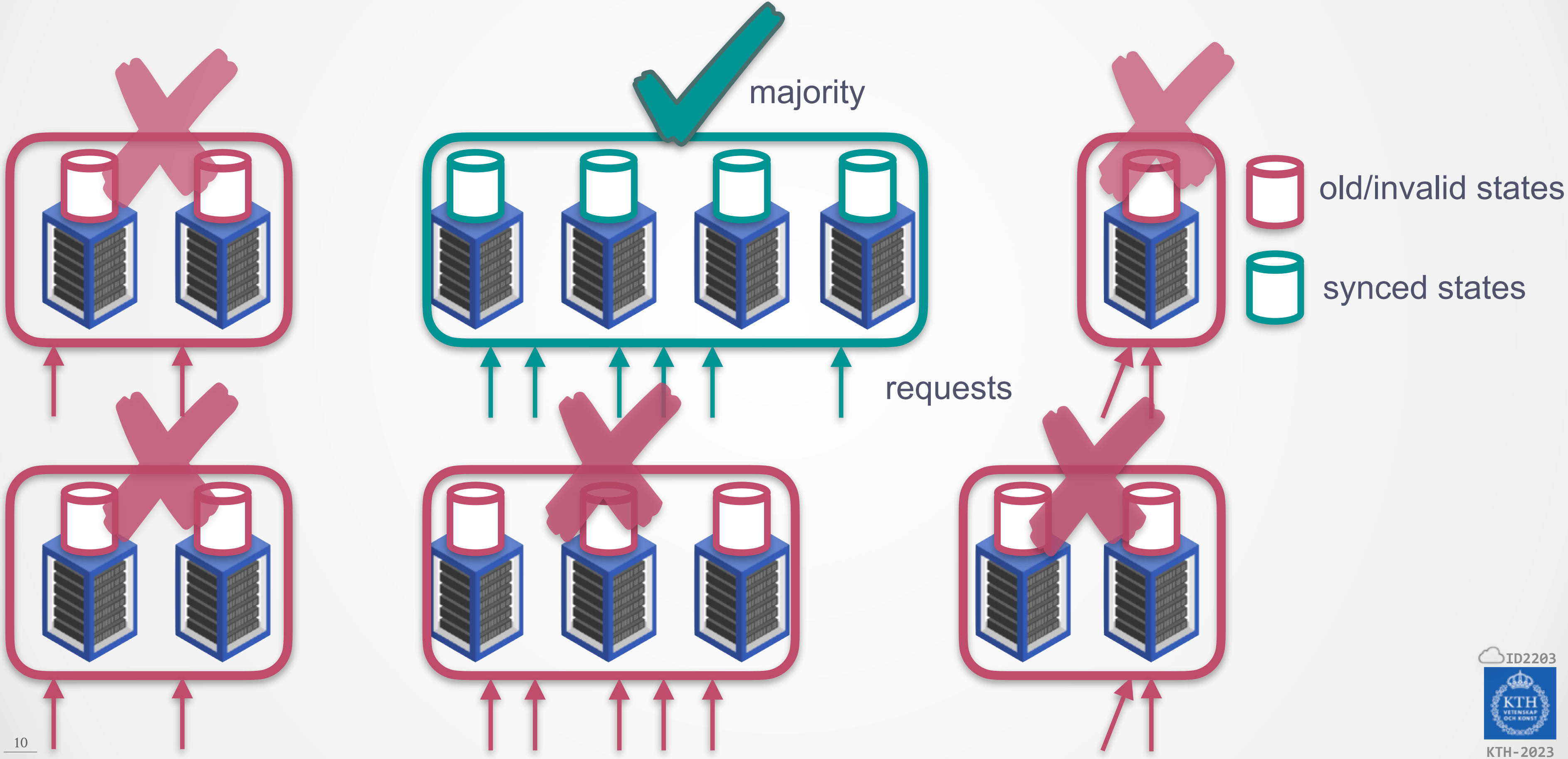
BREWER'S THEOREM (CAP)

- **Network partitions** are often unavoidable! (e.g., mobile computing).

  "Choose either **C**onsistency or **A**vailability to tolerate **P**artitions"

- **Problem**: Linearizability requires quorum-based communication. If quorum not reachable during partitioning system gets stuck.

AVAILABILITY DURING PARTITIONING

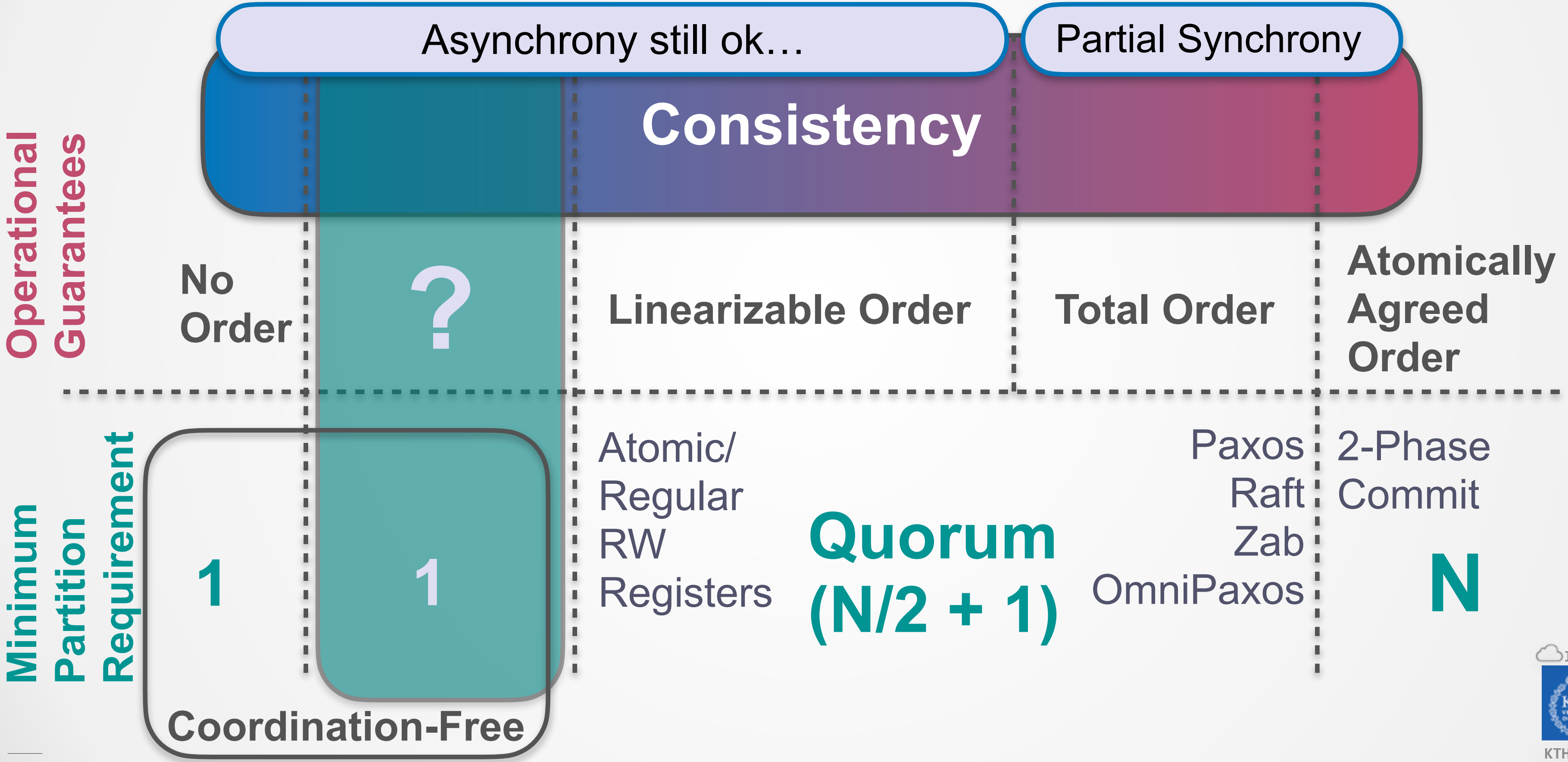


Is it Really a Binary Option?

Consistency		**Consistency**	
	VS		
Availability		**Availability**	

This sounds like a really bad deal…

Drilling Down Consistency



Weaker Consistency Models

- Certain consistency conditions do not require coordination.
- Note: Coordination-free does not imply Synchronization-free.
  - We have already seen a few examples:
    - Causal/FIFO Reliable Broadcast
    - Eventual Consistency

EVENTUAL CONSISTENCY

- State updates can be issued at any replica/correct process.
- All updates are disseminated via BEB, RB,…

- Each correct process that receives all updates should deterministically converge to the same state.
- Eventually every correct process should receive **all** updates…

- **Problem**: When can a process know it has received all updates??

Strong eventual Consistency

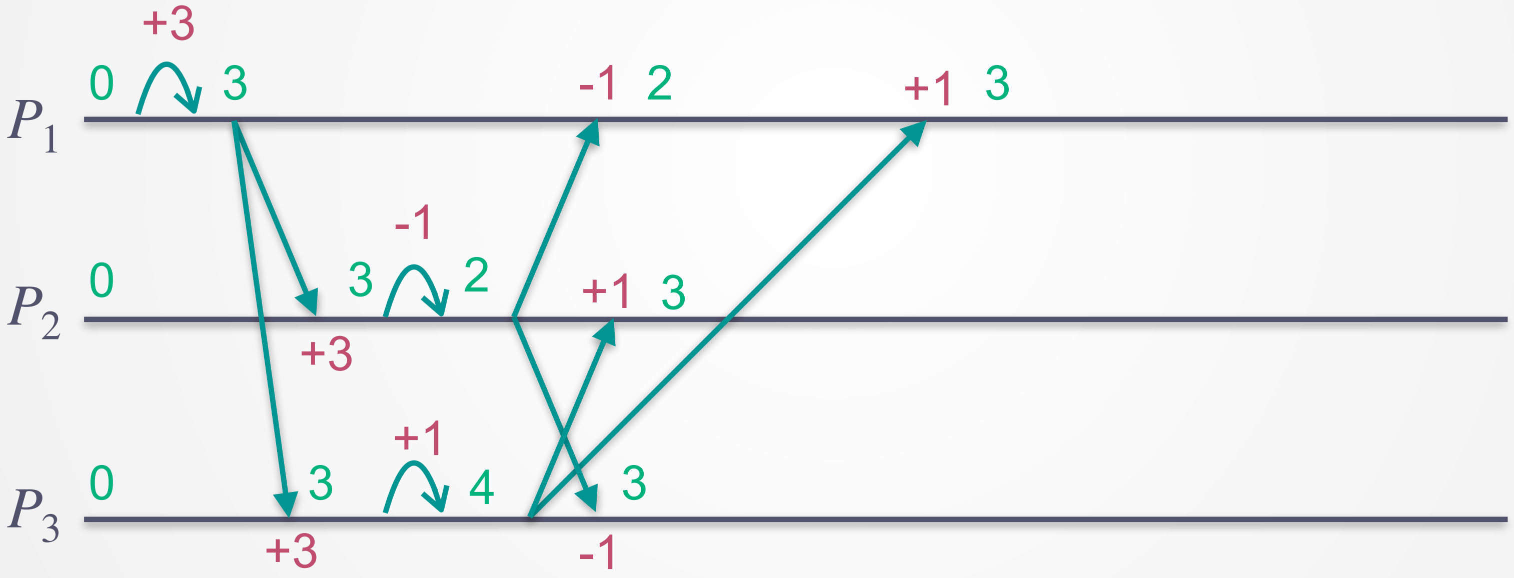
- Same as before, updates can be issued at any process/replica.

- **SEC Property:** If two correct processes p_1, p_2 receive the exact same set of updates, then $p_1 . state = p_2 . state$.

- **Main Idea**: If state operations are **commutative** and processes exchange information, eventually they converge to an identical view.

EXAMPLE

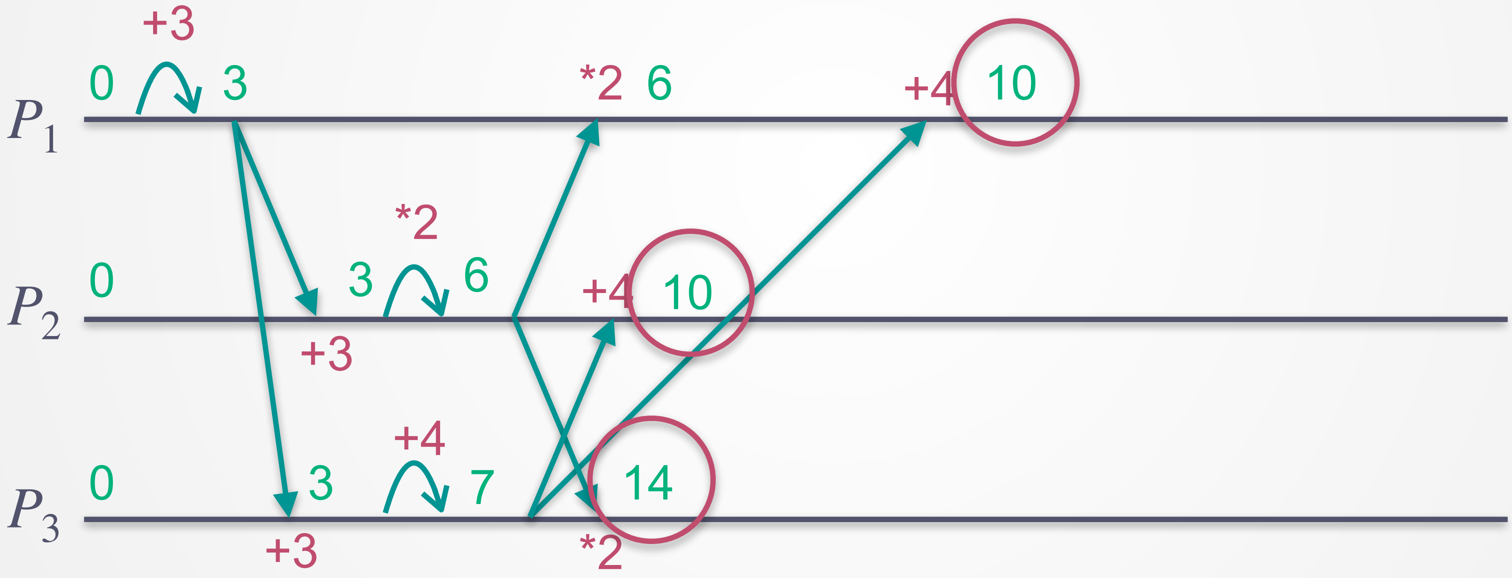
- Processes can either add or subtract (+, - are commutative) to a shared register.
- Assume reliable broadcast. Each process updates + broadcasts each operation



ID2203
KTH-2023

EXAMPLE #2

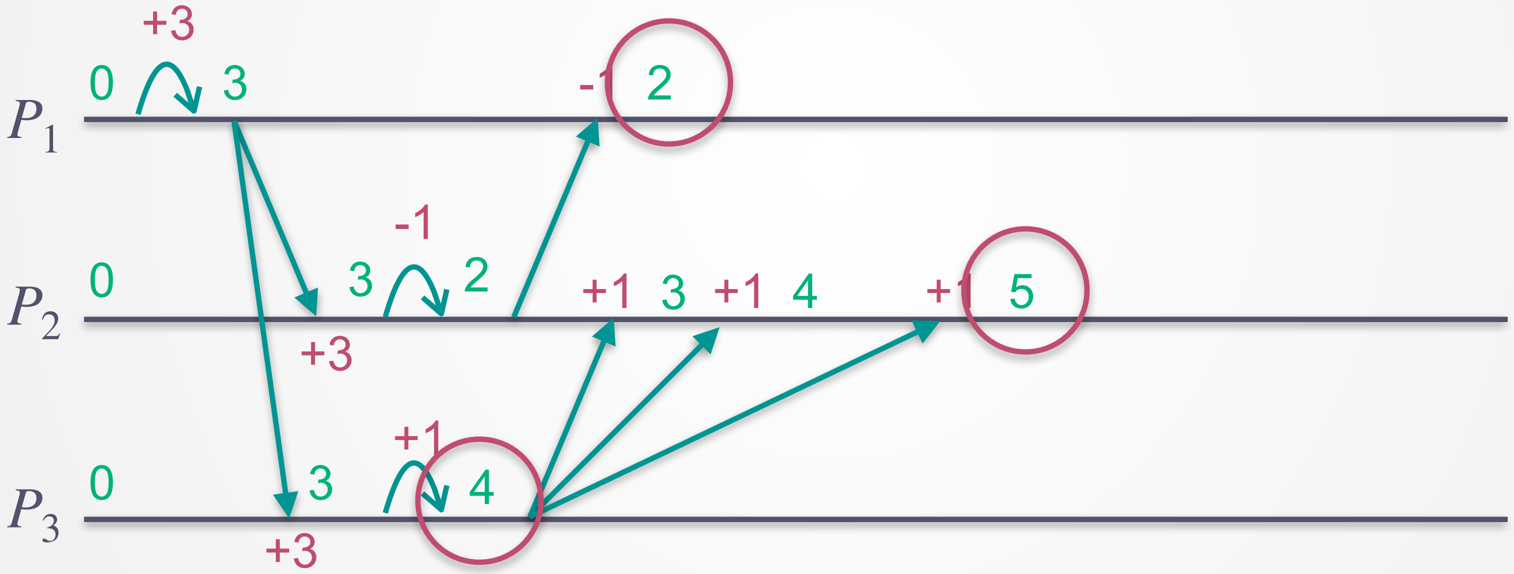
- Processes can either **multiply** or **add** to a shared register.
- Assume reliable broadcast. Each process updates + broadcasts each operation



non-commutative operations do not converge!

EXAMPLE #3

- Processes can either add or subtract (+, - are commutative) to a shared register.
- Each process updates + broadcasts each operation. Assume **unreliable communication.**



if unreliable communication, operations need to be **idempotent**!

CONVERGENT DATA TYPES

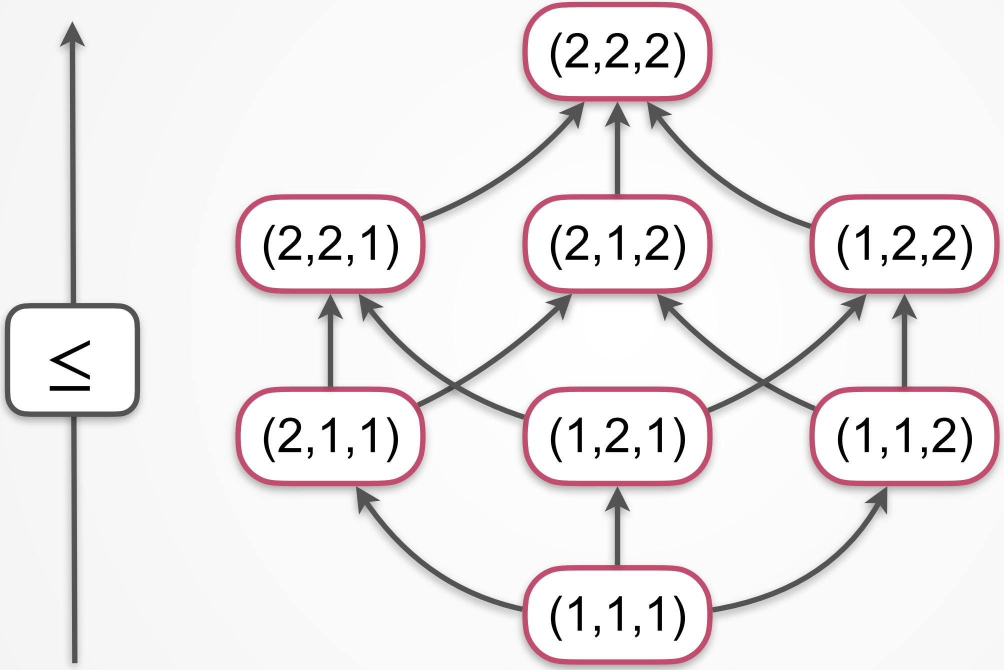
- Data structures that implement strong eventual consistency.
- CRDTs : Conflict-Free Replicated Data Types.
- Two Equivalent Types: Operation-Based and State-Based
- Assumptions:
  - **Arbitrary Network Partitions**
  - **Fail-Recovery:** Process Memory Survives Crashes
  - Asynchronous Process Model
  - Required type of broadcast differs across CRDT types.

RECAP: POSET

- **Partial Order**: binary relation $\leq$ on a set T, written $< T, \leq >$
  - Reflexive: $a \leq a$ for $a \in T$
  - Antisymmetric: $(a \leq b \wedge b \leq a) \Rightarrow (a = b)$ for $a, b \in T$
  - Transitive: $(a \leq b \wedge b \leq c) \Rightarrow (a \leq c)$, for $a, b, c \in T$

- Example:
  - Vector Clocks $< \langle \mathbb{Z}^+, \ldots, \mathbb{Z}^+ \rangle, \leq >$

RECAP: POSET

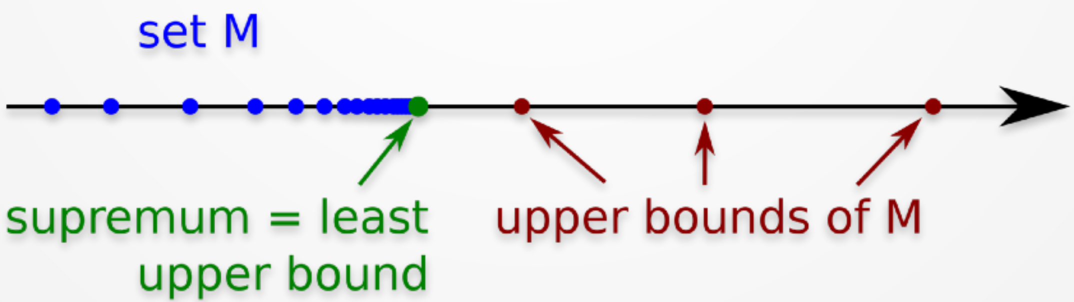


JOIN SEMILATTICE

1)A partially order set T .

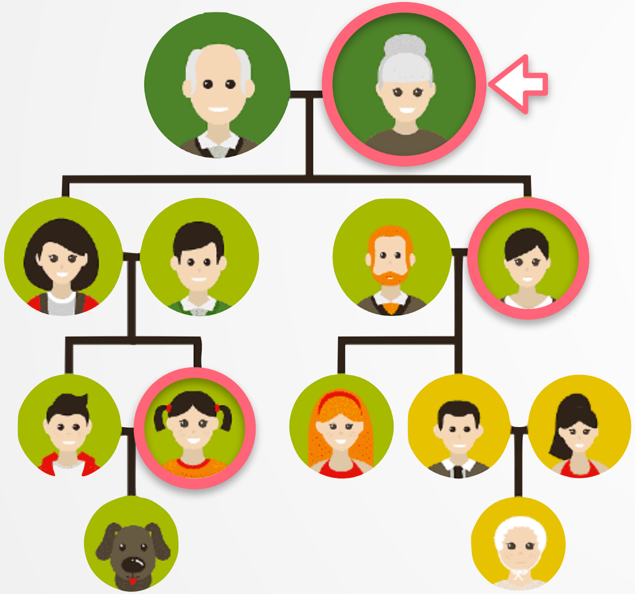
2) A Join is a Least Upper Bound (infimum) $\sqcup$ of any subset $M \subseteq T$

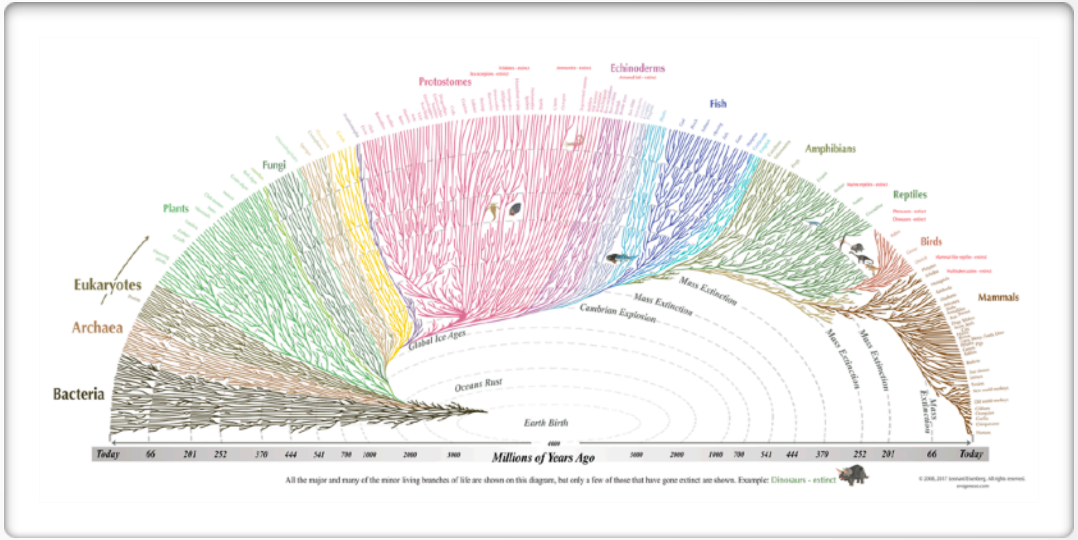
- (1)+(2) yield a join-semilattice with the following properties
- Commutativity: $t \sqcup t' = t' \sqcup t$
- Idempotency: $t \sqcup t = t$
- Associativity: $(t_1 \sqcup t_2) \sqcup t_3 = t_1 \sqcup (t_2 \sqcup t_3)$



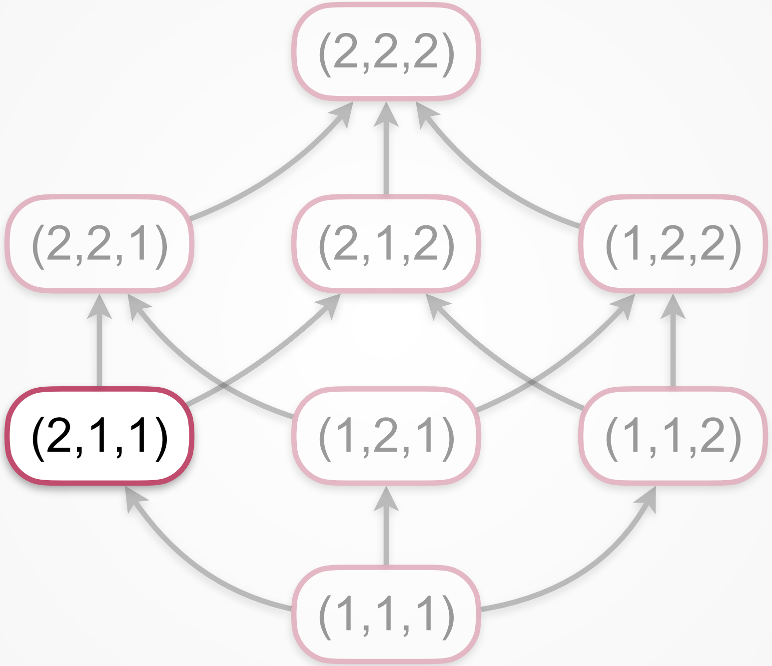
EXAMPLES

Least Upper Bound: First common ancestor in a family/biological tree



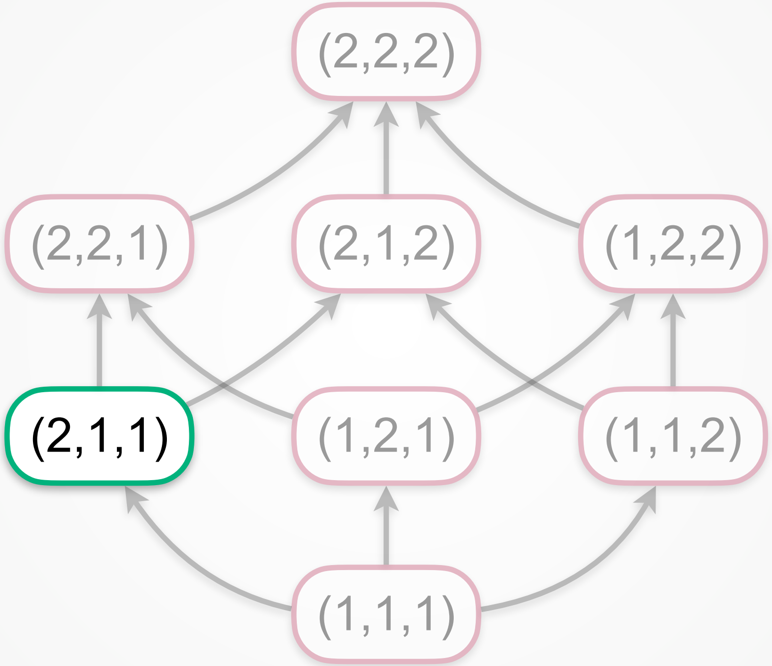


EXAMPLES



$(2,1,1) \sqcup (2,1,1) =$

EXAMPLES

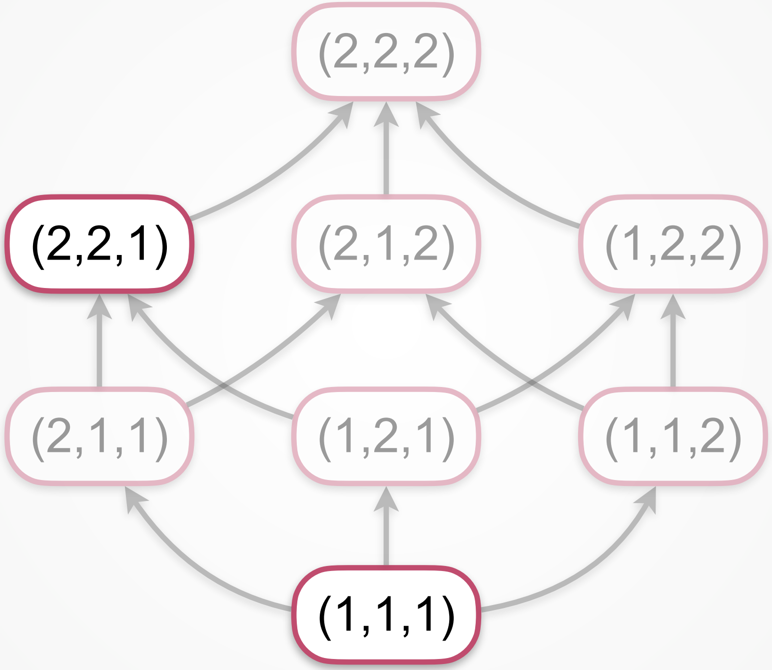


$$(2,1,1) \sqcup (2,1,1) = (2,1,1)$$

ID2203

KTH-2023

EXAMPLES

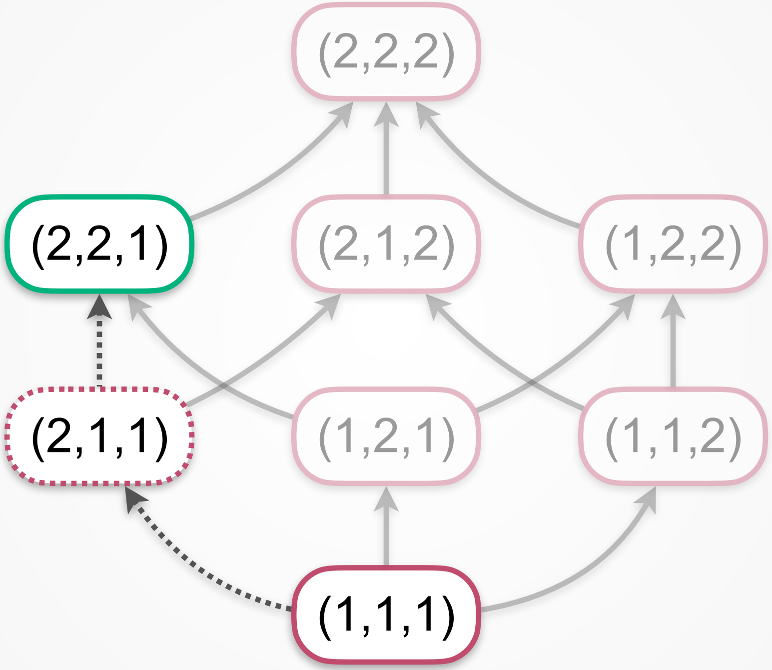


$(1,1,1) \sqcup (2,2,1) =$

ID2203

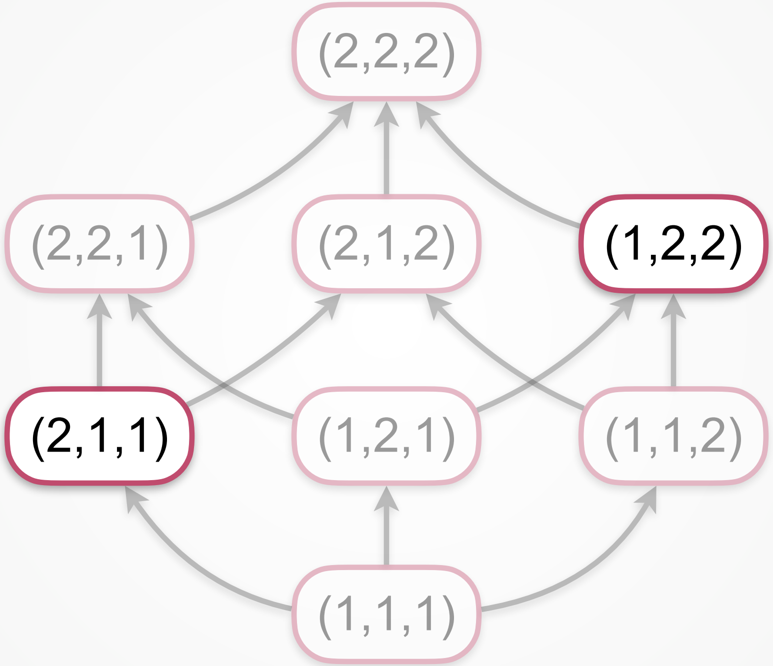
KTH-2023

EXAMPLES



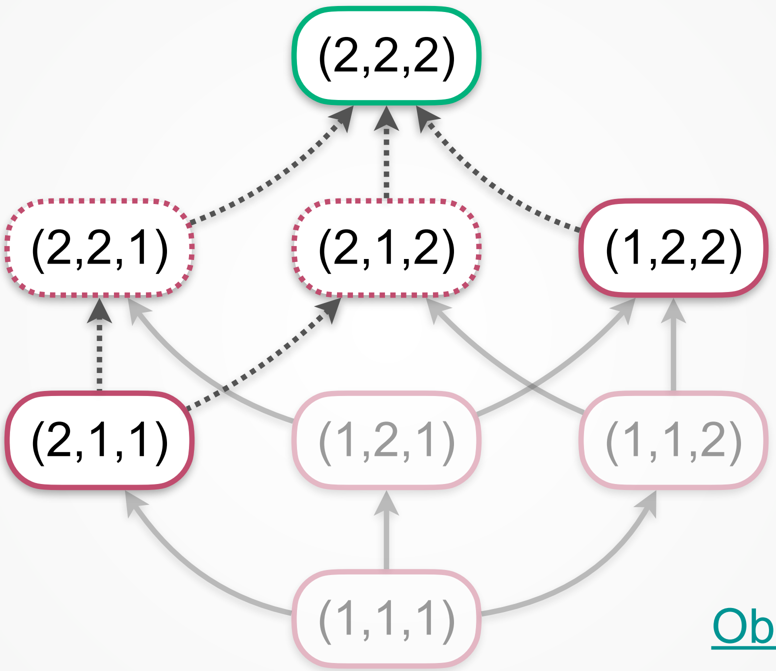
$(1,1,1) \sqcup (2,2,1) = (2,2,1)$

Examples



$(2,1,1) \sqcup (1,2,2) =$

Examples



$(2,1,1) \sqcup (1,2,2) = (2,2,2)$

Observations

$\sqcup$ always moves up the lattice

$\sqcup$ can join concurrent values

More Example

Given poset $(\mathbb{Z}^+, \leq)$ and $\sqcup = \max$

- Commutativity: $10 \sqcup 1000 = 1000 \sqcup 10 = 1000$
- Idempotency: $9000 \sqcup 9000 = 9000$
- Associativity: $(1 \sqcup 120) \sqcup 40 = 1 \sqcup (120 \sqcup 40) = 120$

More Examples

Given set of greek letter combinations and $\sqcup = \cup$

- Commutativity: $\{\lambda\} \sqcup \{\kappa, \omega\} = \{\kappa, \omega\} \sqcup \{\lambda\} = \{\kappa, \lambda, \omega\}$
- Idempotency: $\{\omega\} \sqcup \{\omega\} = \{\omega\}$
- Associativity: $(\{\kappa\} \sqcup \{\lambda\}) \sqcup \{\pi\} = \{\kappa\} \sqcup (\{\lambda\} \sqcup \{\pi\}) = \{\kappa, \lambda, \pi\}$

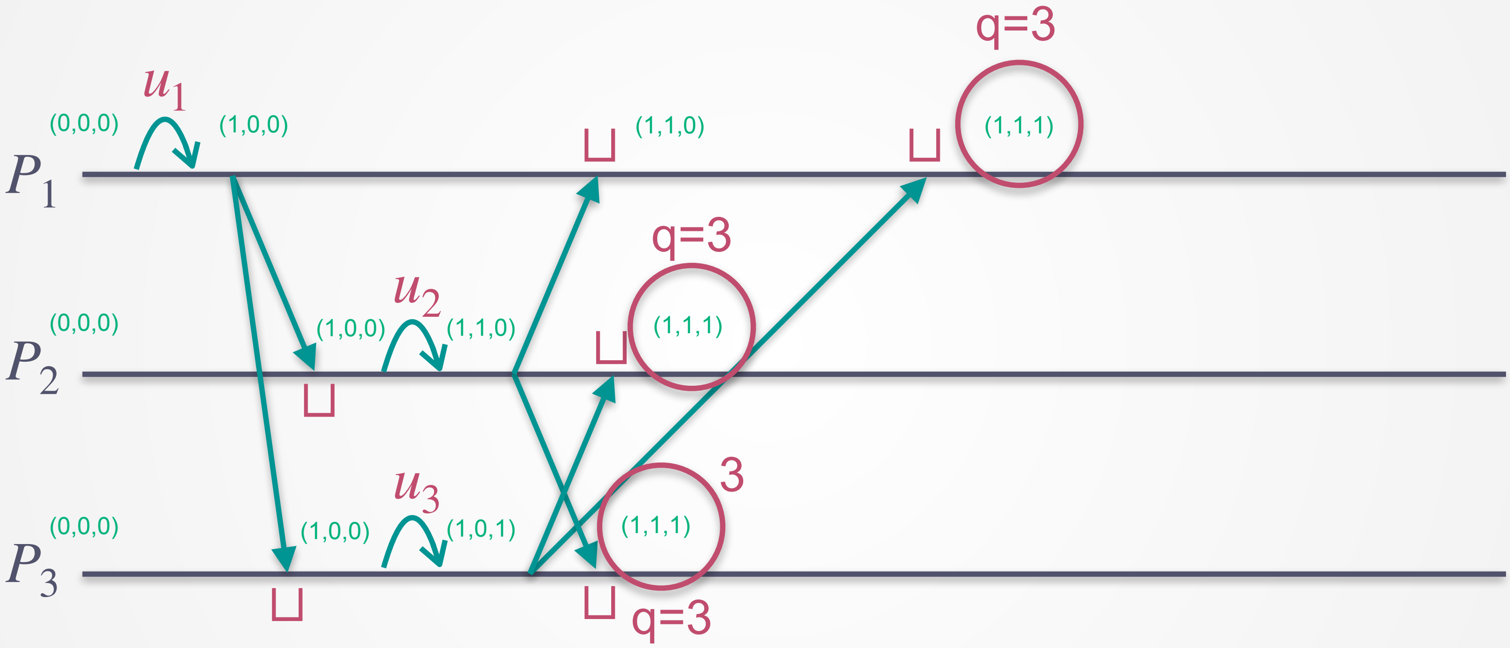
State-Based (CvRDTs)

- Each process maintains a triple $((s_1, \ldots, s_n), u, q)$:
  - $(s_1, \ldots, s_n)$ is the *configuration* on n replicas, $s_i \in S$ **(semilattice)**
- **Operations**
  - **Read q:** S⟶V is a query function
  - **Update** u_i **:** S⟶S is a mutator such that s ⊑ u_i(s) (monotonic)
  - **Merge (⊔)** : $S \times S \rightarrow S$, where ⊔ is a **least upper bound** for S
- **Usage**: Processes exchange (beb broadcast) configurations and merge them

Grow-Only Counter

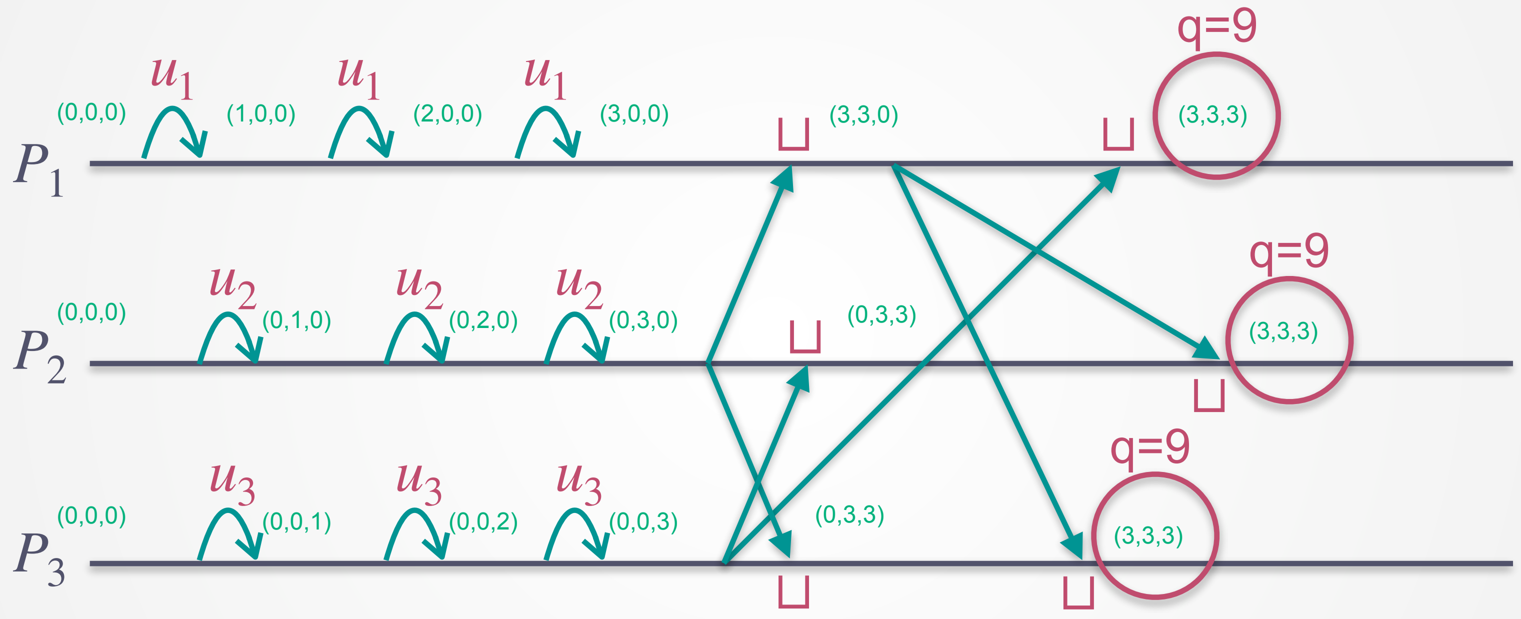
- Configuration
  - $(s_1, \ldots, s_n)$: increments by each process, initially (0,…,0)
- **Operations**
  - **Read: q** = Sum of all elements, e.g., $q((1,2,1)) = 4$
  - **Update:** u_i : Increments i^{th} counter, e.g., $inc_1((1,2,1)) = (1,3,1)$
  - **Merge ($\sqcup$)** : Max of each element, e.g., $(1,2)\sqcup(5,1) = (5,2)$

Grow-Only Counter Example



do we need to disseminate configuration on each inc?

Grow-Only Counter Example



Periodic broadcasts by each process still converge to same state…

CvCRDTs - Observations

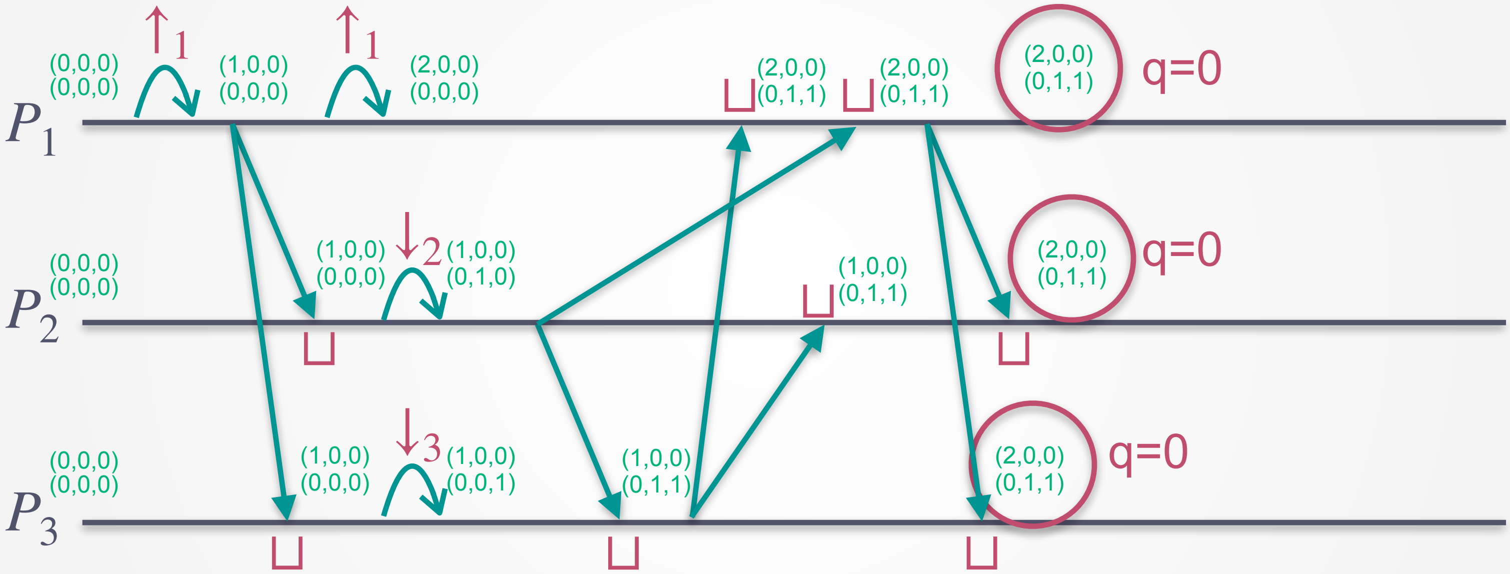
- From the example we can derive that
  - Synchronization can be tuned without violating correctness for State-Based CRDTs (eventual convergence is guaranteed).
  - Any form of reliable broadcast suffices (Order is not important)
  - Causal Order is derived in configurations (through merge)

  - What if we want to support more state operations?
  - e.g., counter that supports decrements? ($\sqcup$ only goes ↑, not ↓)

UP-DOWN COUNTER

- Configuration
  - $((\uparrow_1, \ldots, \uparrow_n), (\downarrow_1, \ldots, \downarrow_n))$ **num of increments and decrements / process**
- **Operations**
  - **Read: q** = $\sum_{i=0}^{n} \uparrow_i - \downarrow_i$, e.g., q((1,2,1),(1,0,1)) = 2
  - **Update** u_i **:**
    - u^{inc}: **increments** $\uparrow_i$, e.g., $u_1^{inc}((1,2,1), (0,0,0)) = ((1,3,1), (0,0,0))$
    - u^{dec}: **increments** $\downarrow_i$, e.g., $u_1^{dec}((1,3,1), (0,0,0)) = ((1,3,1), (0,1,0))$
  - **Merge (⊔)** : Max for both vectors, e.g., ((1,2),(1,1))⊔((5,0),(2,1)) = ((5,2),(2,1))

Up-Down Counter Example



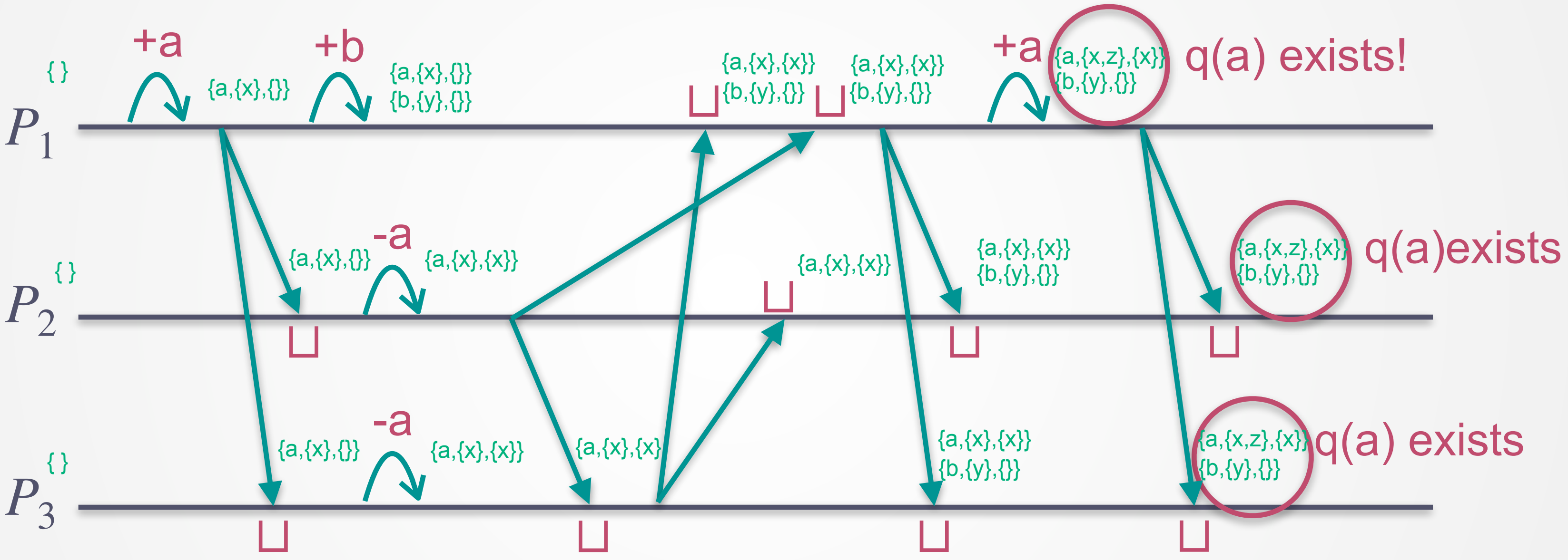
OR-SET

- Assume we want to support the set "add" and "remove" ops on a CvRDT (e.g., shopping cart)
- Both add and remove ops should cause monotonic updates. They are not commutative.
- Configuration : $((o_i, \{add\}, \{rem\}) \in O)$ - **o:** object, **add:** addition tags, **rem:** removal tags
- <u>Operations</u>
  - **Read(e) : exists(e) :** if ∃ (e,{add},{remove}) then return add-remove ≠ { }
  - **Update** u_i **:**
    - u^{add} (e): (e,{add} ∪ x, {remove}) , x : unique identifier
    - u^{rem} (e): (e,{add}, {remove}∪ {add})
  - **Merge (⊔)** : Union of each triplets, e.g., (apple,{a,b},{a})⊔(apple,{c},{}) = (apple,{a,b,c},{a})

ID2203

KTH-2023

OR-Set Example



ID2203

KTH-2023

CvCRDTs - Observations

- From the examples we can derive that
  - Synchronization can be tuned without violating correctness for State-Based CRDTs (eventual convergence is guaranteed).
  - Any form of reliable broadcast suffices (Order is not important)
  - Causality is is preserved in configurations

  - Configuration space can get large: e.g., O(|operations| |P|)
  - CvRDTs send a lot of redundant state. Cant we send just operations?

A DEEPER LOOK

- Why do CvRDTs work again? They always converge to the same state despite arbitrary broadcast delivery order.
- Remember any two updates u_1, u_2 are distributed events. They can be either:
  - 1. Causally Dependent Updates: Encapsulated in the S (semilattice)
    - if $u_1 \rightarrow u_2$ then $u_1(s) \leq u_2(s)$ - since **u is monotonic**
  - 2. Concurrent Updates:
    - $\sqcup$ of S (Join-Semilattice) is **commutative**
- Can we provide the same properties without the overly inflated states?

A Deeper Look

- Why do CvRDTs work again? They always converge to the same state despite arbitrary broadcast delivery order.
- Remember any two updates u_1, u_2 are distributed events. They can be either:
  - 1. Causally Dependent Updates: Encapsulated in the S (semilattice)
    - if $u_1 \rightarrow u_2$ th $u_1(s) \leq u_2(s)$

use causal-order broadcast

  - 2. Concurrent Updates:
    - $\sqcup$ of S (Join-Semilatt) is **co**

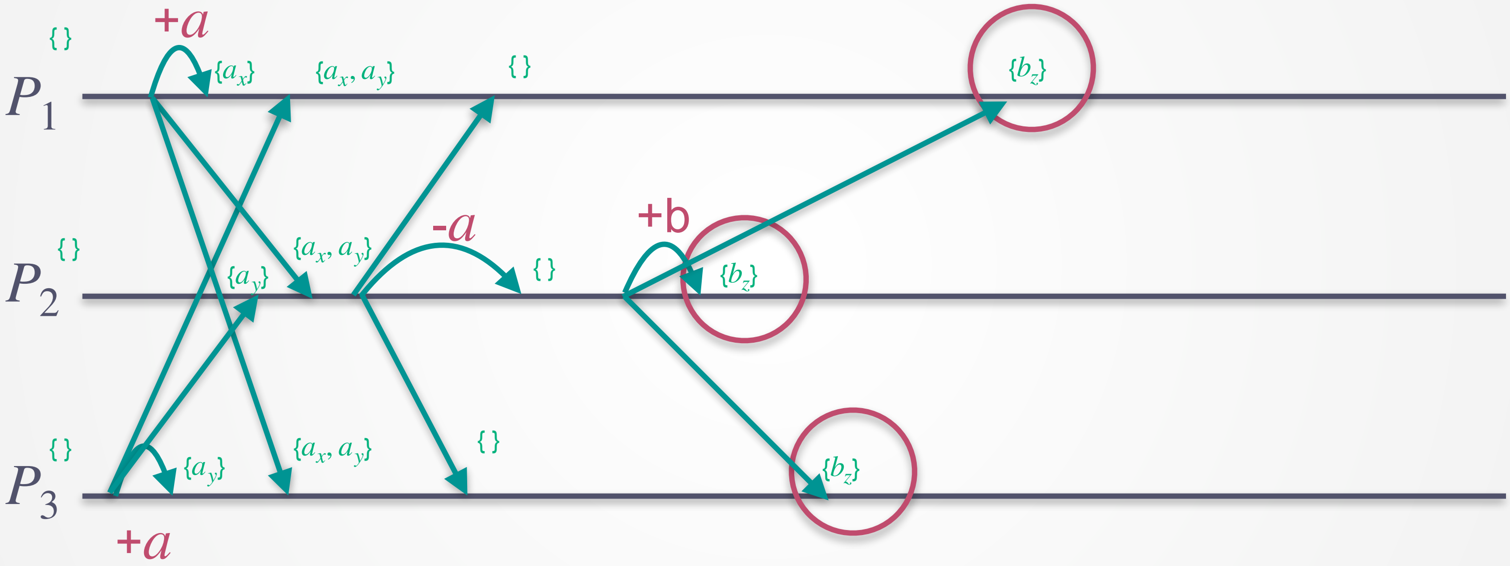
use commutative update function

- Can we provide the same properties without the overly inflated states?

Operation-Based CmRDTs

- Each process maintains a triple (S, u, q): (simplified version)
- **Operations**
  - **Read q:** $S \longrightarrow V$ is a query function
  - **Update** $u_i : S \longrightarrow S$ is a mutator. u is **commutative**
- **Usage**:
  - on update request u, generate u' : crb_broadcast u'.
  - upon receiving u', apply u'.

OR-SET EXAMPLE (CMRDT)



CmRDTs

- For trivially commutative problems (e.g., +, - operators) then we might not necessarily need causal order broadcast.

- Less States and IO
- More restrictions in programming model (commutativity)
- Less Flexible to work with

Other Approaches

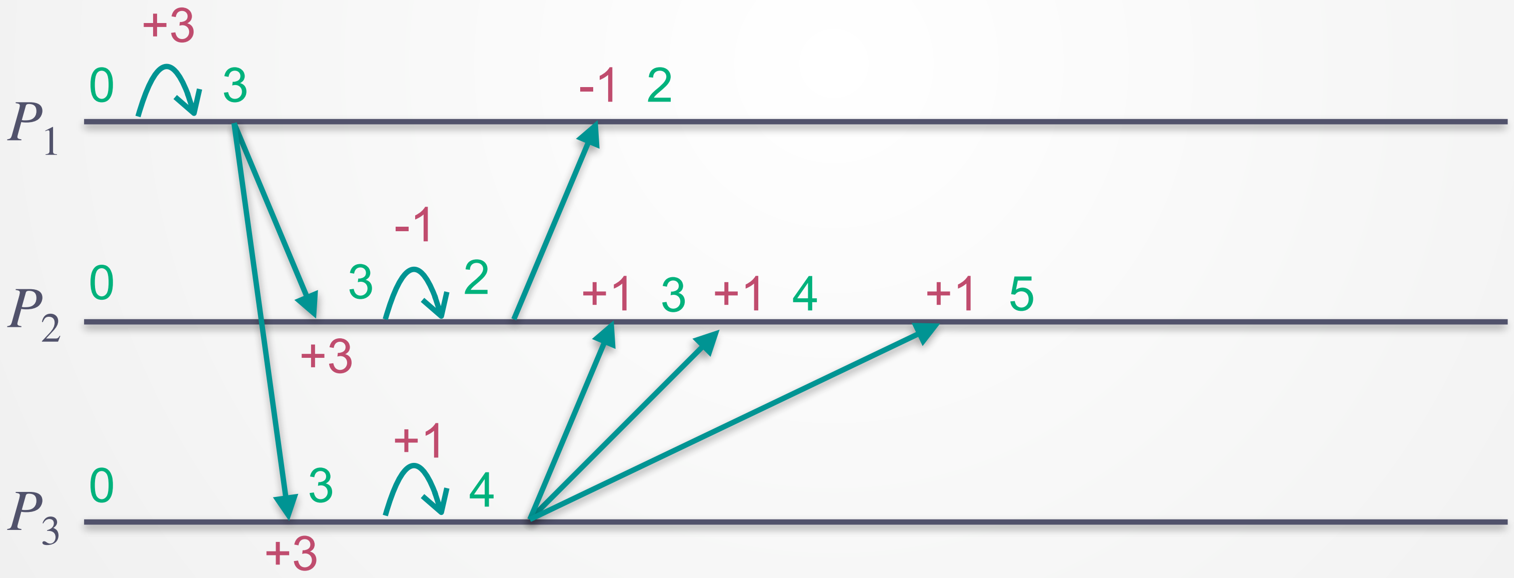
- **MRDTs** : Mergeable Replicated Data Types. Log all local update history in a log. Perform conflict resolution on the update history (similar to git-merge)
- **OT** : Operational Transformation. It is used in Google Docs. Many different approaches, most are not valid. Google Docs re-write concurrent operations based on a set of rules. Also relies on central server to do conflict resolution and relay updates.
- **Known Applications (CRDTs)** : Apple Notes, Fluid (Microsoft), Redis, Riak DB (used by RiotGames-league of legends), Akka Framework

WAIT A MINUTE

- What if we want to disallow counter going below a threshold? e.g., 1

Not Solvable under Strong Eventual Consistency

Managing Global Invariants and Limited Resources requires Coordination (Consensus)



SPECIAL THANKS

- For the inspiring examples and notes from
    - Peter Van-Roy (UCL)
    - Martin Kleppmann (TUM)