Distributed Consistency with CRDTs

- Kishan Sagathiya, @kishansagathiya

Software Engineer at Protocol Labs,
Member of IPFS

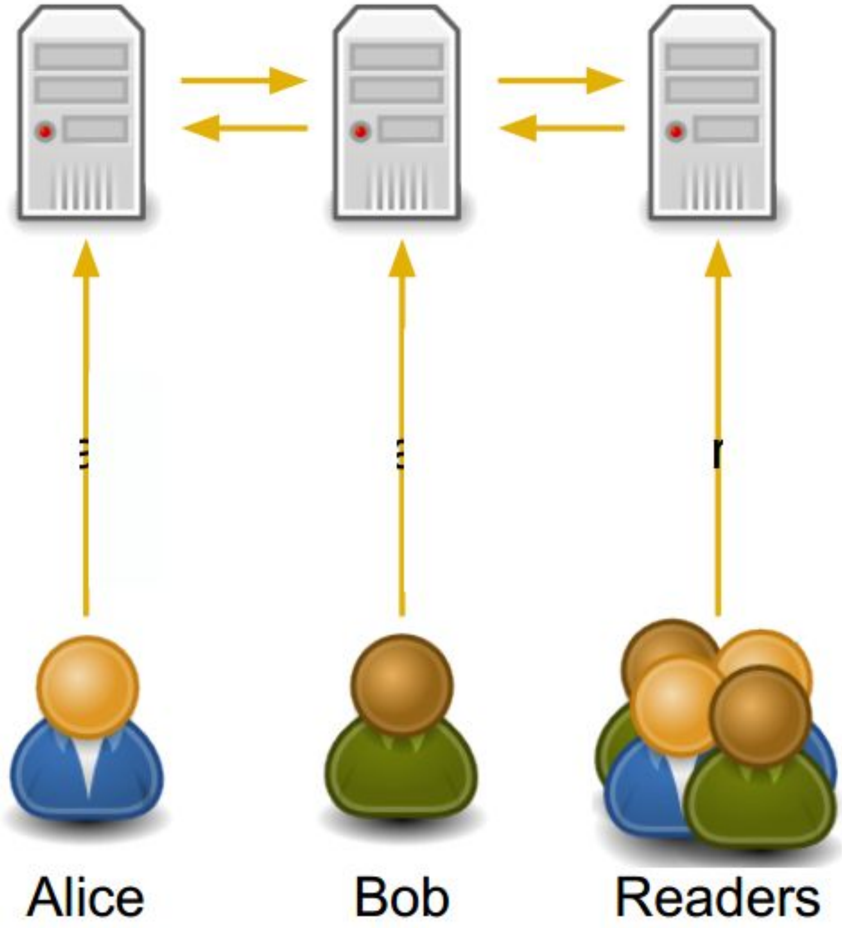
Conflict free

Replicated

Data-Types

Modified by many, but eventually consistent

Distributed Databases



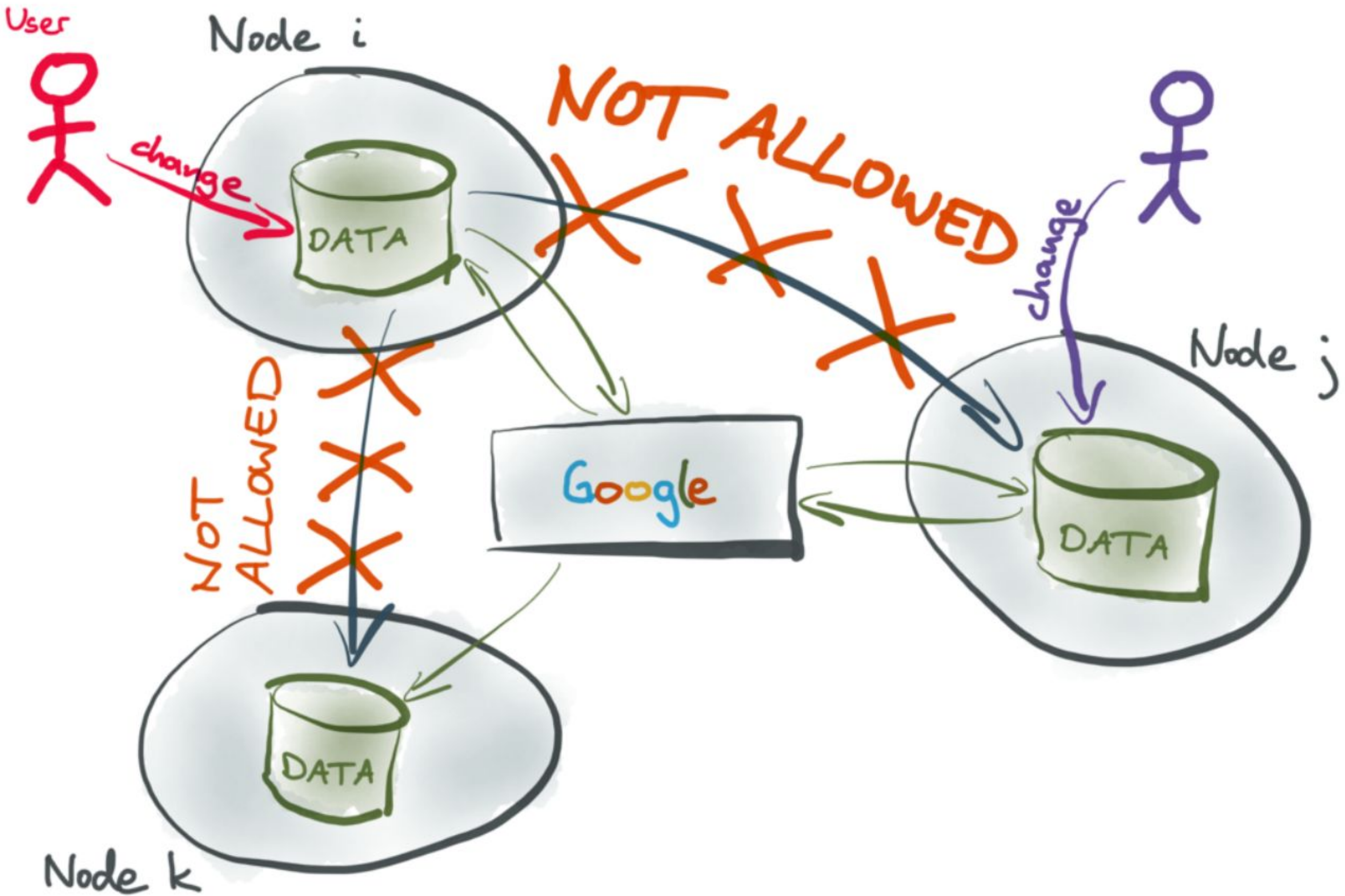
Modified by many, but eventually consistent

Collaborative Text Editors

OPERATIONAL TRANSFORMATION (OT)

- e.g. Google Docs,
  MS Office Online

OPERATIONAL TRANSFORMATION IN GOOGLE DOCS.



- So, *Operational Transformation* requires a server

- Can we do it *without a server*? YES

  https://peerpad.net/

CRDTs

PeerPad α

What have people built?

- *Redis:* distributed, highly available and scalable in-memory database
- *Automerge*: A JSON-like data structure (a CRDT) that can be modified concurrently by different users, and merged again automatically.
- *Orbitdb*: Peer-to-Peer Databases for the Decentralized Web
- *Riak*: decentralized datastore
- *PeerPad*: is a real-time collaborative text editor
- *TomTom GPS* uses it for data synchronization
- *Teletype for atom*: collaborate on code in real time
- *Chat in League of Legends*
- *Cosmos DB* by Microsoft

And other things….

CRDTs are data types which provide *strong eventual consistency* among different *replicas* in a distributed system by requiring some properties from the *state* and/or the *operations* applied to modify it.

Strong Eventual Consistency

If two replicas have received the *same updates*,
their state will be the *same*

State based CRDTs (*Convergent* CRDTs)

Operation based CRDTs (*Commutative* CRDTs)

Operation based CRDTs

Operations that modify states must be *commutative*

$A - 3 + 4 \qquad A + 4 - 3 \qquad$ ✓

$A - 3 * 4 \qquad A * 4 - 3 \qquad$ ✗

Operation based CRDTs

Exactly once delivery semantics

Sum(A, 3) ✗

Max (A, 3) ✓

State based CRDTs

In state-based CRDTs, the states in different replicas and different moments form a *monotonic join semilattice*.


JOIN SEMILATTICE YOU SAY!
TELL ME MORE ABOUT THAT
imgflip.com

- less than or equal to

$a \leq b$ or $b \leq a$

- incomparable

$a \parallel b$

- join

$a \vee b$

- An *order* is a binary relation ≤ on a set S, written <S,≤>

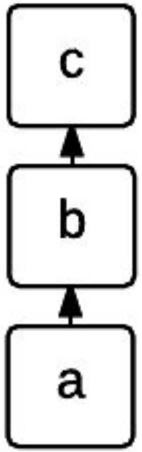
- examples

  less than or equal to 2 ≤ 4

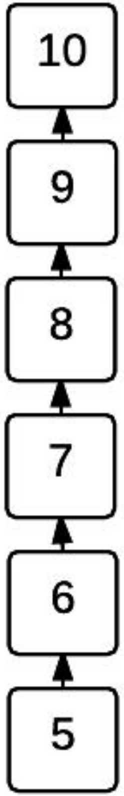
  descendent-of daughter ≤ mother

- *Total Order*

Comes-before order



Less Than or Equal To Order

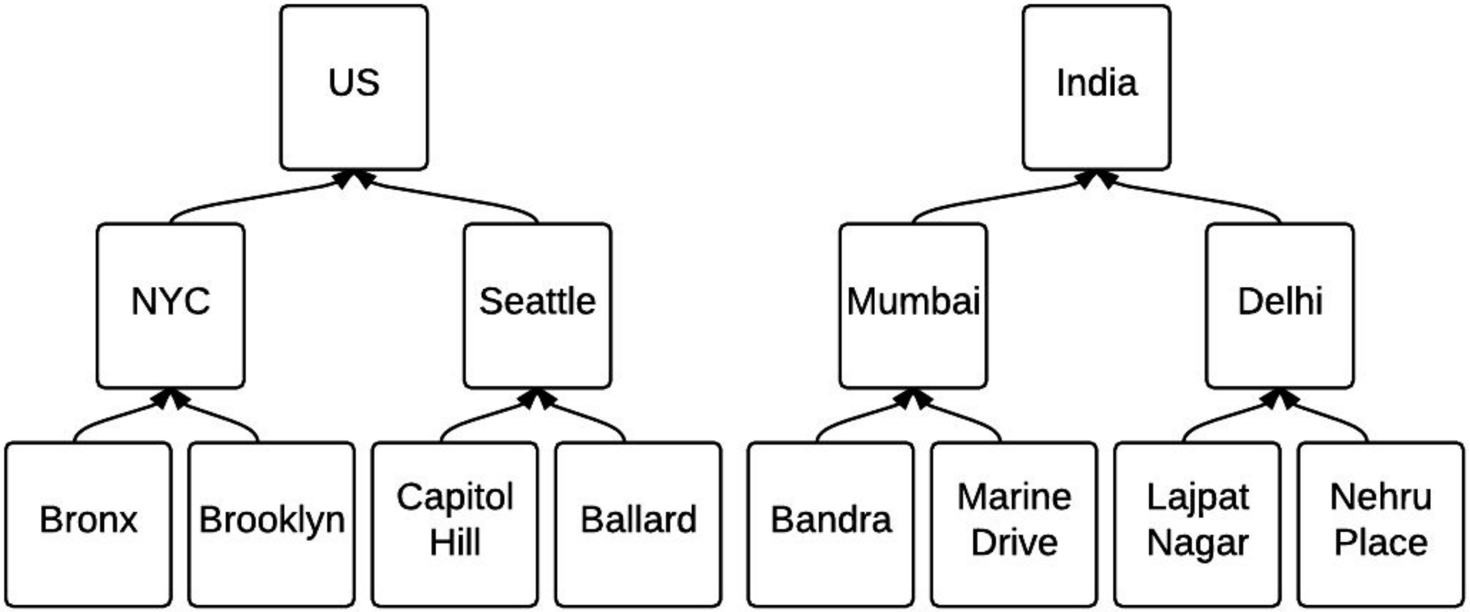


- *Partial Order*

Seattle ≤ US and Brooklyn ≤ US

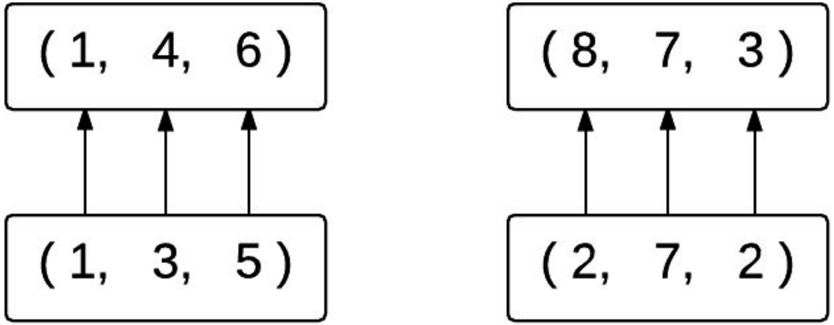
Seattle // NYC and Bronx // Mumbai

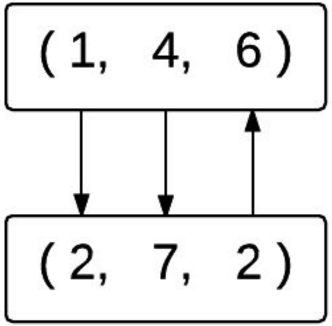
Located-in Order



- A *vector clock timestamp* is a collection of logical timestamps for all the nodes or processes we're interested in.

Happened-Before Order



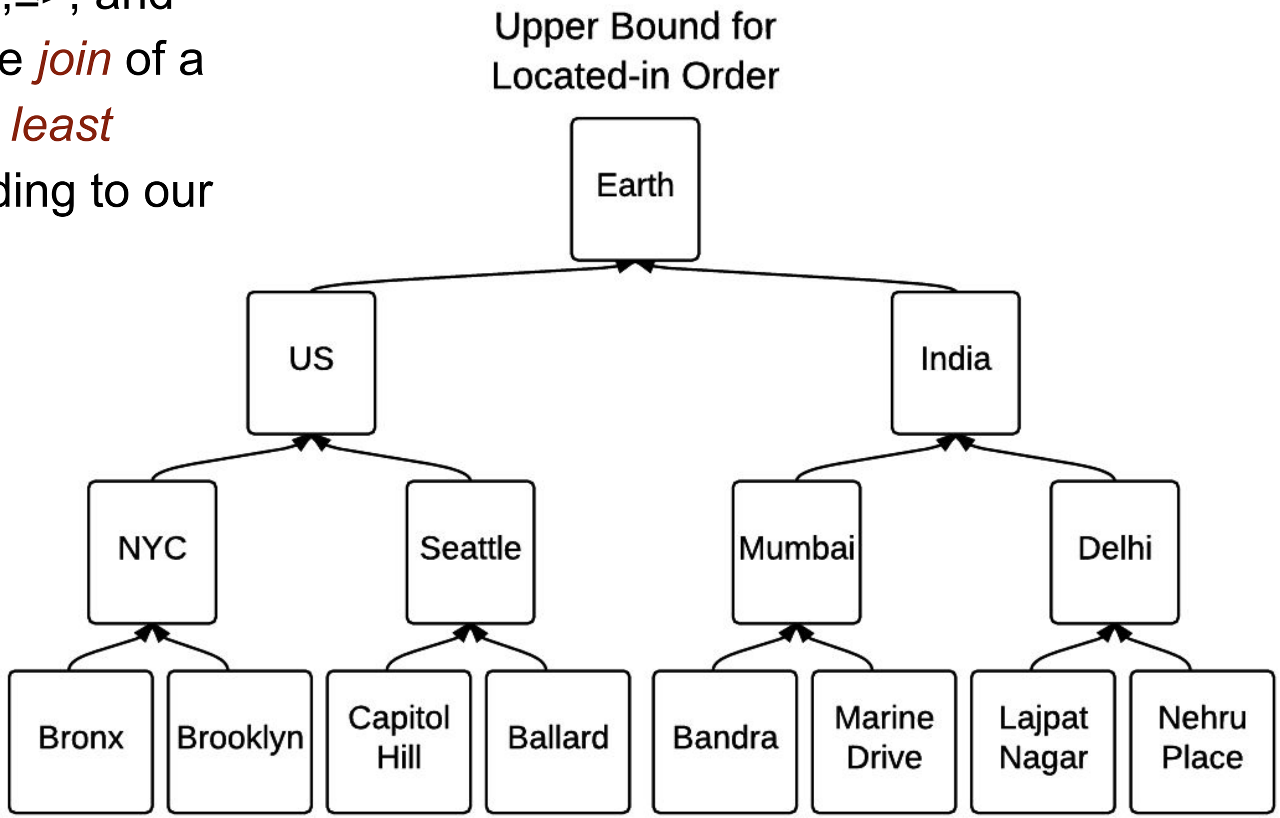


$(1,3,5) \leq (1,4,6)$
$(2,7,2) \leq (8,7,3)$

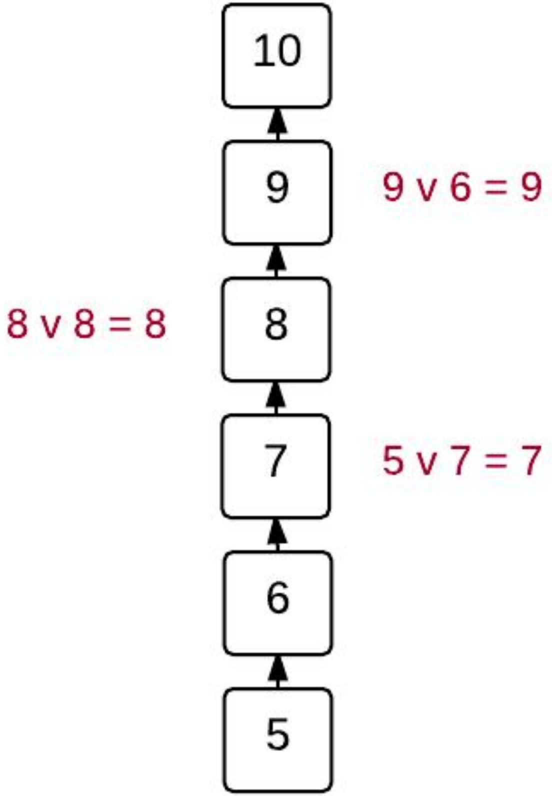
$(1,4,6) \; // \; (2,7,2)$

Join

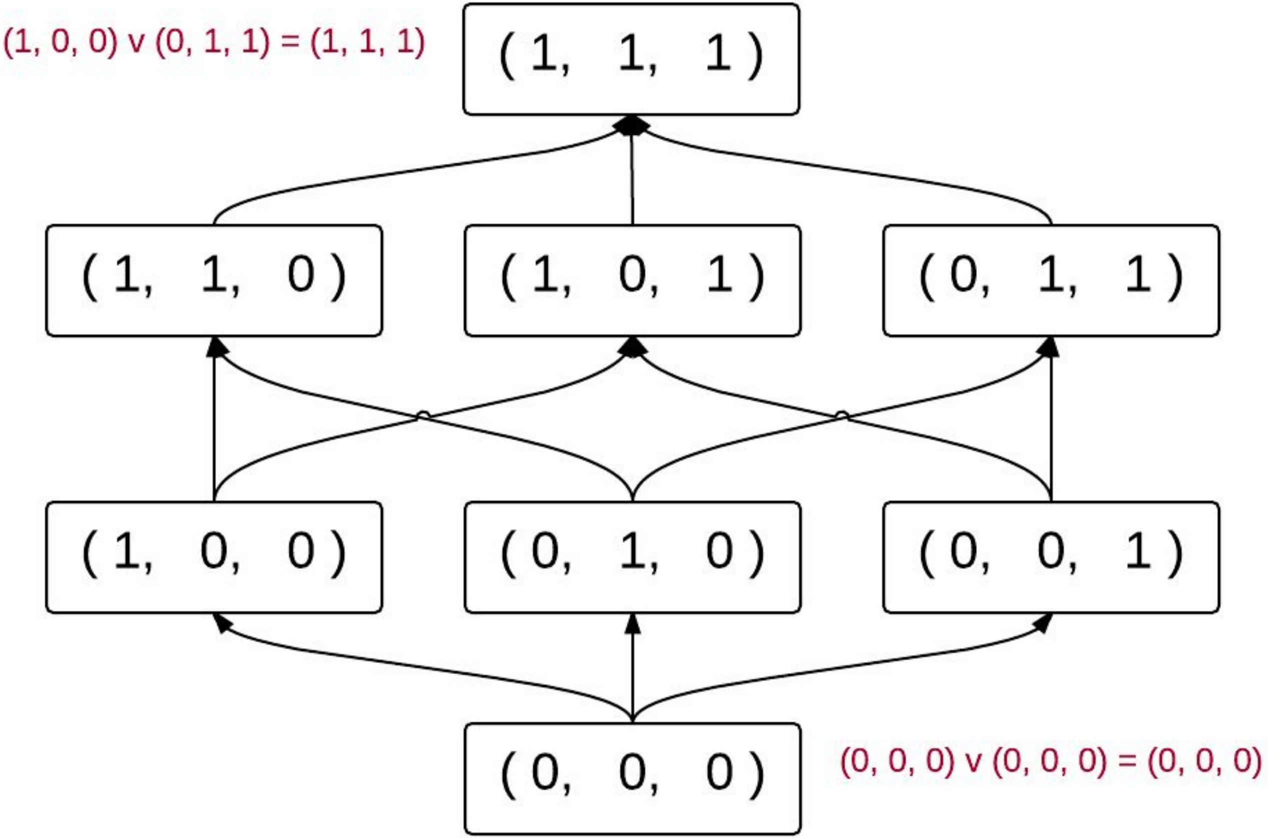
For a set S, an order <S,≤>, and two elements a,b∈S, the *join* of a and b (written a∨b) is a *least upper bound* of S according to our order <S,≤>



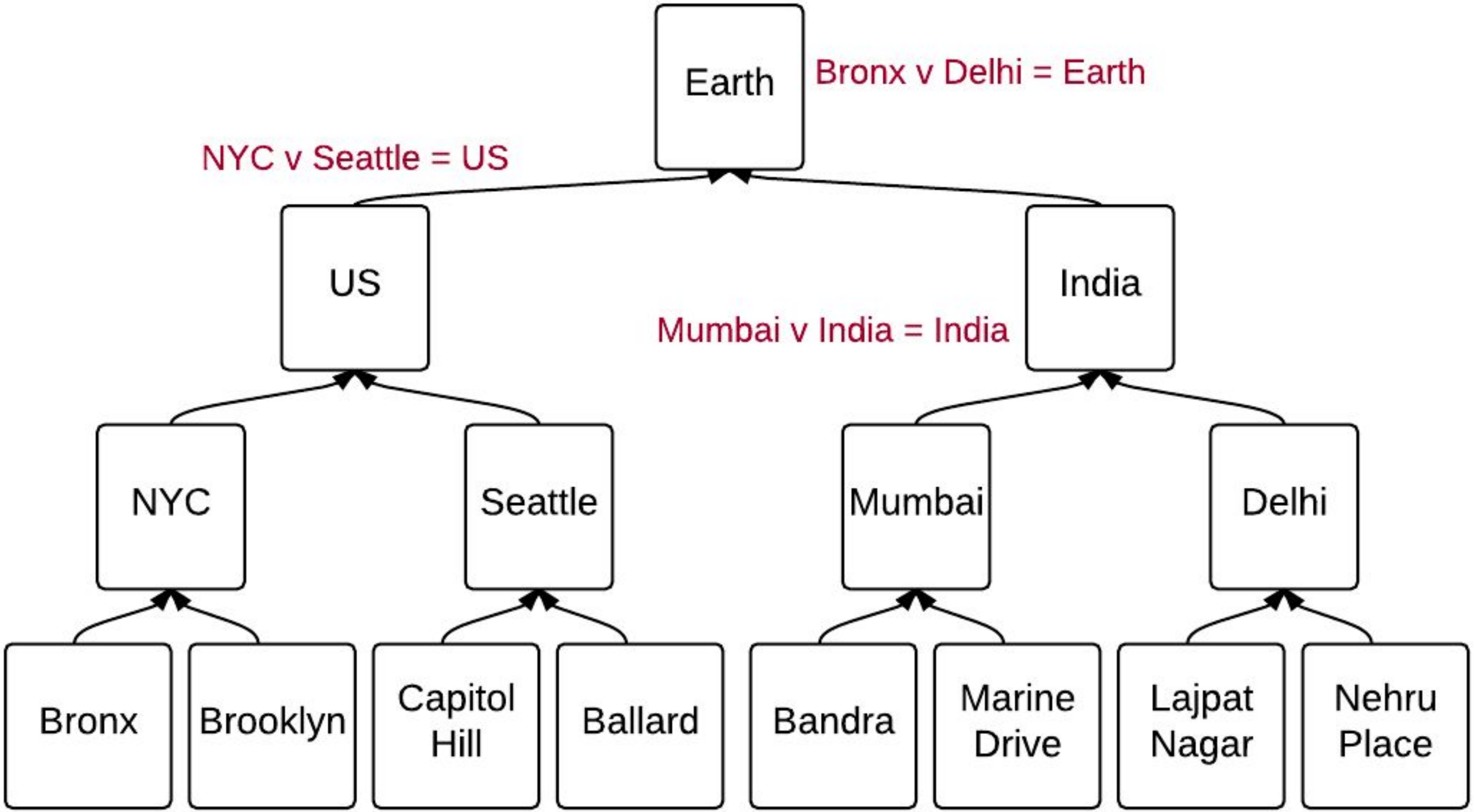
Less Than or Equal To Order



Happened-Before Order



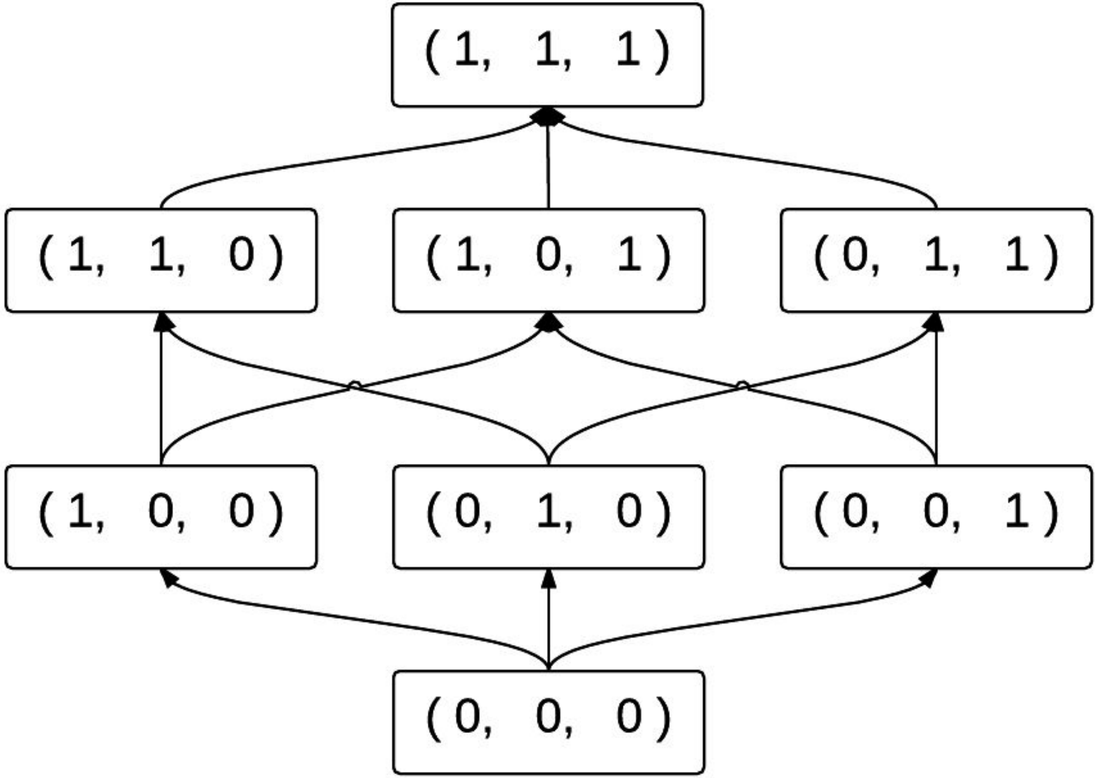
Located-in Order



- A *join semilattice* is an order $<S,\leq>$ for which there exists a join $x \vee y$ for any $x,y \in S$

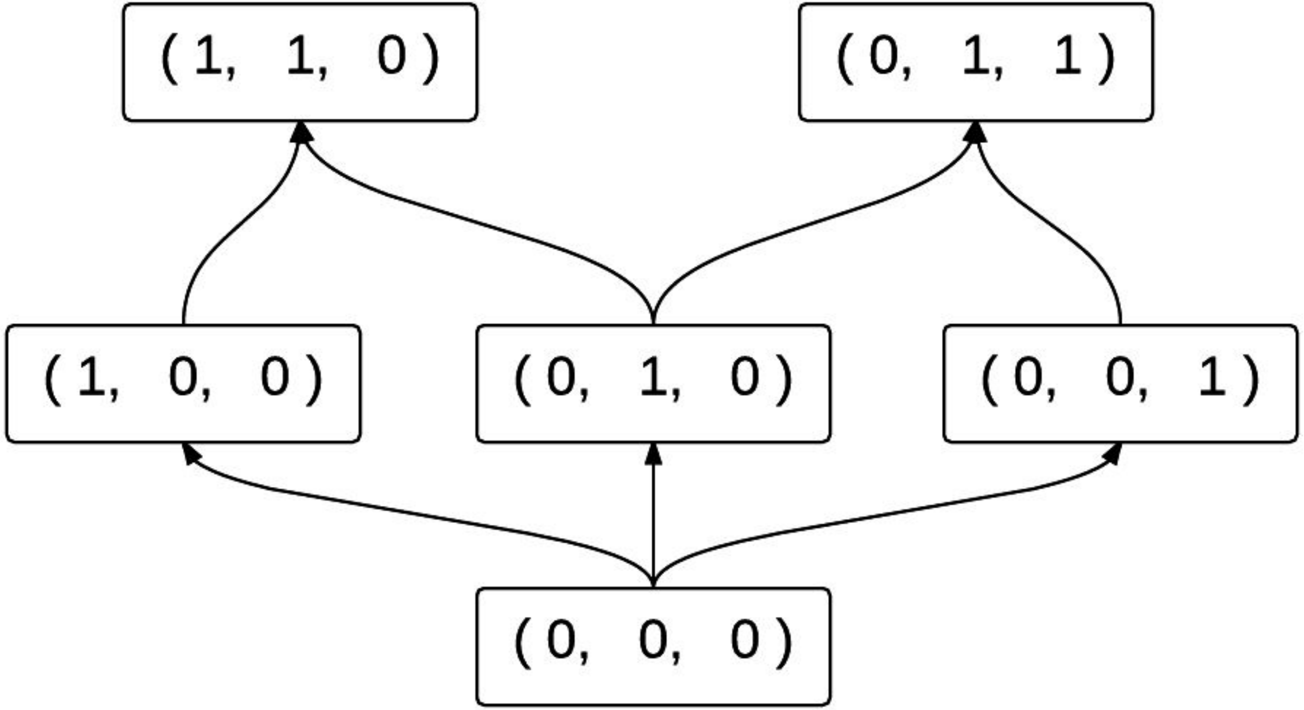
$$(0,0,0) \vee (0,0,1) = (0,0,1)$$
$$(1,0,0) \vee (0,1,1) = (1,1,1)$$
$$(1,0,1) \vee (1,0,1) = (1,0,1)$$
$$(0,1,0) \vee (0,0,1) = (0,1,1)$$

Happened-Before Order

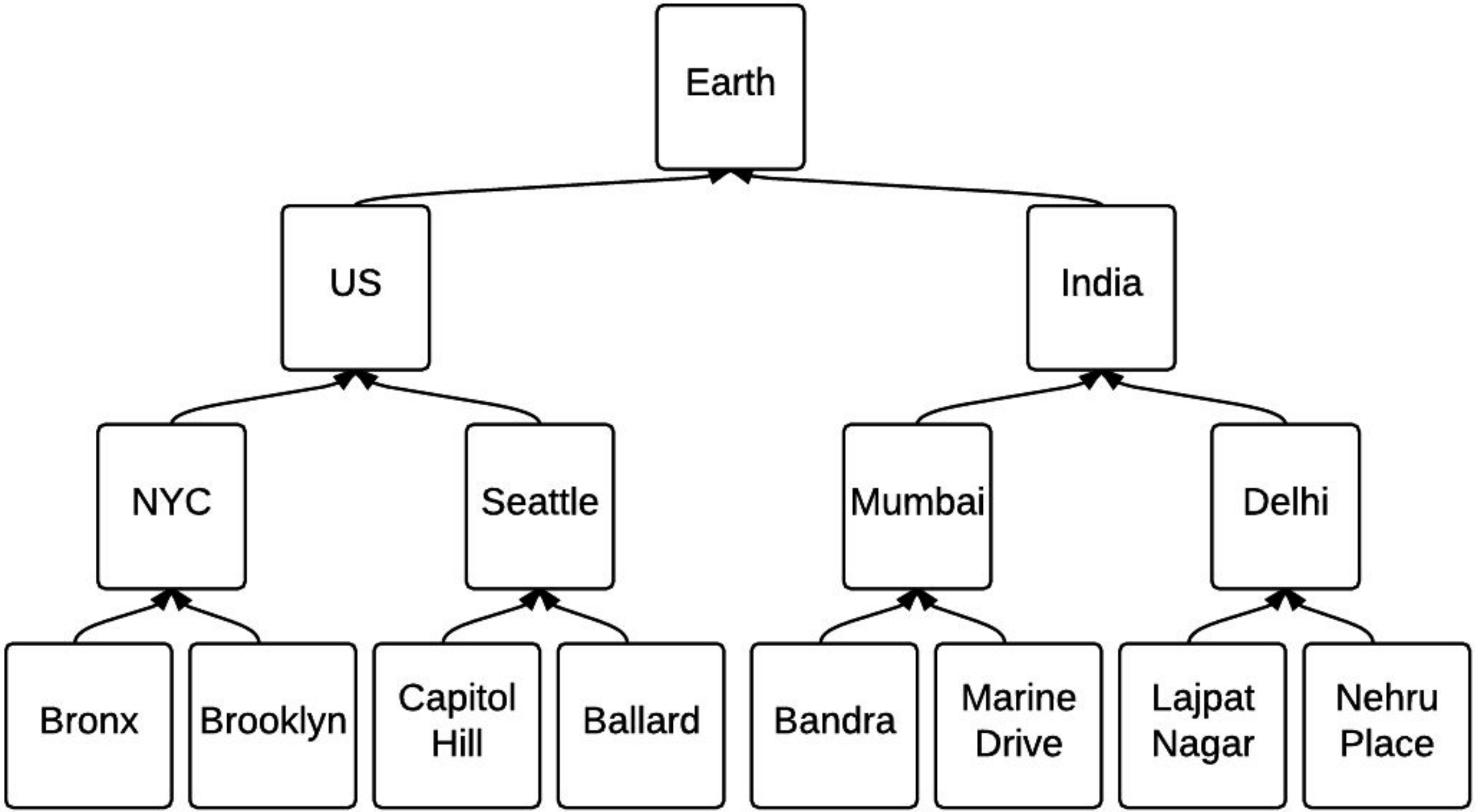


NOT A JOIN SEMI-LATTICE!

(1, 1, 0) v (0, 1, 1)
does not exist



Located-in Order



Joins obey three laws

- *Commutativity*: $a \vee b = b \vee a$
- *Associativity*: $(a \vee b) \vee c = a \vee (b \vee c)$
- *Idempotence*: $a \vee a = a$

- Joins tend to move *“upwards”*, so do merges of state-based CRDTs tend to converge on the *One True Value*

Convergent CRDTs

- *State* (elements of set)
- *merge()* function

$$merge(1, 3) = 3$$
$$merge(9, 5) = 9$$
$$merge(8, 8) = 8$$

$$merge((1, 0, 0), (0, 1, 1)) = (1, 1, 1)$$
$$merge((0, 0, 0), (2, 0, 2)) = (2, 0, 2)$$
$$merge((5, 3, 1), (1, 9, 2)) = (5, 9, 2)$$

- *merge()* is max() here
- Can we use sum()?

$$merge(Seattle, Mumbai) = Earth$$
$$merge(Bronx, NYC) = NYC$$
$$merge(Mumbai, Delhi) = India$$

- *System*: set of available state at the moment

  [2, 5, 7]

- *Background set*: all integers
- *Value* of the System: upper bound of corresponding semilattice diagram (consistent value)

  Value([2,5,7])=7

- The *order of merges* doesn't matter. This is guaranteed by the *associativity* and *commutativity* of joins.

- It doesn't matter how many times we *repeat* a particular merge. This is guaranteed by the *idempotence* of joins.

Why do we care about this ?

Implementing a CvRDT

counter with a simple interface:

- *increment()*: increment the counter
- *value()*: gets the value of the counter

- 3 nodes X, Y, Z
- Set includes all integers
- merge() is max()

Imagine the following history:

- Start with 0 on all nodes
- Node 1 increments 3 times
- Node 2 increments 2 times
- Node 3 increments 1 time

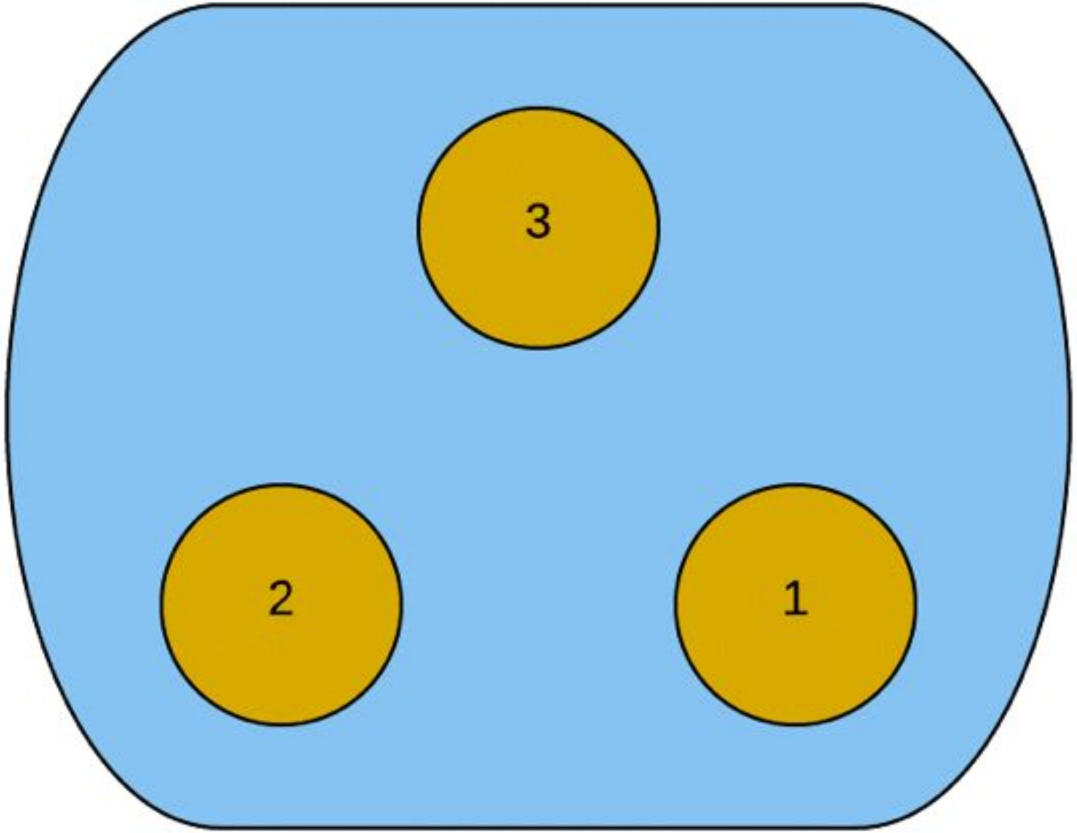
What should be the final result ?

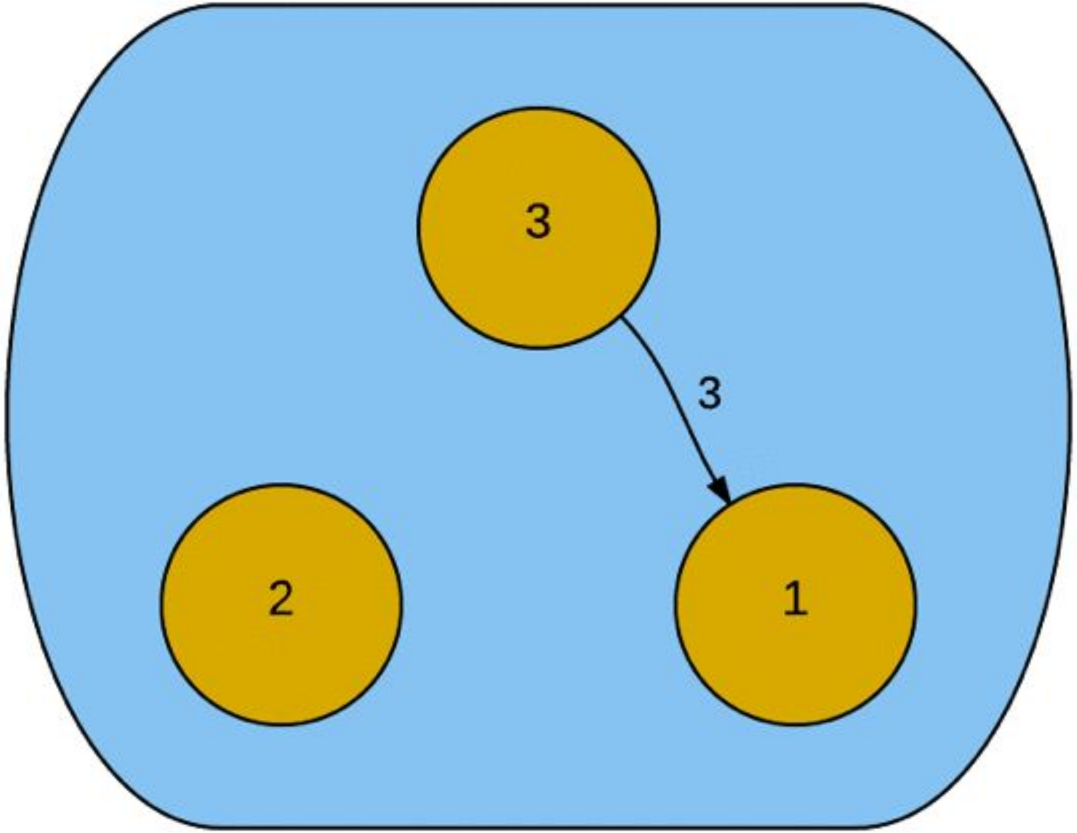
Imagine the following history:

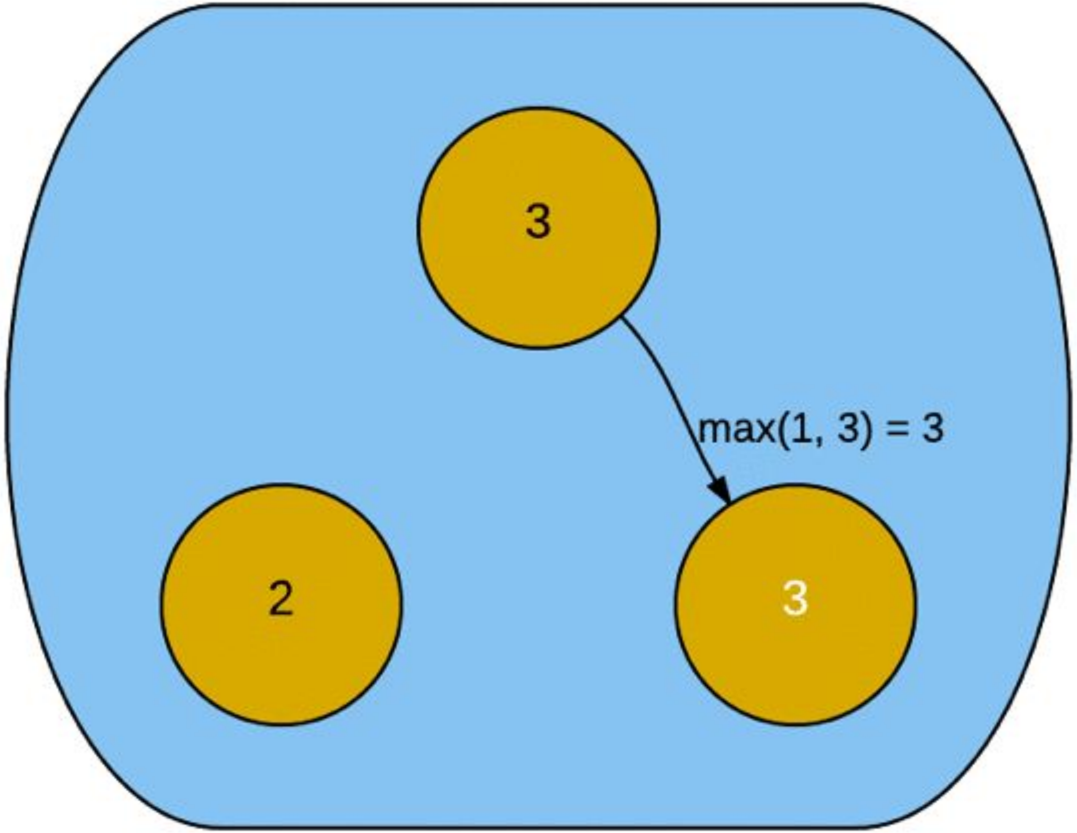
- Start with 0 on all nodes
- Node 1 increments 3 times
- Node 2 increments 2 times
- Node 3 increments 1 time

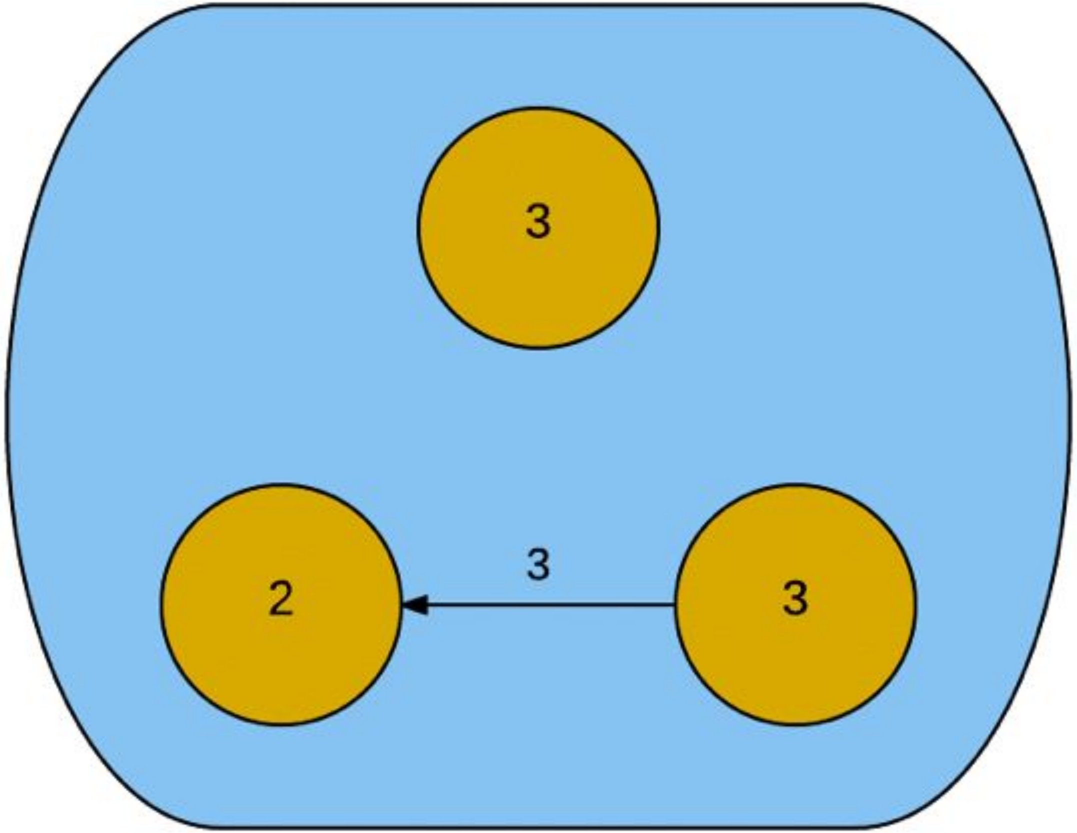
What should be the final result ?

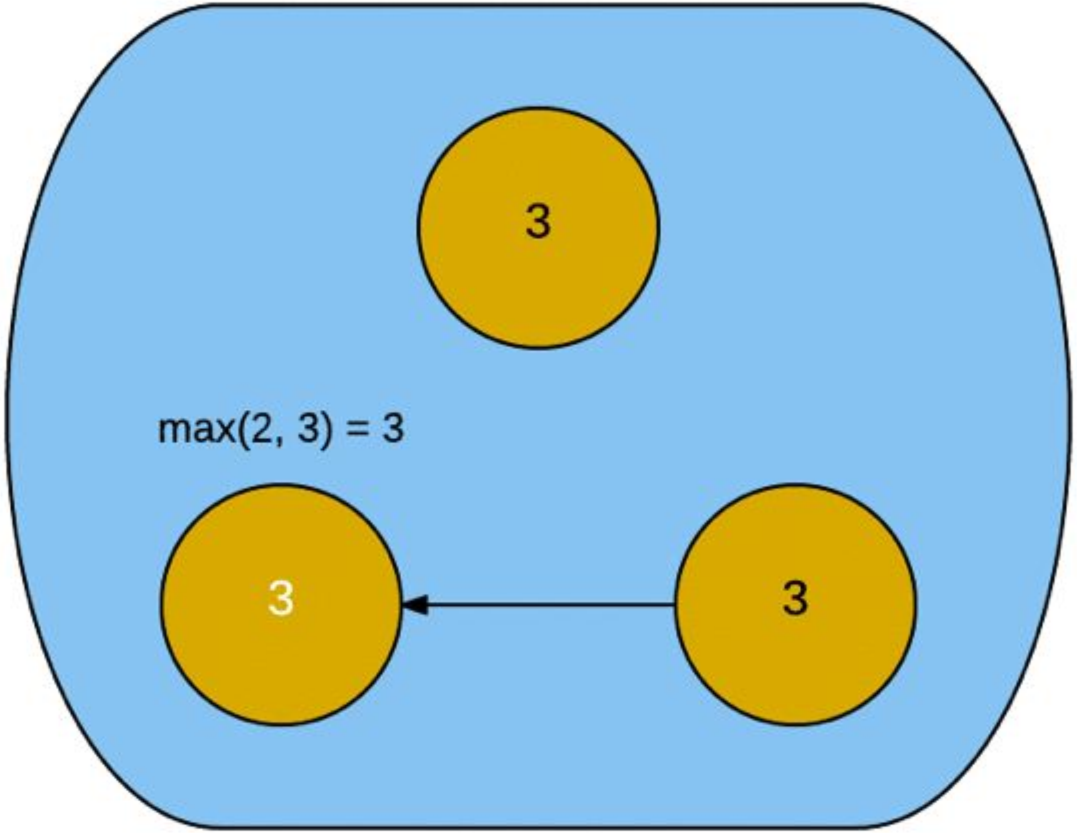
6


3
2
1


3
3
2
1


3
max(1, 3) = 3
2
3


3
2
3
3


3
max(2, 3) = 3
3
3

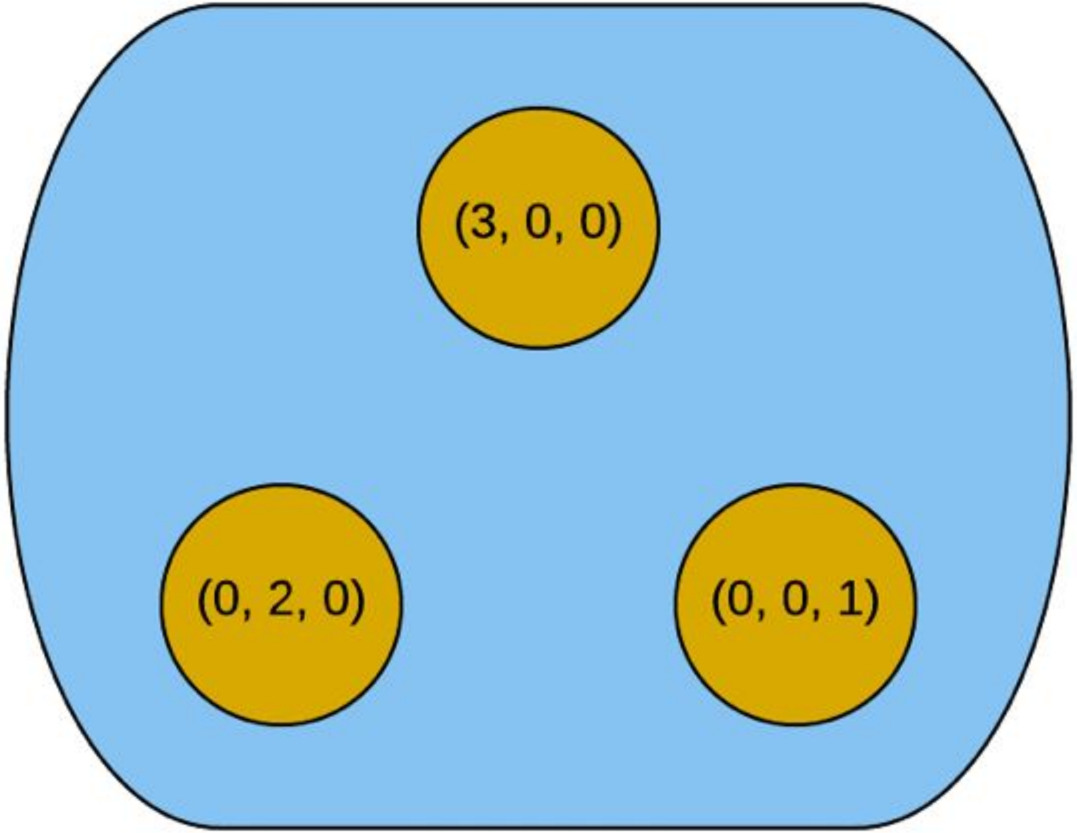
- Weren’t we supposed to get **6**?

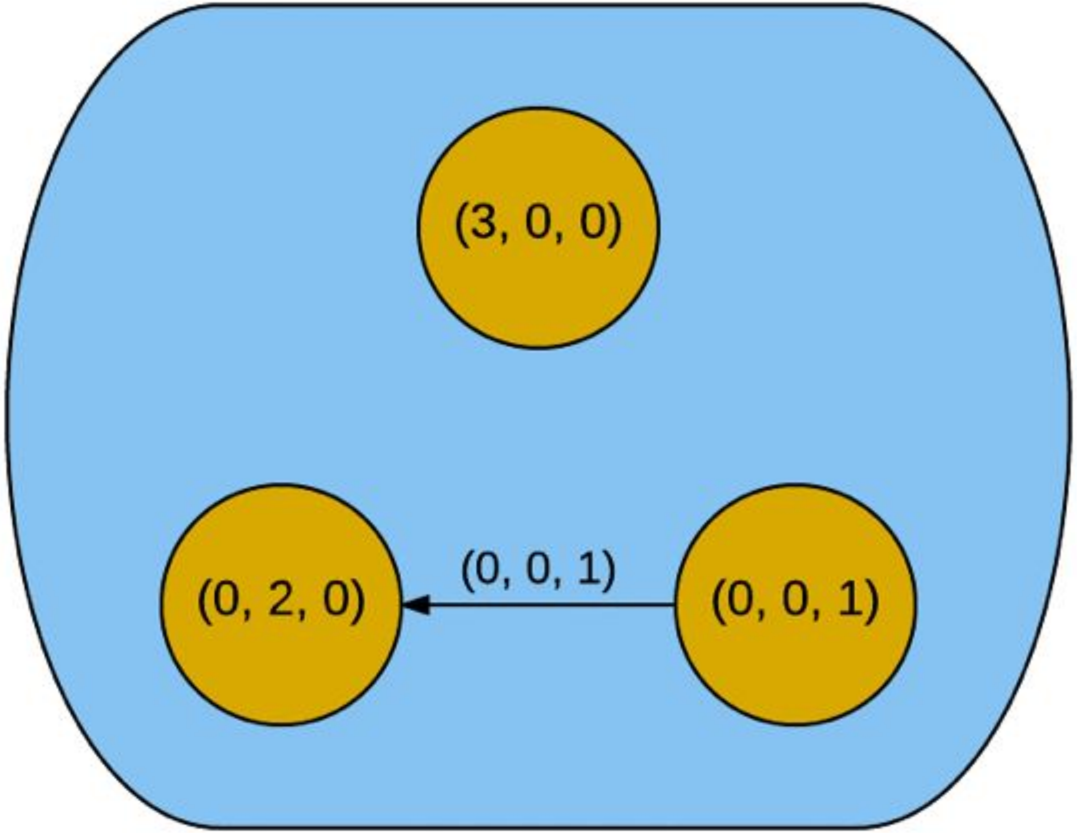
- Let’s use a better approach

- Instead of *integers* use *vector of integers*
- *Value*: sum of all elements in the vectors

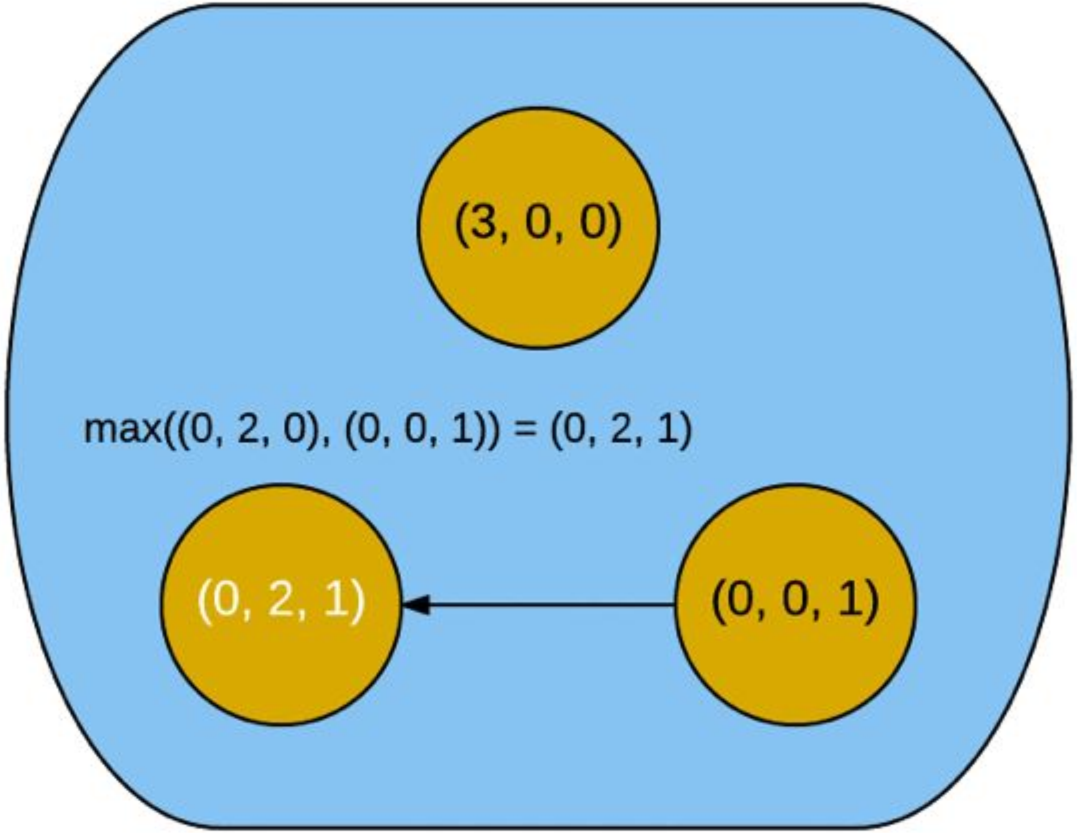
Last example becomes:

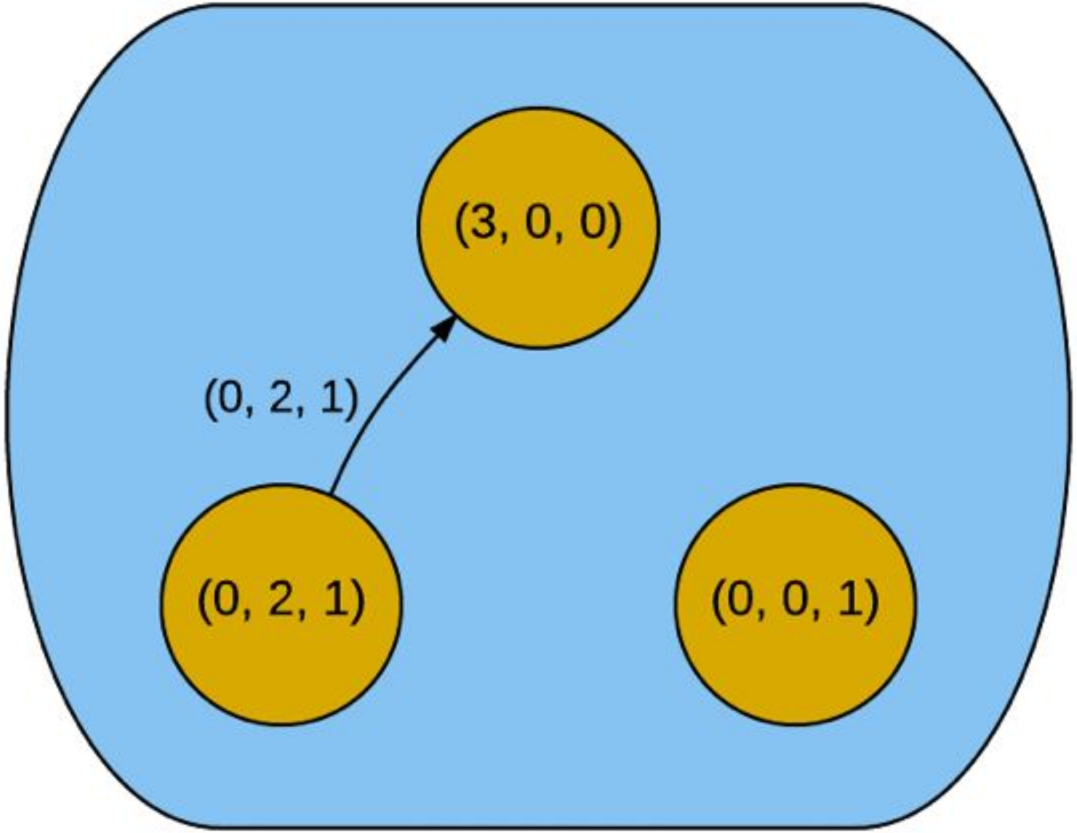
- X: (3, 0, 0)
- Y: (0, 2, 0)
- Z: (0, 0, 1)

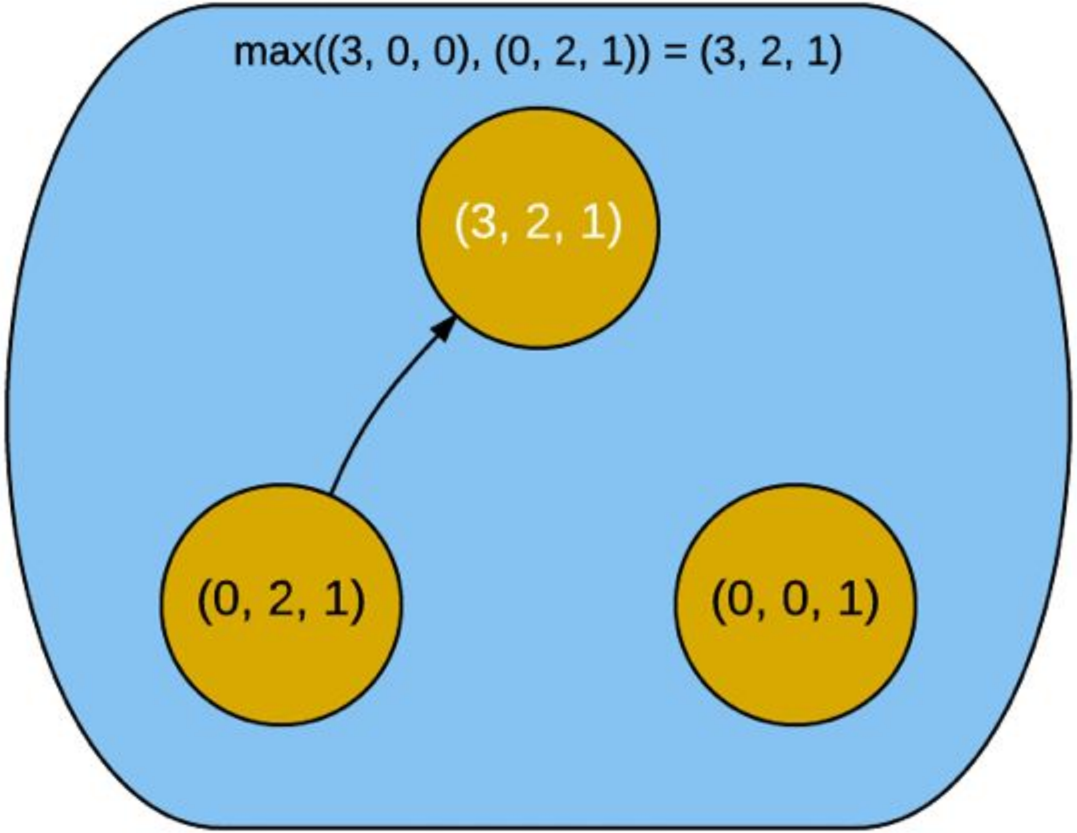

(3, 0, 0)
(0, 2, 0)
(0, 0, 1)

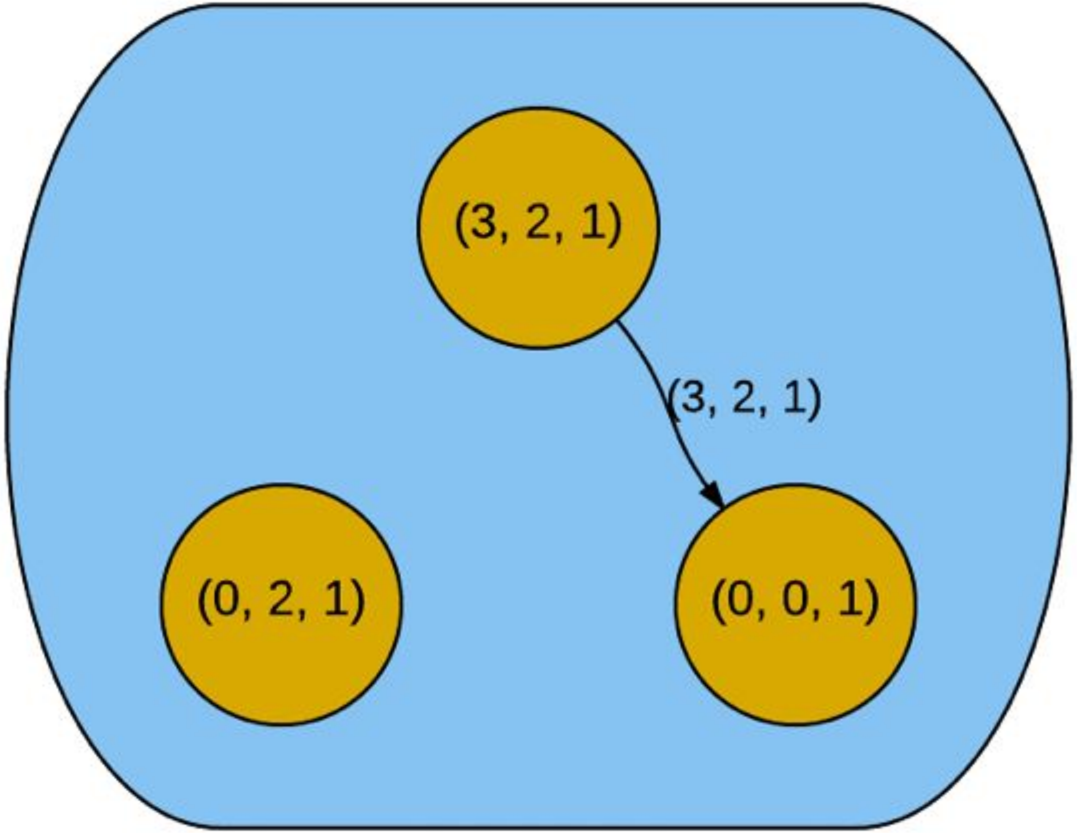


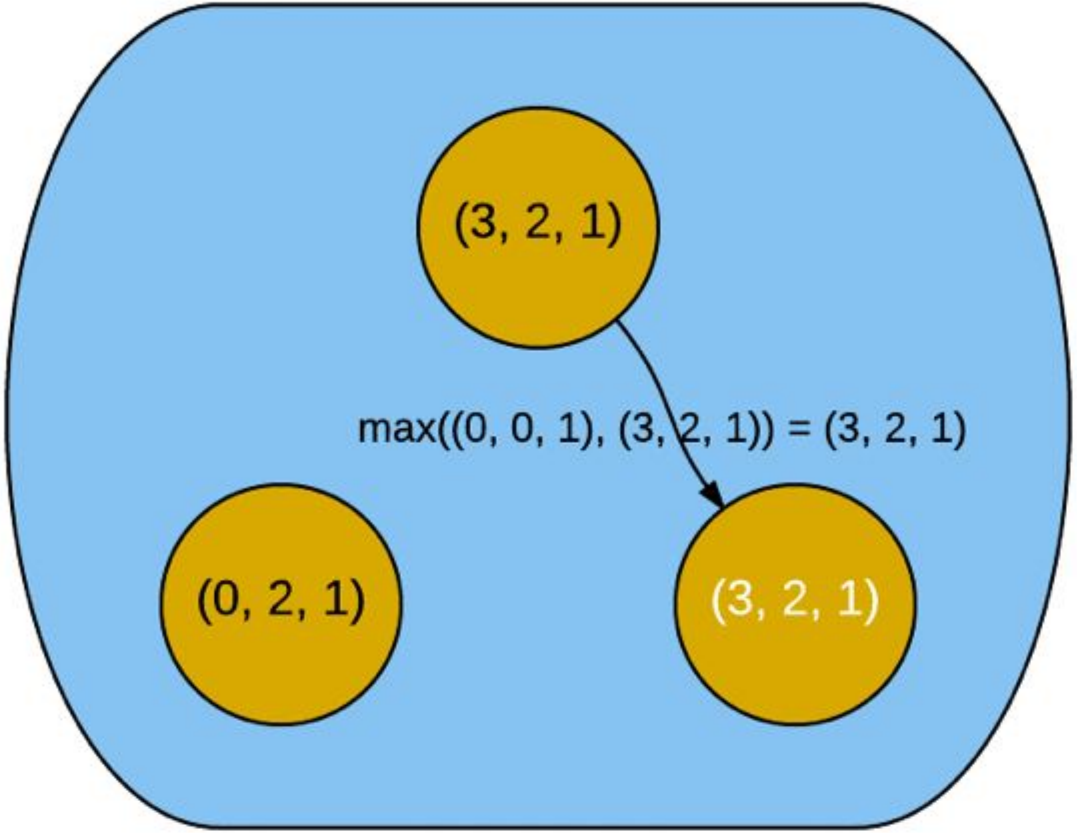
(3, 0, 0)
(0, 2, 0)
(0, 0, 1)
(0, 0, 1)


(3, 0, 0)
max((0, 2, 0), (0, 0, 1)) = (0, 2, 1)
(0, 2, 1)
(0, 0, 1)

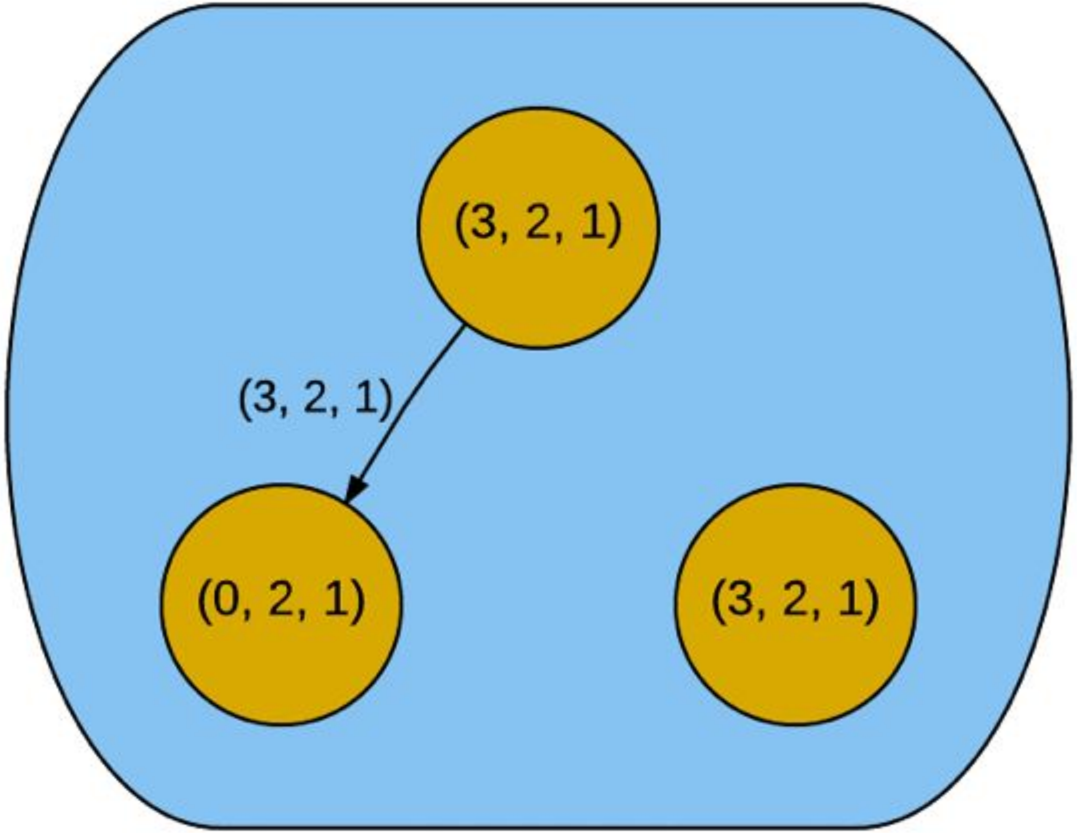

(3, 0, 0)
(0, 2, 1)
(0, 2, 1)
(0, 0, 1)

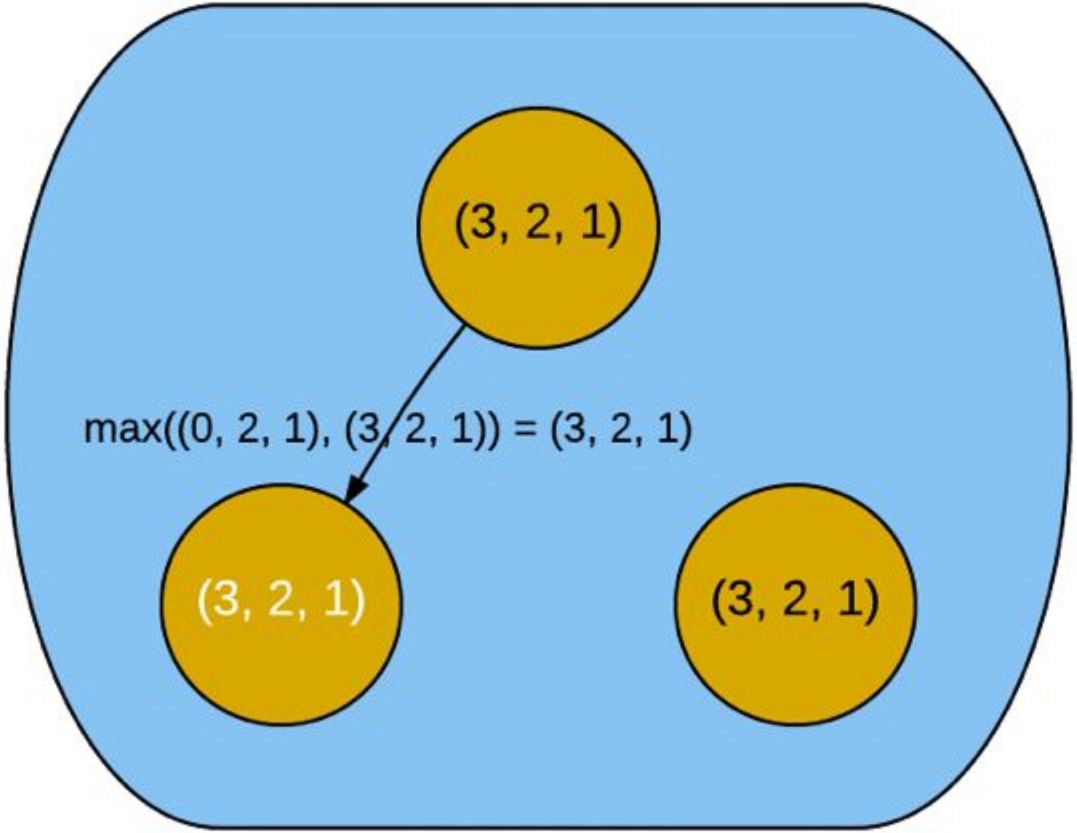

max((3, 0, 0), (0, 2, 1)) = (3, 2, 1)
(3, 2, 1)
(0, 2, 1)
(0, 0, 1)


(3, 2, 1)
(3, 2, 1)
(0, 2, 1)
(0, 0, 1)

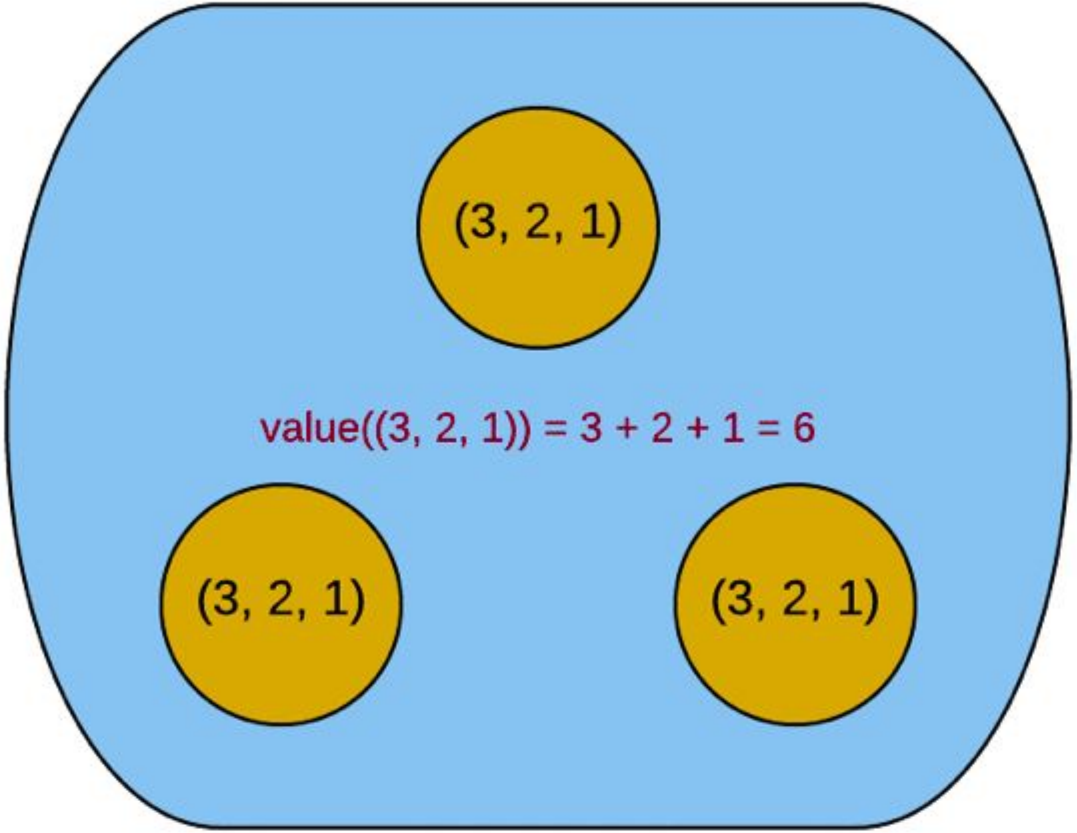


(3, 2, 1)
max((0, 0, 1), (3, 2, 1)) = (3, 2, 1)
(0, 2, 1)
(3, 2, 1)

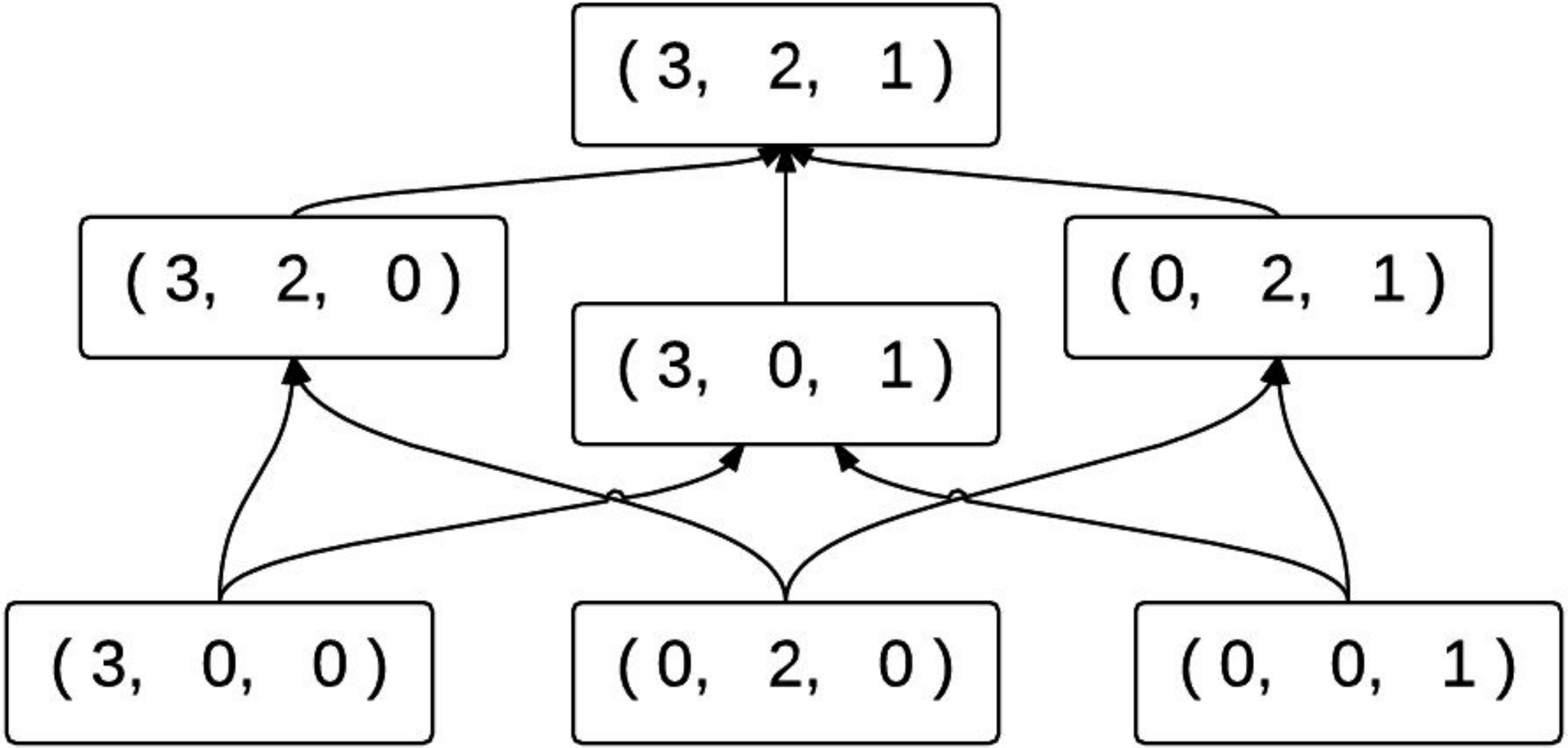

(3, 2, 1)
(3, 2, 1)
(0, 2, 1)
(3, 2, 1)



(3, 2, 1)
max((0, 2, 1), (3, 2, 1)) = (3, 2, 1)
(3, 2, 1)
(3, 2, 1)


(3, 2, 1)
value((3, 2, 1)) = 3 + 2 + 1 = 6
(3, 2, 1)
(3, 2, 1)

Semi-Lattice for Our System



- Create data-types that follow these *requirements*

References

- http://jtfmumm.com/blog/2015/11/17/crdt-primer-1-defanging-order-theory/
- http://jtfmumm.com/blog/2015/11/24/crdt-primer-2-convergent-crdts/
- CRDTs: Consistency without concurrency control https://arxiv.org/pdf/0907.0929.pdf
- "CRDTs Illustrated" by Arnout Engelen https://www.youtube.com/watch?v=9xFfOhasiOE
- CRDTs and the Quest for Distributed Consistency by Martin Kleppmann https://www.youtube.com/watch?v=B5NULPSiOGw
- Paxos Simplified https://www.youtube.com/watch?v=SRsK-ZXTeZ0
- An extensive list of articles here https://github.com/ipfs/research-CRDT/
- https://en.wikipedia.org/wiki/Conflict-free_replicated_data_type