INTRO TO RIAK

Riak Overview

Riak

Distributed

Riak

Distributed, replicated, highly available

Riak

Distributed, highly available, eventually consistent

Riak

Distributed, highly available, eventually consistent, key-Value Database

Riak

Distributed, highly available, eventually consistent,
key-Value Database
Mainly written in Erlang!

Riak

- Modelled after Amazon Dynamo*
  - see annotated version of Dynamo paperwith comparisons to Riak: http://docs.basho.com/riak/latest/references/dynamo/

*https://dl.acm.org/citation.cfm?id=1294281

Amazon Dynamo

- SOSP 2007
- Latency - 100ms of latency cost them 1% in sales.
- “not novel” - synthesis of last 40 years dist-sys research
- Real world application of CS

Riak

- A database
- Key-Value (like a hash table)
- NoSql
- Distributed - Fault Tolerant
- Favours (write) Availability over Consistency

KEY-VALUE STORE

- Simple operations - GET, PUT, DELETE
- Value is opaque (mostly), with metadata
- Extras, e.g.
  - Secondary Indexes (2i)
  - MapReduce
  - CRDTs/Search/Time Series etc etc

FAULT TOLERANT

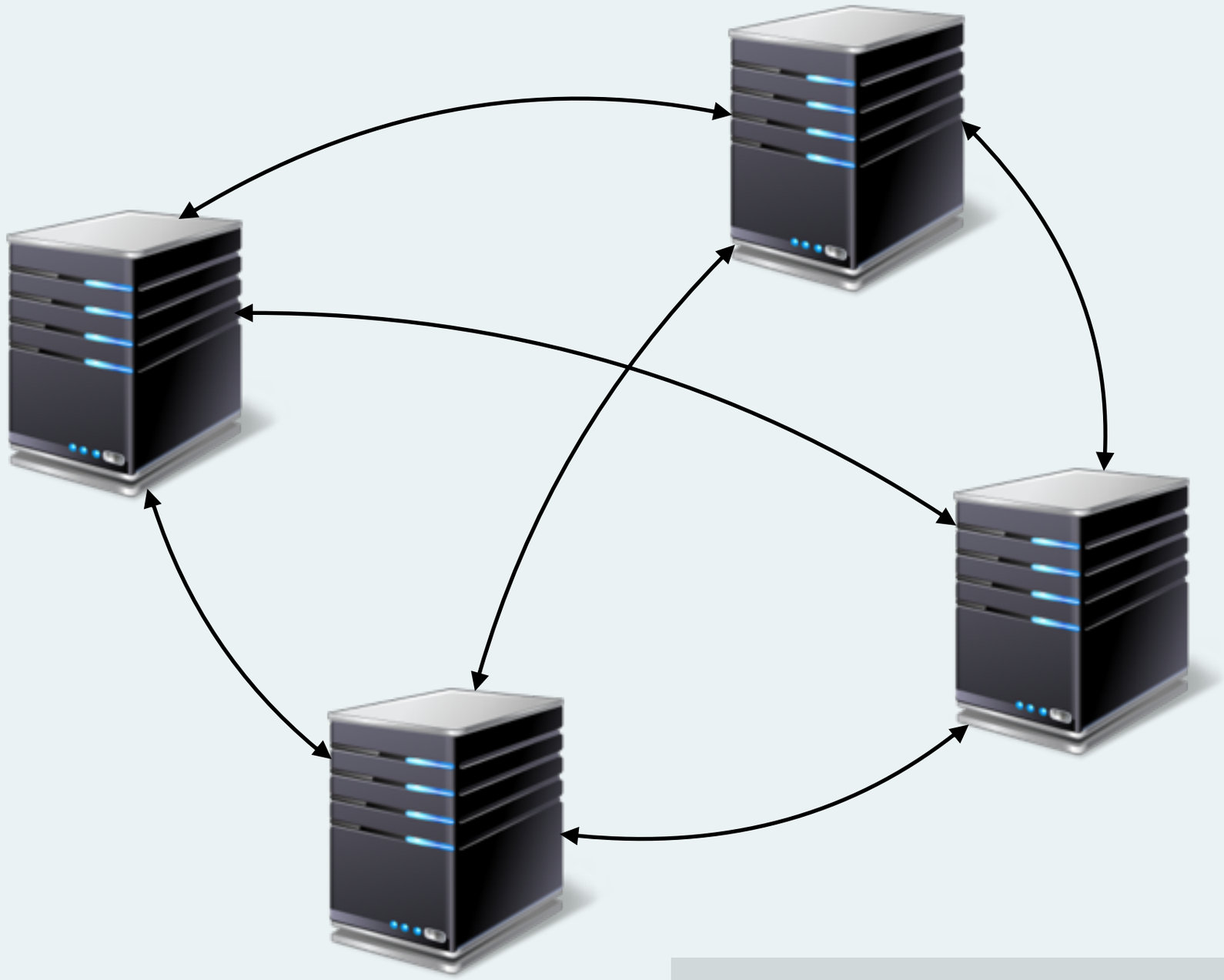
- All nodes participate equally - no single point of failure (SPOF)
- All data is replicated
- Cluster transparently survives...
  - node failure
  - network partitions
- Built on Erlang/OTP (designed for FT)

Riak - Write Available

- Unable to write mean lost dollars
  - Amazon Shopping Cart
- Low Latency matters more than Consistency

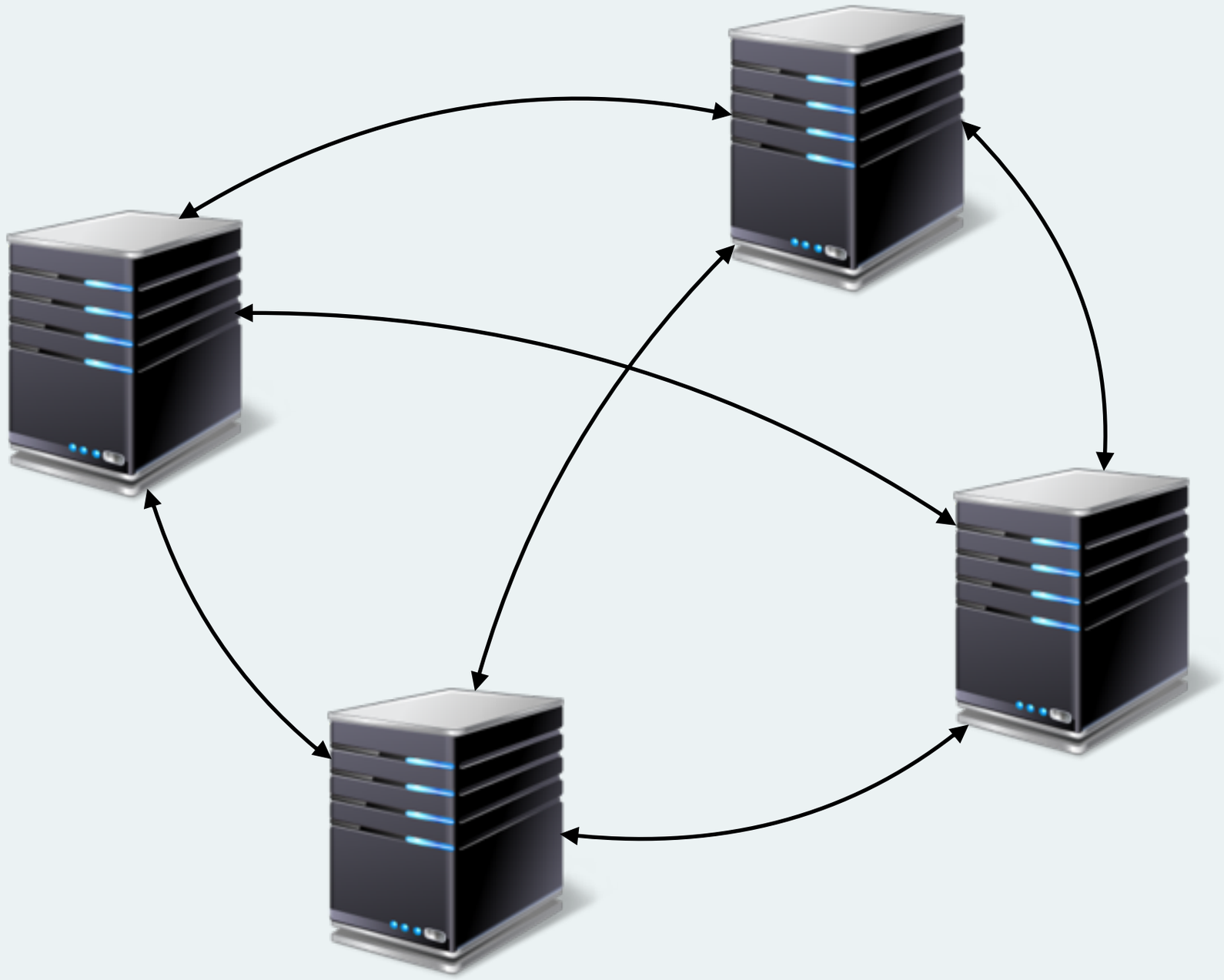
Riak Overview

{“key”: “value”}



riak

Distributed



riak

Homogenous

The ring

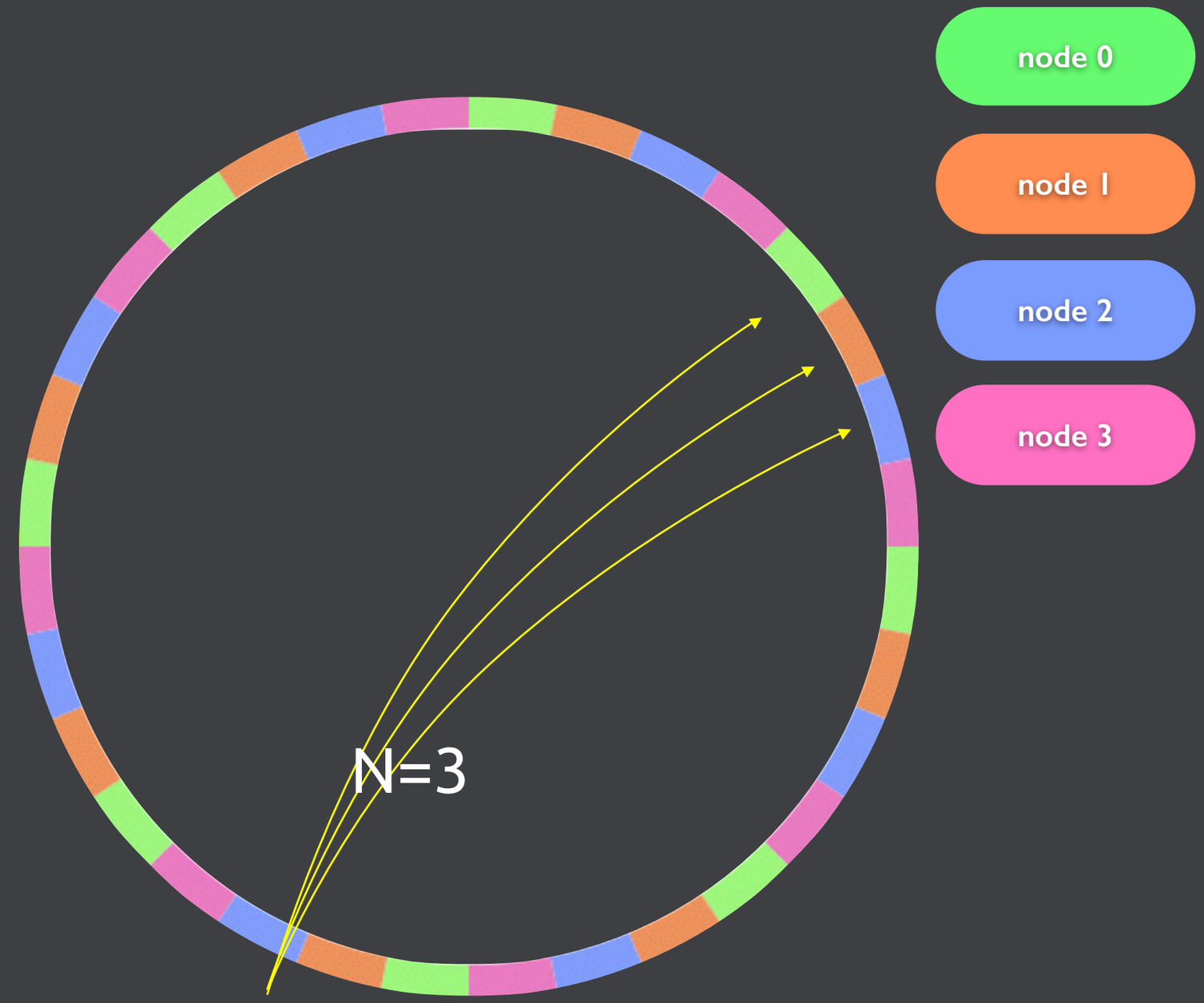
The Ring

- Membership
- Ownership
- Routing
- Abstract and Concrete in Riak

Riak Overview

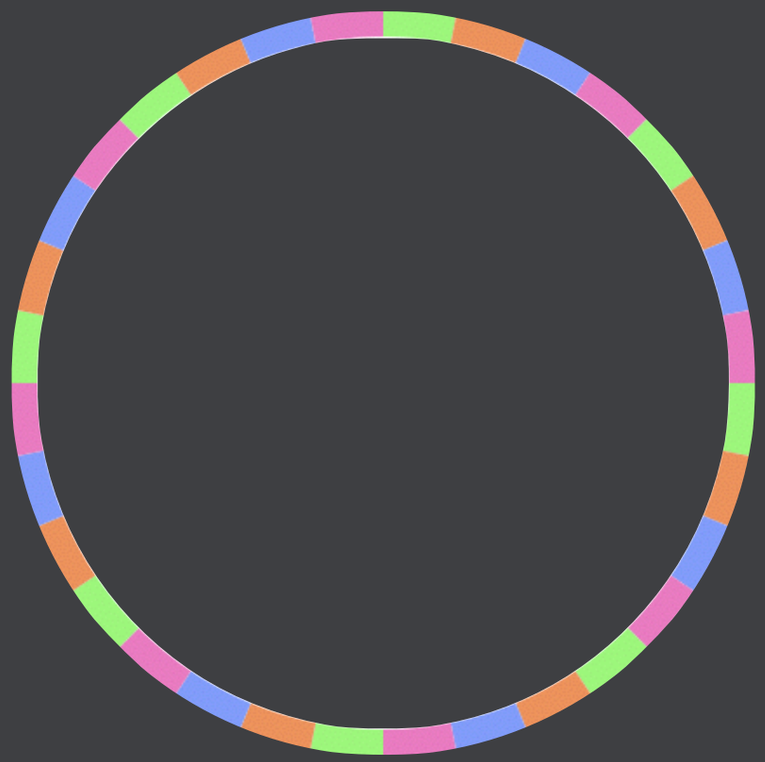
The Ring

- 160-bit integer **keyspace**
- divided into **fixed** number of **evenly-sized partitions/ ranges**
- partitions are **claimed** by nodes in the cluster
- replicas go to the N partitions following the key



hash(“users/clay-davis”)

The Ring - Consistent Hashing

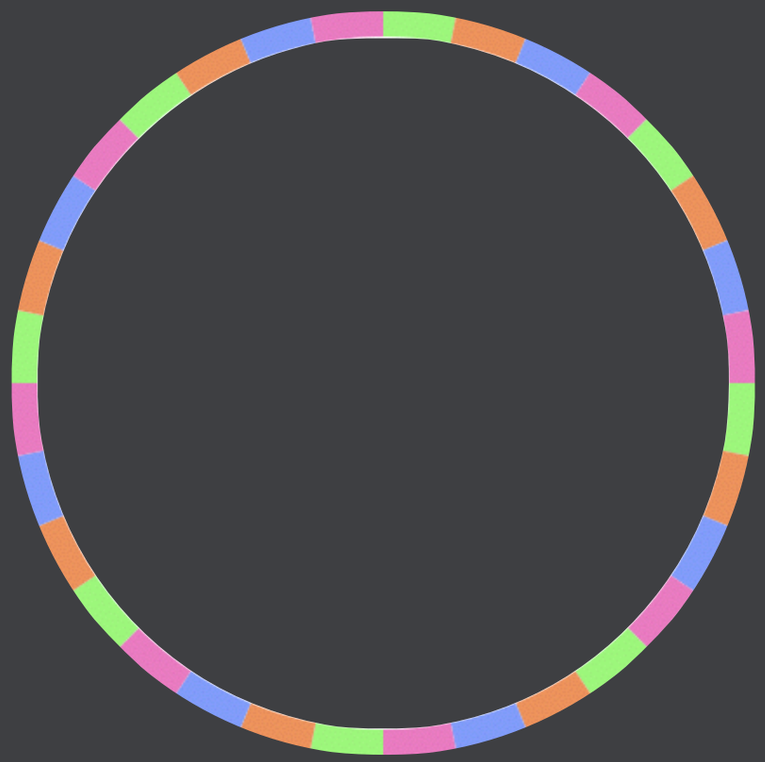


supervisor process

basic unit of concurrency

Process “knows” its range

VNODES

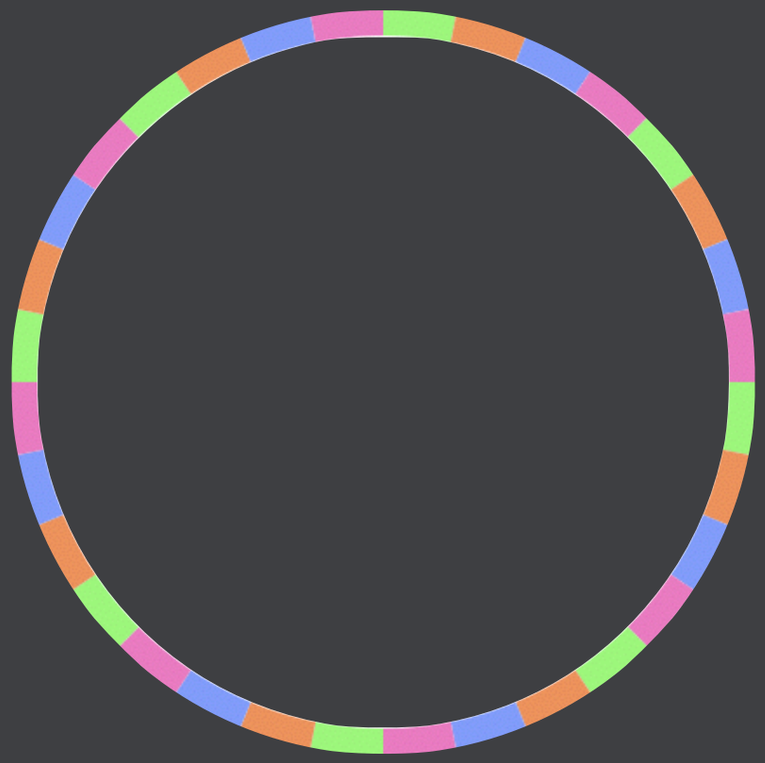


A Key/Value database

local storage

bitcask/leveldb

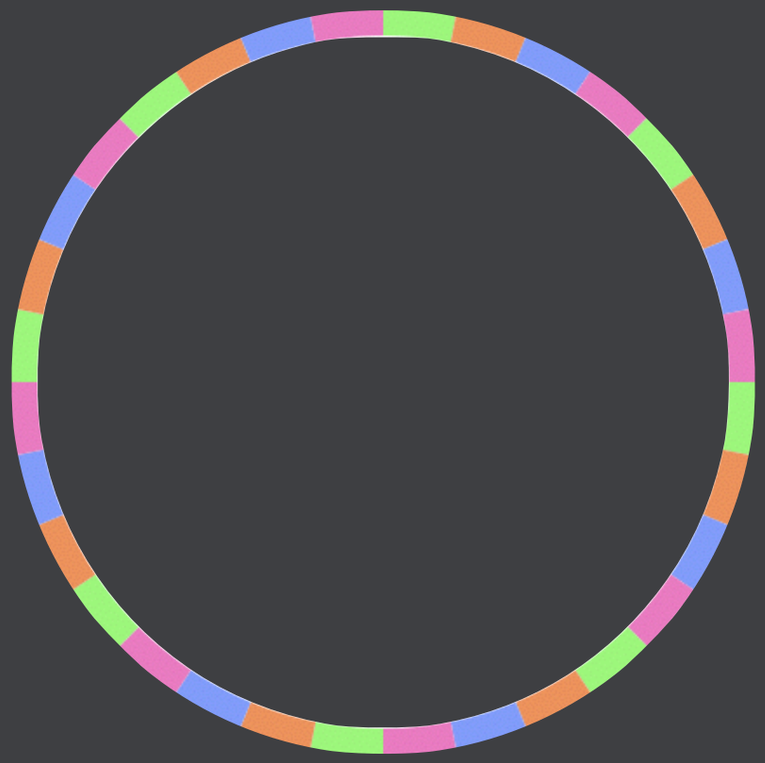
VNODES



routing table

mapping of
ranges/vnodes to nodes

ROUTING

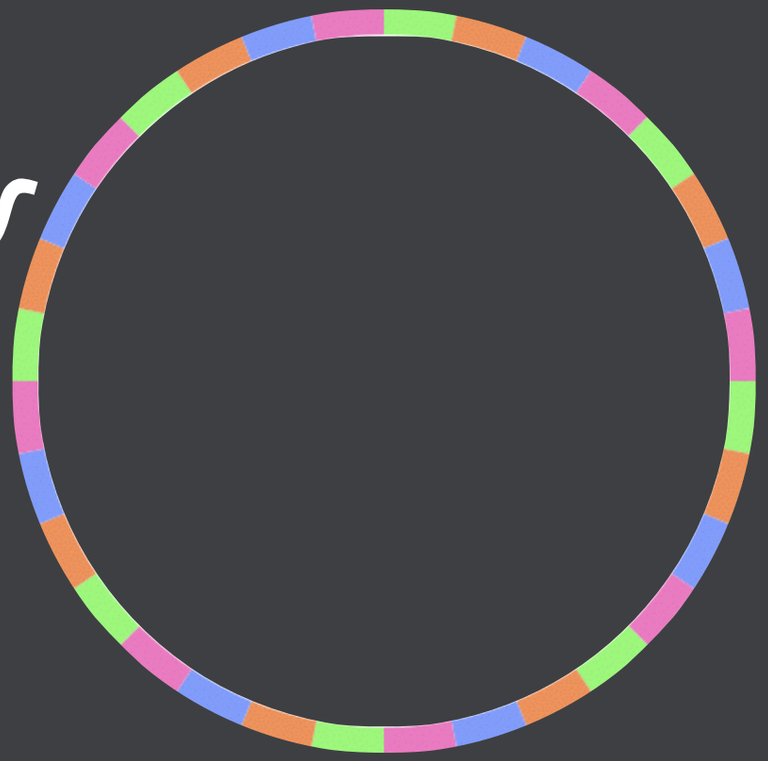


GOSSIP

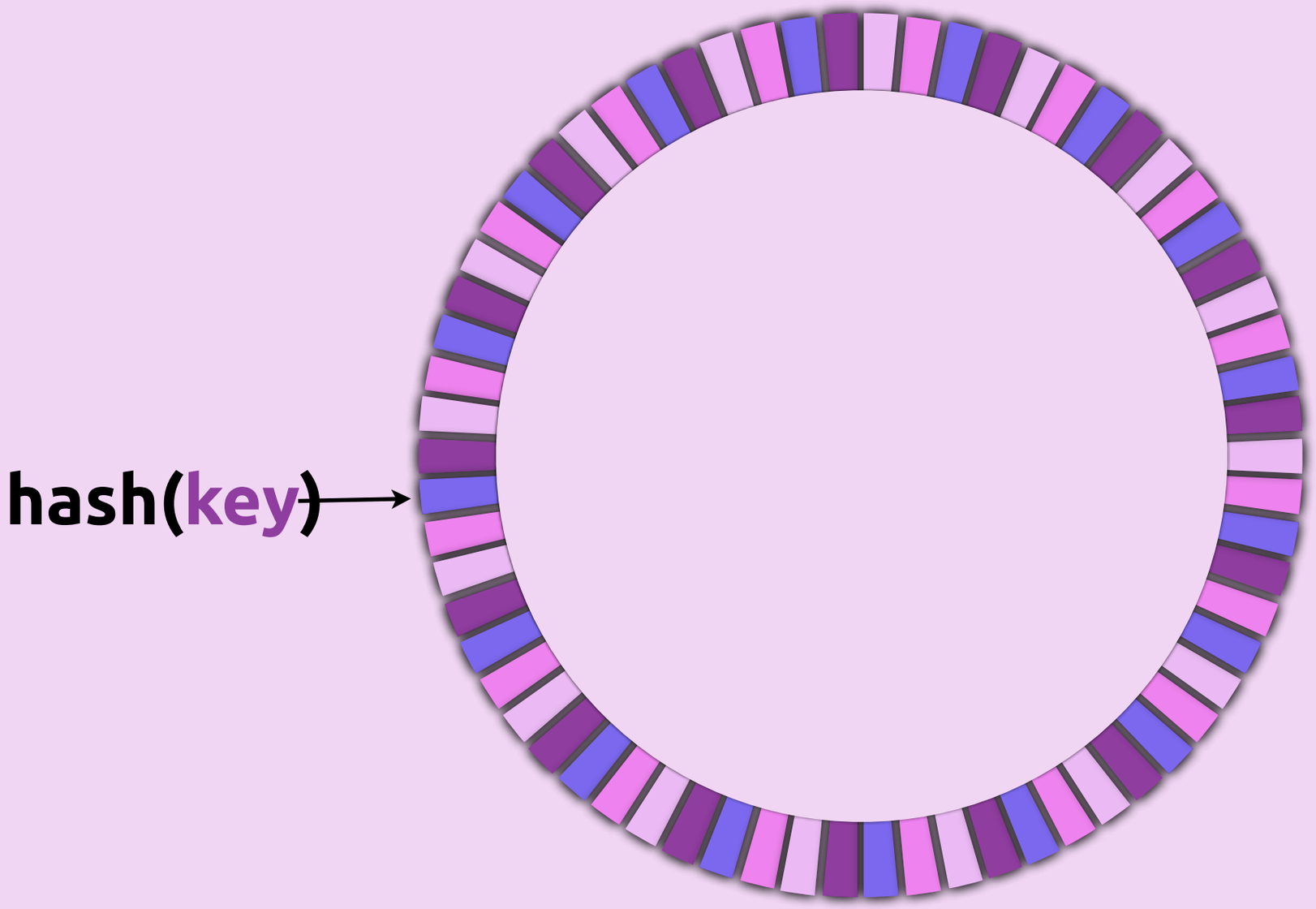
The ring is shared via epidemic gossip

GOSSIP

SHA1(key)



PRIMARY PREFERENCE LIST (preflist)



node 0

node 1

node 2

\+

node 3

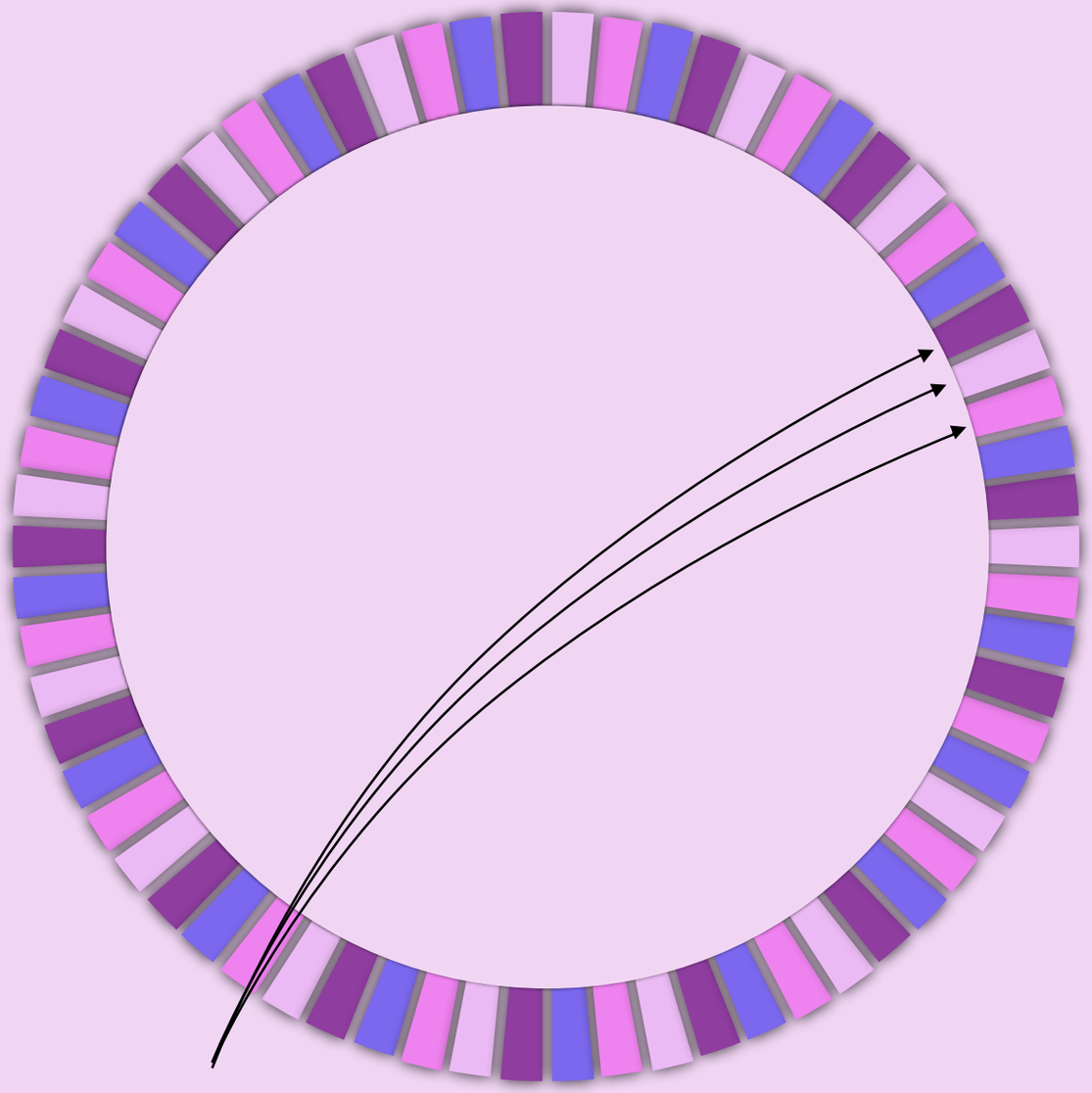
node 0

node 1

node 2

node 3

Replicas are stored N - 1 contiguous partitions



Replication

hash(“cities/london”)

Availability

Any non-failing node can respond to any request

--Gilbert & Lynch

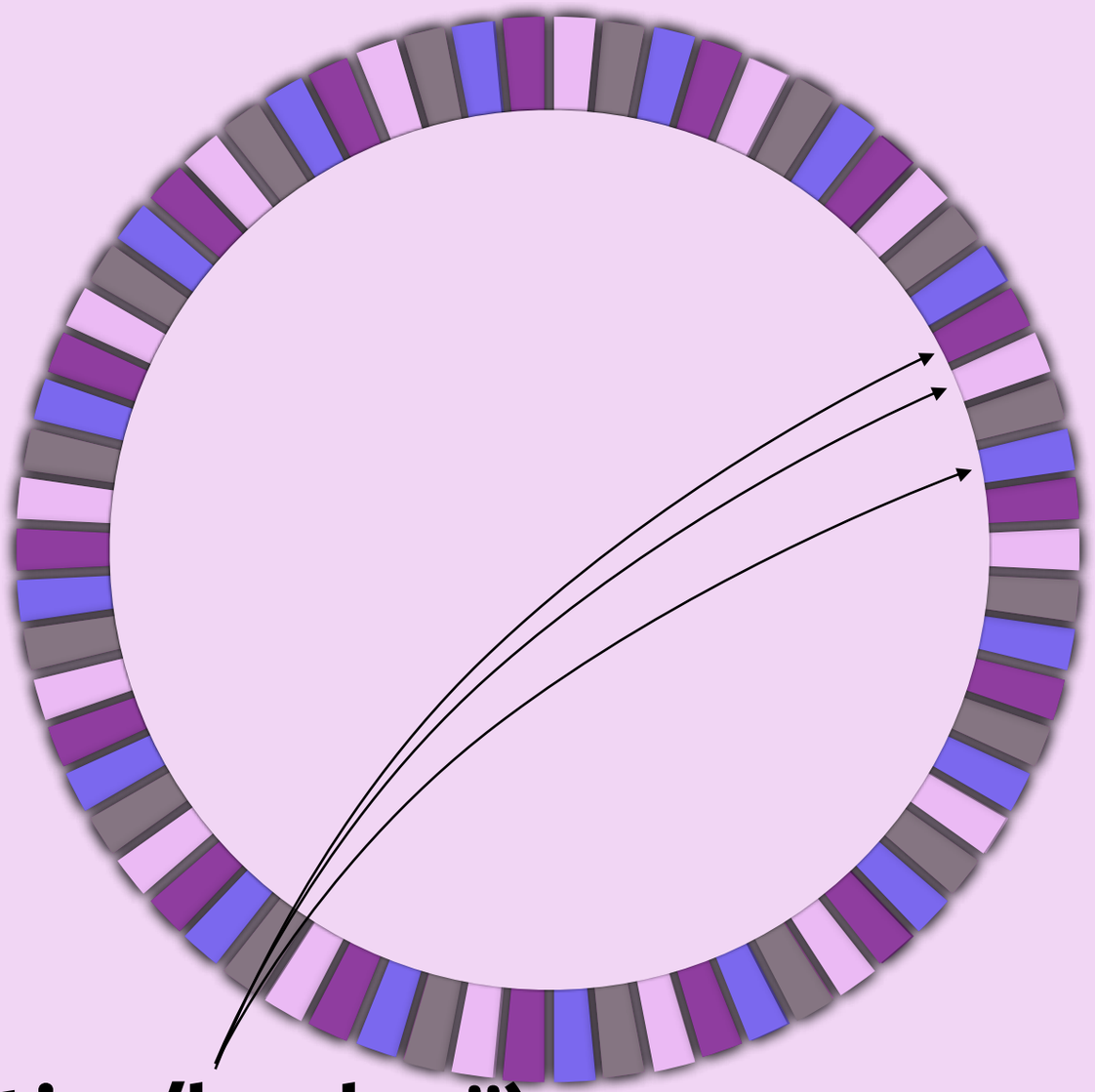
node 0

node 1

offline

node 3

Replicas are stored N - 1 contiguous partitions



Fault Tolerance

put(“cities/london”)

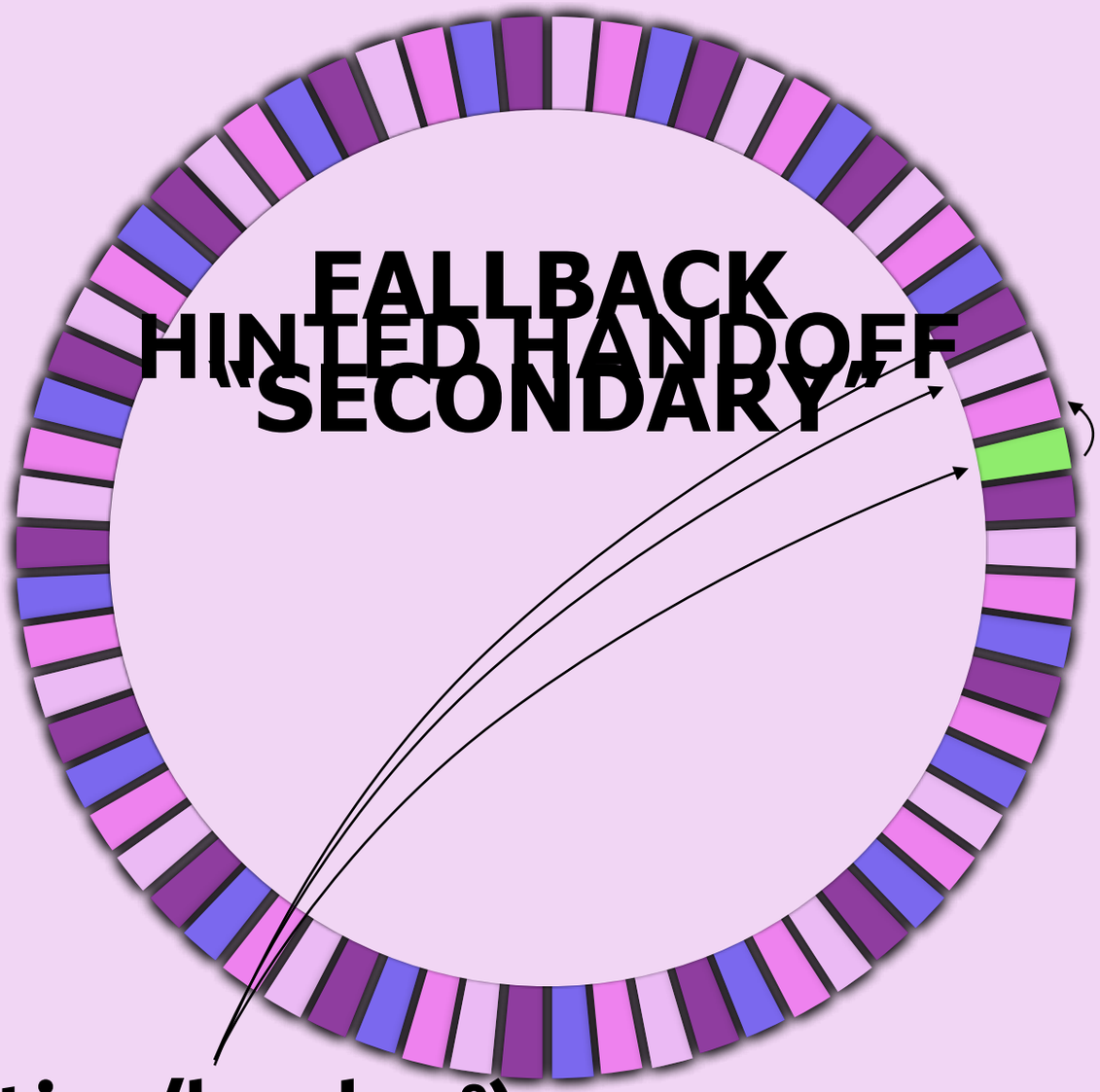
node 0

node 1

node 2

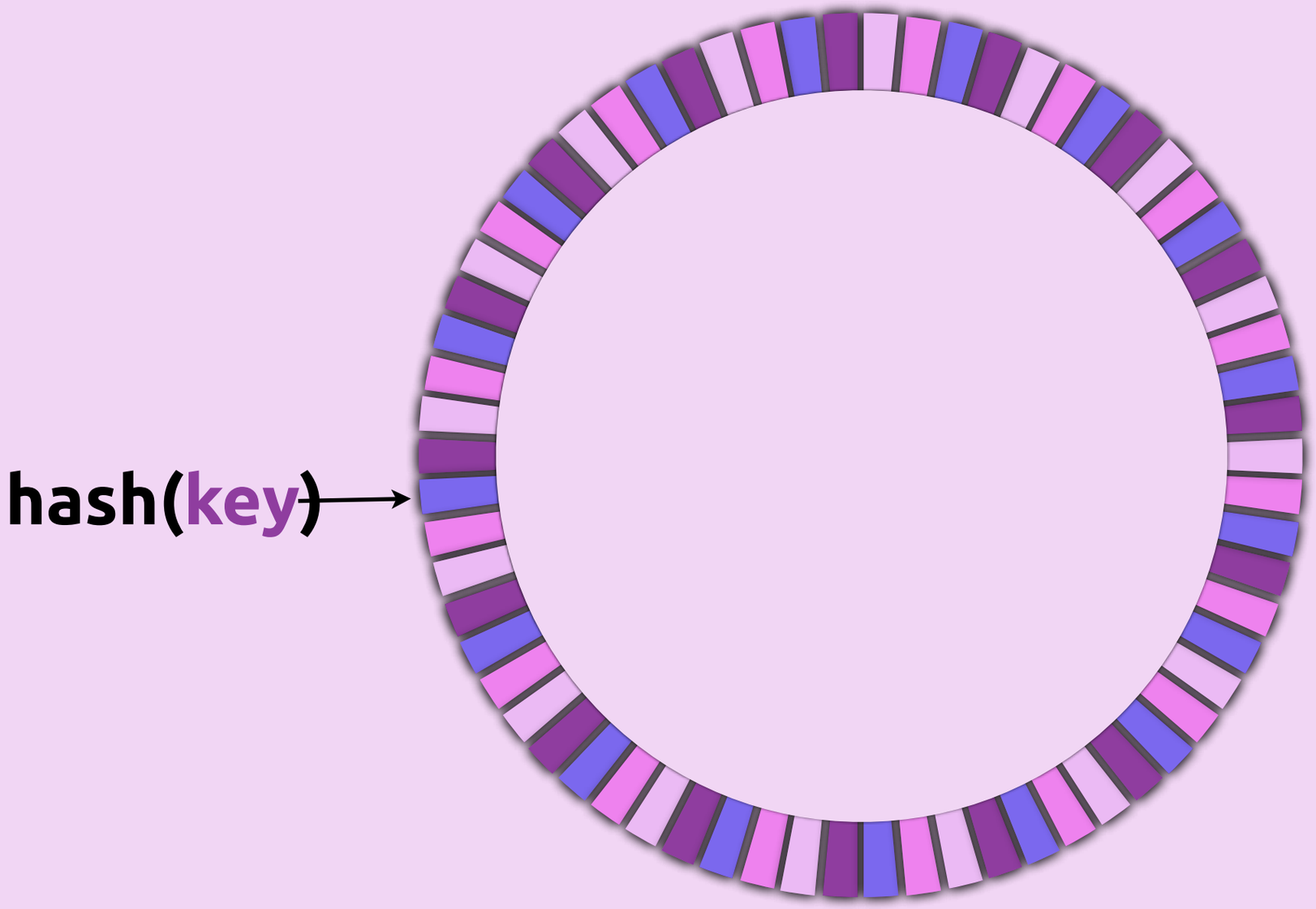
node 3

Replicas are stored N - 1 contiguous partitions



Fault Tolerance

put(⁸cities/london⁹)



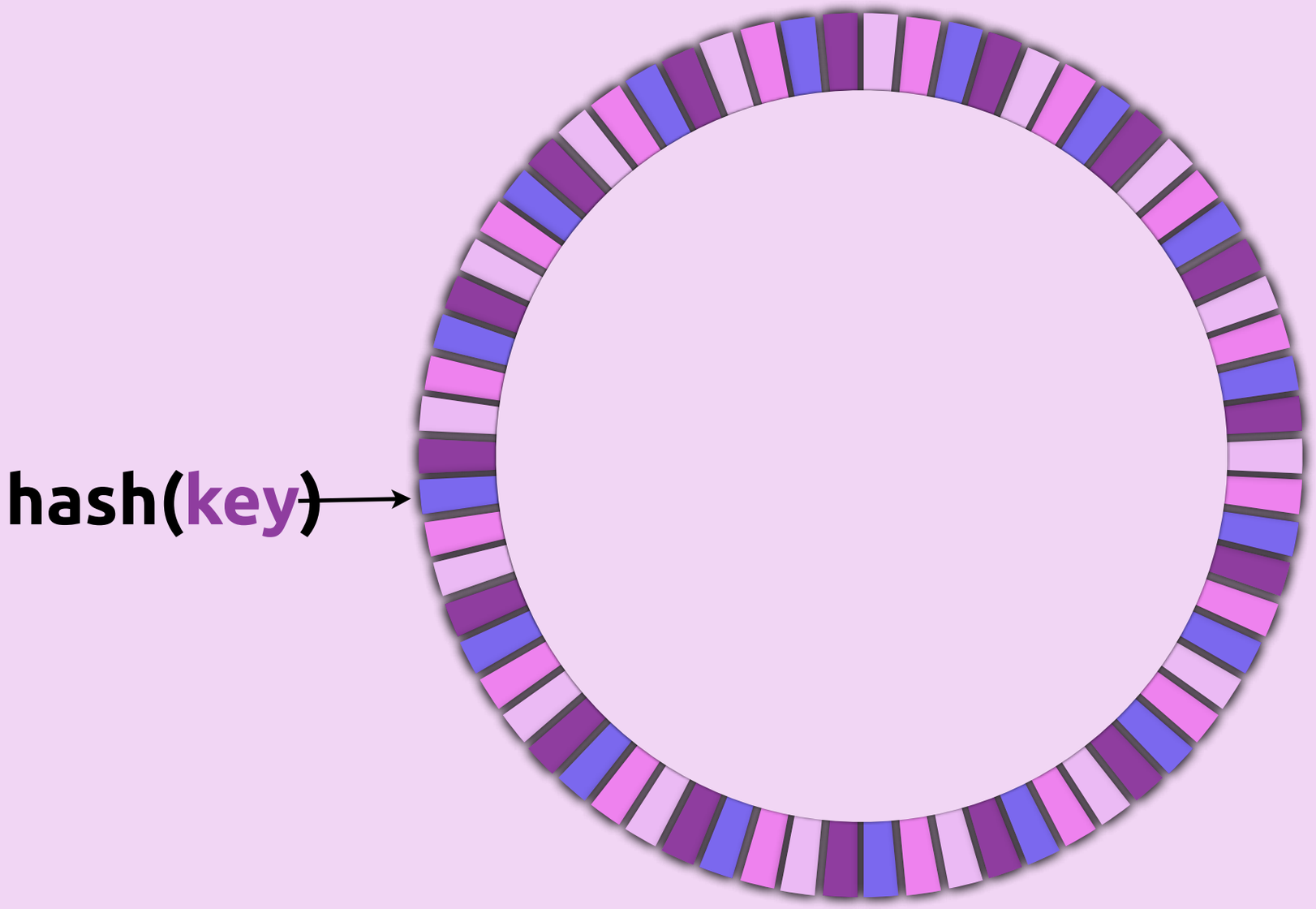
node 0

node 1

node 2

\+

node 3



- node 0
- node 1
- node 2
- node 3

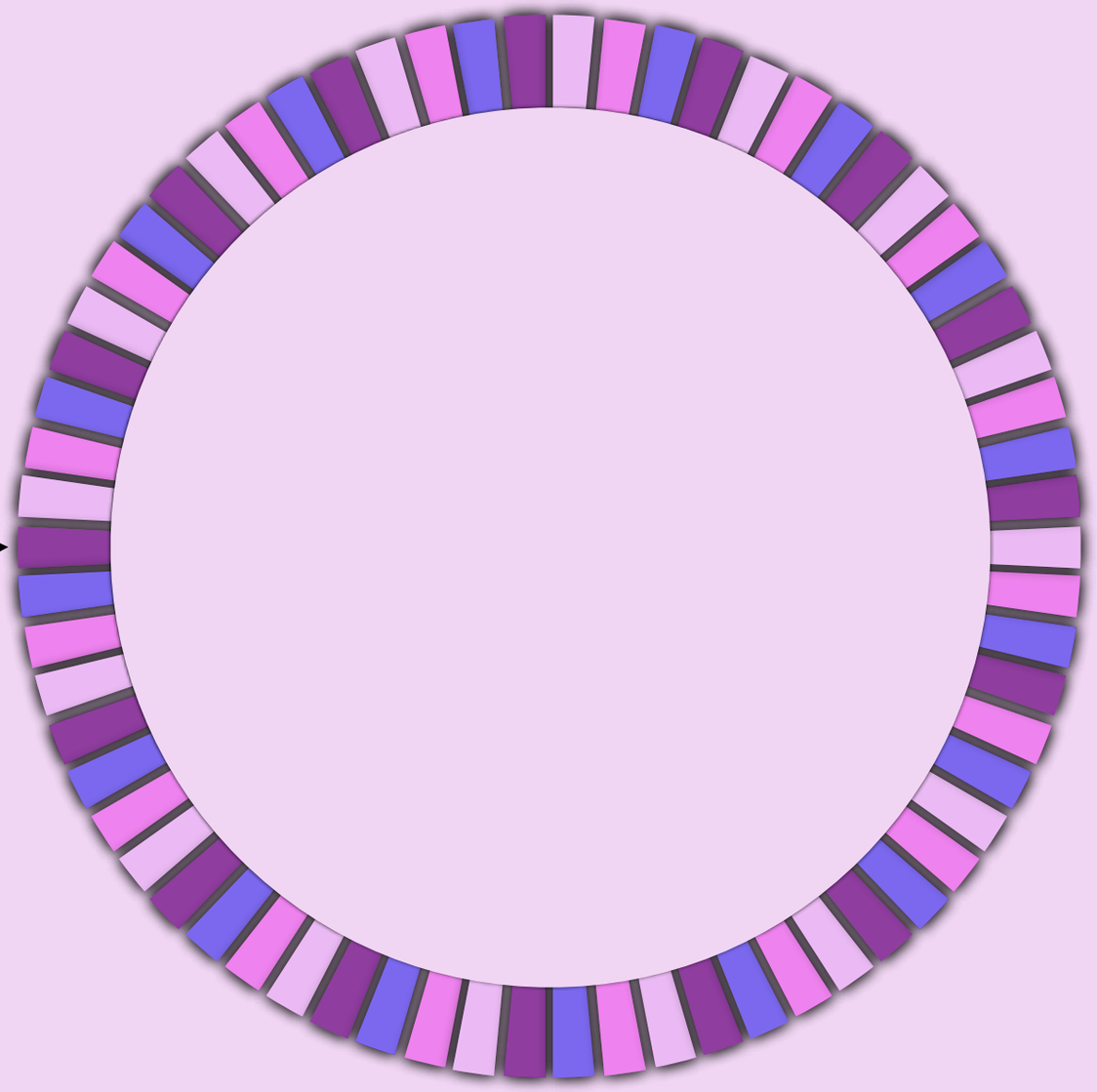
node 0

node 1

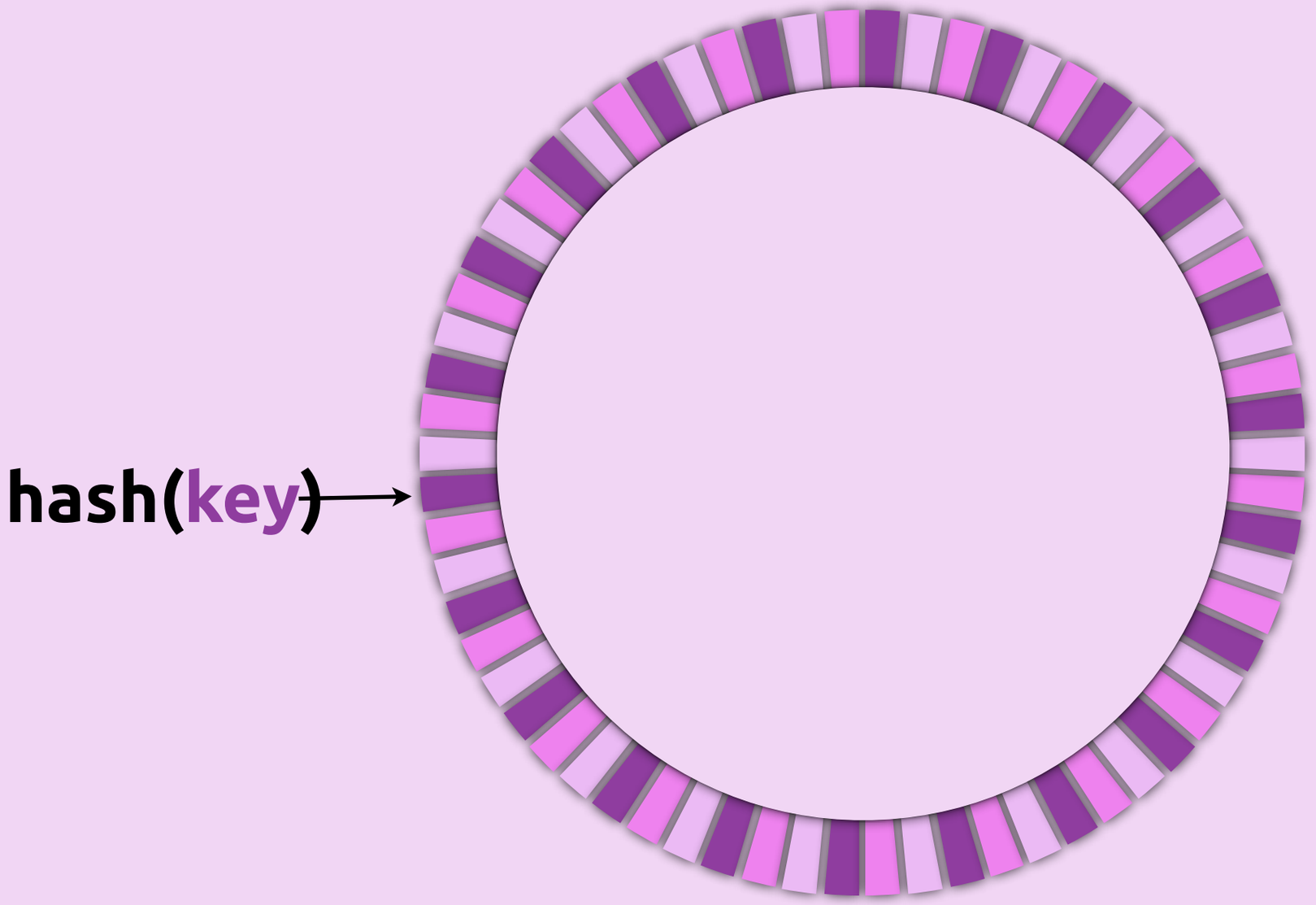
node 2

node 3

hash(key)



OWNERSHIP HANDOFF

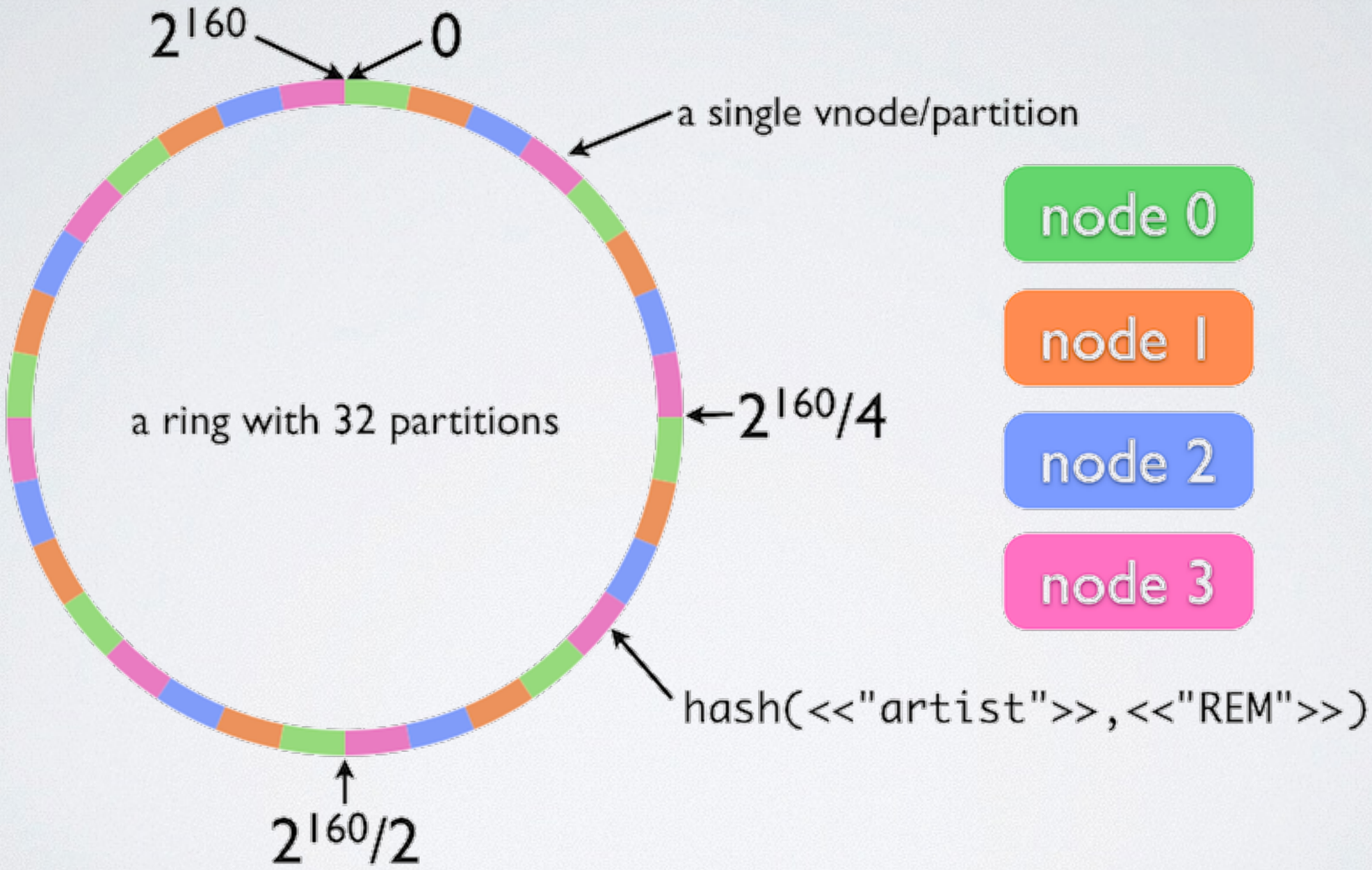


node 0

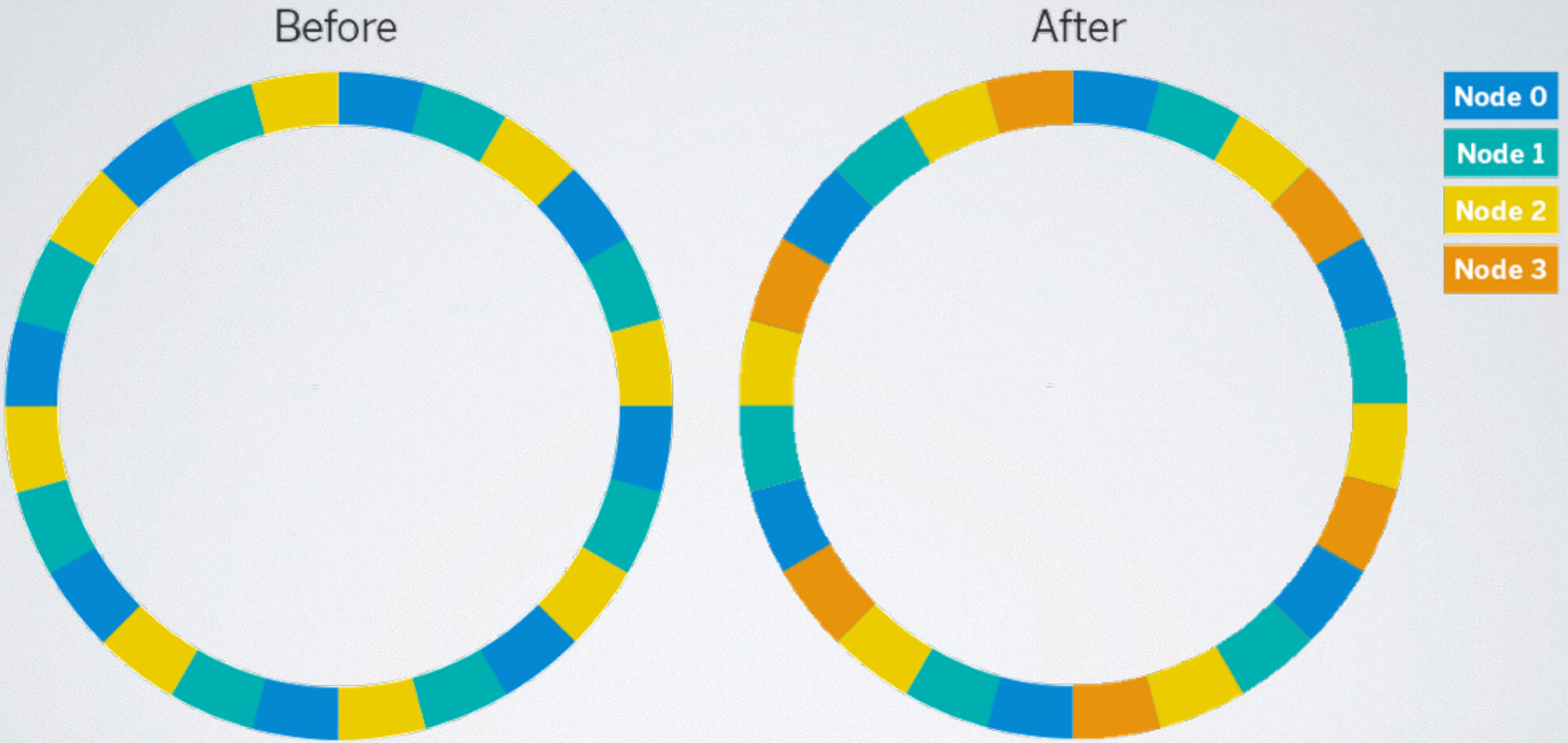
node 1

node 2

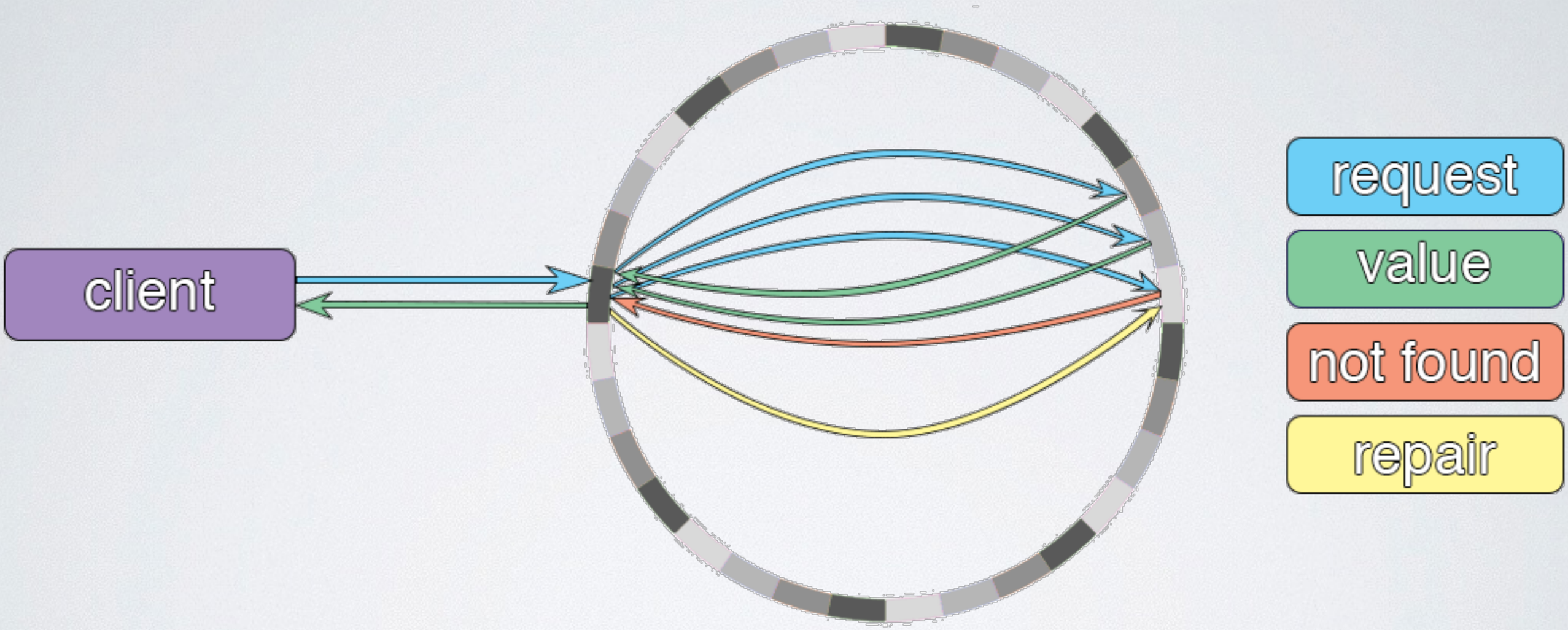
Consistent Hashing



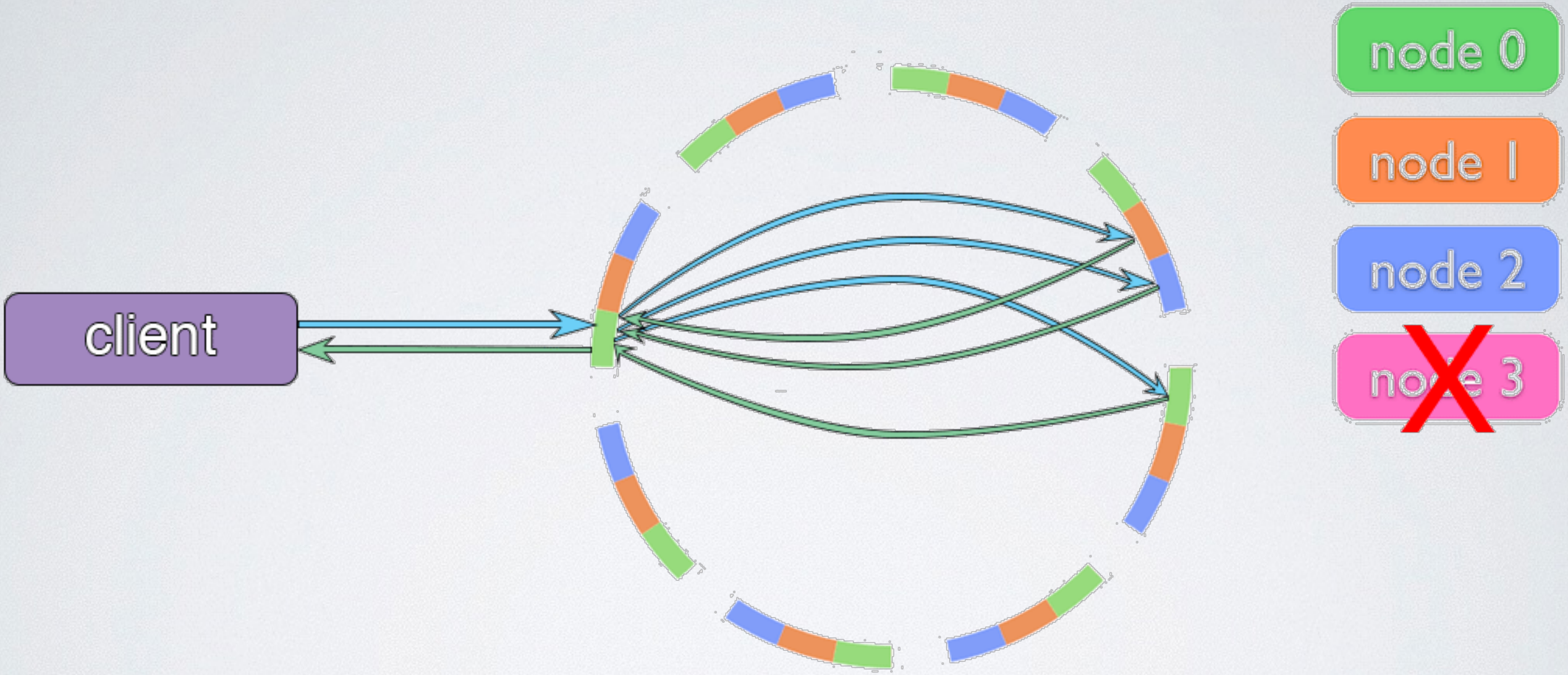
Balanced Scaling



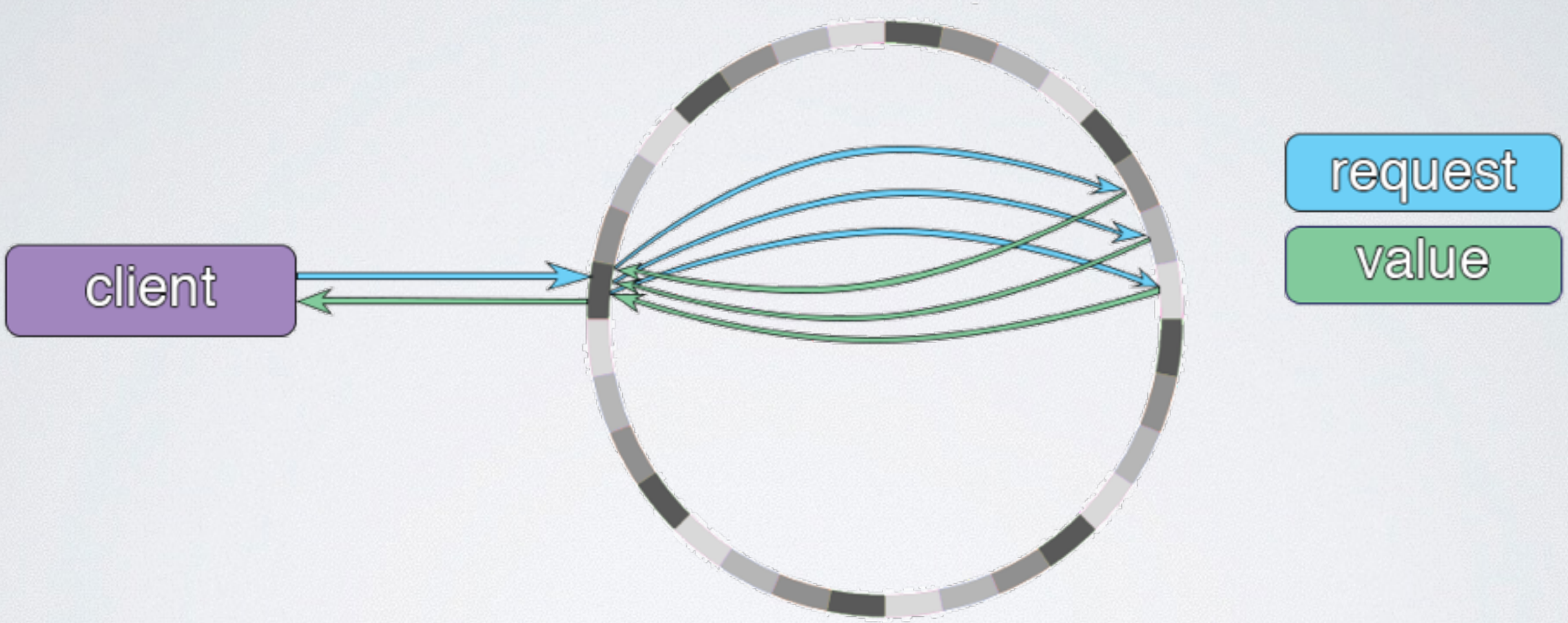
Read Repair



Replication with Sloppy Quorum



One-Hop Request Routing



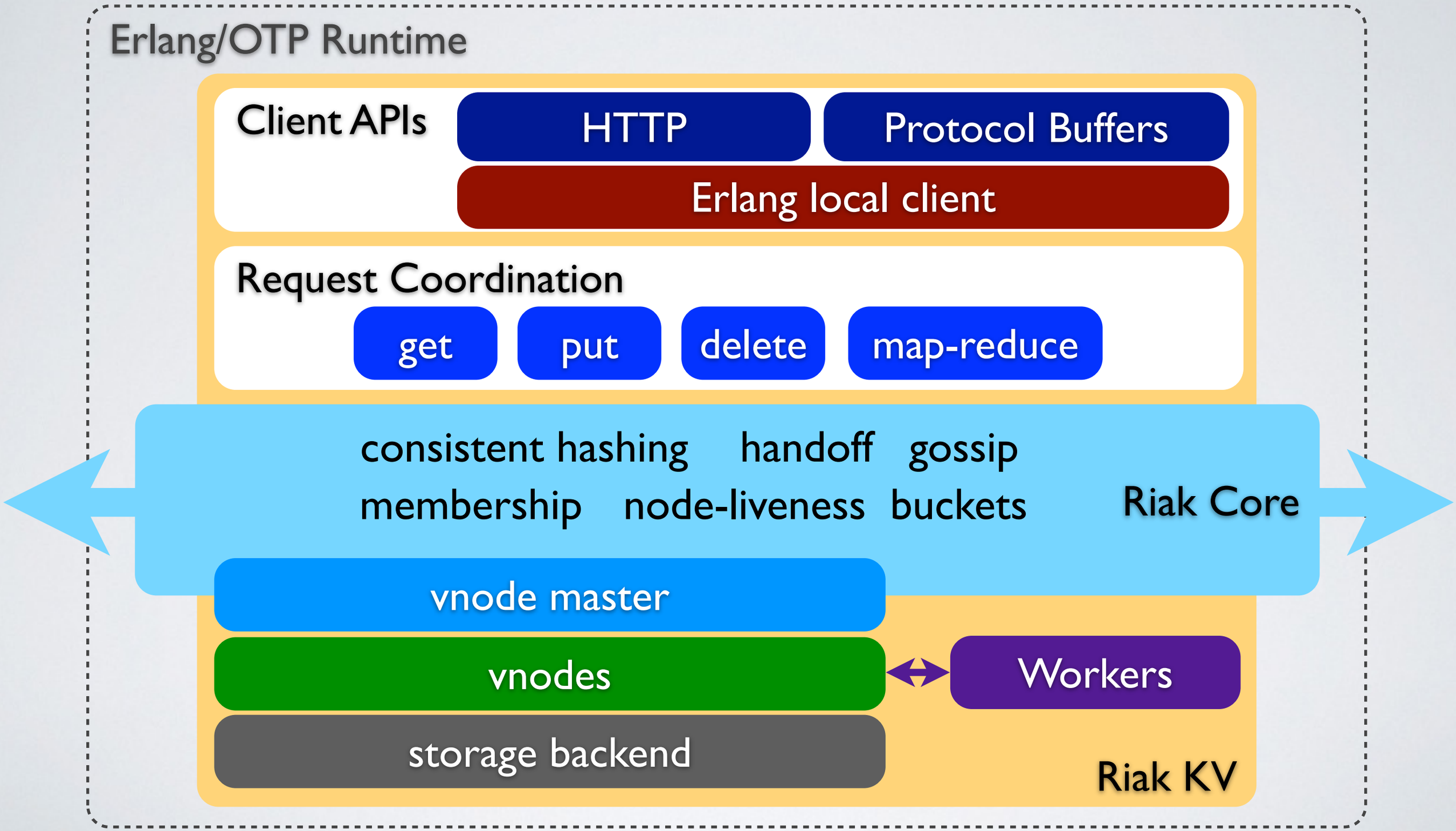
The Ring - Summary

- Membership - nodes in the cluster
- Ownership - claim of vnodes
- Routing - vnodes to nodes
- Vnodes - processes & databases
- Handoff - primaries/fallbacks, ownership change

The Ring - Summary

- Automatic failure detection (heartbeats)
- Automatic fallback data storage
- Automatic healing - hinted handoff
- Automatic ownership transfer - handoff

RIAK ARCHITECTURE



Eventual Consistency

Eventual consistency is a consistency model used in distributed computing that informally guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value.

--Wikipedia

Riak Overview

N, R, W, PR, PW etc

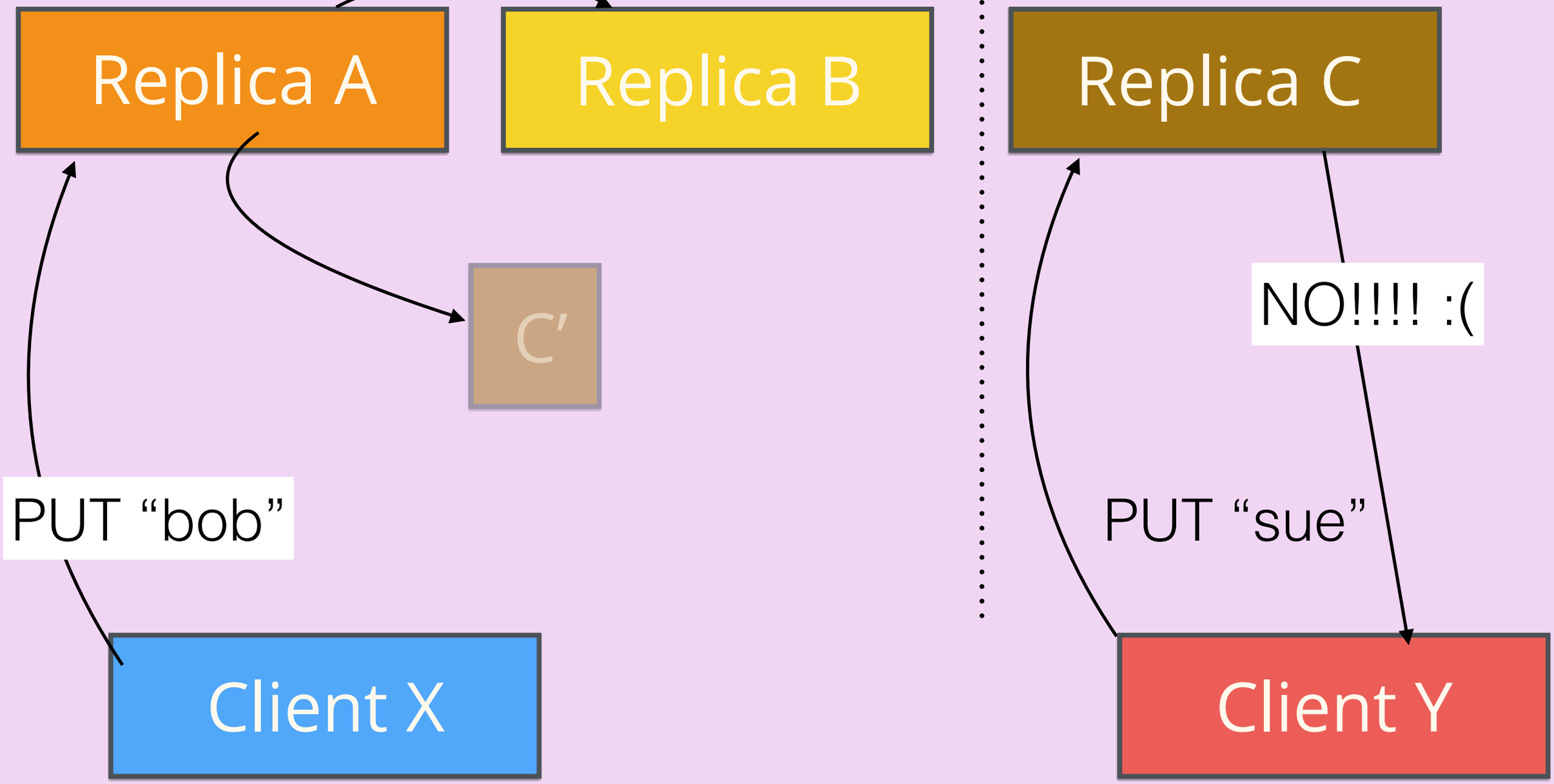
REQUEST QUORUMS

- Every request contacts all replicas of key
  - N - number of replicas (default 3)
  - R - read quorum
  - W - write quorum
- Quorum:

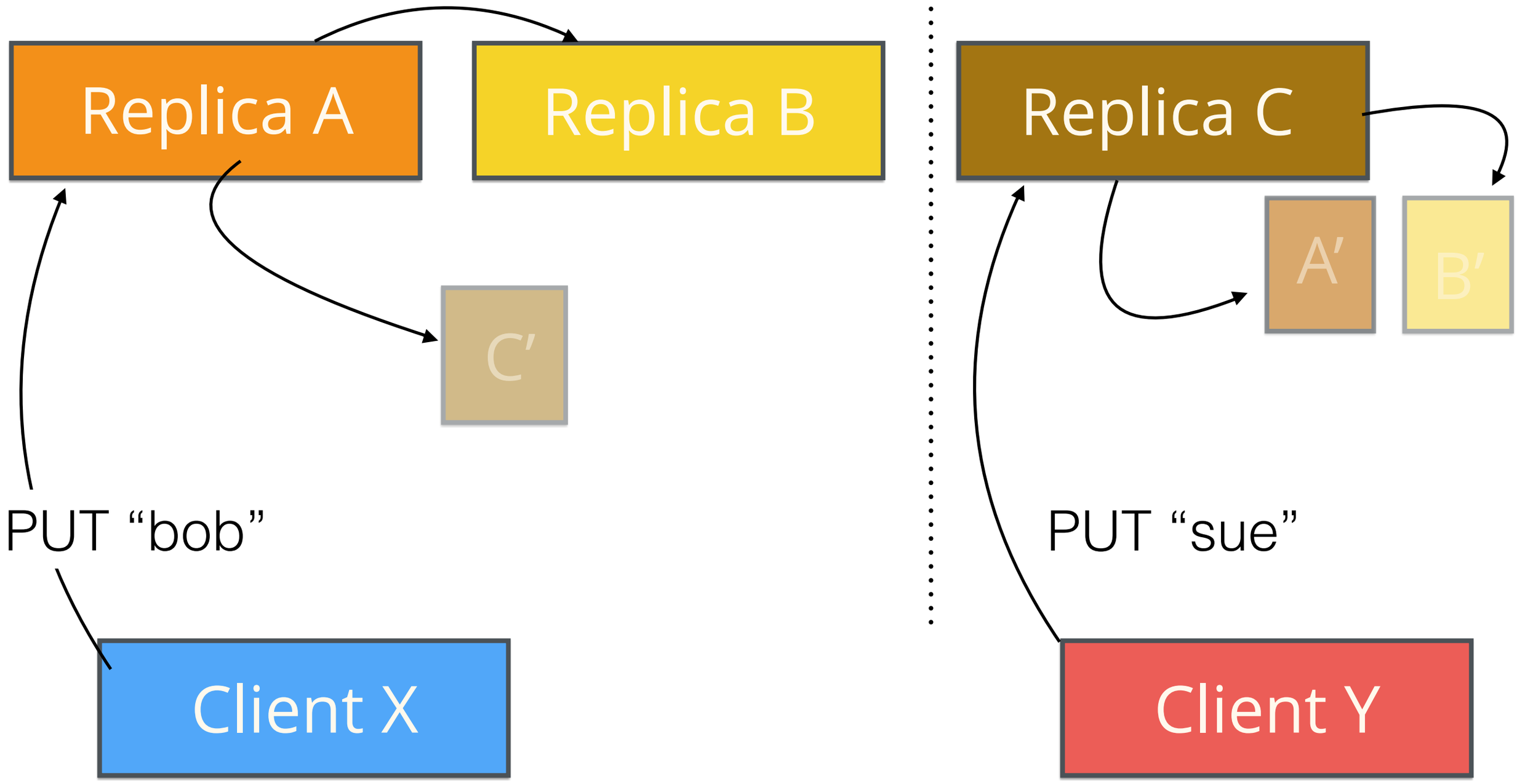
  The quantity of replicas that must respond to a read or write request before it is considered successful. (default 2 - Calculated as: **floor(n_val / 2) + 1**)

Quora: For Consistency

- How many Replicas must respond: 1, quorum, all?
- Strict Quorum: Only Primaries are contacted
- Sloppy Quorum: Fallbacks are contacted
- Fast Writes? W=1
- Fast Reads? R=1
- Read Your Own Writes? PW+PR>N

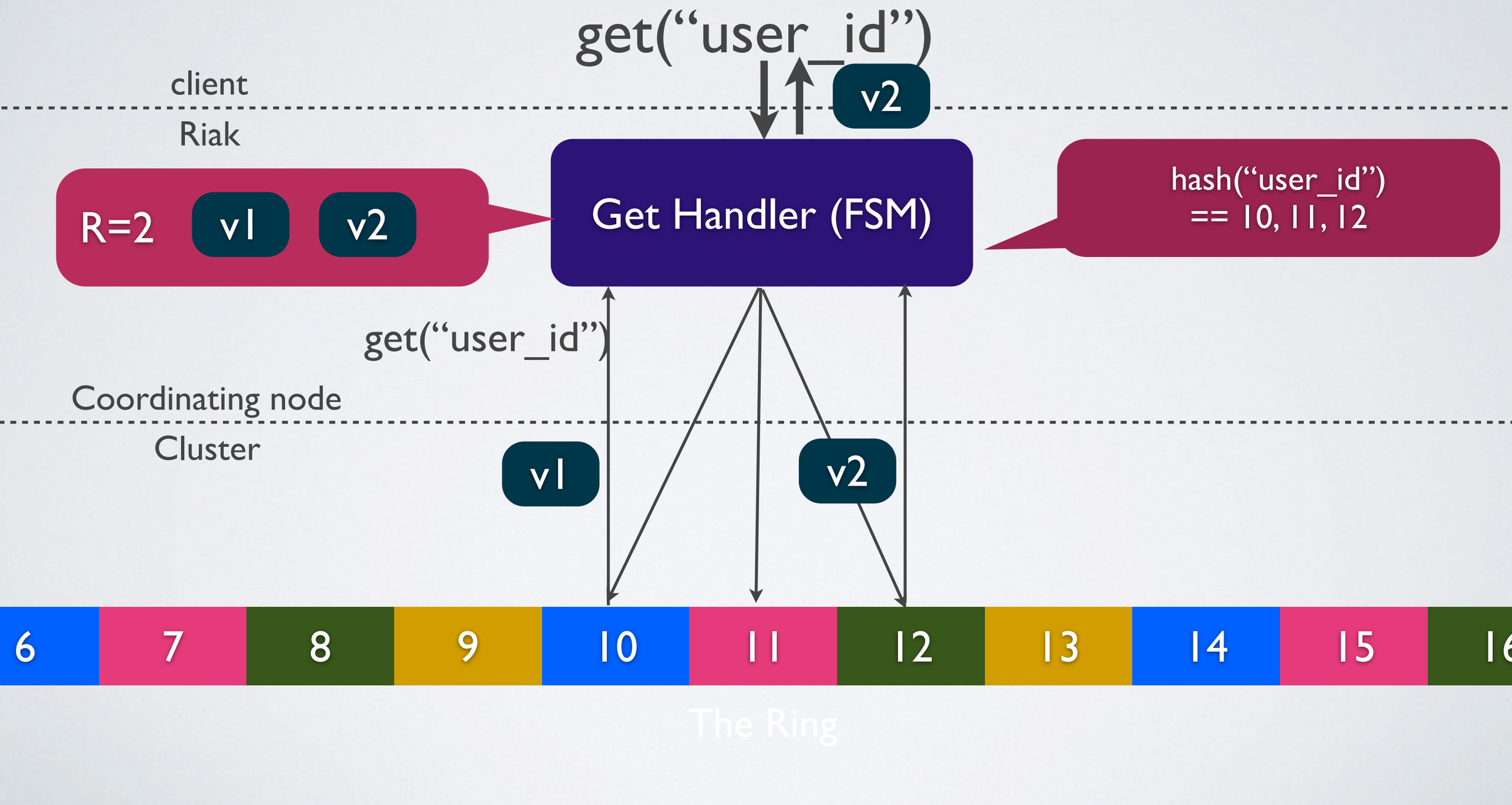

Replica A
Replica B
Replica C
C′
NO!!!! :(
PUT “bob”
PUT “sue”
Client X
Client Y

Strict

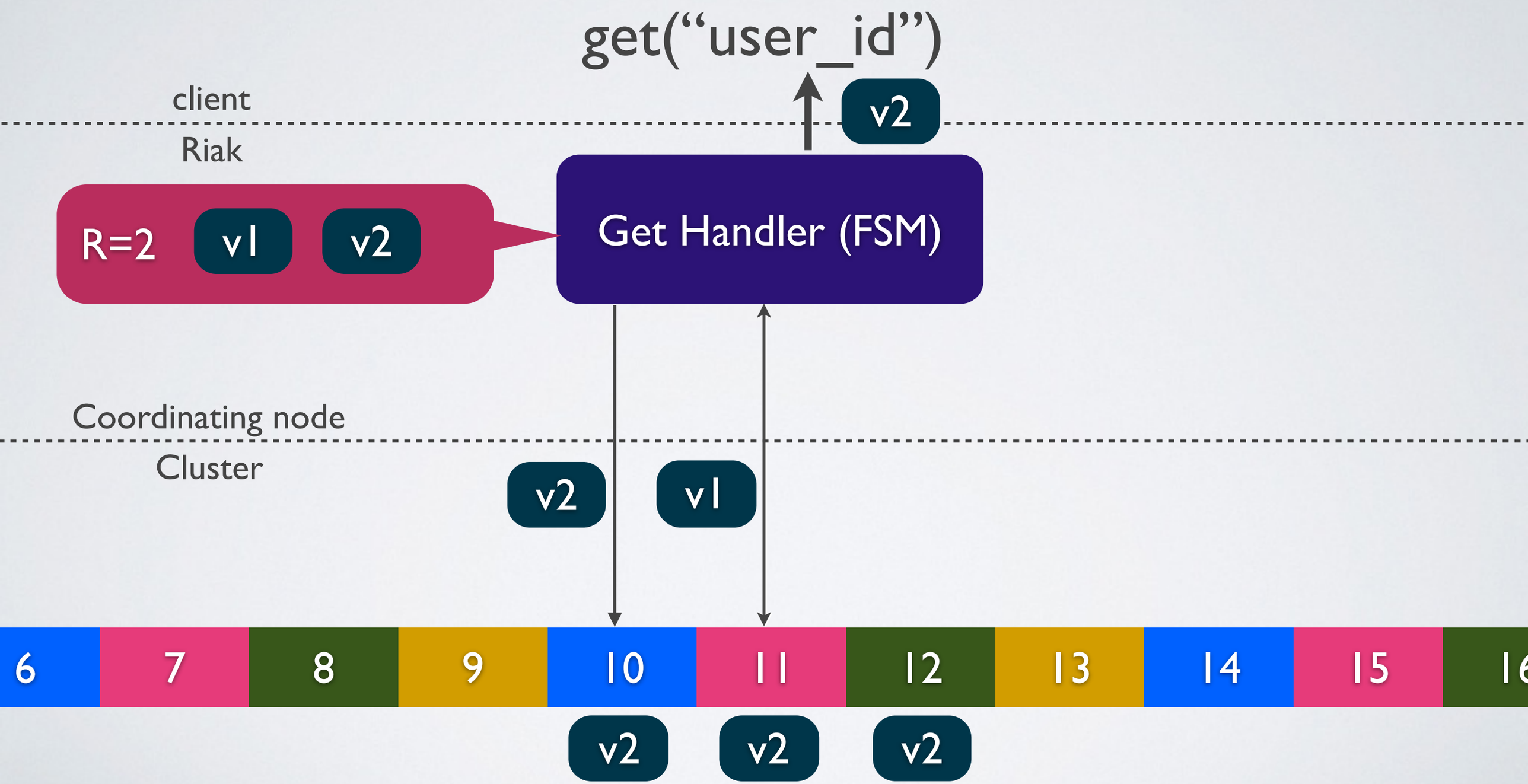

Replica A
Replica B
C′
PUT “bob”
Client X
Replica C
A′
B′
PUT “sue”
Client Y

Sloppy

ANATOMY OF A REQUEST

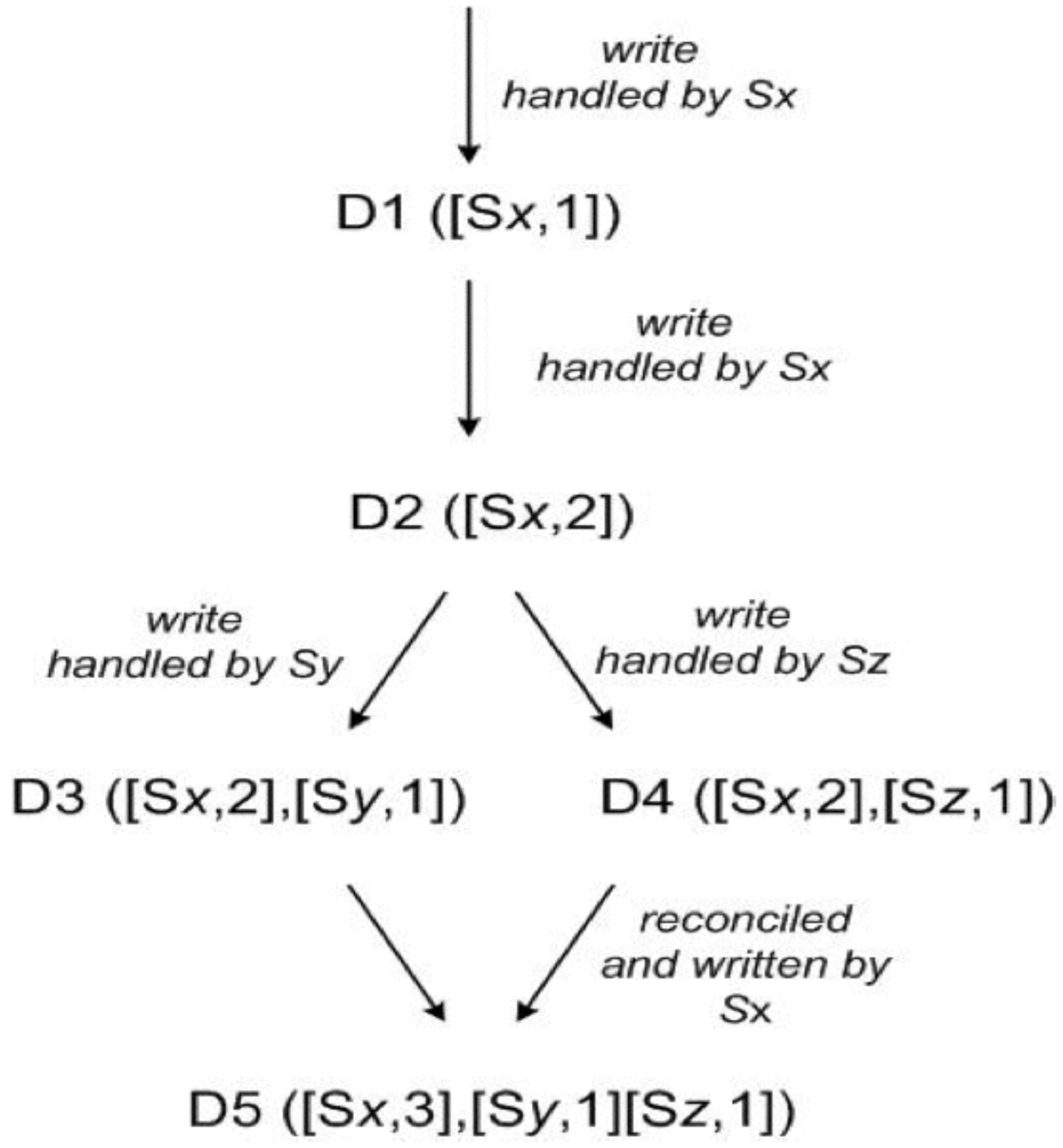


READ REPAIR


get("user_id")
client
v2
Riak
R=2
v1
v2
Get Handler (FSM)
Coordinating node
Cluster
v2
v1
6
7
8
9
10
11
12
13
14
15
16
v2
v2
v2

Version Vectors - Logical Clocks

- Happens before or causal relationship
- vector of pairs {Actor, Counter}
- Each Actor updates own entry only
- Each Object has own Version Vector
- Concurrent Updates Detected


write
handled by Sx
D1 ([Sx,1])
write
handled by Sx
D2 ([Sx,2])
write
handled by Sy
write
handled by Sz
D3 ([Sx,2],[Sy,1])
D4 ([Sx,2],[Sz,1])
reconciled
and written by
Sx
D5 ([Sx,3],[Sy,1][Sz,1])

Summary

- Distributed key Value
- Homogenous nodes
- Ring for membership, routing, ownership
- vnodes for datamanagement
- FSMs for read/write logic
- Always available for writes

Trade Off

CAP

http://aphyr.com/posts/288-the-network-is-reliable

C A

C

A

C

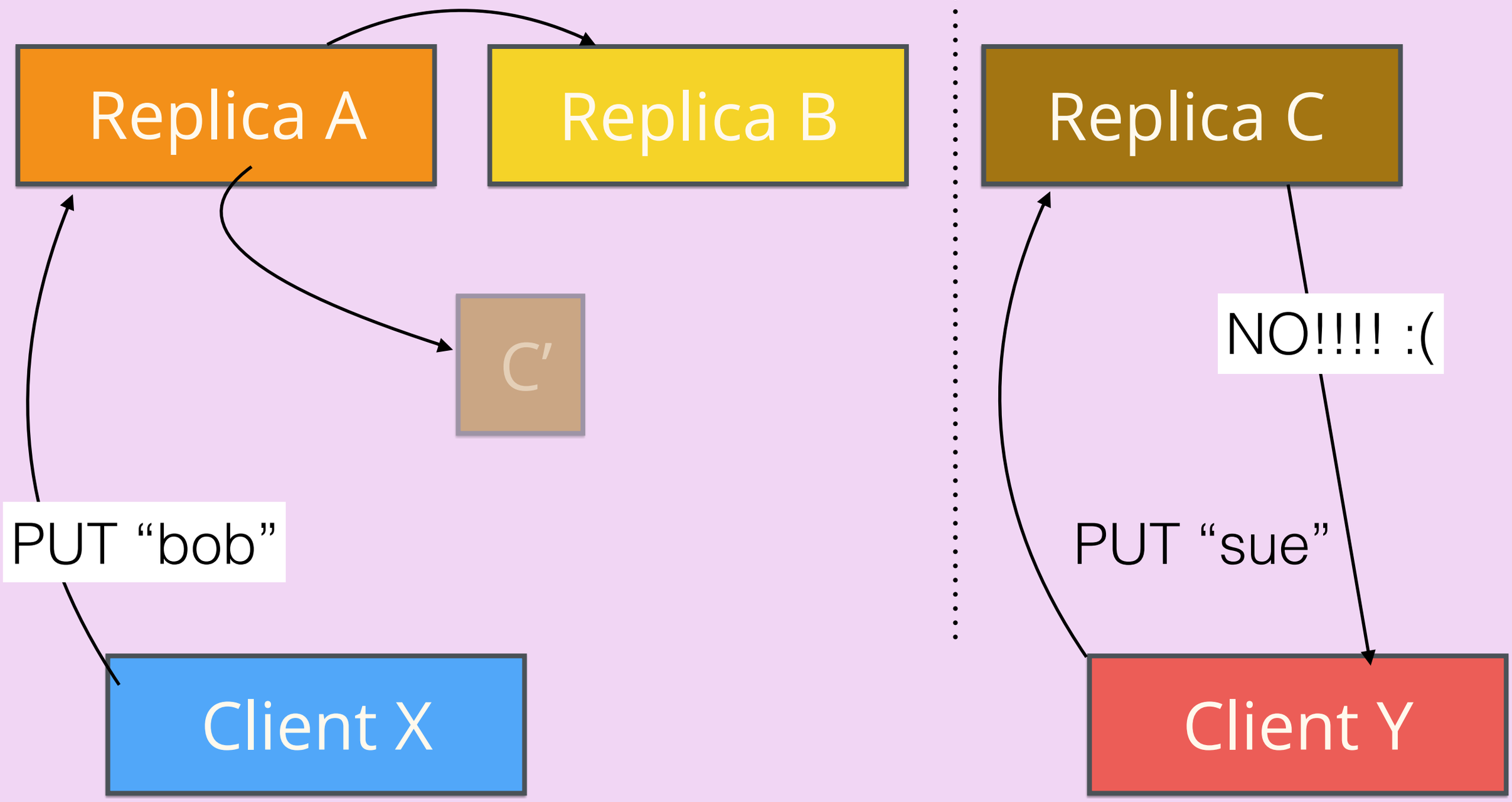
A

C

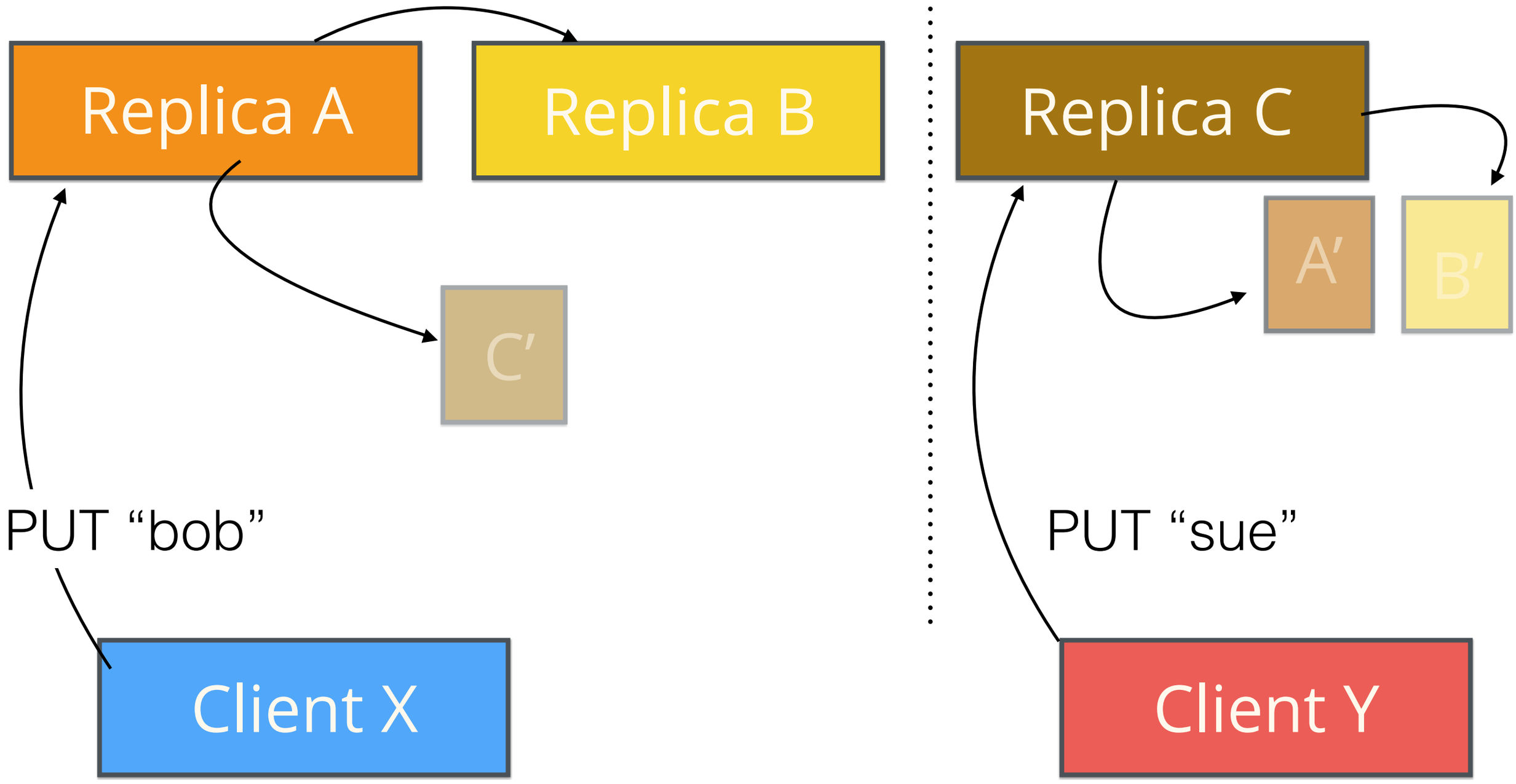
A

PEL

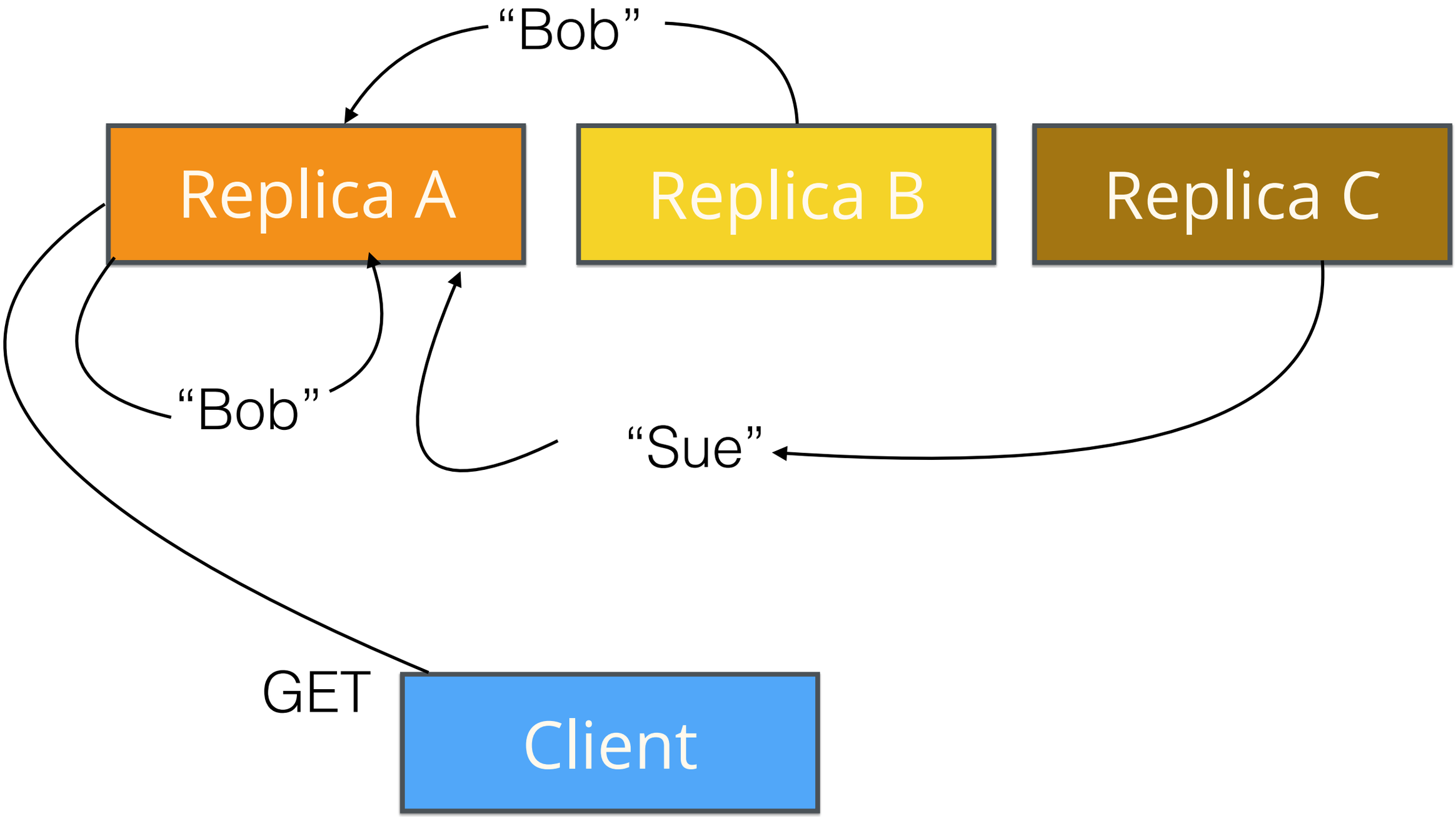
Conflict!


Replica A
Replica B
Replica C
C′
NO!!!! :(
PUT “bob”
PUT “sue”
Client X
Client Y

CP


Replica A
Replica B
C′
PUT “bob”
Client X
Replica C
A′
B′
PUT “sue”
Client Y

AP



Conflict!

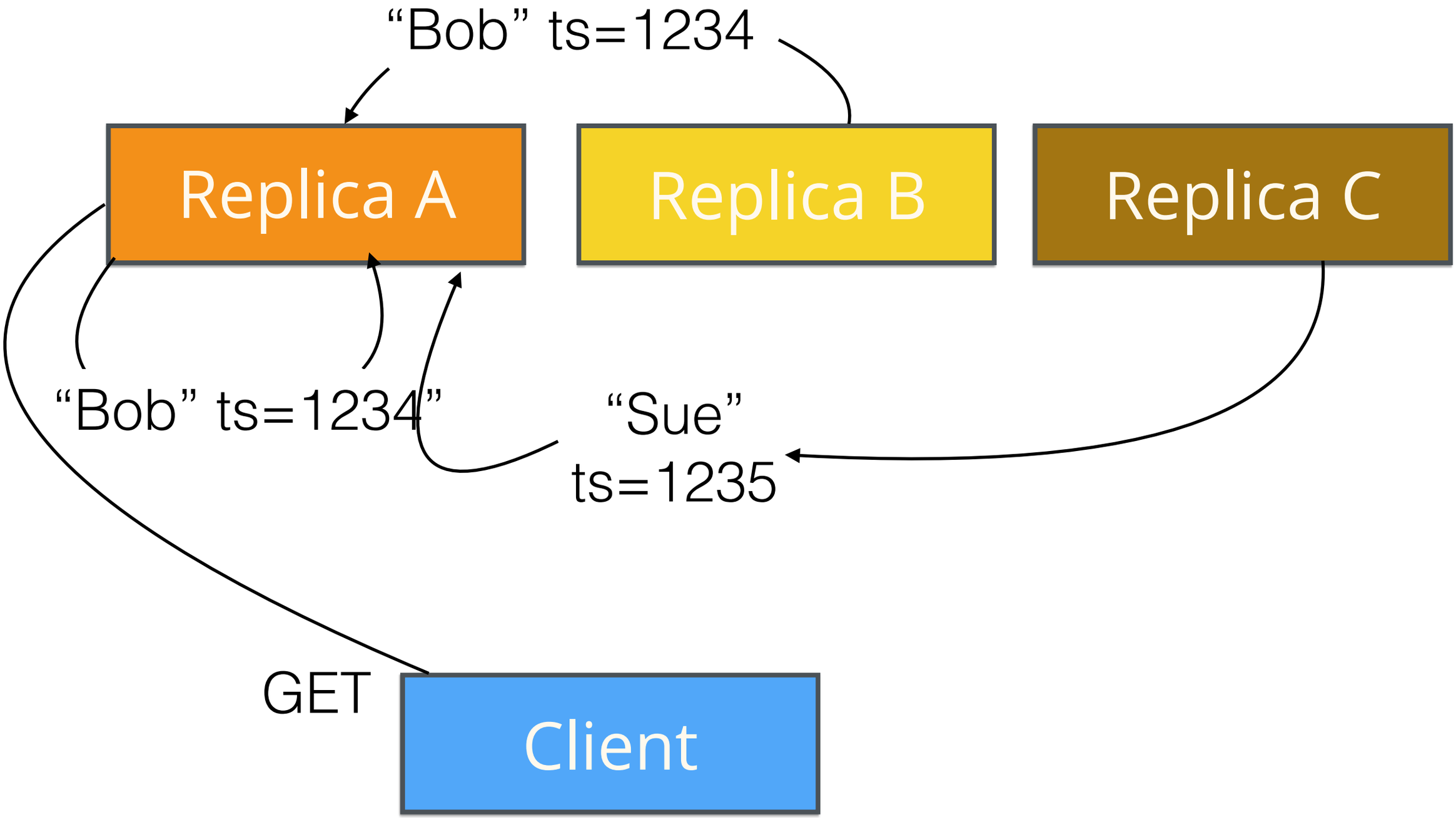
Eventual Consistency

Eventual consistency is a consistency model used in distributed computing that informally guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value.

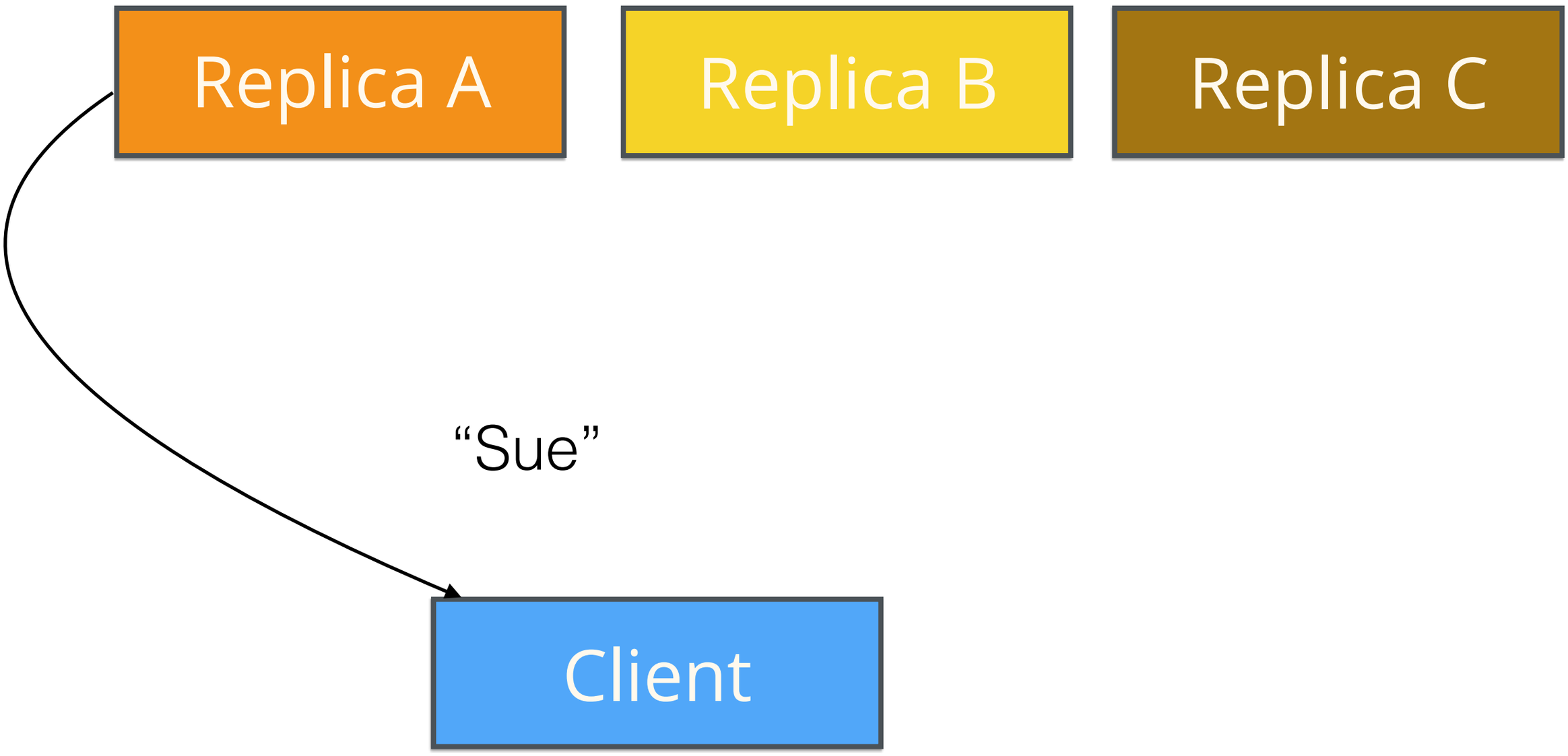
--Wikipedia

LAST UPDATED VALUE?

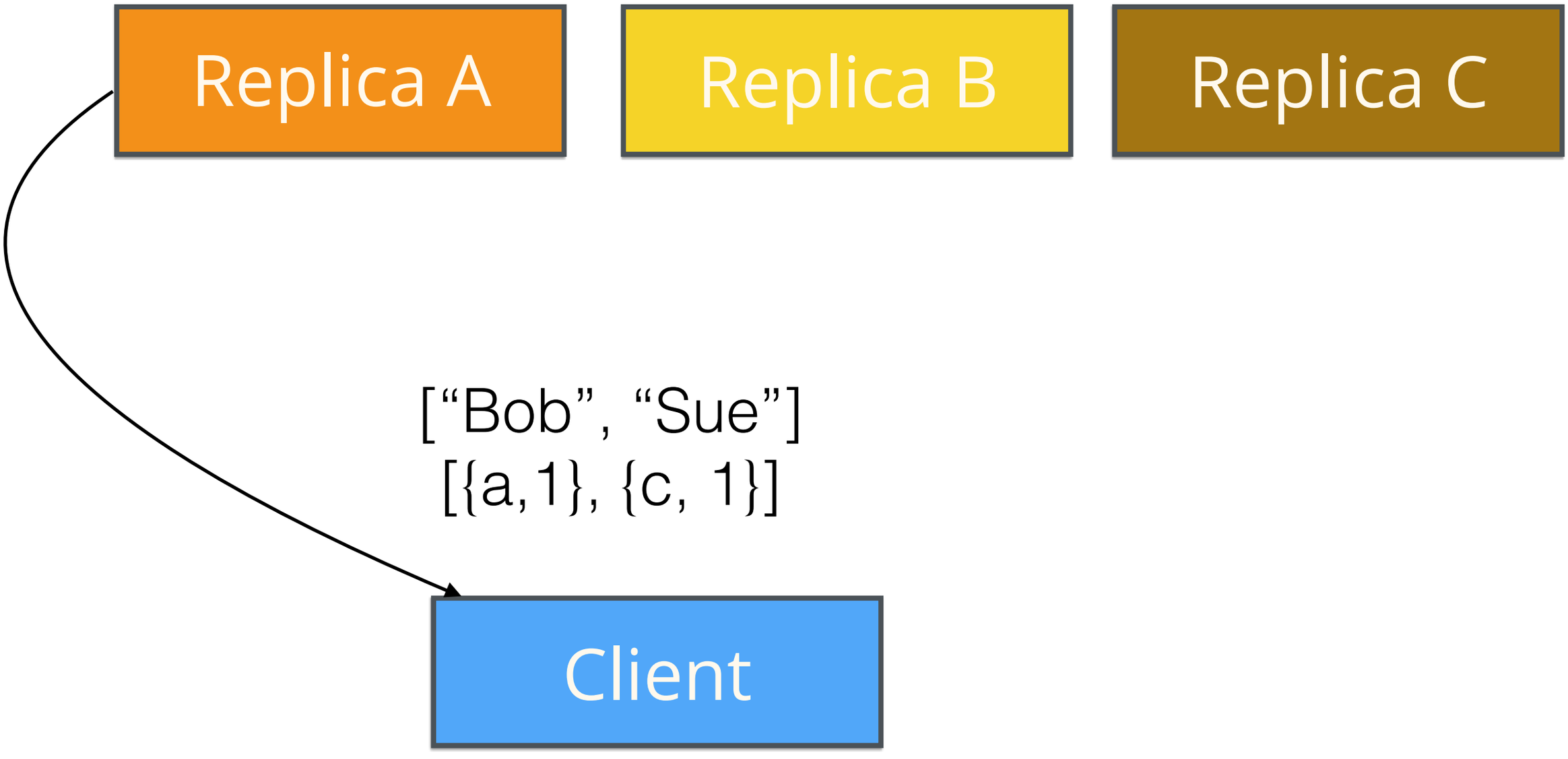
- Last by time?
- What about concurrent operations?
- What is the “last updated value?”



Last Write Wins!



Last Write Wins

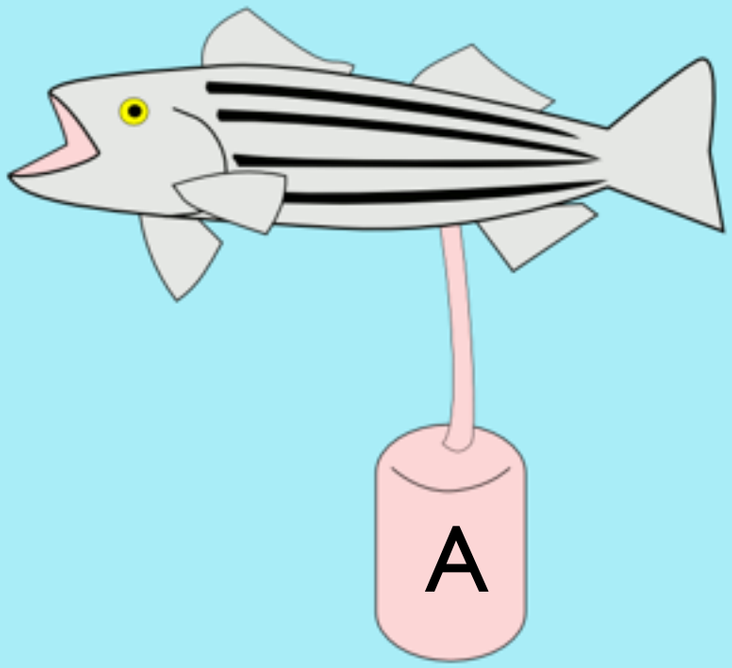


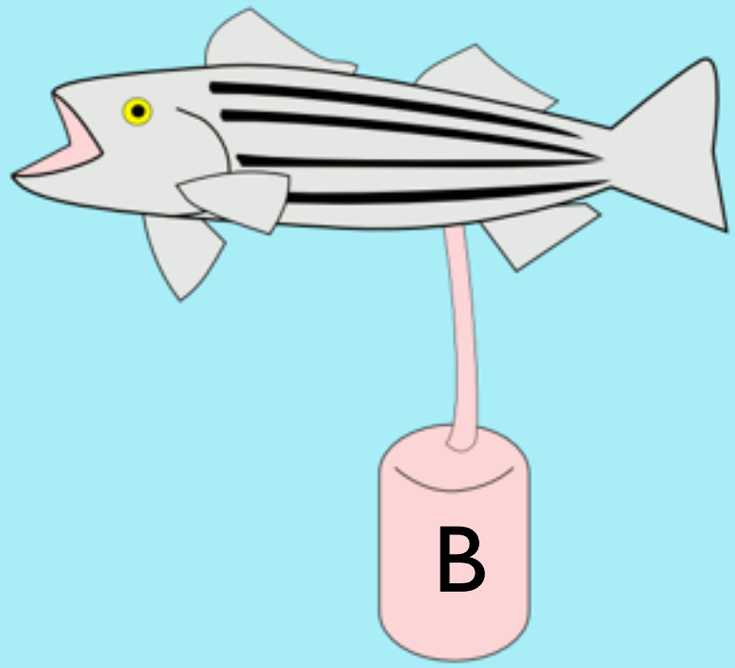
Multi-Value

Semantic Resolution

Dynamo

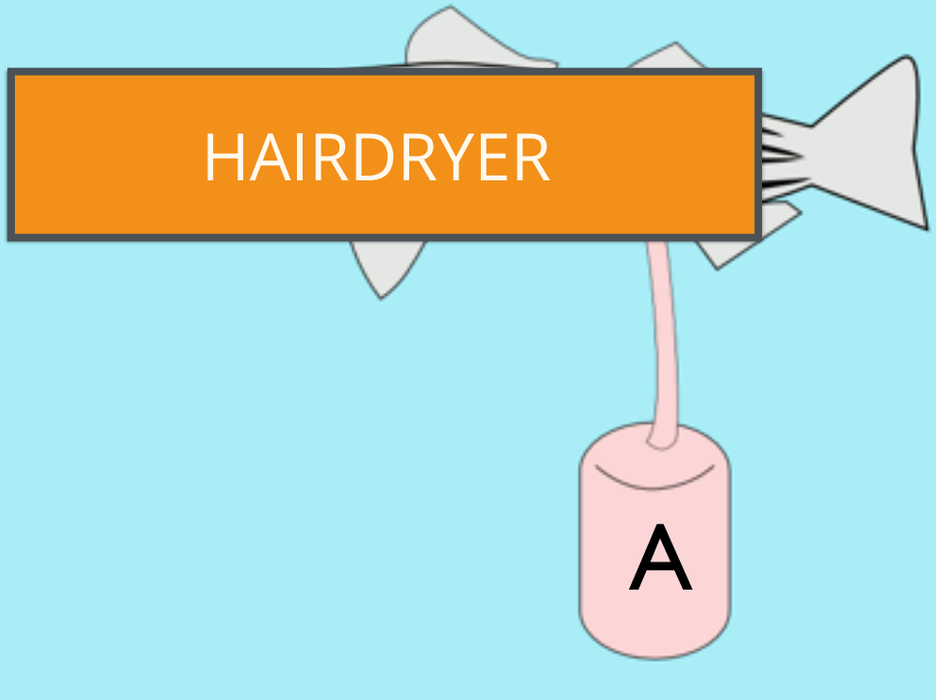
The Shopping Cart

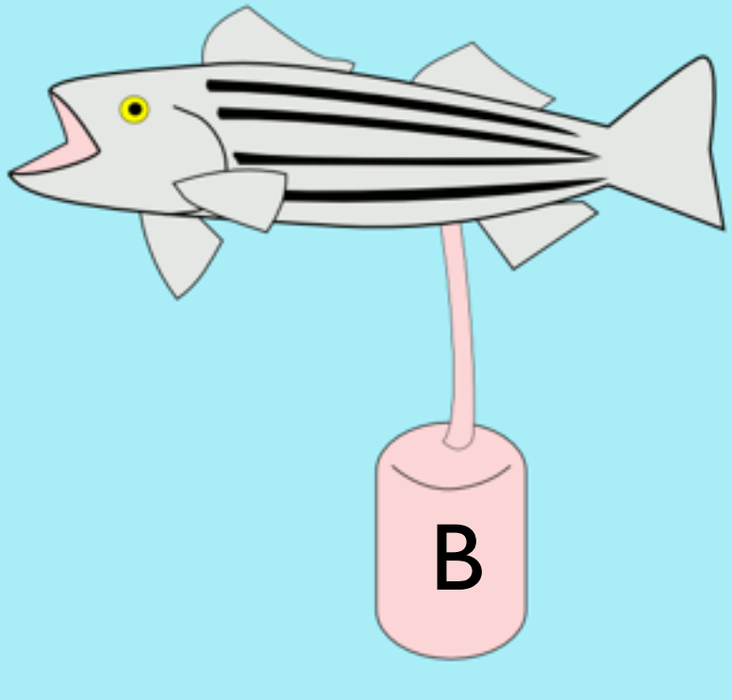




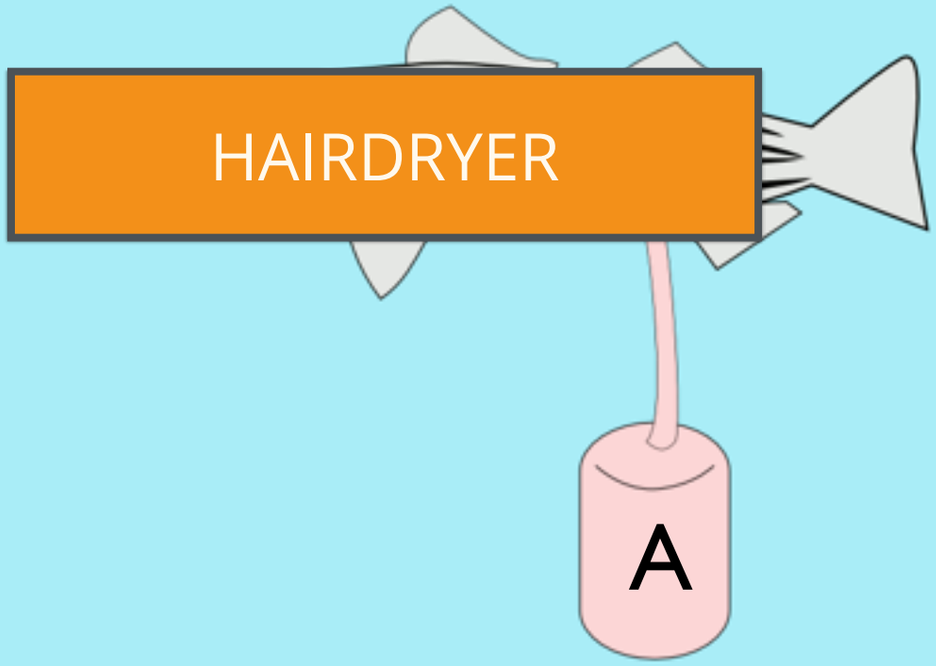
HAIRDRYER

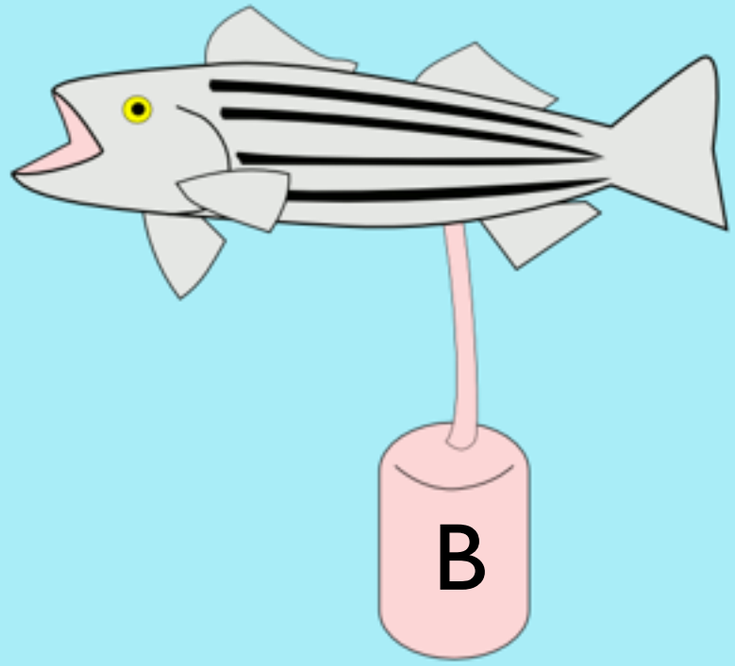
riak





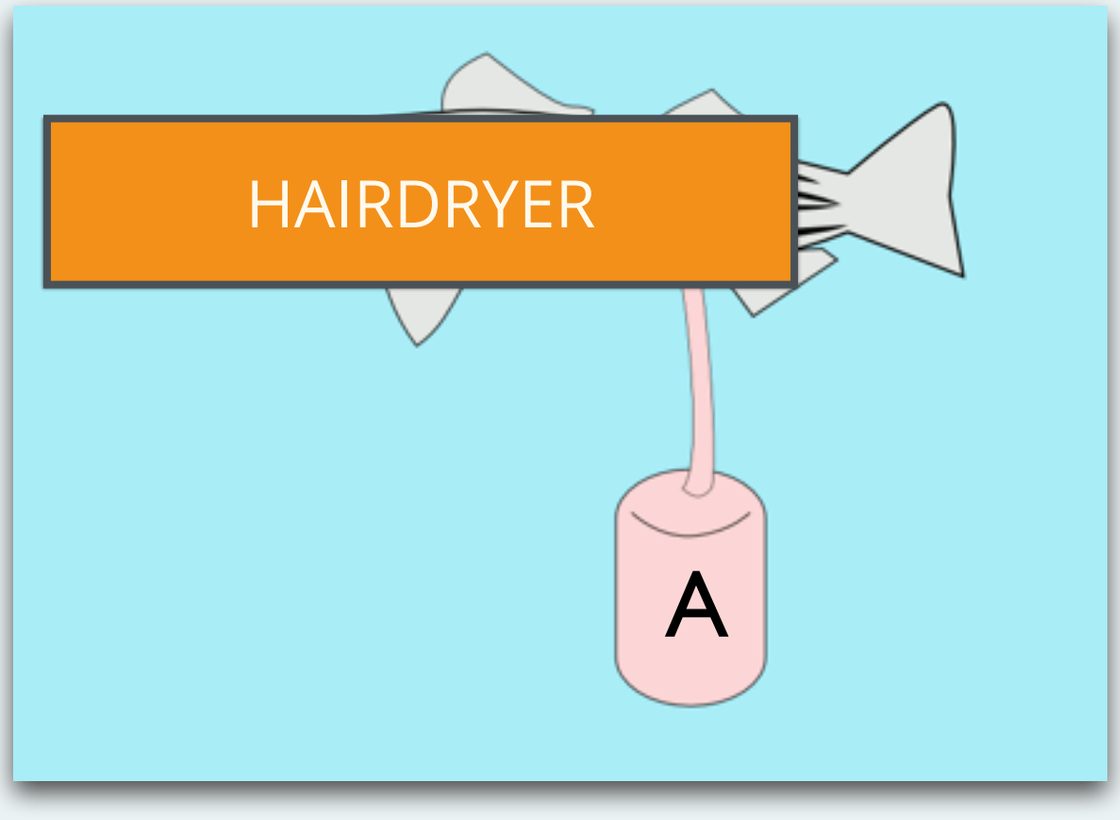
riak

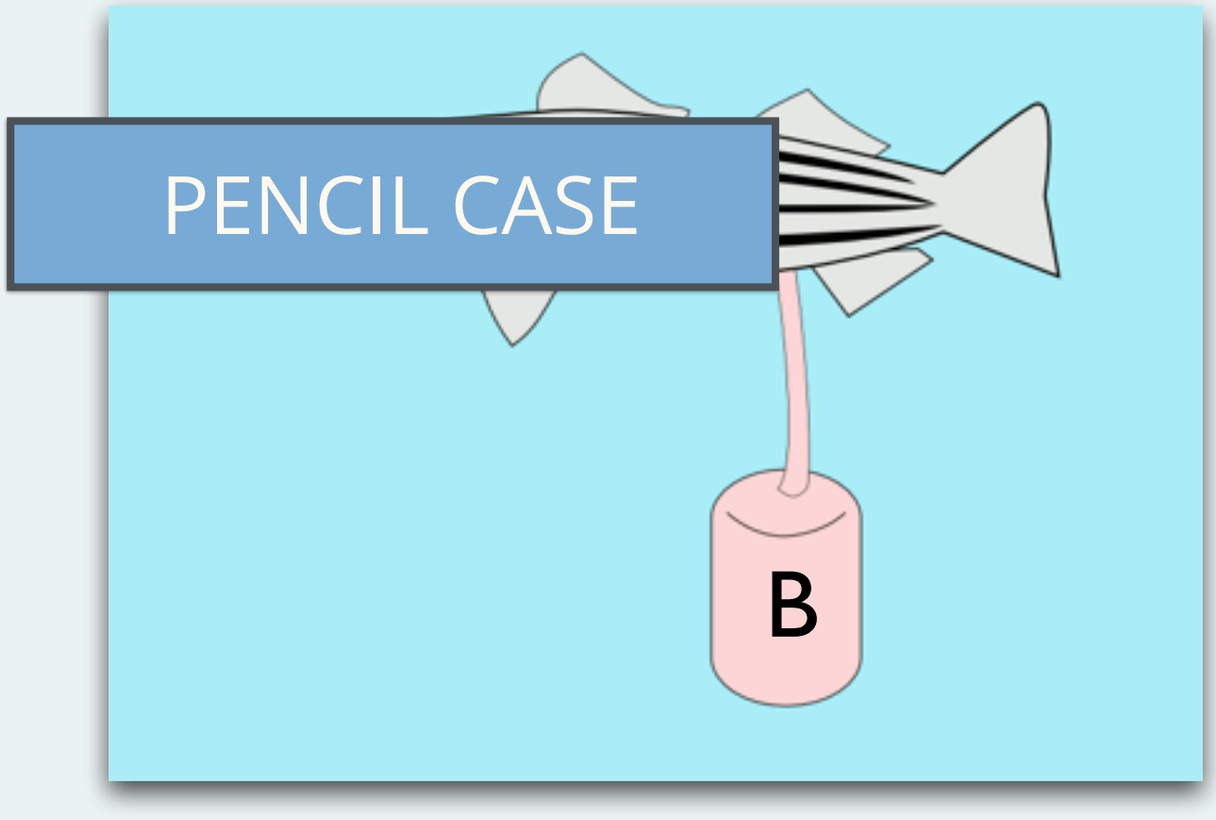


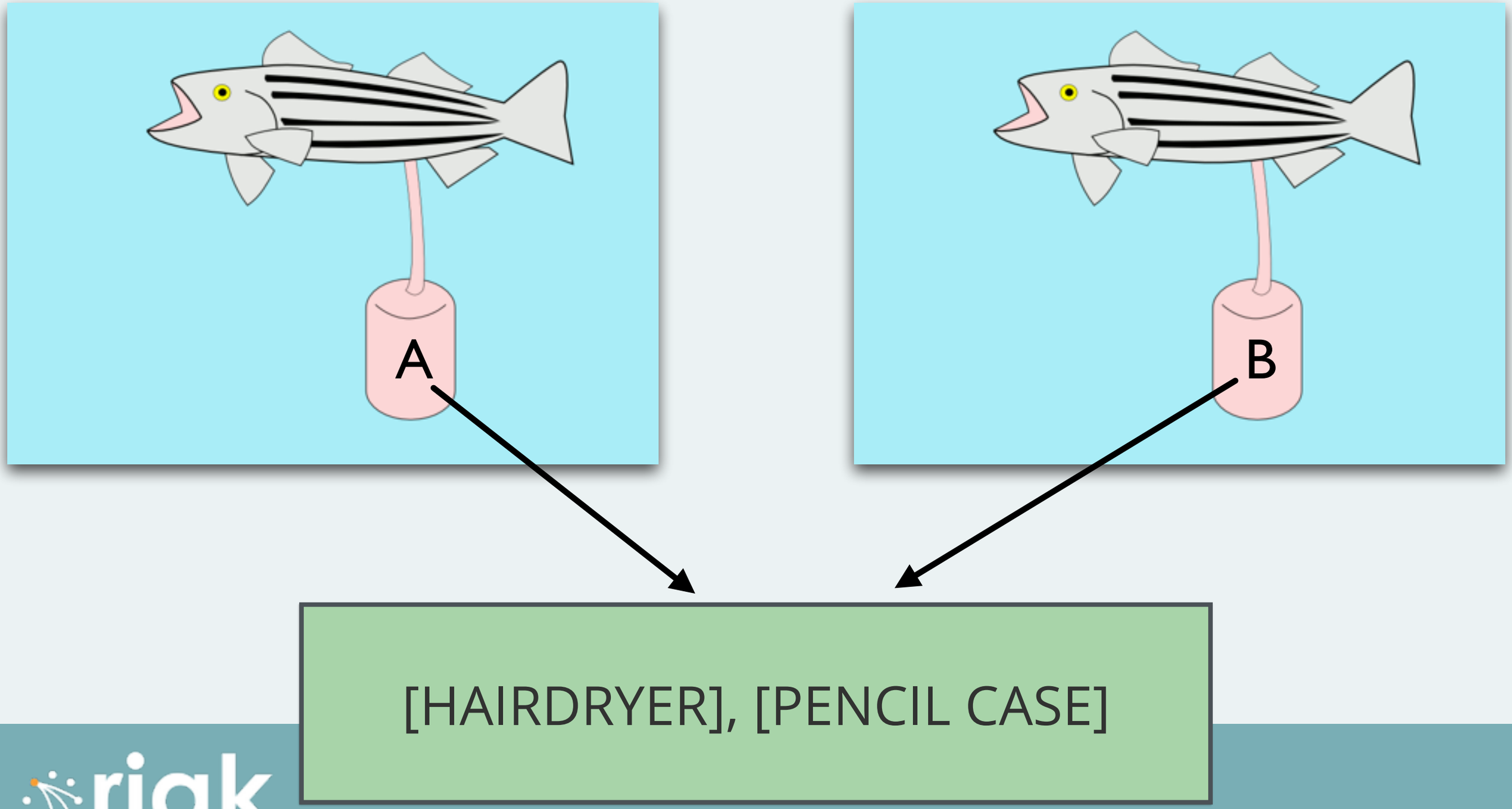


PENCIL CASE

riak


HAIRDRYER
A


PENCIL CASE
B



riak

Semantic Resolution

Merge

Set Union of Values
Simple, right?

Conflicting Writes

- Version Vector Detects Concurrency
- Multi-Value Register
- Application “merges” to a single value
  - Semantic Resolution
- Tells Riak “value is X”

Riak Part 1 Summary

- Consistent Hashing and the ring
- Availability over Consistency
- Trade-Off - Great for Business and Ops
    - Can be challenging for developers

Hands On

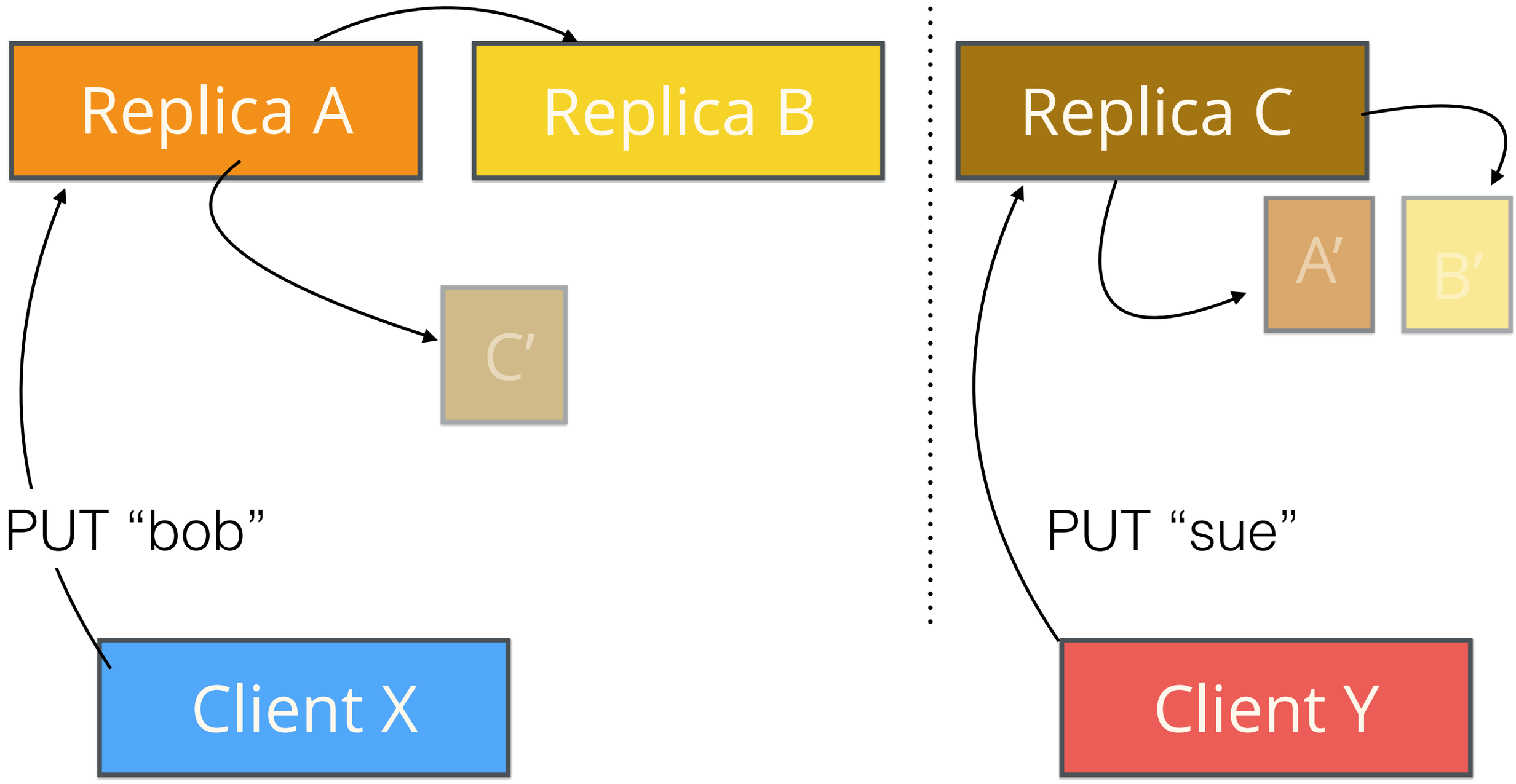
- vagrant up
- vagrant ssh
- cd lecture/riak

Hands On

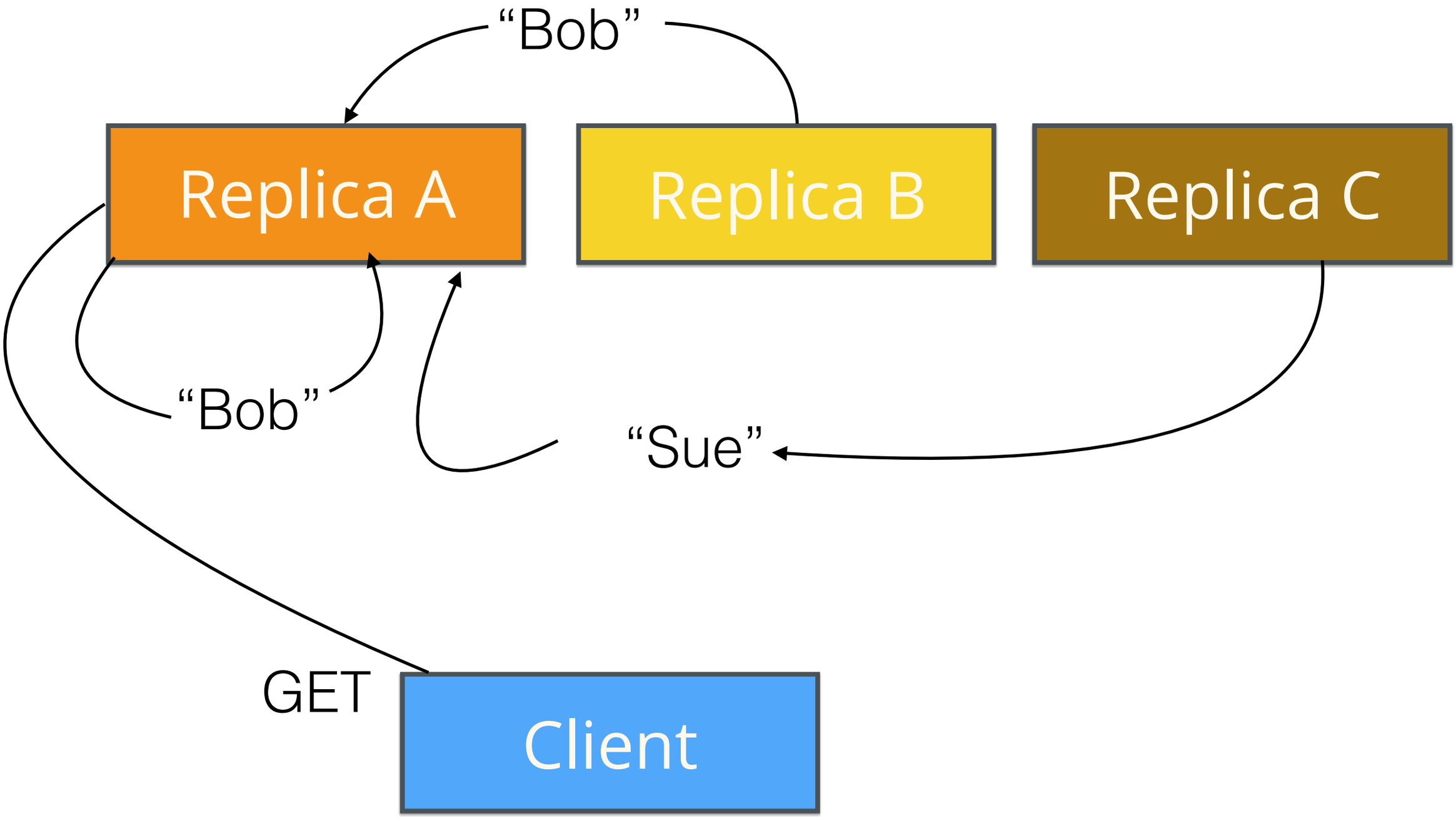
- Create a bucket
- Create some values
- Pass the vclock
  - Create some siblings!
- Data Modelling

Why CRDTs?

Conflict!


Replica A
Replica B
C′
PUT “bob”
Client X
Replica C
A′
B′
PUT “sue”
Client Y

AP



Conflict!

Google+

```
if (result.hasConflicts()) {
    // TODO: What should we do???
```

48:22 Live

Google F1

“Designing applications to cope with concurrency anomalies in their data is very error-prone, time-consuming, and ultimately not worth the performance gains.”

“…writing merge functions was likely to confuse the hell out of all our developers and slow down development…”

http://www.infoq.com/articles/key-lessons-learned-from-transition-to-nosql

Removes?

Set Union?

“Anomaly” Reappear

Absence

How can you tell if X is missing from A but present in B because A hasn't yet seen the addition, or if A has removed it already?

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A comprehensive study of Convergent and Commutative Replicated Data Types

Marc Shapiro, INRIA & LIP6, Paris, France
Nuno Preguiça, CITI, Universidade Nova de Lisboa, Portugal
Carlos Baquero, Universidade do Minho, Portugal

Marek Zawirski, INRIA & UPMC, Paris, France

13 Jan 2011

riak

"…after some analysis we found that much of our data could be modelled within sets so by leveraging CRDT's our developers don't have to worry about writing bespoke merge functions for 95% of carefully selected use cases…"

riak

http://www.infoq.com/articles/key-lessons-learned-from-transition-to-nosql

CRDT Sets

answers the question of "what is in the set?" when presented with siblings:

[x,y,z] | [w,x,y]

CRDT Sets

is w not added by A or removed by A?
is z not added be B or removed by B?

[x,y,z] | [w,x,y]

CRDT Sets

a semantic of “Add-Wins”
via
“Observed Remove”

This project is funded by the European Union,
7th Research Framework Programme, ICT call 10,
grant agreement n°609551.

Hands On

- CRDTs Sets

Data Types

- Counters
- Sets
- Booleans
- Maps
  - compose all the above (recursively)

Riak - Summary

- Always Write Available Key-Value Database
- Self-Healing fault tolerance
- Eventually Consistent
  - Be aware of trade-offs and use cases
  - CRDTs simplify data modelling
- Research on databases is ACTIVE and INTERESTING