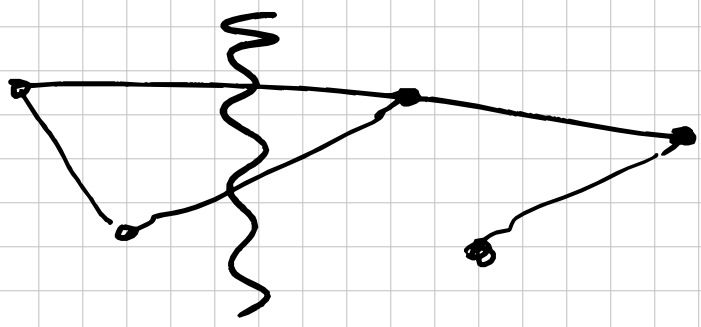
CSE138 Lecture 15

this time:

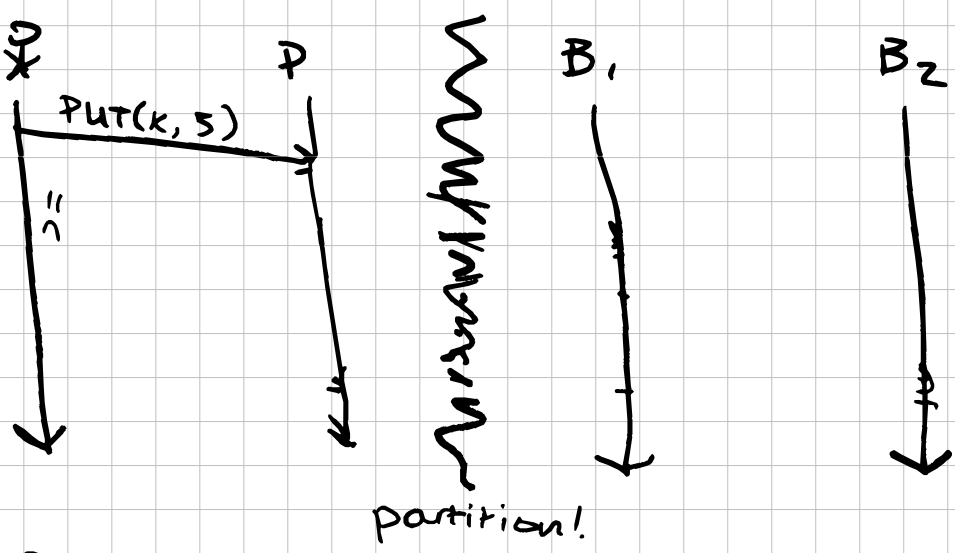
* - the consistency/availability tradeoff
  - Dynamo!
  - some new (for this class) ideas in the Dynamo paper:
    - anti-entropy with Merkle trees
    - gossip
    - quorum consistency

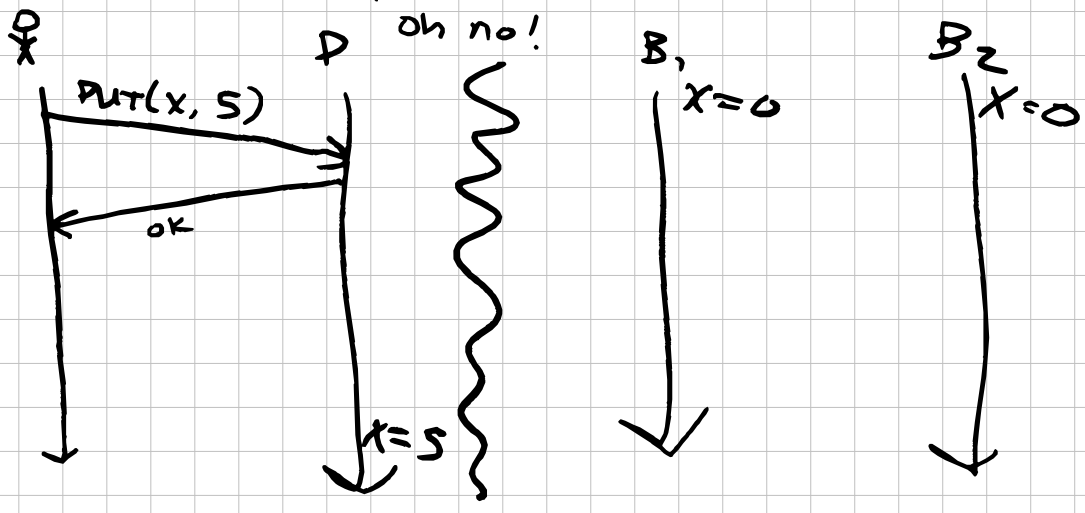


we live in a world where network partitions are inevitable.

What's perfect availability?

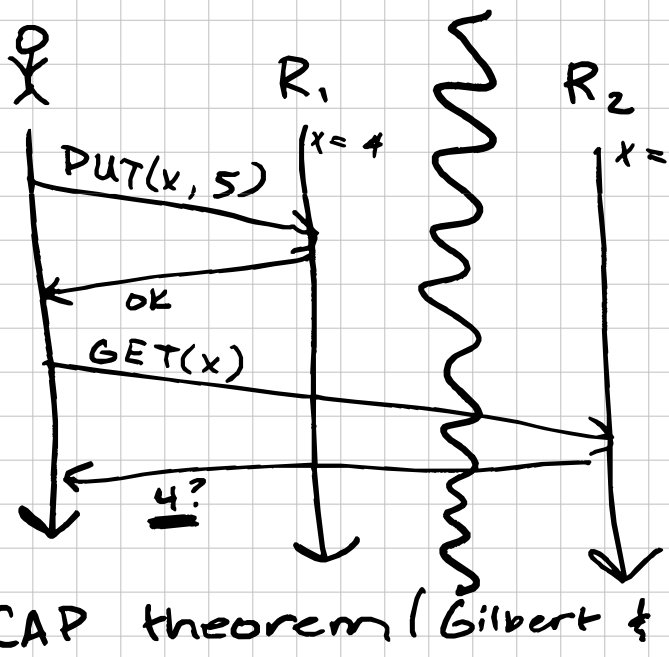
"every request eventually receives a response."





Both options are bad:

- First option sacrifices availability in favor of consistency.
- Second option sacrifices consistency in favor of availability. } Dynamo does this!



CAP theorem (Gilbert & Lynch, 2002)

↑

consistency, availability, (partition-tolerance)

↑ safety property (responses aren't "wrong") e.g. don't violate RYW, or other consistency violations

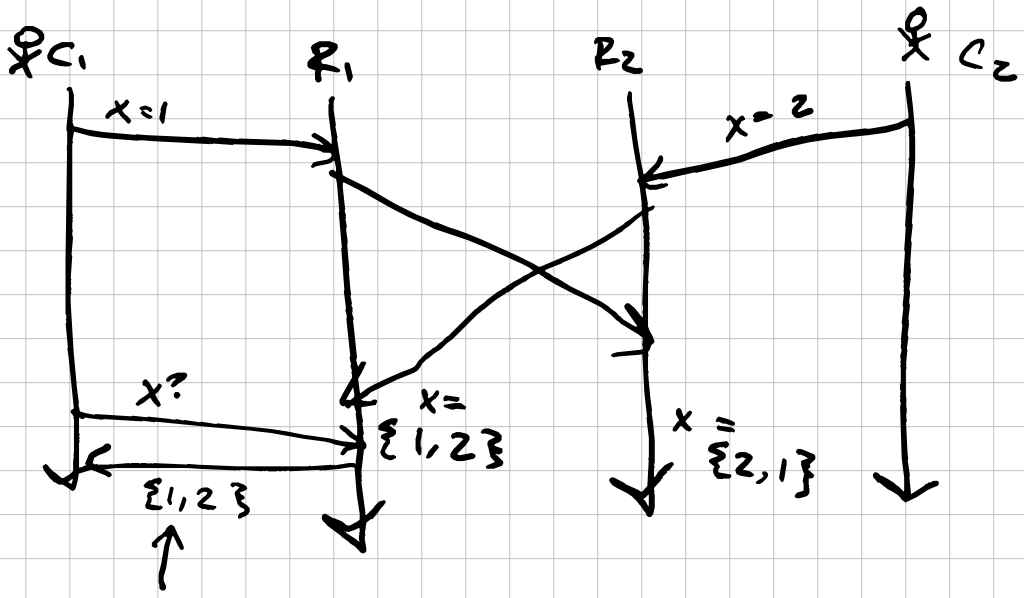
↑ liveness property (requests eventually get a response)

Strong consistency
Causal consistency
FIFO consistency
} consistency policies

Eventual consistency – a liveness property, not part of the above hierarchy

↑

Doug Terry et al.
Bayou (1995)

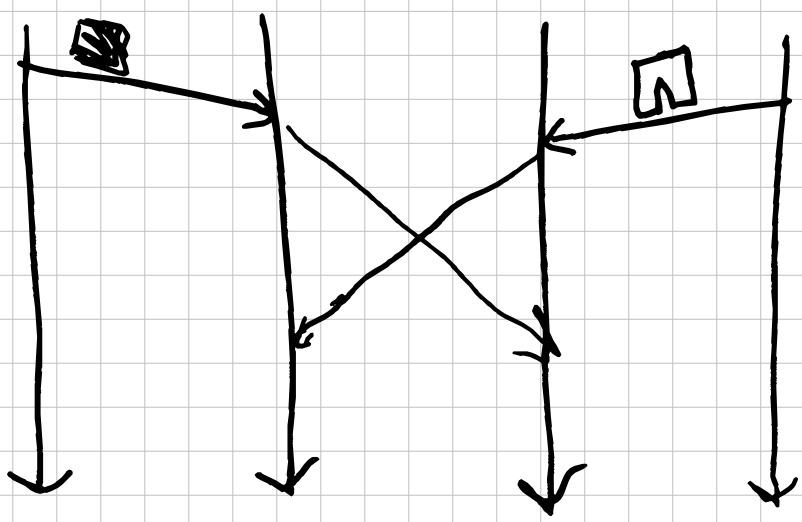


Client has to figure out how to implement their own conflict resolution.

merge of { , } and { } = { , , }
R1 R2

{ , } { , }

Deleted items may mysteriously reappear when we use set union to merge carts!



writes commute with each other:

a+b = b+a

we've decided to prioritize availability at the expense of consistency.

So replicas might disagree sometimes!

But we still want them to eventually agree.

How do replicas find out that they disagree? (and then resolve the conflict, if possible!)

Learning about disagreement

- gossip: every second (?) or some period of time, every node randomly picks another node to contact and see what its state is.
  - ↑ what nodes exist? aka the "view"

Fun Fact: Dynamo used to have a globally consistent view mechanism, but they got rid of it!

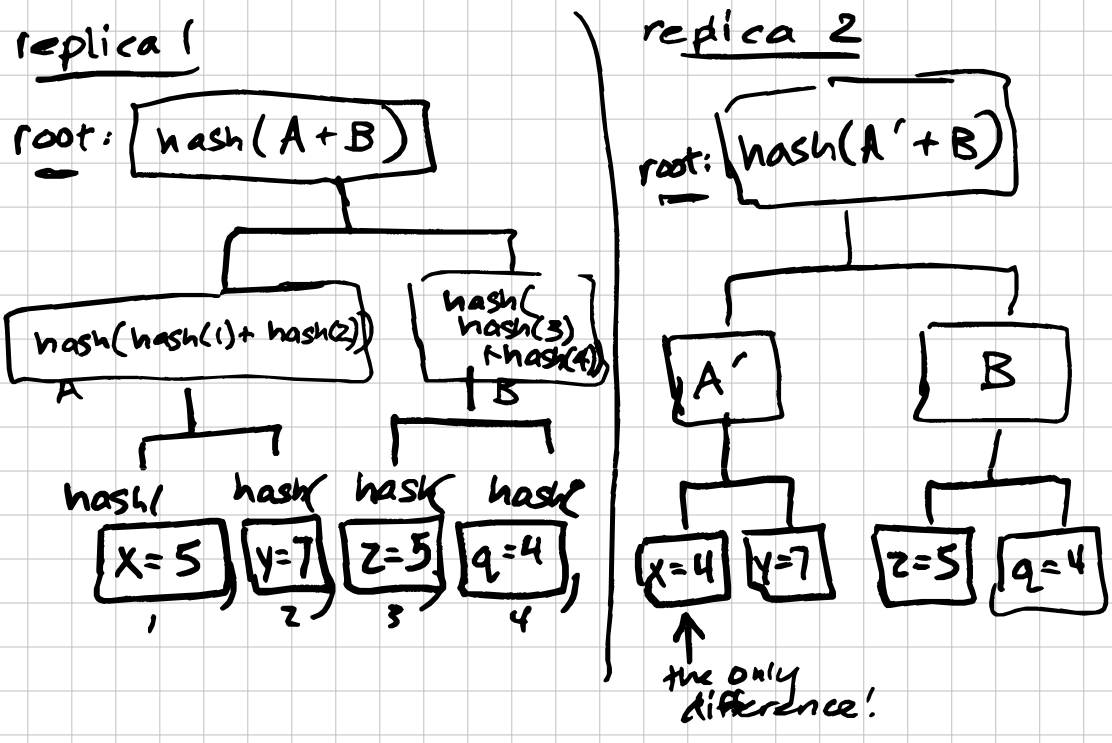
What about application state (actual key-value data)?

- anti-entropy - for resolving conflicts in application state
  - ↑ actual K-V pairs

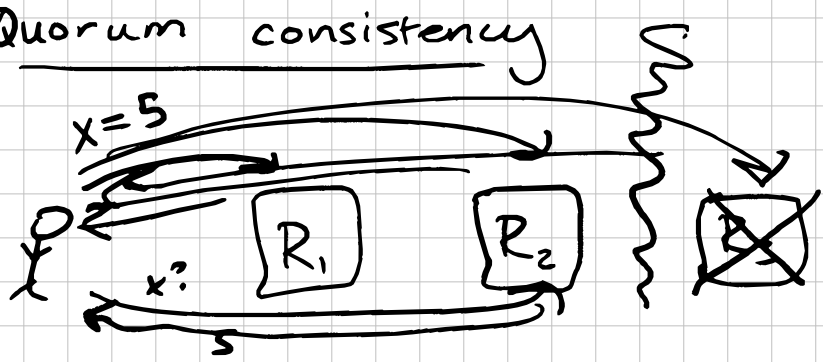
A node may have lots of K-V data, but the view is not much data.

Q: So... how to minimize the cost of K-V data transfer during replica synchronization?

A: Merkle trees! aka hash trees.



Quorum consistency



How many nodes should a client talk to?

In quorum systems, this is configurable!

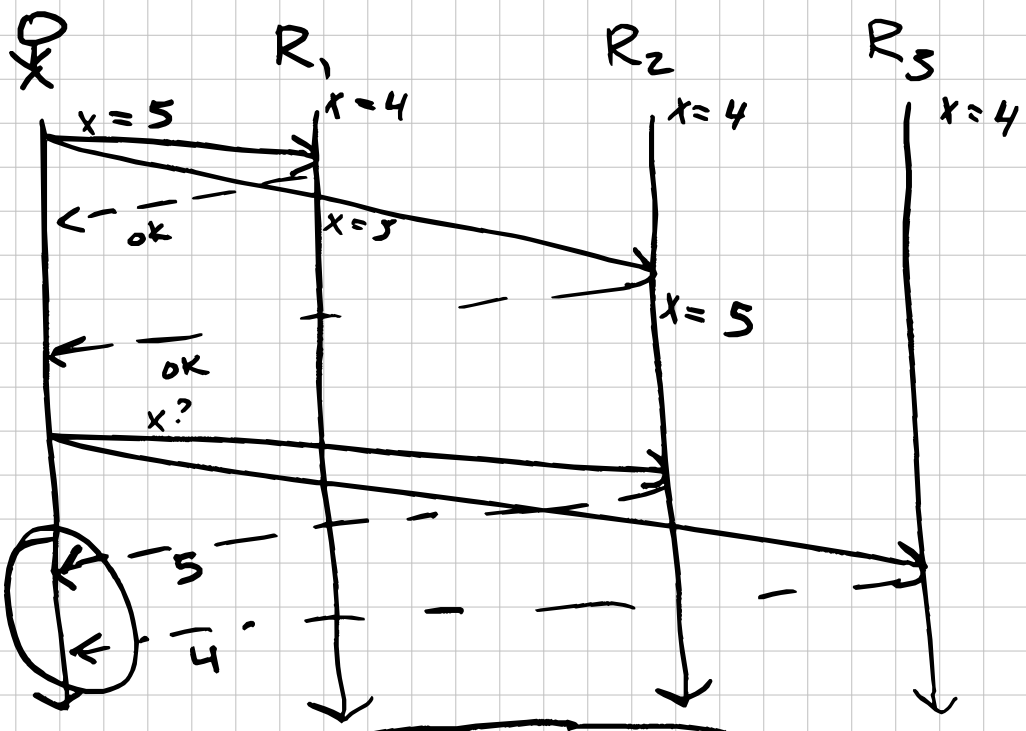
Configuration knobs:

- N — number of replicas (here, it's 3)
- W — "write quorum" — how many replicas have to acknowledge a write (for the write to be considered committed)
- R — "read quorum" — how many replicas have to acknowledge a read (for the read to be considered committed)

idea: set W = 3, R = 1 (where N = 3)

Sometimes called "ROWA" (read one, write all)

The Dynamo paper suggests W = 2, R = 2



Because $W + R > N$,

$(2 + 2 > 3$ in this case)

we're going to get at least one response that has the latest write.

Database systems that support this include Cassandra.

Design principle:

make sure that read + write quorums are intersecting somehow!