14-736: DISTRIBUTED SYSTEMS

LECTURE 12 * SPRING 2019 * KESDEN

WHY REPLICATE DATA/OBJECTS/ETC?

- Increased Throughput/Bandwidth/Parallelism
- Increased Fault Tolerance (Independent Failure)
- Increased Fault Tolerance (Local Risk)
- Improve latency by storing content closer to distributed consumers, e.g. toward edges
- Etc.

1ST CLASS REPLICAS VS 2ND CLASS REPLICAS

1st Class Replicas

- Copies of the original, equal to the original
- Common for file servers, databases, etc.

2nd Class Replicas

- Derivable from the original
- Lower fidelity in some way
  - Caches (possibly stale)
  - Thumbnails (lower resolution, faster to transmit, smaller to store)
  - Compressed copies (slower to access, harder to update)

ASSUMPTIONS FOR TODAY

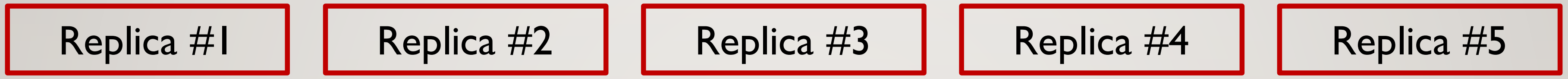
- 1st Class Replicas
- One-Copy Semantics
  - Reads and writes operate as if they are acting upon a single, non-distributed data store
  - Perspective is one of reading and writing objects vs edits
  - But, edits can be viewed as reads and writes of whole blocks/units within store.
- Concurrency control remains important.
  - Insert prior discussion here
  - We are just going to "assume" it to exist, since we've already studied it.

QUORUMS: HOW MANY COPIES TO READ/WRITE?

Replica #1 Replica #2 Replica #3 Replica #4 Replica #5

- Read quorum: Number of server from which a reader should read
- Write quorum: Number of servers to which a writer should write.
- Is there a relationship between these two?
- Can we tune them to achieve different goals?

THINKING ABOUT QUORUMS



- Do multiple versions exist? If so, how can we recognize the most up-to-date version?
- How many replicas do we need to read to get the most up-to-date version?
- How many replicas should we write? Read?
- Are the answers to these questions related to each other?
- Are the answers to these questions related to other goals?

CONFLICTS

- Goal of quorums is to avoid conflicts
- Writes cause conflicts
  - Write-Write
  - Write-Read
- Reads don’t cause conflicts
  - Read-Read is safe

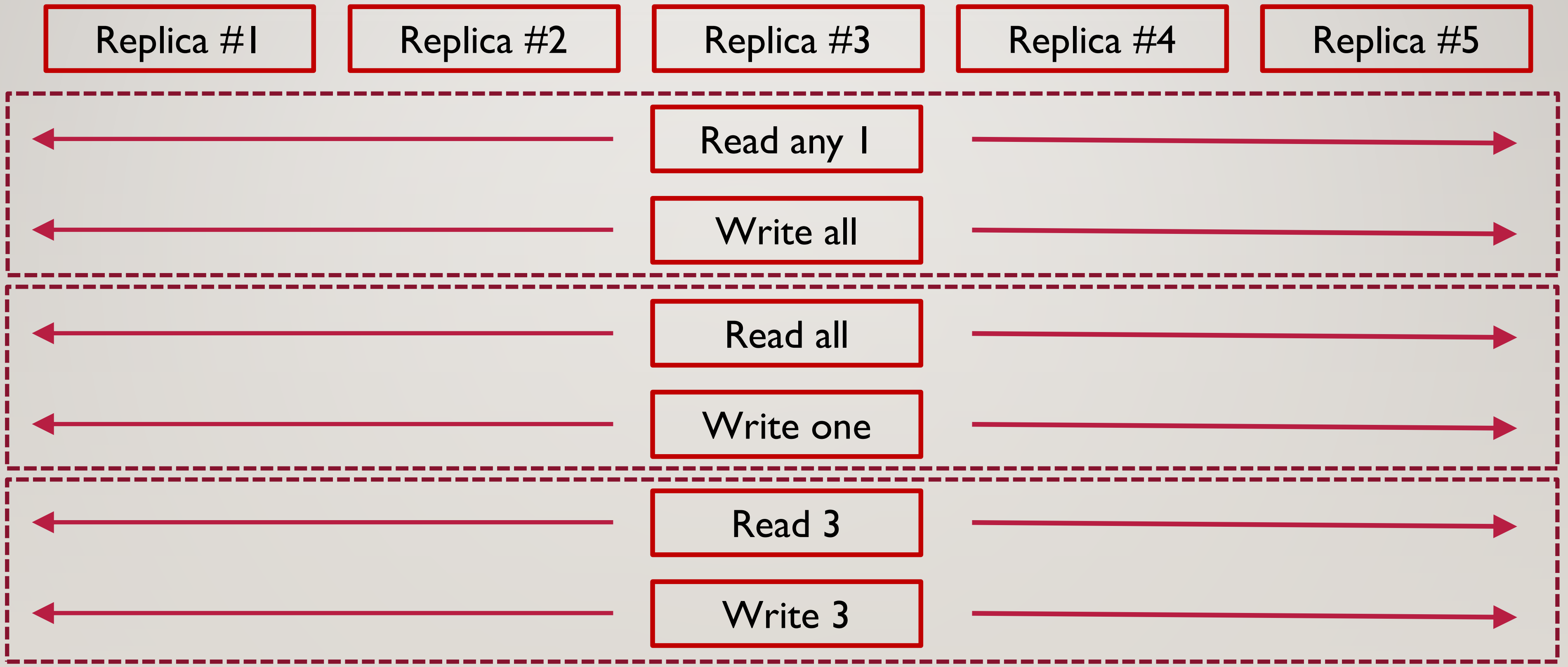
R + W > N

- The read quorum and the write quorum must overlap
  - Why? Without overlap, reads could miss write and get old value
  - R + W > N
- Can rewrite R+ W > N
  - R > N –W
  - W > N – R
- Requires the ability to identify the most recent version from among those read, e.g. by version number or timestamp.

TUNING QUORUMS

- Greater write quorum
  - More redundancy for robustness
  - Greater throughput
  - More local to consumers
  - Higher write cost (Network and device throughout, contention, long tail, etc)
- Lower read quorum
  - Lower cost to read (Network and device throughout, contention, long tail, etc)
  - More choices about where to read (closer to user)
  - More expensive writes (But reads are usually more common)
- Greater overlap
  - More tolerance to failure

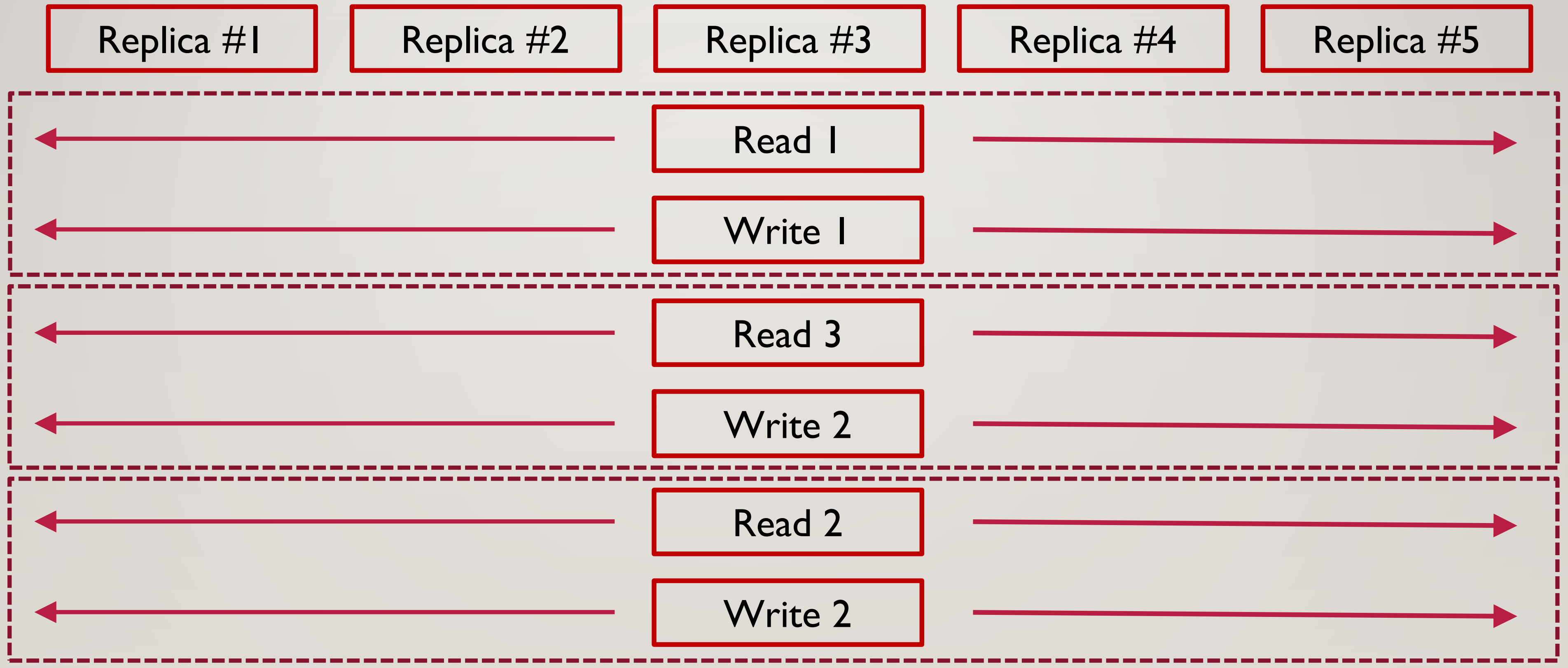
EXAMPLES OF VALID QUORUMS



EXAMPLES OF VALID QUORUMS

Read-1/Write-All	🟥 Replica #1	▨ Replica #2	▨ Replica #3	▨ Replica #4	▨ Replica #5
Write-All/Read-1	🟥 Replica #1	🟥 Replica #2	🟥 Replica #3	🟥 Replica #4	▨ Replica #5
Read-3/Write-3	🟥 Replica #1	🟥 Replica #2	▨ Replica #3	⬜ Replica #4	⬜ Replica #5
Key	🟥 Read	⬜ Write	▨ Read+Write		

EXAMPLES OF INVALID QUORUMS



EXAMPLES OF INVALID QUORUMS

Read-1/Write-1	🟥 Replica #1	▨ Replica #2	▨ Replica #3	▨ Replica #4	⬜ Replica #5
Read-3/Write-2	🟥 Replica #1	🟥 Replica #2	🟥 Replica #3	⬜ Replica #4	⬜ Replica #5
Read-2/Write-2	🟥 Replica #1	🟥 Replica #2	▨ Replica #3	⬜ Replica #4	⬜ Replica #5
Key	🟥 Read	⬜ Write	◩ Read+Write	▨ Unused	

MOST COMMON: READ-1/WRITE ALL

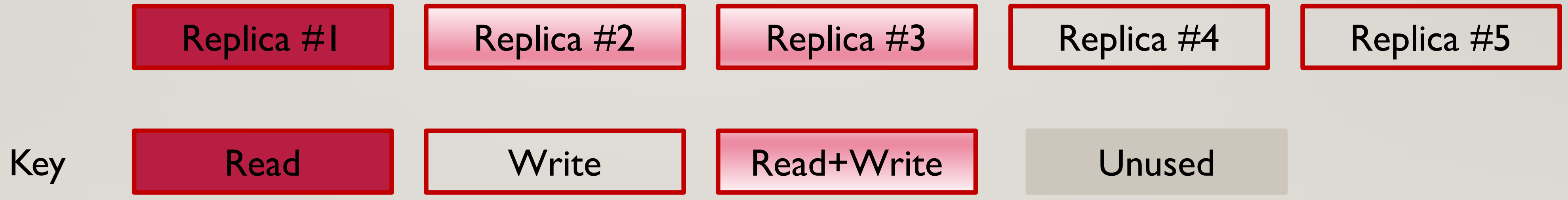
- Read-1/Write All is most common
- Reads are more common than writes
    - Read gets most current data (requirement)
    - Reads can choose any replica (nearby, failure, etc)
    - Reads require low bandwidth
    - Reads protected from long tail, etc
    - Makes common case safe and fast.

VERSION NUMBERS

- It isn’t good enough for a read to simply get the most recent version among multiple versions it might see.
  - It must be able to tell newest version is sees from older versions
- Solutions:
  - Version numbers, use highest (Easiest answer)
  - Update all versions, so they are the same, e.g. Read-All/Write-One
  - Overlap more: Guarantee majority of read quorum will have newest version (Surprisingly painful)

OVERLAPING MORE IS PAINFUL (VS VERSION NUMBERS)

- Overlap must be a majority of read quorum
  - Must write to majority of read quorum, plus
  - Must write to all members not a part of read quorum
  - W > (R/2) + (N-R), which simplifies to
  - W > (N - R/2)
  - Hope for no failures, or overlap even more



QUICK NOTE ABOUT LOCKING

- Repeat for emphasis: Prior discussion about concurrency control applies here
- Uncontrolled concurrent writes can break quorum discipline (and maybe even corrupt data)
  - Also could break version number increment
- What to lock?
  - Object at servers, not whole servers
- How many to lock?
  - L >= R: Lock quorums must cover the most recent version, or it could get an updating version
  - L > =W: Lock quorums must cover everything being written to protect version number and data
  - L > MAX (R,W): More specifically, L must be large enough so it blocks reads and writes of the updating object. Can be relaxed to cover just writes if reading outdating version is okay.

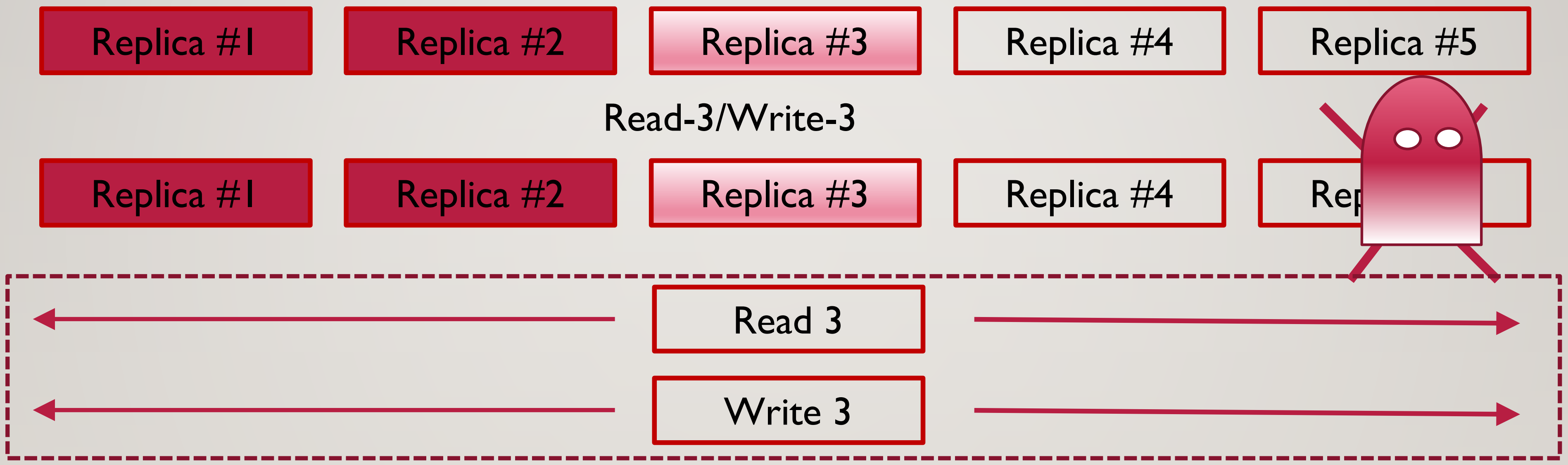
WHAT ABOUT FAILURE?

- If covered by overlap, no problem, e.g.
  - $R + W - F > N$ (Assuming version numbers)
  - $W + R/2 - F > N$ (Assuming no version numbers)
  - Still guaranteed to see up-to-date version
- If not covered by overlap, the failure model becomes important

VOTING WITH GHOSTS (VAN RENESSE, TANNENBAUM)

- If (by magic) we can assume that a dead host is distinguishable from a partitioned host…
  - Or that non-participating partitioned hosts know they are assumed to be "dead" (even more magical)
- If….we can count "ghost" of "dead" hosts toward write quorum
- This is basically like using a smaller N for the quorum
  - Writes are forced into a smaller set of hosts
  - Writes must only overlap within that set of hosts
- Doesn't work if can't meet read quorum since only copy could be on "dead" host
  - Consider write-one/Read-all
- Requires that reincarnating hosts get updates before coming online, since effective quorum will expand.

VOTING WITH GHOSTS
TANNENBAUM, VAN RENESSE



Note that reads still necessarily see most recent version.

Key	🟥 Read	☐ Write	Read+Write	▨ Unused

STATIC QUORUMS, GENERALLY

- *Static quorums* are consistent, pre-defined quorums, such as those we are discussing
  - As compared to those that may be adaptive, e.g. view-based, that we are not discussing
- So far, we have counted each replica equally
  - one replica = one vote
  - This is not a requirement
- We can assign different weights to different hosts
  - Quorum rules still apply, just account for weights not replica counts (as replicas may have different weights)
- Examples:
  - More weight to more reliable hosts
    - It takes more replicas to less reliable hosts to result in the same robustness
  - Don’t count caches
    - They can go away.

CODA VERSION VECTORS (CVVS) (QUICK REFRESHER, IN CONTEXT)

- Each CVV contains one entry for each host server. Each entry is the version number of the file on the corresponding server. In the perfect case, the entry for each replica will be identical. But, should an update reach only a portion of the servers, some servers will have newer versions than others.

CODA CVVS AND CLIENT READS

In Coda, the client request a file via a three-step process.

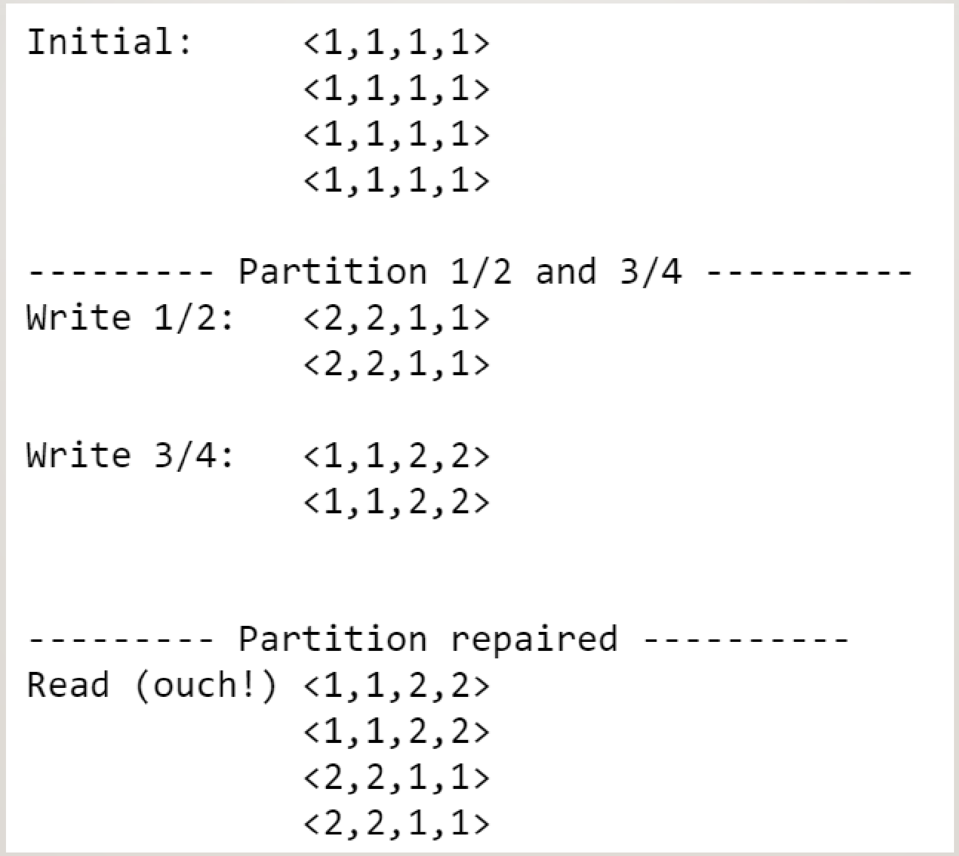
- It asks all replicas for their version number
- It then asks the replica with the greatest version number for the file
- If the servers don't agree about the files version, the client can direct the servers to update a client that is behind, or inform them of a conflict. CVVs are compared just like vector timestamps. A conflict exists if two CVVs are concurrent, because concurrent vectors indicate that each server involved has seen some changes to the file, but not all changes.

CODA CVVS AND CLIENT WRITES

In the perfect case, when the client writes a file, it does it in a multi-step process:

- The client sends the file to all servers, along with the original CVV.
- Each server increments its entry in the file's CVV and ACKS the client.
- The client merges the entries form all of the servers and sends the new CVV back to each server.
- If a conflict is detected, the client can inform the servers, so that it can be resolved automatically, or flagged for mitigation by the user.

CODA CVV EXAMPLE



```
Initial:     <1,1,1,1>
             <1,1,1,1>
             <1,1,1,1>
             <1,1,1,1>

--------- Partition 1/2 and 3/4 ----------
Write 1/2:   <2,2,1,1>
             <2,2,1,1>

Write 3/4:   <1,1,2,2>
             <1,1,2,2>



--------- Partition repaired ----------
Read (ouch!) <1,1,2,2>
             <1,1,2,2>
             <2,2,1,1>
             <2,2,1,1>
```