

CPSC 416 – Distributed Systems

Lecture 16: Consistency Models + Leaderless Replication

UBC Vancouver

2026-03-05

From Last Time

Turn to your neighbor. 2 minutes.

Chain replication gives us strong consistency by structuring the write path: head $\rightarrow$ tail.

Why does that work? What property does the chain structure guarantee?

(Pairs: 2 minutes)

Checkpoint 1

The chain imposes a *total order* on writes.

Reads go to the tail, which has seen every committed write.

But this requires a fixed topology and a clear leader (the head).

What if you don't want a leader?

What if you're willing to give something up?

Failure-First Question

Your distributed database returns a value.

Is it the latest value?

How would you even know?

With chain replication, the answer was clear: the tail has everything.

Without that structure, what does “latest” even mean?

What Does “Consistent” Mean?

We’ve been using the word “consistent” loosely.

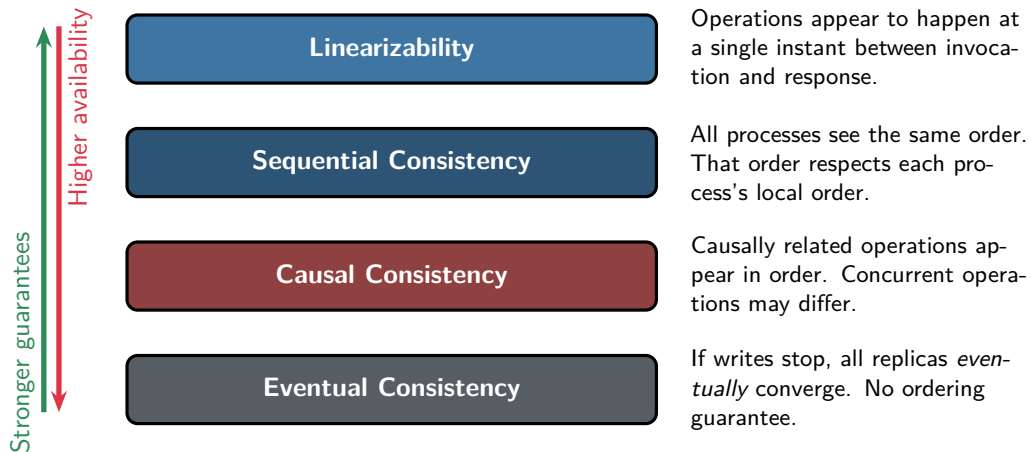
Today we make it precise.

A consistency model is a contract between the system and the client.

It says: “If you do operations in this order, here’s what you’re guaranteed to see.”

Different contracts, different prices.

The Consistency Hierarchy



Linearizability: The Gold Standard

Definition: Every operation appears to take effect at some instant between its invocation and its response.

What this means:

- If write W completes before read R starts, R sees W
- All observers agree on the order
- The system behaves as if there's a single copy of the data

What it costs:

- Coordination on every operation
- Vulnerable to network partitions (must choose: block or risk inconsistency)

Chain replication, Paxos, Raft all provide linearizability.

Sequential Consistency: A Subtle Relaxation

Definition: All processes see the same interleaving of operations, and that interleaving respects each process's program order.

Difference from linearizability:

Linearizability respects *real time*. If W finishes at 10:01 and R starts at 10:02, R must see W .

Sequential consistency only respects *program order*. Operations from different processes can be reordered as long as each process's local order is preserved.

Cheaper. But “stale” reads become legal.

Causal and Eventual: Weaker Still

Causal consistency:

- If A writes x , then reads x , then writes y — everyone sees x before y
- But two unrelated writes can appear in different orders to different readers

Eventual consistency:

- If all writes stop, replicas converge *eventually*
- No bound on how long “eventually” takes
- No ordering guarantee during convergence

“Eventual” is not a timing guarantee. It’s the absence of one.

Checkpoint 2

You read a key K from two different replicas, one millisecond apart. You get different values.

Under which consistency models is this legal?

Under which is it a bug?

(Pairs: 2 minutes)

The CAP Theorem

You've probably heard: "Pick two of three: Consistency, Availability, Partition tolerance."

That's not wrong. But it's imprecise.

CAP, Stated Precisely (Brewer 2000, Gilbert & Lynch 2002)

In a system that can experience network partitions:

You cannot simultaneously provide:

- **Consistency** (linearizability — every read sees the most recent write)
- **Availability** (every non-failed node responds to every request)

Key clarifications:

- “C” means *linearizability*, not just “replicas agree”
- “A” means *every* non-failed node responds, not “most do”
- Partitions are not a choice — they happen. The choice is C or A during a partition

CAP is not “pick two.” It’s “during a partition, pick one.”

CAP in Practice

CP systems (consistent during partitions):

- Paxos, Raft, chain replication
- During partition: minority side stops serving reads/writes
- *Safe but not live for all clients*

AP systems (available during partitions):

- Dynamo, Cassandra, Riak
- During partition: all nodes keep serving, may diverge
- *Live but not linearizable*

Most real systems are *not purely* CP or AP.

They choose different trade-offs for different operations.

Enter Dynamo

Amazon, 2007. The shopping cart problem.

Requirement: The “Add to Cart” button must **always** work.

Even during data center failures. Even during network partitions.

Design choice: Availability over consistency.

A customer might see a stale cart. They must never see a dead page.

This is an engineering decision, not a theoretical one.

Dynamo's Architecture

No leader. No fixed topology. No consensus protocol.

- N replicas for each key (typically 3)
- Any replica can accept reads or writes
- Coordinator forwards to all N replicas
- Waits for W write acknowledgments (write quorum)
- Waits for R read responses (read quorum)

The knobs: N , R , W .

If $R + W > N$, reads and writes overlap.

Where have we seen this before?

L10 Returns

From L10, the quorum intersection property:

$$\forall Q_R, Q_W : Q_R \cap Q_W \neq \emptyset$$

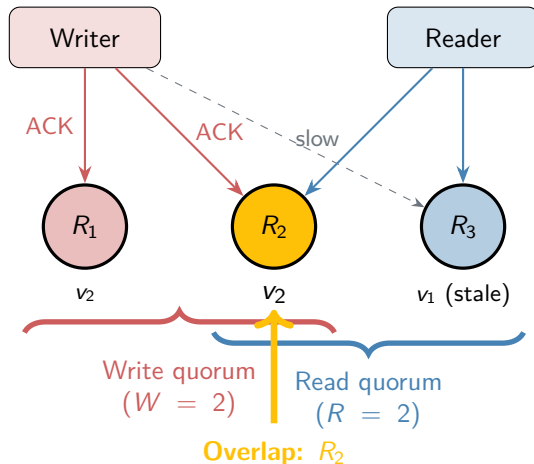
L10's weighted voting (Gifford 1979): $R + W > V$.

Dynamo: $R + W > N$.

Same invariant. Different instantiation. Different era.

The quorum intersection property is the gift that keeps giving.

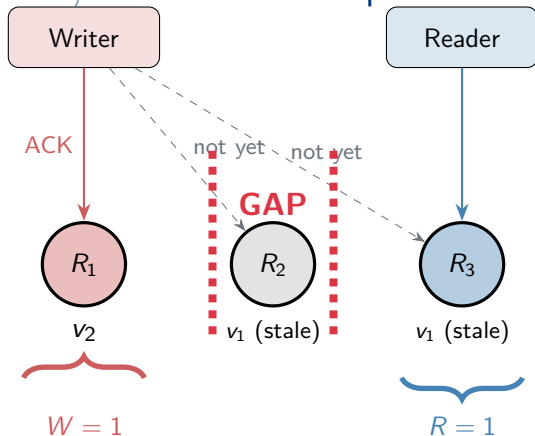
Dynamo: $R = 2, W = 2, N = 3$ — The Overlap



Reader contacts R_2 and R_3 . Sees v_2 and v_1 . Takes the **latest version**. Gets v_2 . ✓

$R + W = 4 > 3 = N$. The overlap guarantees a witness.

Dynamo: $R = 1, W = 1, N = 3$ — The Gap



Reader contacts R_3 . Sees v_1 . **Stale**.

$R + W = 2 \leq 3 = N$. **No overlap guaranteed. No witness.**

This is the L10 intersection failure, dressed up in a key-value store.

The Tuning Knobs

R	W	$R + W$ vs $N=3$	Profile
2	2	$4 > 3$ ✓	Default: strong overlap
1	3	$4 > 3$ ✓	Read-optimized (write is slow)
3	1	$4 > 3$ ✓	Write-optimized (read is slow)
1	1	$2 \leq 3$ ✗	Best effort: eventual only
1	2	$3 = 3$ ≤	Edge case: not >, stale possible

Strict inequality matters: $R + W > N$, not $\geq$.

A write can reach W nodes and a read can reach R nodes with *no overlap* if $R + W = N$ and the timing is unlucky.

Same quorum arithmetic from L10 and L14. Different system, same math.

When Replicas Diverge

With $R + W > N$, reads overlap with writes.

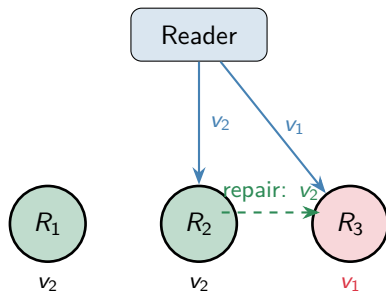
But what about the replicas that *missed* the write?

Three mechanisms:

- 1 **Read repair:** When a read finds stale replicas, push the latest value to them
- 2 **Anti-entropy:** Background process compares replicas, resolves differences
- 3 **Gossip:** Nodes periodically exchange state with random peers

None of these are instant. All of them are probabilistic.

Read Repair



Reader gets v_2 (correct). Then pushes v_2 to R_3 .

Every read is an opportunity to heal.

Gossip and Anti-Entropy

Gossip protocol:

- Periodically, each node picks a random peer
- They exchange digests (Merkle trees, version vectors)
- Stale entries get updated

Properties:

- Convergence is *probabilistic*, not deterministic
- Epidemic model: updates spread like a virus through the cluster
- With n nodes and k rounds of gossip, coverage $\approx 1 - e^{-k}$

Anti-entropy: The full-repair version. Compare entire replicas, reconcile all differences. Expensive, but thorough. Run in the background.

Read repair is eager. Gossip is lazy. Anti-entropy is paranoid.

Version Vectors: Who Wins?

When replicas diverge, how do you know which value is “newer”?

Vector clocks / version vectors:

- Each replica maintains a vector of counters: $[R_1:3, R_2:5, R_3:2]$
- On write, increment your own entry
- On read, take component-wise max

If one vector dominates another: clear winner, take the newer one.

If neither dominates: **conflict**. The vectors are concurrent.

Dynamo's answer to conflicts: **return both versions to the client.**

Let the application decide. (Amazon's cart: union the items.)

This is where “eventually consistent” gets messy.

When Is “Probably Consistent” Good Enough?

The spectrum is not binary.

- **Shopping cart:** Losing an item is worse than seeing a duplicate. Choose availability.
- **Bank transfer:** Seeing a stale balance means double-spending. Choose consistency.
- **Social media feed:** A post appears 2 seconds late. Nobody cares.
- **Inventory count:** Overselling is expensive. Consistency matters.

The right consistency model depends on the cost of being wrong.

In-Class Activity

Setup: A Dynamo-style store with $N = 3$ replicas.

Part A: Set $R = 2$, $W = 2$.

- Write $K \leftarrow v_2$ to replicas $\{R_1, R_2\}$.
- Read K from replicas $\{R_2, R_3\}$.
- **Draw the trace.** Does the reader see v_2 ? Why?

Part B: Now set $R = 1$, $W = 1$.

- Same write: $K \leftarrow v_2$ to replica $\{R_1\}$ only.
- Read K from replica $\{R_3\}$ only.
- **Draw the trace.** What does the reader see? What went wrong?

(Groups of 3: 8 minutes. Then discuss.)

Activity Debrief

Part A ($R = 2, W = 2$):

- Write quorum $\{R_1, R_2\}$, read quorum $\{R_2, R_3\}$
- Overlap at R_2 , which has v_2
- Reader sees v_2 from R_2 and v_1 from R_3 . Takes newest. **Correct.**

Part B ($R = 1, W = 1$):

- Write quorum $\{R_1\}$, read quorum $\{R_3\}$
- No overlap. R_3 still has v_1
- Reader sees v_1 . **Stale read.**

What changed?

The quorum intersection vanished. $R + W = 2 \leq 3 = N$.

No witness exists to connect the write to the read.

Same failure mode as L10's broken intersection. Same math.

Checkpoint 3

Dynamo uses $R + W > N$ for consistency.

Paxos uses $Q_1 \cap Q_2 \neq \emptyset$ for safety.

Are these the same thing?

If not, what's different? What does Paxos give you that Dynamo doesn't?

(Pairs: 3 minutes)

Quorums Are Not Consensus

Dynamo's $R + W > N$:

- Guarantees a read *overlaps* a write
- Does **not** guarantee ordering between concurrent writes
- Conflicts are possible — resolved by application or last-writer-wins

Paxos's quorum intersection:

- Guarantees overlap **plus** ballot ordering
- Chosen values propagate and are never overwritten
- No conflicts by construction

Quorum overlap is necessary for consistency. It is not sufficient.

Dynamo trades the ordering guarantee for availability under partitions.

Evidence Question

You operate a Dynamo-style store with $R = 2$, $W = 2$, $N = 3$.

A client reports stale reads during normal operation (no partitions).

What would you check?

What would you log to distinguish between:

- a quorum that's not actually overlapping (misconfiguration),
- a replica that's slow to apply writes (async lag), and
- a version vector conflict resolved incorrectly?

What We Learned

- **Consistency is a spectrum:** linearizability $\rightarrow$ sequential $\rightarrow$ causal $\rightarrow$ eventual
- Each level is a contract: stronger guarantees, higher coordination cost
- **CAP:** during a partition, choose consistency or availability
- **Dynamo:** leaderless, tunable $R + W > N$ for quorum overlap
- Convergence via read repair, gossip, anti-entropy — all probabilistic
- **Quorum overlap is necessary but not sufficient** for strong consistency

The arc:

L10 gave us quorum intersection.

L14 showed it's more flexible than majority.

L15 gave us strong consistency via chain structure.

L16: what happens when we relax the guarantees?

Next (L17): what if nodes don't just crash — they lie?

- **Reading for L17:** Lamport, Shostak, Pease — “The Byzantine Generals Problem” (1982)
- **Optional:** Dynamo (DeCandia et al., 2007) — the full paper behind today’s lecture
- Self-assessment opens 13:00, due next class 11:30 (Pacific)
- Capstone checkpoint details on Canvas