

---

---

# Client Consistency, Consistency Protocols, Replica Management

21 June 2026  
Lecture 14

Slide Credits: Maarten van Steen

# Topics for Today

---

- Client centric consistency
- Consistency Protocols
- Replica management
  - Content Replication
  - Content Distribution
  
- Source: TvS 7.5, 8

# Client-centric Consistency Models

---

## Overview:

- System Model
  - Monotonic reads
  - Monotonic writes
  - Read-your-writes
  - Write-follows-reads
- 

## Goal

Show how we can perhaps avoid system-wide consistency, by concentrating on what specific **clients** want, instead of what should be maintained by servers.

# Consistency for Mobile Users

---

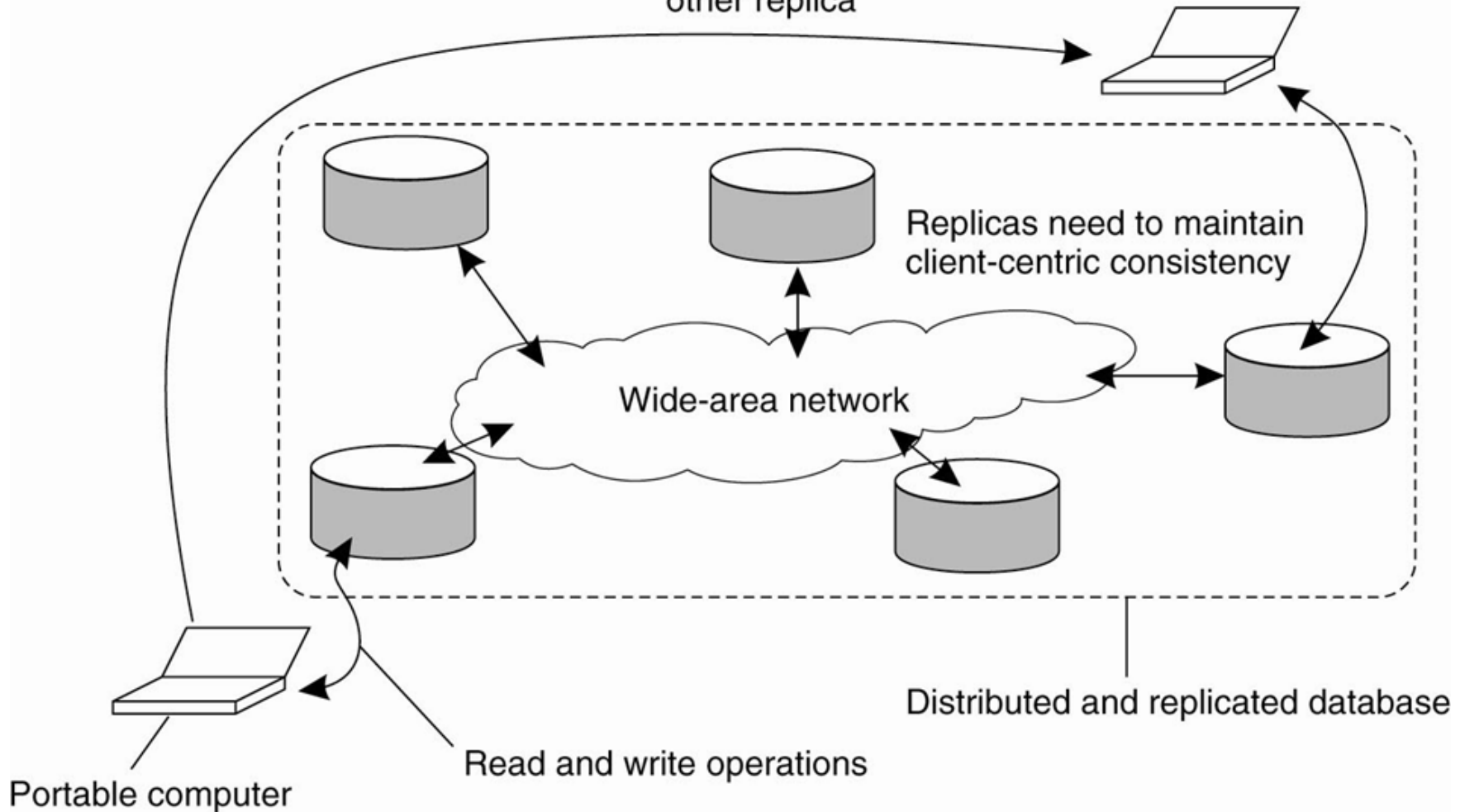
**Example:** Consider a distributed database to which you have access through your notebook. Assume your notebook acts as a front end to the database.

- At location A you access the database doing reads and updates.
- At location B you continue your work, but unless you access the same server as the one at location A, you may detect inconsistencies:
  - your updates at A may not have yet been propagated to B
  - you may be reading newer entries than the ones available at A
  - your updates at B may eventually conflict with those at A

**Note:** The only thing you really want is that the entries you updated and/or read at A, are in B the way you left them in A. In that case, the database will appear to be consistent **to you**.

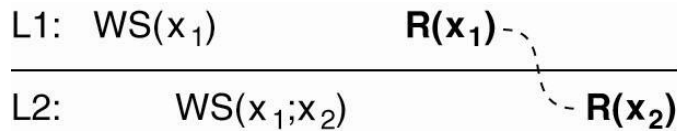
# Basic Architecture

Client moves to other location  
and (transparently) connects to  
other replica

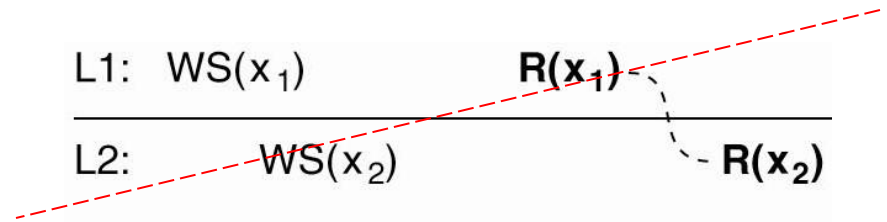


# Monotonic Reads (1/2)

*If a process reads the value of a data item  $x$ , any successive read operation on  $x$  by that process will always return that same or a more recent value.*



(a)



(b)

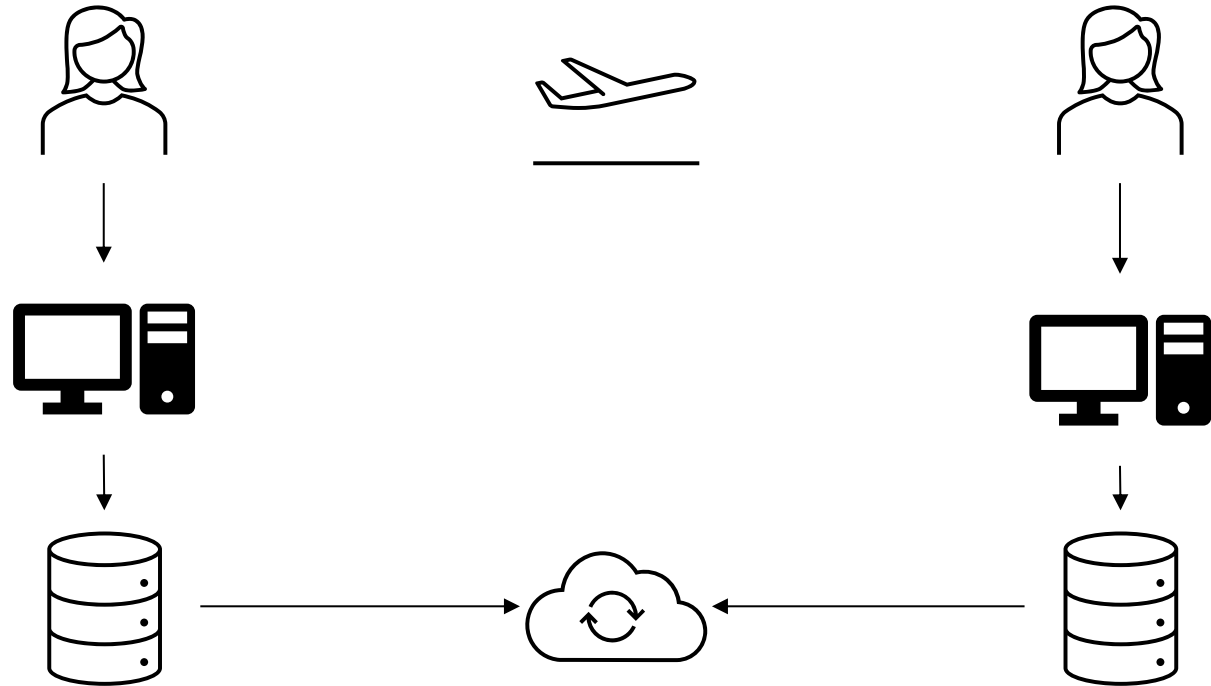
**Notation:**  $WS(x_i[t])$  is the set of write operations (at  $L_i$ ) that lead to version  $x_i$  of  $x$  (at time  $t$ );  $WS(x_i[t_1]; x_j[t_2])$  indicates that it is known that  $WS(x_i[t_1])$  is part of  $WS(x_j[t_2])$ .

**Note:** Parameter  $t$  is omitted from figures

# Monotonic Reads

---

---



File1 last modified at 10:11am  
File2 last modified at 9:14am  
File3 last modified at 6:45pm

File1 last modified at 10:51am  
File2 last modified at 9:14am  
File3 last modified at 7:50pm

# Monotonic Reads (2/2)

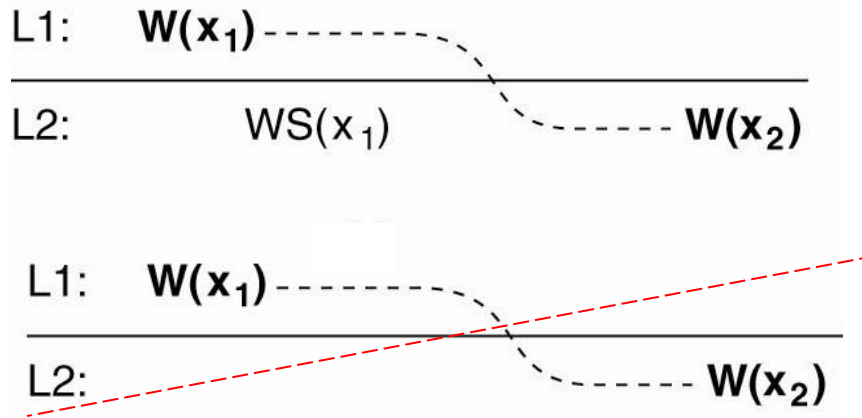
---

**Example:** Automatically reading your personal calendar updates from different servers. Monotonic Reads guarantees that the user sees all updates, no matter from which server the automatic reading takes place.

**Example:** Reading (not modifying) incoming mail while you are on the move. Each time you connect to a different e-mail server, that server fetches (at least) all the updates from the server you previously visited.

# Monotonic Writes

*A write operation by a process on a data item  $x$  is completed before any successive write operation on  $x$  by the same process.*

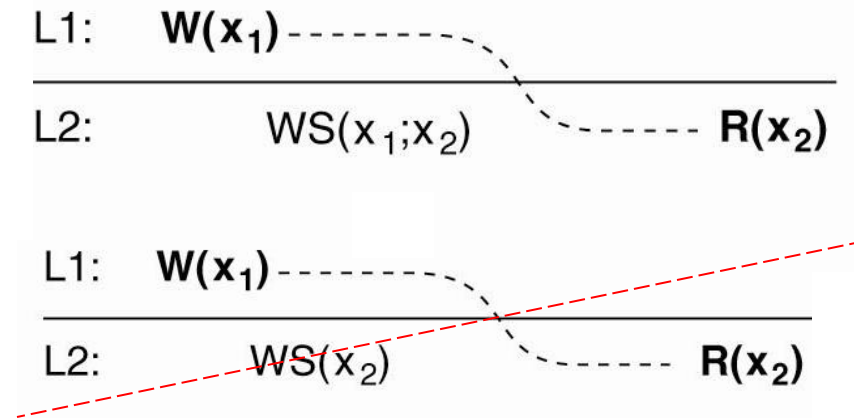


**Example:** Updating a program at server  $S_2$ , and ensuring that all components on which compilation and linking depends, are also placed at  $S_2$ .

**Example:** Maintaining versions of replicated files in the correct order everywhere (propagate the previous version to the server where the newest version is installed).

# Read Your Writes

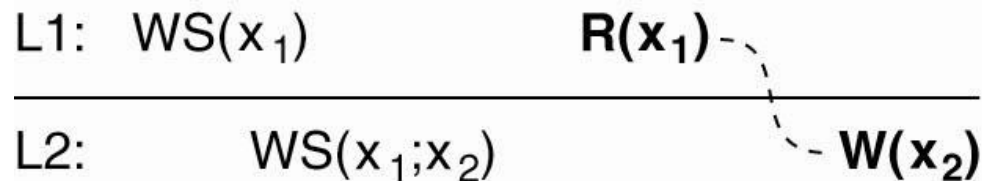
*The effect of a write operation by a process on data item  $x$ , will always be seen by a successive read operation on  $x$  by the same process.*



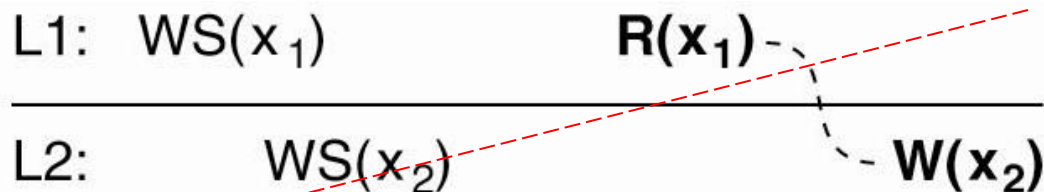
**Example:** Updating your Web page and guaranteeing that your Web browser shows the newest version instead of its cached copy.

# Writes Follow Reads

*A write operation by a process on a data item  $x$  following a previous read operation on  $x$  by the same process, is guaranteed to take place on the same or a more recent value of  $x$  that was read.*



(a)



(b)

**Example:** See reactions to posted articles only if you have the original posting (a read “pulls in” the corresponding write operation).

# So Far

---

- Client centric consistency
- Consistency Protocols
- Replica management
  - Content Replication
  - Content Distribution

# Consistency Protocols

---

**Consistency protocol:** describes the implementation of a specific consistency model.

- Continuous consistency
- Primary-based protocols
- Replicated-write protocols

# Continuous Consistency: Numerical Errors

---

**Principle:** consider a data item  $x$  and let  $weight(W)$  denote the numerical change in its value after a write operation  $W$ . Assume that  $\forall W: weight(W) > 0$

$W$  is initially forwarded to one of the  $N$  replicas, denoted as  $origin(W)$ .  $TW[i, j]$  are the writes execute by server  $S_i$  that originated from  $S_j$ :

$$TW[i, j] = \sum\{weight(W) \mid (origin(W) = S_j) \wedge (W \in L_i)\}$$

**Note:** Actual value  $v(t)$  of  $x$ :

$$v(t) = v_{init} + \sum_{k=1}^N TW[k, k]$$

Value  $v_i$  of  $x$  at replica  $i$ :

$$v_i = v_{init} + \sum_{k=1}^N TW[i, k]$$

# Continuous Consistency: Numerical Errors

---

**Problem:** We need to ensure that  $v(t) - v_i < \delta_i$  for every server  $S_i$ .

**Approach:** Let every server  $S_k$  maintain a **view**  $TW_k[i, j]$  of what it believes is the value of  $TW[i, j]$ . This information can be **gossiped** when an update is propagated.

**Note:**  $0 \leq TW_k[i, j] \leq TW[i, j] \leq TW[j, j]$

**Solution:**  $S_k$  sends operations from its log to  $S_i$  when it sees that  $TW_k[i, k]$  is getting too far from  $TW[k, k]$ , in particular, when

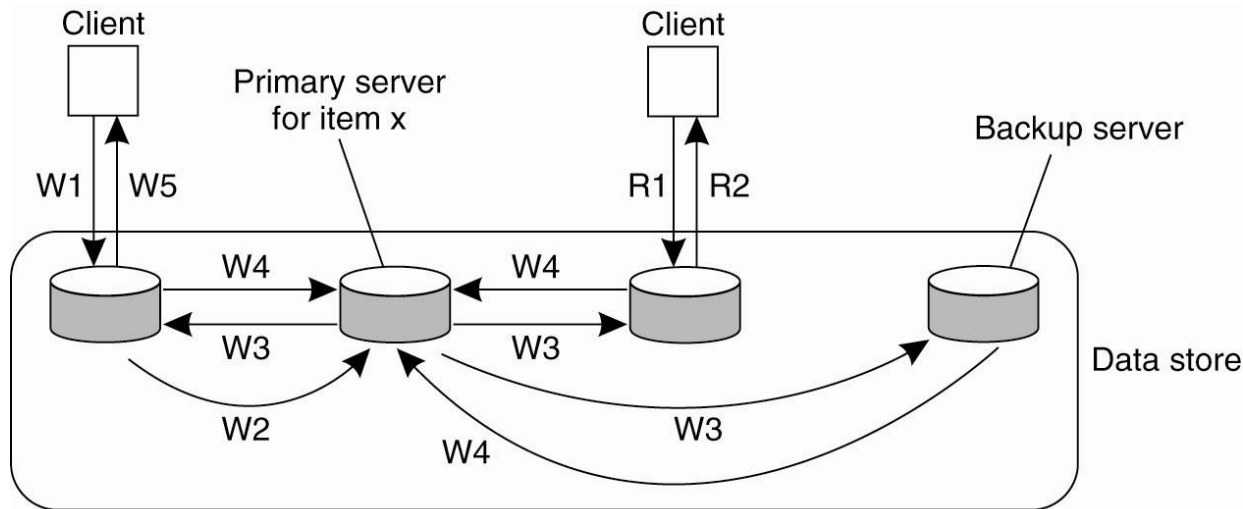
$$TW[k, k] - TW_k[i, k] > \frac{\delta_i}{N - 1}$$

---

**Note:** **Staleness** can be done analogously, by essentially keeping track of what has been seen last from  $S_i$  (see book).

# Primary-Based Protocols (1/2)

## Primary-backup protocol:



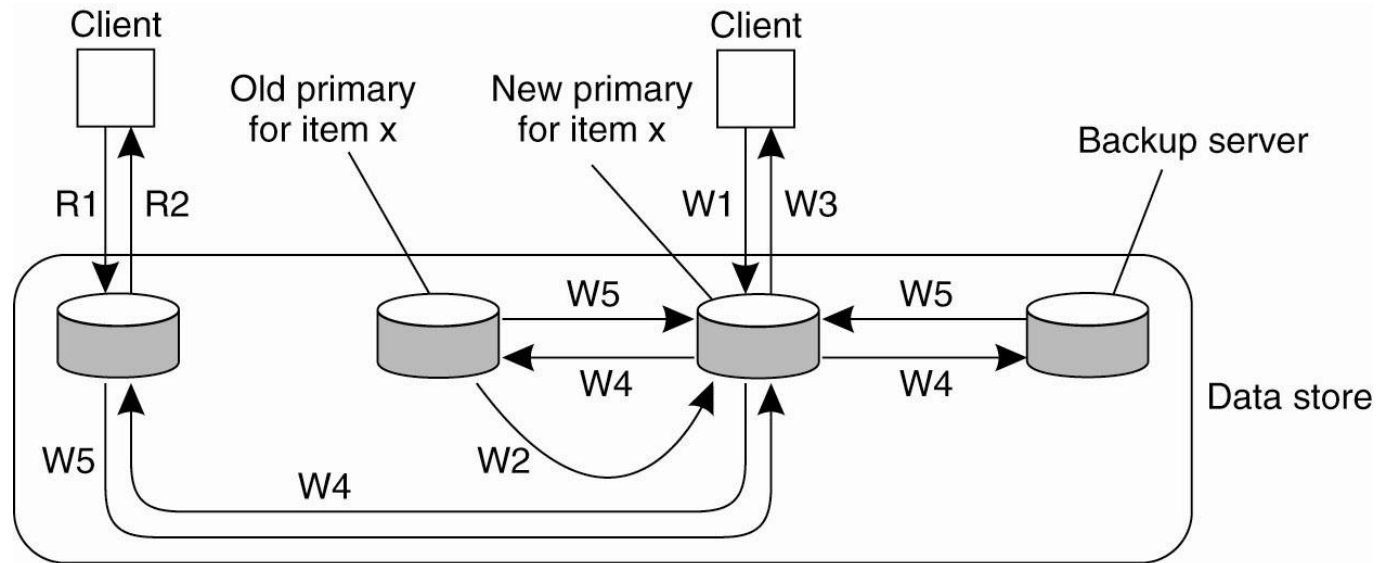
W1. Write request  
W2. Forward request to primary  
W3. Tell backups to update  
W4. Acknowledge update  
W5. Acknowledge write completed

R1. Read request  
R2. Response to read

**Example:** Traditionally applied in distributed databases and file systems that require a high degree of fault tolerance. Replicas are often placed on same LAN.

# Primary-Based Protocols (2/2)

## Primary-backup protocol with local writes:



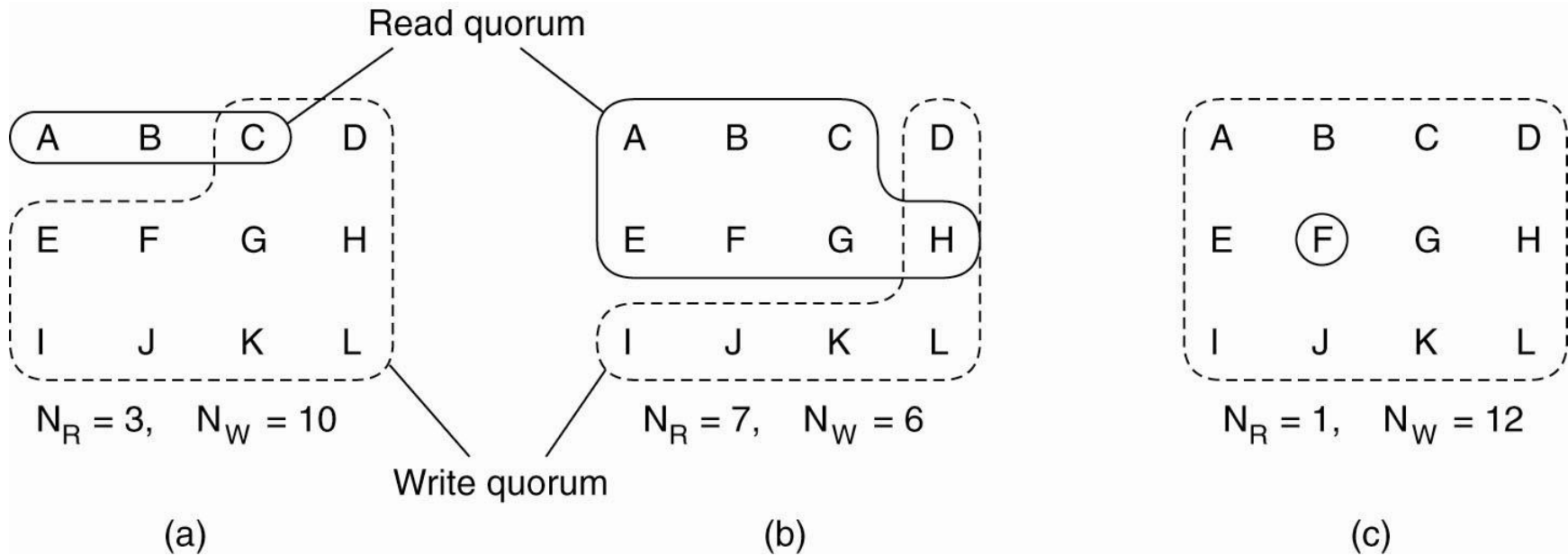
W1. Write request  
W2. Move item x to new primary  
W3. Acknowledge write completed  
W4. Tell backups to update  
W5. Acknowledge update

R1. Read request  
R2. Response to read

**Example:** Mobile computing in disconnected mode (ship all relevant files to user before disconnecting and update later).

# Replicated-Write Protocols

**Quorum-based protocols:** Ensure that each operation is carried out in such a way that a majority vote is established: distinguish **read quorum** and **write quorum**:



# Replicated Write Issues

---

Prevents issues  
with two writes

Still can end up  
with “split-brain”  
without protocol to  
coordinate servers

The larger the  
write quorum, the  
less often sync is  
needed

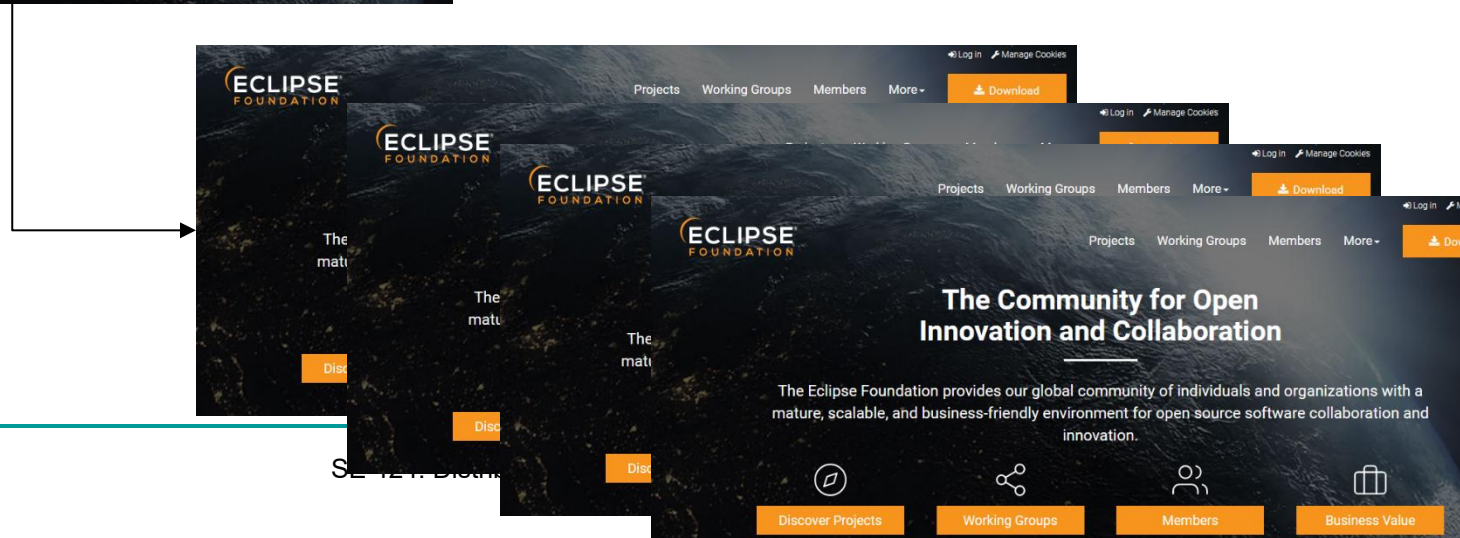
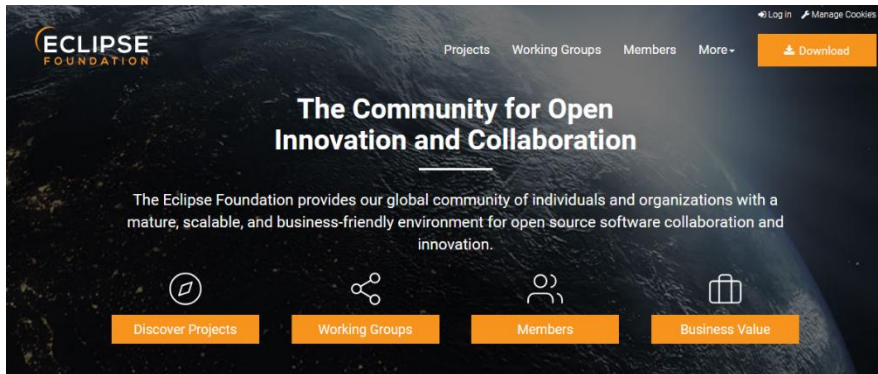
# So Far

---

- Client centric consistency
- Consistency Protocols
- **Replica management**
  - Content Replication
  - Content Distribution

# Distribution Protocols

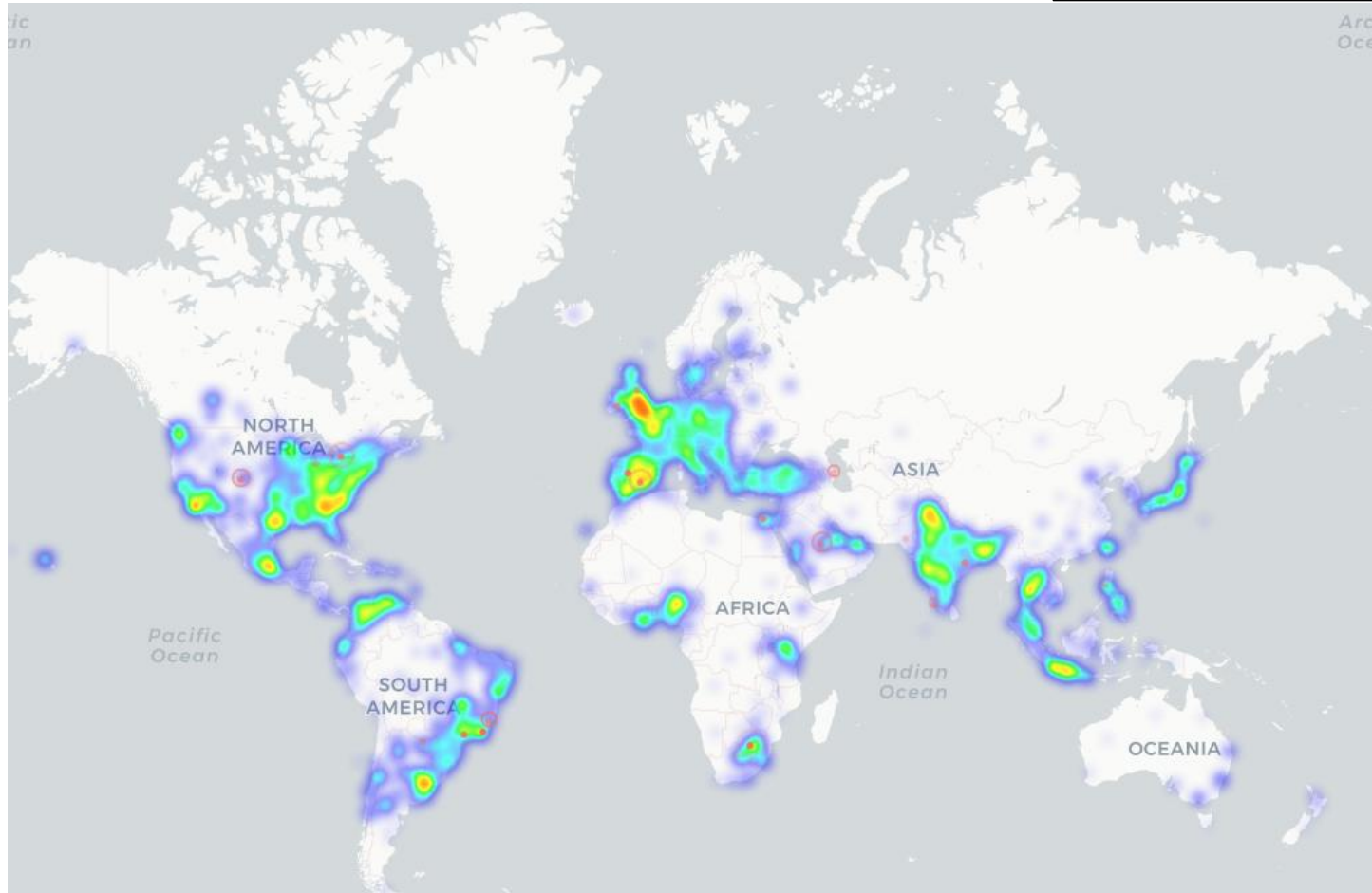
- Replica server placement
- Content replication and placement
- Content distribution



# Replica Placement

**Essence:** Figure out what the best  $K$  places are out of  $N$  possible locations.

Source: <https://onemilliontweetmap.com>



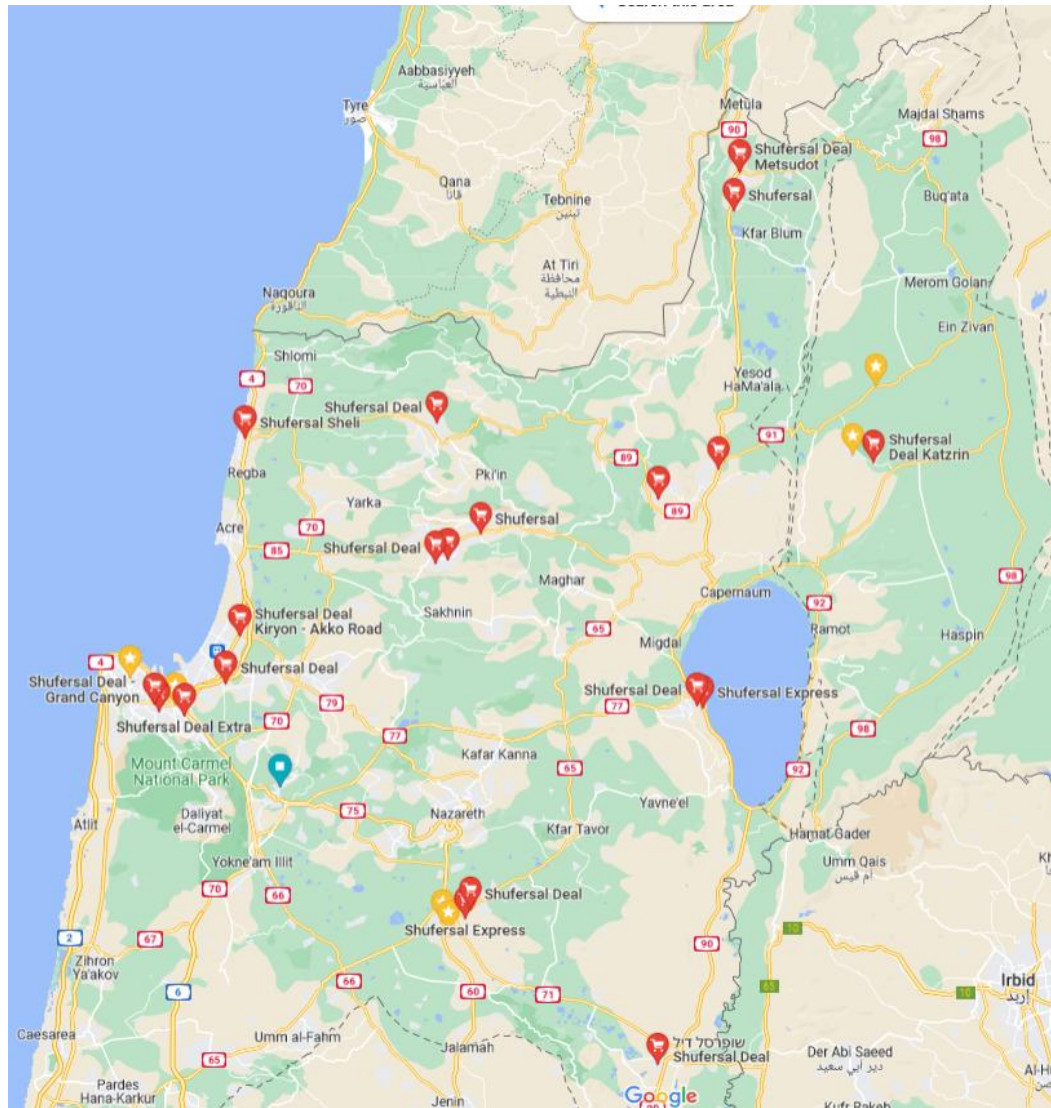
# Replica Placement

---

**Essence:** Figure out what the best  $K$  places are out of  $N$  possible locations.

1. Select best location out of  $N$  for which the **average distance to clients is minimal**. Remove the location from  $N$ , add it to set  $k$ . Then choose the next best server until  $k$  has  $K$  elements. (Note: The first chosen location minimizes the average distance to all clients.) **Computationally expensive** (imagine 50 out of 1K locations, 5M clients).
2. Select the  $K^{th}$  largest **autonomous system** and place a server at the best-connected host. **Computationally expensive** (where is the best connected host?)
3. Position nodes in a  $d$ -dimensional geometric space, where distance reflects latency. Identify the  $K$  regions with highest density and place a server in each. **Computationally cheaper**.

# Going to add one more Shufersal.

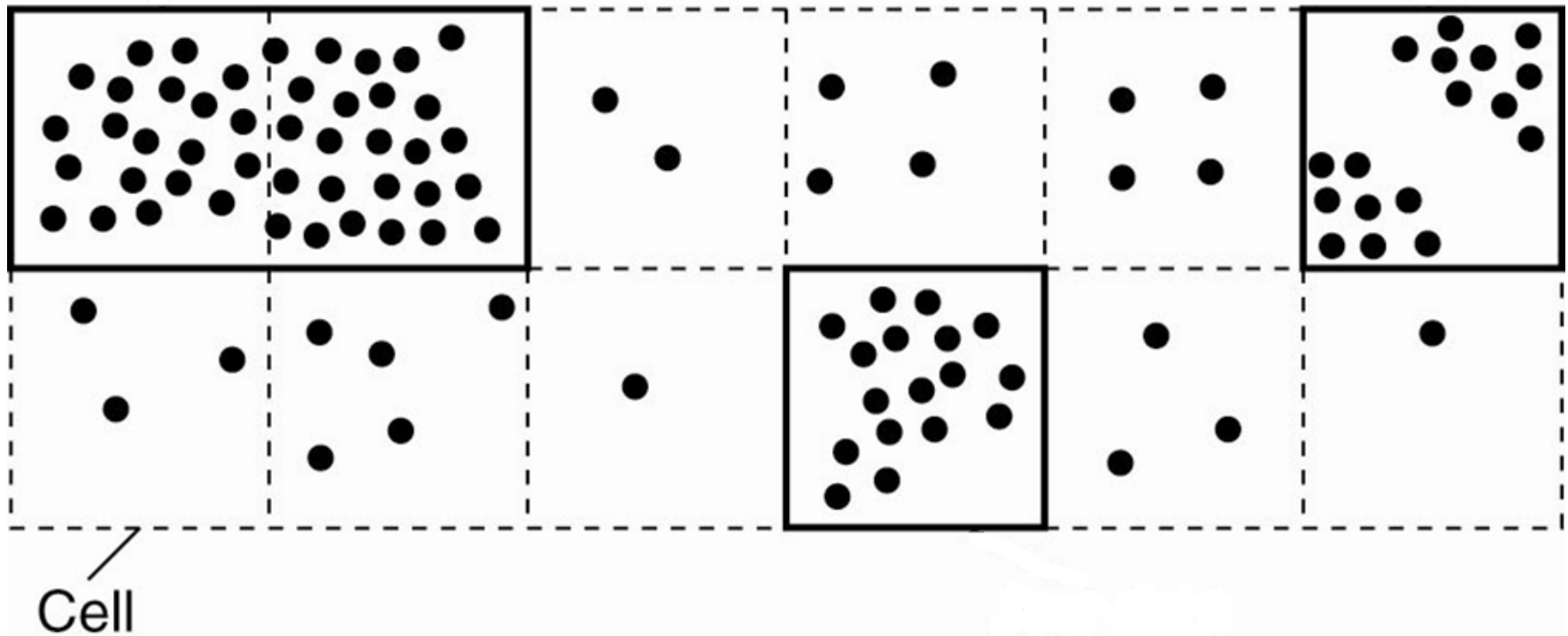


Where should we put it?

# Densest Cells

---

---



---

---

I chose where to put the servers. What content do they store?

# Content Replication

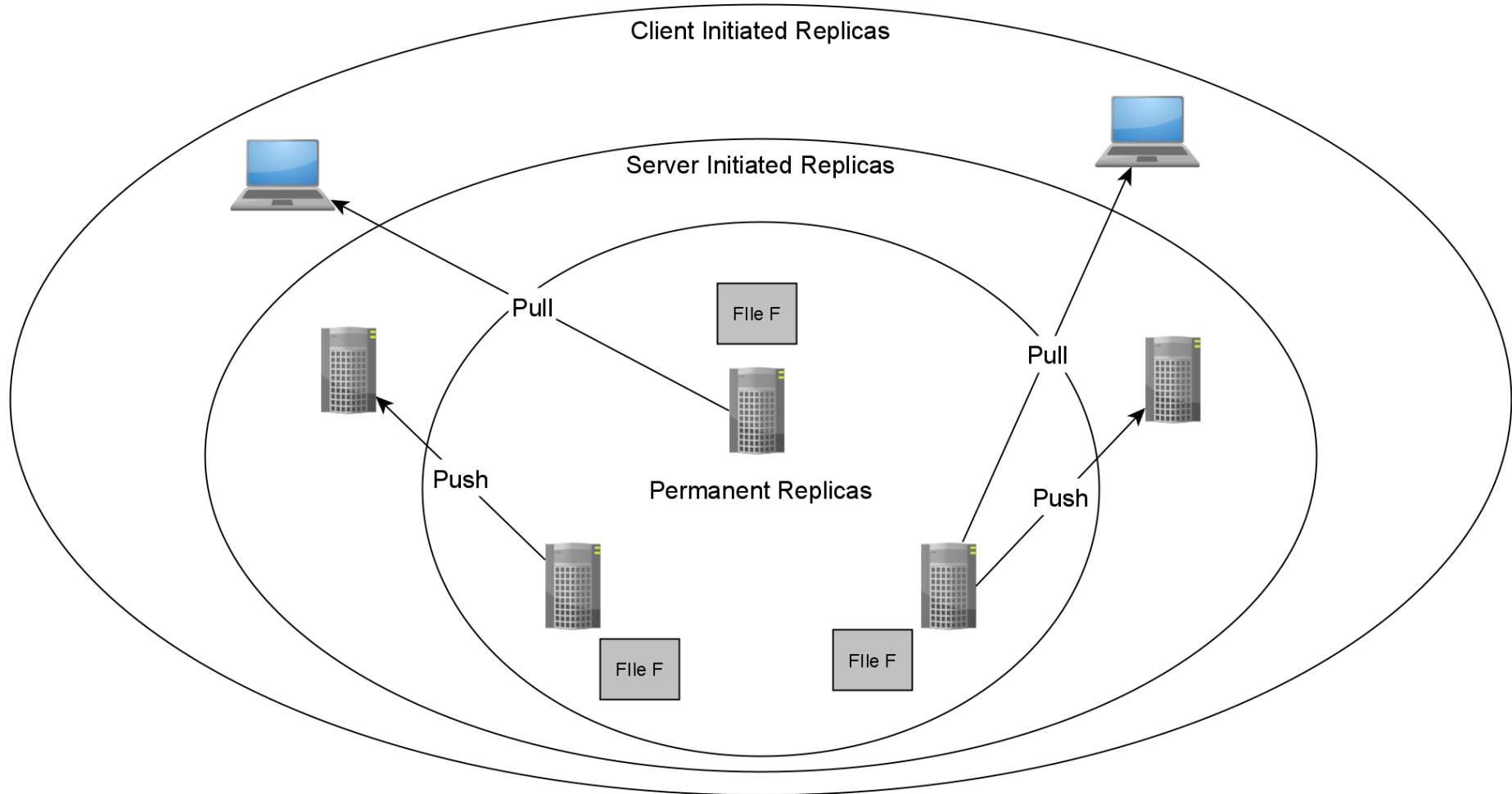
---

**Model:** We consider objects (and don't worry whether they contain just data or code, or both)

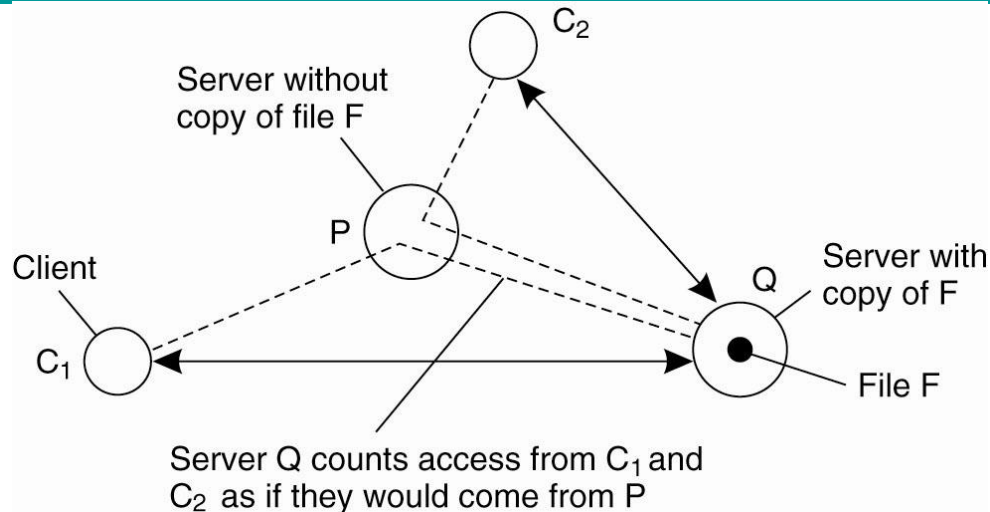
**Distinguish different processes:** A process is capable of hosting a replica of an object or data:

- **Permanent replicas:** Process/machine always having a replica
- **Server-initiated replica:** Process that can dynamically host a replica on request of another server in the data store
- **Client-initiated replica:** Process that can dynamically host a replica on request of a client (**client cache**)

# Content Replication Types



# Server-Initiated Replicas



Keep track of **access counts per file**, aggregated by considering server closest to requesting clients

**When evaluating a file's storage:**

- If the number of accesses **drops below threshold  $D$**   $\Rightarrow$  drop file
- If the number of accesses **exceeds threshold  $R$**   $\Rightarrow$  replicate file
- Between  $D$  and  $R$   $\rightarrow$  You may try to migrate it
  - To the replica which would have gotten **more than half of the requests**
  - The **farthest** replica with **more than  $c$  percent** of the requests

# How do I move content around?

---

**Model:** Consider only a client-server combination:

Propagate only notification/invalidation of update (often used for caches)

Transfer data from one copy to another (distributed databases)

Propagate the update operation to other copies (also called active replication)

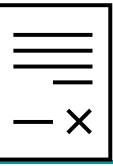
**Observation:** No single approach is the best, but depends highly on available bandwidth and read-to-write ratio at replicas.

# Content Distribution

- **Pushing updates**: server-initiated approach, in which update is propagated regardless whether target asked for it.
- **Pulling updates**: client-initiated approach, in which client requests to be updated.

Issue	Push-based	Pull-based
1:	List of client caches	None
2:	Update (and possibly fetch update)	Poll and update
3:	Immediate (or fetch-update time)	Fetch-update time
<i>1: State at server</i>		
<i>2: Messages to be exchanged</i>		
<i>3: Response time at the client</i>		

# Leasing



**Observation:** We can dynamically switch between pulling and pushing using **leases**: A contract in which the server promises to push updates to the client until the lease expires.

**Tweak:** Make lease expiration time dependent on system's behavior (**adaptive leases**):

- **Age-based leases:** An object that hasn't changed for a long time, will not change in the near future, so provide a long-lasting lease
- **Renewal-frequency based leases:** The more often a client requests a specific object, the longer the expiration time for that client (for that object) will be
- **State-based leases:** The more loaded a server is, the shorter the expiration times become

# Conclusion

---

- Client centric consistency
- Consistency Protocols
- Replica management
  - Content Replication
  - Content Distribution