2110446 - Data Science and Data Engineering

Foundation of Scalable Architecture

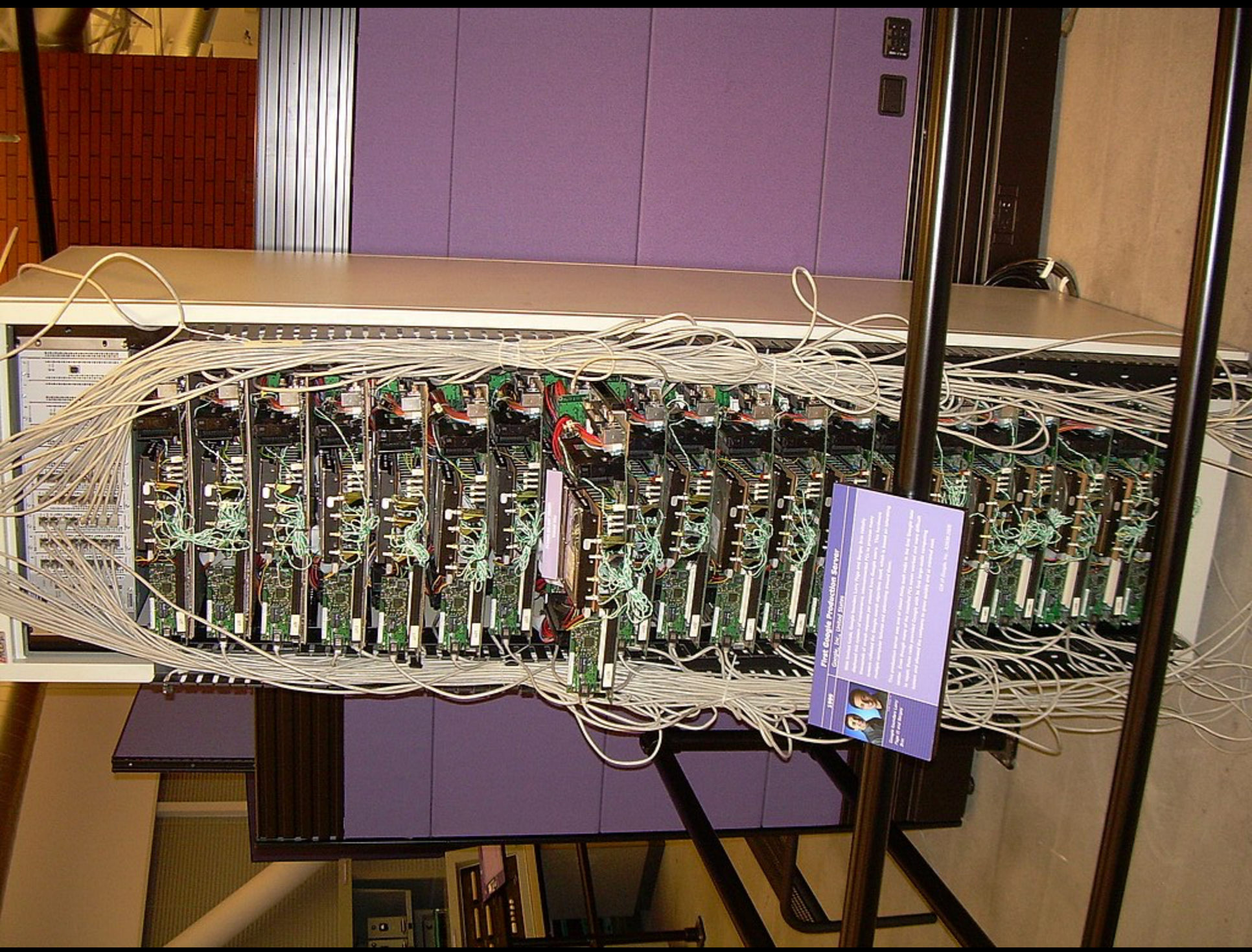
Asst.Prof. Natawut Nupairoj, Ph.D.
Department of Computer Engineering
Chulalongkorn University
natawut.n@chula.ac.th

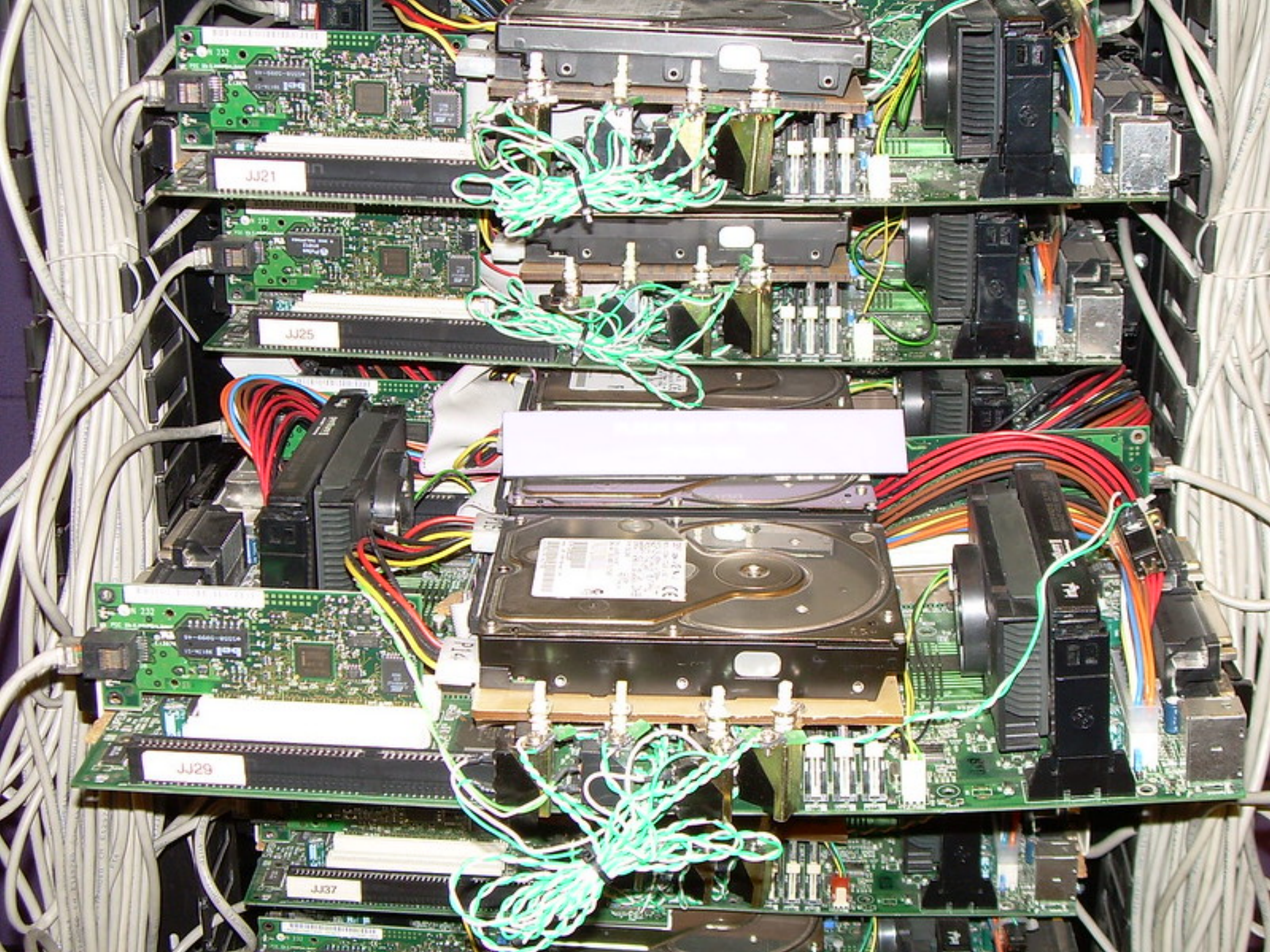
CHULA ENGINEERING
Foundation toward Innovation

Scale Up vs Scale Out

Scale Up

Scale Out


First Google Production Server
1999
Google, Inc., United States


JJ21
JJ25
JJ29
JJ37

Nature of Big Data Architecture

- Base on distributed and parallel processing paradigm
- Hardware or software components located at networked computers communicate and coordinate their actions only by passing messages
- This leads to
  - Concurrency
  - No global clock
  - Independent failure

Concurrency

- Large number of resources (or nodes) are working together
  - Tencent - from 8,800 servers in 2014 to > 30,000 servers in 2017
  - Data is usually replicated into multiple copies to improve performance and availability
- Coordination and agreement are required
  - ensure pace of execution - everyone must reach at some execution point before continue
  - handle shared resources - data, devices
  - maintain data consistency

No Global Clock

- Coordination of programs' actions depends on the shared idea of time, but computers in a network cannot accurately synchronize their clocks

- Each computer has its own internal clock; there is no single global clock for every computer

- Thus, no two computers running independently can have the same clock values

- How can we know the order of executions?

  - This is important for guaranteeing correctness e.g transaction

Independent Failure

- Each component of the system (e.g. network, computer, program) can fail independently, leaving the others still running

- Google's studies showed 3% of hard disks will be failed after 3 months and up to 8% after 2 years
  - for 30,000 servers, 900 servers will experience hard disk failures within 3 months

- How can we mask (to hide or make it less severe) these failures?
  - System can continue executing the jobs correctly even when failures occur

Ordering in Distributed Systems

- Execution of distributed systems usually bases on series of events

- An event is the occurrence of a single action that a process carries out as it executes

- Since we cannot synchronize clocks perfectly, we cannot in general use physical time to find out the ordering of events that occur at different computers

- We usually rely on ordering of events, rather than exact timestamp of occurrence e.g. we are certain that A is executed before B even if A and B are on different nodes

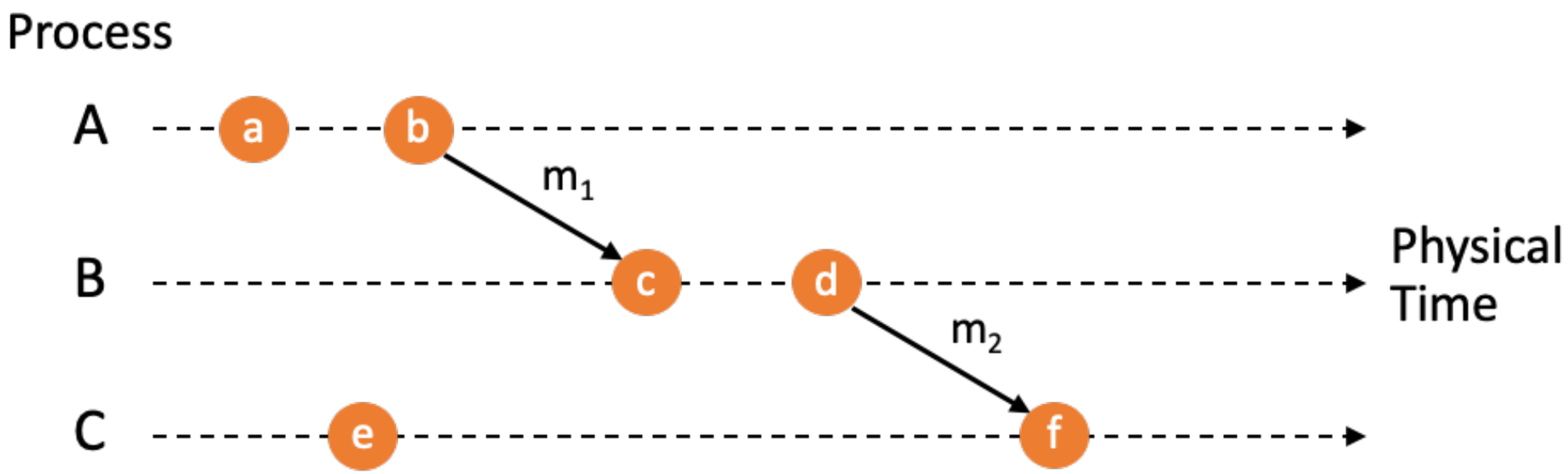
Ordering Models

- Total ordering
  - Each event can be placed in a specific order, which is based on what time they occurred
- Partial ordering
  - We do not know the exact order of every event, just the order of certain events that rely on one another
  - Causal ordering is a special case of partial ordering based on Happened-Before relation

Happened-Before Relation

- A partial order of events that reflects a flow of information
  - HB1: for any events in process A, if e is executed before e' then $e \rightarrow e'$ (e is happened before e')
  - HB2: for any message m, $send(m) \rightarrow recv(m)$
  - HB3: if $e \rightarrow e'$ and $e' \rightarrow e''$ then $e \rightarrow e''$

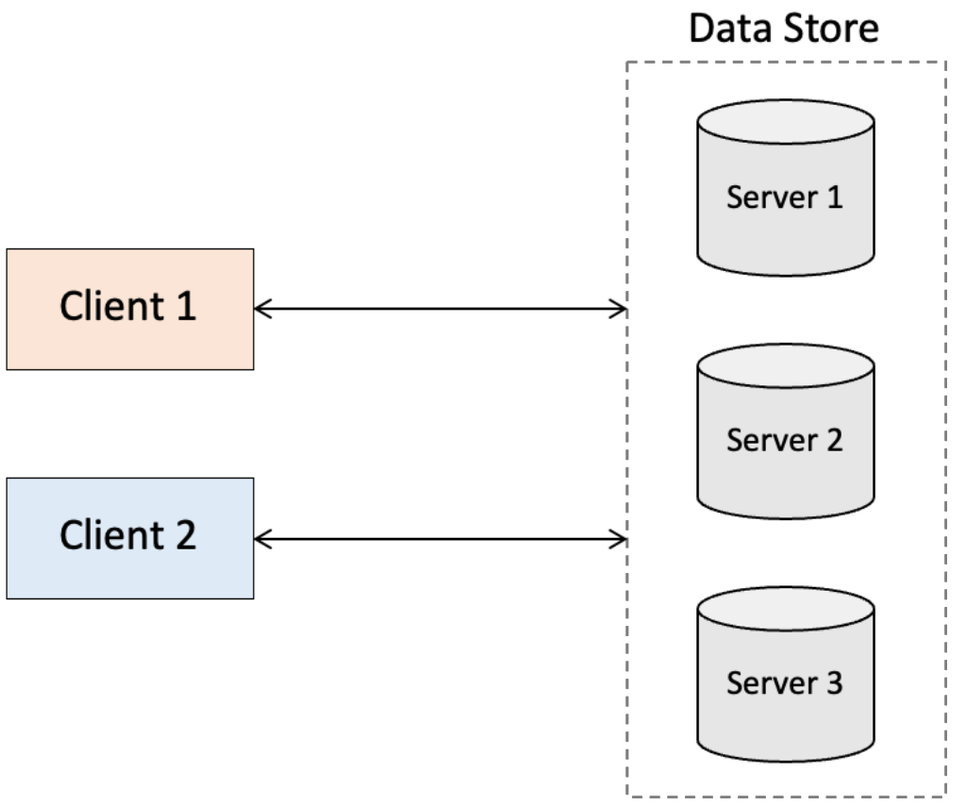
Causal Ordering

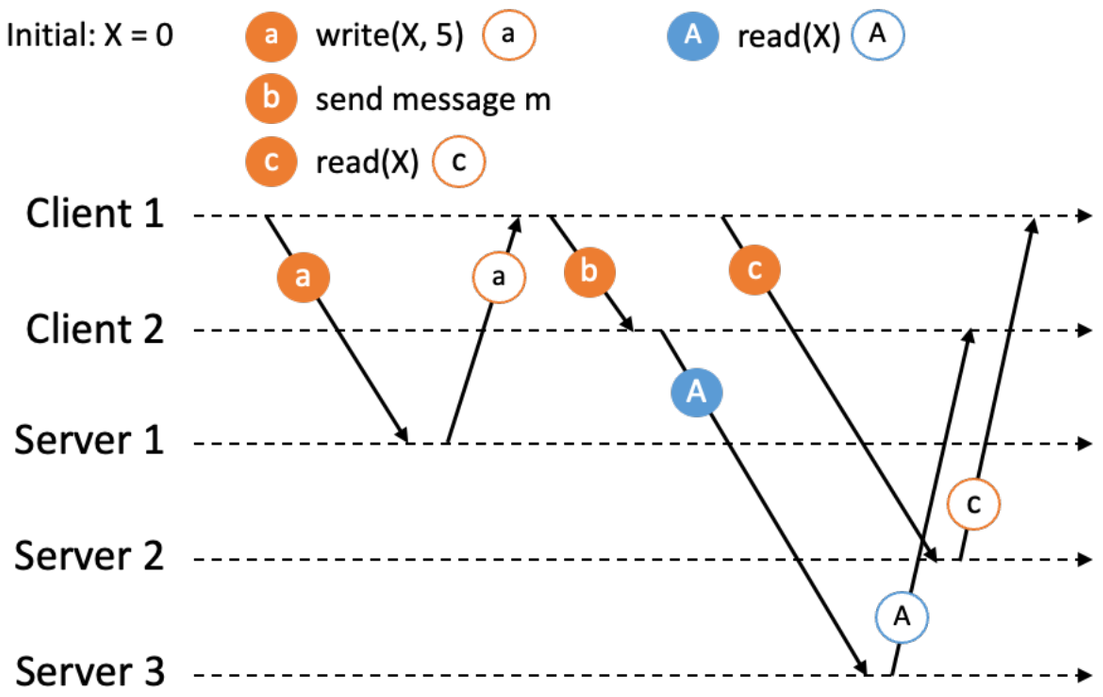


From HB1: $a \rightarrow b$ (at A) $\quad c \rightarrow d$ (at B) $\quad e \rightarrow f$ (at C)
From HB2: $b \rightarrow c$ (at m_1) $\quad d \rightarrow f$ (at m_2)
From HB3: $a \rightarrow c \quad b \rightarrow d \quad \ldots \quad a \rightarrow f$

Not all events are related by Happened-Before relation
Consider a and e (different processes and no chain of messages to relate them), they are not related by $\rightarrow$; they are said to be concurrent; write as a || e

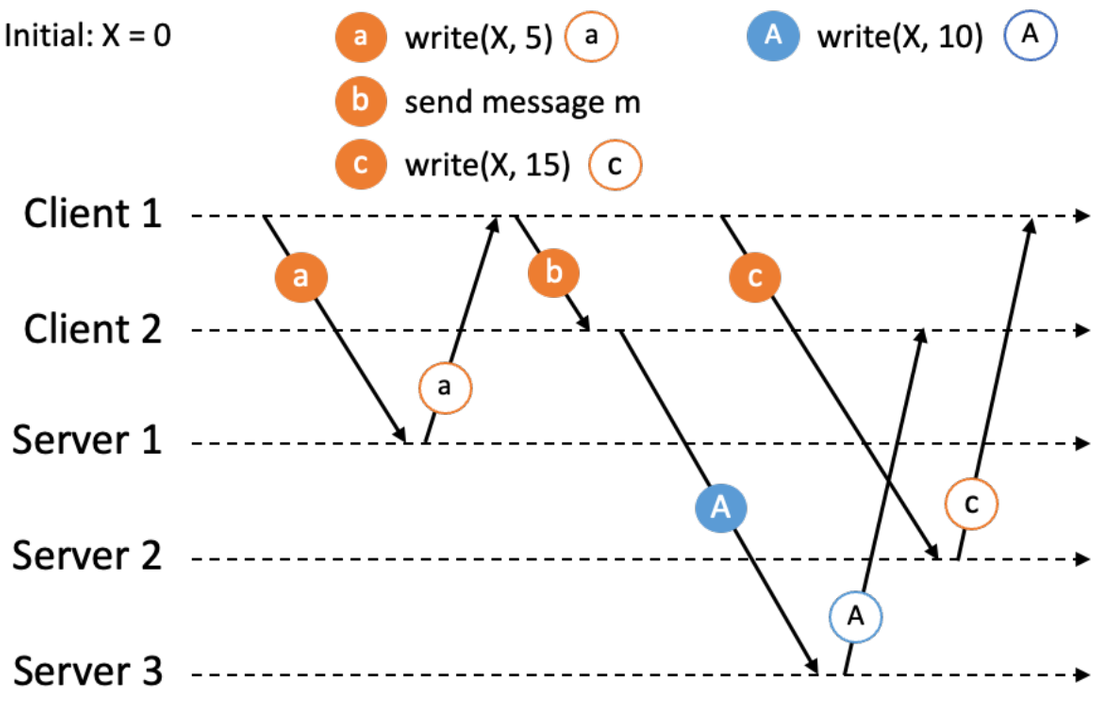
Ordering and Data Store





- Suppose a client can send a request to any server and the system guarantees ordering
- For FIFO ordering: as long as execute (c) after (a) สมมติว่า 3 server เป็นเครื่องเดียวกัน ลำดับเหตุการณ์ที่ Client สังเกตได้
- Causal ordering - as long as execute (A) after (a)

Total Ordering

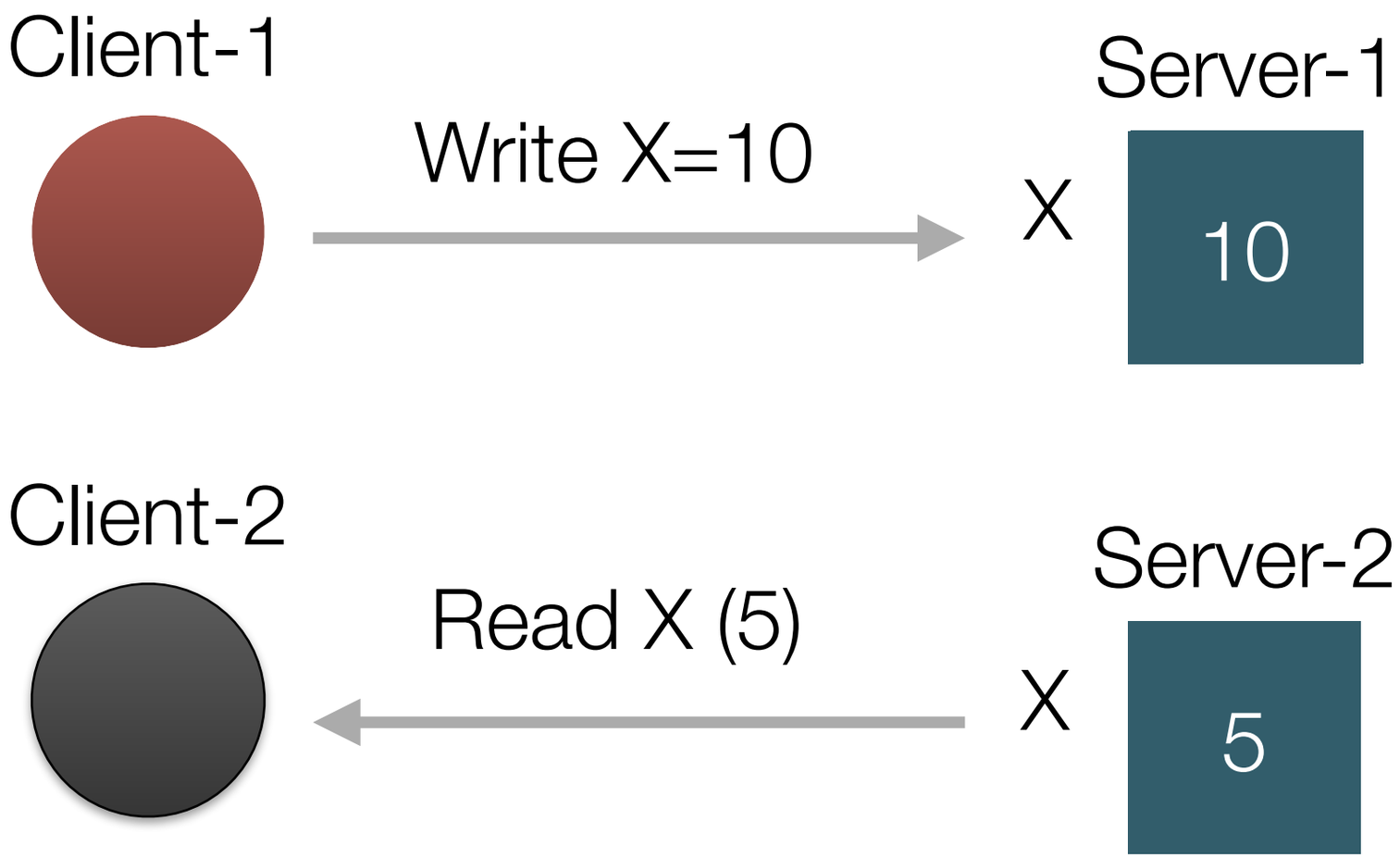


- All servers will execute requests in the same order
- Arbitrary-total - in any order
- FIFO-total - as long as execute (c) after (a)
- Causal-total - as long as execute (A) after (a)

Scalability and Strong Consistency Model

- For scalability, the same data is stored on multiple servers (replica)
- Lead to difficulties to keep all copies in-sync
  - Concurrent write
  - Read and write on different replica
- Even more problems if
  - Data is replicated to lots of servers (hundreds or thousands)
  - Servers are distributed around the world

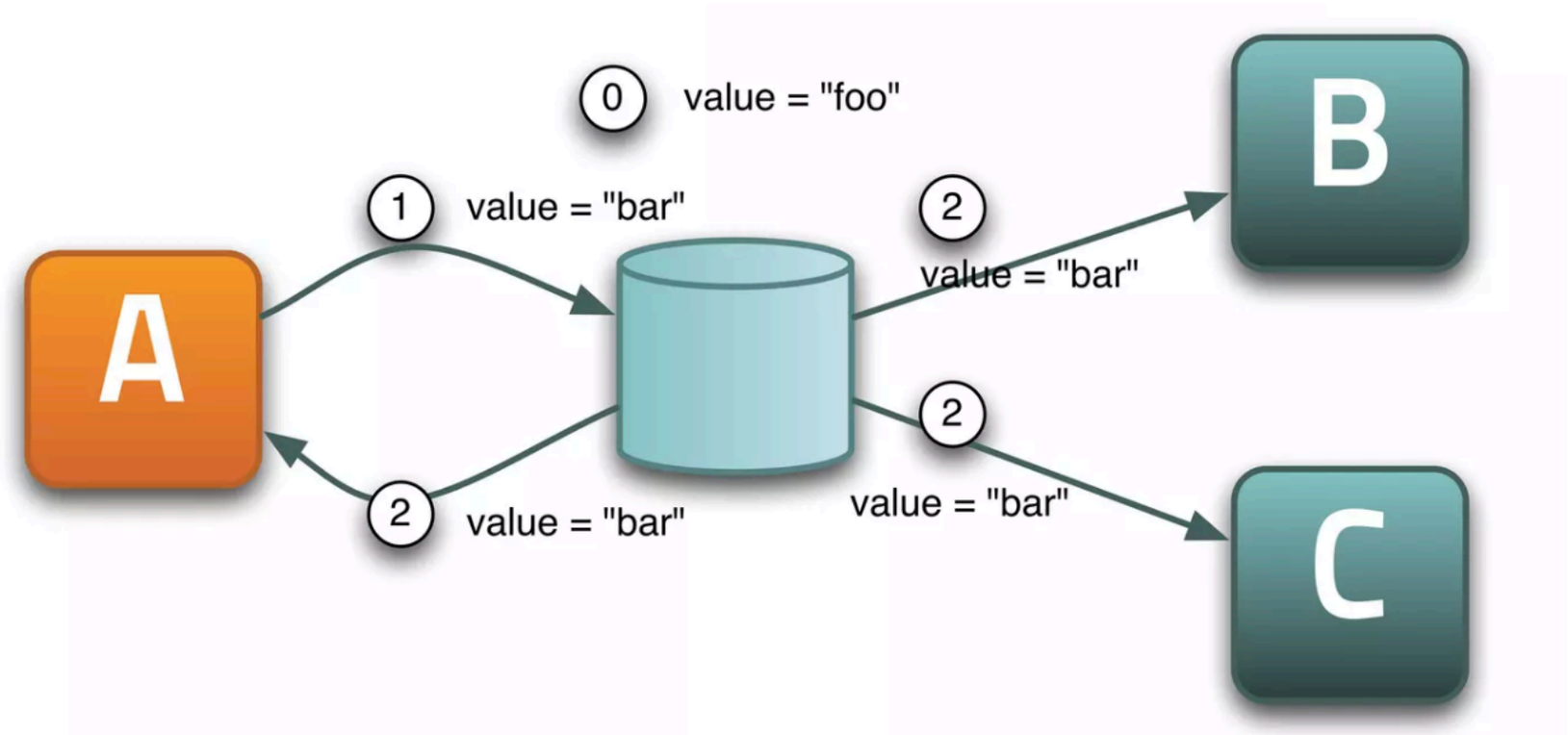
Replication and Data Consistency Problem



Consistency Models (based on Vogels' Paper)

- Strong consistency
  - After update completes, any access will return the updated value
  - High costs for large scale system
- Weak consistency
  - Certain conditions must be met before the consistency is guaranteed
  - Time between update completion and guaranteed consistency is called inconsistency window

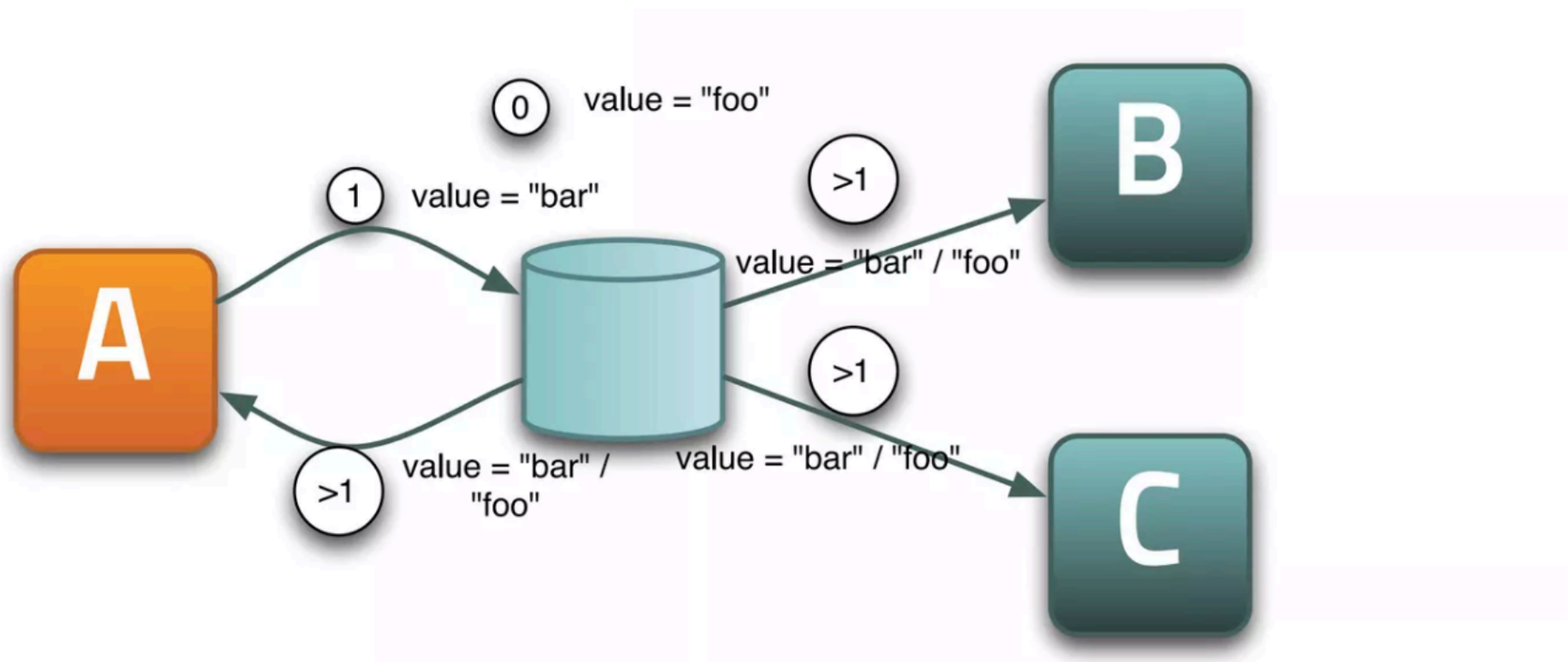
Strong Consistency



After the update, any subsequent access will return the updated value.

Source: https://www.slideshare.net/springerw/eventually-consistent

Weak Consistency



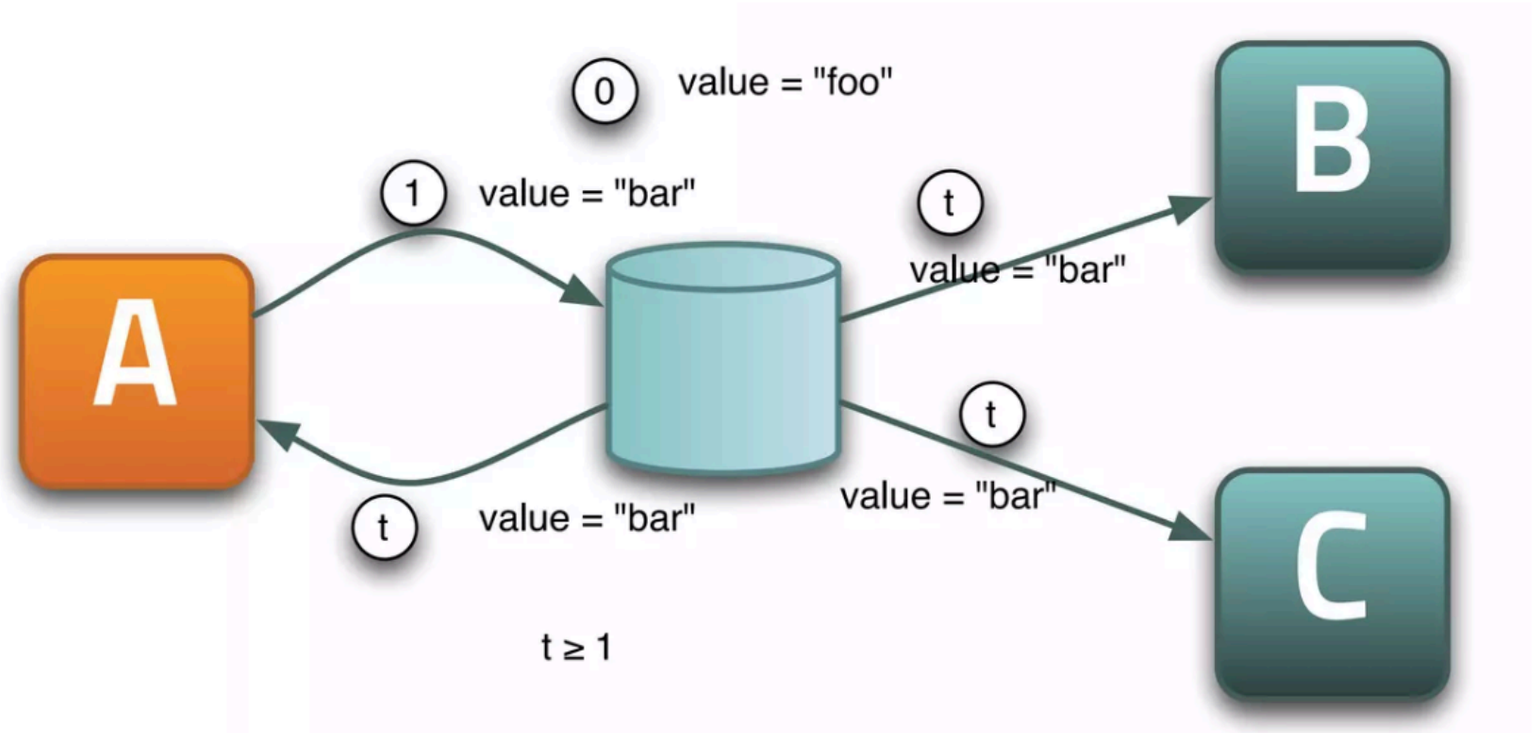
The system does not guarantee that at any given point in the future subsequent access will return the updated value

Source: https://www.slideshare.net/springerw/eventually-consistent

Eventual Consistency

- Specific forms of weak consistency - if no new update are made to the object, eventually all accesses will return the last updated value
  - Causal consistency (A updates data, A informs B, B will always get updated value after that, also true for C if B informs C after B being informed)
  - Read-your-write consistency (A updates data, A will always get updated value after that — special case of causal consistency)
  - Others (reading assignments)

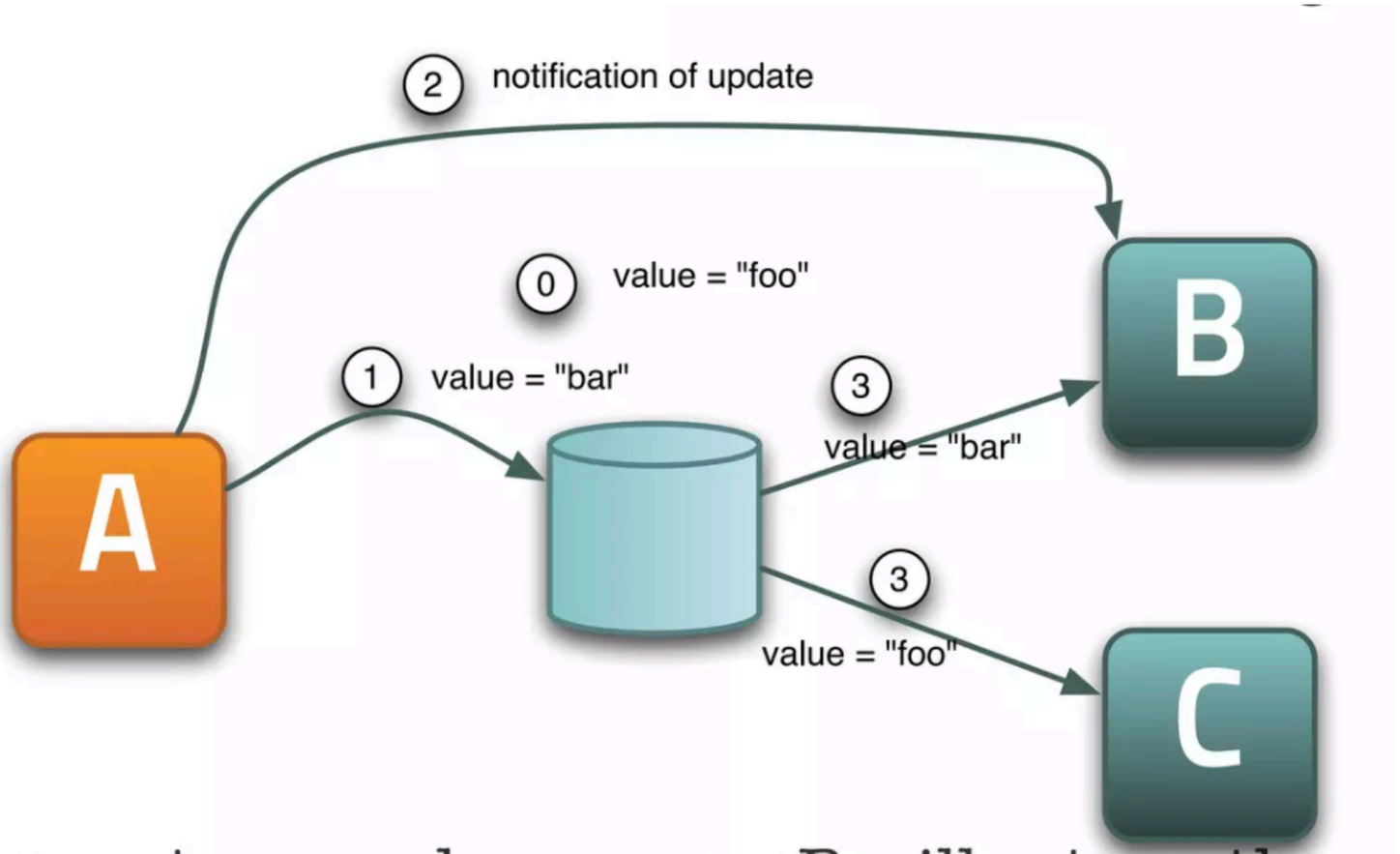
Eventual Consistency



If no updates are made to the object, eventually all accesses will return the last updated value.

Source: https://www.slideshare.net/springerw/eventually-consistent

Causal Consistency

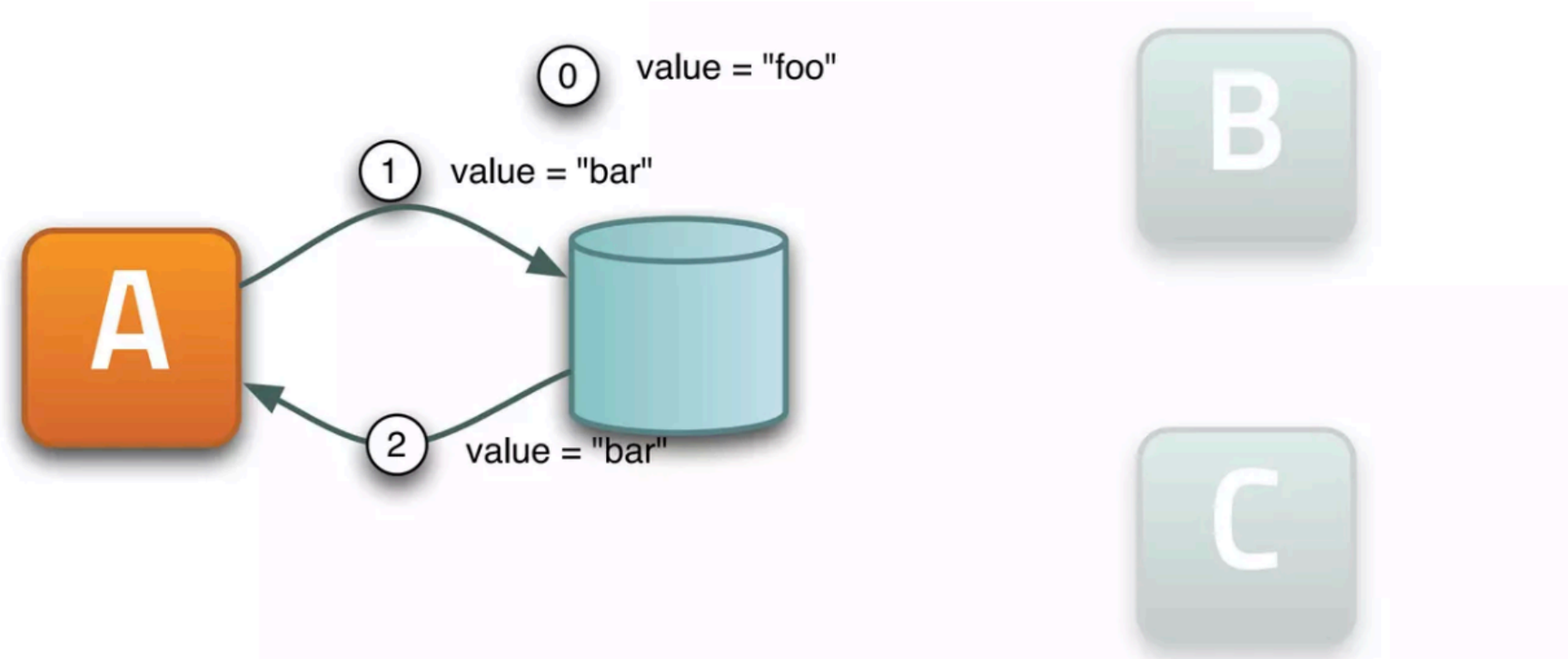


Subsequent access by process B will return the updated value, and a write is guaranteed to supersede the earlier write.

Source: https://www.slideshare.net/springerw/eventually-consistent

Read-Your-Write Consistency

(FIFO)



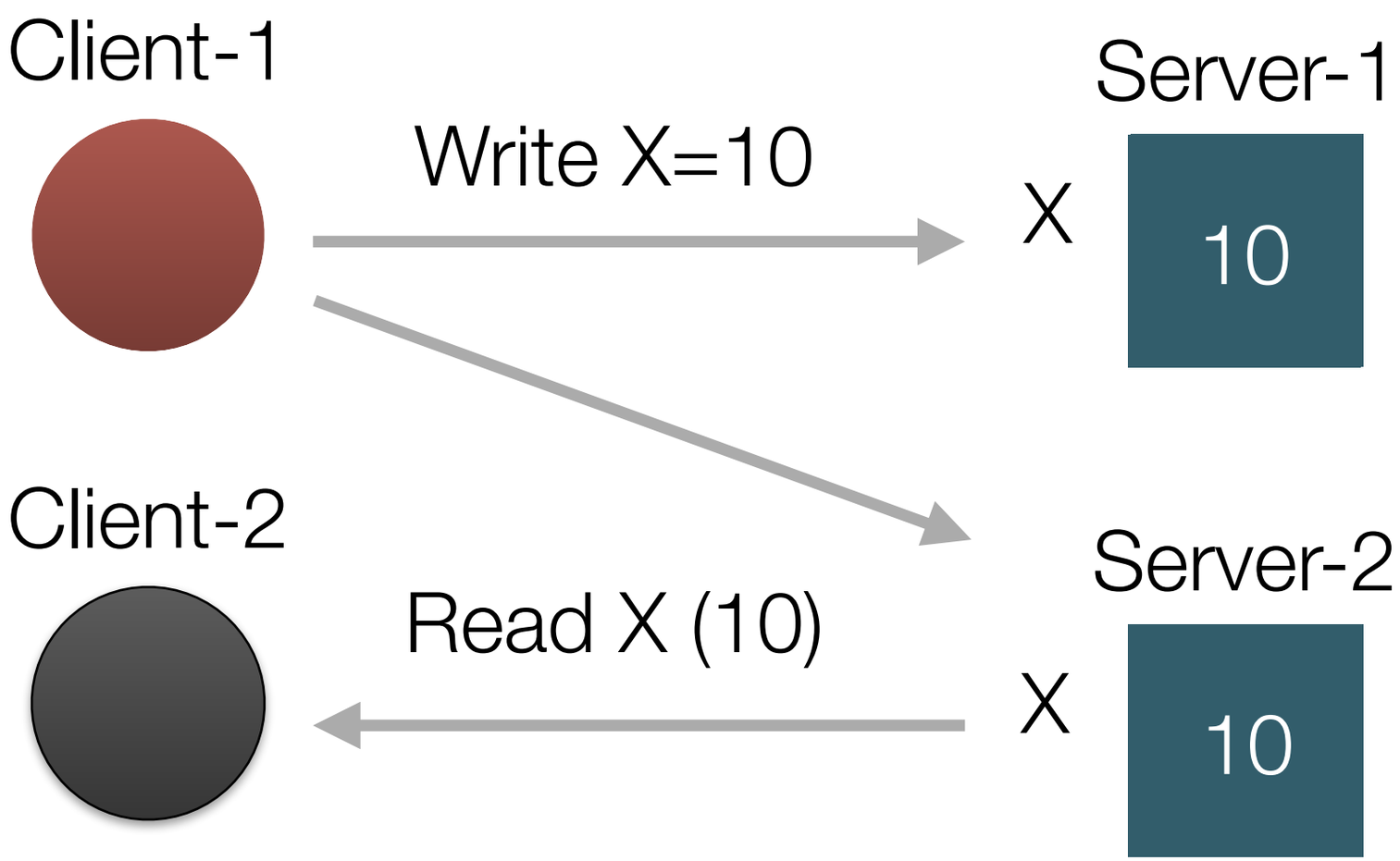
Process A, after updating a data item always access the updated value and never sees an older value

Source: https://www.slideshare.net/springerw/eventually-consistent

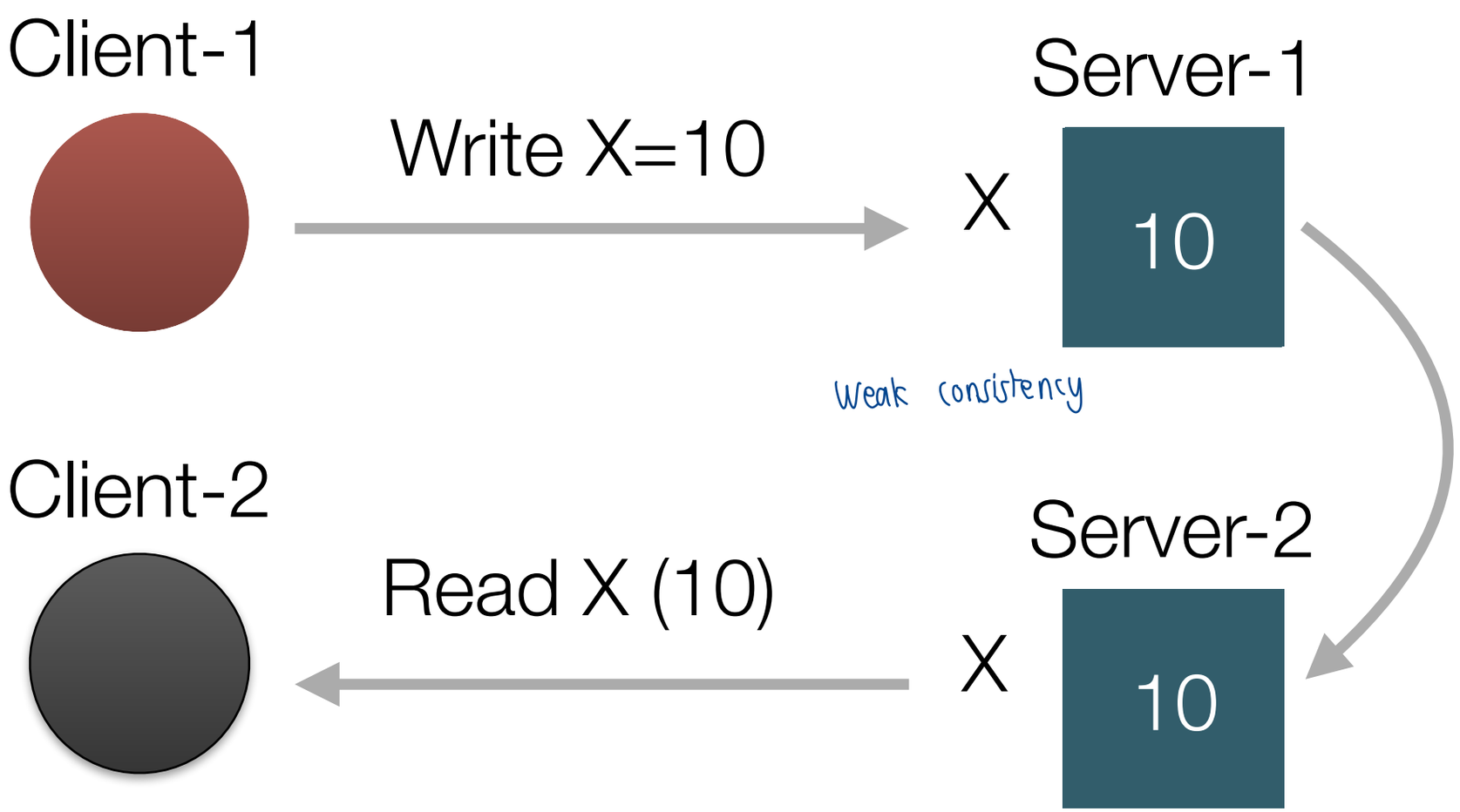
Providing Server-Side Consistency

- Read-only = no synchronization (HDFS)
- Handle consistency
  - Read and Write from multiple servers
    - Strong consistency (R + W > N)
      - Limit number of replications, limit synchronization time
    - Eventual data consistency (R + W < N)
      - Let it inconsistent and resolve it later

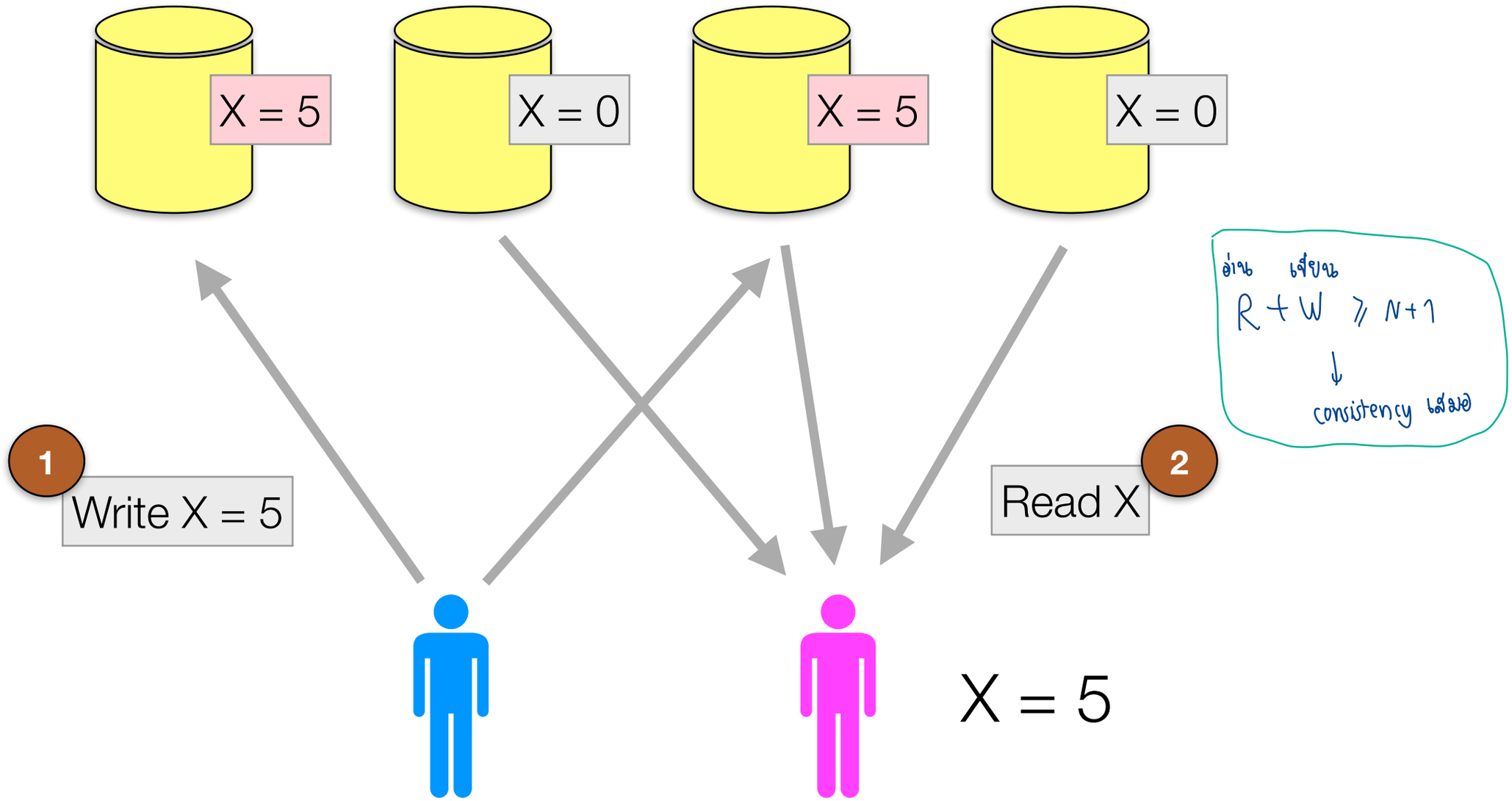
Providing Strong Consistency - Active Replication (W = N, R = 1, R+W > N)



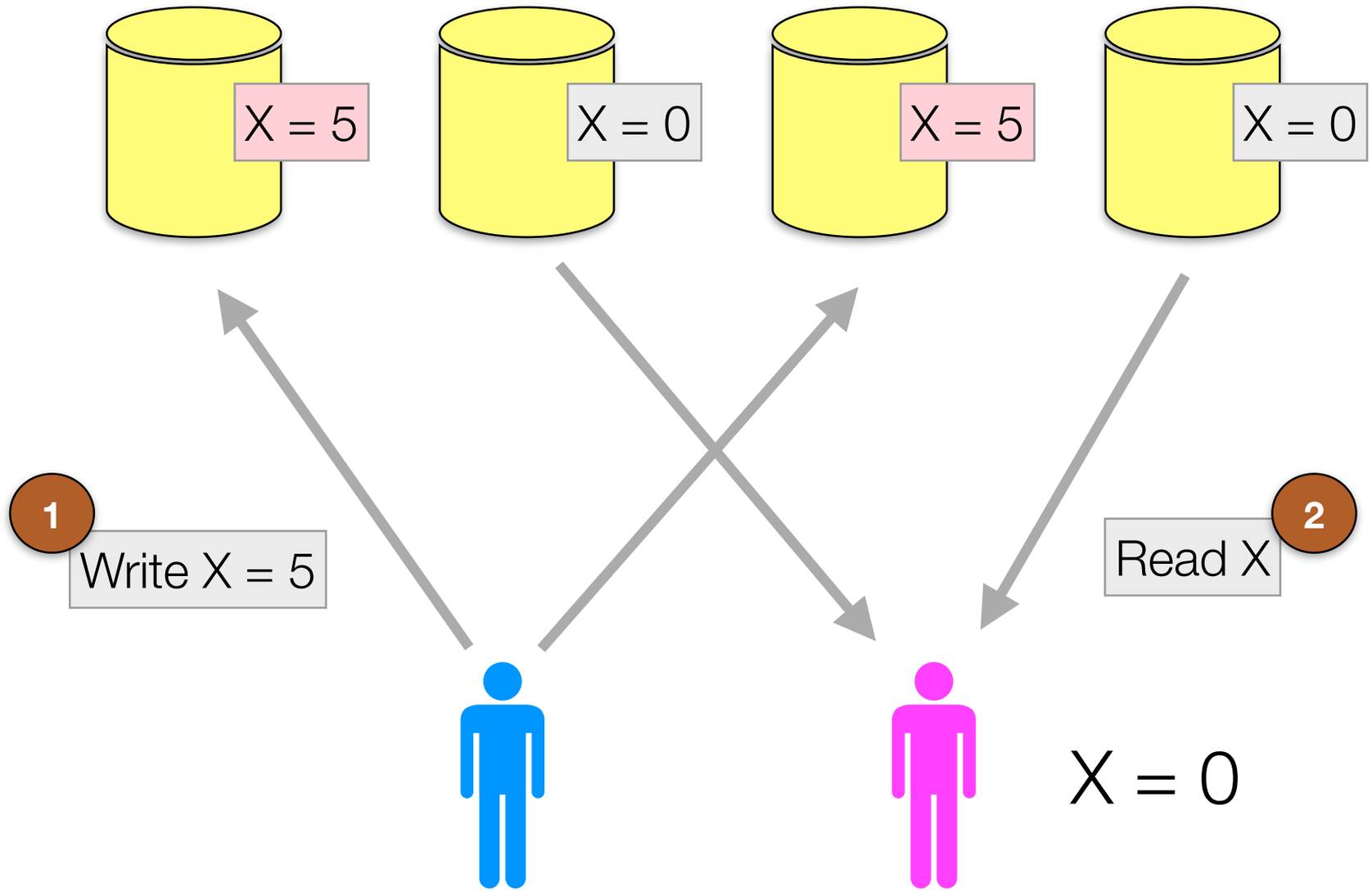
Providing Weak Consistency - Passive Replication (W = 1, R = 1, R+W <= N)



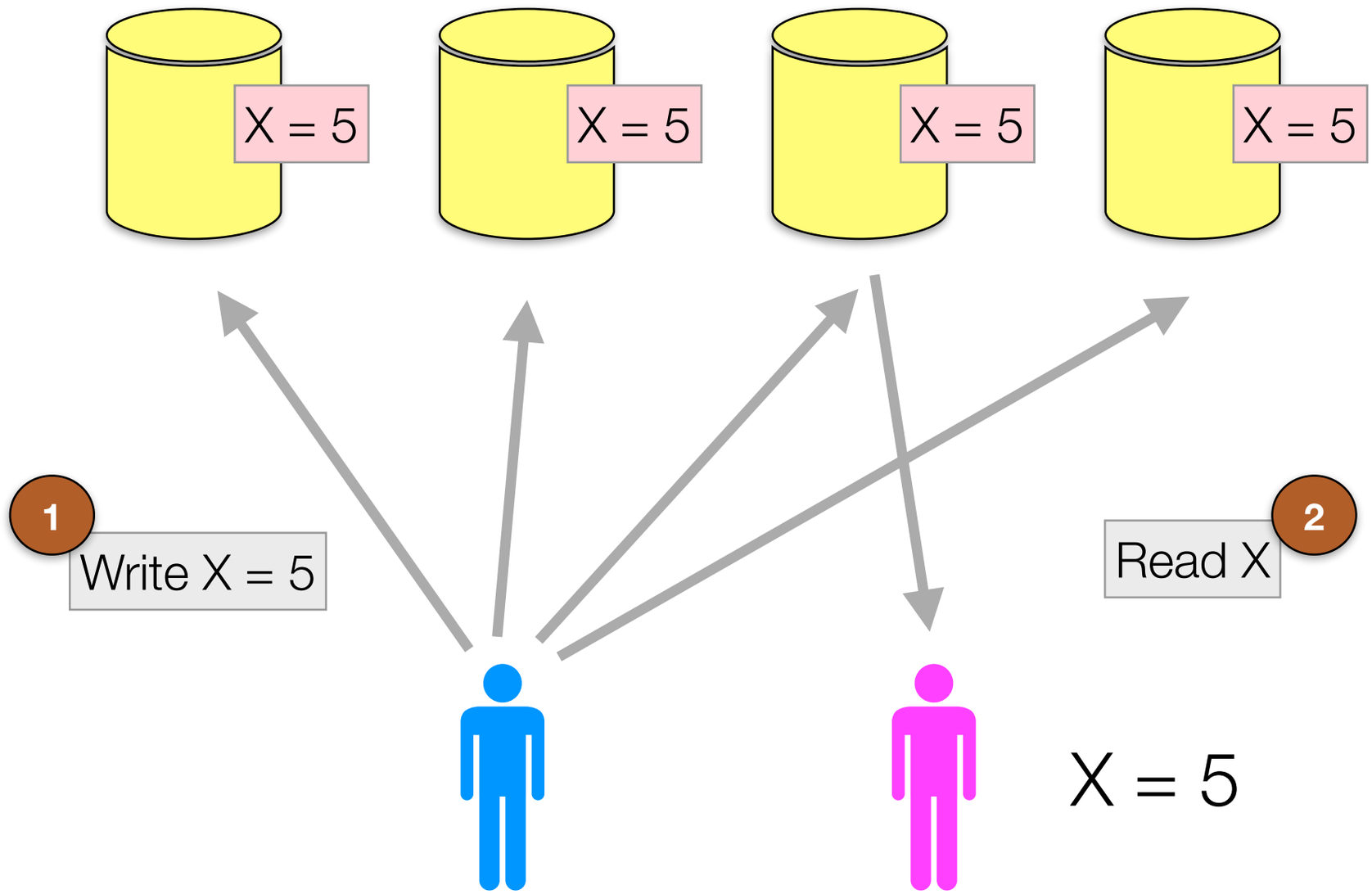
Consistency and Latency Trade-off
N = 4, R = 3, W = 2 (Consistency Guaranteed)



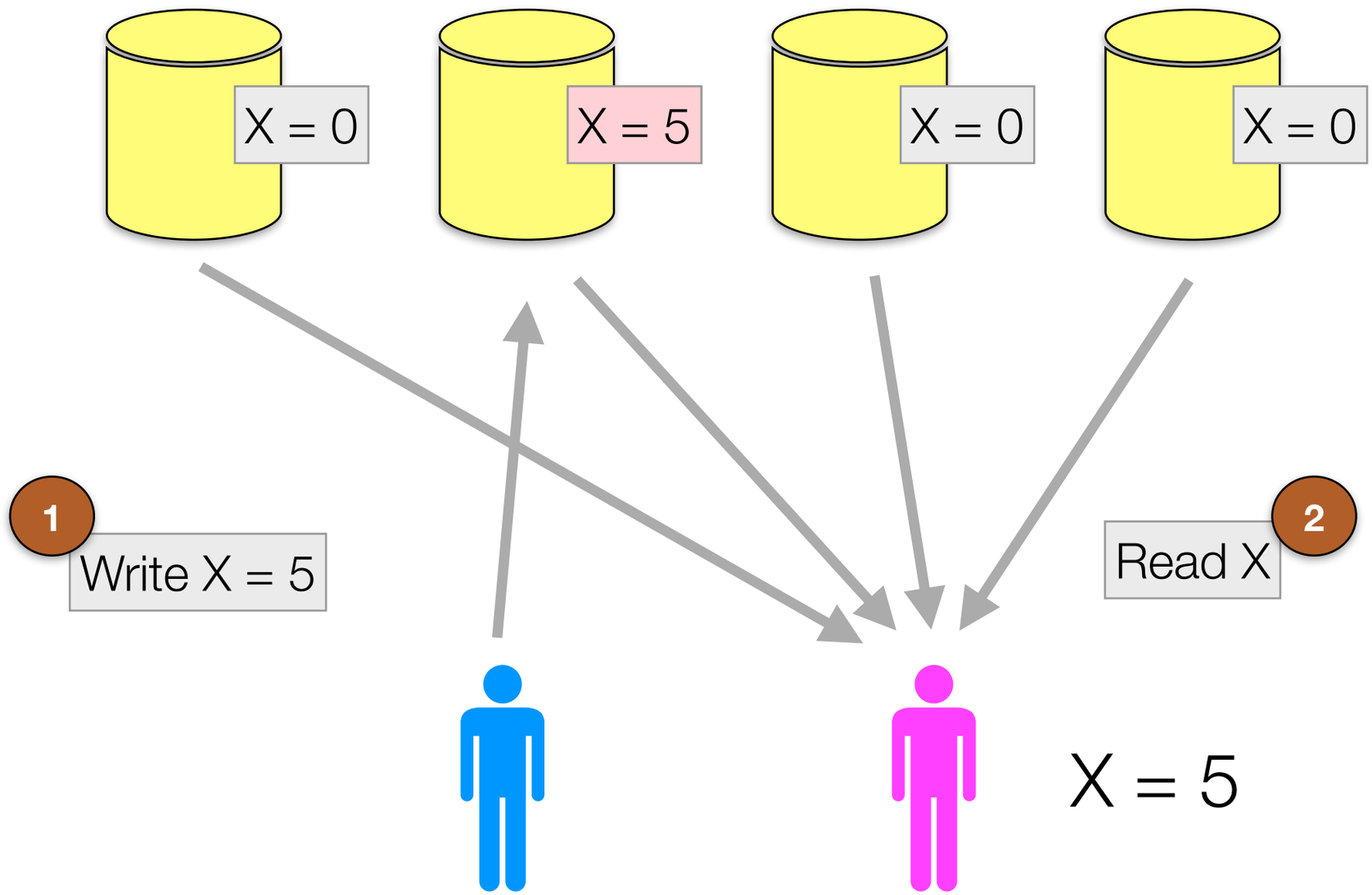
Consistency and Latency Trade-off
N = 4, R = 2, W = 2 (Consistency not Guaranteed)



Consistency and Latency Trade-off
N = 4, R = 1, W = 4 (Fast Read, Slow Write)



Consistency and Latency Trade-off
N = 4, R = 4, W = 1 (Fast Write, Slow Read)



References

- V. Joshi, “Ordering Distributed Events”, https://medium.com/baseds/ordering-distributed-events-29c1dd9d1eff

- W. Vogels, "Eventually consistent”, Communications of the ACM, Vol. 52, pp. 40–44, 2009.

- W. Springer, "Eventually consistent”, https://www.slideshare.net/springerw/eventually-consistent