2110625 - Data Science Architecture

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 ขยายเครื่องเดิมให้ใหญ่ขึ้นโดยไม่เปลี่ยน arch

ข้อดี - ง่าย

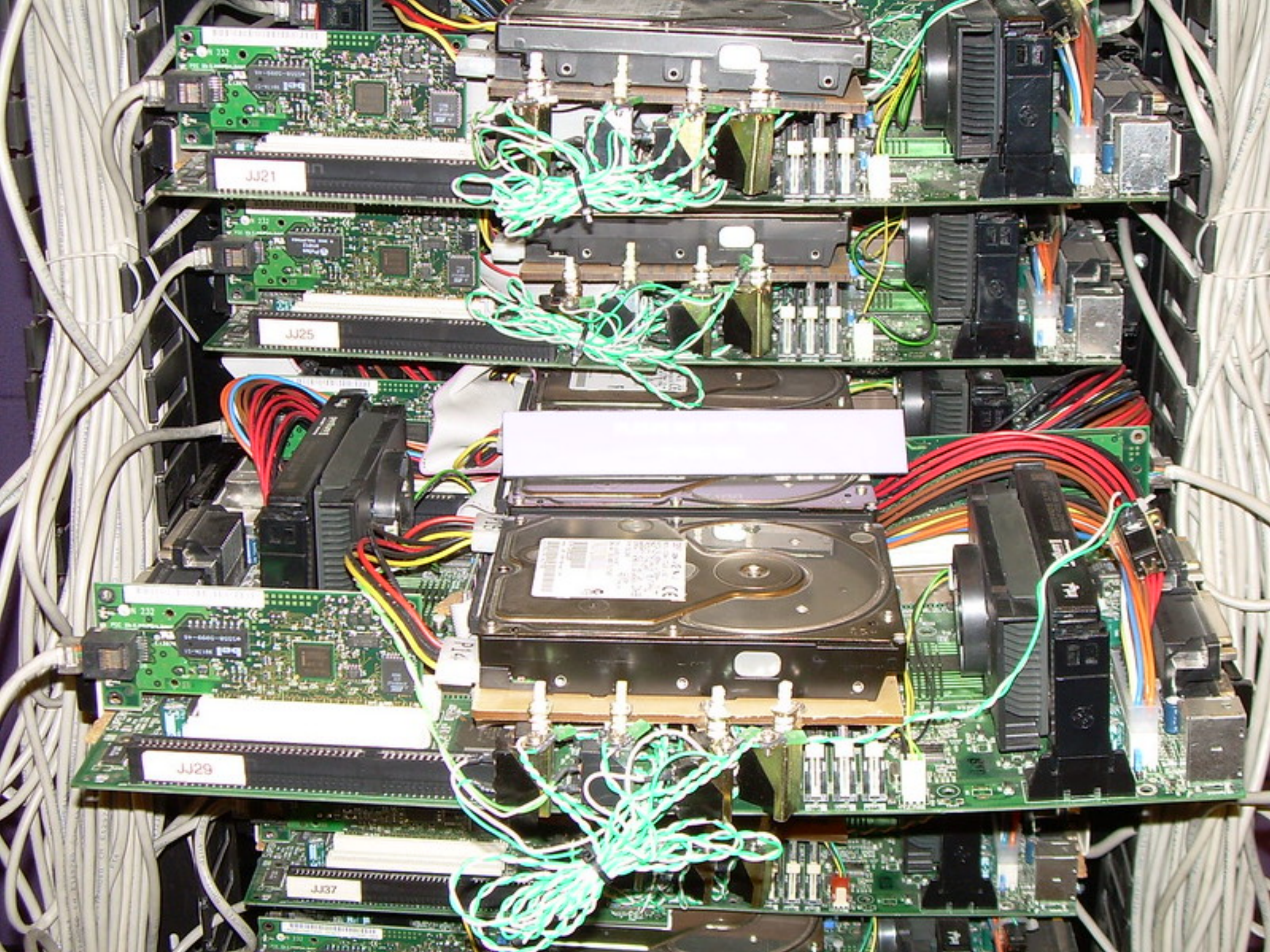
เสีย - network การเข้าออก ส่วนที่เก็บ server เป็นปัญหา I/O bandwidth

Scale Out เพิ่มเครื่องเพิ่ม

Google
Cluster v.1

Reliable สูง
ถ้าอันไหนเจ๊ง
ก็เปลี่ยน HW ใหม่
แล้ว auto sync
ลง

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

โดย data เช่น ข้อมูลจาก censor

สิ่งที่เป็น big data

- This leads to
  - Concurrency
  - No global clock
  - Independent failure

distributed มองที่ function ความสามารถในการทำงานร่วมกัน

parallel มองที่ performance

Concurrency

big data ต้อง scale out เครื่องจำนวนมาก

- 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 replication คือการ duplicate data เพื่อแลก server
- Coordination and agreement are required

ร้องขอข้อมูลจะสามารถขอได้หลายเครื่อง → synchronization

  - ensure pace of execution - everyone must reach at some execution point before continue

มาถึงจุดหนึ่งที่บอกว่าทุกคนทำงานพร้อมกัน / ทำงานเสร็จทุกคน

  - handle shared resources - data, devices
  - maintain data consistency

ปัญหาใหญ่มากของระบบ cluster

No Global Clock ***

เมื่อเวลาผ่านไป clock ทุกเครื่องไม่เท่ากันโอกาสสูงมาก (หลัก nano sec)

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

สัญญาณ clock ไม่ได้ precise เป๊ะๆ ทุกเครื่อง

- 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

อ่านก่อน update หรือ update ก่อนอ่าน

- How can we know the order of executions?
  - This is important for guaranteeing correctness e.g transaction

ถ้าไม่มี global clock จะไม่สามารถการันตีความถูกต้อง

Independent Failure ของเยอะ โอกาสพังเยอะ

- 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 3 เดือน HDD พัง 3% (900 เครื่อง)
  - for 30,000 servers, 900 servers will experience hard disk failures within 3 months 2 ปี HDD พัง 8%

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

Ordering in Distributed Systems

เรียงลำดับ event ที่เกิดขึ้นต่างเครื่องมาเรียงกัน

เครื่องเดียวเรียงได้ เพราะ clock เดียวกัน

- 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

A → B , B → A ? ถึงแม้จะเกิดคนละ node

A และ B อาจทำคนละ node

Ordering Models

- Total ordering เรียงลำดับทุก event ในระบบได้
  - Each event can be placed in a specific order, which is based on what time they occurred
- Partial ordering มักเกิดกับระบบที่ No global clock
  - 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

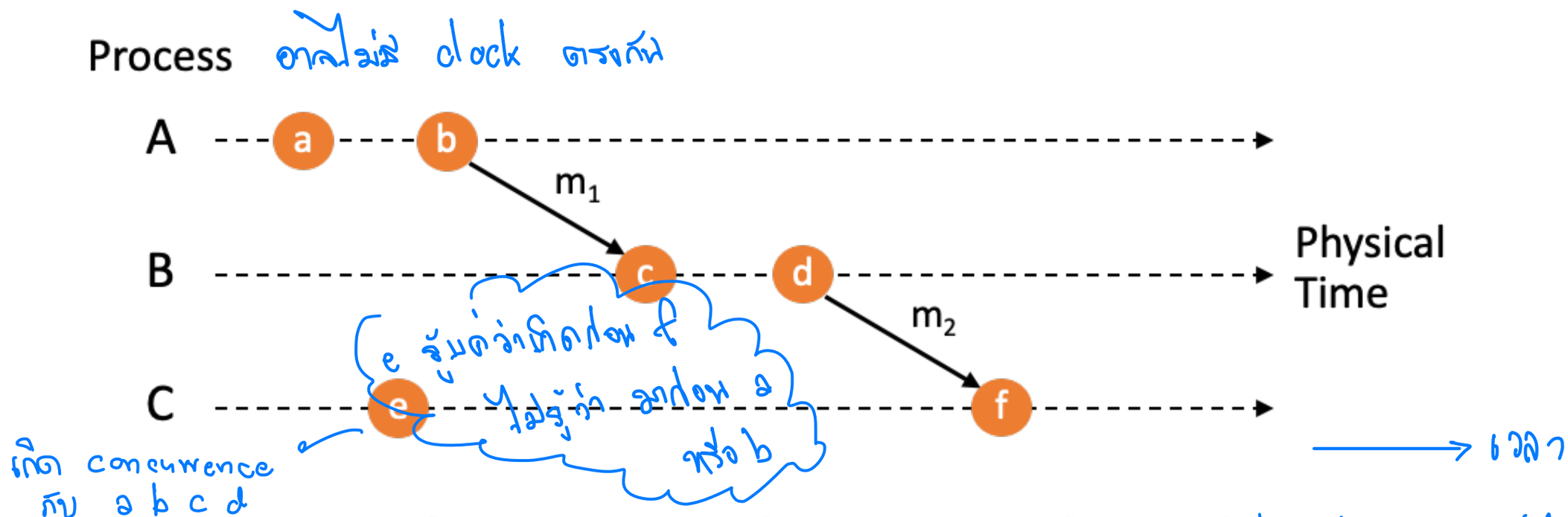
Happened-Before Relation

- 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') e เกิดก่อน e'
  - HB2: for any message m, $send(m) \rightarrow recv(m)$ ถ้ามีการส่งข้อความ m event ส่ง ต้องเกิดก่อนการรับเสมอ
  - HB3: if $e \rightarrow e'$ and $e' \rightarrow e''$ then $e \rightarrow e''$ ถ้า e เกิดก่อน e' และ e' เกิดก่อน e'' ถือว่า e เกิดก่อน e'' (คุณสมบัติถ่ายทอด transitive)

subset
FIFO ⊂ Causal ordering
เกิดเครื่องเดียวกัน

Causal Ordering

จัดตามความรวมโดยใช้ happen before relation



From HB1: $a \rightarrow b$ (at A) $c \rightarrow d$ (at B) $e \rightarrow f$ (at C) เกิดที่ node เดียวกัน อันไหนทำก่อนก็ happen before

From HB2: $b \rightarrow c$ (at m_1) $d \rightarrow f$ (at m_2) จัดลำดับการส่งข้อความได้

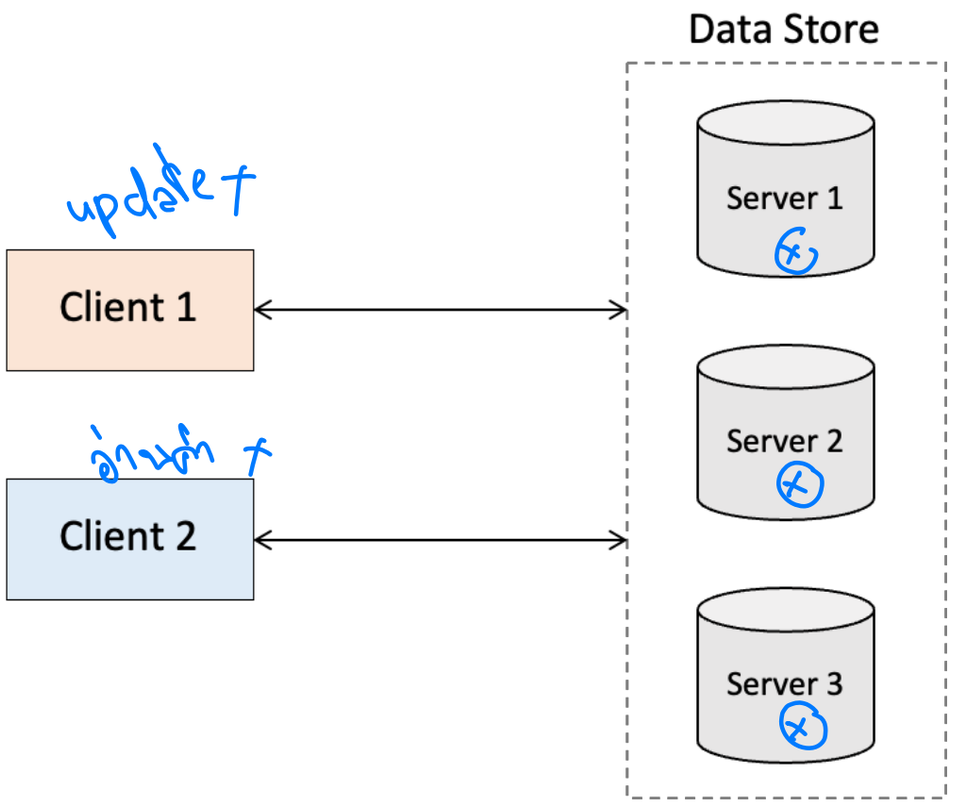
From HB3: $a \rightarrow c$ $b \rightarrow d$ ... $a \rightarrow f$

$A \rightarrow b \wedge b \rightarrow c$ $b \rightarrow c \wedge c \rightarrow d$

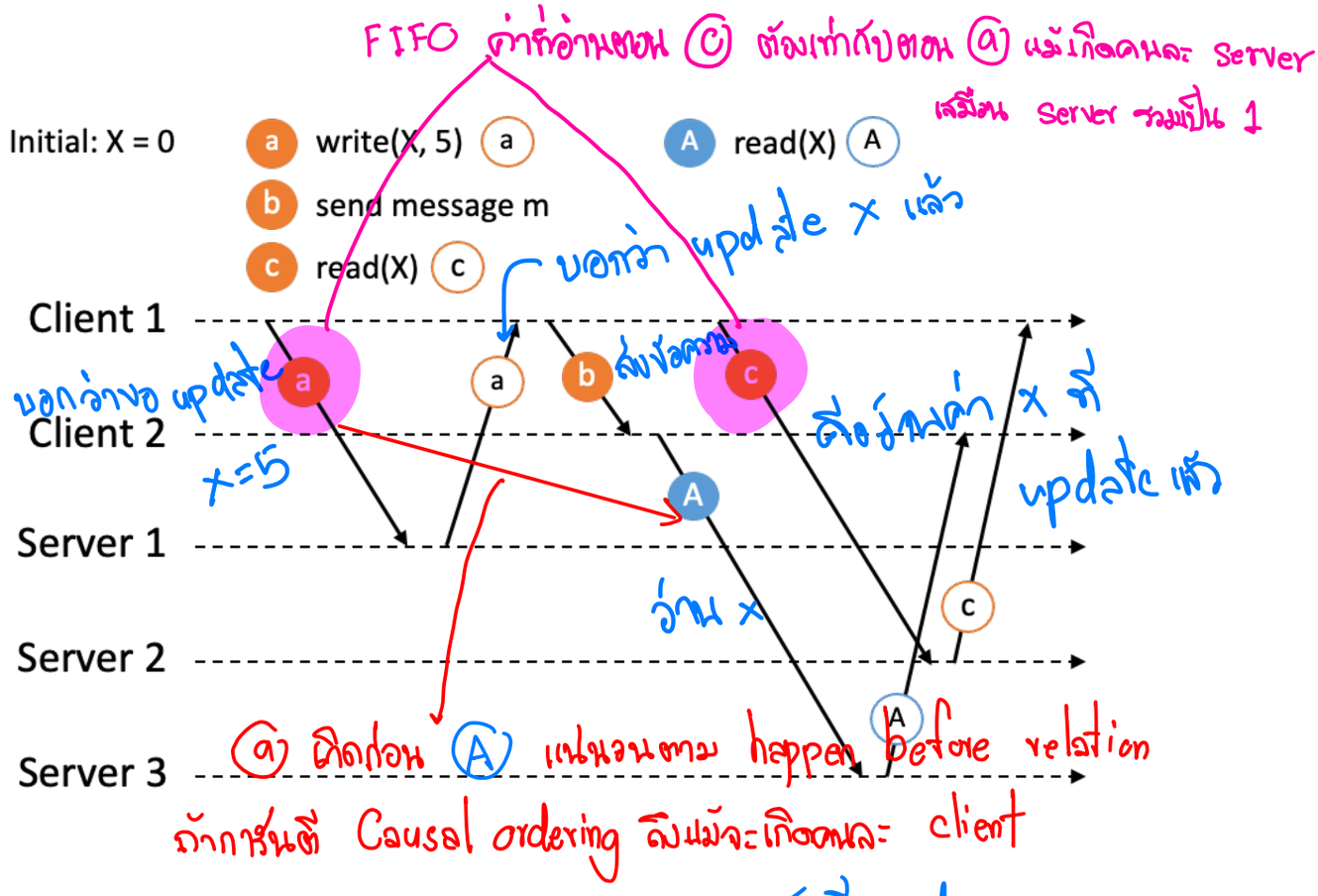
Not all events are related by Happened-Before relation

Consider a and e (different processes and no chain of messages to relate them), they are not related by → ; they are said to be concurrent; write as a || e

Ordering and Data Store



ระบบที่สัญญาว่ารองรับการันตีแบบใด



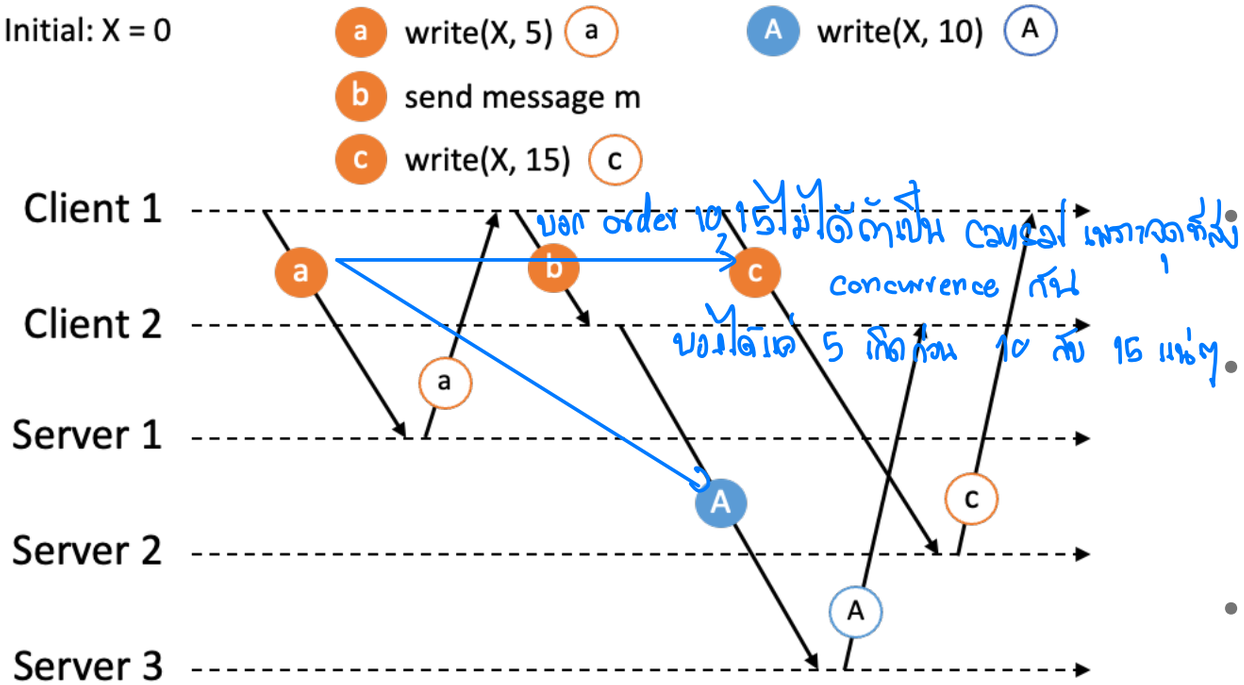
- Suppose a client can send a request to any server and the system guarantees ordering
- For FIFO ordering: as long as execute (c) after (a)
- Causal ordering - as long as execute (A) after (a)

Total Ordering

ที่ server ทุกตัวจะสั่งการ execute คำสั่งที่ถูกส่งเข้ามาด้วยลำดับเดียวกัน

X = 5-10-15, 15-10-5 ได้หมด

ต่อให้จาก update อาจจะก่อนหลังไม่สำคัญ server ทุกตัวสั่งการ execute ด้วยลำดับเดียวกัน



บอก order 10, 15 ไม่ได้ว่าเป็น causal เพราะจุดที่สั่ง concurrence กัน

บอกได้แค่ 5 เกิดก่อน 10 สับ 15 แน่ๆ

- All servers will execute requests in the same order
- Arbitrary-total - in any order

ขึ้นต้องตกลงกันว่าจะจัดลำดับอย่างไร

- FIFO-total - as long as execute (c) after (a)

ลำดับที่ทำงานเหมือน FIFO เกิดการ update 5 ก่อน update 15 แต่ update 10 อยู่ไหนไม่รู้ ไม่เกี่ยวกับ

- Causal-total - as long as execute (A) after (a)

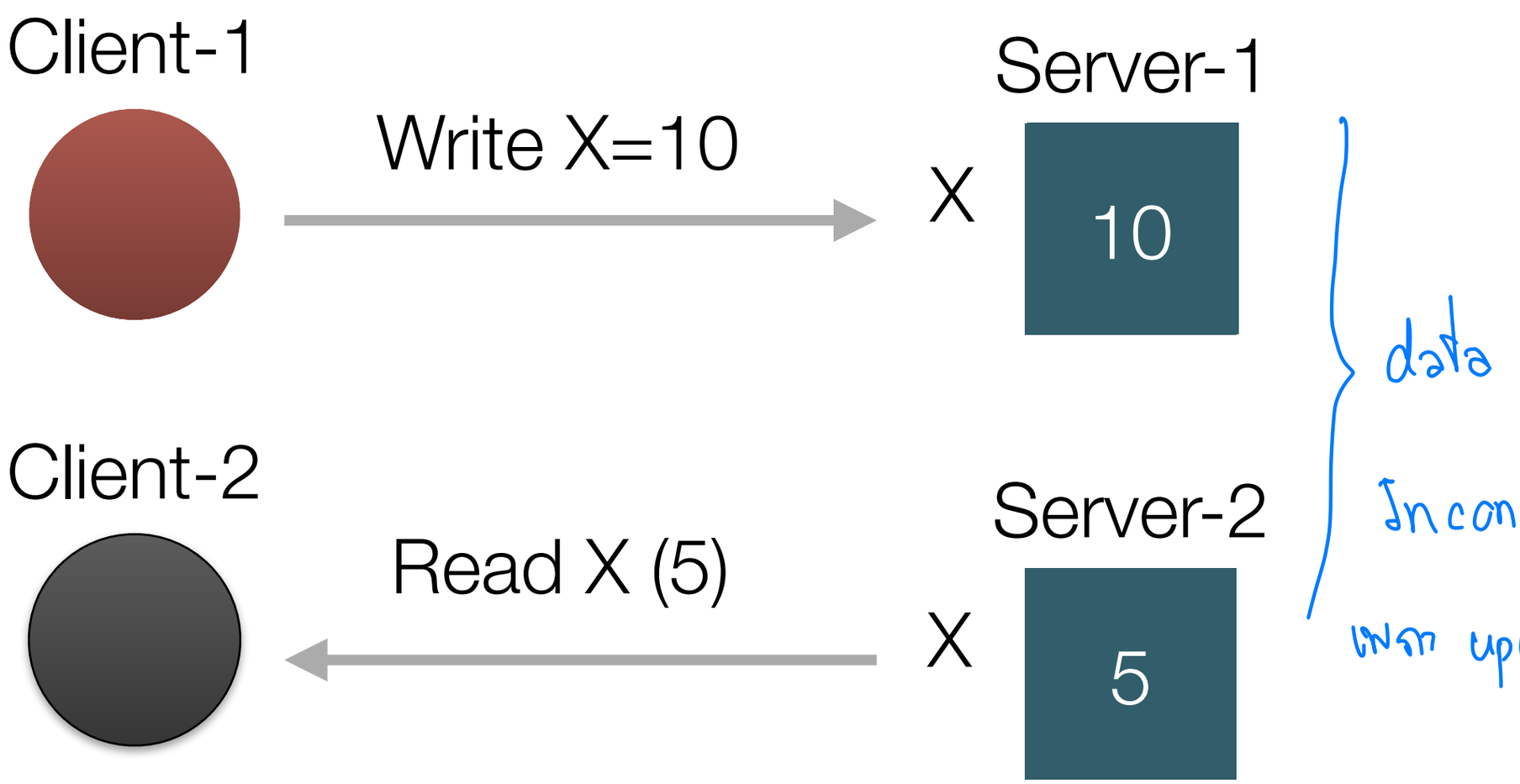
5-15-10, 5-10-15, 10-5-15 ได้หมด

ยกเว้น 15-5-10 กับ 15-10-5

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 การ done หลาย server เกิดปัญหาตรงนี้
  - Concurrent write
  - Read and write on different replica R/W คนละ server
- Even more problems if
  - Data is replicated to lots of servers (hundreds or thousands)
  - Servers are distributed around the world

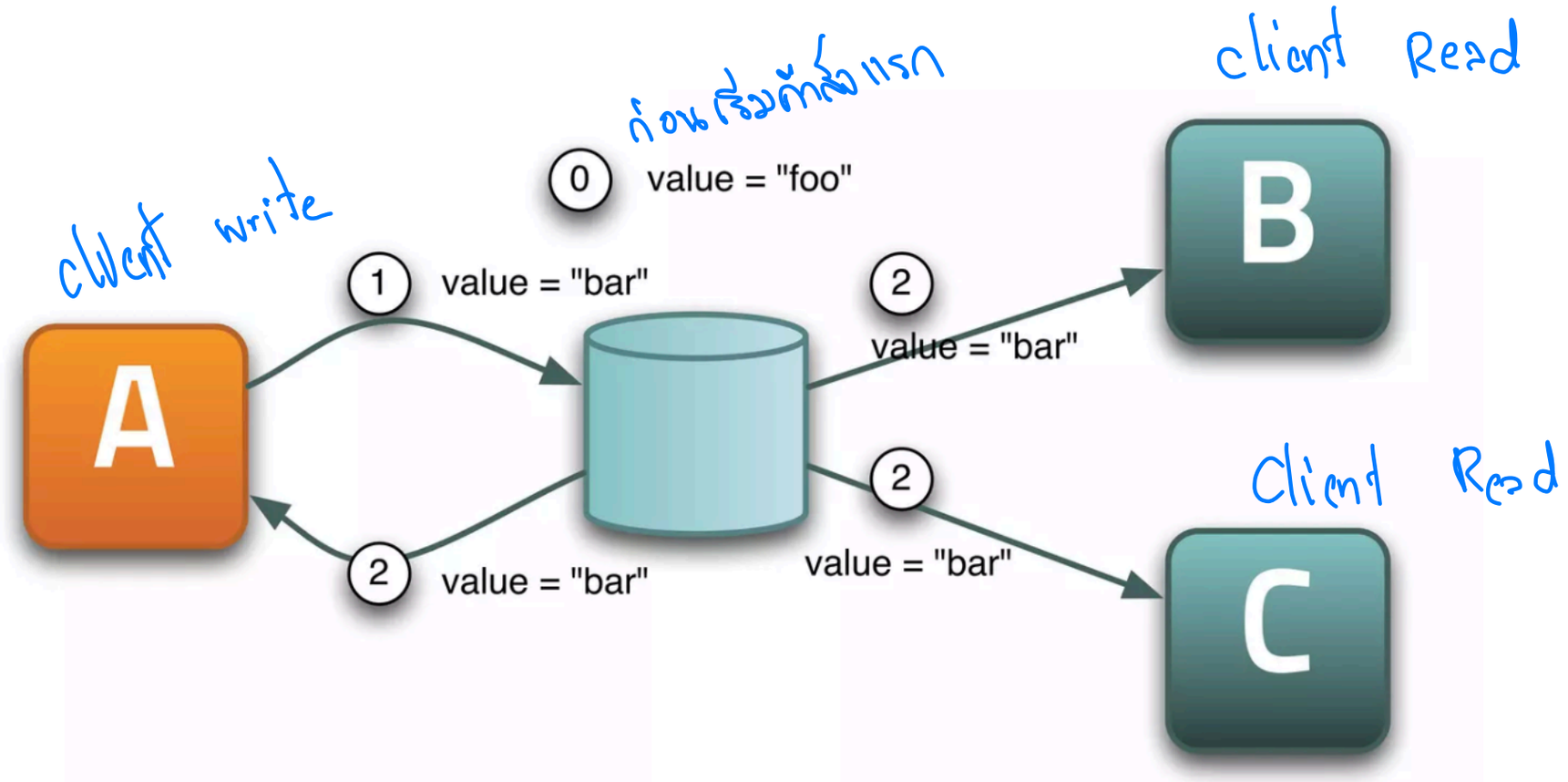
Replication and Data Consistency Problem



Consistency Models (based on Vogels' Paper)

- Strong consistency เสมือนค่าตรงกัน ตลอดเวลา
  - After update completes, any access will return the updated value
  - High costs for large scale system
- Weak consistency ยอมให้ช่วงเวลาหนึ่งที่ data ไม่ตรงกัน
  - Certain conditions must be met before the consistency is guaranteed
  - Time between update completion and guaranteed consistency is called inconsistency window เช่น การ update calendar บนหลายเครื่อง ยังไม่ update ช่วงหนึ่ง

Strong Consistency

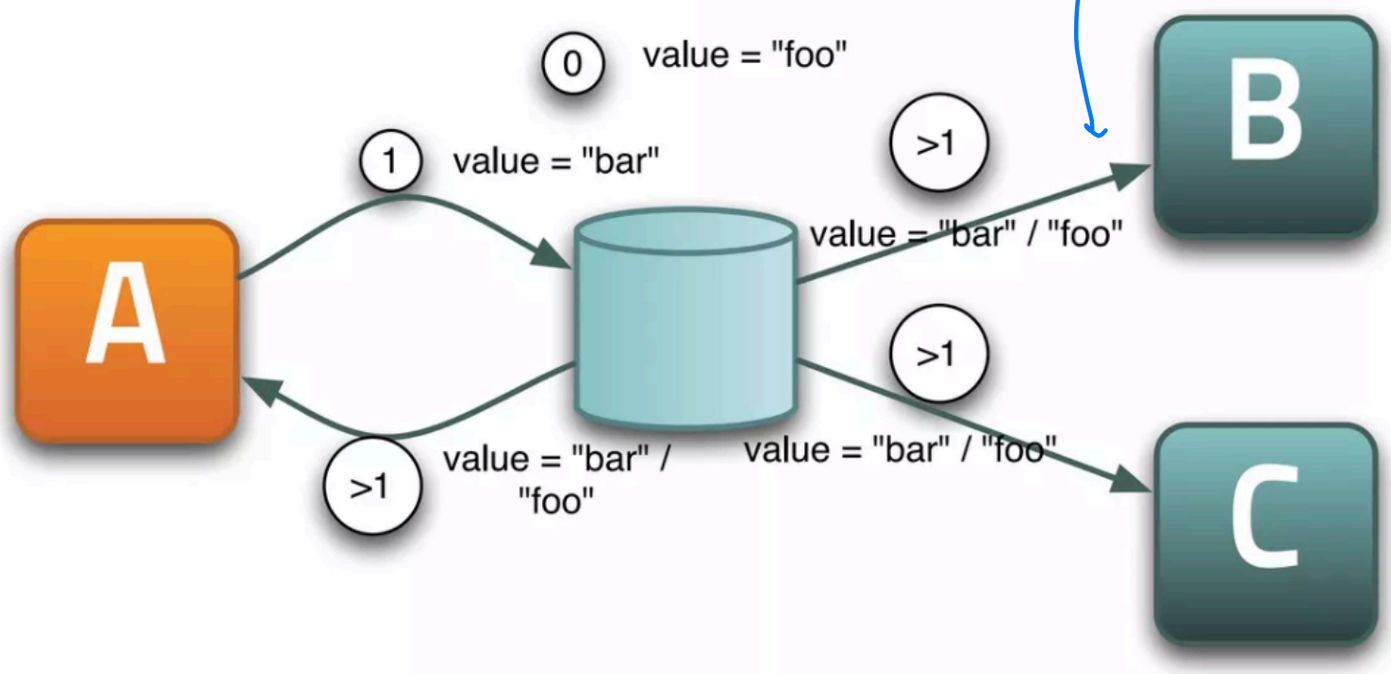


After the update, any subsequent access will return the updated value.

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

Weak Consistency

แต่ไม่บอกว่าเมื่อไหร่จะตรงกัน อาจจะไม่มีวันตรงเลยก็ได้



ข้อมูลอัพเดทเก่า ใหม่ก็ได้

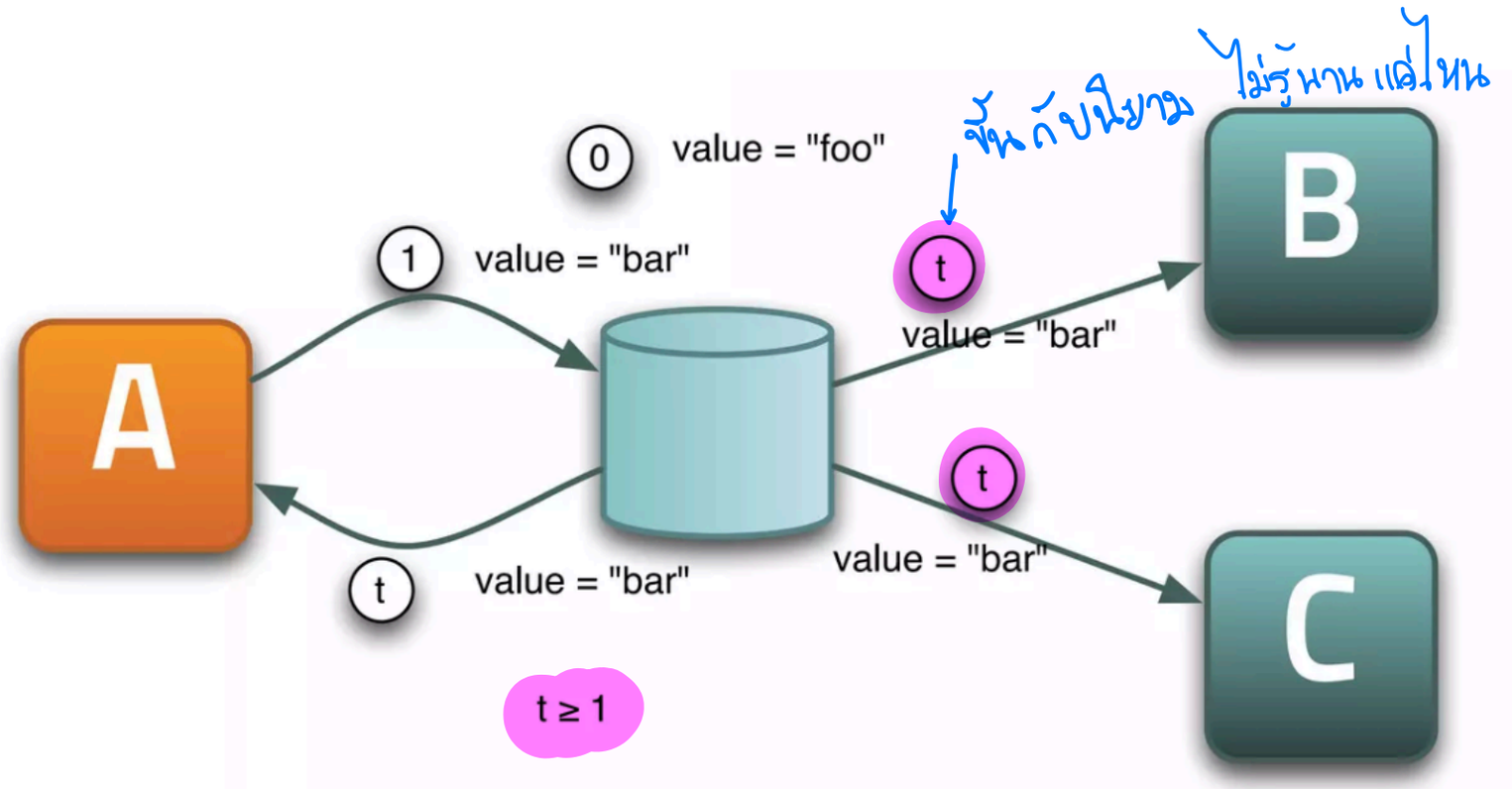
The system does not guarantee that at any given point in the future subsequent access will return the updated value

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 เป็น weak แต่มีคุณๆ นึงบอกว่าข้อมูลตรงกันเดี๋ยว เดี๋ยวไปบางอย่าง
  - 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) A เล่ง B
  - Read-your-write consistency (A updates data, A will always get updated value after that — special case of causal consistency) A update เอง A อ่าน A ได้ล่ะคุณหนอน ตัวอย่างไม่ตรงตัวว่าสอดคล้องยังไง
  - Others (reading assignments)

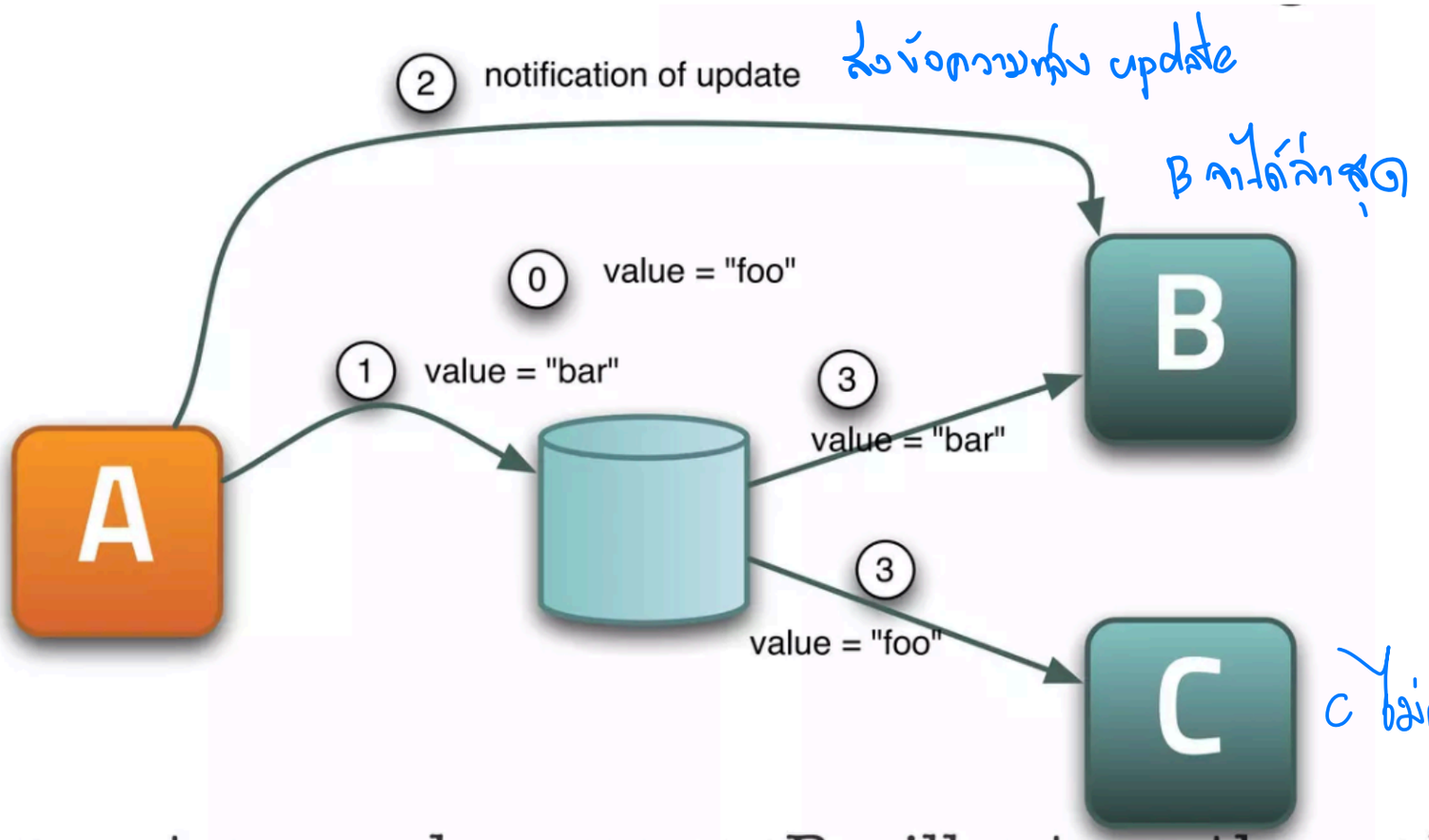
Eventual Consistency



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

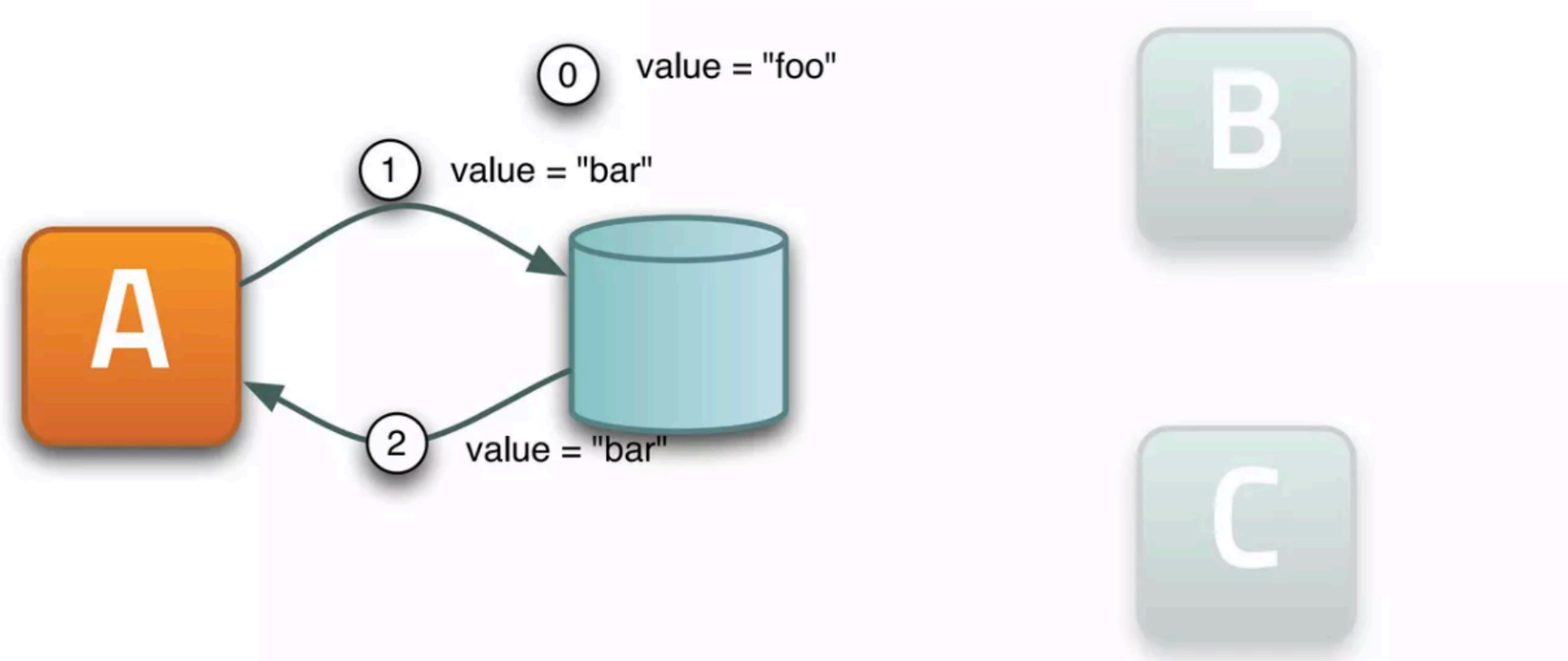
Causal Consistency



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 การอ่านที่เพิ่งทำเอง

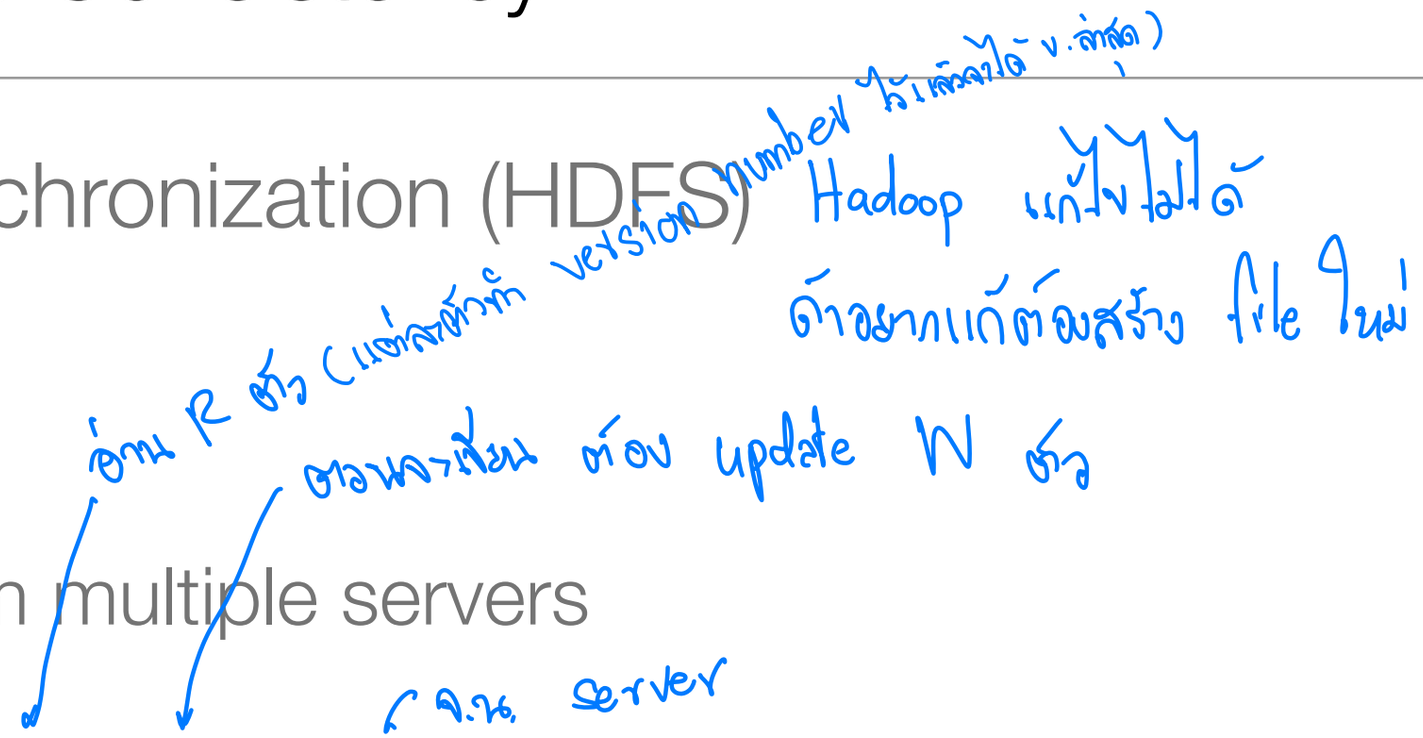


Process A, after updating a data item always access the updated value and never sees an older value

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

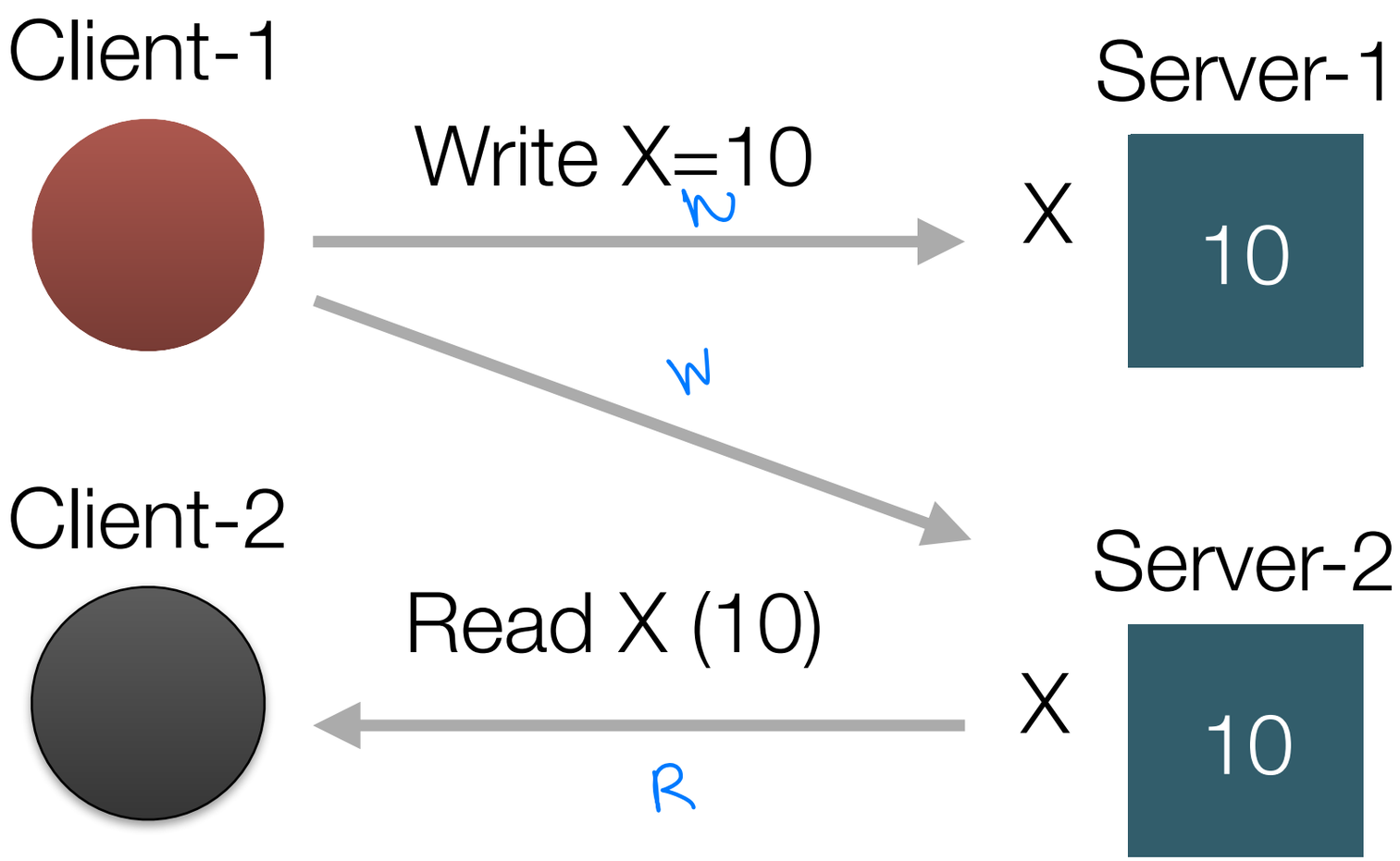
Providing Server-Side Consistency

- 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

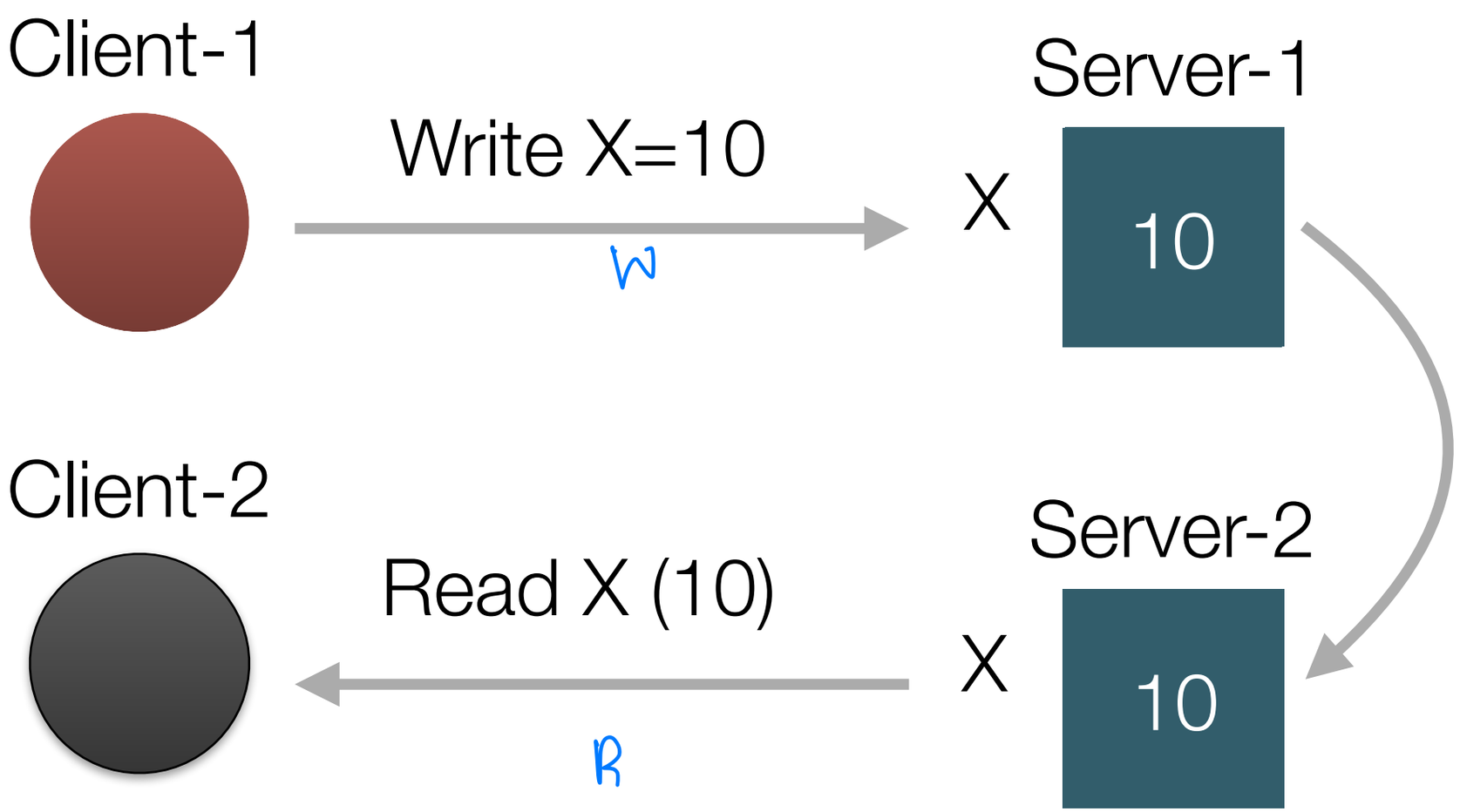


Providing Strong Consistency - Active Replication (W = N, R = 1, R+W > N)

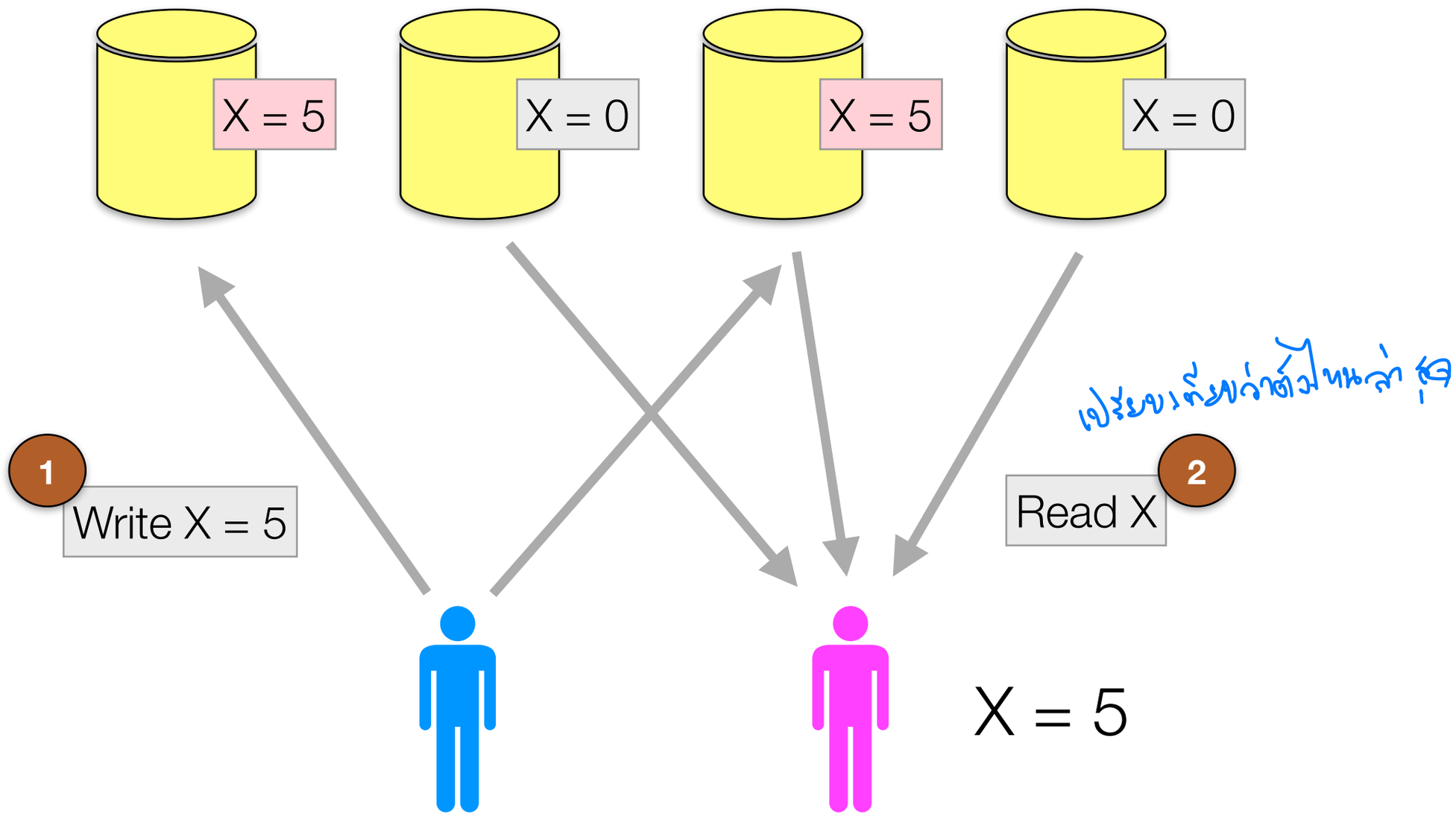
N = 2



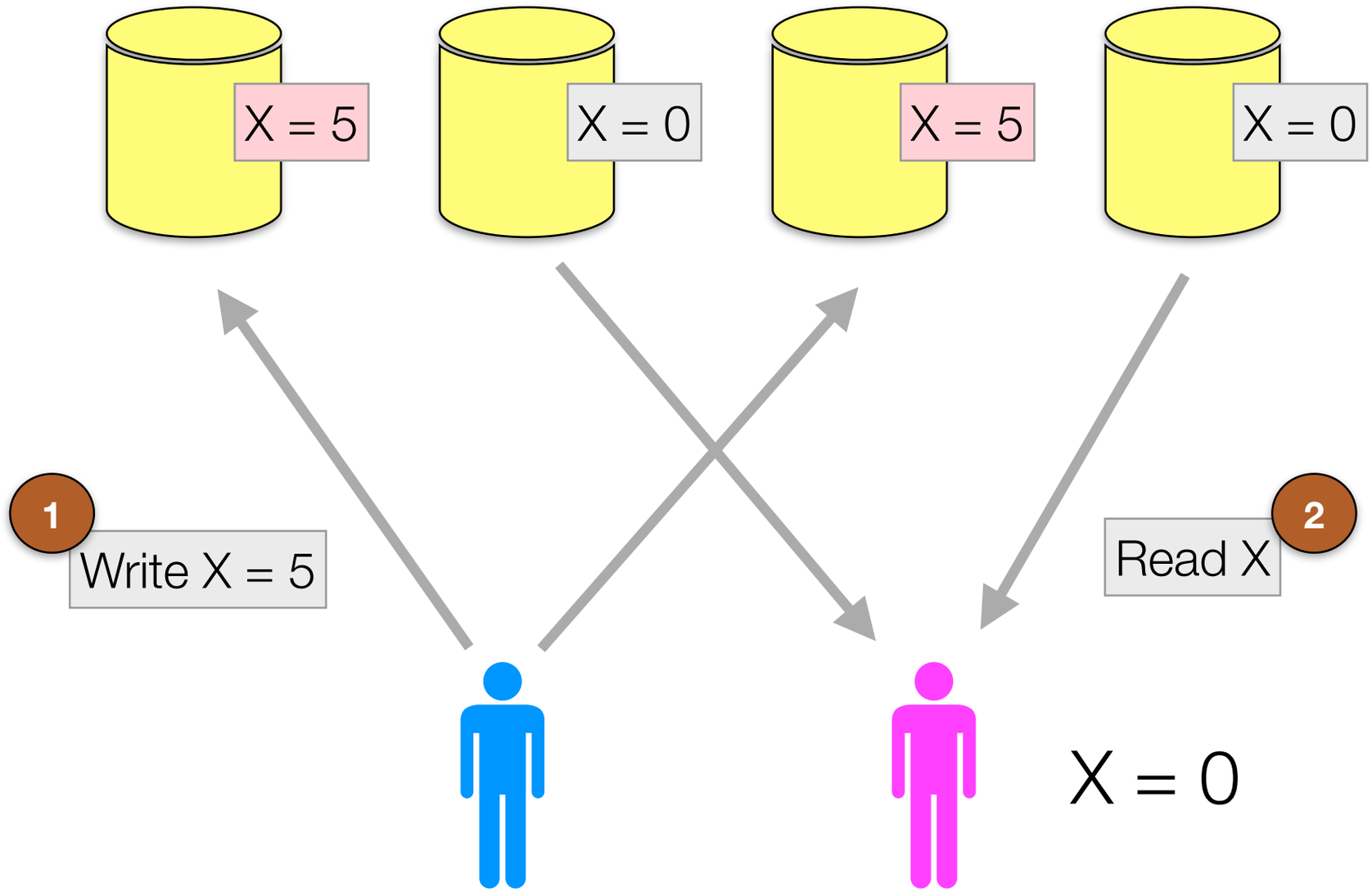
Providing Weak Consistency - Passive Replication (W = 1, R = 1, R+W <= N)



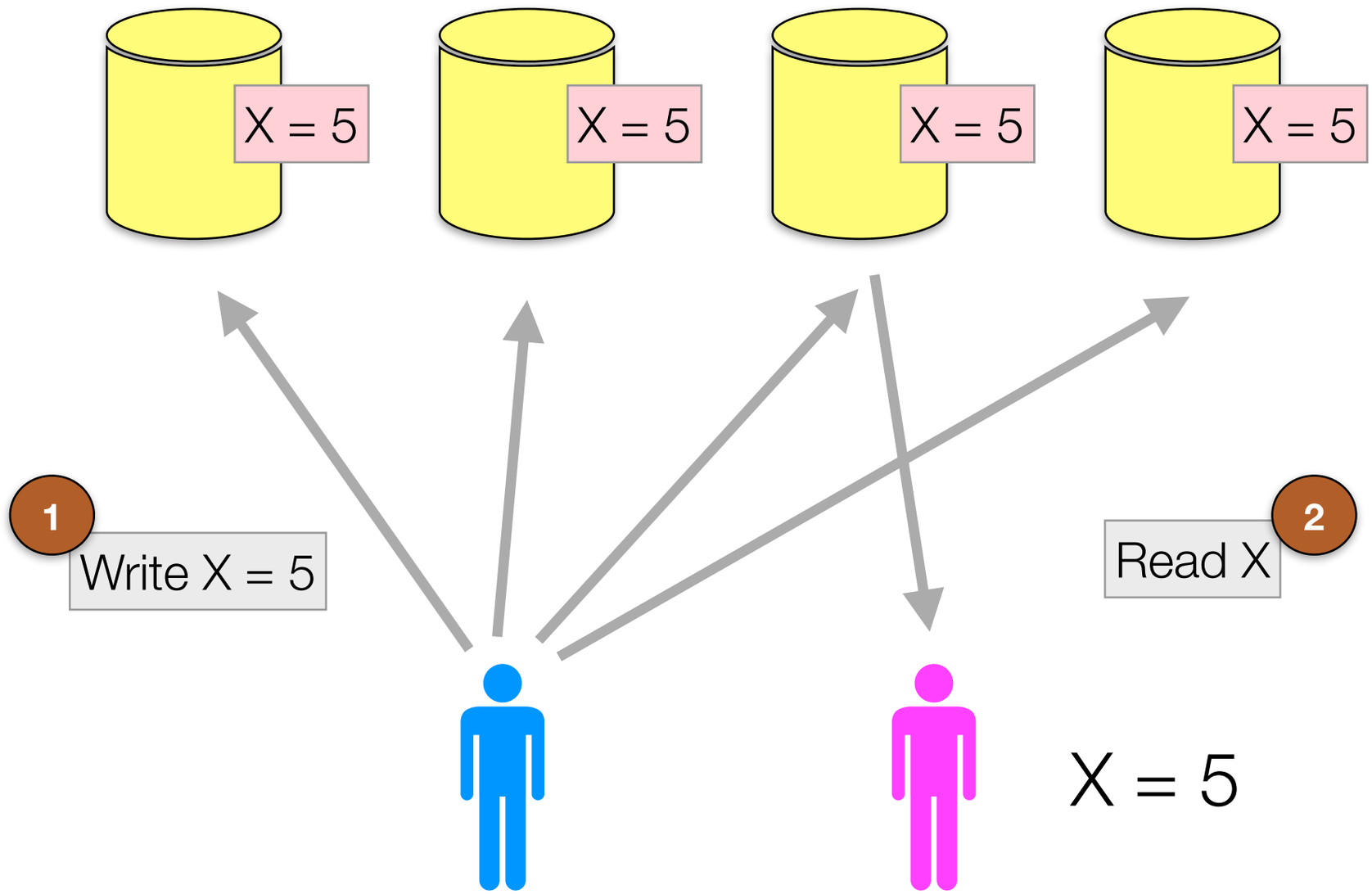
Consistency and Latency Trade-off
N = 4, R = 3, W = 2 (Consistency Guaranteed)



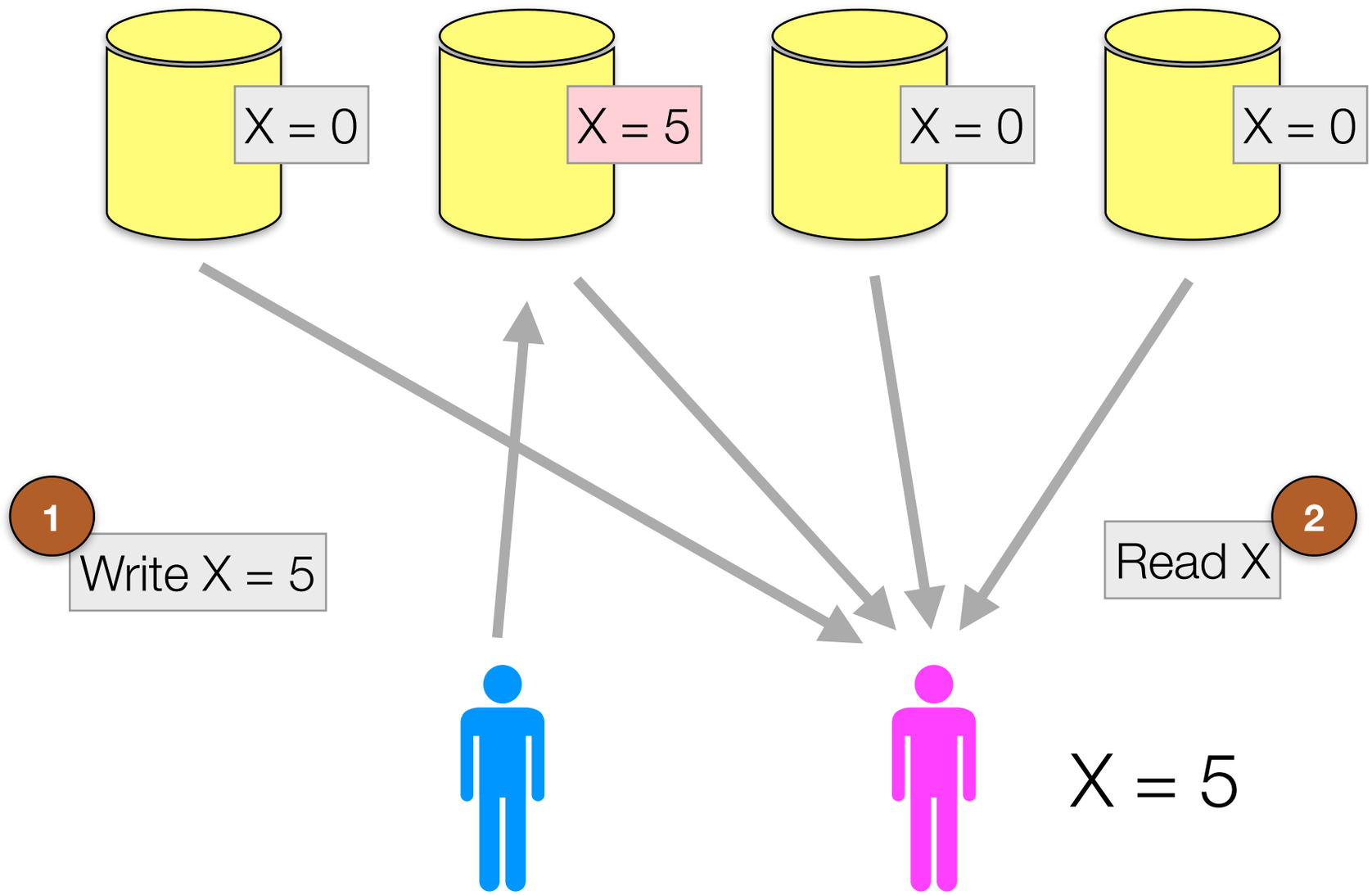
Consistency and Latency Trade-off
N = 4, R = 2, W = 2 (Consistency not Guaranteed)



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