Strong Consistency & CAP Theorem

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

CS 240: Computing Systems and Concurrency
Lecture 15

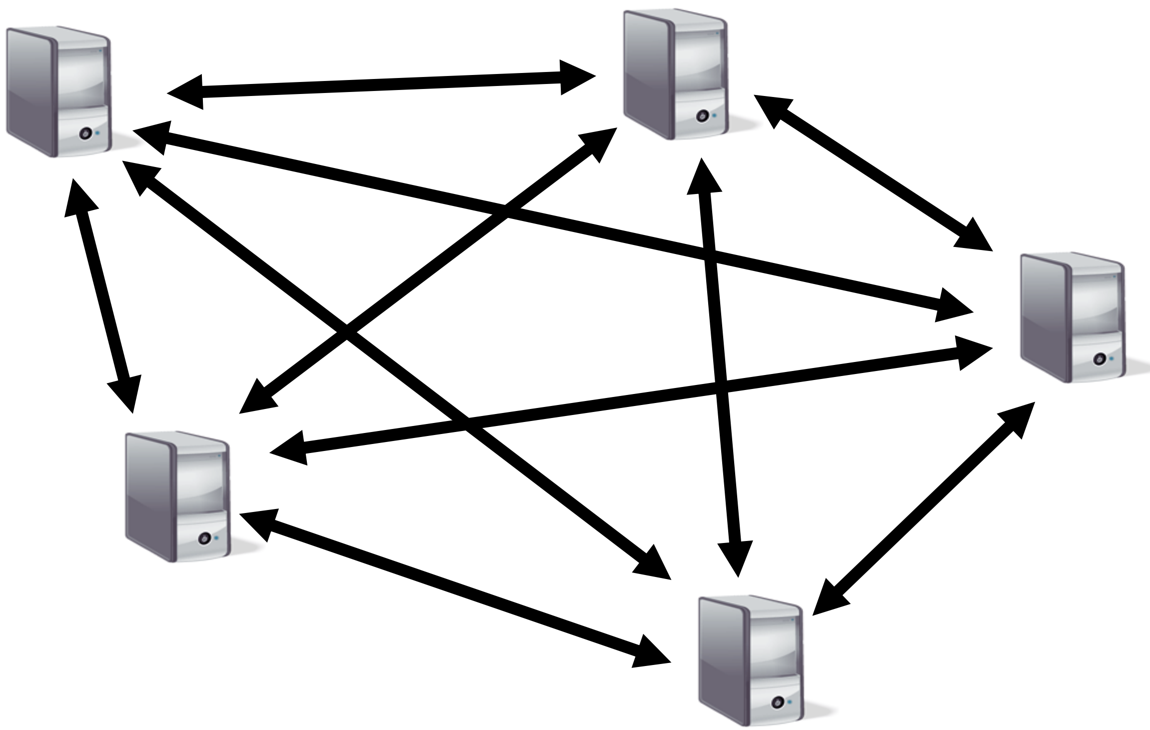
Marco Canini

Credits: Michael Freedman and Kyle Jamieson developed much of the original material.

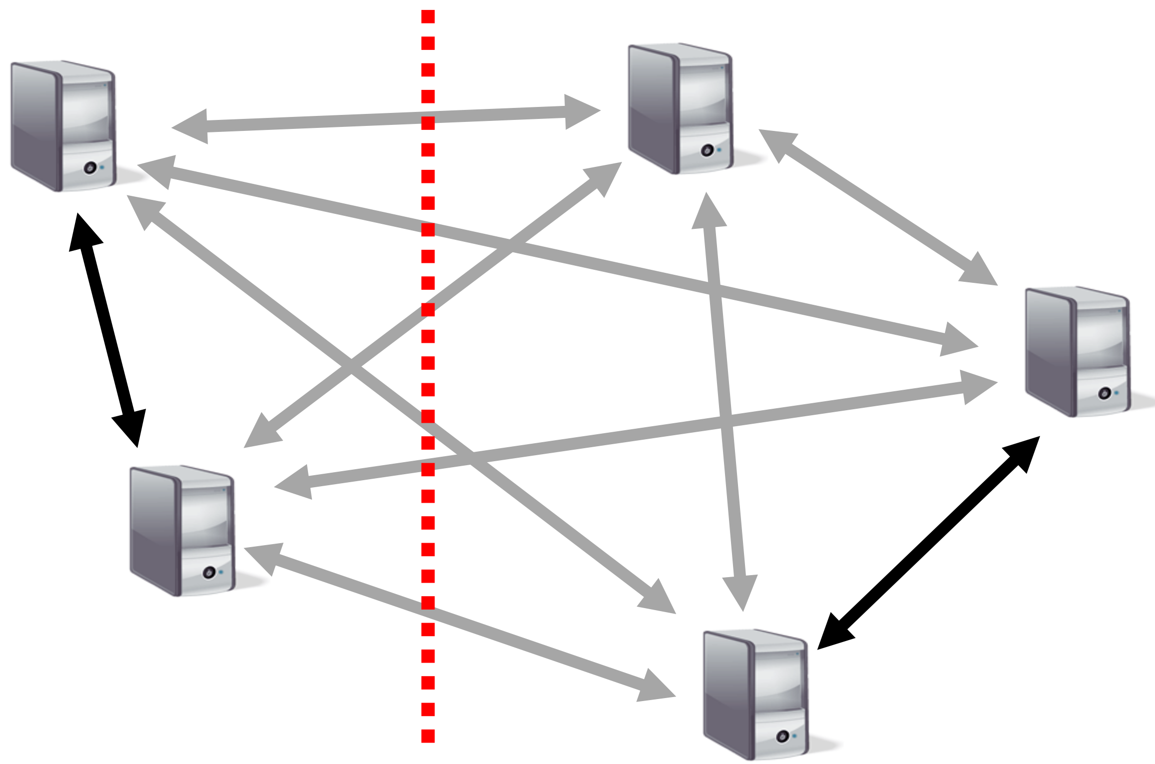
Outline

1. Network Partitions
2. Linearizability
3. CAP Theorem
4. Consistency Hierarchy

Network partitions divide systems



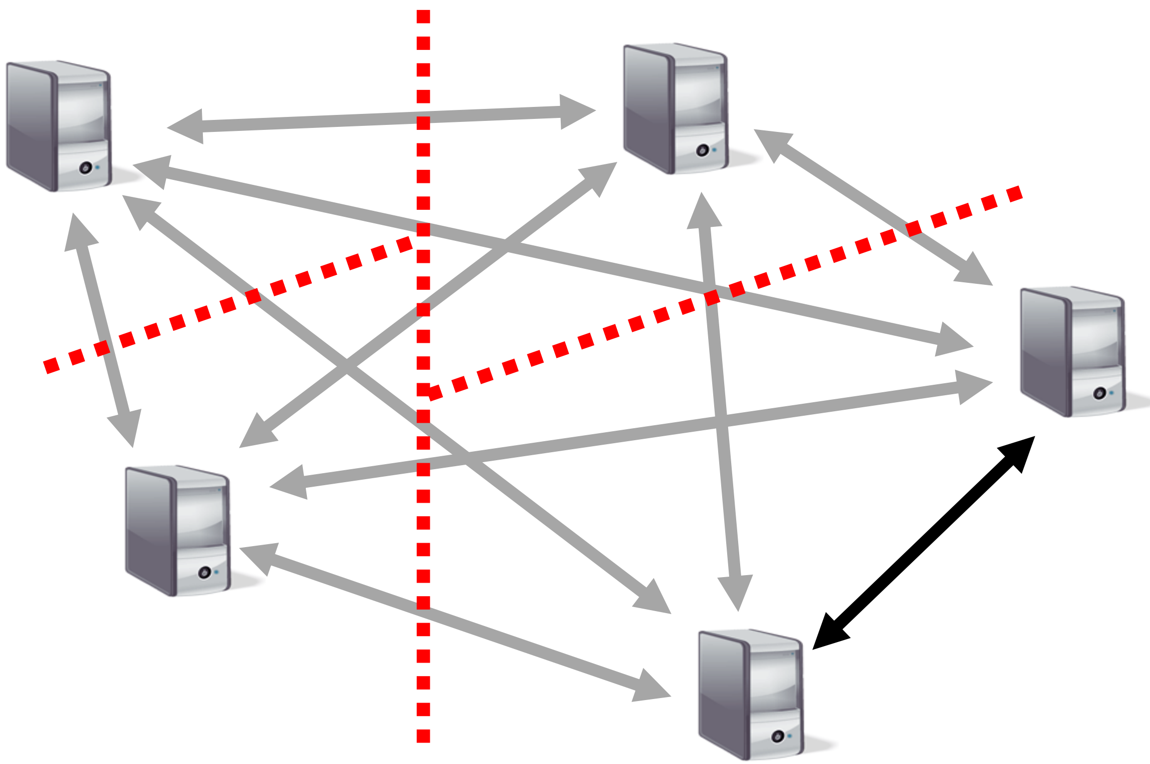
Network partitions divide systems



How can we handle partitions?

- Totally-ordered Multicast?
- Bayou?
- Viewstamped Replication?
- Chord?
- Paxos?
- Dynamo?
- RAFT?

How about this set of partitions?



Fundamental trade-off?

- Replicas appear to be a single machine, but lose availability during a network partition

OR

- All replicas remain available during a network partition but do not appear to be a single machine

CAP theorem preview

- You cannot achieve all three of:
  1. Consistency
  2. Availability
  3. Partition-Tolerance
- Partition Tolerance => Partitions Can Happen
- Availability => All Sides of Partition Continue
- Consistency => Replicas Act Like Single Machine
  - Specifically, Linearizability

Outline

1. Network Partitions
2. Linearizability
3. CAP Theorem
4. Consistency Hierarchy

Linearizability [Herlihy and Wing 1990]

- All replicas execute operations in some total order
- That total order preserves the real-time ordering between operations
  - If operation A completes before operation B begins, then A is ordered before B in real-time
  - If neither A nor B completes before the other begins, then there is no real-time order
    - (But there must be *some* total order)

Linearizability == “Appears to be a Single Machine”

- Single machine processes requests one by one in the order it receives them
  - Will receive requests ordered by real-time in that order
  - Will receive all requests in some order
- Atomic Multicast, Viewstamped Replication, Paxos, and RAFT provide Linearizability

Linearizability is ideal?

- Hides the complexity of the underlying distributed system from applications!
  - Easier to write applications
  - Easier to write correct applications

- But, performance trade-offs, e.g., CAP

Outline

1. Network Partitions
2. Linearizability
3. CAP Theorem
4. Consistency Hierarchy

CAP conjecture [Brewer 00]

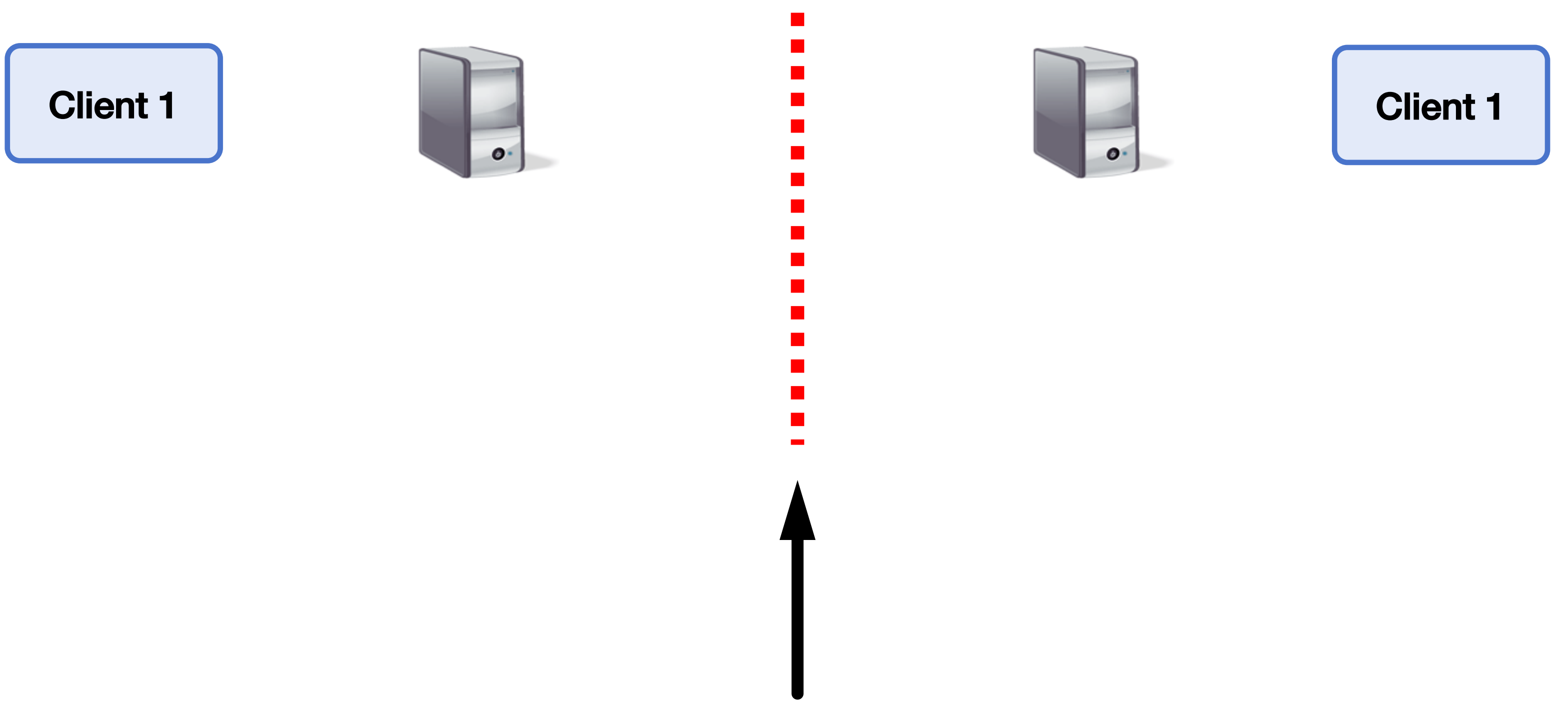
- From keynote lecture by Eric Brewer (2000)
  - History: Eric started Inktomi, early Internet search site based around "commodity" clusters of computers
  - Using CAP to justify "BASE" model: Basically Available, Soft-state services with Eventual consistency
- Popular interpretation: 2-out-of-3
  - Consistency (Linearizability)
  - Availability
  - Partition Tolerance: Arbitrary crash/network failures

CAP theorem [Gilbert Lynch 02]

Assume to contradict that Algorithm *A* provides all of CAP

CAP theorem [Gilbert Lynch 02]

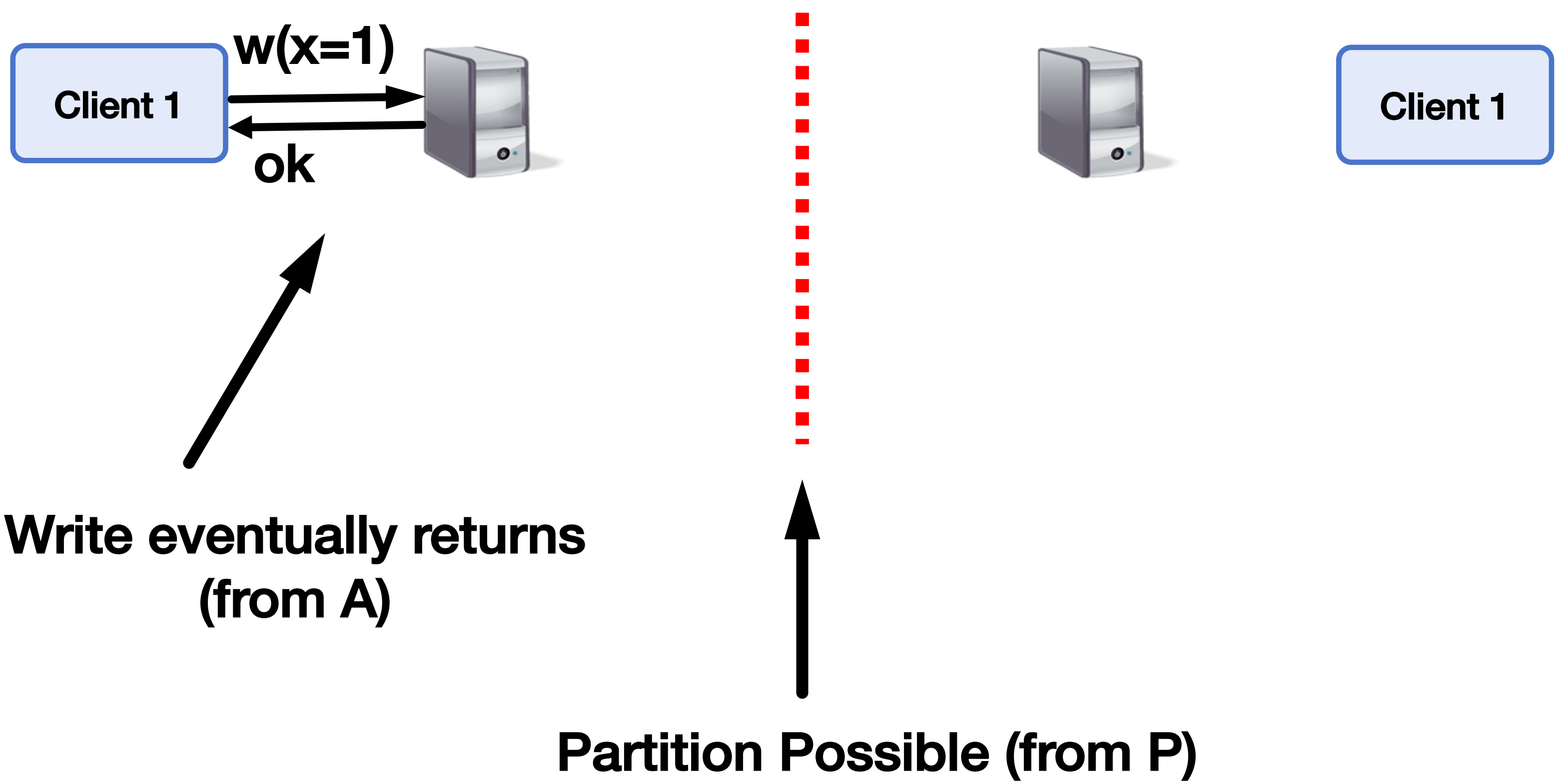
Assume to contradict that Algorithm *A* provides all of CAP



Partition Possible (from P)

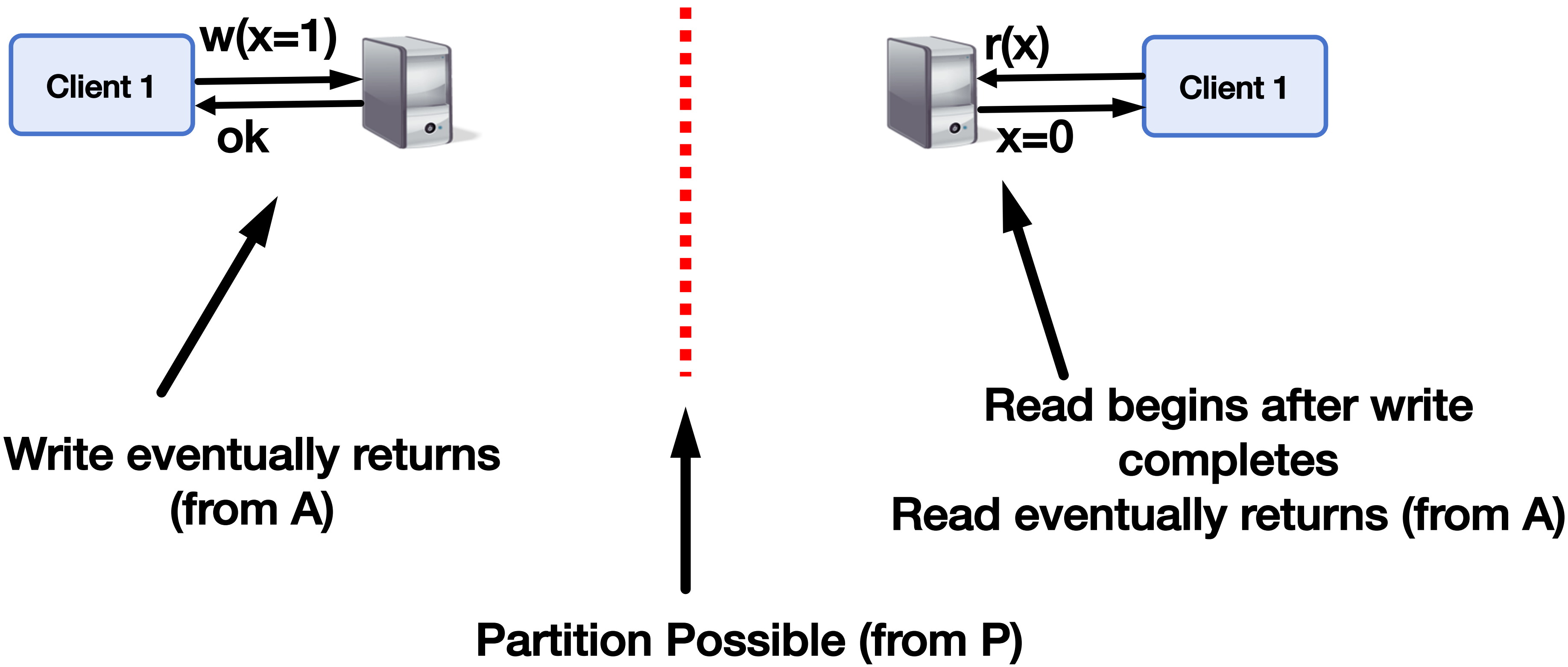
CAP theorem [Gilbert Lynch 02]

Assume to contradict that Algorithm *A* provides all of CAP



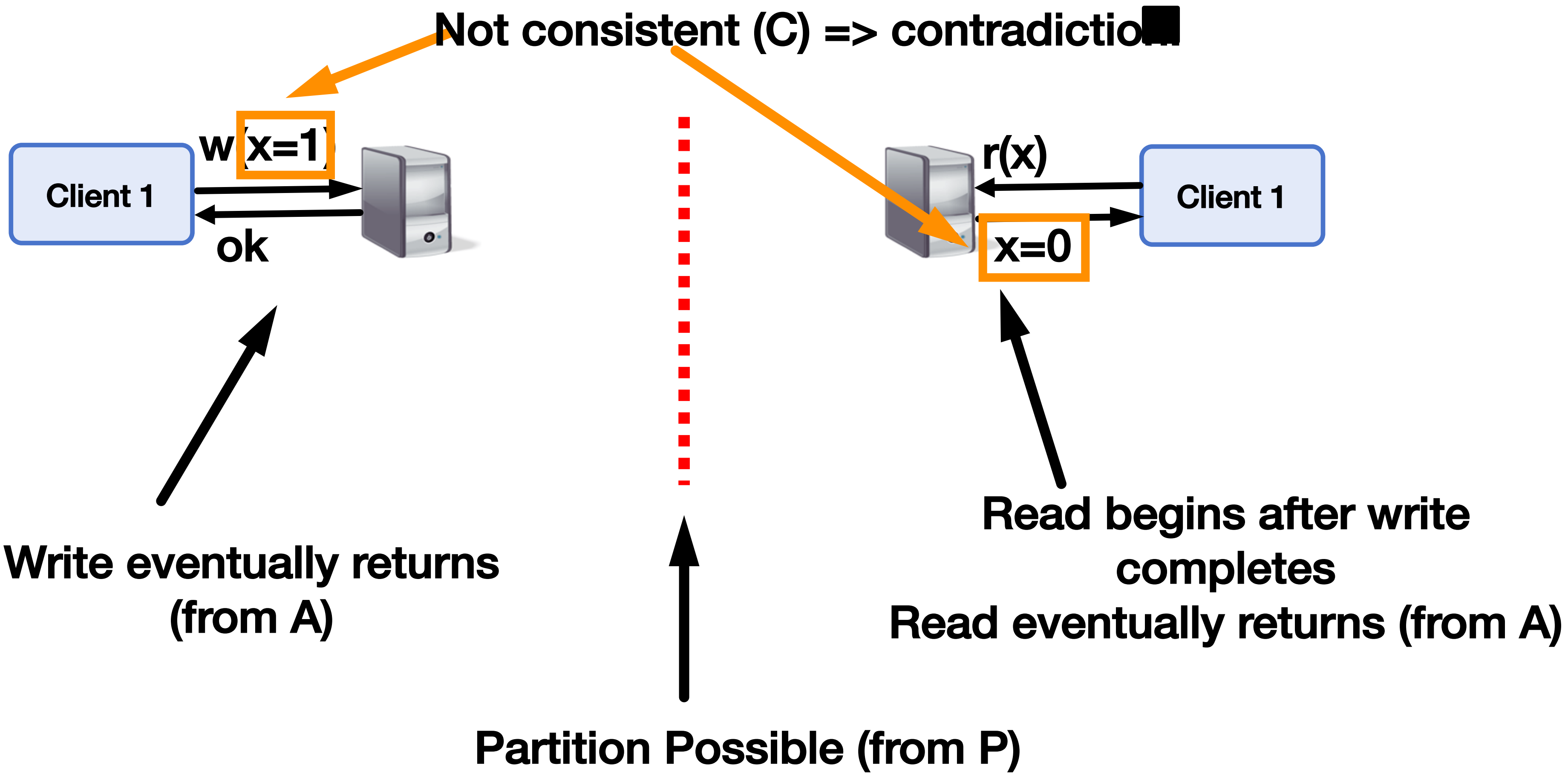
CAP theorem [Gilbert Lynch 02]

Assume to contradict that Algorithm *A* provides all of CAP



CAP theorem [Gilbert Lynch 02]

Assume to contradict that Algorithm *A* provides all of CAP



CAP interpretation 1/2

- Cannot “choose” no partitions
  - 2-out-of-3 interpretation doesn’t make sense
  - Instead, availability OR consistency?
- i.e., fundamental trade-off between availability and consistency
  - When designing system must choose one or the other, both are not possible

CAP interpretation 2/2

- It is a theorem, with a proof, that you understand!
- Cannot “beat” CAP theorem
- Can engineer systems to make partitions extremely rare, however, and then just take the rare hit to availability (or consistency)

More trade-offs L vs. C

- Low-latency: Speak to fewer than quorum of nodes?
  - 2PC: write N, read 1
  - RAFT: write $\lfloor N/2 \rfloor + 1$, read $\lfloor N/2 \rfloor + 1$
  - General: $|W| + |R| > N$

- L and C are fundamentally at odds
  - "C" = linearizability, sequential, serializability (more later)

PACELC

- If there is a partition (P):
  - How does system tradeoff A and C?
- Else (no partition)
  - How does system tradeoff L and C?

- Is there a useful system that switches?
  - Dynamo: PA/EL
  - "ACID" dbs: PC/EC

http://dbmsmusings.blogspot.com/2010/04/problems-with-cap-and-yahoos-little.html

Outline

1. Network Partitions
2. Linearizability
3. CAP Theorem
4. Consistency Hierarchy

Consistency models

- Contract between a distributed system and the applications that run on it

- A consistency model is a set of guarantees made by the distributed system

- e.g., Linearizability
  - Guarantees a total order of operations
  - Guarantees the real-time ordering is respected

Stronger vs weaker consistency

- Stronger consistency models

  \+ Easier to write applications

  \- More guarantees for the system to ensure

  Results in performance tradeoffs

- Weaker consistency models

  \- Harder to write applications

  \+ Fewer guarantees for the system to ensure

Consistency hierarchy

Linearizability (Strong/Strict Consistency)	e.g., RAFT
↓	
Sequential Consistency	
↓	
Causal+ Consistency	e.g., Bayou
↓	
Eventual Consistency	e.g., Dynamo

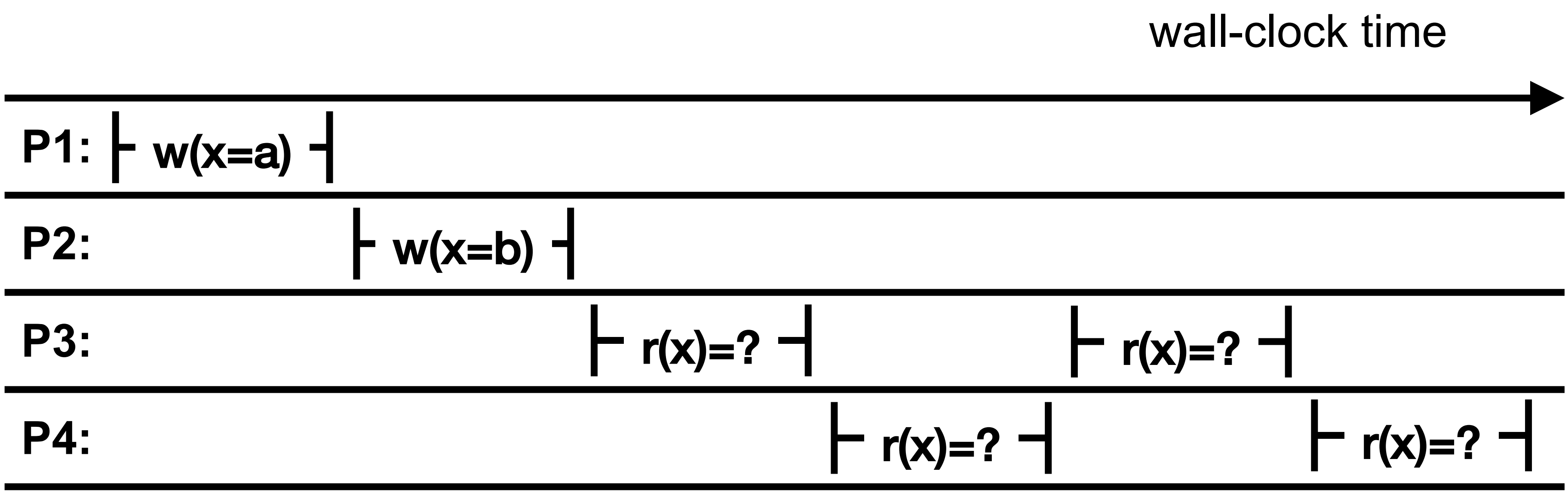
Strictly stronger consistency

- A consistency model *A* is strictly stronger than *B* if it allows a strict subset of the behaviors of B
  - Guarantees are strictly stronger

- Linearizability is strictly stronger than Sequential Consistency
  - Linearizability: ∃total order + real-time ordering
  - Sequential: ∃total order + process ordering
    - Process ordering ⊆ Real-time ordering

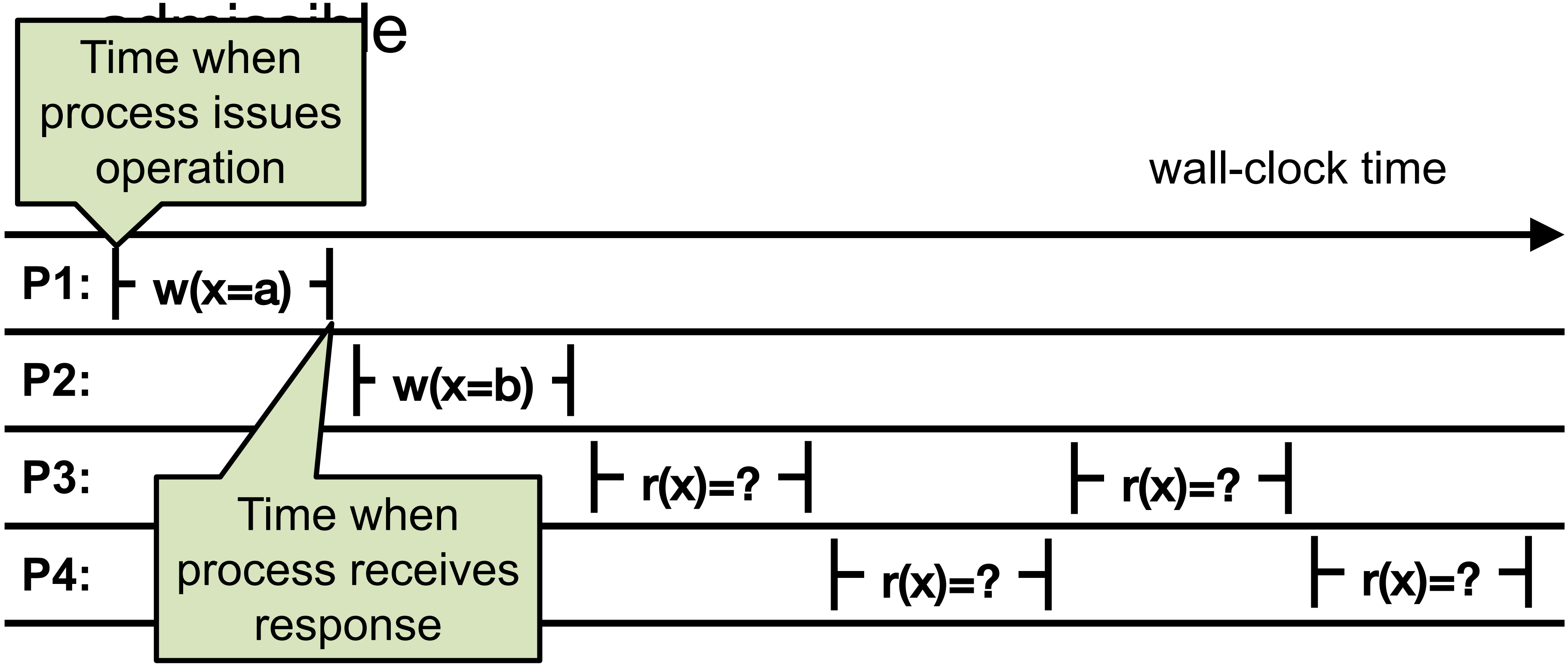
Intuitive example

- Consistency model defines what values reads are admissible



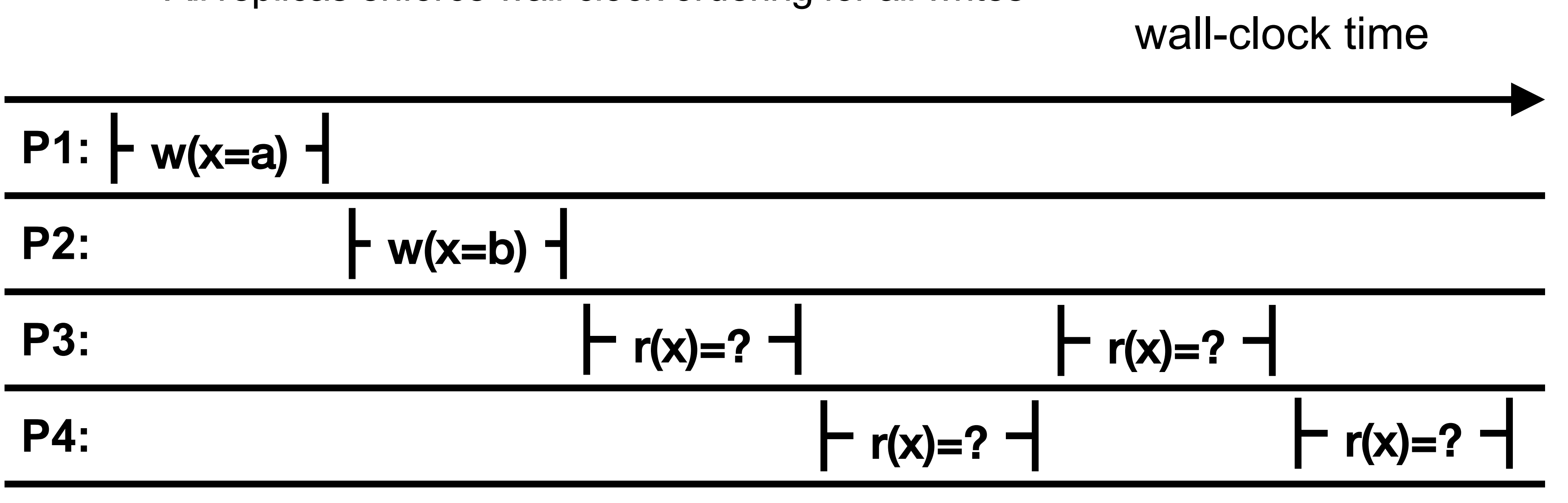
Intuitive example

- Consistency model defines what values reads are admissible



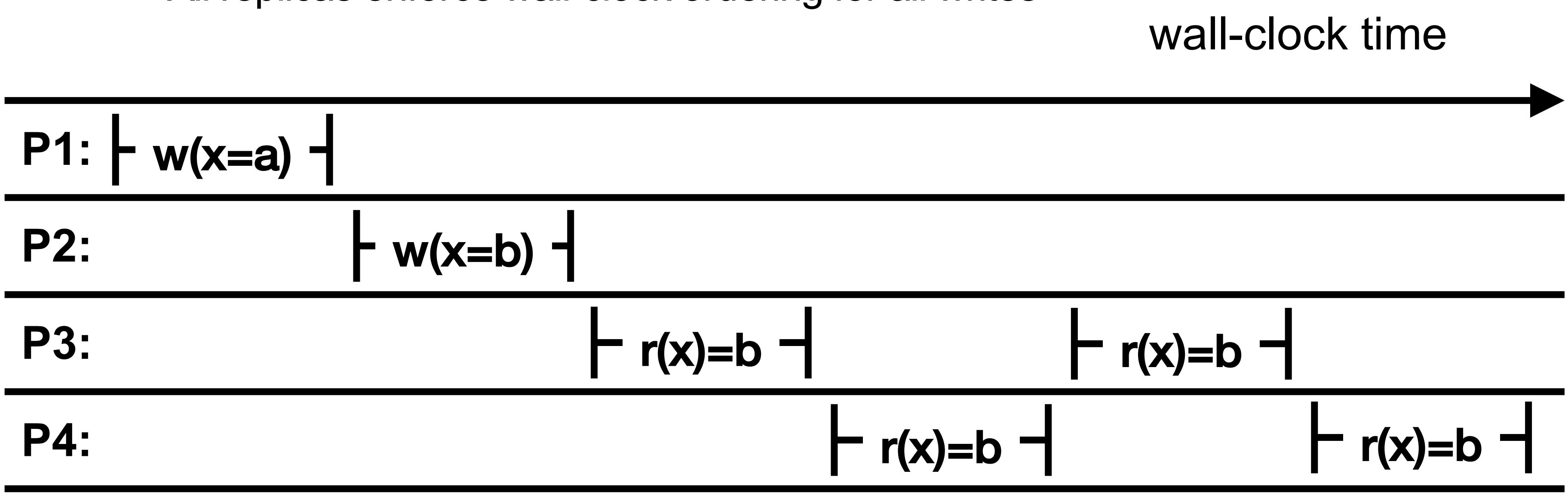
Linearizability

- Any execution is the same as if all read/write ops were executed in order of **wall-clock time** at which they were issued
- Therefore:
  - Reads are never stale
  - All replicas enforce wall-clock ordering for all writes



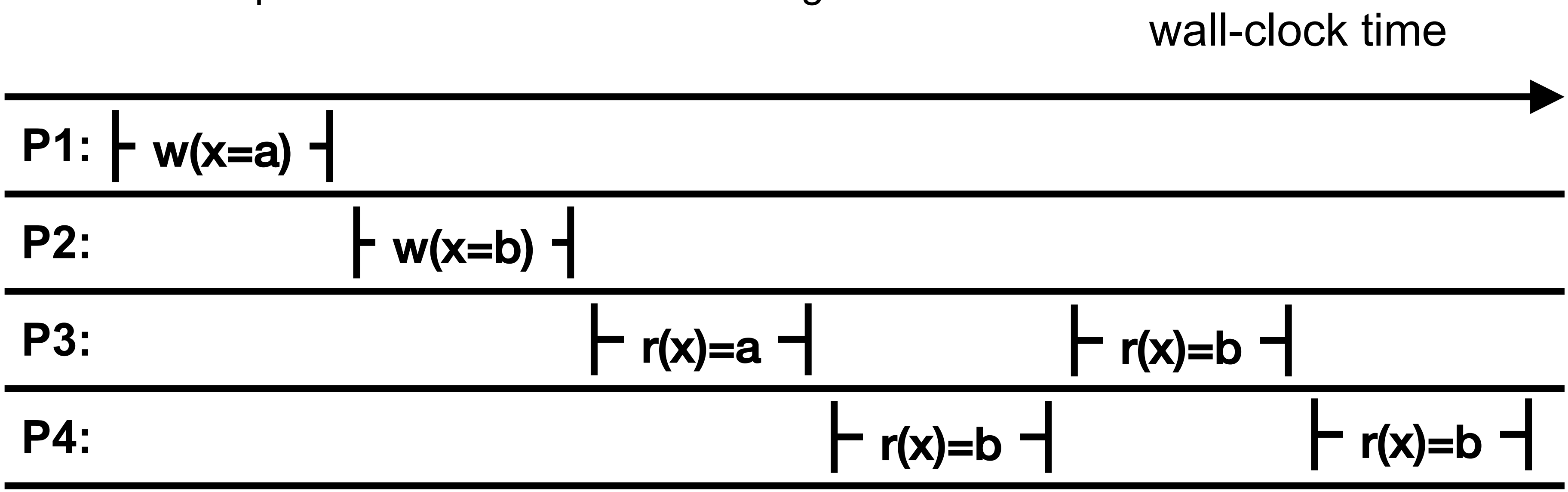
Linearizability: YES

- Any execution is the same as if all read/write ops were executed in order of **wall-clock time** at which they were issued
- Therefore:
  - Reads are never stale
  - All replicas enforce wall-clock ordering for all writes



Linearizability: NO

- Any execution is the same as if all read/write ops were executed in order of **wall-clock time** at which they were issued
- Therefore:
  - Reads are never stale
  - All replicas enforce wall-clock ordering for all writes



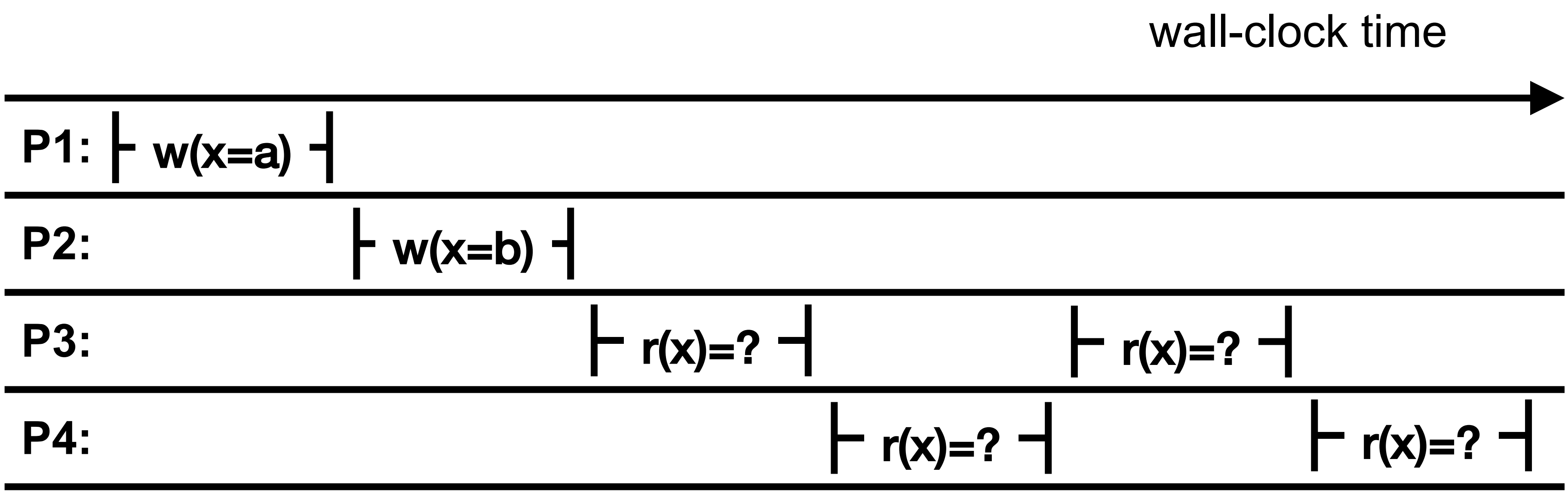
Sequential consistency

- Sequential = Linearizability – real-time ordering
    1. All servers execute all ops in *some* identical sequential order
    2. Global ordering preserves each client's own local ordering

- With concurrent ops, "reordering" of ops (w.r.t. real-time ordering) acceptable, but all servers must see same order
    - e.g., linearizability cares about time
      sequential consistency cares about program order

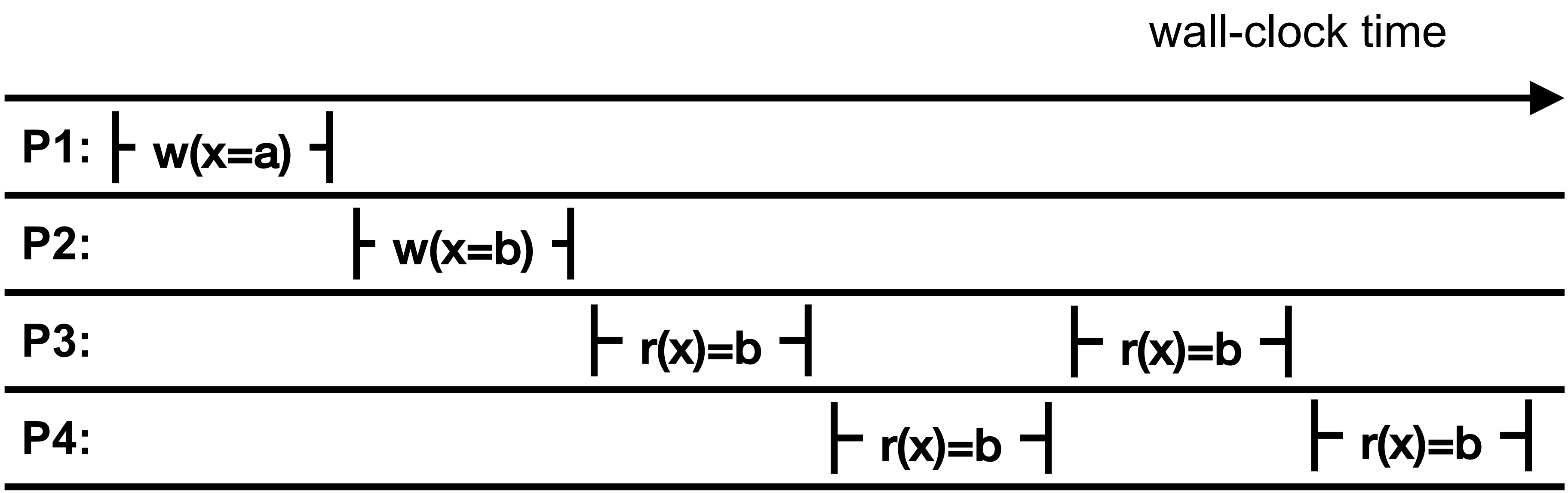
Sequential consistency

- Any execution is the same as if all read/write ops were executed in **some global ordering**, and the ops of each client process appear in the **program order**
- Therefore:
  - Reads may be stale in terms of real time, but not in logical time
  - Writes are totally ordered according to logical time across all replicas



Sequential consistency: YES

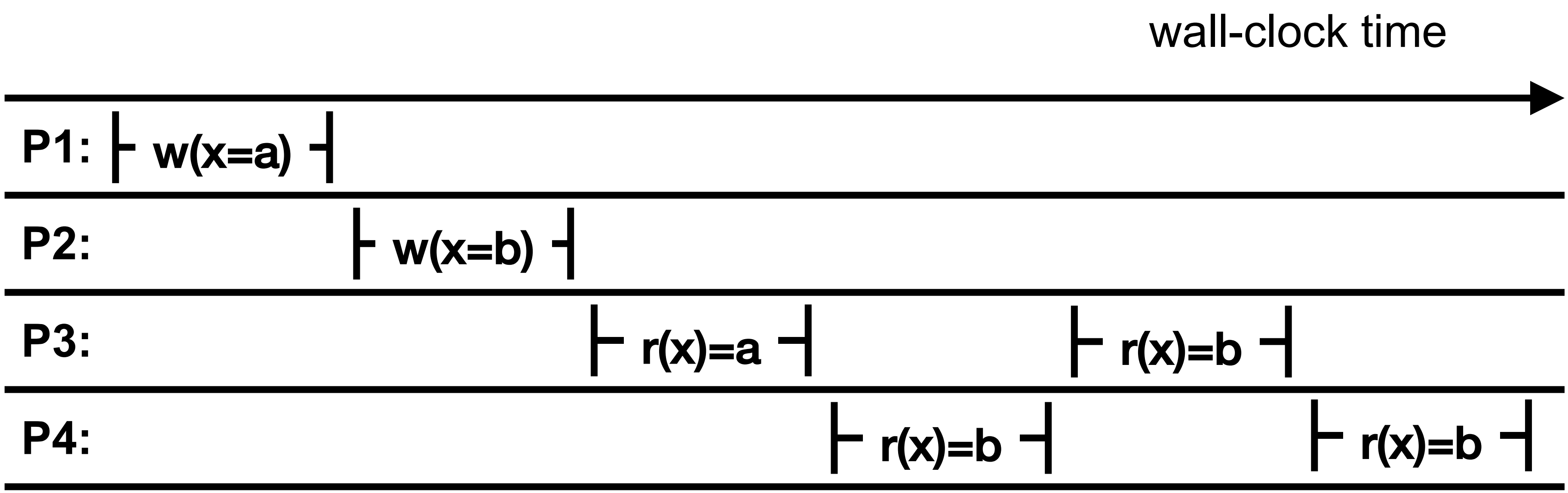
- Any execution is the same as if all read/write ops were executed in **some global ordering**, and the ops of each client process appear in the **program order**
- Therefore:
  - Reads may be stale in terms of real time, but not in logical time
  - Writes are totally ordered according to logical time across all replicas



Also valid with linearizability

Sequential consistency: YES

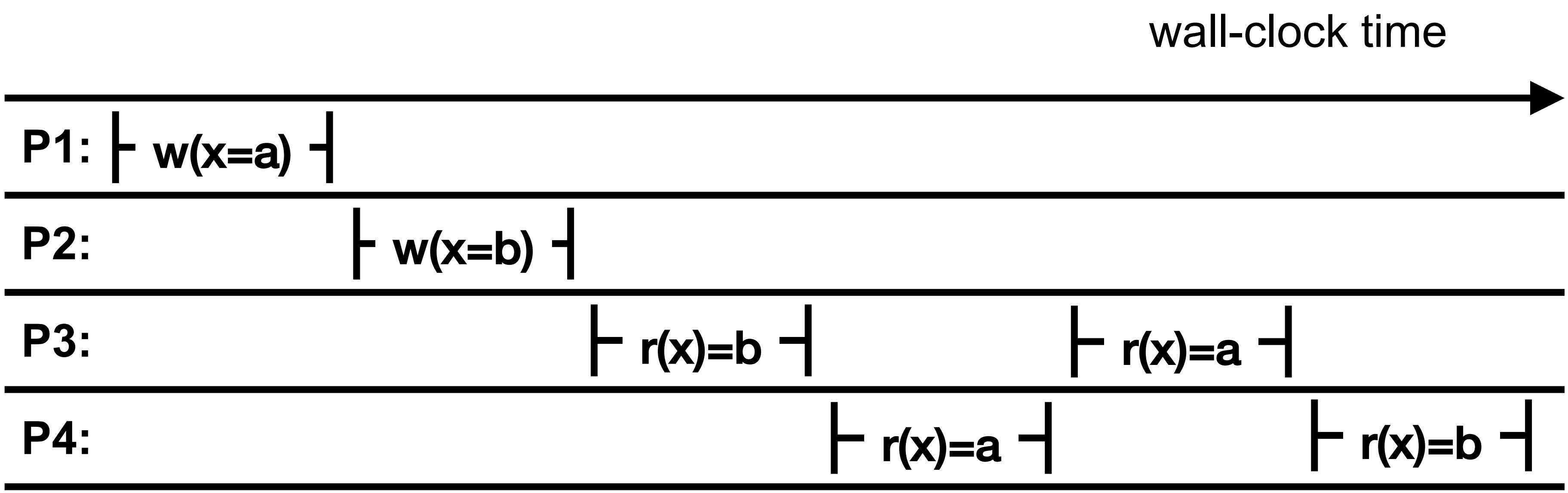
- Any execution is the same as if all read/write ops were executed in **some global ordering**, and the ops of each client process appear in the **program order**
- Therefore:
  - Reads may be stale in terms of real time, but not in logical time
  - Writes are totally ordered according to logical time across all replicas



Not valid with linearizability

Sequential consistency: NO

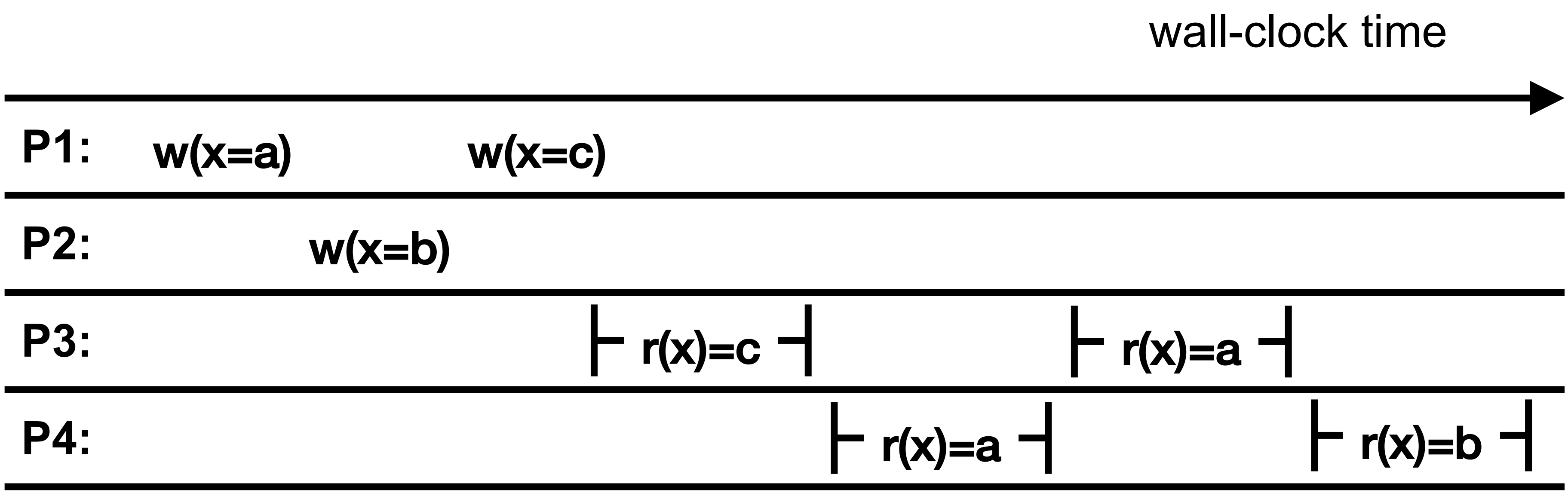
- Any execution is the same as if all read/write ops were executed in **some global ordering**, and the ops of each client process appear in the **program order**
- Therefore:
  - Reads may be stale in terms of real time, but not in logical time
  - Writes are totally ordered according to logical time across all replicas



No global ordering can explain these results

Sequential consistency: NO

- Any execution is the same as if all read/write ops were executed in **some global ordering**, and the ops of each client process appear in the **program order**
- Therefore:
  - Reads may be stale in terms of real time, but not in logical time
  - Writes are totally ordered according to logical time across all replicas



No *sequential* global ordering can explain these results…
E.g.: w(x=c), r(x)=c, r(x)=a, w(x=b) doesn't preserve P1's ordering

Causal+ Consistency

- Partially orders all operations, does not totally order them
  - Does not look like a single machine

- Guarantees
  - For each process, $\exists$ an order of all writes + that process's reads
  - Order respects the happens-before ($\rightarrow$) ordering of operations
  - + replicas converge to the same state
    - Skip details, makes it stronger than eventual consistency

Causal+ But Not Sequential

P_A ├ w(x=1) ┤ ├ r(y)=0 ┤

P_B ├ w(y=1) ┤ ├ r(x)=0 ┤

√ Casual+

Happens Before Order

w(x=1) → r(y)=0

w(y=1) → r(x)=0

P_A Order: w(x=1), r(y=0), w(y=1)

P_B Order: w(y=1), r(x=0), w(x=1)

X Sequential

Process Ordering

w(x=1) → r(y)=0

w(y=1) → r(x)=0

No Total Order

w(x=1) → r(y)=0

w(y=1) → r(x)=0

Eventual But Not Causal+

P_A ├ w(x=1) ┤ ├ w(y=1) ┤

P_B ├ r(y)=1 ┤ ├ r(x)=0 ┤

√ Eventual

As long as P_B eventually would see r(x)=1 this is fine

X Causal+

Happens Before Ordering

w(x=1) → w(y)=1

r(y)=1 → r(x)=0

No Order for P_B

w(x=1) → w(y)=1

r(y)=1 → r(x)=0