Consistency Models

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

CS 240: Computing Systems and Concurrency
Lecture 15

Marco Canini

Consistency Models

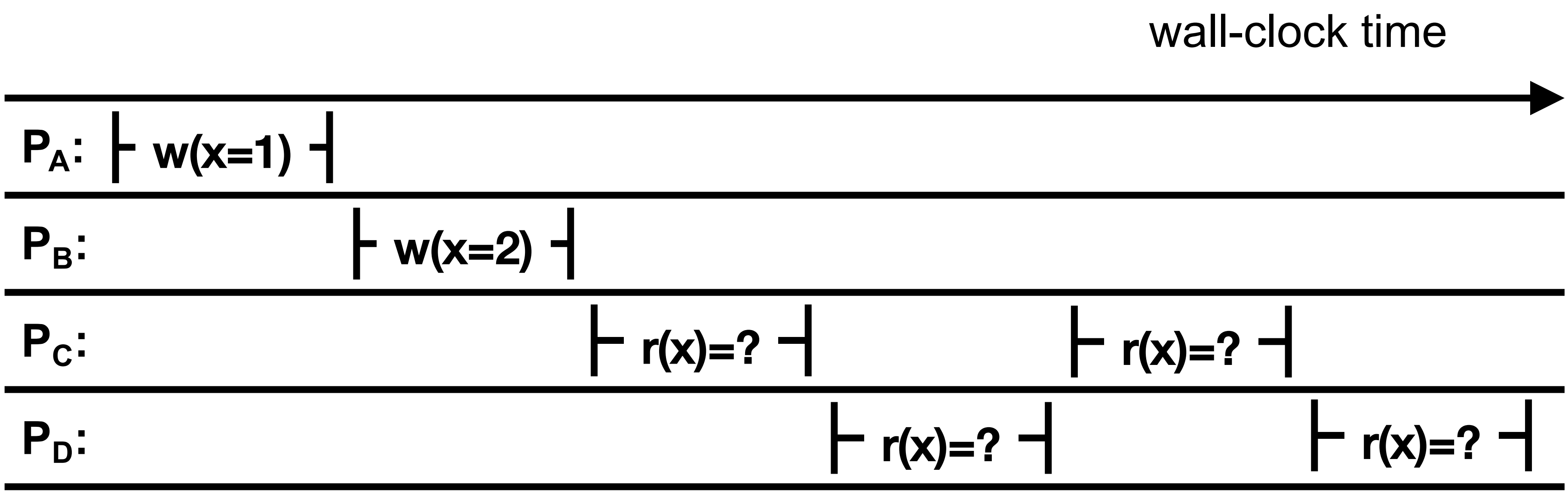
- Contract between a distributed system and the applications that run on it
- A consistency model is a set of guarantees made by the distributed system
- We are concerned with: “what happens if a client modifies some data items and concurrently another client reads or modifies the same items possibly at a different replica”?

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)

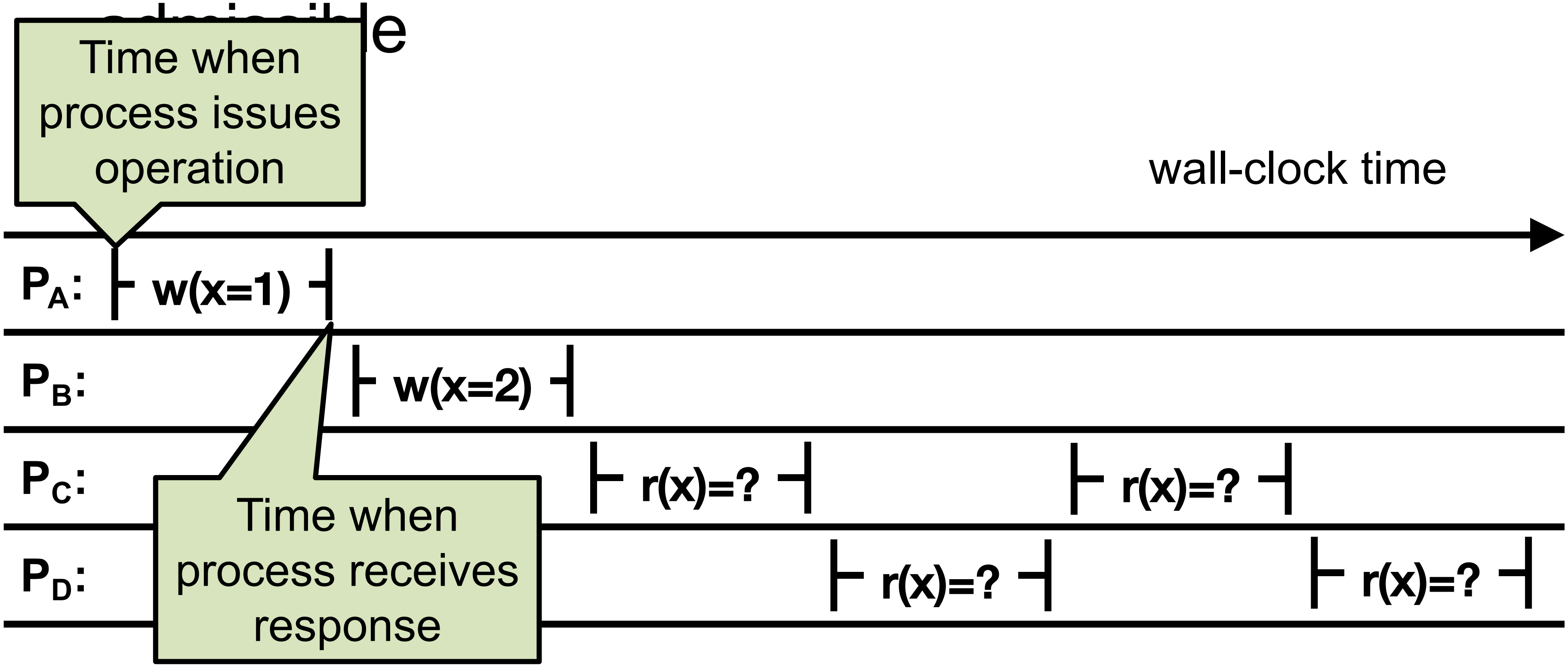
Intuitive example

- Consistency model defines what values reads are admissible



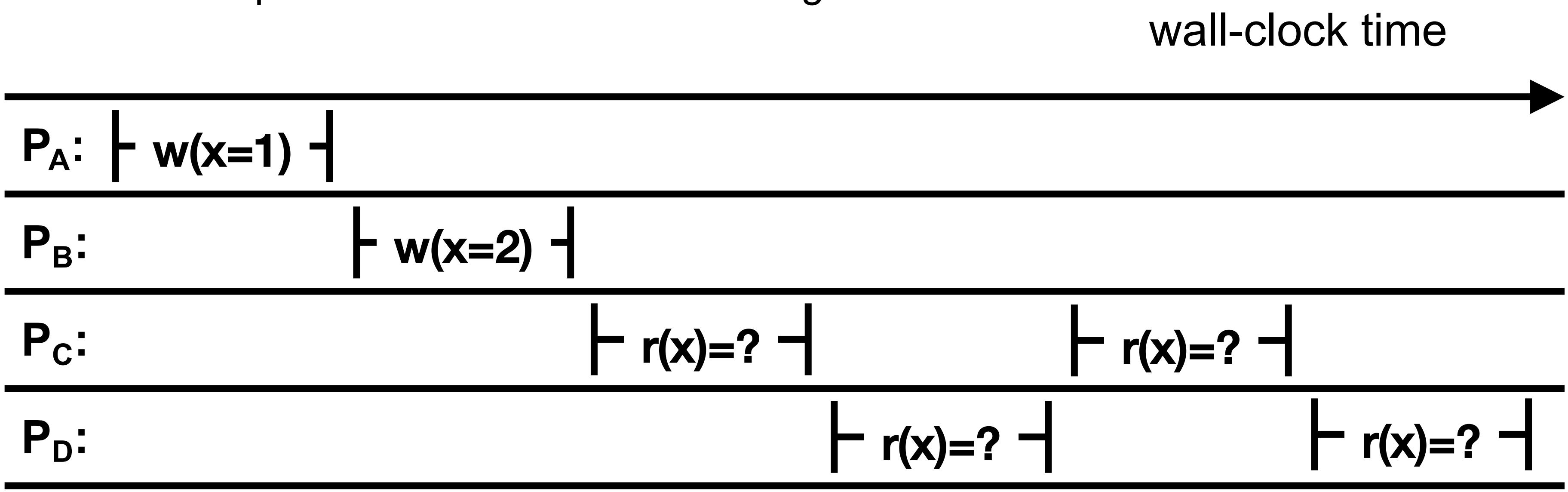
Intuitive example

- Consistency model defines what values reads are admissible



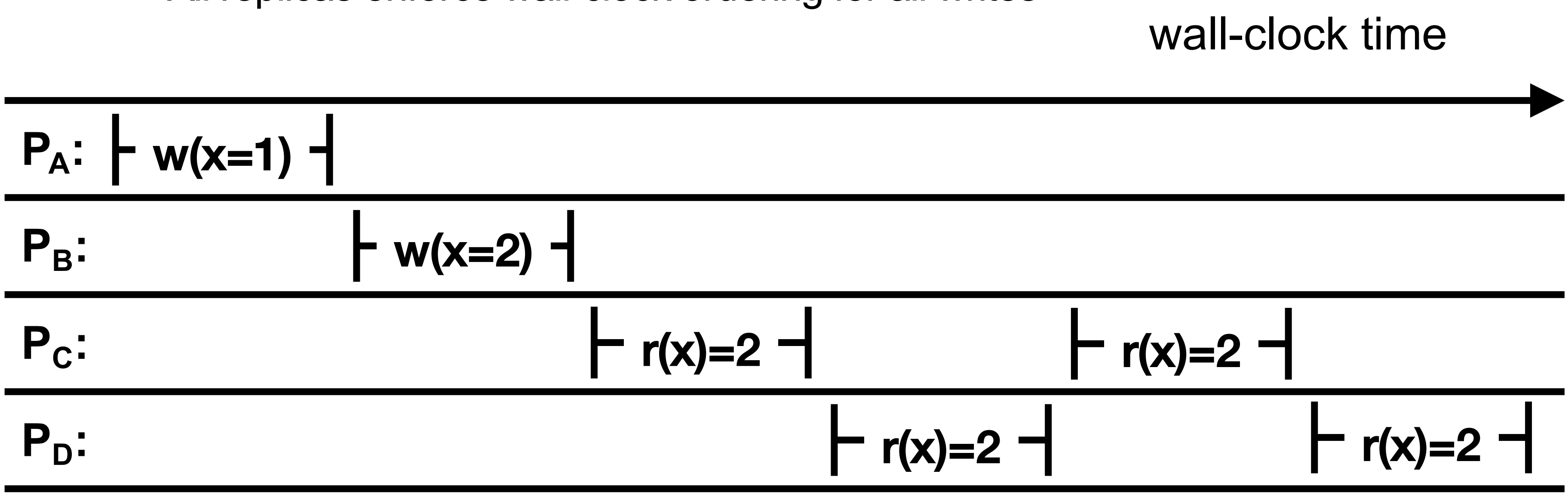
Linearizability

- 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 (i.e., a read returns the value that was last written)
  - All replicas enforce wall-clock ordering for all writes



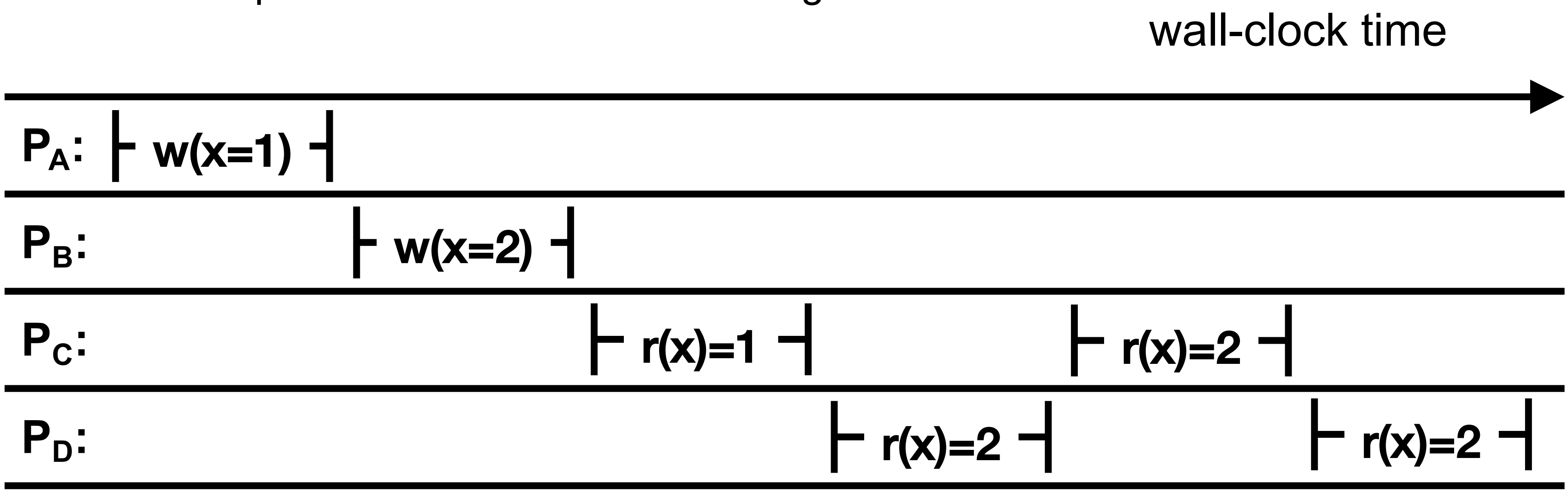
Linearizability: YES

- 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 (i.e., a read returns the value that was last written)
  - All replicas enforce wall-clock ordering for all writes



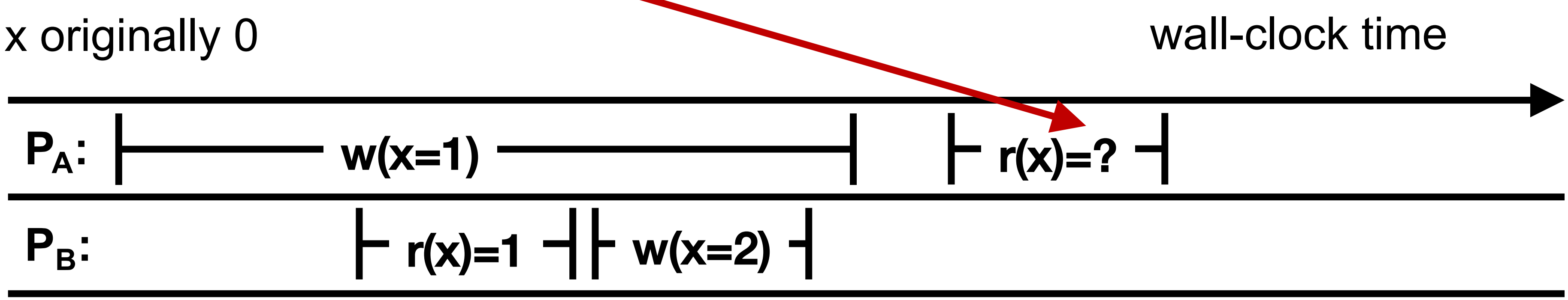
Linearizability: NO

- 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 (i.e., a read returns the value that was last written)
  - All replicas enforce wall-clock ordering for all writes



Linearizability: Quiz

- If the execution is linearizable, what does P_A read here?



P_A sees the latest write that took effect on the system (x=2)

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
- Single machine processing incoming requests one at a time also 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

Stronger vs weaker consistency

- Stronger consistency models

  + Easier to write applications

  - More guarantees for the system to ensure

    Results in performance trade-offs

- Weaker consistency models

  - Harder to write applications

  + Fewer guarantees for the system to ensure

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

Sequential consistency

- All replicas execute operations in some total order

- That total order preserves the process ordering between operations
  - If process P issues operation A before operation B, then A is order before B by the process order
  - If operations A and B are done by different processes then there is no process order between them
    - (But there must be *some* total order)

Sequential Consistency ≈ "Appears to be a Single Machine"

- Single machine processes requests one by one in the order it receives them
  - Will receive requests ordered by process order in that order
  - Will receive all requests in some order

Linearizability is strictly stronger than Sequential Consistency

- Linearizability: $\exists$total order + real-time ordering
- Sequential: $\exists$total order + process ordering
  – Process ordering $\subseteq$ Real-time ordering

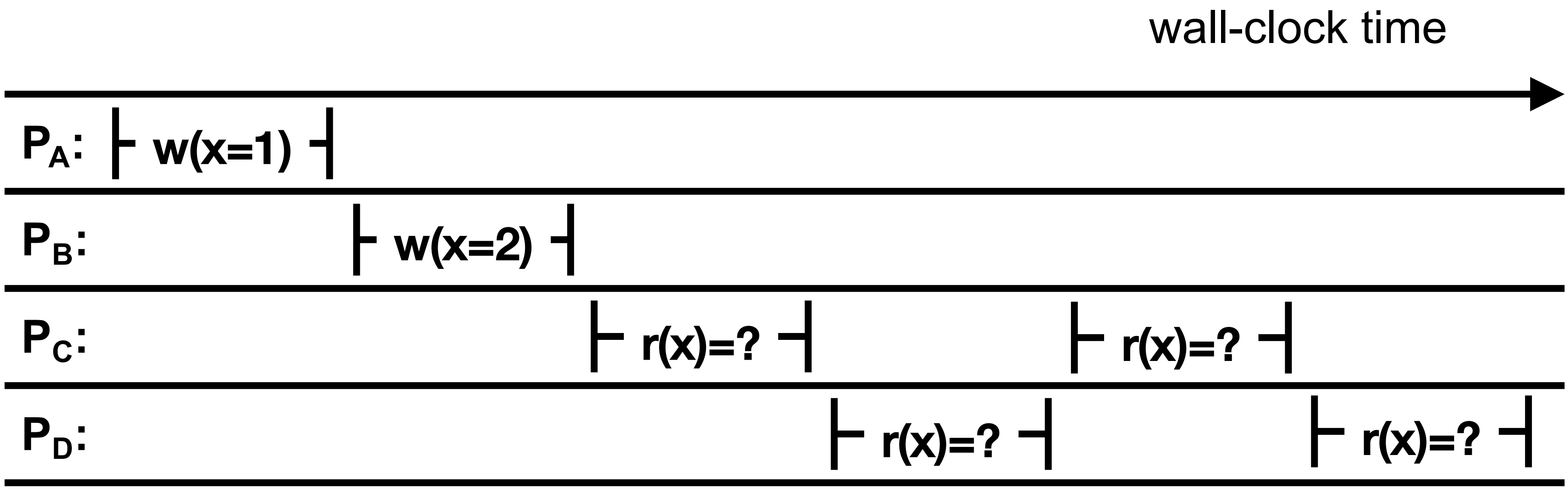
Sequential consistency

- 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**

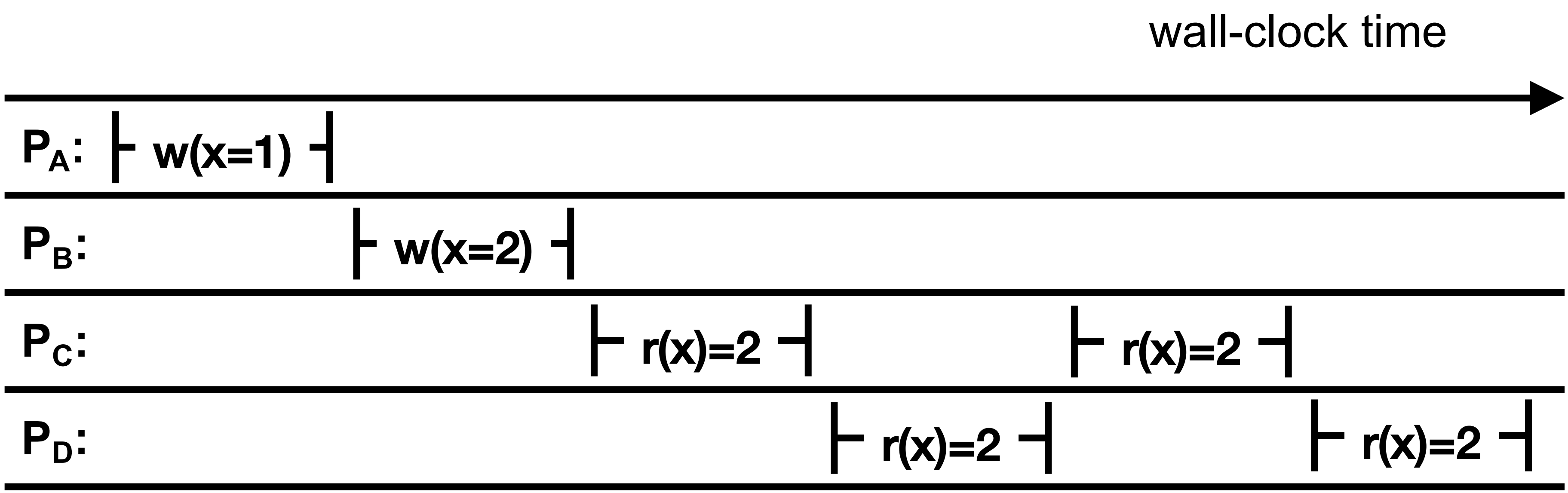
Sequential consistency

- 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

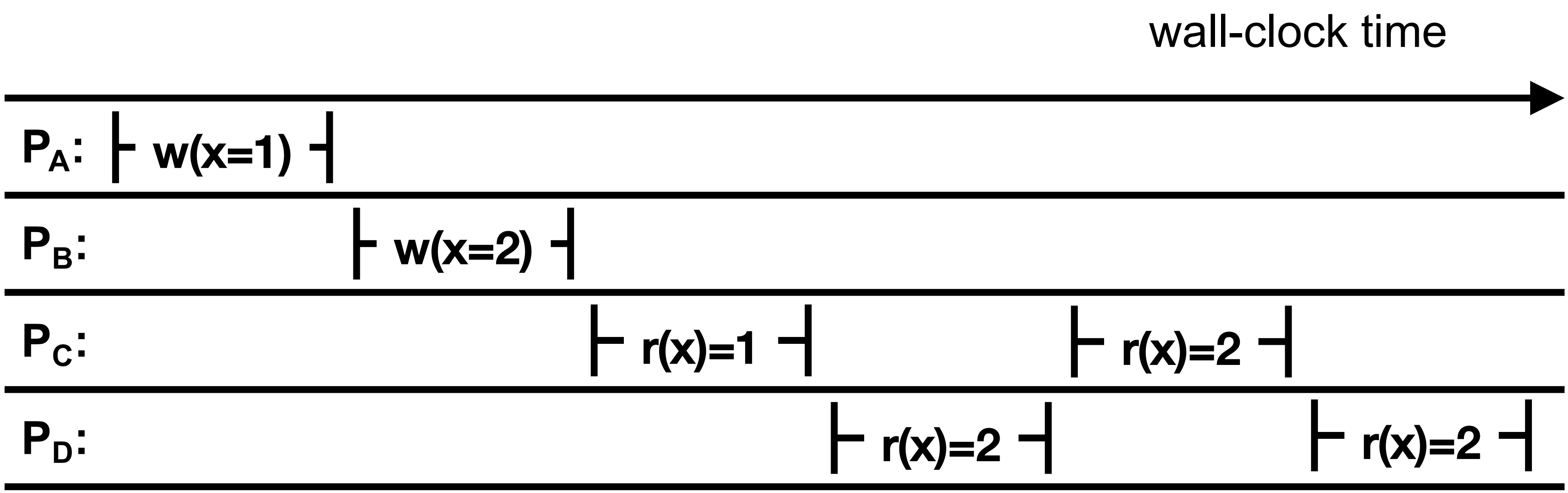
- Any execution is the same as if all read/write ops were executed in **some global ordering**, and the ops of each client process appear in the **program order**
- Therefore:
  - Reads may be stale in terms of real time, but not in logical time
  - Writes are totally ordered according to logical time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



Also valid with linearizability

Sequential consistency: YES

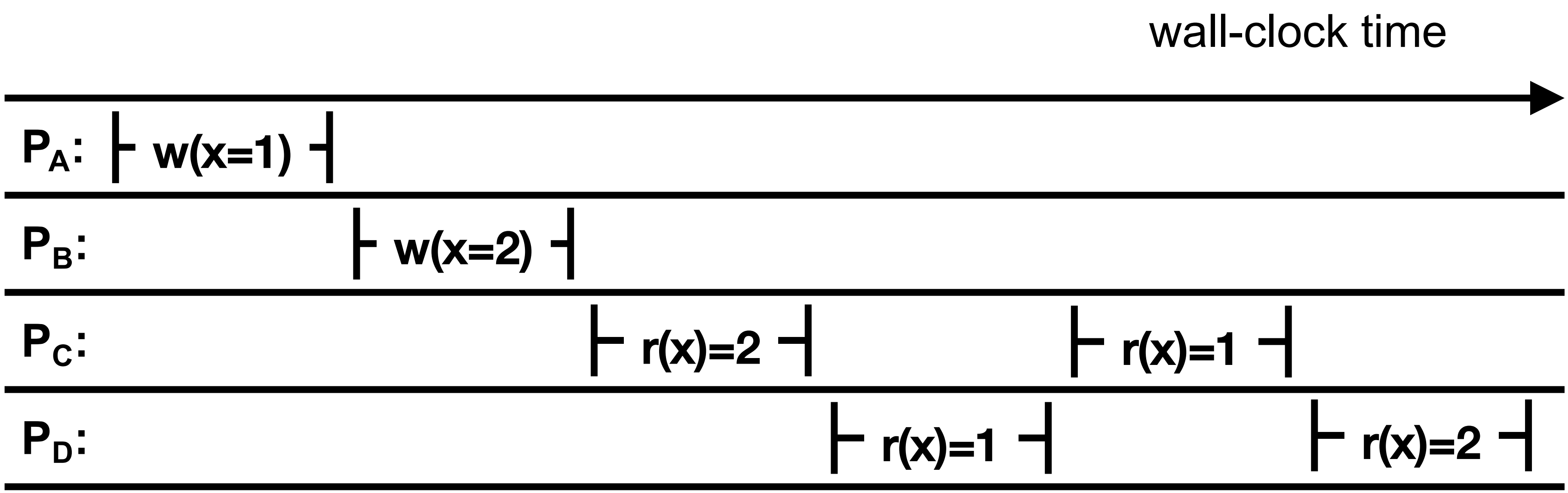
- Any execution is the same as if all read/write ops were executed in **some global ordering**, and the ops of each client process appear in the **program order**
- Therefore:
  - Reads may be stale in terms of real time, but not in logical time
  - Writes are totally ordered according to logical time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



Not valid with linearizability

Sequential consistency: NO

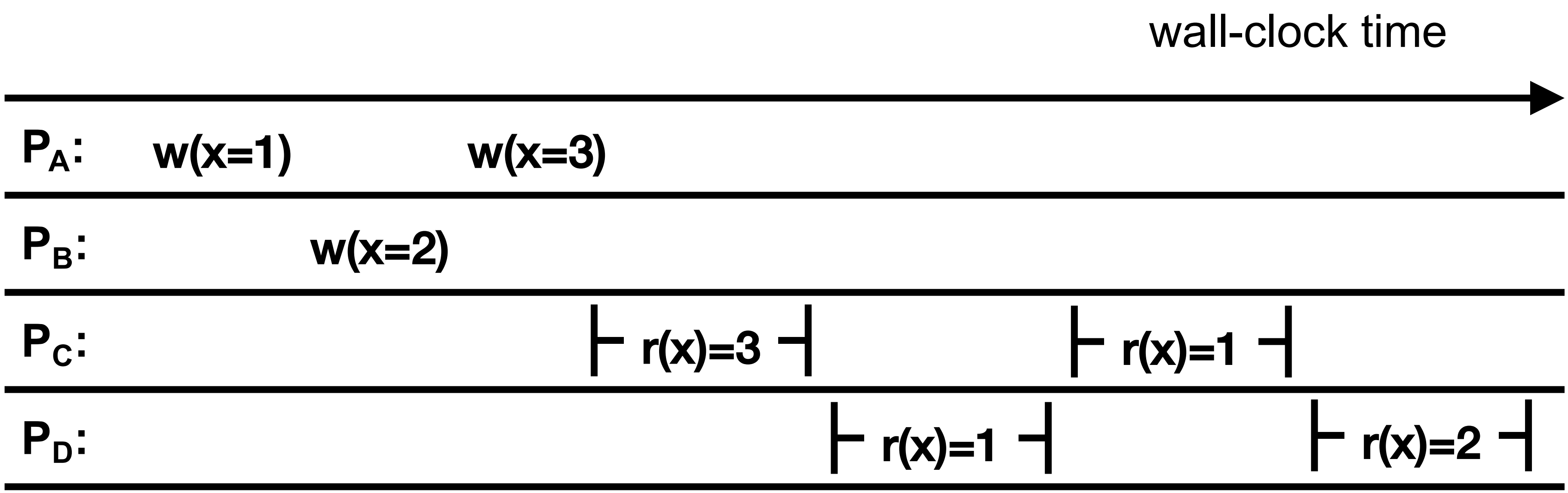
- Any execution is the same as if all read/write ops were executed in **some global ordering**, and the ops of each client process appear in the **program order**
- Therefore:
  - Reads may be stale in terms of real time, but not in logical time
  - Writes are totally ordered according to logical time 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=3), r(x)=3, r(x)=1, w(x=2) doesn't preserve P_A's ordering

Consistency hierarchy

Linearizability	e.g., RAFT
Sequential Consistency	
Causal+ Consistency	e.g., Bayou
Eventual Consistency	e.g., Dynamo

Causal+ Consistency

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

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

Causal Consistency

1. Writes that are **potentially** causally related must be seen by all processes in same order

2. Concurrent writes may be seen in a different order on different processes

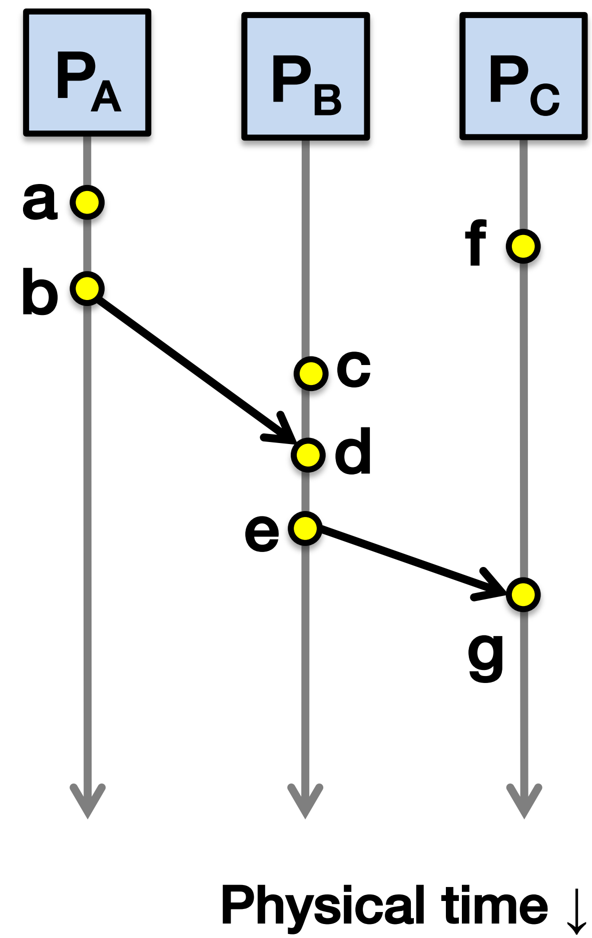
- Concurrent: Ops not causally related

Causal Consistency

1. Writes that are **potentially** causally related must be seen by all processes in same order

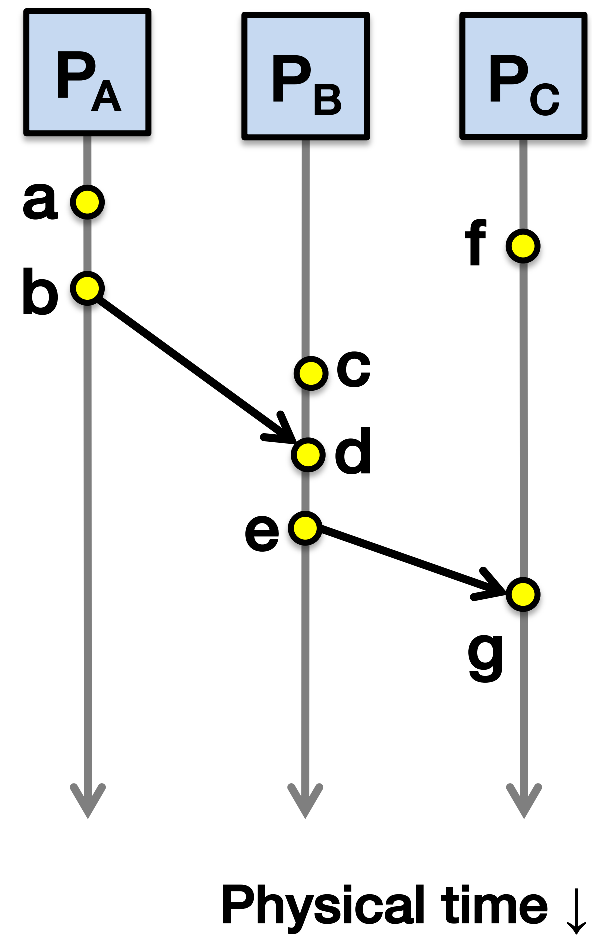
2. Concurrent writes may be seen in a different order on different processes

- Concurrent: Ops not causally related



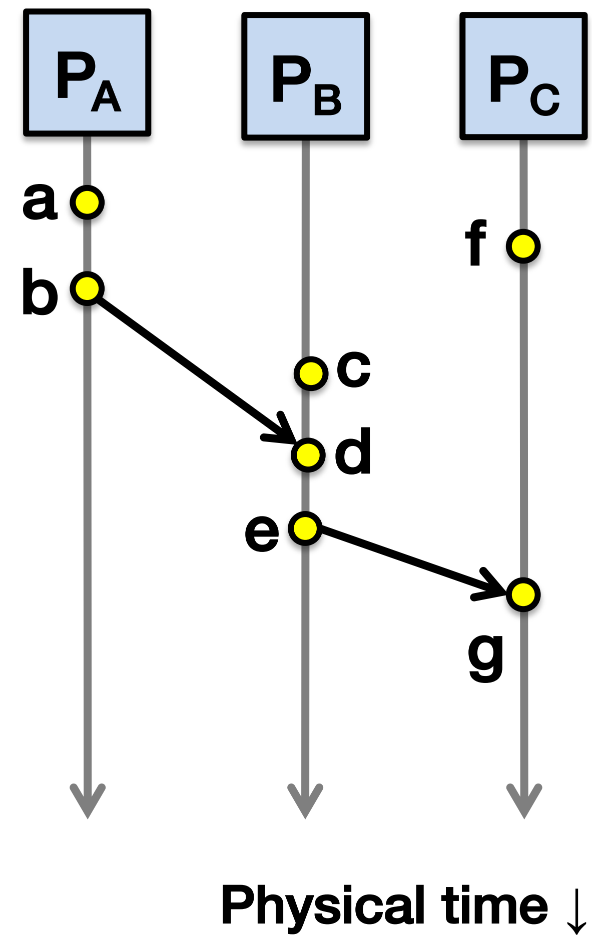
Causal Consistency

Operations	Concurrent?
a, b	
b, f	
c, f	
e, f	
e, g	
a, c	
a, e	



Causal Consistency

Operations	Concurrent?
a, b	N
b, f	Y
c, f	Y
e, f	Y
e, g	N
a, c	Y
a, e	N



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) ⊣

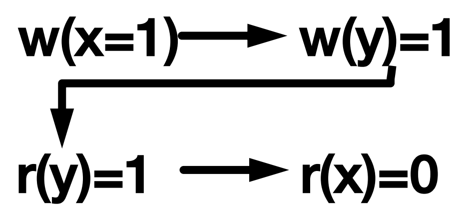
P_B ⊢ r(y)=1 ⊣ ⊢ r(x)=0 ⊣

√ **Eventual**

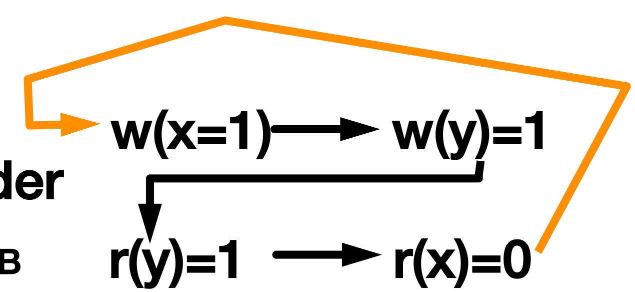
As long as P_B eventually would see r(x)=1 this is fine

X **Causal+**

Happens Before Ordering



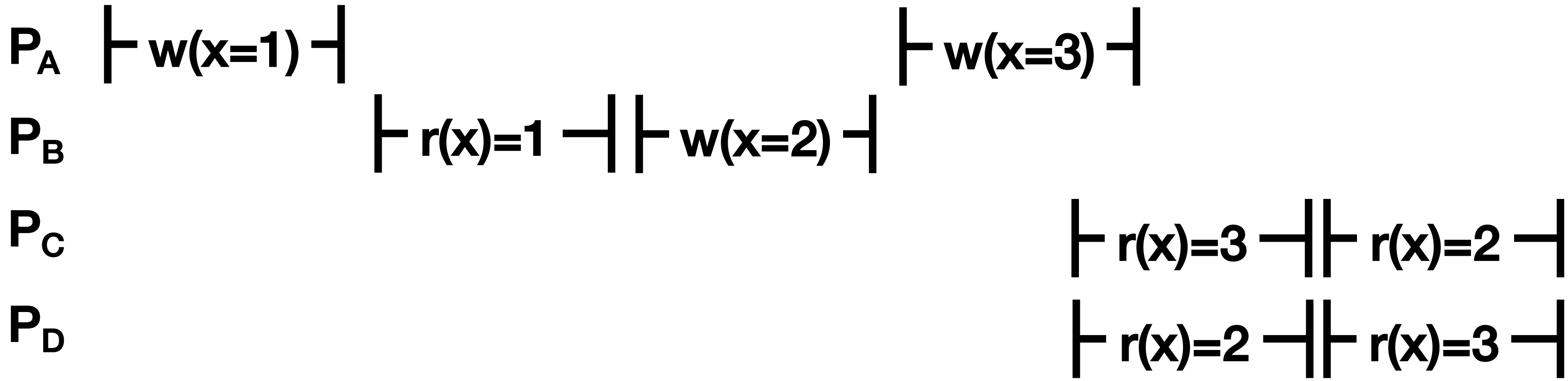
No Order for P_B



Summary: Consistency hierarchy

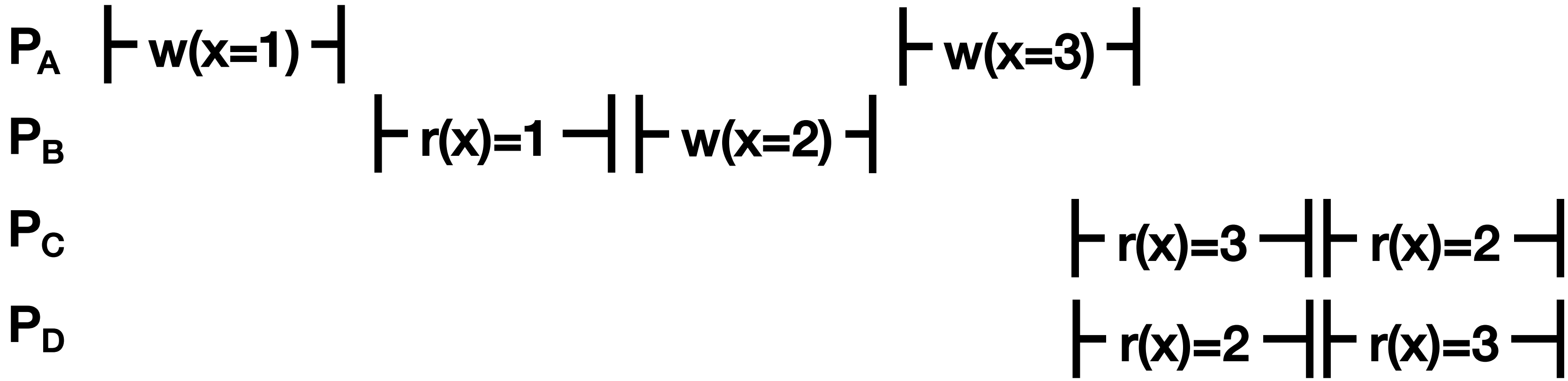
Linearizability	e.g., RAFT
↓	
Sequential Consistency	
↓	
Causal+ Consistency	e.g., Bayou
↓	
Eventual Consistency	e.g., Dynamo

Causal Consistency: Quiz



- Valid under causal consistency
- **Why?** x=3 and x=2 are concurrent
  - So all processes don't (need to) see them in same order
- P_C and P_D read the values '1' and '2' in order as potentially causally related. No 'causality' for '3'.

Sequential Consistency: Quiz



- Invalid under sequential consistency
- Why? P_C and P_D see 2 and 3 in different order
- But fine for causal consistency
  - 2 and 3 are not causally related

Causal Consistency

P_A ├ w(x=1) ┤

P_B ├ r(x)=1 ┤├ w(x=2) ┤

P_C ├ r(x)=2 ┤├ r(x)=1 ┤

P_D ├ r(x)=1 ┤├ r(x)=2 ┤

x x=2 happens after x=1

Causal Consistency

P_A ├ w(x=1) ┤

P_B ├ w(x=2) ┤

P_C ├ r(x)=2 ┤├ r(x)=1 ┤

P_D ├ r(x)=1 ┤├ r(x)=2 ┤

✓ P_B doesn't read value of 1 before writing 2

Visualization of linearizability ☺

- Nice way to see and think when a certain execution is / isn't allowed in linearizability

https://mwhittaker.github.io/consistency_in_distributed_systems/2_cap.html

- Also check out:

  https://mwhittaker.github.io/blog/visualizing_linearizability/

  https://muratbuffalo.blogspot.com/2021/10/linearizability.html