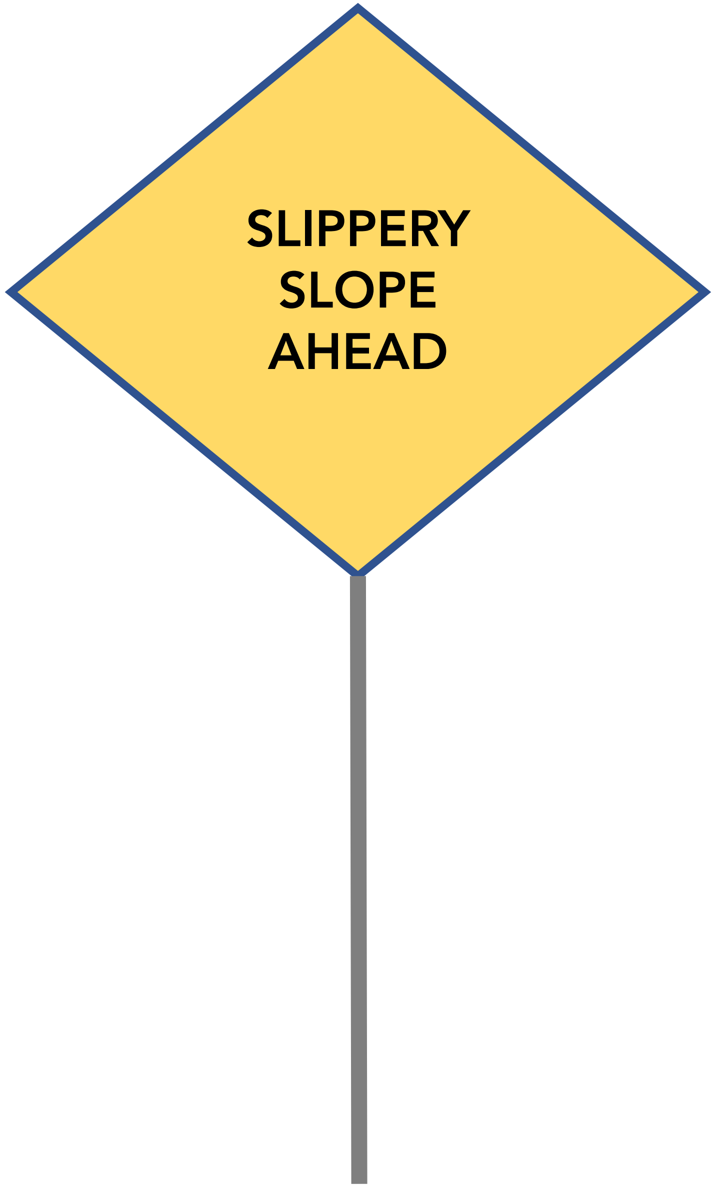
CS 582: Distributed Systems

Consistency Models and CAP

Dr. Zafar Ayyub Qazi

Fall 2024

Acknowledgments: Slides use some material from Ryan Stutsman (University of Utah)


SLIPPERY
SLOPE
AHEAD

Agenda

- Consistency Models
    - Linearizability
    - Sequential Consistency
    - Causal Consistency
    - Eventual Consistency
- CAP Theorem

Specific learning outcomes

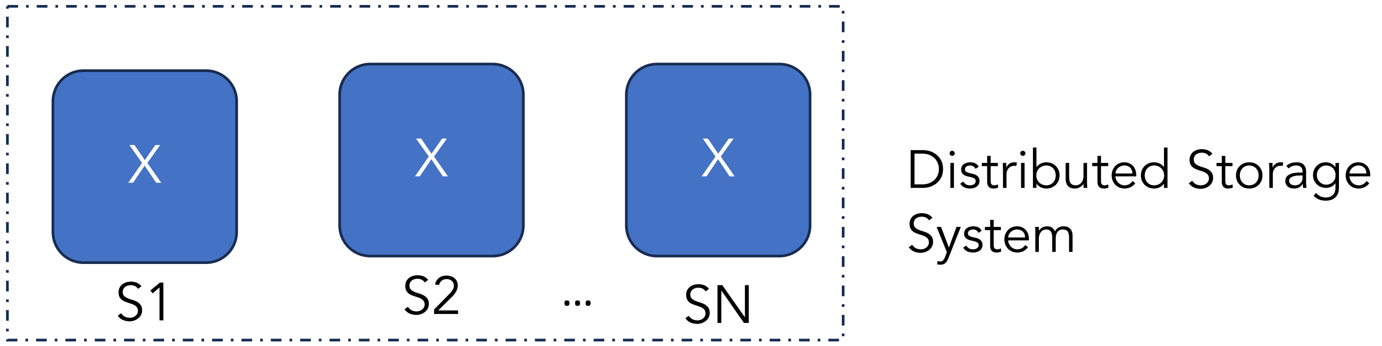
By the end of today's lecture, you should be able to:

- ❑ Define and explain linearizability, sequential consistency, causal consistency, and eventual consistency
- ❑ Compare and contrast linearizability, sequential consistency, causal consistency, and eventual consistency, in terms of their guarantees and tradeoffs
- ❑ Given a scenario of event orderings in a distributed system, determine whether the system exhibits linearizability, sequential consistency, causal consistency, or eventual consistency
- ❑ Analyze the implications of different consistency models on the design and performance of distributed systems
- ❑ Explain the CAP theorem and its fundamental tradeoffs in distributed systems
- ❑ Evaluate the applicability of the CAP theorem to various systems distributed system designs and use case

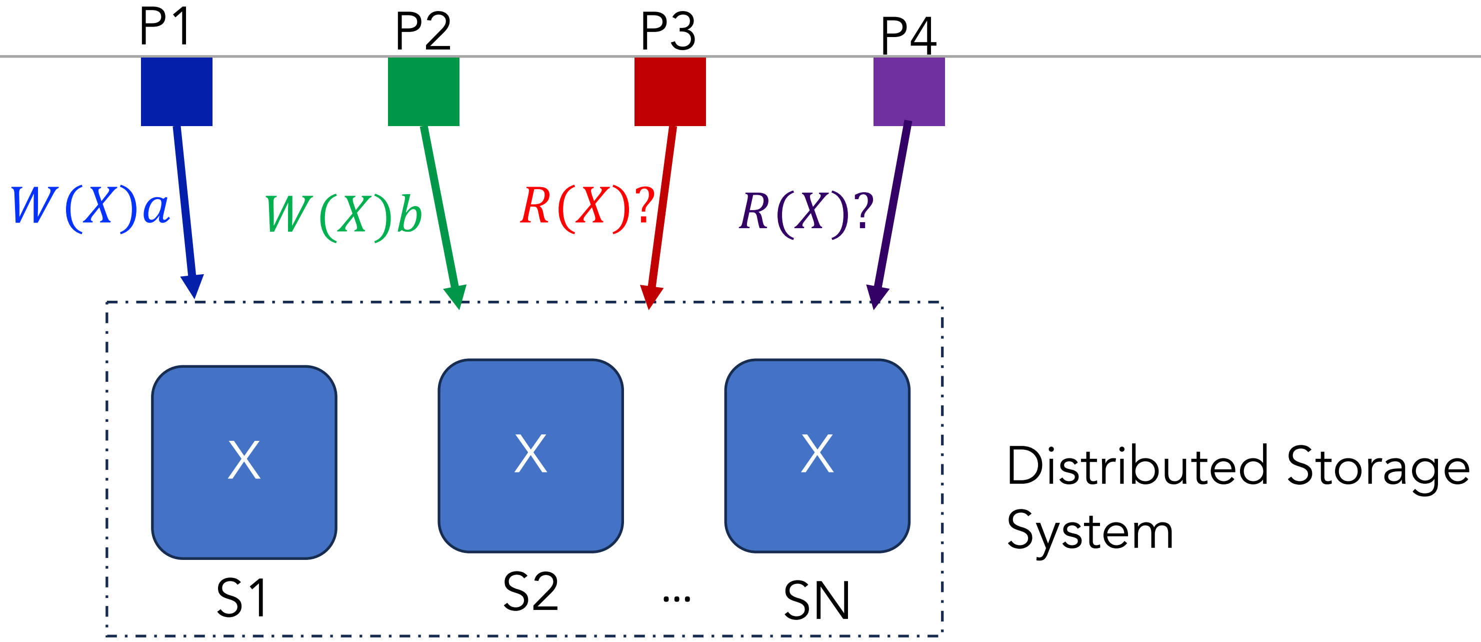
Recap: Consistency Model

- Contract btw a distributed system and applications that run on it
- A consistency model is a set of guarantees made by the dist. system
- E.g., Linearizability, Sequential Consistency, Causal Consistency, Eventual Consistency
- Variations boil down to:
  - Allowable staleness of reads
  - The ordering of writes across all. replicas

Example

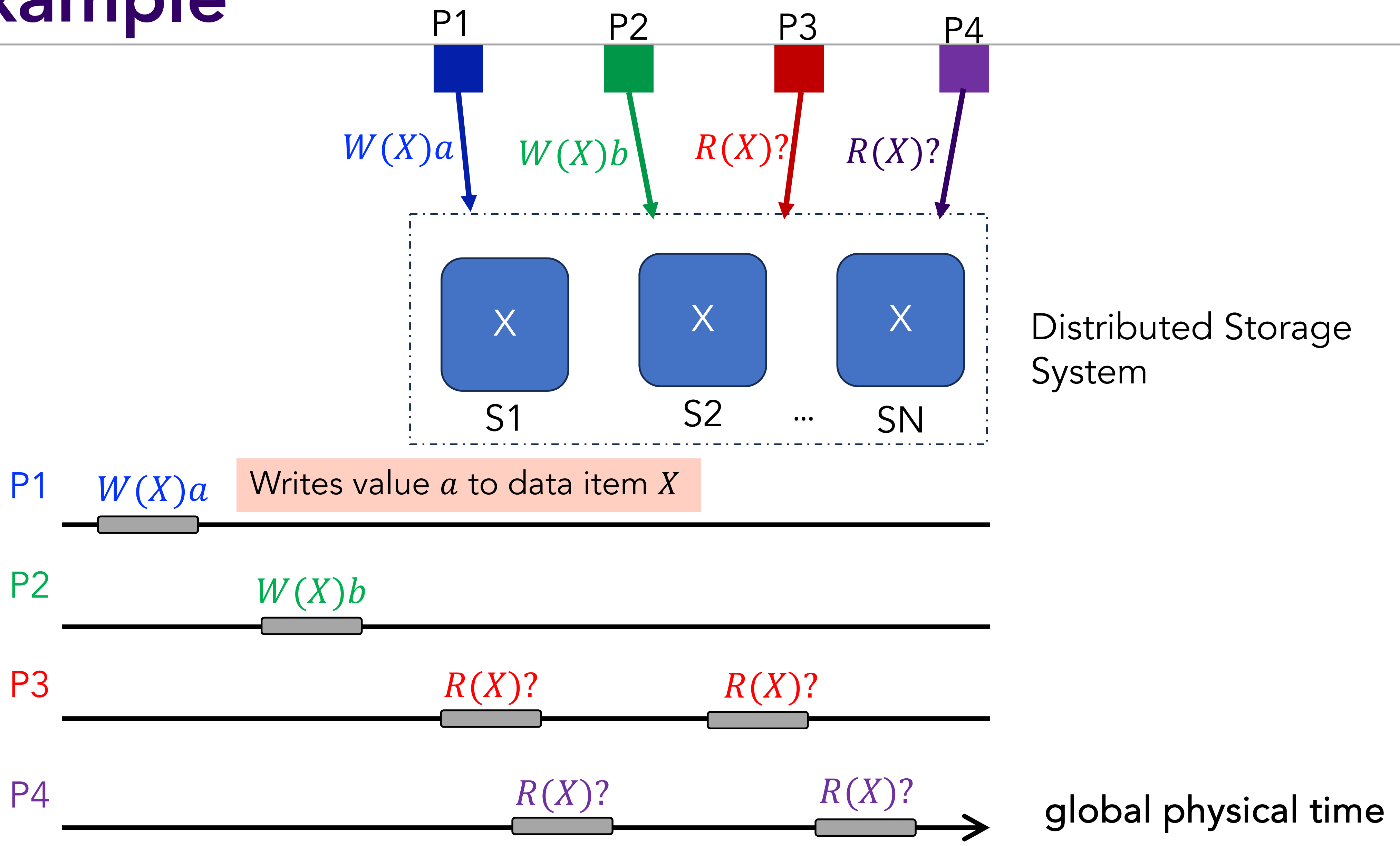


Example



P1
P2
P3
P4
$W(X)a$
$W(X)b$
$R(X)?$
$R(X)?$
X
X
X
S1
S2
...
SN
Distributed Storage System

Example

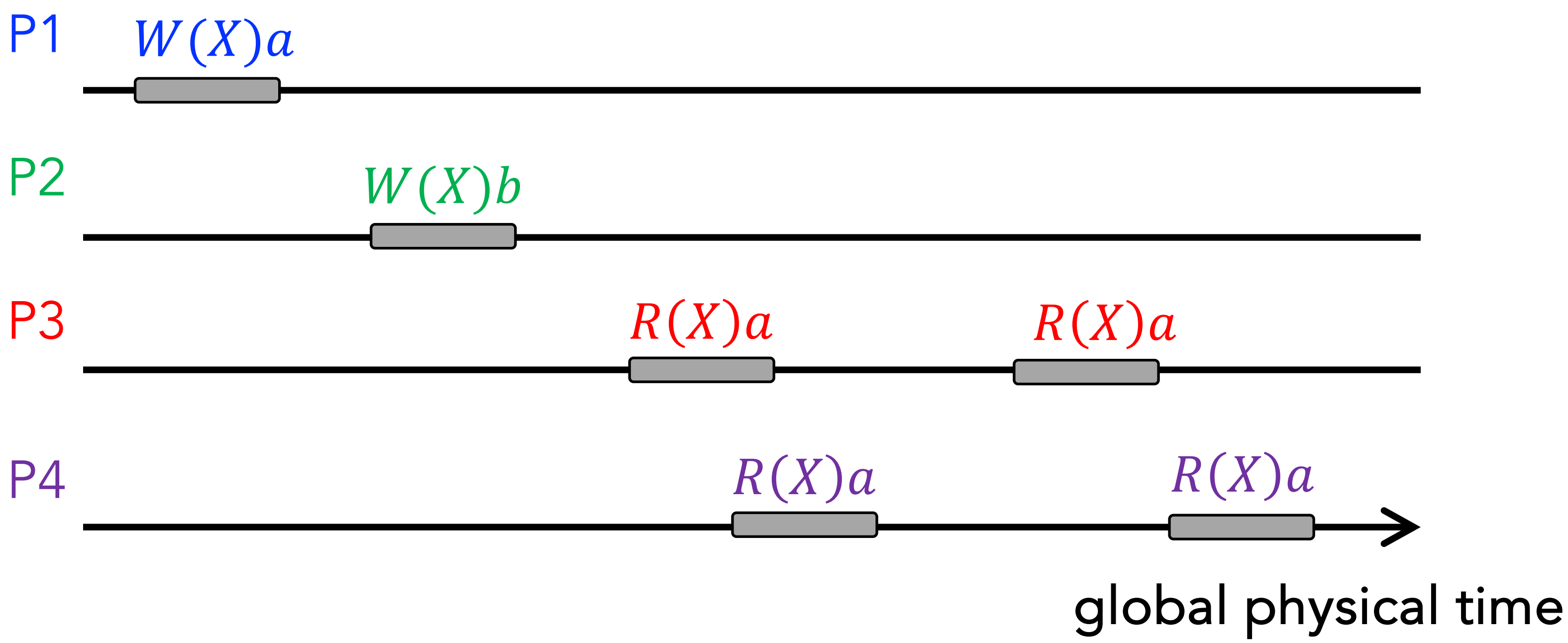


P1
P2
P3
P4
$W(X)a$
$W(X)b$
$R(X)?$
$R(X)?$
X
X
X
S1
S2
...
SN
Distributed Storage System
P1
$W(X)a$
Writes value a to data item X
P2
$W(X)b$
P3
$R(X)?$
$R(X)?$
P4
$R(X)?$
$R(X)?$
global physical time

Recap: Linearizability

- All replicas execute operations in some total order
- That total order preserves the real-time (physical-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: (Counter) Examples

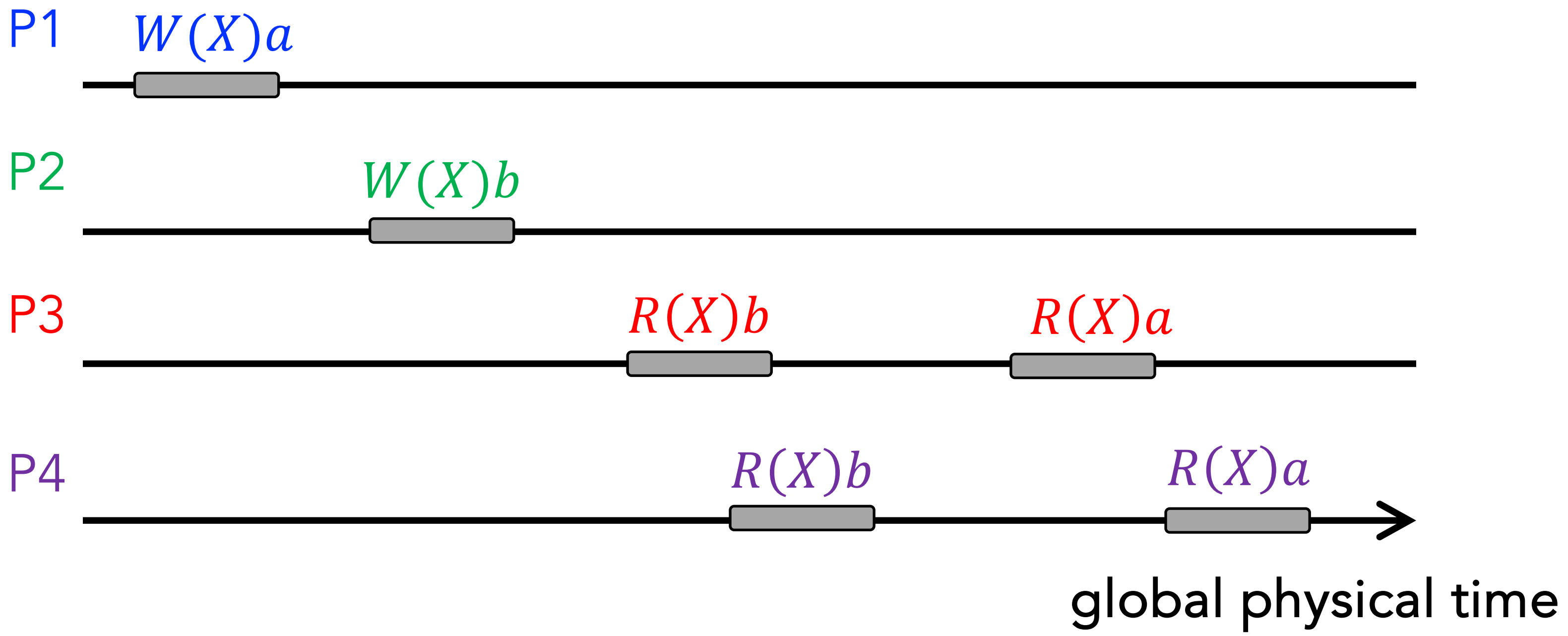


A read after W(X)b finishes must return b

If underlying storage system is linearizable, can the system return these read values?

No, these reads cannot be returned by a linearizable system

Linearizability: (Counter) Examples

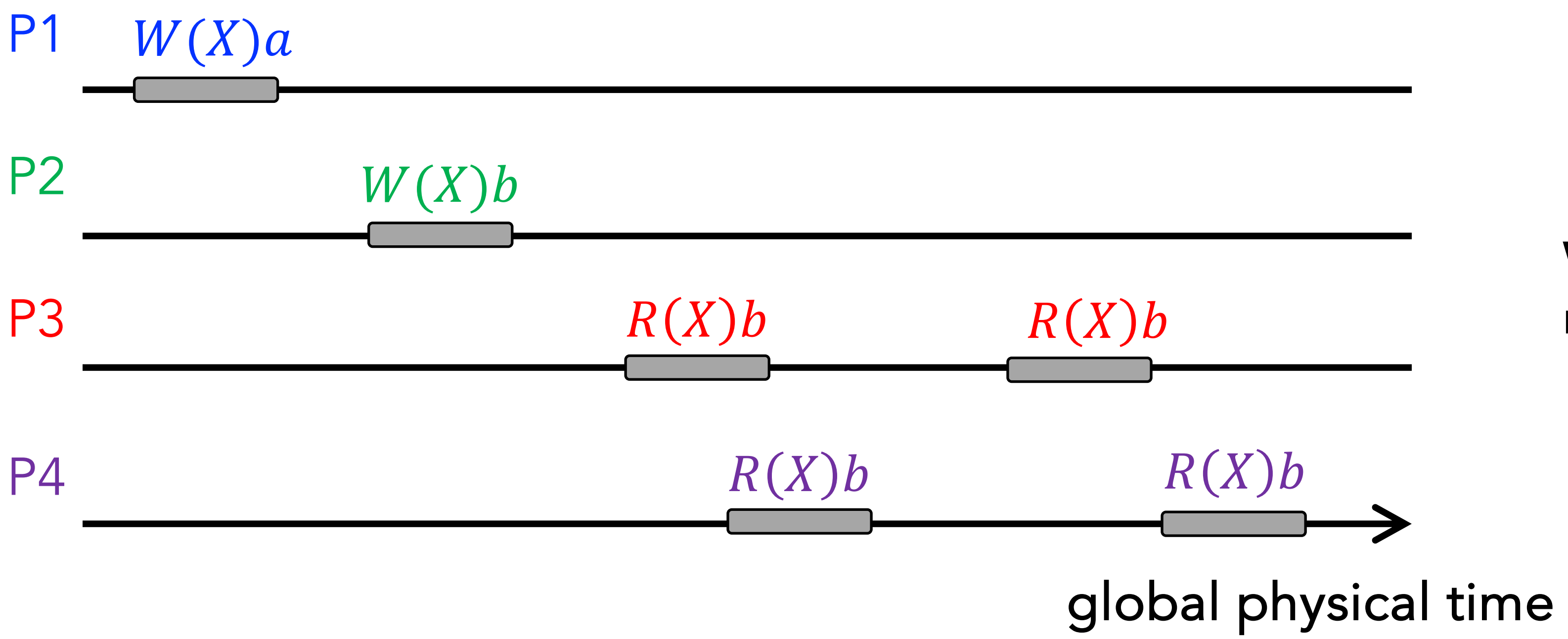


A read after W(X)b finishes must return b or newer values--- the system cannot return value a

If underlying storage system is linearizable, can the system return these read values?

No, these reads cannot be returned by a linearizable system

Linearizability: (Counter) Examples



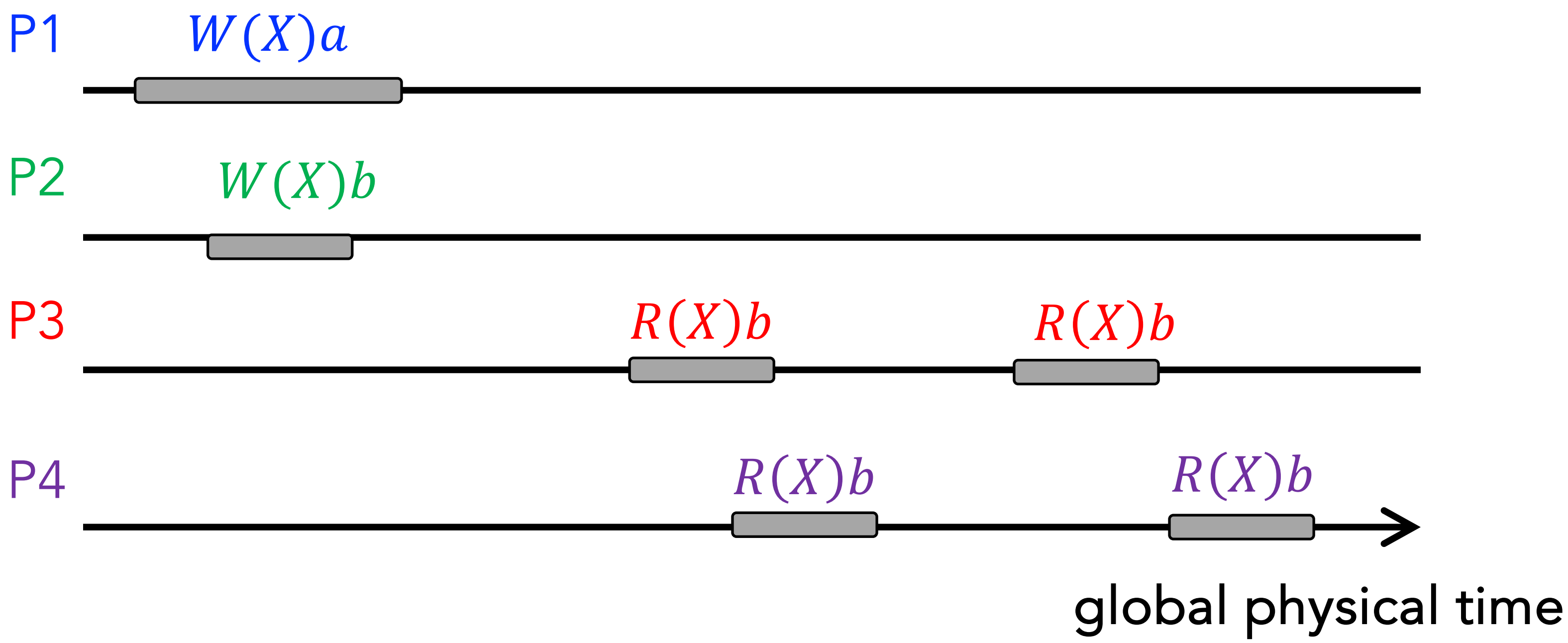
We can find a total order that respects real-time ordering

If underlying storage system is linearizable, can the system return these read values?

Yes, these reads could be returned by a linearizable system

$W(X)a, W(X)b, R(X)b, \ldots$

Linearizability: (Counter) Examples



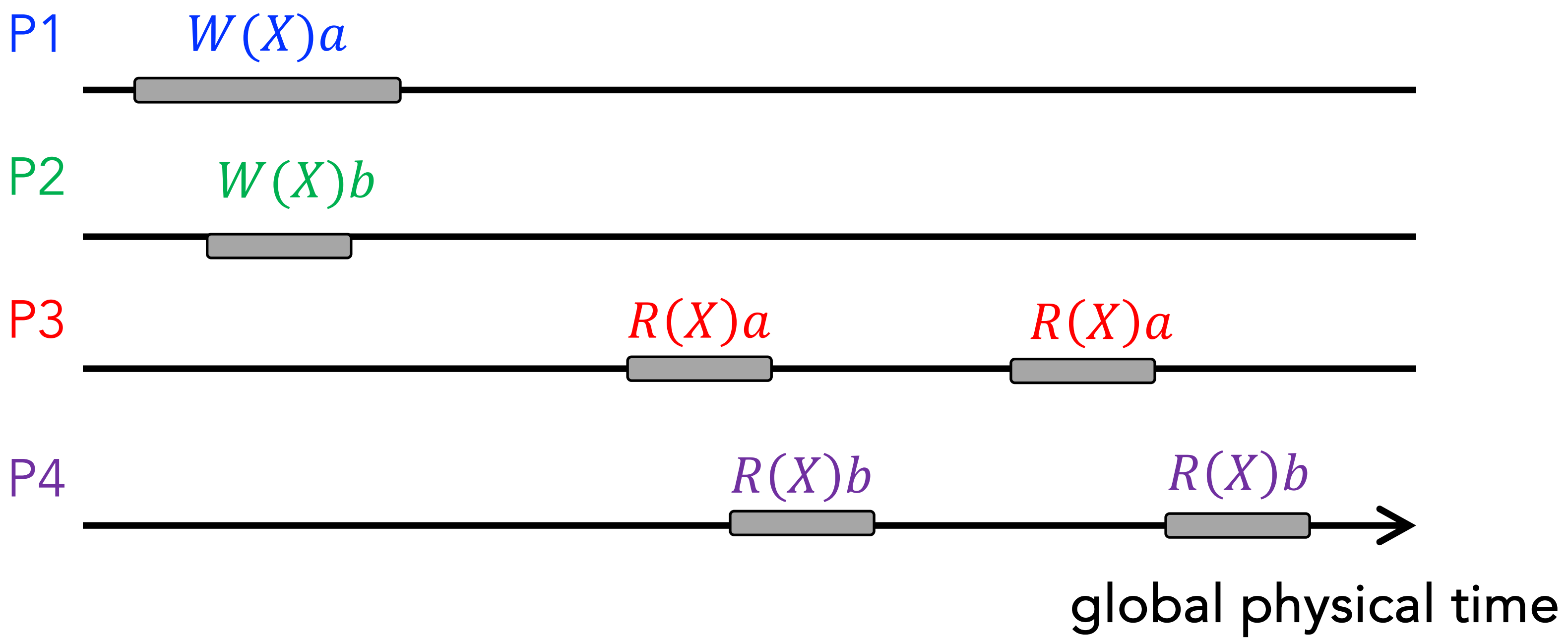
Recall: in linearizability, overlapping write operations can be ordered in any way

If underlying storage system is linearizable, can the system return these read values?

Yes, these reads could be returned by a linearizable system

$W(X)a, W(X)b, R(X)b, \ldots$

Linearizability: (Counter) Examples

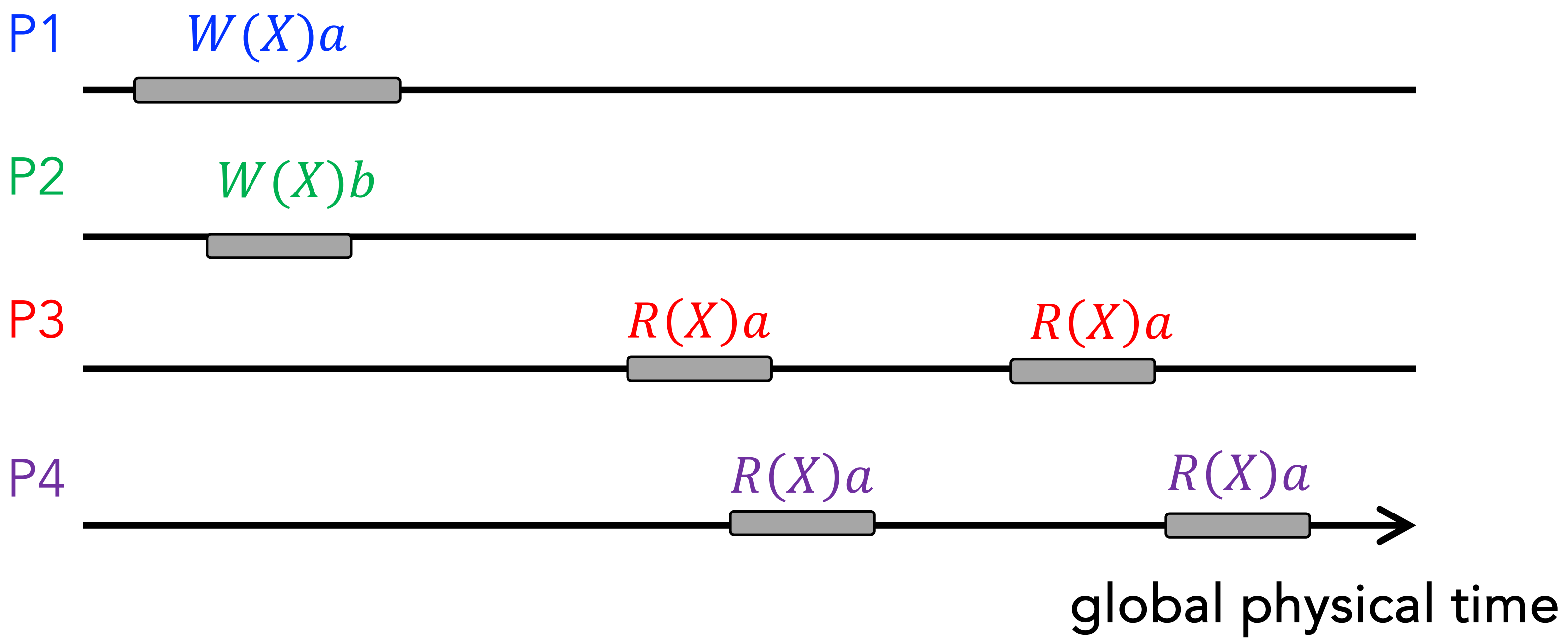


Depending on which write is committed last, that value should be returned by all subsequent reads

If underlying storage system is linearizable, can the system return these read values?

No, these reads cannot be returned by a linearizable system

Linearizability: (Counter) Examples



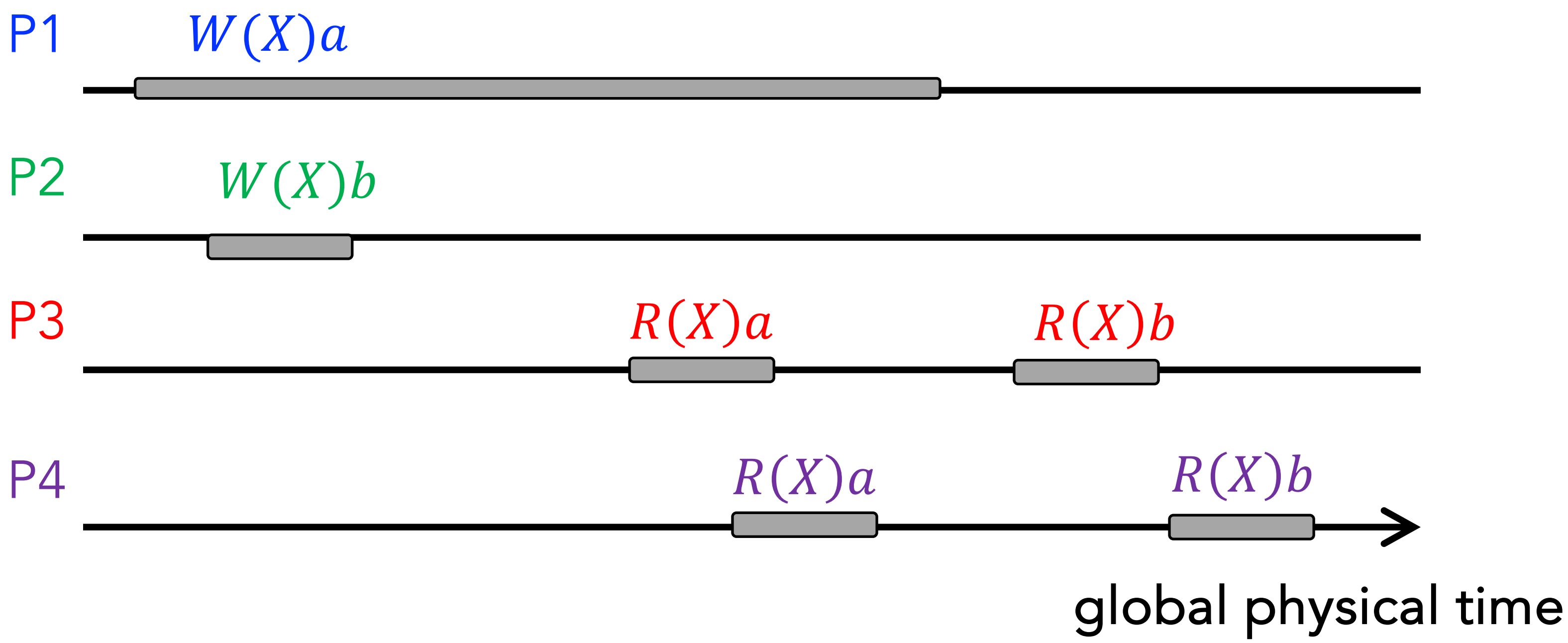
Recall: in linearizability, overlapping write operations can be ordered in any way

If underlying storage system is linearizable, can the system return these read values?

Yes, these reads could be returned by a linearizable system

$W(X)b, W(X)a, R(X)a, \ldots$

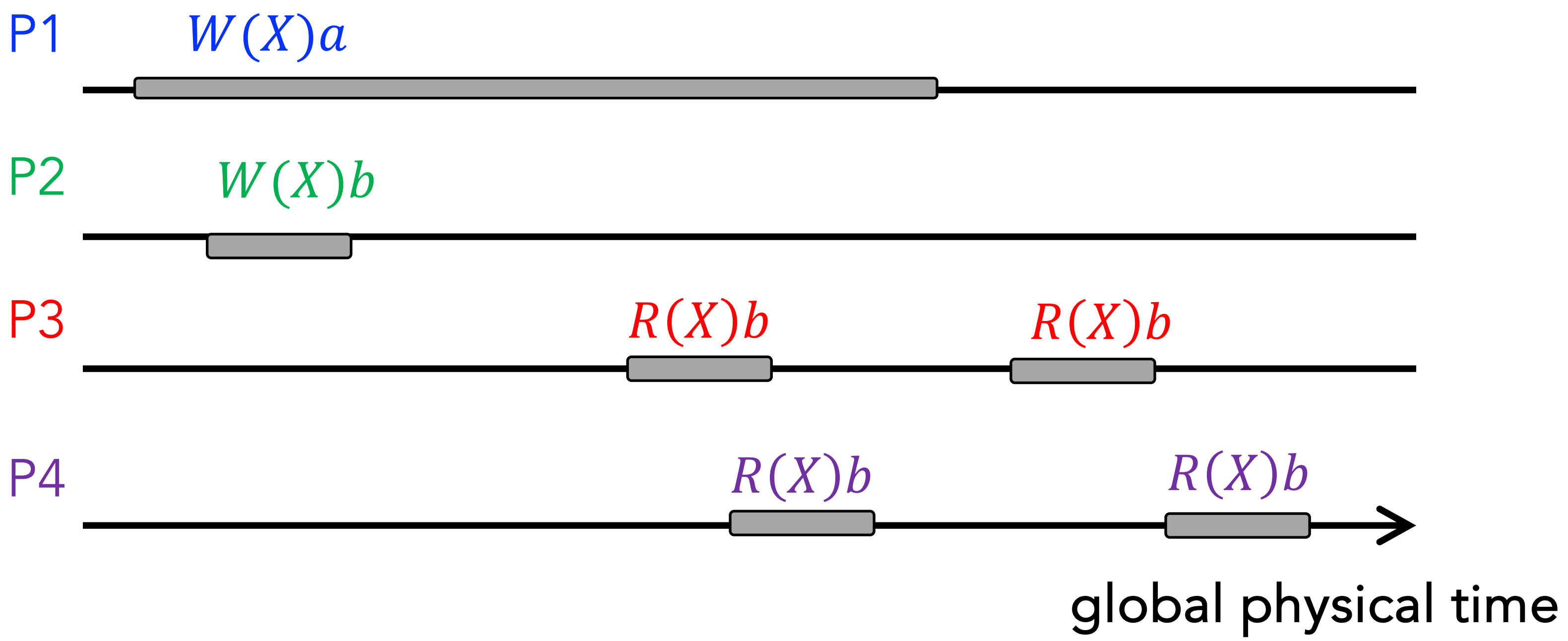
Linearizability: (Counter) Examples



If underlying storage system is linearizable, can the system return these read values?

No, these reads cannot be returned by a linearizable system

Linearizability: (Counter) Examples

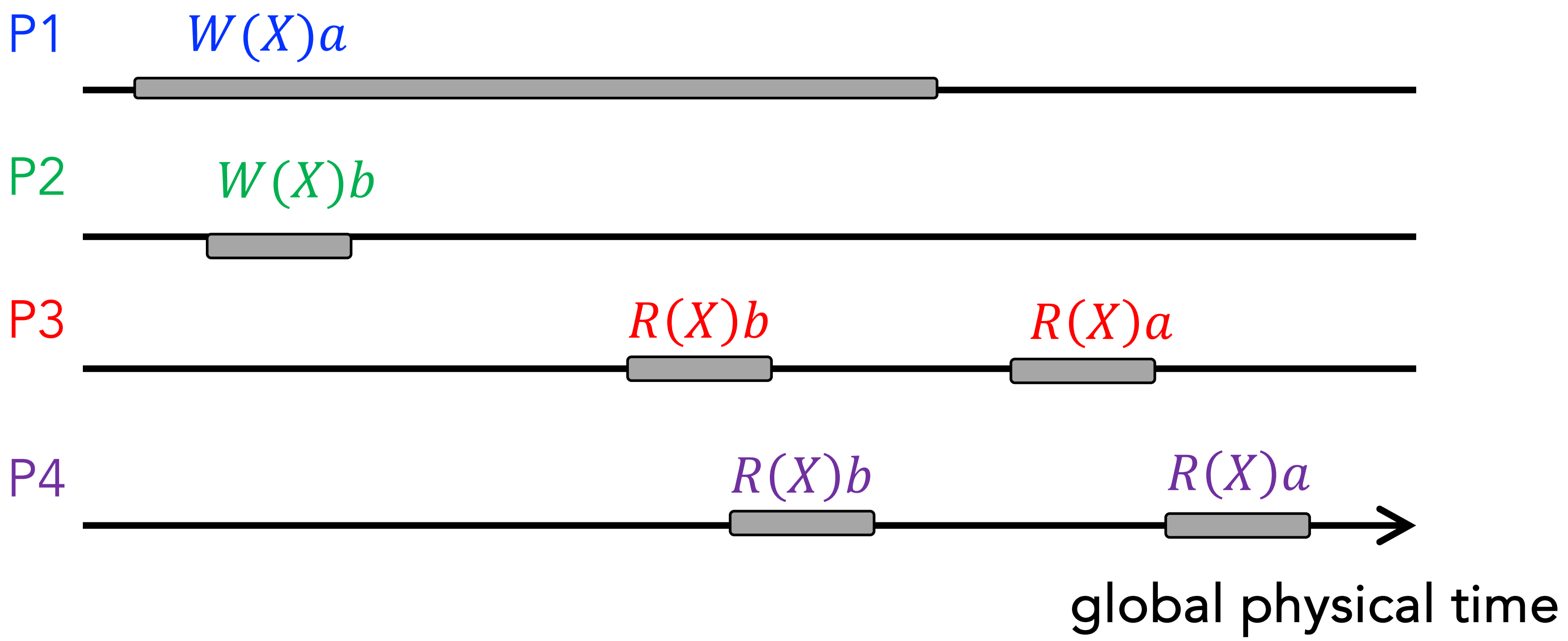


If underlying storage system is linearizable, can the system return these read values?

Yes, these reads could be returned by a linearizable system

$W(X)a, W(X)b, R(X)b, \ldots$

Linearizability: (Counter) Examples

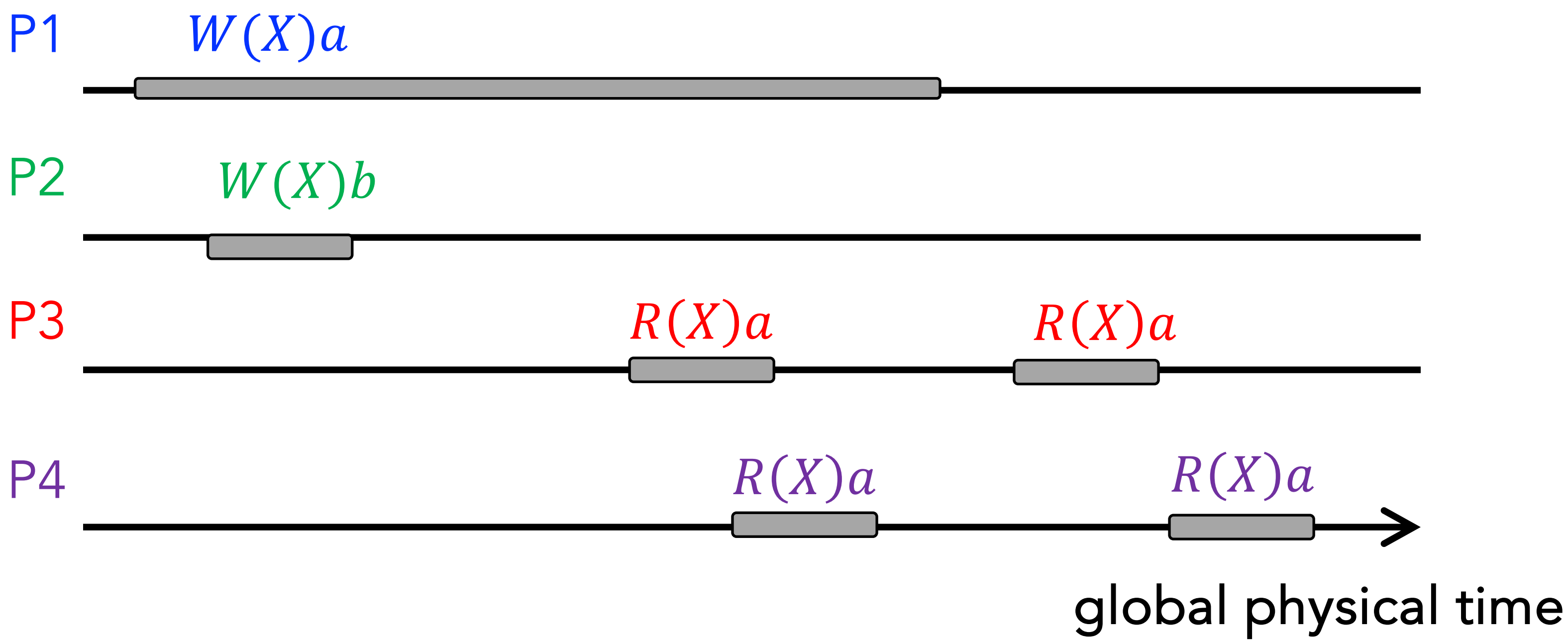


If underlying storage system is linearizable, can the system return these read values?

Yes, these reads could be returned by a linearizable system

$W(X)b, R(X)b, R(X)b, W(X)a, R(X)a, R(X)a$

Linearizability: (Counter) Examples



If underlying storage system is linearizable, can the system return these read values?

Yes, these reads could be returned by a linearizable system

$W(X)b, W(X)a, R(X)a, R(X)a, \ldots$

Linearizability: Implications

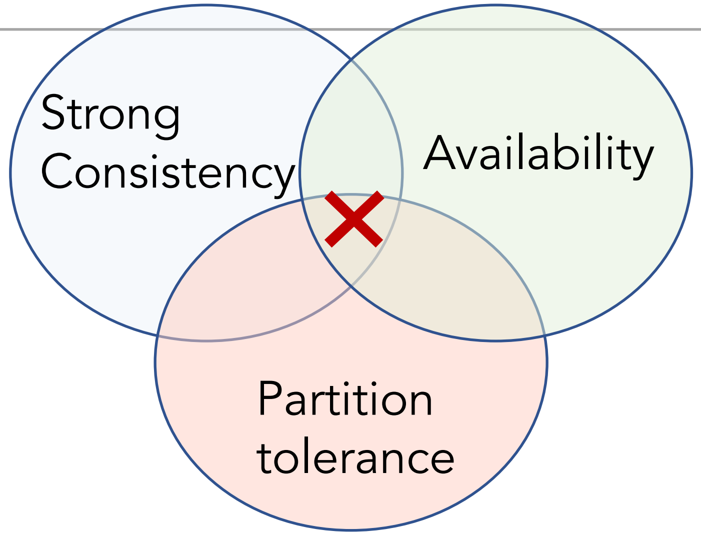
- Once a write completes, all later reads (by physical time) should return the value of that write or the value of a later write
- Once a read returns a particular value, all later reads should return that value or the value of a later write

Linearizability Tradeoffs

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

- But, performance trade-offs

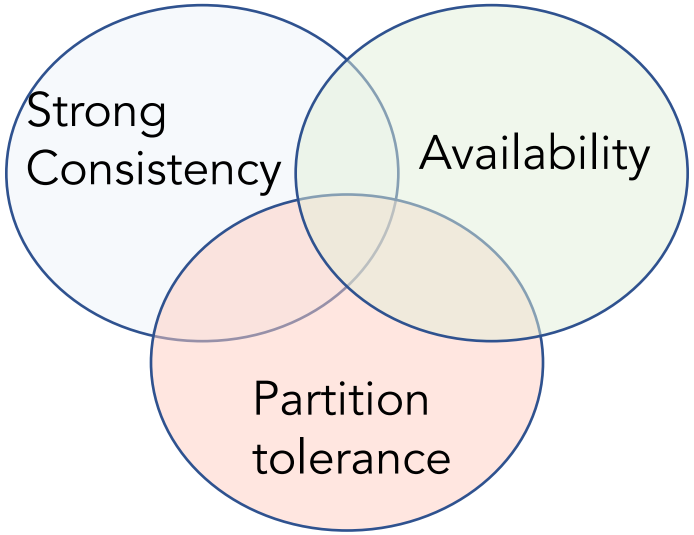
CAP Theorem



- We cannot achieve all three of:
    1. Strong Consistency
    2. Availability
    3. Partition-Tolerance

- Partition Tolerance => Network Partitions Can Happen
- Availability => All Sides of Partition Continue
- Strong Consistency => Replicas Act Like Single Machine
    - Specifically, Linearizability

Visualizing CAP



Three possible systems

1. CA (strong consistency + availability)
2. CP (strong consistency + partition tolerance)
3. AP (availability + partition tolerance)

Four potential conclusions from CAP Theoreom

Conclusion #1

- Many system designs used in early distributed relational database systems did not take into account partition tolerance (e.g., they were CA designs)

- Partition tolerance is an important property for modern systems, since network partitions become much more likely if the system is geographically distributed (as many large systems are)

Conclusion #2

- There is a tension between strong consistency and high availability during network partitions

- The CAP theorem is an illustration of the tradeoffs that occur between strong consistency guarantees and distributed computation

Conclusion #3

- There is a *tension between strong consistency and performance in normal operation*. Strong consistency/linearizability requires that nodes communicate and agree on every operation. This results in high latency during normal operation

Conclusion #4

- Somewhat indirect - that *if we do not want to give up availability during a network partition, then we need to explore whether consistency models other than strong consistency are workable for our purposes*

Next …

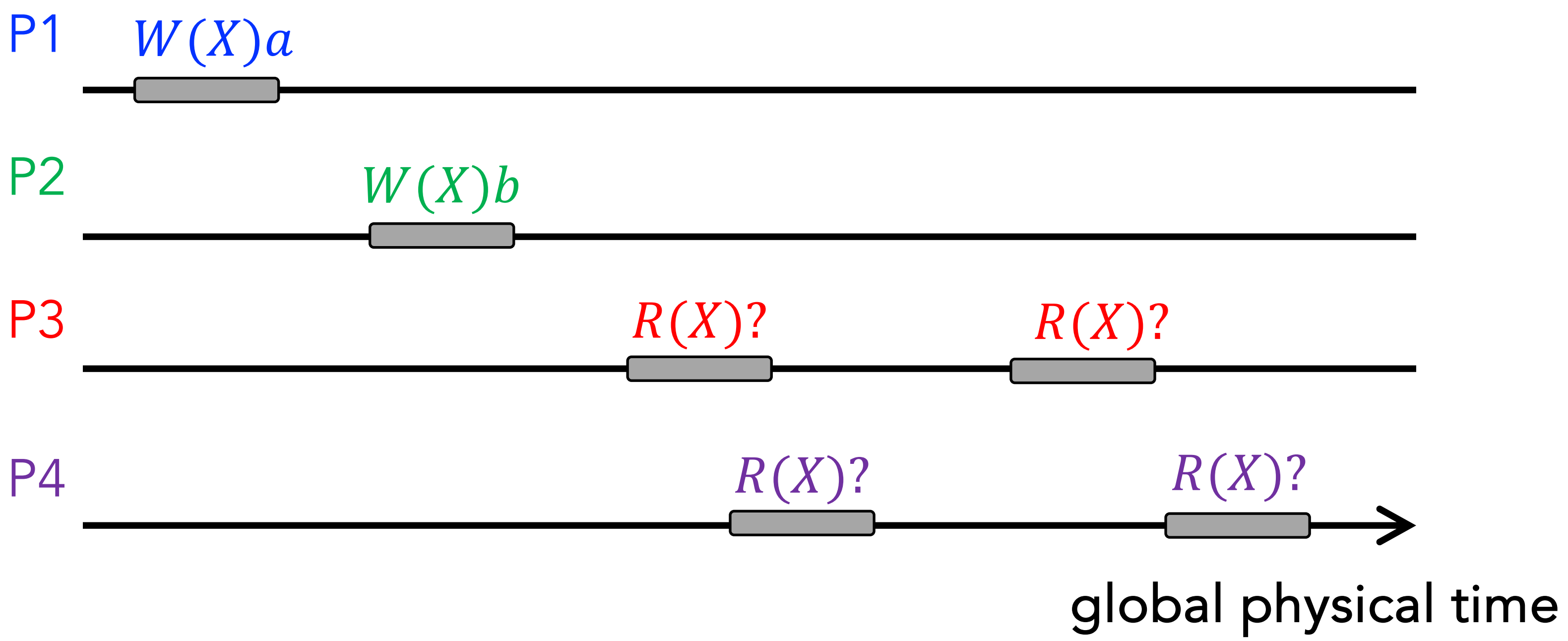
- Other consistency models

Sequential Consistency [Lamport 1979]

Sequential Consistency [Lamport 1979]

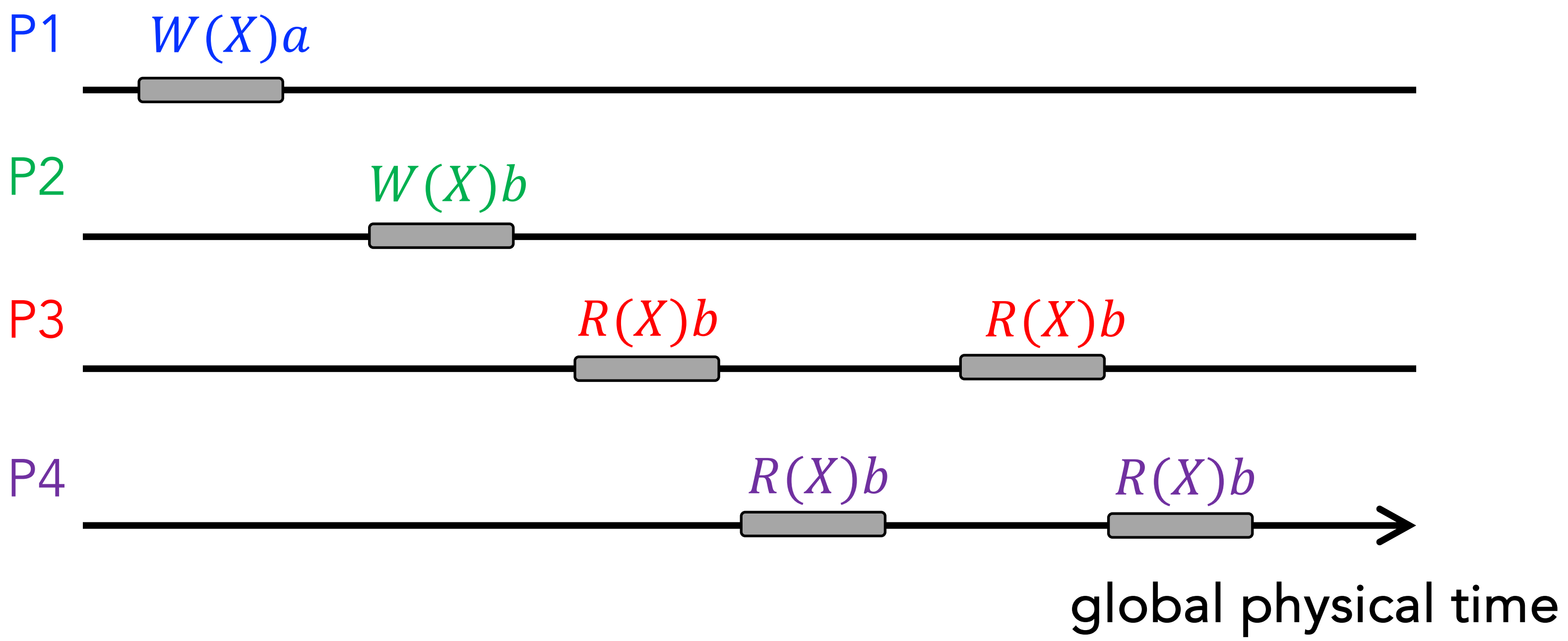
- Implies that operations appear to take place in (1) some total order, and that order is (2) consistent with the order of operations on each individual client process
- Therefore:
  - Reads may be stale in terms of real time, but not in logical time
  - Writes are totally ordered according to logical time across all replicas
- Key difference from linearizability
  - May not preserve real time ordering

Sequential Consistency: (Counter) Examples



If underlying storage system is sequentially consistent, what can these reads return?

Sequential Consistency: (Counter) Examples

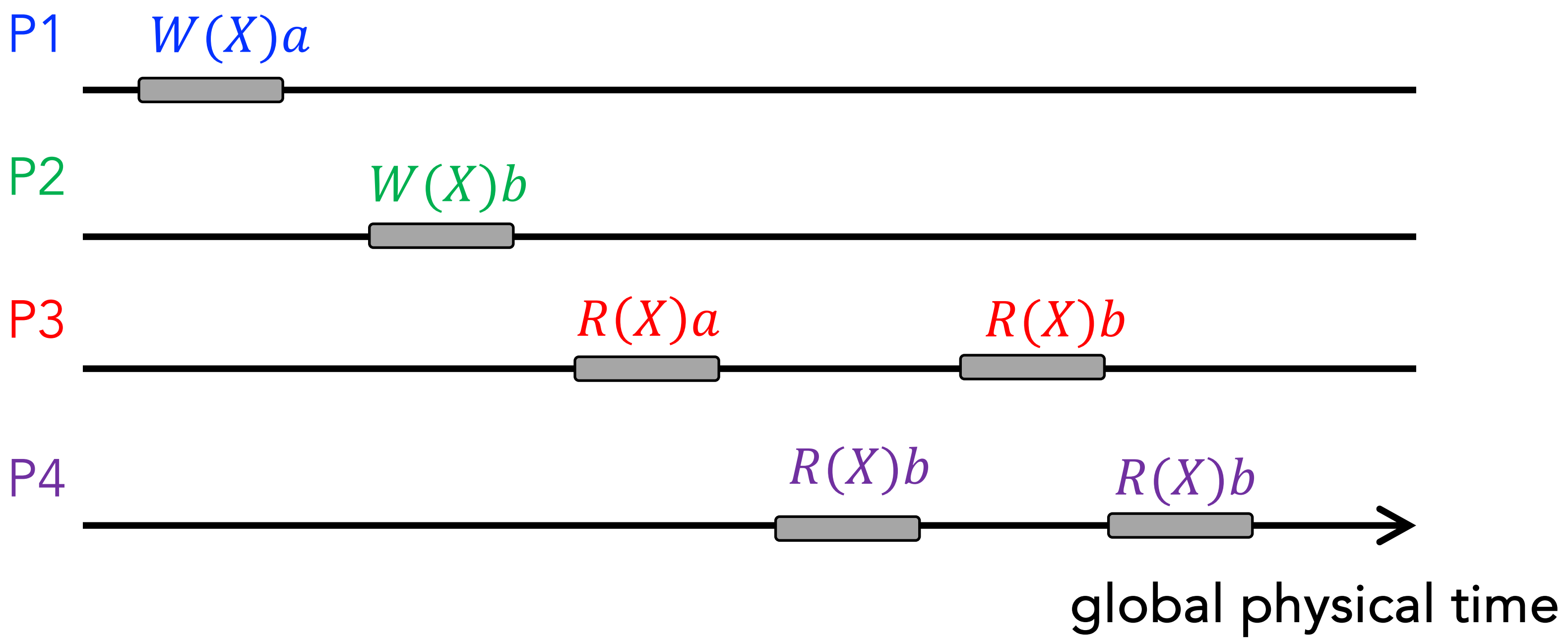


Can these reads be returned by a sequentially consistent system?

Yes, these reads could be returned by a sequentially consistent system

$W(X)a, W(X)b, R(X)b, \ldots$

Sequential Consistency: (Counter) Examples

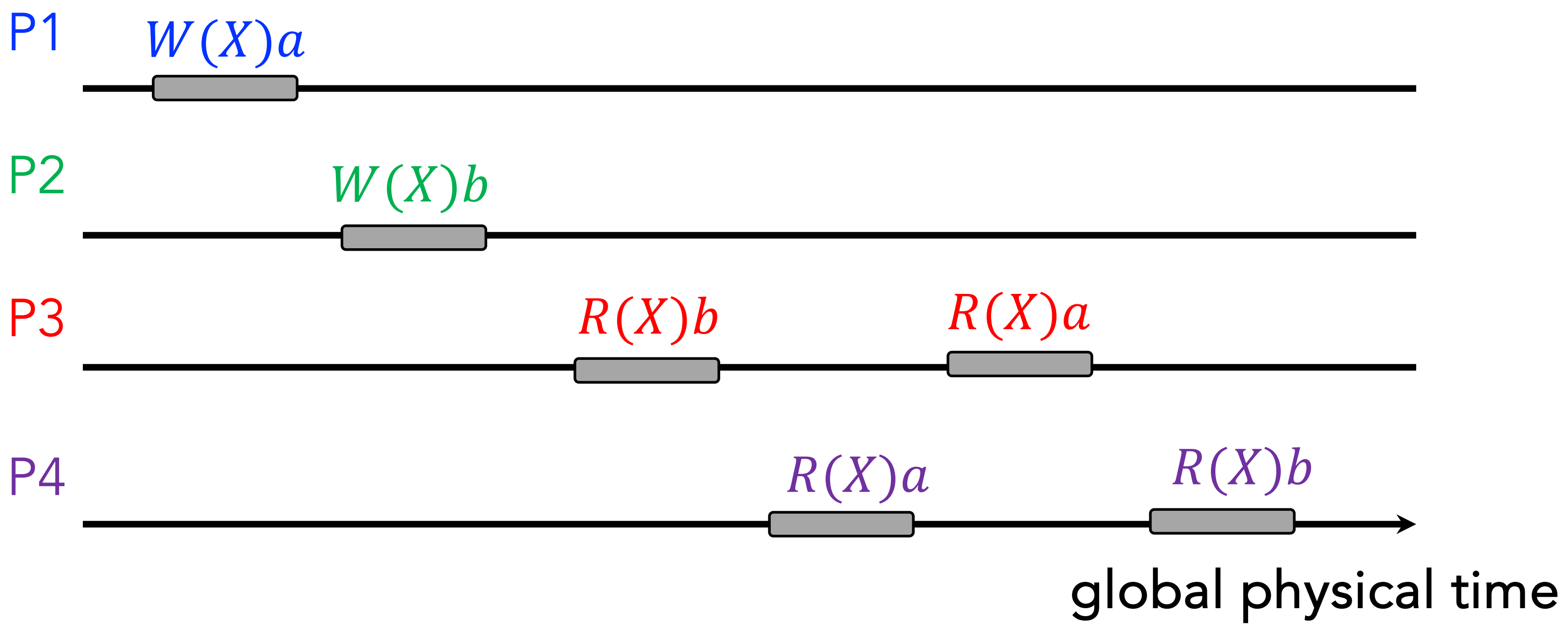


Can these reads be returned by a sequentially consistent system?

What global total order can explain these results?

$W(X)a, R(X)a, W(X)b, R(X)b, \ldots$

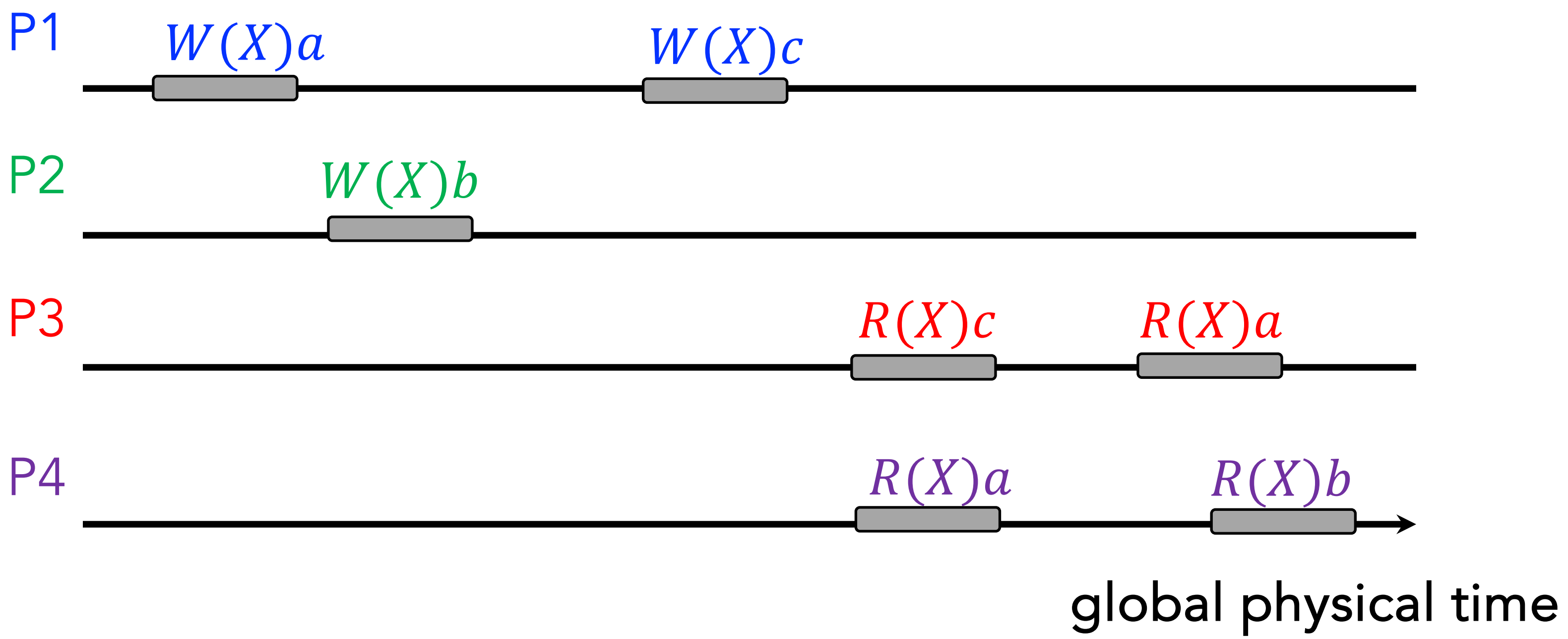
Sequential Consistency: (Counter) Examples



Can these reads be returned by a sequentially consistent system?

No global total ordering can explain these results…

Sequential Consistency: (Counter) Examples



Can these reads be returned by a sequentially consistent system?

No global total ordering can explain these results...
E.g., the following global ordering doesn't preserve P1's ordering $W(X)c, R(X)c, W(X)a, R(X)a, W(X)b$, ...

Causal Consistency [Hutto and Ahamad, 1990]

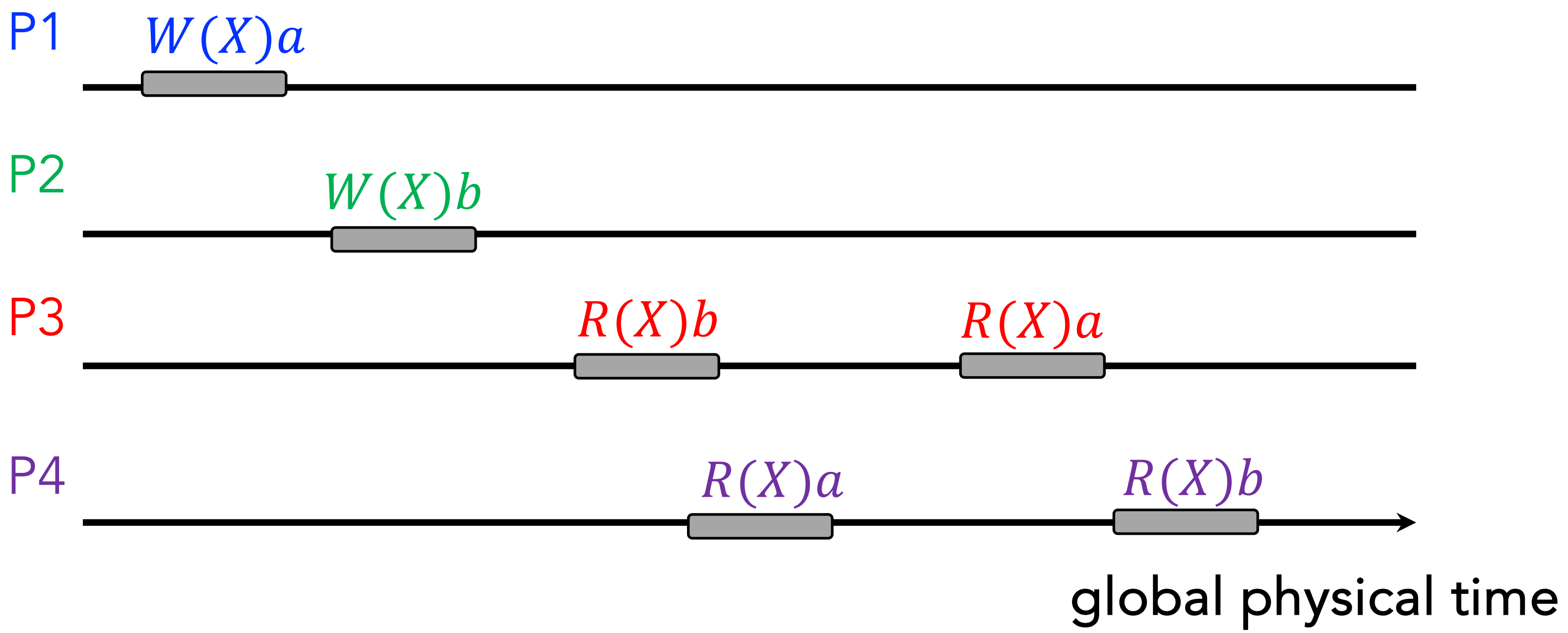
- Recall causality notion from Lamport clocks
  - Remember happens-before implies potential causality
  - That's what causal consistency enforces
- Causal consistency: potentially causally related operations must be seen by all processes in the same order
  - In other words, if $a \rightarrow b$, then a must execute before b on all replicas
  - All concurrent ops may be seen in different orders
- Key differences from sequential consistency
  - Does not require a total order

Causal Consistency: Implications

- Reads are fresh only w.r.t. the writes that they are (potentially) causally dependent on

- Only (potentially) causally-related writes are ordered by all replicas in the same way
  - But concurrent writes may be committed in different orders

Causal Consistency: (Counter) Examples

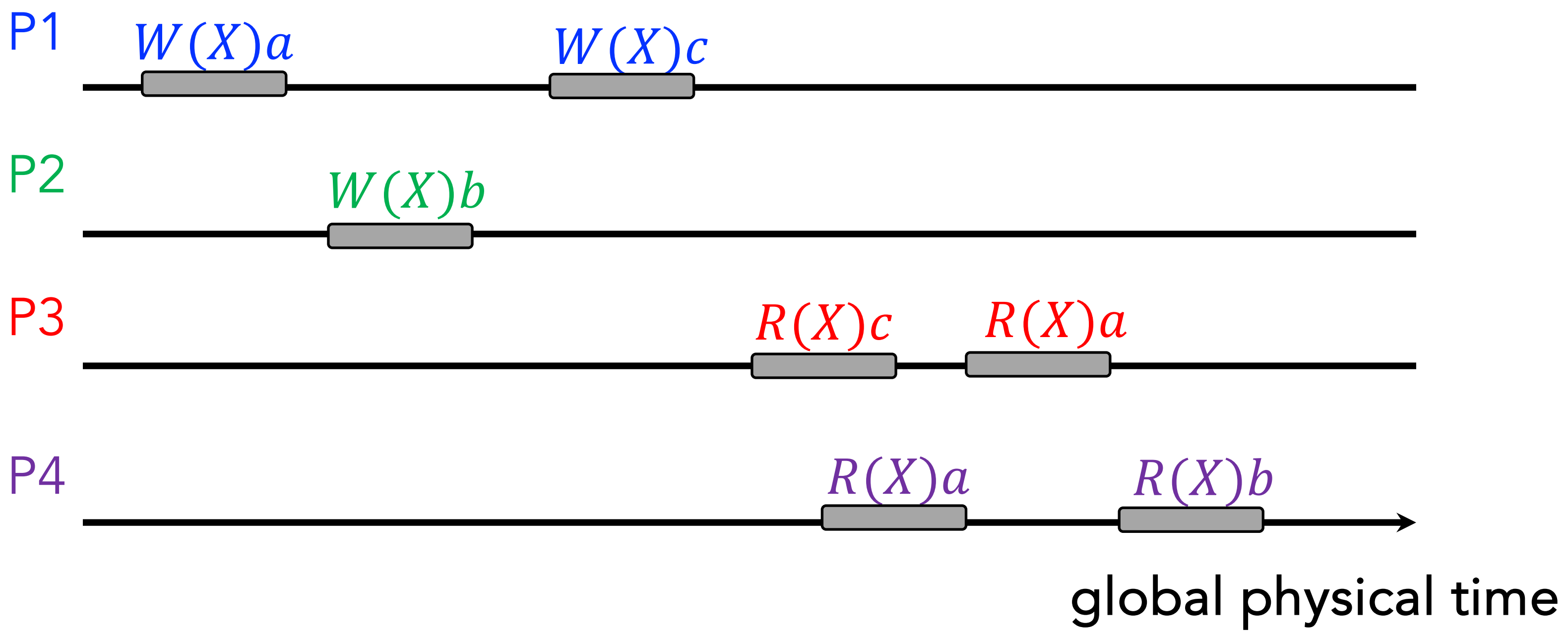


Can these reads be returned by a causally consistent system?

Can be returned by a causally consistent system

W(X)a || W(X)b, hence they can be seen in different orders by different processes

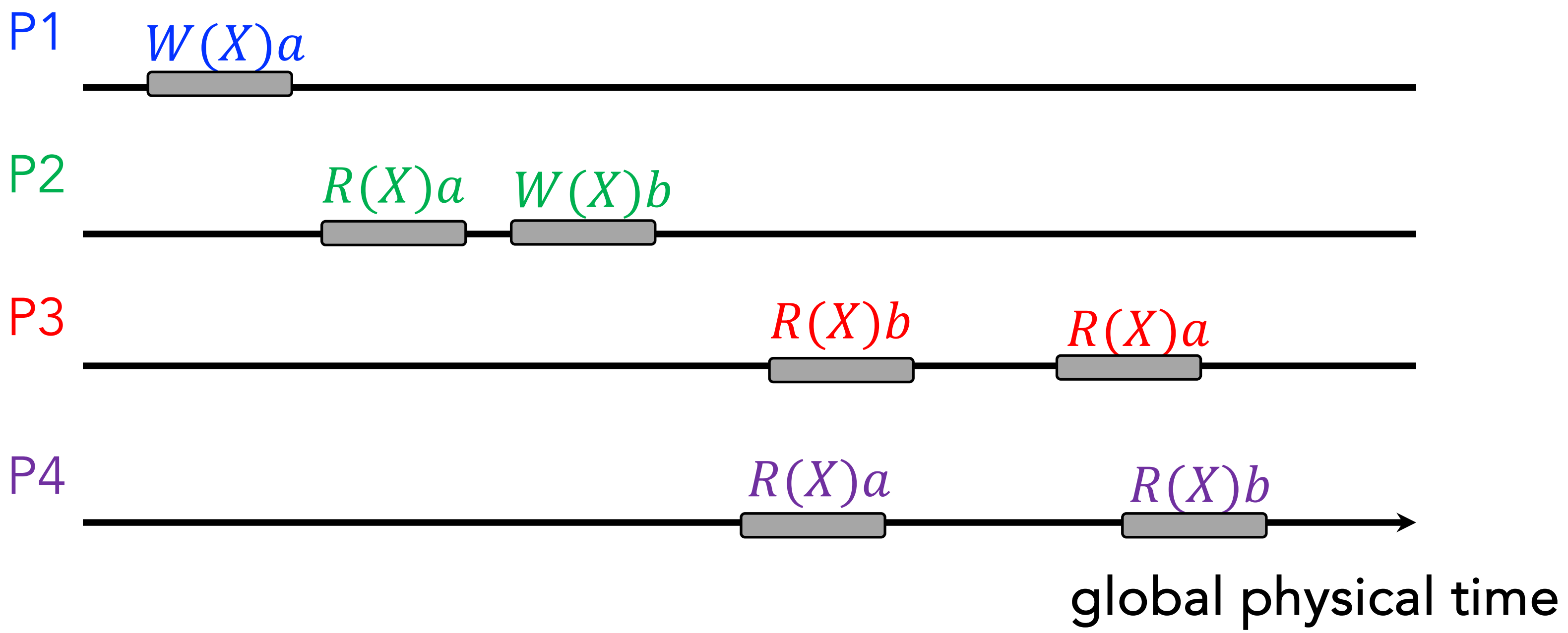
Causal Consistency: (Counter) Examples



Can these reads be returned by a causally consistent system?

Having read c, $R(X)c$, P3 must continue to read c or some newer value (perhaps b), but can't go back to a, because $W(X)c$ was conditional upon $W(X)a$ having finished

Causal Consistency: (Counter) Examples



Can these reads be returned by a causally consistent system?

$W(X)b$ is (potentially) causally-related on $R(X)a$, which is (potentially) causally-related on $W(X)a$. Therefore, system must enforce $W(X)a < W(X)b$ ordering. But P3 violates that ordering, because it reads a after reading b.

Why Causal Consistency?

- Causal consistency is strictly weaker than sequential consistency although can give strange results, as you have seen
  - If system is sequentially consistent → it is also causally consistent
- BUT: it also offers more possibilities for concurrency
  - Concurrent operations (which are not causally-dependent) can be executed in different orders by different people
  - In contrast to sequential consistency, we do not need to enforce a global ordering of all operations
  - Hence, one can get better performance than sequential

Next Lecture

- Eventual Consistency
- Consensus in Distributed Systems