Consistency

COS 316: Principles of Computer System Design

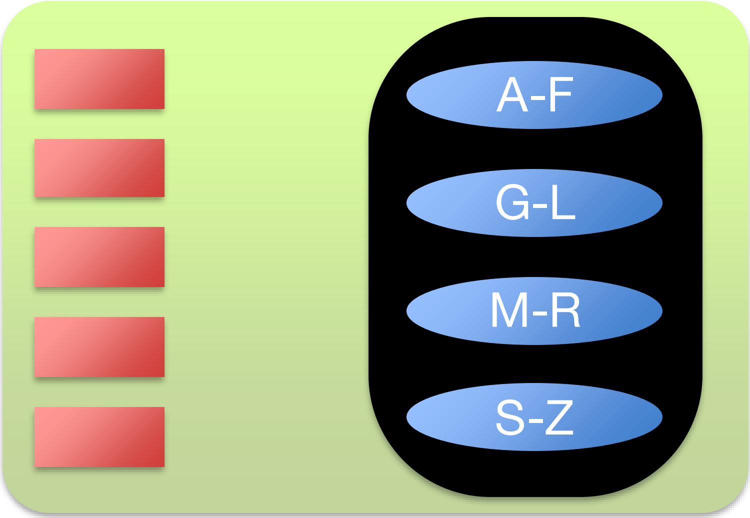
Amit Levy & Ravi Netravali

Why Do We Build Systems?

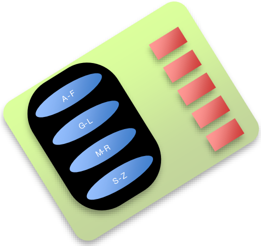
- …
- Abstract away complexity

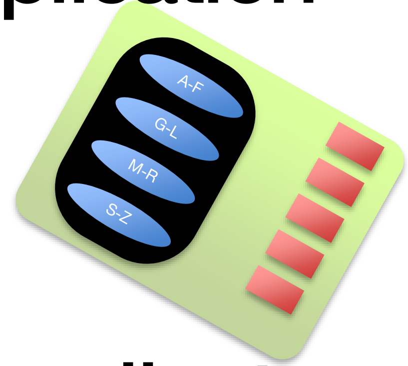
Distributed Systems are Highly Complex Internally

Sharding



(Geo)-Replication

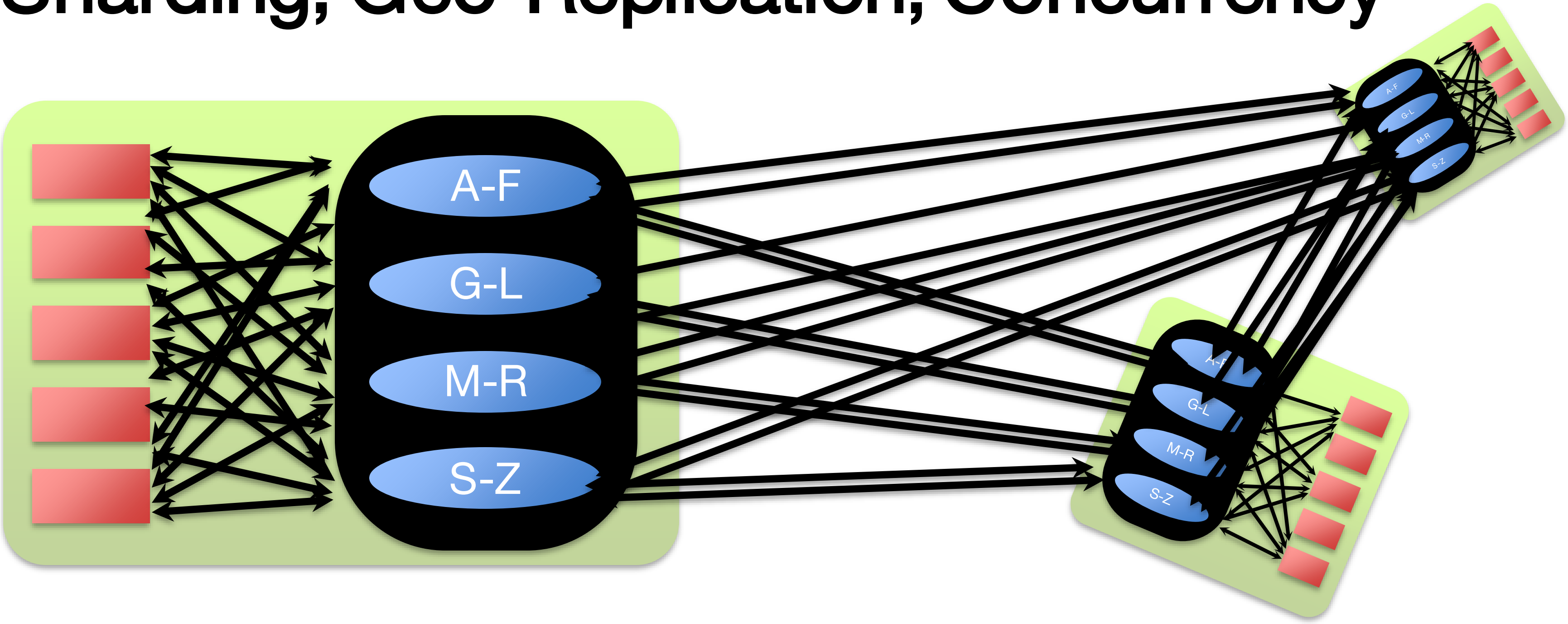




Concurrent access by many client

Distributed Systems are Highly Complex Internally

Sharding, Geo-Replication, Concurrency



Distributed Systems are Highly Complex Internally

Sharding, Geo-Replication, Concurrency

Consistency Models:

Control how much of this complexity is abstracted away

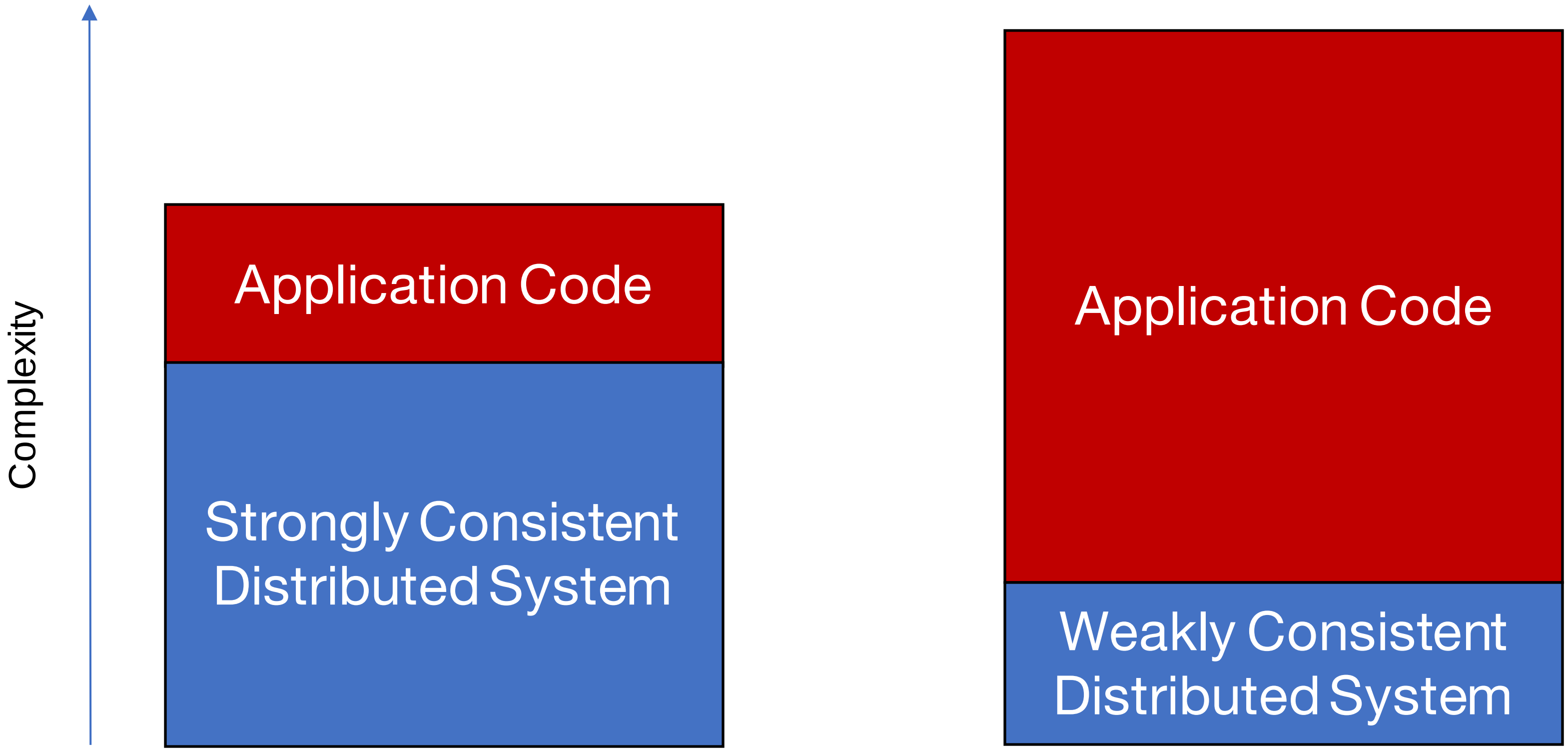
Consistency Models

- 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

- Not the interface, but defines semantics of the interface

Stronger vs Weaker Consistency



Stronger vs Weaker Consistency

- Stronger consistency models
    - \+ Easier to write applications
    - \- System must hide many behaviors
    - \- Might be slow

- Fundamental tradeoffs between consistency, availability, and performance
    - (Discuss CAP, PRAM, SNOW in 418!)

- Weaker consistency models
    - \- Harder to write applications
        - Cannot (reasonably) write some applications
    - \+ System needs to hide few behaviors
    - \+ Can be faster!

Consistency Hierarchy

Linearizability	Behaves like a single processor
↓	
Causal+ Consistency	Everyone sees related operations in the same order
↓	
Eventual Consistency	Anything goes

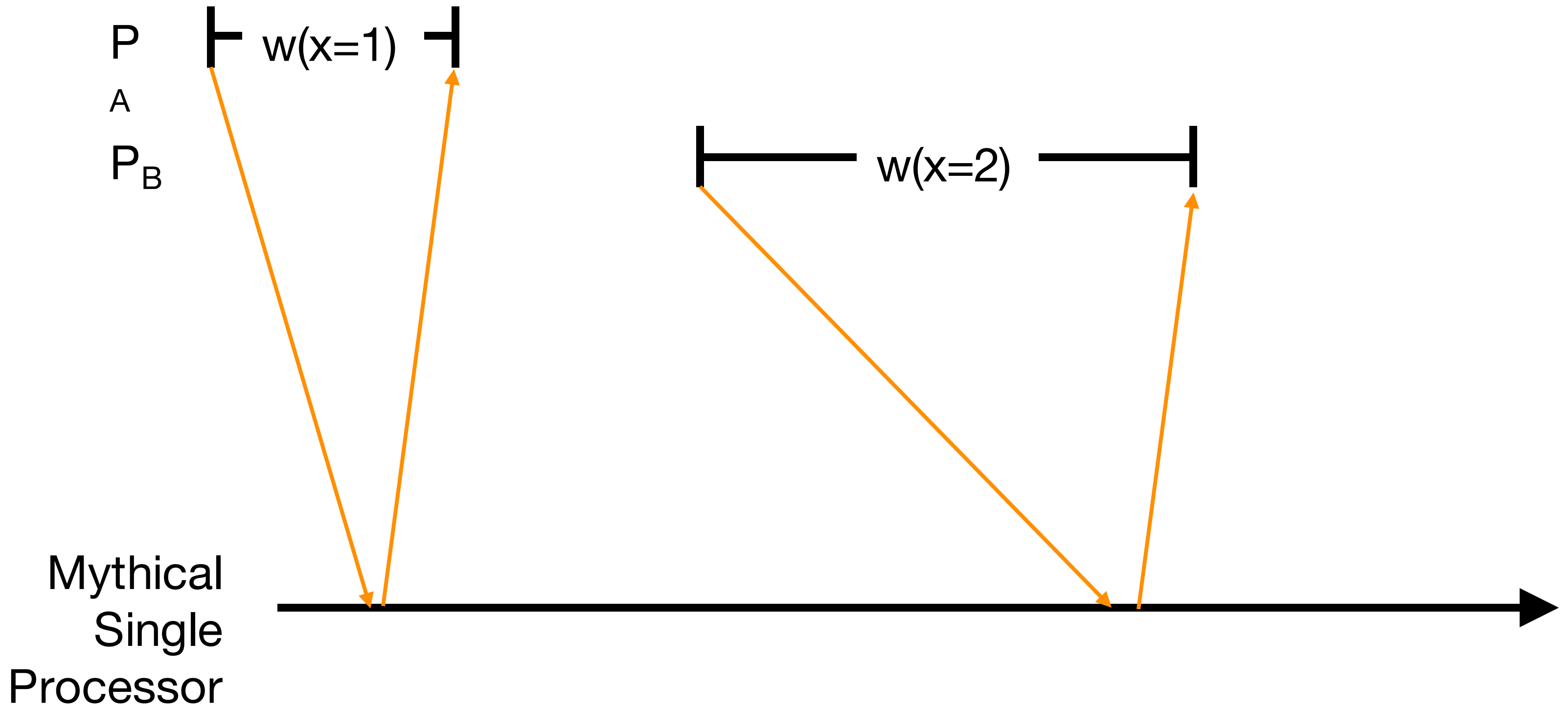
Linearizability == "Appears to be a Single Processor"

- External client submitting requests and getting responses from the system can't tell this is not a single processor!

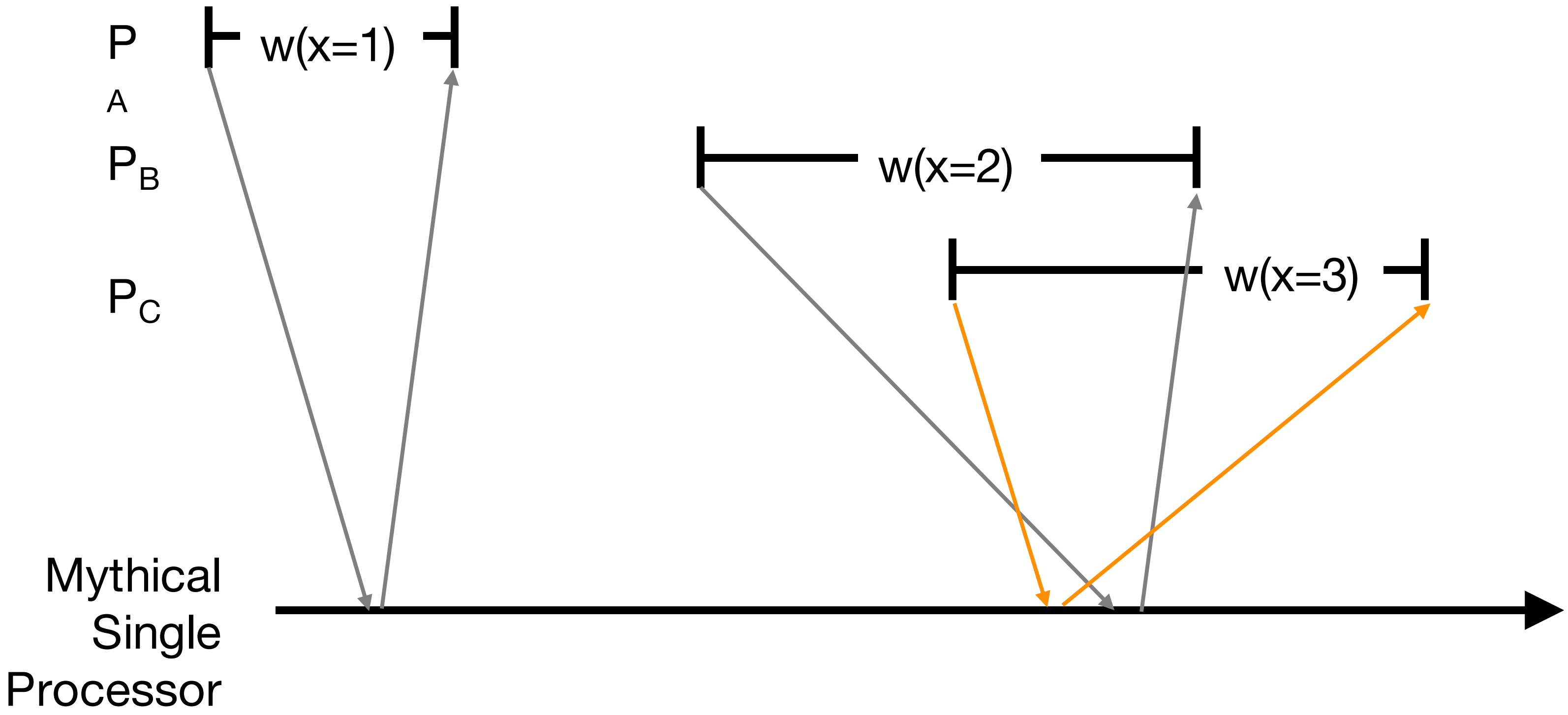
- Consistent with some total order over all operations
  - As though all requests processed one by one in some order
  - Such that…

- 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)

Real-Time Ordering Examples



Real-Time Ordering Examples



Linearizable?

P_A ⊢ w(x=1) ⊣

P_B ⊢ w(x=2) ⊣

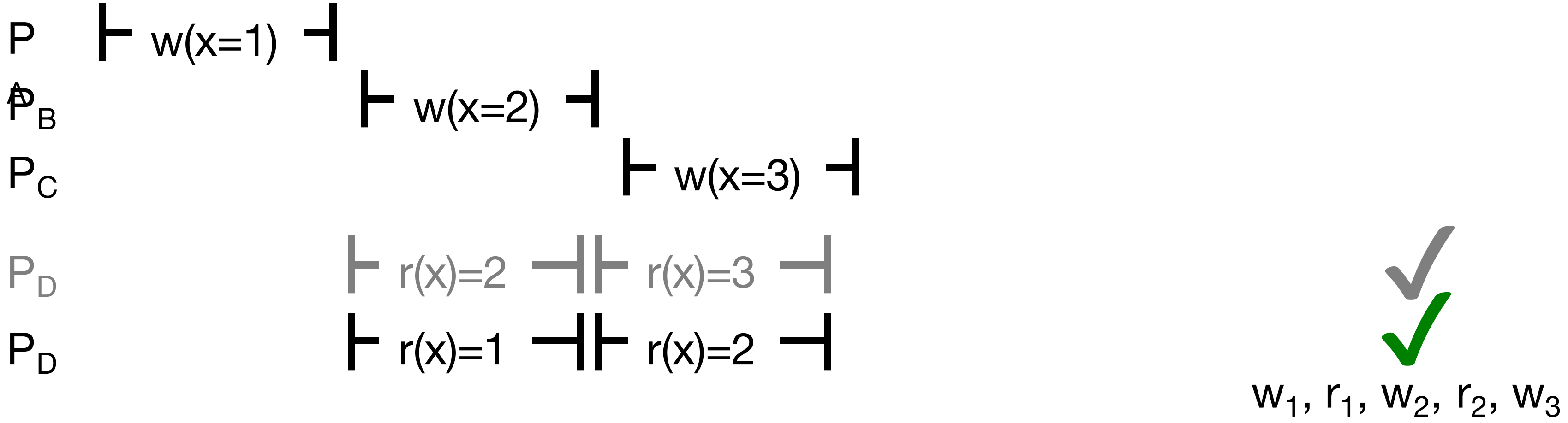
P_C ⊢ w(x=3) ⊣

P_D ⊢ r(x)=2 ⊣⊢ r(x)=3 ⊣

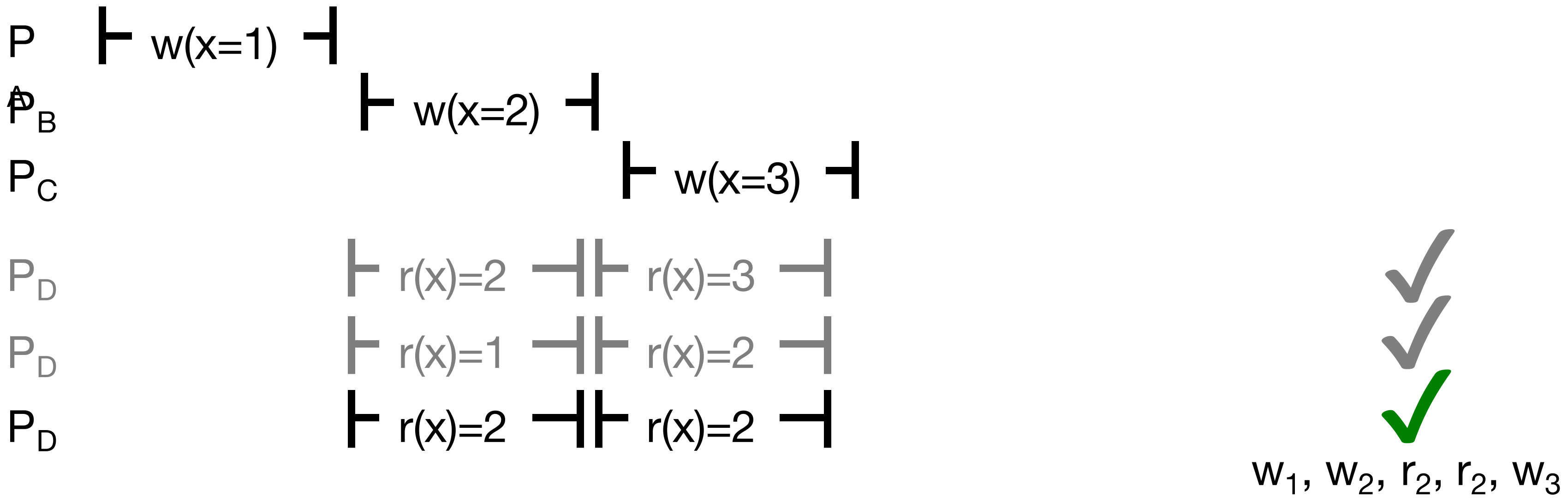
✓

w_1, w_2, r_2, w_3, r_3

Linearizable?



Linearizable?



Linearizable?

P_A ⊢ w(x=1) ⊣

P_B ⊢ w(x=2) ⊣

P_C ⊢ w(x=3) ⊣

P_D ⊢ r(x)=2 ⊣⊢ r(x)=3 ⊣ ✓

P_D ⊢ r(x)=1 ⊣⊢ r(x)=2 ⊣ ✓

P_D ⊢ r(x)=2 ⊣⊢ r(x)=2 ⊣ ✓

P_D ⊢ r(x)=1 ⊣⊢ r(x)=3 ⊣ ✓

w_1, r_1, w_2, w_3, r_3

Linearizable?

P_A ⊢ w(x=1) ⊣

P_B ⊢ w(x=2) ⊣

P_C ⊢ w(x=3) ⊣

P_D ⊢ r(x)=2 ⊣⊢ r(x)=3 ⊣ ✓

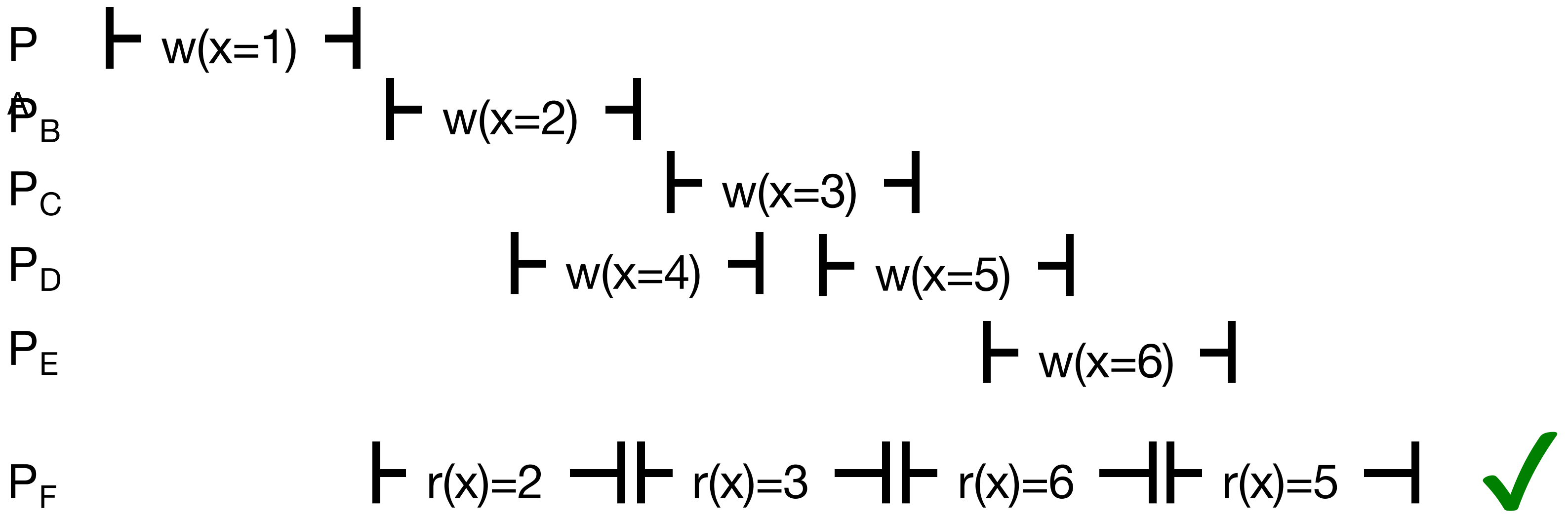
P_D ⊢ r(x)=1 ⊣⊢ r(x)=2 ⊣ ✓

P_D ⊢ r(x)=2 ⊣⊢ r(x)=2 ⊣ ✓

P_D ⊢ r(x)=1 ⊣⊢ r(x)=3 ⊣ ✓

P_D ⊢ r(x)=2 ⊣⊢ r(x)=1 ⊣ X

Linearizable?



$w_1, w_2, r_2, w_4, w_3, r_3, w_6, r_6, w_5, r_5$

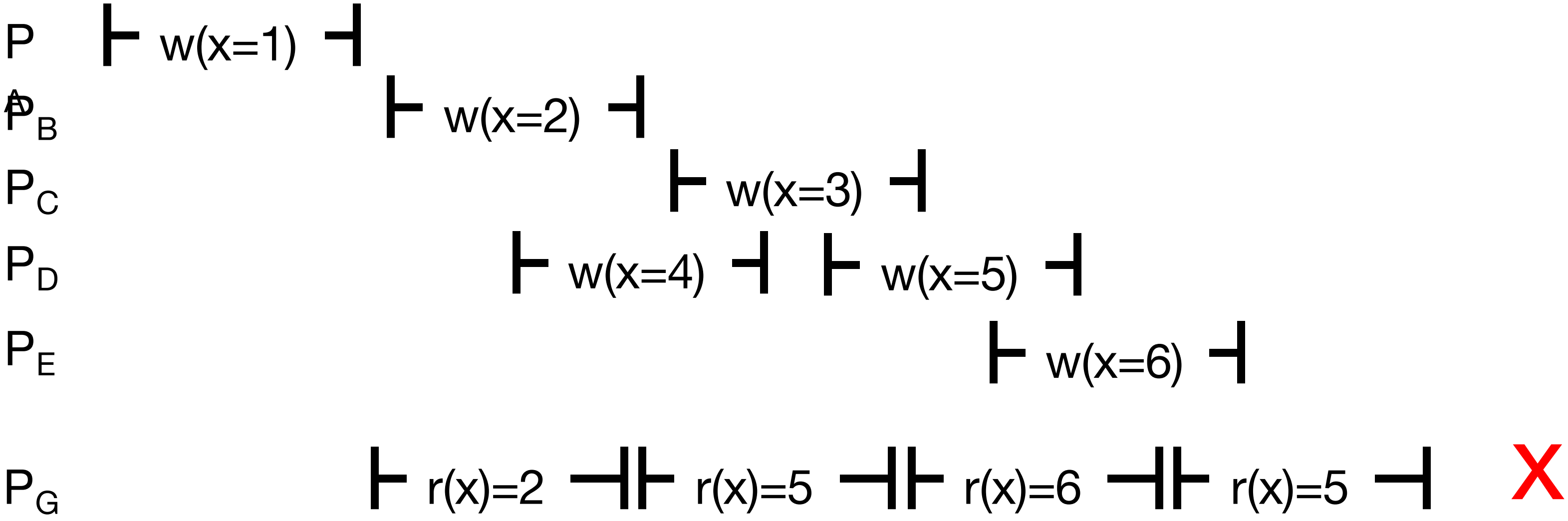
OR

$w_1, \mathbf{w_4}, w_2, r_2, w_3, r_3, w_6, r_6, w_5, r_5$

OR

$w_1, w_2, r_2, w_3, r_3, \mathbf{w_4}, w_6, r_6, w_5, r_5$

Linearizable?



Linearizable?

P_A ⊢ w(x=1) ⊣

P_B ⊢ w(x=2) ⊣

P_C ⊢ w(x=3) ⊣

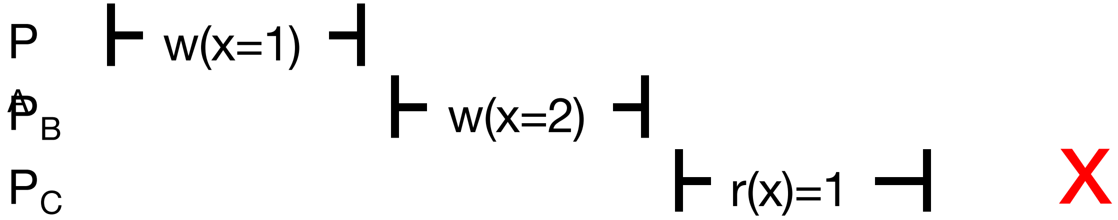
P_D ⊢ w(x=4) ⊣ ⊢ w(x=5) ⊣

P_E ⊢ w(x=6) ⊣

P_H ⊢ r(x)=4 ⊣⊢ r(x)=2 ⊣⊢ r(x)=3 ⊣⊢ r(x)=6 ⊣ ✓

$w_1, w_4, r_4, w_2, r_2, w_3, r_3, w_5, w_6, r_6$

Linearizable?



Linearizability == "Appears to be a Single Processor"

- External client submitting requests and getting responses from the system can't tell this is not a single processor!

- Consistent with some total order over all operations
  - As though all requests processed one by one in some order
  - Such that…

- 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)

How to Provide Linearizability?

1. Use a single processor ☺

1. Use “state-machine replication” on top of a consensus protocol like Paxos
   - Distributed system appears to be single processor that does not fail!!
   - Covered extensively in 418

2. …

Consistency Hierarchy

Linearizability	Behaves like a single processor
↓	
Causal+ Consistency	Everyone sees related operations in the same order
↓	
Eventual Consistency	Anything goes

Causal+ Consistency Informally

1. … must

2. … nt

- Potential causality: event *a* could have a causal effect on event *b.*
  - Think: is there a path of information from *a* to *b*?
    - *a* and *b* done by the same entity (e.g., me)
    - *a* is a write and *b* is a read of that write
    - + transitivity

Causal+ Sufficient

↓ Then ↓

Photo Added

↓ Then ↓

Purchase retained

↓ Then ↓

Deletion retained

Causal+ Not Sufficient
(Need Linearizability)

- Need a total order of operations
    - e.g., Alice's bank account ≥ 0
- Need a real-time ordering of operations
    - e.g., Alice changes her password, Eve cannot login with old password

Consistency Hierarchy

Linearizability	Behaves like a single processor
↓	
Causal+ Consistency	Everyone sees related operations in the same order
↓	
Eventual Consistency	Anything goes

Eventual Consistency

- Anything goes for now…
  - (If updates stop,
    eventually all copies of the data are the same)
- But, eventually consistent systems often try to provide consistency and often do
  - e.g., Facebook's TAO system provided linearizable results 99.9994% of the time [Lu et al. SOSP '15]
- "Good enough" sometimes
  - e.g., 99 vs 100 likes

Consistency Model Summary

- Consistency model specifies strength of abstraction
  - Linearizability □ Causal+ □ Eventual
  - Stronger hides more, but has worse performance
- When building an application, what do you need?
  - Select system(s) with necessary consistency
  - Always safe to pick stronger
- When building a system, what are your guarantees?
  - Must design system such that they always hold
  - Must confront fundamental tradeoffs with performance
    - What is more important?