Wide Area Storage Systems - A Quick Overview

Ashvin Goel

Electrical and Computer Engineering
University of Toronto

ECE1724

Storage Systems

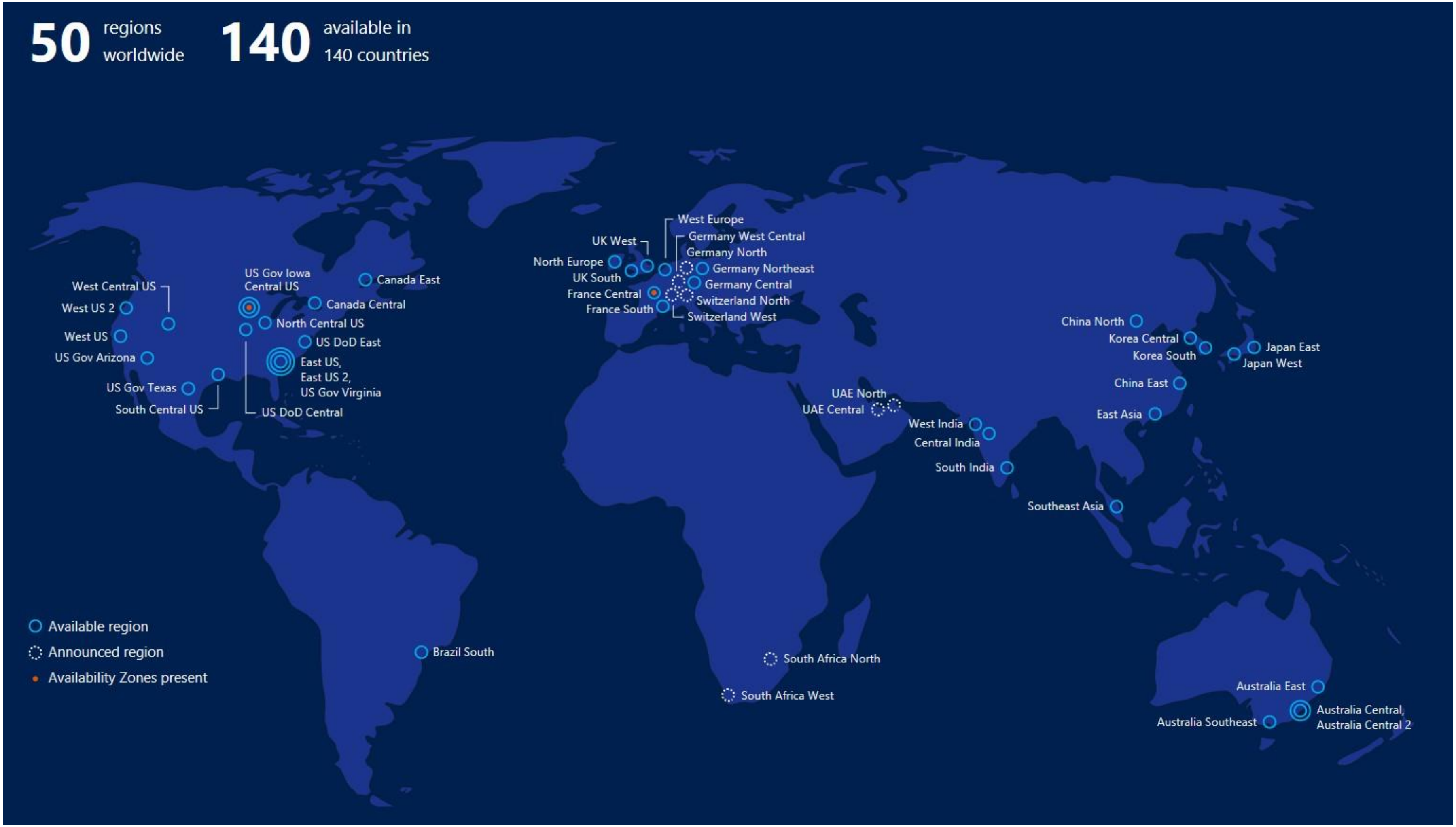
- We have seen GFS and Bigtable
  - GFS is a cluster-scale, file system
  - Bigtable is a cluster-scale, multi-dimensional key-value store
  - Both provide scalability and high availability for cluster-scale applications
- We have seen Sinfonia
  - Provides strong consistency guarantees at cluster scale
- However, modern web-scale applications are used by millions of geographically distributed users

Problem

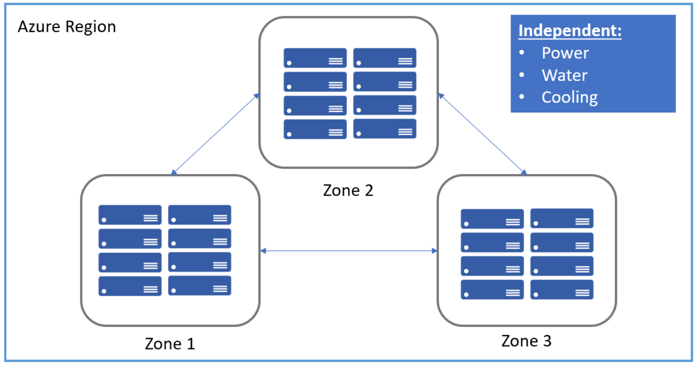
- One data center can’t solve it all
- Servicing data centers requires turning them off
  - Power systems, cooling systems, backbone routers, data center management systems
- Diurnal load patterns
  - Too much load during the day, too little during the night
- Geographically separated users
  - Too much latency for cross-continent operations
  - Cross-continent links are expensive

Solution

- Within a region: 3-5 data centers located within 10-100 miles apart
    - Improves availability – a data center can be turned off
- Across regions: build data centers based on user demand
    - Helps with diurnal load patterns
    - Reduces latency
    - Improves availability, disaster recovery



Microsoft Azure



Azure Availability Zones are separate buildings 10’s of miles apart within a Region

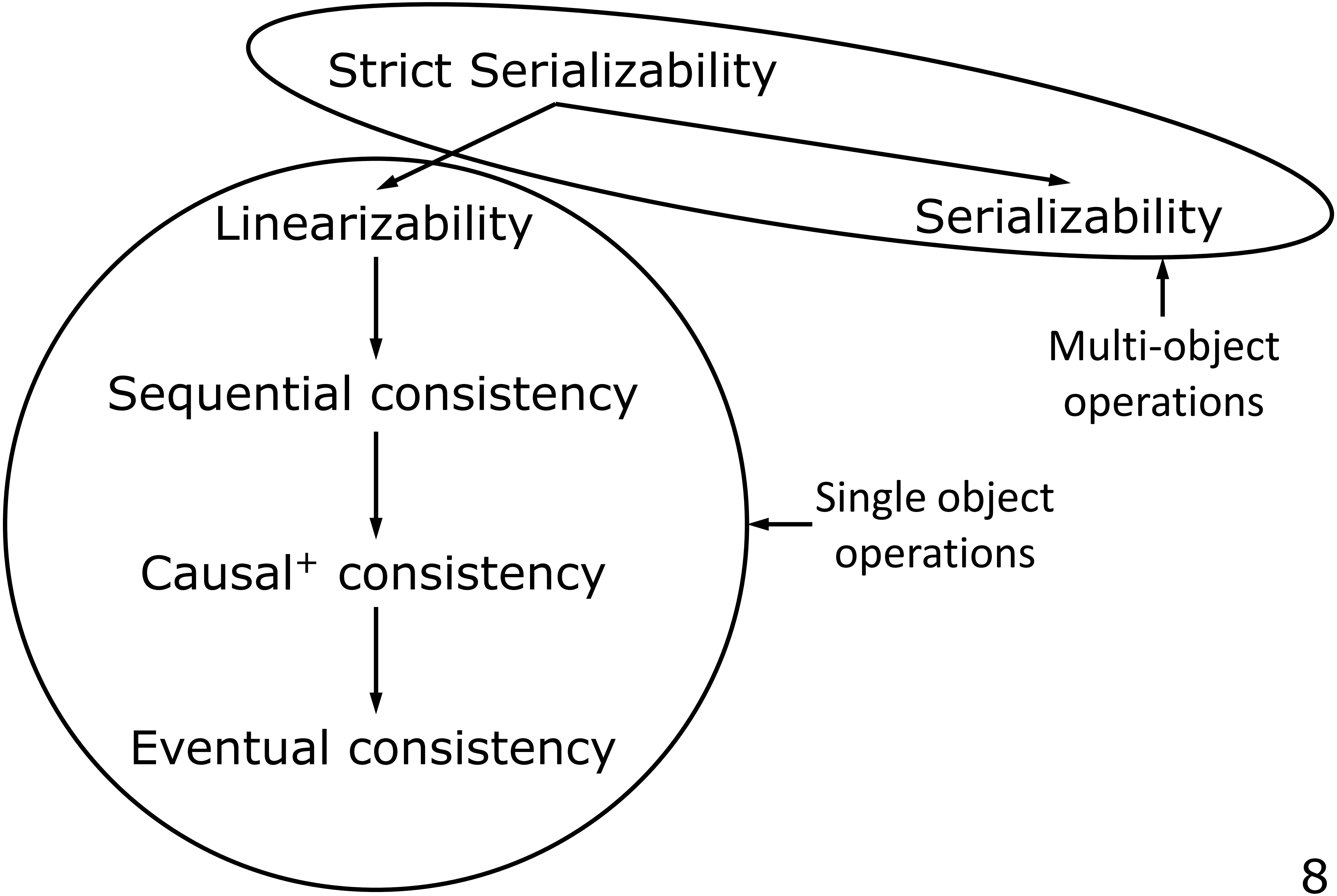
Consistency
(Once Again)

Single or Multi-Object Operations

- Single object operations
  - Think key-value stores
  - get(key), put(key, value) operations
  - Each operation accesses one shard (partition)

- Multi-object operations
  - Think transactions and databases
  - Each transaction accesses multiple rows atomically
  - Each transaction may access one or more shards (partitions)

Consistency Hierarchy



Strict Serializability

Linearizability

Serializability

Sequential consistency

Causal$^+$ consistency

Eventual consistency

Multi-object operations

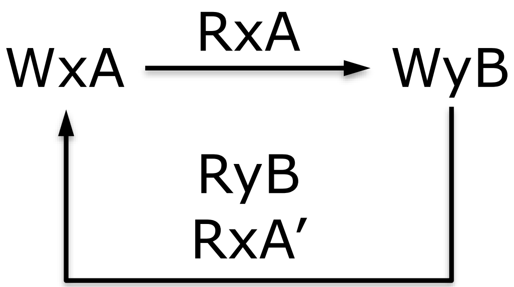
Single object operations

Consistency for Single-Object Operations (Partition Tolerant)

- Eventual consistency
  - All processes execute operations in any order
  - Assuming no new updates to a data item, all accesses to that item will eventually return the last updated value
    - Used in optimistically replicated systems
    - Weakest “reasonable” form of consistency for replicated data
  - Why partition tolerant?
- Causal$^+$ consistency
  - Causal: all processes execute operations in an order that satisfies causality (happens-before relation)
    - i.e., there are no cyclic dependencies among operations
  - Causal$^+$: data is eventually consistent also

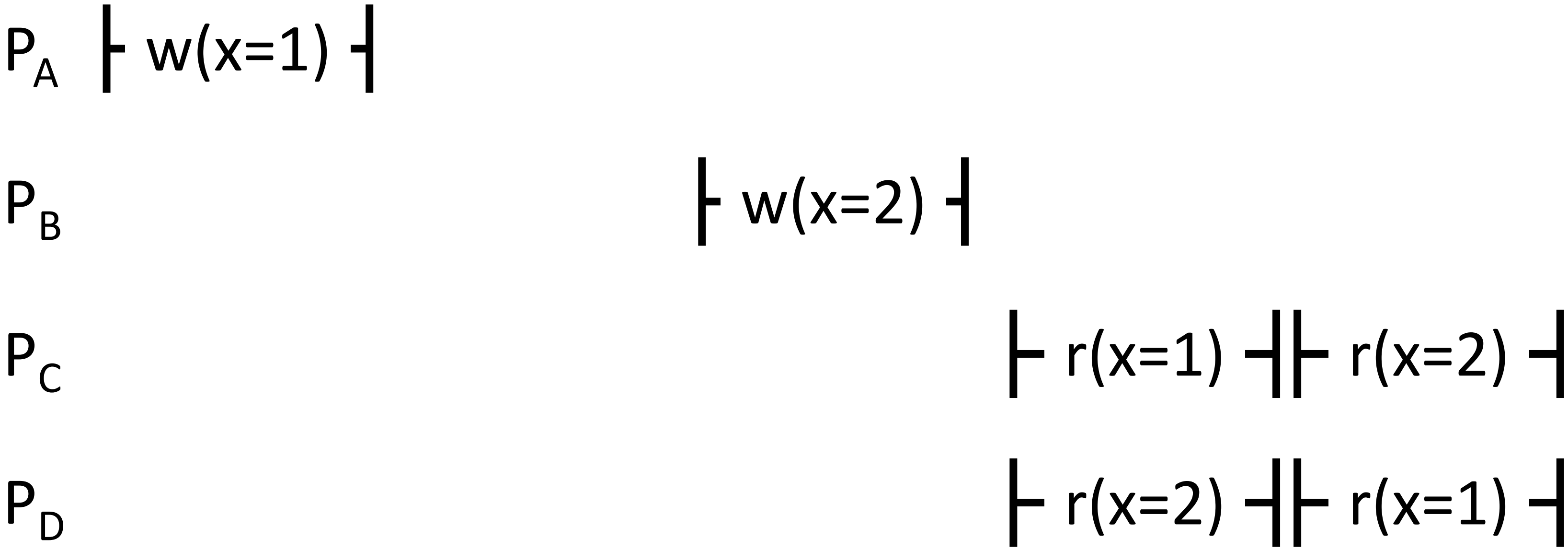
Understanding Causal Consistency

- Causal: All processes execute operations in an order that satisfies causality (happens-before relation)
  - Say Client 1 writes WxA,
    Client 2 reads RxA and then writes WyB,
    then WxA -> WyB (happens-before)
  - If a process reads RyB,
    then it cannot read RxA’, if WxA’ -> WxA
  - Otherwise, there would be a cyclic dependency between WxA and WyB
  - If writes are independent,
    they can be applied/visible
    in any order



Understanding Causal Consistency

- Are these operations causally consistent?



P_A w(x=1)

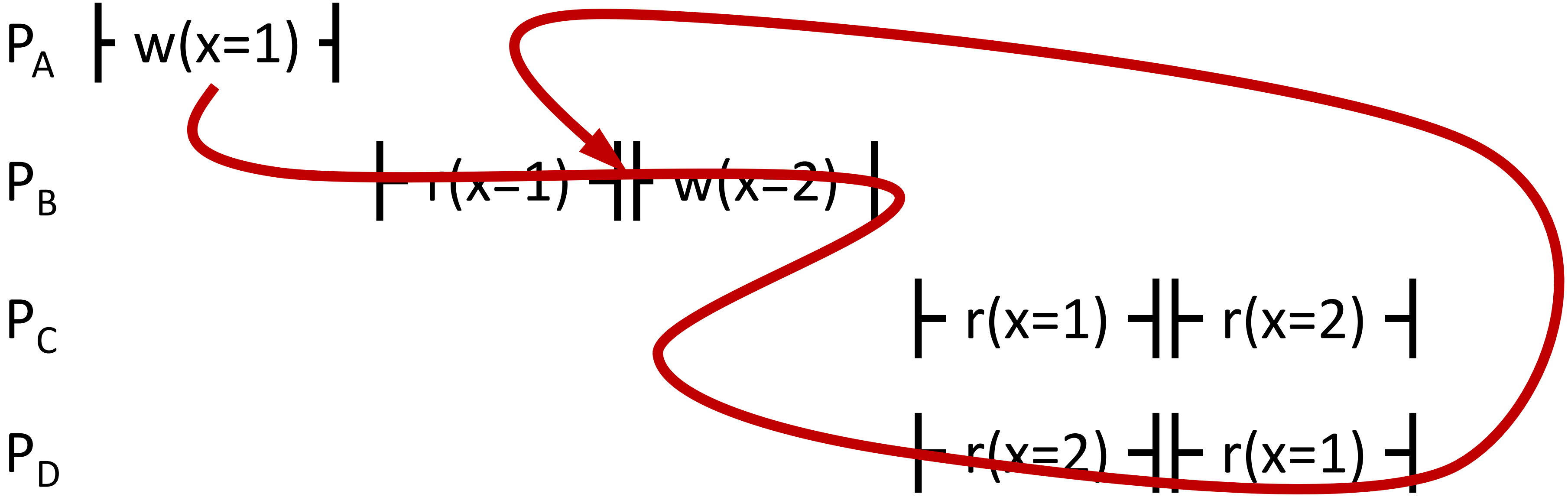
P_B w(x=2)

P_C r(x=1) r(x=2)

P_D r(x=2) r(x=1)

Understanding Causal Consistency

- Are these operations causally consistent?



P_A ├ w(x=1) ┤

P_B ├ r(x=1) ┤├ w(x=2) ┤

P_C ├ r(x=1) ┤├ r(x=2) ┤

P_D ├ r(x=2) ┤├ r(x=1) ┤

Consistency for Single-Object Operations (not Partition Tolerant)

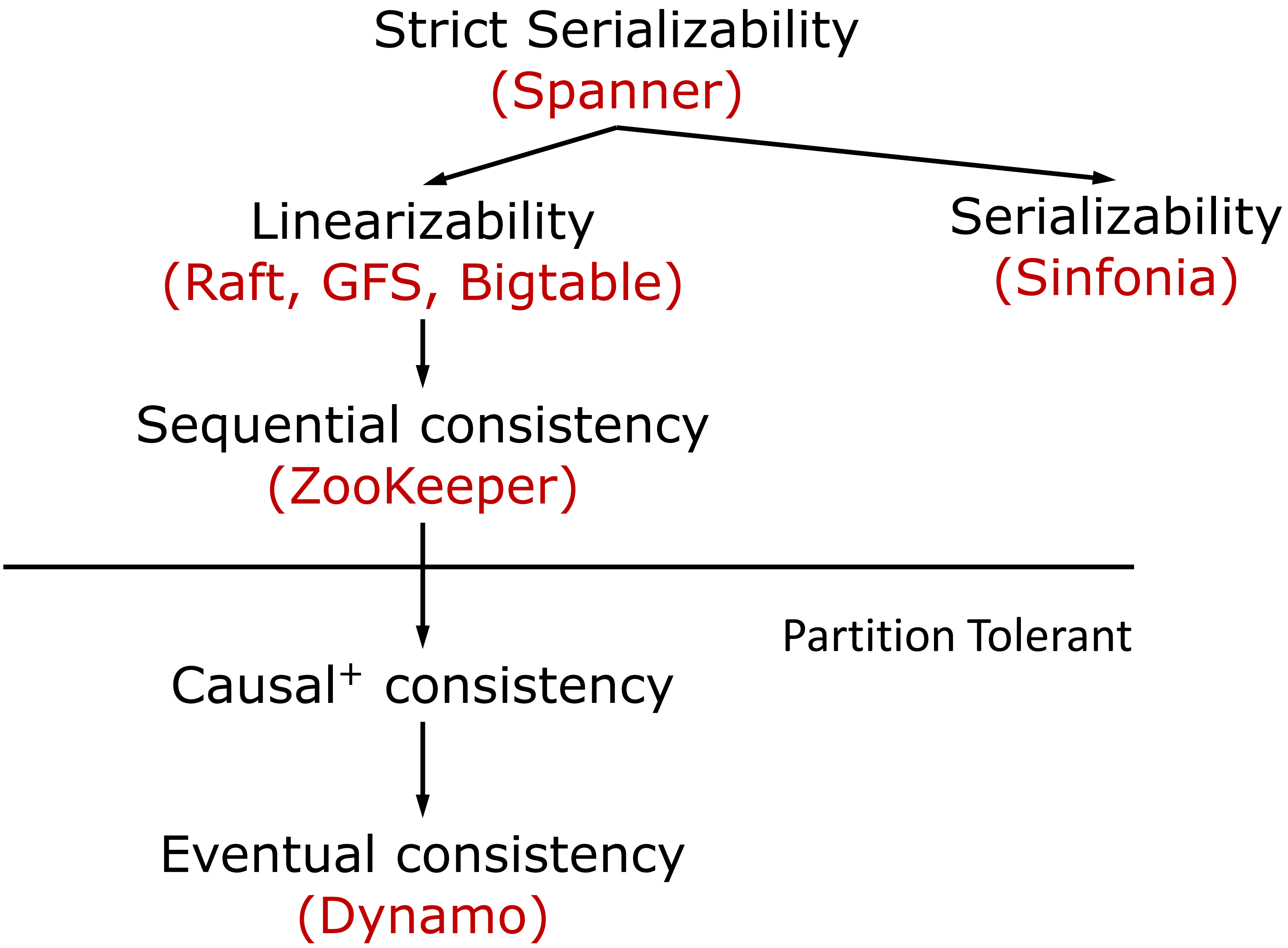
- Sequential consistency
  - All processes execute operations in some total order
    - Operations act as if they occurred (instantaneously), consistent with program order
    - Writes are totally ordered, even when not causally related, hence not partition tolerant

- Linearizability
  - All processes execute operations in some total order, while preserving real-time ordering
    - Operations act as if they occurred (instantaneously), consistent with program order, at some point in between invocation & response

Consistency for Multi-Object Operations

- Serializability
  - The outcome of executing transactions (e.g., resulting state) is equivalent to the outcome of its transactions executed sequentially without interleaving
- Strict serializability
  - Informally: Serializability + Linearizability
  - If Txn A completes before Txn B begins in real time, then A is ordered before B

Consistency Hierarchy



Strict Serializability
(Spanner)

Linearizability
(Raft, GFS, Bigtable)

Serializability
(Sinfonia)

Sequential consistency
(ZooKeeper)

Partition Tolerant

Causal$^+$ consistency

Eventual consistency
(Dynamo)

Questions to Keep in Mind

- We will be discussing Dynamo and Spanner
- How does each system provide the consistency property it says it does?
- What is the impact on availability, performance?
- What are the key differences in their design?