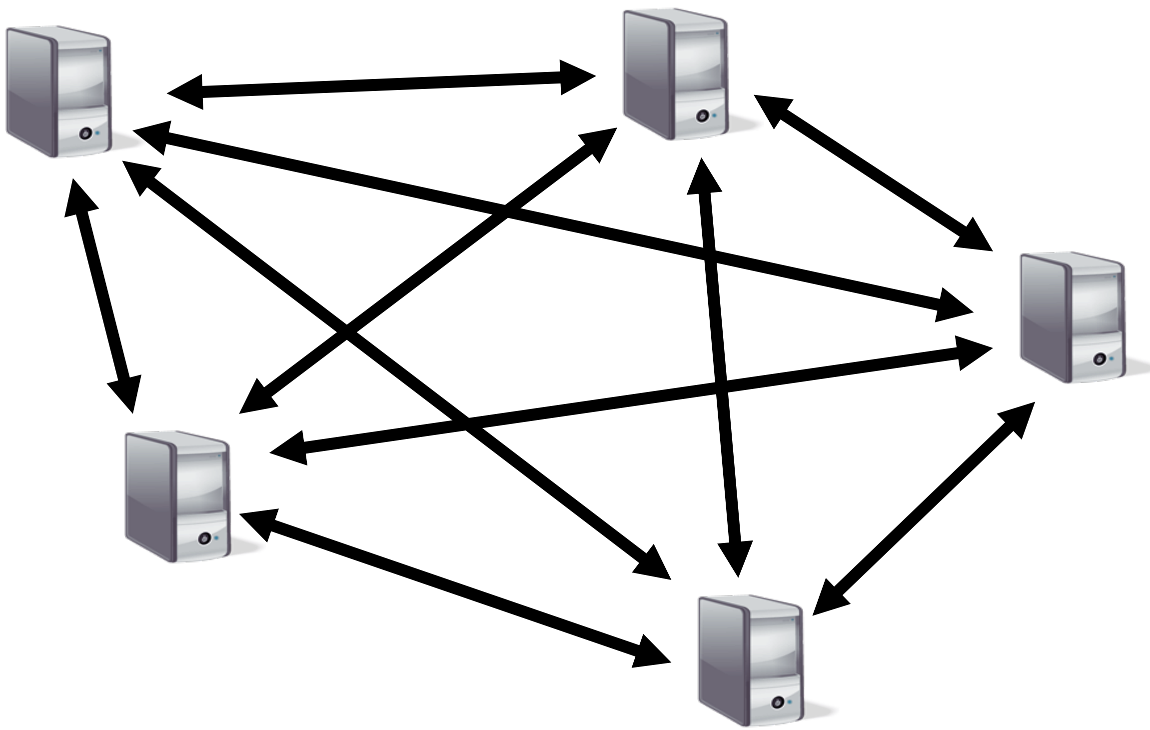
Impossibility Results:
CAP, PRAM, SNOW, PORT, & FLP

COS 418/518: Distributed Systems

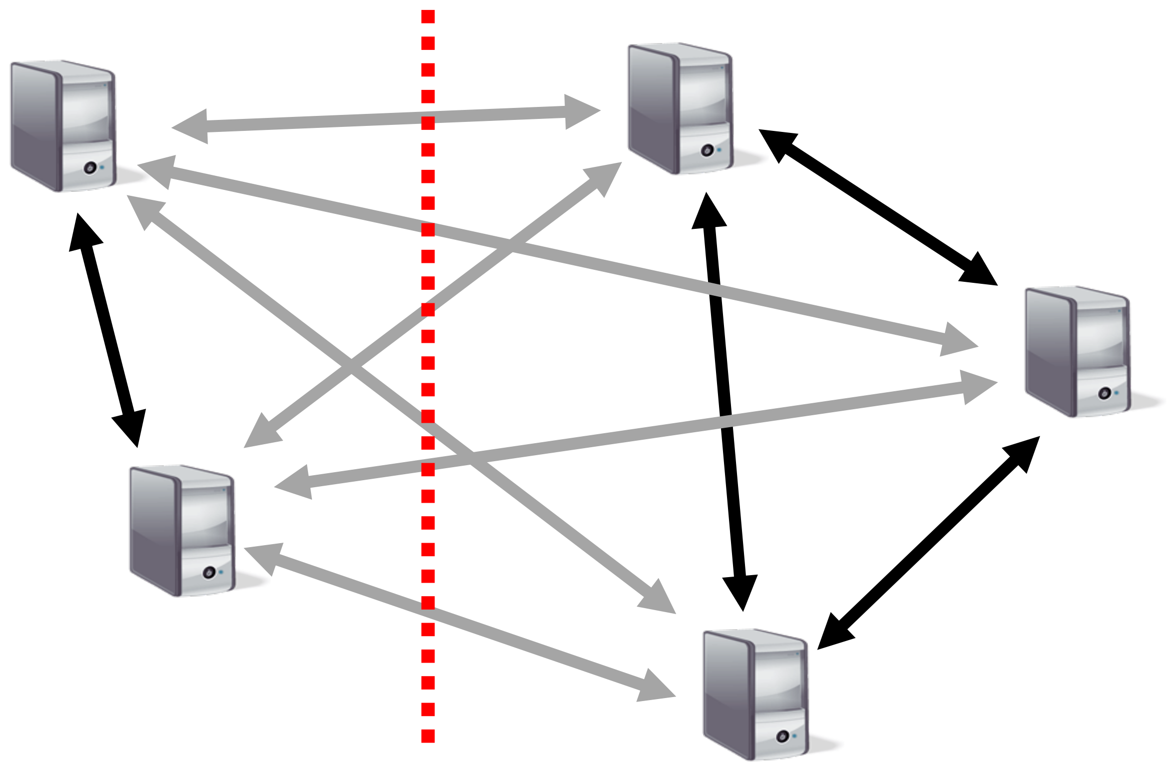
Lecture 19

Wyatt Lloyd

Network Partitions Divide Systems



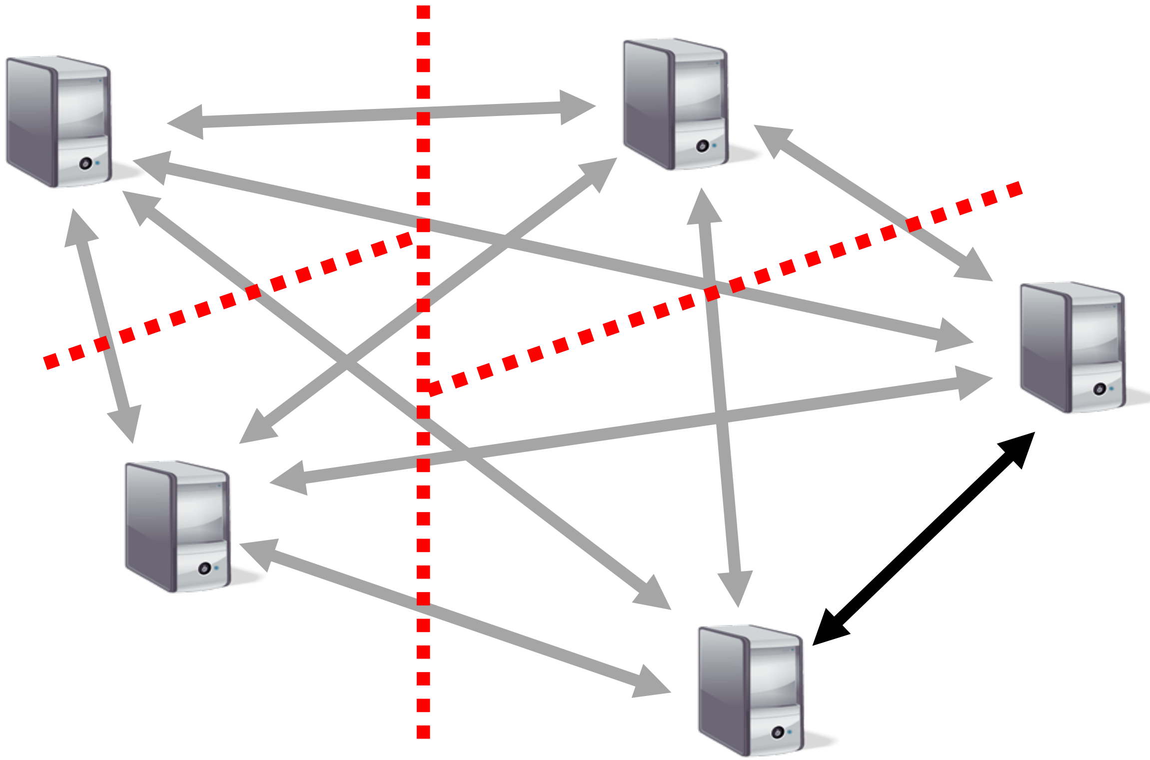
Network Partitions Divide Systems



How Can We Handle Partitions?

- Atomic Multicast?
- Bayou?
- Paxos?
- RAFT?
- COPS?
- Spanner?

How About This Set of Partitions?



Fundamental Tradeoff?

- 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

Linearizability (refresher)

- 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)

CAP Conjecture [Brewer 00]

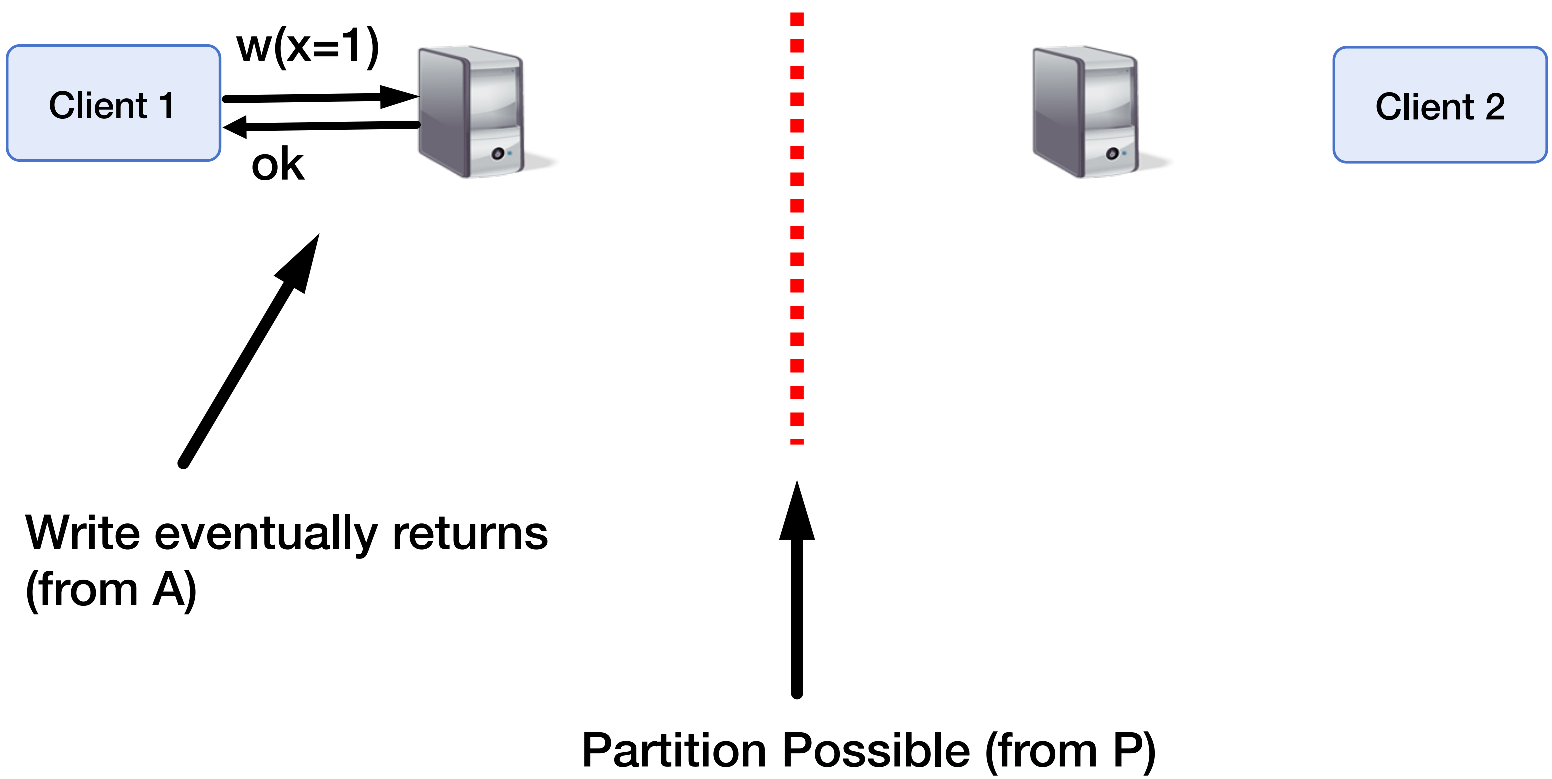
- From keynote lecture by Eric Brewer (2000)
  - History: Eric started Inktomi, early Internet search site based around “commodity” clusters of computers
  - Using CAP to justify “BASE” model: Basically Available, Soft-state services with Eventual consistency
- Popular interpretation: 2-out-of-3
  -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

CAP Theorem [Gilbert Lynch 02]

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

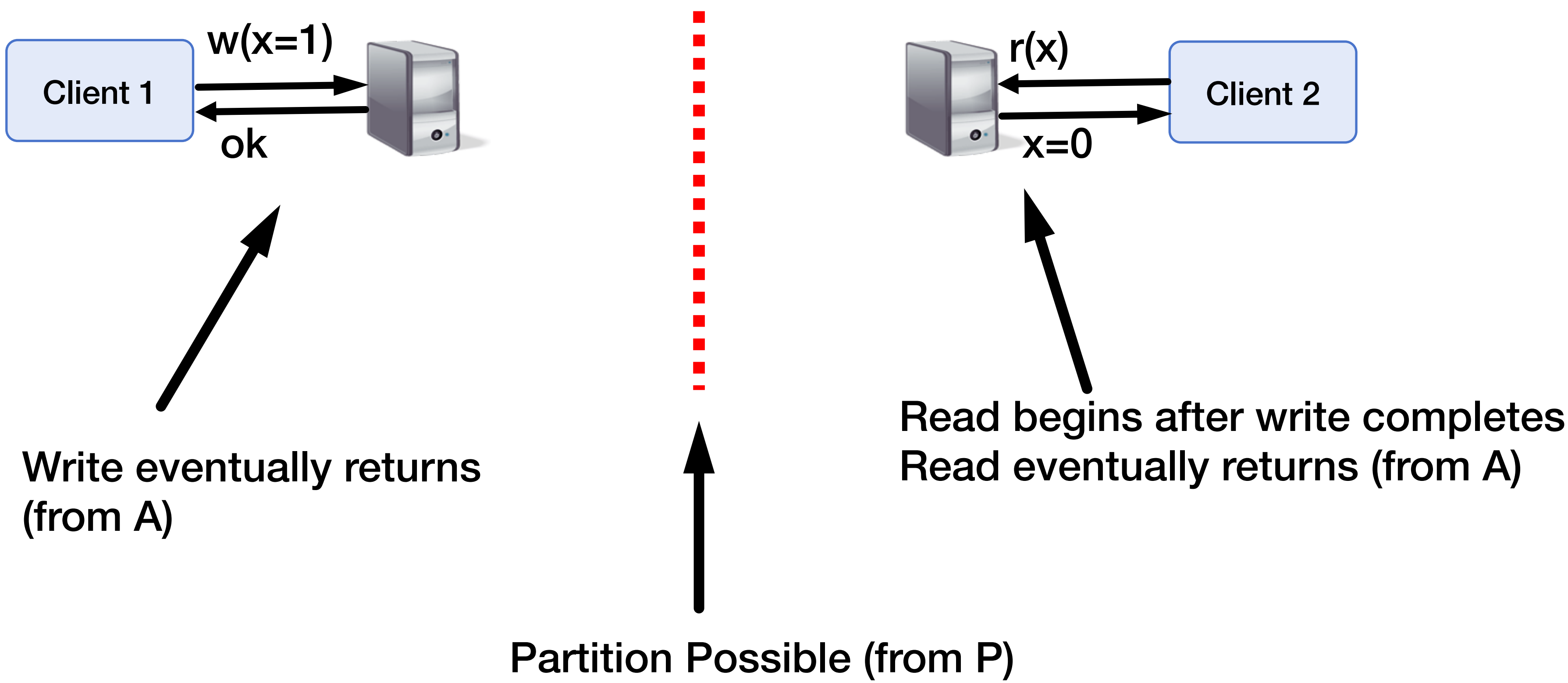
CAP Theorem [Gilbert Lynch 02]

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



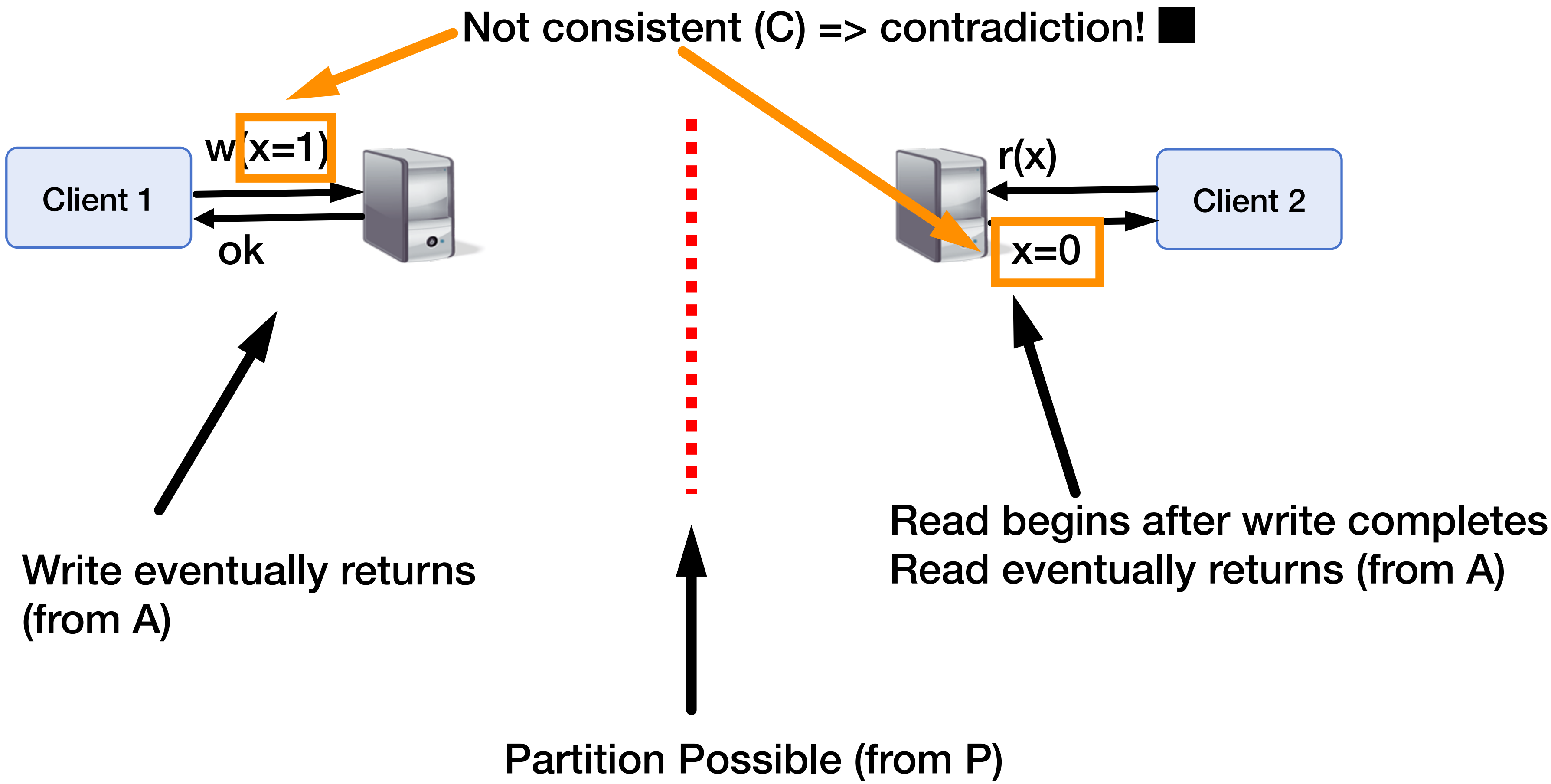
CAP Theorem [Gilbert Lynch 02]

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 Part 1

- Cannot “choose” no partitions
  - 2-out-of-3 interpretation doesn’t make sense
  - Instead, availability OR consistency?

- i.e., fundamental tradeoff between availability and consistency
  - When designing system must choose one or the other, both are not possible

CAP Interpretation Part 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)

Consistency Hierarchy

Strict Serializability	e.g., Spanner
↓	
Linearizability	e.g., RAFT
↓	CAP
Sequential Consistency	
↓	
Causal+ Consistency	e.g., Bayou
↓	
Eventual Consistency	e.g., Dynamo

Impossibility Results Useful!!!!

- Fundamental tradeoff in design space
  - **Must** make a choice

- Avoids wasting effort trying to achieve the impossible

- Tells us the best-possible systems we can build!

PRAM [Lipton Sandberg 88] [Attiya Welch 94]

- *d* is the worst-case delay in the network over all pairs of processes [datacenters]
- Sequentially consistent system
- read time + write time ≥ *d*
- Fundamental tradeoff between consistency and latency!
- (Skipping proof, see presenter notes or papers)

PRAM Theorem:

Impossible for sequentially consistent system to always provide low latency.

Consistency Hierarchy

Strict Serializability — e.g., Spanner

Linearizability — e.g., RAFT

Sequential Consistency

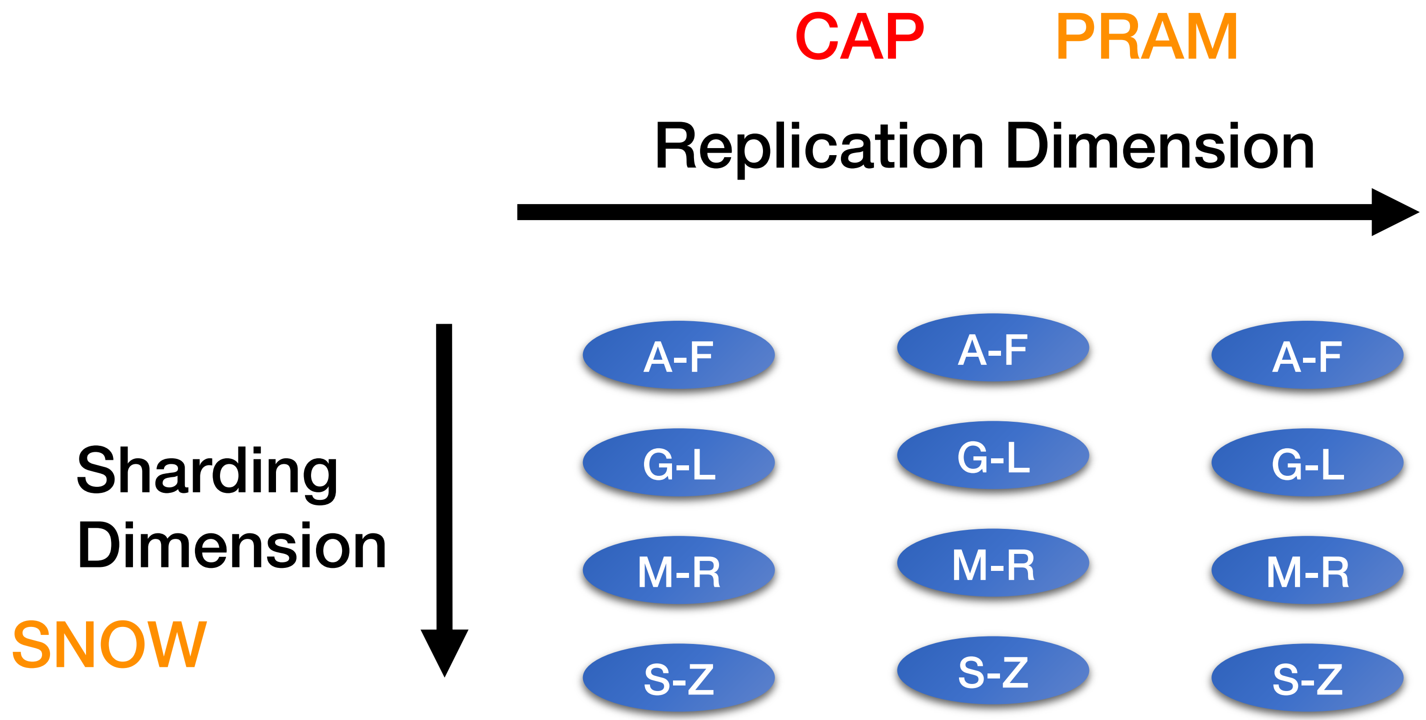
CAP

PRAM 1988 (Princeton)

Causal+ Consistency — e.g., Bayou

Eventual Consistency — e.g., Dynamo

Sharding vs. Replication



The SNOW Theorem [Lu et al. 2016]

- Focus on read-only transactions

- Are the 'ideal' read-only transaction possible?
  - Provide the strongest guarantees
  - AND
  - Provide the lowest possible latency?
    - (Same as eventual consistent non-transactional reads)

- No ☹

The SNOW Properties

[S]trict serializability

[N]on-blocking operations

[O]ne response per read

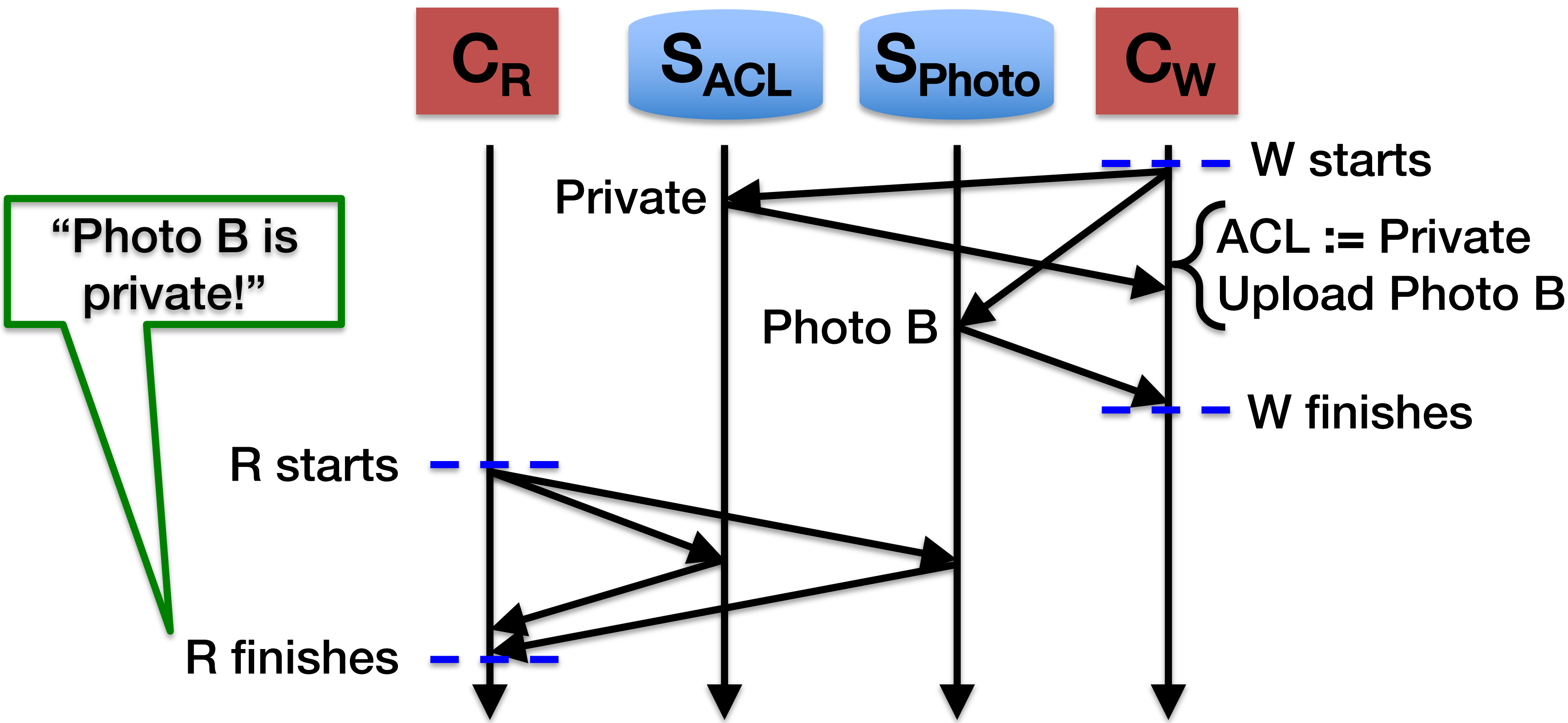
[W]rite transactions that conflict

Strongest Guarantees

Lowest Latency

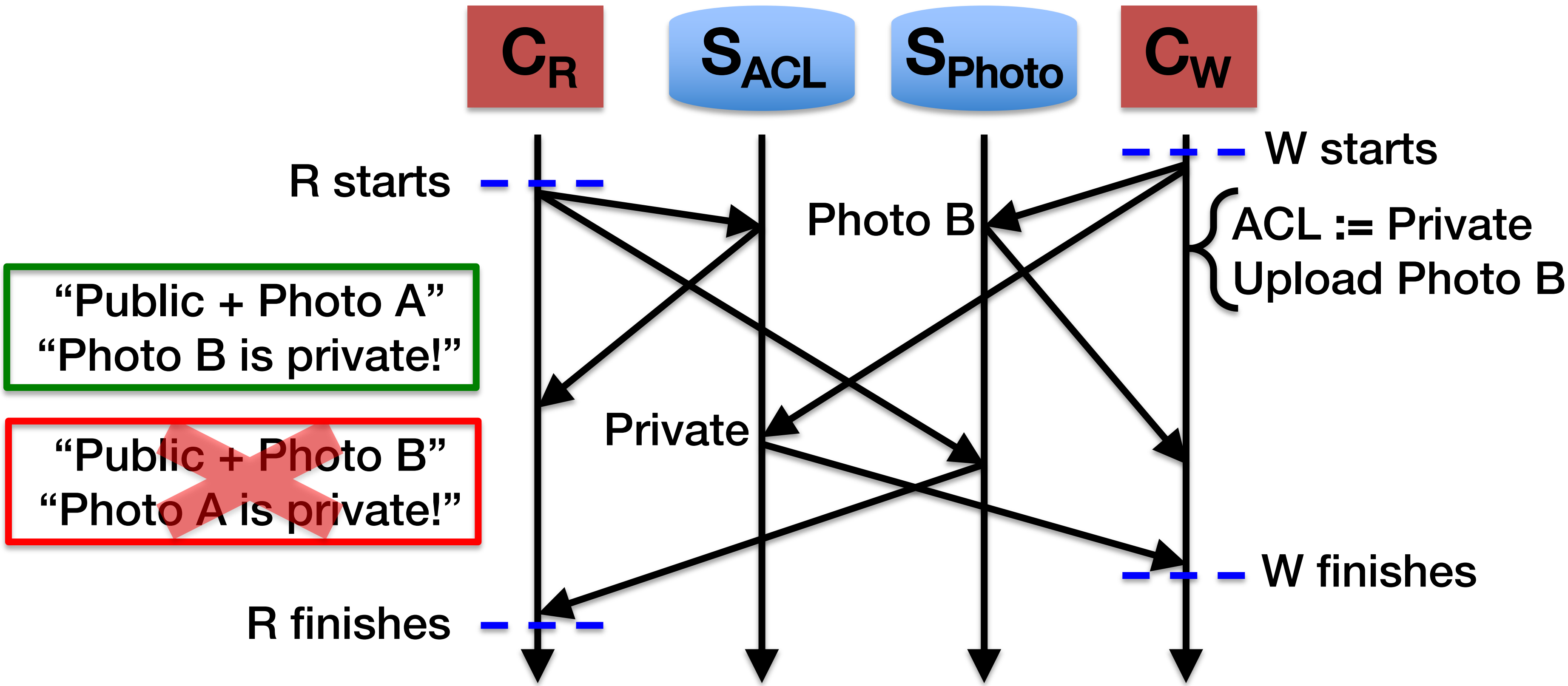
[S]trict Serializability

- Strongest model: real-time + total order



[S]trict Serializability

- Strongest model: real-time + total order



[N]on-blocking Operations

- Do not wait on external events
  - Locks, timeouts, messages, etc.
- Lower latency
  - Save the time spent blocking

[O]ne Response

- One round-trip
  - No message redirection
    - Centralized components: coordinator, etc.
  - No retries
  - Save the time for extra round-trips

- One value per response
  - Less time for transmitting, marshaling, etc.

[W]rite Transactions That Conflict

- Compatible with write transactions
  - Richer system model
  - Easier to program

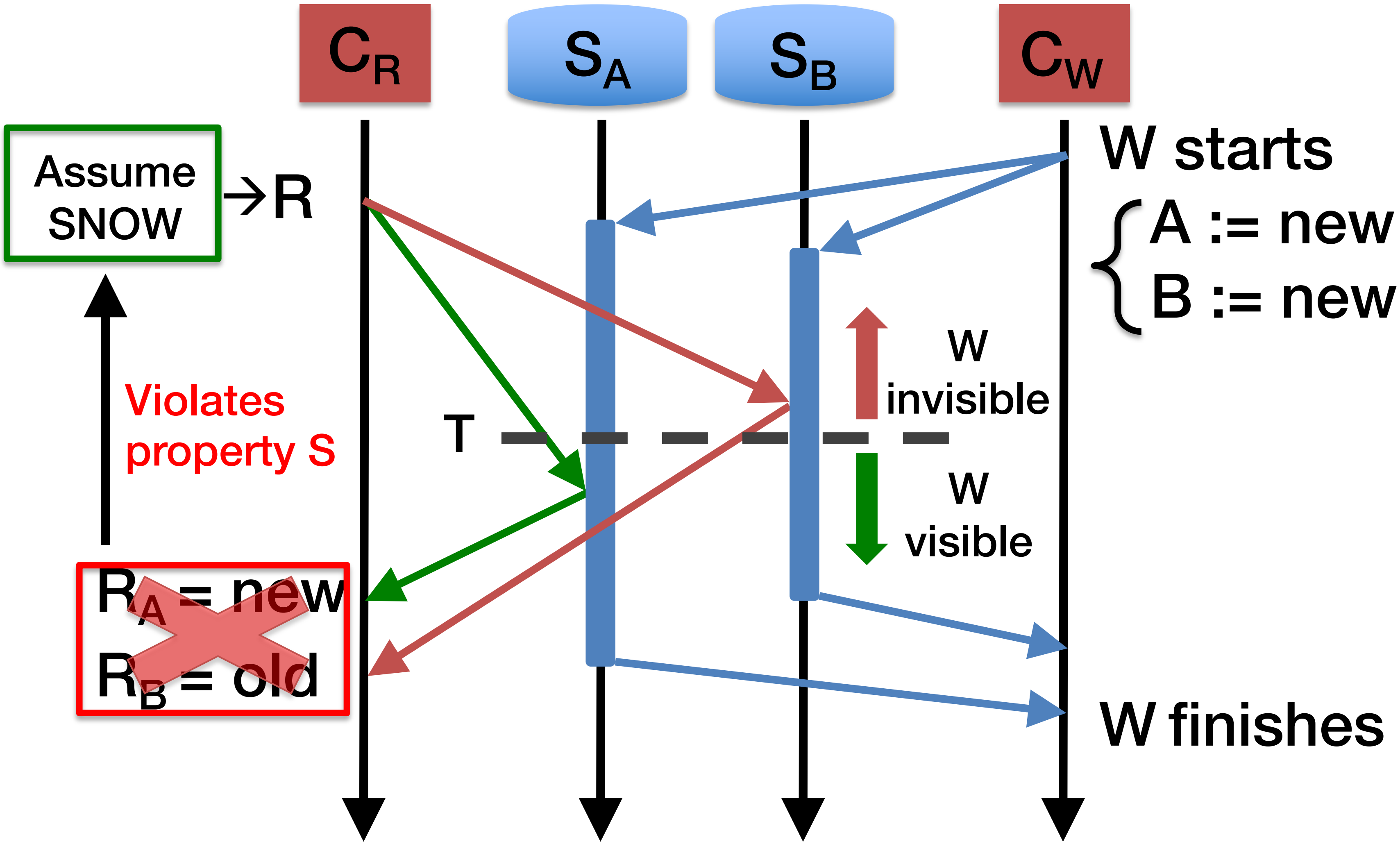
- Spanner has W
- COPS does not have W

The SNOW Theorem:

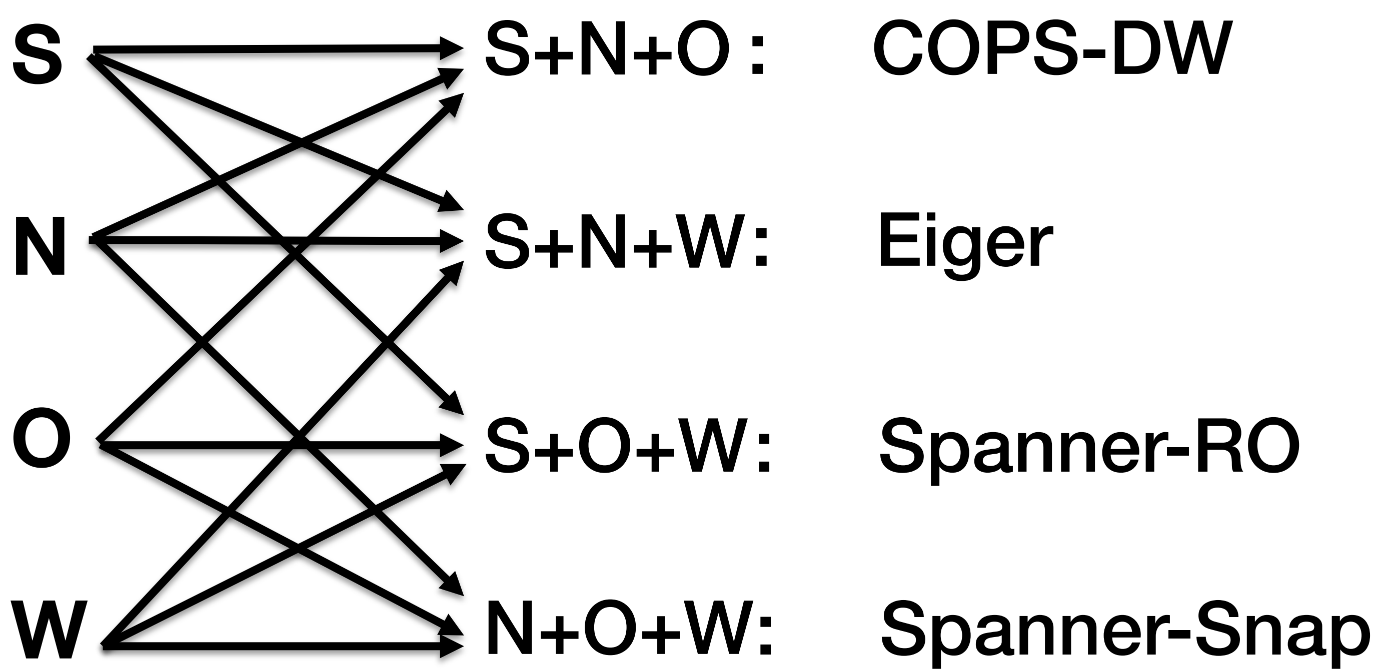
Impossible for read-only transaction algorithms to have all SNOW properties

Must choose strongest guarantees OR lowest latency for read-only transactions

Why SNOW Is Impossible [Intuition]

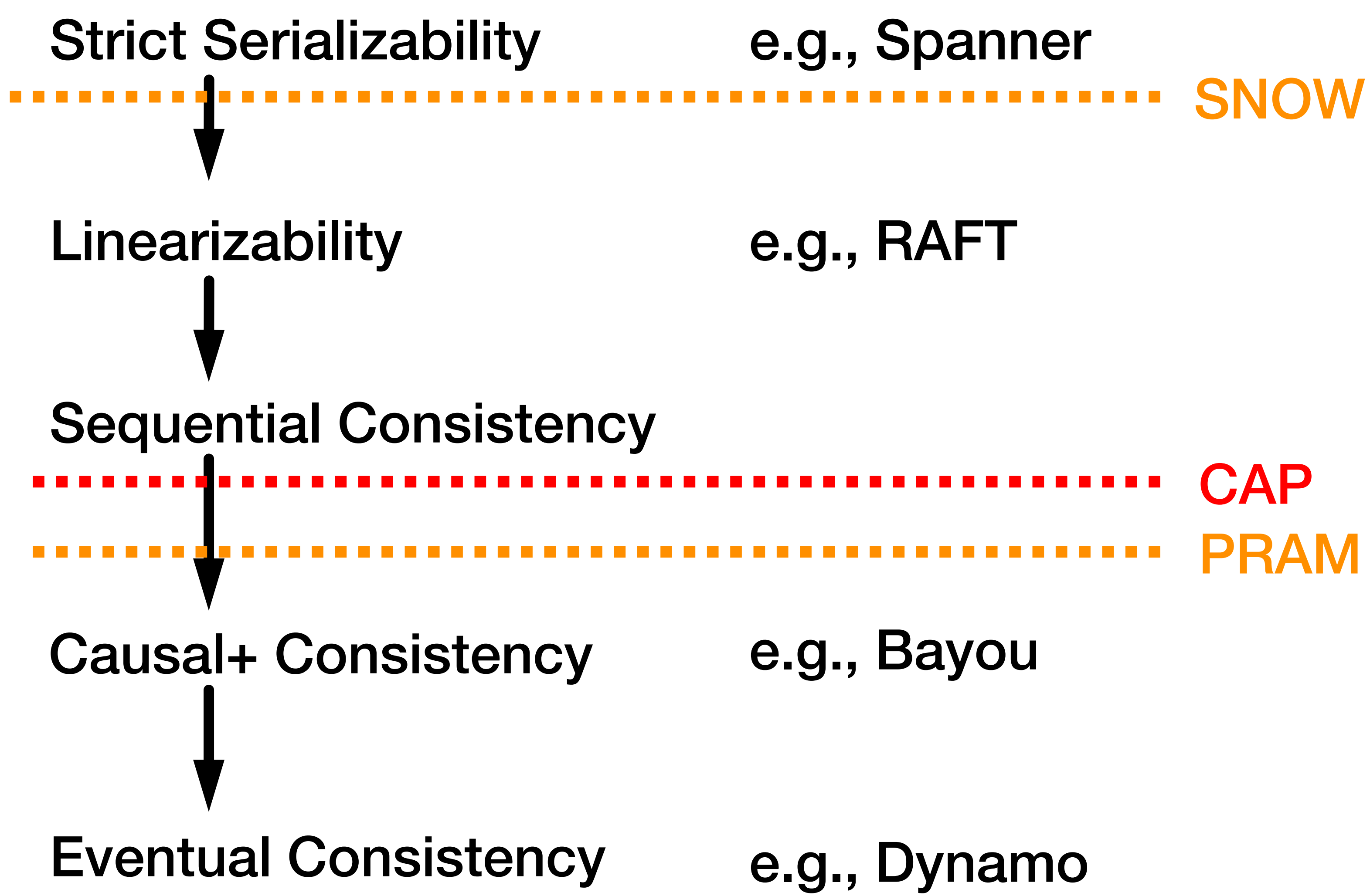


SNOW Is Tight



Spanner's read-only transaction interfaces provide both sides of tradeoff!

Consistency Hierarchy



Latency vs. Throughput

- Latency: How long operations take
  - All results so far about latency/availability

- Throughput: How many operations/sec

The NOCS Theorem [Lu et al. 2020]

- Focus on read-only transaction's latency and throughput
- Are the 'ideal' read-only transaction possible?
  - Provide the strongest guarantees
  - AND
  - Provide the lowest possible latency?
  - AND
  - Provide the highest possible throughput?
- No ☹

The NOCS Properties

[N]on-blocking operations

[O]ne response per read

[C]onstant metadata

Same As Simple Reads

[S]trict serializability

The NOCS Theorem:

Impossible for read-only transaction algorithms to have all NOCS properties

Must choose strongest consistency OR best performance for read-only transactions

"FLP"

- No deterministic 1-crash-robust consensus algorithm exists with asynchronous communication

Impossibility of Distributed Consensus with One Faulty Process

MICHAEL J. FISCHER

Yale University, New Haven, Connecticut

NANCY A. LYNCH

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

MICHAEL S. PATERSON

University of Warwick, Coventry, England

Abstract. The consensus problem involves an asynchronous system of processes, some of which may be unreliable. The problem is for the reliable processes to agree on a binary value. In this paper, it is shown that every protocol for this problem has the possibility of nontermination, even with only one faulty process. By way of contrast, solutions are known for the synchronous case, the "Byzantine Generals" problem.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols–*protocol architecture*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems–*distributed applications; distributed databases; network operating systems*; C.4 [**Performance of Systems**]: Reliability, Availability, and Serviceability; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation–*parallelism*; H.2.4 [**Database Management**]: Systems–*distributed systems*; *transaction processing*

General Terms: Algorithms, Reliability, Theory

Additional Key Words and Phrases: Agreement problem, asynchronous system, Byzantine Generals problem, commit problem, consensus problem, distributed computing, fault tolerance, impossibility proof, reliability

FLP is the original impossibility result for distributed systems!

- Useful interpretation: no consensus algorithm can <u>always</u> reach consensus with an asynchronous network
  - Do not believe such claims!
- Led to lots and lots of theoretical work
  - (Consensus is possible when the network is reasonably well-behaved)

Conclusion

- Impossibility results tell you choices you must make in the design of your systems
- CAP: Fundamental tradeoff between availability and strong consistency (for replication)
- PRAM: Fundamental tradeoff between latency and strong consistency (for replication)
- SNOW: Fundamental tradeoff between latency and strong guarantees (for sharding)
- NOCS: Fundamental tradeoff between performance (latency and throughput) and strong guarantees (for sharding)