Impossibility Results:

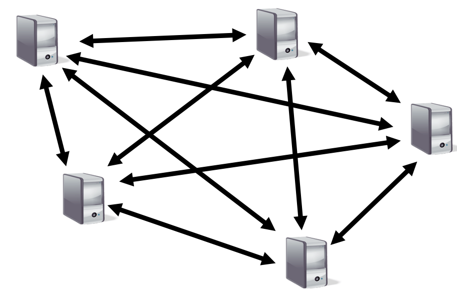
CAP, PRAM, SNOW, & FLP

COS 418: Distributed Systems
Lecture 20

Mike Freedman

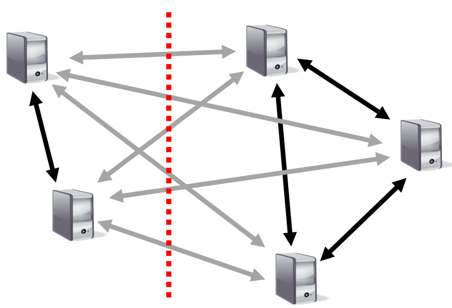
1

Network Partitions Divide Systems



2

Network Partitions Divide Systems



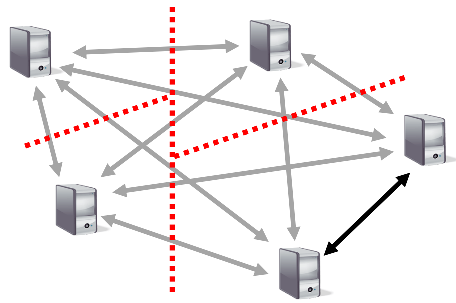
3

How can we handle partitions?

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

4

How About This Set of Partitions?



5

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

6

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

7

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 other begins, then no real-time order. But must be some total order.

8

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

9

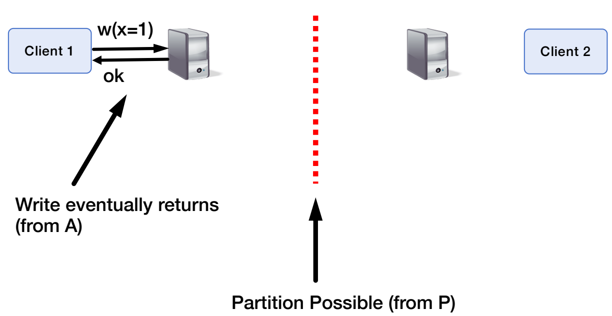
CAP Theorem [Gilbert Lynch 02]

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

10

CAP Theorem [Gilbert Lynch 02]

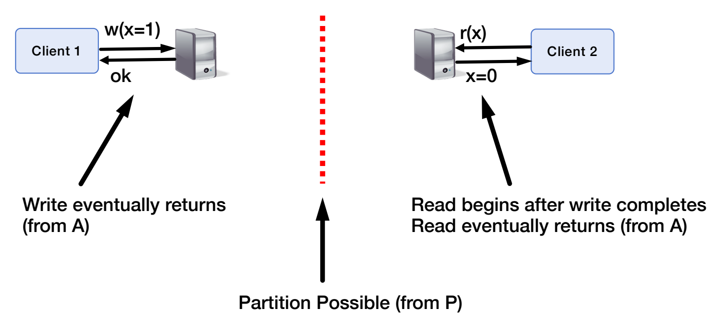
Assume to contradict that Algorithm *A* provides all of CAP



11

CAP Theorem [Gilbert Lynch 02]

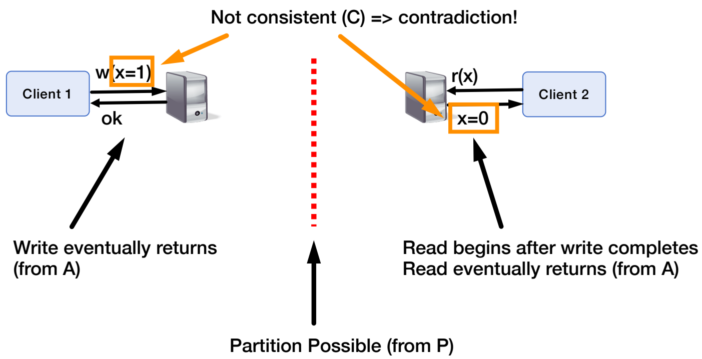
Assume to contradict that Algorithm *A* provides all of CAP



12

CAP Theorem [Gilbert Lynch 02]

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



13

CAP Interpretation Part 1

- Cannot “choose” no partitions
  - 2-out-of-3 interpretation doesn’t make sense
  - Instead, availability OR consistency?
- That is: Fundamental tradeoff between availability and consistency
  - When designing system must choose one or the other, both are not possible

14

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)

15

Consistency Hierarchy

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

16

Impossibility Results Useful!!!!

- Fundamental tradeoff in design space:
  Must make a choice
- Avoids wasting effort trying to achieve impossible
- Tells us the best-possible systems we can build!

17

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 b/w consistency and latency!
- (Skipping proof, see presenter notes or papers)

18

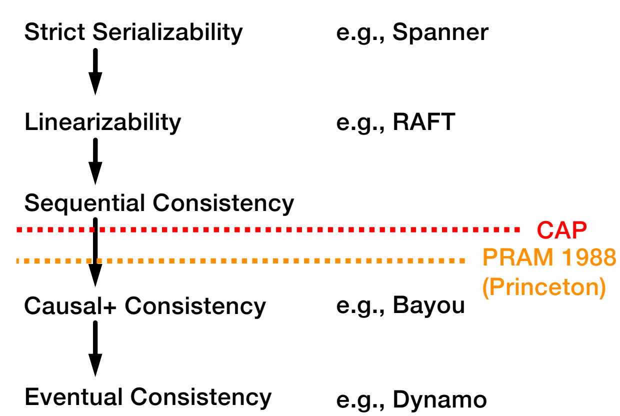
PRAM Theorem:

Impossible for sequentially consistent system to always provide low latency.

19

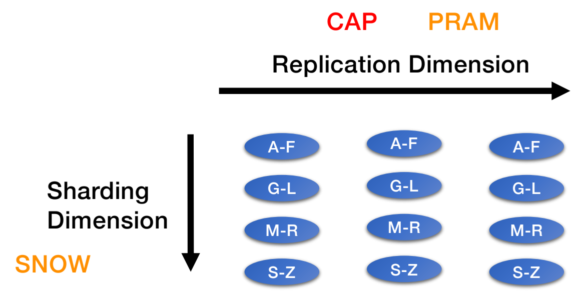
19

Consistency Hierarchy



20

Sharding vs. Replication



21

21

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 ☹

22

The SNOW Properties

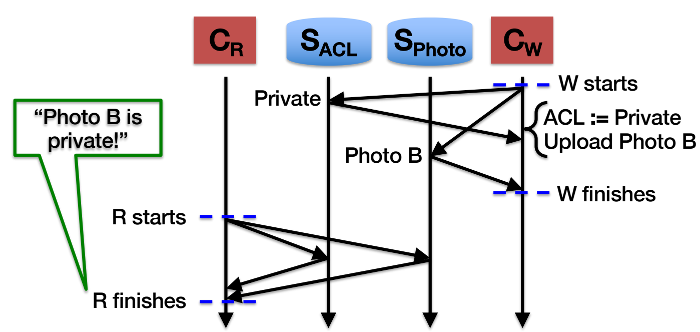
[S]trict serializability [N]on-blocking operations	Strongest Guarantees
[O]ne response per read [W]rite transactions that conflict	Lowest Latency

23

23

[S]trict Serializability

- Strongest model: real-time + total order

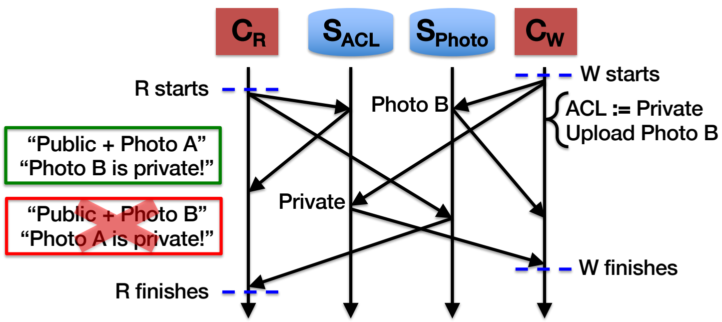


24

24

[S]trict Serializability

- Strongest model: real-time + total order



25

25

[N]on-blocking Operations

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

26

26

[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.

27

27

[W]rite Transactions That Conflict

- Compatible with write transactions
  - Richer system model
  - Easier to program
- Spanner has W
- COPS does not have W

28

28

The SNOW Theorem:

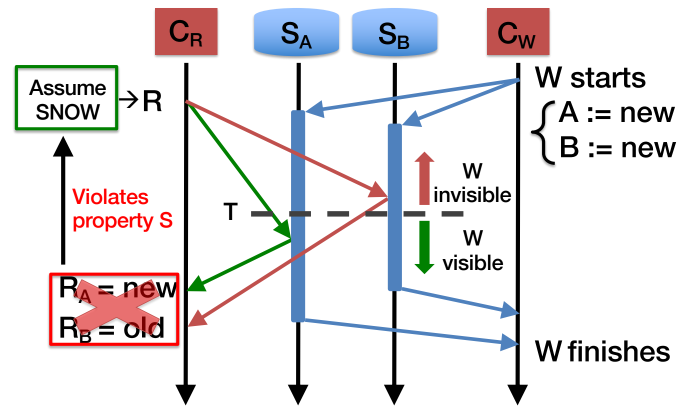
Impossible for read-only transaction algorithms to have all SNOW properties

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

29

29

Why SNOW Is Impossible [Intuition]



30

30

Consistency Hierarchy

Strict Serializability — e.g., Spanner

SNOW

↓ Linearizability — e.g., RAFT

↓ Sequential Consistency

CAP

PRAM

↓ Causal+ Consistency — e.g., Bayou

↓ Eventual Consistency — e.g., Dynamo

32

Latency vs. Throughput

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

33

33

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 ☹

34

34

The NOCS Properties

[N]on-blocking operations

[O]ne response per read

[C]onstant metadata

} Same As Simple Reads

[S]trict serializability

35

35

The NOCS Theorem:

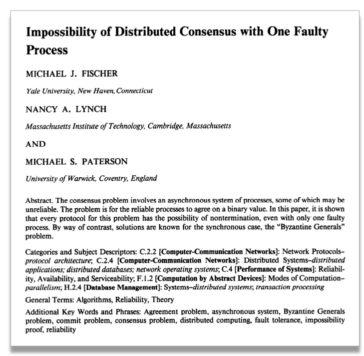
Impossible for read-only transaction algorithms to have all NOCS properties

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

36

36

"FLP" Result



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

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

37

37

FLP is the original impossibility result for distributed systems!

- Useful interpretation: no consensus algorithm can always 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)

38

38

Conclusion

Impossibility results tell you choices you must make in the design of your systems

- CAP: Fundamental tradeoff b/w availability and strong consistency (for replication)
- PRAM: Fundamental tradeoff b/w latency and strong consistency (for replication)
- SNOW: Fundamental tradeoff b/w latency and strong guarantees (for sharding)
- NOCS: Fundamental tradeoff b/w performance (latency and throughput) and strong guarantees (for sharding)

39

39