CPSC 416 Distributed Systems

Winter 2022 Term 2 (January 19, 2023)

Tony Mason (fsgeek@cs.ubc.ca), Lecturer

Logistics

Deadlines

Project 1 Deadline – January 23, 2023 (Extended). 75% cap for late submissions

Project 2 Deadline – January 23, 2023 (Original). 75% cap for late submissions

Zero Penalty Drop Date – January 23, 2023. W standing after this date

Project 3 Release: January 24, 2023. Initially Due: February 13, 2023

Project 4 Release: January 24, 2023. Initially Due: March 13, 2023

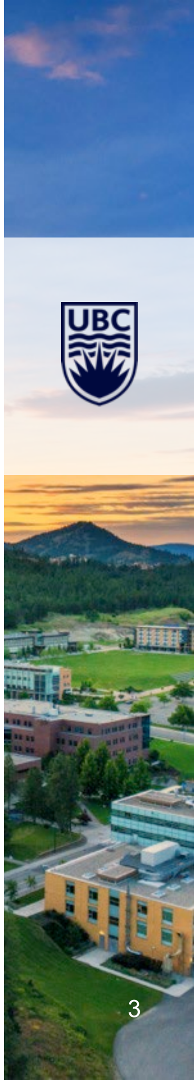
Project 5 Release: January 24, 2023. Initially Due: April 13, 2023

Note: all project work is due April 13, 2023. Late projects have a 75% score cap.

Alternate Path 1 & 2: Initial Proposal due January 30, 2023.

Instructor Office Hours:

- Zoom Office Hours (Tuesday) @ 13:00-14:00
- Discord (Casual) Office Hours (Thursday) @ 16:00-17:00



Recommended Reading

Distributed Snapshots: Determining Global States of Distributed Systems

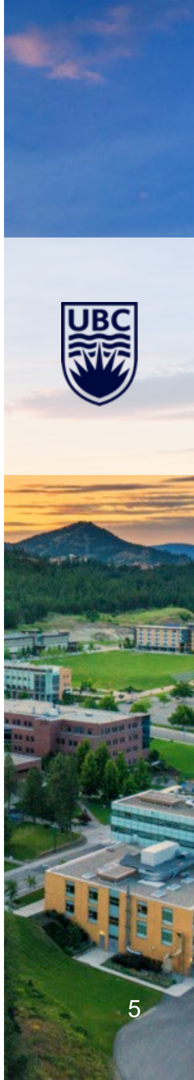
Distributed Computing: Principles, Algorithms, and Systems (Chapter 4)

Distributed Systems: Principles and Paradigms (See 8.6.2)

Alternative Path 2 (OSS Mode)

Objectives:

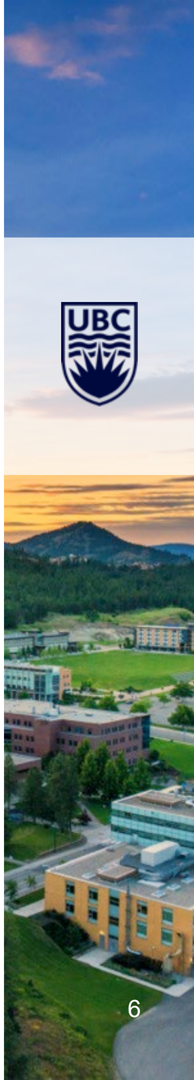
- Identify an existing open source project of interested *related to distributed systems*
- Identify a mentor within the community
- Define specific tasks you will undertake
  - Bugs
  - Features
  - Documentation
  - Tests
- Write a proposal
- Write an evaluation report
  - Identify your contributions (e.g., PRs, commits)



Alternative Path 2: Evaluation

Project

- Value of your contributions
- Project proposal quality
- Project report quality
- Mentor Feedback
- Instructional team review



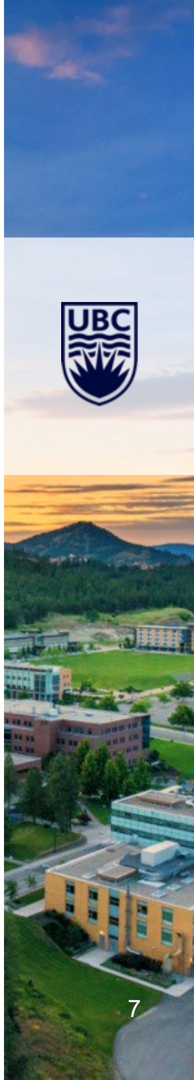
Alternative Path 2: Cost/Benefit Analysis

Benefits:

- Work on existing project
- *You* get to choose the OSS project
- Contribute to something real that will be useful
- Contribution makes you stand out to others
- Can add up to 40% to your final grade

Costs:

- You have to convince a mentor you’re worth mentoring
- Requires you learn how to work in an existing team environment
- You will likely accomplish less than you think that you will
- You may find yourself working on the project after this course is over



Some example projects

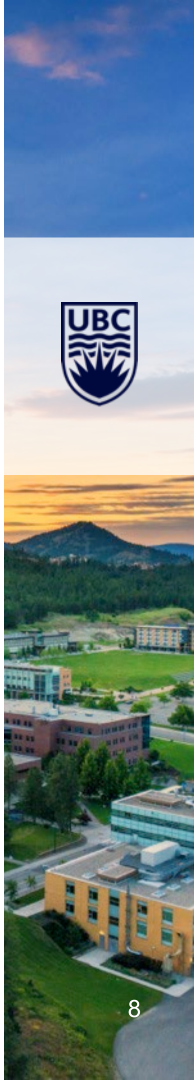
Apache projects

- Zookeeper
- Ignite
- Hadoop
- Kerberos
- Lucene
- CouchDB

Minio – Amazon S3 Open Source Equivalent

Riak – a distributed data storage system

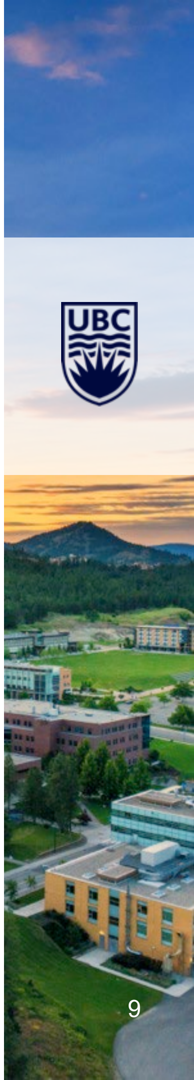
IPFS – Interplanetary File System



Clocks, Time, and Ordering Addendum

Remainder of Clocks, Time, and Ordering is recorded and on-line.

Will *not* be covered in class.

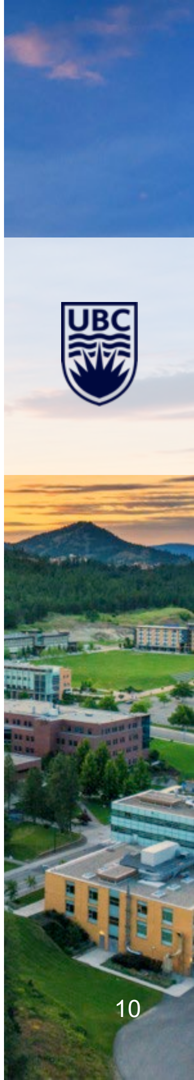


Questions?

Questions about the class?

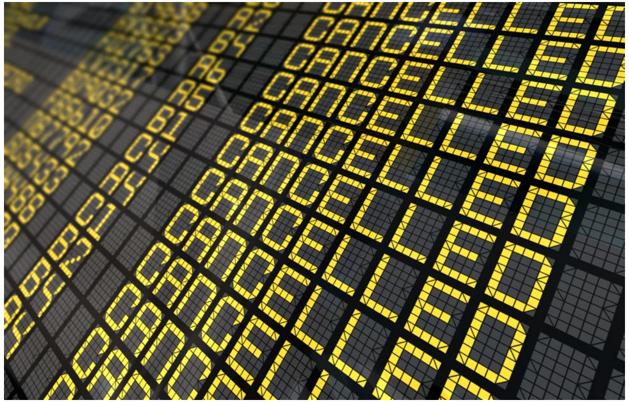
Questions about the previous lecture?

Funny stories to share?



Today’s Failure

Southwest Airlines Meltdown



Began December 21, 2022

Ended December 31, 2022 (sort of)

Root causes

- Scaling limits
- Weather delays (“perfect storm”)
- Manual processes (calling staff *manually* to redirect/reschedule)
- Under-investment
    - Scheduling Software was more than 20 years old
    - Not resilient

Not unique, either, since most major airlines have had similar problems.

Southwest Airlines Meltdown (Optional Reading)

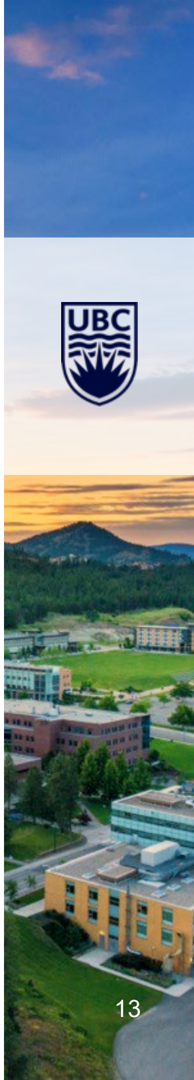
$821 *million* charge for disruption

[T]he system's operations have not changed much since the 1990s.

Why Southwest Airlines is struggling so much to accommodate passengers recently

The Shameful Open Secret Behind Southwest's Failure

(Note that this points out that this is not the *first* time they've had issues, just the worst.)



Lesson Goals

Distributed Systems

Understand challenges of global state detection

Explore algorithms for capturing distributed snapshots

- Actual state
- Possible states

Consider stable properties

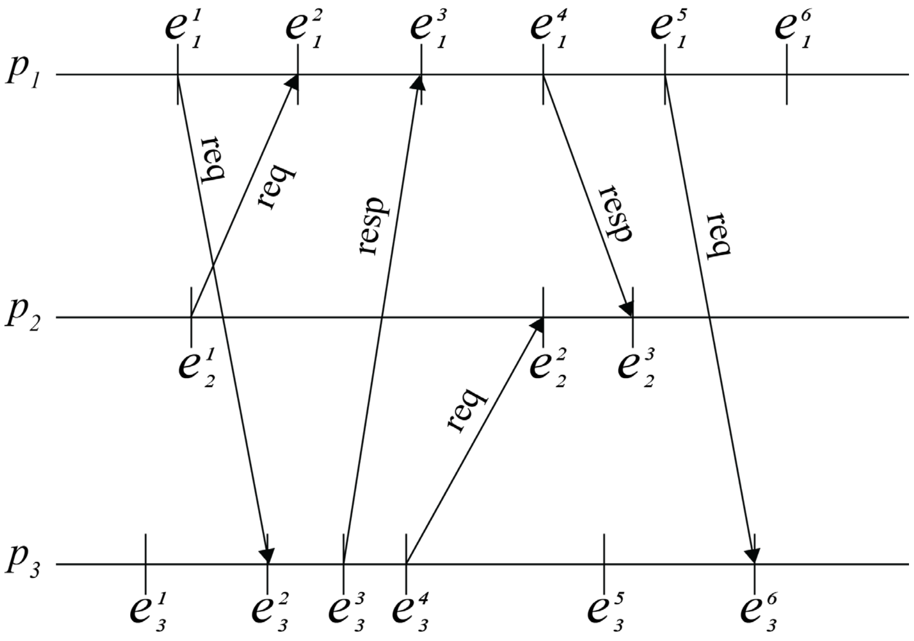
Global State Model

Process and Channels

- Process state = most recent event
- Channel state = inflight messages

State transitions:

- Process change = distributed state change



Run

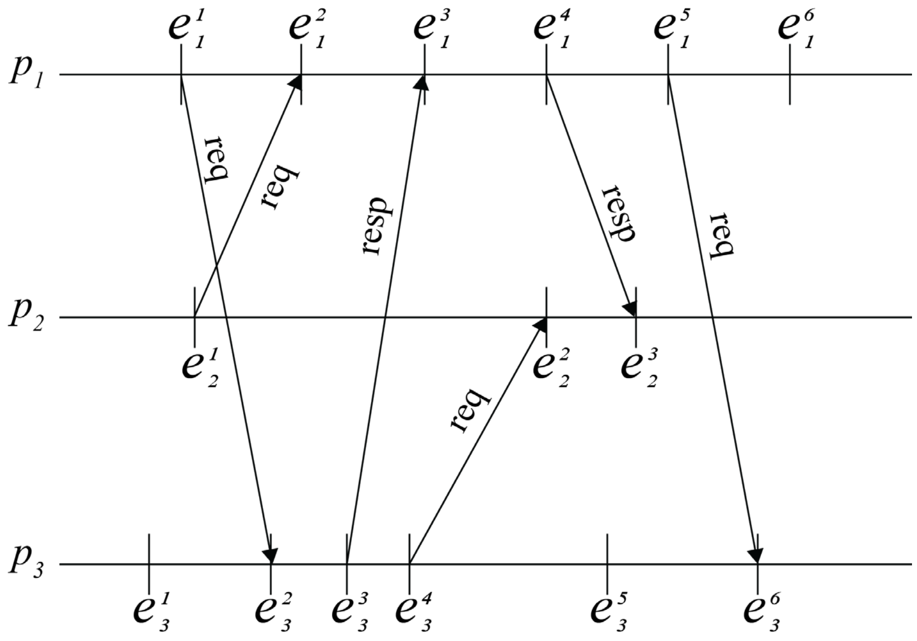
Any valid sequence of events

$$e_1^1, e_1^2, e_2^1, \ldots$$

Versus:

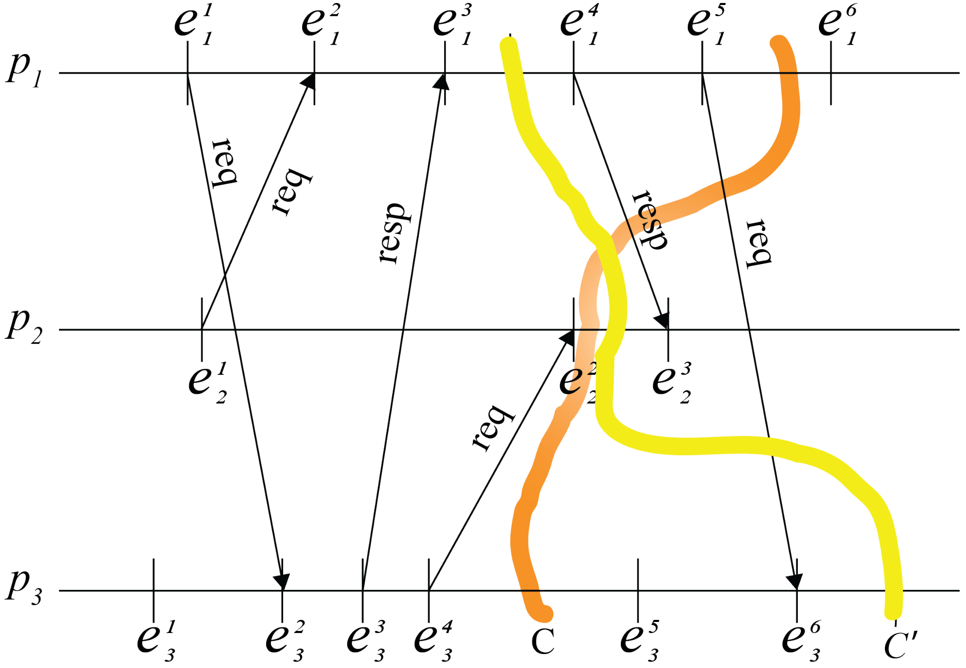
$$e_1^1, e_2^1, e_3^1, e_1^2, \ldots$$

Actual and observed



Distributed System State

Cut: snapshot across processes



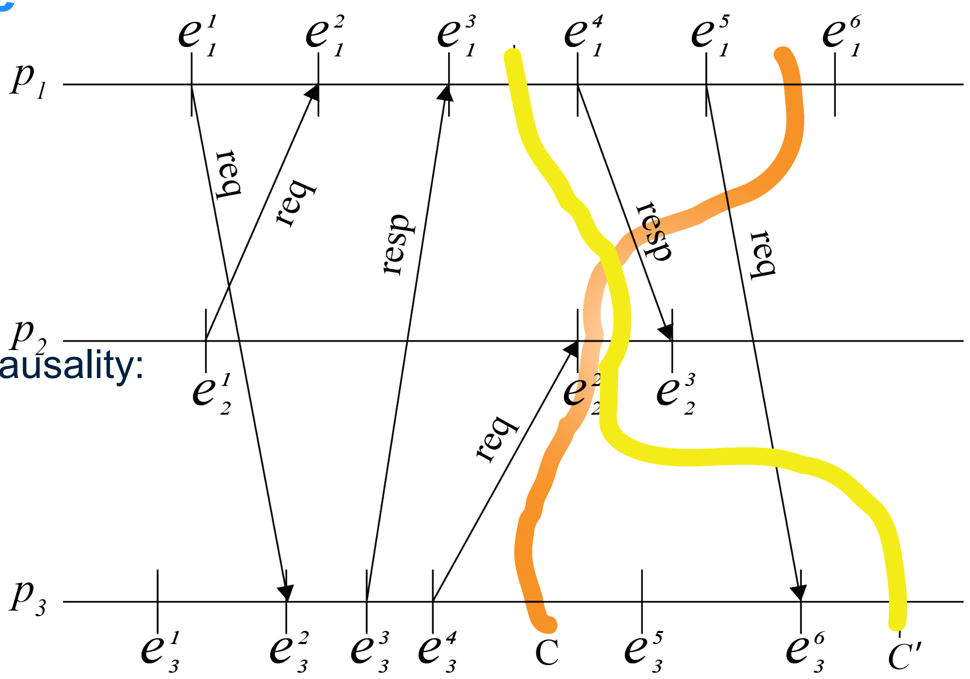
Distributed System State

Cut: snapshot across processes

Consistent cut: obeys causality

Inconsistent cut: cannot guarantee causality:

- Message *send* missing
- Message *receipt* observed
- C’ inconsistent
- C consistent



Distributed System Snapshot

External observer

- Stops the system
- Captures the state
- Resumes the system

Global snapshot is *consistent*

Question: can we get a consistent cut *without* a global observer

If we *can* then we won’t need an external observer

Recording Events

Process:

- Records any message sent *before* its snapshot
- Must not record any message sent *after* its snapshot

Snapshot requests are *messages* sent between processes.

Distributed Systems State Challenges

Do not rely upon an external observer

- No instantaneous snapshot

Do not have a global clock

- Ignore Spanner

Network variability

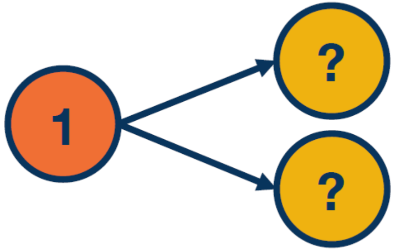
- No node in the network can reliably define event order

Non-determinism in Distributed Systems

Decoupled processes can perform operations in arbitrary order.

Deterministic operations are easy

Non-deterministic operations: event order is not known

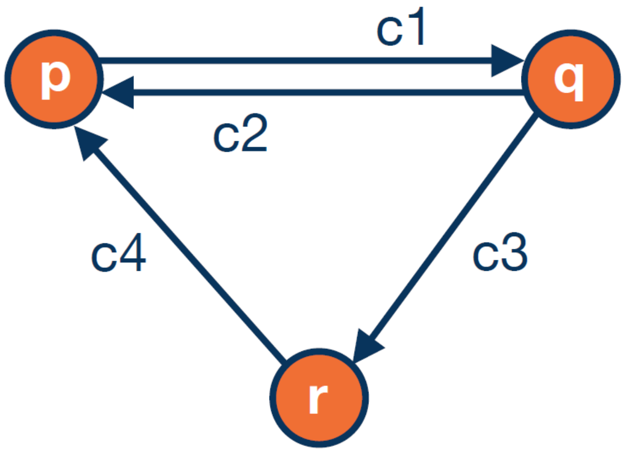


Network can make this happen

Formalize our model

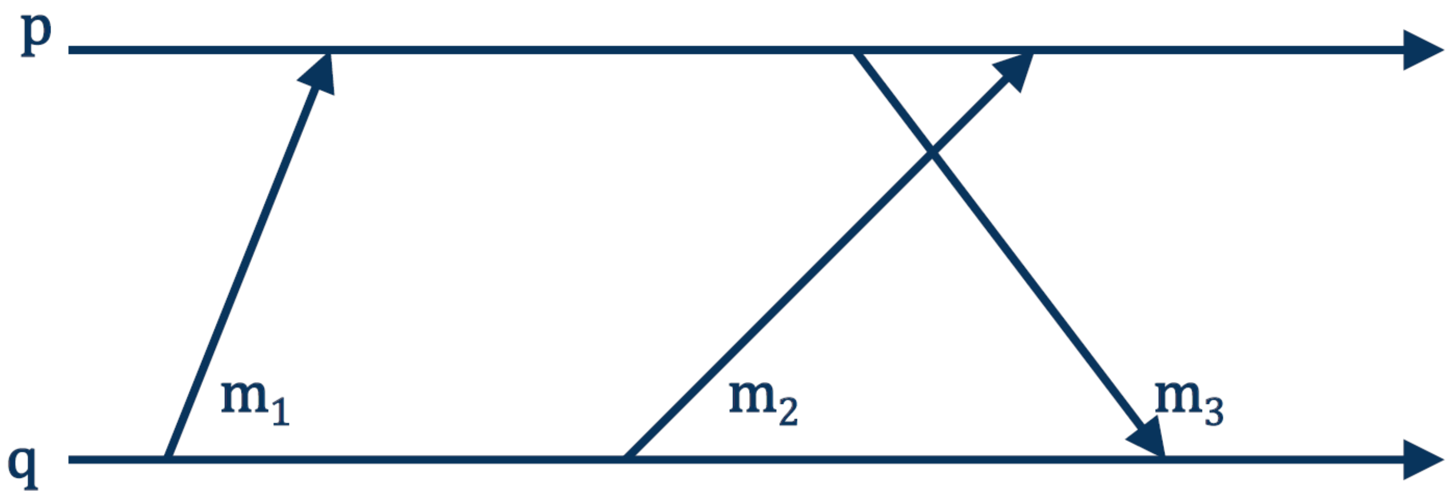
Processes: independent actors within the system

Channels: directed, first-in first-out (FIFO), no errors

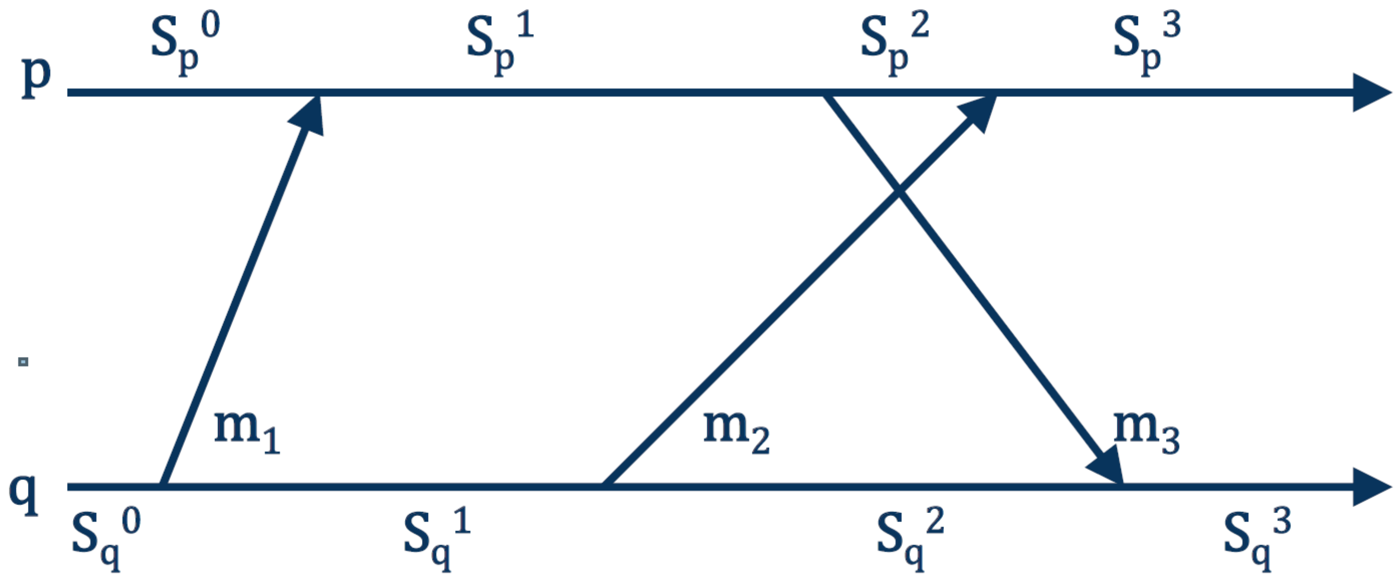


Consistent Cut Algorithm

Goal is to find a consistent cut with *only* processes and channels

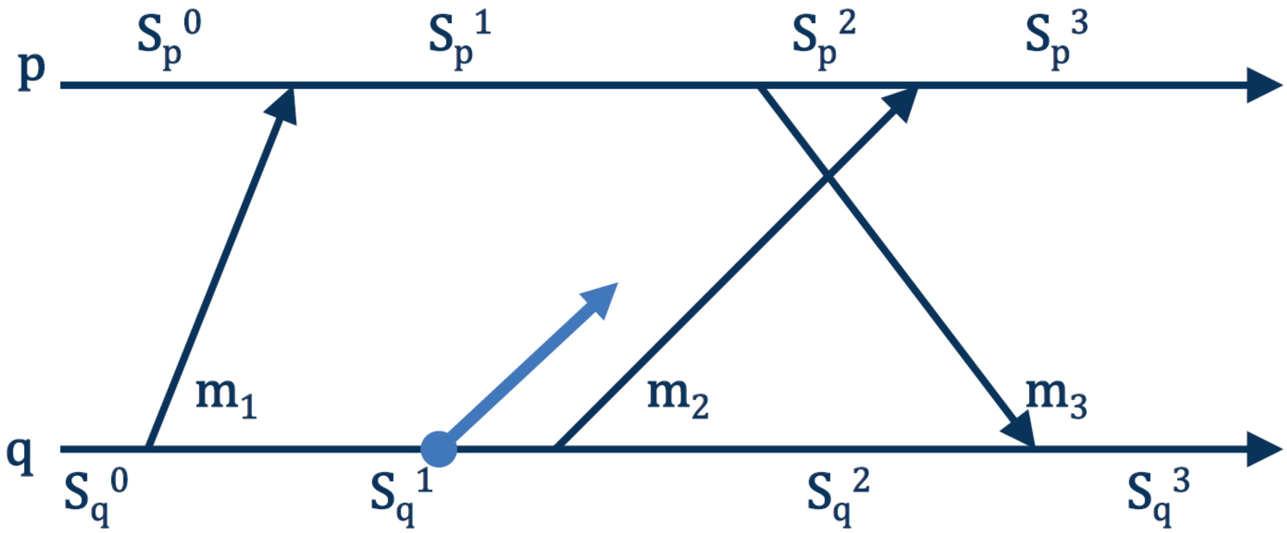


Process snapshots



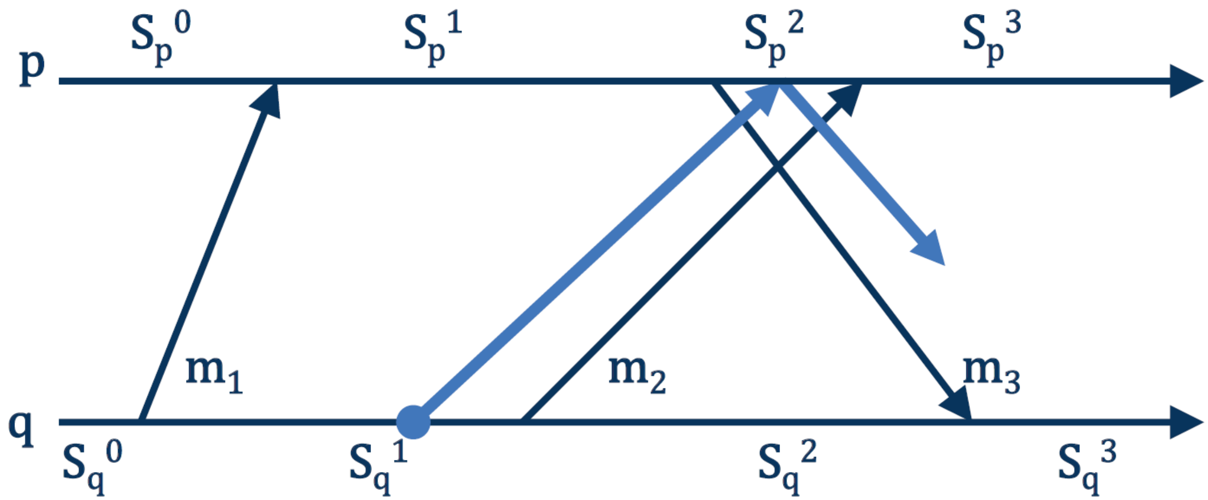
Initiate snapshot

Process q records state S_q^1 sends a *marker* to Process p



Capture second snapshot

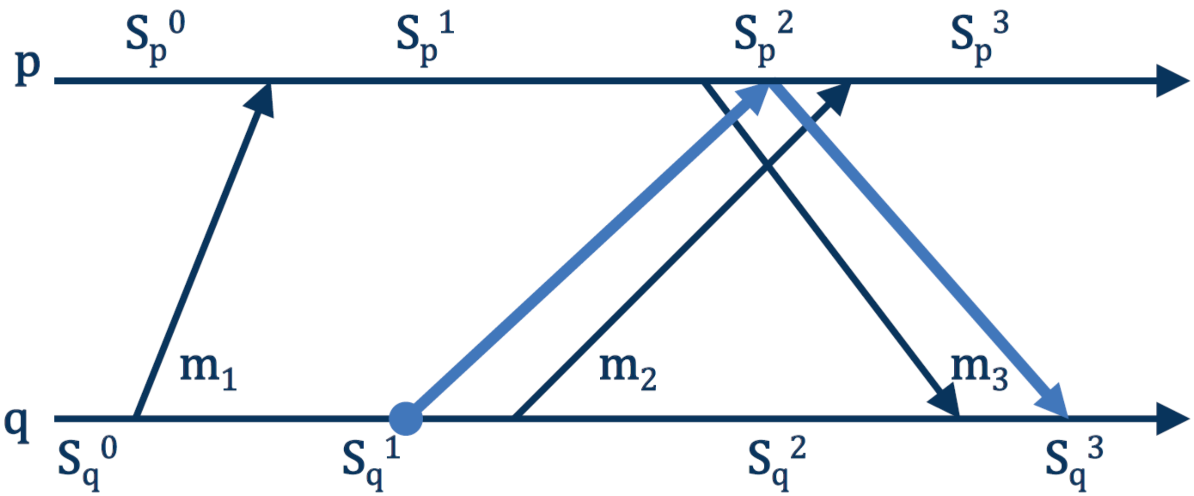
Process p records its state as S_p^2 and the channel state is empty.



Complete snapshot

Process q records snapshot state as S_q^3

Global state is $((S_p^2, S_q^1),(m_3, 0))$



Snapshot Algorithm (Generalized)

Initiator

- Saves local state
- Sends snapshot request (“marker”) on all its channels

Non-initiators:

- Receive *first* marker
  - Save state
  - Send marker on all its channels
  - Resume execution
  - Save incoming messages
  - Wait for another marker

Guarantees a consistent global state

Algorithm Assumptions

No failures

- Messages are intact
- Messages arrive only once

Communications are FIFO ordered, unidirectional

Processes capture:

- Local state
- State information received on channels

Note: this algorithm *does not* change normal execution of processes

Algorithm: Process Perspective

P as initiator:

- Records its own state
- Sends marker message on all its channels
- Resumes sending ordinary messages

P as non-initiator:

- If no recorded state:
  - Record its own state
  - Create empty message list
- If recorded state:
  - Message list = messages received since recording its state (modulo marker)

Chandy-Lamport Algorithm

Does not guarantee we get a state that existed

Guarantees we get a *consistent* state.

That is enough for us: consistency is key.

In fact, it gives us a *possible* global state.

Lattice Theory

This idea of partially ordered sets is *complex*

The field of studying these is known as **lattice theory**.

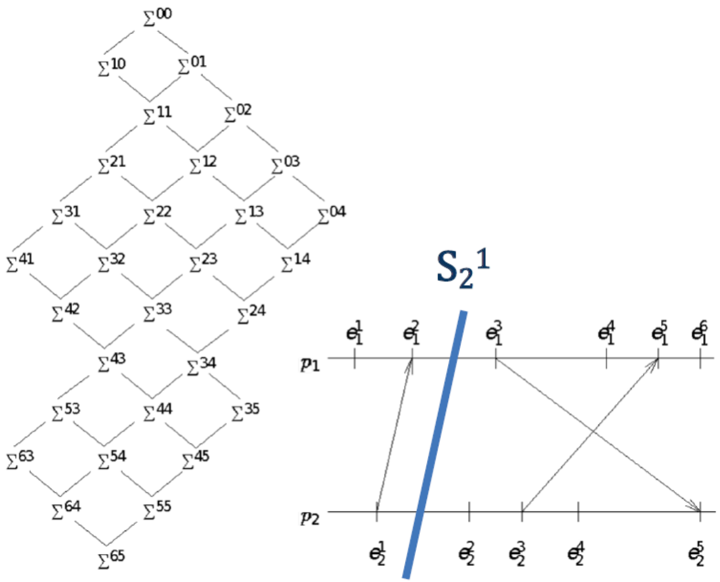
- Used for some data structures in distributed systems (e.g., CRDTs and MRDTs)

Additional Readings:

Notes on Lattice Theory (Chapters 1-6)

Notes on Lattice Theory (Chapters 7+)

Run Permutations



A Distributed Computation and the Lattice of its Global States

Permutations:

- $\Sigma^{10}, \Sigma^{11}, \Sigma\text{^}21$ for run e^{11}, e^{21}, e^{12} ...
- $\Sigma^{01}, \Sigma^{11}, \Sigma\text{^}21$ for run e^{21}, e^{11}, e^{12} ...

Equivalent: both end in global state Σ^{21}

Causal relationships are preserved

These are *isomorphic*.

“If I didn’t see the details and ended up with the same result, it didn’t matter.”

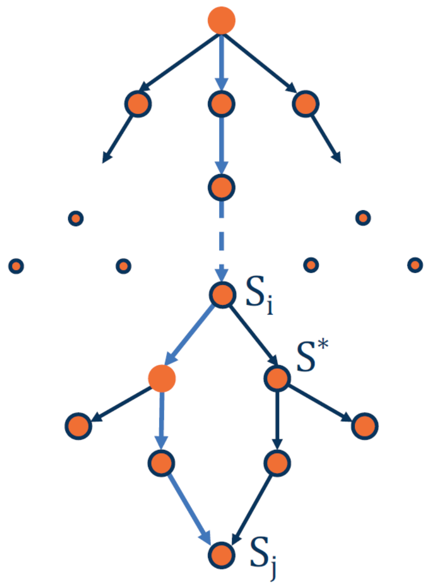
Global State Properties

Let

- S^* be the recorded state
- S_{eq} be the sequence of distributed computations performed by the system
- S_i is the true initial state of the system
- S_j is the true final state of the system

Then:

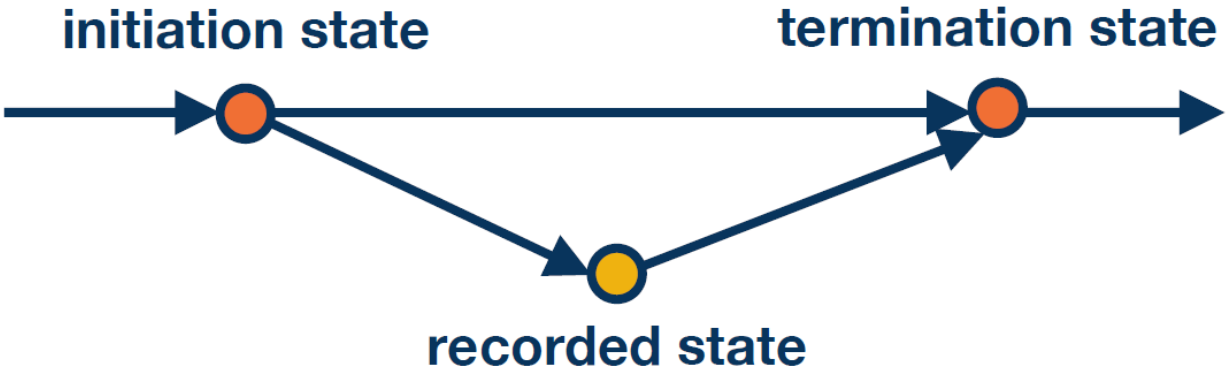
- S^* is reachable from S_i
- S_j is reachable from S^*
- $\exists$ a computation S^*_{eq} which is a permutation of S_{eq}
- Either $S^* = S_i$ or S_i occurs before S^* in S^*_{eq}
- Either $S_j = S^*$ or S^* occurs before S_j in S^*_{eq}



Theorem

The recorded state is reachable from the starting state.

The termination state is reachable from the recorded state.



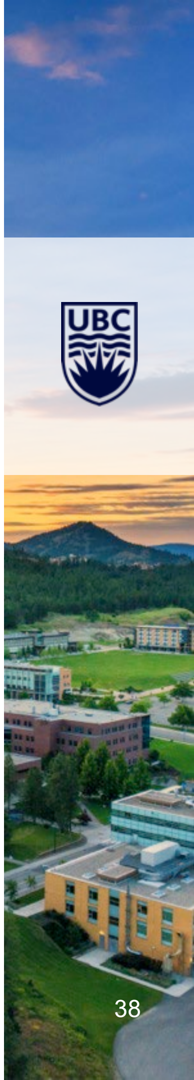
Global State: Stable Properties

Stable

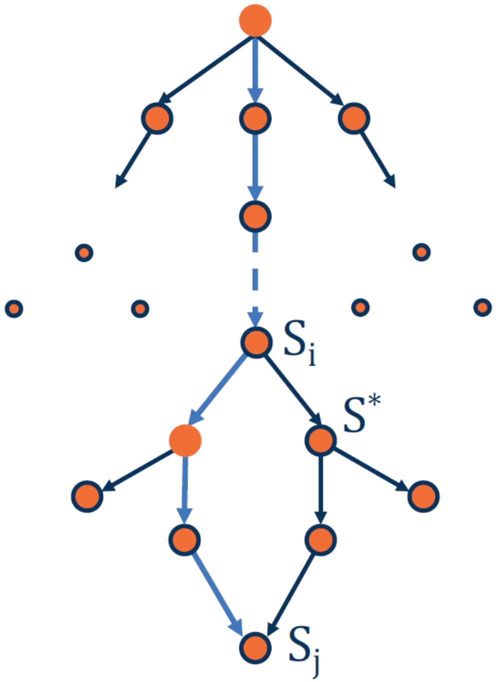
- If it becomes true for state S
  - True for all states S' reachable from S
- Otherwise it is not stable (so “if and only if”)

Examples:

- Deadlock
- Termination



Challenge



Evaluate a property without knowing the system state

Stability helps us reason about the system:

S^* is reachable from S_i

S_j is reachable from S^*

If we know S^* is stable then we know S_j is stable

If we know S^* is not stable then we know S_i is not stable

Unstable Properties

Transient errors:

- Buffer overflow
- Load spikes
- Race conditions (non-determinism)

State S^* may not have happened

Do distributed snapshots help here?

Definite versus possible state

If y is a stable property, then if $y(S^*)$ is true it is definitely true, regardless of the path taken

If y is *not* a stable property, then if $y(S^*)$ is true we don't know (it *could* be true).

Not perfect

- Perhaps we can do better with other techniques

Lesson Summary

What did we discuss?

Global state detection is challenging in a distributed system

Distributed snapshot algorithm can describe *a* possible state

- Isomorphic
- Identifies stable properties

We can (and will) build on this.

Questions?


UBC
THE UNIVERSITY OF BRITISH COLUMBIA