Clocks, Snapshots

Arvind Krishnamurthy
University of Washington

- Why do we need to order events in a distributed system?

Distributed make

- Distributed file servers holds source and object files
- Clients specify modification time on uploaded files
- Use timestamps to decide what needs to be rebuilt
  - if object O depends on source S, and
  - O.time < S.time, rebuild O
- What can go wrong?

Another example

- Remove boss as friend
- Post: “My boss is the worst, I need a new job!”
  - Friendship links, posts, privacy settings stored across a large number of distributed servers
  - lots of copies of data: replicas, caches, cross-datacenter replication, etc.
- Don’t want to get a concurrent read to see the wrong order!

Two approaches

- Synchronize physical clocks
- Logical clocks

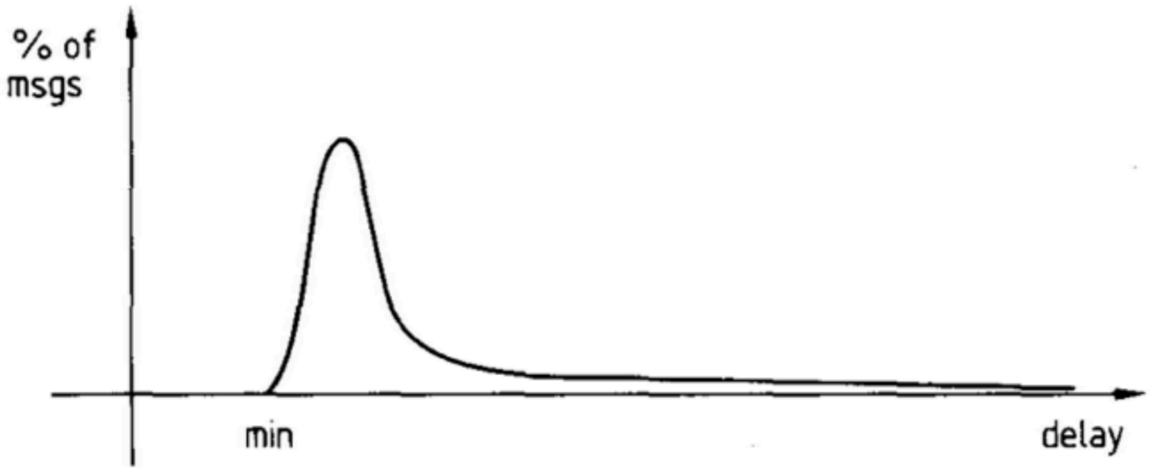
- Design a scheme that synchronizes physical clocks
  - What do you think are the sources of inaccuracy?
  - Why is clock synchronization hard?
  - How to make this scalable?

Simplest Approach

- Designate one server as the master
- Master periodically broadcasts time
- Clients receive broadcast, set their clock to the value in the message
- Is this a good approach?

Variations in Network Latency

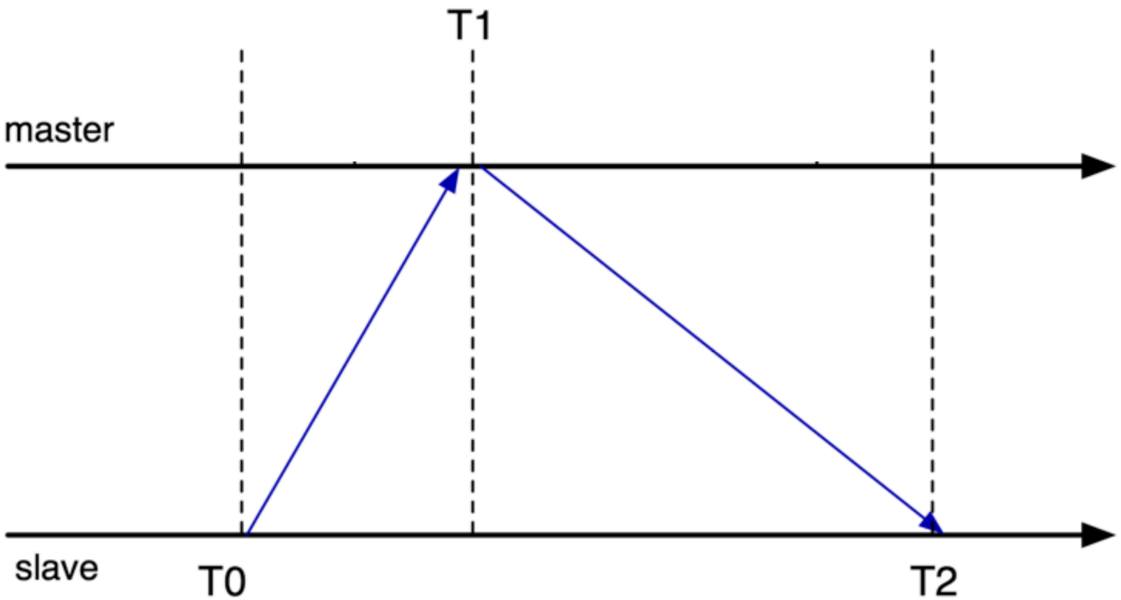
- Latency can be unpredictable and has a lower bound



- Simple approach: Designated server broadcasts time, Clients receive broadcast, set their clock to the value in the message + minimum delay

Interrogation Based Approach

- Client sends a roundtrip message to query server's time
- Set's client's clock to server's clock + half of RTT



- Worst case error (if we know the min latency): (T2-T0)/2 - min

Practical Realization

- NTP uses an interrogation-based approach, plus:
  - taking multiple samples to eliminate ones not close to min RTT
  - averaging among multiple masters

- PTP adds hardware timestamping support to track latency introduced in network

Clock synchronization measurements

- Within a datacenter: ~20-50 microseconds
- Across datacenters: tens of milliseconds

Logical Clocks

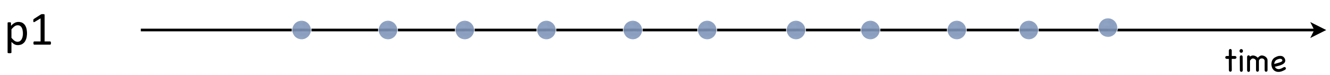
- another way to keep track of time
- based on the idea of causal relationships between events
- doesn’t require any physical clocks

Events and Histories

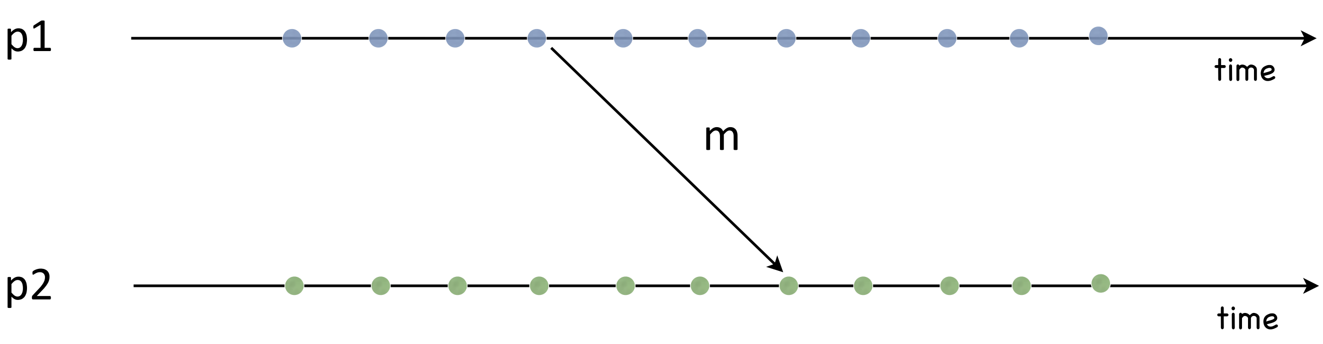
- Processes execute sequences of events
- Events can be of 3 types: local, send, and receive
- The local history of a process is the sequence of events executed by process

Ordering events

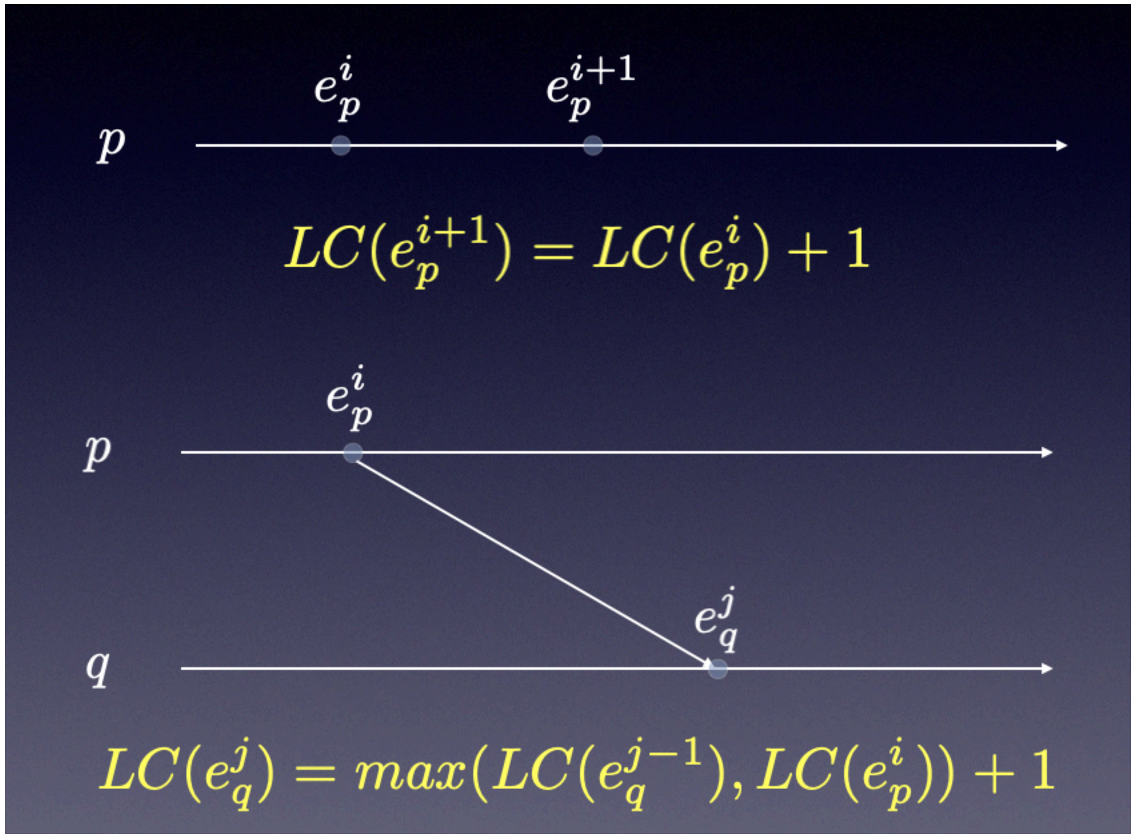
- Observation 1:
  - Events in a local history are totally ordered



- Observation 2:
  - For every message, send precedes receive



Lamport Clock: Increment Rules



- Timestamp m with TS(m) = LC(send(m))

Discussion

- What are the strengths of Lamport clocks?
- What are the limitations of Lamport clocks?

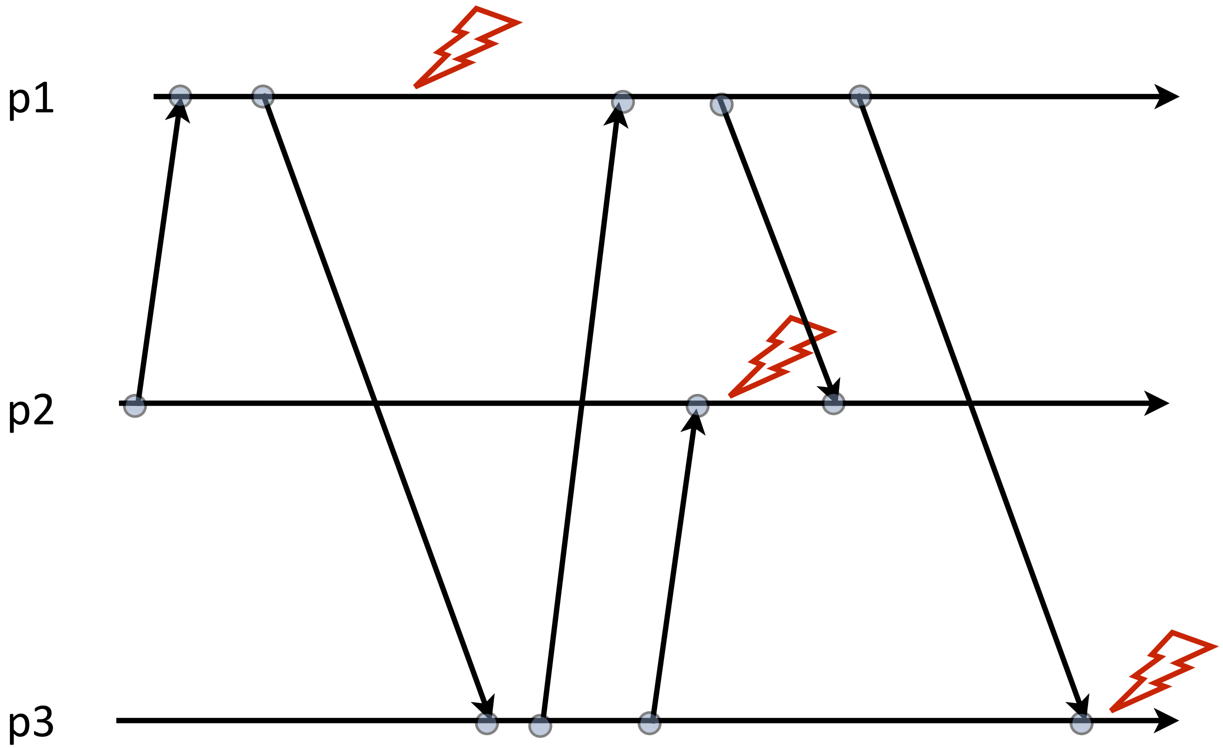
Examples of Global Predicates

- Token ring networks
  - Nodes arranged in a ring
  - Node can transmit to any other node when it has a "token"
  - Node passes along to another node when it is done sending a message
  - Tokens sometimes get lost or corrupted
- Global predicate: is there a token in the network?

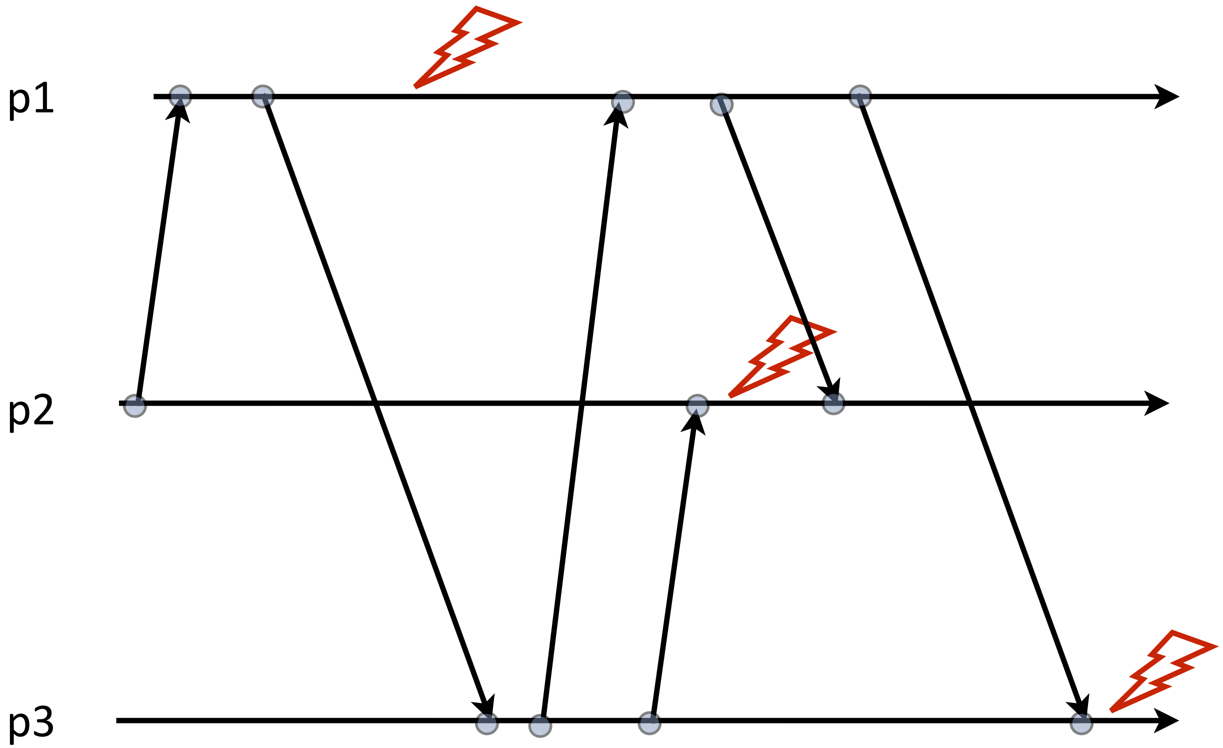
Global states and clocks

- Need to reason about global states of a distributed system
- Global state: processor state + communication channel state
- Consistent global state: causal dependencies are captured
- Use virtual clocks to reason about the timing relationships between events on different nodes

Space Time Diagrams



Cuts



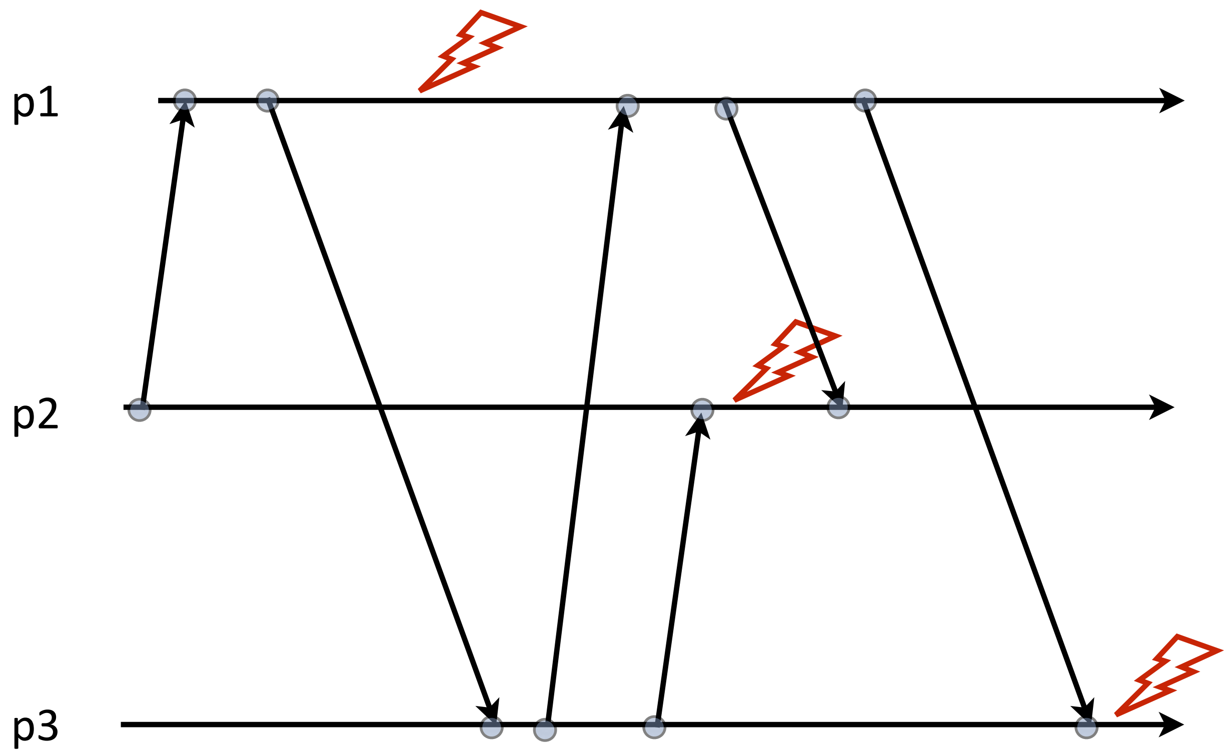
A cut C is a subset of the global history of H

The frontier of C is the set of events

Consistent Cuts

- A cut is consistent if
  - e2 is in the cut and if e1 happens before e2
    - then e1 should also be in the cut
- A consistent global state is one corresponding to a consistent cut

Inconsistent Cut (or global state)



Consistent Global States

- Can we use Lamport Clocks as part of a mechanism to get globally consistent states?

Global Snapshot

- Develop a simple global snapshot protocol
- Refine protocol as we relax assumptions
- Record:
  - processor states
  - channel states
- Assumptions:
  - FIFO channels
  - Each message timestamped with Lamport Clock

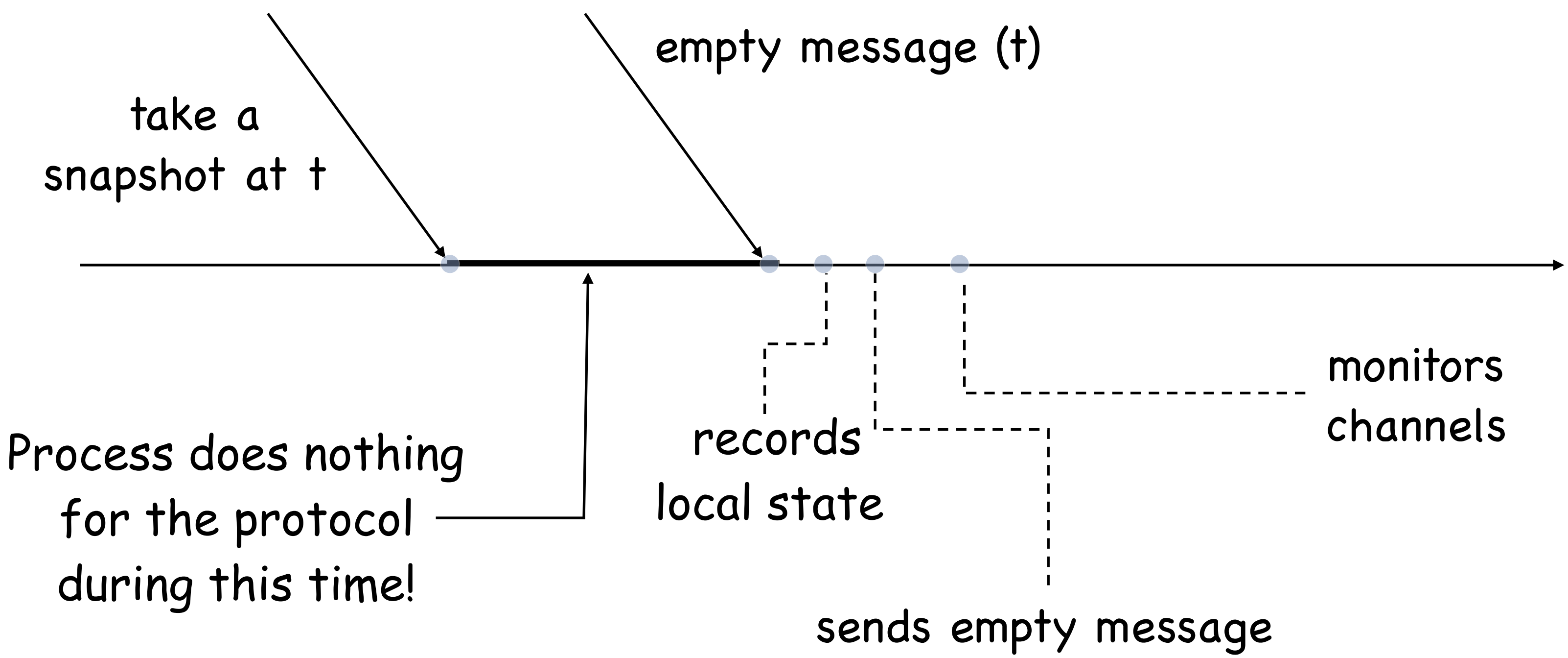
Snapshot Version 1

- p0 selects t
- p0 sends “take a snapshot at t” to all processes
- when clock of p reads t then:
  - records its local state
  - sends an empty message along its outgoing channels
  - starts recording messages received on each of incoming channels
  - stops recording a channel when it receives first message with timestamp greater than or equal to t

Snapshot Version 2

- p0 processor selects t
- p0 sends "take a snapshot at t" to all processes; it waits for all of them to reply and then sets its logical clock to t
- when clock of p reads t then
  - records its local state
  - sends an empty message along its outgoing channels
  - starts recording messages received on each incoming channel
  - stops recording a channel when receives first message with timestamp greater than or equal to t

Relaxing synchrony



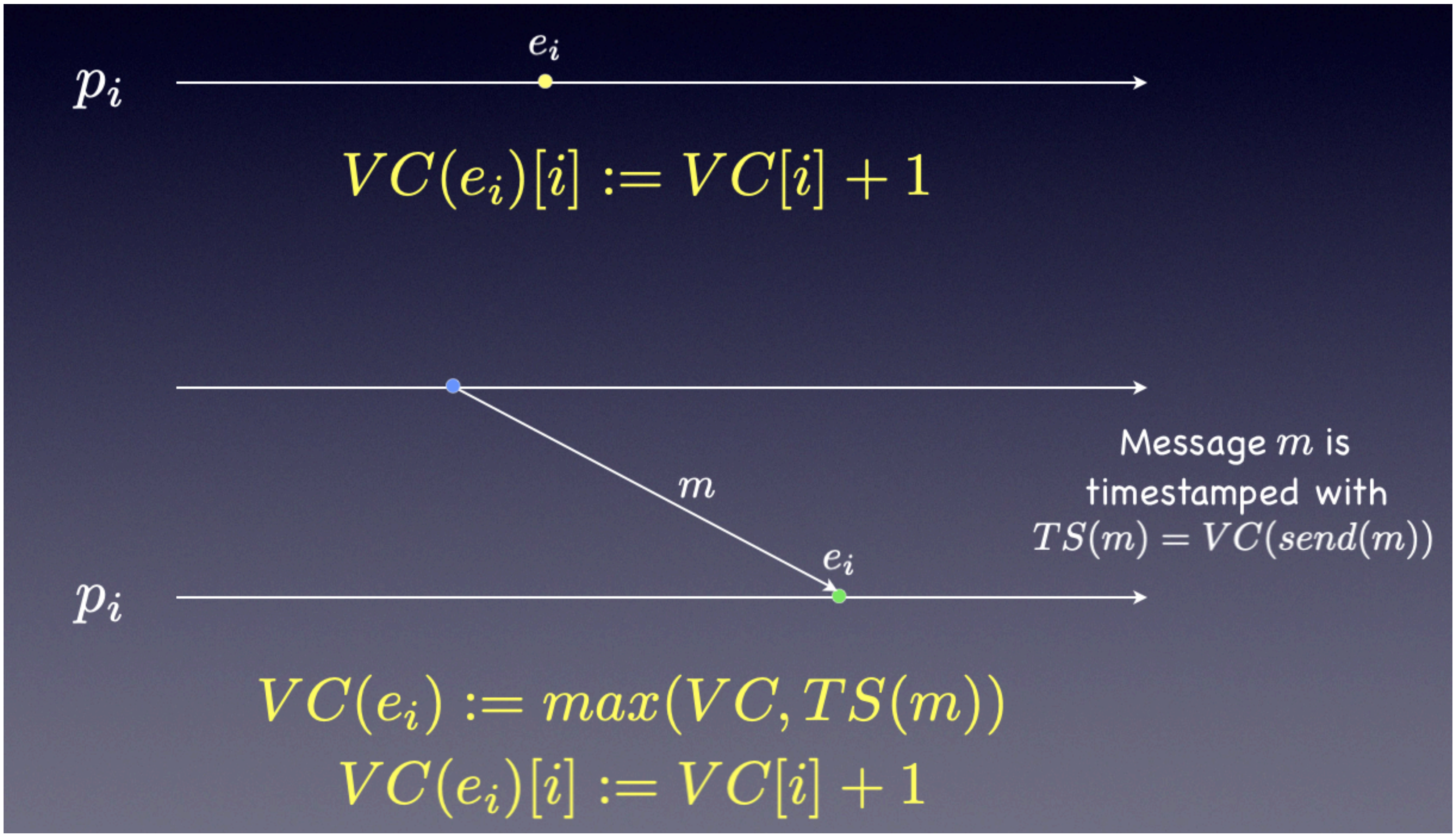
Snapshot Version 3

- processor p_0 sends itself “take a snapshot“
- when p_i receives “take a snapshot” for the first time from p_j:
  - records its local state
  - sends “take a snapshot” along its outgoing channels
  - sets channel from p_j to empty
  - starts recording messages received over each of its other incoming channels
- when pi receives “take a snapshot” beyond the first time from p_k:
  - stops recording channel from p_k
- when p_i has received “take a snapshot” on all channels, it sends collected state to p_0 and stops.

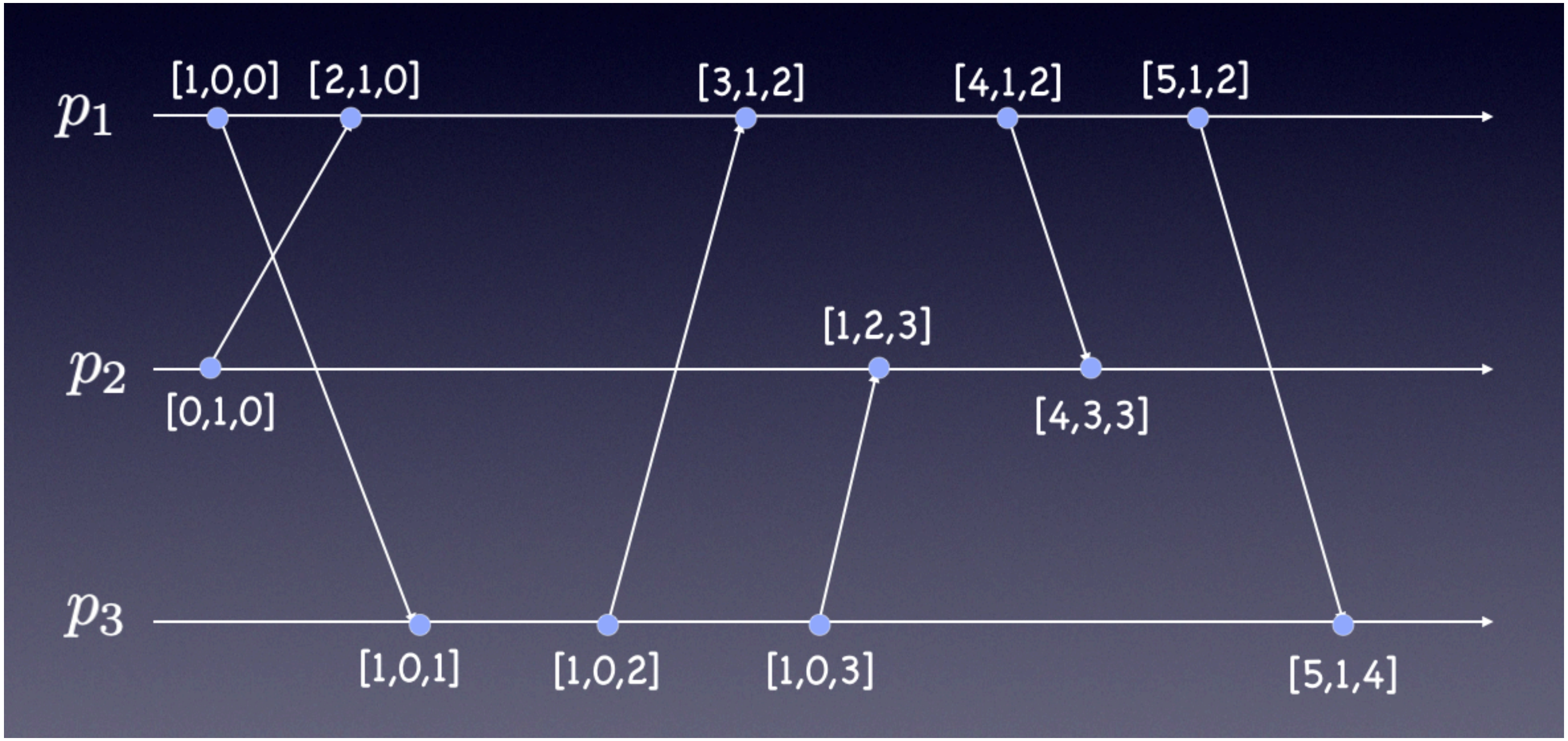
Different Approach

- Monitor process does not query explicitly
- Instead, it passively collects information and uses it to build an observation.
- An observation is an ordering of events of the distributed computation based on the order in which the receiver is notified of the events.

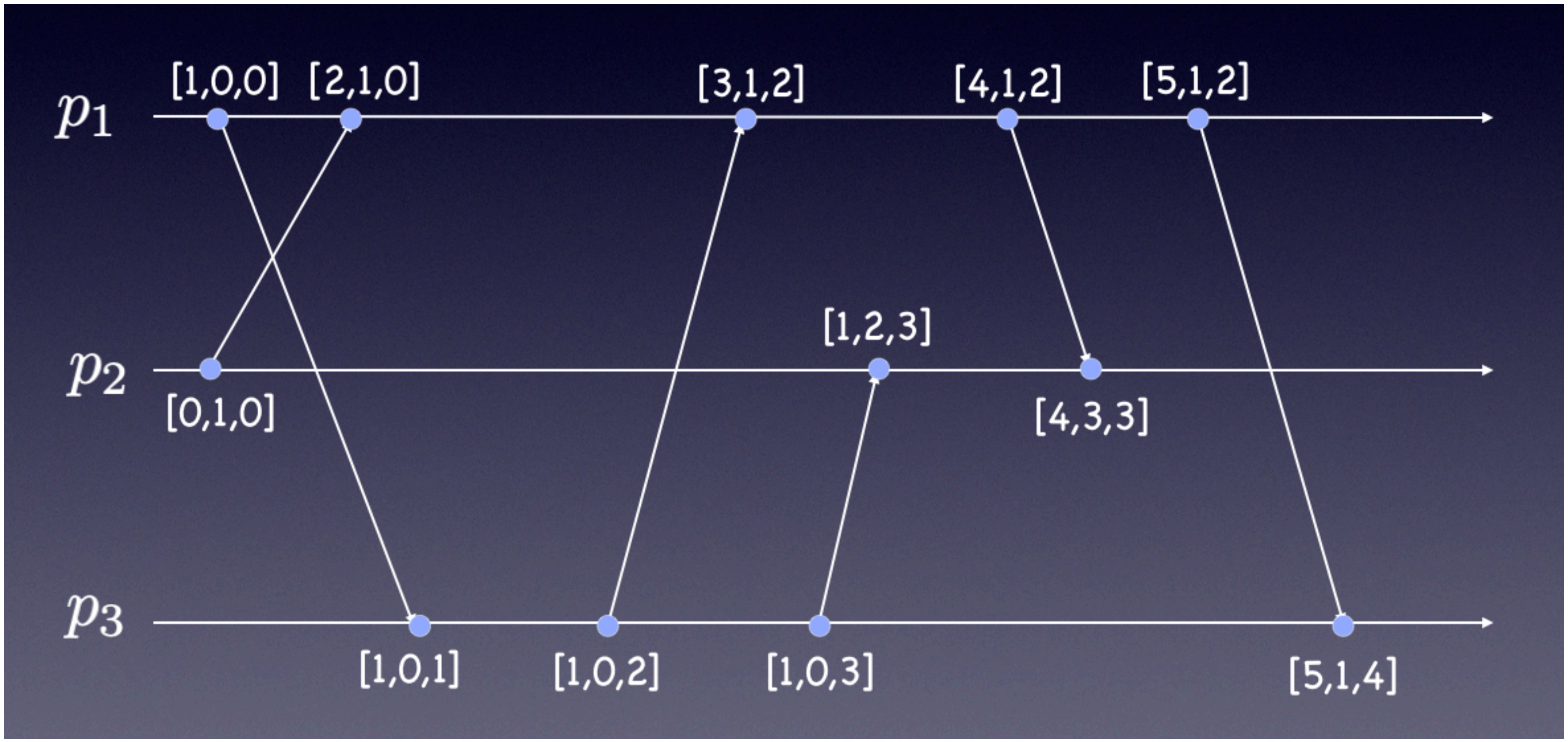
Vector Clocks



Example



Operational Interpretation



Vector clock maintains the count of number of operations performed by each node before a given event

Vector Clock Properties

- Provides strong causal relationships
- Can infer concurrency vs. happened before or after

Snapshot Protocol

- Send to monitor report after every event tagged with its vector clock
- Monitor processes the report respecting the happens-before relationship
  - Completes processing all previous reports before a given report

Summary

- Lamport clocks and vector clocks provide us with good tools to reason about timing of events in a distributed system
- Global snapshot algorithm provides us with an efficient mechanism for obtaining consistent global states