UNIVERSITY of WISCONSIN-MADISON
Computer Sciences Department

CS 739
Distributed Systems

Andrea C. Arpaci-Dusseau

Ordering of Events in Distributed Systems

Two papers:

- Time, Clocks, and the Ordering of Events in a Distributed System
- Distributed Snapshots: Determining Global States of Distributed Systems

Motivation

If want to develop distributed algorithm (and have all participants come to same conclusion), it helps if all see inputs in same order

Questions

- How to know when an event precedes another in a distributed system?
- Sometimes impossible to tell; sometimes it doesn't matter
  - If event A occurs on machine A,
    and event B occurs on machine B,
    but there is no communication between A and B,
    then did event A or event B happen first???

Terminology

Distributed system: A collection of distinct processes which are spatially separated and which communicate with one another by exchanging messages

- How does this differ from our previous definitions?

Process: A sequence of events (instructions, sending messages, receiving messages)

- The events within a process have a total ordering

Partial Ordering

Happened before: ->

Rules

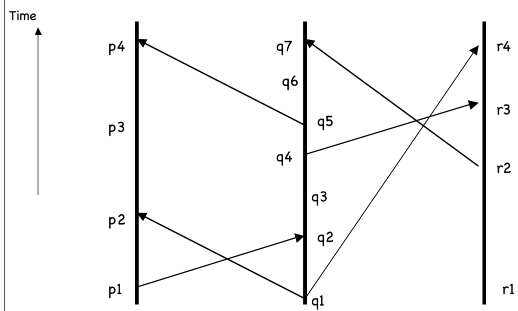
1) if a and b are events in the same process, and a comes before b, then a->b
2) if a is the sending of a message by a process and b is receiving that message, then a->b
3) if a->b and b->c then a->c

a->b: It is possible for a to causally affect b

Concurrent: ↛

- if a ↛ b and b ↛ a, then do not know ordering between a and b
- It is not possible for a to causally affect b

Space-Time Diagram



What is the relationship between (p3,q3)? (p1,q3)? (p2,q3)? (q3,r4)?

Logical Clocks

Abstract view: Logical clock is a way of assigning a number to an event to express ordering

- No relation between logical clock and physical time

Clock Ci for process Pi is a function which assigns a number Ci(a) to any event a in Pi

Clock condition: For any events a, b:

if a->b, then C(a) < C(b)

Converse condition does not hold

- (Can't say concurrent events have same logical time)

Implementation of Logical Clocks

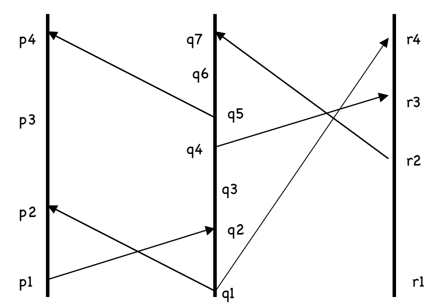
IR1.

- Each process Pi increments Ci between any two successive events

IR2.

- (a) If event a is the sending of message m by process Pi, then m contains a timestamp Tm=Ci(a)
- (b) Upon receiving m, process Pj sets Cj greater than or equal to its presents value and greater than Tm.

Logical Clocks Example



What logical clock values are possible?
Assume initial C(p1)=5, C(q1)=50, C(r1)=2

Total Ordering

Use logical clocks to obtain total ordering across all processes and events

a => b if and only if:

- 1) Ci(a) < Cj(b) OR
- 2) Ci(a) = Cj(b) and Pi < Pj (i.e., use process ids to break ties)

Partial ordering is unique, but total ordering is not!

- Concurrent operations can go in any order
- Depends upon implementation of each Ci()
- Depends upon tie breaking rules

Distributed State Machines

Example: Mutual exclusion

Each process runs same distributed algorithm

Relies upon total ordering of requests

- Agreed upon by all participants
- Can be used to ensure all see events (inputs) in same order and therefore make same decisions

Idea:

- Send timestamped request to all processes
- Handle next request in total order
  - To know next request, must have received request from all possible participants
  - Problems?

Physical Clocks

Motivation: Can observe anomalous behavior if other communication channels exist between processes

- Useful to have physical clock with meaning in physical world

Synchronize independent physical clocks, each running at slightly different rates (skew)

Implementation Idea:

- Send timestamp with each message
- Receiver may update clock to timestamp+minimal network delay
  - Clock must always increase

Lots of work in this area

Distributed Snapshots

Goal

- Want to record global state of distributed system (i.e., state of each process, state of each communication channel)
- Useful so can observe system properties
  - Computation terminated?
  - System deadlocked?
  - Number of tokens?
  - Amount of money?

Complication:

- Distributed system has no shared state nor shared clock
- Cannot record global state simultaneously everywhere

Distributed snapshot: Record local state at different times and combine into meaningful picture

- Obtain cut in logical time, remain consistent by preserving logical ordering (if not ordering in physical time)

System Model

Distributed system: Finite set of processes and channels; described by graph

Processes

- Set of states, initial state, set of events

Channels

- FIFO, error-free, infinite buffers, arbitrary but finite delay

Distributed Snapshot Algorithm

Goal: Record local state (each process plus adjoining channels) that produces a "meaningful" global system state

Idea:

- Send marker along channels to show which messages were sent before snapshot taken
- Receiver records messages in channel before marker

Initial: Some process decides to initiate snapshot (performed periodically)

Marker Rules

Marker-sending rule for p:

- Send marker along each channel (after recording state of p) before sending more messages

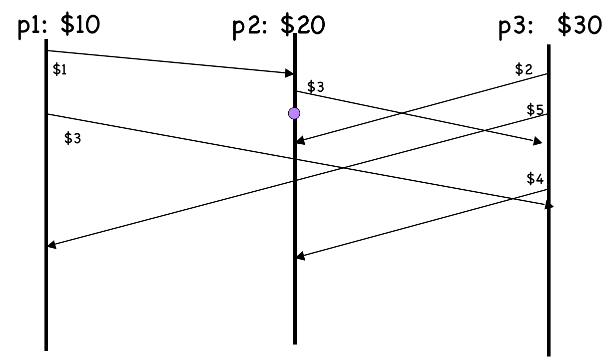
Marking-receiving rule for q on channel c:

- if q has not recorded state yet:
  - record state of q
  - record state of c as empty
- if q has recorded state already:
  - record state of c as the msg sequence that arrived since it recorded its state

Termination

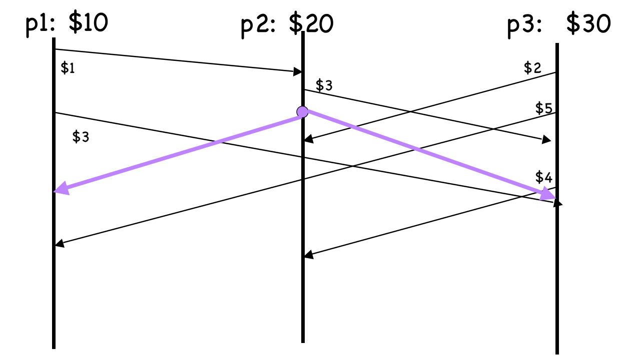
- When state recorded of all processes and all channels
- Must have algorithm to collect and assemble information too

Banking Example



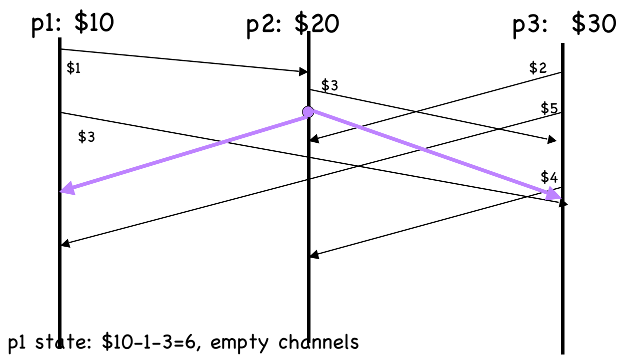
Stable property?

Banking Example



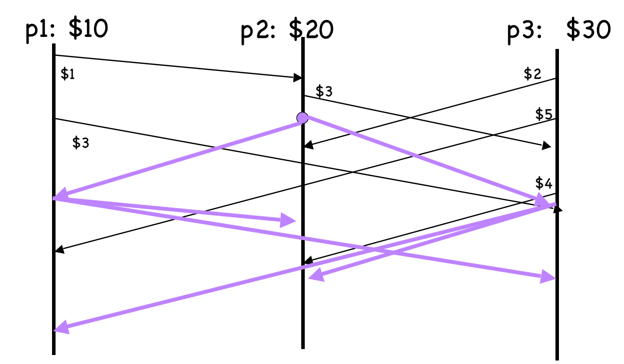
p2 state: $20+1-3 = 18, empty channels

Banking Example

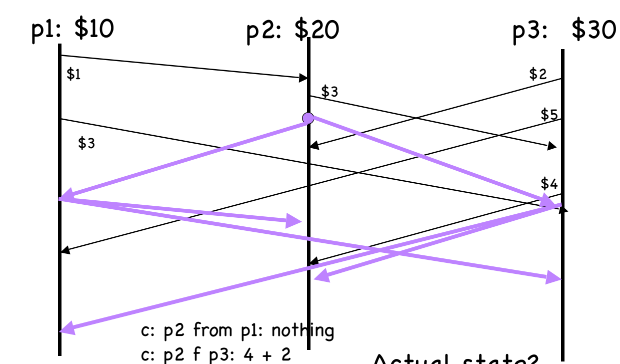


p1 state: $10-1-3=6, empty channels
p3 state: $30-2-5-4+3=22, empty channels
Total money?

Banking Example



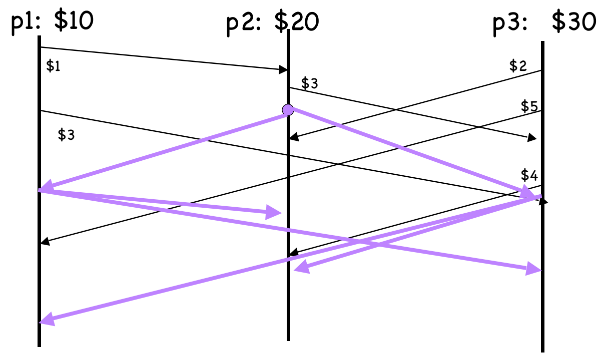
Banking Example



c: p2 from p1: nothing
c: p2 f p3: 4 + 2
c: p3 f p1: 3
c: p1 f p3: 5

Actual state?
p1: 6, p2: 18, p3: 22

Banking Example



c p2 from p3: Never $2 and $4 simultaneously

Properties of Recorded Global State

Recorded global state, S*, may not have occurred

If it bothers you that S* doesn't actually exist...

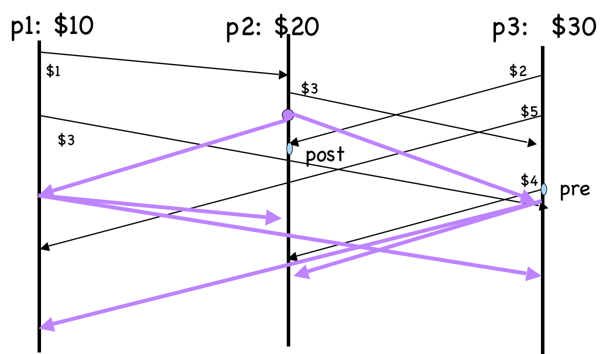
- Given a permutation of the actual sequence of events
- S* is reachable from Sinit
- Sfinal is reachable from S*
- Stable properties will hold in S* as well

How to permute sequence of events?

Goal: Want snapshot to correspond to single logical cut

- Slide events so snapshots taken at same "logical time"
- Some events across processes will switch order with others
  - Specifically, postrecorded events and prerecorded events
  - prerecorded events occurred before state of p was recorded

Banking Example



Example: Need to swap sending $4 from p3 and receiving $2
Still logically consistent; could not observe difference