Distributed Systems

Global states and snapshots

Rik Sarkar
Edinburgh Fall 2014

University of Edinburgh

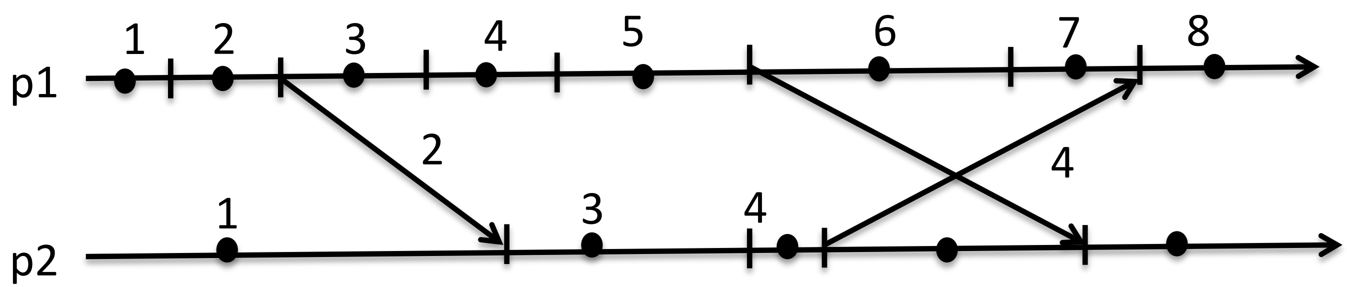
Distributed snapshots

- Take a "snapshot" of a system
- E.g. for backup: If system fails, it can start up from a meaningful state

- Problem:
  - Imagine a sky filled with birds. The sky is too large to cover in a single picture.
  - We want to take multiple pictures that are consistent in a suitable sense
    - Eg. We can correctly count the number of birds from the snapshot

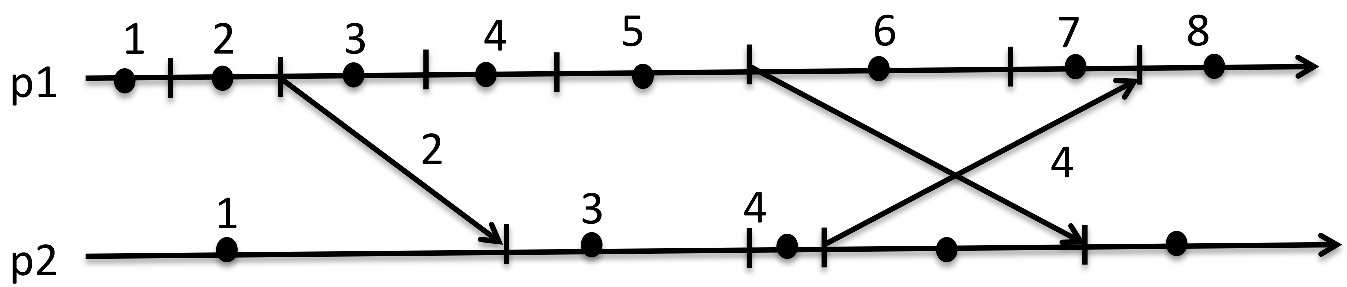
Events and states

- Every process goes through alternate sequence of states and events
- It is enough to count the states for correct clock sequence



Events and states

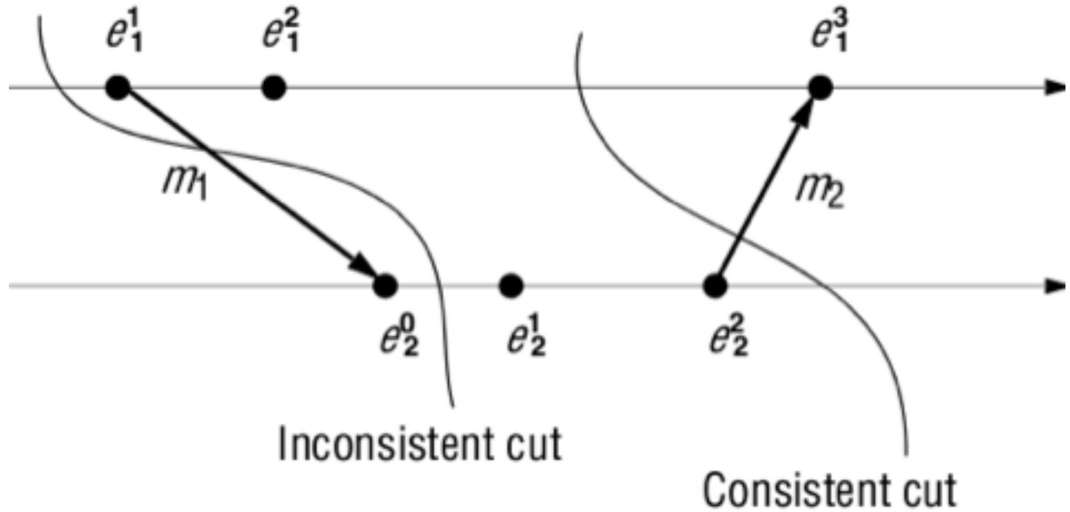
- Happened before and concurrent relations for states are defined similarly



Distributed snapshots

- Global state:
  - State of all processes
  - And state of all communication channels
    - What message it is carrying
- Consistent cuts:
  - A set of states of all processes is a consistent cut if:
  - For any states s, t in the cut, s||t
- If a⟶b, then the following is not allowed:
  - b is before the cut, a is after the cut

Consistent cut



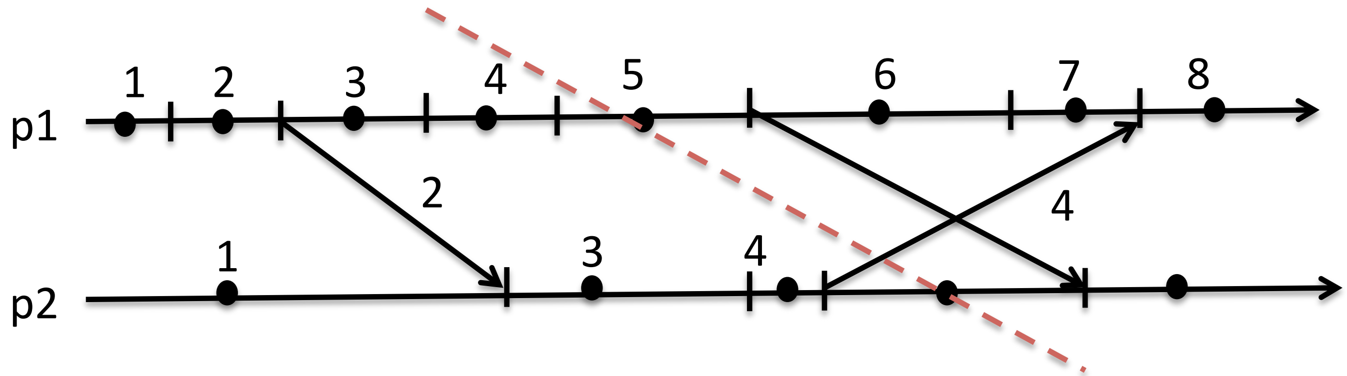
Distributed snapshot algorithm

- Find a set of states: one for each process
  - Ask each process to record its state
- The set of states must be a consistent cut

- Assumptions:
  - Communication channels are FIFO
  - Processes communicate only with neighbors
  - (We assume for now that everyone is neighbor of everyone)
  - Processes do not fail

Global snapshot: Chandy and Lamport algorithm

- One process initiates snapshot and sends a marker
- Marker is the boundary between “before” and “after” the snapshot



Global snapshot: Chandy and Lamport algorithm

- Marker send rule (Process i)
  - Process i records its state
  - On every outgoing channel where a marker has not been sent:
    - i sends a marker on the channel
    - before sending any other message
- Marker receive rule (Process j receives marker on channel C)
  - If j has not received the marker before
    - Record state of j
    - Record state of C as empty
    - Follow marker send rule
  - Else:
    - Record the state of C as the set of messages received on C since recording j's state and before receiving marker on C
- Algorithm stops when all processes have received marker on all incoming channels

Complexity

- Message?

Property

- If s1 (in p1) $\longrightarrow$ s2 (in p2)
  - Then s2 is before the cut $\Longrightarrow$ s1 is before the cut
  - Suppose not & s1 is after the cut.
    - Then p1 recorded its state before s1
    - Consider the message m from p1 to p2
      - This causes the relation s1$\longrightarrow$s2 to be true
    - p1 must have recorded its state before sending m
    - p1 must have sent marker to p2 before sending m
      - By marker sending rule
    - p2 must have received marker before m and before s2
    - s2 must be after the cut – contradiction.

Application of snapshots: Detection of stable predicates

- Stable predicate:
  - A property that once it becomes true, stays true (until detection and intervention)
  - Eg:
    - Deadlocked : every process in some subset is waiting for another
    - Terminated : once ended, computation remains stopped
    - Loss of token : in mutual exclusion, process with token can access a resource. If token gets lost due to failure, it stays lost.
    - Garbage : If no-one has a reference to a file, that file can be deleted
  - So, if such a property was true before the snapshot, it is true in the snapshot, and can be detected by checking the snapshot

Where snapshots are not useful: non-stable predicates

- E.g.
  - Was this file opened at some time?
  - Was $x1 - x2 < \delta$ ever?
  - Non-stable predicates may have happened, but then system state changes..

Types of non-stable predicates

- Possibly B:
  - B could have happened
- Definitely B:
  - B definitely happened
- How can we check for definitely B and possibly B?

Collecting global states

- Each process notes its every state & vector timestamp
  - Sends it to a server for recording
  - Note: we do not need to save every time a state changes: only when it affects the predicates to be checked
    - Assuming we know what predicates will be checked
- The server looks at these and tries to figure out if predicate B was possibly or definitely true