

Distributed Systems: Global Properties

Alistair Rendell

©2009 School Computer Science, Australian National University

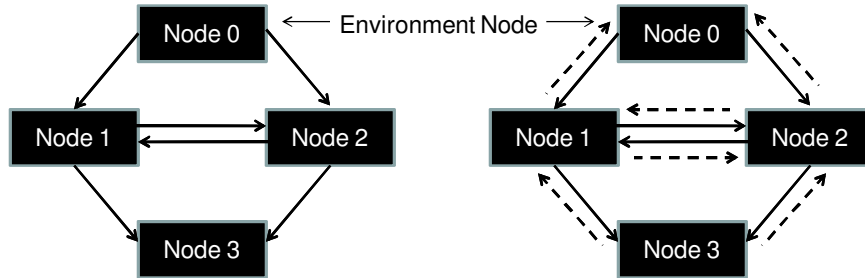
Distributed Termination

- A concurrent program terminates when all of its processes have terminated
 - a distributed program terminates when all of its processes at all of its nodes have terminated
- We consider the Dijkstra-Scholten algorithm

©2009 School Computer Science, Australian National University

Dijkstra-Scholten Algorithm

- Each message is associated with a return signal that is eventually sent back to source node



$inDeficit_i[E]$: Difference between number of messages received by node i on edge E and signals sent back
 $outDeficit_i$: Difference between number of messages sent out by node i and signals received back
 Termination when no more messages sent

©2009 School Computer Science, Australian National University

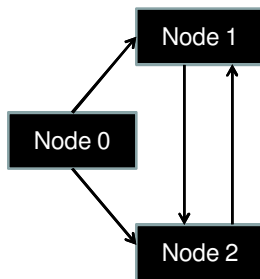
Preliminary Algorithm

- Send message:
 - increment $outDeficit$
- Receive message:
 - increment $inDeficitTot$ on that edge and total $inDeficit$
- Send signal:
 - if $inDeficitTot > 1$ find some edge with non-zero $inDeficit$ and send signal along that edge
 - if $inDeficitTot = 1$ AND is terminated AND $outDeficit = 0$ send final signal
- Receive signal:
 - decrement $outDeficit$

©2009 School Computer Science, Australian National University

Analysis

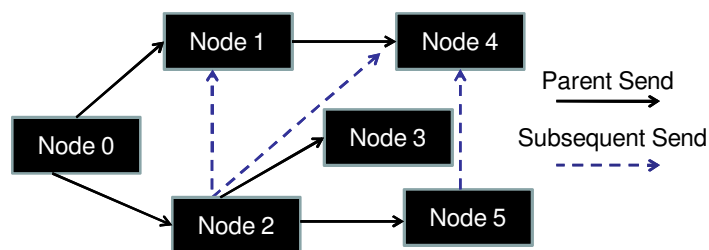
- If the computation terminates at all nodes, eventually the environment node announces termination
 - exercise for reader (or read Ben-Ari)
- But algorithm is not safe



- Node 0 sends messages to both Nodes 1 and 2
- Nodes 1 and 2 exchange messages
- $IndeficitTot = 2$ at both Nodes 1 and 2
- Both Nodes 1 and 2 send signals to Node 0
- Node 0 has $outDeficit=0$ and announces termination, although Nodes 1 and 2 have not terminated

©2009 School Computer Science, Australian National University

Dijkstra-Scholten Algorithm



- Based on spanning tree
 - each node included in tree
 - each has a unique parent
- As before, but always reserve the last send signal for the parent node
 - parent is the node that sends the first message to a node
 - need to add another variable to store this information

©2009 School Computer Science, Australian National University

Performance

- Number of signals equals number of messages sent
 - this can be large if system up for long time
 - could overflow integer values
- Various approaches exist to address this in part

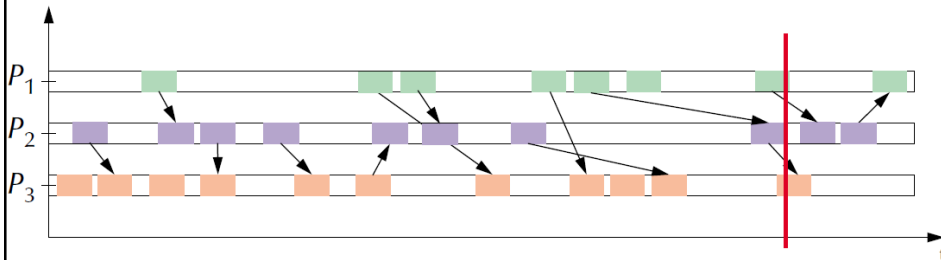
©2009 School Computer Science, Australian National University

Saving State in Distributed Systems

©2009 School Computer Science, Australian National University

Collecting States

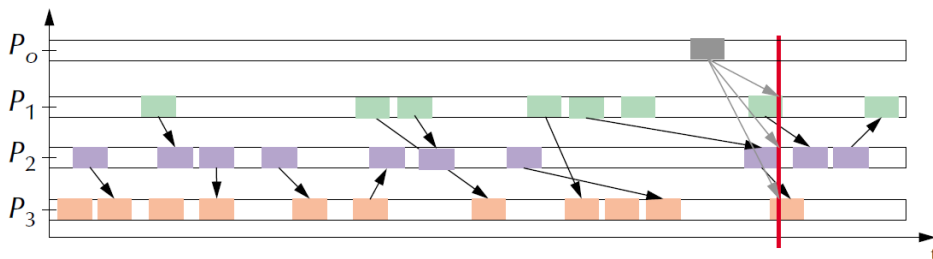
- Collect state at given point in time
 - useful for debugging, performance analysis, termination and deadlock detection, collect a consistent view of the system etc



©2009 School Computer Science, Australian National University

Use of Coordinator Process

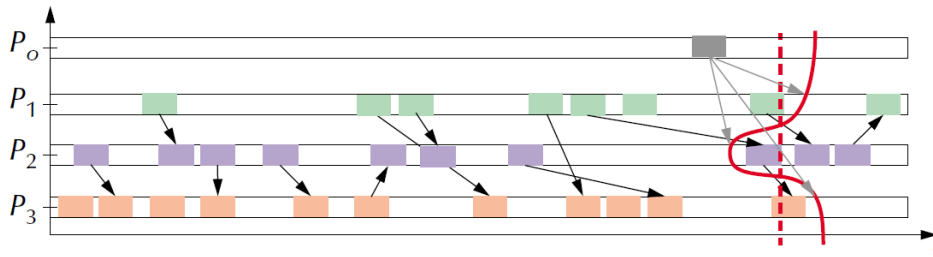
- Introduce snapshot process that contacts all processes requesting they save state



©2009 School Computer Science, Australian National University

Time Delays

- Inevitable time delay in snapshot messages from P_0 leads

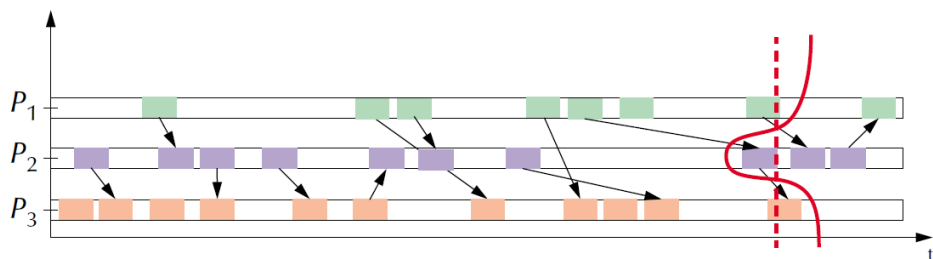


– making collection of local states at an absolute global time impossible

©2009 School Computer Science, Australian National University

Consistent Global Snapshot

- Clearly define events as being before or after the snapshot



©2009 School Computer Science, Australian National University

Consistent Global State

Divide events in two

- Before the snapshot (belonging to the past, P)

$$(e_2 \in P) \wedge (e_1 \rightarrow e_2) \Rightarrow e_1 \in P$$

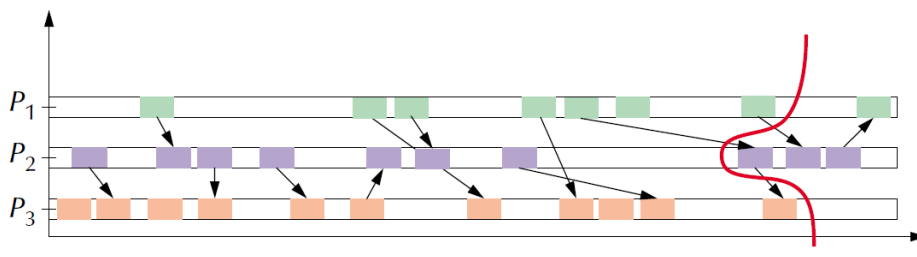
- After the snapshot (belonging to the future, F)

$$(e_1 \in F) \wedge (e_1 \rightarrow e_2) \Rightarrow e_2 \in F$$

©2009 School Computer Science, Australian National University

Check for Consistency

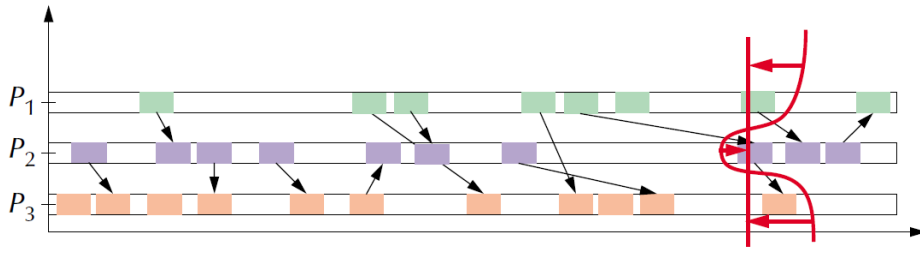
- Straighten out snapshot cut



©2009 School Computer Science, Australian National University

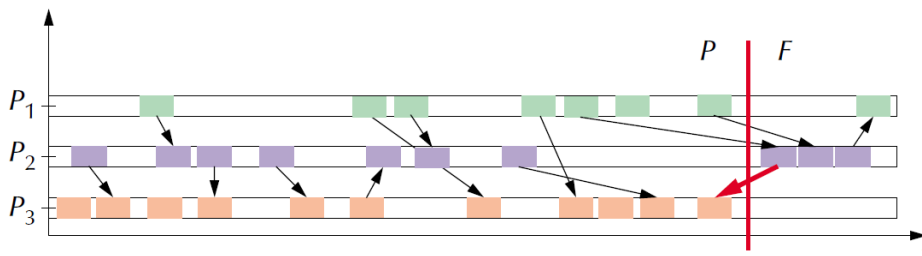
Check for Consistency

- Straighten out snapshot and check for inconsistencies



©2009 School Computer Science, Australian National University

Inconsistent Snapshot



- $(e_1 \in F) \wedge (e_1 \rightarrow e_2) \Rightarrow e_2 \in F$, but in above $e_2 \in P$
– thus inconsistent snapshot

©2009 School Computer Science, Australian National University

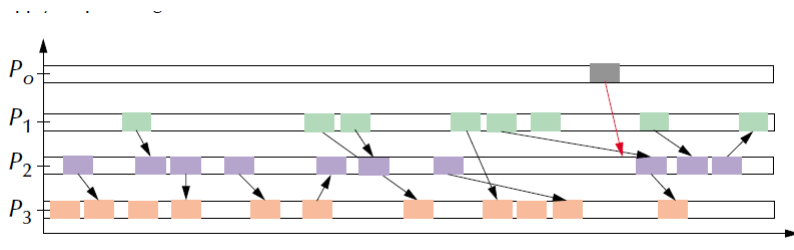
Snapshot Algorithm

- Observer process, P_0 , (any process) creates a snapshot token t_s and saves its local state s_0
- P_0 sends t_s to all other processes
- $\forall P_i$ which receives the t_s (as a token, or as part of another message – see below):
 - saves local state s_i and sends s_i to P_0
 - attaches t_s to all further messages which are sent to other processes
 - saves t_s and ignores all further incoming t_s tokens
- $\forall P_i$ which previously received t_s and receive a message m without t_s :
 - forward m to P_0 (this message belongs to the snapshot, i.e. the past)

©2009 School Computer Science, Australian National University

Applying Snapshot #1

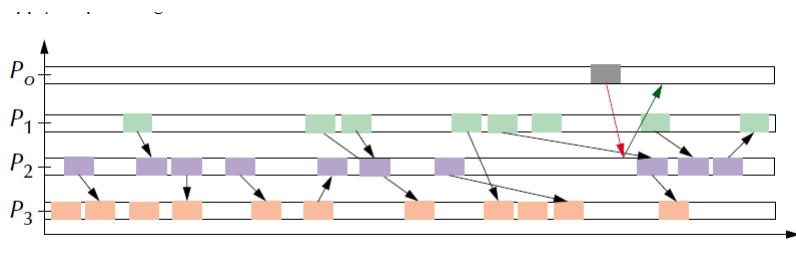
- P_0 sends out snapshot token to all



©2009 School Computer Science, Australian National University

Applying Snapshot #2

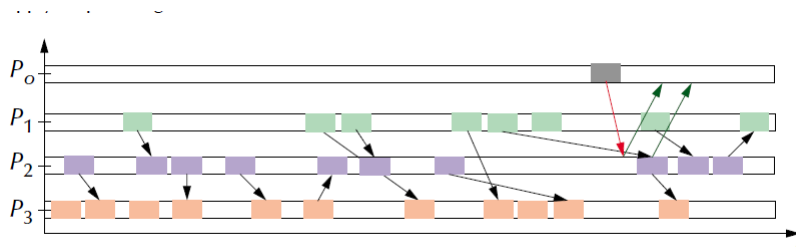
- P_2 responds with local state



©2009 School Computer Science, Australian National University

Applying Snapshot #3

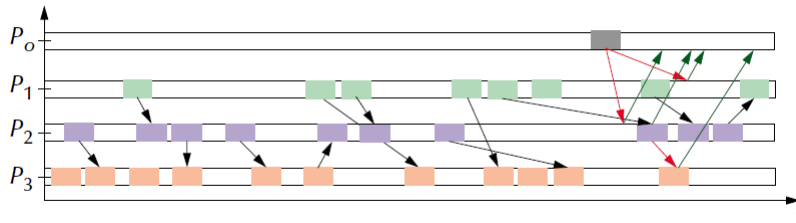
- P_2 forwards an untagged message



©2009 School Computer Science, Australian National University

Applying Snapshot #4

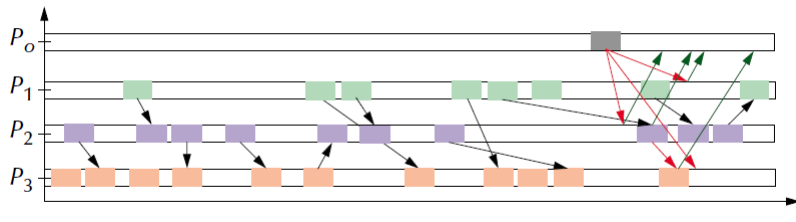
- P_1 responds with local state
- P_3 responds with its local state (due to a tagged message)



©2009 School Computer Science, Australian National University

Applying Snapshot #5

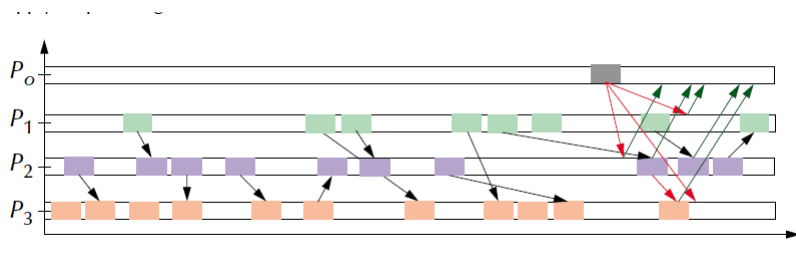
- P_3 ignores the snapshot token (as previously received as part of a message)



©2009 School Computer Science, Australian National University

Applying Snapshot #6

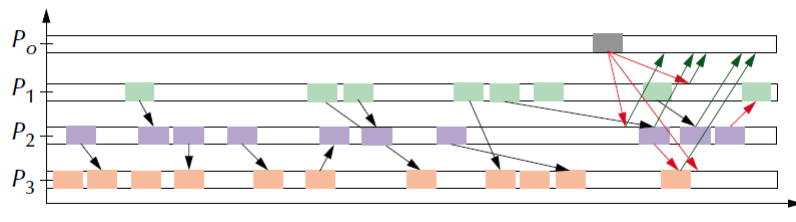
- P_2 forwards and untagged message



©2009 School Computer Science, Australian National University

Applying Snapshot #7

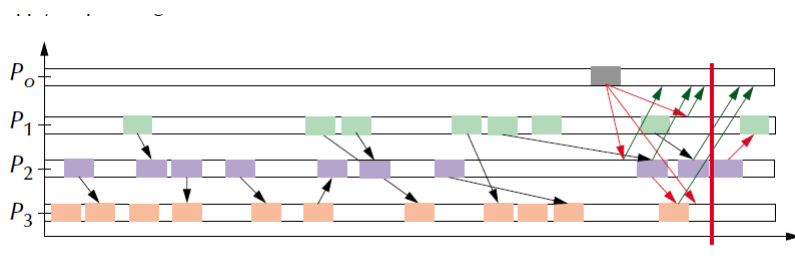
- P_1 ignores a tagged message (token was previously received, local state is already reported)



©2009 School Computer Science, Australian National University

Final Observed Snapshot

- The final snapshot as known to the observer P_0 after receiving all reports



©2009 School Computer Science, Australian National University

Snapshot Termination

- Make assumptions about delays in the system
- or
- Employ Dijkstra-Scholten algorithm to count the send and received messages for each process (include this in its local state) and keep track of outstanding messages in the observer process
- or
- combination of above

©2009 School Computer Science, Australian National University