CS514: Intermediate Course in Operating Systems

Professor Ken Birman
Vivek Vishnumurthy: TA

Recall our discussion of time

- Logical clocks: represent part of $\rightarrow$ relation, small overhead
- Vector clocks: accurately represent $\rightarrow$ but more costly
- Wall clocks: tradeoff between precision and accuracy.
  - Rarely precise enough for use in protocols
  - Hence often view time as an “add on”

Today: “Simultaneous” actions

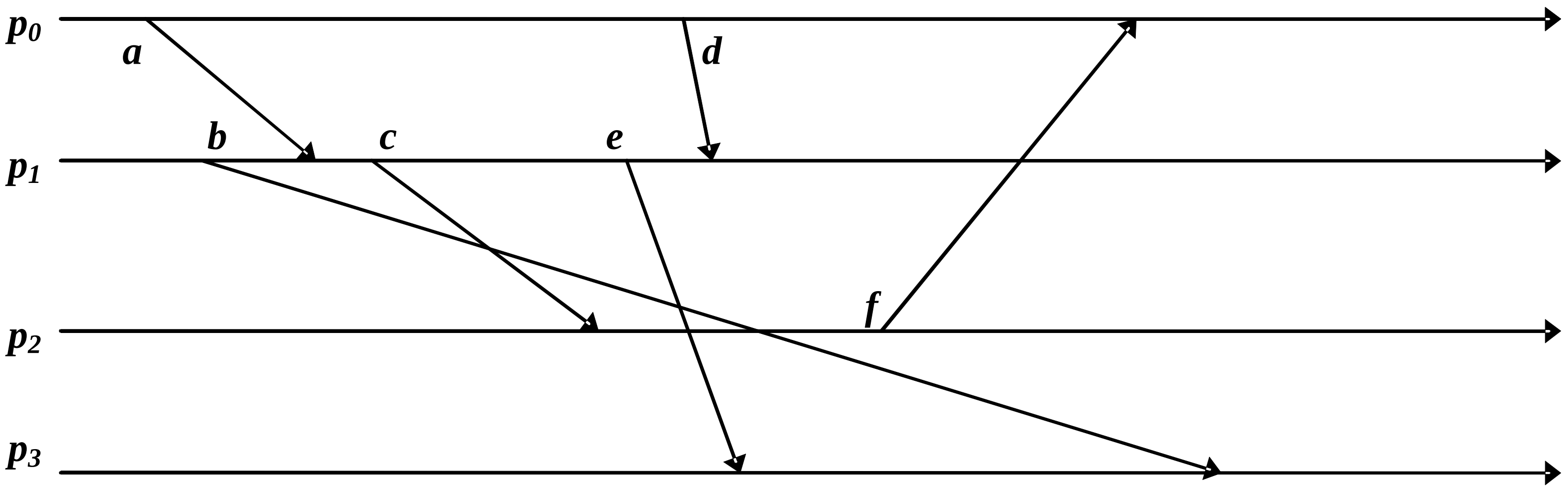
- There are many situations in which we want to talk about some form of simultaneous event
  - Our missile interceptor is one case
  - But think about updating replicated data
    - Perhaps we have multiple conflicting updates
    - The need is to ensure that they will happen in the same order at all copies
    - This “looks” like a kind of simultaneous action

Temporal distortions

- Things can be complicated because we can't predict
  - Message delays (they vary constantly)
  - Execution speeds (often a process shares a machine with many other tasks)
  - Timing of external events
- Lamport looked at this question too

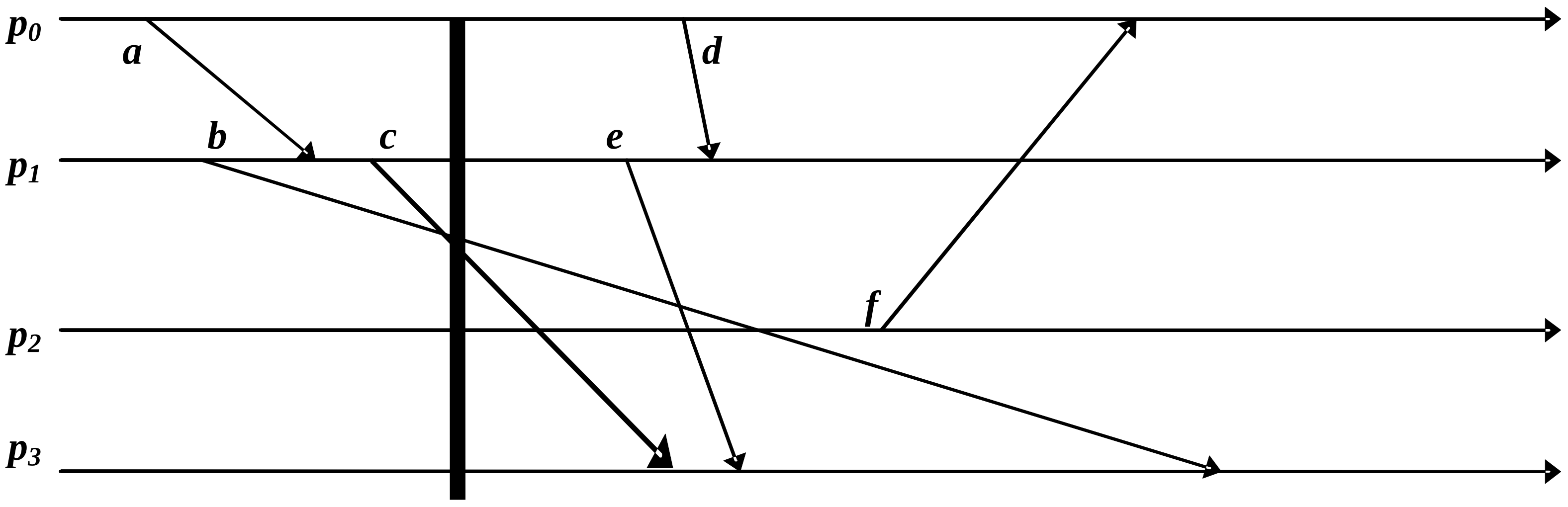
Temporal distortions

- What does "now" mean?



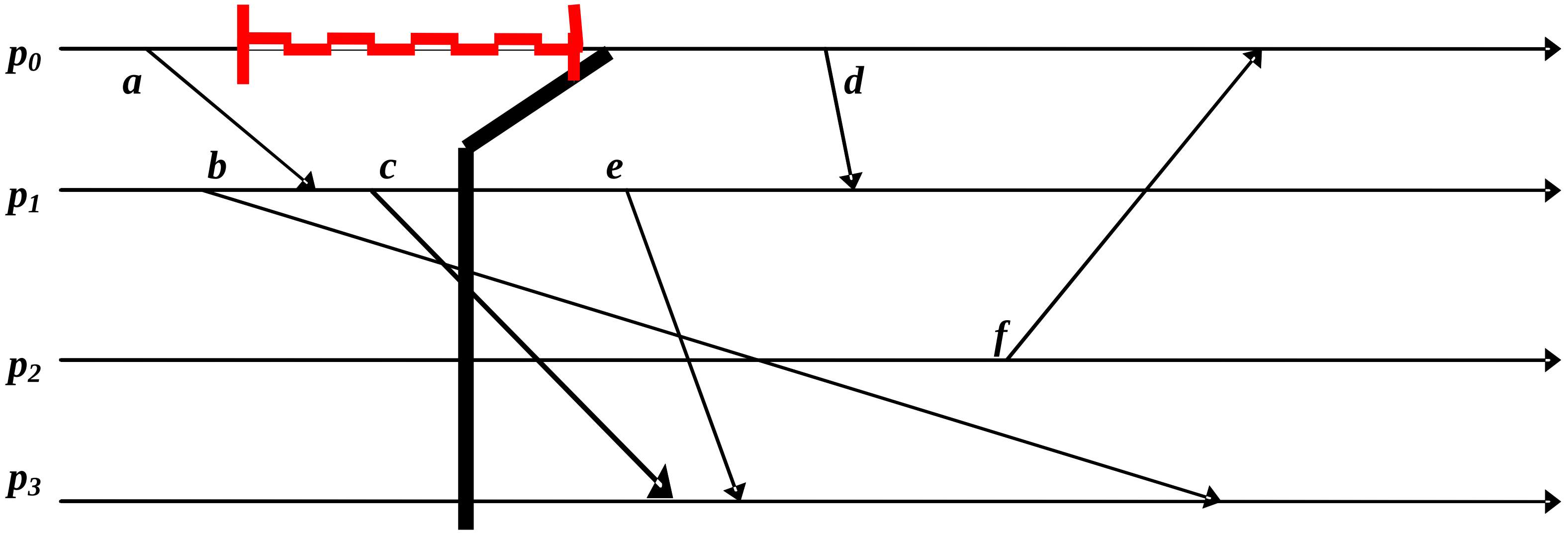
Temporal distortions

- What does “now” mean?



Temporal distortions

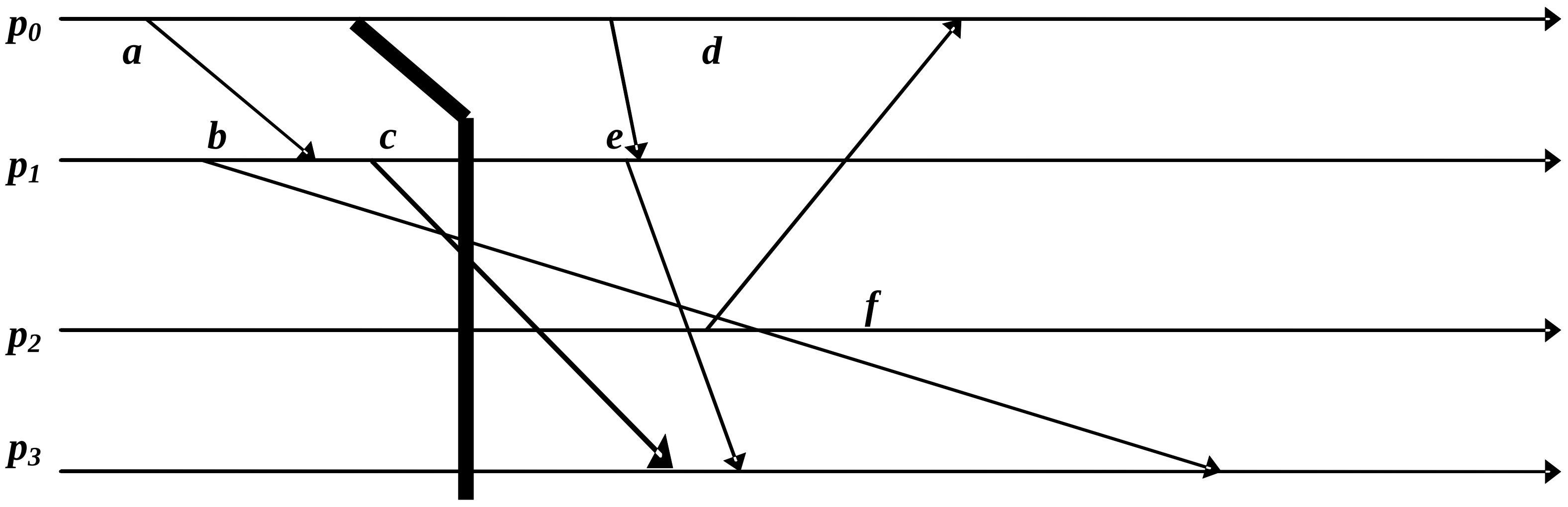
- Timelines can “stretch”...



- ... caused by scheduling effects, message delays, message loss...

Temporal distortions

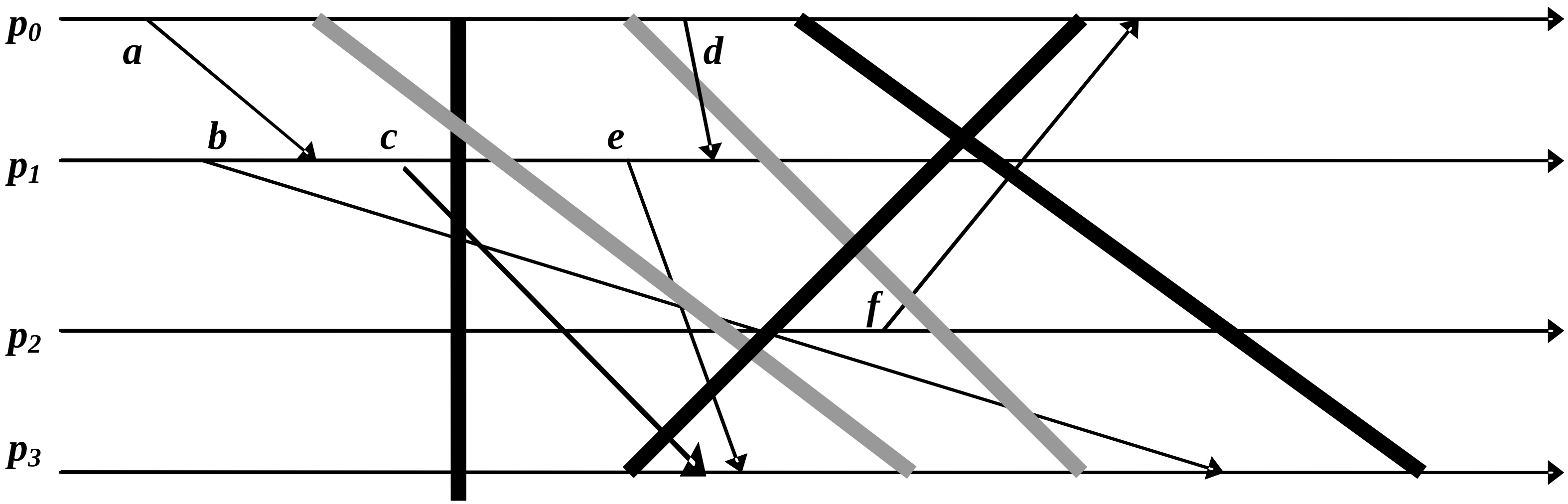
- Timelines can "shrink"



- E.g. something lets a machine speed up

Temporal distortions

- Cuts represent instants of time.



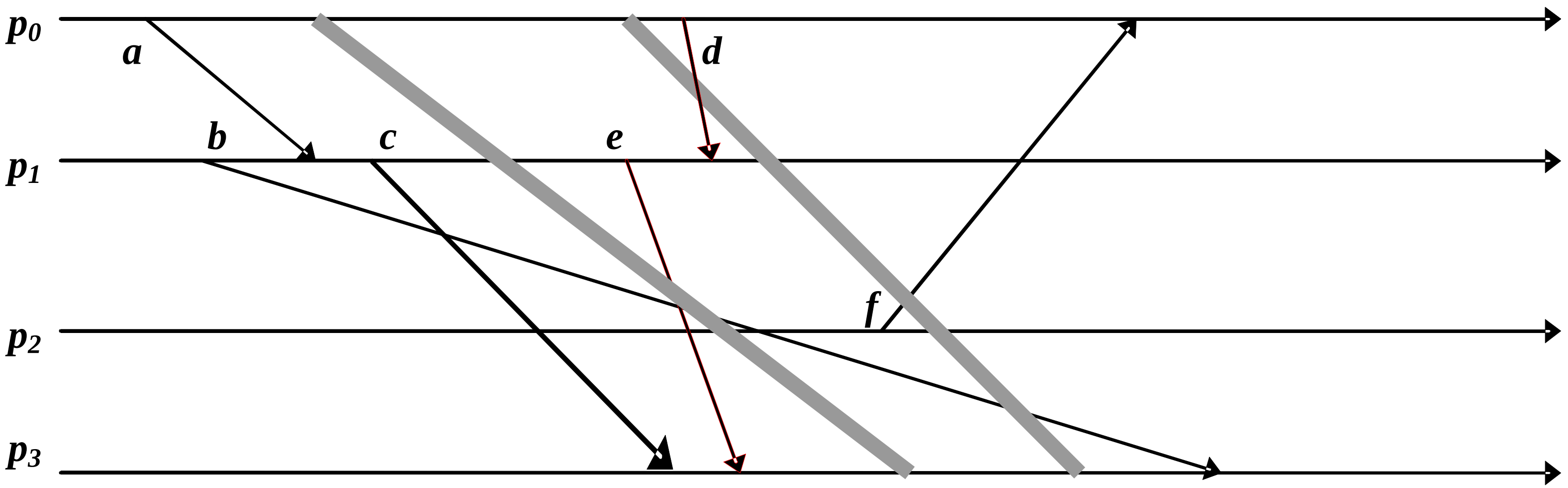
- But not every "cut" makes sense
  - Black cuts could occur but not gray ones.

Consistent cuts and snapshots

- Idea is to identify system states that "might" have occurred in real-life
  - Need to avoid capturing states in which a message is received but nobody is shown as having sent it
  - This the problem with the gray cuts

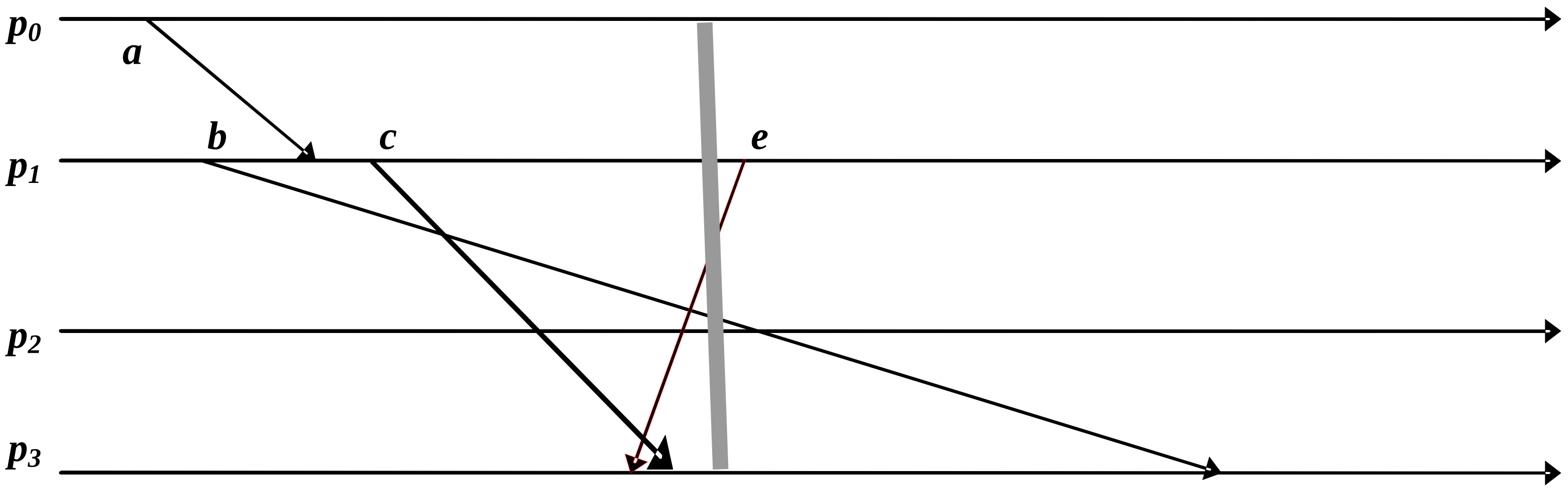
Temporal distortions

- Red messages cross gray cuts “backwards”



Temporal distortions

- Red messages cross gray cuts “backwards”



- In a nutshell: the cut includes a message that “was never sent”

Who cares?

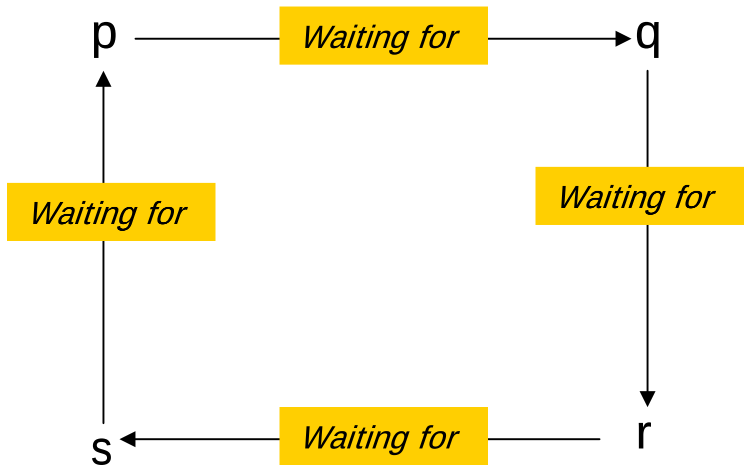
- Suppose, for example, that we want to do distributed deadlock detection
  - System lets processes “wait” for actions by other processes
  - A process can only do one thing at a time
  - A deadlock occurs if there is a circular wait

Deadlock detection "algorithm"

- p worries: perhaps we have a deadlock
- p is waiting for q, so sends "what's your state?"
- q, on receipt, is waiting for r, so sends the same question... and r for s.... And s is waiting on p.

Suppose we detect this state

- We see a cycle...



- ... but is it a deadlock?

Phantom deadlocks!

- Suppose system has a *very high rate* of locking.
- Then perhaps a lock release message "passed" a query message
  - i.e. we see "q waiting for r" and "r waiting for s" but in fact, by the time we checked r, q was no longer waiting!
- In effect: we checked for deadlock on a gray cut – an inconsistent cut.

Consistent cuts and snapshots

- Goal is to draw a line across the system state such that
  - Every message “received” by a process is shown as having been sent by some other process
  - Some pending messages might still be in communication channels
- A “cut” is the frontier of a “snapshot”

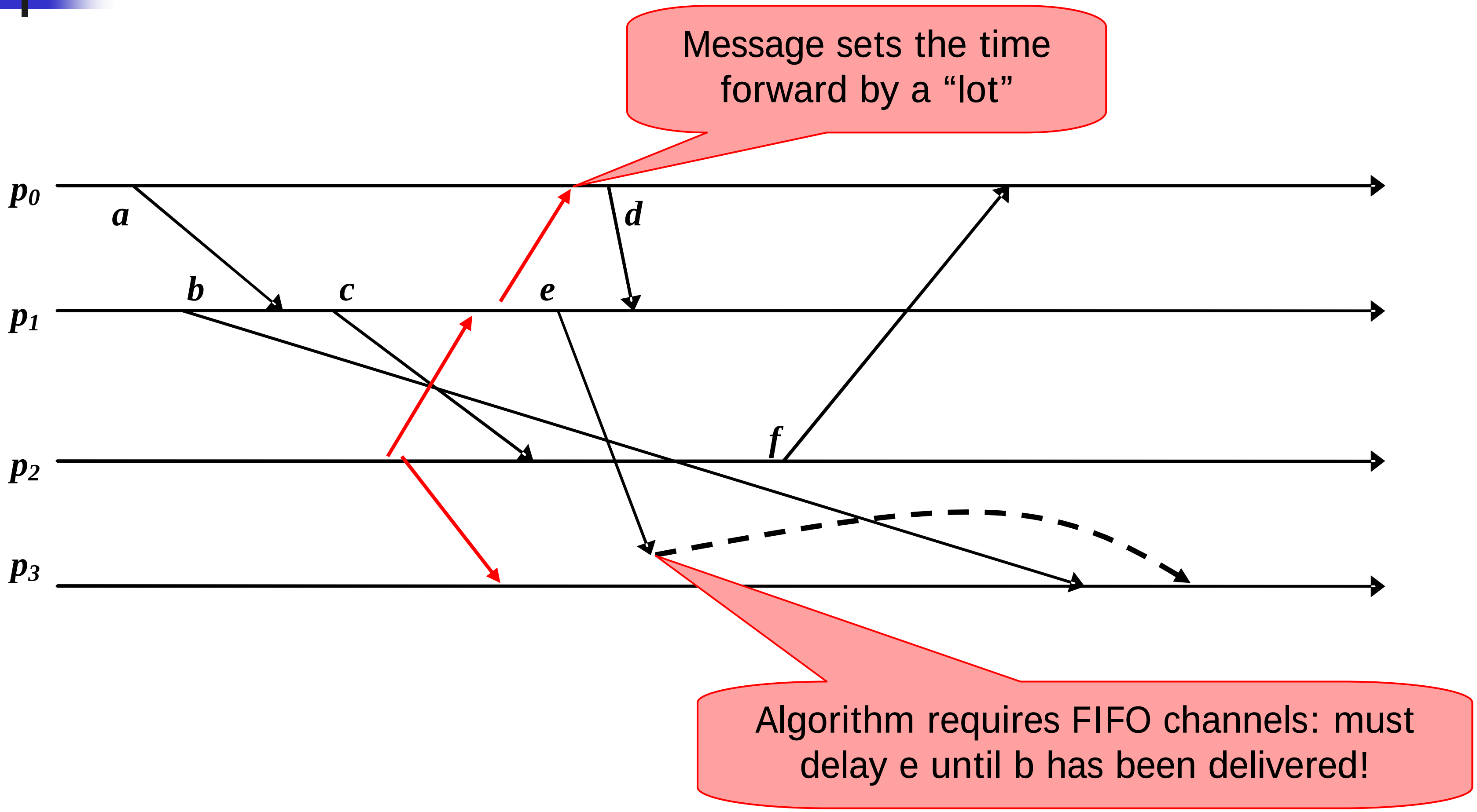
Estudar

- Chandy, K. M., and L. Lamport, “Distributed Snapshots: Determining States of Distributed Systems”, ACM Transactions On Computer Systems:3:1 (February 1985): 63-75
- Ou Cap. 11 Coulouris (Seção 11.5.3)

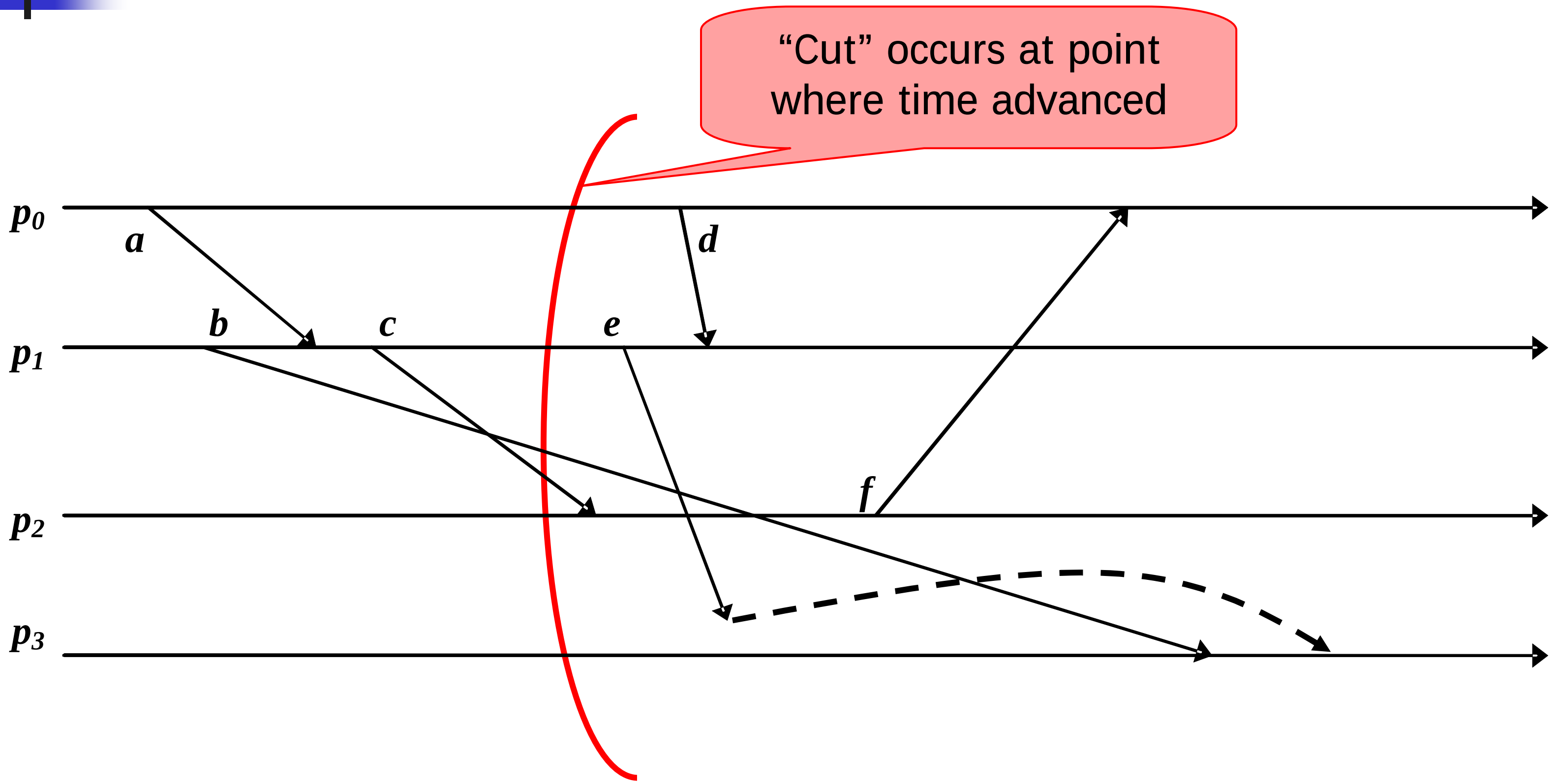
Chandy/Lamport Algorithm

- Assume that if p_i can talk to p_j they do so using a lossless, FIFO connection
- Now think about logical clocks
  - Suppose someone sets his clock way ahead and triggers a “flood” of messages
  - As these reach each process, it advances its own time... eventually all do so.
- The point where time jumps forward is a consistent cut across the system

Using logical clocks to make cuts



Using logical clocks to make cuts



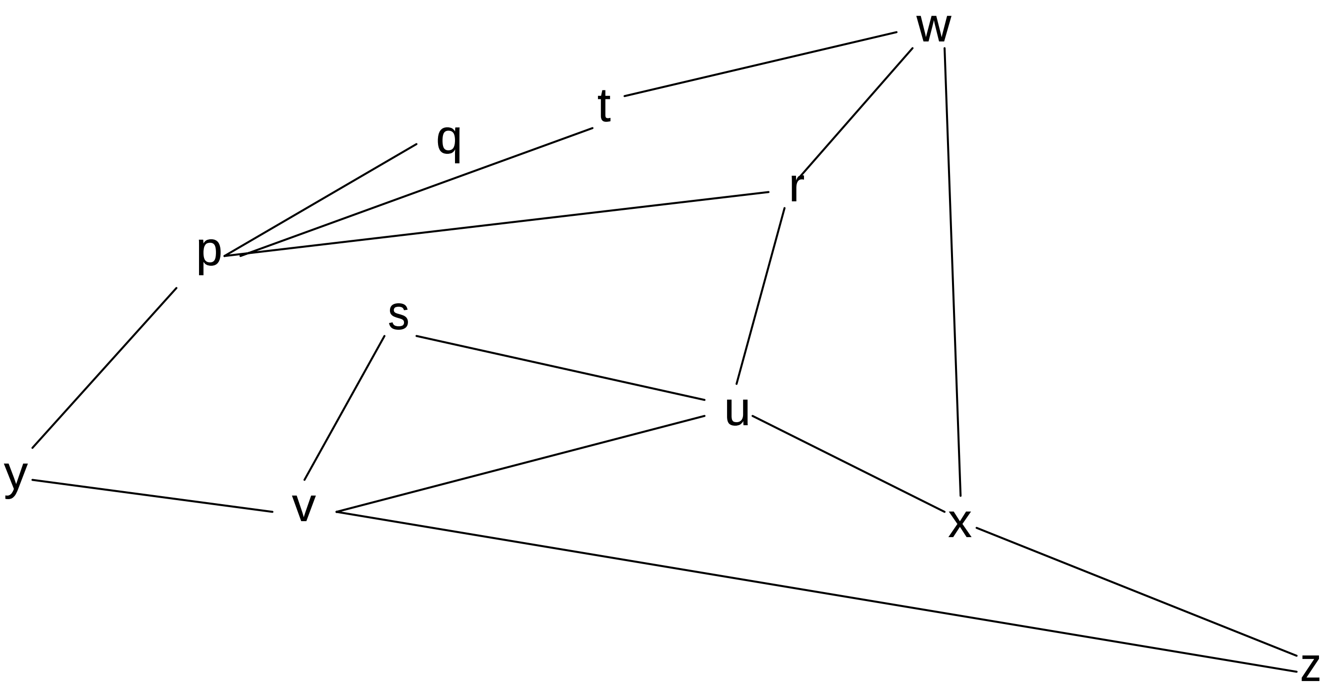
Turn idea into an algorithm

- To start a new snapshot, p_i …
  - Builds a message: “P_i is initiating snapshot k”.
    - The tuple (p_i, k) uniquely identifies the snapshot
- In general, on first learning about snapshot (p_i, k), p_x
  - Writes down its state: p_x’s contribution to the snapshot
  - Starts “tape recorders” for all communication channels
  - Forwards the message on all outgoing channels
  - Stops “tape recorder” for a channel when a snapshot message for (p_i, k) is received on it
- Snapshot consists of all the local state contributions and all the tape-recordings for the channels

Chandy/Lamport

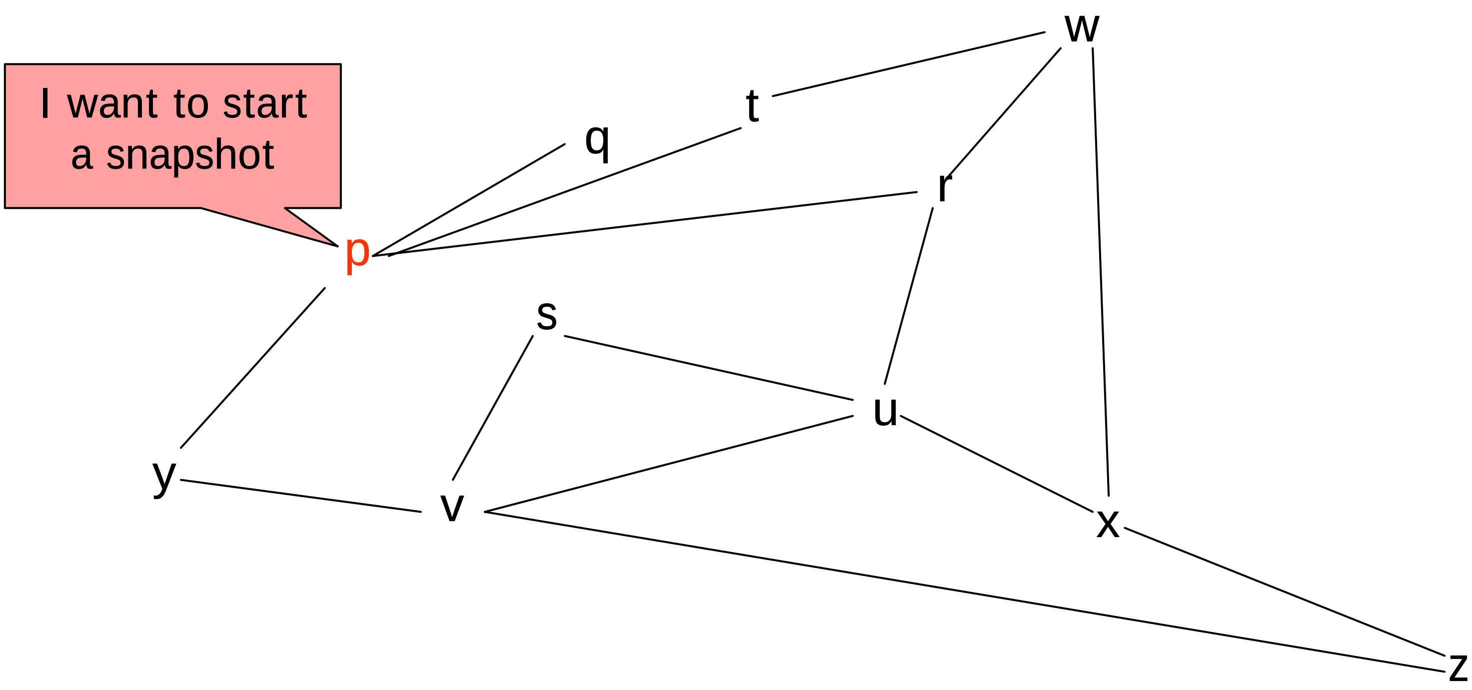
- This algorithm, but implemented with an outgoing flood, followed by an incoming wave of snapshot contributions
- Snapshot ends up accumulating at the initiator, p_i
- Algorithm doesn’t tolerate process failures or message failures.

Chandy/Lamport



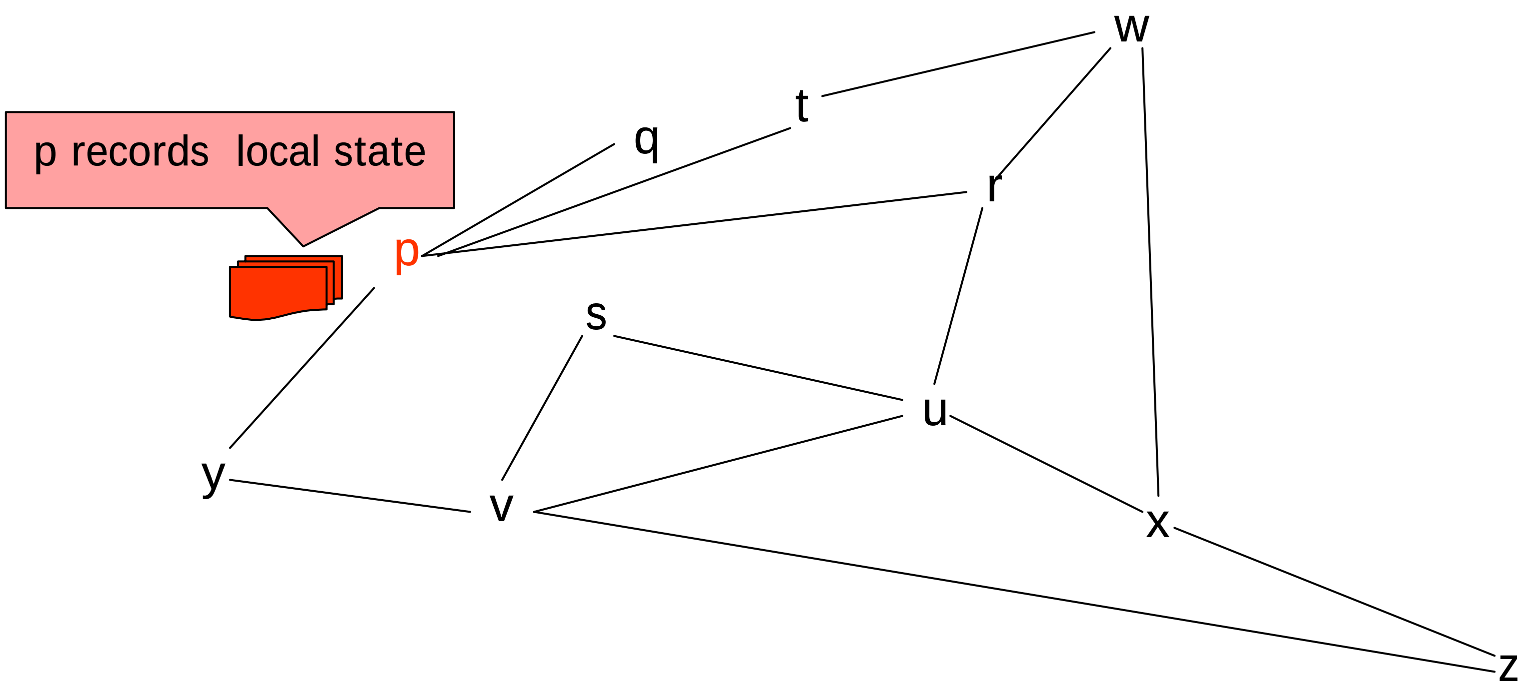
A network

Chandy/Lamport



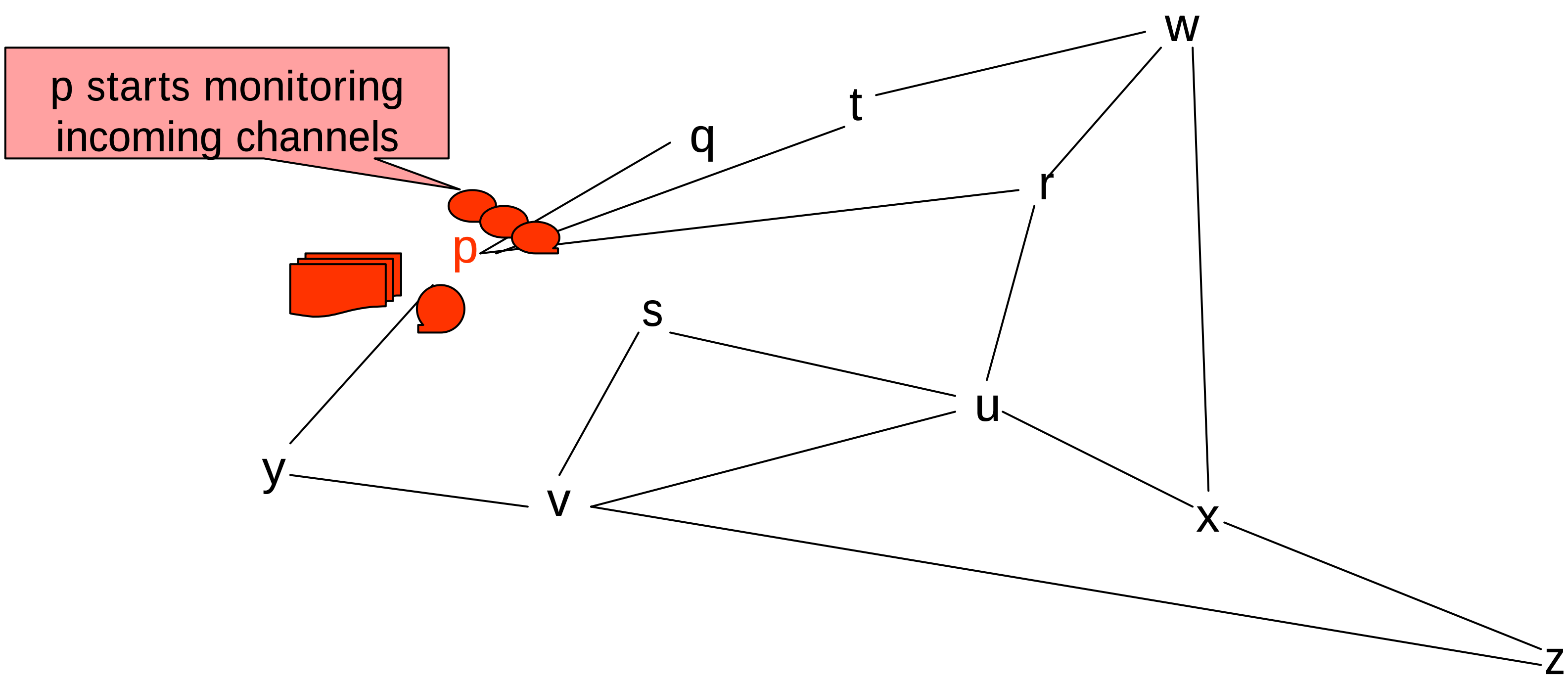
A network

Chandy/Lamport



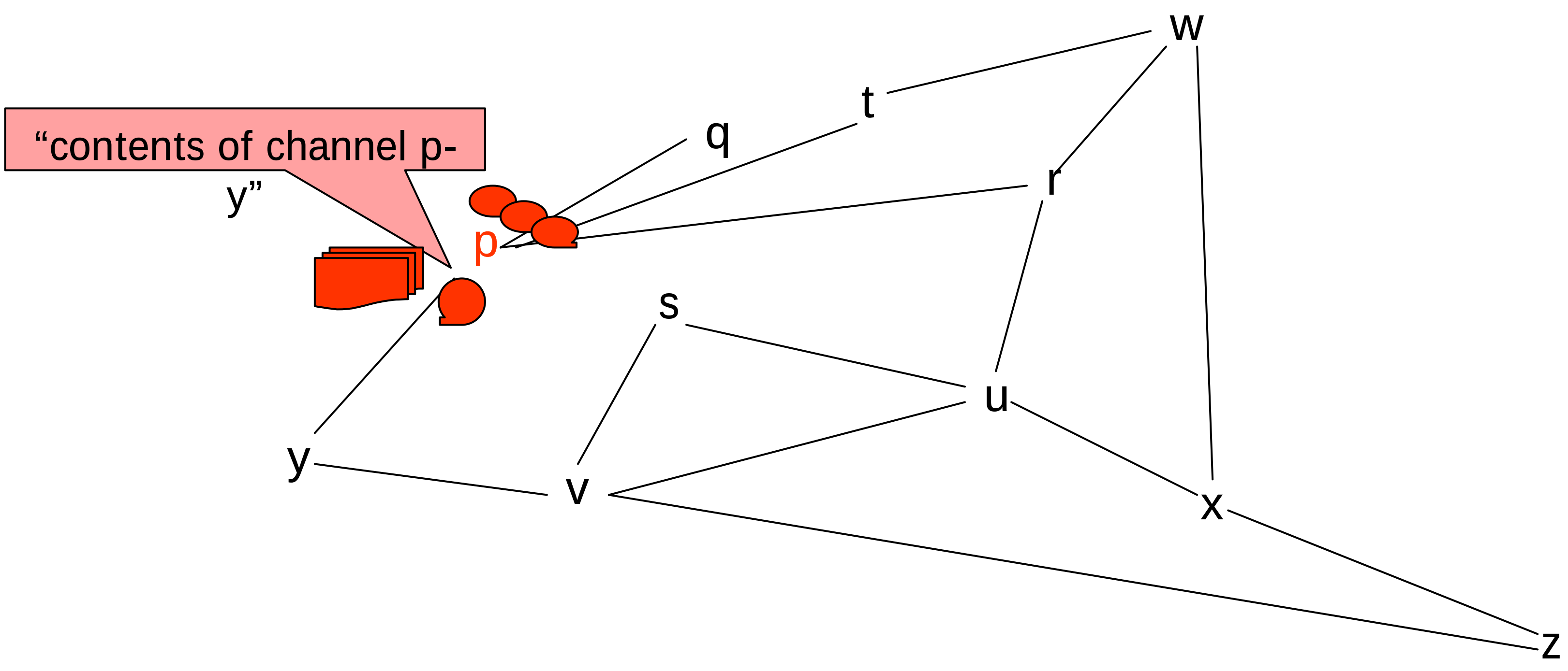
A network

Chandy/Lamport



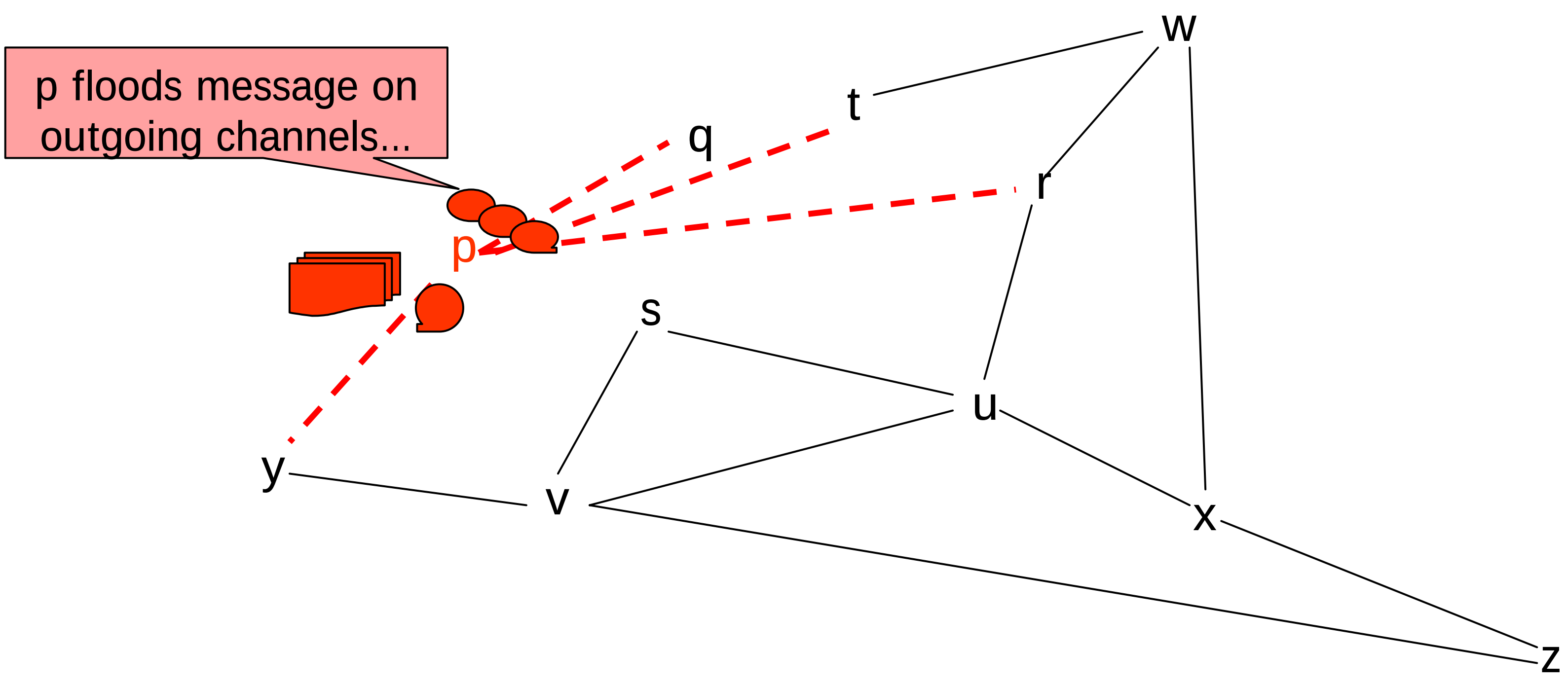
A network

Chandy/Lamport



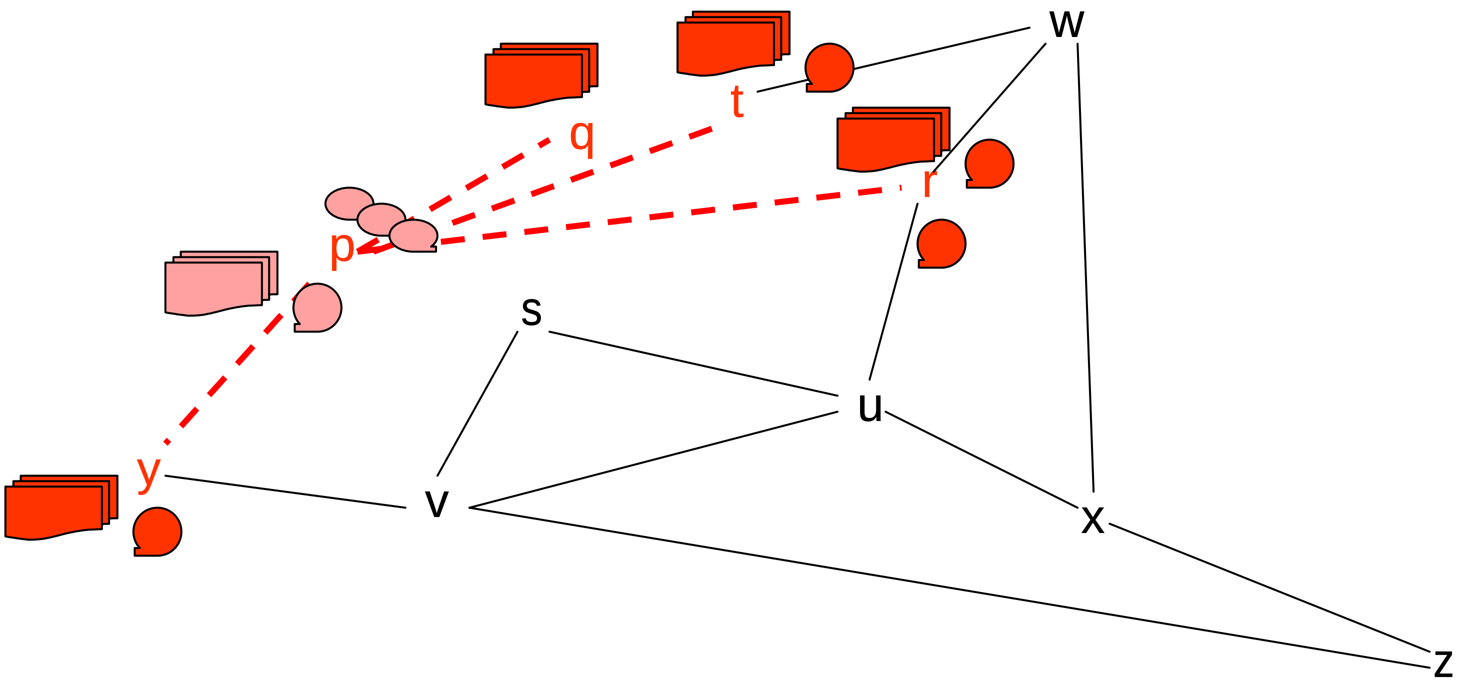
A network

Chandy/Lamport



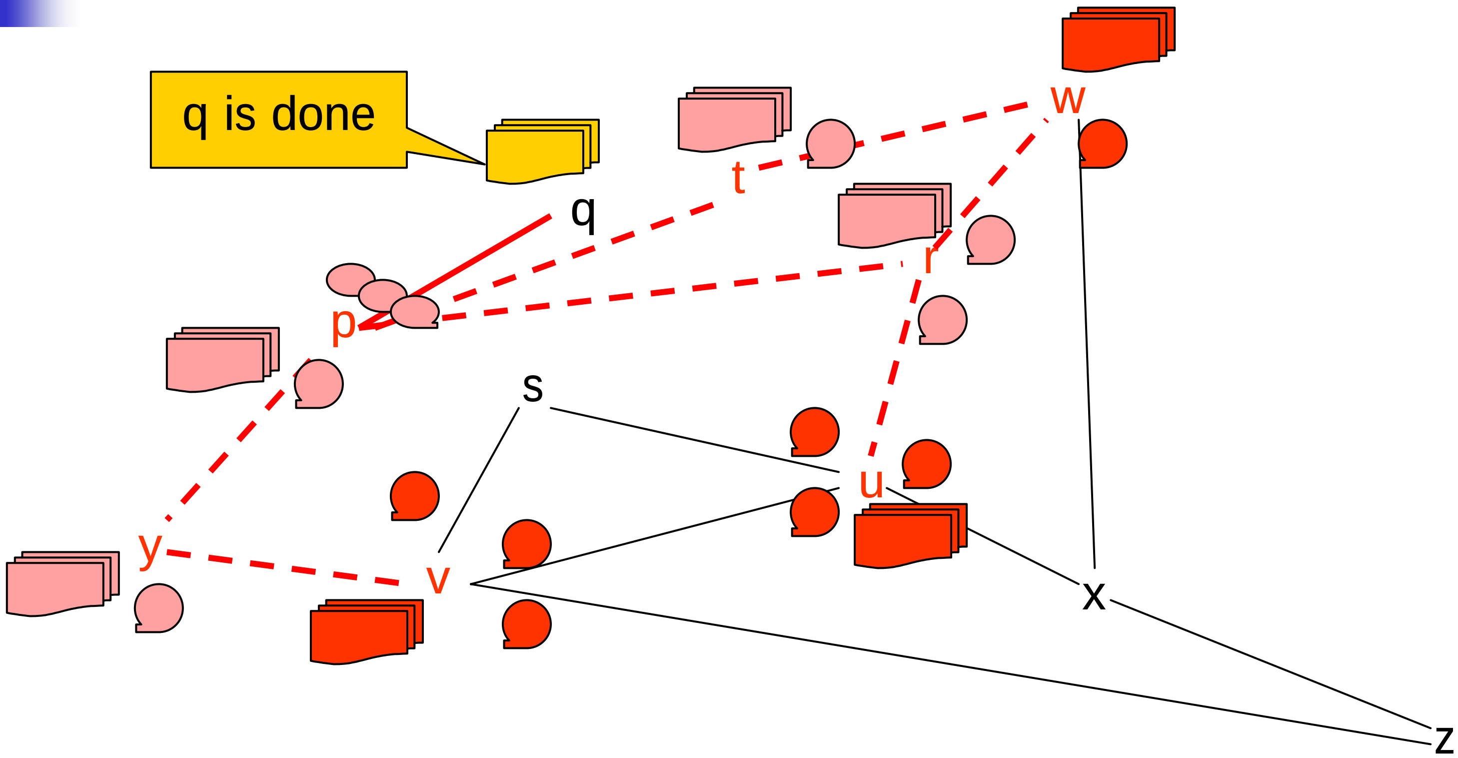
A network

Chandy/Lamport



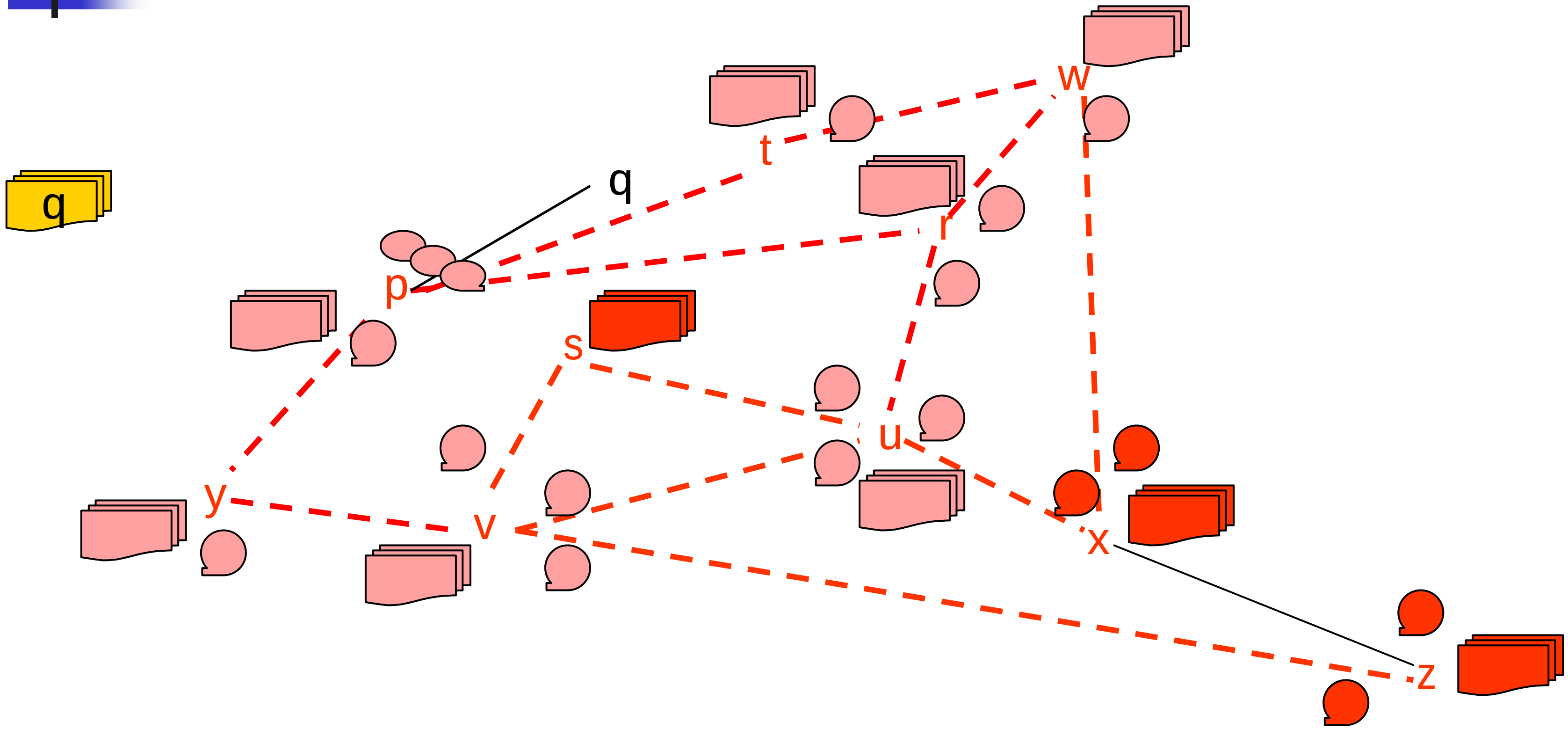
A network

Chandy/Lamport



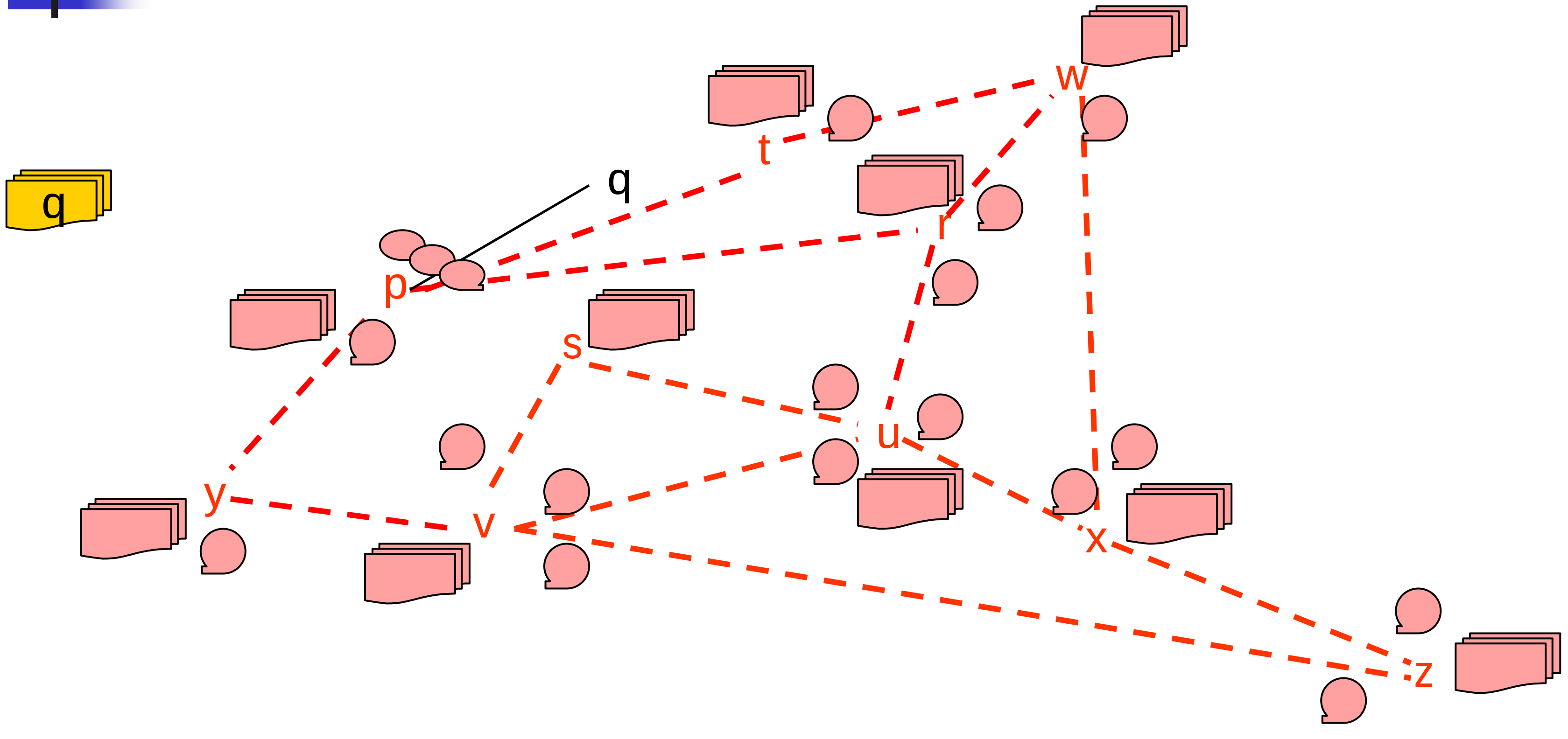
A network

Chandy/Lamport



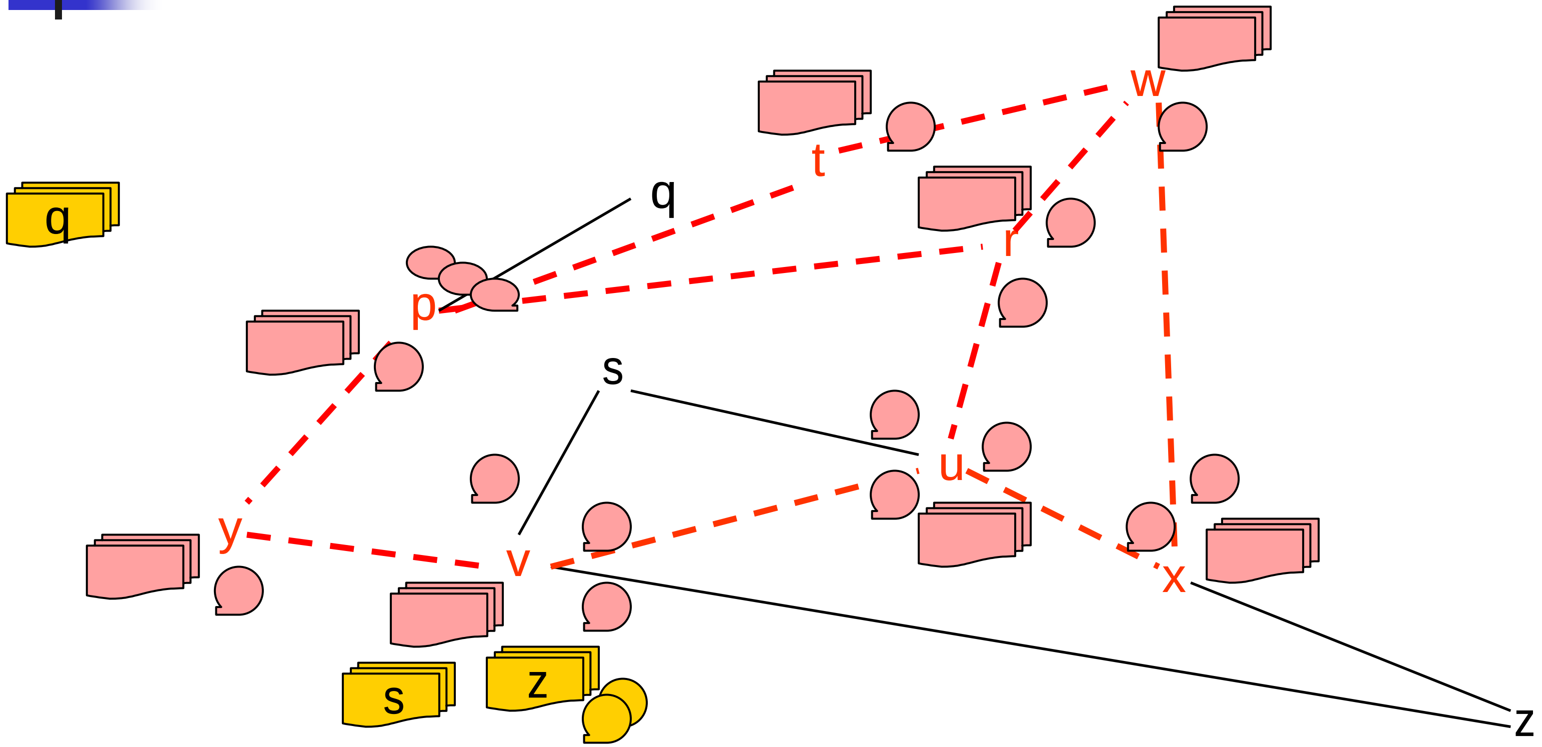
A network

Chandy/Lamport



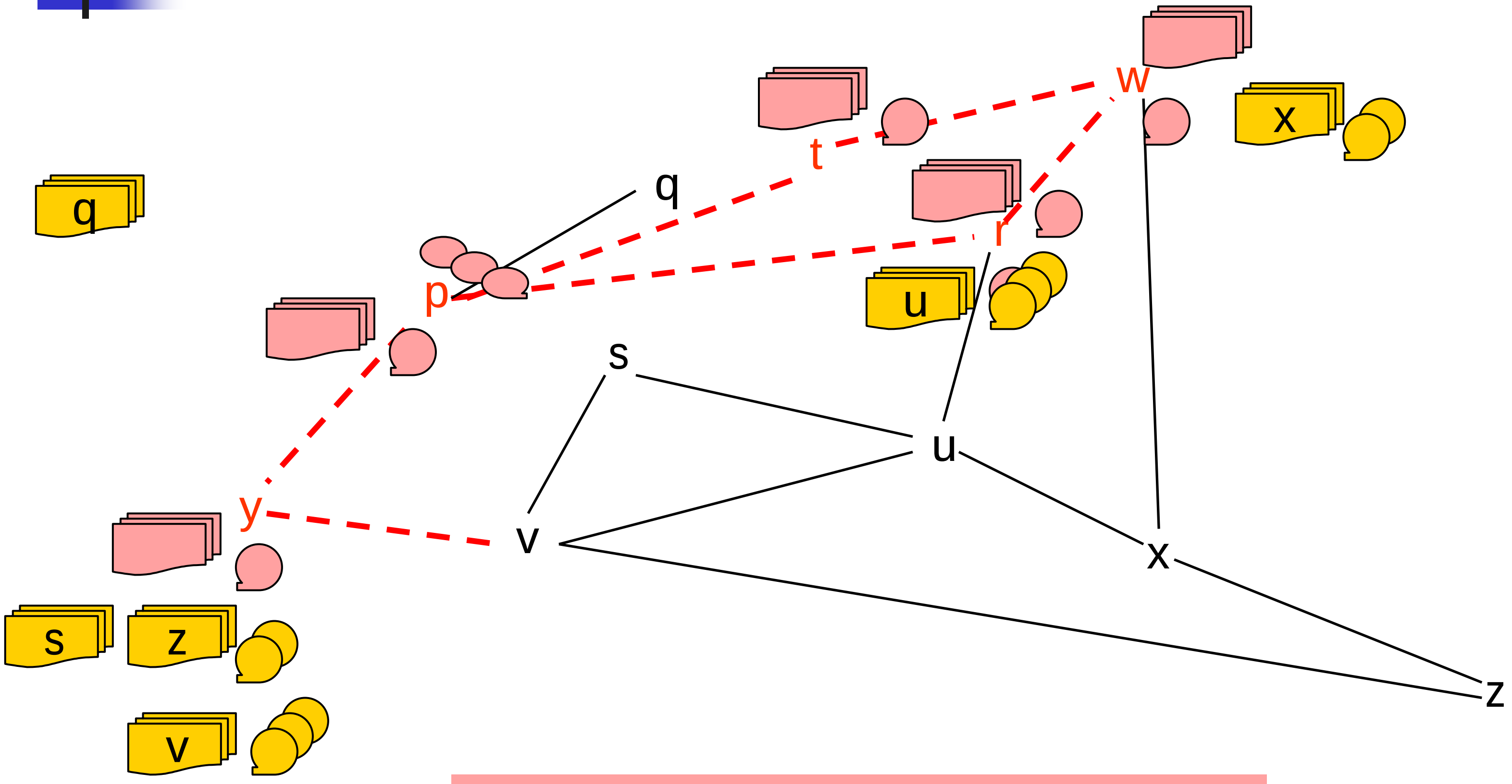
A network

Chandy/Lamport



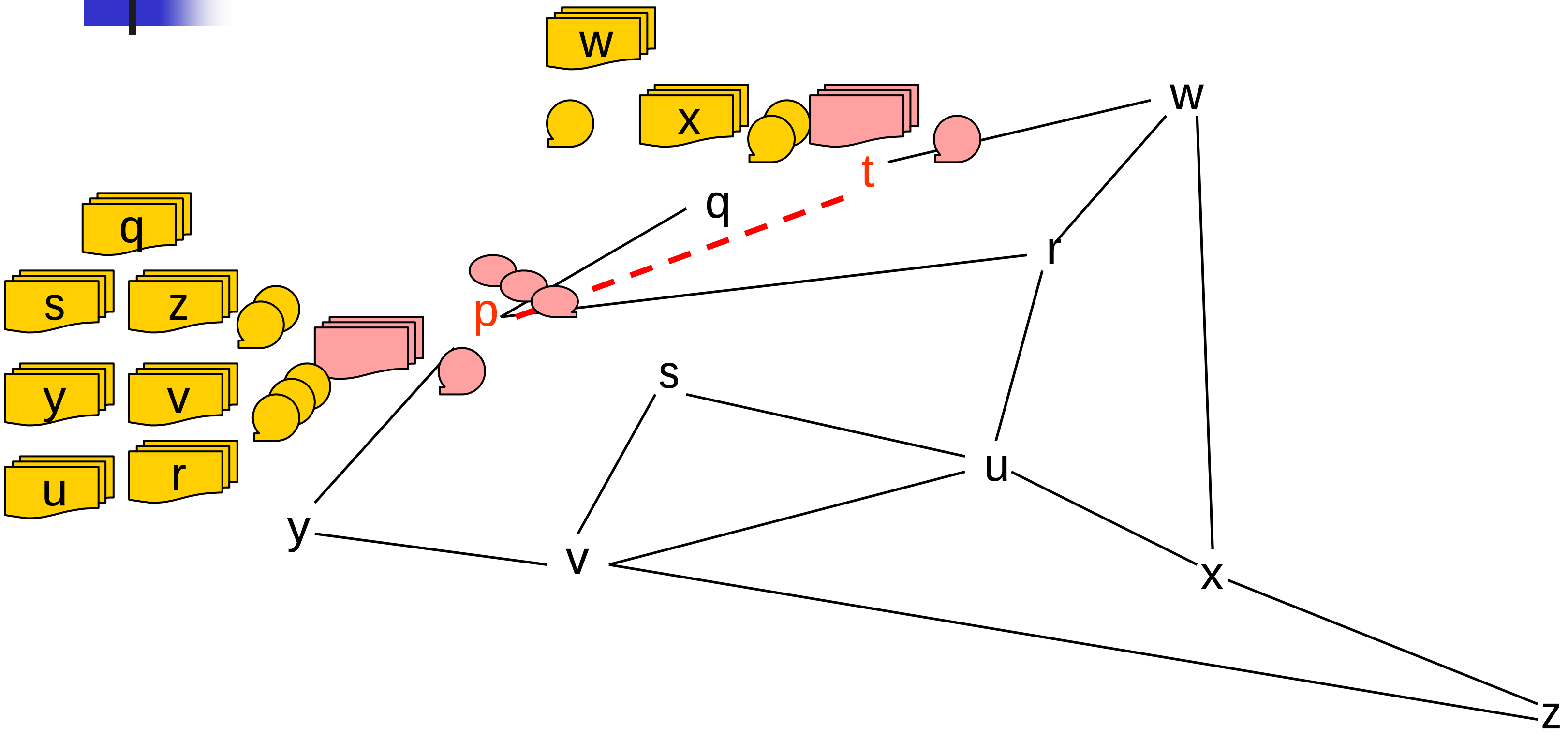
A network

Chandy/Lamport



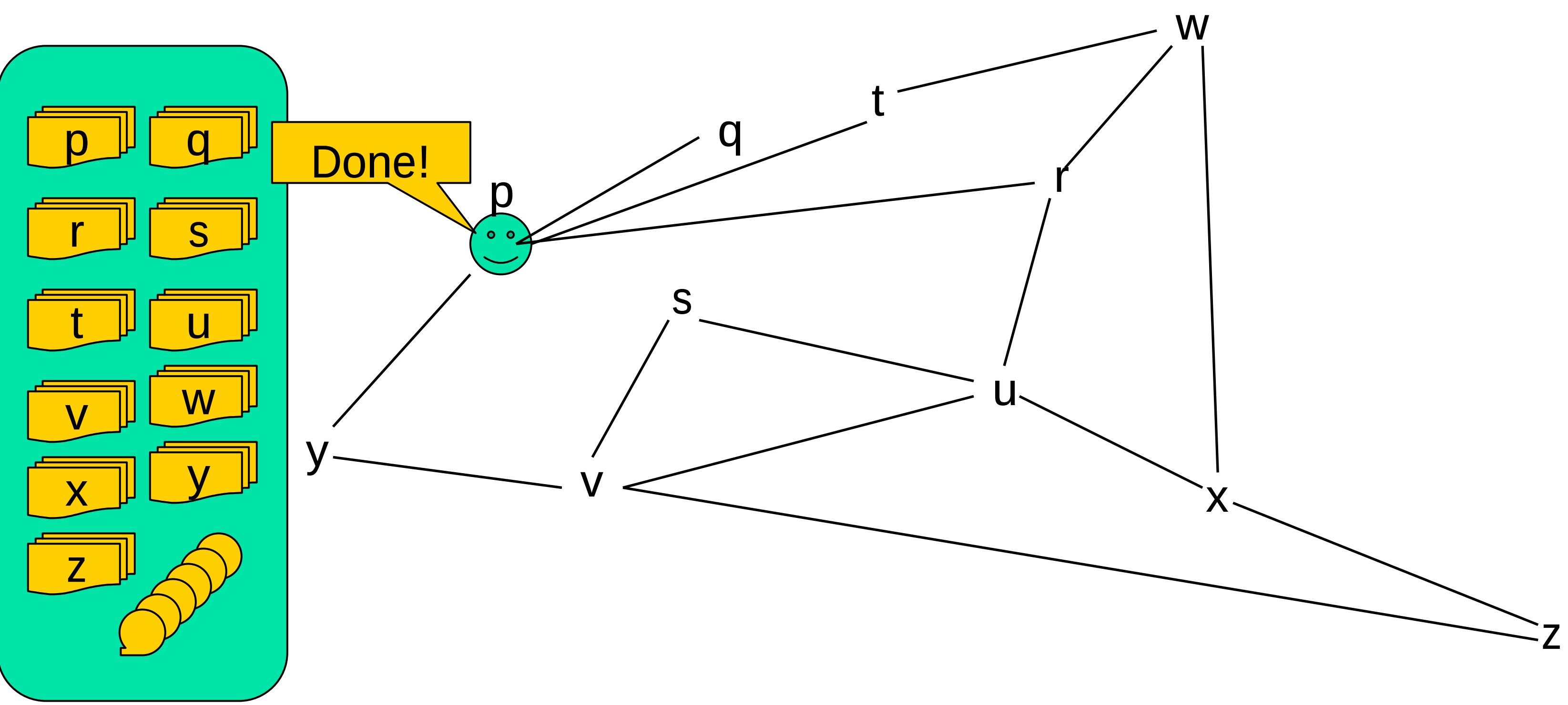
A network

Chandy/Lamport



A network

Chandy/Lamport



A snapshot of a network

What's in the "state"?

- In practice we only record things important to the application running the algorithm, not the "whole" state
  - E.g. "locks currently held", "lock release messages"
- Idea is that the snapshot will be
  - Easy to analyze, letting us build a picture of the system state
  - And will have everything that matters for our real purpose, like deadlock detection

Other algorithms?

- Many algorithms have a consistent cut mechanism hidden within
  - More broadly we'll see that notions of time are *sometimes* explicit in algorithms
  - But are often used as the insight that motivated the developer
  - By thinking about time, he or she was able to reason about a protocol
- We'll often use this approach