15-712:
Advanced Operating Systems & Distributed Systems

Distributed Snapshots: Determining Global States of Distributed Systems

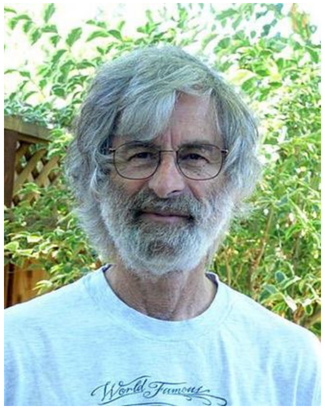
Prof. Phillip Gibbons

Spring 2023, Lecture 5

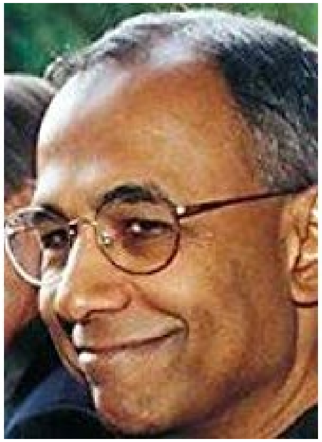
"Distributed Snapshots: Determining Global States of Distributed Systems"

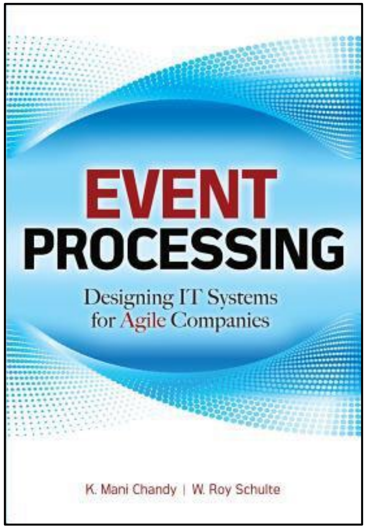
K. Mani Chandy & Leslie Lamport 1985

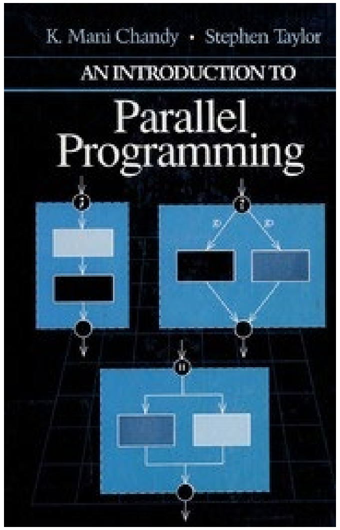
- **Leslie Lamport (SRI, DEC SRC, MSR)**
  - National Academy of Science

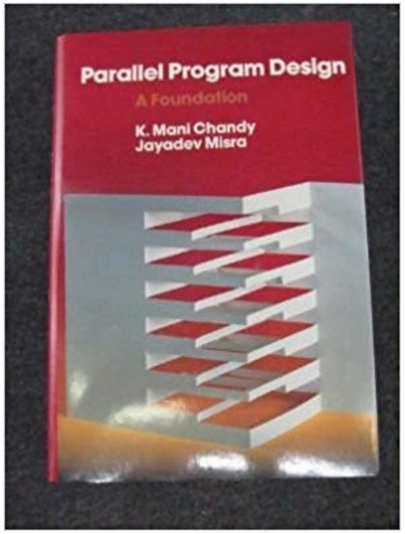


- **Mani Chandy (UT Austin, Caltech)**
  - National Academy of Engineering







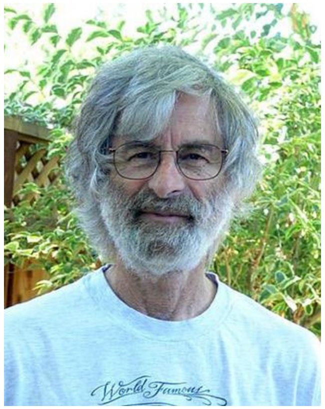


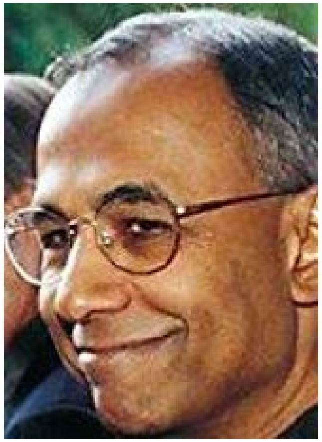
“Distributed Snapshots: Determining Global States of Distributed Systems”

K. Mani Chandy & Leslie Lamport 1985

SigOps HoF citation (2013):

This paper takes the idea of consistency for distributed predicate evaluation, formalizes it, distinguishes between stable and dynamic predicates, and shows precise conditions for correct detection of stable conditions. The fundamental techniques in the paper are the secret sauce in many distributed algorithms for deadlock detection, termination detection, consistent checkpointing for fault tolerance, global predicate detection for debugging and monitoring, and distributed simulation.





Leslie Lamport on Today's Paper

"The distributed snapshot algorithm described in this paper came about when I visited Chandy, who was then at the University of Texas at Austin.

He posed the problem to me over dinner, but we had both had too much wine to think about it right then.

The next morning, in the shower, I came up with the solution.

When I arrived at Chandy's office, he was waiting for me with the same solution.

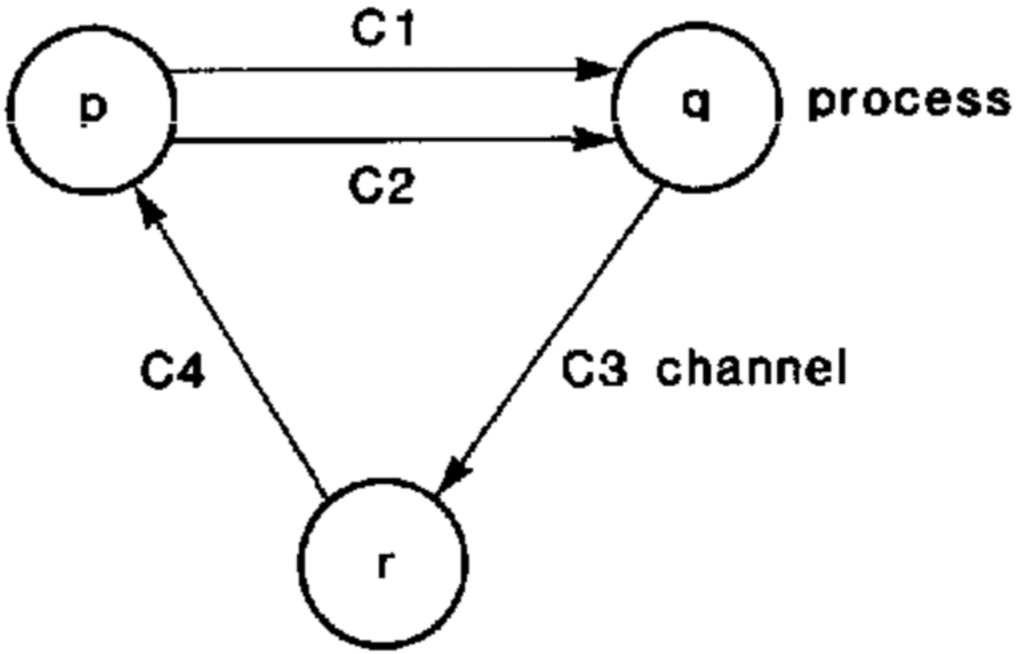
I consider the algorithm to be a straightforward application of the basic ideas from [27]."

[27] is "Time, Clocks, and the Ordering of Events in a Distributed System"

Global State Detection

System Model: Processes & Channels

- Finite labeled, directed graph in which vertices represent processes & edges represent channels



- Channels have infinite buffers, in-order delivery, arbitrary but finite delays, are uni-directional & error-free

System Model: Events

An event is defined by:

- process p
- state s of p immediately before the event
- state s’ of p immediately after the event
- channel c (if any) whose state is altered by the event
- message M (if any) sent/received along c

An event may change state of at most one channel, by either sending or receiving along that channel

An event is legal in a state s iff
all preconditions for that event are satisfied in s

Possible preconditions: state + head of incoming channel for a receive

Example

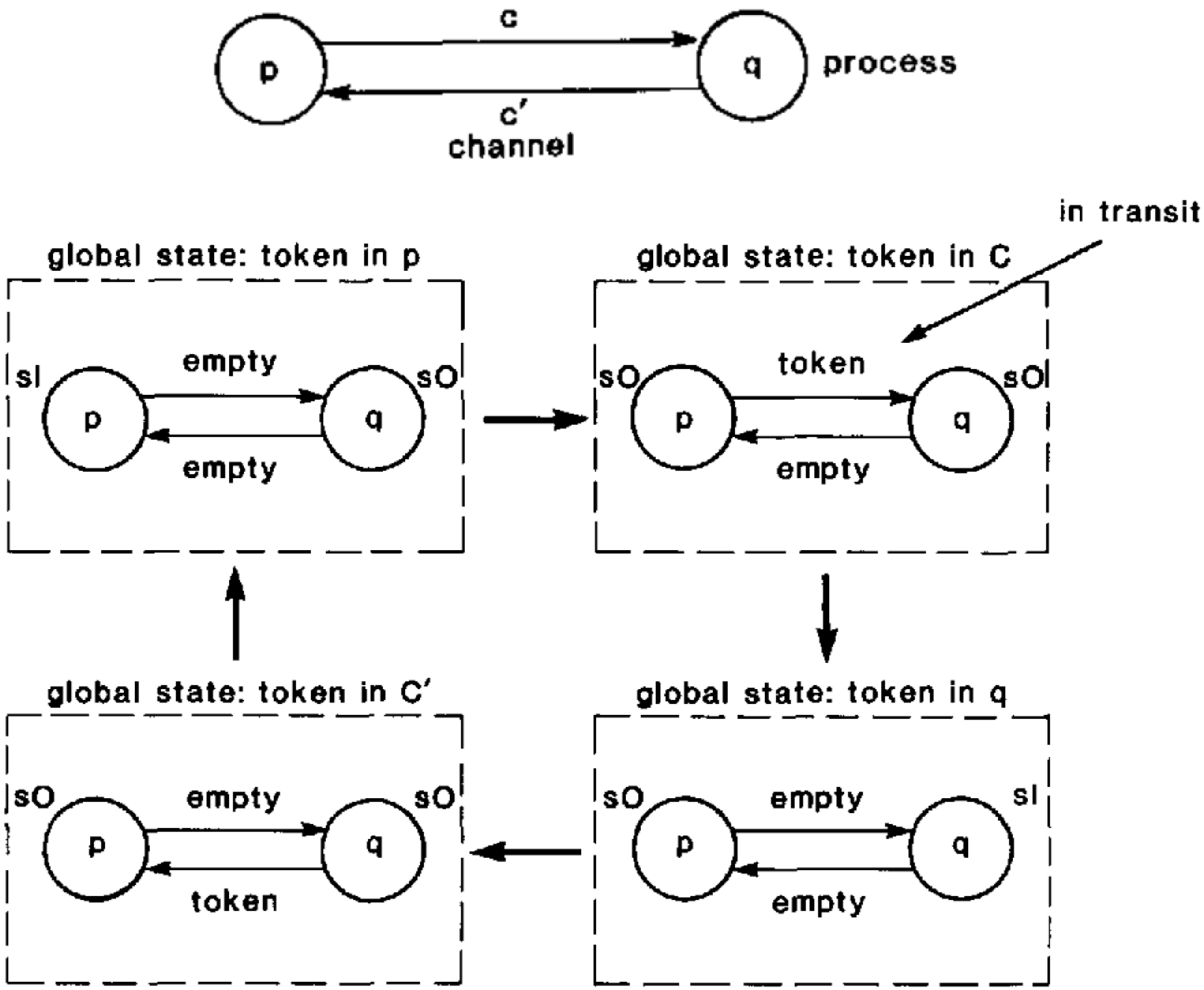
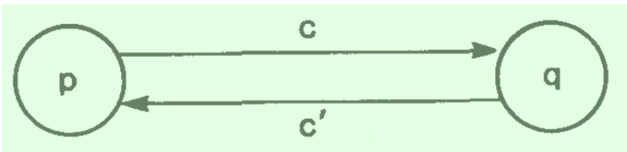
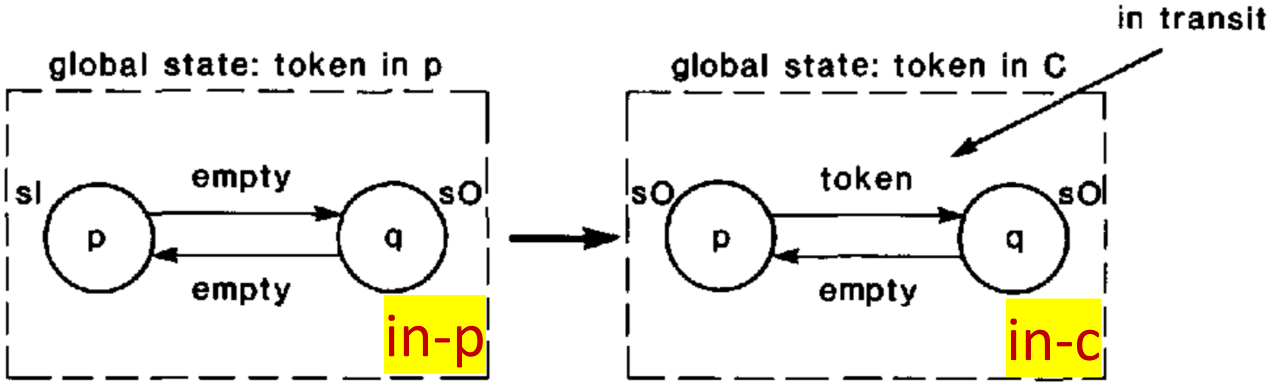


Fig. 4. Global states and transitions of the single-token conservation system.

Recording an Inconsistent Global State





- state of p: in-p (p has token)
- state of c: in-p (empty)

state transitions to in-c

- state of q: in-c
  state of c: has token
  state of c': empty
- state of p: in-c
  state of q: in-c
  state of c': empty

Problem: global state shows

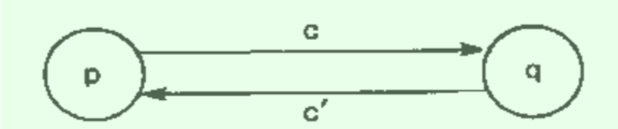
- 2 tokens in system
- 0 tokens in system

Global-State-Detection Algorithm

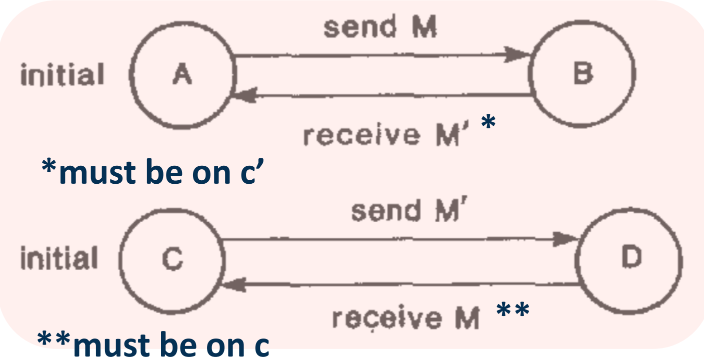
- **Marker-Sending Rule for p**
  For each channel c outgoing from p:
  - p records state, then sends a marker as its next message on c

- **Marker-Receiving Rule for q**
  On receiving a marker along a channel c:
  - If q has not recorded its state then
    q records its state; q records the state c as empty
  - Else q records state of c as the sequence of messages received along c after q's state was recorded yet before q received the marker along c

Termination: As long as at least 1 process spontaneously records it state & no marker remains stuck in a channel & the graph is strongly connected, then all processes record their states in finite time

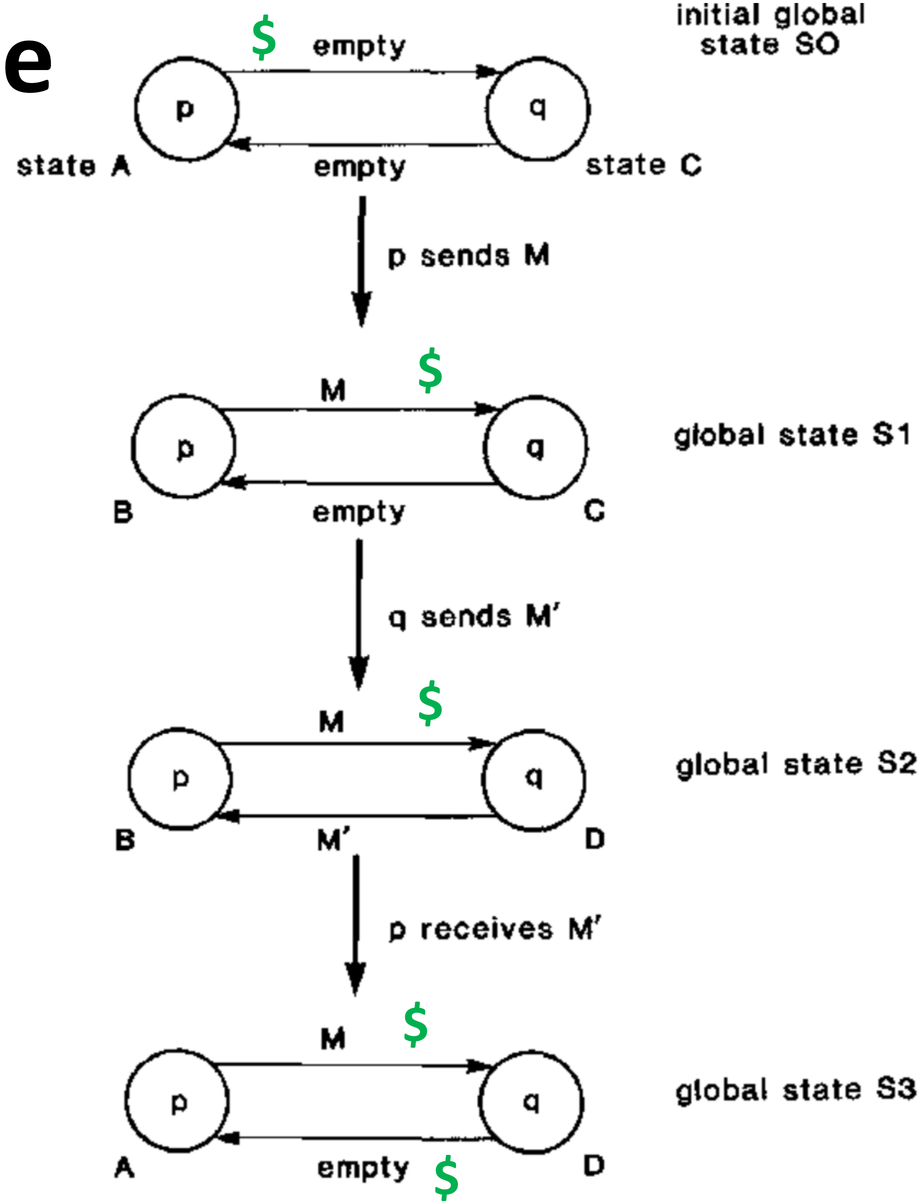


Example

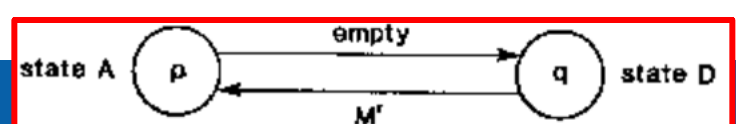


State diagram for p

State diagram for q



1. In S0, p records state=A, puts marker $ on c
2. p puts M on c (S1)
   q puts M' on c' (S2)
   p receives M' (S3) *records M'*
3. Marker received by q;
   q records state=D, c=empty,
   q puts marker on c'
4. Marker received by p;
   p records c'=<M'>



Never Happened!

So…In What Way is the Recorded Global State "Meaningful" ?

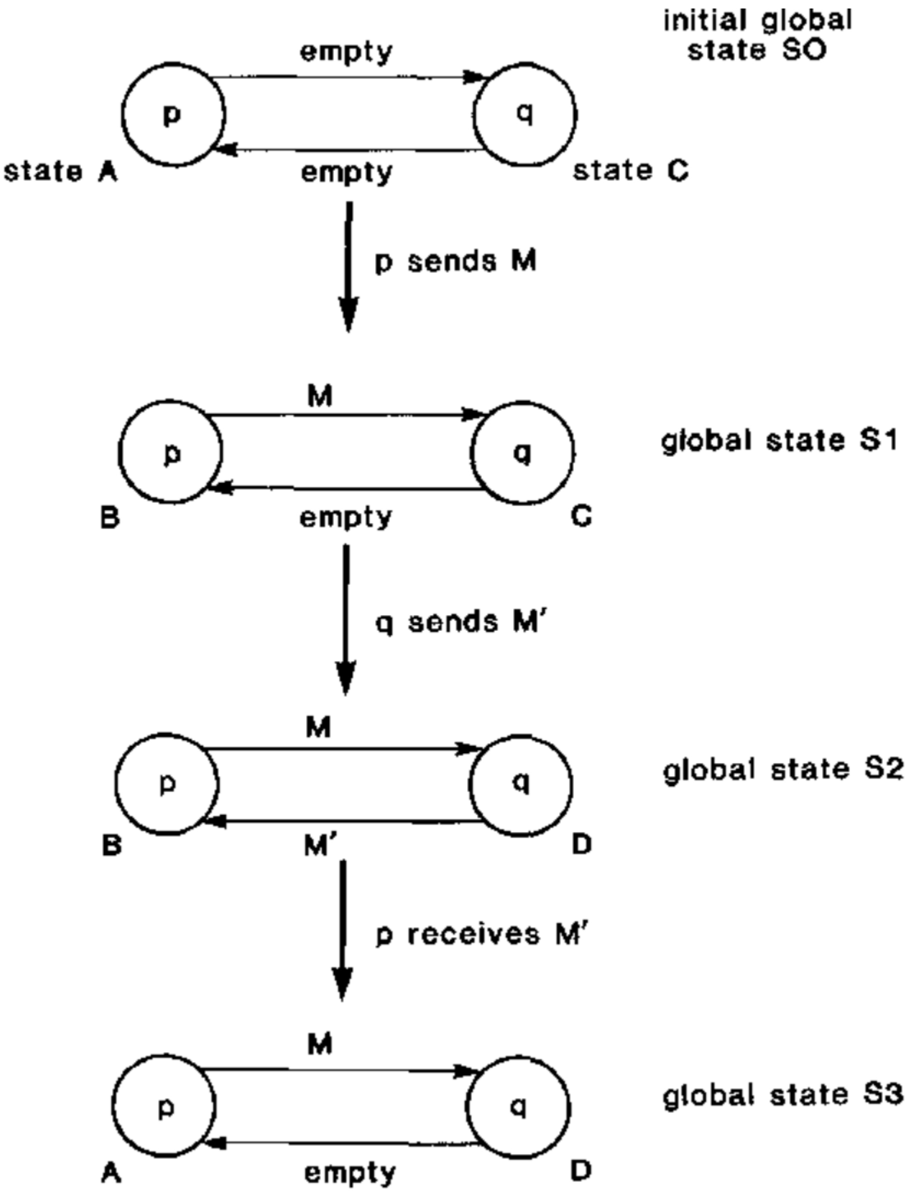
- **It *could* have occurred**

- **Theorem 1: There is a computation where**
  - Sequence of states before the DS algorithm starts is unchanged
  - Sequence of states after the DS algorithm ends is unchanged
  - Sequence of events in between may (only) be reordered
  - Recorded global state is one of the states in between

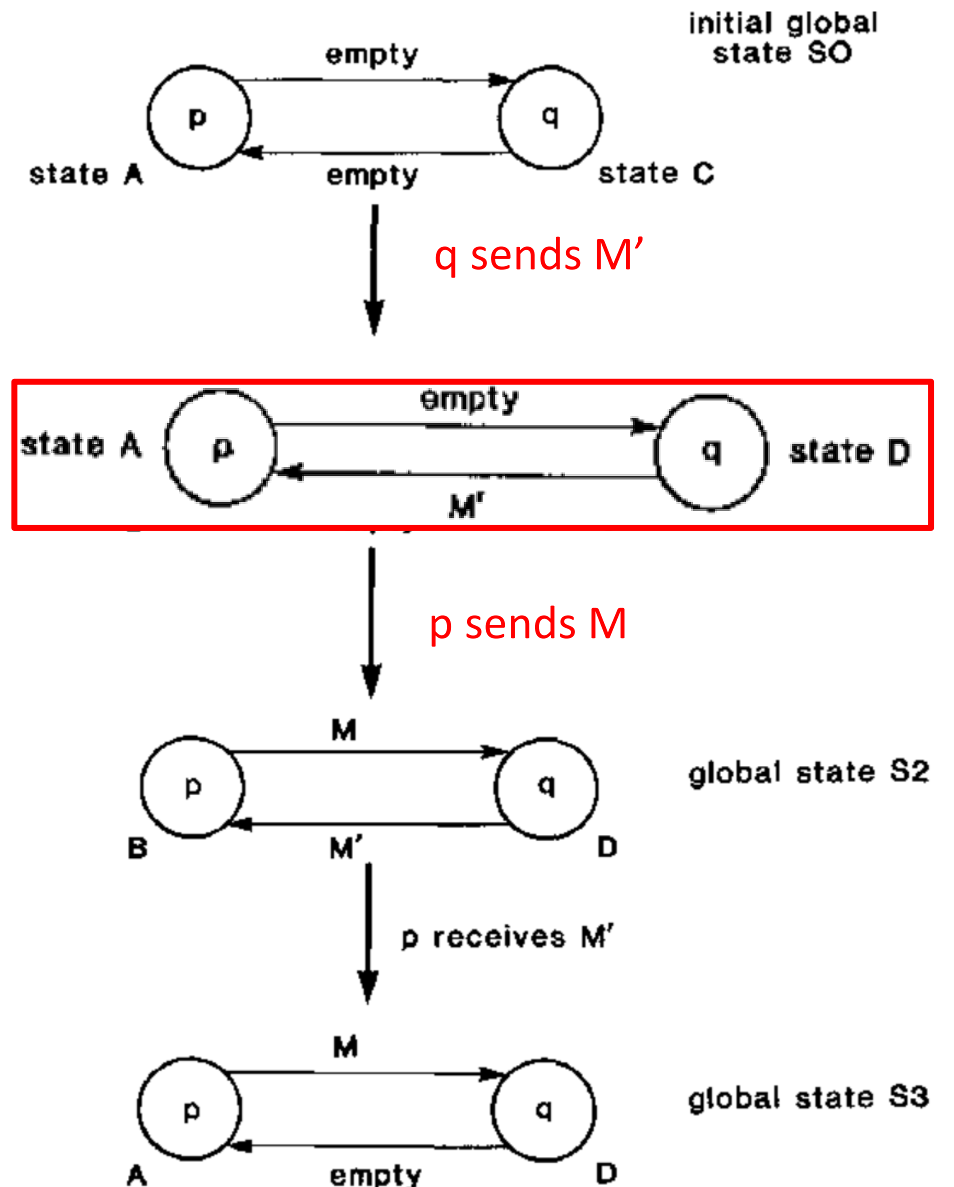
- **But why is that useful???**

Applying to Prior Example

Never Happened!



But Could’a Happened



Discussion: Summary Question #1

- **State the 3 most important things the paper says.** These could be some combination of their motivations, observations, interesting parts of the design, or clever parts of their implementation.

Stability Detection

- **Input: Any stable property y**

  Stable: y(S) implies y(S') for all global states S' reachable from S

  - E.g., the kth computation phase has terminated, k=1,2,…
  - E.g., the computation is deadlocked/livelocked

- **Output: TRUE or FALSE**
  - Returning TRUE implies property y holds when DS algorithm ends
  - Returning FALSE implies property y did not hold when DS algorithm starts
    i.e., property y holds when DS algorithm starts implies must return TRUE

  Note: If y starts holding after DS start, ok to return FALSE

Stability Detection Algorithm

- **Input: Any stable property y**

  Stable: y(S) implies y(S') for all global states S' reachable from S

- **Must Guarantee:**
  - Returning TRUE implies property y holds when DS algorithm ends
  - Property y holds when DS algorithm starts implies must return TRUE

- **Stability Detection Algorithm:** Record a global state S*; Return y(S*)

- **Correctness:**
  - y(S*)=TRUE implies y(DS end state)=TRUE [because reachable, y stable]
  - y(DS start state)=TRUE implies y(S*)=TRUE [because reachable, y stable]

Discussion: Summary Question #2

- **Describe the paper's single most glaring deficiency.** Every paper has some fault. Perhaps an experiment was poorly designed or the main idea had a narrow scope or applicability.

Key Theorem & Proof Sketch

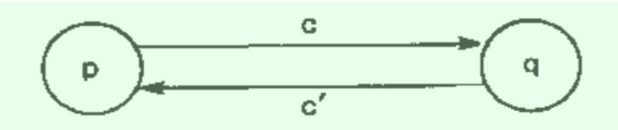
- **Theorem 1: There is a computation seq' derived from seq where**
  - Sequence of states before/after DS starts/ends is unchanged
  - Sequence of events in between may (only) be reordered
  - Recorded global state S* is one of the states in between

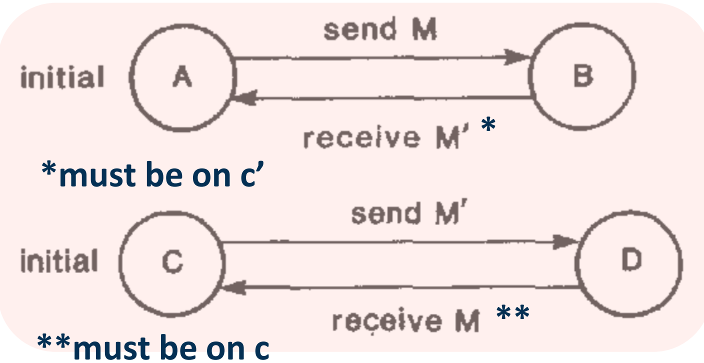
- **Prerecording event: occurs at p before p records its state**
  Postrecording event: …after…

- **seq' is seq permuted such that all prerecording events occur before any postrecording events**

- **Must show:**
  - seq' is a legal computation
  - S* is the global state in seq' at the transition point

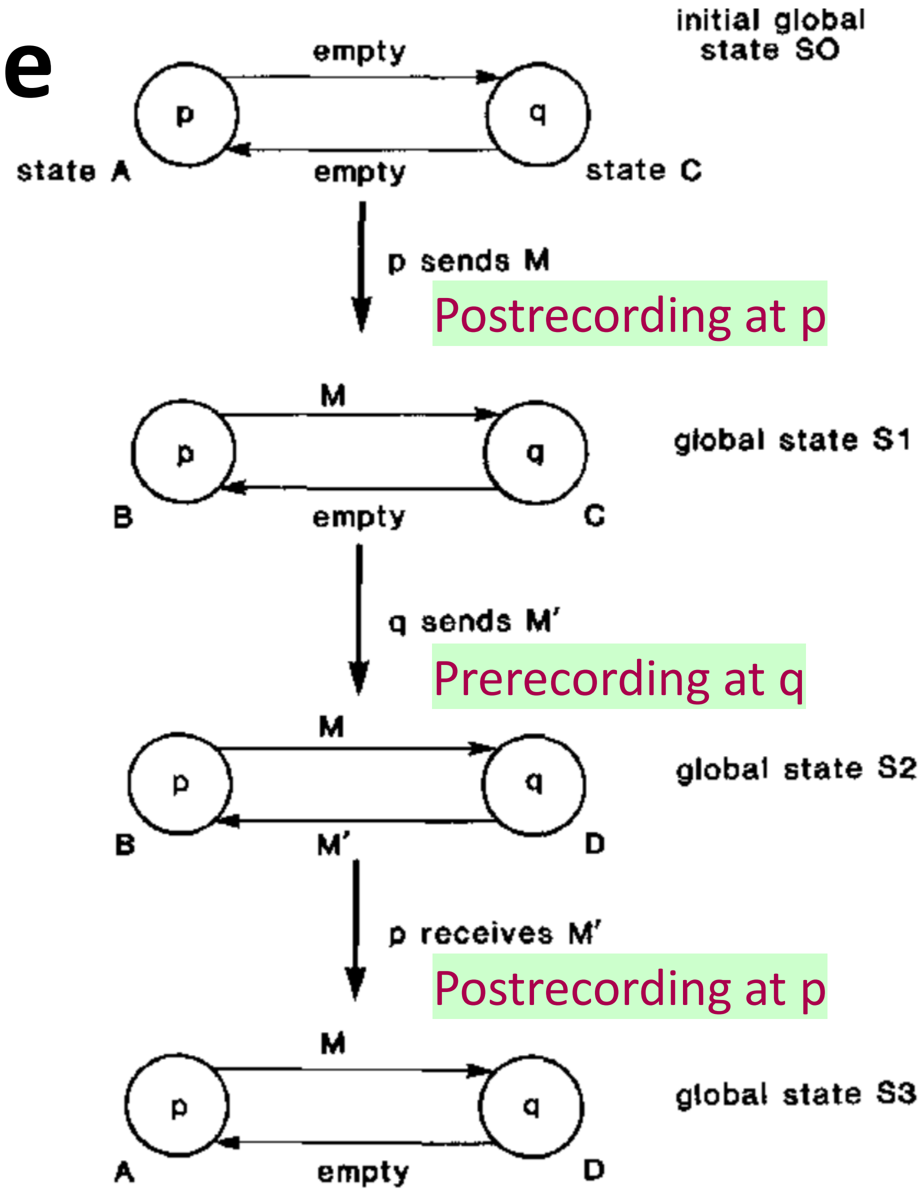
Example





State diagram for p

State diagram for q



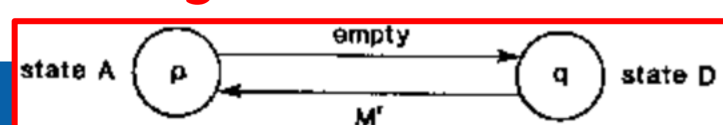
1. In S0, p records state=A,
   puts marker on c

2. p puts M on c (S1)
   q puts M' on c' (S2)
   p receives M' (S3) *records M'*

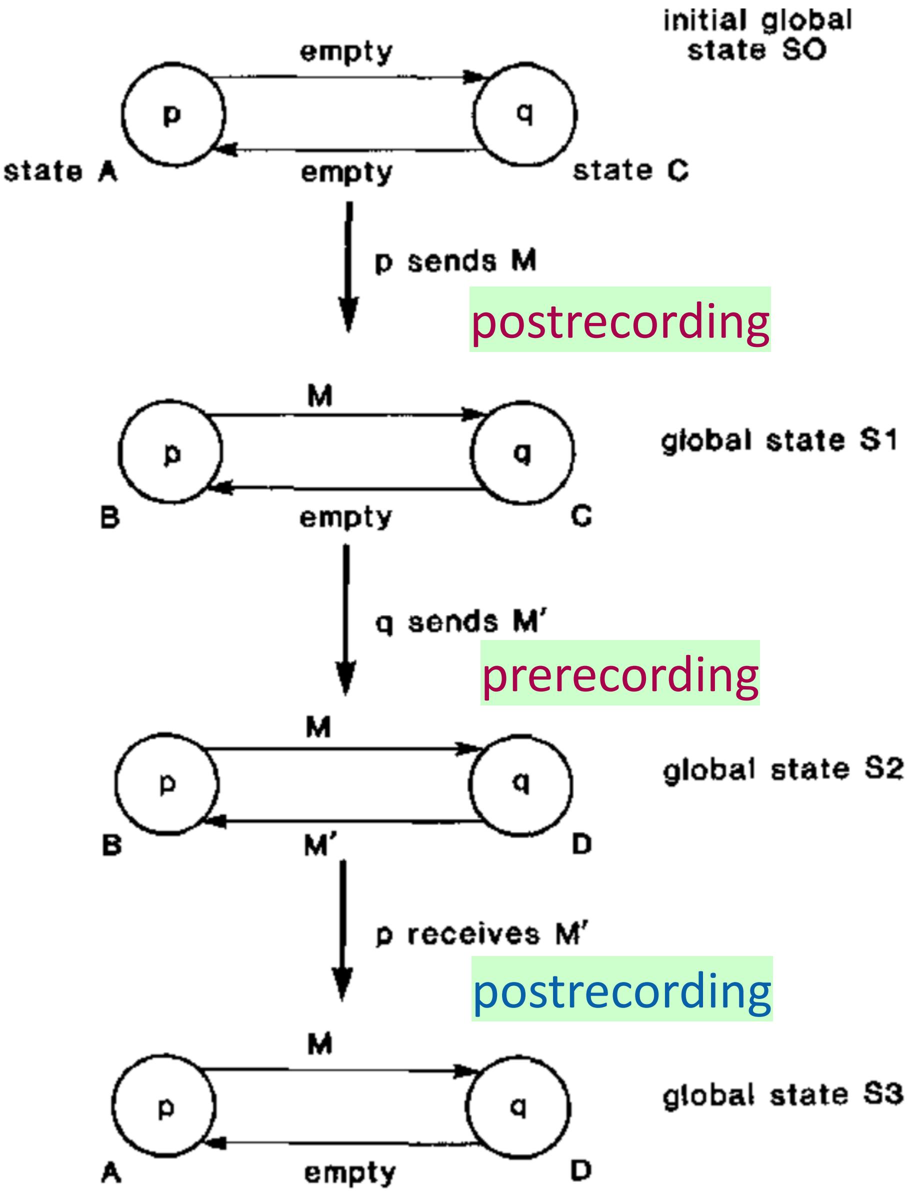
3. Marker received by q;
   q records state=D, c=empty,
   q puts marker on c'

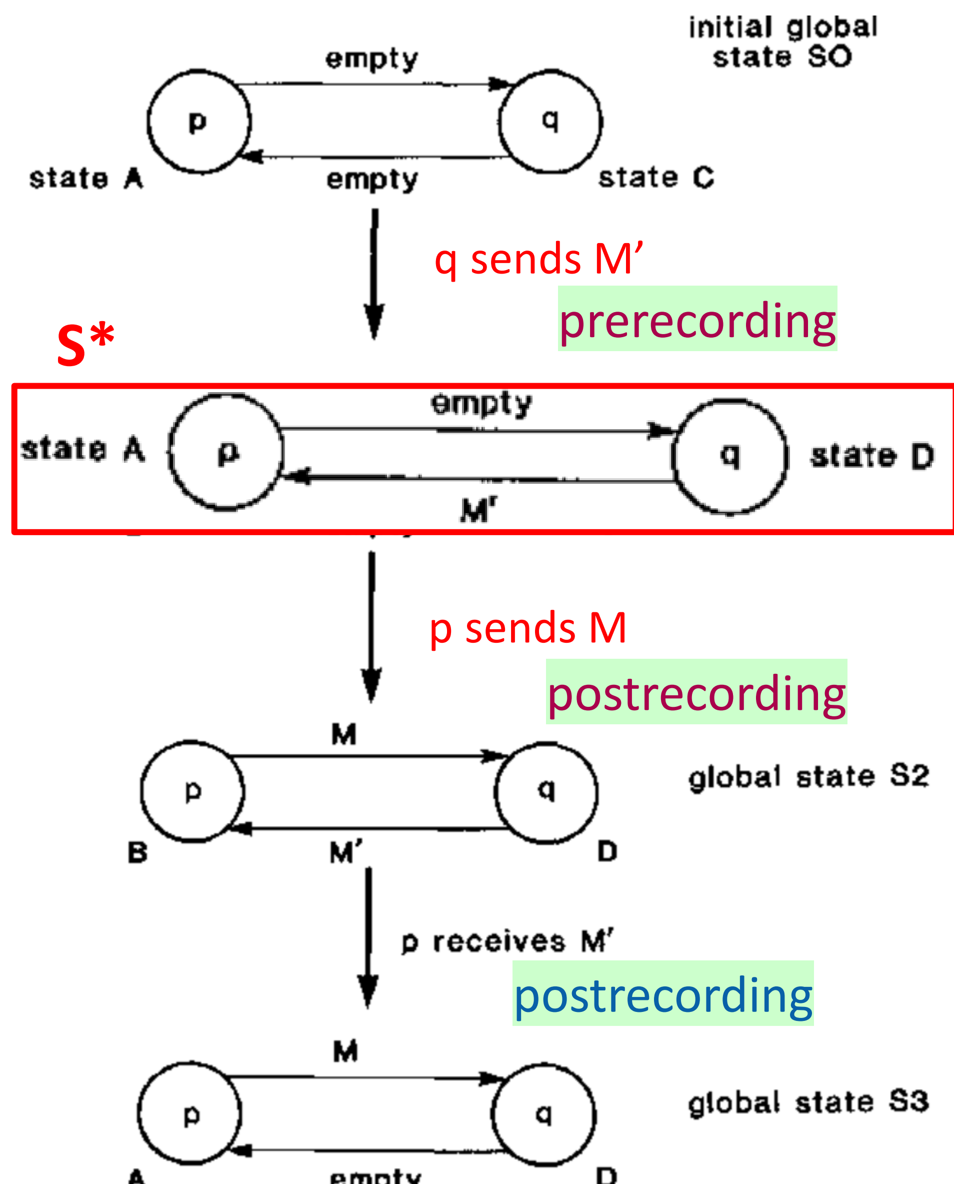
4. Marker received by p;
   p records c'=<M'>

S*



Example: Swapping Post and Pre



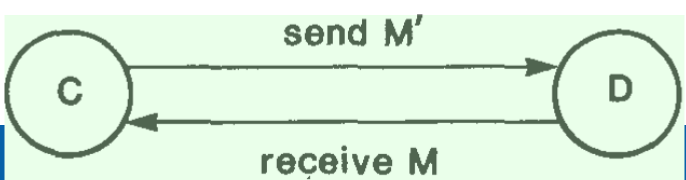


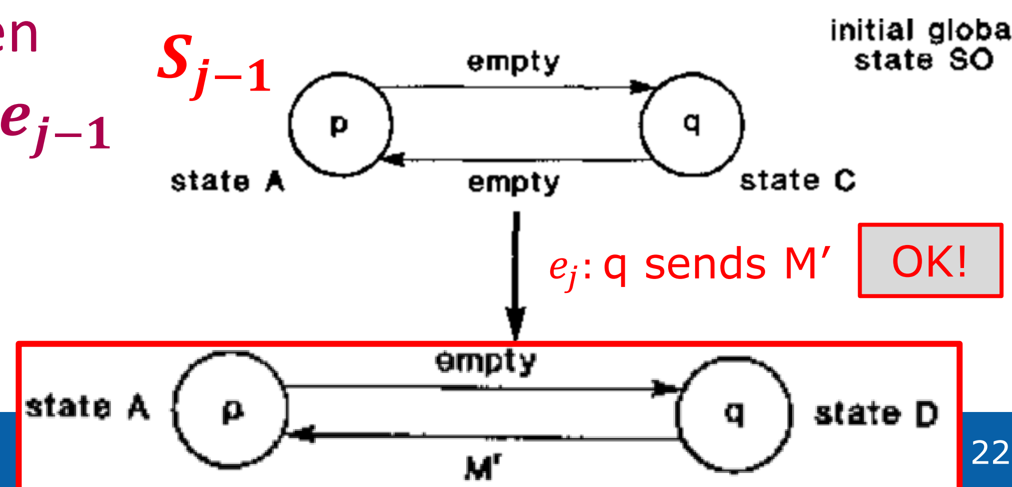
Swapping Post and Pre

Why is it legal to swap e_{j-1} (post) and e_j (pre) ?

First show e_j can occur in S_{j-1}:

- Must be on different processes, say $\boldsymbol{e_{j-1}}$ at p and $\boldsymbol{e_j}$ at q
- No message M sent at $\boldsymbol{e_{j-1}}$ received at $\boldsymbol{e_j}$
  Why? Since $\boldsymbol{e_{j-1}}$ is post, marker already sent
  If M received at $\boldsymbol{e_j}$ then q already received marker
  & recorded state, so e_j would be post
- State of q not altered by occurrence of $\boldsymbol{e_{j-1}}$ since at p
- If $\boldsymbol{e_j}$ is a receive M along c, then
  M already at head of c before $\boldsymbol{e_{j-1}}$
- Thus, $\boldsymbol{e_j}$ can occur in S_{j-1}



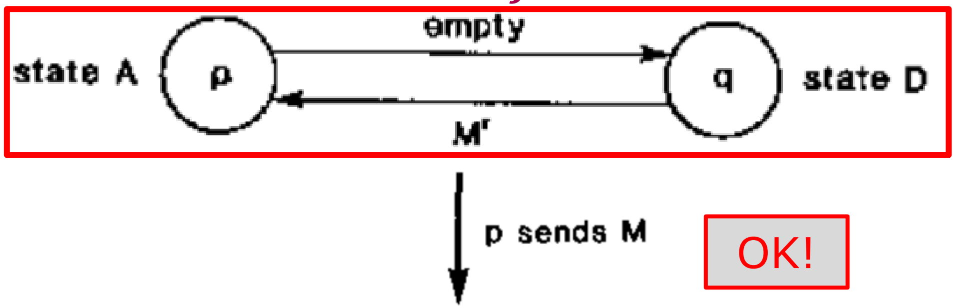


Swapping Post and Pre

Why is it legal to swap e_{j-1} (post) and e_j (pre) ?

Next show e_{j-1} can occur immediately after e_j

- State of p not altered by occurrence of $\boldsymbol{e_j}$ at q
- If $\boldsymbol{e_{j-1}}$ is a receive of M' on c', then M' still at head of c' after $\boldsymbol{e_j}$ (even if $\boldsymbol{e_j}$ were a send along c', it can't jump the FIFO order)
- Thus, $\boldsymbol{e_{j-1}}$ preconditions are satisfied, and hence it can occur immediately after $\boldsymbol{e_j}$



state immediately after e_j

event e_{j-1}

Moreover, state after $e_1, \dots, e_{j-2}, e_j, e_{j-1}$ is same as $e_1, \dots e_{j-1}, e_j$

Completing the Proof

Theorem 1: There is a computation seq' derived from seq where

- Sequence of states before/after DS starts/ends is unchanged
- Sequence of events in between may (only) be reordered
- Recorded global state S* is one of the states in between

- **Repeatedly pairwise swap until all pre before any post**

- **S* is the same as state at pre-to-post transition**
  - Follows from Marker-Send and Marker-Receive rules

QED

Discussion: Summary Question #3

- **Describe what conclusion you draw from the paper as to how to build systems in the future.** Most of the assigned papers are significant to the systems community and have had some lasting impact on the area.

Wednesday

"Efficient and Scalable Thread-Safety Violation Detection"

Guangpu Li, Shan Lu, Madanlal Musuvathi, Suman Nath, Rohan Padhye 2019

Optional Further Reading:

"Eraser: A Dynamic Data Race Detector for Multi-Threaded Programs"

Stefan Savage, Michael Burrows, Greg Nelson, Patrick Sobalvarro, Thomas Anderson 1997