Ordering and Consistent Cuts

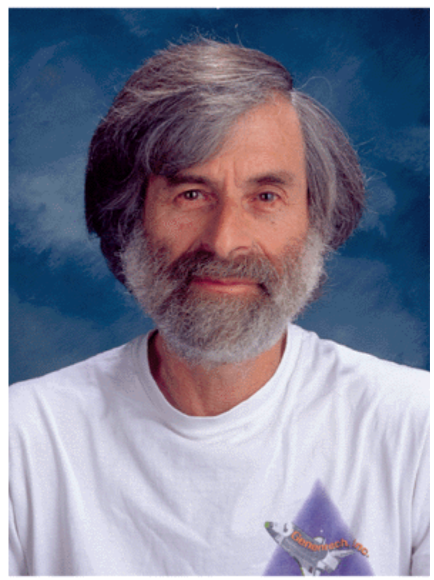
Nicole Caruso
Cornell University
Dept. of Computer Science

Time, Clocks, and the Ordering of Events in a Distributed System

Leslie Lamport
Stanford Research Institute

About the Author

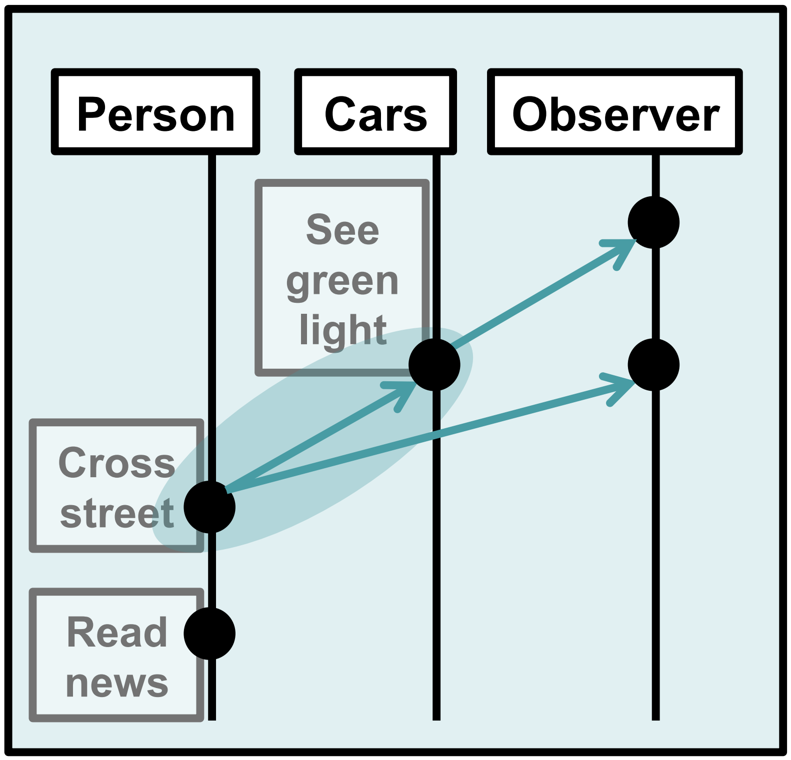
Leslie Lamport
Stanford Research Institute

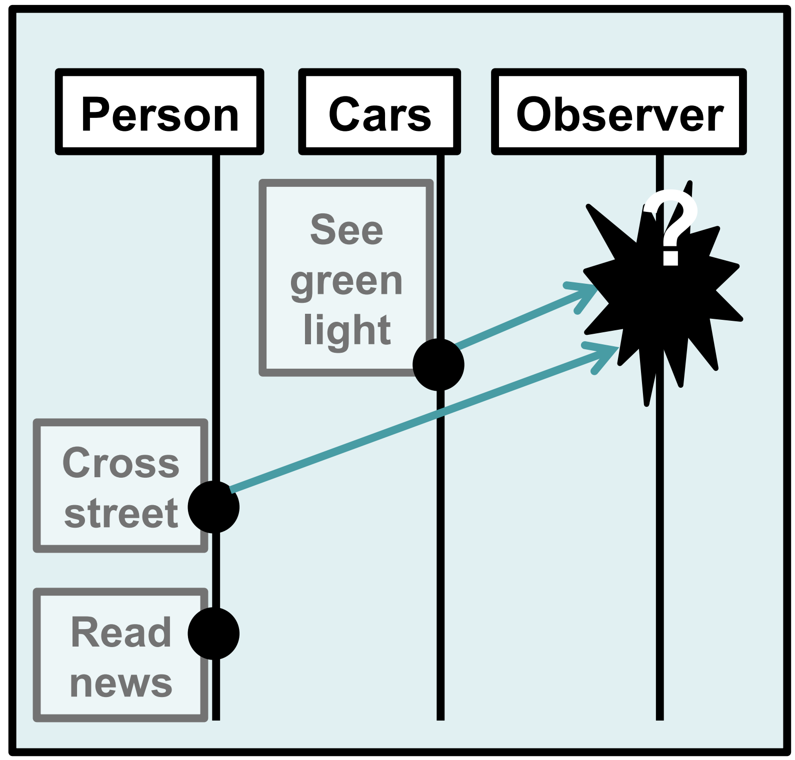


Time, Clocks, and the Ordering of Events in a Distributed System

Introduction

- Our concept of time
- Distributed system's concept of time





Time, Clocks, and the Ordering of Events in a Distributed System

Introduction

- Coordination of Distributed Systems
- Lack of Understanding
- Partial Ordering of Events
- Total Ordering of Events

Outline

- **Partial Ordering of Events**
- Logical Clocks
- Total Ordering of Events
- Anomalous Behavior
- Physical Clocks

Partial Ordering of Events

- **System Definition**
  - **System** contains spatially separated processes
  - **Process** contains a sequence of events
  - **Event** manifestation is arbitrary, but must include message sending and message receiving

Partial Ordering of Events

- **Mathematical Properties**
  - **Asymmetric**
    - If **a\<b**, then **b<a**
    - If **a\<b** and **b\<a**, then **a=b**
  - **Transitive**
    - If **a\<b** and **b\<c**, then **a\<c**
  - **Reflexive**
    - **a\<a**

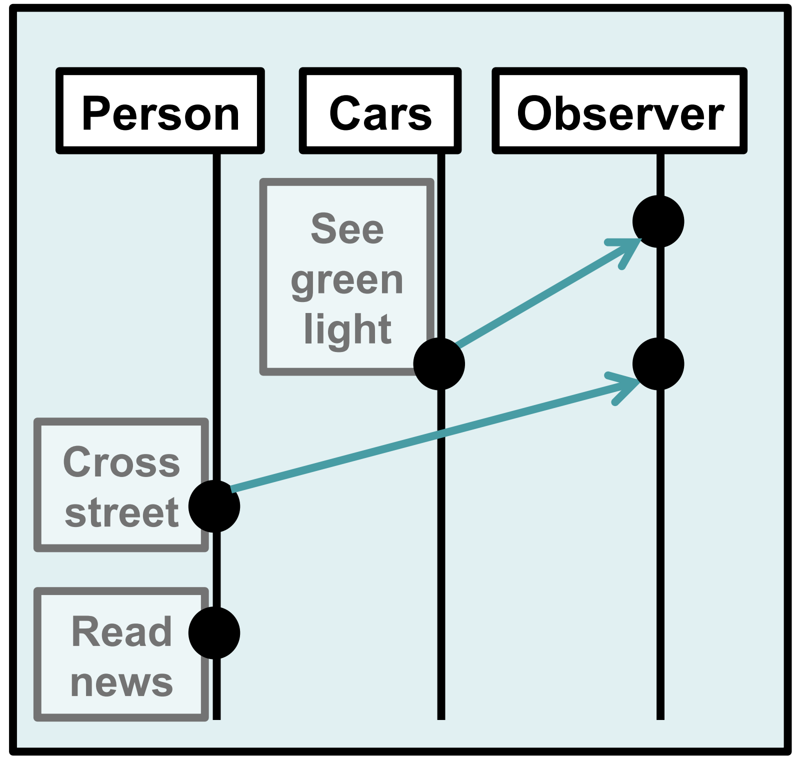
Partial Ordering of Events

- **“Happened Before” Relation (→)**
  - **Asymmetric:** If **a→b**, then **b\→a**
    - If **a** and **b** are in same process
      - **a→b** if **a** occurs before **b**
    - If **a** and **b** are in different processes
      - **a→b** if **a** is sending of message and **b** is receipt of message
    - If **a** is **concurrent** with **b**
      - **a\→b** and **b\→a**
  - **Transitive:** If **a→b** and **b→c**, then **a→c**
  - **Reflexive:** **a\→a**

Time, Clocks, and the Ordering of Events in a Distributed System

Partial Ordering of Events

- **Typical Space-Time Diagram for a Distributed System**



Outline

- Partial Ordering of Events
- **Logical Clocks**
- Total Ordering of Events
- Anomalous Behavior
- Physical Clocks

Logical Clocks

- **Process Clock C_i**
  - Clock assigns number to event to represent time
    - Assigns $\mathbf{C_i(a)}$ to each event **a** within $\mathbf{P_i}$
    - Belongs to one process $\mathbf{P_i}$
- **System Clock C**
  - Clock $\mathbf{C(a) = C_i(a)}$

Logical Clocks

- **Clock Condition**: If **a→b**, then **C(a) < C(b)**
  - If events **a** and **b** are in the same process $\mathbf{P_i}$
    - $\mathbf{C_i(a)<C_i(b)}$
      - if **a** occurs before **b**
  - If events **a** and **b** are in processes $\mathbf{P_i}$ and $\mathbf{P_j}$
    - $\mathbf{C_i(a)<C_j(b)}$
      - **a** is the sending of a message
      - **b** is the receipt of the message

Outline

- Partial Ordering of Events
- Logical Clocks
- **Total Ordering of Events**
- Anomalous Behavior
- Physical Clocks

Total Ordering of Events

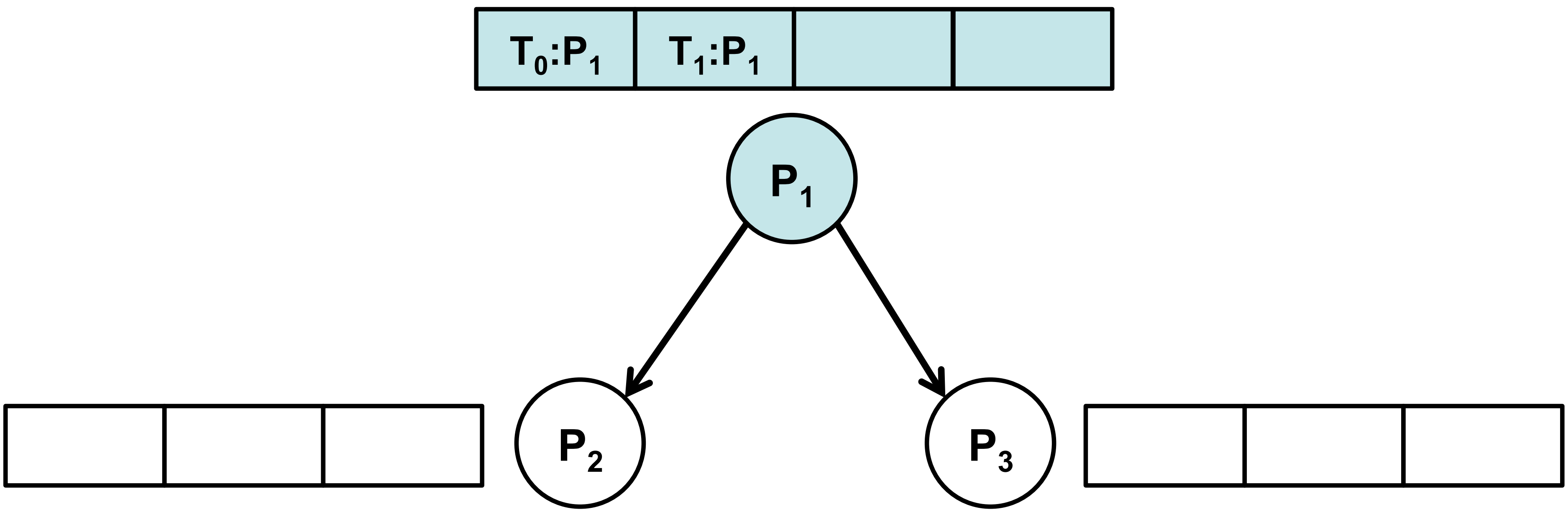
- **Total ordering eliminates concurrency**
- **Identify message with event sending it**
- **Following Example**
  - Multiple processes compete for resource
  - Told from point of view of one process P_i

Total Ordering of Events

- **Process P_i is granted resource**
  - **Request T_m:P_i**
    - In **P_i**'s request queue
    - Time-stamped before other requests in the queue
  - **Acknowledge T_m:P_i**
    - Received from **P_j**
    - Time-stamped later than **Request T_m:P_i**

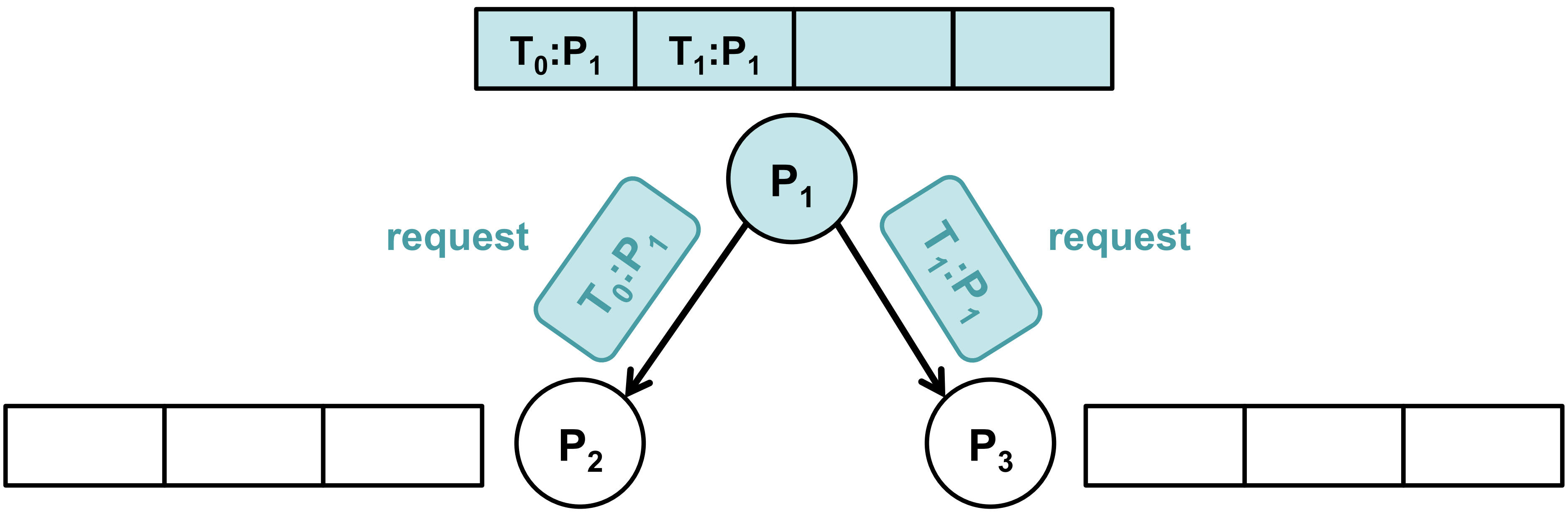
Total Ordering of Events

- **Step 1: P_i Sends Request Resource**
  - **P_i** sends **Request T_m:P_i** to **P_j**
  - **P_i** puts **Request T_m:P_i** on its request queue



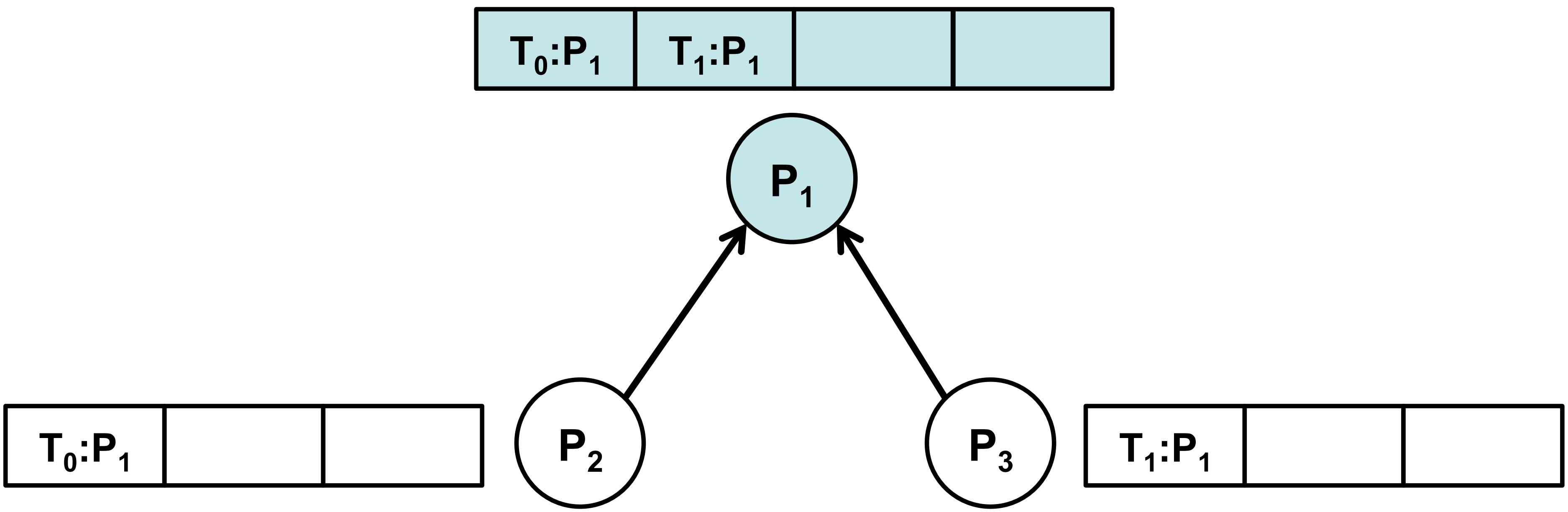
Total Ordering of Events

- **Step 1: P_i Sends Request Resource**
  - P_i sends **Request T_m:P_i** to P_j
  - P_i puts **Request T_m:P_i** on its request queue



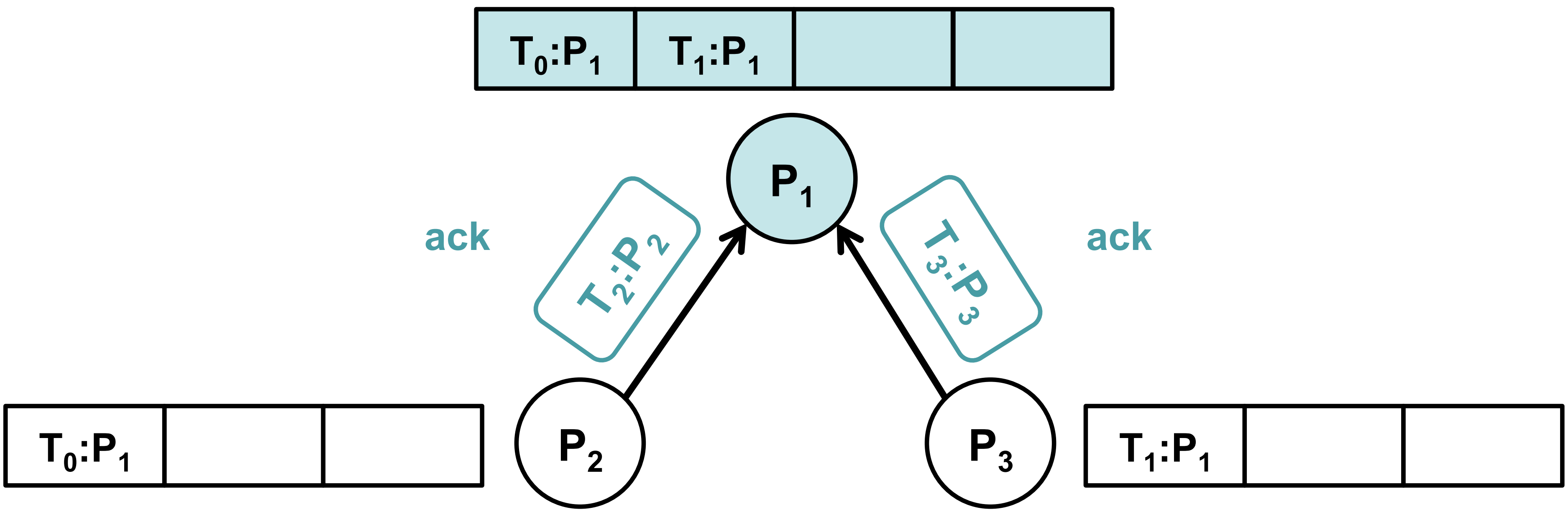
Total Ordering of Events

- **Step 2: P_j Adds Message**
  - **P_j** puts **Request T_m:P_i** on its request queue
  - **P_j** sends **Acknowledgement T_m:P_j** to **P_i**



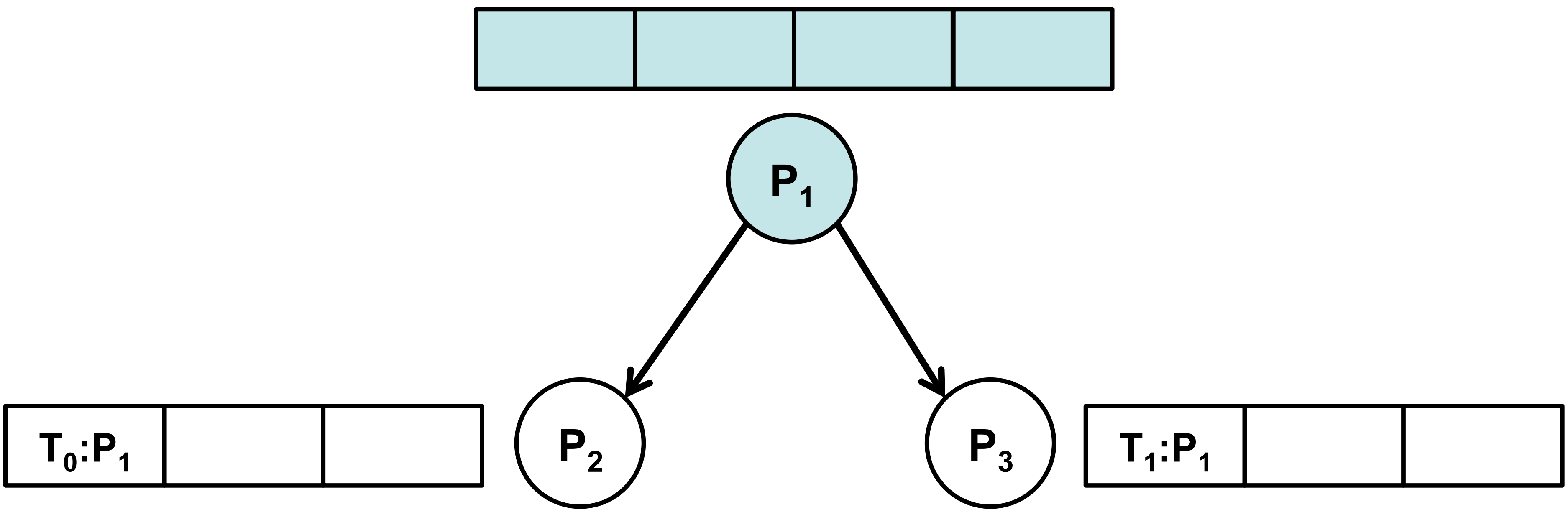
Total Ordering of Events

- **Step 2: P_j Adds Message**
  - P_j puts **Request T_m:P_i** on its request queue
  - P_j sends **Acknowledgement T_m:P_j** to P_i



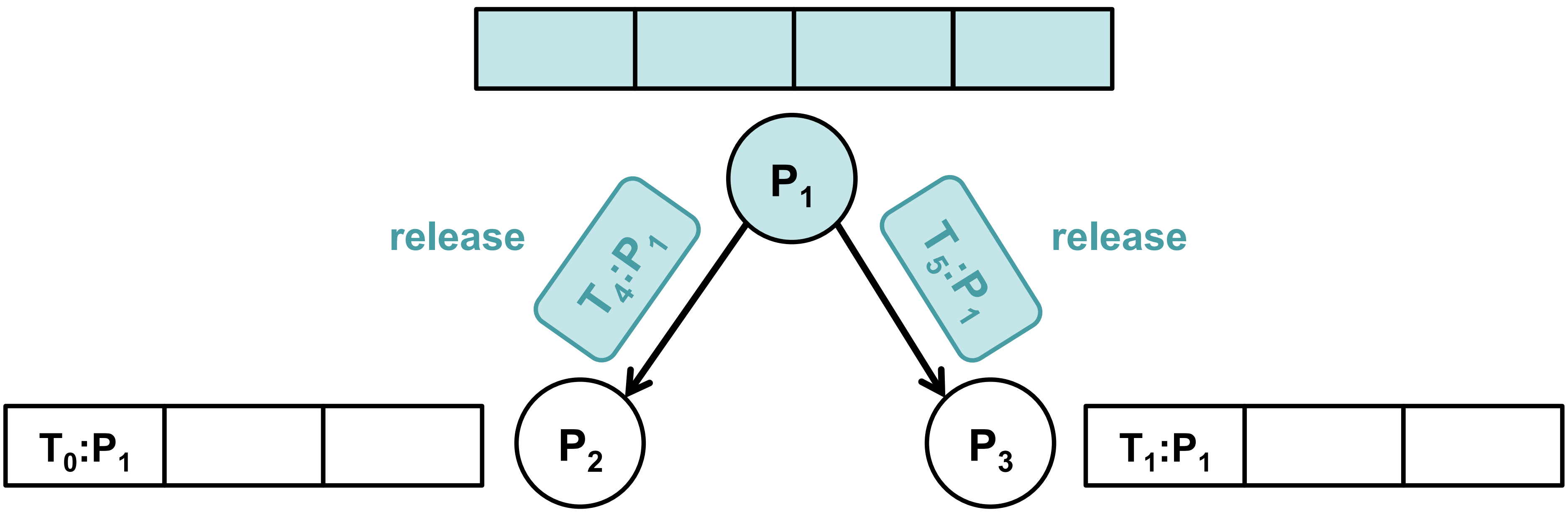
Total Ordering of Events

- **Step 3: P_i Sends Release Resource**
  - **P_i** removes **Request T_m:P_i** from request queue
  - **P_i** sends **Release T_m:P_i** to each **P_j**



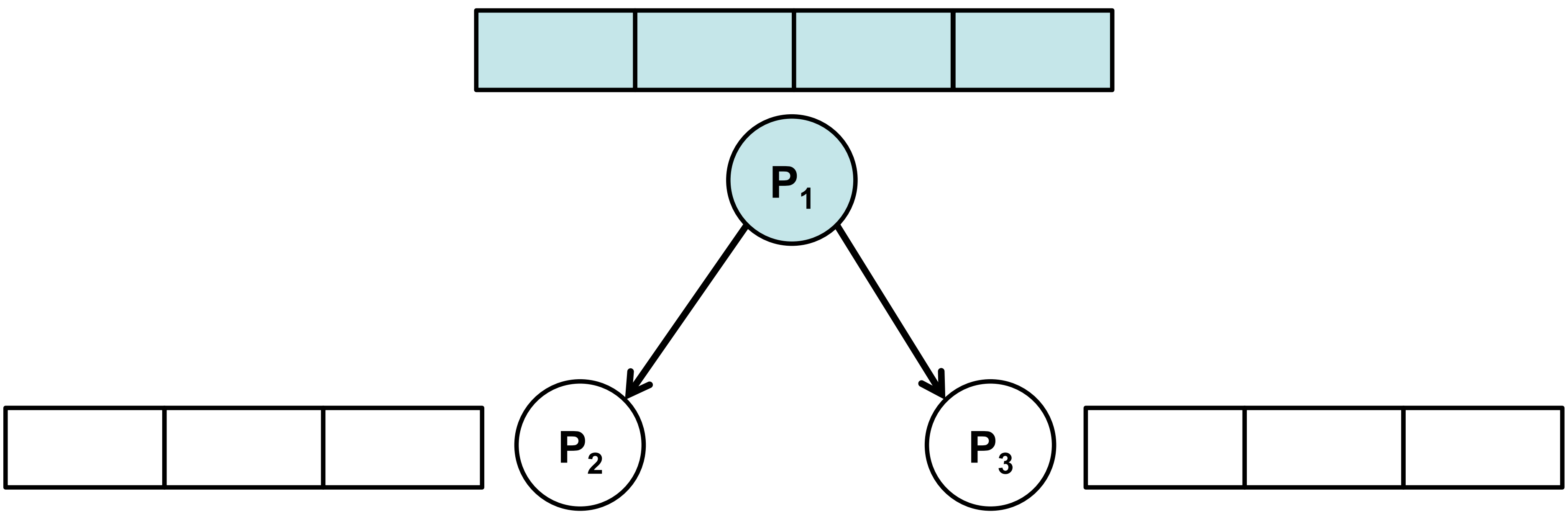
Total Ordering of Events

- **Step 3: P_i Sends Release Resource**
  - P_i removes **Request T_m:P_i** from request queue
  - P_i sends **Release T_m:P_i** to each P_j



Total Ordering of Events

- **Step 4: P_j Removes Message**
  - P_j receives **Release T_m:P_i** from **P_i**
  - P_j removes **Request T_m:P_i** from request queue



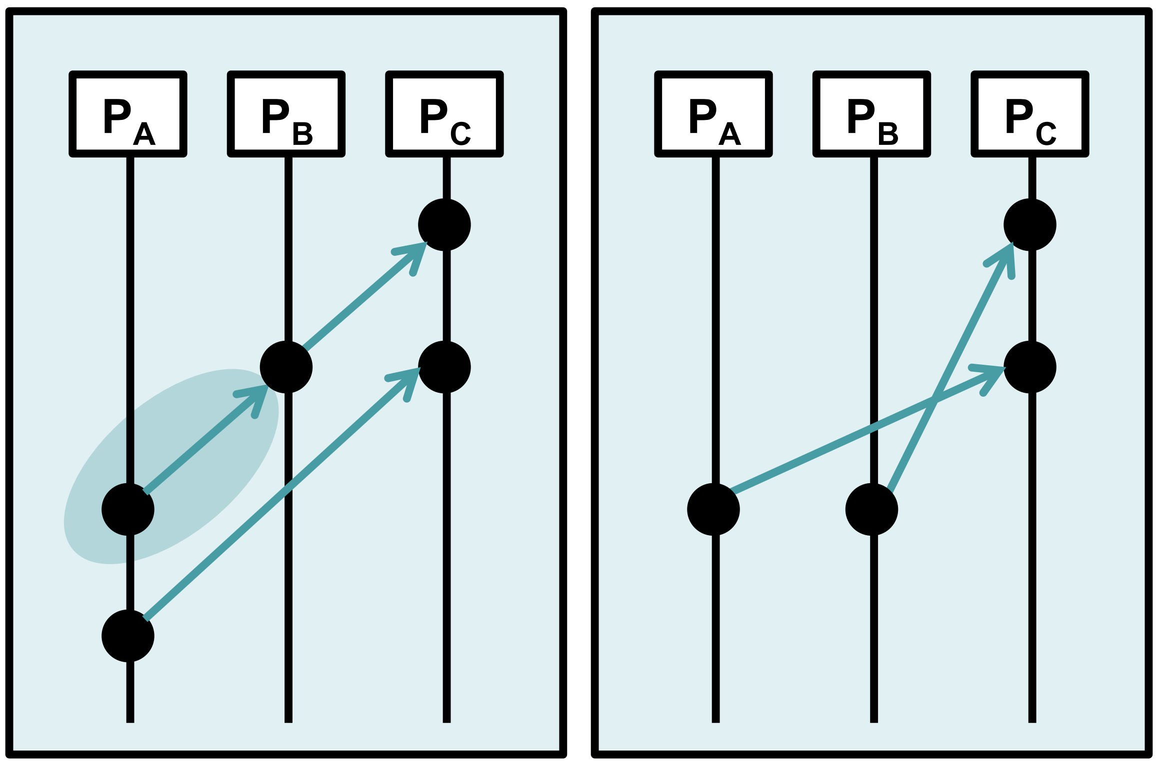
Outline

- Partial Ordering of Events
- Logical Clocks
- Total Ordering of Events
- **Anomalous Behavior**
- Physical Clocks

Anomalous Behavior

- **Discrepancy between universe/system**
  - Event sets and "happens before" relations

Universal
Event Set
S: **a→b**



System
Event Set
S: **a\→b**
and **b\→a**

Time, Clocks, and the Ordering of Events in a Distributed System

Anomalous Behavior

- **Strong Clock Condition**
  - For events **a** and **b** in system event set **S**
  - If **a➔b**, Then **C(a)<C(b)**
  - Attainable via physical clocks

Outline

- Partial Ordering of Events
- Logical Clocks
- Total Ordering of Events
- Anomalous Behavior
- **Physical Clocks**

Physical Clocks

- **Physical Clock $C_i(t)$**
  - **PC1**
    - **$\kappa \ll 1$** for all i: **$| dC_i(t)/dt - 1 | < \kappa$**
  - **PC2**
    - **ϵ** for all i,j: **$| C_i(t) - C_j(t) | < \epsilon$**
  - **Also**
    - **$\mu <$** smallest transmission time

Physical Clocks

- **Prevent anomalous behavior**
  - Must ensure that $\mathbf{C_j(t) < C_i(t+\mu)}$
  - How small must κ and ϵ be? $\boldsymbol{\epsilon/(1-\kappa) < \mu}$

Discussion

- **Partial ordering**
- **Total ordering**
- **Anomalous Behavior**
- **Physical clocks**

Conclusions

- **Coordination of Distributed Systems**
- **Partial Ordering of Events**
- **Total Ordering of Events**

Distributed Snapshots: Determining Global States of Distributed Systems

K. Mani Chandy
University of Texas at Austin

Leslie Lamport
Stanford Research Institute

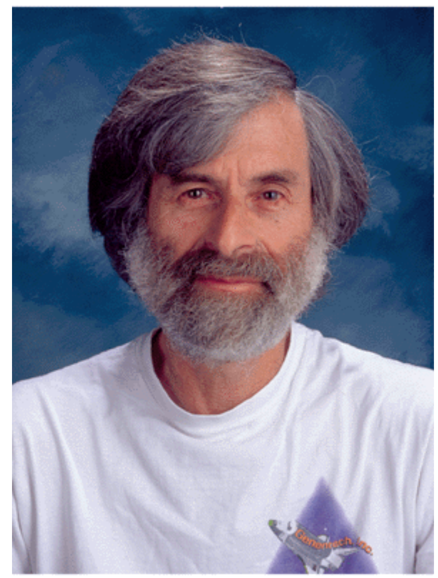
About the Authors

K. Mani Chandy

University of Texas at Austin

Leslie Lamport

Stanford Research Institute



Introduction

- **Panoramic dynamic scene**
  – Cannot capture with single snapshot
  – Must piece together multiple snapshots
- **Questions**
  – How should snapshots be taken?
  – What criteria must overall picture satisfy?

Introduction

- **Process can record its own state**
- **States of all processes form global state**
- **Record valid global system state**
- **Detect stable properties**
  - **y(S) = true** implies
  - **y(**all states reachable from **S) = true**

Outline

- **Distributed System Model**
- Global State Detection Algorithm
- Recorded Global State Properties
- Stability Detection

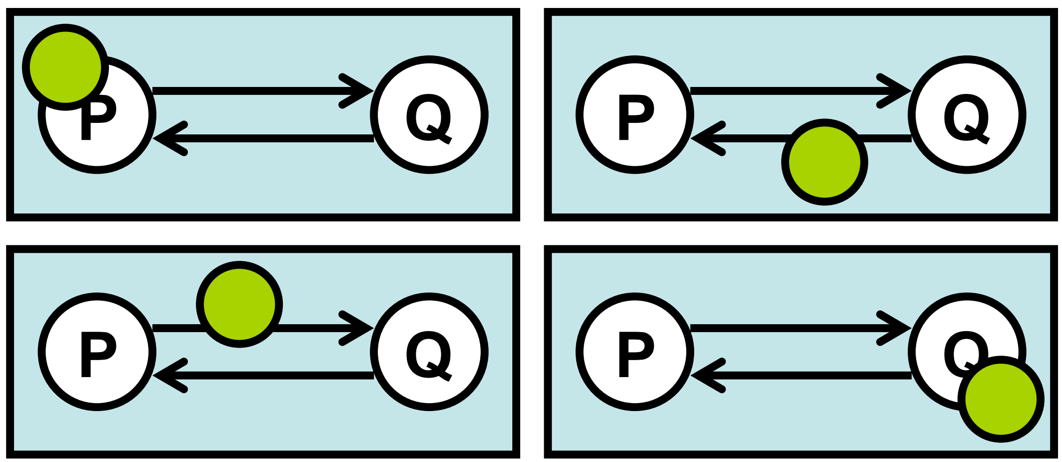
Distributed System Model

- **Process**
  - State(t)
  - Event(t)
- **Channel**
  - MessagesSent(t)
  - Event(t)
- **Event**
  - Process P
    - State S before
    - State S' after
  - Channel C (incoming or outgoing from P)
    - Messages (received by P or sent from P)

Distributed System Model

- **Example 1: Single Token System**

States
Global: in-P
P : sI
C : empty
C': empty
Q : sO



States
Global: in-C'
P : sO
C : empty
C': token
Q : sO

Event
P sends

States
Global: in-C
P : sO
C : token
C': empty
Q : sO

Event
Q sends

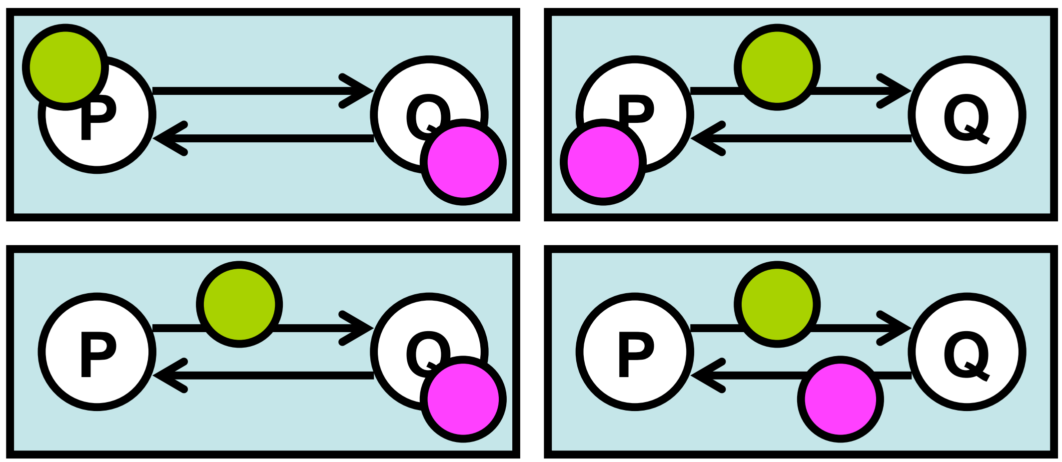
States
Global: in-Q
P : sO
C : empty
C': empty
Q : sI

Event
Q receives

Distributed System Model

- **Example 2: Nondeterministic System**

States
Global: S0
P : sI
C : empty
C’: empty
Q : sI



States
Global: S3
P : sI
C : M
C’: empty
Q : sO

Event
P sends
M

States
Global: S1
P : sO
C : M
C’: empty
Q : sI

Event
Q sends
M’

States
Global: S2
P : sO
C : M
C’: M’
Q : sO

Event
P receives
M’

Outline

- Distributed System Model
- **Global State Detection Algorithm**
- Recorded Global State Properties
- Stability Detection

Algorithm

- **Marker Sending Rule**
  - P records its state
  - P sends marker along each outgoing channel C
- **Marker Receiving Rule**
  - Q receives a marker along incoming channel C
  - If Q has not recorded its state
    - Q records its state
  - Else
    - Q records C's state as a sequence of messages

Algorithm

- **Example: Process P Obtains Global State from Process Q**
  - Q receives P's marker along channel C
  - Q records its state
  - Computation
  - Q receives P's marker along channel C
  - Q records C's state

Outline

- Distributed System Model
- Global State Detection Algorithm
- **Recorded Global State Properties**
- Stability Detection

Recorded Global State Properties

- **Markers produce concurrent subsequence**
  - S* may not actually exist
  - S* from combination of concurrent events
- **No effect on preceding/following events**
  - S* reachable from S_i
  - S_o reachable from S*

Recorded Global State Properties

- **Theorem 1: Exists Computation $\{e_0' \ldots e_n'\}$**
  - **Events**
    - $\{ e_0' \ldots e_i -1' \}$ is equivalent to $\{ e_0 \ldots e_i -1 \}$
    - $\{ e_i' \ldots e_o -1' \}$ is a permutation of $\{ e_i \ldots e_o -1 \}$
    - $\{ e_o' \ldots e_n' \}$ is equivalent to $\{ e_o \ldots en \}$
  - **States**
    - $\{ S_0' \ldots S_i' \}$ is equivalent to $\{ S_0 \ldots S_i \}$
    - For some k, where $i<k<o$, $S_k' = S^*$
    - $\{ S_o' \ldots S_n' \}$ is equivalent to $\{ S_o \ldots S_n \}$

Outline

- Distributed System Model
- Global State Detection Algorithm
- Recorded Global State Properties
- **Stability Detection**

Stability Detection

- **Algorithm**
  - **Initialize:** definite=false, $y(S_i)$=definite
  - **Repeat:** record S*, definite=y(S*)
- **Implications of "definite"**
  - **definite == false:** no stable property at start
  - **definite == true:** stable property at termination
- **Correctness**
  - S_i can lead to S*, S* can lead to S_o
  - for all j: $y(S_j) = y(S_j+1)$

Discussion

- Partial Ordering
- Recorded Global State
- Global State Detection Algorithm
- Stable Property Detection

Conclusions

- **Processes form recorded global state**
  – Record its own state
  – Piece together multiple records
- **Questions addressed**
  – How should the snapshots be taken?
  – What criteria must overall picture satisfy?