Distributed Breakpoints

Presented by

Gokul Nadathur

References

- Breakpoints and Halting in Distributed Programs, B.P.Miller,J.D.Choi. *In Proc. 8th IEEE Int. Conf. On Distributed Computing systems, San Jose , July 1988.*
- Detecting Relational Global Predicates in Distributed Systems,A.I.Tomlinson, V.K.Garg. *In ACM SIGPLAN , 1993*
- Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms, O.Babaoglu, K.Marzullo. *In S.Mullender, editor, Distributed Systems 2nd edition, ACM press, Frontier Series, 1993*

Outline

- Halting of Distributed Computations
- Distributed Breakpoints and Predicates
- Detection of certain types of Predicates

Introduction

- A distributed computation can be recorded
- Next Step - Halting a distributed computation
- Halting when certain specific conditions are satisfied
- Specification of conditions in the form of predicates
- Detection of predicates when they occur

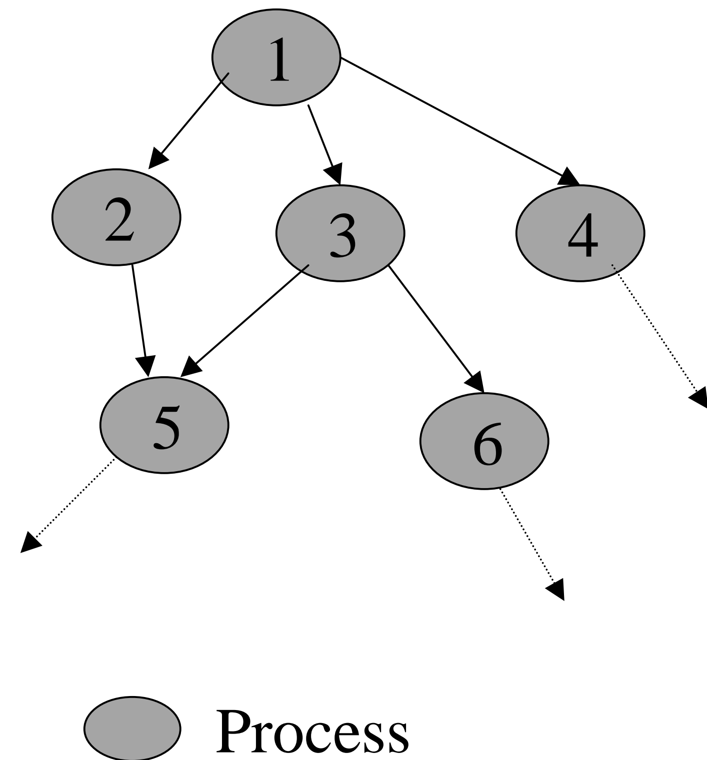
Need for a Halting Algorithm

- For Debugging : We need to ...
  - specify conditions
  - halt when those conditions are satisfied
  - examine the state of the computation

Halting Algorithm

Top level view of Halting algorithm

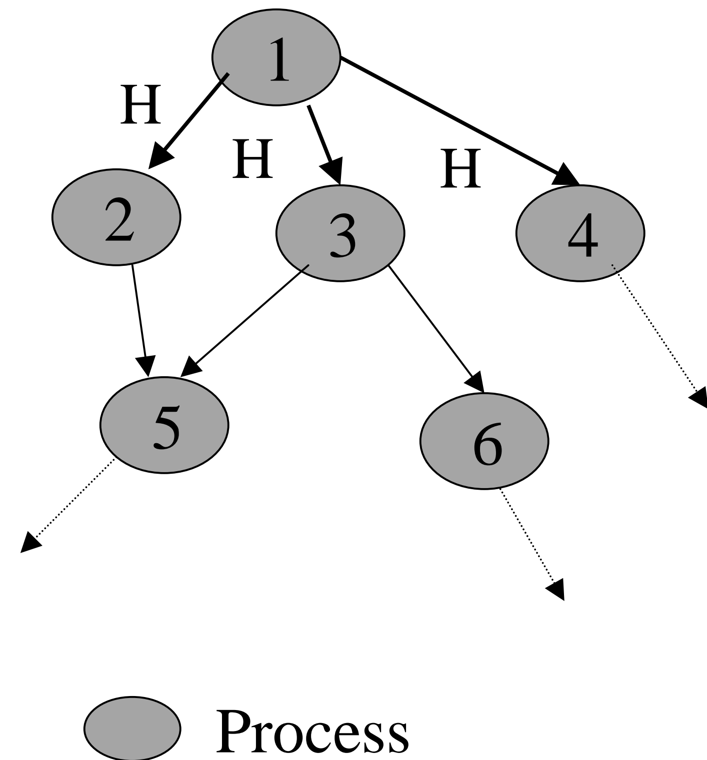
- Belongs to a class of algorithms called diffusion algorithms
- Extension of Lamport's snapshot algorithm



Halting Algorithm

- Belongs to a class of algorithms called diffusion algorithms
- Extension of Lamport's snapshot algorithm

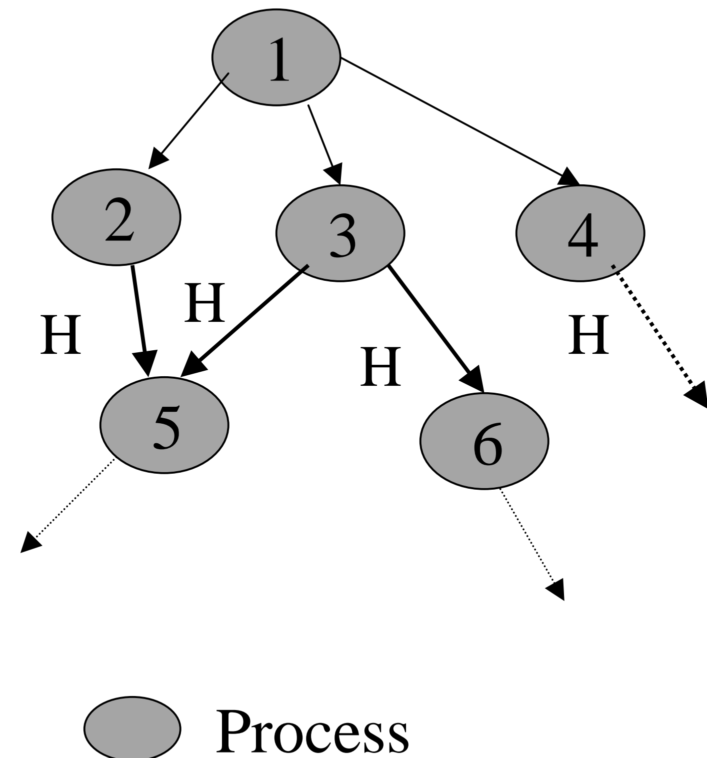
Top level view of halting algorithm



Halting Algorithm

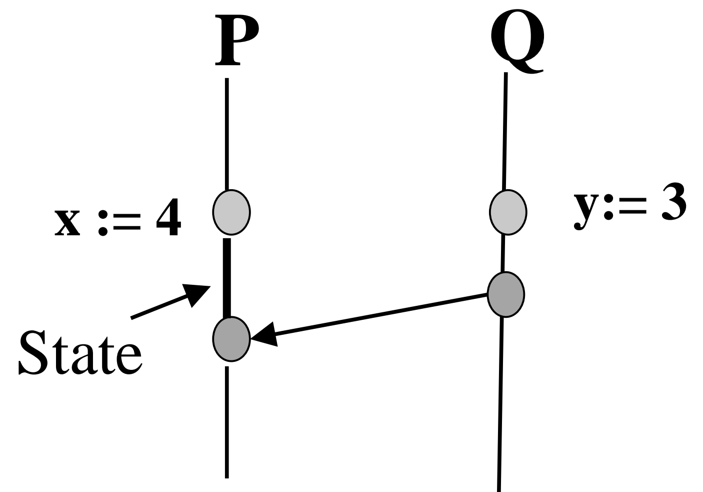
Top level view of Halting algorithm

- Belongs to a class of algorithms called diffusion algorithms
- Extension of Lamport's snapshot algorithm



Events and States

- Event : Anything interesting
  - Local Event : events whose effect is restricted to the process in which they occur. E.g. simple assignment statements
  - External event : Sending or receiving of messages . Interactions with other processes
- States : Every event triggers a new state

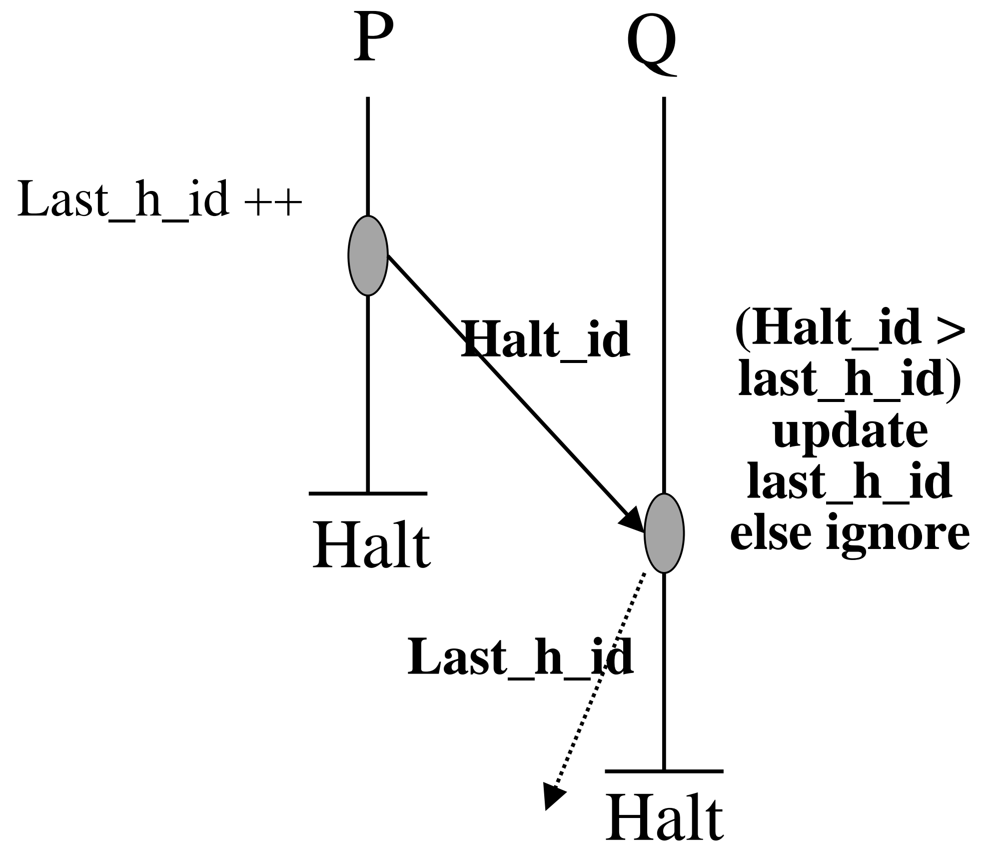


Local Events

External Events

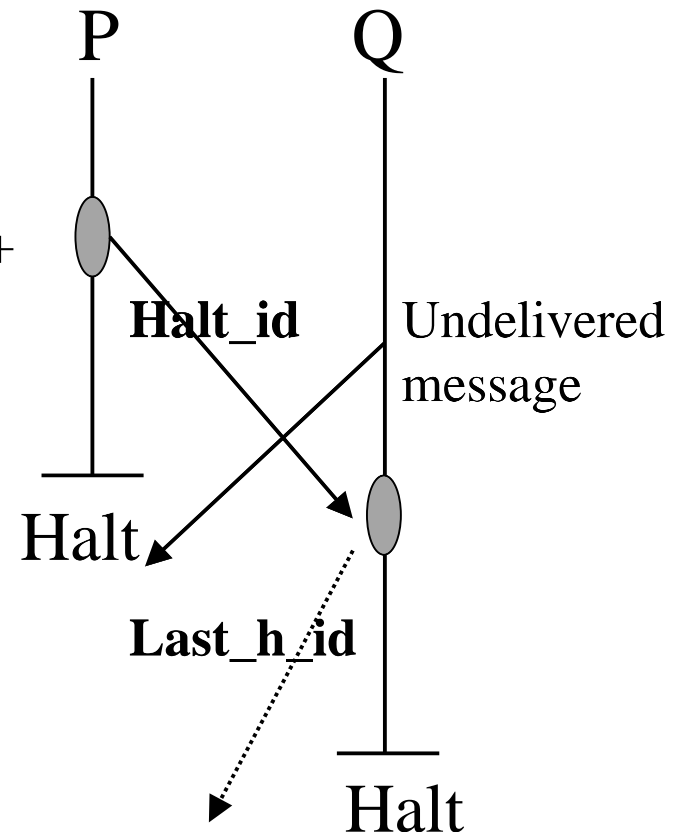
Halting Algorithm

- Extension of Snapshot algorithm
- Uses markers with sequence numbers (Halt markers)
  - Latest Sequence number recorded as last_halt_id



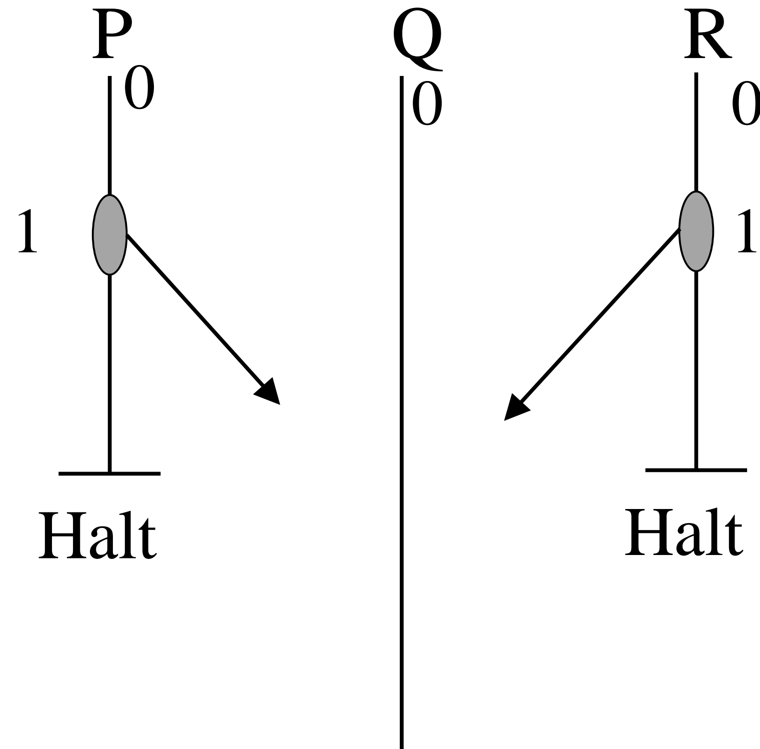
Halting Algorithm ...

- When all processes have finally halted,
  - state of each process is preserved
  - outgoing channels contains undelivered messages with halt marker at the last



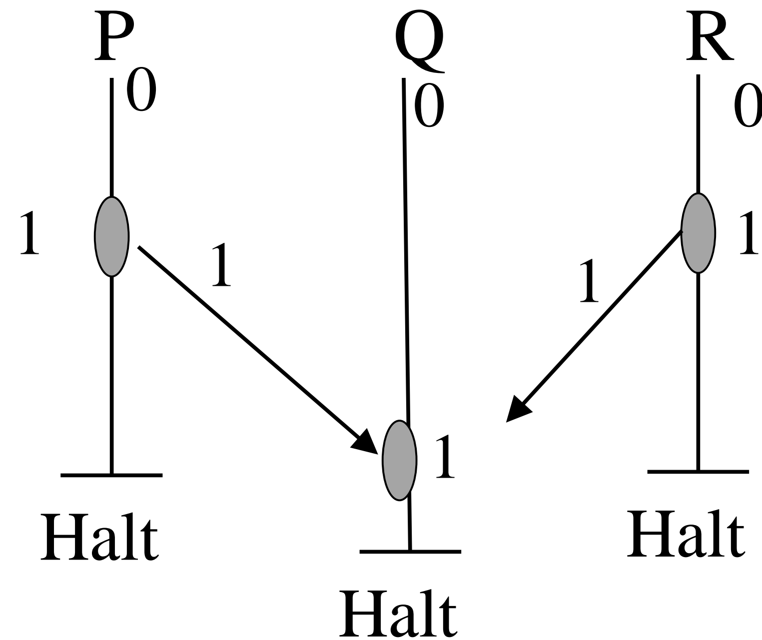
Halting Algorithm ...

- Multiple Processes can initiate halting independently



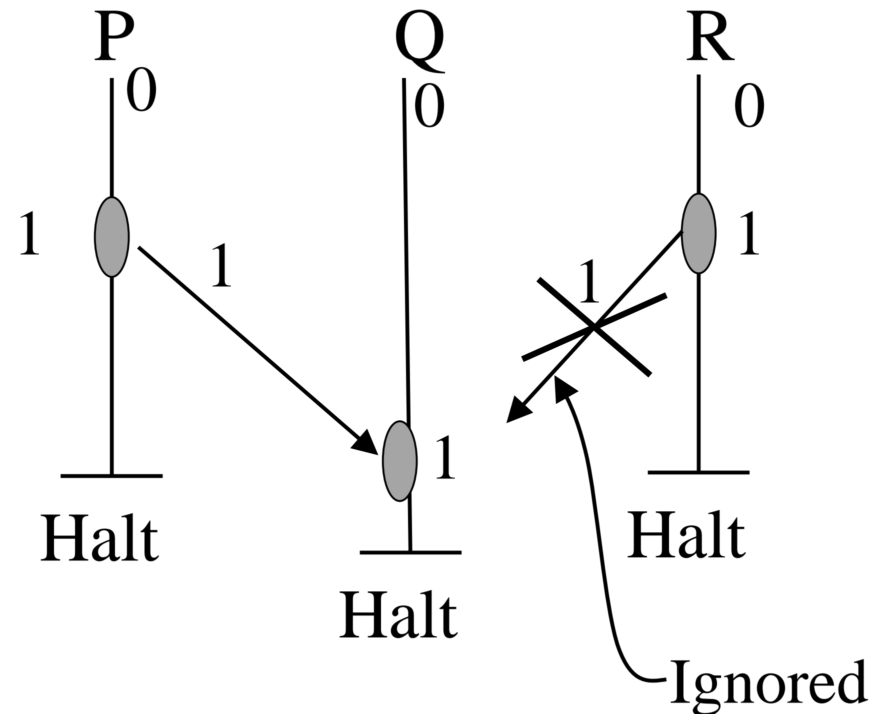
Halting Algorithm ...

- Multiple Processes can initiate halting independently



Halting Algorithm ...

- Multiple Processes can initiate halting independently



Breakpoints and Predicates

- A Distributed computation can be halted
- But we need to *specify* the conditions on which the computation should halt
- When these conditions have become true, we need to *detect* them

Predicates

- Stable Predicates: Once x becomes TRUE it remains true. E.g. Deadlocks
- Detectable by Lamport's Snapshots
- What about unstable predicates ?
  - Predicates whose truth values change with time
- Solution :
  - Detecting Concurrent states

Types of Predicates

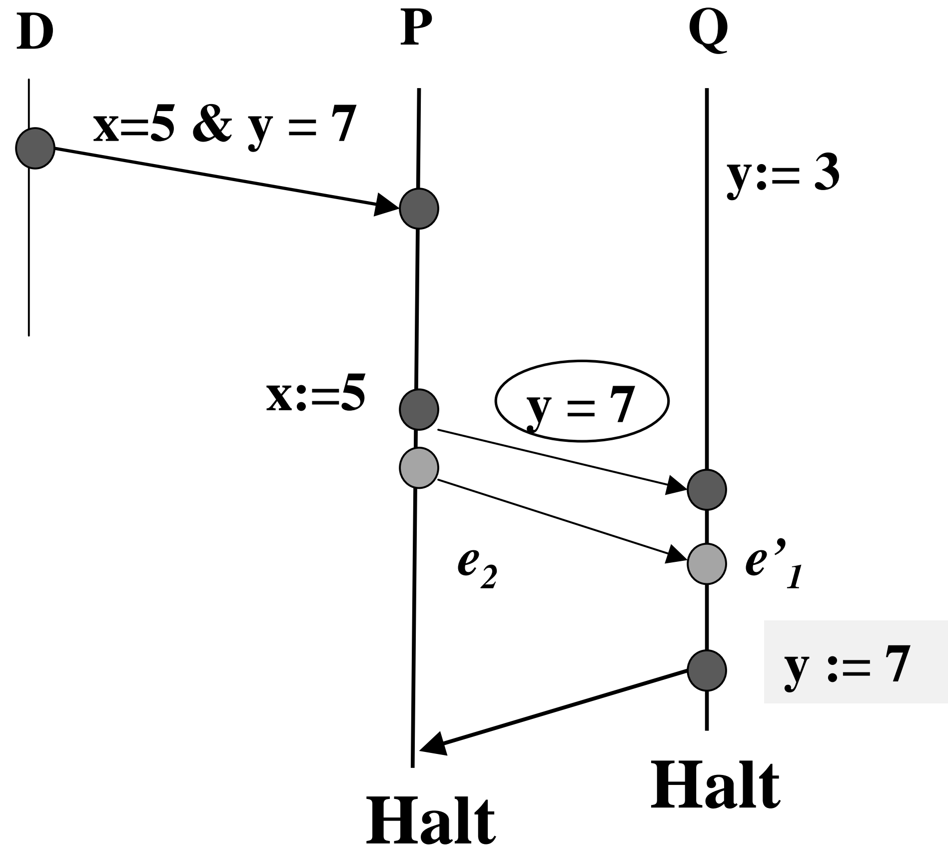
- Simple Predicates : Predicate that is local to a process
- Disjunctive Predicates : SP_1 OR SP_2 OR …
- Linked Predicates :
  - Predicates used to describe sequence of events
  - Order of events is important
- Conjunctive Predicates : SP_1 AND SP_2 AND…
- Relational Predicates :
  - Predicates of the form $x + y < C$ where x is in process P and y is in Q

Detection of Predicates

- Simple Predicates : Detected by local debugger
- Disjunctive Predicates :
  - Each process detects its simple predicate
  - Even if 1 process detects its simple predicate , the disjunctive predicate is successfully detected
- Linked Predicates :
  - A sequence of predicates in a specific order need to be satisfied for the predicate to be detected

Linked Predicate detection

- Debugger process sends predicate markers
- Predicate markers contain unevaluated part of linked predicate

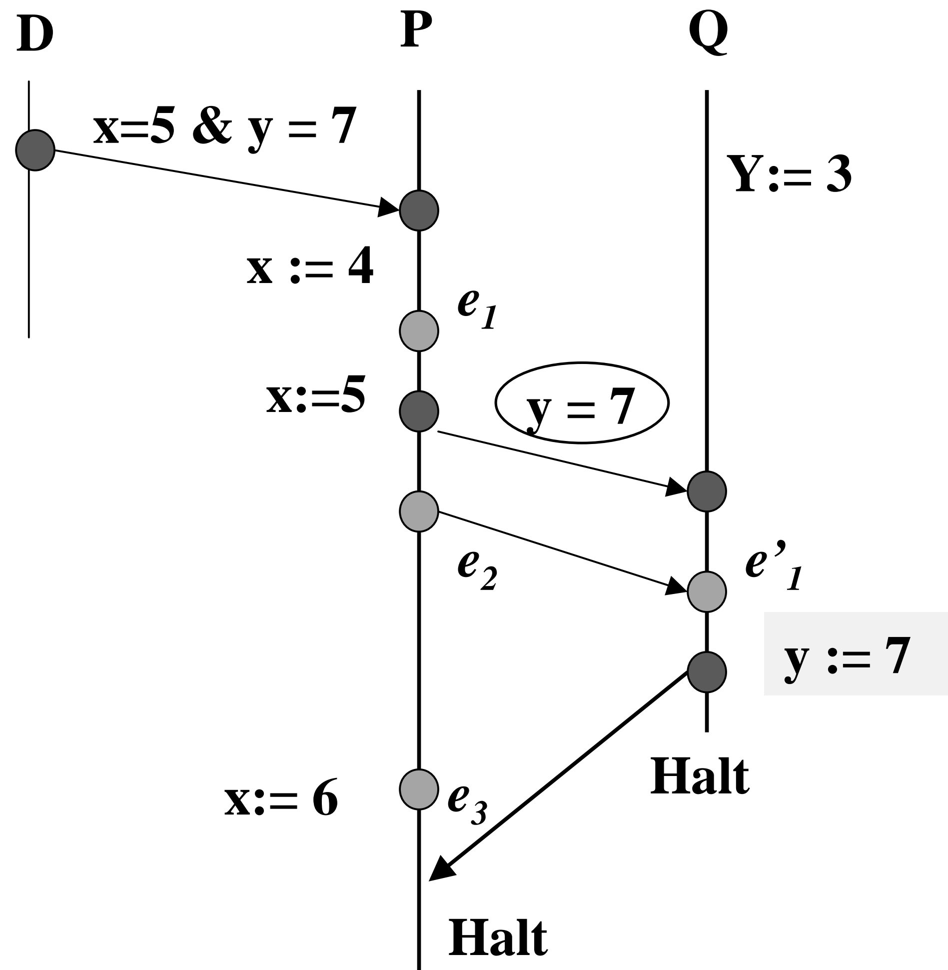


assignment events

Application events

Problem

- Algorithm guarantees that the predicate is detected
- but the halted state may or may not reflect that

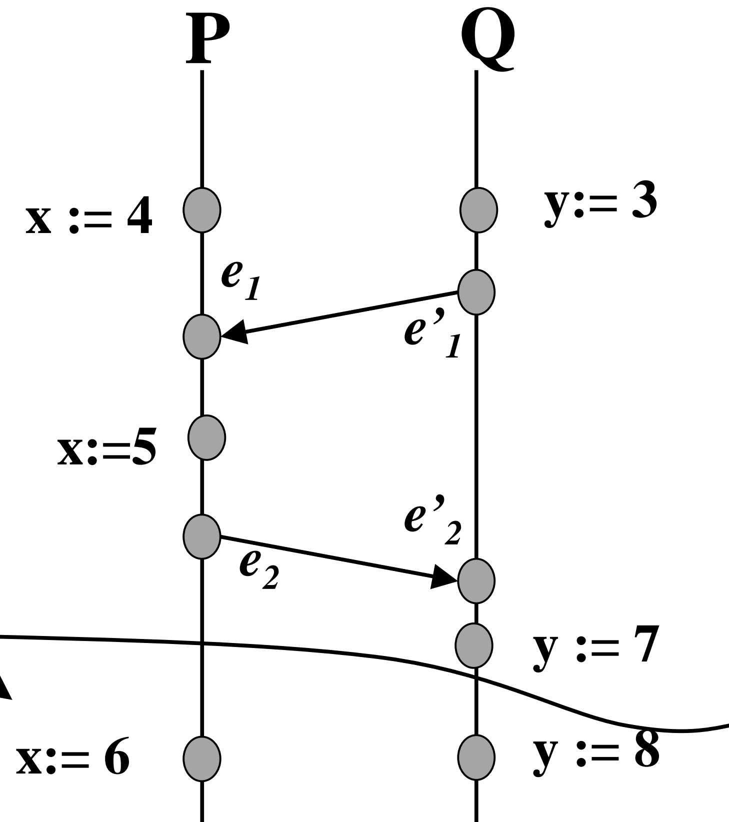


Debug events

Application events

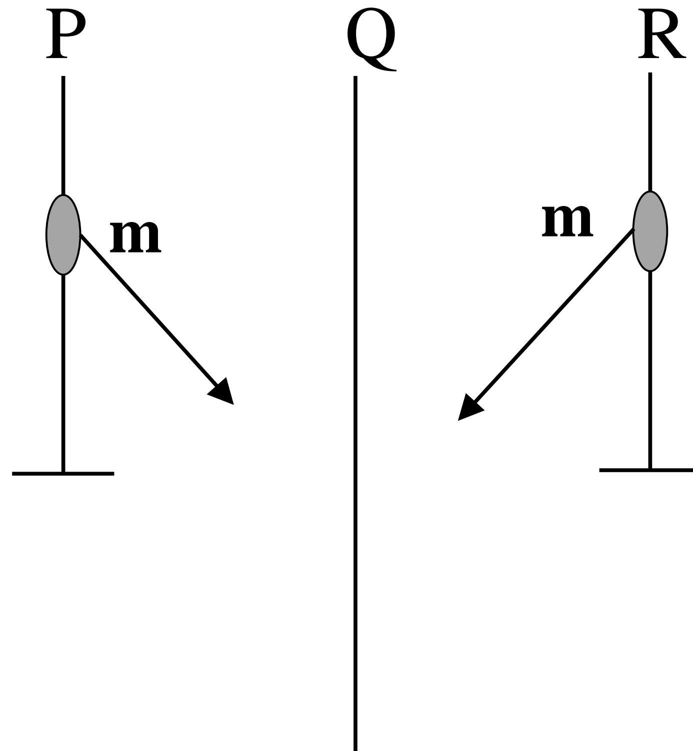
Conjunctive Predicates

- "and" is implicitly means "at the same time"
- E.g. (x=5 & y = 7)



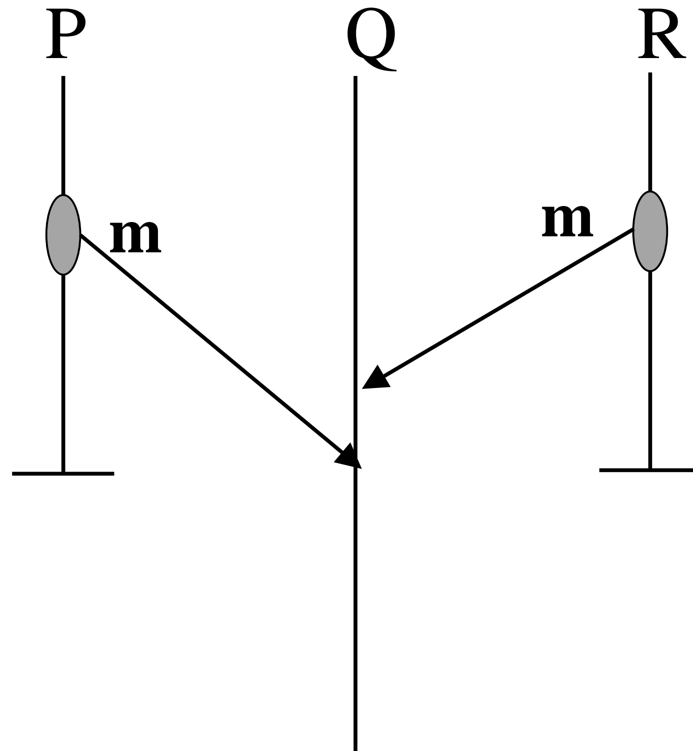
Motivation

- Is detection of conjunctive predicates worthwhile ?
- Race Detection



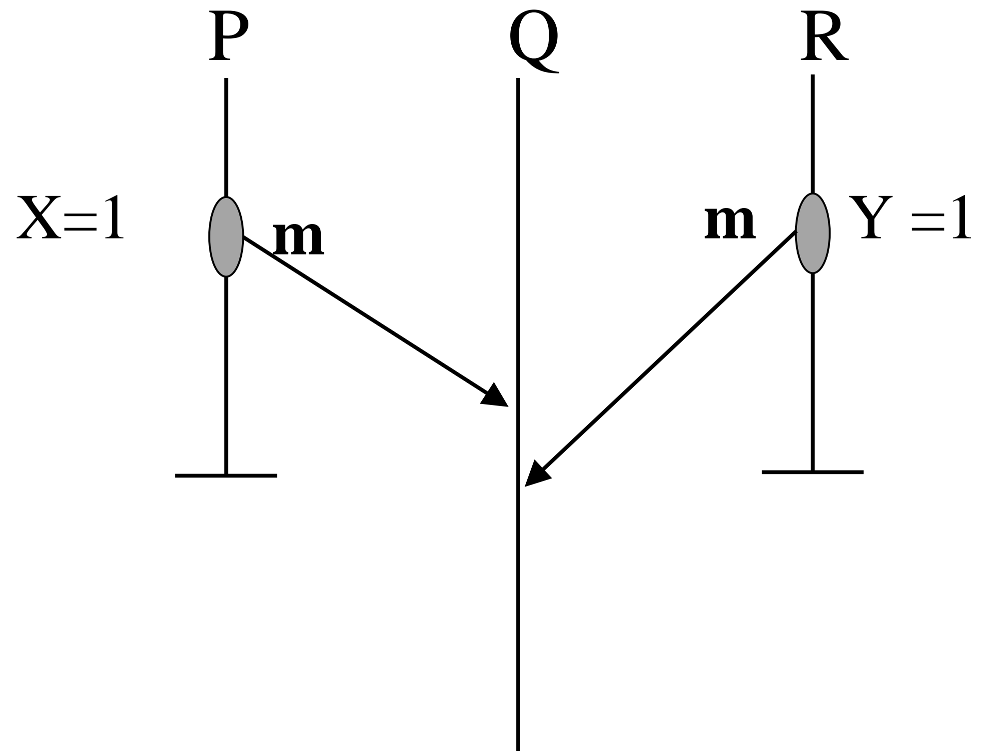
Motivation

- Is detection of conjunctive predicates worthwhile ?
- Race Detection



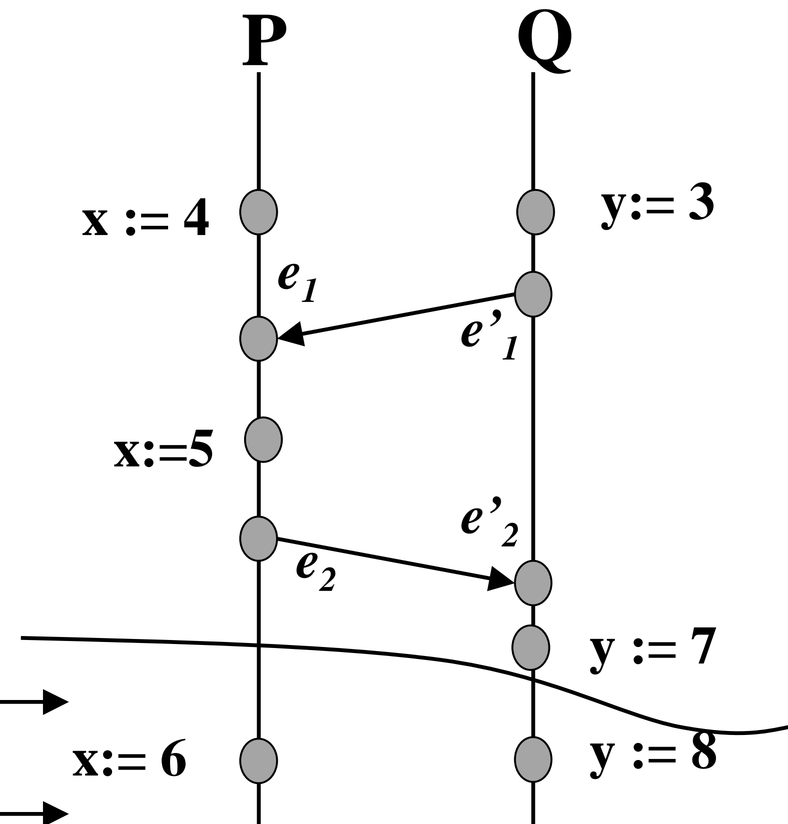
Motivation

- Is detection of conjunctive predicates worthwhile ?
- Race Detection
  (x=1 & y=1) implies a race



Conjunctive Predicates

- "and" is implicitly means "at the same time"
- Can be partitioned into ordered and unordered predicates



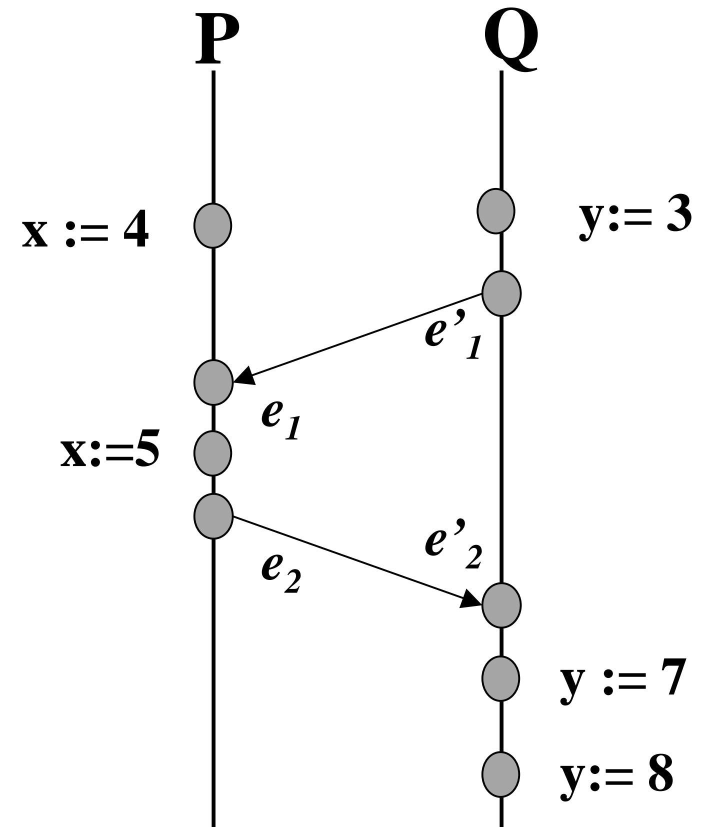
(x=5 & y = 7) Ordered

(x=6 & y=8) Unordered

Ordering in Conjunctive Predicates

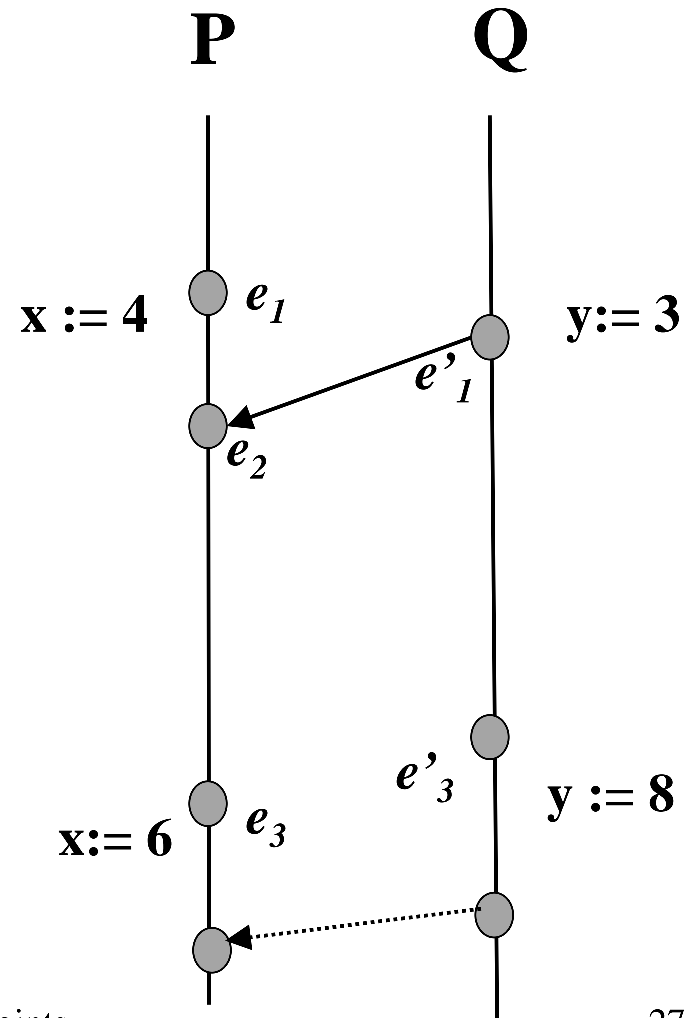
- E.g. x=5 and y=7
- **Ordering** according to ***"happened before"***
- ordered conjunctive predicates can be expressed as **linked predicates**

  (x = 5) ⟶ (y=7)
- Mechanism for detecting linked predicates



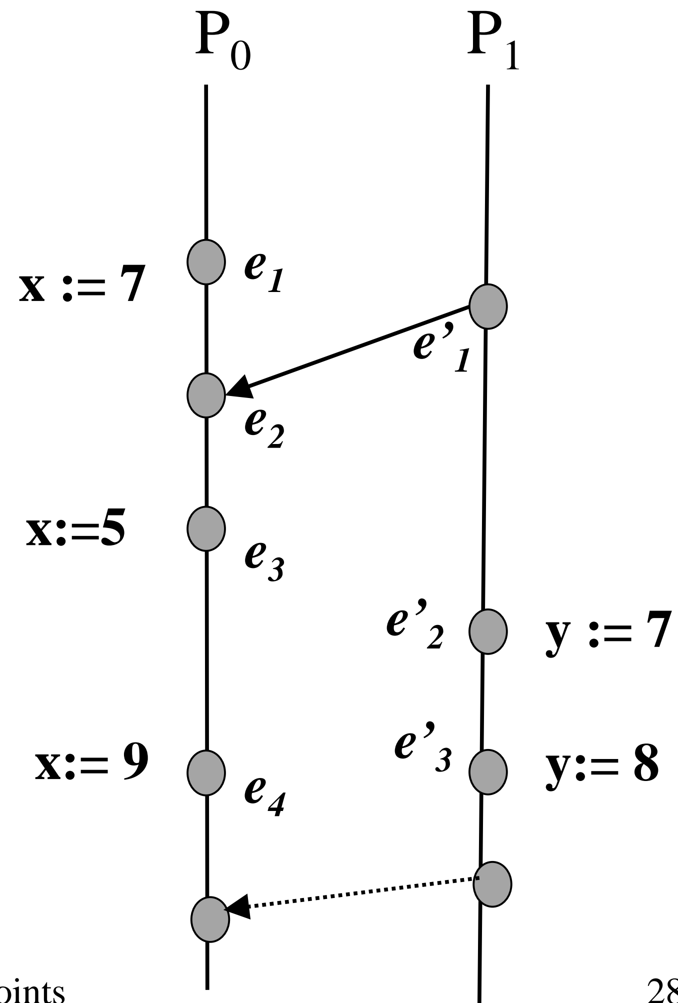
Unordered Conjunctive Predicates

- E.g. x=6 and y=8
- No Ordering even in virtual time
- Cannot be expressed as a sequence of events
- Look for a common state that disrupts this concurrency



Relational Predicates

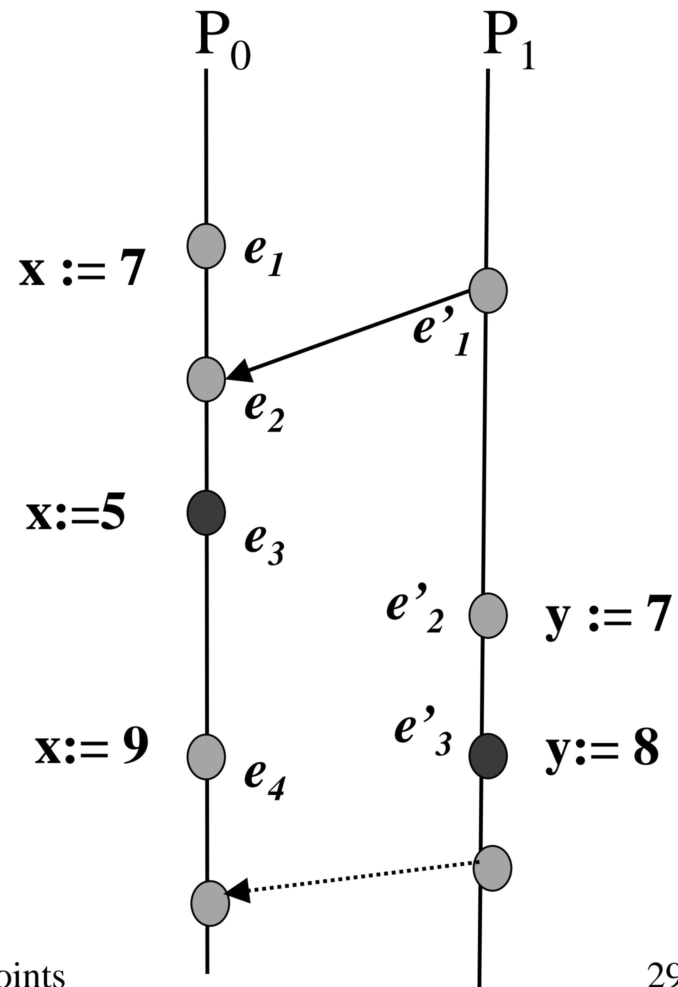
- Predicates of type
  $x + y < C$
- E.g. $x + y < 15$
- $<$:
  - increases the domain to be monitored
  - implies concurrency
  - assumes a centralized reference points
- Generalization of Conjunctive predicates



Relational Predicates

- Predicates of type

  $x + y < C$
- E.g. $x + y < 15$
- < :
  - increases the domain to be monitored
  - implies concurrency
  - assumes a centralized reference points
- Generalization of Conjunctive predicates

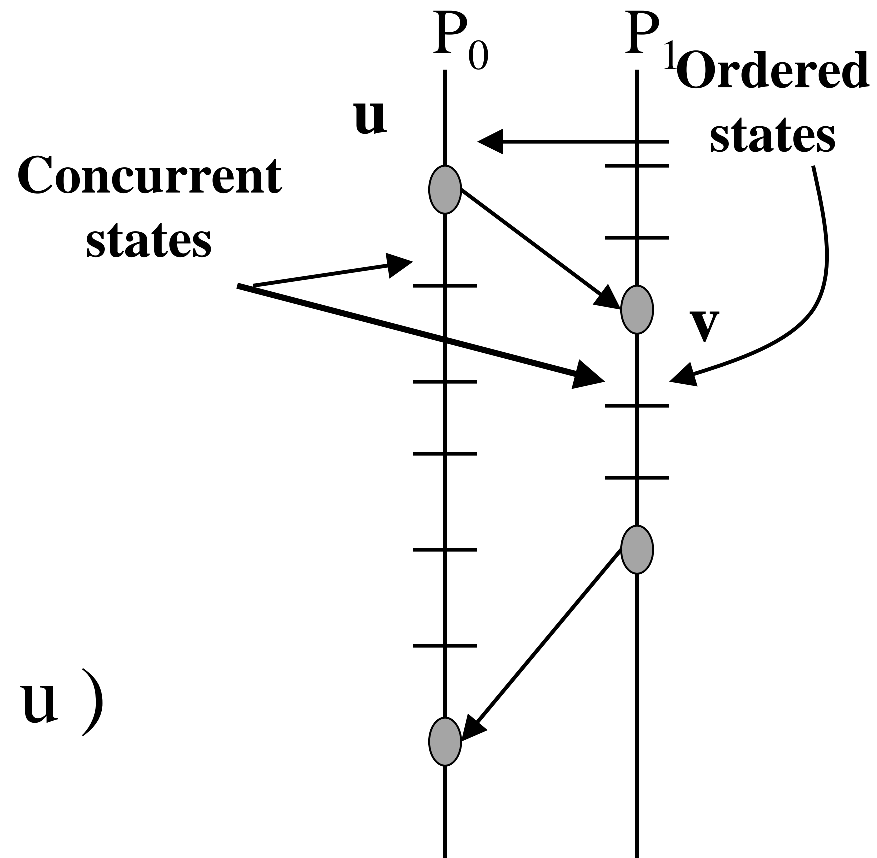


Ordered and Concurrent States

- Ordered states : pair of states for which $\rightarrow$ holds good
- If $(u \rightarrow v)$ then
  - $\max(u,v) = v$
  - $\min(u,v) = u$
- Concurrent states:

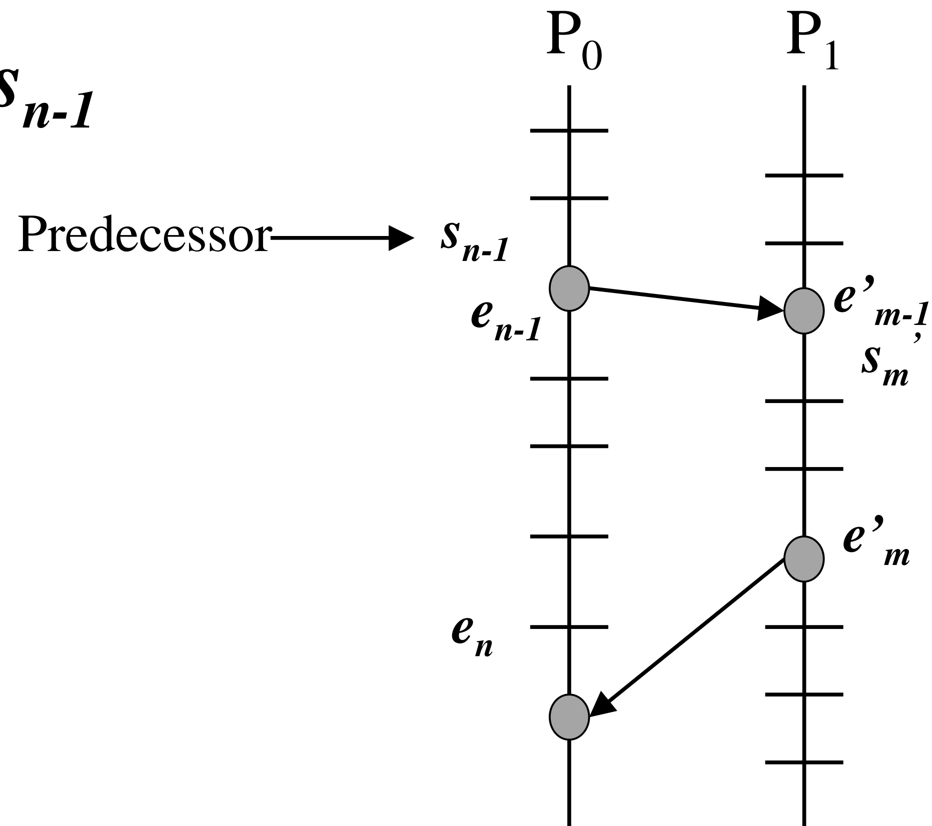
  if $(u \not\rightarrow v$ and $v \not\rightarrow u)$

  $u \parallel v$



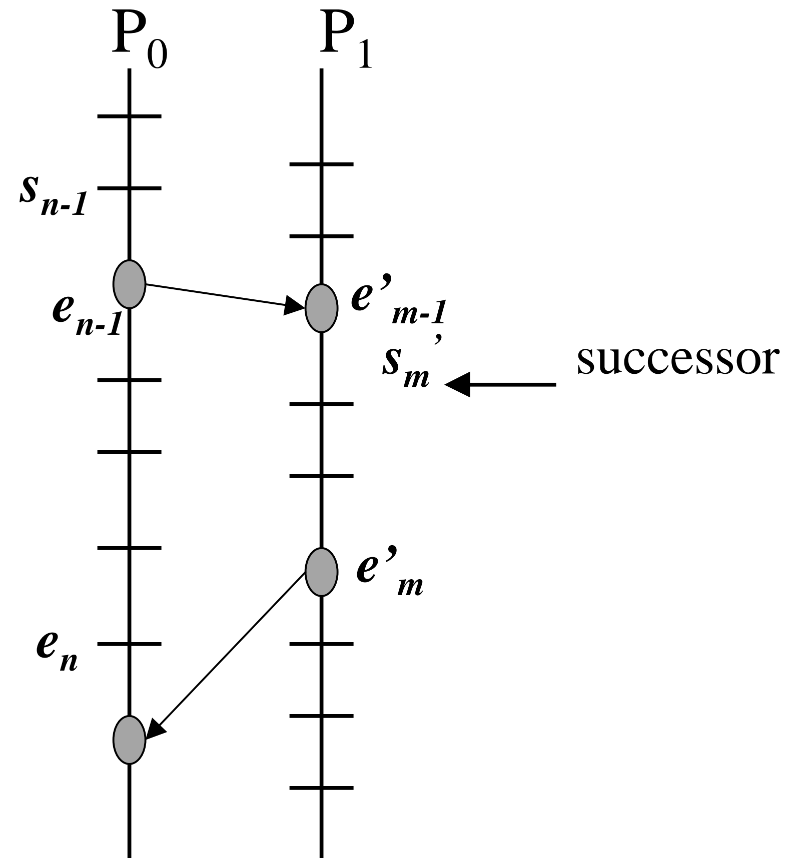
Predecessor and Successor

- $\text{Pred}(s_m{}', P_1) = s_{n-1}$



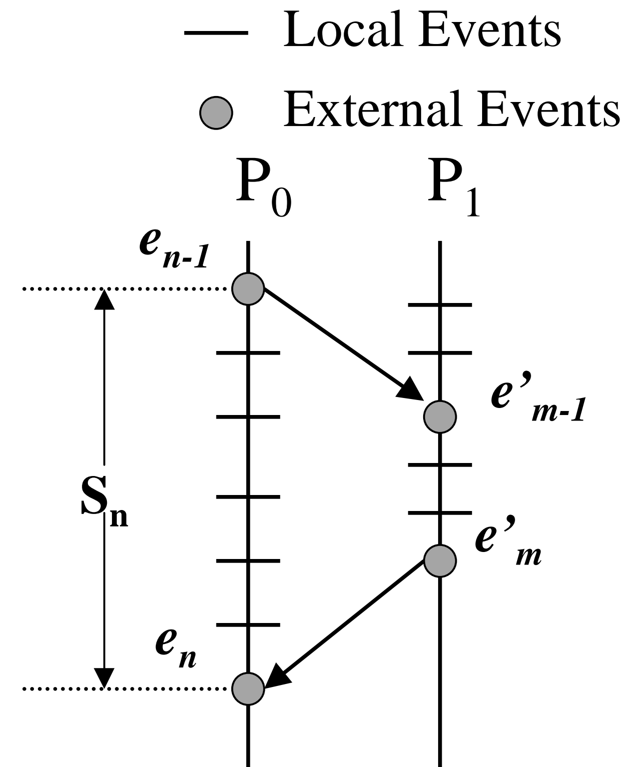
Predecessor and Successor

- $\text{Pred}(\boldsymbol{s_m}', \text{P}_1) = \boldsymbol{s_{n-1}}$
- $\text{Succ}(\boldsymbol{s_{n-1}}, \text{P}_0) = \boldsymbol{s_m}'$



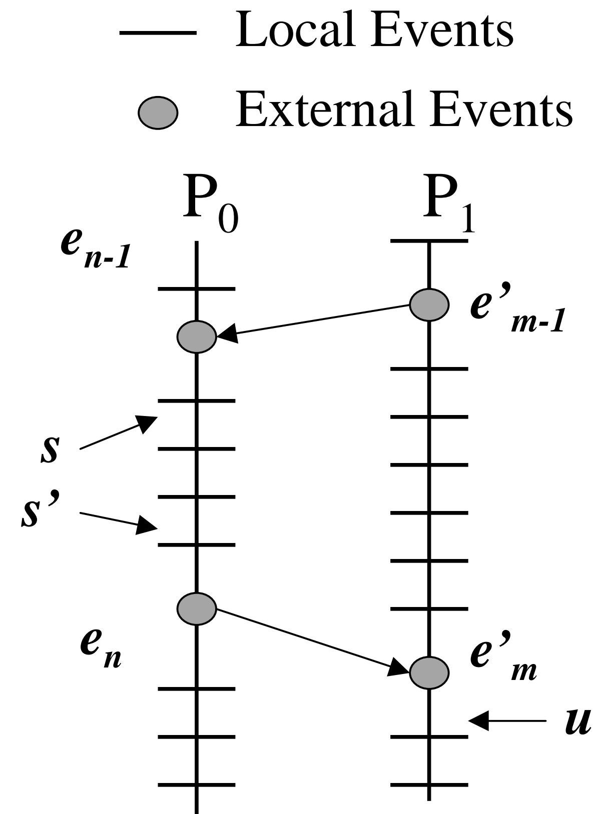
States and State Intervals

- External event - Sending or receiving of messages
- Every external event triggers a new state interval



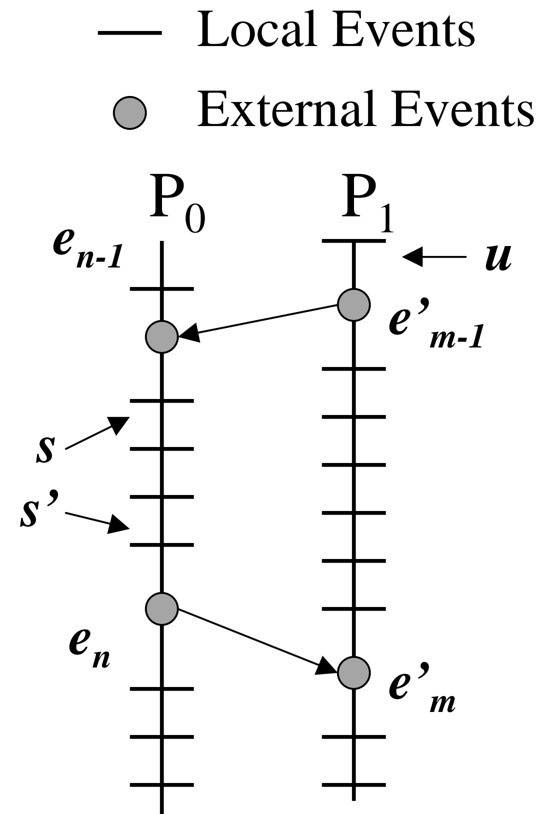
Congruence

- s and s' are congruent with respect to $\longrightarrow$
- s $\rightarrow$ u implies s' $\rightarrow$ u



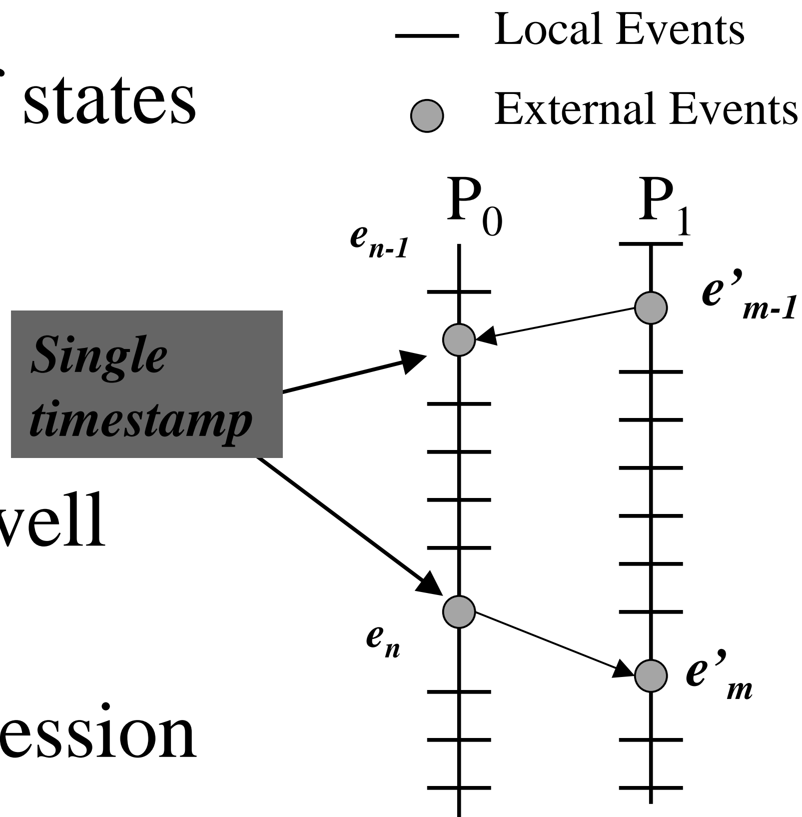
Congruence

- s and s' are congruent with respect to $\longrightarrow$
- s $\rightarrow$ u implies s' $\rightarrow$ u and
- u $\rightarrow$ s implies u $\rightarrow$ s'



Why congruence ?

- Can assign a single timestamp to the set of states in the state intreval
- System becomes more tractable
- pred, succ and $||$ are well defined
- Similar to lossy compression

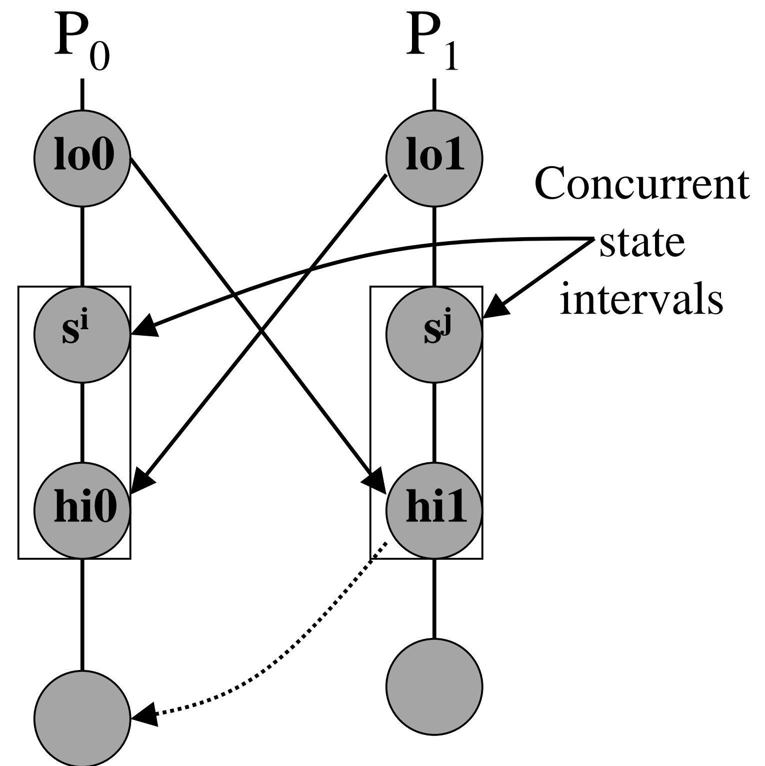


Redefinition in terms of state intervals

- Identification of pair of concurrent state intrevals (s_0^i, s_1^j)
- Maintain the min(x) in each of the state intreval
- Predicate $x_0 + x_1 < C$ is satisfied if
  - $x_0' + x_1' < C$ where x_0' is min(x) in s_0^i
    x_1' is min(x) in s_1^j
- Problem : Detection of concurrent state intrevals

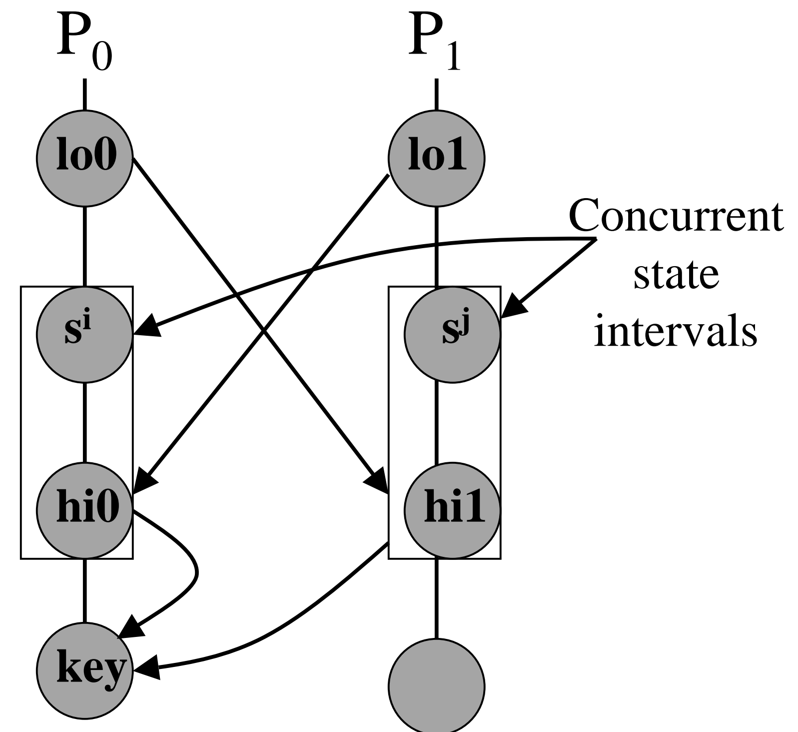
Concurrent state intervals

- Aim : To determine a pair of concurrent state intervals (s_0^i, s_1^j)
- Detect a sequence of state intervals :
  - in P_0 that includes s_0^i
  - in P_1 that includes s_1^j
  - that every interval in P_0 is concurrent with every interval in P_1.



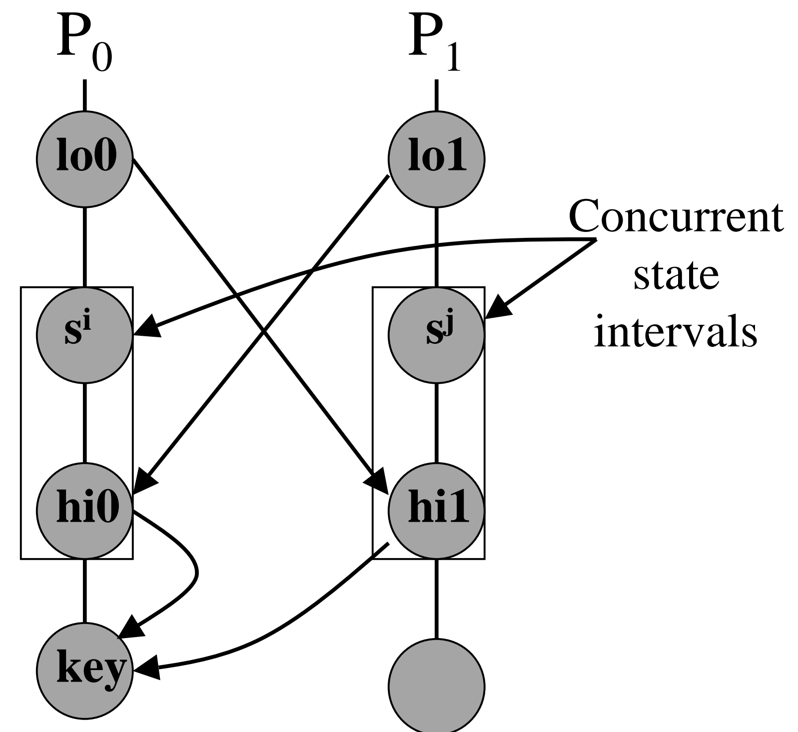
Detection of Concurrent state intervals

- hi0 = pred(key)
- hi1 = pred(key)
- lo0 = pred(hi1)
- li1 = pred(hi0)



Some Observations…

- key interval - disrupts concurrency
- key interval dictates the sequence
- key interval has to be a receive interval
- Every receive interval is a potential key interval



Algorithm- General Schema

- P_0 and P1 maintain respective min(x) in each of the state intreval
- Every interval has a unique timestamp
- When a message is received
  - Identify lo0,lo1,hi0,hi1
  - Ex. check predicate $x_0+x_1'<5$ for all pairs of states

Logical Clocks

Time Stamp P0		Min (x)	Time Stamp P1		Min (x)
lo0	1	3	lo1	1	5
s i	**2**	**2**	**s j**	**2**	**2**
hi0	3	4	hi1	3	3
key	4				

Termination

- To ensure termination :
  - assume P1 terminates first
  - P1 sends FINAL message to P0
  - Delivery of FINAL is delayed till P0 terminates

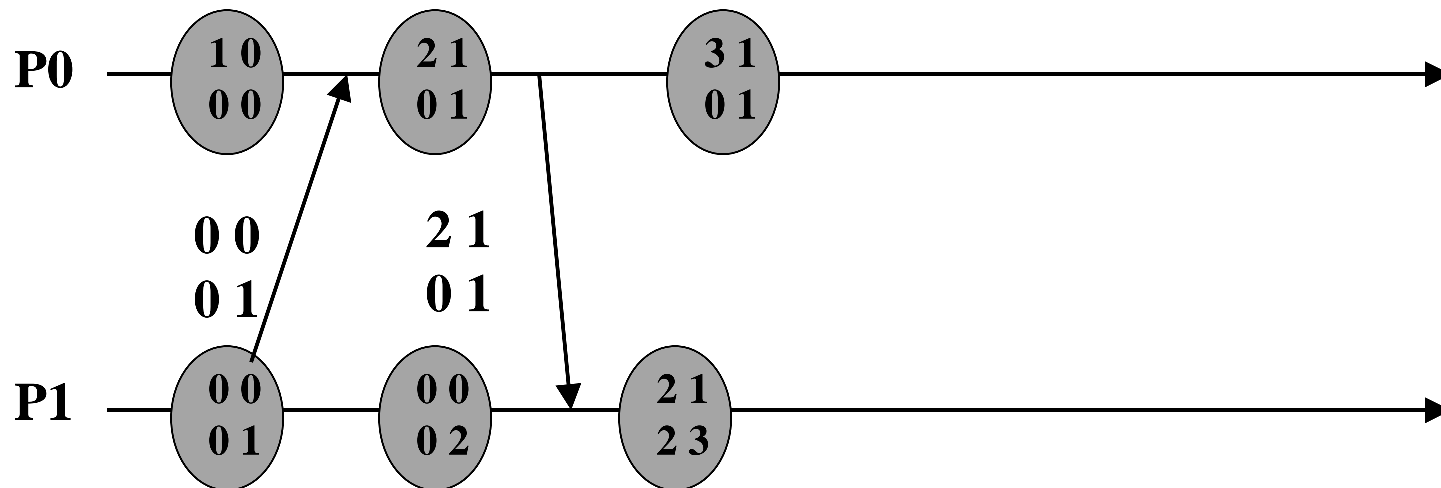
Problem

- How to find the timestamps of lo0, lo1, hi0, hi1 ?
- Solution : Matrix Clocks

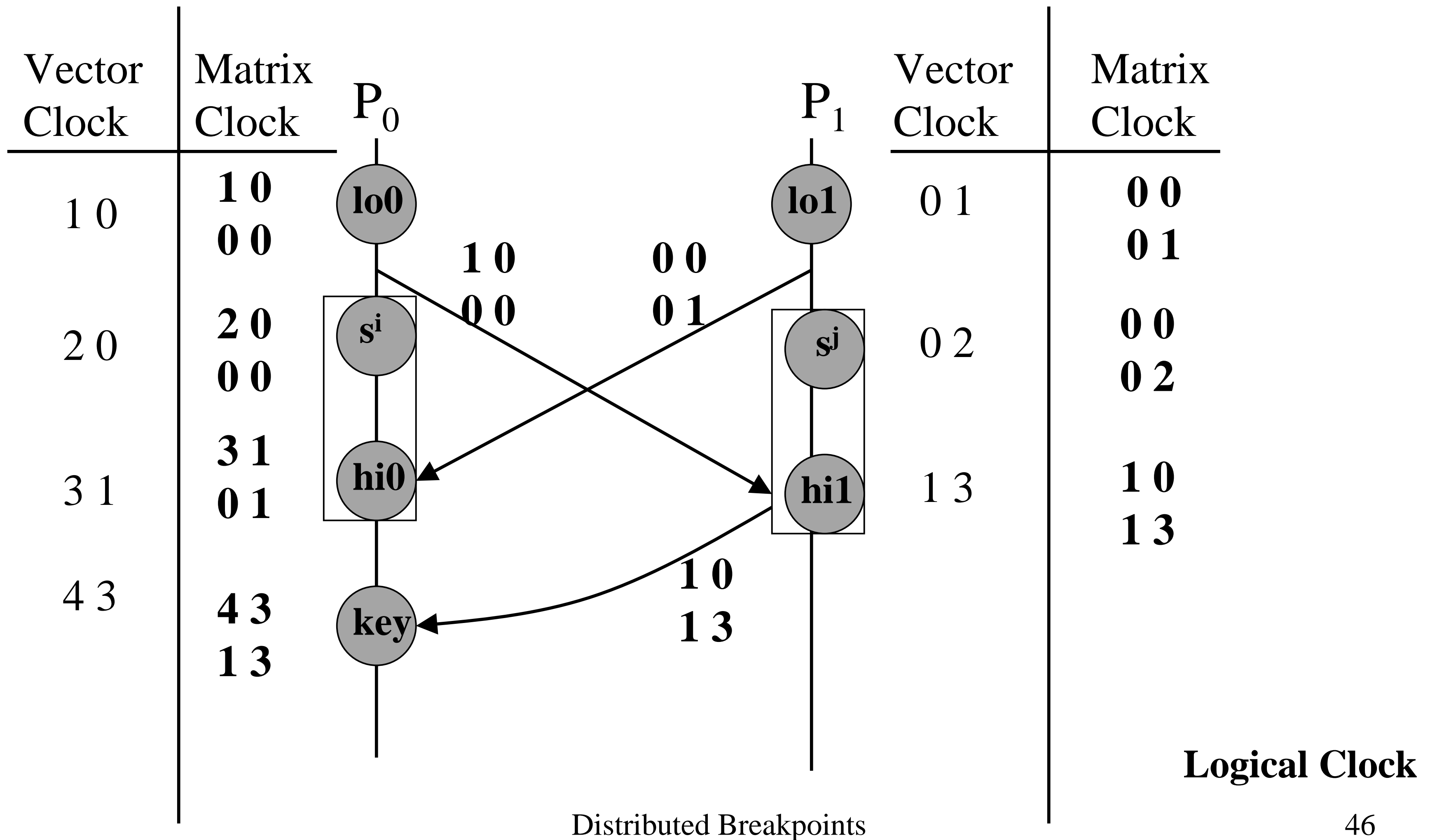
Matrix Clocks

Vector Clock of P0	Logical Clock of P0	What P0 knows about P1	What P0 knows about P2
Vector Clock of P1	What P1 knows about P0	Logical Clock of P1	What P1 knows about P2
Vector Clock of P2	What P2 knows about P0	What P2 knows about P1	Logical Clock of P2

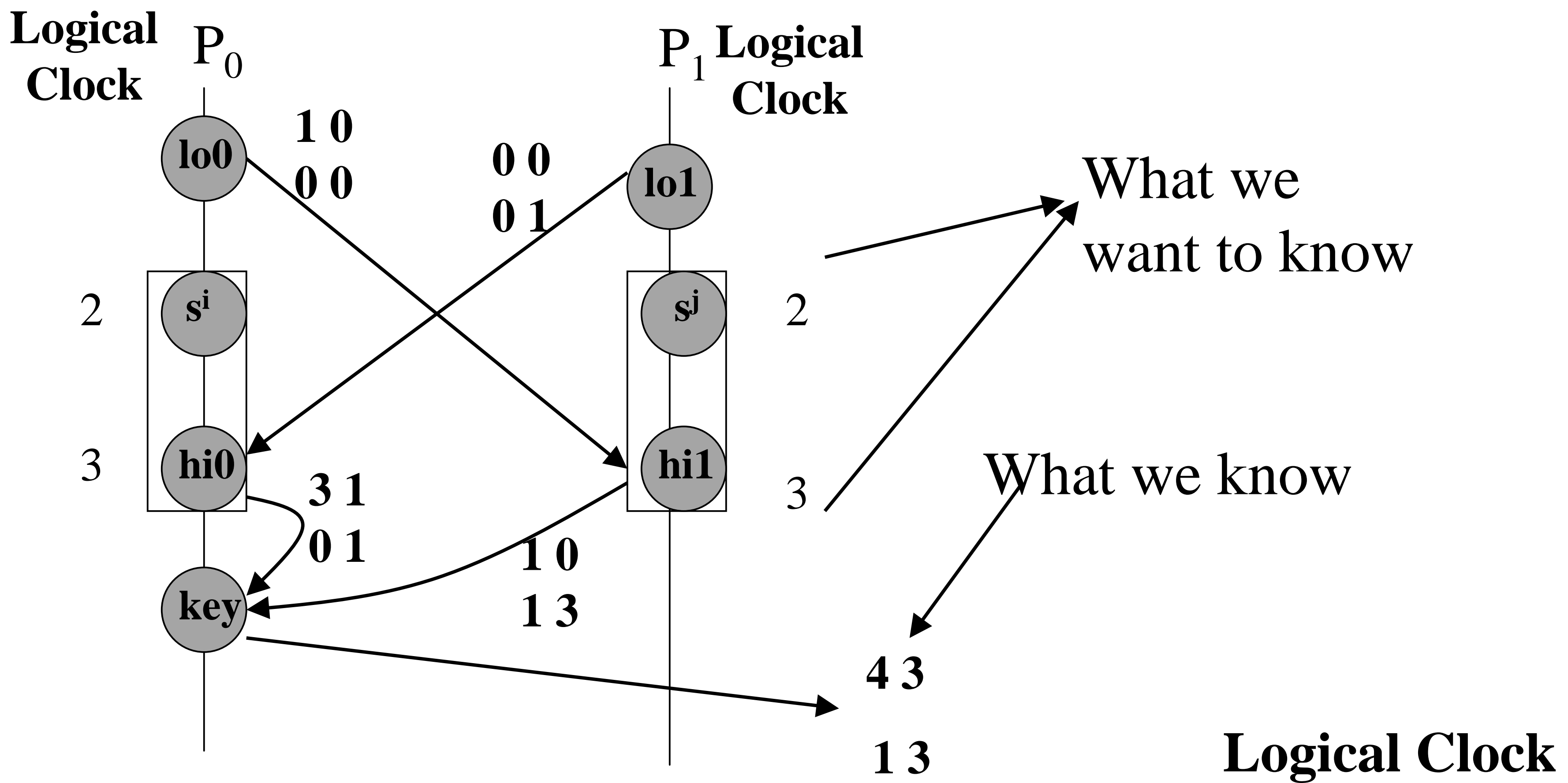
Matrix Clocks Updation



Using Matrix Clocks



Using Matrix Clocks...



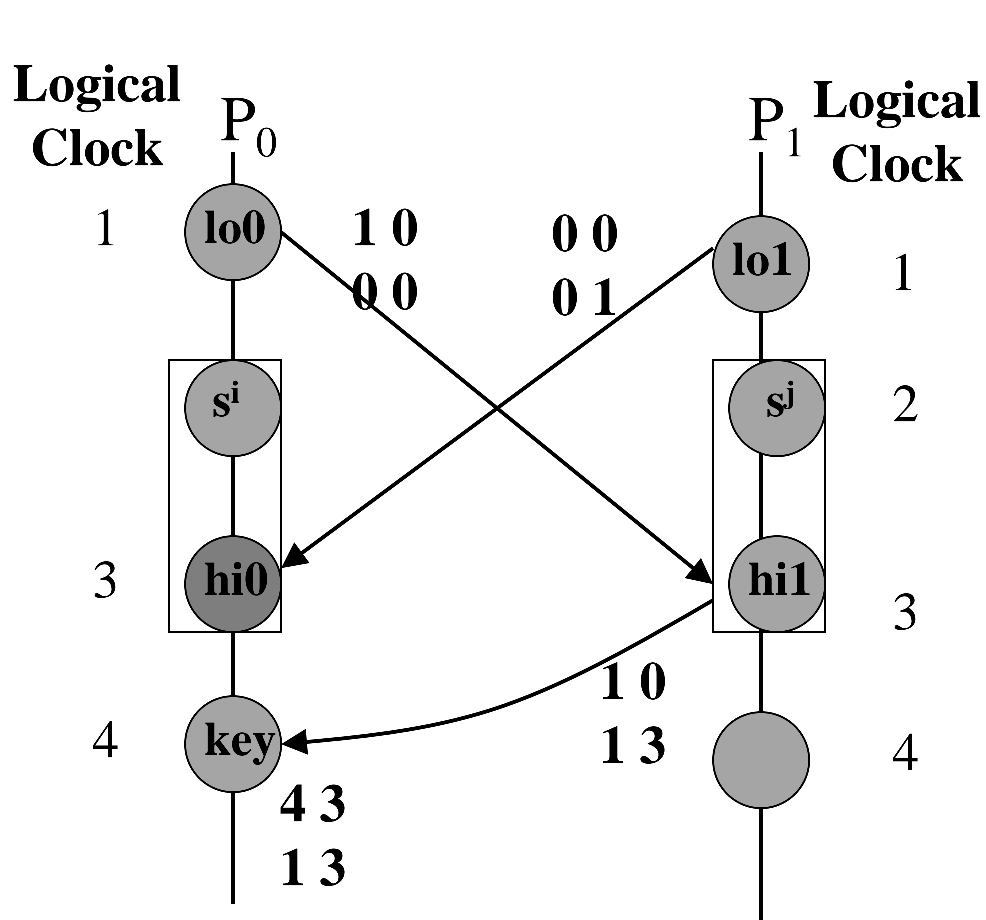
Algorithm- General Schema

- P_0 and P1 maintain respective min(x) in each of the state intreval
- Every interval has a unique timestamp
- When a message is received
  - Identify lo0,lo1,hi0,hi1
  - Ex. check predicate $x_0+x_1' < 5$ for all pairs of states

Logical Clocks

Time Stamp P0		Min (x)	Time Stamp P1		Min (x)
lo0	1	3	lo1	1	5
s i	**2**	**2**	**s j**	**2**	**2**
hi0	3	4	hi1	3	3
key	4				

Using Matrix Clocks...

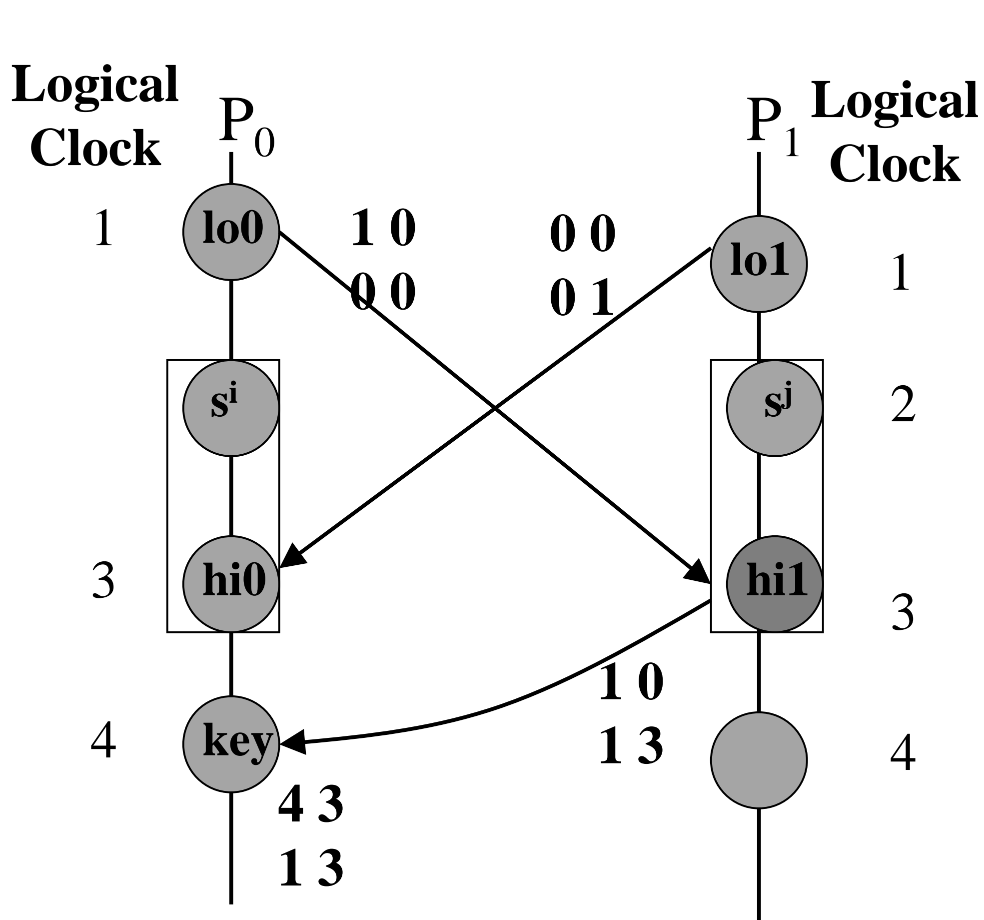


$$\begin{vmatrix} 4 & 3 \\ 1 & 3 \end{vmatrix}$$ = M (Matrix clock at Key)

T(hi0) = M[0,0] - 1

Logical Clock

Using Matrix Clocks...

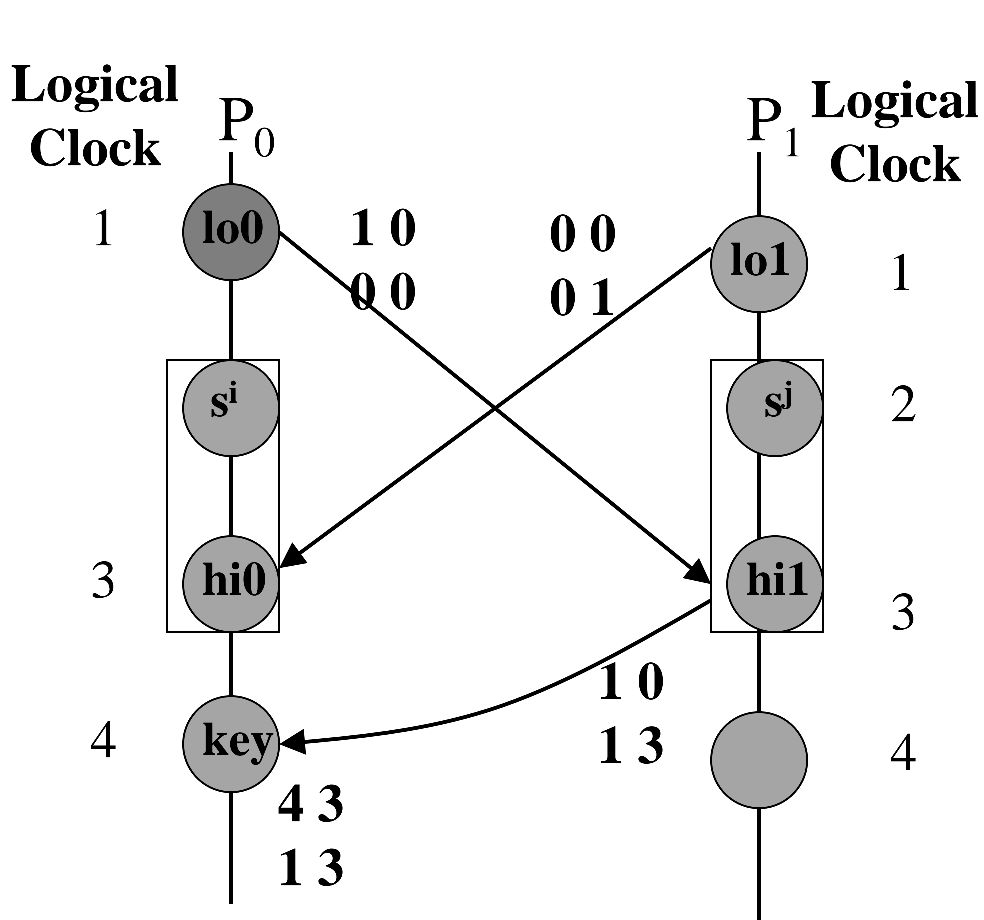


$$\begin{vmatrix} 4 & 3 \\ 1 & 3 \end{vmatrix}$$ = M (Matrix clock at Key)

T(hi0) = M[0,0] - 1
T(hi1) = M[0,1]

Logical Clock

Using Matrix Clocks...



| 4 3 |
| 1 3 | = M (Matrix clock at Key)

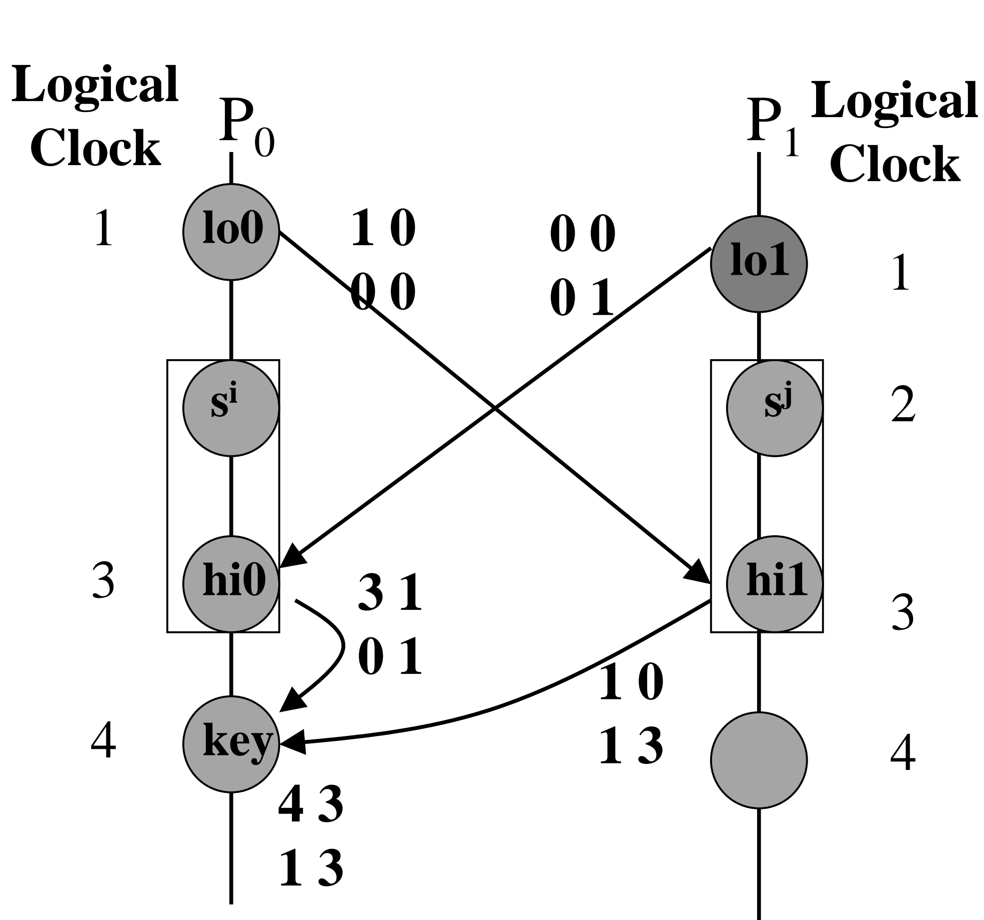
T(hi0) = M[0,0] - 1

T(hi1) = M[0,1]

T(lo0) = M[1,0]

Logical Clock

Using Matrix Clocks...



$$\begin{vmatrix} 4 & 3 \\ 1 & 3 \end{vmatrix} = M$$ (Matrix clock at Key)

T(hi0) = M[0,0] - 1

T(hi1) = M[0,1]

T(lo0) = M[1,0]

T(lo1) = M'[0,1] where

M' = Matrix clock of hi0

Logical Clock

Decentralized Algorithm

Time Stamp P0		Min (x)	Time Stamp P1		Min (x)
lo0	1	3	lo1	1	5
s i	2	2	s j	2	2
hi0	3	4	hi1	3	3
key	4				

- Every process executes the algorithm
- Ex for P0 : We know time stamps of lo0,hi0
- Corresponding sequence of concurrent is (lo0 + 1, hi0)
- Search and find Min(Min(x))
- $x_0' = 2$
- P0 sends (x0',lo1,hi1)
- for P1 $x_1' = 2$
- **$x_0' + x_1' < 5$ is detected**

Implementation

- Send or receive messages ⟵ Application messages
- Matrix clocks are sent as message tags with application messages
- x0’ is computed locally
- P0 sends (x0’,lo1,hi1) ⟵ Debug Message
- P1 finishes calculation

Overhead and Complexity

- Message Overhead
  - No of debug messages = No of receive intervals
  - Debug message (x_i', lo_i, hi_i) ⟵ 3 integers
  - Application messages (carries M.clocks) ⟵ 4 integers
- Memory Overhead
  - Need to store an array of Min(x)
  - Need to store the current Matrix Clock

Overhead and Complexity...

- Computation Overhead
  - Monitoring local variable
  - Calculate x_i' for every debug message sent and received

Is Generalization Necessary

- Over head in linked predicate algorithm
  - Unevaluated predicates had to be sent
- Overhead in generalization
  - it is proportional to total number of application messages
  - Computational overhead is also higher

Summary

- How to halt a distributed computation
- What are the different forms of predicates
- how they can be detected
- what is the cost that is incurred in that process