Chuyên đề hệ thống phân tán

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

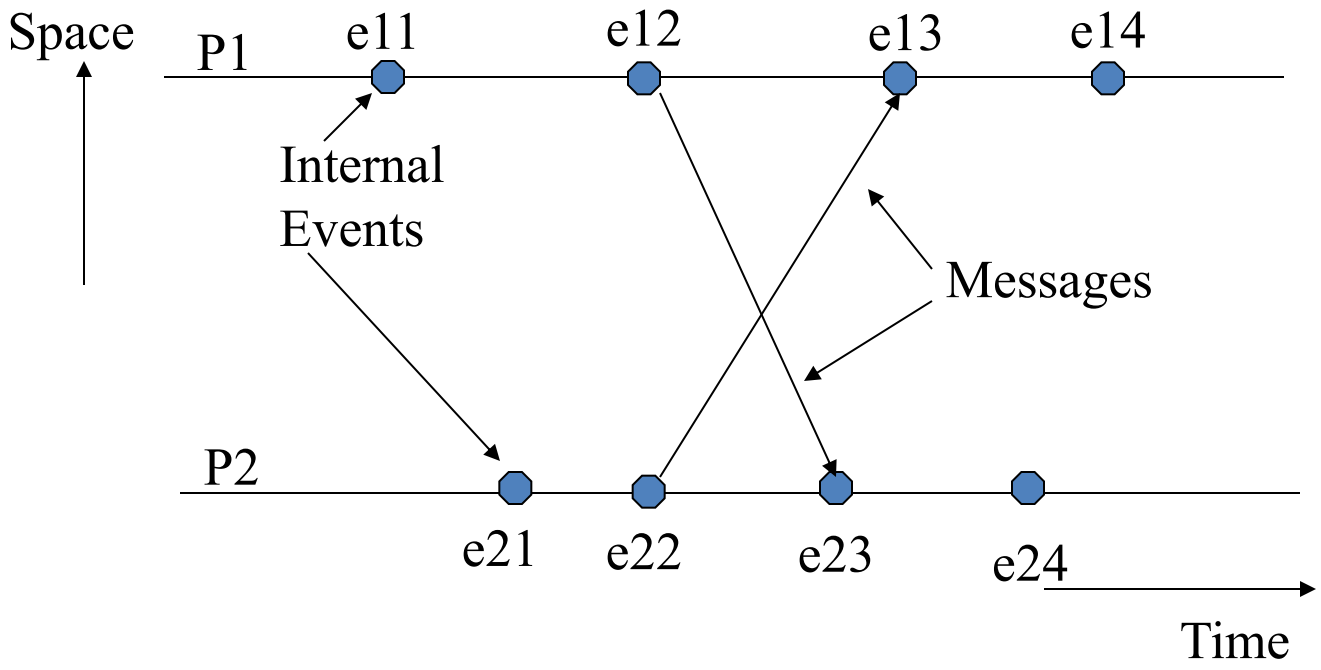
Theory aspects

- Logical Clocks – đồng hồ logic
- Causal Ordering – thứ tự nhân quả
- Global State Recording – Trạng thái toàn cục
- Termination Detection – Nhận dạng kết thúc

Lamport's Clock

- Happened before relation:
  - a -> b : Event a occurred before event b. Events in the same process.
  - a -> b : If a is the event of sending a message m in a process and b is the event of receipt of the same message m by another process.
  - a -> b, b -> c, then a -> c. "->" is transitive.
- Causally Ordered Events
  - a -> b : Event a "causally" affects event b
- Concurrent Events
  - a || b: if a !-> b and b !-> a

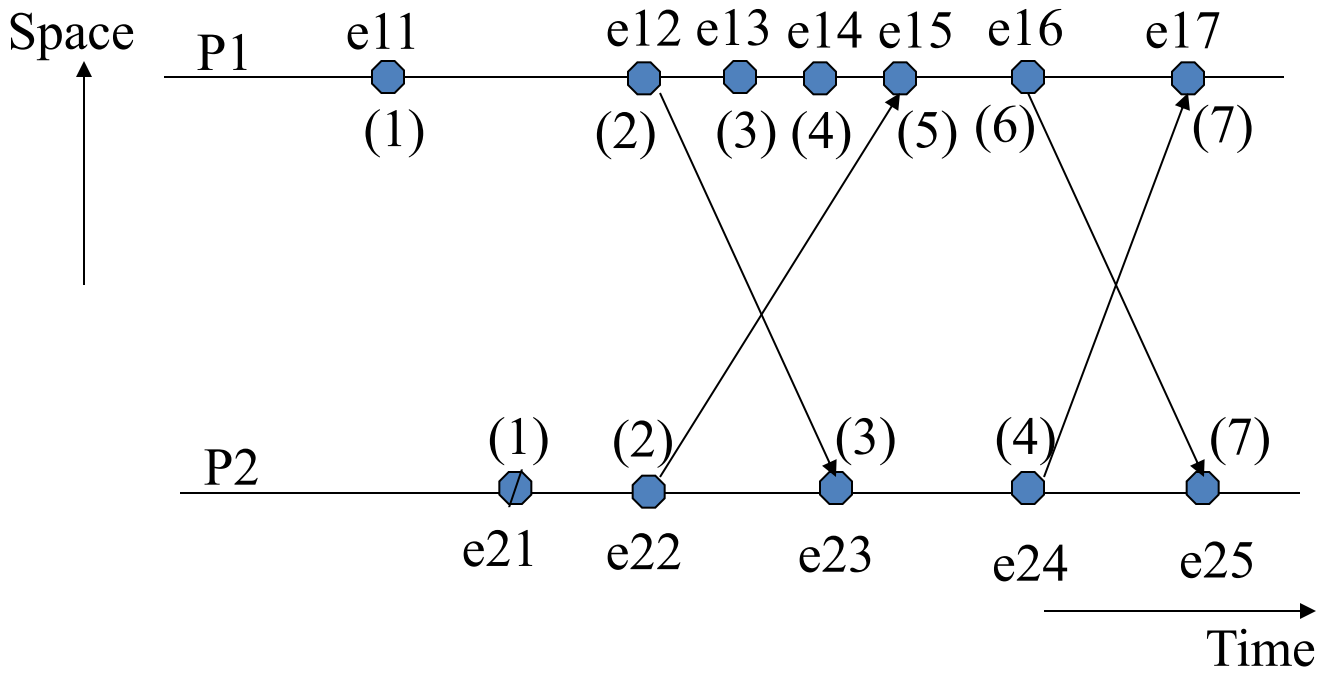
Lượt đồ không-thời gian



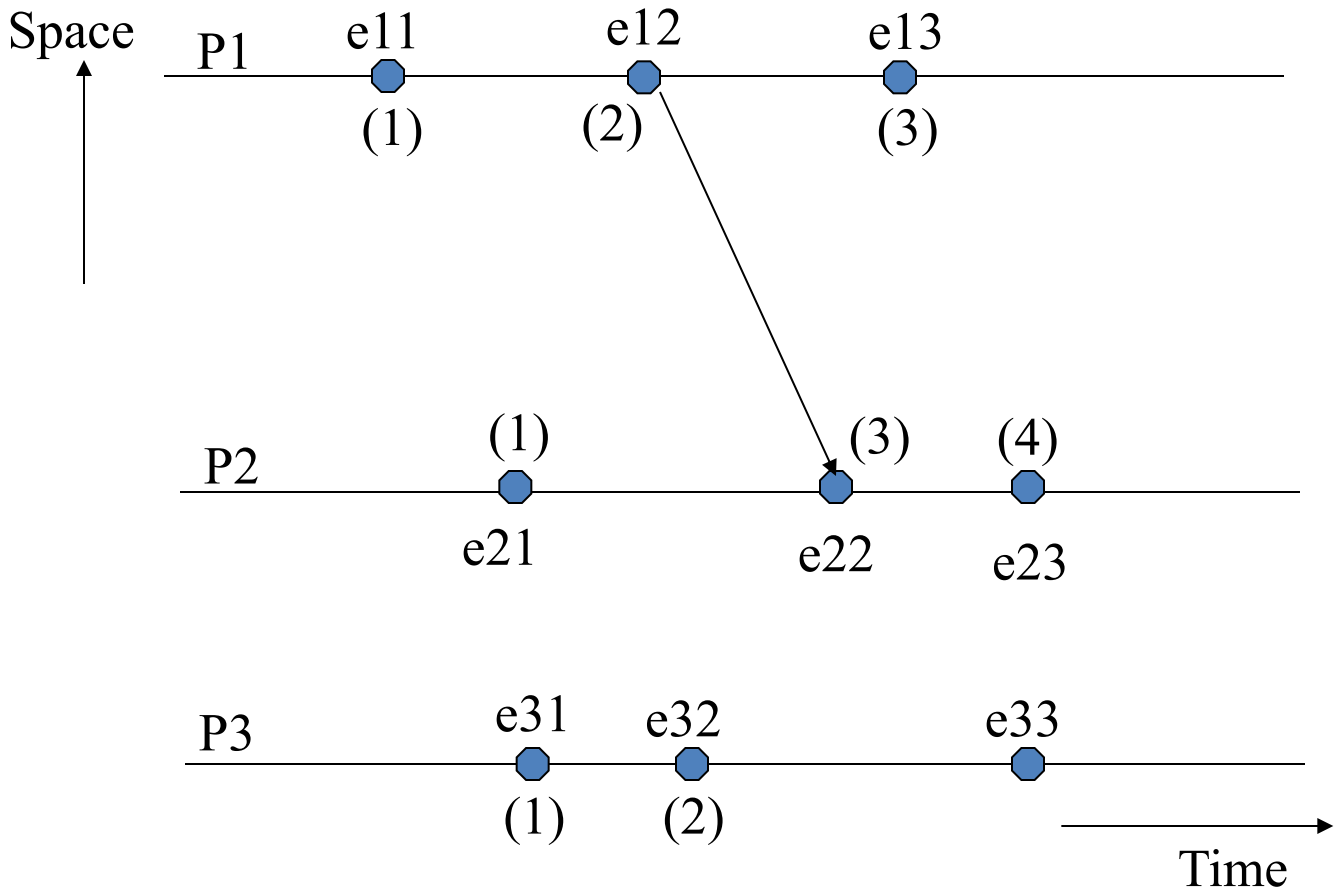
Logical Clocks – đồng hồ logic

- Qui luật:
  - Ci là giờ của Process Pi.
  - Nếu a -> b trong process Pi, Ci(a) < Ci(b)
  - Nếu a: gửi msg m trong Pi; b : nhận msg m trong Pj; thì, Ci(a) < Cj(b).
- Luật cập nhật đồng hồ:
  - R1: Ci = Ci + d (d > 0); đồng hồ đc cập nhật giữa 2 sự kiện liên tiếp.
  - R2: Cj = max(Cj+d, tm+ d); (d > 0); Khi Pj nhận m với time stamp tm (tm gán bởi bên gửi Pi; tm = Ci(a), sự kiện gửi m).
- Giá trị thường dùng cho d là 1

Lượt đồ không-thời gian



Hạn chế của Lamport's Clock



C(e11) < C(e32) nhưng không có quan hệ nhân quả.

Các thứ tự sự kiện dạng này không mô tả đc bằng Lamport's Clock.

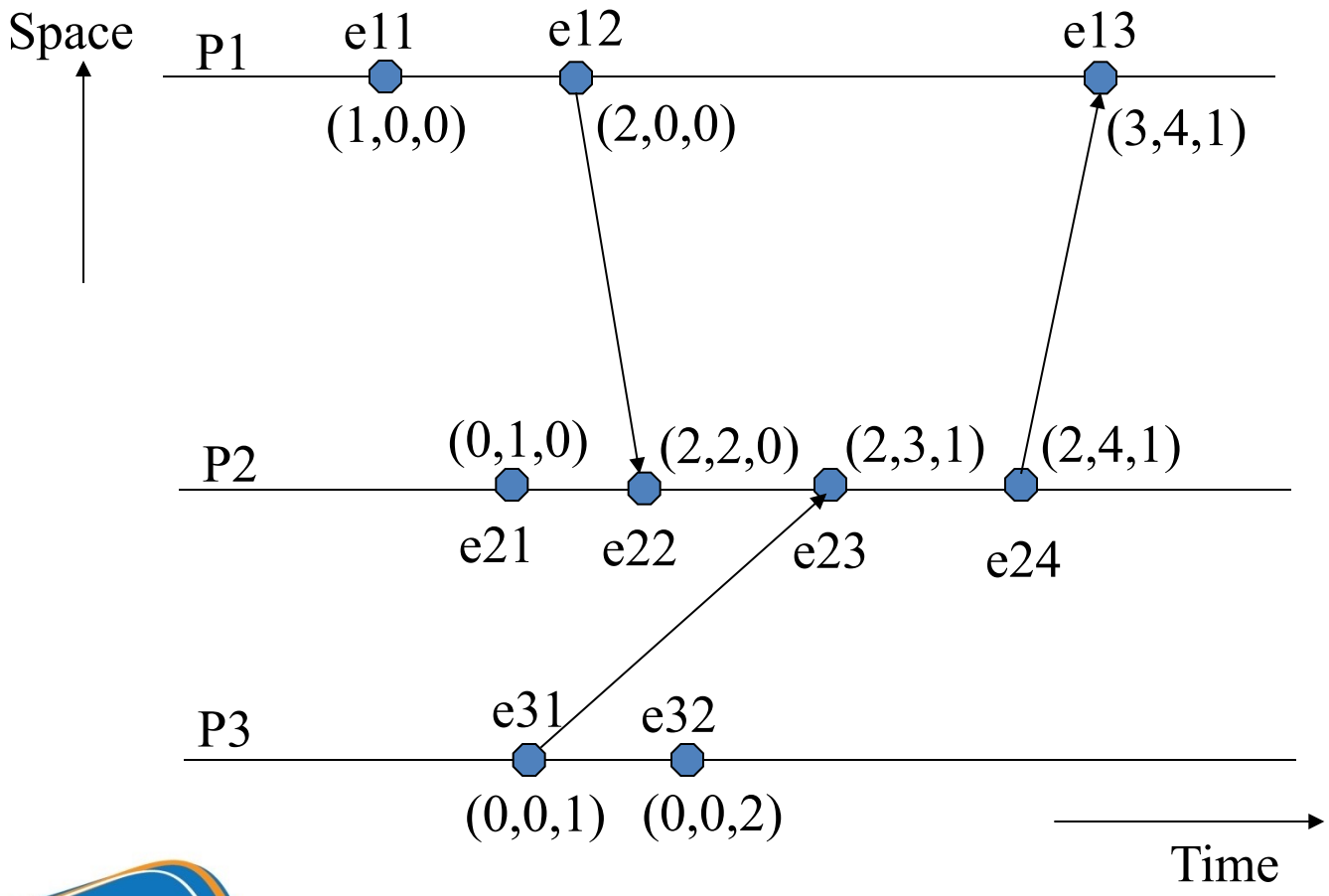
Vector Clocks –đồng hồ vector

- Lưu giữ mối qhe trao đổi giữa các process, để quá trình phục hồi (recovery).
- Ci[1..n]: là một “vector” clock lưu tại process Pi các thành phần của vector là giá trị clock (“assumed/best guess”) của các process khác.
- Ci[j] (j != i) là giá trị dự đoán tốt nhất của Pi về Pj’s clock.
- Luật cập nhật Vector clock:
  - Ci[i] = Ci[i] + d, (d > 0); cho sự kiện liên tiếp của Pi
  - Với mọi k, Cj[k] = max (Cj[k],tm[k]), khi m có time stamp tm gửi bởi Pi và được nhận tại Pj.

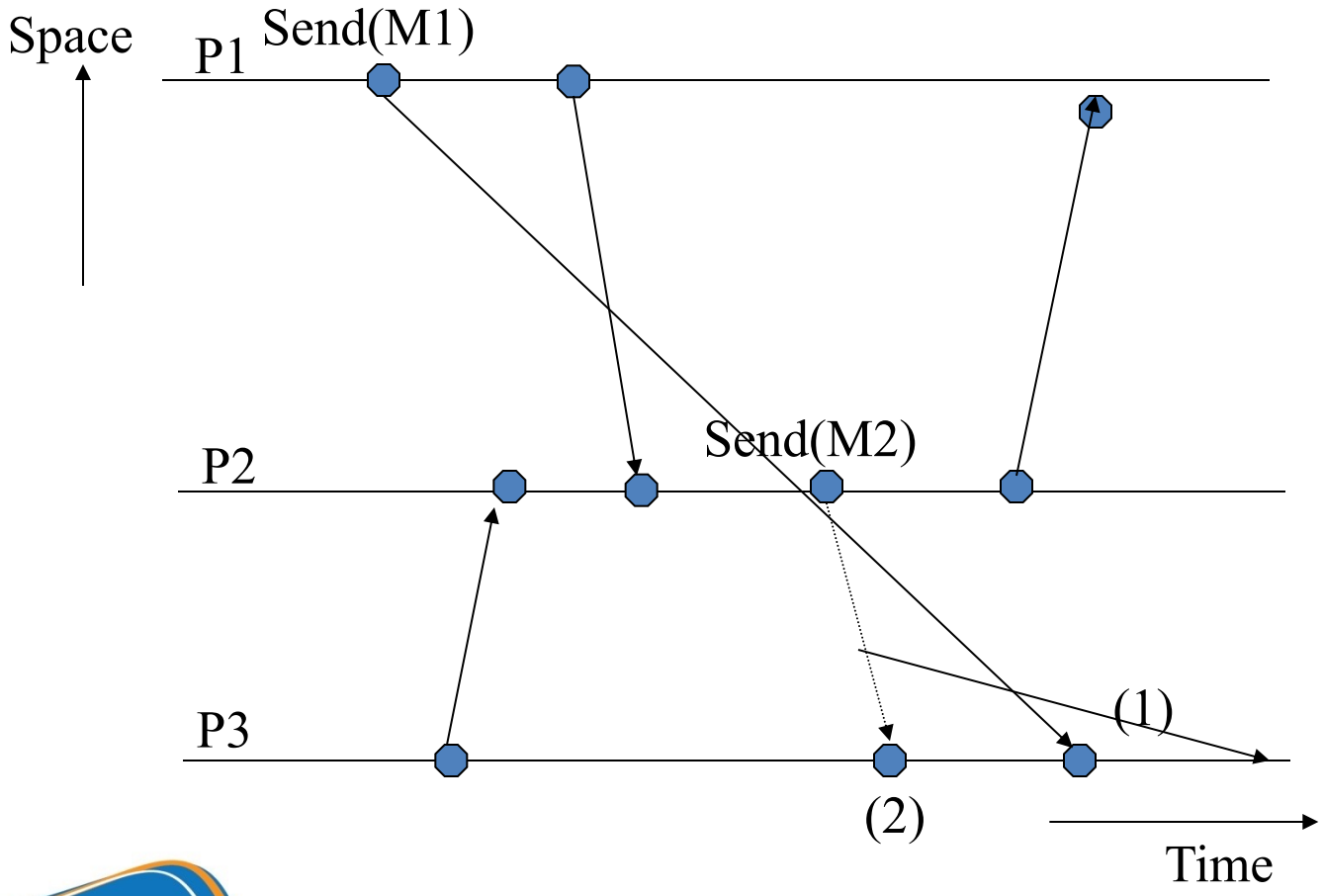
So sánh Vector Clocks

1. Equal: $t^a = t^b$ iff $\forall i, t^a[i] = t^b[i]$
2. Not Equal: t^a != t^b iff $t^a[i]$!= $t^b[i]$, for at least one i
3. Less than or equal: $t^a <= t^b$ iff $t^a[i] <= t^b[i]$, for all i
4. Less than : $t^a < t^b$ iff $t^a[i] <= t^b[i]$ and $t^a[i]$!= $t^b[i]$, for all i
5. Concurrent: $t^a \| t^b$ iff t^a !< t^b and t^b !< t^a
6. Not less than or equal ...
7. Not less than ..

Vector Clock …



Thứ tự nhân quả của Messages



Thứ tự Message …

- Không nhất thiết phải duy trì một clock.
- Thứ tự của các message giữa các processes trong hệ phân tán.
- (v.d.) Send(M1) -> Send(M2), M1 nên đc nhận trước M2.
- Điều này ko đc đảm bảo trong mạng máy tính. M1 có thể từ P1 đến P2 và M2 có thể từ P3 đến P4.
- Thứ tự Message:
  - Chuyển message cho ứng dụng khi mà message trước nó đã đc chuyển.
  - Ngược lại, lưu tạm trong buffer.

indexStr

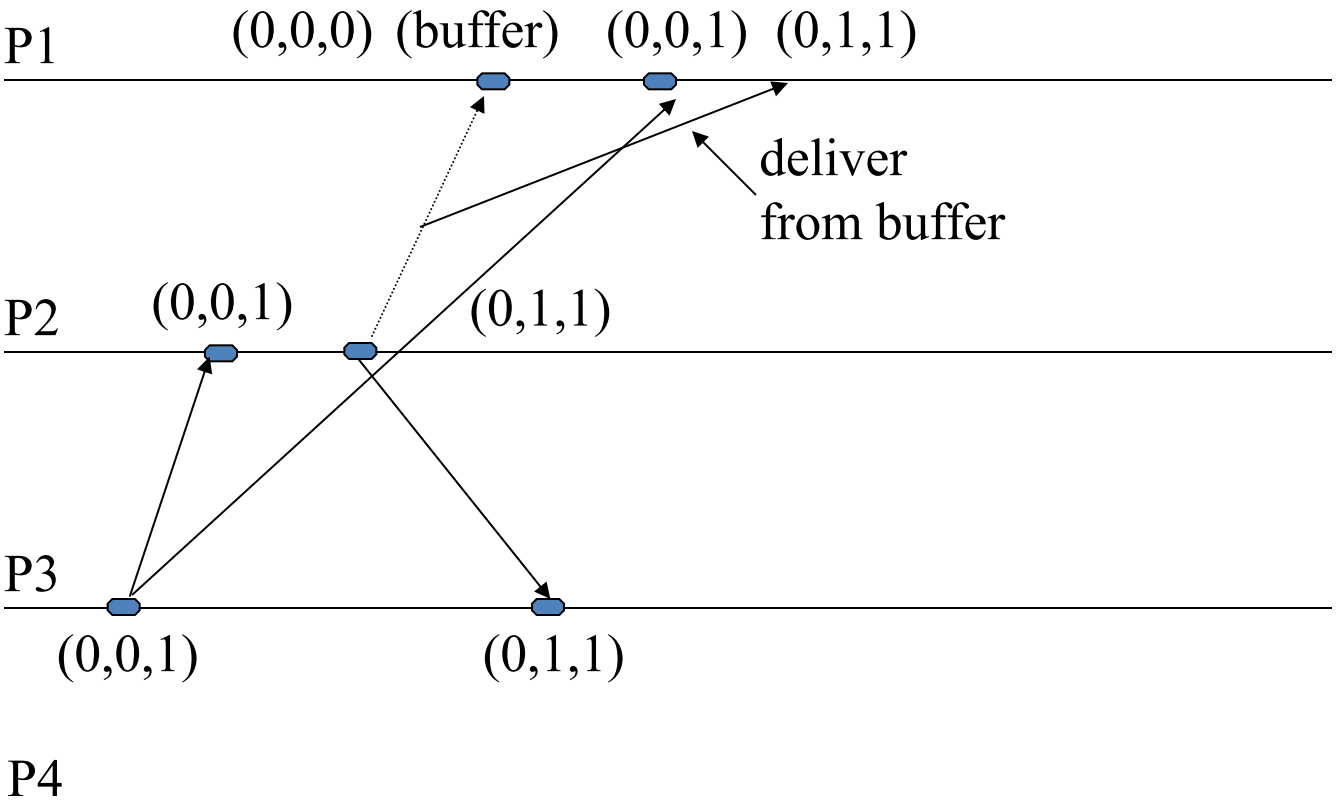
Thuật toán BSS

- BSS: Birman-Schiper-Stephenson Protocol
- Broadcast based: gửi msg tới tất cả processes.
- Chuyển message tới process khi mà message ngay trước nó đã đc chuyển.
- Ngược lại, buffer the message.
- Thực hiện bằng cách sử dụng vector gửi kèm message.

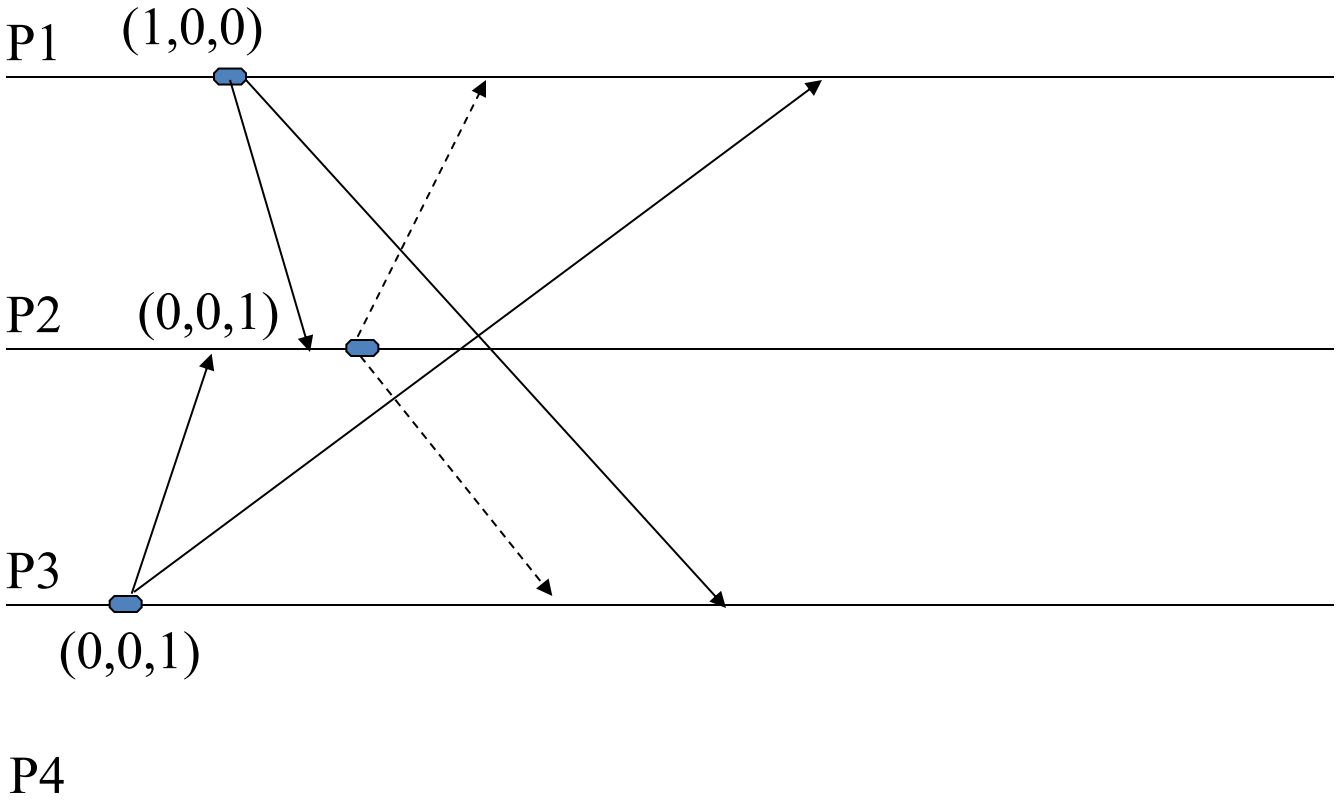
Thuật toán BSS ...

1. Process Pi tăng vector time VTpi[i], time stamps,
   và broadcasts message m. VTpi[i] - 1 là số message trước m.
2. Pj != Pi nhận m. m đc chuyển tới nếu:
   a. VTpj[i] == VTm[i] - 1
   b. VTpj[k] >= VTm[k] với mọi k in {1,2,..n} - {i}, n là tổng số processes.
   c. Messages cùng lúc đc sắp xếp theo thời điểm nhận.
3. Khi m đc chuyển giao tại Pj, VTpj đc cập nhật theo Rule 2 của vector clocks.
   2(a) : Pj nhận tất cả Pi's messages trước m.
   2(b): Pj nhận tất cả messages mà Pi đã nhận đc trước khi gửi m

Thuật toán BSS …



Thuật toán BSS …



Thuật toán SES

- SES: Schiper-Eggli-Sandoz Algorithm. Không cần broadcast message.
- Mỗi process lưu 1 vector V_P kích thước N - 1, N số lượng processes.
- Mỗi phần tử của V_P chứa (P',t): P' là id của process đích và t là vector timestamp.
- tm: thời điểm (logic) gửi m
- tPi: thời điểm (logic) hiện tại tại pi
- Ban đầu, V_P rỗng.

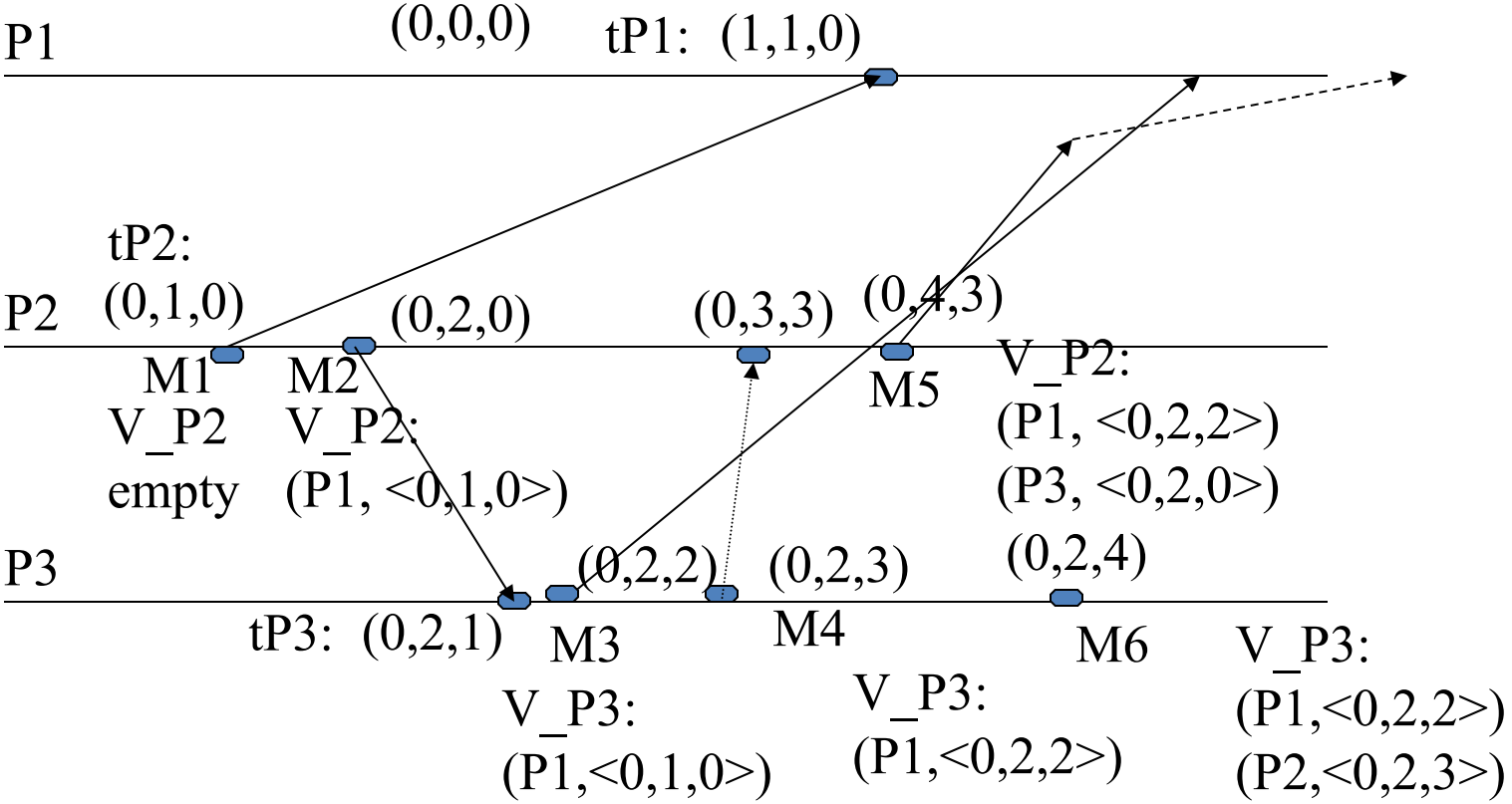
Thuật toán SES

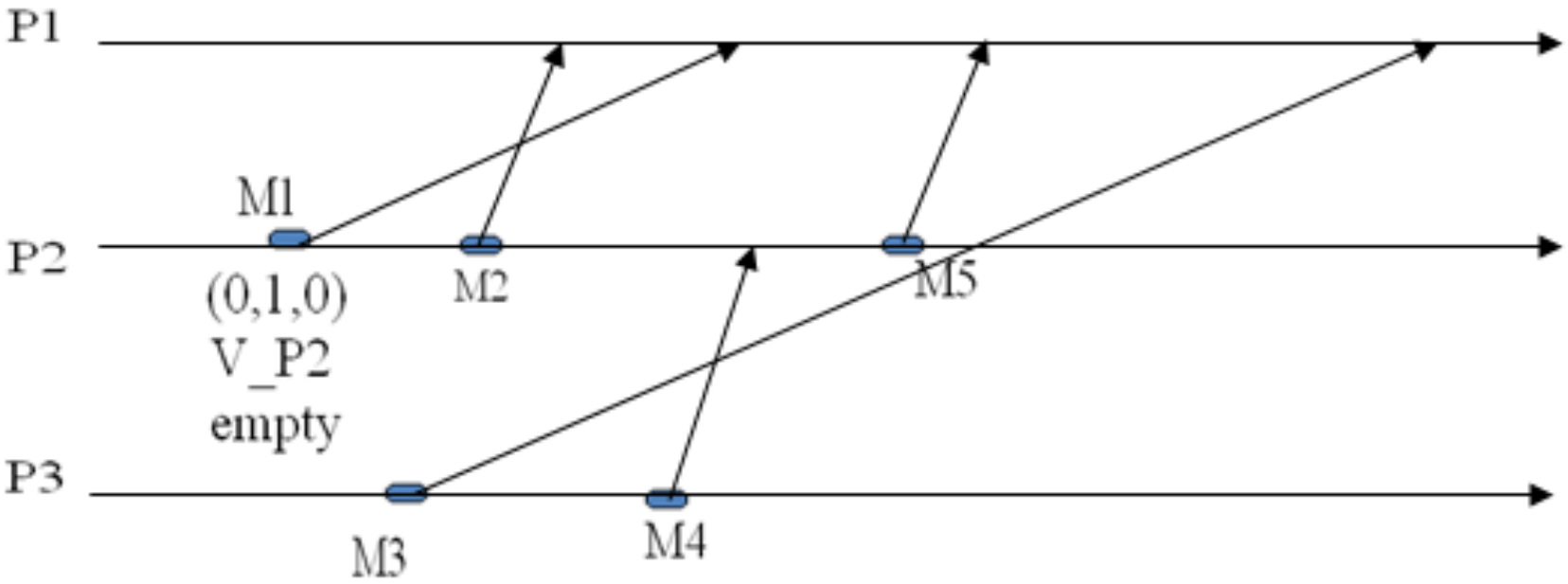
- Gửi Message:
  - Gửi message M, time stamped tm, cùng với V_P1 đến P2.
  - Thêm (P2, tm) vào V_P1. Ghi chồng lên (P2,t), nếu có.
  - (P2,tm) không đc gửi. Các message có (P2,tm) trong V_P1 chỉ đc chuyển đến P2 khi mà tm < tP2.
- Chuyển giao message
  - If V_M (in the message) không chứa (P2, t), thì có thể chuyển msg này.
  - /* (P2, t) exists */ If tm > tP2, buffer the message. (Don't deliver).
  - else (tm <= tP2) deliver it

Thuật toán SES ...

- Điều kiện **t ≥ tP2** nói lên điều gì?
  - t is message vector time stamp.
  - t > tP2 -> For all j, t[j] > tP2[j]
  - Có tồn tại sự kiện trong process khác mà P2's chưa cập nhật. Vì vậy P2 quyết định buffer the message.
- Khi t < tP2, message đc chuyển & tP2 được cập nhật với thông tin trong V_P2 (sau phép trộn).

Ví dụ SES Buffering




P1
P2
P3
M1
(0,1,0)
V_P2
empty
M2
M3
M4
M5

Ví dụ SES Buffering...

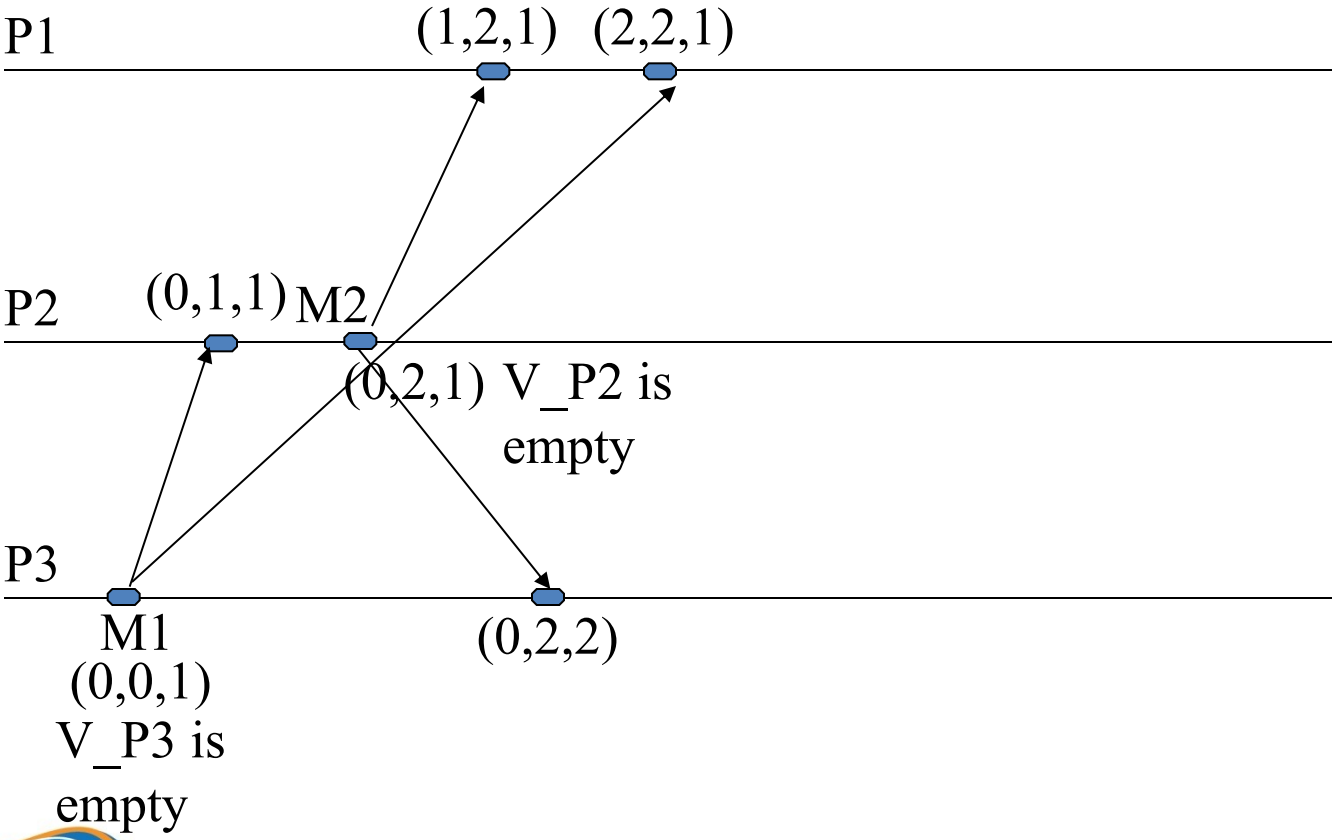
- M1 from P2 to P1: M1 + Tm (=<0,1,0>) + Empty V_P2
- M2 from P2 to P3: M2 + Tm (<0, 2, 0>) + (P1, <0,1,0>)
- M3 from P3 to P1: M3 + <0,2,2> + (P1, <0,1,0>)
- M3 gets buffered because:
  - Tp1 is <0,0,0>, t in (P1, t) is <0,1,0> & so Tp1 < t
- When M1 is received by P1:
  - Tp1 becomes <1,1,0>, by rules 1 and 2 of vector clock.
- After updating Tp1, P1 checks buffered M3.
  - Now, Tp1 > t [in (P1, <0,1,0>].

  So M3 is delivered.

Thuật toán SES ...

- Khi chuyển giao message:
  - Merge V_M (in message) with V_P2 as follows.
    - If (P,t) is not there in V_P2, merge.
    - If (P,t) is present in V_P2, t is updated with max(t[i] in Vm, t[i] in V_P2). {Component-wise maximum}.
  - Message không chuyển cho tới khi t trong V_M nhỏ hơn t trong V_P2
  - Cập nhật site P2's local, logical clock.
  - Kiểm tra các message đang được buffered, logical clock update.

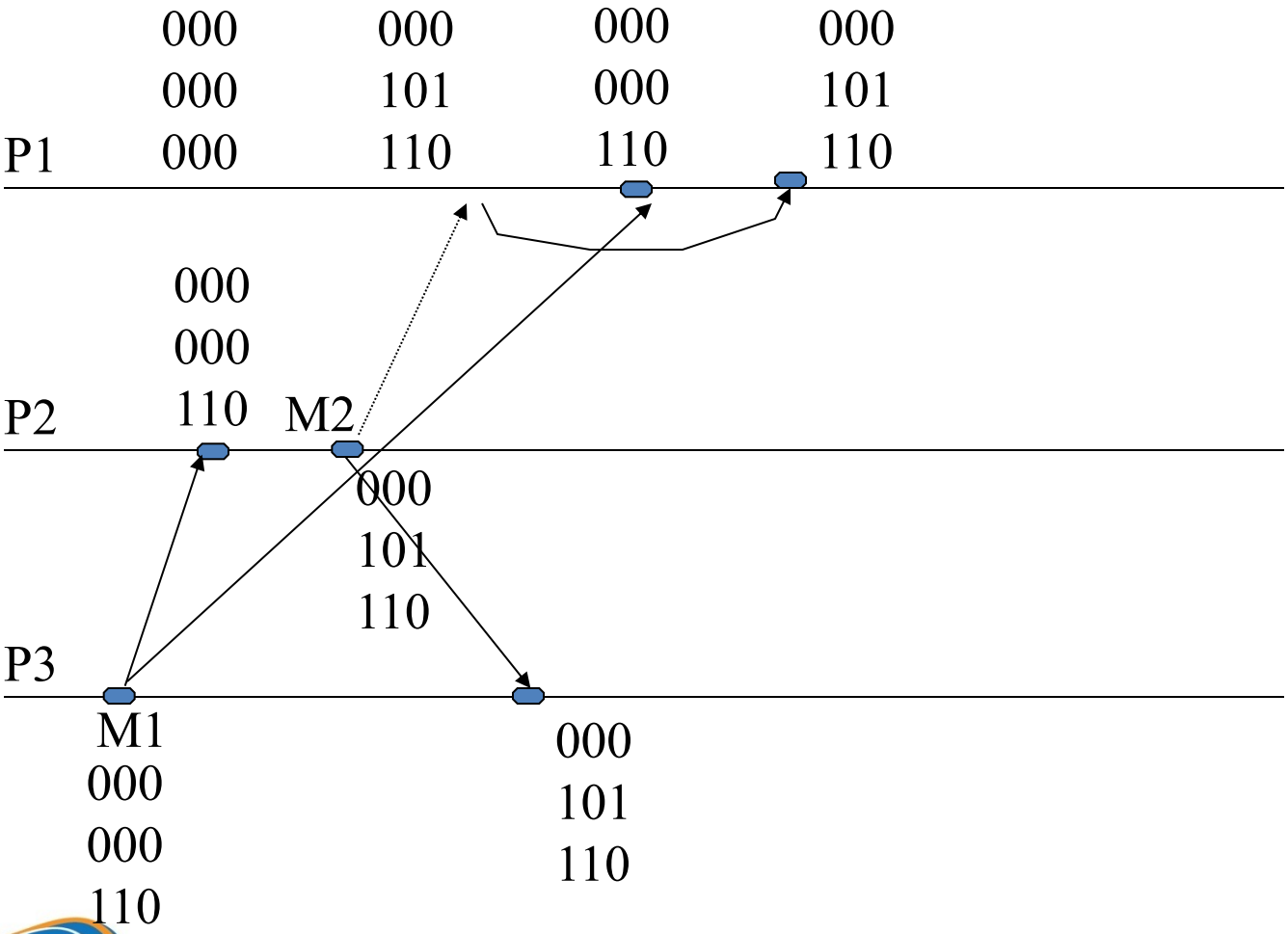
SES Algorithm …



Multicasts

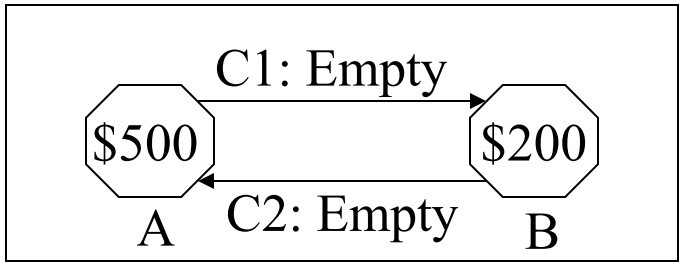
- Mỗi node lưu matran M (*n x n),* n số lượng processes.
- Node i multicasts to j and k: tăng Mi[i,j] và Mi[i,k]. M đc gửi cùng với message.
- Khi node j nhận m từ i, nó có thể đc chuyển nếu và chỉ nếu:
  - Mj[i,j] = Mm[i,j] - 1
  - Mj[k,j] >= Mm[k,j] for all k != i.
- Còn lại buffer the message
- Khi chuyển giao message: Mj[x,y] = max(Mj[x,y], Mm[x,y])

Ví dụ Multicasts

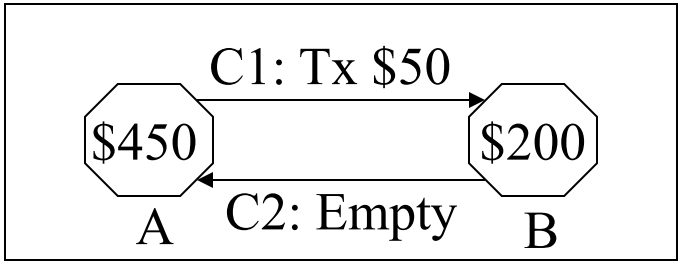


Global State – trạng thái toàn cục

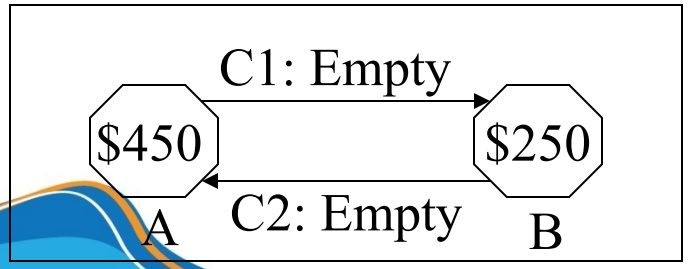
Global State 1



Global State 2



Global State 3



Lưu Global State...

- (v.d.,) Global state of A đc lưu ở (1) không phải ở (2).
  - Trạng thái của B, C1, và C2 đc lưu trong (2)
  - Khoản tiền $50 sẽ xuất hiện trong global state
  - Lý do: A's state được lưu trước khi gửi message và C1's state *lưu sau khi gửi message*.
- Inconsistent global state if n < n', trong đó
  - n số message đc gửi bởi A và được lưu trong A's state
  - n' số message gửi bởi A đang trong channel, và được lưu trong channel's state.
- Consistent global state: n = n'

Lưu Global State...

- Tương tự, để nhất quán m = n’
  - n’: no. of messages trong channel gửi đến B
  - m: no. of messages mà B được nhận từ channel này.
- Nên n’ < m, không tồn tại hệ thống mà số lượng msg gửi ít hơn số lượng msg nhận
- Vậy, **n’ >= m**
- Consistent global state phải thỏa điều kiện trên.
- Consistent global state:
  - Channel state: thứ tự message đc gửi trước khi lưu sender’s state, trừ đi các message đã đc nhận và đã lưu tại receiver’s state.
  - Chỉ có transit messages được lưu trong channel state.

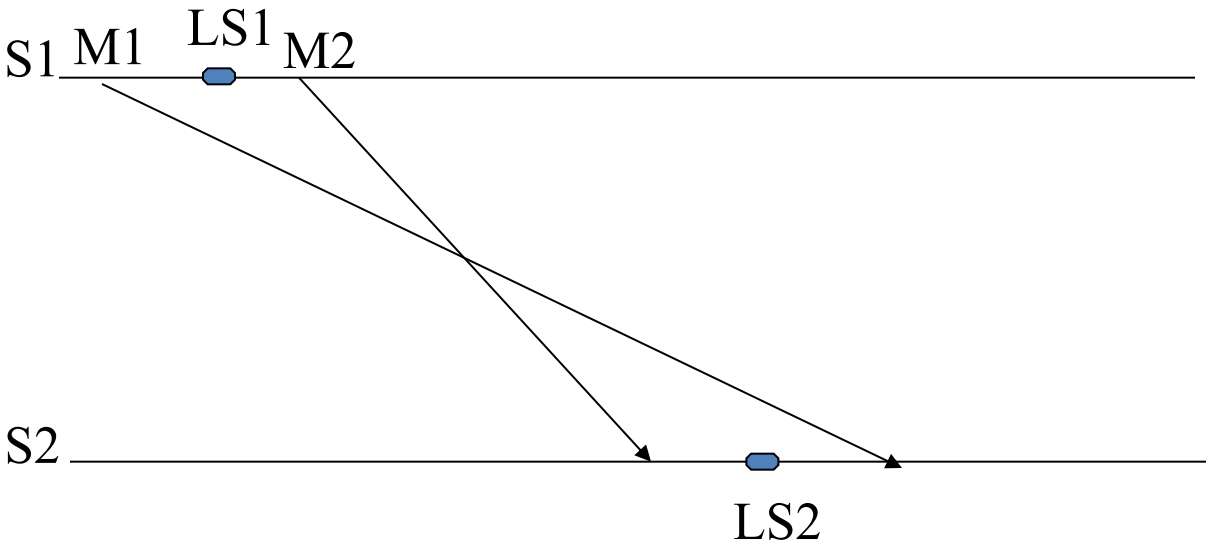
Lưu Global State

- Send(Mij): message M sent from Si to Sj
- rec(Mij): message M received by Sj, from Si
- time(x): Time of event x
- LSi: local state at Si
- send(Mij) is in LSi iff (if and only if) time(send(Mij)) < time(LSi)
- rec(Mij) is in LSj iff time(rec(Mij)) < time(LSj)
- transit(LSi, LSj) : set of messages sent/recorded at LSi and NOT received/recorded at LSj

Lưu Global State …

- inconsistent(LSi,LSj): set of messages NOT sent/recorded at LSi and received/recorded at LSj
- Global State, GS: {LS1, LS2,…., LSn}
- Consistent Global State, GS = {LS1, ..LSn} AND for all i,j in n, inconsistent(LSi,LSj) is null.
- Transitless global state, GS = {LS1,…,LSn} AND for all i,j in n, transit(LSi,LSj) is null.

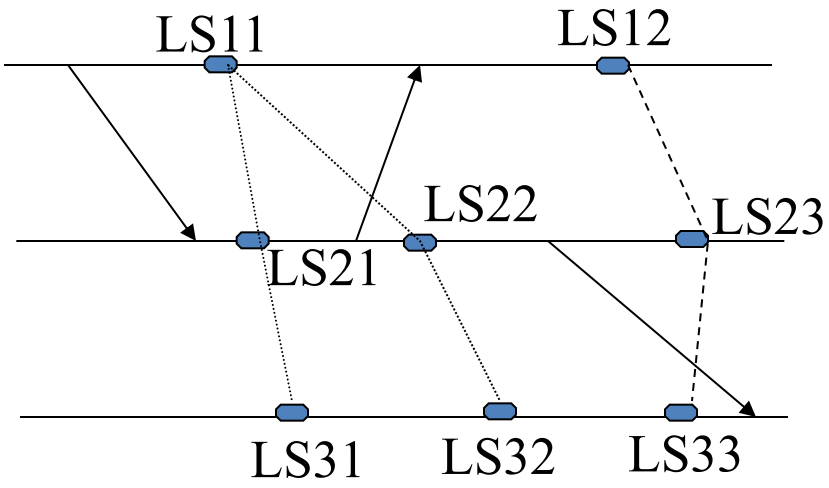
Lưu Global State ..



M1: transit
M2: inconsistent

Lưu Global State...

- Strongly consistent global state: consistent và transitless, i.e., tất cả các sự kiện gửi và sk nhận đã được lưu trong tất cả LSi.

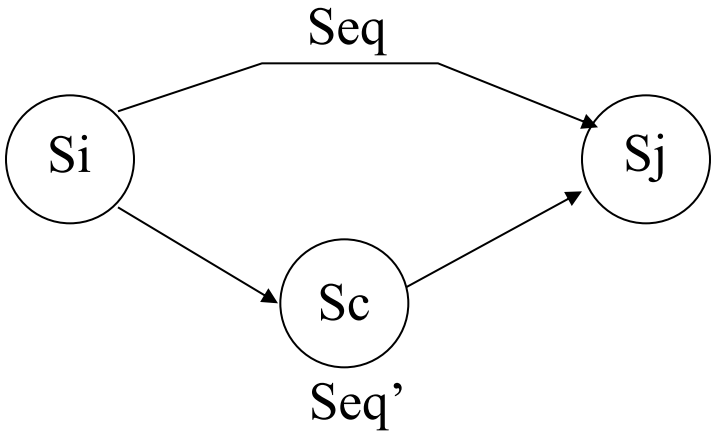


Thuật toán Chandy-Lamport

- Thuật toán phân tán để lưu consistent global state. Kênh truyền giả thiết là FIFO.
- Sử dụng 1 *marker* để khởi tạo thuật toán. Marker sort of dummy message, with no effect on the functions of processes.
- Gửi Marker tại P:
  - P lưu trạng thái của nó.
  - Với mỗi outgoing channel C, P gửi marker qua C trước khi P gửi các message khác qua C.
- Nhận Marker tại Q:
  - Nếu Q CHƯA lưu trạng thái: (a). Lưu trạng thái của C là rỗng. (b) SEND marker (sử dụng luật bên bên trên).
  - Ngược lại (Q đã lưu trạng thái): lưu trạng thái của C là chuỗi message đã nhận từ C, trong khoảng thời gian Q's state được lưu và trước khi Q nhận được marker.
- FIFO channel + markers làm thỏa tính nhất quán.

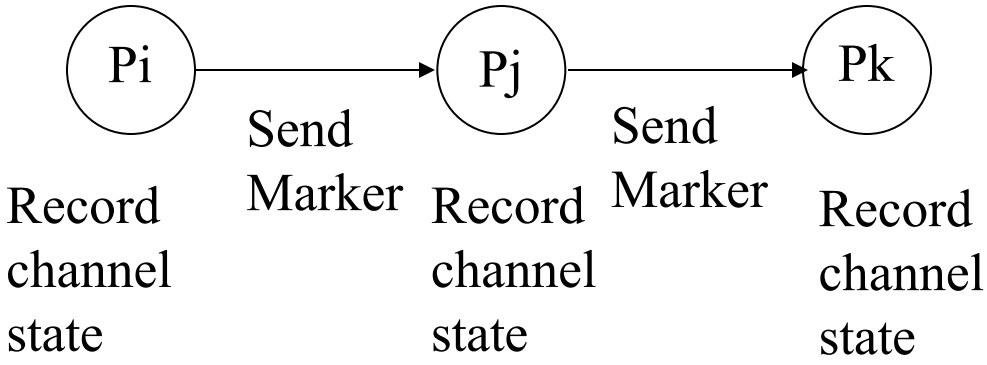
Thuật toán Chandy-Lamport

- Các process có thể khởi tạo marker, giá trị marker duy nhất: <process id, sequence number>.
- Các processes có thể khởi tạo quá trình lưu trạng thái hệ thống cùng lúc.
- One possible way to collect global state: all processes send the recorded state information to the initiator of marker. Initiator process can sum up the global state.



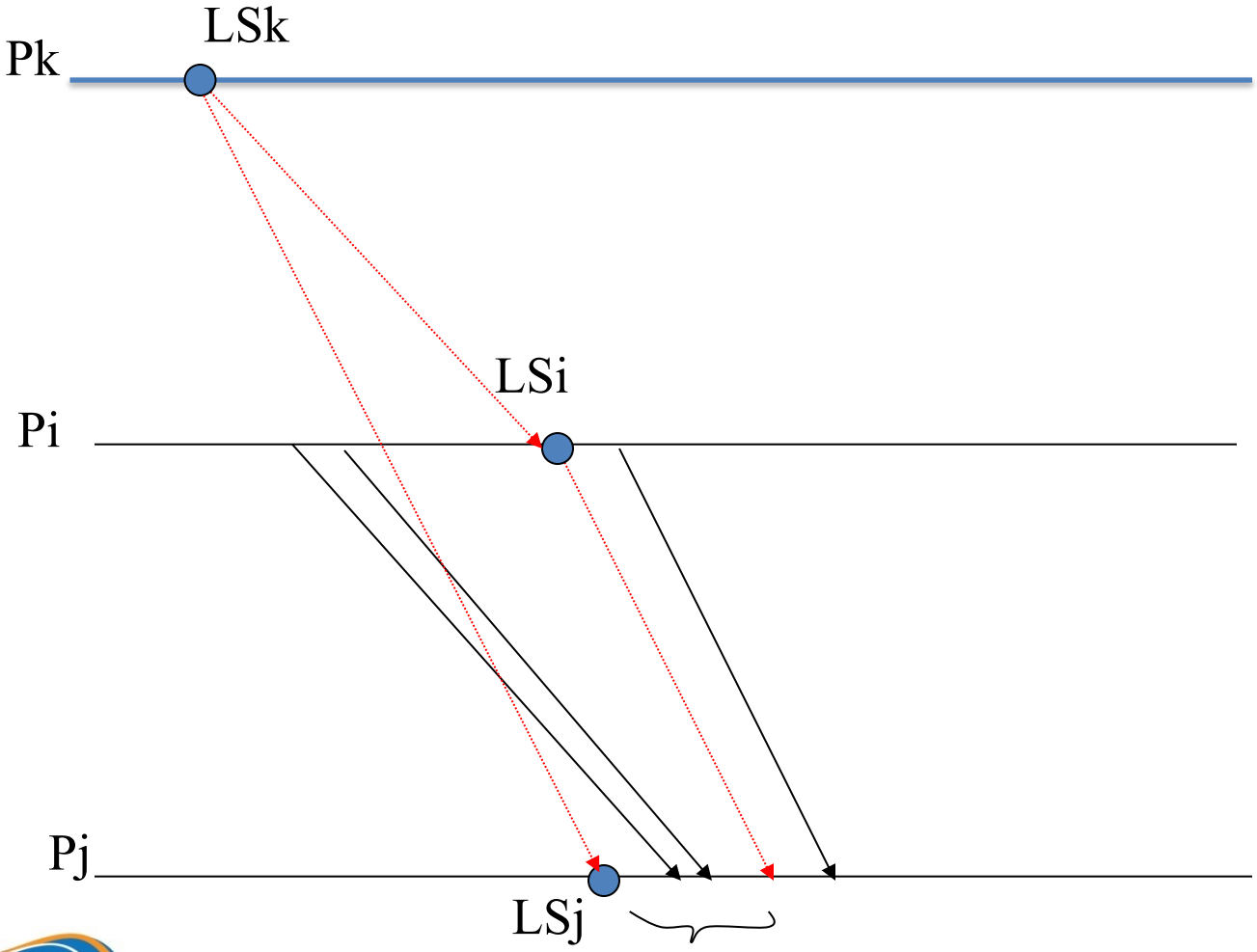
Thuật toán Chandy-Lamport ...

- **Example:**



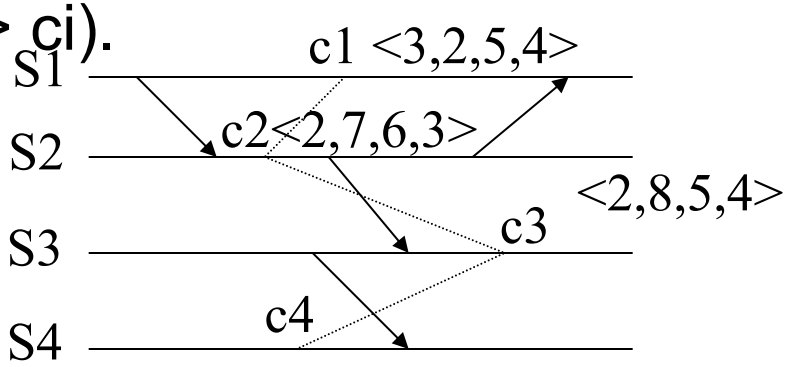
Channel state example: M1 sent to Px at t1, M2 sent to Py at t2, ….

Chandy-Lamport Algorithm ...

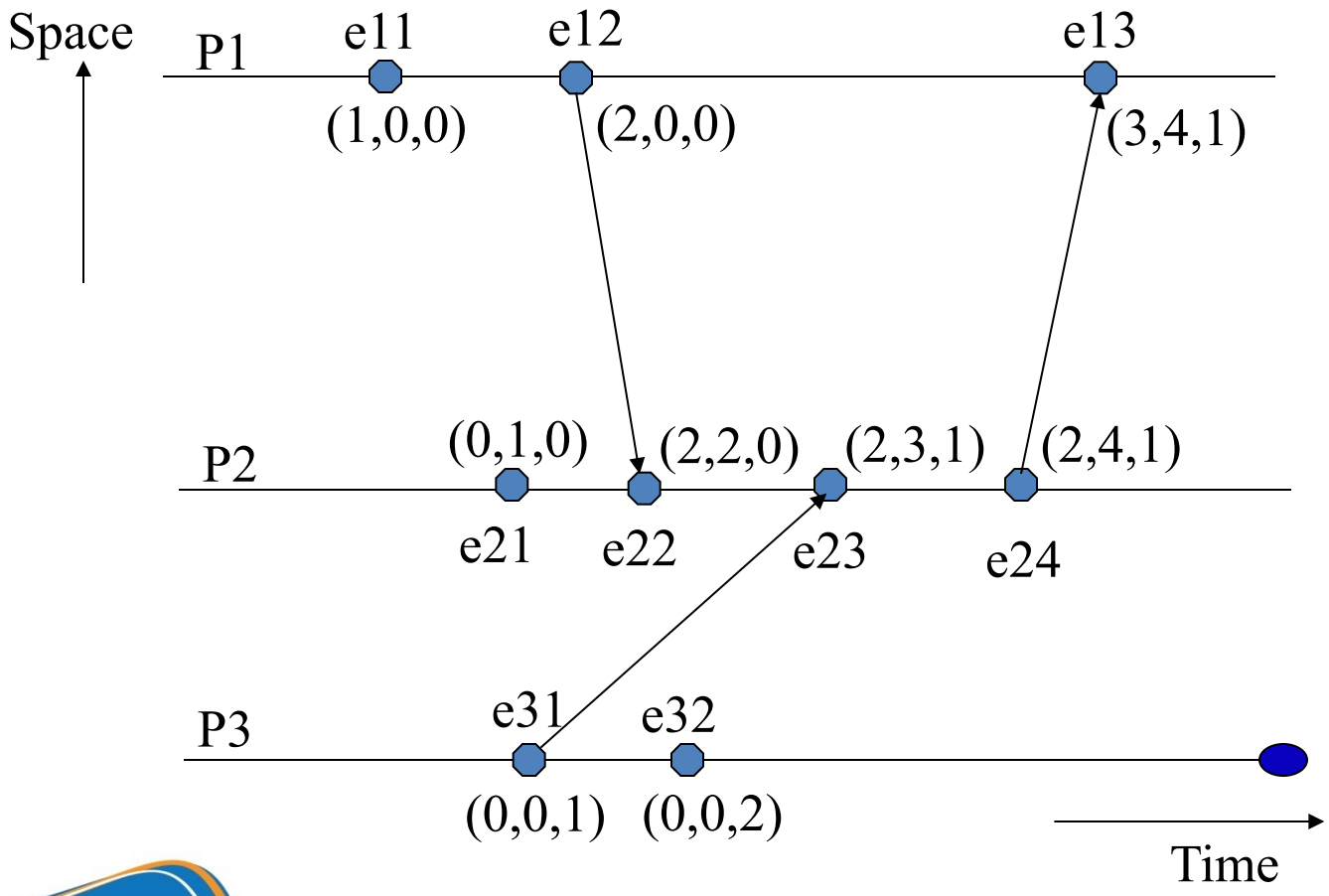


Cuts – lát cắt

- Cuts: biểu diễn bằng hình của một global state.
- Cut C = {c1, c2, .., cn}; ci: sự kiện cut tại Si.
- Consistent Cut: nếu mỗi message đc nhận tại Si *xảy ra trước* cut event, thì phải đc gửi trước cut event tại bên Gửi.
- Định lý: Một cut là một consistent cut iff không tồn tại hai cut events có quan hệ nhân quả, i.e., !(ci -> cj) và !(cj -> ci).



Vector Clock …



Thời điểm của một Cut

- C = {c1, c2, .., cn} with vector time stamp VTci. Vector time of the cut, VTc = *sup*(VTc1, VTc2, .., VTcn).
- *sup* is a component-wise maximum, i.e., VTci = max(VTc1[i], VTc2[i], .., VTcn[i]).
- Định lý: a cut is consistent iff VTc = (VTc1[1], VTc2[2], .., VTcn[n]).

Termination Detection

- Termination: completion of the sequence of algorithm. (e.g.,) leader election, deadlock detection, deadlock resolution.
- Use a *controlling agent* or *a monitor process.*
- Initially, all processes are idle. Weight of controlling agent is 1 (0 for others).
- Start of computation: message from controller to a process. Weight: split into half (0.5 each).
- Repeat this: any time a process send a computation message to another process, split the weights between the two processes (e.g., 0.25 each for the third time).
- End of computation: process sends its weight to the controller. Add this weight to that of controller's. (Sending process's weight becomes 0).
- *Rule:* Sum of W always 1.
- *Termination:* When weight of controller becomes 1 again.

Huang's Algorithm

- B(DW): computation message, DW is the weight.
- C(DW): control/end of computation message;
- Rule 1: Before sending B, compute W1, W2 (such that W1 + W2 is W of the process). Send B(W2) to Pi, W = W1.
- Rule 2: Receiving B(DW) -> W = W + DW, process becomes active.
- Rule 3: Active to Idle -> send C(DW), W = 0.
- Rule 4: Receiving C(DW) by controlling agent -> W = W + DW, If W == 1, computation has terminated.

Huang's Algorithm