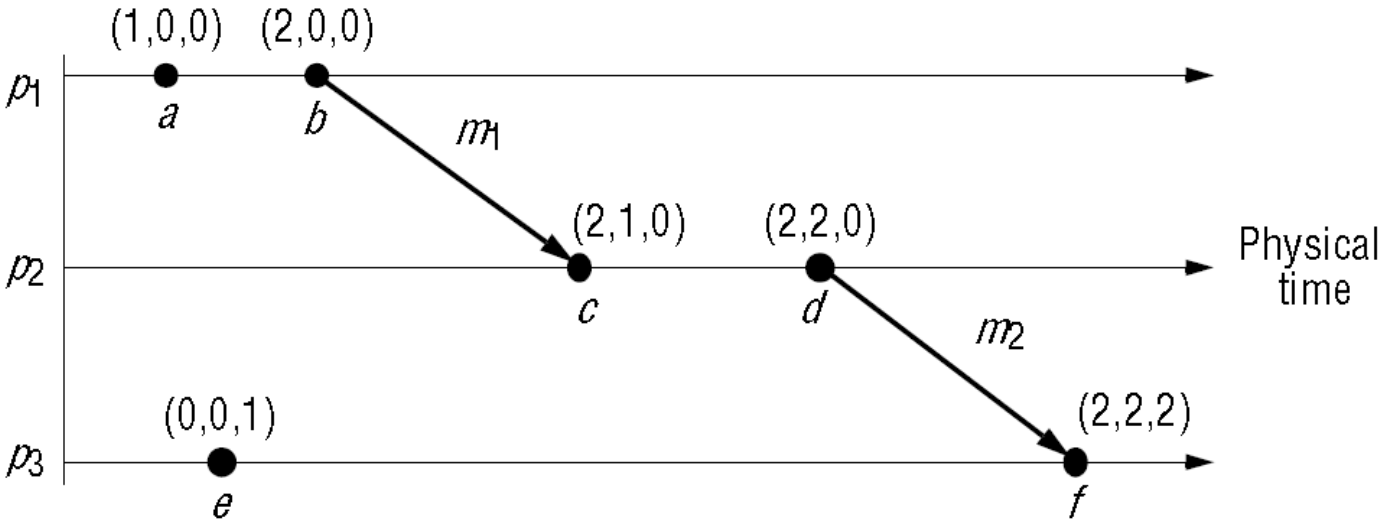
MYE017 Distributed Systems

Kostas Magoutis
magoutis@cse.uoi.gr
http://www.cse.uoi.gr/~magoutis

Causality

- Lamport clocks order all events in a distributed system
- If a *happened-before* b, then C(a) < C(b)
- However, the opposite is not true
- To achieve this, we need to introduce *vector clocks*

Vector clocks



Vector clocks

Each process P_i maintains a vector VC_i with two properties:

1. $VC_i[i]$ is the number of events that have occurred so far at P_i i.e., $VC_i[i]$ is the local logical clock at process P_i

2. If $VC_i[j] = k$ then P_i knows that k events have occurred at P_j. It is thus P_i's knowledge of the local time at P_j .

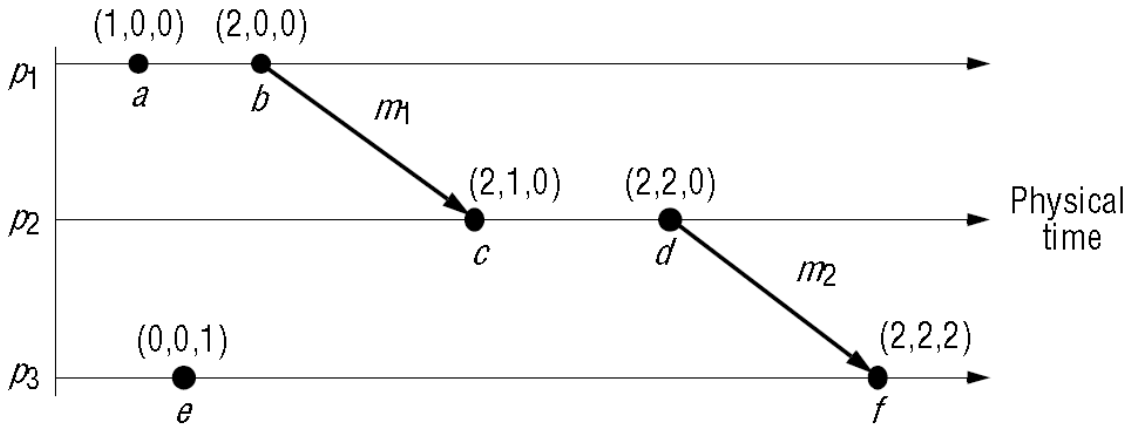
Vector clocks

1. Initially, $VC_i[j] \leftarrow 0$ for i, j = 1, 2, … N

2. When process P_i sends a message *m* to P_j, it executes
   - $VC_i[i] \leftarrow VC_i[i] + 1$
   - Sets $ts(m) \leftarrow VC_i$

3. On receipt of a message *m*, process P_j adjusts its own vector by setting
   $VC_j[k] \leftarrow \max\{VC_j[k], ts(m)[k]\}$ for each *k*,
   updates its vector clock $VC_j[j] \leftarrow VC_j[j] + 1$,
   and delivers the message to the application

Comparing VCs



$V = V' \ \textit{iff} \ V[j] = V'[j] \ \text{for} \ j = 1, 2 \ldots, N$

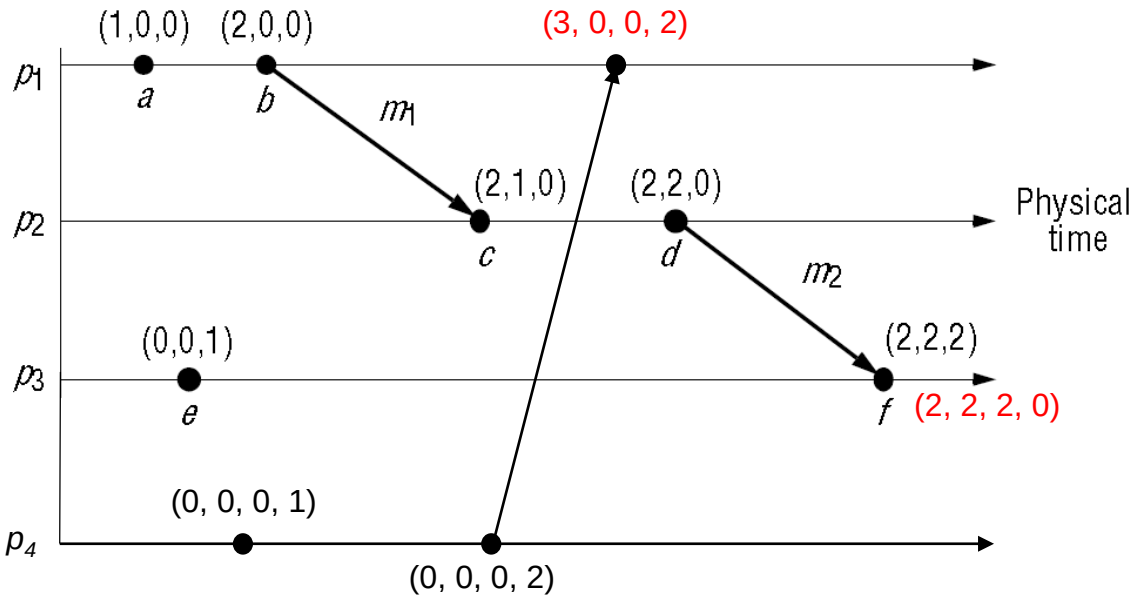
$V \leq V' \ \textit{iff} \ V[j] \leq V'[j] \ \text{for} \ j = 1, 2 \ldots, N$

$V < V' \ \textit{iff} \ V \leq V' \wedge V \neq V'$

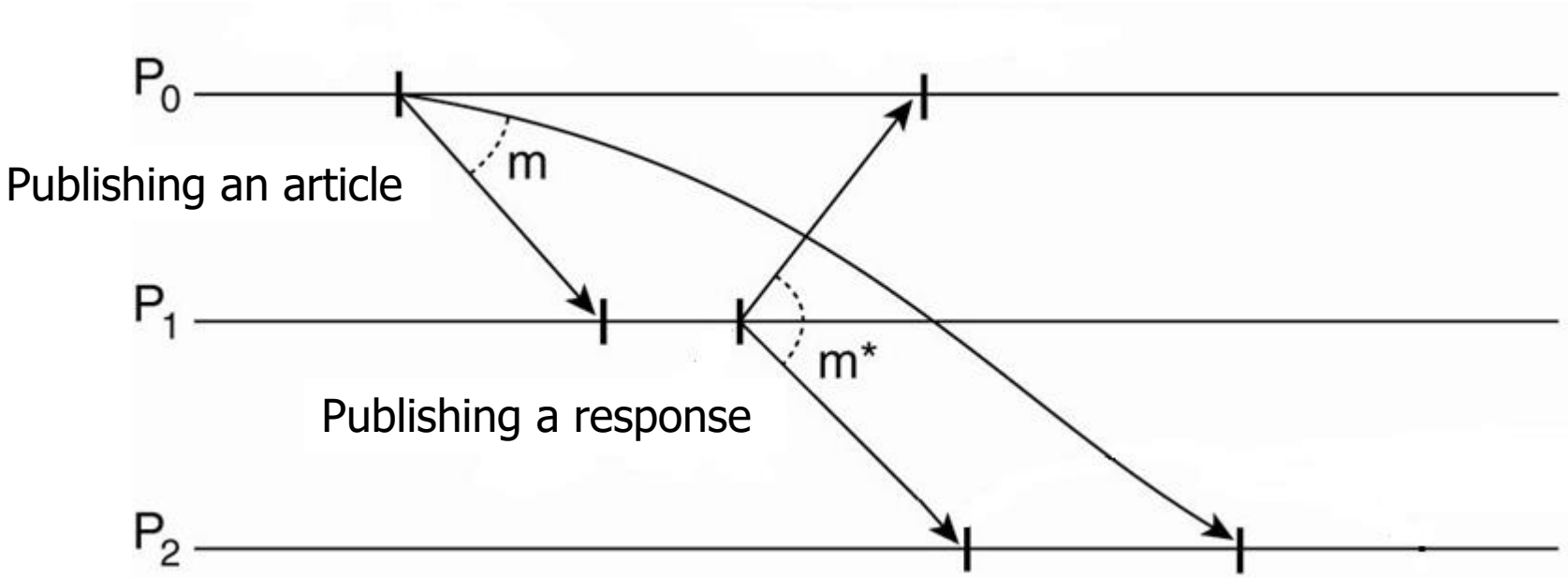
VC(e1) < VC(e2) means that e1 causally precedes e2

If neither VC(e1) < VC(e2) nor VC(e2) < VC(e1) then e1 is concurrent with e2

Comparing VCs



Causality in multicasts



- We know m* may have been *caused by m everywhere,*
- We can use vector clocks to capture this

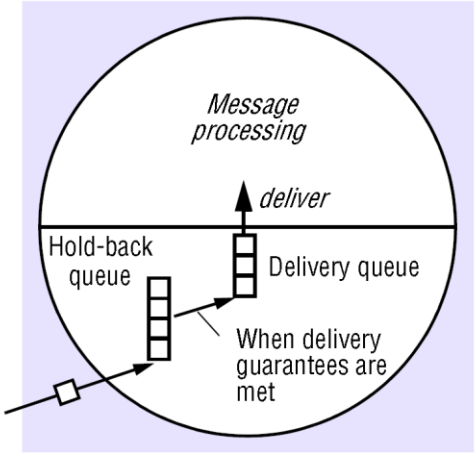
Causal ordering using VCs

1. Initially, $VC_i[j] \leftarrow 0$ for i, j = 1, 2, … N

2. When P_i sends *m* to other processes, it executes
   - $VC_i[i] \leftarrow VC_i[i] + 1$
   - Sets $ts(m) \leftarrow VC_i$

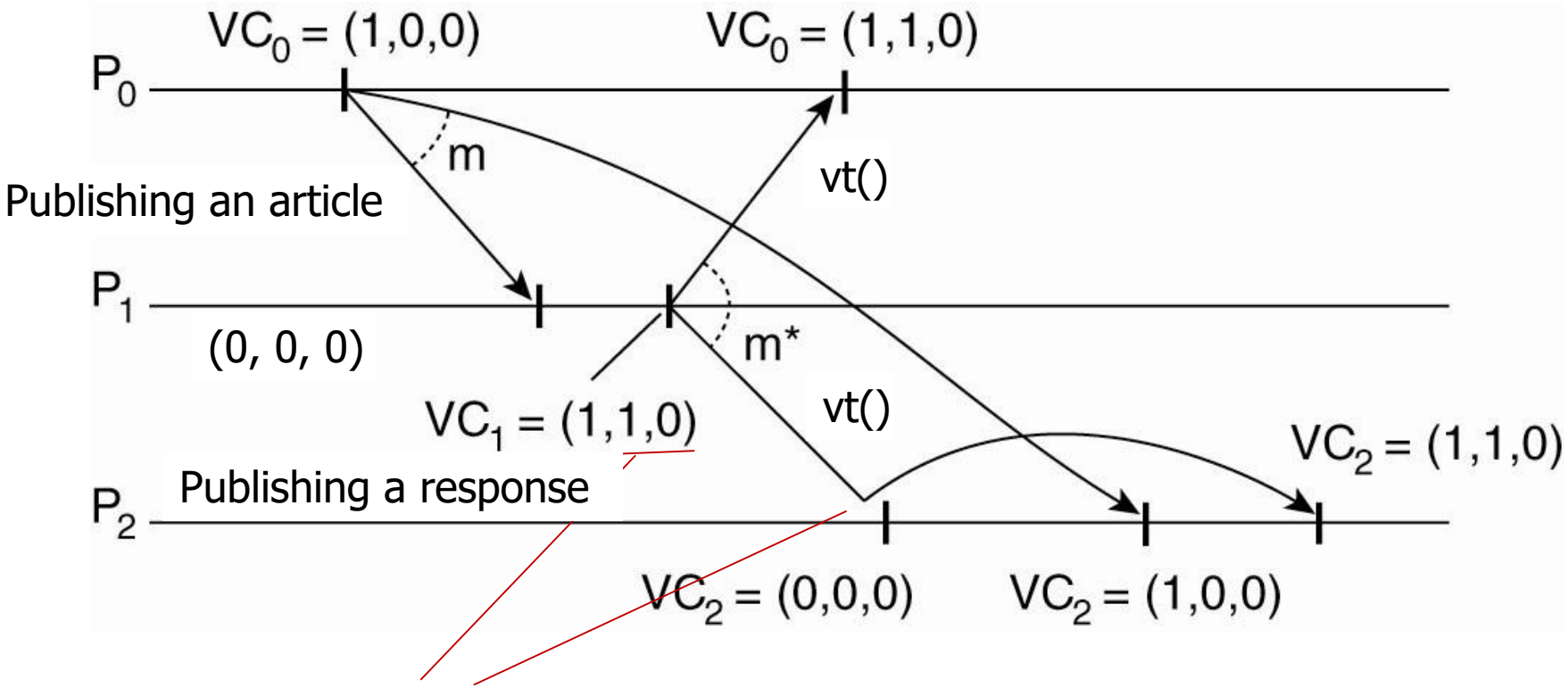
3. On receipt of a message *m* from P_i, process P_j :
   1. Places (*m*, ts(*m*)) in hold-back queue
   2. Wait until
      - $VC_j[i] = ts(m)[i] + 1$
      - $VC_j[k] \geq ts(m)[k]$ for each $k \neq i$
   3. Deliver *m*
   4. Update $VC_j[i]$ $(VC_j[i] = VC_j[i] + 1)$

Other causally precedent events will have been delivered before that, increasing the corresponding entries



Enforcing Causal Communication

Consider that $VC_i(i)$ increases only when P_i sends a message



P1 has seen a message that P2 has not yet seen: Do not deliver

Deliver m* only if:

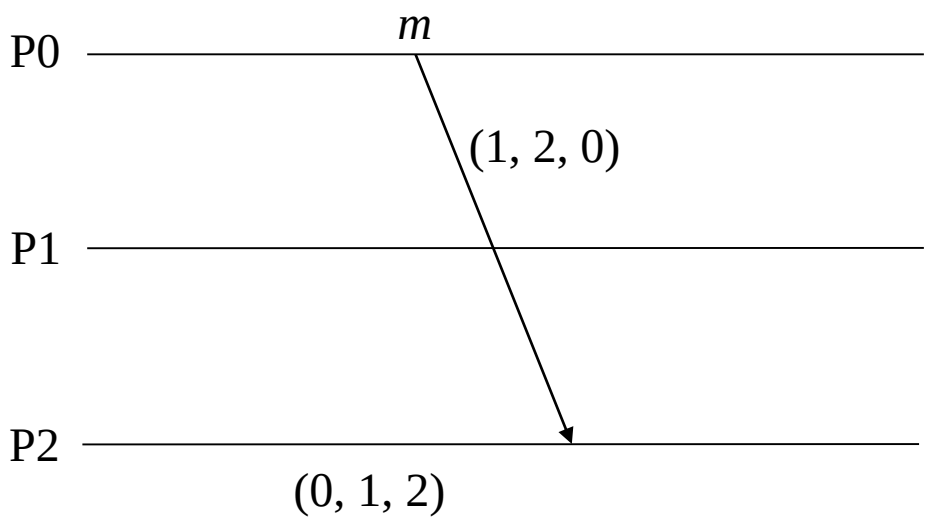
1. $ts_1(1) = VC_2(1) + 1$: m* is the next message expected from P1
2. $ts_1(0) \leq VC_2(0)$: P2 has seen all messages seen by P1 when sending m*

Exercise 2

Three processes 0, 1, 2 of a group communicate with one another, and the requirement is *causal order multicast*. A message from process 0 has a vector time stamp (1, 2, 0), and it reaches node 2 when its local vector clock is (0, 1, 2).

a. Draw a diagram reconstructing the exchange of all the messages in the group.

b. Will the message be accepted by process 2? Explain.



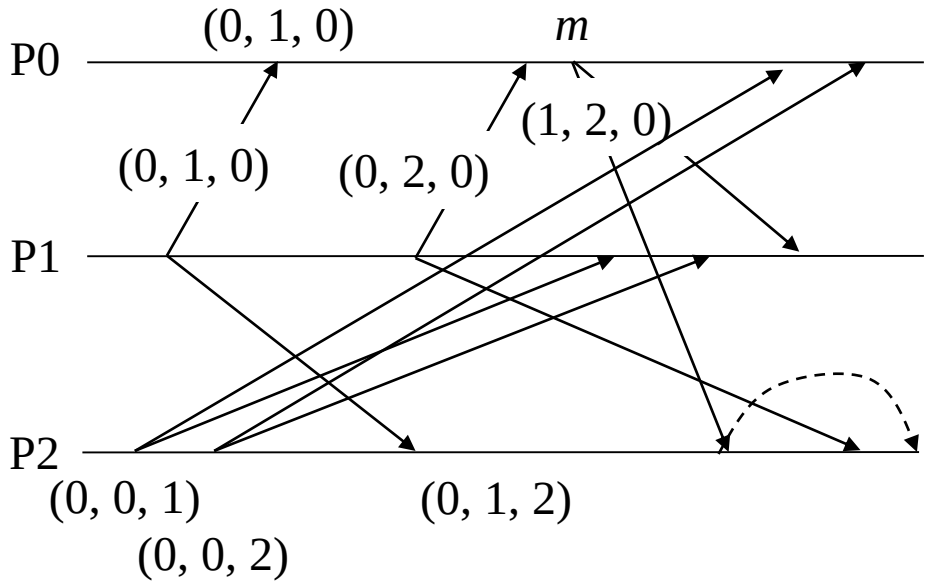
From Sukumar Ghosh, Distributed Systems: An Algorithmic Approach, Second Edition, CRC Press

Exercise 2

Three processes 0, 1, 2 of a group communicate with one another, and the requirement is *causal order multicast*. A message from process 0 has a vector time stamp (1, 2, 0), and it reaches node 2 when its local vector clock is (0, 1, 2).

a. Draw a diagram reconstructing the exchange of all the messages in the group.

b. Will the message be accepted by process 2? Explain.



Next expected from P0: 1

P2 must have seen what P0 has seen: (0, 2, 0)

Uses of vector clocks