
Vector Clocks, Causally Ordered Multicast

31 May 2026
Lecture 11

Slide Credits: Maarten van Steen

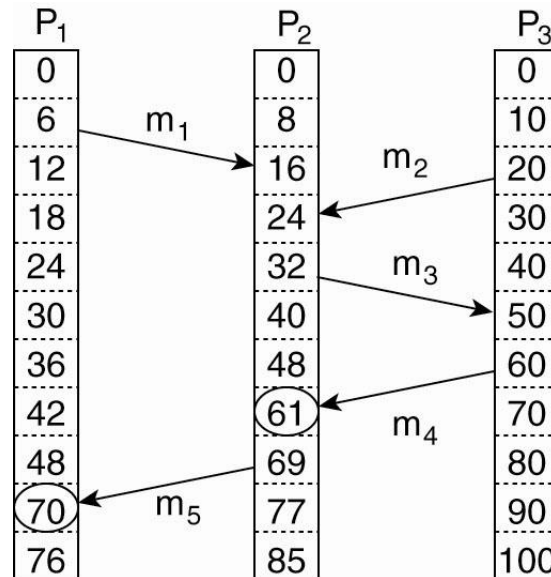
Topics for Today

- (Mattern) Vector Clocks
- Causally Ordered Multicast

Source: TvS 6.2-6.4

Vector Clocks

Observation: Lamport's clocks do not guarantee that if $C(a) < C(b)$ that a causally preceded b :



Observation:

Event a : m_1 is received at $T = 16$.

Event b : m_2 is sent at $T = 20$.

We **cannot** conclude that a causally precedes b .

Vector Clocks

Solution:

- Each process P_i has an array $VC_i[1..n]$, where $VC_i[j]$ denotes the **number of events** that process P_i **knows have taken place** at process P_j
- When P_i sends a message m , it adds 1 to $VC_i[i]$, and sends VC_i along with m as **vector timestamp** $vt(m)$. Result: upon arrival, recipient knows P_i 's timestamp.
- When a process P_j **delivers** a message m that it received from P_i with vector timestamp $ts(m)$, it
 - 1) updates each $VC_j[k]$ to $\max\{VC_j[k], ts(m)[k]\}$ for each k
 - 2) increments $VC_j[j]$ by 1.

Question: What does $VC_i[j] = k$ mean in terms of messages sent and received?

Vector and Lamport Clocks

Lamport Clocks

Rule 1: Each process has its own version of the **global clock**

Rule 2: Each process increments its **global clock** version when it performs an internal event or sends a message (which includes a timestamp)

Rule 3: When a process receives a message from another process it updates its **global clock** version **if the received timestamp is larger.**

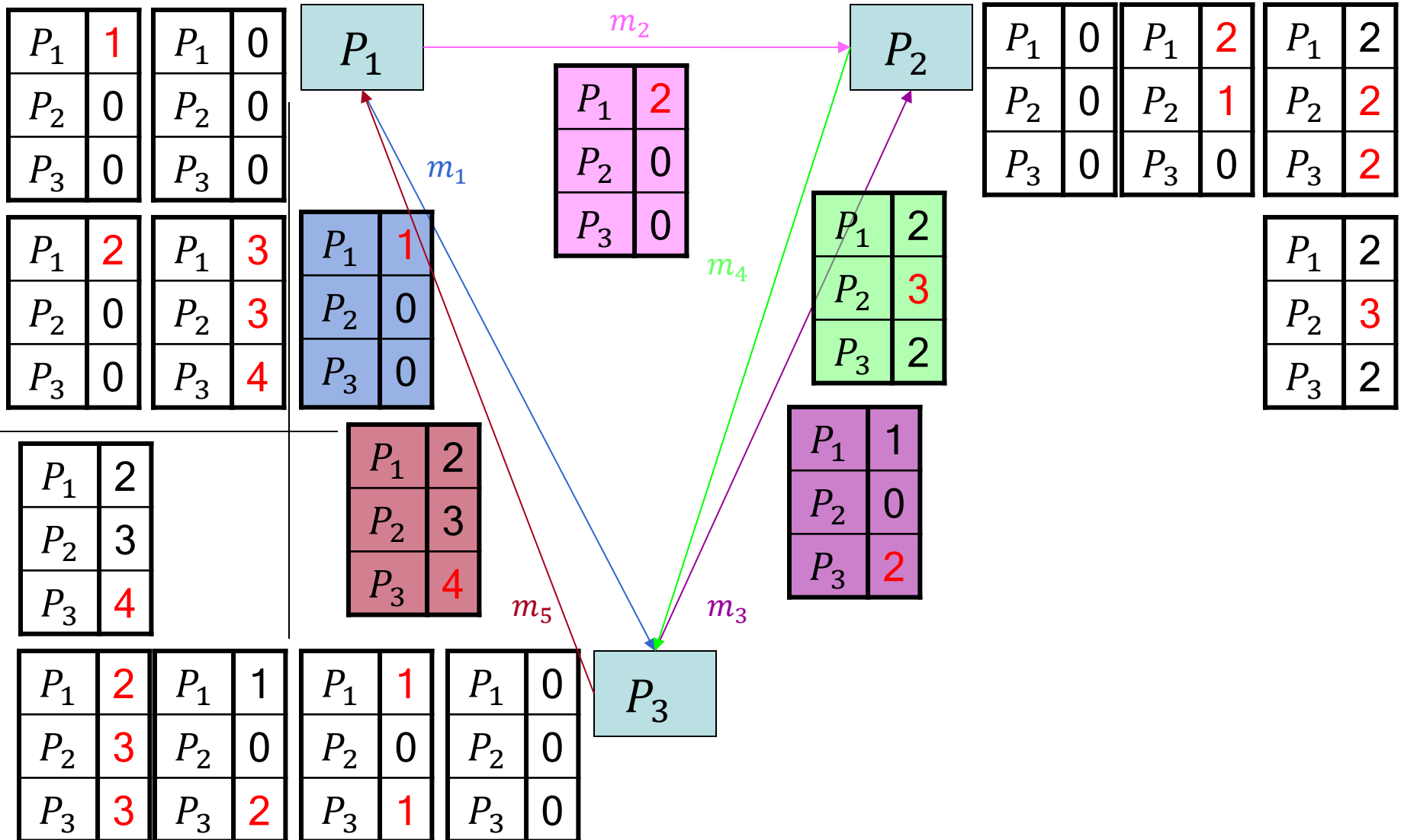
Vector clocks

Rule 1: Each process has its **own clock** and a version of **every other processes' clock.**

Rule 2: Each process increments its **own clock** when it sends or receives a message.

Rule 3: When a process receives a message from another process it updates its version of the **other clocks' timestamps** **if the received timestamp is larger**

Vector Clock Example



Vector Clock Example

m_1

P_1	1
P_2	0
P_3	0

m_4

P_1	2
P_2	3
P_3	2

m_2

P_1	2
P_2	0
P_3	0

m_5

P_1	2
P_2	3
P_3	4

m_3

P_1	1
P_2	0
P_3	2

1. $m_1 < m_2$
2. $m_1 < m_3$
3. $m_1 < m_4$
4. $m_1 < m_5$

5. $m_2 < > m_3$
6. $m_2 < m_4$
7. $m_2 < m_5$

8. $m_3 < m_4$
9. $m_3 < m_5$

10. $m_4 < m_5$

So Far

- (Mattern) Vector Clocks
- Causally Ordered Multicast

Causally Ordered Multicasting

Observation: We can now ensure that a message is delivered only if all causally preceding messages have already been delivered.

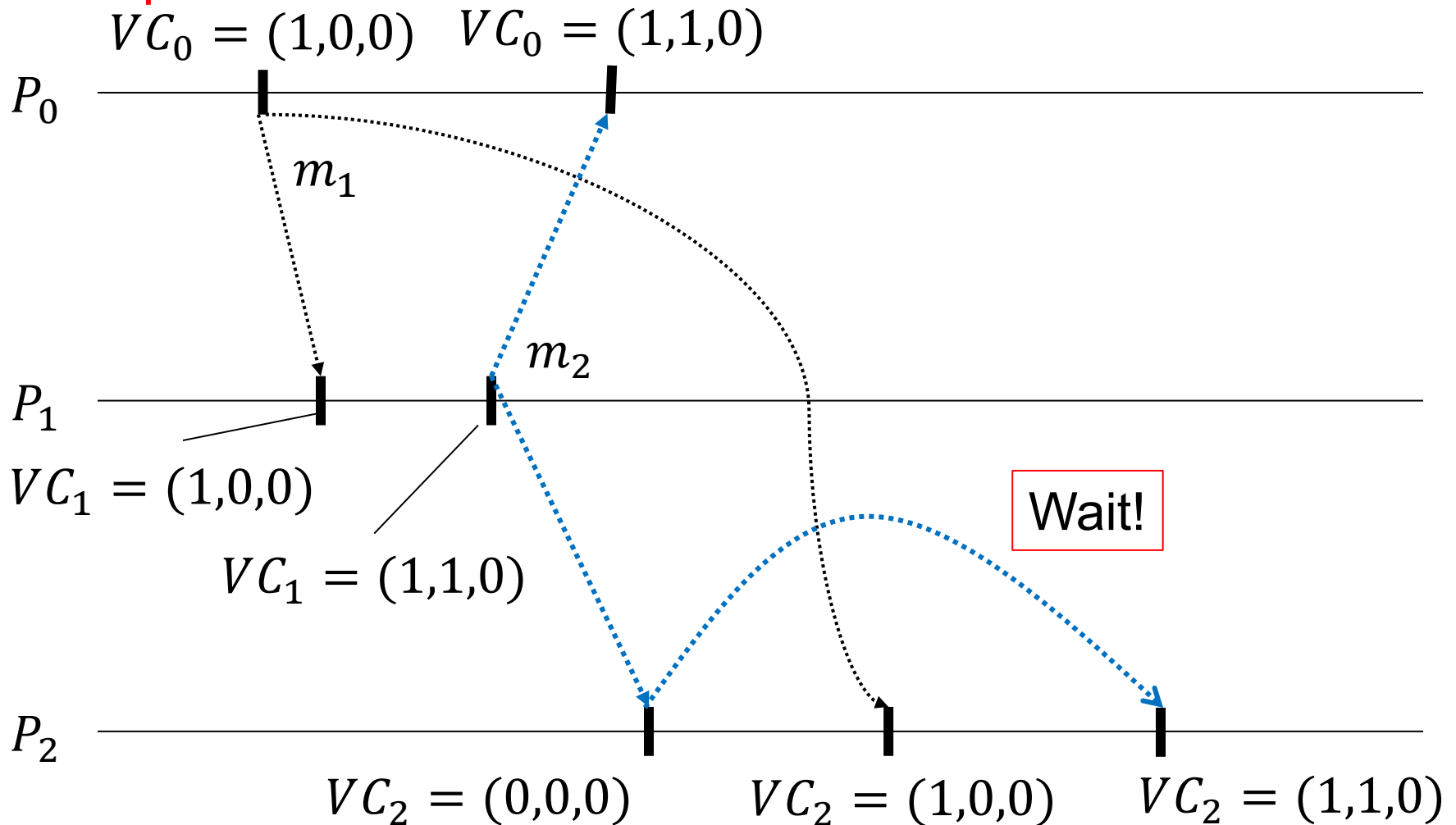
Adjustment: P_i increments $VC_i[i]$ only when sending a message, and P_j “adjusts” VC_j when receiving a message (i.e., effectively does not change $VC_j[j]$).

P_j postpones delivery of m until:

- $ts(m)[i] = VC_j[i] + 1.$
- $ts(m)[k] \leq VC_j[k]$ for $k \neq i$

Causally Ordered Multicasting

Example 1:



Causally Ordered Multicasting

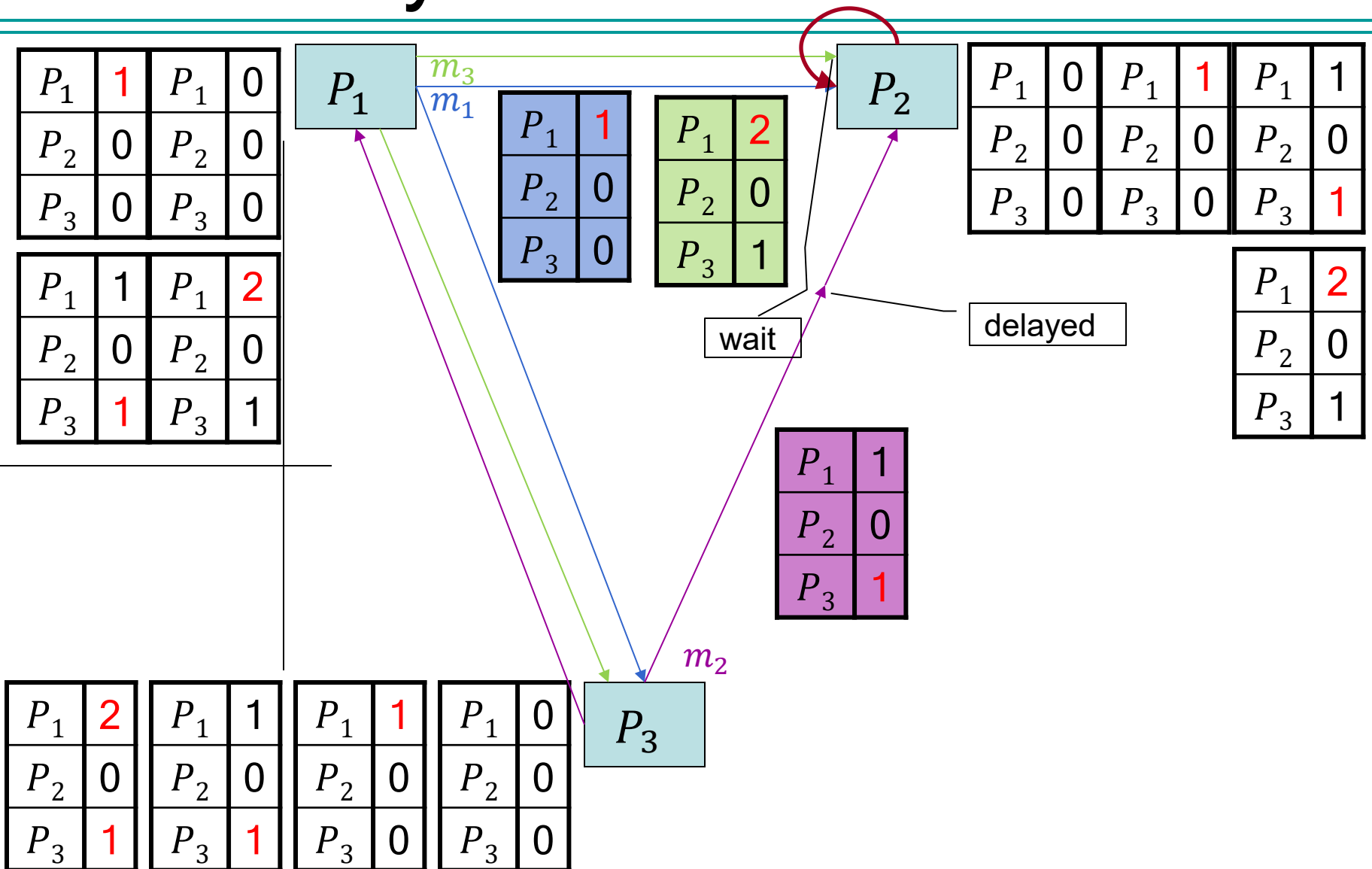
Example 2:

Take $VC_2 = (0,2,2)$ at P_2

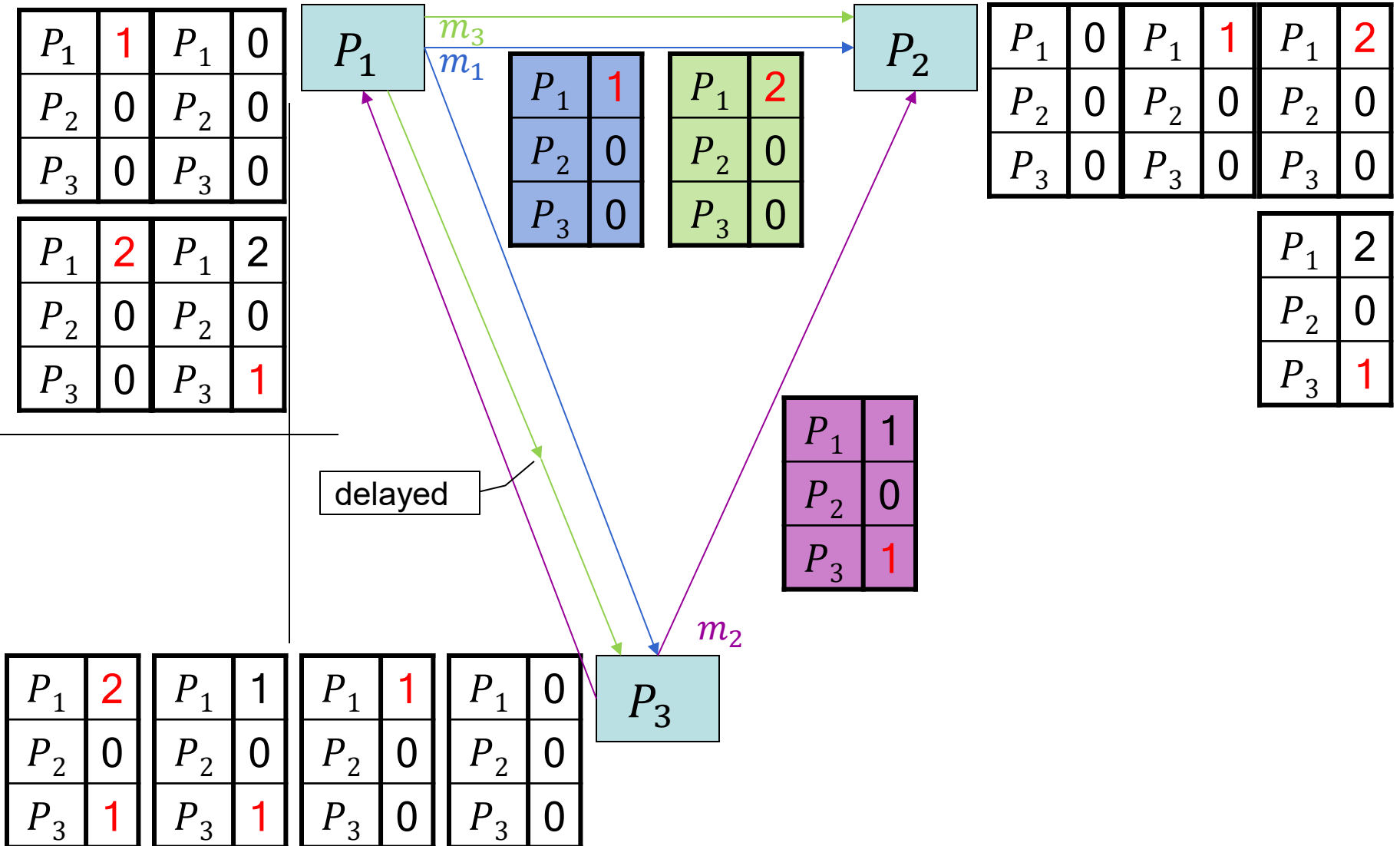
P_2 receives a message with $ts(m) = (1,3,0)$ from P_0 .

What information does P_2 have, and what will it do when it receives m (from P_0)?

Causally Ordered Multicast 1



Causally Ordered Multicast 2



Vector Clocks and COM

Vector clocks

Rule 1: Each process has its **own clock** and a version of **every other processes' clock**.

Rule 2: Each process increments its **own clock** when it sends or receives a message.

Rule 3: When a process receives a message from another process it updates its version of the **other clocks' timestamps** if the received timestamp is larger

Causally Ordered Multicast

Rule 1: Each process has its **own clock** and a version of **every other processes' clock**.

Rule 2: Each process increments its own clock when it sends a message.

Rule 3: When a process receives a message from another process it updates its version of the **sender's timestamp**.

Rule 4: A message is delivered only if it is “next in line”:

1. It's the next expected one for the sender
2. The message's timestamp is less than or equal to the local clock.

Conclusion

- (Mattern) Vector Clocks
- Causally Ordered Multicast