

Feb 4
Consistent Global States
(Part 2)

Vijay Chidambaram

Consistent observations

- A consistent observation is one that corresponds to a consistent run.
- It is the possibility of messages being reordered by channels that leads to undesirable observations
- We can restore order to messages between pairs of processes by defining a delivery rule for deciding when received messages are to be presented to the application process.

FIFO Delivery

- FIFO Delivery: $\text{sendi}(m) \rightarrow \text{sendi}(m') \Rightarrow \text{deliveri}(m) \rightarrow \text{deliveri}(m')$
- Clock Condition: $e \rightarrow e' \Rightarrow \text{Timestamp}(e) < \text{timestamp}(e')$
- Delivery Rule DR1: At time t , deliver all received messages with timestamps up to $t - \text{delta}$ in increasing timestamp order

Gap Detection

- Given two events e and e' along with their clock values $LC(e)$ and $LC(e')$ where $LC(e) < LC(e')$, determine whether some other event e'' exists such that $LC(e) < LC(e'') < LC(e')$
- A message m received by process p is called stable if no future messages with timestamps smaller than $TS(m)$ can be received by p .
- DR2: Deliver all received messages that are stable at p_0 in increasing timestamp order.

Causal Delivery

- FIFO delivery is per-process
- Causal Delivery extends it across processes
- Causal Delivery: $\text{send}_i(m) \rightarrow \text{send}_j(m') \Rightarrow \text{deliver}_i(m) \rightarrow \text{deliver}_j(m')$
 - Note i and j can be different
- FIFO delivery between all pairs of processes not enough for causal delivery
- $A \rightarrow B, A \rightarrow C; B \rightarrow C$
 - $\text{send}_{AB}(M1) \rightarrow \text{send}_{AC}(M2)$
 - B gets M1, sends it to C
 - C gets M2 before M1.
 - FIFO delivery is ensured, causal delivery is not.

Consistent Observations

- If the observer process uses a delivery rule that satisfies Causal Delivery, then all its observations will be consistent
- How do we build this in a practical distributed system?

Building the Observer

- Assume Observer has two parts: the Logic Controller and the Network Controller
- The Network Controller gets events sent by other nodes, decides in what order to show them to Logic Controller
 - Delivery rules are implemented here
- Logic Controller takes actions based on what it sees ("declare deadlock")

Building the Observer

- Network Controller gets two events E1 and E2
- Timestamp = TS
- $TS(E1) < TS(E2)$
- This doesn't mean $E1 < E2$
- Only this is true: $E1 < E2 \Rightarrow TS(E1) < TS(E2)$
- So we know $E2 < E1$ is false, E1 and E2 could still be concurrent

Strong Clock Condition

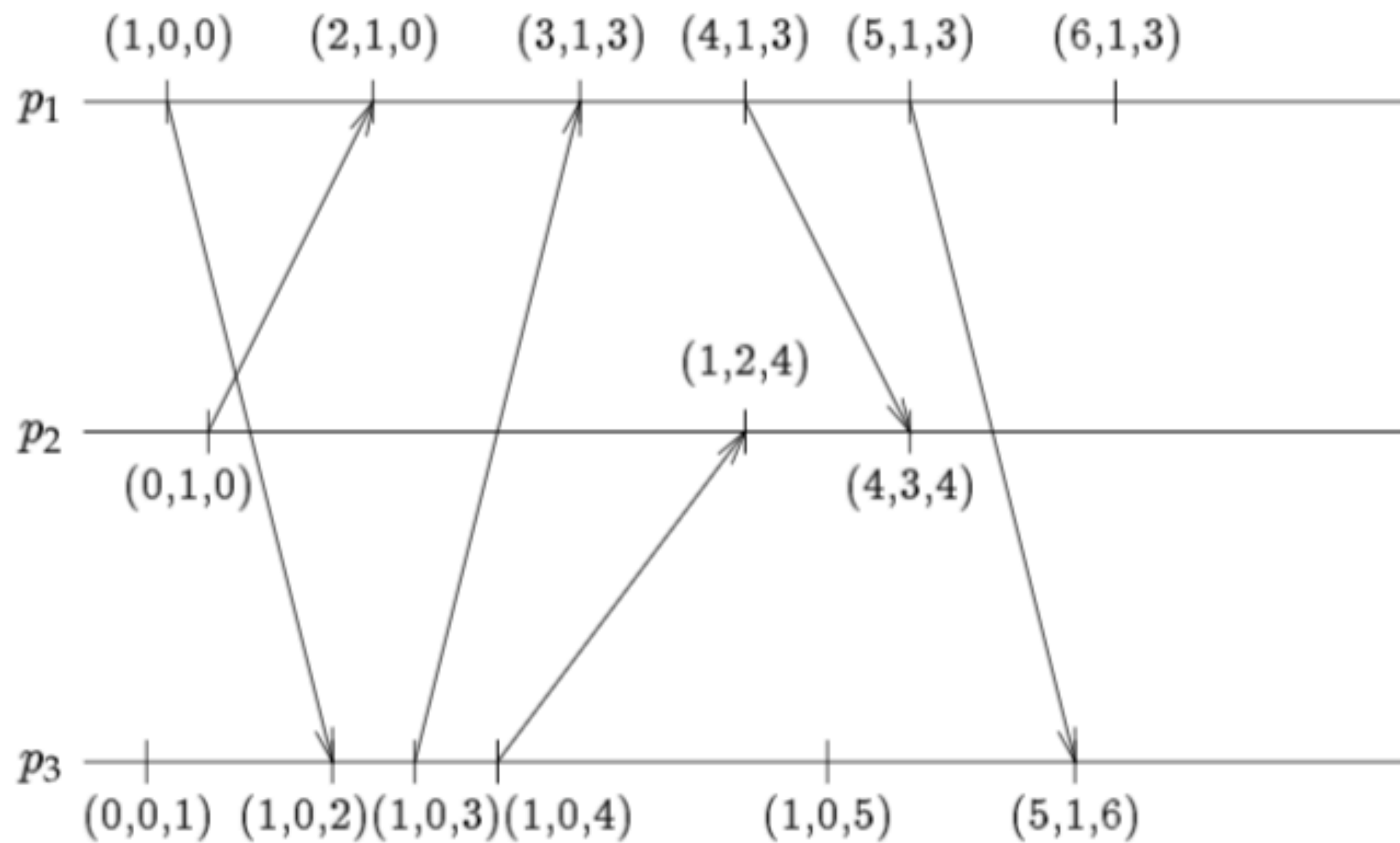
- Need stronger condition to implement Network Controller
- Strong Clock Condition:
 - $E1 < E2 \Rightarrow TS(E1) < TS(E2)$
 - $TS(E1) < TS(E2) \Rightarrow E1 < E2$

Implementing the Strong Clock Condition

- Brute force approach:
 - At each node, keep the Causal History C
 - Causal History $C(e)$ is the set of all events e' such that $e' < e$
 - All previous local events at node are part of $C(e)$
 - When A sends a message to B , $C(\text{receipt of message } M) = C(\text{previous local event at } B) \cup C(\text{sending event at } A)$
 - $E1 < E2$ if $C(E1)$ is a subset of $C(E2)$
- Problem: the set C grows too large too quickly, impractical to maintain

Vector Clocks

- Each node maintains a vector $V[n]$ where there are n nodes
- $V = 0$ on initialization for all nodes
- For local event E_i at node i , update $V[i] = V[i] + 1$
- On receipt(M),
 - $V = \max(V, \text{vector-TS}(M))$ for each element of V
 - $V[i] += 1$
- $V[j]$ = number of events of j that casually precede this event (when this process is not j)
- $V[i]$ = number of local events when this process is i



Using Vector Clocks

- $V1 < V2$ if no element of $V2$ is smaller than its corresponding element in $V1$
- Strong Clock Condition: $E1 < E2 \iff V(E1) < V(E2)$
- $V1$ and $V2$ are concurrent if $V1[i] > V2[i]$ for some i and $V2[j] > V1[j]$ for some j

Using Vector Clocks in the Network Observer

- All events sent to observer have vector timestamps
- Network Observer obtains set of events M , then decides on order to deliver them
- A message can be delivered as soon as Network Observer determines there are no events that casually precede this message
- Consider message M from J
- M' is last message delivered from K ($K \neq J$)
- To deliver M , NO has to determine that:
 - There is no earlier message from J that is undelivered
 - if $V(M)[J] - 1$ message have already been delivered, there cannot be an earlier message
 - There is no undelivered message M'' such that $\text{send}_K(M') < \text{send}(M'') < \text{send}_J(M)$
 - $V(M')[K] \geq V(M)[K]$ for all $K \Rightarrow V(M) < V(M')$

Using Vector Clocks in the Network Observer

- Network Observer maintains array D , initialized to 0
- Deliver message M with time stamp V from J as soon as:
 - $D[J] = V[J] - 1$
 - $D[K] \geq V[K]$, for all $K \neq J$
- When Network Observer delivers M , D is updated by setting $D[j] = V[j]$

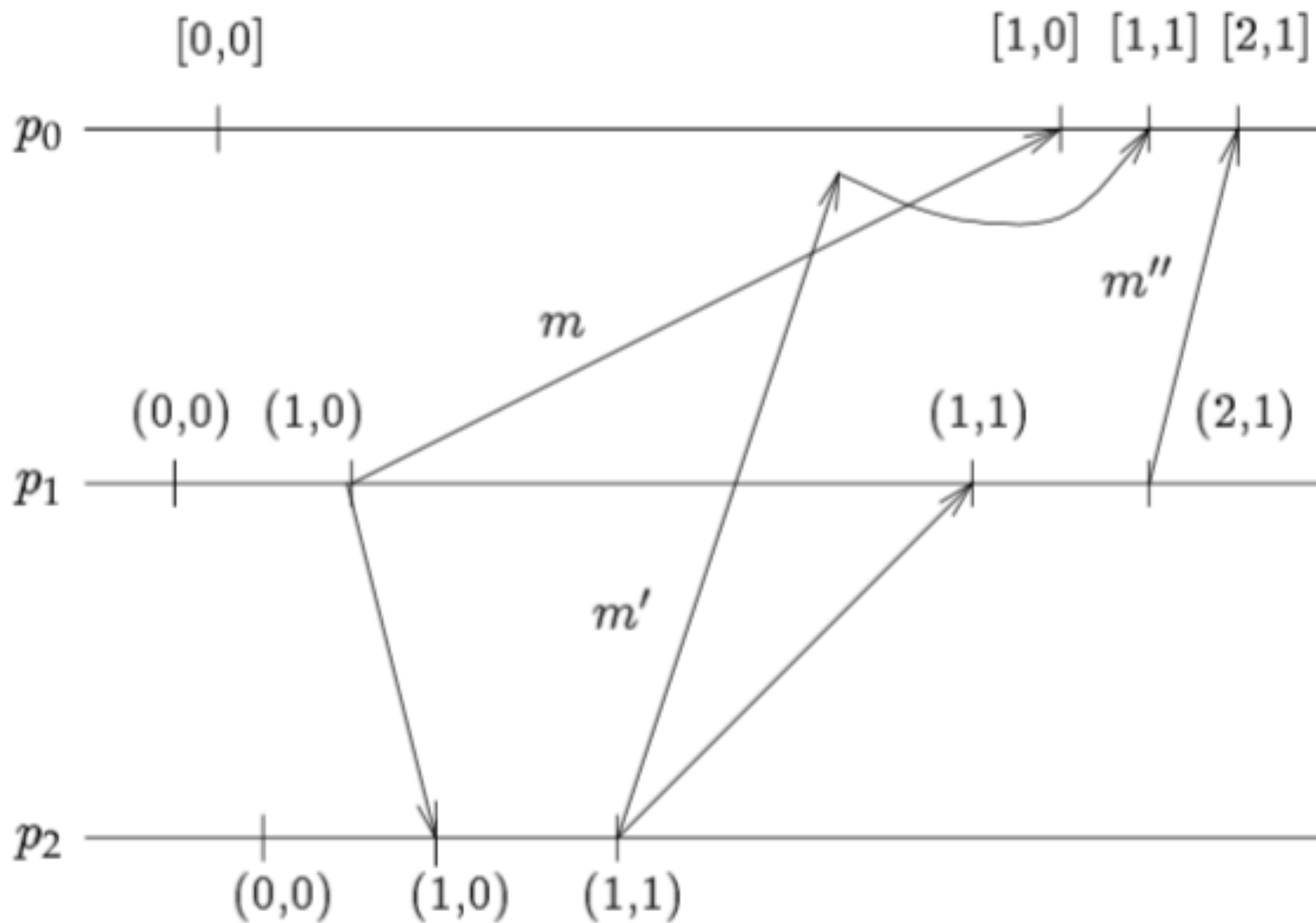


Figure 8. Causal Delivery Using Vector Clocks