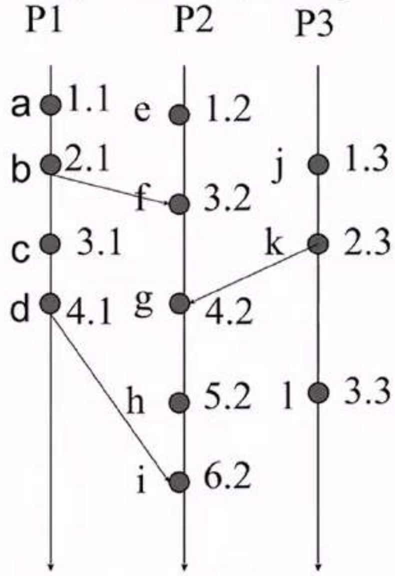
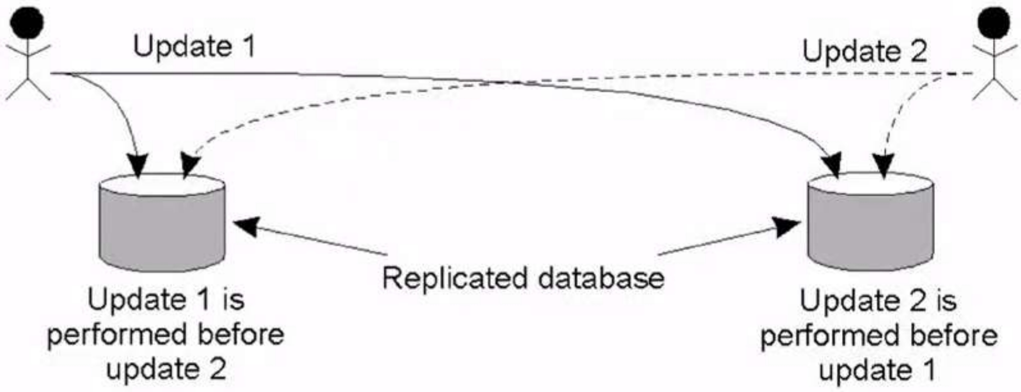
Total Order

- Create total order by attaching process number to an event. If time stamps match, use process # to order



Example: Totally-Ordered Multicasti

- Updating a replicated database and leaving it in an inconsistent state.



Prashant Shen...

Algorithm

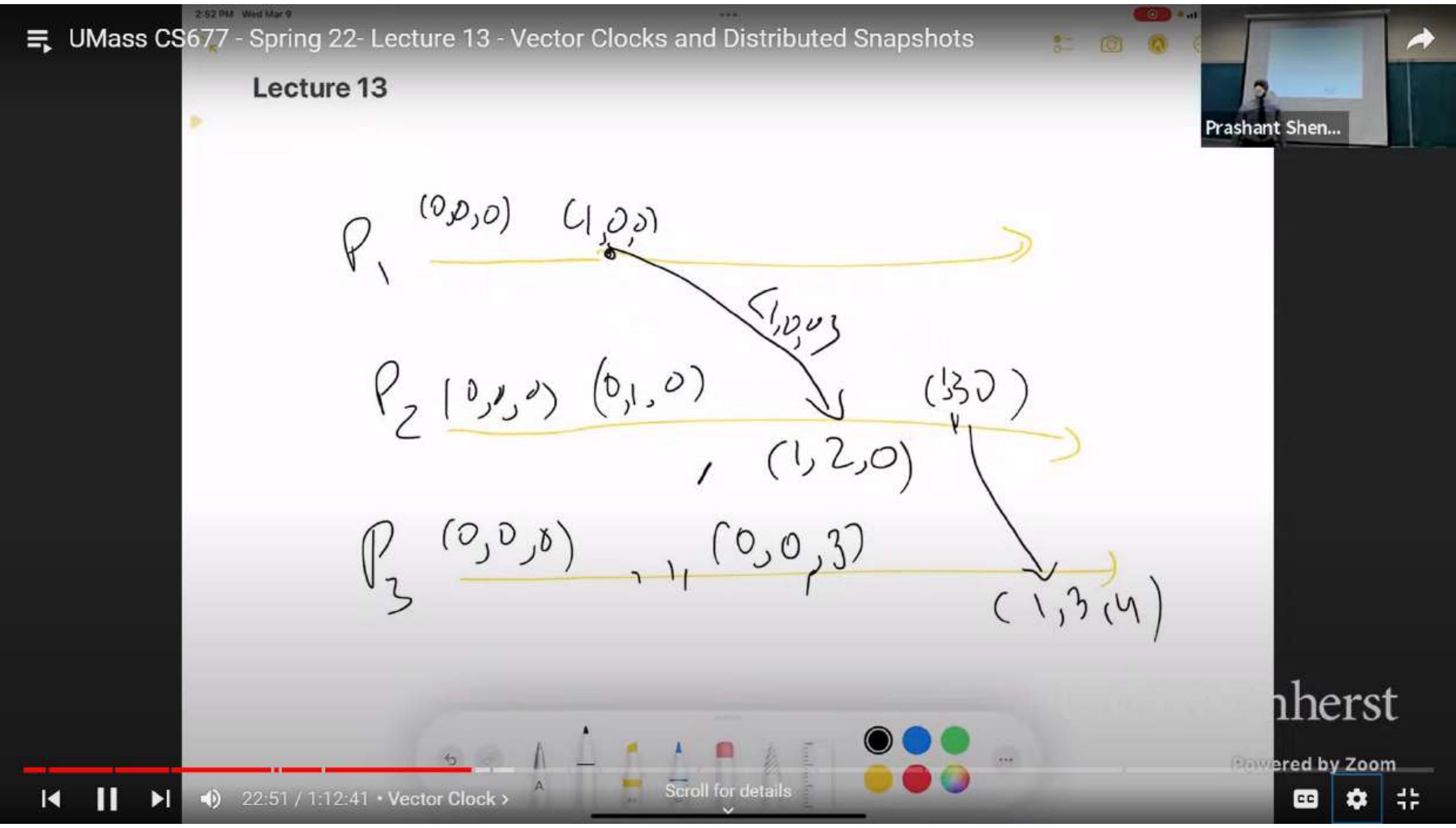
- Totally ordered multicasting for banking example
  - Update is timestamped with sender's logical time
  - Update message is multicast (including to sender)
  - When message is received
    - It is put into local queue
    - Ordered according to timestamp,
    - Multicast acknowledgement
  - Message is delivered
    - It is at the head of the queue
    - IT has been acknowledged by all processes
    - P_i sends ACK to P_j if
      - P_i has not made a request
      - P_i update has been processed and P_i's ID > P_j's Id

Causality

- Lamport's logical clocks
  - If $A \rightarrow B$ then $C(A) < C(B)$
  - Reverse is not true!!
    - Nothing can be said about events by comparing time-stamps!
    - If $C(A) < C(B)$, then ??
- Need to maintain *causality*
  - If a -> b then a is casually related to b
  - *Causal delivery*:If send(m) -> send(n) => deliver(m) -> deliver(n)
  - Capture causal relationships between groups of processes
  - Need a time-stamping mechanism such that:
    - If $T(A) < T(B)$ then A should have causally preceded B

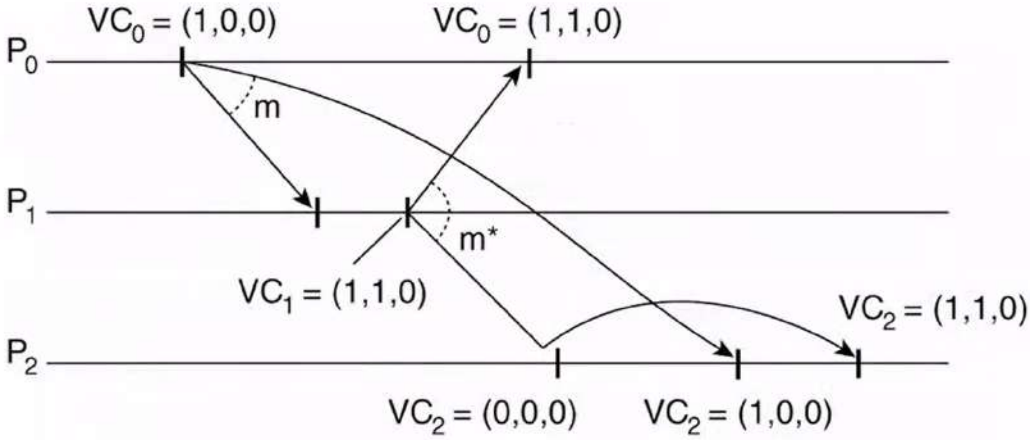
Vector Clocks

- Each process *i* maintains a vector V_i
  - $V_i[i]$: number of events that have occurred at i
  - $V_i[j]$: number of events I knows have occurred at process j
- Update vector clocks as follows
  - Local event: increment $V_i[I]$
  - Send a message :piggyback entire vector V
  - Receipt of a message: $V_j[k] = \max(V_j[k], V_i[k])$
    - Receiver is told about how many events the sender knows occurred at another process *k*
    - Also $V_j[i] = V_j[i]+1$
- *Exercise:* prove that if $V(A)<V(B)$, then *A* causally precedes *B* and the other way around.


UMass CS677 - Spring 22- Lecture 13 - Vector Clocks and Distributed Snapshots
Lecture 13
P1 (0,0,0) (1,0,0)
(1,0,0)
P2 (0,0,0) (0,1,0) (1,3,0)
(1,2,0)
P3 (0,0,0) (0,0,3)
(1,3,4)
Prashant Shen...
Powered by Zoom
Scroll for details
22:51 / 1:12:41 • Vector Clock ›

Enforcing Causal Communicatio

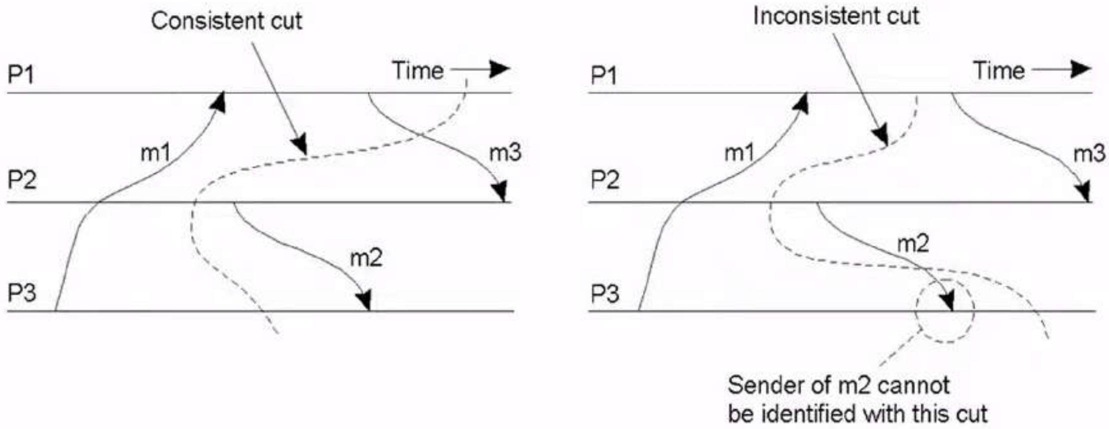
- Figure 6-13. Enforcing causal communication.



Global State

- Global state of a distributed system
  - Local state of each process
  - Messages sent but not received (state of the queues)
- Many applications need to know the state of the system
  - Failure recovery, distributed deadlock detection
- Problem: how can you figure out the state of a distributed system?
  - Each process is independent
  - No global clock or synchronization
- Distributed snapshot: a consistent global state

Global State (1)

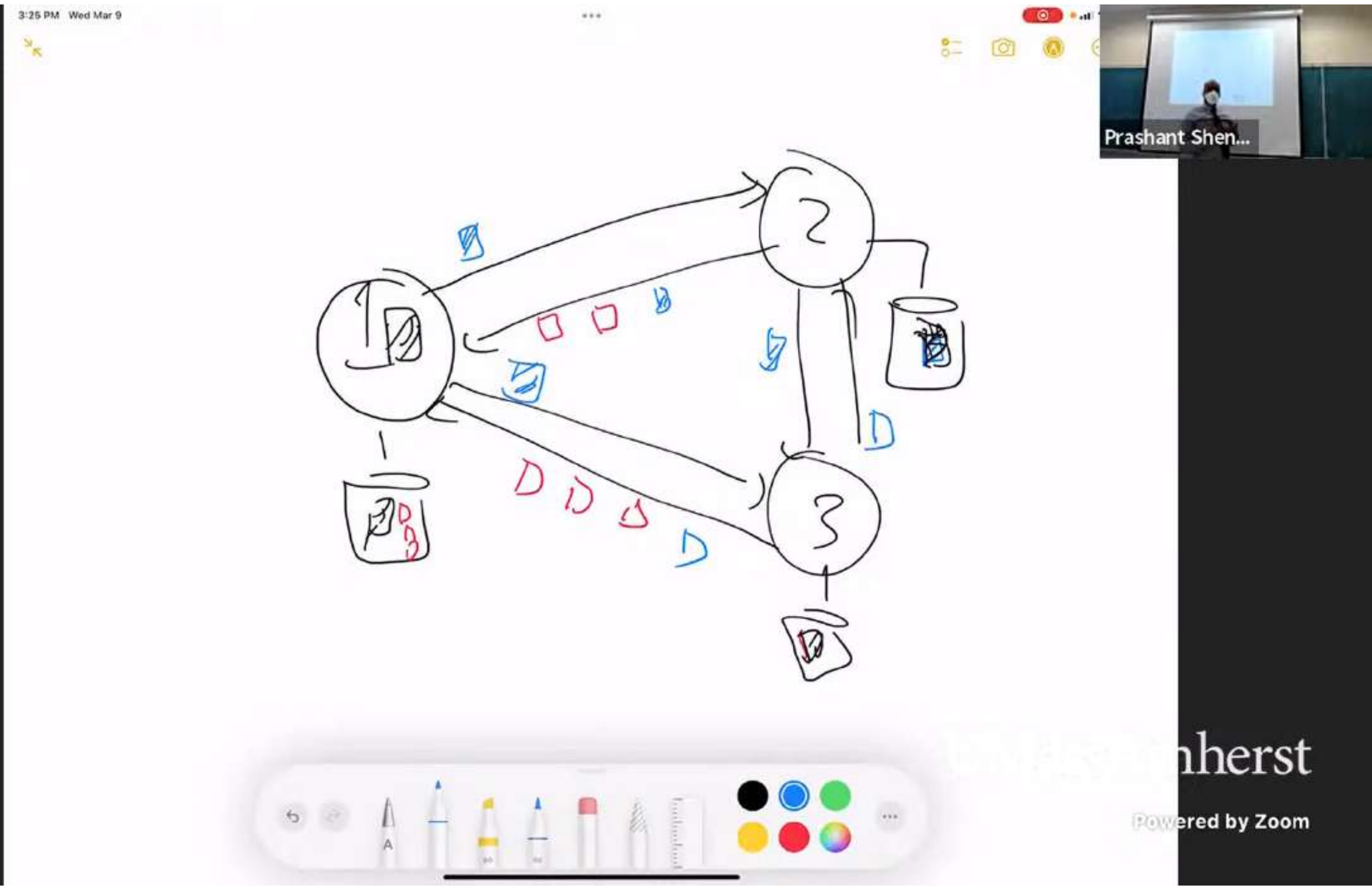


(a)

(b)

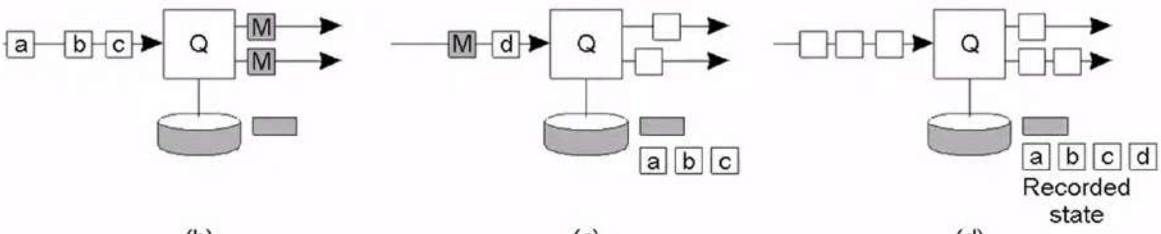
Distributed Snapshot Algorithm

- Assume each process communicates with another process using unidirectional point-to-point channels (e.g, TCP connections)
- Any process can initiate the algorithm
  - Checkpoint local state
  - Send marker on every outgoing channel
- On receiving a marker
  - Checkpoint state if first marker and send marker on outgoing channels, save messages on all other channels until:
  - Subsequent marker on a channel: stop saving state for that channel


3:25 PM Wed Mar 9
Prashant Shen...
1
2
3
Powered by Zoom

Snapshot Algorithm Example

b) Process Q receives a marker for the first time and records its local state
c) Q records all incoming message
d) *Q* receives a marker for its incoming channel and finishes recording the state of the incoming channel



(b) (c) (d)

Termination Detection

- Detecting the end of a distributed computation
- Notation: let sender be *predecessor*, receiver be *successor*
- Two types of markers: Done and Continue
- After finishing its part of the snapshot, process Q sends a Done or a Continue to its predecessor
- Send a Done only when
  - All of Q's successors send a Done
  - Q has not received any message since it check-pointed its local state and received a marker on all incoming channels
  - Else send a Continue
- Computation has terminated if the initiator receives Done messages from everyone