CSC2/458 Parallel and Distributed Systems
Clocks

Sreepathi Pai

March 22, 2018

URCS

Outline

Outline

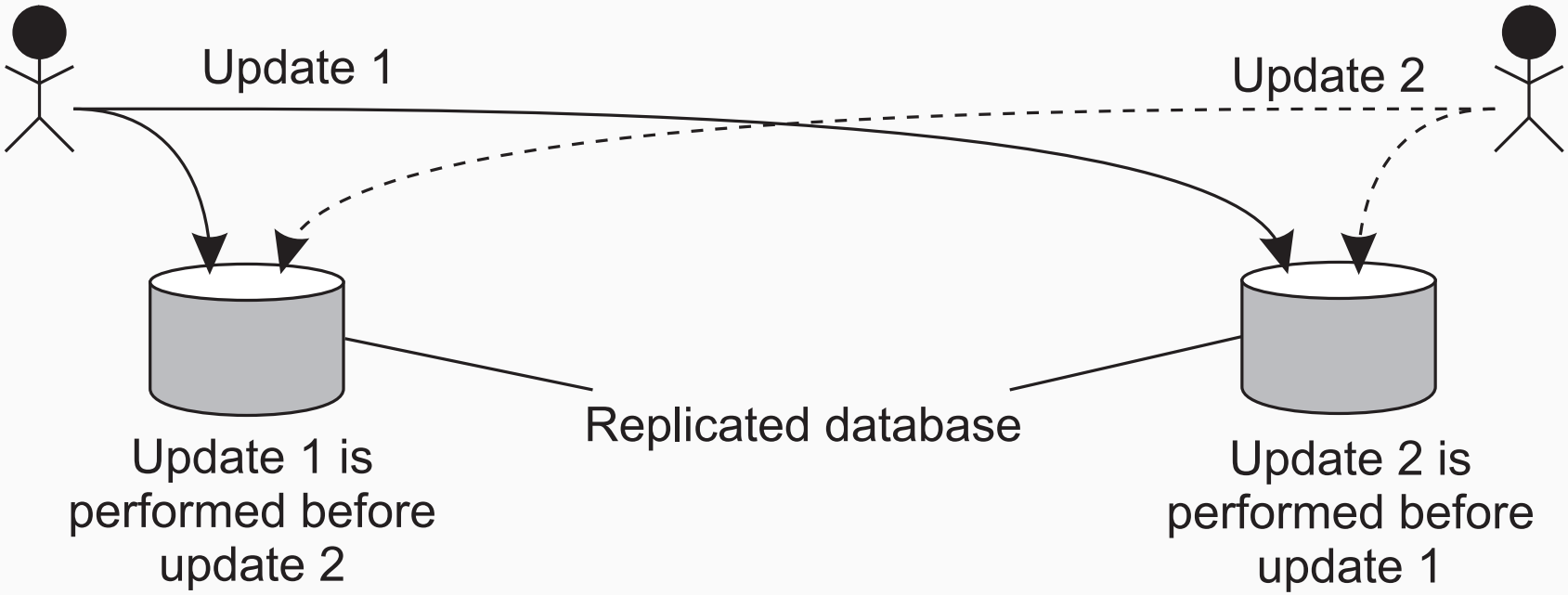
Continuity and Contingency Planning

Securities industry regulations require that brokerage firms inform their clients of their plans to address the possibility of a business disruption that potentially results from power outages, natural disasters, or other events. ... The program provides for continuation of client service **within minutes** in most cases.

- ...
- In the event that our primary data center became unavailable for any reason, we would transition to a separate back-up location, where account access would be made available. Our data centers are on separate power grids, separate flood plains and fault lines, and within different transportation networks.
- ...

Replicating Data

- (client1/update1) CREDIT $100
- (client2/update2) APPLY INTEREST 0.05%



Ensuring order with timestamps

- All clients timestamp their messages using a single clock.
- All replicas process messages in timestamp order.

Distributed clocks

- All clients timestamp their messages using the clocks nearest to them.
- All clocks need to synchronize with each other
- What are the problems with separate clocks?

TrueTime

- Assign timestamps to all database transactions
  - This is an interval
- Use GPS clocks
  - derive clock signal from GPS satellites (which carry atomic clocks)
- Use atomic clocks
  - paper refers to them as "armageddon" clocks
  - rubidium clocks are about $300
- Synchronization and drift are still issues
  - Transactions must be separated by some safe interval to be ordered
- Failures (including liars) are detected and such machines are "evicted"

Corbett et al., Spanner: Google's Globally-Distributed Database, OSDI 2012.

Outline

Logical Clocks: Premise

- Absolute time synchronization among distributed processes is not required
- Processes must agree on **order** in which events happen
    - not their time

Happens-before

$$a \rightarrow b$$

is read as event a happens-before event b

- In same process, $a \rightarrow b$ if a occurs before b.
  - for example, in program order
- If event a is "sending a message", and event b is "receiving that message", then $a \rightarrow b$.
  - Messages cannot be received before they are sent
  - Messages cannot be transmitted "instantaneously"

Properties of happens-before

- $a \rightarrow b$, $b \rightarrow c$
  - Is $a \rightarrow c$?
- If neither $a \rightarrow b$ nor $b \rightarrow a$ hold, then a and b are concurrent

Implementing happens-before

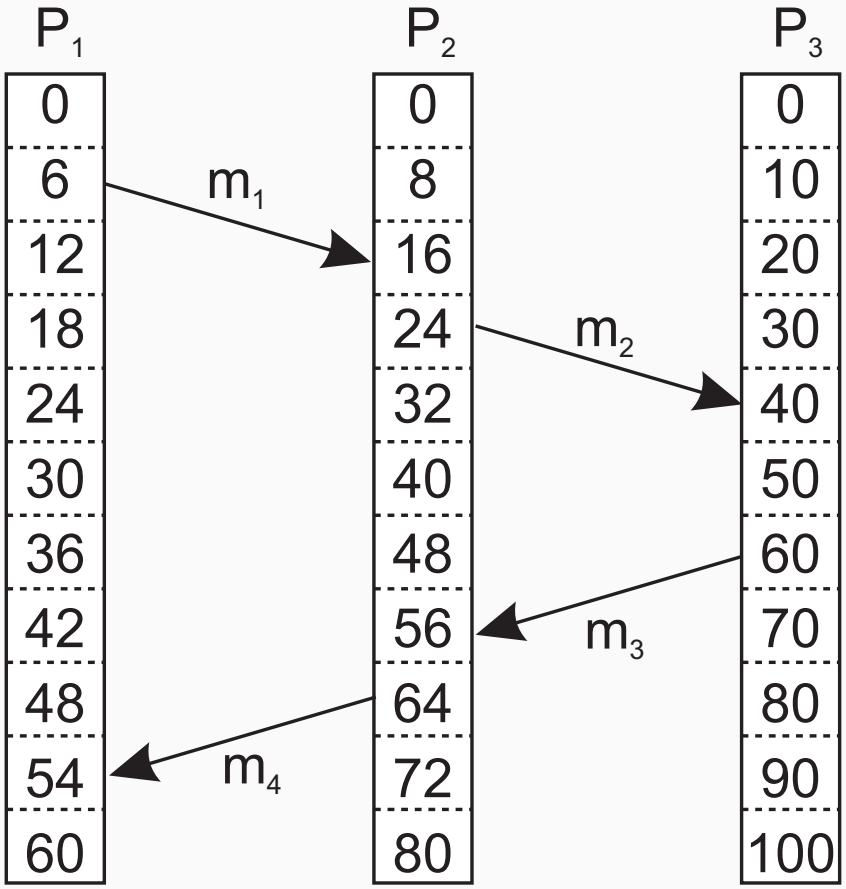
- Assume each process has a monotonic clock
  - Not synchronized
- Now, each event x is assigned a timestamp according to the local process
  - $C(a)$ is timestamp of event a
- Assume $a \rightarrow b$ (e.g. event a is sending of message, event b is receiving of message)
  - $C(a)$ timestamp assigned by sender
  - $C(b)$ timestamp assigned by receiver
  - What relationship must hold between $C(a)$ and $C(b)$?

Implementing happens-before with timestamps

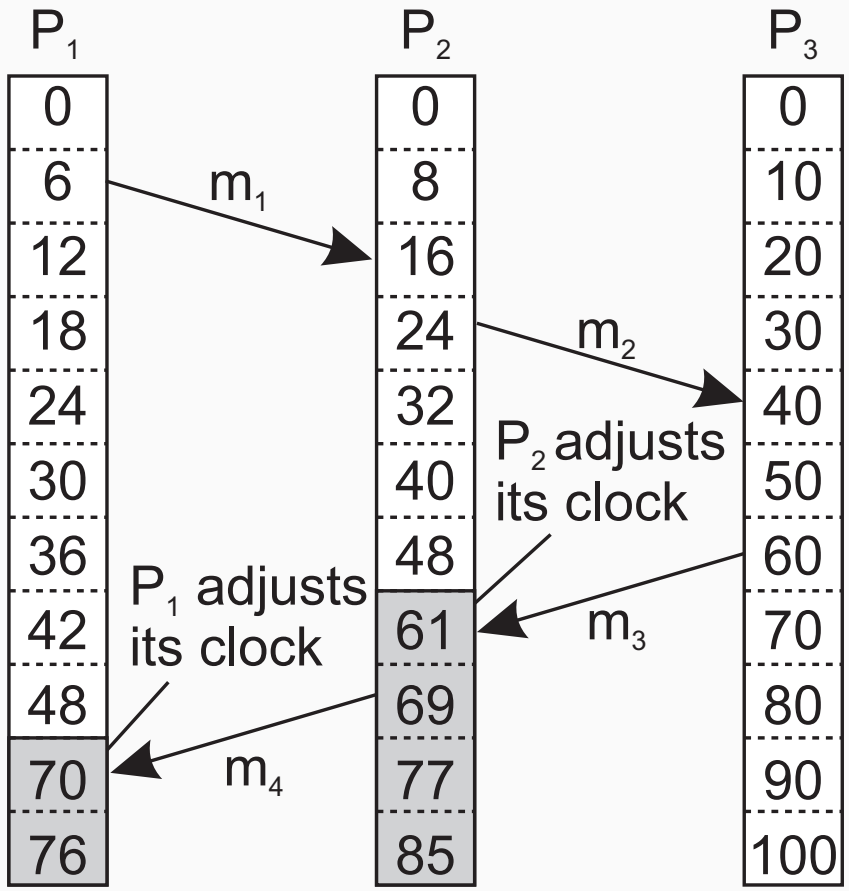
One possible relationship that preserves properties of happens-before.

If $a \rightarrow b$, then $C(a) < C(b)$.

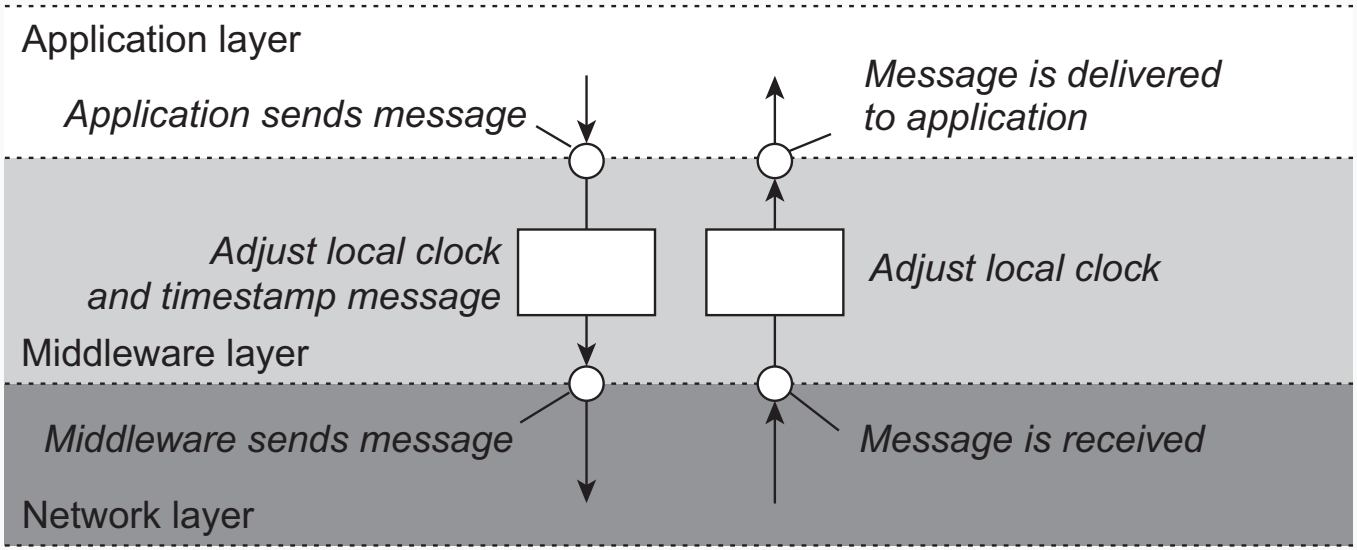
Don't we still need synchronization?



Lamport's logical clocks



Implementing Lamport's logical clocks



Implementing Lamport's logical clocks

- Each process P_i maintains local clock C_i
- Before executing an event, C_i is incremented by 1
- When sending a message M, it is timestamped with C_i
- When receiving a message M with timestamp T:
  - C_j for process P_j is adjusted to $max(C_j, T)$
  - C_j is incremented by 1
  - Message is delivered to application

Solving the Replica Problem: Total-ordered multicast

- Each client multicasts a message to all replicas
  - Each message is timestamped according to local logical clock
  - Assume no loss of messages
  - Assume reliable ordering
- Each replica places received messages in a queue
- Each replica processes messages in order of timestamps
  - Thus, ensuring total order
  - QED?

Total-ordered multicast

- Each client multicasts a message to all replicas
  - Each message is timestamped according to local logical clock
  - Assume no loss of messages
  - Assume reliable ordering
- Each replica places received messages in a queue
- Each replica acknowledges receipt of messages using a multicast
- Each replica processes messages in order of their timestamps
  - Only when it has received acknowledgement for that message from all other replicas

This protocol ensures all processes see the same queue.

Mutual Exclusion with Logical Clocks

- Total ordered multicast produces a total order among all messages
- Can be used to implement mutual exclusion
- Messages:
  - ENTER: process multicasts that it wants to enter a critical section
  - ALLOW: process unicasts permission to ENTERing process
  - RELEASE: process multicasts that it has left a critical section

Mutual Exclusion Condition

- On receiving
  - ENTER: placed into local queue sorted by timestamp, ALLOW sent to requesting process
  - ALLOW: placed into local queue sorted by timestamp
  - RELEASE: item at head of queue deleted
- A process enters the critical section if:
  - its ENTER is at head of queue *and*
  - there are messages (ENTER/ALLOW) from *all* other processes in queue
- When a process releases a critical section, it deletes all ALLOWs from its queue before multicasting RELEASE

Acknowledgements

Figures from van Steen and Tanenbaum, 3rd Edition.