CompSci 131

Parallel and Distributed Systems

Prof. A. Veidenbaum

Today's topics

- **Logical clocks**

- **Reading assignment:**
  - **Today: 6.2**
  - **Next time: 6.2, 6.3**

  - **Complete the assignment *before* next class**
    -

Last Lecture Covered

- Coordination
- Clocks
- Clock synchronization

Logical Clocks

- **Sometime don't need to know exact absolute time**
  - **Enough to know that A is *older* than B**
    - » **Will work for make!**
- **This can be done using *logical clocks***
  - **Seminal paper by L. Lamport from 1979**
- **Defines a "happens-before" relationship**
  - **k -> l means that k happened before l**
    - » **Meaning logical_clock(k) < logical_clock(l) (LC or simply C)**
  - ***all processes agree* that k happened before l**
- **In some cases, a->b cannot be observed**
  - **When?**

Definition of "Happens-Before"

- a and b are events in the same process
  - Know local time Ta, Tb
  - a->b if Ta < Tb

- a->b is also true if
  1. a is the event of sending a message by one process
  2. b is the event of receiving the message by another process

- Happens-before is transitive
  - a->b and b->c means a->c

- In all other cases events are ***concurrent***
  - i.e. -> does not hold between the two events

The Logical Clock

- **Logical clock, C(a), maintains the -> relationship**
  - **a->b implies C(a) < C(b)**

- **A process maintains its own (local) logical clock**
  - **Clock goes forward by ~~at least~~ 1 between *any* two events**
    - » **Receiving a message and forwarding to application**
    - » **Sending a message**

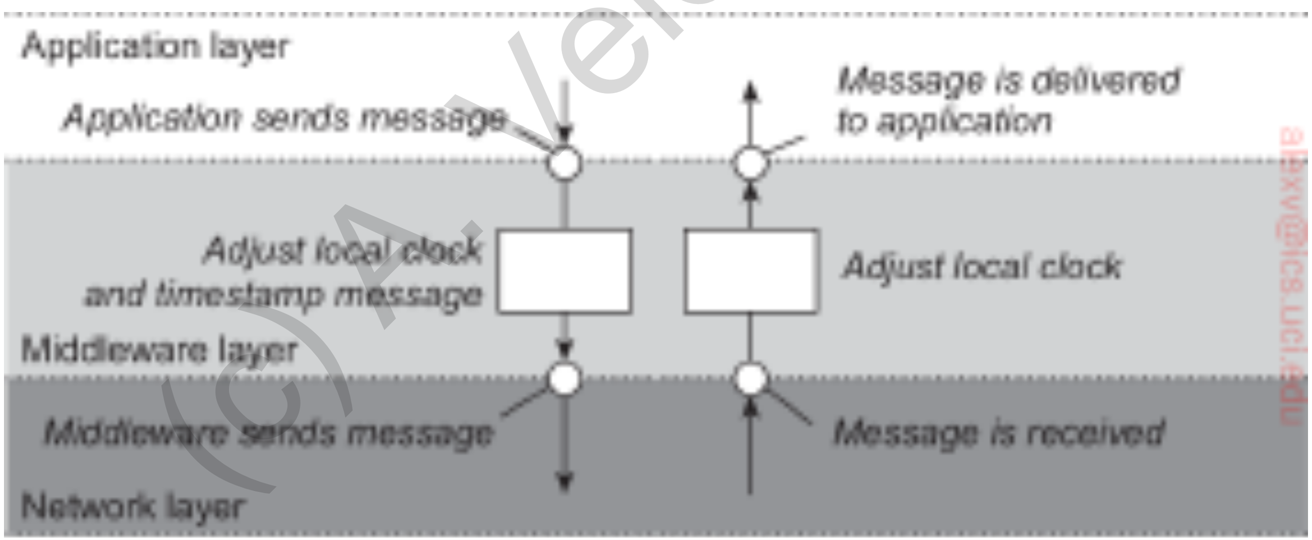
- **Lamport's logical clock algorithm**
  - **Each message carries senders C(a)**
  - **Arrival of a message *<u>may</u>* update a local (receiver) clock:**
    - » **If C(b) <= C(a) then C(b) = C(a) + 1**

- **The clock only goes *forward***

Implementation

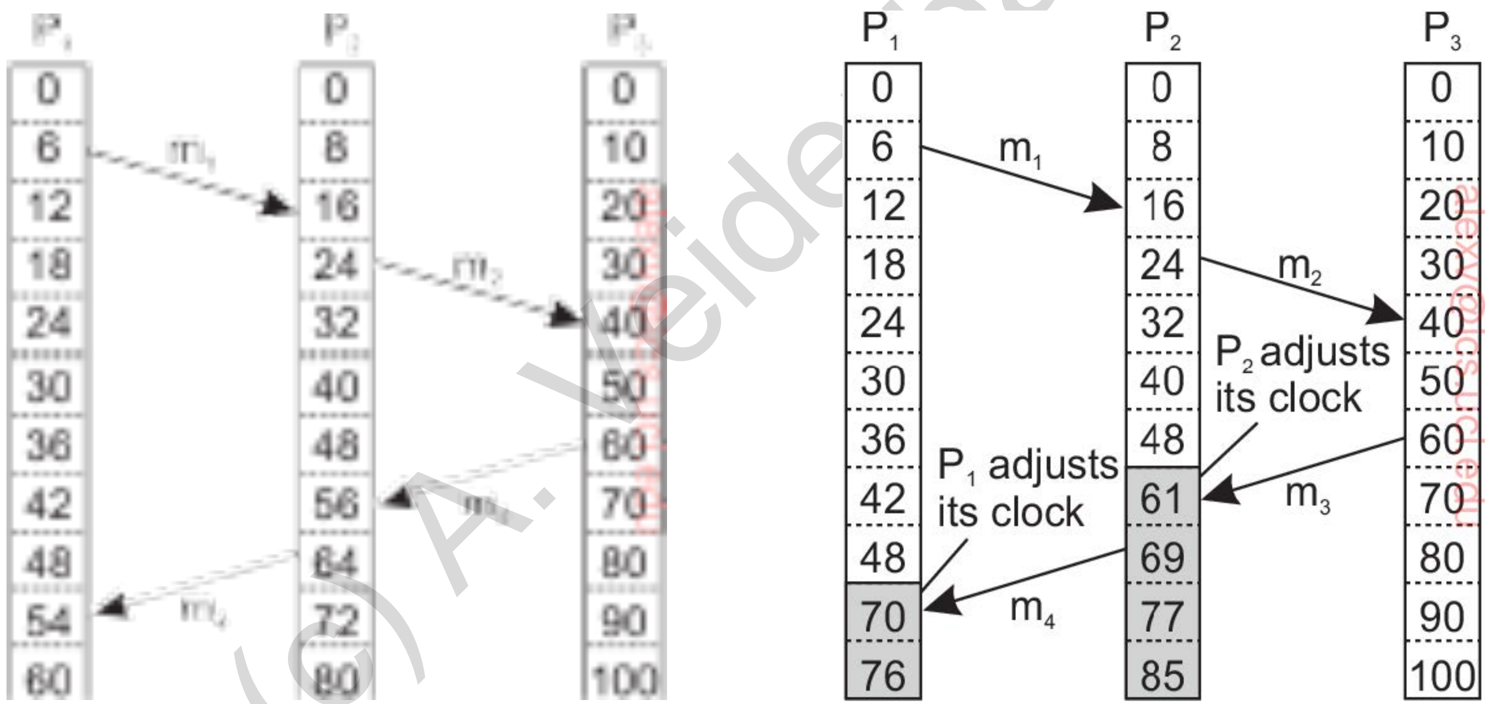
- **A process P_i maintains a local counter C_i**
  1. **Before any event P_i executes $C_i = C_i + 1$**
  2. **A message m to another process Pj carries $ts(m)=C_i$**
  3. **Arrival of message m at Pj causes $C_j=\max\{C_j, ts(m)\}$**
     - » ***Before anything else happens locally***

- **The middleware layer manages the logical clock**

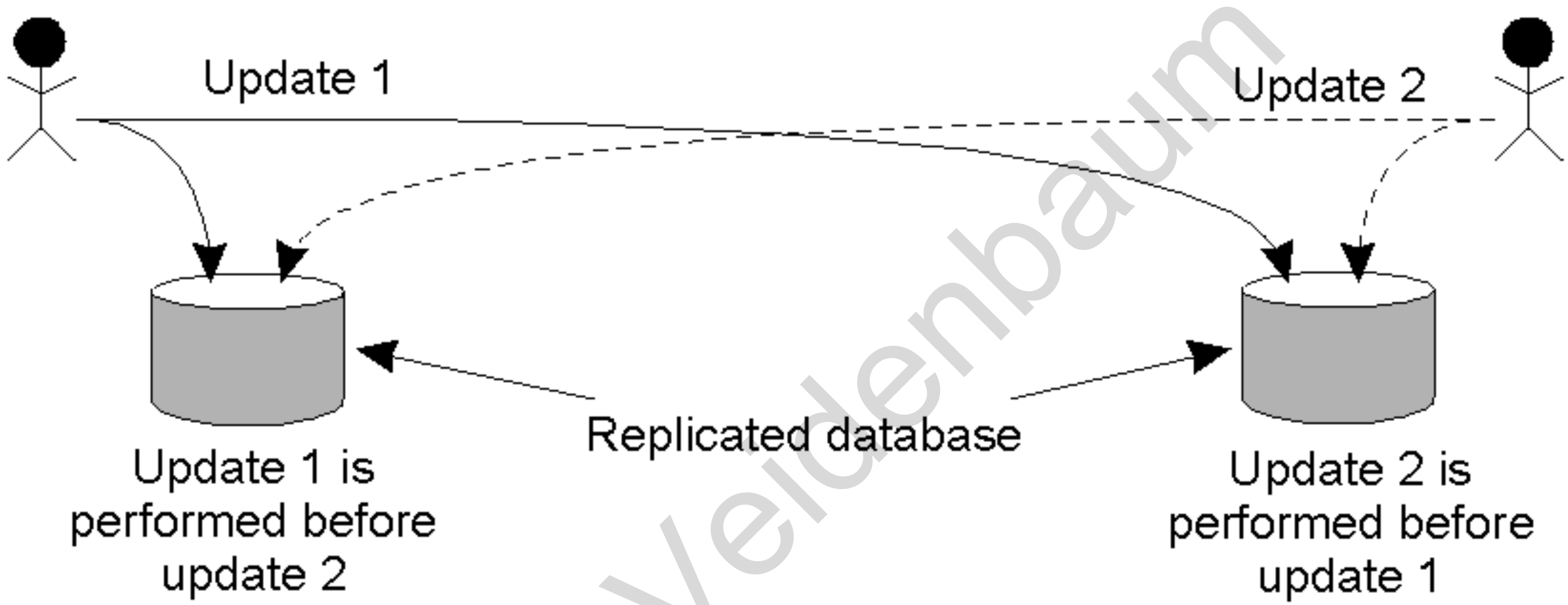


An example

- **3 processes**
- **Relative clock rates 6, 8, 10. No clock sync/LC**



Use in updating multiple replicas



- **Assume there are multiple DB copies for faster access**
  - ***Price to pay: a more complex update***
- **Problem: dependent updates to the same variable**
  - **e.g. deposit, interest calculation**
- ***Want* <u>*same*</u> order in *both places***
  - ***don't really care which order***

Solution

- Use a *totally ordered multicast*
  - *All* messages are delivered in *the same order* to *all* processes
    - » Assumes that no message is lost, in-order from same sender
    - » Each message has a time stamp

- Multicast an update to all processes in the system
  - Including *sending to the sender*
  - Queue up all updates at each node
    - » Messages are put in the queue in TS order

- A receiver multicasts an ACK
  - The ACK has a higher TS than the original message

- **A process delivers a message to an app when**
  1. **The message is at the top of the queue (oldest)**
  2. **All processes have ACKed this message**
- **The message is removed from the queue**
  - » **All its acks as well**
- **Eventually, all queues have the same state**

Vector clocks

- **Problem with logical clocks:**
  1. If C(a) < C(b) what is the relationship between a and b?
     - » C(a) < C(b) ***does not always mean*** a -> b

- **Example**

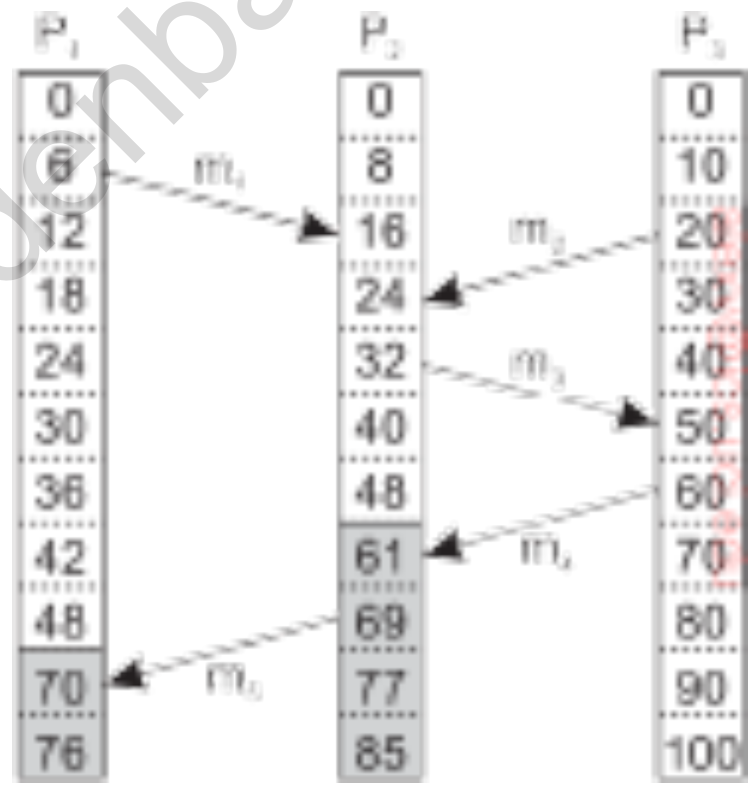
  Consider m1 and m3

  M3 sent after receipt of m1

  Now consider m1 and m2

  $C_{rec}(m1) < C_{snd}(m2)$

  But no relationship!



Vector clocks

- **Also, logical clocks do not capture *causality***
  - *One event (potentially) causing another*
    - » *LC equals event count at process*
- **Solution: vector clocks VC**
- **For a system w/ N processes**
  - VC_i is an integer array of N elements at process i
  - $VC_i[i]$ is the ***local*** logical clock
    - » The process updates it as a Lamport clock
    - » $VC_i[i]$ is a count of local events (snd/rcv's)
- **A process sends V_i with every message**

Vector clock update

- Initially, $VC_i[j] = 0$ for a i, j in [1..N]
- $VC_i[i] = VC_i[i] + 1$ before any new event at i
- If P_j receives a message with vector timestamp ts
  - $VC_j[i] = \max(VC_j[i], ts[i])$ for i in [1..N]
- In other words, *a local update of all clocks* happens on a receipt of a message