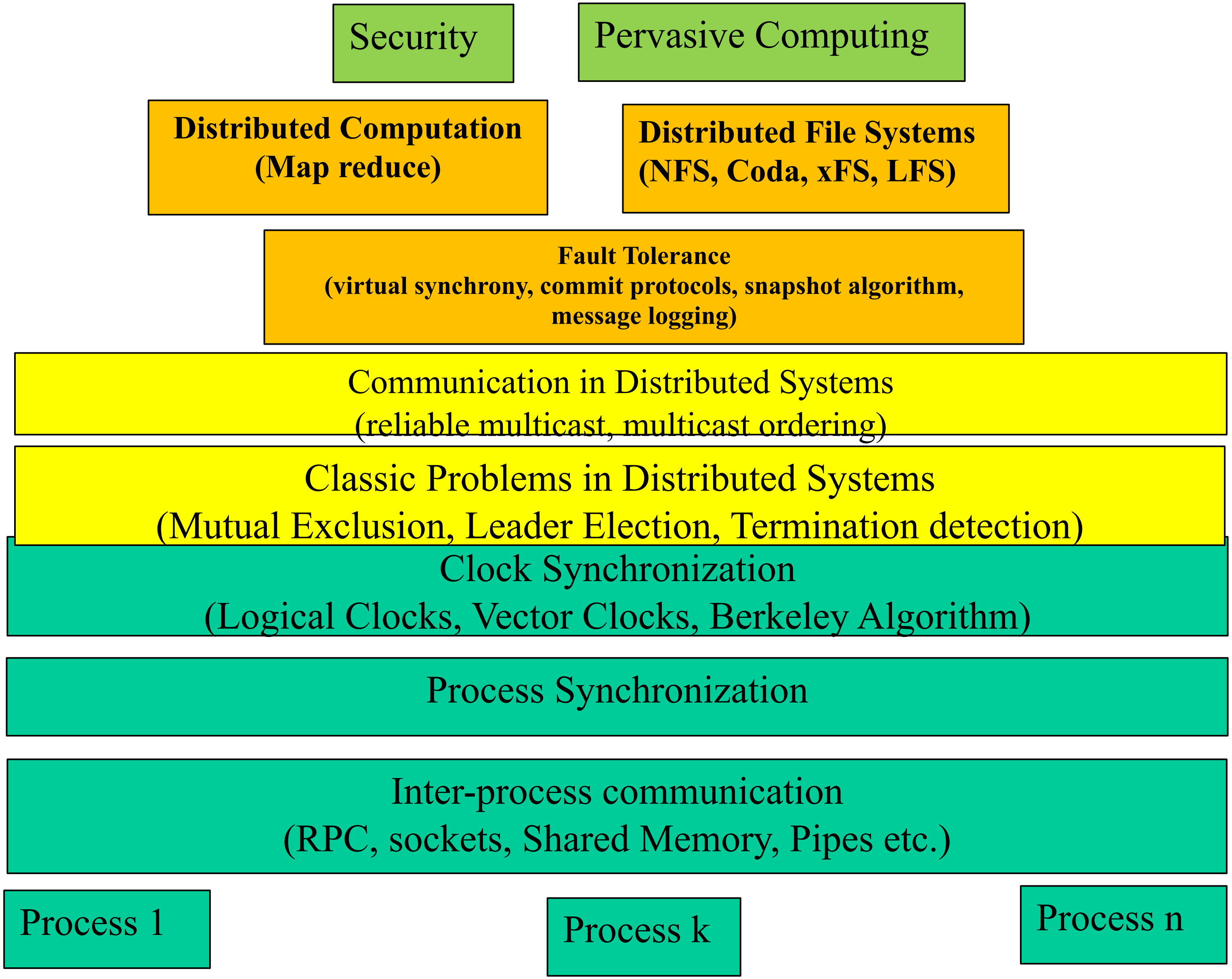
CMSC621: Advanced Operating Systems

Nilanjan Banerjee

Associate Professor, University of Maryland
Baltimore County
nilanb@umbc.edu
http://www.csee.umbc.edu/~nilanb/

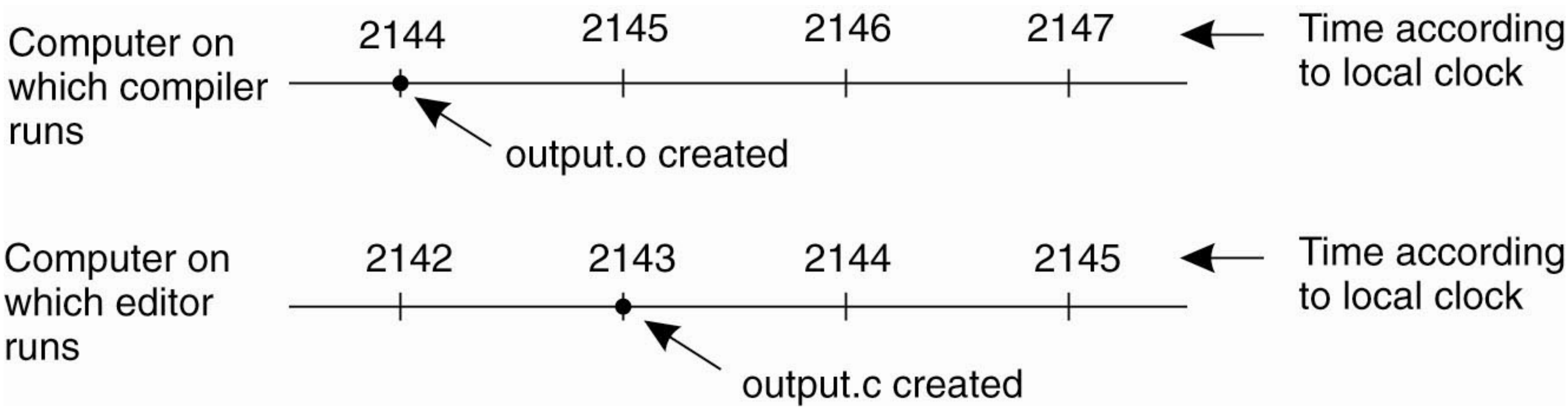
Overview of the course



Core Synchronization problems in Distributed Systems.

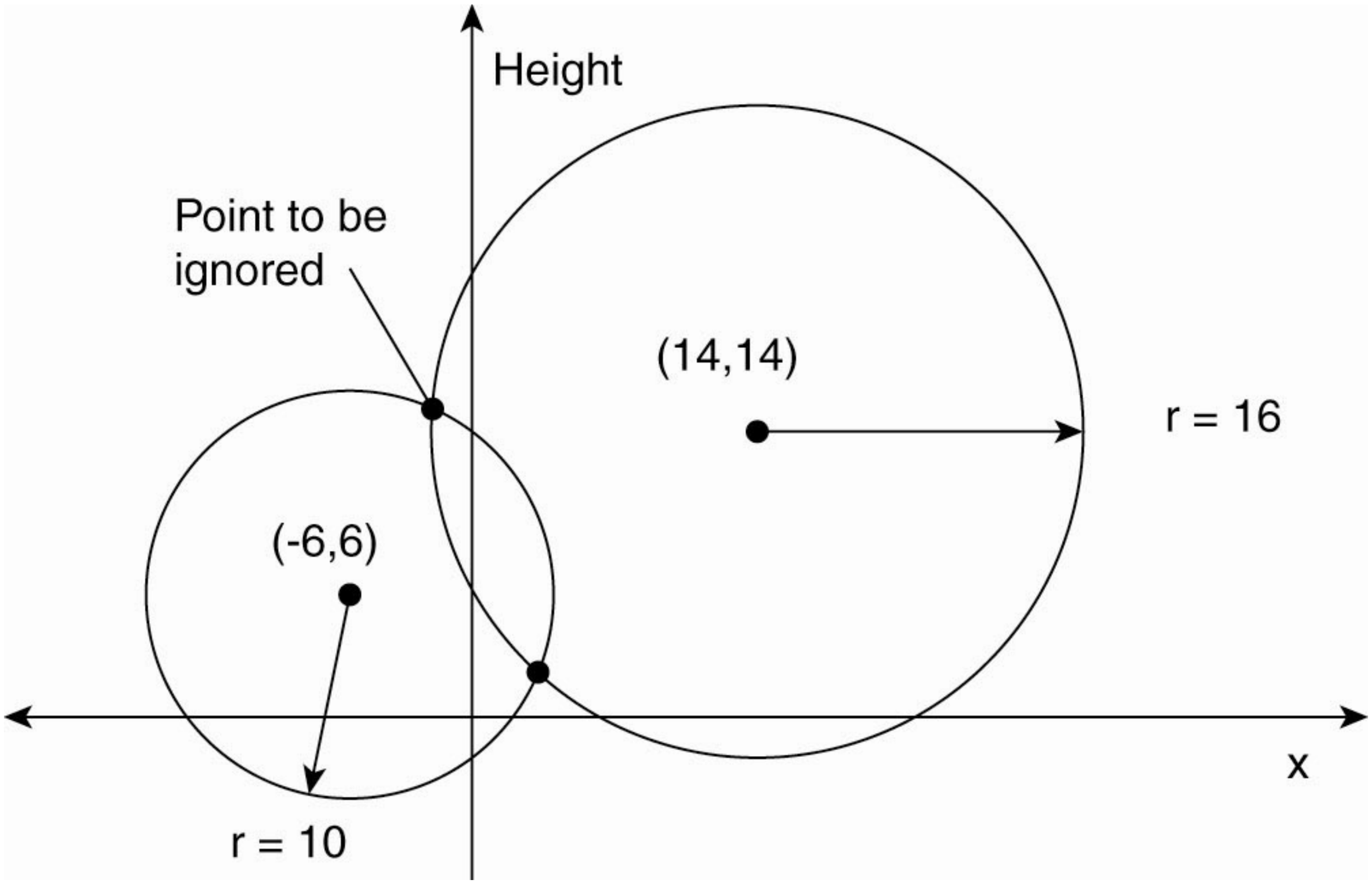
- Distributed Synchronization
  - Clocks, snapshots, leader election, distributed transactions

Why is clock synchronization important?



When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time.

Another example: Global Positioning System



Computing a position in a two-dimensional space.

Global Positioning System

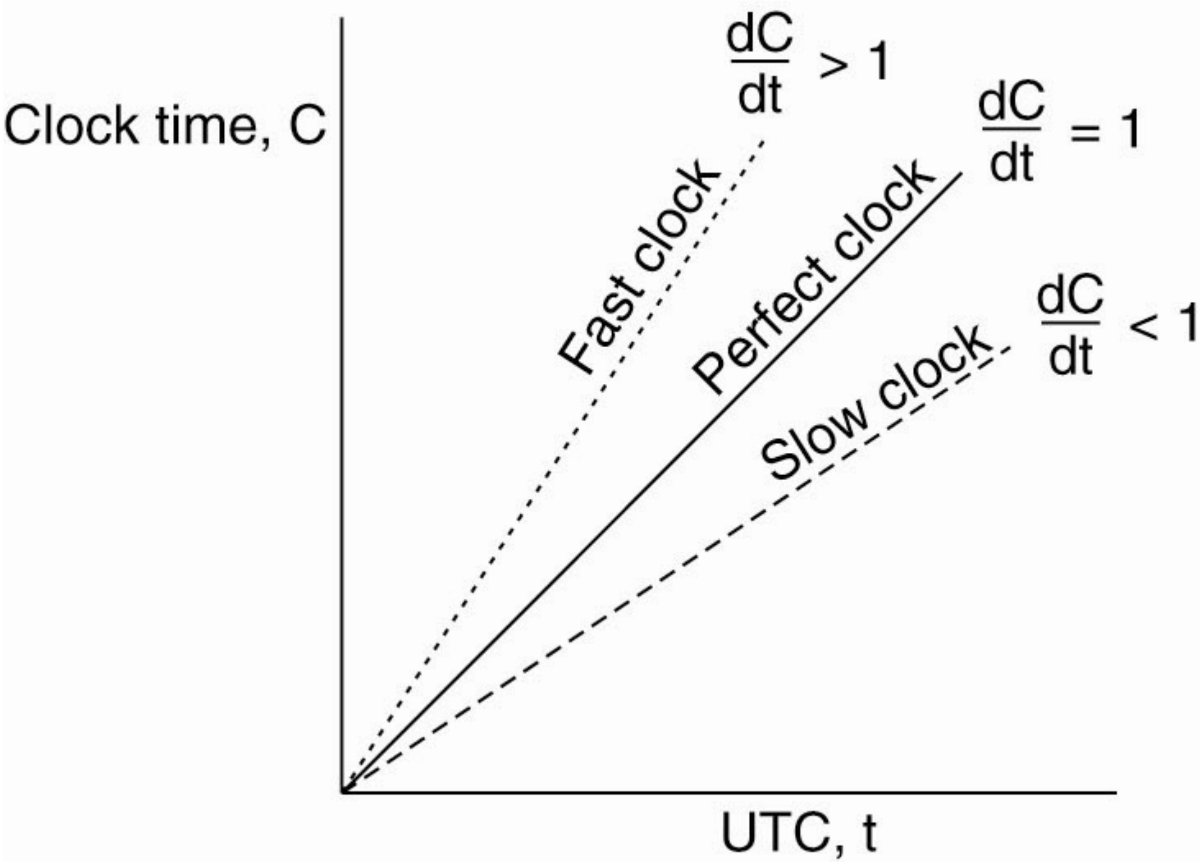
Real world facts that complicate GPS

1. It takes a while before data on a satellite's position reaches the receiver.
2. The receiver's clock is generally not in synch with that of a satellite.

What is the math behind calculating the GPS location?

Clock Synchronization Algorithms

The relation between clock time and UTC when clocks tick at different rates.



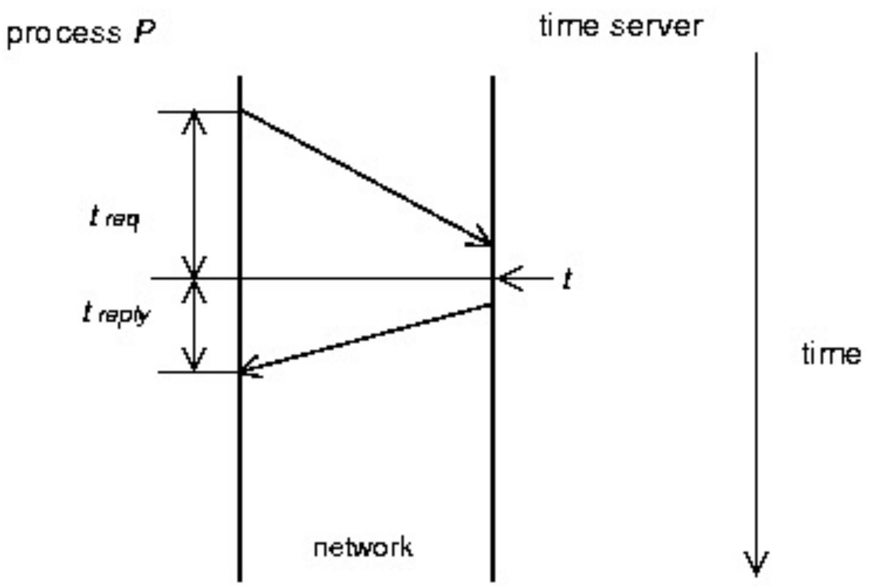
Rate of Synchronization

If a manufacturer assures that the maximum clock drift is p. To assure that the relative offset of the clock is *a*, what the rate of synchronization or maximum interval of synchronization?

Network Time Protocol

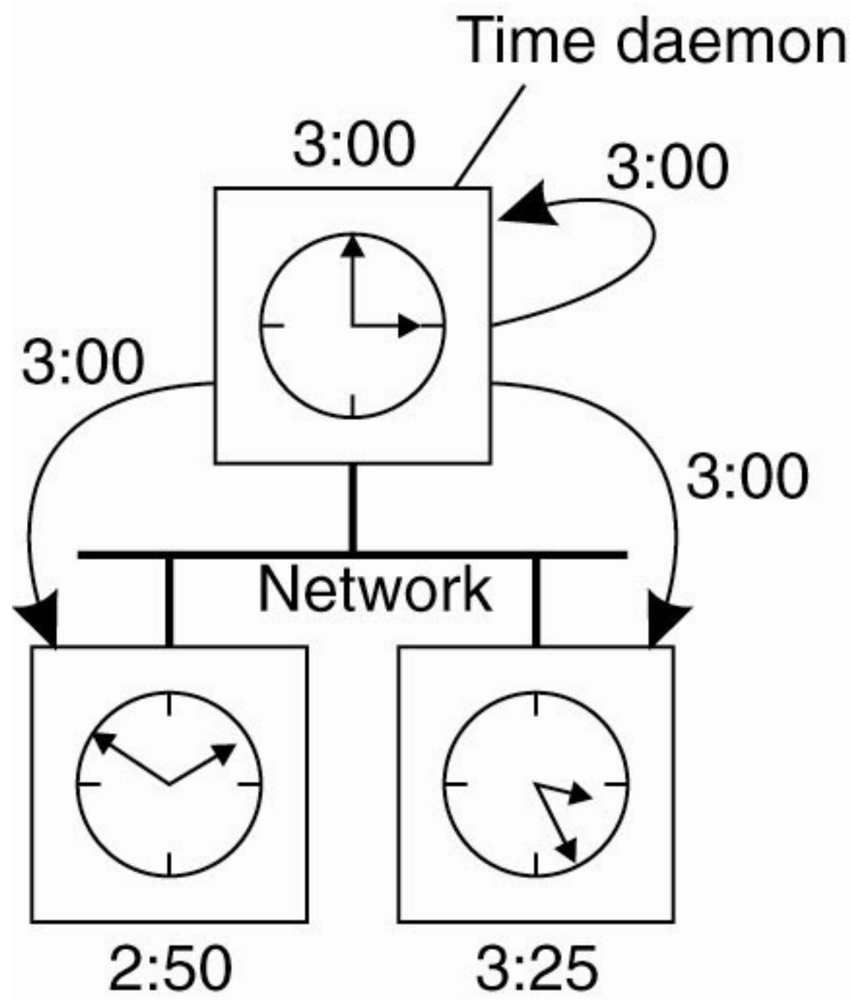
Getting the current time from a time server **(Cristhian's algorithm)**

- Synchronize machines to a *time server* with a UTC receiver
- Machine P requests time from server every $\delta/2\rho$ seconds
  - Receives time t from server, P sets clock to $t+t_{reply}$ where t_{reply} is the time to send reply to P
  - Use $(t_{req}+t_{reply})/2$ as an estimate of t_{reply}
  - Improve accuracy by making a series of measurements



The Berkeley Algorithm (1)

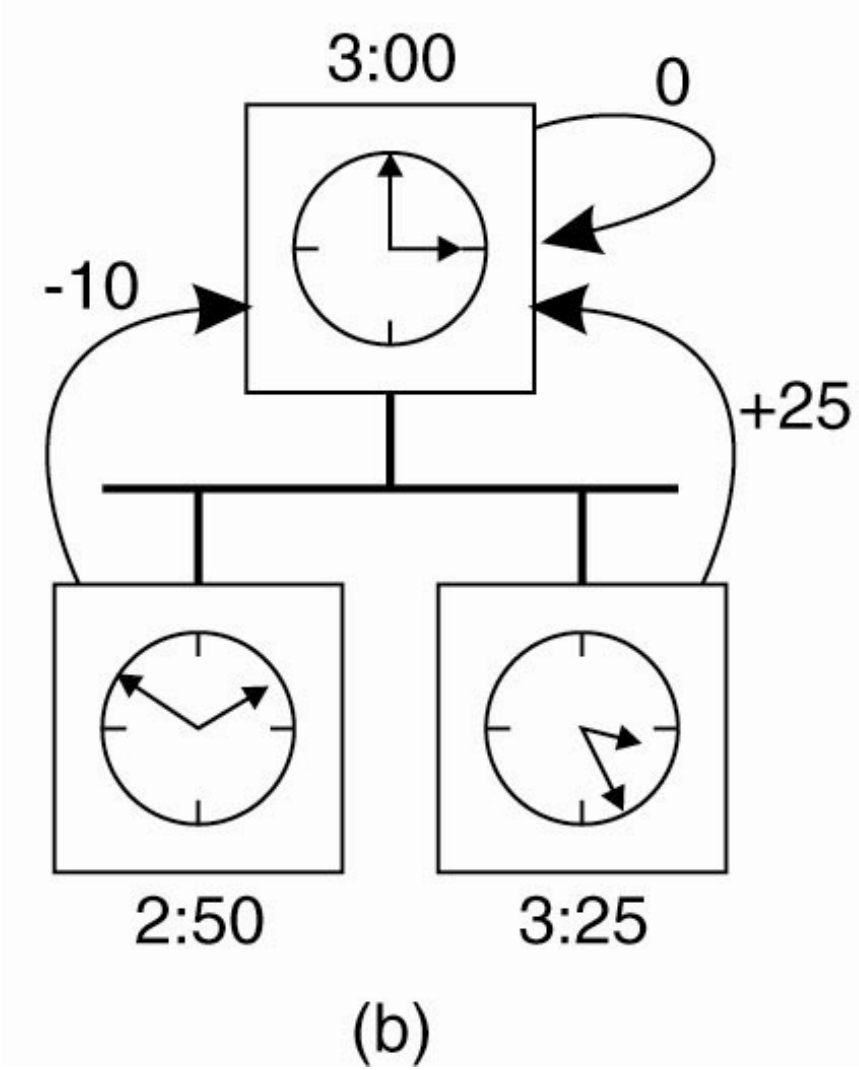
The time daemon asks all the other machines for their clock values.



(a)

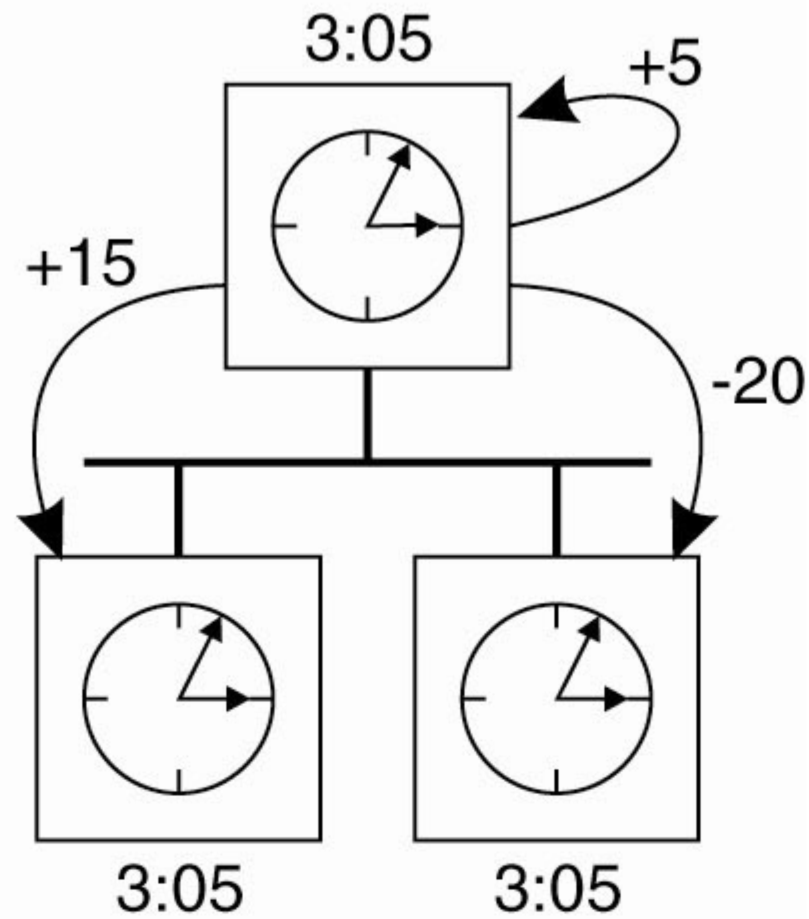
The Berkeley Algorithm (2)

The machines answer.



The Berkeley Algorithm (3)

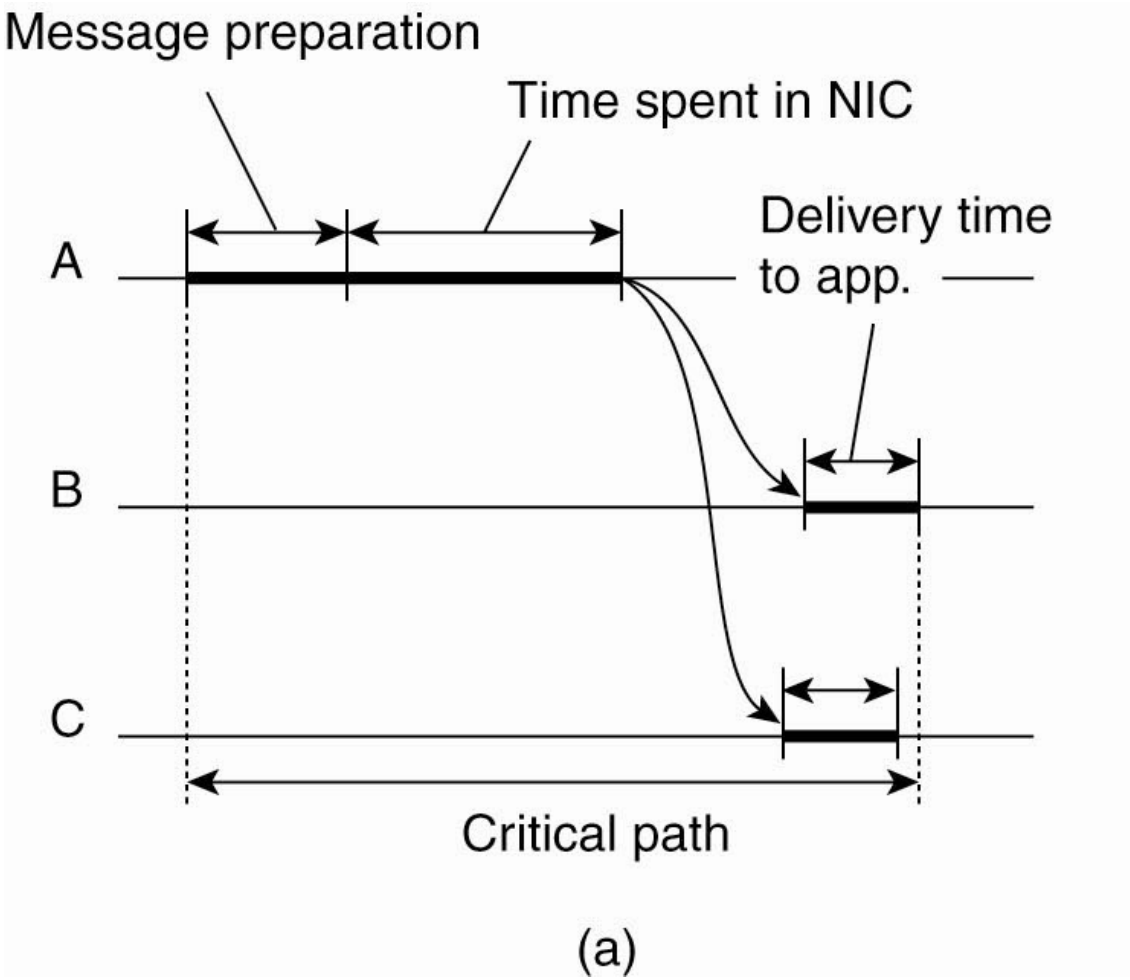
The time daemon tells everyone how to adjust their clock.



(c)

Clock Synchronization in Wireless Networks

The usual critical path in determining network delays.



Logical Clocks

- For many problems, internal consistency of clocks is important
  - Absolute time is less important
  - Use *logical* clocks
- Key idea:
  - Clock synchronization need not be absolute
  - If two machines do not interact, no need to synchronize them
  - More importantly, processes need to agree on the ***order*** in which events occur rather than the *time* at which they occurred

Event Ordering

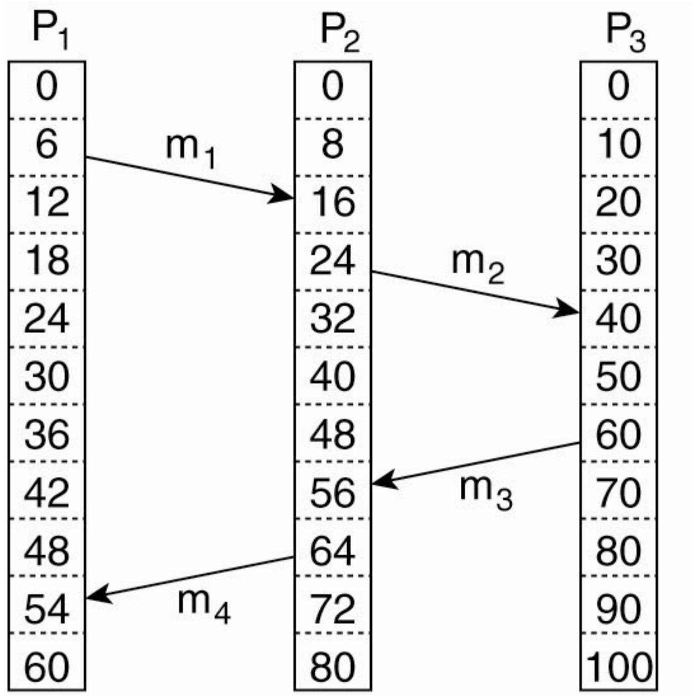
- *Problem:* define a total ordering of all events that occur in a system
- Events in a single processor machine are totally ordered
- In a distributed system:
  - No global clock, local clocks may be unsynchronized
  - Can not order events on different machines using local times
- Key idea [Lamport]
  - Processes exchange messages
  - Message must be sent before received
  - Send/receive used to order events (and synchronize clocks)

Lamport's Logical Clocks (1)

- The "happens-before" relation $\rightarrow$ can be observed directly in two situations:
- If *a* and *b* are events in the same process, and *a* occurs before *b*, then $a \rightarrow b$ is true.
- If a is the event of a message being sent by one process, and *b* is the event of the message being received by another process, then $a \rightarrow b$
- What happens for processes that do not exchange messages?

Lamport's Logical Clocks (2)

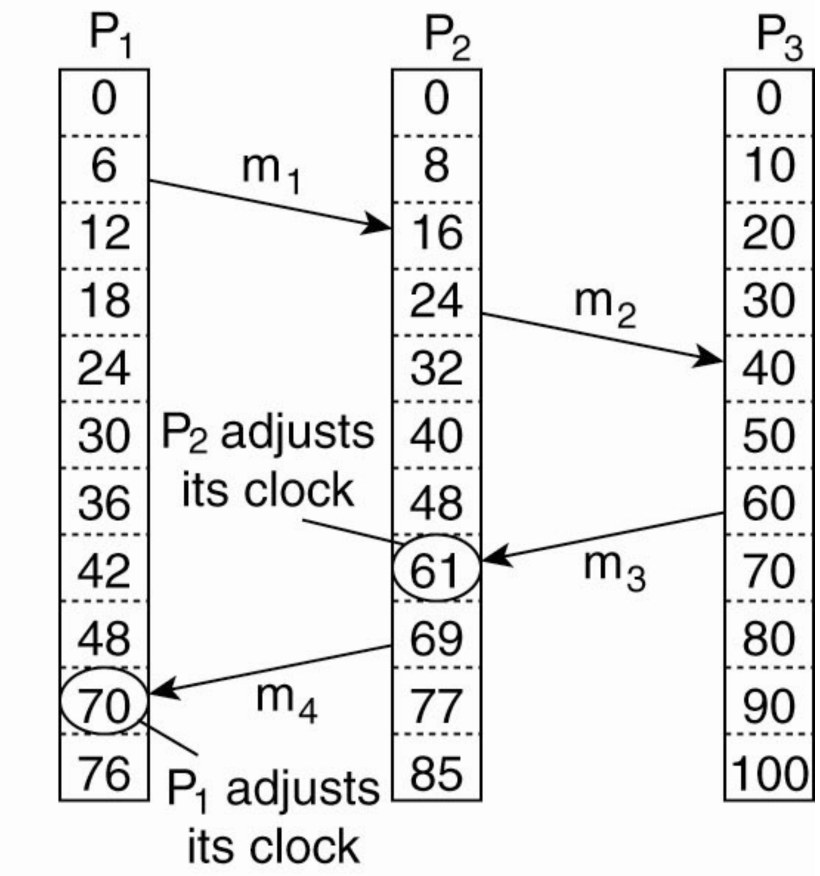
- (a) Three processes, each with its own clock. The clocks run at different rates.



(a)

Where is the happens-before relationship violated?

Lamport's Logical Clocks

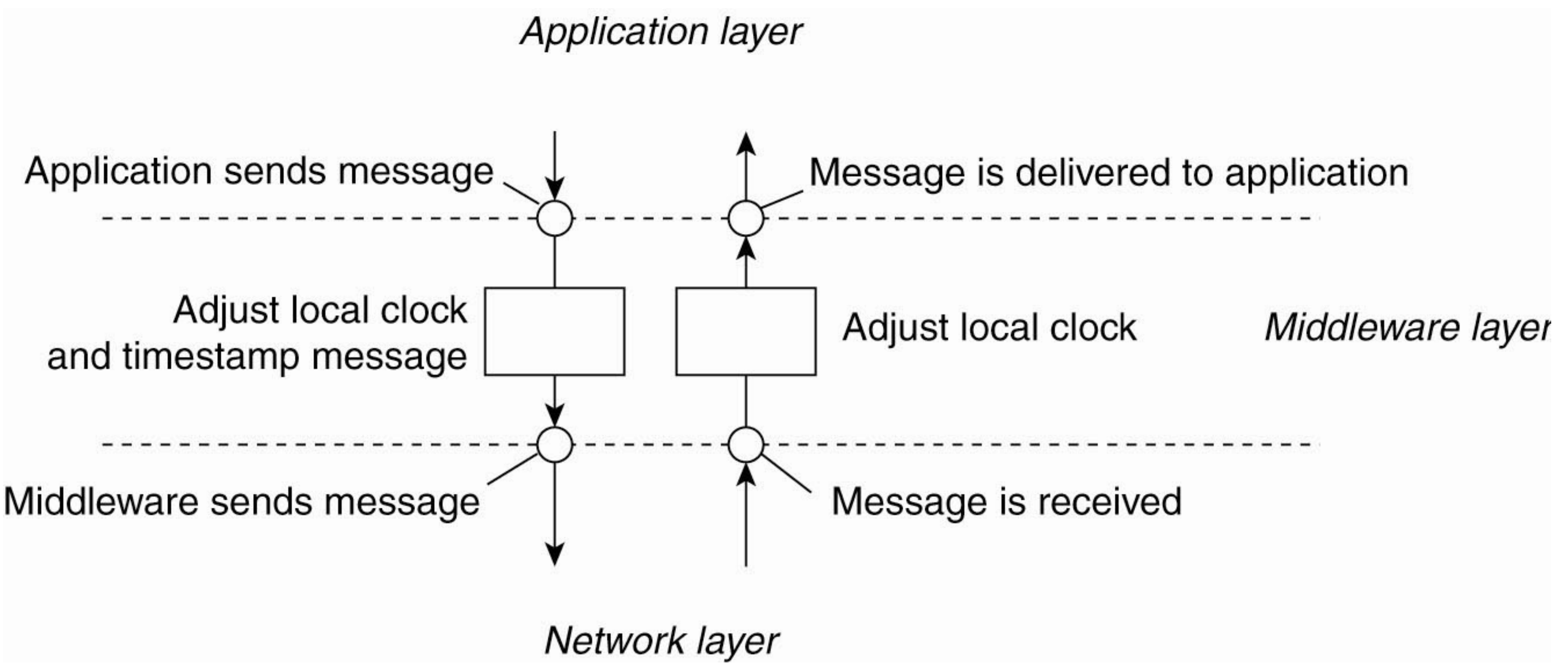


(b)

Lamport's algorithm corrects the clocks.

Lamport's Logical Clocks

The positioning of Lamport's logical clocks in distributed systems.



Lamport's Logical Clocks

Updating counter C_i for process P_i

1. Before executing an event P_i executes $C_i \leftarrow C_i + 1$.
2. When process P_i sends a message m to P_j, it sets *m*'s timestamp *ts (m)* equal to C_i after having executed the previous step.
3. Upon the receipt of a message *m*, process P_j adjusts its own local counter as $C_j \leftarrow \max\{C_j, ts(m)\} + 1$, after which it then executes the first step and delivers the message to the application.

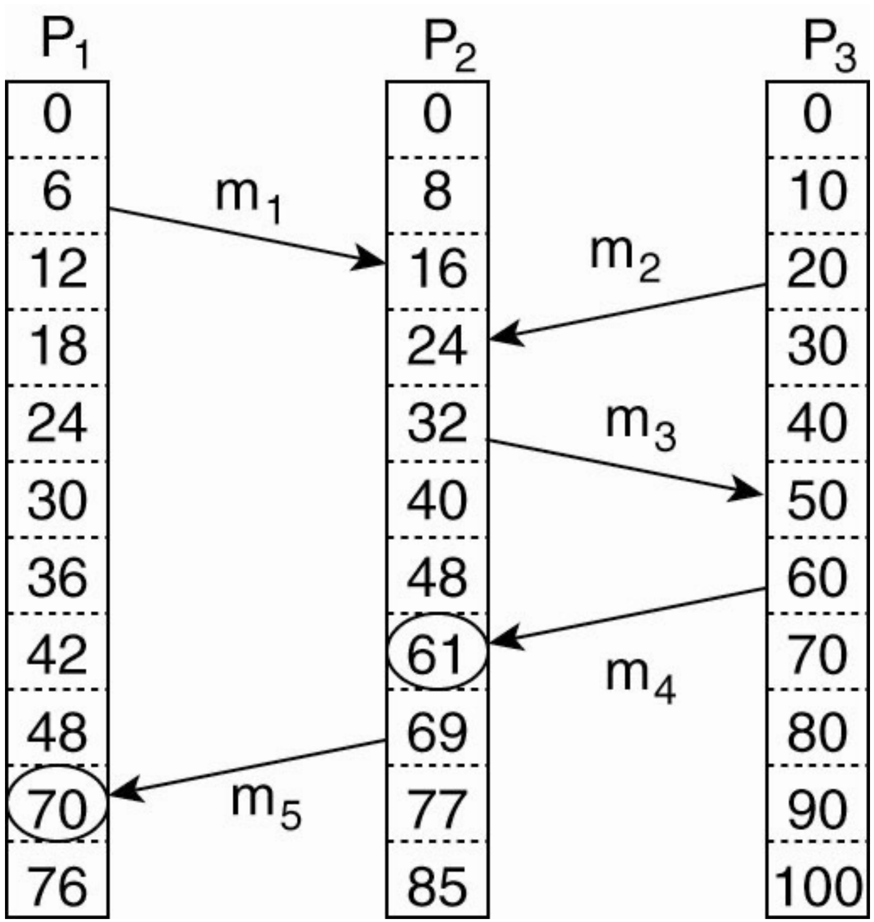
Causality

- Lamport's logical clocks
  - If *A* -> *B* then *C(A)* < *C(B)*: **Proof?**
  - Is the reverse true?
    - Nothing can be said about events by comparing time-stamps!
    - If *C(A)* < *C(B)*, then what can you say?
- Need to maintain *causality*
  - If a -> b then a is causally 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 (1)

Concurrent message transmission using logical clocks.



Vector Clocks (2)

- Vector clocks are constructed by letting each process P_i maintain a vector VC_i with the following two properties:

  1. VC_i [i] is the number of events that have occurred so far at P_i. In other words, VC_i [i] is the local logical clock at process P_i .

  2. If VC_i [j] = k then P_i knows that k events have occurred at P_j. It is thus P_i's knowledge of the local time at P_j .

Vector Clocks (3)

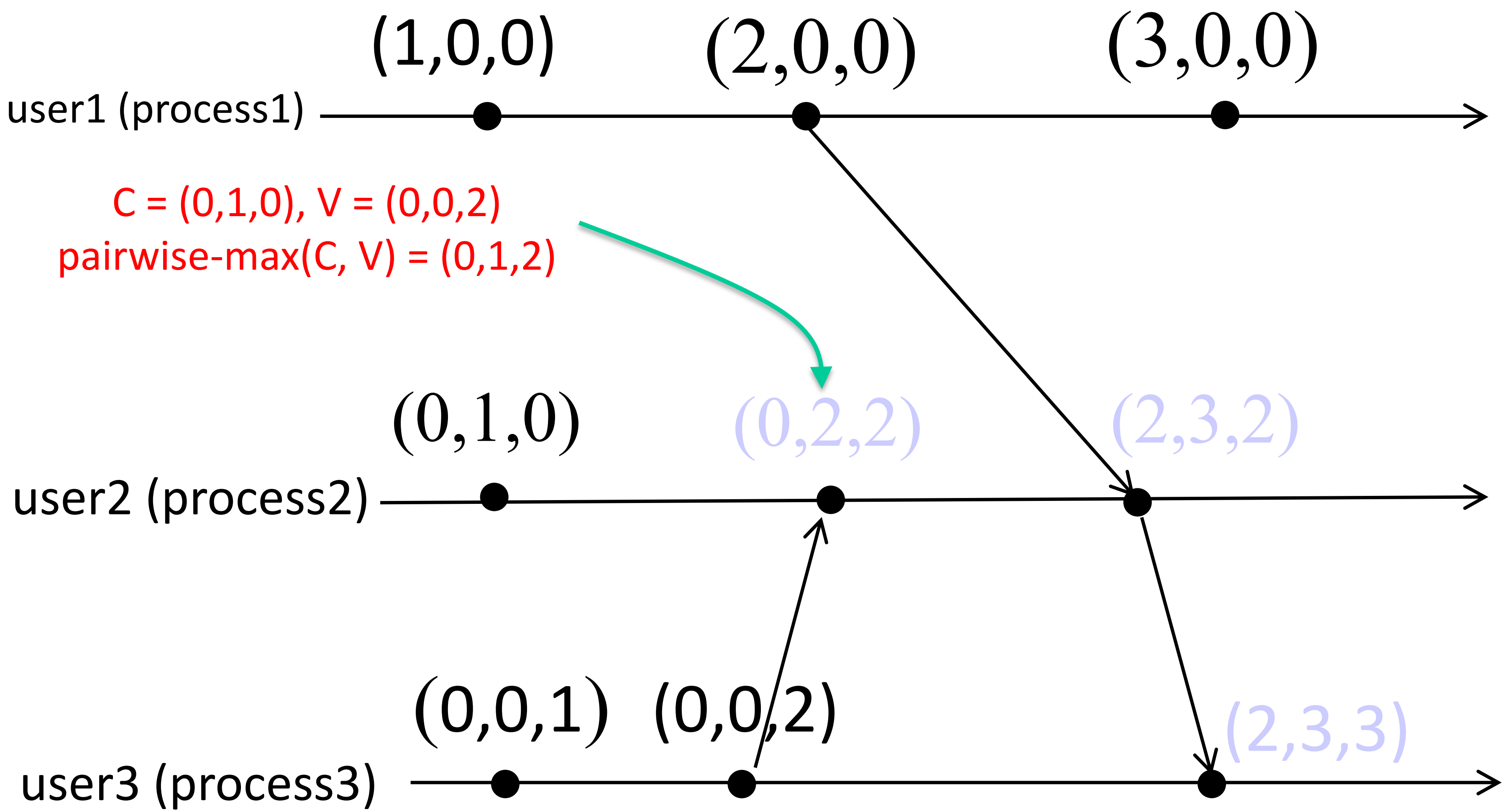
- Steps carried out to accomplish property 2 of previous slide:
  1. Before executing an event P_i executes <u>**$VC_i[i] \leftarrow VC_i[i] + 1$.**</u>
  2. When process P_i sends a message m to P_j, it sets *m*'s (vector) timestamp *ts (m)* equal to VC_i after having executed the previous step.
  3. Upon the receipt of a message m, process P_j adjusts its own vector by setting <u>**$VC_j[k] \leftarrow \max\{VC_j[k], ts(m)[k]\}$**</u> for each *k*, after which it executes the first step and delivers the message to the application.

Software "Clock" 2: Vector Clocks

- Logical clock:
  - Event s happens before event t $\Rightarrow$ the logical clock value of s is smaller than the logical clock value of t.
- Vector clock:
  - Event s happens before event t $\Leftrightarrow$ the vector clock value of s is "smaller" than the vector clock value of t.
- Each event has a vector of *n* integers as its vector clock value
  - v1 = v2 if all n fields same
  - v1 $\leq$ v2 if every field in v1 is less than or equal to the corresponding field in v2
  - v1 < v2 if v1 $\leq$ v2 and v1 $\neq$ v2

Relation "<" here is not a total order

Vector Clock Protocol



Vector Clocks

Lets do an in-class exercise

Vector Clock Properties

- Event s happens before t $\Rightarrow$ vector clock value of s < vector clock value of t: There must be a chain from s to t
- Event s happens before t $\Leftarrow$ vector clock value of s < vector clock value of t
  - If s and t on same process, done
  - If s is on p and t is on q, let Vs be s's vector clock and Vt be t's
  - Vs < Vt $\Rightarrow$ Vs[p] < Vt[p] $\Rightarrow$ Must be a sequence of message from p to q after s and before t

Correctness of Vector Timestamps

Theorem: Vector timestamps implement vector clocks.

Proof: First, show $a \rightarrow b$ implies

$V(a) < V(b)$.

<u>*Case 1:*</u> *a* and *b* both occur at p_i, *a* first. Since V_i increases at each step,

$V(a) < V(b)$.

Correctness of Vector Timestamps

<u>*Case 2:*</u> *a* occurs at p_i and causes *m* to be sent, while *b* occurs at p_j and includes the receipt of *m*.

- During *b*, p_j updates its vector timestamp in such a way that $V(a) \leq V(b)$.
- p_i 's estimate of number of steps taken by p_j is never an over-estimate. Since *m* is not received before it is sent, p_i 's estimate of the number of steps taken by p_j when *a* occurs is less than the number of steps taken by p_j when *b* occurs. So $V(a)[j] < V(b)[j]$.
- Thus $V(a) < V(b)$.

Correctness of Vector Timestamps

Case 3: There exists c such that $a \rightarrow c$ and $c \rightarrow b$.

By induction (from Cases 1 and 2) and transitivity of <, $V(a) < V(b)$.

Next show $V(a) < V(b)$ implies $a \rightarrow b$.

Equivalent to showing $!(a \rightarrow b)$ implies $!(V(a) < V(b))$

Correctness of Vector Timestamps

- Suppose *a* occurs at p_i, *b* occurs at p_j, and *a* does not happen before *b*.
- Let $V(a)[i] = k$.
- Since *a* does not happen before *b*, there is no chain of messages from p_i to p_j originating at p_i 's *k*-th step or later and ending at p_j before *b*.
- Thus $V(b)[i] < k$.
- Thus $!(V(a) < V(b))$.

Overview of the course