Distributed Systems
ECEG-6504

Synchronization

Surafel Lemma Abebe (Ph. D.)

Topics

- Clock synchronization
- Logical clocks
- Mutual exclusion
- Elections

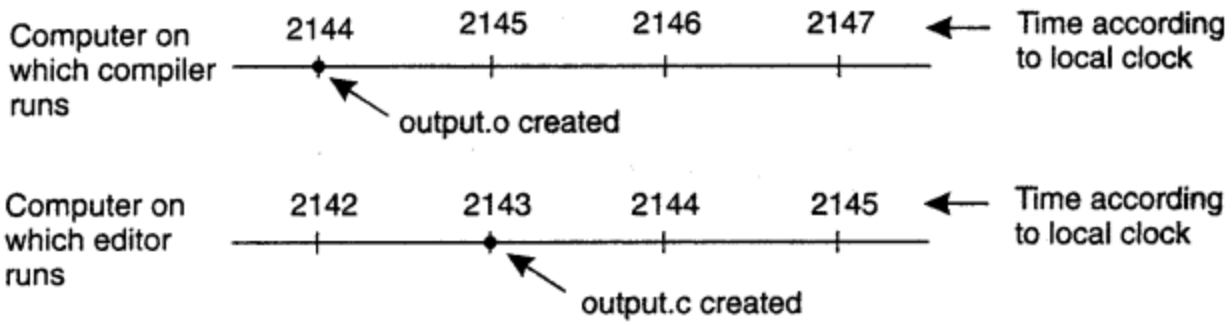
Introduction

- Why is time important in DS?
  1. To know at what time of day a particular event occurred at a particular computer
     - Example
       - For auditing eCommerce transactions
  2. Several algorithms depend upon clock synchronization to address problems in distribution
     - Example
       - Maintaining consistency of distributed data
       - Checking the authenticity of a request sent to a server
       - Eliminating the processing of duplicate updates

- Reasons for problems related to time
  - Measuring time is difficult due to multiple frames of reference
  - Inability to timestamp events at different nodes sufficiently accurately to know the order in which any pair of events occurred
    - No absolute global time

Clock synchronization

- Problem
  - In DS achieving agreement on time is not trivial
    - Example: *make* program in UNIX



Clock synchronization…

- Physical clocks
  - Computer timer is a precisely machined quartz crystal
  - When kept under tension, quartz crystals oscillate at a well-defined frequency
  - Time in computer
    - With each crystal, there are two registers: a counter and a holding register
      - Each oscillation of the crystal decrements the counter by one
      - When the counter gets to zero, an interrupt is generated and the counter is reloaded from the holding register

      ⇒ Used to program a timer to generate an interrupt 60 times a second or any desired frequency

        » The interrupt is called one clock tick
    - At every clock tick, the interrupt service procedure adds one to the time stored in memory
      - Time is stored as the number of ticks after some known starting date and time
      - CMOS RAM

Clock synchronization…

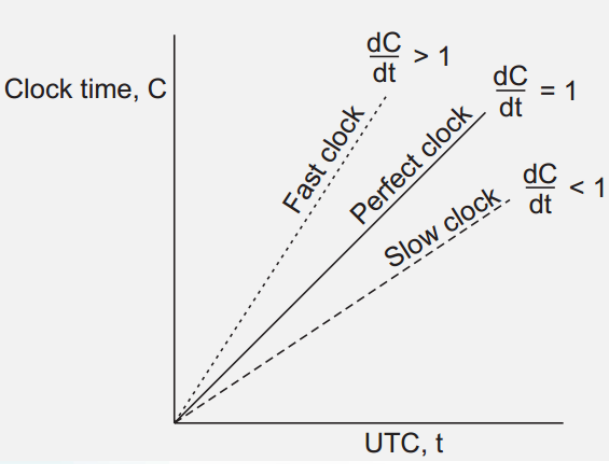
- Physical clocks…
  - UTC (Universal Coordinated Time)
    - Is the basis for all modern civil timekeeping (starting 1948)
    - Based on the number of transitions of the cesium 133 atom
    - To keep in phase with the sun, a leap second is introduced when necessary
      - Total number of leap seconds introduced into UTC so far is about 30
    - Bureau International de l'Heure (BIR)
      - Announces a leap second
      - Computes real time as the average of some 50 cesium-clocks around the world
    - UTC is broadcasted through shortwave radio and satellite
      - Satellites can give an accuracy of about ±0.5 ms

Clock synchronization…

- Physical clocks…
  - Clock skew
    - Is the difference in time values between different clocks
    - Could cause failure of programs
      - Programs expect the time associated with different entities (e.g., file) be correct and independent of the machine on which the time was generated
    - Caused by crystals running in a slightly different rate, clock drift
  - Clock drift
    - The phenomenon of clocks ticking at different rates that widens the gap in perceived time

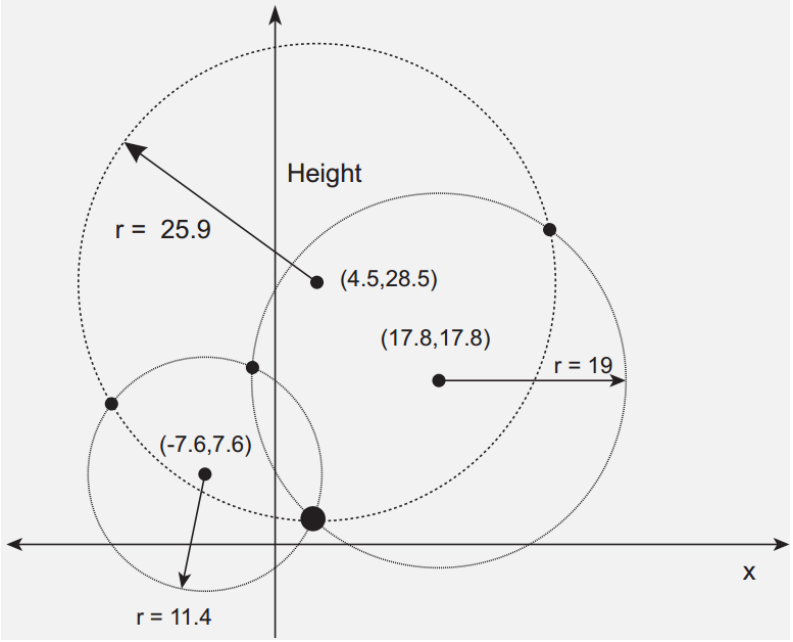
In practice: $1-x < dC/dt \leq 1+x$, x is maximum drift rate

Goal: Never let two clocks in any system differ by more than α time units, then synchronize at least every $\alpha/2x$ seconds



Clock synchronization…

- Global positioning system
  - Could be used to determine one's geographical position on Earth and time



Clock synchronization…

- Global positioning system…
  - Current location
    - Assumptions
      - The clocks in the satellites are accurate and synchronized
    - Facts
      - It takes a while before data on a satellite's position reaches the receiver
      - The receiver's clock is generally not in synch with that of a satellite

Clock synchronization…

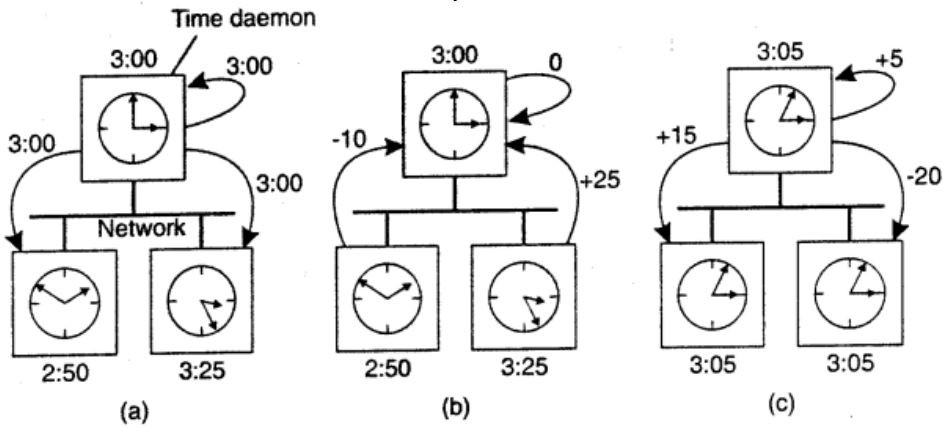
- Global positioning system…
  - Current location
    - Δ_r: unknown deviation of the receiver's clock
    - x_r, y_r, z_r: unknown coordinates of the receiver
    - T_i : timestamp on a message from a satellite I
    - $\Delta_i=(T_{now}-T_i)+\Delta_r$: measured delay of a message sent by satellite I
    - Measured distance to satellite I = c *Δ_i (c is speed of light, $3*10^8$m/s)
    - Real distance is

      $d_i=C\,\Delta_i - C\,\Delta_r = \sqrt{(x_i-x_r)^2+(y_i-y_r)^2+(z_i-z_r)^2}$

  - If we have four satellites, we get four equations with four unknowns allowing us to solve the coordinates for the receiver and Δ_r

Clock synchronization…

- Berkeley algorithm
  - Algorithm for internal synchronization
  - Master: a coordinator computer is chosen
    1. Periodically polls the slave computers whose clocks are to be synchronized (a)
    2. The slaves send back their clock values to it (b)
    3. Computes an average time
    4. Tell the machines (c) to
       - Advance their clock to the new time, or
       - Slow down their clock until some specified reduction has been achieved

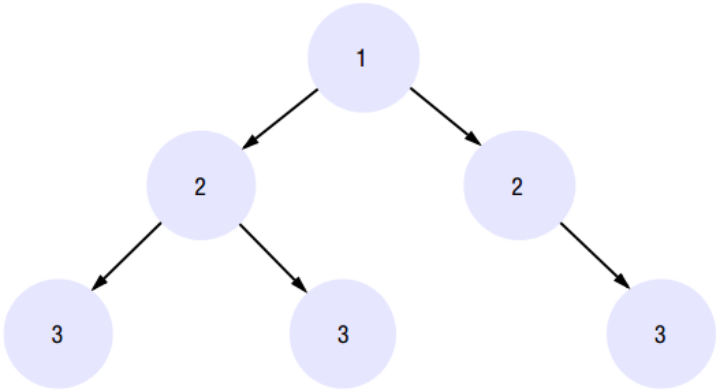


Clock synchronization…

- Network protocol time (NTP)
  - Is an architecture for a time service and a protocol to distribute time information over the Internet
  - Aims
    - To provide a service enabling clients across the Internet to be synchronized accurately to UTC
    - To provide a reliable service that can survive lengthy losses of connectivity
    - To enable clients to resynchronize sufficiently frequently to offset the rates of drift found in most computers
    - To provide protection against interference with the time service, whether malicious or accidental

Clock synchronization…

- Network protocol time (NTP)…
  - NTP service is provided by a network of servers connected in logical hierarchy, called synchronization subnet
    - Primary servers
      - Connected directly to a time source, e.g, radio clock receiving UTC
      - Occupy stratum 1
    - Secondary servers
      - Synchronized with primary servers
      - Occupy stratum 2
    - ….
    - Leaf (lowest-level) servers
      - Execute in users' workstations

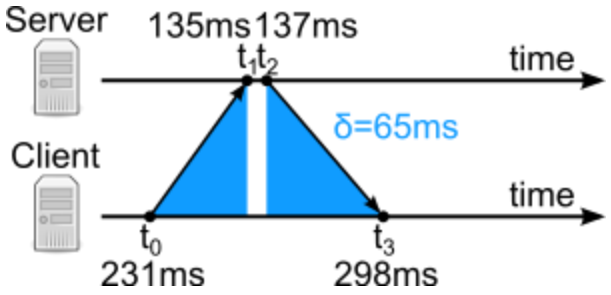


Arrows denote synchronization control, numbers denote strata.

Clock synchronization…

- Network protocol time (NTP)…
  - For a client to synchronize its clock with a remote server, the client must compute
    - Round trip delay, and
    - Offset
  - Round trip delay

    delay = $(t_3-t_0) -(t_2-t_1)$
    - t_3-t_0 is the time elapsed on the client side between the request and reception of the response
    - t_2-t_1 is the time the server waited before sending the answer
  - Offset



$$\theta = \frac{(t_1 - t_0) + (t_2 - t_3)}{2}$$

Clock synchronization…

- Logical clocks
  - Keep track of each other's events rather than maintaining accurate (absolute) clock
  - Observations (Lamport)
    - If two processes do not interact, it is not necessary that their clocks be synchronized
    - For some processes its enough to agree on the order in which events occur rather than exactly what time it is

Clock synchronization…

- Logical clocks – Lamport's logical clock
  - Happens-before relationship
    - a→b, a happens before b
    - Situations
      - If a and b are events in the same process, and a comes before b, then a → b
      - If a is the event of a message being sent, and b is the event of the message being received, then a → b
      - If a → b and b → c, then a → c
      - If events a and b happen in different processes that do not exchange messages, then neither a → b nor b → a are true
    - This introduces a **partial ordering of events** in a system with concurrently operating processes

Clock synchronization…

- Logical clocks – Lamport's logical clock…
  - Problem

    How do we maintain a global view on the system's behavior that is consistent with the happened-before relation?

  - Solution
    - Attach a timestamp C(e) to each event, e, satisfying the following properties

      P1: If a and b are events in the same process and a → b, then C(a) < C(b)

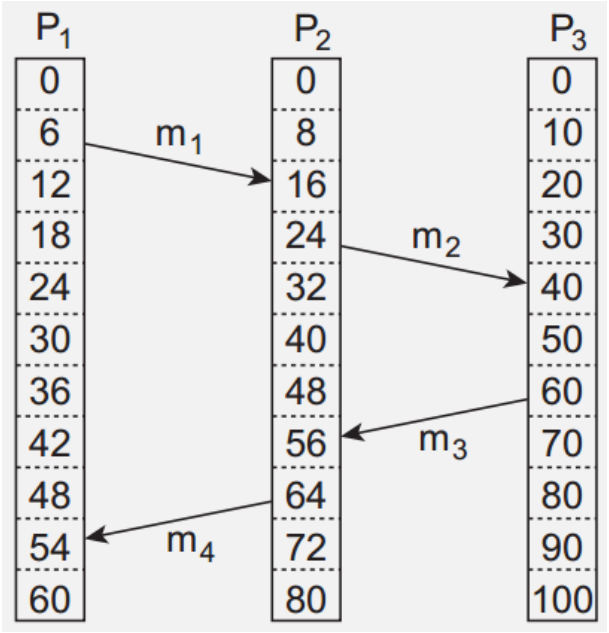
      P2: If a is the sending of a message and b is the reception of that message, then C(a) < C(b)

      P3: The clock time, C, must always go forward (increasing), never backward (decreasing)

      » Correction is done only by adding a positive value

Clock synchronization…

- Logical clocks – Lamport's logical clock…
  - Problem
    - How to attach a timestamp when there is no global clock?



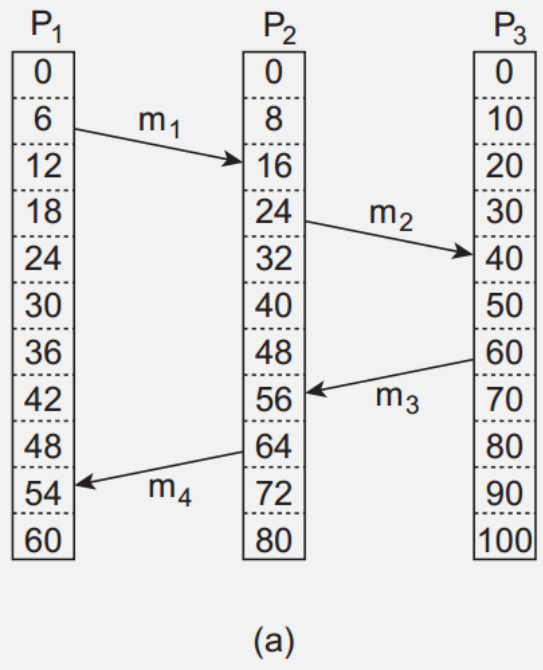
Clock synchronization…

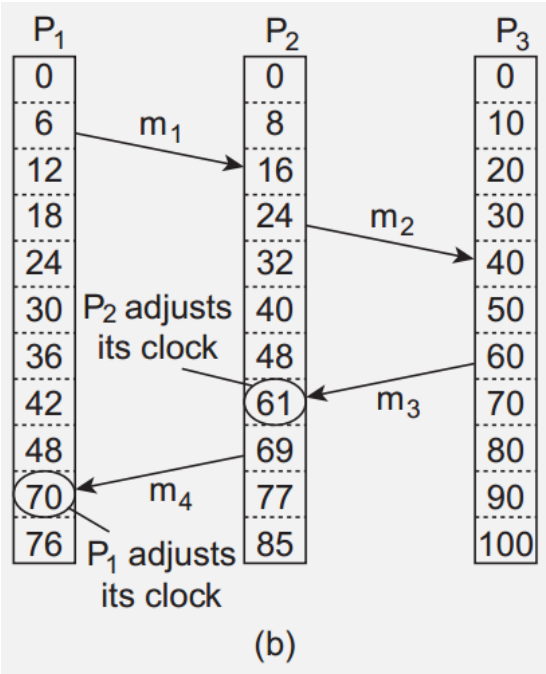
- Logical clocks – Lamport's logical clock…
  - Solution
    - Each process P_i maintains a local counter C_i and updates the counter according to the following rules
      1. Before executing an event, P_i increments C_i by 1
      2. Each time a message m is sent by process P_i, the message is assigned a timestamp $ts(m)=C_i$ (executed after Rule 1)
      3. Whenever a message m is received by a process P_j, P_j adjusts its local counter C_j to $\max\{C_j, ts(m)\}$; then executes the first step and delivers the message to the application

  - Property P1 is satisfied by Rule 1, and property P2 by Rules (2) and (3)

Clock synchronization…

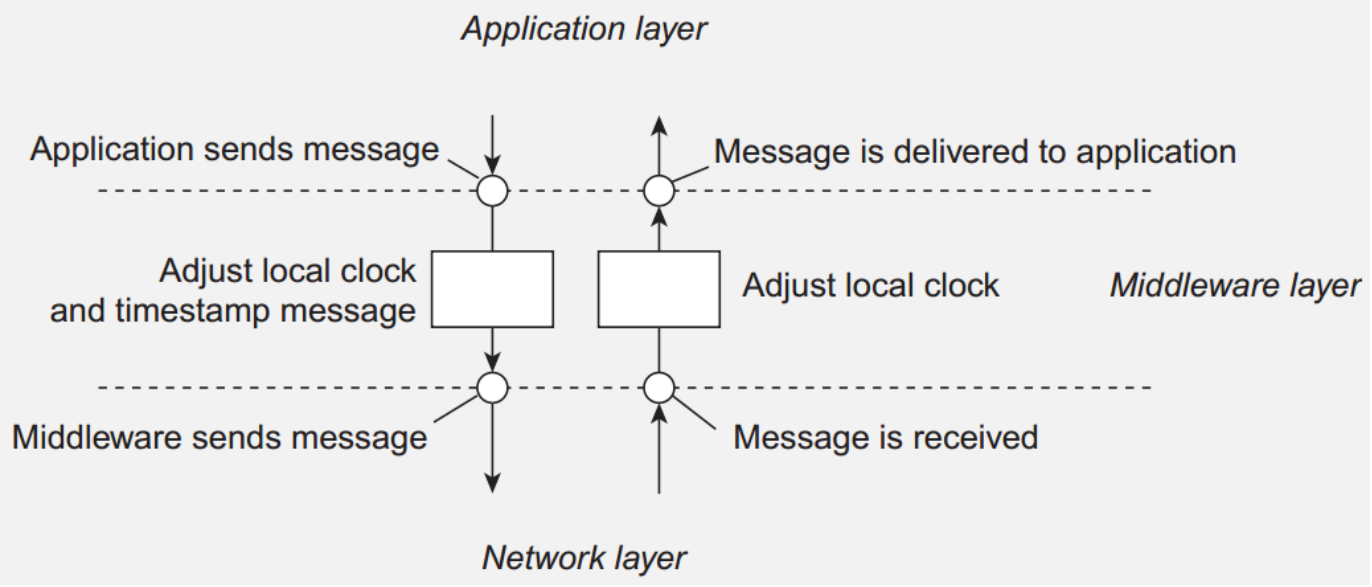
- Logical clocks – Lamport's logical clock…
  - Example





Clock synchronization…

- Logical clocks – Lamport's logical clock…
  - Adjustment takes place at the middleware layer

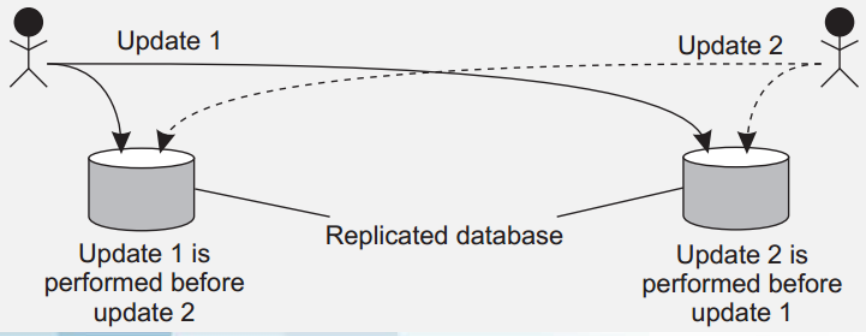


Clock synchronization…

- Logical clocks – Lamport's logical clock…
  - Totally ordered logical clocks
    - Distinct events generated by different process could have identical Lamport timestamp
    - Problem
      - How to create a total order on the set of events?
    - Solution
      - Take the identifier of the processes at which the event occurred into account
        - » For event, e, at process p_i with timestamp T_i, and event, e', at process p_j with timestamp T_j, the global logical timestamp for the events will be (T_i, i) and (T_j, j)
        - » $(T_i, i) < (T_j, j)$ iff either $T_i < T_j$ or $T_i = T_j$ and $i < j$

Clock synchronization…

- Logical clocks – Lamport's logical clock…
  - Totally ordered multicasting
    - Consider a database replicated across several sites for the purpose of improved query performance
    - Scenario
      - P1 adds $100 to an account whose initial value is $1000, while P2 adds an interest of 1% to the same account
    - Question: What would be the balance on the account?
    - Result: Replica #1 will have $1,111, while replica #2 will have $1,110



Clock synchronization…

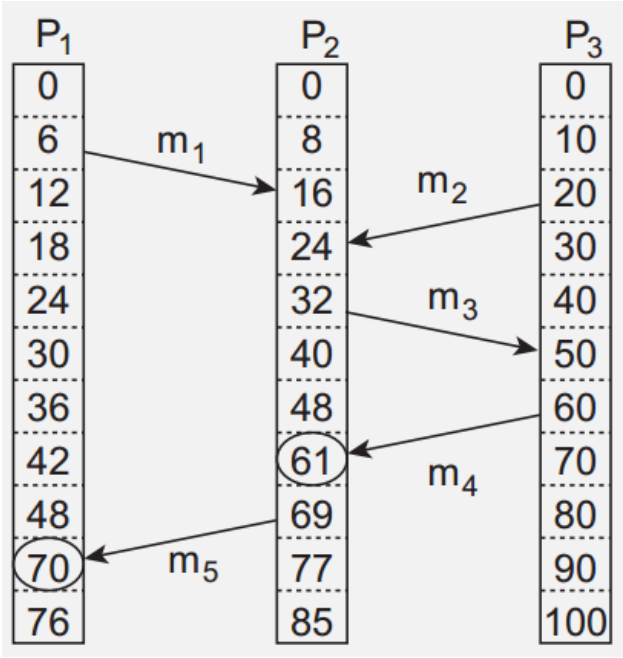
- Logical clocks – Lamport's logical clock…
  - Totally ordered multicasting…
    - Problem
      - Two update operations should have been performed in the same order at each copy
    - Solution
      - Totally-ordered multicast, i.e., all messages are delivered in the same order to each receiver
        - » Each process p_i always sends timestamped message, msg_i
        - » A multicast message is also sent to the sender and put in the local queue, q_i
        - » When a process, p_j, receives a message, it is put into a local queue, q_j, ordered according to its timestamp, and acknowledged to every other process
      - If Lamport algorithm is used, all processes will eventually have the same copy of the local queue

Clock synchronization…

- Logical clocks – Lamport's logical clock…
  - Totally ordered multicasting…
    - p_j passes a message msg_i to its application if
      - msg_i is at the head of $queue_j$, and acknowledged by each other process
      - For process p_k, there is a message msg_k in $queue_j$ with a larger timestamp ($k \neq i$)
    - Assumption
      - Message from the same sender are received in the order they were sent, i.e., FIFO order
      - No message is lost, i.e., communication is reliable

Clock synchronization…

- Logical clocks – Vector clocks
  - Recap: Lamport logical clock
    - If a → b, then C(a) < C(b)
  - Question
    - Can we say if C(a) < C(b), then a → b?
  - Answer
    - Not necessarily



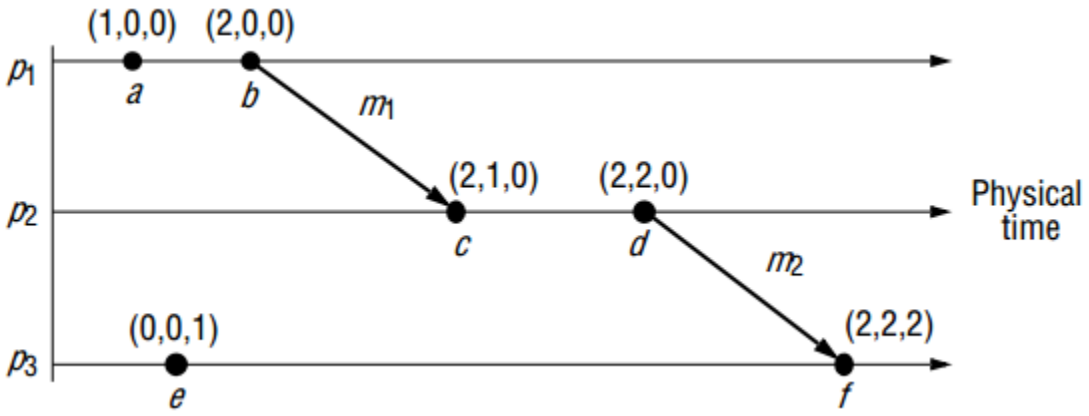
  - Problem
    - Lamport logical clock do not capture causality
  - Solution
    - Vector clocks – capture potential causality

Clock synchronization…

- Logical clocks – Vector clocks…
  - A vector clock for a system of N processes is an array of N integers
  - Each process keeps its own vector clock, V_i, which it uses to timestamp local events
  - Properties
    - $V_j[i]$ is the number of events that have occurred so far at p_i, i.e., $V_j[i]$ is the local logical clock at process p_i
    - If $V_i[j] = k$ then p_i knows that k events have occurred at p_j, i.e., it is p_i's knowledge of the local time at p_j
  - Processes piggyback vector timestamps on the messages they send to one another

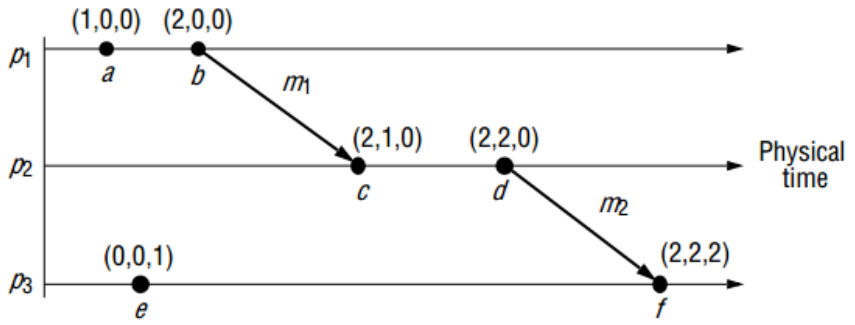
Clock synchronization…

- Logical clocks – Vector clocks…
  - Rules for updating the clocks
    - R1: Initially, $V_i[j]=0$, for i, j=1,2,…,N
    - R2: Just before p_i timestamps an event, e, it sets $V_i[i]=V_i[i] + 1$
    - R3: p_i includes the value $ts(m) =V_i$ in every message it sends (ts = timestamp)
    - R4: When p_i receives a timestamp ts(m) in a message, it sets $V_i[j]=max\{V_i[j], ts[j]\}$, for j=1,2,…,N
    - R5: When a process p_j delivers a message m that it received from p_i to the application, it increments $V_j[j]$ by 1
  - Example



Clock synchronization…

- Logical clocks – Vector clocks…
  - ts(m)[i]-1
    - denotes the number of events processed at p_i that causally precede m
  - $V_i[j]$ $(i \neq j)$
    - is the number of events that have occurred at p_j and have potentially affected p_i
  - Comparing vector timestamps
    - $V = V'$ iff $V[j] = V'[j]$ for $j = 1,2,\ldots,N$
    - $V \leq V'$ iff $V[j] \leq V'[j]$ for $j = 1, 2, \ldots, N$
    - $V < V'$ iff $V \leq V'$ and $V \neq V'$
  - Example
    - $V(a) < V(f) \Rightarrow a \rightarrow f$
    - Neither $V(c) \leq V(e)$ nor $V(e) \leq V(c)$ hence $c \,||\, e$

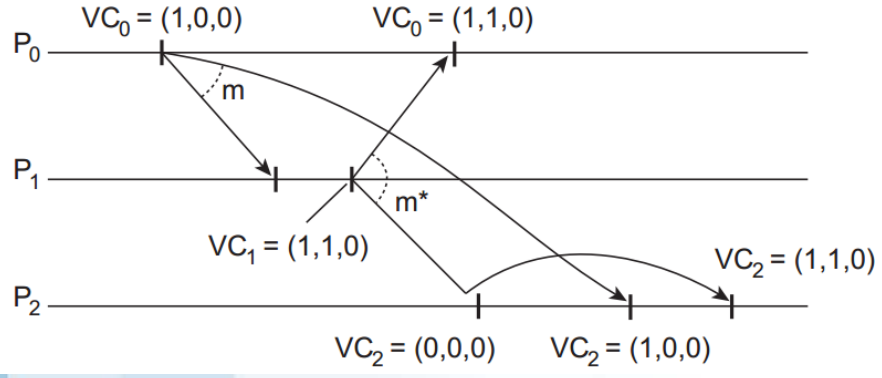


Clock synchronization…

- Logical clocks – Vector clocks…
  - Causally ordered multicasting
    - We can now ensure that a message is delivered *only if all causally preceding messages have already been delivered*
    - Assumption
      - Messages are multicast within a group of processes
        - » If two messages are not related, they could be delivered at any order at different locations
      - Clocks are adjusted when sending and receiving messages, i.e.,
        - » p_i increments $V_i[i]$ only when sending a message
        - » p_J adjusts V_J when receiving a message (i.e., no change to $V_J[j]$)
    - p_J delays delivery of the message, m, from p_i to the application layer until
      - $ts(m)[i] = V_J[i] + 1$
      - $ts(m)[k] \leq V_J[k]$ for $k \neq i$

Clock synchronization…

- Logical clocks – Vector clocks…
  - Causal ordered multicasting…
    - Example 1
      - Assumption: Originally all VC_i have a 0 vector
      - Delivery of m* at P_2
    - Example 2
      - Take VC_2 =[0,2,2], ts(m)=[1,3,0] from P_0
        - » What information does P_2 have and what will it do when receiving m from P_0?



Clock synchronization…

- Logical clocks – Vector clocks…
  - Disadvantage of vector timestamps (compared with Lamport timestamps)
    - Vector timestamps take up an amount of storage and message payload proportional to number of processes
    - Performance is dictated by the weakest link
  - Support to totally-ordered and causally-ordered multicasting
    - In middleware (as part of message-communication layer) vs in application
      - Cons
        - » Only potential causality is captured => overly restrictive + efficiency problem
        - » Not all causality may be captured (e.g., causality due to external communication)
      - Pros
        - » Convenience for the developer

Mutual exclusion

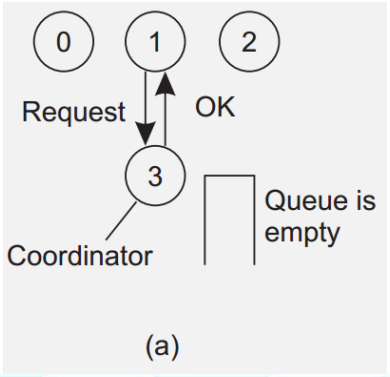
- Fundamental to DS is concurrency and collaboration among multiple processes
- Processes need to simultaneously access the same resource

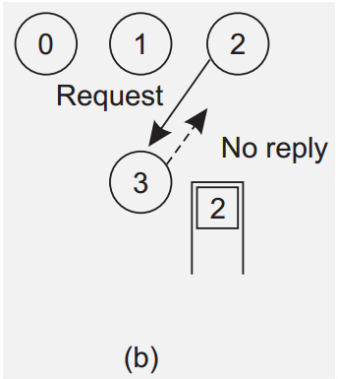
- Problem: Concurrent access could corrupt resource or make it inconsistent

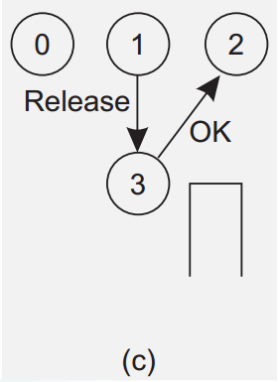
- Solution: Mutual exclusion
  - A condition in which there is a set of processes, only one of which is able to access a given resource or perform a given function at any time
  - Implementations
    - Centralized algorithms
    - Decentralized algorithms
    - Distributed algorithms

Mutual exclusion…

- Centralized algorithm
  - One process is elected as the coordinator
  - Idea
    - Send a request message to the coordinator stating which resource a process wants to access and asking for permission (a and b in the fig.)
    - Coordinator could
      1. Send back a reply granting permission
      2. Deny permission by
         a) Not sending a reply, thus blocking the requesting process
         b) Send back a reply saying “permission denied”







Mutual exclusion…

- Centralized algorithm…
  - Advantage
    - Its fair
      - Requests are granted in the order in which they are received
      - No starvation
    - Easy to implement
    - Requires three messages per use of a resource (request, grant, release)
  - Disadvantage
    - Coordinator is single point of failure
    - Difficult to distinguish a dead coordinator from “permission denied”
    - Single coordinator can become a performance bottleneck

Mutual exclusion…

- Decentralized algorithms
  - Voting algorithm
    - Assumes every resource is replicated *n* times, with each replica having its own coordinator
    - Access to a resource requires a majority vote m > n/2
      - The coordinator notifies the requester when it has been denied access as well as when it is granted
      - Requester must "count the votes", and decide whether or not overall permission has been granted or denied
    - If a process (requester) gets fewer than m votes it will wait for a random time and then ask again

Mutual exclusion…

- Decentralized algorithms…
  - Voting algorithm…
    - Assumption
      - When a coordinator crashes, it recovers quickly but will have forgotten any vote it gave before it crashed
        - » Problem: It could grant permission to other requester again (but this is less probable)
    - Advantage
      - Less vulnerable to failures of a single coordinator
    - Disadvantage
      - If a resource is in high demand, multiple requests will be generated
      - It's possible that processes will wait a long time to get permission
      - Could lead to dead lock

Mutual exclusion…

- Distributed algorithm
  - Ricart and Agrawala's algorithm
    - Requires that there be a total ordering of all events in the system

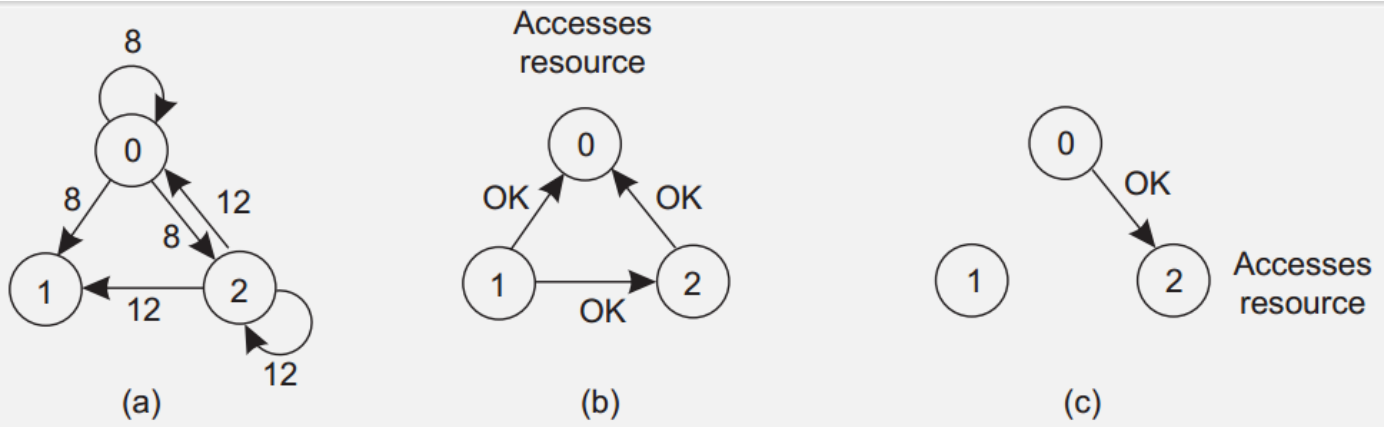
      => No ambiguity on which event happens first
      - Could be achieved using Lamport's logical clock algorithm
    - Assumption
      - Communication is reliable, i.e., no message is lost

Mutual exclusion…

- Distributed algorithm…
  - Ricart and Agrawala's algorithm…
    - How it works
      - When a process wants to access a shared resource, it builds a message containing
        - » Name of the resource
        - » Its process ID
        - » Current (logical) time
      - Message is sent to all processes, including itself
      - Process replies an OK message, i.e., grants access, to a request only when
        1. The receiver is not accessing the resource and doesn't want to access it
        2. The receiver process is waiting for the resource, but has lower priority (known through comparison of timestamp)
        3. For all other cases, the request is queued

Mutual exclusion…

- Distributed algorithm…
  - Ricart and Agrawala's algorithm…
    - Example
      - Process 0 and 2 sends a request to access a resource at the same time. Process 1 is not interested on the resource

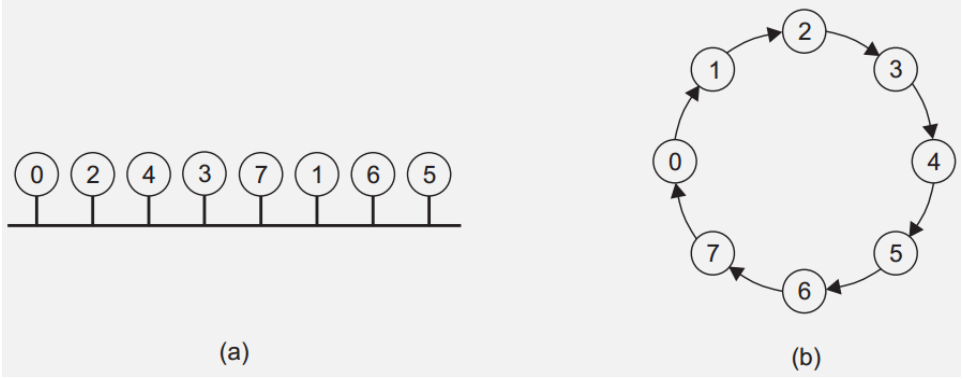


Mutual exclusion…

- Distributed algorithm…
  - Ricart and Agrawala's algorithm…
    - Number of messages required per entry is 2(n-1), n = total number of processes
    - Advantage
      - Mutual exclusion is guaranteed without deadlock or starvation
      - No single point of failure
    - Disadvantage
      - n points of failure
        - » Could be addressed if the receiver always sends a reply (grant or deny)
      - Either multicast communication must be used or each process must maintain the group membership
        - » Works best with small groups of processes that never change their group membership
      - All processes are involved in the decision (if one process has a performance issue its likely that others will have too)

Mutual exclusion…

- Distributed algorithm…
  - A Token ring algorithm
    - Processes are organized in a logical ring, and let a token be passed between them
    - Its important for a process to know who is next in line after itself (not the order)



Mutual exclusion…

- Distributed algorithm…
  - A Token ring algorithm…
    - How it works
      - When the ring is initialized, process 0 is given a token
        - » The token is passed from process k to process k+1 (modulo ring size)
      - When a process acquires the token from its neighbor, it checks if it needs to access the shared resource
        - » If yes, it goes ahead and passes the token to the next process when it finishes
        - » If not, the token is passed to the next process in the ring

Mutual exclusion…

- Distributed algorithm…
  - A Token ring algorithm…
    - Advantage
      - No starvation can occur
    - Disadvantage
      - Difficult to detect when a token is lost
      - Process could crash
        - » Could be detected
          - if acknowledgement is sent
          - When a neighbor wants to give it a token
      - Consumes network bandwidth

Mutual exclusion…

- Comparison of the algorithms
  - Assumption: Messages are passed sequentially over a network

Algorithm	# of msgs per entry and exit of a critical section	Delay before entry (in msg times)	Example problems
Centralized	3	2	Coordinator crash
Decentralized	2mk + m, k= # of attempts	2mk	Starvation, low efficiency
Ricart and Agrawala's	2(n-1)	2(n-1)	Crash of any process
Token ring	1 to ∞	0 to n-1	Lost token, process could crash

Elections

- Many distributed algorithms require one process to act as a coordinator, initiator, or perform some special role
  - Example
    - Berkley algorithm, Centralized mutual exclusion

- Question
  - How to elect this special process dynamically?

- Assumptions
  - Every process knows the process number of every other process
  - A process doesn't know which processes are up or down

- Goal
  - Ensure that when an election starts, it concludes with all processes agreeing on who the new coordinator is to be

Election…

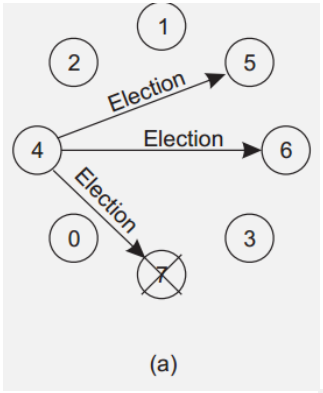
- The bully algorithm
  - Initiates an election when any process notices that the coordinator is no longer responding to a request
  - Gives priority to processes with higher weights (e.g. process numbers)
  - Election is held by a process p as follows
    1. p sends an *election* message to all processes with higher number (priority)
    2. If no one responds, p wins the election and becomes coordinator and sends a victory message to all other processes
    3. If one of the processes with higher priority are up, it sends a take-over message to p. p will then be out of the race

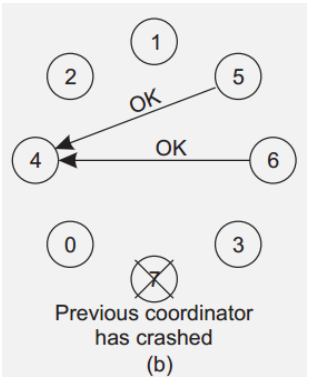
Election…

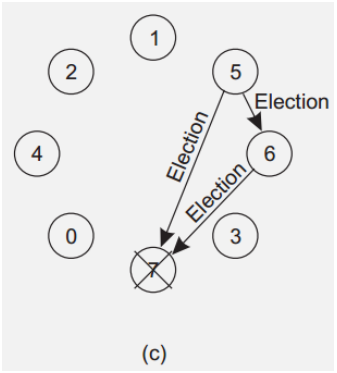
- The bully algorithm…
  - A process can get an election message from one of its lower-numbered (priority) colleagues at anytime
    - If alive, the receiver sends an OK message back to the sender and holds an election (if it didn't hold one before)
  - If a process that was down comes back up, it holds an election

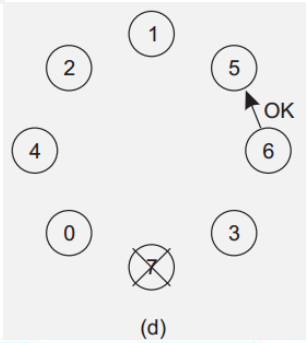
Election…

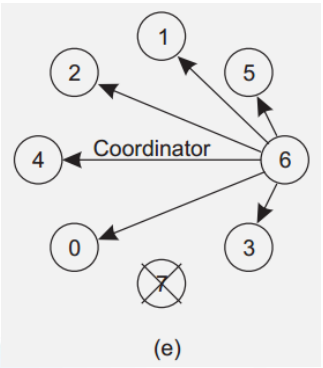
- The bully algorithm…
  - Example
    - Process 4 notices that the coordinator process 7 is down











Election…

- Ring algorithm
  - Goal
    - To elect a single process with the highest priority as the coordinator
  - Assumption
    - The processes are physically or logically ordered, so that each process knows who its successor is
      - Token is not used

Election…

- Ring algorithm…
  - Election can be initiate by any process, p, which notices that the coordinator is not functioning
    - Process, p, starts election by sending an *election* message that contains its process number to its successor
      - If successor is down, it is passed on to the next successor
    - If a message is passed on, the sender adds its own process number to the list in the message
      - When it gets to the initiator, the message contains list of processes that are up (in the ring)
    - The initiator sends a *coordinator* message around the ring containing a list of all living processes. The one with the highest process number (priority) is elected as coordinator

Election…

- Ring algorithm…
  - Example
    - Process 2 and 5 simultaneously notices that the coordinator process 7 is not responding
      - Question. Does it matter if two processes initiate an election simultaneously?