Synchronization in Distributed Systems

CS-4513 Distributed Systems
Hugh C. Lauer

Slides include materials from *Modern Operating Systems*, 3rd ed., by Tannenbaum,
Operating System Concepts, 7th ed., by Silbershatz, Galvin, & Gagne,
Distributed Systems: Principles & Paradigms, 2nd ed. By Tanenbaum and Van Steen, and
Distributed Systems: Concepts and Design, 4th ed., by Coulouris, *et. al.*

Issue

- Synchronization within one system is hard enough
  - Semaphores
  - Messages
  - Monitors
  - …
- Synchronization among processes in a distributed system is much harder

Reading Assignment

- See Coulouris *et al*
  - Chapter 11, *Time and Global States*
  - Chapter 12, *Coordination and Agreement*

- Note that *Atomic Transactions* are an example of coordination and agreement.

Example

- File locking in NFS
  - Not supported directly within NFS v.3
- Need *lockmanager* service to supplement NFS

What about using Time?

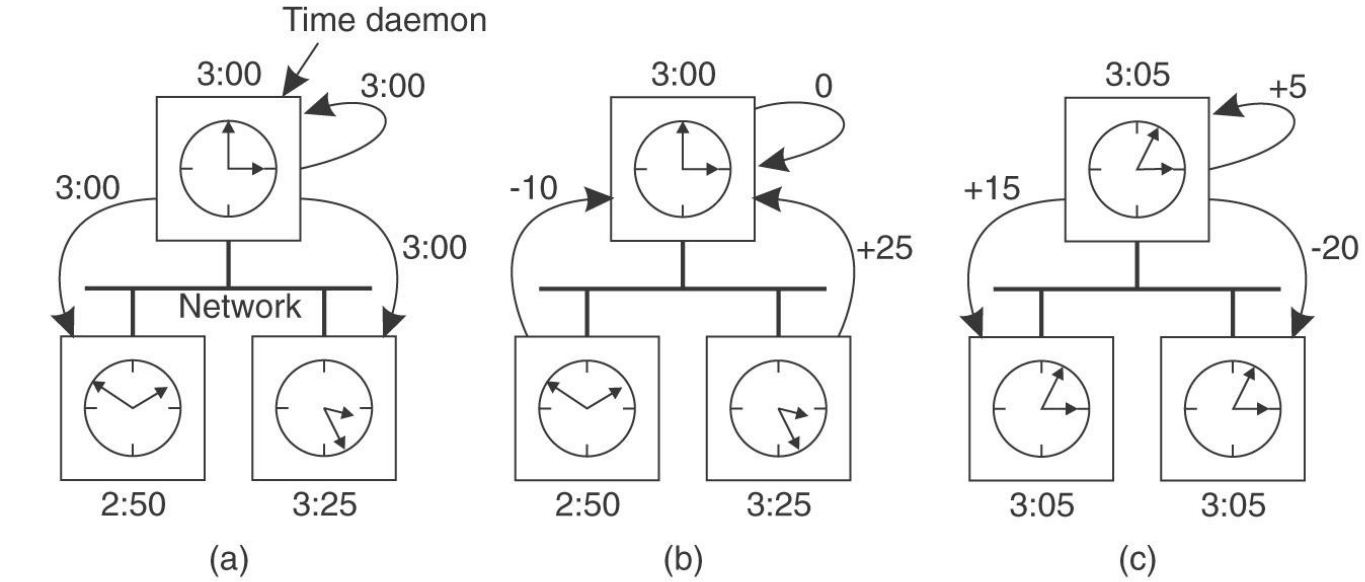
- *make* recompiles if *foo.c* is newer than *foo.o*
- Scenario
  - *make* on machine *A* to build *foo.o*
  - Test on machine *B*; find and fix a bug in *foo.c*
  - Re-run *make* on machine *B*
  - *Nothing happens!*
- Why?

Problem

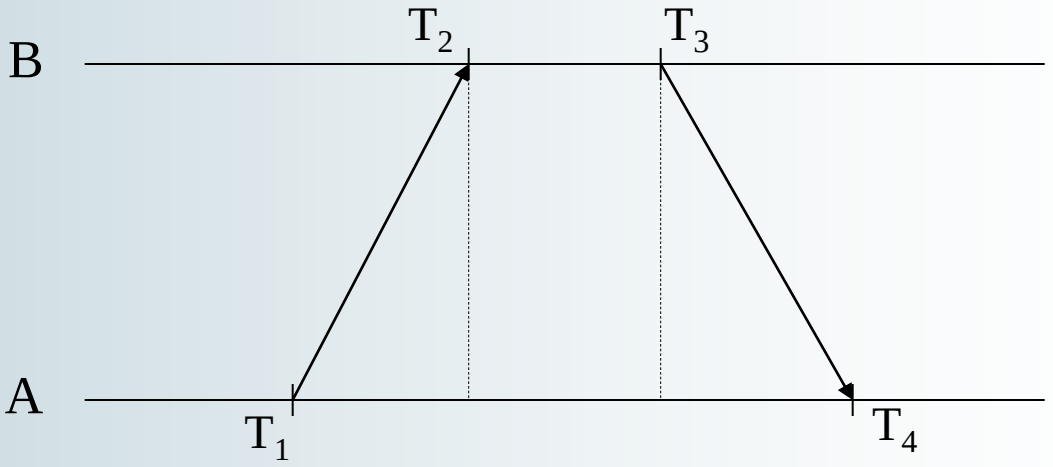
- Time *not* a reliable method of synchronization
- Users mess up clocks
  - (and forget to set their time zones!)
- Unpredictable delays in Internet
- Relativistic issues
  - If *A* and *B* are far apart physically, and
  - two events T_A and T_B are very close in time, then
  - which comes first? how do you know?

Berkeley Algorithm

- Berkeley Algorithm
  - Time Daemon polls other systems
  - Computes average time
  - Tells other machines how to adjust their clocks

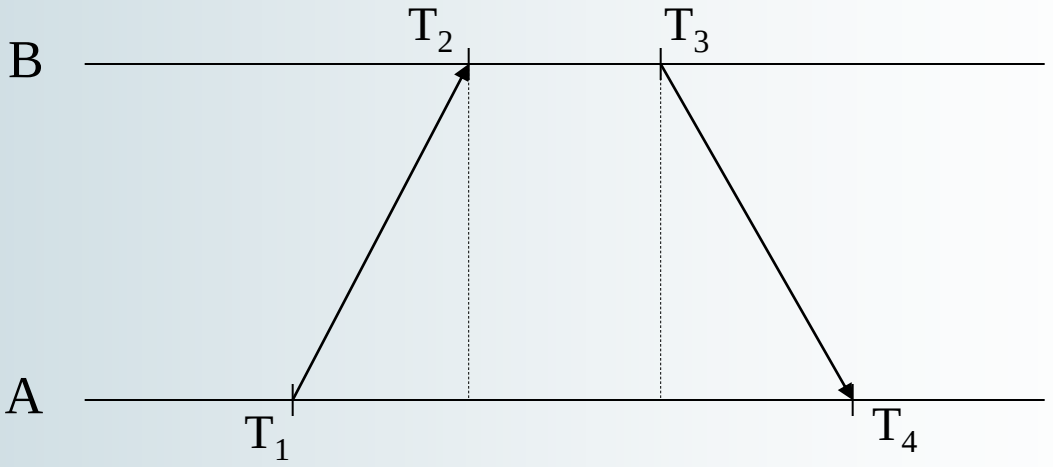


NTP (Network Time Protocol)



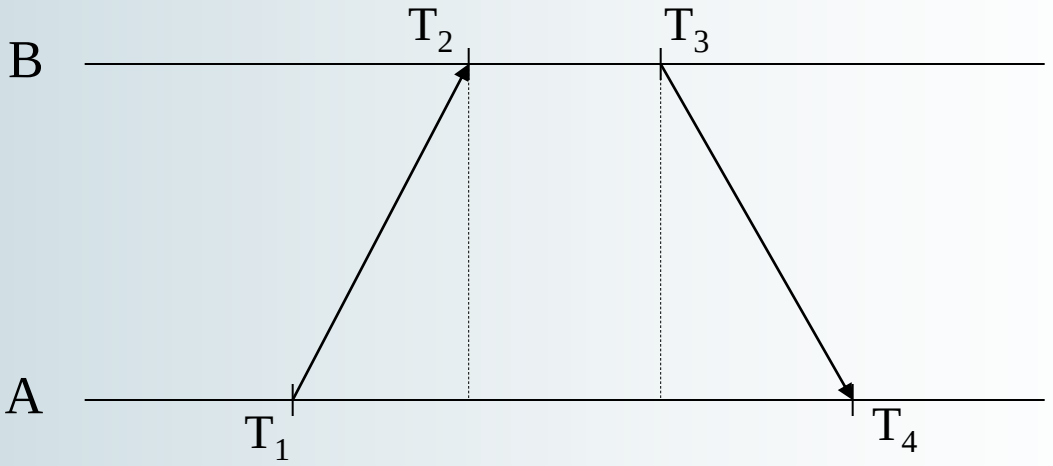
- *A* requests time of *B* at its own T_1
- *B* receives request at its T_2, records T_2
- *B* responds at its T_3, sending values of T_2 and T_3
- *A* receives response at its T_4
- Question: what is $\theta = T_B - T_A$?

NTP (Network Time Protocol)



- Question: what is $\theta = T_B - T_A$?
- Assume transit time is approximately the same both ways
- Assume that *B* is the time server that *A* wants to synchronize to

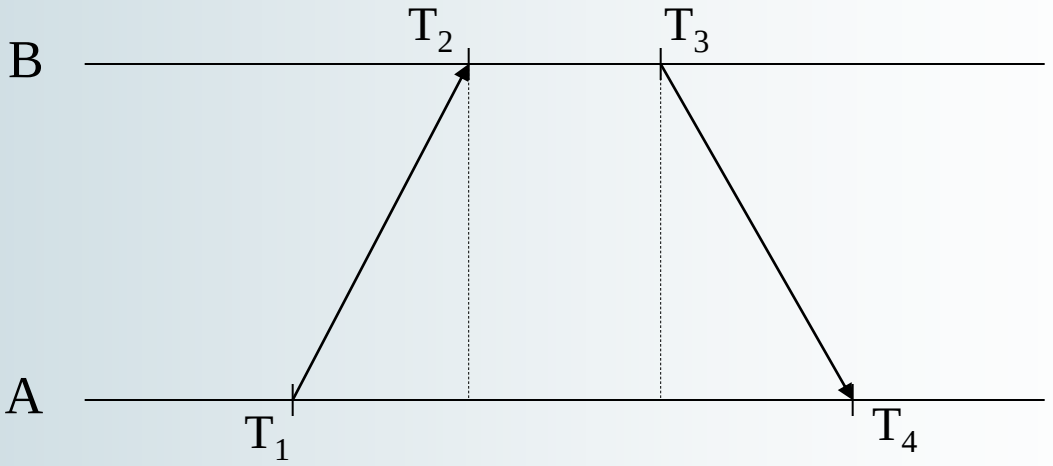
NTP (Network Time Protocol)



- *A* knows ($T_4 - T_1$) from its own clock
- *B* reports T_3 and T_2 in response to NTP request
- *A* computes total transit time of

$$(T_4 - T_1) - (T_3 - T_2)$$

NTP (Network Time Protocol)



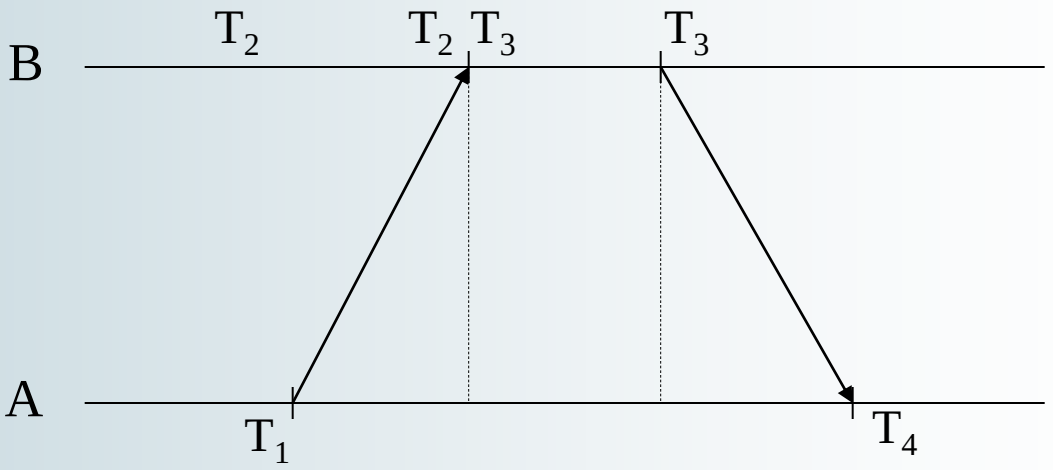
- One-way transit time is approximately ½ total, i.e.,

$$\frac{(T_4 - T_1) - (T_3 - T_2)}{2}$$

- *B*'s clock at T_4 reads approximately

$$T_3 + \frac{(T_4 - T_1) - (T_3 - T_2)}{2} = \frac{(T_4 - T_1) + (T_2 + T_3)}{2}$$

NTP (Network Time Protocol)



- *B*'s clock at T_4 reads approximately (from previous slide)

$$\frac{(T_4 - T_1) + (T_2 + T_3)}{2}$$

- Thus, difference between *B* and *A* clocks at T_4 is

$$\frac{(T_4 - T_1) + (T_2 + T_3)}{2} - T_4 = \frac{(T_2 - T_1) + (T_3 - T_4)}{2}$$

NTP (continued)

- Servers organized as *strata*
  - *Stratum 0* server adjusts itself to WWV directly
  - *Stratum 1* adjusts self to *Stratum 0* servers
  - Etc.
- Within a stratum, servers adjust with each other

Adjusting the Clock

- If T_A is slow, add ε to clock rate
  - To speed it up gradually
- If T_A is fast, subtract ε from clock rate
  - To slow it down gradually

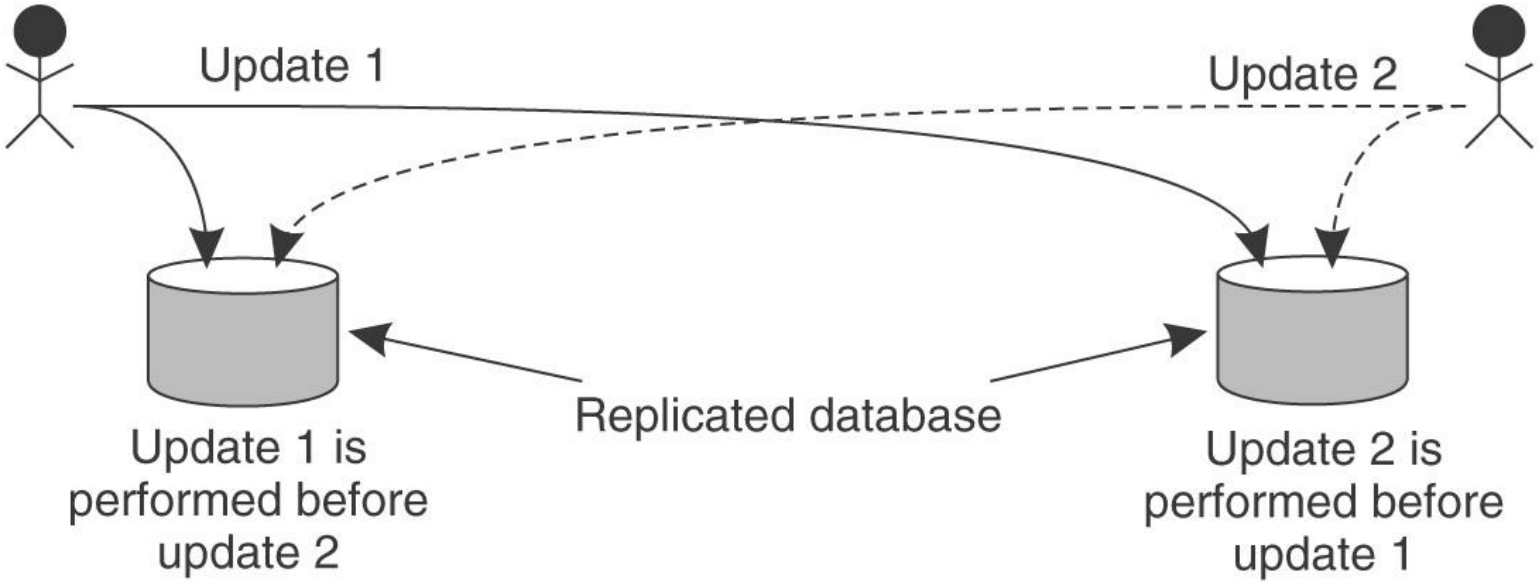
Problem (again)

- All of this helps, but not enough!
- Users mess up clocks
  - (and forget to set their time zones!)
- Unpredictable delays in Internet
- Relativistic issues
  - If *A* and *B* are far apart physically, and
  - two events T_A and T_B are very close in time, then
  - which comes first? how do you know?

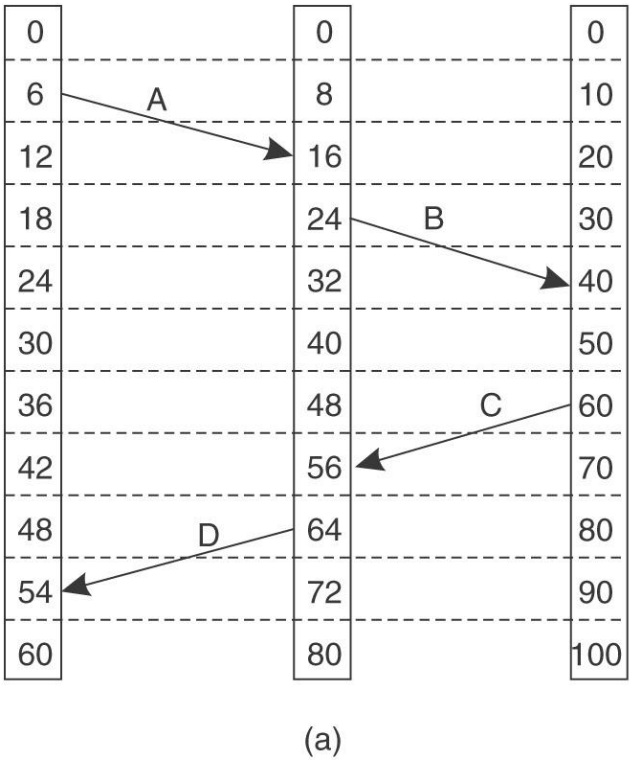
Example

- At midnight PDT, bank posts interest to your account based on current balance.
- At 3:00 AM EDT, you withdraw some cash.
- Does interest get paid on the cash you just withdrew?
- Depends upon which event came first!
- What if transactions made on different replicas?

Example (continued)



Exaggerated View



(a)

It is impossible to conclude anything about order of events by comparing clocks

Solution — Logical Clocks

For example, if *b* is known to be *caused* by something associated with *a*

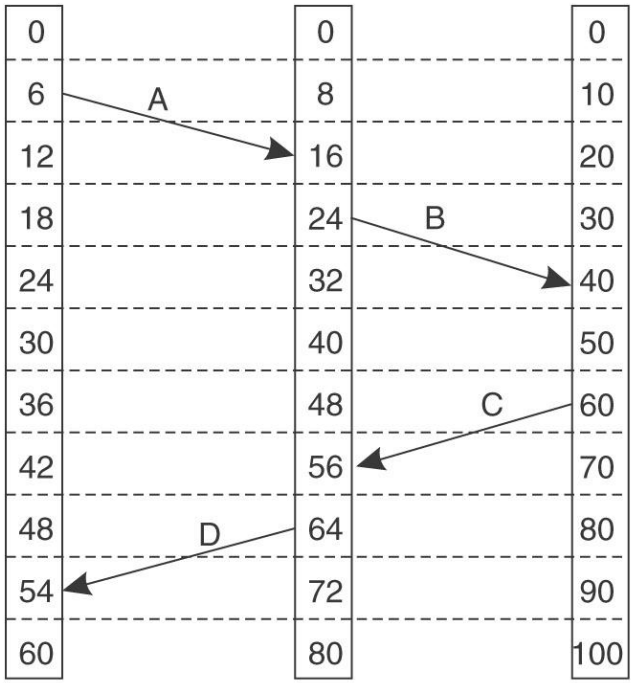
- Not "clocks" at all
- Just monotonic counters
  - Lamport's temporal logic
- Definition: $a \rightarrow b$ means
  - *a* occurs before *b*
  - More specifically, all processes agree that first *a* happens, then later *b* happens
- E.g., *send*(*message*) $\rightarrow$ *receive*(*message*)

Implementation of Logical Clocks

- Every machine maintains its own logical "clock" *C*
- Transmit *C* with *every* message
- If $C_{received} > C_{own}$, then adjust C_{own} forward to $C_{received} + 1$
- Result: Anything that is *known* to follow something else in *time* has larger *logical clock* value.

Logical Clocks (continued)

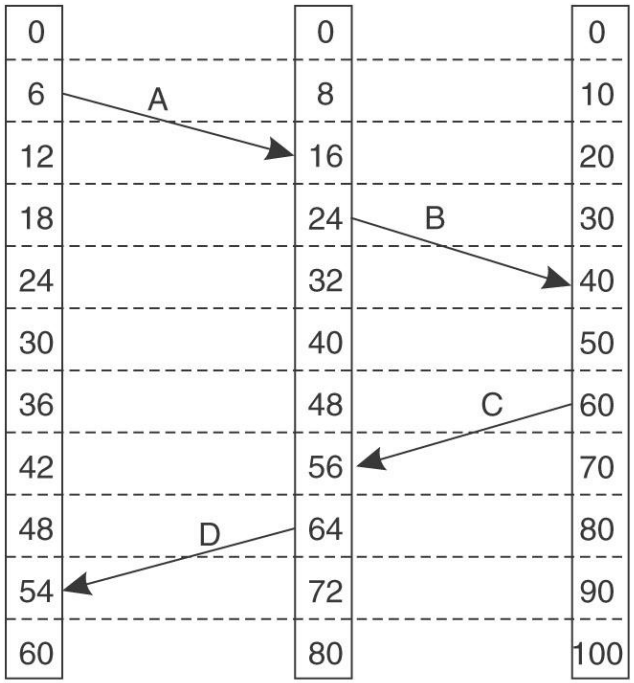
Without *Logical Clocks*



(a)

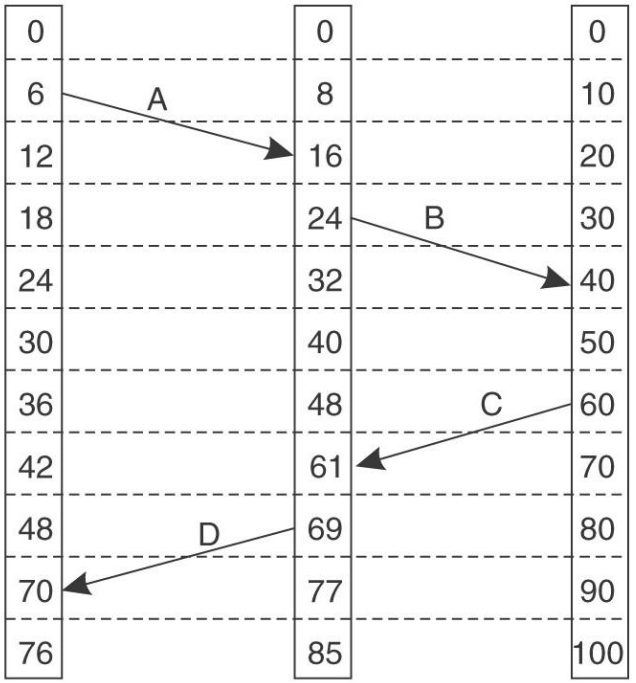
Logical Clocks (continued)

Without *Logical Clocks*



(a)

With *Logical Clocks*



(b)

Variations

- See Coulouris, *et al*, 11.4
- Note: Grapevine *timestamps* for updating its registries behave somewhat like logical clocks.

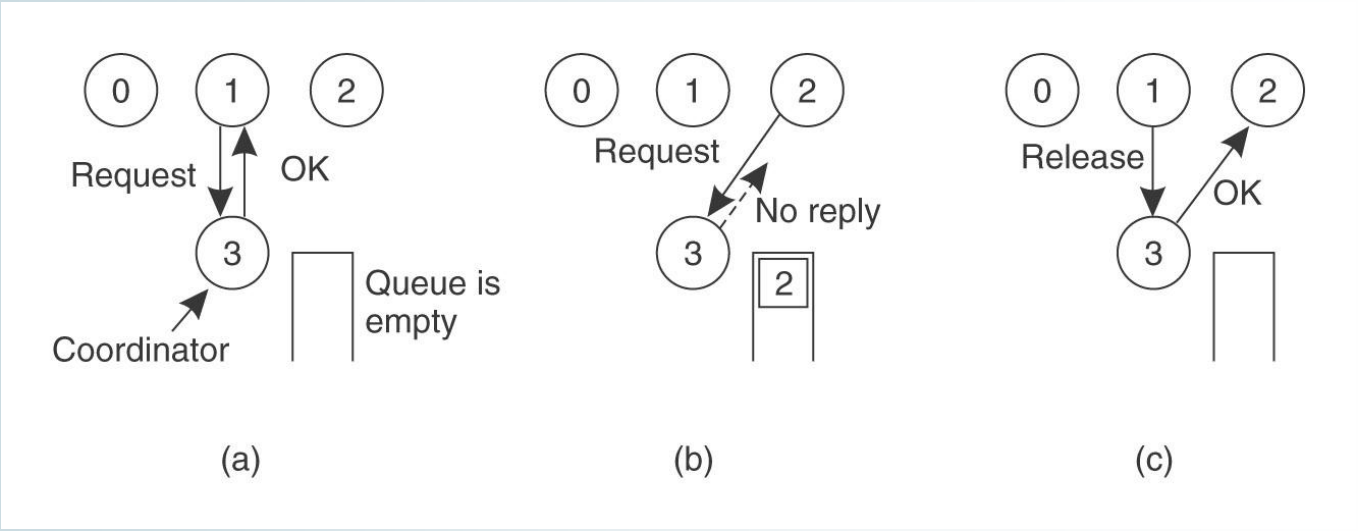
Questions?

Mutual Exclusion in Distributed Systems

- Prevent inconsistent usage or updates to shared data

- Two approaches
  - *Token*
  - *Permission*

Centralized Permissions



- One process is elected *coordinator* for a resource
- All others ask *permission*.
- Possible responses
  - Okay; denied (ask again later); none (caller waits)

Centralized Permissions (continued)

- Advantages
  - Mutual exclusion guaranteed by coordinator
  - “Fair” sharing possible without starvation
  - Simple to implement
- Disadvantages
  - Single point of failure (coordinator crashes)
  - Performance bottleneck
  - ...

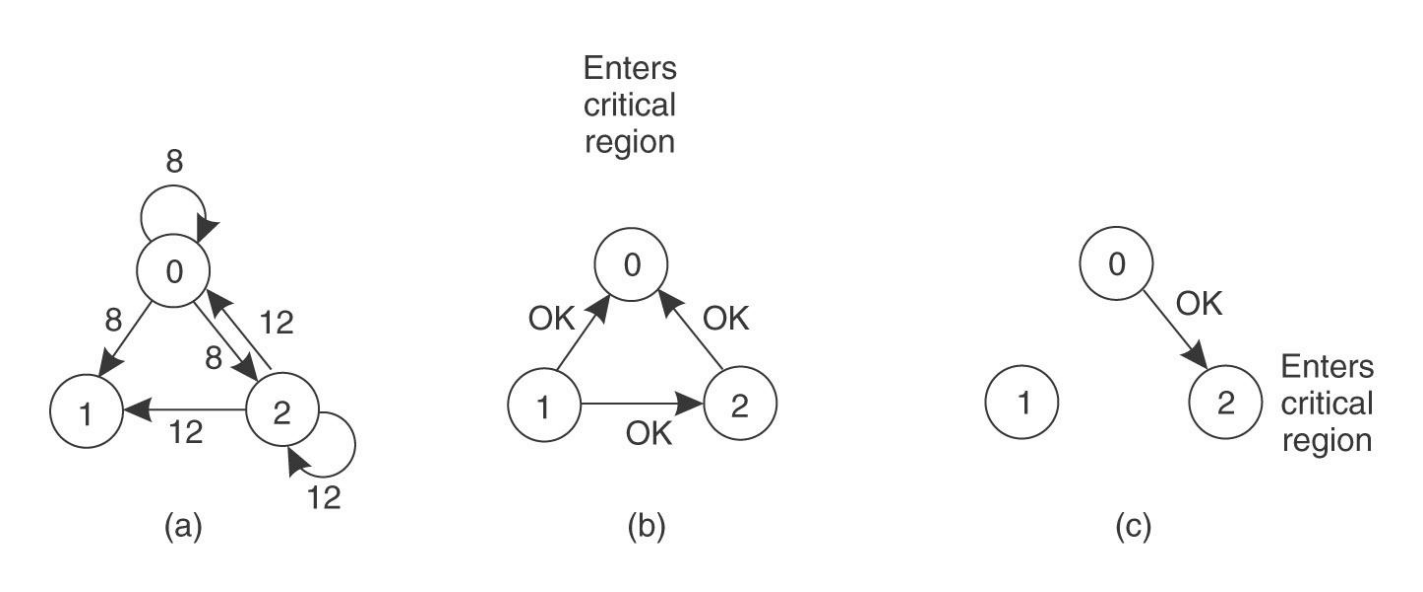
Decentralized Permissions

- *n* coordinators; ask all
  - E.g., *n* replicas
- Must have agreement of $m > n/2$
- Advantage
  - No single point of failure
- Disadvantage
  - *Lots* of messages
  - Really messy

Distributed Permissions

- Use Lamport's *logical clocks*
- Requestor sends reliable messages to *all* other processes (including self)
  - Waits for *OK* replies from all other processes
- Replying process
  - If not interested in resource, reply *OK*
  - If currently using resource, queue request, don't reply
  - If interested, then reply *OK* if requestor is earlier; Queue request if requestor is later

Distributed Permissions *(continued)*



- Process *0* and Process *2* want resource
- Process *1* replies *OK* because not interested
- Process *0* has lower time-stamp, thereby goes first
- …

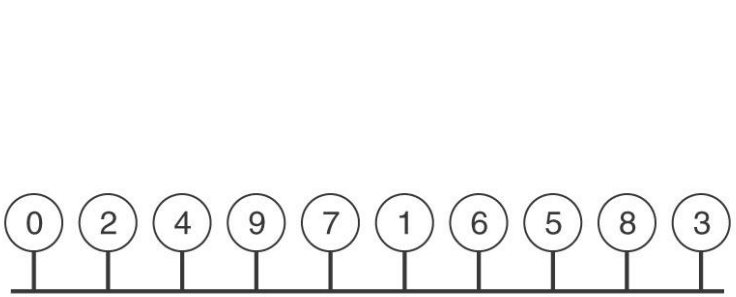
Distributed Permissions (continued)

- Advantage
  - No central bottleneck
  - Fewer messages than Decentralized
- Disadvantage
  - *n* points of failure
  - i.e., failure of one node to respond locks up system

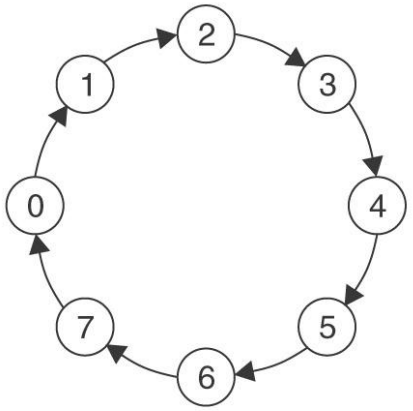
Token system

- Organize processes in logical *ring*
- Each process knows successor
- *Token* is passed around ring
  - If process is interested in resource, it waits for token
  - Releases token when done
- If node is dead, process skips over it
  - Passes token to successor of dead process

Token system (continued)



(a)



(b)

- Advantages
  - Fairness, no starvation
  - Recovery from crashes if token is not lost
- Disadvantage
  - Crash of process holding token
  - Difficult to detect; difficult to regenerate *exactly* one token

Questions?