CS 550:
Advanced Operating Systems

Synchronization

Ioan Raicu
Computer Science Department
Illinois Institute of Technology

CS 550
Advanced Operating Systems
March 1st, 2011

Outline

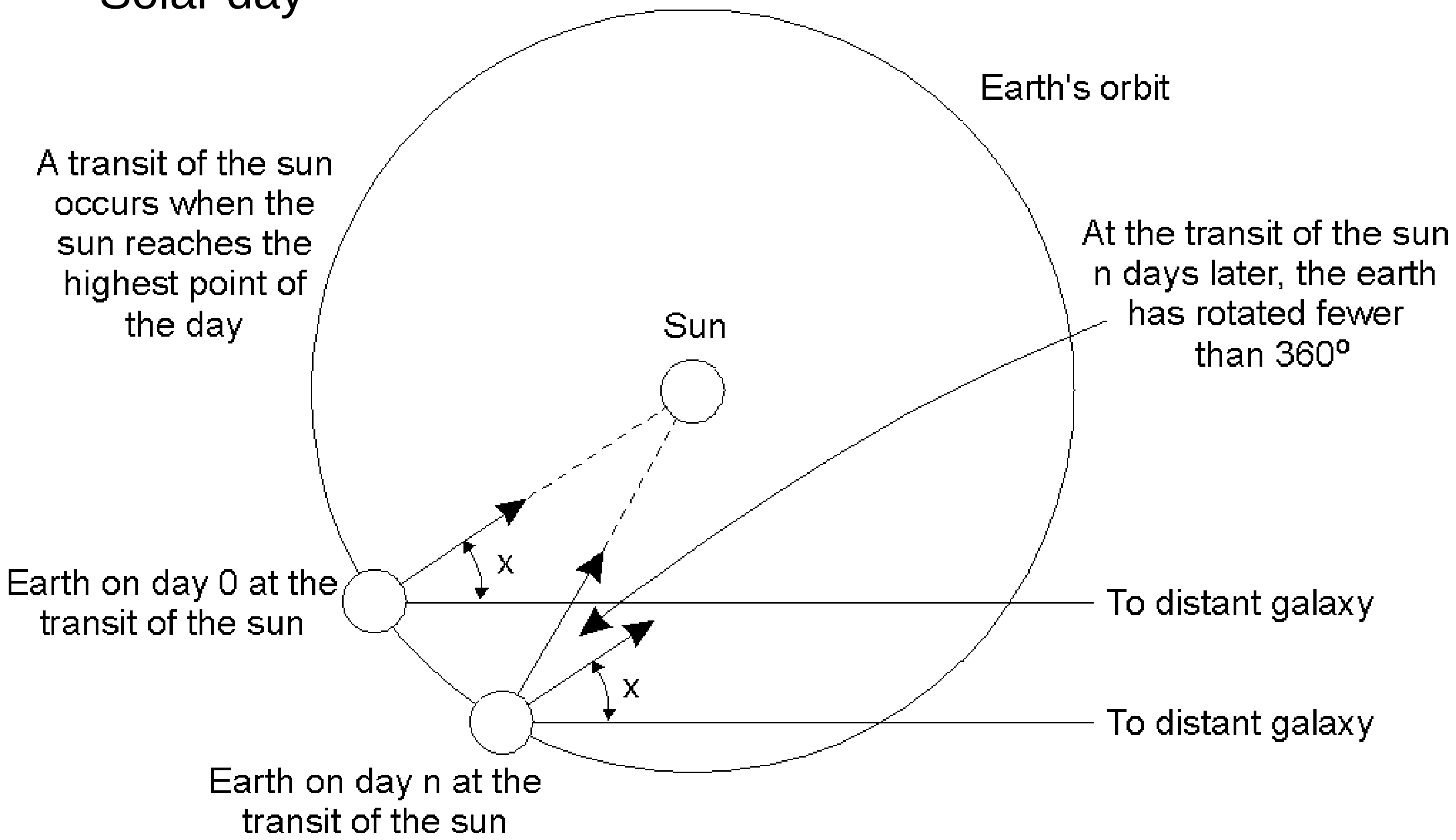
- Clock synchronization
  - Physical clocks
  - Synchronization algorithms
- Logical clock
  - Lamport timestamps
- Election algorithms
  - Bully algorithm
  - Ring algorithm
- Distributed mutual exclusion
  - Centralized algorithm
  - Distributed algorithm
  - Token ring algorithm
- Distributed deadlocks

Canonical Problems in Distributed Systems

- Time ordering and clock synchronization
- Leader election
- Mutual exclusion
- Distributed transactions
- Deadlock detection

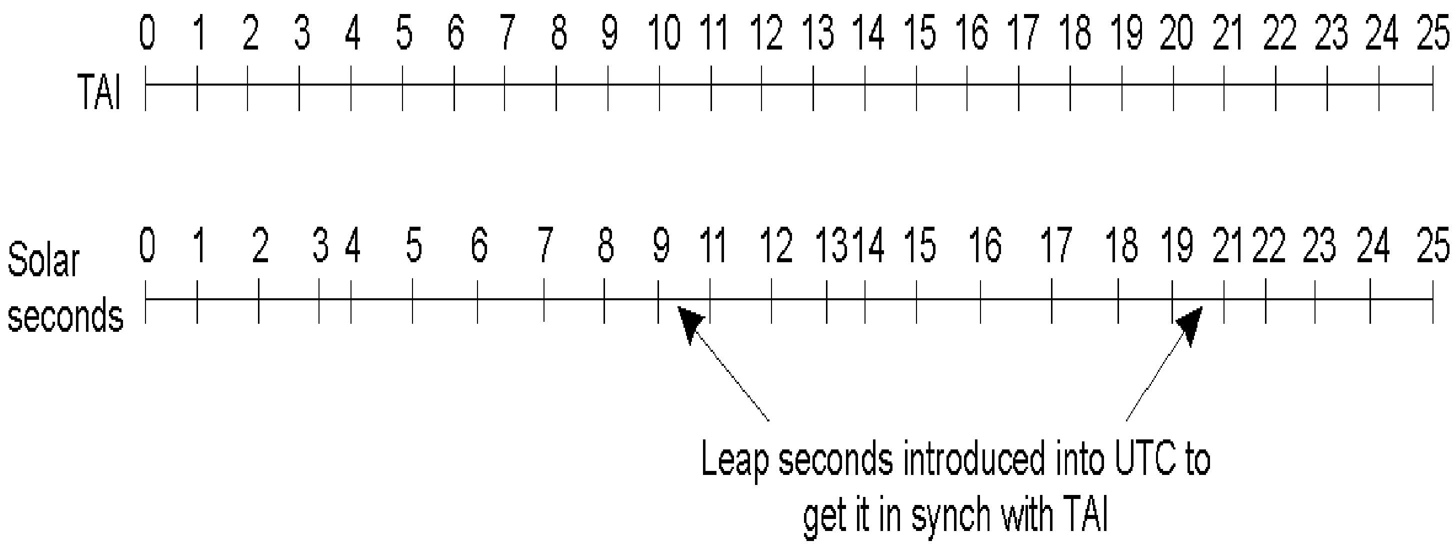
Physical Clocks

- Solar day



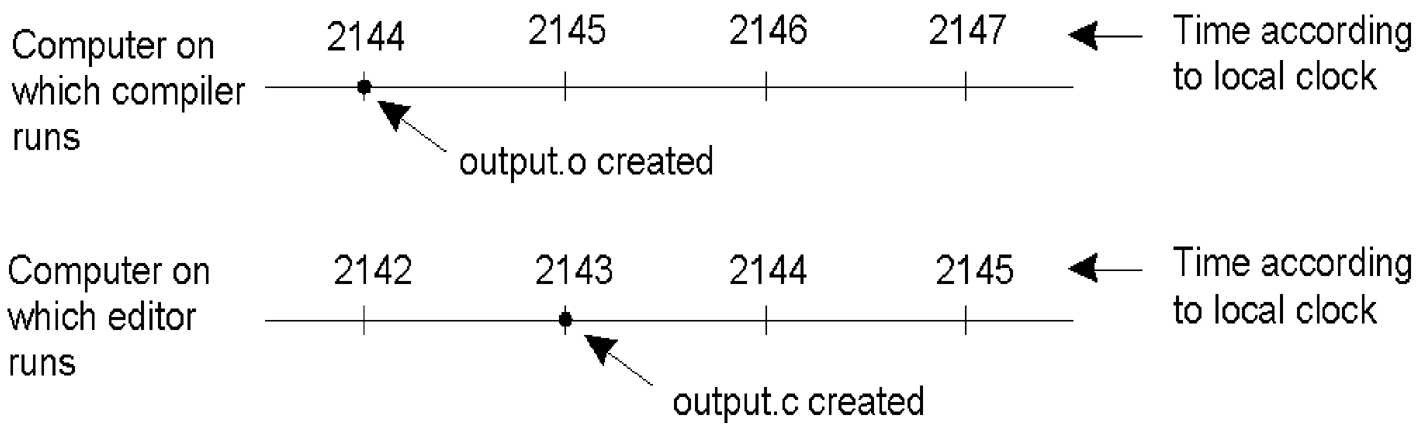
Physical Clocks

- Coordinated universal time *(UTC)* – international standard based on atomic time
  - Add leap seconds to be consistent with astronomical time
  - UTC broadcast on radio (satellite and earth)
  - Receivers accurate to 0.1 – 10 ms



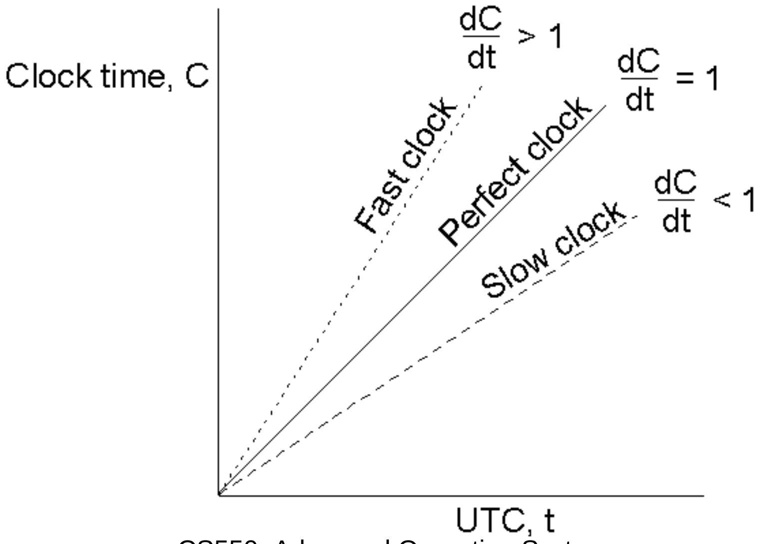
Clock Synchronization

- Time is unambiguous in centralized systems
- Distributed systems: each node has own system clock
  - Crystal-based clocks are less accurate (1 part in million)
  - what is the problem?



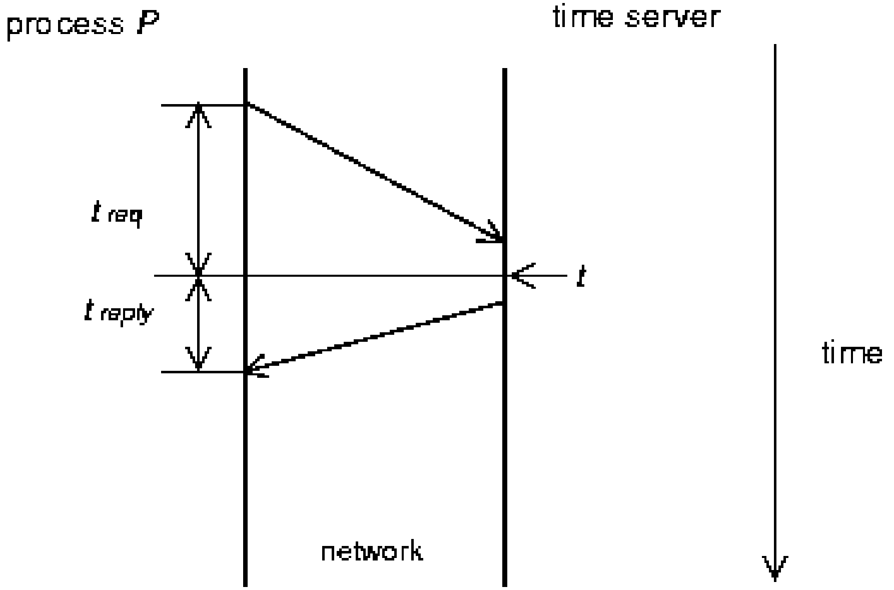
Clock Synchronization

- Each clock has a maximum drift rate ρ
  - $1-\rho <= dC/dt <= 1+\rho$
  - Two clocks may drift by $2\rho\ \Delta t$ in time Δt
  - To limit drift to δ => resynchronize every $\delta/2\rho$ seconds



Cristian's Algorithm

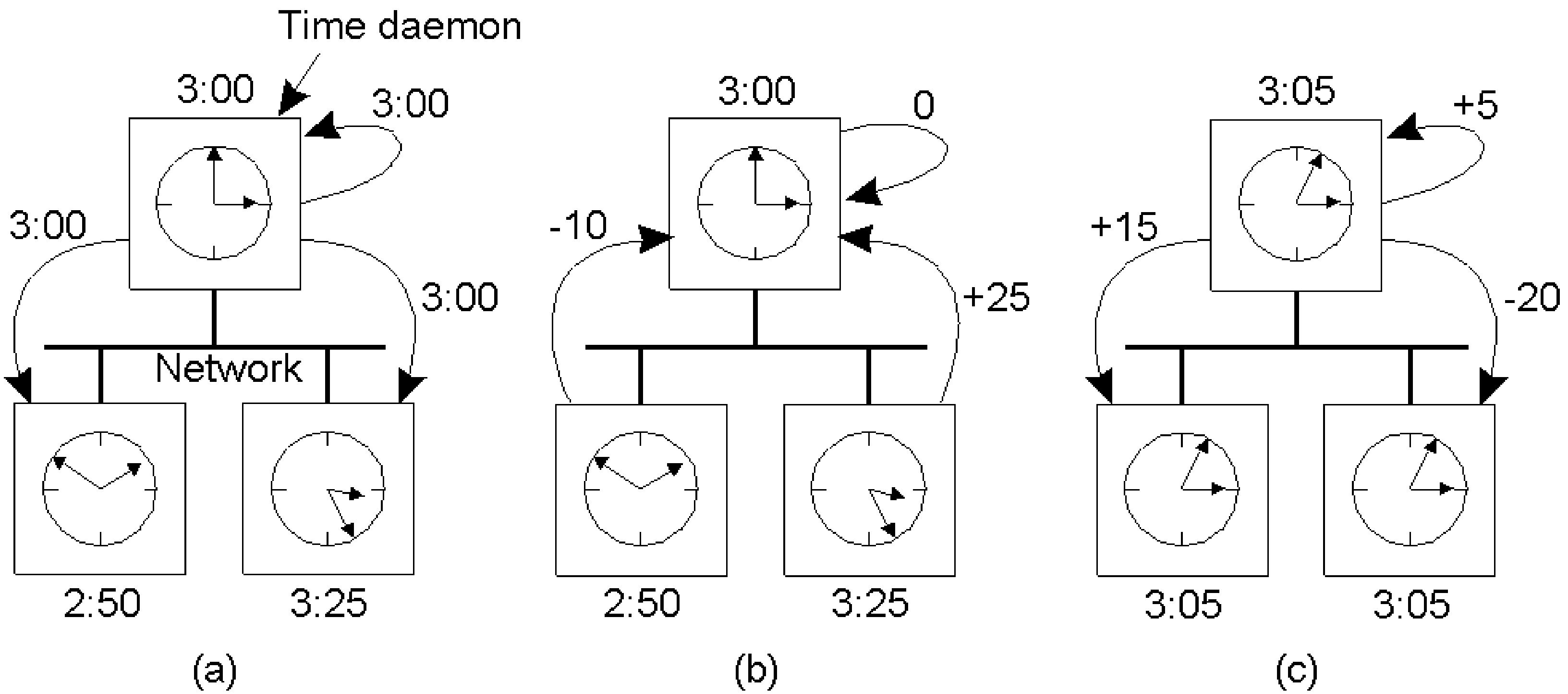
- Synchronize machines to a *time server* with a UTC receiver
- Machine P requests time from server every $\delta/2\rho$ seconds



Berkeley Algorithm

- Used in systems without UTC receiver
  - Keep clocks synchronized with one another
  - One computer is *master*, other are *slaves*
  - Master periodically polls slaves for their times
  - Failure of master => ?

Berkeley Algorithm



Today's Approaches

- Network Time Protocol (NTP)
- Uses advanced techniques for accuracies of 1-50 ms

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

- 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

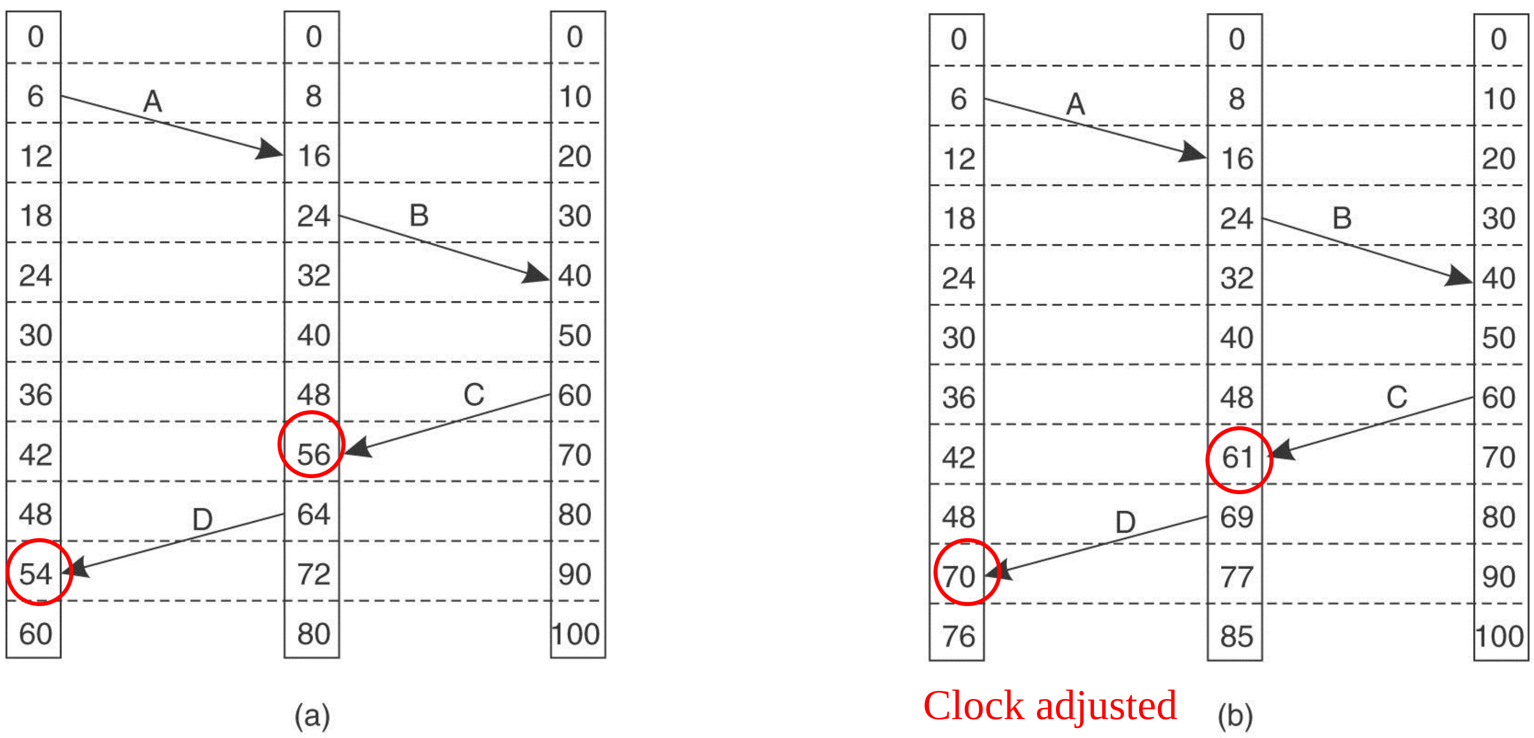
Happened Before Relation

- If *A* and *B* are events in the same process and *A* executed before *B*, then *A -> B*
- If A represents sending of a message and B is the receipt of this message, then A -> B
- Relation is transitive
  - If A -> B and B -> C, then A -> C
- Relation is undefined across processes that do not exchange messages
  - Partial ordering on events

Event Ordering Using *HB*

- Goal: define the notion of time of an event such that
  - If A-> B then C(A) < C(B)
  - If A and B are concurrent, then C(A) <, = or > C(B)
- Lamport algorithm:
  - Each processor maintains a logical clock LC_i
  - Whenever an event occurs locally at i, LC_i = ?
  - When *i* sends message to *j,* ?
  - When *j* receives message from *i*
  - Claim: this algorithm meets the above goals

Lamport's Logical Clocks



Election Algorithms

- Many distributed algorithms need one process to act as coordinator
  - Doesn’t matter which process does the job, just need to pick one
- Election algorithms: technique to pick a unique coordinator (aka *leader election*)
- Types of election algorithms: Bully and Ring algorithms

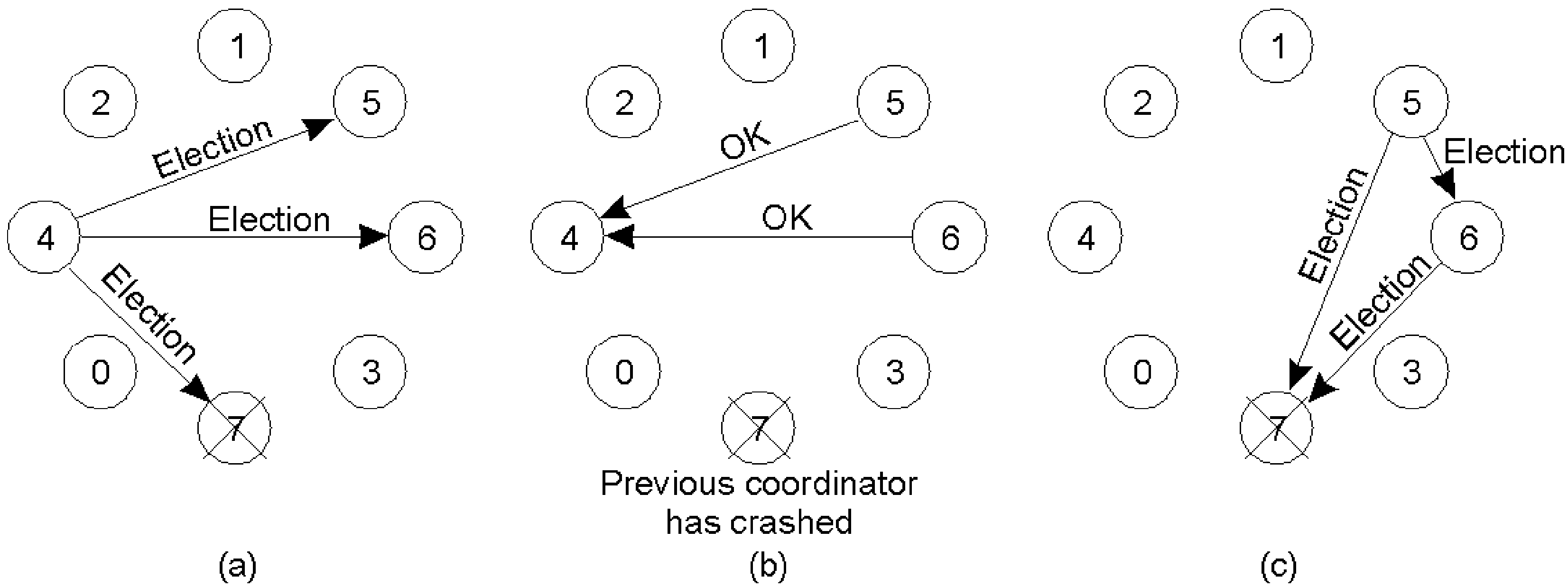
Bully Algorithm

- Assumptions:
  - Each proc has a unique ID
  - Proc know the IDs and address of every other procs
  - Communication is reliable
- Details:
  - Any process *P* can initiate an election
  - *P* sends *Election* messages to all process with higher IDs and awaits *OK* messages
  - If a process receives an *Election* msg from a lower-numbered colleague, ?
  - If a process receives a *Coordinator*, ?

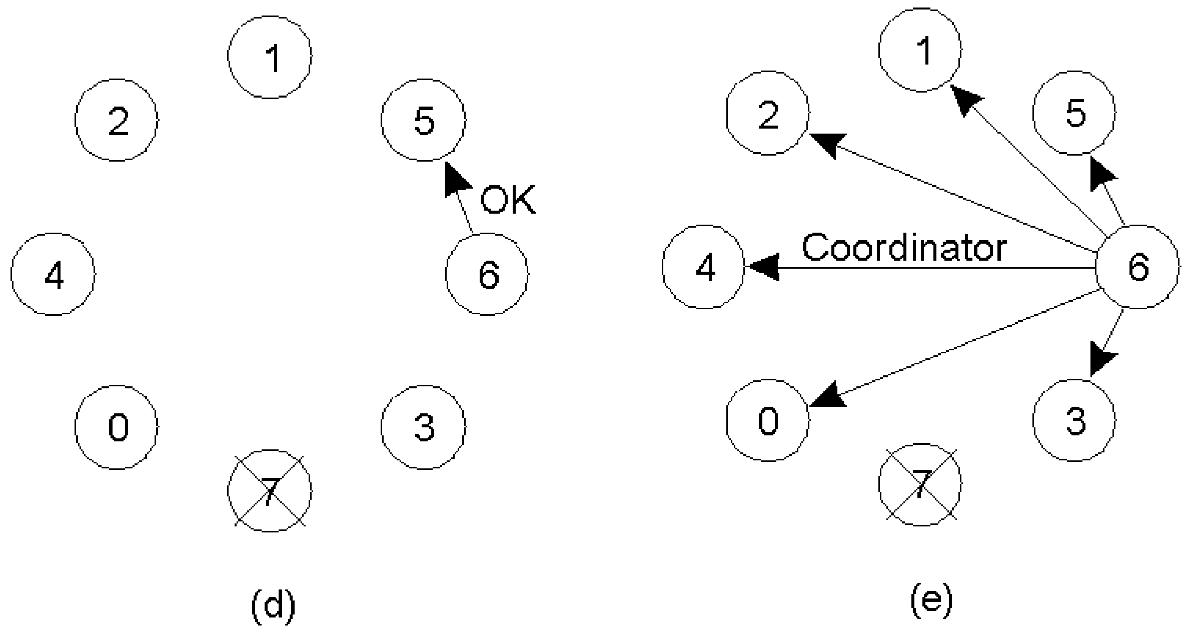
Bully Algorithm

- Process initiates election if it just recovered from failure or if coordinator failed
- Several processes can initiate an election simultaneously
  - Need consistent result
- ? messages required with *n* processes

Bully Algorithm Example



Bully Algorithm Example



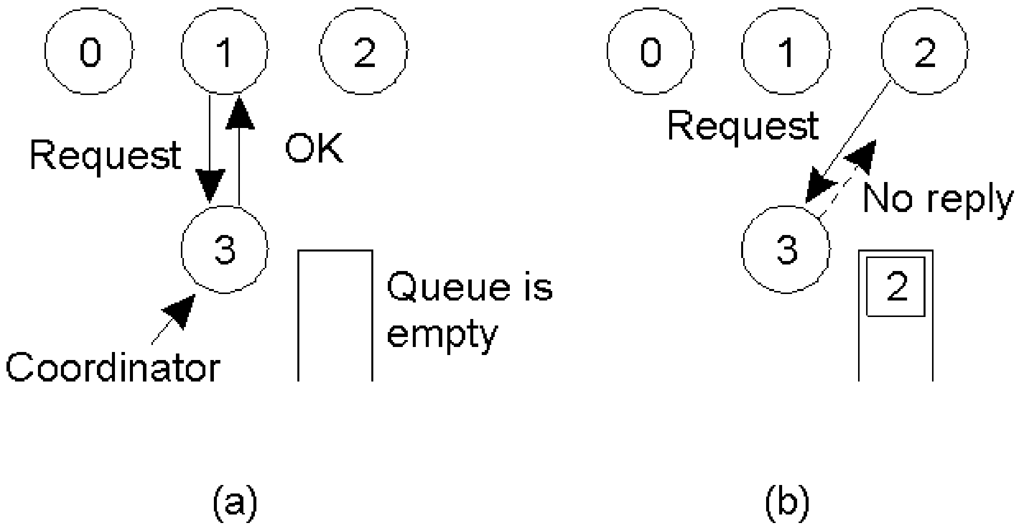
Distributed Mutual Exclusion

- Distributed system with multiple processes may need to share data or access shared data structures
  - Use critical sections with mutual exclusion
- Single process with multiple threads
  - Semaphores, locks, monitors
- How do you do this for multiple processes in a distributed system?
  - Processes may be running on different machines
- Solution: lock mechanism for a distributed environment
  - Can be centralized or distributed

Centralized Algorithm

- Assume processes are numbered
- One process is elected coordinator (highest ID process)
- Every process:
  - Needs to check with coordinator before entering the critical section
  - To obtain exclusive access:
  - To release:
- Coordinator:
  - Receive *request*:
  - Receive *release*:

Example: Centralized Algorithm



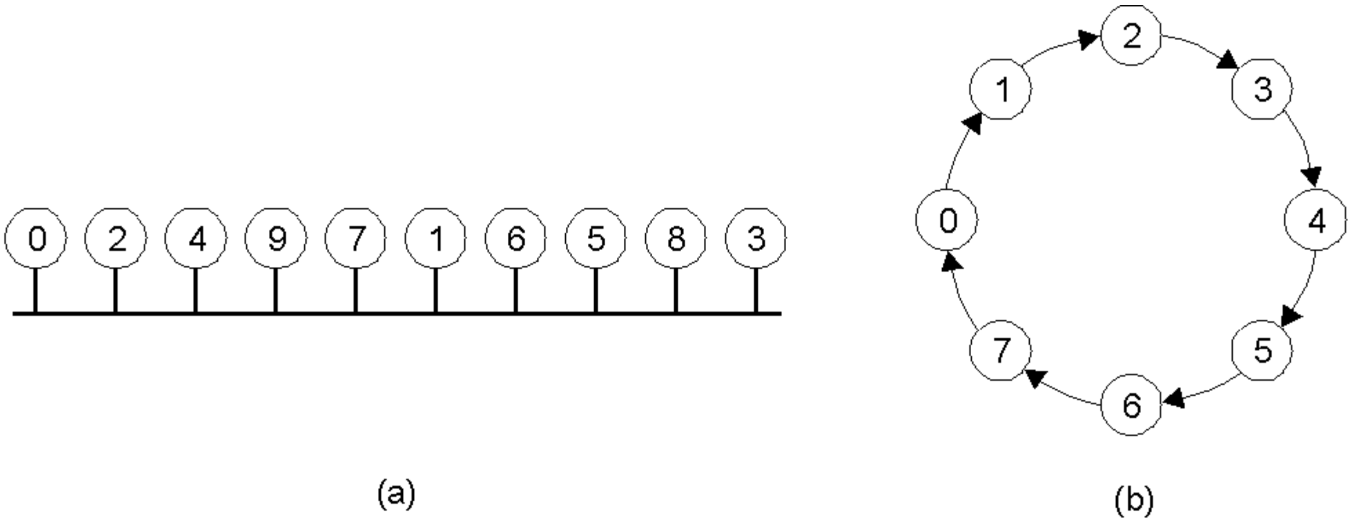
Centralized Algorithm: Comments

- Simulates centralized lock using blocking calls
- Fair: requests are granted the lock in the order they were received
- Simple: three msgs per use of a critical section (request, grant, release)
- Shortcomings:

Distributed Algorithm

- [Ricart and Agrawala]: Based on event ordering and time stamps

Token Ring Algorithm



- Use a token to arbitrate access to critical region
- Must wait for token before entering critical region
- Pass the token to neighbor once done or if not interested
- Con: ?

Comparison

Algorithm	**Messages per entry/exit**	**Delay before entry (in message times)**	**Problems**
Centralized	3	2	Coordinator crash
Distributed	2 (n – 1)	2 (n – 1)	Crash of any process
Token ring	1 to ∞	0 to n – 1	Lost token, process crash

Distributed Deadlocks

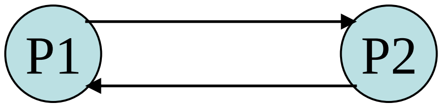
- Resource Deadlocks
  - A process needs multiple resources for an activity
  - Deadlock occurs if each process in a set request resources held by another process in the same set, and it must receive all the requested resources to move further
- Communication Deadlocks
  - Processes wait to communicate with other processes in a set
  - Each process in the set is waiting on another process's message, and no process in the set initiates a message until it receives a message for which it is waiting

Deadlock Handling Strategies

- Deadlock Prevention:
  - Difficult!
  - Before allocation, check for possible deadlocks
    - Difficult as it needs global state info
- Deadlock Detection:
  - Find cycles
  - Deadlock detection algorithms must satisfy 2 conditions:
    - No undetected deadlocks.
    - No false deadlocks.

Graph Models

- Graph models:
  - Nodes: processes
  - Edges of a graph: the pending requests or assignment of resources
- Wait-for Graphs (WFG): P1 -> P2 implies P1 is waiting for a resource from P2.
- Transaction-wait-for Graphs (TWF): WFG in databases.
- Deadlock: directed cycle in the graph.
- Cycle example:



Distributed Deadlocks

- Centralized Control
  - A *control node* constructs wait-for graphs (WFGs) and checks for directed cycles
  - WFG can be maintained continuously (or) built on-demand by requesting WFGs from individual node
- Distributed Control
  - WFG is spread over different nodes. Any node can initiate the deadlock detection process.
- Hierarchical Control
  - Nodes are arranged in a hierarchy.
  - A node checks for cycles only in descendents.

Summary

- Clock synchronization
  - Physical clocks
  - Synchronization algorithms
- Logical clock
  - Lamport timestamps
- Election algorithms
  - Bully algorithm
  - Ring algorithm
- Distributed mutual exclusion
  - Centralized algorithm
  - Distributed algorithm
  - Token ring algorithm
- Distributed deadlocks
- Readings:
  - Chpt 6 of AST

Questions

?