CompSci 131

Parallel and Distributed Systems

Prof. A. Veidenbaum

Today's topics

- **Mutual exclusion**

- **Reading assignment:**
  - **Today: 6.3**
  - **Next time: 6.4**

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

Last Lecture Covered

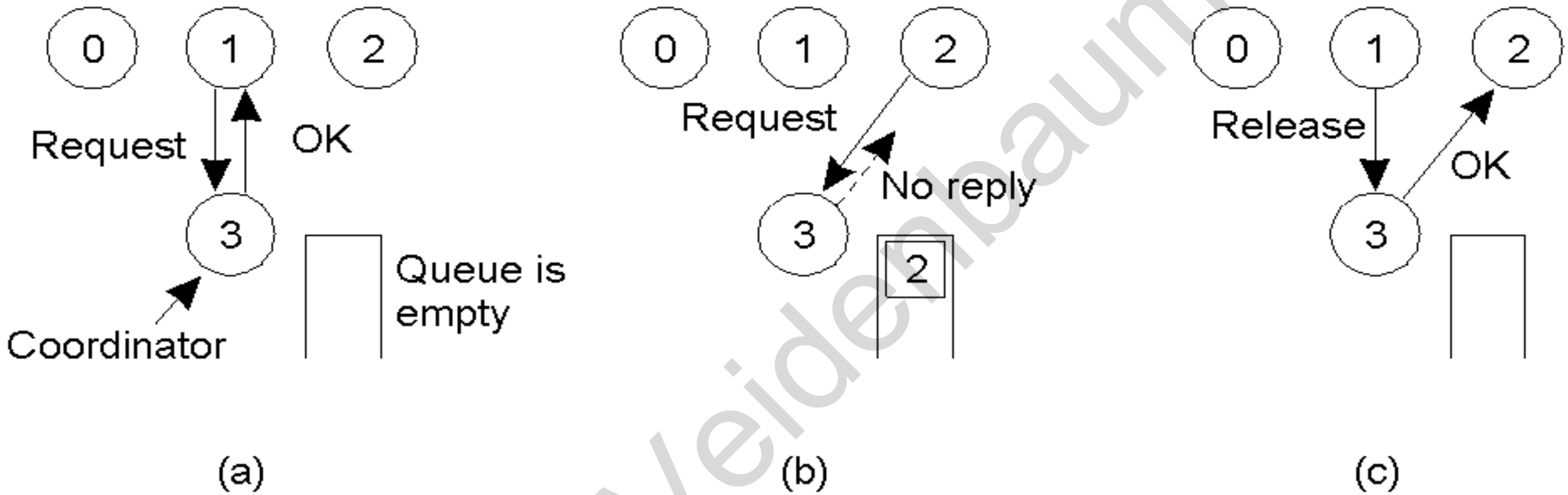
- **Logical Clocks**
- **Vector Clocks**

Mutual Exclusion *in DS*

- **Same goal: protect access to critical resources**
- **Used semaphores, etc in a single system**
  - **Harder in distributed systems**
    - **Because it happens on different systems**
      - **No common memory, clock, etc. etc.**
- **Algorithms for mutual exclusion in DS have to be**
  - **Fair, with no starvation**
  - **Deadlock-free**
  - **Fault-tolerant**

- **Two main approaches in distributed systems**
  - **Token-based – pass a token around in a deterministic way**
    - » **Fair**
    - » **What about fault tolerance?**
  - **Permission-based**
    - » **Get permission from all other processes/nodes**
      - **For instance, send a request and wait for all others to reply**
- **Let us first look a different approach though:**
  - **A centralized algorithm**
    - » **Have one node that can grant permission to a lock**
      - **Called a *coordinator***
      - **Now can use an SMP-like algorithm**
    - » **Issues:**
      1. **?**
      2. **?**

A Centralized Algorithm



a) Process 1 asks the coordinator for permission to enter a critical region
  – Permission is granted
b) Process 2 then asks permission to enter the same critical region.
  • The coordinator does not reply, but queues up the request
c) When process 1 exits the critical region, it tells the coordinator
  – Coordinator then replies to first in queue – 2 in this case

- **Queue holds state of CS requests**

Problems

- **Fault tolerance**
  - **What happens when the coordinator goes down?**
- **Scalability**

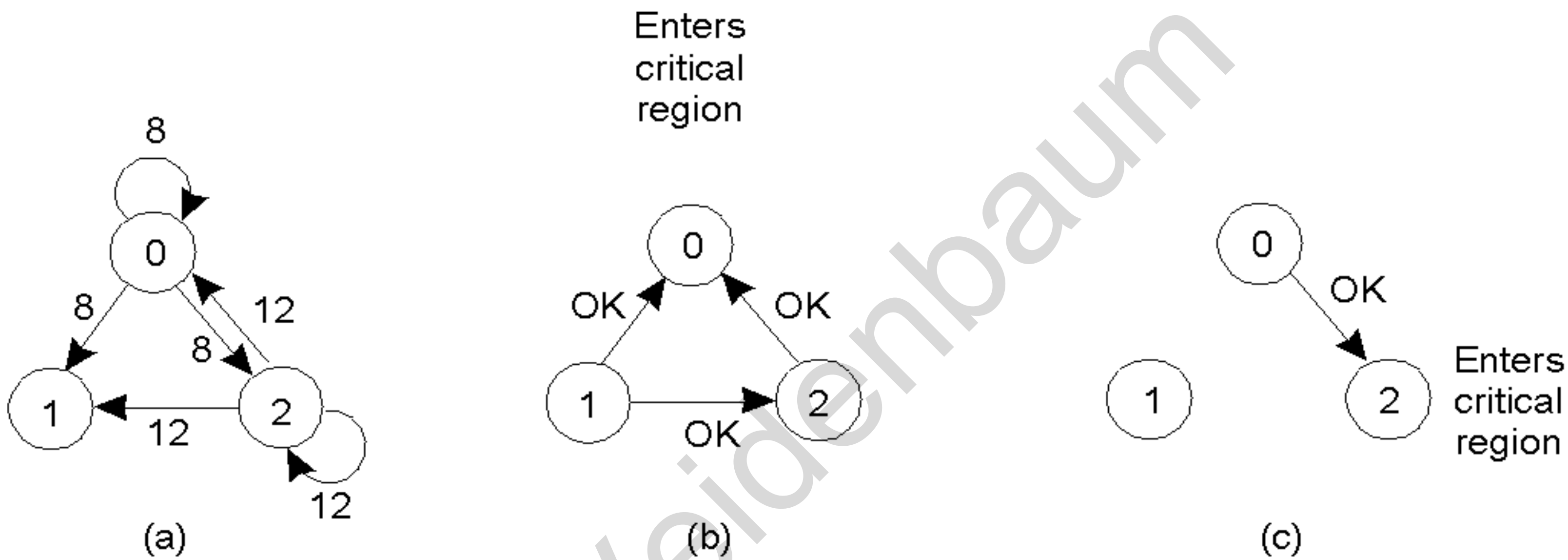
A Permission-based Distr. Algorithm

- A request to lock a region is sent to all with a time stamp and PID
  - Assumes reliable message delivery

- Three cases are possible
  1. A receiver is not accessing, nor wants to access the lock
     » It ACK's
  2. A receiver is currently holding the lock
     » It does not ACK and queues the request
  3. A receiver wants to access the lock concurrently
     » It is waiting for ACKs to its request
     » Uses timestamps
     » Process with the lowest timestamp wins
       - Lower pid, min(i,j), wins when $ts_i = ts_j$
     » The receiver does not reply if it wins

- When a process is done, it sends an ACK to those waiting.

Example



a) **Processes 0 and 2 try**

b) **A request to lock a region is sent to all with a time stamp and PID**

c) **Process 0 wins**

   » **does not reply to other requestors**

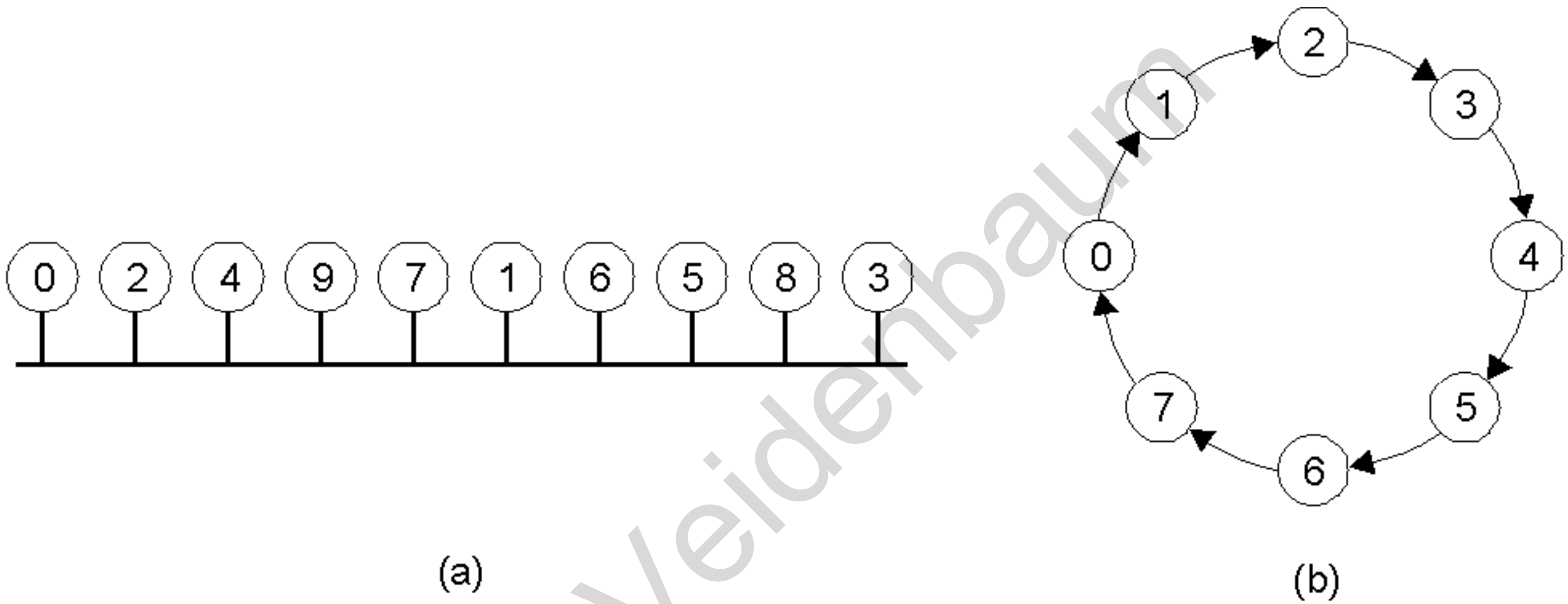
d) **When process 0 is done, it sends an OK, so 2 can now enter the critical region.**

- **No deadlock or starvation**

- **Reliability issue is even worse**
  - **Any process failing will not send an ACK**
    - » **One solution – do ACK and NACK, use timeouts**

- **What about scalability?**

A Token Ring Algorithm



a) An unordered group of processes on a network.
b) A logical ring is constructed in software
c) A single token is given to process 0 at initialization
   – Circulates around the ring
d) A process possessing the token can access resource

- **Can make token circulate fast if no action pending**

- **Correct and fair**

- **Fault-tolerance is hard**
  - **What happens if token is lost?**

A Decentralized Algorithm

Essentially extends the centralized algorithm

a) **Replicate a resource n times,**
   - each replica has its own local coordinator

b) **A process sends requests to all n coordinators**
   - *And* does not reply to other requestors

c) **The process can enter a critical region if it receives k>m/2 replies**
   - Otherwise it waits and tries again

- **Why is this correct?**
  - What does correct mean?

- **This approach is only probabilistically fault tolerant**
  - To single coordinator failure

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	**0**	**0 to O(n)**	**Lost token, process crash**