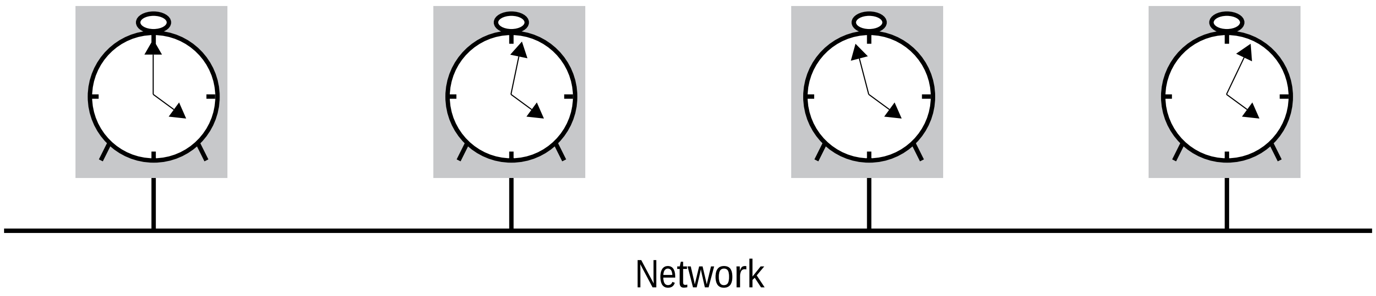
Time and Coordination in Distributed Systems

Distributed Software Systems

Clock Synchronization

- Physical clocks drift, therefore need for clock synchronization algorithms
  - Many algorithms depend upon clock synchronization
  - Clock synch. Algorithms – Christian, NTP, Berkeley algorithm, etc.
- However, since we cannot perfectly synchronize clocks across computers, we cannot use physical time to order events

Figure 10.1 Drift between computer clocks in a distributed system.



Logical time & clocks

- Lamport proposed using logical clocks based upon the "happened before" relation
  - If two events occur at the same process, then they occurred in the order observed
  - Whenever a message is sent between processes, the event of sending occurred before the event of receiving
  - X happened before Y denoted by X$\rightarrow$Y

Figure 10.5 Events occurring at three processes.

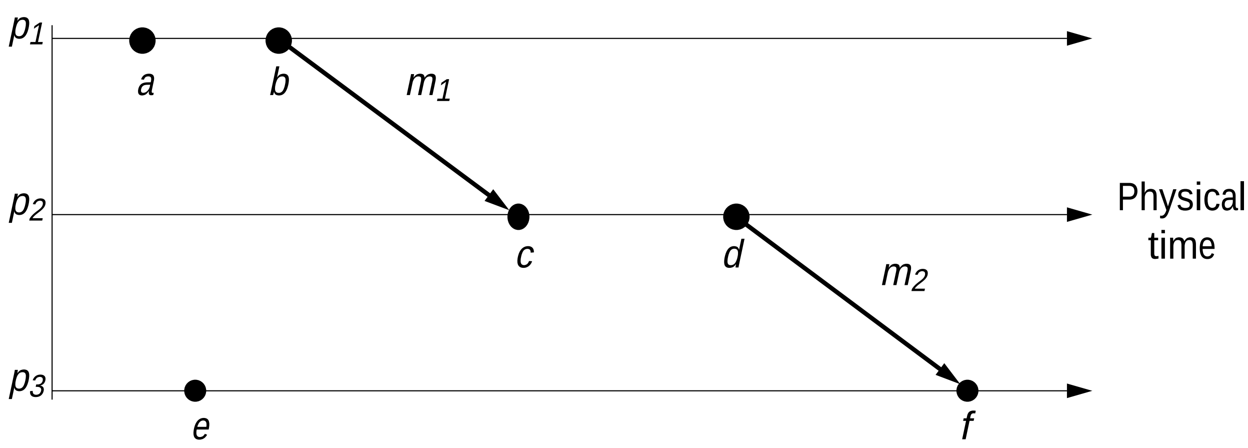
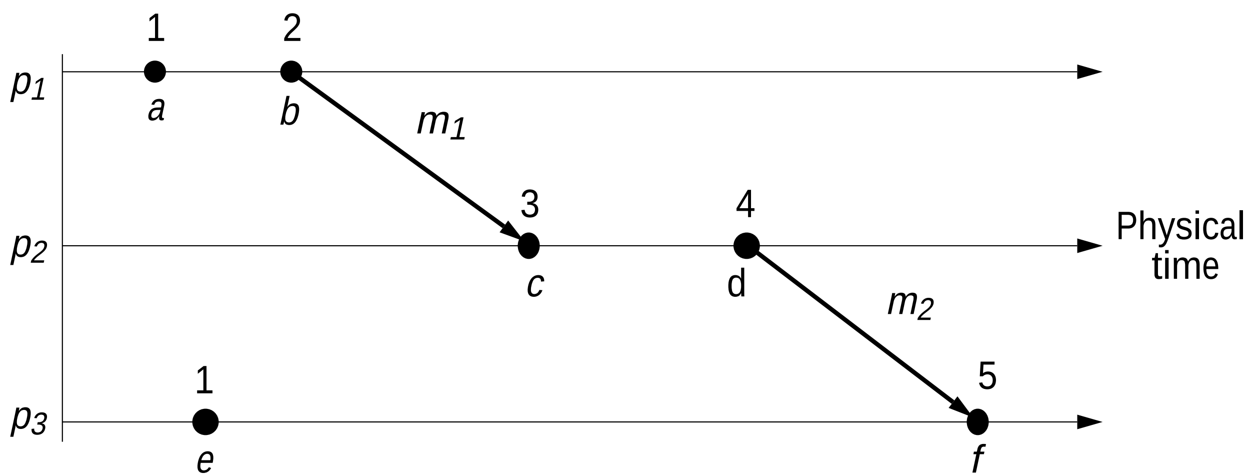


Figure 10.6 Logical timestamps for the events shown in Figure 10.5.



Lamport's algorithm

- Each process has its own logical clock
- LC1: C_p is incremented before each event at process p
- LC2:
  1. When process p sends a message it piggybacks on it the value C_p
  2. On receiving a message (m,t) a process q computes $C_q = \max(C_q, t)$ and then applies LC1 before timestamping the receive event

Totally ordered logical clocks

- Logical clocks only impose partial ordering
- For total order, use (T_a,P_a) where P_a is processor id
- $(T_a,P_a) < (T_b,P_b)$ if and only if either
  $T_a < T_b$ or ($T_a = T_b$ and $P_a < P_b$)

Distributed mutual exclusion

- Central server algorithm
- Ricart and Agrawal algorithm
  - A distributed algorithm that uses logical clocks
- Ring-based algorithms

NOTE: the above algorithms are not fault-tolerant and not very practical. However, they illustrate issues in the design of distributed algorithms

- Several other mutual exclusion algorithms have been proposed
  - We will discuss majority voting in the context of replicated data management

Figure 10.7 Server managing a mutual exclusion token for a set of processes.

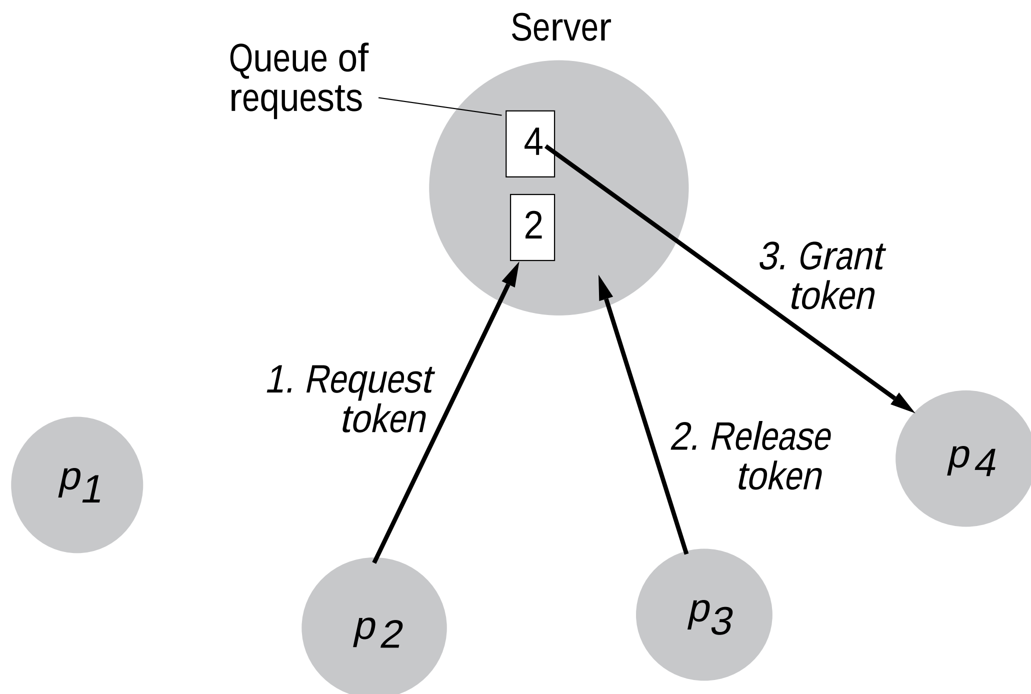


Figure 10.8 Ricart and Agrawala's algorithm.

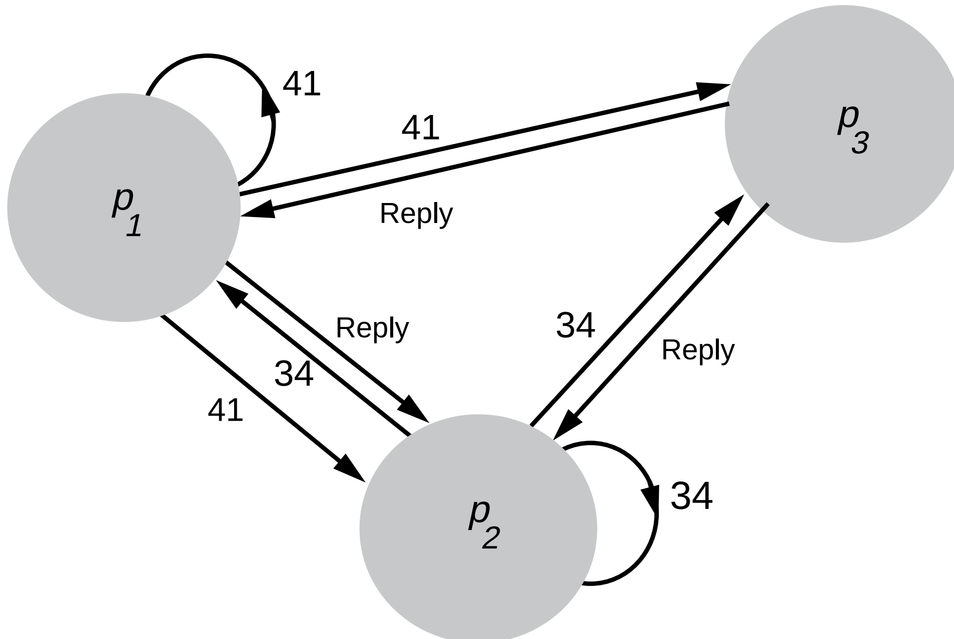
On initialization:
    state := RELEASED;

To obtain the token:
    state := WANTED;
    Multicast request to all processes;   } *Request processing deferred here*
    T := request's timestamp;
    Wait until (number of replies received = (n – 1));
    state := HELD;

On receipt of a request $<T_i, p_i>$ *at* p_j ($i \neq j$):
    if (*state* = HELD *or* (*state* = WANTED *and* (T, p_j) < (T_i, p_i)))
    then
            queue request from p_i without replying;
    else
            reply immediately to p_i;
    end if

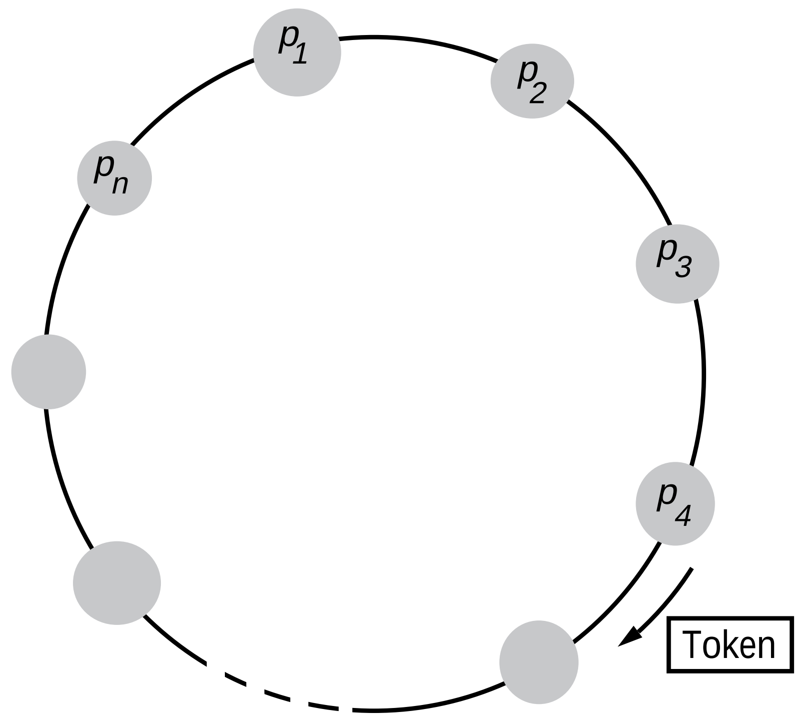
To release token:
    state := RELEASED;
    reply to any queued requests;

Figure 10.9 Multicast synchronization.



Processes request mutual exclusion by multicasting.

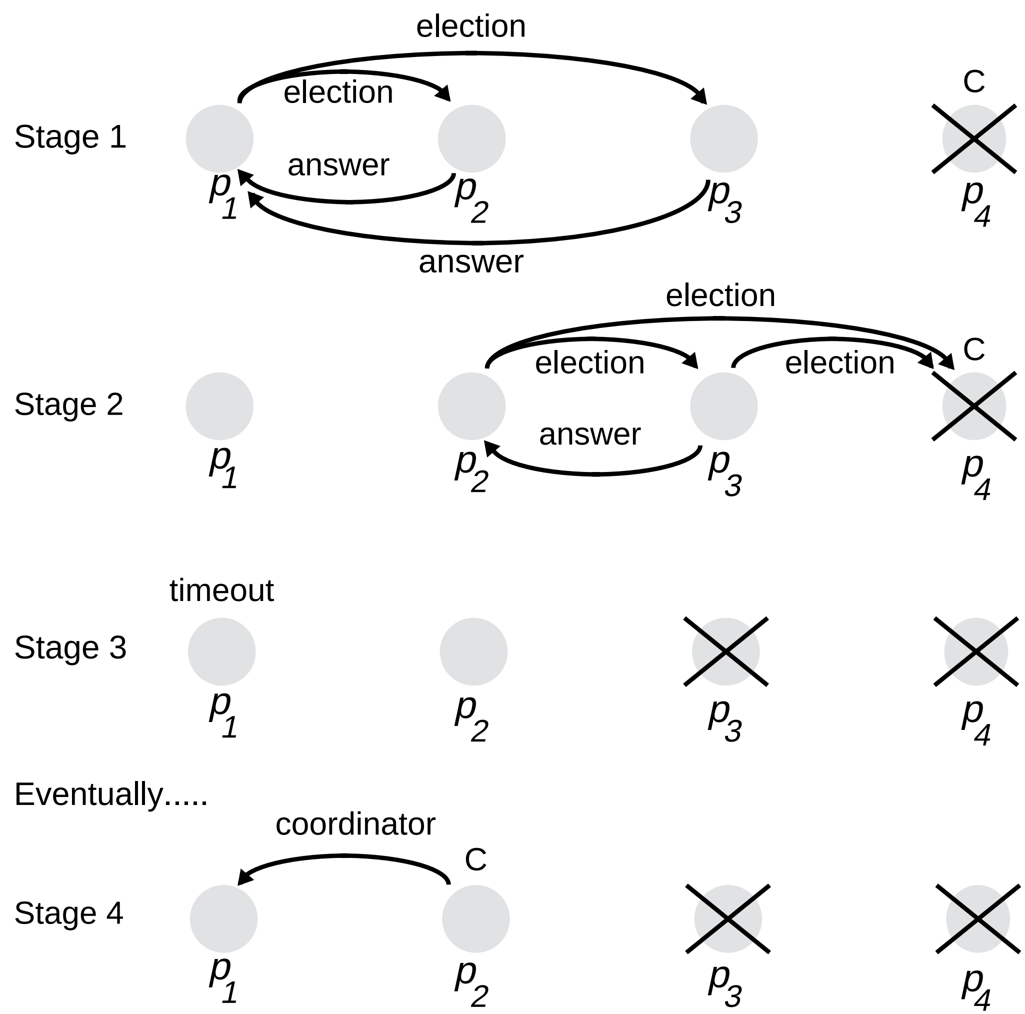
Figure 10.10 A ring of processes transferring a mutual exclusion token.



Election Algorithms

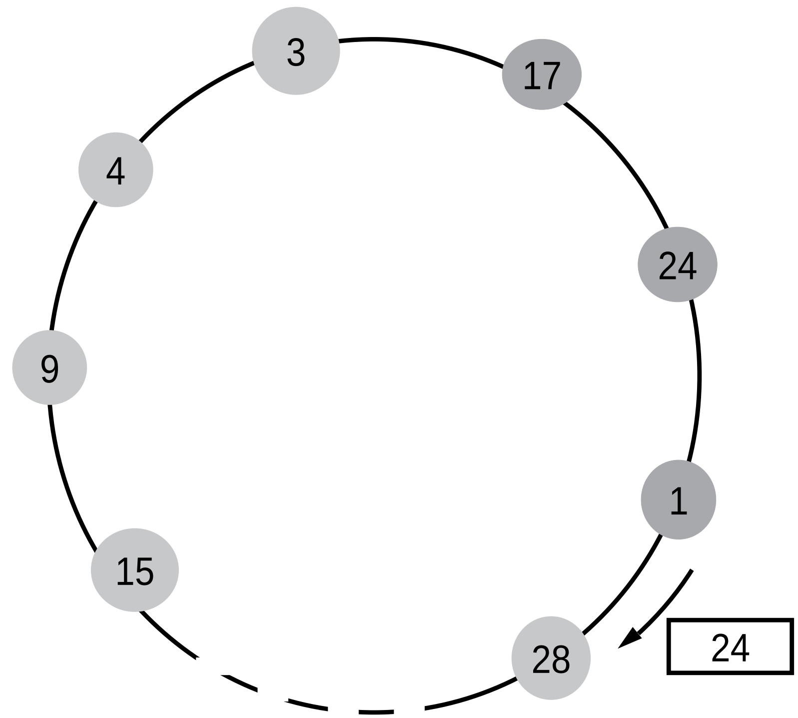
- An election is a procedure carried out to chose a process from a group, for example to take over the role of a process that has failed
- Main requirement: elected process should be unique even if several processes start an election simultaneously
- Algorithms:
  - Bully algorithm: assumes all processes know the identities and addresses of all the other processes
  - Ring-based election: processes need to know only addresses of their immediate neighbors

Figure 10.11 The bully algorithm.



The election of coordinator p_2, after the failure of p_4 and then p_3.

Figure 10.12 A ring-based election in progress.



Note: The election was started by process 17. The highest process identifier encountered so far is 24. Participant processes are shown darkened.