Coordination

Dr. Padraig Corcoran

- This section considers how processes can **synchronize** and **coordinate** their actions.
- **Process synchronization** ensures one process waits for another to complete its operation.
- **Data synchronization** ensures that two sets of data are the same.
- **Coordination** manages interactions and dependencies between activities in a distributed system.
- One could state that coordination encapsulates synchronization.

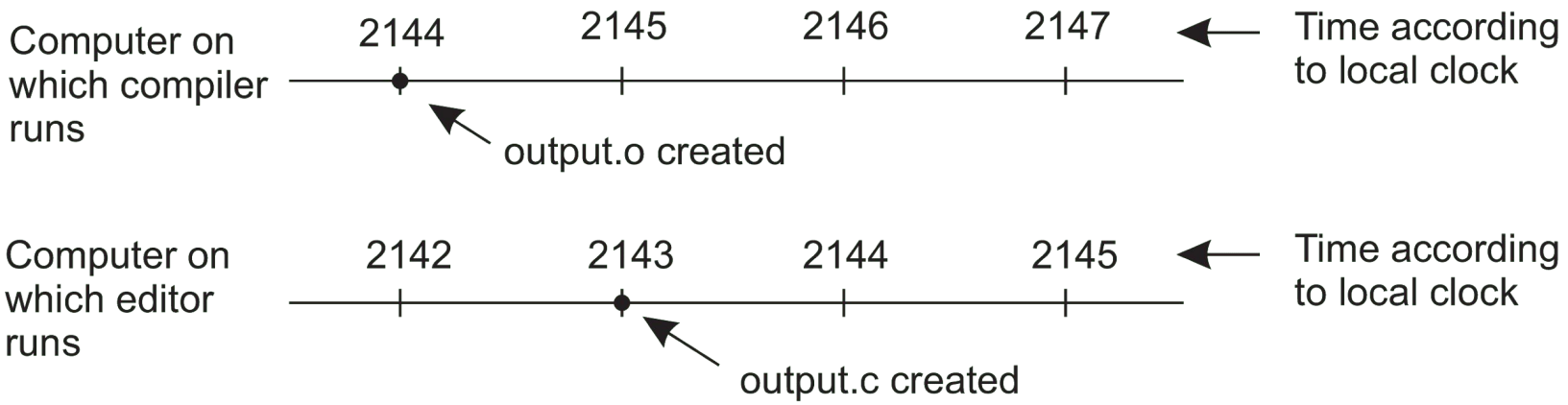
Clock synchronization

- The aim of clock synchronization is to ensure all processes have the same time.
- Clock synchronization provides a platform to perform process synchronization.
- In a centralized system, time is unambiguous:
  - Consider the case where process A asks for the time, and then a little later process B asks for the time.
  - The value that B gets will be higher than (or possibly equal to) the value A got.
- In a distributed system, achieving agreement on time is not trivial.

Unix make example

- Large programs are split up into multiple source files.
- A change to one file requires only that file to be recompiled, not all the files.
- *make* examines the times at which all source and object files were last modified.
- If source file *input.c* has time 2151 and the object file *input.o* has time 2150, *input.c* will be recompiled.
- If output.c has time 2144 and output.o has time 2145, no recompilation is performed.

- Consider a distributed system in which there is no global agreement on time.
- Suppose output.o has time 2144 and shortly after output.c is modified but is assigned time 2143 because the clock on its machine is slightly behind.



When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time.

Physical clocks

- Most computers have an internal clock which uses a *quartz crystal* oscillating at a specific frequency.
- With single computer/clock, since all processes use the same clock, it does not matter if the clock is incorrect.
- For example, if the file *input.c* has time 2151 and file *input.o* has time 2150, make will recompile *input.c*.
- This is the case even if the clock is off by 2 and the true times are 2153 and 2152, respectively.

- When a system has *n* computers, all clocks will run at slightly different rates.
- This will cause the clocks gradually to get out of sync.
- This difference in time values is called **clock drift**.
- Given a distributed system with multiple clocks:
  - How do we synchronize them with real-world clocks?
  - How do we synchronize the clocks with each other?

Universal Coordinated Time (UCT)

- UCT is the primary **time standard** by used to regulate clocks and time (can be considered a true time).
- Some 40 radio stations around the world broadcast a short pulse at the start of each UTC second.
- The accuracy of these stations is about ± 1 msec (due to atmospheric fluctuations, the accuracy is ± 10 msec).
- Earth satellites provide UTC accurately to 0.5 msec.

Process synchronization algorithms

- If one machine has a UTC receiver, the goal becomes keeping all the other machines synchronized to it.
- When the UTC time is *t*, denote by $C_p(t)$ the value of the clock on machine *p* at *t*.
- A set of clocks has **accuracy** within a bound α if the following equation is satisfied:

$$\forall t, \forall \mathrm{p} : |C_p(t) - t| \leq \alpha$$

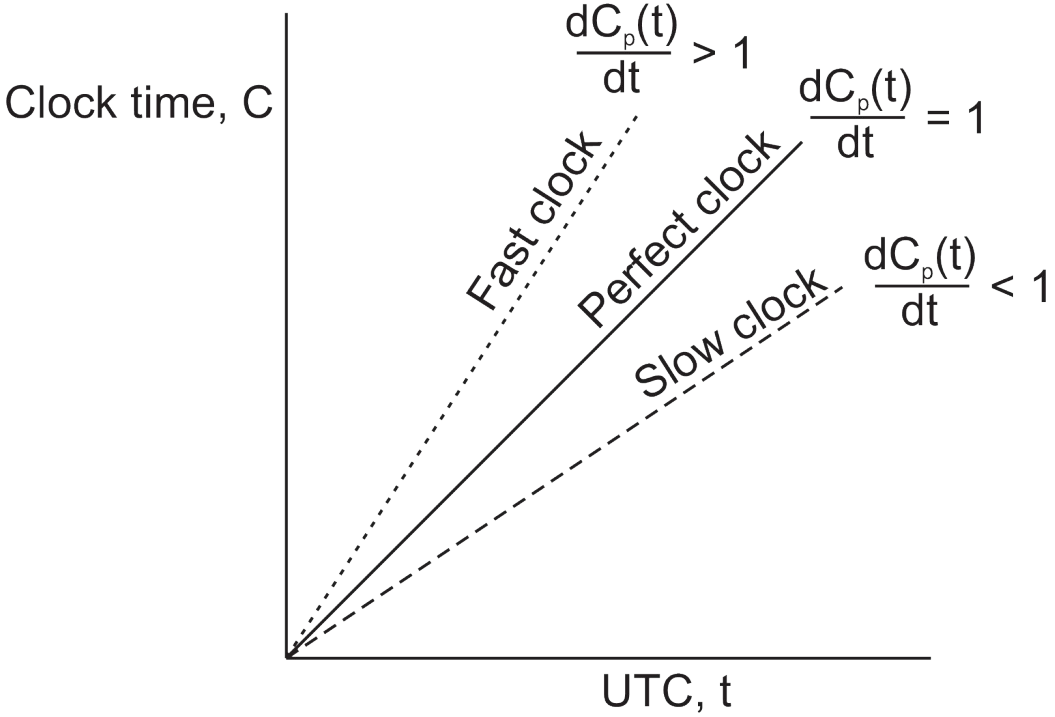
- The above equation refers to the deviation of clock times from UTC time.

- If no machine has a UTC receiver, each machine keeps track of its own time, and the goal is to keep all the machines together as well as possible.
- A set of clocks has **precision** within a bound π if the following equation is satisfied:

$$\forall t, \forall \mathrm{p}, \mathrm{q} : |C_p(t) - C_q(t)| \leq \pi$$

- The above equation refers to the deviation between clock times.

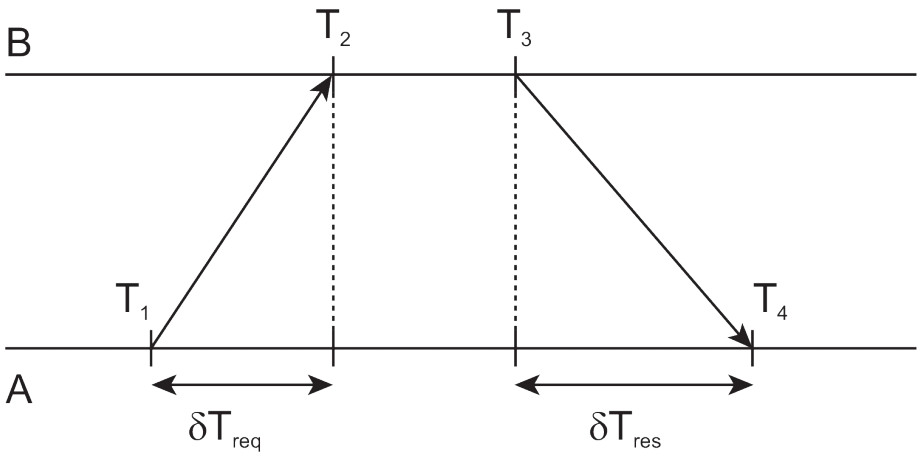
- In a perfect world, we would have $C_p(t) = t$ for all *p* and all *t*, and thus $\alpha = \pi = 0$.
- Clocks are subject to **clock drift** where clocks on different machines will gradually show different times.
- The **clock drift rate** refers to the difference per unit of time from a perfect reference clock.
- A typical quartz-based clock has a clock drift rate of approximately 31.5 seconds per year.



The relation between clock time and UTC when clocks tick at different rates.

Network Time Protocol

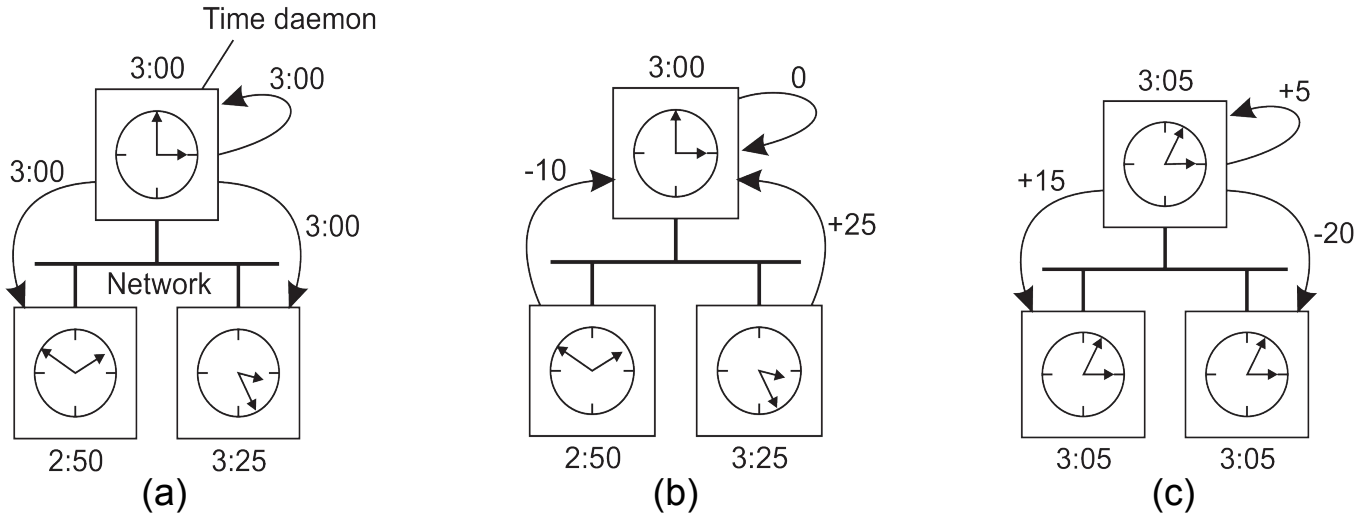
- A common solution to process synchronization is to let clients contact a **time server**.
- The latter can accurately provide the current time because it is equipped with a UTC receiver for example.
- However when contacting the server, message delays will have outdated the reported time.
- The trick is to find a good estimation for these delays.



Getting the current time from a time server.

The Berkeley algorithm

- The time server actively polls every machine from time to time to ask what time it is there.
- Based on the answers, it computes an average time.
- Tells all machines to advance/slow their clocks to this average time.



(a) The time daemon asks all the other machines for their clock values. (b) The machines answer. (c) The time daemon tells everyone how to adjust their clock.

Lamport's logical clock

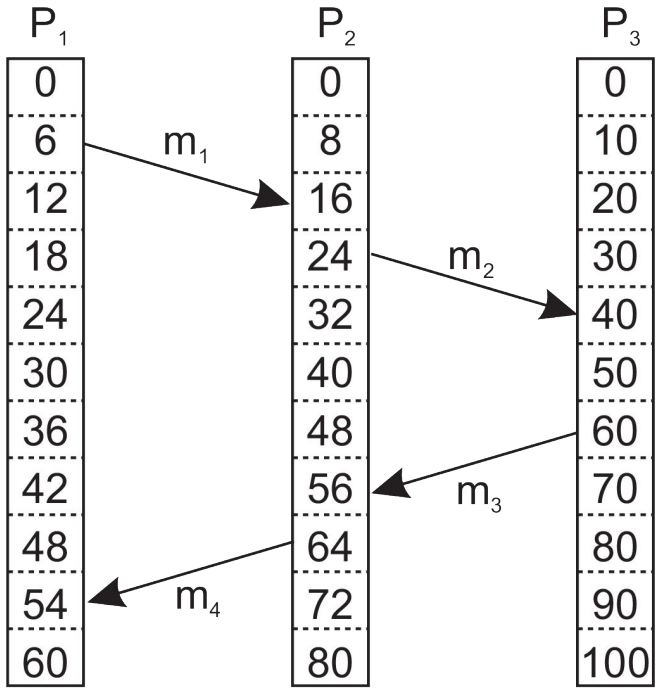
- In most cases it is not necessary all processes agree on exactly what time it is.
- Instead, they need only agree on the **order in which events occur**.
- In the *make* example, what matters is knowing if input.c was edited after input.o was compiled.
- A **logical clock** models the ordering of events.
- The first implementation, the **Lamport logical clock**, was proposed by Leslie Lamport in 1978.

Happens-before relation

- The expression $a \rightarrow b$ is read "event *a* **happens-before** event *b*".
- This means all processes agree that first event *a* occurs, then afterward, event *b* occurs.
- Happens-before can be observed in two situations:
  - If *a* and *b* are events in the same process, and *a* occurs before *b*, then $a \rightarrow b$ is true.
  - If *a* is the event of a message being sent by one process, and *b* is the event of the message being received by another process, then $a \rightarrow b$ is also true.

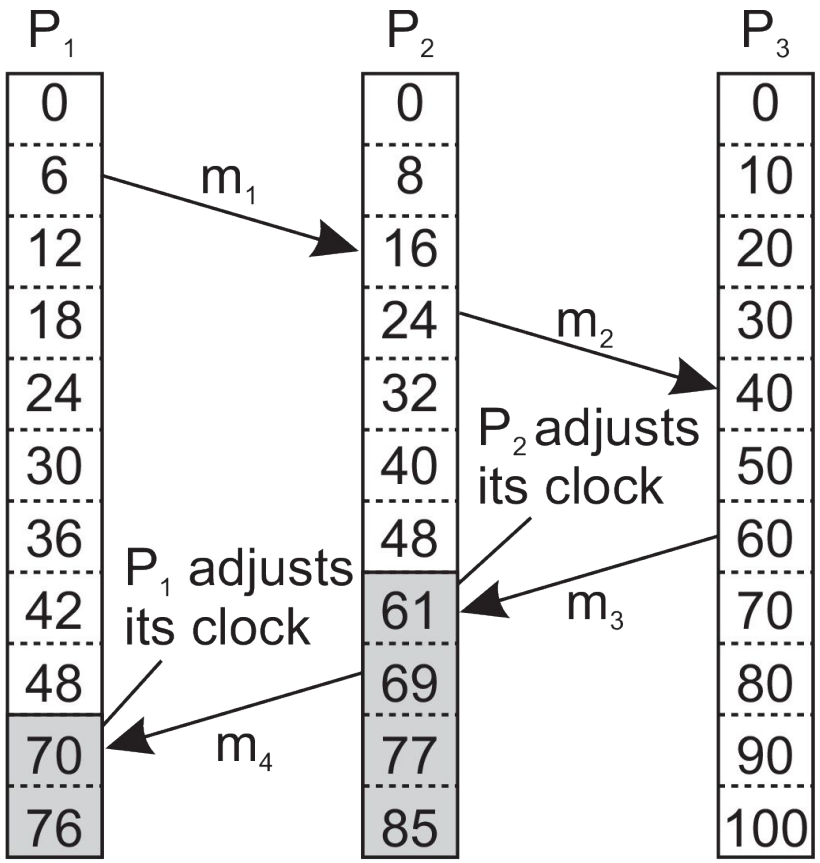
- The relation is transitive; if $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$.
- If events *x* and *y* happen in different processes that do not exchange messages (not even indirectly via third parties), then $x \rightarrow y$ is not true and is $y \rightarrow x$ not true.
- The events *x* and *y* are said to be **concurrent**; nothing can be said (or need be said) about event order.
- For events *a* and *b* where $a \rightarrow b$, Lamport's algorithm assigns **timestamps** *C(a)* and *C(b)* respectively such that $C(a) < C(b)$.

- Consider three processes on different machines.
- Clocks in process P_1, P_2 and P_3 are incremented by 6, 8 and 10 units per step.
- Message m_3 leaves process P_3 at 60 and arrives at P_2 at 56. These values are impossible.



Three processes, each with its own (logical) clock. The clocks run at different rates.

Lamport's algorithm



Three processes, each with its own (logical) clock. The clocks run at different rates. Lamport's algorithm corrects their values.

Each process P_i maintains a local counter C_i . These counters are updated according to the following 3 steps:

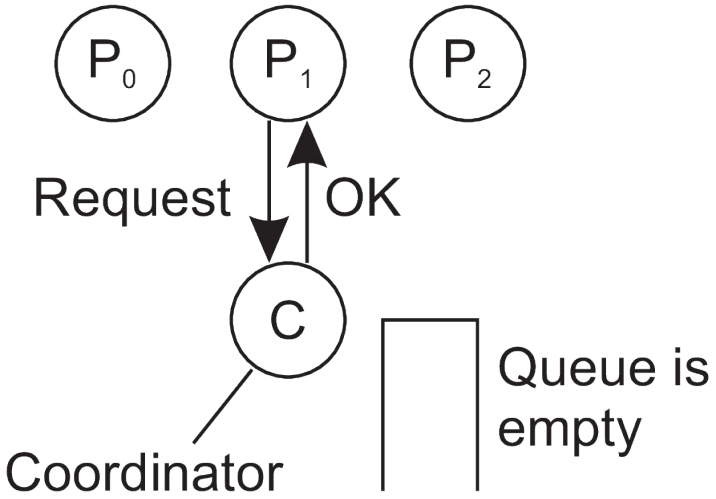
1. Before executing an event P_i increments C_i: $C_i \leftarrow C_i + 1$.
2. When process P_i sends a message m to process P_j , it sets m's timestamp ts(m) equal to C_i after having executed step 1.
3. Upon the receipt of message m, P_j adjusts its counter as $C_j \leftarrow \max \{C_j, ts(m)\}$ after which it executes step 1 and delivers the message to the application.

Mutual exclusion

- In many cases **different processes** will want to simultaneously **access the same resource**.
- Concurrent accesses may corrupt the resource or make it inconsistent.
- Solutions are needed to grant **mutual exclusive** access by processes.
- Algorithms for achieving mutual exclusion include **centralized coordinator** and **token-ring** algorithms.

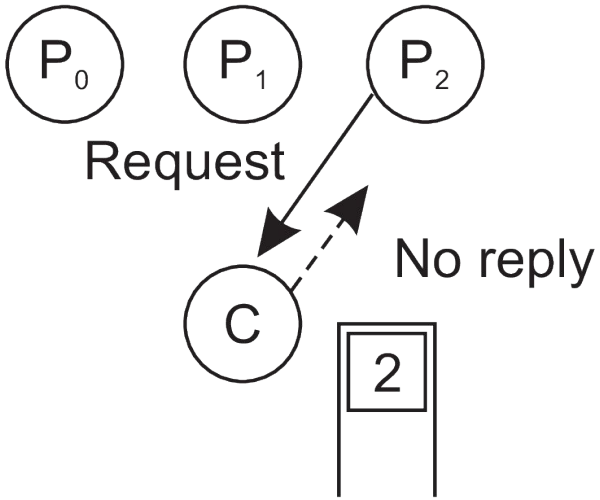
Centralized coordinator

- When a process P_1 wants to access a shared resource, it sends a request to the coordinator.
- If no other process is currently accessing that resource, the coordinator sends a reply granting permission.



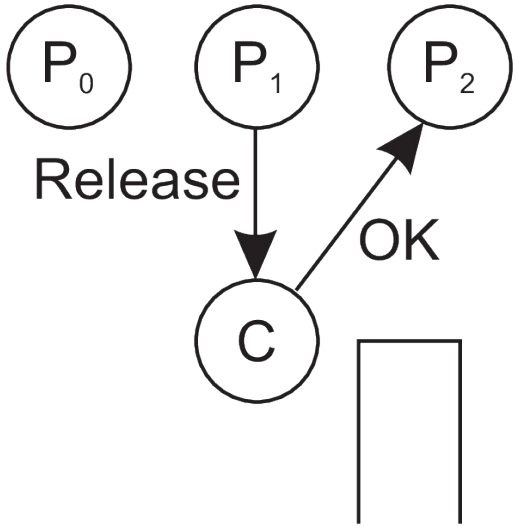
Process P_1 asks for permission to access a shared resource. Permission is granted.

- P_2 requests permission to access the resource.
- The coordinator knows that P_1 is already accessing the resource, so it cannot grant permission.
- It queues the request from P_2.



Process P_2 asks permission to access the same resource, but receives no reply.

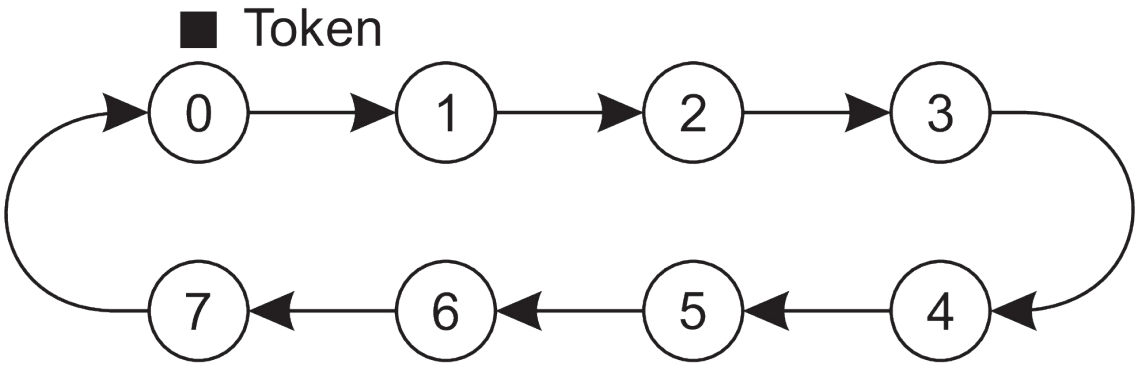
- When P_1 is finished with the resource, it sends a message to the coordinator releasing its access.
- Coordinator takes the first item off the queue of deferred requests and sends that process a grant message.



When P_1 releases the resource, the coordinator replies to P_2.

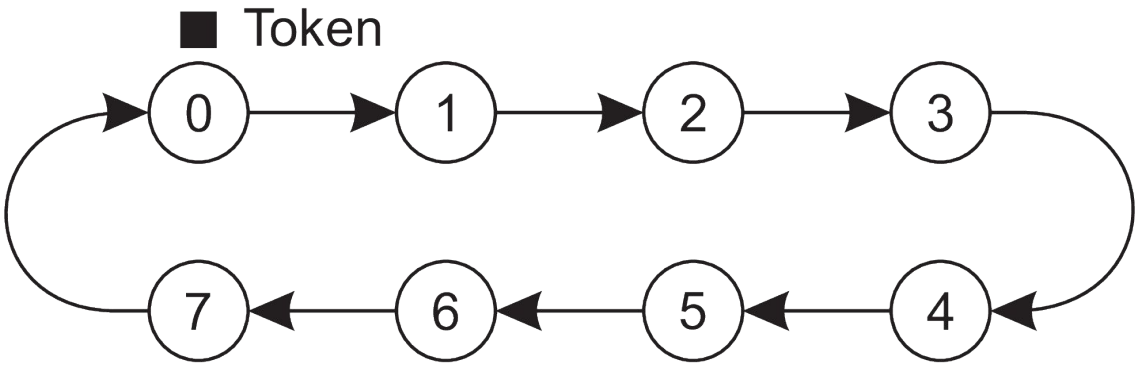
Token-ring algorithm

- Construct an **overlay network** in the form of a **ring** in which each process is assigned a position in the ring.
- All that matters is that each process knows who is next in line after itself.



An overlay network constructed as a logical ring with a token circulating between its members.

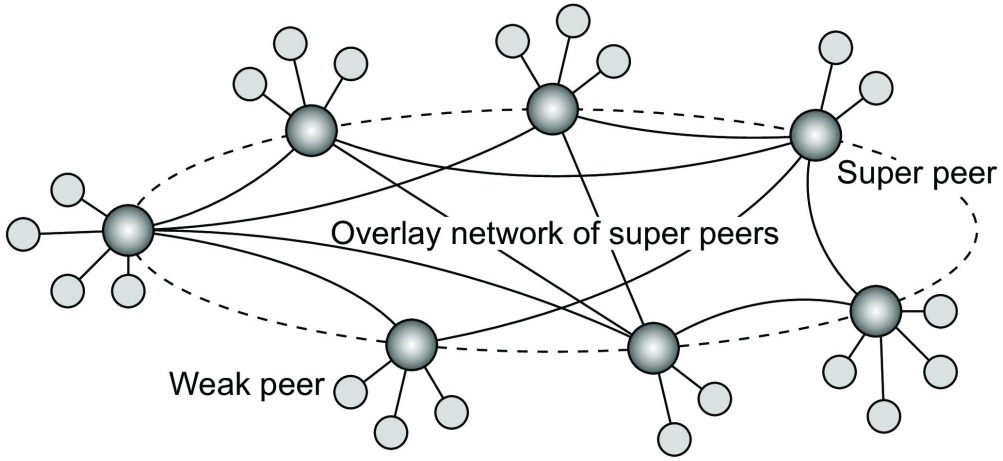
- When the ring is initialized, process P_0 is given a **token** which circulates around the ring.
- The token is required to access the shared resource.
- When a process acquires the token, it checks if it needs to access the shared resource.
- After it has finished, it passes the token along the ring.



An overlay network constructed as a logical ring with a token circulating between its members.

Election algorithms

- Many distributed algorithms require one process to act as coordinator or otherwise perform some special role.
- In general, it does not matter which process takes on this special responsibility.



A hierarchical organization of nodes into a super-peer network.

- Each process P has a unique identifier id(P).
- Election algorithms attempt to locate the process with the highest identifier and designate it as coordinator.
- The goal of an election algorithm is to ensure that all processes agree on who the new coordinator is.

Bully Election Algorithm

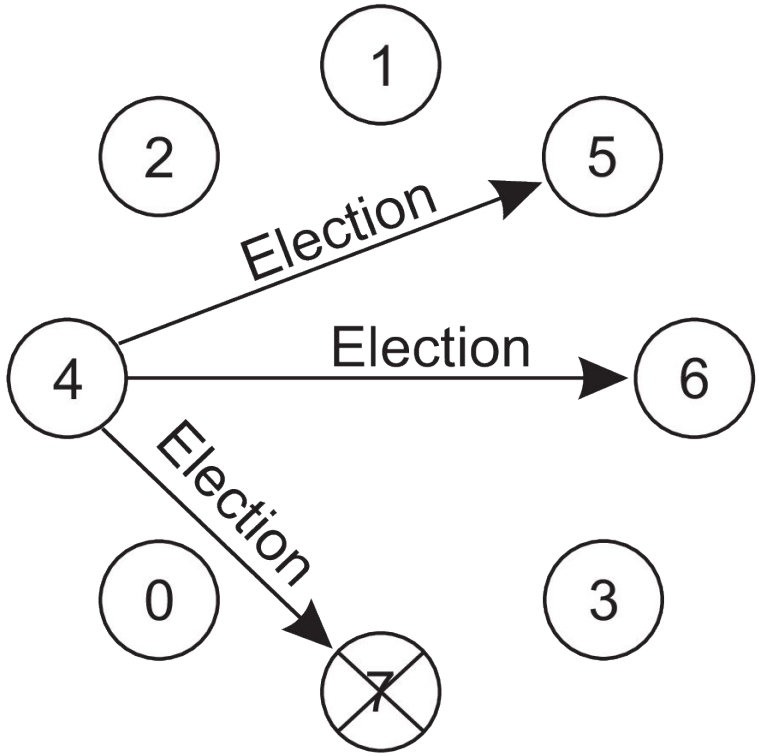
- Consider N processes $\{P_0, \ldots, P_{N-1}\}$ and let $id(P_k) = k$.
- When a process notices that the coordinator is no longer responding to requests, it initiates an election.

A process P_k holds an election as follows:

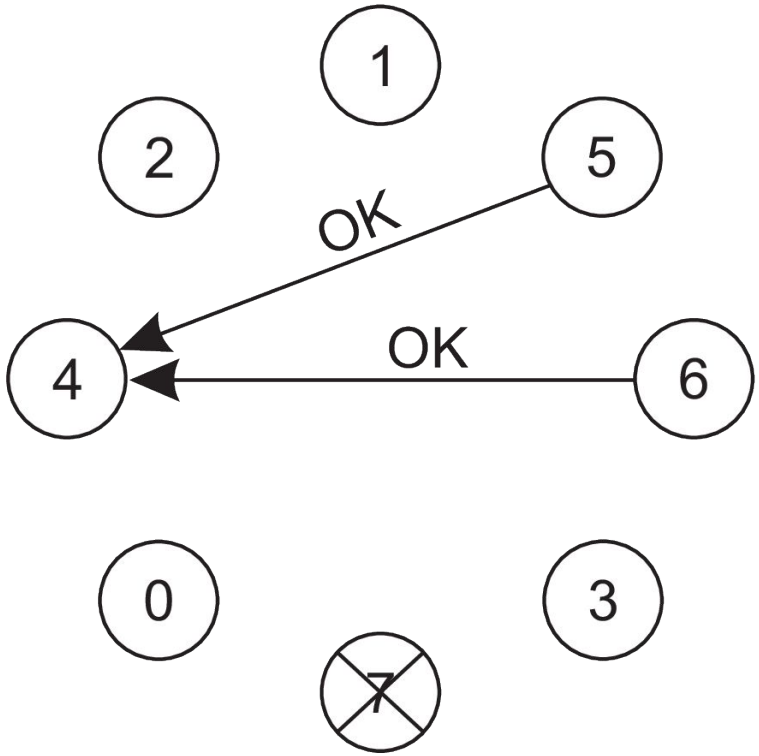
1. P_k sends an ELECTION message to all processes with higher identifiers: P_{k+1}, P_{k+2}, ..., P_{N-1}.
2. If no one responds, P_k wins the election and becomes coordinator.
3. If a process responds, it takes over (holds an election) and P_k's job is done.

- Eventually, all processes give up but one, and that one is the new coordinator.
- It announces its victory by sending all processes a message telling them it is the new coordinator.

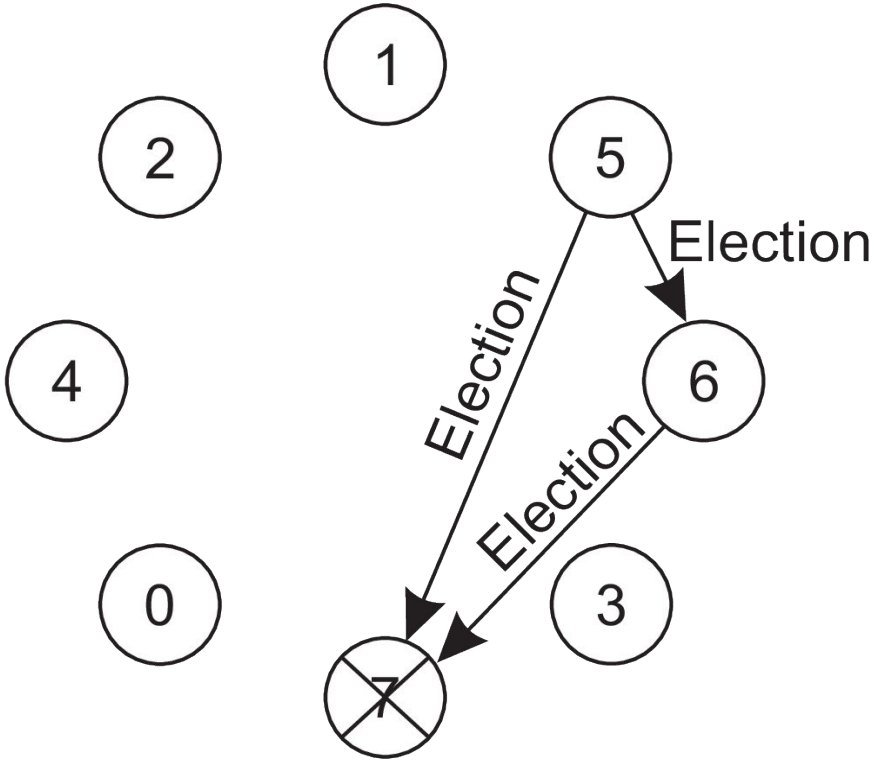
Bully Election Algorithm - Example



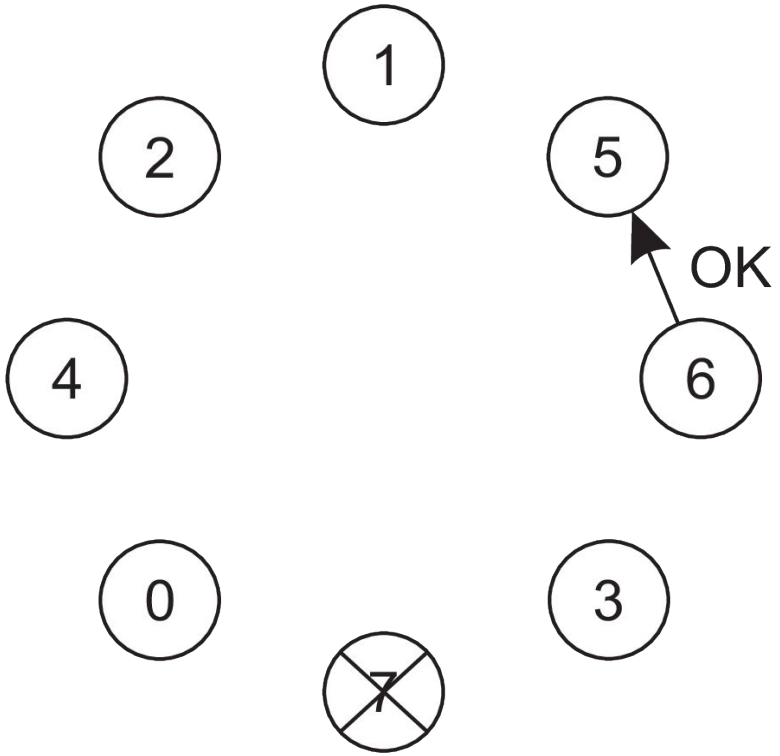
Previously process P_7 was the coordinator, but it has just crashed. Process P_4 is the first one to notice this, so it sends ELECTION messages to all the processes higher.



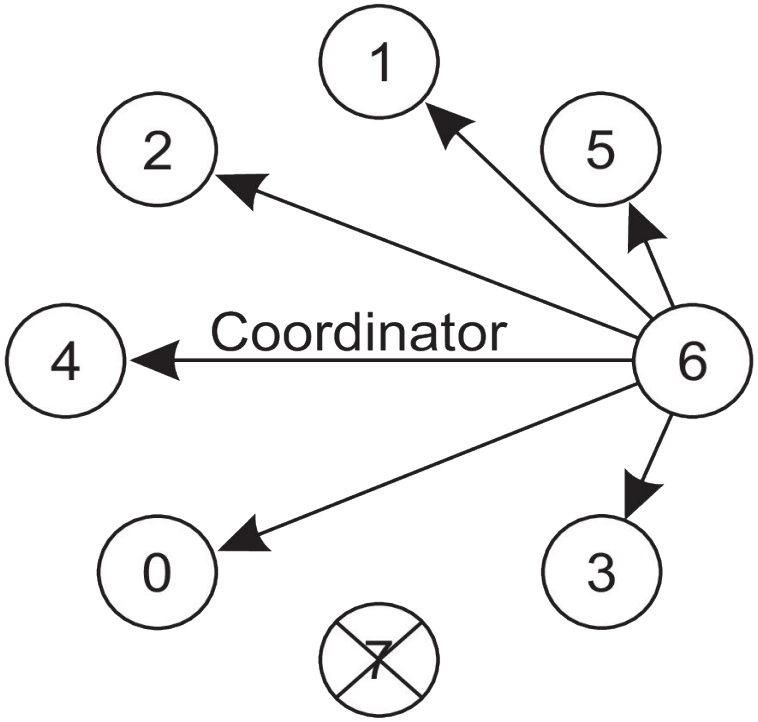
Processes P_5 and P_6 both respond with OK. Upon getting the first of these responses, P_4 knows that its job is done.



Both P_5 and P_6 hold elections, each one sending messages only to those processes with identifiers higher than itself.



P_6 tells P_5 that it will take over. At this point P_6 knows that P_7 is dead and that it (P_6) is the winner.

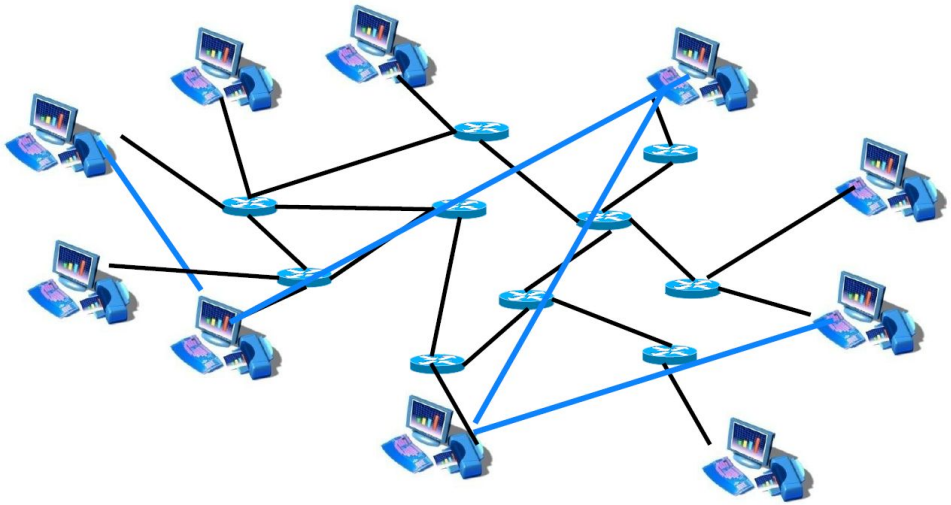


P_6 announces the takeover by sending a COORDINATOR message to all running processes.

- Traditional election algorithms are generally based on assumptions that are not realistic many environments.
- For example, they assume that message passing is reliable and that network topology does not change.
- These assumptions are false in most wireless environments, especially those for mobile ad hoc networks.

Location systems

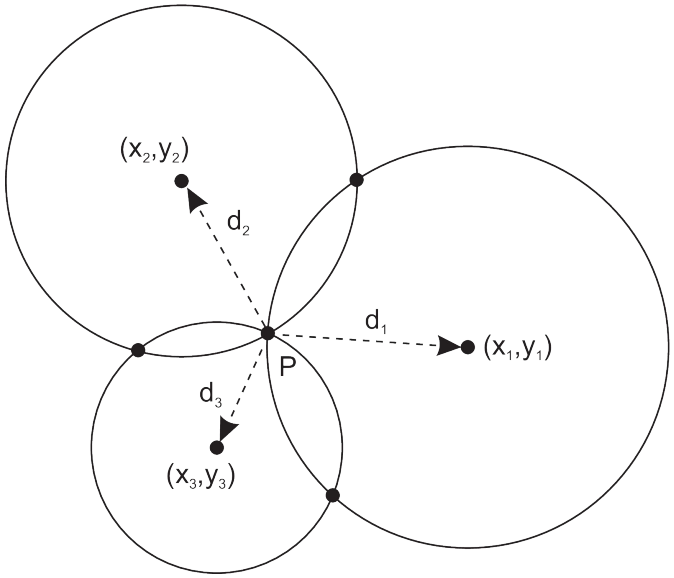
- For large distributed systems dispersed over a large area, it is often necessary to consider spatial location.
- Consider an overlay network in which two processes are neighbors, but are actually placed far apart.
- It may have been better to ensure that they are also physically placed close to each other.



GPS: Global Positioning System

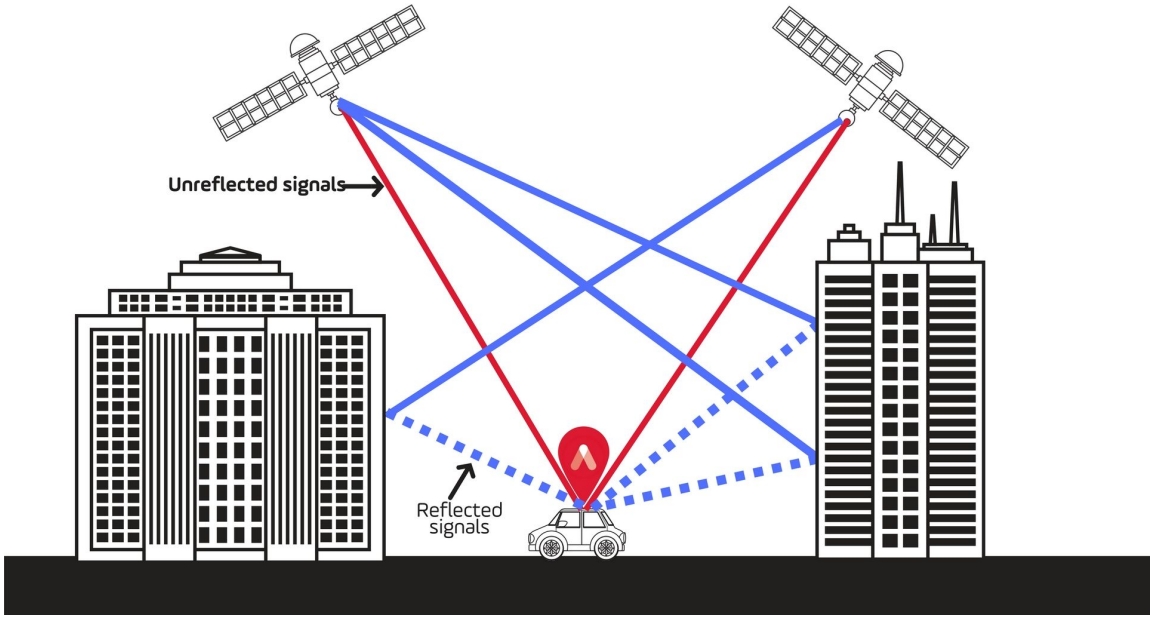
- GPS is a satellite-based distributed system that launched in 1978; initially used for military applications.
- GPS uses up to 72 satellites orbiting the earth.
- A satellite continuously broadcasts its position and time stamps each message with its local time.
- This allows a receiver on Earth to accurately compute its own position using, in principle, only four satellites.
- Let us first assume that all clocks, including the receiver's, are synchronized.

- Consider the two-dimensional case with three satellites.
- Circles represent points at the same distance from each respective satellite.
- The intersection of the three circles is a unique point corresponding to the location of P.



Computing a node's position in a two-dimensional space.

- This principle of intersecting circles can be extended to three dimensions.
- In this case we need to know the distances to four satellites to determine the location of a receiver.
- Determining the distance to a satellite becomes complicated when we move from theory to practice:
  - Clocks are not are perfectly synchronized.
  - Multipath error.
  - etc.



Multipath error

When GPS is not an option

- GPS cannot be used indoors.
- A popular technique is to make use of the WiFi access points available.
- If know the access points plus their coordinates we can estimate our distance to an access point.