TIME, SYNCHRONISATION and COORDINATION in DISTRIBUTED SYSTEMS

References [Col94], [Gos91], [Tan95]

TIME and its REPRESENTATION

- **Time**
- **Real-time**
- **Clock time**
- **Ideal clock**
- **Real Clock**

EVENTS, STATES and INTERVALS

- **Intervals in Real Time**
- **Intervals in Clock Time**
- **Events, States and Temporal Intervals**

EVENT ORDERING

- **Casual (or happened-before or partially) ordering**
- **Global (or total) ordering**
- **Sync (or casual-global) ordering**

CLOCKS in REAL TIME SYSTEMS

- **Physical Clock**
- **Logical clock (or Lamport's clock)**
- **Vector clock**

SYNCHRONISATION

CENTRALISED SYNCHRONISATION

- **Cristian's algorithm**
- **Berkeley algorithm**

DISTRIBUTED SYNCHRONISATION

- **Averaging algorithm**

CO-ORDINATION

CENTRALISED CO-ORDINATION

- **Central process or server**
- **Eventcounts**
- **Static sequencer**

DISTRIBUTED COORDINATION using TIME-BASED EVENT ORDERING

- **Multiple logical clocks**
- **Ring-based algorithm**

ELECTION

- **Bully algorithm for election**
- **Ring algorithm for election**

SYNCHRONISATION and COORDINATION in DISTRIBUTED SYSTEMS

- a DTOS should allocate resources to allow their effective utilisation, to ensure data consistency, lack of deadlock, and secure operations,

- a DTOS should provide partial and consistent view for all co-operating and competing processes,

SYNCHRONISATION = setting of clocks of computers in distributed systems to the same time value,

tasks of synchronisation:

1. to synchronise all the clocks to produce a single time standard

2. to eliminate effects of fast and slow clocks on local time

problems of synchronisation

- **no common clock or accurate global time source in the distributed system,**
- **time values may be different at different computers of the distributed systems,**
- **processes make decisions using only local information**

types of synchronisation:

- **active synchronisation of computer clocks**
- **passive synchronisation of computer clocks**

COORDINATION = temporal ordering of the set of events produced by the concurrent processes in time

tasks of co-ordination:

1. co-ordination in interpocess communication,

2. control of the joint activity of co-operating processes,

3. serialisation of concurrent access to shared processes by multiple processes,

problems of co-ordination:

- **centralised approaches cannot be used in distributed systems because of resource distribution, lack of global information and communication,**

types of co-ordination:

- **distributed approaches and algorithms:**

1. centralised co-ordination

2. distributed co-ordination using time-based event ordering

3. token based event-ordering

4. token passing approach

SYNCHRONISATION

CENTRALISED SYNCHRONISATION

Cristian's algorithm

- **centralised approach based on a time server which is a computer with a WWV server,**
- **the time server is passive, other computers ask it for time periodically and it replies their requests**

step No. 1

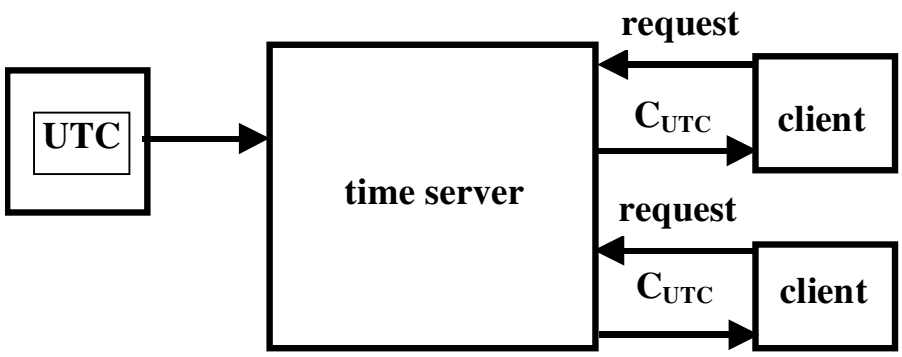
- **each machine sends a message to the time server asking it for the current time periodically at every δ/2ρ seconds**

step No. 2

- **the server machine responds as fast as it can with a message containing the current time C_{UTC},**

step No. 3

- **when the sender gets the reply, it sets its clock to C_{UTC},**



disadvantages of the algorithm:

1. if the sender's clock is, C_{UTC}, will be smaller than the the sender's current value of C

solution:

- **taking over C_{UTC}, could cause problems, thus such change must be introduced gradually**

2. it takes a non-zero amount of time for the time server's reply to get back to the sender and this delay may be large, and may vary with the communication network load

solution:

- **the algorithm measures the communication time between the sender and the time server by recording the time interval between sending the request to the time server T_{send} and arrival the reply T_{rep},**
- **when the reply arrives, value C_{UTC}, is increased by the half of the message communication time i.e. by $(T_{rep}, - T_{sen})/2$**
- **to further increase the accuracy of synchronisation time of interrupt handling and processing of the message is also may be taken into account**
- **in this case the half of the message communication and processing time will be equal to $(T_{rep}, - T_{sen} - T_{pro})/2$**

3. there is a single time server in the system thus, when the time server fails, the synchronisation will be stopped temporarily

solution:

- **a group of time servers replaces the single time server, where each time server has its own WWV server,**
- **a computer multicasts its request to all time servers but processes only the first reply**

Berkeley algorithm

- **centralised approach based on a time server, called time daemon, this approach is suitable even for a system that has no WWV server**

- **the time server is active that polls every computer periodically whose clocks are to be synchronised**

step No. 1

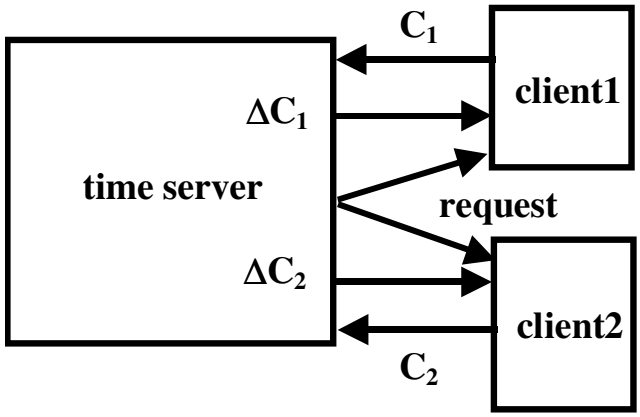
- **the time server sends a message, containing its time, and asks time of other computers**

step No. 2

- **computers reply with a message containing their time i.e. how far ahead or behind the time server they are**

step No. 3

- **having time values of computers, the time server calculates the average time and instruct each machine how to adjust its clock**



advantages of the algorithm:

- **the time server sends the amount of time to each computer by which its clock should be adjusted**

DISTRIBUTED SYNCHRONISATION

Averaging algorithm

- **decentralised approach that divides time into fixed-length synchronisation time slots *R*,**

step No. 1

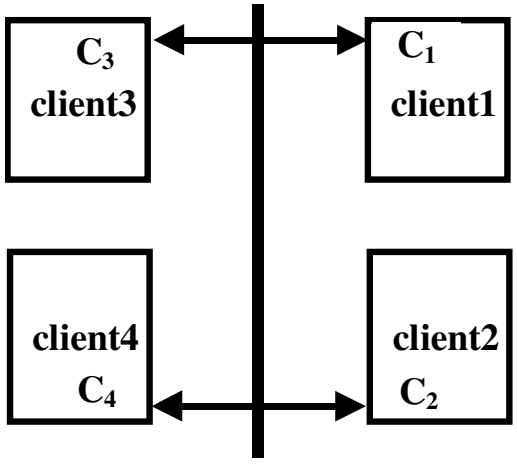
- **at the beginning of each interval, every machine broadcasts the current time according to its clock,**
- **these broadcasts occur at different time instants because local clocks run at different clock rate,**

step No. 2

- **after broadcasting its time, the computer starts to collect all other broadcasts during some interval**

step No. 3

- **having all clock values, the algorithm for example using averaging, computes a new time**



CO-ORDINATION

Co-ordination

- **is a temporal ordering of the set of events produced by the concurrent processes in time,**
- **determines a particular ordering on any given set of events in a distributed system,**
- **enforces a partial or a total ordering on any set of events**

types of distributed coordination:

1. centralised co-ordination

2. distributed co-ordination using time-based event ordering

3. token based event-ordering

4. token passing approach

mechanisms of distributed coordination:

1. mechanism based on mutual exclusion

- **defines a total ordering on the execution of concurrent processes**
- **objects of mechanisms: central process or token**

2. mechanisms not based on mutual exclusion

- **defines sequence of significant events during the computation**
- **objects of mechanisms: event counter or sequencer**

CENTRALISED CO-ORDINATION

- to co-ordinate distributed processes mutual exclusion is required to prevent interference and ensure consistency when accessing resources, which is the problem of the critical section,

- distributed mutual exclusion involves a single process being given a right to access shared resources before another process is granted it,

- there is a central co-ordination unit that has a unique name known to all processes that need to co-ordinate with each other, and at any time, any of these processes may access the central co-ordination unit,

- there is central or server node that

    a. controls access to shared objects,

    b. makes all decisions relating to co-ordination,

    c. stores information about all processes requesting resources,

- centralised co-ordination mechanisms:

    1. central server process

    2. event count

    3. static sequencer

    4. static sequencer

Central process or server

assumptions:

- **the central process (or co-ordinator) receives all requests from processes to be co-ordinated,**
- **the central process or server performs co-ordination via the mutual exclusion,**
- **if the central process fails, one of the processes in the system is chosen to co-ordinate processes**

algorithm:

step No. 1

- **a process sends a request message to the central process to invoke mutual exclusion,**

- **step No. 2**

- **the central process checks whether some other process is in the critical section, if not, it immediately sends a reply message, if yes, the request is queued**

step No. 3

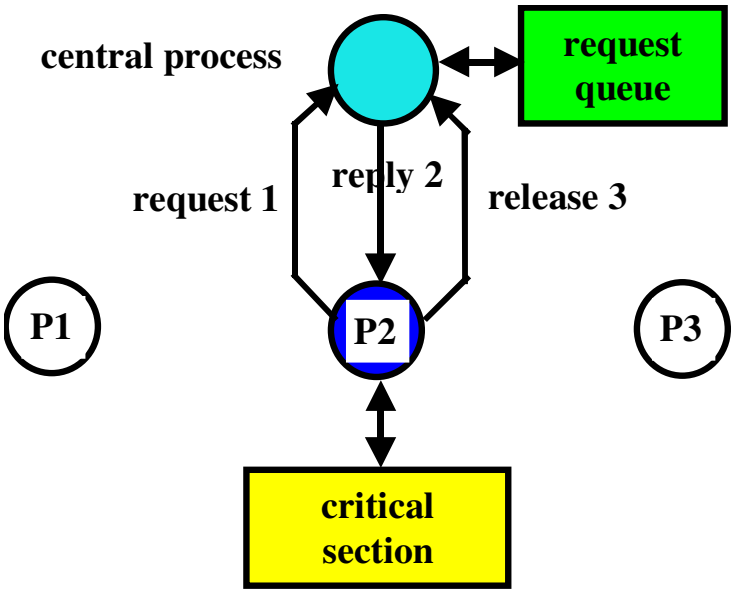
- **after receiving the reply message, the requesting process enters the critical section,**

step No. 4

- **after leaving the critical section, the requesting process sends a release message to the central process,**

step No. 5

- **after receiving the release message, the central process removes one of the request message from the queue,**



central process based co-ordination

properties:

- *safety*: **the reply message is given only to one process at a time by the central process**
- *liveness*: **central process maintains a queue of requests and allocates the reply message to the first process in the queue**
- *ordering*: **the queue of requests is ordered according to the happened-before relation**

advantages:

- **if the scheduling is fair no starvation can occur,**
- **the mechanism ensures the mutual exclusion,**

disadvantages:

- **entering and leaving the critical section requires three messages,**
- **the central process might become a performance bottleneck,**
- **the central process is also a critical point of failure. (If it fails, an election must be called to choose one of the clients to create or act as a server.)**

Eventcounts

assumptions:

- **- eventcount is an abstract object which allows processes to control the ordering of events directly by keeping a count of the number of events in a particular class,**

primitives:

- **execution of primitives constitutes events whose time ordering can be defined as a partial-order relation on events,**
- **the process that increases the eventcount is the signaller, and the process that reads the event count is the observer, they are at different nodes and each of them has its own copy of the event count,**

1. ***advance(E)***

- ♦ **signals the occurrence of an event associated with a particular evencount**
- ♦ **increases the eventcount by one thus, the value of eventcount equal to number of advance operations**

2. ***read*****(*****E*****)**

- ♦ **returns the current value of the eventcount**

3. ***await(E,v)***

- ♦ **suspends the calling process until the value of the eventcount is at least *v,* i.e. suspends the process until some event has happened**

rules of the algorithm: (Reed & Kanodia's algorithm)

1.-3. rules of happened-before ordering

- **Reed and Kanodia added the concurrent relation, denoted by ||, to relations defined by Lamport in order to handle eventcount primitives because they may be concurrent on the same eventcount**

4. eventcount is forwarded by the *advance(E)* primitive to reach the value given by the *await(E,v)* primitive

if W_E is the execution of an await operation *await(E,v)*,

then there are at least *v* members of the

$$\text{set } \{A_E | A_E \text{ - execution of } advance(E) \text{ and } A_E \parallel W_E\}$$

5. value obtained by the *read(E)* primitive is given by *advance(E)* primitive

if R_E is the execution of a read operation *v*:=*read*(*E*),

then there are at most *v* members of the

$$\text{set } \{A_E | A_E \text{ - execution of } advance(E) \text{ and } A_E \rightarrow R_E\}$$

and there are at least *v* members of the

$$\text{set } \{A_E | A_E \text{ - execution of } advance(E) \text{ and } A_E \parallel R_E\}$$

algorithm:

step No. 1

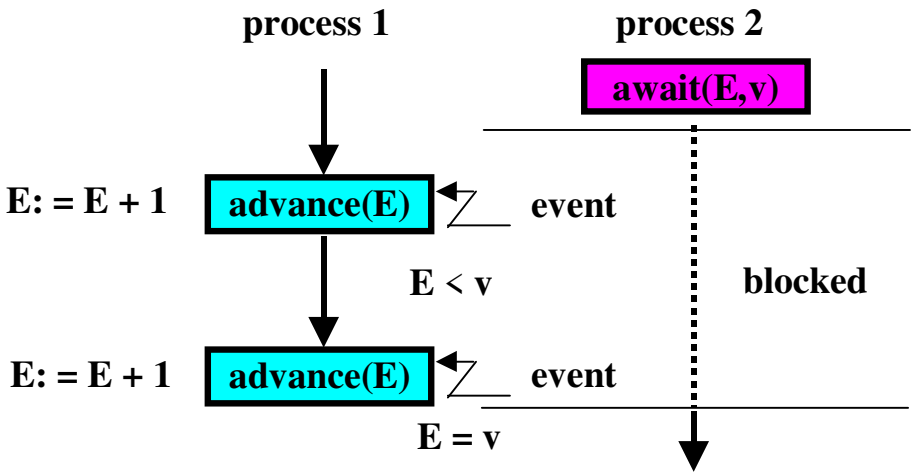
- **a caller is suspended by an await primitive until a corresponding event has happened setting its eventcount *E* to a value *v*,**

step No. 2

- **the advance primitive increases the eventcount *E* by one after each occurrence of the corresponding event,**

step No. 3

- **the caller continues its operation after await primitive is completed, i.e. the eventcount reaches the value given by the await primitive,**



eventcount based co-ordination

advantages:

- **eventcount can co-ordinate concurrent events,**
- **it is possible to construct general eventcount using local eventcounts advanced by only one process,**

disadvantages:

- **eventcount cannot discriminate between two events that happen in an uncontrolled order**

Static sequencer

assumptions:

- **a sequencer is a non-decreasing integer with initial value zero that defines a partial order of events,**
- **sequencer requires a separate exclusion mechanism for ensuring that its primitive is atomic,**

primitives:

1. ***ticket(S)***

- **returns the correct value of the sequencer *S* and then increases the sequencer by one,**

rules of the algorithm:

1. if *a* and *b* are events corresponding to a ticket operation on the same sequencer *S*,

then either $a \rightarrow b$ or $b \rightarrow a$

2. if *a* is an execution of *t:=ticket(S),*

then the value assigned to *t* is the number of elements of the

set $\{X | X$ is execution of a ticket operation on S and $X \rightarrow a\}$

algorithm:

step No. 1

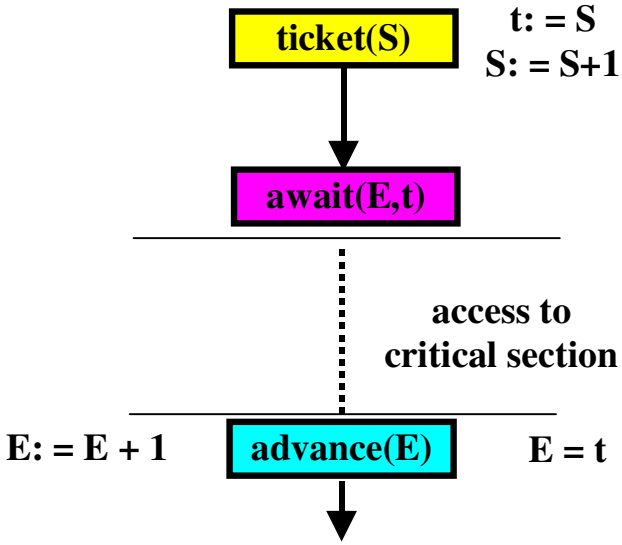
- **ticket primitive reads and returns the value of the sequencer *S* and then increments its value by one,**

step No. 2

- **await primitive sets the eventcount *E* equal to the value of the sequencer *S* obtained by the ticket operation,**

step No. 3

- **advance primitive increases the eventcount *E*,**



static sequencer based co-ordination

advantages:

- **provides arbitration, i.e. it decides on ordering between events that may otherwise be unordered,**

disadvantages:

- **requires separate mutual exclusion mechanism to ensure that the ticket primitive is atomic,**

DISTRIBUTED COORDINATION using TIME-BASED EVENT ORDERING

- there is no central co-ordination unit that is known to all processes that need to co-ordinate with each other, and at any time,

- distributed co-ordination mechanisms:

1. multiple logical clocks

2. ring-based algorithm

Multiple logical clocks

- processes, which want to enter the critical section, multicast a request message, and can enter it only when all the other processes have replied to this message,

assumptions:

- **there are n processes, which share a single resource, know one another's addresses,**
- **a process holding the resource must release it before it can be granted to another process,**
- **each process p_i keeps a logical clock updated according to Lamport's rule,**
- **different requests for the resource must be granted in the order which they are made,**
- **each process maintains its own request queue which is never seen by any other process,**
- **the request queues initially contain the single message m_0 sent by the process p_0, which is the process initially granted the resource, with timestamp TS_0,whose value is less then the initial value of any logical clock,**

solution:

- **processes that require entry to a critical section multicast a request message, and can enter to the critical section when all the other processes have replied to this message.**

algorithm: (Lamport's algorithm)

step No. 1

- **process p_i sends a request resource message m_i to every other process with timestamp TS_i, and puts the message on its request queue,**

step No. 2

- **when a process p_j receives the request resource message m_i from process p_i, it places it on its request queue and sends an acknowledgement to process p_i with a timestamp TS_j,**

step No. 3

- **process p_i accesses the resource when the following two conditions are satisfied:**

  1. there is a request resource message m_i in its request queue which is ordered before any other request in its queue by totally ordered relation,

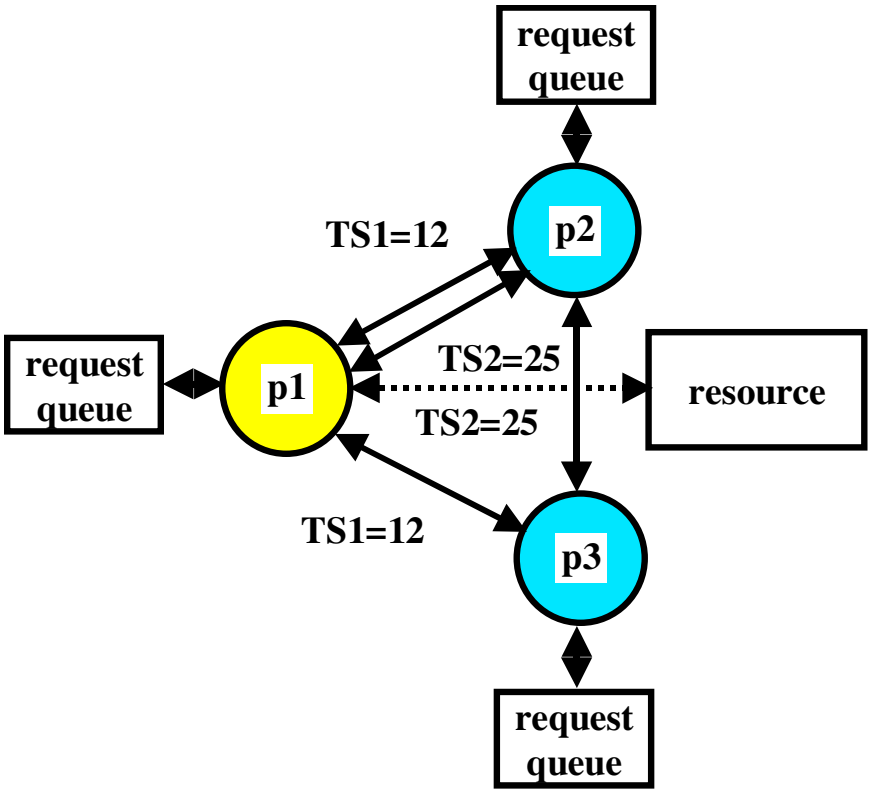
  2. process p_i has received a message from every other process timestamped later than TS_i,

step No. 4

- **to release the resource, process p_i removes any request resource message from its request queue and sends a timestamped release resource message m_i to every other process,**

step No. 5

- **when process p_j receives the release resource message, it removes any request resource message from its request queue,**



- p_3 is not interested in the resource thus, it does not participate in the conquest,

- p_1 and p_2 compete for the resource. Since the timestamp of p_2 is less than the one of p_1, the winner is p_2.

advantages:

- fully distributed algorithm

disadvantages:

- requires $3(n-1)$ messages to allocate the resource i.e. to ensure the critical section entry, where n is the number of processes

- the failure of one node would make progress impossible,

- no performance gain over the single server bottleneck,

Ring-based algorithm

assumptions:

- **processes are arranged in a logical ring where address of each process is given by its neighbour in the clockwise direction,**
- **a process has access to the critical region if it has a token, which is a message passed from process to process in the clockwise direction,**

algorithm:

- **processes compete for the token to access the critical region,**

step No. 1

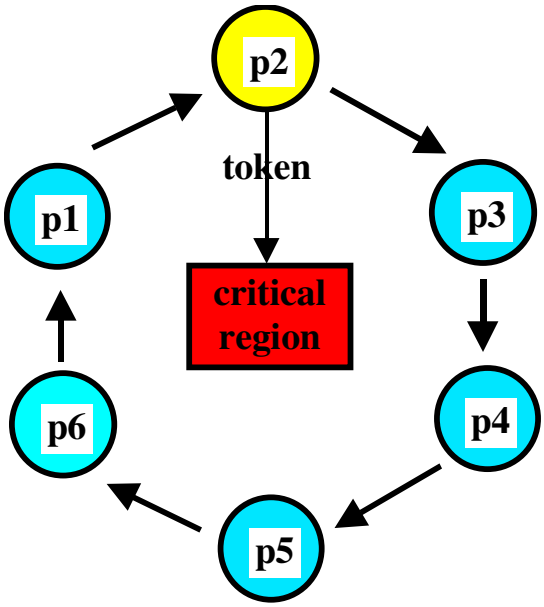
- **any process, which wants to access to the critical region, waits until receives the token,**

step No. 2

- **receiving the token, it retains the token and obtains the access right to the critical region**

step No. 3

- **after completing the access to the critical region, the process sends the token on to its neighbour,**



analysis:

- **the algorithm satisfies all requirements but happened-before relation,**
- **it can take from 1 to n-1 messages to obtain the token, from the point at which the token becomes obtained**

ELECTION

- a set of processors (or group) makes an agreement on a leader when the distributed system needs a centralised co-ordinator,

***co-ordinator* = processor which is the leader of a group**

***participant* = all other processors in a group,**

- co-ordinator is centralised controller that simplifies process co-ordination, however it may fail and limit service availability,

- election has to find another co-ordinator while if either a system starts up or a co-ordinator fails and computation stops; the new co-ordinator has to determine the state of the system and restarts the computation,

- fault detection is based on time-out, i.e. a process that gets no response from the leader for a predefined time suspects a failure of the leader and initiates the election procedure,

- there are two basic election criteria:

- **leader election based on a global priority (extrema finding)**
- **leader election based on personal preference (preference-based)**

- election is similar to co-ordination (mutual exclusion) because all processors must come to an agreement about who owns a token, but there are some differences between them

- **in co-ordination all participants have to know that they do not hold the token**
- **in election all participants have to know who owns the token,**

Bully algorithm for election

- after a failure the algorithm selects the surviving member with the largest identifier is selected as the co-ordinator (bullying),

- low-priority node will not attempt to establish itself as the leader while a higher-priority node is still running in the system,

- when the failed process is restarted, it begins a new election; if it has the highest identifier then it will become again the co-ordinator replacing the functioning co-ordinator (bullying),

assumptions:

- **members of the group know the identities and addresses of all members,**
- **the communication is reliable and only the communicating processes may fail,**
- **the message delivery subsystem delivers all message within T_m of the sending of the message (message propagation time),**
- **a computer responds to all messages within T_p of their delivery (message handling time),**
- **the last two assumptions allow the algorithm to build a reliable fault detector because if a processor does not answer a message within $2\,T_m + T_p$ it must have failed,**
- **there are three types of messages:**

election	**- to announce an election**
answer	**- to response to an election message**
co-ordinator	**- to announce the new co-ordinator**

algorithm:

- **a process begins an election when it notices that the co-ordinator has failed, or a node recovers from a failure,**

step No. 1

- **a process sends an election message to all processes with a higher identifier and awaits answer message(s),**

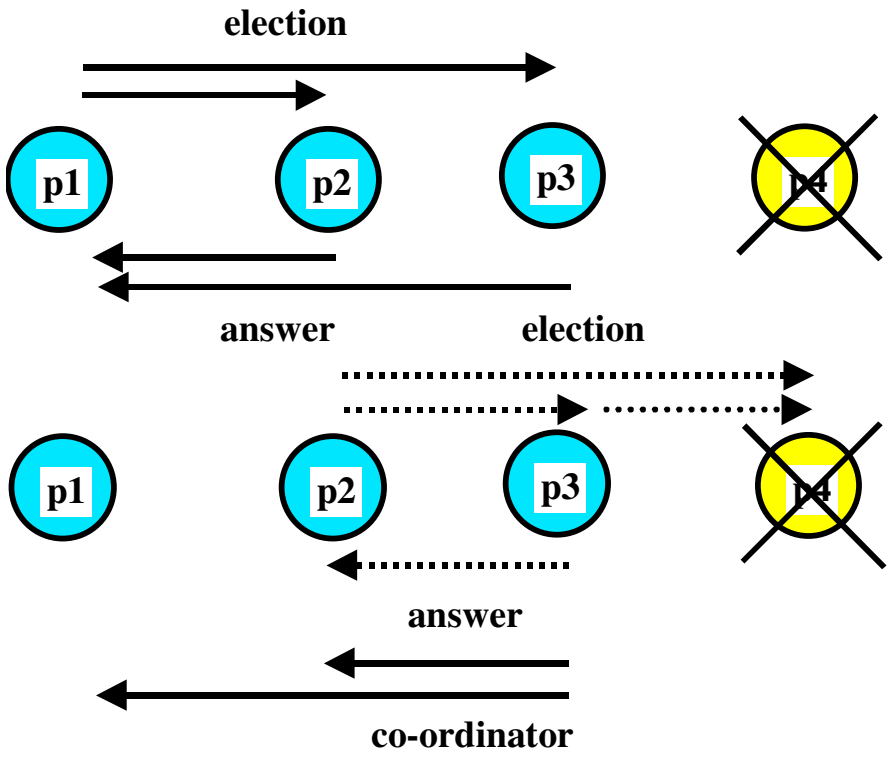
step No. 2

- **processes, receiving election messages, send back answer messages, and begin another election to define the co-ordinator,**

step No. 3

- **if a process with a higher identifier answers, it takes over as co-ordinator, and sends co-ordinator message to all processes with a lower identifier,**

- **if no message arrives within a certain time, the process, sending the election message, considers itself as co-ordinator and sends a co-ordinator message to all processes with a lower identifier,**



analysis of the algorithm:

- **best case**

  - the process with the second highest identifier notices the co-ordinator has failed. It can immediately elect itself sending (n-2) *co-ordinator* messages.

- **worst case**

  - the process with the lowest identifier detects the co-ordinator's failure. It requires $O(n^2)$ messages.

Ring algorithm for election

assumptions:

- **processes do not know the identities of the other processes, they only know the address of the neighbour process (or successor) because they are either physically or logically ordered,**
- **the communication is reliable and processes cannot fail during election,**
- **there are two types of messages:**

  election **to announce an election procedure**

  elected **to announce the new co-ordinator**

- **there are two possible states for processes:**

  non-participant **initial state**

  participant **transient state during election**

algorithm:

- **the algorithm selects a single co-ordinator, which is the process with largest identifier,**
- **when any process notices that the co-ordinator is not functioning, it sends an election message containing its own identifier to its neighbour,**

step No. 1

- **any process can begin an election by marking itself as a participant, placing its identifier in an election message and sending it to its neighbour,**

step No. 2

- **receiving an election message the process compares the identifier in the message with its own identifier,**

- if the identifier in the message is greater than its identifier, it forwards the message to its neighbour and marks itself as participant,

- if the identifier in the message is smaller than its identifier and, it substitutes its own identifier in the message and forwards it and marks itself as participant,

step No. 3

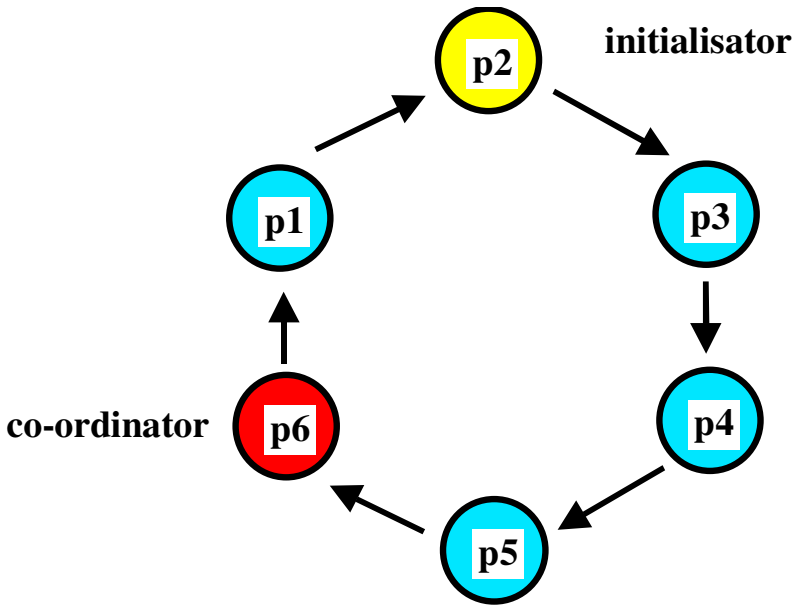
- if the identifier in the message is equal to the identifier of the receiving process,

- the message made a full circle and returned to the node with the highest identifier, i.e. the co-ordinator and the election has been done,

- the process marks itself as non-participant and sends an elected message to its neighbour to announce itself as the new co-ordinator,

step No. 4

- receiving an elected message the process marks itself as non-participant and forwards the elected message to its neighbour,



analysis of the algorithm:

- **worst case**

- if a single process starts an election then the worst case is when its anticlockwise neighbour has the highest identifier,

- total *n-1* messages are required to reach this neighbour, it will not announce its election until its identifier has completed another round, taking further *n* messages, the elected messages will be sent also *n* times,