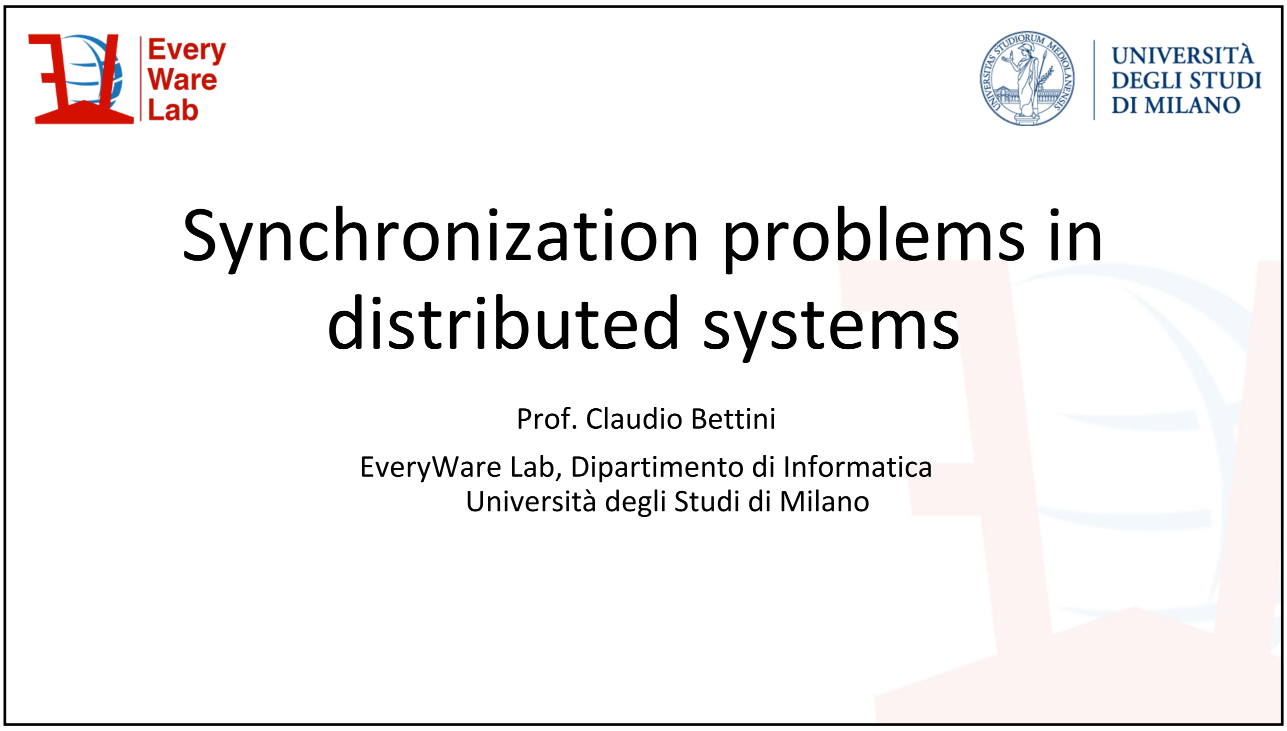
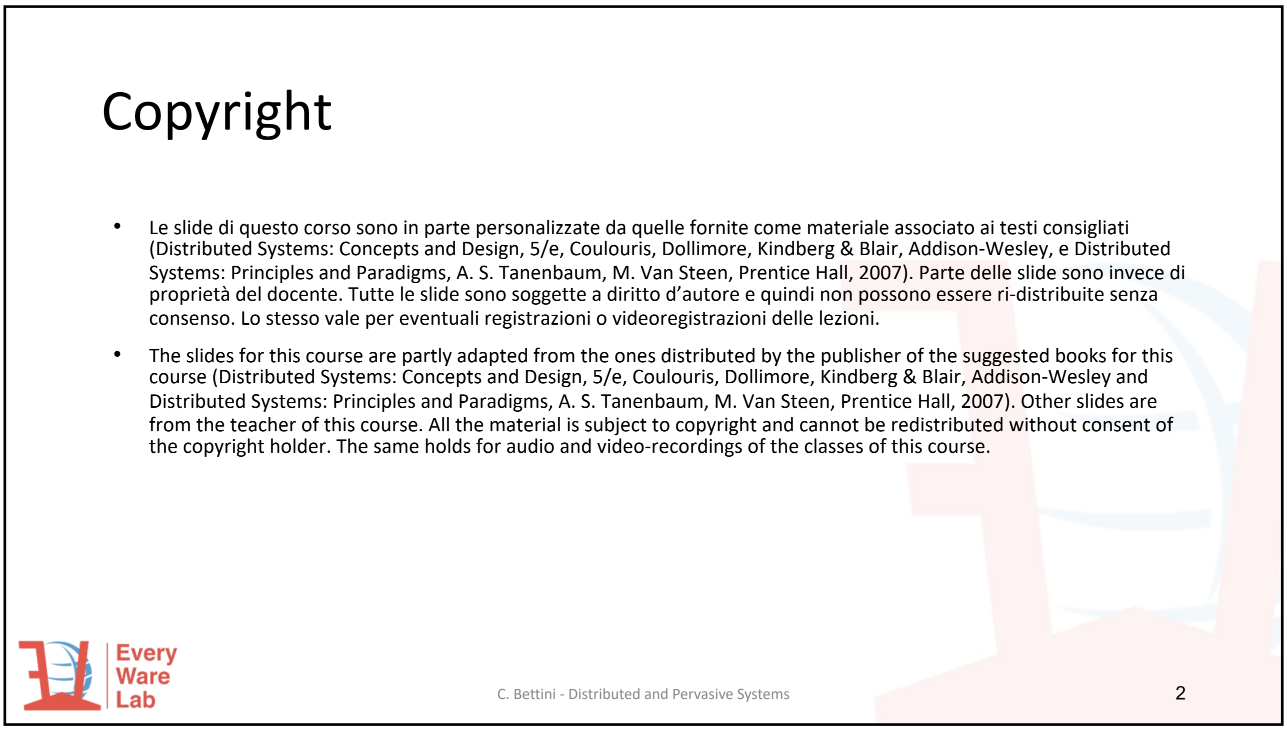


Synchronization problems in distributed systems

Prof. Claudio Bettini

EveryWare Lab, Dipartimento di Informatica
Università degli Studi di Milano

1



Copyright

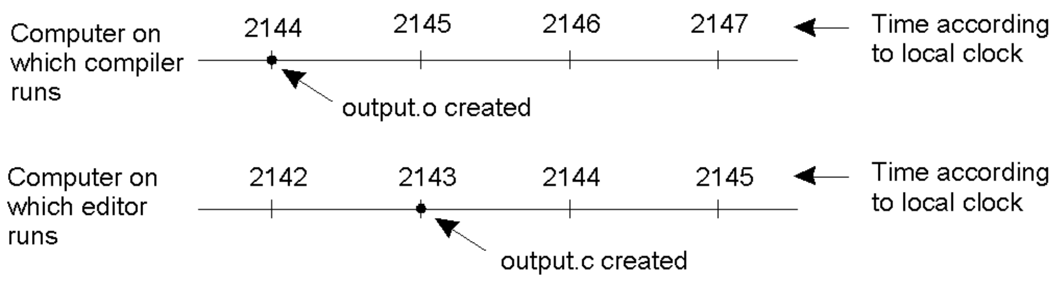
2

Outline

- Physical clock synchronization
- Logical clocks
- Mutual exclusion algorithms
- Election algorithms

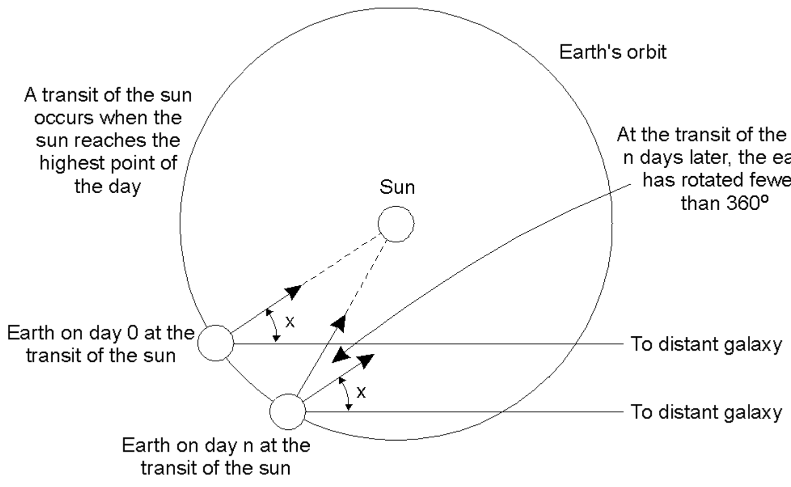
Clock Synchronization

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


Computer on which compiler runs
2144
2145
2146
2147
Time according to local clock
output.o created
Computer on which editor runs
2142
2143
2144
2145
Time according to local clock
output.c created

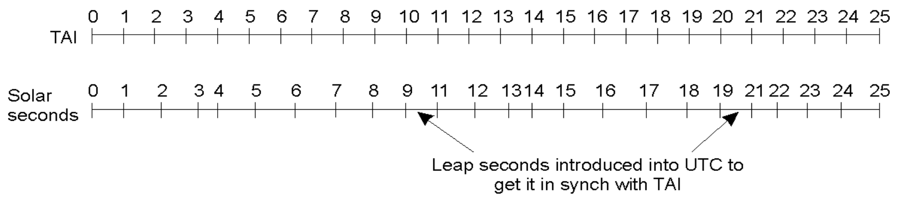
Physical Clocks

Computation of mean solar day


Earth's orbit
A transit of the sun occurs when the sun reaches the highest point of the day
Sun
At the transit of the sun n days later, the earth has rotated fewer than 360°
x
x
Earth on day 0 at the transit of the sun
To distant galaxy
To distant galaxy
Earth on day n at the transit of the sun

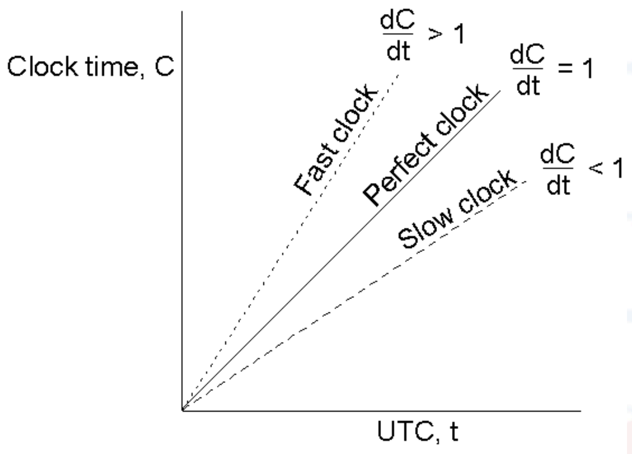
UTC = TAI + leap seconds

TAI (International Atomic Time) seconds are of constant length, unlike solar seconds. Leap seconds are introduced (once in a while) to keep our time in phase with the sun.



Slow and fast clocks

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



7

Synchronizing Physical Clocks

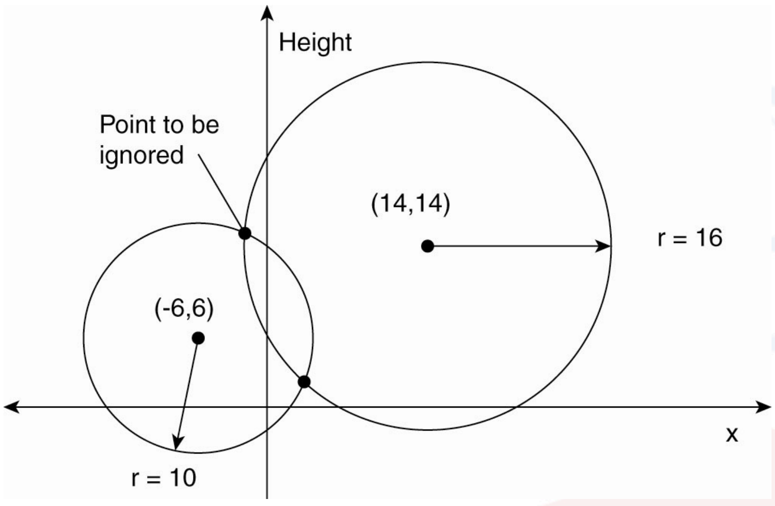
8

Method 1: Using GNSS

- GPS is the most used Global Navigation Satellite System (GNSS)
- GNSS satellites have atomic clocks onboard
- GNSS *satellites broadcast messages* that include timing information
- GNSS *receivers listen* to multiple satellites and use trilateration to determine their own position and UTC time deviation
- The time obtained by GNSS has an accuracy in the order of micro- or even nanoseconds (but really depends on the accuracy of the GNSS)

Global Positioning System (1)

Computing a position in a two-dimensional space.

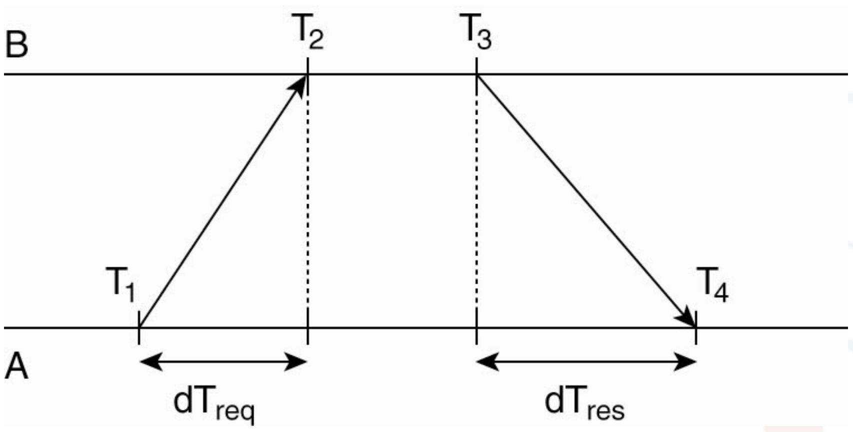


Global Positioning System (2)

Real world facts that complicate GPS

1. It takes a while before data on a satellite's position reaches the receiver.
2. The receiver's clock is generally not in sync with that of a satellite.

Method 2: Cristian’s algorithm and Network Time Protocol

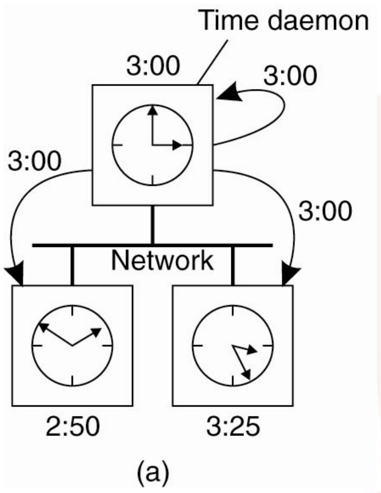


Getting the current time from a time server.

Method 3 (Internal synchronization): The Berkeley Algorithm

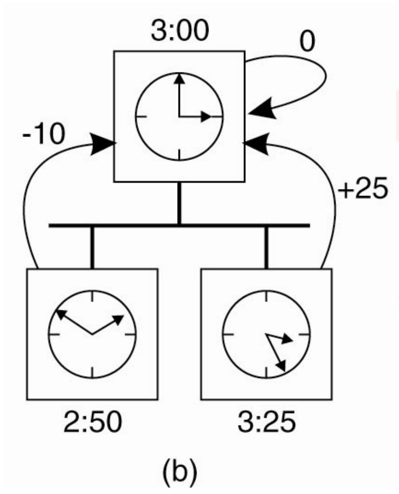
It is not always necessary to align the DS node clocks with external physical clocks!

(a) The time daemon asks all the other machines for their clock values.



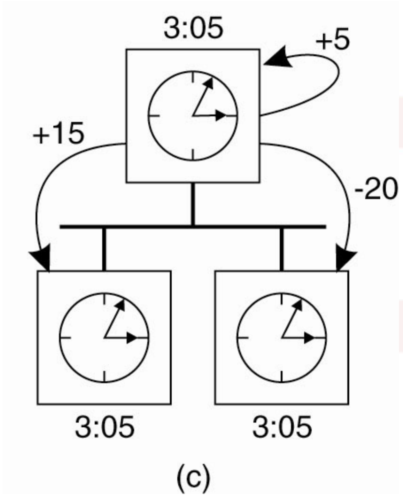
The Berkeley Algorithm (2)

(b) The machines answer.



The Berkeley Algorithm (3)

(c) The time daemon tells everyone how to adjust their clock.



Lamport’s Logical Clocks

Logical Clocks

- In many cases it is not necessary that nodes agree on their physical clocks value
- It may be sufficient to share the knowledge of a partial order of events on the DS (this is the case of the "make" example above)
- Seminal work by Lamport (1978)

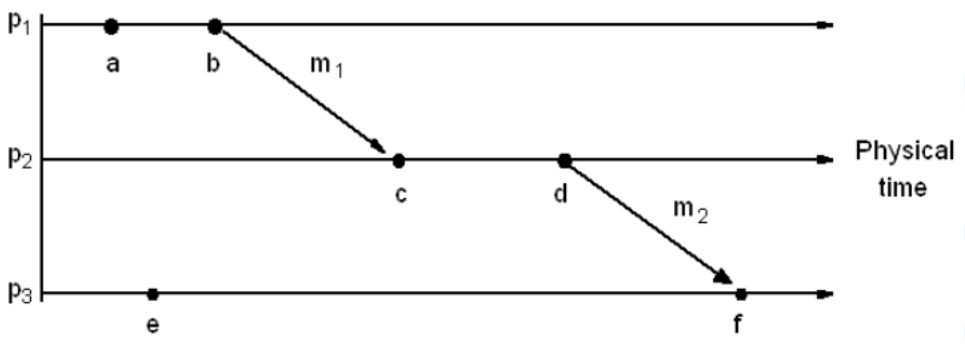
The logical clocks idea

- A way for the nodes to agree on the order in which important events in the system occur
- An event is an internal event or a message being sent/received
- An integer counter is used at each node as a logical clock. It is incremented every time an interesting event occurs
- Considering a given instant in real time, the logical clocks on the different nodes may have different values

Lamport's algorithm (1)

- If *a, b* are events, the expression *a* --> *b* denotes the relation "*a* happens before *b*". It is a transitive relation.
- C(a) is the logical clock value assigned by the process where *a* occurs. The value of a logical clock can only increase.
- Goal: if *a* --> *b* holds, then *C(a)* < *C(b)* should hold

Example



a→b, b→c b→f, e→f, .. We cannot say e→b or b→e

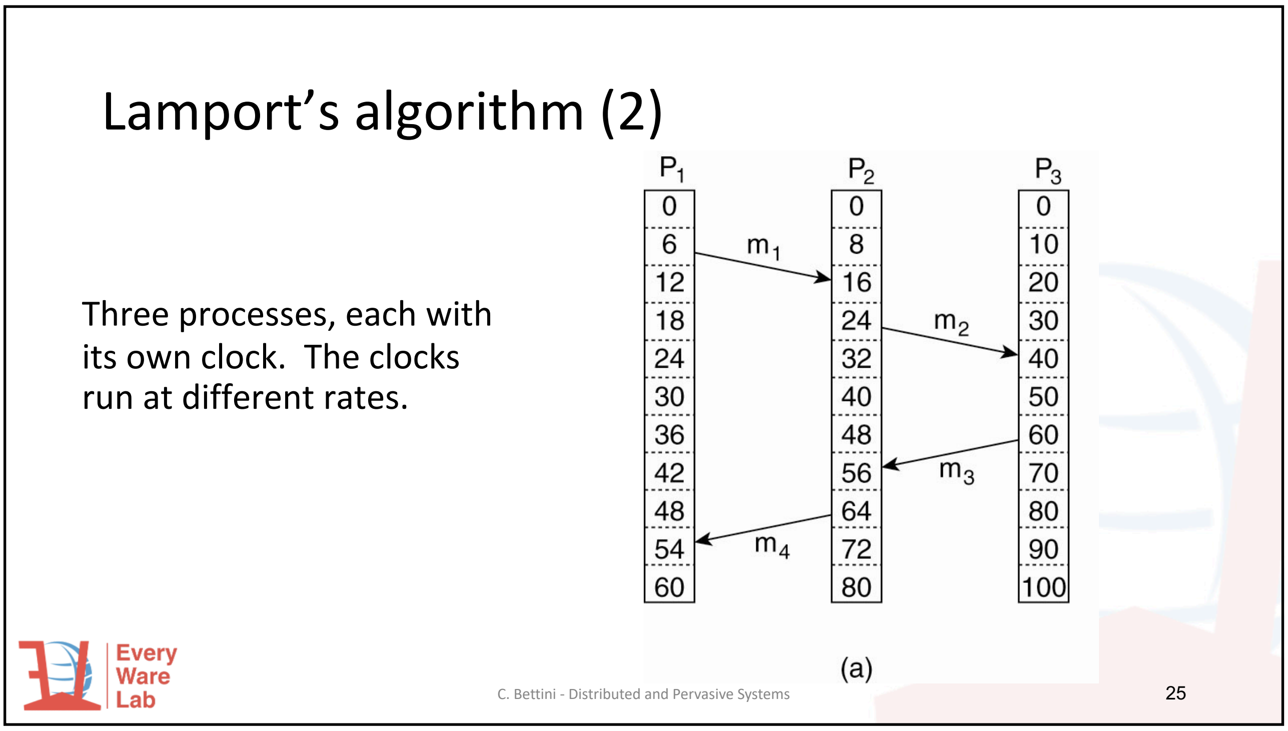
Lamport’s algorithm (1)

- If *a* and *b* occur in the same process the goal is naturally achieved
- If *a* is the event of sending a message *m* by a node and *b* is the event of receiving *m*, then *a* --> *b* holds (a message must be sent before it is received). How to ensure $C(a) < C(b)$?

Lamport’s algorithm

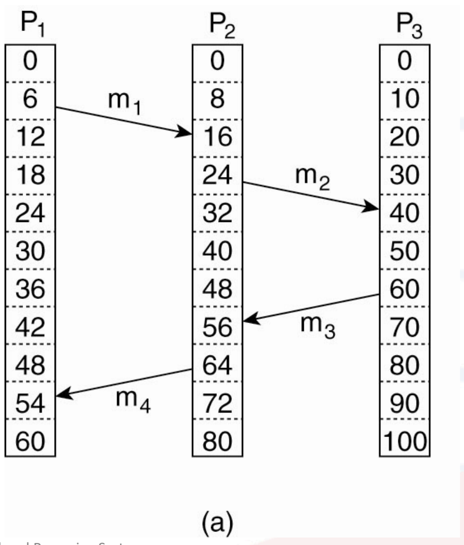
Updating counter C_i for process P_i

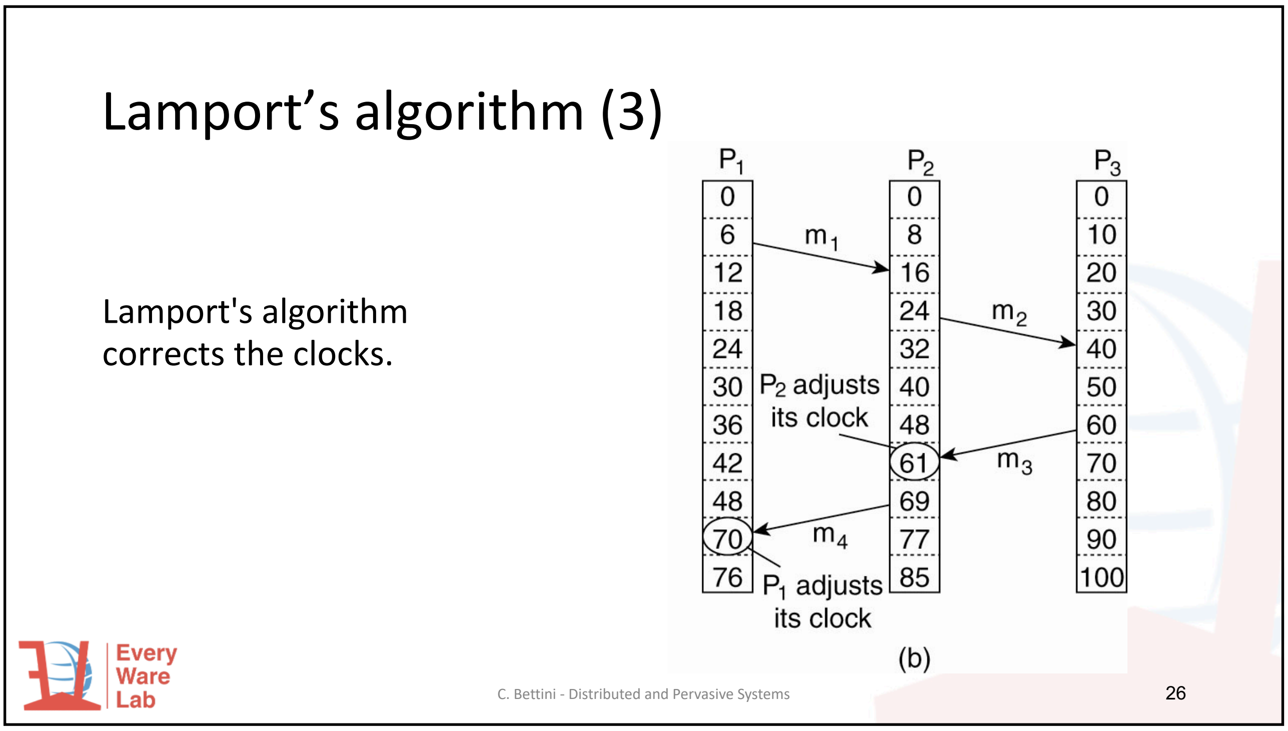
1. Before executing an event (internal or message sending), P_i executes $C_i \leftarrow C_i + 1$
2. When process P_i sends a message *m* to P_j, it sets *m*’s timestamp $ts(m)$ equal to C_i after having executed the previous step
3. When P_j receives message *m at* C_j, adjusts its own local counter as: $C_j \leftarrow \max\{C_j , ts(m)\} +1$



Lamport’s algorithm (2)

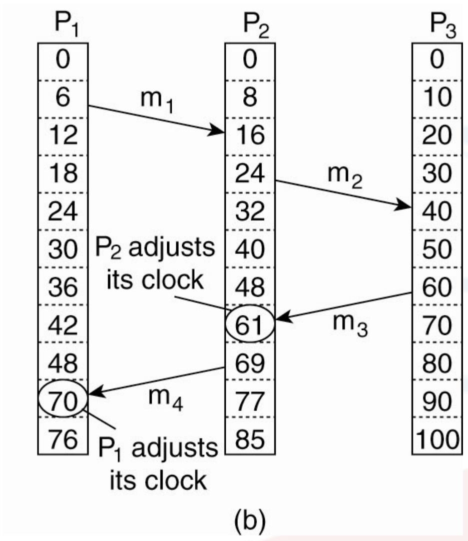
Three processes, each with its own clock. The clocks run at different rates.



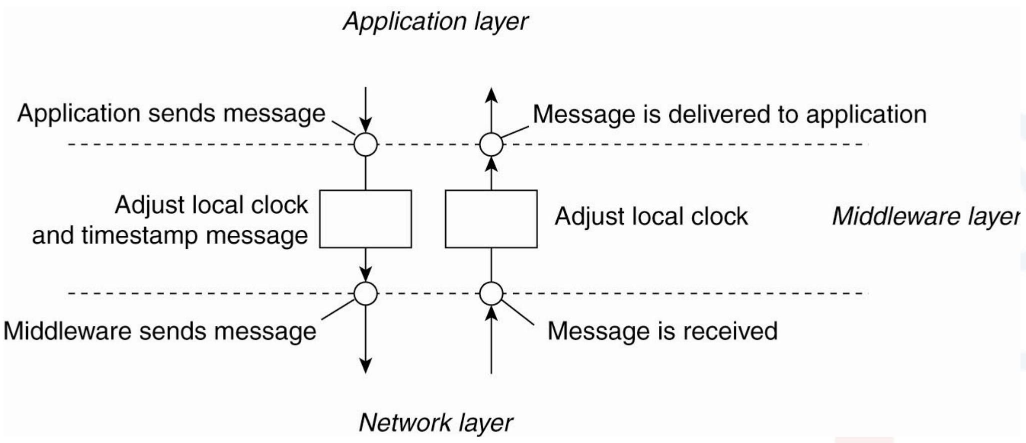


Lamport’s algorithm (3)

Lamport's algorithm corrects the clocks.



Logical clock adjustment in middleware



The positioning of Lamport's logical clocks in distributed systems

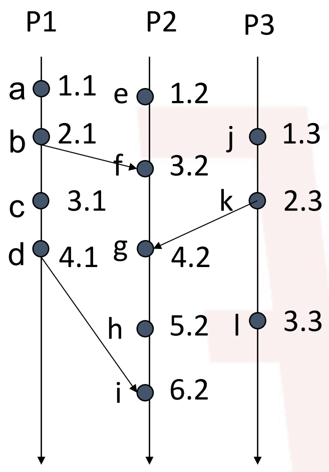
Enforcing Total Order

- The algorithm can be modified by attaching a process number to the timestamp of an event (a unique number can be assigned to each process in the DS).
- P_i timestamps event *e* with $C_i(e).i$ where i is the process number
- For any two timestamps $T1=(C_i(a),i)$ and $T2=(C_j(b),j)$:
  - If $C_i(a) < C_j(b)$ then T1 < T2
  - If $C_i(a) > C_j(b)$ then T2 < T1
  - If $C_i(a) = C_j(b)$ and i < j then T1 < T2
  - If $C_i(a) = C_j(b)$ and i > j then T1 > T2

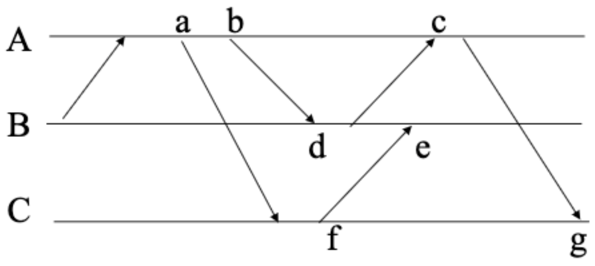
Example (total order)

Each event in the system has a different timestamp.

Note: Total order of timestamps does not mean that we know the actual temporal relationship between any pair of events! (e.g., we still don't know e→b)



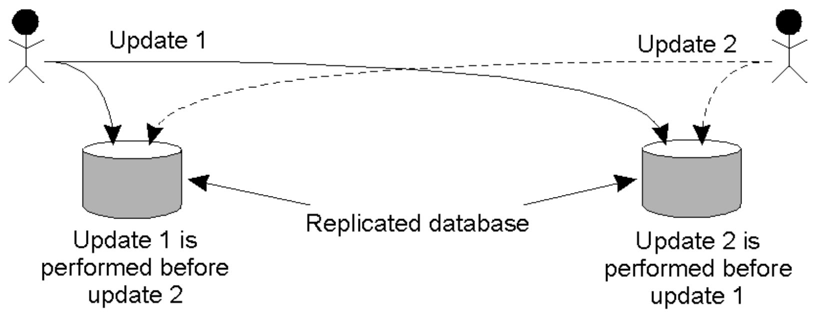
Excercise



Assuming the only events are message send and receive, what are the clock values at events a-g?

Application: Totally-Ordered Multicasting

- Updating a replicated database and leaving it in an inconsistent state.


Update 1
Update 2
Replicated database
Update 1 is performed before update 2
Update 2 is performed before update 1

Implementing Totally Ordered Multicast

- Assumptions:
  - No messages are lost
  - Messages from the same sender are received in the order they were sent
- Process P_i sends timestamped message m_i to all others. The message itself is put in a local queue i
- Any incoming message at P_j is queued in queue j, according to its timestamp, and ACKed to every other process (send and receive events for messages and acks are totally ordered with Lamport)

Implementing Totally Ordered Multicast

- P_j passes a message m_i to its application if
  - m_i is at the head of queue j AND
  - m_i has been ACKed by each of the other processes
- Observation: all processes will eventually have the same copy of the local queue, therefore, all messages are passed to application in the same order everywhere

Algorithm applied to previous example (1)

- P1 = agent requiring update1; P2 = agent requiring update2; P3 = agent managing DB1; P4 = agent managing DB2
- Lamport algorithm is used to impose a total order on logical clocks. Suppose P1 sends its update at t1 and P2 at t2 with t1 < t2
- P1 sends m1=(P1, t1, +100) to all others and puts m1 in Q1
- P2 sends m2= (P2, t2, +1%) to all others and puts m2 in Q2
- P3 receives m1 and puts it in Q3; then sends m3=(P3, t3, ackM1) to all others
- P4 receives m2 and puts it in Q4; then sends m4=(P4, t4, ackM2) to all others
- P1 and P2 also receive m1 and m2 and send the corrisponding acks to everyone

Algorithm applied to previous example (2)

- P3 receives m2 and puts it in Q3; by ordering messages according to timestamps Q3=(m1,m2) ; then sends m5=(P3, t5, ackM2) to all others
- P4 receives m1 and puts it in Q4; by ordering messages according to timestamps Q4=(m1,m2) ; then sends m6=(P4, t6, ackM1) to all others
- When P3 receives ackM1 from ALL the other agents it knows that the order of messages in its queue is the same for everyone and can pass the updates to the application (DB). Similarly for m2. The same will do P4 that has the same order in its queue.

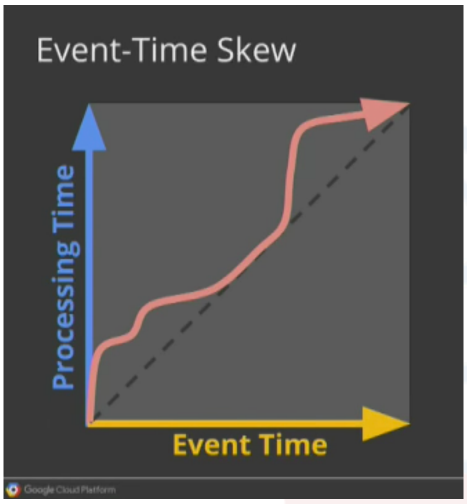
Every Ware Lab

Event vs Processing Time

Example: Mobile gaming with teams of distributed, possibly temporarily disconnected, clients.
Goal: earning points by performing tasks. Partial score for each team?

Problem:

- events have timestamps
- game values time of answer
- unpredictable delays occur in communicating answers with their timestamps



A nice video on how Google dataflow handles the example

https://www.youtube.com/watch?v=3UfZN59Nsk8

Every Ware Lab

Homeworks

- Try to apply Lamport’s algorithm with and without total order starting from a variation of the example in the slides.
- Consider the example of the concurrent updates of the replicated database and apply the algorithm for totally ordered multicast showing each message that is sent and the contents of the queues. Is there a problem if the transmission of an update to one of the DBs is delayed?
- Watch the video on Google stream processing tools https://www.youtube.com/watch?v=3UfZN59Nsk8

Every Ware Lab

Every Ware Lab

UNIVERSITÀ DEGLI STUDI DI MILANO

Mutual exclusion

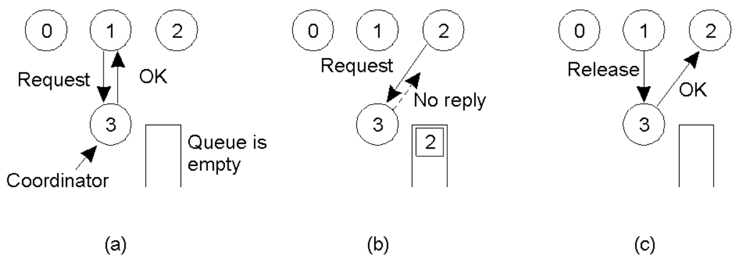
Mutual exclusion

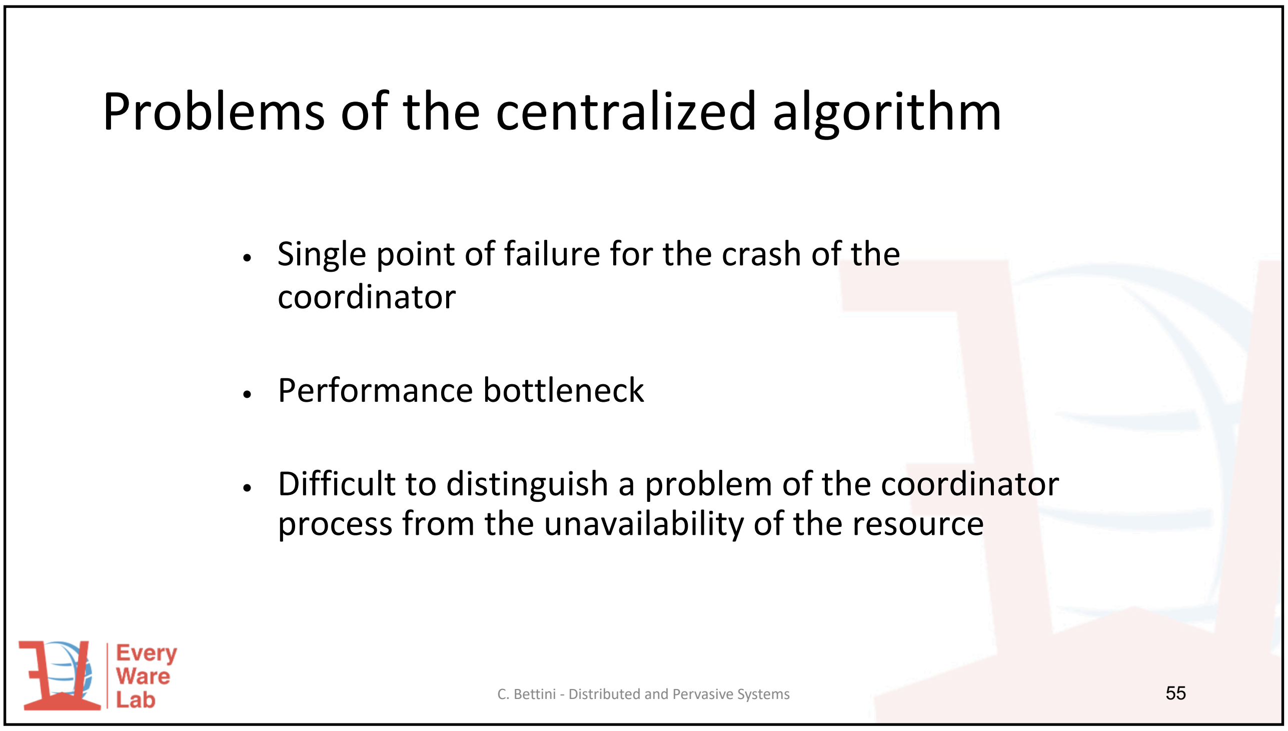
- A set of processes needs to concurrently access a shared resource
- Solution in a single node:
  - Identification of critical regions (sets of instructions that need to access a resource in exclusive mode in order to preserve consistency)
  - Use of semaphores or monitors to ensure that no other process can access the resource when the process is in its critical region.
- Solutions in distributed systems?

A centralized algorithm

a) Process 1 asks the coordinator to access the resource. Access is granted. Then, process 2 asks for access. The coordinator does not answer, but adds the request to the queue.

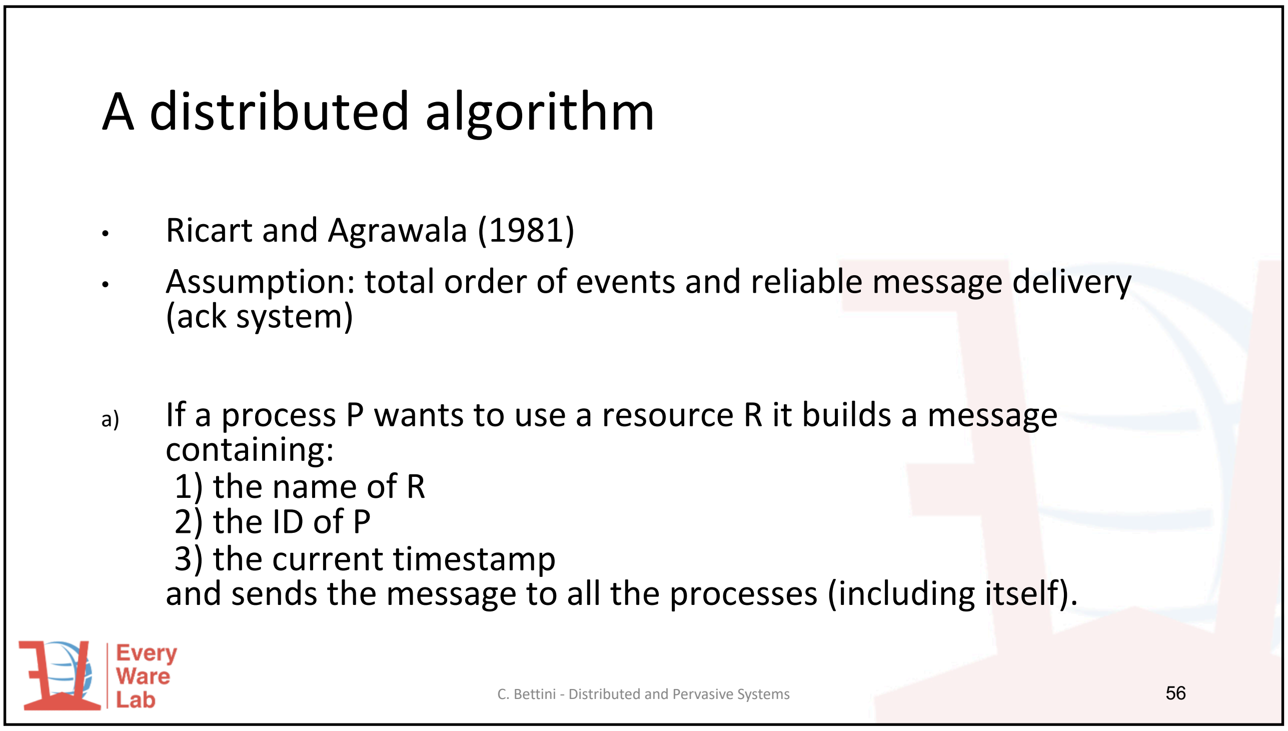
b) When process 1 is done it notifies the coordinator that grants access to the first process in the queue in FIFO order (process 2).





Problems of the centralized algorithm

- Single point of failure for the crash of the coordinator
- Performance bottleneck
- Difficult to distinguish a problem of the coordinator process from the unavailability of the resource



A distributed algorithm

- Ricart and Agrawala (1981)
- Assumption: total order of events and reliable message delivery (ack system)

a) If a process P wants to use a resource R it builds a message containing:
 1) the name of R
 2) the ID of P
 3) the current timestamp
and sends the message to all the processes (including itself).

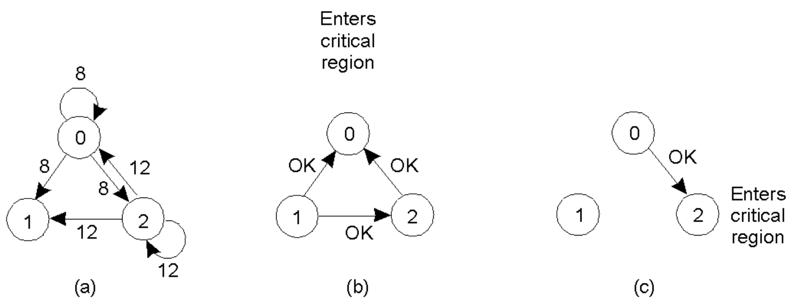
A distributed algorithm (cont.)

b) When a process Q receives a message we have 3 cases:
   i. If Q is not using R and it does not require R, it answers OK to P
   ii. If Q is using R it does not answer and it queues the request
   iii. If Q wants to use R but it did not yet, it compares the message timestamp with the one in the message it sent out with its own request. The earliest wins. If the one in the message sent by P is lower it answers OK to P, otherwise it queues the message.

c) After sending out its message process P waits for OK from all processes before accessing R.

d) When P is done with R it sends OK to all processes in its queue and empties the queue.

Example



a) Processes 0 and 2 want to access R at the same time; P0 broadcasts a message with timestamp 8. P2 broadcasts a message with timestamp 12

b) P0 has the earliest timestamp and it wins; it adds 2 to its local queue and does not answer. P1 sends OK to both P0 and P2, since it is not interested. P2 compares timestamp 8 with its own (12) and sends OK

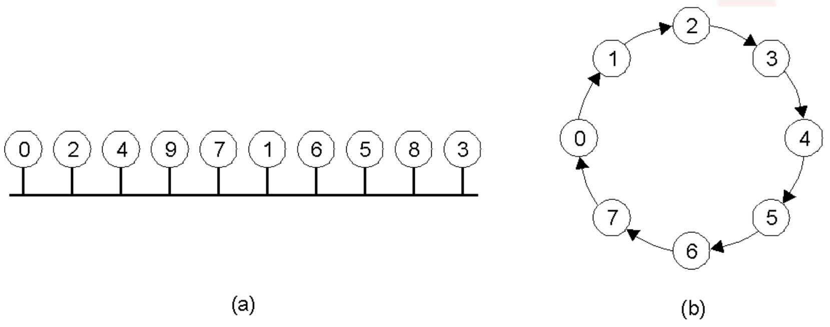
c) When P0 is done with R, it sends OK to P2 (the only one in its queue). P2, having received OK from both P1 and P0 can now access R.

Problems of the distributed algorithm

- No answer from a process may also be due to its crash
- Involving all processes of a distributed system may be a waste of resources

A ring algorithm (1)

a) Unordered group of processes;
b) A software defined logic ring.



A ring algorithm (2)

a) At start, the token is given to process 0.

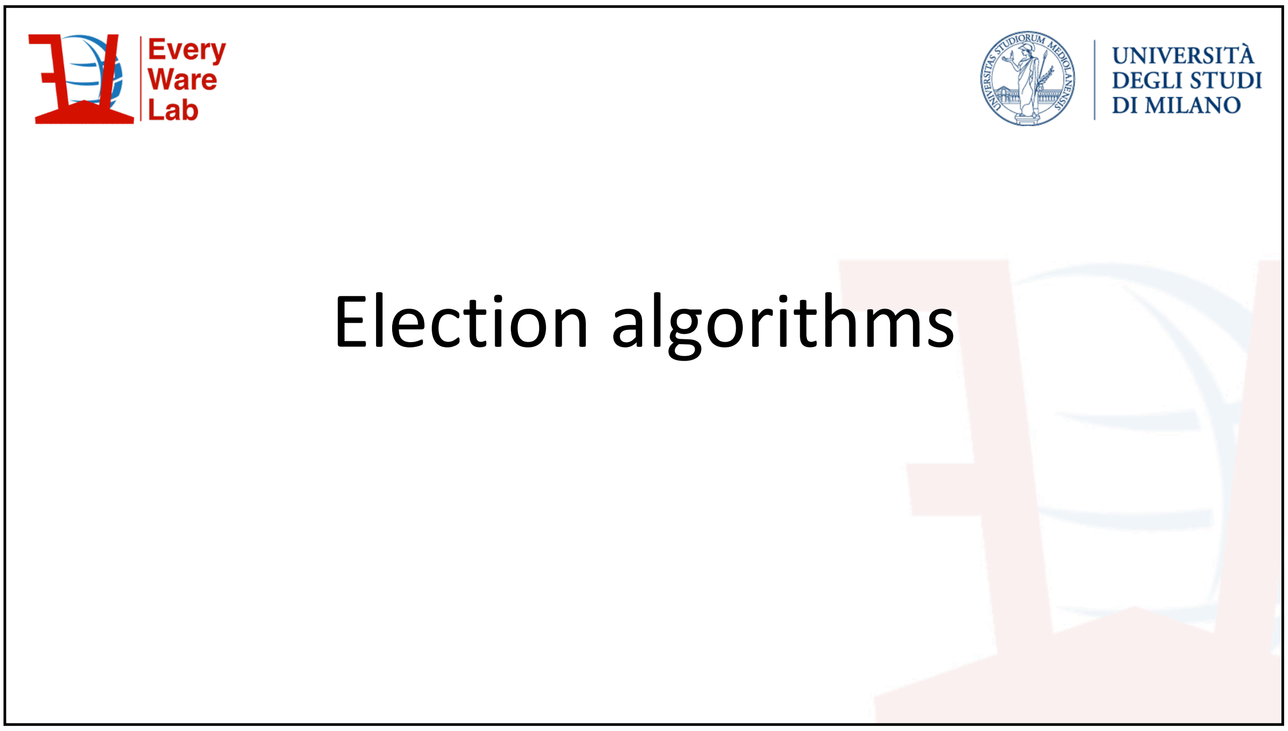
b) The token is sent with a message from process i to process (i+1)MOD n.

c) If a process is interested in the resource, it can access when it receives the token. When done it sends the token to the next process in the ring. It cannot use twice the same token.

Comparison of the algorithms

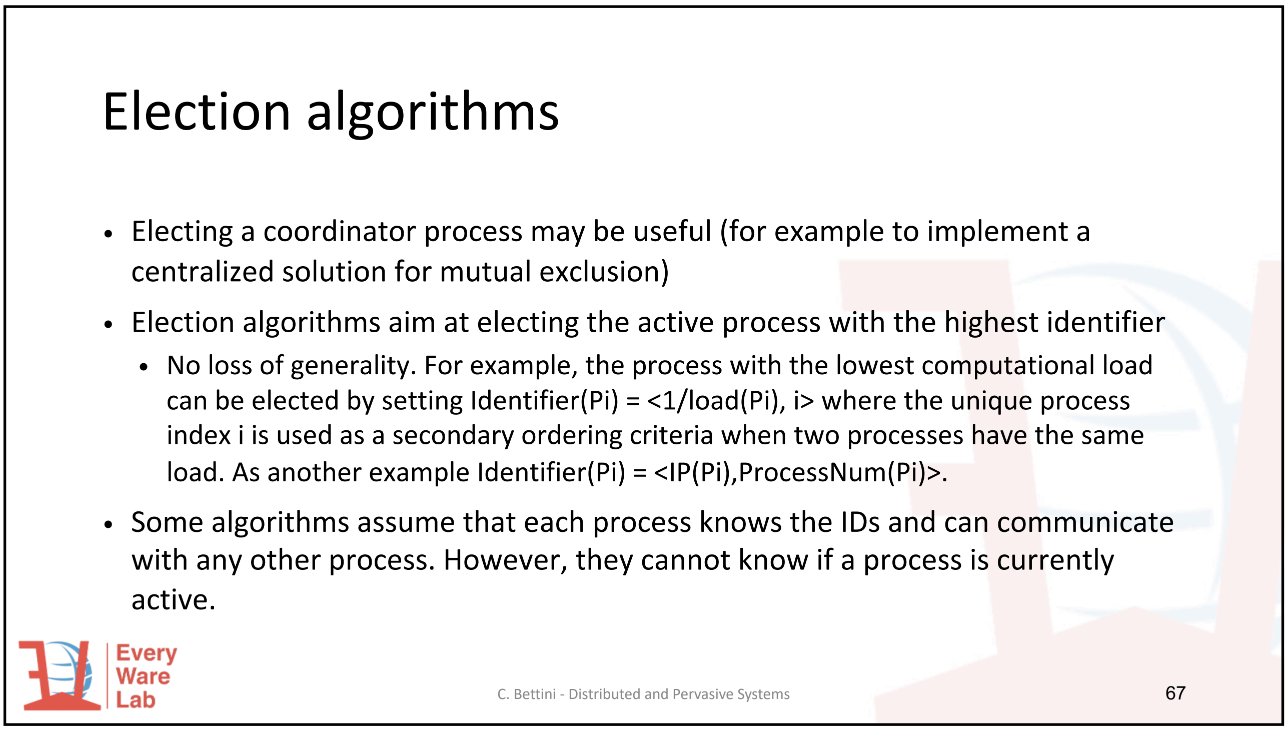
Algorithm	Messages to acquire/release R	Delay to acquire R (num. of messages)	Problems
Centralized	3	2	Crash of coordinator
Distributed	2 (n – 1)	2 (n – 1)	Crash of any process
Distributed as ring	1 to ∞	0 to n – 1	token loss, crash of a process

worst case in which no process requires to acquire R



Election algorithms

63



Election algorithms

- Electing a coordinator process may be useful (for example to implement a centralized solution for mutual exclusion)
- Election algorithms aim at electing the active process with the highest identifier
  - No loss of generality. For example, the process with the lowest computational load can be elected by setting Identifier(Pi) = <1/load(Pi), i> where the unique process index i is used as a secondary ordering criteria when two processes have the same load. As another example Identifier(Pi) = <IP(Pi),ProcessNum(Pi)>.
- Some algorithms assume that each process knows the IDs and can communicate with any other process. However, they cannot know if a process is currently active.

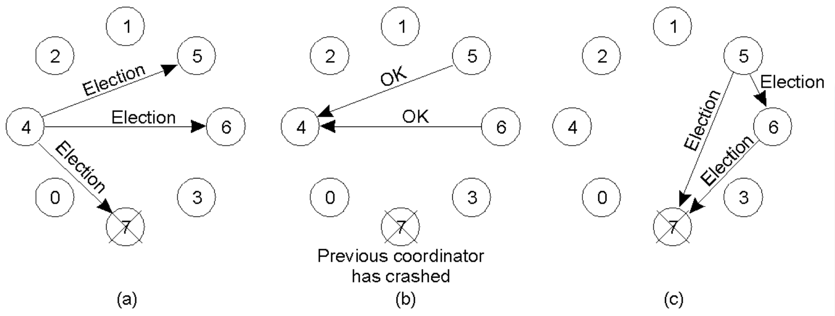
67

Bully Algorithm (1)

a) Process 4 contacts the old coordinator; it understands that it is not active anymore and starts an election by sending a message to all processes with an ID greater than their own

b) Processes 5 and 6 answer telling 4 that they will take care

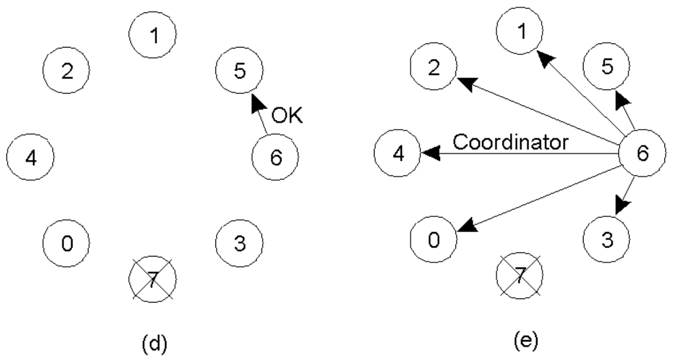
c) Both 5 and 6 start an election

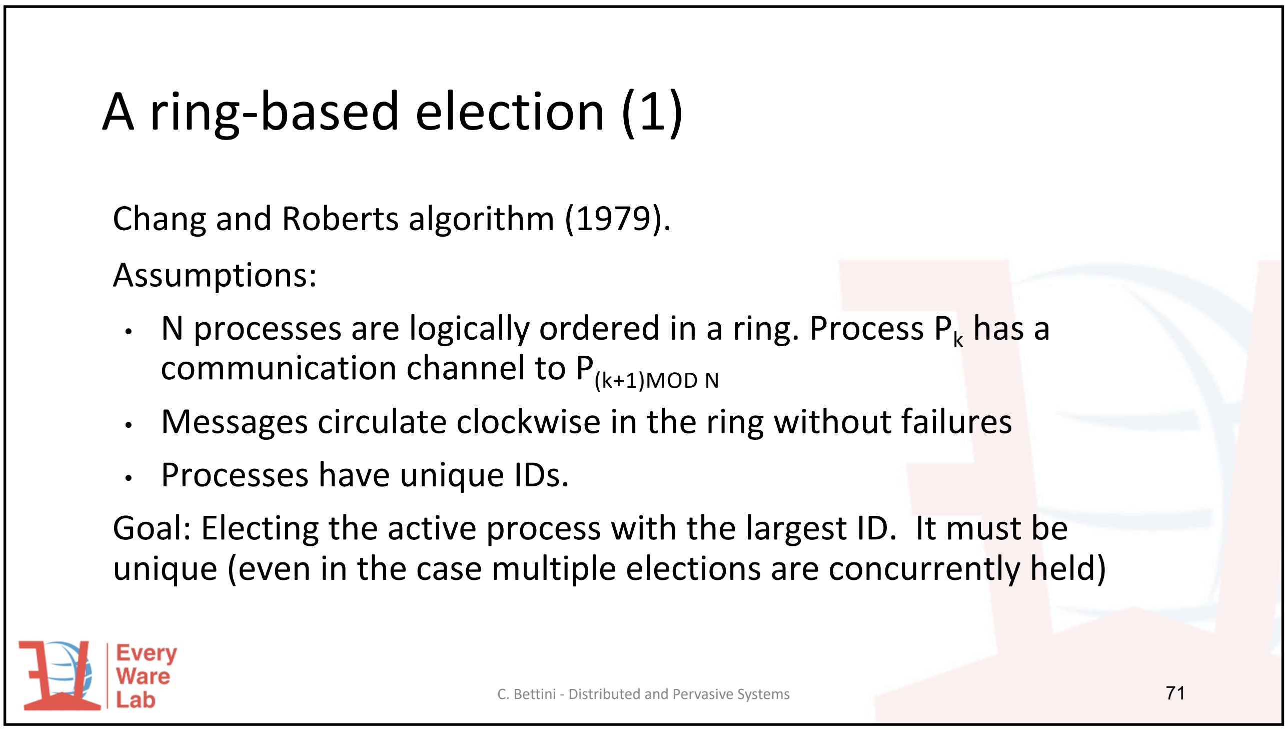


Bully Algorithm (2)

d) Process 6 tells 5 that it will take care of the election

e) Process 6 wins and tells everybody





A ring-based election (1)

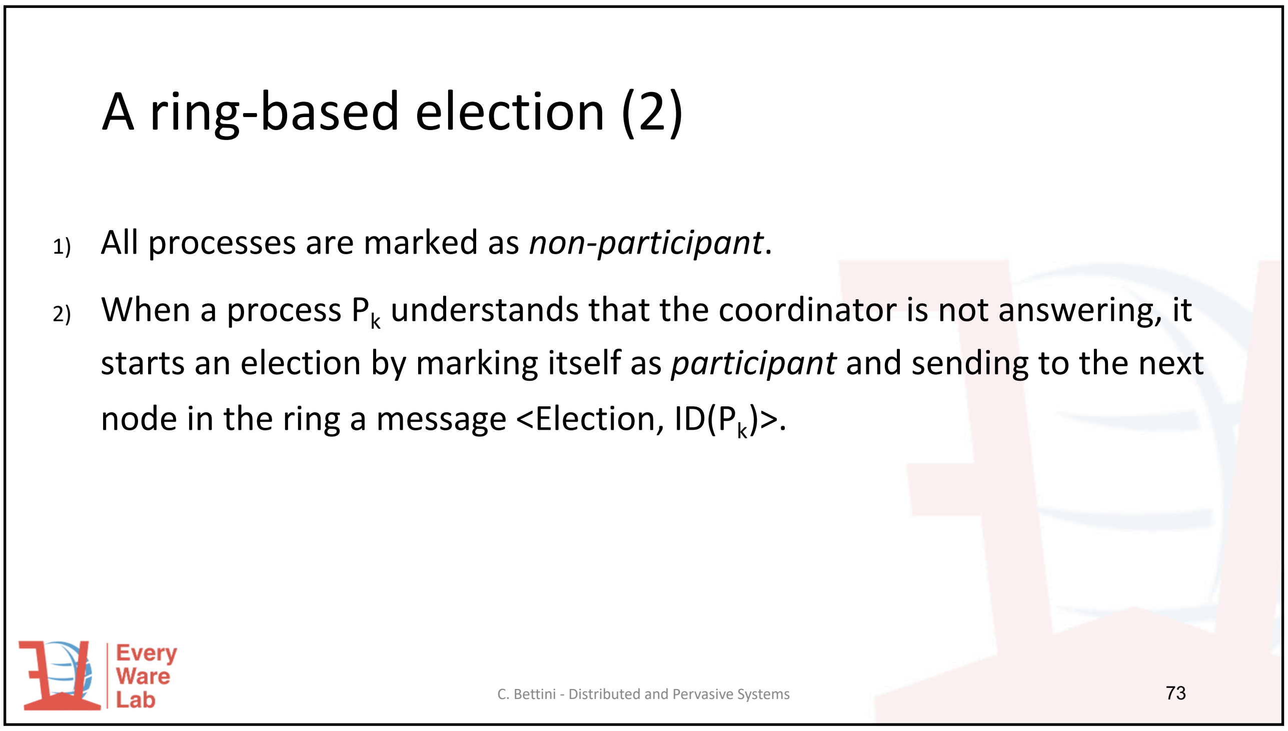
Chang and Roberts algorithm (1979).

Assumptions:

- N processes are logically ordered in a ring. Process P_k has a communication channel to $P_{(k+1)MOD\ N}$
- Messages circulate clockwise in the ring without failures
- Processes have unique IDs.

Goal: Electing the active process with the largest ID. It must be unique (even in the case multiple elections are concurrently held)

71



A ring-based election (2)

1) All processes are marked as *non-participant*.
2) When a process P_k understands that the coordinator is not answering, it starts an election by marking itself as *participant* and sending to the next node in the ring a message <Election, ID(P_k)>.

73

A ring-based election (3)

3) When a process P_m receives <ELECTION, ID(P_k)>,

- if ID(P_k) > ID(P_m) P_m forwards the message in the ring and marks itself as *participant*,
- if ID(P_k) < ID(P_m) if P_m is *non-participant* it changes to *participant* and forwards <Election, ID(P_m)>. If it is already *participant* it does not forward the message.
- If k=m (the received ID is its own) then it is the coordinator: it marks itself as *non-participant* and sends <ELECTED, ID(P_m)> to the next process.

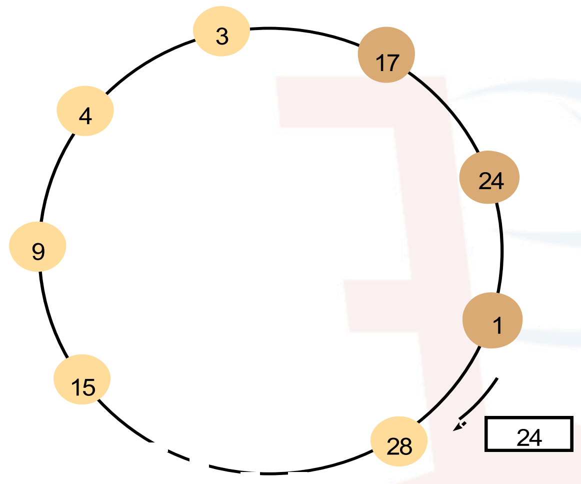
4) When a process P_k receives <ELECTED, ID(P_m)>,

- it marks itself as *non-participant*, stores the ID of the coordinator, and unless k=m it forwards the message to the next process.

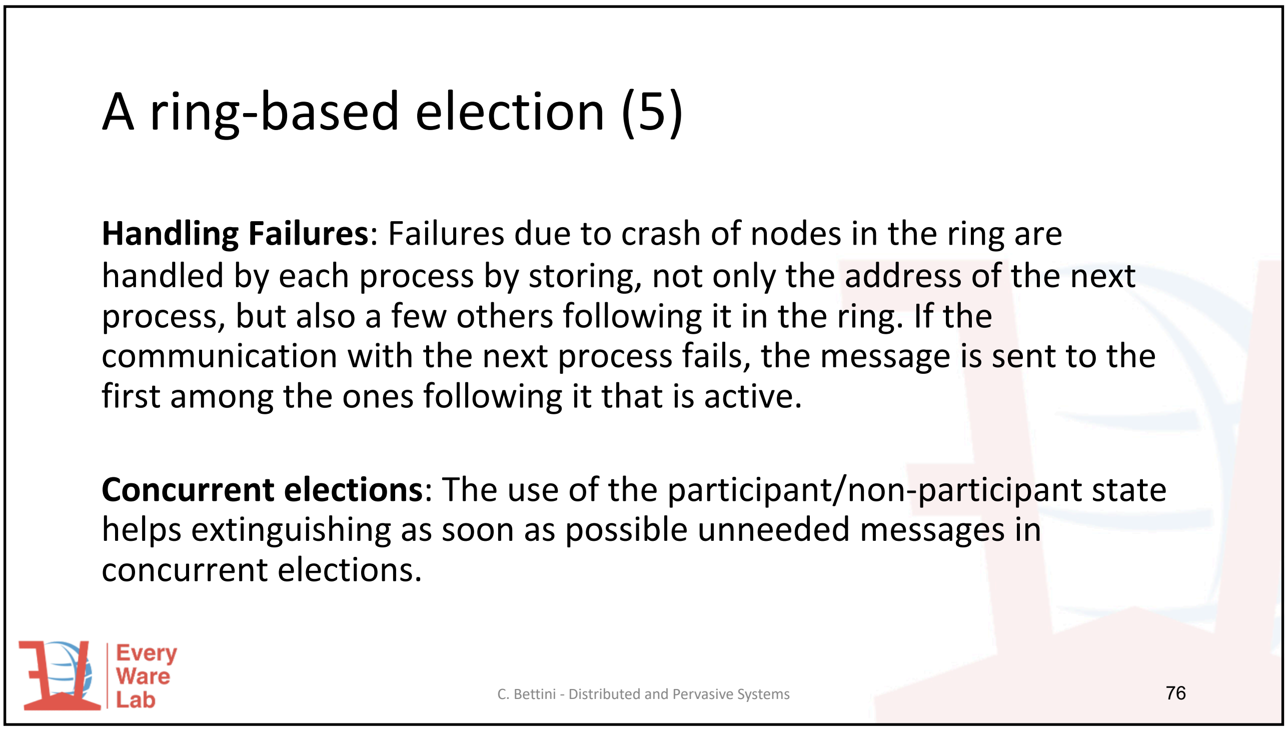
Every Ware Lab

A ring-based election (4)

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



Every Ware Lab

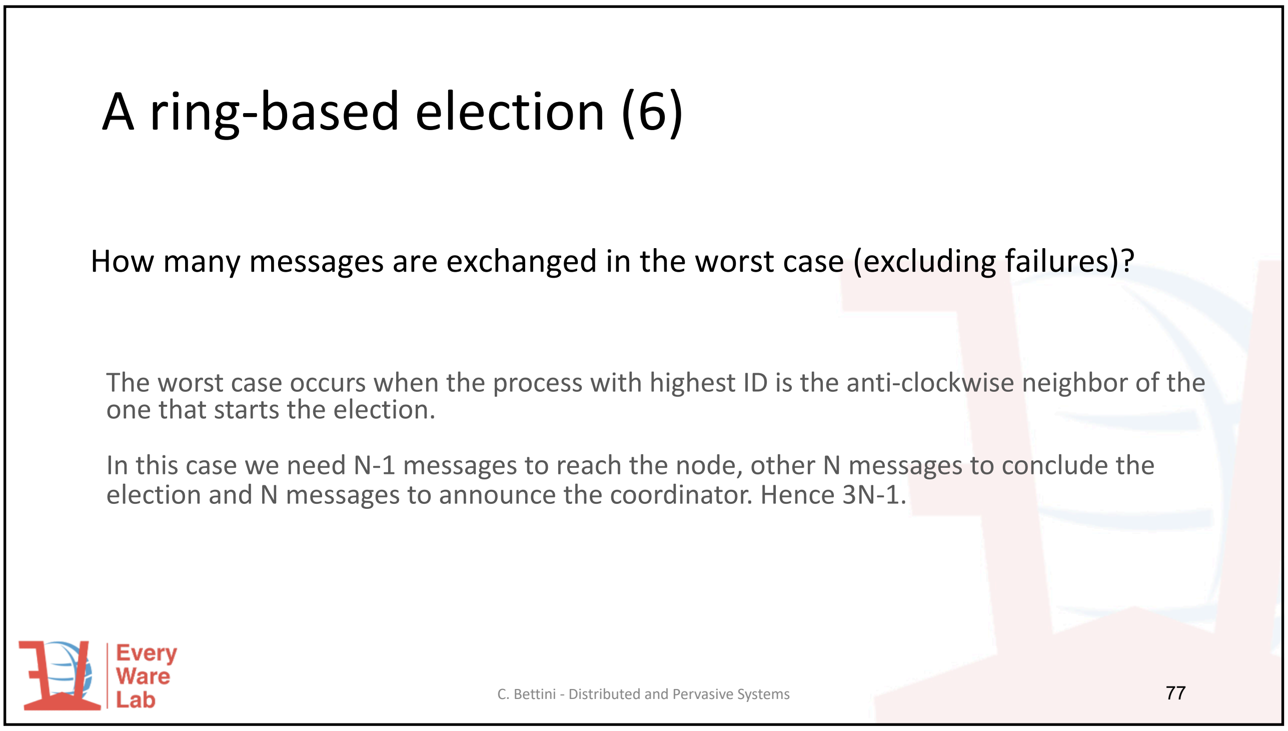


A ring-based election (5)

Handling Failures: Failures due to crash of nodes in the ring are handled by each process by storing, not only the address of the next process, but also a few others following it in the ring. If the communication with the next process fails, the message is sent to the first among the ones following it that is active.

Concurrent elections: The use of the participant/non-participant state helps extinguishing as soon as possible unneeded messages in concurrent elections.

76



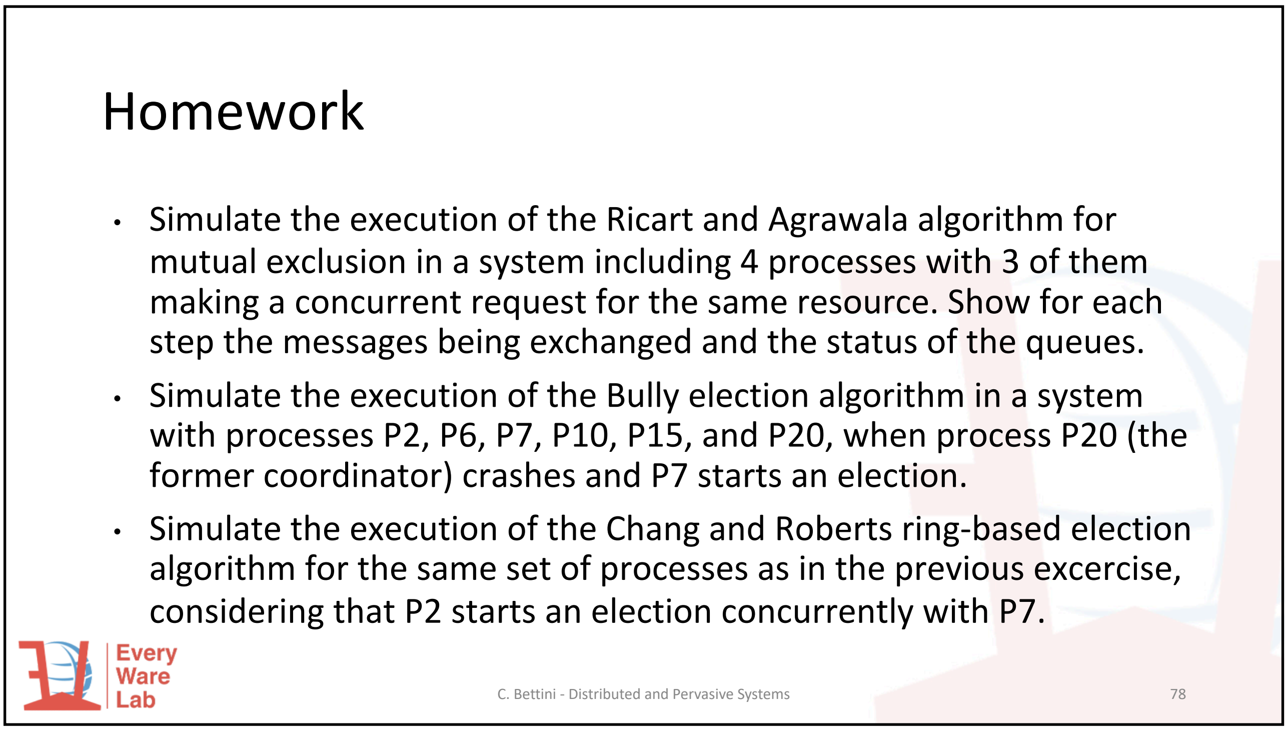
A ring-based election (6)

How many messages are exchanged in the worst case (excluding failures)?

The worst case occurs when the process with highest ID is the anti-clockwise neighbor of the one that starts the election.

In this case we need N-1 messages to reach the node, other N messages to conclude the election and N messages to announce the coordinator. Hence 3N-1.

77



Homework

- Simulate the execution of the Ricart and Agrawala algorithm for mutual exclusion in a system including 4 processes with 3 of them making a concurrent request for the same resource. Show for each step the messages being exchanged and the status of the queues.
- Simulate the execution of the Bully election algorithm in a system with processes P2, P6, P7, P10, P15, and P20, when process P20 (the former coordinator) crashes and P7 starts an election.
- Simulate the execution of the Chang and Roberts ring-based election algorithm for the same set of processes as in the previous excercise, considering that P2 starts an election concurrently with P7.

Every Ware Lab