Dependable Distributed Systems
Master of Science in Engineering in Computer Science

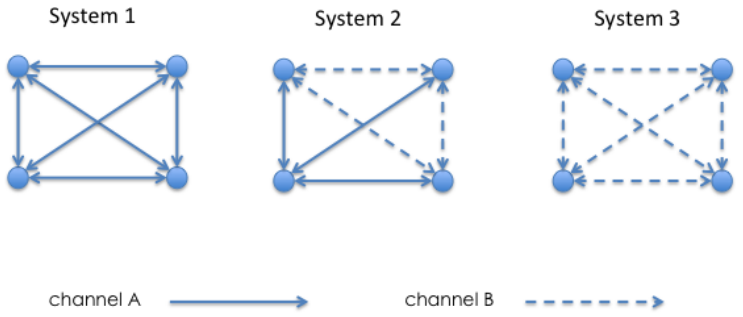
AA 2023/2024

Lecture 8 – Exercises

Ex 1: Let channel A and channel B be two different types of point-to-point channels satisfying the following properties:

- channel A: if a correct process p_i sends a message *m* to a correct process p_j at time *t*, then *m* is delivered by p_j by time $t+\delta$.
- channel B: if a correct process p_i sends a message *m* to a correct process p_j at time *t*, then *m* is delivered by p_j with probability p_{cons} (p_{cons} <1).

Let us consider the following systems composed by 4 processes p1, p2, p3 and p4 connected through channels A and channels B.



Assuming that each process pi is aware of the type of channel connecting it to any other process, answer to the following questions:

1. is it possible to design an algorithm implementing a perfect failure detector in system 2 if only processes having an outgoing channel of type B can fail by crash?
2. is it possible to design an algorithm implementing a perfect failure detector in system 2 if any process can fail by crash?
3. is it possible to design an algorithm implementing a perfect failure detector in system 3?

For each point, if an algorithm exists write its pseudo-code, otherwise show the impossibility.

Ex 2: Consider a distributed system composed by n processes {p1,p2,…,pn} that communicate by exchanging messages on top of a line topology, where p1 and pn are respectively the first and the last process of the network.
Initially, each process knows only its left neighbour and its right neighbour (if they exist) and stores the respective identifiers in two local variables LEFT and RIGHT.

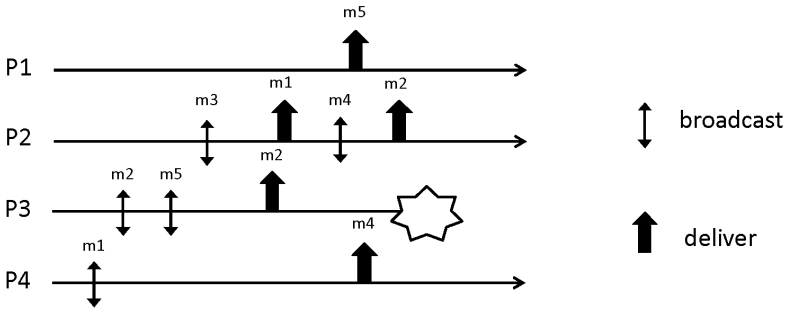
Processes may fail by crashing, but they are equipped with a perfect oracle that notifies at each process the new neighbour (when one of the two fails) through the following primitives:

- **Left_neighbour(p_x)**: process px is the new left neighbour of pi
- **Right_neighbour(p_x)**: process px is the new right neighbour of pi

Both the events may return a NULL value in case pi becomes the first or the last process of the line.
Each process can communicate only with its neighbours.

Write the pseudo-code of an algorithm implementing a Leader Election primitive assuming that channels connecting two neighbour processes are perfect.

Ex 3: Consider the partial execution depicted in the Figure



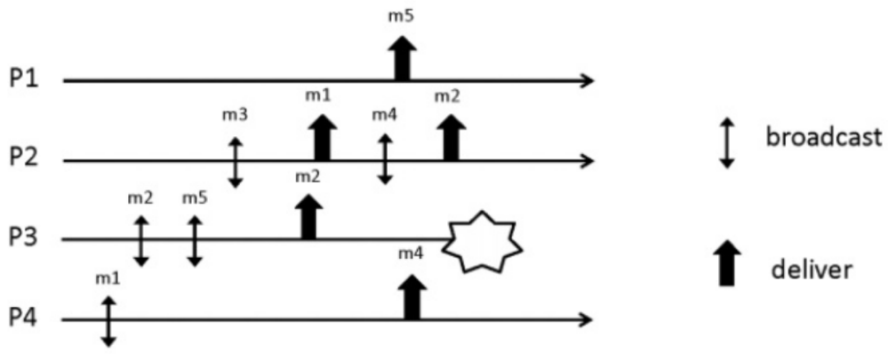
Answer to the following questions:

1. Complete the execution in order to have a run satisfying Uniform Reliable Broadcast.
2. Complete the execution in order to have a run satisfying Regular Reliable Broadcast but not Uniform Reliable Broadcast.
3. Complete the execution in order to have a run satisfying Best Effort Broadcast but not Regular Reliable Broadcast.

NOTE: In order to solve the exercise you can add broadcast, deliver and crash events but you cannot remove anything from the run.

Ex 4: Consider a distributed system composed by n processes {p1,p2,…,pn} identified through unique integer identifiers. Processes may communicate using perfect point-to-point links. Links are available between any pair of processes.
Processes may fail by crash and each process has access to a perfect failure detector.
Modify the algorithms implementing a distributed mutual exclusion abstraction discussed during the lectures to allow them to tolerate crash failures.

Ex 3: Consider the partial execution depicted in the Figure

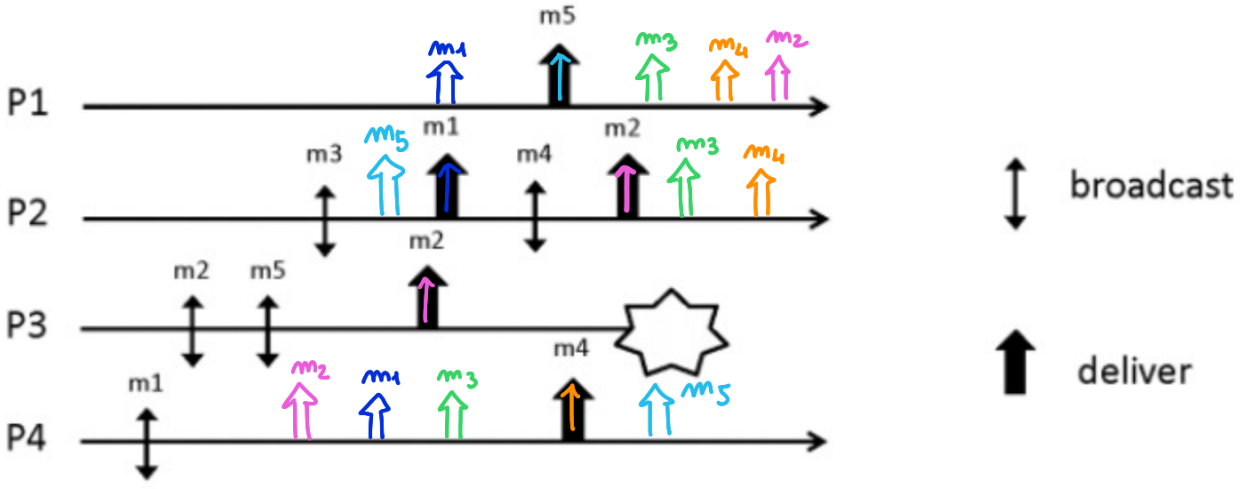


Answer to the following questions:

1. Complete the execution in order to have a run satisfying Uniform Reliable Broadcast.
2. Complete the execution in order to have a run satisfying Regular Reliable Broadcast but not Uniform Reliable Broadcast.
3. Complete the execution in order to have a run satisfying Best Effort Broadcast but not Regular Reliable Broadcast.

NOTE: In order to solve the exercise you can add broadcast, deliver and crash events but you cannot remove anything from the run.

① URB → uniform agreement

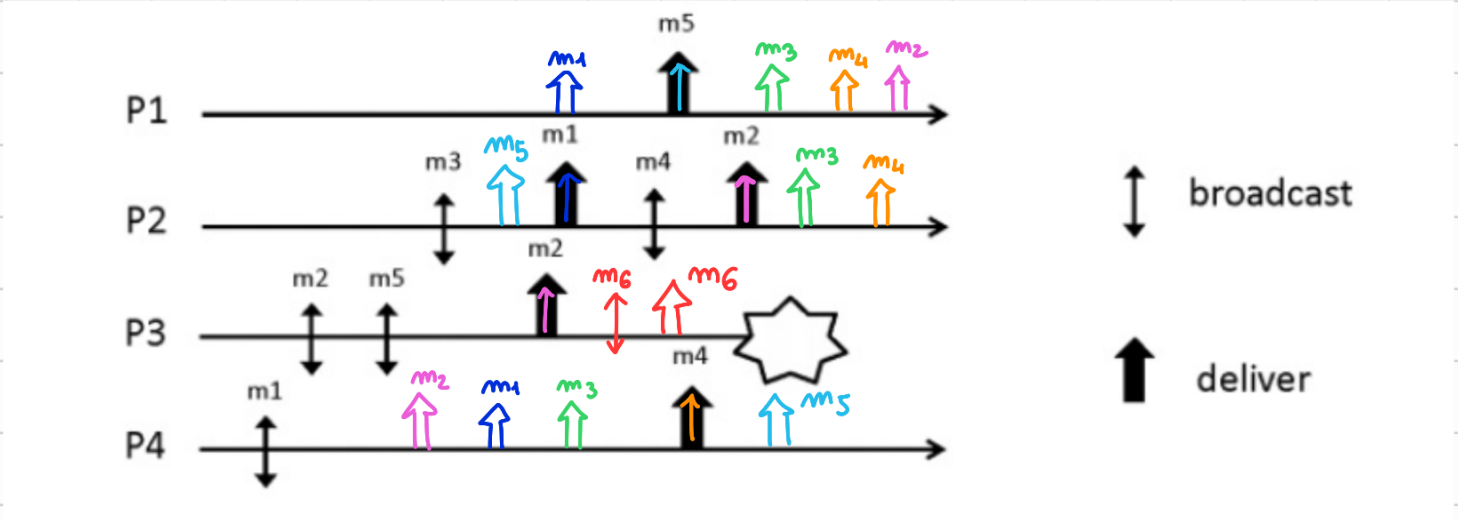


SUPER SET → m_4, m_1, m_2, m_5, m_3 (MANDATORY)

SET → m_2 (mandatory)

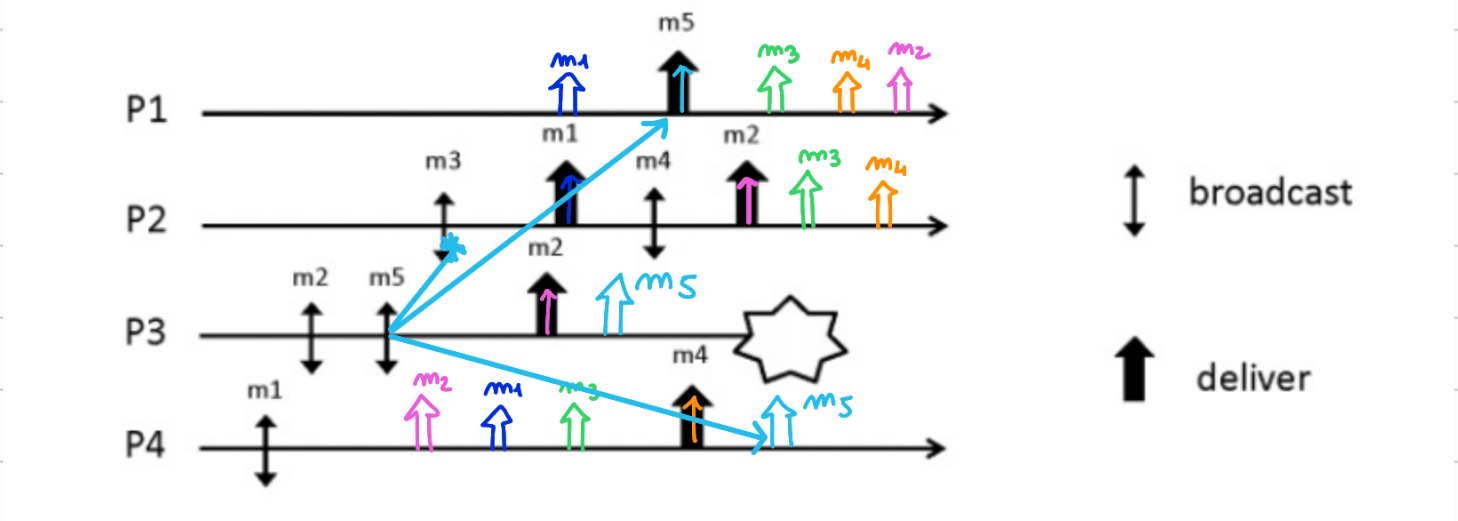
} same set for every correct process; every correct process delivered m

② RB but not URB

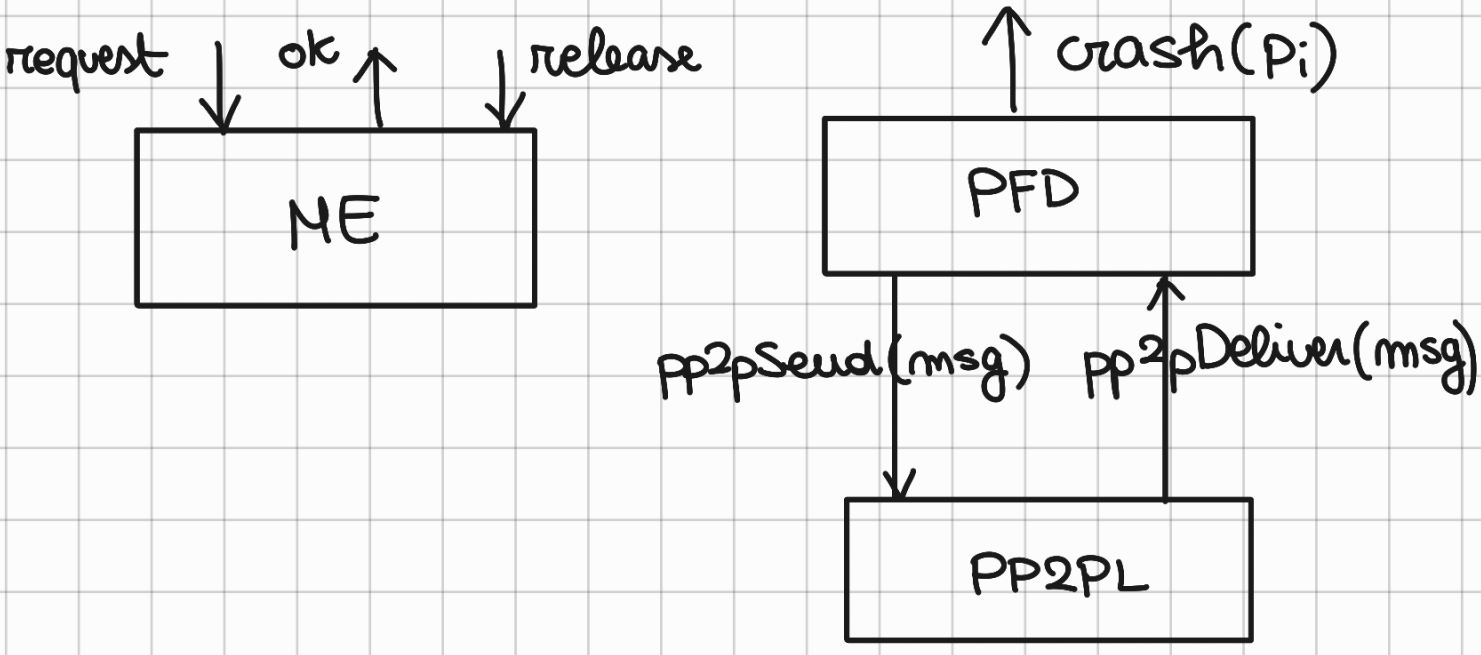


Every correct process have the same set : m_6 there isn't in the correct processes

③ BEB but not RB



Ex 4: Consider a distributed system composed by n processes {p1,p2,…,pn} identified through unique integer identifiers. Processes may communicate using perfect point-to-point links. Links are available between any pair of processes. Processes may fail by crash and each process has access to a perfect failure detector.
Modify the algorithms implementing a distributed mutual exclusion abstraction discussed during the lectures to allow them to tolerate crash failures.



1) INIT: - each p_n initializes its state
- coordinator C initializes its state

2) REQUEST: - when a p_n want to access to CS, sends a request to C ("I want to access to CS", ID)

3) MANAGE REQUEST BY C: - receives a request
- queque Q, storing request by p_n
  - empty → REPLY to p_n the permission to enter in CS
  - full → ADD p_n at the end of Q // waiting

4) REPLY ok() : - p_n enters in CS
- at the end release the CS notifing the C

5) RELEASE NOTIFICATION to C : - removing p_n from Q
- reply permission to the subsequent p_n

6) PERF. FAILURE DETECTOR : if the C crash, p_n re-elect a new C

7) CRASHES : a new C can be elected among p_n (ex. max ID)

Coordinator

init
$Q = \emptyset$
CS = ⊥
suspected_crash = $\emptyset$

upon event receive_request (REQ, n) from p_n
if p_n not in Q
$Q = Q \cup \{p_n\}$
trigger Permission CS to p_n

upon event receive_release (REL, n) from p_n
$Q = Q \setminus \{p_n\}$
if Q is not empty
next_p = first in Q
trigger Permission CS to next_p

```
upon event detect_crash of p_n
        suspected_crash = suspected_crash ∪ {p_n}
        Q = Q \ {p_n}
        if p_n is the next_p
            if Q is not empty
                next_p = first in Q
                trigger Permission CS to next_p

upon event crash_Coordinator
        Q = Q \ {all processes in queue}
        if new_coor. is not in suspected_crash
            new_coor = max rank (all processes)
            trigger new_coor as coordinator
```

Process

```
init
state = idle
C = getCoordinatorID()

upon event request_CS
        state = waiting
        trigger (REQ, n) to C

upon event permission_CS from C
        state = CS
        when action_CS are terminated
            trigger release_CS to C
```

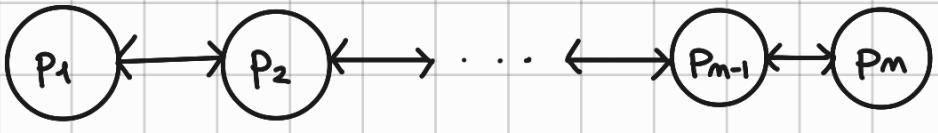
Ex 2: Consider a distributed system composed by n processes {p1,p2,…,pn} that communicate by exchanging messages on top of a line topology, where p1 and pn are respectively the first and the last process of the network.
Initially, each process knows only its left neighbour and its right neighbour (if they exist) and stores the respective identifiers in two local variables LEFT and RIGHT.

Processes may fail by crashing, but they are equipped with a perfect oracle that notifies at each process the new neighbour (when one of the two fails) through the following primitives:

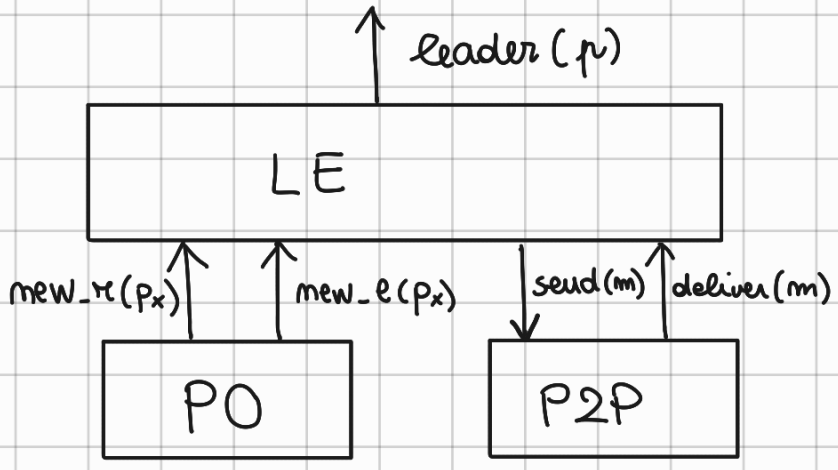
- **Left_neighbour(p_x):** process px is the new left neighbour of pi
- **Right_neighbour(p_x):** process px is the new right neighbour of pi

Both the events may return a NULL value in case pi becomes the first or the last process of the line.
Each process can communicate only with its neighbours.

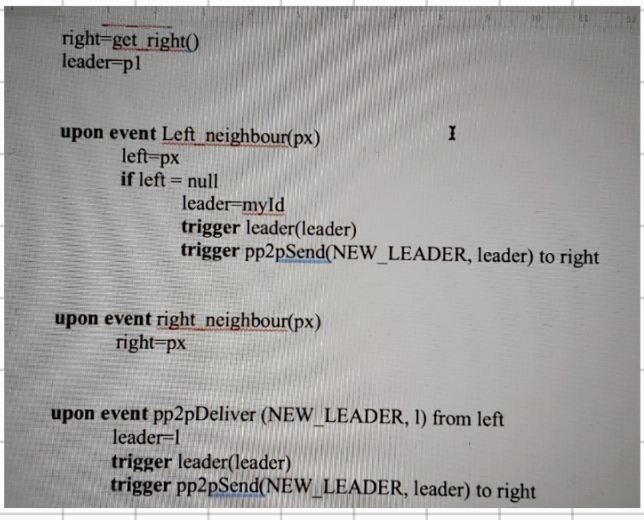
Write the pseudo-code of an algorithm implementing a Leader Election primitive assuming that channels connecting two neighbour processes are perfect.



live topology



upon event < le, INIT > do
    left = get_left() left neighbour with the current p
    right = get_right()
    leader = ⊥



```
right=get_right()
leader=p1


upon event Left_neighbour(px)
        left=px
        if left = null
                leader=myId
                trigger leader(leader)
                trigger pp2pSend(NEW_LEADER, leader) to right


upon event right_neighbour(px)
        right=px


upon event pp2pDeliver (NEW_LEADER, l) from left
        leader=l
        trigger leader(leader)
        trigger pp2pSend(NEW_LEADER, leader) to right
```