Dependable Distributed Systems
Master of Science in Engineering in Computer Science

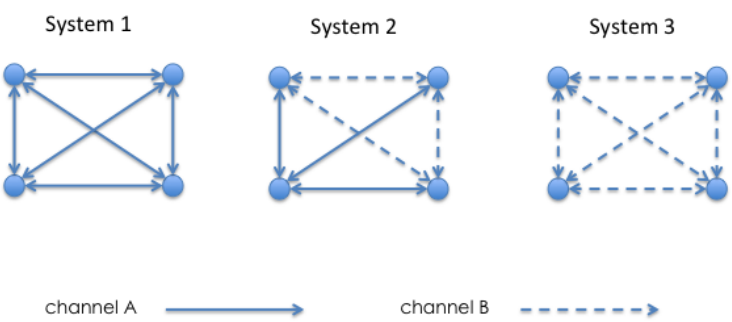
AA 2024/2025

Lecture 10 – 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 p_i 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 when 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.

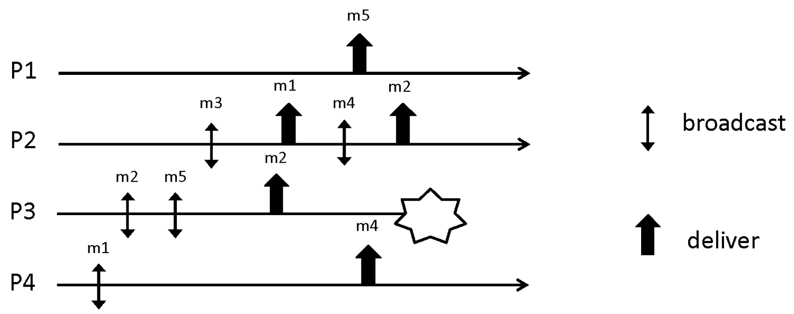
Ex 2: Consider a distributed system composed by n processes {p1,p2,…,pn} that communicate by exchanging messages on top of a line topology, where p1 and pn are respectively the first and the last process of the network.
Initially, each process knows only its left neighborsand its right neighbor (if they 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 neighbor (when one of the two fails) through the following primitives:

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

Write the pseudo-code of an algorithm implementing a Leader Election primitive assuming that channels connecting two neighbor 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.

<u>**NOTE:**</u> 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.

Leader based Algorithm

PROCESS CODE

Init
state = idle
coordinator = getCoordinatorId()
correct = {p1, p2, … pn}
transition = false

```
upon event request()
    state = waiting
    trigger pp2pSend(REQ, i) to coordinator

upon event pp2pDeliver(GRANT_CS)
    state = CS
    trigger ok()

upon event release()
    state = idle
    trigger pp2pSend(REL, i) to coordinator

upon event crash(pj)
        correct = correct \{pj}
        if coordinator == pj
                coordinator = getCoordinatorId(correct)
                if coordinator == self
                        % manage transition
                        transition = true
                        pending = ∅
                        current_in_CS=⊥
                        ACK=NACK=∅
                        for every pk ∈ correct
                                trigger pp2pSend(CHECK_CS) to pk
                else
                        if state == waiting
                                trigger pp2pSend(REQ, i) to coordinator


upon event pp2pDeliver(CHECK_CS) from coordinator
        if state== CS
                trigger pp2pSend(ACK, self) to coordinator
        else
                trigger pp2pSend(NACK, self) to coordinator
```

COORDINATOR CODE

Init
pending = ∅

current_in_CS=$\perp$
correct = {p1, p2, … pn}
transition = false

upon event pp2pDeliver(REQ, j) from p_j
    pending = pending $\cup$ {p_j}

when pending $\neq \emptyset$ and current_in_CS=$\perp$ and not transition
    candidate=select_process (pending)
    pending = pending \ candidate
    current_in_CS = candidate
    trigger pp2pSend(GRANT_CS) to candidate

upon event pp2pDeliver(REL, j) from p_j
    if current_in_CS == j
        current_in_CS = $\perp$

upon event crash(pj)
    correct = correct \{pj}
    if pj $\in$ pending
        pending = pending \{pj}
    if current_in_CS= pj
        current_in_CS=$\perp$

upon event pp2pDeliver (ACK, j)
    current_in_CS = j
    transition= false

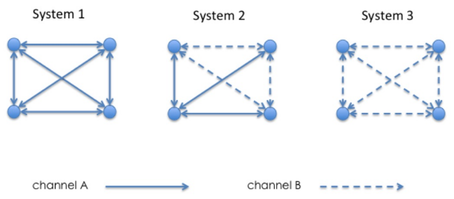
upon event pp2pDeliver (NACK, j)
    NACK= NACK $\cup$ {j}

when correct $\subseteq$ NACK
    transition = false

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 p_i 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 when 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.

1) YES BECAUSE AT LEAST ONE PROCESS RECEIVE ALWAYS THE RIGHT INFORMATION.

```
INIT
  ALIVE = Π
  DETECTED = ∅
  TIMER = Δ
  PHASE = 1
  STARTTIMER (TIMER)

UPON EVENT < TIMEOUT AND PHASE == 1 > DO
  FORALL p ∈ Π DO
    TRIGGER SEND (LIST, ALIVE) TO p
  PHASE == 2
  STARTTIMER (TIMER)

UPON EVENT < TIMEOUT AND PHASE == 2 > DO
  FORALL p ∈ Π DO
    IF (p ∉ ALIVE ∧ p ∉ DETECTED) THEN
      DETECTED = DETECTED ∪ {p}
      TRIGGER < CRASH | p >
    TRIGGER < SEND | [HBRQ] >
  ALIVE = ∅
  PHASE = 1
  STARTTIMER (TIMER)

UPON EVENT  DELIVER (HBRQ) FROM q
  TRIGGER  SEND (HBRP) TO q

UPON EVENT  DELIVER (HBRP) FROM q
  ALIVE = ALIVE ∪ {q}

UPON EVENT  DELIVER (LIST, ALIVE.LIST)
  ALIVE = ALIVE ∪ {ALIVE.LIST}
```

2) NO, BECAUSE IN THIS CASE WE CANNOT HAVE A PROCESS THAT HAS THE COMPLETE VIEW OF THE ALIVE PROCESSES. SO WE CAN DO IT UNTIL THE PROCESS P_1 REMAIN CORRECT.

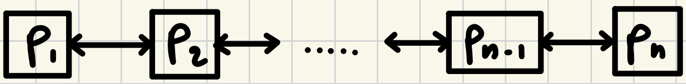
3) NO, BECAUSE IN THIS CASE THE SYSTEM IS COMPOSED ONLY BY CHANNEL B THAT IS FAIR LOSS LINK, AND WE CANNOT IMPLEMENT PFD OVER FAIR LOSS

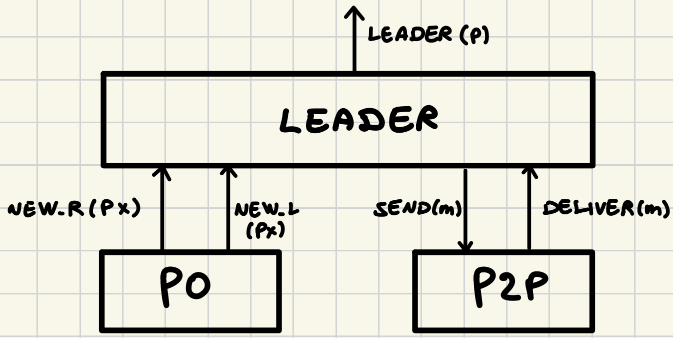
Ex 2: Consider a distributed system composed by n processes {p1,p2,…,pn} that communicate by exchanging messages on top of a line topology, where p1 and pn are respectively the first and the last process of the network.
Initially, each process knows only its left neighborsand its right neighbor (if they 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 neighbor (when one of the two fails) through the following primitives:

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

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





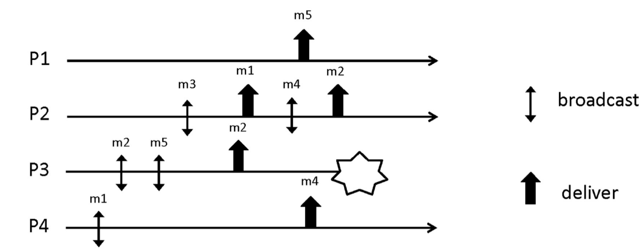

```
UPON EVENT <le, INIT> DO
    LEFT : GET_LEFT()
    RIGHT : GET_RIGHT()
    LEADER = P1

UPON EVENT LEFT_NEIGHBOUR (Px)
    LEFT = Px
    IF LEFT = NULL
        LEADER = MY.ID
        TRIGGER LEADER (LEADER)
        TRIGGER PP2PSEND (NEW.LEADER, LEADER) TO RIGHT

UPON EVENT RIGHT_NEIGHBOUR (Px)
    RIGHT = Px

UPON EVENT PP2PDELIVER (NEW.LEADER, Px) FROM LEFT
    LEADER : Px
    TRIGGER LEADER (LEADER)
    TRIGGER PP2PSEND (NEW.LEADER, LEADER) TO RIGHT
```

Ex 3: Consider the partial execution depicted in the Figure

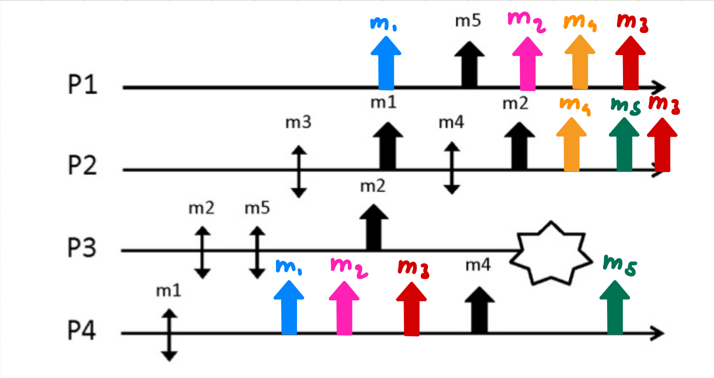


Answer to the following questions:

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.

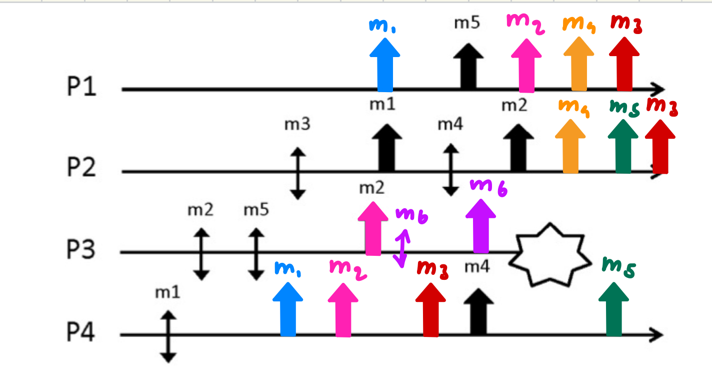
NOTE: In order to solve the exercise, you can add broadcast, deliver and crash events but you cannot remove anything from the run.

1) URB



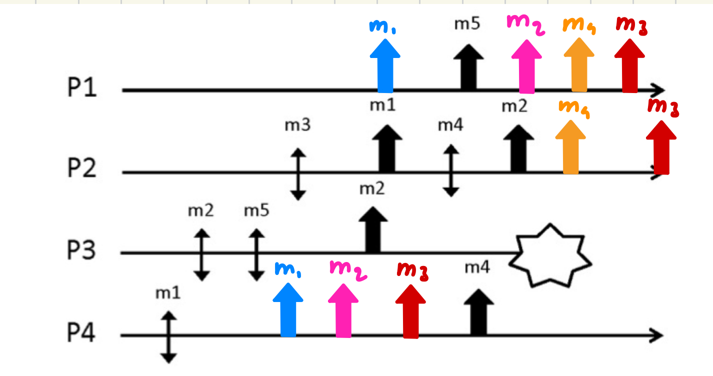
ALL CORRECT PROCESSES MUST DELIVER EVERY MESSAGE THAT ANOTHER PROCESS (CORRECT OR FAILED) HAS DELIVERED.

2) RB BUT NO URB



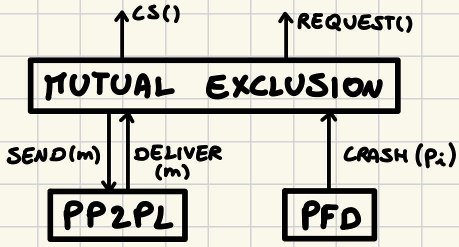
P_1, P_2 AND P_4 DON'T DELIVER m6 BECAUSE IT'S ONLY DELIVERED BY P3, WHICH IS A CRASHED PROCESS.

3) BEB BUT NO RB



m_5 NO AGREEMENT

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.




```
INIT
  ID ← IDENTIFIER
  PENDING = ∅
  REPLIES = ∅
  CORRECT = π
  STATE = NCS
  LAST.REQ = 0

UPON EVENT <ME, REQUEST> DO
  STATE = REQUESTING
  LAST.REQ = ID
  FORALL q ∈ π DO
    TRIGGER <PP2PL, SEND|REQUEST, ID>

UPON EVENT <PP2PL, DELIVER|REQUEST, q>
  IF(STATE == CS) OR (STATE == REQUESTING ∧ LAST.REQ ≠ ID)
    PENDING = PENDING ∪ {q}
  ELSE
    TRIGGER <PP2PL, SEND|[REPLAY]>

UPON EVENT <PP2PL, DELIVER|[REPLAY], q>
  REPLIES = REPLIES ∪ {q}

UPON EVENT <P, CRASH|p>
  CORRECT = CORRECT \ {p}

WHEN REPLIES == CORRECT DO
  STATE = CS
  TRIGGER <ME, CS>
  FORALL q ∈ PENDING
    TRIGGER <PP2PL, SEND|[REPLAY]> TO q
  STATE = NCS
  REPLIES = ∅
```