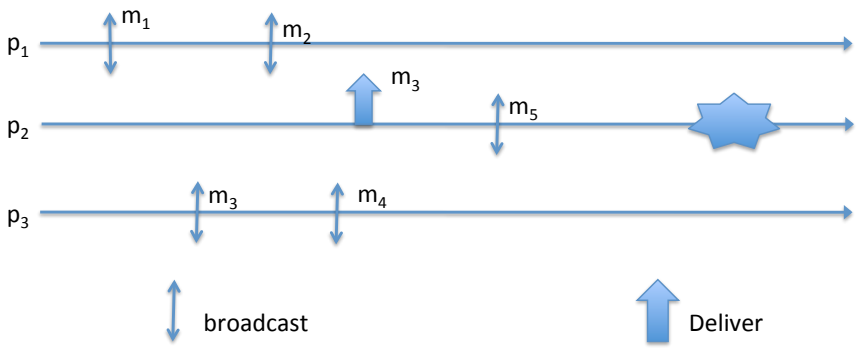
Dependable Distributed Systems
Master of Science in Engineering in Computer Science

AA 2023/2024

Lecture 10 – Exercises
October 18th, 2023

Ex 1: Consider the partial execution depicted in the following figure:



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

Ex 2: Consider a distributed system composed by n processes $\{p_1,p_2,\ldots,p_n\}$. Each process is connected to all the others through fair-loss point-to-point links and has access to a perfect failure detector.

Write the pseudo-code of an algorithm implementing a Uniform Reliable Broadcast primitive.

Given the system model described here, additionally answer to the following questions:

1. Is it possible to provide a quiescent implementation of the Uniform Reliable Broadcast primitive (a quiescent implementation is one where all processes eventually stop sending messages)?

2. Is it possible to provide an implementation that uses only data structure with finite size?

Ex 3: Consider a distributed system composed by N servers $\{s_1, s_2, \ldots s_n\}$ and M clients $\{c_1, c_2, \ldots c_m\}$.
Each client c_i runs its algorithm and it can request to servers the execution of a particular task T_i. A Server will execute the task T_i and, after that, a notification will be sent to c_i that T_i has been completed.
The Figure shows the code executed by a generic client c_i.

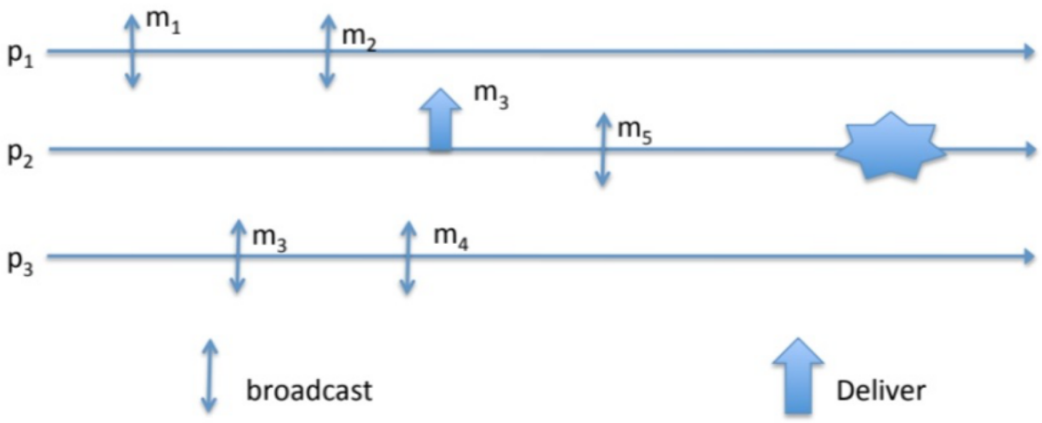
Operation executeTask (T_i)	Upon pp2pdeliver (TASK_COMPLETED, T_i) from s_j
1. For each $s_i \in \{s_1, s_2, \ldots s_n\}$ 2. pp2psend (TASK_REQ, T_i, c_i) to s_i;	1. **trigger** completedTask (T_i);

Write the pseudo-code of an algorithm, executed by servers, able to allocate tasks assuming that:

- Once clients ask for a task execution, they remain blocked until the task is not terminated.
- Any two clients c_i and c_j can concurrently require the execution of two different tasks T_i and T_j;
- Each task is univocally identified by the pair (T_i, c_i);
- Each server can manage at most one task at every time;
- At most N-1 servers can crash;
- If a server crashes while executing a task, such task needs to be re-allocated and re-processed by a different server;
- Servers have access to a uniform consensus primitive;
- Servers have access to a perfect failure detector P;
- Servers communicate through a uniform reliable broadcast primitive.

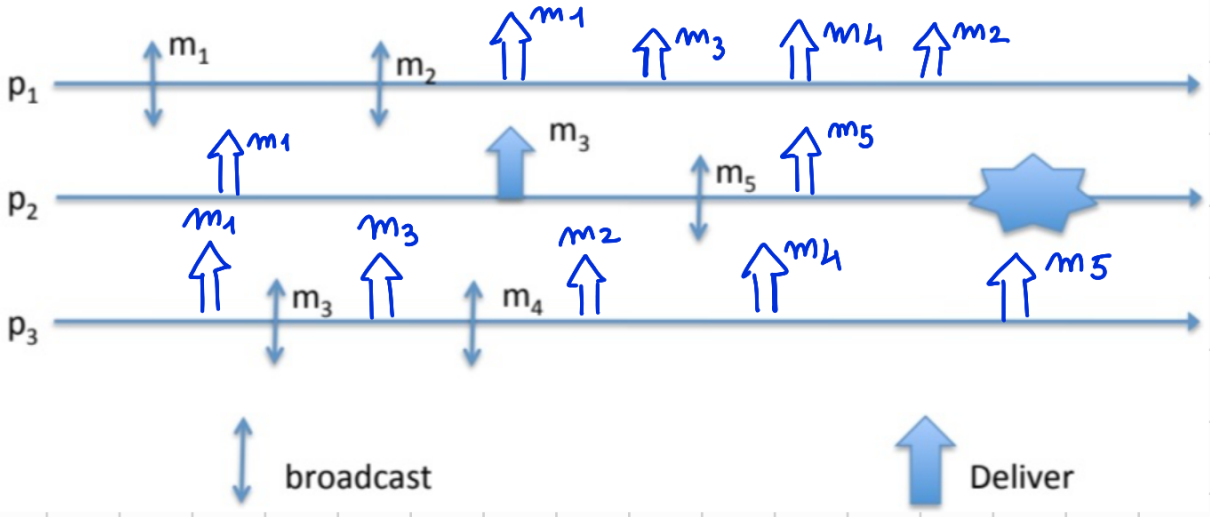
Ex 4: Consider a distributed system formed by n processes p_1, p_2, …, p_n connected along a ring i.e., a process p_i is initially connected to a process $p_{(i+1) \bmod n}$ through a unidirectional perfect point-to-point link.
Write the pseudo-code of a distributed algorithm implementing a consensus primitive.

Ex 1: Consider the partial execution depicted in the following figure:



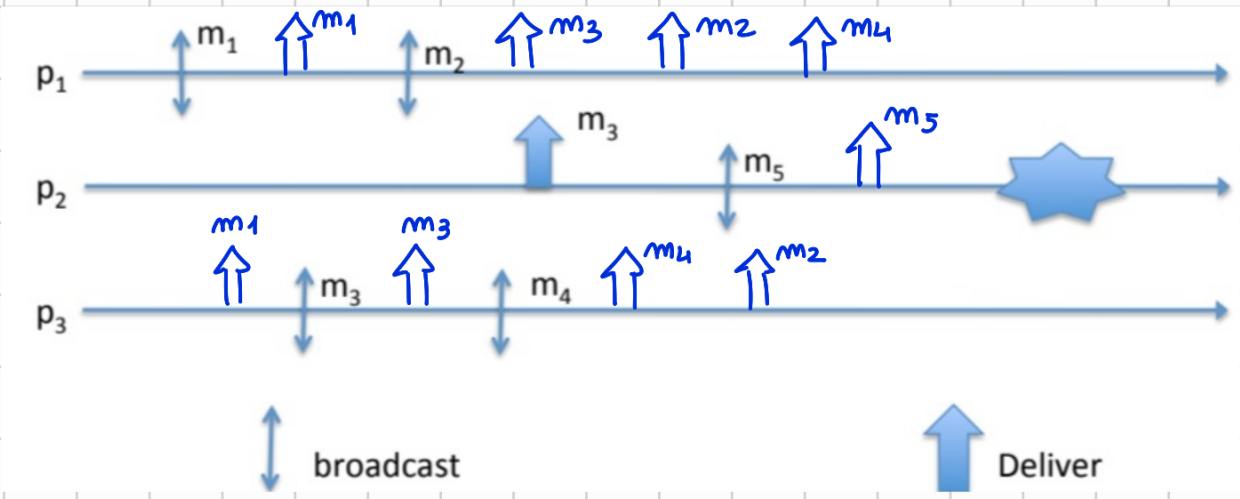
1. Complete the execution in order to obtain a run satisfying *Best Effort Broadcast* but *not Reliable Broadcast.*
2. Complete the execution in order to obtain a run satisfying *Regular Reliable Broadcast* but *not Uniform Reliable Broadcast.*
3. Complete the execution in order to obtain a run satisfying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

① BEB not RB



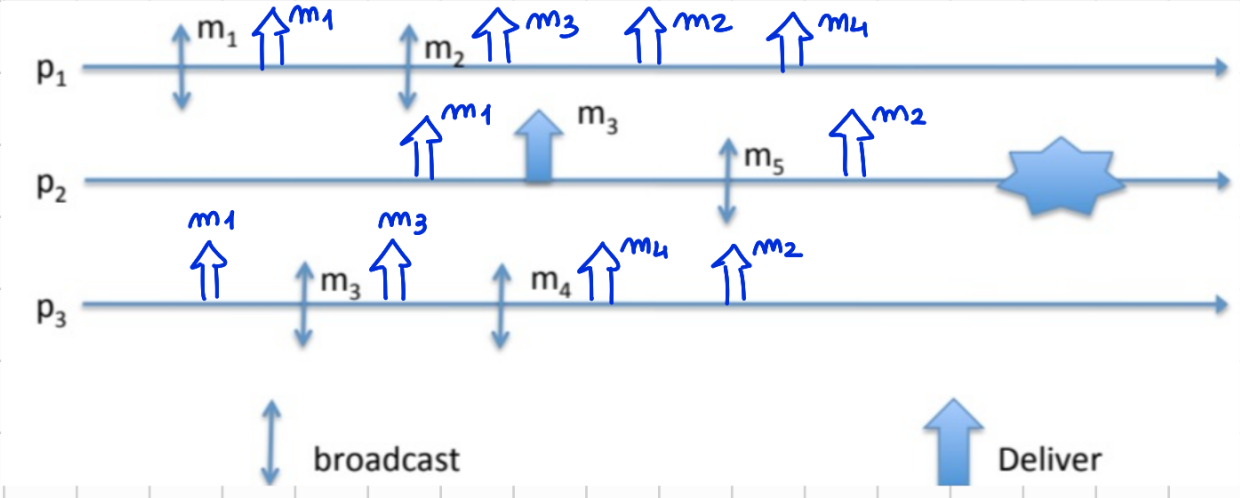
Correct processes not same set

② RB not URB



RB : every correct process same set
not URB : faulty not super set of correct

③ URB



Set : m_1, m_2, m_3, m_4 Correct

m_1, m_2, m_3 Faulty

Ex 2: Consider a distributed system composed by n processes $\{p_1,p_2,\ldots,p_n\}$. Each process is connected to all the others through fair-loss point-to-point links and has access to a perfect failure detector.

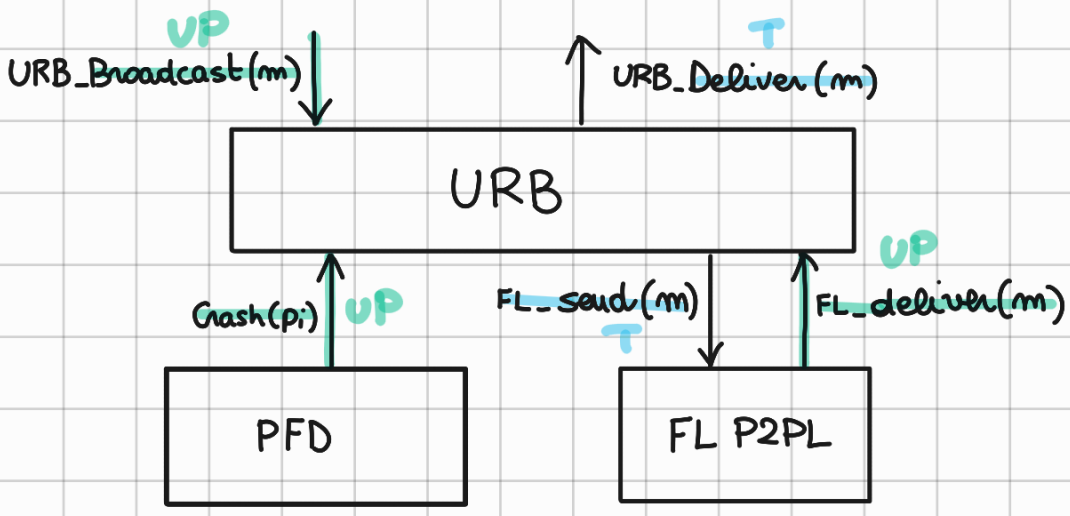
Write the pseudo-code of an algorithm implementing a Uniform Reliable Broadcast primitive.

Given the system model described here, additionally answer to the following questions:

1. Is it possible to provide a quiescent implementation of the Uniform Reliable Broadcast primitive (a quiescent implementation is one where all processes eventually stop sending messages)?

2. Is it possible to provide an implementation that uses only data structure with finite size?

- Fair-loss P2PL
- PFD



upon event < URB, INIT > do
    pending = ∅
    delivered = ∅
    correct = Π
    ack_m = ∅
    starttimer(Δ)

upon event ⟨ URB, URB_Broadcast | m ⟩ do
    pending = pending ∪ { m }
    for every $p_i \in \Pi$ do
        trigger ⟨ fl, FL_send | (MSG, m, p_i) ⟩ to p_j

upon event ⟨ fl, FL_deliver | (MSG, m, p_i) ⟩ from p_j do
    $ack_m = ack_m \cup \{ p_j \}$
    for every $p_i \in \Pi$ do
        trigger ⟨ fl, FL_send (MSG, m, p_i) ⟩ to p_j

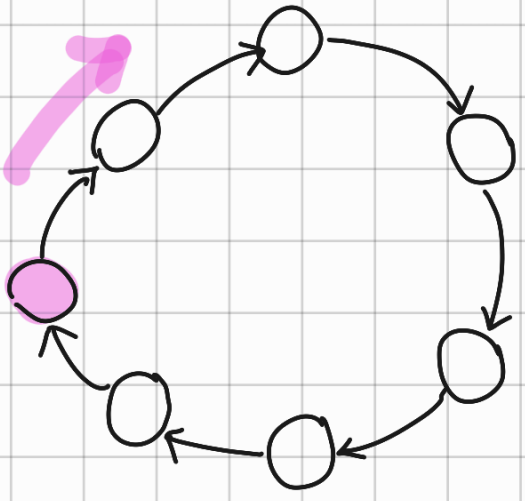
upon event ⟨ PFD, Crash | p_j ⟩ do
    correct = correct $\setminus \{ p_j \}$

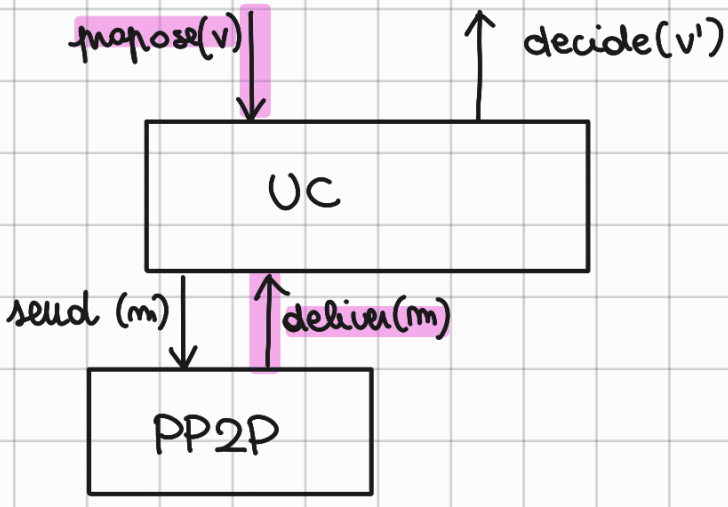
upon exist $m \in$ pending such that correct $\subseteq ack_m \wedge$
        $m \notin$ delivered do
    delivered = delivered $\cup \{ m \}$
    trigger ⟨ URB, URB_Deliver | m ⟩

upon ⟨ Timeout ⟩
    forall $m \in$ pending do
        forall $p_j \in \Pi$ do
            trigger ⟨ fl, FL_send (MSG, m, p_i) ⟩ to p_j
    start timer (Δ)

Ex 4: Consider a distributed system formed by n processes p_1, p_2, …, p_n connected along a ring i.e., a process p_i is initially connected to a process $p_{(i+1) \bmod n}$ through a unidirectional perfect point-to-point link.
Write the pseudo-code of a distributed algorithm implementing a consensus primitive.

- $p_{(i+1) \bmod n}$
- PP2PL





We have several processes connected to the next process and a process can communicate just to the $p_{(i+1) \bmod n}$. Assuming that processes are not going to fail, each of process start with a value v and they have to exchange this value v. Processes have to send the value proposed to other. At first we start a round in the ring, the p_i starts to propose the value v to the next and it is added to proposal set. At the end when p_i receive the proposal set full of value v, can check if there are all the value. Then there is the decide part for a value.

```
upon event < c, INIT > do
    next = p(i+1) mod n
    decision = ⊥
    proposal_set = ∅
    alive = Π

upon event < c, PROPOSE | v > do
    proposal = v
    proposal_set = proposal_set ∪ {v}
    trigger < pp2pl, pp2pl_SEND | (PROPOSAL,
                                   proposal_set,
                                   p_i) > to next

upon event < pp2pl, pp2pl_DELIVER | (PROPOSAL,
        (check if I have the full set of propose)   proposal_set,
                                                    p_t) > from p_J
    if p_t = p_i then (1° process?)
        if alive ⊆ proposal_set ∧ decision = ⊥ do
            decision = min(proposal_set)
            trigger < c, DECIDE | decision >
    else (process in the middle)
        proposal_set = ps ∪ ps   update the set
        trigger < pp2pl, pp2pl_SEND | (PROPOSAL,
                                       proposal_set,
                                       p_t) > to next
```