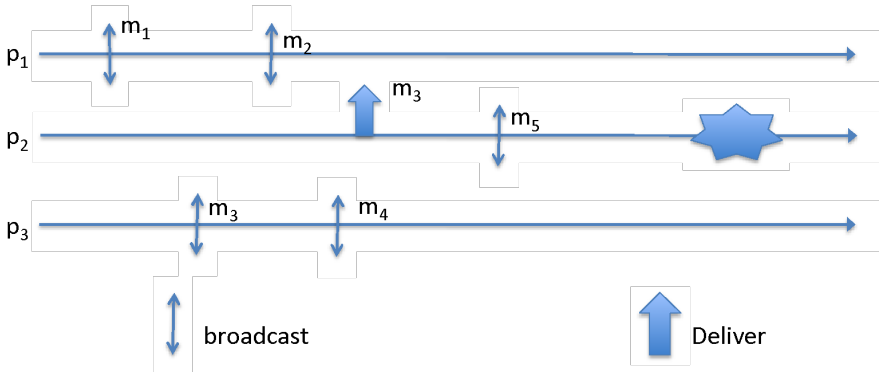
**Dependable Distributed Systems**
**Master of Science in Engineering in Computer Science**

**AA 2024/2024**

**Lecture 13 – Exercises**
**October 20th, 2024**

**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 $c_i$.

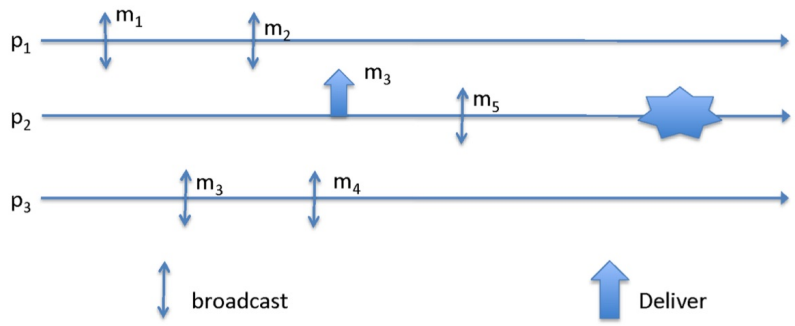
| Operation executeTask ($T_i$) | Upon pp2pdeliver (TASK_COMPLETED, $T_i$) from $s_j$ |
|---|---|
| 1. For each $s_i \in \{s_1, s_2, \ldots s_n\}$<br>2. pp2psend (TASK_REQ, $T_i$, $c_i$) to $s_i$; | 1. **trigger** completedTask ($T_i$); |

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.

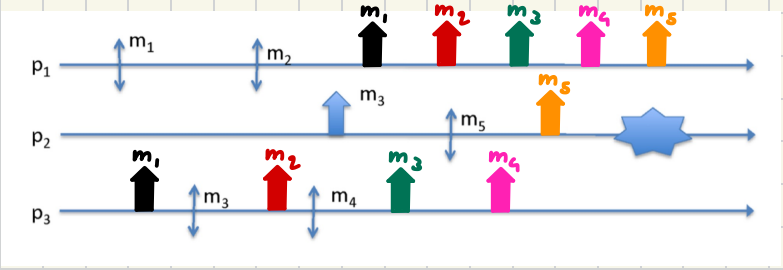
**Ex 4:** Consider a distributed system formed by n processes $p_1$, $p_2$, …, $p_n$ connected along a ring i.e., a process $p_i$ is initially connected to a process $p_{(i+1) \bmod n}$ through a unidirectional perfect point-to-point link.
Write the pseudo-code of a distributed algorithm implementing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

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

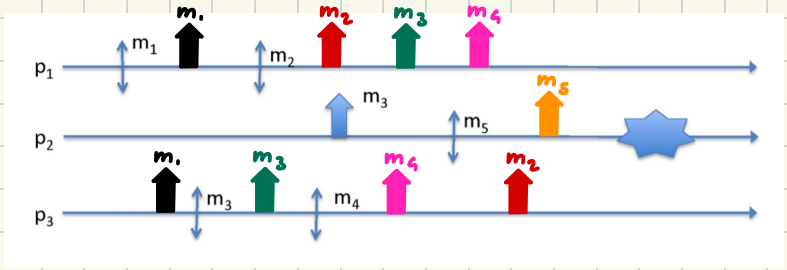



1. Complete the execution in order to obtain a run satisfying *Best Effort Broadcast* but *not Reliable Broadcast.*
2. Complete the execution in order to obtain a run satisfying *Regular Reliable Broadcast* but *not Uniform Reliable Broadcast.*
3. Complete the execution in order to obtain a run satisfying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

## 1) BEB BUT NOT RB

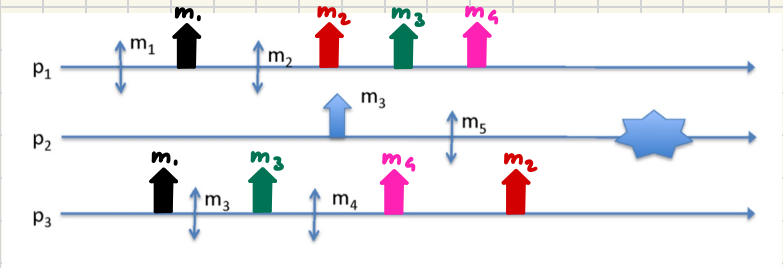



## 2) RRB BUT NOT URB





## 3) URB



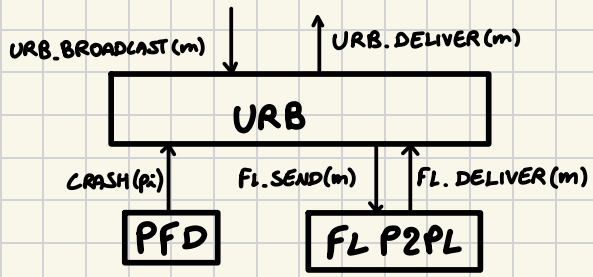
**Ex 2:** Consider a distributed system composed by n processes $\{p_1,p_2,\ldots,p_n\}$. Each process is connected to all the others through fair-loss point-to-point links and has access to a perfect failure detector.

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?



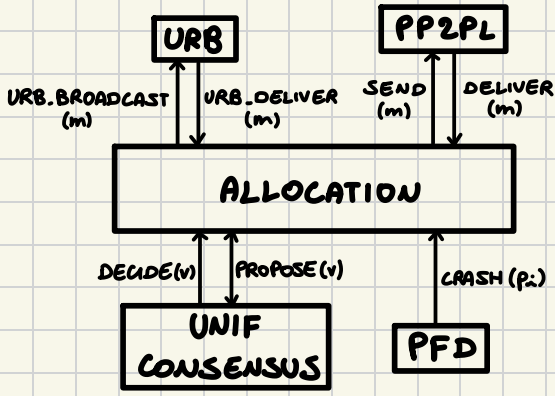

```
UPON EVENT < URB, INIT > DO
    PENDING = ∅
    DELIVERED = ∅
    CORRECT = Π
    FORALL m DO ACK[m] = ∅
    STARTTIMER (Δ)

UPON EVENT < URB, BROADCAST | m > DO
    PENDING = PENDING U {m}
    FORALL pi IN Π DO
        TRIGGER < fl, FL_SEND | (MSG, m, pi) > TO pj

UPON EVENT < fl, FL_DELIVER | (MSG, m, pi) > FROM pj DO
    ACK[m] = ACK[m] U {pj}
    FORALL pi IN Π DO
        TRIGGER < fl, FL_SEND | (MSG, m, pi) > TO pj

UPON EVENT < PFD, CRASH | pj > DO
    CORRECT = CORRECT \ {pj}

UPON EXIST m IN PENDING SUCH THAT CORRECT ⊆ ACK[m] ∧ m NOT IN DELIVERED DO
    DELIVERED = DELIVERED U {m}
    TRIGGER < URB, DELIVER | m >

UPON < TIMEOUT >
    FORALL m IN PENDING DO
        FORALL pj IN Π DO
            TRIGGER < fl, FL_SEND | (MSG, m, pi) > TO pj
    STARTTIMER (Δ)
```

**Ex 3:** Consider a distributed system composed by N servers $\{s_1, s_2, \dots s_n\}$ and M clients $\{c_1, c_2, \dots c_m\}$.
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 $s_i \in \{s_1, s_2, \dots s_n\}$<br>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.




**CLIENT:**

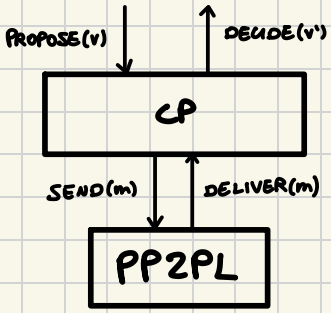
```
OPERATION EXECUTETASK (Ti)
  FOR EACH Si ∈ {S1 ... Sn}
    PP2PL.SEND (TASK.REQ, Ti, ci) TO Si

UPON PP2PL.DELIVER (TASK_COMPLETED, Ti) FROM Sj
  TRIGGER COMPLETEDTASK (Ti)
```

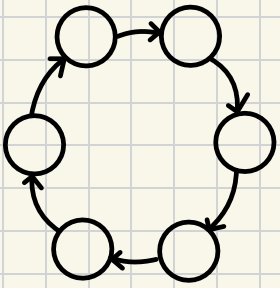
**SERVER:**

```
INIT
  PENDING = ϕ
  BUSY = FALSE
  CURRENT_ALLOCATION = ϕ

UPON EVENT < PP2PL.DELIVER (TASK.REQ, Ti, ci) > FROM ci
  IF <Ti , ci> NOT IN CURRENT_ALLOCATION
    PENDING = PENDING U {(Ti, ci)}

WHEN PENDING != ϕ
  IF !BUSY
    CANDIDATE = SELECT_FROM (PENDING)
    TRIGGER PROPOSE (< ID, CANDIDATE >)
  ELSE
    TRIGGER PROPOSE (< ID, NULL >)

UPON EVENT CRASH (S)
  IF THERE EXISTS <ID, TASK > IN CURRENT_ALLOCATION SUCH THAT S == ID
    CURRENT_ALLOCATION = CURRENT_ALLOCATION \ {ID, TASK}
    PENDING = PENDING U {(Ti, ci)}

UPON EVENT DECIDE (ID, TASK)
  CURRENT_ALLOCATION = CURRENT_ALLOCATION U {ID, TASK}
  IF ID == MY_ID AND TASK != NULL
    BUSY = TRUE
    EXECUTE (Ti)
    BUSY = FALSE
    TRIGGER < PP2PL_SEND (TASK_COMPLETED, Ti) > TO ci
    TRIGGER < URB_BROADCAST (TASK_COMPLETED, Ti, ci)

UPON EVENT URB_DELIVER (TASK_COMPLETED, Ti, ci)
  CURRENT_ALLOCATION = CURRENT_ALLOCATION \ {Ti , ci}
  PENDING = PENDING \ {Ti , ci}
```

**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.

$P_{(i+1) \bmod n}$



```
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, Pi) > TO NEXT

UPON EVENT < PP2PL, PP2PL.DELIVER | (PROPOSAL, PROPOSAL_SET, Pr) > FROM Pj
    IF Pr = Pi THEN
        IF ALIVE ⊆ PROPOSAL_SET ∧ DECISION = ⊥ DO
            DECISION = min(PROPOSAL_SET)
            TRIGGER <C, DECIDE | DECISION>
    ELSE
        PROPOSAL_SET = Ps ∪ Ps
        TRIGGER < PP2PL, PP2PL.SEND | (PROPOSAL, PROPOSAL_SET, Pr) > TO NEXT
```