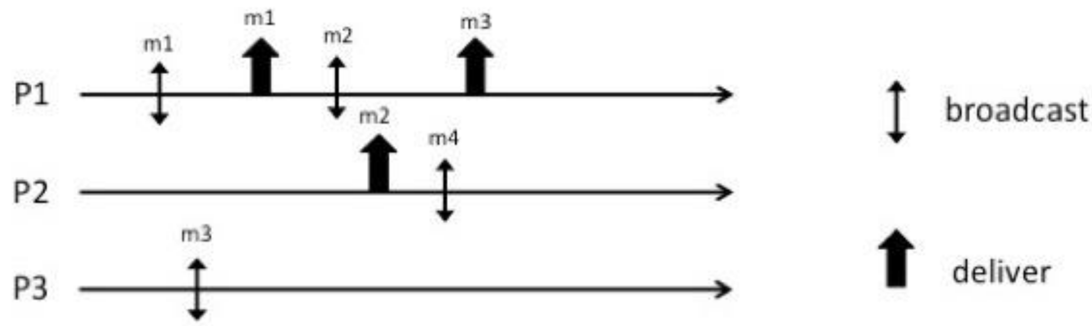
Dependable Distributed Systems
Master of Science in Engineering in Computer Science

AA 2023/2024

Lecture 13 – Exercises
October 25th, 2023

Ex 1: Let us consider the following partial execution



Answer the following points:

1. Provide all the possible sequences satisfying Causal Order
2. Complete the execution to have a run satisfying FIFO order but not causal order

Ex 2: Consider a distributed system constituted by n processes $\prod=\{p_1, p_2 \ldots p_n\}$ with unique identifiers that exchange messages through FIFO perfect point-to-point links and are structured through a line (i.e., each process p_i can exchange messages only with processes p_{i-1} and p_{i+1} when they exists). Processes may crash and each process is equipped with a perfect oracle (having the interface *new_right(p)* and *new_left(p)*) reporting a new neighbor when the previous one is failing.
Write the pseudo-code of an algorithm implementing a Perfect failure detector primitive.

Ex 3: Consider a distributed system constituted by n processes $\prod=\{p_1, p_2 \ldots p_n\}$ with unique identifiers that exchange messages through perfect point-to-point links and are structured through a ring (i.e., each process p_i can exchange messages only with processes and $p_{i+1(\text{mod } n)}$). Processes may crash and each process is equipped with a perfect oracle (having the interface *new_next(p)*) reporting a new neighbor when the previous one is failing.
Write the pseudo-code of an algorithm implementing a Uniform Reliable Broadcast communication primitive.

Ex 4: A transient failure is a failure that affects a process temporarily and that randomically alter the state of the process (i.e., when the process is affected by a transient failure, its local variables assume a random value).
Let us consider a distributed system composed by *N* processes where f_c processes can fail by crash and f_t processes can suffer transient failures between time t_0 and t_{stab}.

Let us consider the following algorithm implementing the Regular Reliable Broadcast specification

Algorithm 3.2: Lazy Reliable Broadcast

```
Implements:
    ReliableBroadcast, instance rb.

Uses:
    BestEffortBroadcast, instance beb;
    PerfectFailureDetector, instance P.

upon event ⟨ rb, Init ⟩ do
    correct := Π;
    from[p] := [∅]^N;

upon event ⟨ rb, Broadcast | m ⟩ do
    trigger ⟨ beb, Broadcast | [DATA, self, m] ⟩;

upon event ⟨ beb, Deliver | p, [DATA, s, m] ⟩ do
    if m ∉ from[s] then
        trigger ⟨ rb, Deliver | s, m ⟩;
        from[s] := from[s] ∪ {m};
        if s ∉ correct then
            trigger ⟨ beb, Broadcast | [DATA, s, m] ⟩;

upon event ⟨ P, Crash | p ⟩ do
    correct := correct \ {p};
    forall m ∈ from[p] do
        trigger ⟨ beb, Broadcast | [DATA, p, m] ⟩;
```

Answer to the following questions:

1. For every property of the Regular Reliable Broadcast specification, discuss if it is guaranteed between time t0 and tstab and provide a motivation for your answer.

2. For every property of the Regular Reliable Broadcast specification, discuss if it is eventually guaranteed after tstab and provide a motivation for your answer.

3. Assuming that the system is synchronous, explain if and how you can modify the algorithm (no pseudo-code required) to guarantee that No Duplication, Validity and Agreement properties will be eventually guaranteed after tstab.

Ex 5: Let us consider the following algorithm

```
upon event ⟨ frb, Init ⟩ do
   lsn := 0;
   pending := ∅;
   next := [1]^N ;

upon event ⟨ frb, Broadcast | m ⟩ do
for each p ∈Π do
   trigger ⟨ l, send | p, [DATA, self, m, lsn] ⟩;
lsn := lsn + 1;

upon event ⟨ l, deliver | p, [DATA, s, m, sn] ⟩ do
   pending := pending ∪ {(s, m, sn)};

while exists (s, m′, sn′) ∈ pending such that sn′ = next[s] do
   next[s] := next[s] + 1;
   pending := pending \ {(s, m′, sn′)};
  trigger ⟨frb, Deliver | s, m′⟩;
```

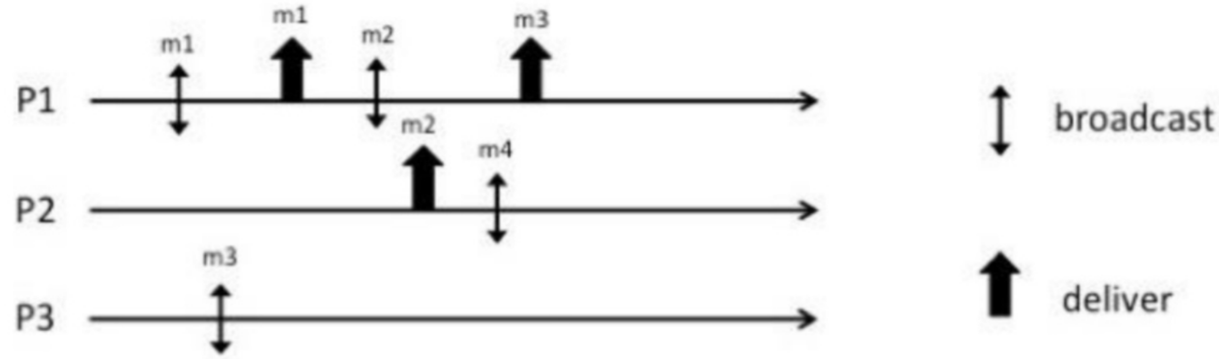
Let us consider the following properties:

- ***Validity:*** If a correct process p broadcasts a message m, then p eventually delivers m.
- ***No duplication***: No message is delivered more than once.
- ***No creation:*** If a process delivers a message m with sender s, then m was previously broadcast by process s.
- ***Agreement***: If a message m is delivered by some correct process, then m is eventually delivered by every correct process.
- ***FIFO delivery***: If some process broadcasts message m_1 before it broadcasts message m_2, then no correct process delivers m_2 unless it has already delivered m_1.

Assuming that every process may fail by crash, address the following points:

1. Considering that messages are sent by using *perfect point to point links*, for each property mentioned, discuss if it satisfied or not and provide a motivation for your answer;
2. Considering that messages are sent by using *fair loss links*, for each property mentioned, discuss if it satisfied or not and provide a motivation for your answer.

Ex 1: Let us consider the following partial execution



Answer the following points:

1. Provide all the possible sequences satisfying Causal Order
2. Complete the execution to have a run satisfying FIFO order but not causal order

① All sequences for CAUSAL ORDER

FIFO + LOCAL: $m_1 \to m_2$, $m_2 \to m_4$

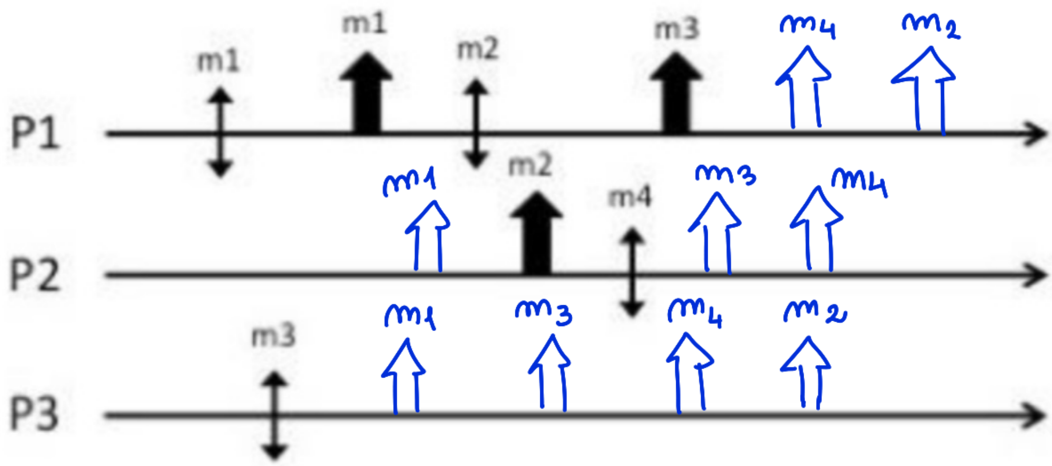
$m_1\ m_2\ m_3\ m_4$

$m_1\ m_3\ m_2\ m_4$

$m_1\ m_2\ m_4\ m_3$

$m_3\ m_1\ m_2\ m_4$

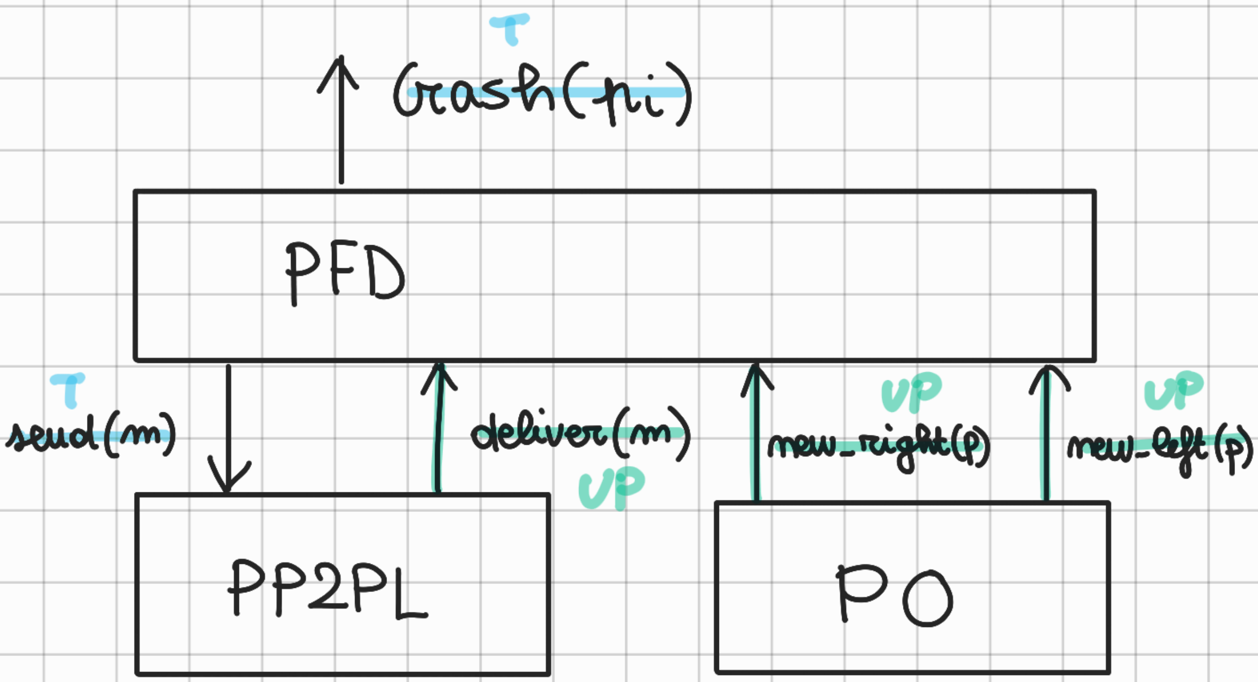
② FIFO not causal



FIFO: $m_1 \to m_2$ not CAUSAL: $m_2 \to m_4$

Ex 2: Consider a distributed system constituted by n processes $\prod=\{p_1, p_2\dots p_n\}$ with unique identifiers that exchange messages through FIFO perfect point-to-point links and are structured through a line (i.e., each process p_i can exchange messages only with processes p_{i-1} and p_{i+1} when they exists). Processes may crash and each process is equipped with a perfect oracle (having the interface *new_right(p)* and *new_left(p)*) reporting a new neighbor when the previous one is failing.
Write the pseudo-code of an algorithm implementing a Perfect failure detector primitive.

$\Pi = \{p_1, \dots, p_n\}$ FIFO PP2PL



upon event < PFD, Init > do
    alive = Π
    suspected := $\emptyset$ empty
    left := get_left()
    right := get_right()   } can be null
    starttimer (Δ) to check the failure

CRASH → I have to assign a new value to right

upon event <PO, new_right | p> do
  if right != null
    crash_r = right
    suspected := suspected ∪ { crash_r }
    trigger <PFD, Crash | crash_r>
   trigger <PP2PL, PP2P_send | [Crash, crash_r]> to right   are you alive?

upon event <PO, new_left | p> do
  if left != null
    crash_l = left
    suspected := suspected ∪ { crash_l }
    trigger <PFD, Crash | crash_l>
   trigger <PP2PL, PP2PL_send | [Crash, crash_l]> to left

upon event <PP2PL, PP2PL_deliver | [Crash, p]> do
 if suspected != ∅
  for each p_i in suspected
   right := new_right
   left := new_left
   trigger <PP2PL, PP2PL_send | [Crash] to left
   trigger <PP2PL, PP2PL_send | [Crash] to right