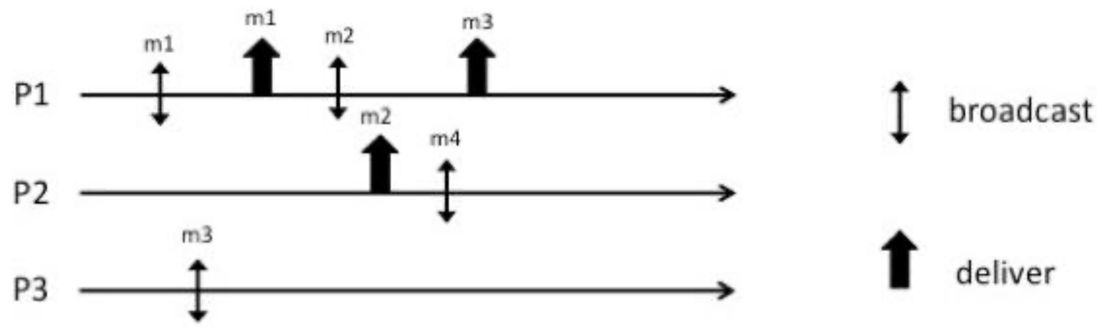
Dependable Distributed Systems
Master of Science in Engineering in Computer Science

AA 2024/2025

Lecture 16 – Exercises
November 6th, 2024

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 $\Pi=\{p_1, p_2\ldots p_n\}$ 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 $\Pi=\{p_1, p_2\ldots p_n\}$ 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  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′ ⟩;
```

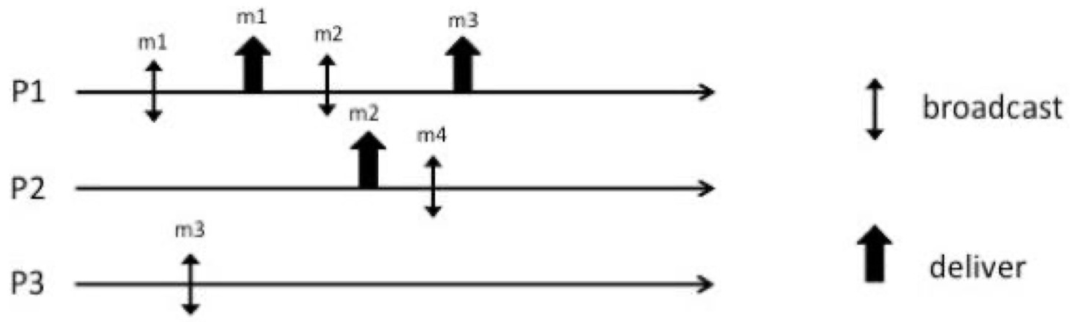
Let us consider the following properties:

- ***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



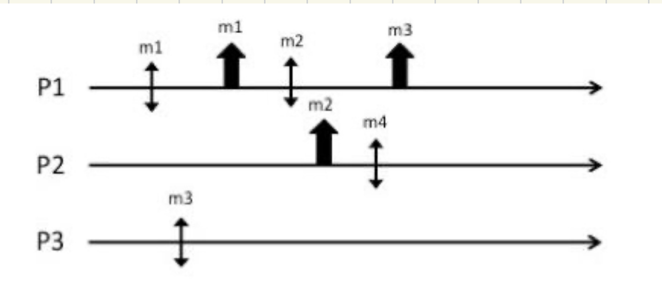
Answer the following points:

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

CASUAL ORDER = FIFO ORDER + LOCAL ORDER

LOCAL ORDER: IF A PROCESS DELIVERS A MESSAGE m_1 BEFORE SENDING A MESSAGE m_2, THEN NO CORRECT PROCESS DELIVER m_2 IF IT HAS NOT ALREADY DELIVERED m_1.

1)

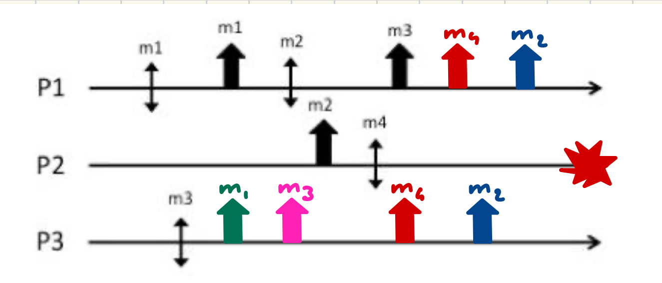


$m_1 \rightarrow m_2$ FIFO ORDER
$m_2 \rightarrow m_4$ LOCAL ORDER

POSSIBLE SEQUENCES.

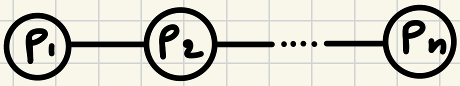
m_1, m_2, m_3, m_4
m_1, m_2, m_4, m_3
m_1, m_3, m_2, m_4
m_3, m_1, m_2, m_4 NOT FOR P_1

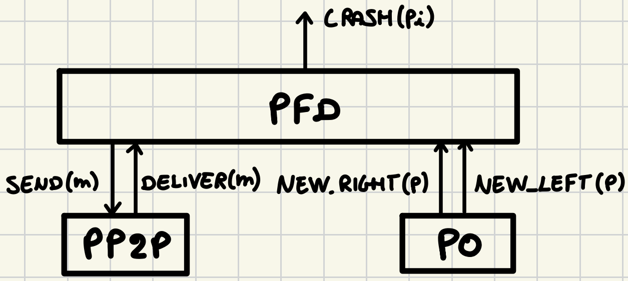
2)



$m_1 \rightarrow m_2$ FIFO ORDER
$\cancel{m_2 \rightarrow m_4}$ NOT CASUAL ORDER

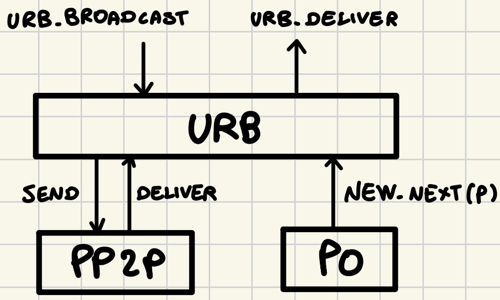
Ex 2: Consider a distributed system constituted by n processes $\Pi=\{p_1, p_2\dots p_n\}$ with unique identifiers that exchange messages through FIFO perfect point-to-point links and are structured through a line (i.e., each process p_i can exchange messages only with processes p_{i-1} and p_{i+1} when 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.

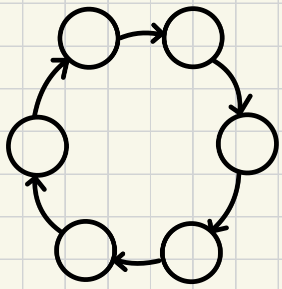





```
INIT
  ALIVE= Π
  SUSPECTED= ∅
  IF ID==1
    LEFT=NULL
  ELSE IF ID==n
    RIGHT=NULL
  ELSE
    LEFT = Pi-1
    RIGHT= Pi+1
  STARTTIMER(Δ)

UPON EVENT < PO, NEW_RIGHT | P > DO
  IF RIGHT != NULL
    CRASH.R= RIGHT
    SUSPECTED= SUSPECTED U {CRASH_R}
    TRIGGER < PFD, CRASH | CRASH.R >
    TRIGGER < PP2PL, PP2PL.SEND | [CRASH, CRASH.R]> TO RIGHT

UPON EVENT < PO, NEW_LEFT | P > DO
  IF LEFT != NULL
    CRASH.L = LEFT
    SUSPECTED= SUSPECTED U {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 Pi IN SUSPECTED
      LEFT = NEW_LEFT
      RIGHT= NEW_RIGHT
      TRIGGER < PP2PL,PP2PL.SEND | [CRASH]> TO LEFT
      TRIGGER < PP2PL,PP2PL.SEND | [CRASH]> TO RIGHT
```

Ex 3: Consider a distributed system constituted by n processes $\Pi=\{p_1, p_2\ldots p_n\}$ 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.





```
INIT
  CORRECT = Π
  PENDING = ∅
  DELIVERED = ∅
  ACK = ∅
  FLAG = FALSE

UPON EVENT <URB, SEND | m> DO
  FLAG = TRUE
  TRIGGER <PP2PL.SEND | (ACK, m)> TO NEXT

UPON EVENT <PP2PL.DELIVER | (ACK, m)> FROM p DO
  ACK = ACK ∪ {p}
  IF m ∉ PENDING
    PENDING = PENDING ∪ {m}
    IF RIGHT.FLAG != TRUE
      TRIGGER <PP2PL.SEND | (ACK, m)> TO NEXT

FUNCTION CANDELIVER (m) RETURNS BOOLEAN IS
  RETURN CORRECT ⊆ ACK (m)

UPON EXISTS m ∈ PENDING SUCH THAT CANDELIVER(m) ∧ m ∉ DELIVERED DO
  DELIVERED = DELIVERED ∪ {m}
  TRIGGER <URB, DELIVER | m>

UPON EVENT <CRASH | p> DO
  CORRECT = CORRECT \ {p}
  TRIGGER <NEW.NEXT | p>

UPON EVENT <NEW.NEXT | p> DO
  NEXT = p
  FORALL m IN PENDING
    TRIGGER <PP2PL.SEND | (ACK, m)> TO NEXT
```

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.

1) BETWEEN t_0 AND t_{STAB} LOCAL VARIABLES OF f_t PROCESSES CAN ASSUME RANDOM VALUES.

VALIDITY: NOT GUARANTEED BECAUSE OF TRANSIENT FAILURE THAT CAN ALTER THE VALUE OF THE MESSAGE SENT BUT NOT ALREADY DELIVERED.
NO DUPLICATION: NOT GUARANTEED SINCE THE VALUES ARE RANDOM AND COULD HAPPEN TO HAVE TWO SAME VALUES TO DELIVER.
NO CREATION: NOT GUARANTEED BECAUSE OF TRANSIENT FAILURE THAT ALTERING VALUES COULD CREATE A NEW MESSAGE.
AGREEMENT: NOT GUARANTEED. DUE TO TRANSIENT FAILURE, ONE OR MORE PROCESSES MAY DELIVER m, WHILE OTHERS MAY NOT.

2) AFTER t_{STAB}, THE PROCESSES BECAME NORMAL AND THEY CANNOT CHANGE ARBITRARY THEIR LOCAL VARIABLES AND THOSE VALUES CAN BE WRONG. SO VALIDITY, NO DUPLICATION AND AGREEMENT ARE NOT GUARANTEED, BUT NO CREATION IS GUARANTEED BECAUSE A PROCESS WILL NOT DELIVER m IF THE MESSAGE WAS NEVER SENT.

3) IN ORDER TO GUARANTEE VALIDITY, NO DUPLICATION AND AGREEMENT WE HAVE TO STORE THE STATUS OF SENT AND RECEIVED MESSAGES TO RETRIEVE THE STATUS, TO UNIQUELY IDENTIFY MESSAGES THROUGH TIMESTAMPS AND TO RETRANSMIT MESSAGES AFTER t_{STAB}.

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 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.

Proprietà	PP2PL	FLL
Perdita di messaggi	Nessuna perdita.	I messaggi possono essere persi.
Duplicazione	Nessun messaggio duplicato.	I messaggi possono essere duplicati.
Creazione	Nessun messaggio creato arbitrariamente.	Nessun messaggio creato arbitrariamente.
Garanzia di consegna	Sempre garantita.	Garantita solo con ritrasmissione.

1) **PP2PL**

VALIDITY: GUARANTEED ONLY IF THE PROCESS DOESN'T CRASH BEFORE TRANSMITTING m.
NO DUPLICATION: GUARANTEED THANKS TO PERFECT CHANNELS AND DUPLICATE CONTROL.
NO CREATION: GUARANTEED, BECAUSE MESSAGES MUST BE TRANSMITTED BY AN IDENTIFIABLE PROCESS.
AGREEMENT: NOT GUARANTEED, BECAUSE NOT HAVING A PFD WE CANNOT KNOW IF m IS DELIVERED BY A CORRECT PROCESS OR NOT.
FIFO DELIVERY: NOT GUARANTEED, SAME AS ABOVE.

2) **FLL**

VALIDITY: NOT GUARANTEED, NOT HAVING PERFECT LINKS A MESSAGE CAN BE LOST.
NO DUPLICATION: NOT GUARANTEED, SINCE DUPLICATES ARE ALLOWED IN FLL.
NO CREATION: GUARANTEED, FROM NO CREATION PROPERTY OF FLL.
AGREEMENT: NOT GUARANTEED, BECAUSE NOT HAVING A PFD WE CANNOT KNOW IF m IS DELIVERED BY A CORRECT PROCESS OR NOT.
FIFO DELIVERY: NOT GUARANTEED, SAME AS ABOVE.