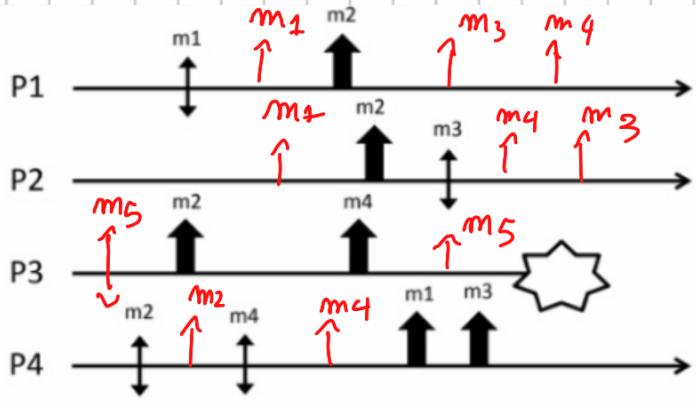
Exercise 4

1. Reliable broadcast but not URB



2. Causal order and total order

$m_2 \longrightarrow m_3$ and $m_2 \longrightarrow m_4$

- m_1, m_2, m_3, m_4
- m_1, m_2, m_4, m_3
- m_2, m_1, m_3, m_4
- m_2, m_1, m_4, m_3
- m_2, m_4, m_1, m_3

3. Violating casual order, TO (UA, WNUTO) but not satisfying TO (UA, SUTO)

- m_1, m_3, m_2, m_4
- m_1, m_3, m_4, m_2
- m_4, m_1, m_2, m_3
- m_4, m_2, m_1, m_3
- m_4, m_1, m_3, m_1

↳ only the correct processes in P_3 We don't edd any delivs and any combination violate SUTO

Exercise 5

a. F, we have NVA all correct processes have the same set of delivery, but P_2 faulty have m_4

b. F, all the correct processes have the same set of delivery, NVA is satisfied

c. F, because P_2 deliver m_3 but not m_2 and violate SUTO

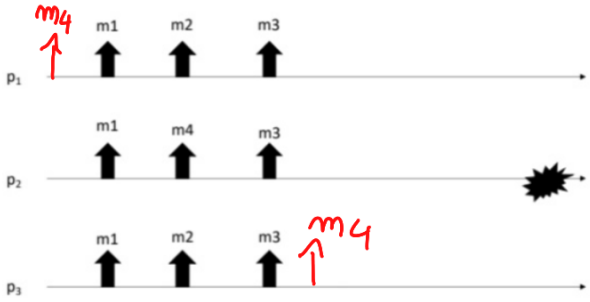
d. V, because P_1 and P_2 have the same order and P_2 respect the order of m_1 and m_3, is WUTO

e. F, this run satisfy WUTO that implies SNUTO

f. V, because is missing m_2 in P_2 that not satisfy SUSO

g. F, because like F P_2 deliver m_3 must deliver m_2 also for satisfy SUTO

h. T, this run:

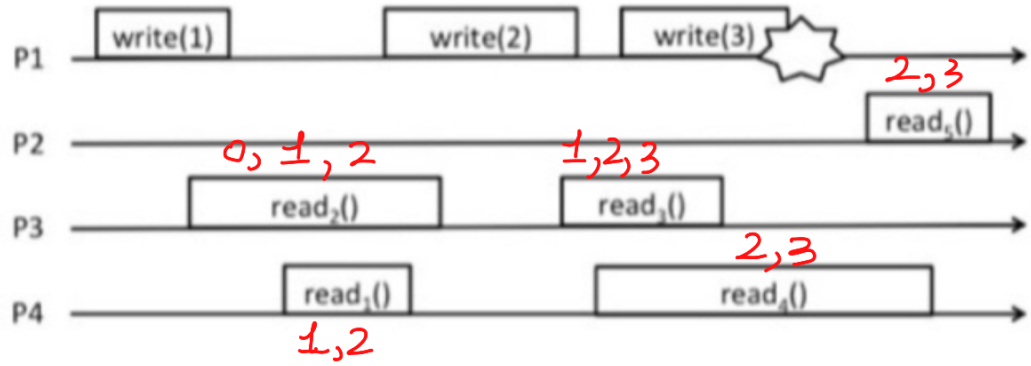


i. F, because P_2 is correct and deliver m_4 but other correct don't deliver it, NVA is violated

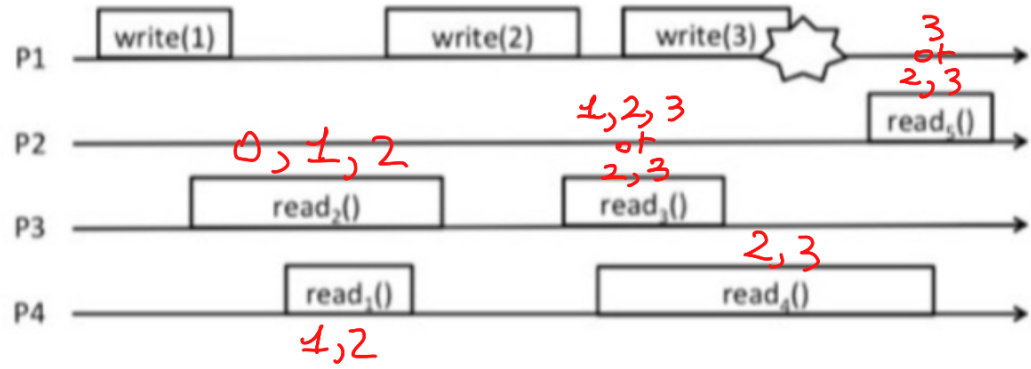
l. F, P_2 is missing m_2 and there is m_4 before m_3.

Exercise 6

1. R_x possible value for Regular Register



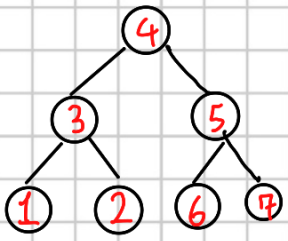
2. R_x possible value for Atomic Register



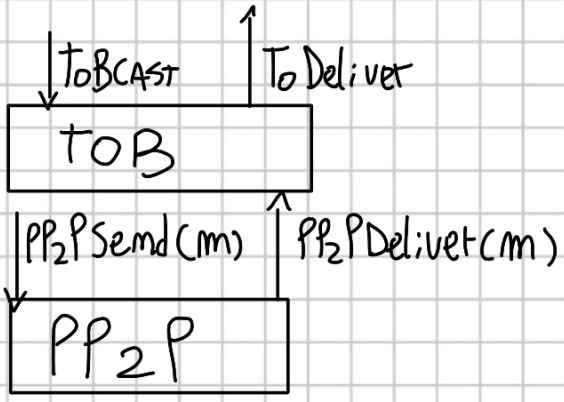
↳ because R_1 and R_2 are concurrent can have casual value

- R_3 depend on value chosen by R_1, if R_1 choose 2 then R_3 must return 2,3 otherwise can return 1,2,3
- R_5 depend on value chosen by R_3 if R_3 choose 3 then R_5 must return 3 otherwise can return 2,3

Exercise 7 - solution



- processes have unique names.
- each process can talk only with neighbor processes.
- Links are PP2P Link.



P_1: $FATHER_1 = P_3$

$L\_CHILD_1 = \perp$

$R\_CHILD_1 = \perp$

one single process have $FATHER_i = \perp$, here P_4.

init:

$FATHER_i$ = get_father()

L_CHILD_i = get_left()

R_CHILD_i = get_right()

$Sequence\_number_i = sn_i = 0$

$LAST\_Delivered_i = 0$

$pending_i = \emptyset$

upon event ToBcast(m)

if $FATHER_i \neq \perp$

trigger PP2PSend('UNORDERED', m) to $FATHER_i$ (type of message)

else

$sn_i = sn_i + 1$

trigger PP2Psend('ORDERED', m, sn_i) to $LEFT_i$

trigger PP2Psend('ORDERED', m, sn_i) to $RIGHT_i$

Upon event PP_2PDeliver ('UNORDERED', m) from P_j:

If $FATHER_i \neq \perp$

trigger PP_2PSend ('UNORDERED', m) to $FATHER_i$

else

$SM_i = SM_i + 1$

trigger PP_2PSend ('ORDERED', m, SM_i) to $LEFT_i$ and $Right_i$

Upon event PP_2PDeliver ('ORDERED', m, s)

$pending_i = pending_i \cup \{<m, s>\}$

trigger PP_2PSend ('ORDERED', m, s) to $LEFT_i$ and $Right_i$

Upon $\exists <m, s_m> \in pending_i$ s.t. $SM = LAST\_DELIV. + 1$

$pending_i = pending_i \setminus \{<m, s>\}$

trigger ToDeliver (m)

$LAST\_DELIVERED = SM$

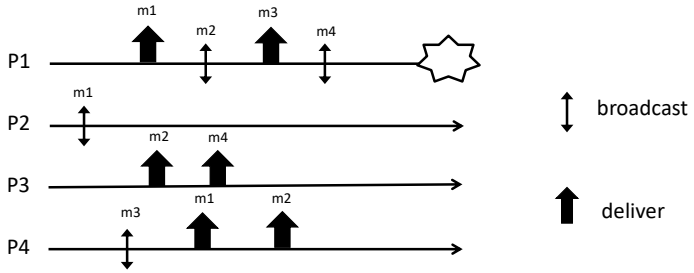
NO ID process

Dependable Distributed Systems
Master of Science in Engineering in Computer Science

AA 2022/2023

Lecture 18 – Exercises
November 9th, 2022

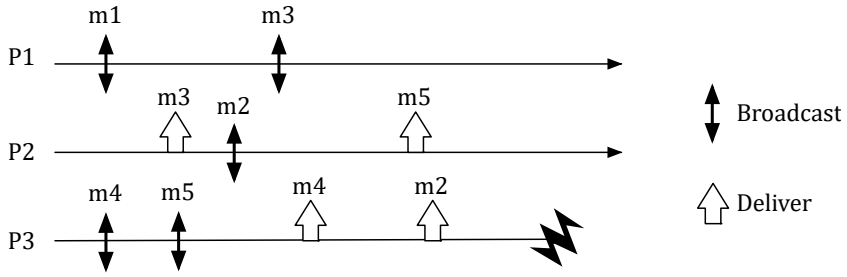
Ex 1: Consider the execution depicted in the Figure



Answer to the following questions:

1. Provide all the delivery sequences that satisfy both causal order and total order
2. Complete the execution in order to have a run satisfying TO(UA, WNUTO), FIFO order but not causal order

Ex 2: Consider the partial execution shown in the Figure and answer to the following questions:



1. Complete the execution in order to have a run satisfying the Regular Reliable Broadcast specification but not Uniform Reliable Broadcast one.
2. For each process, provide <u>ALL</u> the delivery sequences satisfying FIFO Reliable Broadcast but not satisfying causal order.
3. For each process, provide <u>ALL</u> the delivery sequences satisfying total order and causal order.

Ex 3: Consider a distributed system composed of N processes p_1, p_2, … p_N, each having a unique identifier myID. Initially, all processes are correct (i.e. correct={ p_1, p_2, … p_N }). Consider the following algorithm:

```
upon event Xbroadcast(m)
        mysn = mysn+1;
        ∀ p ∈ correct
                pp2pSend ("MSG", m, mysn, myId);

upon event pp2pReceive ("MSG", m, sn, i)
       mysn= mysn+1;
       if (m ∉ delivered)
          trigger XDeliver (m);
          delivered = delivered ∪ {m};

upon event crash (pi)
        correct = correct / {pi}
```

Let us assume that: (i) links are perfect, (ii) the failure detector is perfect and (iii) initially local variables are initialized as follows mysn=0 e delivered =∅.
Answer to the following questions:

1. Does the `Xbroadcast()` primitive implement a Reliable Broadcast, a Best Effort Broadcast or none of the two?
2. Considering only the ordering property of broadcast communication primitives discussed during the lectures (FIFO, Causal, Total), explain which ones can be satisfied by the `Xbroadcast()` implementation.

Provide examples to justify your answers.

Ex 4: Consider a distributed system constituted by *n* processes $\prod$={p_1, p_2… 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 5: Consider a distributed system composed of n processes p1, p2,… pn connected through a ring topology. Initially, each process knows the list of correct processes and maintains locally a `next` variable where it stores the id of the following process in the ring.
Each process can communicate only with its next trough FIFO perfect point-to-point channels (i.e. the process whose id is stored in the `next` variable).
Processes may fail by crash and each process has access to a perfect failure detector.

Write the pseudo-code of a distributed algorithm implementing a (1, N) atomic register.

Ex 6: A transient failure is a failure that affects a process temporarily and that alter randomically 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** $\mathcal{P}$.

upon event $\langle$ *rb*, *Init* $\rangle$ **do**
correct := Π;
from[p] := $[\emptyset]^N$;

upon event $\langle$ *rb*, *Broadcast* | m $\rangle$ **do**
trigger $\langle$ *beb*, *Broadcast* | [DATA, *self*, m] $\rangle$;

upon event $\langle$ *beb*, *Deliver* | p, [DATA, s, m] $\rangle$ **do**
if $m \notin$ *from*[s] **then**
trigger $\langle$ *rb*, *Deliver* | s, m $\rangle$;
from[s] := *from*[s] $\cup \{m\}$;
if $s \notin$ *correct* **then**
trigger $\langle$ *beb*, *Broadcast* | [DATA, s, m] $\rangle$;

upon event $\langle$ $\mathcal{P}$, *Crash* | p $\rangle$ **do**
correct := *correct* $\setminus \{p\}$;
forall $m \in$ *from*[p] **do**
trigger $\langle$ *beb*, *Broadcast* | [DATA, p, m] $\rangle$;

Answer to the following questions:

1. For every property of the Regular Reliable Broadcast specification, discuss if it guaranteed between time t_0 and t_{stab} and provide a motivation for your answer.
2. For every property of the Regular Reliable Broadcast specification, discuss if it eventually guaranteed after t_{stab} and provide a motivation for your answer.
3. Assuming that the system is synchronous, explain how you can modify the algorithm (no pseudo-code required) to guarantee that No Duplication, Validity and Agreement properties will be eventually guaranteed after t_{stab}.

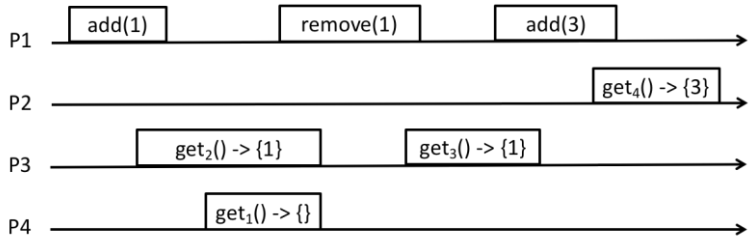
Ex 7: A service is delivered through 3 components that sequentially handles the requests and it receives around 75 requests per second. The components, A, B and C, serve respectively 100, 80 and 90 requests per second. The clients are complaining about the response time of the system. There is the possibility either 1) to double one component employing a perfect load balancer or 2) to substitute one component with an improved one serving 40% more requests per second.
Which component upgrade and option reduce the response time the most? Compute the expected response time.

Ex 8: Consider a set object that can be accessed by a set of processes. Processes may invoke the following operations on the object:

- add(v): it adds the value v in to the set
- remove(v) it removes the value v from the set
- get(): it returns the content of the set.

Informally, every get() operation returns all the values that have been added before its invocation and that have not been removed by any remove().
For the sake of simplicity, assume that a value can be added/removed just once in the execution.

Consider the distributed execution depicted in the Figure



Answer to the following questions:

1. Is the proposed execution linearizable? Motivate your answer with examples.
2. Consider now the following property: “every get() operation returns all the values that have been added before its invocation and that have not been removed by any remove(). If an add(v)/remove(v) operation is concurrent with the get, the value v may or may be not returned by the get()”.
   Provide an execution that satisfy get validity and that is not linerizable.

Exercise 1

1. From the figure we have this CASUAL ORDER:

$m_1 \longrightarrow m_2 \longrightarrow m_4$, $m_3 \longrightarrow m_4$

All possible delivery that satisfy causal order and Total order:

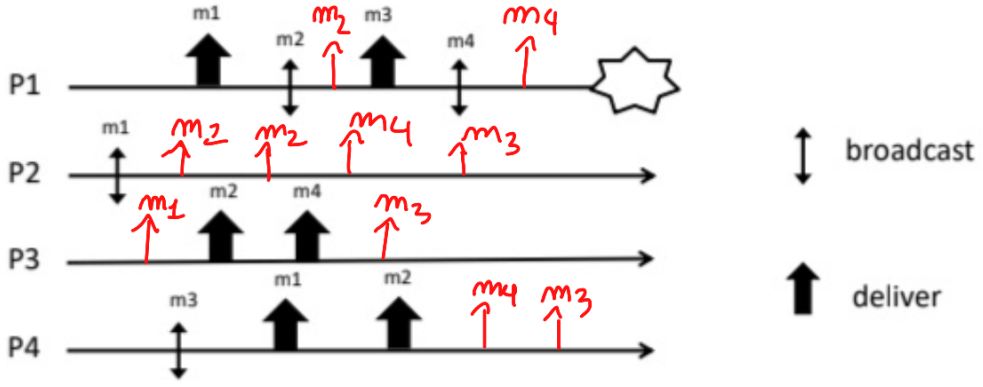
for total order we must notice that m_3 must be deliver after m_1 (P_1), also m_2 must be delivered before m_4 (P_3)

m_1, m_2, m_3, m_4

m_1, m_3, m_2, m_4

(P_1 is faulty can act differently, but satisfy causal order

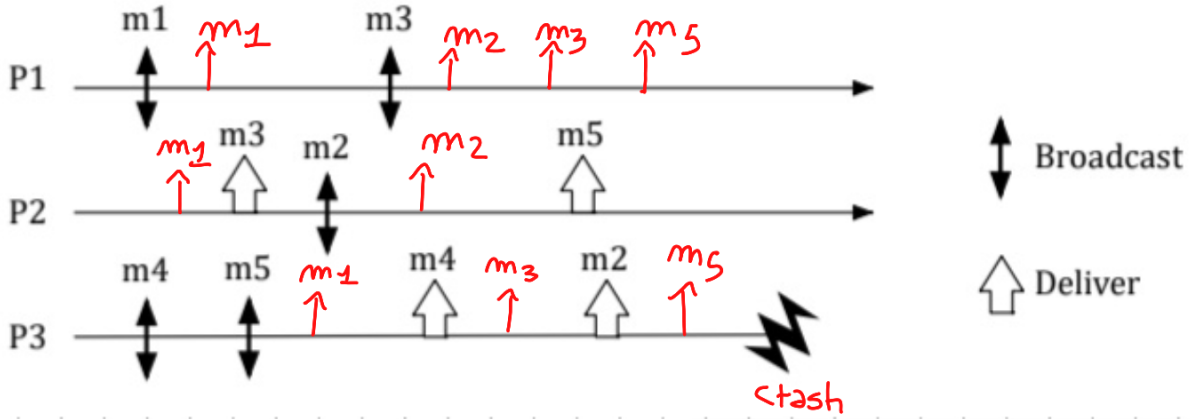
2. TO (UA, WNUTO), FIFO ORDER but not CAUSAL ORDER



FIFO ORDER: $m_2 \longrightarrow m_4$ $\quad m_4 \not\rightarrow m_3$

Exercise 2

1. RRB but not URB



P_3 faulty, m_4 is not delivered by correct one, not URB

2. FIFO: $m_4 \longrightarrow m_5$, $m_1 \longrightarrow m_3$
CAUSAL: $m_3 \longrightarrow m_2$
All delivery satisfying FIFO Relieble Broadcast but not Causal order for each process:

We must deliver also m_4 because we deliver m_5 for have FIFO order, we SATISFY URB that implies RRB!

P_1: $(m_1, m_2, m_3, m_4, m_5)$, $(m_1, m_2, m_4, m_3, m_5)$
$(m_1, m_2, m_4, m_5, m_3)$, $(m_2, m_1, m_3, m_4, m_5)$
$(m_2, m_1, m_4, m_3, m_5)$, $(m_2, m_1, m_4, m_5, m_3)$
$(m_4, m_1, m_2, m_3, m_4, m_5)$, $(m_4, m_1, m_2, m_3, m_5)$
$(m_4, m_1, m_2, m_5, m_3)$, $(m_4, m_2, m_1, m_3, m_5)$
$(m_4, m_2, m_1, m_3, m_5)$, $(m_4, m_2, m_1, m_5, m_3)$
$(m_1, m_4, m_2, m_3, m_5)$, $(m_1, m_4, m_2, m_5, m_3)$
$(m_1, m_4, m_5, m_2, m_3)$, $(m_2, m_4, m_1, m_5, m_3)$
$(m_2, m_4, m_5, m_1, m_3)$

P_2: if we deliver m_4 before m_2 we violate the causal order in P_3, for the other delivery we can have any combination but satisfy $m_4 \rightarrow m_5$ and $m_1 \rightarrow m_3$
EX.: $(m_4, m_1, m_3, m_2, m_5)$

P_3: if in P_2 deliver m_4 before m_2, we can have any combination, only respecting FIFO ORDER. EX. $(m_1, m_4, m_2, m_5, m_3)$

3. $m_4 \to m_5$ and $m_1 \to m_3 \to m_2$, in P_3 m_4 before m_2 and in P_2 m_3 before m_5

$(m_4, m_1, m_3, m_2, m_5)$

$(m_1, m_4, m_3, m_2, m_5)$

$(m_1, m_3, m_4, m_2, m_5)$

$(m_4, m_1, m_3, m_5, m_2)$

$(m_1, m_4, m_3, m_5, m_2)$

For each processes these are possible delivery.

Exercise 3

N processes : $\{P_1, \ldots, P_N\}$

upon event XBroadcast(m)

  mysn = mysn + 1

  $\forall p \in$ correct

    PP2PSend("MSG", m, mysn, myId)

upon event PP2PReceive("MSG", m, sn, i)

  mysn = mysn + 1

  if ($m \notin$ delivered)

    trigger XDeliver(m);

    delivered = delivered $\cup \{m\}$

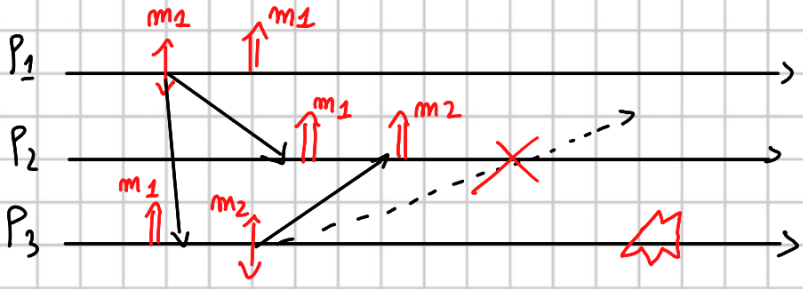
upon event Crash(P_i)

  correct = correct $\setminus \{P_i\}$

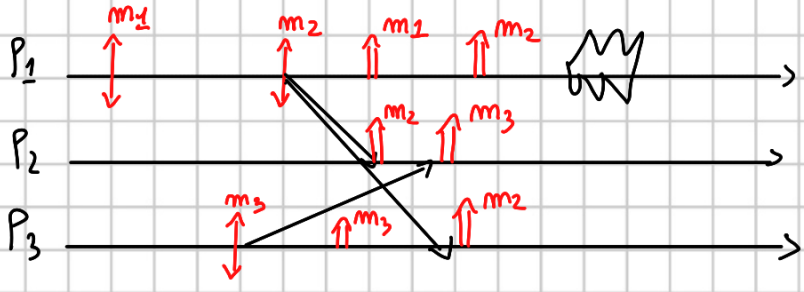
We have Perfect link, a perfect failure detector and in init: mysn = 0 and delivered = $\emptyset$

1. XBroadcast implements a Best Effort Broadcast because we satisfy No duplication, Validity, No creation given that we have PP2P link, but don't satisfy RB because a message sent by a faulty process could be delivered only by a part of correct processes, we don't retrasmitt.

Example: ↕ XBroadcast ⇑ X-Deliver



2. Considering only the ordering, we have no guarantee on the ordered because we do a simple Broadcast, the sequence number is not used, given that don't satisfy nothing This example is only BEB:



Not FIFO!

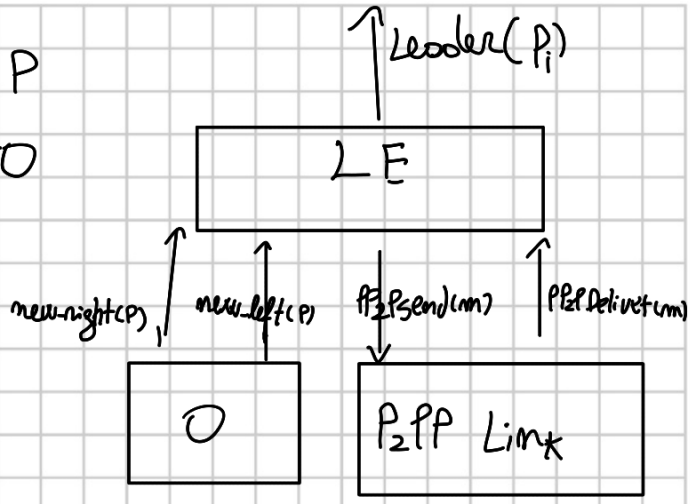
Exercise 4

$\Pi = \{P_1, \ldots, P_m\}$ n processes, FIFO PP2P link, we have a line:

P_i can comunicate only with P_{i-1} and P_{i+1}. We have an Oracle with primitive next_right(P) and next_left(P). Implement a Perfect failure detector.

implements: Perfect Failure Detector, P

Uses: Oracle O, PP_2P link PL FIFO



Upon event $\langle P, Init \rangle$ do

    LEFT = get_left()

    RIGHT = get_right()

    alive = TT

    detected = $\emptyset$

Upon event $\langle O, new\_right \mid p \rangle$ do

    Ex_right = RIGHT

    RIGHT = p

    detected = detected $\cup$ {Ex_RIGHT}

    trigger clash (Ex_Right)

    IF RIGHT $\neq$ null do

        trigger PP_2PSend (|CRASH|, ExRight) to RIGHT

    IF LEFT $\neq$ null

        trigger PP_2PSend (|CRASH|, ExRight) to LEFT

Upon event $\langle O, new\_left \mid p \rangle$ do

    Ex_left = LEFT

    LEFT = p

    detected = detected $\cup$ {Ex_LEFT}

    trigger clash (Ex_LEFT)

    IF RIGHT $\neq$ null do

        trigger PP_2PSend (|CRASH|, detected) to RIGHT

    IF LEFT $\neq$ null

        trigger PP_2PSend (|CRASH|, detected) to LEFT

upon event PP2PDeliver (|CRASH|, P_i) from P_j

detected = detected ∪ P_i

trigger crash(P_i)

If P_j = LEFT AND $P_j \neq$ null

trigger PP2PSend (|CRASH|, detected) to RIGHT

else if P_j = RIGHT AND $P_j \neq$ null

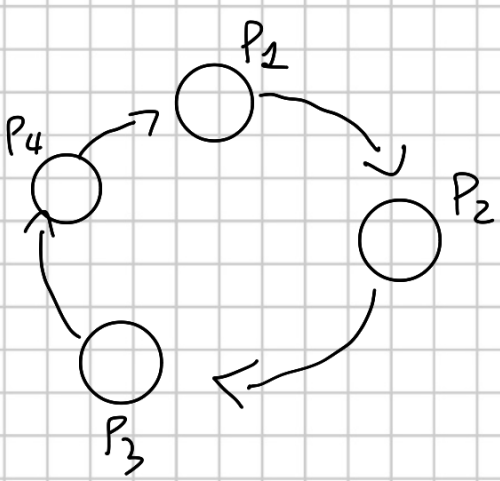
trigger PP2PSend (|CRASH|, detected) to LEFT

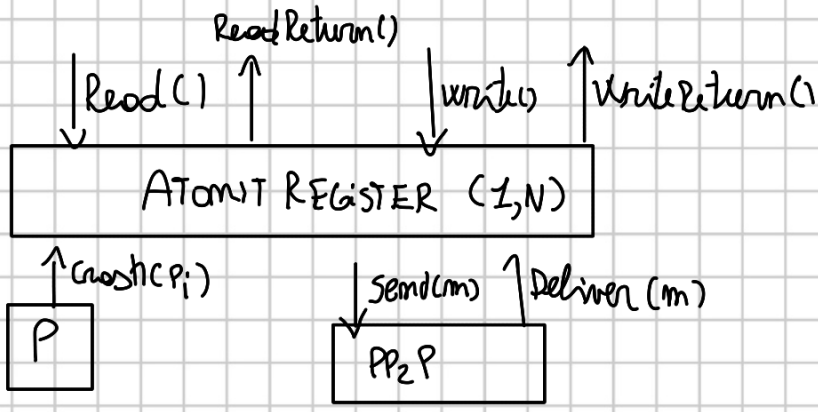
✗ NO MULTIPLE CRASH ADD

In this way we satisfy strong completeness and accuracy given that we have PP2P link and the oracle.

Exercise 5

Ring topology with N processes. We have PP2P link and a Perfect Failure detector P:





Idea is to write the value on a token and send it in the ring, invoke WriteReturn() only when turn back, and can read a process only when have the token. When a process crash and is our next, change next & recreate a new token. Use sequence number for discard past token

implements: (1-N)-Atomic Register, AR

Uses: PP2P link, PL

Perfect Failure Detector, P

init:

alive = Π

val = ⊥ , sn = 0 sequence_number of token send

Reading = FALSE

Write = FALSE

next = $P_{(i+1) \bmod N}$

if self = P_0

sn = sn + 1

TOKEN = val

trigger PP2PSend (⟨READING, sn⟩, TOKEN) to next

Upon event ⟨AR, Write | v⟩ do

write = TRUE ⇝ now when recive token can WRITE and wait for return

val = v

Upon event ⟨AR, Read⟩ do

Reading = TRUE

Upon event PP2PDeliver (⟨READING, s⟩, TOKEN)

if s = sn

do_nothing, have recived same TOKEN

else if WRITE = TRUE

TOKEN = val, sn = sn + 1

trigger PP2PSend (⟨WRITING, sn⟩, TOKEN) to next

else if READING = TRUE

val = TOKEN, READING = FALSE, SN = sn + 1

trigger PP2PSend(<READING, sn>, TOKEN) to next
trigger ReadReturn(val)
else
val = TOKEN, sn = sn + 1
trigger PP2PSend(<READING, sn>, TOKEN) to next
upon event PP2PDeliver(<WRITING, s>, TOKEN)
if sn = s
do-nothing, have recived same TOKEN
else if write = TRUE
sn = sn + 1
trigger WriteReturn()
trigger PP2PSend(<READING, sn>, TOKEN) to next
else if READING = TRUE
val = TOKEN, READING = FALSE, SN = Sn + 1
trigger PP2PSend(<WRITING, sn>, TOKEN) to next
trigger ReadReturn(val)
else
val = TOKEN, sn = sn + 1
trigger PP2PSend(<WRITING, sn>, TOKEN) to next
upon event Crash(P_i)
alive = alive / {P_i}
if P_i = next
next = next_alive() → return the alive next
trigger PP2PSend(<READING, sn>, TOKEN) to next

↓

if next have same sn have receive the precedent token

Exercise 6

1.

No duplication: is not guarantee because if a process that have a set of delivery, it is effected by a transient failure and his variable From [p] is changed with other values, there is the possibility that in <bebDeliver> the condition $m \notin$ From [s] is satisfied multiple times and have multiple delivery of some message.

No creation: is not guarantee because if a process is effected by a transient failure, its From[p] is changed randomly, also with values not given by a broadcast. If the some process in the some interval time crash, then the other correct processes may deliver some wrong messages that are in From[p], not previosly broadcast by a process.

Validity: is guarantee, because we consider a transient failure processes as actually failed, if a correct process p broadcast a message m, thanks to "if $m \notin$ From[s]", p will deliver it at some time.

Agreement: is guarantee because if we consider a transient failure processes as actually failed, all the correct processes will deliver the some set of messages, but some messages could be not satisfy no creation. This is guarantee by the From[s] variable that

check if the message is already delivered and by the fact that in every crash we broadcast all messages received from that processes, in this way all correct processes recive missing messages.

2. After t_{STAB} the processes will not affected by transient failure, but some processes are been effected, the algorithm is the same, now the precedent processes that was affected by transient failure return correct, for these also validity and agreement is not eventually guarantee.

Validity: if in the transient failure of p, FromCp] is changed with randomly messages and after t_{STAB} that process want to broadcast a message and is already in FromCp], he will not deliver it, violating agreement.

Agreement: We have the same problem of validity, for example like we say for validity p could not deliver message m, but other correct processes could deliver it, and we have that correct processes have different set of messages, violating agreement.

3. For guarantee the No Duplication, validity and agreement after t_{STAB} we can use the assumption that by N processes F_t can suffer transient failure. We use a Quorum

with $N - F_c$ (faults not transient) messages, we send with every messages the from[p] variable, when we recieve $N-F_c$ from[p] we do an interception between all from[p], and eventually we achieve all the property, because the missing messages are retrasmitted and the randomizing is corrected.

Exercise 7

$$E[R]^{M/M/1} = \frac{1}{100-75} + \frac{1}{160-75} + \frac{1}{90-75}$$

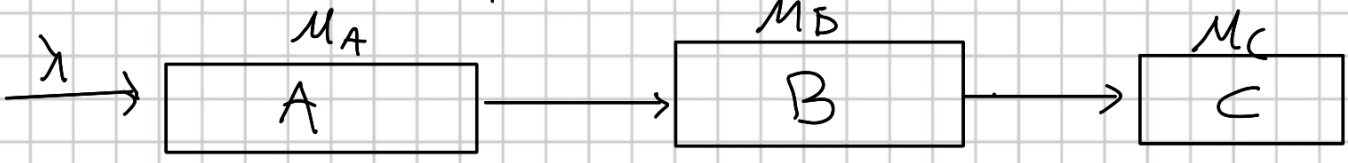
3 components handles sequentially the service.

$\lambda = 75$ req/sec is the arrival rate

$\mu_A = 100$ req/sec
$\mu_B = 80$ req/sec
$\mu_C = 90$ req/sec
} service rate

1) double one component employing a perfect load balancer.

2) substitute one component with one 40% faster.



1) in the first case we have a perfect load balancer, we double the slowest component $\mu_B = 160$ req/sec.

$$E[R]^{M/M/1} = \frac{1}{100-75} + \frac{1}{160-75} + \frac{1}{90-75} =$$

$$= 0,045 + 0,0118 + 0,0667 = 0,1194$$

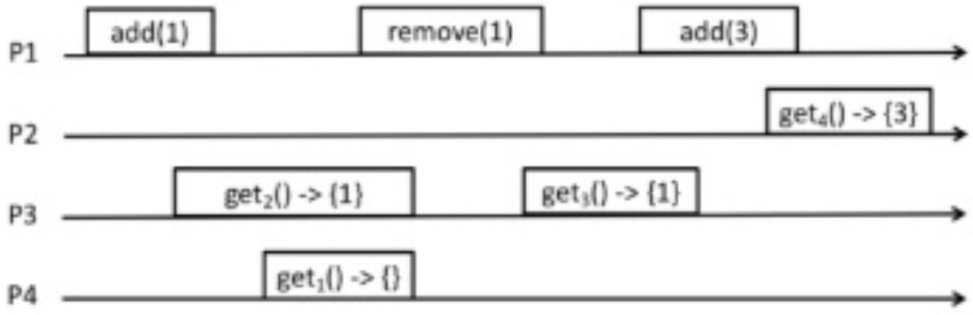
2) With second option upgrade always the slowest component and obtain $\mu_B = 112$ req/sec

$$E[R]^{M/M/1} = \frac{1}{25} + \frac{1}{37} + \frac{1}{15} =$$

$$= 0{,}045 + 0{,}0275 + 0{,}06 = 0{,}1335$$

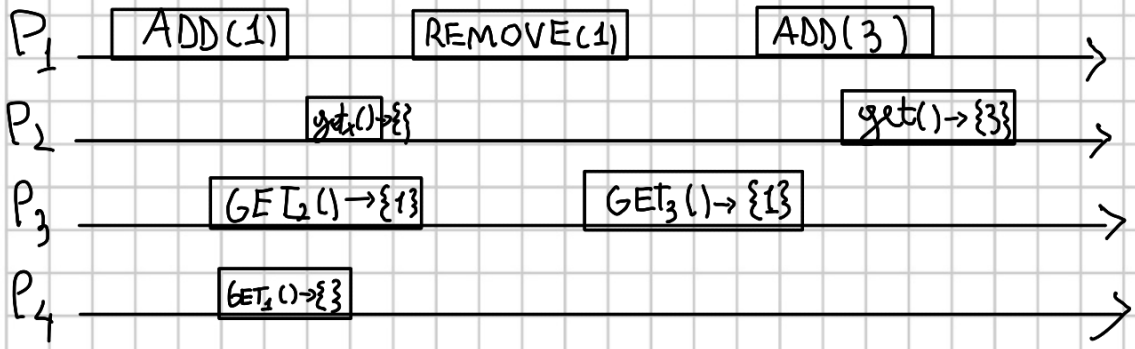
We chose option 1) and upgrade component B!

Exercise 8



1. No is not linearizable because of $get_1() \to \{\}$ and $get_3() \to \{3\}$ are in contradiction, we have no problem with $get_1() \to \{\}$, $get_2() \to \{1\}$ and remove(1) because are concurrent and we can rearrange, but $get_3()$ we can swap only with remove(1). we can not provide a sequence S legal.

2. The execution with the get-validity provided in the text, it is linearizable, we provide a new execution that is not linearizable and satisfy get validity.



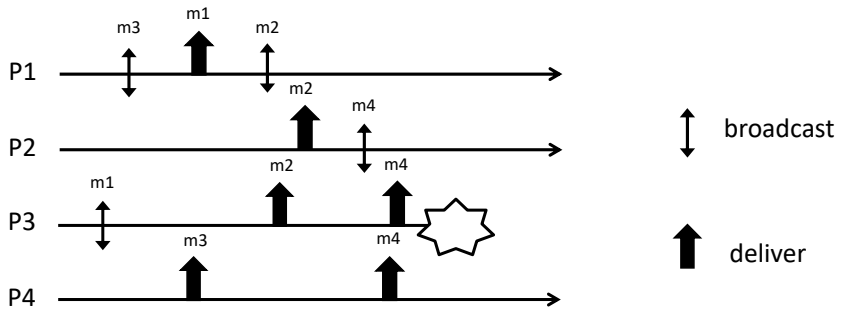
we change the execution, now we have three concurrent events get_x, get_2 and get_1 but get_x violate validity because return 1 but no often ADD(1).

Dependable Distributed Systems
Master of Science in Engineering in Computer Science

AA 2022/2023

Lecture 19 – Exercises
November 16th, 2022

Ex 1: Consider the partial execution depicted in the Figure

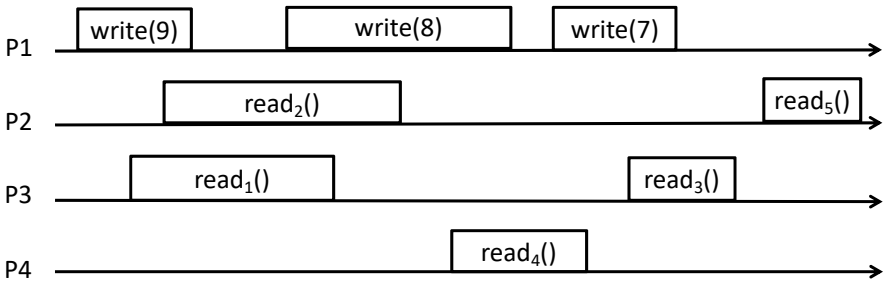


Answer to the following questions:

1. Provide ALL the possible delivery sequences that satisfies causal order and TO (UA, SUTO)
2. Complete the execution in order to have a run satisfying TO (UA WNUTO), FIFO order Broadcast but not Causal Order Broadcast
3. Complete the execution in order to have a run satisfying Regular Reliable Broadcast but not Uniform Reliable Broadcast and not satisfying Total Order.

NOTE: In order to solve the exercise, you can only add broadcast, deliveries and failures.

Ex 2: Consider the partial execution depicted in the Figure



Answer to the following questions:

1. Define ALL the values that can be returned by read operations (Rx) assuming that the run refers to a regular register.
2. Define ALL the values that can be returned by read operations (Rx) assuming that the run refers to an atomic register.

3. Assign to each read operations (Rx) a return value that makes the execution linearizable.

Ex 3: 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 ⟨ Send | [DATA, self, m, lsn] ⟩ to p;
lsn := lsn + 1;

upon event ⟨ 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 4: 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 5: Let us consider the following algorithm implementing a (1, N) atomic register in synchronous system.

```
1   upon event ⟨ onar, Init ⟩ do
2   (ts, val) := (0, ⊥);
3   correct := Π;
4   writeset := ∅;
5   readval := ⊥;
6   reading := FALSE;

7   uponevent⟨P,Crash |p⟩do
8   correct := correct \ {p};

9   upon event ⟨ onar, Read ⟩ do
10  reading := TRUE;
11  readval := val;
12  trigger ⟨ beb, Broadcast | [WRITE, ts, val] ⟩;

13  upon event ⟨ onar, Write | v ⟩ do
    trigger ⟨ beb, Broadcast | [WRITE, ts + 1, v] ⟩;
```

```
14  upon event ⟨ beb, Deliver | p, [WRITE, ts', v'] ⟩ do
15  if ts' > ts then
16          (ts, val) := (ts' , v' );
17  trigger ⟨ pl, Send | p, [ACK] ⟩;

18  upon event ⟨ pl, Deliver | p, [ACK] ⟩ then
19  writeset := writeset ∪ {p};

20  upon correct ⊆ writeset do
21  writeset := ∅;
22  if reading = TRUE then
23          reading := FALSE;
24          trigger ⟨ onar, ReadReturn | readval ⟩;
25  else
26      trigger ⟨ onar, WriteReturn ⟩;
```

Assuming that messages are sent by using perfect point-to-point links and that the broadcast is best effort answer the following questions:

1. Discuss what does it happen to every atomic register property (i.e., termination, validity and ordering) if the failure detector in eventually perfect and not perfect
2. Discuss what does it happen to every atomic register property (i.e., termination, validity and ordering) if we change line 12 with **trigger ⟨** *beb*, Broadcast | [WRITE, ts+1, val] ⟩;

Ex 6: Consider a distributed system composed of *n* processes $\prod=\{p_1, p_2 \ldots p_n\}$ with unique identifiers that exchange messages through perfect point-to-point links. Processes are connected through a directed 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 Leader Election primitive at every process p_i.

Exercise 1

1. $m_3 \rightarrow m_2 \rightarrow m_4$, $m_1 \rightarrow m_2$

$m_3 \longrightarrow m_4$ FOR TO

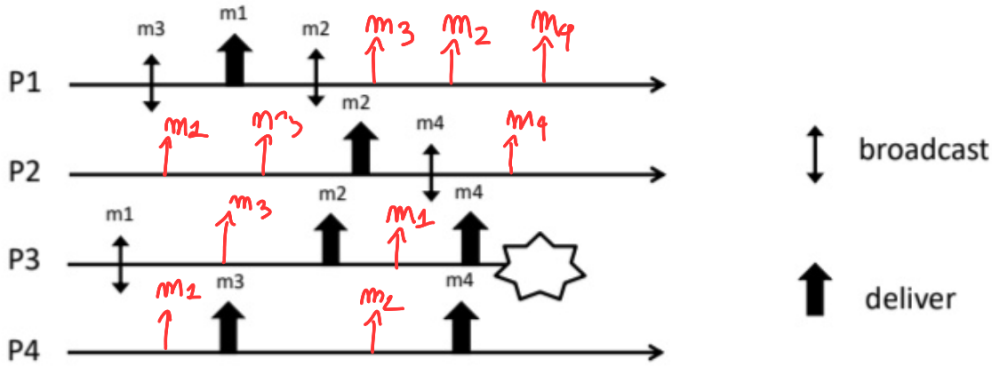
m_3, m_1, m_2, m_4

m_1, m_3, m_2, m_4

must be the possible sequences for all processes for satisfy TO(UA, SUTO) and causal order.

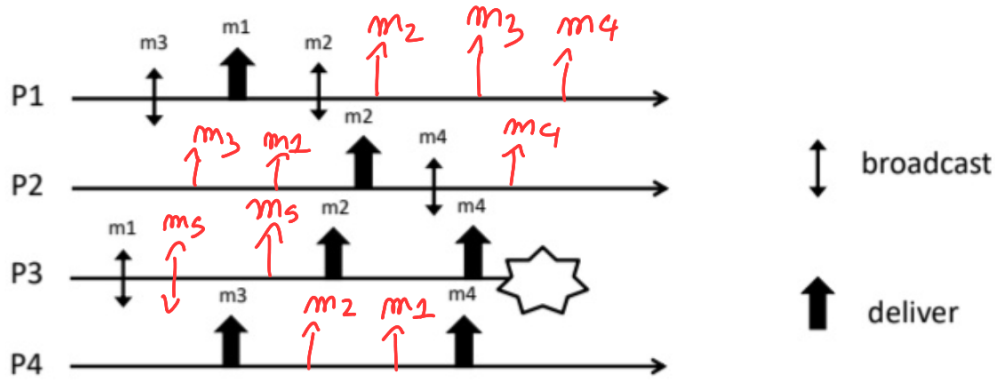
2. $m_3 \longrightarrow m_2$ FIFO

$m_1 \rightarrow m_2 \rightarrow m_4$ CAUSAL ORDER. $m_3 \longrightarrow m_4$



Satisfy TO(UA, WNUTO) and FIFO order broadcast but not causal, because all correct processes have same set of delivery and also faulty P_3 this satisfy UA, WNUTO because correct processes have same order of delivery but the faulty have an inversion ($m_3 \rightarrow m_1$). Not causal because in P_3 we have $m_1 \rightarrow m_2$.

3. We show an execution that satisfy regular reliable broadcast but not uniform and not total order.



without adding a broadcast in P_3 and the delivery of m_5 we can not have RB but not have UA, because we must have that the faulty process have a different set of messages respect to corrects one.

Exercise 2

1. R_x values with regular register

$R_2: 0, 9, 8$, $R_1: 0, 9, 8$, $R_3: 8, 7$, $R_4: 9, 8, 7$, $R_5: 7$

2. R_x values with an atomic register

$R_1: 0, 9, 8$, $R_2: 0, 9, 8$, $R_5: 7$

$R_4: 9, 8, 7$ o $8, 7$ because if R_1 o R_2 return 8 then R_4 can not return 9.

$R_3: 8, 7$ o 7 because if R_4 return 7 then R_3 can not return 8.

3. With these assignment this execution is linearizable:

$R_1: 0$, $R_2: 9$, $R_4: 8$, $R_3: 7$, $R_5: 7$

We respect causal order and also validity.

Exercise 3

1. Considering we have PP2P link:

Validity: is not satisfied because all processes set lsn=0 and next=[1]^N, when they do the first broadcast send the message with sn=0 and increment lsn only after the send. The message will be add in pending but never delivered because we never satisfy 0 = next[s].

No duplication: is satisfied because we use sequence number for messages, send the messages only one time in the broadcast and we continuously check if we have something in pending with sn = next[s] and increment sn when deliver the message.

No creation: is satisfied because we have PP2P link that guarantee this property and also by the algorithm we add in pending only messages sent by a previous broadcast.

Agreement: is not satisfied because if a faulty process broadcast a message m and is added in pending by some correct processes, they will deliver it, but they don't retrasmit it or neither when a processes crash, and for this if a correct proces don't recive the same broadcast it never deliver m.

FiFo delivery: is not satisfied because the second message sent by any process with sn=1 will be always delivered before the first because it will remain forever in pending with sn=0, for the other

messages after the second the property is satisfied, but the second violate FIFO delivery.

2. Using fair loss links:

Validity: same as before and else we don't have the guarantee that when we send the message it will arrive at destination, and the algorithm doesn't have any retransmission.

No duplication: is guarantee because we have the sequence number and the check in the while if we succed in send the message in FLL. Otherwise we have no problem because we don't send the message and can not be delivered.

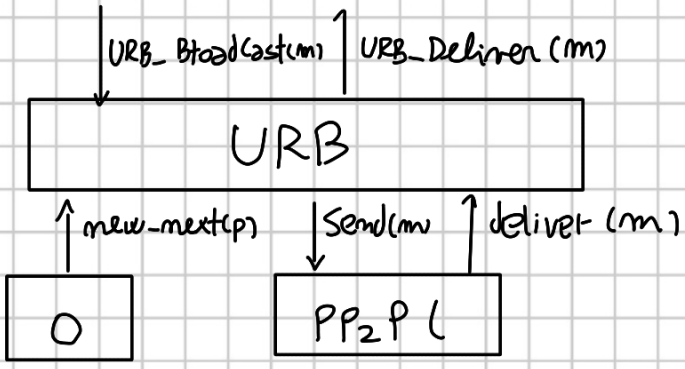
Agreement: is not satisfied like before for the problem with faulty processes and also we can lost the messages on the channel for these some correct processes can deliver a message m but other correct processes could not deliver it.

No creation: is guarantee by the FSL and by the algorithm itself

Fifo Delivery: is not satisfied because for the same problem that we show in 1., the second message is deliver before the first.

Exercise 4

n processes : $\Pi = \{P_1, \dots, P_n\}$, unique identifiers, We use PP2P links and we have a ring structure. We can have crashes. We have an oracle with new_next(p) when neighbor is failing.
Implementing Leader election for each P_i.



We use a token, every process that want to do a broadcast must have the token, and can deliver only when the token has complete one loop. We manage the crash retransmitting and check if the TOKEN is a duplication using the last delivery tag and S.N..

implementation: Uniform Reliable Broadcast, URB

Used: Perfect Point to Point link, PP2P Link
Oracle, O

init:
pending = ⊥
next = $P_{i+1 \pmod n}$
delivered = ∅
wait = FALSE
S.N. = 1 ⟶ sequence number
last_delivered = ⊥
If self = P_1 do
TOKEN = ⊥
trigger PP2PSend (FREE |<TOKEN, sn>) to next

Upon event URB_Broadcast(m)

WAIT = TRUE ⟶ wait for token for Broadcast m

Pending = m

Upon event PP2PDeliver (|FREE| , <TOKEN, S>)

sn = S + 1

IF WAIT = TRUE do

TOKEN = pending,

trigger PP2PSend (|BROADCAST| , <TOKEN, sn>) to next

pending = ⊥

else

trigger PP2PSend (|FREE| , <TOKEN, S>) to next

Upon event PP2PDeliver (|BROADCAST| , < TOKEN, S >)

IF TOKEN = last_delivered do

If sn > S

< DISCARD TOKEN IS A DUPLICATIONS

else

sn = S + 1

trigger PP2PSend (|FREE| , <TOKEN, S>) to next

else IF WAIT = TRUE and pending = ⊥

WAIT = FALSE

delivered = delivered ∪ TOKEN

last_delivered = TOKEN

trigger URB_Deliver (TOKEN)

TOKEN = ⊥ , sn = S + 1

trigger PP2PSend (|FREE|, <TOKEN, sn>) to next

else

sm = s + 1

delivered = delivered ∪ TOKEN

last_delivered = TOKEN

trigger URB_Deliver (TOKEN)

trigger PP2PSend (|BROADCAST|, <TOKEN,sm>) to next

upon event new_next (p)

next = p

TOKEN = last_delivered

trigger PP2PSend (|BROADCAST|, <TOKEN, sm>) to next

We guarantee no duplication of TOKEN checking CAST-DELIVER and the sequence number, if the next that received the new TOKEN finds that his sm is greater then the sender, discard the TOKEN.

Exercise 5

(1-N) atomic register, PP2P links and BeB.

1. we have eventually failure detector:

termination: is satisfied because if a correct process is suspected for error then the condition correct ⊆ writeset is satisfied with less ack, but the write or read will reach the termination.

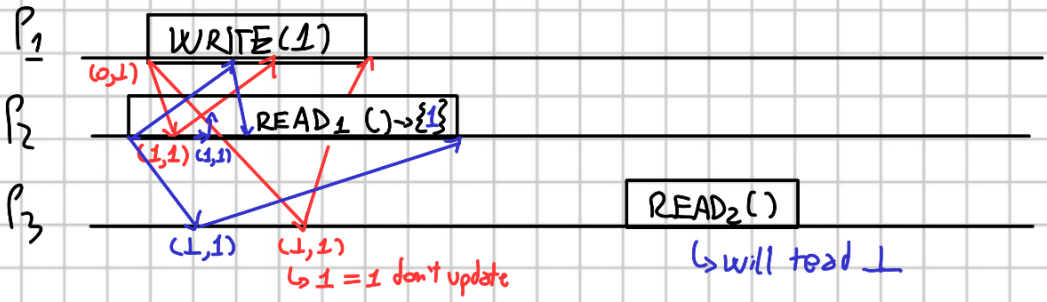
validity: is not satisfied because if a process p that want to read is suspected for error and a process do a writeReturn because belive that p is crashed and also p doesn't recive the new value because writer has used BEB, it is possible that p that is not concurrent with last write, it read the precedent value violating validity.

ordering: is not satisfied for the same motivation of validity. A corect process p that is been suspected could not recive the other messages and if he read a value could return a value that violating the corect order.

2. in line 12: trigger ⟨BeB, Broadcast | [WRITE, ts+1, v]⟩

termination: is satisfied because we change only the timestamp, the process will recive the ack like befre the change and satisfied "correct ⊆ writeset"

validity: is not satisfied because if we have this situation:



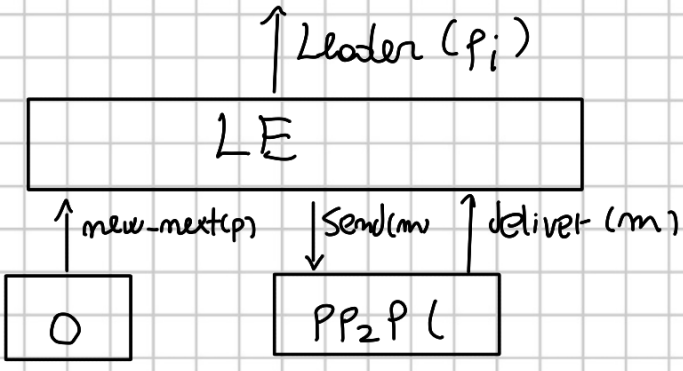
we can have a process after a write that return a value written before it, violate validity.

agreement: is not satisfied like we see in the example we can have a $read_2$ that is after $read_1$ that return a value that violate the ordering.

Exercise 6

n processes : $\Pi = \{p_1, \dots, p_n\}$, unique identifiers, We use PP2P links and we have a ring structure. We can have crashes. We have an oracle with new-next(p) when neighbor is failing.

Implementing Uniform Reliable Broadcast:



We initialize P_n as the leader for all processes. when a process recive new-next(p), check if it is the leader the next that is crashed, then if it is true become himself the leader and send to other the comunication. otherwise send the actual leader to next.

implementation: Leader Election, LE

Used: Perfect Point to Point link, PP2P Link
Oracle, O

init:

next = $P_{(i+1) \bmod N}$
alive = Π
leader = P_n
WAIT = FALSE