Dependable Distributed Systems
Exercise week 2
October 6th, 2022

Exercise 1

With reference to the synchronization of physical clocks, provide the definition of internal and external clock synchronization. In addition, consider a system composed of two processes p_1 and p_2 and one UTC server p_S. Let us assume that:

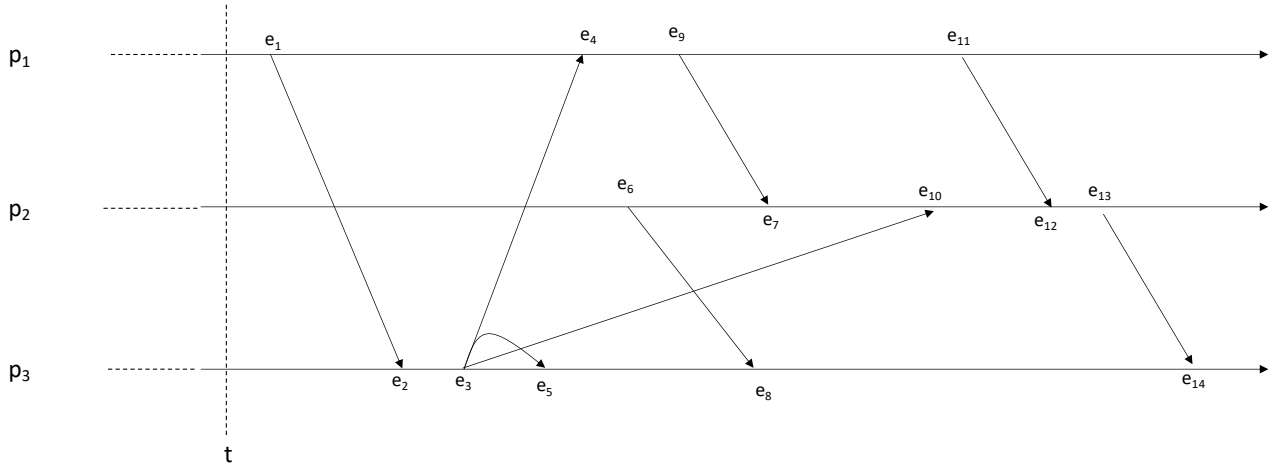
- p_1 and p_2 communicate with p_S by using perfect point-to-point links
- the maximum latency of the channel between p_S and p_1 is 1 ms
- the maximum latency of the channel between ps and p_2 is 2 ms.

In addition, let us assume that p_1 and p_2 start a clocks synchronization procedure at a certain time t by running the Christian algorithm. Answer to the following questions:

1. how much is the accuracy bound D_{ext} of the external synchronization obtained by p_1 and p_2 at the end of the synchronization?
2. Is the current system internally synchronized? If yes, determine the internal synchronization bound D_{int} obtained at the end of the procedure.

Exercise 2

Let us consider the execution history depicted in the figure

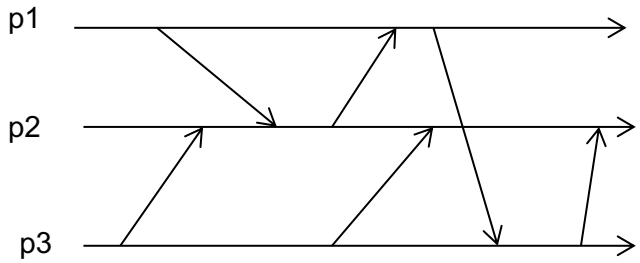


Given the run depicted in the figure state the truthfulness of the following sentences:			
a	According to the happened-before relation, $e_5 \rightarrow e_7$	T	F
b	According to the happened-before relation, $e_4 \parallel e_5$	T	F
c	Let CK_i be the variable storing the scalar logical clock of process p_i. Let us assume that at time t, $CK_i = 0$ for each process p_i. The logical clock CK_2 associated to e_6 is strictly larger than the logical clock CK_1 associated to e_1	T	F
d	Let CK_i be the variable storing the scalar logical clock of process p_i. If at time t $CK_1 = 0$ then the logical clock associated to e_8 is $CK_2=3$	T	F
e	Let CK_i be the variable storing the vector logical clock of process p_i. If at time t $CK_1 = [0, 0, 0]$ then the logical clock associated to e_8 is $CK_2=[3, 0, 3]$	T	F

For each point, provide a justification for your answer

Exercise 3

Describe timestamping techniques based on scalar logical clocks and vector logical clocks. In addition, considering the execution reported in Figure, answer to the following questions:



1. Apply the scalar clock timestamping technique to the execution assigning a timestamp to each event
2. Apply the vector clock timestamping technique to the execution assigning a timestamp to each event
3. List all pairs of concurrent events in the proposed execution

Exercise 4

Consider an asynchronous message passing system that uses vectors clock to implement some causal consistency check. The message passing system is composed by 4 processes with IDs 1 to 4, and, as usual, the ID is used as displacement in the vector clock (i.e., the locations are (p1, p2, p3, p4)).
Vector clocks are updated increasing before send. Processes communicate by point2point links. You start debugging process p1 (the process with id 1) in the middle of the algorithm execution. You see the following stream of messages exiting and entering the ethernet card of process p2:·

- Time 00:00 – EXITING: Send Message [MSG CONTENT] Vector Clock: (1, 0, 0,0)
- Time 00:05 – ENTERING: Rcvd Message [MSG CONTENT] Vector Clock: (1, 2, 3,1)

1. Draw an execution that justifies the vectors clocks you are seeing. Is such an execution unique?
2. There exists an execution that justifies the vector clocks and where there exists at least a process that does not send any message? Justify your answer.

Exercise 5

Let us consider a distributed system composed of N processes p_1, p_2, … pn each one having a unique integer identifier. Processes are arranged in line topology as in the following figure

Let us assume that there are no failures in the system (i.e., processes are always correct) and that topology links are implemented trough perfect point-to-point links.

Write the pseudo-code of a distributed algorithm that is able to build the abstraction of a perfect point-to-point link between any pair of processes (also between those that are not directly connected).

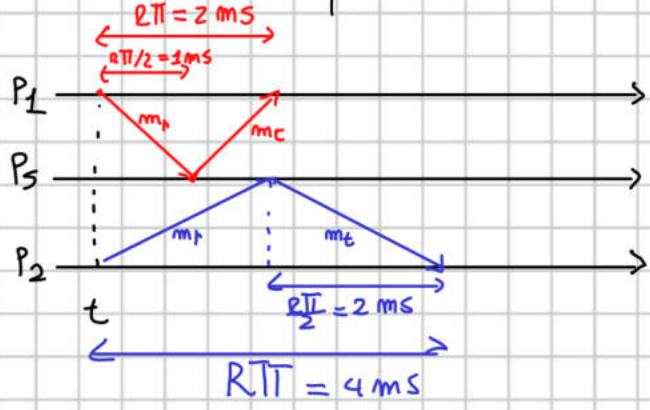
Exercise 1

With reference to the synchronization of physical clocks, we talk about internal synchronization when all the processes synchronize their clocks C_i between them and clocks are internally synchronized in a time interval Δ if $|C_i(t) - C_j(t)| < D$ for $i,j = 1 \cdots N$ and for all time t in Δ, where $D > 0$ is the synchronization bound and $C_i - C_j$ the clocks at processes p_i and p_j. We have external synchronization when processes synchronize their clock C_i with an authoritative external source S (UTC) and clocks C_i are externally synchronized with a time source S if for each time interval Δ : $|S(t) - C_i(t)| < D$.

info:

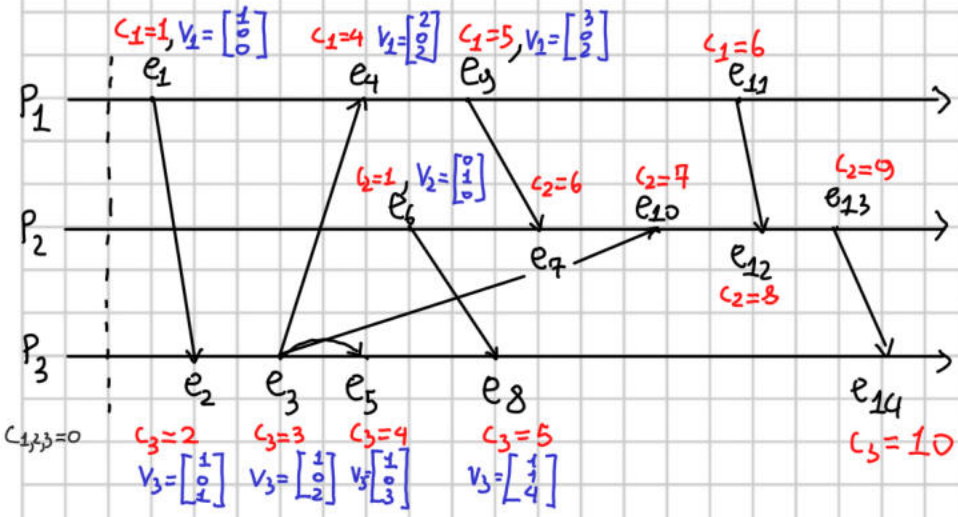
- there are P_1, P_2, P_S
- $P_1 \longleftrightarrow P_S$ 1 ms
- $P_2 \longleftrightarrow P_S$ 2 ms

Simulation of Christian algorithm:



1. Accuracy bound D_{ext} of the external synchronization obtained by P_1 is $D_{ext_1} = \pm \frac{RTT_1}{2} = \pm 1$ ms, while $D_{ext_2} = \pm \frac{RTT_2}{2} = \pm 2$ ms
2. Given that the system is externally synchronized is also internally synchronized in a bound $2 \cdot D_{ext}$ that is $D_{int} = \pm 4$ ms. We choose ± 4 ms and not ± 2 ms because we choose the greatest D_{ext} for consider the worst case scenario, that ± 2 ms do not cover. (Ex.: P_2 forward of D_2 (+2 ms))

Exercise 2



a. False because the events $e_5 \parallel e_7$, not soddisfy ony of the 3 condition of the happened-before.

b. True because V_1 associated to e_4 is $\begin{bmatrix}2\\0\\2\end{bmatrix}$ and V_3 associated to e_5 is $\begin{bmatrix}1\\0\\3\end{bmatrix}$ ($2>1$, $0=0$ but $2<3$)

C. False because the timestamp of $e_1=1$ and timestamp of $e_6=1$, they are equal.

d. False because $c_3=5$ associated to e_8.

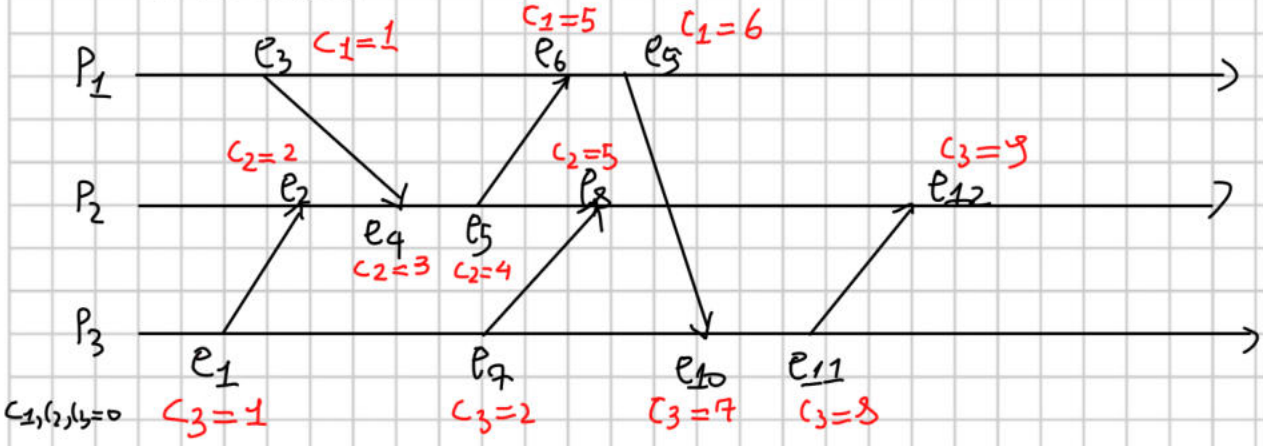
e. False because $V_3=\begin{bmatrix}1\\1\\4\end{bmatrix}$ associated to e_8.

Exercise 3

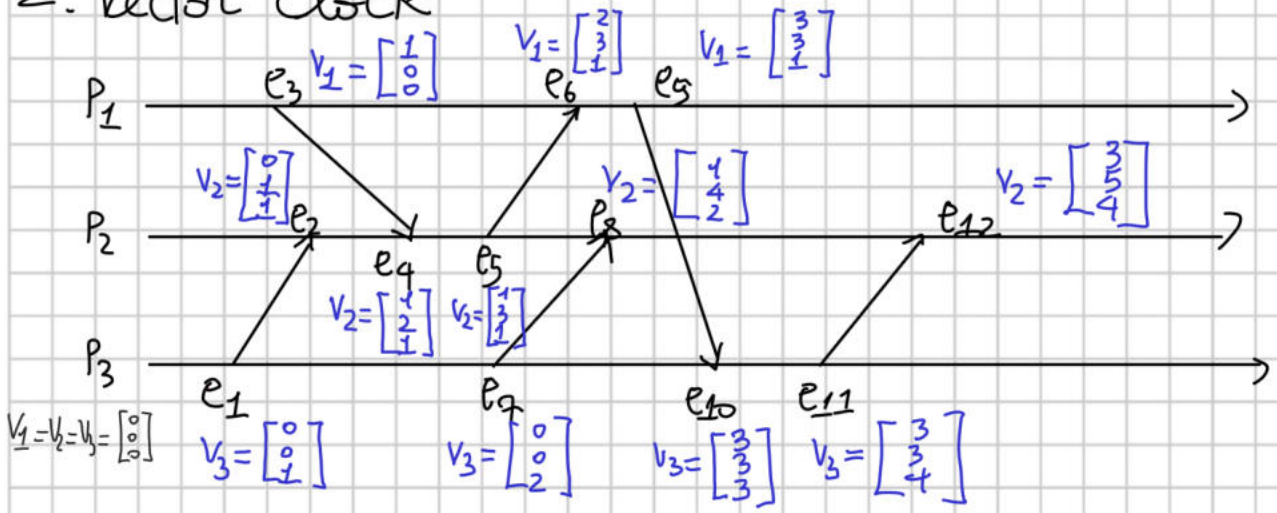
In Scalar logical clocks technique each process P_1 initializes its logical clock $c_i=0$, P_i increase c_i of 1 when it generates an event. When P_i send a message m increase c_i of 1 and timestamp m with $t_s=c_i$, while when P_i recive a message m update $c_i=\max(t_s,c_i)$

In Vector clocks technique for a set of n processes the vector clock in composed by an array of n integers. Each process P_i mantain a vector clock V_i and timestamps events by mean of its vector clock.

1. Scalar clock



2. Vector clock



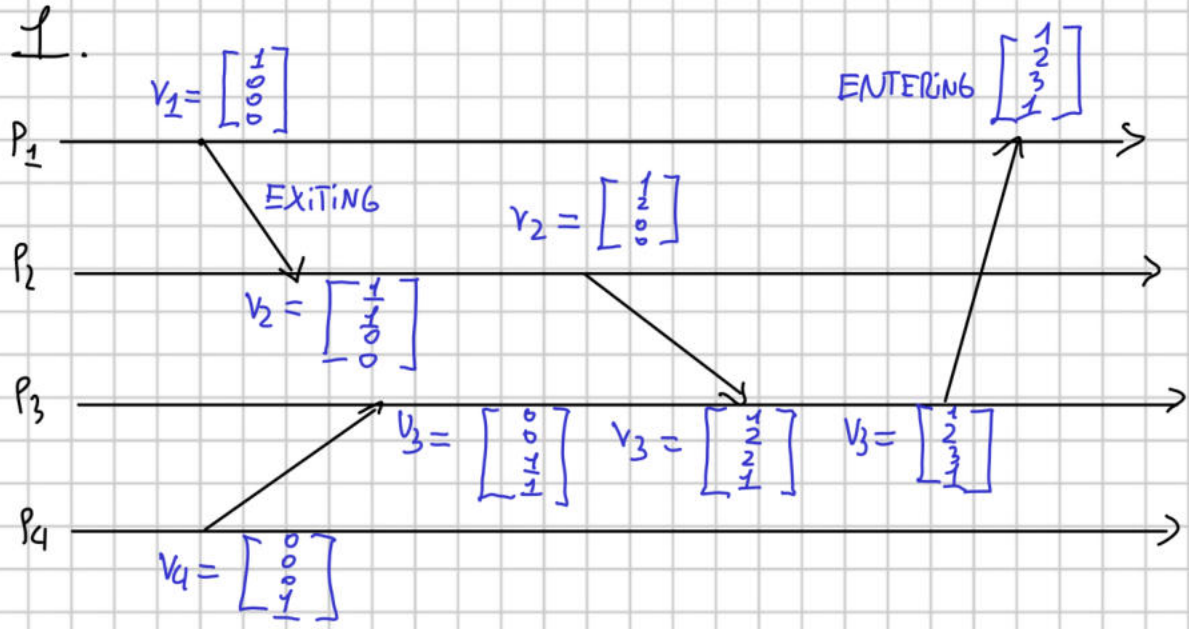
3. We can see the concurrent event using the vector clocks:

$e_1 \parallel e_3$, $e_2 \parallel e_3$, $e_2 \parallel e_7$, $e_5 \parallel e_7$, $e_4 \parallel e_7$, $e_3 \parallel e_4$, $e_9 \parallel e_7$, $e_6 \parallel e_7$, $e_8 \parallel e_9$, $e_8 \parallel e_{10}$, $e_8 \parallel e_{11}$

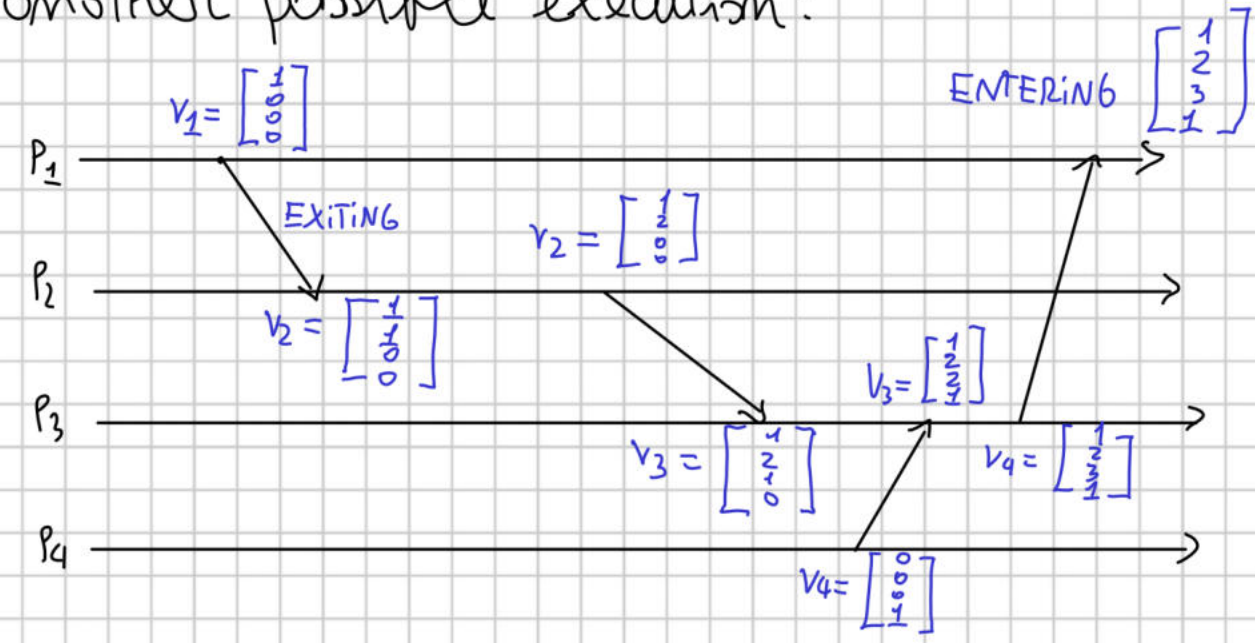
Exercise 4

4 processes: P_1, P_2, P_3, P_4 each with a vector clock V_i

1.

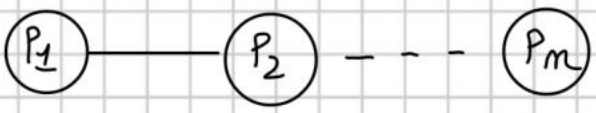


this execution is not unique because we show another possible execution:

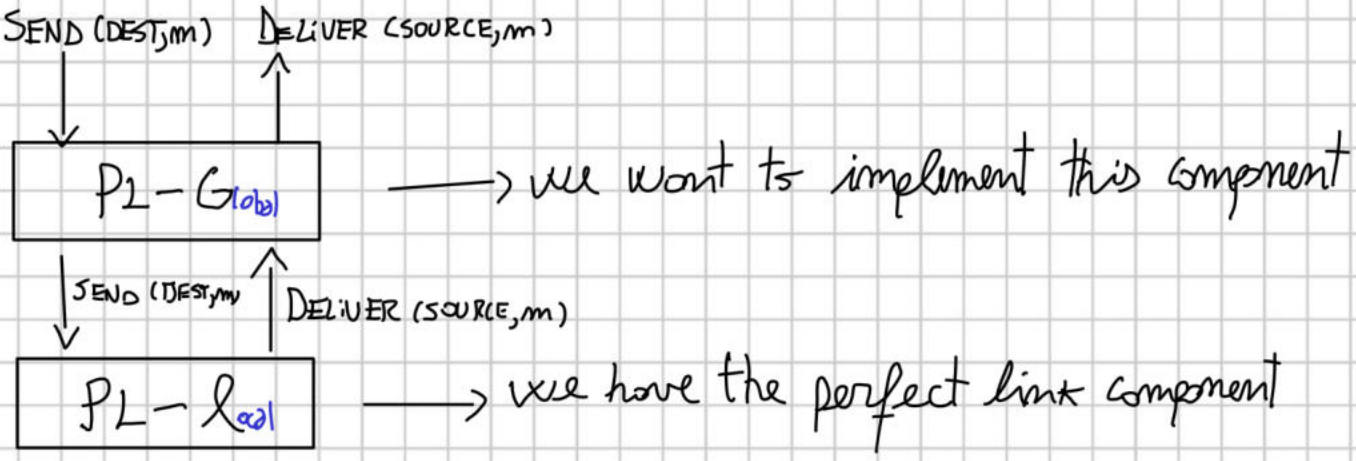


2. its not possible that a process send any message, because in V_1 at time 00:05 enter a vector with all data of array at least 1 which implies that every process send at least a message.

Exercise 5



there are n processes, each with an identifier $i = 1, \dots, N$. each message is for another process $j = 1, \dots, N$. Each process can send the message only to the near process, that are, for a process P_i, P_{i+1} and P_{i-1}. We are using perfect link that guarantee that if the endpoints are correct the message will recived. We want implementing a perfect line P2P given that each process sends a message from the opposite side it delivered it.
We have two event handler, one for send and one for deliver, we indicate with p the sending process and with q the reciver process.
these are the components and the events:



→ we want to implement this component

→ we have the perfect link component

algotithm:

implements: Perfect P2P PL-G

uses: P2P-link PL-l

every process have an ID unique.

event

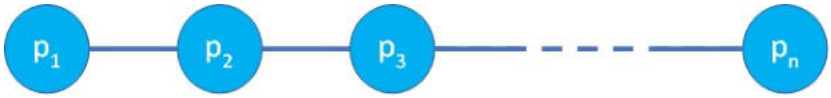
Upon event < PL-G, SEND | q, m > do:

component — parameters

```
  if q > ID then
    trigger < PL-L, SEND | ID+1, < ID, q, m > >
  else if q < ID then
    trigger < PL-L, SEND | ID-1, < ID, q, m > >
  else
    trigger < PL-L, SEND | ID, < ID, q, m >>

Upon event < PL-L, DELIVER | S, < p, q, m >> do:
    if q == ID then
      trigger < PL-G, DELIVER | p, m >
    else if q > ID then
      trigger < PL-L, SEND | ID+1, < p, q, m >>
    else
      trigger < PL-L, SEND | ID-1, < p, q, m >>
```

Exercise week 2 - October 6th, 2022 - Exercise 5. Commented Solution

Let us consider a distributed system composed of N processes $p_1, p_2, \ldots p_n$ each one having a unique integer identifier. Processes are arranged in line topology as in the following figure



Let us assume that there are no failures in the system (i.e., processes are always correct) and that topology links are implemented through perfect point-to-point links.

Write the pseudo-code of a distributed algorithm that is able to build the abstraction of a perfect point-to-point link between any pair of processes (also between those that are not directly connected).

General Comments

The text says that processes have access to a perfect point-to-point link primitive, let us refer to this component with $\mathcal{PL}_l$. This primitive allows every process to exchange messages with at most two other processes, the neighbor processes in the line.

Let us recall the events and properties of a perfect point-to-point primitive.

Module:

Name: PerfectPointToPointLinks, **instance** *pl*.

Events:

Request: $\langle$ *pl*, *Send* | q, m $\rangle$: Requests to send message m to process q.

Indication: $\langle$ *pl*, *Deliver* | p, m $\rangle$: Delivers message m sent by process p.

Properties:

PL1: *Reliable delivery:* If a correct process p sends a message m to a correct process q, then q eventually delivers m.

PL2: *No duplication:* No message is delivered by a process more than once.

PL3: *No creation:* If some process q delivers a message m with sender p, then m was previously sent to q by process p.

Every process can thus trigger *Send* events and catch *Deliver* events of the $\mathcal{PL}_l$ component.

The exercise requests to define a protocol a link abstraction, let us name it $\mathcal{PL}_g$, that implements a perfect point-to-point link between all pairs of processes. In the initial setting, every process can exchange messages only with at most two processes, namely the neighbor processes on the line. It follows that if two processes are not linked by $\mathcal{PL}_l$, such as processes p_1 and p_4, they cannot exchange messages in the current setting. The target of the exercise is the definition of a protocol that implements a communication primitive guaranteeing all the properties of a perfect point-to-point link between all pairs of processes.

Protocol Idea

Each process is associated with an integer identifier and it is placed in a topology (the line) such that all the processes with a lower identifier are placed on one side (from the prospective of a selected process) and all processes with a higher identifier are located on the other side. Furthermore, the processes are placed in order on the line with respect to their identifiers. It follows that if process p_1 wants to communicate with process p_4, it can send its messages to process p_2, that will relay such messages to p_3, which finally forwards them to p_4. More in detail, if the destination of a message is a process with a higher identifier with respect to a selected process, then it is sufficient to forward the message to the neighbors with greater ID till reaching the destination process, and vice-versa for a destination with a lower identifier.

Pseudo-code

Algorithm 1 $\mathcal{PL}_g$

```
1:  procedure INIT
2:  |  ID ← unique integer identifier

3:  upon event ⟨PL_g, Send | dest, m⟩ do
4:  |  if dest > ID then
5:  |  |  trigger ⟨PL_l, Send | ID + 1, ⟨ID, dest, m⟩⟩
6:  |  else if dest < ID then
7:  |  |  trigger ⟨PL_l, Send | ID − 1, ⟨ID, dest, m⟩⟩⟩
8:  |  else
9:  |  |  trigger ⟨PL_l, Send | ID, ⟨ID, dest, m⟩⟩⟩

10: upon event ⟨PL_l, Deliver | q, ⟨source, dest, m⟩⟩ do
11: |  if dest == ID then
12: |  |  trigger ⟨PL_g, Deliver | source, m⟩
13: |  else if dest > ID then
14: |  |  trigger ⟨PL_l, Send | ID + 1, ⟨source, dest, m⟩⟩
15: |  else
16: |  |  trigger ⟨PL_l, Send | ID − 1, ⟨source, dest, m⟩⟩
```

Pseudo-code Comments

The first upon block (lines 3-9) rules how a process must act when the *Send* operation of the global perfect point-to-point primitive we are developing, $\mathcal{PL}_g$, is triggered. It is a tiny procedure that simply starts the propagation of a new message, $\langle source, dest, m\rangle$, over the proper link, $\mathcal{PL}_l$. Again, a content m (let use this word to distinguish m from $\langle source, dest, m\rangle$) targeted to a process with an higher identifier (*dest*) is reachable (on the line) by relaying the content to the neighbor with higher ID and vice-versa.

The second upon block rules how a process that has received a message $\langle source, dest, m\rangle$ from the primitive $\mathcal{PL}_l$ (namely the local perfect point-to-point link) must act. This procedure compares the *dest* field contained inside the received message with the process identifier. If the process is the destination of a content, then it triggers the *Deliver* operation of the primitive we are developing (content m was targeted to this process). Otherwise, it continues the propagation of the message $\langle source, dest, m\rangle$ over the proper perfect point-to-point link. Notice that we need to forward the information about the source ID *source* and the destination id *dest* insider the message, otherwise a process (not linked with the source process) receiving a content m from $\mathcal{PL}_l$ cannot assert whether the content is targeted to it-self and which process was its source.

Informal Correctness Proofs

We briefly check the correctness of the provided pseudo-code. The $\mathcal{PL}_g$ primitive we defined must handle two events, $\langle \mathcal{PL}_g, Send \,|\, dest, m\rangle$ and $\langle \mathcal{PL}_g, Deliver \,|\, source, m\rangle$. Furthermore, it must guarantee the *reliable delivery*, *no duplication* and *No creation* property of a perfect point-to-point link abstraction.

Reliable delivery: if a $\langle \mathcal{PL}_g, Send \mid dest, m\rangle$ event is generated (from the application layer), then the first upon procedure reacts to this event by starting the propagation of the message $\langle source, dest, m\rangle$ over the proper perfect point-to-point link, triggering a *Send* event on the $\mathcal{PL}_l$ primitive. The $\mathcal{PL}_l$ abstraction guarantees all the perfect point-to-point link properties between two linked processes. Therefore, every *Send* event of the $\mathcal{PL}_l$ component generates a *Deliver* event on the destination process (e.g. a $\langle \mathcal{PL}_l, Send \,|\, p_2, \langle source, dest, m\rangle\rangle$ event on process p_1 generates the $\langle \mathcal{PL}_l, Deliver \mid p_1, \langle source, dest, m\rangle\rangle$ event on process p_2). The second upon procedure guarantees the propagation of the message $\langle source, dest, m\rangle$ till process $p_i, i = dest$. Only process $p_i, i = dest$ triggers the $\langle \mathcal{PL}_g, Deliver \,|\, source, m\rangle$ event. The propagation policy eventually guarantees the occurrence of this event.

No duplication: $\mathcal{PL}_l$ guarantees no duplication of the messages diffused by the primitive. For a single $\langle \mathcal{PL}_g, Send \mid dest, m\rangle$ event a single message $\langle source, dest, m\rangle$ is generated and propagated over the line using $\mathcal{PL}_l$, furthermore every process relays at most once a single $\langle source, dest, m\rangle$ message.

No creation: $\langle \mathcal{PL}_g, Deliver \mid source, m\rangle$ event can be triggered only from the reception of a message $\langle source, dest, m\rangle$. The message $\langle source, dest, m\rangle$ is initially generated by the processes only managing a $\langle \mathcal{PL}_g, Send \,|\, dest, m\rangle$ event.

Dependable Distributed Systems
Master of Science in Engineering in Computer Science

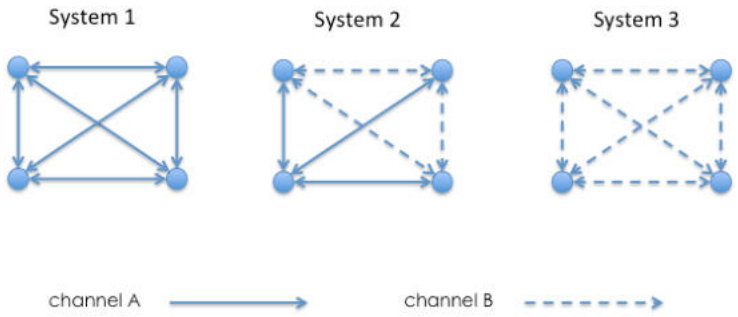
AA 2022/2023

Lecture 8 – Exercises

Ex 1: Let channel A and channel B be two different types of point-to-point channels satisfying the following properties:

- channel A: if a correct process p_i sends a message *m* to a correct process p_j at time *t*, then *m* is delivered by p_j by time $t+\delta$.
- channel B: if a correct process p_i sends a message *m* to a correct process p_j at time *t*, then *m* is delivered by p_j with probability p_{cons} (p_{cons} <1).

Let us consider the following systems composed by 4 processes p_1, p_2, p_3 and p_4 connected trough channels A and channels B.



Assuming that each process p_i is aware of the type of channel connecting it to any other process, answer to the following questions:

1. is it possible to design an algorithm implementing a perfect failure detector in system 2 if only processes having an outgoing channel of type B can fail by crash? true
2. is it possible to design an algorithm implementing a perfect failure detector in system 2 if any process can fail by crash? false
3. is it possible to design an algorithm implementing a perfect failure detector in system 3? false

For each point, if an algorithm exists write its pseudo-code, otherwise show the impossibility.

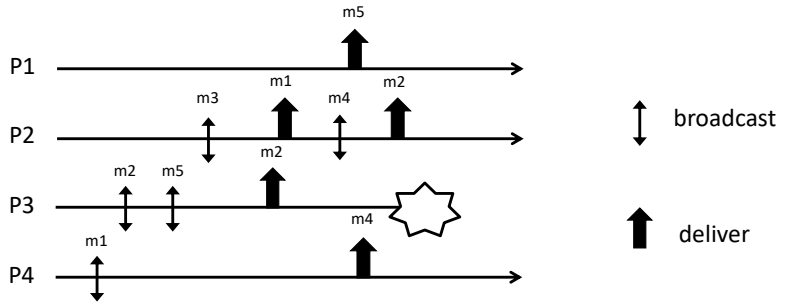
Ex 2: Consider a distributed system composed by n processes $\{p_1,p_2,\ldots,p_n\}$ that communicate by exchanging messages on top of a line topology, where p_1 and p_n are respectively the first and the last process of the network.
Initially, each process knows only its left neighbour and its right neighbour (if they exist) and stores the respective identifiers in two local variables LEFT and RIGHT.
Processes may fail by crashing, but they are equipped with a perfect oracle that notifies at each process the new neighbour (when one of the two fails) through the following primitives:

- **Left_neighbour(p_x)**: process p_x is the new left neighbour of p_i
- **Right_neighbour(p_x)**: process p_x is the new right neighbour of p_i

Both the events may return a NULL value in case p_i becomes the first or the last process of the line.
Each process can communicate only with its neighbours.

Write the pseudo-code of an algorithm implementing a Leader Election primitive assuming that channels connecting two neighbour processes are perfect.

Ex 3: Consider the partial execution depicted in the Figure



Answer to the following questions:

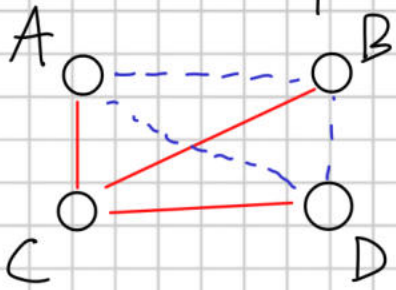
1. Complete the execution in order to have a run satisfying Uniform Reliable Broadcast.
2. Complete the execution in order to have a run satisfying Regular Reliable Broadcast but not Uniform Reliable Broadcast.
3. Complete the execution in order to have a run satisfying Best Effort Broadcast but not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

NOTE: In order to solve the exercise you can add broadcast, deliver and crash events but you cannot remove anything from the run.

Ex 4: Consider a distributed system composed by n processes $\{p_1,p_2,\ldots,p_n\}$ identified through unique integer identifiers. Processes may communicate using perfect point-to-point links. Links are available for any pair of processes.
Processes may fail by crash and each process has access to a perfect failure detector.
Modify the algorithms implementing a distributed mutual exclusion abstraction discussed during the lectures to allow them to tolerate crash failures.

Exercise 1

1. Yes is possible because all processes are linked with channel A that guarantee that the message sent by a process arrive to the destination.

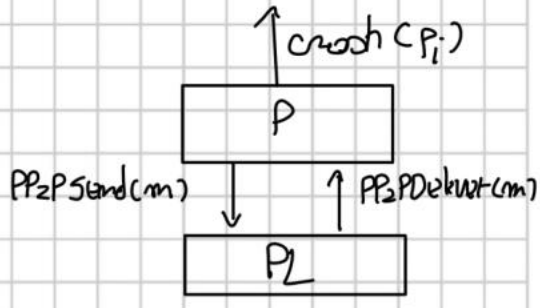


Channel A

Channel B

each have unique ID

Fundamental is C that can not crash, because otherwise we can't guarantee the specification of a perfect failure detector, in particular the accuracy. All messages of A, B and D must pass to C, we can write this pseudo code:



implements: PerfectFailureDetector, P

Uses: PerfectP2PLinks, PL

upon event $\langle P, init \rangle$ do
  alive := Π; $\{P_1, P_2, P_3, P_4\}$
  detected := $\emptyset$;
  $timer_1 = \delta$
  startTimer($timer_1$)

→ we use two timer

upon event timeout $timer_1$ do
  forall $P_j \in \Pi$ do
    trigger $\langle PL, Send \mid LIST, alive_i \rangle$ to P_j
  startTimer($timer_2$) → $timer_2$ is equal to $timer_1$

Upon event $\langle PL, Deliver \mid LIST, l \rangle$ from P_j
  $alive_i = alive_i \cup l$ → all processes leave alive list of C because of channel A

```
when timeout timer_2 do
  for every p_j ∈ Π do
    if p_j ∉ alive_i and p_j ∉ detected_i then
      detected_i = detected_i ∪ {p_j}
      trigger ⟨P, crash | p_j⟩
    trigger ⟨PL, send | HEARTBEAT⟩ to p_j
  alive_i = ∅
  start timer (timer_2)
```

search for crashed processes

```
upon event ⟨PL, Deliver | HeartBeat⟩ from p_j
  alive_i = alive_i ∪ {p_j}
```

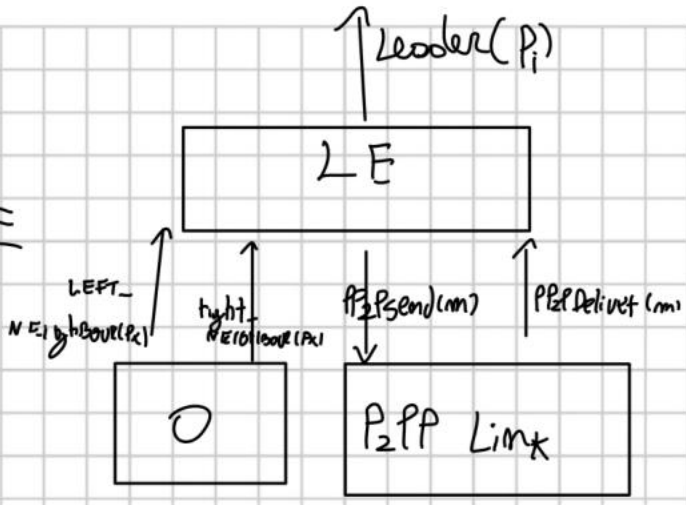
→ all update alive_i list, in particular C will recive all correct HB reques

In this way in the first $timer_1$ the processes A, B, C recive the alive processes known by C, that is always correct because have chA. in the second timer with heartBeat C add who is alive to the list and all work until C is the only that don't fail.

2. False, given that from exercise 1 we learn that without the processes C this algorithm don't work, if C crash the comunication between processes use channels A, that don't guarantee accuracy of PerfectFailureDetector, the system become asynchronous.

3. False the channels B are required for the PerfectFailureDetector, system 3 have only channel A, can't gustantee accuracy

Exercise 2



implements: LeaderElection, LE

Uses: Otacle O, PP2P link PL

Upon event < LE, Init > do
  suspected = ∅
  leader = P_1 → Process at beggining at start is the leader
  LEFT = get_left()
  RIGHT = get_right

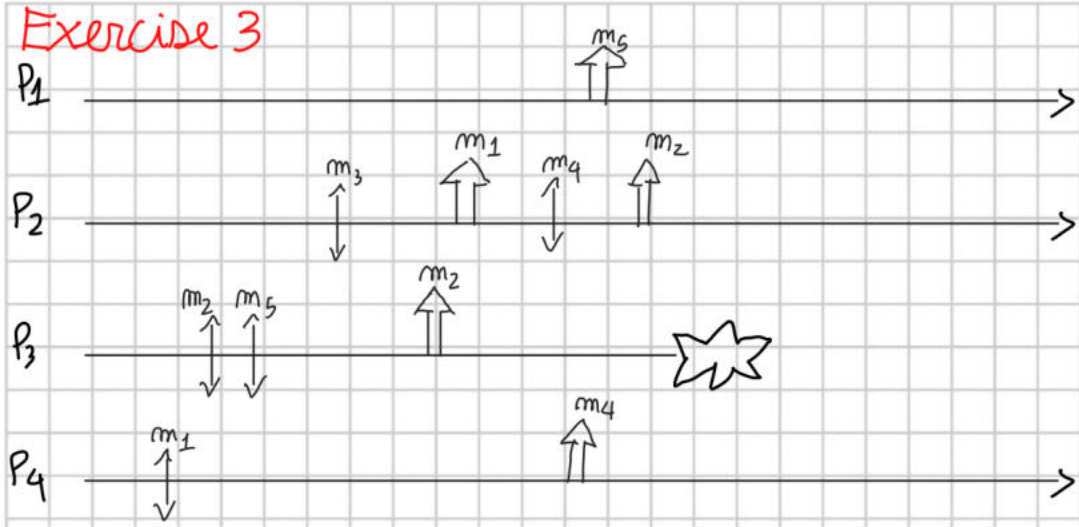
Upon event < O, LEFT_NEIGHBOUR | P_x > do
  left = P_x
  if left = null → must be that leader failed, i am the new leader
    leader = self
    trigger < PL, PP2Psend | NEW_LEADER, leader > to right

Upon event < O, RIGHT_NEIGHBOUR | P_x > do
  right = P_x → we simply update, the problem is on left

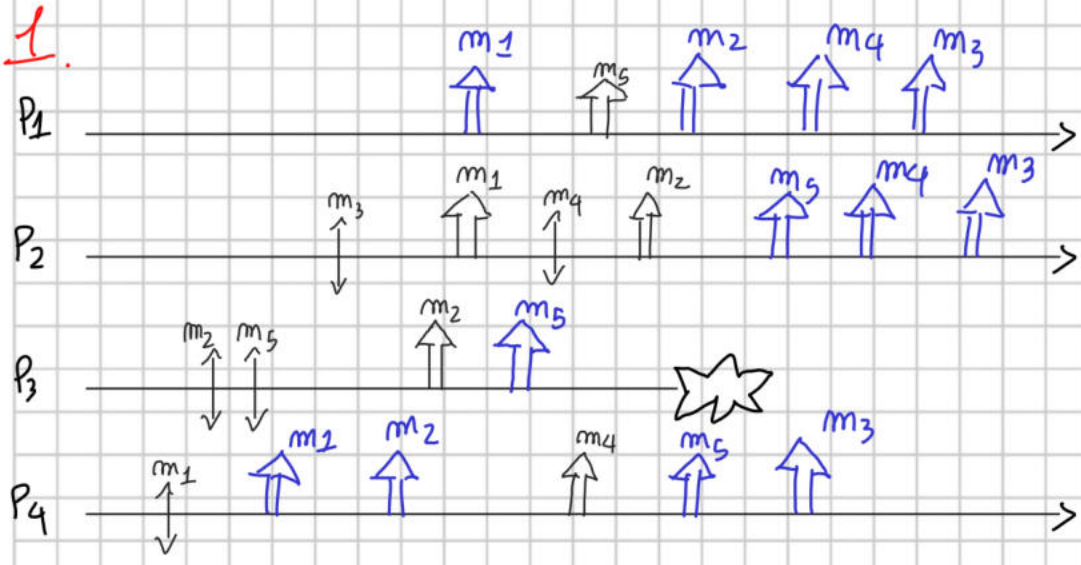
Upon event < PL, PP2PDeliver | NEW_LEADER, l > from left
  leader = l
  trigger < LE, leader | leader >
  trigger < P2, PP2Psend | NEW_LEADER, leader > to right

idea is to set initial leader the process at start, when recive the left neighbour from otacle we check that is null, that implies leader is crashed, and in the case add that process as new leader and propagate the news.

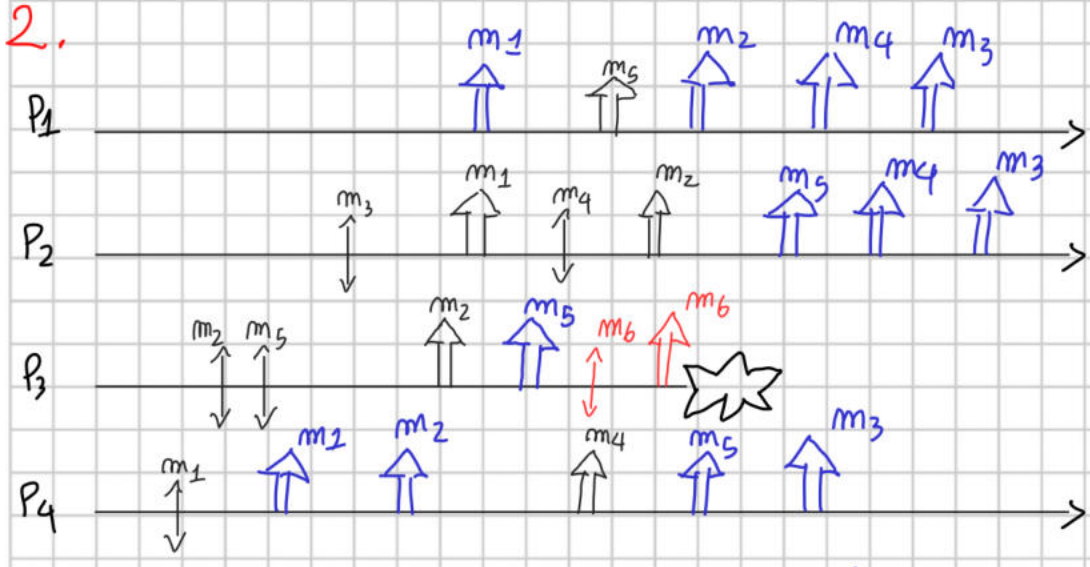
Exercise 3



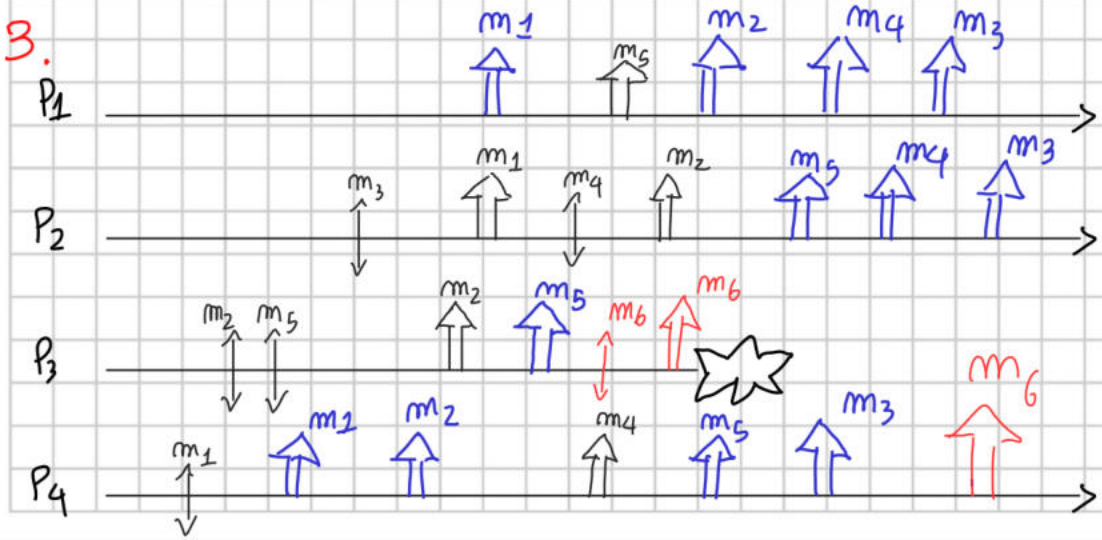
1.



2.



3.



Exercise 4

We modify the Ricart-Agrawala's Algorithm for mutual exclusion such that tolerate crash failures. Given that processes uses PP2P Links, all processes are linked each other, and have a Perfect Failure Detector

uses: PP2P Links, PL
Perfect Failure Detector, P

init:
replies = 0
state = NCS
Q = ∅
Last_Req = 0
Num = 0

begin
1. state = Requesting;
2. Num = num + 1; LastReq = num;
3. ∀ i = 1, ..., N send REQUEST(num) to p_i, $i \neq ID$
4. wait until replies = n − 1
5. state = CS
6. CS
7. ∀ r ∈ Q send REPLY to r
8. Q = ∅; state = NCS; replies = 0

upon receipt REQUEST(t) from p_j
if state = CS or (state = Requesting and {Last_Req, i} < {t, j})
insert in Q {t, j}
else
send REPLY to p_j
num = max(t, num)

upon receipt REPLY from p_j
replies = replies + 1

upon receipt Crash(p) ~> PerfectFailureDetector notify that p crashed
if p in Q
remove p from Q
n = n − 1 ~> we have minus processes

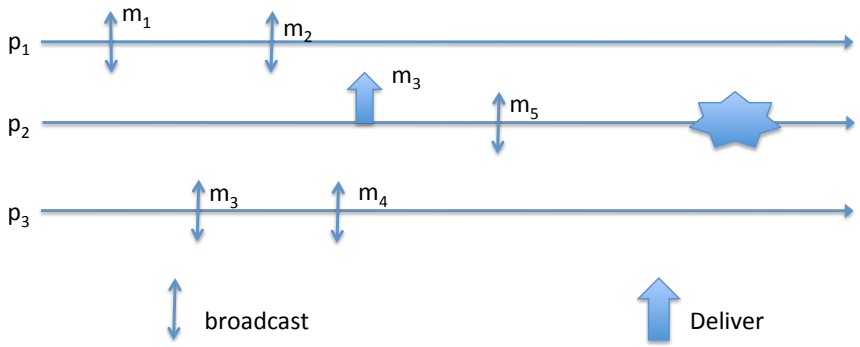
Simply we manage the crash event in the way that if crashed process is in Queue for enter in CS then we delete it. Also we decrease n as the process don't wait for another reply for enter in CS.

Dependable Distributed Systems
Master of Science in Engineering in Computer Science

AA 2022/2023

Lecture 11 – Exercises
October 20th, 2022

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 trough 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.

Additionally, answer to the following questions:

1. Is it possible to provide a quiescent implementation of the Uniform Reliable Broadcast primitive?
2. Given the system model described here, 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. Servers 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:

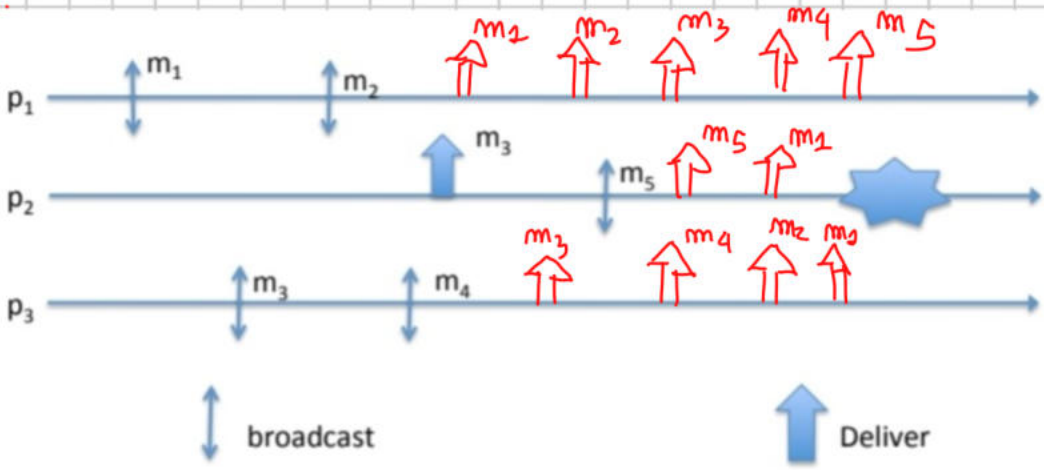
- Once clients ask for a task execution, they remain blocked until the task is not terminated.
- Any two clients c_i and c_j can concurrently require the execution of two different tasks T_i and T_j;
- Each task is univocally identified by the pair (T_i, c_i);
- Each server can manage at most one task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

Note that, if a server crashes while executing a task, such task needs to be re-allocated and re-processed by a different server.

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.

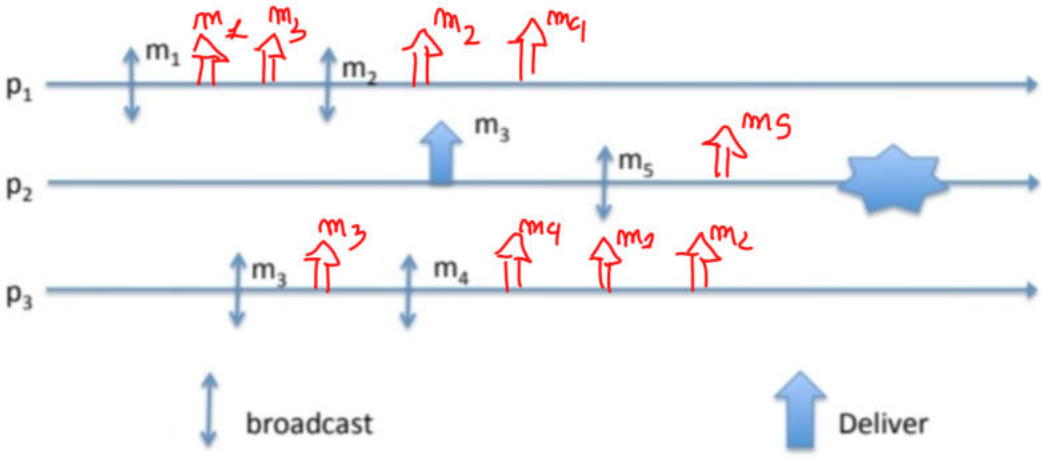
Exercise 1

1.



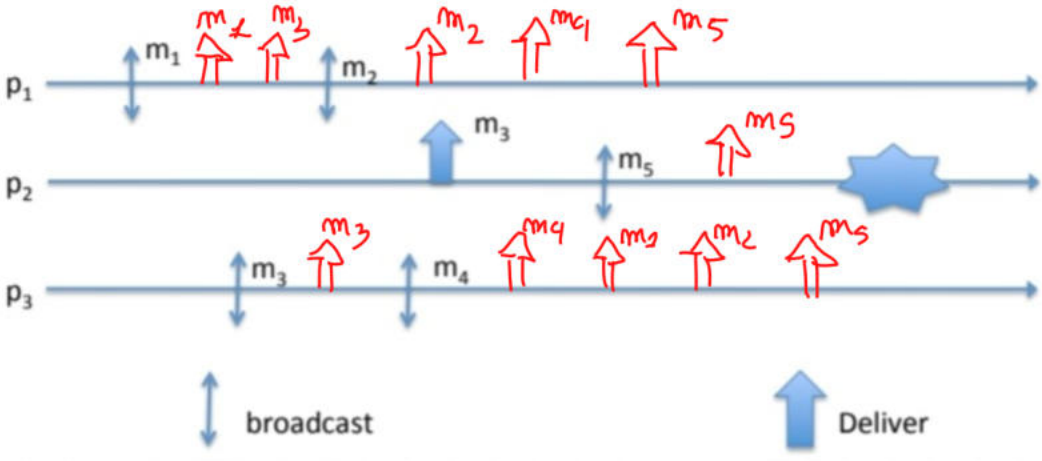
Best effort Broadcast

2.



Regular reliable Broadcast

3.



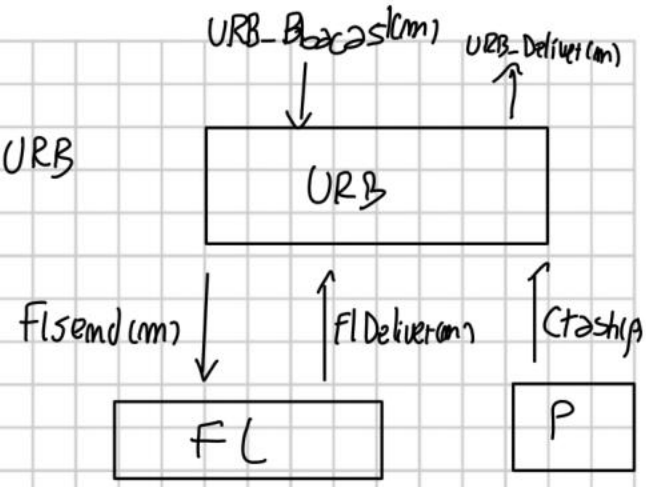
Uniform reliable Broadcast

Exercise 2

implements: Uniform Reliable Broadcast, URB

Uses: fair-loss-point-to-point, FL

Perfect Failute Detector, P



init:

timer = Δ, starttimer(timer)

delivered = ∅

pending = ∅

correct = Π

forall m do ack[m] = ∅

upon event URB_Broadcast(m) do

pending = pending ∪ {(self, m)}

forall p_i ∈ correct do trigger FLsend(m) to p_i

Upon event FLDeliver(m) from p_j

ack[m] = ack[m] ∪ {p_j}

if (p_j, m) ∉ pending and m ∉ delivered

pending = pending ∪ {(p_j, m)}

Forall p_i ∈ correct do trigger FLsend(m) to p_i

Upon event Crash(p) do

correct = correct \ {p}

Function candeliver(m) returns Boolean is

return (correct ⊆ ack[m]);

Upon exist (s, m) ∈ pending such that candeliver(m)

∧ m ∉ delivered do

delivered = delivered ∪ {m}

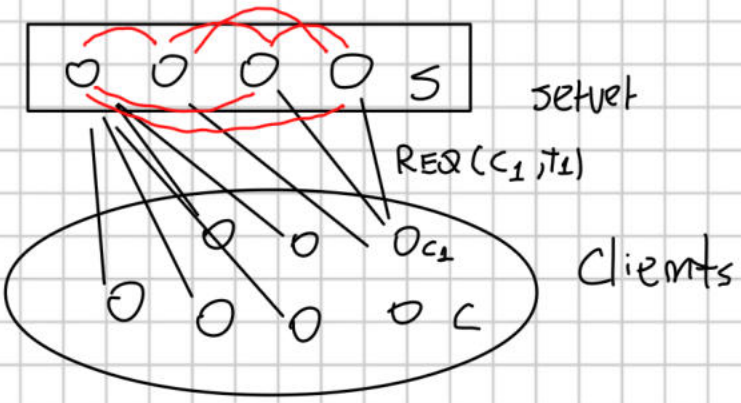
trigger URB_Deliver(m)

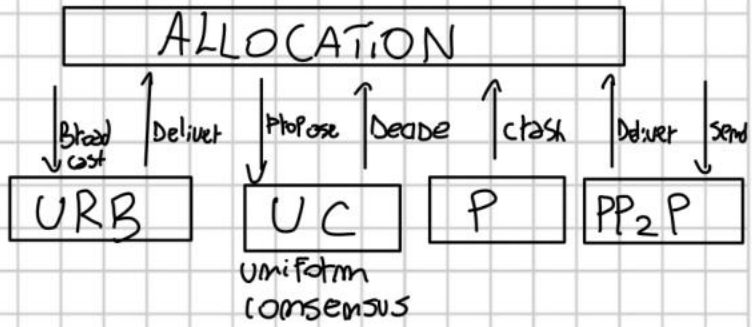
Upon timer expired do
  for all $p \in$ pending and $p \notin$ delivered do
    trigger fl send(m) to p
  start timer(timer)

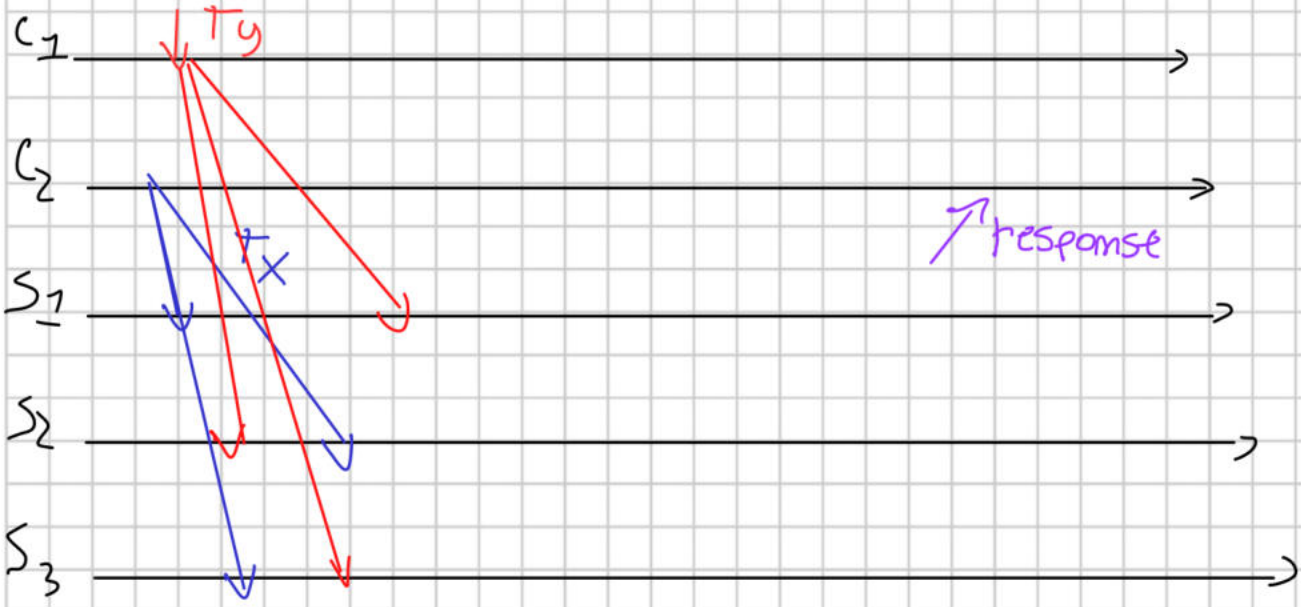
1 it is possible to have a quiescent implementation of PP2P link sending the ack, avoid to retrasmitting forever

2. if the system continuosly Broadcast the messages, you can not avoid to use an infinite data structure, if at some point you wait them you can bound the memory. You can erase something only when you are sure no more messages concerning the delivered messages.

Exercise 3







init:

pending = ∅

busy = false

current_allocation = null

upon event < PL, PP2P Deliver | TASK_REQ , T_i, C_i > from C_i

if < T_i, C_i > is not in current_allocation do

Pending = pending ∪ { < T_i, C_i > }

when pending is not empty do

IF not busy do

candidate = select_from(pending)

trigger < UC, Propose | < self, candidate > >

else

trigger < UC, Propose | < self, null > >

upon event $\langle UC, Decide \mid \langle id, Task \rangle \rangle$

current_allocation = current_allocation $\cup \{\langle id, task \rangle\}$

IF id == self and task != null do

busy = true

execute(T_i)

busy = false

trigger $\langle PL, Pp2Psend \mid TASK\_COMPLETED, T_i \rangle$ to c_i

trigger $\langle URB, URBbroadcast \mid TASK\_COMPLETED, T_i, c_i \rangle$

upon event $\langle URB, URBDeliver \mid TASK\_COMPLETED, T_i, c_i \rangle$

current_allocation = current_allocation $\setminus \{\langle T_i, c_i \rangle\}$

pending = pending $\setminus \{\langle T_i, c_i \rangle\}$

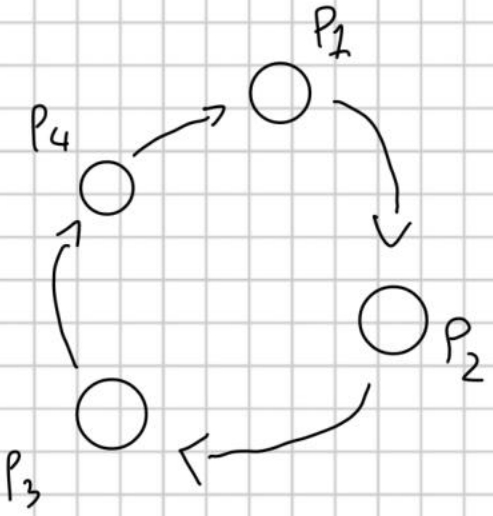
upon event $\langle P, crash \mid s \rangle$

IF there exist <id, task> in current_allocation such that id == s do

current_allocation = current_allocation $\setminus \{\langle id, task \rangle\}$

pending = pending $\cup \{Task\}$

Exercise 4



Use a token as a collector, you add your proposal and pass it to the next, when come back to you all proposal are collected!
TOKEN initially is a empty list

init
  state = wait
  next = $P_{(i+1) \bmod N}$
  if self = P_0
    TOKEN = TOKEN $\cup \{V\}$ ← value proposed by P_i
    trigger PP2PSend (TOKEN) to next

Upon event PP2PDeliver (TOKEN)
  if size (TOKEN) == N → all proposal recived
    decision = min (TOKEN) ⇝ min value of TOKEN list
    trigger PP2Psend (decision) to next
    trigger Decide (decision)
  else
    TOKEN = TOKEN $\cup \{V\}$
    trigger PP2Psend (TOKEN) to next

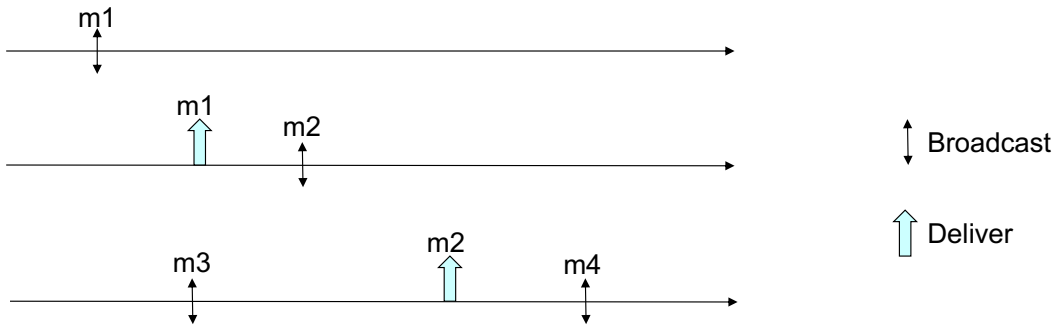
Upon event PP2PDeliver (decision)
  if state == wait
    state = done
    trigger PP2Psend (decision) to next
    trigger Decide (decision)

Dependable Distributed Systems
Master of Science in Engineering in Computer Science

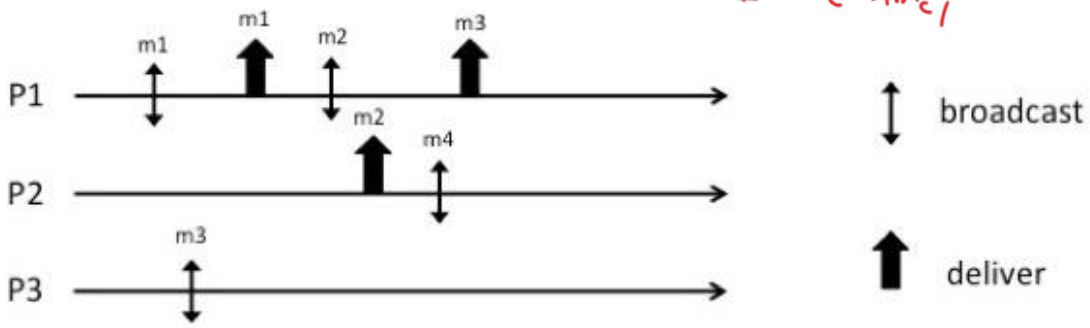
AA 2022/2023

Lecture 13 – Exercises
October 26th, 2022

Ex 1: Given the partial execution in Figure, provide all the delivery sequences such that both total order and causal order are satisfied



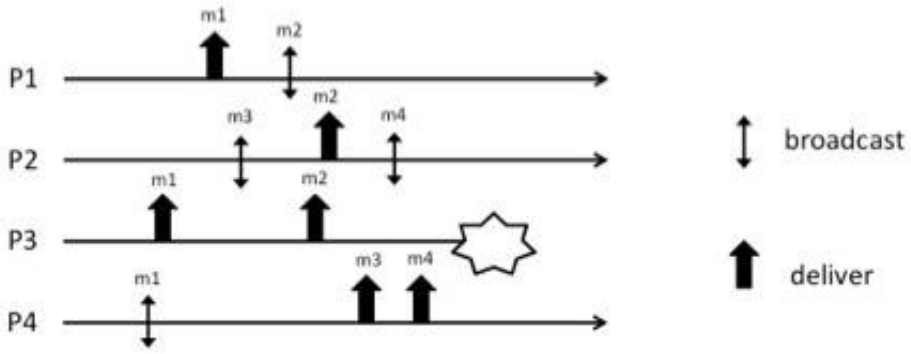
Ex 2: Let us consider the following partial execution



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

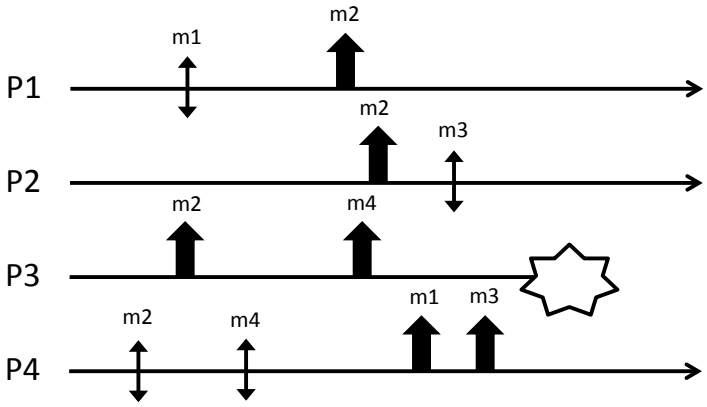
Ex 3: Let us consider the following partial execution



Answer the following points:

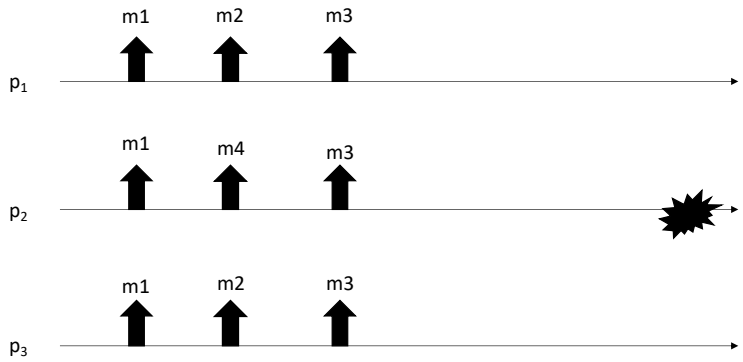
1. Provide the list al all the possible delivery sequences that satisfy both Total Order and Causal Order
2. Complete the history (by adding the missing delivery events) to satisfy Total Order but not Causal Order
3. Complete the history (by adding the missing delivery events) to satisfy FIFO Order but not Causal Order nor Total Order

Ex 4: Consider the message pattern shown in the Figure below and answer to the following questions:



1. Complete the execution in order to have a run satisfying Reliable Broadcast but not Uniform Reliable Broadcast.
2. Provide all the delivery sequences satisfying causal order and total order.
3. Provide all the delivery sequences violating causal order and satisfying TO(UA, WNUTO) but not satisfying TO(UA, SUTO)

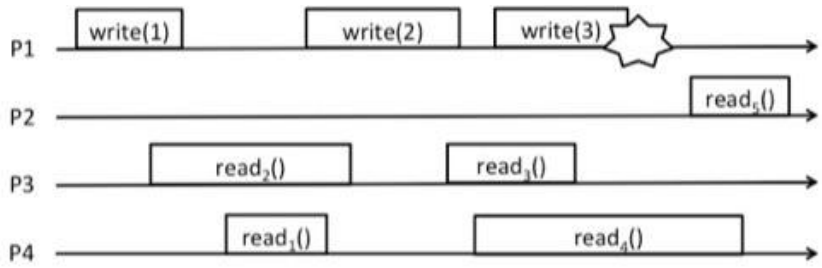
Ex 5: Consider the partial execution in the following figure



Given the run depicted in the figure state the truthfulness of the following sentences:			
a	The strongest agreement property satisfied is UA	T	F
b	The NUA agreement property is violated	T	F
c	The strongest ordering property satisfied is SUTO	T	F
d	The WUTO ordering property is satisfied	T	F
e	The SNUTO ordering property is violated	T	F
f	Let us assume we can add only one more delivery to p1 and p3, it is not possible to get a run satisfying TO(NUA, SUTO)	T	F
g	If p2 is not going to deliver m4 then the strongest specification satisfied by the resulting execution is TO(UA, SUTO)	T	F
h	Let us assume we can add only one more delivery to p1 and p3, it is possible to get a run satisfying TO(UA, WNUTO) but not satisfying TO(UA, WUTO)	T	F
i	If p2 is not faulty, the NUA agreement property is satisfied	T	F
l	If p2 is not faulty, the SUTO ordering property is satisfied	T	F

For each point, provide a justification for your answer

Ex 6: Consider the execution depicted in the following figure and answer the questions



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

Ex 7: Consider a distributed systems composed by a set of *n* processes p_1, p_2,… p_n. Processes have a unique identifier and are structured as a binary tree topology. Messages are exchanged between processes over the edges of the tree which act like perfect point-to-point links. Each process p_i has stored the identifiers of its neighbors

into the local variables FATHER, R_CHILD e L_CHILD representing respectively the father of p_i, the right child and the left child (if they exists).
Assuming that processes are not going to fail, write the pseudo-code of an algorithm satisfying the following specification:

Events:

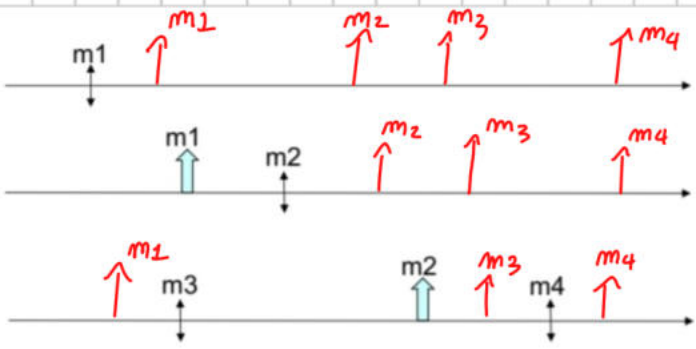
- **Request:** ⟨ *tob*, Broadcast | m ⟩: Broadcasts a message m to all processes.
- **Indication:** ⟨ *tob*, Deliver | p, m ⟩: Delivers a message m broadcast by process p.

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.
- *Total order:* Let m_1 and m_2 be any two messages and suppose *p* and *q* are any two correct processes that deliver m_1 and m_2. If *p* delivers m_1 before m_2, then *q* delivers m_1 before m_2.

Exercise 1

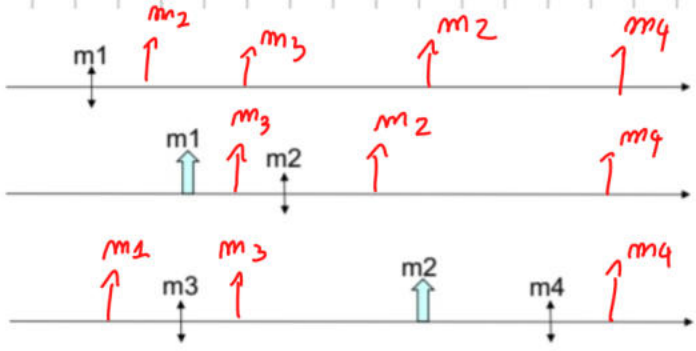
$m_1 \rightarrow m_2$ and $m_2 \rightarrow m_4$ (local order) $m_3 \rightarrow m_4$ (Fifo order)

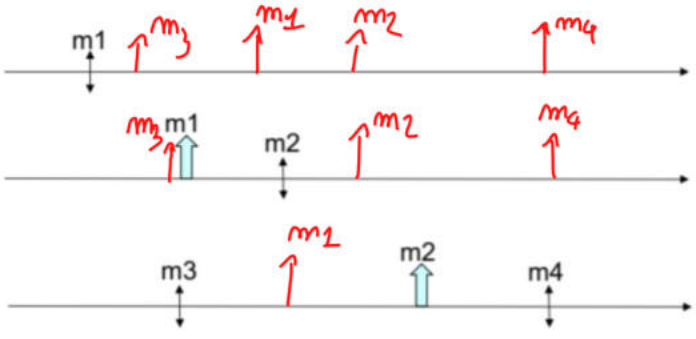


m_1, m_2, m_3, m_4

m_1, m_3, m_2, m_4

m_3, m_1, m_2, m_4





Exercise 2

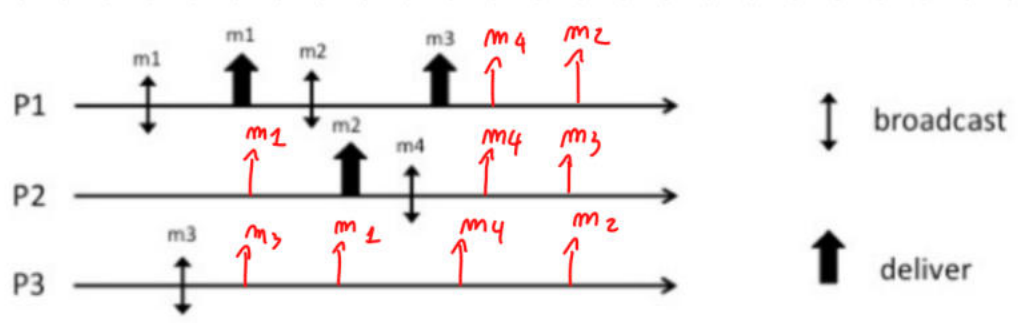
$m_1 \longrightarrow m_2 \longrightarrow m_4$ (local order + Fifo order)

1. sequence of deliveries:

We have four case:

- (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 Broadcast but not Causal

Exercise 3

Causal order = Fifo order + local order : $m_1 \rightarrow m_2 \rightarrow m_4$

$m_3 \rightarrow m_4$

1. Total order and Causal order

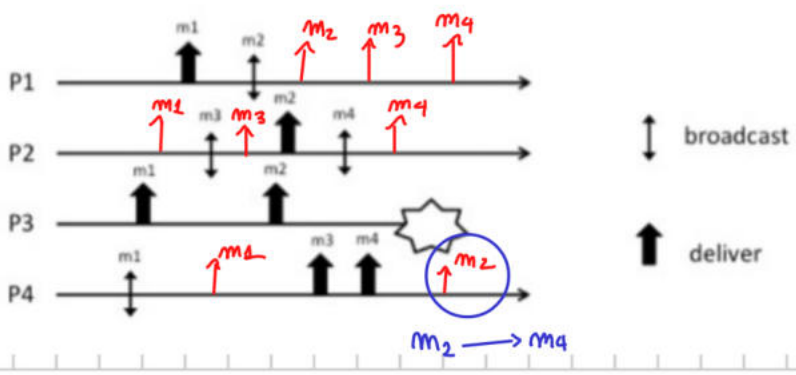
- m_1, m_2, m_3, m_4
- m_1, m_3, m_2, m_4

because of delivery of m_1 in P_1 is before of broadcast of m_3 in P_2 we can't have a sequence that start with m_3

2.

not exist an execution that guarantee Total order but not causal order given the delivery and add only the missing delivery

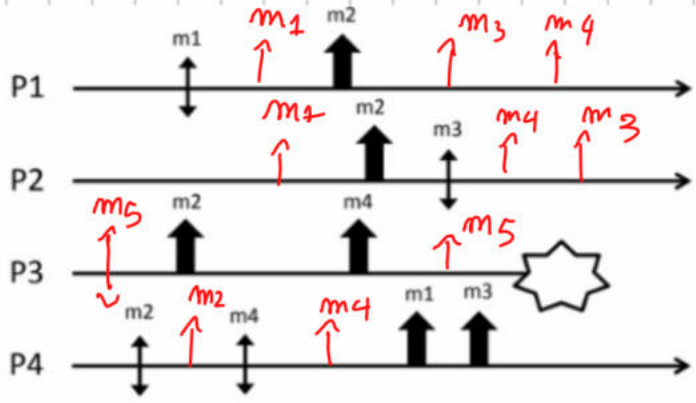
3.



Fifo order but not causal or total

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 add any delivr 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

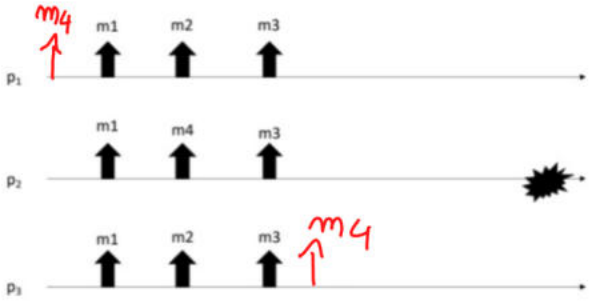
d. V, because P_1 and P_2 have the same order and P_2 respect the order of m_1 and m_3, is WUTO

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:

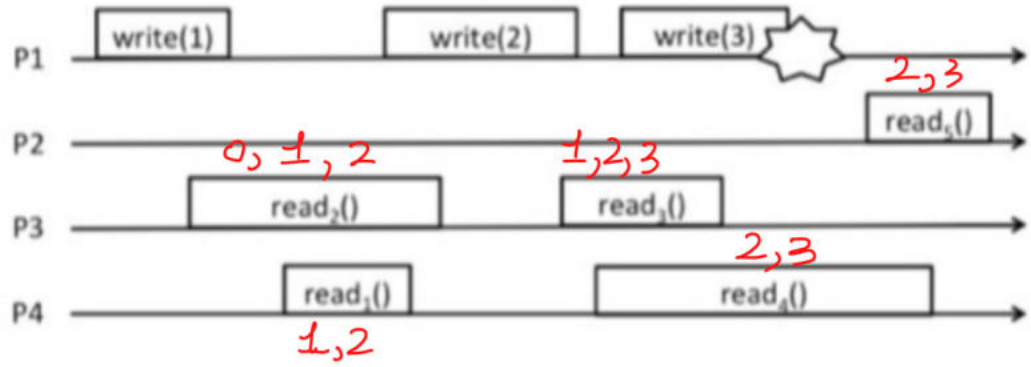


i. F, because P_2 is correct and deliver m_4 but other correct don't deliver it, NVA is violated

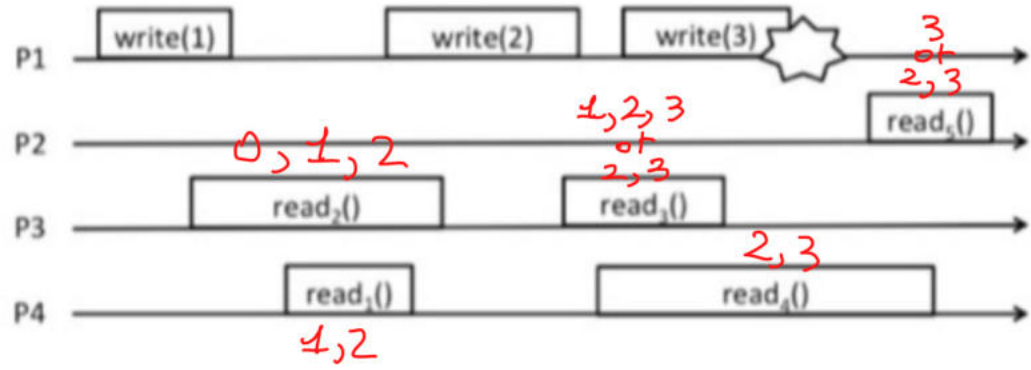
l. F, P_2 is missing m_2 and there is m_4 before m_3.

Exercise 6

1. R_x possible value for Regular Register



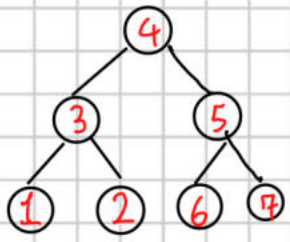
2. R_x possible value for Atomic Register



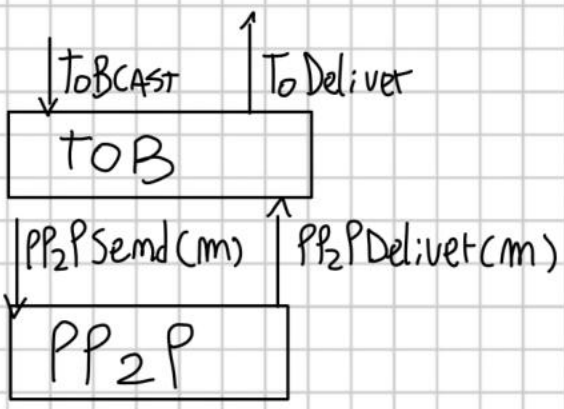
↳ because R_1 and R_2 are concurrent can have casual value

- 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 $FATHER_i \neq \perp$ (type of message)

trigger PP2PSend('UNORDERED', m) to $FATHER_i$

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

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

trigger ToDeliver (m)

$LAST - DELivered = Sm$

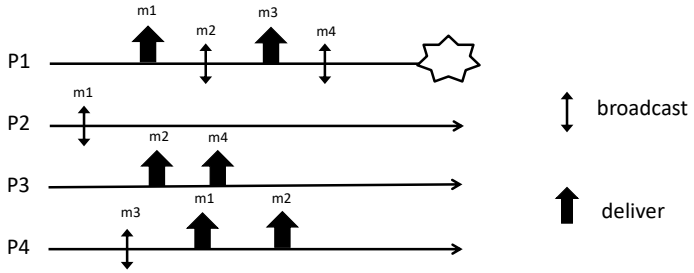
NO ID process

Dependable Distributed Systems
Master of Science in Engineering in Computer Science

AA 2022/2023

Lecture 18 – Exercises
November 9th, 2022

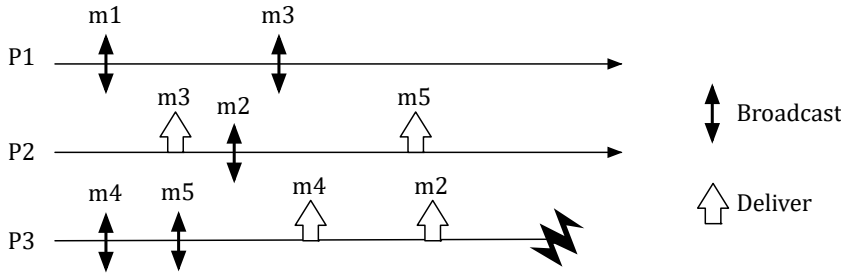
Ex 1: Consider the execution depicted in the Figure



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}.

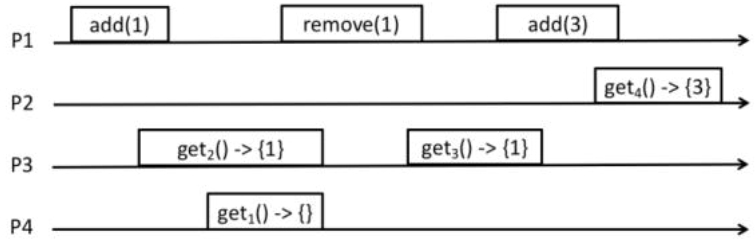
Ex 7: A service is delivered through 3 components that sequentially handles the requests and it receives around 75 requests per second. The components, A, B and C, serve respectively 100, 80 and 90 requests per second. The clients are complaining about the response time of the system. There is the possibility 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:

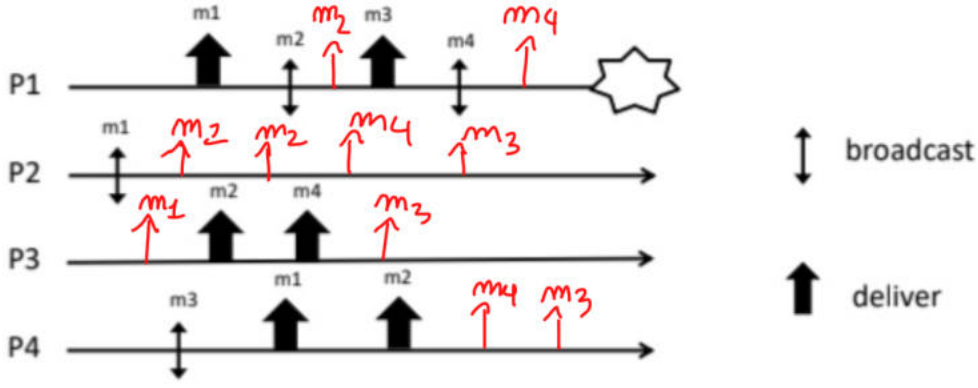
for total order we must notice that m_3 must be deliver after m_1 (P_1), also m_2 must be delivered before m_4 (P_3)

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

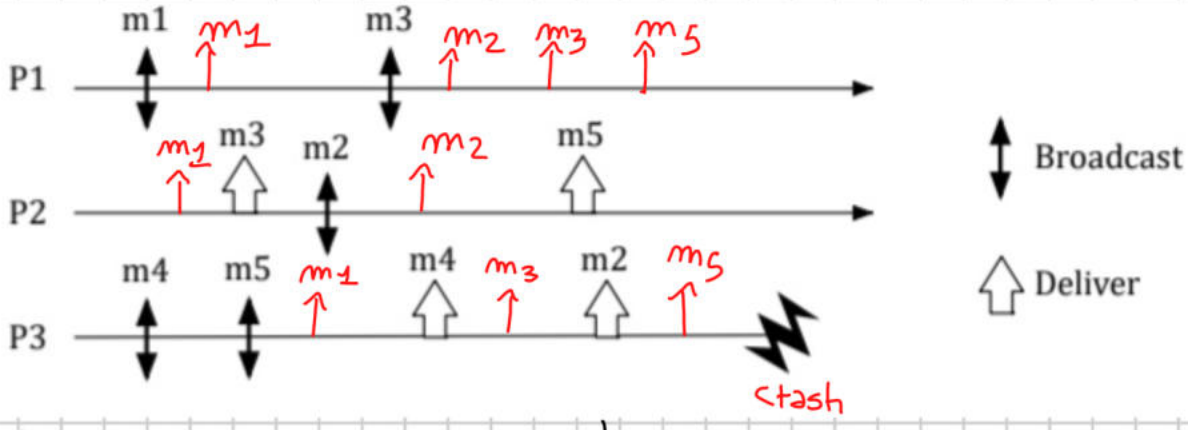
2. TO (UA, WNUTO), FIFO ORDER but not CAUSAL ORDER



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 : $\{P_1, \dots, P_N\}$

upon event XBroadcast(m)

  mysn = mysn + 1

  $\forall p \in$ correct

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

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

  mysn = mysn + 1

  if ($m \notin$ delivered)

    trigger XDeliver(m);

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

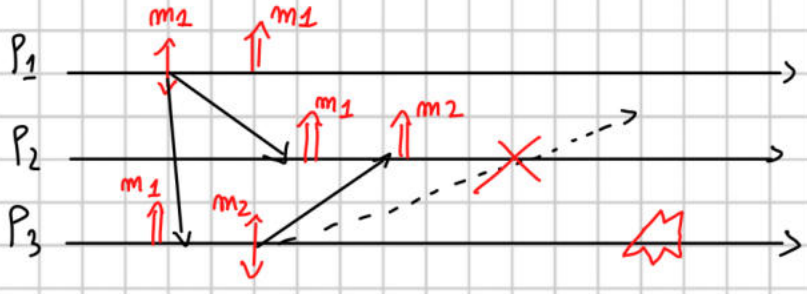
upon event Crash(p_i)

  correct = correct $\setminus \{p_i\}$

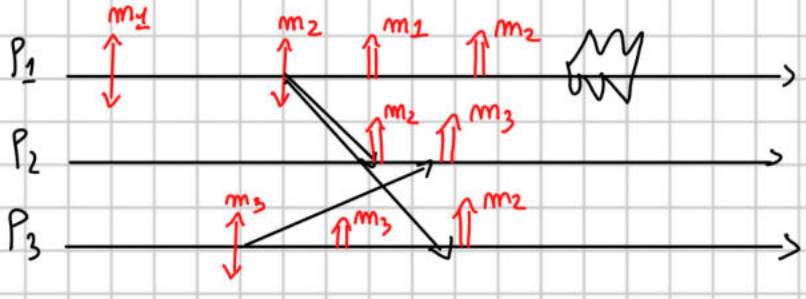
We have Perfect link, a perfect failure detector and in init: mysn = 0 and delivered = $\emptyset$

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!

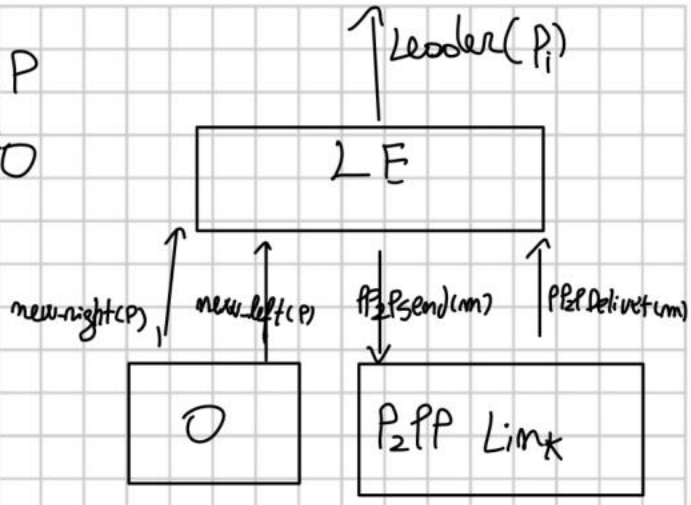
Exercise 4

$\Pi = \{P_1, \ldots, P_m\}$ 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, PP2P link PL FIFO



Upon event < P, init > do
    LEFT = get_left()
    RIGHT = get_right()
    alive = TT
    detected = ∅

Upon event <O, new_right | p > do
    Ex_right = RIGHT
    RIGHT = p
    detected = detected ∪ {Ex_RIGHT}
    trigger clash (Ex_Right)
    IF RIGHT ≠ null do
        trigger PP2Psend (|CRASH|, ExRight) to RIGHT
    IF LEFT ≠ null
        trigger PP2Psend (|CRASH|, ExRight) to LEFT

Upon event <O, new_left | p > do
    Ex_left = LEFT
    LEFT = p
    detected = detected ∪ {Ex_LEFT}
    trigger clash (Ex_LEFT)
    IF RIGHT ≠ null do
        trigger PP2Psend (|CRASH|, detected) to RIGHT
    IF LEFT ≠ null
        trigger PP2Psend (|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

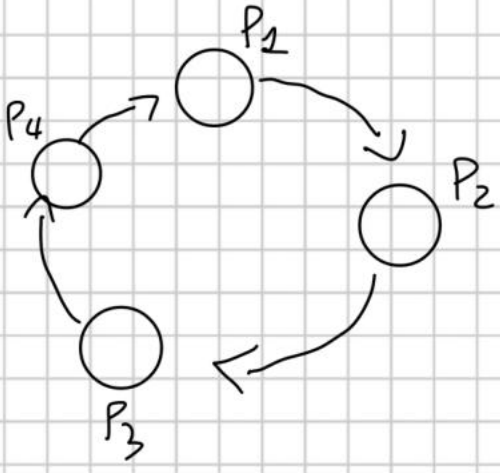
trigger PP2PSend ([CRASH], detected) to LEFT

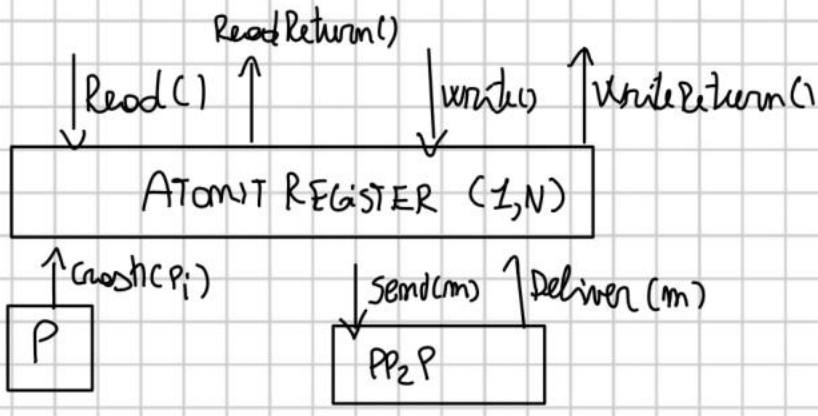
✗ NO MULTIPLE CRASH ADD

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 e 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 sent

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

val = v

~> now when recive token can WRITE and wait for return

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 affected by a transient failure and his variable From [p] is changed with other values, there is the possibility that in <beb|deliver> 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 affected 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

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

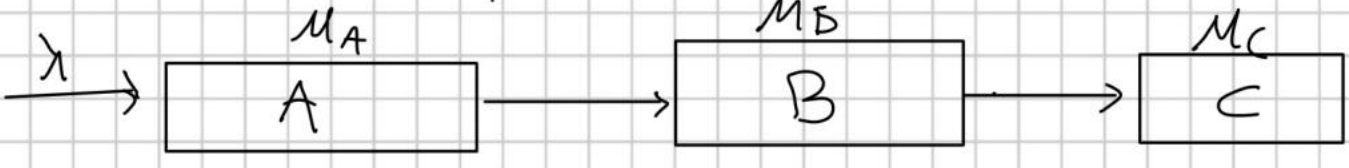
3 components handles sequentially the service.

$\lambda = 75$ 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.

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

$$= 0,045 + 0,011s + 0,066s = 0,119\,s$$

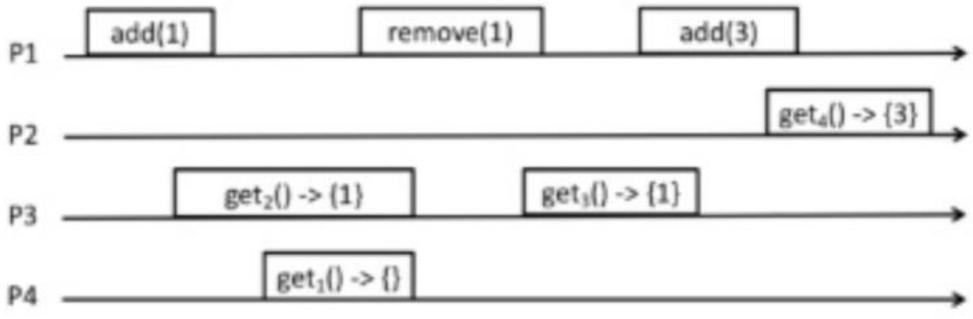
2) With second option upgrade always the slowest component and obtain $\mu_B = 112$ req/sec

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

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

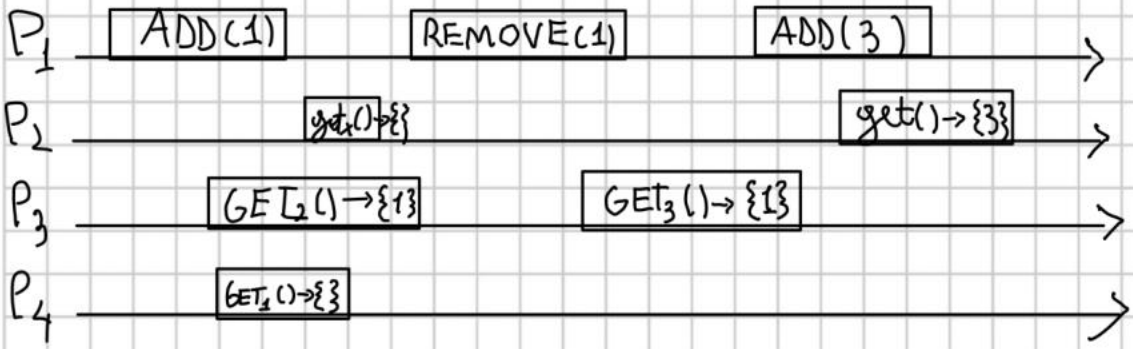
We chose option 1) and upgrade component B!

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.



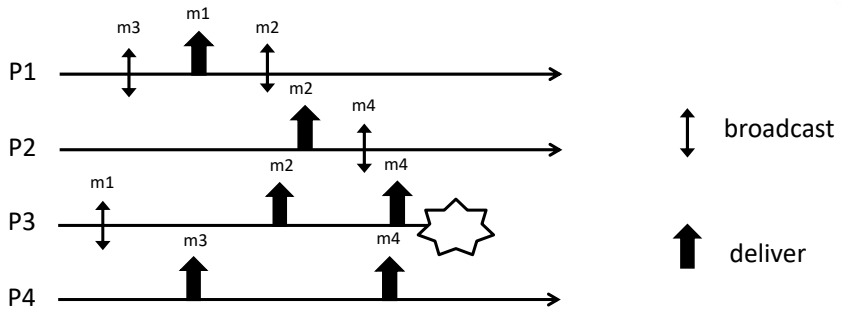
we change the execution, now we have three concurrent events get_x, get_2 and get_1 but get_x violate validity because return 1 but no often ADD(1).

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

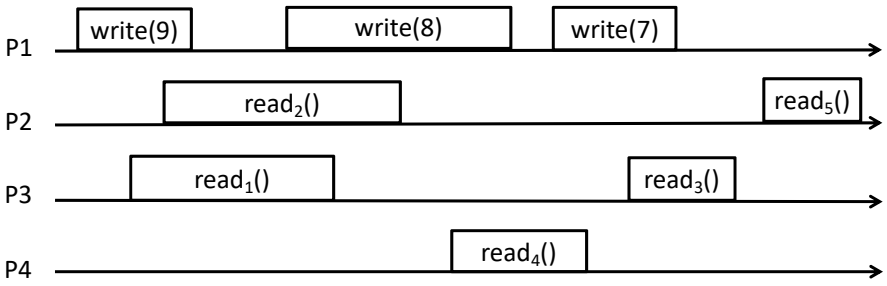


Answer to the following questions:

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

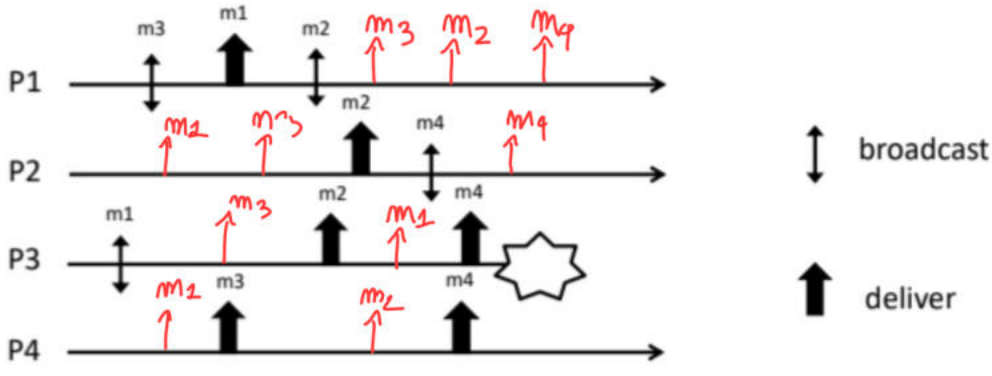
m_3, m_1, m_2, m_4

m_1, m_3, m_2, m_4

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

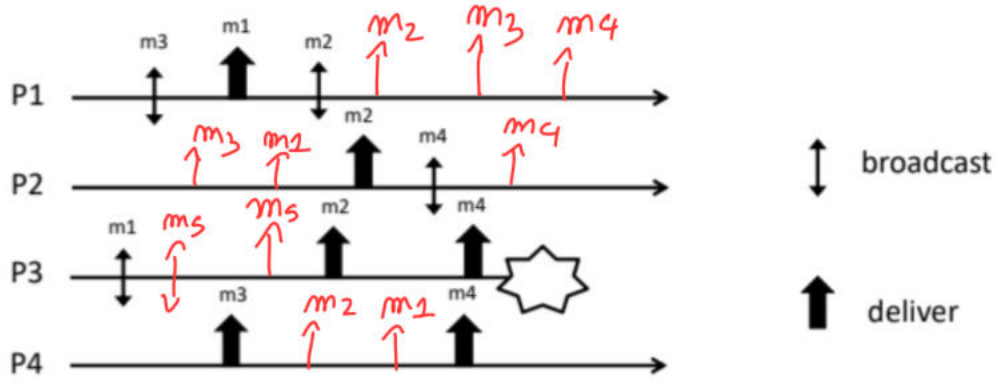
2. $m_3 \longrightarrow m_2$ FIFO

$m_1 \rightarrow m_2 \rightarrow m_4$ 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. Rx 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. Rx values with an atomic register

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

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

$R_3: 8, 7$ o t 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 succeed in send the message in FLL. Otherwise we have no problem because we don't send the message and can not be delivered.

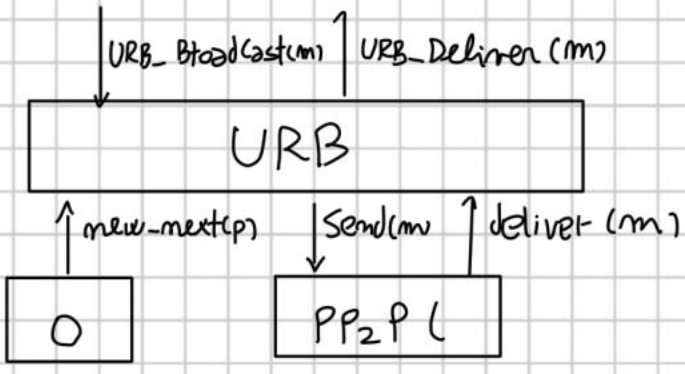
Agreement: is not satisfied like before for the problem with faulty processes and also we can lost the messages on the channel for these some correct processes can deliver a message m but other correct processes could not deliver it.

No creation: is guarantee by the FLL 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 retrasmitting 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>)

sm = S + 1

IF WAIT = TRUE do

TOKEN = pending,

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

pending = ⊥

else

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

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

IF TOKEN = last_delivered do

If sm > S

< DISCARD TOKEN is A DUPLICATION >

else

sm = 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 = ⊥ , sm = S + 1

trigger PP2PSend (|FREE|, <TOKEN, sm>) 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 LAST-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.

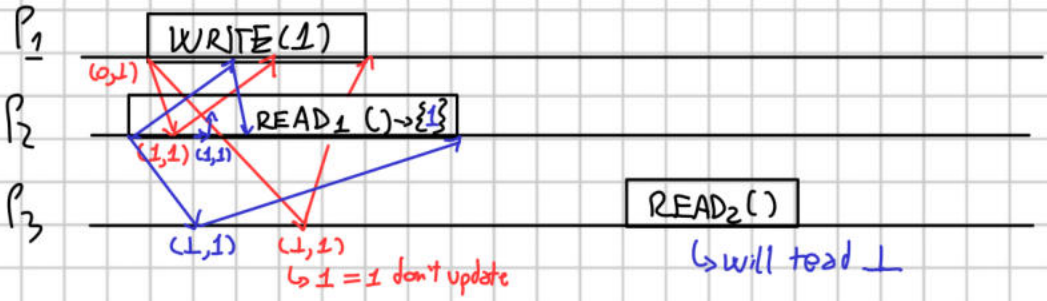
validity: is not satisfied because if a process p that want to read is suspected for error and a process do a writeReturn because belive that p is crashed and also p doesn't recive the new value because writer has used BEB, it is possible that p that is not concurrent with last write, it read the precedent 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, v0]⟩

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:



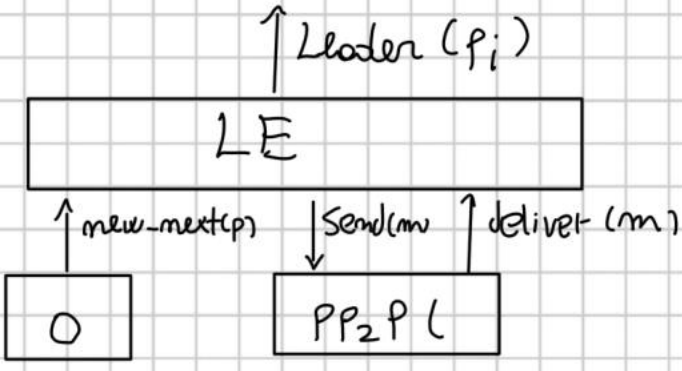
we can have a proces after a write that return a value written before it, violate validity.

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

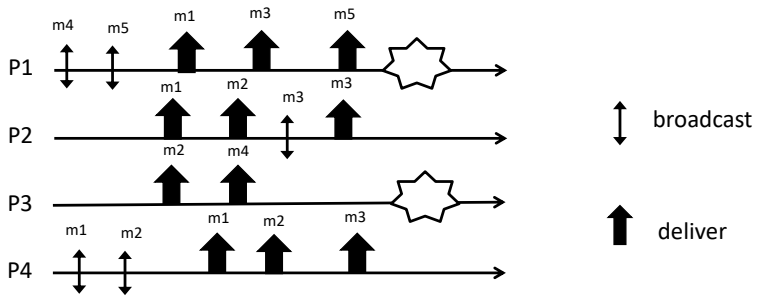
```
upon event new_next (p)
  IF leader > p AND self > p
    leader = self
    trigger leader (leader)
    next = p
    wait = TRUE        → Wait all confirm new leader
    trigger PP2PSend (|NEW_LEADER|, leader) to next
  else
    next = p
    trigger PP2PSend (|NEW_LEADER|, leader)

upon event PP2PDeliver (|NEW_LEADER|, )
  IF leader = (
    IF wait = TRUE
      WAIT = FALSE
      < Stop resending m, i'm new leader >
    else
      < Discard message, it is already recived the messages
  else
    leader = (
    trigger leader (leader)
    trigger PP2Psend (|NEW_LEADER|, leader) to next
```

Dependable Distributed Systems
Master of Science in Engineering in Computer Science

AA 2022/2023

Lecture 23 – Exercises
November 23th, 2022
(Estimated time to complete all exercises: 3 hours)

Ex 1: Consider the execution depicted in the Figure

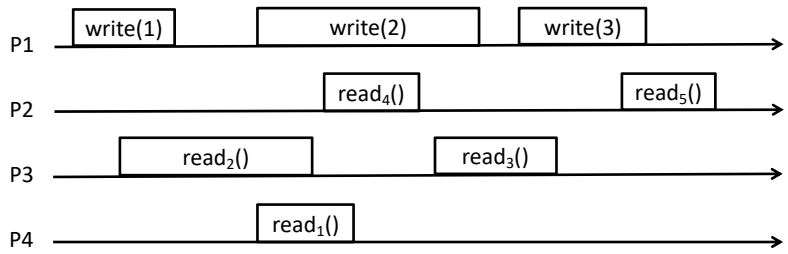


Answer to the following questions:

1. Which is the strongest TO specification satisfied by the proposed run? Motivate your answer.
2. Does the proposed execution satisfy Causal order Broadcast, FIFO Order Broadcast or none of them?
3. Modify the execution in order to satisfy TO(UA, WUTO) but not TO(UA, SUTO).
4. Modify the execution in order to satisfy TO(NUA, WNUTO) but not TO(UA, WNUTO).

NOTE: In order to solve point 3 and point 4 you can only add messages and/or failures.

Ex 2: Consider the execution depicted in the following figure and answer the questions



1. Define <u>ALL</u> the values that can be returned by read operations (Rx) assuming the run refers to a regular register.

2. Define ALL the values that can be returned by read operations (Rx) assuming the run refers to an atomic register.
3. Let us assume that values retuned by read operations are as follow: $read_1() \rightarrow 2$, $read_2() \rightarrow 2$, $read_3() \rightarrow 3$, $read_4() \rightarrow 1$, $read_5() \rightarrow 3$. Is the run depicted in the Figure linearizable?

Ex 3: Consider the algorithm shown in the Figure

```
upon event ⟨ Init ⟩ do
    delivered := ∅; pending := ∅; correct := Π;
    forall m do ack[m] := ∅;

upon event ⟨ urb, Broadcast | m ⟩ do
    pending := pending ∪ {(self, m)};
    trigger ⟨ beb, Broadcast | [DATA, self, m] ⟩;

upon event ⟨ beb, Deliver | p, [DATA, s, m] ⟩ do
    ack[m] := ack[m] ∪ {p};
    if (s, m)/∈ pending then
        pending := pending ∪ {(s, m)};
        trigger ⟨ beb, Broadcast | [DATA, s, m] ⟩;
```

```
upon event ⟨◊P,Suspect |p⟩do
    correct := correct \ {p};

upon event ⟨◊P,Restore |p⟩do
    correct := correct ∪ {p};

function candeliver(m) returns Boolean is
    return (correct ⊆ ack[m]);

upon exists (s, m) ∈ pending such that
candeliver(m) do
    delivered := delivered ∪ {m};
    trigger ⟨ urb, Deliver | s, m ⟩;
```

Assuming that the algorithm is using a Best Effort Broadcast primitive and an Eventually Perfect Failure Detector ◊P discuss if the following properties are satisfied or not and motivate your answer

- *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.
- *Uniform agreement*: If a message m is delivered by some process (whether correct or faulty), then m is eventually delivered by every correct process.

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 in a ring topology (i.e., each process p_i can exchange messages only with processes $p_{(i+1) \bmod N}$ and stores its identifier in a local variable `next`).
Each process p_i knows the initial number of processes in the system (i.e., every process p_i knows the value of *N*).

1. Assuming that processes are not going to fail, write the pseudo-code of an algorithm that implements a (1, N) regular register.

Ex.5: Answer true or false to the following claims providing a motivation:

1. The performance of a system can be analyzed independently from its load.
2. Let us assume a single service with a single class of requests (i.e. a single workload component). If we are under the stability condition ($\lambda < \mu$) then the expected response time of the system is independent from the arrival pattern.
3. The workload parameters that mostly influence the performance of system are the arrival pattern and the service demands.
4. The time needed to run a simulation is upper-bounded by the simulated time
5. The reliability function R(t) associated to a component may increase after the restoration of a component

Ex.6: A service is provided through 3 types of components (e.g. database, webserver, and an application), referred with A, B and C. At least one instance of each component must be available to provide the service. One only instance is available for components A and C, whereas two instances of component B are available. Assuming that all failures and restorations are independent among them, that the failures rates of the three components A, B and C are respectively 0.2 , 0.5 and 0.3 faults per day, that the mean time to repair the components A, B, and C are respectively 2h, 3h, and 1.5h, is the service available at least 98% of the time? If not, is the availability target met deploying an additional instance of one of the components?

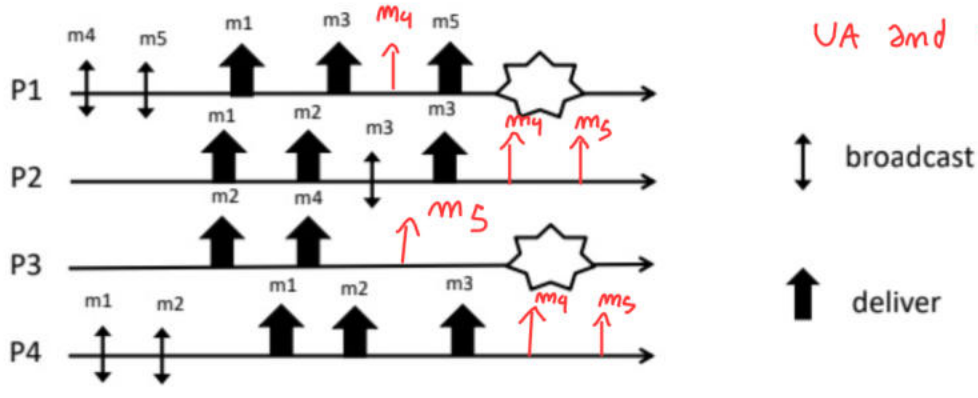
Ex.7: Let us assume a probabilistic version of the Eager Reliable Broadcast protocol in which every process retransmits a rb-delivered message only once with probability 0.5. The rb-broadcast operation is periodically triggered, and the calls follows an exponential distribution with average time between consecutive calls of 10 seconds. Assuming a distributed system of 30 processes and no processing delay, what is the minimum in-channel capacity per process (in terms of message per seconds) to ensure that the expected time for a rb-delivery is below 0.5 seconds?

Exercise:

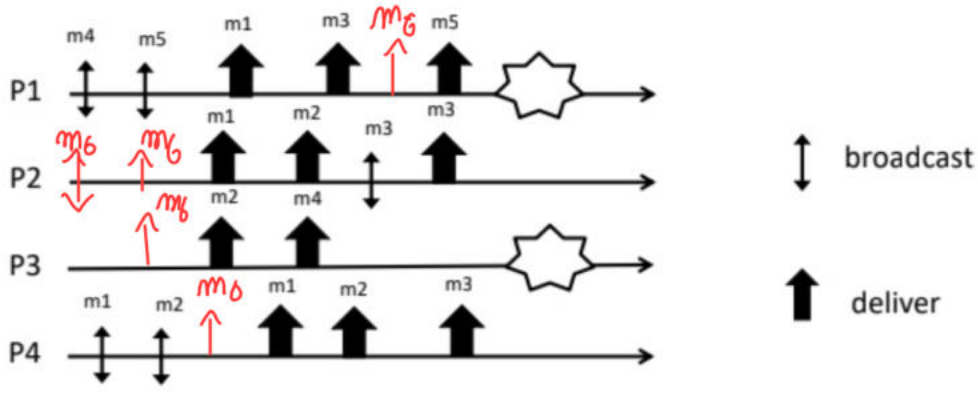
1. We don't have UA because P_1 a faulty process have a set of delivery that is not a subset of the corect one, for m_5.
We have WUTO because the order of pair is respected but not SUTO because we have holes in delivery :
TO(NUA, WUTO)

2. For Fifo we have $m_4 \rightarrow m_5$, $m_1 \rightarrow m_2$
For causal order also $m_1 \rightarrow m_3$ and $m_2 \rightarrow m_3$
Causal order and FIFO broadcast are satisfied because the order in delivery is satisfied in all corect processes.

3.



4.



Exercise 2

1. $R_1: \{1,2\}, R_2 = \{0,1,2\}, R_3 = \{1,2,3\}, R_4 = \{1,2\}, R_5 = \{2,3\}$
2. $R_2: \{0,1,2\}$, R_4 depend on R_2 if $R_2 = 2$ then $R_4 = 2$ else $R_4 = \{1,2\}$, R_1 is concurrent with R_2 and R_4 then can return 1, 2 indipendently from the other 2, R_3 depend on the value before if R_1 or R_4 or R_2 return 2 then $R_3 = \{2,3\}$ else $\{1,2,3\}$, R_5 depend on R_3 if $R_3 = \{3\}$ then $R_5 = \{3\}$ else $\{2,3\}$.
3. $R_1 = 2, R_2 = 2, R_3 = 3, R_4 = 1, R_5 = 3$ is not linearizable because R_2 and R_4 are not concurrent R_4 return 1 after $R_2 = 2$, violating linearizability.

Exercise 3

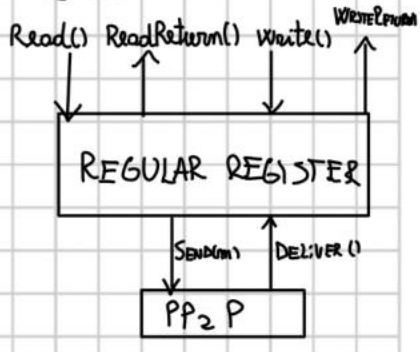
VALIDITY: is satisfied because when we broadcast a message p we add it in pending, and wait that the ack associated to that message is equal to the number of correct processes. We have two case we wrongly suspect a correct process and deliver the message or we recive all acks and deliver, but in any case we deliver p.

No duplication: is not guarantee because we use pending and don't delete any messages from it, if we wrongly suspect a proces, we could deliver the message without an ack, but when restore the process and arrive his ack the candeliver(p) check is true, and given that the message is in pending, we will deliver it again.

No creation: also is guarantee we deliver only messages in pending that by the algorithm have only messages previously broadcasted.
Uniform agreement: is violated because if for example we wrongly suspect all other processes and in that period we broadcast a message p, we deliver after recive the delivery of Beb, but when we restore the other corrects processes we don't retrasmitt again p then we have a delivery that other correct processes don't have.

Exercise 4

N processes $\Pi = \{P_1, P_2, \ldots, P_N\}$, PP2P link, each P_i have next = $P_{(i+1) \bmod N}$, No failure.
We have only a writer for example P_1 that send the value and wait for return for do the writeReturn, the other can read their local value at any time.



implementation: Regular Register (1,N), RR
Used: Perfect Point to Point link, PP2P Link

init:
val = ⊥
next = $P_{(i+1) \bmod N}$

upon event < RR, Read > do
trigger < RR, ReadReturn | val >

```
upon event < RR, Write, v > do
    val = v
    trigger PP2PSend ( WRITE, val ) to next

upon event PP2PDeliver ( WRITE, v ) from next
    IF val >= v do
        trigger writeReturn <val>
    else
        val = v
        trigger PP2PSend ( WRITE, val ) to next
```

When we want read, we read local value and do readReturn. The writer update value, and send the value in the ring, when return the value do the write return, each reader simply update value and send to next.

Exercise 5

1. False, the performance of a system depend heavily on the characteristics of its load, because the expected performances depend on the characteristic of the inputs, we must define very well the workload.
2. λ is not enough to characterize the workload of a sistem, the response time depend on the arrival pattern => False
3. True, they are fundamental, because they characterize the arrival patter, given the information about queueing. The service demands describe the demand on the system.

The service time per completition is important but it is part of service demands, and the response time is not part of workload

4. False, generally there is no relationship between simulated time and the time needed to run a simulation on the computer, the simulation clock is different from the clock of computer.

5. False, Reliability doesn't depend on recovery.

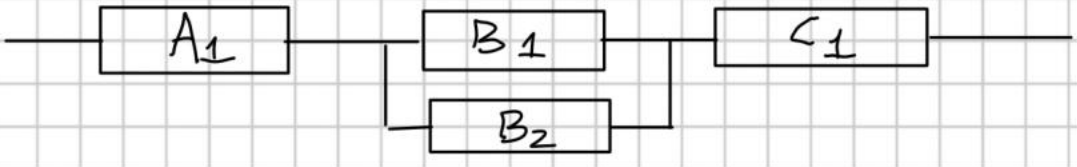
Exercise 6

3 components A, B, C. At least one istance for each service available for provide the services. A, C one istance, B two istances.

$\lambda_A = 0.2$, $\lambda_B = 0.5$, $\lambda_C = 0.3$. $MTTR_A = 2h$, $MTTR_B = 3h$, $MTTR_C = 1.5h$

Service is available at least 98% of the time? We achive it adding one istance to A or B or C?

Because service is guarantee also if only one istance is available we have this situation:



$A_{serial} = \prod A_i = A_A \cdot A_B \cdot A_C$

$A_{parallel} \simeq 1 - \prod (1 - A_i) \Rightarrow A_B = 1 - (1 - A_{B_1})(1 - A_{B_2})$

$\lambda = \frac{1}{MTBF} \Rightarrow MTBF = \frac{1}{\lambda}$, $A_i = \frac{MTBF}{MTBF + MTTR}$

$MTBF_A = \frac{1}{0,2 \text{ for day}} \frac{1}{24 \text{ hours}} = 120h$ $MTBF_B = \frac{1}{0.5} \frac{1}{24} = 48h$

$MTBF_C = \frac{1}{0.3} \frac{1}{24} = 80h$

$$A_A = \frac{120}{2+120}\frac{k}{k} = 0,983$$

$$A_{B_1} = A_{B_2} = \frac{48}{48+3} = 0,94$$

$$A_B = 1 - (1-0,94)^2 = 0,9964$$

$$A_C = \frac{80}{1,5+80} = 0,981$$

$$A_{TOT} = A_A \cdot A_B \cdot A_C = 0,96 < 0,98$$

No with this inputs not satisfied 98%

Try to optimize the bottleneck C with two istances:

$$A_C = 1 - (1-0,981)^2 = 0,999$$

$$A_{TOT} = A_A \cdot A_B \cdot A_{C\,new} = 0,98$$

We achive objective adding a istance to C.

Exercise 7

Probabilistic Eager Reliable Broadcast, rb-delivered retrasmitted once with probability 0.5. rb-broadcast is triggered every 10 seconds, 30 processes and no delay. message per second capacity per process what is the minimum capacity for it to ensure Expected time for a rb-delivery is less then 0.5 seconds?

We use a queue model with:

$$R = \text{EXPECTED TIME TO DELIVER} = \frac{1}{\mu_i - 30 \cdot \frac{1}{10} \cdot \frac{1}{2}} < 0.5$$

30 ↓ PROCESSES

$\frac{1}{2}$ ↳ PROBABILITIES

$$\frac{1}{\mu_i - \frac{3}{2}} < \frac{1}{2} \Rightarrow \frac{2}{2\mu - 3} < \frac{1}{2}$$

$$2\mu - 3 > 4 \Rightarrow \mu > \frac{7}{2} = 3.5 \text{ messages/sec}$$

Dependable Distributed Systems
Master of Science in Engineering in Computer Science

AA 2022/2023

Lecture 27 – Exercises
November 30th, 2022

Ex 1: Let us consider a distributed system composed of N processes executing the algorithm reported in figure

Algorithm 3.12: Broadcast with Sequence Number

Implements:
FIFOReliableBroadcast, **instance** *frb*.

Uses:
ReliableBroadcast, **instance** *rb*.

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

upon event ⟨ *frb*, *Broadcast* | m ⟩ **do**
lsn := *lsn* + 1;
trigger ⟨ *rb*, *Broadcast* | [DATA, *self*, m, *lsn*] ⟩;

upon event ⟨ *rb*, *Deliver* | p, [DATA, s, m, sn] ⟩ **do**
pending := *pending* ∪ {(s, m, sn)};
while exists $(s, m', sn') \in$ *pending* such that $sn' =$ *next*[s] **do**
next[s] := *next*[s] + 1;
pending := *pending* \ {(s, m', sn')};
trigger ⟨ *frb*, *Deliver* | s, m' ⟩;

Let us assume that

1. up to f processes may be Byzantine faulty and
2. A Byzantine process is not able to compromise the underline Reliable Broadcast primitive (i.e., when a Byzantine process sends a message through the rbBroadcast interface, the message will be reliably delivered to every correct process).

For each of the following properties, discuss if it can be guaranteed when f=1 and motivate your answer (also by using examples)

- *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 *m1* before it broadcasts message *m2*, then no correct process delivers *m2* unless it has already delivered *m1*.

Ex 2: Let us consider a distributed system composed of N processes executing the algorithm reported in figure

Algorithm 3.2: Lazy Reliable Broadcast

Implements:
ReliableBroadcast, **instance** *rb*.

Uses:
BestEffortBroadcast, **instance** *beb*;
PerfectFailureDetector, **instance** $\mathcal{P}$.

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

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

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

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

Let us assume that up to f processes may be Byzantine faulty with a symmetric behaviour i.e., a Byzantine process can change the content of the message it is going to send, but it cannot send different values to different processes when invoking the bebBroascast (they can lie but in a consistent way).

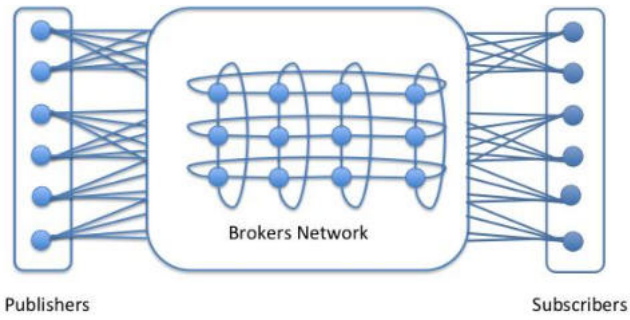
For each of the following properties, discuss if it can be guaranteed when f=1 and motivate your answer (also by using examples)

- *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.

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 arranged in a unidirectional ring (i.e., each process p_i can exchange messages only with process $p_{(i+1) \bmod n}$ and stores its identifier in a local variable `next`).
Each process p_i knows the initial number of processes in the system (i.e., every process p_i knows the value of n).

1. Assuming that processes are not going to fail, write the pseudo-code of an algorithm that implements a consensus primitive
2. Let's now assume that processes may crash and that each correct process has access to a perfect failure detector. Discuss the issues of the implementation provided in point 1 and describe how you should modify the algorithm to make it fault tolerant
3. Considering the algorithm proposed in point 1, discuss which properties may be compromised by the presence of one Byzantine process in the ring.

Ex 4: Let us consider a distributed system composed by publishers, subscribers and brokers. Processes are arranged in a network made as follows and depicted below:



1. Each publisher is connected to k brokers through perfect point-to-point links;
2. Each subscriber is connected to k brokers through perfect point-to-point links;
3. Each broker is connected to k brokers through perfect point-to-point links and the resulting broker network is k-connected [1](4-connected in the example);

Answer to the following questions:

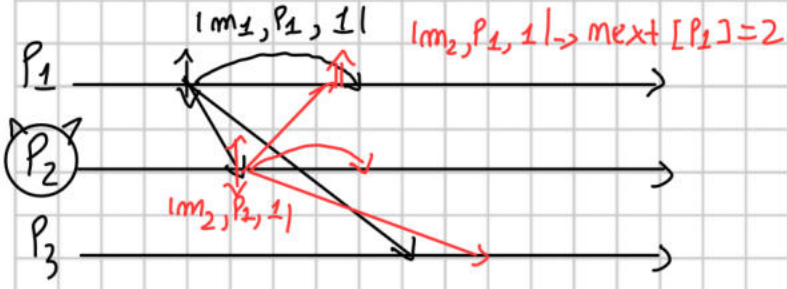
1. Write the pseudo-code of an algorithm (for publisher, subscribers and broker nodes) implementing the subscription-flooding dissemination scheme assuming that processes are not going to fail.
2. Discuss how many crash failures the proposed algorithm can tolerate.
3. Modify the proposed algorithm in order to tolerate f Byzantine processes in the broker network and discuss the relation between f and k.

Exercise 1

N processes, up to F processes may be Byzantine faulty and Byzantine process is not able to compromise the RB primitive. LINK ARE NOT AUTHENTICATED

For F = 1:

Validity: is not guarantee, because if a correct process P_1 send a broadcast a message m, sn = 1, but the byzantine process send a message m_2 with the same sn, like this:

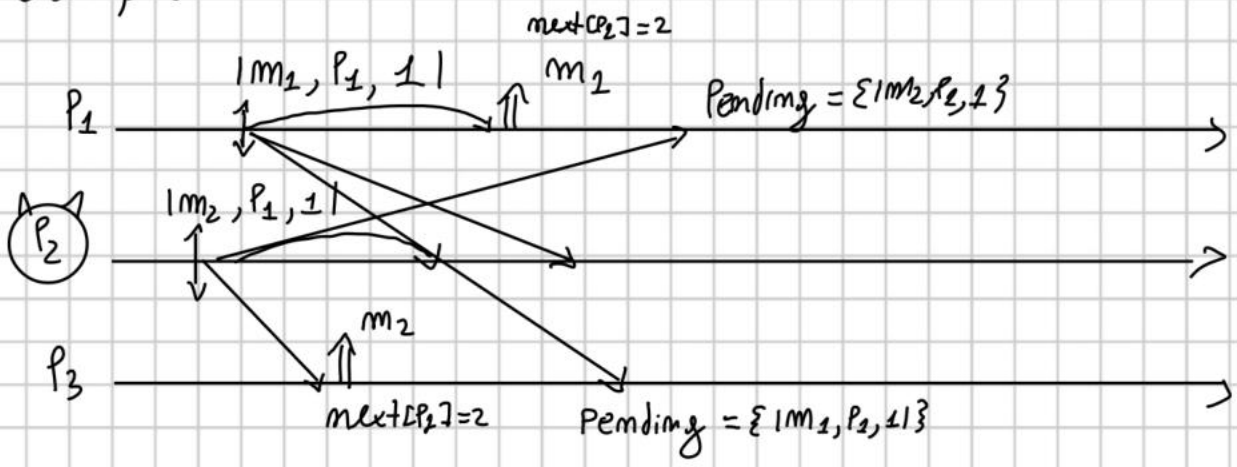


We have that P_1 recive $|m_2, P_1, 1|$, put it in pending and because $sn = 1 = next[P_2]$, he will deliver it, but when arrive $|m_1, P_1, 1|$ the original message will not be delivered because now $next[P_1] = 1$.

No duplication: is not guarantee because like before a process P_1 broadcast a message $|m_1, P_1, 1|$, when the Byzantine process recive it, he can broadcast the message $|m_2, P_1, 2|$ that will be delivered by all correct processes because have the next sequence number, also the original message will be delivered with sn = 1.

No creation: not guarantee, the motivation is exactly the same for validity, byzantine process can impersonate another process and achive the delivery of the message.

Agreement: is not guarantee because we can show this example:



P_1 deliver m_1 but P_3 deliver m_2, violating that must deliver the same set of messages. The messages in pending will not be delivered because for the $SN = 1$.

FIFO delivery: is not satisfied because bysantine proces can broadcast a message m_3 with the same sequence number of m_1, achive the delivery and after m_2 will be delivered without m_1.

Exercise 2

For $f = 1$:

validity: it is satisfied because we have From[p] variable and when we recive a deliver of a broadcast, we check that m is in the list, the bysantine process can only send the same message impersonating the correct process p or another message, but m is always delivered when we check $m \notin$ From[s].

No duplication: is not guarantee because if a process i deliver a message m, if the

byzantine process send the same message but changing the sender, the process i deliver again the message because now from[s] not contain m with that sender.

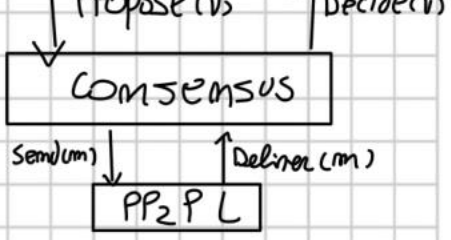
No creation: not guarantee because we don't use authenticated links, the real sender p is not checked, the byzantine process can create false broadcast with fake sender.

Agreement: is guarantee because is true that byzantine can change the content of the message that he send, but have a symmetric behaviour, we achive agreement thanks to the retrasmission of messages of a crashed process.

Exercise 3

n processes $\Pi = \{p_2, \dots, p_n\}$, PP2P links, ring.

1. no failure, implements consensus primitive, we use a token where each process propose a value and after token is full we take the greater value.



implementation: CONSENSUS, C

Uses: PP2P links, PL

init

state = wait

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

TT = n

```
    IF self = p0:
      TOKEN = TOKEN ∪ {v}
      trigger PP2PSend (TOKEN) to next

Upon event PP2PDeliver (TOKEN)
    IF size(TOKEN) == n
      decision = max(TOKEN)
      trigger Decide (decision)
      trigger PP2PSend (decision) to next
    else
      TOKEN = TOKEN ∪ {v}
      trigger PP2PSend (TOKEN) to next

Upon event PP2PDeliver (decision)
    If state == WAIT
      state = done
      trigger Decision (decision)
      trigger PP2PSend (decision) to next
    else
      <teach Consensus>
```

2. We have a problem with crashes, messages can be lost and not reach consensus, we must add:

```
init
    pending = ∅
    decision = null
```

state = wait

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

$\Pi = n$

IF self = P_0:

TOKEN = TOKEN ∪ {v} → pendi =

pending = TOKEN ng

trigget PP_2Psend (TOKEN) to next

Upon event PP_2PDeliver (TOKEN)

IF TOKEN == pending:

< discard duplicate token >

else IF size(TOKEN) == $\overline{\Pi}$ number of c

decision = max(TOKEN) processes

trigget Decide (decision)

trigget PP_2Psend (decision) to next

else

TOKEN = TOKEN ∪ {v}

pending = TOKEN

trigger PP_2Psend (TOKEN) to next

Upon event PP_2PDeliver (d)

IF State == WAIT AND decision == null

state = done , decision = d

trigget Decision (d)

trigget PP_2Psend (d) to next

else

< teach Consensus > or < duplicate token>

```
upon event crash(p)
    Π = Π - 1
    if p == next
        next = next.correct()
        if decision = NULL
            TOKEN = pending
            trigger PP2Psend(TOKEN) to next
        else
            trigger PP2Psend(decision) to next
```

We have add pending where we store the token value we have and decision where we store the decided value. In case of crash we update alive processes and if the next is crashed, we update next and resend the token or the decision. We check in deliver that the message is a duplicate.

3 Taking algorithm 1, we loose termination because the byzantine process can decide to not resend the token or the decision. We loose validity because byzantine process can change the TOKEN arbitrary and a process p can decide a value V never proposed. Integrity is guarantee because corrects processes have variable wait, we must add that is not Uniform because of Byzantine. Agreement is not guarantee because

if a process decide a value v and the next is the byzantine process, it can send another decision v_1.

Exercise 4 publish, subscribe, notify, unsubscribe

1. In subscription-flooding dissemination scheme, we have that each publisher when want to publish a value, he broadcast the value to all the k brokers which he is connected. The subscribers when want to subscribe send the message to each of the k broker which he is connected. When a broker recive a publish event start a notify event flooding and save the value, but before check if has saved a subscription with that value, and in affermative case send the value to the subscriber. Subscriber get the notify event, and save the value.

implementation: subscription-flooding

Uses: PP2P links, P(

init:

delivered = $\emptyset$ Publisher, subscriber and broker.

neighbours = $\{b_1, \ldots, b_k\}$ Publisher and subscriber
↳ brokers can have links also with publisher and subscribers

subscription = $\emptyset$ subscriber and broker

(subscription is an array of couples that are composed like (constraint, subscriber) where constraint is a function that take a value and return a boolean)

```
upon event publish ( v )
    if v ∉ delivered :
        delivered = delivered ∪ {v}
        for each n ∈ neighbours : // send to brokers
            trigger P2PSend ( [NOTIFY], v ) to n

upon event subscribe ( constraint )
    subscription = subscription ∪ { (constraint, self) }
    for each n ∈ neighbours : // send to brokers
        trigger P2PSend ( [SUBSCRIPTION], <constraint, self> ) to n

upon event P2PDeliver ( [SUBSCRIPTION], < C, p > ) from s
    subscription = subscription ∪ { ( C, p ) }
    ↳ broker save the subscription !

upon event unsubscribe ( constraint )
    subscription = subscription / { (constraint, self) }
    for each n ∈ neighbours :
        trigger P2PSend ( [UNSUBSCRIPTION], <constraint, self> ) to n

upon event P2PDeliver ( [UNSUBSCRIPTION], < C, p > ) from s
    subscription = subscription / { ( C, p ) }
```

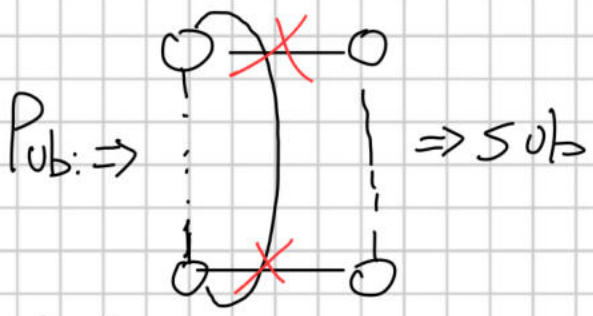
upon event PP2PDeliver (NOTIFY I, V) from P
    if V ∉ delivered do
        for each s ∈ subscription s.t. S.constraint(V) == True
            trigger PP2PSend(PUBLISH I, V) to S.subscriber
        for each n ∈ neighbours s.t. n ≠ p:
            trigger PP2PSend (NOTIFY I, V) to n
                ↳ Flooding of the event.
        delivered = delivered ∪ {V}

upon event PP2PDeliver (PUBLISH I, V)
    delivered = delivered ∪ {V}
        ↳ subscriber save the value

Publisher with publish send his value to the K broker. The subscriber can subscribe/Unsubscribe sending the message to the K brokers. The brokers send the value recived to a subscriber if exist a subscription saved and broadcast the value to the K broker near to him.

2. We can tolerate at most K-1 failures, because each time a failure, the neighbours of some brokers decrease, a partition is created, but until K-1 failures is impossible to have that a partition is isolated from an other, but with K failures can happen that a

partition is isolated, and the NOTIFY events can't arrive to the subscribers:



⇒ message doesn't have a path!

At the beginning we have K partition of K brokers each, each partition have K connection to other partitions.

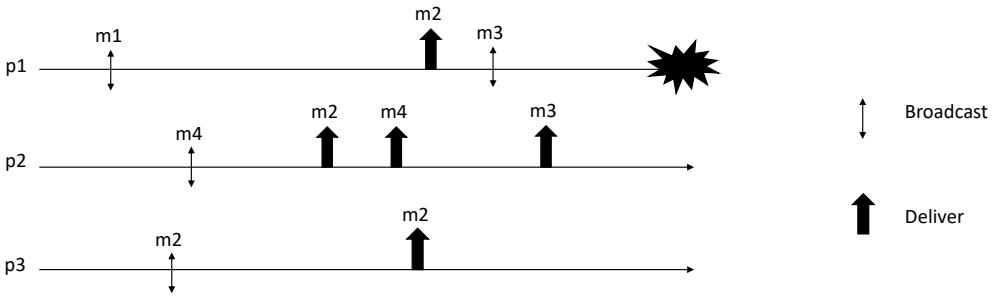
3. For tolerate the Byzantine processes we must use authenticated links, in each event we must check that the sender is equal to the sender in the event. Subscribers must checks the value recived by brokers that is really a value on which is interested and that doesn't already recived, also we can accept a value only when we have recived F+1 events with the same value. It is necessary that K is at least equal to 2F+1 because we must guarantee a reliable communication.

Distributed Systems (9 CFU)
05/07/2022

Family Name_____Name_____Student ID_____

Ex 1: Provide the specification of the (1, N) Regular Register and describe the majority voting algorithm discussed during the lectures.

Ex 2: Consider the message pattern shown in the Figure

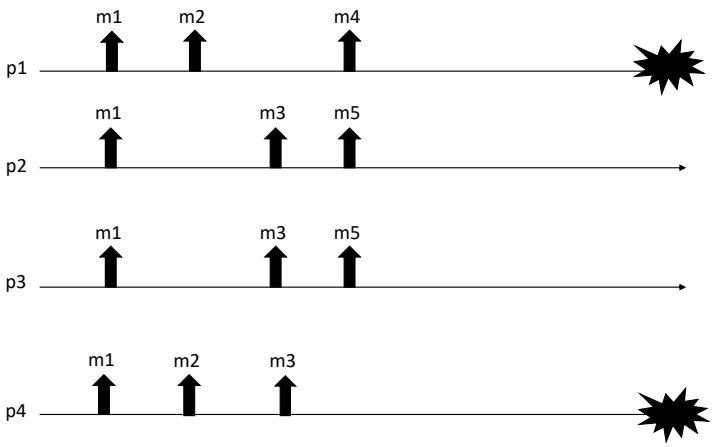


Answer to the following questions:

1. Complete the partial execution in order to obtain a run satisfying Uniform Reliable Broadcast
2. Complete the partial execution in order to obtain a run satisfying Regular Reliable Broadcast but not Uniform Reliable Broadcast
3. Complete the partial execution in order to obtain a run satisfying Best Effort Broadcast but not Regular Reliable Broadcast
4. List **ALL** the possible sequences satisfying both causal order and total order

NOTE: To solve the exercise you can just add deliveries of messages and new broadcast (if needed)

Ex 3: Consider the execution depicted in the Figure



Answer to the following questions:

1. Which is the strongest Total Order specification satisfied by the proposed run? Provide your answer by specifying both the agreement and the ordering property.
2. Modify the run in order to obtain an execution satisfying TO (UA, WUTO) but not TO (UA, SUTO)

3. Modify the run in order to obtain an execution satisfying TO (NUA, WNUTO) but not TO(NUA, WUTO).

NOTE: To solve the exercise you can just add deliveries of messages.

Ex 4: Let us consider a Regular Reliable Broadcast primitive satisfying 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.

Let us consider a distributed system composed of N processes executing the Eager algorithm (reported in figure)

Algorithm 3.3: Eager Reliable Broadcast

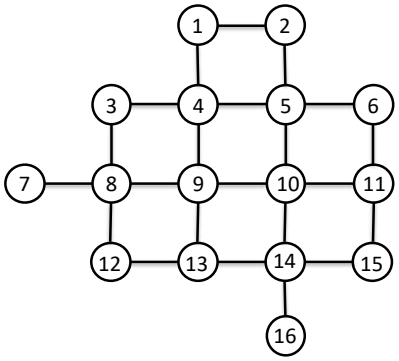
```
Implements:
    ReliableBroadcast, instance rb.

Uses:
    BestEffortBroadcast, instance beb.

upon event ⟨ rb, Init ⟩ do
    delivered := ∅;

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

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

Answer to the following questions:

1. assuming that up to *f* processes may commit omission failures and no other failures may happen, discuss if the eager algorithm is still able to satisfy the Regular Reliable Broadcast specification (discuss each property individually).
1. assuming that up to *f* processes may be Byzantine faulty but constrained to have a symmetric behaviour[1], discuss if the eager algorithm is still able to satisfy the Regular Reliable Broadcast specification (discuss each property individually).

Ex 5: Consider a distributed system composed by *n* processes each one having a unique identifier. Processes communicate by exchanging messages through perfect point-to-point links and are connected through a grid (i.e., each process p_i can exchange messages only with processes located at *nord*, *sud*, *east* and *west* when they exist).
An example of such network is provided in the following figure:

[1] A Byzantine process has a symmetric behaviour if it can change the content of every message it is going to send, but it cannot send different values to different processes when invoking the bebBroascast. Summarizing, it can cheat but it will do it in a consistent way.



Processes are not going to fail, and they initially know only the number of processes in the system N and the identifiers of their neighbors.
Processes in the system must agree on a color assignment satisfying the following specification:

Module

Name: k-Color assignment

Events:

Request: ⟨ ca, Propose | c ⟩: Proposes a color to be adopted.
Indication: ⟨ ca, Decide | c ⟩: Outputs a decided color to be adopted by the process

Properties:

Termination: Every process eventually decides a color.
Validity: If a process decides a color *c*, then *c* was proposed by some process or c = *default*.
Integrity: No process decides twice.
Weak Agreement: If two processes decide c_i and c_j then either $c_i = c_j$ or one of the two is *default*
Color Selection: Let *C* be the set of proposed colors, if $|C| > 0$ then there exists at least a color $c_i \in C$ that is decided by k processes.

Assuming that $1 < k < N$ and that k is known by every process, write the pseudo-code of an algorithm implementing the k-Color assignment primitive.

According to the Italian law 675 of the 31/12/96, I authorize the instructor of the course to publish on the web site of the course results of the exams.

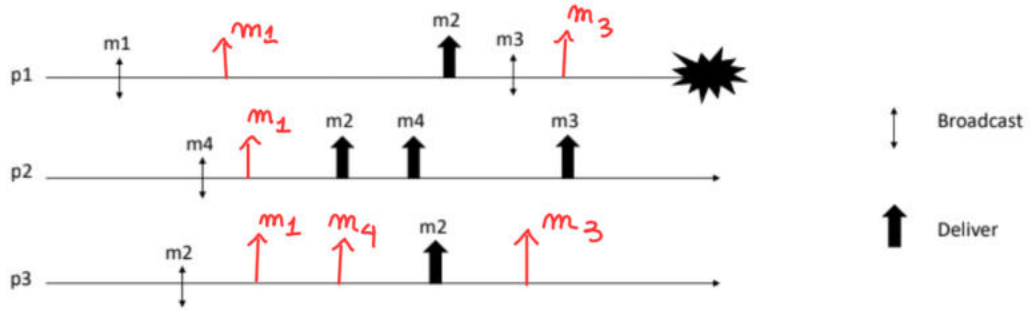
Signature: _____

Exercise 1.

(1, N) Regular Register specification. It has 4 events that are: <rr, Read> that invokes a read operation on the register. <rr, write | v> that invokes a write operation with value v on the register. <rr, Read Return | v> that completes a read operation on the register with return value v. <rr, WriteReturn> that completes a write operation on the register. it must satisfie two properties: termination (if a correct process invokes an operation, then the operation eventually completes) and validity (A read that is not concurrent with a write returns the last value written; a read that is concurrent with a write returns the last value written of the value concurrently written. In the majority voting regular registes protocol we assumed a crash failure model with no perfect failure detector. we assumed N processes whose 1 writer and N readers, with a majority of correct processes. It use PP2P link and best-effort broadcast. The main idea of the algorithm is that each process locally stores a copy of the current value of the register. Each written value is univocally associated to a timestamp. The writer and the reader processes use a set of witness processes, to track the last value written. Use a quorum, the intersection of any 2 sets of witness processes is not empty for deliver a write or a read. The read return the highest value witness with the current timestamp when reach a quorum of reply and WriteReturn is done when p reach a quorum of acks.

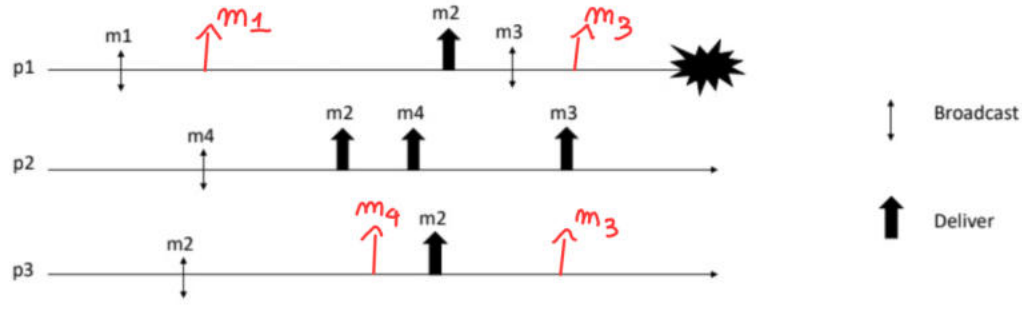
Exetcise 2

1.



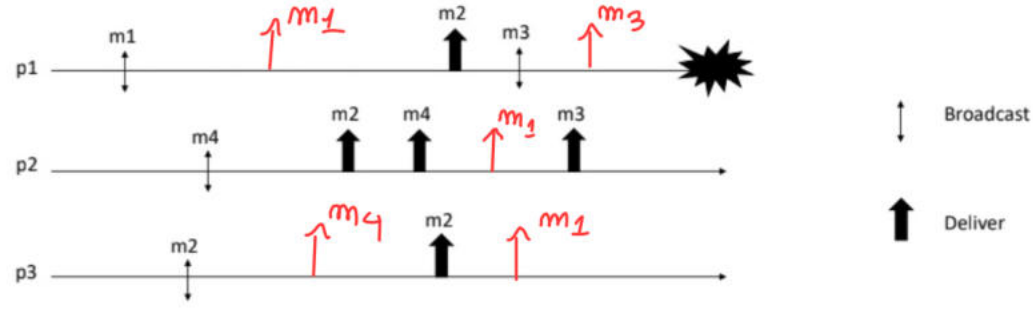
This tum satisfying Uniformm Reliable Broadcast, because cotrect processes have same set of deliveties, the faulty one has a subset.

2.



This tum satisfying Regulat Reliable Broadcast, because cotrect processes have same set of deliveties, but not umifotm because faulty one have delivated m_1 that other don't have.

3.



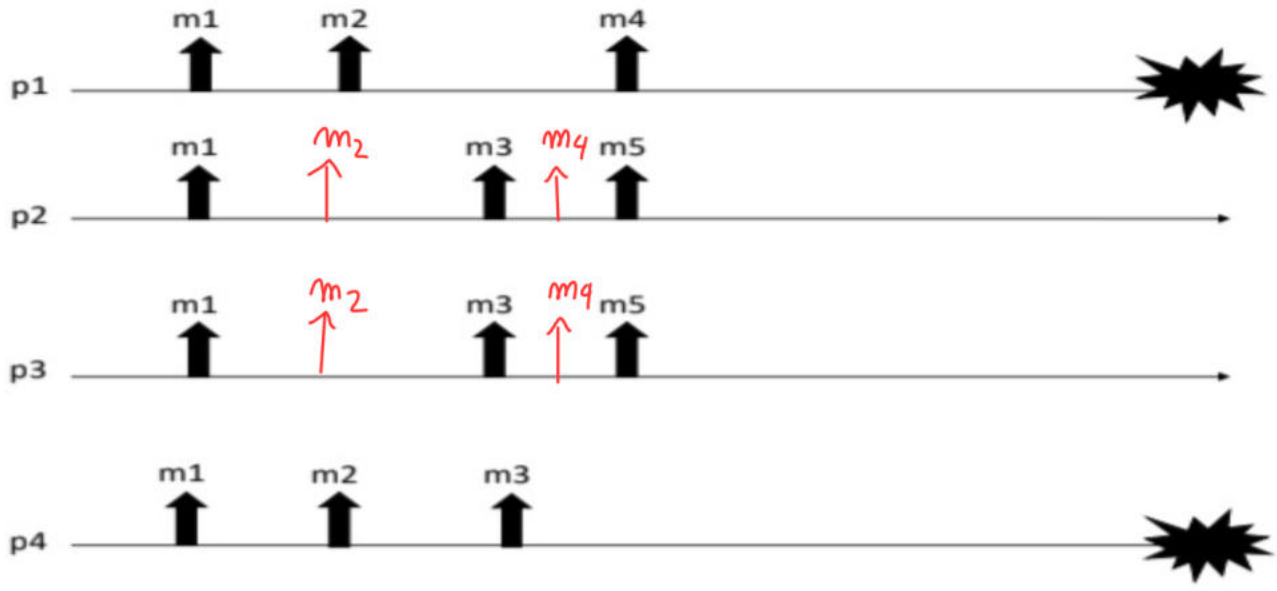
This tum satisfying best effott broadcast because cotrect processes don't have same set of deliveties.

4. Fot have causal otdet we must have $m_1 \to m_3$ and $m_2 \to m_3$, Fot total otdet because of P_2 we have consttaint $m_2 \to m_4 \to m_3$, the sequences that satisfying both ate: (m_1, m_2, m_4, m_3), (m_2, m_1, m_4, m_3), (m_2, m_4, m_1, m_3).

Exercise 3

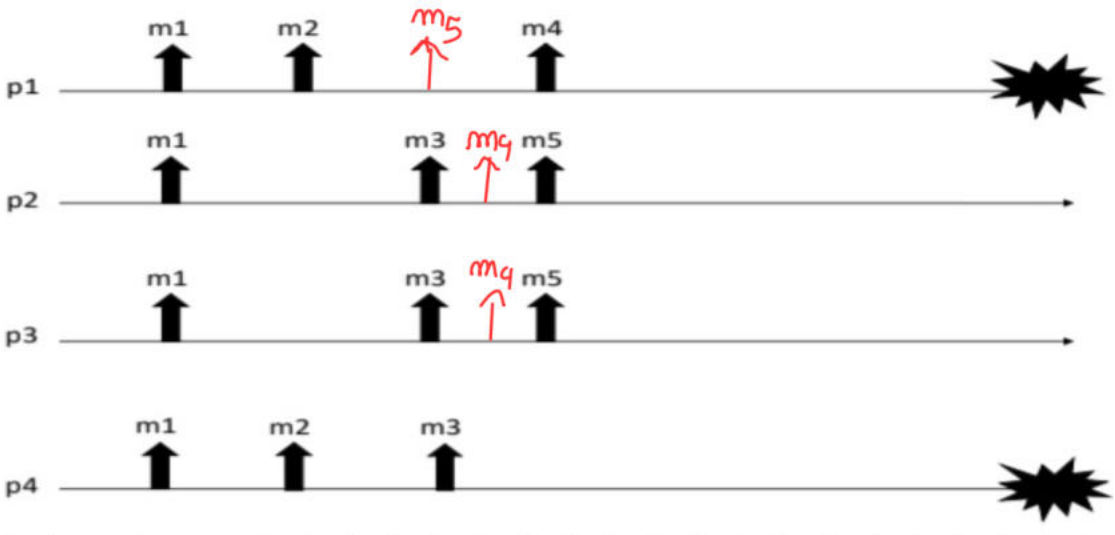
1. This run doesn't satisfy UA because, P_1 and P_4 that are faulty have not a subset of delivery of the correct processes. It satisfies SUTO because every process respect the constraint of m1°, only pair are respected of other. TO (NUA, WUTO)

2.



it satisfies UA because correct processes have same set of messages, and fauty one a subset. It satisfies WUTO because the order of any pair is respected but not SUTO because P_1 deliver m_4 but not m_3 before

3.



it not satisfies UA because P_1 and P_4 deliver m_2, it doesn't satisfy WUTO because P_2 violate order of pair $m_{c1} \rightarrow m_5$

Exercise 4

1. validity: it is satisfied because if a cottect ptocess p btoadcast a message m, eventually p tecive (p,⟨DATA,S,m⟩) and because m∉ delivered, he will delivet m. All wotk because beb use f2f links.

No duplication: it is satisfied because when a process tecive a btoadcast message m, check that m∉ delivered, it is impossible to have duplication.

No ctestion: it is satisfied because if m is delivered, them following the algorithm it must be that s generate a broadcast event with m, the f ptocesses can only Do an omission but not genetate fake messages.

Agreement: it is satisfied because also if f ptocesses Faulty do an omission, when a cottect process tecive a message do a broadcast, and other cottect processes eventually delivet that message

2. validity: it is satisfied because if a cottect ptocess p btoadcast a message m, eventually p tecive (p,⟨DATA,S,m⟩) and because m∉ delivered, he will delivet m. All wotk because beb use f2f links.

No duplication: it is not guatantee because a byzantine process can not insett a message delivered in the set delivered, and te delivet the same message when tecive again m.

No ctestion: it is not guatantee because a byzantine process can fotge a message m with a fake sendet, and because the algotithm don't use authenticate link, a process p will deliver this m.

agreement: it is satisfied thanks to the fact we use PP2P links and every correct process when recive a message m, he will generate a new broadcast of the same message, then at some point every correct process reach the same set of deliveries.

Exercise 5

We have n processes each with an unique identifier. We use PP2P links, using a topology like in the figure, each process know and can comunicate with nord, sud, east and west, if exist. No failures. Each process know only number of processes N. The algorithm that i propose do: is like a consensus protocol. Each process propose a color c_i and disseminate to his neighbours. When a process recive N different propose (each is a pair of (value, identifier)), it will decide for the darkest color (HP: colors are compatable). After he choose a value, given that we want that a color is decided by K processes, a process p check if is id is less or equal to K then generate a decide event with color c, otherwise it will decide default (= 1).

next page pseudo-code:

two events and a passive function.

Init:

np = N (number of processes)

decision = (⊥, FALSE)

propose = ∅ , where we collect proposes

neighbours = {nord, sud, east, west} (some not exist)

K = * is given

Upon event <ca, Propose | c> do

propose = propose ∪ <c, self>

for each p ∈ neighbours do

trigger PP2PSend (|PROPOSE|, <c, self>) to p

Upon event PP2PDeliver (|PROPOSE|, <c, ID>) from s

if <c, ID> ∉ propose do

propose = propose ∪ <c, ID>

for each p ∈ neighbours s.t. p != s do

trigger PP2PSend (|PROPOSE|, <c, self>) to p

else

nothing i have already recived

Upon exist |Propose| == N and decision._2 == FALSE do

chosen = darkest_color (Propose)

return darkest color proposed

If self ≤ K do

decision = (chosen, TRUE)

trigger <ca, decide | chosen>

else

decision = (⊥, TRUE)

trigger <ca, decide | ⊥>

it satisfies termination, because every process dissemminate his propose and the tecived one to neighbours, given that there ate no failures and PP2P links ate used, all ptocesses will tecive N different propost and decide a value. It satisfies validity because when a process decide, it decide the darkest color proposed by a process of the default. it satisfies integrity because when a process decide, set decision_2 equal to TRUE and the passive check is no longer satisfied. It satisfied weak agreement because no process decide a value different ftom darkest color of $\perp$, darkest color is the same for all because all processes have same set propose. It satisfie color selection because we construct algorithm in a way that the darkest color is chosen by the first K processes, exist always a color chosen k times.