OPERATIONAL LAW AND QUEUING THEORY

EXAMPLE 1

SINGLE SERVER S.

SOFTWARE COMPONENT A (single thread) manage each request in 0.1 s on average. Assume that:

1) on average 5 requests per second arrive at S
2) exponentially distributed of the arrivals
3) the service time of A follows an exponential distribution

COMPUTE THE EXPECTED RESPONSE TIME (how an user waits for a response)

λ = arrival rate = 5 req/sec

μ = service rate = $1/0.1$ = 10 req/sec

system is stable since $\lambda < \mu$

M/M/1

RESPONSE TIME:

$$R = \frac{1}{\mu - \lambda} = \frac{1}{10-5} = 0,2 \text{ sec}$$

EXAMPLE 2

TWO SOFTWARE COMPONENTS A and B SEQUENTIALLY. ON AVERAGE SOFTWARE COMPONENT A and B CAN HANDLE 10 and 6 REQUESTS PER SECOND. ASSUME:

1) 5 req/sec arrive
2) ARRIVALS ARE EXPONETIALLY DISTRIBUTED
3) SERVICE TIME EXP. DISTR.

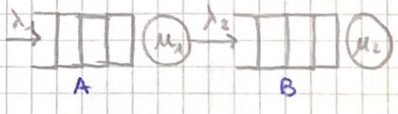
a) COMPUTE THE EXPECTED RESPONSE TIME.

b) CAN THE SYSTEM HANDLE A WORKLOAD OF 8 REQUESTS/SEC

If not, it's possible to vertically scale one of the component doubling its service rate?

$\lambda = 5$ req/s

$\mu_1 = 10$ req/s $\quad \mu_2 = 6$ req/s



$$R = R_1 + R_2 = \frac{1}{\mu_1 - \lambda} + \frac{1}{\mu_2 - \lambda} = \frac{1}{10-5} + \frac{1}{6-5} = 1,2 \text{ s}$$

B is acting as a bottleneck, so if we want to improve performance of the system we need to improve it.

if $\lambda = 8$ req/s → system is not stable since $\lambda > \mu_2$

so $\mu_2' = \mu_2 \cdot 2$ doubling service

$$R = \frac{1}{10-8} + \frac{1}{12-8} =$$

EXAMPLE 3

TWO SOFTWARE COMPONENT A e B. A caches part of data and B caches all the data.

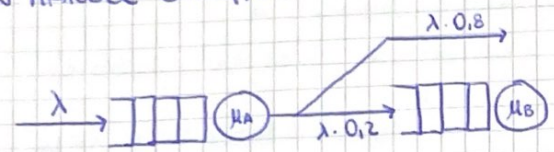
Probability of cache missing is 0.2

$\mu_A = 10$ req/sec $\mu_B = 6$ req/s

1) $\lambda = 5$ req/s

COMPUTE THE EXPECTED RESPONSE TIME

CAN HANDLE 8 req/s? IF NOT SCALE AND COMPUTE THE RESPONSE TIME.



$$R(\text{CACHE HIT}) = R_A = \frac{1}{10-5}$$

$$R(\text{CACHE MISS}) = R_A + R_B = \frac{1}{\mu_A - \lambda} + \frac{1}{\mu_B - 0.2\lambda} = \frac{1}{10-5} + \frac{1}{6-\frac{5}{5}}$$

$$R = 0.8 \cdot R(\text{CACHE-HIT}) + 0.2\ R(\text{CACHE-MISS})$$

EXAMPLE 4 - AVAILABILITY

λ_A = FAILURE RATE = 0.2 fault/day

λ_B = 0.5 fault/day

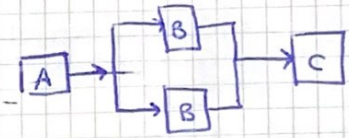
λ_C = 0.3 fault/day

$MTTR_A$ = 2h

$MTTR_B$ = 3h

$MTTR_C$ = 1.5h

$A > 98\%$!



$$A = \frac{MTBF}{MTBF + MTTR}$$

$$MTBF_A = \frac{1}{\lambda_A}$$

$$MTBF_B = \frac{1}{\lambda_B}$$

$$MTBF_C = \frac{1}{\lambda_C}$$

$$A_1 = \frac{MTBF_A}{MTBF_A + MTTR_A}$$

$$A_2 = 1 - \left(1 - \frac{MTBF_B}{MTBF_B + MTTR_B}\right)^2$$

$$A_3 = \frac{MTBF_C}{MTBF_C + MTTR_C}$$

$$A_{TOT} = A_1 \cdot A_2 \cdot A_3$$

WEEK 2 EXERCISES

1. EXTERNAL SYNCHRONIZATION IS WHEN PROCESSES SYNCHRONIZE THEIR CLOCK C_i WITH THE UTC SERVER.
LET $D>0$ BE THE SYNCHRONIZATION BOUND AND S BE THE SOURCE OF UTC, THE CLOCKS C_i (for $i=1,\dots,N$) ARE EXTERNALLY SYNCHRONIZED IF

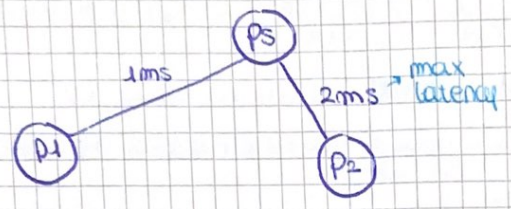
$|S(t) - C_i(t)| < D$ for each time interval

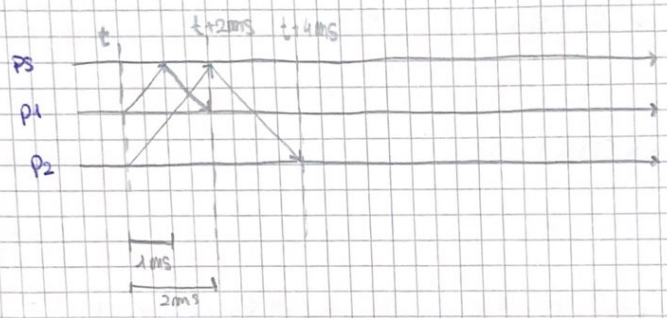
WE SAY THAT CLOCKS C_i ARE ACCURATE WITHIN THE BOUND OF D

INTERNAL SYNCHRONIZATION IS WHEN PROCESSES SYNCHRONIZE THEIR CLOCK BETWEEN THEM. LET $D>0$ BE THE SYNCHRONIZATION BOUND AND C_i AND C_j THE CLOCKS OF PROCESSES P_i AND P_j, THEY ARE 8 INTERNALLY SYNCHRONIZED IF:

$|C_i(t) - C_j(t)| < D$ for each time interval

WE SAY THAT C_i AND C_j AGREE WITHIN THE BOUND D





Accuracy: $\frac{RTT}{2} - t_{min}$

IN THIS CASE WE ASSUME $t_{min}=0$.

Accuracy $P_1 = \frac{RTT_1}{2} = \frac{2ms}{2} = 1ms$

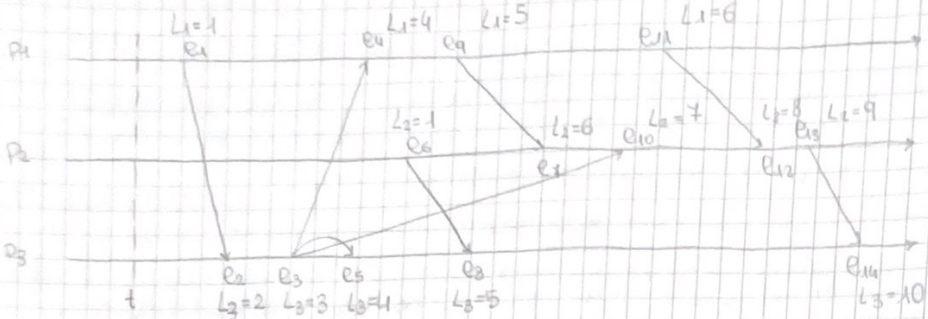
Accuracy $P_2 = \frac{RTT_2}{2} = \frac{4ms}{2} = 2ms$

EXTERNAL SYNCHRONIZATION → INTERNAL SYNCHRONIZATION with bound 2D

$\max(1ms, 2ms) = 2ms = D$

internal synchronization $2D = 4ms$

2.



a. $e_5 \rightarrow e_7$? F — We don't have a path from e_5 to e_7, so we can't say that e_5 happen-before e_7

b. $e_4 \parallel e_5$? T — Because they are generated from the same process (in BROADCAST), so by definition they are concurrent

c. $CK_2(e_6) > CK_1(e_1)$? 1. Compute the logical clock, or
2. Check if they are related by happened-before relation

1. $CK_1(e_1) = 1$ $CK_2(e_6) = 1$ 2. $e_1 \parallel e_6 \Rightarrow$ F

d. $CK_3(e_8) = 3$? $CK_3(e_8) = 5$

e. $CK_3(e_8) = [3,0,3]$? F

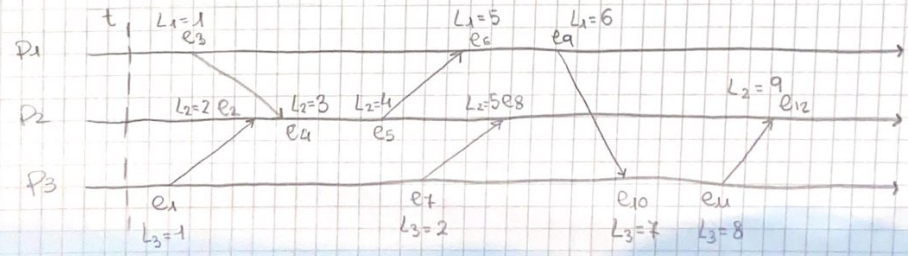
$e_1 : [1,0,0] \rightarrow e_2 : [1,0,1] \rightarrow e_3 : [1,0,2] \rightarrow e_5 : [1,0,3]$
$e_6 : [0,1,0] \rightarrow e_8 : [1,1,4]$

3. LOGICAL CLOCK technique:

THE LOGICAL CLOCK IS A SOFTWARE COUNTER THAT MONOTONICALLY INCREASES ITS VALUE.

1. EACH PROCESS INITIALISE ITS CLOCK $L_i = 0$
2. P_i INCREASE THE CLOCK WHEN GENERATE AN ELENT: $L_i = L_i + 1$ (send or recieve)
3. WHEN A PROCESS RECIVE A MESSAGE IT UPDATES THE CLOCK $L_i = \max(ts, L_i)$ where ts it's the clock of the process that send it the message.

①



②

③

4.

①

IT'S NO
P1 →

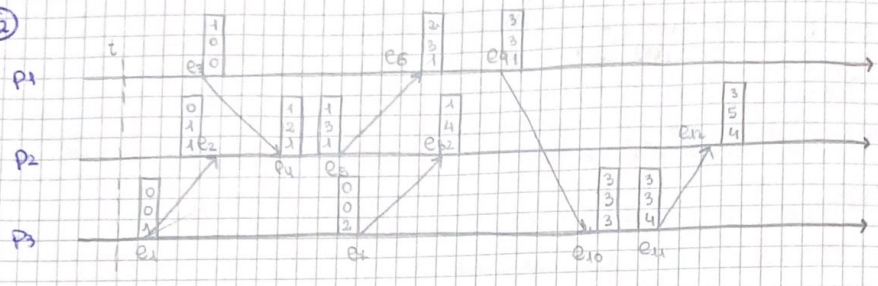
② No, k
recieu
of the

VECTOR CLOCK technique

VECTOR CLOCK IS COMPOSED BY AN ARRAY OF N (number of processes) INTEGER COUNTERS.

1. EACH PROCESS INITIALIZE ITS $V_i = [j] = 0 \quad \forall j = 1..N$
2. INCREASE WHEN IT'S GENERATES AN EVENT $V_i[i] = V[i] + 1$
3. WHEN p_i RECIEVE A MESSAGE UPDATES $V_i[j] = \max(ts[j], V_i[j])$

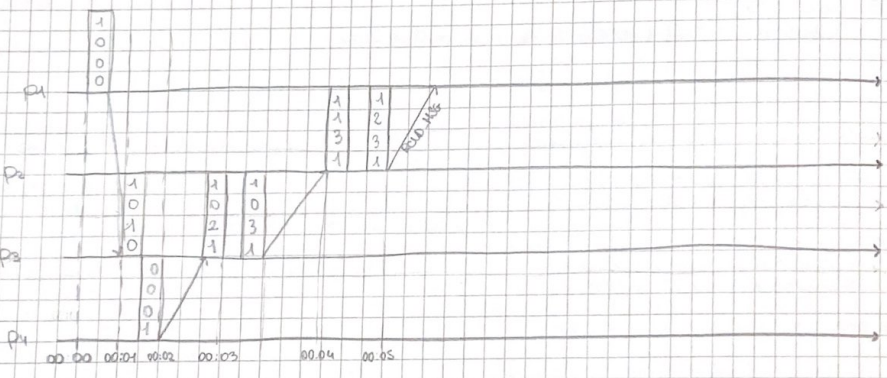
②



③ $e_1 \| e_3$, $e_2 \| e_3$, $e_3 \| e_7$, $e_5 \| e_7$, $e_6 \| e_7$, $e_7 \| e_9$, $e_8 \| e_{10}$, $e_2 \| e_{11}$, $e_8 \| e_9$, $e_6 \| e_8$

4.

①



IT'S NOT UNIQUE, another possible solution could be:

$p_1 \rightarrow p_2$, $p_2 \rightarrow p_3$, $p_4 \rightarrow p_3$, $p_3 \rightarrow p_1$

② No, because process p_4 has just one increment. So if p_4 just recieve a message the other processes never increment component n.4 of the vector.

2.

INIT

ID ← identifier

UPON EVENT <ppl | Send | dest, m > DO ← in the message also send the destination ID

IF (dest > ID) THEN
TRIGGER <pp2p | Send | ID+1, m>

ELSE IF (dest == ID) THEN
TRIGGER <pp2p | Send | ID, m>

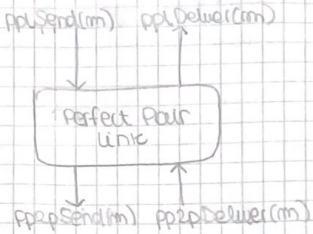
ELSE
TRIGGER <pp2p | Send | ID-1, m>

UPON EVENT <pp2p | Deliver | q , <source,dest,m>> DO

IF (dest > ID) THEN
TRIGGER <pp2p | Send | ID+1, <source,dest,m>>

IF (dest == ID) THEN
TRIGGER <ppl | Deliver | source, m >

IF (dest < ID)
TRIGGER <pp2p | Send | ID-1, <source,dest,m>>



DISTRIBUT

RICART-AGRA

INIT

State = reque
num = num +
last_req =
∀i, ..., N sen
wait until
State = CS
CS
∀r ∈ Q send
Q = ∅
State = NCS
#replies =

No Starvati
COORDIN

PROCESS

INIT

State = idl
coordinat

UPON EVE
Stat
TRI

UPON EV
Sta
TRI

UPON E
Sta
TR

P1

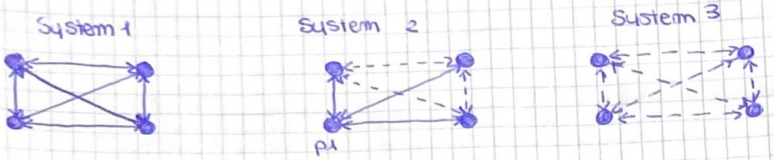
P2

P3

WEEK 3

1.

System 1 | System 2 | System 3



——→ channel A - - - - - - → channel B

1. Yes, because at least one process receive alive is the right information.

INIT

alive = Π
detected = ∅
timer = Δ
phase = 1
starttimer(Δ timer)

UPON EVENT <Timeout and phase == 1 > DO
  forall p∈Π DO
    TRIGGER Send(UST, alive) to p
  phase = 2
  starttimer(timer)

UPON EVENT <Timeout and phase == 2 > DO
  forall p∈Π DO
    IF (p ∉ alive ∧ p ∉ detected) THEN
      detected = detected ∪ {p}
      TRIGGER <Crash | p>
    TRIGGER <Send | [HBRQ]>
  alive = null
  phase = 1
  starttimer(timer * 2)

UPON EVENT Deliver(HBREQ) FROM q :
  TRIGGER Send(HBREP) TO q

UPON EVENT Deliver(HBREP) FROM q
  alive = alive ∪ {q}

UPON EVENT Deliver(UST, alivelist)
  alive = alive ∪ {alivelist}

INSTEAD OF IMPLEMENTING REQUEST-REPLAY, WE USE AN IMPLICIT ACKNOLEDGE. INSTEAD OF REPLYING AND PUT INSIDE THE ALIVE WHEN I RECIEVE IT, I WILL PUT INSIDE ALIVE IF I'M IN PHASE 1 AND I RECIEVE HBRQ.

2. No, because in this case we cannot have a process that has the complete view of the alive processes. So we can do it until the process pi remain correct, if it fail we can not.

3. No, because system 3 is composed only by channel B, that is fair loss link, and we can not implement perfect failure detector over fair loss.

EX.2

INIT

ID ← id
FOR AL
IF i
ELS
EL

leade
suspe

UPON
S
T
T

UPON
lea
IF
E

UPON

UPON

n 3

annel B

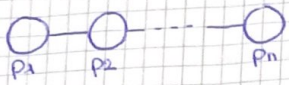
rt information.

IMPLEMENTING
PLAH, WE USE
: ACKNOLEDGE,
: REPLYING
USIDE THE ALLE
ECIEVE IT, I WILL
: ALLE IF I'H
I AND I RECIEVE

ess that
m do it
not.

, that
ailure

EX.2



INIT

ID ← identifier
FOR ALL p∈Π:
  IF ID(p)==1:
    LEFT(p) = null
  ELSE IF ID(p)==n:
    RIGHT(p) = null
  ELSE
    LEFT(p) = ID-1
    RIGHT(p) = ID+1
leader = ⊥
suspected = ∅

UPON EVENT <P | crash | p> DO
  suspected = suspected ∪ {p};
  TRIGGER <Send | left-neighbour(ID(p+1))> to RIGHT
  TRIGGER <Send | Right-neighbour(ID(p-1))> to LEFT

UPON EVENT leader ≠ maxrank(Π \ suspected) DO
  leader := maxrank(Π \ suspected)
  IF ID≠n THEN
    TRIGGER <Send | leader> to RIGHT
  ELSE
    TRIGGER <Ie, leader | leader>

UPON EVENT <Deliver | left-neighbour(ID(p-1))> from p

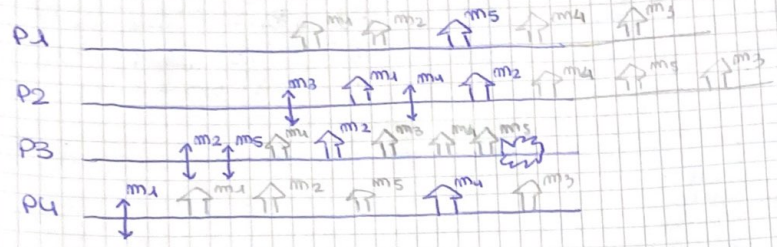
  IF ID(p) > minrank(Π \ suspected)
    LEFT = ID-1
  ELSE
    LEFT = null

UPON EVENT <Deliver | Right-neighbour(ID(p+1))> from p

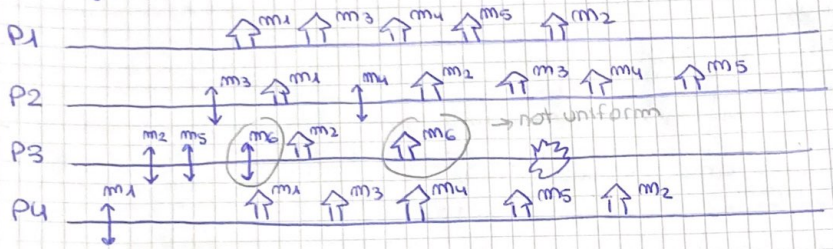
  IF ID < maxrank(Π \ suspected)
    RIGHT = ID+1
  ELSE
    RIGHT = null

EX.3

1. Uniform Reliable Broadcast



2. Regular Reliable but not Uniform

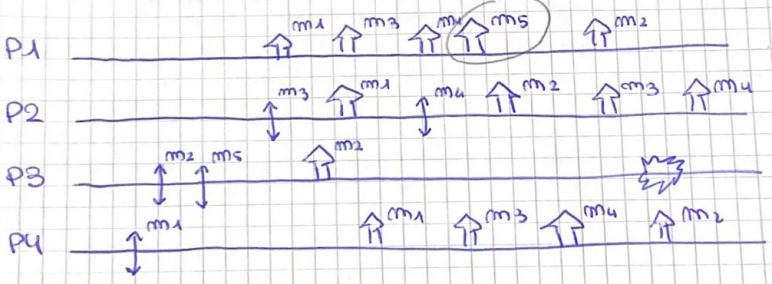


Validity: add m_1, m_3, m_4 to correct

Agreement: all correct must deliver the same set : m_2, m_5 to all correct

No Uniform : new broadcast event

3. Best Effort but not Reliable



m_5 NO AGREEMENT → not Reliable.

EX.4

INIT

ID ← identifier
pending = ∅
replies = ∅
correct = Π
state = NC
last-req =

UPON EVE
state =
last-req
FOR A
<D

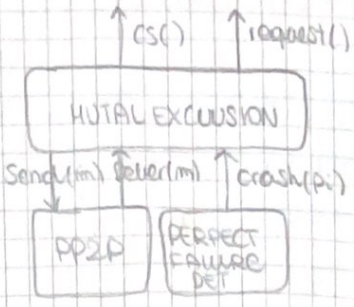
UPON EVE
IF stat
pen
ELSE
TRI

UPON EVE
replie

UPON E
corre

IF
stat
TRU

EX.4



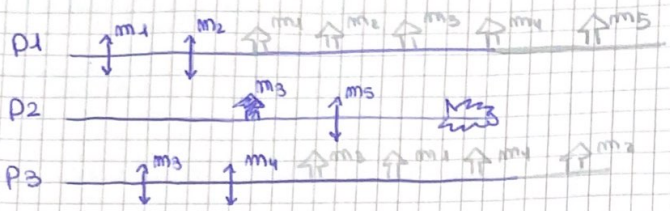
INIT

```
ID ← identifier
pending = ∅
replies = ∅
correct = Π
state = NCS
last_req = 0

UPON EVENT <me, request> DO
  state = requesting
  last_req = ID
    FOR ALL q ∈ Π DO
      <PP2P, Send | request, ID>

UPON EVENT <pp2p, Deliver | request, q>
  IF state == CS or (state == requesting ∧ last_req ≠ ID)
    pending := pending ∪ {q}
  ELSE
    TRIGGER <pp2p, Send | [REPLAY]>

UPON EVENT <PP2P, Deliver | [REPLAY], q>
  replies = replies ∪ {q}

UPON EVENT <P, Crash | p>
  correct = correct \ {p}

IF replies == correct DO
    state = CS
    TRIGGER <me, CS>
    FORALL q ∈ pending
      TRIGGER <PP2P, Send | [REPLAY] TO q>
    state = NCS
    replies = ∅
```

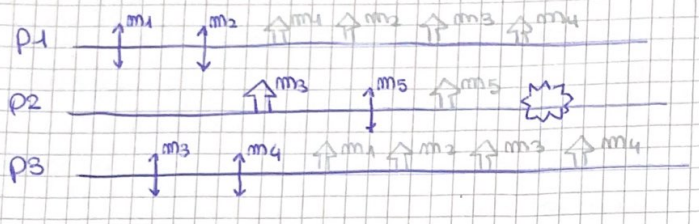
WEEK 4

EX. 1

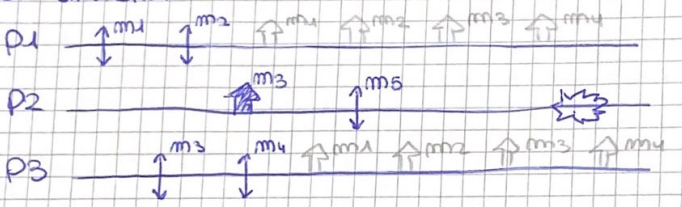
1. Best Effort but not Reliable

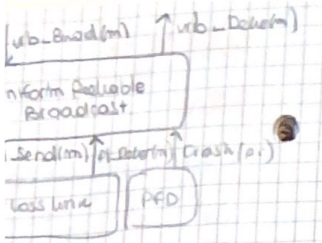


2. Regular not Uniform

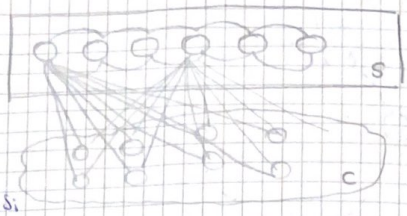


3. Uniform





EX.3 (svolta dalla prof)

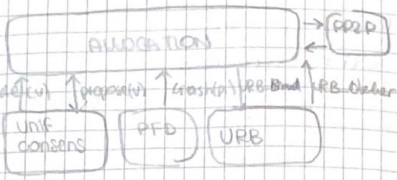


CODE EXECUTED BY A GENERIC CLIENT C_i

OPERATION EXECUTE TASK(T_i)
  FOR EACH $S_i \in \{S_1, S_2, \dots, S_n\}$
    PP2P Send (TASK-REQ, T_i, C_i) TO S_i

UPON PP2PDeliver(TASK_COMPLETED, T_i) from S_i
  TRIGGER completed TASK(T_i).

CODE EXECUTED BY A GENERIC SERVER S_i:



INIT

pending = ∅
busy = false
current_allocation = ∅

UPON EVENT pp2pDeliver(TASK-REQ, T_i, C_i) from C_i
  IF $\langle T_i, C_i \rangle$ IS not IN current_allocation
    pending := pending ∪ {(T_i, C_i)}

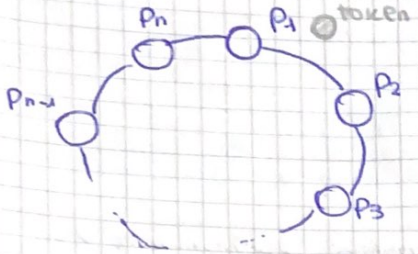
WHEN pending IS not EMPTY
  IF not busy
    candidate = select_from(pending)
    TRIGGER Propose(<ID, candidate>)
  ELSE
    TRIGGER Propose(<ID, null>)

UPON EVENT Crash(S)
  IF THERE EXISTS <id, task> in current_allocation such that S == id
    current_allocation = current_allocation \ {ID, TASK}
    pending = pending ∪ {T_i, C_i}

UPON EVENT Decide(ID, TASK)
  current_allocation = current_allocation ∪ {id, task}
  IF id == myId and task ≠ null
    busy = true
    execute(T_i)
    busy = false
    TRIGGER pp2pSend(TASK_COMPLETED, T_i) TO C_i
    TRIGGER ~~pp2p~~ URBBroadcast(TASK_COMPLETED, T_i, C_i)

UPON EVENT URBDeliver(TASK_COMPLETED, T_i, C_i)
  current_allocation = current_allocation \ {T_i, C_i}
  pending = pending \ {T_i, C_i}

EX. 4



INIT

next = p(i+1) mod n
IF self = p0
  TRIGGER pp2pSend(TOKEN) TO next
proposal = ∅

UPON EVENT pp2pDeliver(TOKEN)
  TRIGGER <c, Propose | v>

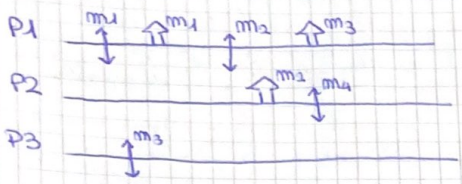
UPON EVENT <c, Propose | v> DO
  IF (v, ID) ∉ proposal DO
    proposal = proposal ∪ {v, ID}
  TRIGGER pp2pSend([PROPOSE, proposal])
  TRIGGER pp2pSend(TOKEN) to next

UPON EVENT pp2pDeliver([PROPOSE, proposal])
  proposal = proposal ∪ {proposal}

UNTIL #proposal == n
  decision = maxTronic(proposal)
  TRIGGER <c, Decide, decision>

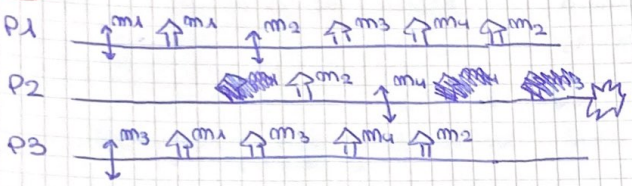
WEEK 5

EX.1



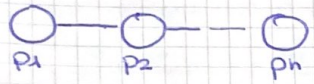
1. $m_1 \rightarrow m_2$ FIFO order
   $m_2 \rightarrow m_4$ local order
   POSSIBLE SEQUENCES:
   m_1, m_2, m_3, m_4
   m_1, m_3, m_2, m_4
   m_1, m_2, m_4, m_3
   m_3, m_1, m_2, m_4 (not for p_1)

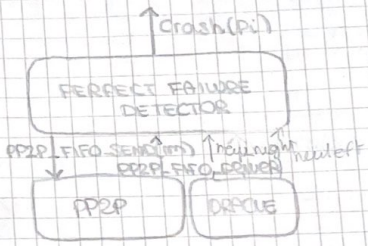
2.



FIFO : $m_1 \rightarrow m_2$
not causal ~~$m_2 \rightarrow m_4$~~

EX. 2





INIT

```
alive = Π
detected = ∅
IF ID == 1
   LEFT = null
ELSE IF ID == n
   RIGHT = null
ELSE
   RIGHT = Pi+1
   LEFT = Pi-1
startimer(Δ)

UPON EVENT <PP2P_FIFO_Deliver | [HBRQ], q>
   TRIGGER <PP2P_FIFO_Send | q, [HBRP]>

UPON EVENT <Timeout> DO
   FORALL p ∈ Π DO
      IF p ∉ alive ∧ p ∉ detected
         detected = detected ∪ {p}
         TRIGGER <P, Crash | p>
         TRIGGER <le, Grade | new_right(p)>
         TRIGGER <le, Grade | new_left(p)>
      TRIGGER <PP2P_FIFO_SEND | [HBRQ], p>
   alive = ∅
   startimer(Δ)

UPON EVENT <PP2P_FIFO_Deliver | [HBRP], q>
   alive = alive ∪ {q}
```

```
UPON EVENT <new_right(p)>
   IF ID == maxID(c)
      RIGHT = null
   ELSE
      RIGHT = RIGHT + 1

UPON EVENT <new_left(p)>
   IF ID == minID(c)
      LEFT = null
   ELSE
      LEFT = LEFT - 1
```

EX.3

INIT

```
correct := Π
pending := =
delivered :=
ack = ∅
flag = false

UPON EVENT
   flag = tr
   TRIGGER

UPON EVEN
   ack = a
   IF m ∈
      pend
      IF

FUNCTION
   RETUR

UPON EX
   delive
   TRIG

UPON E
   com
   TRIG

UPON E
   nex
   FOR
```

EX.4

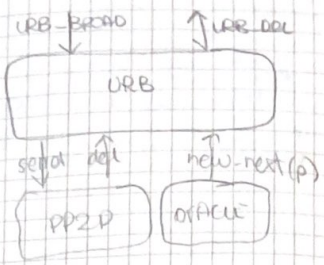
1. Bet
   par

Val

N

I

EX.3



INIT

correct := Π
pending := ∅
delivered := ∅
ack := ∅
flag = false

UPON EVENT <urb, send | m> DO
  flag = true
  TRIGGER pp2pSend(ack, m) to next

UPON EVENT pp2pDeliver(ack, m) FROM p DO
  ack = ack ∪ {p}
  IF m ∉ pending
    pending := pending ∪ {m}
    IF right.flag ≠ true
      TRIGGER pp2pSend(ack, m) to next

FUNCTION candeliver(m) return Boolean IS:
  RETURN correct ⊆ acks

UPON EXISTS m ∈ pending s.t. candeliver(m) ∧ m ∉ delivered:
  delivered = delivered ∪ {m}
  TRIGGER URBDeliver(m)

UPON EVENT Crash(p) DO
  correct = correct \ {p}
  TRIGGER new_next(p)

UPON EVENT new_next(p) DO
  next = p
  FORALL m IN PENDING
    TRIGGER pp2pSend(ack, m) to next.

EX.4.

1. Between t_0 and t_{stab} local variables of f_t processes can assume random values

Validity. It is not guaranteed since the delivery happens after the check that message is not in from[s], but from[s] can assume random values.

No duplication. It is not guaranteed since in from[s] the values are random and could happen to have two same values in the array.

No creation. It is not guaranteed since in from[s] could be any value, also a value not created by anyone.

Agreement. It is not guaranteed since the local variable correct can assume any value, a process could be perceived as correct but it isn't.

2. After t_{stab} (the f_t processes became normal) they cannot change, but may be arbitrarily their local variables but the values they have may be a wrong one (the last assumed before t_{stab}). so the three property are not guaranteed, except for no creation, since after t_{stab} the messages are really generated.

3. In order to guaranteed No Duplication, Validity and Agreement after t_{stab} we need that the local variables are re-initialized

EX.5

1. Perfect-point-to-point links

Validity. YES. This property derives from Reliable Delivery property of perfect point to point link.

No duplication. } YES. From No duplication and no creation properties
No creation. } of pp2p link.

Agreement. NO. Since there is no a perfect failure detector, so you can not know if m is delivered by every correct or not.

FIFO delivery. NO. Same of above.

2. Fair-loss-links

Validity. NO. Since the links are not perfect., so if the message is lost a process can not recieve it.

No duplication NO. Since fair loss link admit finite duplicate.

No creation. YES. From NO creation property of fair loss

Agreement. NO, since there is no a perfect failure detector

FIFO delivery. NO. same of above.

WEEK 6

EX.1

1. $m_1 \to m_2$ local order
   $m_3 \to m_4$ local order
   $m_2 \to m_4$ FIFO order
   total order
   $m_1 \to m_3$ NON UNIF
   $m_2 \to m_4$
   $m_1 \to m_2$

   m_1, m_2, m_3, m_4
   m_1, m_3, m_2, m_4
   m_3, m_1, m_2, m_4
   non uniform

violate this

2. 

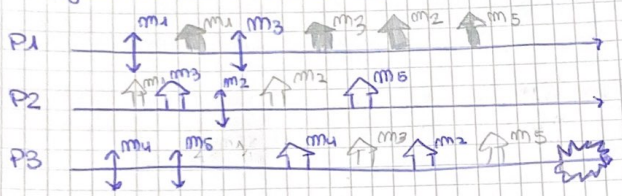
UNIFORM = failure can deliver in different order

TO(UA, UNIFTO)

we can not violate $m_1 \to m_2$ because is a total order constraint

EX.2

1. Regular Reliable not Uniform



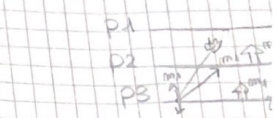
2. FIFO, not Causal
   $m_1 \to m_3$ FIFO
   $m_3 \to m_2$ local order
   $m_4 \to m_5$ FIFO
   TOTAL
   $m_3 \to m_5$
   $m_4 \to m_2$

   m_2, m_1, m_3, m_4, m_5
   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_4, m_3, m_5
   m_2, m_1, m_4, m_5, m_3

3. m_1, m_3, m_4, m_2, m_5
   m_1, m_4, m_3, m_2, m_5
   m_1, m_4, m_3, m_5, m_2
   m_1, m_3, m_4, m_5, m_2
   m_4, m_1, m_3, m_2, m_5
   m_4, m_1, m_3, m_5, m_2

EX.3

1. X broadcast() given by the instead agree can be help



2. [illegible] m4 sn so

EX.4

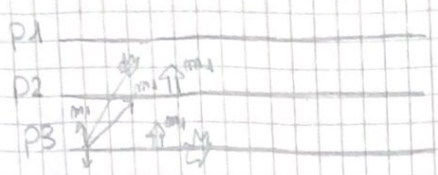
1. P1 —
   P2 —
   P3 —
   P4 —

2. Causa
   $m_2 \to$
   m_2
   Toto
   m_2
   m_1

3. TO(
   TCAU
   m
   TO
   m
   m

EX.3

1. XbroadcastC) guarantee Validity, No Creation, No Duplication given by the properties of perfectLink, so it ensure Best Effort Broadcast, instead agreement is not guaranteed since when a process crash can be happen this situation:

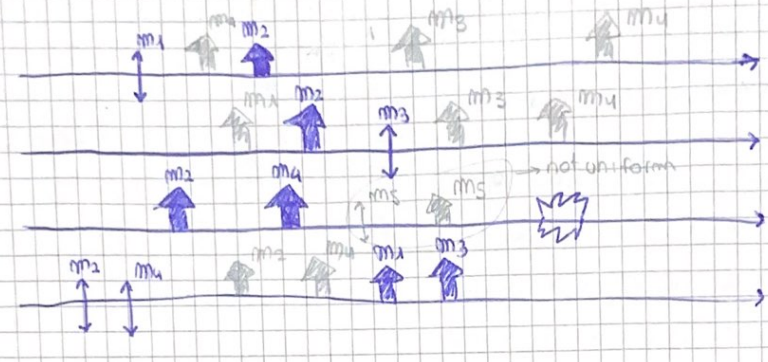


If process p_3 broadcast a message m, then p_2 deliver it. In the meanwhile p_3 crash and p_1 doesn't receive m. We have that p_2 had delivered m, and p_1 no, so agreement it's not respected.

2. You never check mysn so it's not FIFO and so not Causal and no Total.

EX.4

1.



2. Causal order:

$m_2 \to m_3$ local order
$m_2 \to m_4$ FIFO order

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

Total

$m_2 \to m_4$
$m_1 \to m_3$

3. TO(UA, WUTO) ok not TO(UA, SUTO) udating causal order

TCAUSAL
$m_4 \to m_2$ since the faulty deliver m_2, m_4

TOTAL

$m_1 \to m_3$

$m_2 \to m_3$ (from p_2)

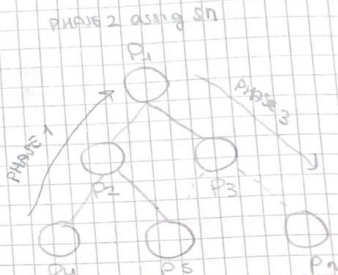
$m_4 \to m_2 \to m_3$
$m_1 \quad m_1 \quad m_1$

$m_1 \to m_4 \to m_2 \to m_3$
$m_4 \to m_1 \to m_2 \to m_3$
m_4, m_2, m_1, m_3

EX.5

a. F because p_2 deliver m_4 that others processes don't deliver
b. F NUA is satisfied
c. F
d. T
e. F because SUTO → SUTO
f. T if we add m_4 to p_1 and p_3 NUA is satisfied (also UA) but not SUTO
g. T
h. T
i. F It is not satisfy
l. F

EX.6 - TREE TOPOLOGY total order broadcast



we can use the root as coordinator
sent up to the root the messages (the root assign the sequence number and push the messages to the leaves locally check who is the next and put in waiting who is not the next

INIT

```
pending = {}
father = getFather()
RChild = getRChild()
LChild = getLChild()
SN = 0
```

```
UPON EVENT <tob, Broadcast | m>
  IF (father == null)   //I'm the root
    SN = SN + 1
    TRIGGER <tob, Deliver | m> from self
    IF (RChild != null)
      TRIGGER pp2pSend ([DEL, SN, m, self]) to RChild
    IF (LChild != null)
      TRIGGER pp2pSend ([DEL, SN, m, self]) to LChild
  ELSE
    TRIGGER pp2pSend ([YES, m, self]) to father   //I need to send message to the root
```

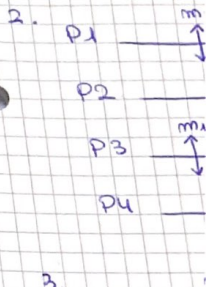
```
UPON EVENT pp2pDeliver ([YES, m, p]) from q
  IF (father == null)
    SN = SN + 1
    TRIGGER <tob, Deliver(m)> from p
    IF (RChild != null)
      TRIGGER pp2pSend ([DEL, SN, m, SELF)] to RChild
    IF (LChild != null)
      TRIGGER pp2pSend ([DEL, SN, m, SELF)] to LChild
  ELSE
    TRIGGER pp2pSend ([YES, m, self] to father
```

```
UPON EVENT <tob, Deliver | [DEL, SN', m, p]>
  pending := pending ∪ {m, p, SN'}
  // propagate the order of delivery to my children
  IF (RChild != null)
    TRIGGER pp2pSend ([DEL, SN', m, p)] to RChild
  IF (LChild != null)
    TRIGGER pp2pSend ([DEL, SN' m, p] to LChild

UPON EXISTS (m, p, SN') SUCH THAT SN' = SN + 1
  SN = SN + 1
  TRIGGER Deliver(m) FROM p.
```

WEEK 4

EX.1

1. causal order
   $m_1 \rightarrow m_2$ bec
   $m_3 \rightarrow m_2$ F(
   $m_2 \rightarrow m_4$ b
   total order
   $m_2 \rightarrow m_4$
   $m_3 \rightarrow m_4$

2. 

3. p1
   p2
   p3
   p4

EX.2

WEEK 4

EX.1

1. causal order:

$m_1 \rightarrow m_2$ local order

$m_3 \rightarrow m_2$ FIFO

$m_2 \rightarrow m_4$ local order

total order

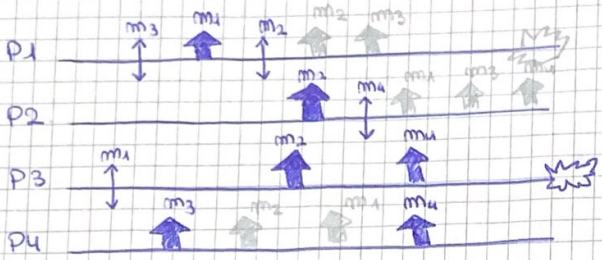
$m_2 \rightarrow m_4$

$m_3 \rightarrow m_4$

m_1, m_2, m_3, m_4

m_1, m_3, m_2, m_4

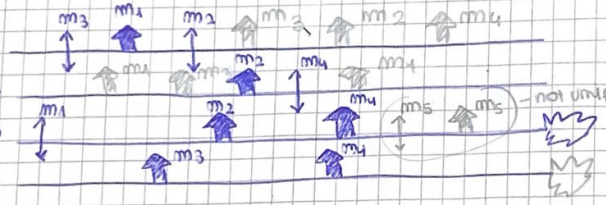
2.



TO (UA, UNUTQ)

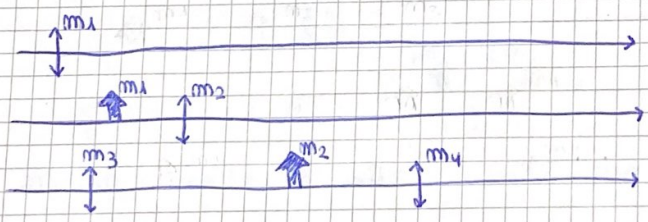
$m_2 \rightarrow m_4$

3.



not total $m_4 \rightarrow m_3$

EX.2 OK



m_1, m_2, m_3, m_4

m_1, m_3, m_2, m_4

m_3, m_1, m_2, m_4

Causal:

$m_1 \rightarrow m_2$ local

$m_3 \rightarrow m_4$ FIFO

$m_2 \rightarrow m_4$ local

EX.5

1. Regular Register

$r_2() \to 0,1,2$
$r_4() \to 1,2$
$r_3() \to 1,2,3$
$r_4() \to 2,3$
$r_5() \to 2,3$

2. Atomic Register

$r_2() \to 0,1,2$
$r_4() \to 1,2$
$r_3() =$ if $r_2() \to 2$ then $r_3() \to 2,3$ else $r_3() \to 1,2,3$
if $r_4() \to 2$ then $r_3() \to 2,3$ else $r_3() \to 1,2,3$
$r_4() \to 2,3$
$r_5() \to$ if $r_3() \to 3$ then $r_5() \to 3$ else $r_5() \to 2,3$

EX.6

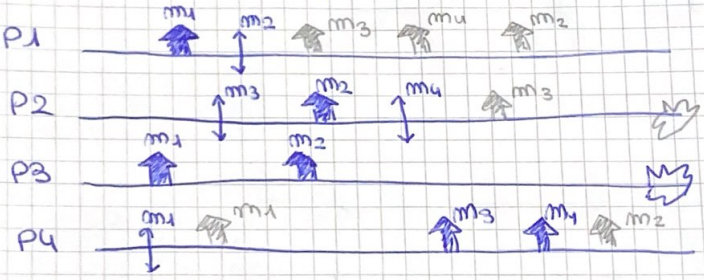
1. Causal

$m_1 \to m_2$ local
$m_2 \to m_4$ local
$m_3 \to m_4$ FIFO

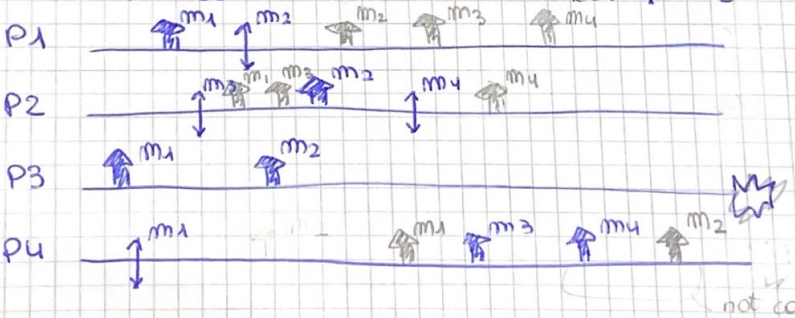
Total:
$m_1 \to m_2$
$m_3 \to m_4$

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

2. Total not Causal ($m_4 \to m_2$)



3. FIFO not Causal not Total ($m_2 \to m_1$ e $m_4 \to m_2$)



not causal

EX.3

1. Termina
The
poss
Dut
term

Validi

Orde

2.

2.

EX.4

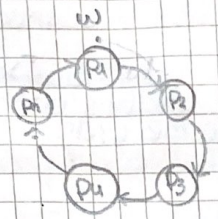
INIT
next = (p_rank + 1) mod n
correct = Π
reading = FALSE
ack = ∅
readval = ⊥

UPON EVENT ⟨P, Crash | p⟩ DO
  correct = correct \ {p}
  next = next + 1

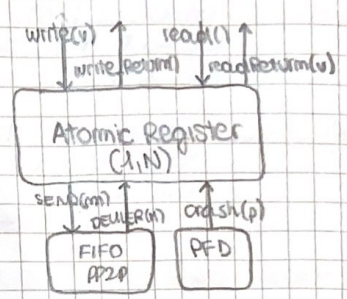
UPON EVENT ⟨ onar, Write | v ⟩
  TRIGGER ⟨ pp2pFIFO, Send | [WRITE, v, ID] ⟩ to next

UPON EVENT ⟨ pp2pFIFO, Deliver | [WRITE, v', ID'] ⟩ DO
  IF ID' ≠ self
    readval = v'
    ack = ack ∪ {self}
    TRIGGER ⟨ pp2pFIFO, Send | [ACK, ack] ⟩ to next
    TRIGGER ⟨ pp2pFIFO, Send | [WRITE, v', ID'] ⟩ to next

UPON EVENT ⟨ pp2pFIFO, Deliver | [ACK, ack] ⟩ DO
  IF correct \ {self} ⊆ ack DO
    TRIGGER ⟨ onar, WriteReturn ⟩
  ELSE
    ack = ack ∪ {self}
    TRIGGER ⟨ pp2pFIFO, Send | [ACK, ack] ⟩ to next



ID of writer



just one process can write, when it recive all acks it returns the write

WEEK 8

EX.1

1) Regular Register

$r_1() \to 0,9,8$

$r_2() \to 0,9,8$

$r_4() \to 9,8,7$

$r_5() \to 7$

$r_3() \to 8,7$

2) Atomic Register

$r_1() \to 0,9,8$

$r_2() \to 0,9,8$

$r_4() \to 9,8,7$

$r_3()$: [illegible] $\to r_3() \to 8,7$

$r_5() \to 7$

3) $w(9)\, r_1(9) r_2(9)\, w(8) r_4(8)\, w(7)\, r_3(7) r_5(7)$

EX.3

1. NO or $S = add(1), get_2() \to \{1\}, remove(1), get_4() \to \{\}, get_3() \to \{1\}$, $add(3), get_4() \to \{3\}$

not legal

2. The original sequence provide is what we are looking for.

EX.5

1. TO (NUA, WUTO)

   NUA = because the faulty process deliver messages that correct proceses don't deliver (m_4, m_5)

   WUTO = because the messages that both correct and faulty delivers are in the common order.

2. Causal Order:

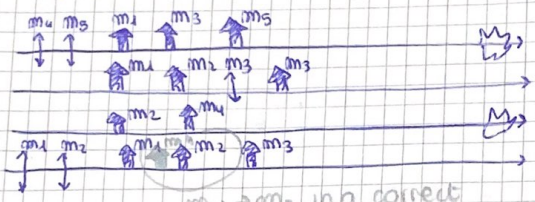
   $m_1 \rightarrow m_2$ FIFO
   $m_2 \rightarrow m_3$ local
   $m_4 \rightarrow m_5$ FIFO

   FIFO ORDER BROADCAST IS NOT SATISFIED BECAUSE YOU DON'T HAVE AGREEMENT (m_4 will be delivered before m_5 but p1)

3. 

   TO(UA, WUTO)

   key: not adding m_4 over p3, because if you add it, you have TOTO (some prefix)

4. 

   TO(NUA, WNUTO)

   $m_4 \rightarrow m_2$ in a correct

EX.4

Validity: Satisfied. This property is overpmeed by the beb Broadcast property, that ensure validity.

No duplicatum: Not satisfied. The check of (m ∉ delivered) is missing, so could happem that a process have already deliver it.

No creatum: Satisfied.

Uniform agreement: Not satisfy. Since the failure detector is not perfect, could happem that a process is suspected, in the meanwhile the other process deliver a message, and when it restore as correct it never deliver such message.