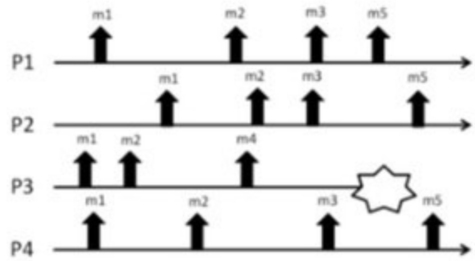
Distributed Systems
10/06/2020

Family Name______________________ Name__________________ Student ID____________

Please, tick the appropriate option:	
☐ Master of Science in Engineering in Computer Science	☐ Erasmus
☐ Master of Science in Artificial Intelligence and Robotics	☐ Other

Ex 1: Concerning software replication techniques, describe the primary-backup and the active replication approach with particular emphasis on how each technique handles failures.

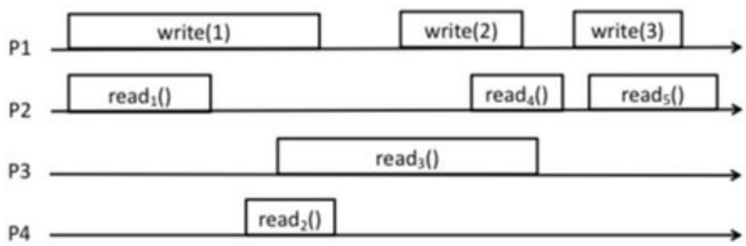
Ex 2: Consider the run depicted in the figure:



1. Which type of total ordering is satisfied by the run? Specify both the agreement and the ordering properties.
2. Modify the run in order to satisfy TO(UA, WUTO) but not TO (UA SUTO)
3. Modify the run in order to satisfy TO(NUA, WNUTO) but not TO(NUA, WUTO)

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

Ex 3: 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.
3. Let us assume that values retuned by read operations are as follow: read$_1$() $\rightarrow$ 1, read$_2$() $\rightarrow$ 0, read$_3$() $\rightarrow$ 1, read$_4$() $\rightarrow$ 2, read$_5$() $\rightarrow$ 3. Is the run depicted in the Figure linearizable?

Ex 4: 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
14. trigger ⟨ beb, Broadcast | [WRITE, ts + 1, v] ⟩;
```

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

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

21. upon correct ⊆ writeset do
22. writeset := ∅;
23. if reading = TRUE then
24.         reading := FALSE;
25.         trigger ⟨ onar, ReadReturn | readval ⟩;
26. else
27.         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 happen to every atomic register property (i.e., termination, validity and ordering) if messages can be lost.
2. Discuss what happen to every atomic register property (i.e., termination, validity and ordering) if we change line 12 (i.e., **trigger** ⟨ *beb*, Broadcast | [WRITE, ts, val] ⟩; in the read handler) with **trigger** ⟨ *beb*, Broadcast | [WRITE, ts+1, val] ⟩;

Ex 5: Consider a distributed system composed of *n* processes $\prod=\{p_1, p_2 \ldots p_n\}$ with unique identifiers that exchange messages through fair loss 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 Regular Reliable Broadcast.

Ex 1: Concerning software replication techniques, describe the primary-backup and the active replication approach with particular emphasis on how each technique handles failures.

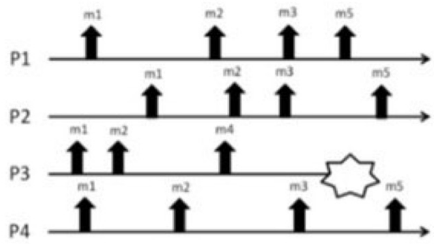
REPLICATION IS ESSENTIAL TO GUARANTEE FAULT TOLERANCE AND MAINTAIN THE AVAILABILITY OF A SERVICE EVEN IN THE PRESENCE OF FAILURES.

IN THE PRIMARY-BACKUP ACTION, THE PRIMARY RECEIVES THE OPERATION FROM THE CLIENT, UPDATES THE BACKUPS AND WAITS FOR THE ACKS. AFTER RECEIVING THE ACKS FROM THE BACKUPS, THE PRIMARY SENDS THE RESULT TO THE CLIENT.
IF THE PRIMARY FAILS, A NEW PRIMARY IS ELECTED AMONG THE BACKUPS.
THERE ARE 3 SCENARIOS:

- CRASH AFTER RESPONSE: THE NEW PRIMARY RECOGNIZE THAT THE RESPONSE HAS ALREADY BEEN PROCESSED AND RESENDS IT.
- CRASH BEFORE UPDATES: THE NEW PRIMARY UPDATES THE BACKUPS AFTER A NEW REQUEST.
- CRASH DURING UPDATES: THE NEW PRIMARY CHECKS THE CONSISTENCY STATE.

IN THE ACTIVE REPLICATION APPROACH, ALL REPLICAS PROCESS REQUESTS IN PARALLEL, THEY START FROM THE SAME STATE, PERFORM THE SAME OPERATIONS IN THE SAME ORDER VIA TO BRDCST, AND PRODUCE THE SAME OUTPUT INDEPENDENTLY, THUS CLIENTS CAN RECEIVE ANSWERS EVEN IF SOME NODES FAIL.
NO RECOVERY ACTIONS ARE NEEDED.

Ex 2: Consider the run depicted in the figure:



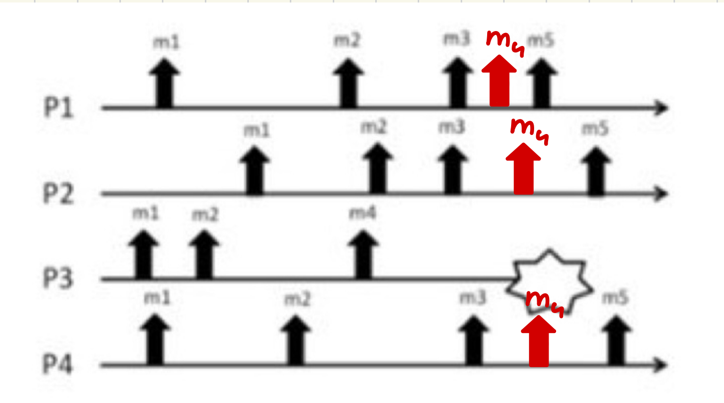
1. Which type of total ordering is satisfied by the run? Specify both the agreement and the ordering properties.
2. Modify the run in order to satisfy TO(UA, WUTO) but not TO (UA SUTO)
3. Modify the run in order to satisfy TO(NUA, WNUTO) but not TO(NUA, WUTO)

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

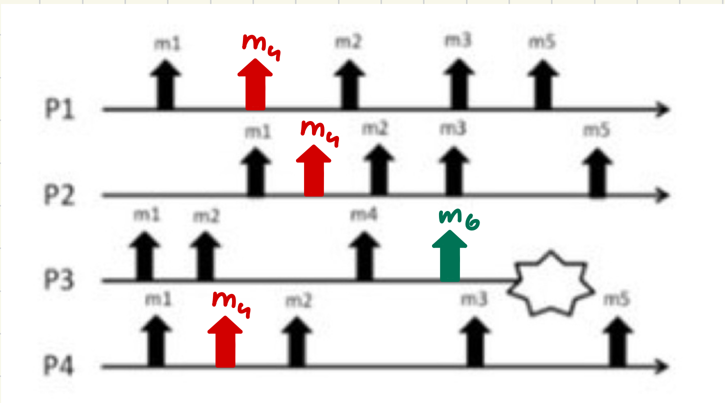
1) TO (NUA, SUTO)

NUA: CORRECT PROCESSES DON'T DELIVER m_4, DELIVERED BY p_3 (FAULTY)
SUTO: IF SOME PROCESS TODELIVERS SOME MESSAGE m BEFORE MESSAGE m', THEN A PROCESS TODELIVERS m' ONLY AFTER IT HAS TODELIVERED m.

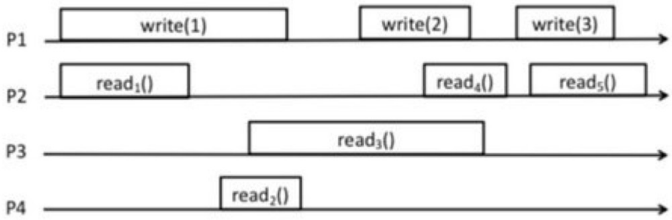
2) TO (UA, WUTO) BUT NOT TO (UA, SUTO)



3) TO (NUA, WNUTO) BUT NOT TO (NUA, WUTO)



Ex 3: 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.
3. Let us assume that values retuned by read operations are as follow: read$_1$() → 1, read$_2$() → 0, read$_3$() → 1, read$_4$() → 2, read$_5$() → 3. Is the run depicted in the Figure linearizable?

1) $R_1()$: 0,1
$R_2()$. 0,1
$R_3()$: 0,1,2
$R_4()$: 1,2
$R_5()$: 2,3

2) $R_1()$: 0,1
$R_2()$. 0,1 IF $R_1()$: 0
1 IF $R_1()$: 0,1
$R_3()$: 0,1,2 IF $R_1()$: 0
1,2 IF $R_1()$: 0,1
$R_4()$: 1,2
$R_5()$: 2,3

3) NO, R_2 CANNOT READ 0 AFTER $R_1()$: 1

Ex 4: 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
14. trigger ⟨ beb, Broadcast | [WRITE, ts + 1, v] ⟩;
```

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

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

21. upon correct ⊆ writeset do
22. writeset := ∅;
23. if reading = TRUE then
24.         reading := FALSE;
25.         trigger ⟨ onar, ReadReturn | readval ⟩;
26. else
27.         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 happen to every atomic register property (i.e., termination, validity and ordering) if messages can be lost.

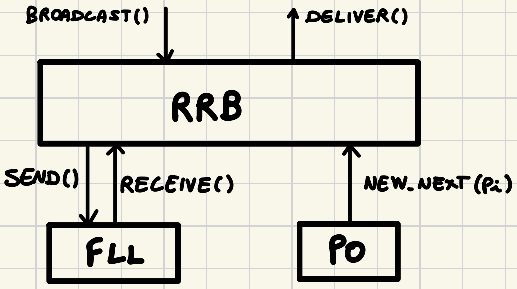
2. Discuss what happen to every atomic register property (i.e., termination, validity and ordering) if we change line 12 (i.e., **trigger** ⟨ *beb*, Broadcast | [WRITE, ts, val] ⟩; in the read handler) with **trigger** ⟨ *beb*, Broadcast | [WRITE, ts+1, val] ⟩;

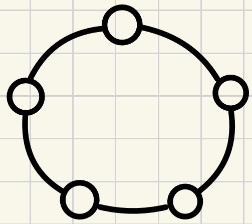
1) TERMINATION: NOT SATISFIED. WRITING MAY NEVER END IF THE ACKS ARE NOT RECEIVED. THE READINGS MAY ALSO BLOCK IF THE WRITING MESSAGE IS LOST AND NO UPDATED VALUE IS RECEIVED.
VALIDITY: NOT SATISFIED. A READING COULD RETURN A OLDER VALUE IF THE MOST RECENT WRITING MESSAGE IS LOST.
ORDERING:

2) TERMINATION: SATISFIED. THE WRITING AND READING OPERATIONS CONTINUE TO PROCEED WITHOUT INDEFINABLE EXPECTATIONS.
VALIDITY:

ORDERING:

Ex 5: Consider a distributed system composed of n processes $\prod=\{p_1, p_2\ldots p_n\}$ with unique identifiers that exchange messages through fair loss point-to-point links. Processes are connected through a directed ring (i.e., each process p_i can exchange messages only with processes and $p_{i+1(\mathrm{mod}\ n)}$). 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 Regular Reliable Broadcast.





```
UPON EVENT <RRB, INIT> DO
  NEXT = Pi+1 (MOD n)
  PENDING = ∅
  TIMER = Δ
  DELIVERED = ∅
  STARTTIMER (TIMER)

UPON EVENT <RRB, BROADCAST | m> DO
  PENDING = PENDING ∪ {m}
  TRIGGER <FLL, SEND | m, SELF> TO NEXT

UPON EVENT <FLL, DELIVER | m, Pi> DO
  IF Pi == SELF AND m IN PENDING
    TRIGGER <RRB, DELIVER | m>
    PENDING = PENDING / {m}
    DELIVERED = DELIVERED ∪ {m}
  ELSE
    IF m NOT IN DELIVERED
      DELIVERED = DELIVERED ∪ {m}
      PENDING = PENDING ∪ {m}
      TRIGGER <RRB, DELIVER | m>
    TRIGGER <FLL, SEND | m, Pi> TO NEXT

UPON EVENT <TIMEOUT>
  IF PENDING != 0
    FORALL m IN PENDING
      TRIGGER <FLL, SEND | m, SELF> TO NEXT
  STARTTIMER (TIMER)

UPON EVENT <PO | NEW_NEXT (Pi)> DO
  NEXT = Pi
```