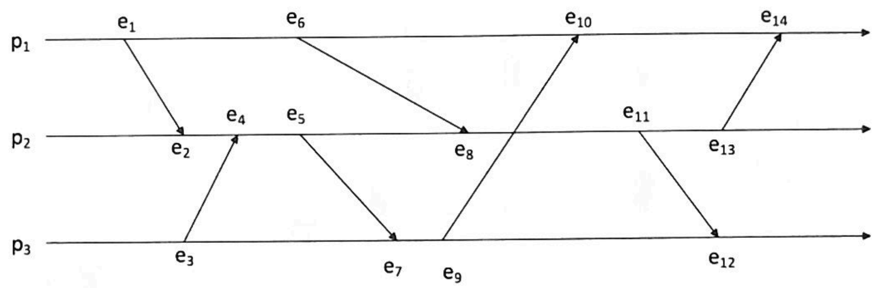
Distributed Systems
05/07/2024
(6 CFU)

Family Name________________________ Name____________________ Student ID____________

Ex 1: Introduce the leader election problem. Discuss how it is possible to build an oracle implementing the leader election abstraction in a synchronous system and explain why it is not possible to build such an oracle in an eventually synchronous system.

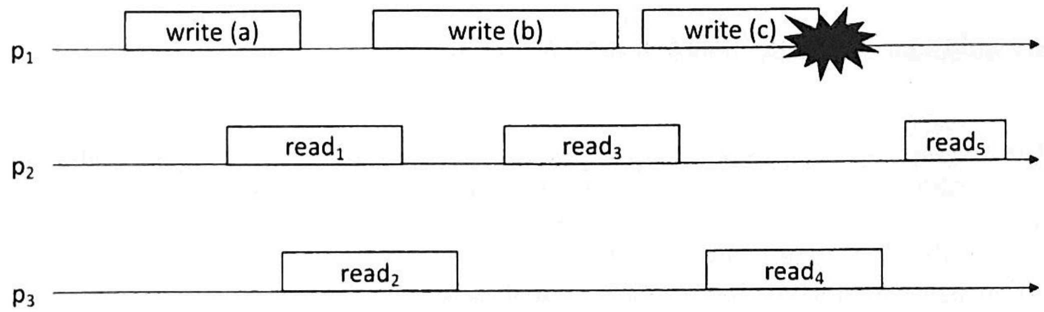
Ex 2: Let us consider the following execution history



Let us denote with $ck(e_i)$ the logical clock associated to event e_i. Considering the execution history shown in the figure above, assess the truthfulness of every statement and provide a motivation for your answer:

1	If we use scalar clocks for timestamping events, then ck (e_6) > ck (e_5)	T	F
2	e_2 happened before e_3 (according with Lamport's definition of happened-before)	T	F
3	If we use scalar clocks for timestamping events, then ck (e_{14}) = ck (e_{10}) +1	T	F
4	e_6 and e_7 are concurrent events	T	F
5	If we use scalar clocks for timestamping events, then ck (e_9) < ck (e_{11})	T	F
6	If we use vector clocks for timestamping events, then ck (e_{13}) = [4, 6, 4]	T	F
7	If we use vector clocks for timestamping events, then ck(e_1) = ck (e_3)	T	F
8	If we use vector clocks for timestamping events, then ck(e_5) and ck (e_{10}) are not comparable	T	F
9	If we use vector clocks for timestamping events, then ck(e_8) < ck (e_{10})	T	F
10	If we use vector clocks for timestamping events, then ck (e_4) = [1, 1, 1]	T	F

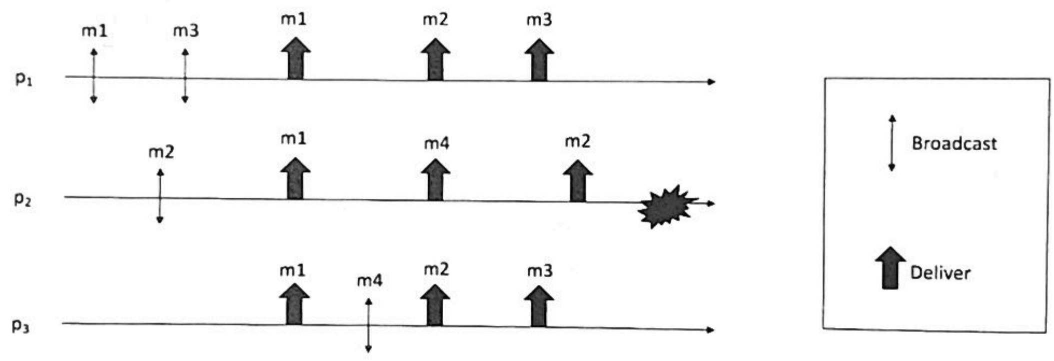
Ex 3: Let us consider the following execution history



Assuming that the initial value stored in the register is 0, assess the truthfulness of every statement and provide a motivation for your answer:

1	If the proposed run refers to a regular register, then a is a valid value for $read_1$	T	F
2	If the proposed run refers to a regular register, then 0 is not a valid value for $read_3$	T	F
3	If the proposed run refers to a regular register, then a is not a valid value for $read_2$	T	F
4	If the proposed run refers to a regular register, then b is a valid value all read operations	T	F
5	If the proposed run refers to a regular register, then $read_3$ may return only a and b	T	F
6	If the proposed run refers to an atomic register, then $read_1$ and $read_2$ may return different values	T	F
7	If the proposed run refers to an atomic register, then $read_3$ returns b if and only if $read_2$ returns b	T	F
8	If the proposed run refers to an atomic register, then $read_4$ and $read_5$ must always return the same value	T	F
9	If the proposed run refers to an atomic register and $read_4$ returns c, then $read_5$ may returned b or c	T	F
10	If the proposed run refers to an atomic register then $read_5$ can never return the value c	T	F

Ex 4: Consider the partial execution in the following figure



Given the run depicted in the figure state the truthfulness of the following sentences:			
a	The run satisfies the Reliable Broadcast specification	T	F
b	The run satisfies the Best Effort Broadcast specification	T	F
c	The strongest ordering property satisfied is SUTO	T	F
d	The WUTO ordering property is satisfied	T	F
e	The run satisfies the FIFO ordering property	T	F
f	The run satisfies the Causal ordering property	T	F
g	If correct processes add the delivery of m4 as last message, then the resulting run satisfies the Reliable Broadcast specification	T	F
h	If correct processes add the delivery of m4 as last message, then the resulting run satisfies TO (UA, WNUTO)	T	F
i	If correct processes add the delivery of m4 between the delivery of m2 and m3, then the resulting run does not satisfy causal order	T	F
l	If p2 adds the delivery of m4 as last message, then the resulting run satisfies the Reliable Broadcast specification	T	F

<u>For each point, provide a justification for your answer</u>

Ex 5: Consider a distributed systems composed by n processes $p_1, p_2, \ldots p_n$, with a unique identifier that are structured as a binary tree. Messages are exchanged between processes over the edges of the tree which acts 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).

a) Write the pseudo code of an algorithm implementing a primitive of total order broadcast assuming no process fails.

b) Discuss which properties of the total order are violated in the presence of even a single crash of a process.
c) Discuss how to modify your algorithm assuming that processes may fail by crash but are equipped with a perfect failure detector.

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: __________________________

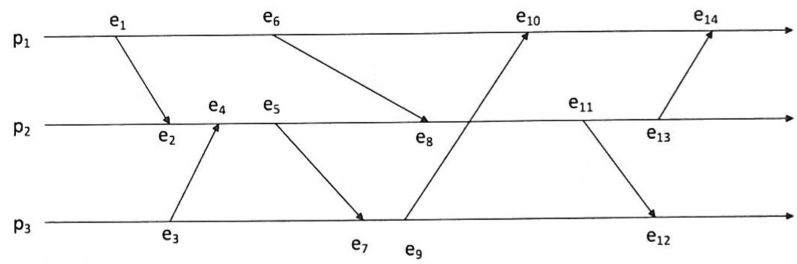
Ex 1: Introduce the leader election problem. Discuss how it is possible to build an oracle implementing the leader election abstraction in a synchronous system and explain why it is not possible to build such an oracle in an eventually synchronous system.

ELECTING A LEADER CONSISTS OF IDENTIFYING A PROCESS THAT ACTS AS A COORDINATOR BETWEEN DISTRIBUTED PROCESSES. A LEADER CAN MANAGE EXCLUSIVE ACCESS TO A SHARED RESOURCE.

IN A SYNCHRONOUS SYSTEM, IT'S POSSIBLE TO BUILD AN ORACLE FOR THE ELECTION BY EXPLOITING THE FACT THAT THE MESSAGES ARE DELIVERED WITHIN A WELL KNOW TIME AND THAT THE FAULTS CAN BE RELIABLY DETECTED USING A PFD. A POSSIBLE APPROACH IS TO ASSIGN EACH PROCESS A UNIQUE ID AND ENSURE THAT THE PROCESS WITH THE HIGHEST ID BECOMES THE LEADER, ENSURING THAT ALL THE PROCESSES ELECT THE SAME LEADER IN A FINISHED NUMBER OF STEPS.

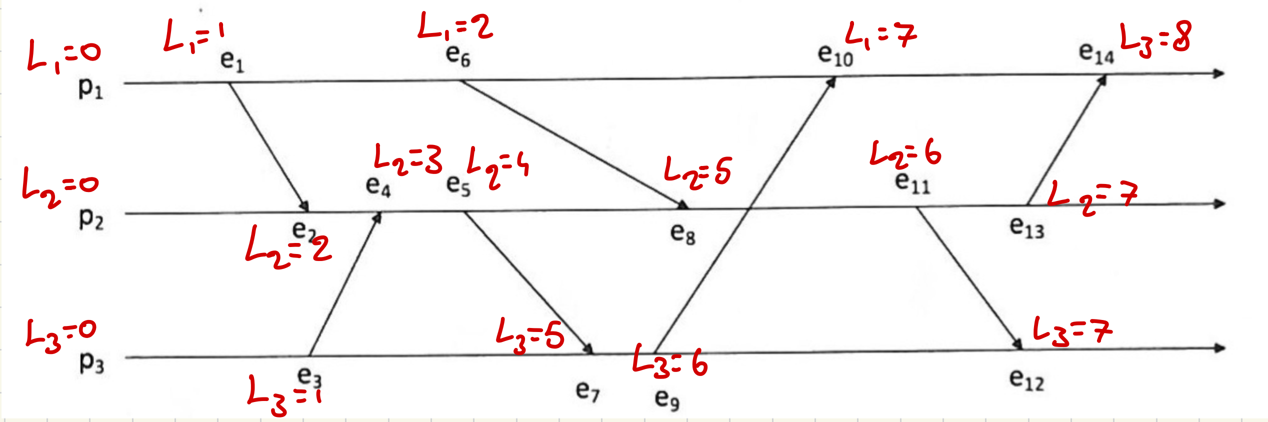
IN AN EVENTUALLY SYNCHRONOUS SYSTEM, THE ABSENCE OF A WELL KNOW LIMIT ON COMMUNICATION TIMES AND THE DETECTION OF FAULTS PREVENTS THE CONSTRUCTION OF A PERFECT ORACLE. A PROCESS COULD BE ERRONEOUSLY CONSIDERED FAILED AND REPLACED AS A LEADER, LEADING TO INFINITE CHANGES IN THE ROLE OF LEADER WITHOUT EVER ACHIEVING A DEFINITE STABILIZATION.

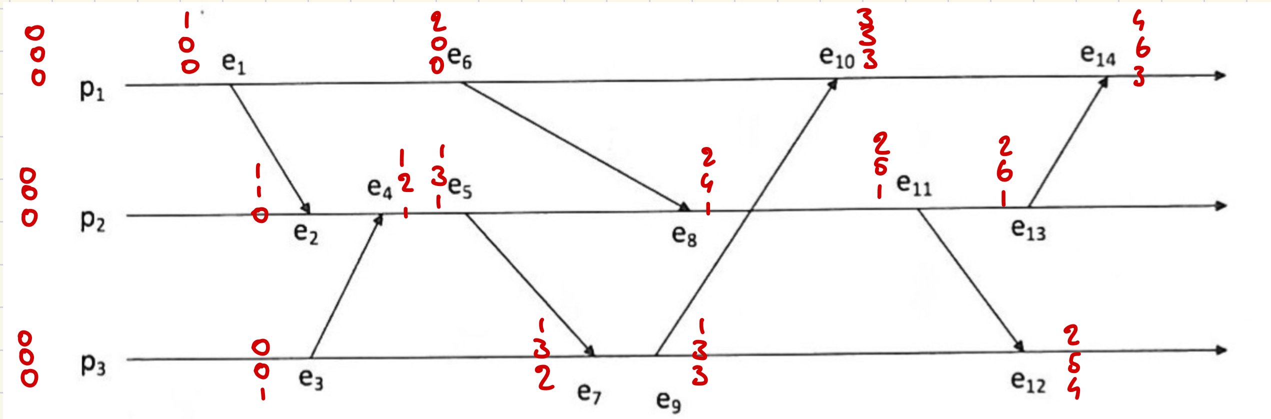
Ex 2: Let us consider the following execution history



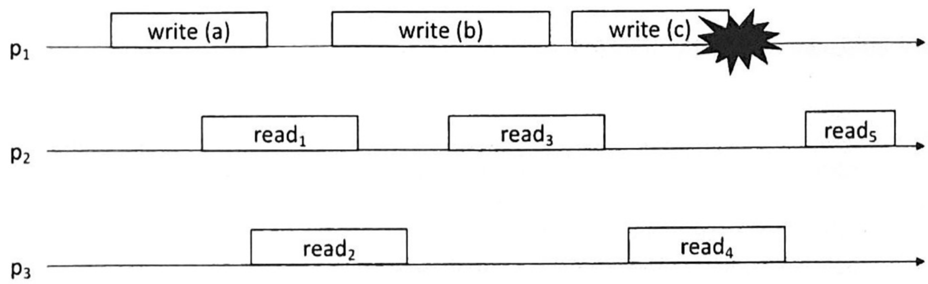
Let us denote with ck(e_i) the logical clock associated to event e_i. Considering the execution history shown in the figure above, assess the truthfulness of every statement and provide a motivation for your answer:

1	If we use scalar clocks for timestamping events, then ck (e_6) > ck (e_5)	T	F
2	e_2 happened before e_3 (according with Lamport's definition of happened-before)	T	F
3	If we use scalar clocks for timestamping events, then ck (e_{14}) = ck (e_{10}) +1	T	F
4	e_6 and e_7 are concurrent events	T	F
5	If we use scalar clocks for timestamping events, then ck (e_9) < ck (e_{11})	T	F
6	If we use vector clocks for timestamping events, then ck (e_{13}) = [4, 6, 4]	T	F
7	If we use vector clocks for timestamping events, then ck(e_1) = ck (e_3)	T	F
8	If we use vector clocks for timestamping events, then ck(e_5) and ck (e_{10}) are not comparable	T	F
9	If we use vector clocks for timestamping events, then ck(e_8) < ck (e_{10})	T	F
10	If we use vector clocks for timestamping events, then ck (e_4) = [1, 1, 1]	T	F





Ex 3: Let us consider the following execution history



Assuming that the initial value stored in the register is 0, assess the truthfulness of every statement and provide a motivation for your answer:

		T	F
1	If the proposed run refers to a regular register, then a is a valid value for read$_1$	✓	F
2	If the proposed run refers to a regular register, then 0 is not a valid value for read$_3$	✓	F
3	If the proposed run refers to a regular register, then a is not a valid value for read$_2$	T	✗
4	If the proposed run refers to a regular register, then b is a valid value all read operations	✓	F
5	If the proposed run refers to a regular register, then read$_3$ may return only a and b	T	✗
6	If the proposed run refers to an atomic register, then read$_1$ and read$_2$ may return different values	✓	F
7	If the proposed run refers to an atomic register, then read$_3$ returns b if and only if read$_2$ returns b	T	✗
8	If the proposed run refers to an atomic register, then read$_4$ and read$_5$ must always return the same value	T	✗
9	If the proposed run refers to an atomic register and read$_4$ returns c, then read$_5$ may returned b or c	T	✗
10	If the proposed run refers to an atomic register then read$_5$ can never return the value c	T	✗

1) $R_1()$: 0, a, b

2) $R_3()$: a, b, c

3) $R_2()$: a, b, c

4) $R_5()$: b, c

5) $R_3()$: a, b, c

6) ONE CAN READ 0 AND THE OTHER a

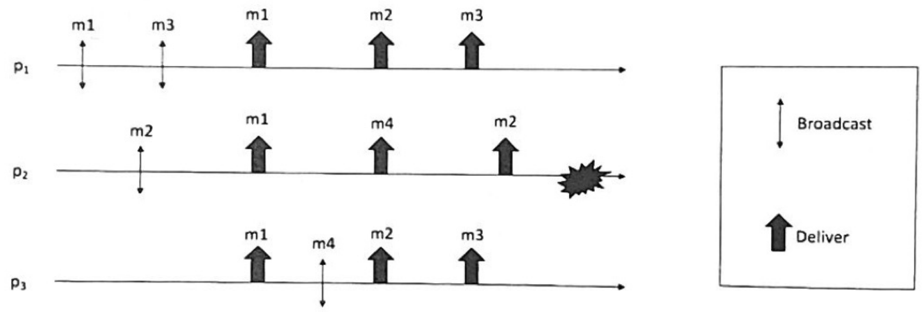
7) $R_3()$: b ALSO IF $R_2()$: a

8) R_5 CAN RETURN c AND R_4 b

9) R_5 MUST RETURN c TO SATISFIE ATOMIC REGISTER

10) IT CAN

Ex 4: Consider the partial execution in the following figure



	Given the run depicted in the figure state the truthfulness of the following sentences:		
a	The run satisfies the Reliable Broadcast specification	T	F ✗
b	The run satisfies the Best Effort Broadcast specification	T	F ✗
c	The strongest ordering property satisfied is SUTO	T	F ✗
d	The WUTO ordering property is satisfied	T ✓	F
e	The run satisfies the FIFO ordering property	T ✓	F
f	The run satisfies the Causal ordering property	T ✓	F
g	If correct processes add the delivery of m4 as last message, then the resulting run satisfies the Reliable Broadcast specification	T ✓	F
h	If correct processes add the delivery of m4 as last message, then the resulting run satisfies TO (UA, WNUTO)	T ✓	F
i	If correct processes add the delivery of m4 between the delivery of m2 and m3, then the resulting run does not satisfy causal order	T	F ✗
l	If p2 adds the delivery of m4 as last message, then the resulting run satisfies the Reliable Broadcast specification	T	F ✗

a) VALIDITY IS NOT SATISFIED, m_4 IS NOT DELIVERED BY P_4

b) VALIDITY IS NOT SATISFIED, m_4 IS NOT DELIVERED BY P_4

c) THE STRONGEST IS WUTO

d) IF p AND q BOTH TODELIVER m AND m', THEN p TODELIVERS m BEFORE m' IF AND ONLY IF q TODELIVERS m BEFORE m'.

e) $m_1 \rightarrow m_3$ FIFO ORDER

f) CASUAL ORDER: FIFO + LOCAL
$m_1 \rightarrow m_3$ FIFO ORDER / $m_1 \rightarrow m_4$ LOCAL ORDER

g) VALIDITY IS NOW SATISFIED

h) P_2 DOESN'T SATISFY WNUTO

i) THE RUN SATISFIES CASUAL

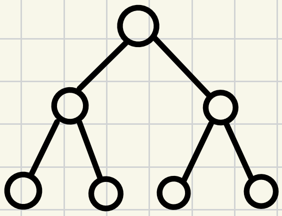
l) VALIDITY IS NOT SATISFIED, m_4 IS NOT DELIVERED BY P_4

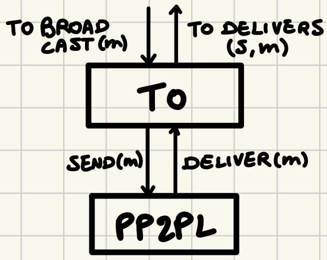
Ex 5: Consider a distributed systems composed by n processes $p_1, p_2, \ldots p_n$, with a unique identifier that are structured as a binary tree. Messages are exchanged between processes over the edges of the tree which acts like perfect point-to-point links. Each process p_i has stored the identifiers 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).

a) Write the pseudo code of an algorithm implementing a primitive of total order broadcast assuming no process fails.

b) Discuss which properties of the total order are violated in the presence of even a single crash of a process.

c) Discuss how to modify your algorithm assuming that processes may fail by crash but are equipped with a perfect failure detector.





a)

```
UPON EVENT <TO, INIT> DO
  VALUES = ∅
  FATHER = GET(FATHER, SELF)      SE NULL È IL LEADER
  R.CHILD = GET(R.CHILD, SELF)
  L.CHILD = GET(L.CHILD, SELF)
  τs = 0                          USED BY LEADER
  NEXT.τs = 0                     USED BY ALL

UPON EVENT <TO, BROADCAST | [m, P]> DO
  IF FATHER == NULL THEN
    τs = τs + 1
    IF L.CHILD != NULL
      TRIGGER <PP2PL, SEND | [BRDCST, m, τs, p]> TO L.CHILD
    IF R.CHILD != NULL
      TRIGGER <PP2PL, SEND | [BRDCST, m, τs, p]> TO R.CHILD
  ELSE
    TRIGGER <PP2PL, SEND | [REQ, m, SELF]> TO FATHER

UPON EVENT <PP2PL, DELIVER | [REQ, m, p]> FROM q
  IF FATHER != NULL THEN
    TRIGGER <PP2PL, SEND | [REQ, m, p]> TO FATHER
  ELSE
    TRIGGER <TO, BROADCAST | [m, p]>

UPON EVENT <PP2PL, DELIVER | [BRDCST, m, τs', p]> FROM q
  VALUES = VALUES ∪ {m, τs', p}

WHEN ∃ (m, τs', p) ∈ VALUES SUCH AS τs' = NEXT.τs THEN
  VALUES = VALUES \ {m, τs', p}
  NEXT.τs = NEXT.τs + 1
  TRIGGER <TO, DELIVER | m, p>
```

b) WE WOULD BREAK VALIDITY, SINCE WE MAY LOOSE MESSAGES SO A BRDCST REQ MAY NEVER BE DELIVERED BY ALL PROCESSES.

c) IF WE HAD A CRASH OF A LEAF, NO PROBLEM. IN THE CASE OF A NODE IN BETWEEN WE MAY NEED TO CREATE A PARTITION. WE ALSO NEED TO CLEAR THE BUFFER OF FAILED PROCESSES AND UPDATE NEXT.TS ACCORDING "IF NEXT.TS == TS.OF.FAILED.PROCESS THEN NEXT.TS ++".