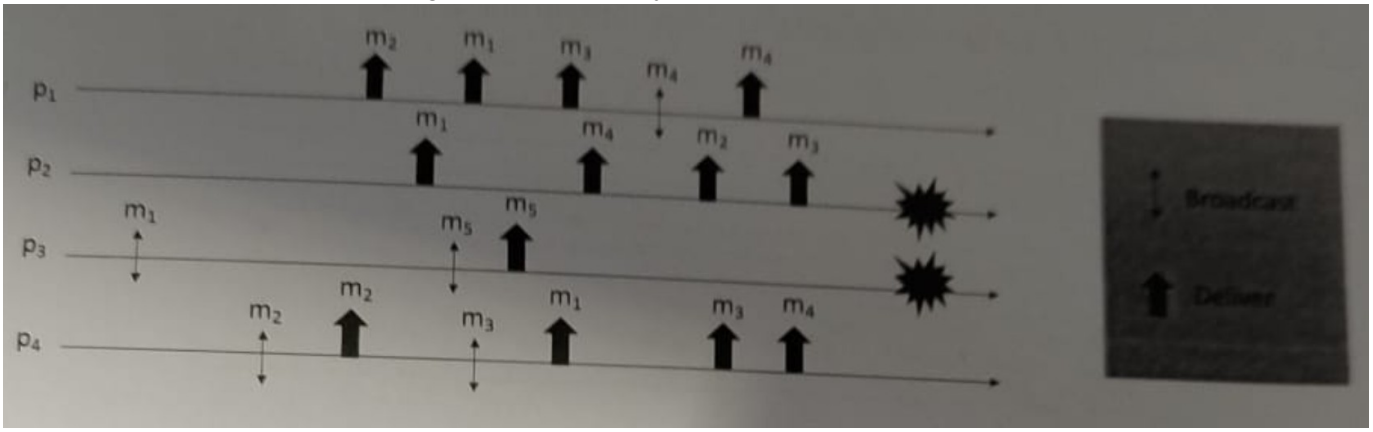
2025-01-29 (Exam A)

Ex. 1: Provide the specification of the leader election primitive, describe its implementation discussion in detail the system model

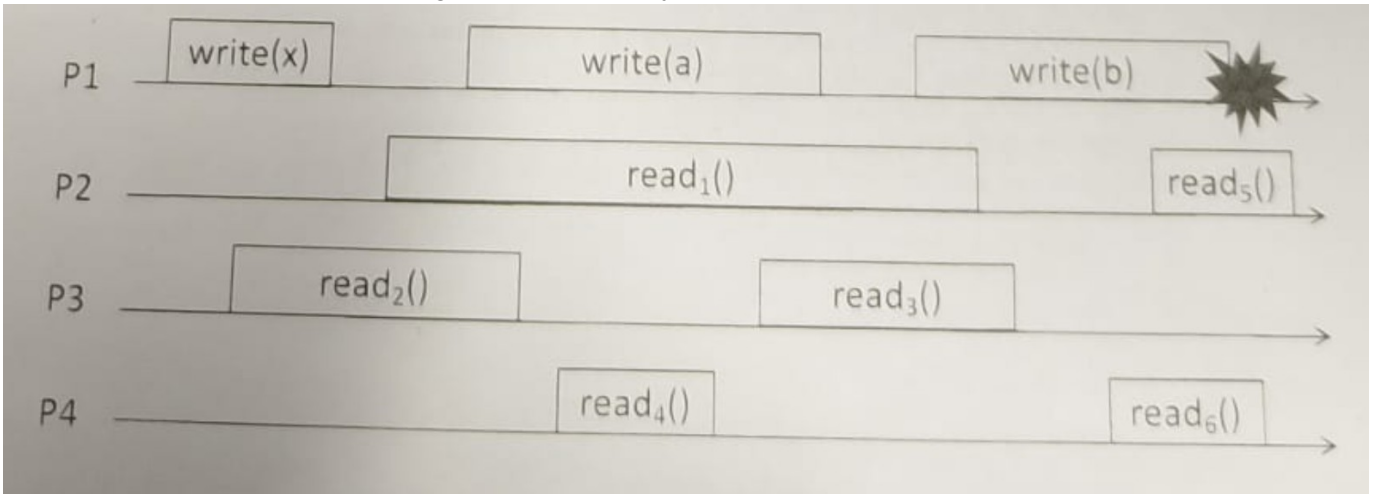
Ex 2. Let us consider the following execution history



Asses the truthfulness of every statement and provide a motivation for your answer:

1. The proposed run satisfies the Best Effort Broadcast specification
2. The strongest specification satisfied by the proposed run is Uniform Reliable Broadcast
3. If p_3 delivers m_1, then the resulting run satisfies the Uniform Reliable Broadcast specification
4. If p_3 does not deliver m_5, then the resulting run satisfies the Uniform Reliable Broadcast specification
5. If p_3 does not crash, then the resulting run satisfies the Best Effort Broadcast specification
6. The run satisfies TO(NUA, WNUTO)
7. The run satisfies the FIFO Broadcast specification
8. Let us assume p_3 does not deliver m_5, then the resulting run satisfies TO(UA, WNUTO)
9. Let us assume p_2 does not crash, then the resulting run satisfies TO(NUA, WNUTO)
10. If p_2 does not deliver m_2 and m_3, then the resulting run satisfies TO(NUA, WUTO)

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 0 is a valid value for $read_1$
2. If the proposed run refers to a regular register, then 0 is not a valid value for $read_2$
3. If the proposed run refers to a regular register, then a is a valid value for $read_4$
4. If the proposed run refers to a regular register, then x can be returned only by $read_2$

5. If the proposed run refers to a regular register, then $read_5$ may return only b
6. If the proposed run refers to an atomic register, then $read_3$ and $read_2$ must return the same value
7. If the proposed run refers to an atomic register, then $read_5$ returns b if and only if $read_3$ returns b
8. If the proposed run refers to an atomic register, then $read_6$ and $read_5$ must always return the same value
9. If the proposed run refers to an atomic register and $read_3$ returns b, then $read_6$ necessarily returned b
10. If the proposed run refers to an atomic register and $read_2$ returned a then $read_4$ can return only the value a

Ex. 4: Let us consider the following algorithm implementing a publish-subscribe primitive

Algorithm 1: Publisher code

```
1  Init
2  brokers[] ← get_brokers(); /* array of length |T| where in the i-th entry the
       publisher stores the set of brokers managing topic t_i                    */
3  upon event publish(e_i, t_i)
4  foreach b_j ∈ broker[t_i] do
5  |  trigger pp2pSend(PUBLISH, e_i, t_i) to b_j;
6  end
```

Algorithm 2: Subscriber code

```
1  Init
2  brokers[] ← get_brokers(); /* array of length |T| where in the i-th entry the
       subscriber stores the set of brokers managing topic t_i                   */
3  my_subscription ← get_my_initial_subscription(); /* set of topics            */
4  notified[] ← [∅]^|T|; /* array of length |T| where in the i-th entry the
       subscriber stores the set of notified events for topic t_i                */
5  upon event subscribe(t_i)
6  my_subscription ← my_subscription ∪ {t_i};
7  foreach b_j ∈ brokers[t_i] do
8  |  trigger pp2pSend(SUBSCRIBE, s_i, t_i) to b_j;
9  end
10 upon event unsubscribe(t_i)
11 my_subscription ← my_subscription \ {t_i};
12 foreach b_j ∈ brokers[t_i] do
13 |  trigger pp2pSend(UNSUBSCRIBE, s_i, t_i) to b_j;
14 end
15 upon event pp2pDeliver(NOTIFY, e_j, t_j)
16 if t_j ∈ my_subscription AND e_j ∉ notified[t_j] then
17 |  trigger NOTIFY(e_j, t_j);
18 |  notified ← notified ∪ {(e_j, t_j)};
19 end
```

Algorithm 3: Broker code

```
1  Init
2  subscription[] ← [∅]^|T|; /* array of length |T| where in the i-th entry the
       broker stores the set of subscribers for topic t_i                        */
3  topic_to_manage ← get_topic_to_manage(); /* set of topics that the current
       broker must manage                                                        */
4  upon event pp2pDeliver(PUBLISH, e_j, t_j)
5  if t_j ∈ topic_to_manage then
6  |  foreach s_k ∈ subscription[t_j] do
7  |  |  trigger pp2pSend(NOTIFY, e_j, t_j) to s_k;
8  |  end
9  end
10 upon event pp2pDeliver(SUBSCRIBE, s_j, t_j)
11 subscription[t_j] ← subscription[t_j] ∪ {s_j};
12 upon event pp2pDeliver(UNSUBSCRIBE, s_j, t_j)
13 subscription[t_j] ← subscription[t_j] \ {s_j};
```

Let us assume that

1. the underlying communication system is synchronous and that every message sent over perfect point to point link takes at most δ time to be delivered.
2. links guarantee the FIFO property
3. for each topic t_i, there are at least 3 brokers managing it

Asses the truthfulness of every statement and provide a motivation for your answer:

1. If subscriptions are not modified and there is at most one failure in the system, then every event e publish on a topic t_i will be eventually notified to all subscribers subscribed to it
2. If subscriptions are not modified and there are no failures in the system, then any pair of subscribers s_i and s_j both subscribed to the same topic t_i will deliver the same set of events published on t_i
3. If subscriptions are not modified and there is at most one failure in the system, if a subscriber s_i notifies an event e before an event e' (both published on a topic t_i), then any other subscriber having t_i in its subscriptions will notify e before e'
4. In absence of failures, every event published on a topic t_i at time τ will be eventually notified to every subscriber having t_i in its subscription at time τ
5. In absence of failures, if a subscribers s_i unsubscribes a topic t_i at time τ, it will not notify any event for topic t_i after time τ

Ex. 5:
Let us consider a distributed system composed of n processes p_1, p_2, …, p_n partitioned in two disjoint sets acting as clients and replicas. Each process is identified by a unique integer identifier and can communicate with other processes by exchanging messages using perfect point-to-point links. Processes collaborate to implement a replication protocol emulating a linearizable append-only log.

The append-only log is characterized by the following operations:

- **Append(v)**: appends the value v to the queue of the log
- **Get**: returns the content of the log, e.g. the list of values in the order in which they have been appended.

Assuming that:

1. processes may fail by crash
2. Every pair of processes can communicate exchanging messages over perfect point-to-point links
3. All processes have access to a perfect failure detector
4. Replicas have access to a uniform reliable broadcast primitive

Write the pseudocode of the distributed algorithm implementing the append-only log.

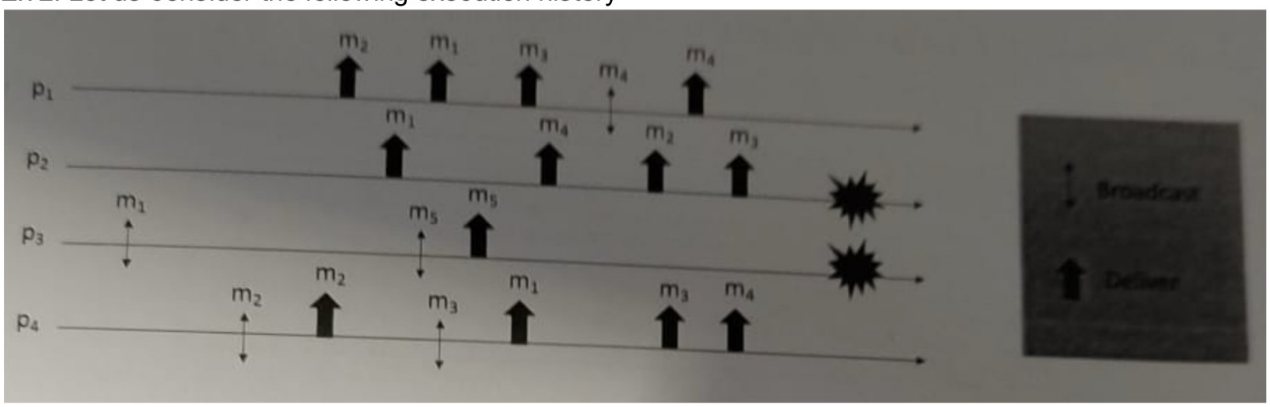
Ex. 1: Provide the specification of the leader election primitive, describe its implementation discussion in detail the system model

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, 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



Asses the truthfulness of every statement and provide a motivation for your answer:

1. The proposed run satisfies the Best Effort Broadcast specification
2. The strongest specification satisfied by the proposed run is Uniform Reliable Broadcast
3. If p_3 delivers m_1, then the resulting run satisfies the Uniform Reliable Broadcast specification
4. If p_3 does not deliver m_5, then the resulting run satisfies the Uniform Reliable Broadcast specification
5. If p_3 does not crash, then the resulting run satisfies the Best Effort Broadcast specification
6. The run satisfies TO(NUA, WNUTO)
7. The run satisfies the FIFO Broadcast specification
8. Let us assume p_3 does not deliver m_5, then the resulting run satisfies TO(UA, WNUTO)
9. Let us assume p_2 does not crash, then the resulting run satisfies TO(NUA, WNUTO)
10. If p_2 does not deliver m_2 and m_3, then the resulting run satisfies TO(NUA, WUTO)

1) T: VALIDITY, NO DUPLICATION AND NO CREATION ARE SATISFIED.

2) F: m_5 IS NOT DELIVERED BY p_1, p_2 AND p_4.

3) F: SAME AS 2.

4) T: ALL MSGs DELIVERED (BY FAULTY OR CORRECT PROCESSES) ARE DELIVERED BY ALL CORRECT PROCESSES.

5) F: VALIDITY IS NOT SATISFIED.

6) T: NUA: m_5 IS NOT DELIVERED BY CORRECT PROCESSES.
WNUTO: IF p AND q, CORRECT PROCESSES, BOTH DELIVER m AND m', p DELIVERS m BEFORE m' ONLY IF q DELIVERS m BEFORE m'

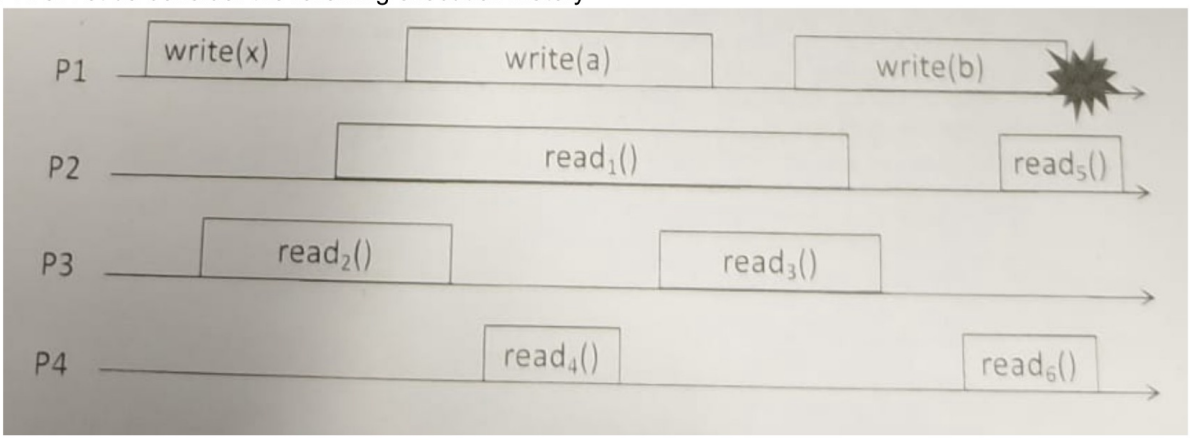
7) T: $m_2 \rightarrow m_3$ SATISFIED (ONLY CORRECT)

8) T: SAME AS 4

9) F: WNUTO NOT SATISFIED (Ex. m_1 AND m_2)

10) T. WUTO SATISFIED

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 0 is a valid value for read$_1$
2. If the proposed run refers to a regular register, then 0 is not a valid value for read$_2$
3. If the proposed run refers to a regular register, then a is a valid value for read$_4$
4. If the proposed run refers to a regular register, then x can be returned only by read$_2$
5. If the proposed run refers to a regular register, then read$_5$ may return only b
6. If the proposed run refers to an atomic register, then read$_3$ and read$_2$ must return the same value
7. If the proposed run refers to an atomic register, then read$_5$ returns b if and only if read$_3$ returns b
8. If the proposed run refers to an atomic register, then read$_6$ and read$_5$ must always return the same value
9. If the proposed run refers to an atomic register and read$_3$ returns b, then read$_6$ necessarily returned b
10. If the proposed run refers to an atomic register and read$_2$ returned a then read$_4$ can return only the value a

1) F : $R_1()$: x, a, b

2) F : $R_2(2)$. 0, x, a

3) T : $R_4()$: x, a

4) F : x CAN BE RETURNED BY R_1, R_2, R_3, R_4

5) F : $R_5()$: a, b

6) F : THEY ARE NOT CONCURRENT

7) F : $R_5()$: b ALSO IF $R_3()$: a

8) F : THEY ARE CONCURRENT AND RETURN SAME VALUES, BUT NOT ALWAYS. IT CAN BE $R_6()$: b , $R_5()$: a

9) T

10) T

Ex. 4: Let us consider the following algorithm implementing a publish-subscribe primitive

Algorithm 1: Publisher code

```
1 Init
2 brokers[] ← get_brokers(); /* array of length |T| where in the i-th entry the
      publisher stores the set of brokers managing topic t_i                */
3 upon event publish(e_i, t_i)
4 foreach b_j ∈ broker[t_i] do
5 |  trigger pp2pSend(PUBLISH, e_i, t_i) to b_j;
6 end
```

Algorithm 2: Subscriber code

```
1 Init
2 brokers[] ← get_brokers(); /* array of length |T| where in the i-th entry the
      subscriber stores the set of brokers managing topic t_i               */
3 my_subscription ← get_my_initial_subscription(); /* set of topics        */
4 notified[] ← ∅^|T|; /* array of length |T| where in the i-th entry the
      subscriber stores the set of notified events for topic t_i            */
5 upon event subscribe(t_i)
6 my_subscription ← my_subscription ∪ {t_i};
7 foreach b_j ∈ brokers[t_i] do
8 |  trigger pp2pSend(SUBSCRIBE, s_i, t_i) to b_j;
9 end
10 upon event unsubscribe(t_i)
11 my_subscription ← my_subscription \ {t_i};
12 foreach b_j ∈ brokers[t_i] do
13 |  trigger pp2pSend(UNSUBSCRIBE, s_i, t_i) to b_j;
14 end
15 upon event pp2pDeliver(NOTIFY, e_j, t_j)
16 if t_j ∈ my_subscription AND e_j ∉ notified[t_j] then
17 |  trigger NOTIFY(e_j, t_j);
18 |  notified ← notified ∪ {(e_j, t_j)};
19 end
```

Algorithm 3: Broker code

```
1 Init
2 subscription[] ← [∅^|T|]; /* array of length |T| where in the i-th entry the
      broker stores the set of subscribers for topic t_i                    */
3 topic_to_manage ← get_topic_to_manage(); /* set of topics that the current
      broker must manage                                                    */
4 upon event pp2pDeliver(PUBLISH, e_j, t_j)
5 if t_j ∈ topic_to_manage then
6 |  foreach s_k ∈ subscription[t_j] do
7 |  |  trigger pp2pSend(NOTIFY, e_j, t_j) to s_k;
8 |  end
9 end
10 upon event pp2pDeliver(SUBSCRIBE, s_j, t_j)
11 subscription[t_j] ← subscription[t_j] ∪ {s_j};
12 upon event pp2pDeliver(UNSUBSCRIBE, s_j, t_j)
13 subscription[t_j] ← subscription[t_j] \ {s_j};
```

Let us assume that

1. the underlying communication system is synchronous and that every message sent over perfect point to point link takes at most δ time to be delivered.
2. links guarantee the FIFO property
3. for each topic t_i, there are at least 3 brokers managing it

Asses the truthfulness of every statement and provide a motivation for your answer:

1. If subscriptions are not modified and there is at most one failure in the system, then every event e publish on a topic t_i will be eventually notified to all subscribers subscribed to it
2. If subscriptions are not modified and there are no failures in the system, then any pair of subscribers s_i and s_j both subscribed to the same topic t_i will deliver the same set of events published on t_i
3. If subscriptions are not modified and there is at most one failure in the system, if a subscriber s_i notifies an event e before an event e' (both published on a topic t_i), then any other subscriber having t_i in its subscriptions will notify e before e'
4. In absence of failures, every event published on a topic t_i at time τ will be eventually notified to every subscriber having t_i in its subscription at time τ
5. In absence of failures, if a subscribers s_i unsubscribes a topic t_i at time τ, it will not notify any event for topic t_i after time τ

1) F: IF A SUBSCRIBER FAILS, THEN HE WILL NOT RECEIVE ANY NEW EVENT.

2) T: SINCE FIFO PROPERTY, PERFECT P2P LINKS AND NO FAILURES ARE GUARANTEED, S_i AND S_j WILL DELIVER THE SAME SET IN THE SAME ORDER.

3) T: SINCE FIFO PROPERTY IS GUARANTEED, IF S_i NOTIFIES e BEFORE e` THEN EVERY S OF $T_i$ NOTIFIES e BEFORE e`, EVEN WITH A FAILURE.

4) F: TIMES CAN BE DIFFERENT.

5) F: IF THERE IS A DELAY IN UNSUBSCRIBE, S_i COULD RECEIVE SOME EVENT PUBLISHED JUST BEFORE.

Ex. 5:
Let us consider a distributed system composed of n processes p_1, p_2, …, p_n partitioned in two disjoint sets acting as clients and replicas. Each process is identified by a unique integer identifier and can communicate with other processes by exchanging messages using perfect point-to-point links. Processes collaborate to implement a replication protocol emulating a linearizable append-only log.

The append-only log is characterized by the following operations:

- **Append(v)**: appends the value v to the queue of the log
- **Get**: returns the content of the log, e.g. the list of values in the order in which they have been appended.

Assuming that:

1. processes may fail by crash
2. Every pair of processes can communicate exchanging messages over perfect point-to-point links
3. All processes have access to a perfect failure detector
4. Replicas have access to a uniform reliable broadcast primitive

Write the pseudocode of the distributed algorithm implementing the append-only log.