Dependable Distributed Systems Master of Science in Engineering in Computer Science

AA 2023/2024

LECTURES 16: SOFTWARE REPLICATION

Schema

Motivation

- Fault Tolerance
  - Guarantee the availability of a service (also called object) despite failures

- Assuming p the failure probability of an object O. O's availability is 1-p.

- Replicating an object O on n nodes and assuming p the failure probability of each replica, O's availability is $1- p^n$ (considering independent failures probability)

System Model

The system is composed of a set of processes (clients)

- Processes are connected through Perfect point-to-point links
- Processes may fail by crash

Processes interacts wit a set of objects X located at different sites managed managed by processes

- Each object has a state accessed trough a set of operations
- An operation by a process p_i on an object $x \in X$ is a pair invocation/response
  - The operation invocation is noted [*x* op(arg) p_i] where arg are the arguments of the operation op
  - The operation response is noted [x ok(res) p_i] where res is the result returned
  - The pair invocation/response is noted [x op(arg)/ok(res) p_i]
- After issuing an invocation a process is blocked until it receives the matching response

Replication: requirements

In order to tolerate process crash failures a logical object must have several physical replicas located at different sites of the distributed system

- replicas of an object *x* are noted $x^1, x^2, \ldots x^l$
- Invocation of replica x^j located on site s is handled by a process p_j also located on s
- We assume that p_j crashes exactly when x^j crashes

Replication is transparent to the client processes

Consistency criteria

A consistency criterion defines the result returned by an operation

- It can be seen as a contract between the programmer and the system implementing replication

Three main consistency criteria are defined in literature

- Linearizability
- Sequential consistency
- Causal consistency

Linearizability, Sequential consistency: Strong Consistency

Causal consistency: Weak Consistency

The three consistency criteria differ however in the definition of the most recent state

Linearizability

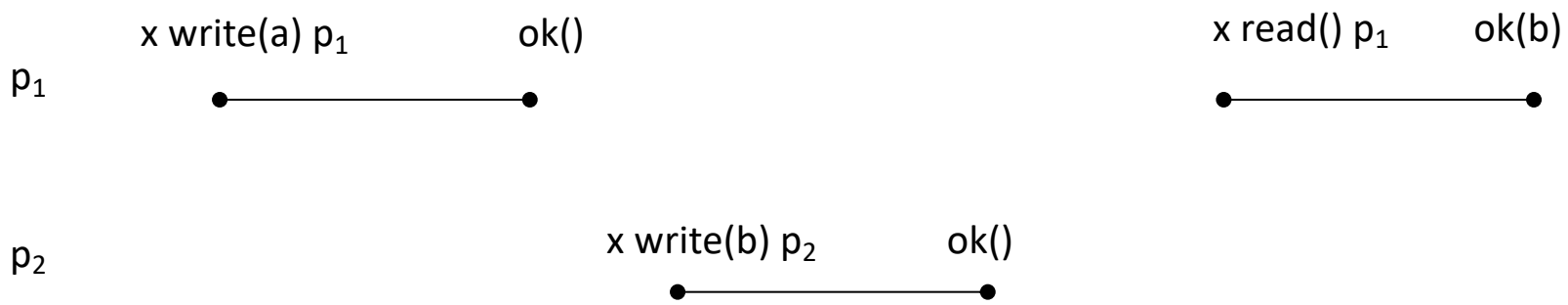
Let us consider the precedence relation (denoted $\prec$) and the concurrency relation (denoted $||$) defined between two operations.

An execution E is linearizable if there exists a sequence S including all operations of E such that the following two conditions hold:

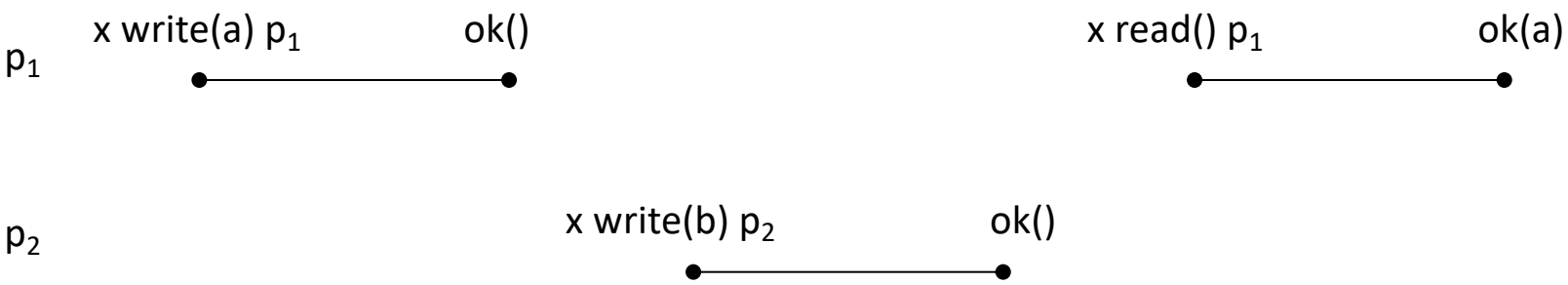
1. for any two operations O_1 and O_2 such that $O_1 \prec O_2$, O_1 appears before O_2 in the sequence S
2. the sequence S is *legal* i.e., for every object x the subsequence of S of which operations are on x belongs to the sequential specification of x.

Example: simple variable

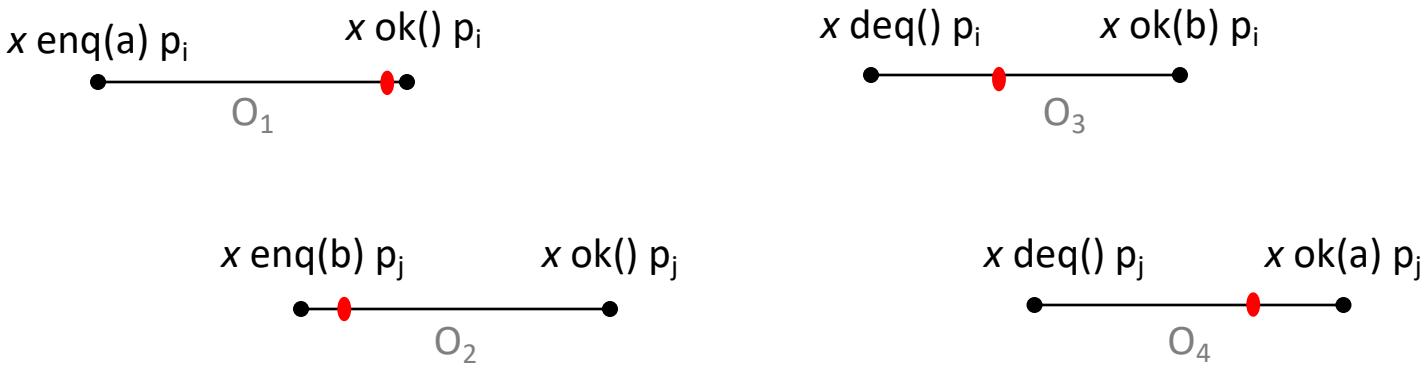
1. LINEARIZZABILE



2. NON LINEARIZZABILE



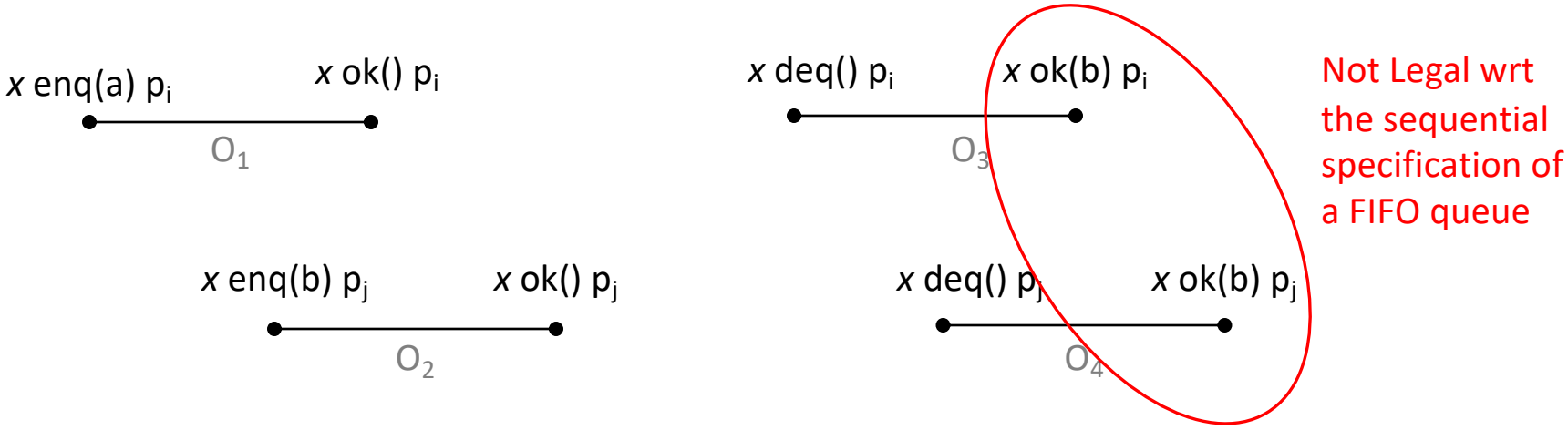
Example: FIFO Queue



Linearizable

S= {O2, O1, O3, O4}

Example: FIFO Queue



NON-Linearizable

A Sufficient Condition for Linearizability

Replicas must agree on the set of invocations they handle and on the order according to which they handle these invocations

Atomicity: Given an invocation [*x* op(arg) p_i], if one replica of the object x handles this invocation, then every correct replica of x also handles the invocation [*x* op(arg) p_i].

Ordering: Given two invocations [*x* op(arg) p_i] and [*x* op(arg) p_j] if two replicas handle both the invocations, they handle them in the same order

Replication Techniques

Two main techniques implementing linearizability:

- Primary Backup
- Active Replication

Passive Replication

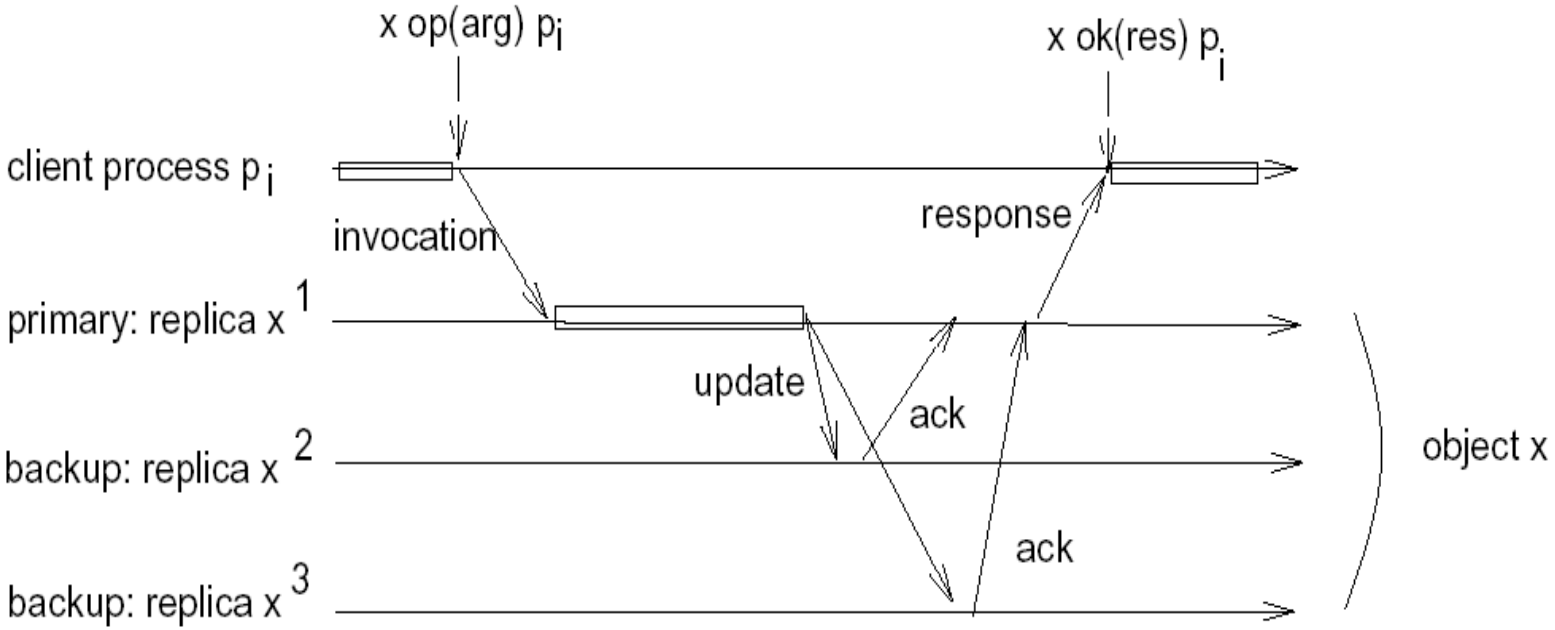
PRIMARY-BACKUP

Primary Backup

- *Primary*:
  - Receives invocations from clients and sends back the answers.
  - Given an object x, *prim(*x*)* returns the primary of x.

- *Backup:*
  - Interacts with *prim(*x*)*
  - is used to guarantee fault tolerance by replacing a primary when crashes

Primary Backup Scenario



Primary Backup: the case of no crash

1. When update messages are received by backups, they update their state and send back the ack to prim(x).
2. prim(x) waits for an ack message from each correct backup and then sends back the answer, res, to the client.

How to guarantee Linearizability: the order in which prim(x) receive clients' invocations define the order of the operation on the object.

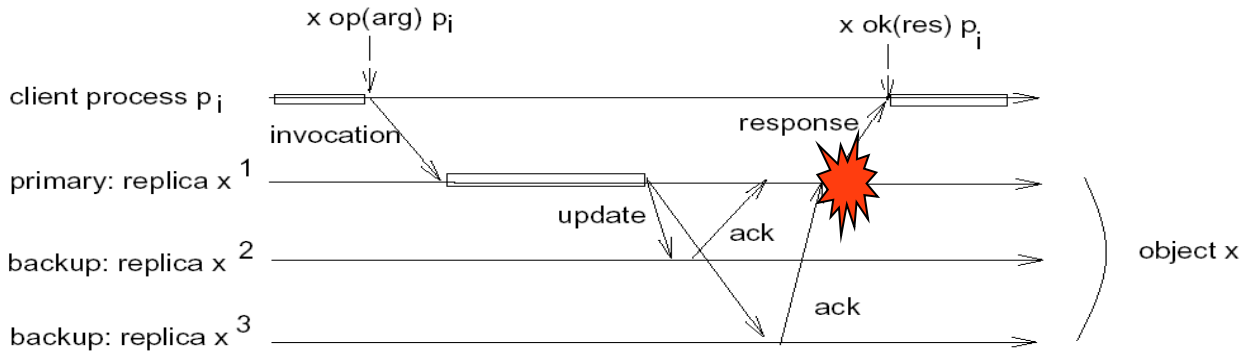
Primary Backup: Presence of Crash

Three scenarios :

- **Scenario 1**: Primary fails after the client receives the answer.
- **Scenario 2**: Primary fails before sending update messages
- **Scenario 3**: Primary fails after sending update messages and before receiving all the ack messages.

In all cases there is the need of electing a new leader.

Scenario 1
Primary fails after sending the answer

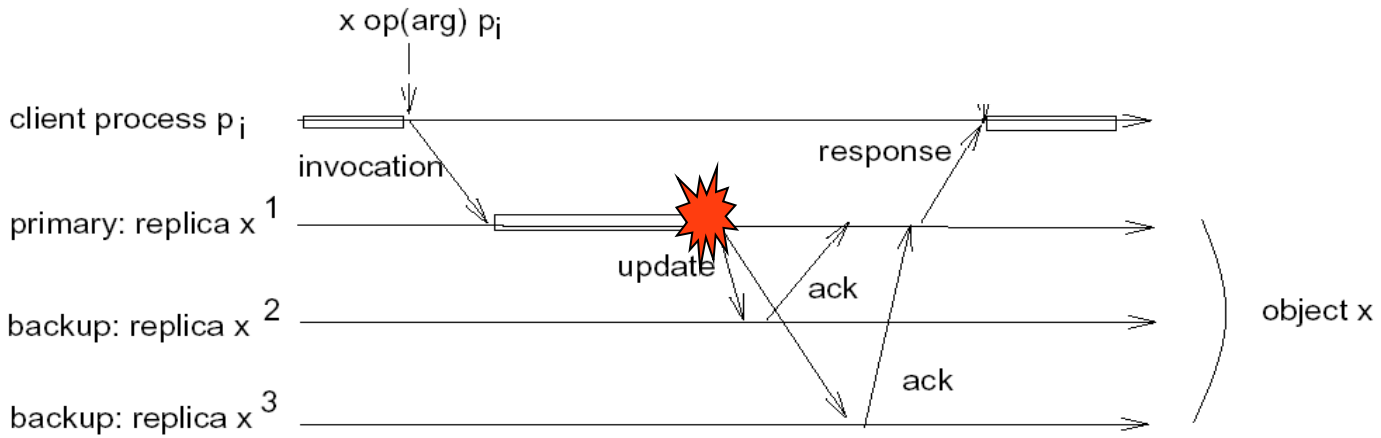


Two cases

1. Client does not receive the response due to perfect point-to-point link. If the response is lost, client retransmits the request after a timeout
2. Client receives the answer, everybody is happy (but the primary ☺)

The new primary will recognize the request re-issued by the client as already processed and sends back the result without updating the replicas

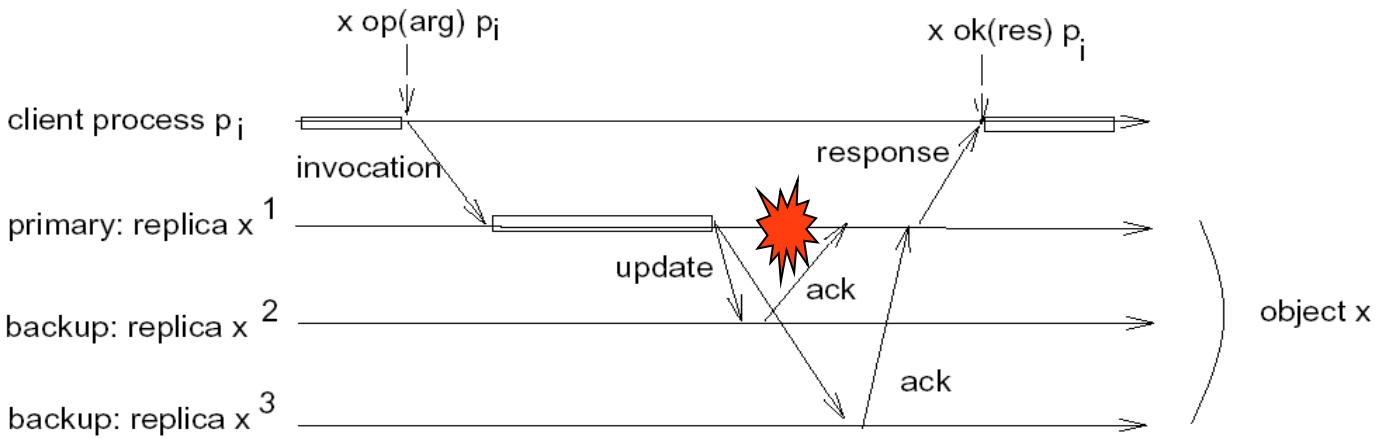
Scenario 2
Primary fails before sending update messages



Client does not get an answer and resends the requests after a timeout

The new primary will handle the request as new

Scenario 3
Primary fails after sending update messages and before receiving all the ack messages



How to Guarantee atomicity? update it is received either by all or by no one.

When a primary fails there is the need to elect another primary among the correct replicas

Active Replication

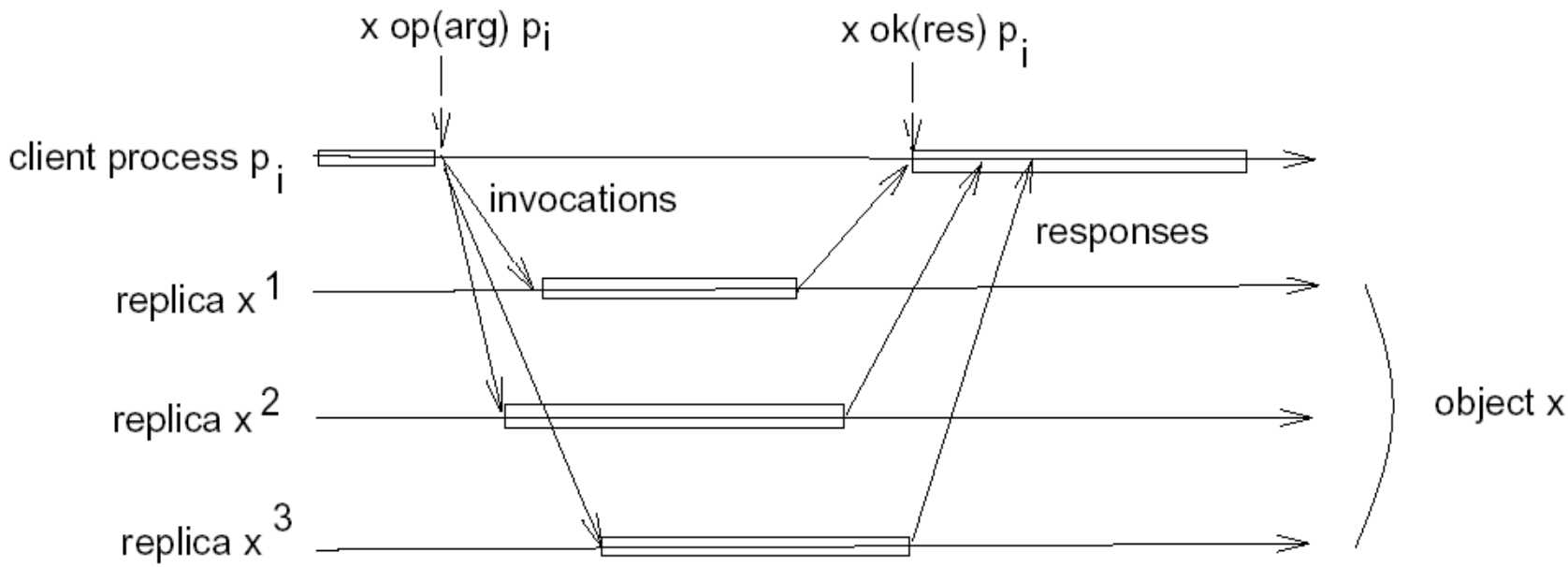
Active Replication

There is no coordinator all replicas have the same role

Each replica is deterministic. If any replica starts from the same state and receives the same input, they will produce the same output

As a matter of fact clients will receive the same response one from each replica

Active Replication



Active Replication
How to Guarantee Linearizability

To ensure linearlizability we need to preserve:

- Atomicity: if a replica executes an invocation, all correct replicas execute the same invocation.
- Ordering: (at least) no two correct replicas have to execute two invocations in different order.

We need: TOTAL ORDER Broadcast

- INCLUDING THE CLIENTS!

Active Replication Crash

Active Replication does not need recovery action upon the failure of a replica

References

Rachid Guerraoui and André Schiper: "*Fault-Tolerance by Replication in Distributed Systems*". In *Proceedings of the 1996 Ada-Europe International Conference on Reliable Software Technologies* (Ada-Europe '96).

The primary-backup and the active replication are two techniques that implement linearizability.
The first can be explain as: primary means that receives invocation from clients and sends back the answers; also given an objects x, prim(x) returns the primary of x. While backup means that it interacts with prim(x) and is used to guarantee fault tolerance by replacing a primary when there're crashes.
In the case of NO CRASH scenario, before sending back the response to the client, the primary replica sends an update to all the other correct backups and, only after it gets an ACK from them, it will send a response to the client.
In the case of CRASH, we have different scenarios:
① the primary fails after the client receives the answer and there are two cases:
the client doesn't receive the response due to PP2P link and the client needs to retrasmit the requests;
the client receives the answer and it's fine.
② the primary fails before sending update msgs:
client doesn't gets answer and needs to resend the requests that is handled as new.
③ the primary fails after sending update msgs but before receiving all ACK.
The update is received either by all or by no one, so the primary have to be elected among all replicas.

In the active replication there isn't a coordinator, all replicas have the same role. Each replica is deterministic, so the client receives the same response for all replicas. It have to ensure fivenizability and needs to preserve atomicity: if a replica executes an invocation, all correct replicas execute the same invocation, and ordering: no two correct replica have to execute two invocations in different order.
It doesn't need recovery action upon the failure of a replica.