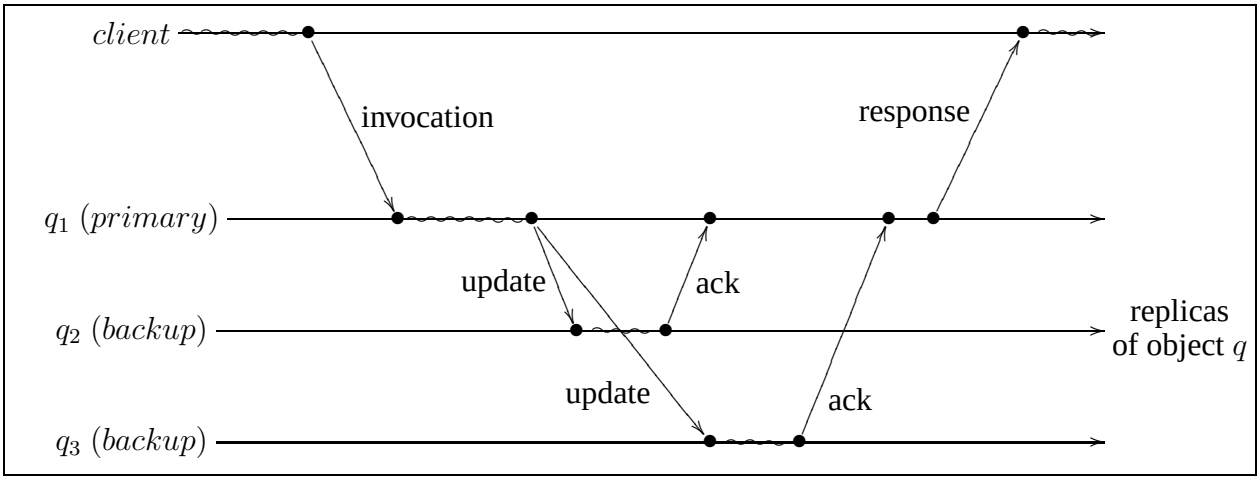
2.5.1 Primary-backup replication

We describe the principle of this technique, and ignore failures for a while.



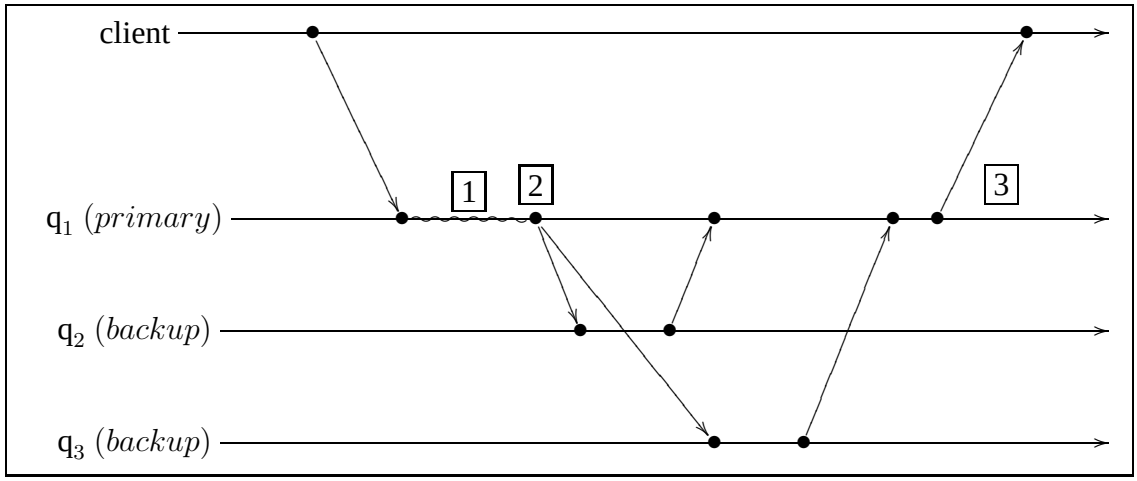
The principle is the following:

- q_1: primary copy; q_2, q_3: backups
- the client sends its invocation to the primary q_1
- q_1 receives the invocation and performs the operation. At the end of the operation, the change of the state of q_1 is forwarded to q_2 and q_3 ("update" message). The "ack" is sent by the backups after they have updated their states. The primary sends the response to the client after having received "ack" from all backups.

If the primary does not crash, then *order* and *atomicity* are ensured.

- the order is defined by the primary.
- atomicity is ensured because the update is forwarded to all the backups and "ack" is awaited by the primary before sending the response.

The crash of a backup is easy to handle. The crash of the primary is more difficult to handle. There are three cases to distinguish:



1. The primary crashes before sending the "update" message. In this case, the client will *time-out* waiting for the *response*, and must re-send the invocation to the new primary.

2. The primary crashes while sending the "update" message and before sending the response. This is the most difficult case to handle.

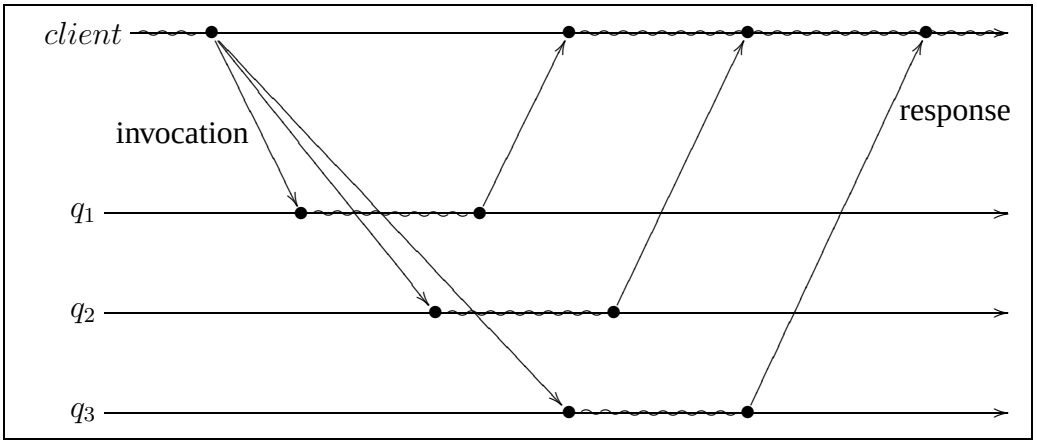
   - Atomicity has to be guaranteed: the "update" message has to be received by *all* or by *none* of the backups.
   - The client must send the invocation to the new primary, but the operation must not be performed twice. *Solution:* every invocation has a unique *InvID* and the "update" message carries the pair $(InvID,\ response)$. If a server q_i receives an invocation with *InvID*, it first checks whether the pair $(InvID,\ response)$ is available. If yes, the invocation is not processed, and $response$ is sent immediately to the client. Otherwise, the invocation is processed.

3. Crash of the primary after sending the response. In this case a new primary has to be selected.

Selection of a new primary and ensuring atomicity in Case 2 is discussed later.

Remark Crash detection is usually based on timeouts, and this mechanism may lead to mistakes, i.e., to suspect a process to have crashed, while the process actually did not crashed. The *order* and *atomicity* properties must be ensured in spite of unreliable failure detection. This makes the problem very difficult. We come back on this issue later.

2.5.2 Active replication



With active replication, the client sends its invocation to all replicas:

- All replicas handle the invocation and send the response.
- The client waits for the first response (all responses are identical).

With active replication, the crash of a replica is transparent to the client. However, active replication requires an adequate communication primitive called *Total Order Broadcast* or *Atomic Broadcast*, which ensures the *order* and *atomicity* properties. The primitive is also called *ABCAST*. All the complexity of active reaplication is hidden in the implementation of Atomic Broadcast.

2.5.3 Comparison between primary-backup and active replication

- Active replication uses more CPU resources (as all replicas handle the invocation).
- In case of a crash (of the primary), the latency for the client between the invocation and the reception of the response is longer with primary-backup replication (timeout of the client, which must re-send the invocation).
- Active replication requires the operations to be *deterministic*! This is not the case with primary-backup replication.