Network and Distributed Computing
Lecture 3

SCS 3103 & IS 3008
Middleware Architectures

Hakim Usoof

University of Colombo School of Computing

Basic Concepts

- What is a Dependable System
  - Availability
  - Reliability
  - Safety
  - Maintainability

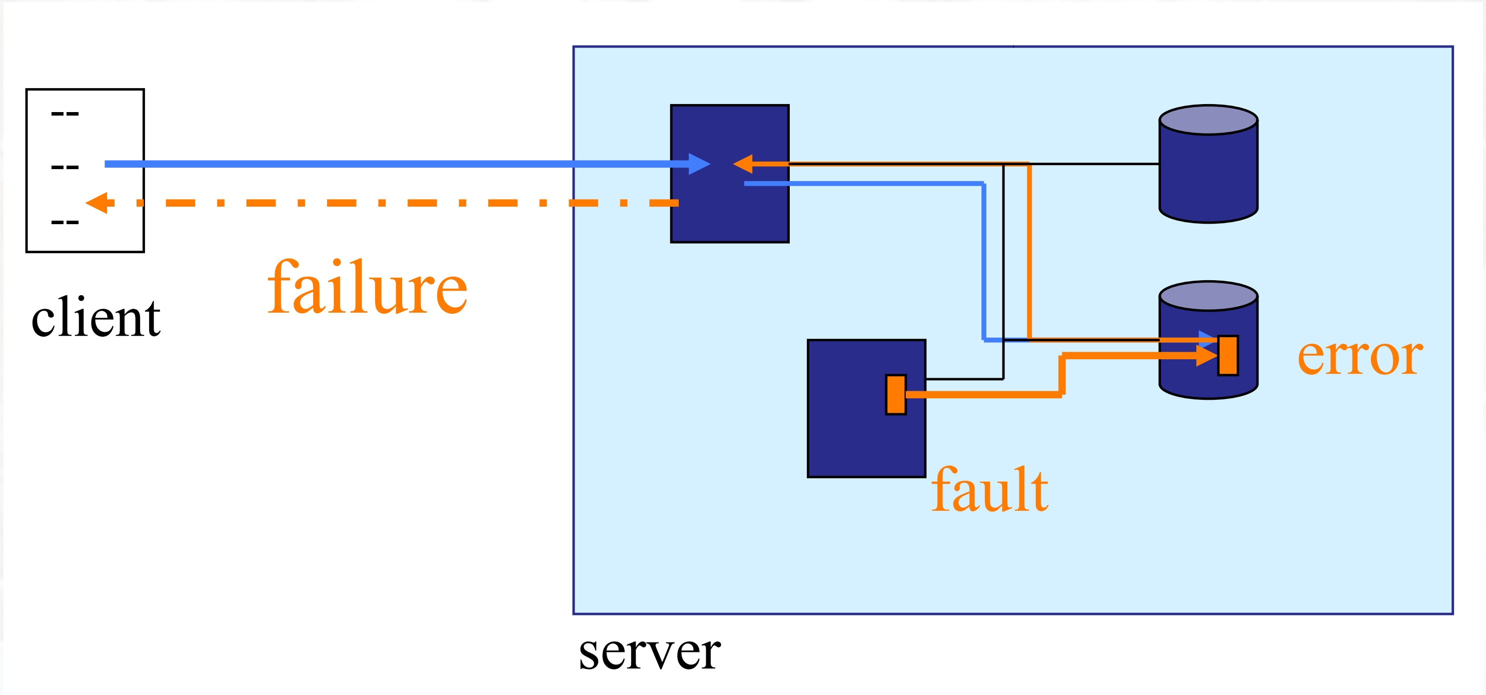
Reliable Systems

- Since more things can go wrong in distributed systems, they are intrinsically less reliable than centralized systems. Many mechanisms have to be built in, e.g., to handle the fact that the client and the server may fail independently *(partial failures)*.
- The network necessary for remote communication between distributed components is another source of unreliability that distributed applications have to deal with.

Aspects of Reliability in a Distributed System

- **Fault tolerance:** fault tolerance is the ability of a distributed computing system to recover from the failure of some component. A component is considered *faulty* once its behavior is no longer consistent with its specification.
- **High availability:** a highly available system provides uninterrupted service in spite of failures.
- **Consistency:** consistency is the ability of a distributed computing system to coordinate related actions of multiple components, despite concurrency and failures. It generally encompasses the ability of making a distributed system behave like a non-distributed system.
- **Security:** security is the ability of a system to protect data, services, and resources against unauthorized access.
- **Privacy:** privacy is the ability of a system to protect user identity and data from other users.

Fault, Error vs. Failure



Failure Models

- Challenge: Independent failures
- Detection
  - Which component
  - What went wrong?
- Recovery
  - Failure dependent
  - If ignored can cause catastrophic failure

Fault Tolerance Model

- Detection
- Recovery
  - Mask Error
  - Fail Predictably
- Designer
  - Possible Failure types
  - Recovery Action

Failure Models (1)

- **Halting failures:** in the halting failures model, a process either works correctly, or simply stops and crashes without performing incorrect actions.
- **Fail-stop failures:** in the fail-stop failures model, in addition to the process crashing without performing incorrect actions, processes that are interacting with the faulty process have an accurate way to detect such failures.

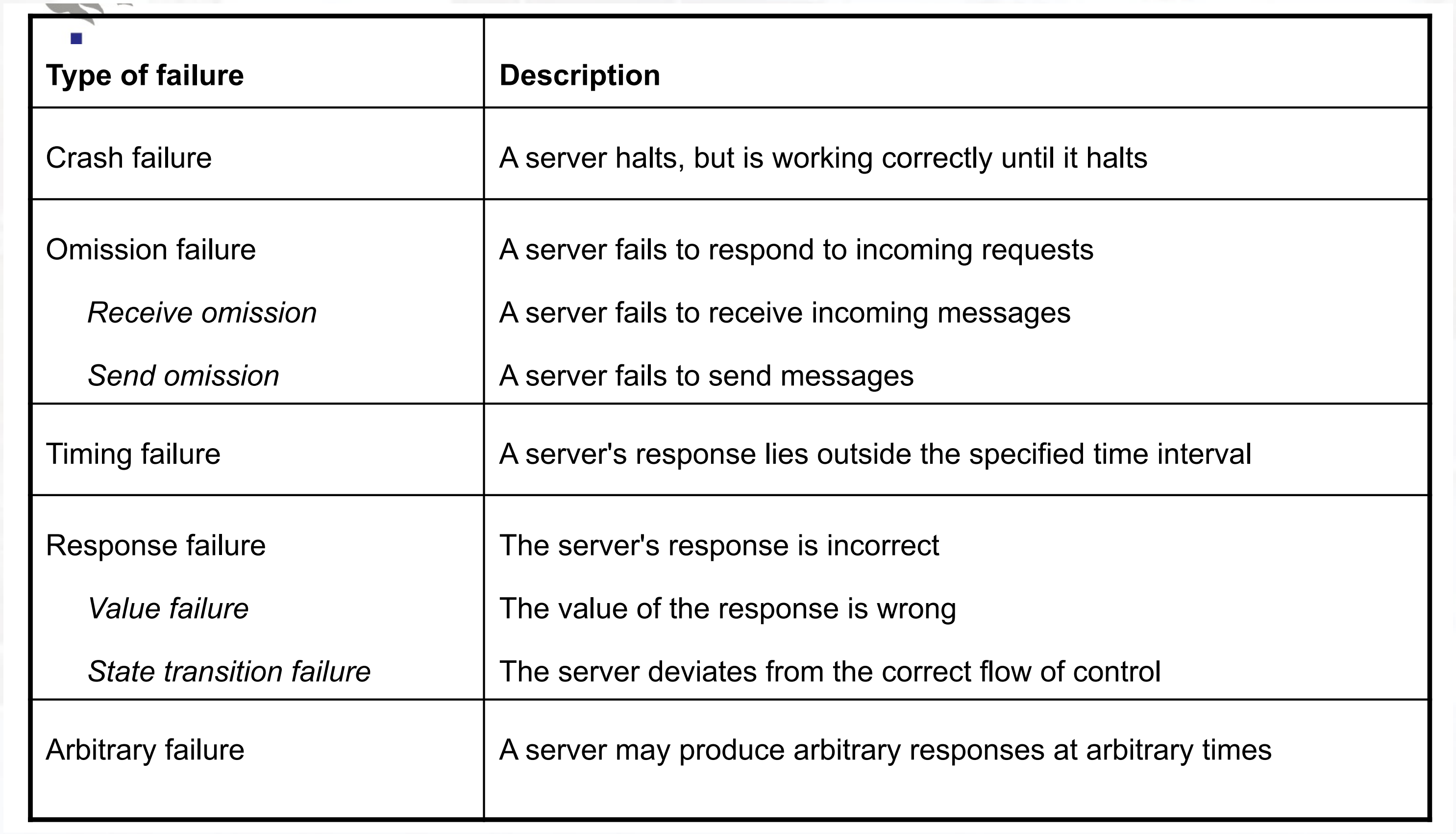
Failure Models (2)

- **Send-omission failures:** Server fails to send a message that should have been sent.
- **Receive-omission failures:** Server fails to receive a message that has actually been sent and correctly transmitted by the communication channel. This may happen because of a lack of memory for buffering messages in the destination process.
- **Response value failure:** The value of the response in wrong
- **Response State transmission failure:** The server deviates from the correct flow of control

Failure Models (3)

- **Network failures:** these are failures that occur when the network loses messages, or is fragmented in disconnected sub-networks that cannot communicate with each other.
- **Timing failures:** Servers responses lies outside the specified time interval
- **Byzantine failures:** Byzantine failures encompass a wide variety of faulty behavior. Malfunctioning or malicious processes can send messages with wrong data, or omit sending messages when they have to. These types of failures are difficult to even detect, let alone correct, since failed processes exhibit unpredictable behavior; some systems, such as Rampart, focus on this problem.

Summary



Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts
Omission failure	A server fails to respond to incoming requests
Receive omission	A server fails to receive incoming messages
Send omission	A server fails to send messages
Timing failure	A server's response lies outside the specified time interval
Response failure	The server's response is incorrect
Value failure	The value of the response is wrong
State transition failure	The server deviates from the correct flow of control
Arbitrary failure	A server may produce arbitrary responses at arbitrary times

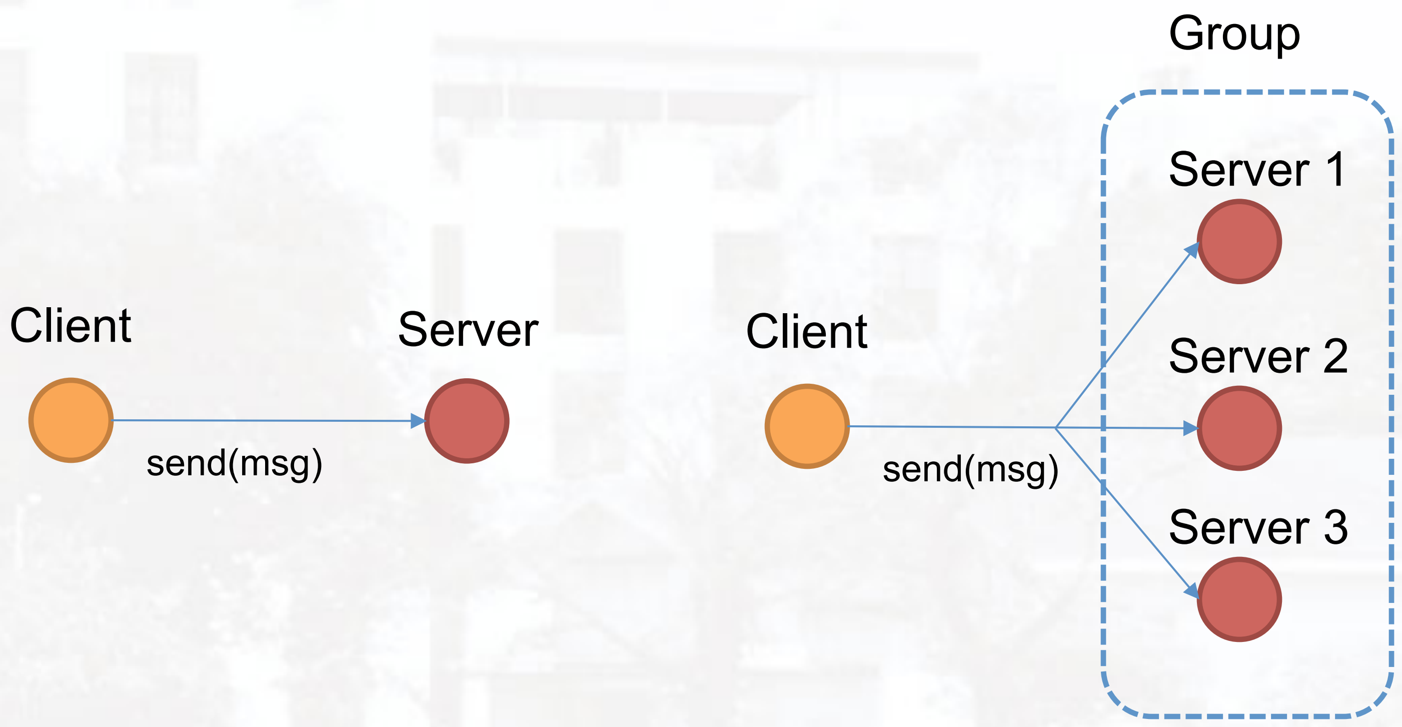
Assignment

- List and explain the Possible actions, processes or approaches of Failure Masking (Handling Failures)
- Detection Phase
  - Eg: Checksum
- Recovery Phase
  - Eg: Error correcting Codes
- Submit Assignment by 4th April 23.55

Group-Based Computing

- *Group communication* is used gather a set of processes or objects into a logical group, and to provide primitives for sending messages to all group members at the same time with various ordering guarantees.
- A group constitutes a logical addressing facility since messages can be issued to groups without having to know the number, identity, or location of individual members.
- Groups have proven to be very useful for providing *high availability* through *replication*: a set of replicas constitutes a group, viewed by clients as a single entity in the system.

Point-to-Point vs. Group Communication

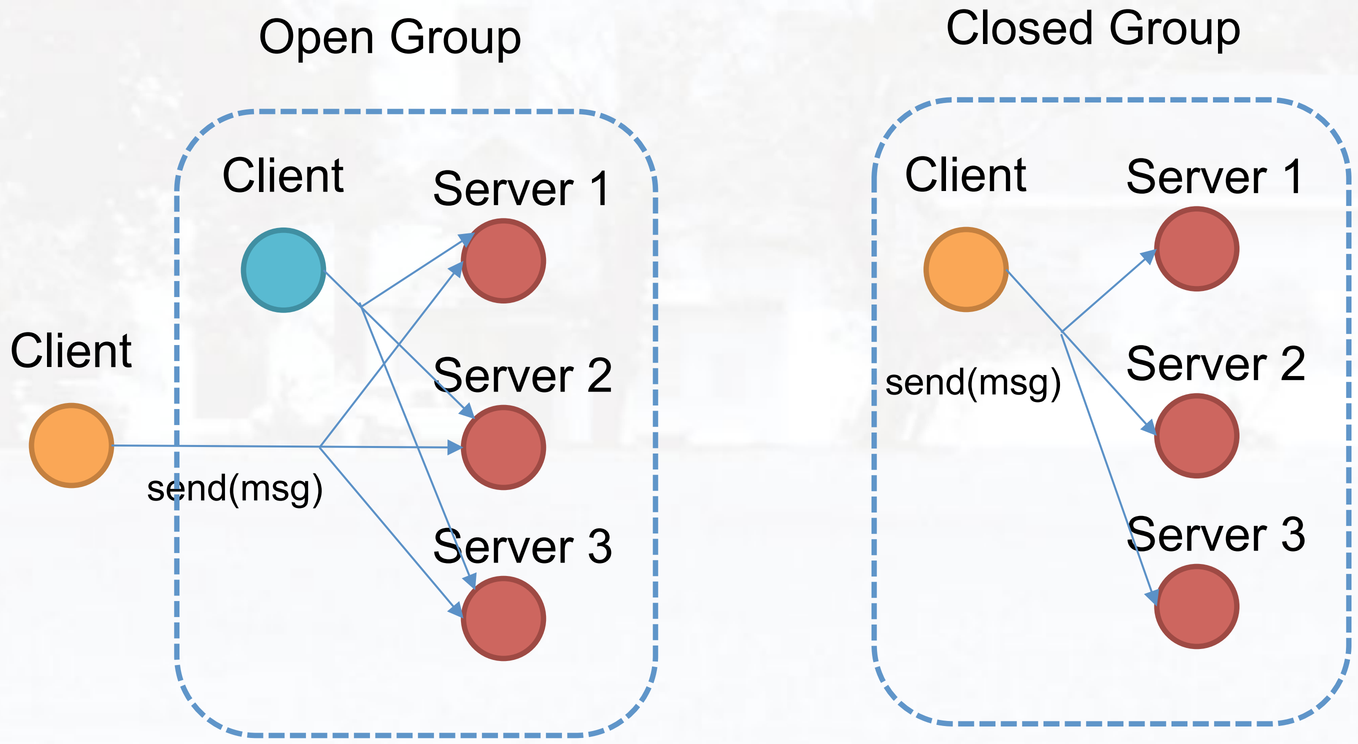


Groups

- Groups may be *static* or *dynamic*.
- A static group is a group whose membership does not change during the system's lifetime. Members that crash are not excluded from the group.
- Dynamic groups are groups whose membership changes over time, as the result of the crash of a member, or of a new member (re-)joining the group.
- Dynamic groups are more powerful than static ones, but they are also more complex to implement.

Groups

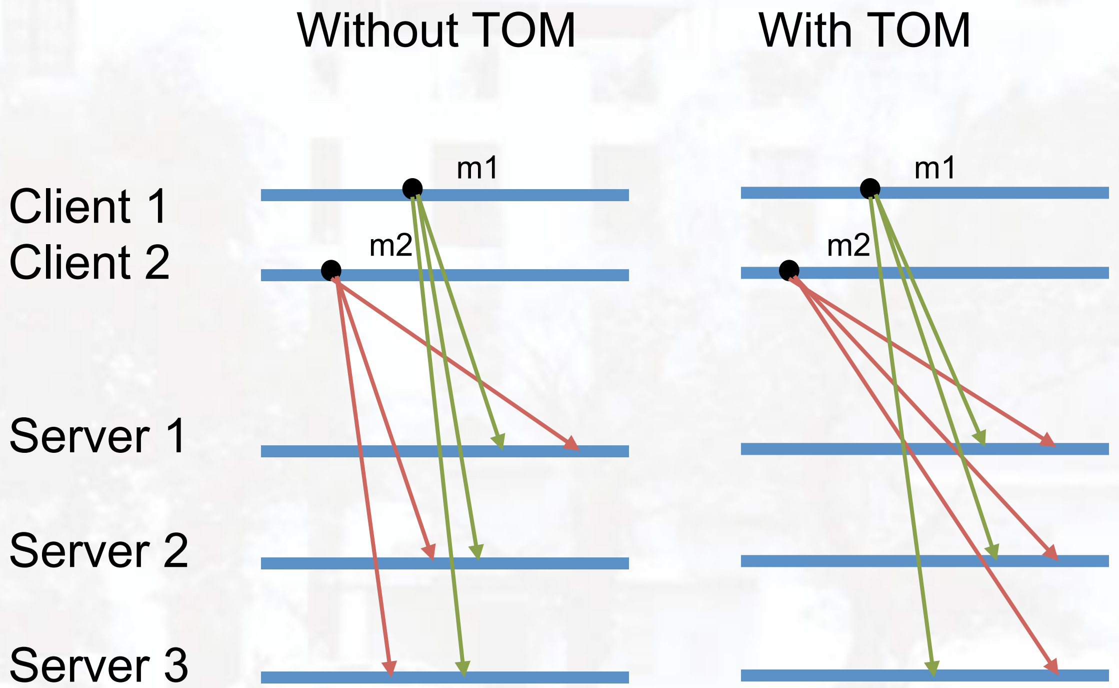
- Groups may be modeled as closed structures, where only members can issue multicasts to their group, or as open structures, where non-members can issue multicasts as well.



Total order multicast

- This is one of the most useful primitives for group communication. Simply stated, it ensures that messages sent to a group are delivered in the *same* order to *all* members of the group. Total ordering of messages is required for instance in replication, to ensure that the replicated data is kept consistent.

Total Order Multicast

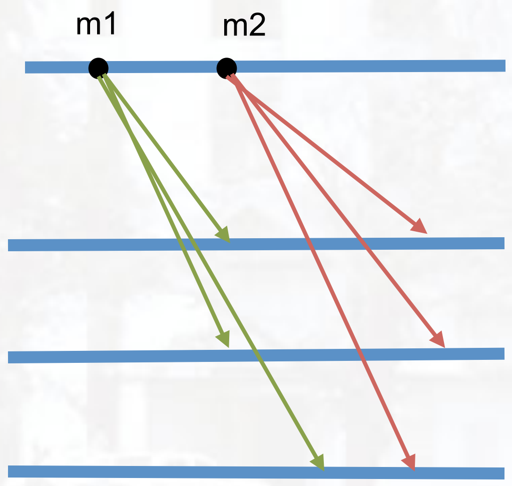


Synchrony

- Virtual: an abstraction of a set of processes or objects so group members see the same events in the same order (logical time) and asynchronous in physical time. All execute the same algorithm and have same consistent states, assuming their behavior is deterministic

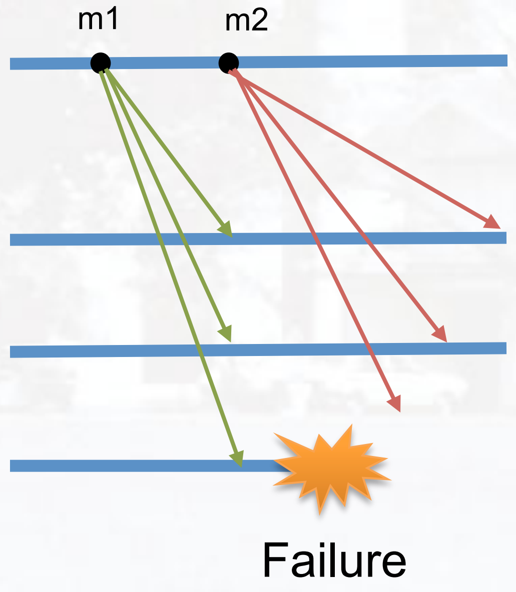
Virtual Synchrony

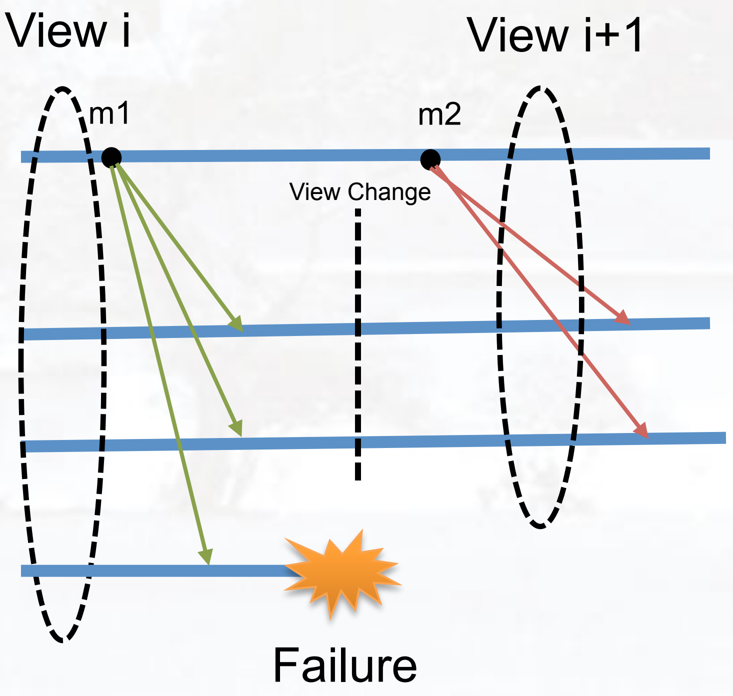
Done using Total ordering of messages



Synchrony

- View: Total order is guaranteed based on view changes

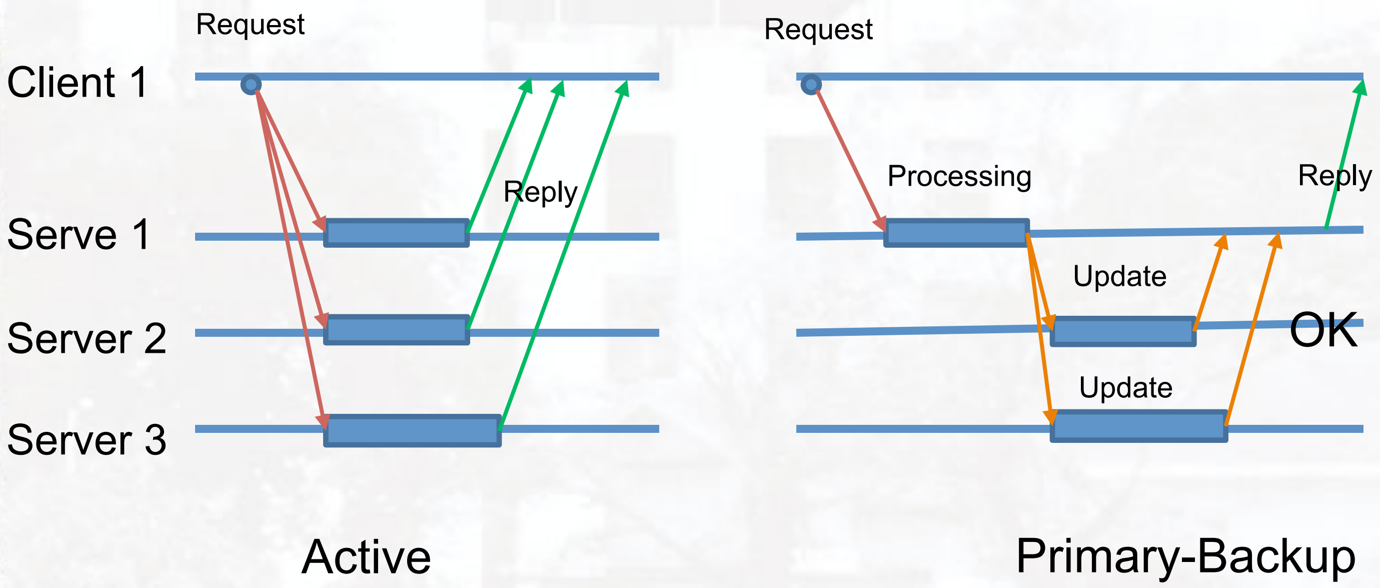




Replication

- ***Replication*** or the use of ***redundancy*** is a way of masking failure of individual components
- ***Active replication***: All copies of the object plays the same role (Receive -> Process -> Update -> Respond)
- ***Primary-Backup replication***: Primary object receives the input -> Process -> Updates self & others others -> Respond

Replication



Activity

- Suggest solutions to “How the Group would know if the Primary Server has failed” and “How would another (Backup) server know to take up the role of the new Primary or the client know which is the new Primary server”