# CSE 486/586 Distributed Systems Project 3

# Replicated Key-Value Storage


**Team Members**

**Maithreyi Buddhiraju**
**Vineet Aguiar**
**Shishir Garg**
**Ashwin R S**
**Shashank Raj Holavanalli**

Replica1 and Replica2 of a node are nothing but its Successor1 and Successor2.

**INSERT / QUERY**

**insert_or_query (coordinator, replica1, replica2)**

1. Send request to coordinator
2. Coordinator forwards request to replica1 and waits for response. If reply from replica1 is a success then increase vote count else do nothing.
3. Coordinator forwards request to replica2 and waits for response. If reply from replica2 is a success then increase vote count else do nothing.
4. Coordinator votes and increases vote count by 1.
5. Coordinator checks vote count >= 2. If true then reply to client with success message else send failure message to client.

**Step1**:
Client calls insert_or_query (coordinator, replica1, replica2).

**Step2**:
If timeout at step1 client learns that coordinator has failed so client calls insert_or_query (replica1, coordinator, replica2).

**Step3**:
If timeout at step2 client learns that replica1 has failed so client calls insert_or_query (replica2, coordinator, replica1).

**Step4:**
If timeout at step3 then perform steps 1 to 3 once again and then give up.

**UPDATE**

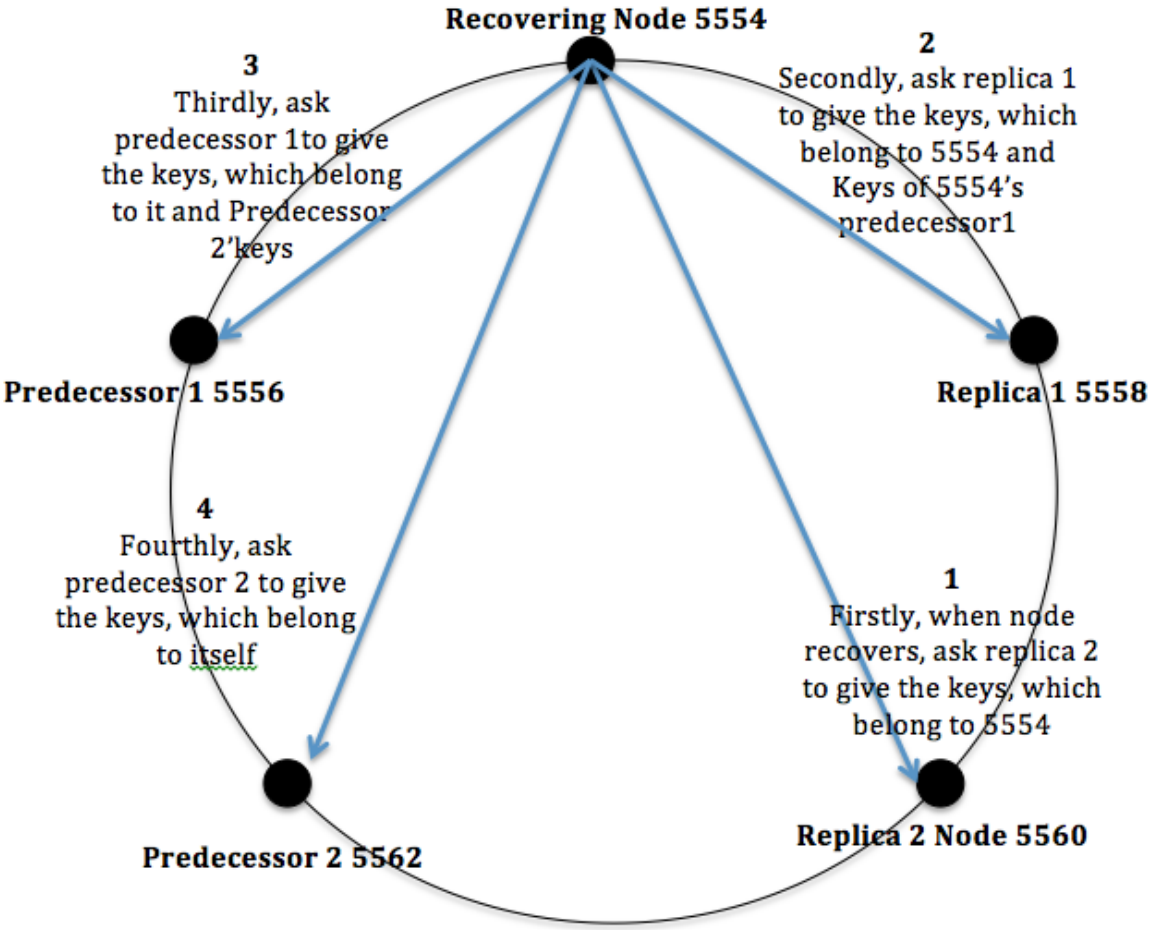
When a node is recovering from a failure it does the following things and we enforce this order for managing recovery under failure of some nodes. For example if predecessor1 has failed then its key-value pairs can be obtained from replica1. Under no failures there is performance degradation but we don't care about this as our model promises strong consistency.

**Step1:**
Get key-value pairs of itself from replica2.

**Step2:**
Get key-value pairs of itself and predecessor1 from replica1.

**Step3:**
Get key-value pairs of predecessor1 and predecessor2 from predecessor1.

**Step4:**
Get key-value pairs of predecessor2 from predecessor2.




**CONSISTENCY SCHEME -**

Our system promises sequential consistency because the events of a particular client are FIFO ordered. Linearizability is not promised since we are not implementing a time synchronization algorithm among the emulators. Our system promises strong consistency since our replication model is synchronous and requests from more than one client are always serialized. Availability is also high since local reads are successful and return consistent data at every replica. Each node sends replies on the same socket it receives a connection on. With this we enforce the flow of the program to wait at that point, after which we check for a timeout.

NOTE: Please give at least 1 minute after each PUT or GET press to see the results. And in some cases if you get popup for wait and force close, Please press wait button.