- ## Setup
    - System starts up by agreeing on how the data should be partitioned given the defined view. To do this the node with the ip address that appears first in the environment variable acts as the setup coordinator and makes unilateral assignments, and all other nodes contact it as they start. Once all nodes have contacted the startup coordinator and received the generated partitions the system can begin processing requests.
- ## Consistent Key Hashing
    - ### Devising custom hash
        - To ensure good distribution of keys and values in our system, a solid hash function was needed in order to expand a broad range of values in the hash space
        - We utilized the md5 library to generate a hash which we would later mod by the hash space. The hash value tells us where a node is in the hash space. We could also figure out where keys would go by computing its hash and seeing the node with the range that takes on that key's hash. Each node took on the range starting from it's own position all the way till the next node to the right. For nodes that are at the end of the array, we loop over to set the first node as the end range that encompasses the keys.
    - ### Performing binary search
        - In order to figure out where a key should go based on its hash value, we created a binary search function to figure out the host to send the key to. In addition binary search is used during the resharding process that's discussed in further detail below
    - ### Tokens Array
        - Each node had a tokens array that contained all the tokens of all the nodes in the distributed system. Such an array is needed as nodes need to know where to forward a request when the key being processed does not belong to the node.
    - ### Gets, Sets, Deletes
        - All gets, sets, and delete requests compute the hash of the key that's being sent from a client. After computing the hash, the node then performs binary search to see whether the hash computed is within the range of one of its tokens. If so, we process the request as normal, but if the token found is one that does not belong to the node, we forward that request to the node that actually contains the key. Once such a request is received the node will intelligently process the request without going through the same checking process the original node went through.

- View Changes
    - During a view change the node that processes the view change request acts as the coordinator.
        - It compares the incoming view with its view to find which nodes left and which joined. It generates new tokens for the joined nodes and then uses the current partition state to determine which partitions were affected. It creates a change object to be sent to each node saying which partitions need to be recomputed.
    - The new view and necessary changes are then propagated to the other nodes.
        - The coordinator first sends the updated view and partitions to each node. If the node is newly joining the system it also sends its partitions so that the new node can create entries for them.
        - After every node has acknowledged the view change the coordinator then sends the required changes to the nodes with affected partitions.
        - The nodes that receive changes then recompute the keys in the affected partitions and push them to the new nodes as necessary.
    - Once all keys have been pushed and changes acknowledged the coordinator gets updated key counts from all nodes and returns them to the client ending the view change.