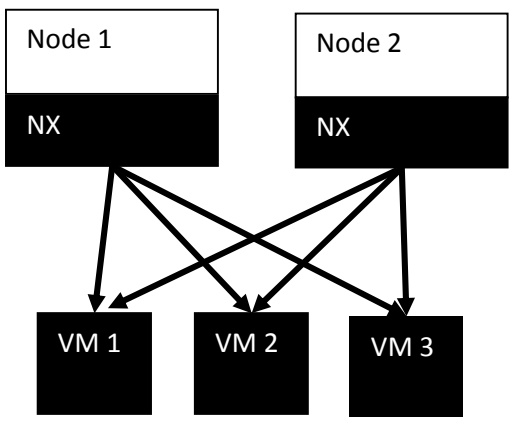
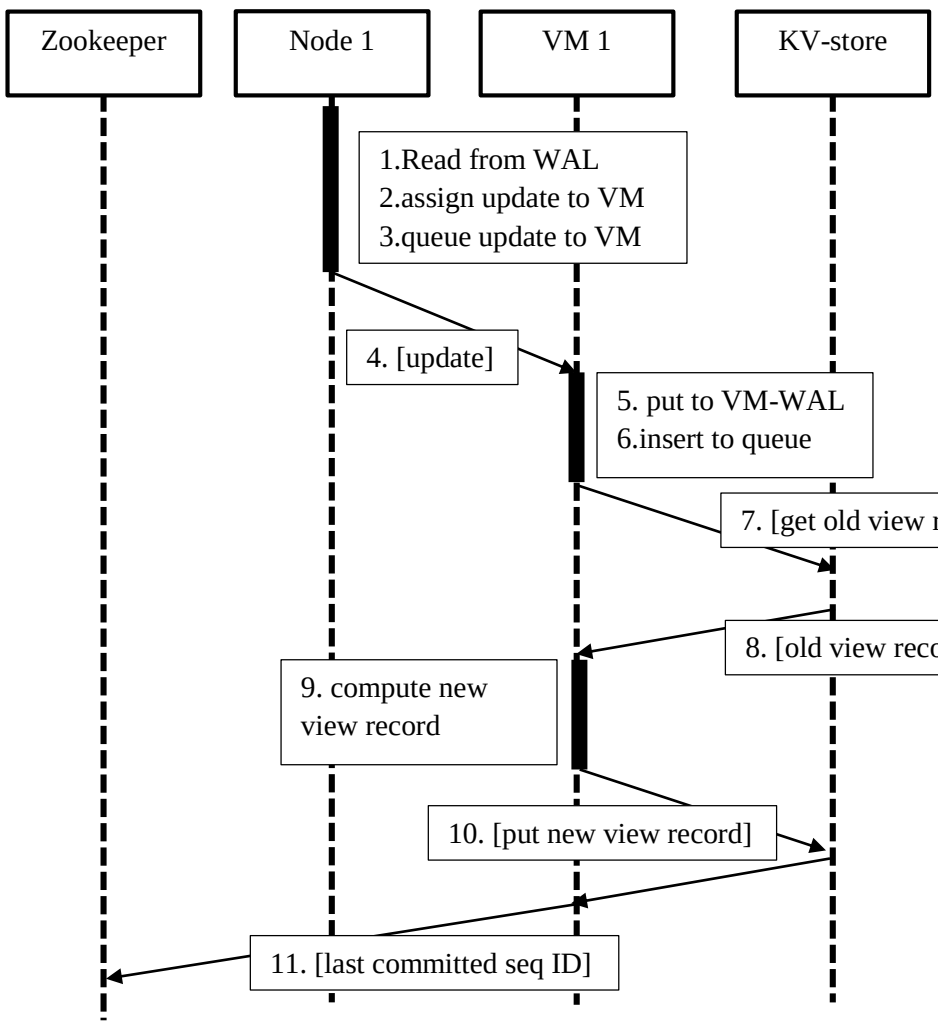
Local Hashring

Process update

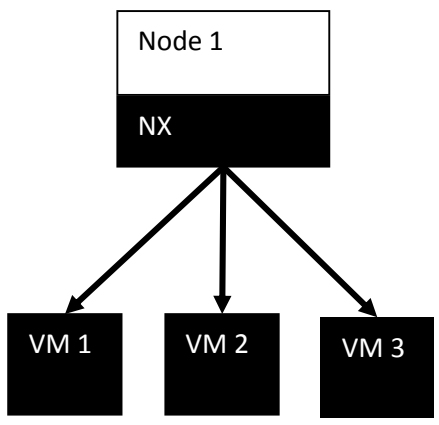


Description:
Updates are sent through a different communication channel than messages. The update stream is continuously flowing from the database nodes to the view managers. One update passes the following stages.

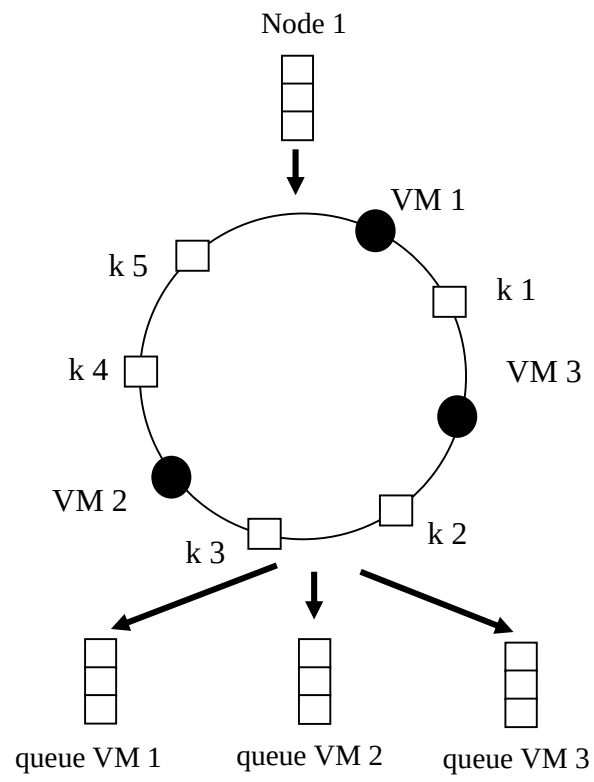
Algorithm:
All Nodes:[create znode] → Zookeeper
Zookeeper:[election finished] → all nodes



Send marker



Description:
To perform actions like the assign and withdraw view managers in a dynamic environment, the Node requires information about whether a view manager has already processed a set of updates or not. For that reason markers are placed in the view manager's queue like a normal update. From here they are sent to the view manager. At some point the VM hits the marker in the stream of updates and sends back an acknowledgement to the Node. Thus, the Node knows that all updates have been processed.



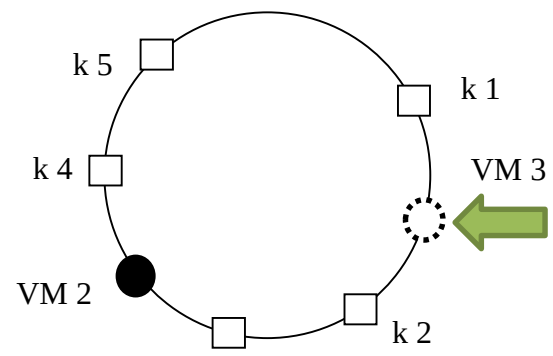
Algorithm:
All Nodes:[create znode] → Zookeeper
Zookeeper:[election finished] → all nodes

Local Hashring

Node operations

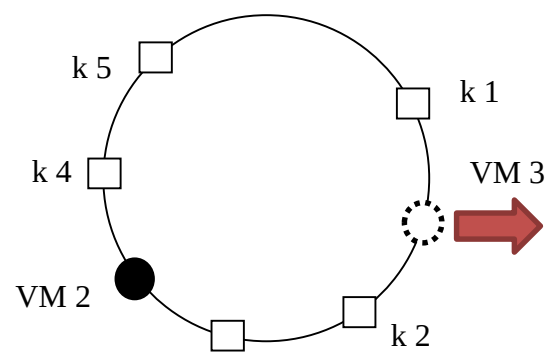
Insert hash:
A hash is computed from the view manager's name and inserted into the hash ring. The function also returns the view manager that releases a key-range

Signature:
insertHash(String vmName):vmReleasing



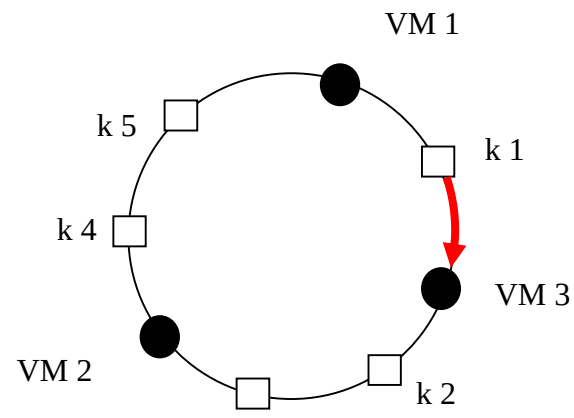
Remove hash:
A hash of a view manager is removed from the ring. The function also returns the view manager that takes the key-range of the removed VM.

Signature:
removeHash(String vmName):vmTaking



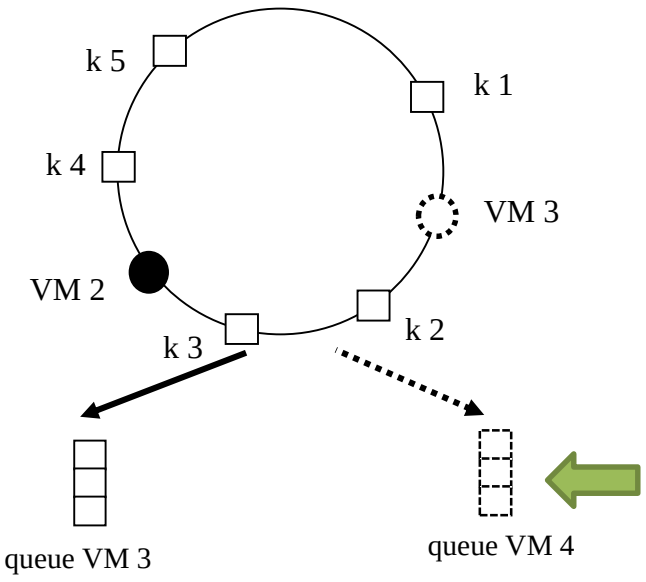
Assign update:
A hash is computed from the key of the update. Starting at the position of the hash-value, the function travels clockwise on the ring until it hits the next view manager. This VM is determined responsible for the update.

Signature:
computeVM(String key):vmResponsible
computeVMs(String startKey, endKey):List<VM>



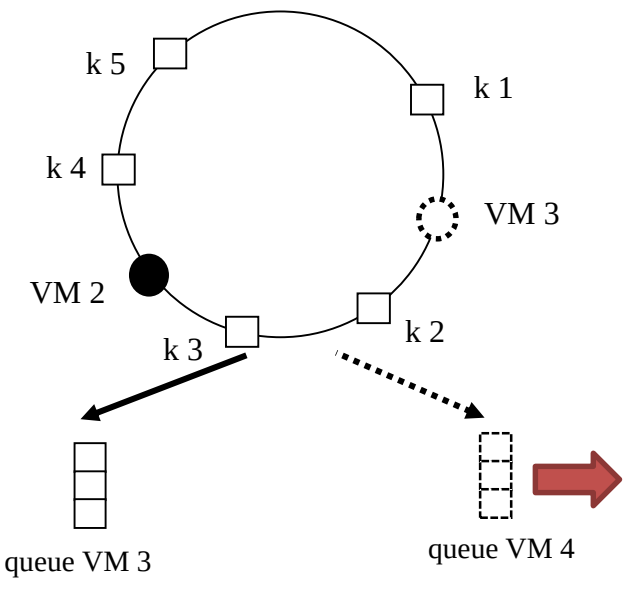
Create queue:
A queue is created for the view manager. All updates, for which the view manager is assigned responsible, are inserted into that queue.

Signature:
assignUpdate(String key):vmResponsible



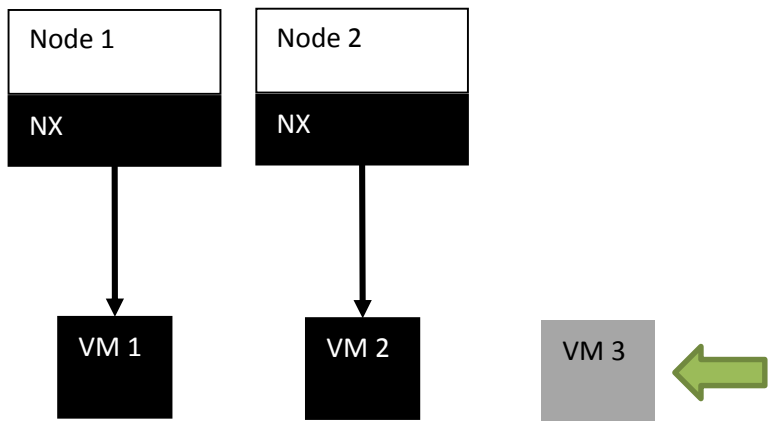
Remove queue:
The view managers queue is deactivated

Signature:
assignUpdate(String key):vmResponsible

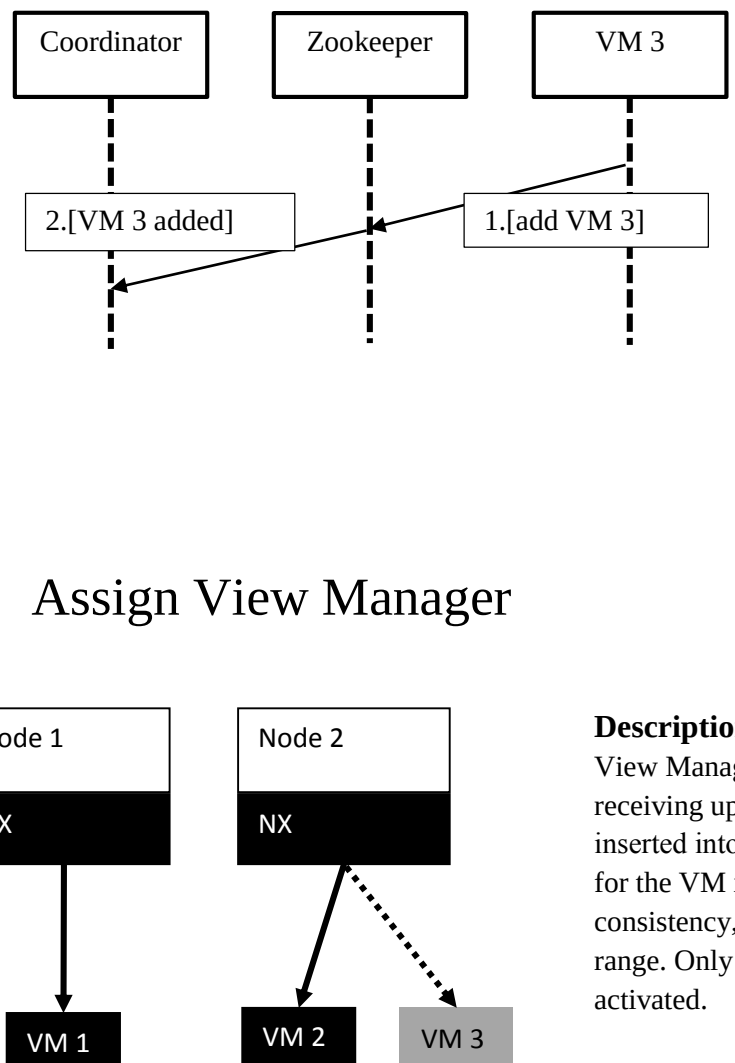


Local Hashring

Add View Manager



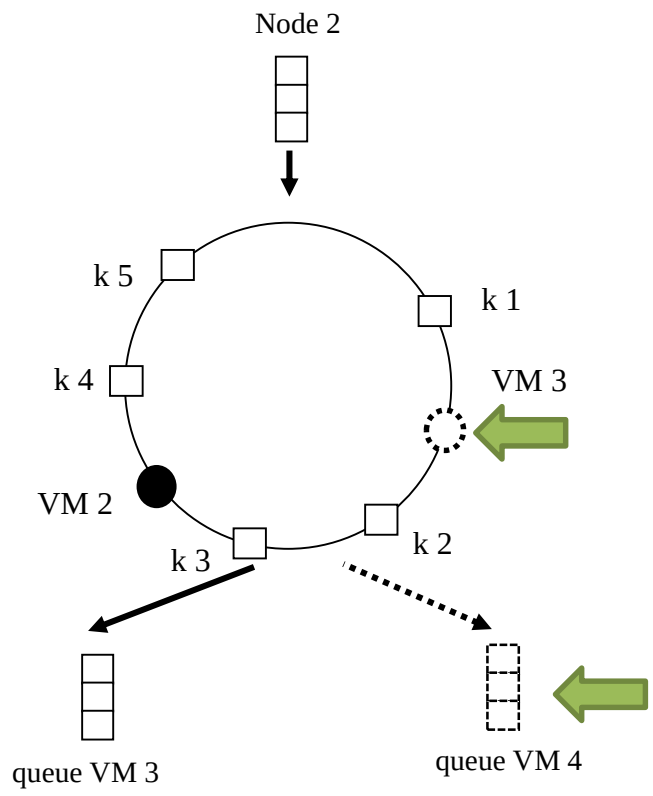
Description:
View Manager is started and registers to the VMS. After the procedure, VM is idle, waiting for further commands.

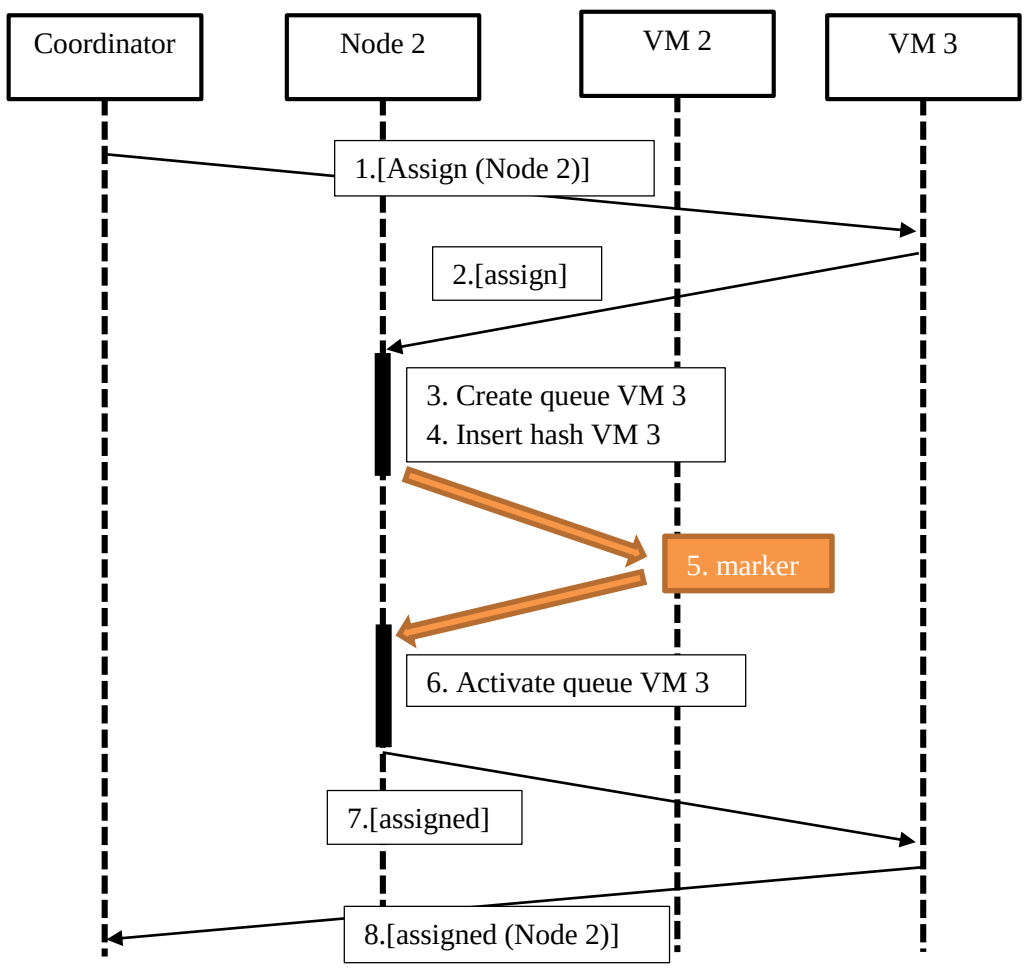


Algorithm:
1. VM: [Add VM] → Zookeeper
2. Zookeeper: [VM 3 added] → Coordinator

Assign View Manager

Description:
View Manager is assigned to a Node, i.e. it starts receiving updates from this Node. VM's hash is inserted into the Node's hash ring. An update queue for the VM is created at the Node. To preserve consistency, markers are send to VMs that lose a key-range. Only then, the update queue of the VM is activated.

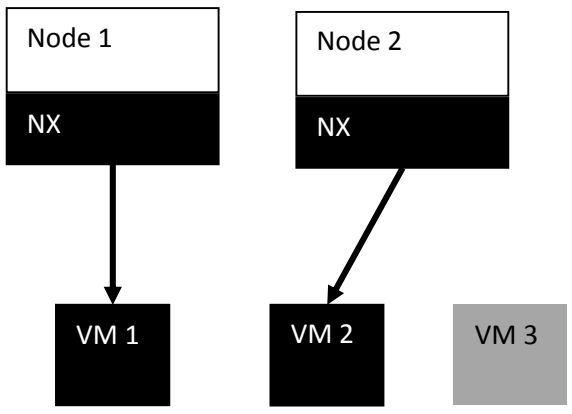




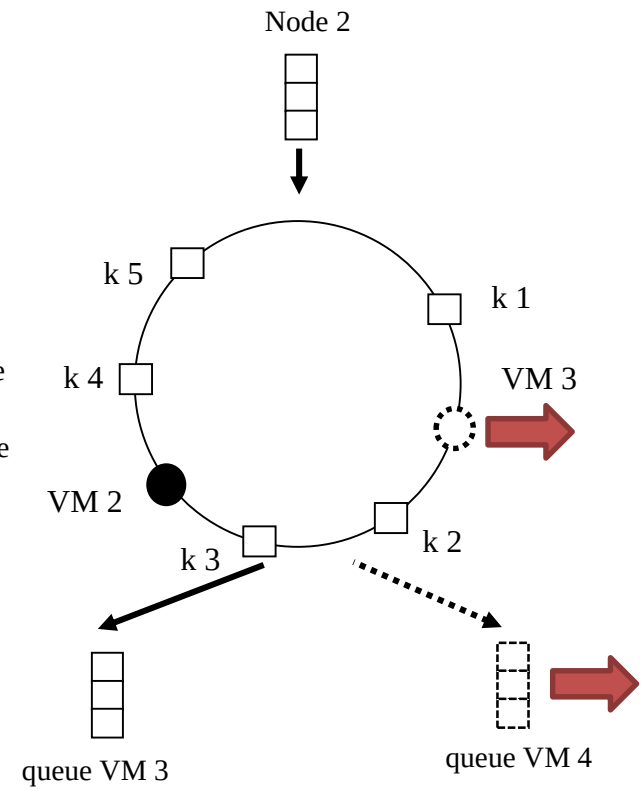
Algorithm:
1. Coordinator: Send [assign to Node] to VM
2. VM: Send [assign] to Node
3. Node: Add VM to hash ring
4. Node: Create queue for VM
5. Node: Send [marker] to all VMs that lose a key-range
6. VMs: Send [ack]
7. Node: Send [assigned] to VM
8. VM: Send [assigned to Node] to coordinator

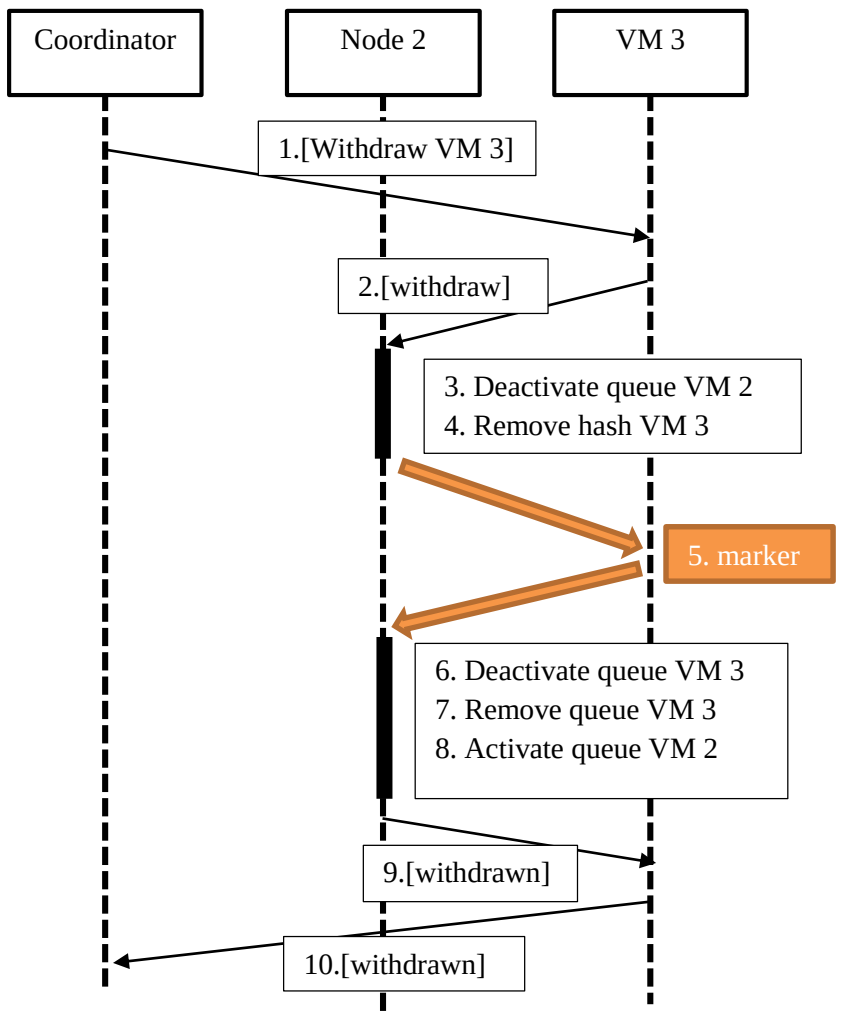
Local Hashring

Withdraw View Manager



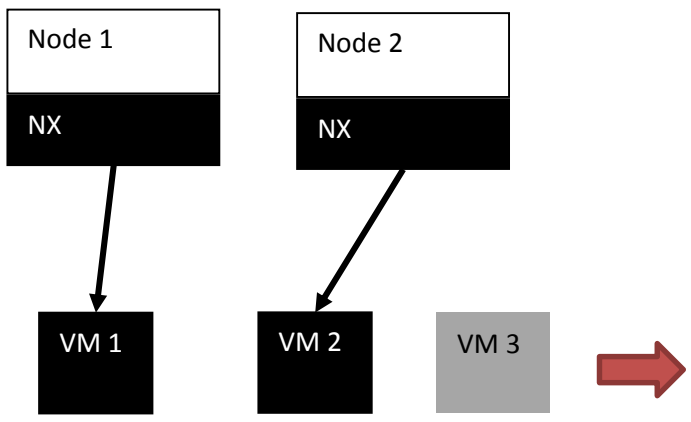
Description:
View Manager is withdrawn from a Node, i.e. it stops receiving updates from this Node. VM's hash is removed from the Node's hash ring. To preserve consistency, the queues of VMs that obtain a key-range are deactivated. Since the withdrawn VM loses a key-range a marker is sent to it. Only then, the update queue of the VM is deactivated and removed.



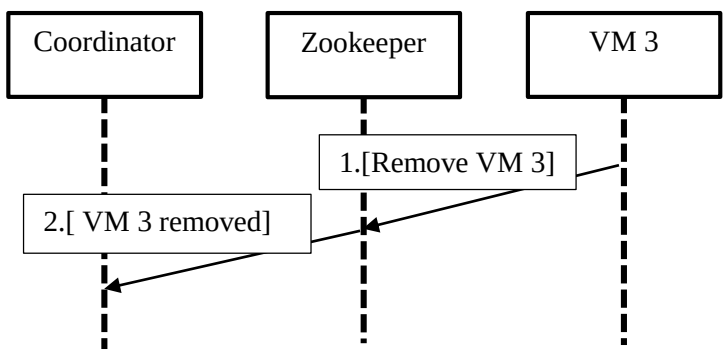


Algorithm:
1. Coordinator: Send [withdraw from Node] → VM
2. VM: Send [withdraw] → Node
3. Node: Deactivate queues of VMs that obtain key-range
4. Node: Remove hash from VM
5. Node: Send [marker] → VM
6. VMs: Send [ack] → Node
7. Node: Remove queue of VM
8. Node: Activate queues of VMs that obtain key-range
9. Node: Send [withdrawn] → VM
10. VM: Send[withdrawn] → Coordinator

Remove View Manager

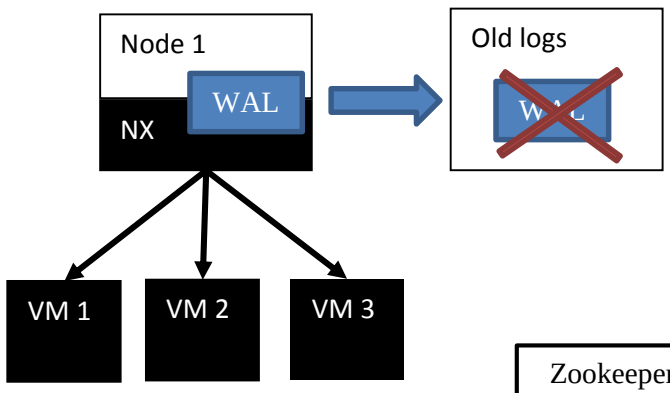


Description:
View Manager is deregistered from the VMS. After the procedure, VM is not available any more.

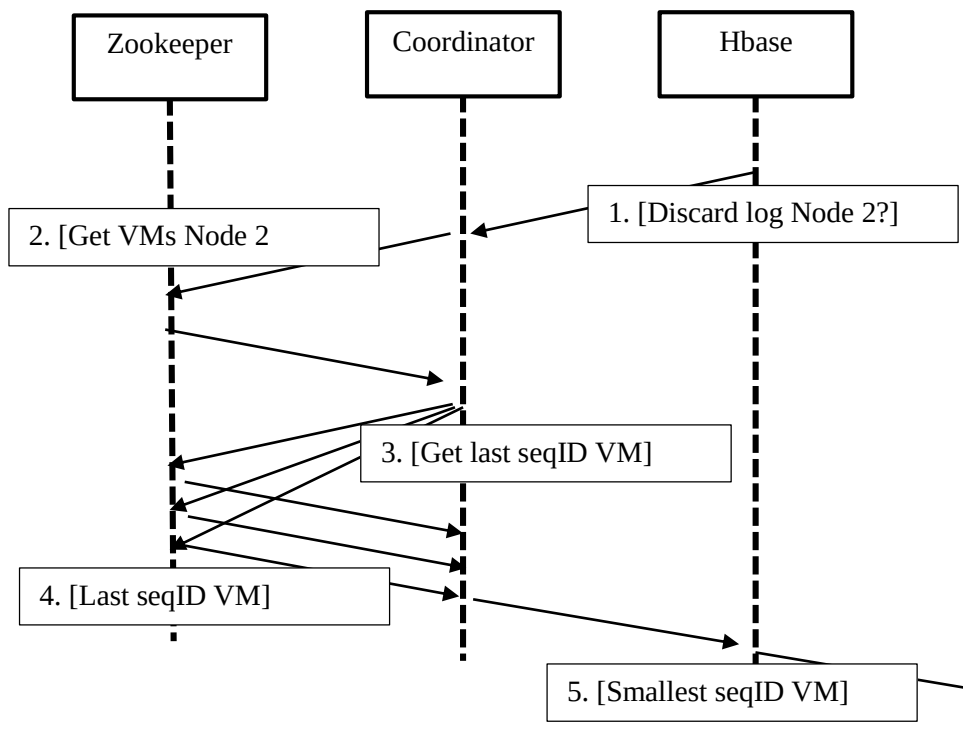


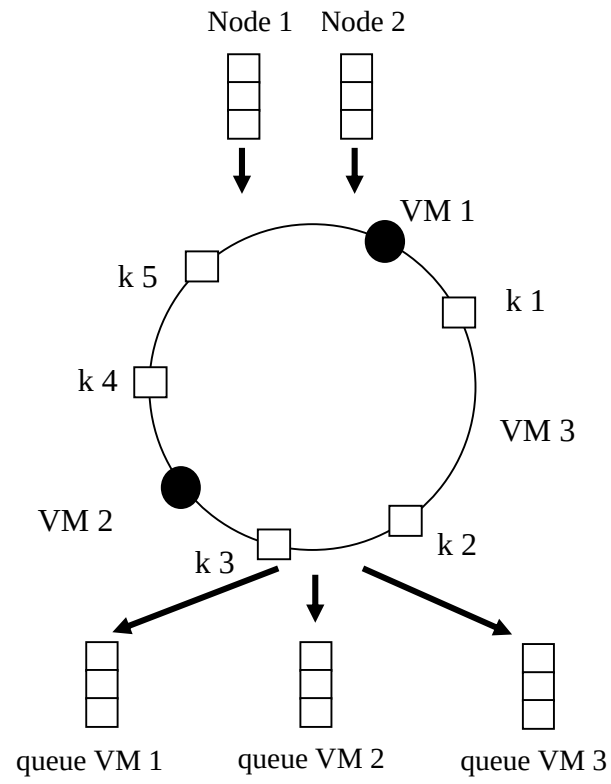
Algorithm:
1. VM: [Remove VM] → Zookeeper
2. Zookeeper: [VM 3 removed] → Coordinator

Discard logs

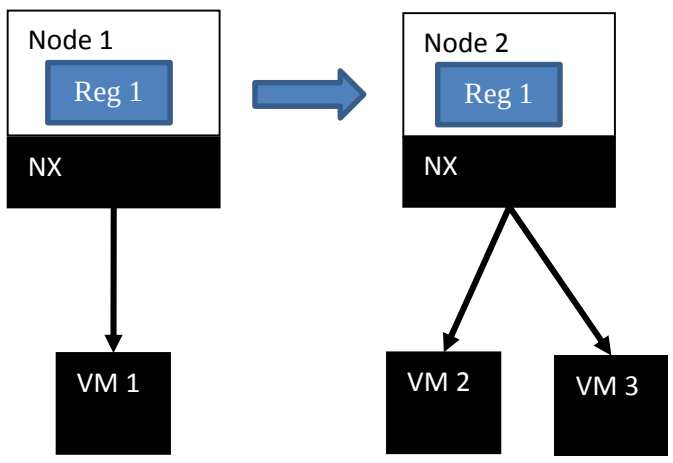


Description:
Physically, HBase writes to multiple WAL files in HDFS. As all entries of a log file are flushed to disk, the KV-store discards the log. The VMS has to assure that all entries are propagated and applied by a view manager. Otherwise, some of the entries will be lost.

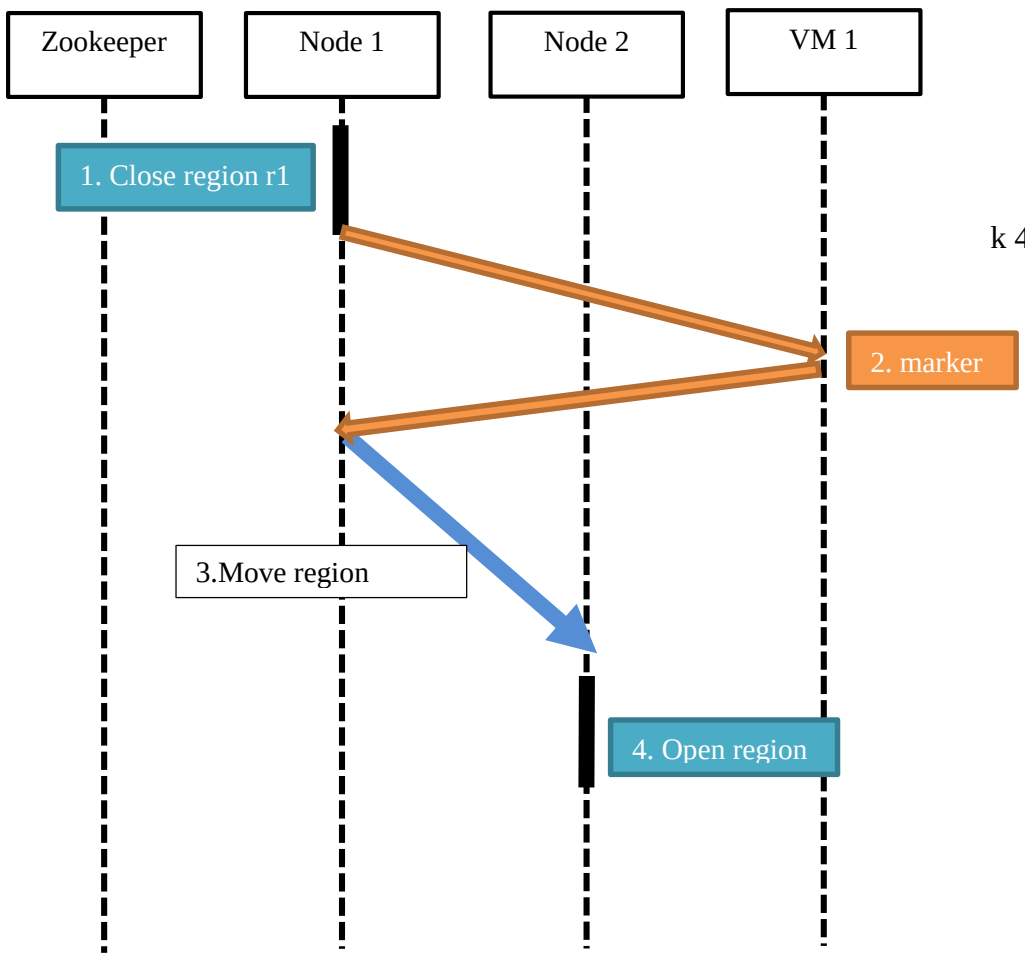


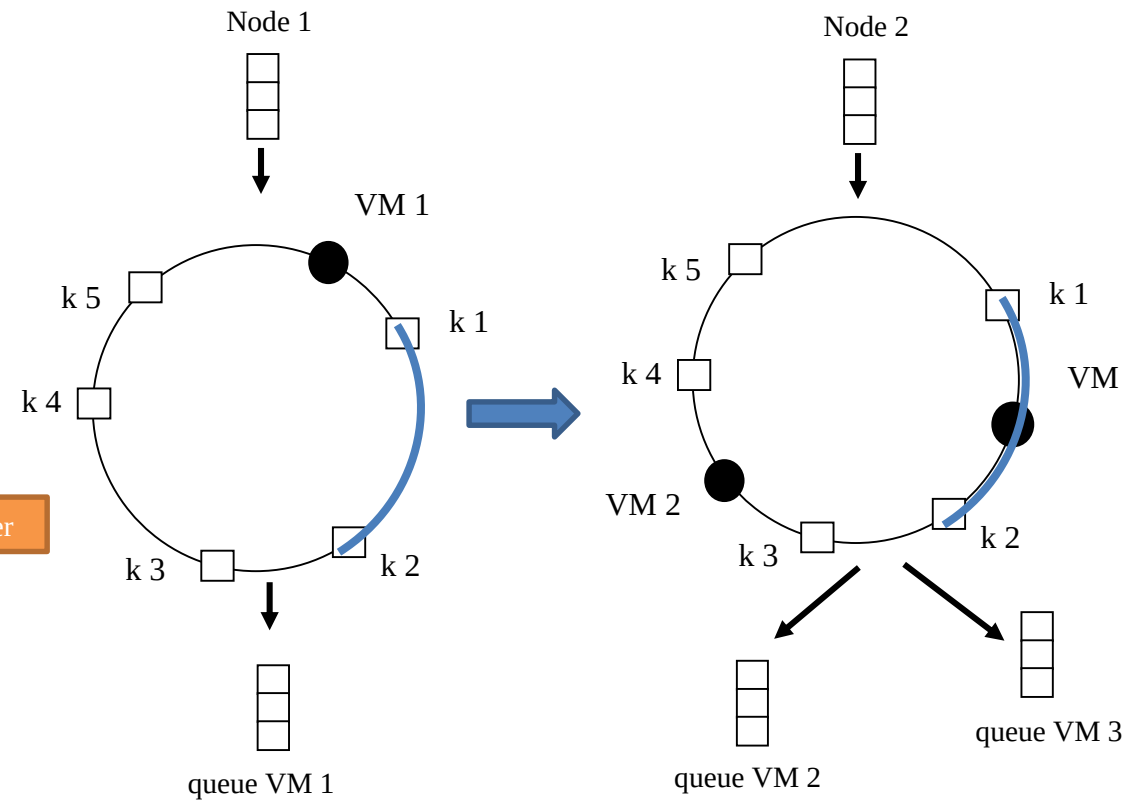


Move Region



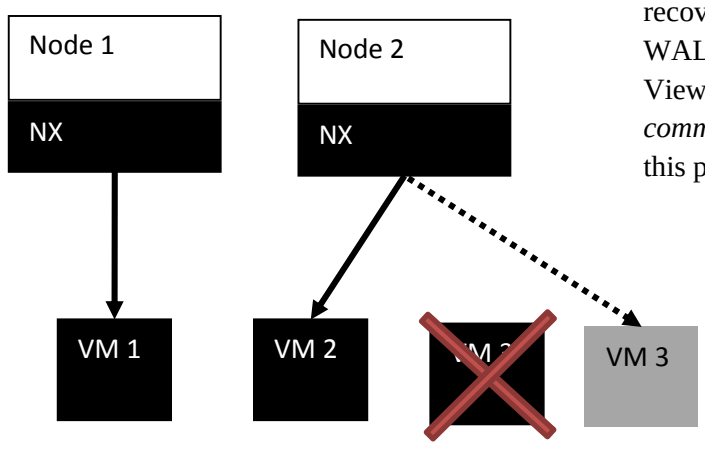
Description:
In order to load balance the system the KV-store keeps moving key-regions from one Node to another. This can cause concurrent update processing considering the two involved nodes. The VMS needs to preserve time line of records, that is, it needs to ensure updates on the closing region's node have to be finished before processing updates on the opening region's node.



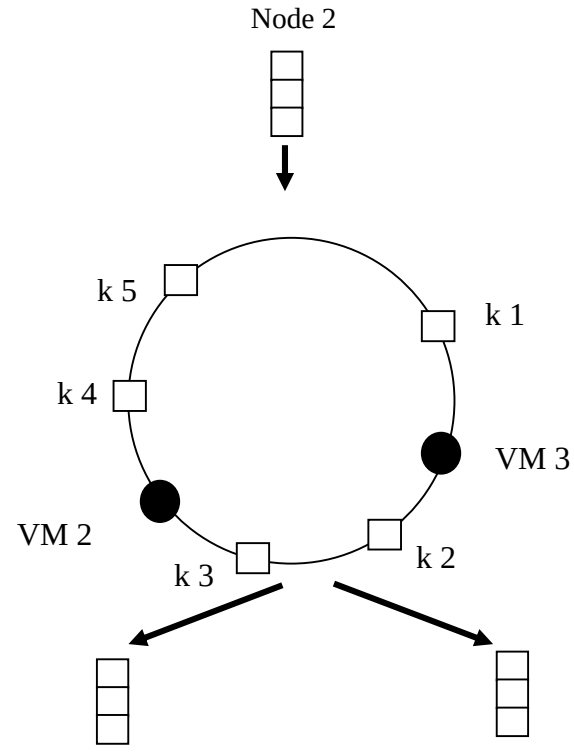


Local Hashring

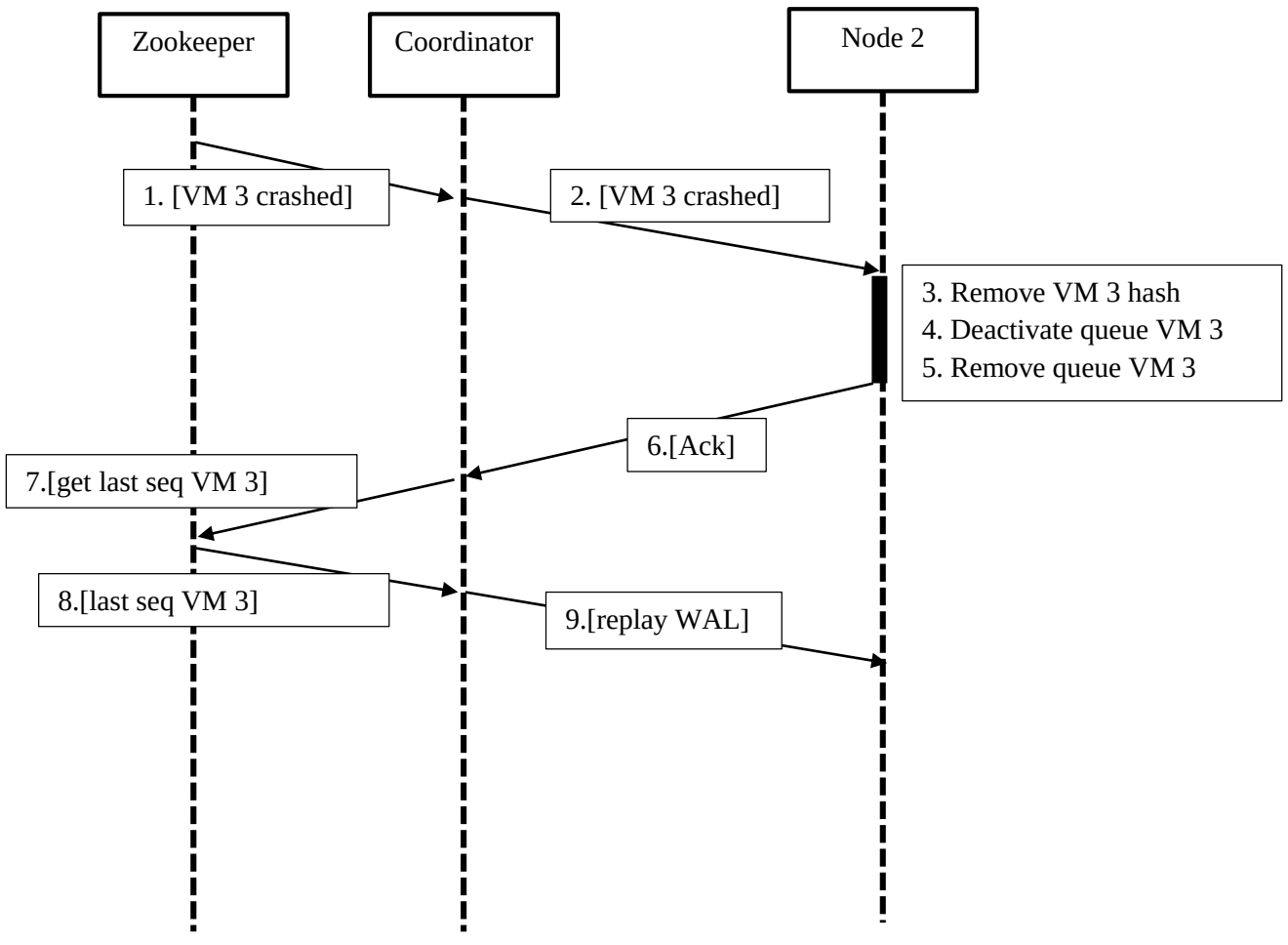
Crash View Manager



Description:
In case a View Manager crashes, two alternatives of recovery can be distinguished: The Nodes replays its WAL **or** the VM replays its WAL. In the first case the View Manager needs only to keep track of its *last committed* sequence ID. The WAL is replayed from this point



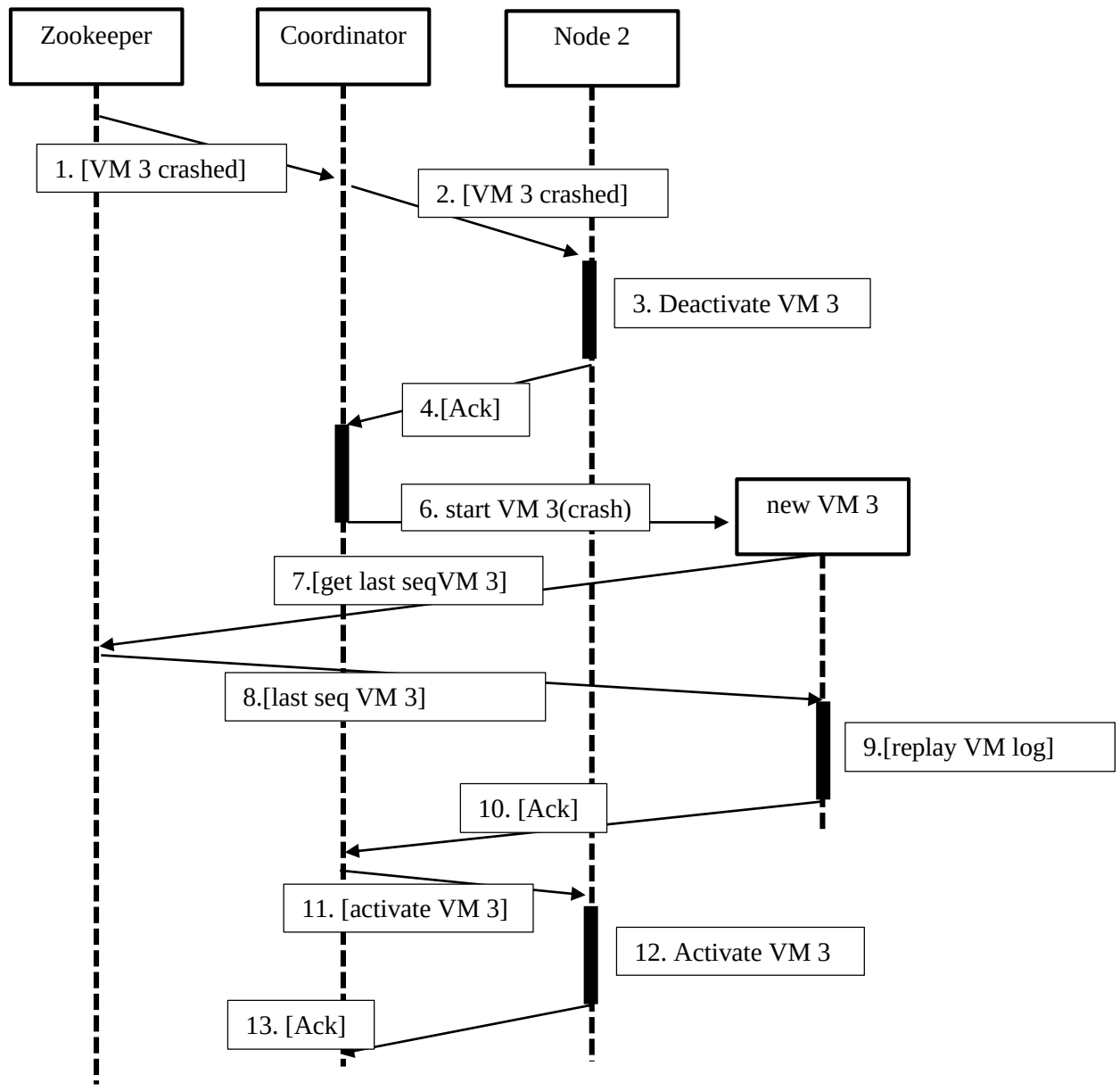
Alternative 1 (replay Node WAL, requires signatures):



Algorithm:

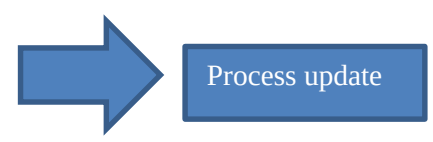
1. Zookeeper:[VM crashed] → Coordinator
2. Coordinator: Send [deactivate queue of VM] → Node
3. Node: Deactivate queue of crashed VM
4. Deactivate queue VM
5. Remove queue VM
6. Node: Send [ack] → Coordinator
7. Coordinator: get last committed seq ID of VM → Zookeeper
8. Zookeeper: Send [last seq ID] → Coordinator
9. Coordinator: [replay log, seq ID] → Node

Alternative 2 (replay VM WAL):

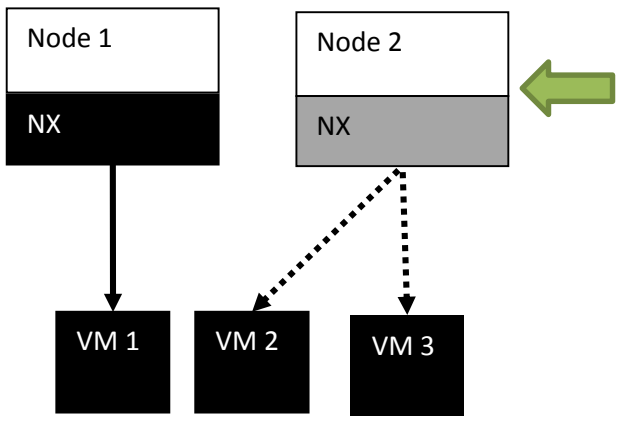


Algorithm:

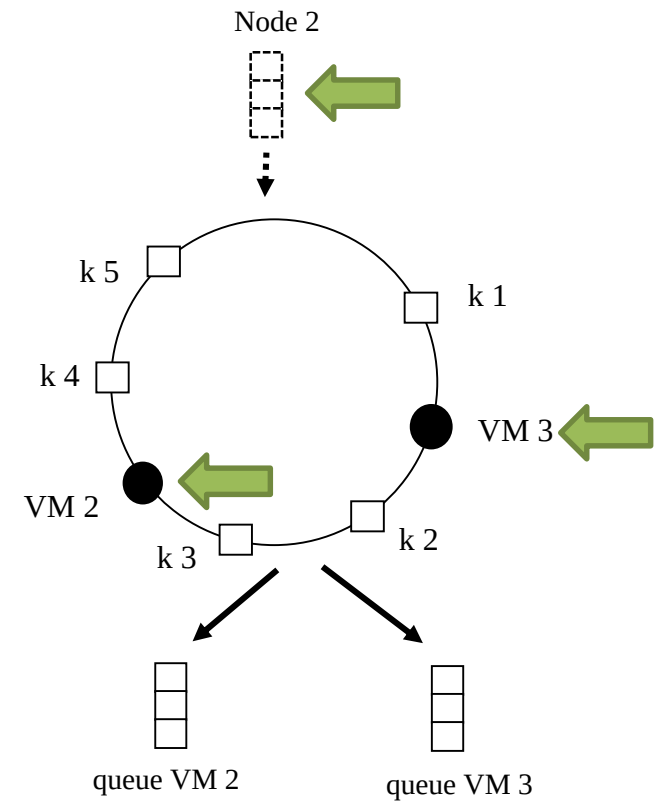
1. Zookeeper:[VM crashed] → Coordinator
2. Coordinator: Send [VM crashed] → Node
3. Node: Deactivate queue of crashed VM
4. Node: Send [ack] → Coordinator
5. Coordinator: start VM (on a different server)
6. New VM: Send [add VM] → Zookeeper
7. New VM: Retrieve last committed seq ID → Zookeeper
8. New VM: replay log of crashed VM up to the last committed seq ID
9. New VM: Replay VM log
10. New VM: send [ack] → Coordinator
11. Coordinator: Send [activate new VM] → new VM
12. Node: Activate new VM
13. Node: Send [Ack] → Coordinator

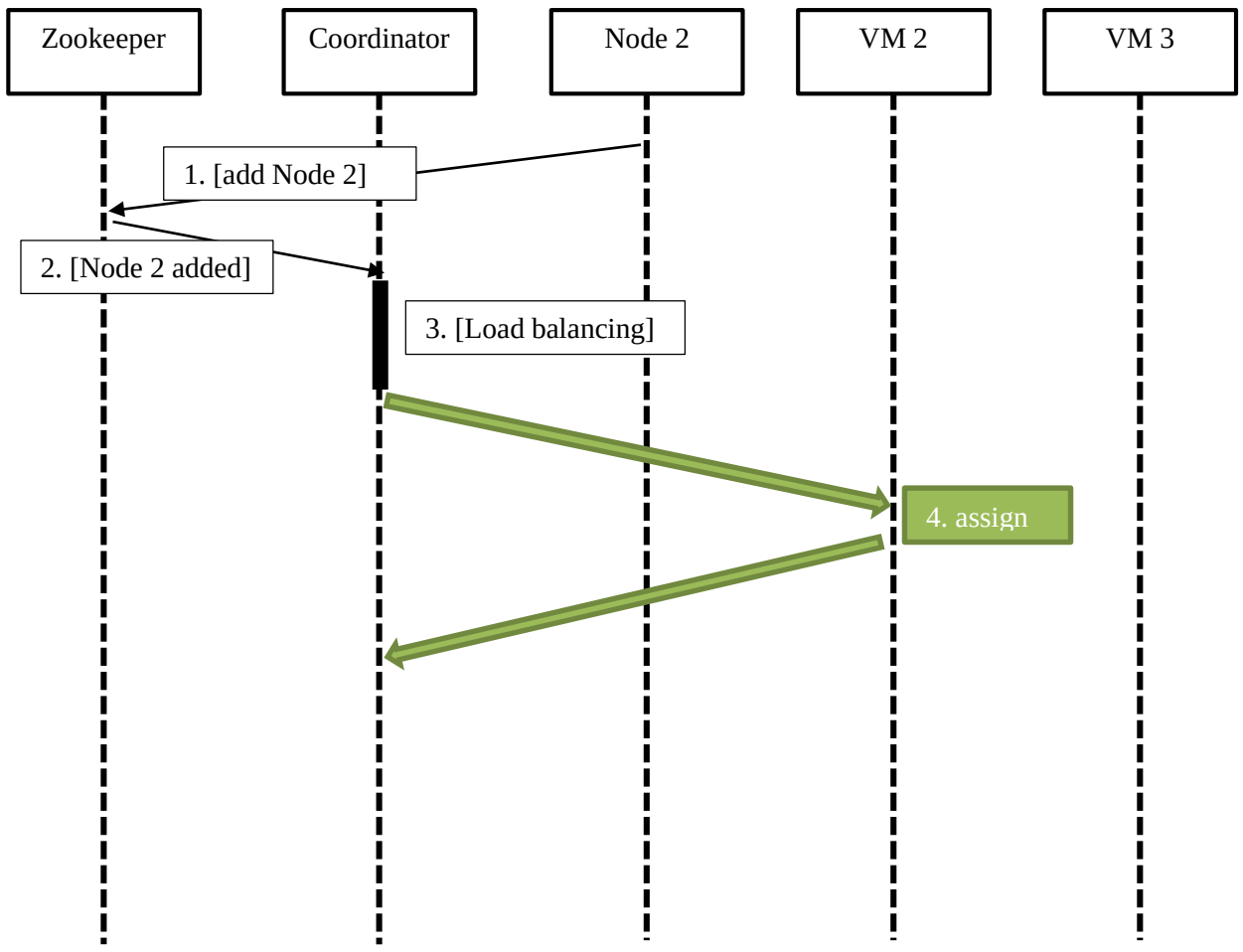


Add Node



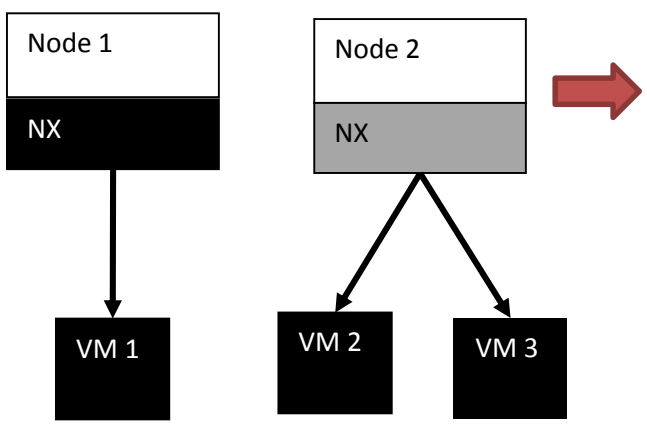
Description:
In the KV-store architecture new nodes can be added to scale up the system. The VMS needs to react and assign at least one View Manager to the new Node. Else, updates cannot be processed.



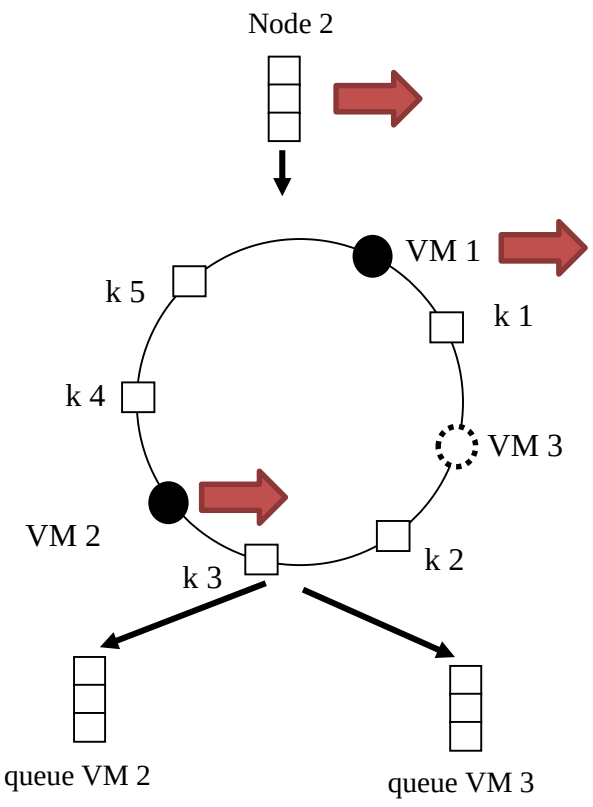


Algorithm:
1. Node:[add Node] → Zookeeper
2. Zookeeper:[Node added] → Coordinator
3. Coordinator: run load balancing algorithm
4. Assign VMs to new Node

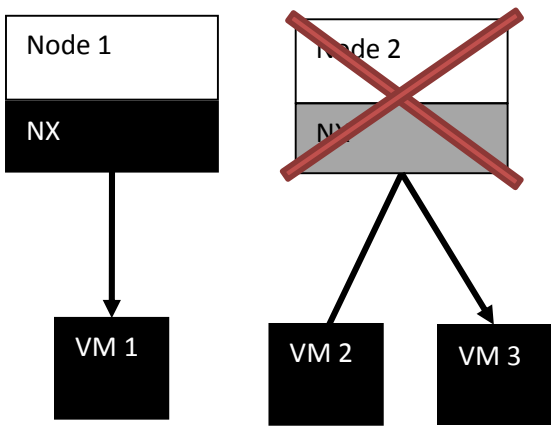
Remove Node



Description:
In the KV-store architecture nodes can be removed from the system. The VMS needs to react and withdraw all View Managers from the Node until the hash ring is empty. Then, the znode in Zookeeper is destroyed.



Crash Node (Move region + discard logs)



Description:
If a node crashes in the KV-Store architecture, the abandoned WAL is recovered and replayed. During the replay the log entries are grouped by key-region. The key-regions are assigned to the remaining nodes. Finally, the entries are inserted into the memstore and flushed to disk directly.