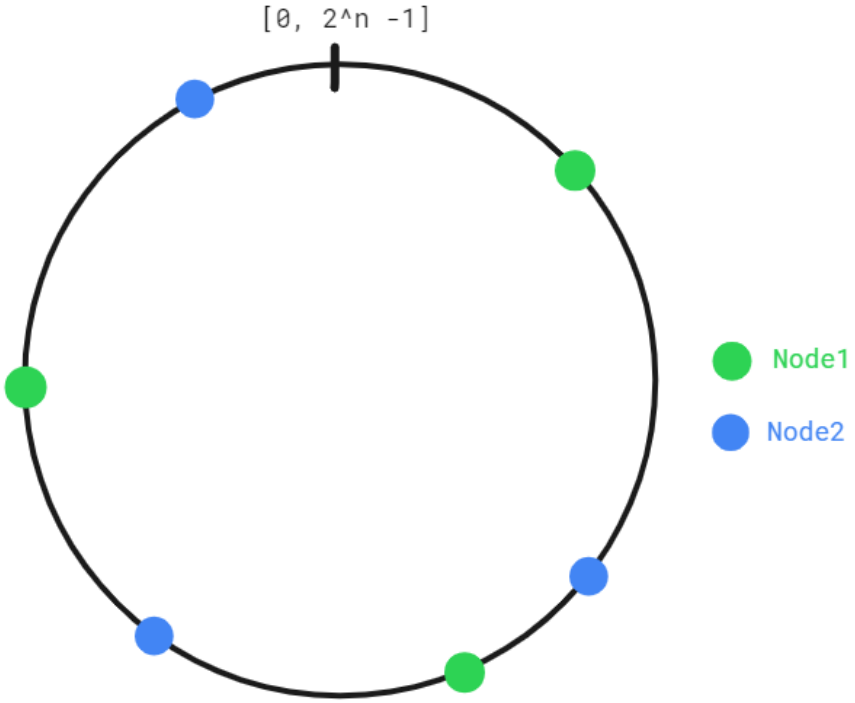
Author: Diego JIN

Consistent Hashing

Server Node

1. Treat Hash function as a cycle.
2. Each Server is assigned as **multiple Nodes uniformly**.

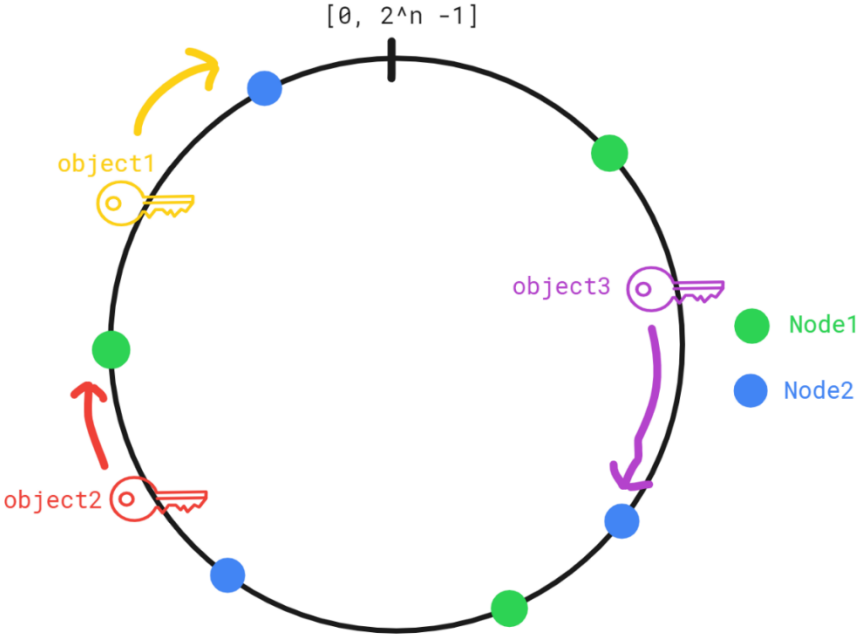


Object hashing

Compute object's **index by key** in circle, and find **closet clockwise Node** to store.

index = hash_function*(key)*

When search Object, pointer search from key point to find closet clockwise Node.

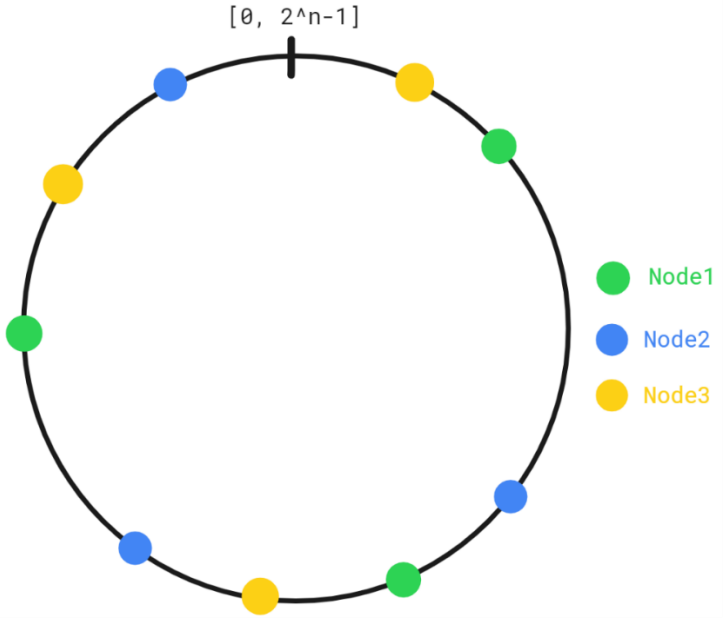


Author: Diego JIN

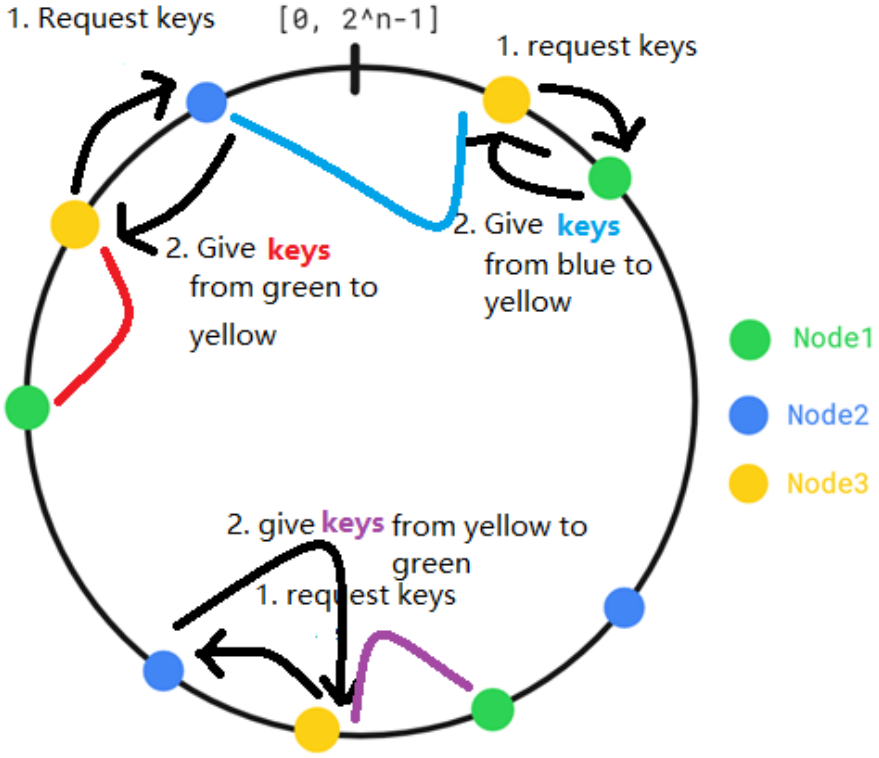
Author: Diego JIN

Add a Node

1. Assign multiple nodes to new server uniformly.



2. Each new Node require **next clockwise node** for the object keys from previous Node to new node.



Time complexity:

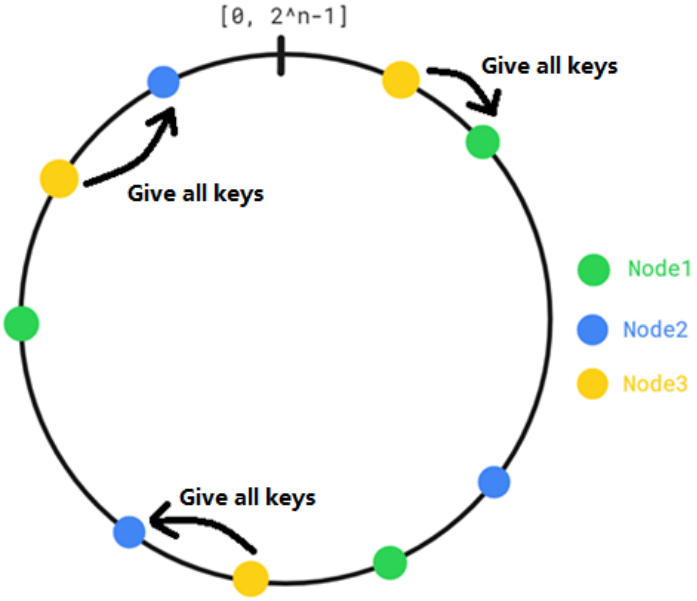
$O(\frac{K}{N} + \log(N))$, K is total k numbers; N is total Node numbers.

Note: log(N) for binary search for next Node. O(k/n) for moving keys.

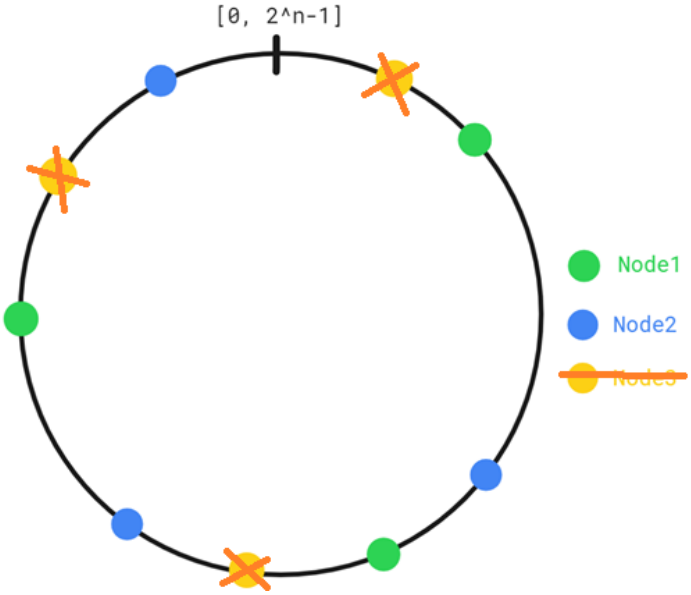
Author: Diego JIN

Remove a node

1. The removed node gives all keys to next clockwise node.



2. Remove the Node.



Note: log(N) for binary search for next Node. O(k/n) for moving keys.

Author: Diego JIN

Complexity and advantage

Advantage

1. No matter removes or add a Node, it ensures that keys are uniformly distributed.
2. When keys are in transferring, the traffic load is balanced in each server.

Complexity

Asymptotic time complexities for N nodes (or slots) and K keys.

Note: log(N) due to find closet Node via **binary search**.

	Consistent hashing
add a node	O(K/N + log(N))
remove a node	O(K/N + log(N))
add a key	O(log(N))
remove a key	O(log(N))