Dynamo DB - Implementation insight

pm_jat @ daiict

"Eventual Consistency"

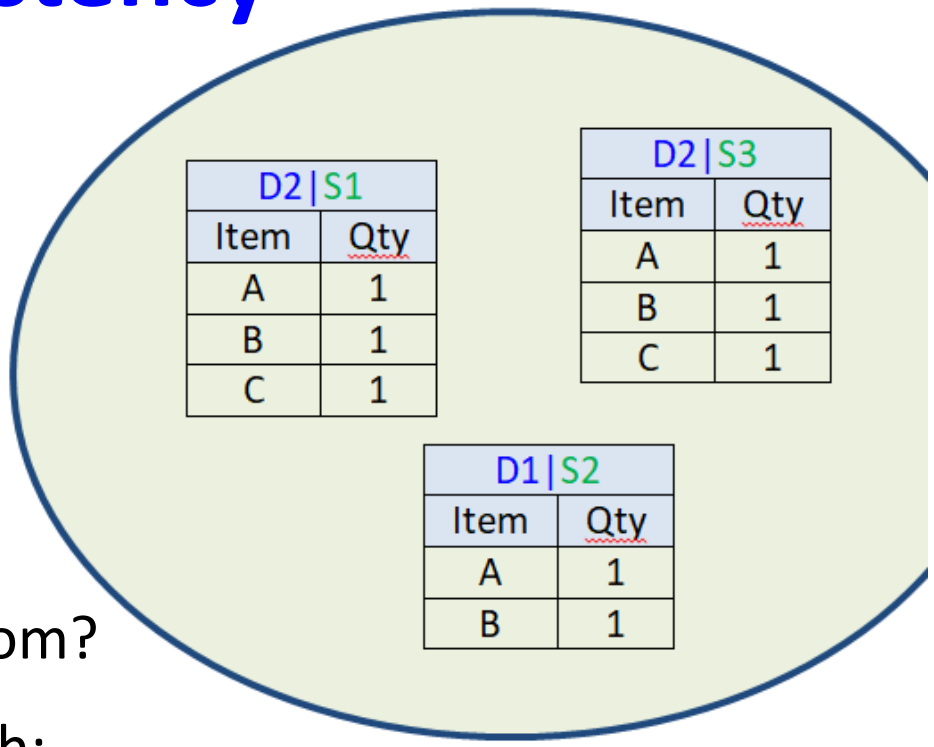
- Classical understanding of "consistency" does not fit into the context of data "Partitioned" and "replicated" environment.
- Therefore, let consistency be simplified and redefined that can be used in this context.
- **Strict Consistency**:
  - For a process, reads always reads latest data, that is, reads immediately preceding update.
- **Eventual Consistency**:
  - It is OK getting (reading) little older version till a time update reaches to all replicas. But eventually it will lead to replicas having same data.

“Eventual Consistency”

- What is classical understanding of consistency?
- User’s perspective:
  - No “Dirty Read”
  - No “Lost Updates”
  - Repeatable Reads
  - No Phantom Rows
- At the server end: all this is achieved by “Atomic” and “Serializable” execution of transactions, through so called ACID properties.

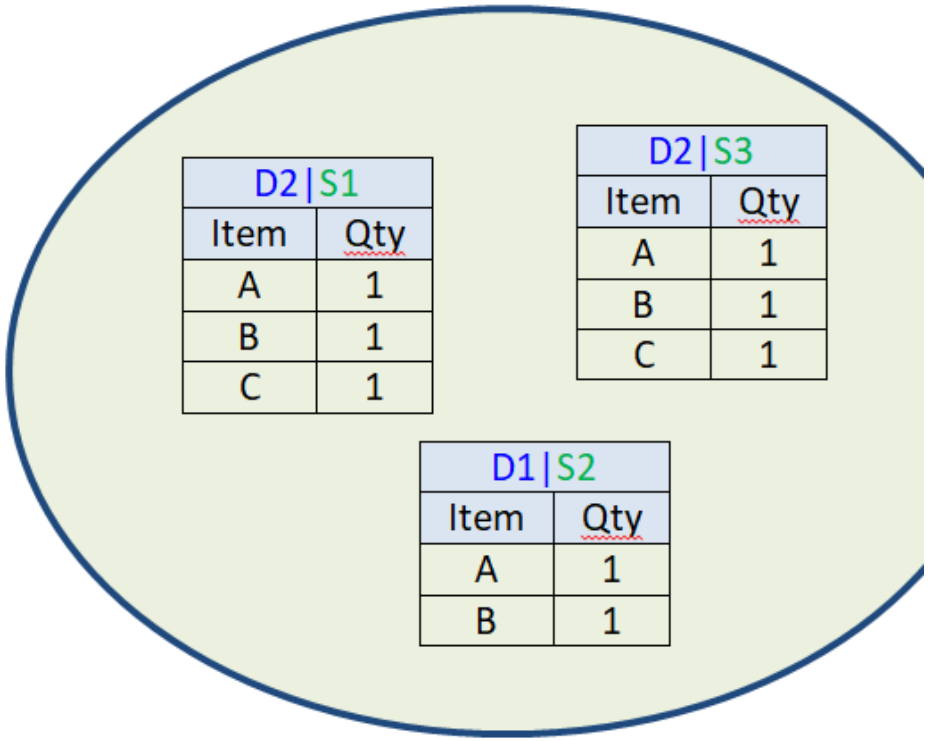
Eventual Consistency

- Suppose here is snapshot of a shopping cart in replicated database?
- What should be returned if we attempt reading the cart?
- Depends where do we read from?
- This is temporary stage, though; eventually all replicas shall be same and we will not have this problem!
- This what defines eventual consistency.



Eventual Consistency

- Eventual Consistency means "temporarily, reads will have problem", but that problem is till a time all replicas come to a sync.
- There are variation of Eventual Consistency that different No SQL systems may provide.
  - Read Your Write
  - Session level Read Your Write,
  - So forth



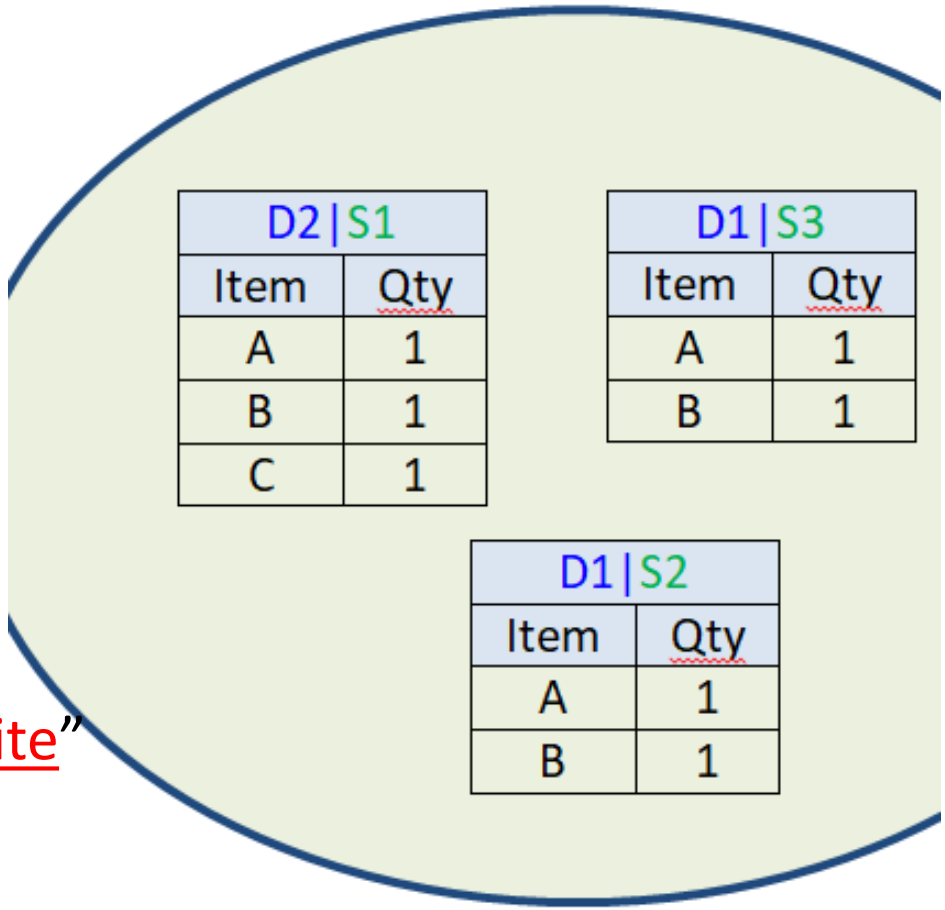
Eventual Consistency

- What is **Read Your Write**?

Eventual Consistency

What is **Read Your Write**?

- Suppose a process has just written data item D on replica S1 (ver D2)
- Now same process attempts reading D, and let us say Read is done from replica S2 or S3?
- This is "Does not Read Your Write"



Eventual Consistency

- Different No SQL systems have their way of providing consistency, but mostly it is some variation of "eventual consistency"!
- We plan to look into Dynamo DB and Mongo DB that
  - "what" consistency they provide, and
  - "how" do they implement them?

Dynamo DB – Design Considerations[1]

- As have already understood, “high availability” and “strong consistency” conflict, both cannot be achieved simultaneously.
- Dynamo designed to be highly available (may require adding more replicas), with Eventual Consistency!
- Dynamo designed to be always writable (that is write never fails). Amazon feels that write failing would be annoying to its customers!
- It lets client to resolve the write conflicts. Always writable may result “write conflict”, though rare, systems lets client resolving, rather than system itself doing it.

Dynamo DB – Design Considerations[1]

- Other key principles embraced in design are:
- **Incremental scalability**: Dynamo should be able to scale out one node at a time, with minimal impact on both operators of the system and the system itself.
- **Peer to Peer and Symmetry**: All nodes in a instance (cluster) are same in terms of responsibilities. Any node can be used for read and any node can be used for write. Symmetry simplifies the process of system provisioning and maintenance.
- **Heterogeneity**: Load assigned to a node accounts for the capability of the node; powerful ones can take more load, and so

Dynamo DB Implementation techniques [1]

- Here is summary of techniques that are used in implementing Dynamo DB

Problem	Technique	Advantage
Partitioning	Consistent Hashing	Incremental Scalability
High Availability for writes	Vector clocks with reconciliation during reads	Version size is decoupled from update rates.
Handling temporary failures	Sloppy Quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available.

Dynamo DB Implementation techniques [1]

- Here is summary of techniques that are used in implementing Dynamo DB

Problem	Technique	Advantage
Recovering from permanent failures	Anti-entropy using Merkle trees	Synchronizes divergent replicas in the background.
Membership and failure detection	Gossip-based membership protocol and failure detection.	Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information.

"Partitioning" and "Replication" in Dynamo DB

- Let us first see how "Partitioning" and "Replication" is done in dynamo DB
- As already said Dynamo DB use technique called "consistent hashing" for this purpose!

Hashing for partitioning

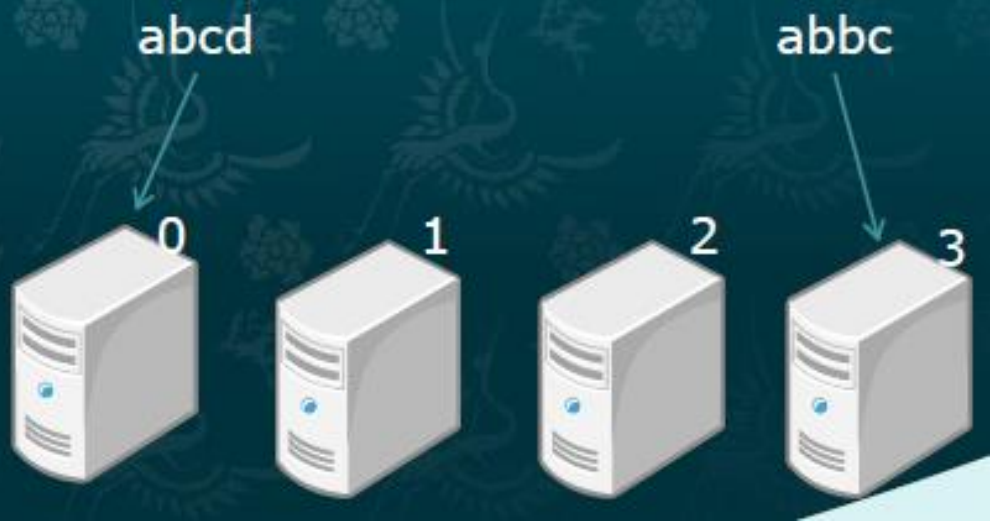
- Let us say, we use hashing for partitioning, that is
- An item “key” should map to a “Node”
- Consider a number of nodes N, onto which we want to distribute our data.
- Let there be a “partition function” that returns node_id for given search key value: node_id = hash_code(key) mod N
- We use this function to determine a “partition” for a data item (for putting as well as getting)

Example[1](Suppose, we have 4 nodes)

Binary	01100001	01100010	01100011	01100100
Hex	6 1	6 2	6 3	6 4
Ascii	a	b	c	d

abcd hashes to 0
0x61626364 = 1633831724
16338831724 % 4 = 0

Abbc hashes to 3
0x61626263 = 1633837667
1633837667 % 4 = 3



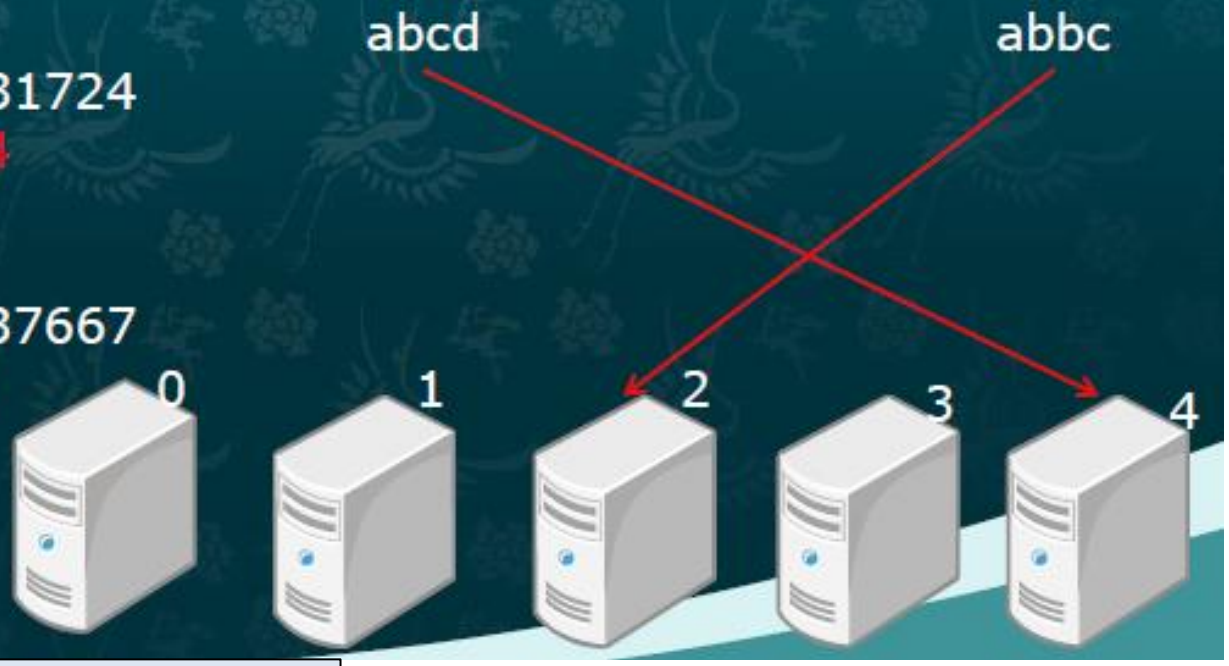
1: https://www.slideshare.net/jooholee81/consistent-hashing

Example[1], we add one, and make 5 nodes

Binary	01100001	01100010	01100011	01100100
Hex	6 1	6 2	6 3	6 4
Ascii	a	b	c	d

abcd hashes to 0
0x61626364 = 1633831724
16338831724 % 5 = 4

Abbc hashes to 3
0x61626263 = 1633837667
1633837667 % 5 = 2



1: https://www.slideshare.net/jooholee81/consistent-hashing

Problem with simple Hashing

- It should be easy to note that adding a node require reshuffling all the existing data.
- In a scalable system, it is reality that N is quiet dynamic. New nodes may be added any time, or an node may be removed any time, and so forth.
- So this simple approach does not work!
- "**Consistent Hashing**" used as solution for dealing with dynamic nature of number of noes in a system, which is integral part of a elastic system.

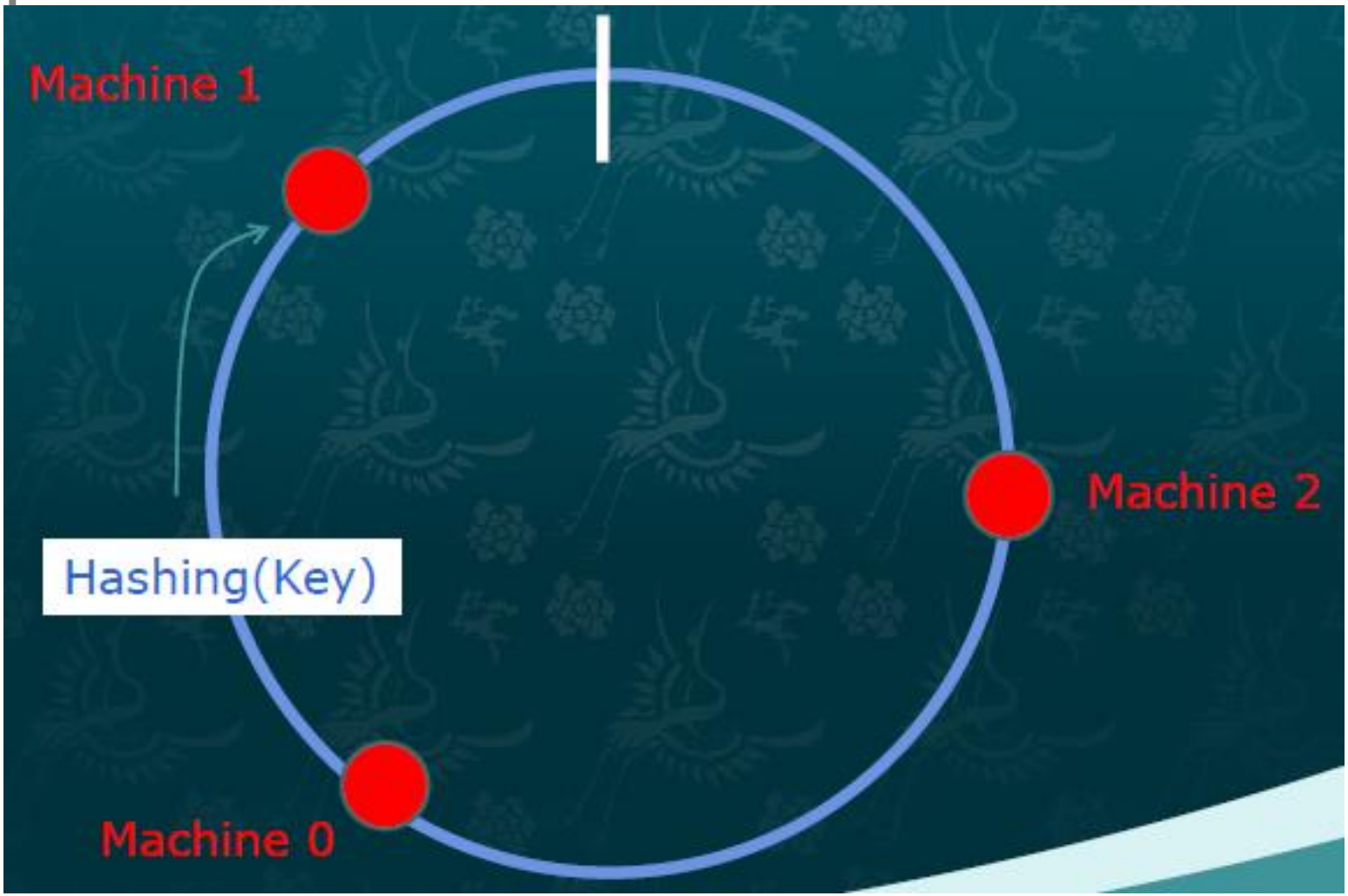
Consistent Hashing

- The idea of consistent hashing was introduced by David Karger et al. in 1997 through paper “Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web.” [2]
- Though originally the paper came in the context of “using distributed caches for a popular website”
  - Where number of caches are dynamic to deal with the load so!
- There after, the technique has been used in many cases-
  - Distributed Hash Table implementations like Chord
  - Some distributed memory object caching systems
  - No SQL databases like Amazon’s Dynamo

Consistent Hashing

- All keys are distributed on a circle ring. Let us say circle ring has been calibrated to 0 to 2^{32}-1 points.
- Each server/node is mapped to a point on a circle.
- Each data item "key" maps to some point on the ring, and maps to next server on the ring in clockwise direction.
- To make algorithm effective: uniform distribution of data items over nodes is desirable.

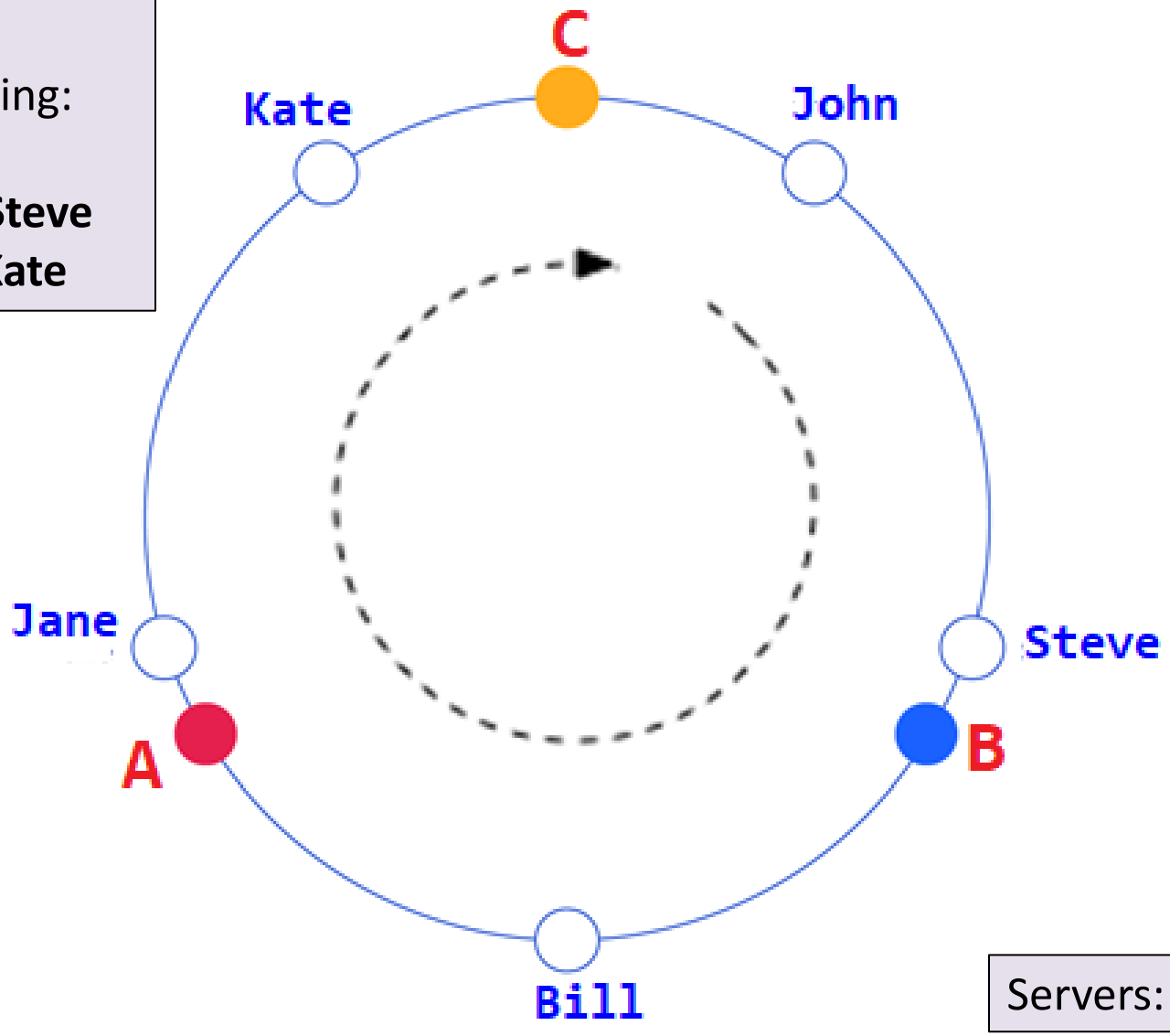
Consistent Hashing: Idea



1: https://www.slideshare.net/jooholee81/consistent-hashing

Consistent Hashing

Key values hashed onto ring and lands to servers as following:
Server A: Bill
Server B: John, Steve
Server C: Jane, Kate

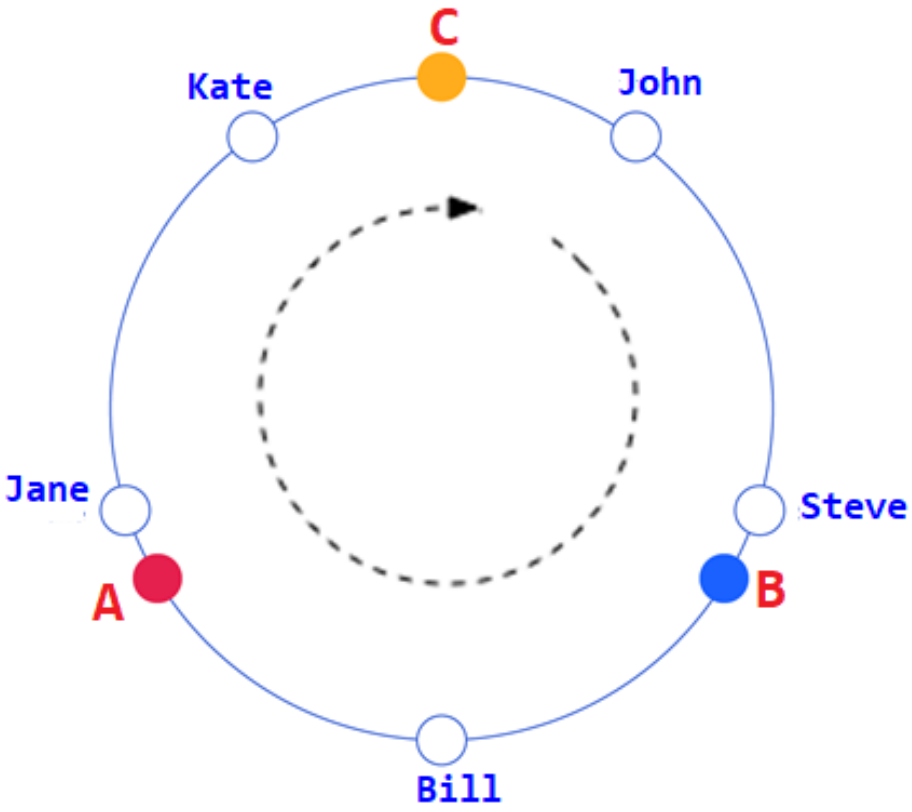


Servers: **A,B,C**

2: https://www.toptal.com/big-data/consistent-hashing

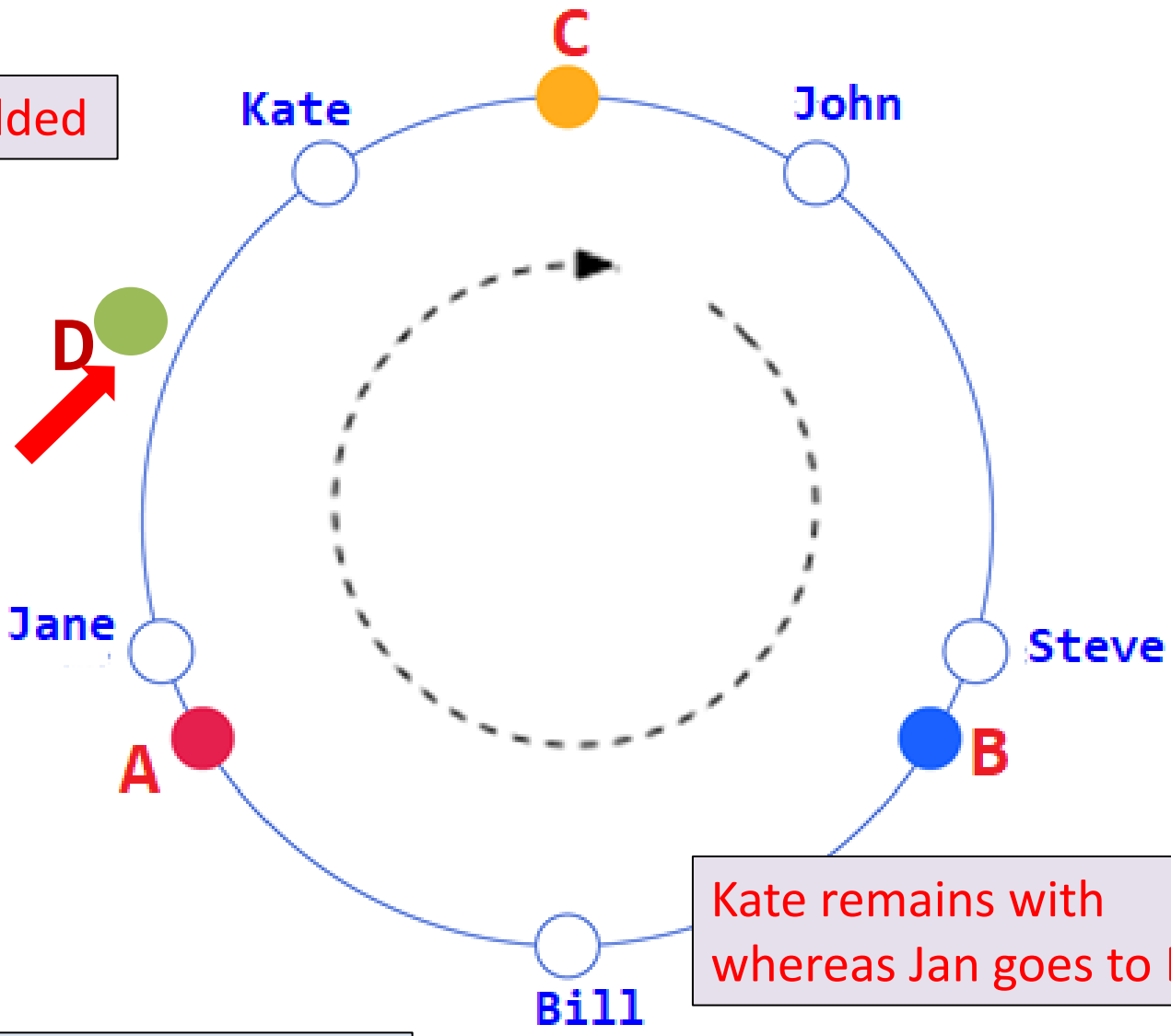
Consistent Hashing

- Server A stores all values falling in segment BA
- Similarly, Node C and B store data of segments AC and CB respectively!



Consistent Hashing

New Server D added



Kate remains with whereas Jan goes to D

2: https://www.toptal.com/big-data/consistent-hashing

Consistent Hashing - Objective

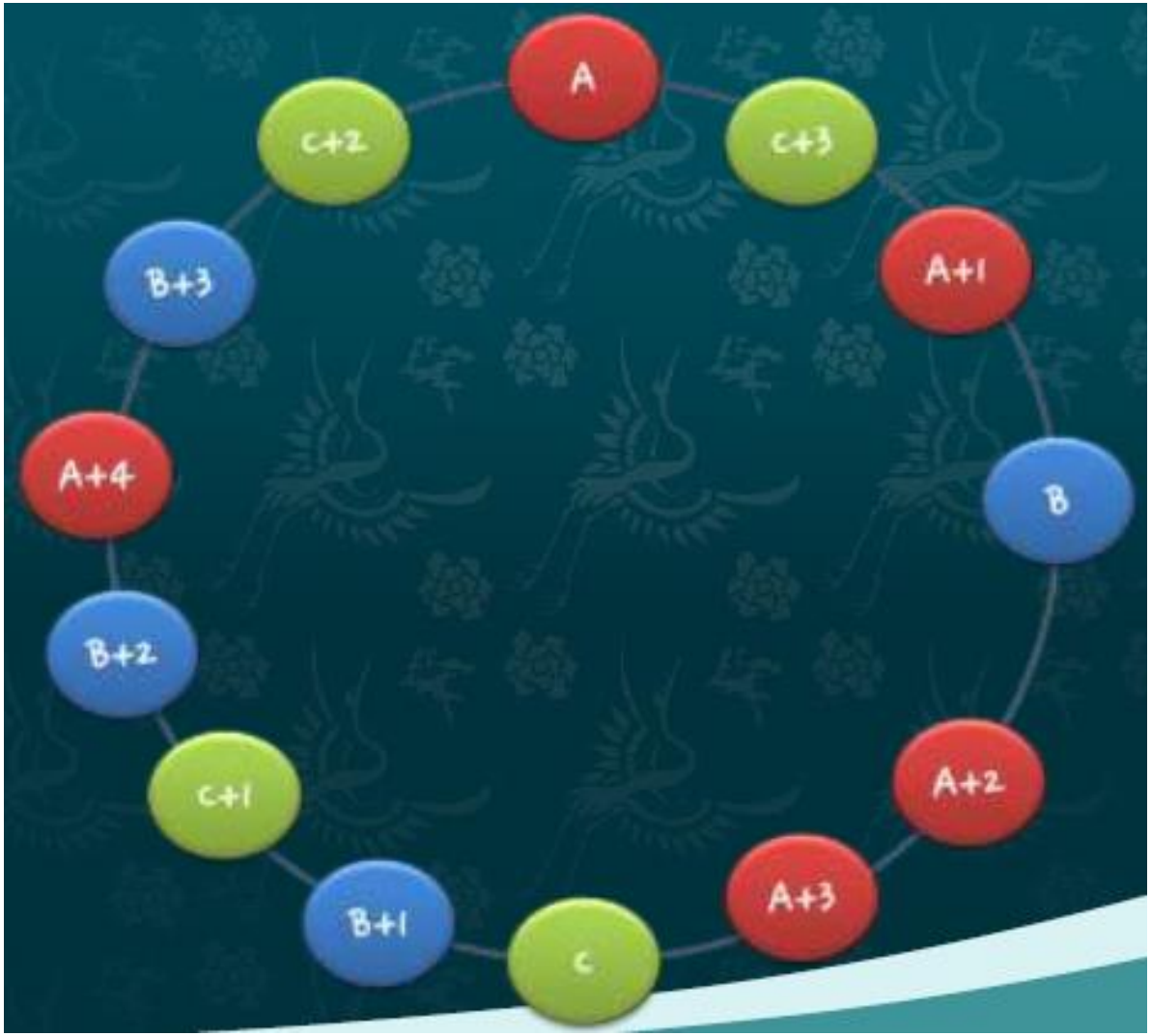
- When new node is added, there should be minimum shifting of data still maintaining uniform distribution.
- For adding Nth node, only 1/N of the keys should move.
- Suppose, there are already three nodes, and fourth one is to be added.
- Fourth node should get 25% of each of existing three node.

Consistent Hashing - Challenges

- Dynamo DB article[1] reports following two challenges:
- **First**: uniform distribution of data across nodes
- **Second**: It is unfair to assume that all nodes have the same power, more powerful nodes, in terms of RAM, CPUS, Disk, etc., should be able to take more load!
- To address these issues, Dynamo DB uses the concept of "**virtual nodes**".
- The idea of "virtual nodes" is that we have **multiple random representations on ring for each physical node**.

Concept of “Virtual Node”

- Servers A, B, C have multiple representations, randomly placed on the ring - **Virtual Nodes**
- This way we each node gets share from different segments of ring.



1: https://www.slideshare.net/jooholee81/consistent-hashing

Concept of "Virtual Node"

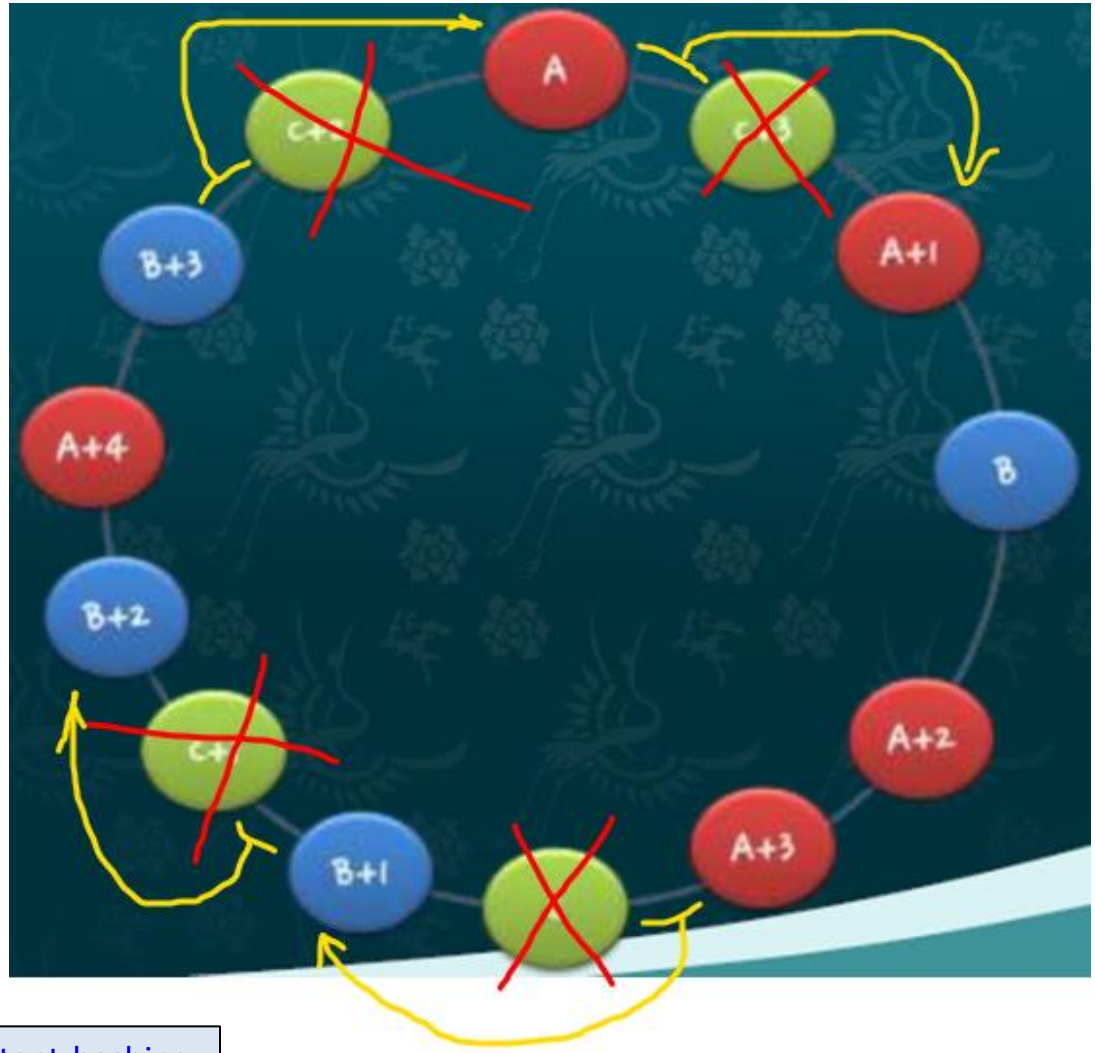
- "Virtual Node" is assignment of a physical node at multiple positions on the ring, and therefore Dynamo also calls them as **tokens** for the physical node.
- The concept of virtual nodes server following purposes:
  - If a node becomes unavailable (due to failures or routine maintenance), the load handled by this node is evenly dispersed across the remaining available nodes.
  - When a node becomes available again, or a new node is added to the system, the newly available node accepts a roughly equivalent amount of load from each of the other available nodes.

How virtual nodes help?

- In terms of uniform distribution:
  - By having representation of a node in various small-small segment, should be able to get fair distribution.
- In terms of loads
  - More powerful system can have more than average virtual nodes; and less powerful system can have lesser number of virtual nodes.
- When some node gets dropped, it is easy to distribute its load to remaining nodes in approximately equal amount.
- When addition of node happens, it is easy to gets approximately equal amount of data from all existing nodes

How does “virtual node” helps?

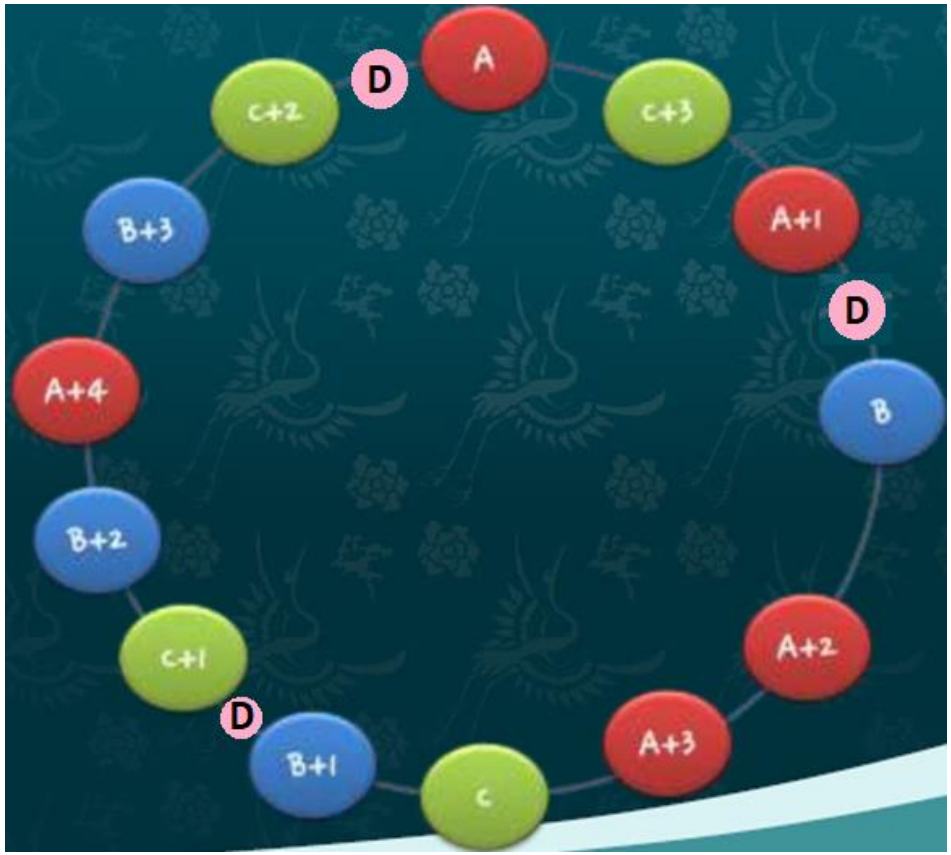
- Suppose node C is down or getting removed!
- All the segments landing to C are distributed A and B both, and hopefully equally!



1: https://www.slideshare.net/jooholee81/consistent-hashing

How does “virtual node” helps?

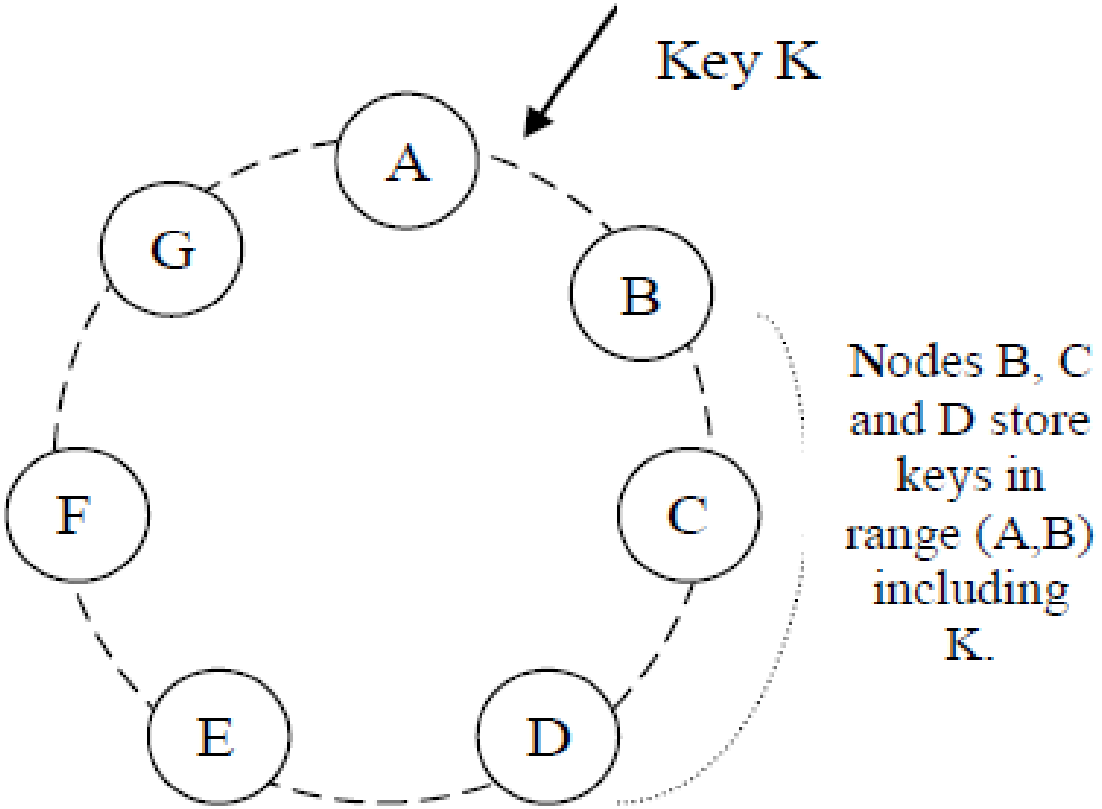
- When D new node comes in
- It take load from A, B, and C, and hopefully equal.



1: https://www.slideshare.net/jooholee81/consistent-hashing

Replication – Dynamo DB[1]

- Dynamo replicates its data on multiple hosts. Each data item is replicated at N hosts, where N is a parameter configured.



Replication – Dynamo DB[1]

- "Key segments" are replicated on multiple node
- For a segment, in addition to storing all data on first node, data are also replicated onto clockwise successor N-1 "nodes" on the ring
  - Where N is replica factor (configured, say N=3)
- For example, in diagram (previous slide) data for range AB are stored on B, and replicated on C, and D
- This results in a system where each node is responsible for the region of the ring between it and its Kth predecessor.
- So, in figure, node D stores the keys that fall in the ranges (A, B], (B, C], and (C, D].

Replication – Dynamo DB[3]

- Also, there is a notion of *preference list* for a key (every key).
  - The list of nodes that is responsible for storing a particular key.
- To account for node failures, preference list contains more than N nodes.
- At the time of write, case of failure of a node, additional node from preference list is taken and used for “write”!
- Note that the preference list for a key is constructed by skipping positions in the ring to ensure that the list contains only distinct physical nodes.

Summarized

- So, hopefully, this demonstrates
  - What is consistent hashing is?
  - how consistent hashing in with Virtual Nodes is used for “partitioning” and “replication” in dynamo db!

- Now reading/writing on “Partitioned” and “Replicated” data
- Before, actually looking into read/write, let us see how data are “versioned” and “managed” in Dynamo.

References/Further Readings

[1] DeCandia, Giuseppe, et al. "Dynamo: amazon's highly available key-value store." *ACM SIGOPS operating systems review* 41.6 (2007): 205-220. https://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf

[2] Karger, David, et al. "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web." *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. 1997.

[3] White, Tom: *Consistent Hashing*. November 2007. – Blog post http://tom-e-white.com/2007/11/consistent-hashing.html .

[4] Deshpande, Tanmay. *Mastering DynamoDB*. Packt Publishing Ltd, 2014.