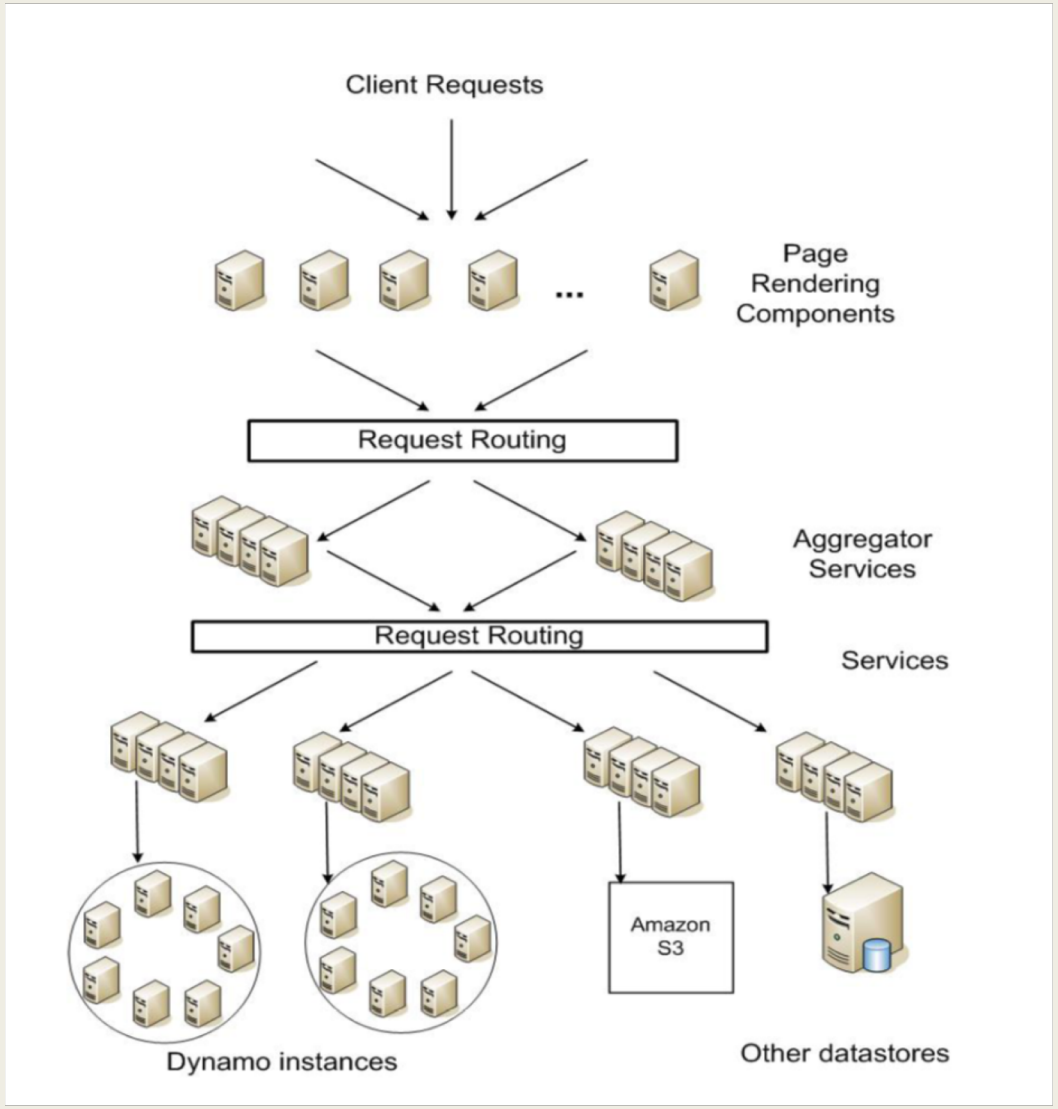
# DYNAMO: AMAZON'S HIGHLY AVAILABLE KEY-VALUE STORE

Presenter: Jill Zhou

# What is Dynamo?

- A highly available key-value storage system
- To achieve always-on experience, Dynamo sacrifices consistency under certain failure scenarios.
- Dynamo is a reliable and efficient, highly scalable storage system.




# Service-oriented architecture of Amazon's platform

- dynamic web content is generated by page rendering components which in turn query many other services
- Some services act as aggregators by using several other services to produce a composite response.
- A service can use different data stores to manage its state and these data stores are only accessible within its service boundaries

# SYSTEM ARCHITECTURE

# the core distributed systems techniques used in Dynamo:

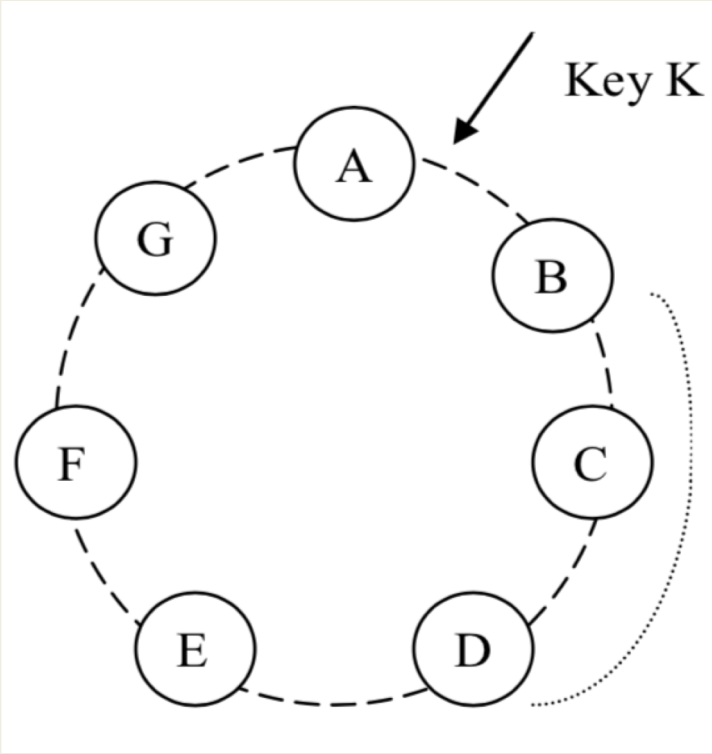
- Partitioning
- Replication
- Versioning
- Membership
- failure handling
- scaling

# System Interface

- Two operations:
  - *get(key)*
    - Return a single object or a list of objects
  - *put(key, context, object)*
    - Determine where the replicas of object should be placed based on the key
    - Writes the replicas to disk

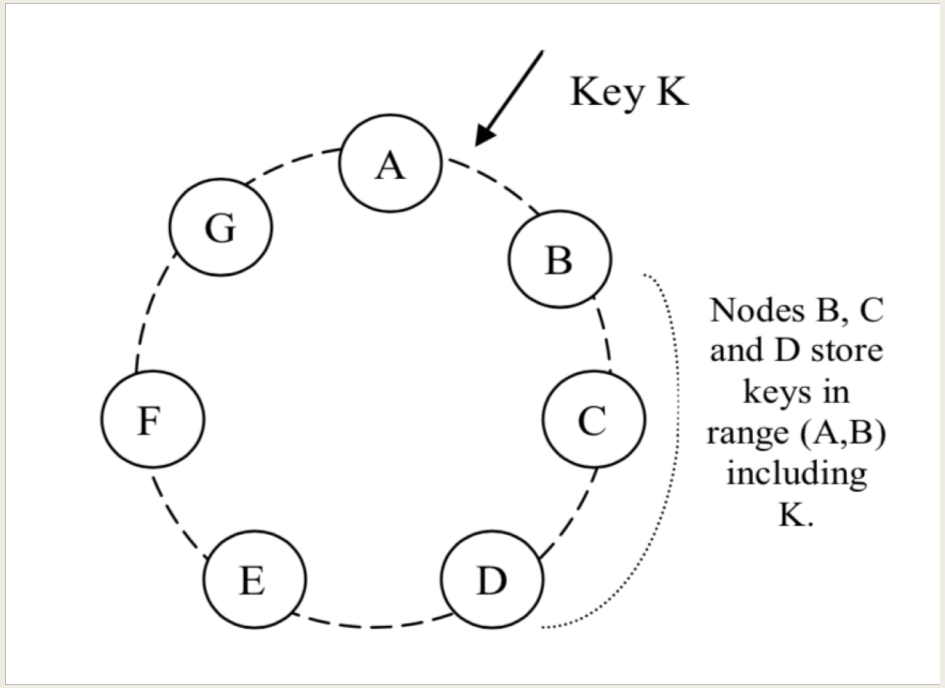
# Partitioning Algorithm

- Technique:
  - *Consistent Hashing :*
    - h(k) in [min, max]
    - [min, max] is the positions in a ring
    - Each node (storage host) has its own position (a number)
    - For a data item (key, value), h(key) can be used to decide the store position.
- Advantage:
  - *Scale incrementally*



# Replication

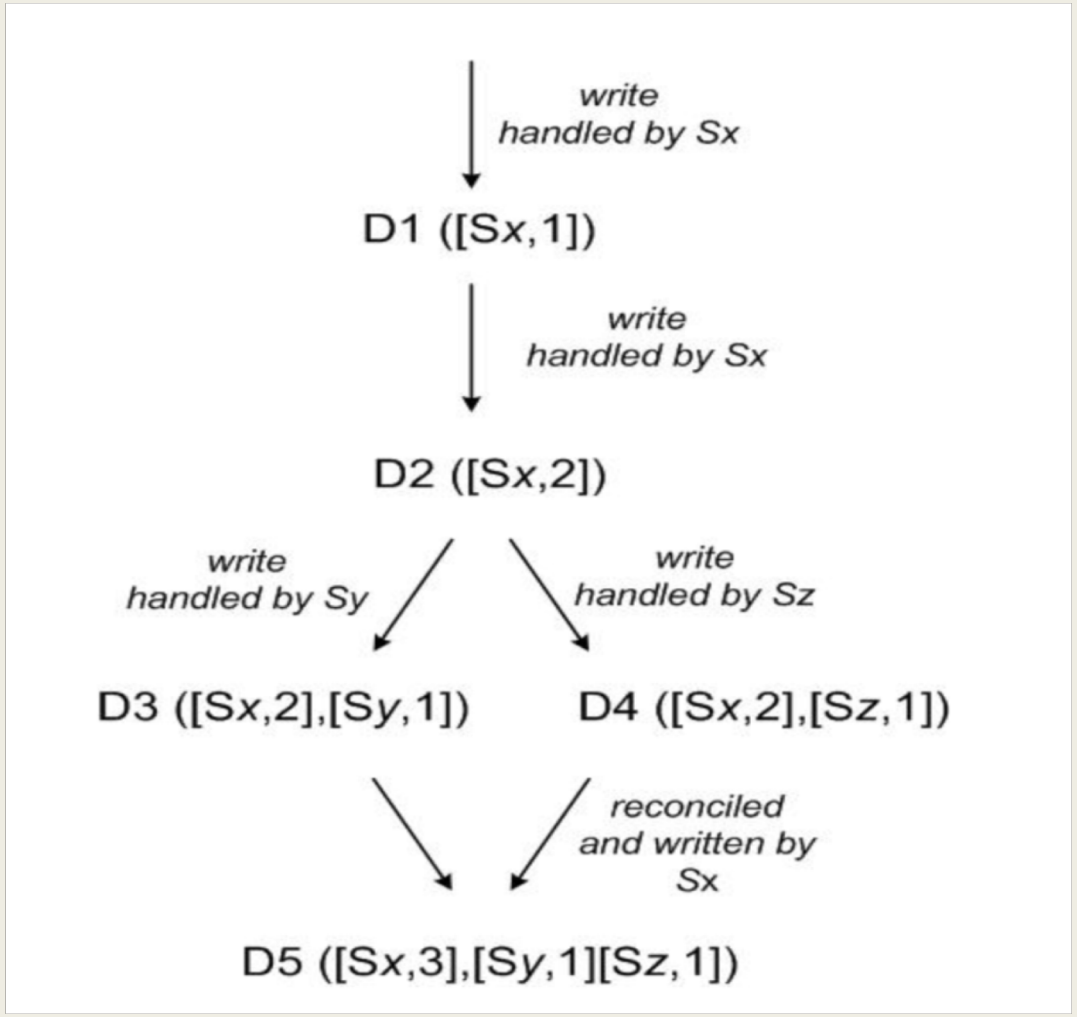
- To achieve **high availability** and **durability**, Dynamo replicates its data:
  - *On N hosts (clockwise successor)*
  - *Usually set N = 3*



# Data Versioning

- Updates to be propagated to all replicas asynchronously
  - *Put() may return before the updates applied to all replicas*
  - *Get() may return an object that does not have the latest updates.*
- It causes many versions of data

- E.g.:
  - *“add to cart” and “delete item from cart” are two put requests*
  - *If a customer wants to add an item to (or remove from) a shopping cart and the latest version is not available*
  - *the item is added to (or removed from) the older version and the divergent versions are reconciled later*




# Data Versioning

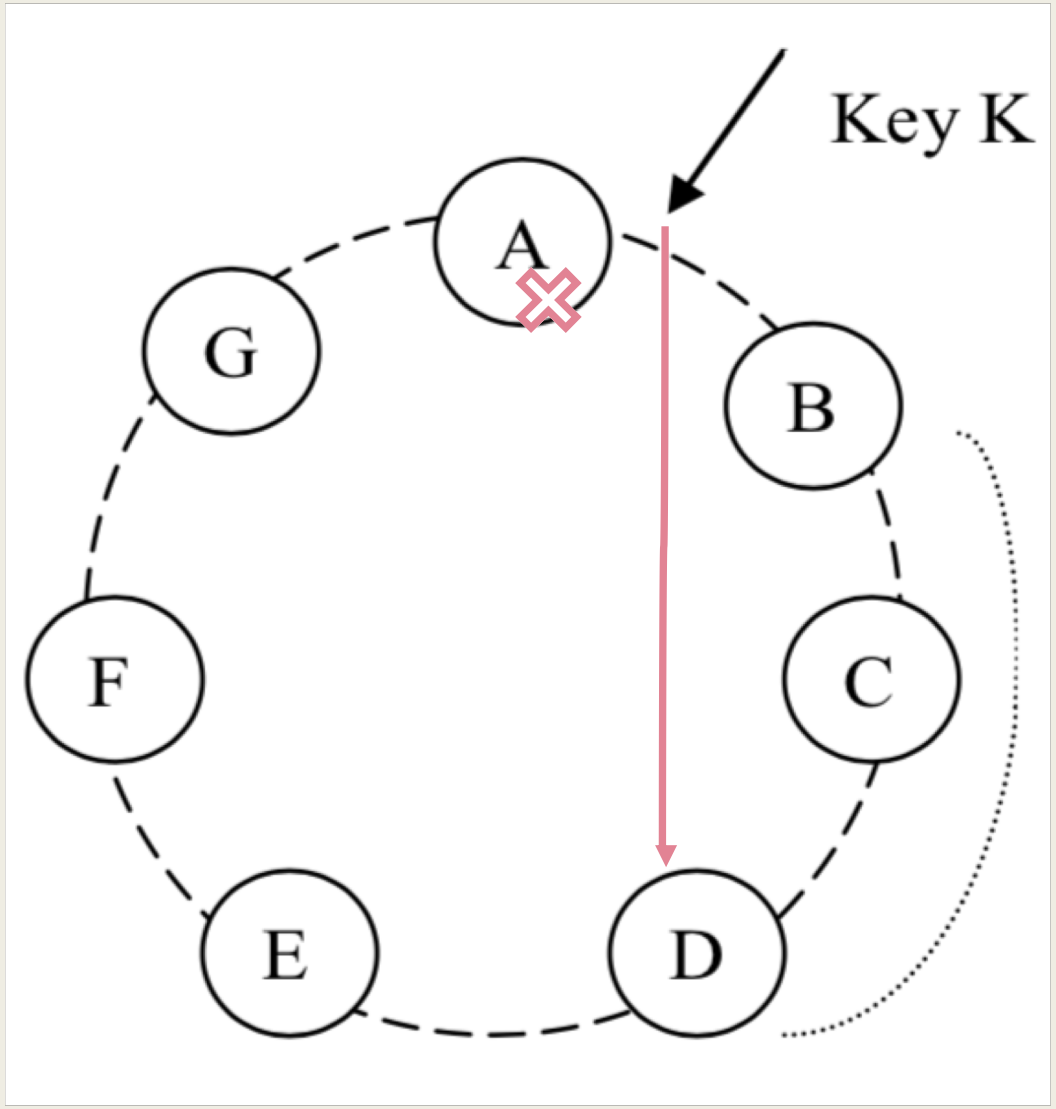
- In order to merge different versions of data and preserve all information correctly, Dynamo use **Vector clock:**
  - *A list of (node, counter) pairs*
  - *One vector clock is associated with every version of every object*
  - *From vector clock, we can tell whether two versions of an object are on parallel branches or have a causal ordering*
  - *If the counters on the first object's clock are less-than-or-equal to all of the nodes in the second clock, then the first is an ancestor of the second and can be forgotten.*
  - *Otherwise, the two changes are considered to be in conflict and require reconciliation.*

# Execution of get () and put () operations

- Read and write operations involves the first N healthy nodes in the preference lists
  - *Nodes not in the list will not be in charge*
  - *A node handling a read or write operation is known as the coordinator.*
- Dynamos uses consistency protocol to maintain consistency among its replicas
  - *This protocol has two key configurable values: R and W*
    - R is the minimum number of nodes that must participate in a successful read operation.
    - W is the minimum number of nodes that must participate in a successful write operation.

# Execution of get () and put () operations

- Upon receiving a put() request for a key
  - *the coordinator generates the vector clock for the new version*
  - *writes the new version locally.*
  - *The coordinator then sends the new version (along with the new vector clock) to the N highest-ranked reachable nodes.*
  - *If at least W-1 nodes respond then the write is considered successful.*
- for a get() request:
  - *the coordinator requests all existing versions of data for that key from the N highest-ranked reachable nodes in the preference list for that key*
  - *waits for R responses before returning the result to the client.*
  - *If gathering multiple versions of the data, it returns all the versions it deems to be causally unrelated.*




# Handling Failures: Hinted Handoff

- If A is temporarily failed, send the replica to node D.
  - *The replica has hint in meta data say it should be A*
- Node D receive hinted replicas will keep them in a separate local database
- Upon detecting that A has recovered, D will attempt to deliver the replica to A, and may delete the replica.

# Handling permanent failures: Replica synchronization

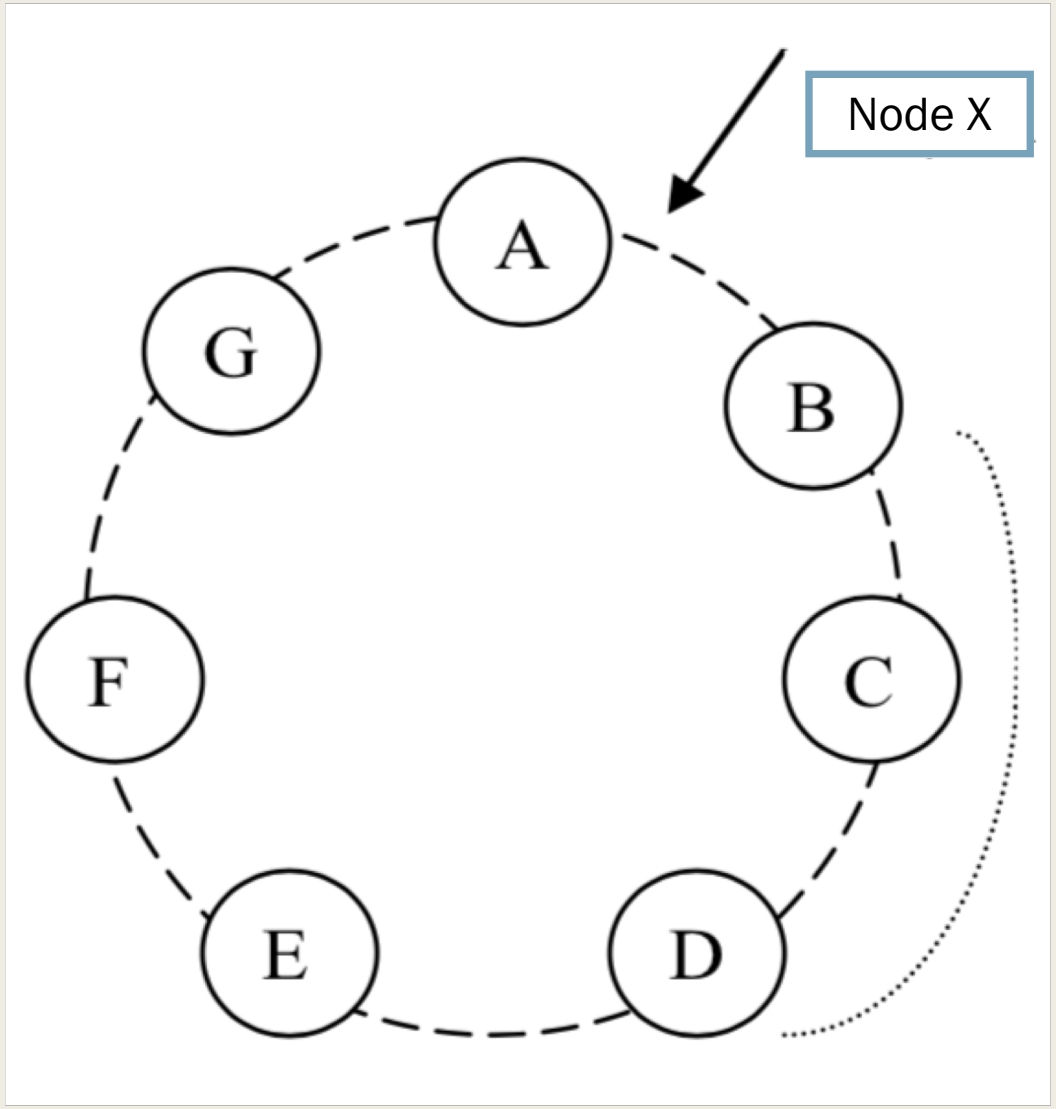
- There are scenarios under which **hinted replicas become unavailable** before they can be returned to the original replica node
- To detect the **inconsistencies between replicas faster** and to **minimize the amount of transferred data**, Dynamo uses **Merkle trees**:
  - *Merkle tree is a hash tree where leaves are hashes of the values of individual keys.*
    - Advantage:
      - *each branch of the tree can be checked independently without requiring nodes to download the entire tree or the entire data set*
  - *if the hash values of the root of two trees are equal, then the values of the leaf nodes in the tree are equal and the nodes require no synchronization. If not, the nodes may exchange the hash values of children and the process continues until it reaches the "out of sync" leaves of the trees*

# Membership

- *Ring Membership*
  - *When a node starts for the first time, it chooses **its set of tokens** (virtual nodes in the consistent hash space) and **maps nodes to their respective token sets.***
  - *The mapping is persisted on disk*
- *External Discovery*
  - *The mechanism described above could temporarily result in **a logically partitioned** Dynamo ring.*
    - For example, the administrator could contact node A to join A to the ring, then contact node B to join B to the ring. In this scenario, nodes A and B would each consider itself a member of the ring, yet neither would be immediately aware of the other.
  - *To prevent logical partitions, some Dynamo nodes play the role of **seeds.***
    - Seeds are nodes that are discovered via an external mechanism and are known to all nodes
    - all nodes eventually reconcile their membership with a seed

# Failure Detection

- node A may consider node B failed if node B does not respond to node A’s messages (even if B is responsive to node C's messages).
- If there is a steady rate of client requests generating inter-node communication in the Dynamo ring:
  - *a node A quickly discovers that a node B is unresponsive when B fails to respond to a message*
  - *Node A then uses alternate nodes to service requests that map to B's partitions*
  - *A periodically retries B to check for the latter's recovery*
- In the absence of client requests to drive traffic between two nodes, both nodes do nothing




# Adding/Removing Storage Nodes

- When X is added to the system, it is in charge of storing keys in the ranges (F, G], (G, A] and (A, X].
- As a consequence, nodes B, C and D no longer have to store the keys in these respective ranges.
- When a node is removed from the system, the reallocation of keys happens in a reverse process.

# IMPLEMENTATION

# the main patterns in which Dynamo is used

- *Business logic specific reconciliation:*
  - *In case of divergent versions, the client application performs its own reconciliation logic.*
- Timestamp based reconciliation:
  - *In case of divergent versions, Dynamo performs simple timestamp based reconciliation logic of “last write wins”*
- *High performance read engine:*
  - *these services have a high read request rate and only a small number of updates*
  - *Dynamo replicates their data across multiple nodes thereby offering incremental scalability.*

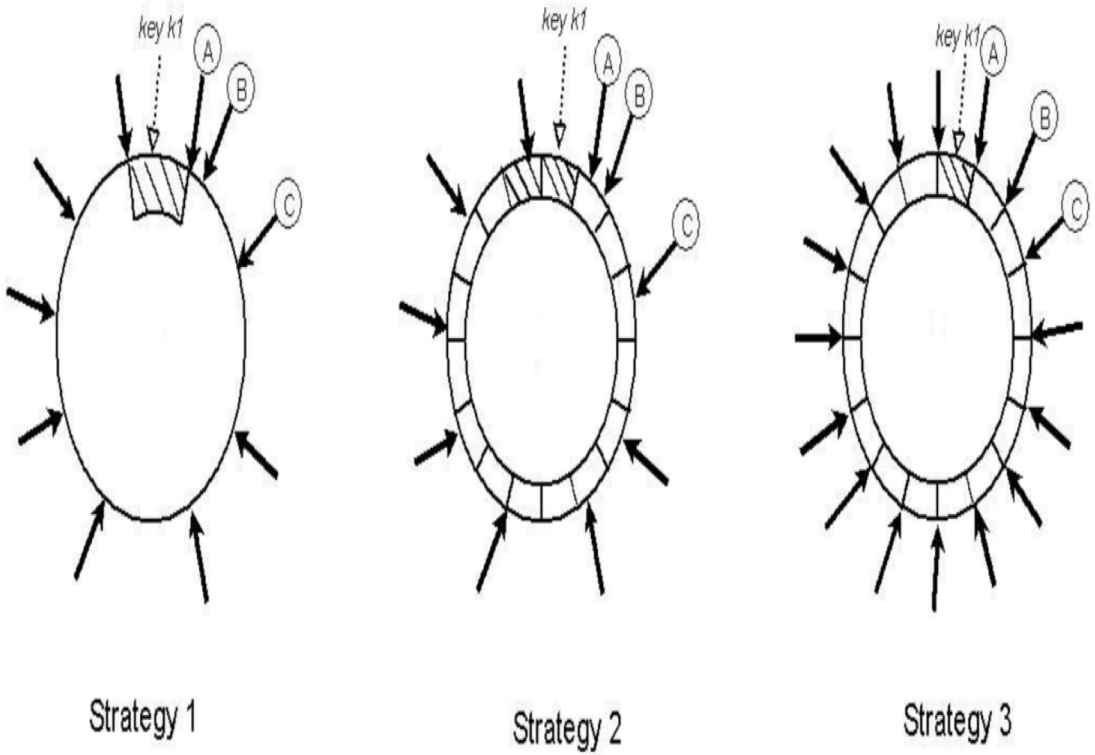



Figure 7: Partitioning and placement of keys in the three strategies. A, B, and C depict the three unique nodes that form the preference list for the key k1 on the consistent hashing ring (N=3). The shaded area indicates the key range for which nodes A, B, and C form the preference list. Dark arrows indicate the token locations for various nodes.

# Ensuring Uniform Load distribution

- *Strategy 1: T random tokens per node and partition by token value*
- *Strategy 2: T random tokens per node and equal sized partitions*
- *Strategy 3: Q/S tokens per node, equal-sized partitions*
  - *divides the hash space into Q equally sized partitions*
  - *S is the number of nodes in the system*

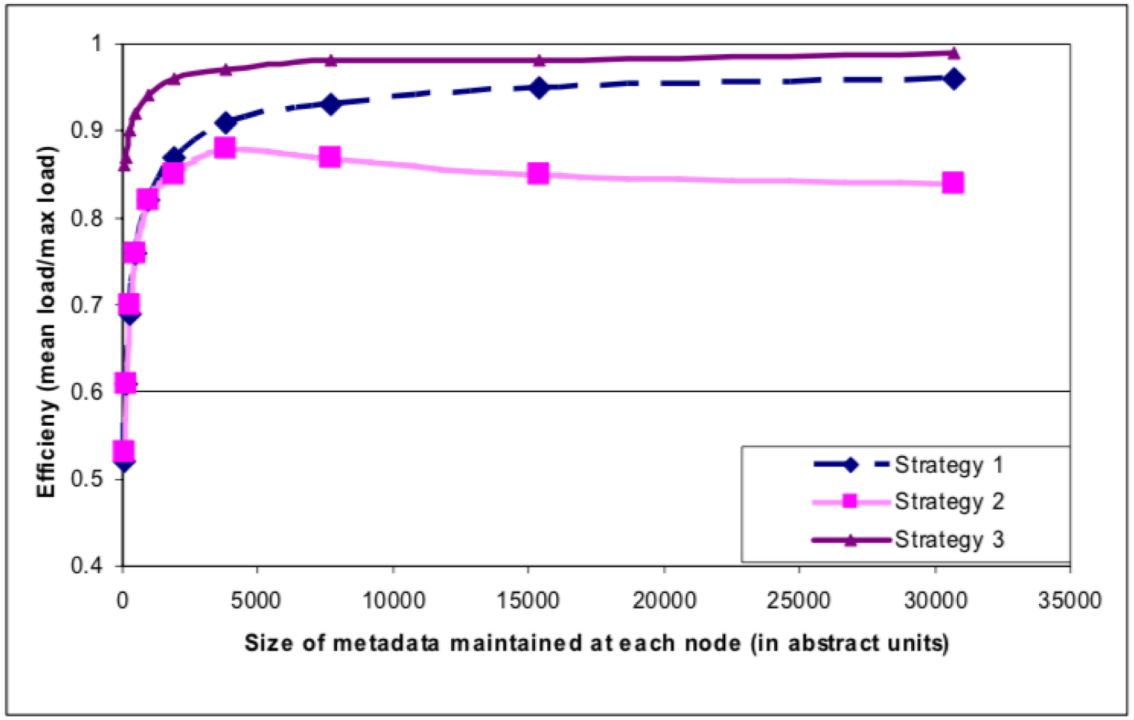



**Figure 8: Comparison of the load distribution efficiency of different strategies for system with 30 nodes and N=3 with equal amount of metadata maintained at each node. The values of the system size and number of replicas are based on the typical configuration deployed for majority of our services.**

# Ensuring Uniform Load distribution

- strategy 3 achieves the best load balancing efficiency
- strategy 2 has the worst load balancing efficiency.

# TAKE AWAY

# A SUMMARY OF THE LIST OF TECHNIQUES DYNAMO USES AND THEIR RESPECTIVE ADVANTAGES

| 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. |
| 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. |

# Q&A