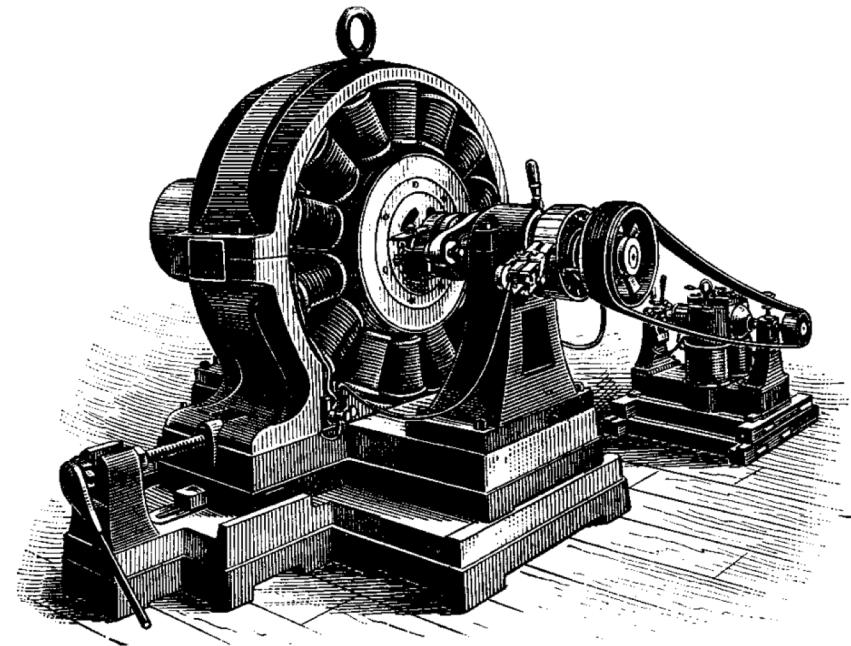
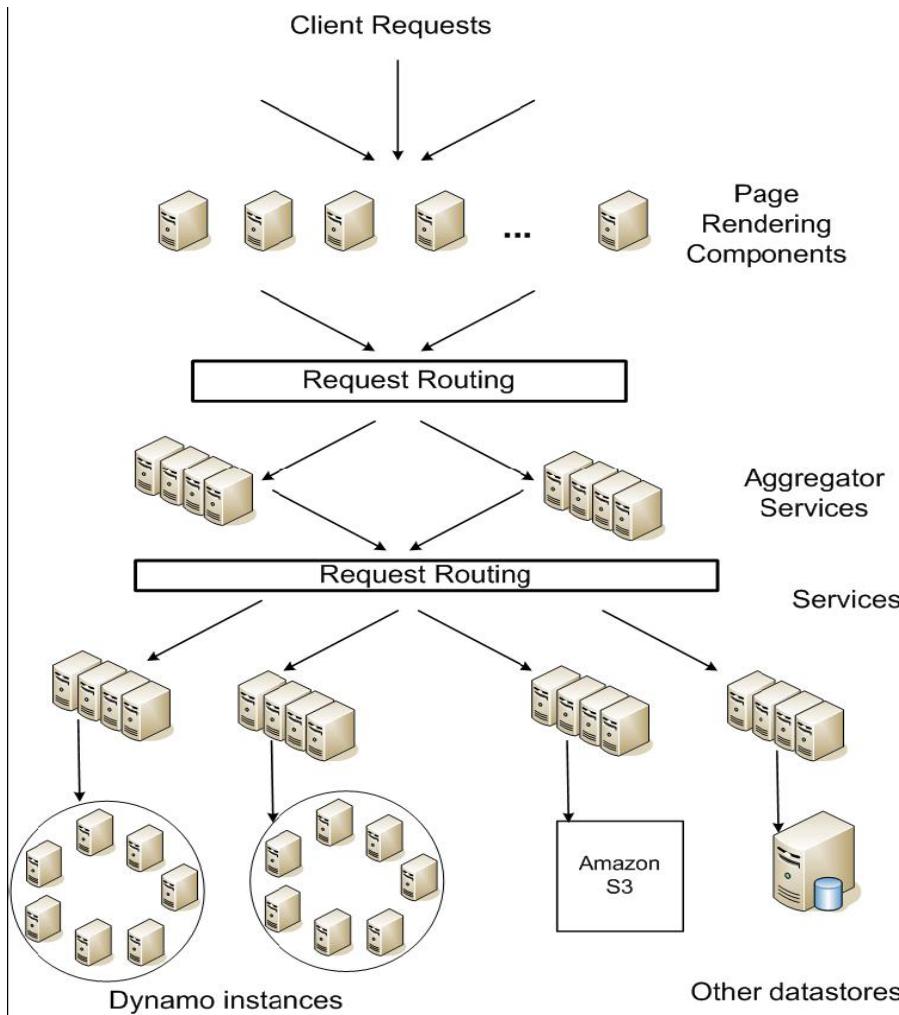


CS 5450

# Amazon Dynamo



# Amazon's Architecture



# Dynamo

---

- The platform for Amazon's e-commerce services: shopping chart, best seller list, produce catalog, promotional items etc.
- A **highly available, distributed key-value storage system**: put() & get() interfaces
- Aims to provide "**always on**" guarantee at the **cost of losing some consistency**

# Design Considerations

---

- Incremental scalability
  - Minimal management overhead
- Symmetry
  - For ease of maintenance, no master/slave nodes
- Decentralized control
  - Centralized approach leads to outages
- Heterogeneity
  - Exploit capabilities of different nodes

# CAP Theorem

---

**Consistency**  
single up-to-date copy



**Availability**  
for updates

**Partition  
Tolerance**

Pick 2 out of 3

# What Does CAP Really Mean?

---

“No system where P is possible can  
at all times guarantee both C and A”

if your network is highly reliable (and fast), so that P is extremely rare, you can aim for both C and A



Google Spanner

# Requirements

---

## Reliability:

customers should be able to edit their shopping cart even when there are:

- disk failures
- network failures
- tornados!

## Efficiency:

latency measurement is done at 99.9th percentile of the distribution

# ACID Properties

---

- ◆ Atomicity
- ◆ Consistency
- ◆ Isolation
- ◆ Durability

# Requirements

---

Query model:

only simple read/write to small data (less than 1MB), uniquely identified by a key

ACID properties:

- atomicity: important!
- consistency: weak is sufficient
- isolation: not at all
- durability: important!

# Consistency

---

- **BASE**
  - Basically Available, Soft State, Eventually Consistent
- **Always writable**
  - Can always write to shopping cart
  - Conflict resolution on reads
- **Application-driven conflict resolution**
  - Merge conflicting shopping carts to never lose “Add Cart”
  - push inconsistencies to Customer Service

# Requirements

---

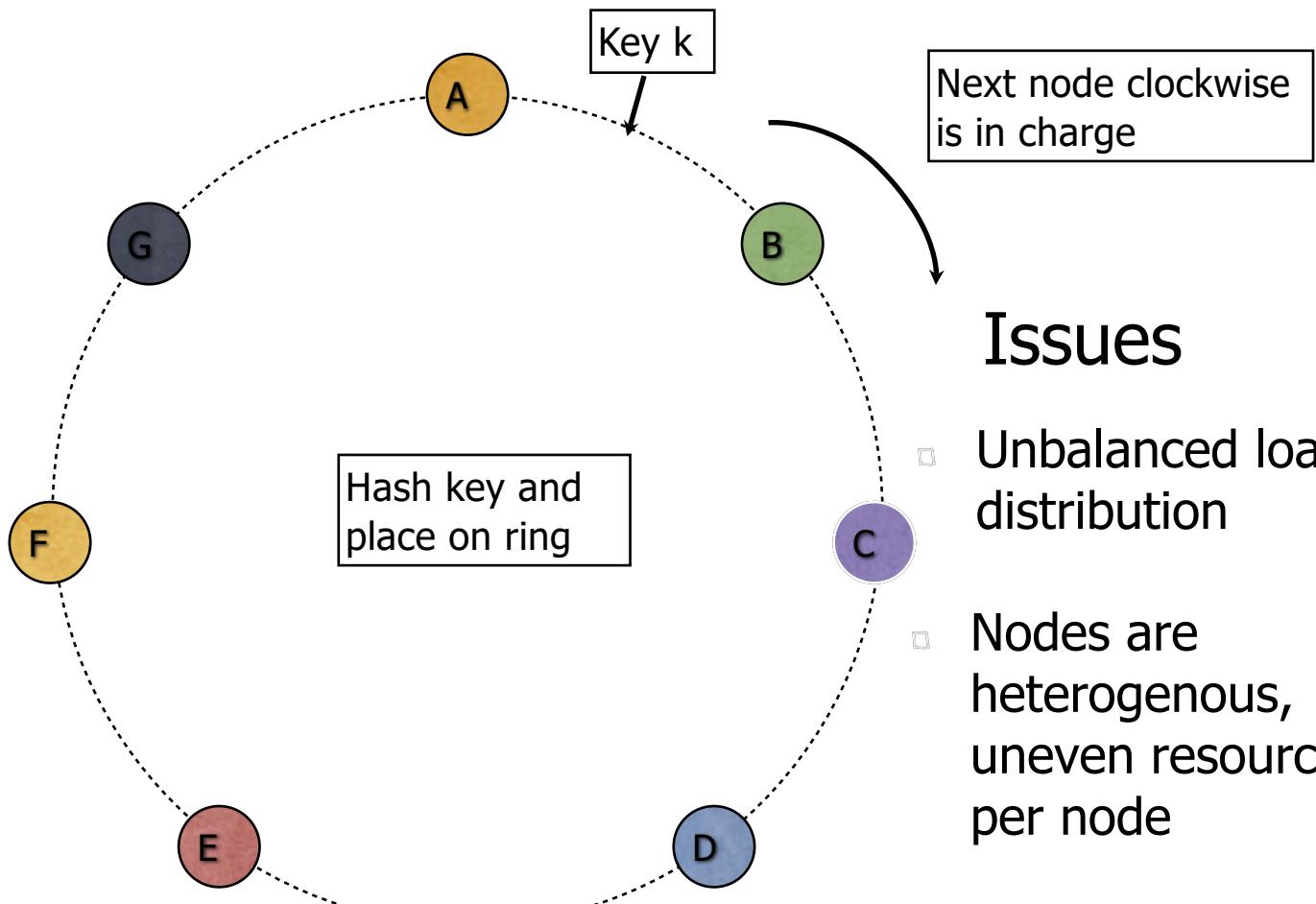
## Service Level Agreement (SLA):

- client and server agree on several characteristics related to the system
  - e.g 300 ms response time for peak load of 500 requests/second
- putting together a single webpage can require responses from 150 services for typical request

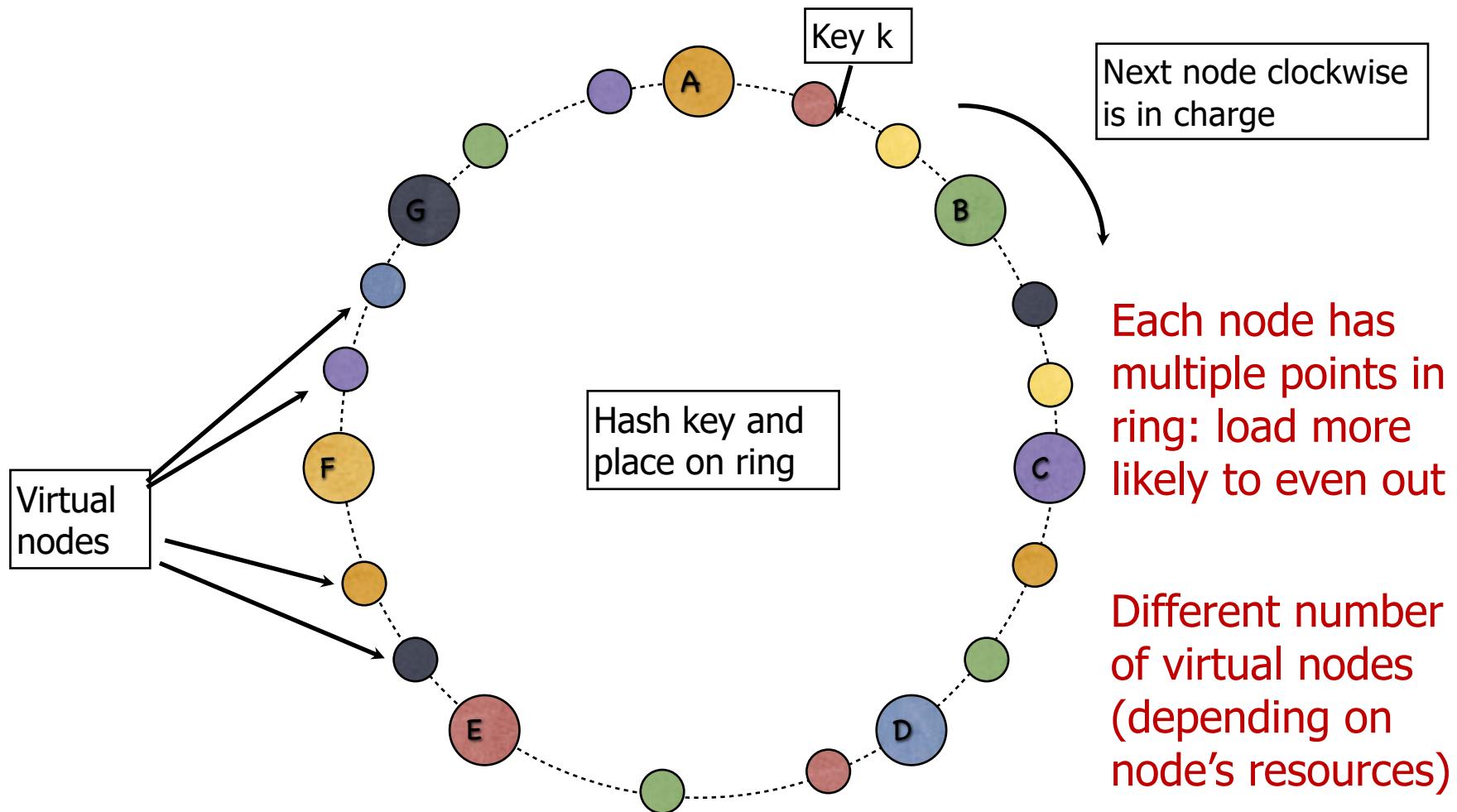
# Techniques Used by Dynamo

| 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 |

# Consistent Hashing

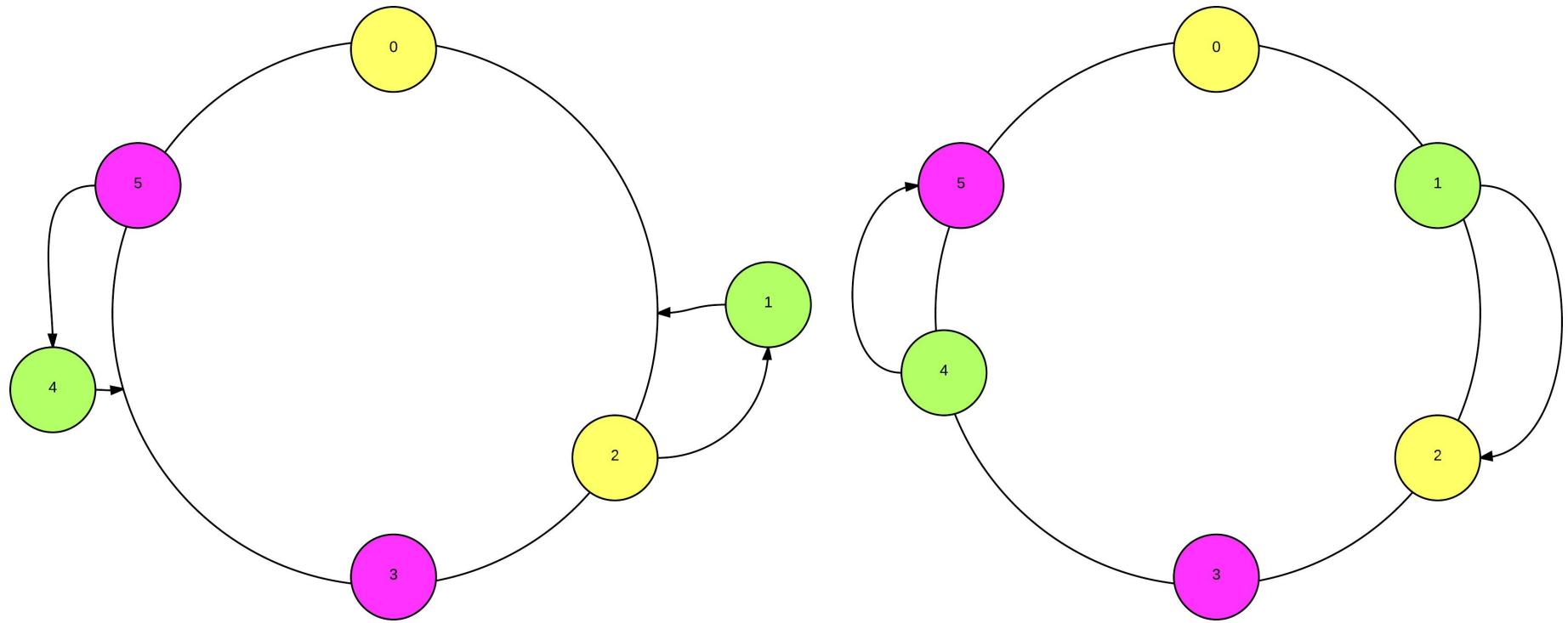


# Virtual Nodes



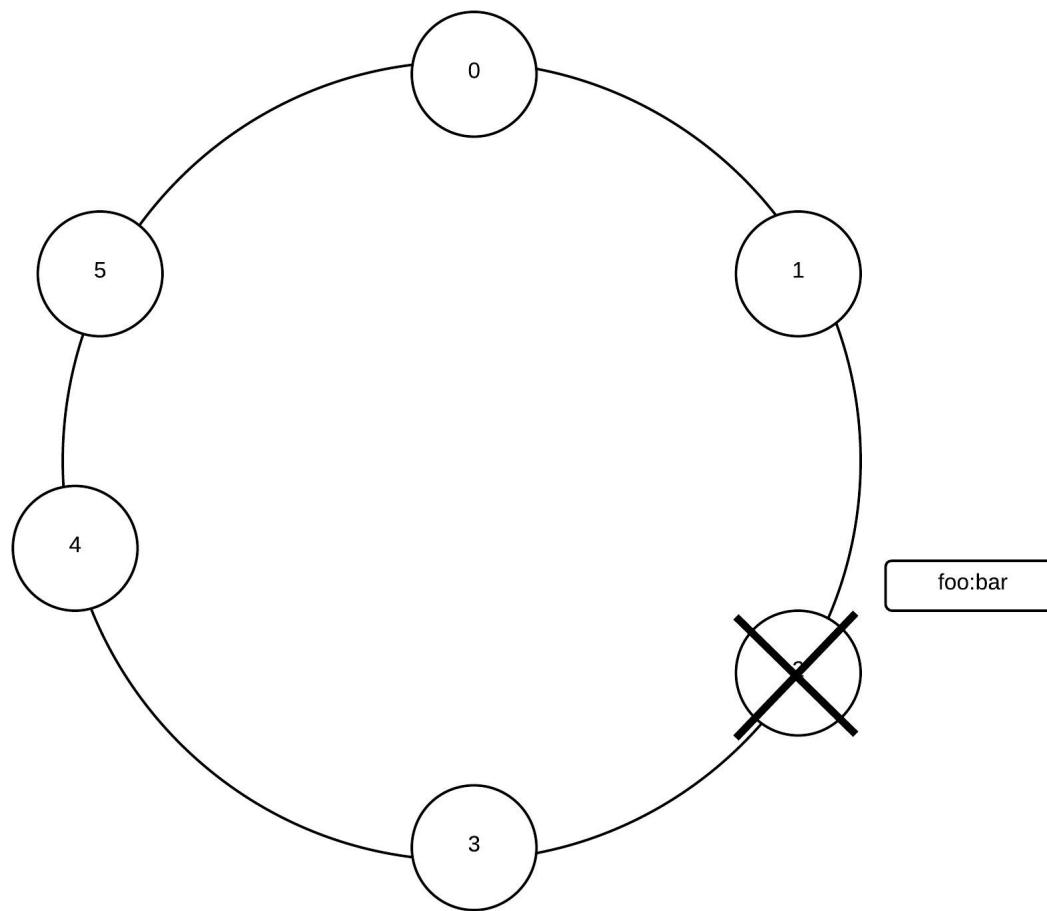
# Node Joining or Leaving

---

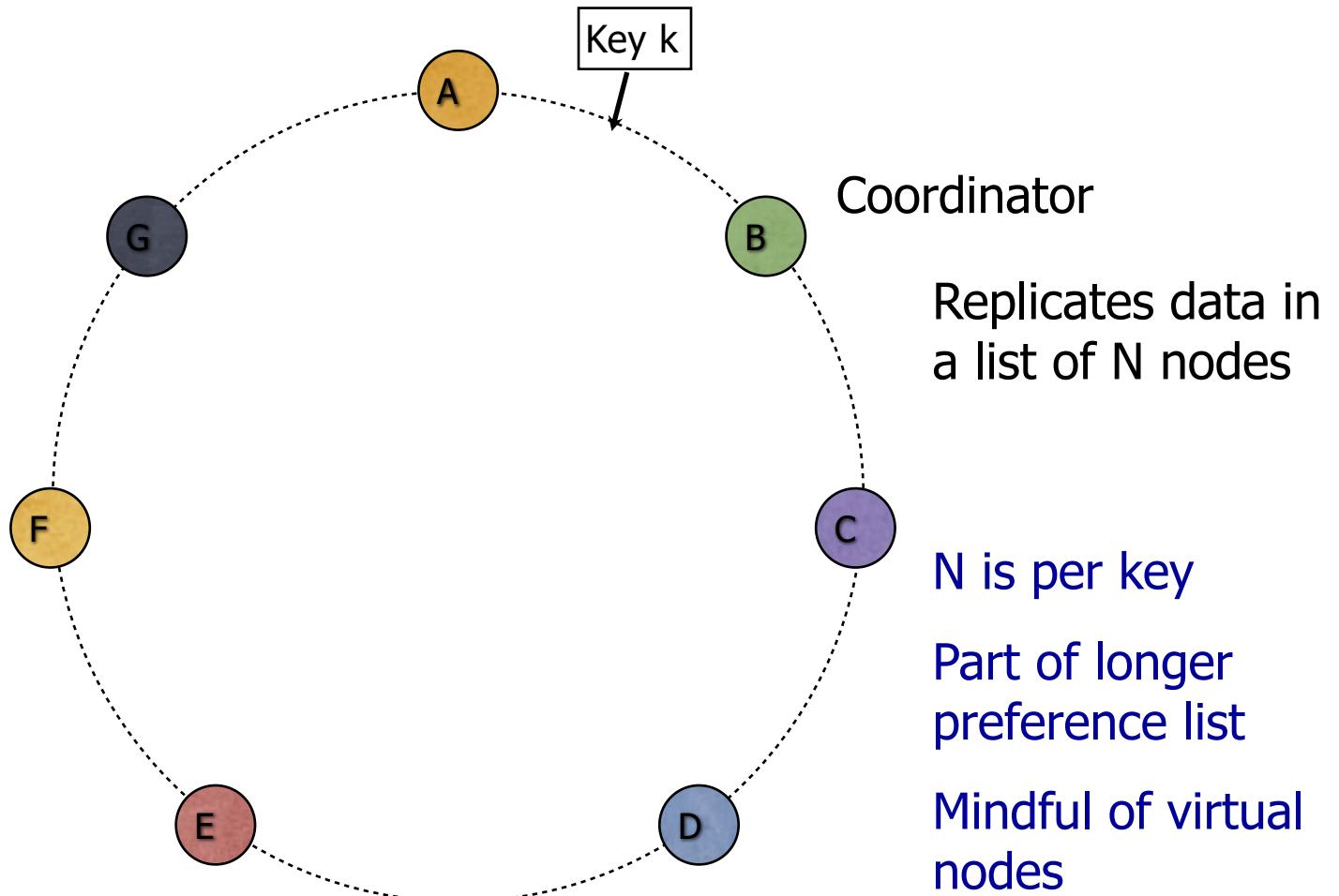


# What If

---



# Replication



# Techniques Used by Dynamo

| 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 |

# Quorums

---

put():

- coordinator writes new data locally
- sends to next  $n-1$  nodes
- when  $w-1$  respond, considered successful

get():

- coordinator requests next  $n-1$  nodes
- when  $r-1$  respond, considered successful
- $r + w > n$

# Data Versioning

---

- A put() call may return successfully to caller before the update has been applied to **all** the replications
  - If latest object version unavailable, apply put to earlier version, merge later
- A get() call may return **many** versions of the same object
- An object may have distinct version sub-histories that need to be reconciled

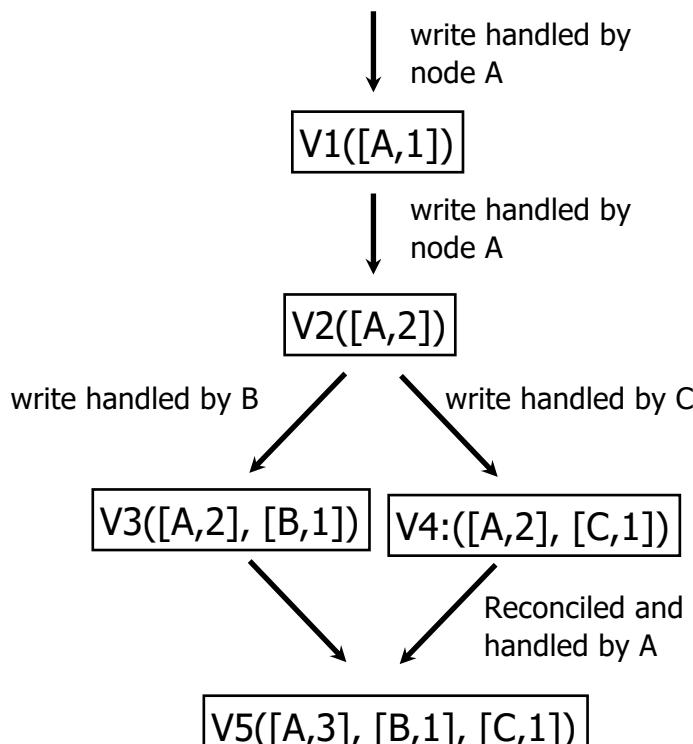
# Delayed Reconciliation

---

The goal: "add to cart" operation should never be forgotten or rejected

- When 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
- The divergent versions are reconciled later

# Vector Clocks in Dynamo



- List of (node, counter) pairs  
Counters are logical integer timestamps assigned by coordinator
- Each object version has one vector clock
- Used to determine whether a version is subsumed
- Stored in “context” in get(), put()

# Vector Clocks in Dynamo

---

Size of vector clock can get arbitrarily long

- bounded by N in practice
- drop oldest one when beyond certain threshold

Reconciliation

- Easy if values are causally ordered
- Otherwise, applications will handle

# Reconciliation (Summary)

---

- An "add to cart" operation will never be lost,
- but removed items may reappear

# Reconciliation in Practice

---

- In one 24 hour period, 99.94% of requests saw exactly one version
- Divergent version usually not caused by failure but concurrent writers....(rarely human beings, usually automated client programs)

# Techniques Used by Dynamo

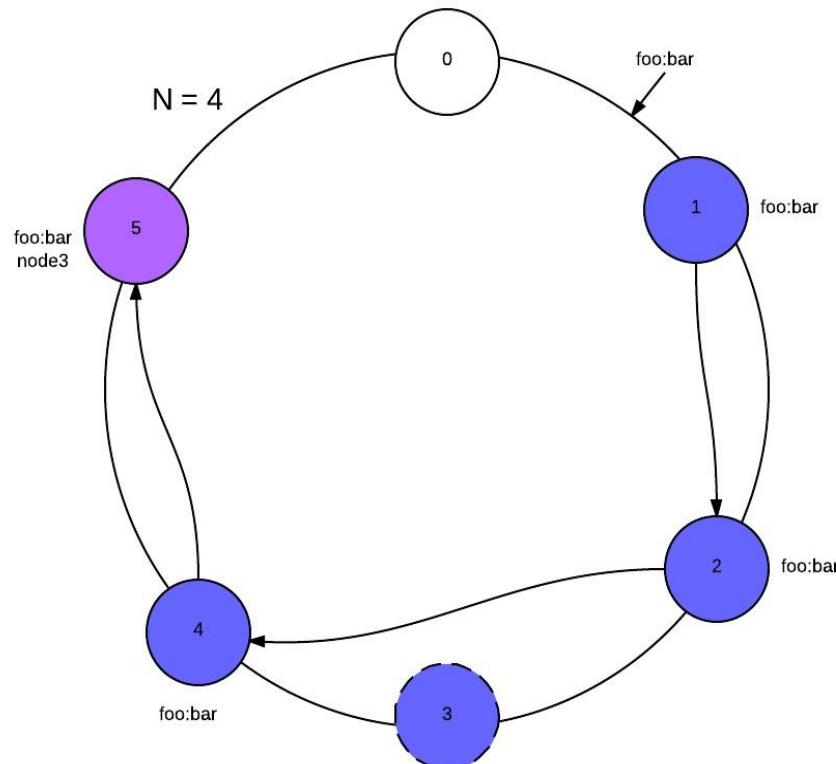
| Problem                            | Technique  | Advantage  |
|------------------------------------|--|--|
| Partitioning                       | Consistent Hashing                                     | Incremental scalability  |
| High Availability for writes       | Vector clocks with reconciliation 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 |

# Quasi-Quorums

---

- get() and put() driven by two parameters
  - R: minimum number of replicas to read from
  - W: minimum number of replicas to write to
- if  $R+W > N$ , we have a quorum system!
  - ... but latency dictated by slowest replica
- What if we want to execute a put() but the number of available replicas in the next node is less than  $w-1$ ?

# Sloppy Quorum and Hinted Handoff



- Use next N healthy nodes for read/write
- Data tagged with the node it should go to  
(notified later if temporarily unavailable)
- Transfer the data to the node when it becomes available
- **Always writeable!**

# Techniques Used by Dynamo

| 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 |

# Handling Permanent Failures

---

Replica synchronization:

- Node synchronizes with another node
- A Merkle Tree is used to detect inconsistency and minimize the data that needs to be transferred
  - leaves are hash of objects
  - parents are hash of children

# Membership and Failure Detection

---

- Gossip-based protocol propagates membership change and maintains an eventually consistent view
- Seeds are used to prevent partitioned ring
- Use timeout to for failure discovery