CE 528 Cloud Computing

Lecture 11: Cloud Database
Spring 2026

Prof. Yigong Hu

BOSTON UNIVERSITY

Slides courtesy of Ata Turk and Doug Terry

Administrivia

Second Milestone Demo in next Wednesday

- Upload the video+slide before the class
- The rubric is updated on the course website

Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

Amazon

SOSP'07

Amazon's Requirement

Serve tens of millions of customers at peak using tens of thousands of servers across multiple regions and DCs (all growing)

- Amazon Service Oriented Arch (SOA)
  - hundreds of services (some stateless, some stateful),
  - highly decentralized, loosely coupled, talking via APIs

Example of Service Oriented Architecture

Amazon’s Requirement

Serve tens of millions of customers at peak using tens of thousands of servers across multiple regions and DCs (all growing)

- Amazon Service Oriented Arch (SOA)
  - hundreds of services (some stateless, some stateful),
  - highly decentralized, loosely coupled, talking via APIs
- Need to be **always available**, e.g.
  - Always be able to add items to shopping cart even if disk are failing or data centerns are being destroyed by tornados, …
- Large Scale
- Strict Latency requirement

For Stateful Services

Need storage technologies that are always available & scalable

- S3/Simple Storage Service is one – (all should check it out)
- Dynamo is another that we'll cover today

Most service needs are very simple:

- Access data using only primary key, relatively small data (<1MB)
- Can relax consistency requirements if it means more availability
    - Don't need to be "always consistent"; consistency depend on app

Why not use a regular RDBMS?

- Expensive (Requires expensive HW, trained ops staff)
- Designed to solve more complex problem, difficult to scale
- Difficult to make 99.9% latency guarantees

System Interface/Operations/ Query Model

Simple read & write ops to a uniquely id'ed data item

No op spans multiple data items

No need for a relational schema

Get(key)

- Returns object, or list of objects & context

Put(key, context, object)

- New version of object
- Context meta-data obtained from previous read

Dynamo Overview

Dynamo: Replicated DHT with consistency management

- Consistent hashing
- Optimistic replication
- “Sloppy quorum”
- Anti-entropy mechanisms
- Object versioning

Goals

Highly available

- Resilient to failures

Scalable

- Can easily add nodes & the system will scale to handle more data/requests

Flexible, enable app find best config based on its own tradeoff bw:

- Availability
- Consistency
- Cost-effectiveness
- Latency
- Scalability

But provide enough abstraction so each team do not have to solve the same problem independently

DynamoDB Interface

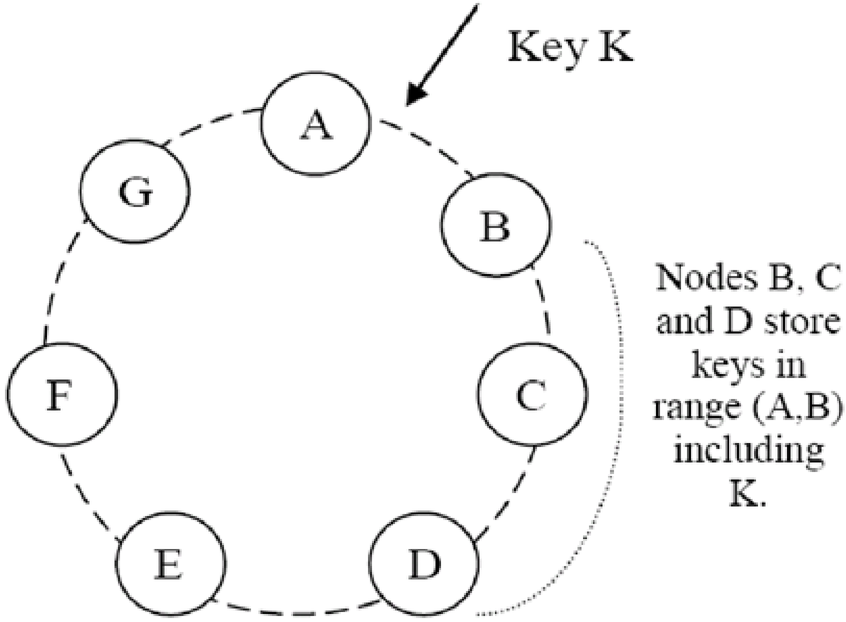
Operation	Functionality
PutItem	Insert a new item or replace an existing item with a new item
UpdateItem	Updates an existing item or adds a new item to the table if it doesn't exist
DeleteItem	Delete a single item from the table based on the primary key
GetItem	Returns a set of attributes for the item for the given primary key

Data Partitioning Algorithm

How to distribute data to nodes, so we can scale easily?

Map key space on a ring (Consistent Hashing)

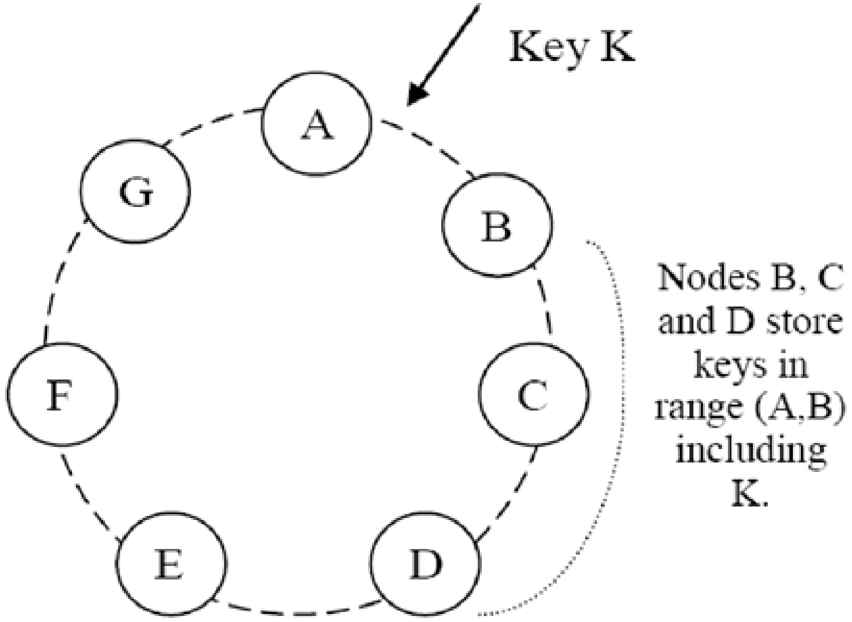
- MD5 Hash of Key to get a 128 bit identifier
- Output of hash function is treated as a “ring” in modulo
- Each node has a position in the ring
- All data whose key hash falls between a node and previous node is owned by the former (coordinator)
- Data is replicated on the owner node & on the following N-1 nodes on the ring
- Replication is asynchronous



Data Partitioning Algorithm

Map key space on a ring (Consistent Hashing)

- Adding/removing a node only affects neighboring nodes
- Enough information to route directly
- Virtual Nodes for load balancing/supporting heterogeneity:
  Each node is responsible from multiple points distributed around the ring



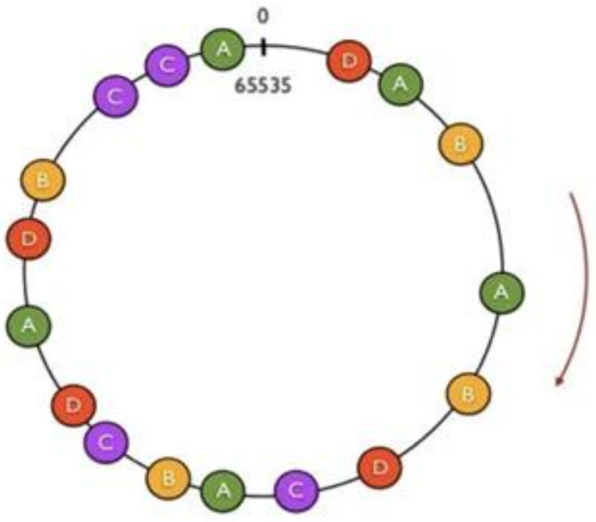
Advantage of using virtual nodes

If a node is unavailable, disperse load across the remaining available nodes.

When a node becomes available, accepts load from each of the other available nodes.

Allow heterogeneity among nodes:

- Number of virtual nodes / physical node ratio is determined based on node capacity, accounting for heterogeneity in the physical infrastructure



Replication

Each data item is replicated at N hosts.

- N is configured by application

Virtual node hashed to is called "coordinator"

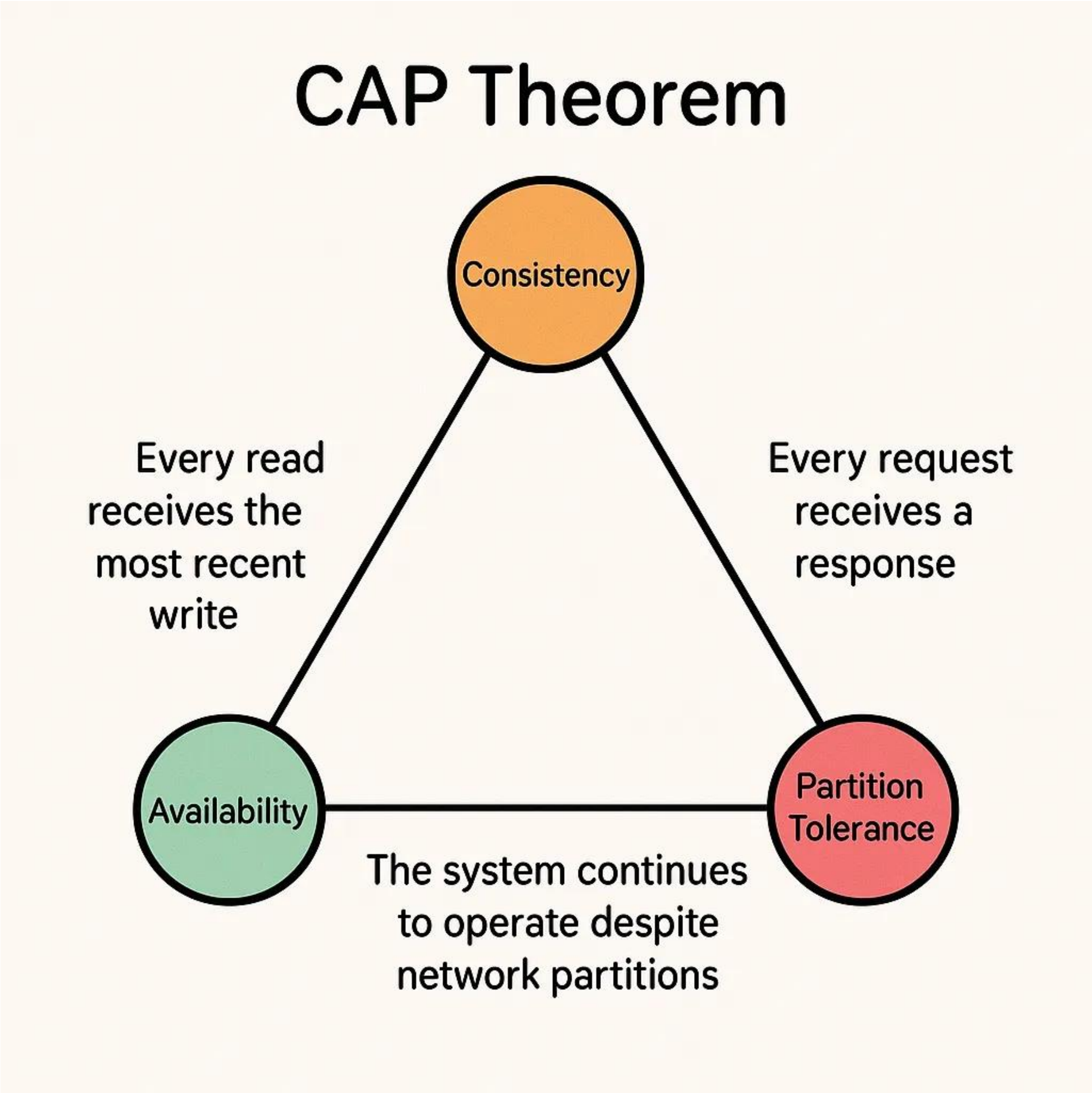
- Replicates keys to the N-1 successor nodes

"preference list": nodes responsible for storing a particular key

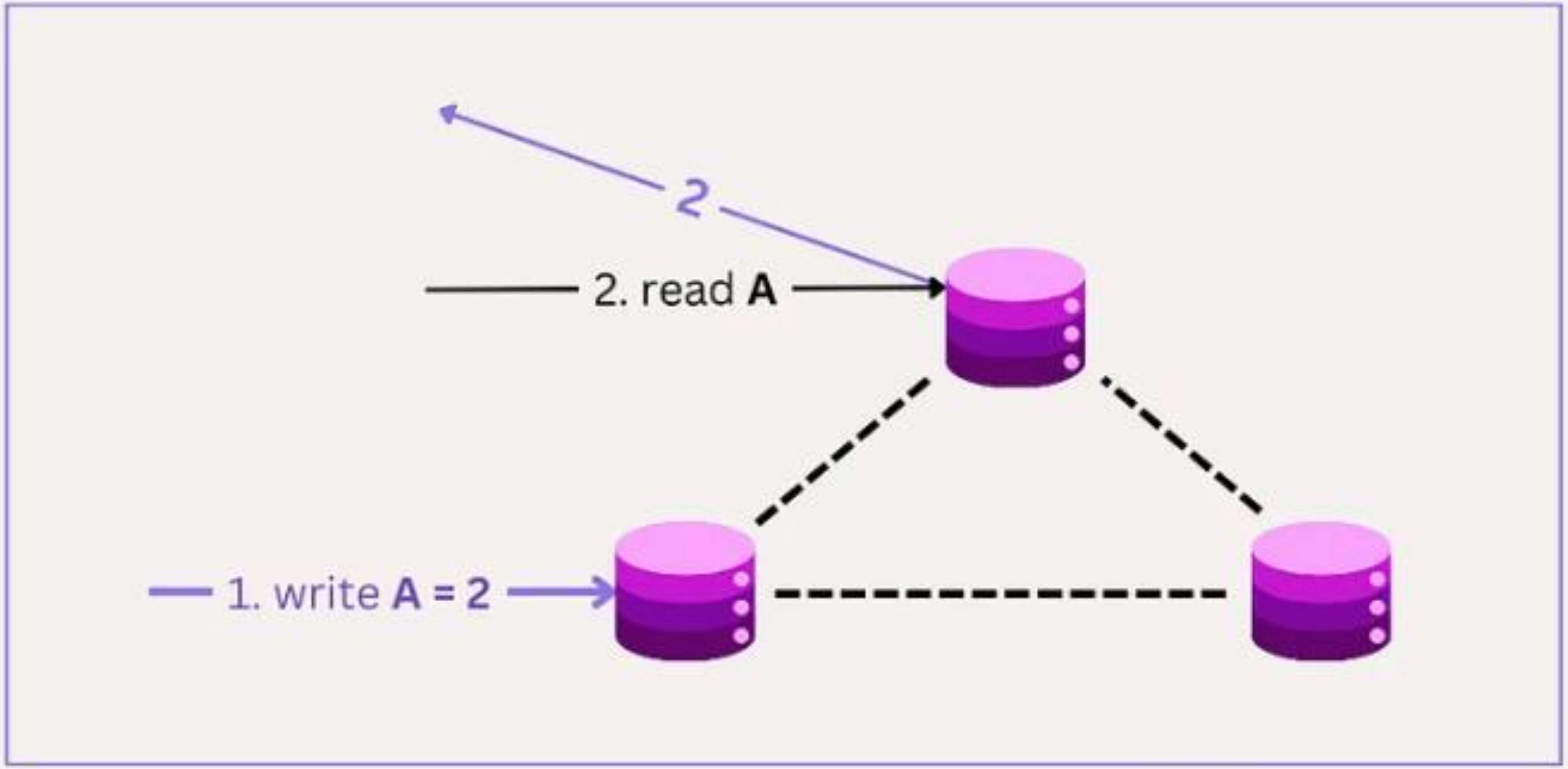
Replication is asynchronous

Can replicate to multiple DCs to protect against DC failures

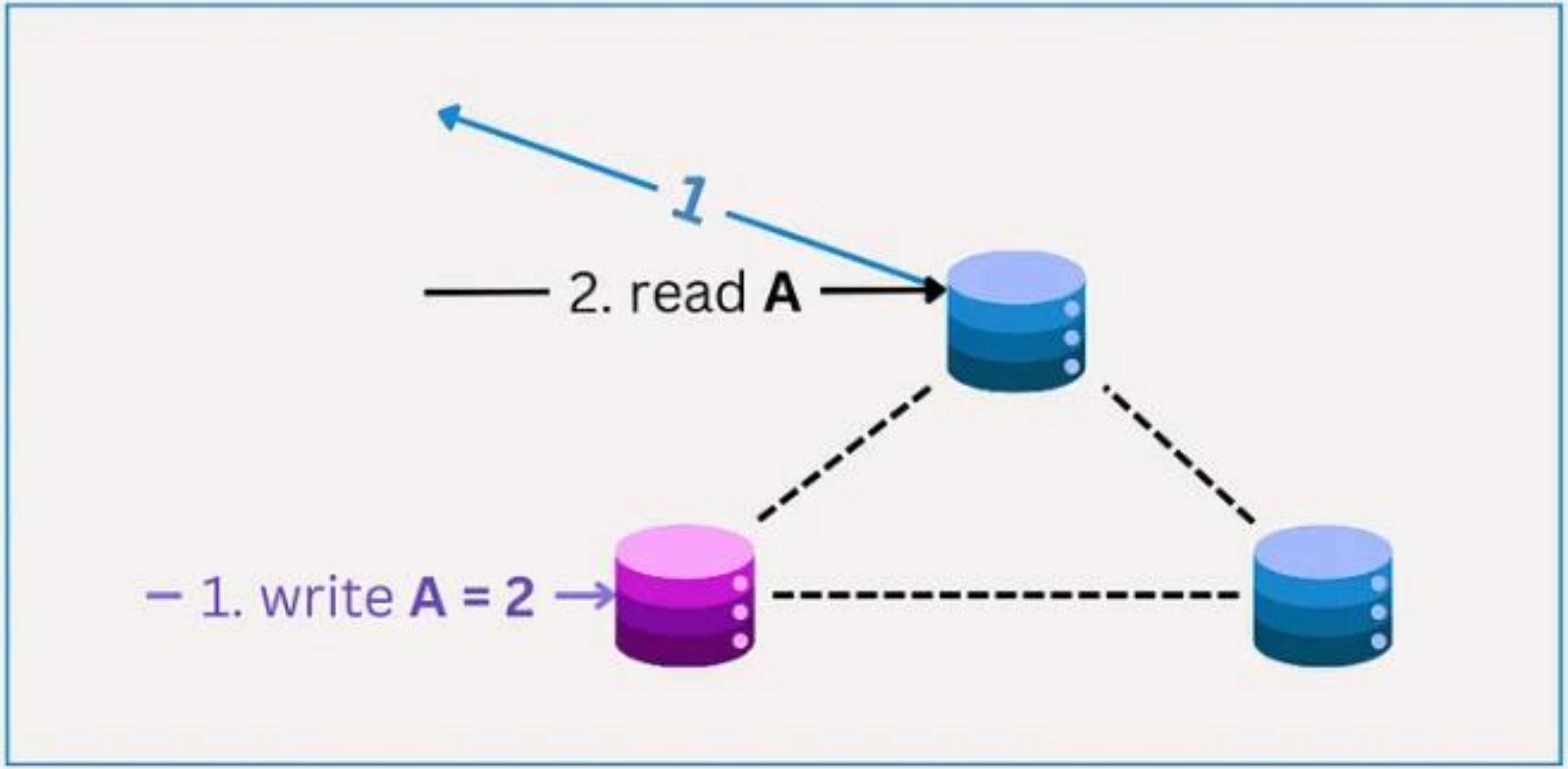
CAP Theorem


CAP Theorem
Consistency
Every read receives the most recent write
Every request receives a response
Availability
Partition Tolerance
The system continues to operate despite network partitions

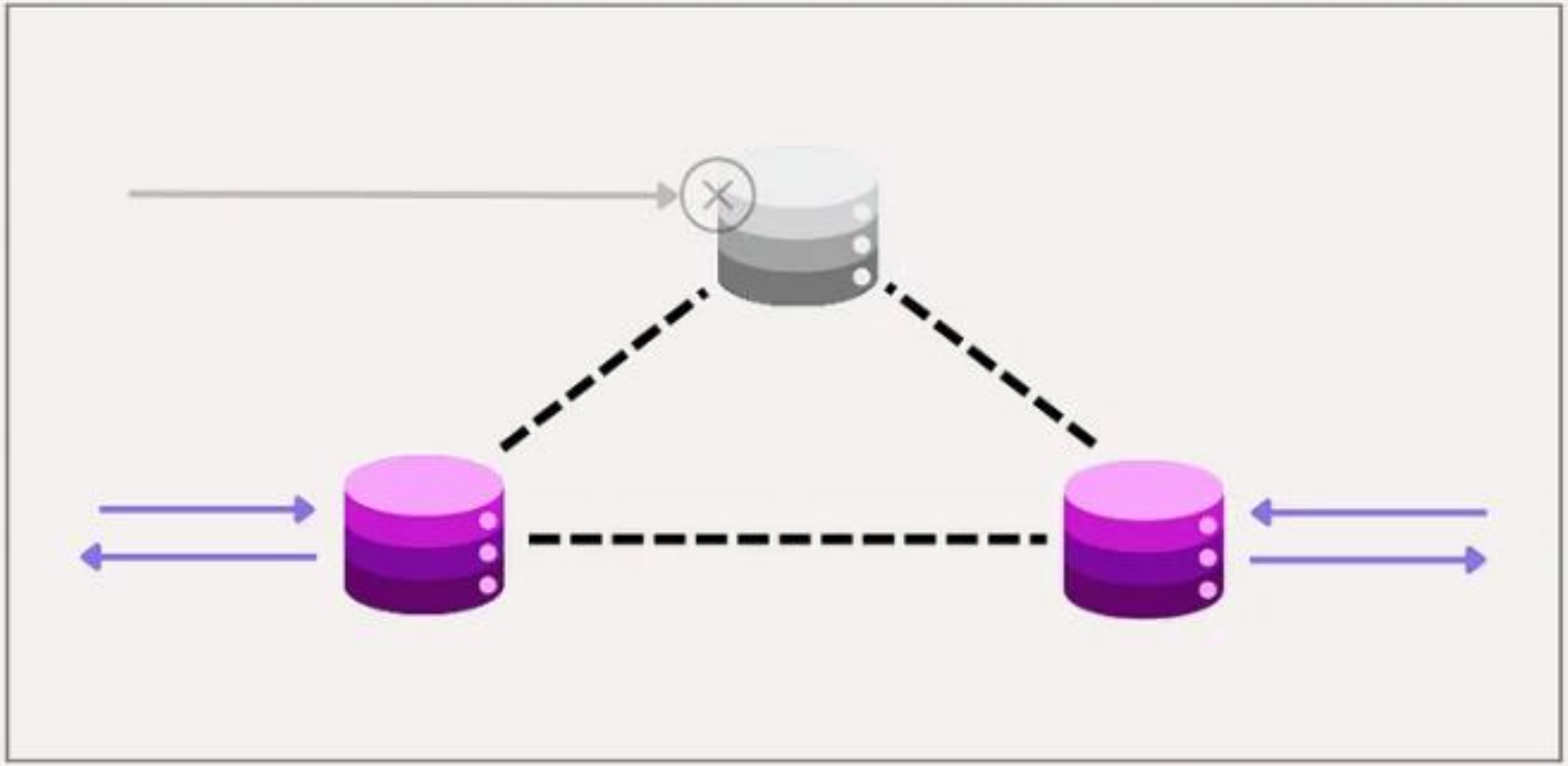
Consistency



Availability



Partition Tolerance



Consistency, Availability & Partitioning Tolerance

CAP Theorem: A distributed database system can only provide two of three guarantee

- Prove: contradiction

In other word: when dealing with possibility of network failures, strong consistency & high data availability cannot be achieved simultaneously

- We want to be always available => we cannot always have strong consistency

Consistency, Availability & Partitioning Tolerance

Loss Consistency

- e.g. Imagine DC A & B are disconnected
- Cust A talks to DC A, writes to OBJ X, while
- Cust B talks to DC B and writes to OBJ X at the same time, two X versions?

Optimistic replication:

- reconcile replica versions eventually
- when to reconcile, who reconciles

Dynamo:

- When to reconcile: during reads (always writable)
- Who reconciles:
  - data store: e.g. last write wins
  - application: e.g. merge the carts/tweet history, ...

Sloppy Quorum

R & W are the min # of nodes that must successfully read & write

- Put succeeds if W nodes successfully write
- Get succeeds if R nodes successfully read
- Dynamo lets user/client to set N, R, W

Setting R + W > N yields a quorum-like system

- E.g. High write rate N=3, W=2, R=2
- E.g. High read rate N=3, W=3, R=1

But quorum approach can lead to reduced availability

- Server failures and network partitions

Use sloppy quorum

- all read and write operations are performed on the first N healthy nodes
- may not always be first N nodes encountered walking the consistent hashing ring

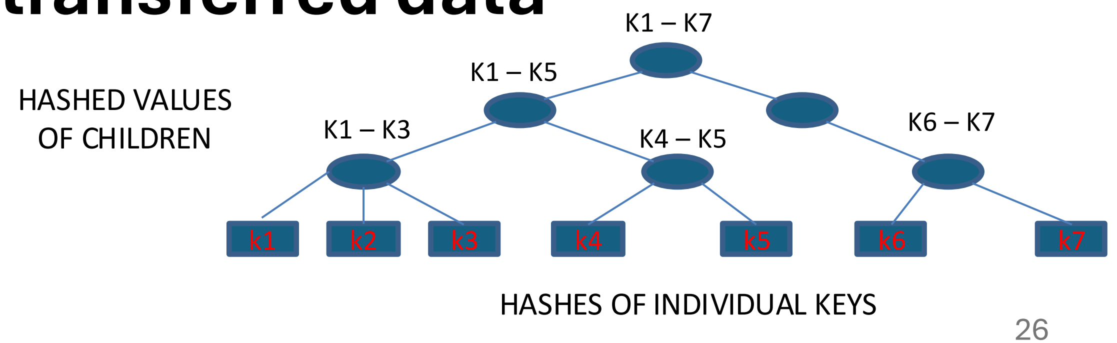
Sloopy Quorum

E.g. if B is temporarily down during a write (N=3), replicate to E with a metadata hint marking intended recipient

Nodes that receive hinted replicas keep them in a separate DB (scanned periodically).

Upon detecting B recovered, E will deliver replica to B.

Keep keys in a Merkle tree to detect inconsistencies between replicas faster and to minimize the amount of transferred data



Configuration

N	R	W	Application
3	2	2	Consistent, durable, interactive user state
N	1	N	High performance read engine
1	1	1	Distributed web cache

Consistency Management

Each put() creates new, immutable version

Dynamo tracks version history

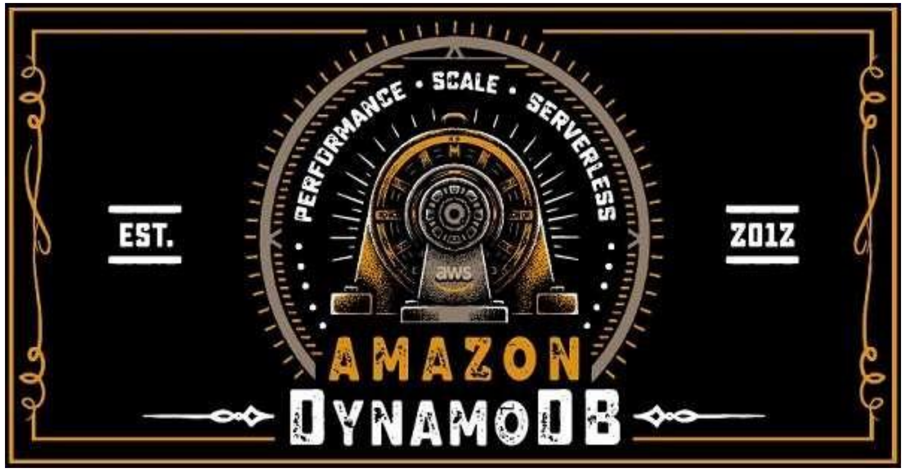
Automatic syntactic reconciliation

Application-level semantic reconciliation

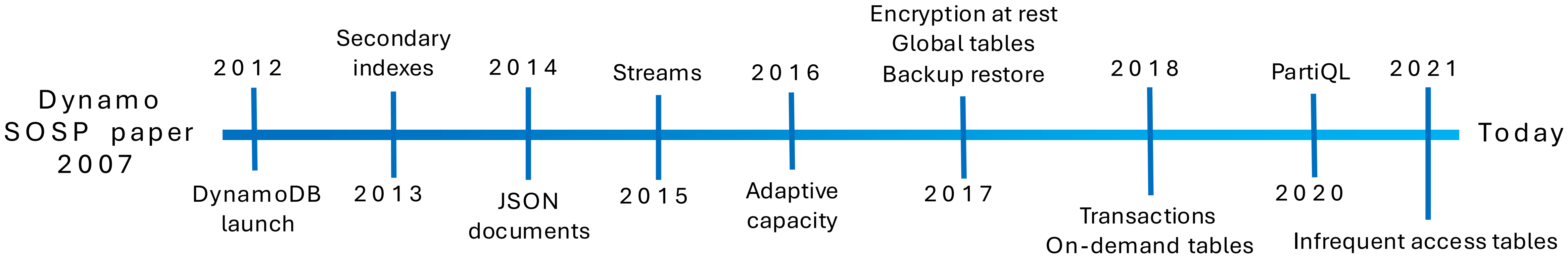
Vector clocks

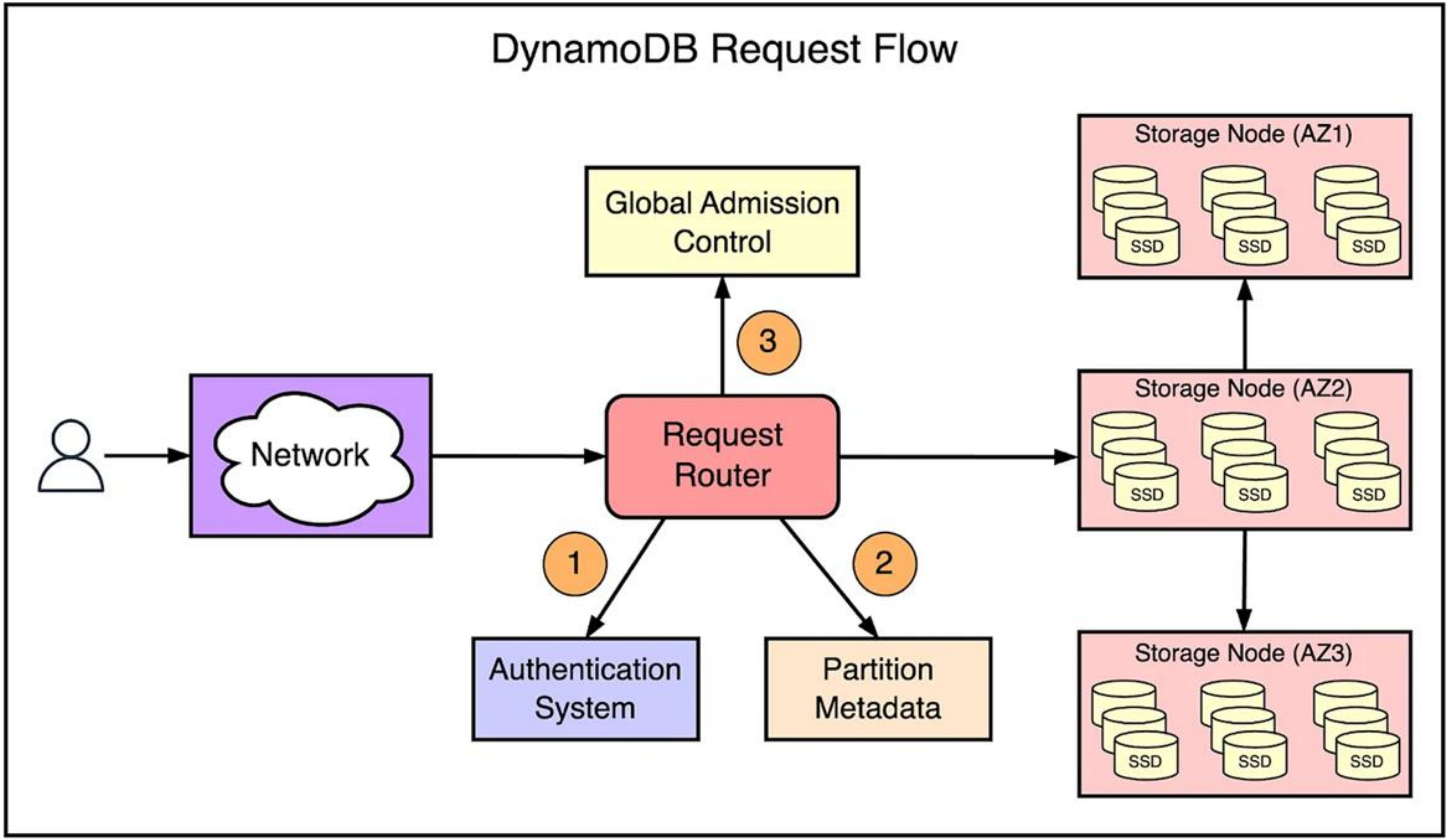
- logical timestamps
- capture causality
- detect conflicts

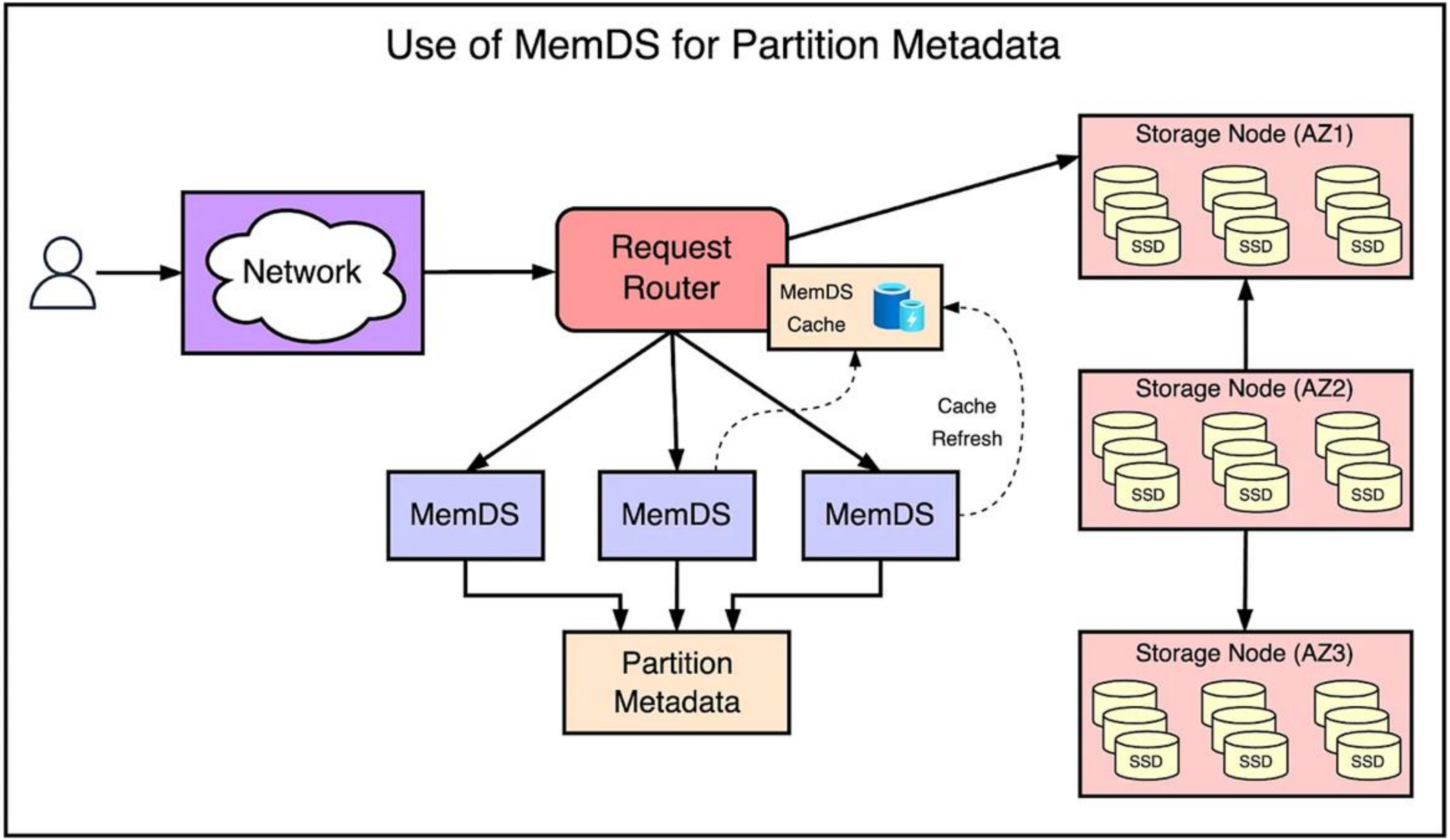
Amazon DynamoDB: A Scalable, Predictably Performant, Fully Managed NoSQL Database Service



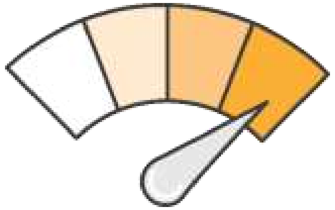
DynamoDB over the Years



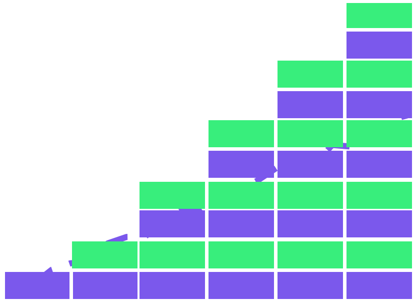

DynamoDB Request Flow
Global Admission Control
Storage Node (AZ1)
SSD
SSD
SSD
3
Network
Request Router
Storage Node (AZ2)
SSD
SSD
SSD
1
2
Authentication System
Partition Metadata
Storage Node (AZ3)
SSD
SSD
SSD


Use of MemDS for Partition Metadata
Network
Request Router
MemDS Cache
Cache Refresh
MemDS
MemDS
MemDS
Partition Metadata
Storage Node (AZ1)
SSD
SSD
SSD
Storage Node (AZ2)
SSD
SSD
SSD
Storage Node (AZ3)
SSD
SSD
SSD

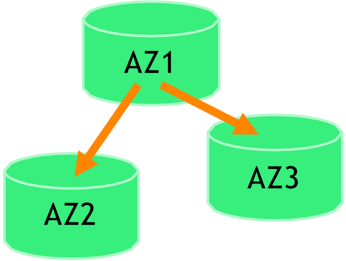
Key Aspects of DynamoDB



Predictability



Scalability



Availability

Consistency