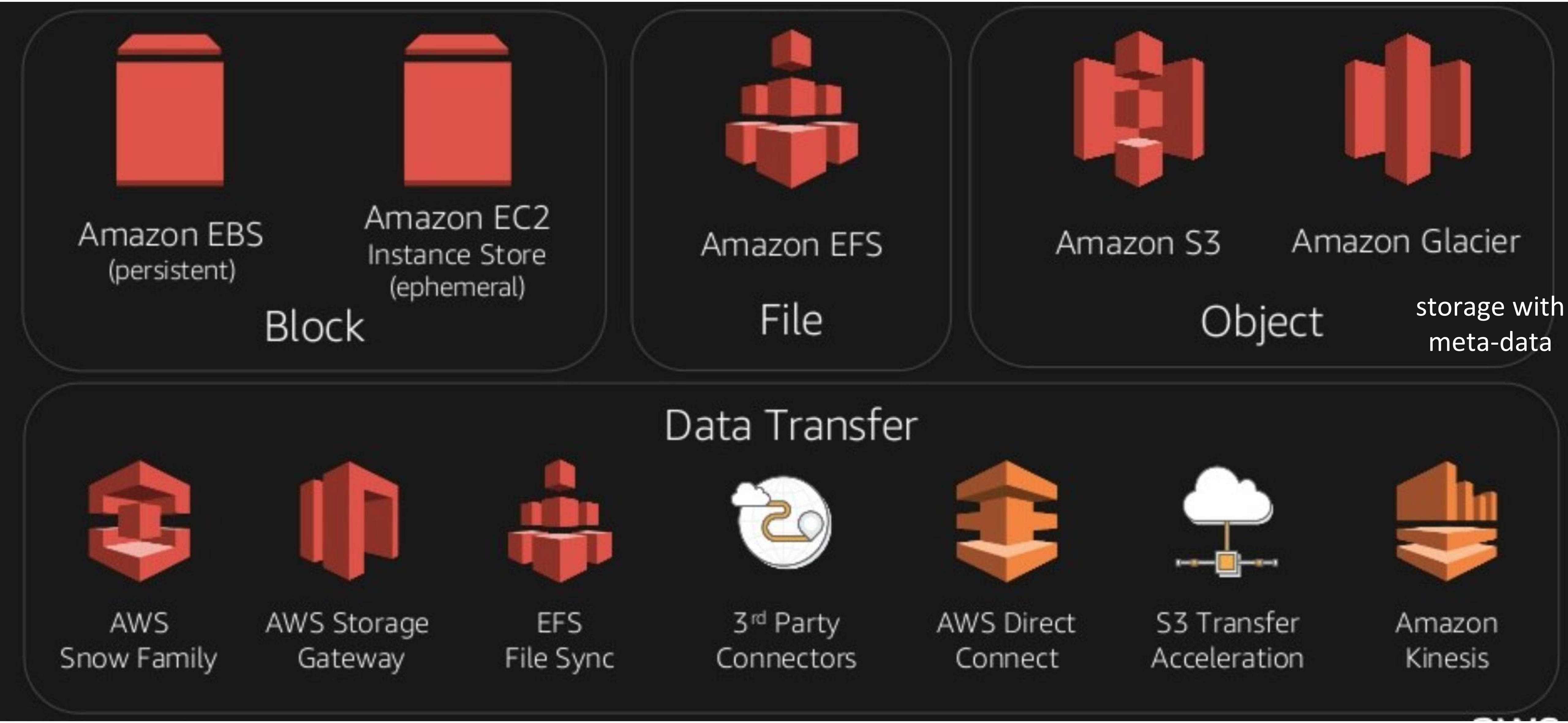
DSECL ZG 522: Big Data Systems

Session 12: Storage on Cloud (AWS examples)

Janardhanan PS

Professor

janardhanan.ps@wilp.bits-pilani.ac.in



storage with
meta-data

Topics for today

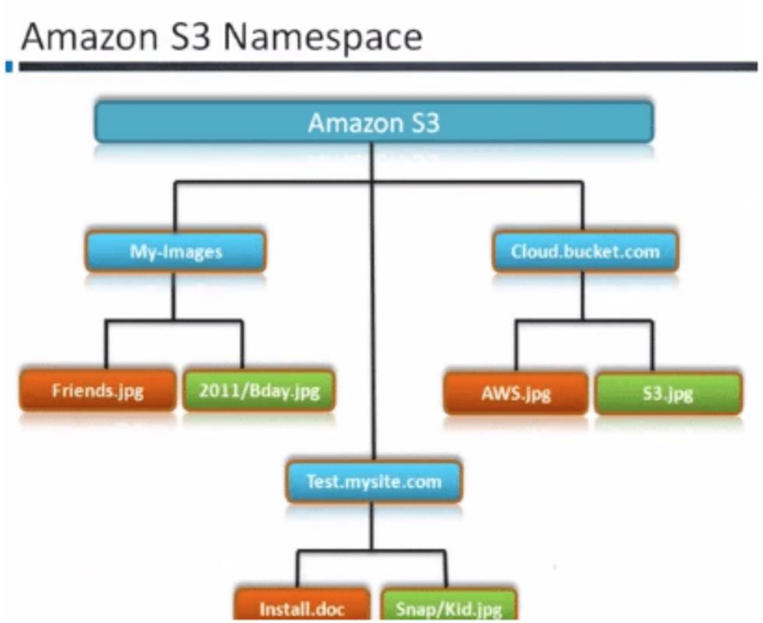
- **Object storage**
  - ✓ S3
- File storage
  - ✓ Elastic File System (EFS)
- Block storage
  - ✓ Elastic Block Storage (EBS)
- Databases
  - ✓ Key-value: DynamoDB

S3 - Object storage

- S3 is an object storage (Simple Storage Service)
  - ✓ which means it stores objects which are files with lot of meta-data associated
- A bucket is the container for objects
- An S3 account can have hundreds of buckets and a bucket can have hundreds of objects
- A bucket can also have folders to organise objects
- An object
  - ✓ can be 1 byte to 5TB
  - ✓ is uniquely identified by a developer assigned key and a URL
  - ✓ has an ACL to control who can access from anywhere - not necessarily from within AWS
  - ✓ supports versioning and “eventual consistency” across multiple reads / writes
  - ✓ is partitioned and replicated

Example

- 3 Buckets with folders inside buckets for grouping objects
- Can be access via public URL:
  - ✓ Option 1: bucketname.s3.amazonaws.com/objectname
  - ✓ Option 2: s3.amazon.aws.com/bucketname/objectname

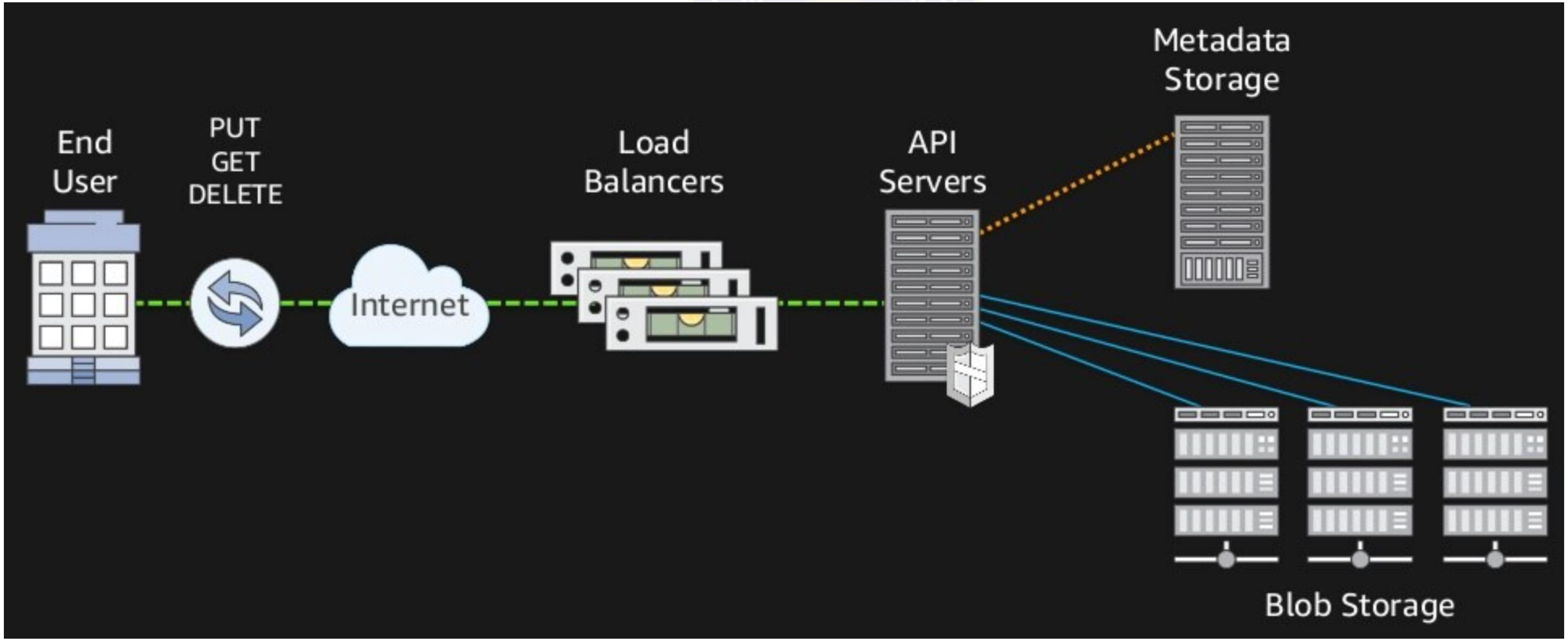


Consistency model

- S3 - Follows AP design wrt CAP Theorem
- So when a user uploads an object, it is replicated but readers can access inconsistent replicas and all replicas are "eventually consistent"
- Option for reduced redundancy for lower cost - so writes / updates may not be durable but cost is reduced significantly
  - ✓ RRS: https://aws.amazon.com/blogs/aws/new-amazon-s3-reduced-redundancy-storage-rrs/
- Some observations:
  - ✓ User uploads object but a reader can get "key does not exist"
  - ✓ An object may not appear in a listing of a bucket immediately
  - ✓ Old data may be returned if a write is not propagated by then
  - ✓ A deleted object may be seen in a listing for some time

Architecture

- 20+ Regions with multiple Availability Zones (AZ) per region
  - ✓ AZs are physically isolated connected over low latency network
- 60+ AZ with each AZ is upto 8 DCs
- Data can be stored in at least 3 physically separated AZ within a Region for HA
- Private network connections across AZ and DCs for low latency

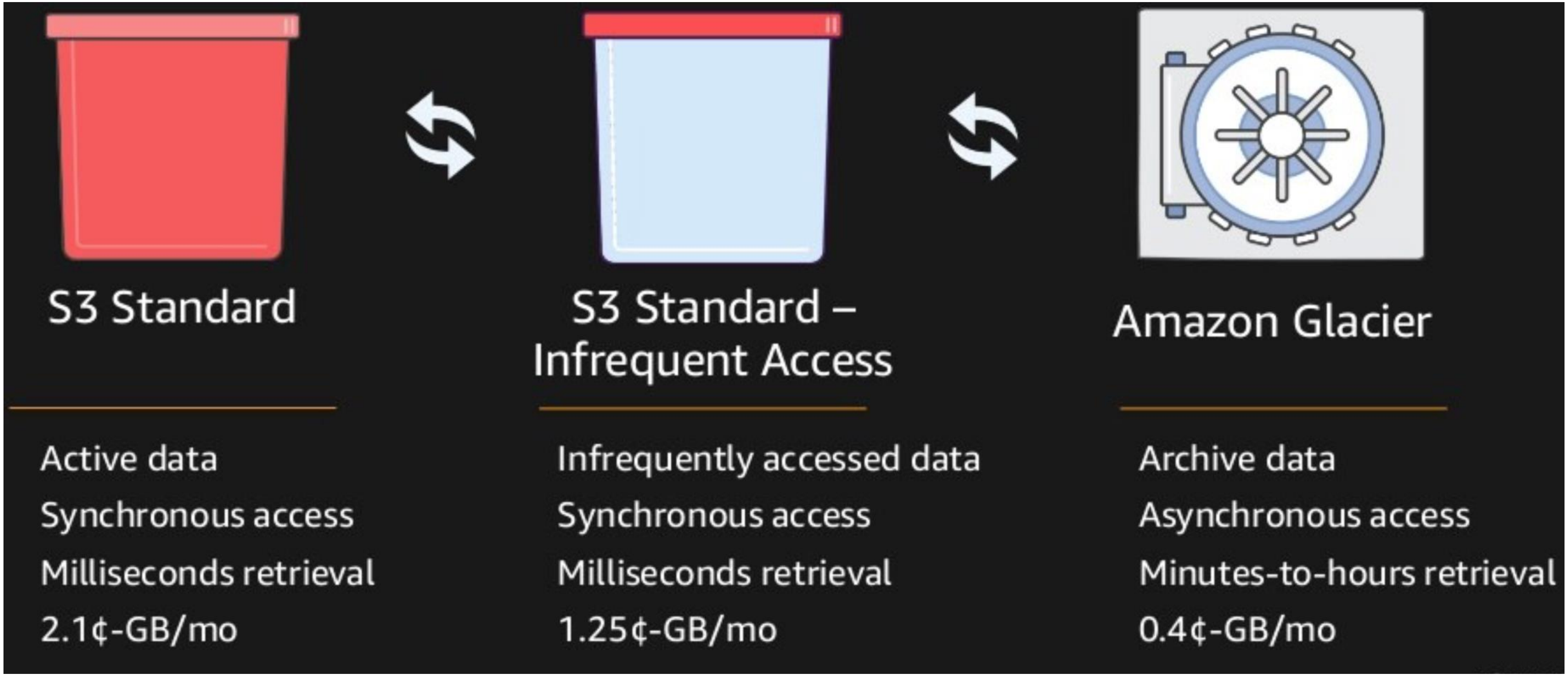


https://aws.amazon.com/about-aws/global-infrastructure/regions_az/

Features

- Storage tiering
- Object Lambda
- Access points associated with ACLs for customised client access
- Batch mode operations

Storage classes

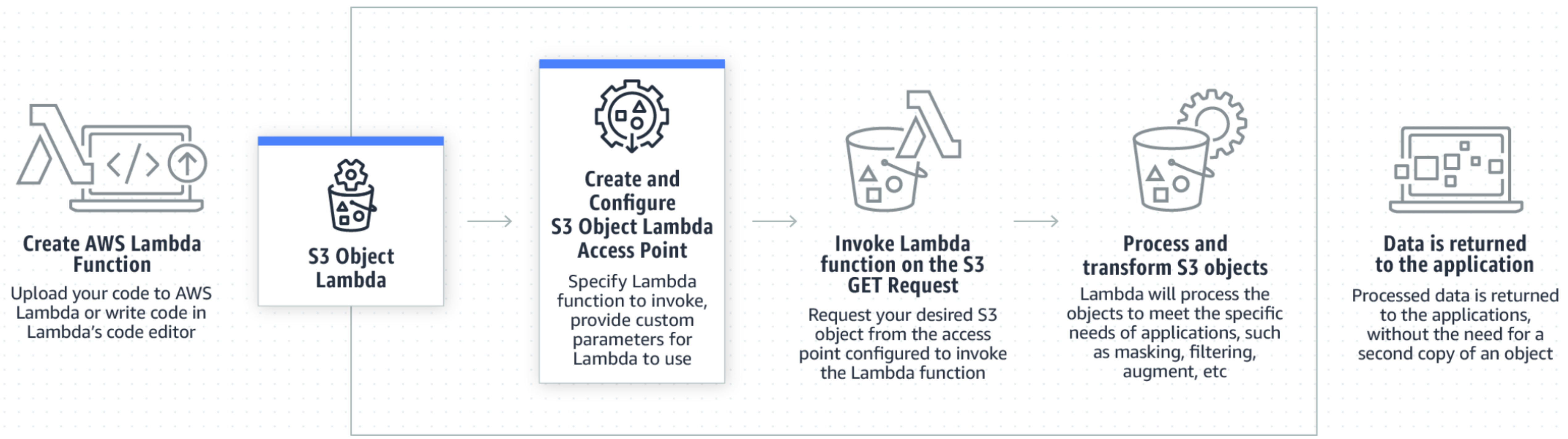


S3 Standard	S3 Standard – Infrequent Access	Amazon Glacier
Active data	Infrequently accessed data	Archive data
Synchronous access	Synchronous access	Asynchronous access
Milliseconds retrieval	Milliseconds retrieval	Minutes-to-hours retrieval
2.1¢-GB/mo	1.25¢-GB/mo	0.4¢-GB/mo

S3 Outposts also enables to store data on customer premises for faster local access or local residency requirements

Object Lambda

Remember Function-as-a-Service ? What if you could move the function closer to Data ?



Create AWS Lambda Function

Upload your code to AWS Lambda or write code in Lambda's code editor

S3 Object Lambda

Create and Configure S3 Object Lambda Access Point

Specify Lambda function to invoke, provide custom parameters for Lambda to use

Invoke Lambda function on the S3 GET Request

Request your desired S3 object from the access point configured to invoke the Lambda function

Process and transform S3 objects

Lambda will process the objects to meet the specific needs of applications, such as masking, filtering, augment, etc

Data is returned to the application

Processed data is returned to the applications, without the need for a second copy of an object

Object Lambda

- Add code to S3 GET request for data processing

```
      GET /my-image.jpg HTTP/1.1
Host: bucket.s3.<Region>.amazonaws.com
Date: Mon, 3 Oct 2016 22:32:00 GMT
Authorization: authorization string
```

- Register an access point with Lambda transformation function

```
PUT /v20180820/accesspointforobjectlambda/name HTTP/1.1
x-amz-account-id: AccountId
<?xml version="1.0" encoding="UTF-8"?>
<CreateAccessPointForObjectLambdaRequest xmlns="http://awss3control.amazonaws.com/doc/2018-08-20/">
  <Configuration>
    <AllowedFeatures>
      <AllowedFeature>string</AllowedFeature>
    </AllowedFeatures>
    <CloudWatchMetricsEnabled>boolean</CloudWatchMetricsEnabled>
    <SupportingAccessPoint>string</SupportingAccessPoint>
    <TransformationConfigurations>
      <TransformationConfiguration>
        <Actions>
          <Action>string</Action>
        </Actions>
        <ContentTransformation>
          <AwsLambda>
            <FunctionArn>string</FunctionArn>
            <FunctionPayload>string</FunctionPayload>
          </AwsLambda>
        </ContentTransformation>
      </TransformationConfiguration>
    </TransformationConfigurations>
  </Configuration>
</CreateAccessPointForObjectLambdaRequest>
```

Use cases

- Backup and restore for Cloud as well as on-prem data
- Disaster recovery with cross region replication
- Data archival
- Cloud based applications with scalable storage access from anywhere
- Create data lakes for big data analytics

Topics for today

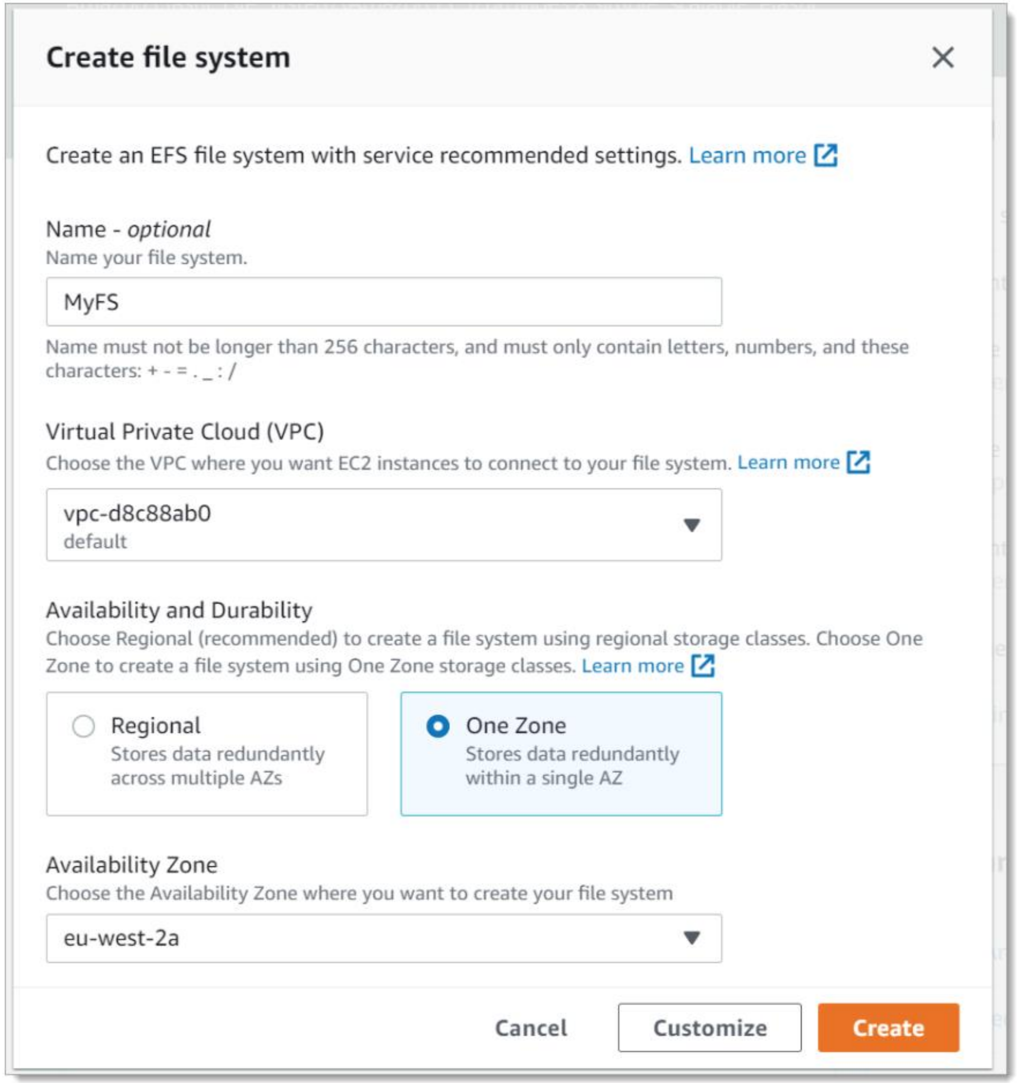
- Object storage
  - ✓ S3
- **File storage**
  - ✓ Elastic File System (EFS)
- Block storage
  - ✓ Elastic Block Storage (EBS)
- Databases
  - ✓ Key-value: DynamoDB

Overview

- File system built using SSD storage that can be accessed from multiple Cloud instances or even customer premises servers
  - ✓ Supports NFS protocol
- Applications can simply attach the file system within VPC - so not much change to get elastic scalability and IOPS
  - ✓ 10GB/sec, 500K IOPS
  - ✓ Multiple NFS clients can attach EFS
- Multiple storage classes

EFS Storage classes

- Standard - with replication to 3+ AZs
  - ✓ EFS Standard and EFS Standard - Infrequent Access
- One Zone - replication only within one AZ at lower cost
  - ✓ EFS One Zone and EFS One Zone - Infrequent Access
  - ✓ 80% files typically in this category
- Setup lifecycle policies to move data based on age to Infrequent Access (IA) class



Create file system

Create an EFS file system with service recommended settings. Learn more

Name - *optional*
Name your file system.
MyFS
Name must not be longer than 256 characters, and must only contain letters, numbers, and these characters: + - = . _ : /

Virtual Private Cloud (VPC)
Choose the VPC where you want EC2 instances to connect to your file system. Learn more
vpc-d8c88ab0
default

Availability and Durability
Choose Regional (recommended) to create a file system using regional storage classes. Choose One Zone to create a file system using One Zone storage classes. Learn more

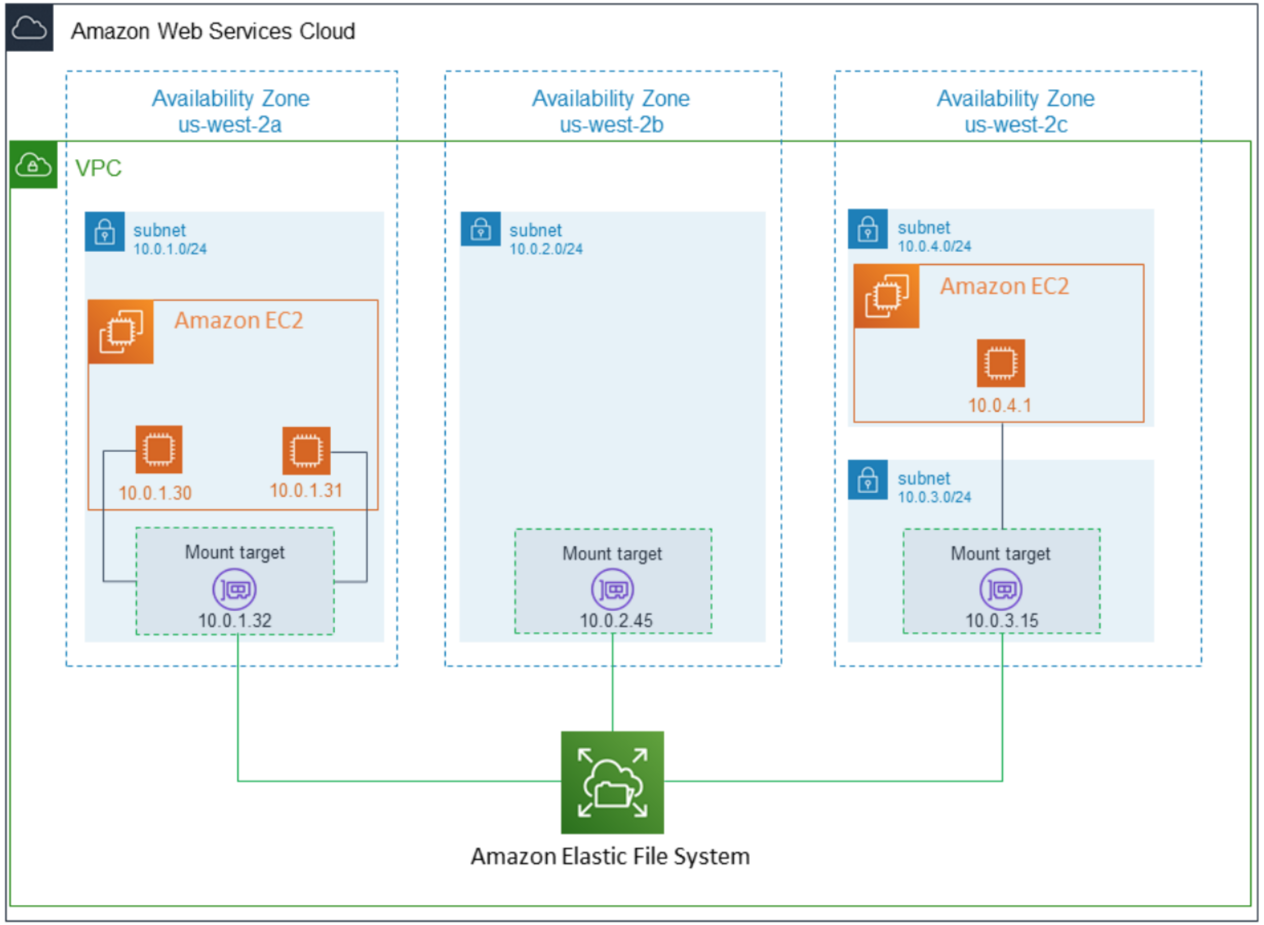
○ Regional
Stores data redundantly across multiple AZs

● One Zone
Stores data redundantly within a single AZ

Availability Zone
Choose the Availability Zone where you want to create your file system
eu-west-2a

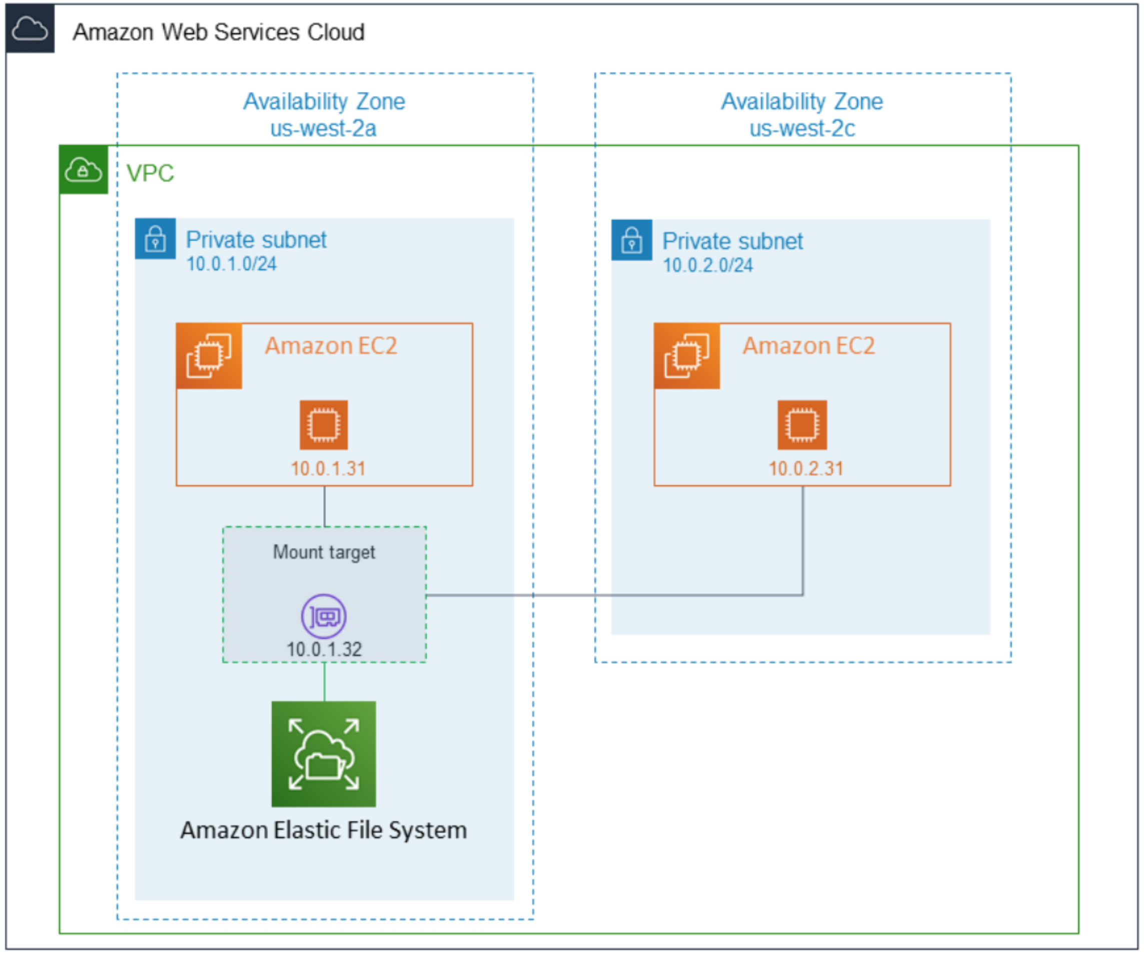
Standard storage class

- A VPC has 3 AZs
- Each AZ has a mount target
- Typically should access EFS from a mount target within same AZ for performance and cost
- Can create a mount target in one of the subnets within an AZ



One Zone storage class

- Single mount target in 1 AZ
- So instance in another AZ has to pay for data access cost



What if you have to connect customer premise ?

Read: https://docs.aws.amazon.com/efs/latest/ug/how-it-works.html

Topics for today

- Object storage
  - ✓ S3
- File storage
  - ✓ Elastic File System (EFS)
- **Block storage**
  - ✓ Elastic Block Storage (EBS)
- Databases
  - ✓ Key-value: DynamoDB

Overview

- Block storage for an EC2 instance that can be attached and detached anytime
- Provides a volume as a collection of network attached blocks that are exposed as disks
- Depending on performance / cost one can opt for HDD-backed or SSD-backed volumes
- EBS volumes are durable and replicated within AZ
- Can’t move a volume to another AZ without snapshot

EBS or EFS

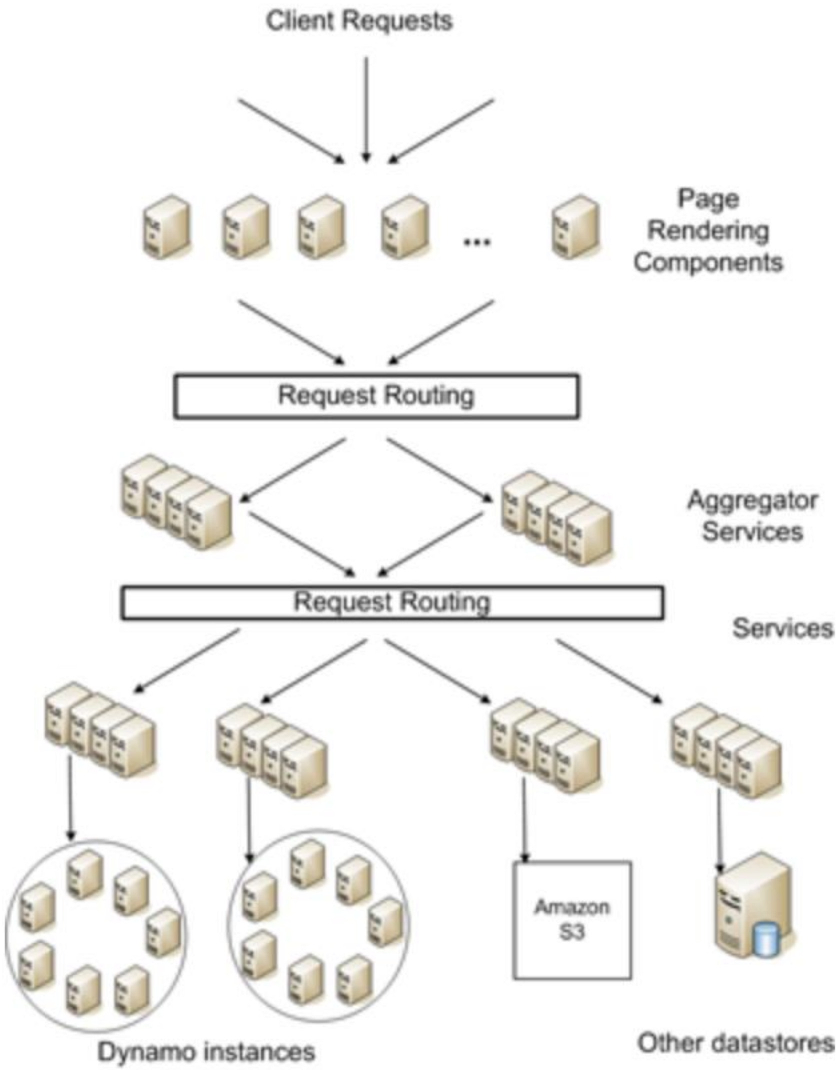
- EBS
  - ✓ Can be only accessed by one instance at a time
  - ✓ Steady predictable performance for a single instance use case
  - ✓ upto: 4GBps, 64TB, 260K IOPS, sub-millisec latency per volume
  - ✓ I/O intensive applications, e.g. relational databases, OLAP engines
  - ✓ Multiple performance / cost options
  - ✓ Cheaper than EFS per volume but only for one instance - so effectively more expensive
- EFS
  - ✓ Like a distributed multi-user network file system
  - ✓ Scalable access across many users with decent performance
  - ✓ Costs more per GB but is shared by multiple instances - so turns out cheaper for cost sensitive shared storage applications

Topics for today

- Object storage
  - ✓ S3
- File storage
  - ✓ Elastic File System (EFS)
- Block storage
  - ✓ Elastic Block Storage (EBS)
- **Databases**
  - ✓ Dynamo paper: https://www.allthingsdistributed.com/2007/10/amazons_dynamo.html (concepts map to Cassandra, DynamoDB etc.)
  - ✓ DynamoDB: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html

Applications and requirements

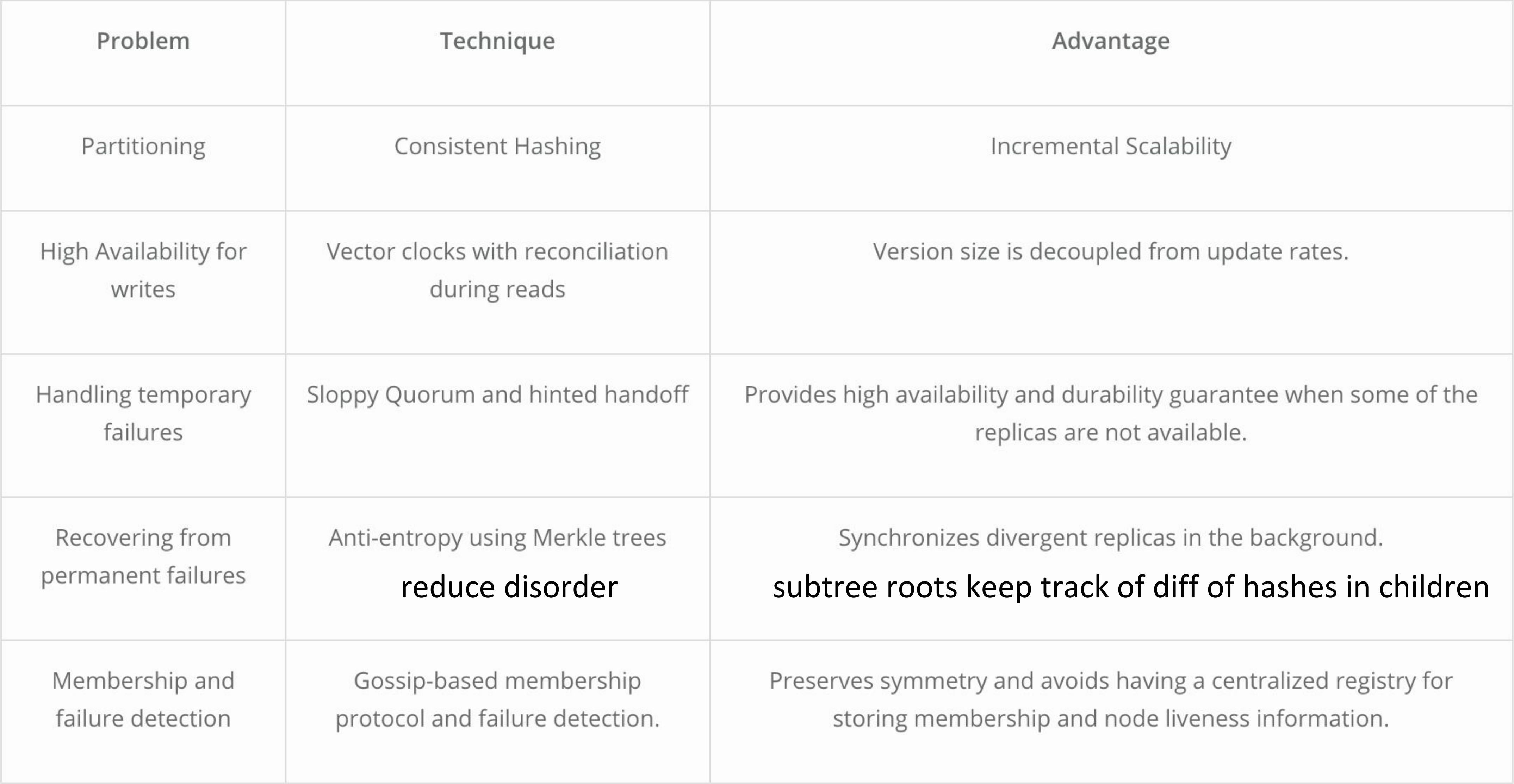
- Best seller lists, shopping carts, customer preferences, session management, sales rank, and product catalog
- Query: no relational queries, simple key based object retrieval with object size <1 MB
- ACID: Transactions reduce availability. No strict consistency requirement.
- Efficiency: Commodity hardware, strict low latency at 99.9th percentile, tradeoff performance, cost, availability, durability



Design considerations

- Highly available for writes, e.g. update shopping cart even on network partitions
- Complexity of conflict resolution is pushed to read
- Who resolves conflicts on read ?
  - ✓ If data store then only choice is “last write wins” or leave it to the application developer
- Incrementally add nodes and all nodes have symmetric role, like peers
- Heterogenous system - work is proportional to capability

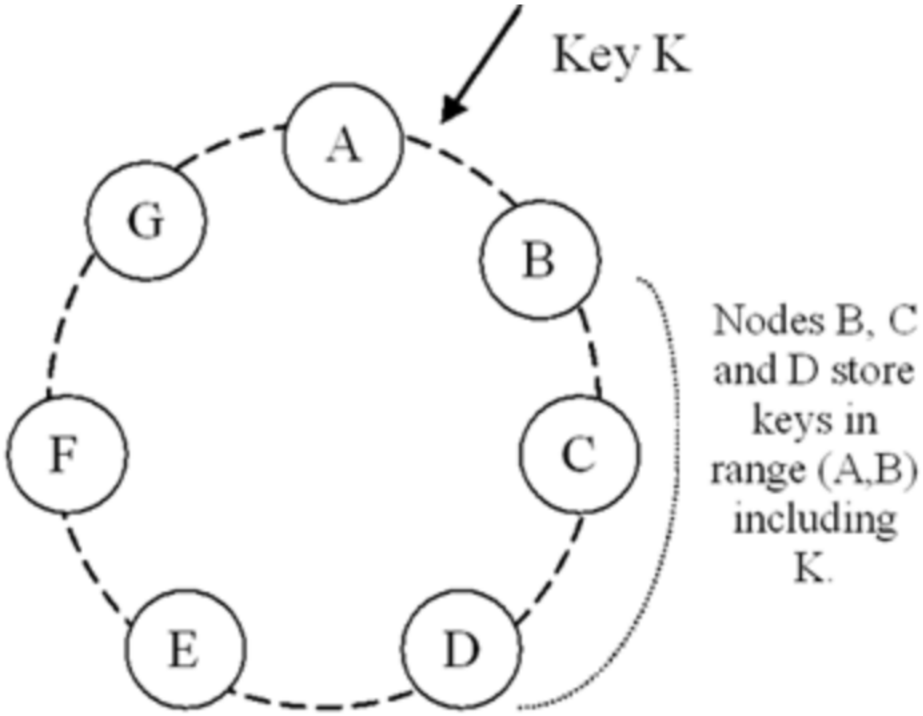
Architectural techniques



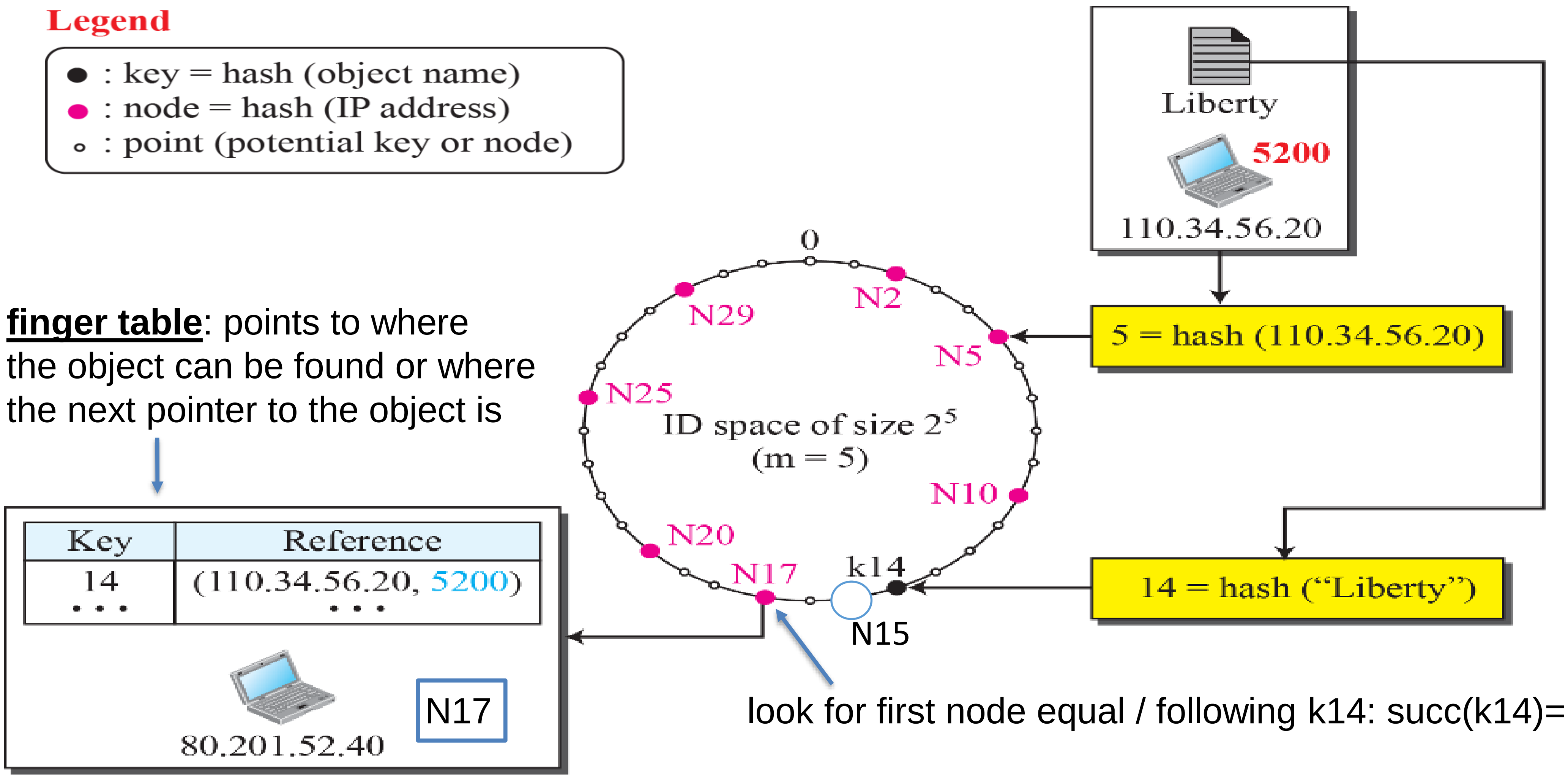
Problem	Technique	Advantage
Partitioning	Consistent Hashing	Incremental Scalability
High Availability for writes	Vector clocks with reconciliation during reads	Version size is decoupled from update rates.
Handling temporary failures	Sloppy Quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available.
Recovering from permanent failures	Anti-entropy using Merkle trees reduce disorder	Synchronizes divergent replicas in the background. subtree roots keep track of diff of hashes in children
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.

Object storage and interface

- Objects stored with associated keys
  - ✓ object, context = get(key)
  - ✓ put(key, context, object)
  - ✓ context encodes system meta-data along with versions
- Partitioning and replication
  - ✓ A variant of "consistent hashing" of the key is used to allocate storage nodes
  - ✓ Each data item is replicated on a configurable number of nodes



A Chord DHT Example



Legend

- ● : key = hash (object name)
- ● : node = hash (IP address)
- ∘ : point (potential key or node)

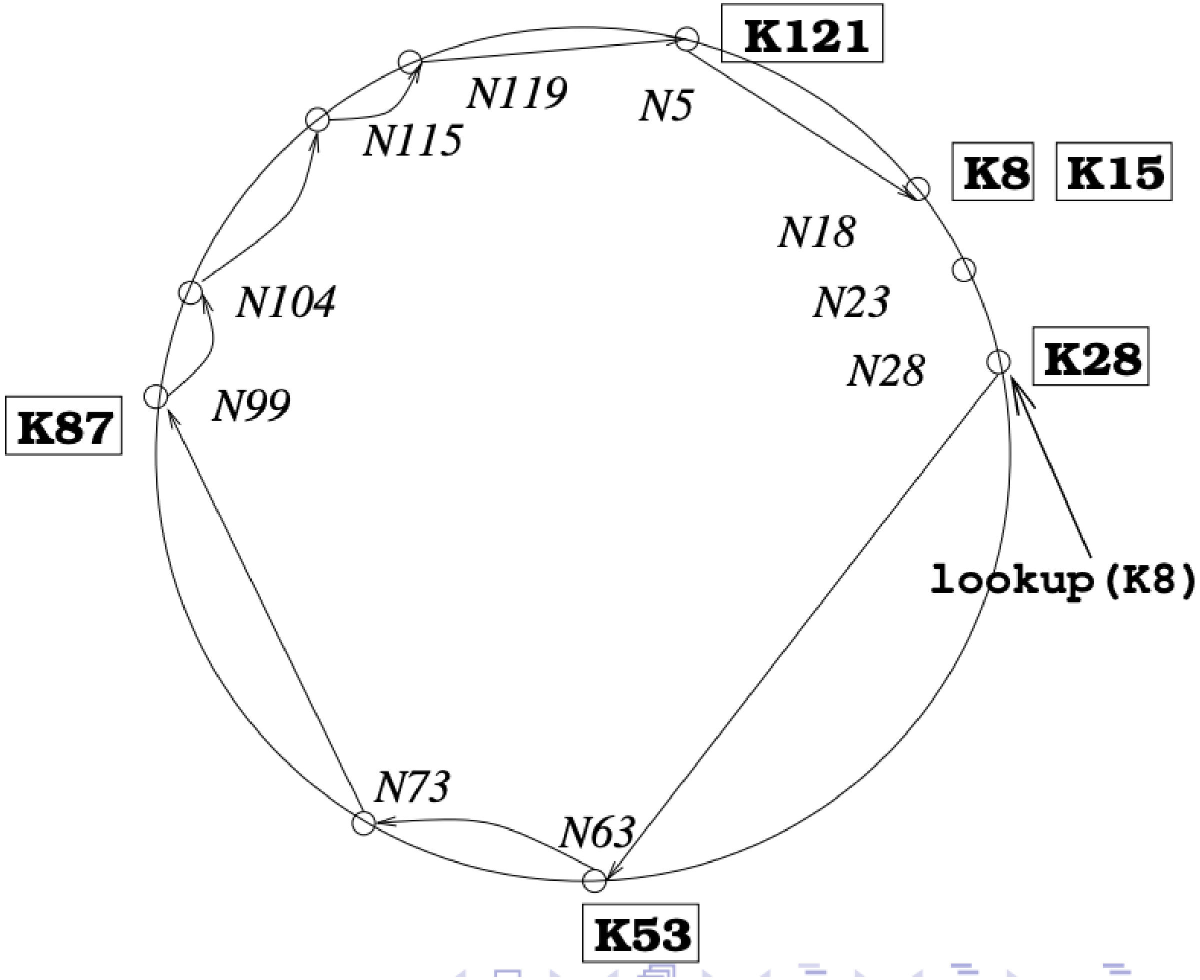
finger table: points to where the object can be found or where the next pointer to the object is

Key	Reference
14 . . .	(110.34.56.20, 5200) . . .

look for first node equal / following k14: succ(k14)=N17

Source: MGH Co.

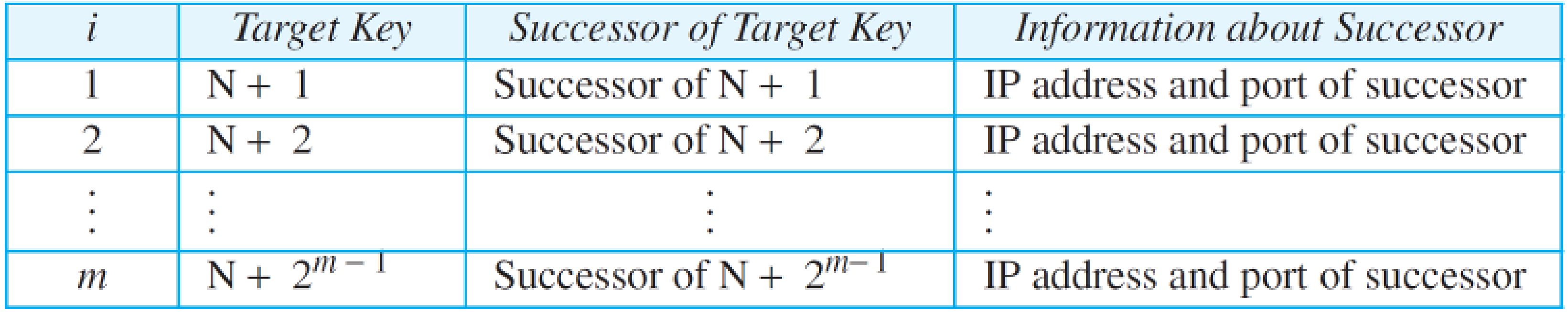
Linear search example



- Lookup(x) goes to a node
- Then keep following the successor link of nodes to check if object exists
- O(n) message hops
- Routing table in each node is O(1)

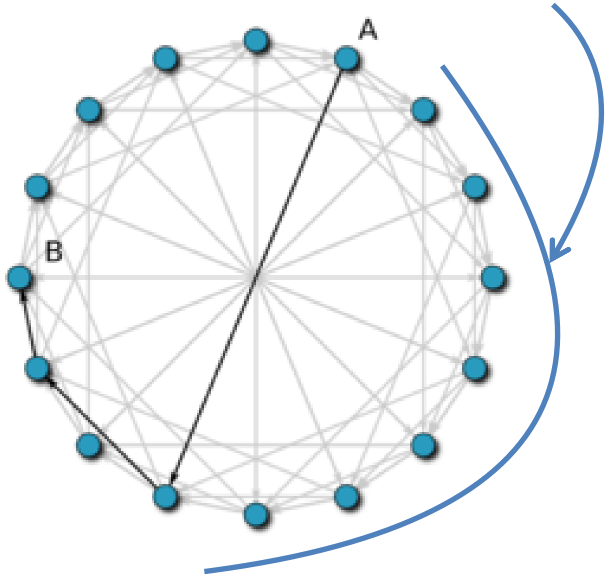
Source: MGH Co.

Chord Finger table for log search



i	*Target Key*	*Successor of Target Key*	*Information about Successor*
1	N + 1	Successor of N + 1	IP address and port of successor
2	N + 2	Successor of N + 2	IP address and port of successor
⋮	⋮	⋮	⋮
m	N + 2^{m-1}	Successor of N + 2^{m-1}	IP address and port of successor

skip searching these nodes

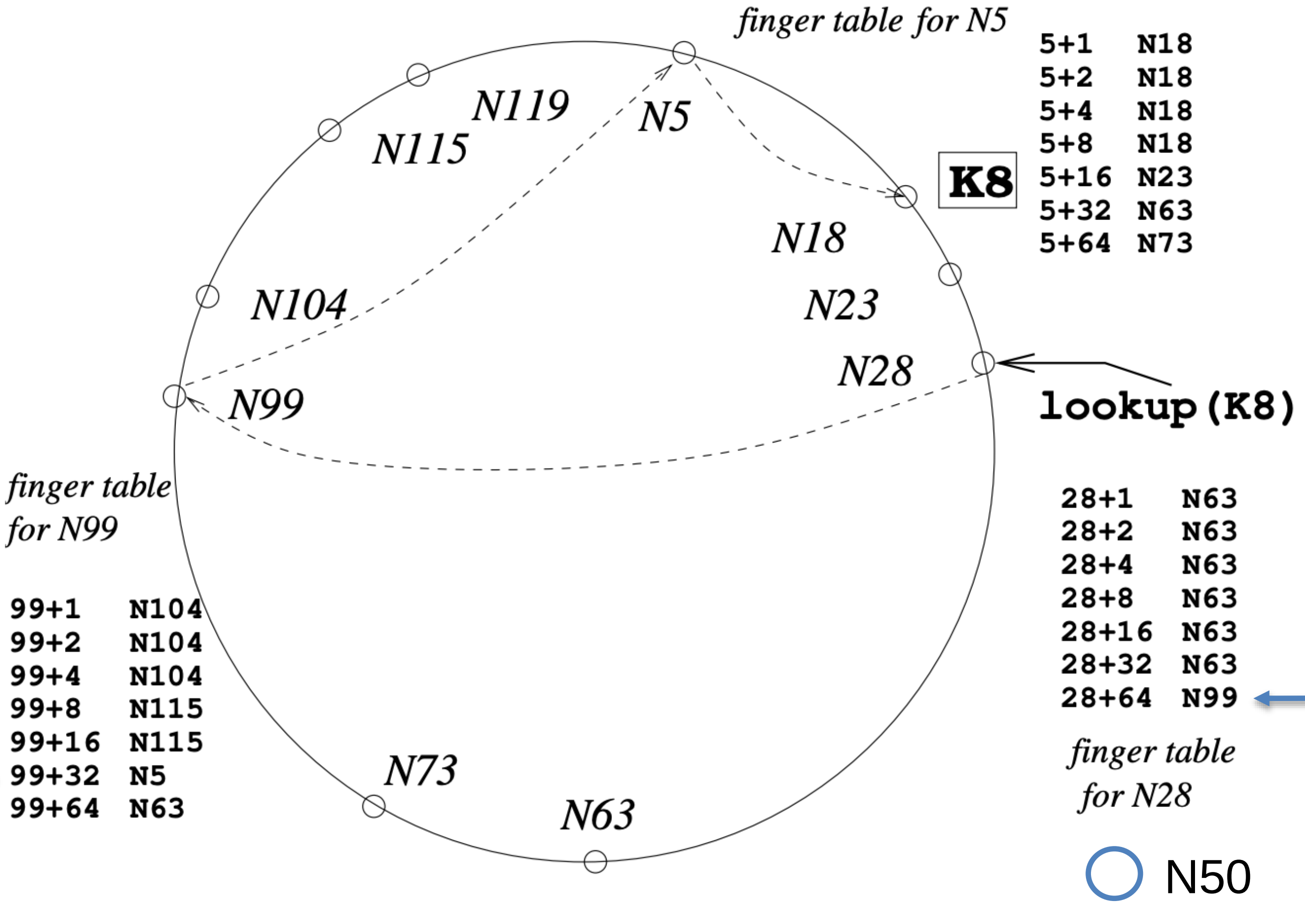


- Each node stores a routing table which helps to search with O(log n) message hops
- Space taken in routing table is O(m)
- When a node joins or leaves, the finger tables in some nodes need to be updated so that the change doesn't impact

https://pdos.csail.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf

Source: Wiki
Source: MGH Co.

Example



finger table for N5

5+1	N18
5+2	N18
5+4	N18
5+8	N18
5+16	N23
5+32	N63
5+64	N73

finger table for N99

99+1	N104
99+2	N104
99+4	N104
99+8	N115
99+16	N115
99+32	N5
99+64	N63

131 mod 128 = 3 —> N5

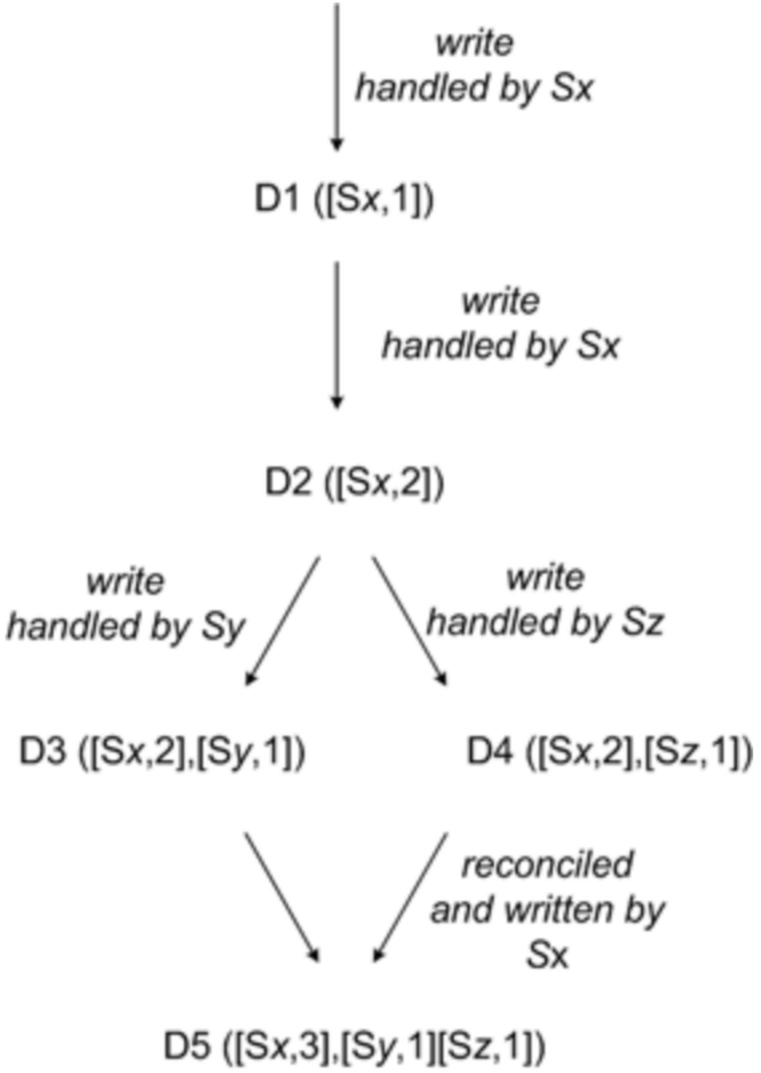
28+1	N63
28+2	N63
28+4	N63
28+8	N63
28+16	N63
28+32	N63
28+64	N99

finger table for N28

7-bit address
or 128 size key space

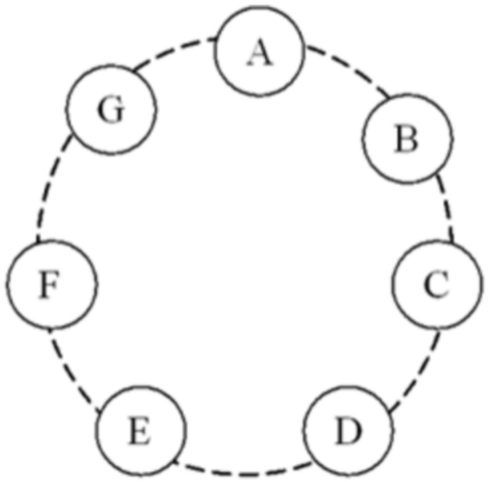
Consistency using versions

- Vector clocks associated with every write
- Every read operation can understand the causal order and reconcile multiple versions
- Vector clocks can grow over time
- Dynamo maintains a timestamp when last time a node updated a data item and deletes old nodes beyond a limit



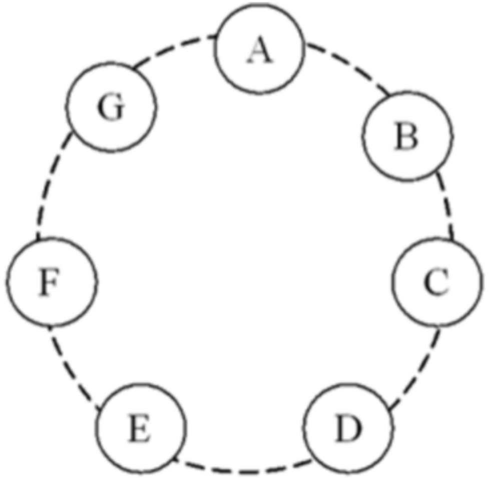
Handling faults - hinted handoff

- All reads and writes are performed by first N healthy nodes and not N nodes in exact sequence
  - ✓ This is called a “sloppy quorum”
  - ✓ Improves Availability when nodes fail
- Suppose A was supposed to receive a replica along with B and C but since A failed, it will go to D temporarily, along with a hint to D that A was the original choice
- Once A is up, D will send replica to A remove from itself
- Typically the key hash based replica nodes are kept across DCs to be highly durable
- Also typically ack for writes is kept > 1 so that at least 2 copies are written before ack to client



What about permanent failures of nodes ?

- Detect inconsistencies between replicas and solve it using minimum data transfers, e.g. is A is permanently down, how to make sure replicas are fixed on affected neighbors ?
- Use Merkel trees (hash trees) to track down subtrees where replicas have gone out of sync
  - ✓ Leaves are hashes of individual keys
  - ✓ Parents are hashes of children



Tweaking N, R, W for performance, durability, consistency, availability

- N - how many replicas total for a data item
- R - how many replicas to return read success
- W - how many replicas to return write success

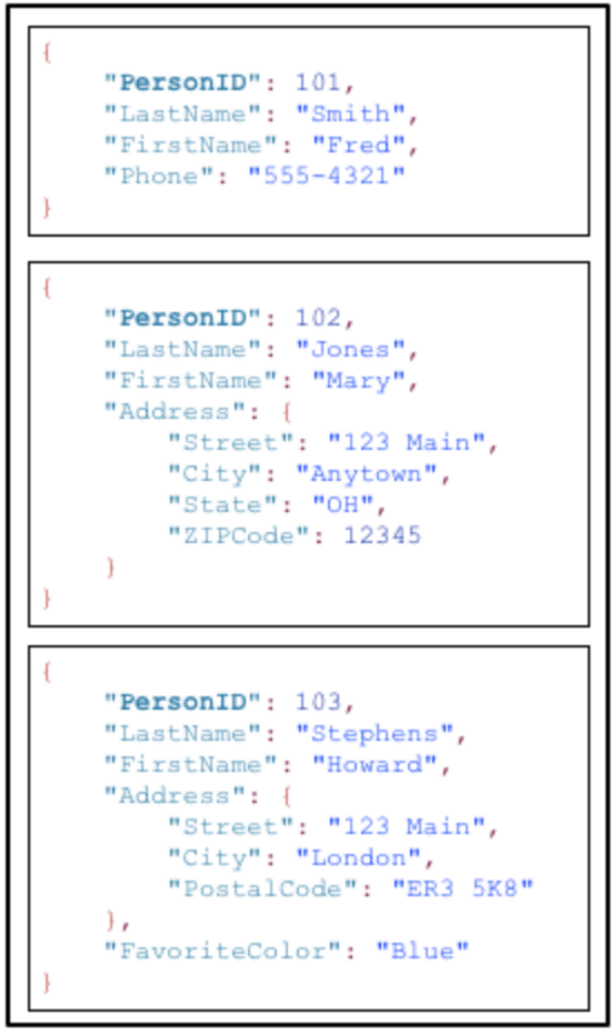
- W = 1: High performance writes
- W = N : Highly durable writes
- R = 1: High performance reads
- R = N: Highly consistent reads

- Typical config: (N, R, W) = (3, 2, 2)
- Hard engineering problem to use commodity heterogenous hardware and provide 99.9th percentile SLA on latency and accommodate slowest replicas for R and W targets

DynamoDB: Overview (1)

- NoSQL store that can store nested JSON documents in tables. Upto 32 levels of nesting.
- Partition key using consistent hashing
- Optional sorting key to sort within partition

People



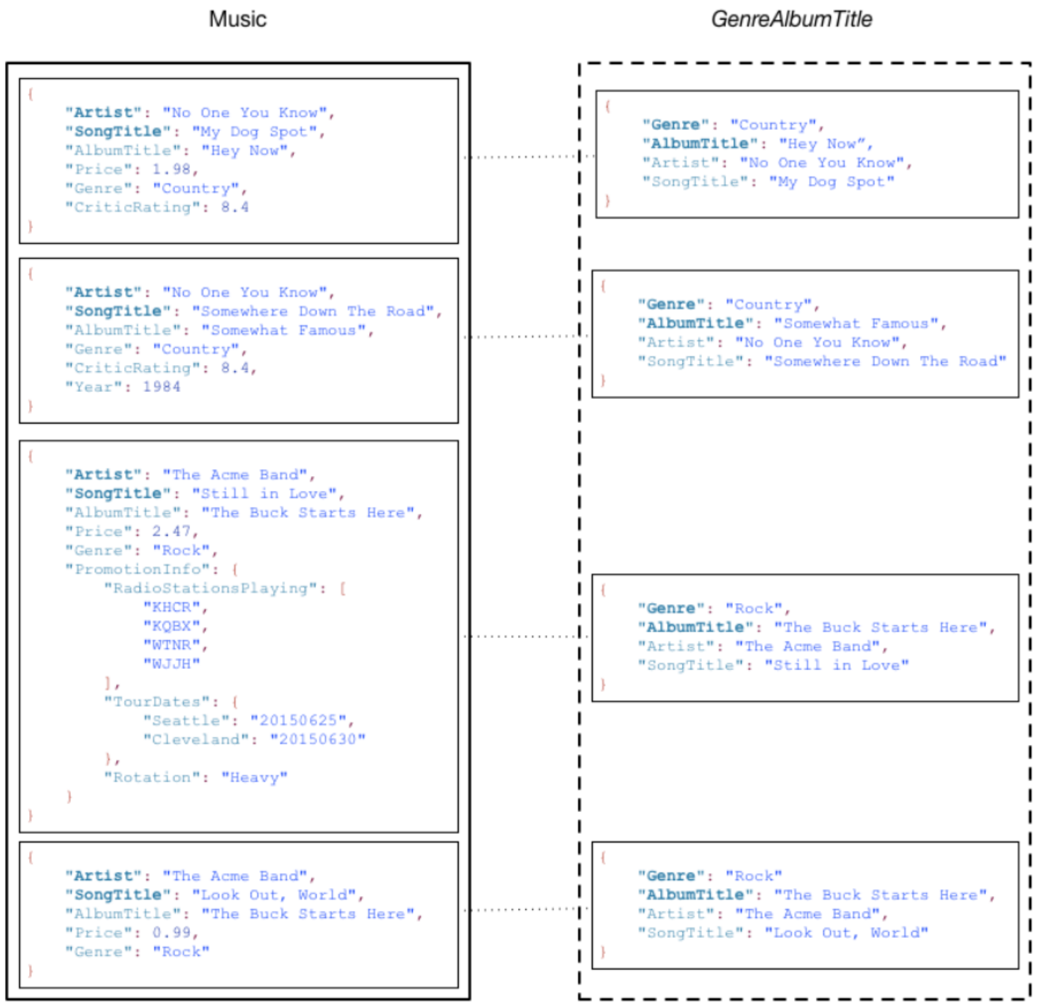
```
{
    "PersonID": 101,
    "LastName": "Smith",
    "FirstName": "Fred",
    "Phone": "555-4321"
}
```

```
{
    "PersonID": 102,
    "LastName": "Jones",
    "FirstName": "Mary",
    "Address": {
        "Street": "123 Main",
        "City": "Anytown",
        "State": "OH",
        "ZIPCode": 12345
    }
}
```

```
{
    "PersonID": 103,
    "LastName": "Stephens",
    "FirstName": "Howard",
    "Address": {
        "Street": "123 Main",
        "City": "London",
        "PostalCode": "ER3 5K8"
    },
    "FavoriteColor": "Blue"
}
```

DynamoDB: Overview (2)

- Secondary indices
  - ✓ Global: Different from partition and sort keys
  - ✓ Local: Same partition but different from sort key



```
{
    "Artist": "No One You Know",
    "SongTitle": "My Dog Spot",
    "AlbumTitle": "Hey Now",
    "Price": 1.98,
    "Genre": "Country",
    "CriticRating": 8.4
}
```

```
{
    "Artist": "No One You Know",
    "SongTitle": "Somewhere Down The Road",
    "AlbumTitle": "Somewhat Famous",
    "Genre": "Country",
    "CriticRating": 8.4,
    "Year": 1984
}
```

```
{
    "Artist": "The Acme Band",
    "SongTitle": "Still in Love",
    "AlbumTitle": "The Buck Starts Here",
    "Price": 2.47,
    "Genre": "Rock",
    "PromotionInfo": {
        "RadioStationsPlaying": [
            "KHCR",
            "KQBX",
            "WTNR",
            "WJJH"
        ],
        "TourDates": {
            "Seattle": "20150625",
            "Cleveland": "20150630"
        },
        "Rotation": "Heavy"
    }
}
```

```
{
    "Artist": "The Acme Band",
    "SongTitle": "Look Out, World",
    "AlbumTitle": "The Buck Starts Here",
    "Price": 0.99,
    "Genre": "Rock"
}
```

```
{
    "Genre": "Country",
    "AlbumTitle": "Hey Now”,
    "Artist": "No One You Know",
    "SongTitle": "My Dog Spot"
}
```

```
{
    "Genre": "Country",
    "AlbumTitle": "Somewhat Famous",
    "Artist": "No One You Know",
    "SongTitle": "Somewhere Down The Road"
}
```

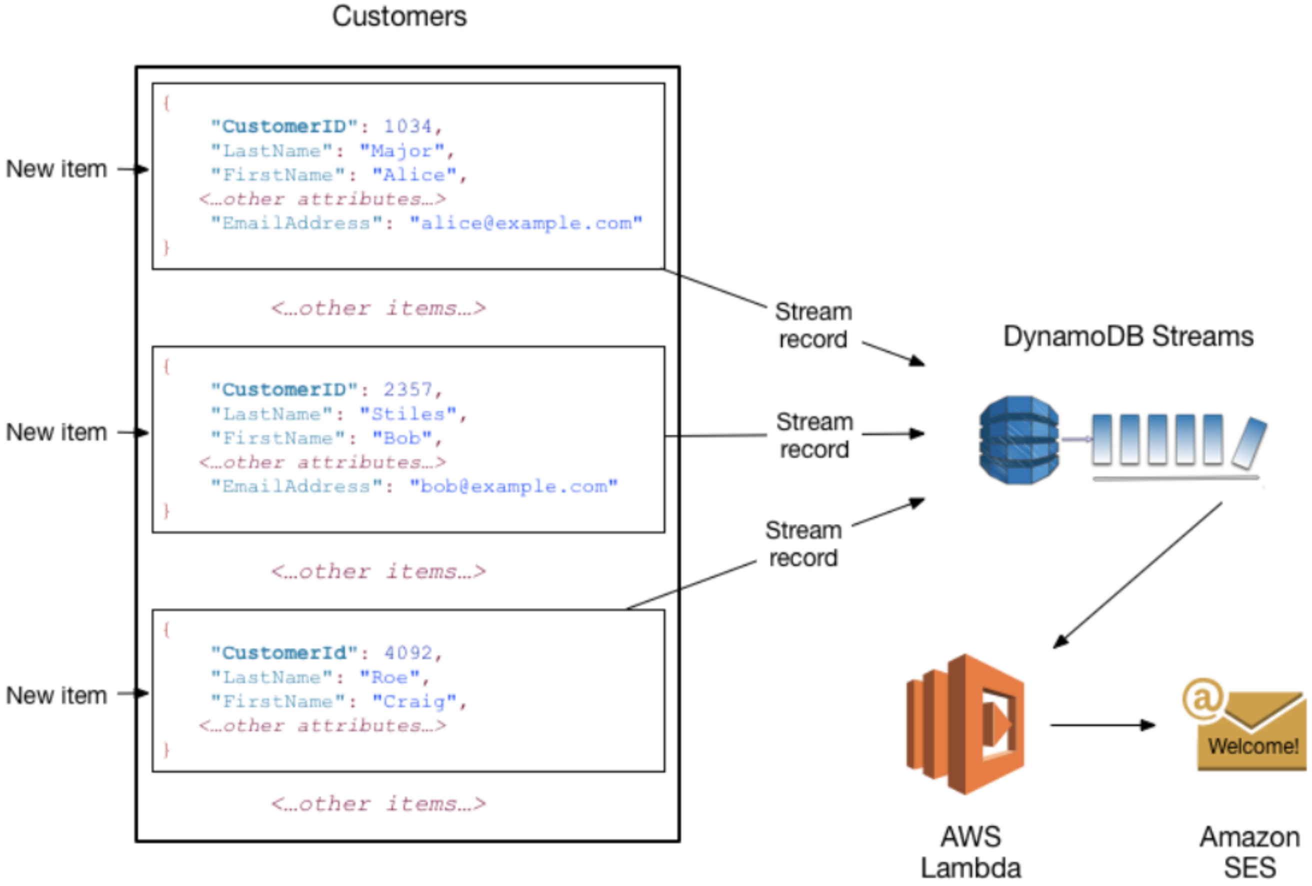
```
{
    "Genre": "Rock",
    "AlbumTitle": "The Buck Starts Here",
    "Artist": "The Acme Band",
    "SongTitle": "Still in Love"
}
```

```
{
    "Genre": "Rock"
    "AlbumTitle": "The Buck Starts Here",
    "Artist": "The Acme Band",
    "SongTitle": "Look Out, World"
}
```

Global secondary index

DynamoDB: Overview (3)

- Streams
  - ✓ Captures data modification event if enabled on a table
  - ✓ On new item, update, delete the item (before+after for a change), timestamp, table name, meta-data create a Stream record
- Triggers can be created by adding a Lambda fn



```
{
   "CustomerID": 1034,
   "LastName": "Major",
   "FirstName": "Alice",
  <…other attributes…>
   "EmailAddress": "alice@example.com"
}
```

<…other items…>

```
{
   "CustomerID": 2357,
   "LastName": "Stiles",
   "FirstName": "Bob",
  <…other attributes…>
   "EmailAddress": "bob@example.com"
}
```

<…other items…>

```
{
   "CustomerId": 4092,
   "LastName": "Roe",
   "FirstName": "Craig",
  <…other attributes…>
}
```

<…other items…>

e.g. send welcome email to new customers

DynamoDB: Consistency

- A table may be in multiple regions and replicated within region at AZs
- Eventual consistency reads
  - ✓ Faster reads
  - ✓ But stale data may be returned
  - ✓ Typically consistent within 1 sec
- Strong consistency (set ConsistentRead=True during query)
  - ✓ Returns latest write for reads
  - ✓ Reads may be unavailable on outages
  - ✓ Higher latency reads
  - ✓ Global secondary indices not supported for reads
  - ✓ Uses more throughput capacity for reads

DynamoDB: Read/Write Capacity modes

- Billing depends on how mode is set on a table
- On-demand
  - ✓ Single digit ms response on R/W
  - ✓ Needs no capacity planning
  - ✓ Good for unpredictable workloads that need fast response and pay-for-use
  - ✓ Auto-scaling happens within time limits. E.g. scales if workload doubles after 30min and throttles otherwise
- Provisioned (default)
  - ✓ Reserve capacity for read / write throughput needed
  - ✓ Cost predictability
  - ✓ Can still auto-scale between levels
- Capacity units
  - ✓ Read request unit = 1 x Strongly consistent read or 2 x Eventually consistent reads of 4KB items
  - ✓ Write request unit = 1 x 1 KB write or 0.5 x transactional 1KB write

DynamoDB: CRUD

```
{
   TableName : "Music",
   KeySchema: [
      {
         AttributeName: "Artist",
         KeyType: "HASH", //Partition key
      },
      {
         AttributeName: "SongTitle",
         KeyType: "RANGE" //Sort key
      }
   ],
   AttributeDefinitions: [
      {
         AttributeName: "Artist",
         AttributeType: "S"
      },
      {
         AttributeName: "SongTitle",
         AttributeType: "S"
      }
   ],
   ProvisionedThroughput: {      // Only specified if using provisioned mode
      ReadCapacityUnits: 1,
      WriteCapacityUnits: 1
   }
}
```

```
INSERT INTO Music
   (Artist, SongTitle, AlbumTitle,
   Year, Price, Genre,
   Tags)
VALUES(
   'No One You Know', 'Call Me Today', 'Somewhat Famous',
   2015, 2.14, 'Country',
   '{"Composers": ["Smith", "Jones", "Davis"],"LengthInSeconds": 214}'
);
```

```
/* Return all of the songs by an artist, with a particular word in the title...
...but only if the price is less than 1.00 */

SELECT * FROM Music
WHERE Artist='No One You Know' AND SongTitle LIKE '%Today%'
AND Price < 1.00;
```

```
// Return all of the songs by an artist, matching first part of title

{
   TableName: "Music",
   KeyConditionExpression: "Artist = :a and begins_with(SongTitle, :t)",
   ExpressionAttributeValues: {
      ":a": "No One You Know",
      ":t": "Call"
   }
}
```

- Options
  - ✓ PartiQL
  - ✓ APIs

Next Session:
Spark introduction