DynamoDB

basics

What?

“Amazon DynamoDB is a key-value and document database that delivers single-digit millisecond performance at any scale. It's a fully managed, multiregion, multimaster, durable database with built-in security, backup and restore, and in-memory caching for internet-scale applications. DynamoDB can handle more than 10 trillion requests per day and can support peaks of more than 20 million requests per second.”

What?

“Amazon DynamoDB is a key-value and document database that delivers single-digit millisecond performance at any scale. It's a fully managed, multiregion, multimaster, durable database with built-in security, backup and restore, and in-memory caching for internet-scale applications. DynamoDB can handle more than ***10 trillion requests per day*** *and can support peaks of more than* ***20 million requests per second.****”*

What?

~40,000 Searches per second

~9000 Tweets per second

~1000 Photos uploaded on instagram per second

~173 rides per second

History

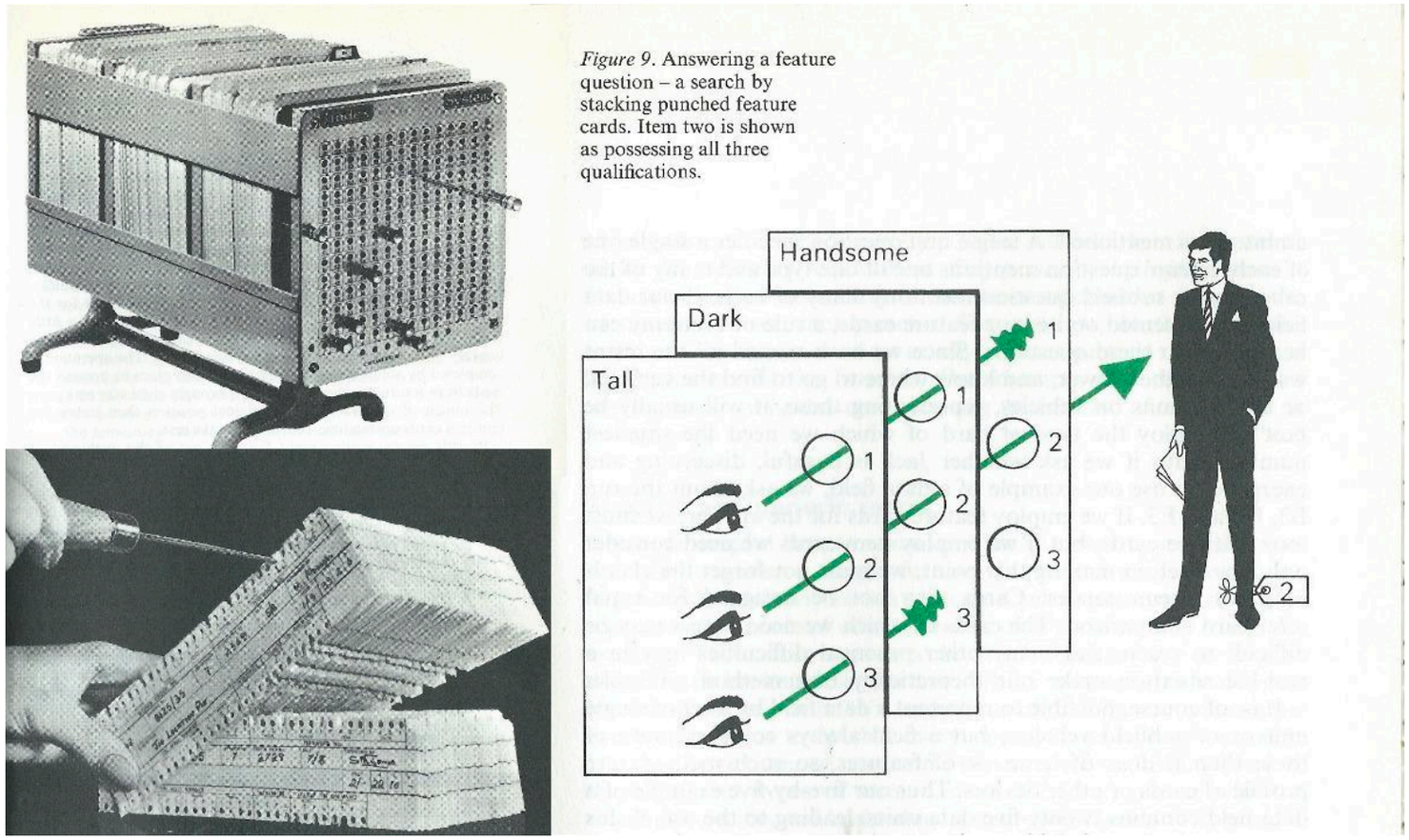
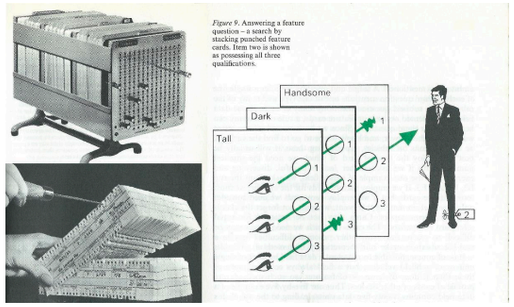


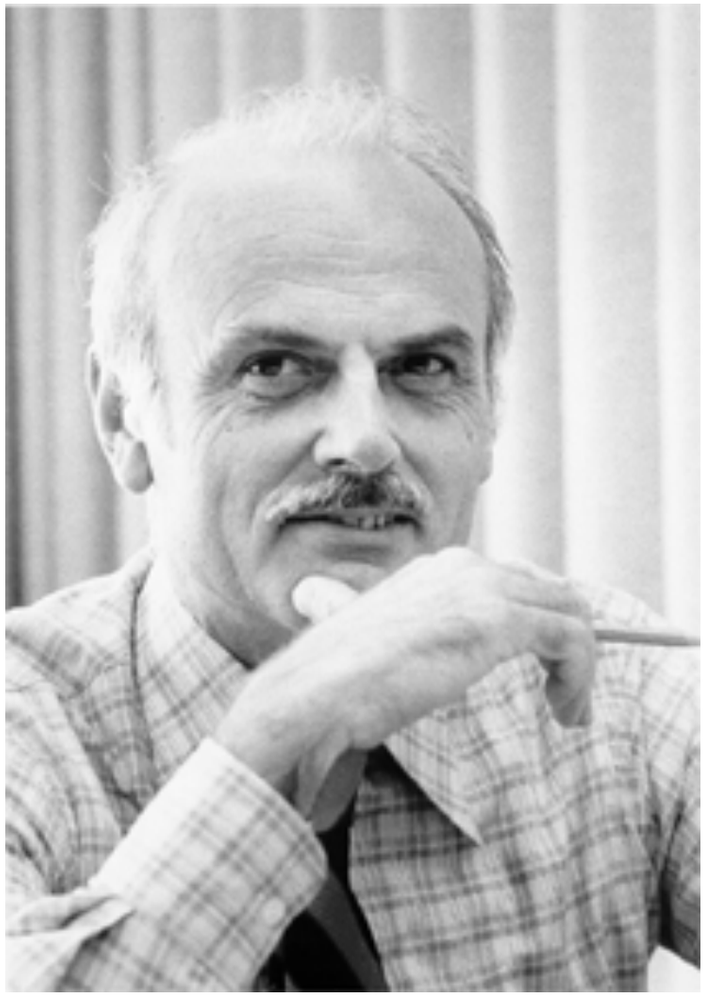
Figure 9. Answering a feature question – a search by stacking punched feature cards. Item two is shown as possessing all three qualifications.

Punchcards
— IBM

1920s

Edgar Frank



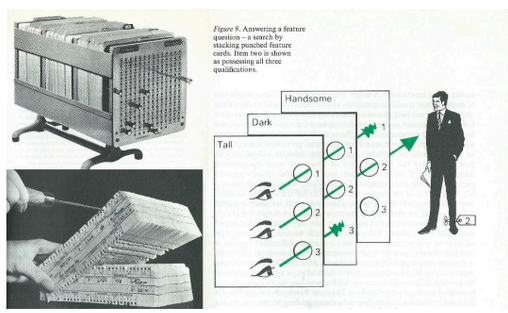


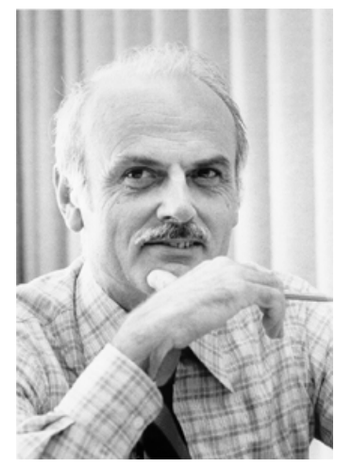
Punchcards

Relational Databases & SQL

1920s

1970s





Punchcards

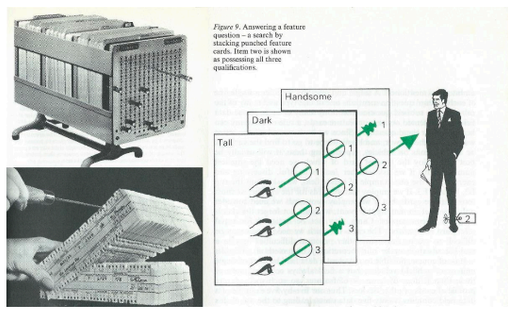
RDBMS & SQL

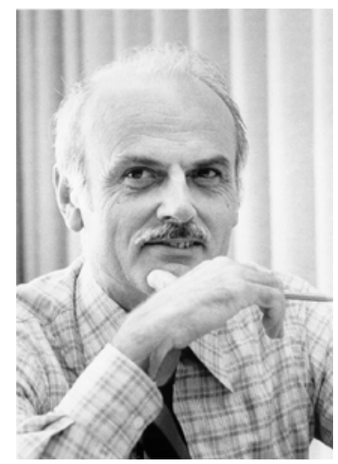
Internet is invented!

1920s

1970s

Paul is born!





Punchcards | **RDBMS & SQL** | **NoSQL**

1920s | **1970s** | **2000s**

1970s

- Compute and storage expensive
- Internet was slow
- Data didn’t change much
- ACID properties super important
- Data mostly used for analytical purposes

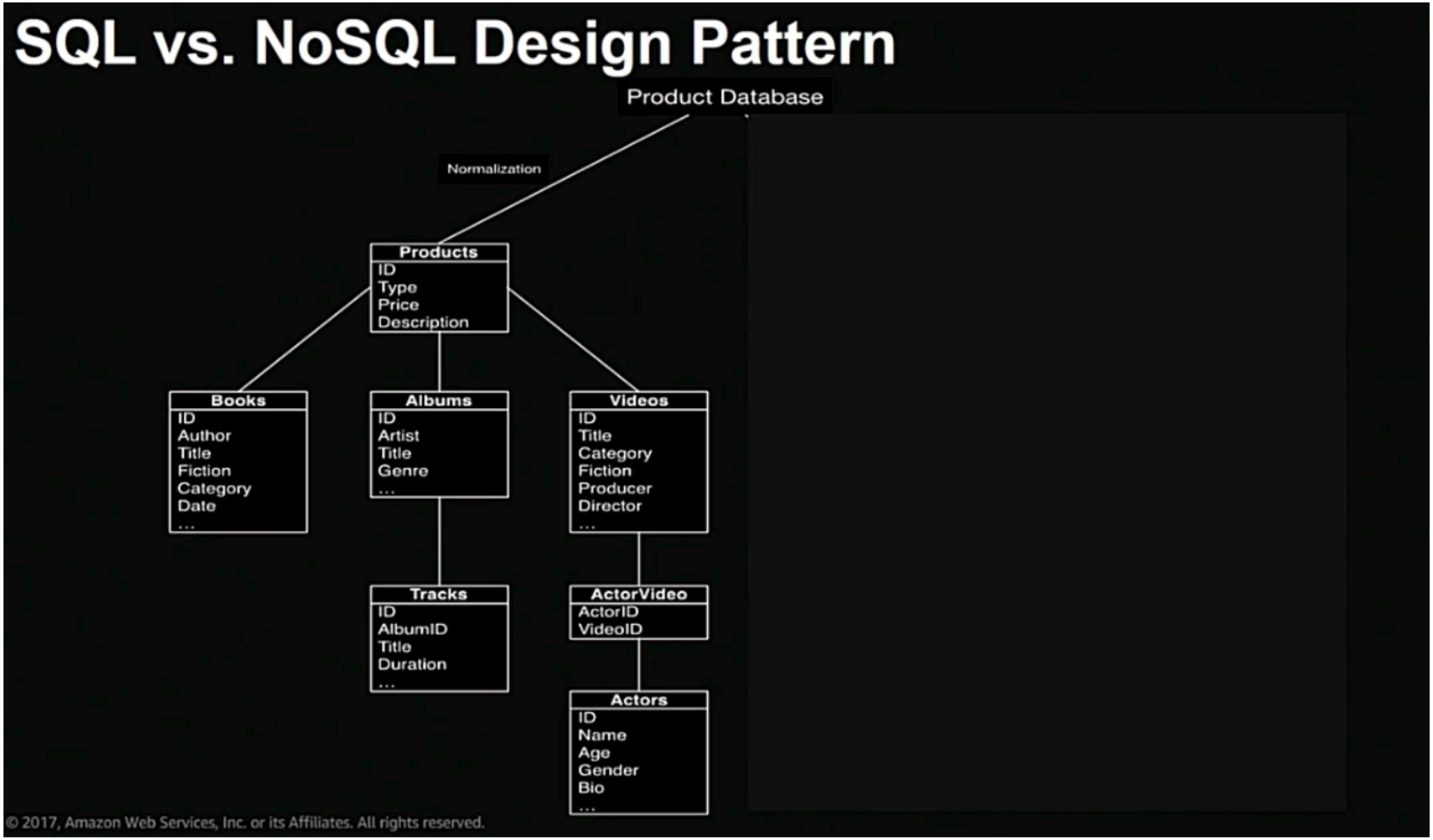
RDBMS

- Fixed schemas
- Data stored in normal form (less redundancy)
- No one really cared about horizontal scalability.
- ACIDity was important over latency
- SQL like support for complicated querying was important built in.

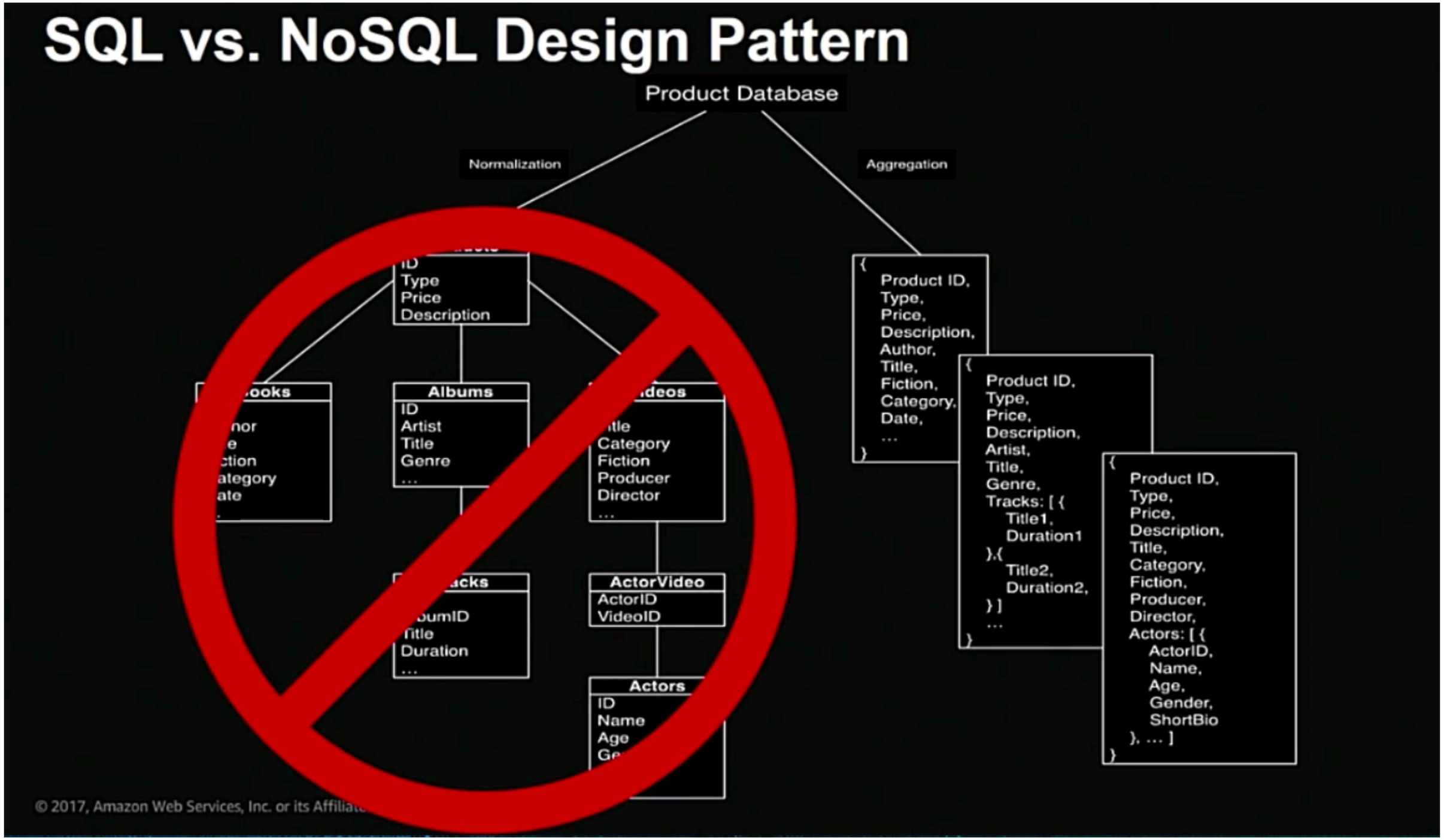
2000s

- Storage is cheap!
- Internet is fast!
- Data changes all the time!
- Super low latencies pls!
- Varying differences in data usage: search, analytical vs transactional processing, …

NoSQL


SQL vs. NoSQL Design Pattern
Product Database
Normalization
Products
ID
Type
Price
Description
Books
ID
Author
Title
Fiction
Category
Date
...
Albums
ID
Artist
Title
Genre
...
Videos
ID
Title
Category
Fiction
Producer
Director
...
Tracks
ID
AlbumID
Title
Duration
...
ActorVideo
ActorID
VideoID
Actors
ID
Name
Age
Gender
Bio
...
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

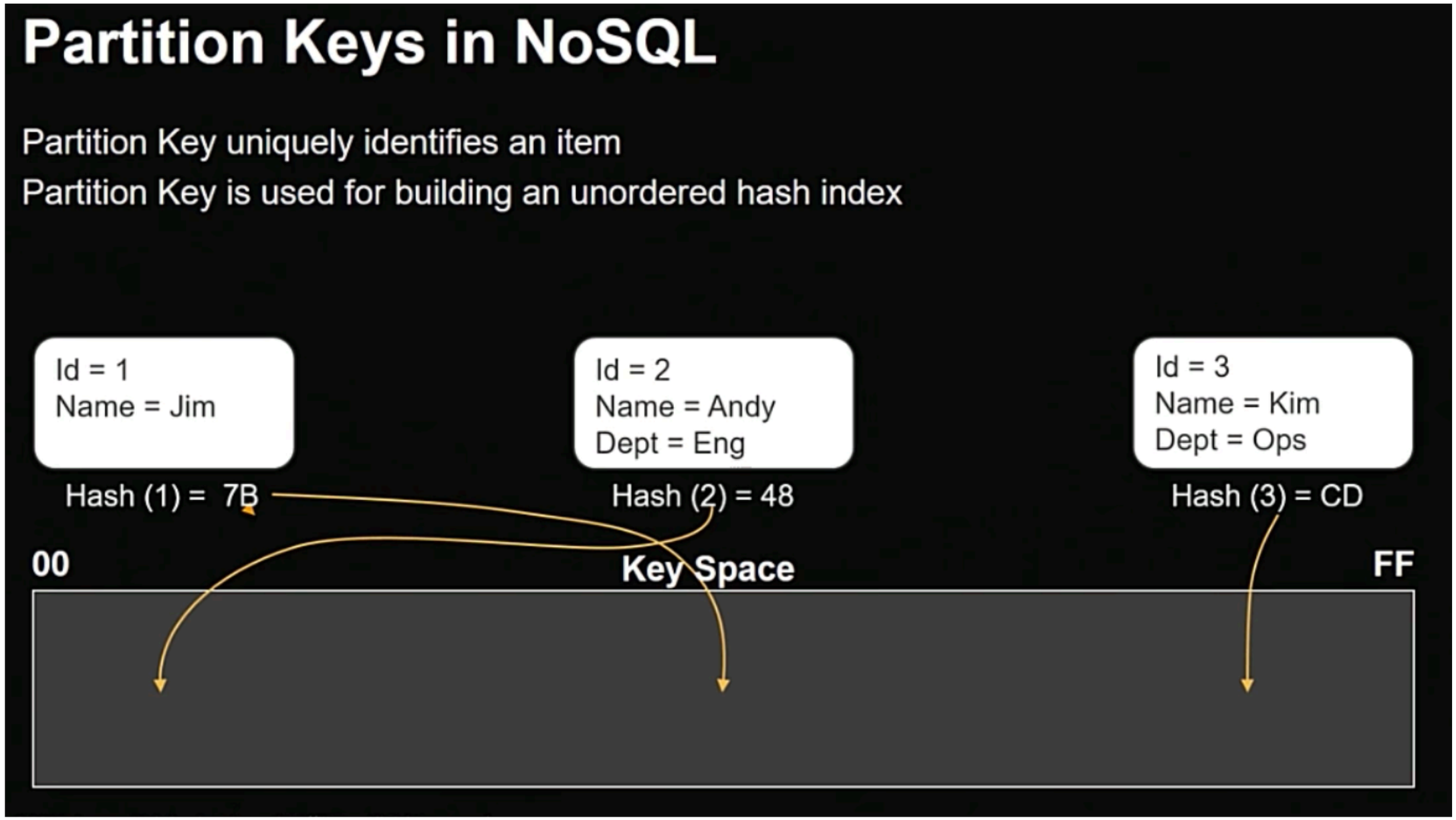
NoSQL


SQL vs. NoSQL Design Pattern
Product Database
Normalization
Aggregation
Type
Price
Description
Albums
ID
Artist
Title
Genre
...
Category
Fiction
Producer
Director
...
ActorVideo
ActorID
VideoID
Actors
ID
Name
Age
Duration
...
Product ID,
Type,
Price,
Description,
Author,
Title,
Fiction,
Category,
Date,
...
Product ID,
Type,
Price,
Description,
Artist,
Title,
Genre,
Tracks: [{
Title1,
Duration1
},{
Title2,
Duration2,
}]
...
Product ID,
Type,
Price,
Description,
Title,
Category,
Fiction,
Producer,
Director,
Actors: [{
ActorID,
Name,
Age,
Gender,
ShortBio
}, ...]
© 2017, Amazon Web Services, Inc. or its Affiliat

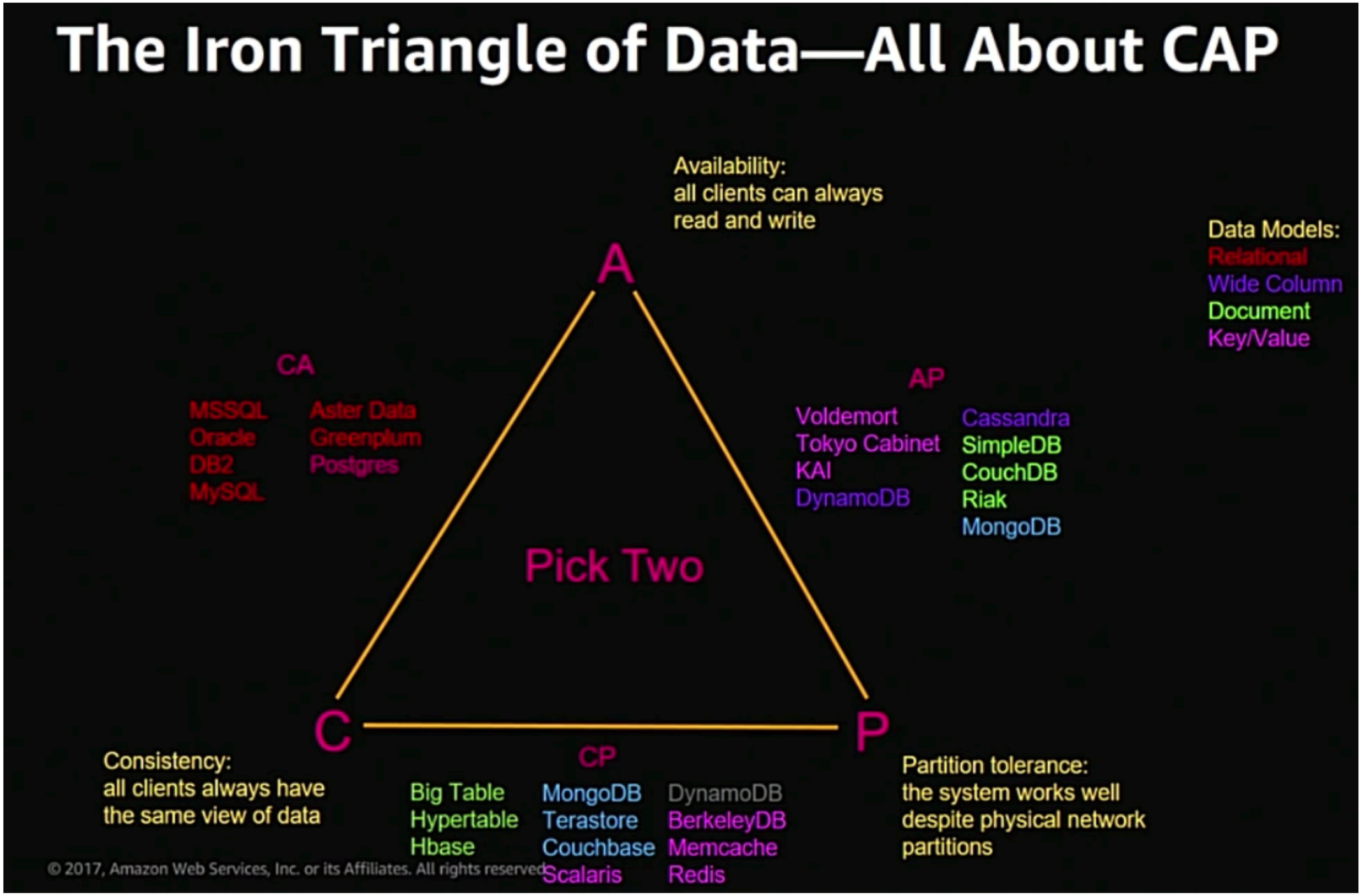
SQL vs NoSQL

SQL	NoSQL
Optimized for storage	**Optimized for compute**
Normalized/relational	Denormalized/hierarchical
Ad hoc queries	Instantiated views
Scale vertically	Scale horizontally
Good for OLAP	Built for OLTP at scale

Horizontal Scaling



CAP


The Iron Triangle of Data—All About CAP
Availability:
all clients can always
read and write
A
Data Models:
Relational
Wide Column
Document
Key/Value
CA
MSSQL
Oracle
DB2
MySQL
Aster Data
Greenplum
Postgres
AP
Voldemort
Tokyo Cabinet
KAI
DynamoDB
Cassandra
SimpleDB
CouchDB
Riak
MongoDB
Pick Two
C
P
Consistency:
all clients always have
the same view of data
CP
Big Table
Hypertable
Hbase
MongoDB
Terastore
Couchbase
Scalaris
DynamoDB
BerkeleyDB
Memcache
Redis
Partition tolerance:
the system works well
despite physical network
partitions
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved

DynamoDB

DynamoDB

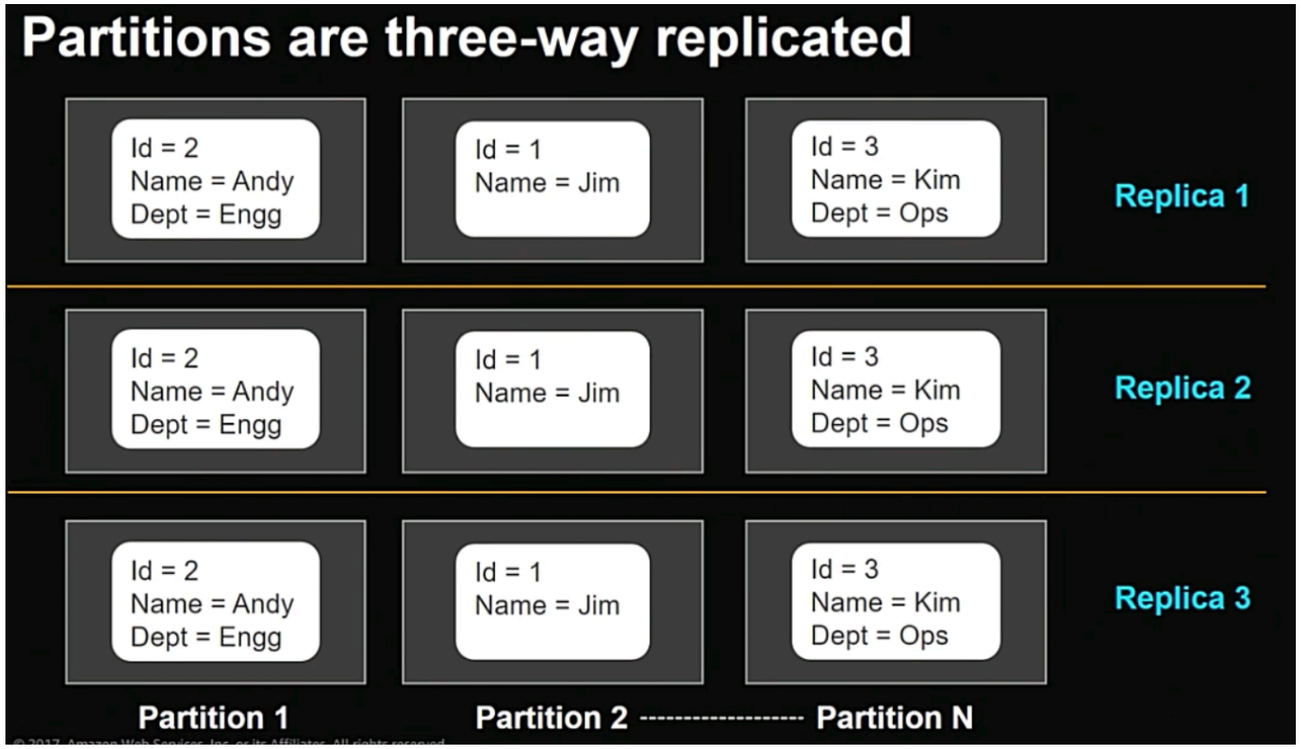
- Basics:
  - Partitions, Storage Nodes, & Read & Write Units
- Operations:
  - Creating a table
  - Get
  - Put
- Secondary Indexes:
  - GSI
  - LSI
- Provisioning/Autoscaling
  - Bursting
  - Adaptive Capacity

DynamoDB

- Backups & Restore:
  - Point in Time Recovery
  - On demand
- Streams
  - Triggers
  - Global Tables
- DAX
- Transactions

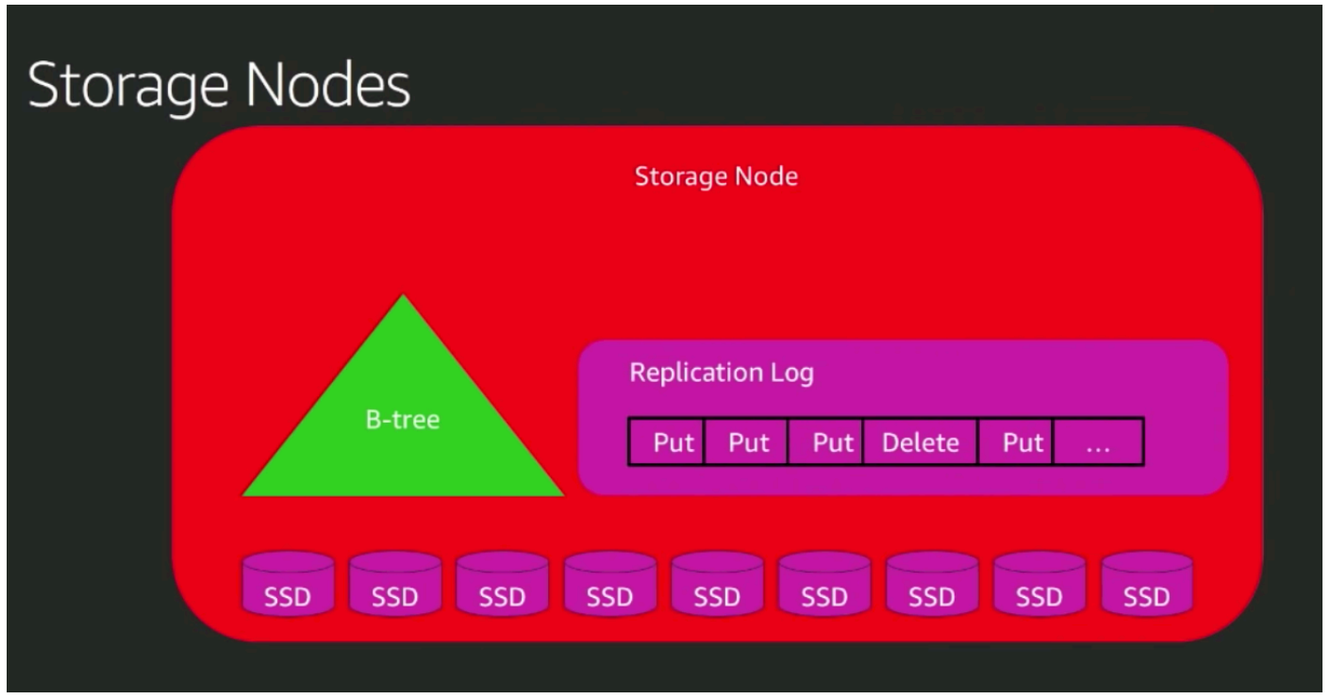
Partitions

- Each partition can store up to 10GB of data and handle by default 1,000 write capacity units (WCU) and 3,000 read capacity units (RCU).
- One RCU represents one strongly consistent read per second or two eventually consistent reads per second for items up to 4KB in size.
- One WCU represents one write per second for an item up to 1KB in size.



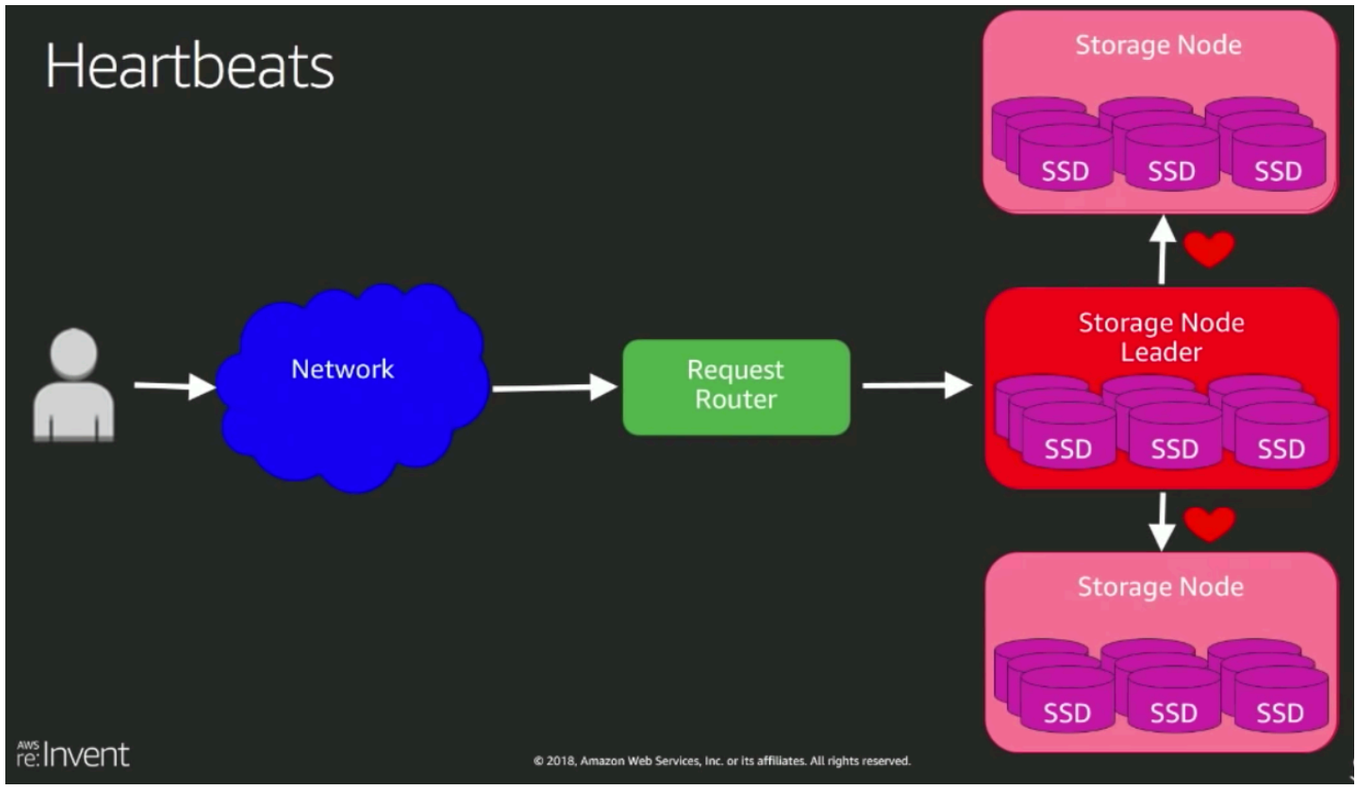
Storage Nodes

- Each partition features three nodes, each of which contains a copy of that partition's data.
- Each node also contains two data structures: a B tree used to locate items, and a replication log that notes all changes made to the node.



Storage Nodes

- In each partition, one of the three nodes is designated the “leader node”. All write operations travel first through the leader node before propagating, which makes writes consistent in DynamoDB. To maintain its status, the leader sends a “heartbeat” to each other node every 1.5 seconds. Should another node stop receiving heartbeats, it can initiate a new leader election.



Operations

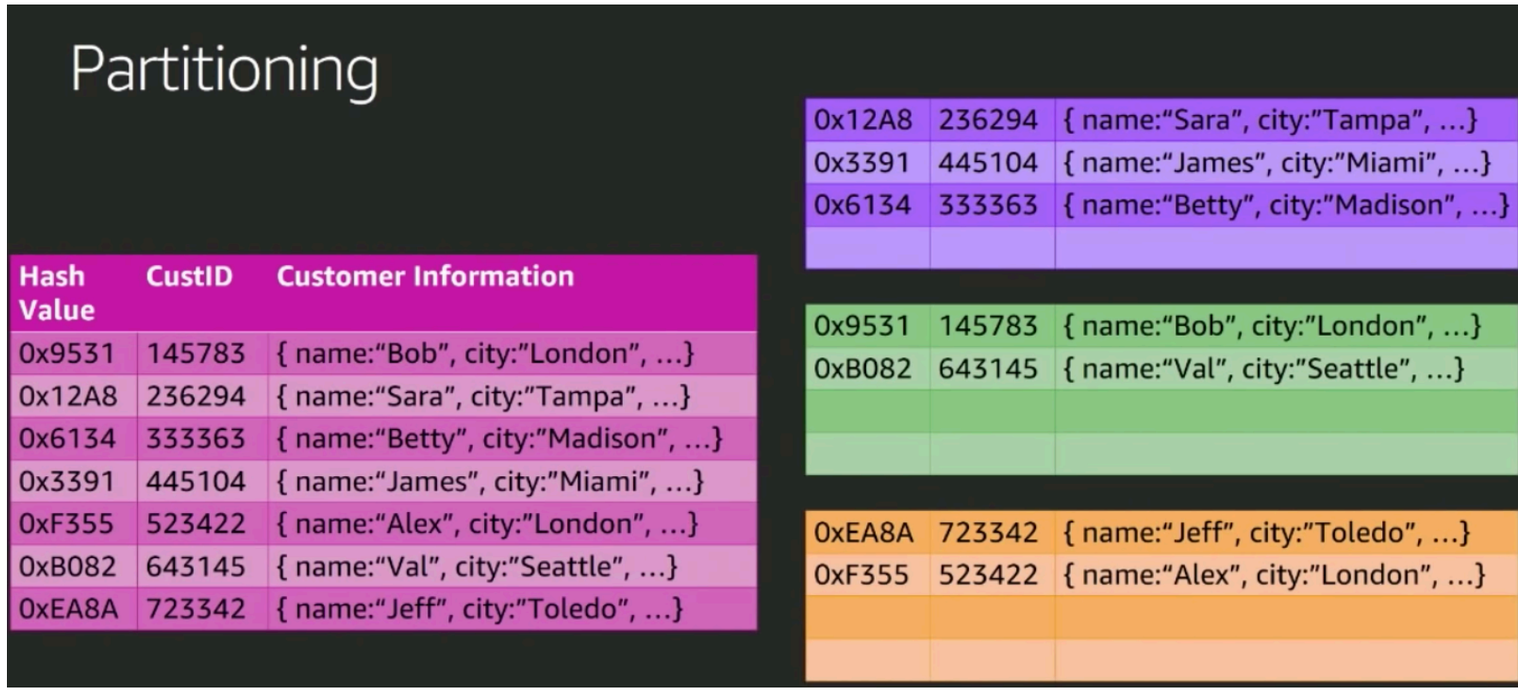
Creating a table

- Creating a table requires a TableName, KeySchema (list containing a partition key and an optional sort key), and a list of attributes to be defined which must at least contain definitions for the attributes used as partition and sort keys.
- Ideally, this partition key should be equally distributed across your access patterns.

Hash Value	CustID	Customer Information
0x9531	145783	{ name:"Bob", city:"London", ...}
0x12A8	236294	{ name:"Sara", city:"Tampa", ...}
0x6134	333363	{ name:"Betty", city:"Madison", ...}
0x3391	445104	{ name:"James", city:"Miami", ...}
0xF355	523422	{ name:"Alex", city:"London", ...}
0xB082	643145	{ name:"Val", city:"Seattle", ...}
0xEA8A	723342	{ name:"Jeff", city:"Toledo", ...}

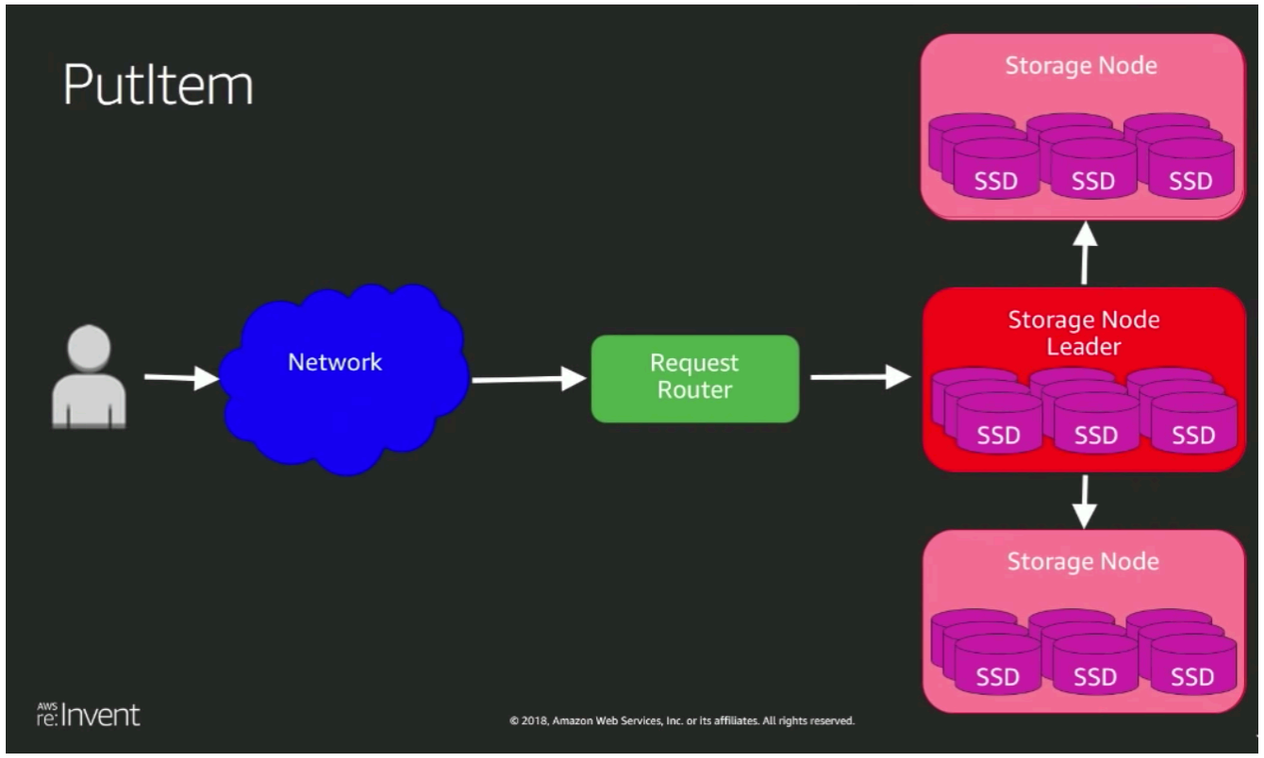
Creating a table

- Dynamo automagically partitions your table based on the partition keys.
- It also magically repartitions your data based on whether your RCUs, WCUs, and storage token capacity.



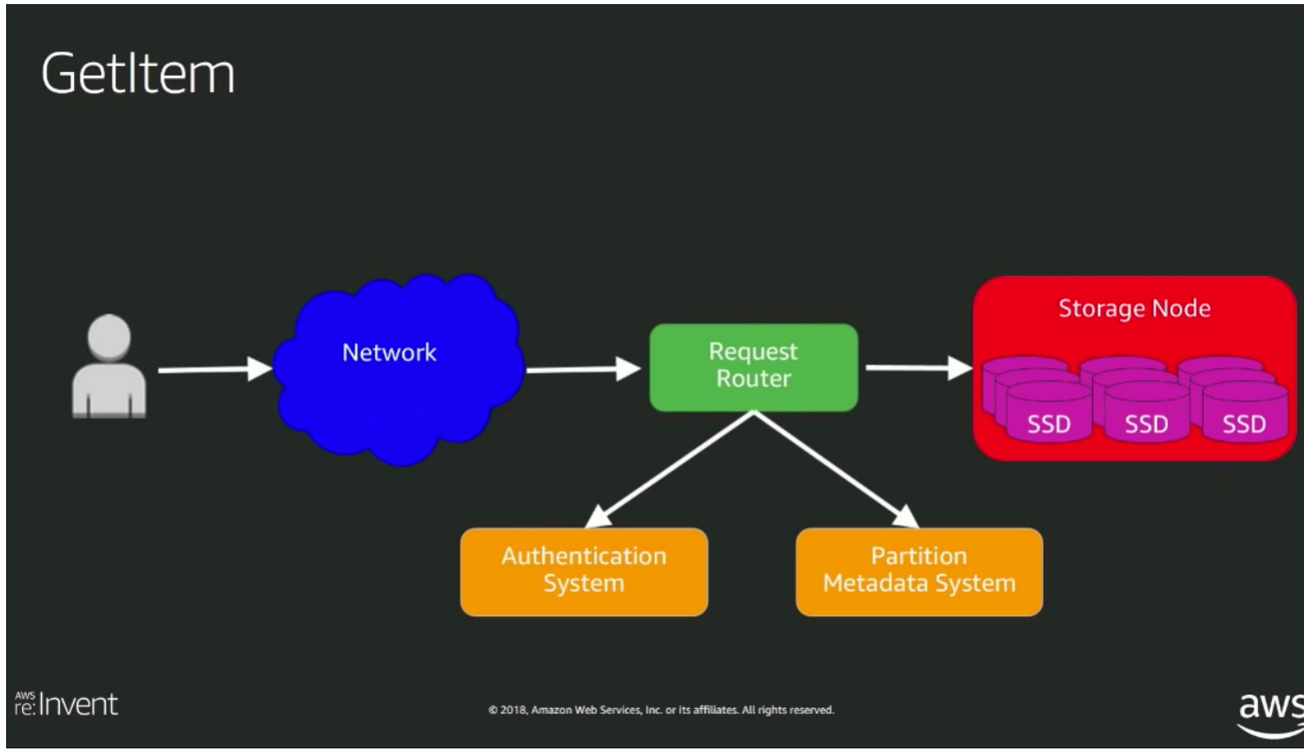
Put

- The system hashes the request's partition key to arrive in the appropriate partition. Then, the system first writes to the leader node, then to a second node, then sends a “success” message, and finally continues propagating to the third node. Writes are consistent because they always travel first through the leader node.



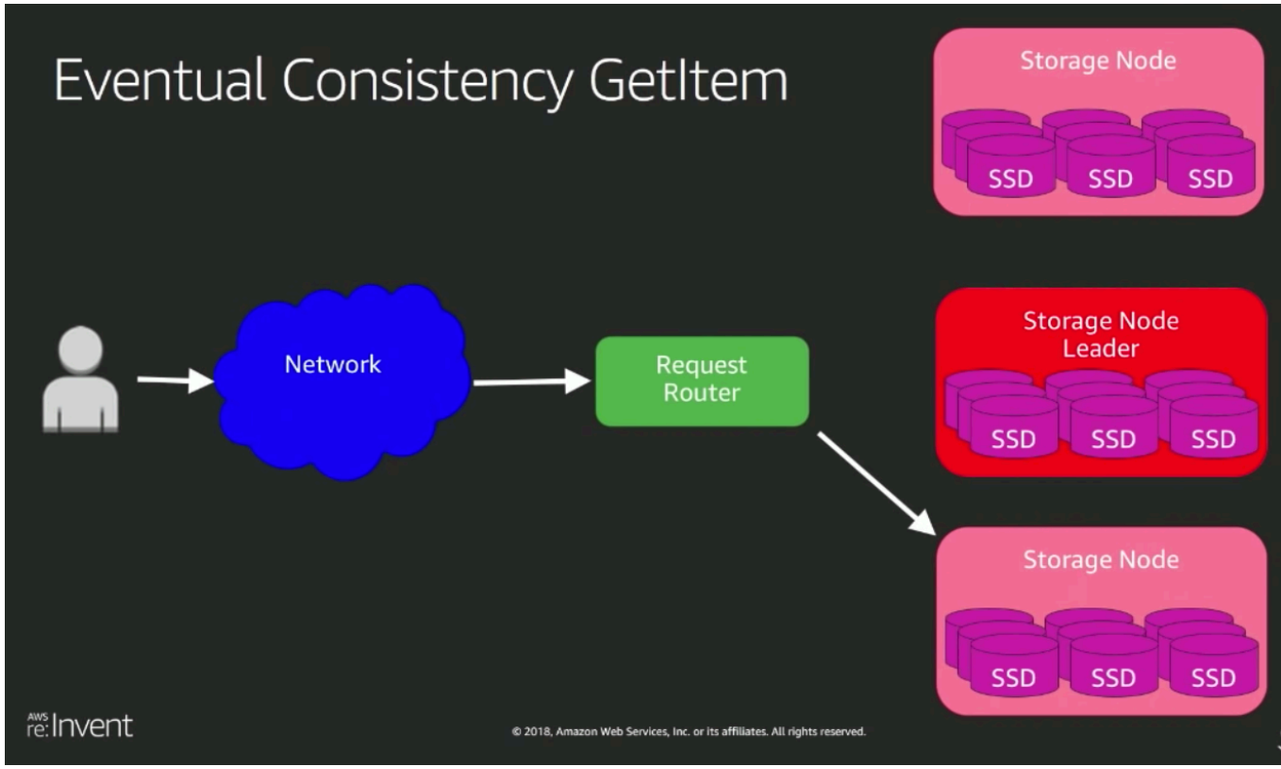
Get

- To perform a get we first hash our partition key to arrive in the appropriate hash.
- Now, DynamoDB affords the user two options when issuing a read: consistent and eventually consistent. A consistent read visits the leader node.
- But the consistency-availability trade-off rears its head again here: in read-heavy systems, always reading from the leader can overwhelm a single node and reduce availability.



Get

- An eventually consistent read, selects a random node. This is where DynamoDB trades consistency for availability. If we take this route, the odds of an inconsistency are 1-in-3. We'd need a write operation to return “success” and begin propagating to the third node, but not finish. We'd also need the Get to target this third node.



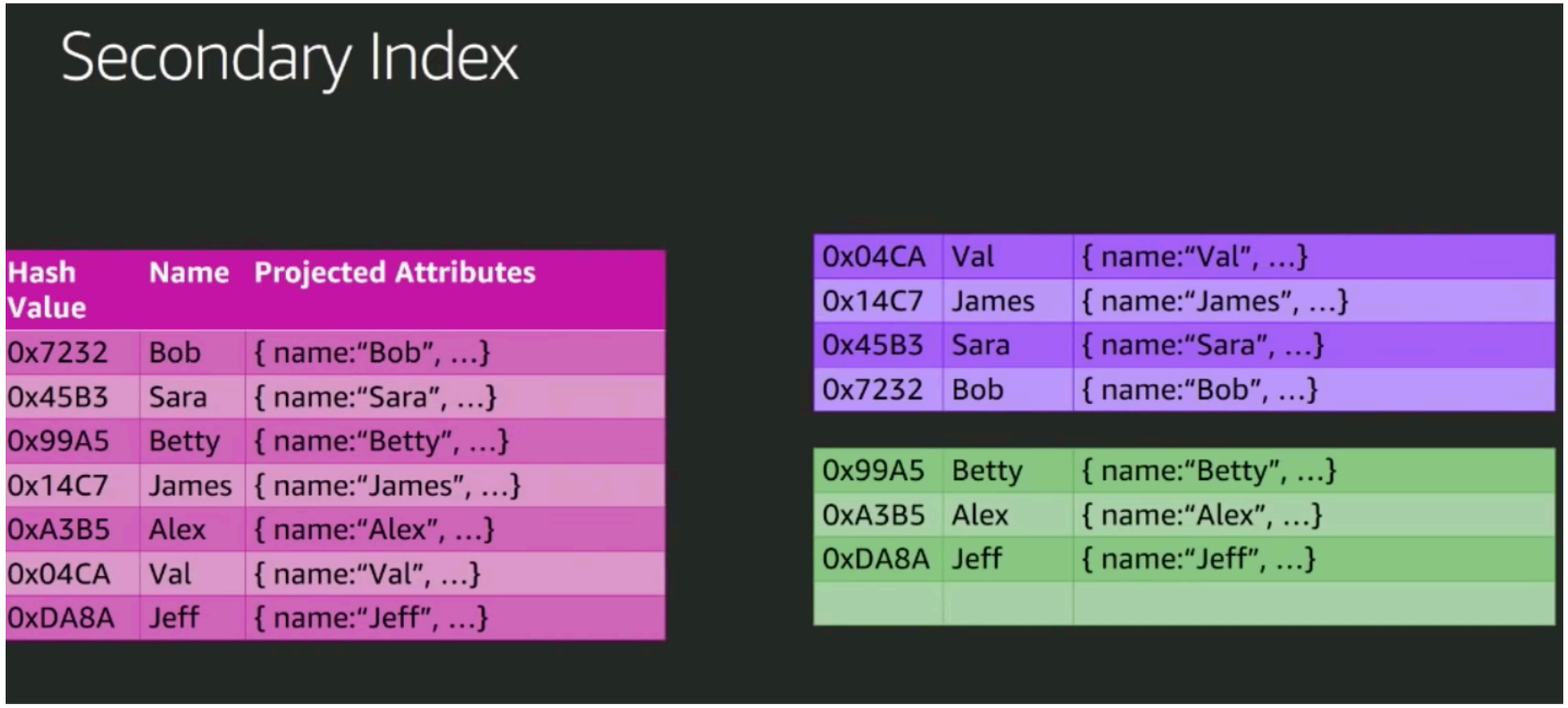
Indexes

Indexes

- In DynamoDB, there is no query optimizer, and an index is simply another table with a different key (or two).
- When a developer creates an index, she creates a new copy of her data, but only the fields that she specifies get copied over (at a minimum, the fields that she indexes on and the original table's primary key).

LSI

- A local secondary index is an index that has the same partition key as the base table, but a different sort key. It is "local" in the sense that every partition of a local secondary index is scoped to a base table partition that has the same partition key value.

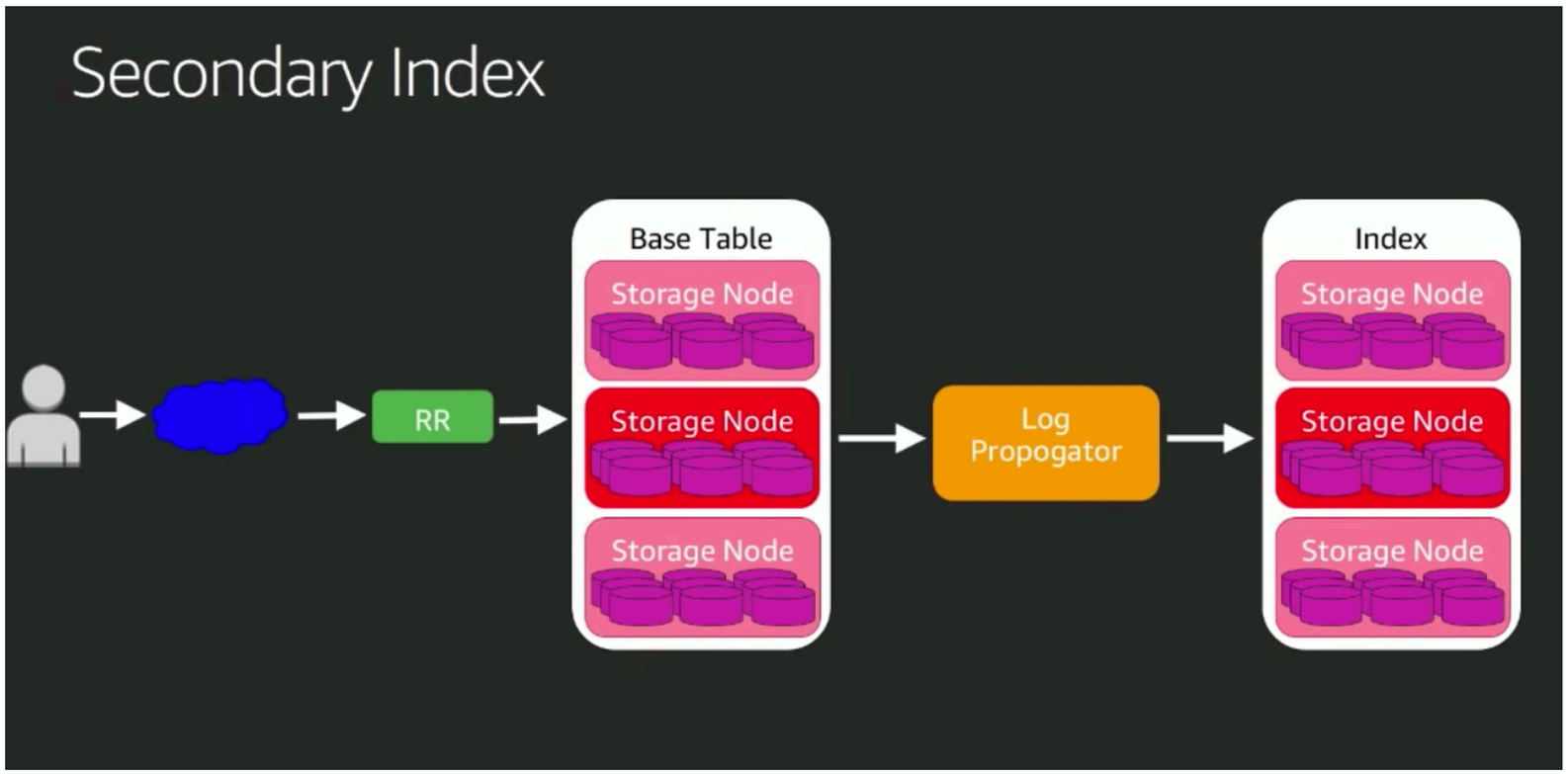


GSI

- An index with a partition key and a sort key that can be different from those on the base table. Considered "global" because queries on the index can span all of the data in the base table, across all partitions.
- There are no size restrictions for global secondary indexes and queries on global secondary indexes support eventual consistency only.
- A global secondary index lets you query over the entire table, across all partitions.

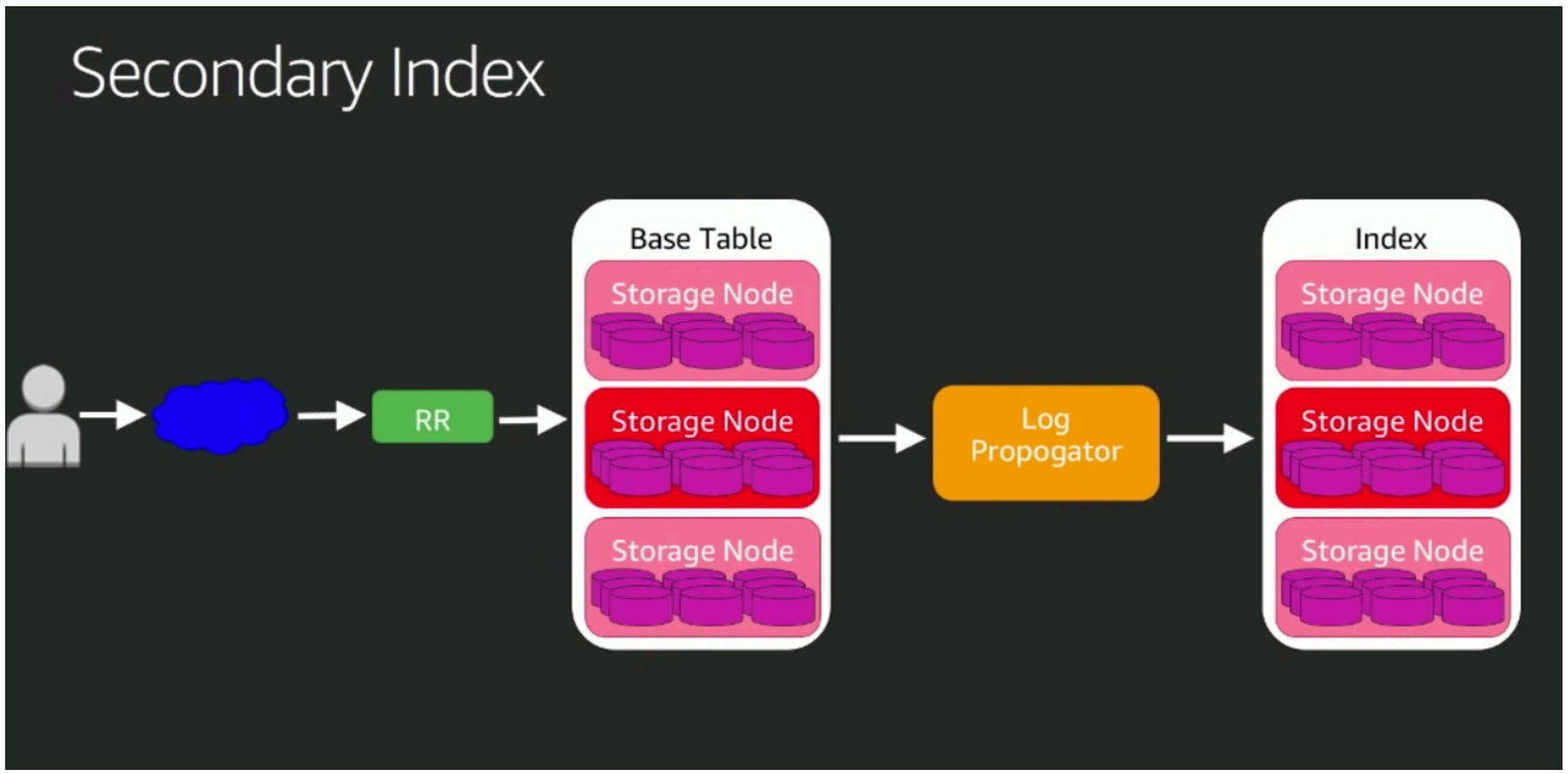
Indexes

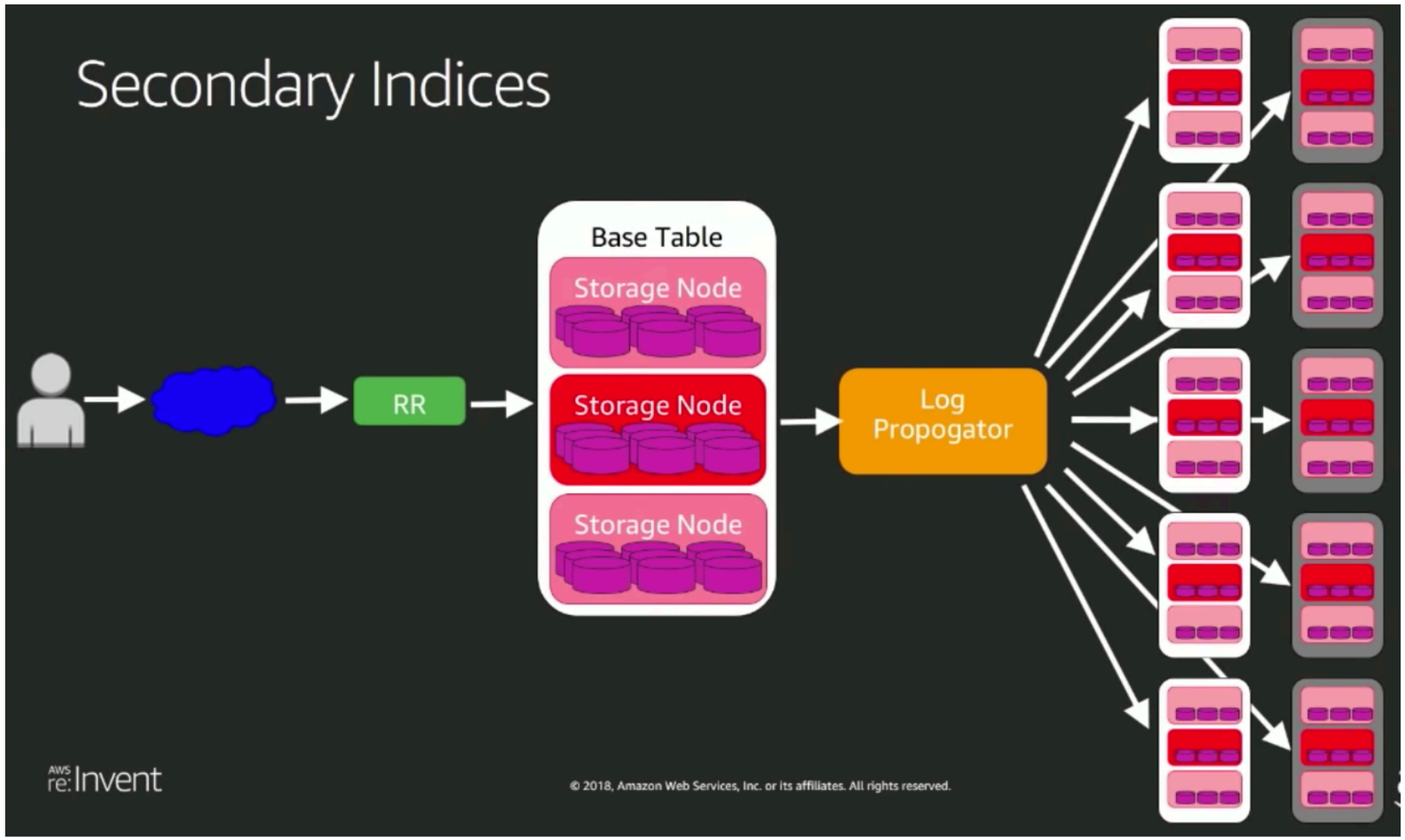
- Updates to a table be reflected in each of the table's indices. DynamoDB handles this using a service it calls the “log propagator”, which subscribes to the replication logs in each node and sends additional Put, Update, and Delete requests to indices as necessary.
- Thus, one should be careful while updating indices.



Indexes

- Updates to a table be reflected in each of the table's indices. DynamoDB handles this using a service it calls the “log propagator”, which subscribes to the replication logs in each node and sends additional Put, Update, and Delete requests to indices as necessary.
- Thus, one should be careful while updating indices.




Secondary Indices
Base Table
Storage Node
Storage Node
Storage Node
RR
Log
Propogator
AWS re:Invent
© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AutoAdmin & RequestRouter

AutoAdmin

Auto Admin

Auto Admin

Partition Repair

Create Tables

Table Provisioning

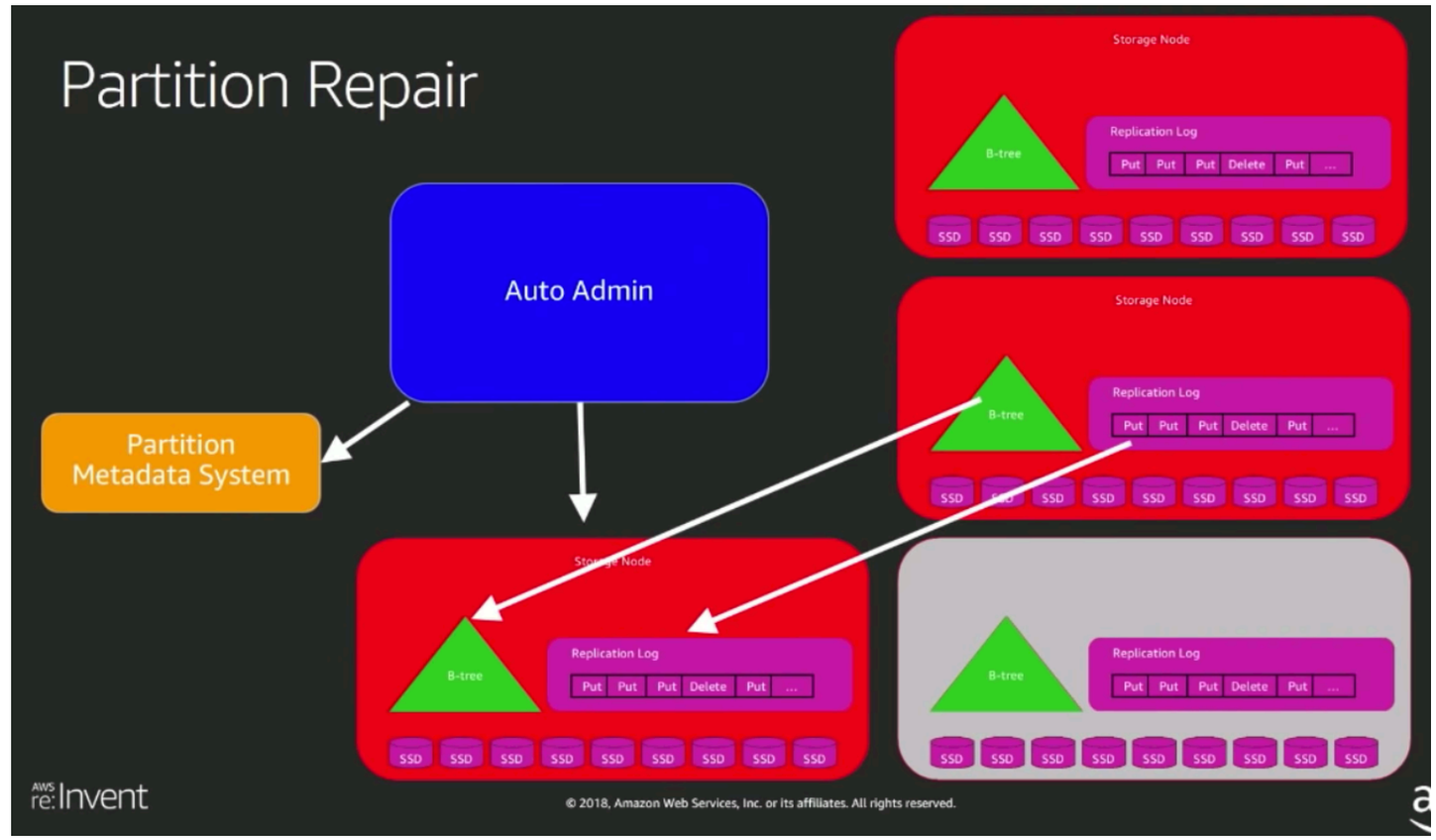
Split Partition

… (DBA for DynamoDB)

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Partition Repair

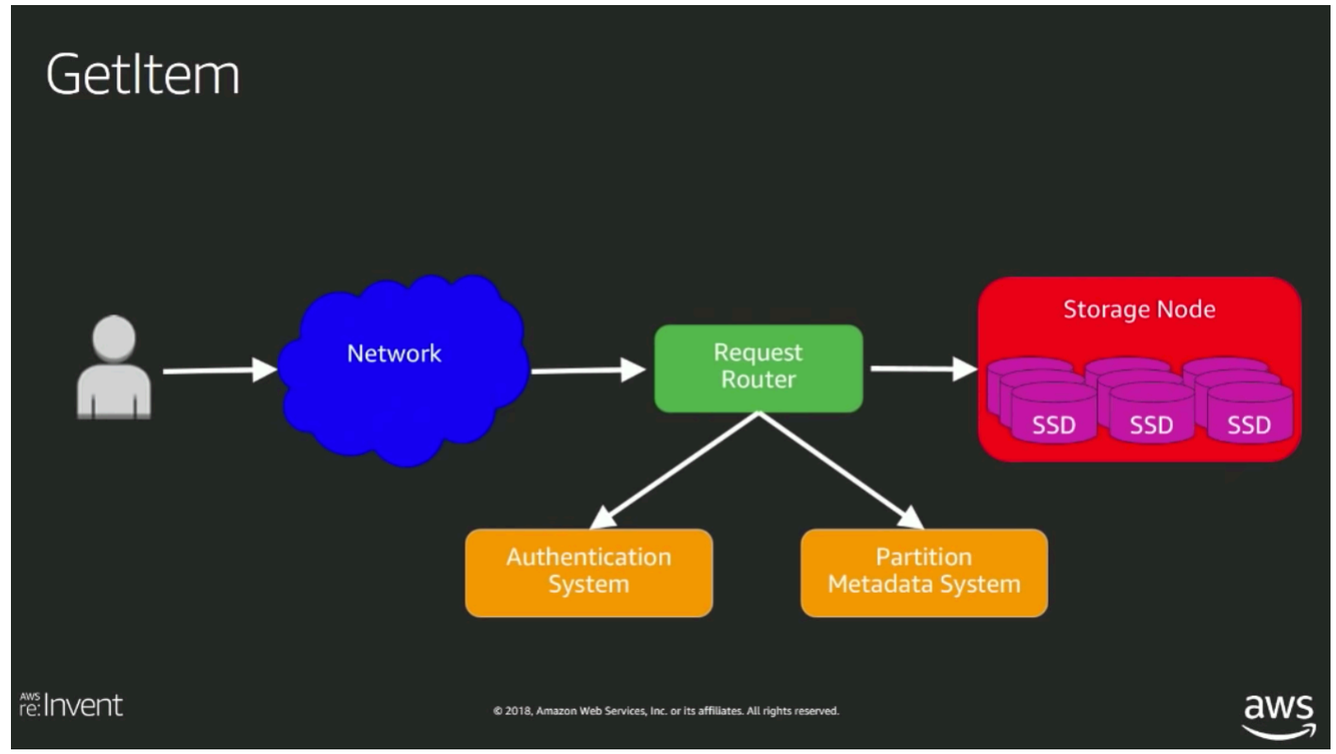
- When one storage node goes down, auto admin creates a new node with from the BTree + replication log of the leader.

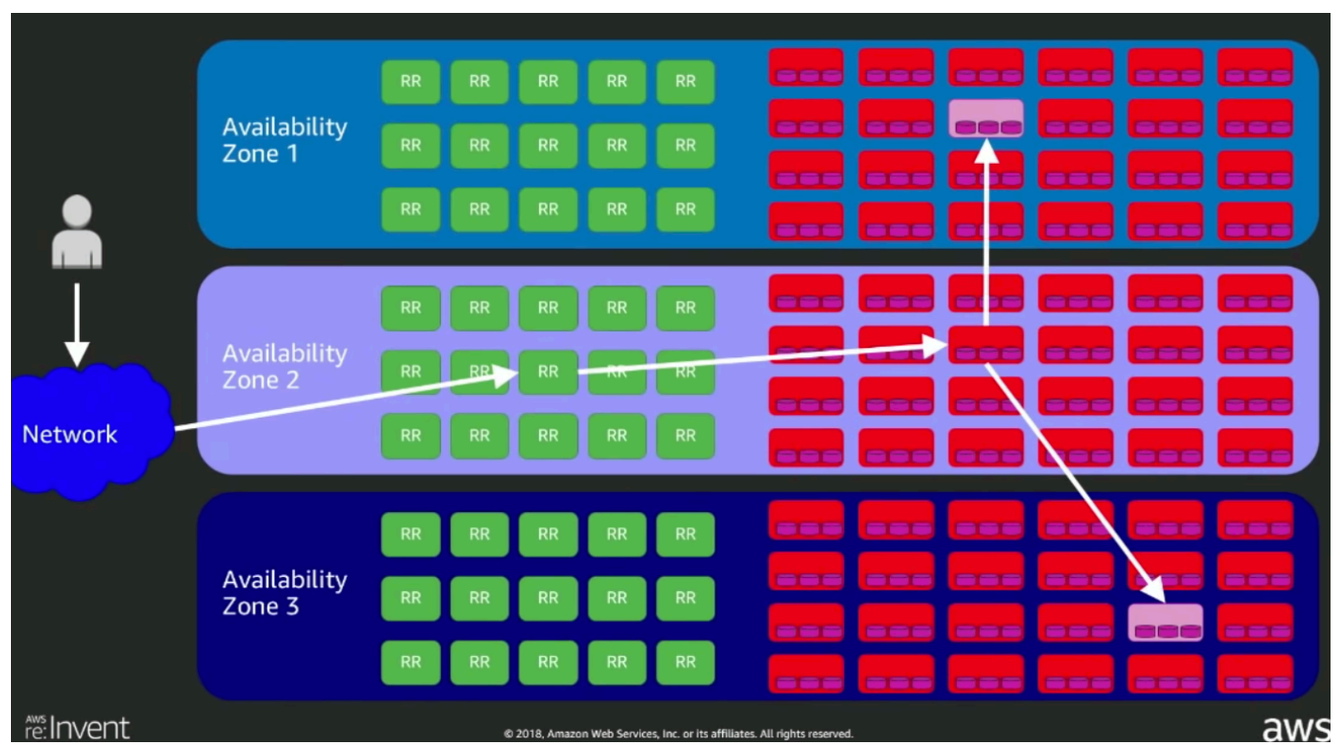


Request Router

Request Router

- Stateless service that does authentication + authorization.
- Also communicates with Partition Metadata System to get the partition hash and find the leader.





Provisioning

aws Services Resource Groups jaso Oregon Support

Create DynamoDB table

Tutorial

DynamoDB is a schema-less database that only requires a table name and primary key. The table's primary key is made up of one or two attributes that uniquely identify items, partition the data, and sort data within each partition.

Table name* ExampleTable

Primary key* Partition key

ExampleKey String

Add sort key

Table settings

Default settings provide the fastest way to get started with your table. You can modify these default settings now or after your table has been created.

Use default settings

Secondary indexes

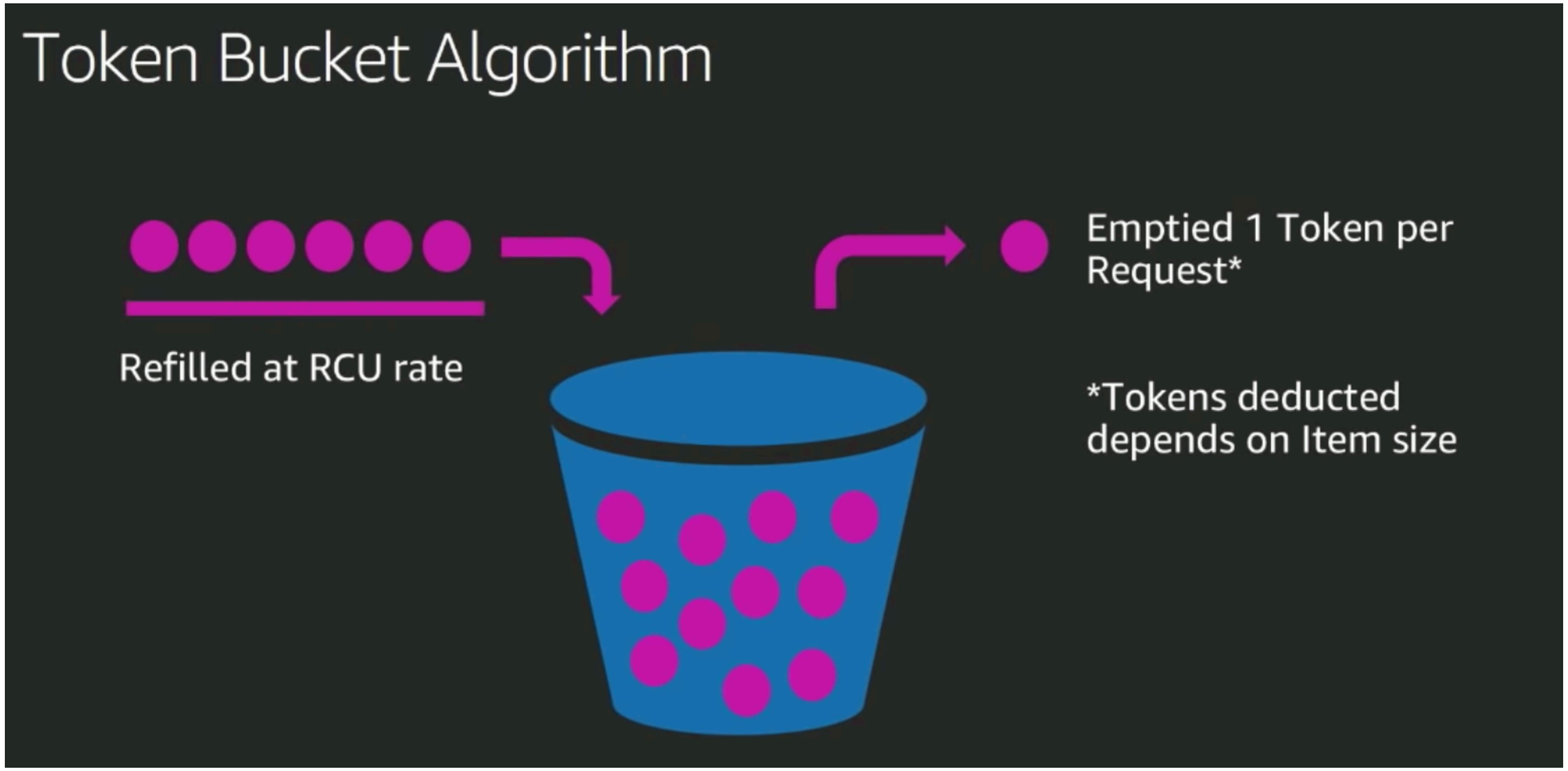
Name	Type	Partition key	Sort key	Projected Attributes

+ Add index

Provisioned capacity

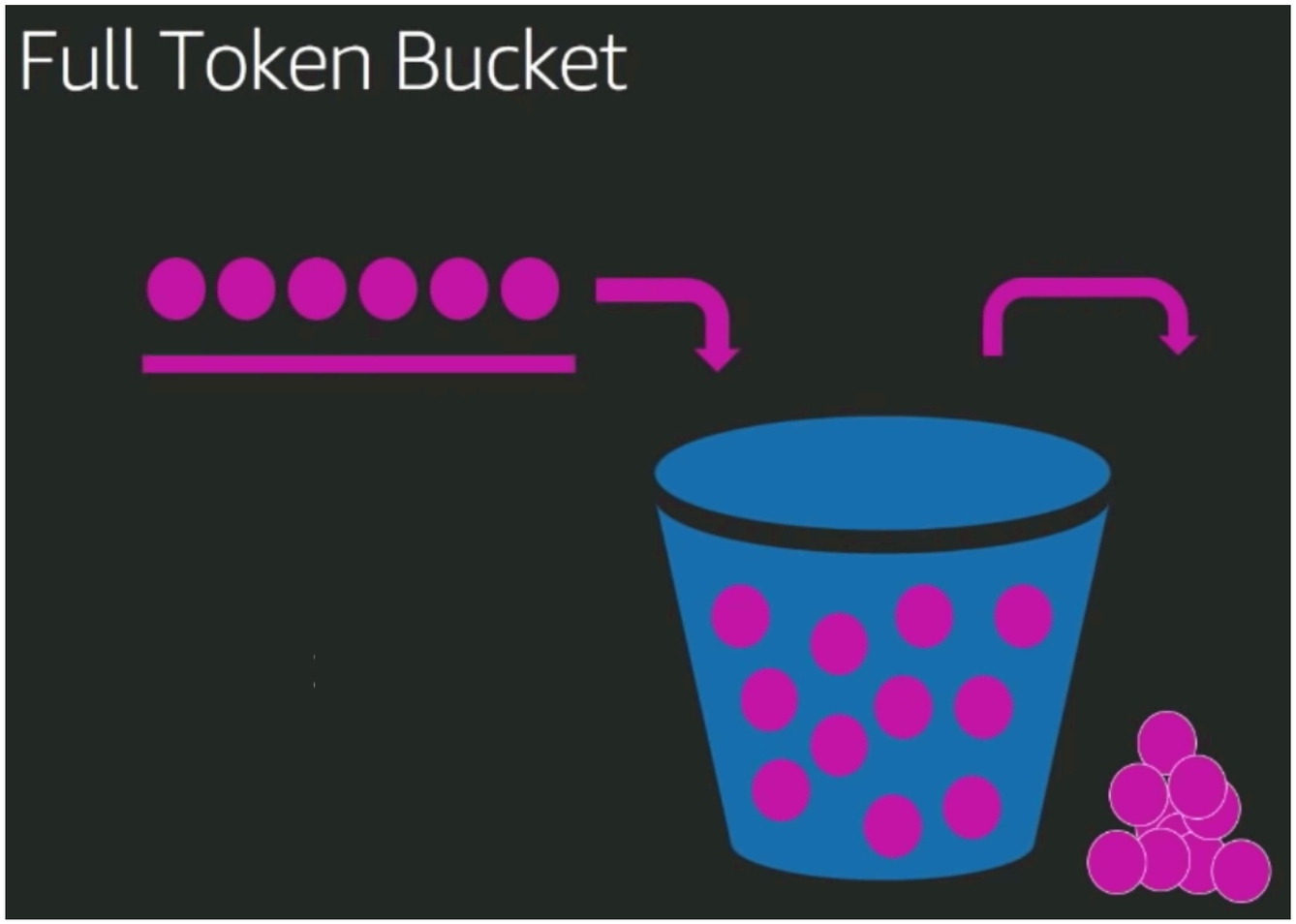
	Read capacity units	Write capacity units
Table	5	5

Provisioning



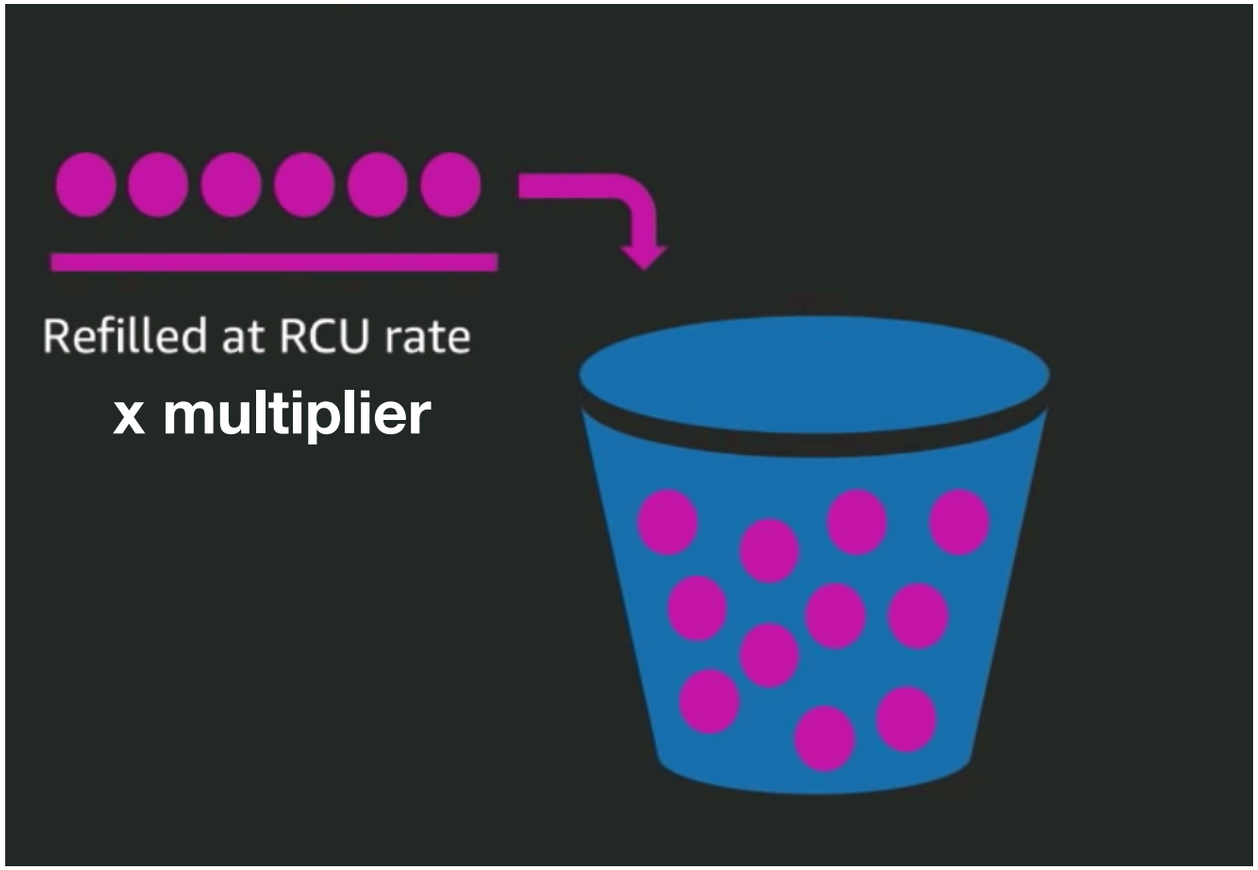
Bursting

- Certain services can have bursts where load suddenly increases. — Ceres, S2
- DynamoDB allows you to accumulate 5 mins worth of tokens. (300x) This means that if you don't use it for 5 min and then have huge bursts, it will allow you to use unused tokens without throttling.

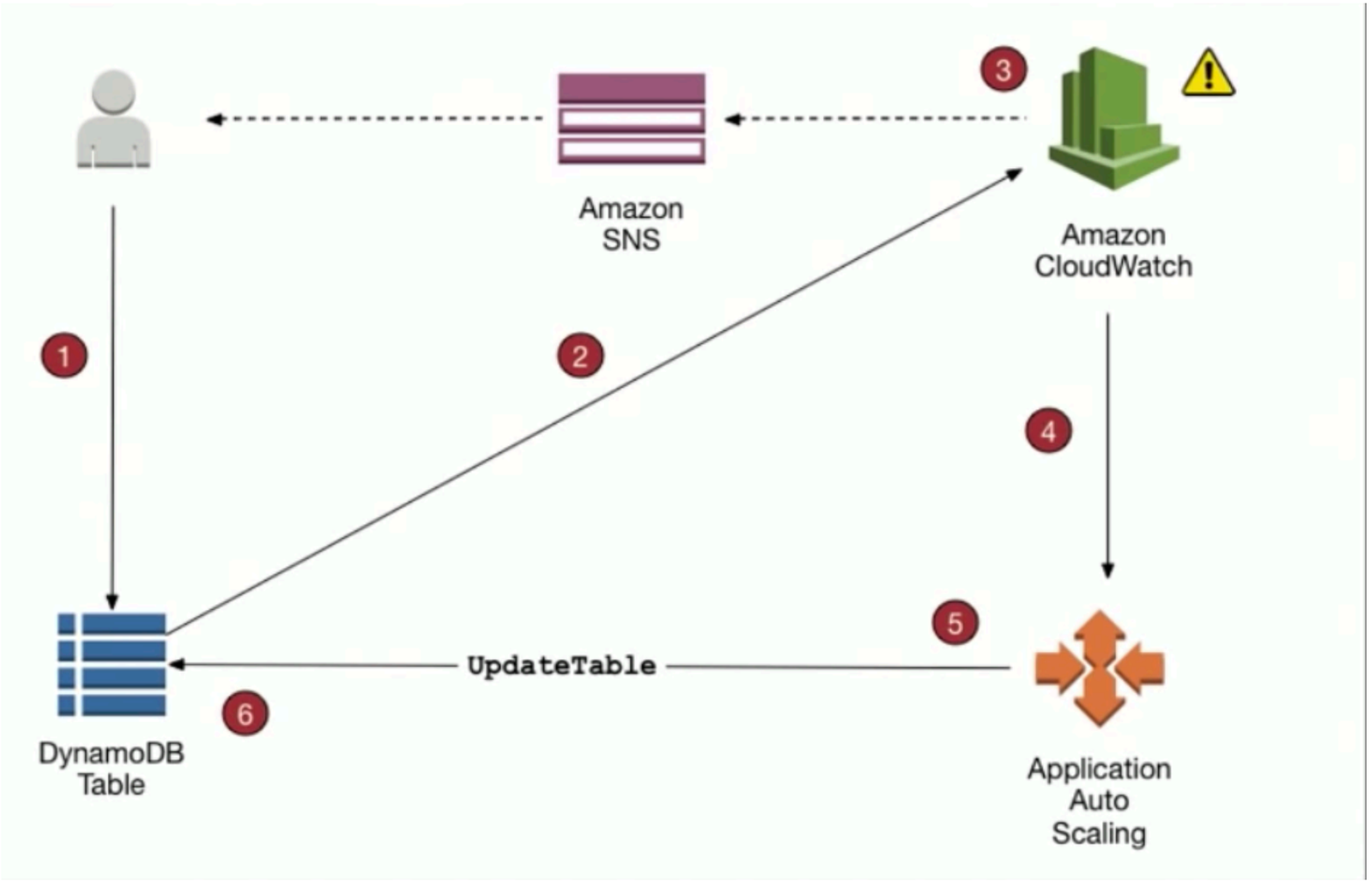


Adaptive Capacity

- Imbalances in key space: Ex: Imagine building a DB for census with primary key province/city. You may be < your total RCU amount but might be exceeding it for a partition - NYC.
- DynamoDB assigns each partition a dynamic multiplier(PID Controller) and automagically doesn't throttle you as long as your total < provisioned.



Auto Scaling



- DynamoDB also allows you to configure min, max provision capacity and target utilization. It then uses the consumption metrics automatically scale your provision between these.

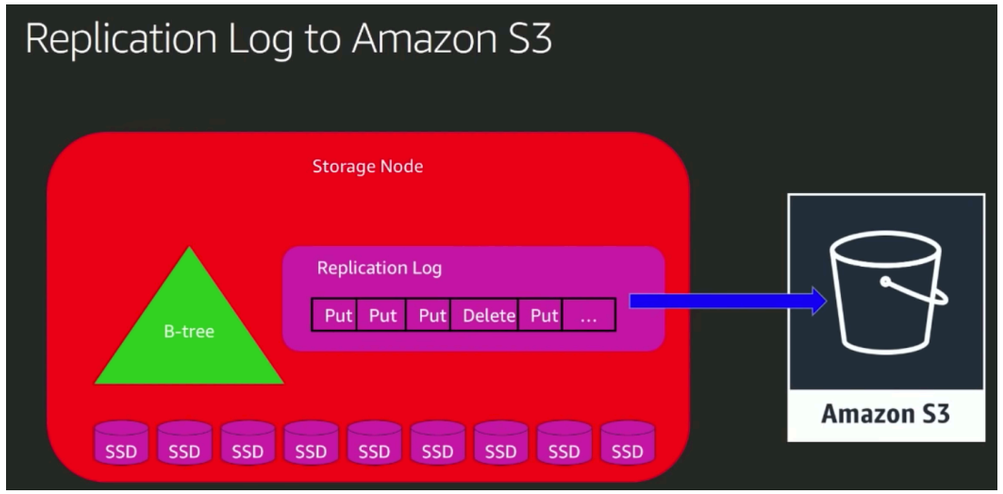
Provisioning

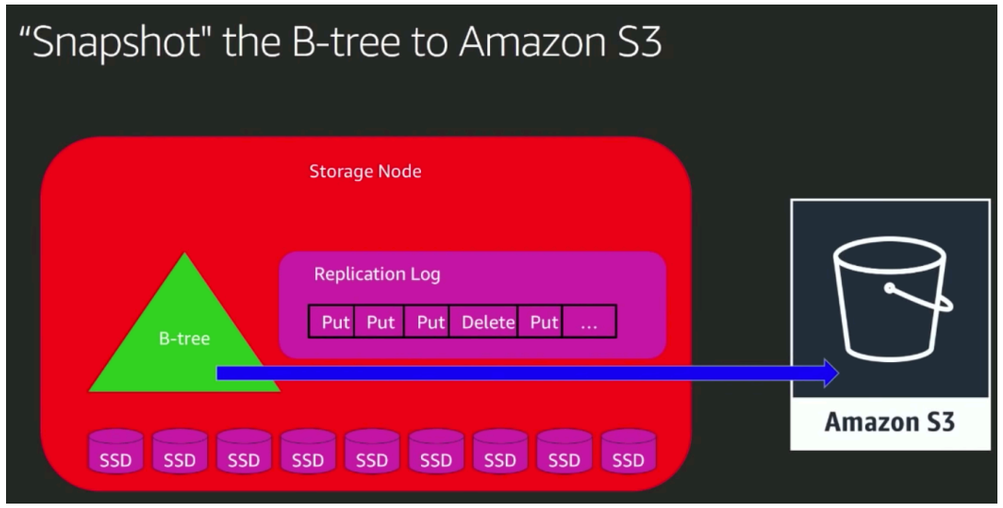
- Imbalances:
  - Imbalance in time: Bursting
  - Imbalances in key space: Adaptive Capacity
  - Continuous combination of above: Auto Scaling

Backup & Restore

Point In Time Restore

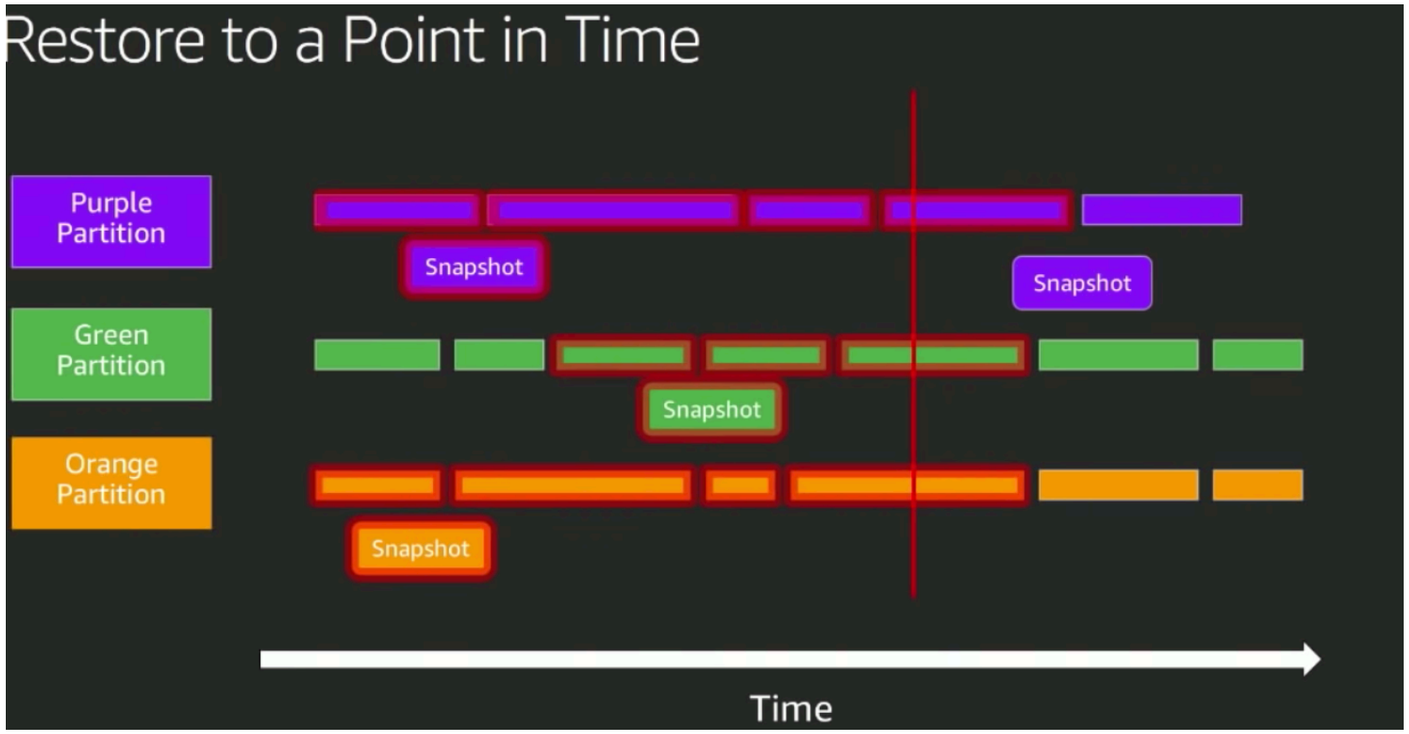
- In order to facilitate PITR, each partition backs up its replication log and a snapshot of the B tree to S3.
- These processes are independent of each other.





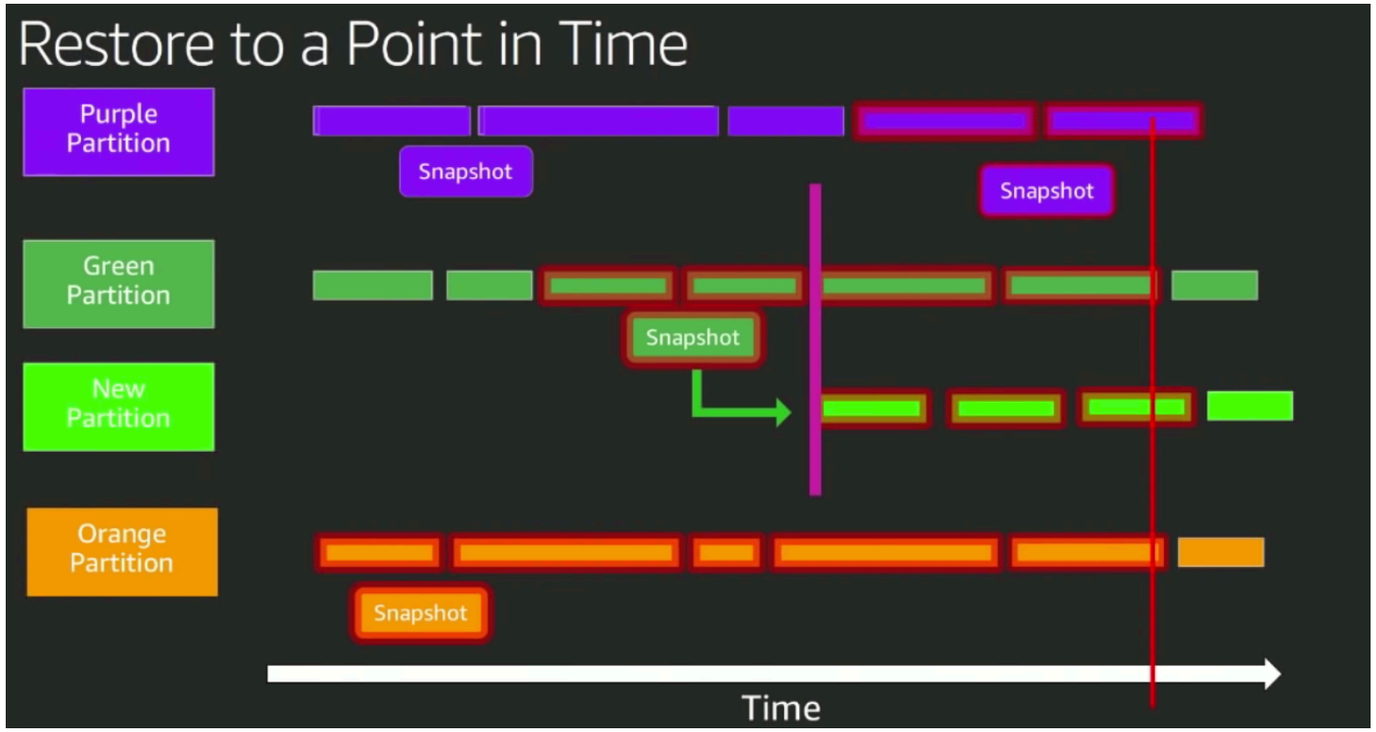
Point In Time Restore

- In order to restore at a point in time, for each partition we identify the last snapshot and replication logs up until right after the point in time restore.
- We then apply the updates to each snapshot



Point In Time Restore

- What if a new partition was created for which we don't have snapshots?
- We apply logs till before the partition and then split the data based on the key space so as to recreate the partition and then apply logs from each partition.



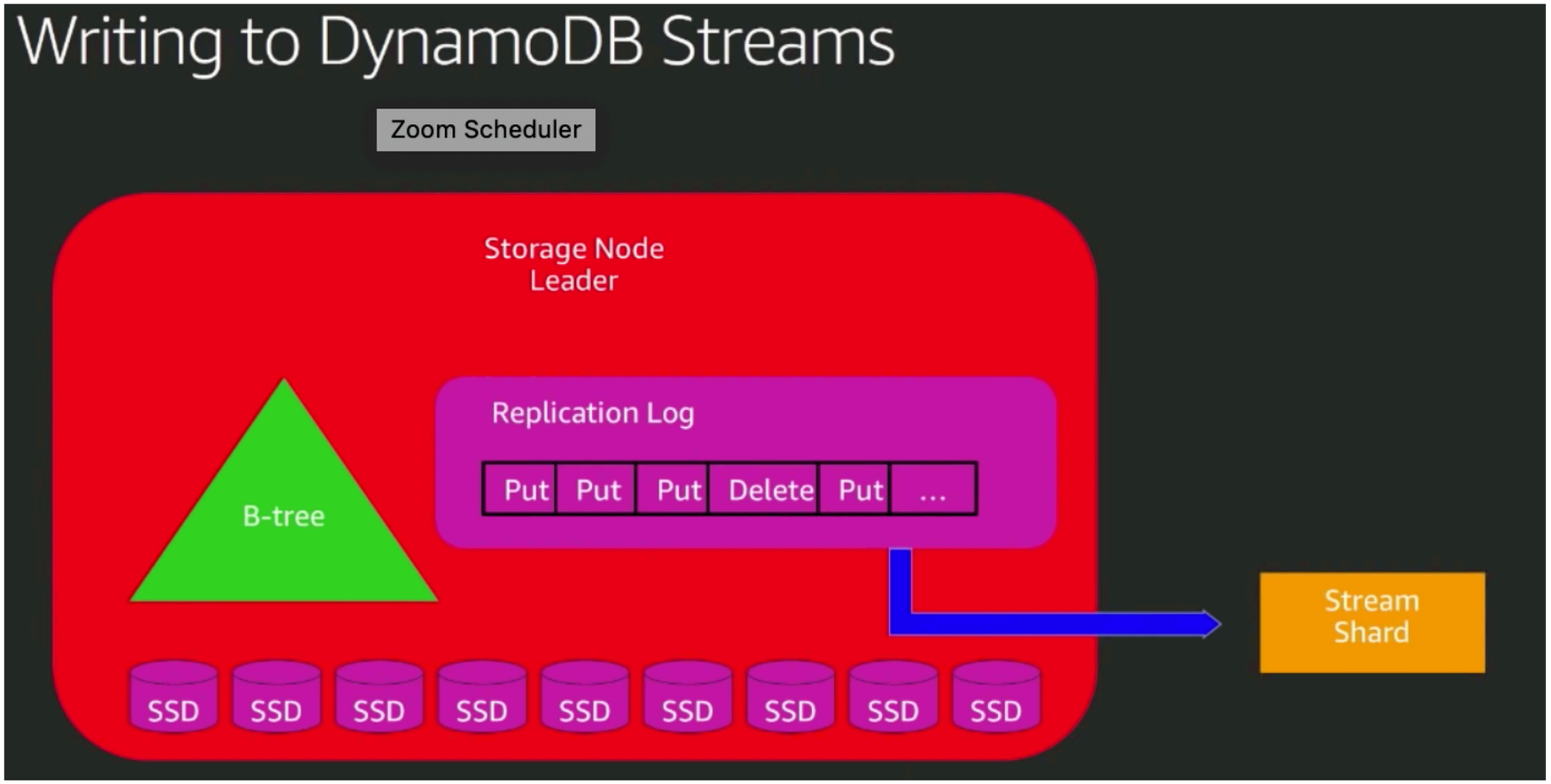
Ondemand Restore

- Ondemand restores are basically PITR with the exception that we force the partitions to update their logs to S3 to the current time.

Dynamo Streams

Dynamo Streams

- All table mutations
- No duplicates
- In order (by key)

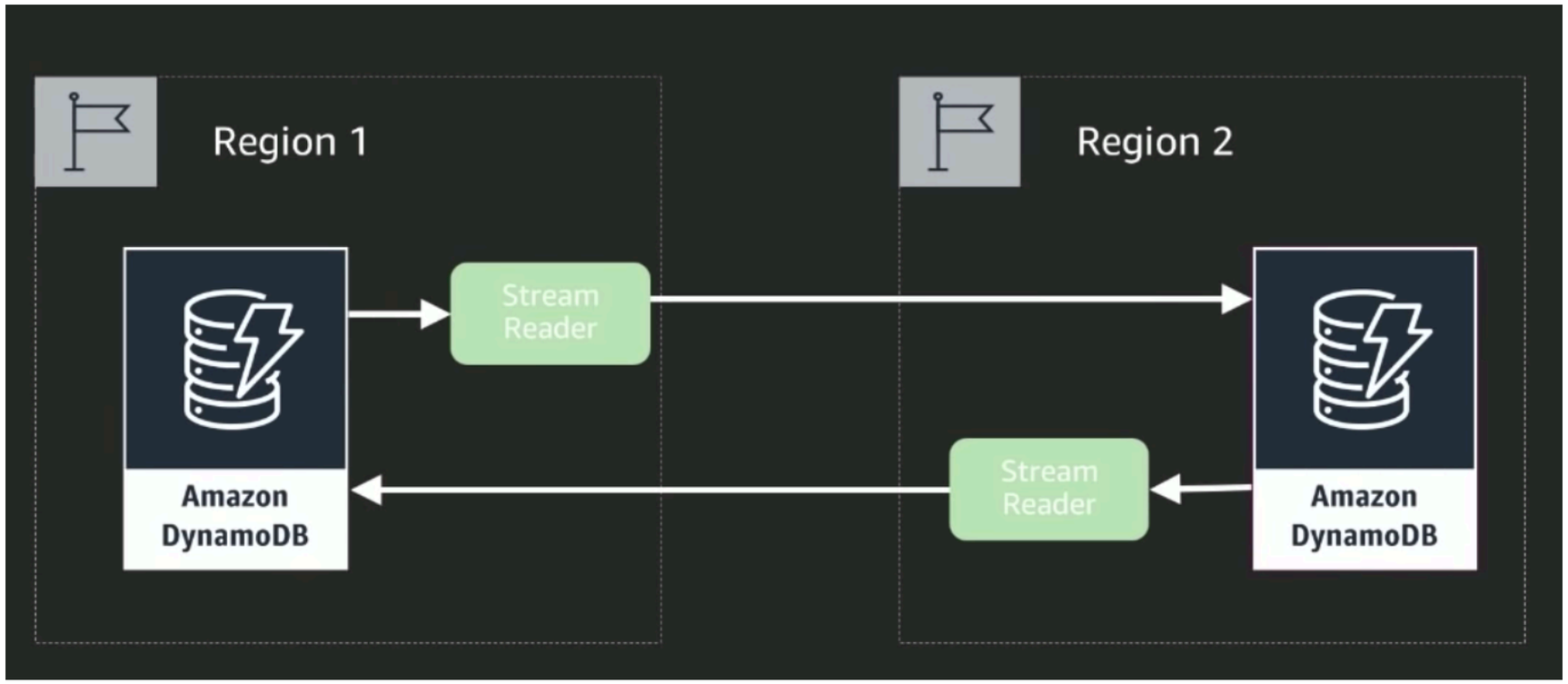


Triggers

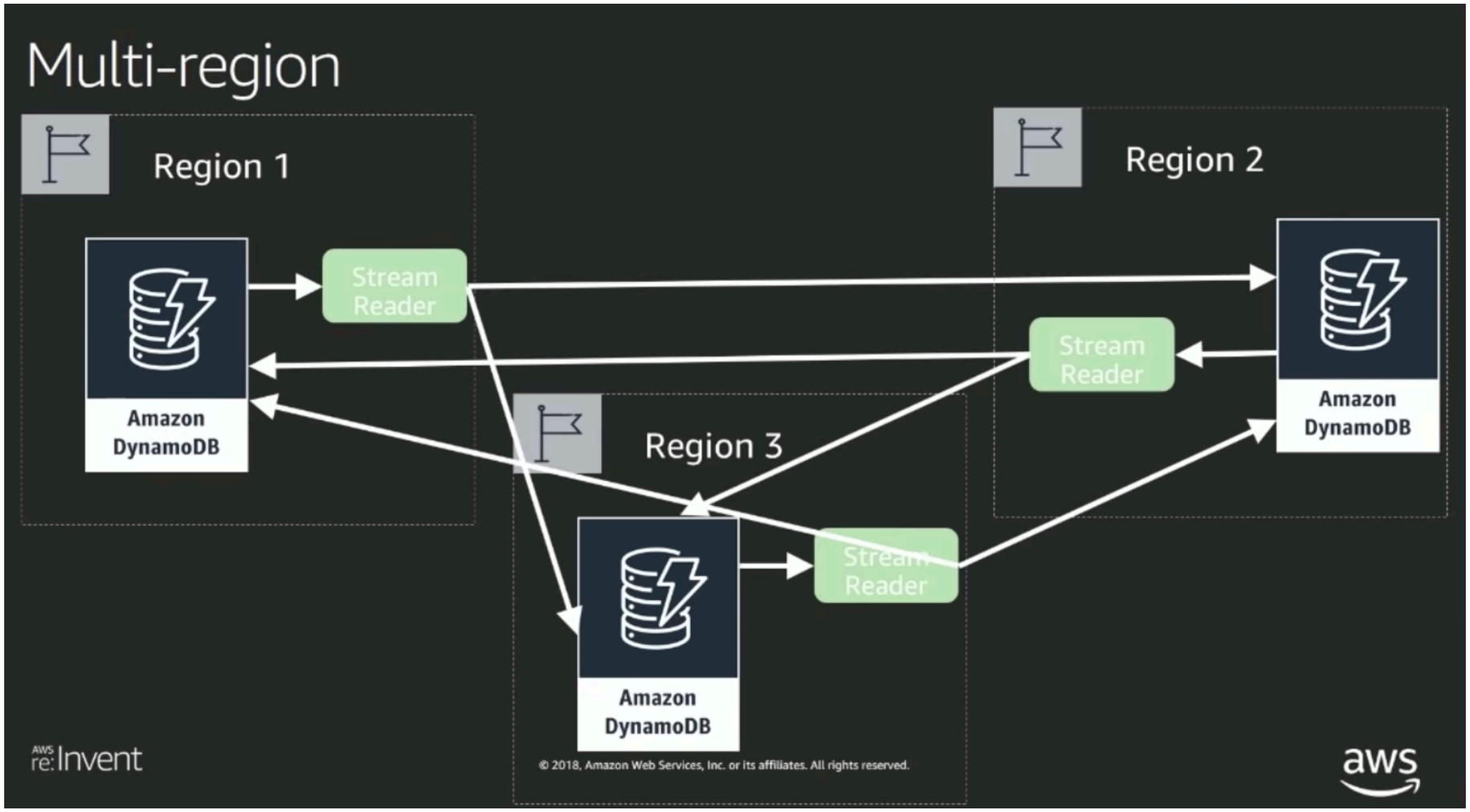
- DynamoDB allows you to define your own triggers by using AWS Lambda functions set to trigger on streams

Global Tables

Multi Master



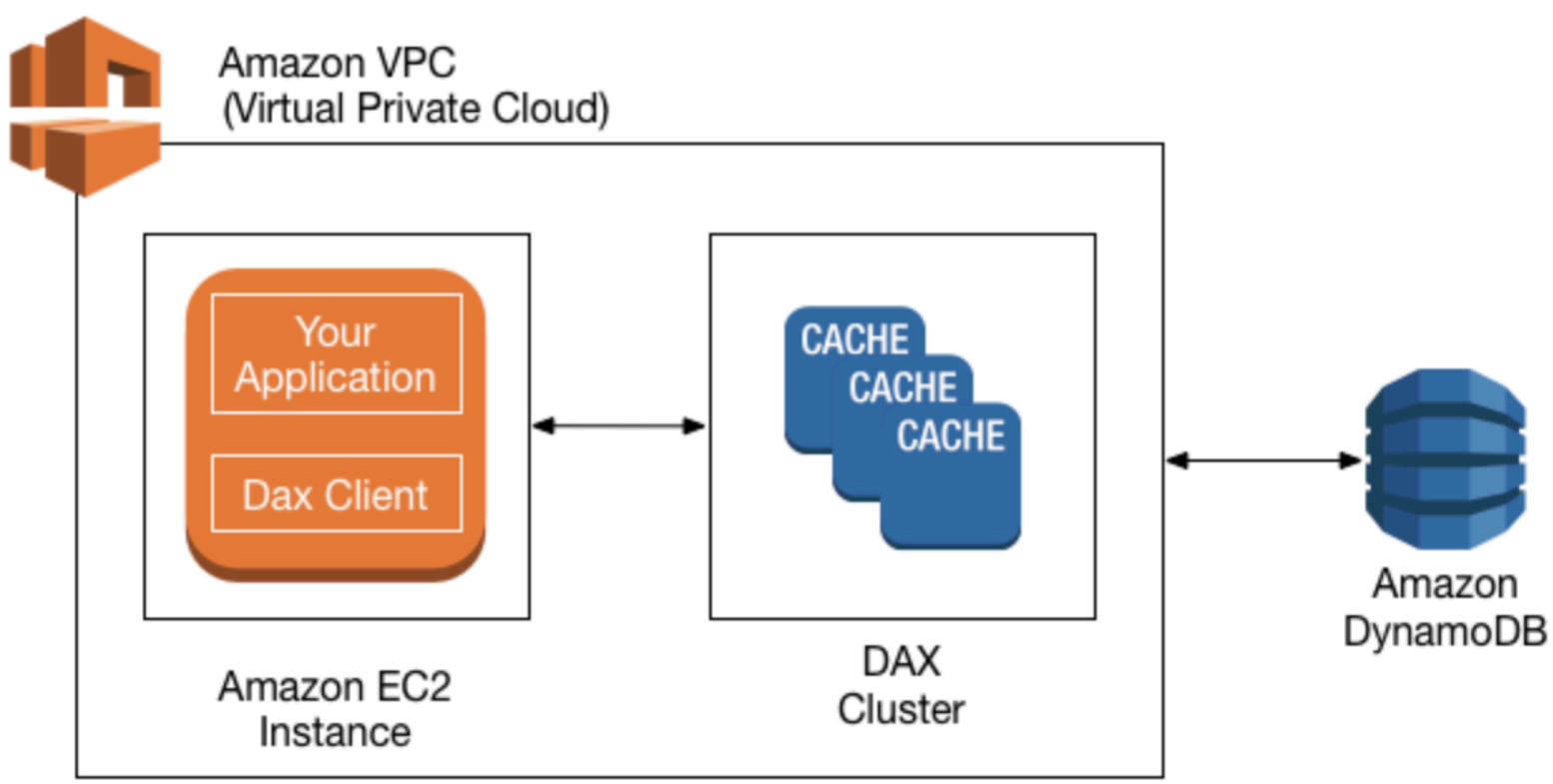
Multi Region



DAX

DAX

- In-memory write through cache layer built on top of DynamoDB.
- Sub-single digit performance for queries where eventual consistency is fine.
- Allows for quicker queries and scans
- Support for both TTL and LRU cache.
- No write back support yet



DynamoDB Limits

DynamoDB Limits

- You can define a maximum of 5 local secondary indexes.
- There is an initial limit of 20 global secondary indexes per table
- The minimum length of a partition key value is 1 byte. The maximum length is 2048 bytes
- The maximum item size in DynamoDB is 400 KB, which includes both attribute name binary length (UTF-8 length) and attribute value lengths (again binary length). The attribute name counts towards the size limit.

En fin.

Dynamo

- Incremental scalability: Dynamo should be able to scale out one storage host (or “node”) at a time, with minimal impact on both operators of the system and the system itself.
- Symmetry: Every node in Dynamo should have the same set of responsibilities as its peers; there should be no distinguished node or nodes that take special roles or extra set of responsibilities.
- Decentralization: An extension of symmetry, the design should favor decentralized peer-to-peer techniques over centralized control.
- Heterogeneity: The system should be able to exploit heterogeneity in the infrastructure it runs on. For example, the work distribution must be proportional to the capabilities of the individual servers. This is essential in adding new nodes with higher capacity without having to upgrade all hosts at once.

Dynamo

Dataset partitioning	Consistent Hashing	Incremental, possibly linear scalability in proportion to the number of collaborating nodes.
Highly available writes	Vector Clock or Dotted-Version-Vector Sets, 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 tree	Can be used to identify differences between replica owners and synchronize divergent replicas pro-actively.
Membership and failure detection	Gossip-based membership protocol and failure detection	Avoids having a centralized registry for storing membership and node liveness information, preserving symmetry.

SimpleDB

“With Dynamo we had taken great care to build a system that met the requirements of our engineers. After evaluations, it was often obvious that Dynamo was ideal for many database use cases. But ... we learned that engineers found the prospect of running a large software system daunting and instead looked for less ideal design alternatives that freed them from the burden of managing databases and allowed them to focus on their applications.”

SimpleDB

Domain scaling limitations. SimpleDB requires customers to manage their datasets in containers called Domains, which have a finite capacity in terms of storage (10 GB) and request throughput. Although many customers worked around SimpleDB's scaling limitations by partitioning their workloads over many Domains, this side of SimpleDB is certainly not simple. It also fails to meet the requirement of incremental scalability, something that is critical to many customers looking to adopt a NoSQL solution.

Predictability of Performance. SimpleDB, in keeping with its goal to be simple, indexes all attributes for each item stored in a domain. While this simplifies the customer experience on schema design and provides query flexibility, it has a negative impact on the predictability of performance. For example, every database write needs to update not just the basic record, but also all attribute indices (regardless of whether the customer is using all the indices for querying). Similarly, since the Domain maintains a large number of indices, its working set does not always fit in memory. This impacts the predictability of a Domain's read latency, particularly as dataset sizes grow.
Consistency. SimpleDB's original implementation had taken the "eventually consistent" approach to the extreme and presented customers with consistency windows that were up to a second in duration. This meant the system was not intuitive to use and developers used to a more traditional database solution had trouble adapting to it. The SimpleDB team eventually addressed this issue by enabling customers to specify whether a given read operation should be strongly or eventually consistent.

Pricing complexity. SimpleDB introduced a very fine-grained pricing dimension called "Machine Hours." Although most customers have eventually learned how to predict their costs, it was not really transparent or simple.

References

- DynamoDB - Wikipedia
- AWS
- Dynamo
- SimpleDB
- CAP Theorem
- DynamoDB - AWS Docs
- DAX
- Vogel's blog
- History of databases
- AWS reinvent: DynamoDB Talk
- Raft Consensus
- Paxos Consensus