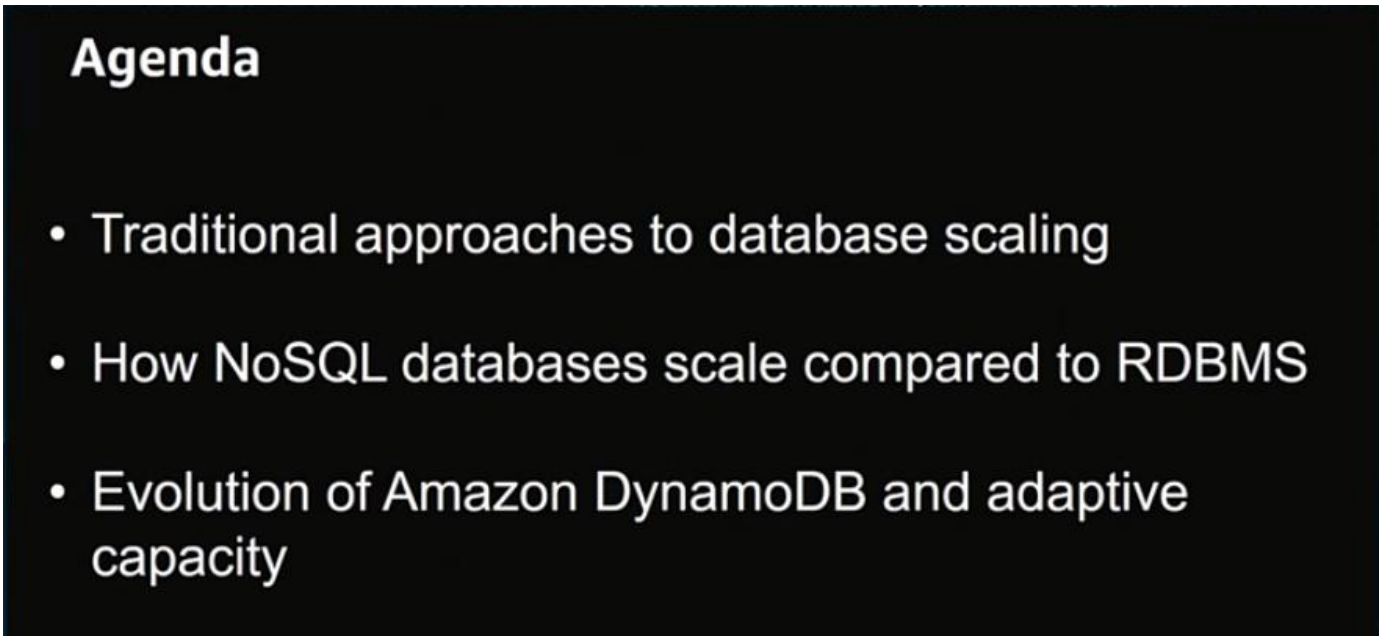
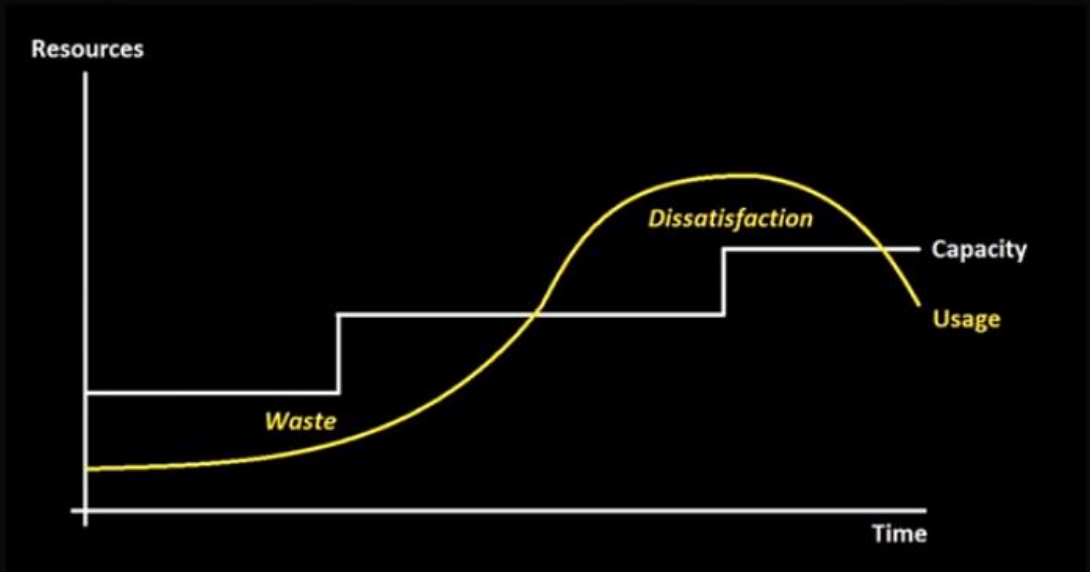
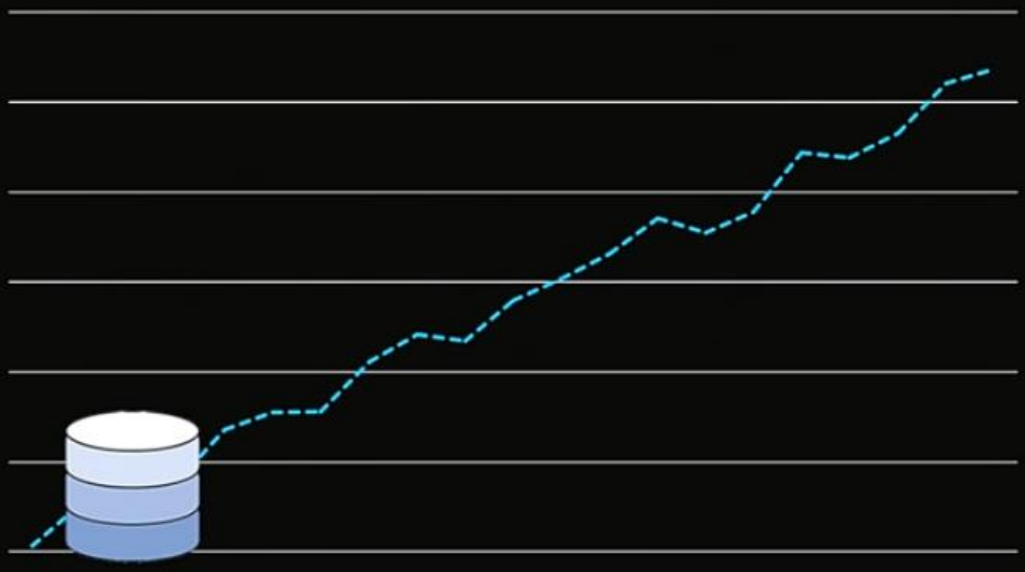
Database capacity planning is critical to running your business, but it’s also hard. In this session we’ll compare how scaling is usually performed for relational databases and NoSQL databases. We’ll look behind the scenes at how DynamoDB shards your data across multiple partitions and servers. Finally, we’ll talk about some of the recent enhancements to DynamoDB that make scaling even simpler, particularly a new feature called adaptive throughput that eliminates much of the throttling issues that you may have experienced.

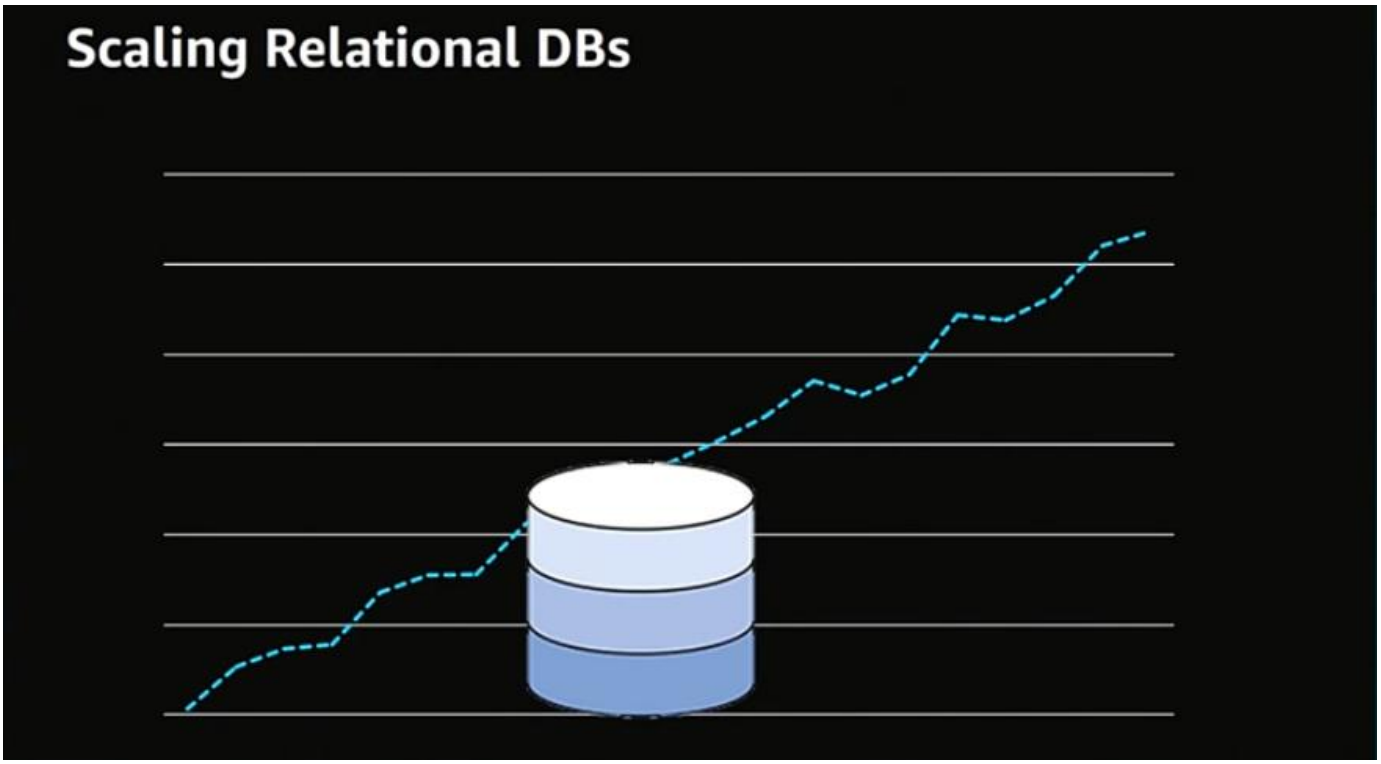


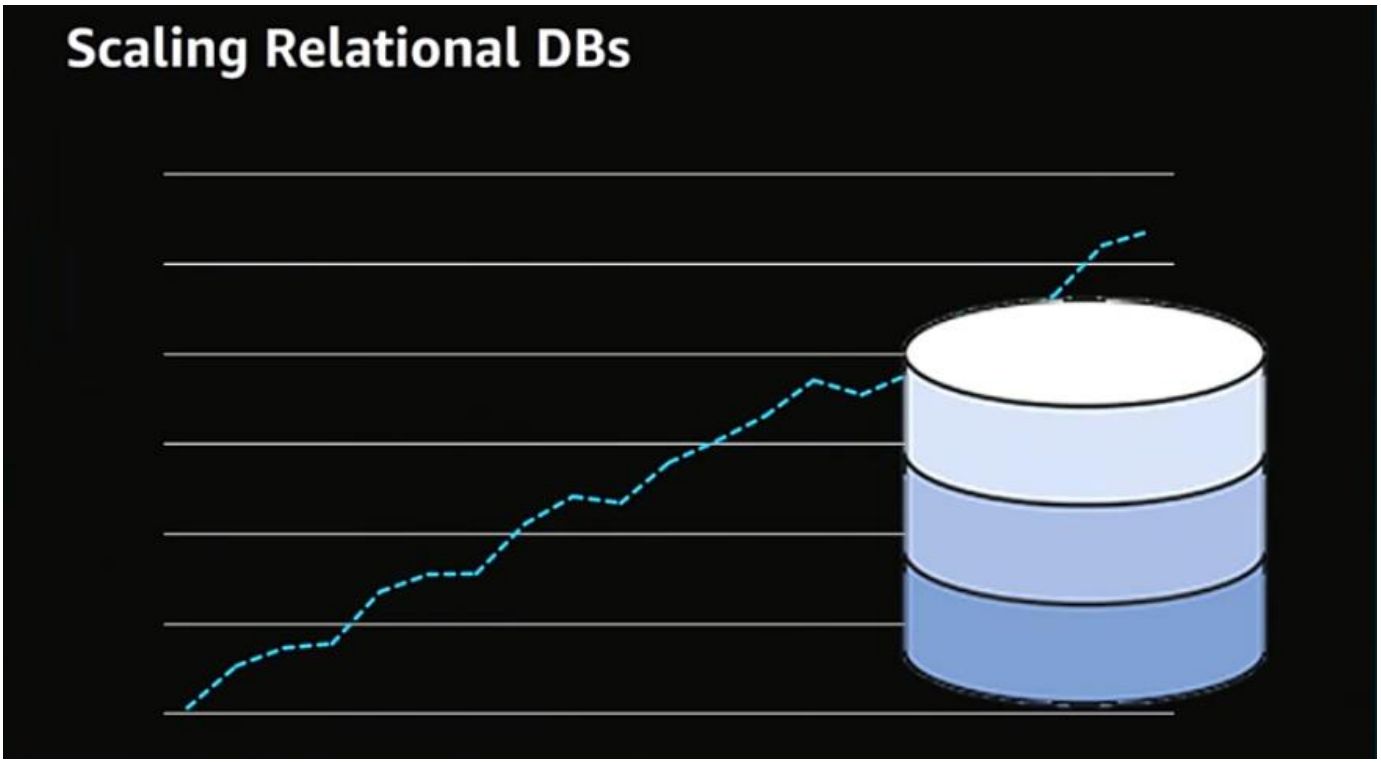
Capacity Planning Is Important (and Hard)



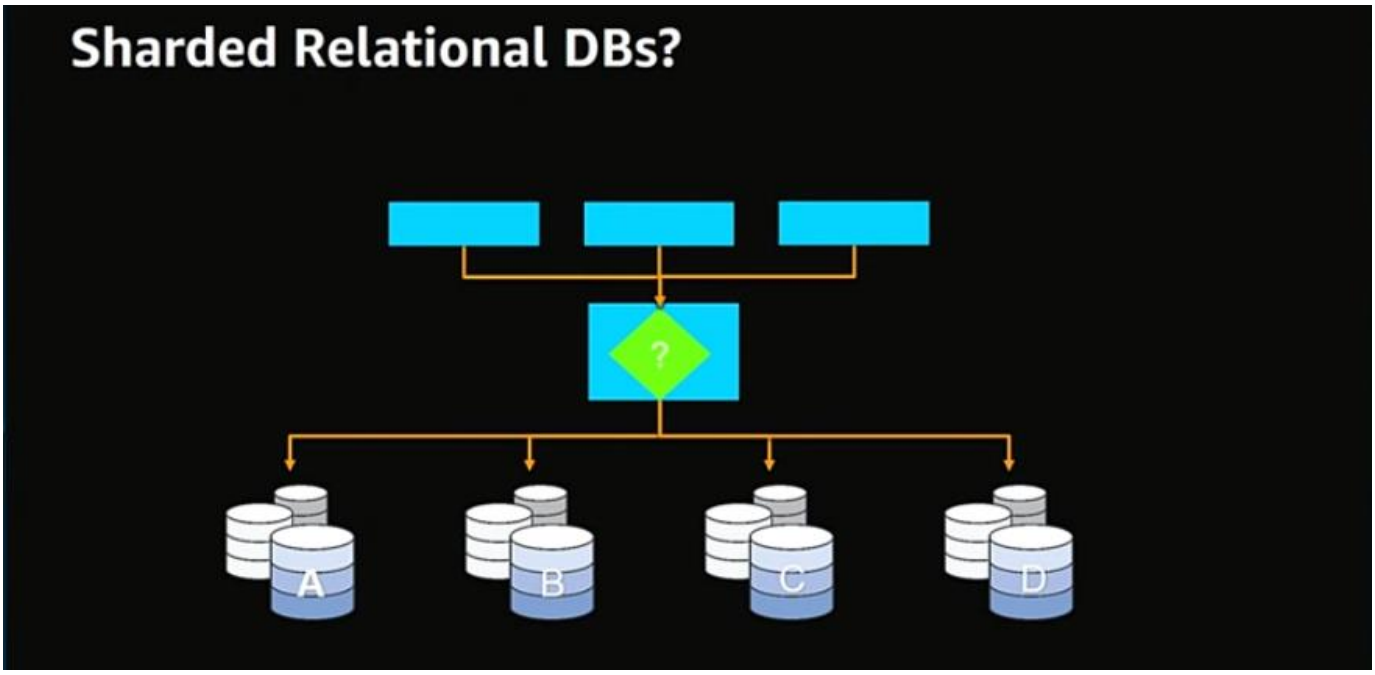
Scaling Relational DBs



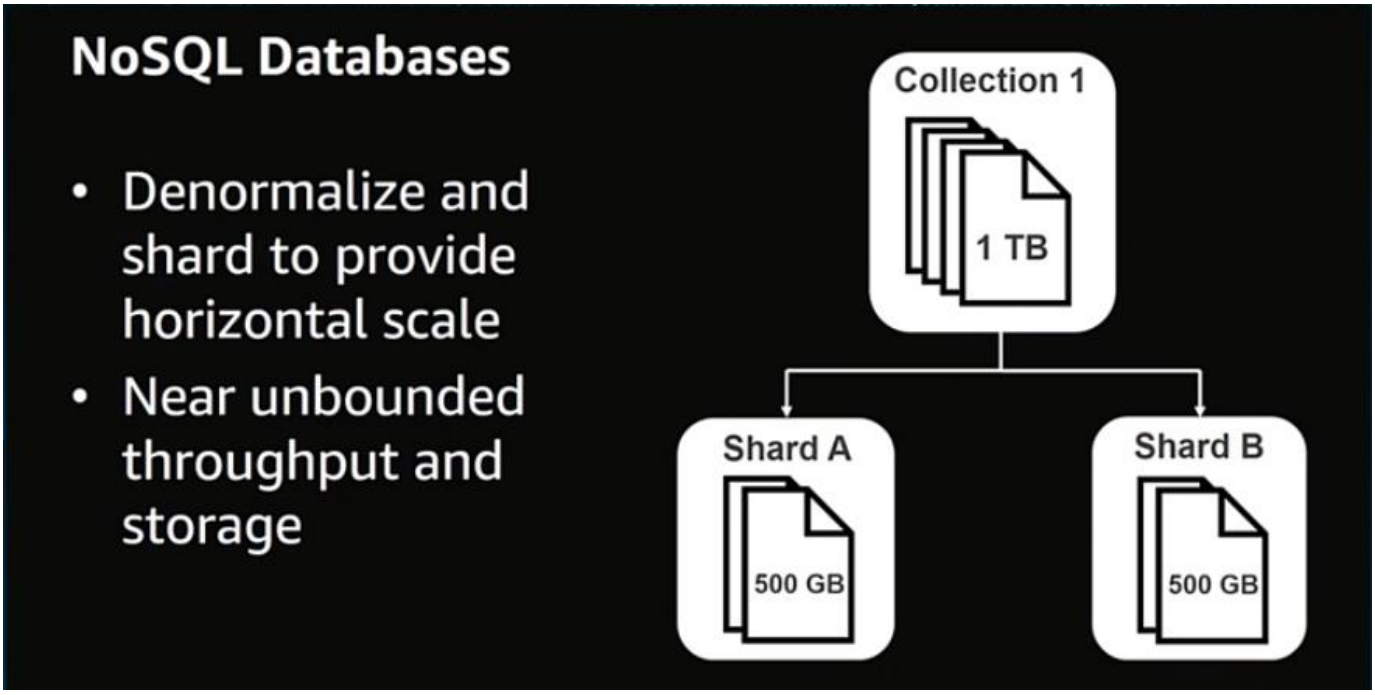




There are several very large single database rack deployments that companies have implemented as data gets bigger

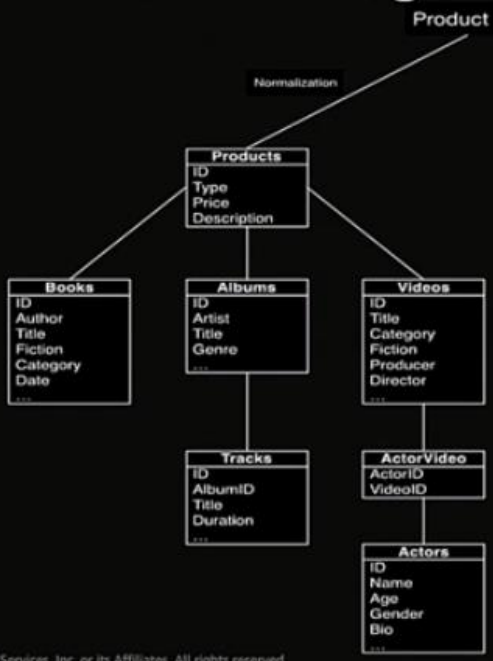


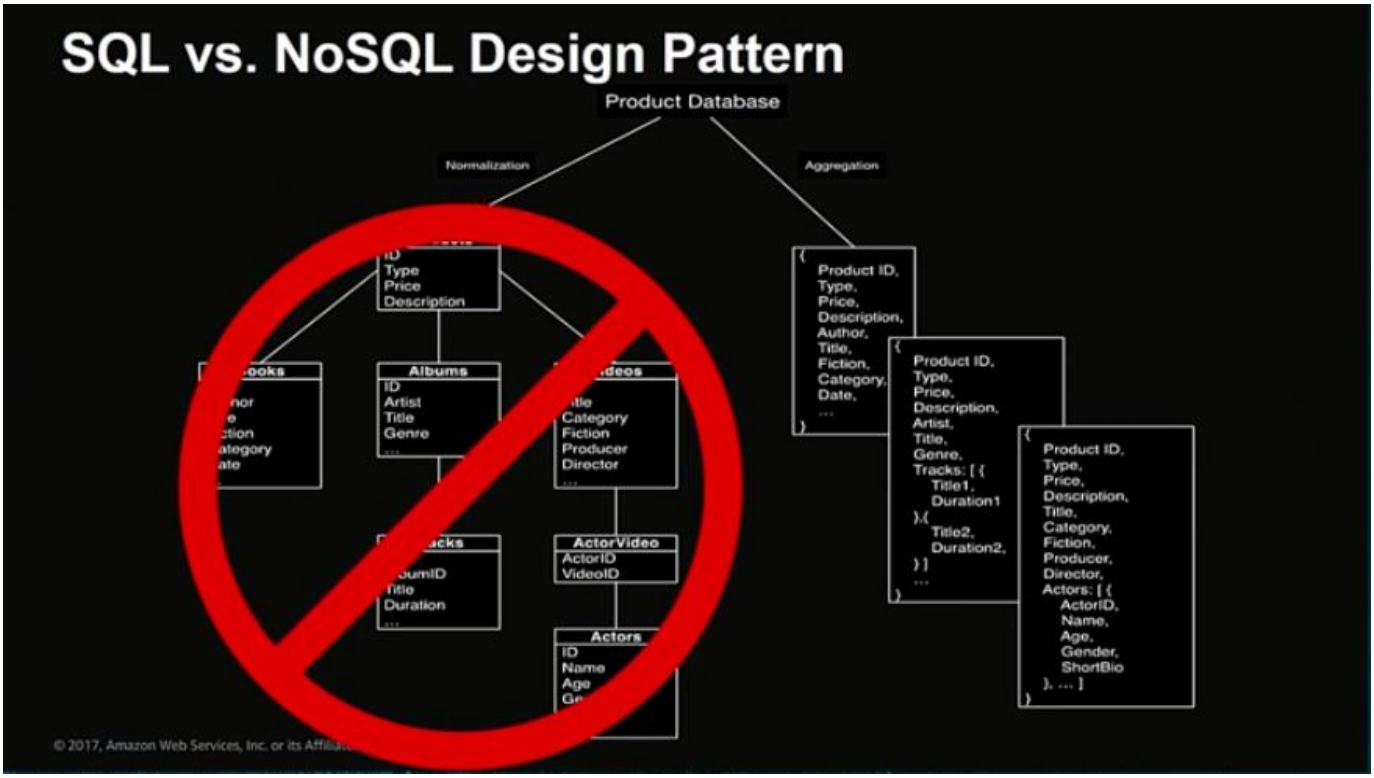
As the data gets even bigger, then you start to partition the database maybe by time or segment/categories to spread your queries as above



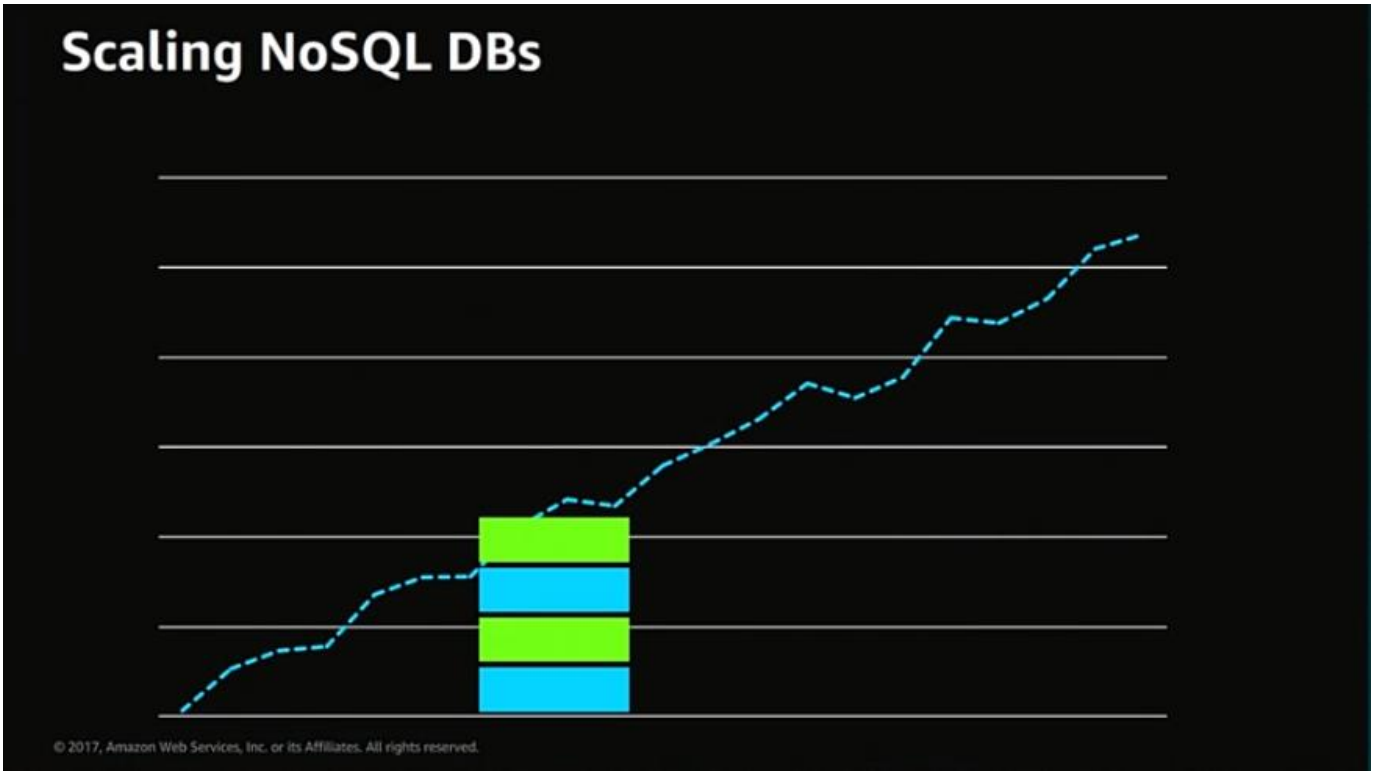
The Iron Triangle of Data—All About CAP

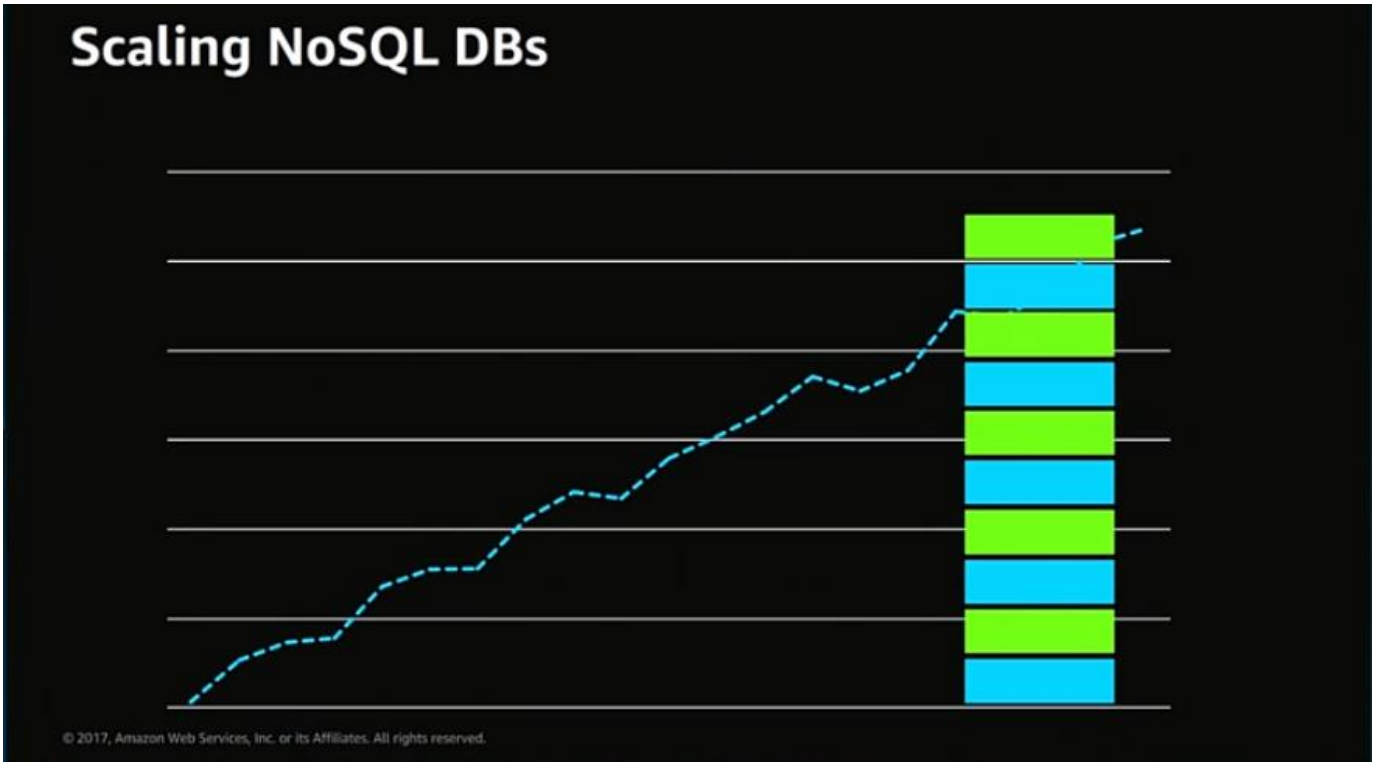
SQL vs. NoSQL Design Pattern





We can use a denormalized datastore by trying to flatten the datastore and push all our data into a set of documents items or collections.

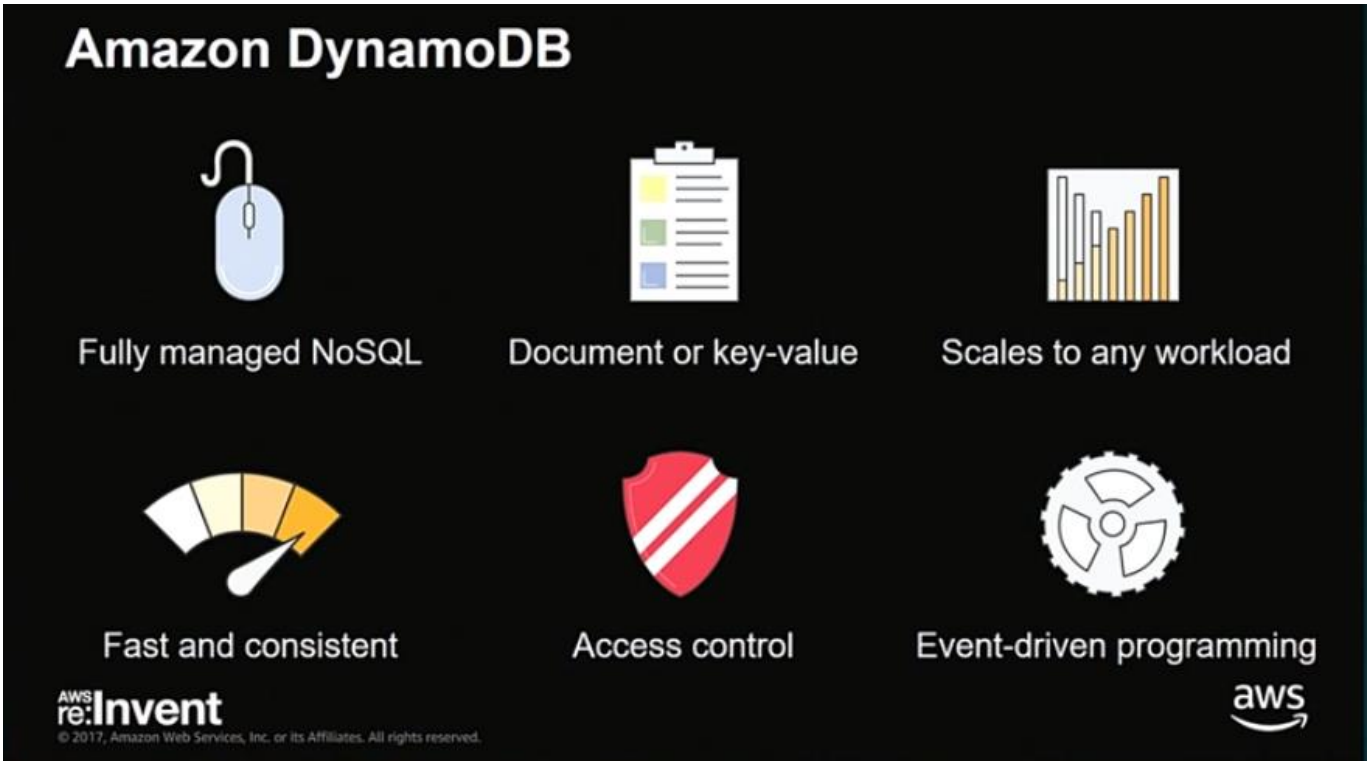


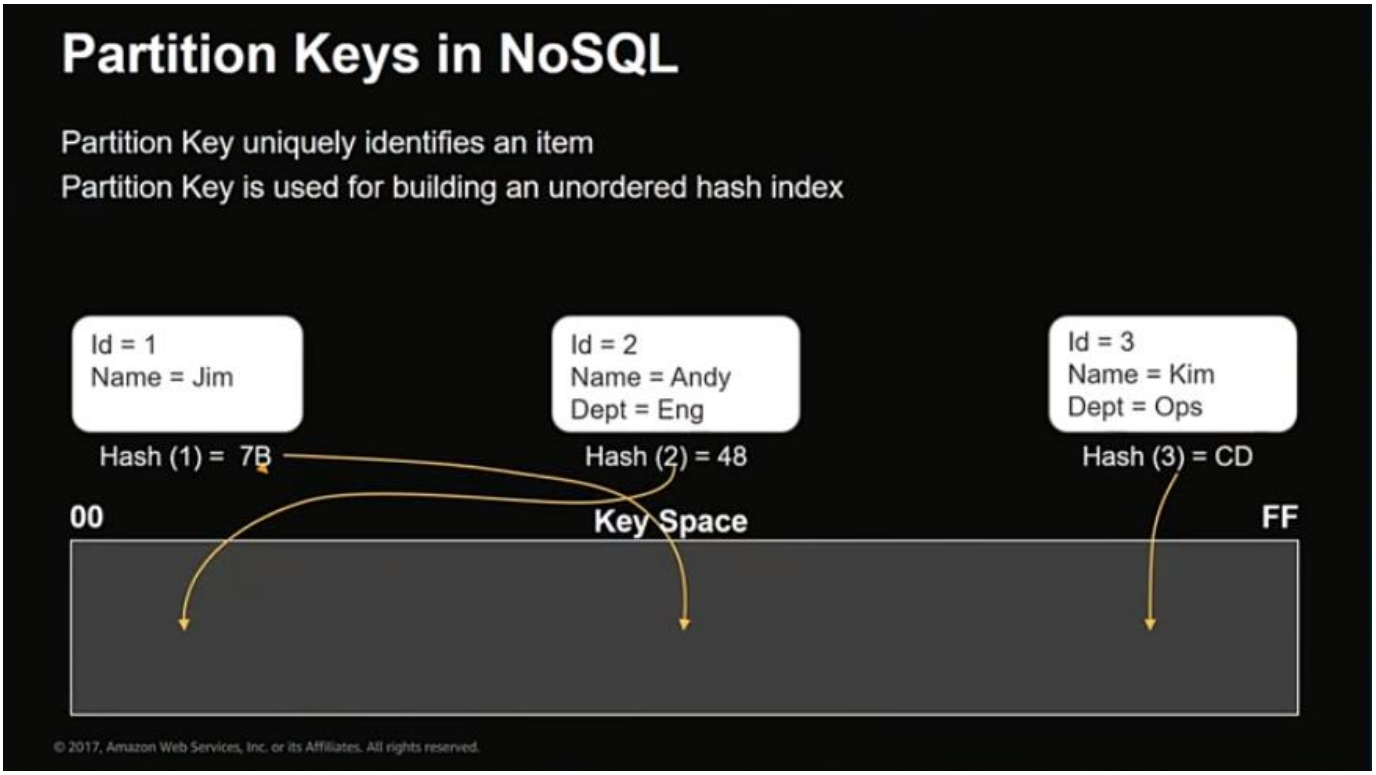


There is a replication process or shard as you do this incremental process of replicating data from all the shards in a NoSQL database

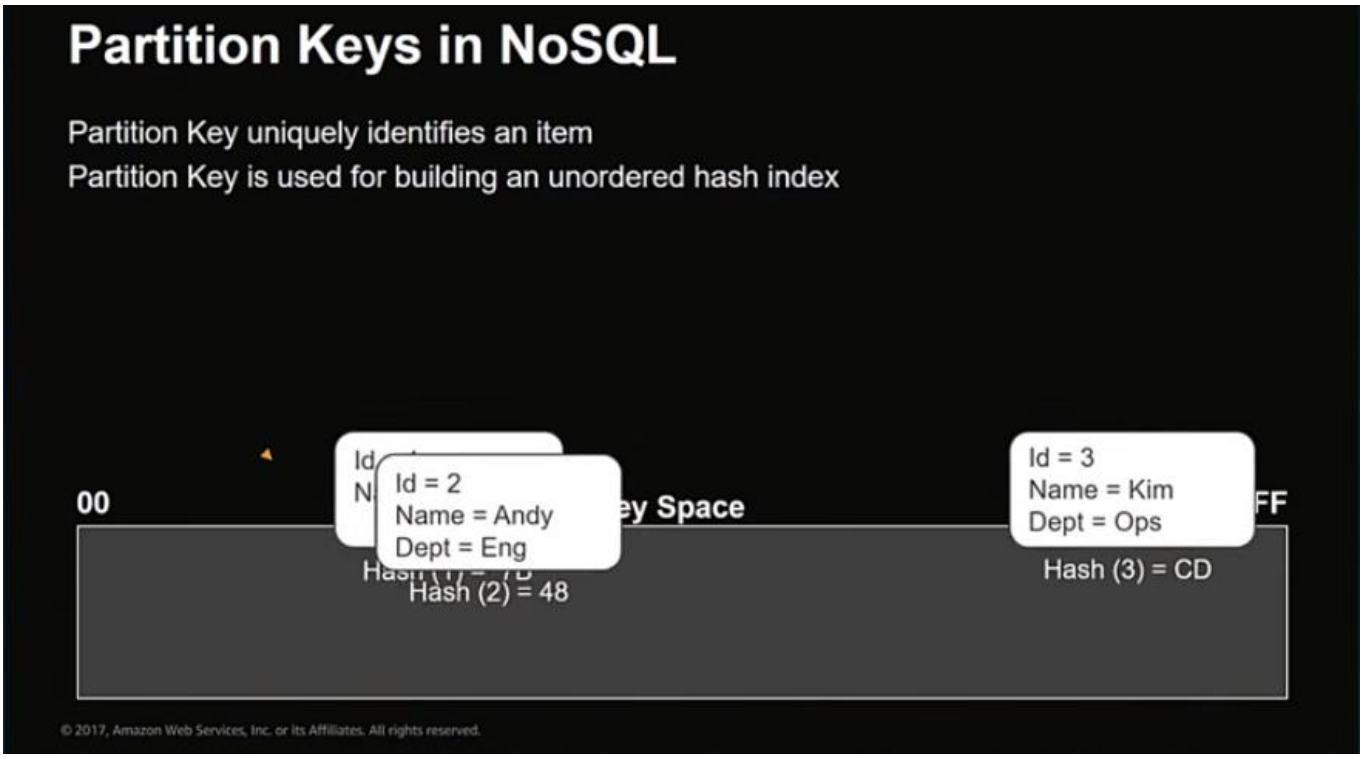
Why NoSQL?

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

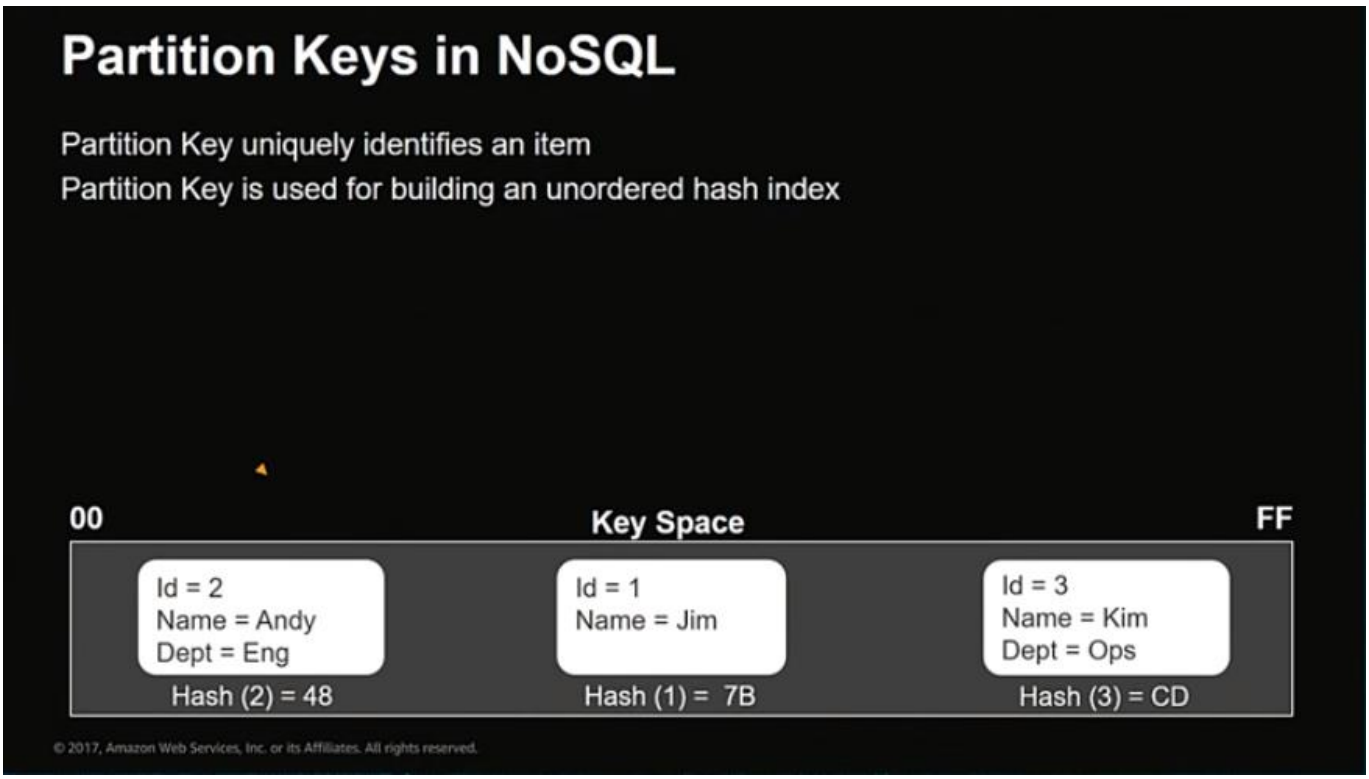




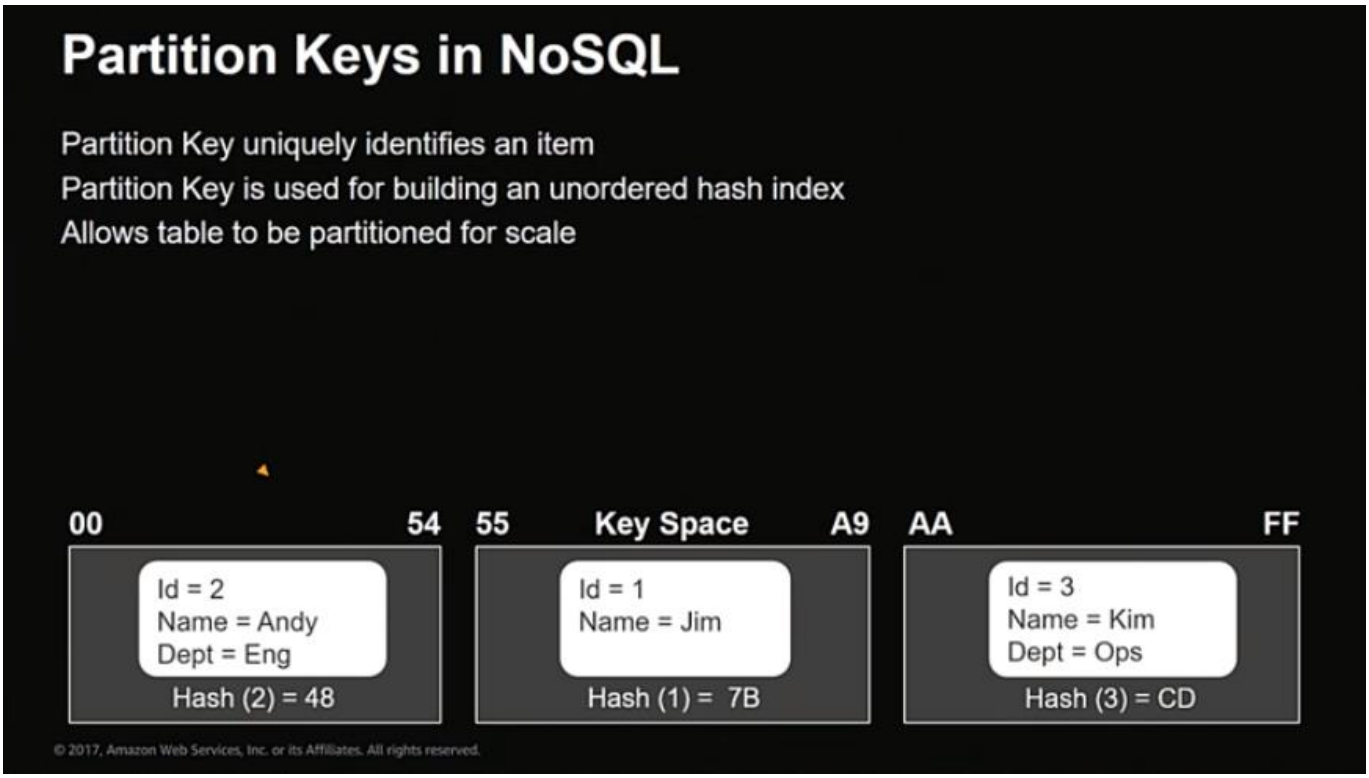
Data is distributed in a NoSQL DB using some shard/partition key/attribute that is mandatory for all items, it uniquely identifies that particular item.



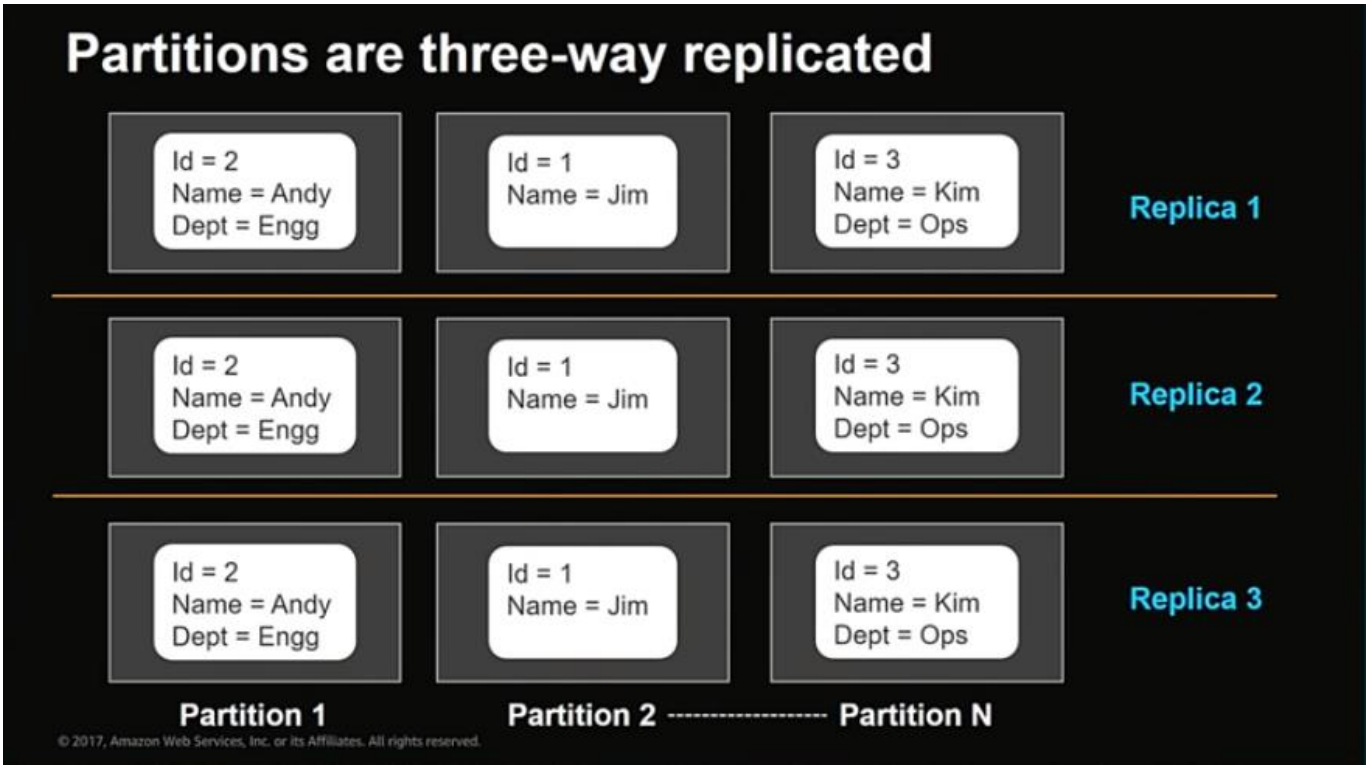
What we do is to hash the unique value and create an unordered item index and then lay out the values in an arbitrary key space as below

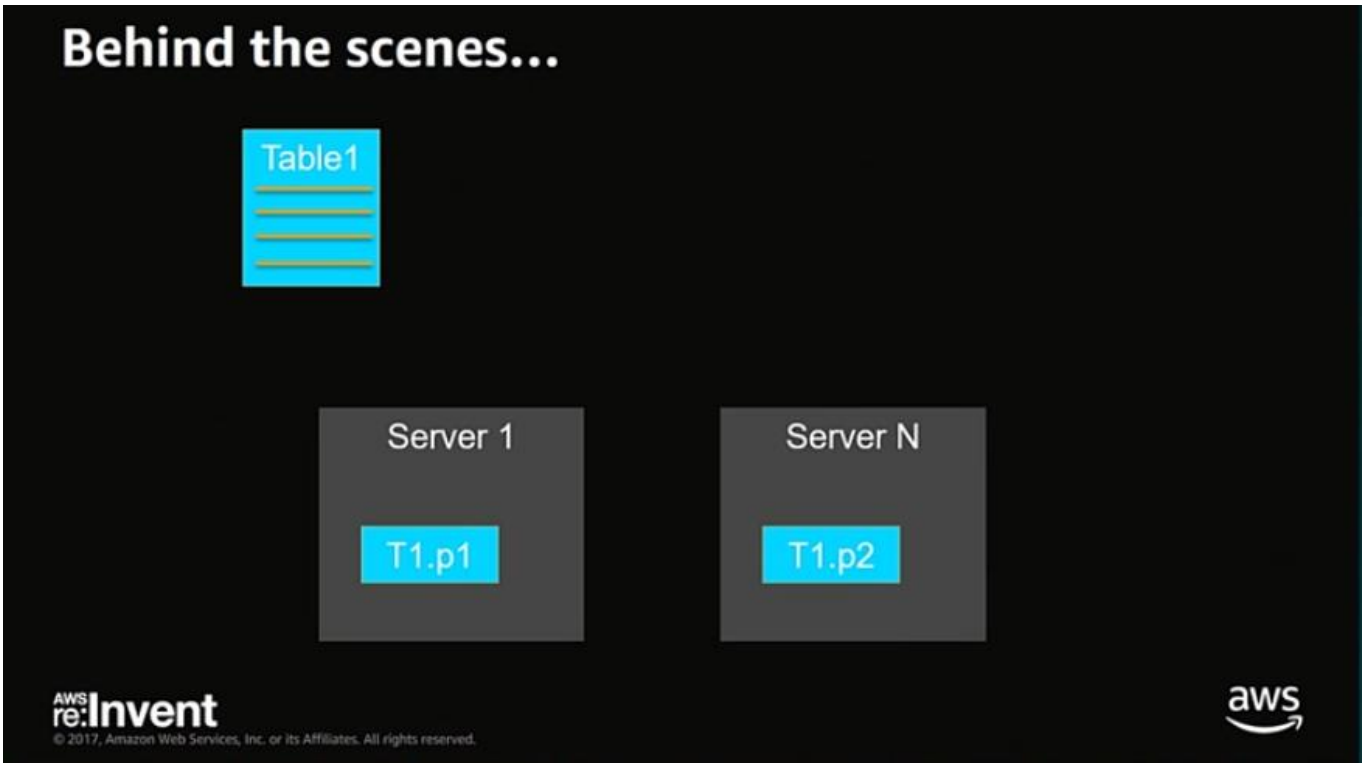


Once we've done this, as we add more items we are then going to start splitting up the arbitrary key space up into visible boxes/servers as below

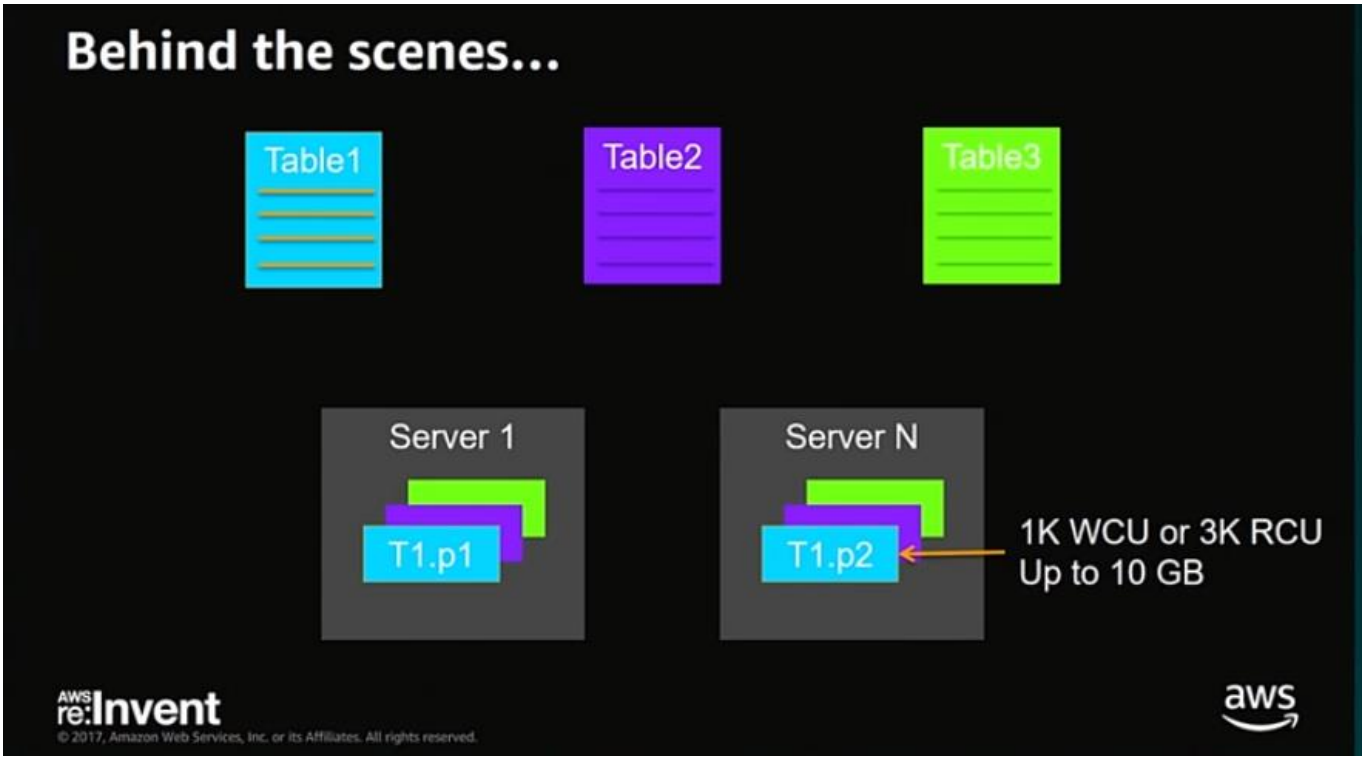


This is how all NoSQL DBs scale

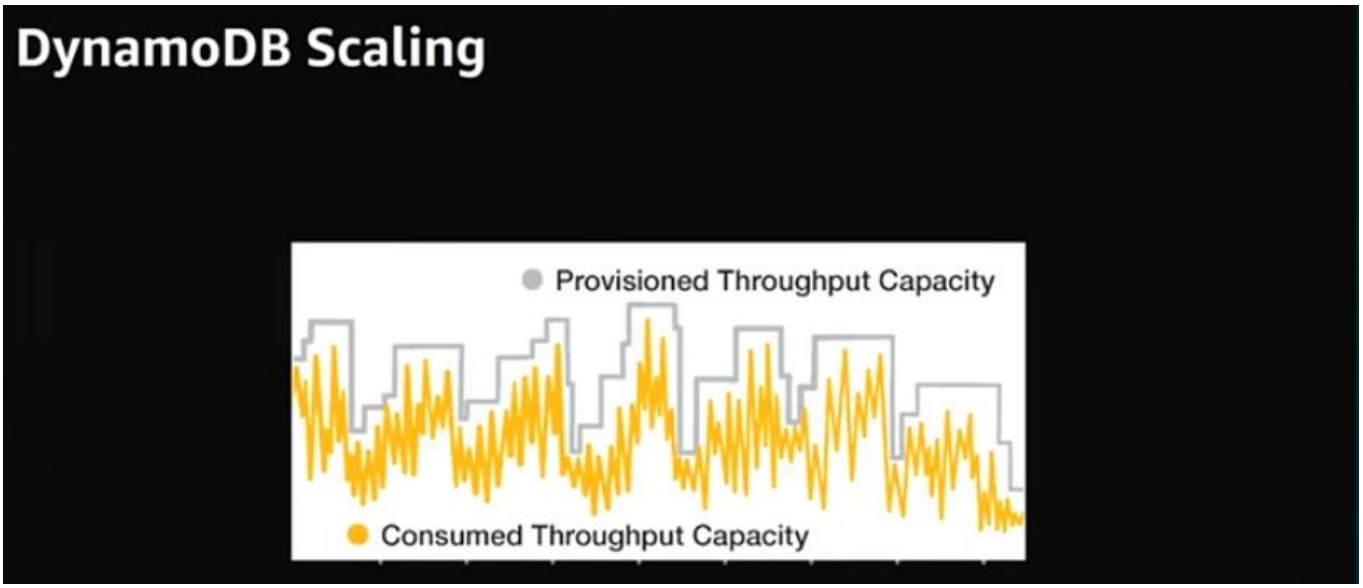




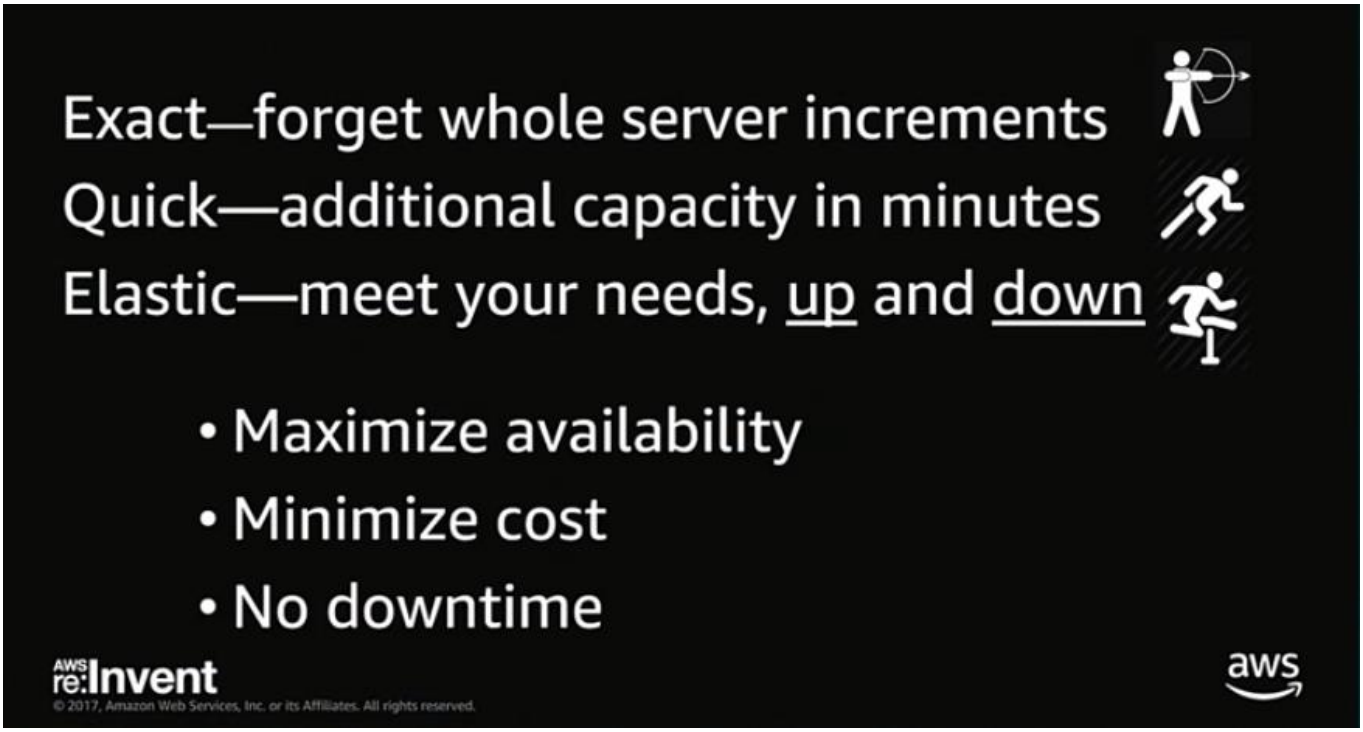
When you create a new DynamoDB table, AWS does not roll out new infrastructure for you. Instead they reclaim capacity for you within the Massively Multi-tenant infrastructure underlying the DynamoDB service.



It simply takes your table and your data, chop it up and spread around the infrastructure and share it with other user's tables as above. WCU is write capacity unit, RCU is Read Capacity unit. They both define a partition in DynamoDB.



In DynamoDB when the load rises you can simply dial up the provisioned capacity as needed and dial it down again when the load falls using DynamoDB's dynamic back plane.



Results

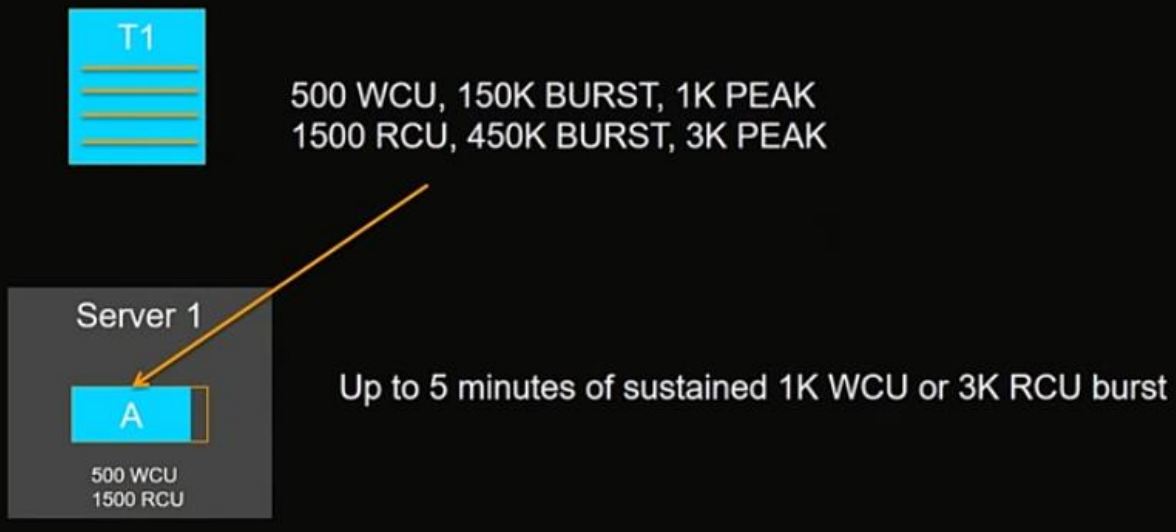
Hundreds of thousands of customers
Millions of requests per second
Hundreds of billions of items
Petabytes of storage

Adaptive Capacity

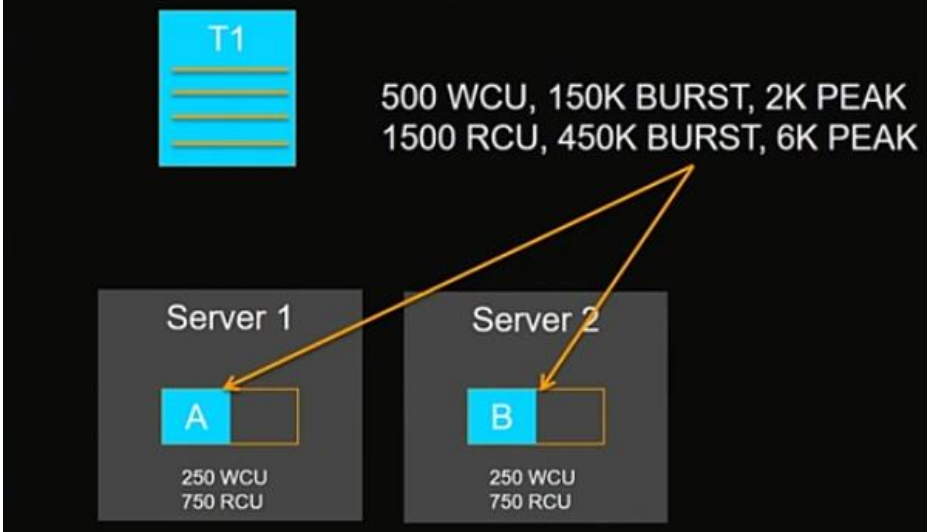
Get it right, nobody notices.

Get it wrong, everybody will.

Dynamic Scaling—Throughput and Storage

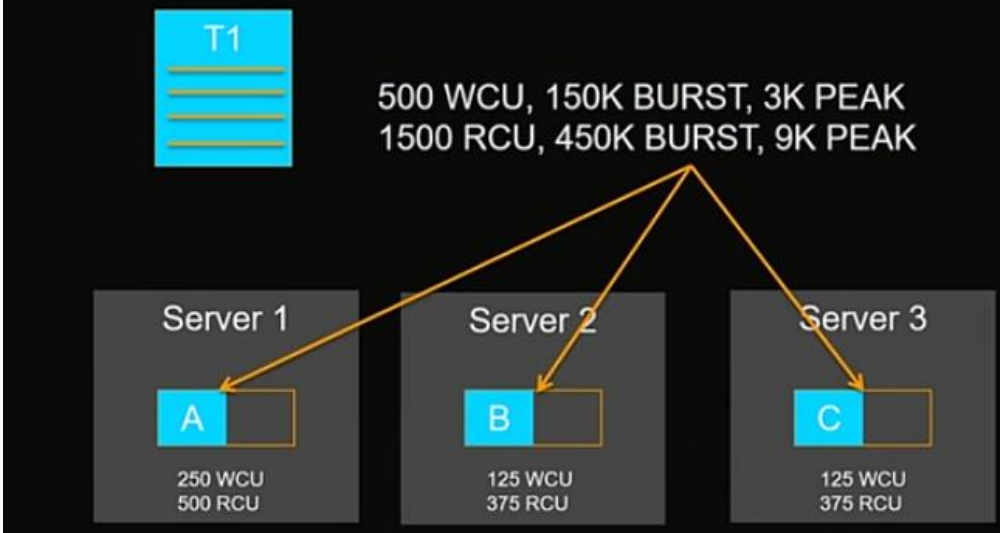


Dynamic Scaling—Throughput and Storage



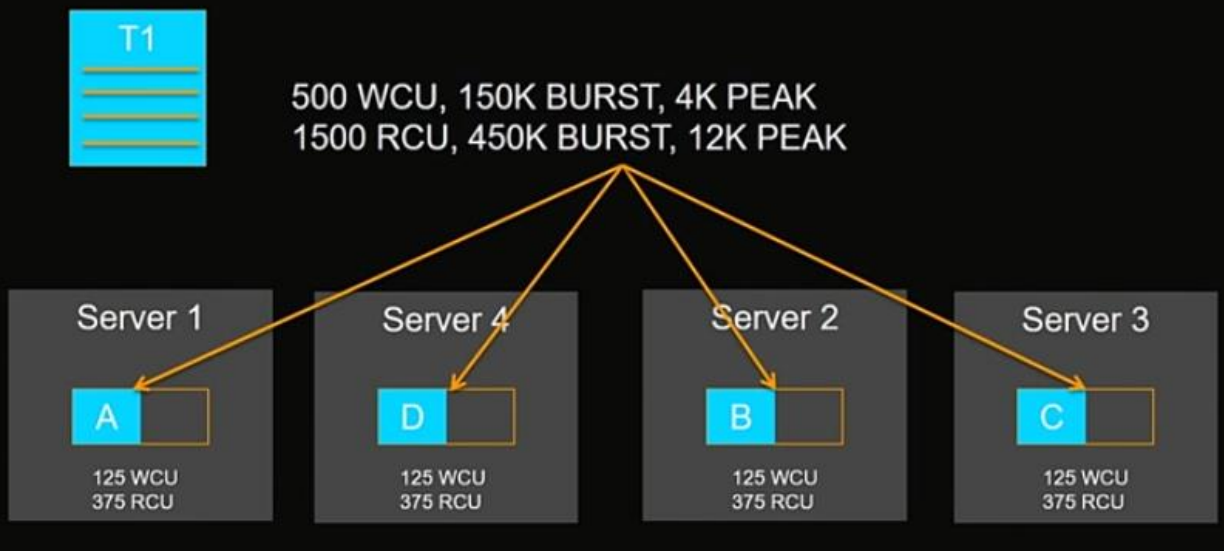
Up to 100 seconds of sustained 2K WCU or 6K RCU burst

Dynamic Scaling—Throughput and Storage



Up to 1 minute of sustained 3K WCU or 9K RCU burst

Dynamic Scaling—Throughput and Storage



Up to 43 seconds of sustained 4K WCU or 12K RCU burst

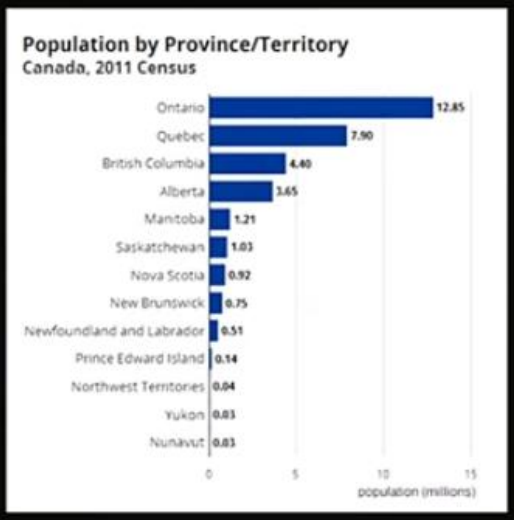
Scenario: Census Application

Statistics Canada (Canada's national statistical agency) hires you to build an online census application

You choose DynamoDB with the following key schema:

- Partition Key: **province**
- Sort Key: **id**

What you didn't realize...



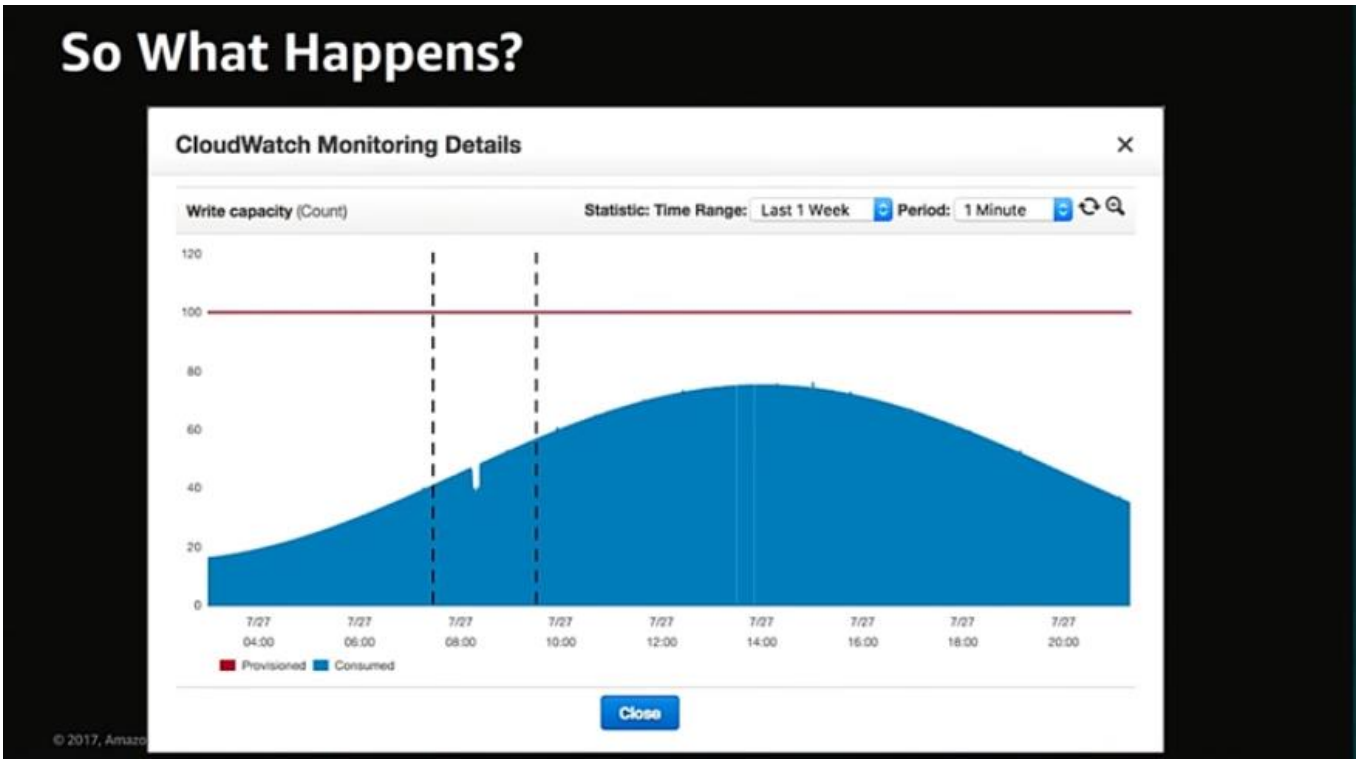
DynamoDB Throughput Allocation

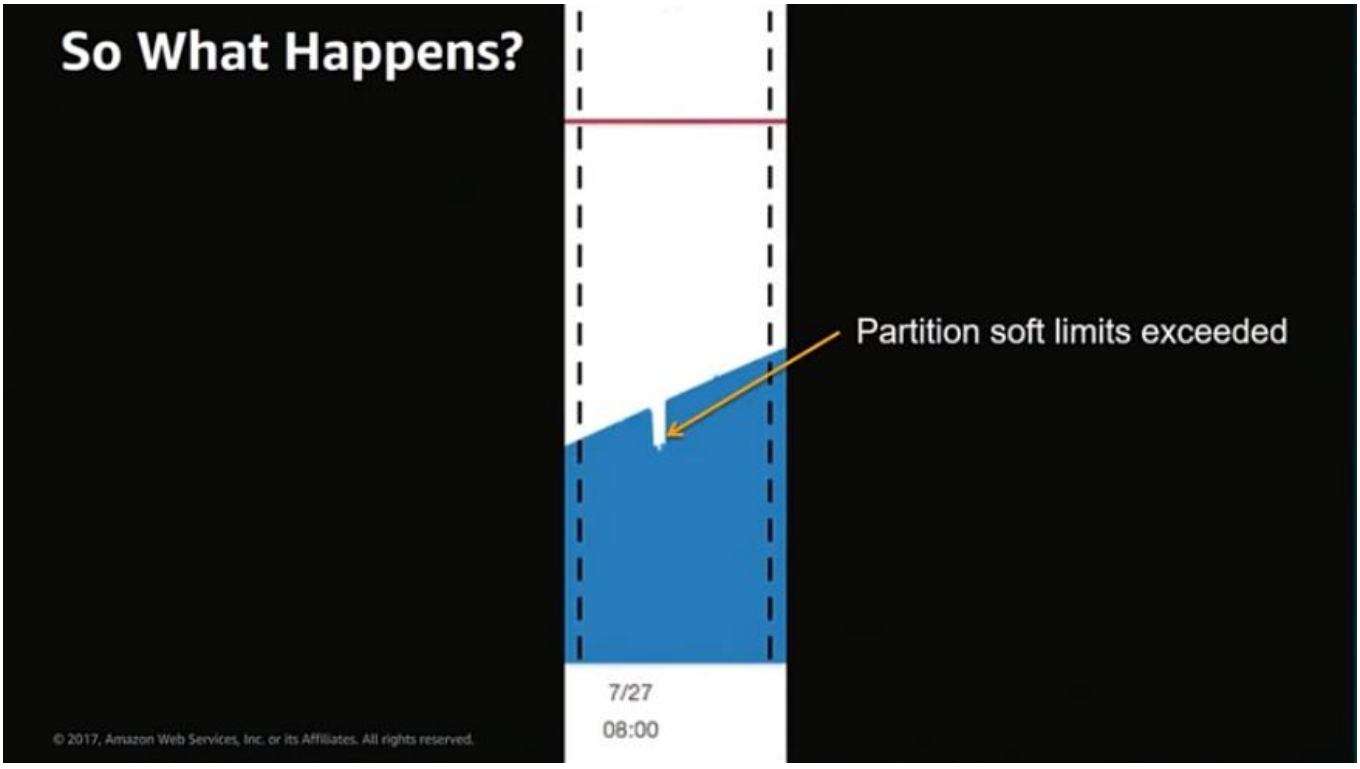
Ontario, Quebec, Newfoundland, Prince Edward Island

British Columbia, Alberta, Sakatchewan

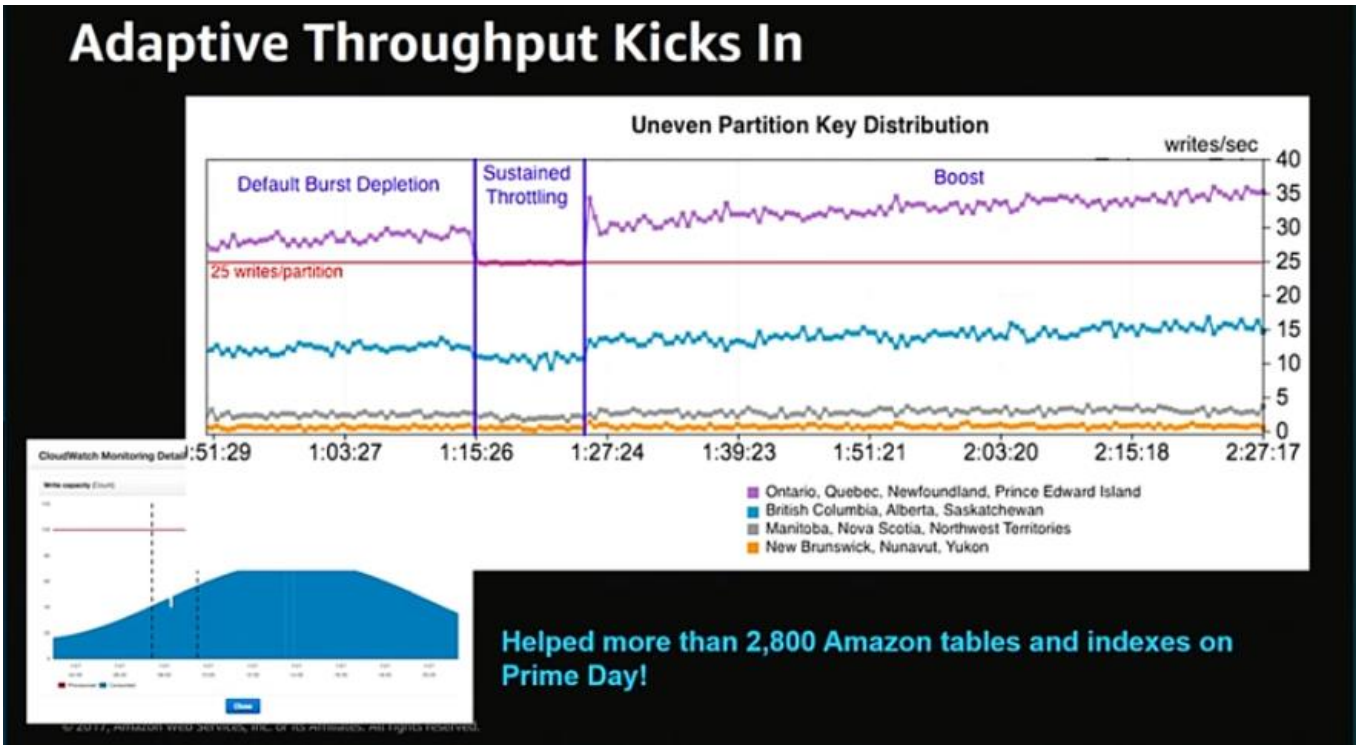
Manitoba, Nova Scotia, Northwest Territories

New Brunswick, Nunavut, Yukon



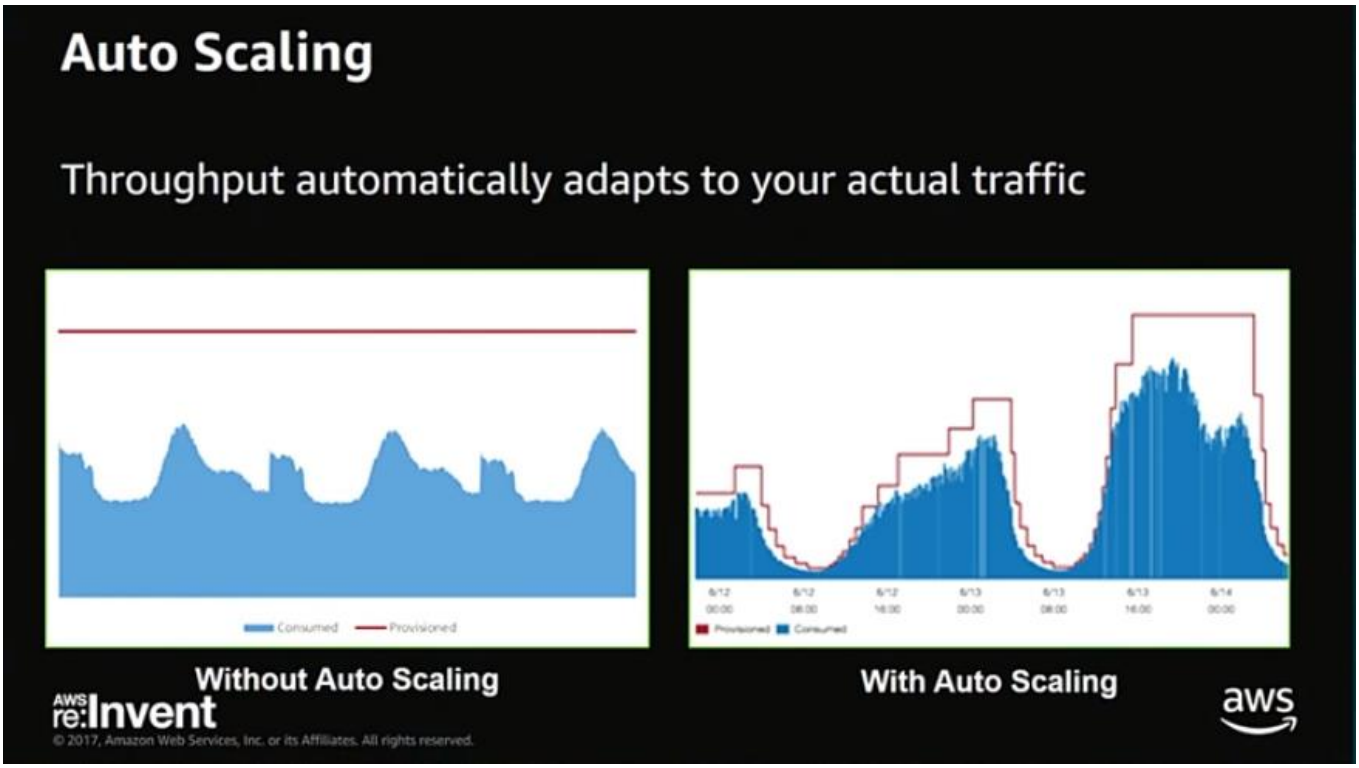


At this point there might be a little bit of throttling on the table as we exceed the soft limit and also exceeded the burst bucket and we figure this table need sustained bursting, we then enable adaptive throughput mode and capacity and scaling and start borrowing throughput from the other table partitions WCUs



You will only get adaptive throughput if you have capacity left on your table's other partitions WCU and have adaptive throughput enabled

Everybody wants a fully hands-off experience



For auto-scaling, you just need to set a high-water mark and a low-water mark or thresholds for the system to increase or decrease capacity depending on the load

Time-to-Live (TTL)

Features

- **Automatic:** deletes items from a table based on expiration timestamp
- **Customizable:** user-defined TTL attribute in epoch time format
- **Audit log:** TTL activity recorded in Amazon DynamoDB Streams

Benefits

- **Reduce costs:** delete items when no longer needed
- **Performance:** optimize app performance by controlling table size growth
- **Extensible:** trigger custom workflows (e.g. auto-archive to Glacier) with DynamoDB Streams, AWS Lambda, Amazon Kinesis, etc.

You can use the TTL attribute on your processed data that can get stale and you want them removed from the table. When the time stamp expires, there is a sweeper that comes along to clear out all the stale event data items for you. This allows you to reduce your cost, TTL items are going to show up on the DynamoDB stream and can be processed by a lambda function.

TTL Results

TUNE

Purged 85 terabytes of stale data and reduced their costs by over $200K per year, while also simplifying their application logic

Amazon CloudWatch

- "Reduced our overall provisioned throughput by 75 percent,"
- "And with that reduction, data-retrieval latencies were also reduced by up to 10 percent."
- "Expect to save millions of dollars annually."

Managing Databases

Servers

- Capacity planning
- Provisioning
- Monitoring
- OS patching
- Hardware upgrades
- Provision new regions

Databases

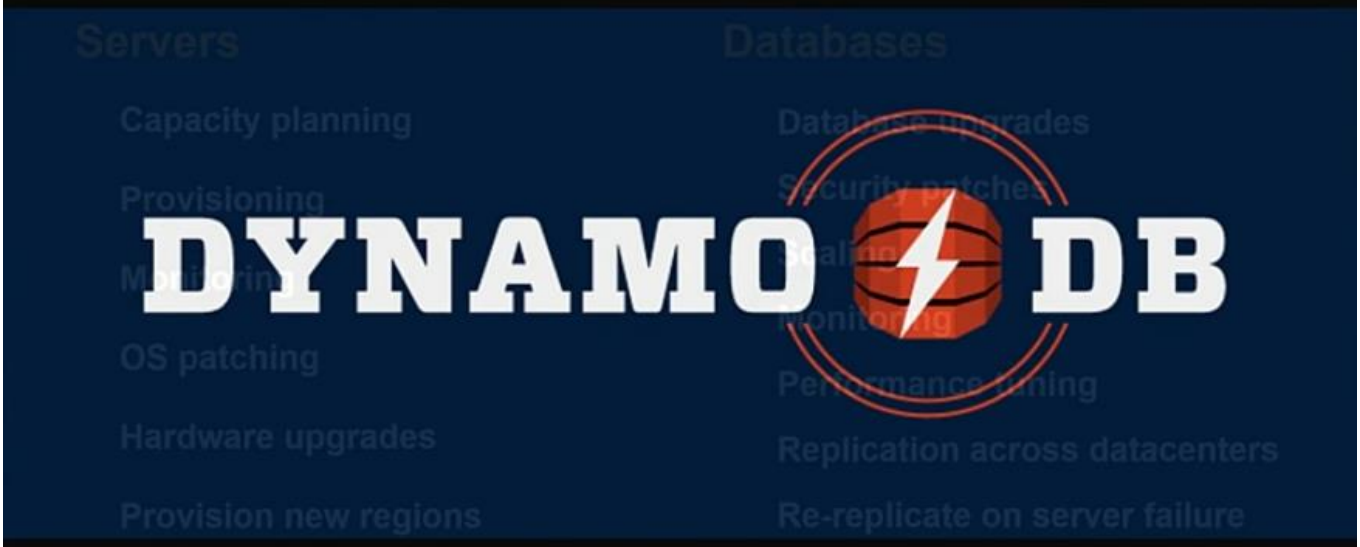
- Database upgrades
- Security patches
- Scaling
- Monitoring
- Performance tuning
- Replication across datacenters
- Re-replicate on server failure

AWS re:Invent

With zero downtime???

aws

Fully Managed, Adaptive, Database Service



AWS re:Invent

All with zero downtime!!!

aws


AWS re:Invent
Thank you!
AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws