DynamoDB (Part 1)

Database Modernization Week

Jason Hunter
Principal Solution Architect

Agenda

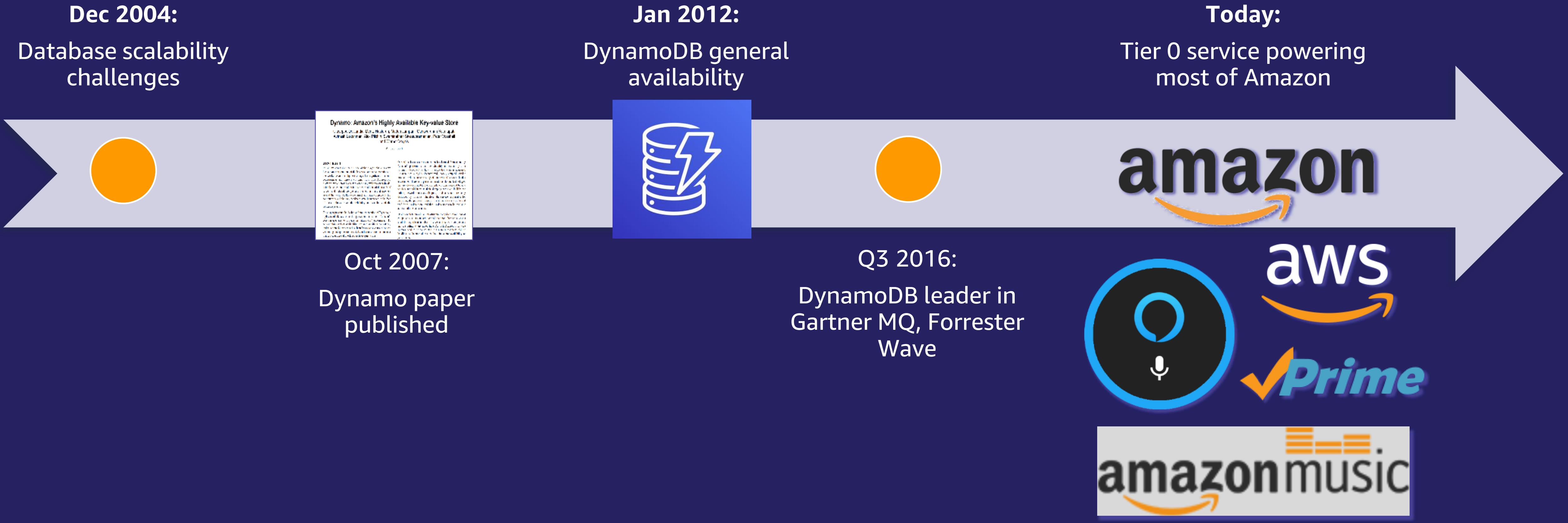
Part 1

- **What's the purpose of DynamoDB?**
- **What are its main features?**
- **Understanding its key concepts**

Part 2

- Looking under the hood
- Managing throughput
- Advanced usage patterns
- Introducing Standard-Infrequent Access table class

The Amazon NoSQL journey



Dec 2004:
Database scalability challenges

Oct 2007:
Dynamo paper published

Jan 2012:
DynamoDB general availability

Q3 2016:
DynamoDB leader in Gartner MQ, Forrester Wave

Today:
Tier 0 service powering most of Amazon

Characteristics of internet-scale apps

E-commerce

Media streaming

Social media

Online gaming

Shared economy

Users	1 million+
Data volume	TB, PB, EB
Locality	Global
Performance	Microsecond latency
Request rate	Millions per second
Access	Mobile, IoT, devices
Scale	Up and down
Economics	Pay as you go
Developer access	Instant API access

Hundreds of thousands of customers have chosen DynamoDB

expedia group
lyft
amazon
airbnb
Adobe
COMCAST
UNDER ARMOUR
mlbam
Arcesium
Avalara
camerite
twilio
TOYOTA RACING DEVELOPMENT TRD
tinder
zynga
FANDUEL
SUPERCELL
Storybulbs
armut
BLACK & VEATCH
CEQUENS
duolingo
dataxu
REDFIN
VidRoll
AdRoll
mapbox
PLANGRAM
Amplitude
Autotask
boomtrain
CLEARCODE
clutch
Clever
CYLANCE
ČESKÁ SPOŘITELNA
Rue La La
druva
embraco
EasyStore
eMoney
etobee
EVRYTHNG
ITT
JUMIA
JUMPRAMP GAMES
KEMPPI
KOCH
Maas GLOBAL
LA NACION
miovision rethink traffic
MARKSYSTEMS
NEWFORMA
Nishitetsu
TTI
playkot
proov by scalabill.it
Q
REALITY REFLECTION
RUMBLE
Reltio
rentcars.com
DOORDASH
SCANIA
triumph learning
TRIOMOBIL
ticketea
ZOOMDATA
VIVID GAMES
tuya
UNIwise
ViaSat
whitepages
workiva
FASHION VALET
Australian Government Geoscience Australia
ZOONA LET'S MAKE IT REAL
Financial Engines
Global Intellect Service
ibotta
frantishex
HIJUP
IGG Software
shop pr shoppers 24/7
GAMETIME
hydrane
Iot-Inc.
SPOKEO
Optimizely
Paragora
PEOPLEADMIN
STARTDATE LABS
SendGrid
Serenova shine bright
ServiceChannel
TalkingData
天润融通 证券代码：835653
THINSLICES
BYJU'S The Learning App
SGK A Matthews International company

aws

DynamoDB use cases by industry

Customers rely on DynamoDB to support their mission-critical workloads

Banking and finance

Fraud detection
User transactions
Mainframe offloading
(Capital One, Vanguard, Fannie Mae)

Gaming

Game states
Leaderboards
Player data stores
(Riot Games, Electronic Arts, PennyPop)

Software and internet

Metadata caches
Ride-tracking data stores
Relationship graph data stores
(Uber, Lyft, Swiggy, Snap, Duolingo)

Ad tech

User profile stores
Metadata stores for assets
Popular-item cache
(AdRoll, GumGum, Branch, DataXu)

Retail

Shopping carts
Workflow engines
Customer profiles
(Nordstrom, Nike, Zalando, Mercado Libre)

Media & Entertainment

User data stores
Media metadata stores
Digital rights management stores
(Airtel Wynk, Amazon Prime, Netflix)

DynamoDB

Fast and flexible NoSQL database service for any scale

Performance at scale

- Handles millions of requests per second
- Delivers single-digit-millisecond latency
- Automated global replication
- New advanced streaming with Amazon Kinesis Data Streams for DynamoDB

No servers to manage

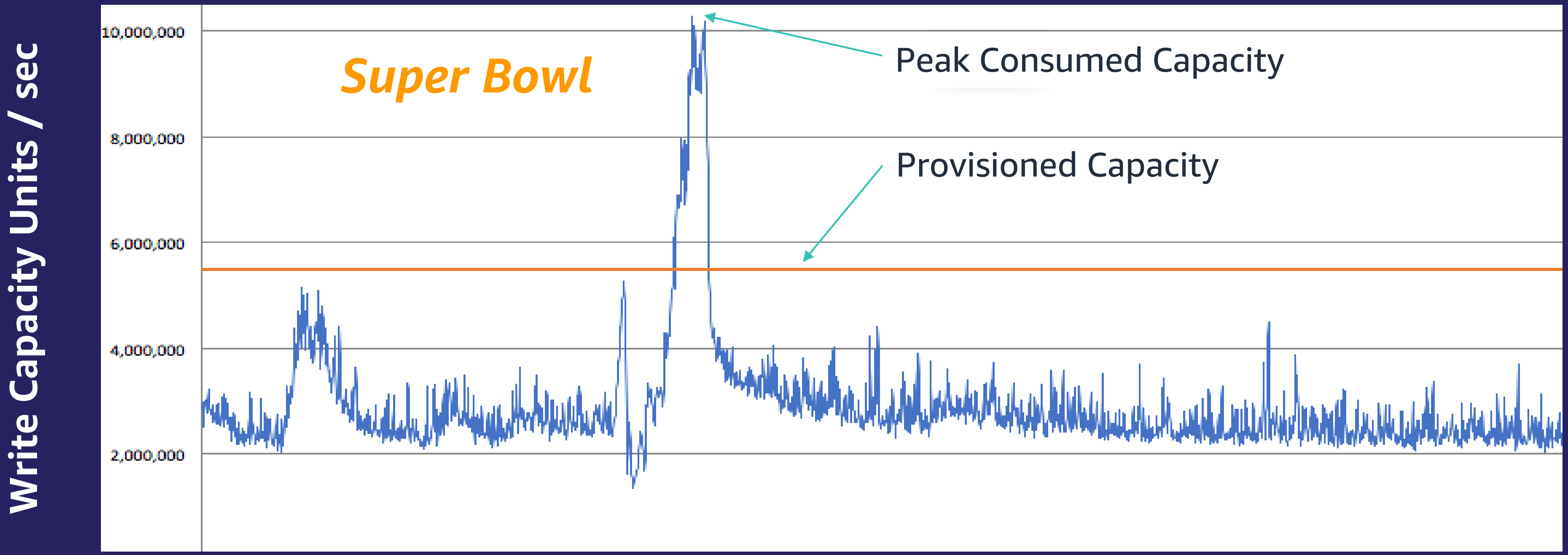
- Maintenance free
- Auto scaling
- On-demand capacity mode
- Change data capture for integration with AWS Lambda, Amazon Redshift, Amazon Elasticsearch Service

Enterprise ready

- ACID transactions
- Encryption at rest
- Continuous backups (PITR), and on-demand backup and restore
- NoSQL Workbench
- Export table data to S3
- PartiQL (a SQL-compatible query language) support

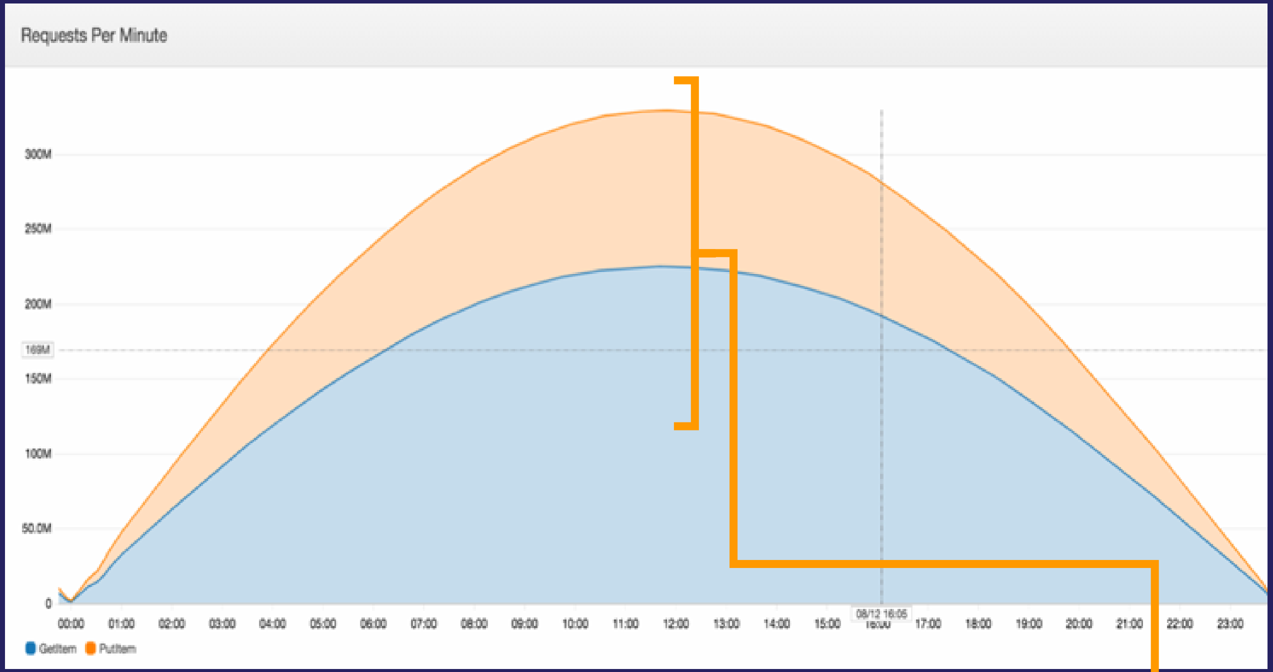
Performance at Scale

Global-Scale Events: Elastic is the New Normal



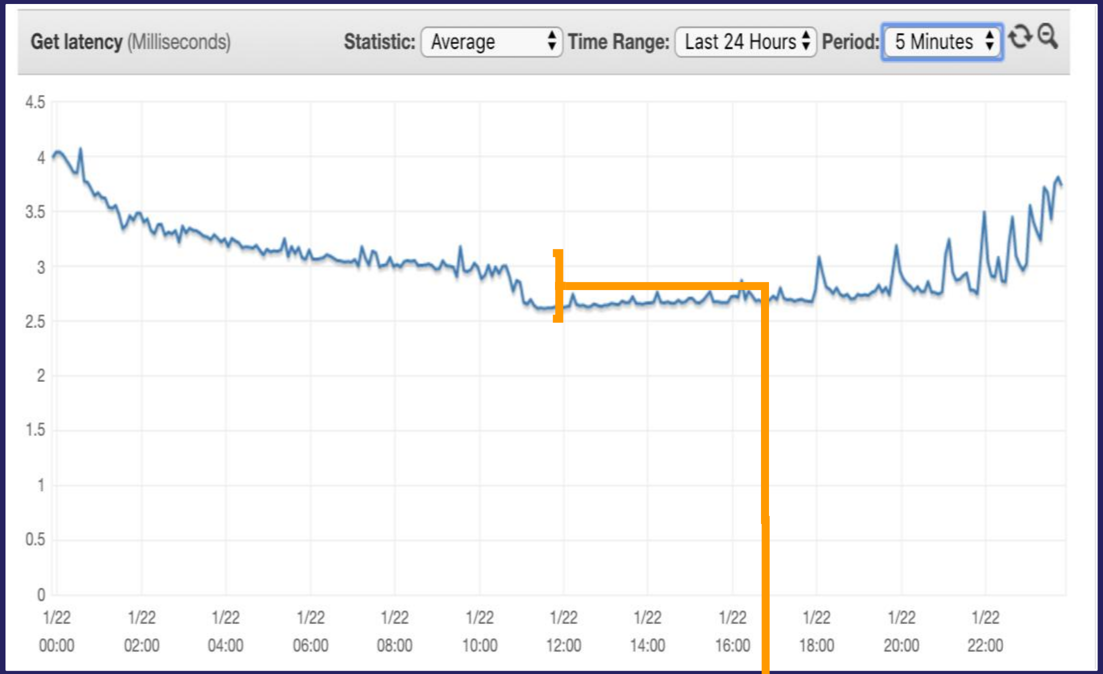
Performance at any scale

High request volume



Over **5 million requests per second** per table

Consistent low latency

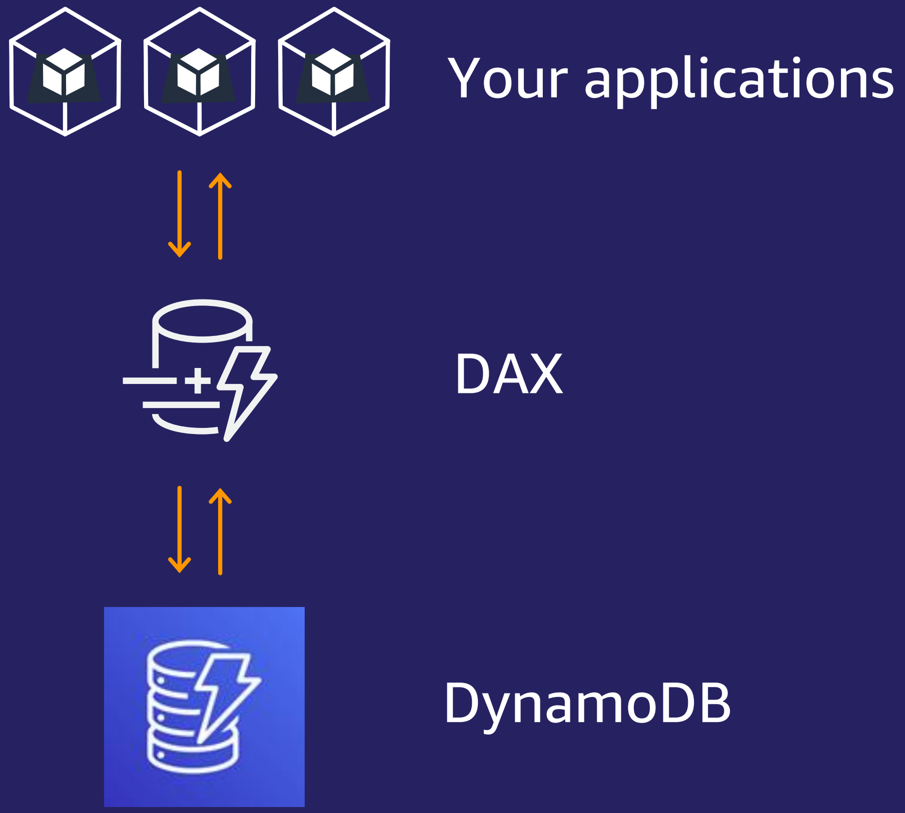


Get latency (Milliseconds) **Statistic:** Average **Time Range:** Last 24 Hours **Period:** 5 Minutes

Millisecond variance

DynamoDB Accelerator (DAX) adds read cache

Performance at scale



Fully managed,
highly available cache
for DynamoDB

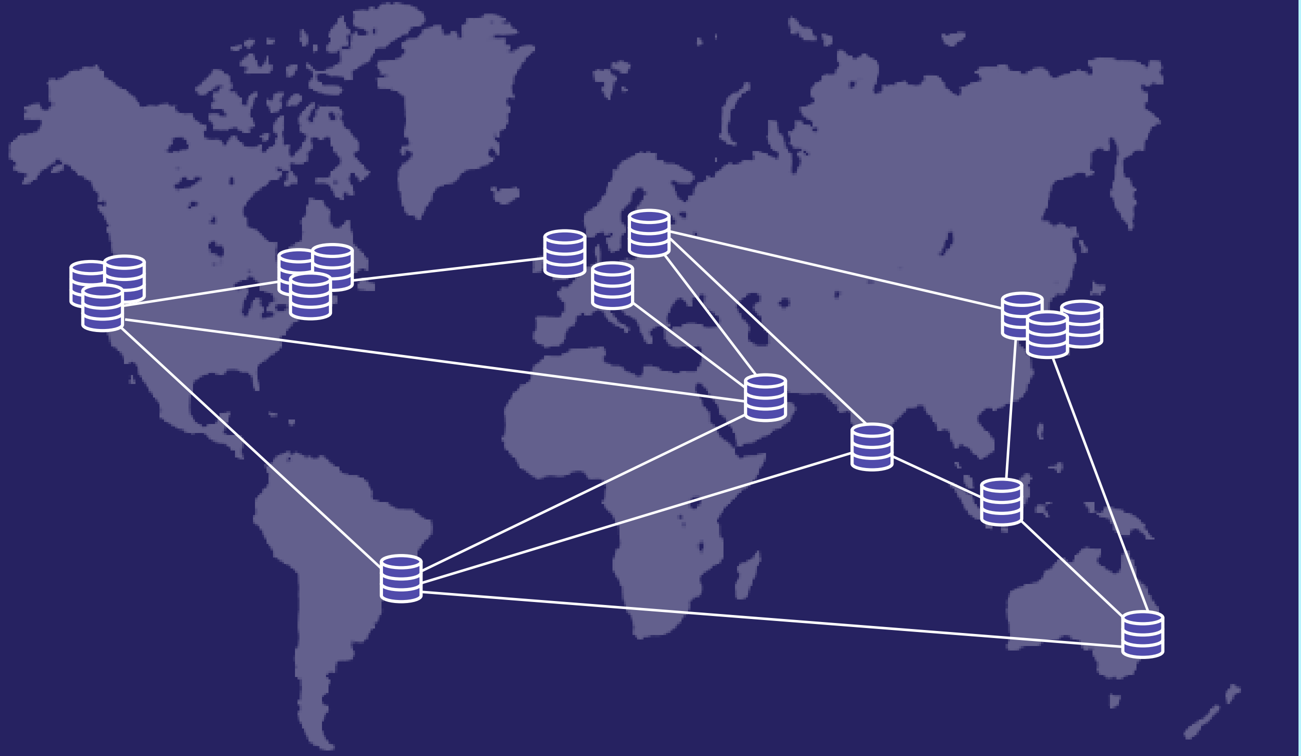
Even faster—microsecond latency

Scales to millions of read requests per second

API compatible

Global tables provide apps with multi-Region replication

Performance at scale



Build high-performance, globally distributed applications

Low-latency reads and writes to locally available tables

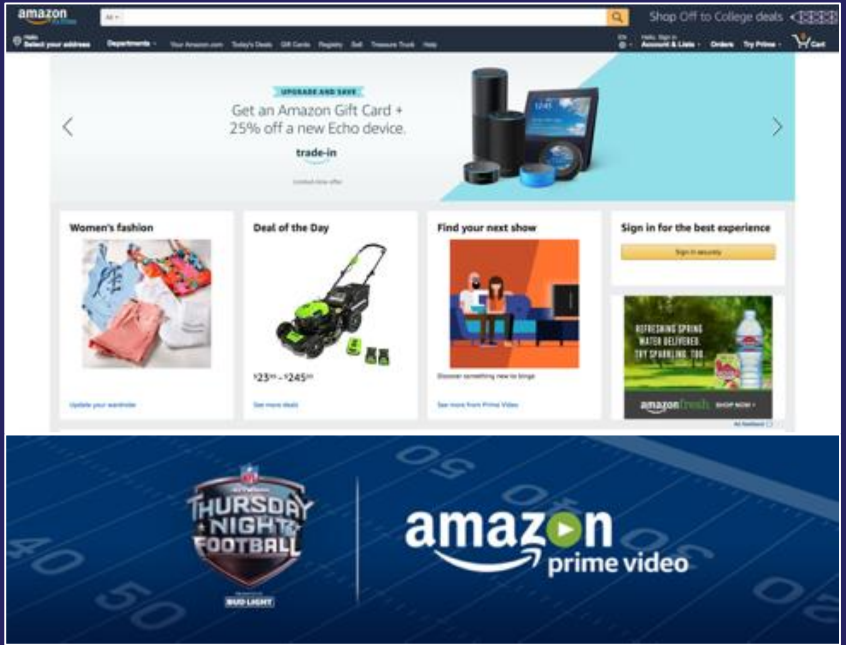
Multi-Region redundancy and resiliency and 99.999% availability

Multi-active writes from any Region

Easy to set up and no application rewrites required

Why Amazon.com depends on DynamoDB for performance at scale

Retail



Amazon DynamoDB supports multiple high-traffic sites and systems including the Amazon.com sites, Alexa, and 442 Amazon fulfillment centers. Across the 66-hour 2020 Prime Day, these sources made 16.4 trillion calls to the DynamoDB API, peaking at 80.1 million requests per second.

The internal Amazon.com Herd system supports 100s of millions of active workflows.

Migrated from Oracle to DynamoDB

- **Improved customer experience:** Workflow processing delays dropped from 1 second to 100 milliseconds.
- **Reduced cost:** Scaling and maintenance effort dropped 10 times.
- **Reduced complexity and risk:** Retired more than 300 Oracle hosts.

No Servers to Manage

Getting back valuable time for your business

No servers to manage

As a fully managed database service, DynamoDB does the heavy lifting for you across:

Security

- Operating-system patching
- Database patching
- Access control enforcement
- Audit activities
- Encryption management
- Compliance

Durability

- Sustain server, rack, and datacenter outages
- Re-replicate data quickly upon hardware failure
- Manage backup and restore

Availability

- High availability configuration
- Monitoring reporting
- Cross-Region replication management

Performance

- Performance tuning
- Index management
- Cache management

Scalability

- Host provisioning
- Host repair and retirement

On-demand capacity mode: rapid, flexible scaling

Pay per request pricing

No capacity management

No need to specify how much read/write throughput you expect to use

Ideal for unpredictable workloads

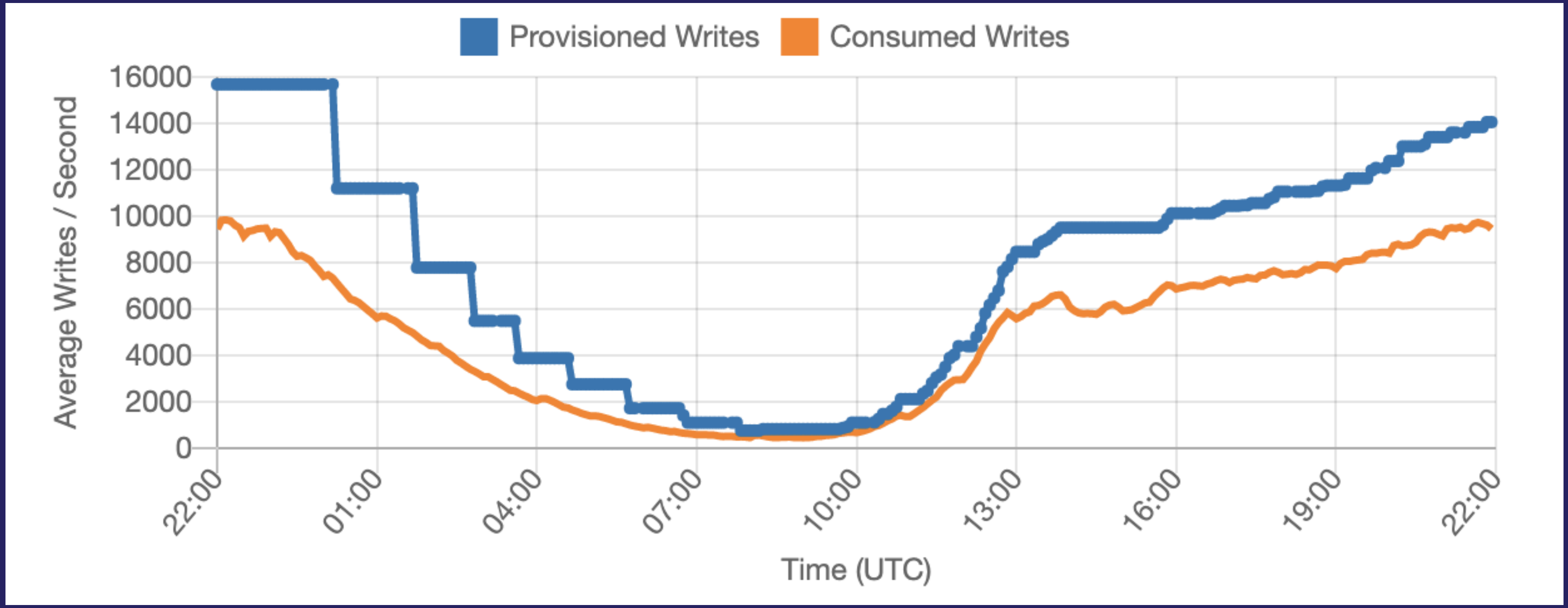
Ramp from zero to tens of thousands of requests per second on demand

Pay only for what you use

Pay-per-request pricing

Provisioned capacity mode: auto scaling, maintains performance

Provision capacity as needed



Provision at a given amount of capacity

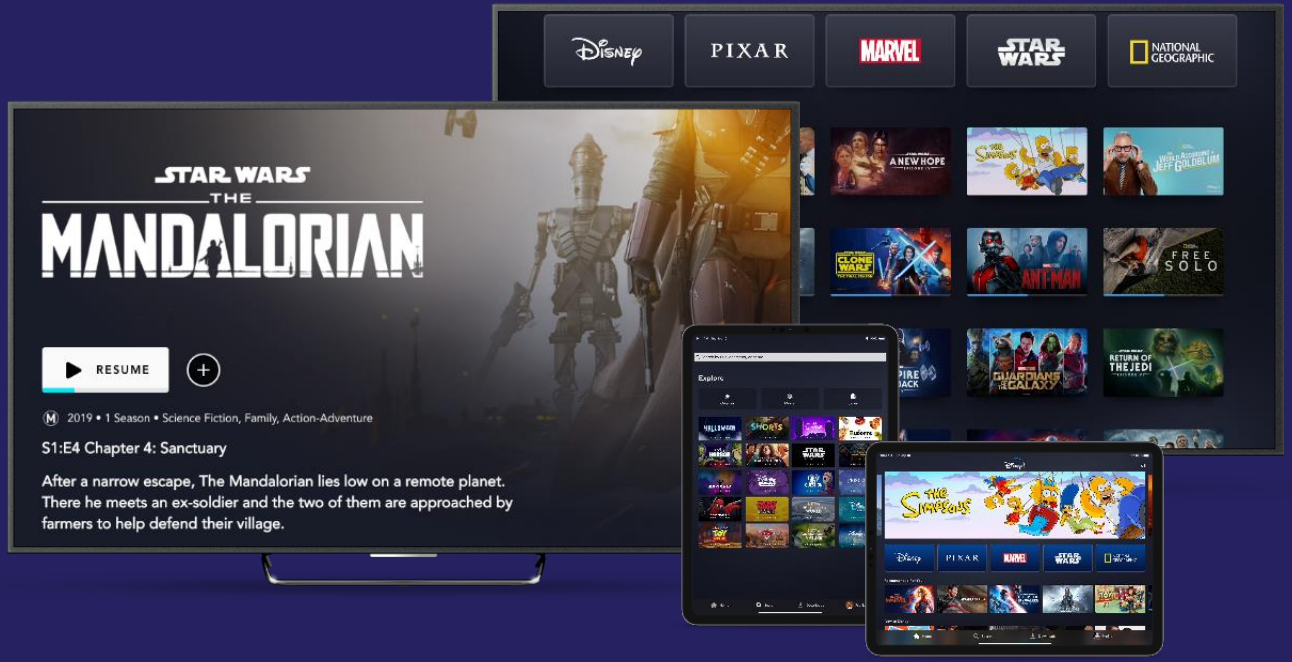
Lower cost per request than On Demand

"Auto-scale" scales up when you need it, down when you don't

Schedule any scaling events (bulk load, launch day)

PROVISIONED CAPACITY MODE

How Disney has simplified operations and reduced risk through serverless DynamoDB



2019 • 1 Season • Science Fiction, Family, Action-Adventure

S1:E4 Chapter 4: Sanctuary

After a narrow escape, The Mandalorian lies low on a remote planet. There he meets an ex-soldier and the two of them are approached by farmers to help defend their village.

> “Billions of bookmarks ingested a day over Amazon Kinesis and into Amazon DynamoDB.”

—**Attilio Giue**
Director of Content Discovery, Disney+

Disney+ launched in November 2019 and delivers its extensive library of digital content directly to the homes of over 60.5 million subscribers, and DynamoDB is one of the technologies that supports this global footprint.

Disney+ chose DynamoDB to help with:

- Utilizing multi-Region replication with single-digit latency to shift traffic without experiencing data issues
- Adding another AWS Region in global tables to launch into new countries, providing low latency
- Scaling Recommendations and Bookmarks with little to no operational overhead
- Having the ability to switch back and forth between on-demand and provisioned capacity modes when entering new Regions

Enterprise Ready

Native, server-side support for ACID transactions

Enterprise ready

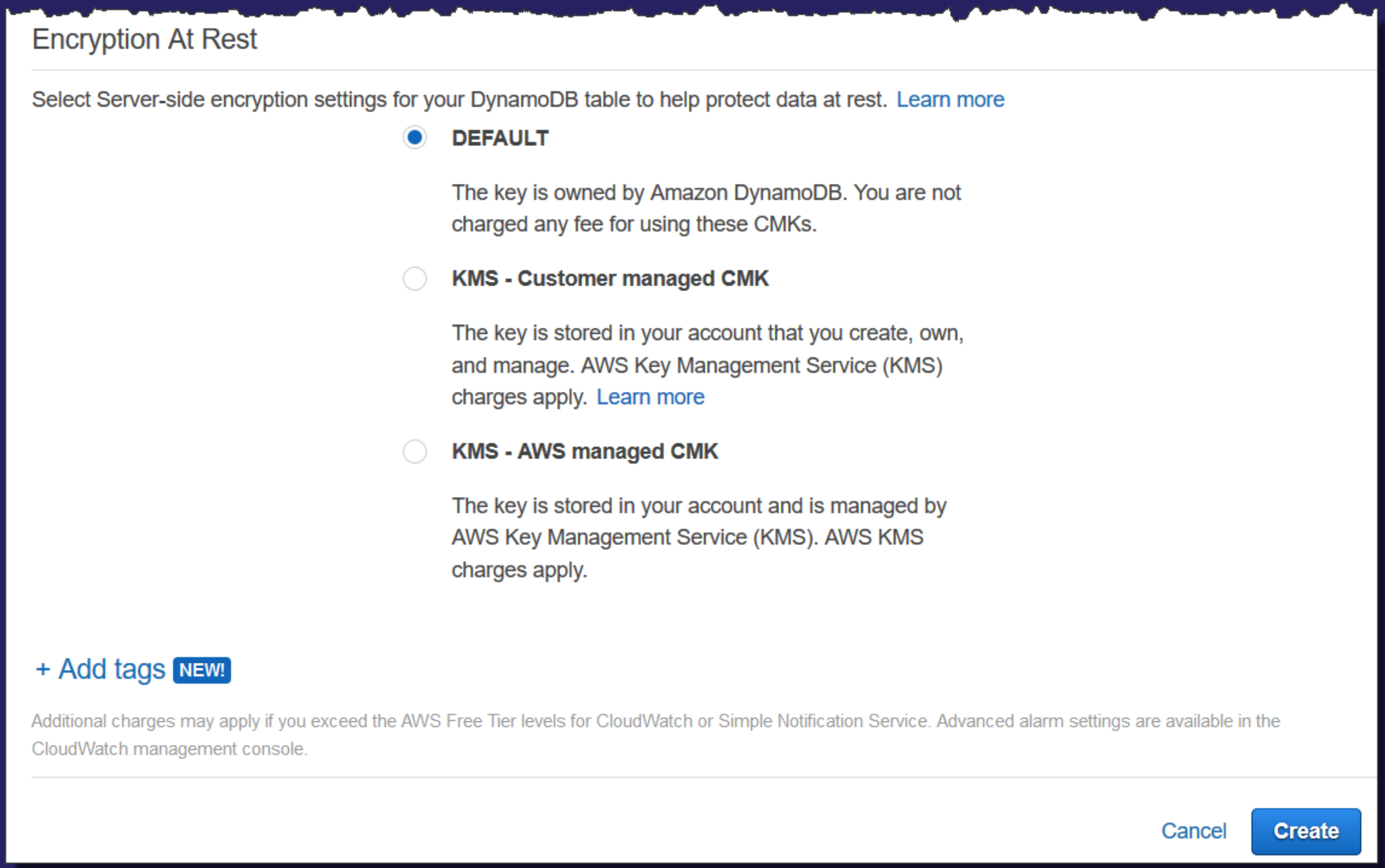
Simplify application code with ACID guarantees

Run transactions for large-scale workloads

Accelerate legacy migrations

Security – Access Controls and Encryption at Rest

Enterprise ready



Encryption At Rest

Select Server-side encryption settings for your DynamoDB table to help protect data at rest. Learn more

● **DEFAULT**

The key is owned by Amazon DynamoDB. You are not charged any fee for using these CMKs.

○ **KMS - Customer managed CMK**

The key is stored in your account that you create, own, and manage. AWS Key Management Service (KMS) charges apply. Learn more

○ **KMS - AWS managed CMK**

The key is stored in your account and is managed by AWS Key Management Service (KMS). AWS KMS charges apply.

Additional charges may apply if you exceed the AWS Free Tier levels for CloudWatch or Simple Notification Service. Advanced alarm settings are available in the CloudWatch management console.

All tables encrypted in transit, at rest by default

Fully integrated with AWS Identity and Access Management (IAM)

Access DynamoDB from a VPC via secure VPC endpoints

Security – Audit Logging with AWS CloudTrail

Enterprise ready

- Capture and log all **control-plane operations** and **data-plane operations** for compliance, operational, and risk auditing
- Record table-level and item-level activity, trigger actions when important events are detected, and analyze events and logs with Amazon Athena or CloudWatch Logs Insights

Compliance aid

Visibility into activity

Detect data exfiltration

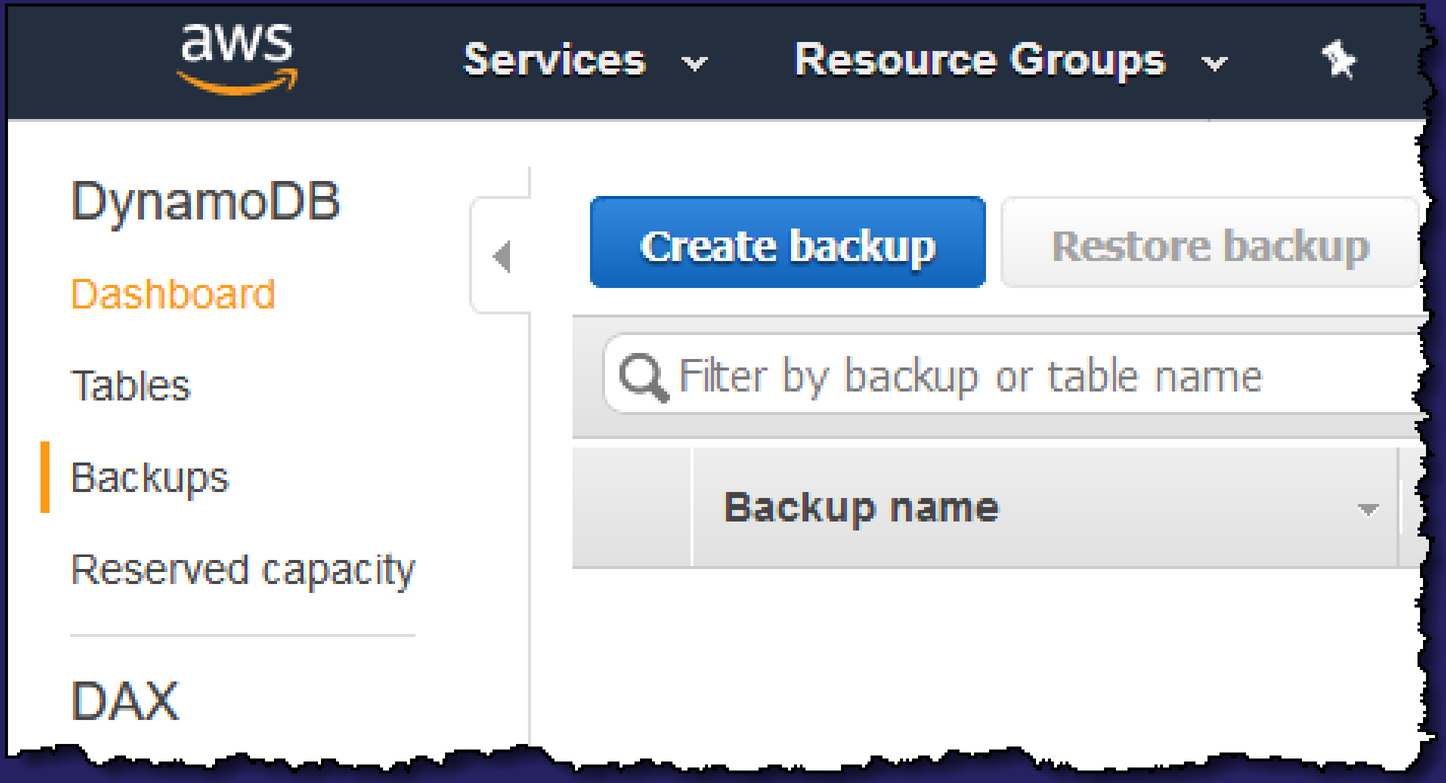
Automate security analysis

Troubleshoot anomalies

Analyze permissions

Backup and restore

Enterprise ready



On-demand backups for long-term data archiving and compliance

Continuous backups for point-in-time recovery (PITR)

Zero performance impact

NoSQL Workbench

Enterprise ready

Data modeler

Visualizer

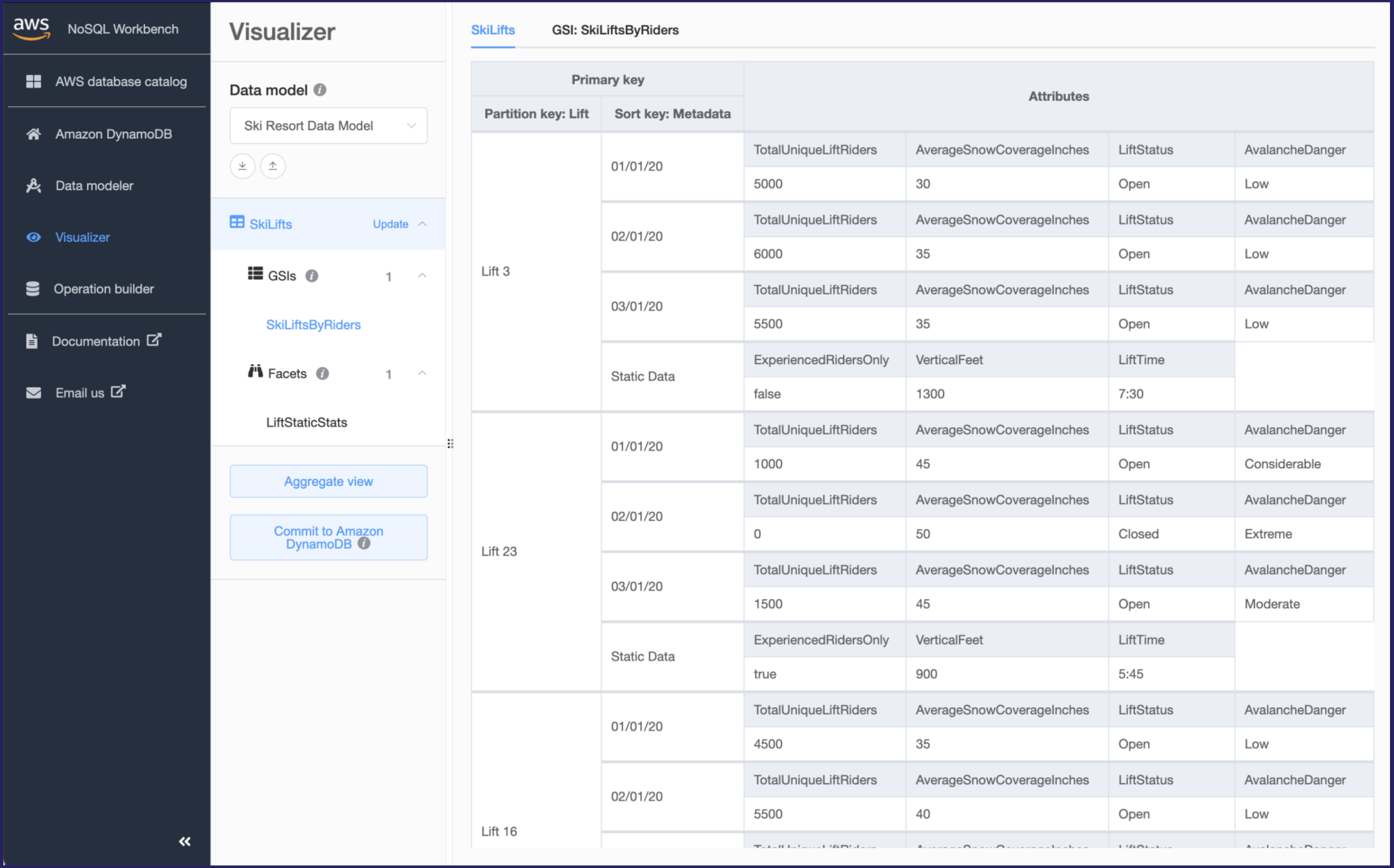
Operation builder

A **client-side application** that helps you build scalable, high-performance data models

Simplifies query development and testing

A rich GUI-based tool that **helps you visualize data models** and **perform DynamoDB operations**

Available for **Windows, macOS, and Linux**



Primary key		Attributes			
Partition key: Lift	Sort key: Metadata				
Lift 3	01/01/20	TotalUniqueLiftRiders	AverageSnowCoverageInches	LiftStatus	AvalancheDanger
		5000	30	Open	Low
	02/01/20	TotalUniqueLiftRiders	AverageSnowCoverageInches	LiftStatus	AvalancheDanger
		6000	35	Open	Low
	03/01/20	TotalUniqueLiftRiders	AverageSnowCoverageInches	LiftStatus	AvalancheDanger
		5500	35	Open	Low
	Static Data	ExperiencedRidersOnly	VerticalFeet	LiftTime	
		false	1300	7:30	
Lift 23	01/01/20	TotalUniqueLiftRiders	AverageSnowCoverageInches	LiftStatus	AvalancheDanger
		1000	45	Open	Considerable
	02/01/20	TotalUniqueLiftRiders	AverageSnowCoverageInches	LiftStatus	AvalancheDanger
		0	50	Closed	Extreme
	03/01/20	TotalUniqueLiftRiders	AverageSnowCoverageInches	LiftStatus	AvalancheDanger
		1500	45	Open	Moderate
	Static Data	ExperiencedRidersOnly	VerticalFeet	LiftTime	
		true	900	5:45	
Lift 16	01/01/20	TotalUniqueLiftRiders	AverageSnowCoverageInches	LiftStatus	AvalancheDanger
		4500	35	Open	Low
	02/01/20	TotalUniqueLiftRiders	AverageSnowCoverageInches	LiftStatus	AvalancheDanger
		5500	40	Open	Low
		TotalUniqueLiftRiders	AverageSnowCoverageInches	LiftStatus	AvalancheDanger

Export DynamoDB data to S3 for analysis and insights

Enterprise ready

Extract actionable insights

Export DynamoDB table data to your data lake in Amazon S3, and use other AWS services to analyze data and highlight key takeaways.

Integrate with backups

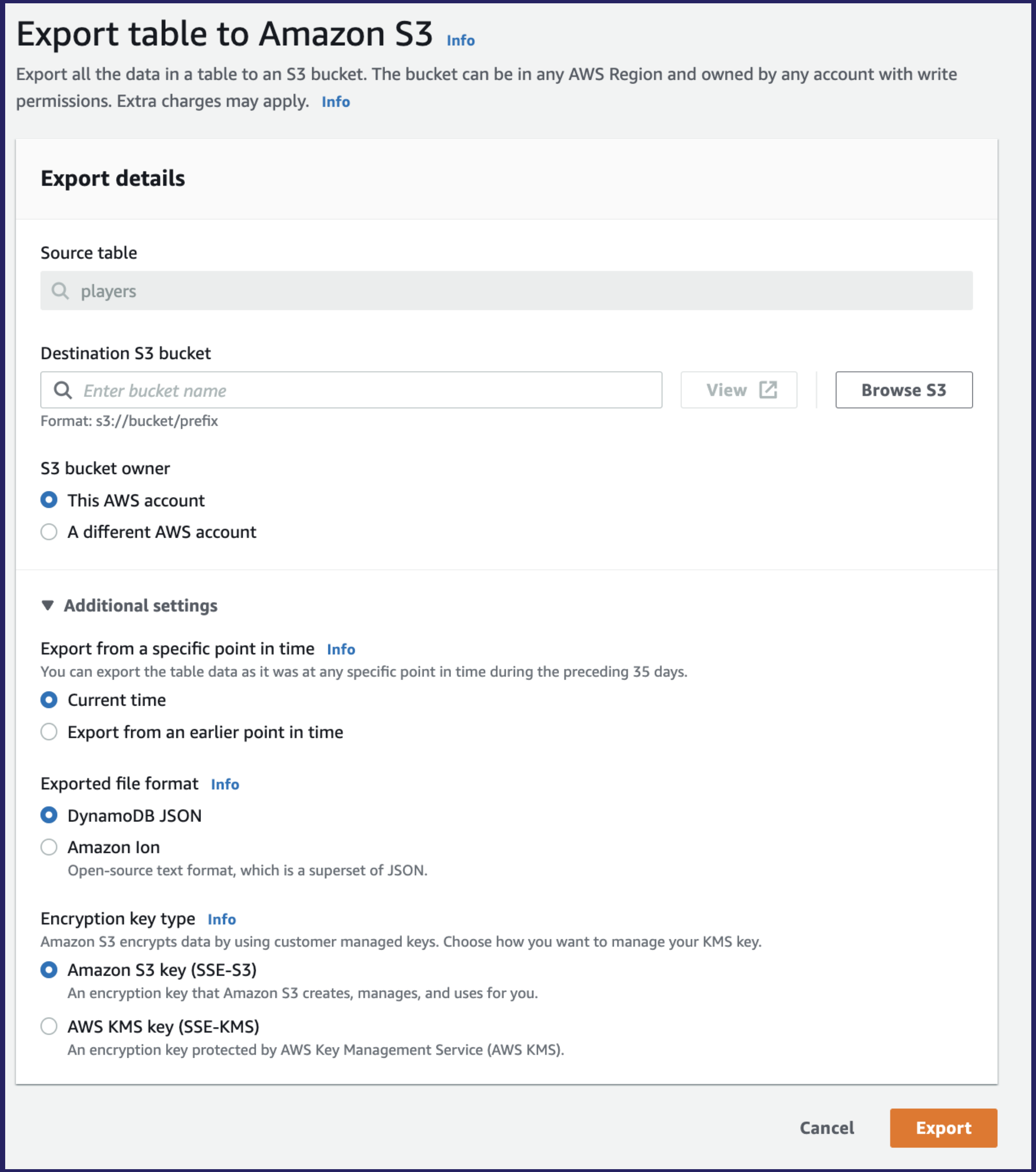
To export, select a DynamoDB table that has point-in-time recovery (PITR) enabled, specify any point in the last 35 days, and choose the target Amazon S3 bucket. The output data formats supported are DynamoDB JSON and Amazon Ion.

Work across Regions

Export data to S3 across AWS Regions and accounts to help comply with regulatory requirements, and to develop a disaster recovery and business continuity plan.

No impact on performance

Does not consume table capacity, and has zero impact on performance and availability. All DynamoDB data added to your Amazon S3 data lake is easily discoverable, encrypted at rest and in transit, and retained in your S3 bucket until you delete it.



Export table to Amazon S3 Info

Export all the data in a table to an S3 bucket. The bucket can be in any AWS Region and owned by any account with write permissions. Extra charges may apply. Info

Export details

Source table

players

Destination S3 bucket

Enter bucket name

Format: s3://bucket/prefix

S3 bucket owner

● This AWS account

○ A different AWS account

▼ Additional settings

Export from a specific point in time Info

You can export the table data as it was at any specific point in time during the preceding 35 days.

● Current time

○ Export from an earlier point in time

Exported file format Info

● DynamoDB JSON

○ Amazon Ion

Open-source text format, which is a superset of JSON.

Encryption key type Info

Amazon S3 encrypts data by using customer managed keys. Choose how you want to manage your KMS key.

● Amazon S3 key (SSE-S3)

An encryption key that Amazon S3 creates, manages, and uses for you.

○ AWS KMS key (SSE-KMS)

An encryption key protected by AWS Key Management Service (AWS KMS).

PartiQL now supported for easier queries

Enterprise ready

Easier queries

You can now use PartiQL (a SQL-compatible query language) to query, insert, update, and delete table data in the DynamoDB console.

Improved productivity

Because PartiQL is supported for all data-plane operations, developers can use a familiar, structured query language to perform these operations.

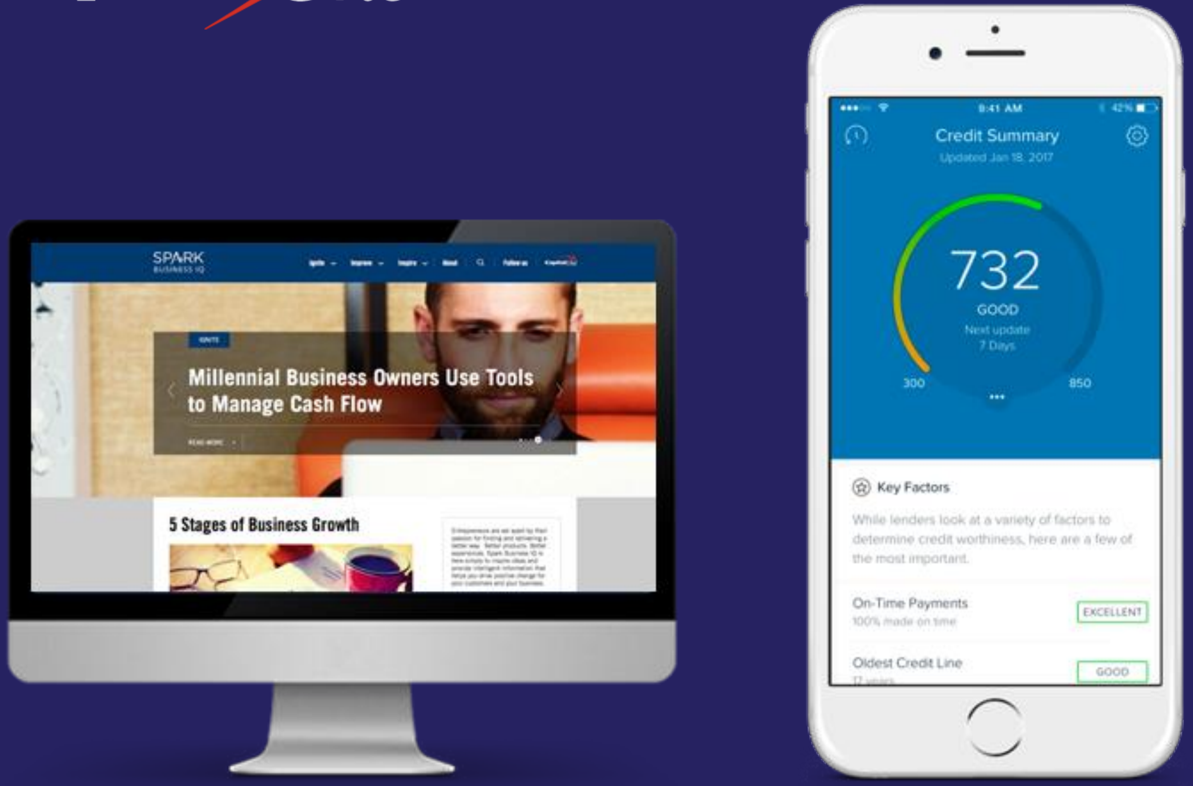
Consistent performance

With PartiQL, DynamoDB continues to provide consistent, single-digit-millisecond latency at any scale. You can expect the same availability, latency, and performance when performing DynamoDB operations.

How Capital One increased their speed of innovation because of enterprise-ready DynamoDB

Capital One

Banking and finance


SPARK BUSINESS IQ
Millennial Business Owners Use Tools to Manage Cash Flow
5 Stages of Business Growth
9:41 AM
Credit Summary
Updated Jan 18, 2017
732
GOOD
Next update 7 Days
300
850
Key Factors
While lenders look at a variety of factors to determine credit worthiness, here are a few of the most important.
On-Time Payments
100% made on time
EXCELLENT
Oldest Credit Line
GOOD

Capital One completes migration in 2020 from data centers to AWS, becomes first US bank to go all in on the cloud

Migrated from mainframe to DynamoDB:

- Previously all apps were served by a single mainframe sitting in the middle of their physical business
- Product teams busy coming up with new mobile products for customers were often blocked by the mainframe
- DynamoDB and microservices give app developers unbounded scale, nimbleness, and the ability to roll out all new services

> "The new solution is so much faster…with an average response time of 55 ms."

—**Srini Uppalapati**
Capital One

aws

Key Concepts

SQL and NoSQL side by side

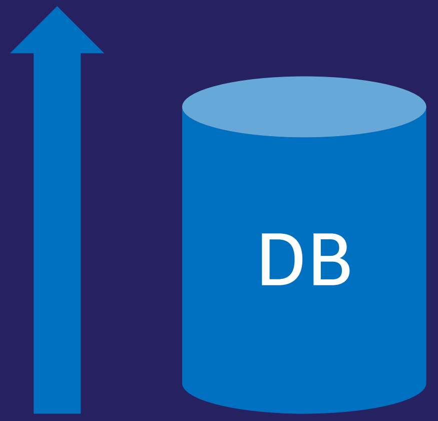
SQL	NoSQL
Optimized for storage	**Optimized for compute**
Normalized/relational	Denormalized/hierarchical
Ad hoc queries	Instantiated views
Scale vertically	Scale horizontally
Good for OLTP / OLAP	Built for OLTP* at scale

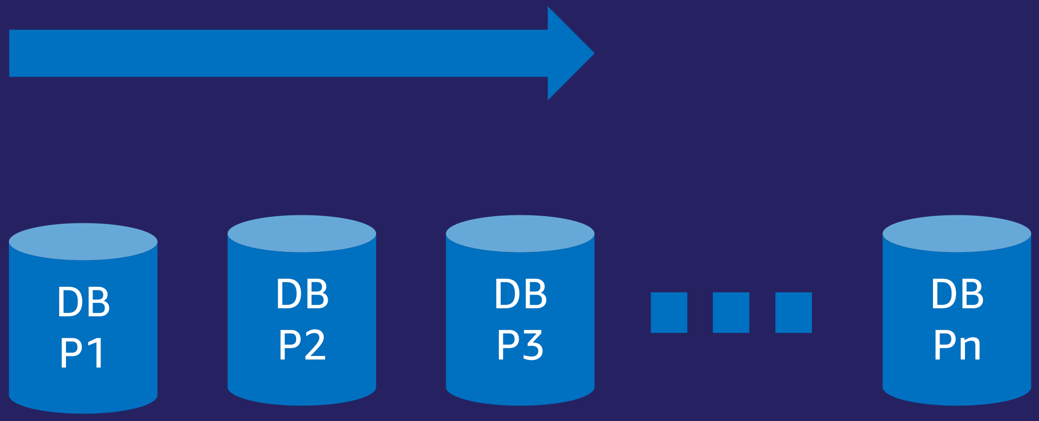
(*) DynamoDB is. Some NoSQL databases are built for analytical workloads.

Scaling databases

Traditional SQL

NoSQL



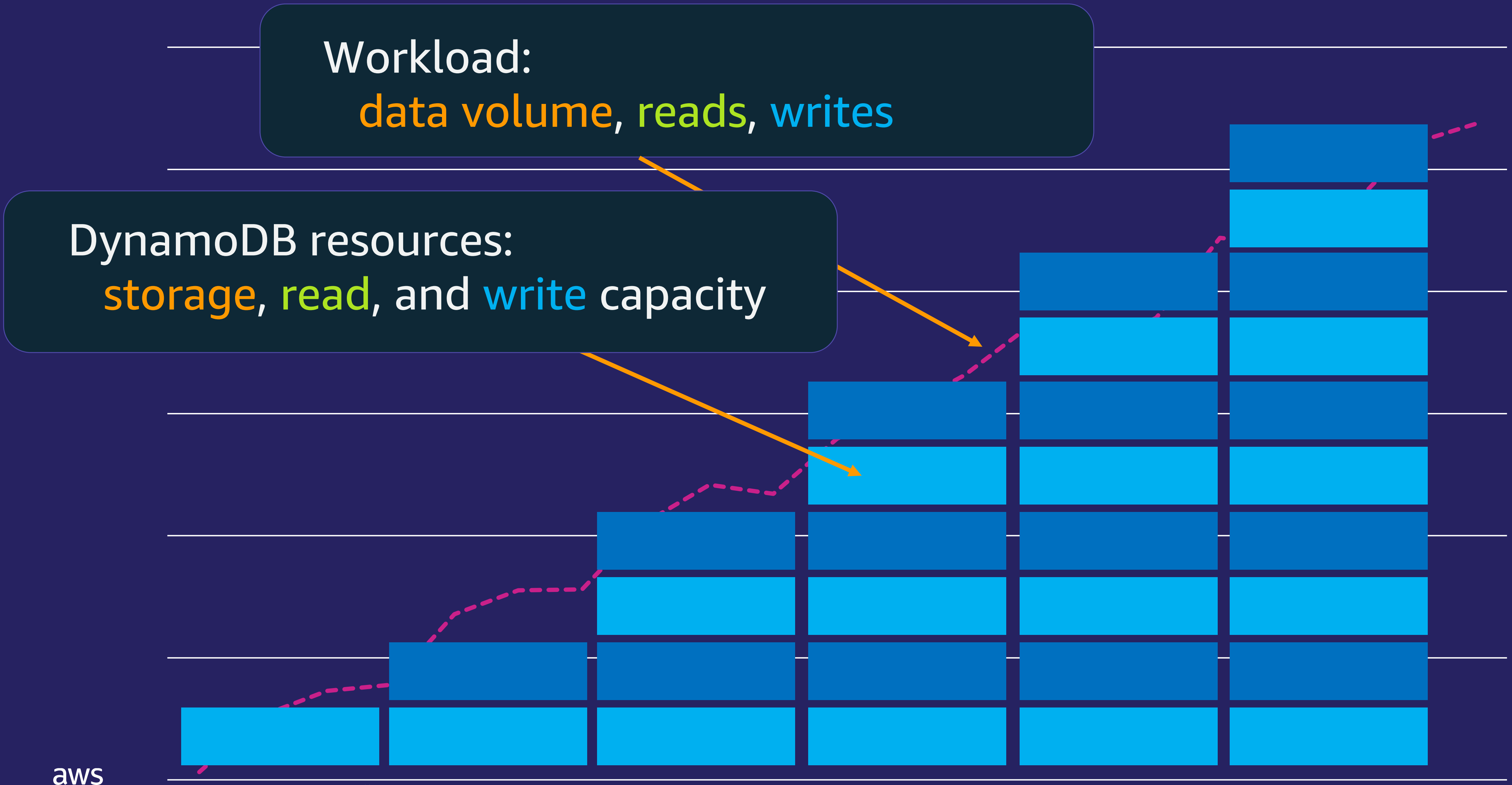


Scale up

Scale out to many shards

Basic premise: There is a way to design data that’s horizontally scalable.

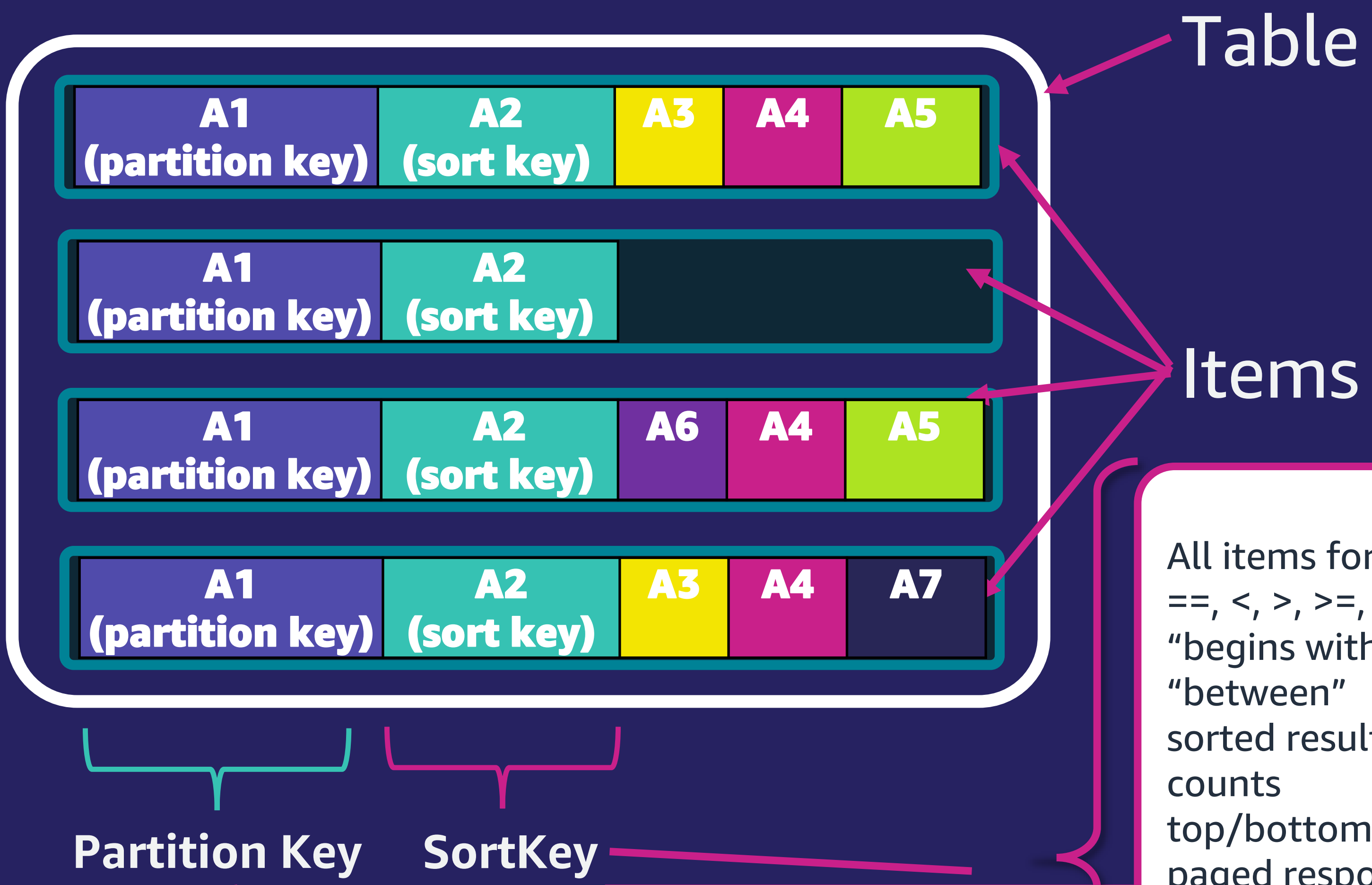
Horizontal scaling with DynamoDB



Workload:
data volume, reads, writes

DynamoDB resources:
storage, read, and write capacity

DynamoDB Table



Table

Items

A1 (partition key)	A2 (sort key)	A3	A4	A5
A1 (partition key)	A2 (sort key)			
A1 (partition key)	A2 (sort key)	A6	A4	A5
A1 (partition key)	A2 (sort key)	A3	A4	A7

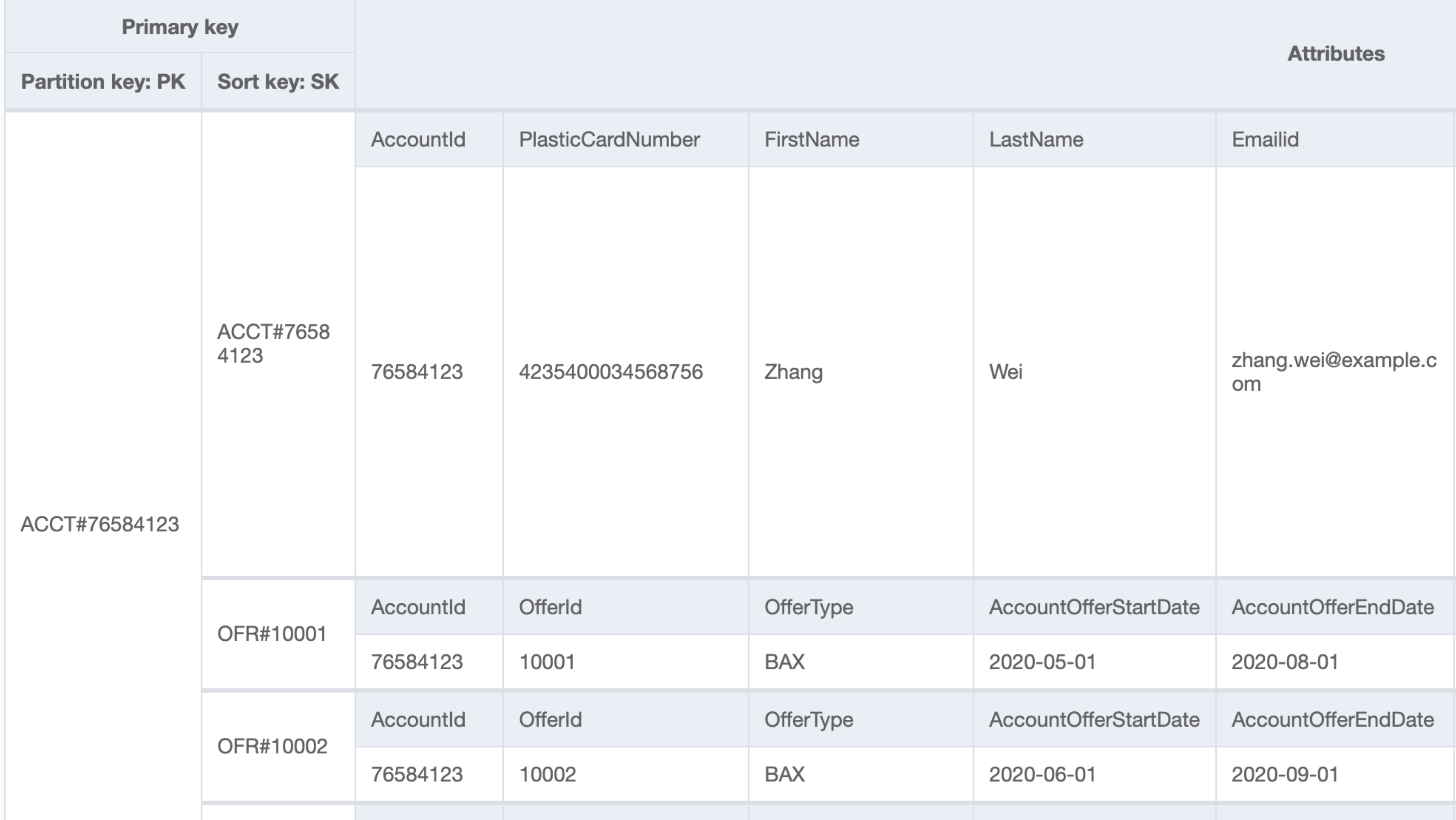
Partition Key

SortKey

All items for a partition key
==, <, >, >=, <=
“begins with”
“between”
sorted results
counts
top/bottom N values
paged responses

Mandatory
Key-value access pattern
Determines data distribution

Optional
Model 1:N relationships
Enables rich query capabilities



Primary key		Attributes				
Partition key: PK	Sort key: SK					
ACCT#76584123	ACCT#76584123	AccountId	PlasticCardNumber	FirstName	LastName	Emailid
		76584123	4235400034568756	Zhang	Wei	zhang.wei@example.com
	OFR#10001	AccountId	OfferId	OfferType	AccountOfferStartDate	AccountOfferEndDate
		76584123	10001	BAX	2020-05-01	2020-08-01
	OFR#10002	AccountId	OfferId	OfferType	AccountOfferStartDate	AccountOfferEndDate
		76584123	10002	BAX	2020-06-01	2020-09-01

How to read data from DynamoDB

- GetItem
  - Specify exact value of primary key (partition key & sort key)
  - Returns exactly 0 or 1 items
  - Will consume Read Capacity Units (RCUs) based on the size of the item
- Query
  - Specify exact value of partition key and optionally a sort key condition
  - Optionally add filter conditions on non-key attributes
  - Returns matching items (possibly multiple)
  - Will consume RCUs based on the size of the items matching the key conditions, returning a single aggregated result
- Scan
  - Don't specify any keys! Optionally specify filter conditions on non-key attributes
  - Returns all items from the table that match filter expression
  - Will consume RCUs to read all items on the table (think carefully)

Data types

Data Type	DynamoDB Type
String	String
Integer, Float	Number
Timestamp	Number or String
Blob	Binary
Boolean	Bool
Null	Null
List	List
Set	Set of String, Number, or Binary
Map	Map

Operation types

Data Operations
GetItem
Query
Scan
BatchGetItem
PutItem
UpdateItem
DeleteItem
BatchWriteItem
TransactGetItems
TransactWriteItems

SQL-compatible access to DynamoDB

- Use PartiQL (a SQL-compatible query language) to query, insert, update, and delete table data in Amazon DynamoDB
- PartiQL is supported for all data plane operations
- PartiQL Operations
  - *ExecuteStatement:* Supports single/multiple item SELECT and single item INSERT, UPDATE and DELETE
  - *BatchExecuteStatement:* Supports a batch of single item SELECT OR batch of single item INSERT, UPDATE or DELETE of up to 25 items
  - *ExecuteTransaction:* Supports all-or-nothing changes to multiple items both within and across tables

Global secondary index (GSI)

Alternate partition and/or sort key
Index is across all partition keys

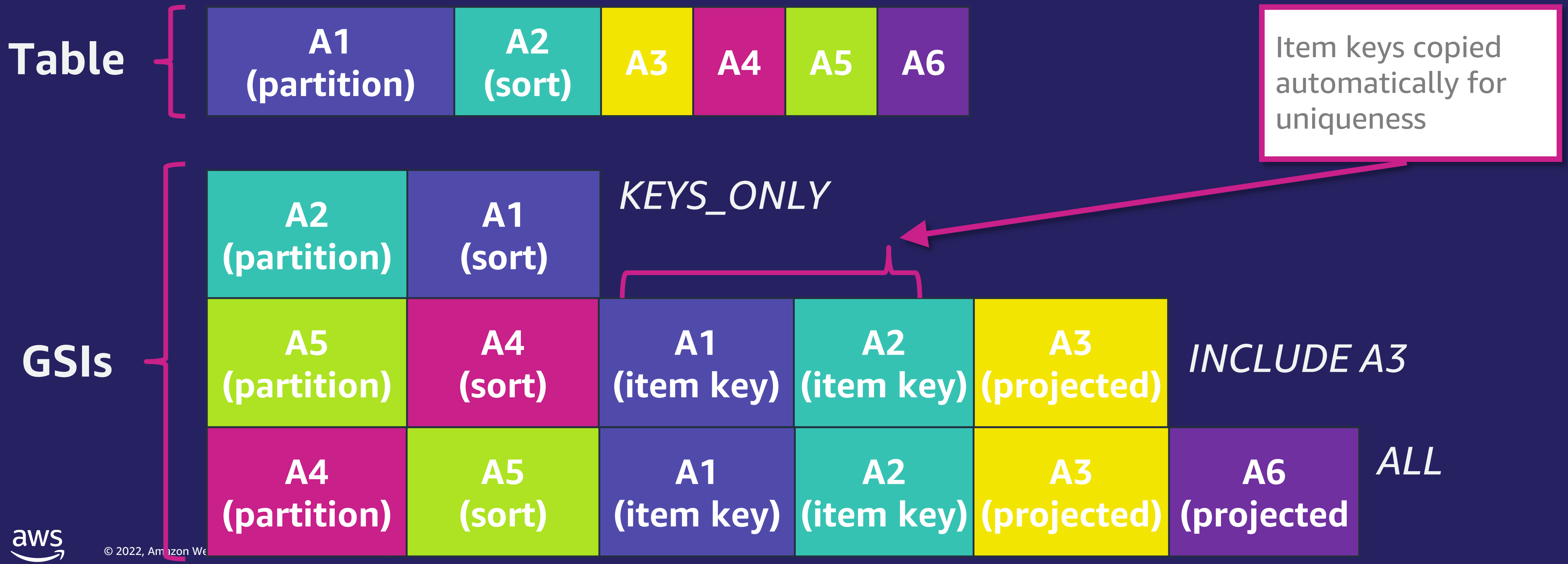
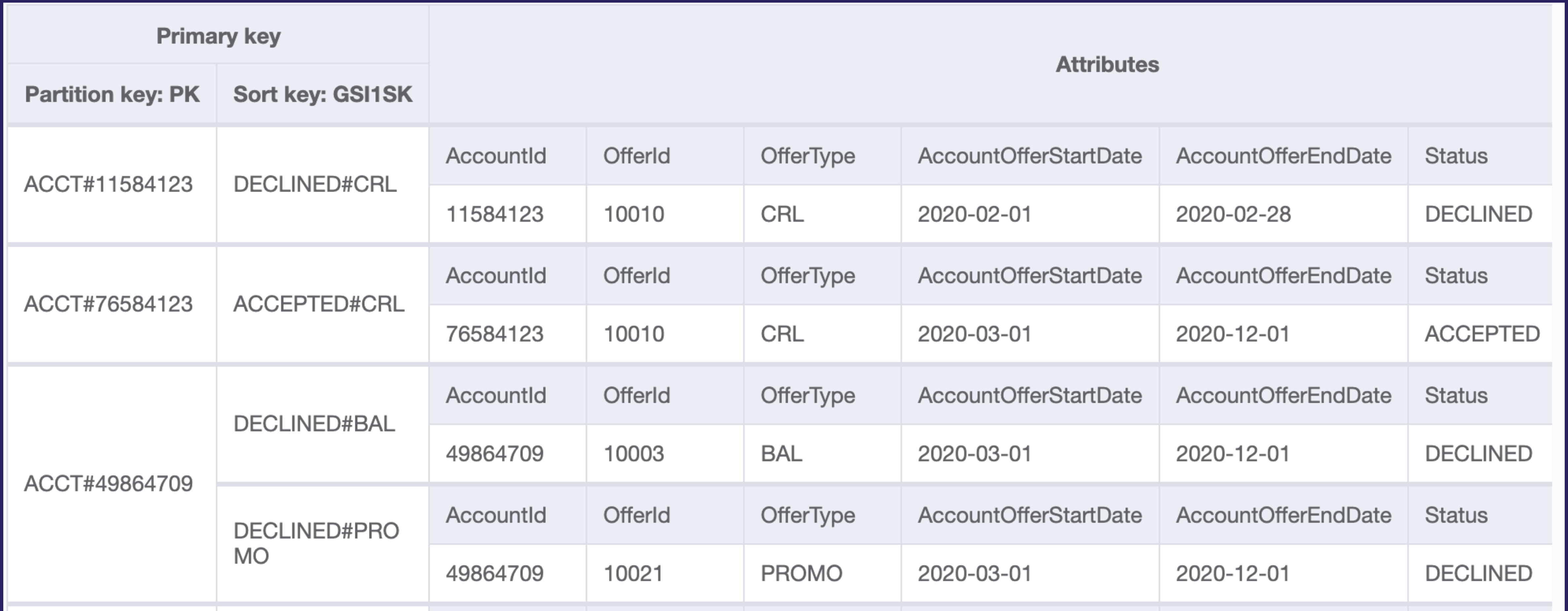


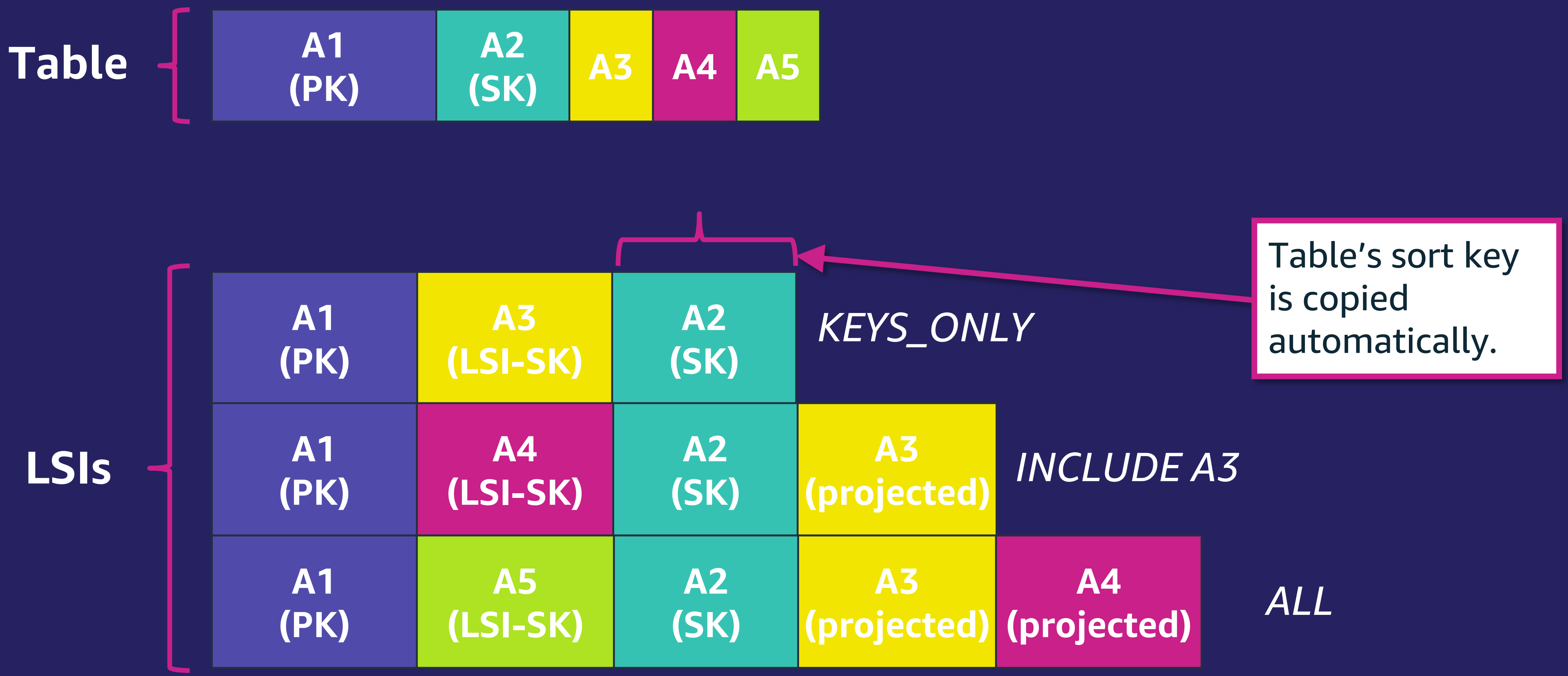
Table	A1 (partition)	A2 (sort)	A3	A4	A5	A6	
GSIs	A2 (partition)	A1 (sort)					*KEYS_ONLY*
	A5 (partition)	A4 (sort)	A1 (item key)	A2 (item key)	A3 (projected)		*INCLUDE A3*
	A4 (partition)	A5 (sort)	A1 (item key)	A2 (item key)	A3 (projected)	A6 (projected	*ALL*

Item keys copied automatically for uniqueness



Primary key		Attributes					
Partition key: PK	Sort key: GSI1SK						
ACCT#11584123	DECLINED#CRL	AccountId	OfferId	OfferType	AccountOfferStartDate	AccountOfferEndDate	Status
		11584123	10010	CRL	2020-02-01	2020-02-28	DECLINED
ACCT#76584123	ACCEPTED#CRL	AccountId	OfferId	OfferType	AccountOfferStartDate	AccountOfferEndDate	Status
		76584123	10010	CRL	2020-03-01	2020-12-01	ACCEPTED
ACCT#49864709	DECLINED#BAL	AccountId	OfferId	OfferType	AccountOfferStartDate	AccountOfferEndDate	Status
		49864709	10003	BAL	2020-03-01	2020-12-01	DECLINED
	DECLINED#PROMO	AccountId	OfferId	OfferType	AccountOfferStartDate	AccountOfferEndDate	Status
		49864709	10021	PROMO	2020-03-01	2020-12-01	DECLINED

Local secondary index (LSI)



Table

A1 (PK)	A2 (SK)	A3	A4	A5

LSIs

A1 (PK)	A3 (LSI-SK)	A2 (SK)			*KEYS_ONLY*
A1 (PK)	A4 (LSI-SK)	A2 (SK)	A3 (projected)		*INCLUDE A3*
A1 (PK)	A5 (LSI-SK)	A2 (SK)	A3 (projected)	A4 (projected)	*ALL*

Table’s sort key is copied automatically.

LSI and GSI side by side

LSI	GSI
Create at table creation	Create at any time
Shares WCU/RCU with table	WCU/RCU independent of table
Collection size <= 10GB	No size limits
Limit = 5	Limit = 20
Strong Consistency	Eventual Consistency

DynamoDB (Part 2)

Database Modernization Week

Jason Hunter
Principal Solution Architect

Agenda

Part 1

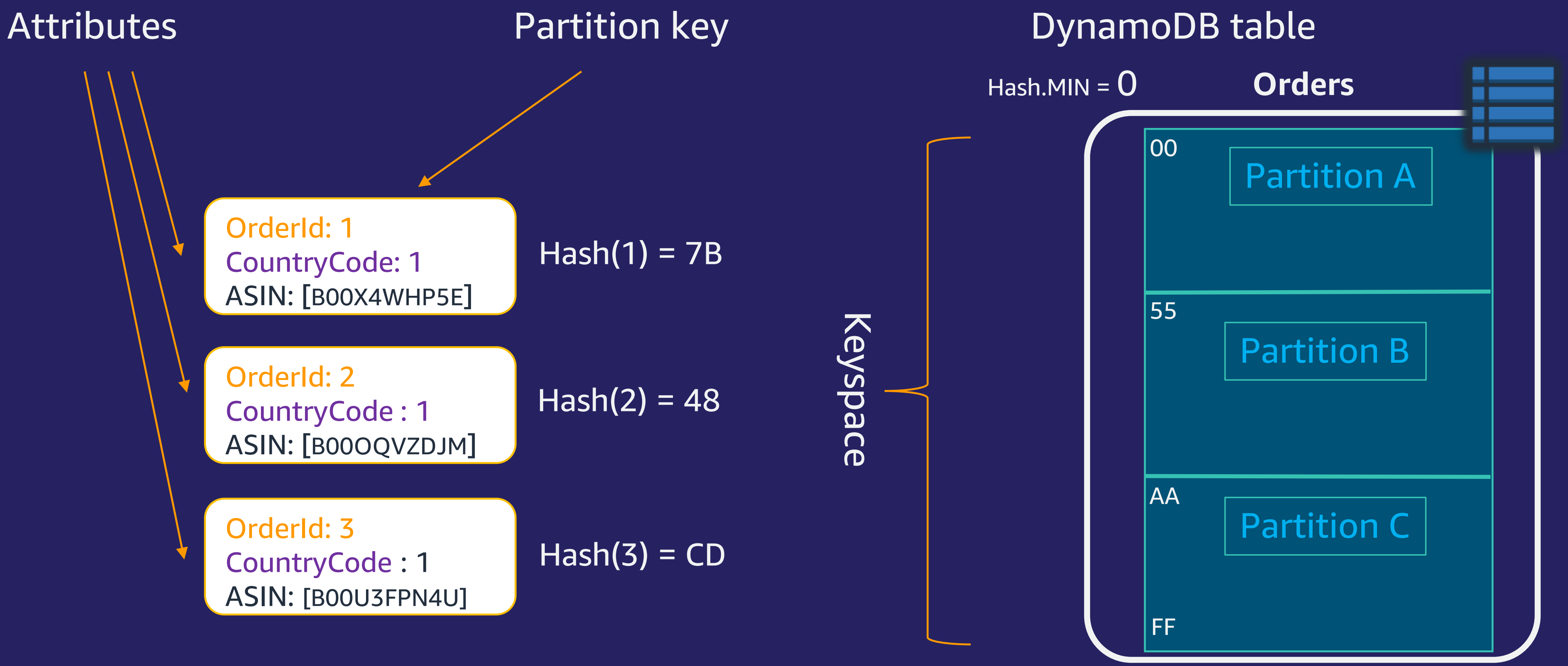
- What's the purpose of DynamoDB?
- What are its main features?
- Understanding its key concepts

Part 2

- **Looking under the hood**
- **Managing throughput**
- **Advanced usage patterns**
- **Introducing Standard-Infrequent Access table class**

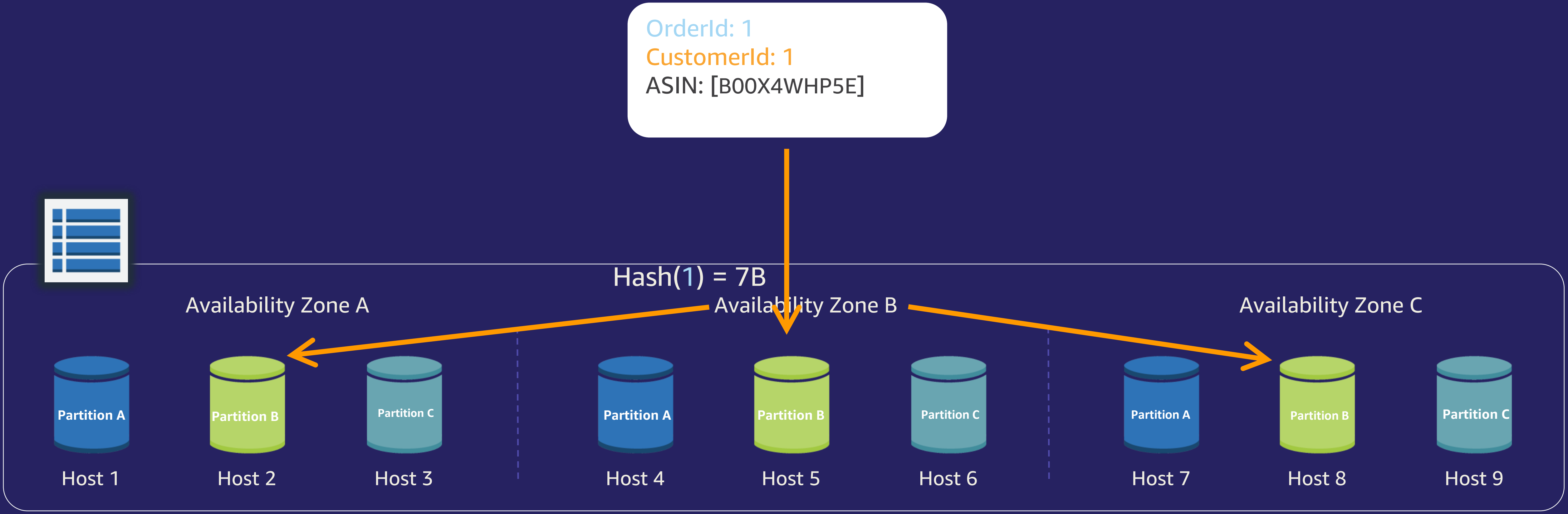
Under the Hood

Item Distribution



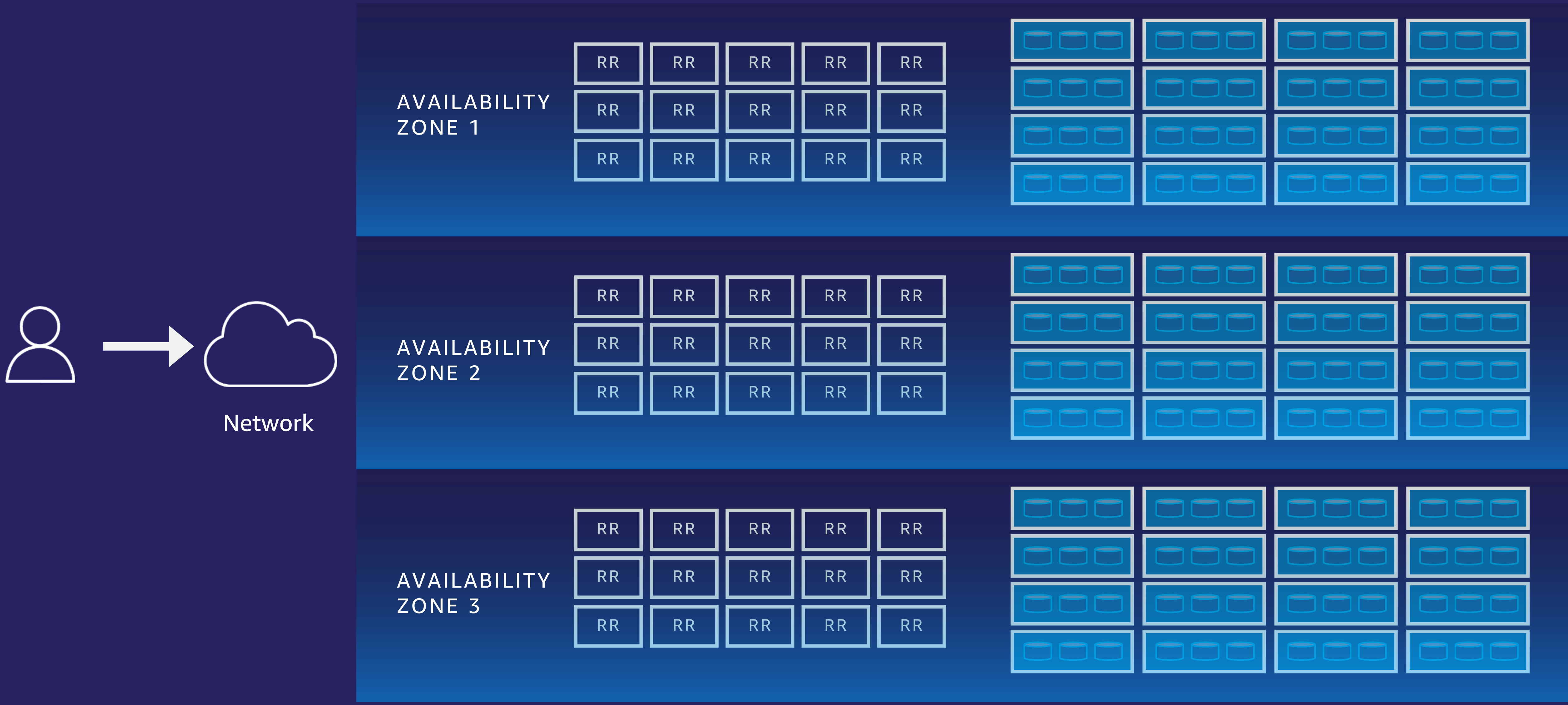
Whole item is stored together for efficient access

A view “from a different angle”

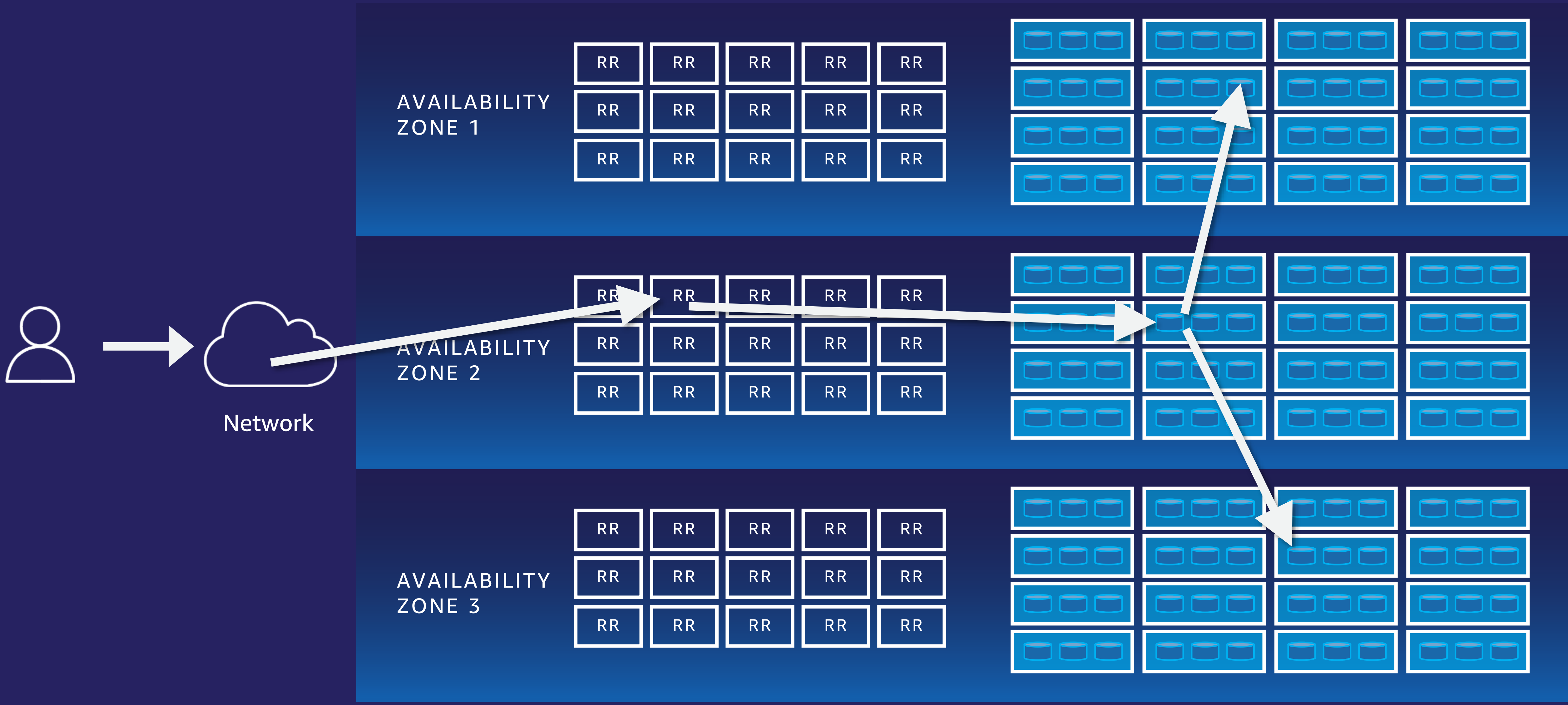


CustomerOrdersTable

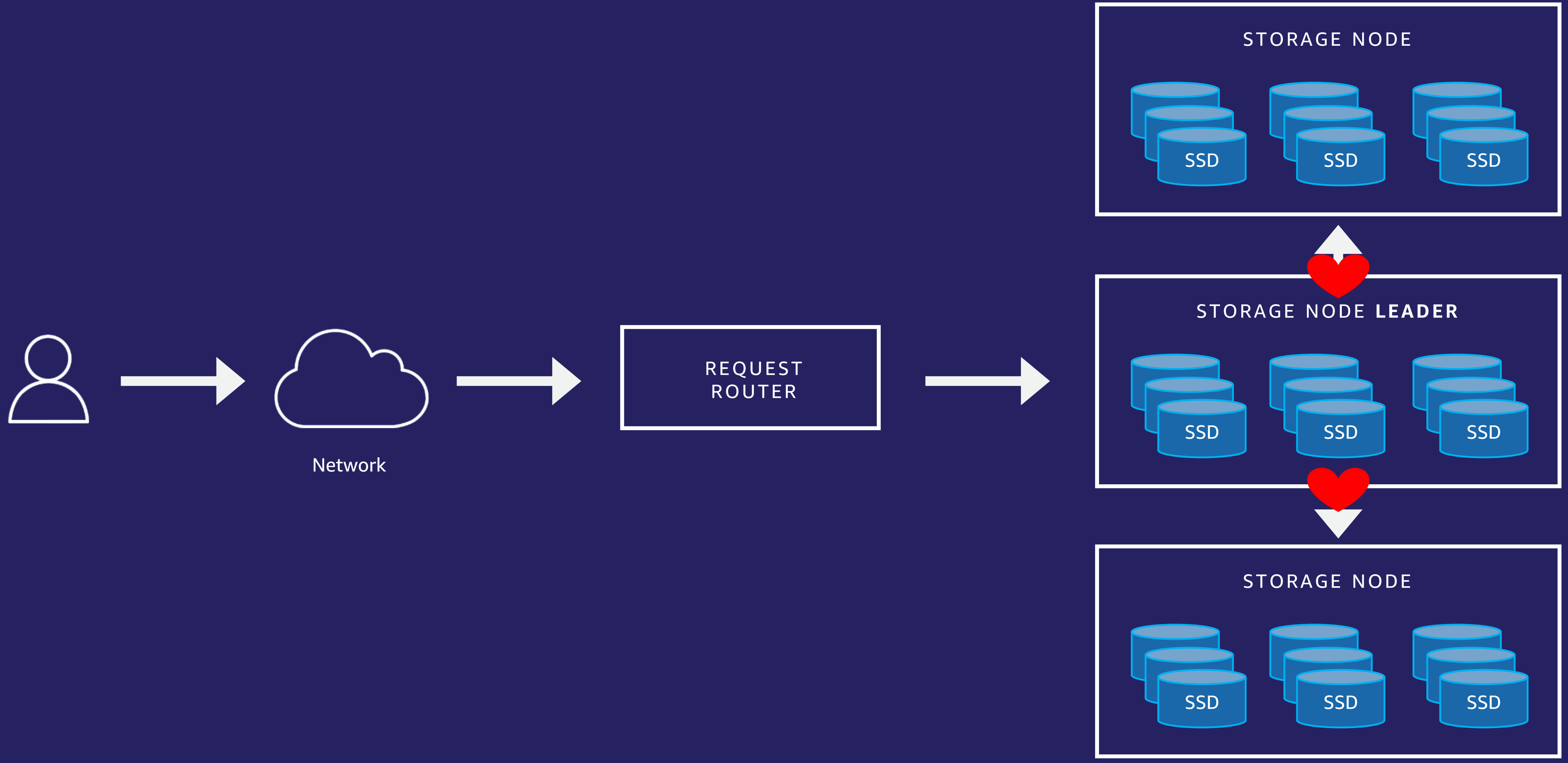
Service at Scale



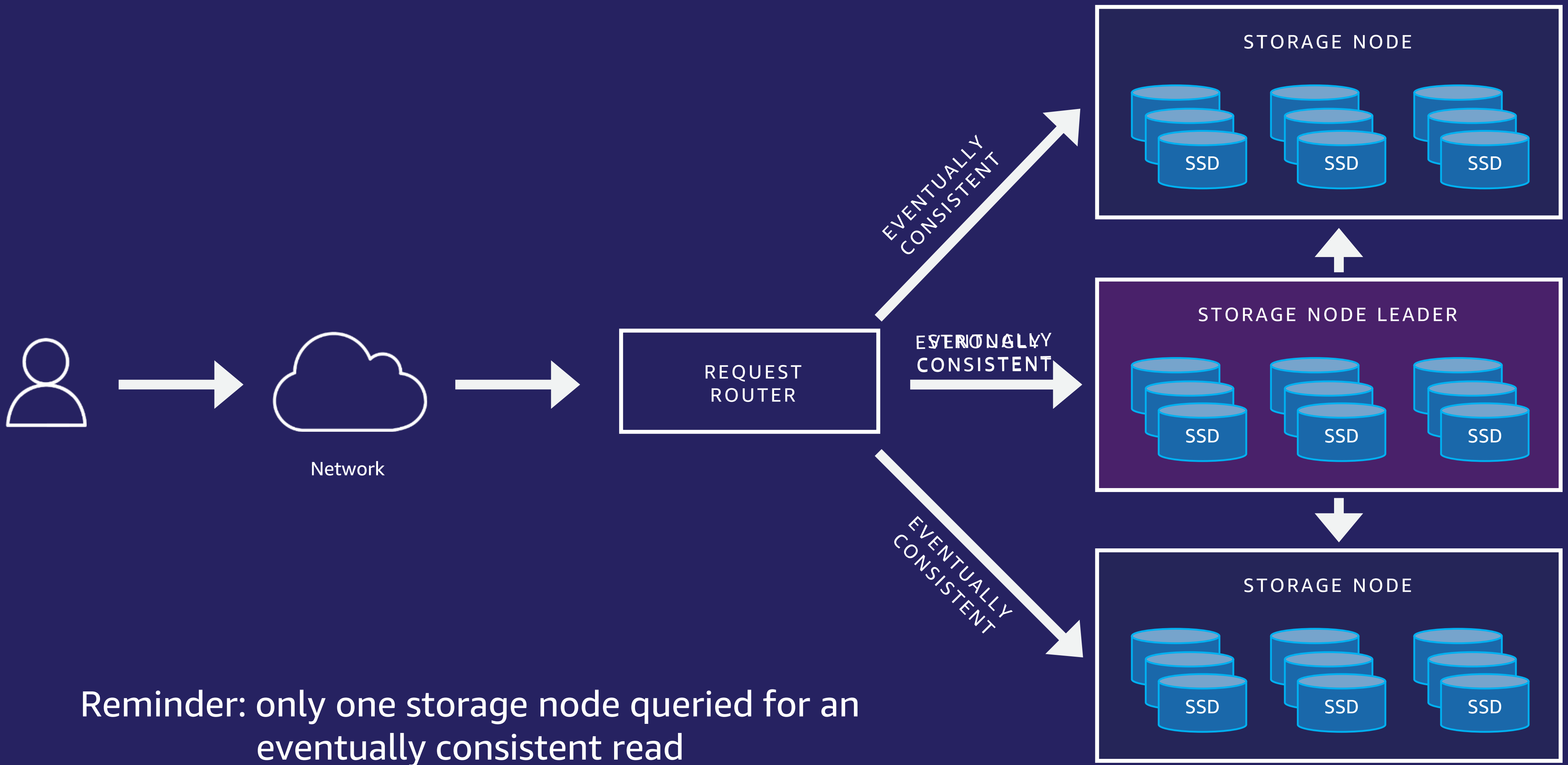
Path of a PutItem request



Heartbeats

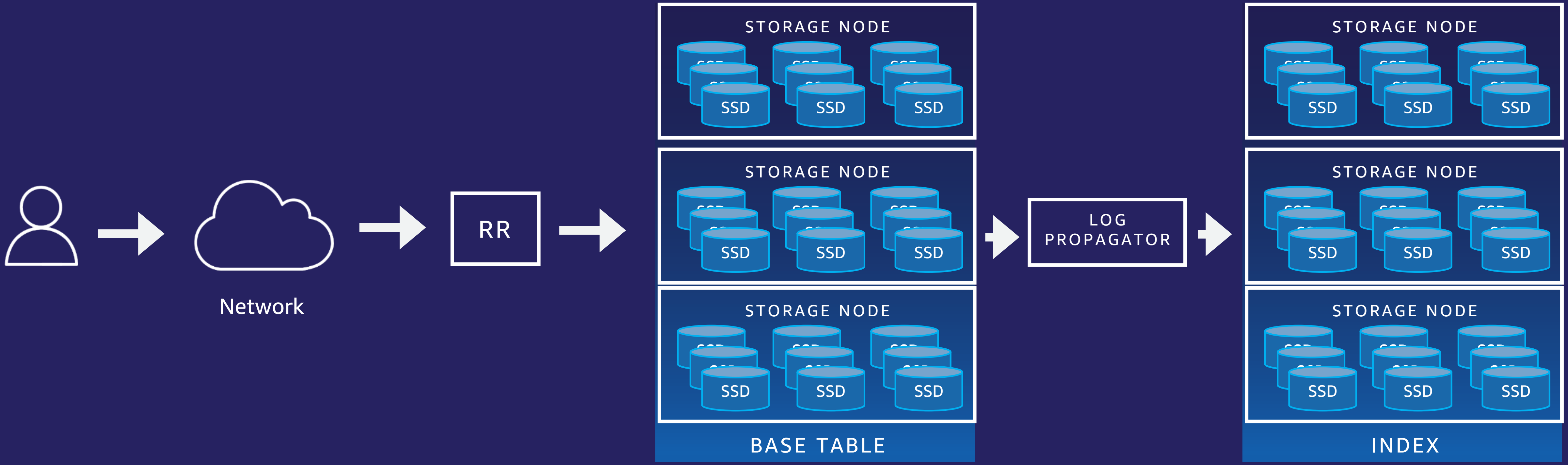

Network
REQUEST ROUTER
STORAGE NODE
SSD
SSD
SSD
STORAGE NODE **LEADER**
SSD
SSD
SSD
STORAGE NODE
SSD
SSD
SSD

GetItem Consistency

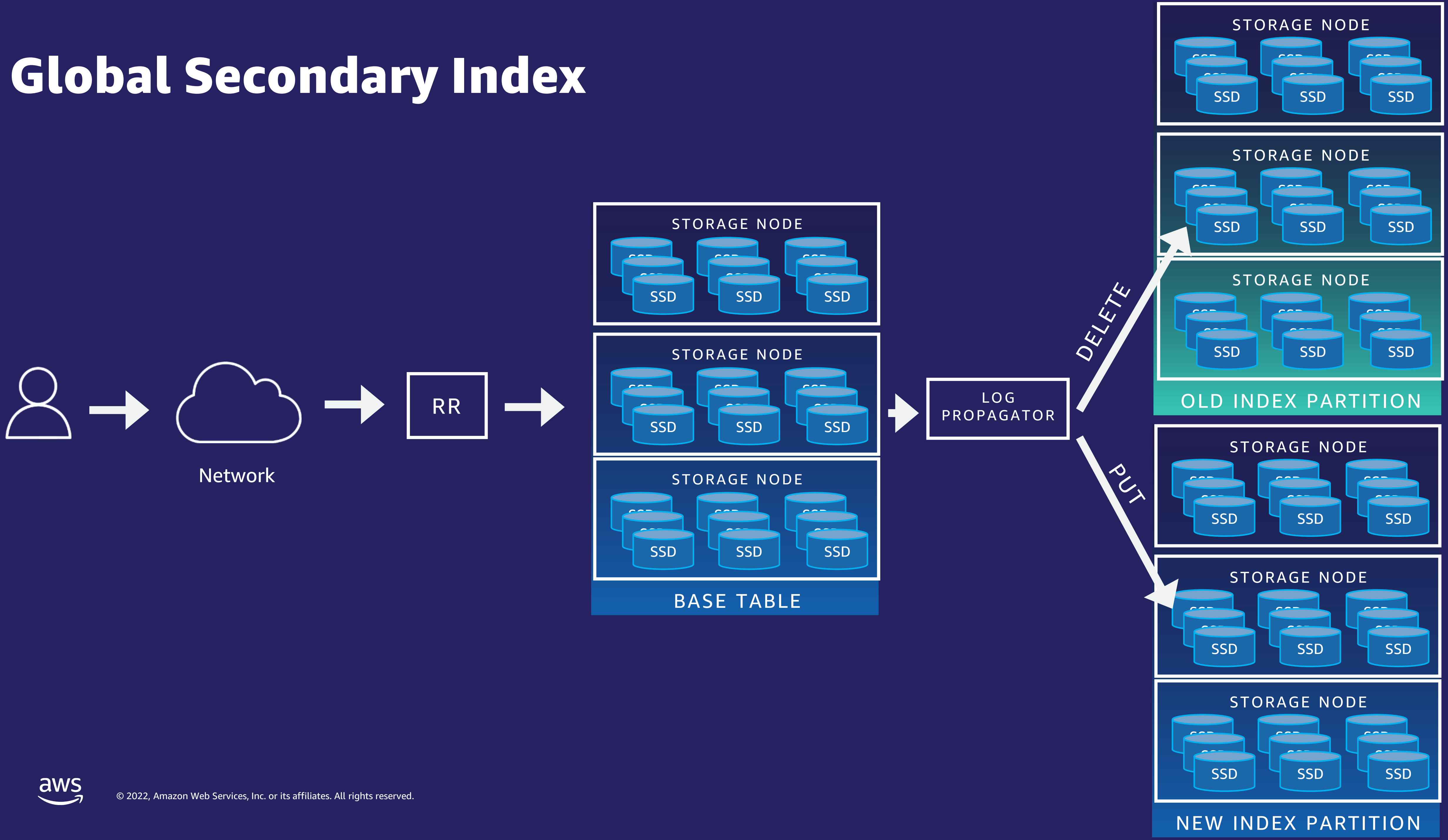


Reminder: only one storage node queried for an eventually consistent read

Global Secondary Index



Global Secondary Index



Auto Admin

- Create tables and indexes
- Table and index provisioning
- Splitting partitions
- Partition repairs

… (Automated DBA for DynamoDB)

Managing Throughput

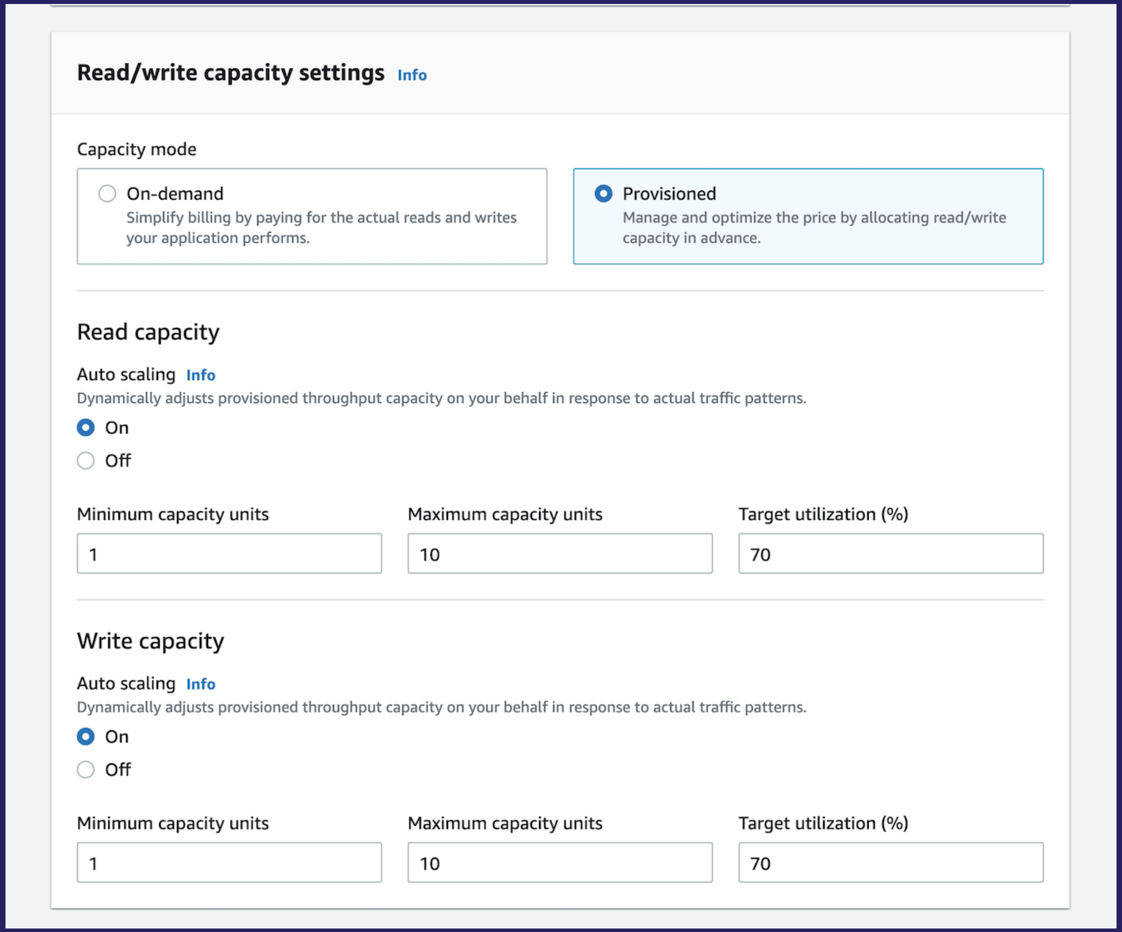
Scaling

- Throughput
  - Provision any amount of throughput to a table
  - Read Capacity Unit (RCU) – a 4 KB request On Demand, 4 KB/sec Provisioned
  - Write Capacity Unit (WCU) – a 1 KB request On Demand, 1 KB/sec Provisioned
  - Independent of each other
  - Eventually Consistent reads consume at half the rate

- Size
  - Add any number of items to a table
    - Max item size is 400 KB

- Scaling is achieved through partitioning
  - Each virtual partition delivers 1000 WCU/second or 3000 RCU/second (or a mix)
  - Split based on Capacity = When exceeding this limit
  - Split based on Size = When exceeding 10 GB

Provisioning Table Capacity



Read/write capacity settings Info

Capacity mode

○ On-demand
Simplify billing by paying for the actual reads and writes your application performs.

● Provisioned
Manage and optimize the price by allocating read/write capacity in advance.

Read capacity

Auto scaling Info
Dynamically adjusts provisioned throughput capacity on your behalf in response to actual traffic patterns.

● On
○ Off

Minimum capacity units	Maximum capacity units	Target utilization (%)
1	10	70

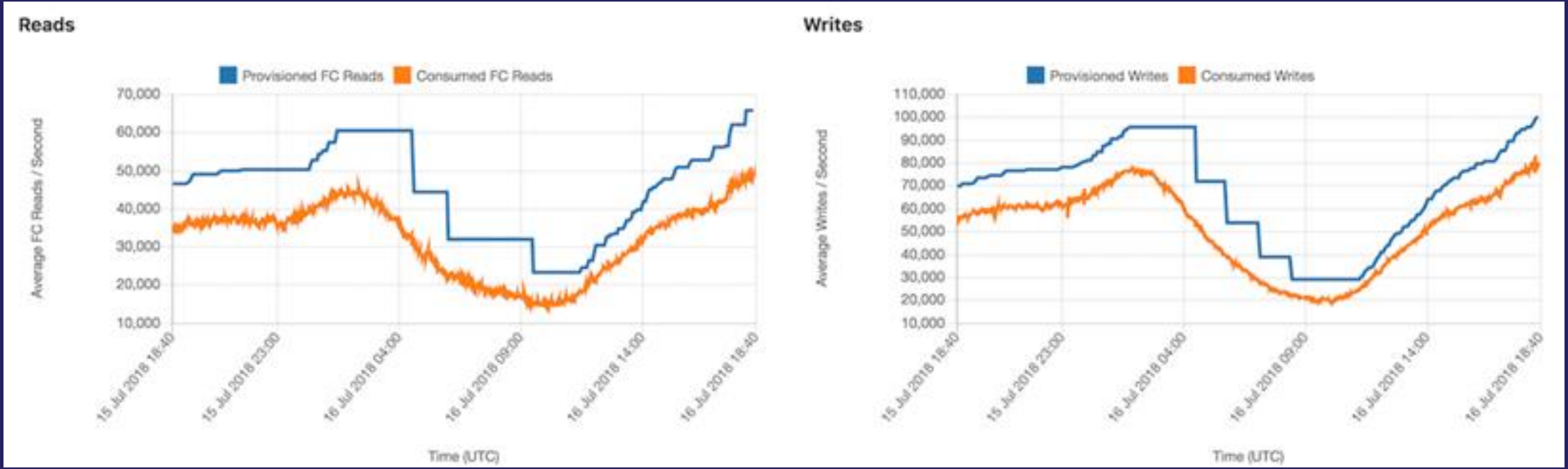
Write capacity

Auto scaling Info
Dynamically adjusts provisioned throughput capacity on your behalf in response to actual traffic patterns.

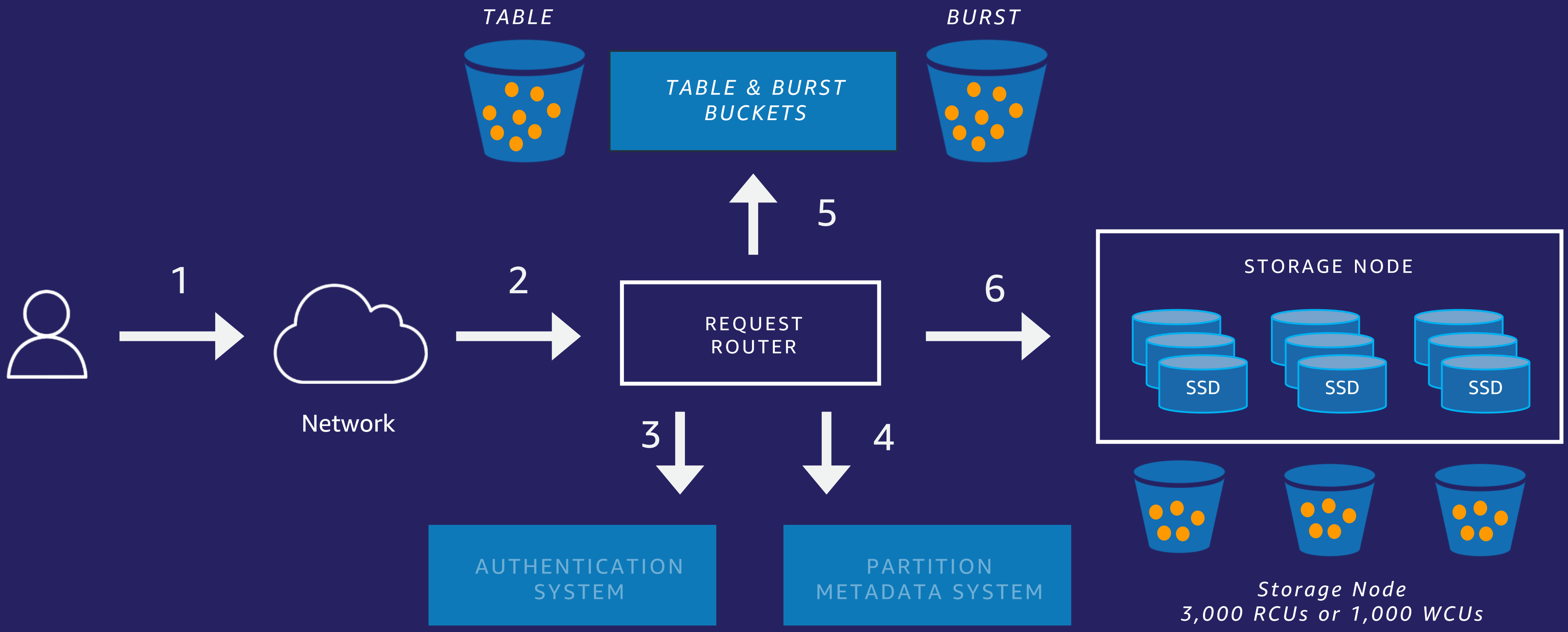
● On
○ Off

Minimum capacity units	Maximum capacity units	Target utilization (%)
1	10	70

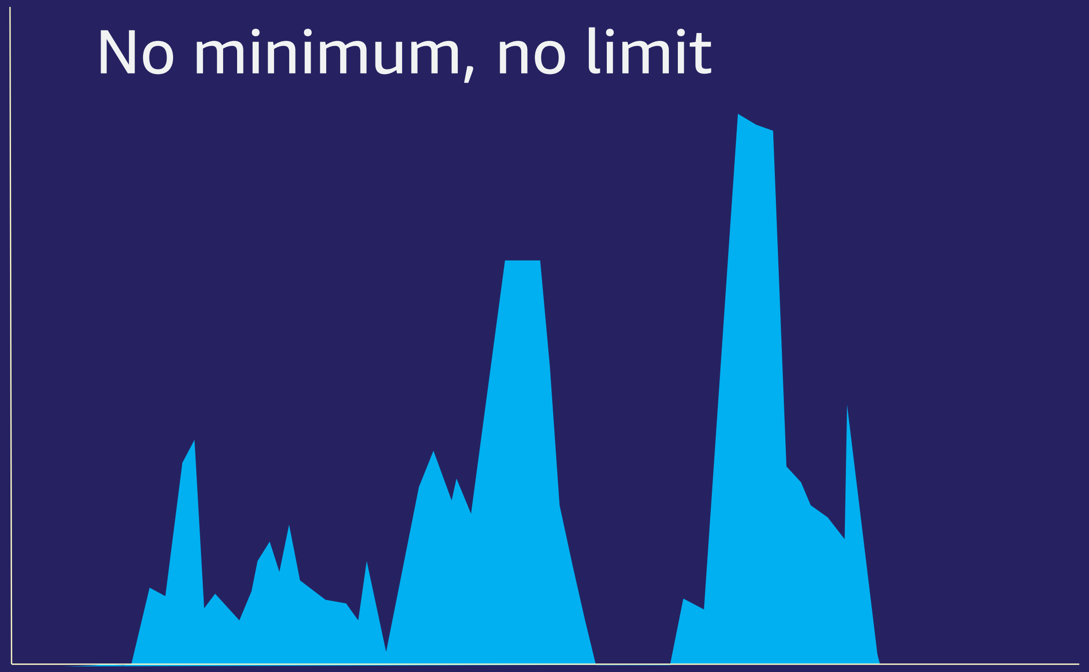
Auto Scaling



Token Buckets Manage Provisioned Throughput



DynamoDB on-demand capacity mode



Features

- No capacity planning, provisioning, or reservations—simply make API calls
- Pay only for the reads and writes you perform

Key benefits

- Eliminates tradeoffs of overprovisioning or underprovisioning
- Instantly accommodates your workload as traffic ramps up or down

On-demand scaling properties

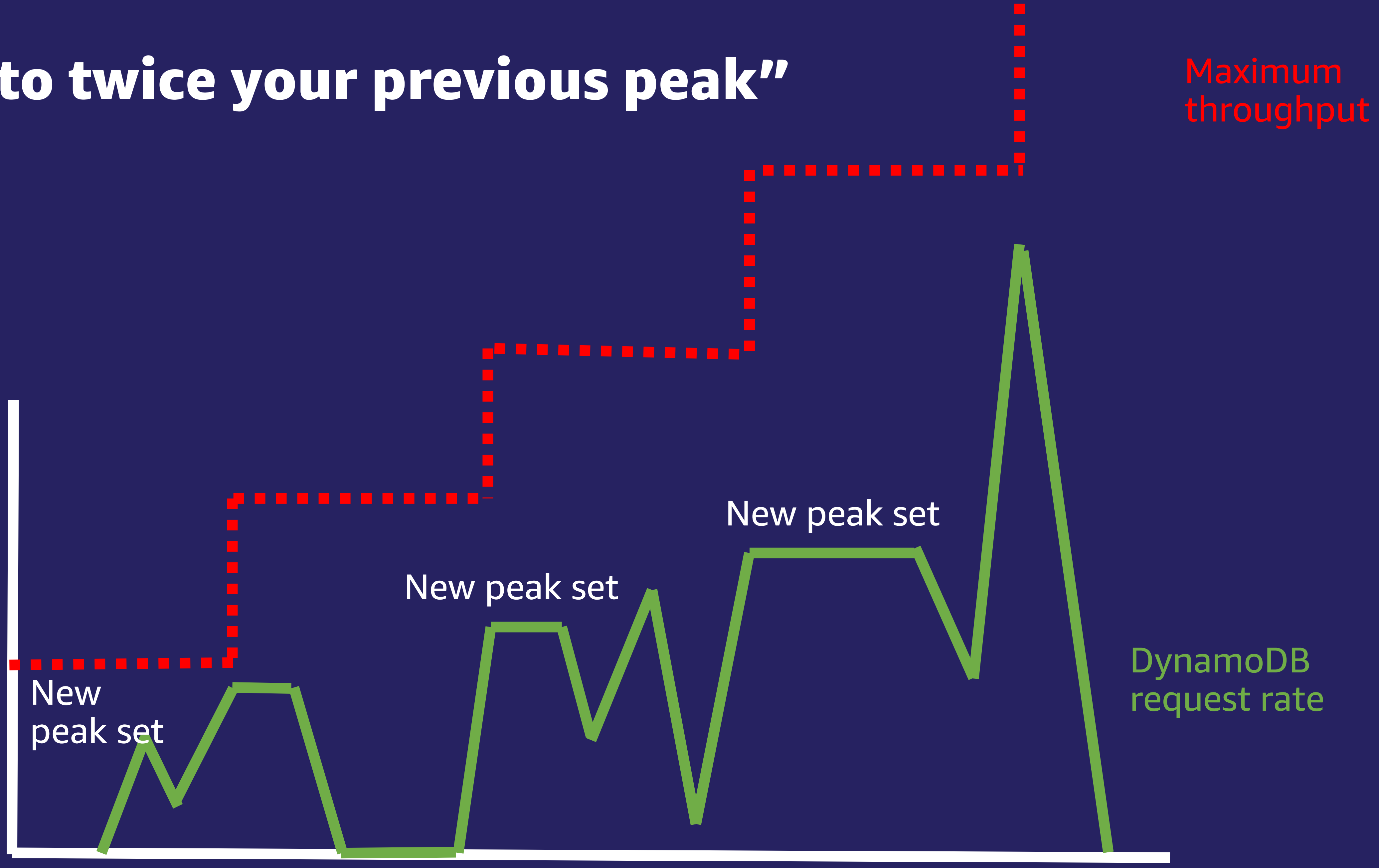
New table default throughput

- Up to 4,000 write request units: 4,000 writes per second
- Up to 12,000 read request units: 24,000 eventually consistent reads per second
- Any linear combination of the two
- **Grows under load to support twice the previous peak**

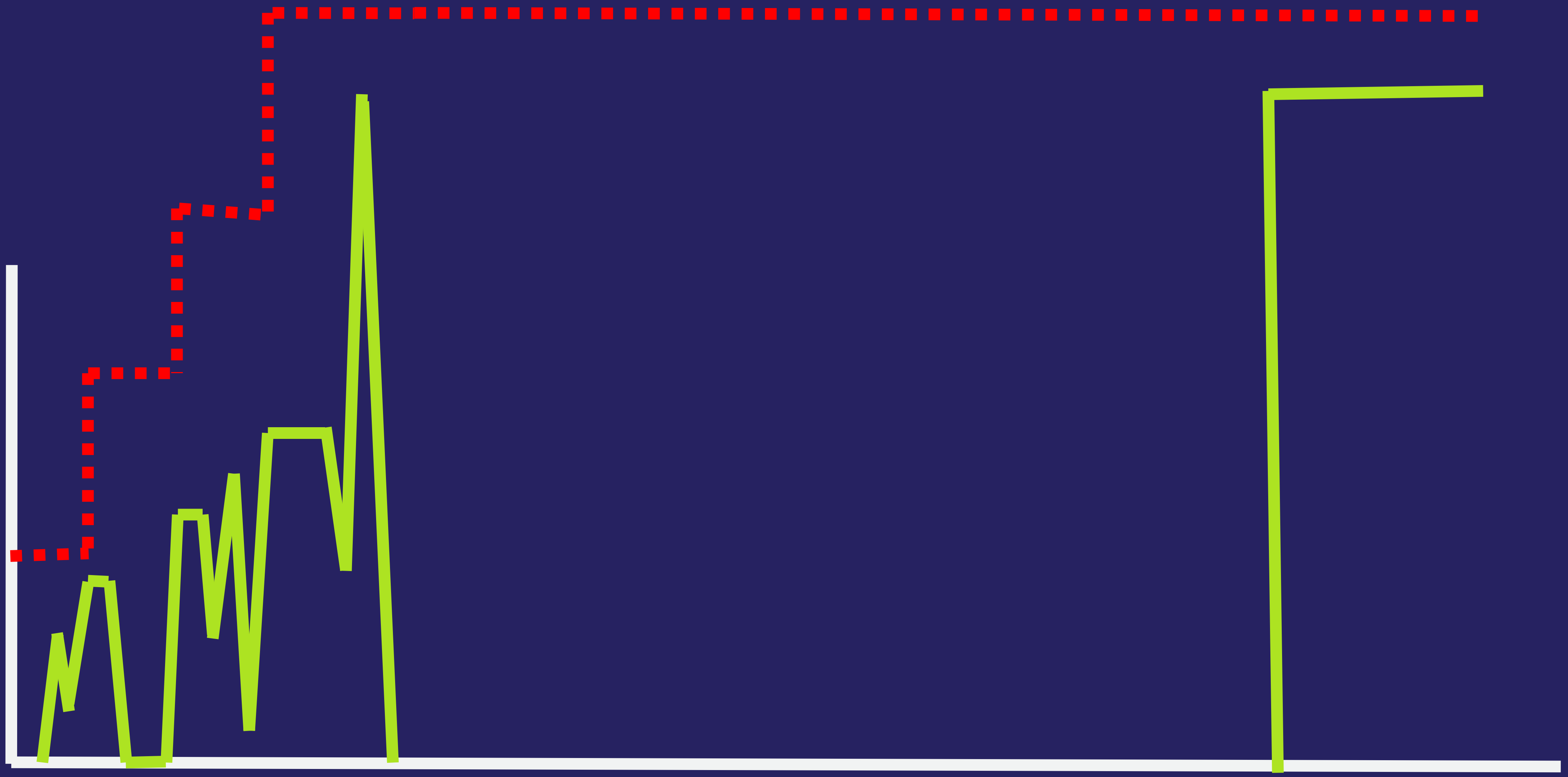
Maximum throughput

Unlimited!

“Up to twice your previous peak”



On-demand tables do not “scale down”



Provisioned or On Demand?

Use provisioned mode

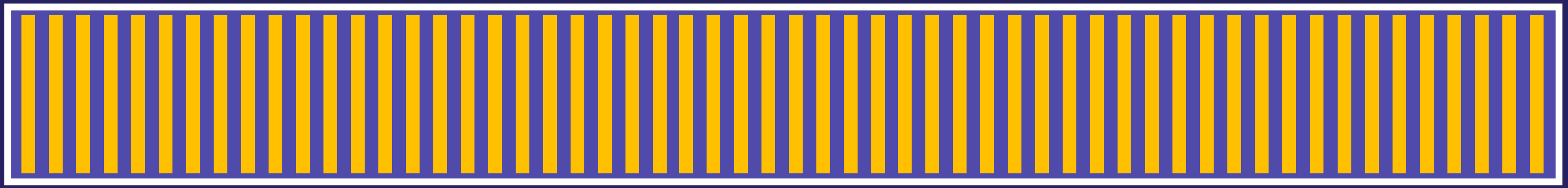
- Steady workloads
- Gradual ramps
- Events with known traffic
- Ongoing monitoring

Use on-demand mode

- Unpredictable workloads
- Frequently idle workloads
- Events with unknown traffic
- “Set it and forget it”

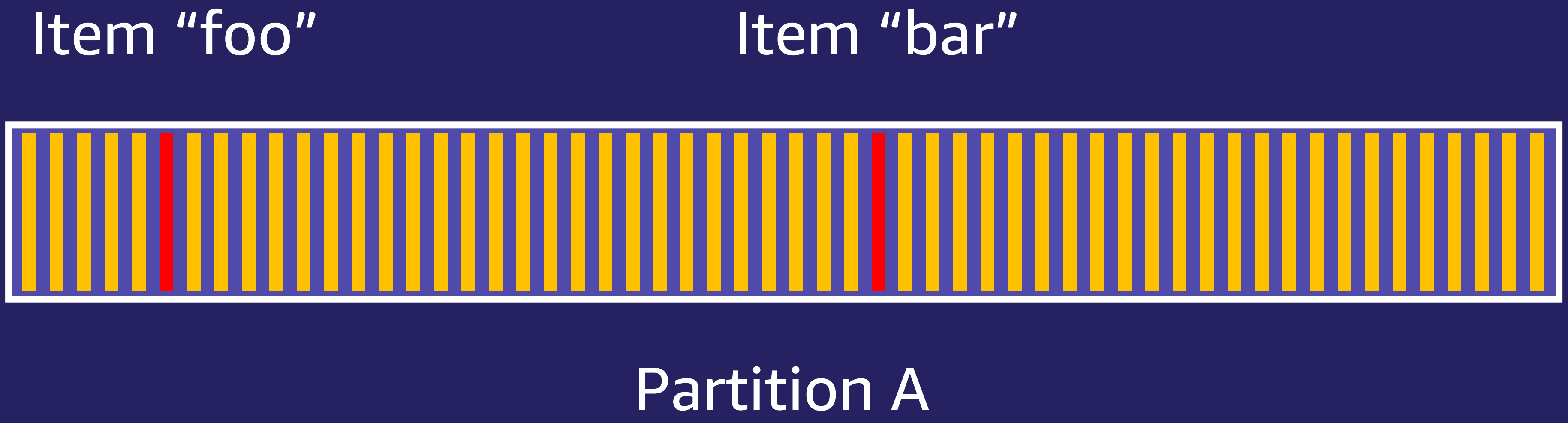
Consider your tolerance for operational overhead and overprovisioning

High-traffic item isolation



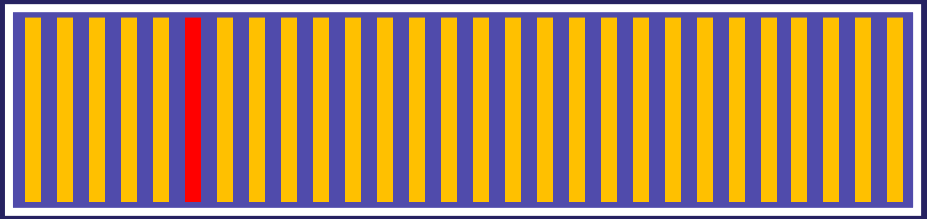
Partition A

High-traffic item isolation



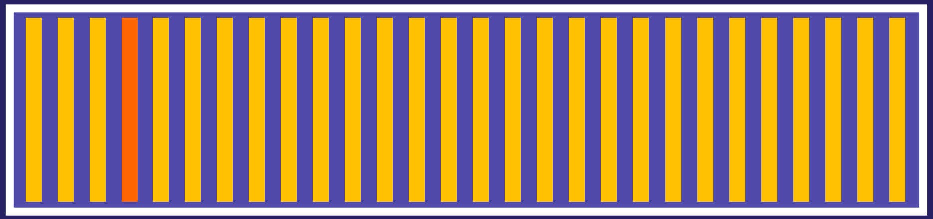
High-traffic item isolation

Item “foo”



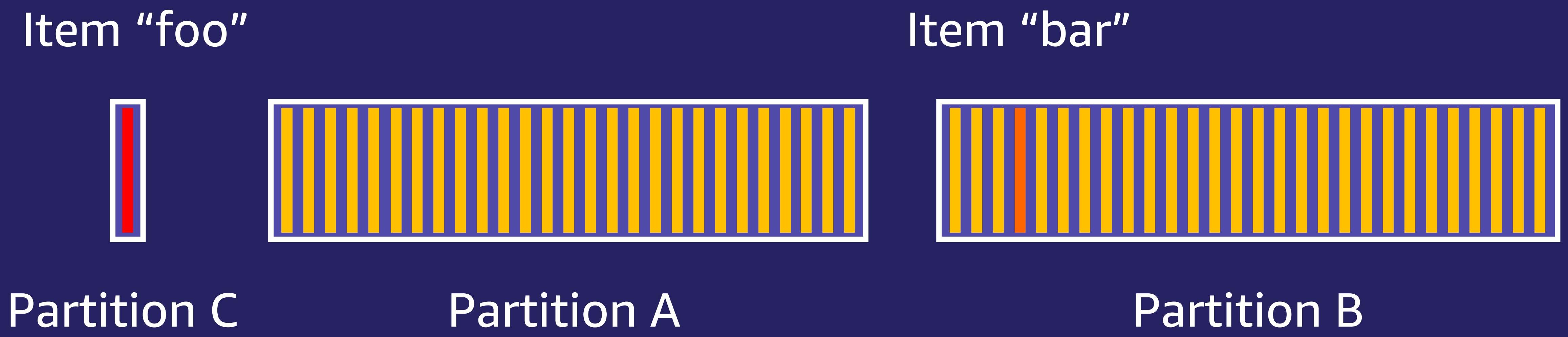
Partition A

Item “bar”



Partition B

High-traffic item isolation



Item “foo”

Item “bar”

Partition C

Partition A

Partition B

Amazon CloudWatch Contributor Insights for DynamoDB

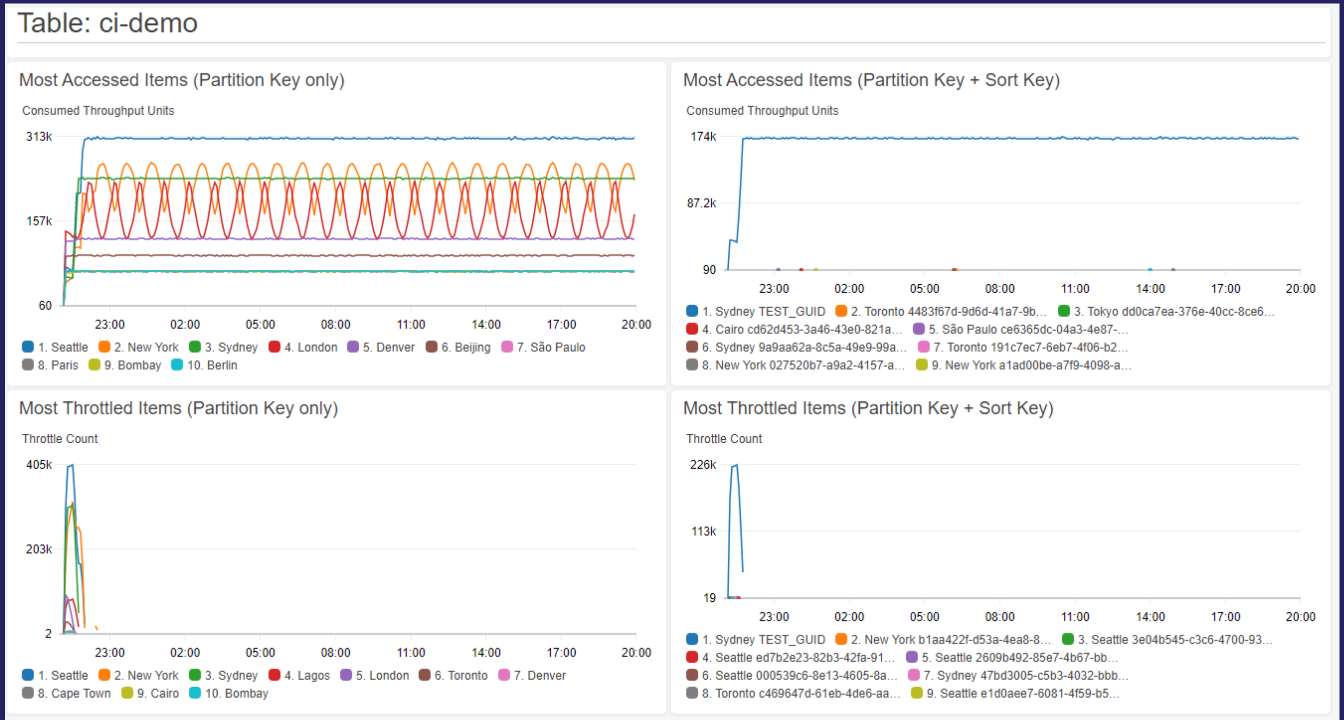


Table: ci-demo
Most Accessed Items (Partition Key only)
Consumed Throughput Units
313k
157k
60
23:00
02:00
05:00
08:00
11:00
14:00
17:00
20:00
🟦 1. Seattle
🟧 2. New York
🟩 3. Sydney
🟥 4. London
🟪 5. Denver
🟫 6. Beijing
7. São Paulo
8. Paris
9. Bombay
10. Berlin
Most Accessed Items (Partition Key + Sort Key)
Consumed Throughput Units
174k
87.2k
90
23:00
02:00
05:00
08:00
11:00
14:00
17:00
20:00
🟦 1. Sydney TEST_GUID
🟧 2. Toronto 4483f67d-9d6d-41a7-9b...
🟩 3. Tokyo dd0ca7ea-376e-40cc-8ce6...
🟥 4. Cairo cd62d453-3a46-43e0-821a...
🟪 5. São Paulo ce6365dc-04a3-4e87-...
🟫 6. Sydney 9a9aa62a-8c5a-49e9-99a...
7. Toronto 191c7ec7-6eb7-4f06-b2...
8. New York 027520b7-a9a2-4157-a...
9. New York a1ad00be-a7f9-4098-a...
Most Throttled Items (Partition Key only)
Throttle Count
405k
203k
2
23:00
02:00
05:00
08:00
11:00
14:00
17:00
20:00
🟦 1. Seattle
🟧 2. New York
🟩 3. Sydney
🟥 4. Lagos
🟪 5. London
🟫 6. Toronto
7. Denver
8. Cape Town
9. Cairo
10. Bombay
Most Throttled Items (Partition Key + Sort Key)
Throttle Count
226k
113k
19
23:00
02:00
05:00
08:00
11:00
14:00
17:00
20:00
🟦 1. Sydney TEST_GUID
🟧 2. New York b1aa422f-d53a-4ea8-8...
🟩 3. Seattle 3e04b545-c3c6-4700-93...
🟥 4. Seattle ed7b2e23-82b3-42fa-91...
🟪 5. Seattle 2609b492-85e7-4b67-bb...
🟫 6. Seattle 000539c6-8e13-4605-8a...
7. Sydney 47bd3005-c5b3-4032-bbb...
8. Toronto c469647d-61eb-4de6-aa...
9. Seattle e1d0aee7-6081-4f59-b5...

Features

- Key-level activity graphs
- 1-click integration between DynamoDB and CloudWatch

Key benefits

- Identify frequently accessed keys and traffic trends at a glance
- Respond appropriately to unsuccessful requests

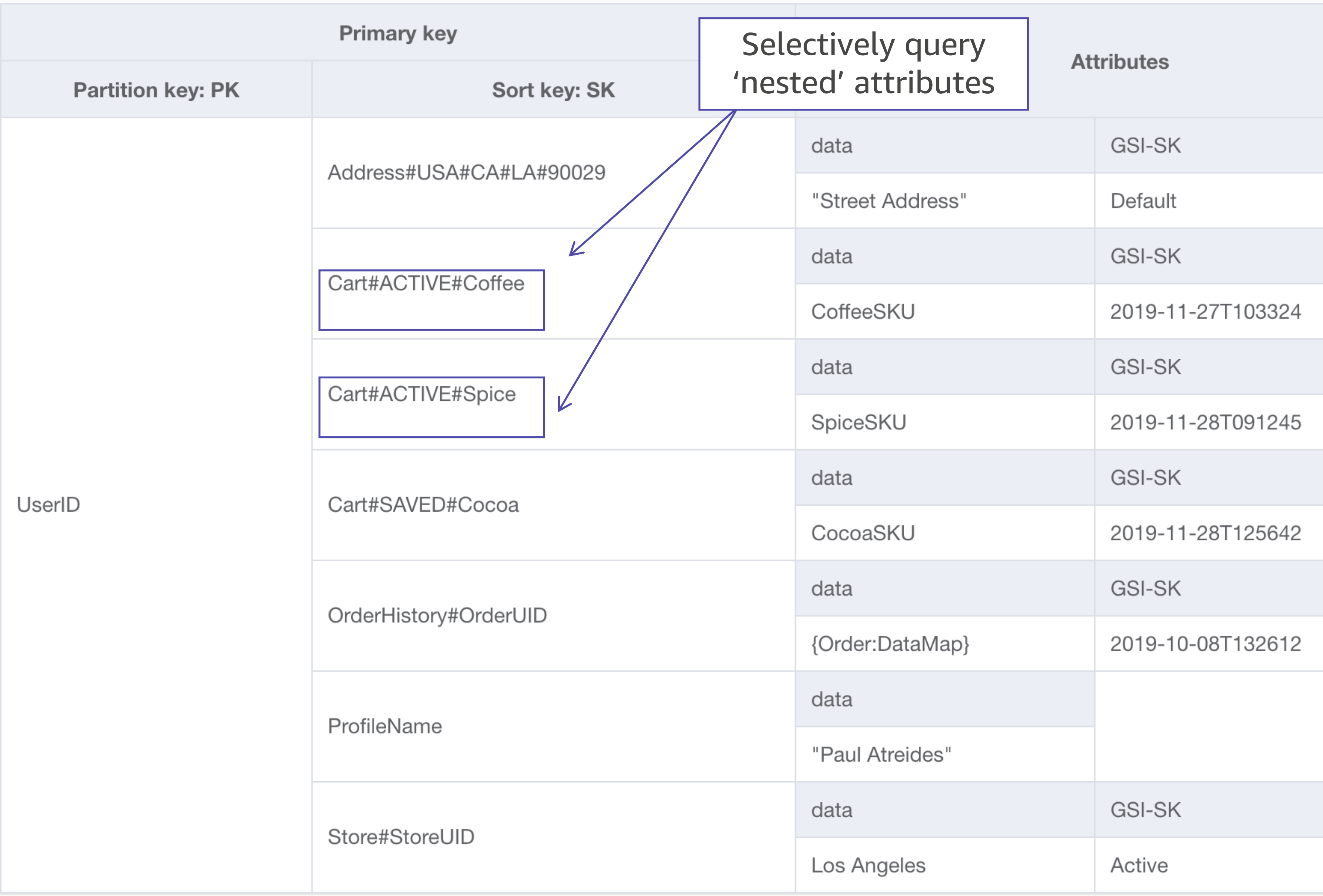
Schema Design Patterns

Example – Shopping Cart : Document Indexing

```
{
         "UserProfile" : {
                   "FirstName": "Paul",
                   "LastName": "Atreides",
                   "DateJoined":"1965-08-01"},
         "Store" : {
                   "store_id": "STOREUID",
                   "city": "Los Angeles",
                   "zip_code": "90029”}
         "ShoppingCart" : [
                  { "Spice":
                          { "SKU": "SpiceSKU",
                            "CategoryID": "FictionalSpice",
                            "DateAddded": "2019-06-11"}},
                  { "EspressoBeans":
                          { "SKU": "CaffeineSKU",
                            "CategoryID": "FOODANDDRINK",
                            "DateAddded": "2019-06-10"}}],
         "ShippingAddress" : {
                   "street_address": "1234 Arrakis Dr",
                   "city": "Los Angeles",
                   "zip_code": "90029",
                   "status": "default"}
         "OrderHistory#OrderUID" : {
                  "ProductA": "SKU_A",
                  "ProductB": "SKU_B",
                  "DateOrdered": "2018-09-28"}
}
```

aws

Vertical Partitioning



Primary key		Attributes	
Partition key: PK	Sort key: SK		
UserID	Address#USA#CA#LA#90029	data	GSI-SK
		"Street Address"	Default
	Cart#ACTIVE#Coffee	data	GSI-SK
		CoffeeSKU	2019-11-27T103324
	Cart#ACTIVE#Spice	data	GSI-SK
		SpiceSKU	2019-11-28T091245
	Cart#SAVED#Cocoa	data	GSI-SK
		CocoaSKU	2019-11-28T125642
	OrderHistory#OrderUID	data	GSI-SK
		{Order:DataMap}	2019-10-08T132612
	ProfileName	data	
		"Paul Atreides"	
	Store#StoreUID	data	GSI-SK
		Los Angeles	Active

Selectively query ‘nested’ attributes

Fetch items for a specific user that are active in the shopping cart.

BEGINS_WITH 'Cart#ACTIVE'

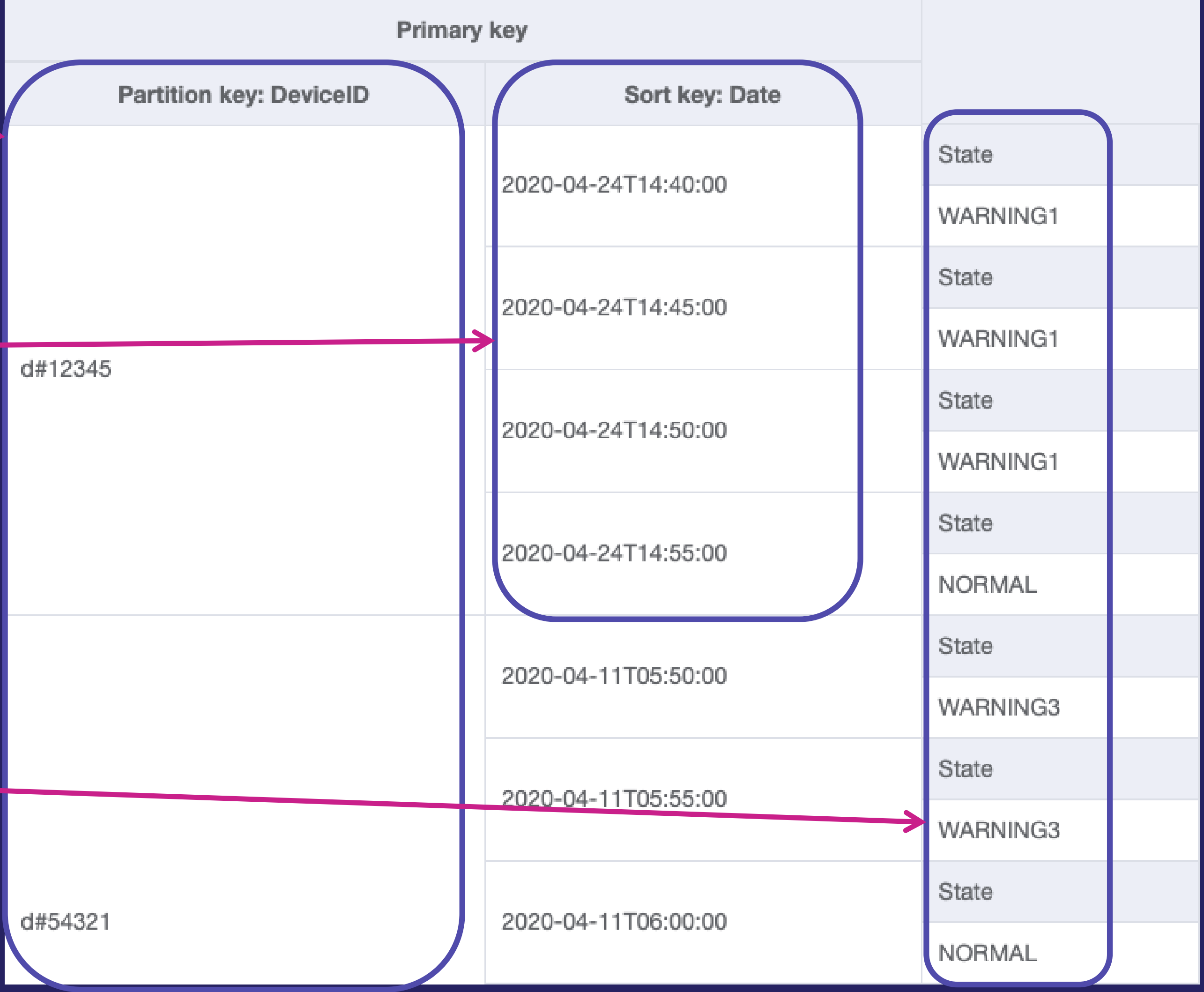
- Optimize object size
- Selective Queries
- Reduce capacity and cost
- Improve App performance

Example – Device Log

High cardinality

Sorted by date time

Filter by Status



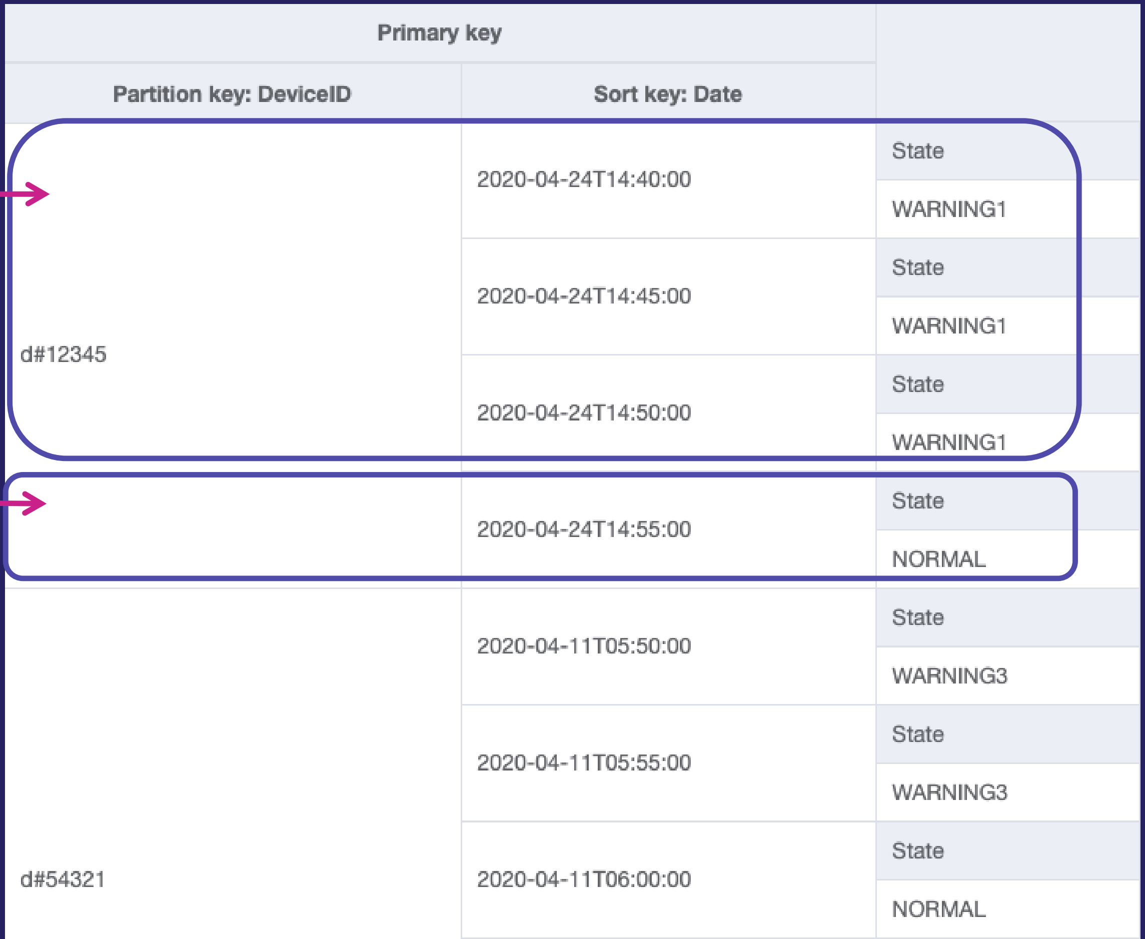
Primary key		
Partition key: DeviceID	Sort key: Date	
d#12345	2020-04-24T14:40:00	State WARNING1
	2020-04-24T14:45:00	State WARNING1
	2020-04-24T14:50:00	State WARNING1
	2020-04-24T14:55:00	State NORMAL
d#54321	2020-04-11T05:50:00	State WARNING3
	2020-04-11T05:55:00	State WARNING3
	2020-04-11T06:00:00	State NORMAL

Access Pattern: Fetch all warning logs for a device that are sorted in descending order

```
SELECT * FROM DeviceLog
WHERE DeviceID = 'd#12345'
ORDER BY Date DESC
FILTER ON State = 'WARNING1'
```

Returned →

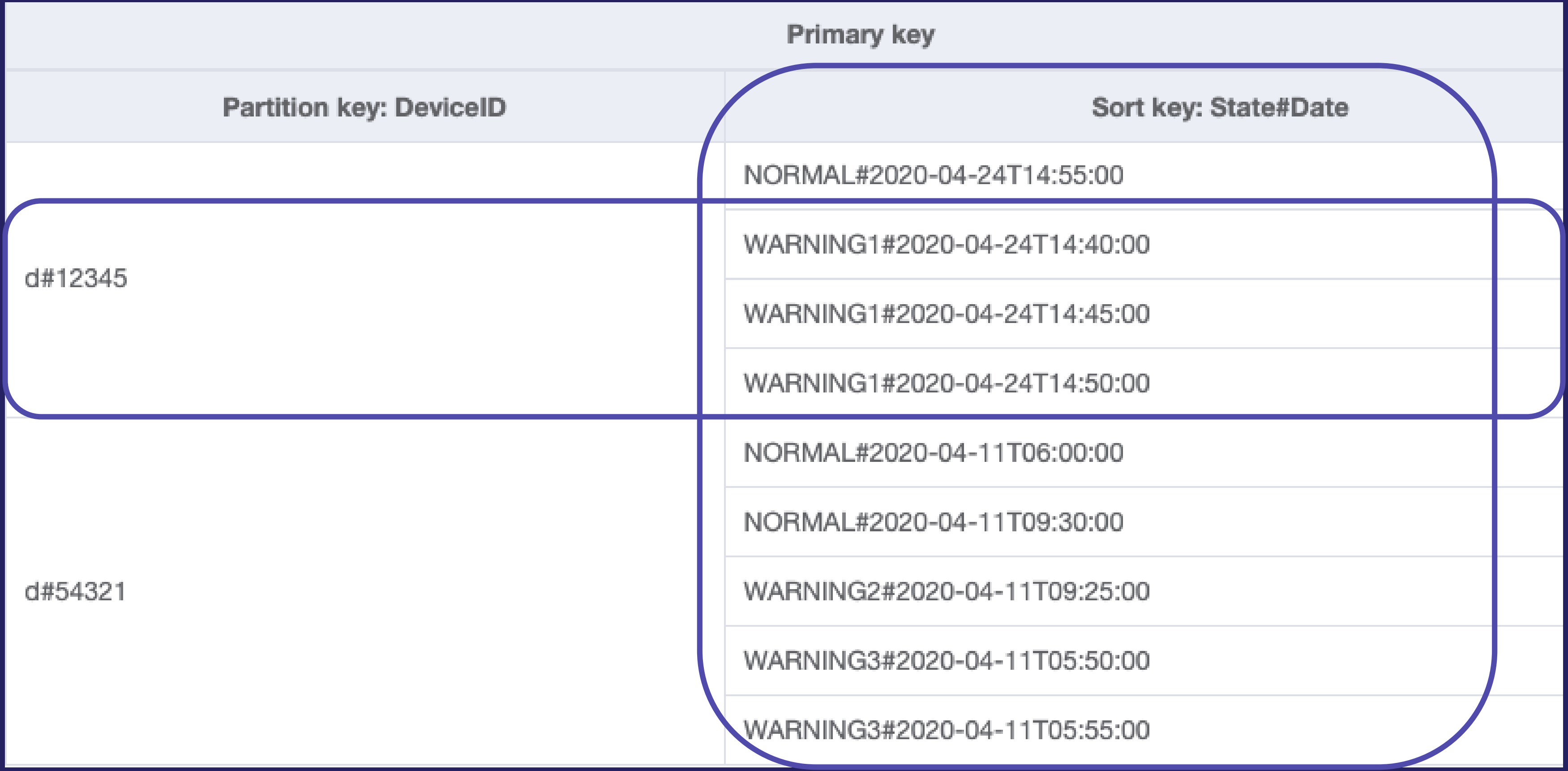
Filtered →



Primary key		
Partition key: DeviceID	Sort key: Date	
d#12345	2020-04-24T14:40:00	State WARNING1
	2020-04-24T14:45:00	State WARNING1
	2020-04-24T14:50:00	State WARNING1
	2020-04-24T14:55:00	State NORMAL
d#54321	2020-04-11T05:50:00	State WARNING3
	2020-04-11T05:55:00	State WARNING3
	2020-04-11T06:00:00	State NORMAL

```
aws dynamodb query
--table-name DeviceLog
--key-condition-expression ”#dID = :dID”
--no-scan-index-forward
--filter-expression "#s = :s”
--expression-attribute-names ‘{“#dID": ”DeviceID”, “#s”: “State}’
--expression-attribute-values ‘{“:dID”: {“S”:”d#12345”}, “:s”: {“S”:”WARNING1”}}’
```

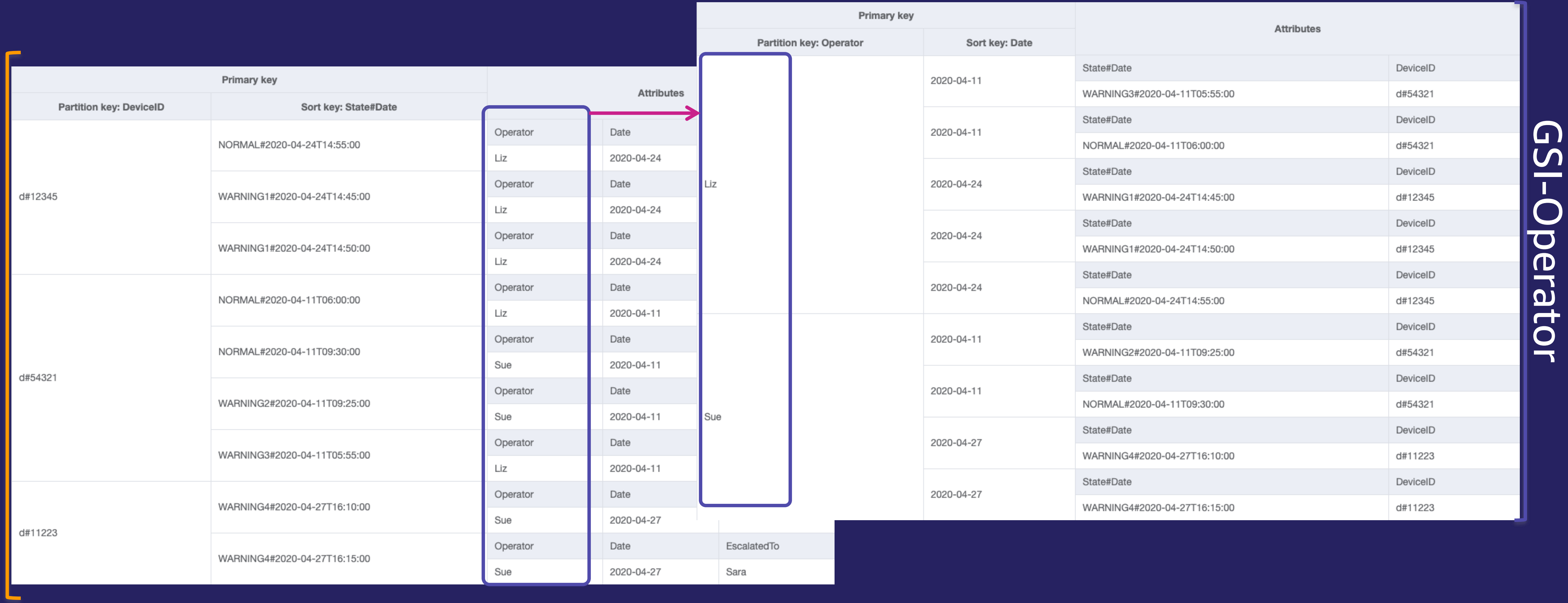
Use Composite Sort Key instead



Primary key	
Partition key: DeviceID	Sort key: State#Date
d#12345	NORMAL#2020-04-24T14:55:00
	WARNING1#2020-04-24T14:40:00
	WARNING1#2020-04-24T14:45:00
	WARNING1#2020-04-24T14:50:00
d#54321	NORMAL#2020-04-11T06:00:00
	NORMAL#2020-04-11T09:30:00
	WARNING2#2020-04-11T09:25:00
	WARNING3#2020-04-11T05:50:00
	WARNING3#2020-04-11T05:55:00

```
aws dynamodb query
--table-name DeviceLog
--no-scan-index-forward
--key-condition-expression ”#dID = :dID  AND begins_with(#s, :sd)”
--expression-attribute-names ‘{“#cId": ”DeviceID”, “#s”: “State#Date”}’
--expression-attribute-values ‘{“:cId”: {“S”:”d#12345”}, “:sd”: {“S”:”WARNING1#”}}’
```

Access Pattern: Fetch all device logs for a given operator between two dates



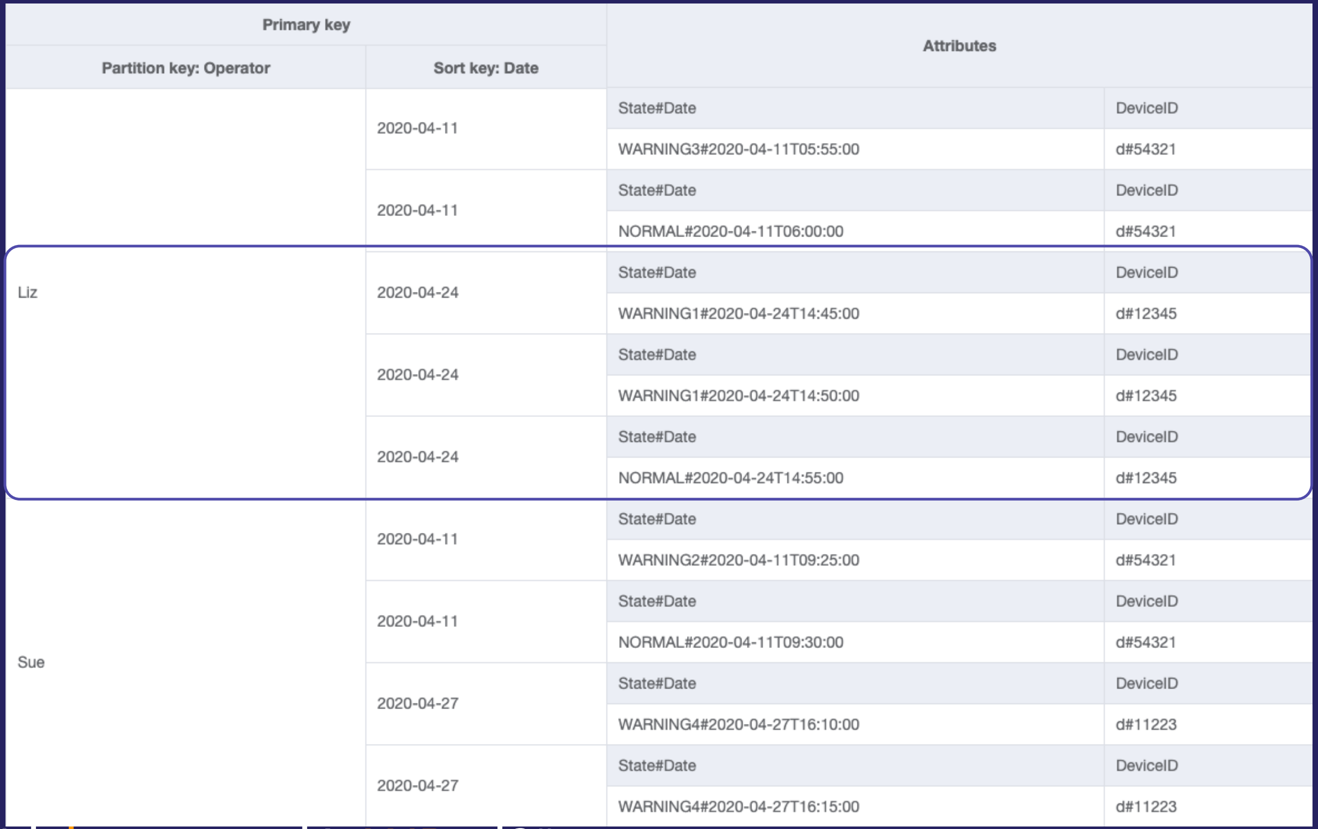
Base Table

Primary key		Attributes		
Partition key: DeviceID	Sort key: State#Date			
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date	
		Liz	2020-04-24	
	WARNING1#2020-04-24T14:45:00	Operator	Date	
		Liz	2020-04-24	
	WARNING1#2020-04-24T14:50:00	Operator	Date	
		Liz	2020-04-24	
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date	
		Liz	2020-04-11	
	NORMAL#2020-04-11T09:30:00	Operator	Date	
		Sue	2020-04-11	
	WARNING2#2020-04-11T09:25:00	Operator	Date	
		Sue	2020-04-11	
	WARNING3#2020-04-11T05:55:00	Operator	Date	
		Liz	2020-04-11	
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date	
		Sue	2020-04-27	
	WARNING4#2020-04-27T16:15:00	Operator	Date	EscalatedTo
		Sue	2020-04-27	Sara

GSI-Operator

Primary key		Attributes	
Partition key: Operator	Sort key: Date		
Liz	2020-04-11	State#Date	DeviceID
		WARNING3#2020-04-11T05:55:00	d#54321
	2020-04-11	State#Date	DeviceID
		NORMAL#2020-04-11T06:00:00	d#54321
	2020-04-24	State#Date	DeviceID
		WARNING1#2020-04-24T14:45:00	d#12345
	2020-04-24	State#Date	DeviceID
		WARNING1#2020-04-24T14:50:00	d#12345
	2020-04-24	State#Date	DeviceID
		NORMAL#2020-04-24T14:55:00	d#12345
Sue	2020-04-11	State#Date	DeviceID
		WARNING2#2020-04-11T09:25:00	d#54321
	2020-04-11	State#Date	DeviceID
		NORMAL#2020-04-11T09:30:00	d#54321
	2020-04-27	State#Date	DeviceID
		WARNING4#2020-04-27T16:10:00	d#11223
	2020-04-27	State#Date	DeviceID
		WARNING4#2020-04-27T16:15:00	d#11223

Access Pattern: Fetch all device logs for a given operator between two dates

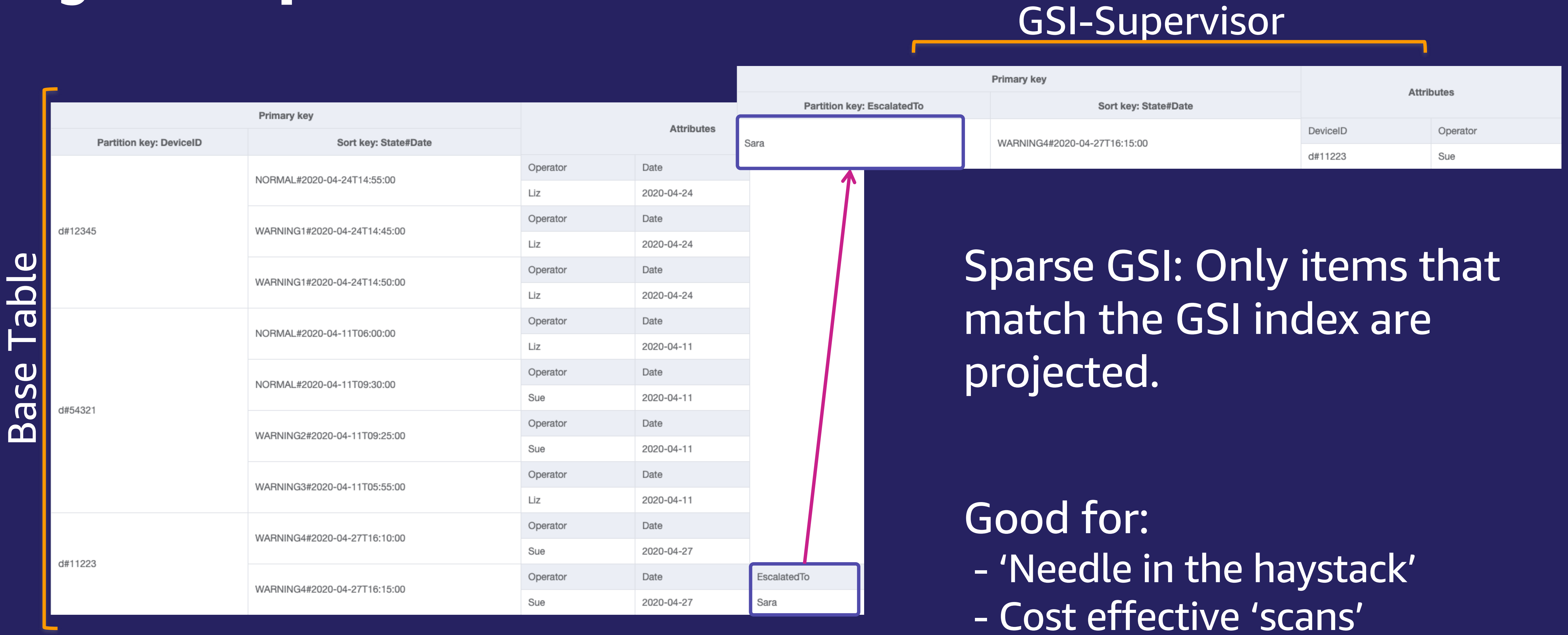


Primary key		Attributes	
Partition key: Operator	Sort key: Date		
Liz	2020-04-11	State#Date	DeviceID
		WARNING3#2020-04-11T05:55:00	d#54321
	2020-04-11	State#Date	DeviceID
		NORMAL#2020-04-11T06:00:00	d#54321
	2020-04-24	State#Date	DeviceID
		WARNING1#2020-04-24T14:45:00	d#12345
	2020-04-24	State#Date	DeviceID
		WARNING1#2020-04-24T14:50:00	d#12345
	2020-04-24	State#Date	DeviceID
		NORMAL#2020-04-24T14:55:00	d#12345
Sue	2020-04-11	State#Date	DeviceID
		WARNING2#2020-04-11T09:25:00	d#54321
	2020-04-11	State#Date	DeviceID
		NORMAL#2020-04-11T09:30:00	d#54321
	2020-04-27	State#Date	DeviceID
		WARNING4#2020-04-27T16:10:00	d#11223
	2020-04-27	State#Date	DeviceID
		WARNING4#2020-04-27T16:15:00	d#11223

GSI – Operator

```
aws dynamodb query
--table-name DeviceLog
--index-name GSI-Operator
--key-condition-expression ”#op = :op AND #d  between :d1 AND :d2”
--expression-attribute-names ‘{“#op": ”Operator” , “#d”: “Date”}’
--expression-attribute-values ‘{“:op”: {“S”:”Liz”} , “:d1”: {“S”:”2020-04-20”}, “:d2”:{“S”:”2020-04-25”}}’
```

Access Pattern: Fetch all escalated device logs for a given supervisor



GSI-Supervisor

Primary key		Attributes	
Partition key: EscalatedTo	Sort key: State#Date	DeviceID	Operator
Sara	WARNING4#2020-04-27T16:15:00	d#11223	Sue

Base Table

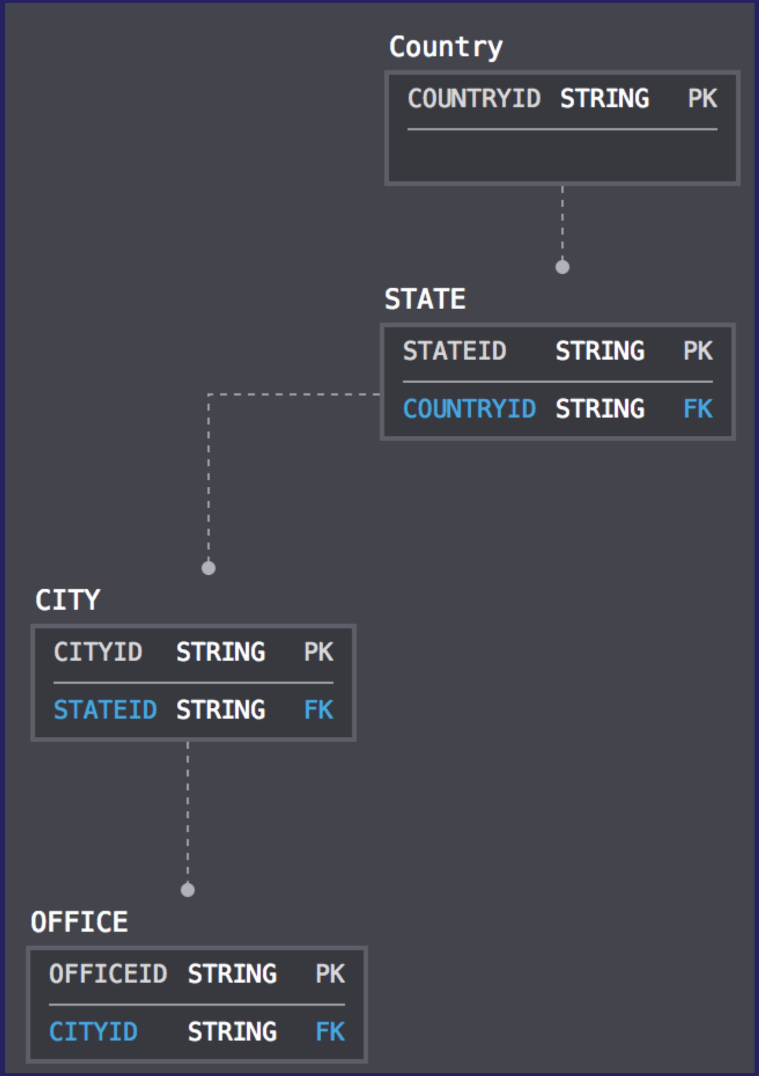
Primary key		Attributes		
Partition key: DeviceID	Sort key: State#Date			
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date	
		Liz	2020-04-24	
	WARNING1#2020-04-24T14:45:00	Operator	Date	
		Liz	2020-04-24	
	WARNING1#2020-04-24T14:50:00	Operator	Date	
		Liz	2020-04-24	
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date	
		Liz	2020-04-11	
	NORMAL#2020-04-11T09:30:00	Operator	Date	
		Sue	2020-04-11	
	WARNING2#2020-04-11T09:25:00	Operator	Date	
		Sue	2020-04-11	
	WARNING3#2020-04-11T05:55:00	Operator	Date	
		Liz	2020-04-11	
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date	
		Sue	2020-04-27	
	WARNING4#2020-04-27T16:15:00	Operator	Date	EscalatedTo
		Sue	2020-04-27	Sara

Sparse GSI: Only items that match the GSI index are projected.

Good for:

- ‘Needle in the haystack’
- Cost effective ‘scans’
- Item management

Example – Phone Tool: Hierarchical data in the Sort Key



Country

COUNTRYID STRING PK

STATE

STATEID STRING PK

COUNTRYID STRING FK

CITY

CITYID STRING PK

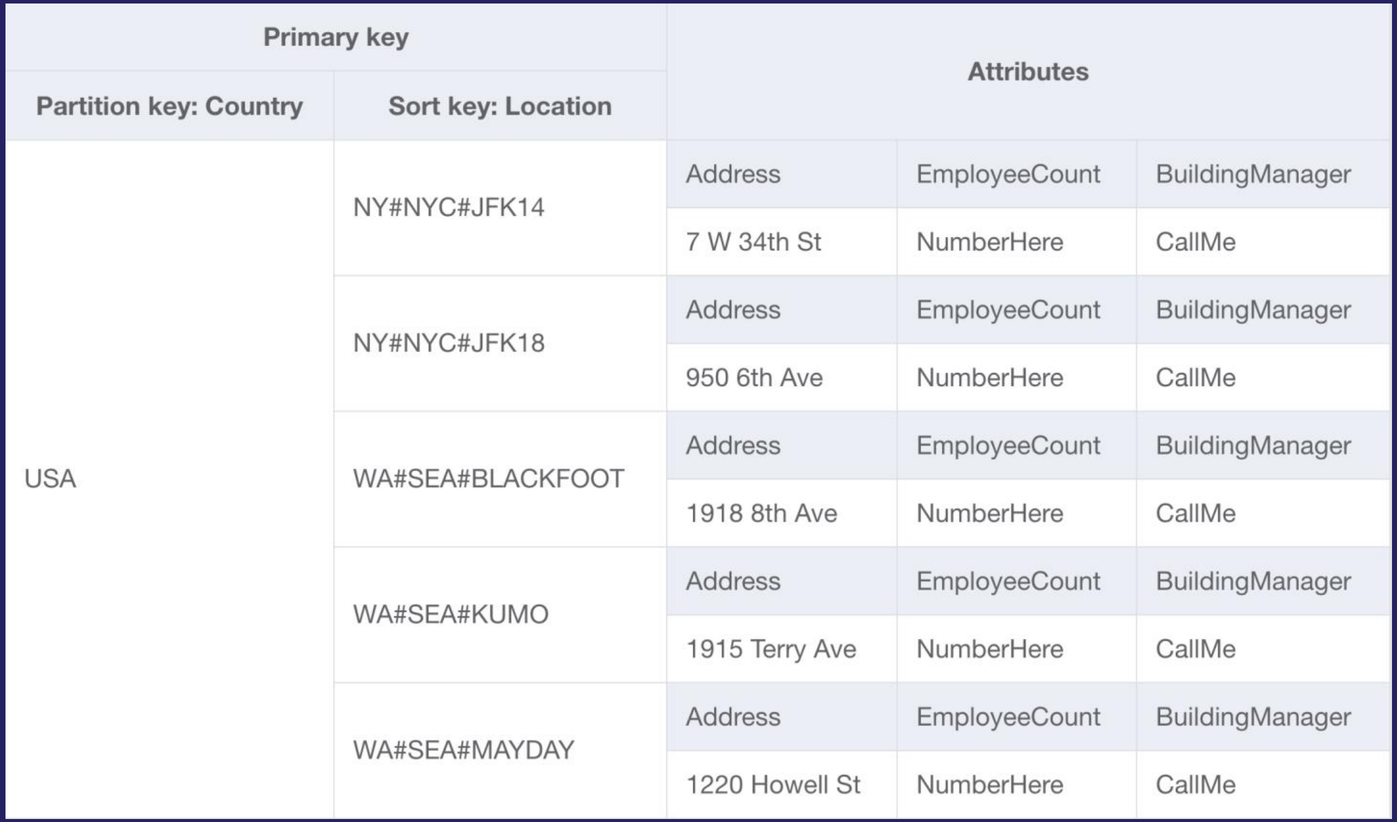
STATEID STRING FK

OFFICE

OFFICEID STRING PK

CITYID STRING FK

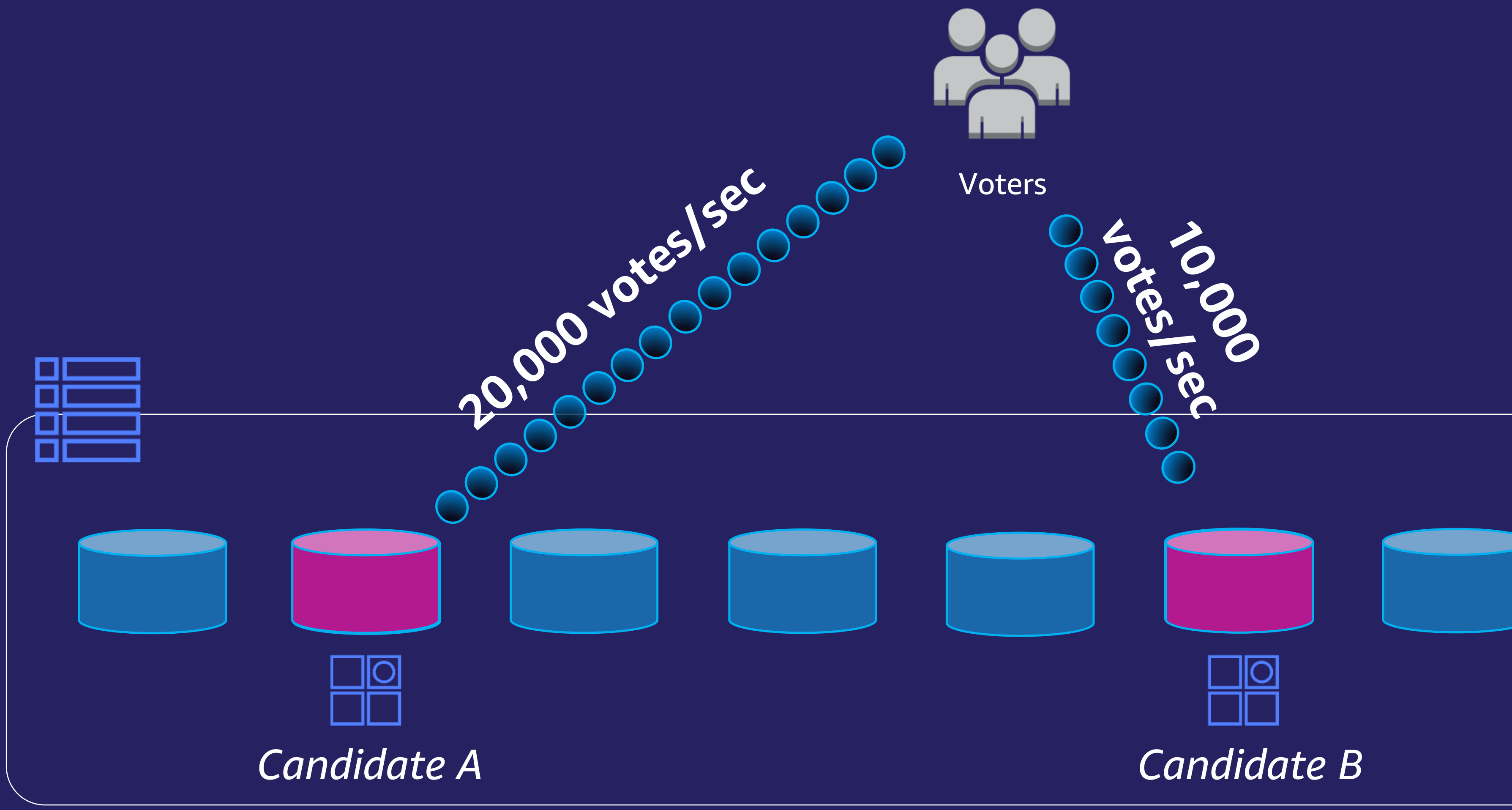
- Use composite sort key to define a hierarchy
- Highly selective queries with sort conditions
- Reduce query complexity



Primary key		Attributes		
Partition key: Country	Sort key: Location			
USA	NY#NYC#JFK14	Address	EmployeeCount	BuildingManager
		7 W 34th St	NumberHere	CallMe
	NY#NYC#JFK18	Address	EmployeeCount	BuildingManager
		950 6th Ave	NumberHere	CallMe
	WA#SEA#BLACKFOOT	Address	EmployeeCount	BuildingManager
		1918 8th Ave	NumberHere	CallMe
	WA#SEA#KUMO	Address	EmployeeCount	BuildingManager
		1915 Terry Ave	NumberHere	CallMe
	WA#SEA#MAYDAY	Address	EmployeeCount	BuildingManager
		1220 Howell St	NumberHere	CallMe

Advanced Scenarios

Example – Voting: Scaling high write throughput & low cardinality…



Votes Table

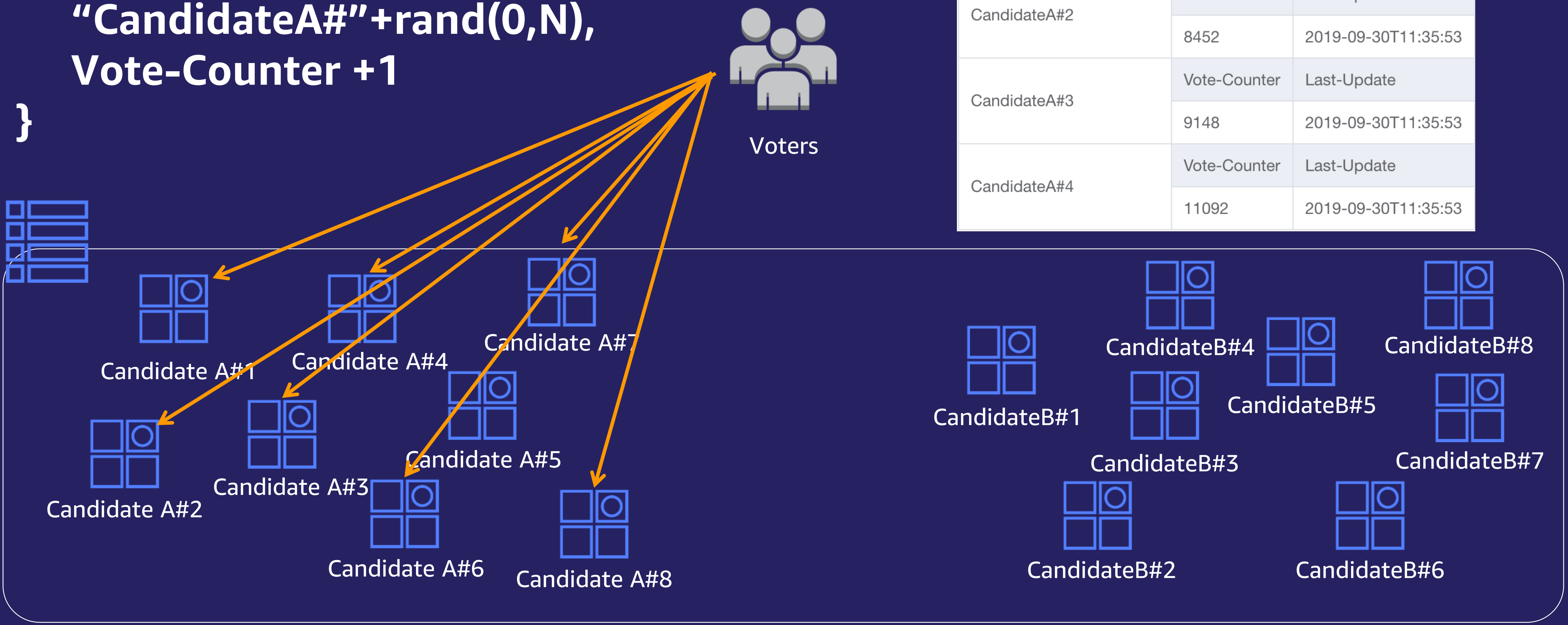
Primary Key	Attributes
Partition Key: Candidate	
CandidateA	Vote-Counter
	21943
CandidateB	Vote-Counter
	10203

Single Item limit: 1000 WCUs

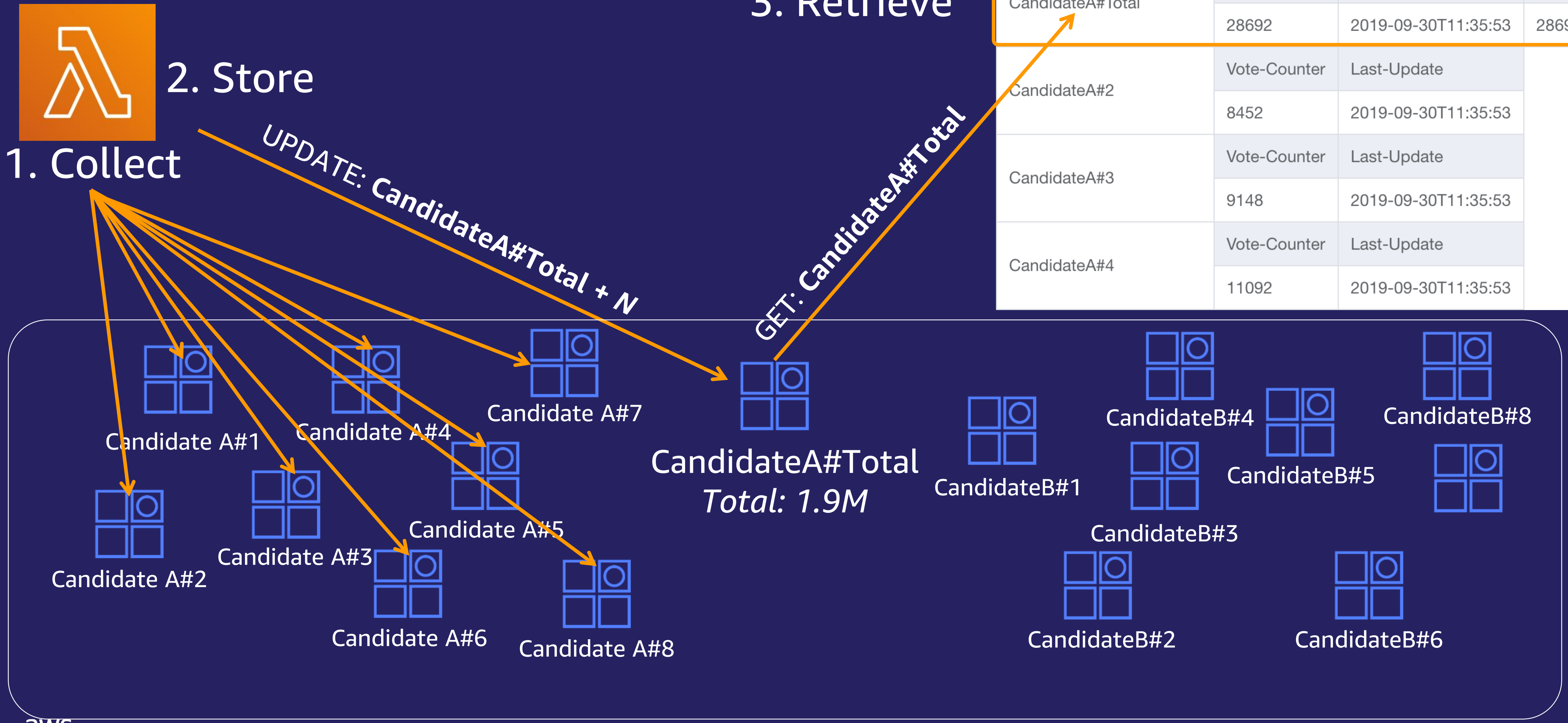
Write shard the partition key…

```
UPDATE Item :
{
    “CandidateA#”+rand(0,N),
    Vote-Counter +1
}
```

Primary Key Partition Key: Candidate	Attributes	
CandidateA#1	Vote-Counter	Last-Update
	10238	2019-09-30T11:35:53
CandidateA#2	Vote-Counter	Last-Update
	8452	2019-09-30T11:35:53
CandidateA#3	Vote-Counter	Last-Update
	9148	2019-09-30T11:35:53
CandidateA#4	Vote-Counter	Last-Update
	11092	2019-09-30T11:35:53



Retrieve result: *(in parallel)*



1. Collect

2. Store

3. Retrieve

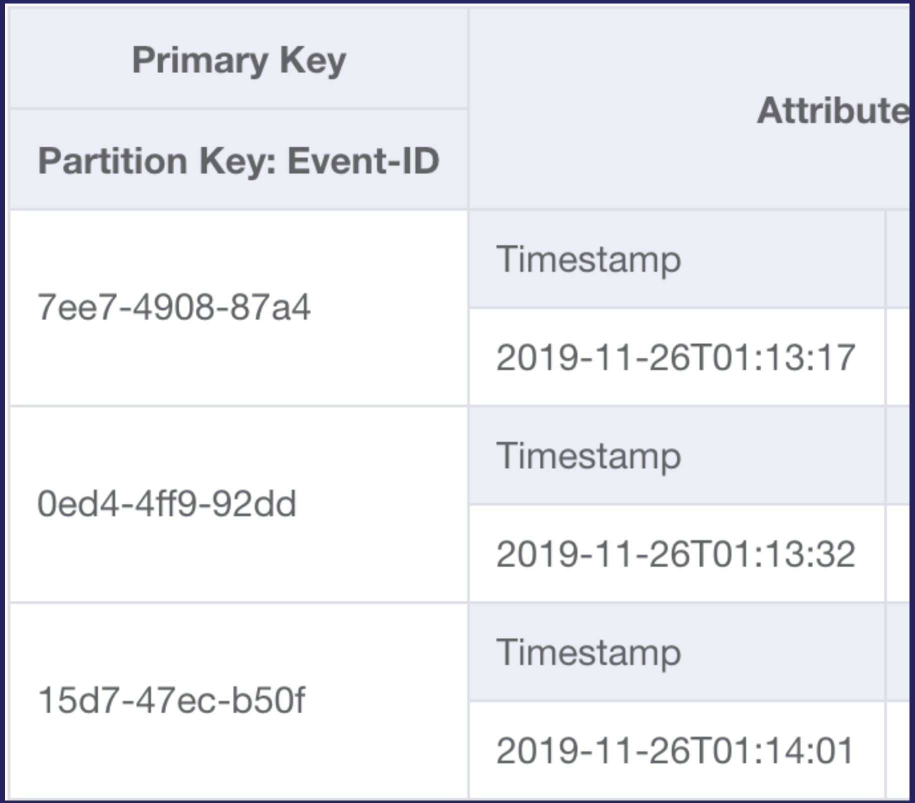
UPDATE: **CandidateA#Total + N**

GET: **CandidateA#Total**

Primary Key Partition Key: Candidate	Attributes		
CandidateA#Total	Vote-Counter	Last-Update	Total
	28692	2019-09-30T11:35:53	28692
CandidateA#2	Vote-Counter	Last-Update	
	8452	2019-09-30T11:35:53	
CandidateA#3	Vote-Counter	Last-Update	
	9148	2019-09-30T11:35:53	
CandidateA#4	Vote-Counter	Last-Update	
	11092	2019-09-30T11:35:53	

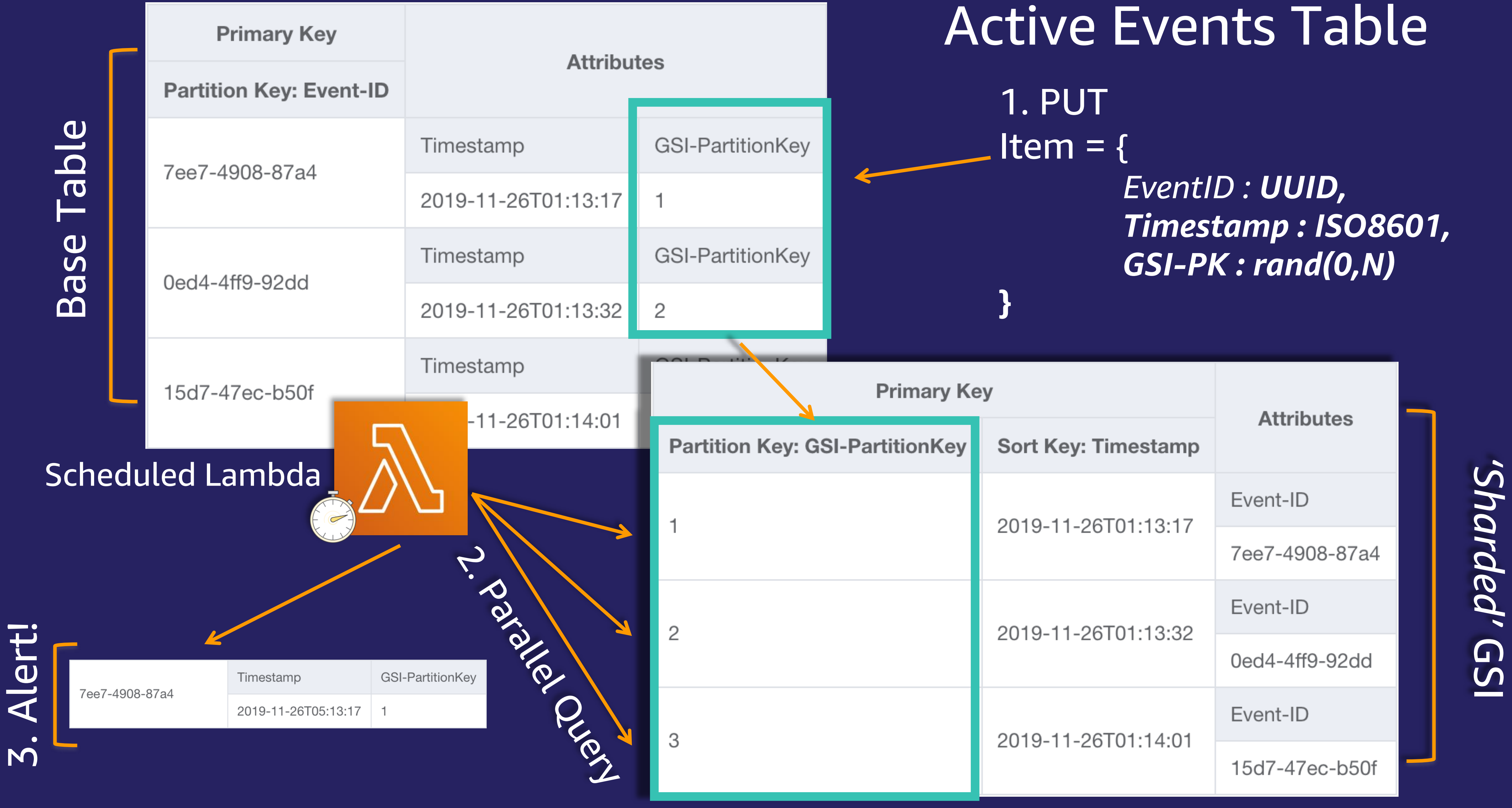
Example – Event Tracking

Access Pattern: Fetch all the events that are older than 4 hours



Primary Key	Attribute
Partition Key: Event-ID	
7ee7-4908-87a4	Timestamp
	2019-11-26T01:13:17
0ed4-4ff9-92dd	Timestamp
	2019-11-26T01:13:32
15d7-47ec-b50f	Timestamp
	2019-11-26T01:14:01

Active Events Table



Base Table

Primary Key	Attributes	
Partition Key: Event-ID		
7ee7-4908-87a4	Timestamp	GSI-PartitionKey
	2019-11-26T01:13:17	1
0ed4-4ff9-92dd	Timestamp	GSI-PartitionKey
	2019-11-26T01:13:32	2
15d7-47ec-b50f	Timestamp	
	-11-26T01:14:01	

1. PUT
Item = {
*EventID : **UUID,***
Timestamp : ISO8601,
GSI-PK : rand(0,N)
}

'Sharded' GSI

Primary Key		Attributes
Partition Key: GSI-PartitionKey	Sort Key: Timestamp	
1	2019-11-26T01:13:17	Event-ID
		7ee7-4908-87a4
2	2019-11-26T01:13:32	Event-ID
		0ed4-4ff9-92dd
3	2019-11-26T01:14:01	Event-ID
		15d7-47ec-b50f

Scheduled Lambda

2. Parallel Query

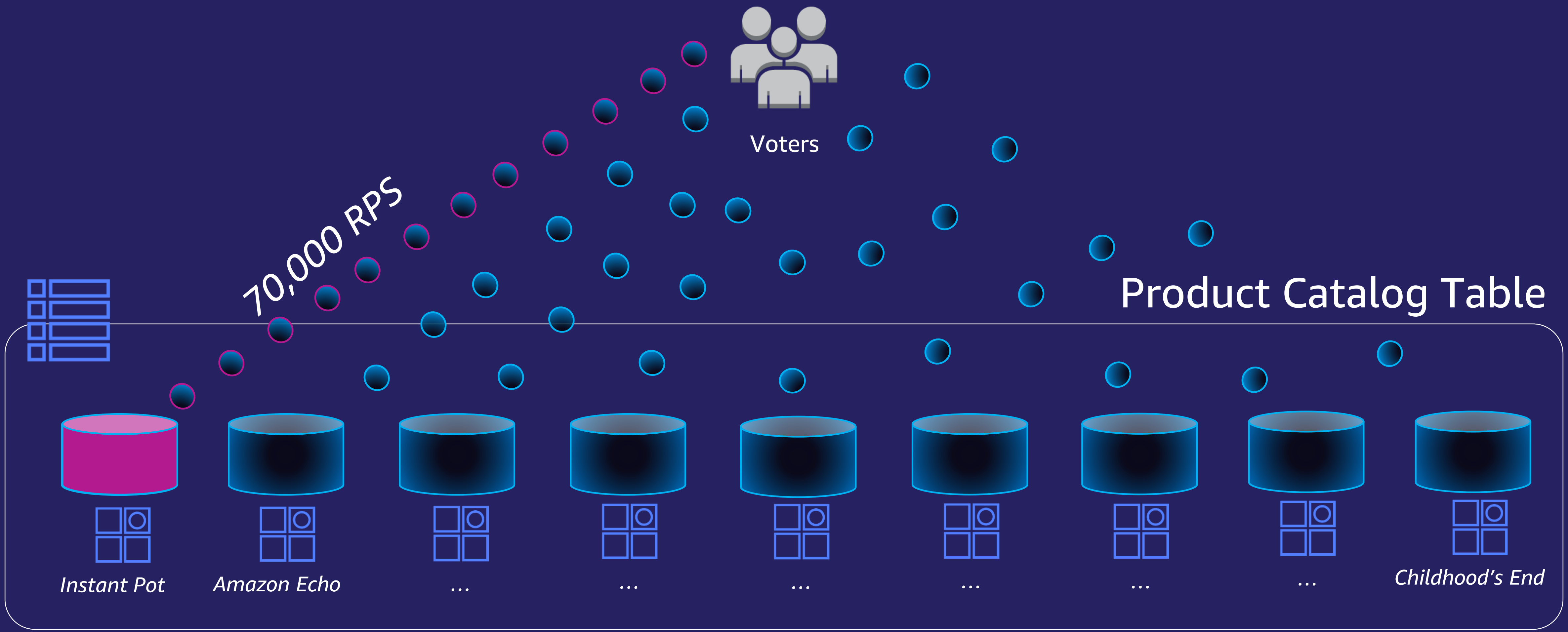
3. Alert!

7ee7-4908-87a4	Timestamp	GSI-PartitionKey
	2019-11-26T05:13:17	1

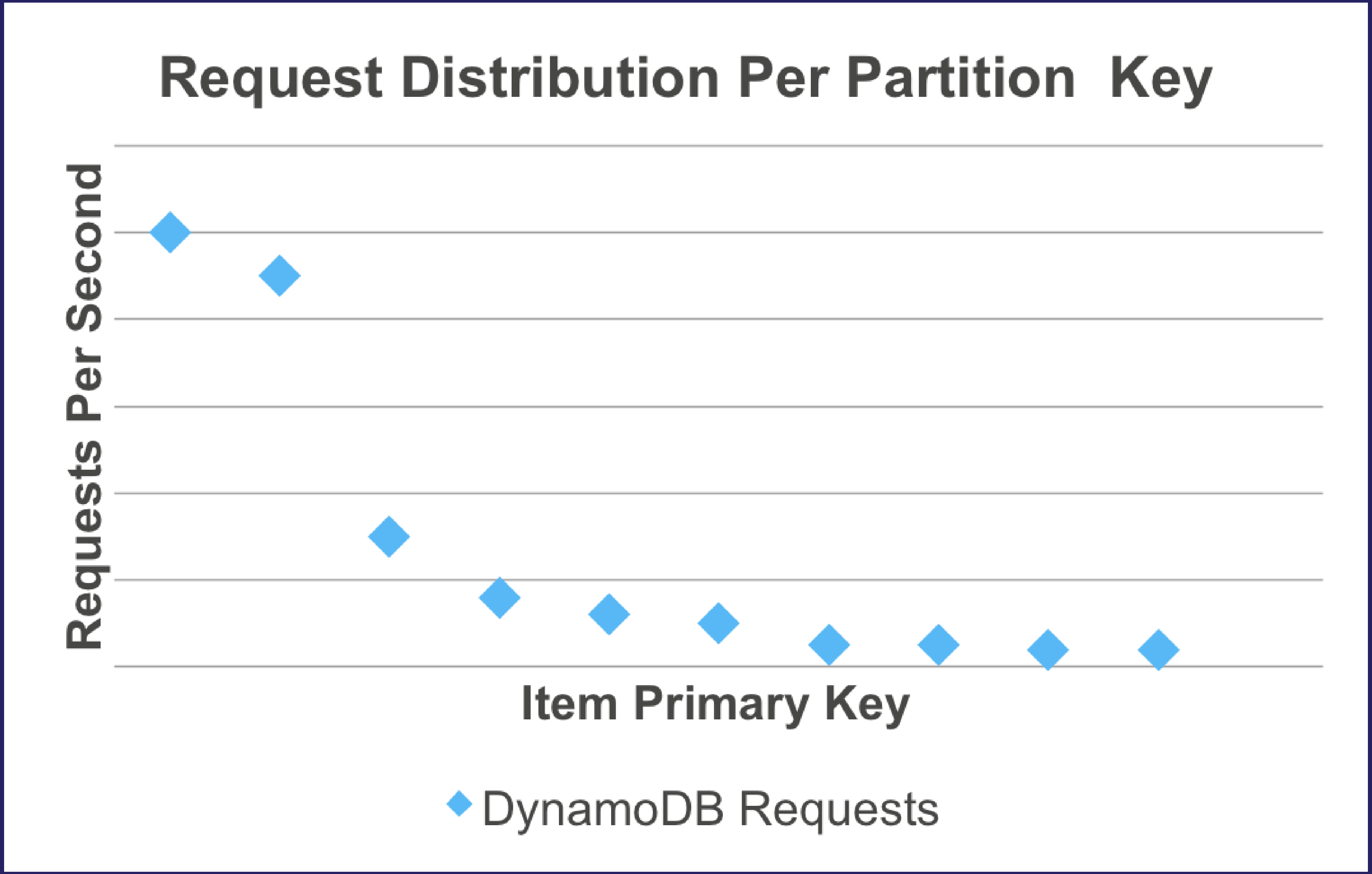
Selectively query the entire table..

Example – Product Catalog

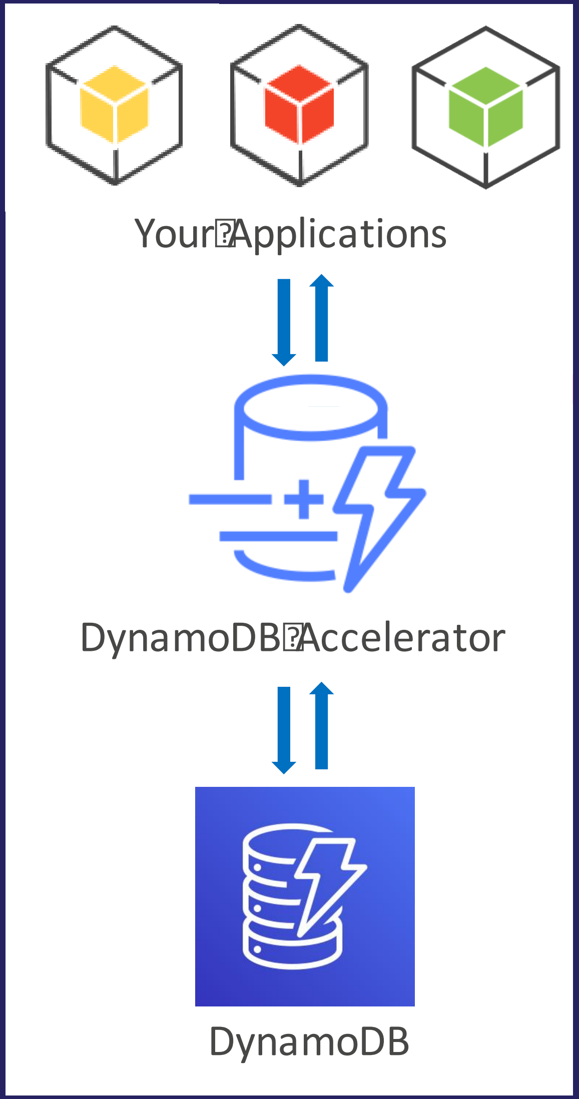
Read distribution imbalance: “popular items”



Total throughput per Item = 3000 RCUs

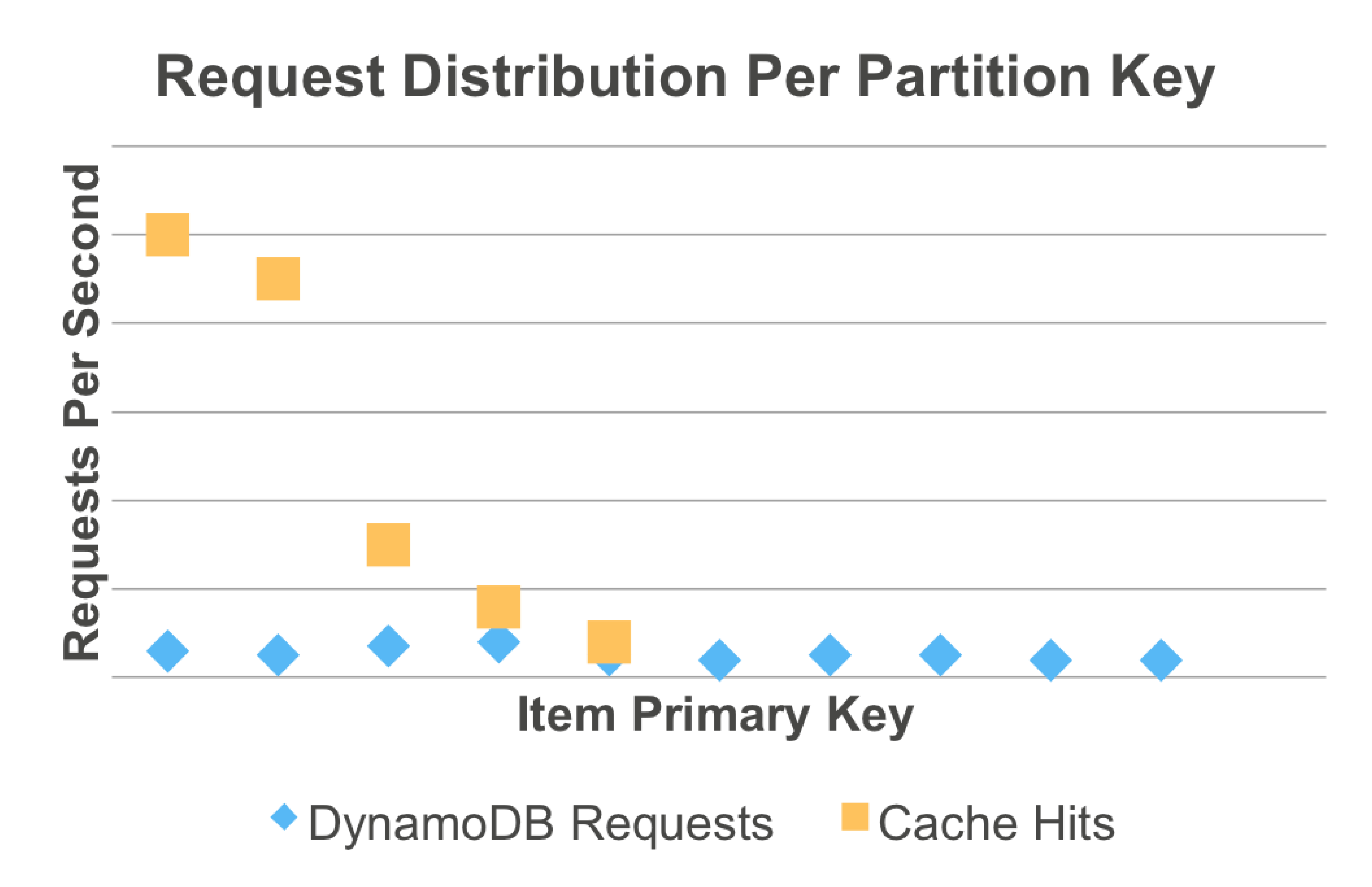

Request Distribution Per Partition Key
Requests Per Second
Item Primary Key
◆ DynamoDB Requests

DynamoDB Accelerator (DAX)



- **Fully managed, highly available:** handles all software management, fault tolerant, replication across multi-AZs within a region
- **DynamoDB API compatible:** seamlessly caches DynamoDB API calls, no application re-writes required
- **Write-through:** DAX handles caching for writes
- **Flexible**: configure DAX for one table or many
- **Scalable**: scales-out to any workload with up to 10 read replicas
- **Manageability:** fully integrated AWS service: Amazon CloudWatch, tagging for DynamoDB, AWS Console
- **Security:** Amazon VPC, AWS IAM, AWS CloudTrail, AWS Organizations

After using DAX



DynamoDB Transactions API

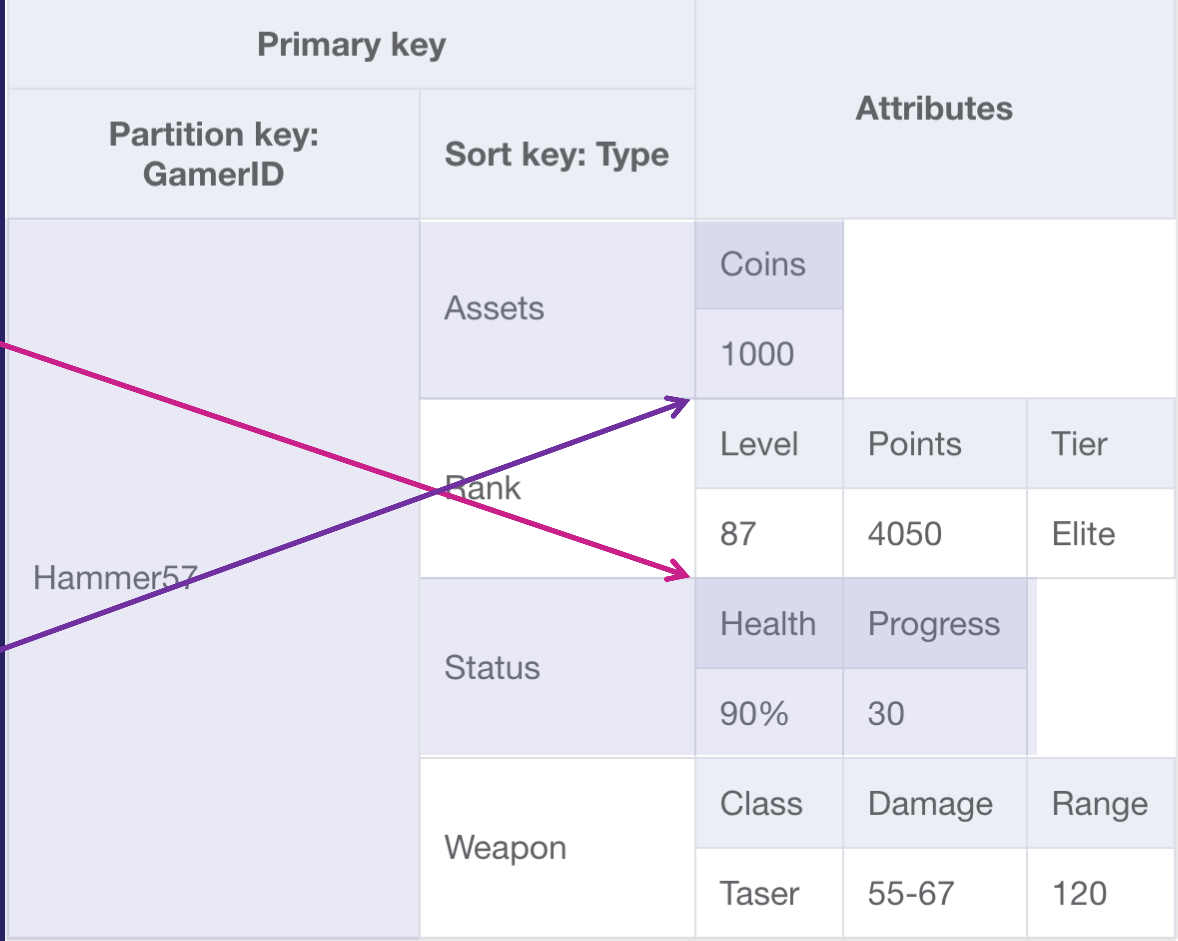
- TransactWriteItems
  - Synchronous and atomic update, put, delete, and check
  - Up to 25 items within a transaction
  - Supports multiple tables
  - Complex conditional checks
  - Uses 2x the WCU

- Good Use Cases
  - Commit changes across items
  - Conditional batch inserts/updates

Example – Game state: Transactions API

Atomic update of gamer “Hammer57’s” Health & Coins

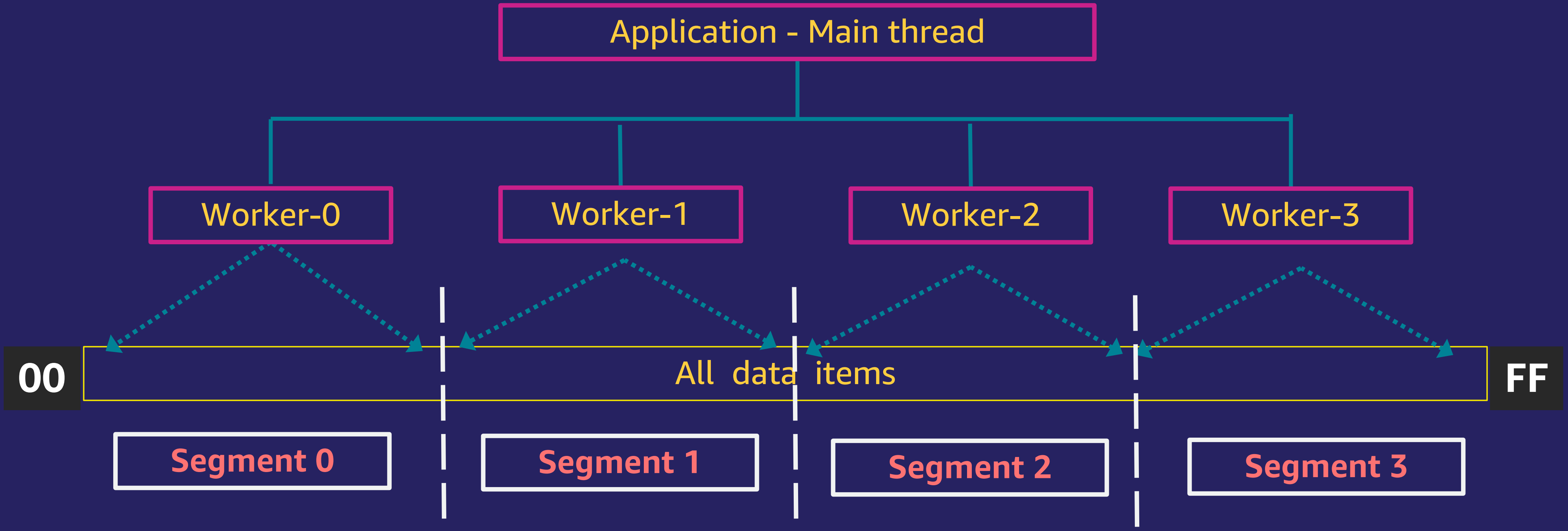
```
{ "TransactItems" :  [ {
   "Update ": {
    "TableName": "Gamers",
    "Key" :{"GamerID" : {"S": "Hammer57"},
    "Type" : {"S" : "Status"}},
    "UpdateExpression" : "Set health = :nhealth",
    "ExpressionAttributeValues":{":nhealth":{“N”:“100”}}
   },
  {
   "Update ": {
    "TableName": "Gamers",
    "Key" :{"GamerID" : {"S": "Hammer57"},
    "Type" : {"S" : "Assets"} },
      "ConditionExpression" : "coins > :cost",
      "UpdateExpression" : "Set coins = coins - :cost",
      "ExpressionAttributeValues" :{":cost":{“N”:“400”}}
   }
 }]
}
```



Primary key		Attributes		
Partition key: GamerID	Sort key: Type			
Hammer57	Assets	Coins		
		1000		
	Rank	Level	Points	Tier
		87	4050	Elite
	Status	Health	Progress	
		90%	30	
	Weapon	Class	Damage	Range
		Taser	55-67	120

Parallel Scan

- Need to read all the items from a table as quickly as possible?
- Set `TotalSegments` = number of application workers; each worker scans a different *segment*



Sequential versus Parallel Scan

Scenario: Scan server logs data for response code <> 200 (OK)

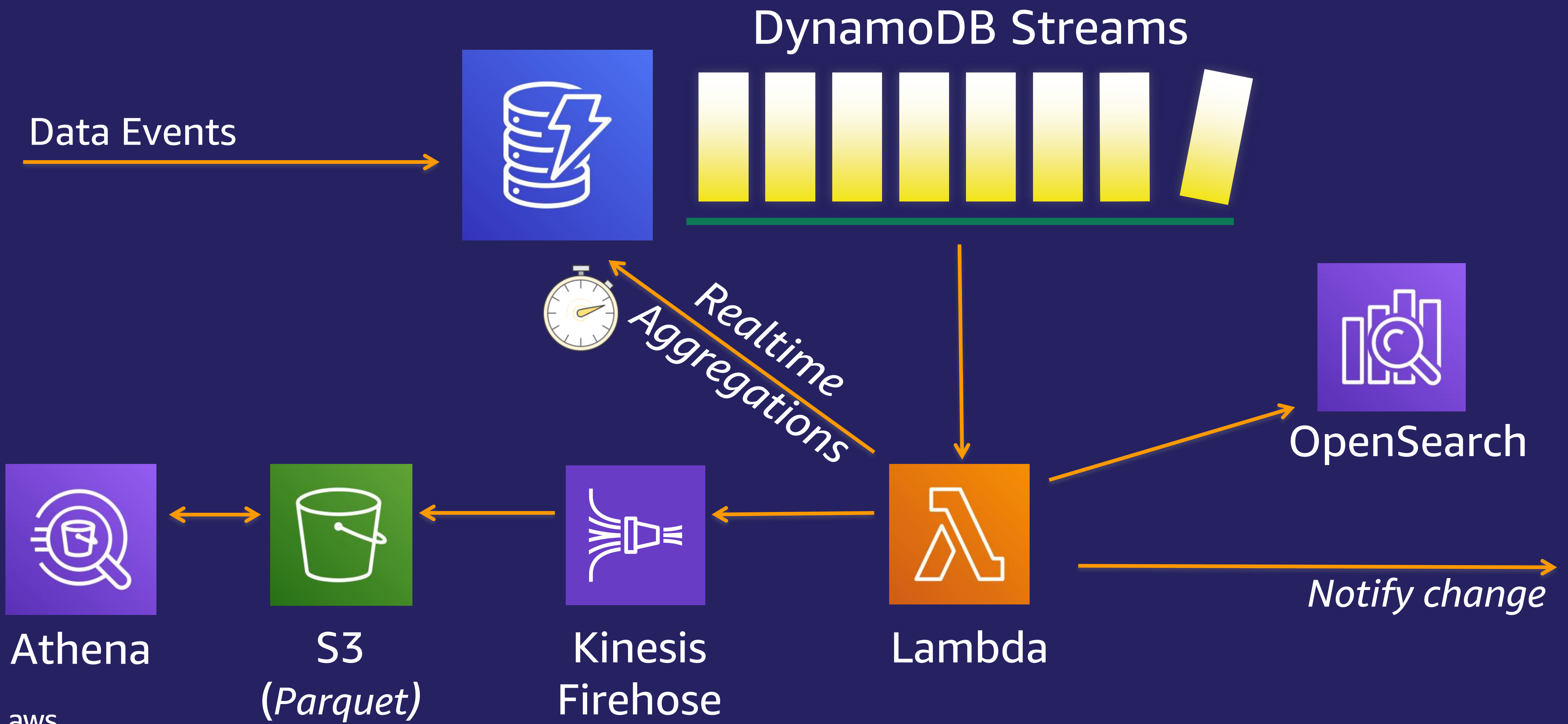
Sequential Scan

```
fe = "responsecode <> :f"
eav = {":f": 200}
response = table.scan(
    FilterExpression=fe,
    ExpressionAttributeValues=eav,
    Limit=pageSize
)
```

Parallel Scan

```
fe = "responsecode <> :f"
eav = {":f": 200}
response = table.scan(
    FilterExpression=fe,
    ExpressionAttributeValues=eav,
    Limit=pageSize,
     TotalSegments=totalsegments,
     Segment=threadsegment
    )
```

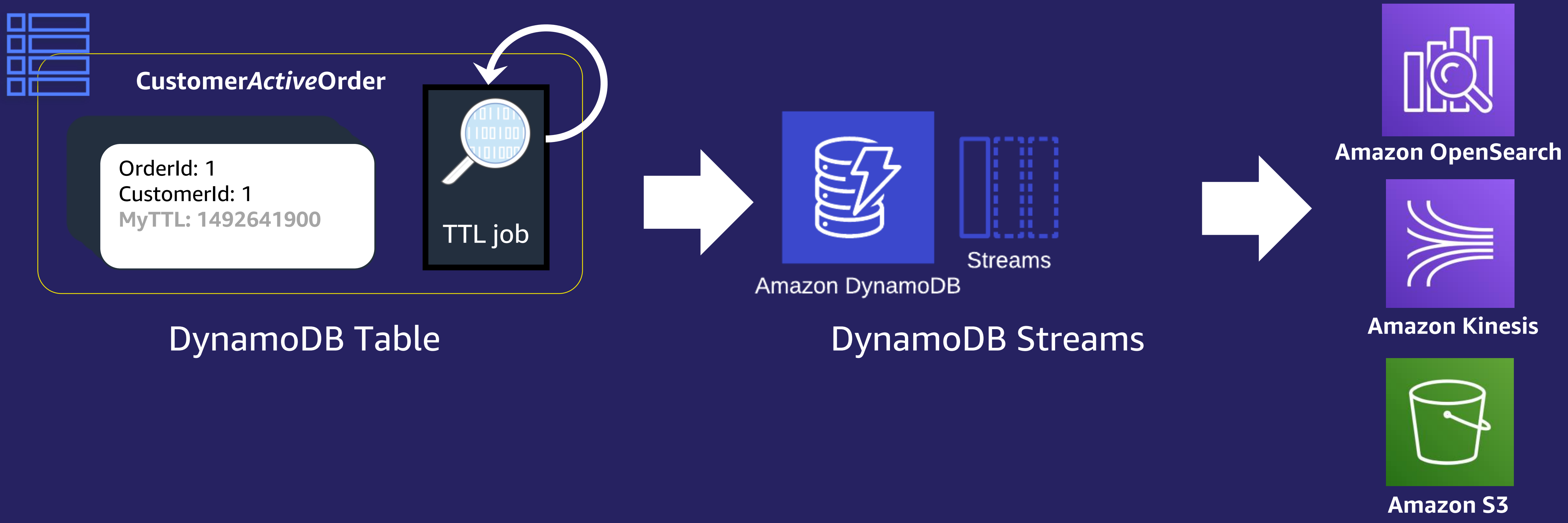
Serverless and Event-Driven Architecture



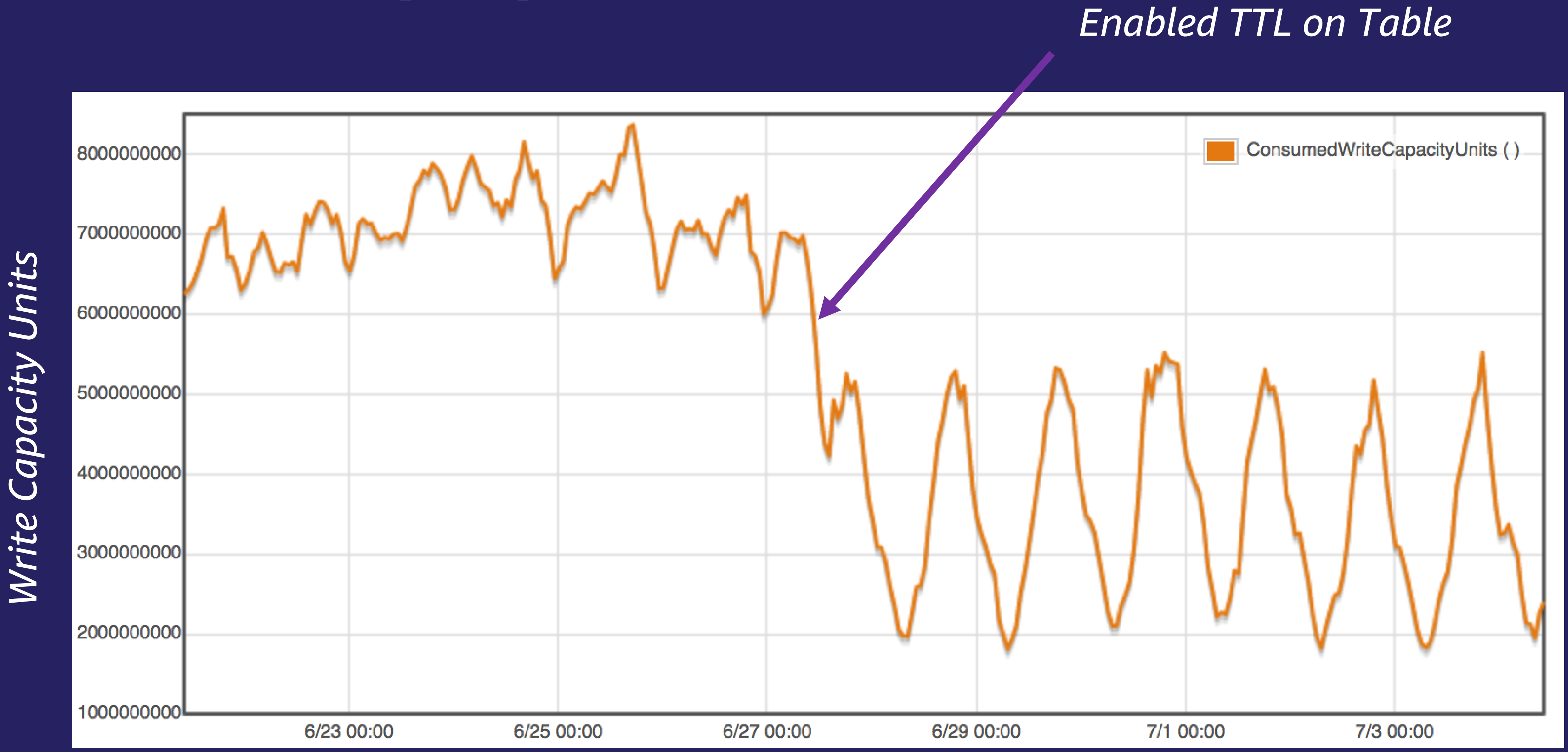
Time-To-Live (TTL) – Archive Design Pattern

Time-To-Live

An epoch timestamp marking when an item can be deleted by a background process, without consuming any provisioned capacity



Time-To-Live (TTL)



Feature Highlights

DynamoDB feature highlights

99.999% SLA

DynamoDB Accelerator (DAX)

Global tables

DynamoDB Streams and Kinesis Data Streams support

Auto scaling

Adaptive capacity

Time To Live (TTL)

NoSQL Workbench

Transactions

Encryption at rest

Point-in-time Recovery (PITR)

On-demand backup and restore

Export to Amazon S3

Amazon CloudWatch Contributor Insights for DynamoDB

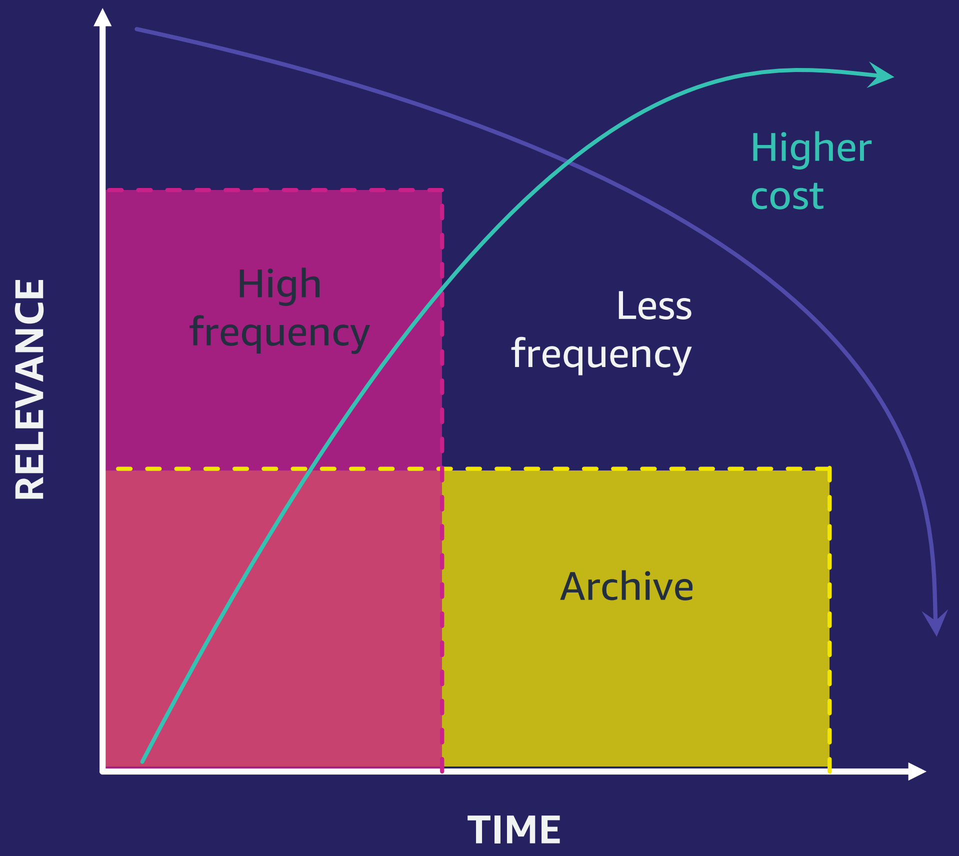
Audit logging with AWS CloudTrail

NEW

Amazon DynamoDB Standard-Infrequent Access

Reduce costs by up to 60%

Data lifecycle



- Data volume grows over time
- Data relevance decreases over time
- Older data gets less frequently accessed
- Storing data can be expensive at scale

Common use cases for infrequently accessed data

Social media

Active users expect older posts to be available whenever they want, immediately

Data analytics

Businesses need to capture and refine billions of data points to deliver the most accurate and actionable data analytics

Retail

Online shoppers sometimes want to look up their past orders, reorder the same item, or get product information anytime

NEW

Amazon DynamoDB Standard-Infrequent Access (Standard-IA) table class

Lower storage costs

The Standard-IA table class offers 60% lower storage costs than DynamoDB Standard tables.

No performance trade-offs

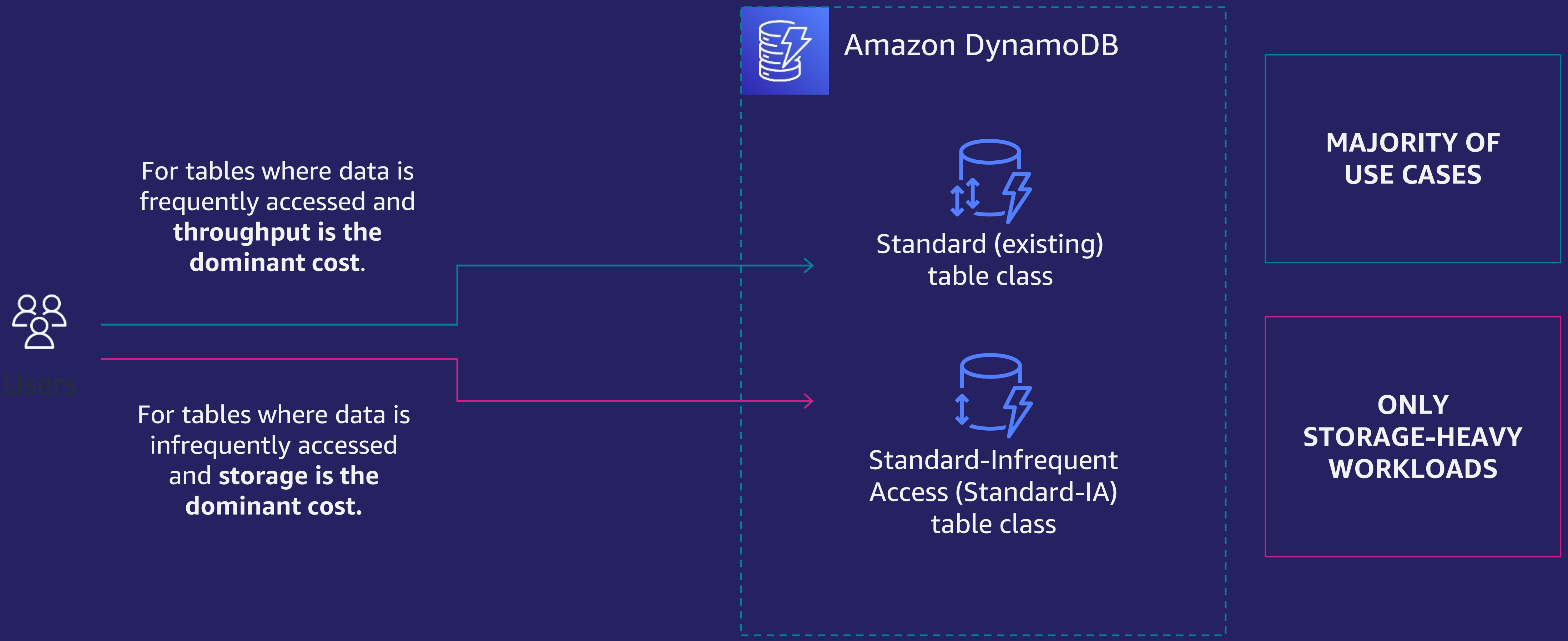
Standard-IA tables offer the same performance, durability, data availability, and massive scalability as existing DynamoDB Standard tables.

No developer overhead

Switch between table classes with a single click in the DynamoDB console, or using the AWS CLI or AWS SDK. Also, use the same DynamoDB APIs and service endpoints.

NEW

Flexibility to manage your data with a new table class



For tables where data is frequently accessed and **throughput is the dominant cost**.

Users

For tables where data is infrequently accessed and **storage is the dominant cost.**

Amazon DynamoDB

Standard (existing) table class

Standard-Infrequent Access (Standard-IA) table class

MAJORITY OF USE CASES

ONLY STORAGE-HEAVY WORKLOADS

Determine which table class is right for your use case

Analyze costs

Log in to the AWS Management Console and use AWS Cost and Usage Reports and AWS Cost Explorer to analyze your tables' cost structure

Storage cost ratio

When storage exceeds 50% of your throughput (reads and writes) cost, Amazon DynamoDB Standard-IA can help you reduce your table's cost mercado libre

"Amazon DynamoDB Standard-IA will provide us with the ability to store our users' infrequently accessed data at a significant cost savings, while continuing to deliver for our users by maintaining the same high performance, accessibility, and reliability we've come to expect from Amazon DynamoDB."

Oscar Mullin

Director of IT – Core Services SRE & DBA Head, Mercado Libre

You can use Standard-IA today

Amazon DynamoDB Standard-IA is the most cost-effective table class when storage represents the majority of a table's cost.

Get started with the DynamoDB Standard-IA table class today in the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS SDK.

Create a new DynamoDB table, or change existing tables to Standard-IA.

https://aws.amazon.com/dynamodb/standard-ia

Thank You!

Data Modernization Week: DynamoDB

Jason Hunter @dynamodb