AWS

SUMMIT

Deep Dive: Amazon DynamoDB

Richard Westby-Nunn – Technical Account Manager, Enterprise Support

28 June 2017

amazon web services

Agenda

DynamoDB 101 - Tables, Partitioning

Indexes

Scaling

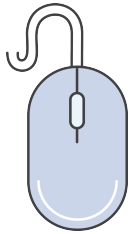
Data Modeling

DynamoDB Streams

Scenarios and Best Practices

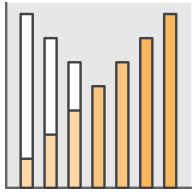
Recent Announcements

Amazon DynamoDB

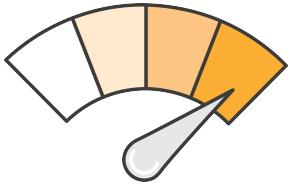


Fully Managed NoSQL

Document or Key-Value

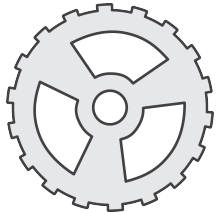


Scales to Any Workload



Fast and Consistent

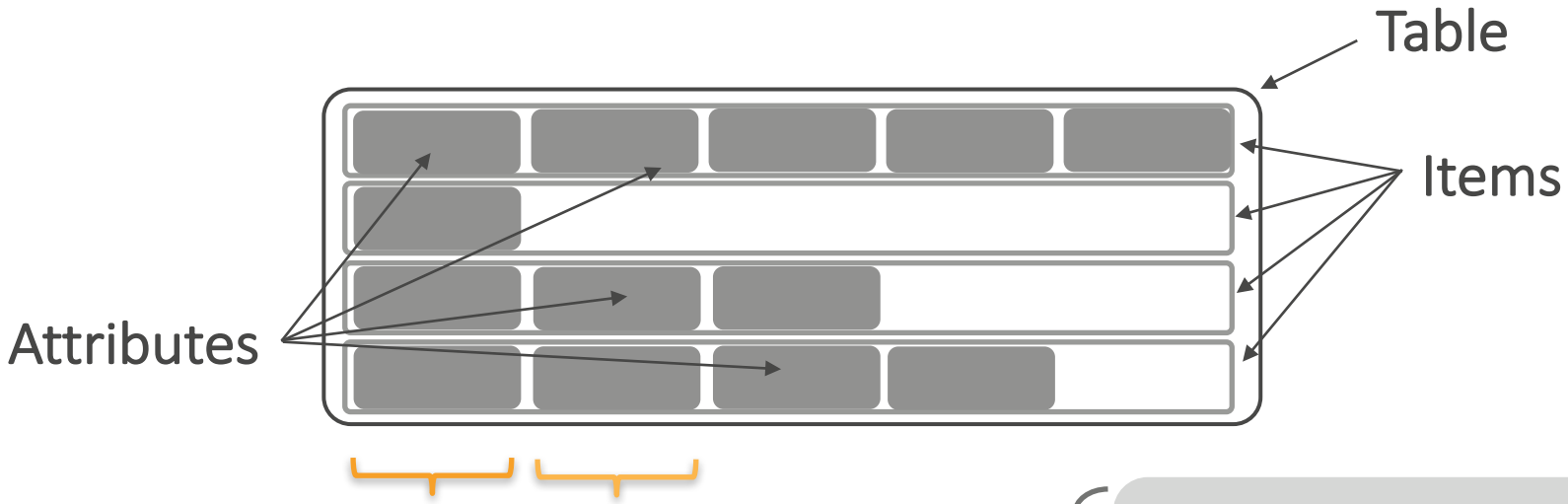
Access Control



Event Driven Programming

Tables, Partitioning

Table



Mandatory
Key-value access pattern
Determines data distribution

Optional
Model 1:N relationships
Enables rich query capabilities

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

Partition table

Partition key uniquely identifies an item
Partition key is used for building an unordered hash index
Table can be partitioned for scale

Id = 1
Name = Jim

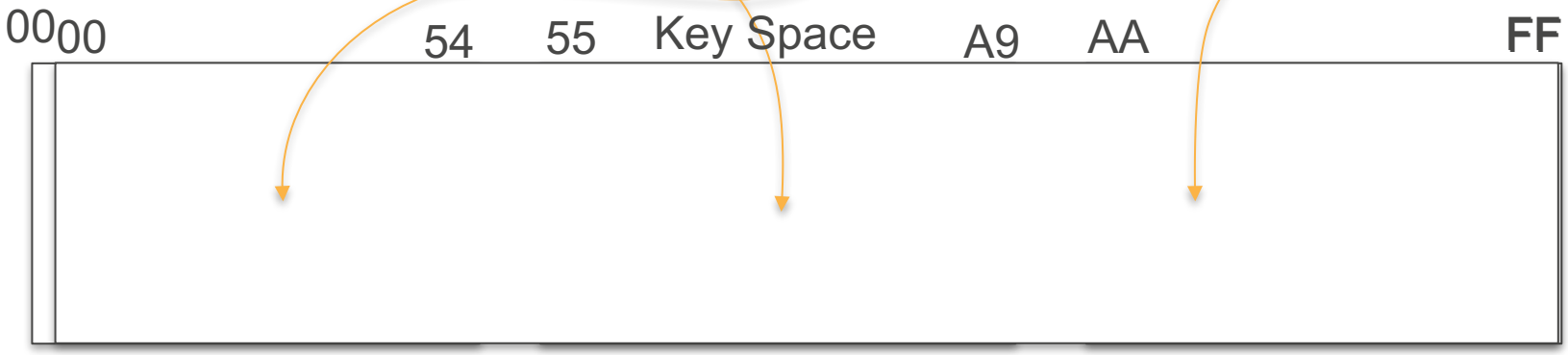
Hash (1) = 7B

Id = 2
Name = Andy
Dept = Engg

Hash (2) = 48

Id = 3
Name = Kim
Dept = Ops

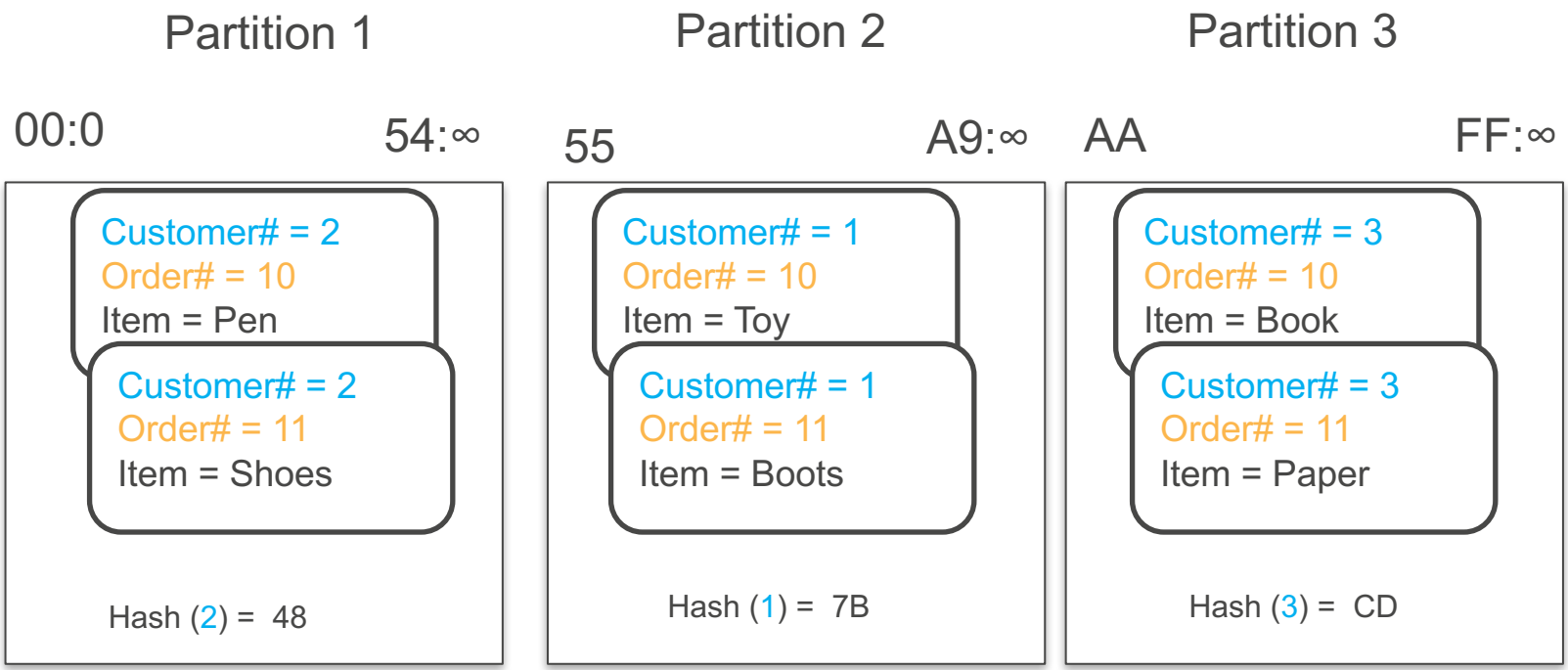
Hash (3) = CD



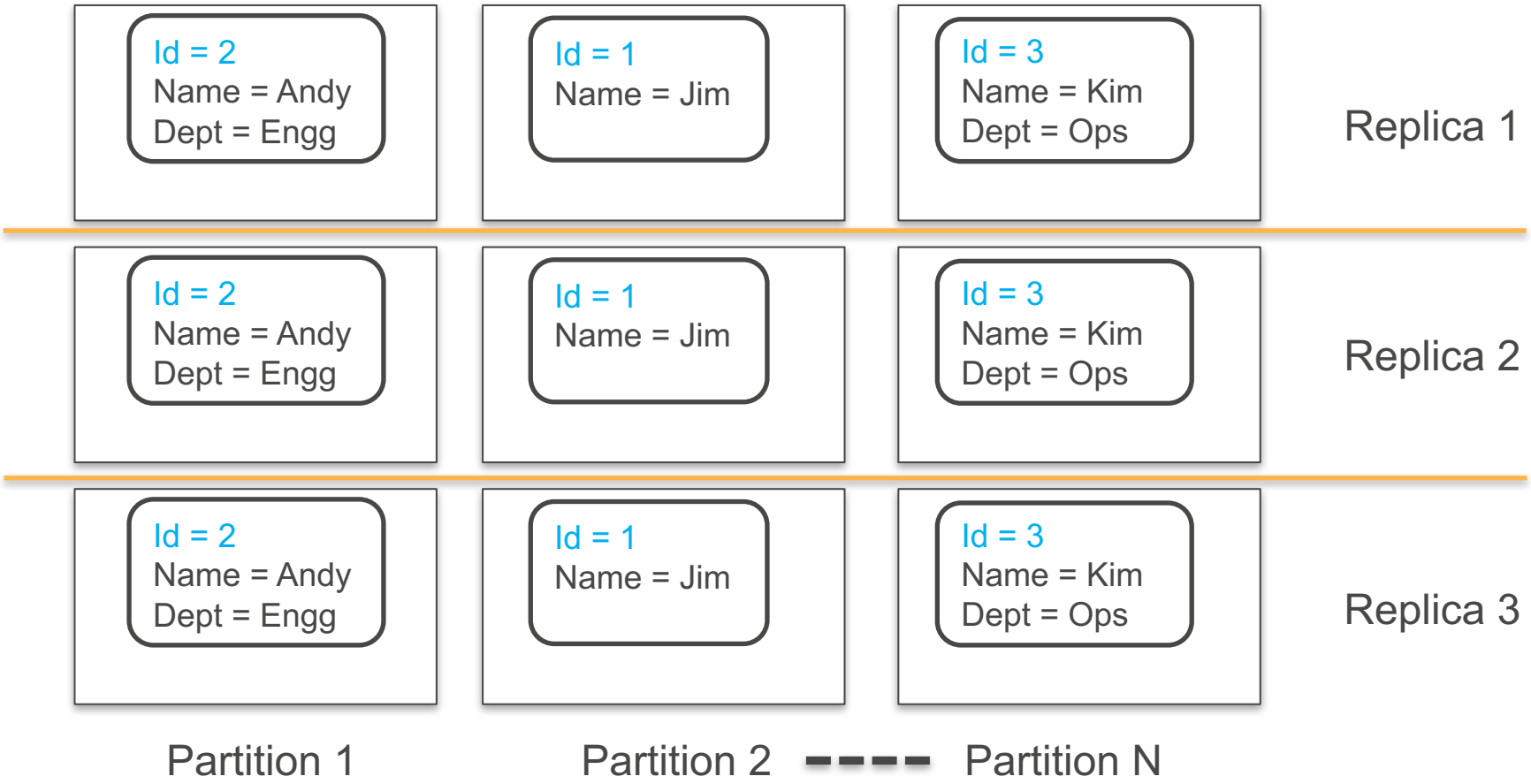
Partition-sort key table

Partition key and sort key together uniquely identify an Item
Within unordered partition key-space, data is sorted by the sort key
No limit on the number of items (∞) per partition key

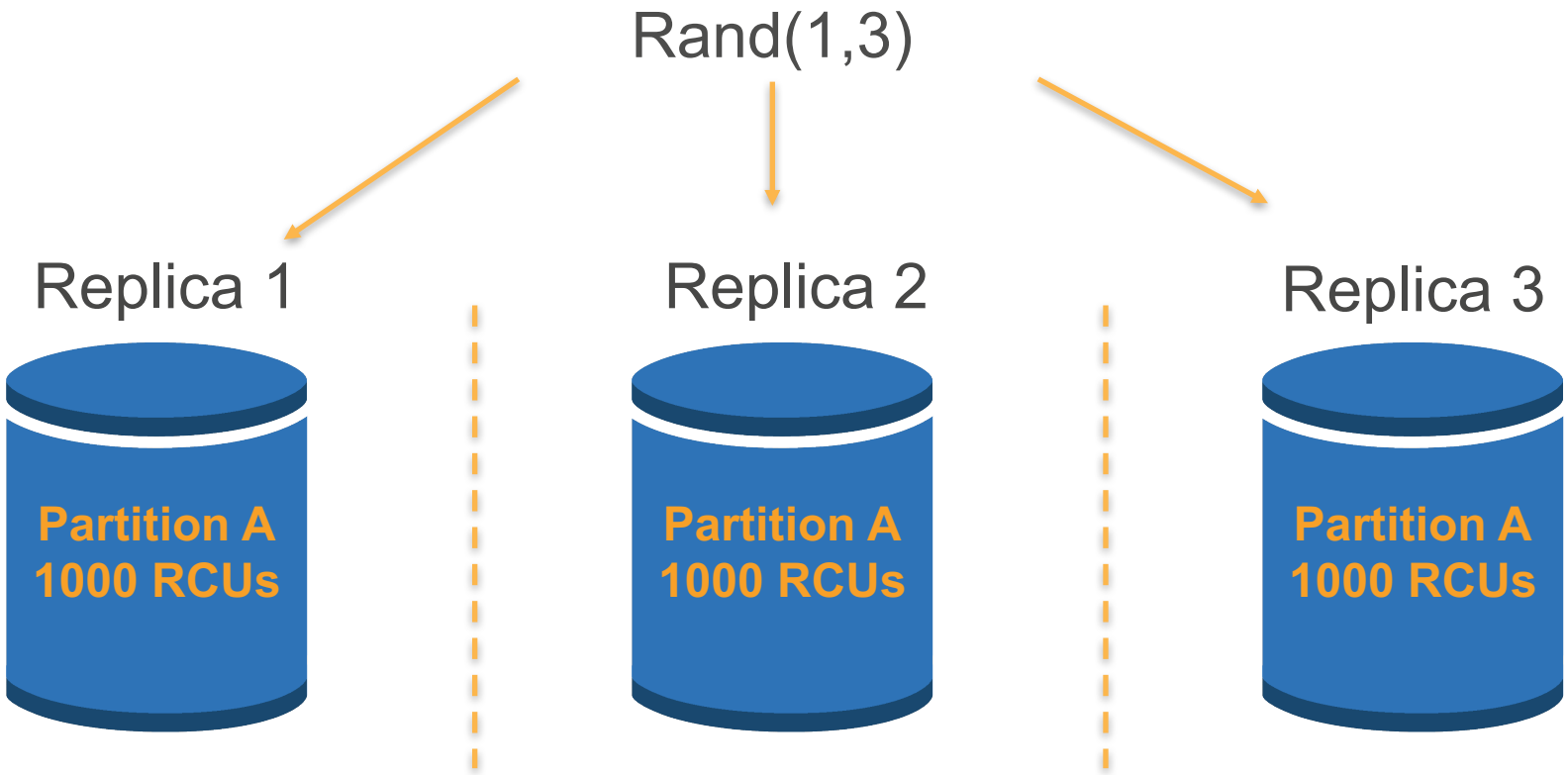
- Except if you have local secondary indexes



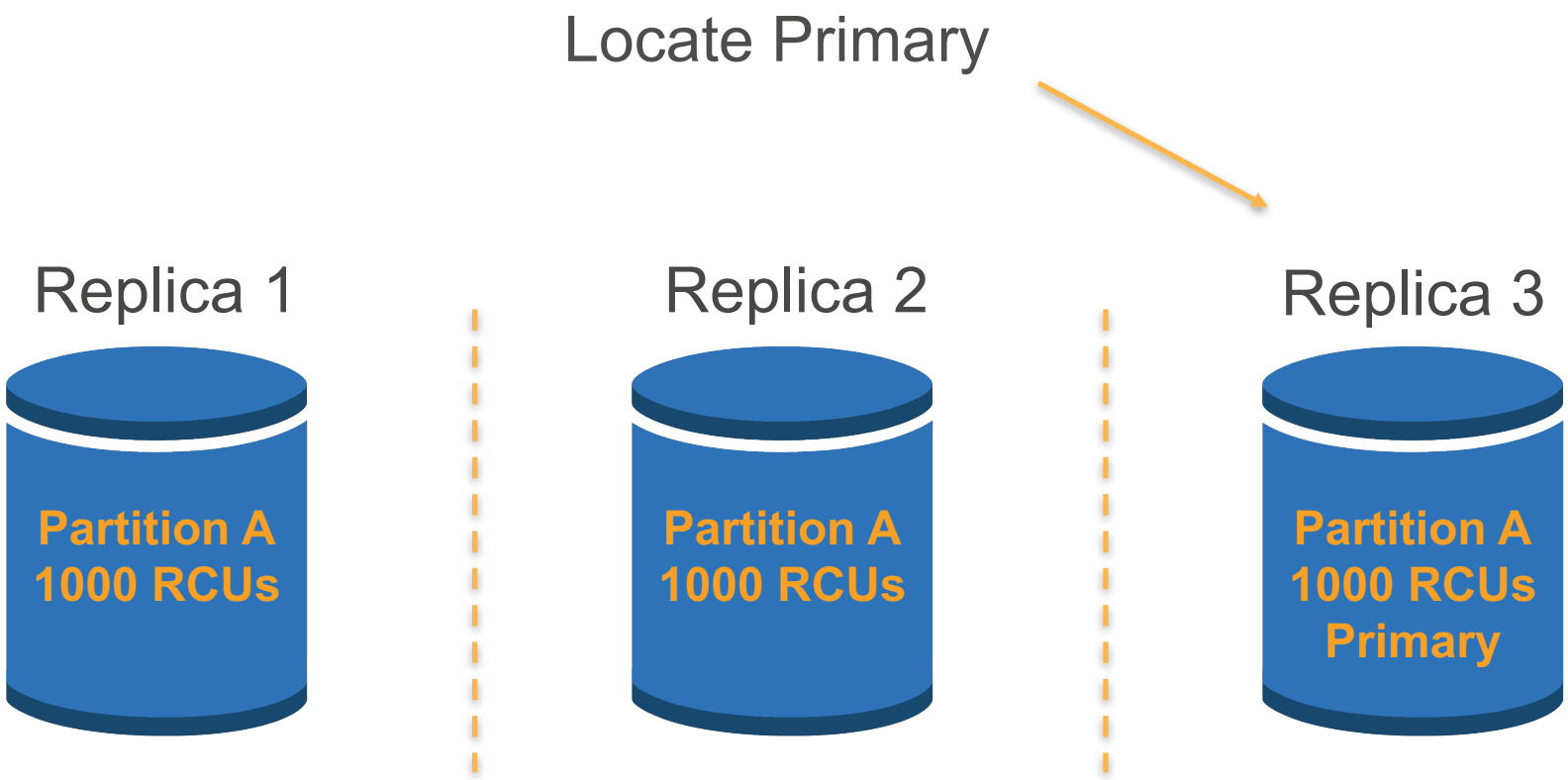
Partitions are three-way replicated



Eventually Consistent Reads



Strongly Consistent Reads

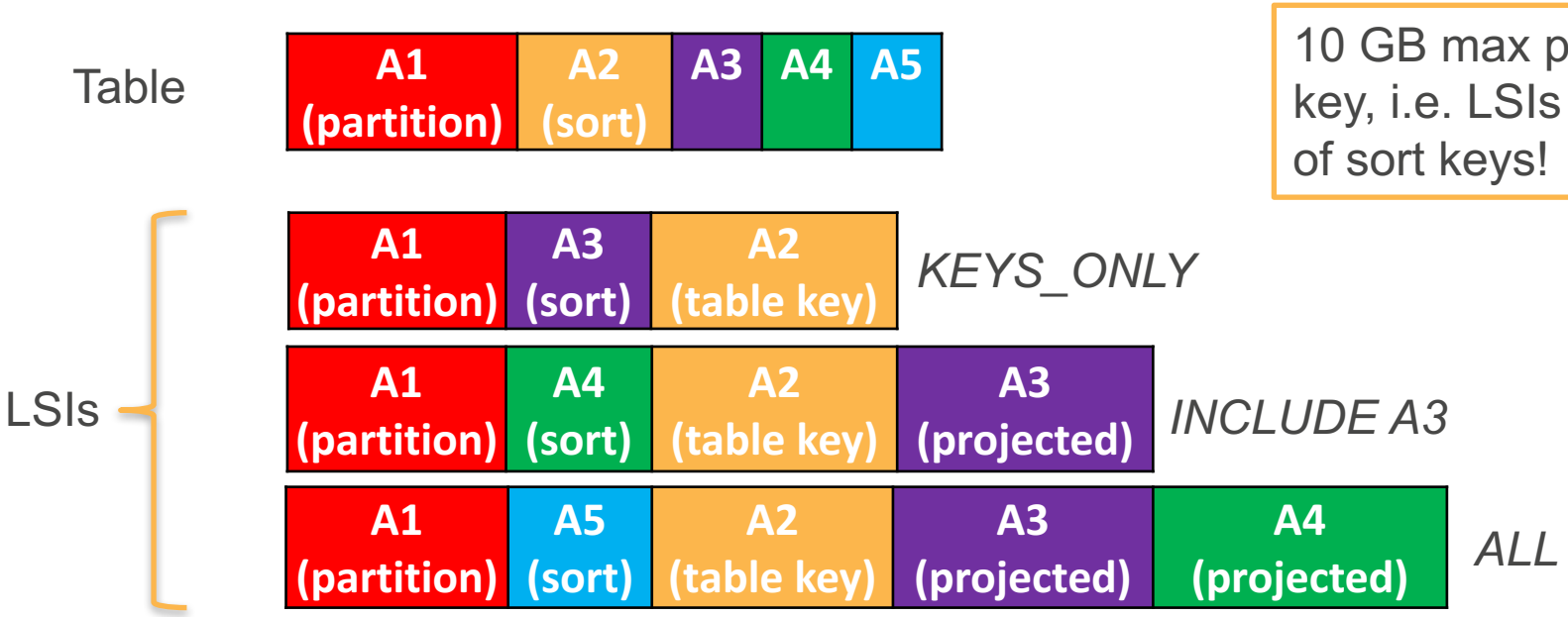


Indexes

Local secondary index (LSI)

Alternate sort key attribute

Index is local to a partition key



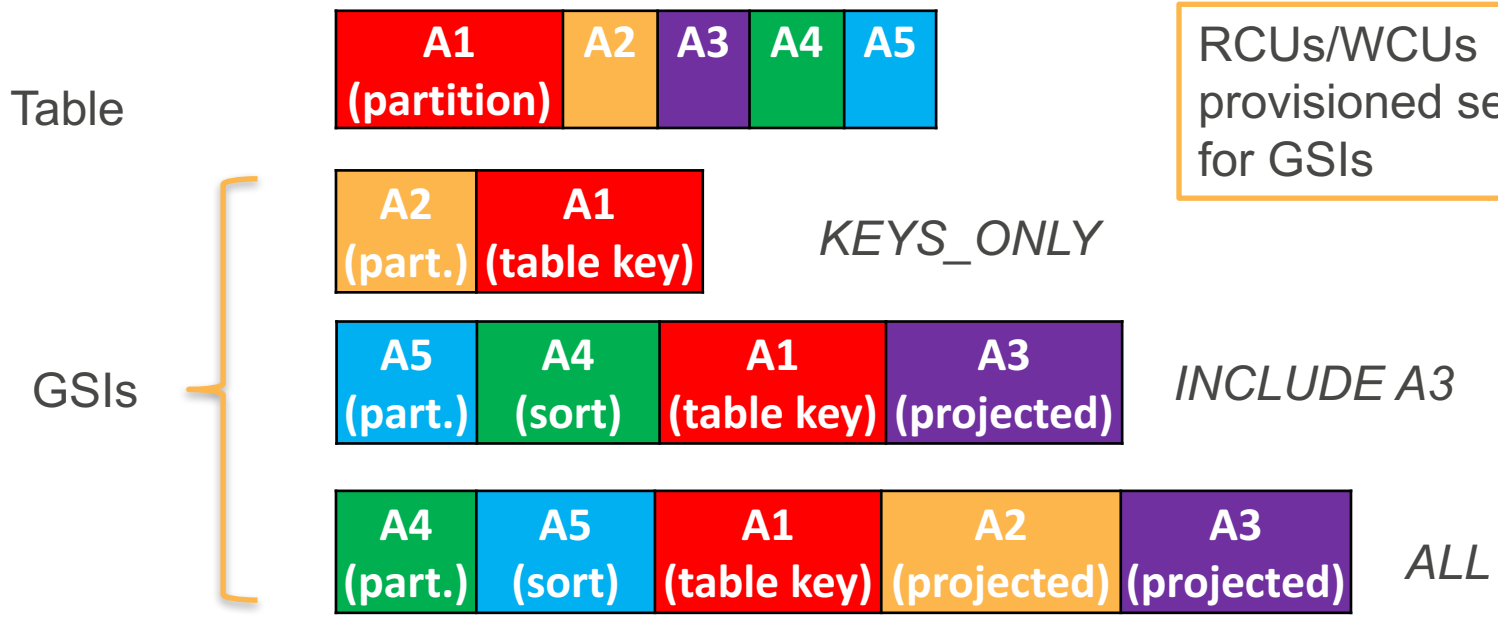
10 GB max per partition key, i.e. LSIs limit the # of sort keys!

Global secondary index (GSI)

Online Indexing

Alternate partition (+sort) key

Index is across all table partition keys



RCUs/WCUs provisioned separately for GSIs

LSI or GSI?

LSI can be modeled as a GSI

If data size in an item collection > 10 GB, use GSI

If eventual consistency is okay for your scenario, use GSI!

Scaling

Scaling

Throughput

- Provision any amount of throughput to a table

Size

- Add any number of items to a table
  - Max item size is 400 KB
  - LSIs limit the number of range keys due to 10 GB limit

Scaling is achieved through partitioning

Throughput

Provisioned at the table level

- Write capacity units (WCUs) are measured in 1 KB per second
- Read capacity units (RCUs) are measured in 4 KB per second
  - RCUs measure strictly consistent reads
  - Eventually consistent reads cost 1/2 of consistent reads

Read and write throughput limits are independent

Partitioning Math

Number of Partitions	
By Capacity	(Total RCU / 3000) + (Total WCU / 1000)
By Size	Total Size / 10 GB
Total Partitions	CEILING(MAX (Capacity, Size))

Partitioning Example

Table size = 8 GB, RCUs = 5000, WCUs = 500

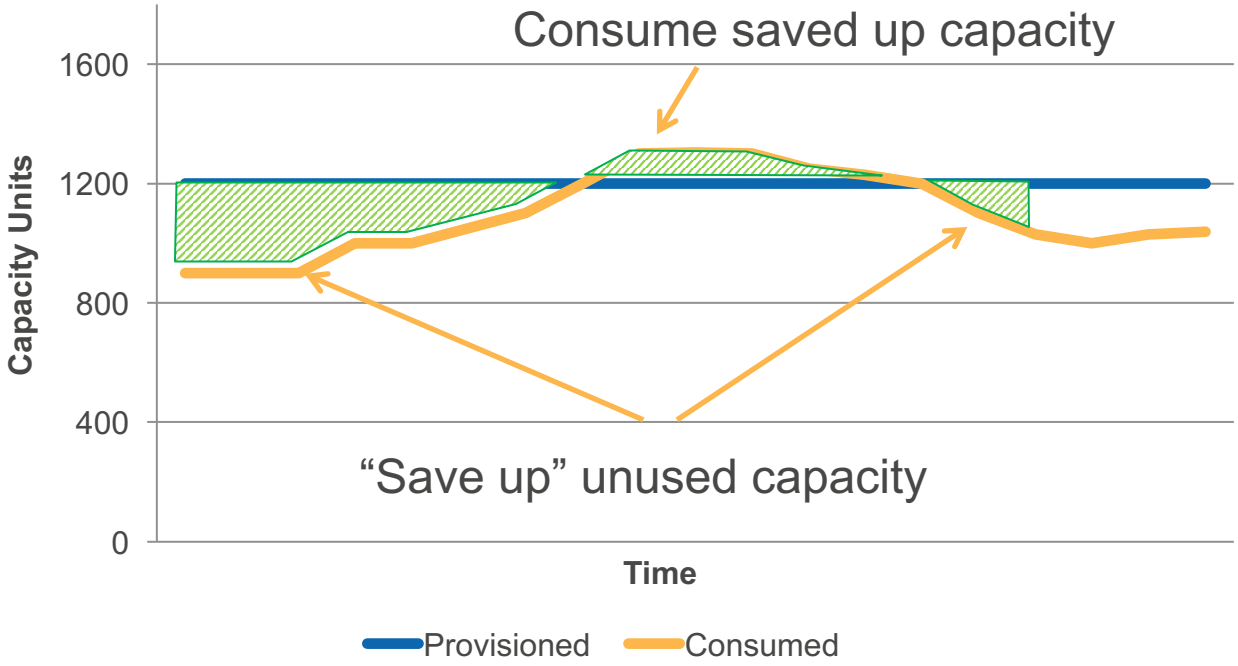
Number of Partitions	
By Capacity	(5000 / 3000) + (500 / 1000) = 2.17
By Size	8 / 10 = 0.8
Total Partitions	CEILING(MAX (2.17, 0.8)) = 3

RCUs and WCUs are uniformly spread across partitions

→

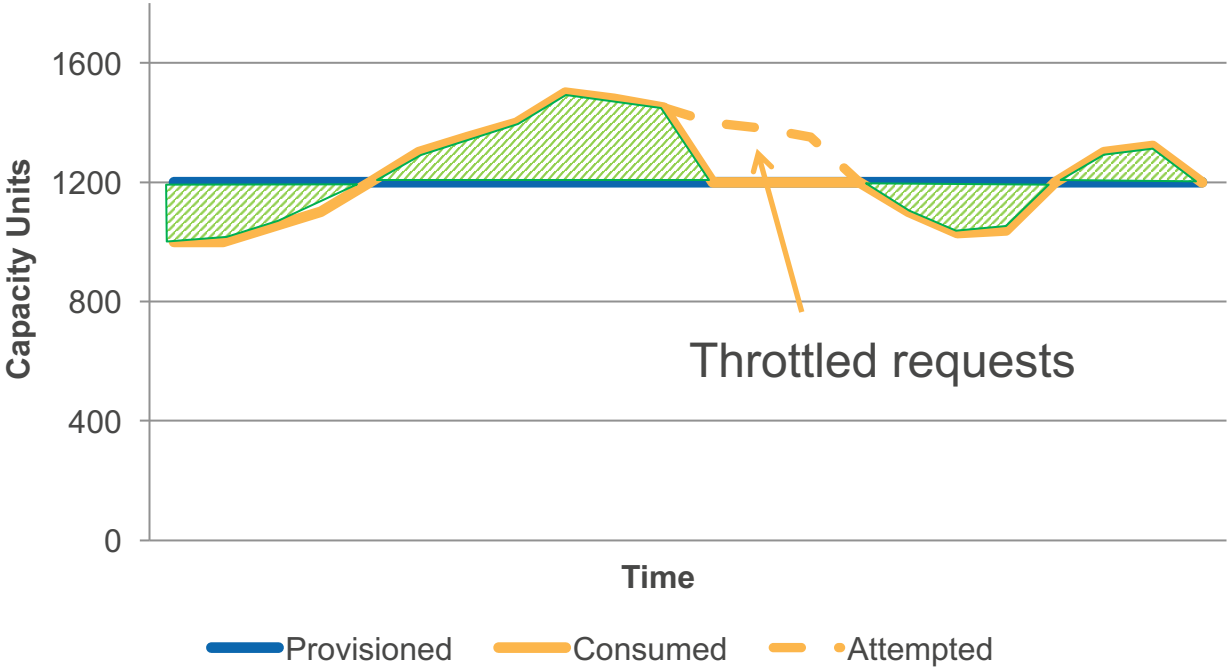
RCUs per partition = 5000/3 = 1666.67
WCUs per partition = 500/3 = 166.67
Data/partition = 10/3 = 3.33 GB

Burst capacity is built-in



Burst: 300 seconds
(1200 × 300 = 360k CU)

Burst capacity may not be sufficient



Burst: 300 seconds
(1200 × 300 = 360k CU)

Don’t completely depend on burst capacity… provision sufficient throughput

What causes throttling?

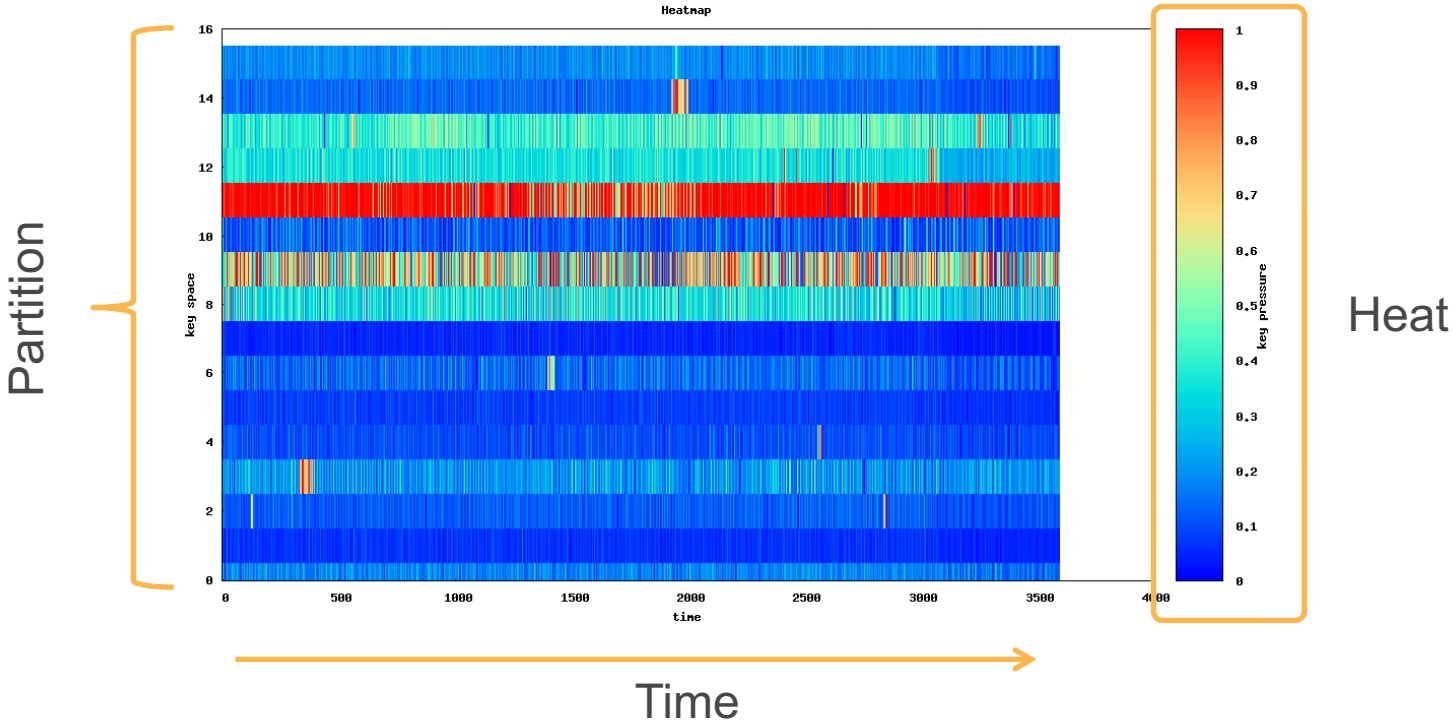
Non-uniform workloads

- Hot keys/hot partitions
- Very large bursts

Dilution of throughput across partitions caused by mixing hot data with cold data

- Use a table per time period for storing time series data so WCUs and RCUs are applied to the hot data set

Example: Key Choice or Uniform Access



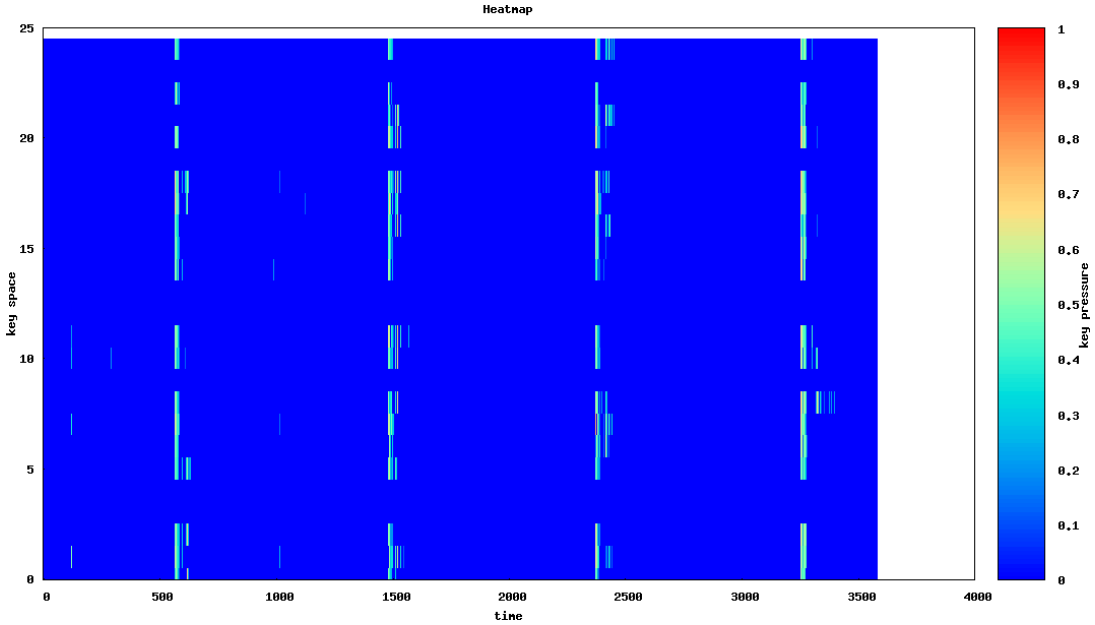
Getting the most out of DynamoDB throughput

“To get the most out of DynamoDB throughput, create tables where the partition key has a large number of distinct values, and values are requested fairly uniformly, as randomly as possible.”

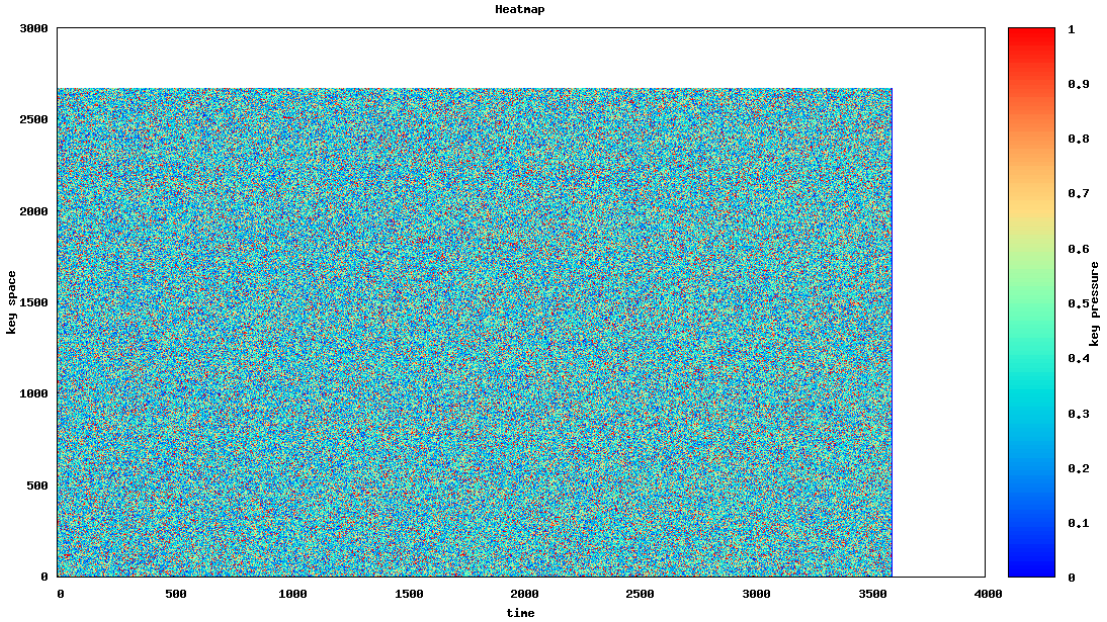
—*DynamoDB Developer Guide*

1. Key Choice: High key cardinality (“uniqueness”)
2. Uniform Access: access is evenly spread over the key-space
3. Time: requests arrive evenly spaced in time

Example: Time-based Access



Example: Uniform access



Data Modeling

Store data based on how you will access it!

1:1 relationships or key-values

Use a table or GSI with a partition key

Use GetItem or BatchGetItem API

Example: Given a user or email, get attributes

Users Table	
Partition key	Attributes
UserId = bob	Email = bob@gmail.com, JoinDate = 2011-11-15
UserId = fred	Email = fred@yahoo.com, JoinDate = 2011-12-01

Users-Email-GSI	
Partition key	Attributes
Email = bob@gmail.com	UserId = bob, JoinDate = 2011-11-15
Email = fred@yahoo.com	UserId = fred, JoinDate = 2011-12-01

1:N relationships or parent-children

Use a table or GSI with partition and sort key

Use Query API

Example: Given a device, find all readings between epoch X, Y

Device-measurements		
Part. Key	Sort key	Attributes
DeviceId = 1	epoch = 5513A97C	Temperature = 30, pressure = 90
DeviceId = 1	epoch = 5513A9DB	Temperature = 30, pressure = 90

N:M relationships

Use a table and GSI with partition and sort key elements switched

Use Query API

Example: Given a user, find all games. Or given a game, find all users.

User-Games-Table	
Part. Key	Sort key
UserId = bob	GameId = Game1
UserId = fred	GameId = Game2
UserId = bob	GameId = Game3

Game-Users-GSI	
Part. Key	Sort key
GameId = Game1	UserId = bob
GameId = Game2	UserId = fred
GameId = Game3	UserId = bob

Documents (JSON)

Data types (M, L, BOOL, NULL) introduced to support JSON

Document SDKs

- Simple programming model
- Conversion to/from JSON
- Java, JavaScript, Ruby, .NET

Cannot create an Index on elements of a JSON object stored in Map

- They need to be modeled as top-level table attributes to be used in LSIs and GSIs

Set, Map, and List have no element limit but depth is 32 levels

Javascript	DynamoDB
string	S
number	N
boolean	BOOL
null	NULL
array	L
object	M

```
var image = { // JSON Object for an image
    imageid: 12345,
    url: 'http://example.com/awesome_image.jpg'
};
var params = {
    TableName: 'images',
    Item: image, // JSON Object to store
};
dynamodb.putItem(params, function(err, data){
    // response handler
};
```

Rich expressions

Projection expression

- Query/Get/Scan: ProductReviews.FiveStar[0]

Filter expression

- Query/Scan: #V > :num (#V is a place holder for keyword VIEWS)

Conditional expression

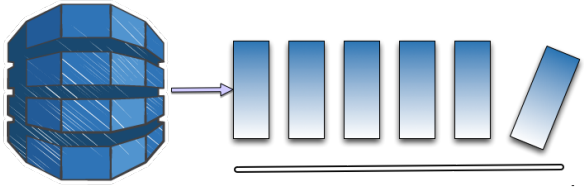
- Put/Update/DeleteItem: attribute_not_exists (#pr.FiveStar)

Update expression

- UpdateItem: set Replies = Replies + :num

DynamoDB Streams

DynamoDB Streams



Stream of updates to a table

Asynchronous

Exactly once

Strictly ordered

- Per item

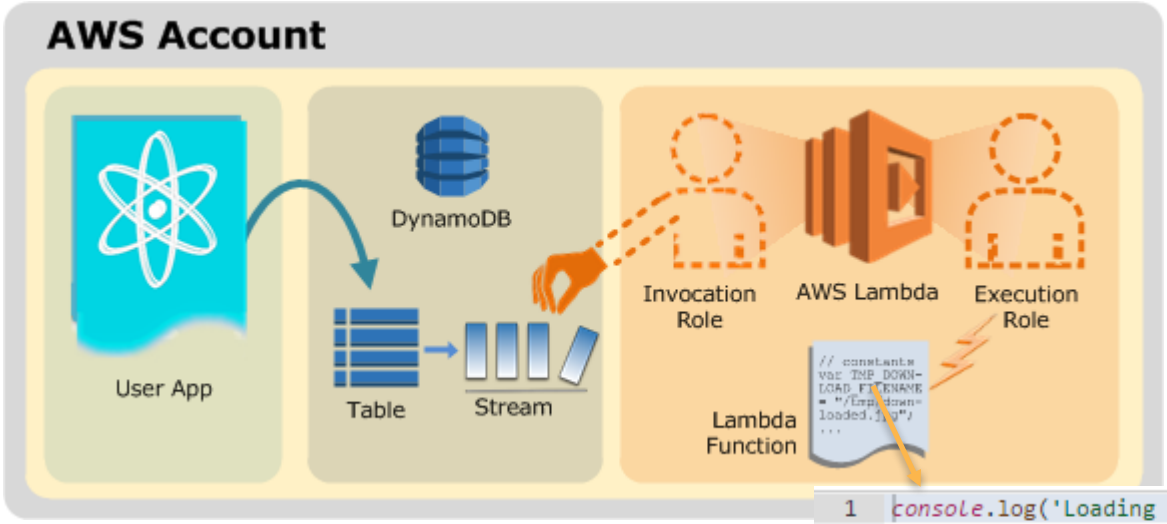
Highly durable

- Scale with table

24-hour lifetime

Sub-second latency

DynamoDB Streams and AWS Lambda

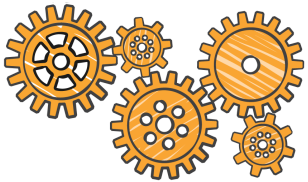


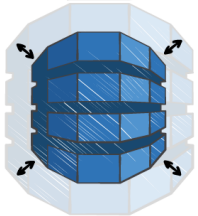

```
1  console.log('Loading event');
2  exports.handler = function(event, context) {
3    console.log("Event: %j", event);
4    for(i = 0; i < event.Records.length; ++i) {
5      record = event.Records[i];
6      console.log(record.EventID);
7      console.log(record.EventName);
8      console.log("DynamoDB Record: %j", record.Dynamodb);
9    }
10   context.done(null, "Hello World");  // SUCCESS with message
11 }
```

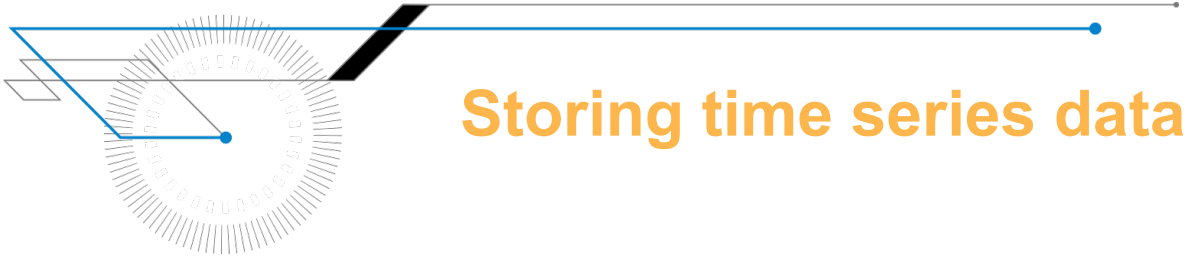
```
2015-03-21T07:44:58.883Z 2ca3769a-cf9e-11e4-b270-ad4d24b312ff INSERT
2015-03-21T07:44:58.883Z 2ca3769a-cf9e-11e4-b270-ad4d24b312ff DynamoDB Record:{ "NewImage": { "name": { "S": "sivar" }, "hk": { "S": "3" } }, "SizeBytes": 15, "StreamViewType": "NEW_AND_OLD_IMAG
2015-03-21T07:44:58.883Z 2ca3769a-cf9e-11e4-b270-ad4d24b312ff Message: "Hello World"
```

Scenarios and Best Practices

Event Logging

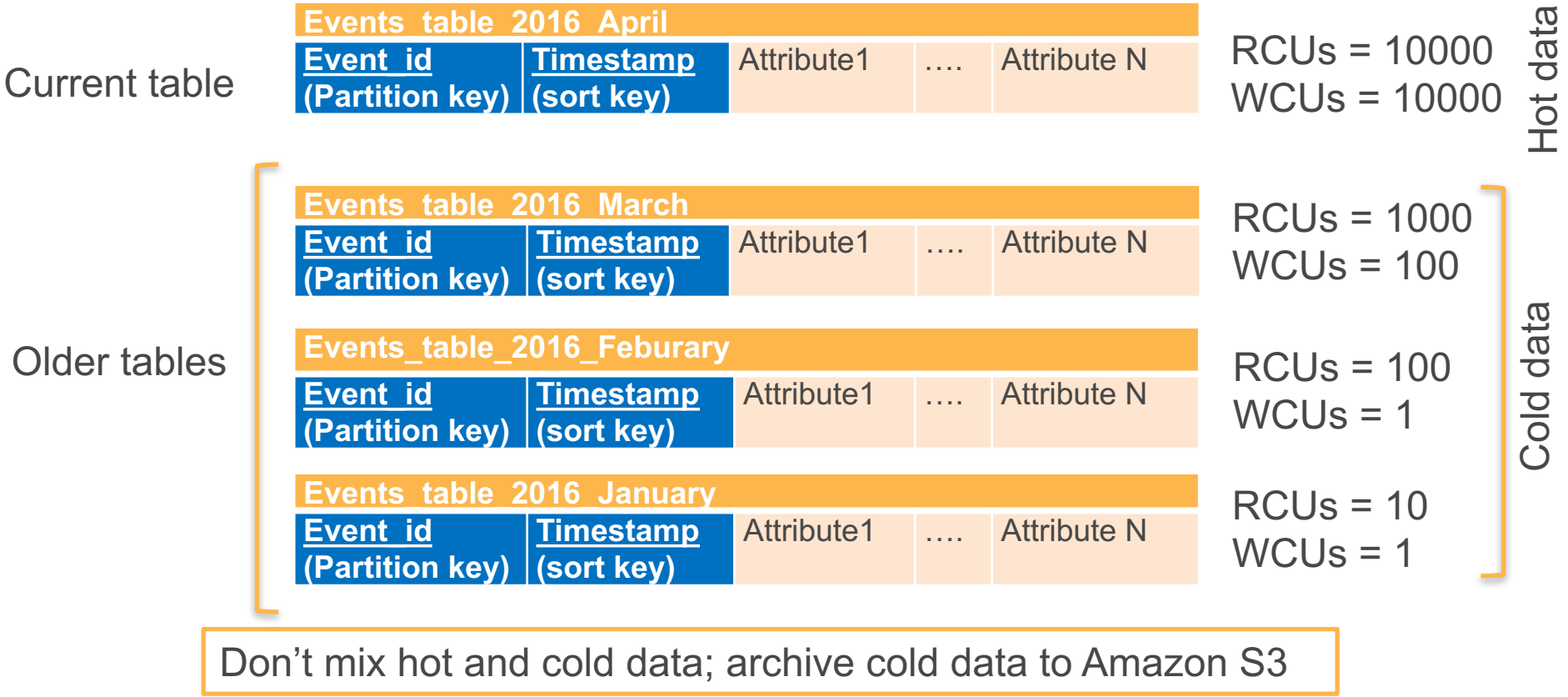






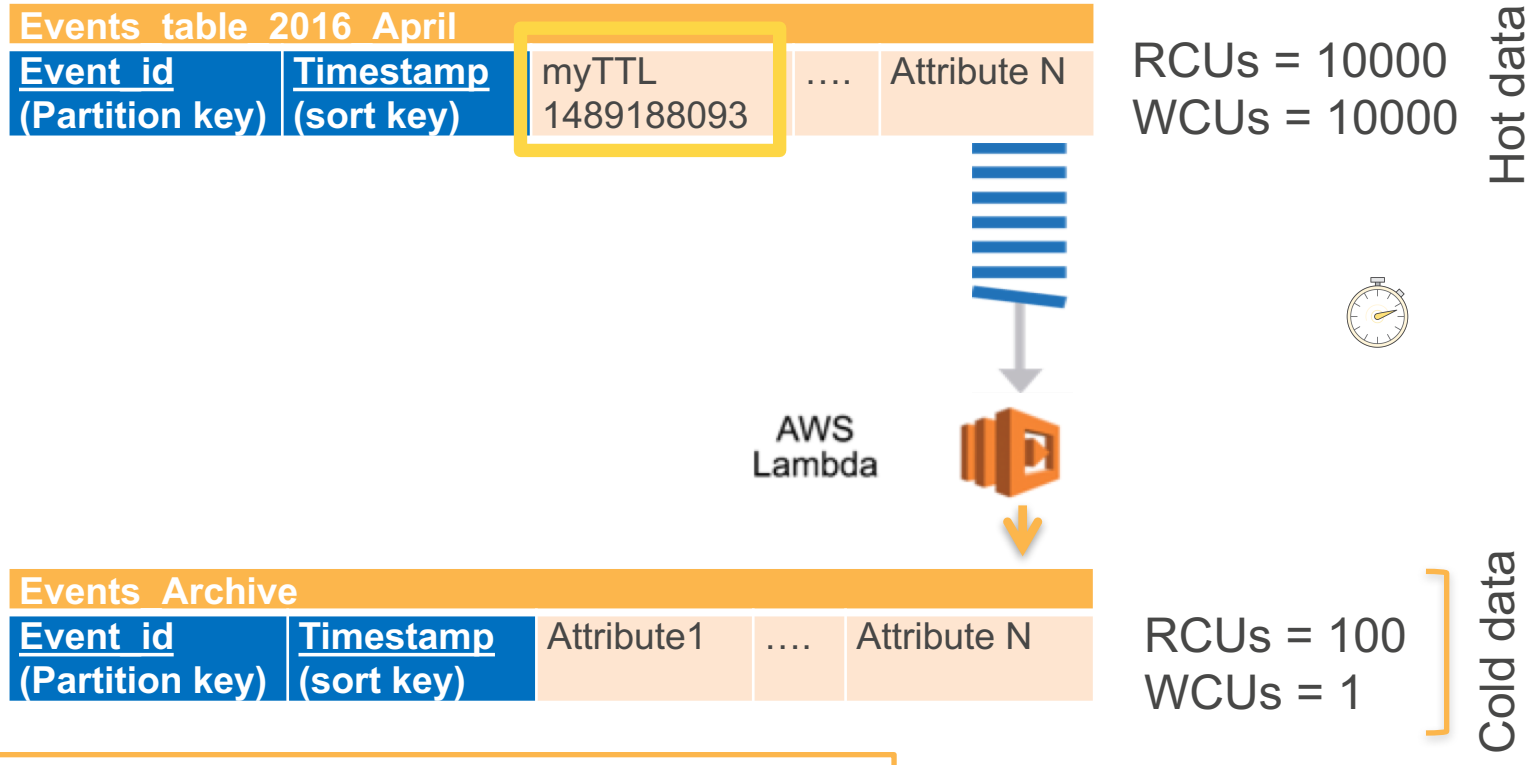
Storing time series data

Time series tables



DynamoDB TTL

Current table



Use DynamoDB TTL and Streams to archive

Isolate cold data from hot data

Pre-create daily, weekly, monthly tables

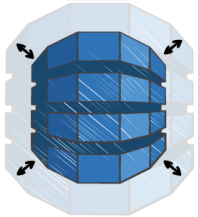
Provision required throughput for current table

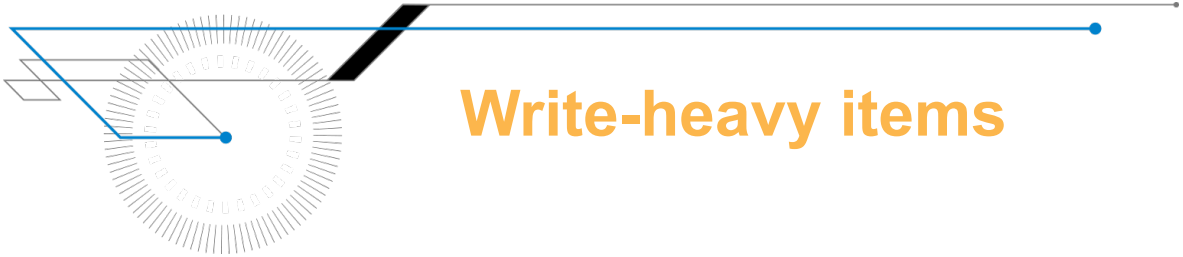
Writes go to the current table

Turn off (or reduce) throughput for older tables
OR move items to separate table with TTL

Important when: Dealing with time series data

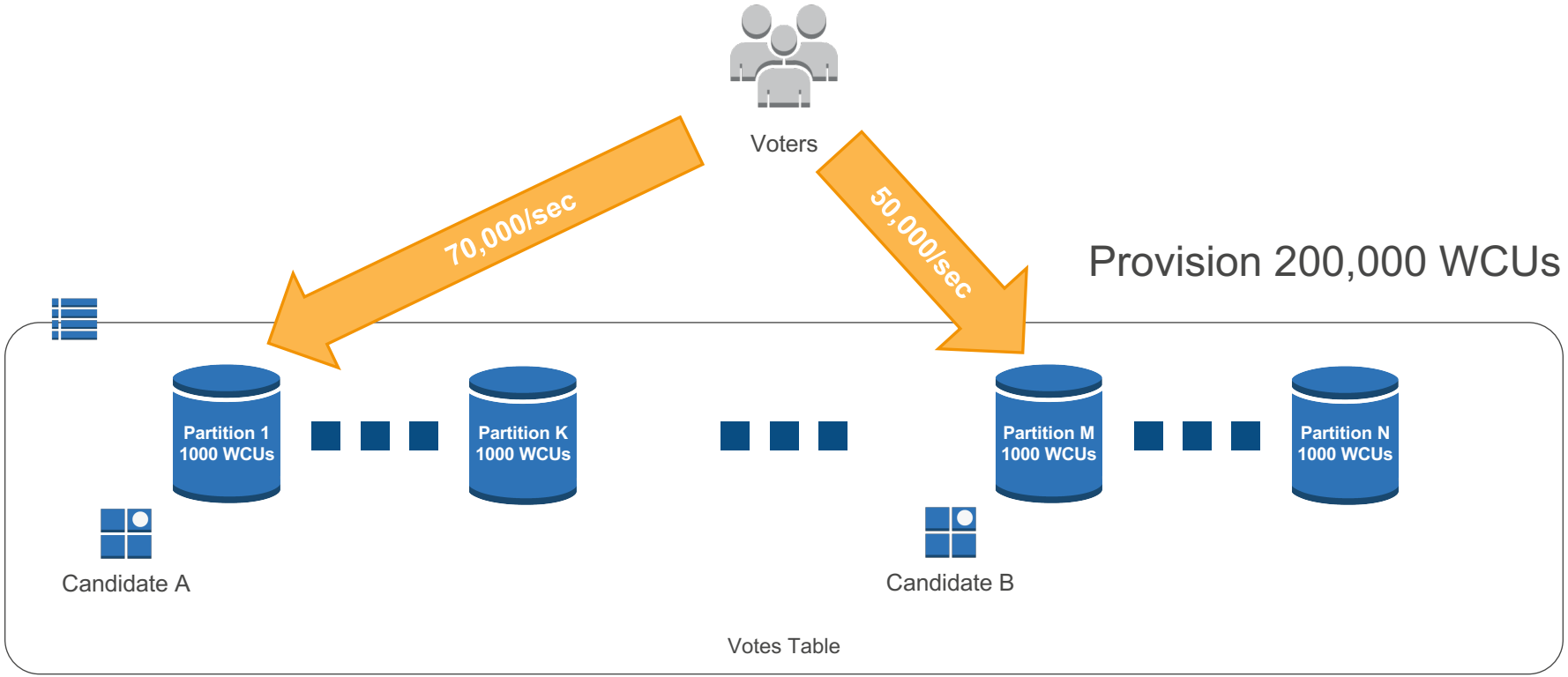
Real-Time Voting



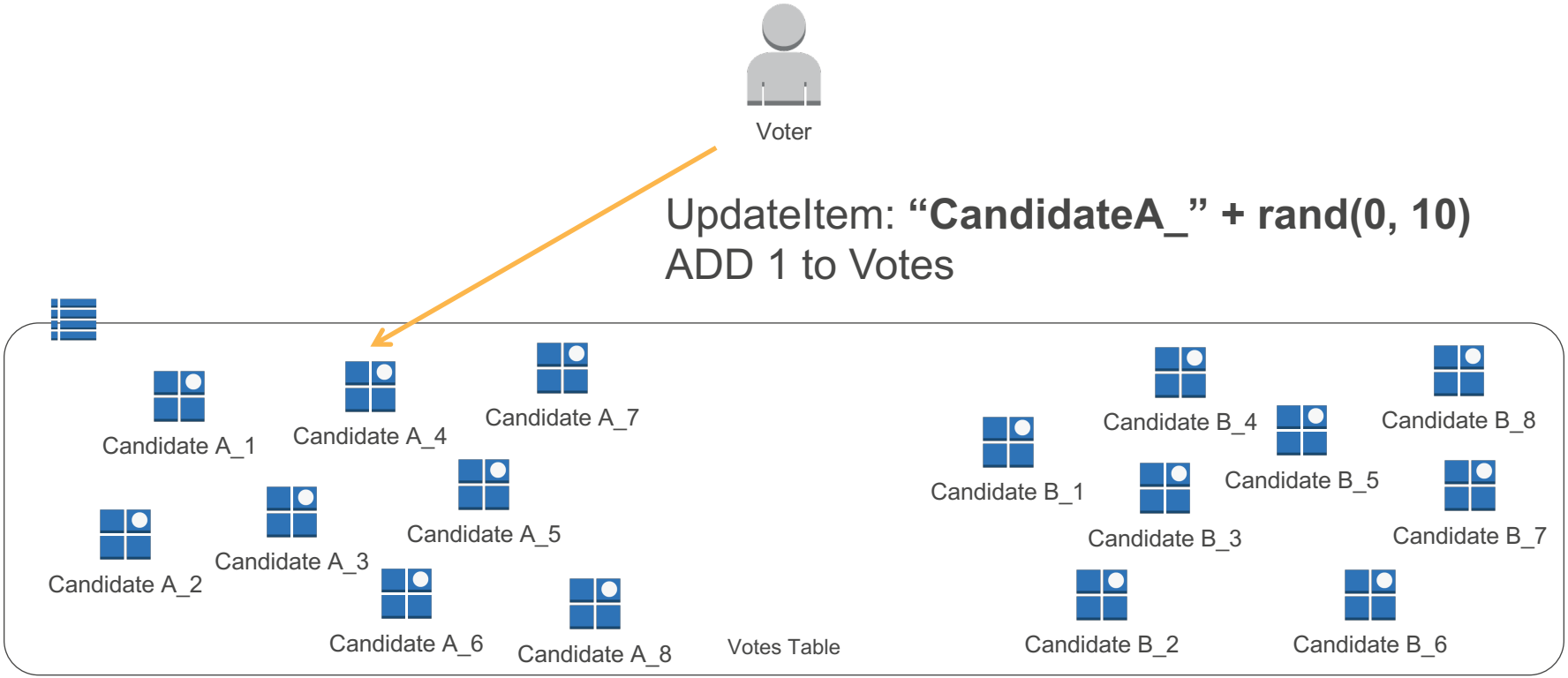


Write-heavy items

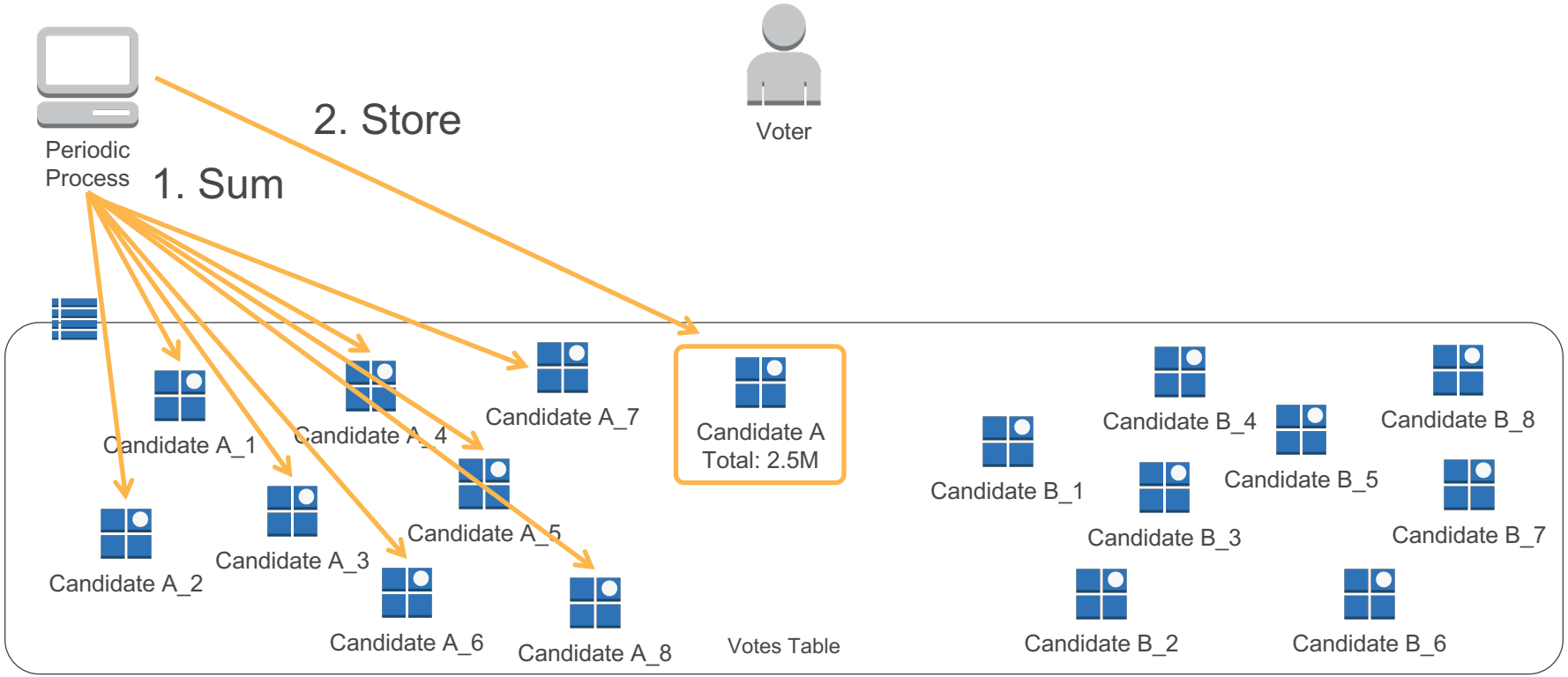
Scaling bottlenecks



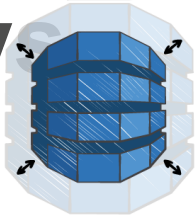
Write sharding



Shard aggregation

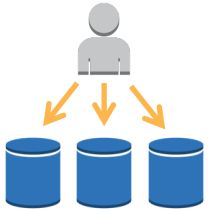


Shard write-heavy partition key



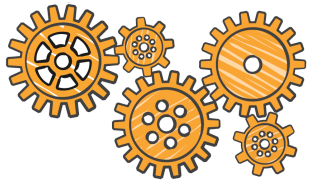
Trade off read cost for write scalability

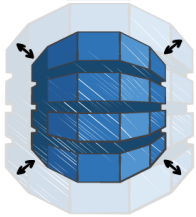
Consider throughput per partition key and per partition

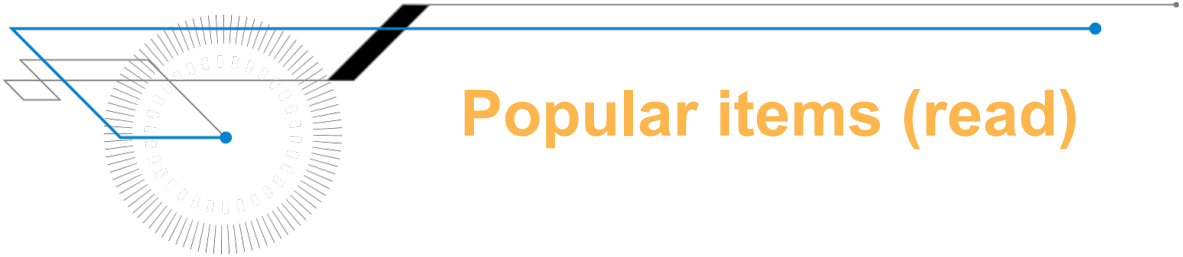


Important when: Your write workload is not horizontally scalable

Product Catalog





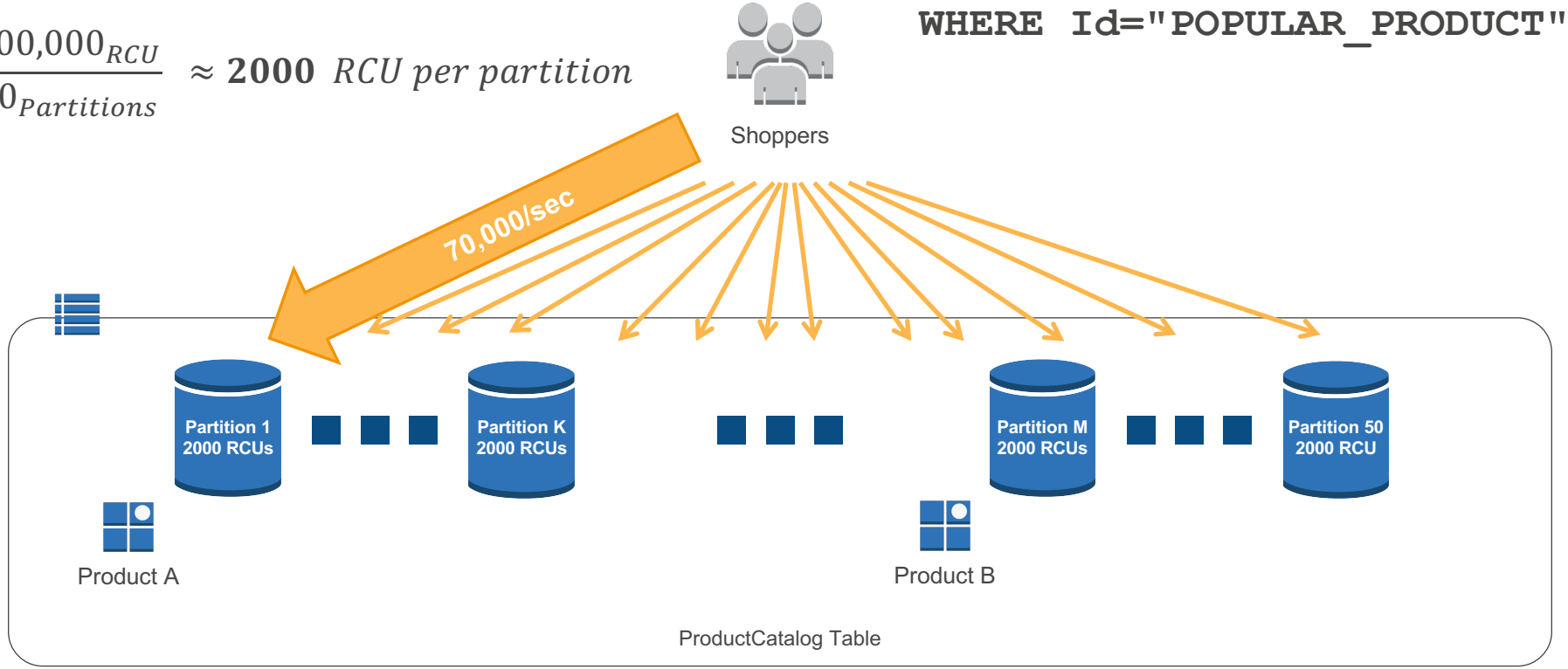


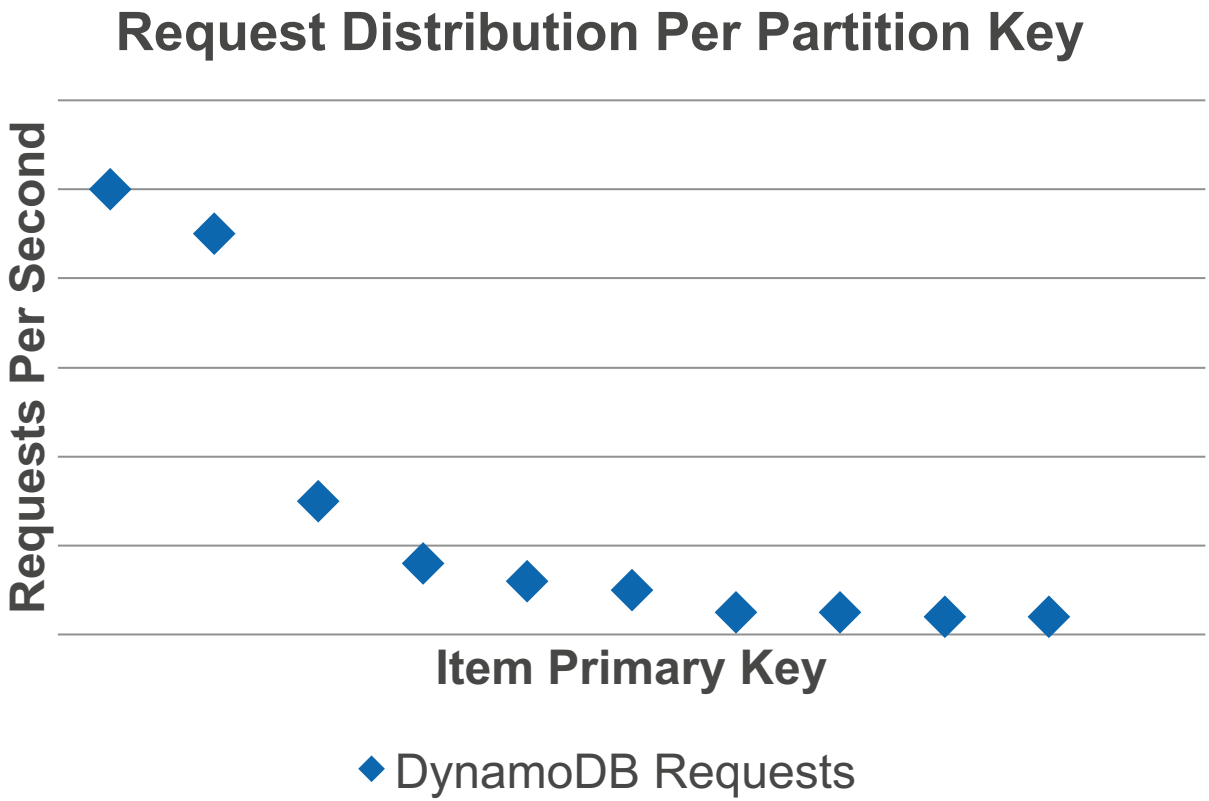
Popular items (read)

Scaling bottlenecks

$$\frac{100{,}000_{RCU}}{50_{Partitions}} \approx \mathbf{2000}\ RCU\ per\ partition$$

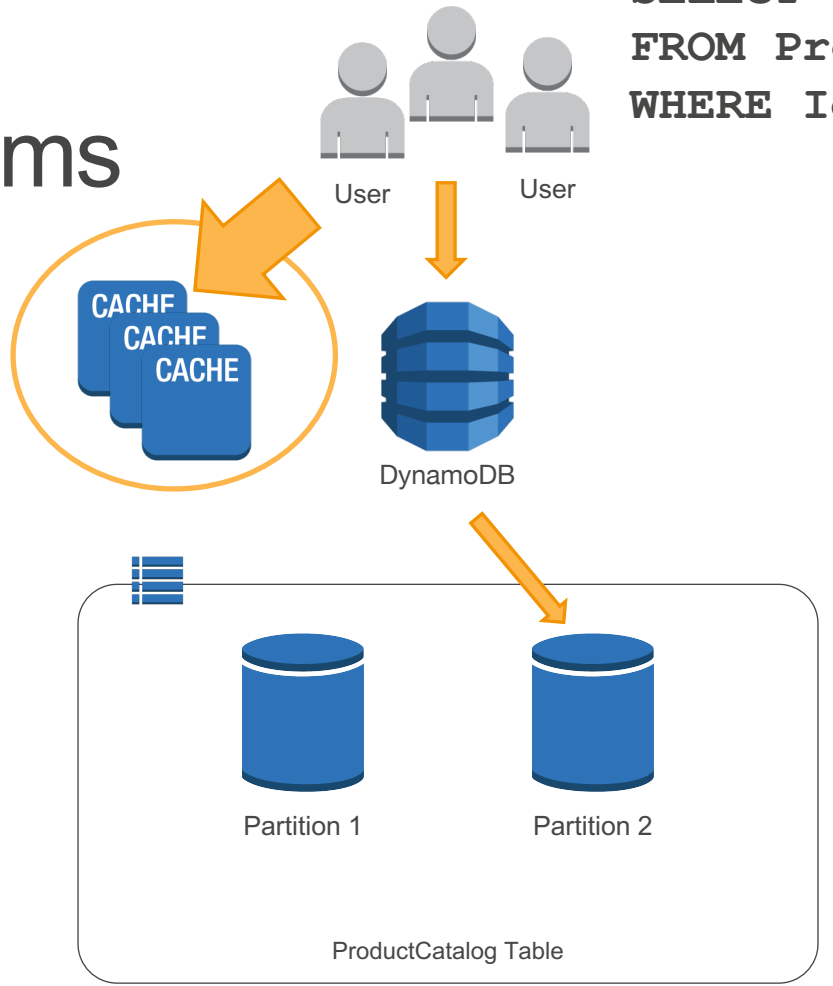
```
SELECT Id, Description, ...
FROM ProductCatalog
WHERE Id="POPULAR_PRODUCT"
```

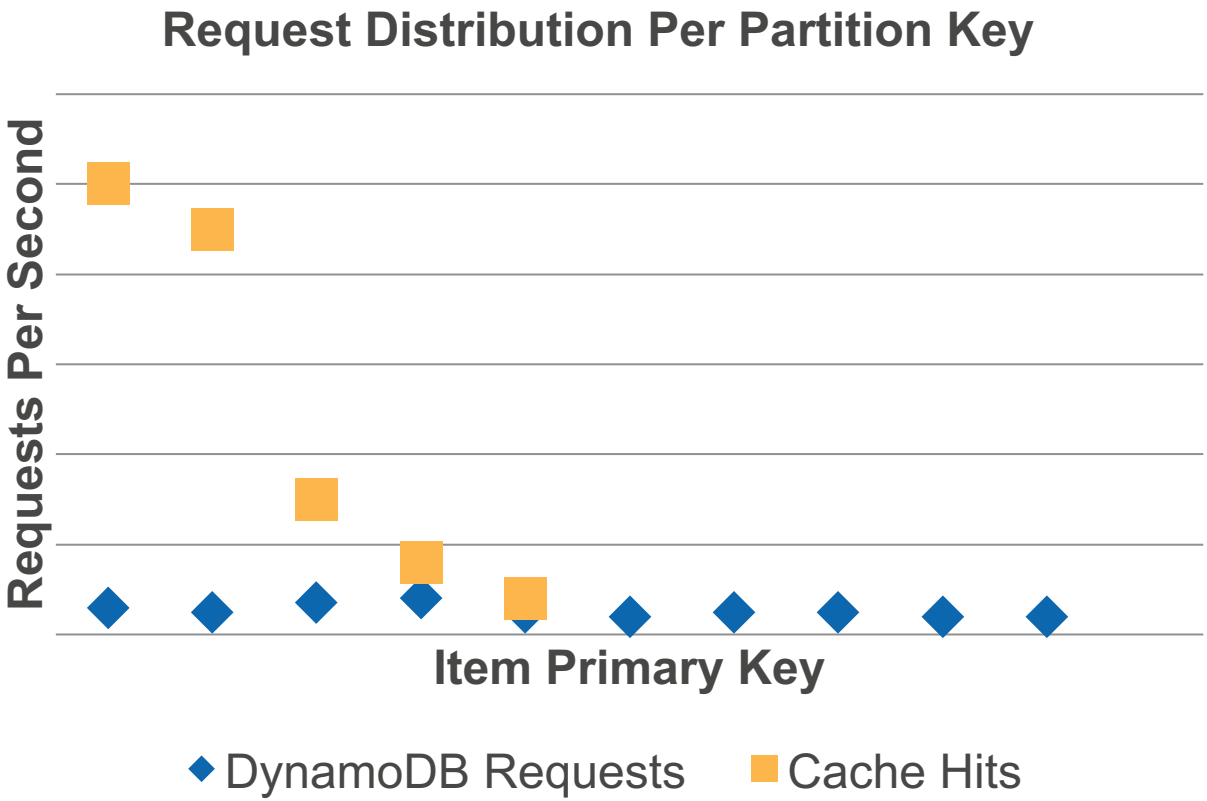



Request Distribution Per Partition Key
Requests Per Second
Item Primary Key
DynamoDB Requests

Cache popular items

```
SELECT Id, Description, ...
FROM ProductCatalog
WHERE Id="POPULAR_PRODUCT"
```




Request Distribution Per Partition Key
Requests Per Second
Item Primary Key
DynamoDB Requests
Cache Hits

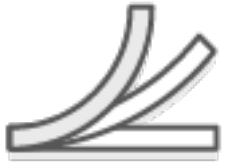
Amazon DynamoDB Accelerator (DAX)

Extreme Performance

Highly Scalable

Fully Managed

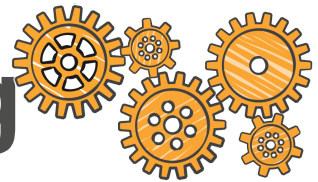
Ease of Use

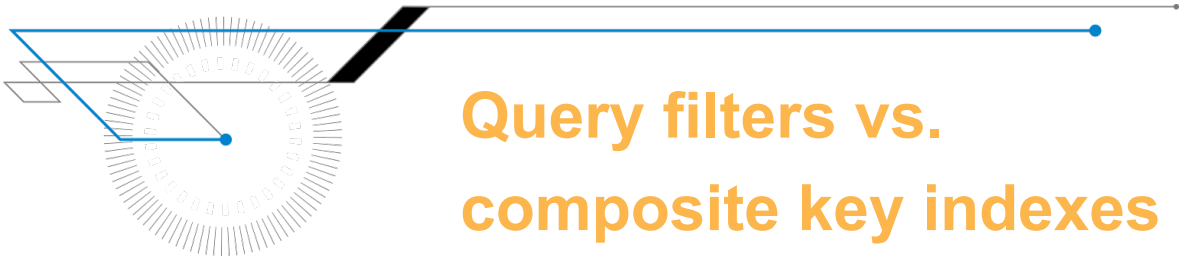


Flexible

Secure

Multiplayer Online Gaming





Query filters vs. composite key indexes

Multivalue sorts and filters

Partition key

Sort key

Secondary index

Opponent	Date	GameId	Status	Host
Alice	2014-10-02	d9bl3	DONE	David
Carol	2014-10-08	o2pnb	IN_PROGRESS	Bob
Bob	2014-09-30	72f49	PENDING	Alice
Bob	2014-10-03	b932s	PENDING	Carol
Bob	2014-10-03	ef9ca	IN_PROGRESS	David

Approach 1: Query filter

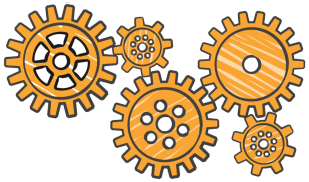
```
SELECT * FROM Game
WHERE Opponent='Bob'
ORDER BY Date DESC
FILTER ON Status='PENDING'
```

Secondary Index

Opponent	Date	GameId	Status	Host
Alice	2016-10-02	d9bl3	DONE	David
Carol	2016-10-08	o2pnb	IN_PROGRESS	Bob
Bob	2016-09-30	72f49	PENDING	Alice
Bob	2016-10-03	b932s	PENDING	Carol
Bob	2016-10-03	ef9ca	IN_PROGRESS	David

(filtered out)

Use query filter



Send back less data “on the wire”

Simplify application code

Simple SQL-like expressions

- AND, OR, NOT, ()

Important when: Your index isn’t entirely selective

Approach 2: Composite key

Status
DONE
IN_PROGRESS
IN_PROGRESS
PENDING
PENDING

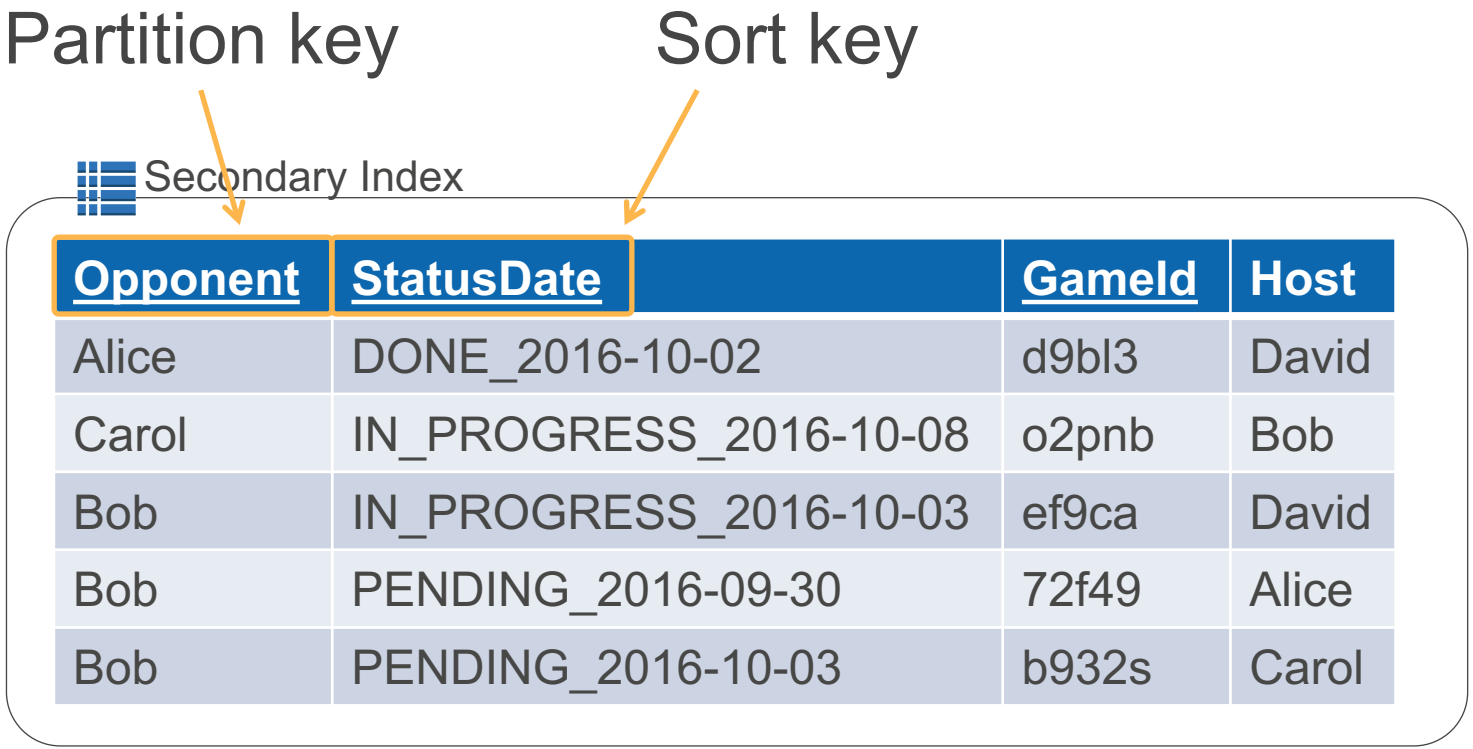
+

Date
2016-10-02
2016-10-08
2016-10-03
2016-10-03
2016-09-30

=

StatusDate
DONE_2016-10-02
IN_PROGRESS_2016-10-08
IN_PROGRESS_2016-10-03
PENDING_2016-09-30
PENDING_2016-10-03

Approach 2: Composite key



Opponent	StatusDate	GameId	Host
Alice	DONE_2016-10-02	d9bl3	David
Carol	IN_PROGRESS_2016-10-08	o2pnb	Bob
Bob	IN_PROGRESS_2016-10-03	ef9ca	David
Bob	PENDING_2016-09-30	72f49	Alice
Bob	PENDING_2016-10-03	b932s	Carol

Approach 2: Composite key

```
SELECT * FROM Game
WHERE Opponent='Bob'
    AND StatusDate BEGINS_WITH 'PENDING'
```

Secondary Index

Opponent	StatusDate	GameId	Host
Alice	DONE_2016-10-02	d9bl3	David
Carol	IN_PROGRESS_2016-10-08	o2pnb	Bob
Bob	IN_PROGRESS_2016-10-03	ef9ca	David
Bob	PENDING_2016-09-30	72f49	Alice
Bob	PENDING_2016-10-03	b932s	Carol

Sparse indexes

Game-scores-table

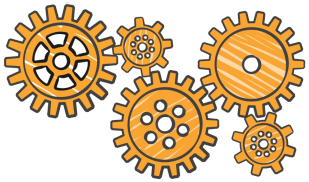
Id (Part.)	User	Game	Score	Date	Award
1	Bob	G1	1300	2015-12-23	
2	Bob	G1	1450	2015-12-23	
3	Jay	G1	1600	2015-12-24	
4	Mary	G1	2000	2015-10-24	Champ
5	Ryan	G2	123	2015-03-10	
6	Jones	G2	345	2015-03-20	

Scan sparse partition GSIs

Award-GSI

Award (Part.)	Id	User	Score
Champ	4	Mary	2000

Replace filter with indexes



Concatenate attributes to form useful secondary index keys

Take advantage of sparse indexes

Important when: You want to optimize a query as much as possible

Scaling with DynamoDB at Ocado

/OcadoTechnology

ocado TECHNOLOGY


ocado
The online supermarket

Who is Ocado

Ocado is the world's largest dedicated online grocery retailer.

Here in the UK we deliver to over 500,000 active customers shopping with us.

Each day we pick and pack over 1,000,000 items for 25,000 customers

/OcadoTechnology

What does Ocado Technology Do?

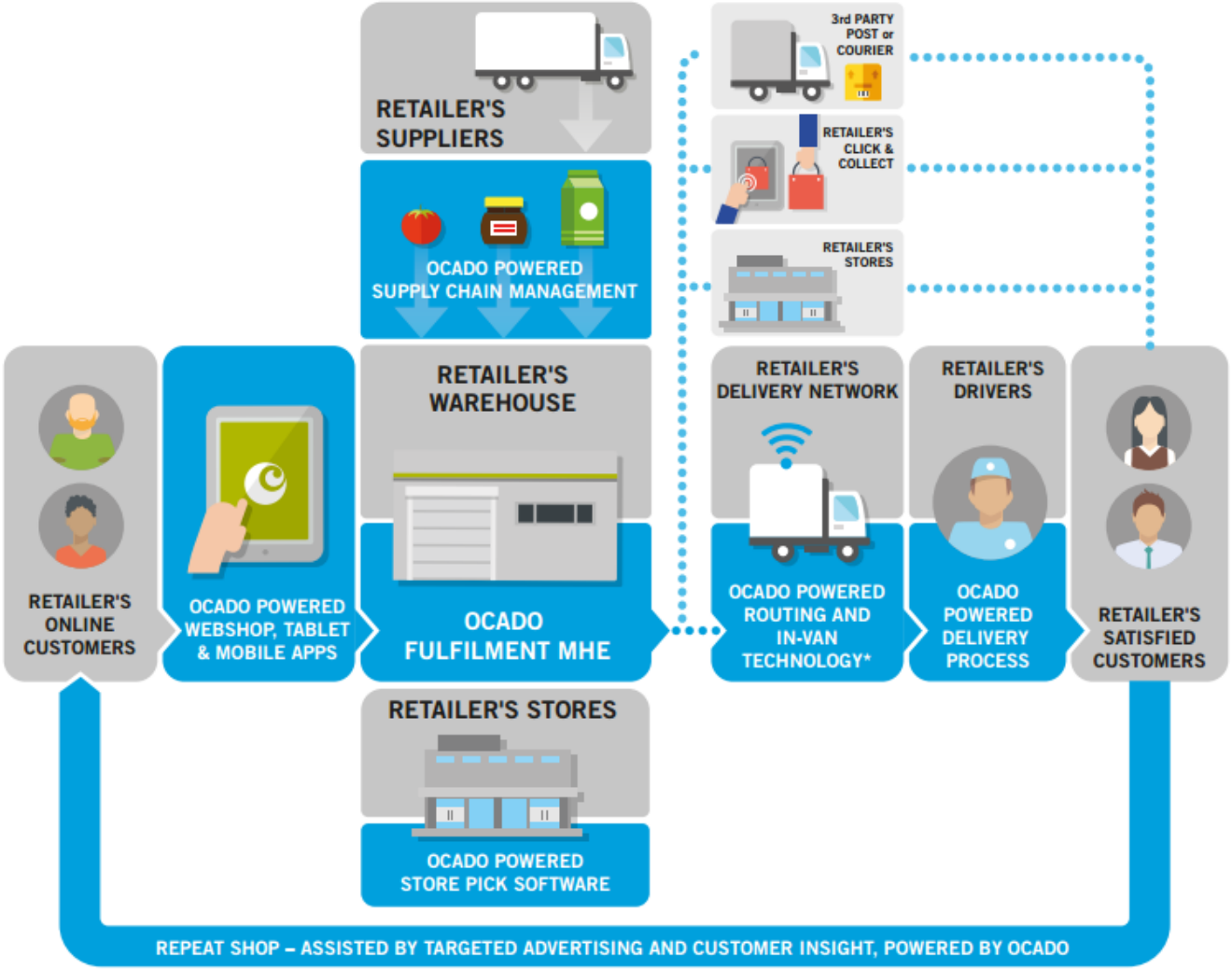
We create the websites and webapps for Ocado, Morrisons, Fetch, Sizzled, Fabled and more
Design build robotics and software for our automated warehouses.
Optimise routes for thousands of miles of deliveries each week

Ocado Smart Platform

We've also developed a proprietary solution for operating an online retail business.

This has been built with the Cloud and AWS in mind.

Allows us to offer our solutions international retailers and scale to serve customers around the globe.


RETAILER'S SUPPLIERS
OCADO POWERED SUPPLY CHAIN MANAGEMENT
RETAILER'S WAREHOUSE
OCADO FULFILMENT MHE
RETAILER'S STORES
OCADO POWERED STORE PICK SOFTWARE
3rd PARTY POST or COURIER
RETAILER'S CLICK & COLLECT
RETAILER'S STORES
RETAILER'S DELIVERY NETWORK
OCADO POWERED ROUTING AND IN-VAN TECHNOLOGY*
RETAILER'S DRIVERS
OCADO POWERED DELIVERY PROCESS
RETAILER'S SATISFIED CUSTOMERS
RETAILER'S ONLINE CUSTOMERS
OCADO POWERED WEBSHOP, TABLET & MOBILE APPS
REPEAT SHOP – ASSISTED BY TARGETED ADVERTISING AND CUSTOMER INSIGHT, POWERED BY OCADO

Our Challenges

Starting from scratch, how do we design a modern application suite architecture?

How do we minimise bottlenecks and allow ourselves to scale?

How can we support our customers (retailers) and their customers without massively increasing our size?

Application Architecture

How about them microservices?

Applications as masters of their own state

In the past we had ETLs moving data around
We wanted to control how data moved through the platform
For us this mean RESTful services or asynchronous messaging

Name-spaced Microservices

Using IAM policies each application can have it's own name-spaced AWS resources

Applications need to stateless themselves, all state must be stored in the database

Decouple how data is stored from how it's accessed.

Opting for DynamoDB

Every microservice needs it’s own database.
Easy to manage by developers themselves
Need to be highly available and as fast as needed

Things we wanted to avoid

Shared resources or infrastructure between apps
Manual intervention in order to scale
Downtime

Recommendations

Allow applications to create their own DynamoDB tables.

Allowing them to tune their own capacity too

Use namespacing to control access

```
arn:aws:dynamodb:region:account-id:table/mymicroservice-*
```

Scaling Up

While managing costs

Managed Service vs DIY

We’re a team of 5 engineers
Supporting over 1,000 developers
Working on over 200 applications

Scaling With DynamoDB

Use account level throughput limits as a cost safety control no as planned usage.

Encourage developers to scale based on a metric using the SDK.

Make sure scale down is happening too.

DynamoDB Autoscaling NEW!

Recently released is the DynamoDB Autoscaling feature.

It allows you to set min and max throughput and a target utilisation

Some Stats - Jan 2017

Total Tables: 2,045
Total Storage: 283.42 GiB
Number of Items: 1,000,374,288
Total Provisioned Throughput (Reads): 49,850
Total Provisioned Throughput (Writes): 20,210

Some Stats - Jun 2017

Total Tables: 3,073
Total Storage: 1.78 TiB
Number of Items: 2,816,558,766
Total Provisioned Throughput (Reads): 63,421
Total Provisioned Throughput (Writes): 32,569

Looking to the Future

What additional features do we want

Backup

Despite DynamoDB being a managed service we still have a desire to backup our data

Highly available services don’t protect you from user error

Having a second copy of live databases makes everyone feel confident

Introducing Hippolyte

Hippolyte is an at-scale DynamoDB backup solution

Designed for point-in-time backups of hundreds of tables or more

Scales capacity up to complete as fast as needed.

Available Now

Github: https://github.com/ocadotechnology/hippolyte

Blog Post: http://hippolyte.oca.do

Thank you

Recent Announcements

Amazon DynamoDB now Supports Cost Allocation Tags

AWS DMS Adds Support for MongoDB and Amazon DynamoDB

Announcing VPC Endpoints for Amazon DynamoDB (Public Preview)

Announcing Amazon DynamoDB Auto Scaling

Amazon DynamoDB AutoScaling

Automate capacity management for tables and GSIs

Enabled by default for all new tables and indexes

Can manually configure for existing tables and indexes

Requires DynamoDBAutoScale Role

Full CLI / API Support

No additional cost beyond existing DynamoDB and Cloudwatch alarms

Amazon DynamoDB Resources

Amazon Dynamo FAQ: https://aws.amazon.com/dynamodb/faqs/

Amazon DynamoDB Documentation (includes Developers Guide):

https://aws.amazon.com/documentation/dynamodb/

AWS Blog: https://aws.amazon.com/blogs/aws/

AWS

SUMMIT

Thank you!

amazon web services