
AWS
re:Invent
DECEMBER 2 – 6, 2024 | LAS VEGAS, NV

DAT404

Advanced data modeling with Amazon DynamoDB

Alex DeBrie

AWS Data Hero
Principal
DeBrie Advisory

Previously at re:Invent

- 2023: Airline reservations (complex filtering, maintaining constraints)
- 2022: MMORPG (architecture, constraints, transactions)
- 2021: Ecommerce (primary keys, write operations)

Related talks

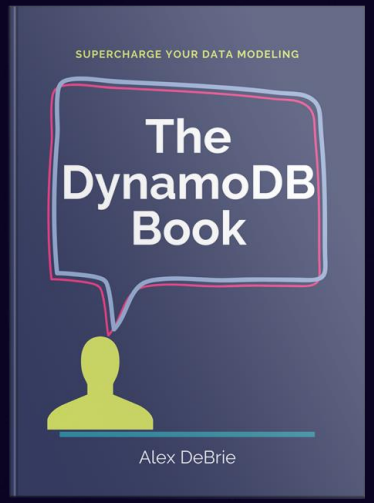
- DAT305: Data modeling core concepts for Amazon DynamoDB
- DAT406: Deep dive into Amazon DynamoDB
- DAT419: An insider’s look into architecture choices for Amazon DynamoDB

Agenda

- Background
- Data modeling basics
- DynamoDB + napkin math
- Using Amazon DynamoDB Streams
- Make it Dynamo

Alex DeBrie

- AWS Data Hero
- Independent consultant
- Author, *The DynamoDB Book*



dynamodbbook.com

DynamoDB background

“

Mechanical sympathy is when you use a tool or system with an understanding of how it operates best.

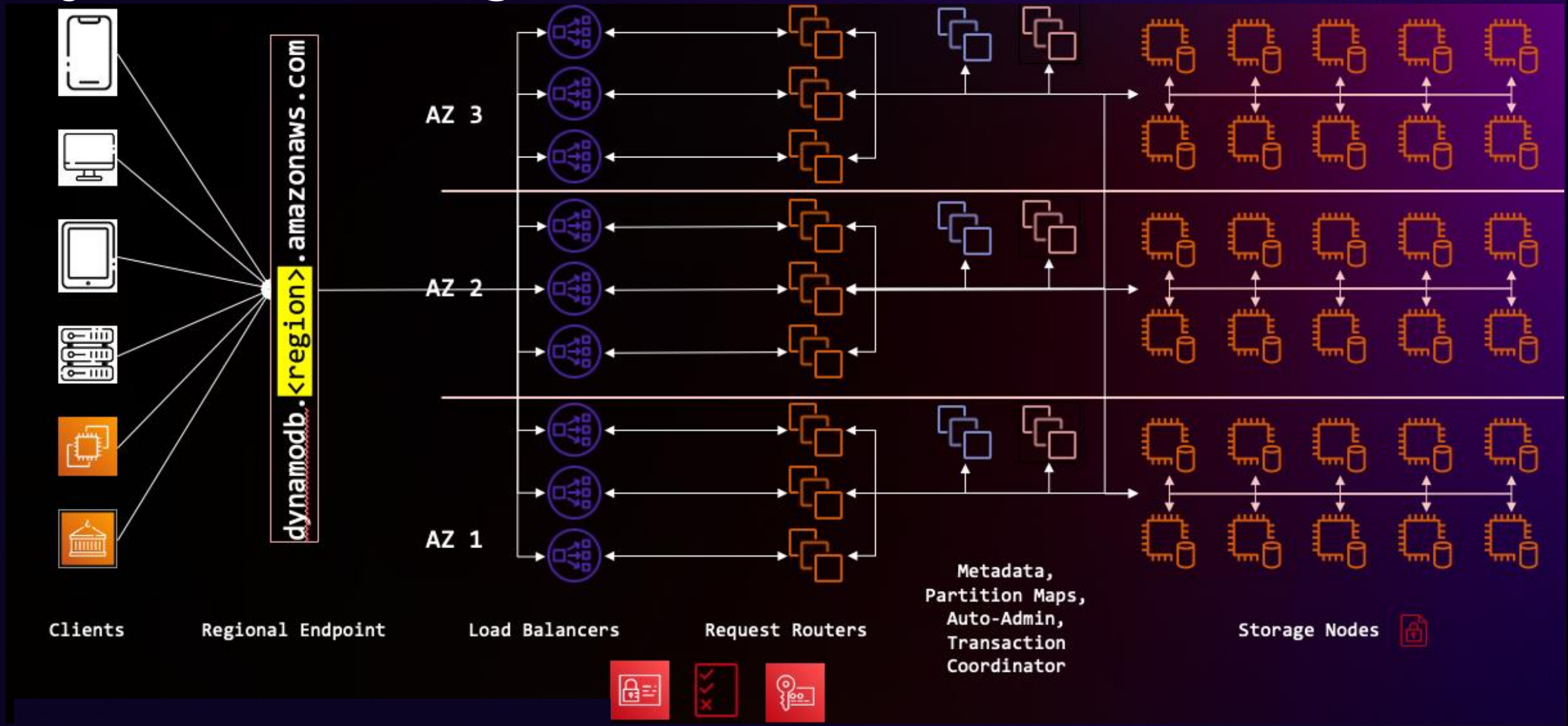
AWS Well-Architected Framework

https://wa.aws.amazon.com/wellarchitected/2020-07-02T19-33-23/wat.concept.mechanical-sympathy.en.html

Key characteristics

- Fully managed and proprietary to AWS

DynamoDB – High level architecture



Key characteristics

- Fully managed and proprietary to AWS

Key characteristics

- Fully managed and proprietary to AWS
- Consistent performance at any scale

Key characteristics

- Fully managed and proprietary to AWS
- Consistent performance at any scale
- Serverless-friendly

Key characteristics

- Fully managed and proprietary to AWS
- **Consistent performance at any scale**
- Serverless-friendly

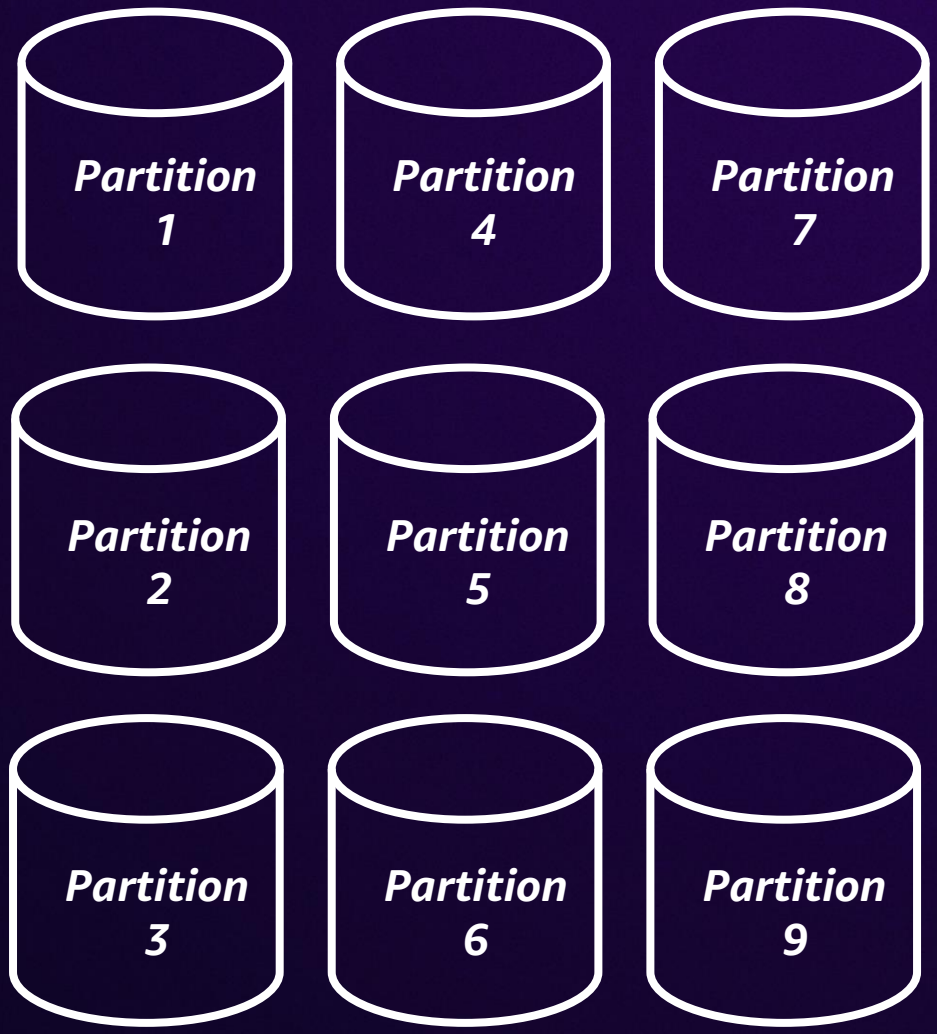
Primary key

Primary key Partition key: Username	Attributes			
alexdebrie	FirstName	LastName	UserPreferences	TeamName
	Alex	DeBrie	{ "darkMode": true, "timezone": "America/Chicago" }	AWS Heroes
boss_matt	FirstName	LastName	UserPreferences	TeamName
	Matt	Garman	{ "isAdmin": true, "timezone": "America/Los_Angeles" }	S-Team
product_joe	FirstName	LastName	UserPreferences	TeamName
	Joe	Idziorek	{ "darkMode": true, "timezone": "America/Los_Angeles" }	DynamoDB
algo_amrith	FirstName	LastName	UserPreferences	TeamName
	Amrith	Kumar	{ "timezone": "America/New_York" }	DynamoDB

PutItem:
Username:"alexdebrie.."
FirstName: "Alex"

fx(Username):
This item belongs to Partition 1

Constant time



Primary key

All access through primary key

Primary key Partition key: Username	Attributes			
alexdebrie	FirstName	LastName	UserPreferences	TeamName
	Alex	DeBrie	{ "darkMode": true, "timezone": "America/Chicago" }	AWS Heroes
boss_matt	FirstName	LastName	UserPreferences	TeamName
	Matt	Garman	{ "isAdmin": true, "timezone": "America/Los_Angeles" }	S-Team
product_joe	FirstName	LastName	UserPreferences	TeamName
	Joe	Idziorek	{ "darkMode": true, "timezone": "America/Los_Angeles" }	DynamoDB
algo_amrith	FirstName	LastName	UserPreferences	TeamName
	Amrith	Kumar	{ "timezone": "America/New_York" }	DynamoDB

Primary key

Primary key		Attributes		
Partition key: Username	**Sort key: OrderId**			
Grouped by partition key	Ordered by sort key	OrderCreatedAt	OrderStatus	OrderAmount
alexdebrie	01HNZNG5QDMB4014EW20R54VY4	2024-02-06T16:54:55.981Z	Cancelled	34.99
	01JCF6Z3ATX2RXTE6TDKXT111Y	OrderCreatedAt	OrderStatus	OrderAmount
	01JV7M94CQEBGNB263Z7X48S7R	2024-11-12T03:34:07.450Z	Delivered	172.14
		OrderCreatedAt	OrderStatus	OrderAmount
		2025-05-14T14:48:19.607Z	Placed	94.35
product_joe	01K5RM9988AH0P7ZYHPAK2S3FF	OrderCreatedAt	OrderStatus	OrderAmount
		2025-09-22T11:52:28.168Z	Delivered	237.44
algo_amrith	01J2YKVT6S0CZBDBX5YCZXV05E	OrderCreatedAt	OrderStatus	OrderAmount
		2024-07-16T20:31:09.529Z	Delivered	311.64

Partitioning + the DynamoDB API

- Items spread across partitions by partition key
- Single-item actions
  - Basic CRUD – PutItem, GetItem, UpdateItem, DeleteItem
  - Requires full primary key
  - All write operations
- Query operation (composite primary key only)
  - Fetch many
  - Requires partition key; sort key optional
- Scan
  - Fetch all (use sparingly)

Primary key

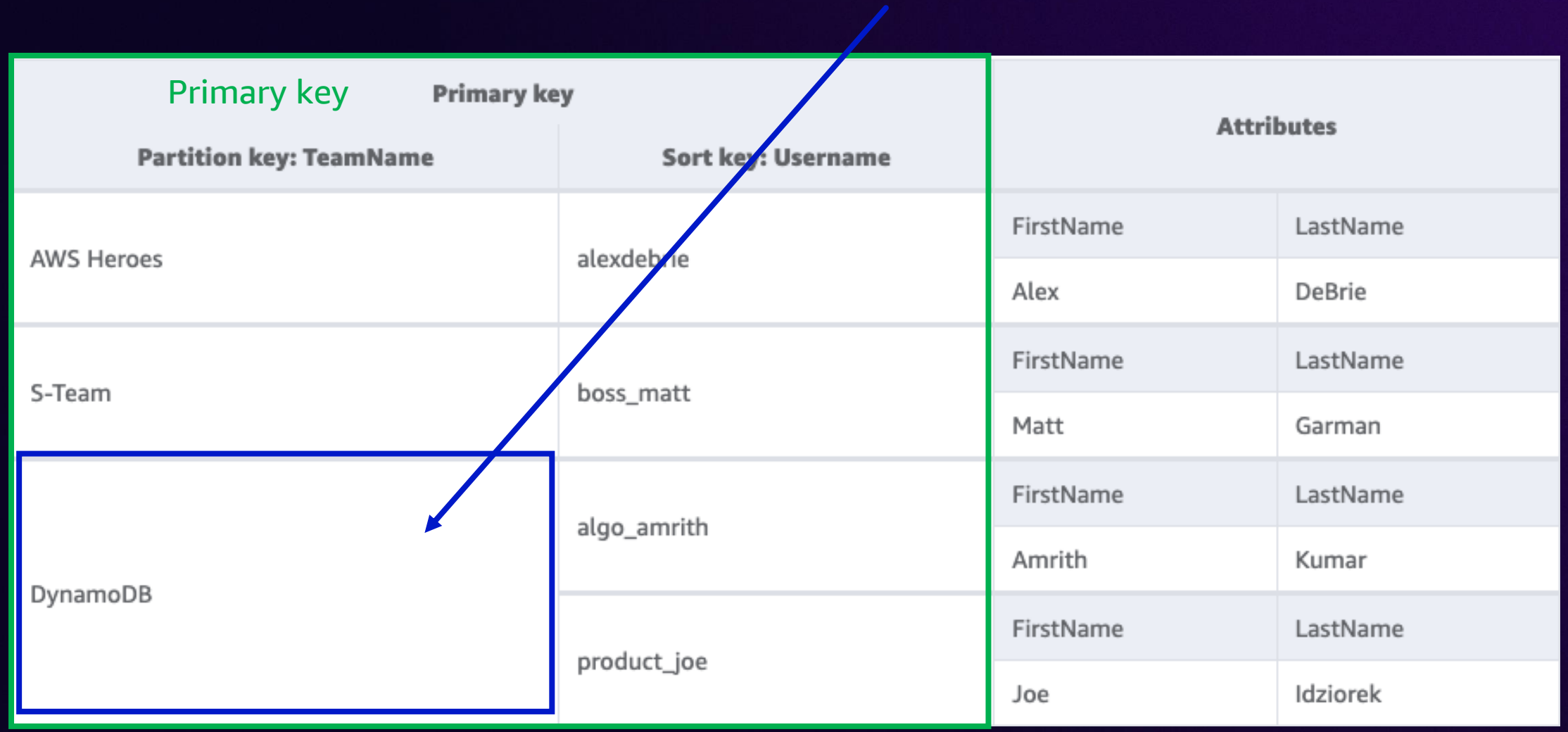
All access through primary key

Primary key Partition key: Username	Attributes			
alexdebrie	FirstName	LastName	UserPreferences	TeamName
	Alex	DeBrie	{ "darkMode": true, "timezone": "America/Chicago" }	AWS Heroes
boss_matt	FirstName	LastName	UserPreferences	TeamName
	Matt	Garman	{ "isAdmin": true, "timezone": "America/Los_Angeles" }	S-Team
product_joe	FirstName	LastName	UserPreferences	TeamName
	Joe	Idziorek	{ "darkMode": true, "timezone": "America/Los_Angeles" }	DynamoDB
algo_amrith	FirstName	LastName	UserPreferences	TeamName
	Amrith	Kumar	{ "timezone": "America/New_York" }	DynamoDB

All access through primary key

Primary key

Primary key Partition key: Username	Attributes			
alexdebrie	FirstName	LastName	UserPreferences	TeamName
	Alex	DeBrie	{ "darkMode": true, "timezone": "America/Chicago" }	AWS Heroes
boss_matt	FirstName	LastName	UserPreferences	TeamName
	Matt	Garman	{ "isAdmin": true, "timezone": "America/Los_Angeles" }	S-Team
product_joe	FirstName	LastName	UserPreferences	TeamName
	Joe	Idziorek	{ "darkMode": true, "timezone": "America/Los_Angeles" }	DynamoDB
algo_amrith	FirstName	LastName	UserPreferences	TeamName
	Amrith	Kumar	{ "timezone": "America/New_York" }	DynamoDB



Secondary indexes

- Fully managed copies of your data
- Enable additional read-based access patterns
- Two kinds:
  - Global secondary indexes (prefer)
  - Local secondary indexes (understand before using!)

What you don’t need to know about DynamoDB

- Paxos vs. Raft
- Two-phase vs. three-phrase commit
- Memory buffer configuration settings

What you do need to know about DynamoDB

- Partitions + importance of primary key
- API structure
  - Single-item actions vs. query vs. scan
- Secondary indexes
- Billing
- Limits
- Pagination mechanics
- Consistency model

Data modeling basics

First, understand your needs

Before you design your data model

- Know your domain
  - What are your constraints?
  - What's your data distribution?
  - How big are your items?
- Know your access patterns

Access patterns

Write access patterns

Write pattern	Item(s) altered	Condition(s)	Frequency	Notes
Create User				
Increase Gold for User				
Add Inventory to User				
Remove Inventory for User				
Add User to Guild				
Create Quest for User				
Mark Quest Completed for User				

Read access patterns

Pattern	Operation	Target	Filters / Projections	Notes
Get User				
Fetch Inventory for User				
Fetch Users by Guild				
Get Quest Details by User and ID				
Fetch Quests for User				
Fetch Completed Quests for Users				

Before you design your data model

- Know your domain
  - What are your constraints?
  - What's your data distribution?
  - How big are your items?
- Know your access patterns

Before you design your data model

- Know your domain
  - What are your constraints?
  - What's your data distribution?
  - How big are your items?
- Know your access patterns
- Know the DynamoDB basics

Before you design your data model

- Know your domain
  - What are your constraints?
  - What's your data distribution?
  - How big are your items?
- Know your access patterns
- Know the DynamoDB basics
  - Primary key + API + secondary indexes

Then, design your table for your needs

Use the basics!

- Single-item actions

Read access patterns

Pattern	Operation	Target	Filters / Projections	Notes
Get User				
Fetch Inventory for User				
Fetch Users by Guild				
Get Quest Details by User and ID				
Fetch Quests for User				
Fetch Completed Quests for Users				

Use the basics!

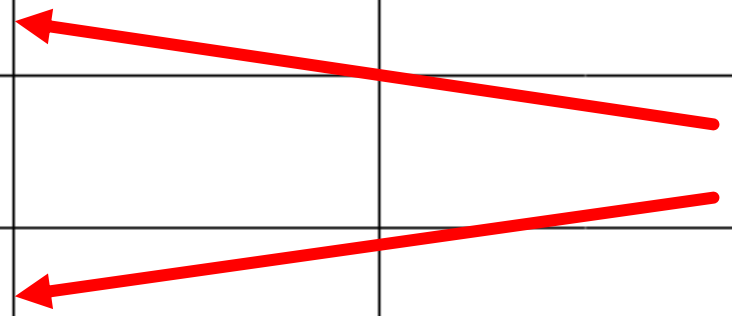
- Single-item actions

Use the basics!

- Single-item actions
- Query for "List" operations

Read access patterns

Pattern	Operation	Target	Filters / Projections	Notes
Get User				
Fetch Inventory for User				
Fetch Users by Guild				
Get Quest Details by User and ID				
Fetch Quests for User				
Fetch Completed Quests for Users				



Use the basics!

- Single-item actions
- Query for "List" operations

Use the basics!

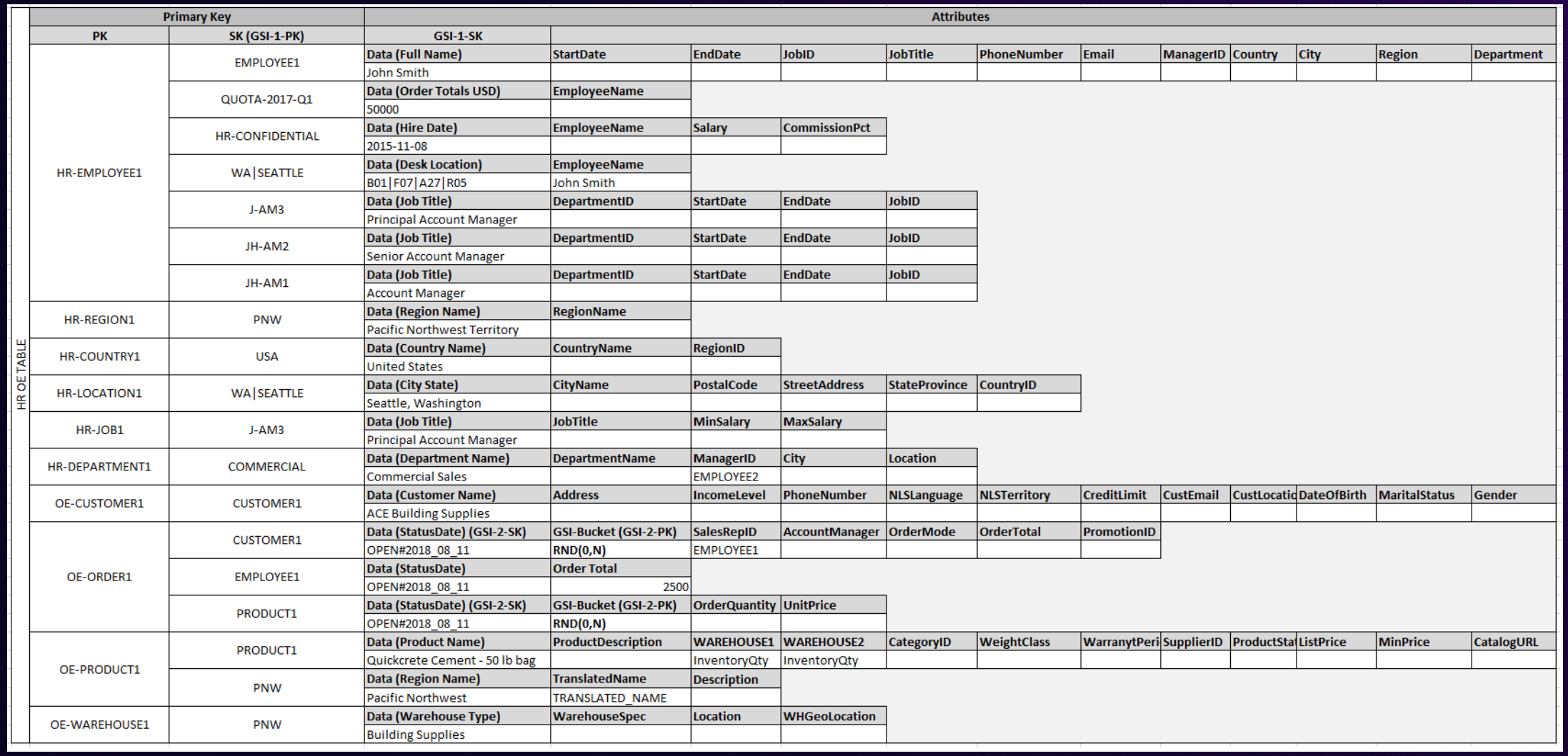
- Single-item actions
- Query for "List" operations
- Secondary indexes for additional read-based patterns

Use the basics!

- Single-item actions
- Query for "List" operations
- Secondary indexes for additional read-based patterns
- Transactions

Flexibility vs. efficiency

- Why are you using DynamoDB?
  - Predictable performance at enormous scale?
  - Migrating a legacy application to DynamoDB?



HR OE TABLE

Primary Key		Attributes											
PK	SK (GSI-1-PK)	GSI-1-SK											
HR-EMPLOYEE1	EMPLOYEE1	Data (Full Name)	StartDate	EndDate	JobID	JobTitle	PhoneNumber	Email	ManagerID	Country	City	Region	Department
		John Smith											
	QUOTA-2017-Q1	Data (Order Totals USD)	EmployeeName										
		50000											
	HR-CONFIDENTIAL	Data (Hire Date)	EmployeeName	Salary	CommissionPct								
		2015-11-08											
	WA|SEATTLE	Data (Desk Location)	EmployeeName										
		B01|F07|A27|R05	John Smith									
	J-AM3	Data (Job Title)	DepartmentID	StartDate	EndDate	JobID							
		Principal Account Manager											
	JH-AM2	Data (Job Title)	DepartmentID	StartDate	EndDate	JobID							
		Senior Account Manager											
	JH-AM1	Data (Job Title)	DepartmentID	StartDate	EndDate	JobID							
		Account Manager											
HR-REGION1	PNW	Data (Region Name)	RegionName										
		Pacific Northwest Territory											
HR-COUNTRY1	USA	Data (Country Name)	CountryName	RegionID									
		United States											
HR-LOCATION1	WA|SEATTLE	Data (City State)	CityName	PostalCode	StreetAddress	StateProvince	CountryID						
		Seattle, Washington											
HR-JOB1	J-AM3	Data (Job Title)	JobTitle	MinSalary	MaxSalary								
		Principal Account Manager											
HR-DEPARTMENT1	COMMERCIAL	Data (Department Name)	DepartmentName	ManagerID	City	Location							
		Commercial Sales		EMPLOYEE2									
OE-CUSTOMER1	CUSTOMER1	Data (Customer Name)	Address	IncomeLevel	PhoneNumber	NLSLanguage	NLSTerritory	CreditLimit	CustEmail	CustLocatic	DateOfBirth	MaritalStatus	Gender
		ACE Building Supplies											
OE-ORDER1	CUSTOMER1	Data (StatusDate) (GSI-2-SK)	GSI-Bucket (GSI-2-PK)	SalesRepID	AccountManager	OrderMode	OrderTotal	PromotionID					
		OPEN#2018_08_11	RND(0,N)	EMPLOYEE1									
	EMPLOYEE1	Data (StatusDate)	Order Total										
		OPEN#2018_08_11	2500										
	PRODUCT1	Data (StatusDate) (GSI-2-SK)	GSI-Bucket (GSI-2-PK)	OrderQuantity	UnitPrice								
		OPEN#2018_08_11	RND(0,N)										
OE-PRODUCT1	PRODUCT1	Data (Product Name)	ProductDescription	WAREHOUSE1	WAREHOUSE2	CategoryID	WeightClass	WarranytPeri	SupplierID	ProductSta	ListPrice	MinPrice	CatalogURL
		Quickcrete Cement - 50 lb bag		InventoryQty	InventoryQty								
	PNW	Data (Region Name)	TranslatedName	Description									
		Pacific Northwest	TRANSLATED_NAME										
OE-WAREHOUSE1	PNW	Data (Warehouse Type)	WarehouseSpec	Location	WHGeoLocation								
		Building Supplies											

Example of modeling relational data in DynamoDB

https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-modeling-nosql-B.html

Flexibility vs. efficiency

- Why are you using DynamoDB?
  - Predictable performance at enormous scale?
  - Migrating a legacy application to DynamoDB?

Flexibility vs. efficiency

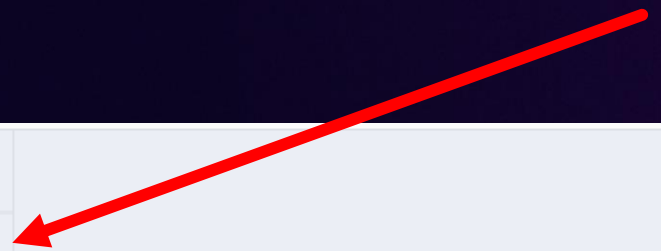
- Why are you using DynamoDB?
  - Predictable performance at enormous scale?
  - Migrating a legacy application to DynamoDB?
  - Integration with AWS AppSync/GraphQL?

Flexibility vs. efficiency

- Why are you using DynamoDB?
  - Predictable performance at enormous scale?
  - Migrating a legacy application to DynamoDB?
  - Integration with AWS AppSync/GraphQL?
  - Ease of use in serverless architecture?

Tips for all modeling styles

- Design for your access patterns
  - Use meaningful primary keys



Primary key	Attributes				
Partition key: Username					
amazing_amrith	Class	Gold	Inventory	TotalPlayTime	Guild
	Mage	1452	[{ "Type": "Weapon", "Name": "Sword of Partitioning" } ....]	892341	DynamoDestroyers
astute_alex	Class	Gold	Inventory	TotalPlayTime	Guild
	Bard	247	[{ "Type": "Armor", "Name": "Shield of Sharding" } ...]	629831	HelpingHeroes
jumping_janelle	Class	Gold	Inventory	TotalPlayTime	Guild
	Paladin	391	[{ "Type": "Weapon", "Name": "Arrow of Availability" } ...]	23483	DynamoDestroyers
calming_chad	Class	Gold	Inventory	TotalPlayTime	Guild
	Priest	12	[{ "Type": "Potion", "Name": "Potion of Understanding" } ...]	42786	DynamoDestroyers

Tips for all modeling styles

- Design for your access patterns
  - Use meaningful primary keys

Tips for all modeling styles

- Design for your access patterns
  - Use meaningful primary keys
  - Don’t needlessly overload item collections

Primary key		Attributes			
Partition key: PK	Sort key: SK				
astute_alex	FRIENDSHIP#amazing_amrith	FriendshipCreatedAt			
		2023-04-12 12:35:32			
	FRIENDSHIP#jumping_janelle	FriendshipCreatedAt			
		2023-06-19 15:44:21			
	INVENTORY	Inventory			
		[{ "Type": "Armor", "Name": "Shield of Sharding" }, ...]			
	QUEST#01GGFG8CCVEE7C59EV77MN5892	QuestName	QuestStartedAt		
		A Lost Cause	2022-08-22 19:19:12		
	QUEST#01GGFG9QH4B0DP04CYP9XWBD7G	QuestName	QuestStartedAt	QuestCompletedAt	
		Sole Survivor	2022-05-18-21:09:20	2022-05-21 19:33:41	
	QUEST#01GGFGCNPES703VR9396PBR0X0	QuestName	QuestStartedAt	QuestCompletedAt	
		Answer the Call	2022-10-19 20:30:14	2022-10-20 22:16:24	
	USER	Class	Gold	TotalPlayTime	Guild
		Bard	247	629831	HelpingHeroes

Friendships

Inventory

Quests

User

Primary key		Attributes			
Partition key: PK	Sort key: SK				
astute_alex	INVENTORY	Inventory	User + Inventory		
		[{ "Type": "Armor", "Name": "Shield of Sharding" }, ...]			
	USER	Class	Gold	TotalPlayTime	Guild
		Bard	247	629831	HelpingHeroes
astute_alex#QUEST	01GGFG8CCVEE7C59EV77MN5892	QuestName	QuestStartedAt		
		A Lost Cause	2022-08-22 19:19:12		
	01GGFG9QH4B0DP04CYP9XWBD7G	QuestName	QuestStartedAt	QuestCompletedAt	
		Sole Survivor	2022-05-18-21:09:20	2022-05-21 19:33:41	
	01GGFGCNPES703VR9396PBR0X0	QuestName	QuestStartedAt	QuestCompletedAt	
		Answer the Call	2022-10-19 20:30:14	2022-10-20 22:16:24	
astute_alex#FRIENDSHIP	amazing_amrith	FriendshipCreatedAt			
		2023-04-12 12:35:32			
	jumping_janelle	FriendshipCreatedAt			
		2023-06-19 15:44:21			

Quests

Friendships

Tips for all modeling styles

- Design for your access patterns
  - Use meaningful primary keys
  - Don’t needlessly overload item collections

Tips for all modeling styles

- Design for your access patterns
  - Use meaningful primary keys
  - Don’t needlessly overload item collections
- Think about your writes early

Tips for all modeling styles

- Design for your access patterns
  - Use meaningful primary keys
  - Don’t needlessly overload item collections
- Think about your writes early
  - Use conditional writes

Tips for all modeling styles

- Design for your access patterns
  - Use meaningful primary keys
  - Don’t needlessly overload item collections
- Think about your writes early
  - Use conditional writes
- Flatten hierarchies

Tips for all modeling styles

- Design for your access patterns
  - Use meaningful primary keys
  - Don’t needlessly overload item collections
- Think about your writes early
  - Use conditional writes
- Flatten hierarchies
  - Denormalize where prudent

Denormalization

- Embedding
  - ✅ One-to-one or limited one-to-many relationships

Primary key Partition key: Username	Attributes			
alexdebrie	FirstName	LastName	UserPreferences	TeamName
	Alex	DeBrie	{ "darkMode": true, "timezone": "America/Chicago" }	AWS Heroes
boss_matt	FirstName	LastName	UserPreferences	TeamName
	Matt	Garman	{ "isAdmin": true, "timezone": "America/Los_Angeles" }	S-Team
product_joe	FirstName	LastName	UserPreferences	TeamName
	Joe	Idziorek	{ "darkMode": true, "timezone": "America/Los_Angeles" }	DynamoDB
algo_amrith	FirstName	LastName	UserPreferences	TeamName
	Amrith	Kumar	{ "timezone": "America/New_York" }	DynamoDB

Denormalization

- Embedding
  - ✅ One-to-one or limited one-to-many relationships

Denormalization

- Embedding
  - ✅ One-to-one or limited one-to-many relationships
  - 🚫 Unbounded one-to-many

Primary key Partition key: Username	Attributes				
alexdebrie	FirstName	LastName	UserPreferences	TeamName	Orders
	Alex	DeBrie	{ "darkMode": true, "timezone": "America/Chicago" }	AWS Heroes	[{ "OrderId": "01HN....", "Amount": "34.99" } ...]
boss_matt	FirstName	LastName	UserPreferences	TeamName	
	Matt	Garman	{ "isAdmin": true, "timezone": "America/Los_Angeles" }	S-Team	
product_joe	FirstName	LastName	UserPreferences	TeamName	Orders
	Joe	Idziorek	{ "darkMode": true, "timezone": "America/Los_Angeles" }	DynamoDB	[{ "OrderId": "01K5....", "Amount": "237.44" } ...]
algo_amrith	FirstName	LastName	UserPreferences	TeamName	Orders
	Amrith	Kumar	{ "timezone": "America/New_York" }	DynamoDB	[{ "OrderId": "01J2....", "Amount": "311.64" } ...]

Denormalization

- Embedding
  - ✅ One-to-one or limited one-to-many relationships
  - 🚫 Unbounded one-to-many

Denormalization

- Embedding
  - ✅ One-to-one or limited one-to-many relationships
  - 🚫 Unbounded one-to-many
  - Key tradeoff: Item size vs. multiple reads

Denormalization

- Embedding
  - ✅ One-to-one or limited one-to-many relationships
  - 🚫 Unbounded one-to-many
  - Key tradeoff: Item size vs. multiple reads
- Duplication

Denormalization

- Embedding
  - ✅ One-to-one or limited one-to-many relationships
  - 🚫 Unbounded one-to-many
  - Key tradeoff: Item size vs. multiple reads
- Duplication
  - Key tradeoff: Faster reads vs. harder / more expensive writes

Denormalization

- Embedding
  - ✅ One-to-one or limited one-to-many relationships
  - 🚫 Unbounded one-to-many
  - Key tradeoff: Item size vs. multiple reads
- Duplication
  - Key tradeoff: Faster reads vs. harder / more expensive writes
    - Ideal: immutable values

Denormalization

- Embedding
  - ✅ One-to-one or limited one-to-many relationships
  - 🚫 Unbounded one-to-many
  - Key tradeoff: Item size vs. multiple reads
- Duplication
  - Key tradeoff: Faster reads vs. harder / more expensive writes
    - Ideal: immutable values
  - Benefits:

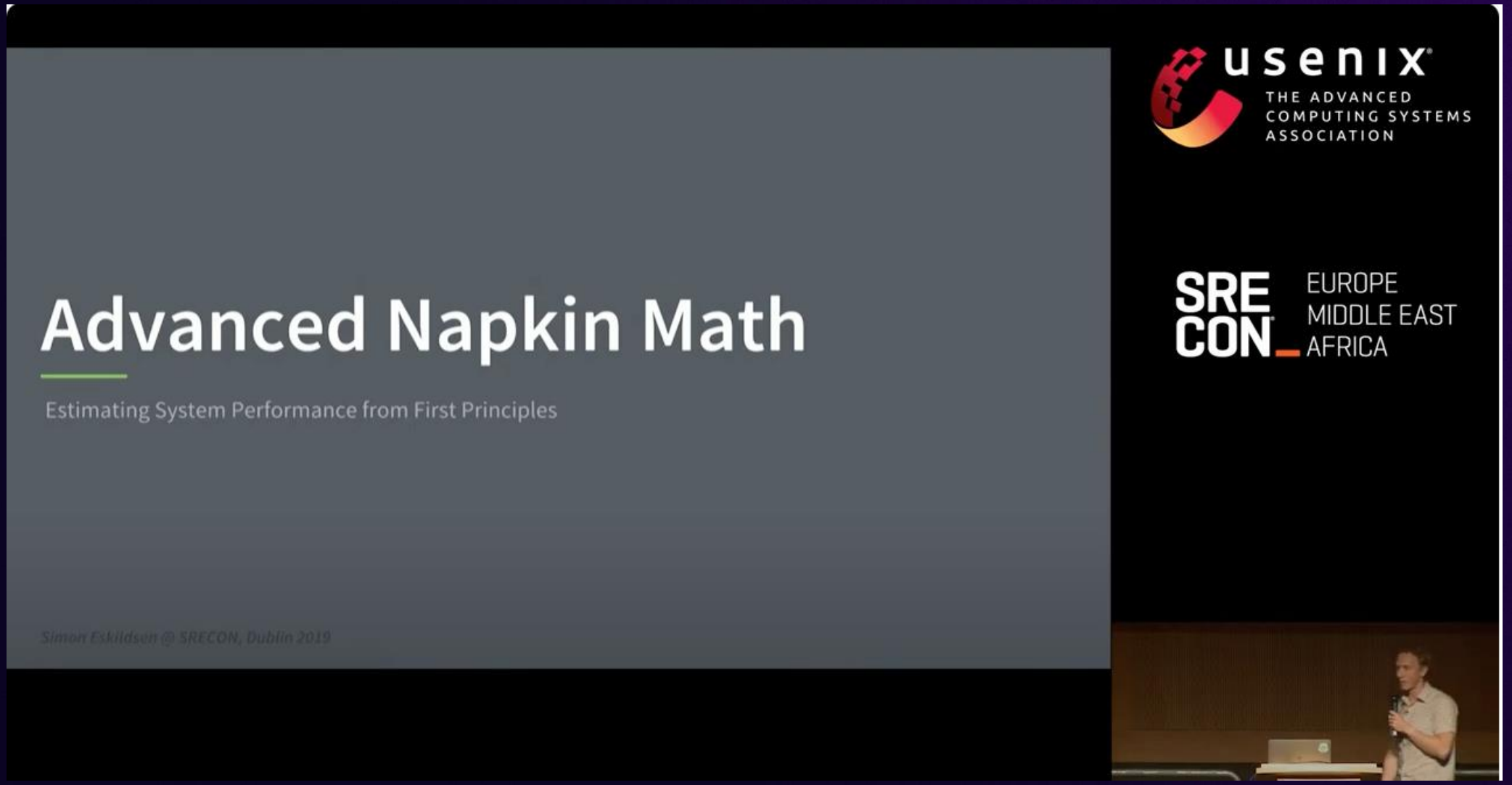
Denormalization

- Embedding
  - ✅ One-to-one or limited one-to-many relationships
  - 🚫 Unbounded one-to-many
  - Key tradeoff: Item size vs. multiple reads
- Duplication
  - Key tradeoff: Faster reads vs. harder / more expensive writes
    - Ideal: immutable values
  - Benefits:
    - Reducing number of reads (faster + cheaper)

Denormalization

- Embedding
  - ✅ One-to-one or limited one-to-many relationships
  - 🚫 Unbounded one-to-many
  - Key tradeoff: Item size vs. multiple reads
- Duplication
  - Key tradeoff: Faster reads vs. harder / more expensive writes
    - Ideal: immutable values
  - Benefits:
    - Reducing number of reads (faster + cheaper)
    - Moving reads from sequential to parallel (faster)

Napkin math and DynamoDB



Advanced Napkin Math
Simon Eskildsen, SRECON, Dublin 2019

Source: Simon Eskildsen, Advanced Napkin Math

Napkin math + DynamoDB base rates

- Performance

Performance base rates

- Client-side operation latency:
  - GetItem/Query: ~5 ms
  - PutItem: ~20 ms
  - Transaction: 100 ms
- Limits:
  - Query response: 1 MB per request
  - Hot items: 1000 WCU per second and 3000 RCU per second

Napkin math + DynamoDB base rates

- Performance

Napkin math + DynamoDB base rates

- Performance
- Billing

DynamoDB billing background

- Traditional databases: CPU, memory, IOPS
- DynamoDB:
  - Read Capacity Unit (RCU)
  - Write Capacity Unit (WCU)

Billing base rates

- Capacity units:
  - RCU: Up to 4 KB of data read
    - Cut in half if not strongly consistent read
  - WCU: Up to 1 KB of data written
- Prices:
  - 12.5¢ per 1 million RCUs
  - 67.5¢ per 1 million WCUs
  - $0.25 per GB-month

Billing base rates

- Capacity units:
  - RCU: Up to 4 KB of data read
    - Cut in half if not strongly consistent read
  - WCU: Up to 1 KB of data written
- Prices*:
  - 12.5¢ per 1 million RCUs
  - 67.5¢ per 1 million WCUs
  - $0.25 per GB-month

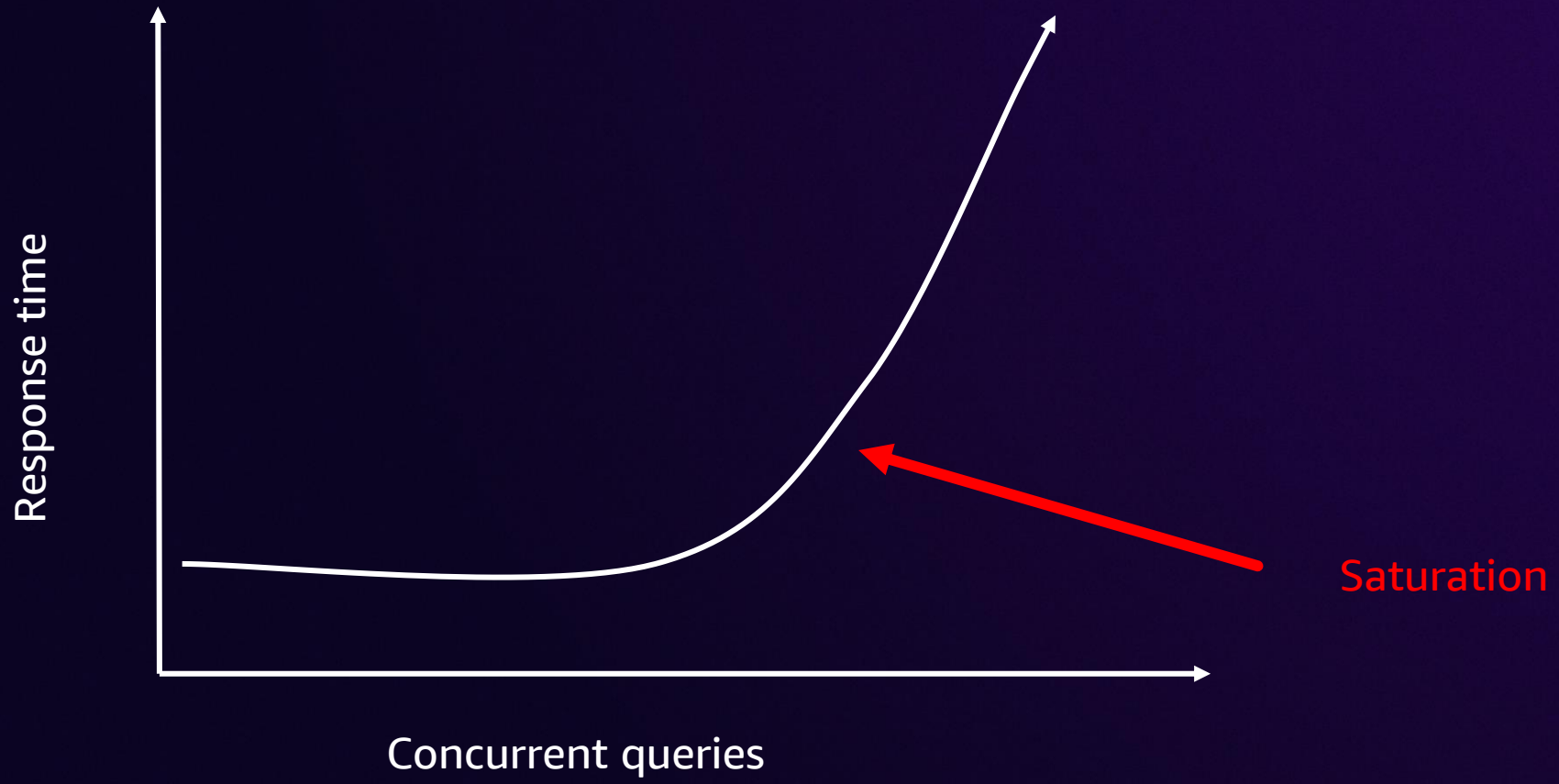
* us-east-1 on-demand numbers

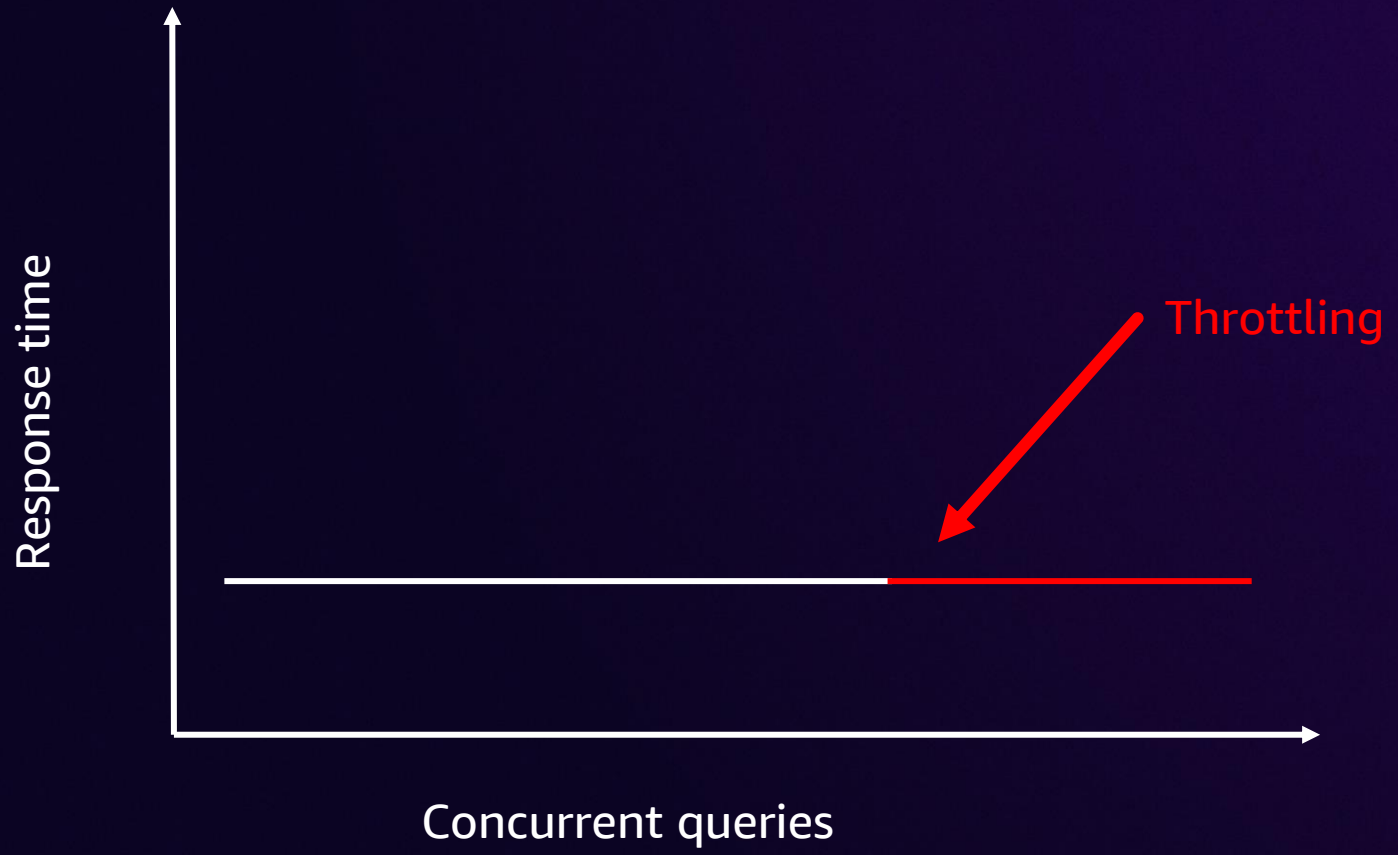
Napkin math + DynamoDB base rates

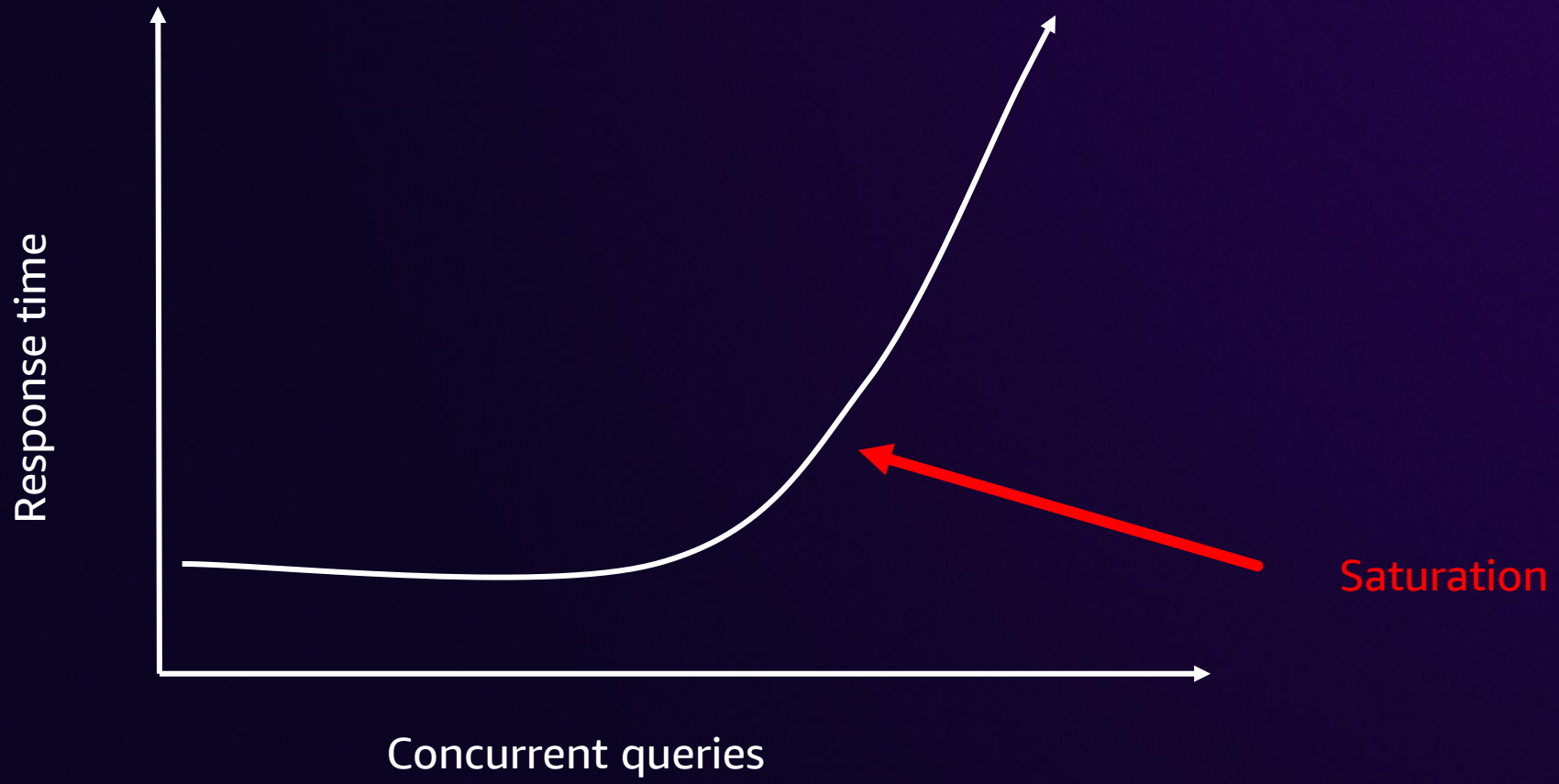
- Performance
- Billing

Napkin math + DynamoDB implications

- Performance + billing are separate concerns







Performance base rates

- Client-side operation latency:
  - GetItem/Query: ~5 ms
  - PutItem: ~20 ms
  - Transaction: 100 ms
- Limits:
  - Query response: 1 MB per request
  - Hot items: 1000 WCU per second and 3000 RCU per second

Napkin math + DynamoDB implications

- Performance + billing are separate concerns

Napkin math + DynamoDB implications

- Performance + billing are separate concerns
- DynamoDB pricing should affect how you build applications

Billing base rates

- Capacity units:
  - RCU: Up to 4 KB of data read (cut in half if not strongly consistent read)
  - WCU: Up to 1 KB of data written
- Prices*:
  - 12.5¢ per 1 million RCUs
  - 67.5¢ per 1 million WCUs
  - $0.25 per GB-month

* us-east-1 numbers

Advanced napkin math: WCU + RCU multipliers

- Item size ← Larger items require more resources
- Secondary indexes ← More writes
- Transactions ← Coordination
- Global tables ← Replication infra + conflict resolution
- Consistent read ← Requires routing to leader

Mind your multipliers!

Mind your multipliers

- Review item sizes carefully
  - Remove unused attributes
  - Compress large values (or send to Amazon S3)

Mind your multipliers

- Review item sizes carefully

Mind your multipliers

- Review item sizes carefully
- Limit secondary indexes

Mind your multipliers

- Review item sizes carefully
- Limit secondary indexes
  - Do you need a secondary index? Writes are ~20x more than reads

Mind your multipliers

- Review item sizes carefully
- Limit secondary indexes
  - Do you need a secondary index? Writes are ~20x more than reads
  - Use projections to limit size of items in the index

Mind your multipliers

- Review item sizes carefully
- Limit secondary indexes

Mind your multipliers

- Review item sizes carefully
- Limit secondary indexes
- Limit transactions

Mind your multipliers

- Review item sizes carefully
- Limit secondary indexes
- Limit transactions
- Ensure that you need global tables

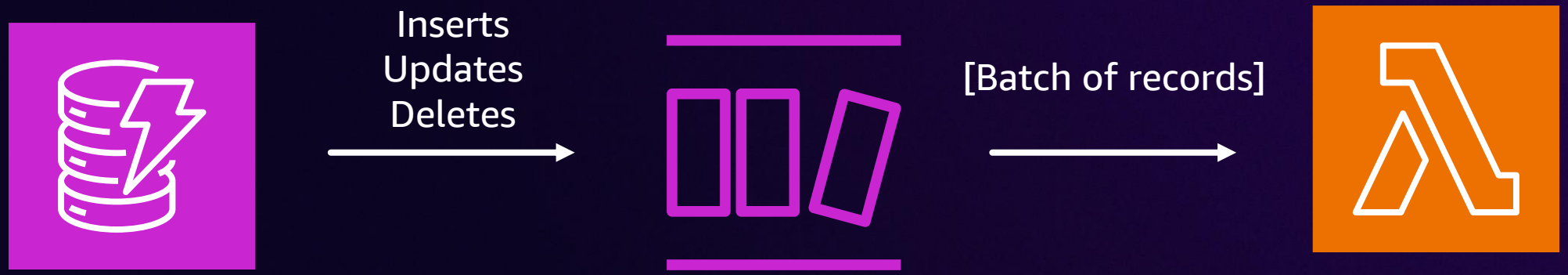
Mind your multipliers

- Review item sizes carefully
- Limit secondary indexes
- Limit transactions
- Ensure that you need global tables
- Don’t use consistent reads

Napkin math + DynamoDB implications

- Performance + billing are separate concerns
- DynamoDB pricing should affect how you build applications

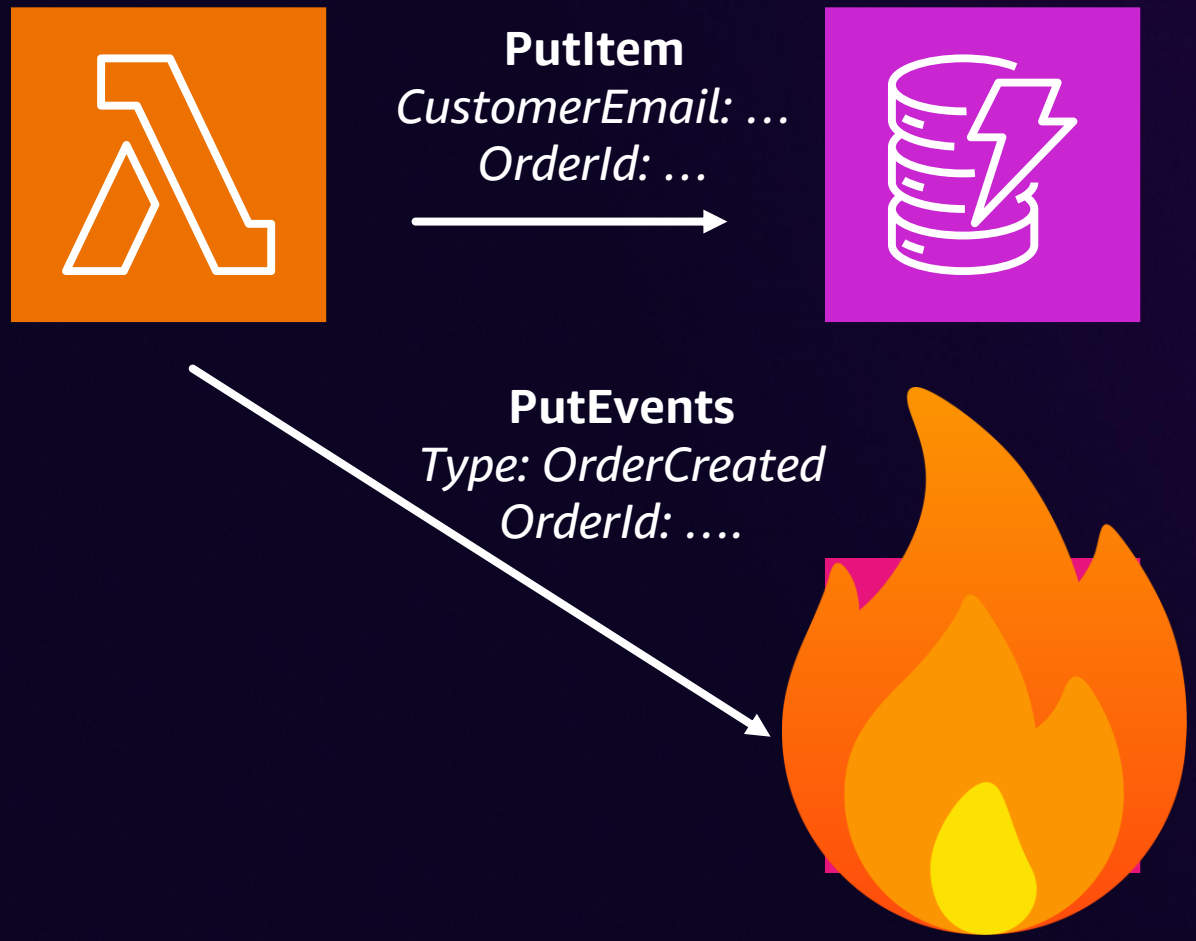
Amazon DynamoDB Streams



DynamoDB Streams use cases

- Solving the dual-write problem

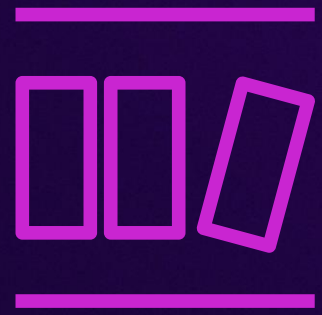
The dual-write problem



The dual-write problem

PutItem
CustomerEmail: …
OrderId: …

Inserts
Updates
Deletes



[Batch of records]

PutEvents
Type: OrderCreated
OrderId: ….

Tips for DynamoDB Streams

- Clean up your events before pushing elsewhere
  - Consumer stream limit
  - Raw event format

```
{
  "eventID":"c4ca4238a0b923820dcc509a6f75849b",
  "eventName": "INSERT",
  "eventVersion": "1.1",
  "eventSource": "aws:dynamodb",
  "awsRegion": "us-east-1",
  "dynamodb":
    {
     "ApproximateCreationDateTime": 1628899200,
     "Keys":
       { "id": { "S": "123456789" } },
       "NewImage": {
         "id": { "S": "123456789" },
         "name": { "S": "John Doe" },
         "email": { "S": "john.doe@example.com" },
         "createdAt": { "N": "1628899200" } },
       "SequenceNumber": "44215845000000000017...",
       "SizeBytes": 26,
       "StreamViewType": "NEW_AND_OLD_IMAGES" },
       "eventSourceARN": "arn:aws:dynamodb:..." }
    }
}
```

```
{
  "type": "USER_CREATED",
  "timestamp": 1628899200000,
  "version": "1.0",
  "data": {
    "userId": "123456789",
    "name": "John Doe",
    "email": "john.doe@example.com",
    "createdAt": 1628899200000 },
  },
  "metadata": {
    "source": "dynamodb",
    "region": "us-east-1",
    "streamId": "44215845000000000017450439091"
  }
}
```

Tips for DynamoDB Streams

- Clean up your events before pushing elsewhere
  - Consumer stream limit
  - Raw event format

Tips for DynamoDB Streams

- Clean up your events before pushing elsewhere
  - Consumer stream limit
  - Raw event format
- Understand stream processing mechanics + failure modes

Tips for DynamoDB Streams

- Clean up your events before pushing elsewhere
  - Consumer stream limit
  - Raw event format
- Understand stream processing mechanics + failure modes
  - Kafka/Amazon Kinesis

Tips for DynamoDB Streams

- Clean up your events before pushing elsewhere
  - Consumer stream limit
  - Raw event format
- Understand stream processing mechanics + failure modes
  - Kafka/Amazon Kinesis
- Consider your stream implementation carefully

Tips for DynamoDB Streams

- Clean up your events before pushing elsewhere
  - Consumer stream limit
  - Raw event format
- Understand stream processing mechanics + failure modes
  - Kafka/Amazon Kinesis
- Consider your stream implementation carefully
  - DynamoDB Streams: fully managed, better guarantees, but max 2 consumers

Tips for DynamoDB Streams

- Clean up your events before pushing elsewhere
  - Consumer stream limit
  - Raw event format
- Understand stream processing mechanics + failure modes
  - Kafka/Amazon Kinesis
- Consider your stream implementation carefully
  - DynamoDB Streams: fully managed, better guarantees, but max 2 consumers
  - Kinesis Streams: more consumers but more management + cost

DynamoDB Streams use cases

- Solving the dual-write problem

DynamoDB Streams use cases

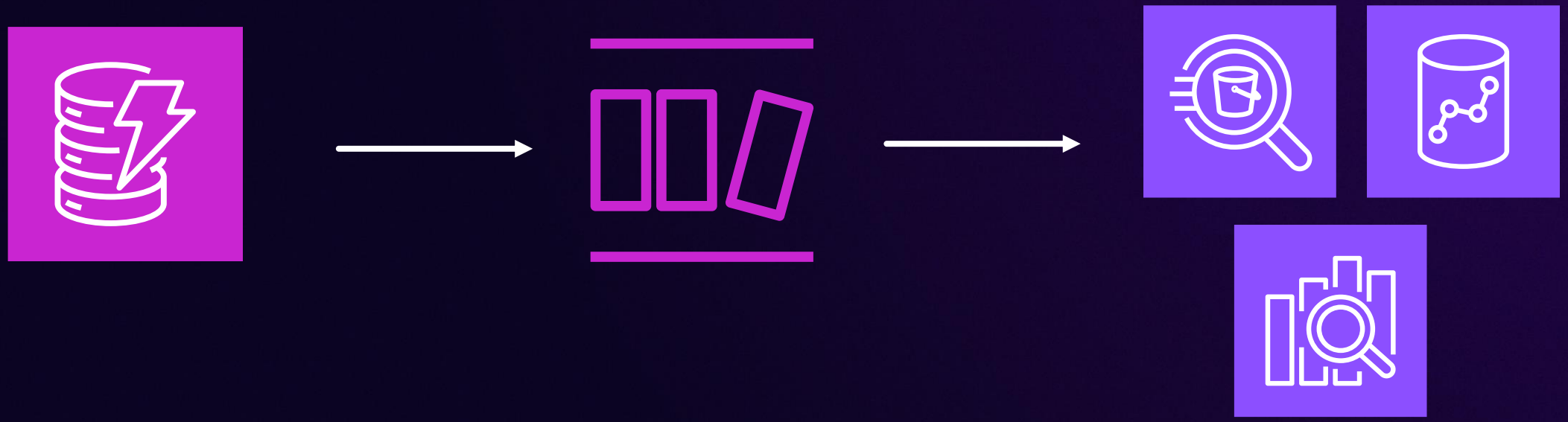
- Solving the dual-write problem
- Exporting to secondary system

Good at:

- Classic OLTP workloads
  - Small reads + writes
  - Transactions
  - Conditional writes
  - Low latency/high availability

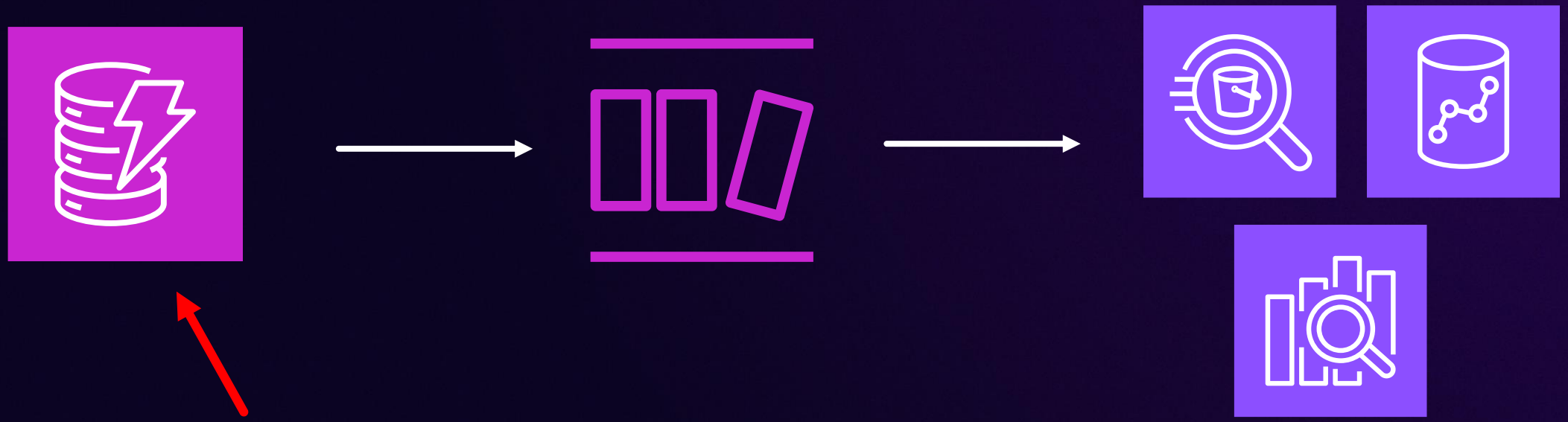
Bad at:

- Flexible workloads
  - Analytics/aggregations
  - Complex filtering
  - Full-text search



Complexities with syncing to external system

- Initial export



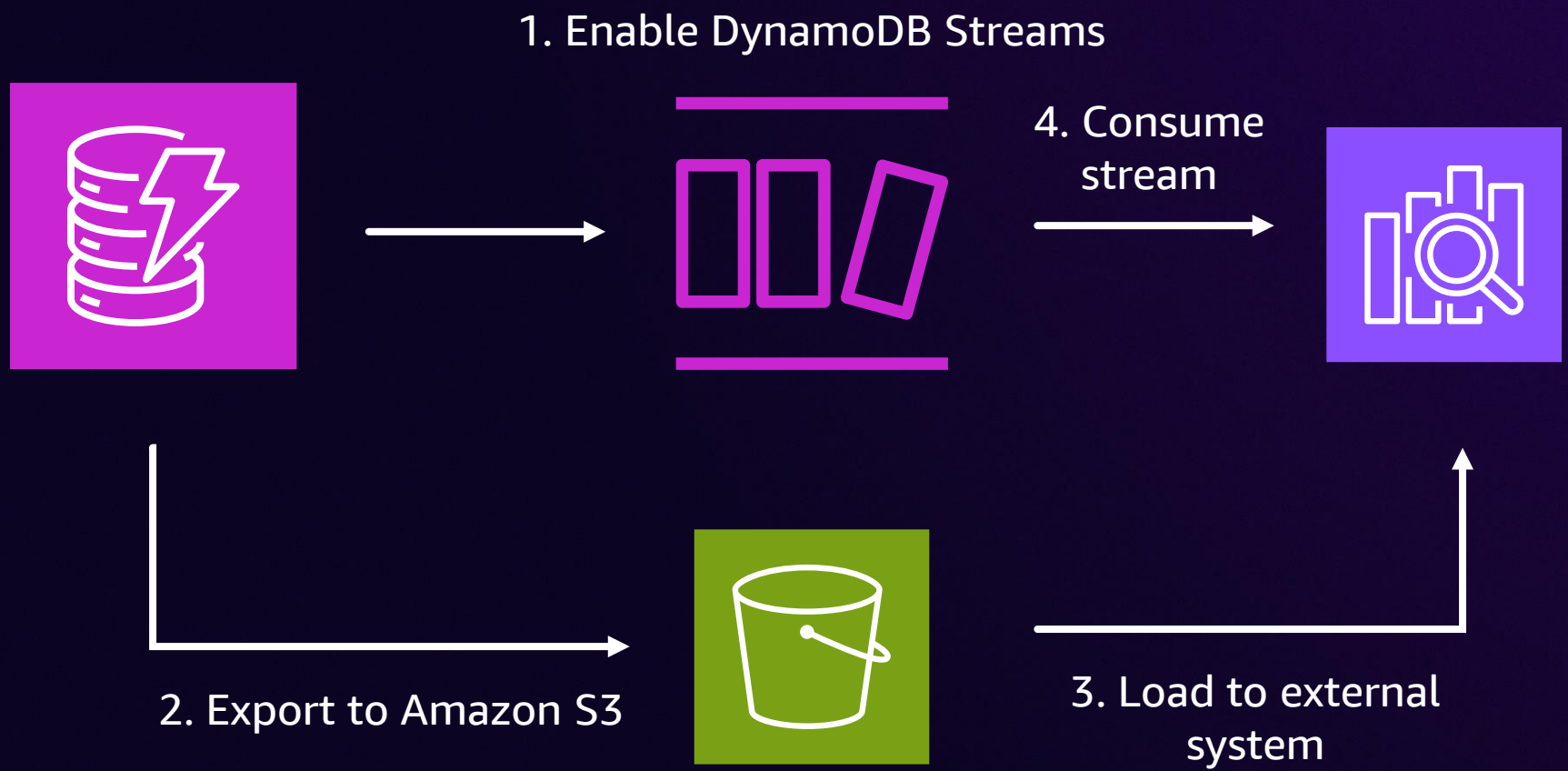
How to get initial data into external system?

DynamoDB export

DynamoDB data export to Amazon S3: how it works

PDF | RSS

DynamoDB export to S3 is a fully managed solution for exporting your DynamoDB data to an Amazon S3 bucket at scale. Using DynamoDB export to S3, you can export data from an Amazon DynamoDB



Complexities with syncing to external system

- Initial export

Complexities with syncing to external system

- Initial export
- OLTP/OLAP impedance mismatch

AWS News Blog

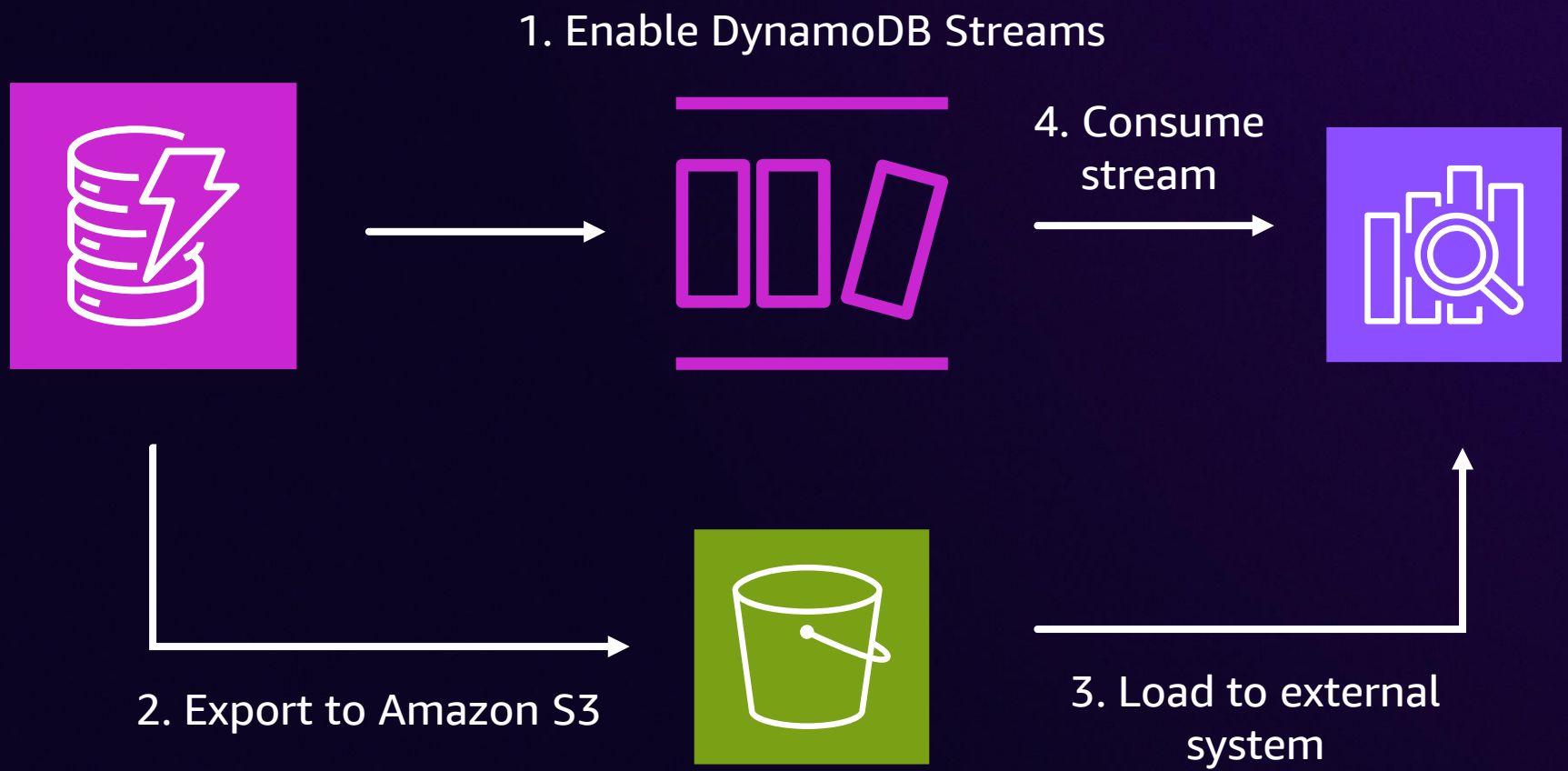
Amazon DynamoDB zero-ETL integration with Amazon OpenSearch Service is now available

by Channy Yun (윤석찬) | on 28 NOV 2023 | in Amazon DynamoDB, Amazon OpenSearch Service, Analytics, Announcements, AWS re:Invent, Database, Launch, News | Permalink | Share

0:00 / 0:00

Voiced by Amazon Polly

Today, we are announcing the general availability of **Amazon DynamoDB zero-ETL integration with Amazon OpenSearch Service**, which lets you



DynamoDB Streams use cases

- Solving the dual-write problem
- Exporting to secondary system

DynamoDB Streams use cases

- Solving the dual-write problem
- Exporting to secondary system
- Triggers + stored procedures

Trigger use case: Array indexing

- Example: Issue tracking system
  - Projects have issues
  - Issues have tags
    - Release version
    - Type (feature vs. bugfix vs. enhancement vs. tech debt vs. documentation)
    - Component (frontend vs. backend vs. database vs. auth)
  - **Goal: Allow for tag-based search**
    - feature + auth + sprint-23

Primary key		Attributes					
Partition key: Project	**Sort key: IssueID**						
NoSQLWorkbench	NWB-123	Title	Description	Status	CreatedAt	CreatedBy	Tags
		Add UUIDv7 support	Users want to use UUIDv7 in their models ...	closed	2022-02-14T08:30:00Z	amrith_kumar	["release-v3.0", "user-experience", "enhancement"]
DynamoDash	DD-1242	Title	Description	Status	CreatedAt	CreatedBy	Tags
		Login page crashes on Safari mobile	Users report white screen after clicking login button ...	in-progress	2024-01-15T08:30:00Z	alex_debrie	["bug", "frontend", "p0", "customer-reported", "ios", "auth"]
	DD-1258	Title	Description	Status	CreatedAt	CreatedBy	Tags
		Add OAuth support for Google SSO	Implement Google as additional SSO provider...	ready-for-review	2024-01-16T10:00:00	joe_idziorek	["feature", "auth", "sprint-23", "team-atlas"]
	DD-2891	Title	Description	Status	CreatedAt	CreatedBy	Tags
		Optimize database queries for dashboard	Dashboard loading times exceeding 3s in production	open	2024-01-17T09:15:00Z	alex_debrie	["performance", "backend", "database", "production"]
	DD-3219	Title	Description	Status	CreatedAt	CreatedBy	Tags
		Add table capacity cost forecasting	Add predictive analytics for DynamoDB table capacity costs ...	open	2024-01-18T11:45:00Z	alex_debrie	["feature", "cost-optimization", "sprint-24", "team-atlas", "backend", "analytics"]

Can I brute force it?

Issue tracking: Napkin math

Issue tracking: Napkin math

- Issue item size: 1–5 KB

Issue tracking: Napkin math

- Issue item size: 1–5 KB
- Issues per project:

Issue tracking: Napkin math

- Issue item size: 1–5 KB
- Issues per project:
  - Median: 1000

Issue tracking: Napkin math

- Issue item size: 1–5 KB
- Issues per project:
  - Median: 1000
  - P95: >10,000

Issue tracking: Napkin math

- Issue item size: 1–5 KB
- Issues per project:
  - Median: 1000
  - P95: >10,000

Brute-force calculations:

Issue tracking: Napkin math

- Issue item size: 1–5 KB
- Issues per project:
  - Median: 1000
  - P95: >10,000

Brute-force calculations:

- Median: 2 KB * 1000 == 2 MB (2 requests + 500 RCUs)

Issue tracking: Napkin math

- Issue item size: 1–5 KB
- Issues per project:
  - Median: 1000
  - P95: >10,000

Brute-force calculations:

- Median: 2 KB * 1000 == 2 MB (2 requests + 500 RCUs)
- P95: 5 KB * 10,000 == 50 MB (50 requests + 12,500 RCUs)

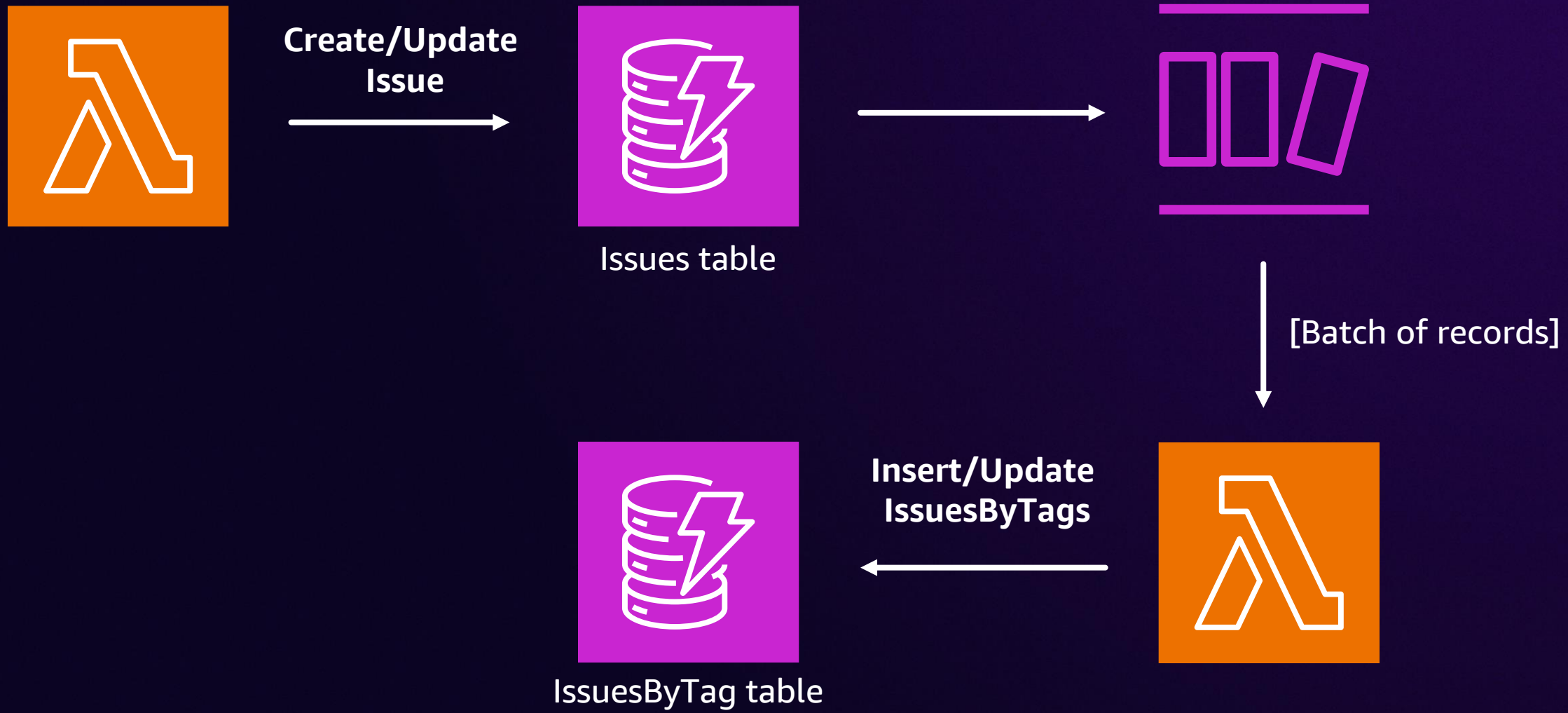
Issue tracking: Napkin math

- Issue item size: 1–5
- Issues per proje
  - Median: 1000
  - P95: >10,000

Brute-force calcula

- Median: 2 KB * 100 + 500 RCUs)
- P95: 5 KB * 10,000 == 5 uests + 12,500 RCUs)

Array indexing with streams



Primary key		Attributes
Partition key: ProjectTag	Sort key: CreatedAt	
NoSQLWorkbecnh#user-experience	2022-02-14T08:30:00Z	IssueID
		NWB-123
DynamoDash#feature	2024-01-15T10:00:00	IssueID
		DD-1258
	2024-01-18T11:45:00Z	IssueID
		DD-3219
DynamoDash#sprint-23	2024-01-18T11:45:00Z	IssueID
		DD-3219
DynamoDash#auth	2024-01-18T11:45:00Z	IssueID
		DD-3219
DynamoDash#performance	2024-01-17T09:15:00Z	IssueID
		DD-2891

Find issues with tags: feature + auth + sprint-23

Issue tracking: Napkin math part 2

- Tags per query: 3
- Issue item size: 200 bytes
- Issues per tag
  - Median: 10
  - P95: 200

Calculations:

- Median: 200 * 10 == 2 KB (1 RCU) * 3 requests == 3 RCUs
- P95: 200 * 200 == 40 KB (10 RCUs) * 3 requests == 30 RCUs

Issue tracking + denormalization

- Embedding

Primary key		Attributes					
Partition key: Project	**Sort key: IssueID**						
NoSQLWorkbench	NWB-123	Title	Description	Status	CreatedAt	CreatedBy	Tags
		Add UUIDv7 support	Users want to use UUIDv7 in their models ...	closed	2022-02-14T08:30:00Z	amrith_kumar	["release-v3.0", "user-experience", "enhancement"]
DynamoDash	DD-1242	Title	Description	Status	CreatedAt	CreatedBy	Tags
		Login page crashes on Safari mobile	Users report white screen after clicking login button ...	in-progress	2024-01-15T08:30:00Z	alex_debrie	["bug", "frontend", "p0", "customer-reported", "ios", "auth"]
	DD-1258	Title	Description	Status	CreatedAt	CreatedBy	Tags
		Add OAuth support for Google SSO	Implement Google as additional SSO provider...	ready-for-review	2024-01-16T10:00:00	joe_idziorek	["feature", "auth", "sprint-23", "team-atlas"]
	DD-2891	Title	Description	Status	CreatedAt	CreatedBy	Tags
		Optimize database queries for dashboard	Dashboard loading times exceeding 3s in production	open	2024-01-17T09:15:00Z	alex_debrie	["performance", "backend", "database", "production"]
	DD-3219	Title	Description	Status	CreatedAt	CreatedBy	Tags
		Add table capacity cost forecasting	Add predictive analytics for DynamoDB table capacity costs ...	open	2024-01-18T11:45:00Z	alex_debrie	["feature", "cost-optimization", "sprint-24", "team-atlas", "backend", "analytics"]

Issue tracking + denormalization

- Embedding

Issue tracking + denormalization

- Embedding
- Duplication

Primary key		Attributes
Partition key: ProjectTag	Sort key: CreatedAt	
NoSQLWorkbecnh#user-experience	2022-02-14T08:30:00Z	IssueID
		NWB-123
DynamoDash#feature	2024-01-15T10:00:00	IssueID
		DD-1258
	2024-01-18T11:45:00Z	IssueID
		DD-3219
DynamoDash#sprint-23	2024-01-18T11:45:00Z	IssueID
		DD-3219
DynamoDash#auth	2024-01-18T11:45:00Z	IssueID
		DD-3219
DynamoDash#performance	2024-01-17T09:15:00Z	IssueID
		DD-2891

Issue tracking + denormalization

- Embedding
- Duplication

Issue tracking + denormalization

- Embedding
- Duplication

Further consideration – how much to duplicate?

Primary key		Attributes
Partition key: ProjectTag	Sort key: CreatedAt	
NoSQLWorkbecnh#user-experience	2022-02-14T08:30:00Z	IssueID
		NWB-123
DynamoDash#feature	2024-01-15T10:00:00	IssueID
		DD-1258
	2024-01-18T11:45:00Z	IssueID
		DD-3219
DynamoDash#sprint-23	2024-01-18T11:45:00Z	IssueID
		DD-3219
DynamoDash#auth	2024-01-18T11:45:00Z	IssueID
		DD-3219
DynamoDash#performance	2024-01-17T09:15:00Z	IssueID
		DD-2891

Primary key		Attributes					
Partition key: ProjectTag	**Sort key: CreatedAt**						
NoSQLWorkbecnh#user-experience	2022-02-14T08:30:00Z	IssueID	Title	Description	Status	CreatedBy	Tags
		NWB-123	Add UUIDv7 support	Users want to use UUIDv7 in their models ...	closed	amrith_kumar	["release-v3.0", "user-experience", "enhancement"]
DynamoDash#feature	2024-01-15T10:00:00	IssueID	Title	Description	Status	CreatedBy	Tags
		DD-1258	Add OAuth support for Google SSO	Implement Google as additional SSO provider ...	ready-for-review	joe_idziorek	["feature", "auth", "sprint-23", "team-atlas"]
	2024-01-18T11:45:00Z	IssueID	Title	Description	Status	CreatedBy	Tags
		DD-3219	Add table capacity cost forecasting	Add predictive analytics for DynamoDB table capacity costs ...	open	alex_debrie	["feature", "cost-optimization", "sprint-23", "team-atlas", "backend", "analytics"]
DynamoDash#sprint-23	2024-01-18T11:45:00Z	IssueID	Title	Description	Status	CreatedBy	Tags
		DD-3219	Add table capacity cost forecasting	Add predictive analytics for DynamoDB table capacity costs ...	open	alex_debrie	["feature", "cost-optimization", "sprint-23", "team-atlas", "backend", "analytics"]
DynamoDash#auth	2024-01-18T11:45:00Z	IssueID	Title	Description	Status	CreatedBy	Tags
		DD-3219	Add table capacity cost forecasting	Add predictive analytics for DynamoDB table capacity costs ...	open	alex_debrie	["feature", "cost-optimization", "sprint-23", "team-atlas", "backend", "analytics"]
DynamoDash#performance	2024-01-17T09:15:00Z	IssueID	Title	Description	Status	CreatedBy	Tags
		DD-2201	Optimize database queries for dashboard	Dashboard loading times exceeding 3s in production ...	open	alex_debrie	["performance", "backend", "database", "production"]

Find issues with tags: feature + auth + sprint-23

Issue tracking + denormalization

- More duplication:
  - **Benefits:** Fewer read operations/lower latency
  - **Downside:** Higher write cost/maintenance

Other trigger + stored procedure use cases

- Maintaining aggregations
- Tracking version histories
- Hierarchical rollups

DynamoDB Streams use cases

- Solving the dual-write problem
- Exporting to secondary system
- Triggers + stored procedures

Make it Dynamo

Make it Dynamo

- Basics first

Tips for all modeling styles

- Design for your access patterns
  - Use meaningful primary keys
  - Don’t needlessly overload item collections
- Think about your writes early
  - Use conditional writes
- Flatten hierarchies
  - Denormalize where prudent

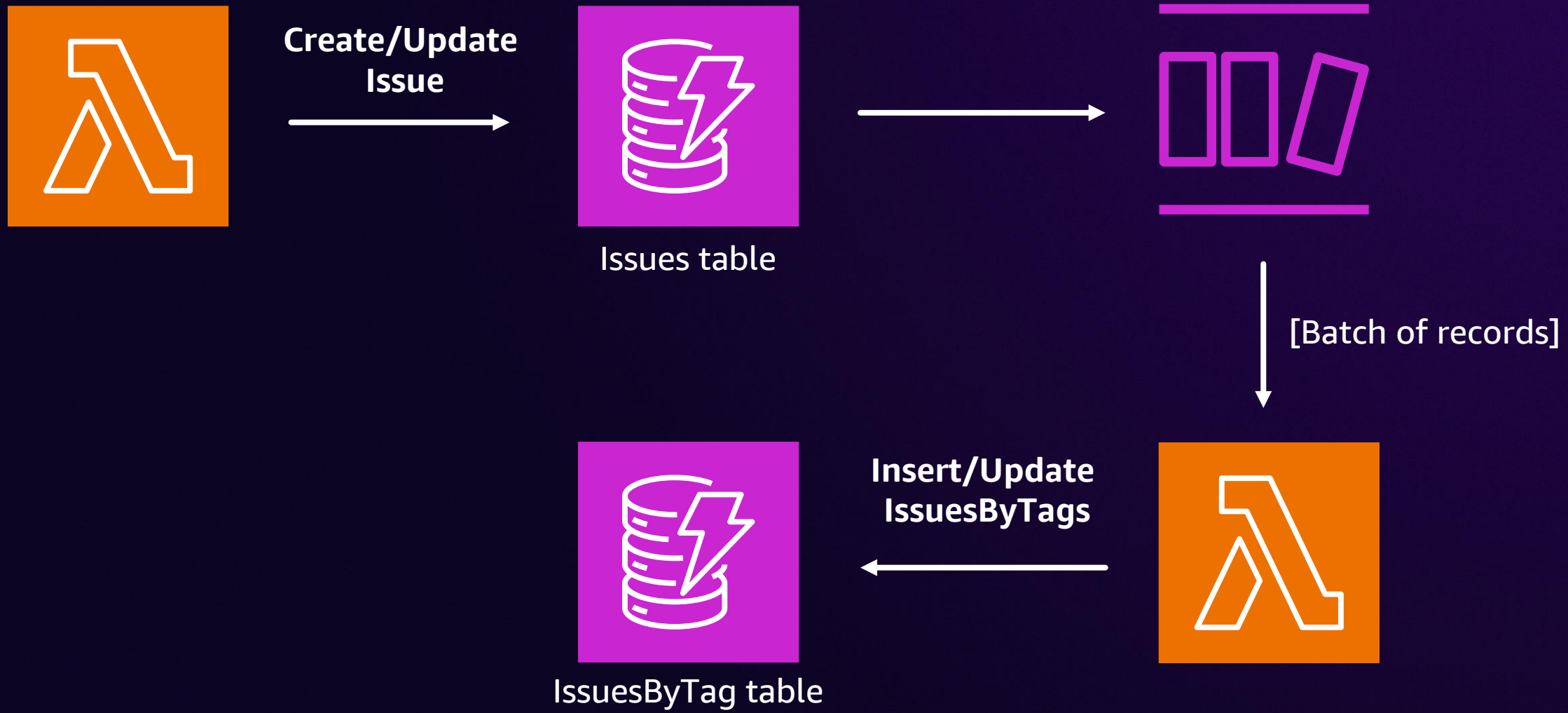
Make it Dynamo

- Basics first

Make it Dynamo

- Basics first
- Use the building blocks

Array indexing with streams



Issue tracking: Napkin math part 2

- Tags per query: 3
- Issue item size: 200 bytes
- Issues per tag
  - Median: 10
  - P95: 200

Calculations:

- Median: 200 * 10 == 2 KB (1 RCU) * 3 requests == 3 RCUs
- P95: 200 * 200 == 40 KB (10 RCUs) * 3 requests == 30 RCUs

Issue tracking + denormalization

- Embedding
- Duplication

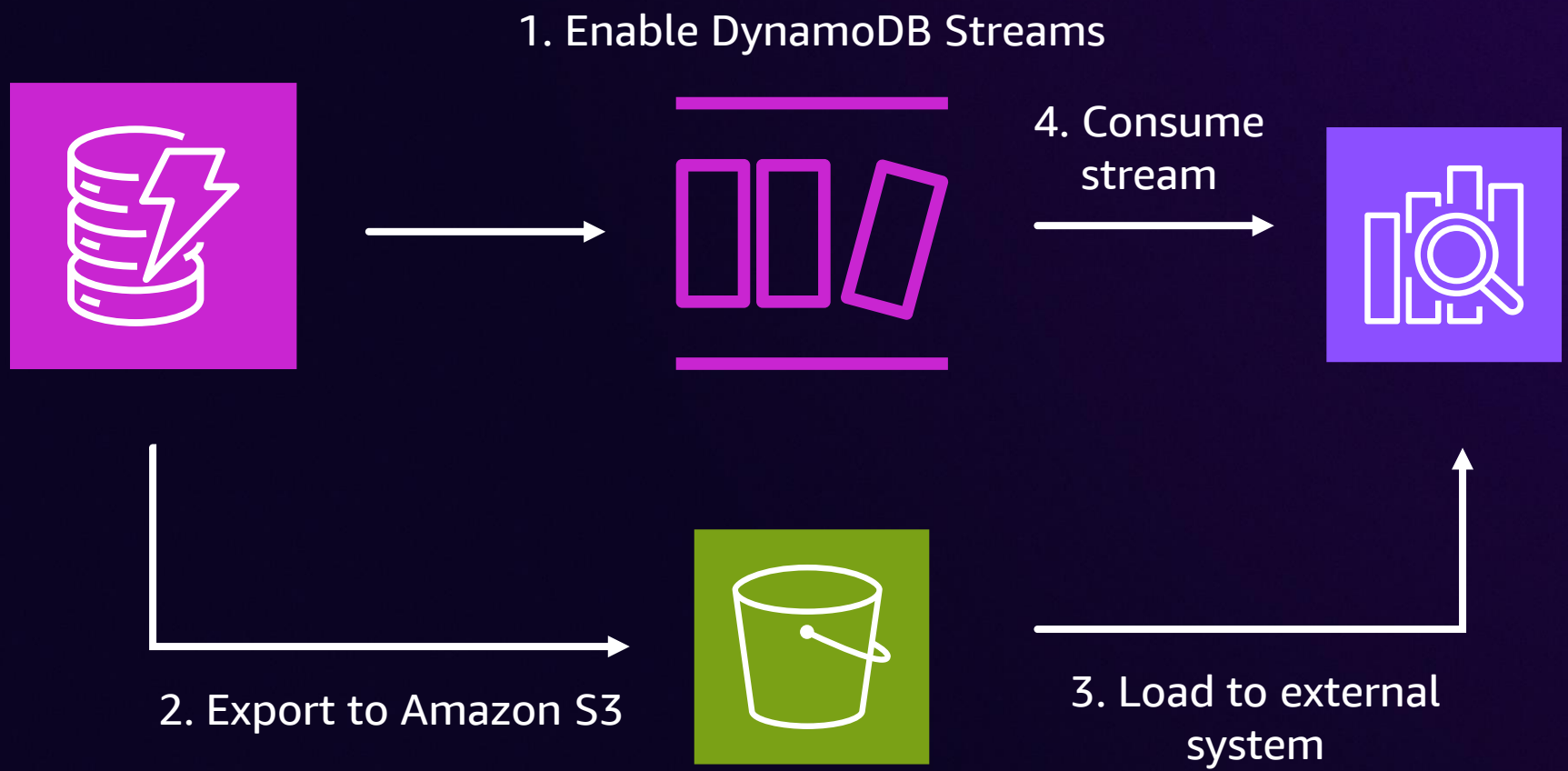
Further consideration – how much to duplicate?

Make it Dynamo

- Basics first
- Use the building blocks

Make it Dynamo

- Basics first
- Use the building blocks
- Secondary system when necessary



Make it Dynamo

- Basics first
- Use the building blocks
- Secondary system when necessary

Make it Dynamo

- Basics first
- Use the building blocks
- Secondary system when necessary
- Split + sort are surprisingly powerful!

Primary key

Primary key		Attributes		
Partition key: Username	**Sort key: OrderId**			
Grouped by partition key	Ordered by sort key	OrderCreatedAt	OrderStatus	OrderAmount
alexdebrie	01HNZNG5QDMB4014EW20R54VY4	2024-02-06T16:54:55.981Z	Cancelled	34.99
	01JCF6Z3ATX2RXTE6TDKXT111Y	OrderCreatedAt	OrderStatus	OrderAmount
	01JV7M94CQEBGNB263Z7X48S7R	2024-11-12T03:34:07.450Z	Delivered	172.14
		OrderCreatedAt	OrderStatus	OrderAmount
		2025-05-14T14:48:19.607Z	Placed	94.35
product_joe	01K5RM9988AH0P7ZYHPAK2S3FF	OrderCreatedAt	OrderStatus	OrderAmount
		2025-09-22T11:52:28.168Z	Delivered	237.44
algo_amrith	01J2YKVT6S0CZBDBX5YCZXV05E	OrderCreatedAt	OrderStatus	OrderAmount
		2024-07-16T20:31:09.529Z	Delivered	311.64

Split + sort

- Group by high cardinality
  - TenantID, UserEmail, DeviceID
- Sort by meaningful value
  - Timestamp, VersionId, ULID/UUIDv7

This can be used in surprising ways!

- Geohashing
- IP lookup

Make it Dynamo

- Basics first
- Use the building blocks
- Secondary system when necessary
- Split + sort are surprisingly powerful!

Thank you!

Please complete the session survey in the mobile app

Alex DeBrie
@alexbdebrie
alexdebrie1@gmail.com