Creating Elaborate DynamoDB Tables

Stefan Roman
DEVOPS ENGINEER

www.katapult.cloud

Planning Table Throughput Capacity

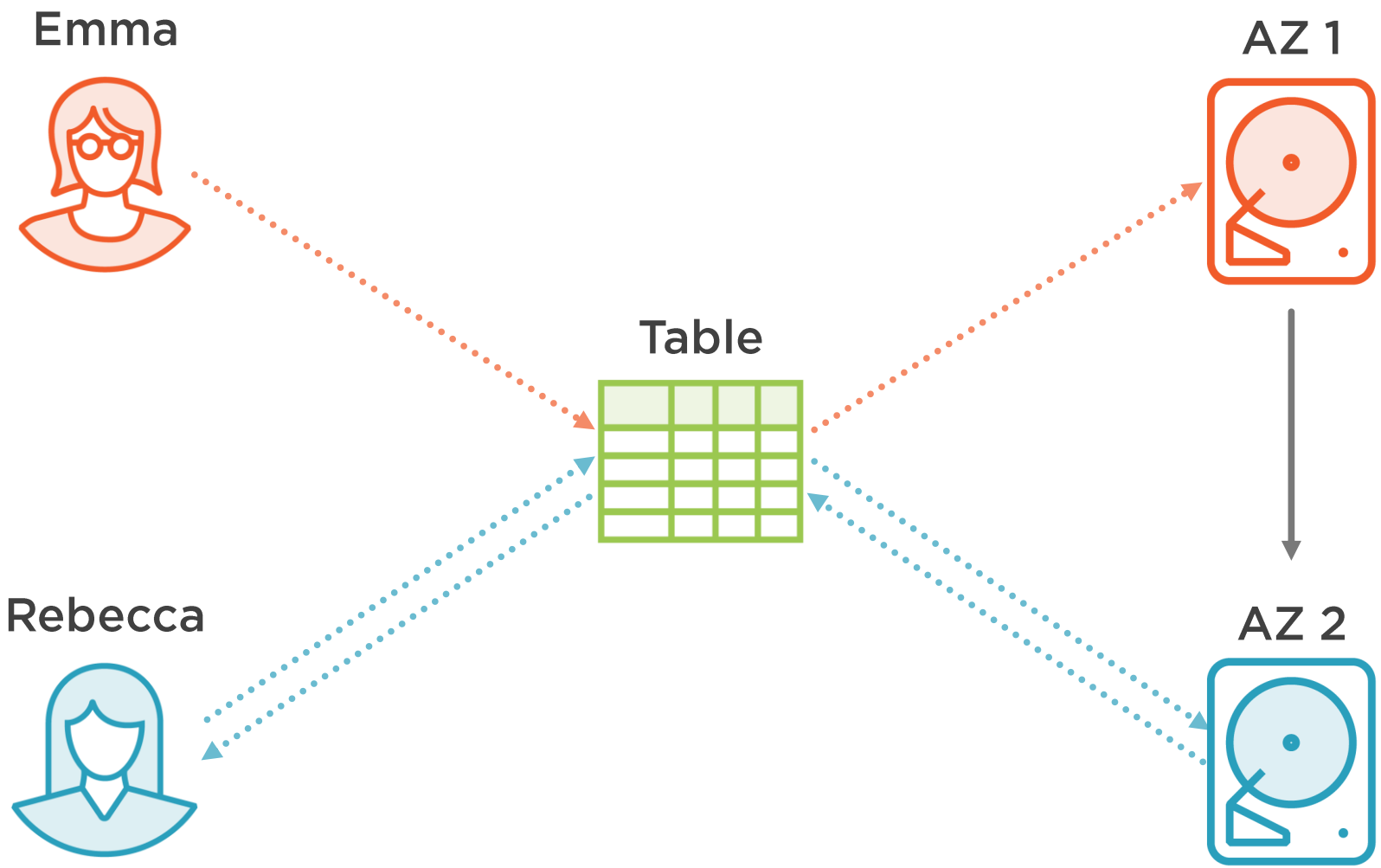
Write Capacity Units (WCU)

1 write operation per second of 1KB

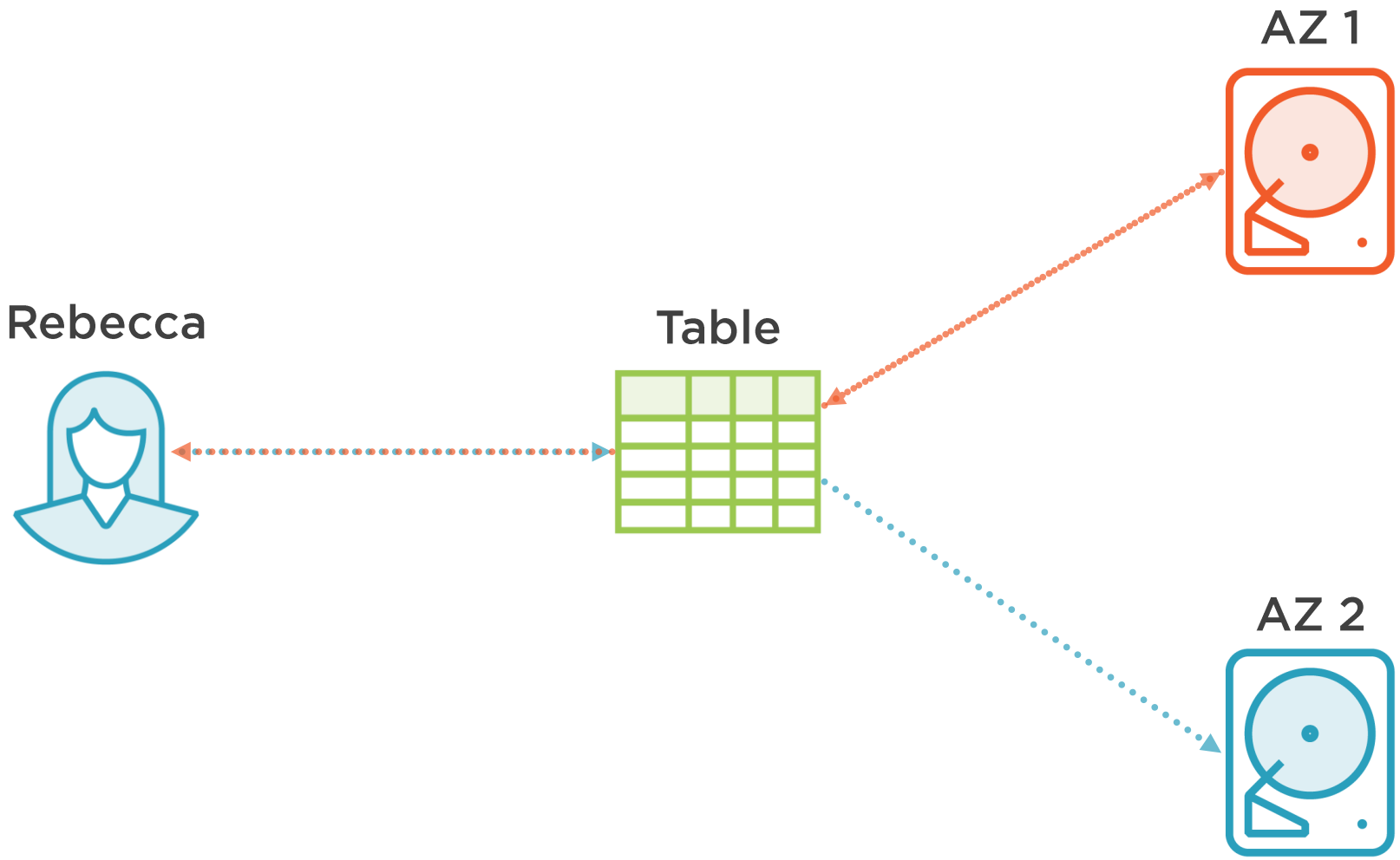
Read Capacity Units (RCU)

1 or 2 read operation per second of 4KB

Planning Table Throughput Capacity



Planning Table Throughput Capacity



Planning Table Throughput Capacity

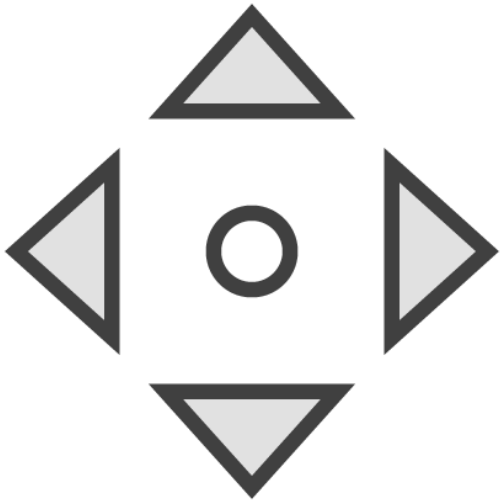
Eventually Consistent Reads

Applications that DO NOT immediately query updated items.

Strongly Consistent Reads

Applications that DO immediately query updated items.

Planning Table Throughput Capacity



Requirements to calculate expected throughput:

- size of items
- amount of items
- read consistency

Application requires to read 10 items of around 1KB each per second, using eventually consistent reads.

Planning Table Throughput Capacity

How many read units per item needed?

- Round up item size to nearest 4KB increment
  - 4KB / 4KB = *1 Read Unit*

How many read capacity units?

- Multiply read unit per item by number of reads required per second
  - 1 x 10 = *10 Read Capacity Units*

What read consistency our app uses?

- Eventually consistent divide by 2

Application requires to read 4 items of around 14KB each per second, using strongly consistent reads.

Planning Table Throughput Capacity

How many read units per item needed?

- Round up item size to nearest 4KB increment
  - 16KB / 4KB = *4 read units*

How many read capacity units?

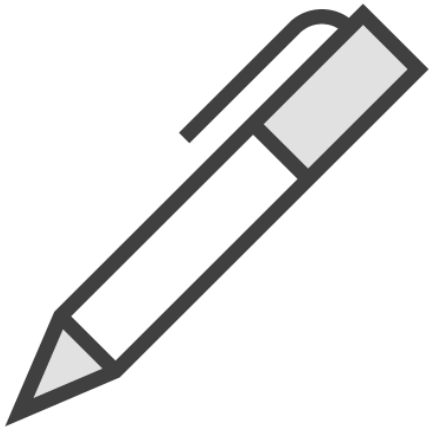
- Multiply read unit per item by number of reads required per second
  - 4 x 10 = *40 Read Capacity Units*

What read consistency our app uses?

- Strongly consistent divide by 1

Application requires to write 10 items of around 7KB each per second.

Planning Table Throughput Capacity



How many write capacity units?

- Multiply kilobytes by number of writes required per second
  - 7 x 10 = *70 Write Capacity Units*

Request Throttling

An application that exceeds provisioned capacity will have additional requests throttled.

Planning Table Throughput Capacity

Provisioned Autoscaling

Predict traffic patterns

Paying for provisioned capacity

Capping performance and price

On-demand Autoscaling

Unpredictable traffic patterns

Paying for amount of requests

Unlimited performance and price

Demo

On-demand autoscaling table

Provisioned throughput autoscaling table

Understanding Secondary Indices

```
{
   "Name": "Bob",
   "Age": 26,
   "Company": "Globomantics",
   "Position": "DevOps"
}
```

Understanding Secondary Indices

Local Secondary Index

Global Secondary Index

Understanding Secondary Indices

Local Secondary Index

Same Partition but different Sort key

Scoped to base table partitions

Shares throughput settings and pricing

5 indexes maximum

Global Secondary Index

Different Partition and Sort key

Spans across all partitions

Own throughput settings and pricing

20 indexes maximum

Understanding Secondary Indices

```
{
   "Name": "Bob",
   "Age": 26,
   "Company": "Globomantics",
   "Position": "DevOps"
}
```

Understanding Secondary Indices

Local Secondary Index

Same Partition but different Sort key

Scoped to base table partitions

Shares throughput settings and pricing

5 indexes maximum

Global Secondary Index

Different Partition and Sort key

Spans across all partitions

Own throughput settings and pricing

20 indexes maximum

Understanding Secondary Indices

```
{
   "Name": "Bob",
   "Age": 26,
   "Company": "Globomantics",
   "Position": "DevOps"
}
```

Understanding Secondary Indices

Local Secondary Index

Same Partition but different Sort key

Scoped to base table partitions

Shares throughput settings and pricing

5 indexes maximum

Global Secondary Index

Different Partition and Sort key

Spans across all partitions

Own throughput settings and pricing

20 indexes maximum

Understanding Secondary Indices

KEYS ONLY

Only primary keys

INCLUDE

Includes selected attributes

ALL

All attributes in an item

Demo

Create a table with Local Secondary Index

Add Global Secondary Index to the same table

Master to Master Replication

Collection of Tables

Replicates Items to All Tables

Ideal for Multi-region Applications

Master to Master Replication

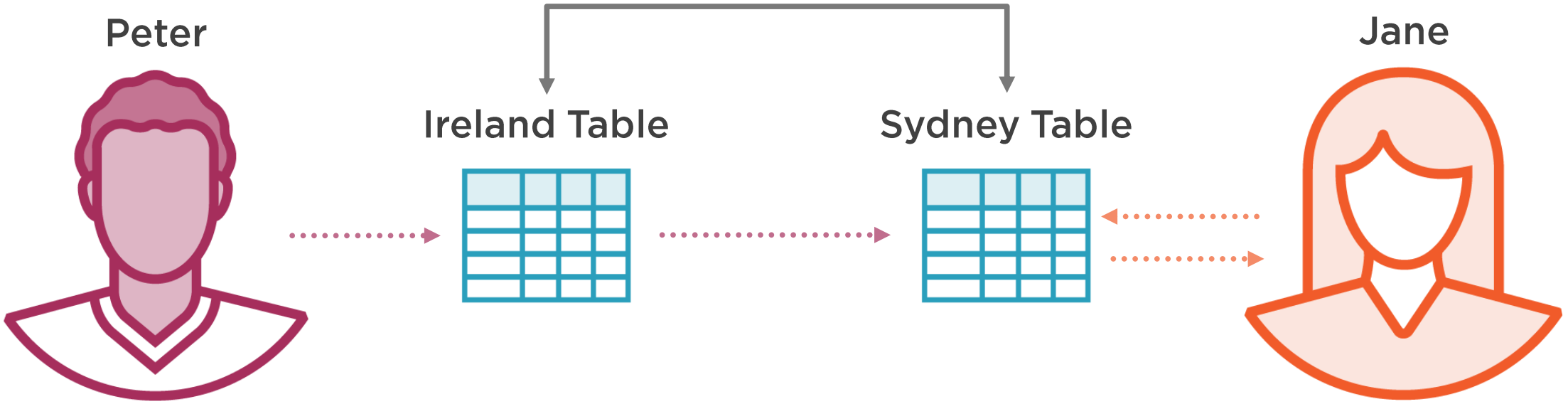
Same write capacity is required

Must have the same name

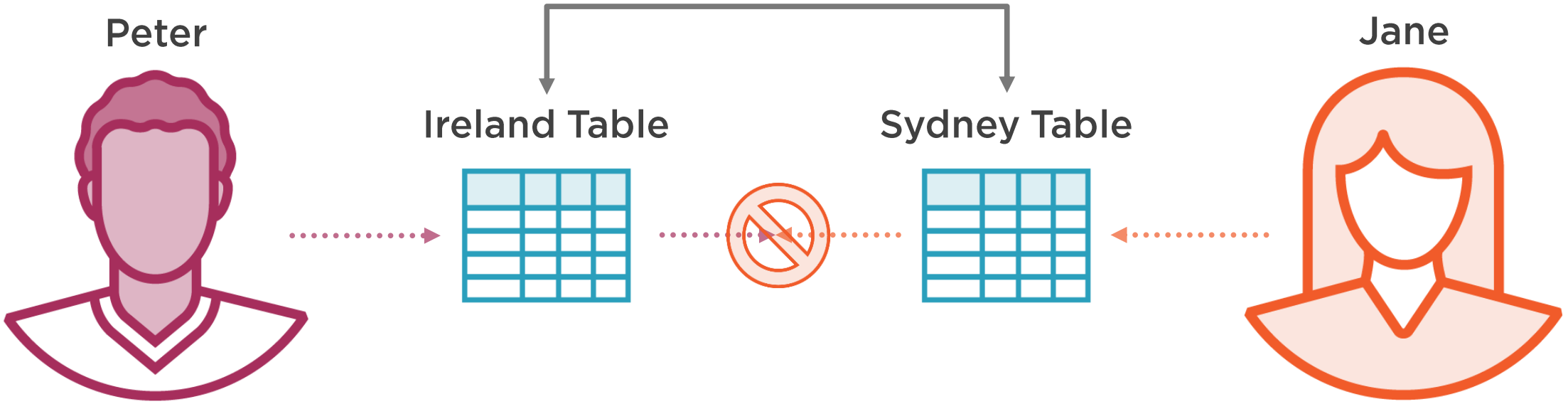
Must have the same primary key

Tables must be empty

Master to Master Replication



Master to Master Replication



Master to Master Replication

Capturing table activity

Contains data modification information

Logs activity up to 24 hours

Master to Master Replication

Keys Only

New Image

Old Image

New and Old Images

Demo

Create a basic table

Enable DynamoDB streams and create a replica table

Observe replication

Encrypting Items

Encryption is enabled by default

- At-rest
- In-transit

Encrypts DynamoDB streams

Encrypts Local and Global secondary indices

Setting Item Expiration

```
{
  "ID": 562651,
  "User": "Bob",
  "Access": "Granted"
}
```

Setting Item Expiration

```
{
  "ID": 562651,
  "User": "Bob",
  "Access": "Granted",
  "Expire": 1525568400
}
```

Demo

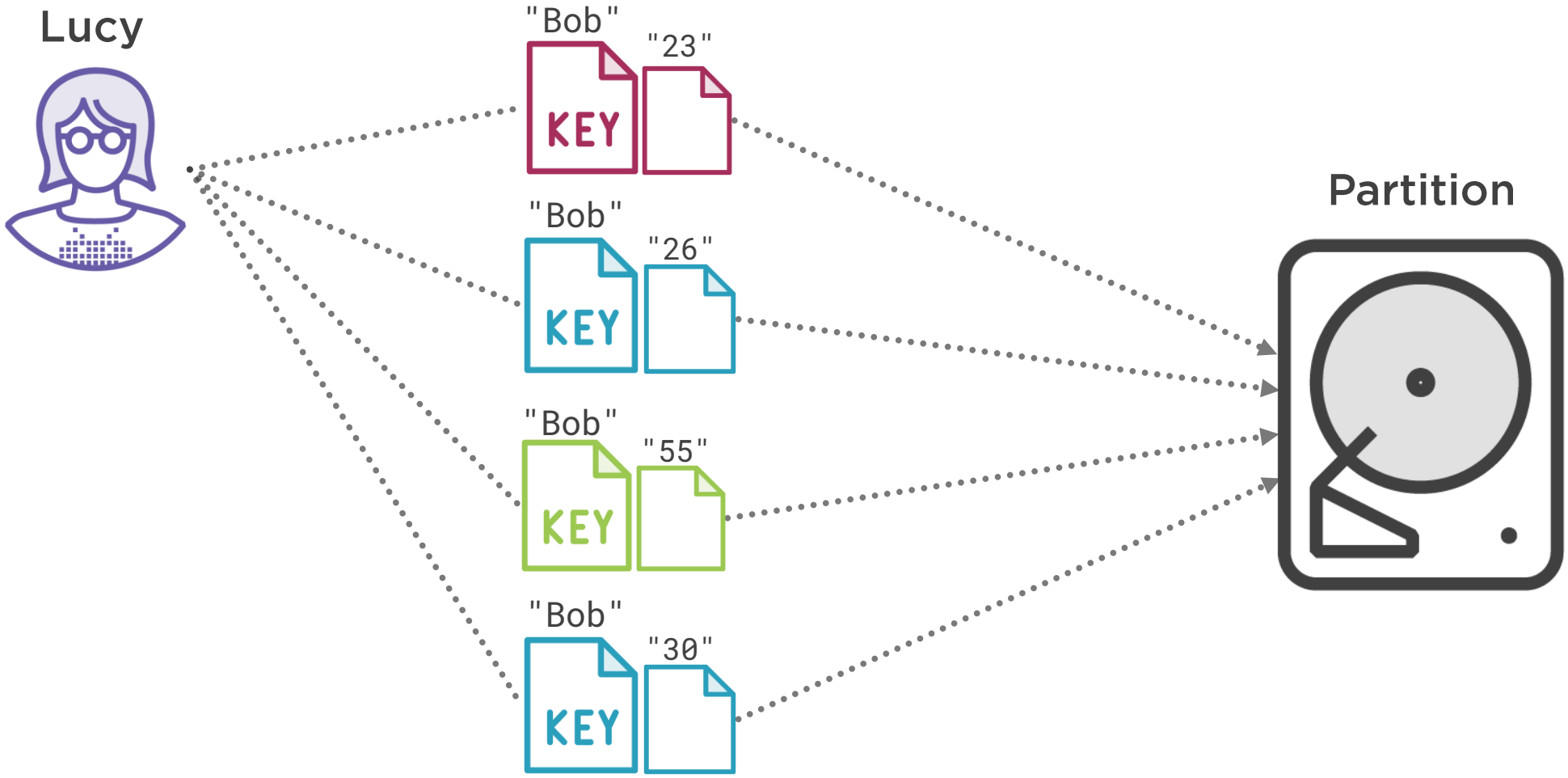
Create a basic table

Enable Time-to-Live option

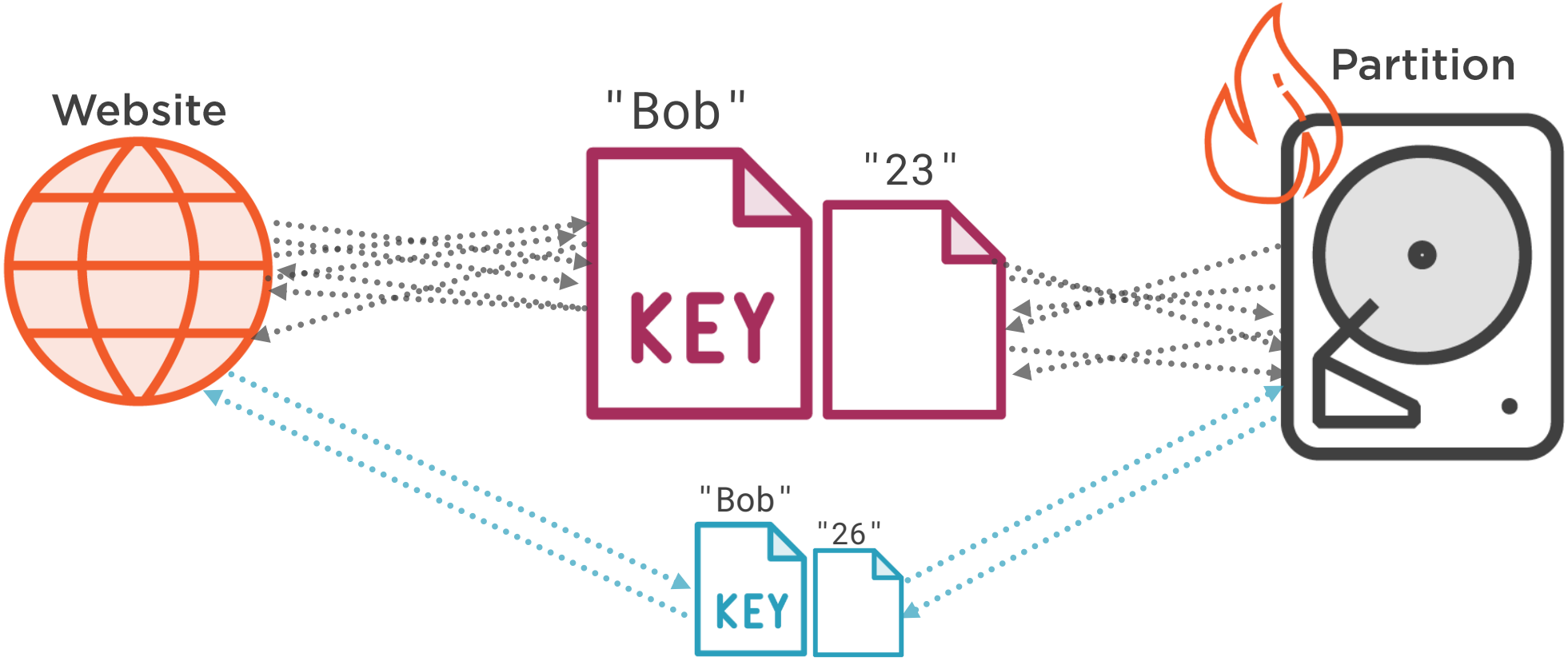
Add items with Time-to-Live attribute

Observe item deletion

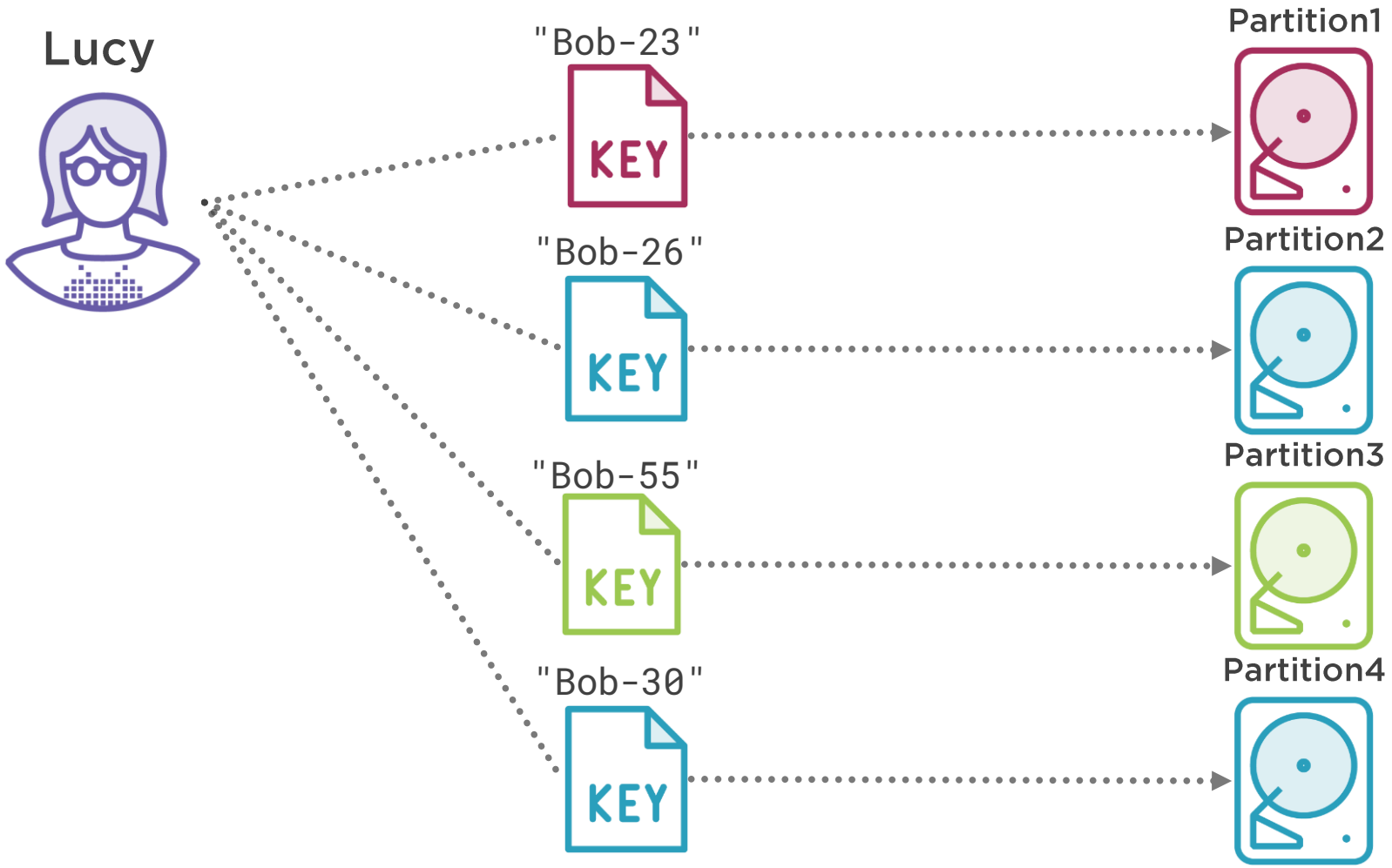
Designing Efficient Primary Keys



Designing Efficient Primary Keys



Designing Efficient Primary Keys

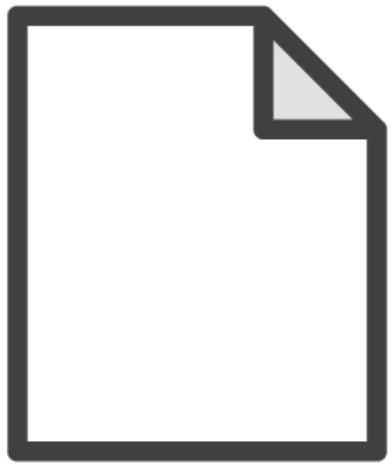


Designing Efficient Primary Keys

Partition key recommendations:

- Use combination of known information
- Querying data is difficult with randomly generated Partition keys

Designing Efficient Primary Keys



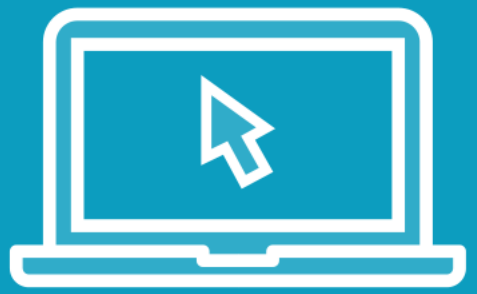
Sort key recommendations:

- Data can be queried with:
  - `starts-with`
  - `between`
  - `>`
  - `<`

Designing Efficient Primary Keys

```
{
   "Name": "Bob",
   "Location":[continent]#[state]#[city]
}
```

Demo



Create an elaborate employee directory table

Accessed by busy website

- Ability to query data based on *random strings*

Demo

Application that tracks access

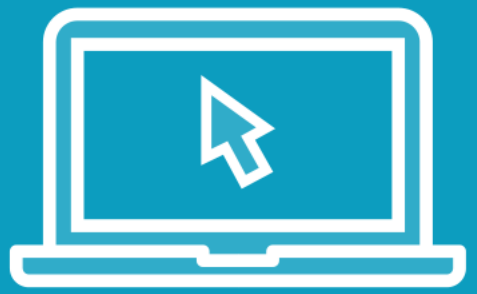
- Ability to query data based on *random strings*
- Updates *date accessed*
- Sort items based on *Name*

Demo

Application that tracks down temporary employees:

- Needs to have access to all attributes in items
- Query based on *Position* and *Contract Type*

Demo



Accessed from multiple parts of the world – *Sydney*, *Dublin*, *Oregon* and *India*

Remove *expired* temporary employee items

Extremely unpredictable

Summary

Calculating throughput capacity

Read consistency

- Eventual consistency
- Strong consistency

DynamoDB autoscaling

- Provisioned capacity
- On-demand

Local and Global secondary index

Global tables and streams

Summary

Time-to-Live attribute

Designing partition and sort keys

Summary

Elaborate global table

- Design of primary key
- Deployed table globally
- Designed secondary indices
- Implemented TTL