Azure Cosmos DB

AGENDA

Reviewing Azure Cosmos DB

Partitioning

Querying

Programming

Troubleshooting

Modeling & Planning

MODERN APPS FACE NEW CHALLENGES

Managing and syncing data distributed around the globe

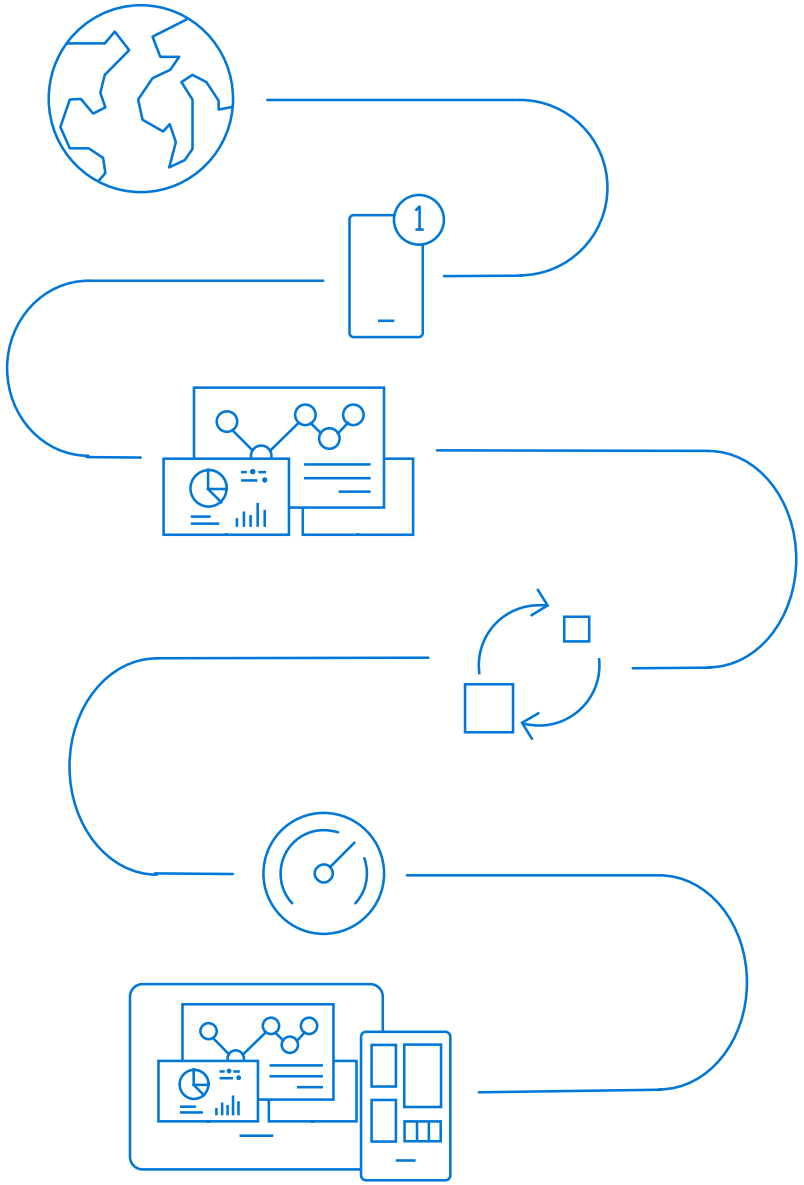
Delivering highly-responsive, real-time personalization

Processing and analyzing large, complex data

Scaling both throughput and storage based on global demand

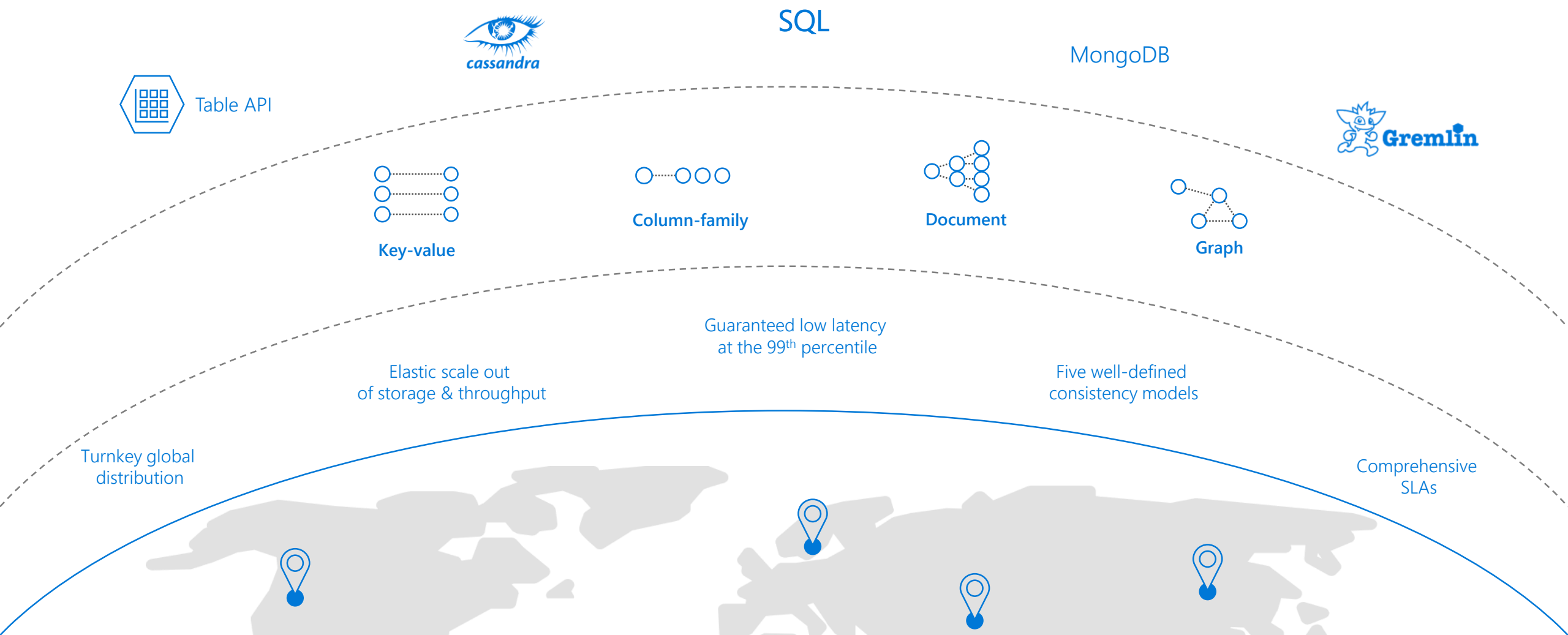
Offering low-latency to global users

Modernizing existing apps and data



AZURE COSMOS DB

A globally distributed, massively scalable, multi-model database service



AZURE COSMOS DB

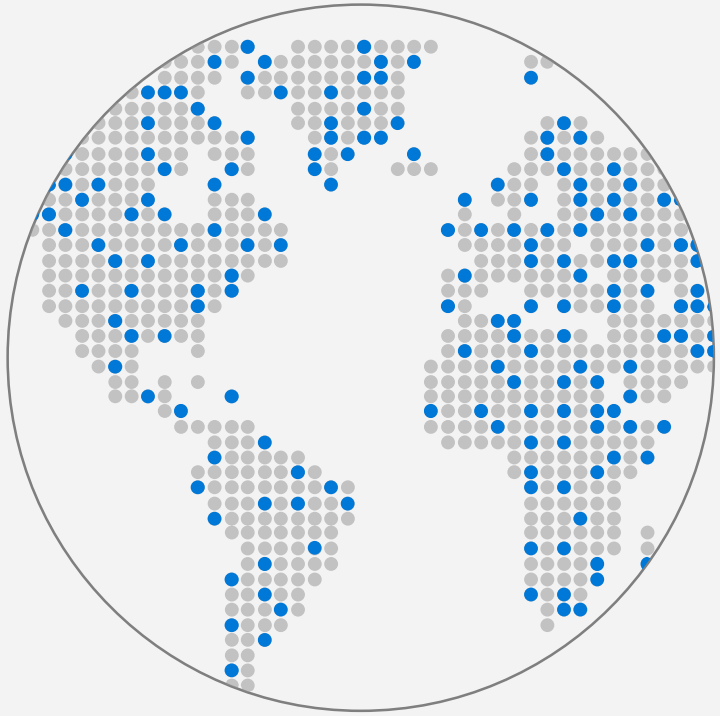
A FULLY-MANAGED GLOBALLY DISTRIBUTED DATABASE SERVICE BUILT TO GUARANTEE EXTREMELY LOW LATENCY AND MASSIVE SCALE FOR MODERN APPS

TURNKEY GLOBAL DISTRIBUTION

PUT YOUR DATA WHERE YOUR USERS ARE

Automatically replicate all your data around the world, and across more regions than Amazon and Google combined.

- Available in all Azure regions
- Manual and automatic failover
- Automatic & synchronous multi-region replication

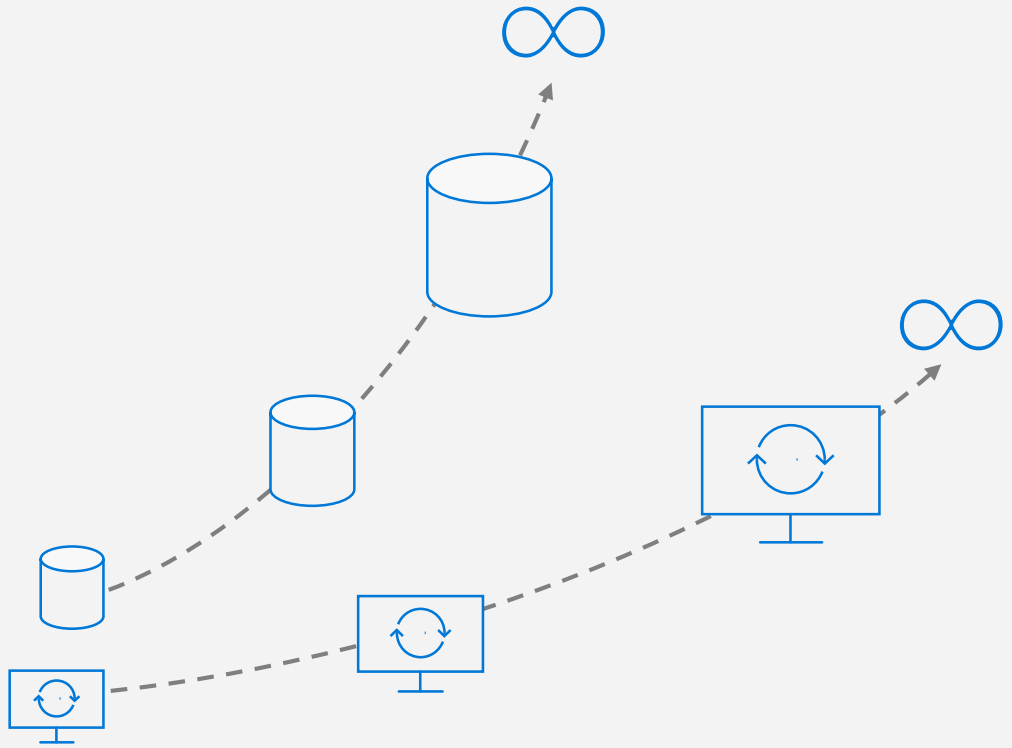


ELASTIC SCALE OUT OF STORAGE AND THROUGHPUT

SCALES AS YOUR APPS' NEEDS CHANGE

Independently and elastically scale storage and throughput across regions – even during unpredictable traffic bursts – with a database that adapts to your app's needs.

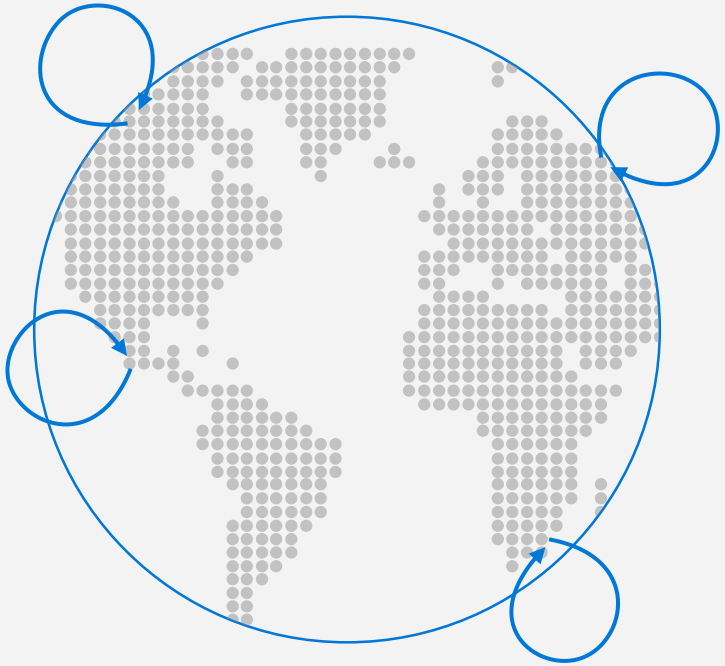
- Elastically scale throughput from 10 to 100s of millions of requests/sec across multiple regions
- Support for requests/sec for different workloads
- Pay only for the throughput and storage you need



GUARANTEED LOW LATENCY

PROVIDE USERS AROUND THE WORLD WITH FAST ACCESS TO DATA

Serve <10 ms read and <15 ms write requests at the 99th percentile from the region nearest to users, while delivering data globally.



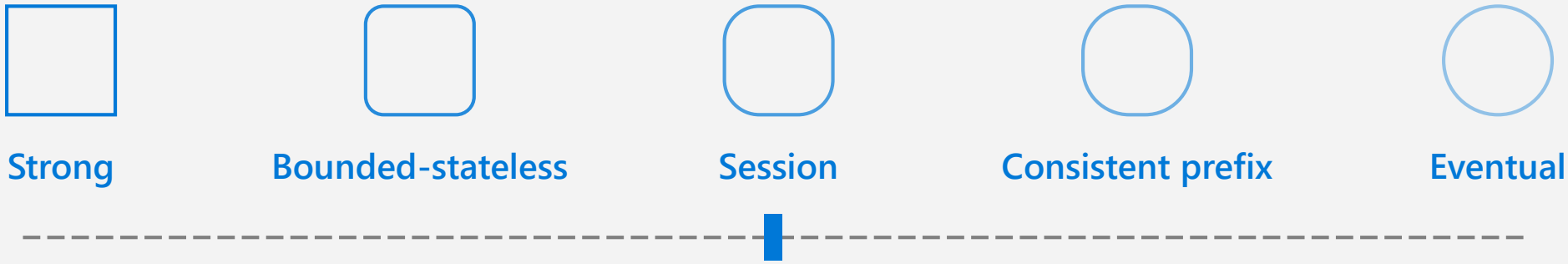
FIVE WELL-DEFINED CONSISTENCY MODELS

CHOOSE THE BEST CONSISTENCY MODEL FOR YOUR APP

Offers five consistency models

Provides control over performance-consistency tradeoffs, backed by comprehensive SLAs.

An intuitive programming model offering low latency and high availability for your planet-scale app.



MULTIPLE DATA MODELS AND APIs

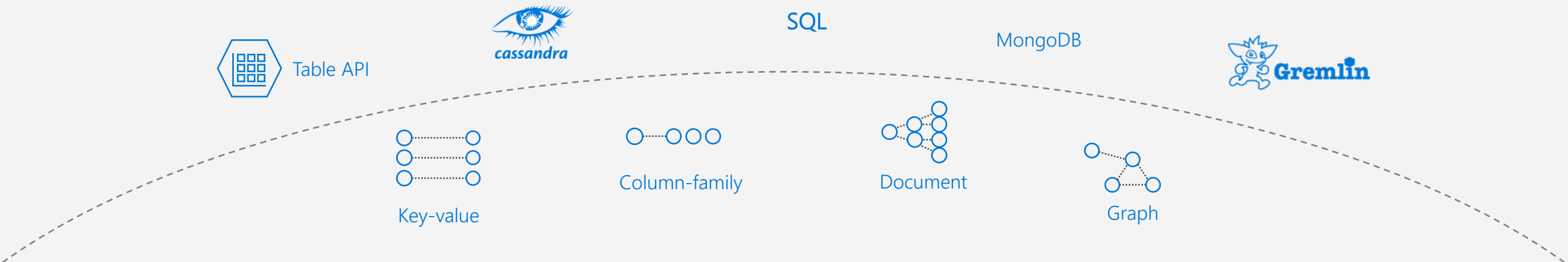
USE THE MODEL THAT FITS YOUR REQUIREMENTS, AND THE APIS, TOOLS, AND FRAMEWORKS YOU PREFER

Cosmos DB offers a multitude of APIs to access and query data including, SQL, various popular OSS APIs, and native support for NoSQL workloads.

Use key-value, tabular, graph, and document data

Data is automatically indexed, with no schema or secondary indexes required

Blazing fast queries with no lag

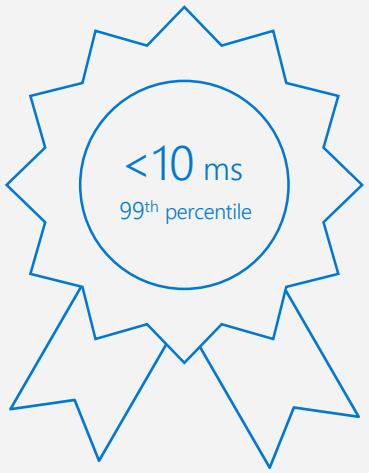


COMPREHENSIVE SLAs

RUN YOUR APP ON WORLD-CLASS INFRASTRUCTURE

Azure Cosmos DB is the only service with financially-backed SLAs for millisecond latency at the 99th percentile, 99.999% HA and guaranteed throughput and consistency

Latency



HA

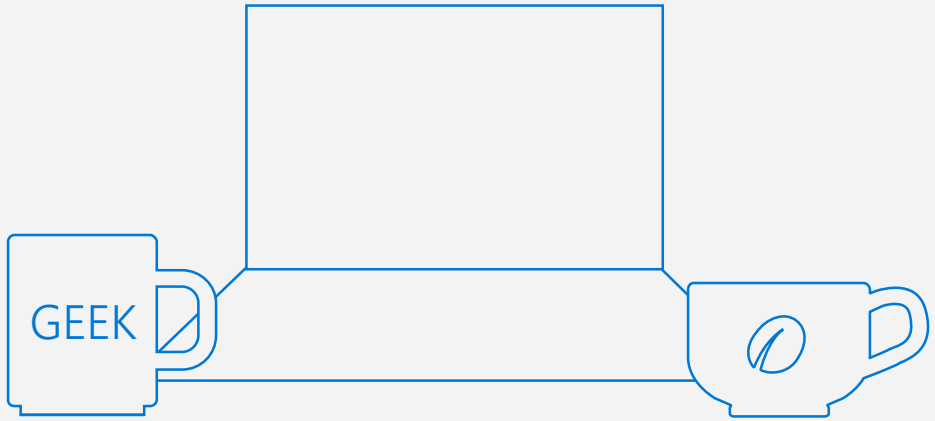
Throughput

Consistency

HANDLE ANY DATA WITH NO SCHEMA OR INDEXING REQUIRED

Azure Cosmos DB's schema-less service automatically indexes all your data, regardless of the data model, to delivery blazing fast queries.

- Automatic index management
- Synchronous auto-indexing
- No schemas or secondary indices needed
- Works across every data model



Item	Color	Microwave safe	Liquid capacity	CPU	Memory	Storage
Geek mug	Graphite	Yes	16ox	???	???	???
Coffee Bean mug	Tan	No	12oz	???	???	???
Surface book	Gray	???	???	3.4 GHz Intel Skylake Core i7-6600U	16GB	1 TB SSD

TRUST YOUR DATA TO INDUSTRY-LEADING SECURITY & COMPLIANCE

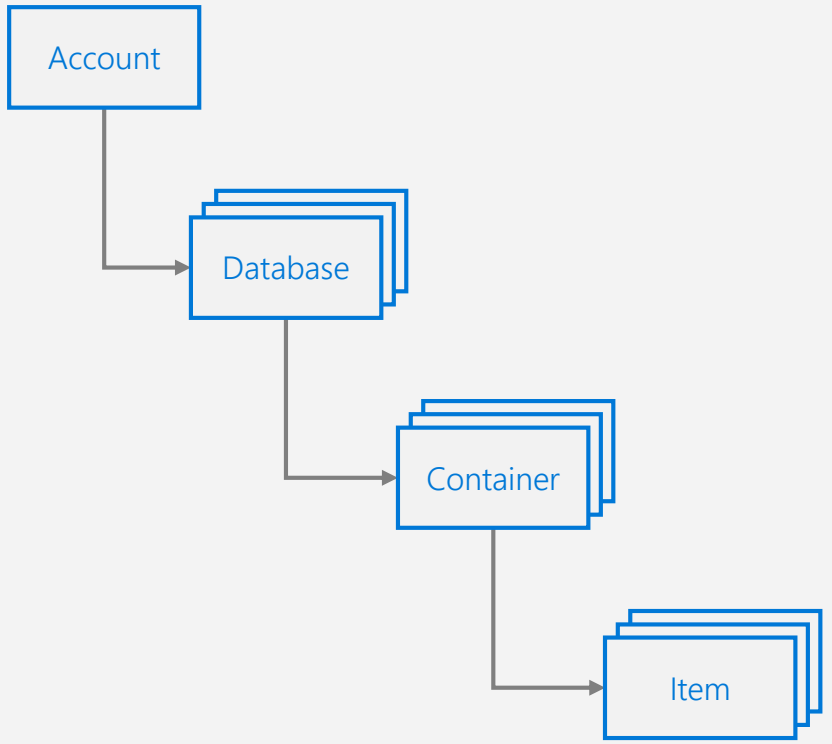
Azure is the world's most trusted cloud, with more certifications than any other cloud provider.

- Enterprise grade security
- Encryption at Rest
- Encryption is enabled automatically by default
- Comprehensive Azure compliance certification

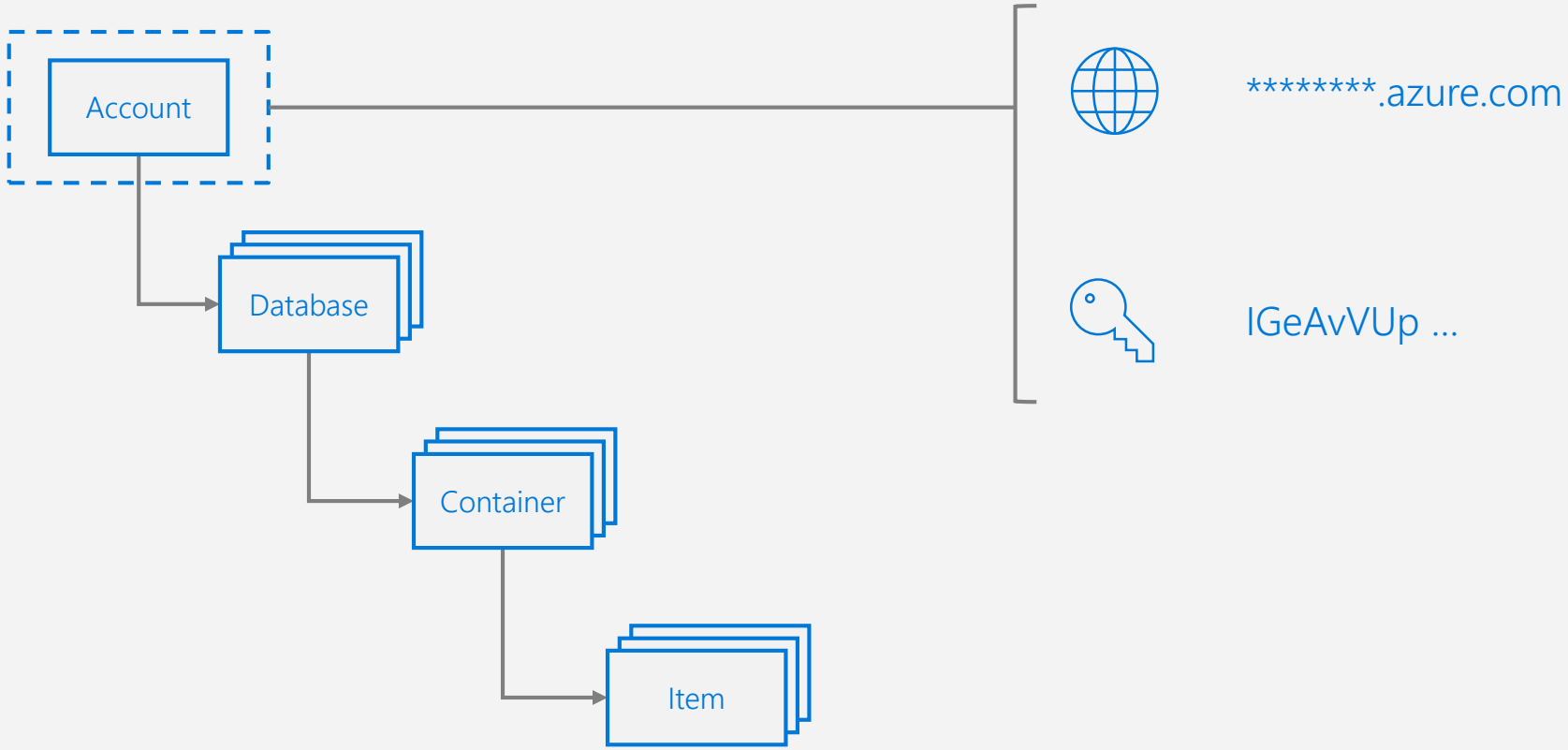
RESOURCE MODEL

Leveraging Azure Cosmos DB to automatically scale your data across the globe

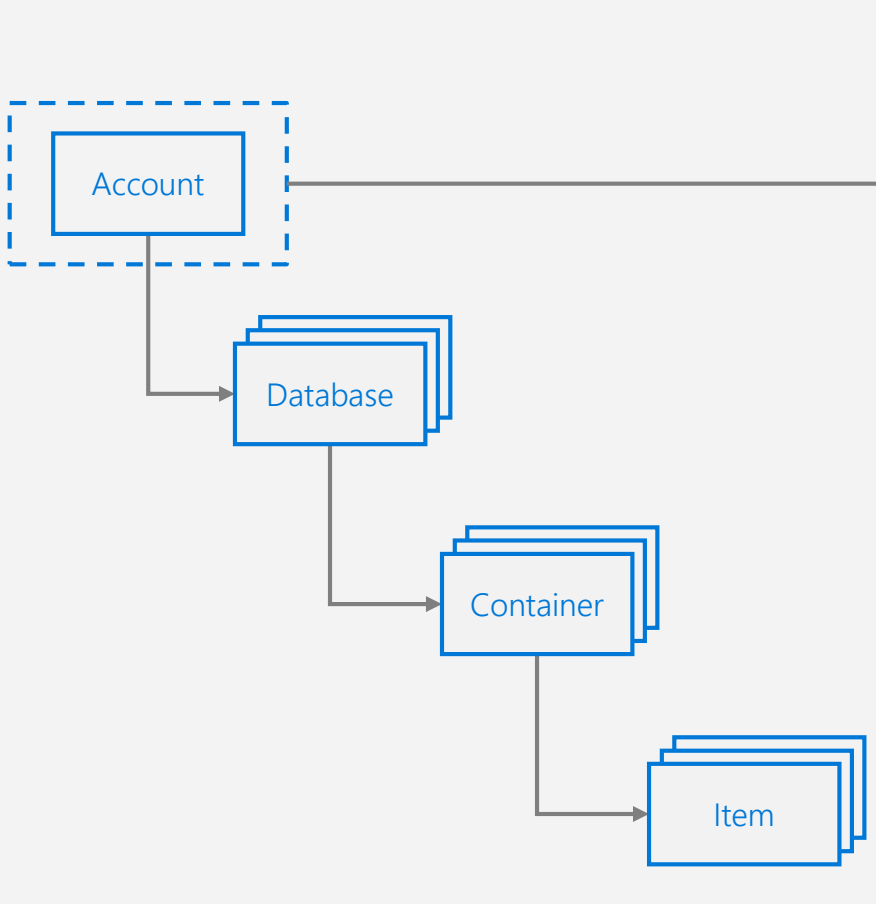
This module will reference partitioning in the context of all Azure Cosmos DB modules and APIs.

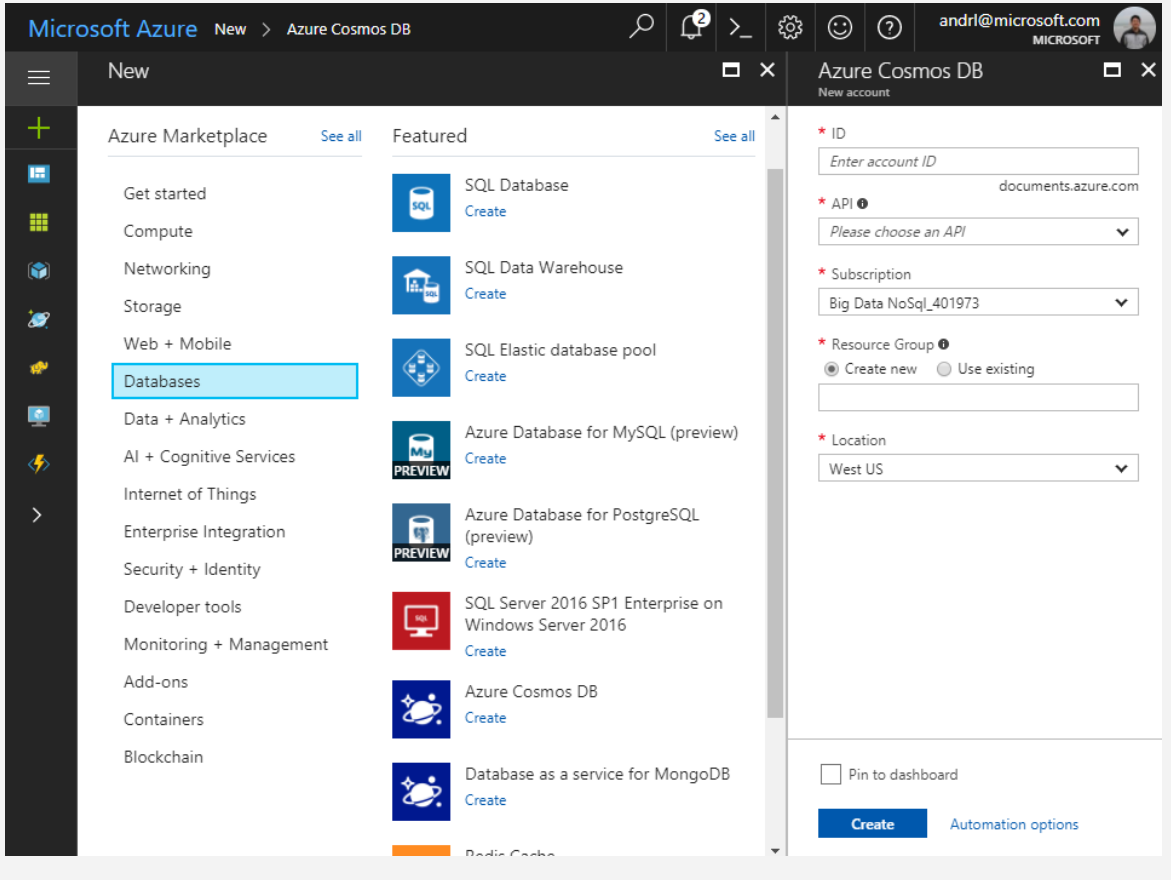


ACCOUNT URI AND CREDENTIALS

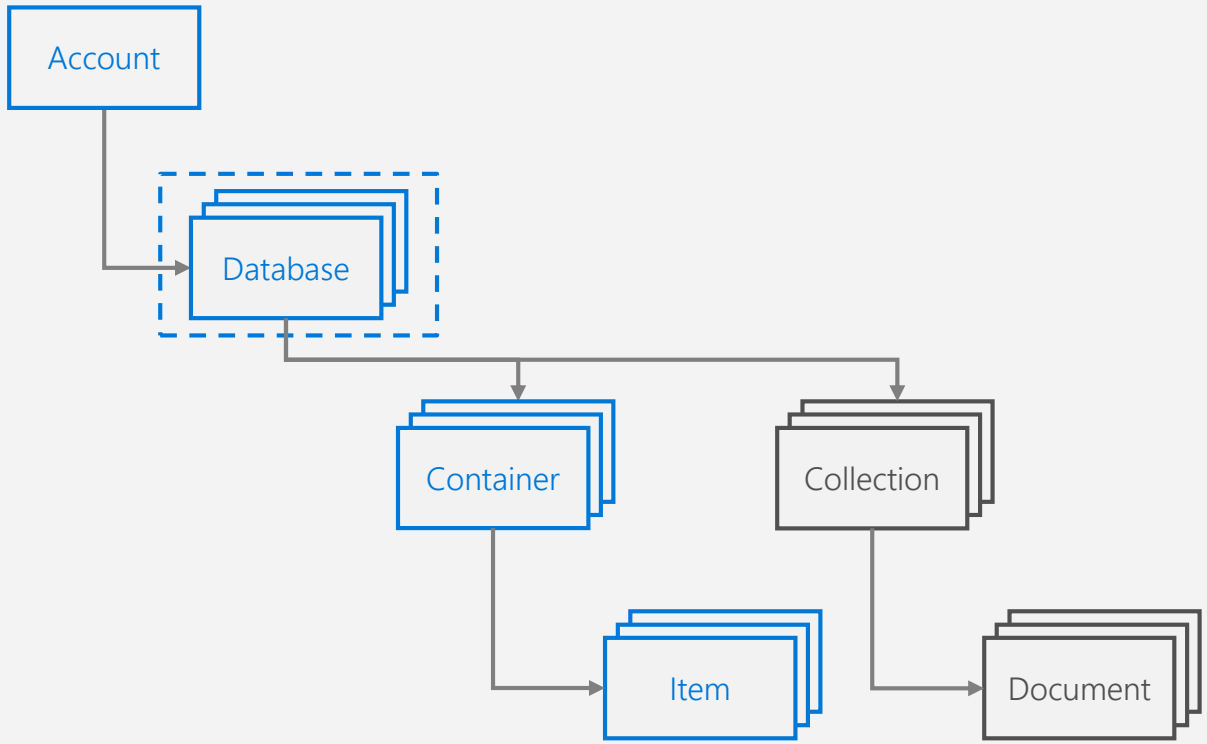


CREATING ACCOUNT

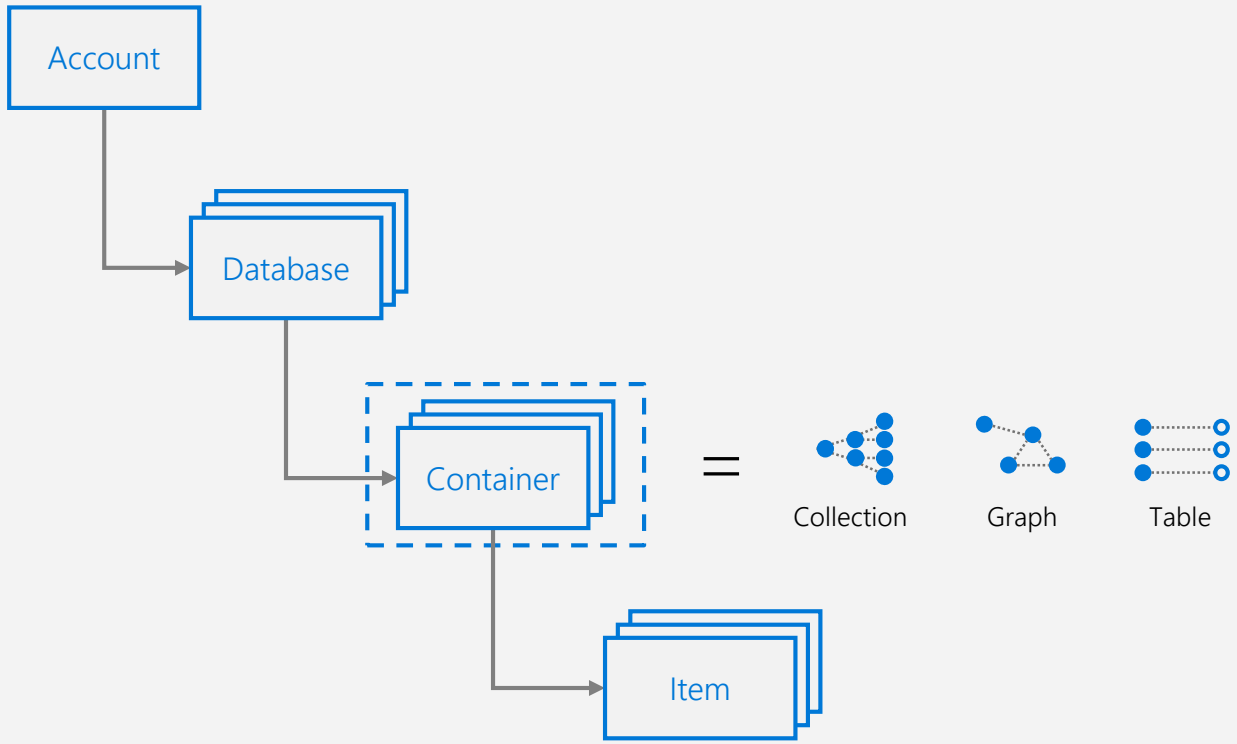




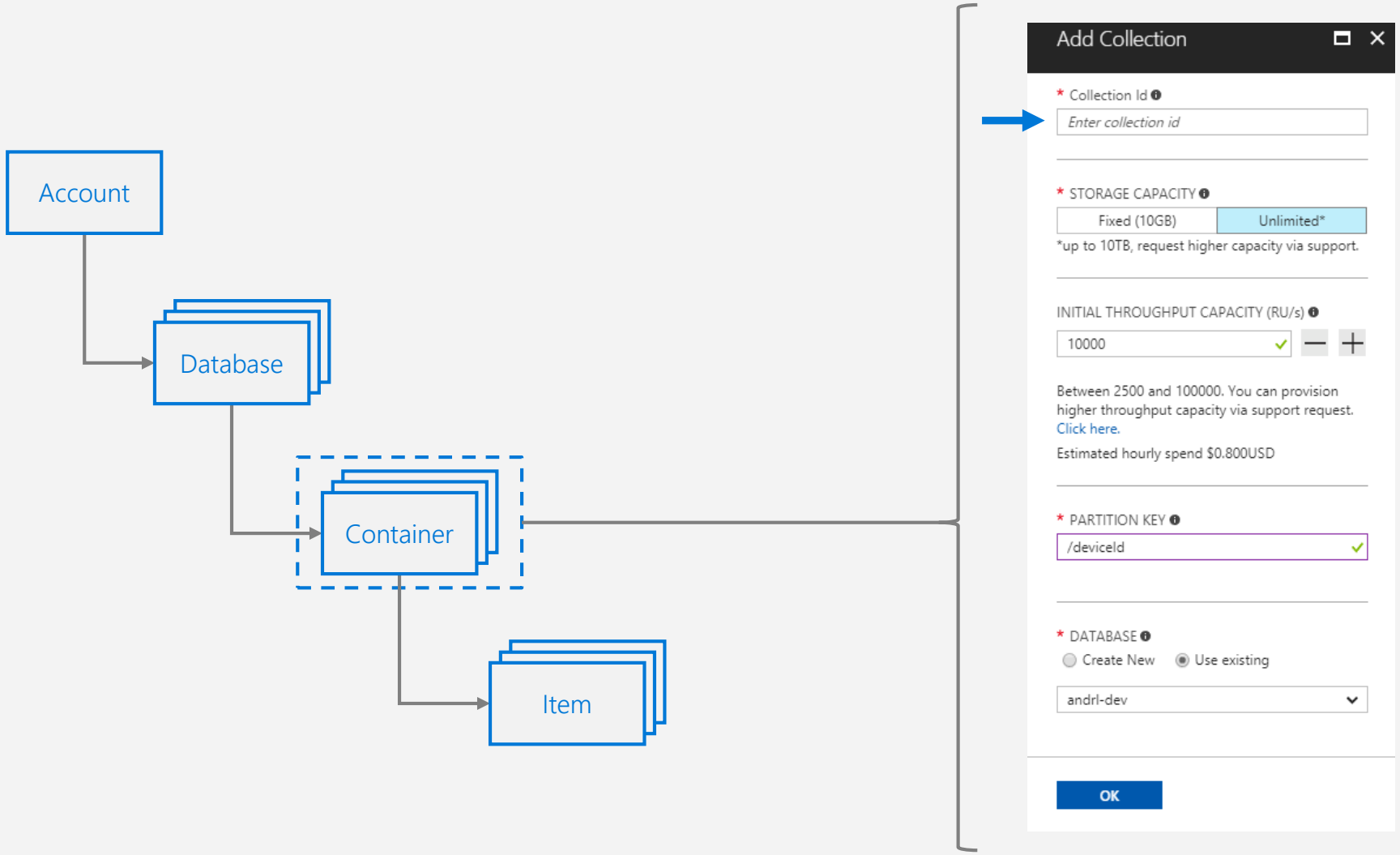
DATABASE REPRESENTATIONS



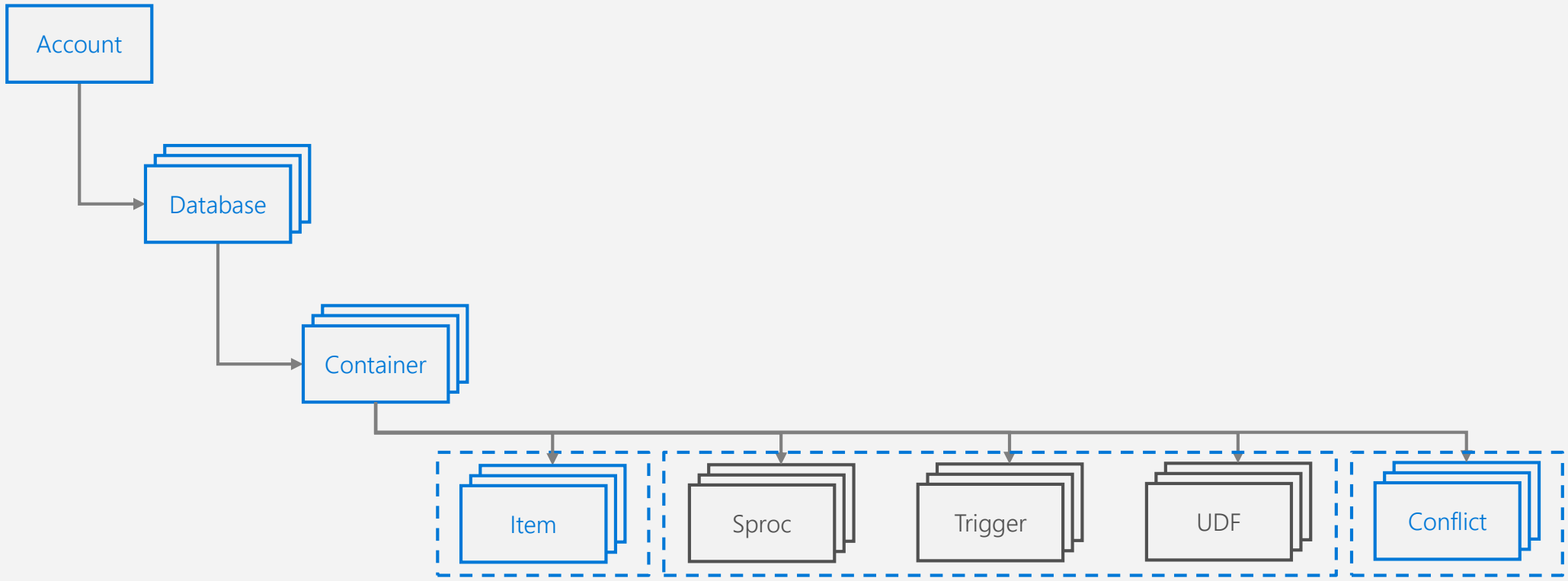
CONTAINER REPRESENTATIONS



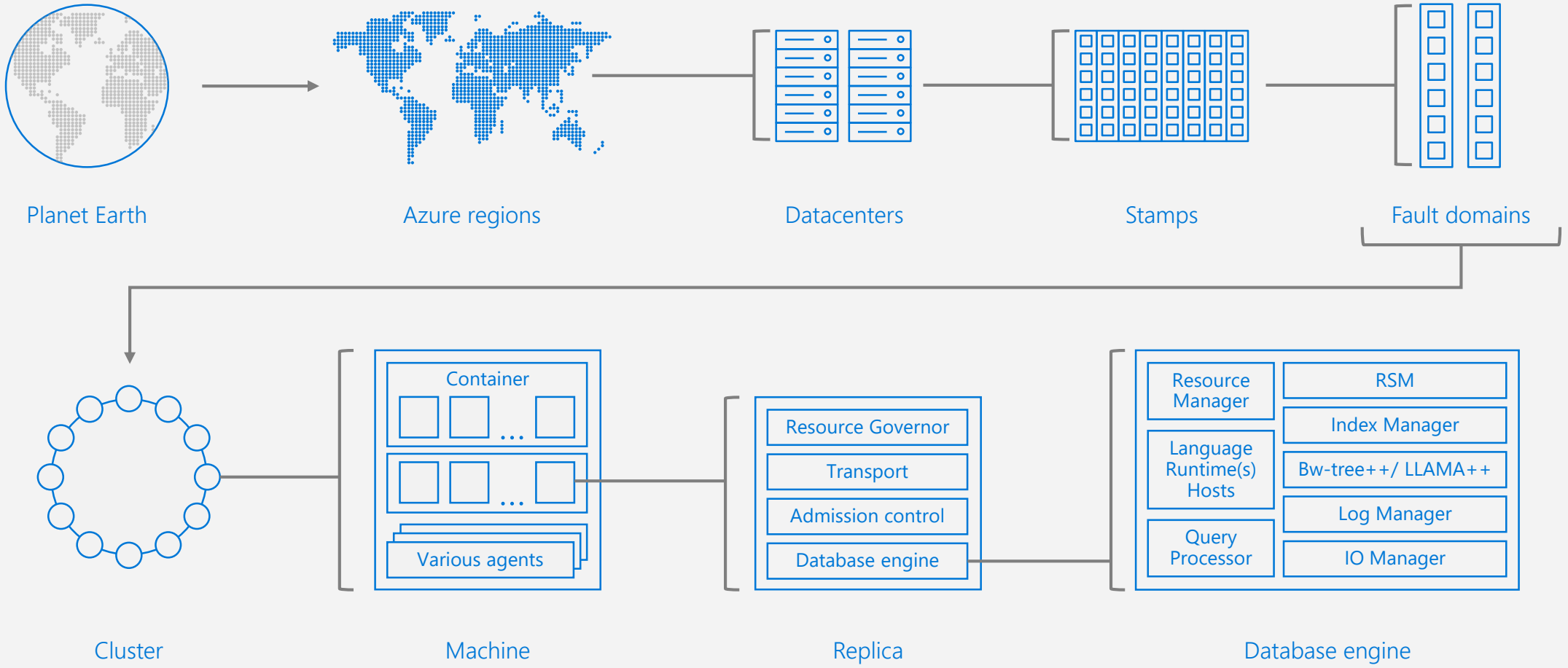
CREATING COLLECTIONS – SQL API



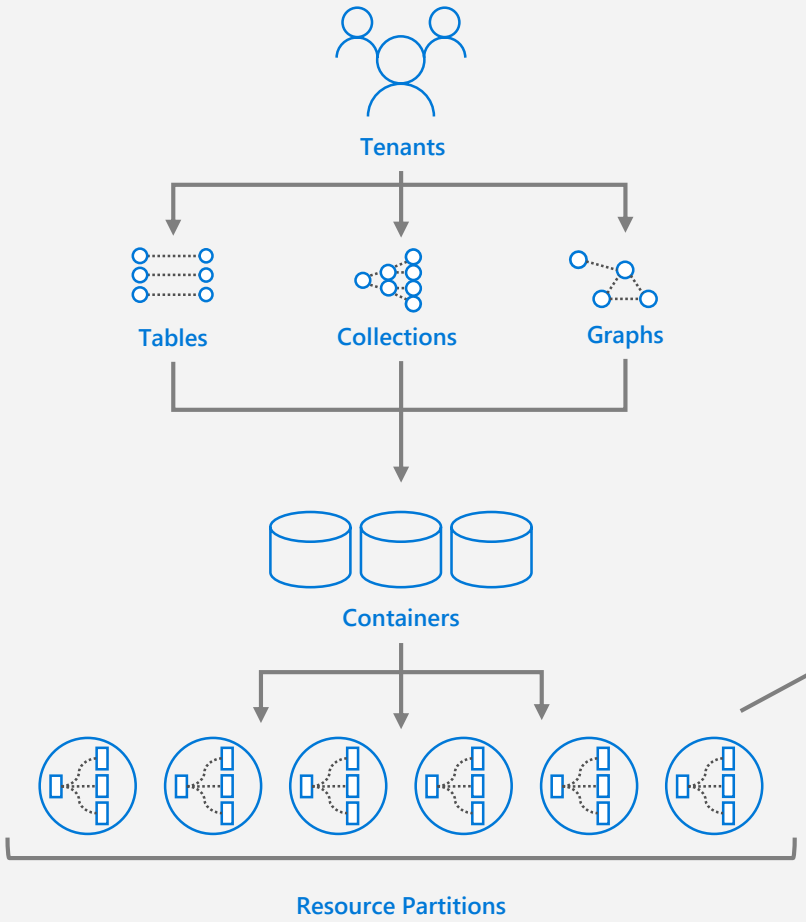
CONTAINER-LEVEL RESOURCES



SYSTEM TOPOLOGY (BEHIND THE SCENES)

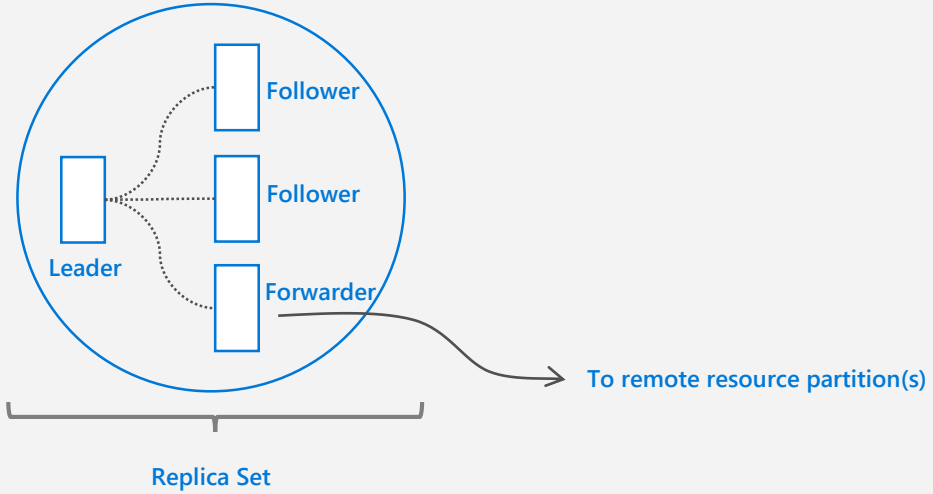


RESOURCE HIERARCHY



CONTAINERS

Logical resources “surfaced” to APIs as tables, collections or graphs, which are made up of one or more physical partitions or servers.



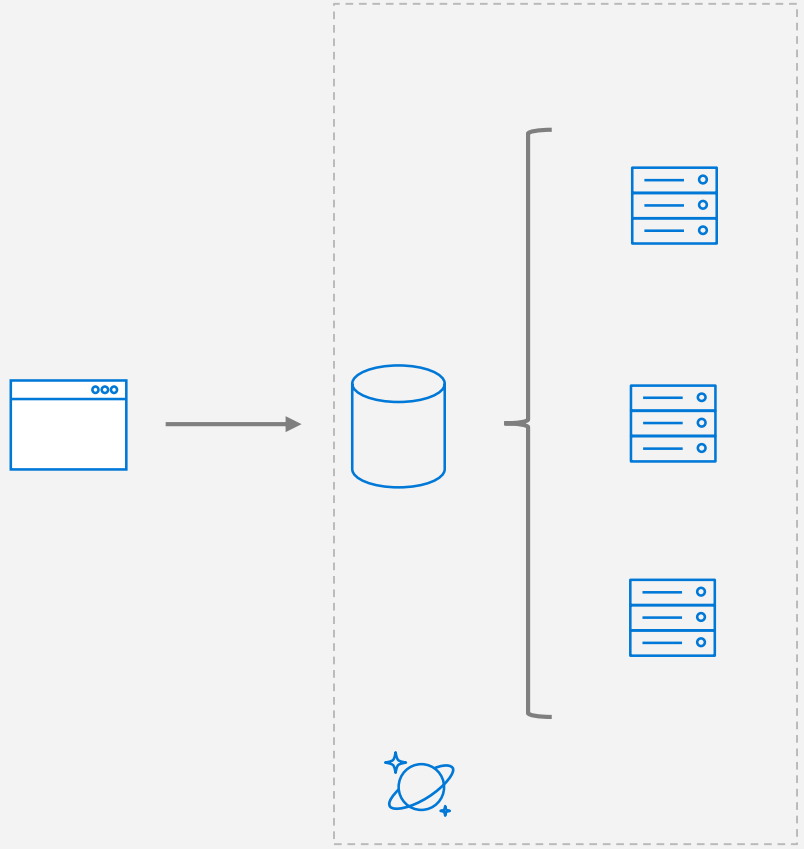
RESOURCE PARTITIONS

- Consistent, highly available, and resource-governed coordination primitives
- Consist of replica sets, with each replica hosting an instance of the database engine

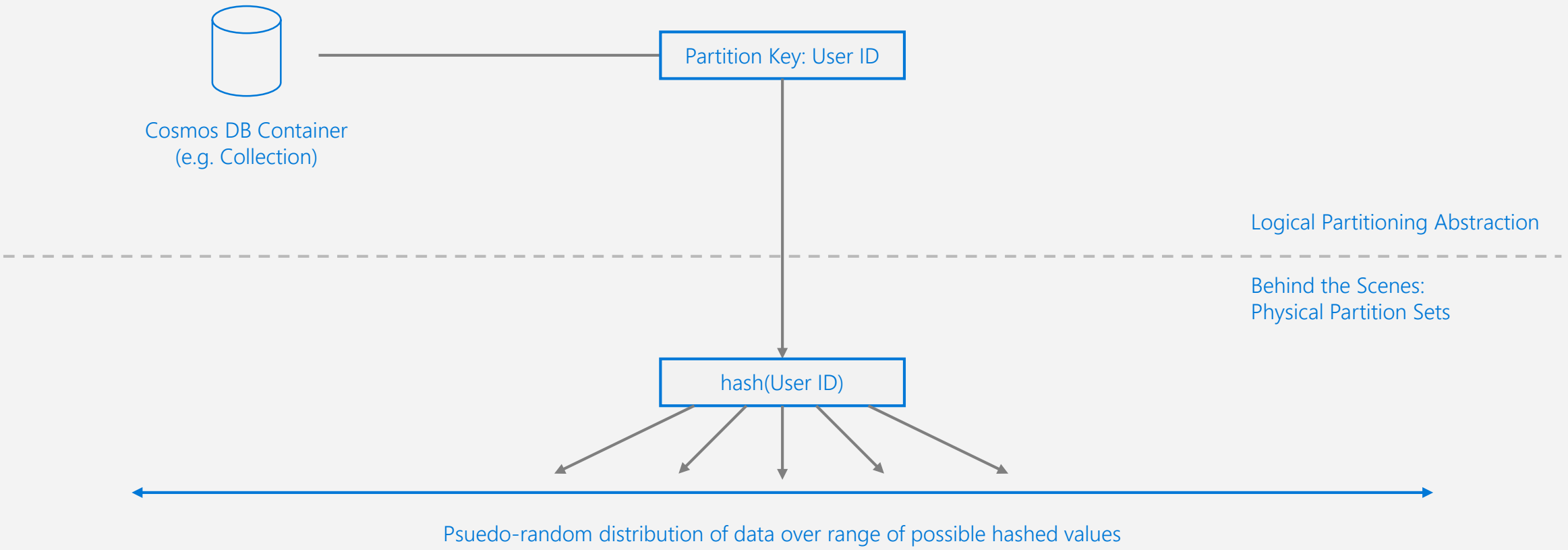
PARTITIONING

Leveraging Azure Cosmos DB to automatically scale your data across the globe

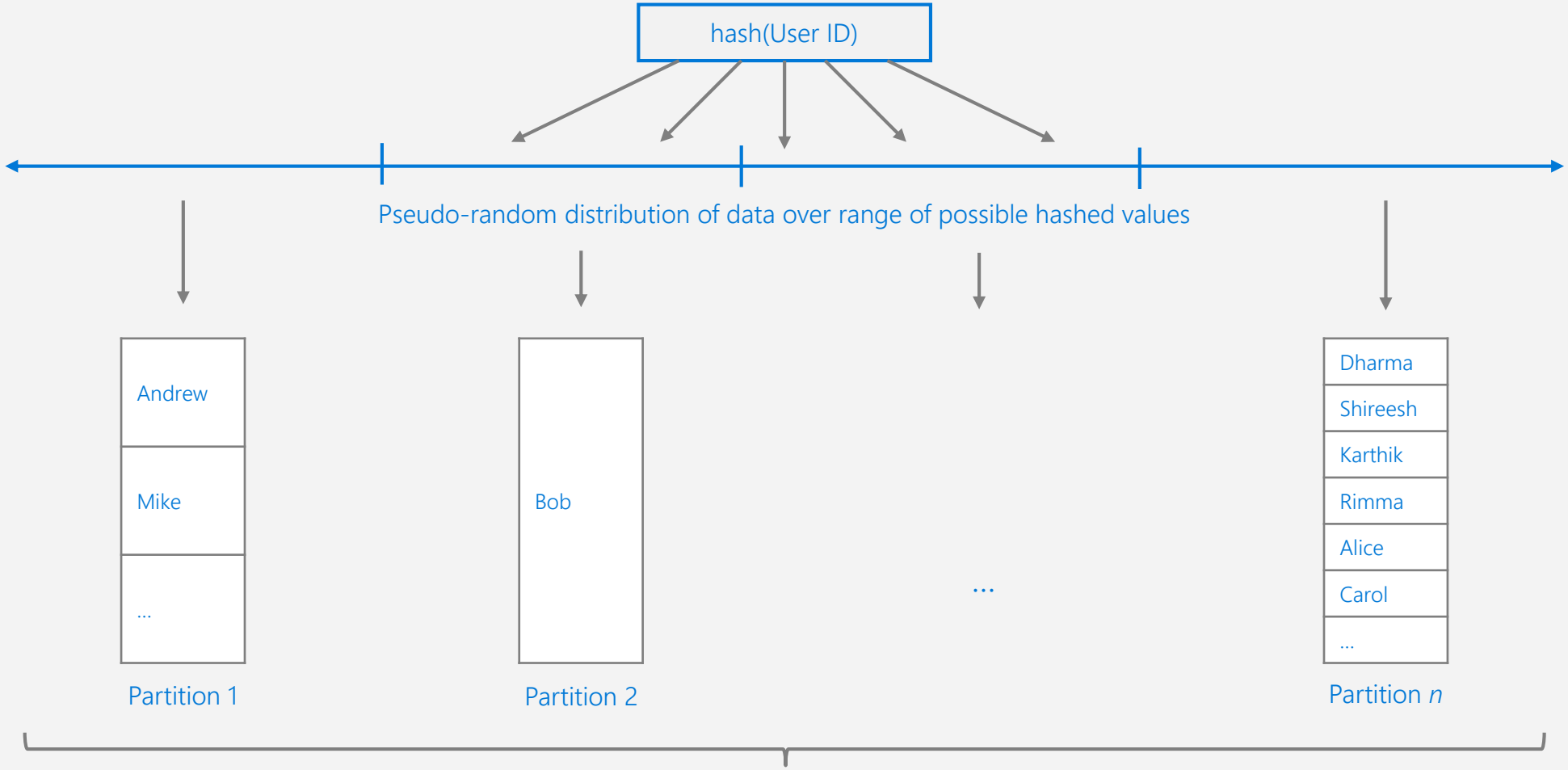
This module will reference partitioning in the context of all Azure Cosmos DB modules and APIs.



PARTITIONS

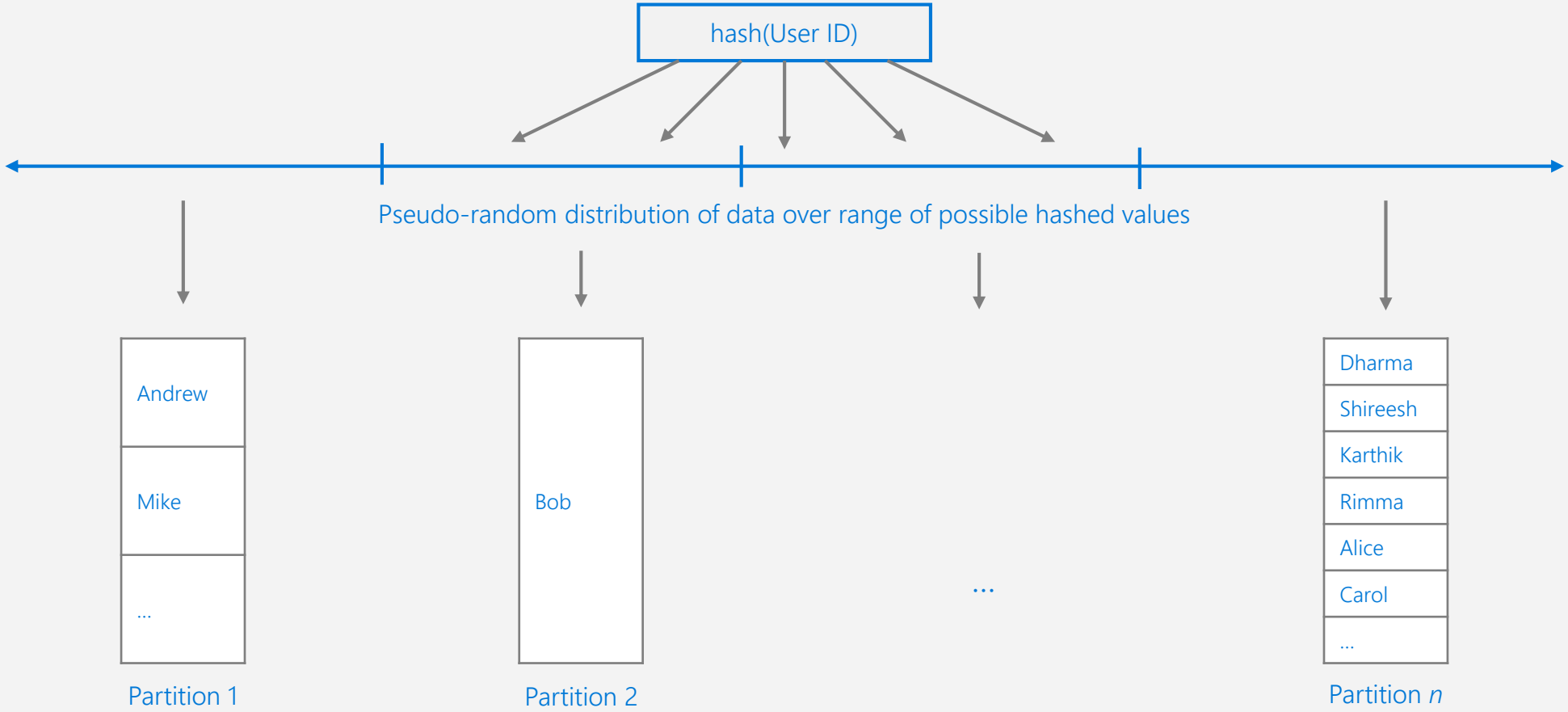

Cosmos DB Container
(e.g. Collection)
Partition Key: User ID
Logical Partitioning Abstraction
Behind the Scenes:
Physical Partition Sets
hash(User ID)
Psuedo-random distribution of data over range of possible hashed values

PARTITIONS



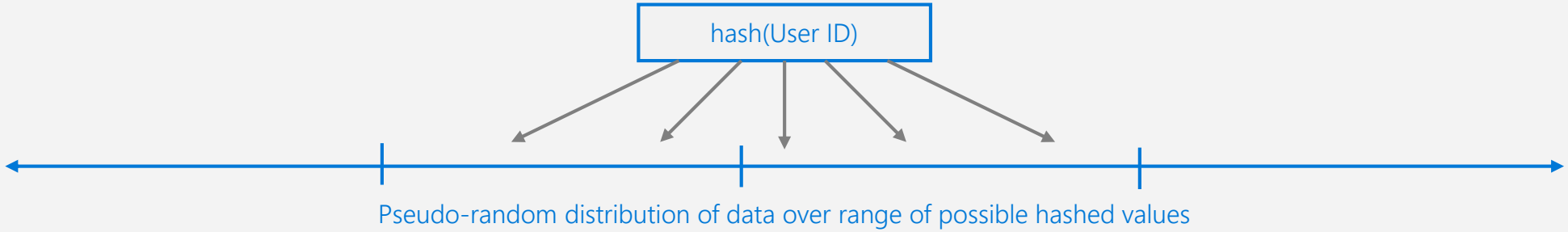
Frugal # of Partitions based on actual storage and throughput needs
(yielding scalability with low total cost of ownership)

PARTITIONS



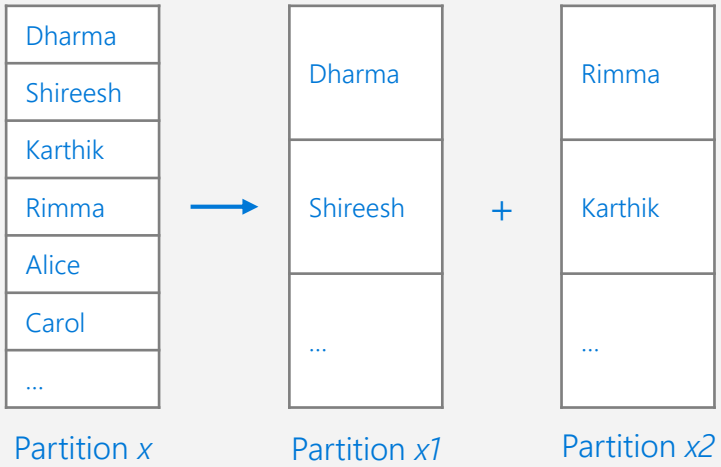
What happens when partitions need to grow?

PARTITIONS



Partition Ranges can be dynamically sub-divided to seamlessly grow database as the application grows while simultaneously maintaining high availability.

Partition management is fully managed by Azure Cosmos DB, so you don't have to write code or manage your partitions.



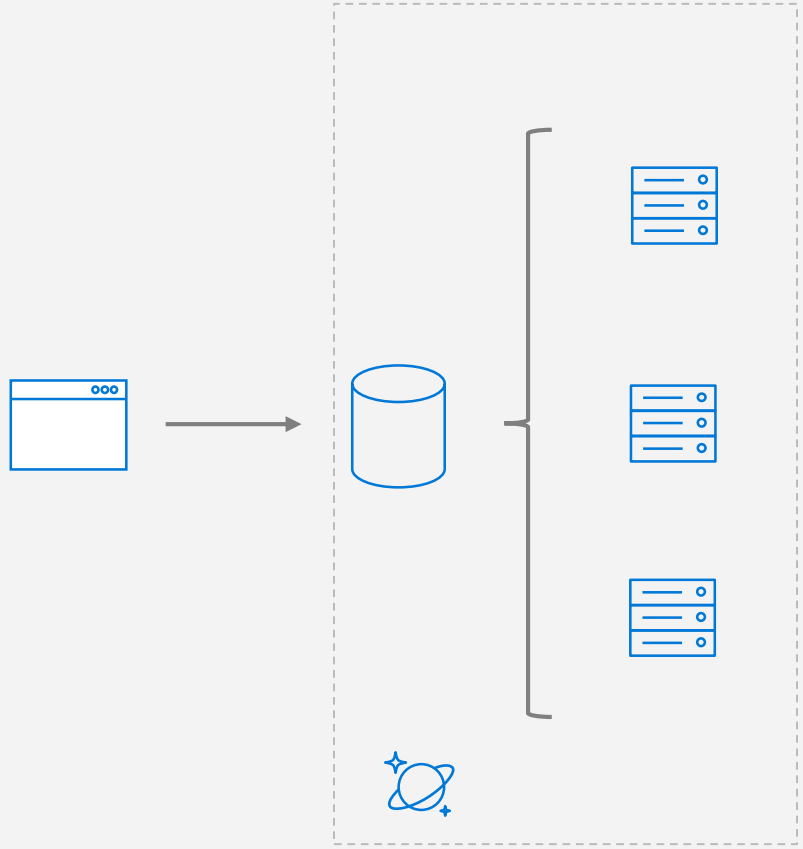
PARTITION DESIGN

IMPORTANT TO SELECT THE "RIGHT" PARTITION KEY

Partition keys acts as a **means for efficiently routing queries** and as a boundary for **multi-record** transactions.

KEY MOTIVATIONS

- Distribute Requests
- Distribute Storage
- Intelligently Route Queries for Efficiency



PARTITION DESIGN

EXAMPLE SCENARIO

Contoso Connected Car is a vehicle telematics company. They are planning to store vehicle telemetry data from millions of vehicles every second in Azure Cosmos DB to power predictive maintenance, fleet management, and driver risk analysis.

The partition key we select will be the scope for multi-record transactions.

WHAT ARE A FEW POTENTIAL PARTITION KEY CHOICES?

- Vehicle Model
- Current Time
- Device Id
- Composite Key – Device ID + Current Time

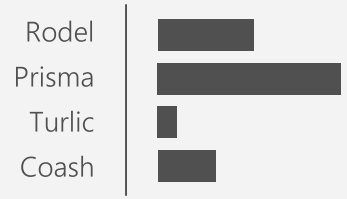
PARTITION KEY CHOICES

VEHICLE MODEL (e.g. Model A)

Most auto manufactures only have a couple dozen models. This will create a fixed number of logical partition key values; and is potentially the least granular option.

Depending how uniform sales are across various models – this introduces possibilities for hot partition keys on both storage and throughput.

Storage Distribution



Throughput Distribution

CURRENT MONTH (e.g. 2018-04)

Auto manufacturers have transactions occurring throughout the year. This will create a more balanced distribution of storage across partition key values.

However, most business transactions occur on recent data creating the possibility of a hot partition key for the current month on throughput.

Storage Distribution

Throughput Distribution

Example – Contoso Connected Car

PARTITION KEY CHOICES

DEVICE ID (e.g. Device123)

Each car would have a unique device ID. This creates a large number of partition key values and would have a significant amount of granularity.

Depending on how many transactions occur per vehicle, it is possible to a specific partition key that reaches the storage limit per partition key

Storage Distribution

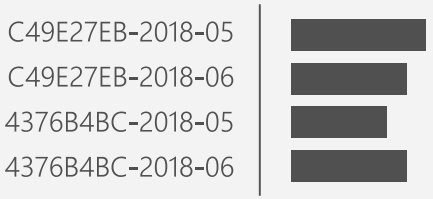
Throughput Distribution

COMPOSITE KEY (Device ID + Time)

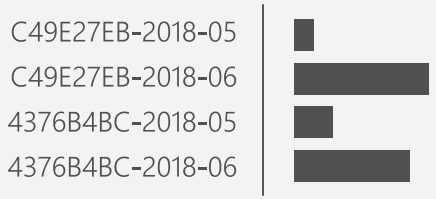
This composite option increases the granularity of partition key values by combining the current month and a device ID. Specific partition key values have less of a risk of hitting storage limitations as they only relate to a single month of data for a specific vehicle.

Throughput in this example would be distributed more to logical partition key values for the current month.

Storage Distribution



Throughput Distribution



Example – Contoso Connected Car

PARTITION GRANULARITY

SELECT THE "RIGHT" LEVEL OF GRANULARITY FOR YOUR PARTITIONS

Partitions should be based on your most often occurring query and transactional needs. The goal is to **maximize granularity** and **minimize cross-partition requests**.

Don't be afraid to have more partitions!

More partition keys = More scalability

Example – Contoso Connected Car

PARTITION GRANULARITY

SELECT THE "RIGHT" LEVEL OF GRANULARITY FOR YOUR PARTITIONS

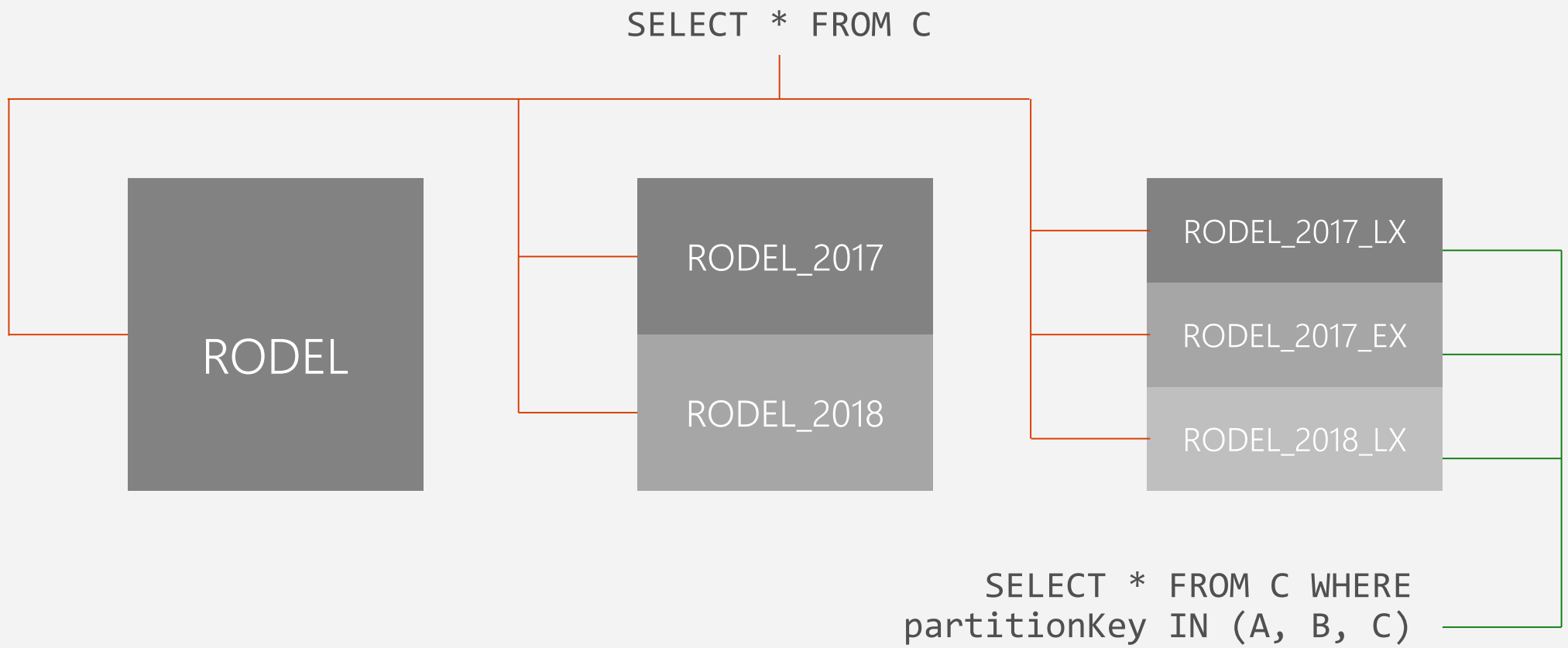
Don't be afraid to have more partitions!

More partition keys = More scalability

Example – Contoso Connected Car

PARTITION GRANULARITY

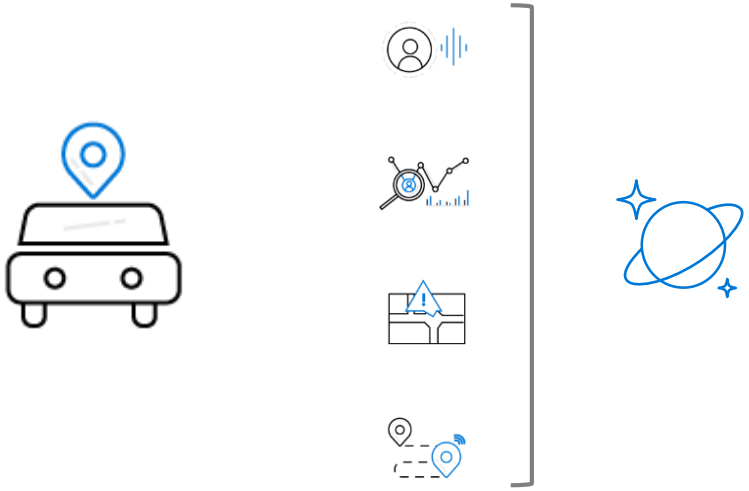
A CROSS-PARTITION QUERY IS NOT ALWAYS A BLIND FAN OUT QUERY

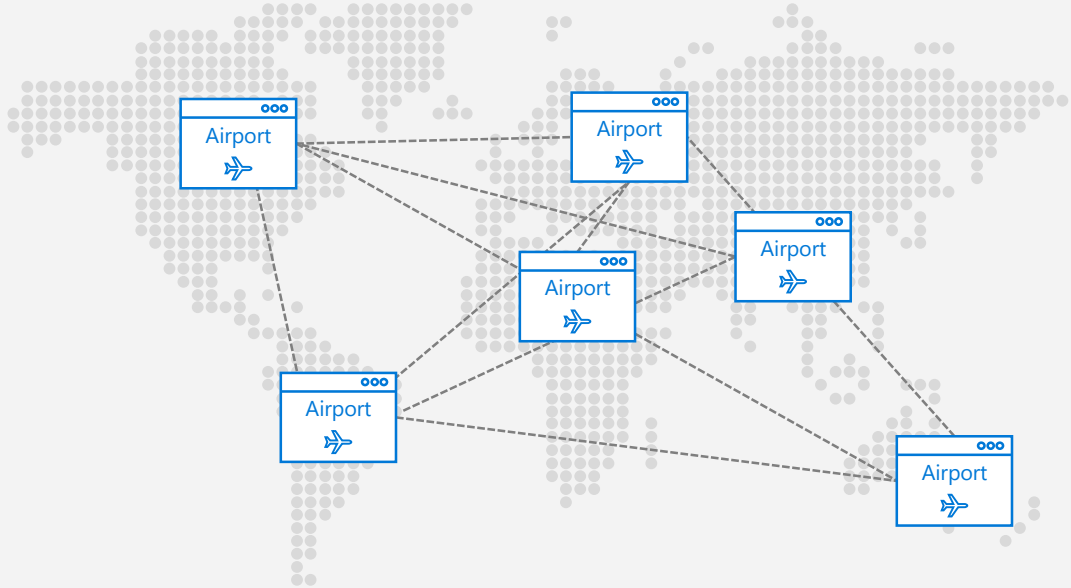


Example – Contoso Connected Car

PARTITION KEY SELECTION

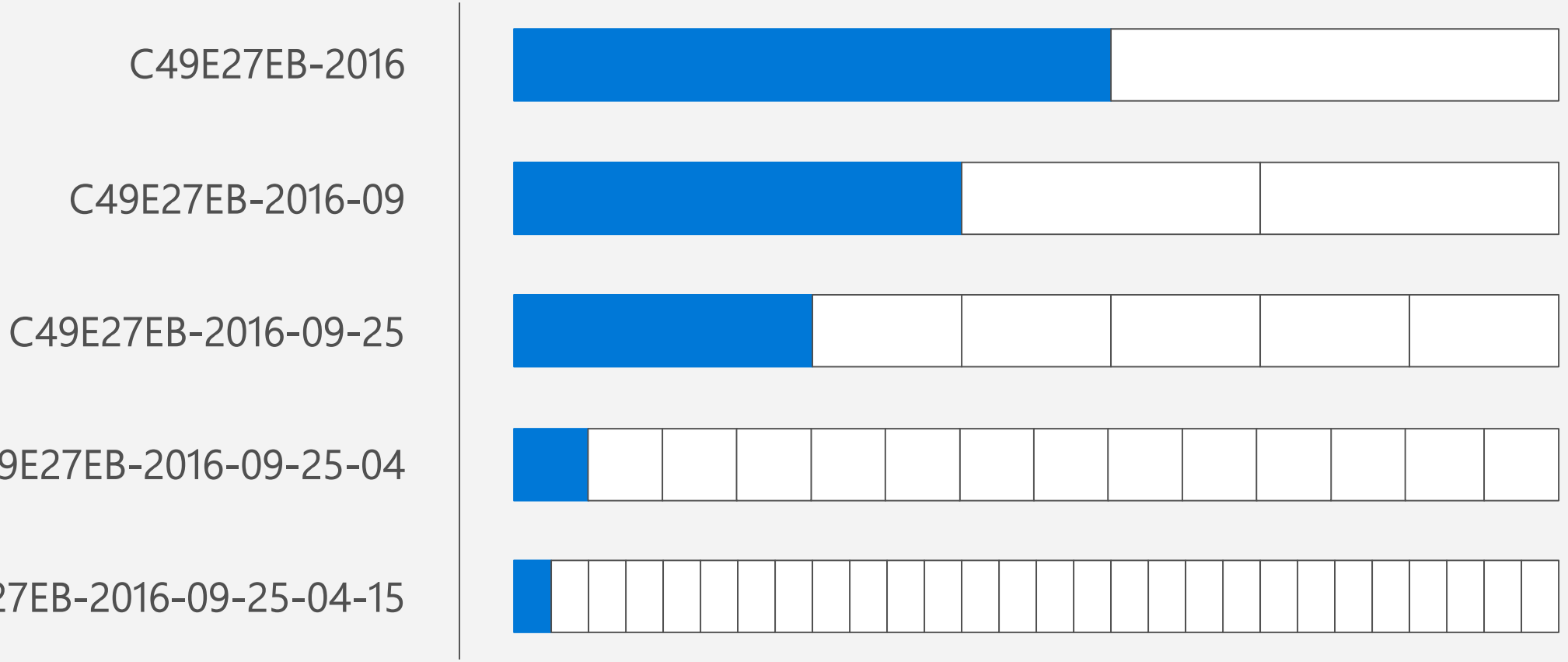
Contoso Connected Car is collecting and storing vehicle telemetry data from millions of vehicles. The team has decided to partition based on a composite key consisting of device id + current time when the interaction occurred.





PARTITION KEY SCENARIO

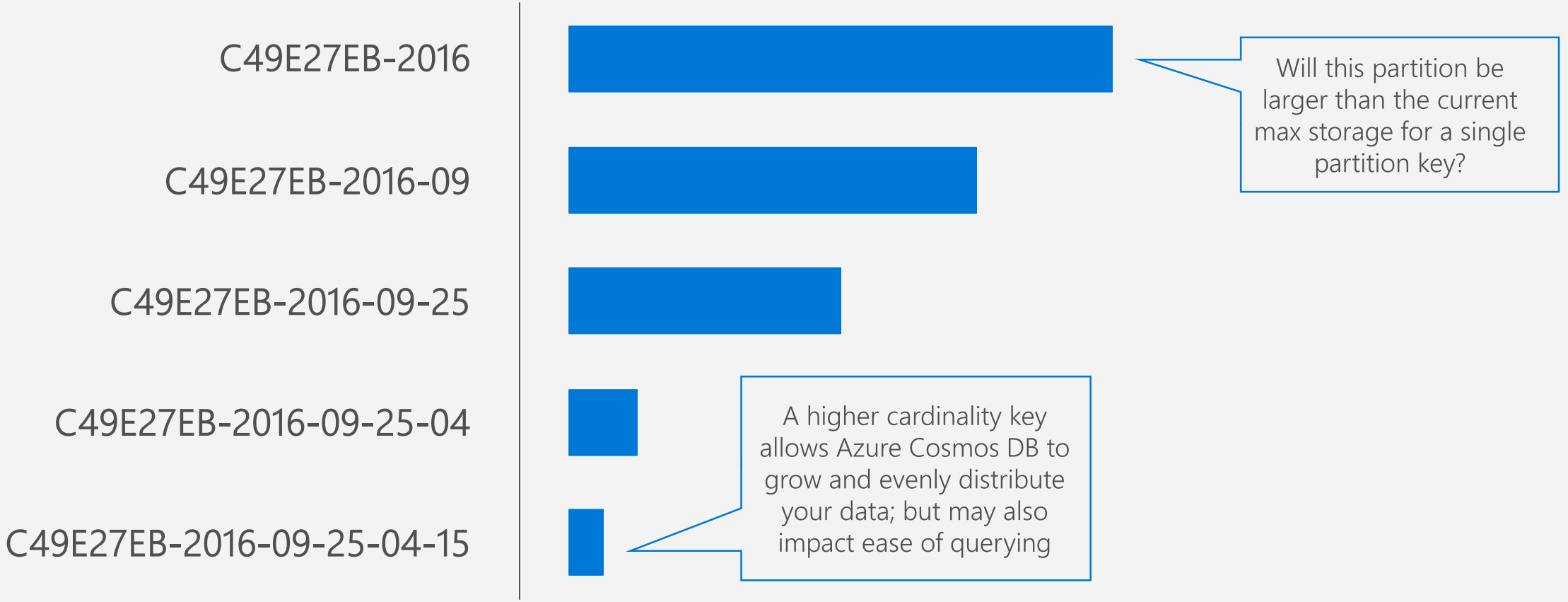
Interaction that occurred on: **September 25, 2016 at 4:15 AM UTC**



Example – Contoso Connected Car

PARTITION KEY SCENARIO

Interaction that occurred on: **September 25, 2016 at 4:15 AM UTC**



Example – Contoso Connected Car

PARTITIONS

Best Practices: Design Goals for Choosing a Good Partition Key

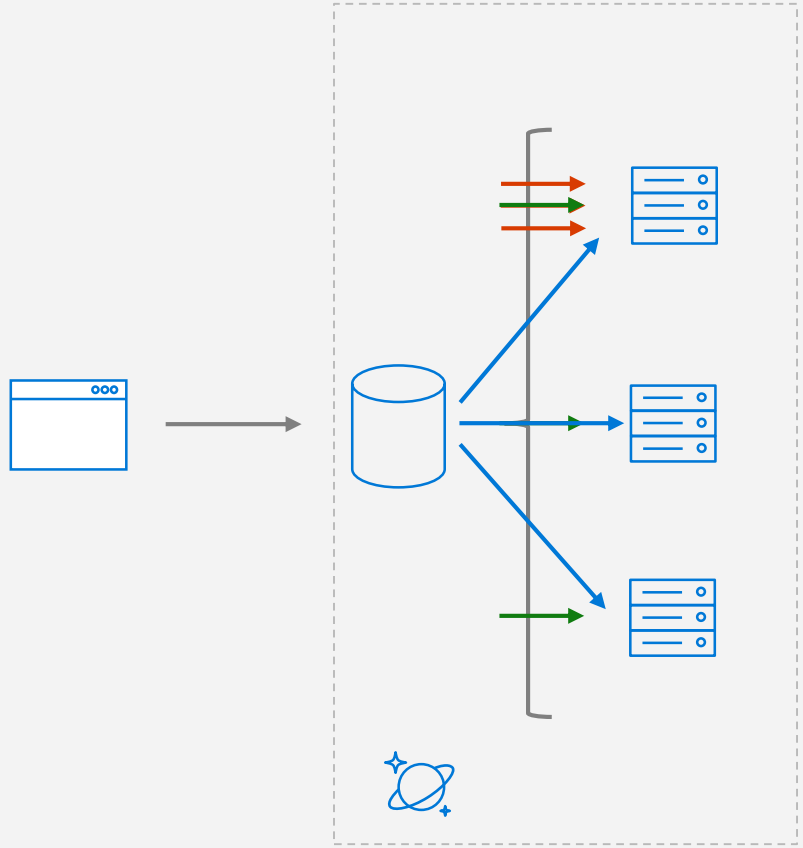
- Distribute the overall request + storage volume
  - Avoid "hot" partition keys
- Partition Key is scope for multi-record transactions and routing queries
  - Queries can be intelligently routed via partition key
  - Omitting partition key on query requires fan-out

Steps for Success

- Ballpark scale needs (size/throughput)
- Understand the workload
- # of reads/sec vs writes per sec
  - Use pareto principal (80/20 rule) to help optimize bulk of workload
  - For reads – understand top 3-5 queries (look for common filters)
  - For writes – understand transactional needs

General Tips

- Build a POC to strengthen your understanding of the workload and iterate (avoid analyses paralysis)
- Don't be afraid of having too many partition keys
  - Partitions keys are logical
  - More partition keys → more scalability



PARTITION KEY STORAGE LIMITS

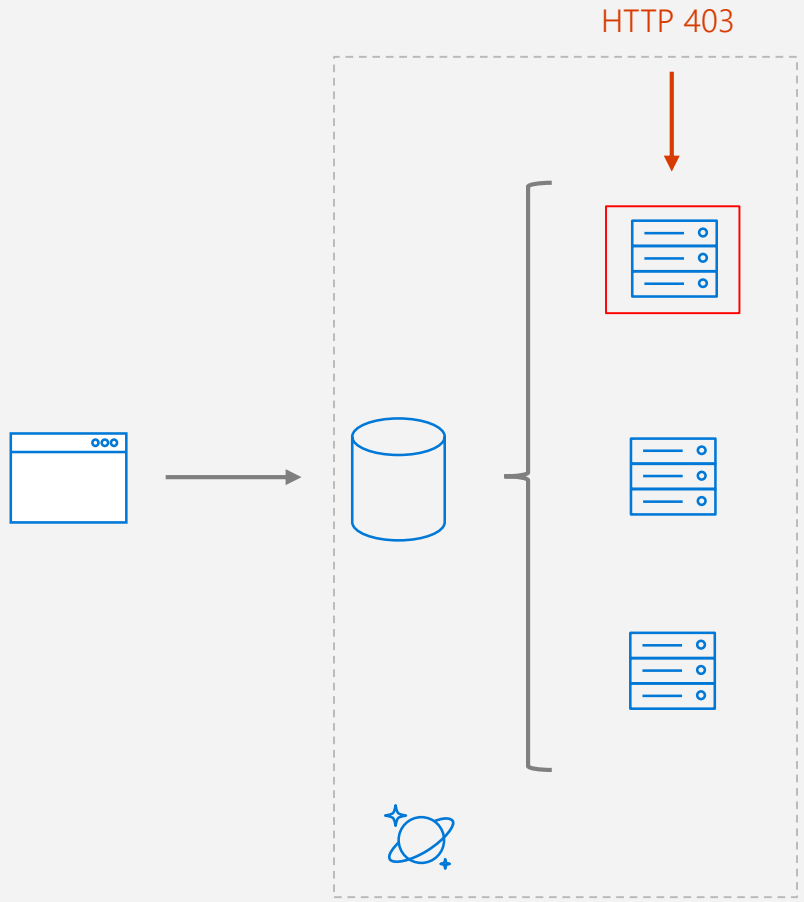
Containers support unlimited storage by dynamically allocating additional physical partitions

Storage for single partition key value (logical partition) is quota'ed to 10GB.

When a partition key reaches its provisioned storage limit, requests to create new resources will return a HTTP Status Code of 403 (Forbidden).

Azure Cosmos DB will automatically add partitions, and may also return a 403 if:

- An authorization token has expired
- A programmatic element (UDF, Stored Procedure, Trigger) has been flagged for repeated violations

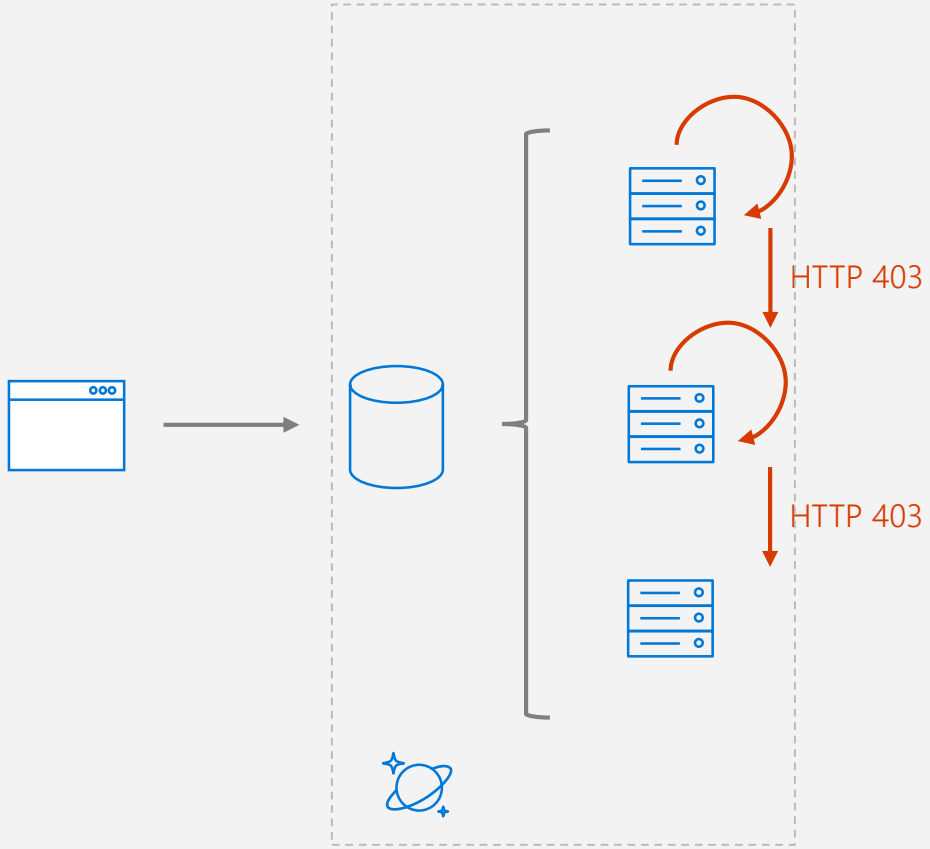


DESIGN PATTERNS FOR LARGE PARTITION KEYS

"LINKED LIST APPROACH" BY SREADING DATA ACROSS INCREMENTAL PARTITION KEY VALUES

For workloads that exceed quotas for a single partition key value, you can logically spread items across multiple partition keys within a container by using a suffix on the partition key value.

As a partition fills up, you can determine when to **increment** the partition key value by looking for the 403 status code in your application's logic.

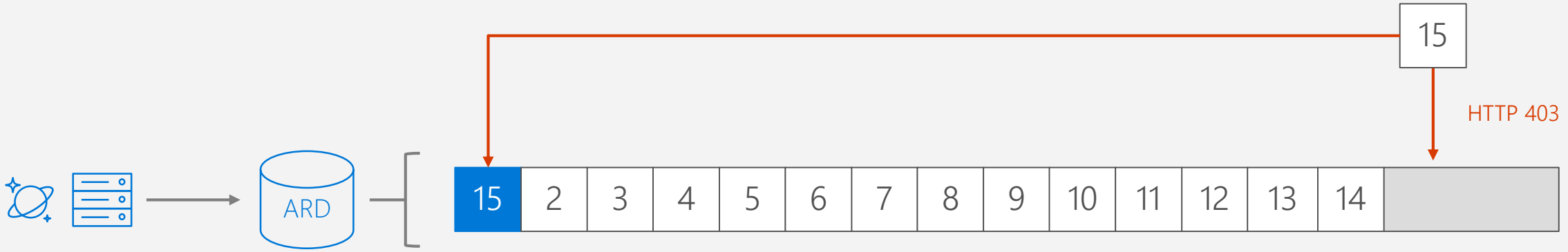


DESIGN PATTERNS FOR LARGE PARTITION KEYS

"CIRCULAR BUFFER" APPROACH BY REUSING UNIQUE IDS

As you insert new items into a container's partition, you can increment the unique id for each item in the partition.

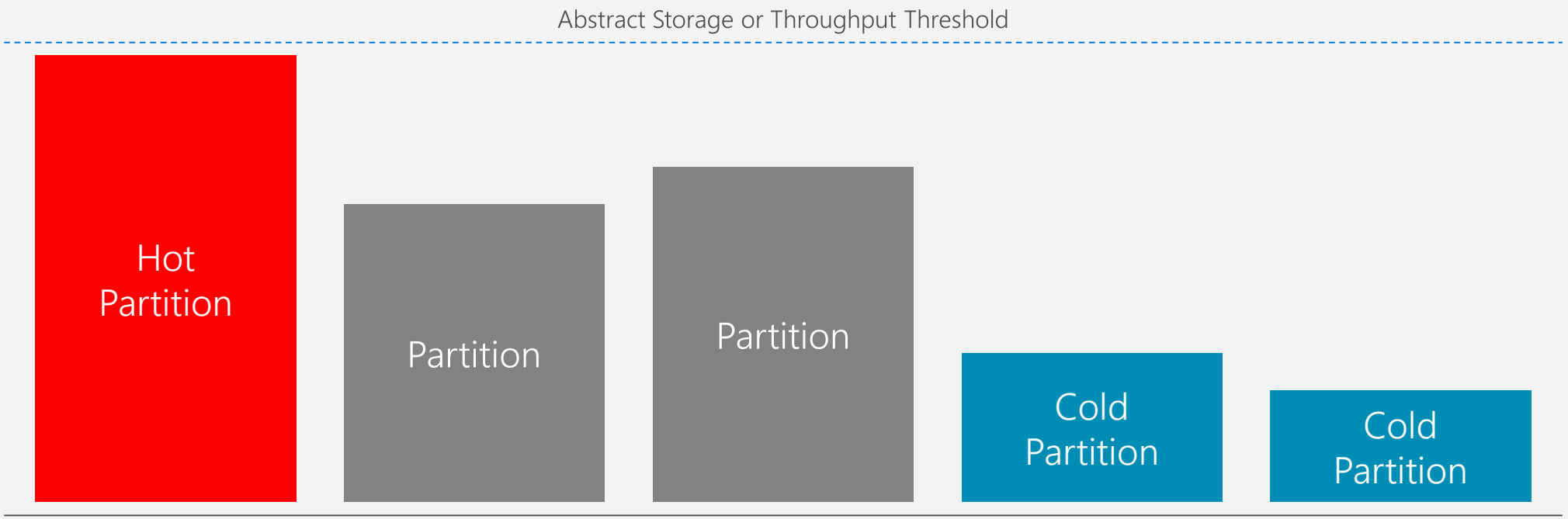
When you get a 403 status code, indicating the partition is full, you can restart your unique id and upsert the items to replace older documents.



HOT/COLD PARTITIONS

PARTITION USAGE CAN VARY OVER TIME

Partitions that are approaching thresholds are referred to as **hot**. Partitions that are underutilized are referred to as **cold**.



QUERY FAN-OUT

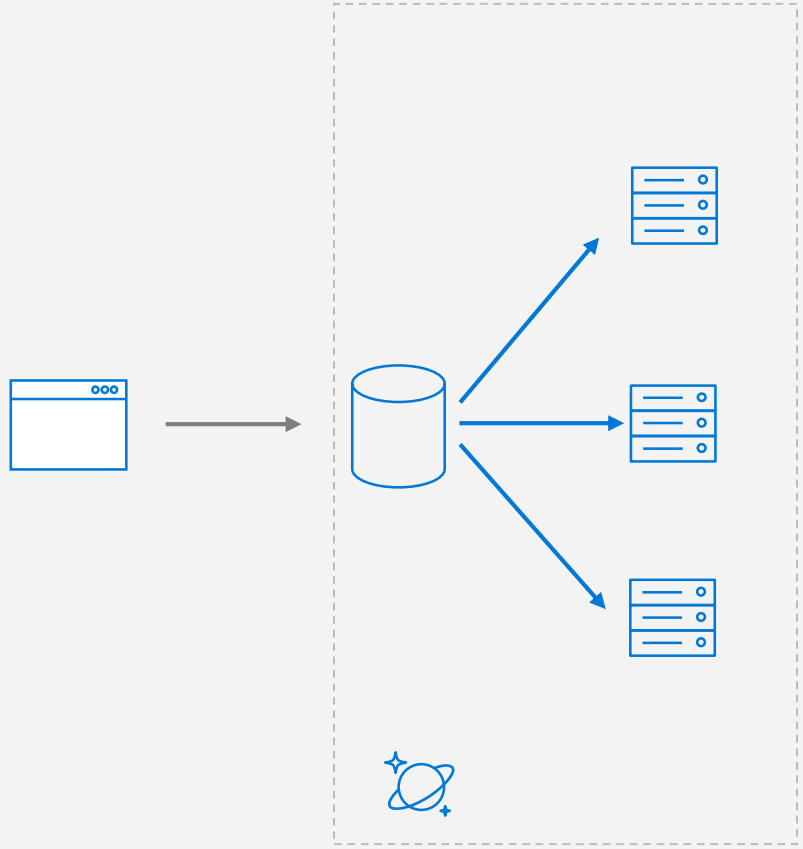
CROSS-PARTITION QUERIES CAN BE PERFORMED SERVER-SIDE OR CLIENT-SIDE

Cross-partition queries are opt-in

- Cross-partition queries can be tuned and parallelized

Creates a bottleneck

- Must wait for all partitions to return before the query is "done"



CROSS-PARTITION SDK EXAMPLE

```
IQueryable<DeviceReading> crossPartitionQuery = client.CreateDocumentQuery<DeviceReading>(
    UriFactory.CreateDocumentCollectionUri("db", "coll"),
    new FeedOptions {
          EnableCrossPartitionQuery = true,
          MaxDegreeOfParallelism = 10,
          MaxBufferedItemCount = 100
    })
    .Where(m => m.MetricType == "Temperature" && m.MetricValue > 100)
    .OrderBy(m => m.MetricValue);
```

.NET

CROSS-PARTITION SDK EXAMPLE

```
var querySpec = {
          query: 'SELECT * FROM container c’
};

var feedOptions = {
          enableCrossPartitionQuery = true
          maxDegreeOfParallelism = 10
};

client.queryDocuments(collectionLink, querySpec, feedOptions)
          .toArray(function (err, results) {
          }
```

CROSS-PARTITION SDK EXAMPLE

```
FeedOptions options = new FeedOptions();
options.setEnableCrossPartitionQuery(true);

Iterator<Document> it = client.queryDocuments(
    collectionLink,
    "SELECT * from r",
    options
).getQueryIterator();
```