DP-203 Microsoft Azure Data Engineer

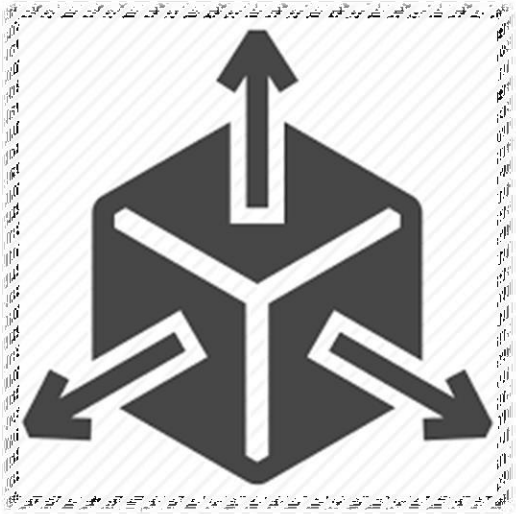
Day5 - NoSQL – CosmosDB(cont…)

29th July 2021

Vinodkumar Bhovi

databoq®

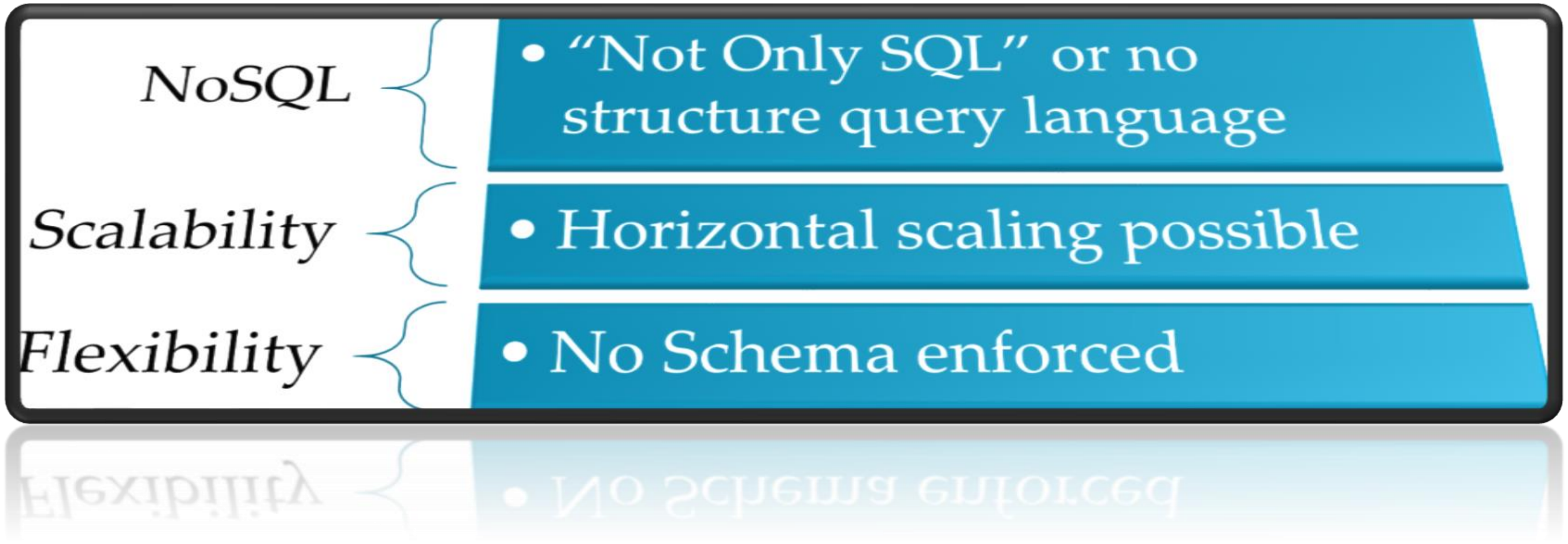
RDBMS were lacking



Scalability

Flexibility

What is NoSQL



NoSQL

- "Not Only SQL" or no structure query language

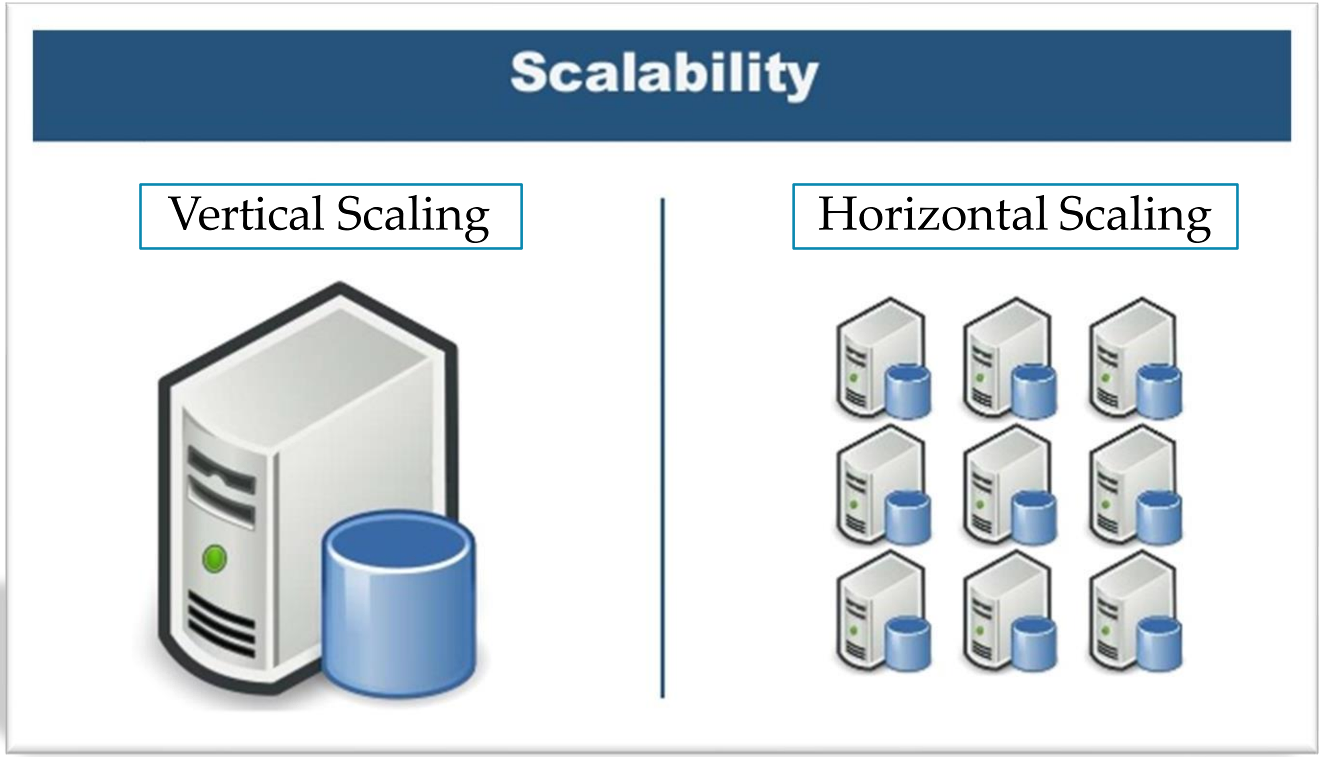
Scalability

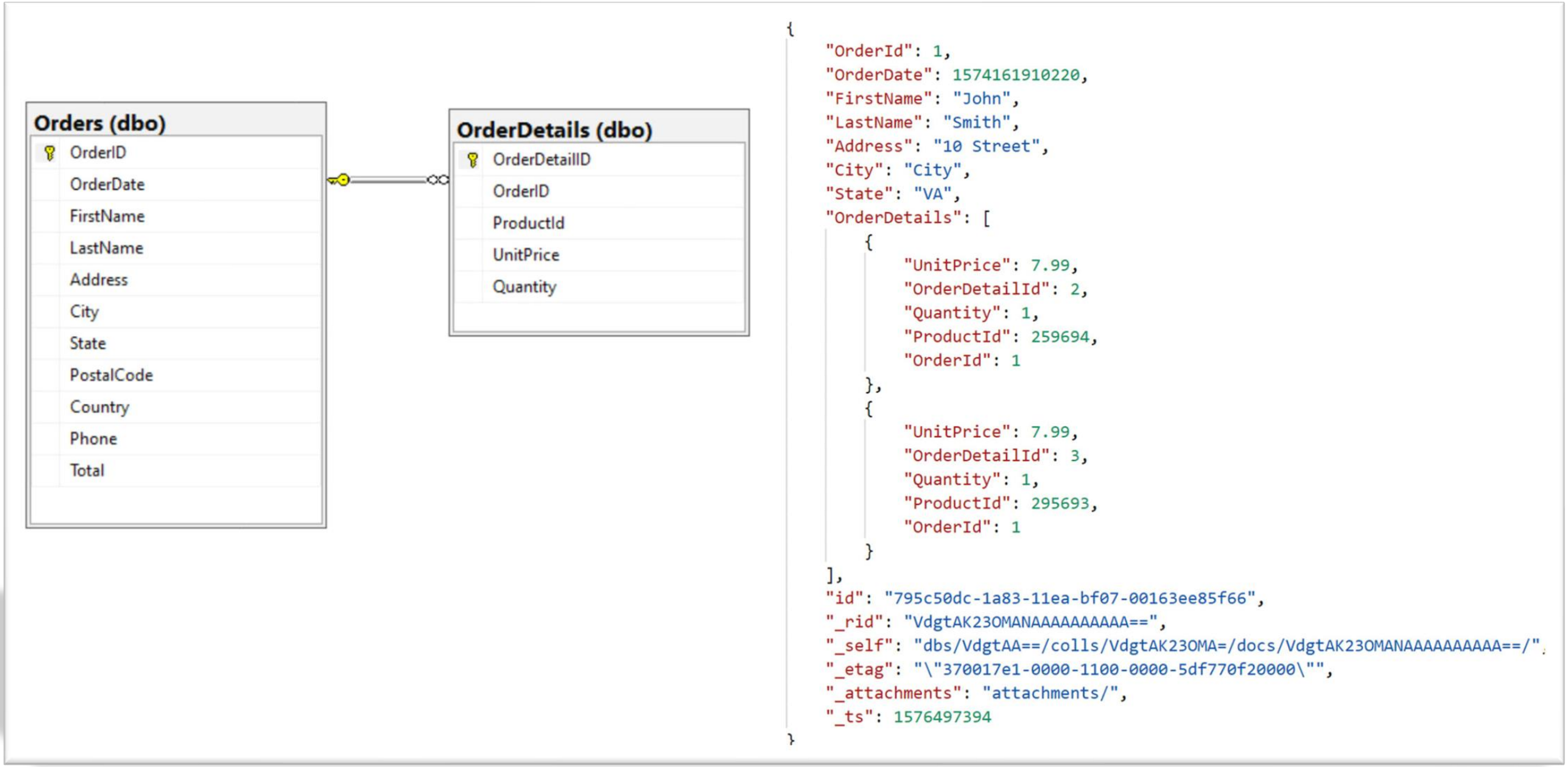
- Horizontal scaling possible

Flexibility

- No Schema enforced

- Vertical scaling
  - Add more CPU, RAM, HDD in same system
- Horizontal Scaling
  - Add more commodity machines in system





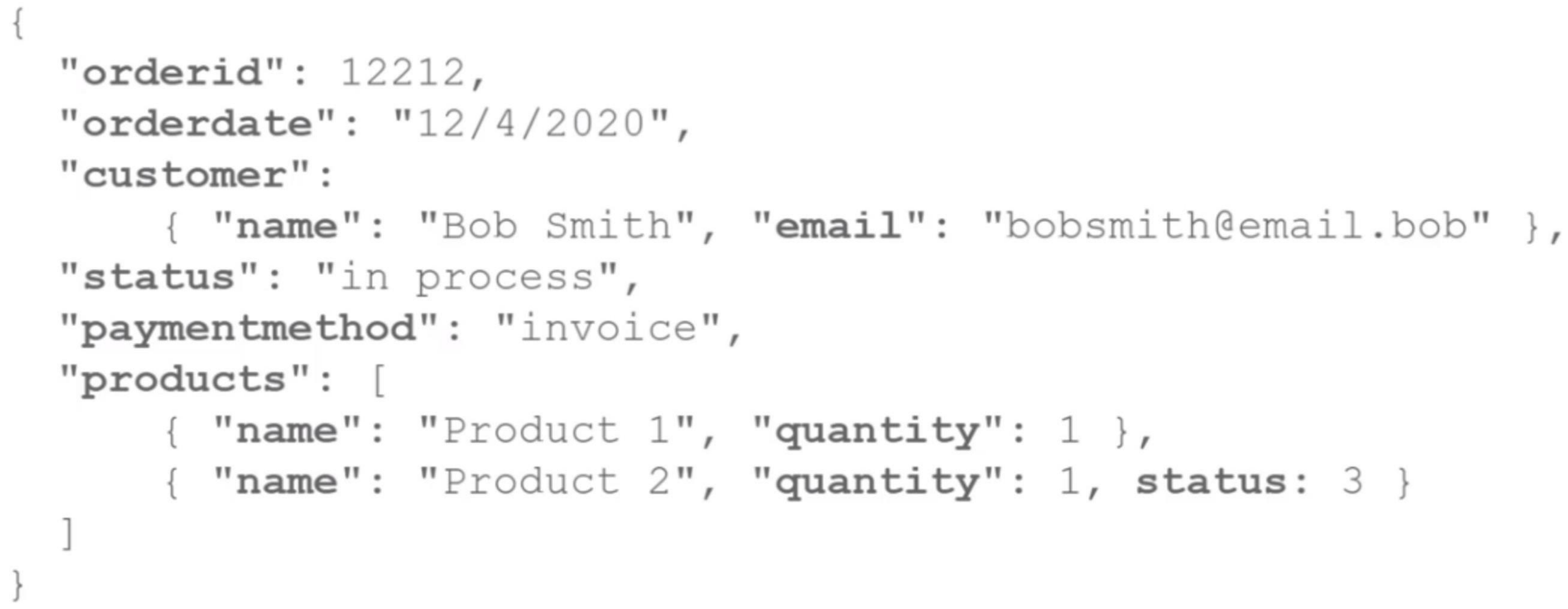
Orders (dbo)

- 🔑 OrderID
- OrderDate
- FirstName
- LastName
- Address
- City
- State
- PostalCode
- Country
- Phone
- Total

OrderDetails (dbo)

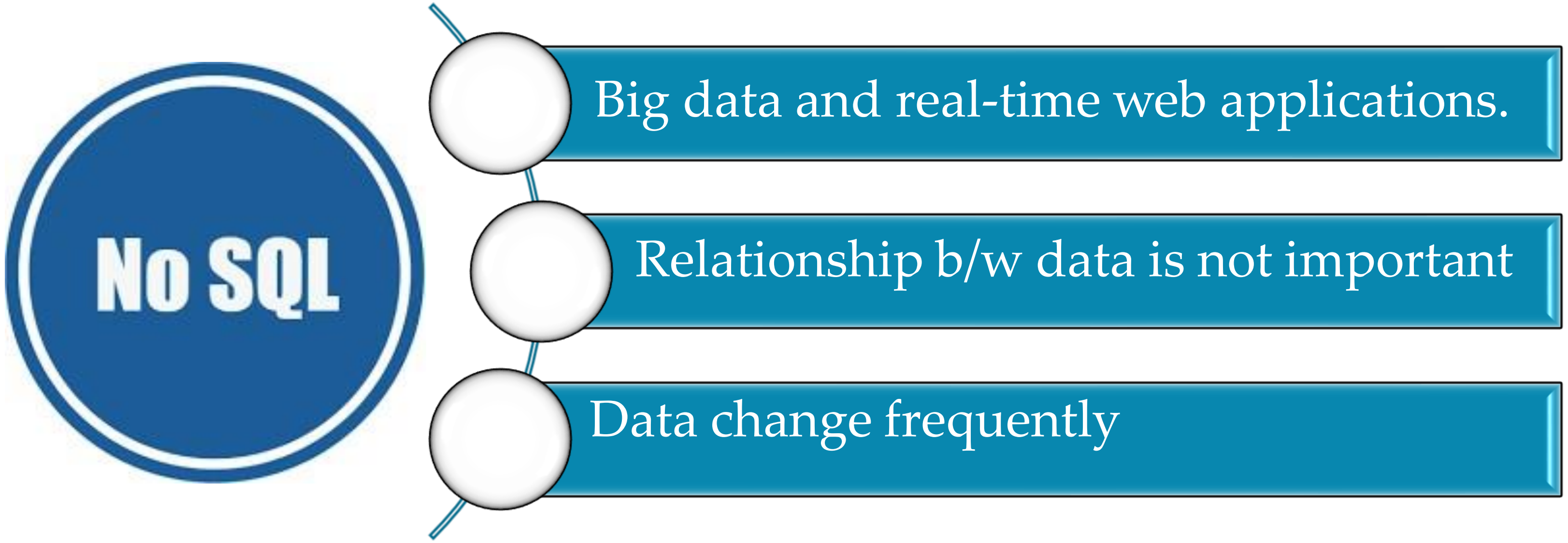
- 🔑 OrderDetailID
- OrderID
- ProductId
- UnitPrice
- Quantity

```
{
    "OrderId": 1,
    "OrderDate": 1574161910220,
    "FirstName": "John",
    "LastName": "Smith",
    "Address": "10 Street",
    "City": "City",
    "State": "VA",
    "OrderDetails": [
        {
            "UnitPrice": 7.99,
            "OrderDetailId": 2,
            "Quantity": 1,
            "ProductId": 259694,
            "OrderId": 1
        },
        {
            "UnitPrice": 7.99,
            "OrderDetailId": 3,
            "Quantity": 1,
            "ProductId": 295693,
            "OrderId": 1
        }
    ],
    "id": "795c50dc-1a83-11ea-bf07-00163ee85f66",
    "_rid": "VdgtAK23OMANAAAAAAAAAA==",
    "_self": "dbs/VdgtAA==/colls/VdgtAK23OMA=/docs/VdgtAK23OMANAAAAAAAAAA==/",
    "_etag": "\"370017e1-0000-1100-0000-5df770f20000\"",
    "_attachments": "attachments/",
    "_ts": 1576497394
}
```



```
{
  "orderid": 12212,
  "orderdate": "12/4/2020",
  "customer":
      { "name": "Bob Smith", "email": "bobsmith@email.bob" },
  "status": "in process",
  "paymentmethod": "invoice",
  "products": [
      { "name": "Product 1", "quantity": 1 },
      { "name": "Product 2", "quantity": 1, status: 3 }
  ]
}
```

NoSQL Use Cases



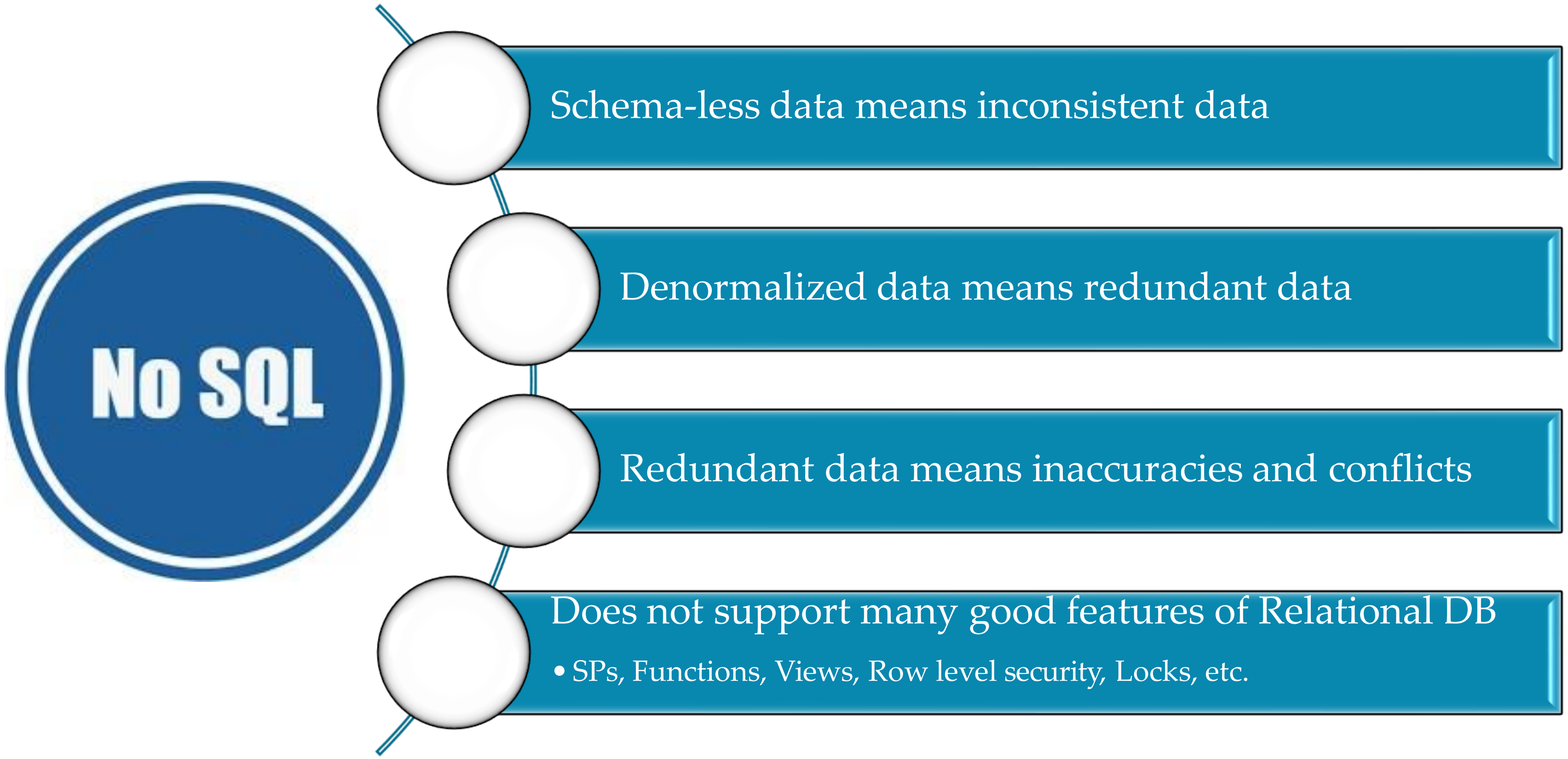
No SQL

Big data and real-time web applications.

Relationship b/w data is not important

Data change frequently

NoSQL Limitations

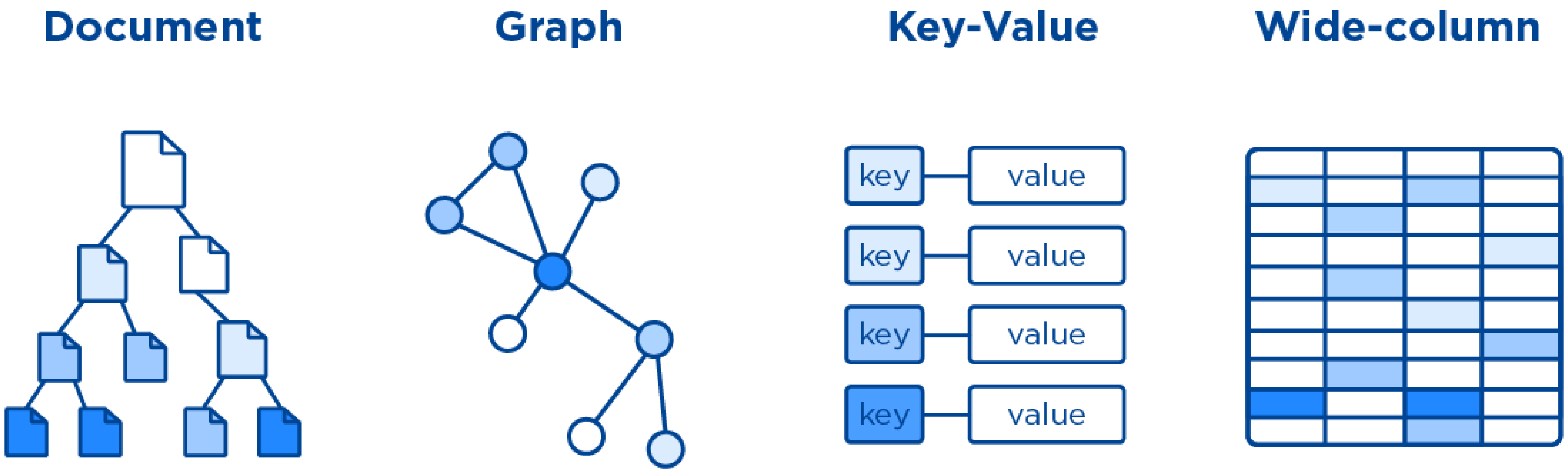


- Schema-less data means inconsistent data
- Denormalized data means redundant data
- Redundant data means inaccuracies and conflicts
- Does not support many good features of Relational DB
  - SPs, Functions, Views, Row level security, Locks, etc.

SQL vs NoSQL

SQL	NoSQL
• Relational database	• Non-relational or distributed
• Fixed schema	• Dynamic
• Designed for complex queries	• Not for complex queries
• SQL, MySql, Oracle, Postgres	• MongoDB, Redis, Hbase
• Vertical scaling	• Horizontal scaling
• Row Oriented	• Multi-model oriented
• Tables	• Collections
• Limited for big data	• Great for big data

4 Types of NoSQL Databases



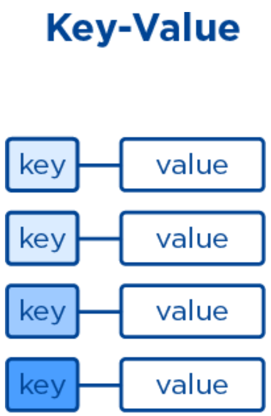
Document

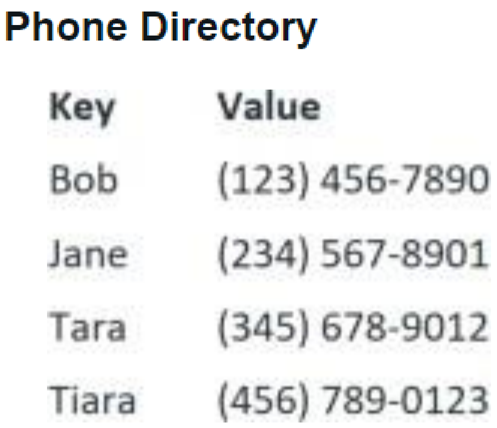
Graph

Key-Value

Wide-column

Key-value store





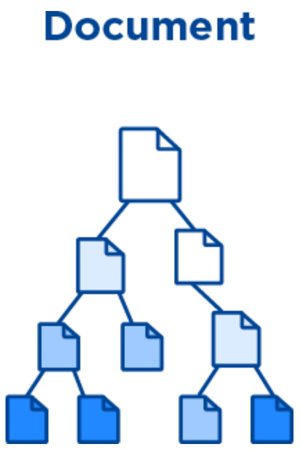
Phone Directory

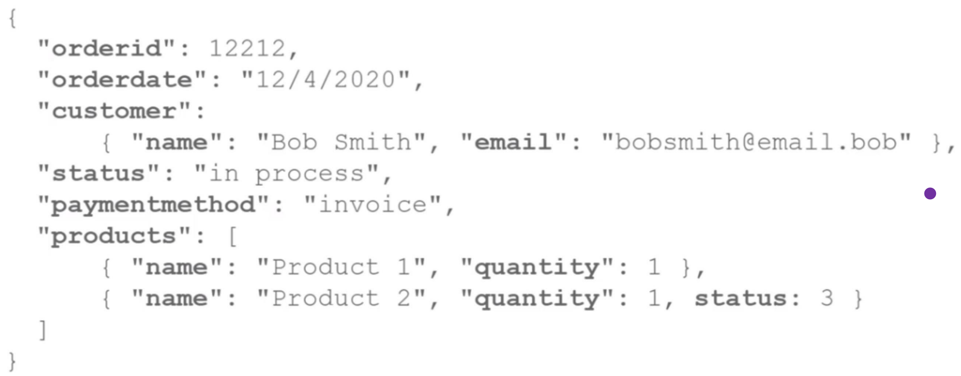
Key	Value
Bob	(123) 456-7890
Jane	(234) 567-8901
Tara	(345) 678-9012
Tiara	(456) 789-0123

- Uses a simple key/value to store data
- Quick to query due to its simplicity
- Value can be JSON, BLOB, String etc.
- Use Cases:
  - User profiles and session info on a website, blog comments, telecom directories, IP forwarding tables, shopping cart contents on e- commerce sites, and more.
- Examples
  - Cosmos DB Table API, Redis, Table Storage, Oracle NoSQL Database,

    Voldemorte, Aerospike, Oracle Berkeley DB

Document store

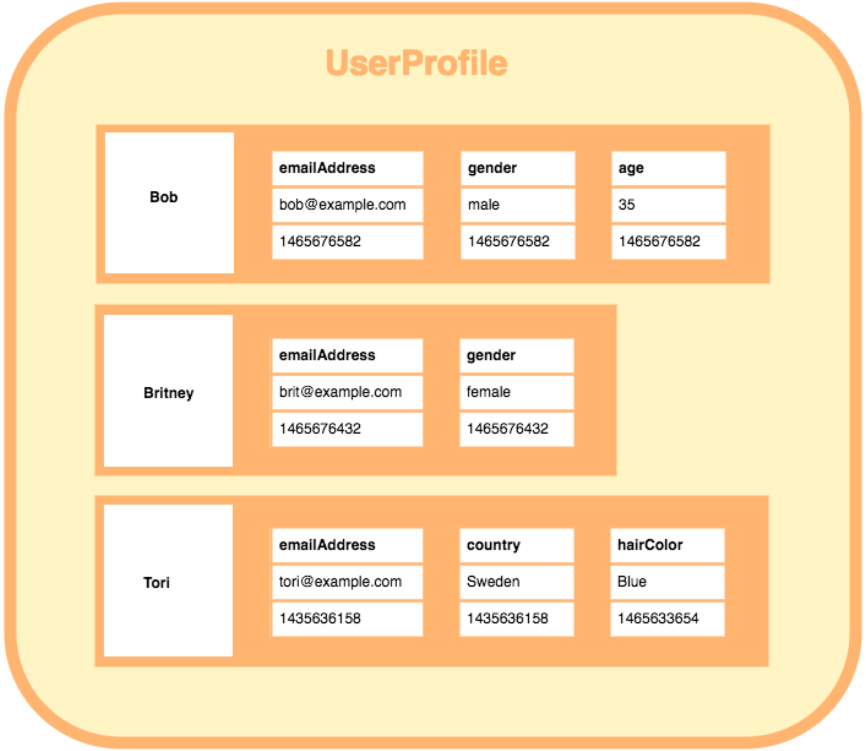




```
{
  "orderid": 12212,
  "orderdate": "12/4/2020",
  "customer":
      { "name": "Bob Smith", "email": "bobsmith@email.bob" },
  "status": "in process",
  "paymentmethod": "invoice",
  "products": [
      { "name": "Product 1", "quantity": 1 },
      { "name": "Product 2", "quantity": 1, status: 3 }
  ]
}
```

- Document-oriented model to store data
- Similar to key/value store, difference is that, the value in a document store database consists of semi-structured data.
- Each record and its associated data within a single document.
- Document stores are usually XML, JSON, BSON, YAML, etc.
- Use Cases:
  - Content management systems, blogging platforms, and other web applications, blog comments, chat sessions, tweets, ratings, etc.
- Examples
  - Cosmos DB, MongoDB, DocumentDB, CouchDB, MarkLogic, OrientDB

Column store



UserProfile

Bob	emailAddress	gender	age
	bob@example.com	male	35
	1465676582	1465676582	1465676582

Britney	emailAddress	gender
	brit@example.com	female
	1465676432	1465676432

Tori	emailAddress	country	hairColor
	tori@example.com	Sweden	Blue
	1435636158	1435636158	1465633654

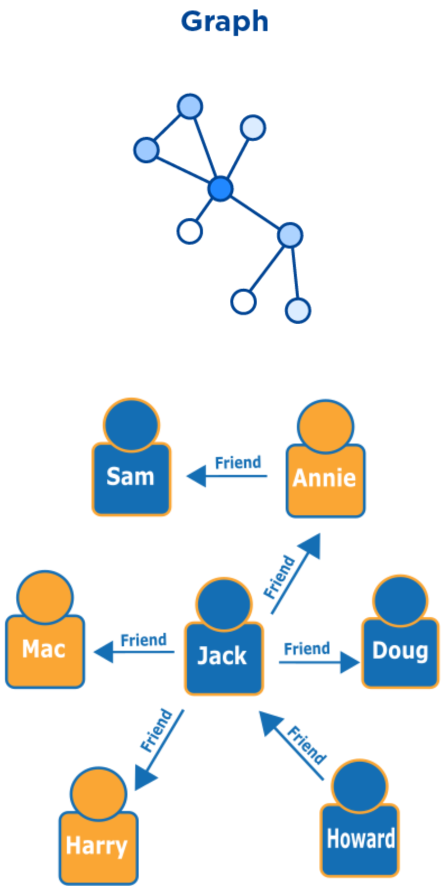
- Stores data using a column oriented model
- Columns in each row are contained within that row
- Each row can have different columns to the other rows.
- Extremely quick to load and query
- Use Cases:
  - Sensor Logs [Internet of Things (IOT)], User preferences, Geographic

    information, Reporting systems, Time Series Data, Logging

    and other write heavy applications
- Examples
  - Cosmos DB, Bigtable, Cassandra, Hbase, Vertica, Druid, Accumulo,

    Hypertable

Graph store



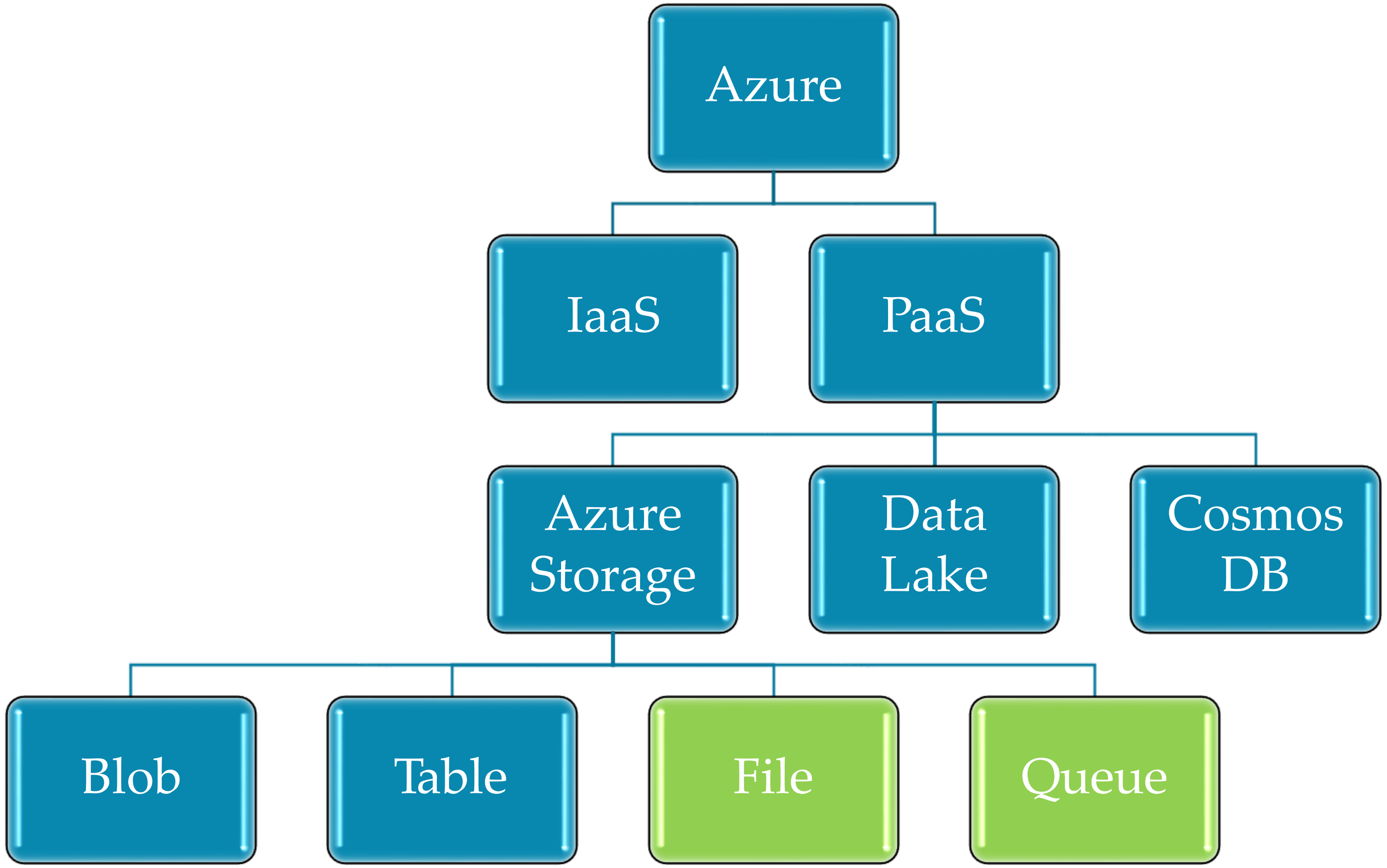
- Focuses on how data relates to other data points.
- A node is a specific entity or piece of information
- Edge simply specifies the relationship between two nodes.
- Use Cases:
  - Social networks, realtime product recommendations, network diagrams, fraud detection, access management, and more.
- Examples
  - Cosmos DB Gremlin API, Neo4j, Blazegraph, and OrientDB.

Multi-model

- Include features/characteristics of more than one data model.

- **Example:**
    - **OrientDB:** OrientDB combines a graph model with a document model.
    - **ArangoDB:** Uses key/value, document, and graph models.
    - **Virtuoso:** Combines relational, graph, and document models.

NoSQL Offerings by Microsoft Azure



Advantages of Blob storage

- Extremely cheap
- Simple to setup
- No configuration
- Doesn’t require powerful computing to manage

Microsoft Azure
Blob Storage

Limitations of Blob storage

- No Indexes
- No Search Tools
- Not optimized for performance
- You are responsible for replication and synchronization
- Requires external compute to process

What is Cosmos DB?

Globally Distributed multi model database service for mission critical applications

Why Cosmos DB?

FULLY MANAGED

- Database as a service (DaaS)
- Serverless architecture
- No operational overhead
- No schema or Index management

GLOBALLY DISTRIBUTED

- Turnkey global distribution

MULTIMODEL & MULTI-LANGUAGE

- Supports Json documents, table graph and columnar data models
- Java, .NET, Python, Node.js, JavaScript, etc.

CONSISTENCY CHOICES

- Azure Cosmos DB's support for consistency levels like strong, eventual, consistent prefix, session, and bounded-staleness.

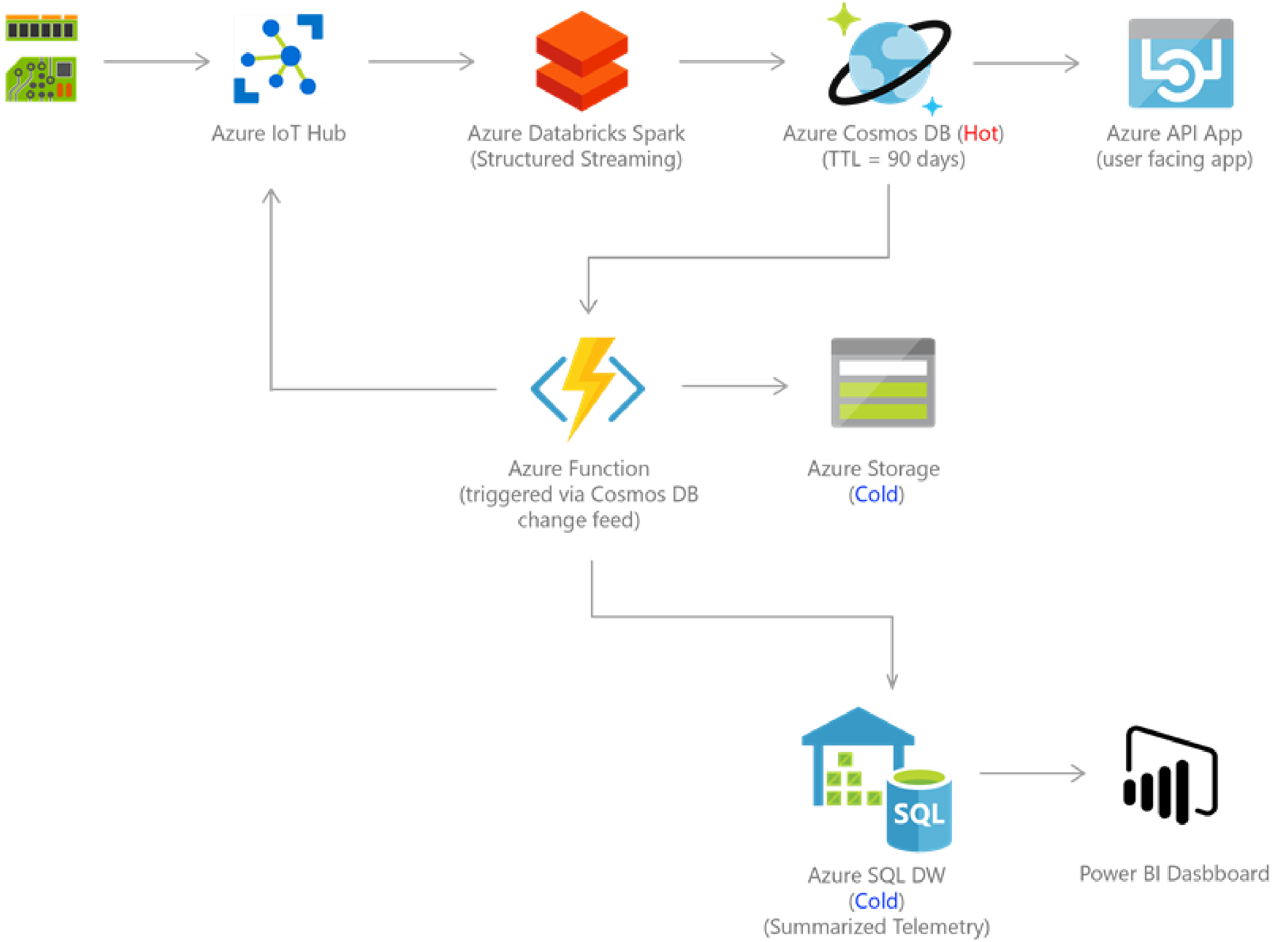
SCALABLE

- Unlimited scale for both storage and throughput.

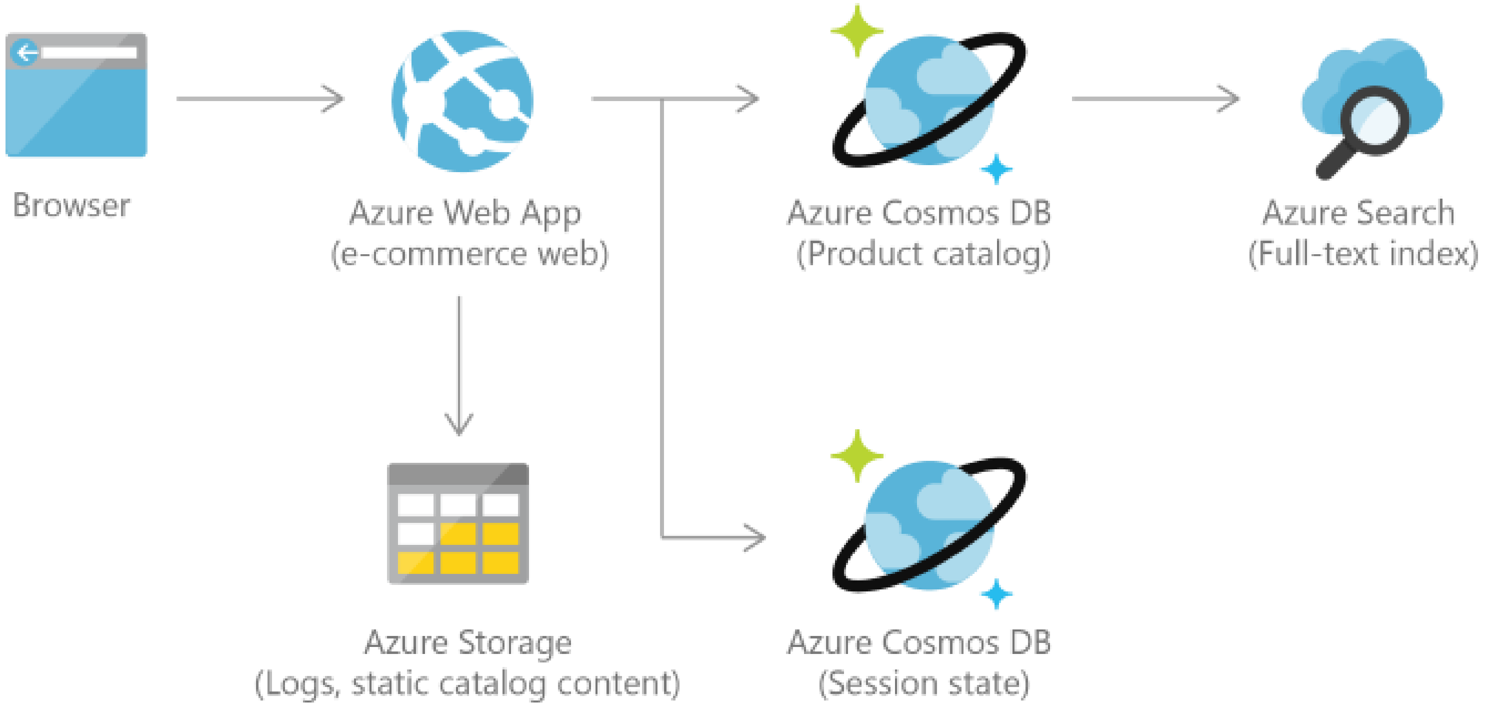
HIGHLY AVAILABLE, RELIABLE & SECURE

- Always on
- 99.999% SLA
- < 10ms latency

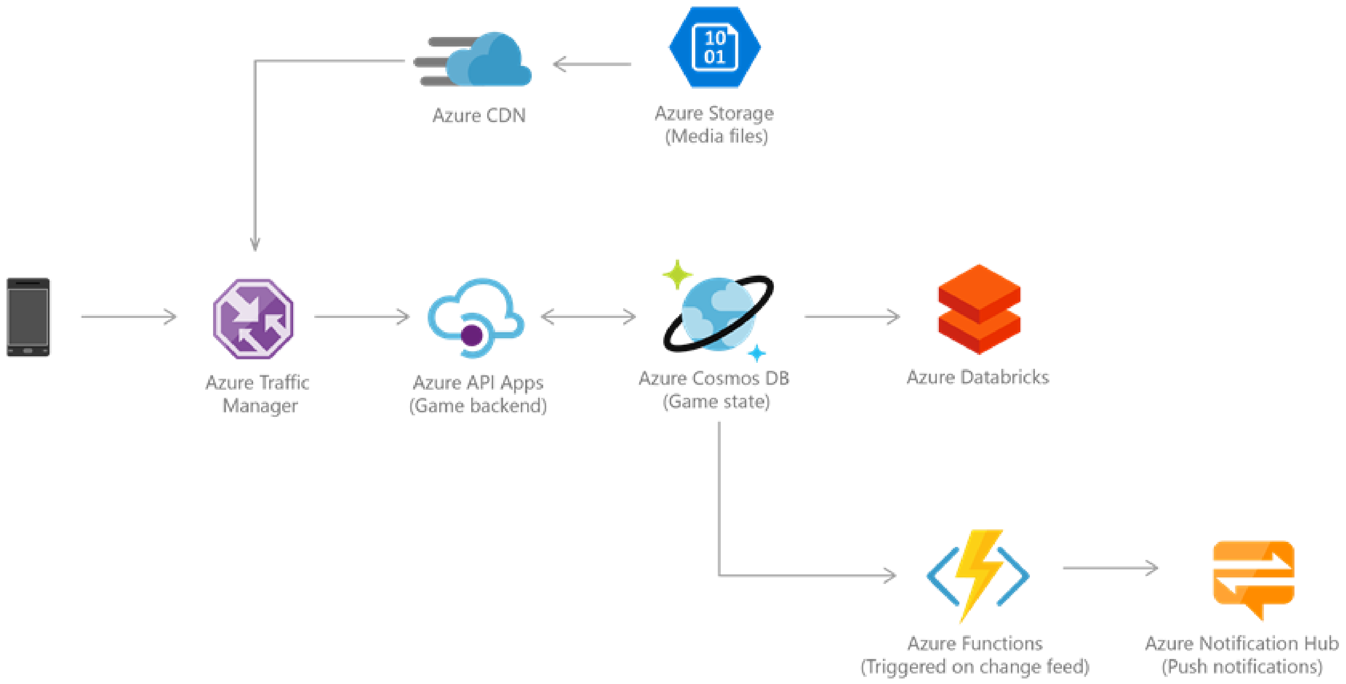
Use case - IOT


Azure IoT Hub
Azure Databricks Spark
(Structured Streaming)
Azure Cosmos DB (Hot)
(TTL = 90 days)
Azure API App
(user facing app)
Azure Function
(triggered via Cosmos DB
change feed)
Azure Storage
(Cold)
SQL
Azure SQL DW
(Cold)
(Summarized Telemetry)
Power BI Dasbboard

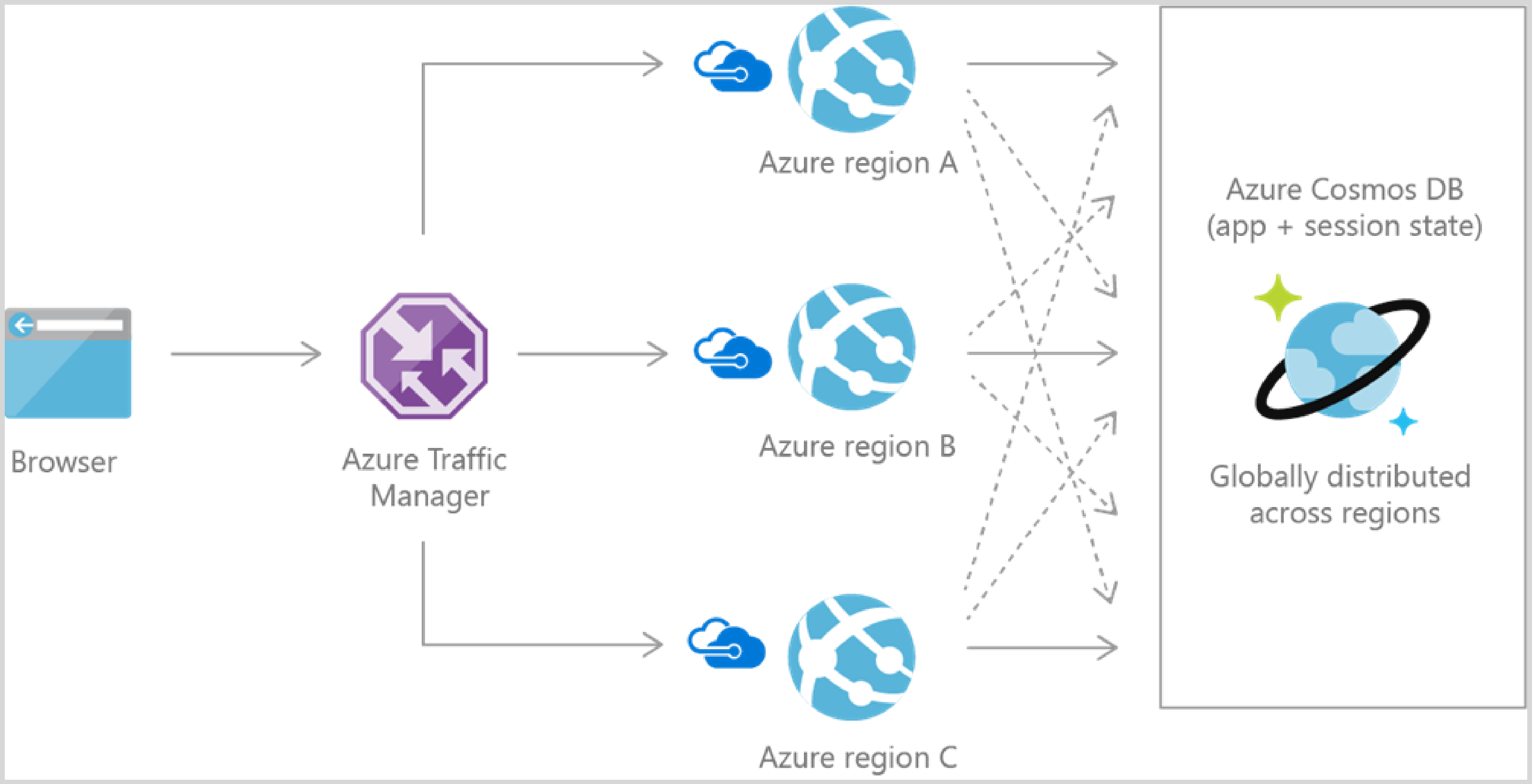
Use case – Retail and Marketing

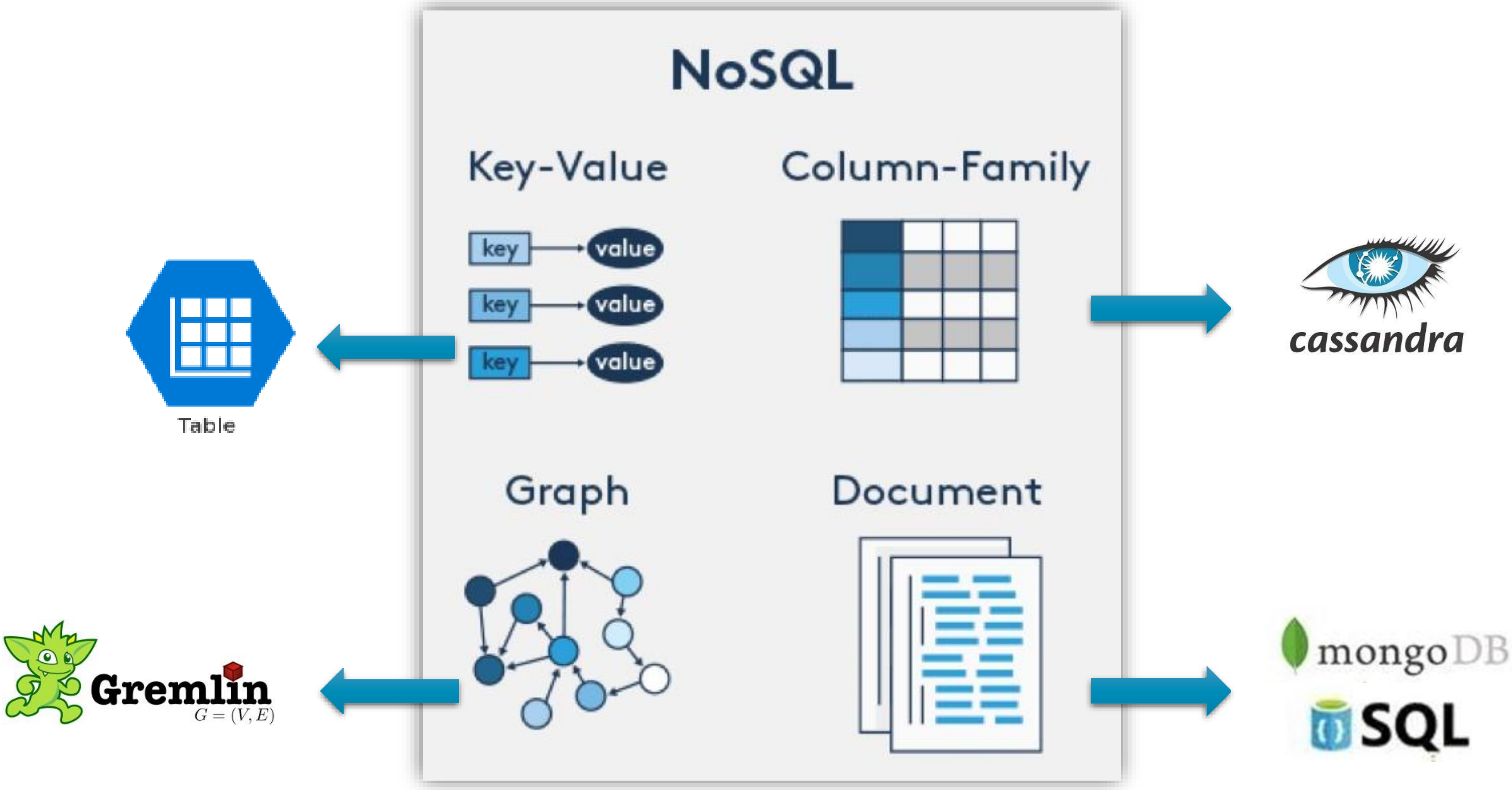


Use case – Gaming



Use case – Web and mobile


Browser
Azure Traffic Manager
Azure region A
Azure region B
Azure region C
Azure Cosmos DB (app + session state)
Globally distributed across regions



NoSQL

Key-Value

Column-Family

Graph

Document

SQL API vs MongoDB API

SQL(CORE) API

JSON Documents

Microsoft original Document DB platform
Supports server side programming model

You can use SQL like language to query JSON documents.

MongoDB API

BSON Documents

Implement Wire protocol

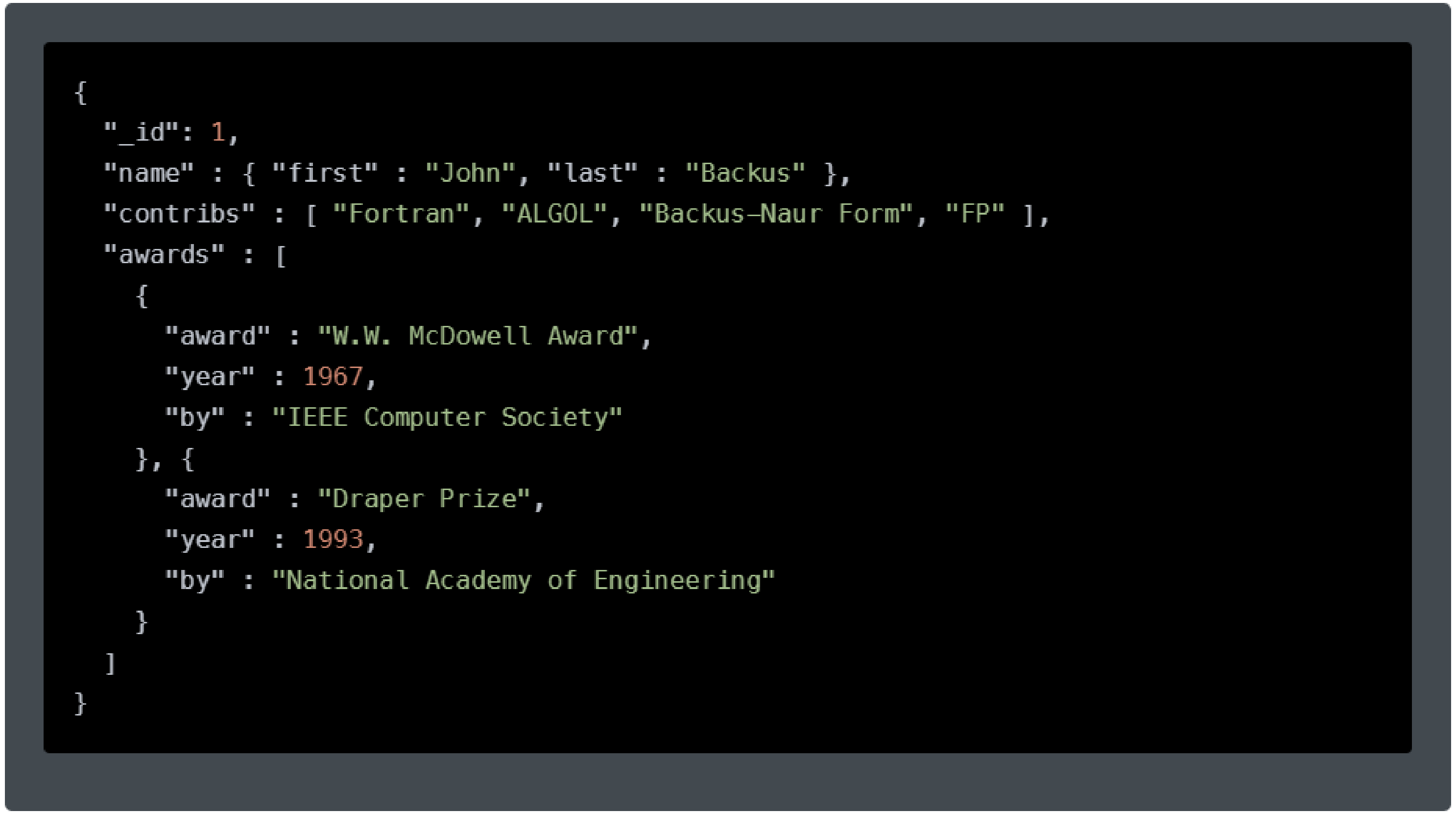
Fully compatible with Mongo DB application code

Migrate existing Cosmos DB without much change of logic

Use SQL(CORE) API for new development

JSON File

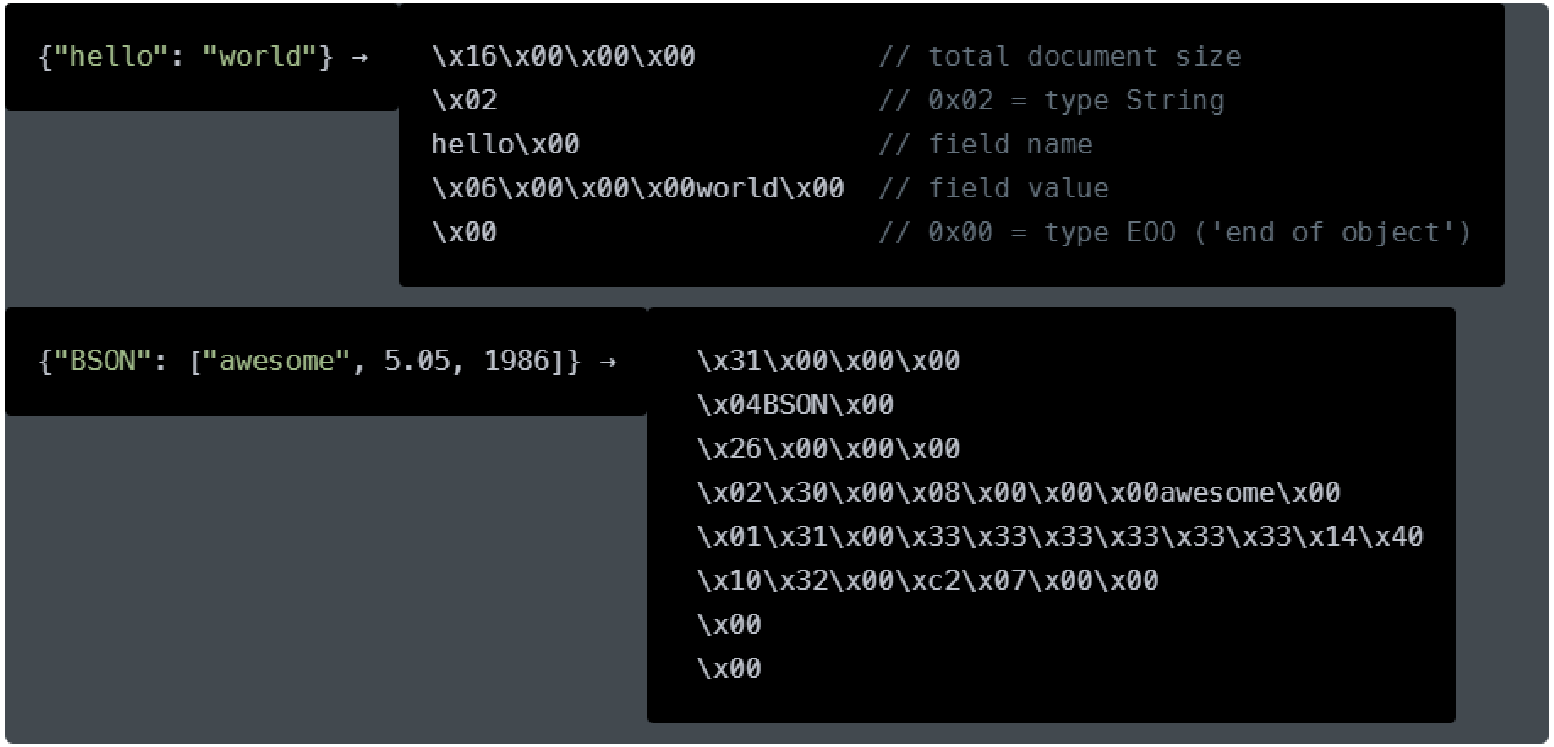
JavaScript objects are simple associative containers, wherein a string key is mapped to a value (which can be a number, string, function, or even another object)



```
{
  "_id": 1,
  "name" : { "first" : "John", "last" : "Backus" },
  "contribs" : [ "Fortran", "ALGOL", "Backus-Naur Form", "FP" ],
  "awards" : [
    {
      "award" : "W.W. McDowell Award",
      "year" : 1967,
      "by" : "IEEE Computer Society"
    }, {
      "award" : "Draper Prize",
      "year" : 1993,
      "by" : "National Academy of Engineering"
    }
  ]
}
```

BSON File

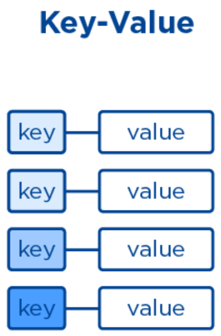
BSON simply stands for “Binary JSON,” and that’s exactly what it was invented to be. BSON’s binary structure encodes type and length information, which allows it to be parsed much more quickly.



```
{"hello": "world"} →   \x16\x00\x00\x00            // total document size
                       \x02                        // 0x02 = type String
                       hello\x00                   // field name
                       \x06\x00\x00\x00world\x00   // field value
                       \x00                        // 0x00 = type EOO ('end of object')
```

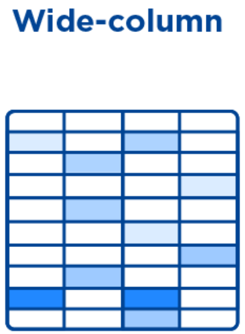
```
{"BSON": ["awesome", 5.05, 1986]} →   \x31\x00\x00\x00
                                      \x04BSON\x00
                                      \x26\x00\x00\x00
                                      \x02\x30\x00\x08\x00\x00\x00awesome\x00
                                      \x01\x31\x00\x33\x33\x33\x33\x33\x33\x14\x40
                                      \x10\x32\x00\xc2\x07\x00\x00
                                      \x00
                                      \x00
```

Cosmos DB Table API



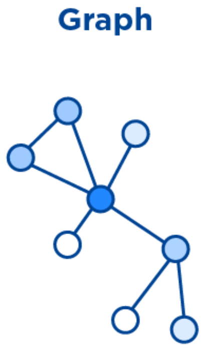
- Key-Value store
- Premium offering for Azure Table Storage
- Existing Table Storage customers will migrate to Cosmos DB Table API
- Row value can be simple like number or string
- Row cannot store object

Cosmos DB Cassandra API



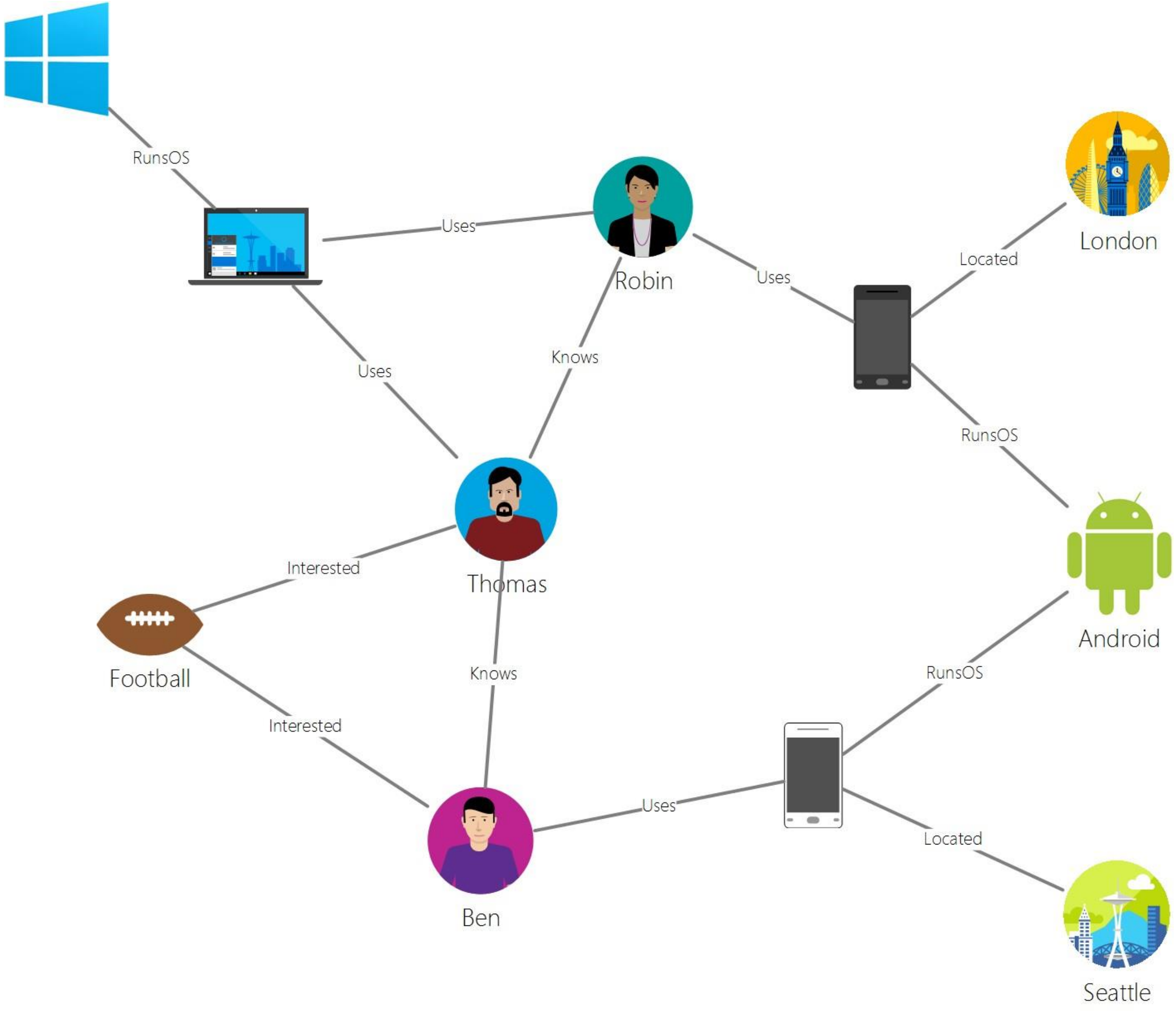
- Wide column No SQL Database
- Name and format of column can vary from row to row.
- Simple migrate your Cassandra application to Cosmos Cassandra API and change connection string.
- Interact
  - Cassandra based tools
  - Data Explorer
  - Programmatically, using SDK (CassandraCSharpdriver)

Cosmos DB Gremlin API

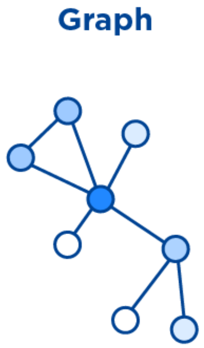


- Graph Data Model
- Real world data connected with each other
- Graph database can persist relationships in the storage layer

Graph Model


RunsOS
Uses
Robin
Uses
Located
London
Uses
Knows
RunsOS
Thomas
Interested
Android
Football
Knows
RunsOS
Interested
Uses
Located
Ben
Seattle

Cosmos DB Gremlin API

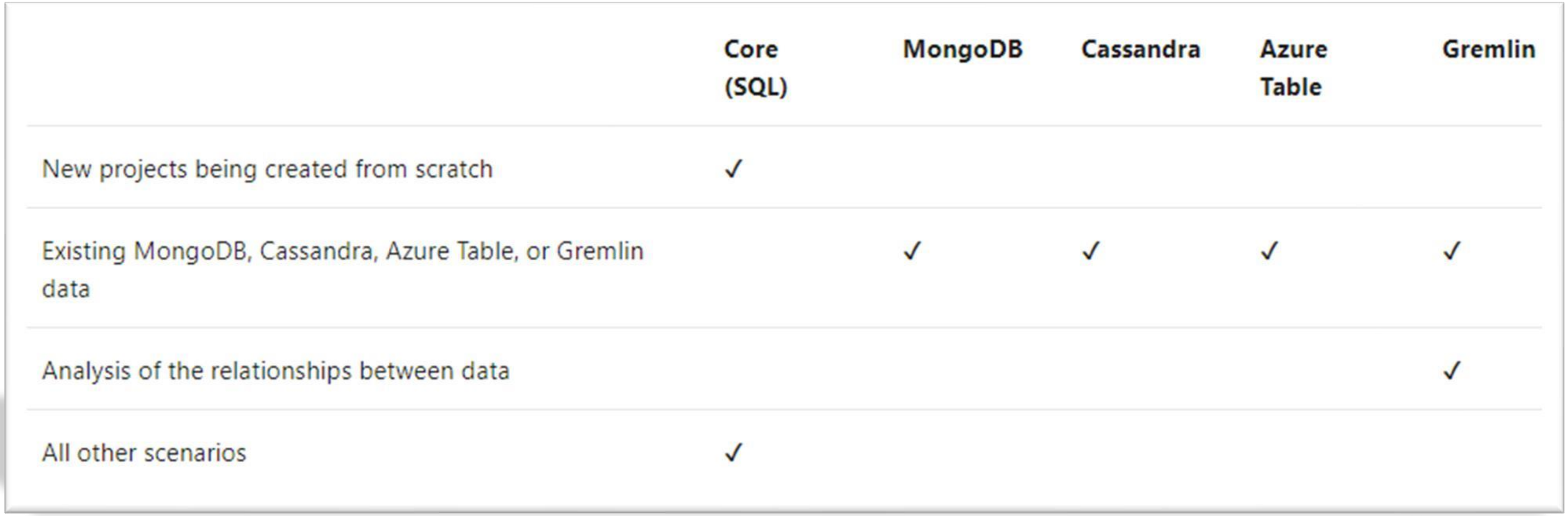


Gremlin

$G = (V, E)$

- Graph Data Model
- Real world data connected with each other
- Graph database can persist relationships in the storage layer
- Use cases
  - Social networks
  - Recommendation engines
  - Geospatial
  - Internet of things
- Migrate existing apps to Cosmos DB Gremlin API
- Graph traverse a language

Analyze the decision criteria



	Core (SQL)	MongoDB	Cassandra	Azure Table	Gremlin
New projects being created from scratch	✓				
Existing MongoDB, Cassandra, Azure Table, or Gremlin data		✓	✓	✓	✓
Analysis of the relationships between data					✓
All other scenarios	✓				

Azure Table storage vs Cosmos DB Table API

- Cosmos DB Table API is a prime version of Azure Table Storage

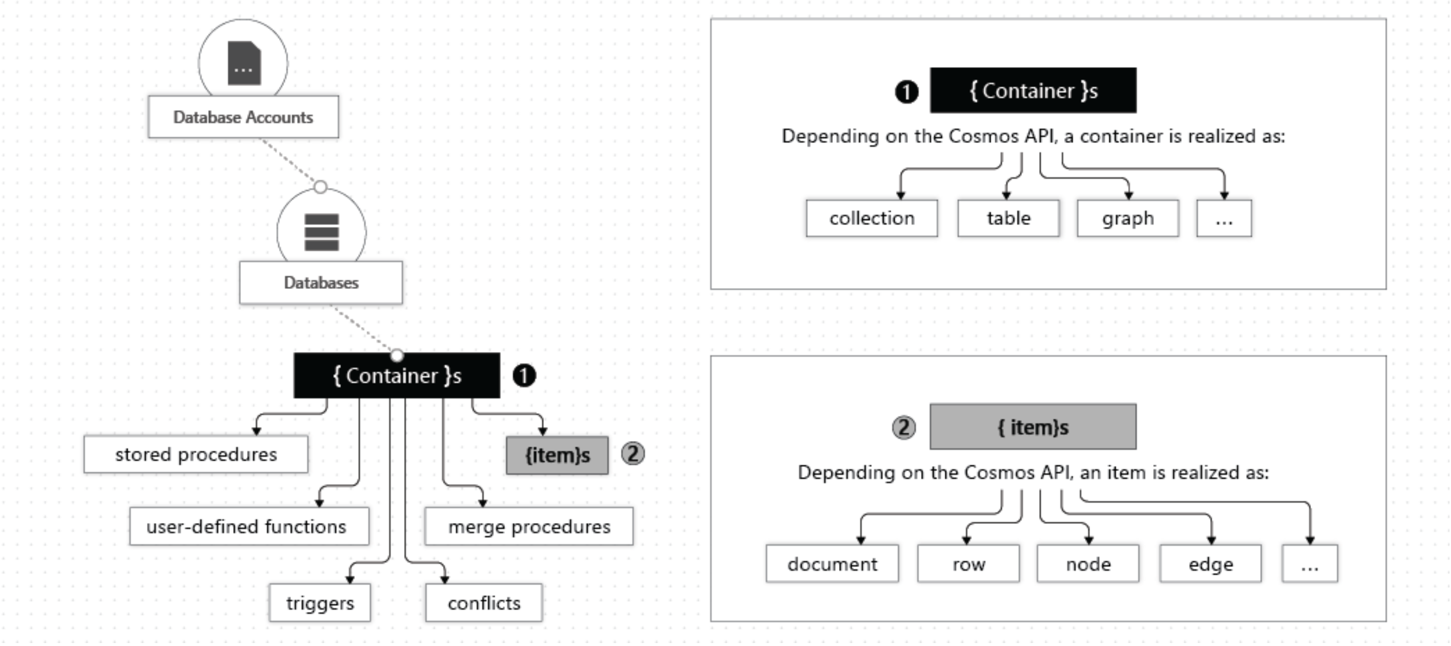
Azure Table Storage

- ➢ Geo replication is restricted
  - Only 1 additional pair region
- ➢ Support for primary key lookups only
- ➢ Price optimized for cold storage
- ➢ Lower performance
  - Throughput is capped
  - Latency is higher
- ➢ No consistency options

Cosmos DB Table API

- ➢ Geo replication across your choice of any number of regions
- ➢ Secondary index support for lookups across multiple dimensions
- ➢ Better performance
  - ➢ Unlimited and predictable throughput
  - ➢ latency is lower
- ➢ 5 consistency options

Database Containers and Items



1 { Container }s

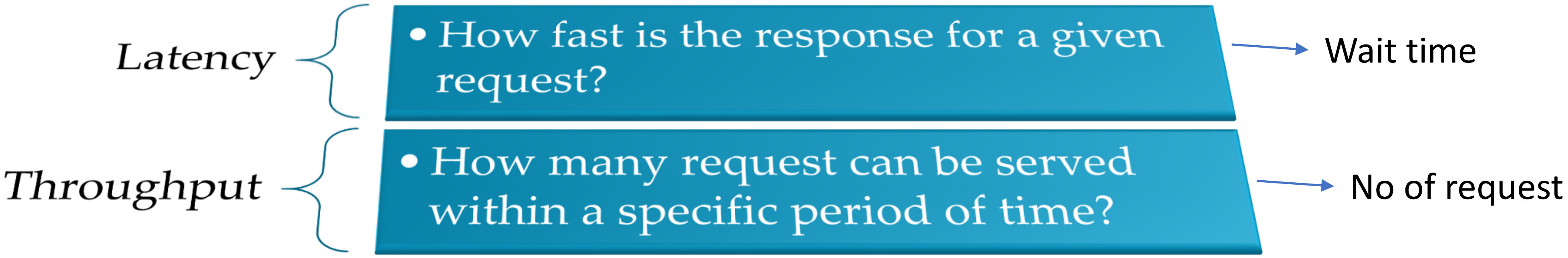
Depending on the Cosmos API, a container is realized as:

2 { item}s

Depending on the Cosmos API, an item is realized as:

Azure Cosmos entity	SQL API	Cassandra API	MongoDB API	Gremlin API	Table API
Azure Cosmos database	Database	Keyspace	Database	Database	NA
Azure Cosmos container	Container	Table	Collection	Graph	Table
Azure Cosmos item	Document	Row	Document	Node or edge	Item

Measuring Performance(throughput)



Latency

- How fast is the response for a given request?

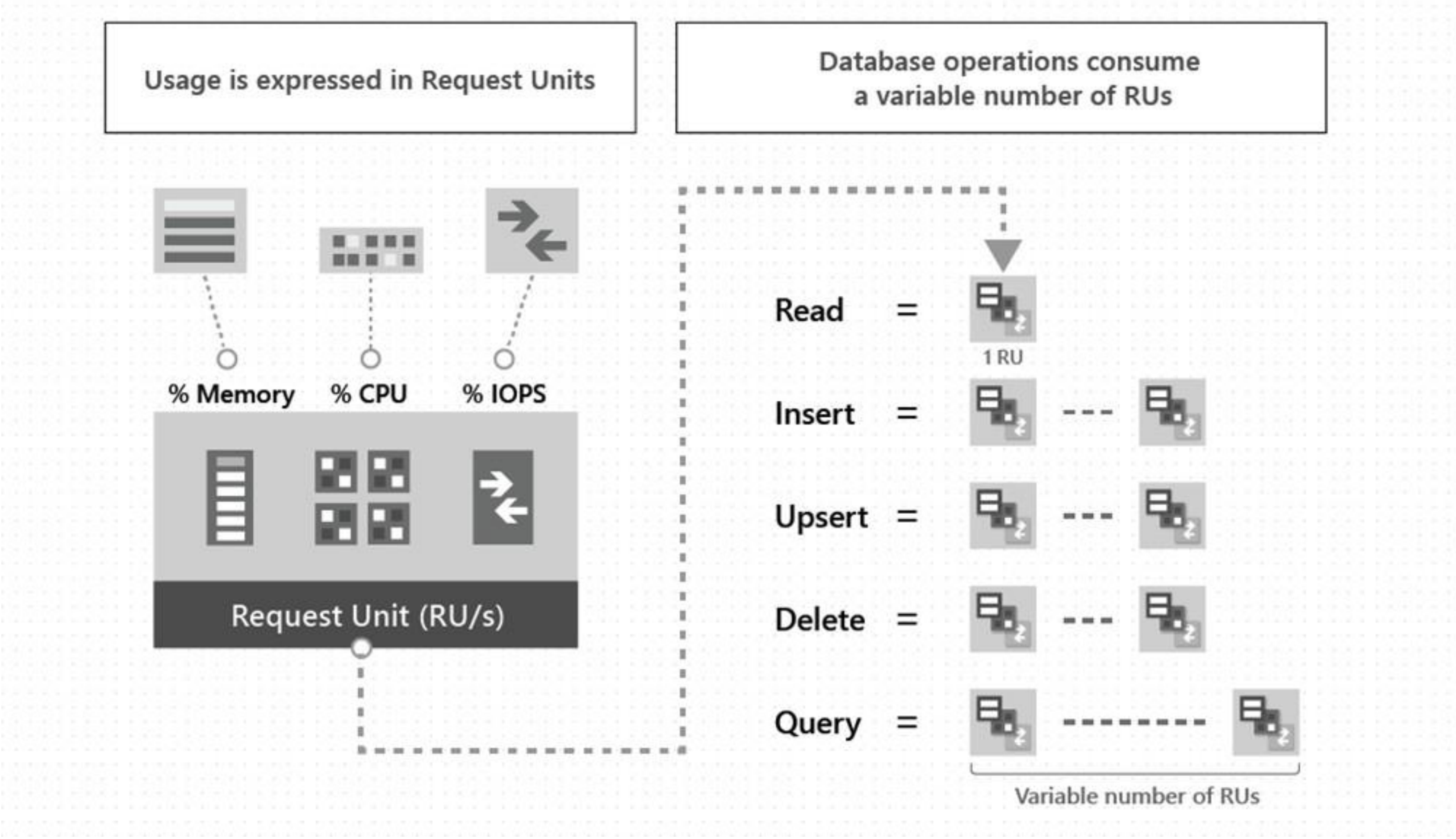
Wait time

Throughput

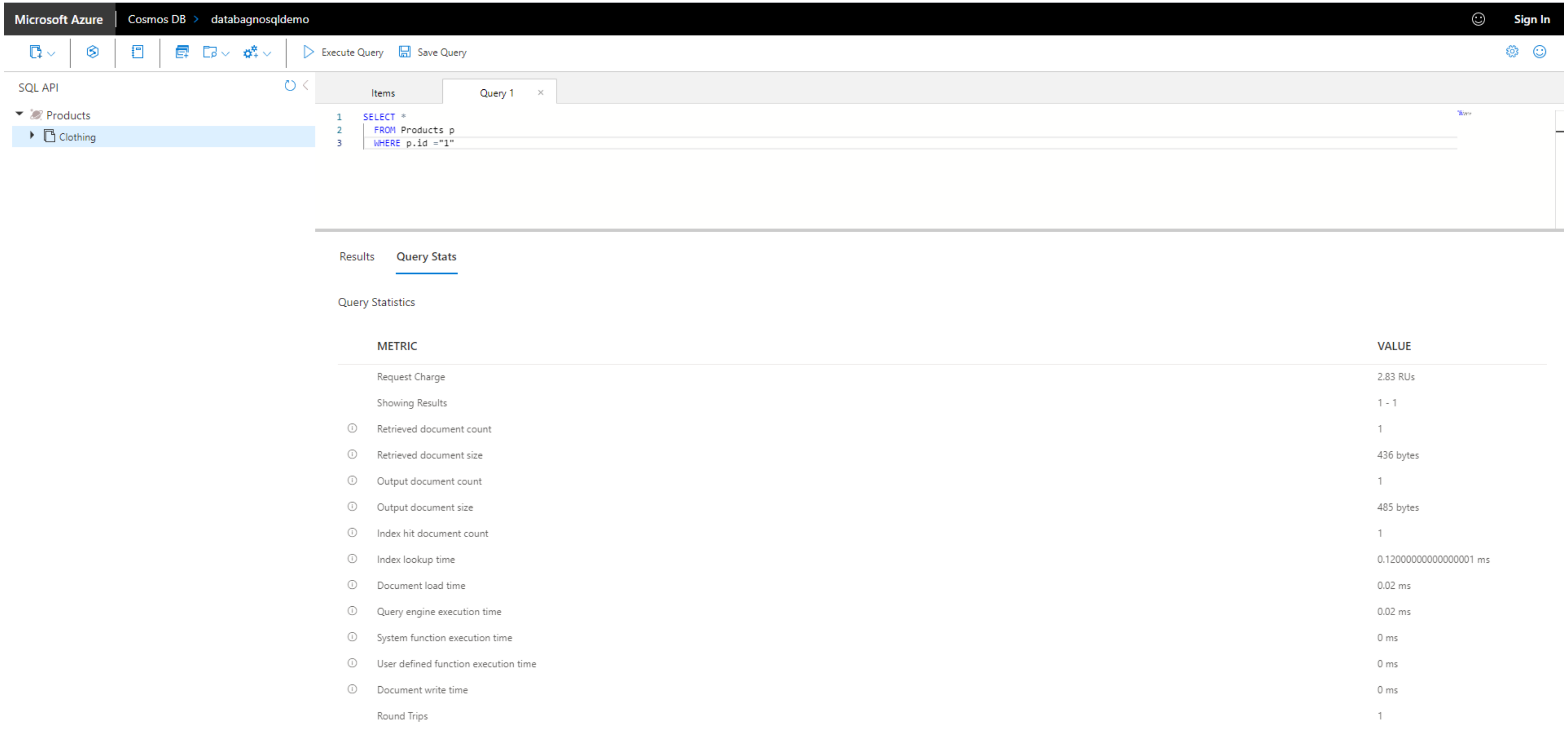
- How many request can be served within a specific period of time?

No of request

Introducing Request Units



Introducing Request Units



```
SELECT *
  FROM Products p
  WHERE p.id ="1"
```

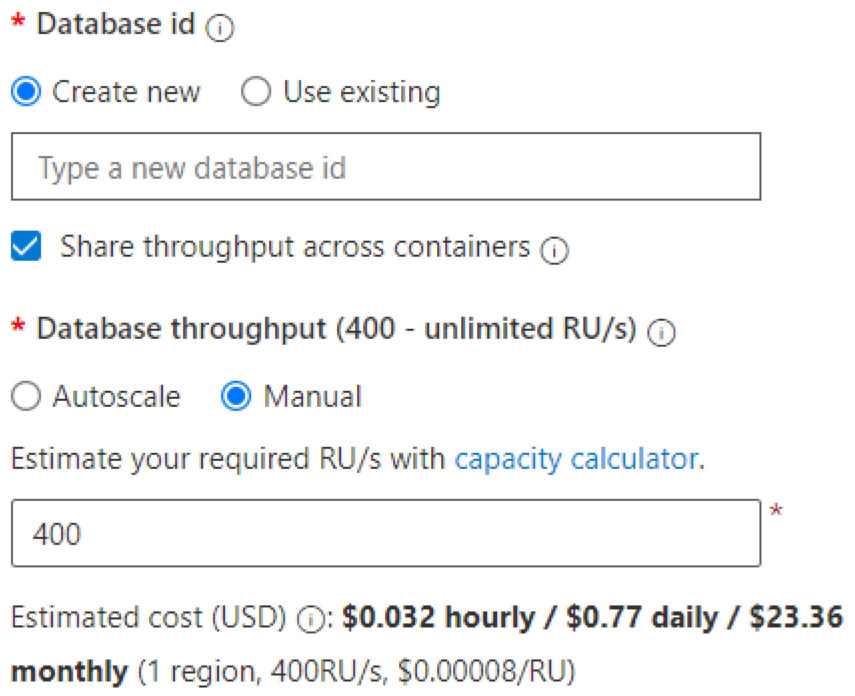
Query Statistics

METRIC	VALUE
Request Charge	2.83 RUs
Showing Results	1 - 1
Retrieved document count	1
Retrieved document size	436 bytes
Output document count	1
Output document size	485 bytes
Index hit document count	1
Index lookup time	0.12000000000000001 ms
Document load time	0.02 ms
Query engine execution time	0.02 ms
System function execution time	0 ms
User defined function execution time	0 ms
Document write time	0 ms
Round Trips	1

Reserving requests units

- Provision Request units per second (RU/s)
  - How many request units (not requests) per second are

    available to your application
- Exceeding reserved throughput limits
  - Requests are “throttled” (HTTP 429)



* Database id ⓘ

● Create new ○ Use existing

Type a new database id

☑ Share throughput across containers ⓘ

* Database throughput (400 - unlimited RU/s) ⓘ

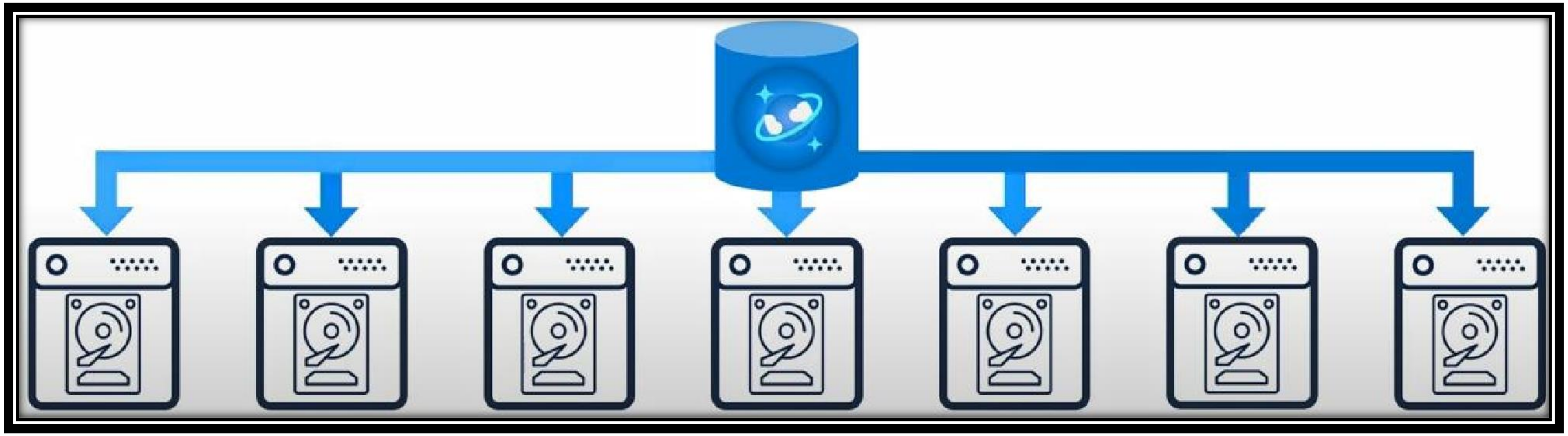
○ Autoscale ● Manual

Estimate your required RU/s with capacity calculator.

400 *

Estimated cost (USD) ⓘ: **$0.032 hourly / $0.77 daily / $23.36 monthly** (1 region, 400RU/s, $0.00008/RU)

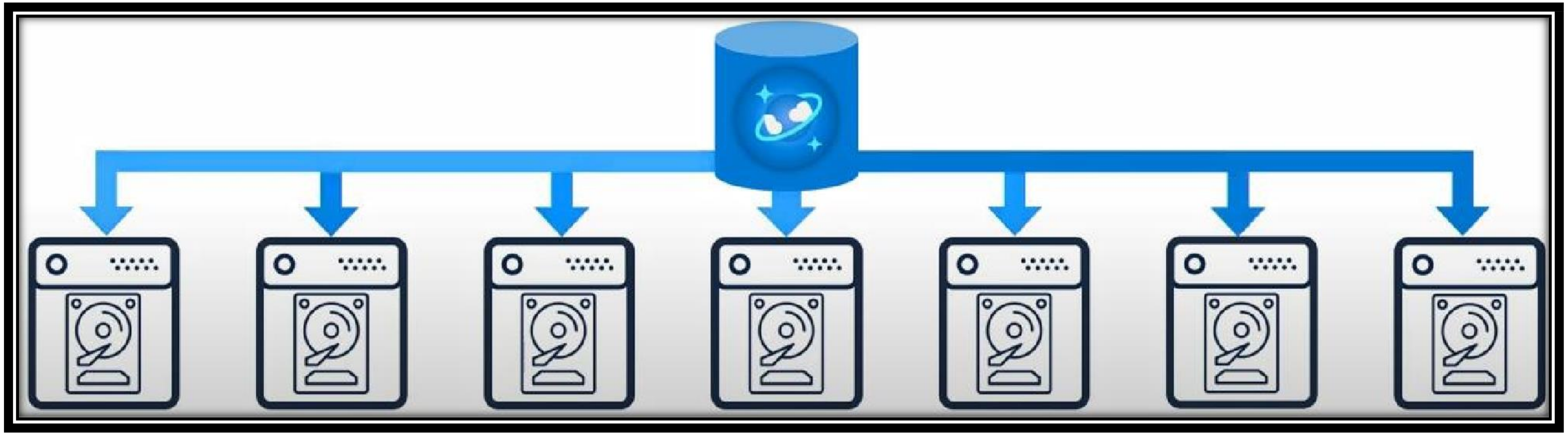
Horizontally Scalable



Unlimited Storage

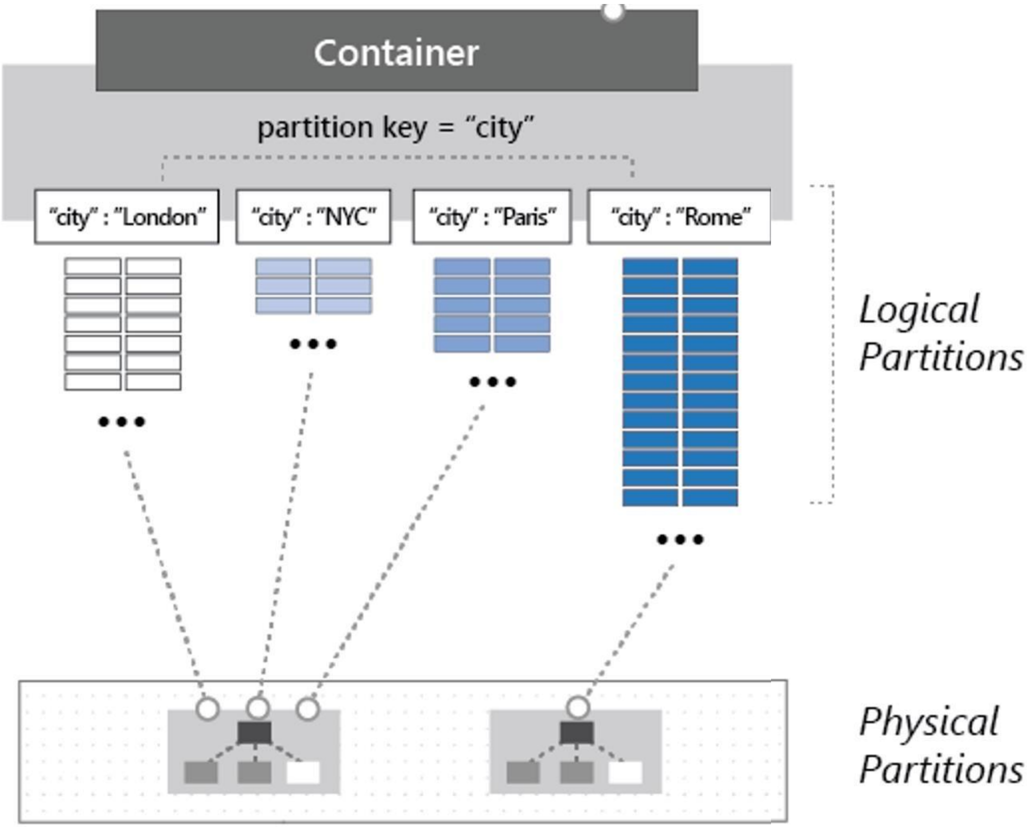
Unlimited Throughput

Partitioning

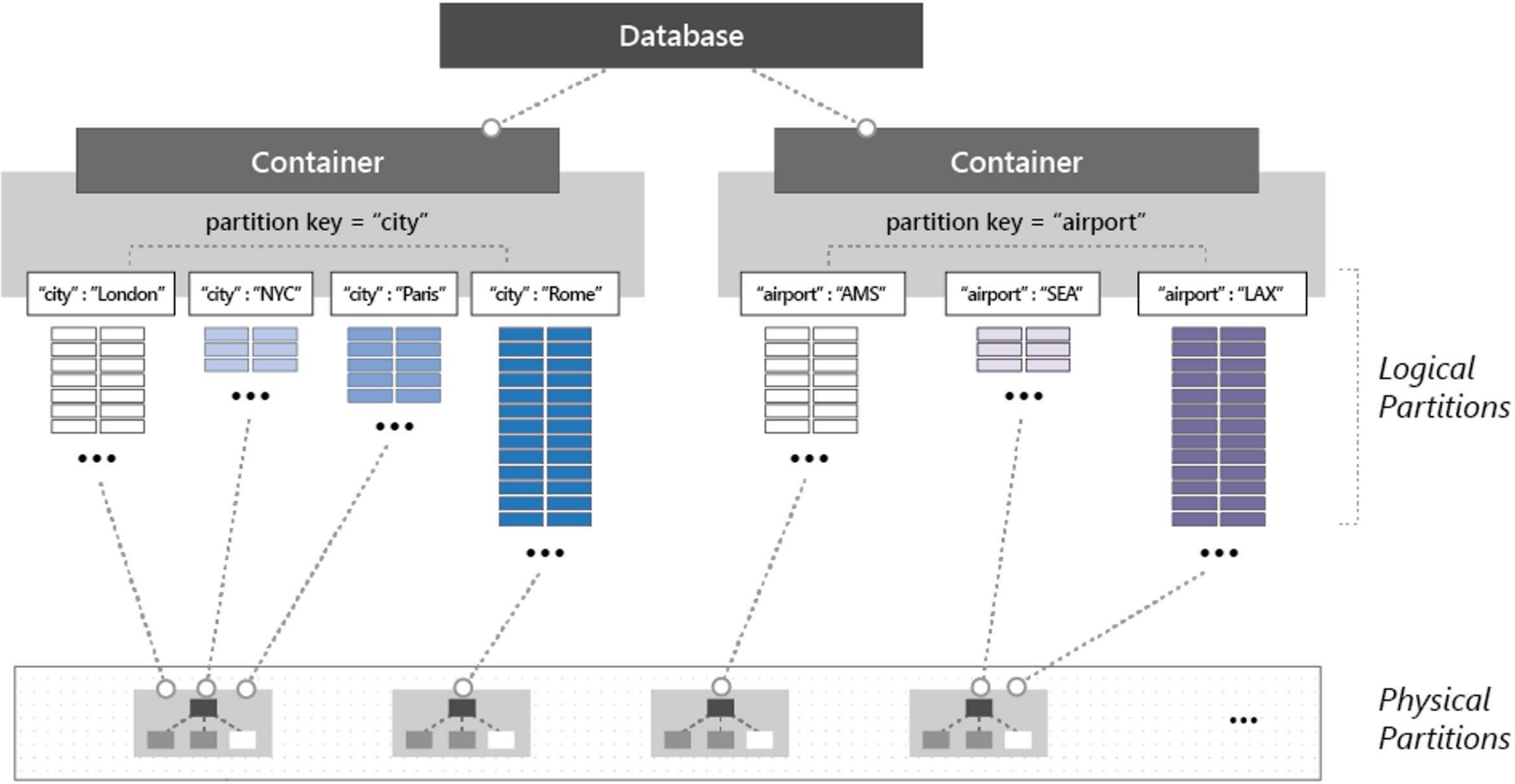


Partitioning

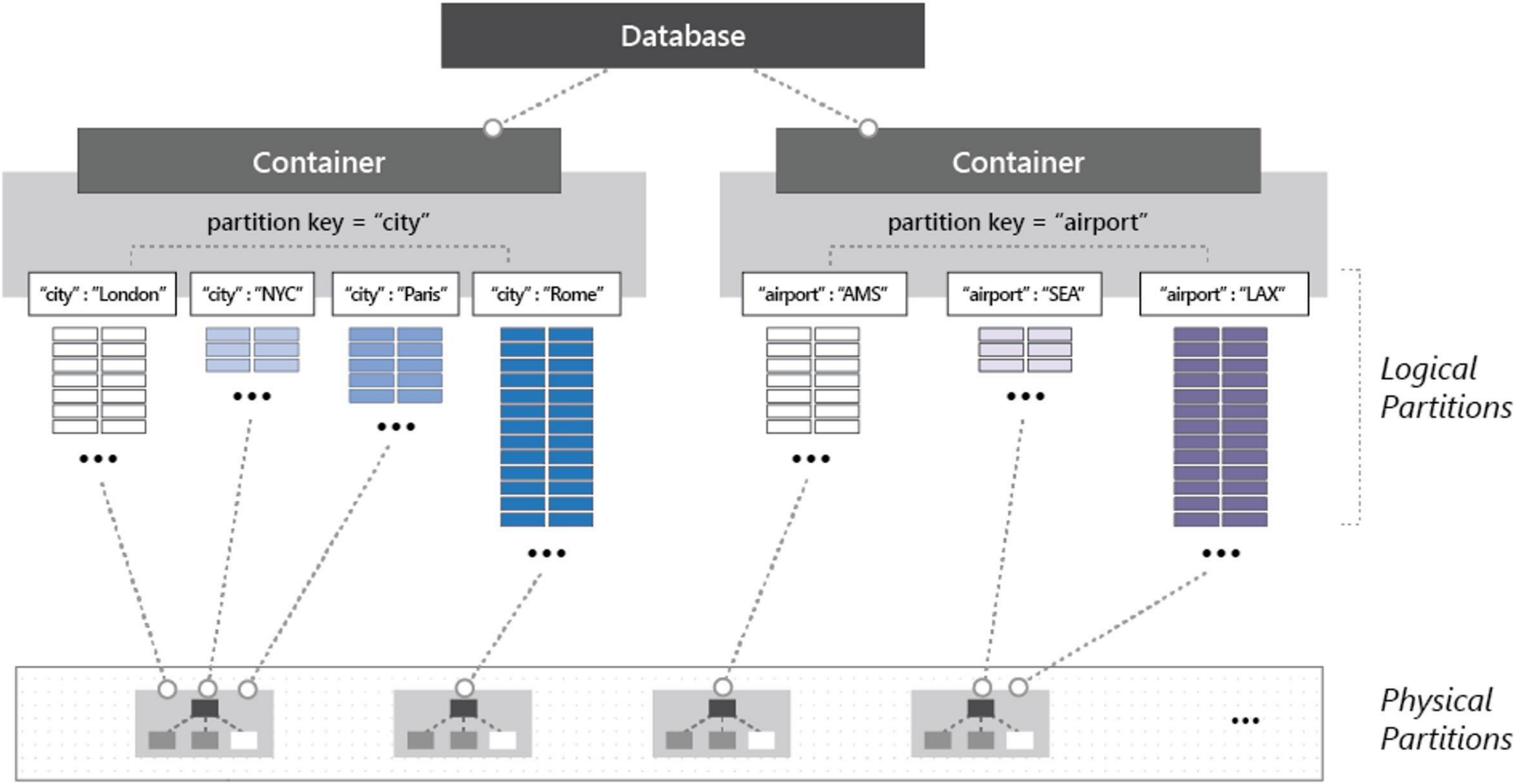
- **Partitioning**: the items in a container are divided into distinct subsets called logical partitions.
- **Partition key** is the value by which Azure organizes your data into logical divisions.
- **Logical partitions** are formed based on the value of a partition key that is associated with each item in a container.
- **Physical partitions**: Internally, one or more logical partitions are mapped to a single physical partition.



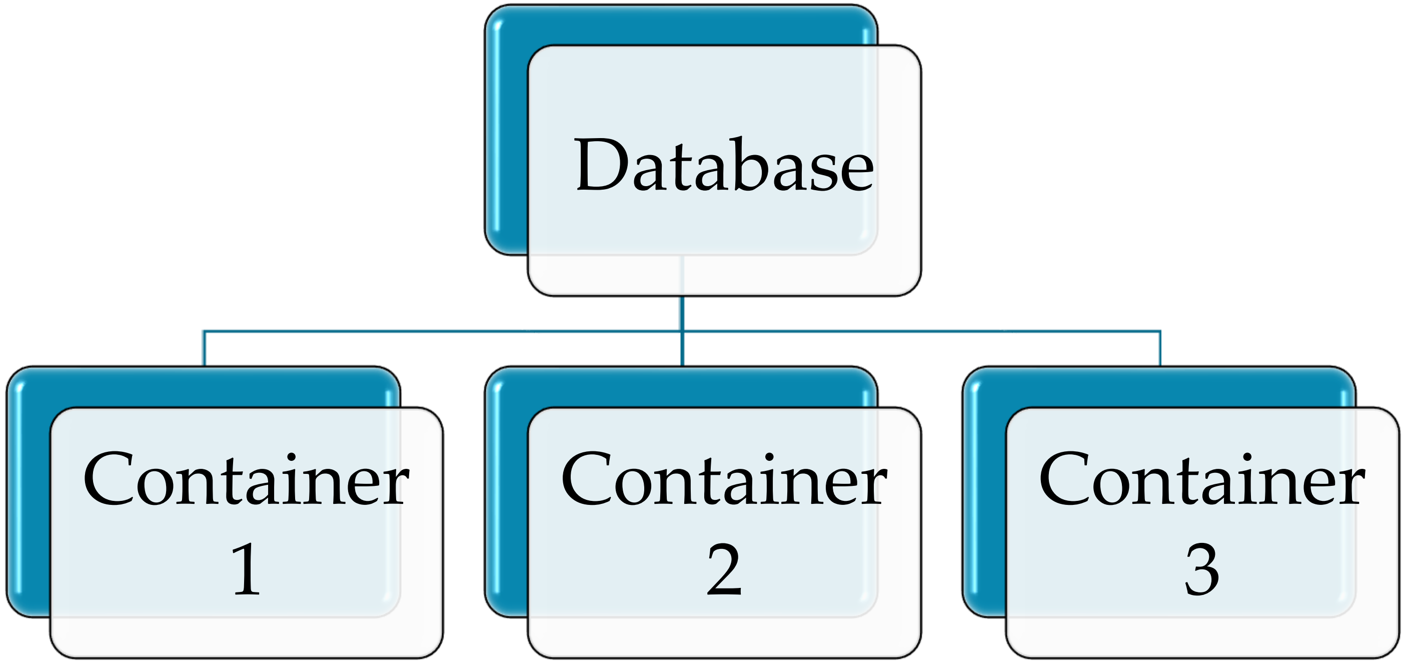
Partitioning



Dedicated vs Shared throughput



Dedicated vs Shared throughput



Database

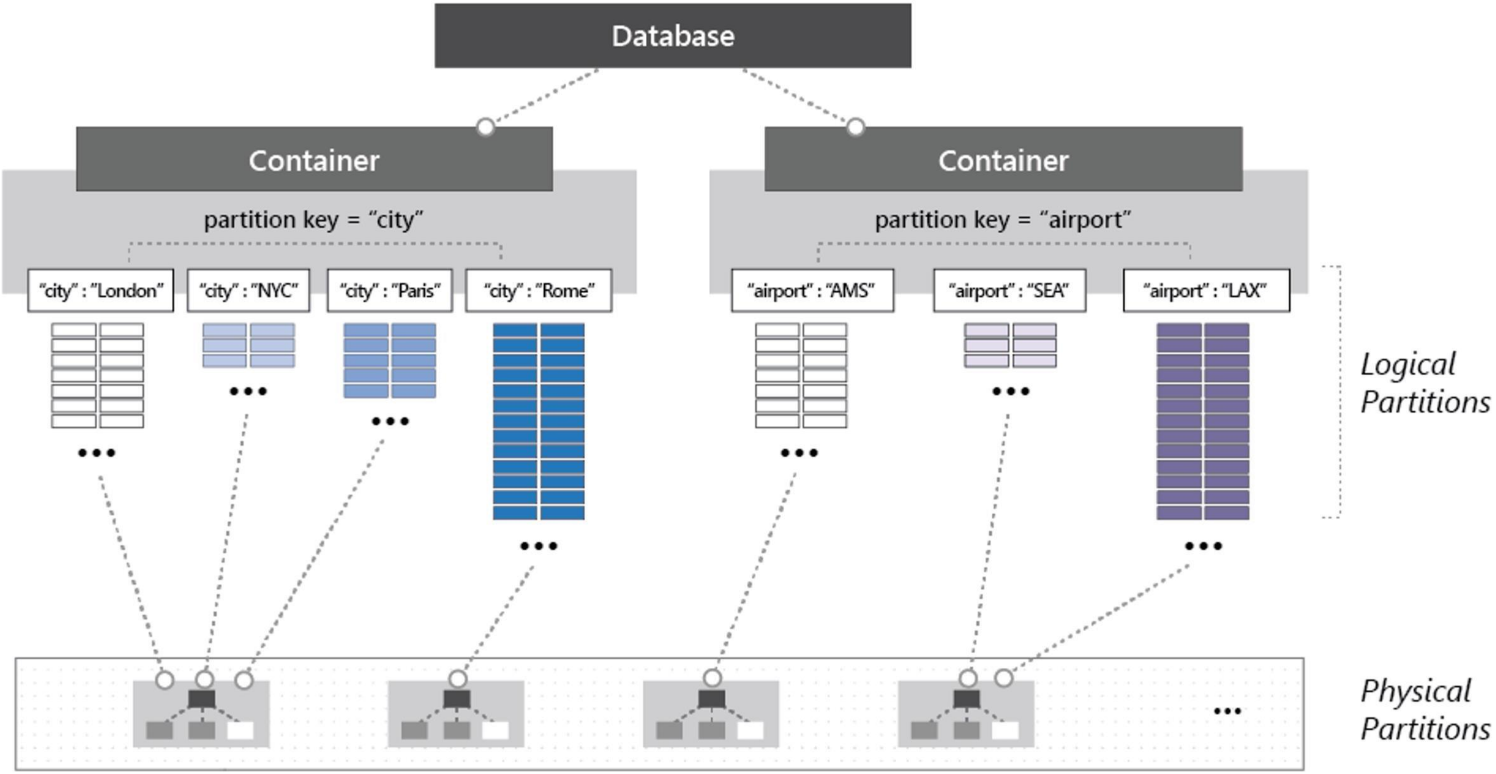
Container 1

Container 2

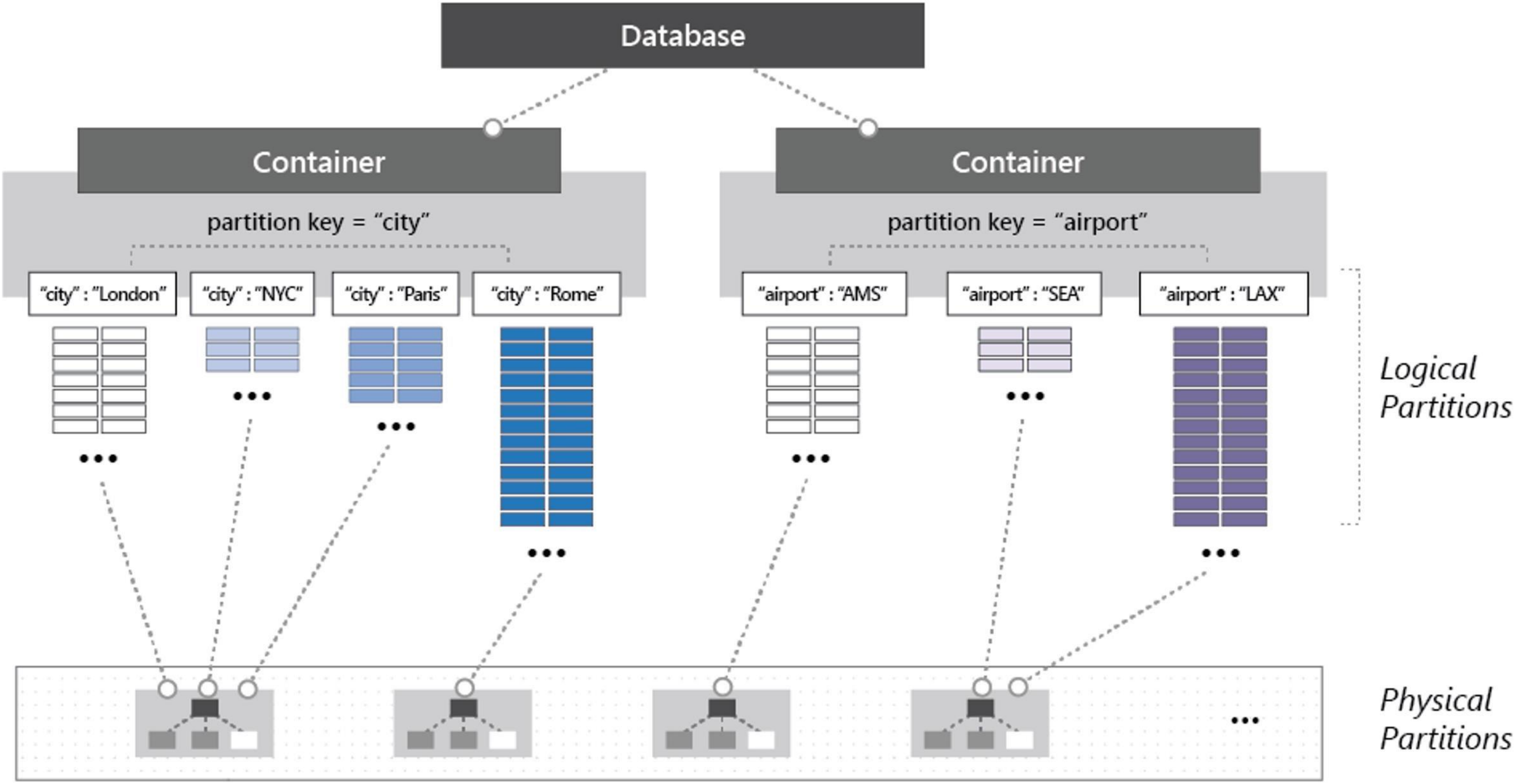
Container 3

Dedicated vs Shared throughput

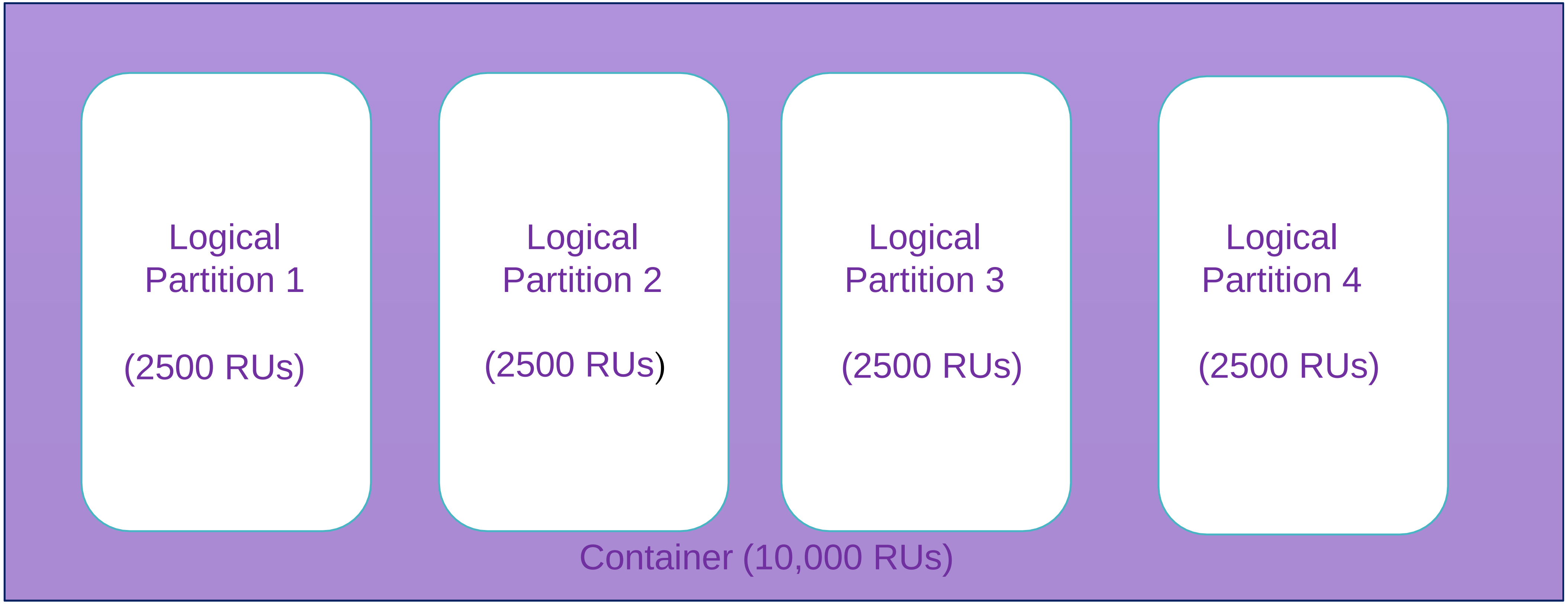
- You can set throughput at:
  - Database level – Shared throughput
  - Container level – Dedicated throughput
  - It is recommend to set throughput at container level.
- Rate-Limited
- Choose at the time of creation



Avoiding hot partition



Avoiding Hot Partitions



Logical Partition 1

(2500 RUs)

Logical Partition 2

(2500 RUs)

Logical Partition 3

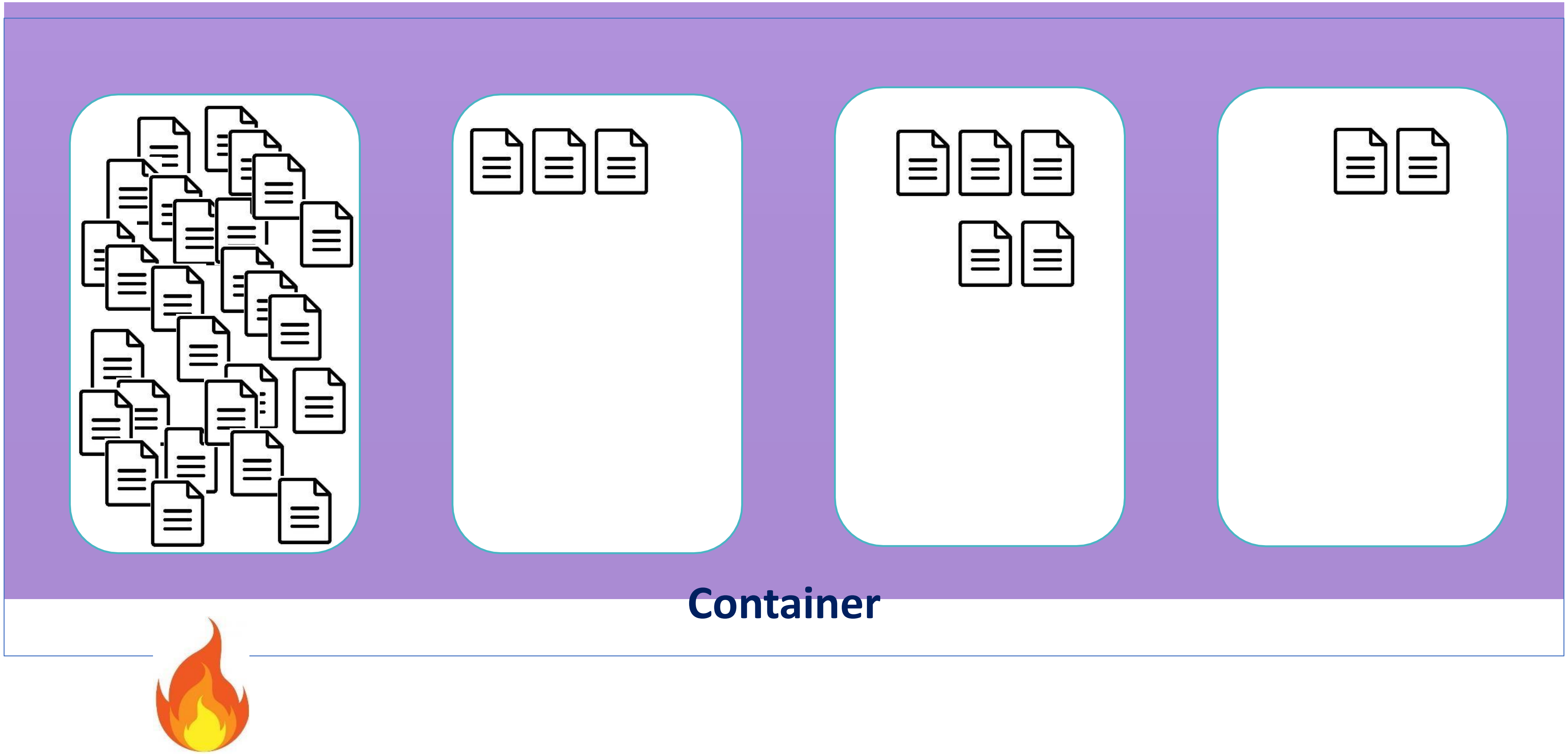
(2500 RUs)

Logical Partition 4

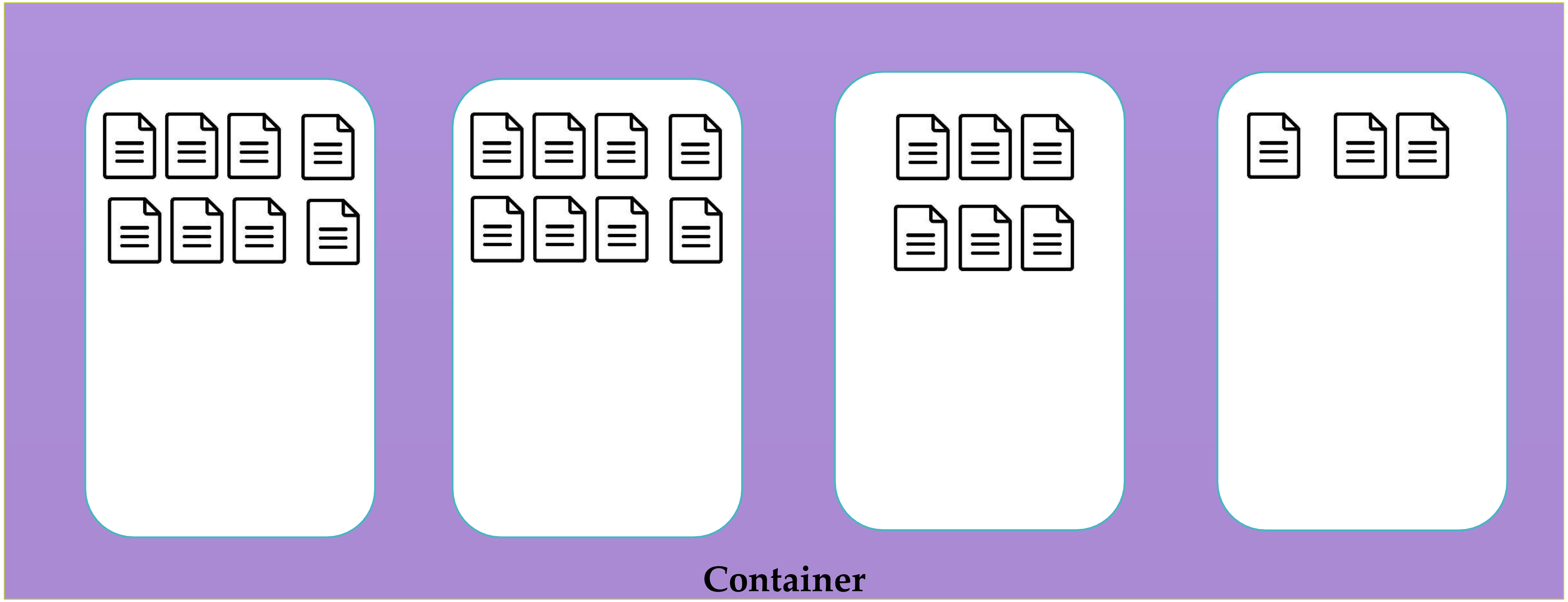
(2500 RUs)

Container (10,000 RUs)

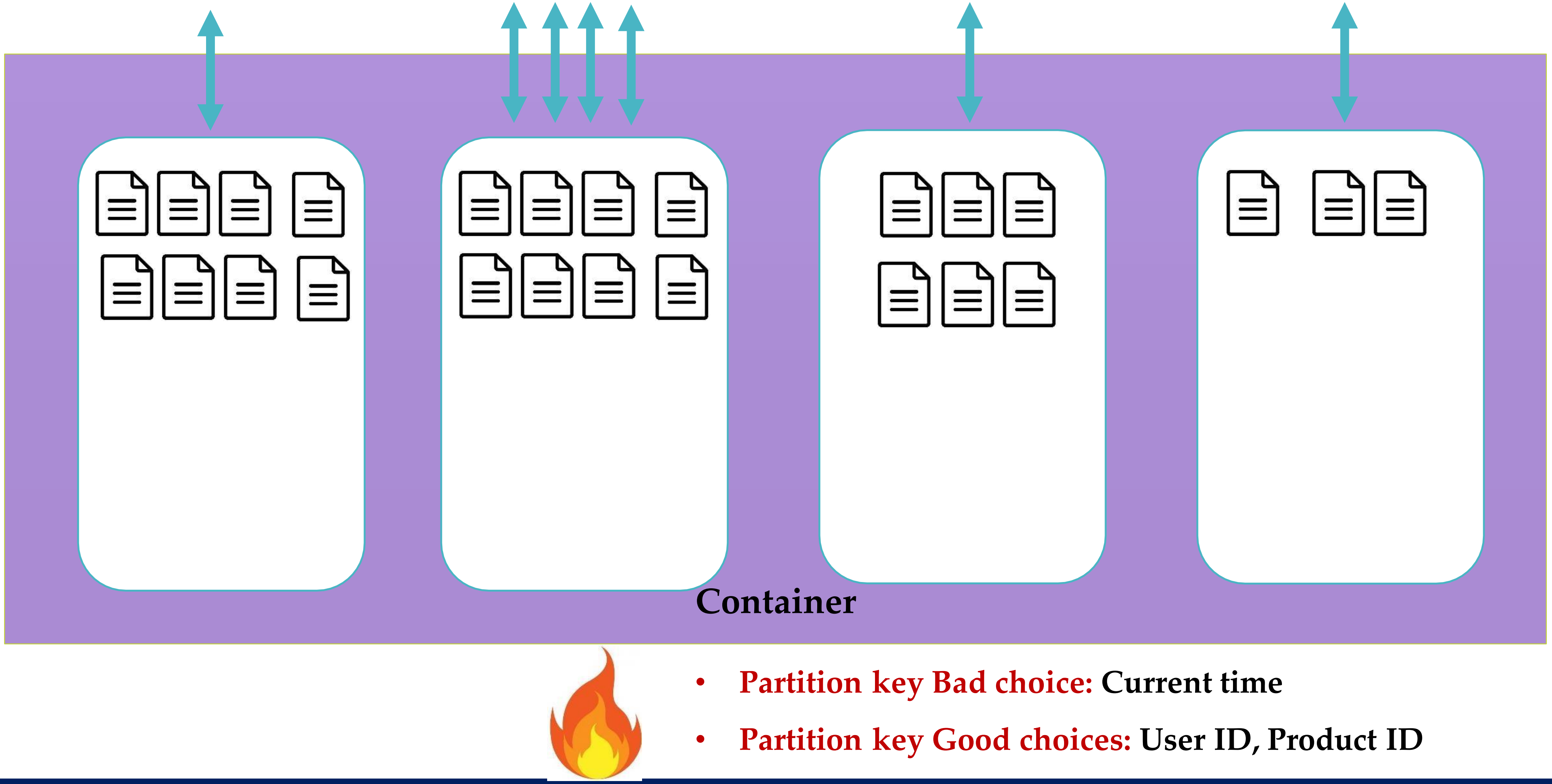
Avoid Hot partitions on storage



Avoiding Hot Partitions at store

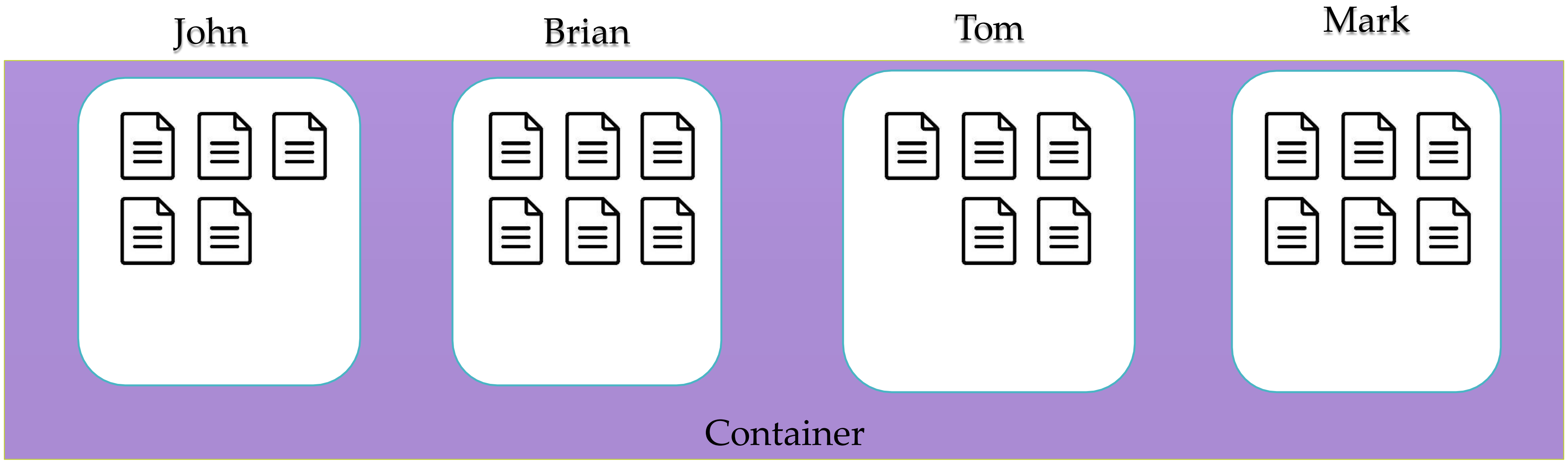


Avoid Hot partitions on throughput



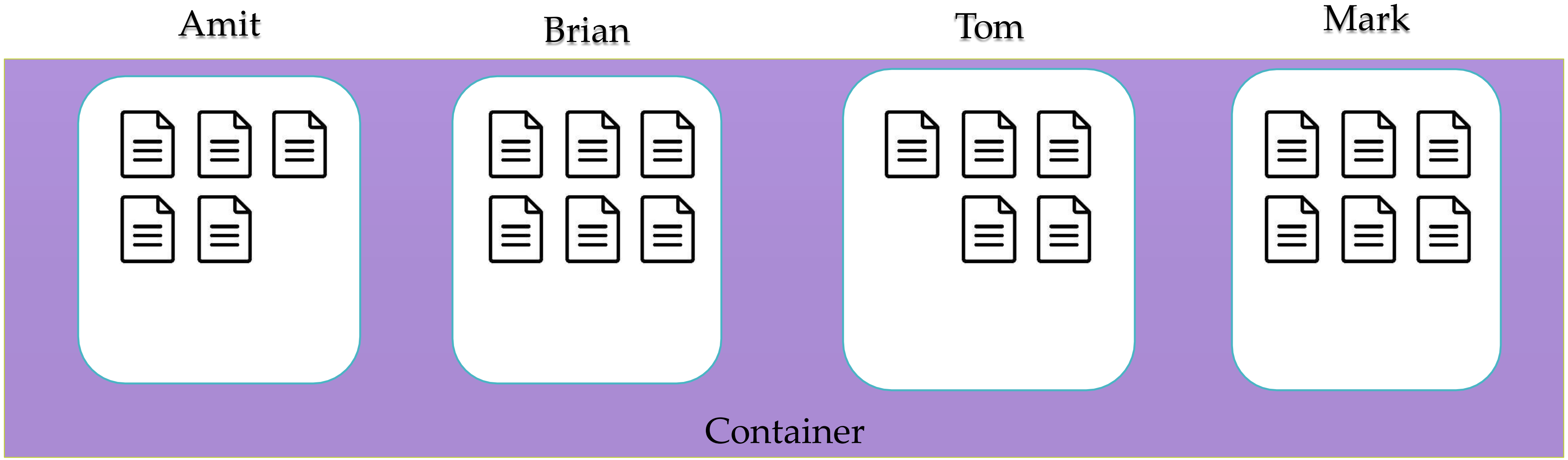
- Partition key Bad choice: Current time
- Partition key Good choices: User ID, Product ID

Single partition Query



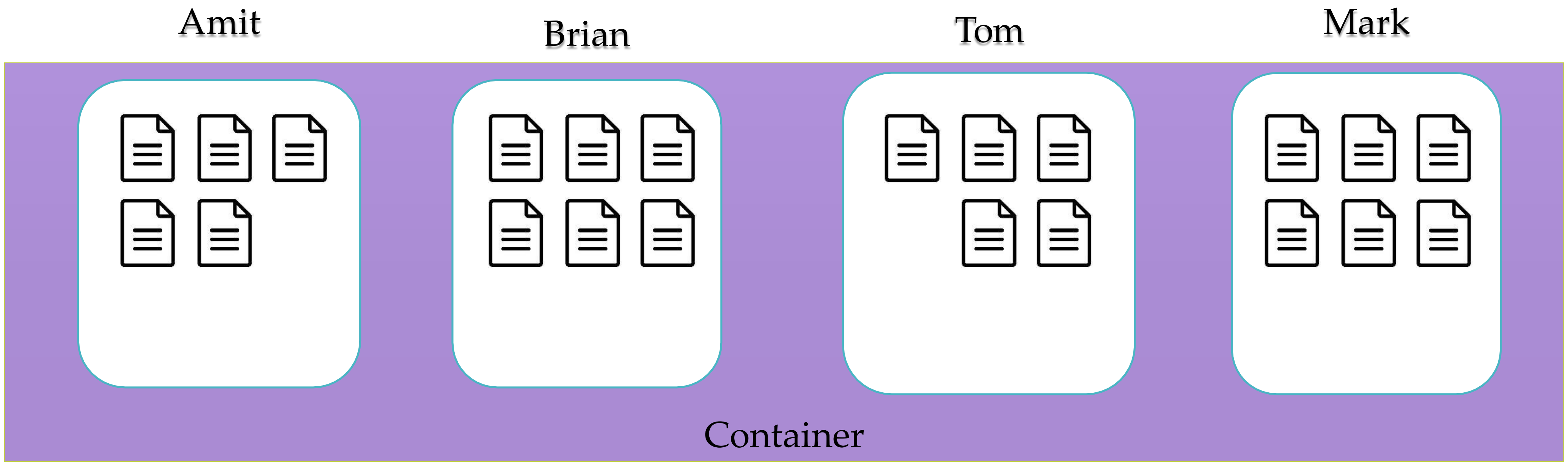
SELECT * FROM c WHERE c.username = ‘Brian’

Cross partition Queries (fan out queries)



SELECT * FROM c WHERE c.favoritecolor= ‘Blue’

Composite Key



Composite Key: CustomerName-mmddyyyy

Choosing a Partition key

- Evenly distribute storage
  - Make sure you pick your partition key that doesn't result in hot spots within your applications
  - Have a high cardinality (high uniqueness)
  - Don’t be afraid of choosing a partition key that has a large number of values
  - Example User Id & Product Id
- Evenly distribute requests.
  - RUs evenly distribute across all partitions.
  - Review where clause of top queries
- Consider document and partition limit while designing partition key.
  - Max document size – 2 MB
  - Max logical partition size – 20 GB

Choosing a Partition key

Question: Your organization is planning to use Azure Cosmos DB to store Motor Bike telemetry data generated from millions of Motor Bikes every second. Which of the following options for your Partition Key will optimize storage distribution?

Answer choices:

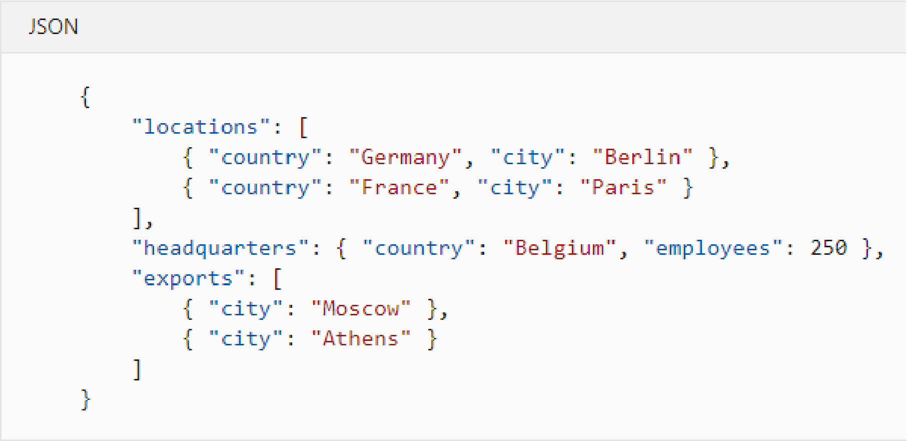
1. Motor Bike model
2. Motor Bike Identification Number (BIN) which looks like CYINFGYA032037

Automatic Indexing

- Index all data without requiring Index management
- Every property of every record automatically index
- Index update synchronously as you create, update

  or delete items
- Not specific for SQL, but available for all APIs

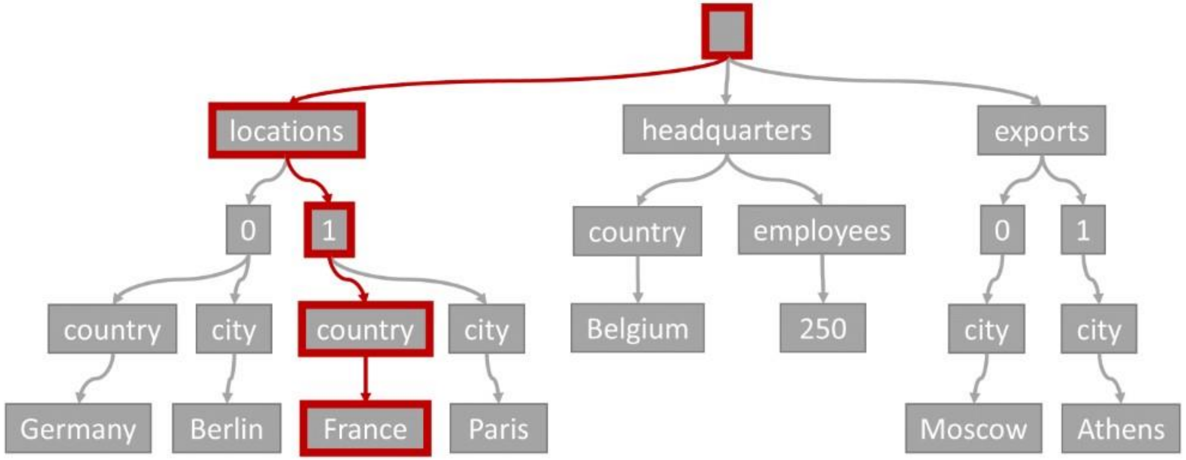
Automatic Indexing

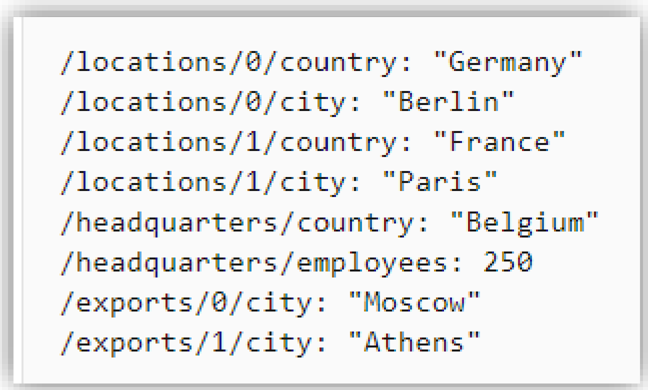


JSON

```
{
    "locations": [
        { "country": "Germany", "city": "Berlin" },
        { "country": "France", "city": "Paris" }
    ],
    "headquarters": { "country": "Belgium", "employees": 250 },
    "exports": [
        { "city": "Moscow" },
        { "city": "Athens" }
    ]
}
```

```
SELECT location
FROM location IN company.locations
WHERE location.country = 'France'
```





```
/locations/0/country: "Germany"
/locations/0/city: "Berlin"
/locations/1/country: "France"
/locations/1/city: "Paris"
/headquarters/country: "Belgium"
/headquarters/employees: 250
/exports/0/city: "Moscow"
/exports/1/city: "Athens"
```

Time to Live (TTL)

Time to Live (TTL)

- You can set the expiry time for Cosmos DB data items
- Time to live value is configured in seconds.
- The system will automatically delete the expired items based on the TTL value
- Consume only leftover Request units
- Data deletion delay if not enough Request units
  - Though the data deletion is delayed, data is not returned by any queries (by any API) after the TTL has expired.

Multi Reads

Performance

- Ensures high availability within a region
- Across regions, brings data closer to the consumer.

Business continuity

- In the event of major failure or natural disaster

Multi Writes

Performance

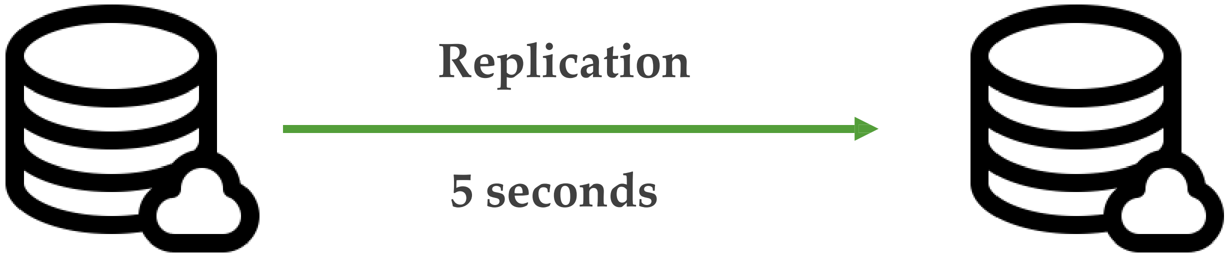
- Ensures high availability within a region
- Across regions, brings data closer to the consumer.

Business continuity

- In the event of major failure or natural disaster

Manual – Automatic Failover

Data consistency



Replication

5 seconds

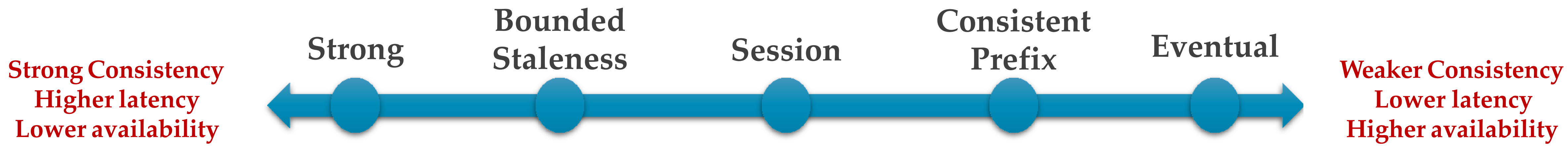
West US

East US

At 10.00 AM
Update CreditScore = 750

At 10:00:02 AM
Read CreditScore

Five consistency Levels



Strong: No dirty reads, high latency, cost highest, closest to RDBMS

Bounded staleness: Dirty reads possible, bounded by time and updates

Session: No dirty reads for writers (within same session), dirty read possible for other users

Consistency prefix: Dirty reads possible but sequence maintain, reads never see out-of-order writes

Eventual: No guaranteed order, but eventually everything gets in order

Setting the consistency level

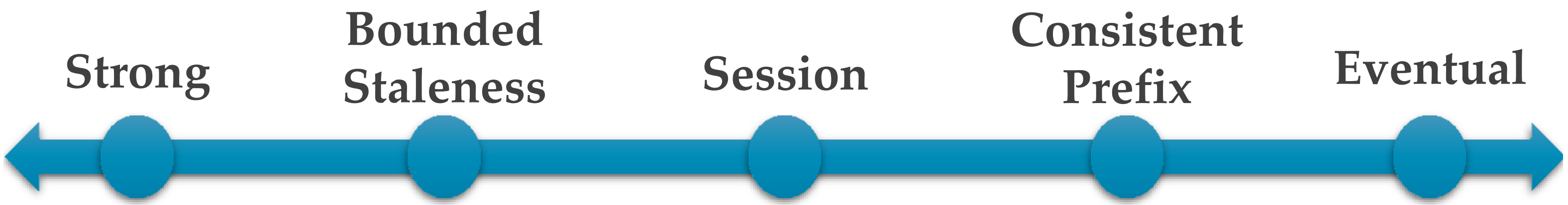
Set default for entire account

Can be changed any time

Override at request level

Any request can weaken the default consistency level

Strong Consistency
Higher latency
Lower availability



Strong

Bounded Staleness

Session

Consistent Prefix

Eventual

Weaker Consistency
Lower latency
Higher availability

Thank you!