Why Cosmos DB?

What traditional databases were lacking?

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

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.

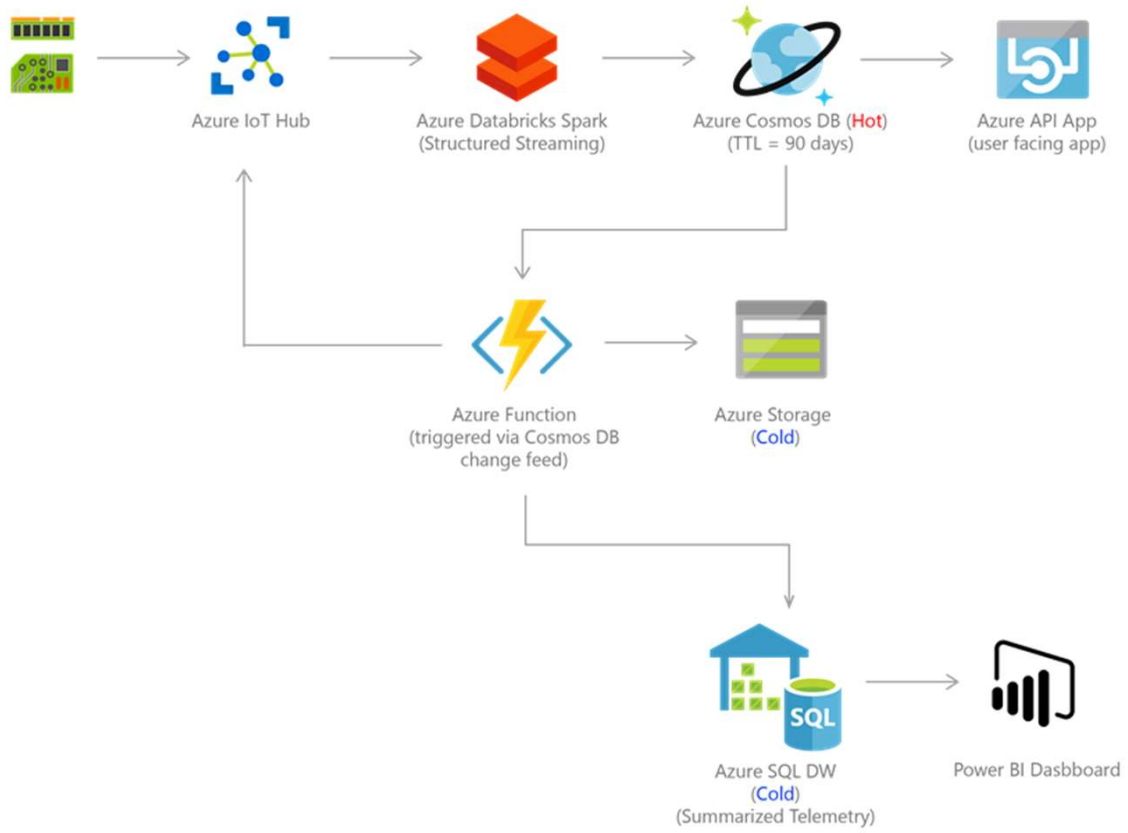
MULTIMODEL & MULTI-LANGUAGE

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

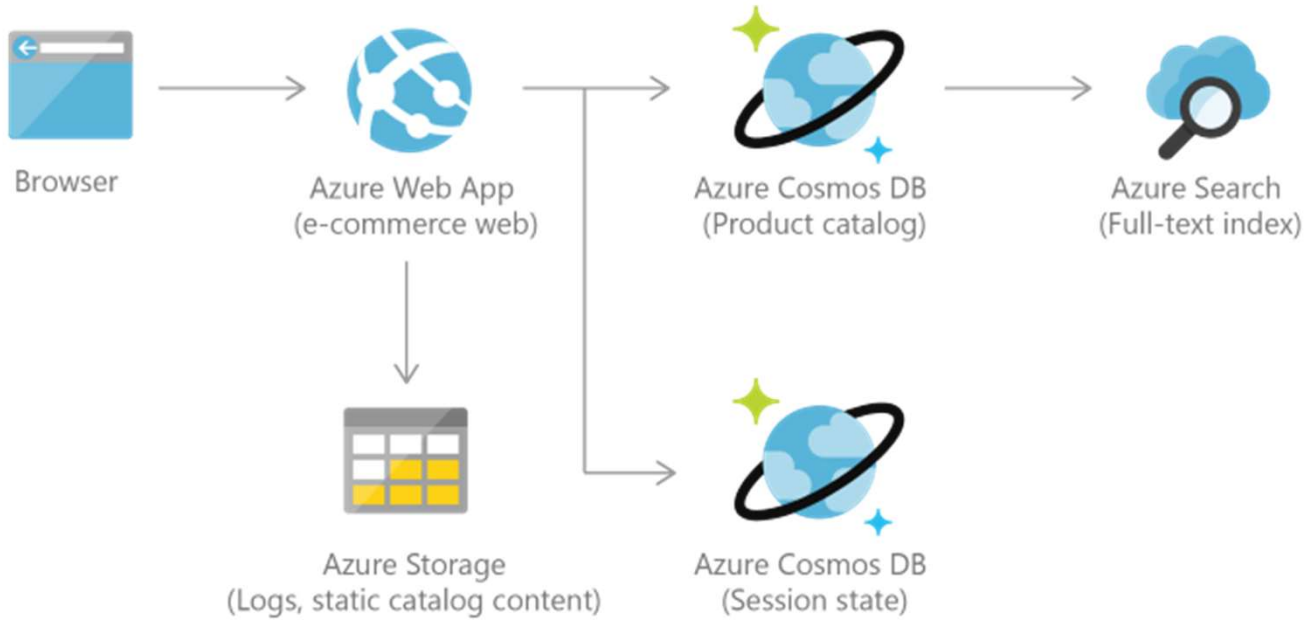
HIGHLY AVAILABLE, RELIABLE & SECURE

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

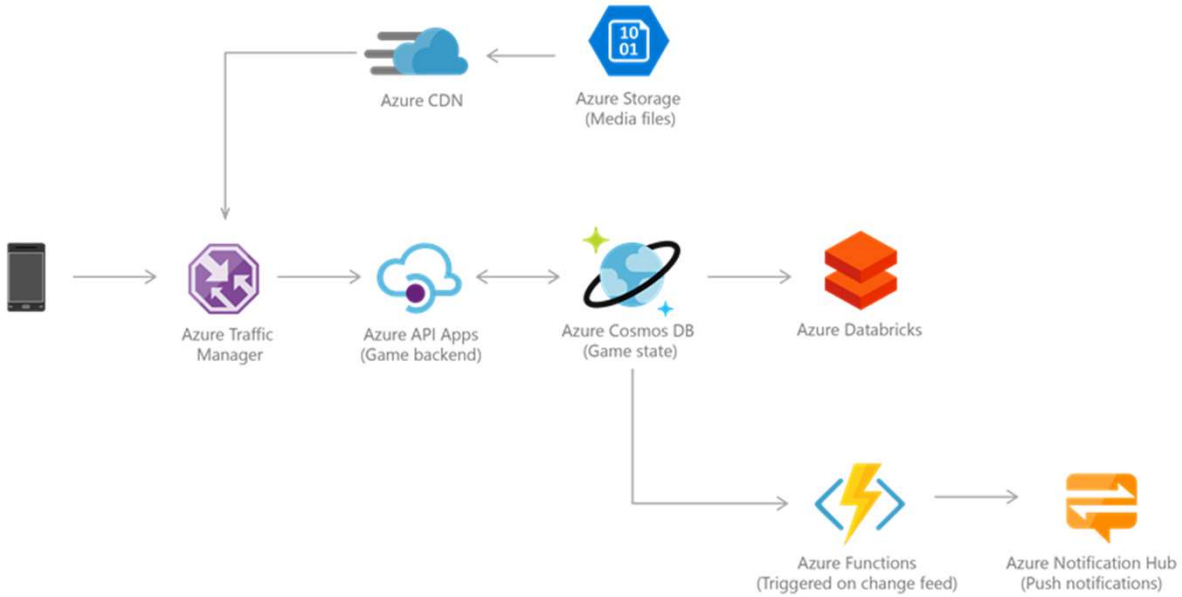
Use case - IOT


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

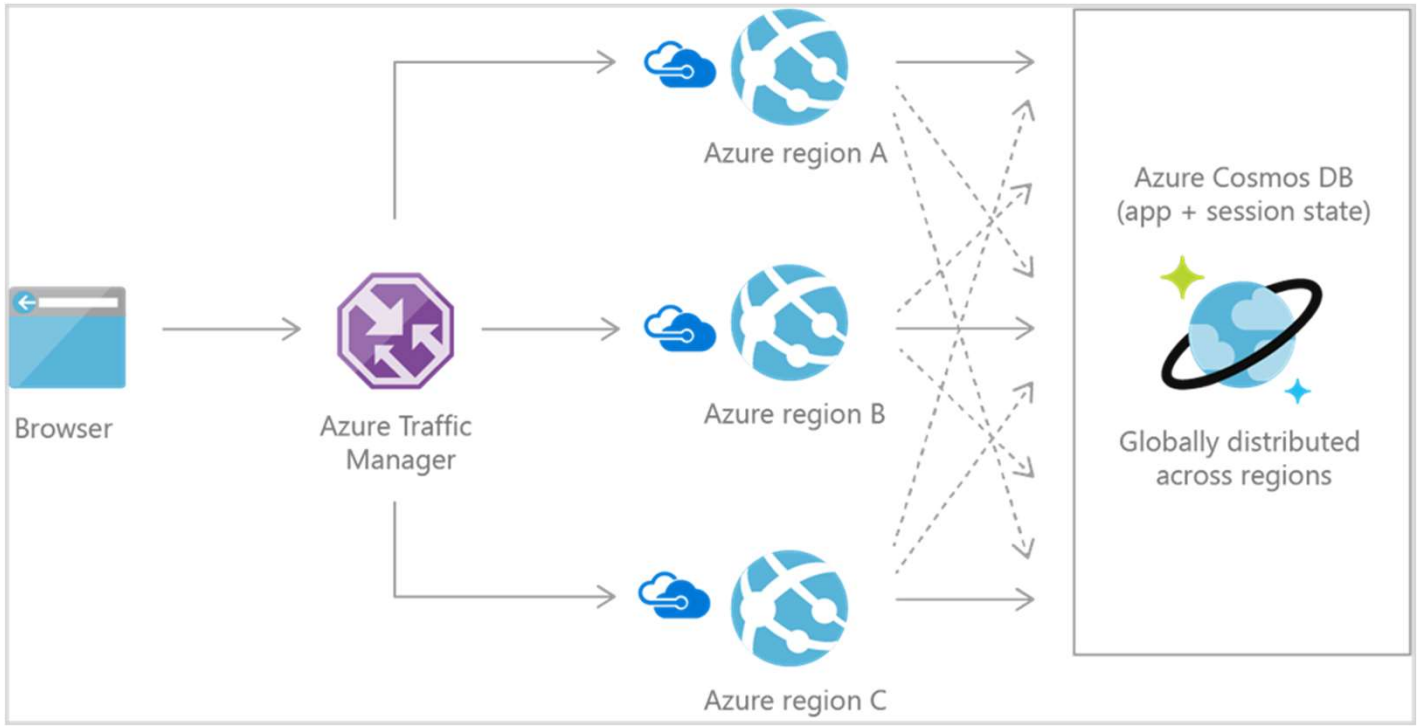
Use case – Retail and Marketing

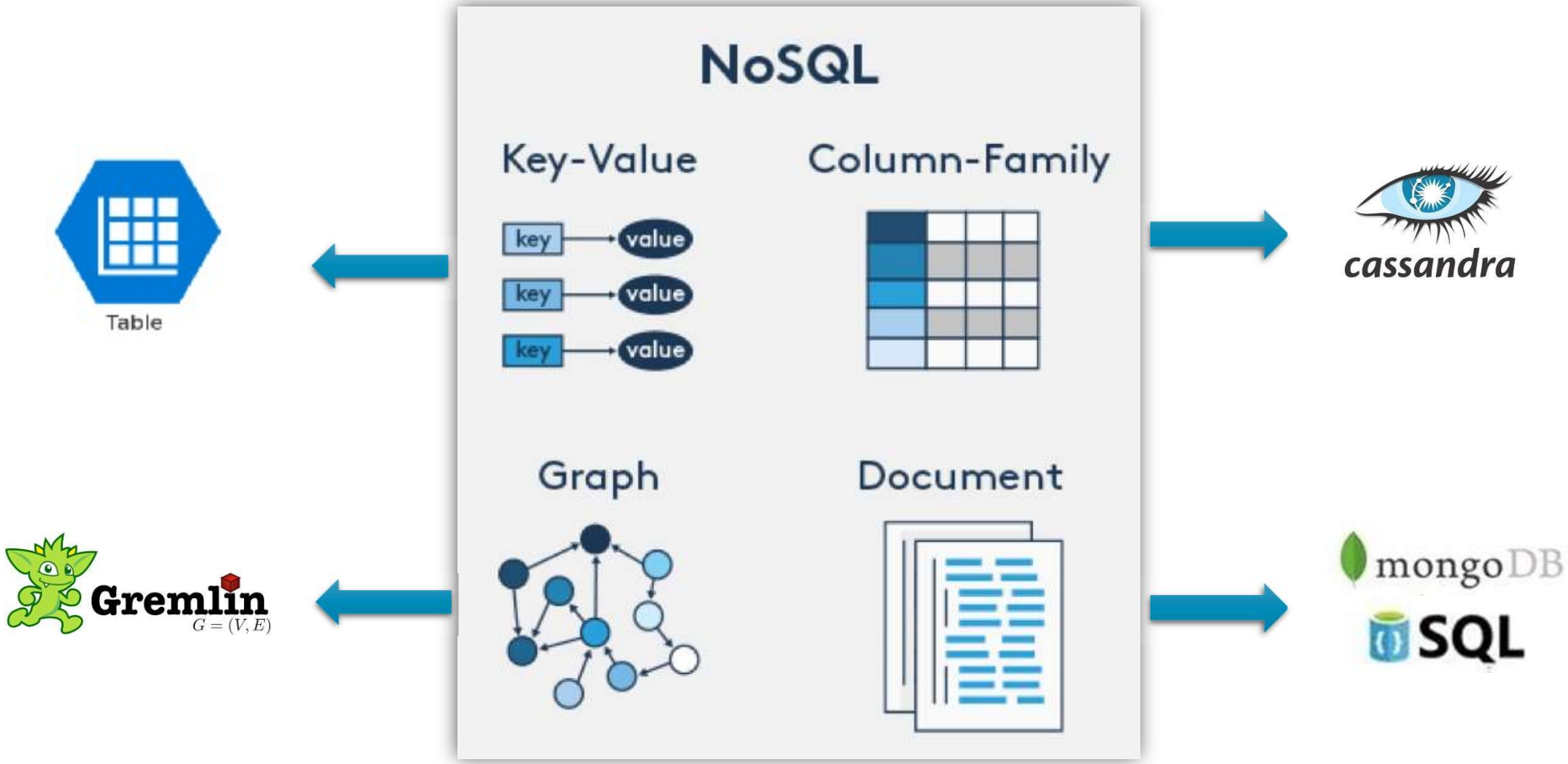


Use case – Gaming



Use case – Web and mobile




NoSQL
Key-Value
key
value
key
value
key
value
Column-Family
Graph
Document
Table
Gremlin
$G = (V, E)$
cassandra
mongoDB
SQL

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

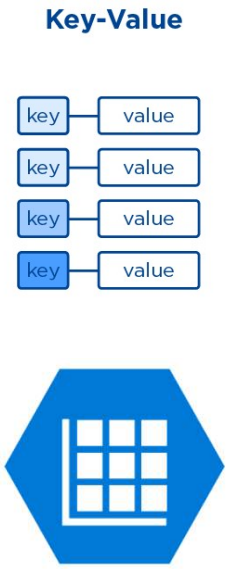
Implement Wire protocol

Fully compatible with Mongo DB application code

Migrate existing Cosmos DB without much change of logic

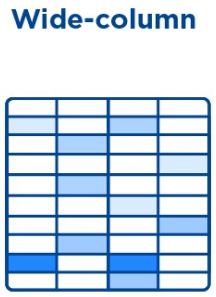
Use SQL(CORE) API for new development

Cosmos DB Table API



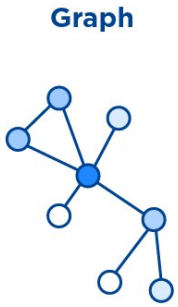
- Key-Value store
- Premium offering for Azure Table Storage
- Existing Table Storage customers will migrate to Cosmos DB Table API
- Row value can be simple like number or string
- Row cannot store object

Cosmos DB Cassandra API



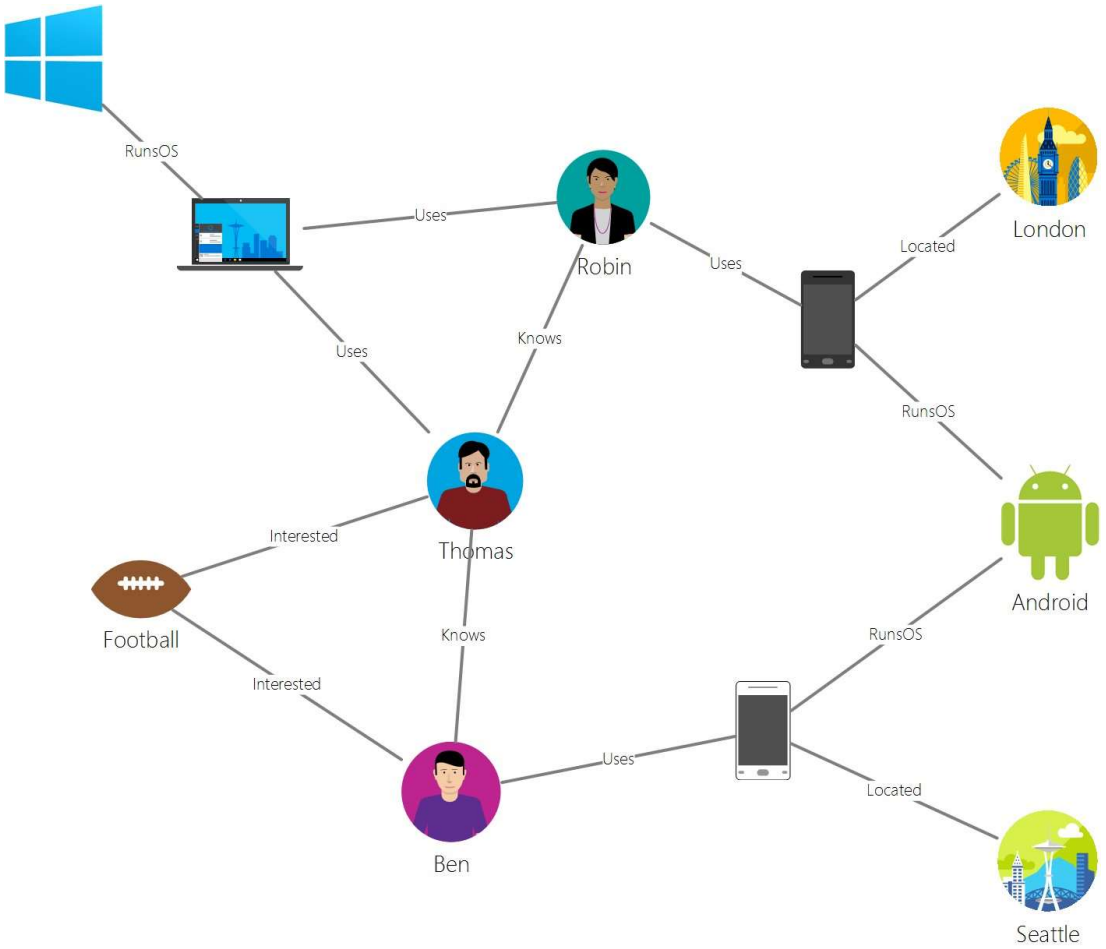
- Wide column No SQL Database
- Name and format of column can vary from row to row.
- Simple migrate your Cassandra application to Cosmos Cassandra API and change connection string.
- Interact
  - Cassandra based tools
  - Data Explorer
  - Programmatically, using SDK (CassandraCSharpdriver)

Cosmos DB Gremlin API

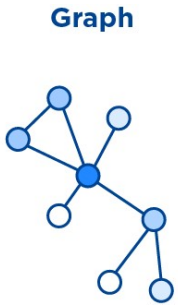


- Graph Data Model
- Real world data connected with each other
- Graph database can persist relationships in the storage layer

Graph Model


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

Cosmos DB Gremlin API



- 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

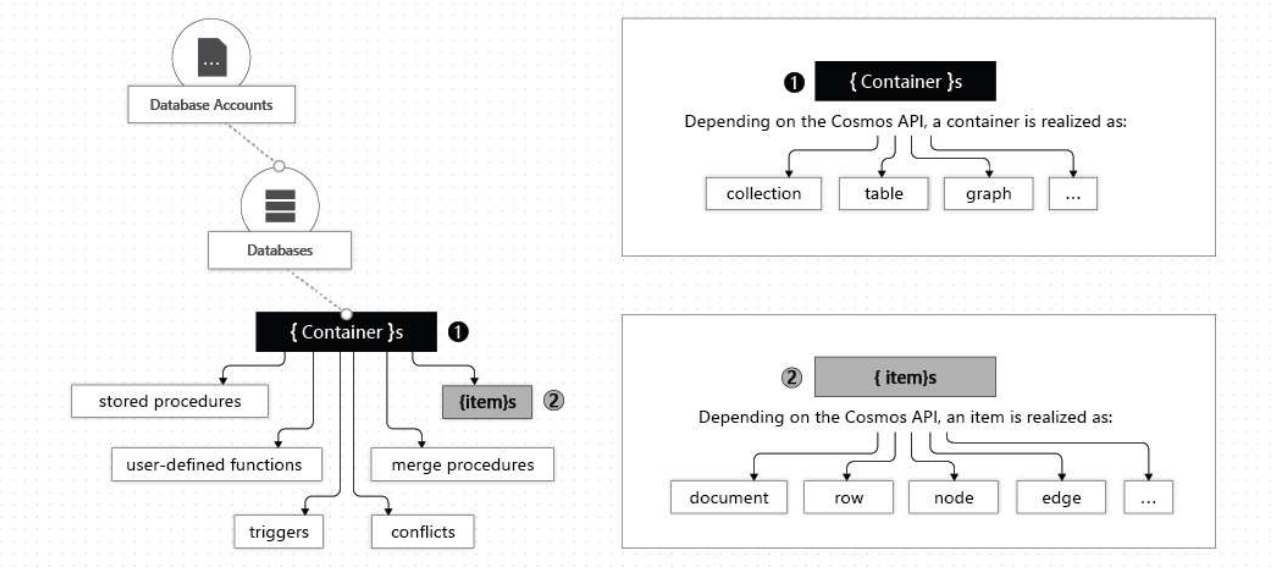
Azure Table Storage

- 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

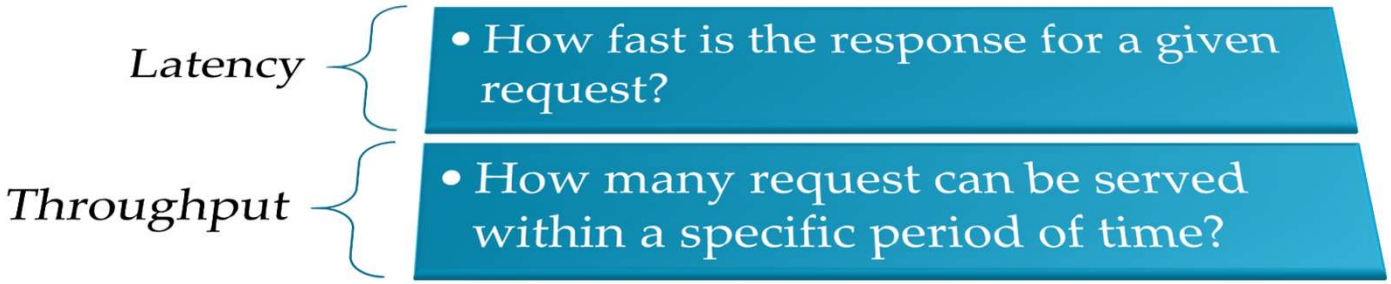
- Geo replication across your choice of any number of regions
- Secondary index support for lookups across multiple dimensions
- Better performance
  - Unlimited and predictable throughput
  - latency is lower
- 5 consistency options

Database Containers and Items

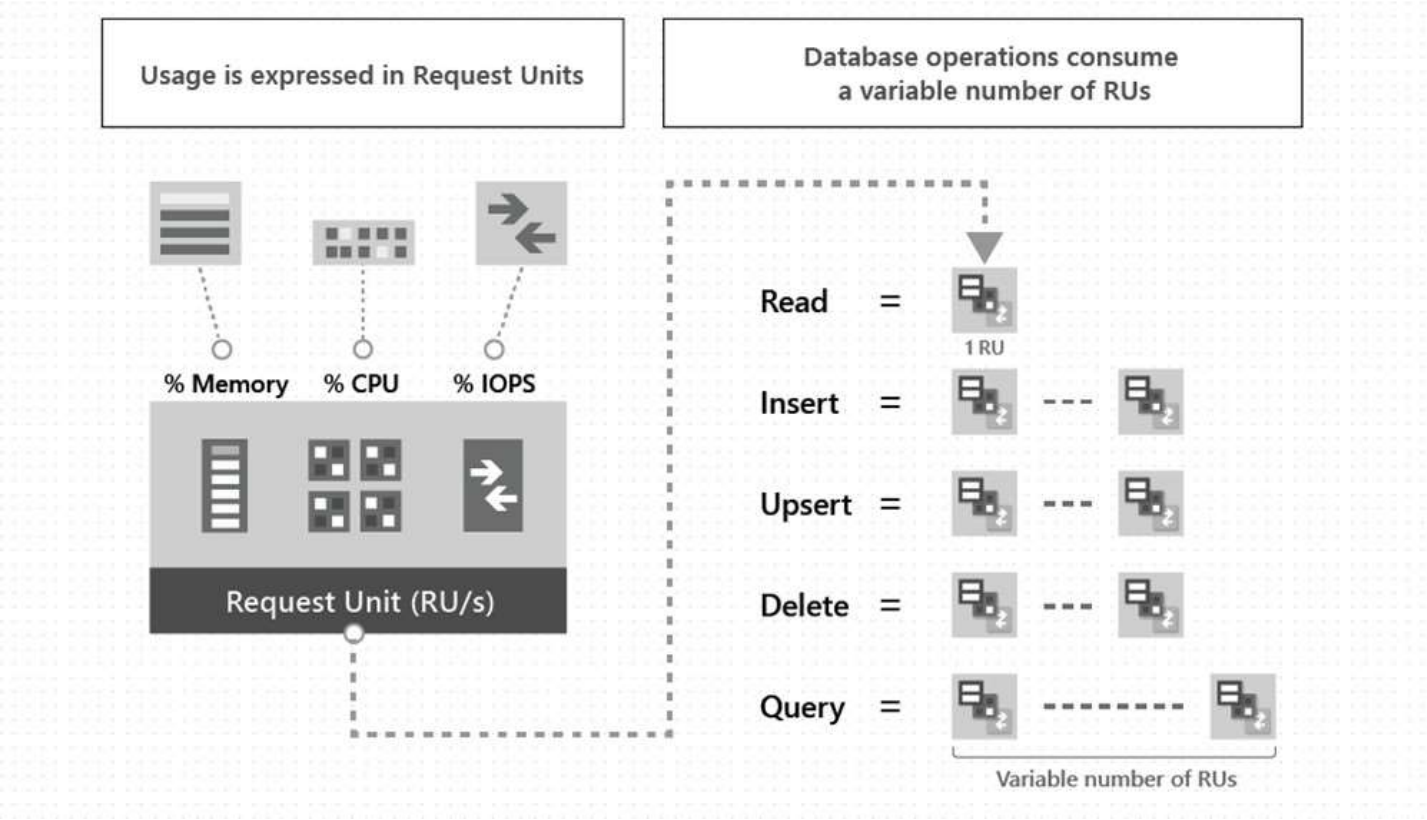


Azure Cosmos entity	SQL API	Cassandra API	MongoDB API	Gremlin API	Table API
Azure Cosmos database	Database	Keyspace	Database	Database	NA
Azure Cosmos item	Document	Row	Document	Node or edge	Item

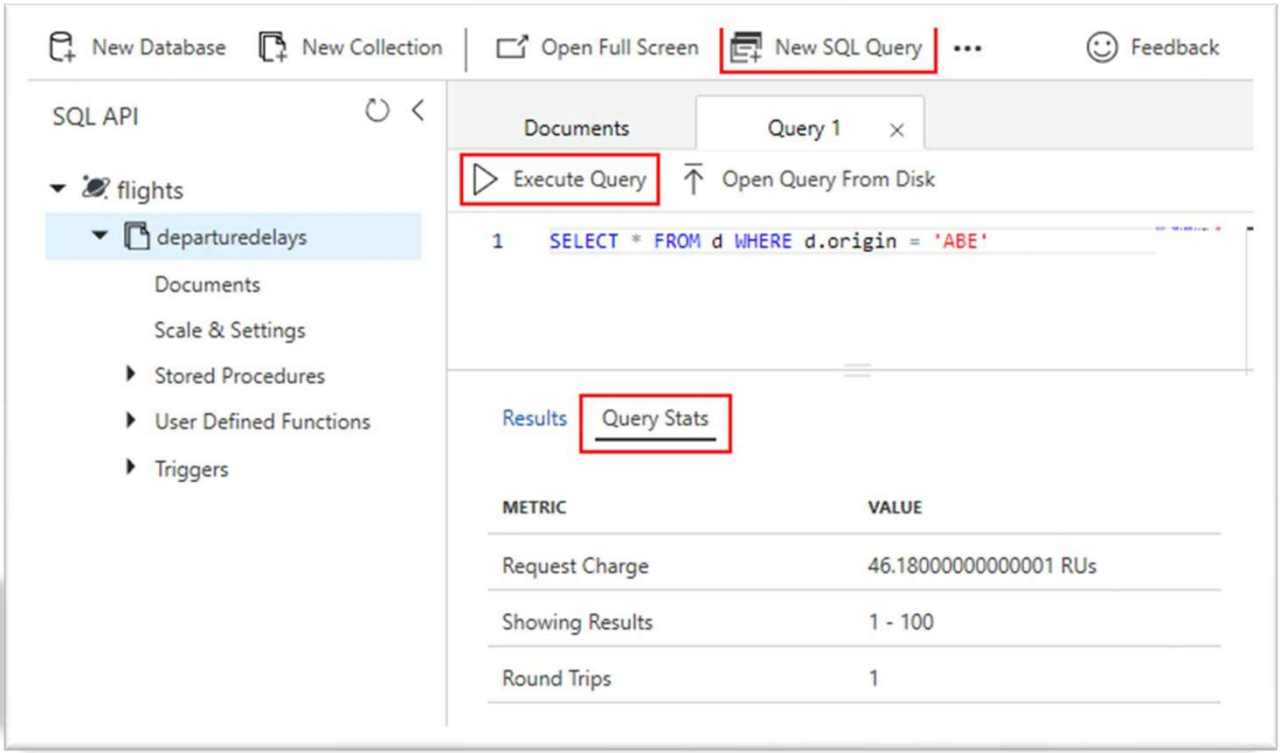
Measuring Performance



Introducing Request Units

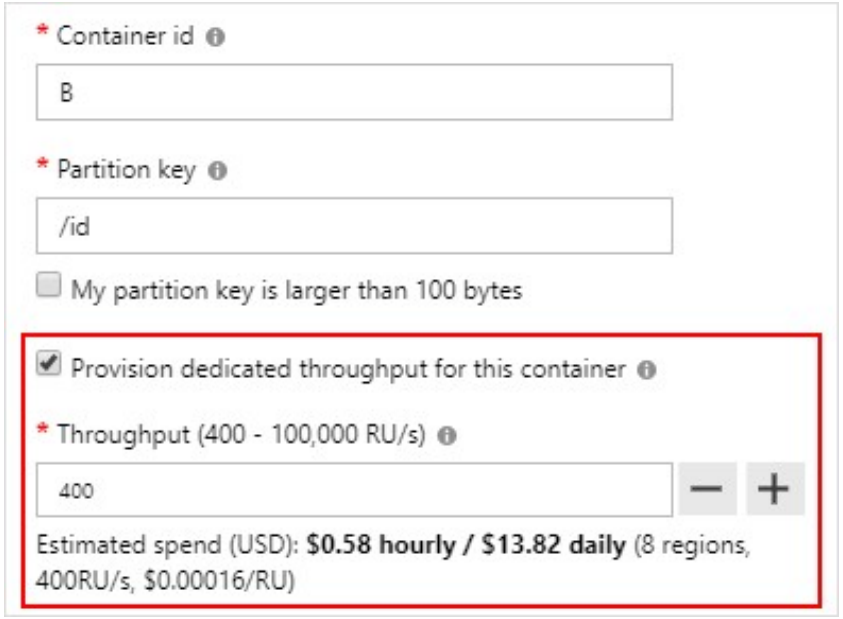


Introducing Request Units

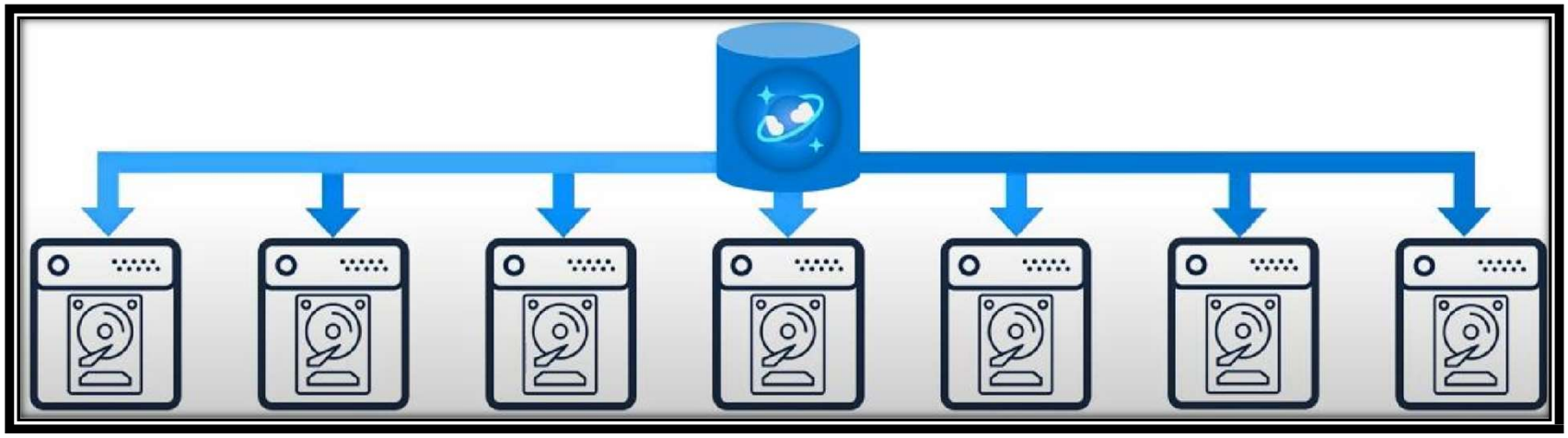


Reserving requests units

- 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)



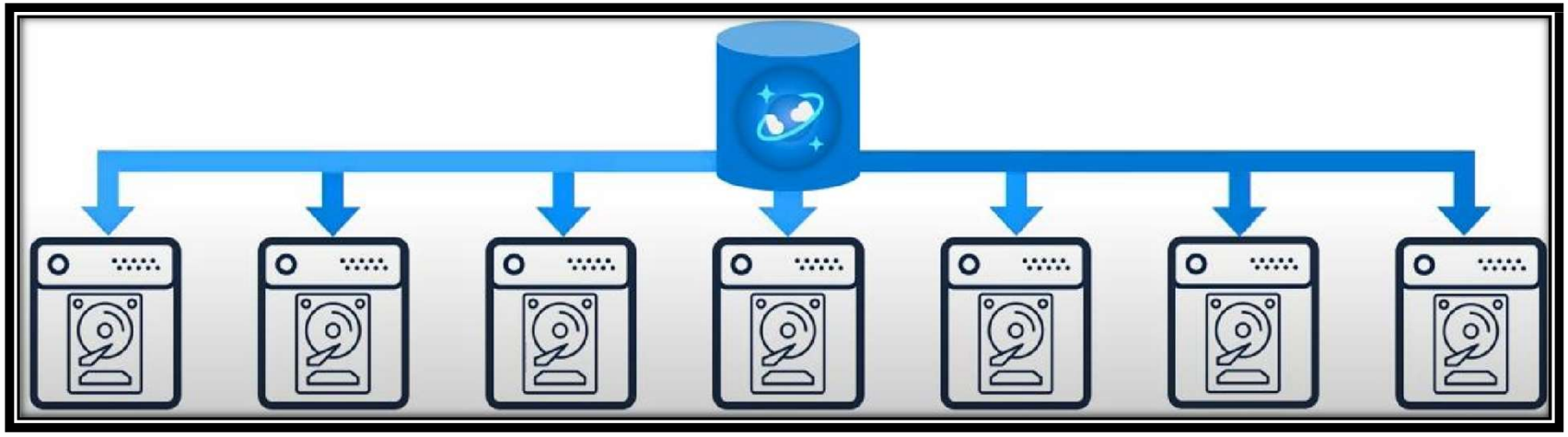
Horizontally Scalable



Unlimited Storage

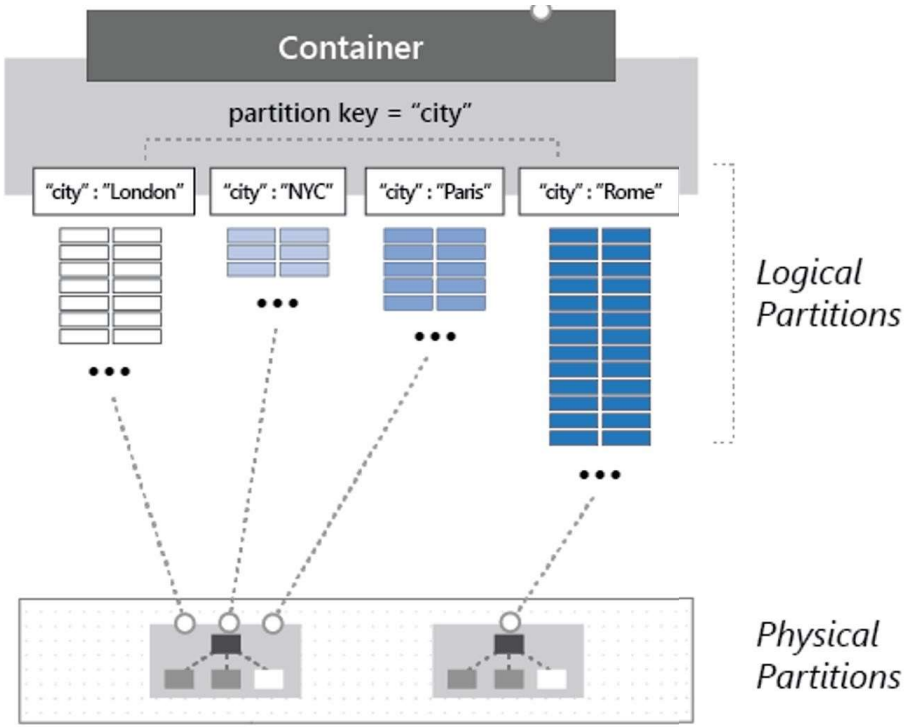
Unlimited Throughput

Partitioning

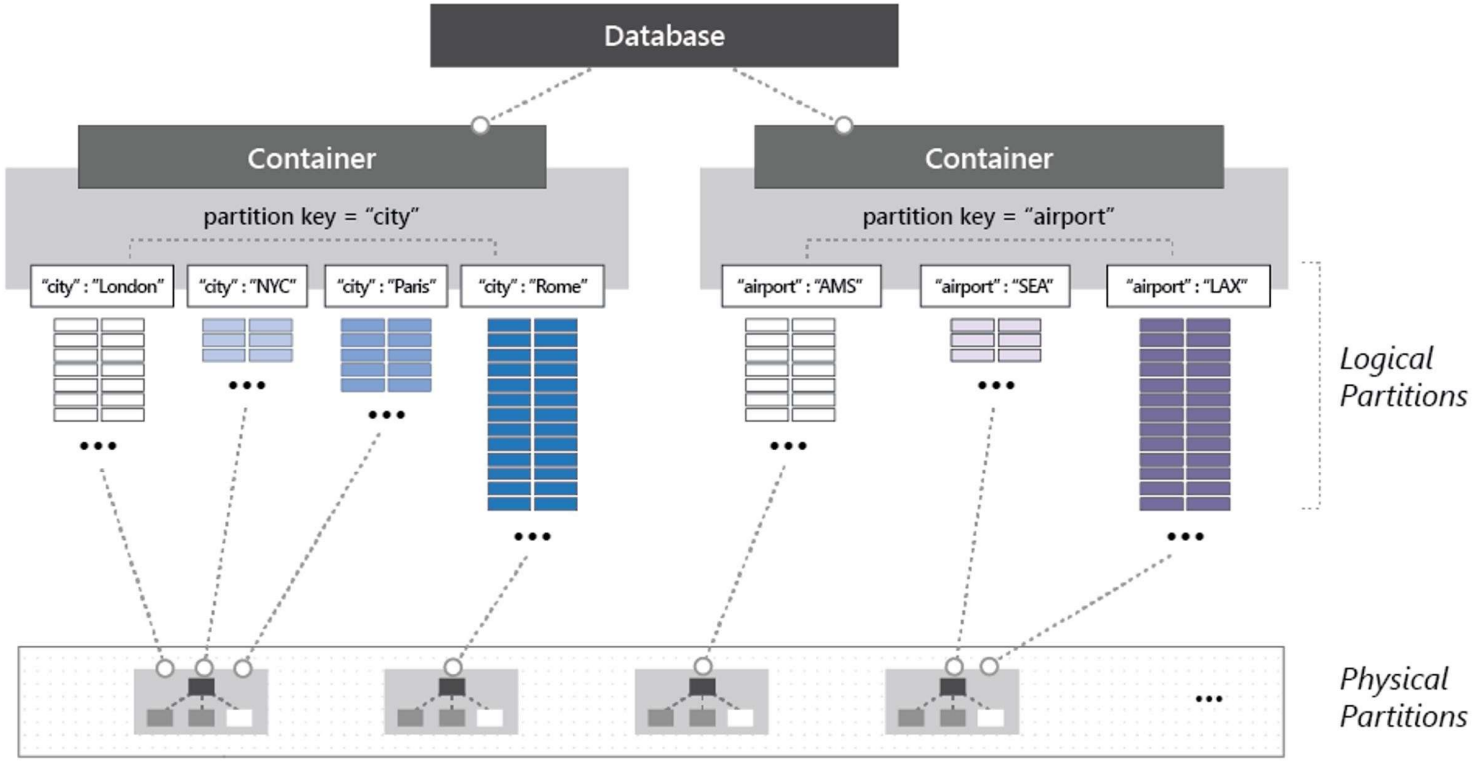


Partitioning

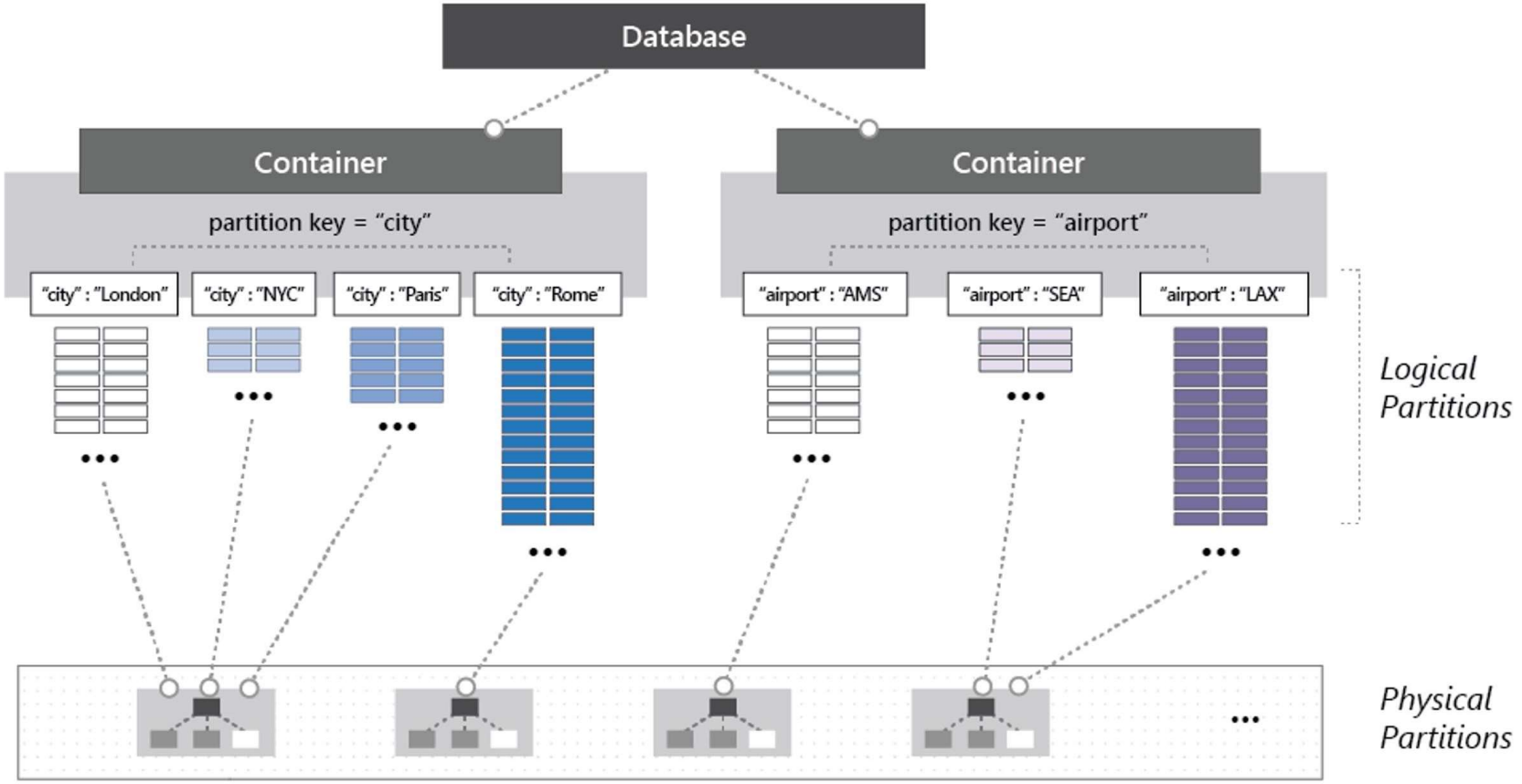
- **Partitioning:** the items in a container are divided into distinct subsets called logical partitions.
- **Partition key** is the value by which Azure organizes your data into logical divisions.
- **Logical partitions** are formed based on the value of a partition key that is associated with each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



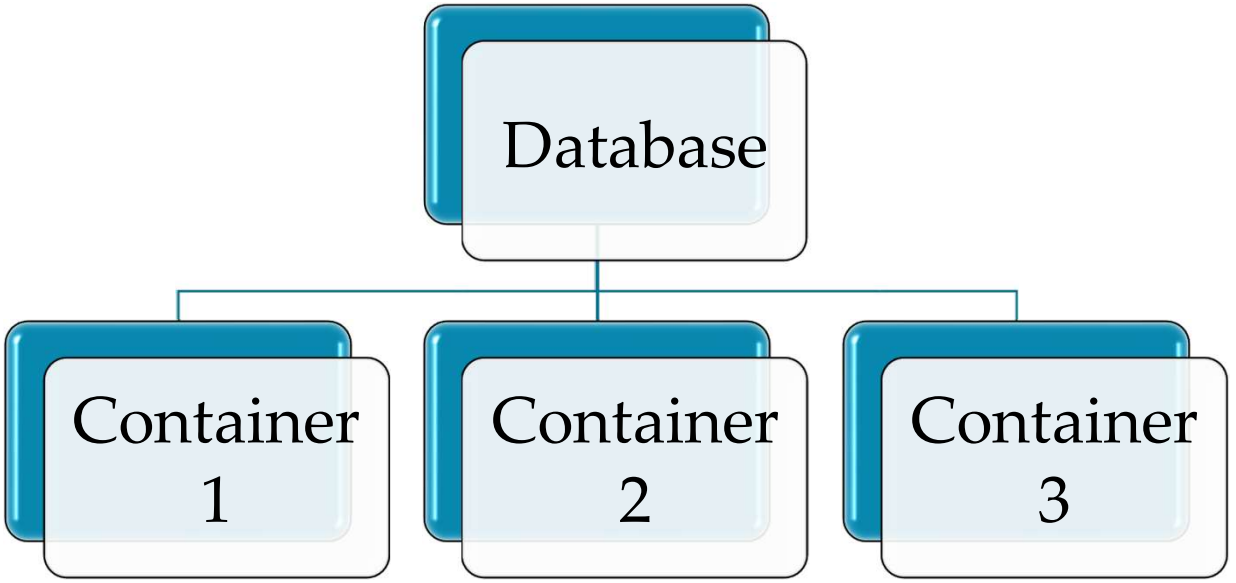
Partitioning



Dedicated vs Shared throughput

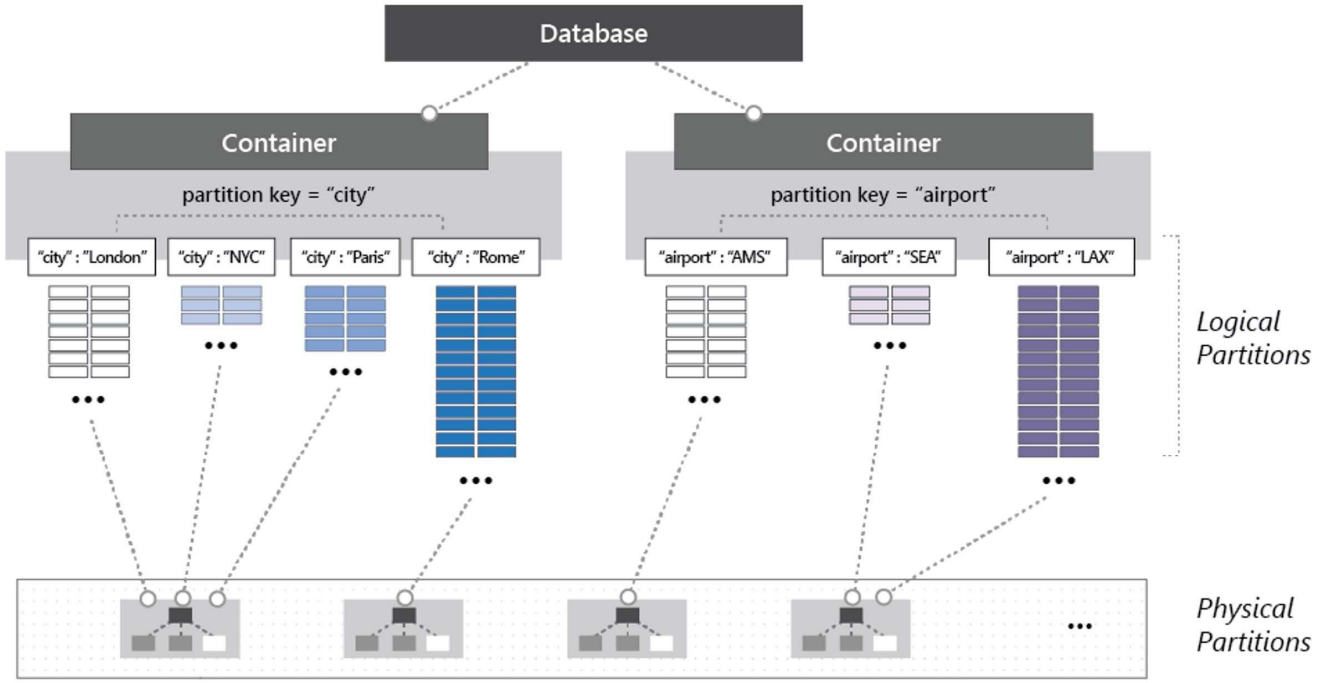


Dedicated vs Shared throughput

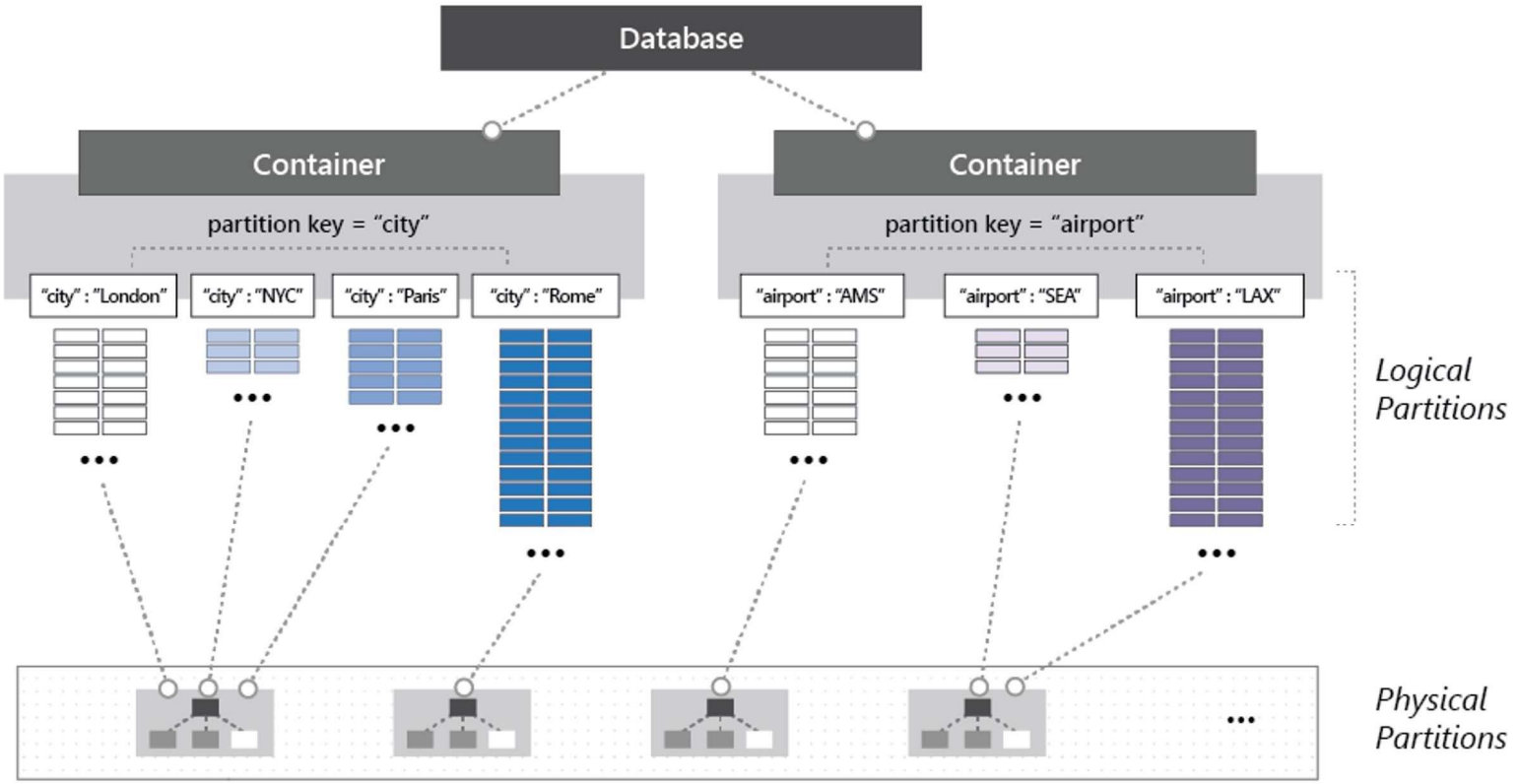


Dedicated vs Shared throughput

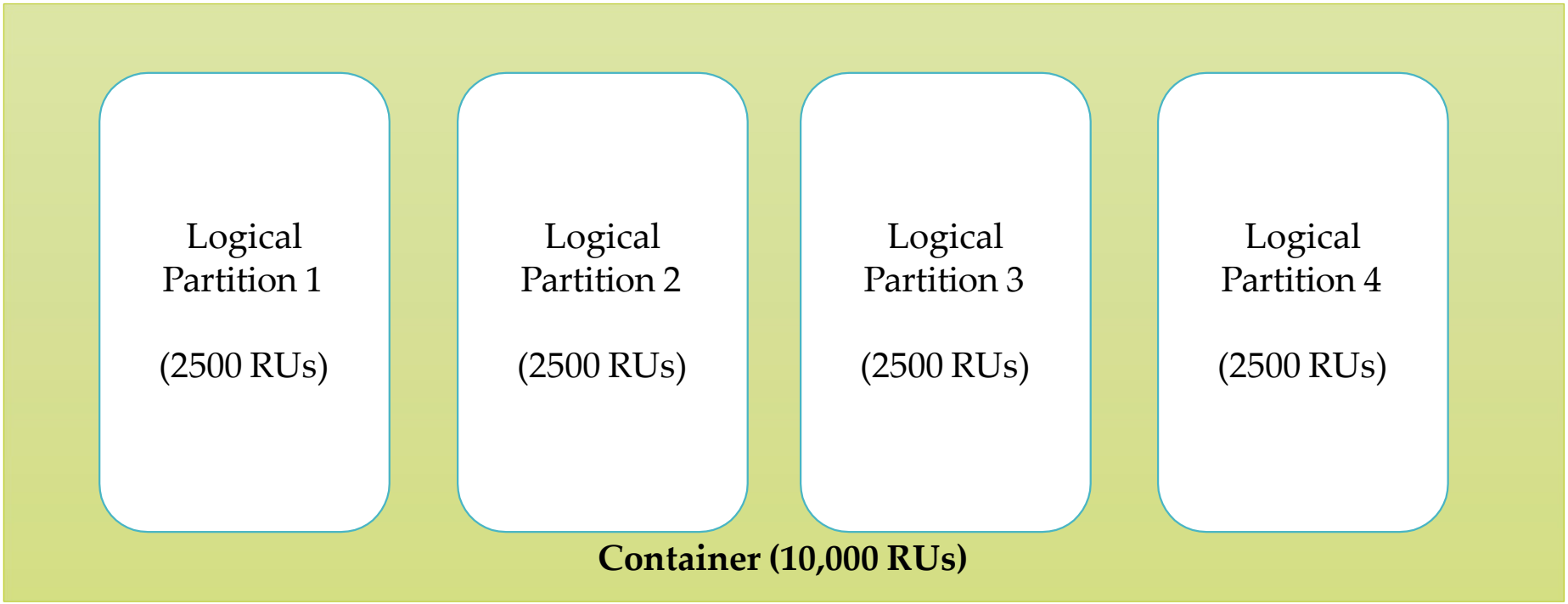
- You can set throughput at:
  - Database level – Shared throughput
  - Container level – Dedicated throughput
  - It is recommend to set throughput at container level.
- Rate-Limited
- Choose at the time of creation



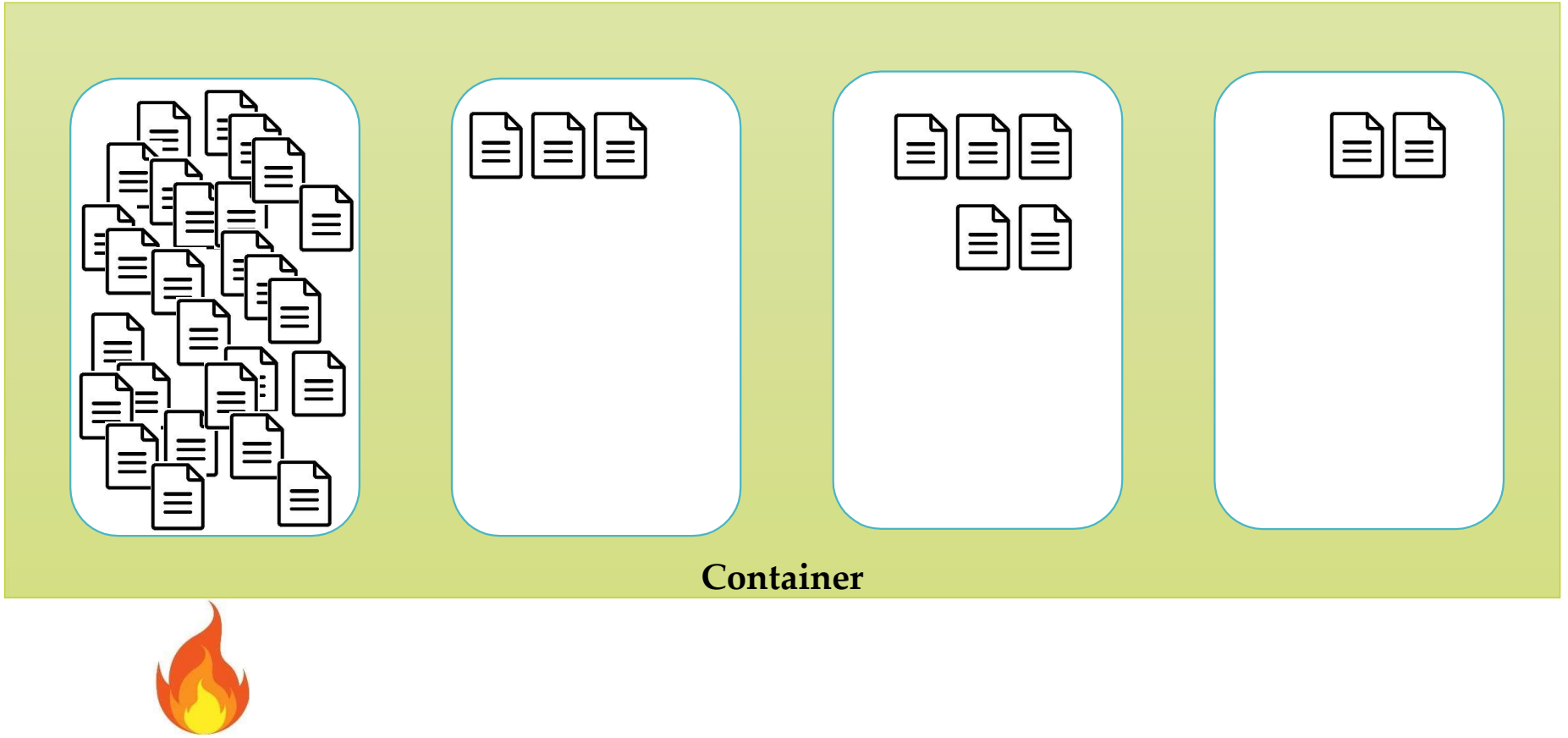
Avoiding hot partition



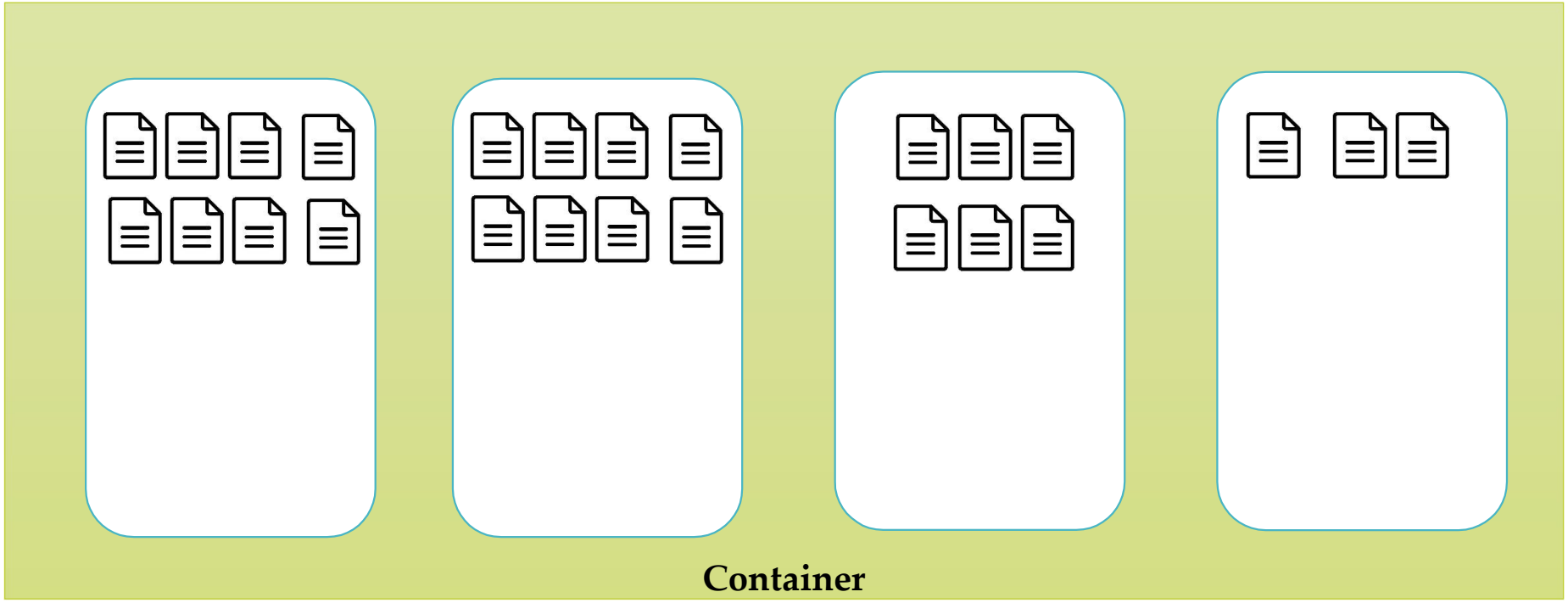
Avoiding Hot Partitions



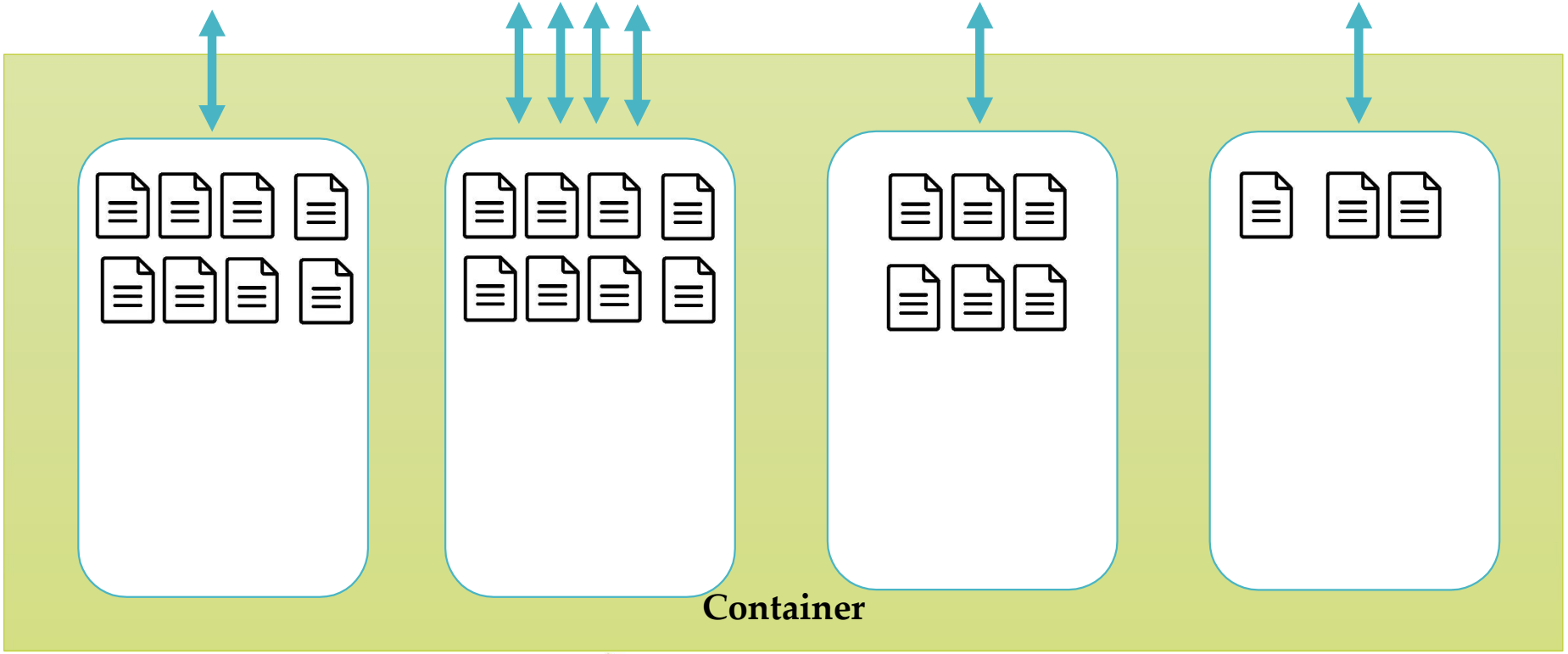
Avoid Hot partitions on storage



Avoiding Hot Partitions at store

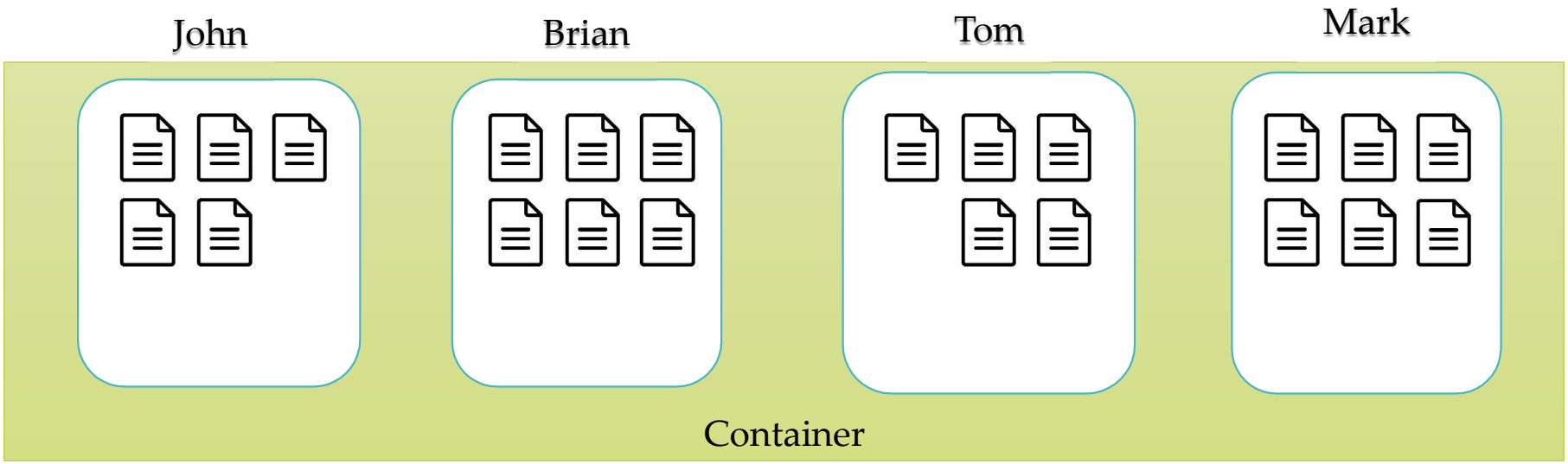


Avoid Hot partitions on throughput



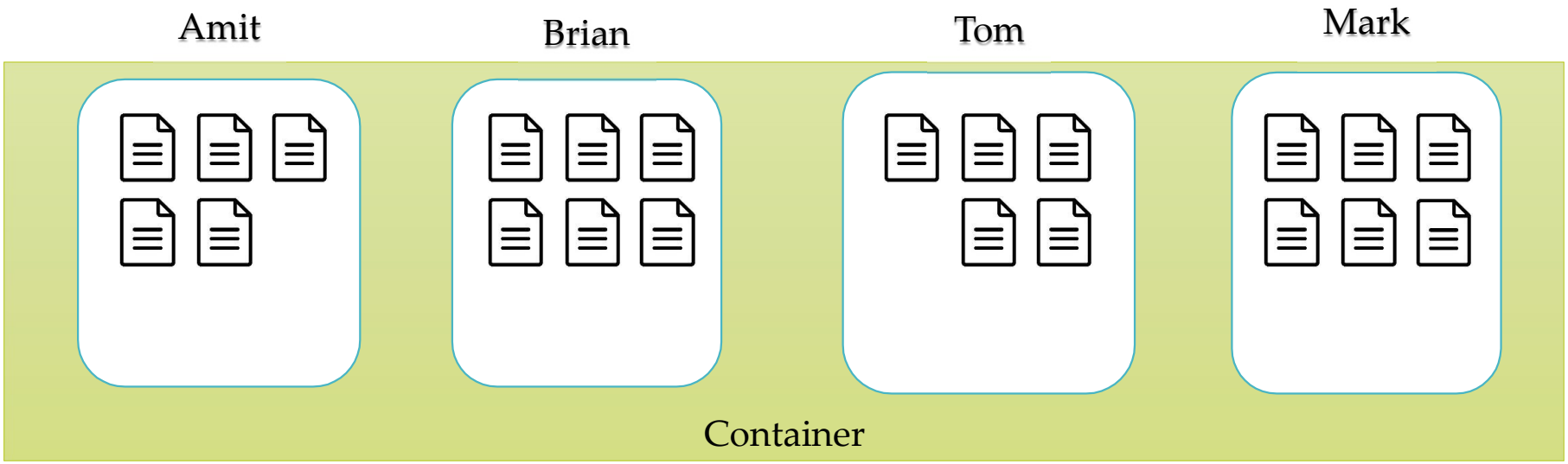
- **Partition key Bad choice:** Current time
- **Partition key Good choices:** User ID, Product ID

Single partition Query



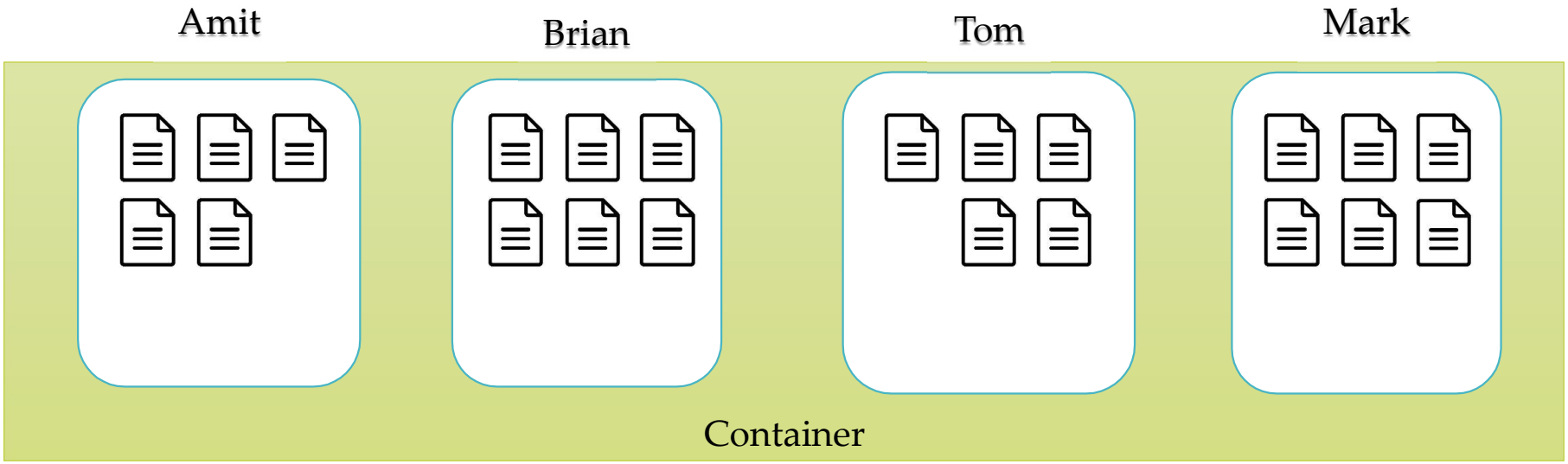
SELECT * FROM c WHERE c.username = 'Brian'

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
  - 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 vehicle telemetry data generated from millions of vehicles every second. Which of the following options for your Partition Key will optimize storage distribution?

Answer choices:

1. Vehicle model
2. Vehicle Identification Number (VIN) which looks like WDDEJ9EB6DA032037

Choosing a Partition key

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

Answer choices:

1. Vehicle model

   Most auto manufactures only have a couple dozen models. This option is potentially the least granular, will create a fixed number of logical partitions, and may not distribute data evenly across all physical partitions.

2. **Vehicle Identification Number (VIN) which looks like WDDEJ9EB6DA032037**

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

Automatic Indexing

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

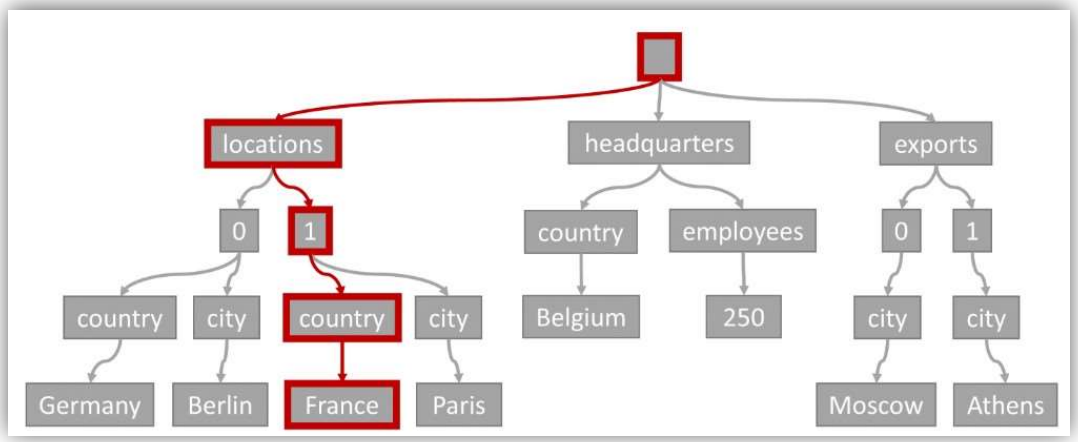
  or delete items
- Not specific for SQL, but available for all APIs

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

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

Global Distribution benefits

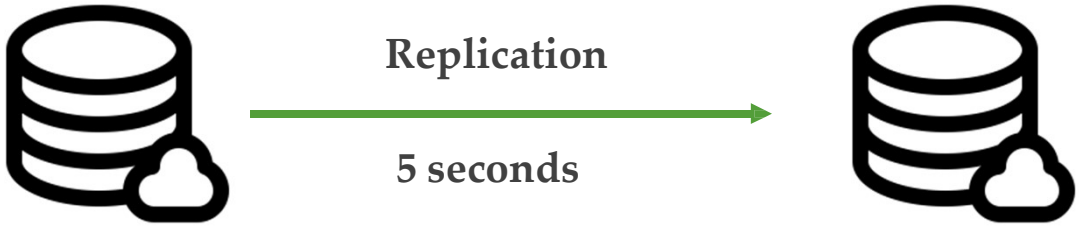
Performance

- 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

Data consistency



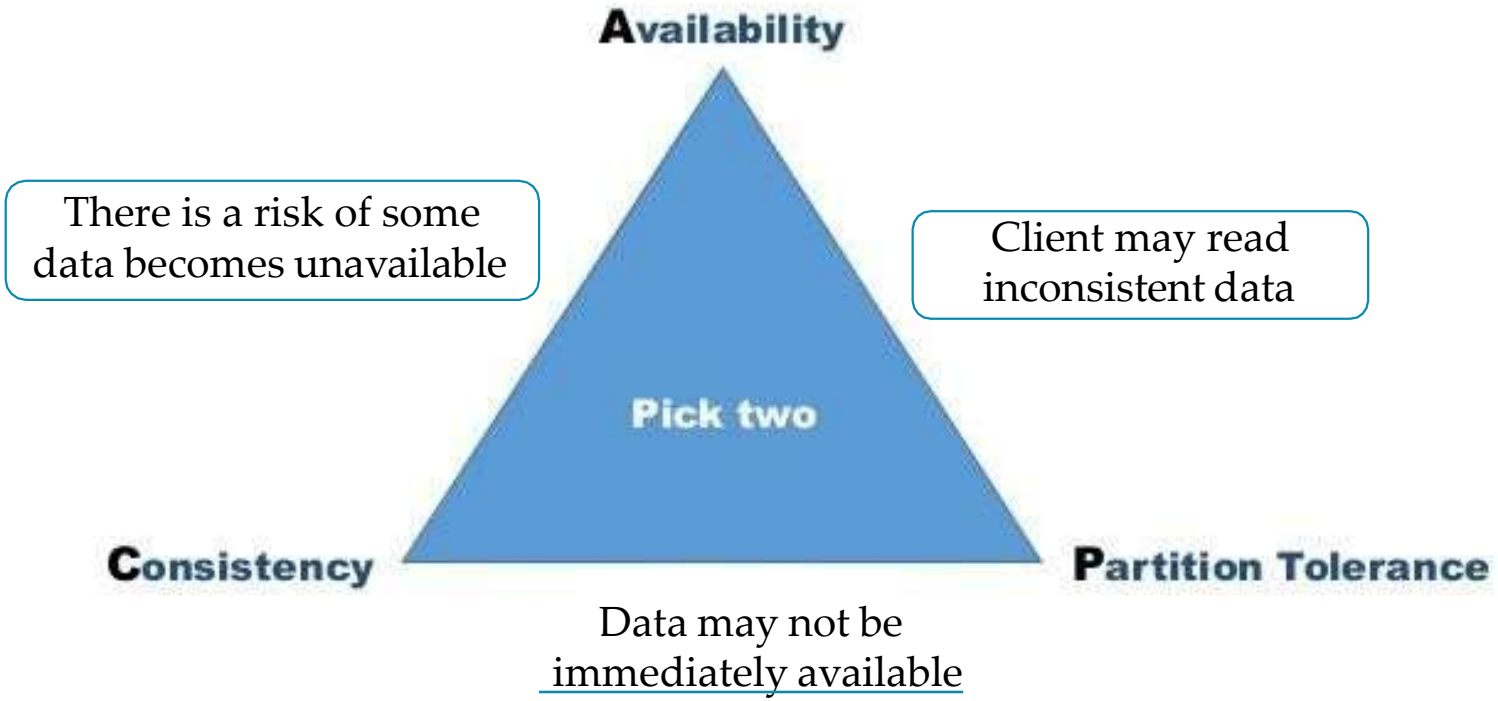
West US

At 10.00 AM
Update CreditScore = 750

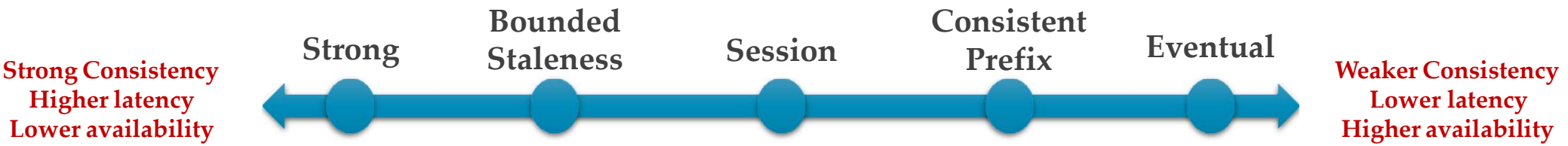
East US

At 10:00:02 AM
Read CreditScore

CAP Theorem


Availability
There is a risk of some data becomes unavailable
Client may read inconsistent data
Pick two
Consistency
Partition Tolerance
Data may not be immediately available

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