Andre Essing

Cosmos DB - NoSQL strikes back

Introduction to the dark side of data

Sponsors

Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Many thanks to our sponsors, without whom such an event would not be possible.

Sponsors

Many thanks to our sponsors, without whom such an event would not be possible.

Andre Essing

Technology Solutions Professional
Microsoft Deutschland GmbH

Andre advises customers in topics all around the Microsoft Data Platform. Since version 7.0, Andre gathering experience with the SQL Server product family. Today Andre concentrates on working with data in the cloud, like Modern Data Warehouse architectures, Artificial Intelligence and new scalable database systems like Azure Cosmos DB.

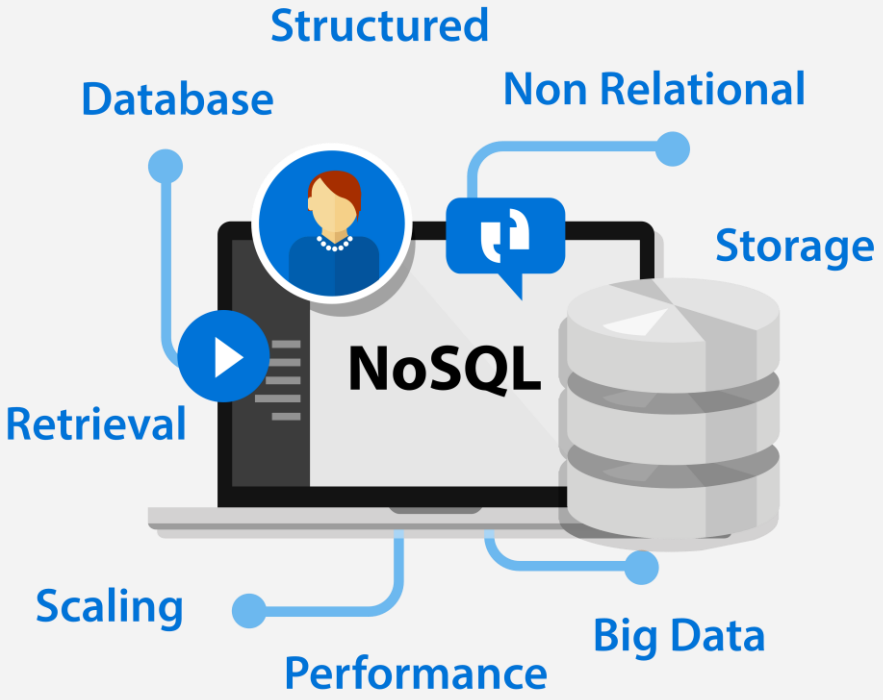
andre.essing@microsoft.com andreessing.de /aessing @aessing aessing

WHAT IS NOSQL

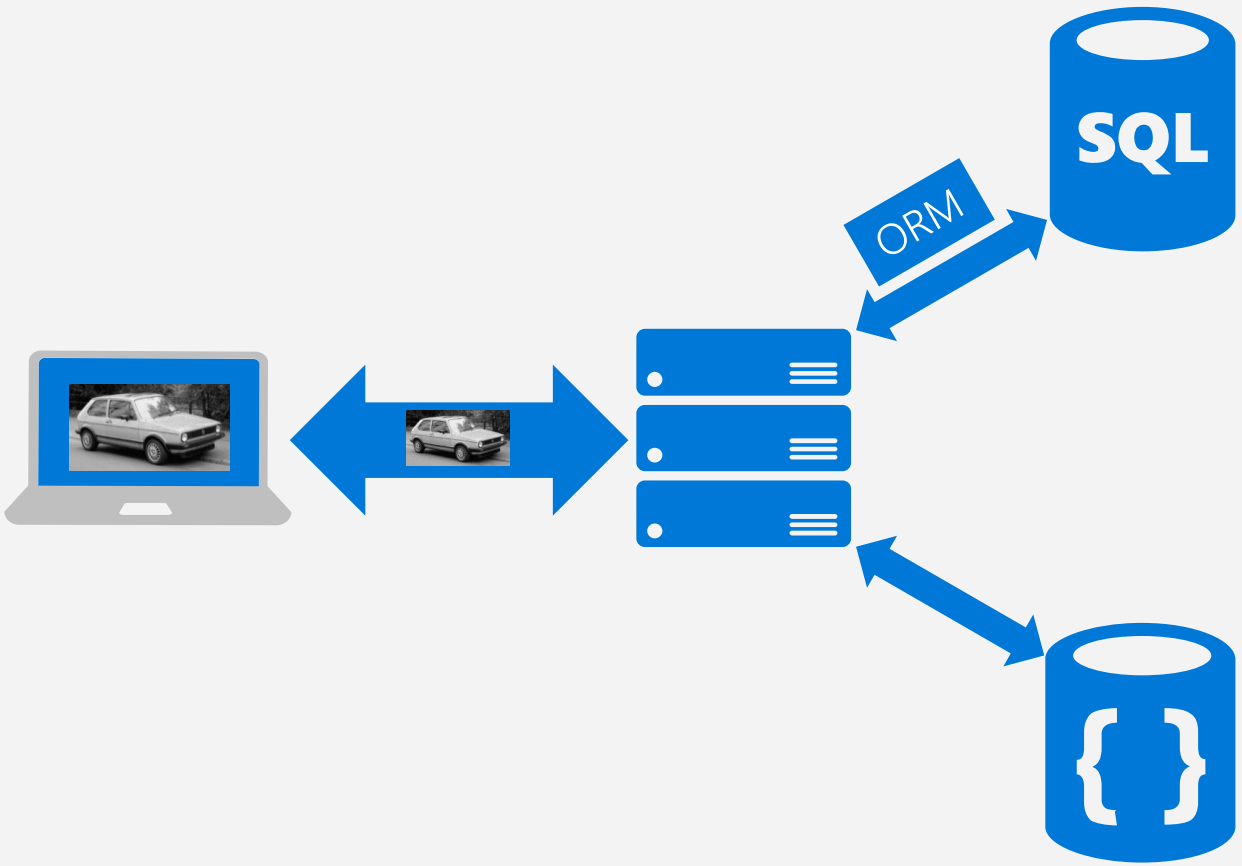
NOSQL, BUILT FOR SIMPLE AND FAST APPLICATION DEVELOPMENT

NoSQL, referring most times to “Non-SQL”, “Not Only SQL” or also “non-relational” is a kind of database where the data is modeled differently to relational systems.

- Different kinds available
  - Document
  - Key/Value
  - Columnar
  - Graph
  - etc.
- Non-Relational
- Schema agnostic
- Built for scale and performance
- Different consistency model



DIFFERENT WAYS OF STORING DATA WITH YOUR MODERN APP

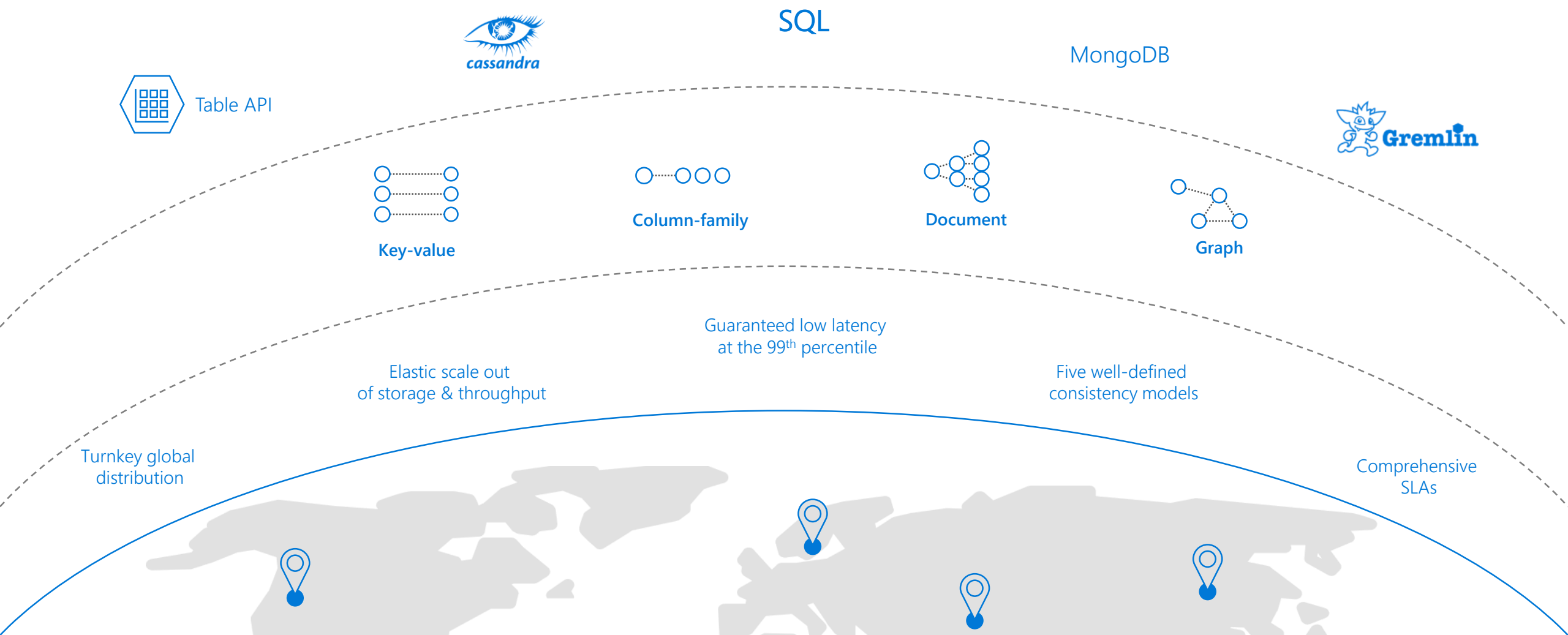


Data normalization

Come as you are

AZURE COSMOS DB

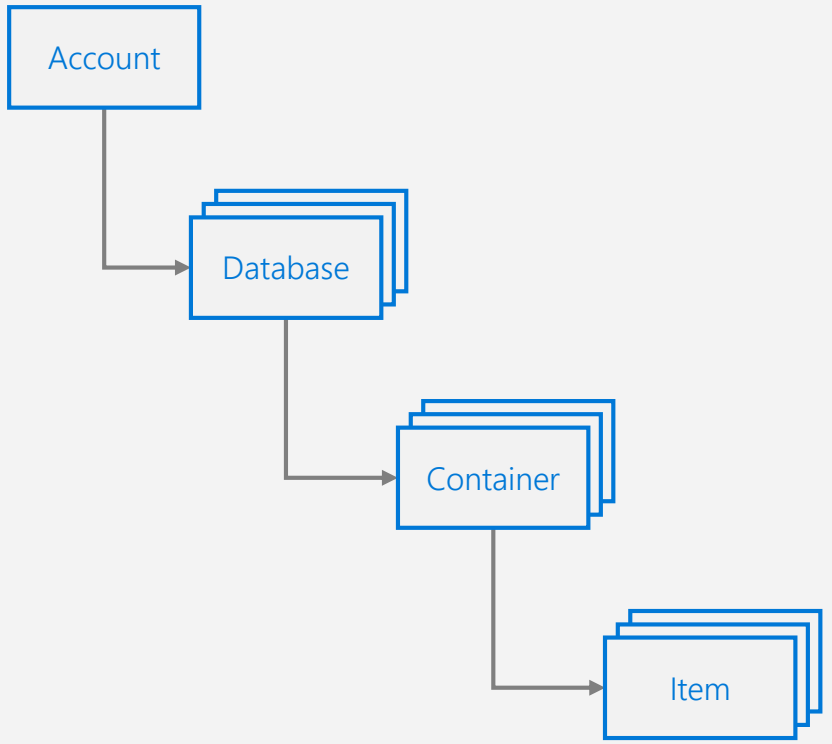
A globally distributed, massively scalable, multi-model database service



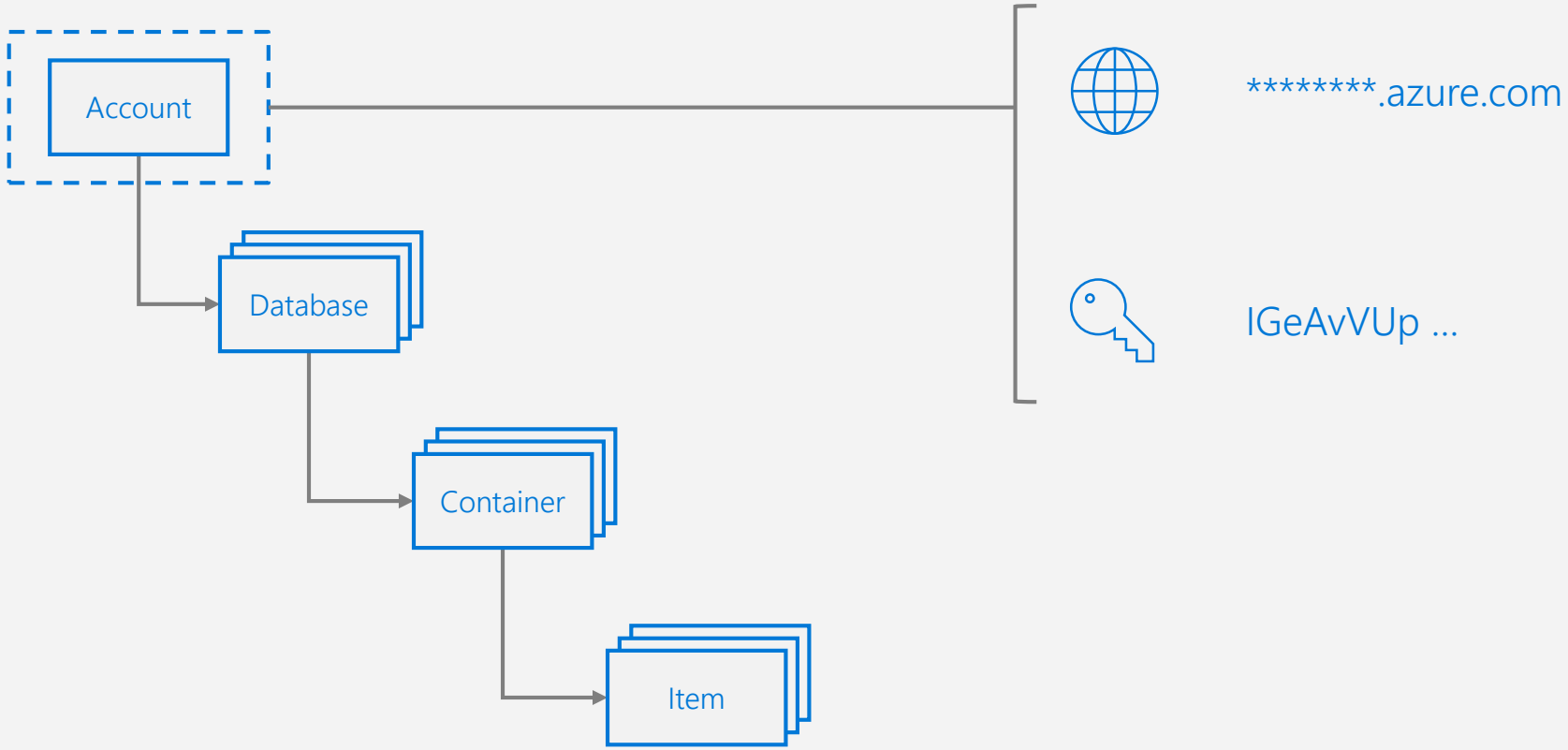
RESOURCE MODEL

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

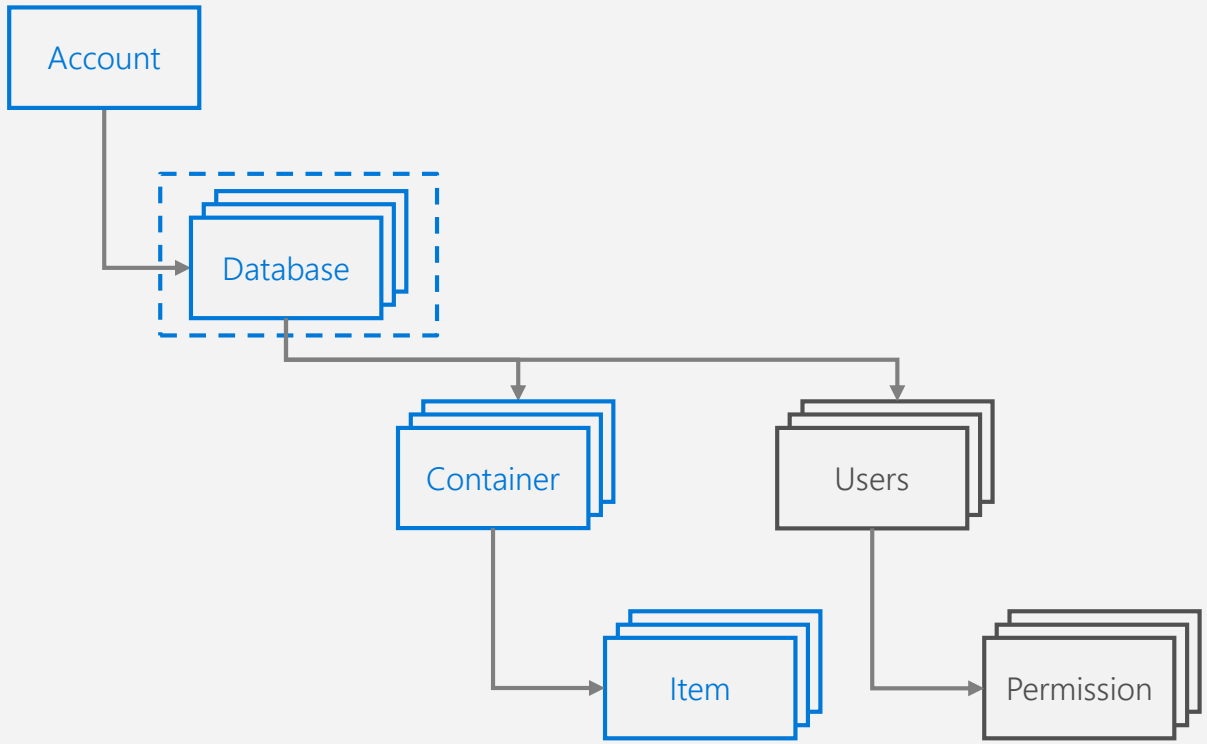
This module will reference partitioning in the context of all Azure Cosmos DB modules and APIs.



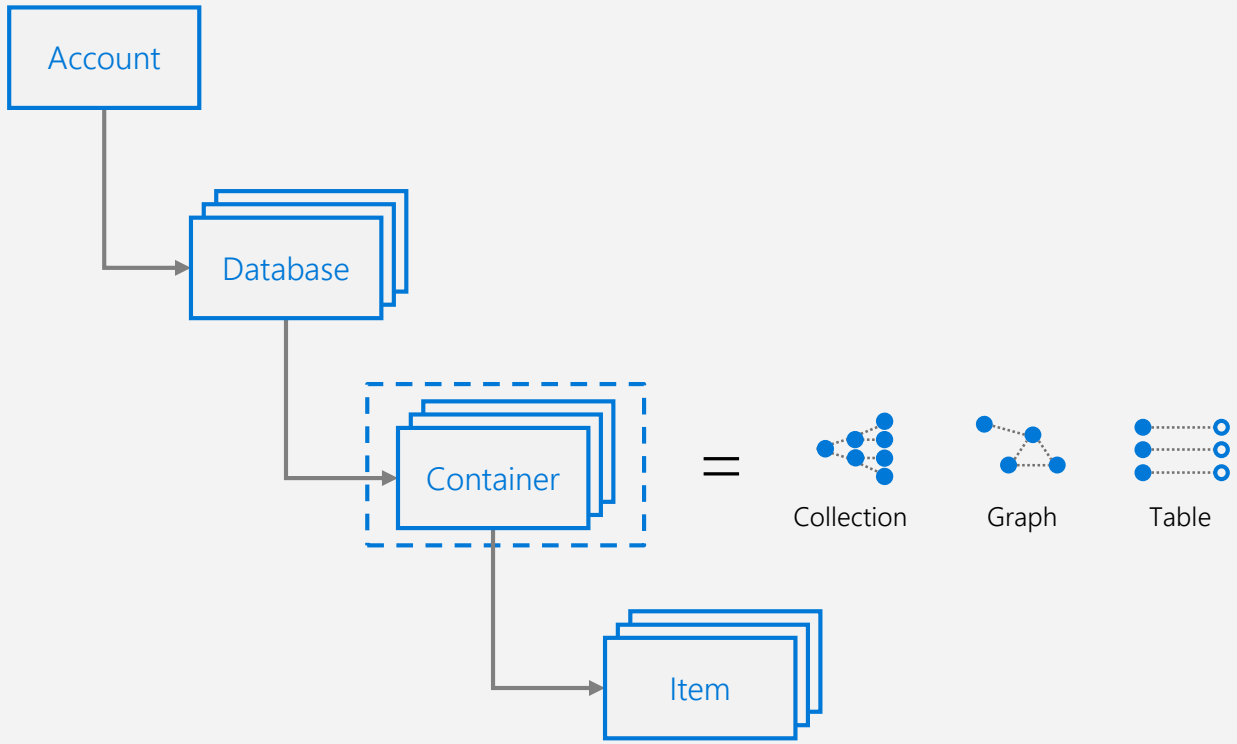
ACCOUNT URI AND CREDENTIALS



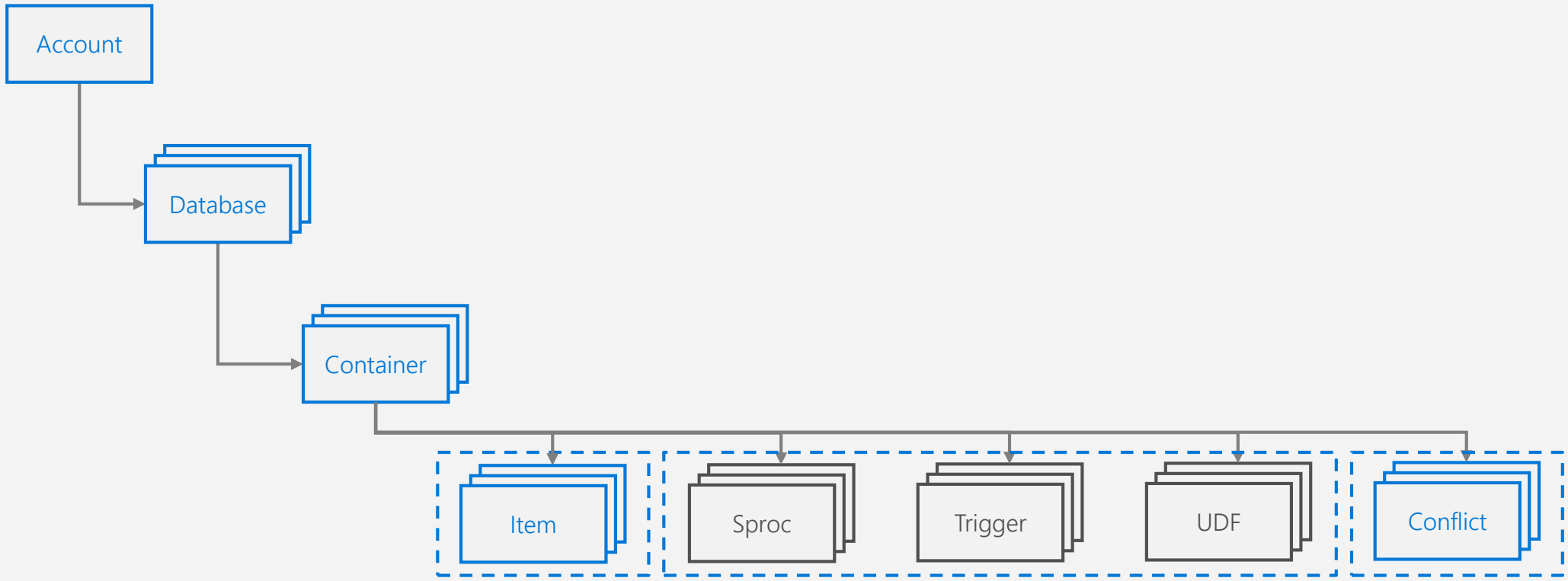
DATABASE REPRESENTATIONS



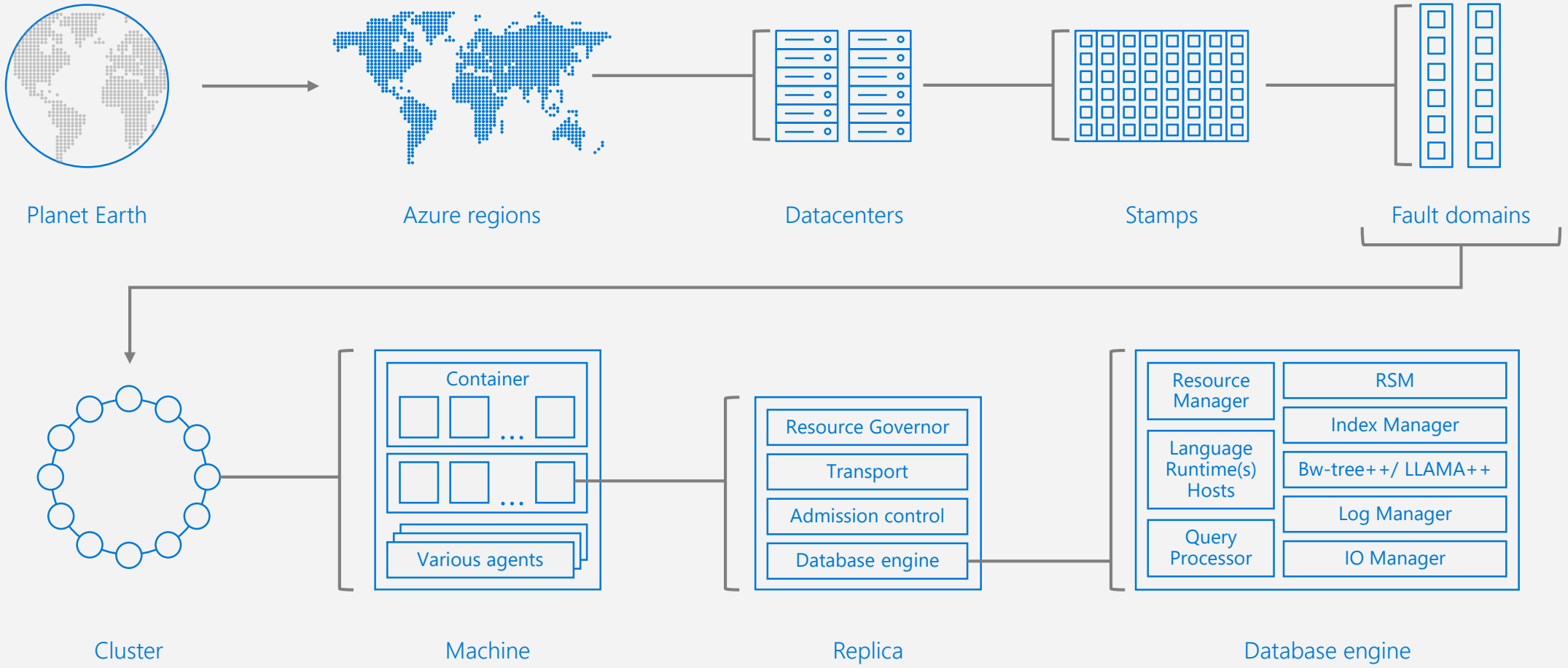
CONTAINER REPRESENTATIONS



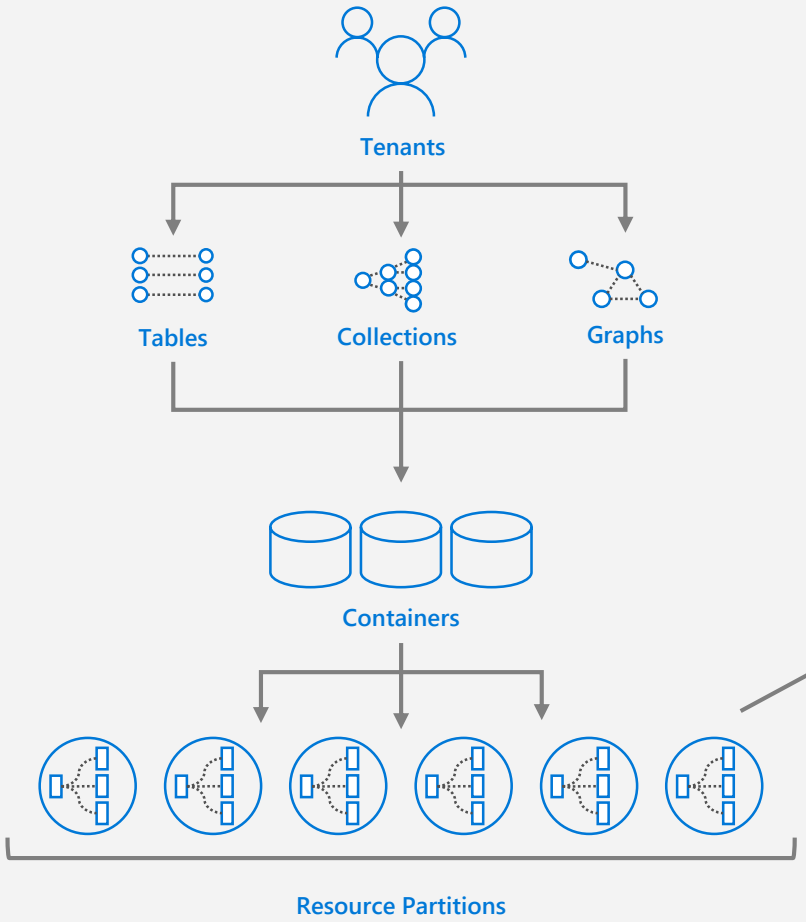
CONTAINER-LEVEL RESOURCES



SYSTEM TOPOLOGY (BEHIND THE SCENES)

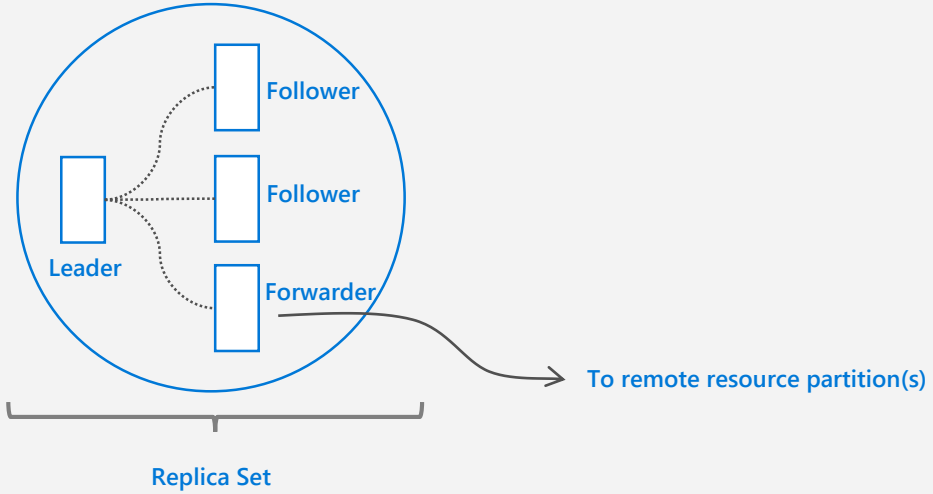


RESOURCE HIERARCHY



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

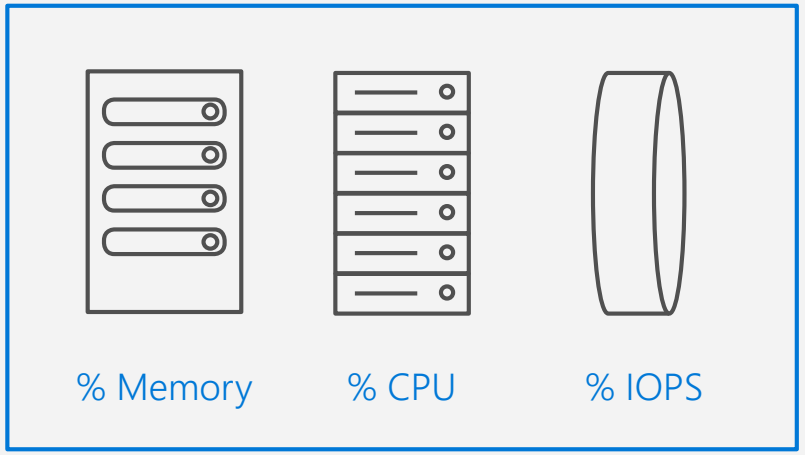
REQUEST UNITS

Request Units (RUs) is a rate-based currency

Abstracts physical resources for performing requests

Key to multi-tenancy, SLAs, and COGS efficiency

Foreground and background activities



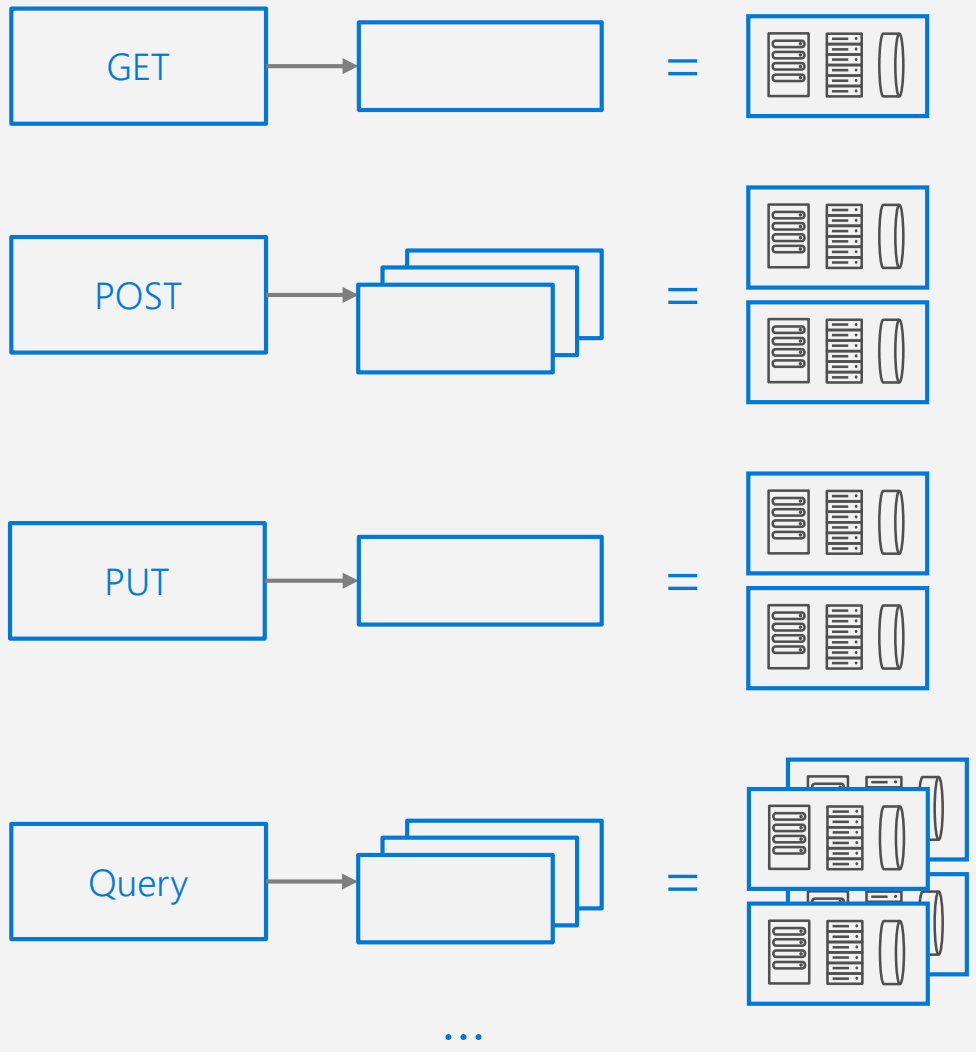
REQUEST UNITS

Normalized across various access methods

1 RU = 1 read of 1 KB document

Each request consumes fixed RUs

Applies to reads, writes, queries, and stored procedure execution



REQUEST UNITS

Normalized across various access methods

1 read of 1 KB document from a single partition

Each request consumes fixed RUs

Applies to reads, writes, queries, and stored procedure execution

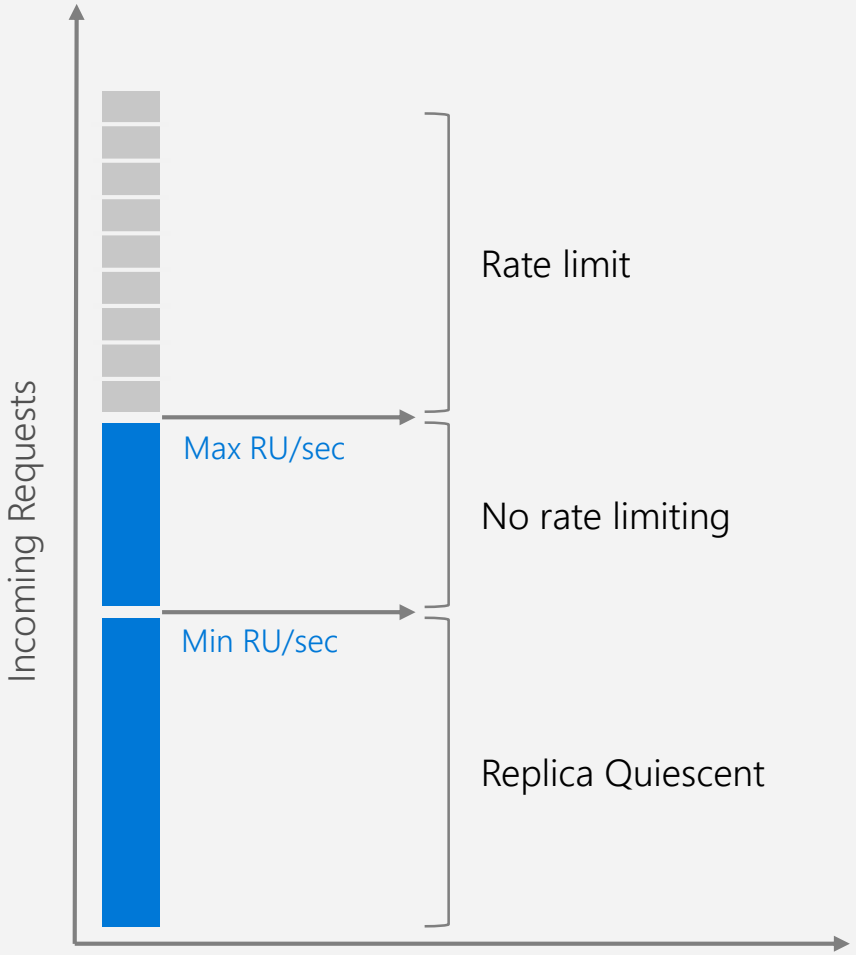
Provisioned in terms of RU/sec

Rate limiting based on amount of throughput provisioned

Can be increased or decreased instantaneously

Metered Hourly

Background processes like TTL expiration, index transformations scheduled when quiescent

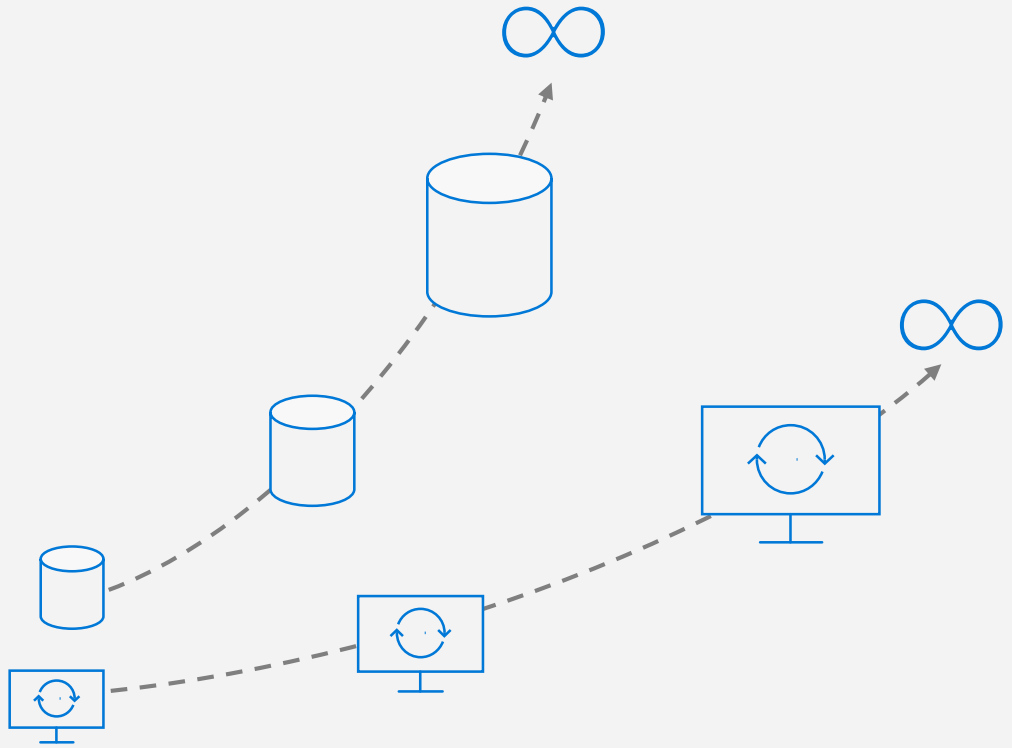


ELASTIC SCALE OUT OF STORAGE AND THROUGHPUT

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.

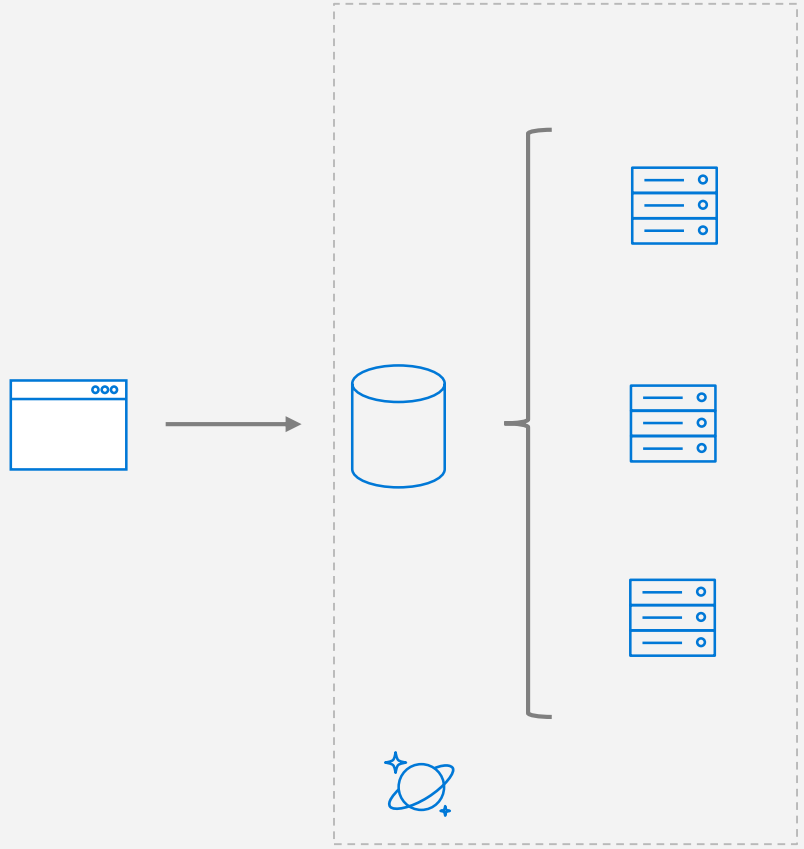
- Elastically scale throughput from 10 to 100s of millions of requests/sec across multiple regions
- Support for requests/sec for different workloads
- Pay only for the throughput and storage you need



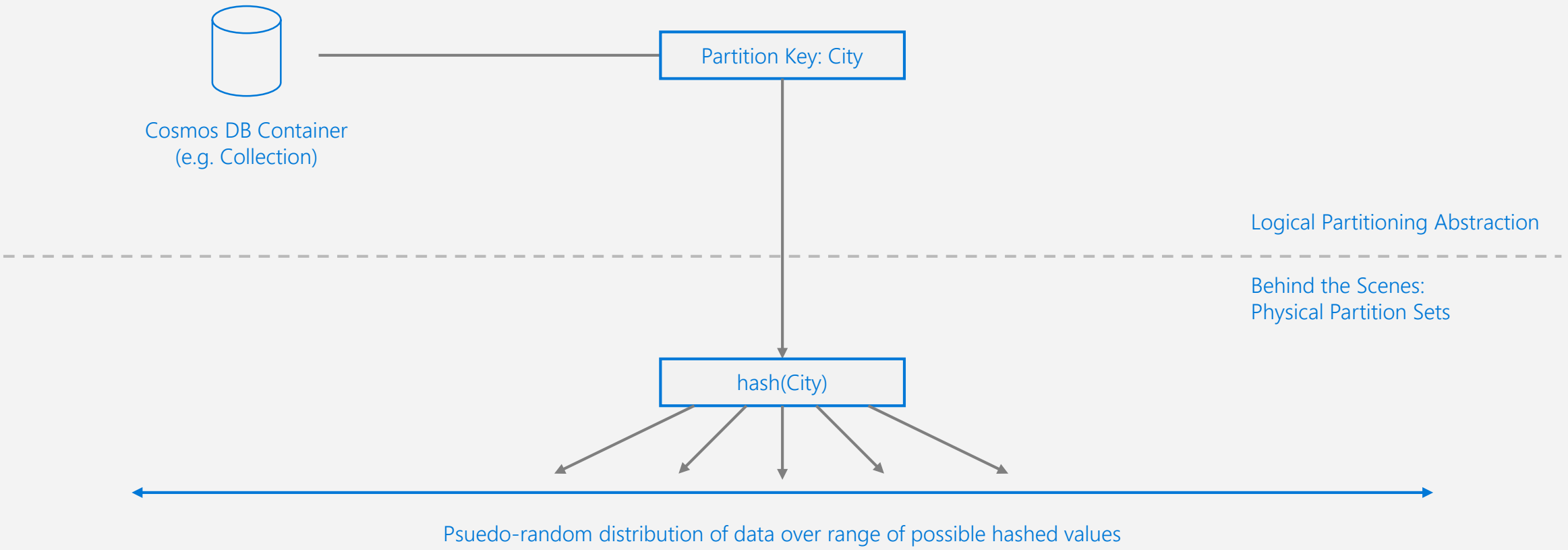
PARTITIONING

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

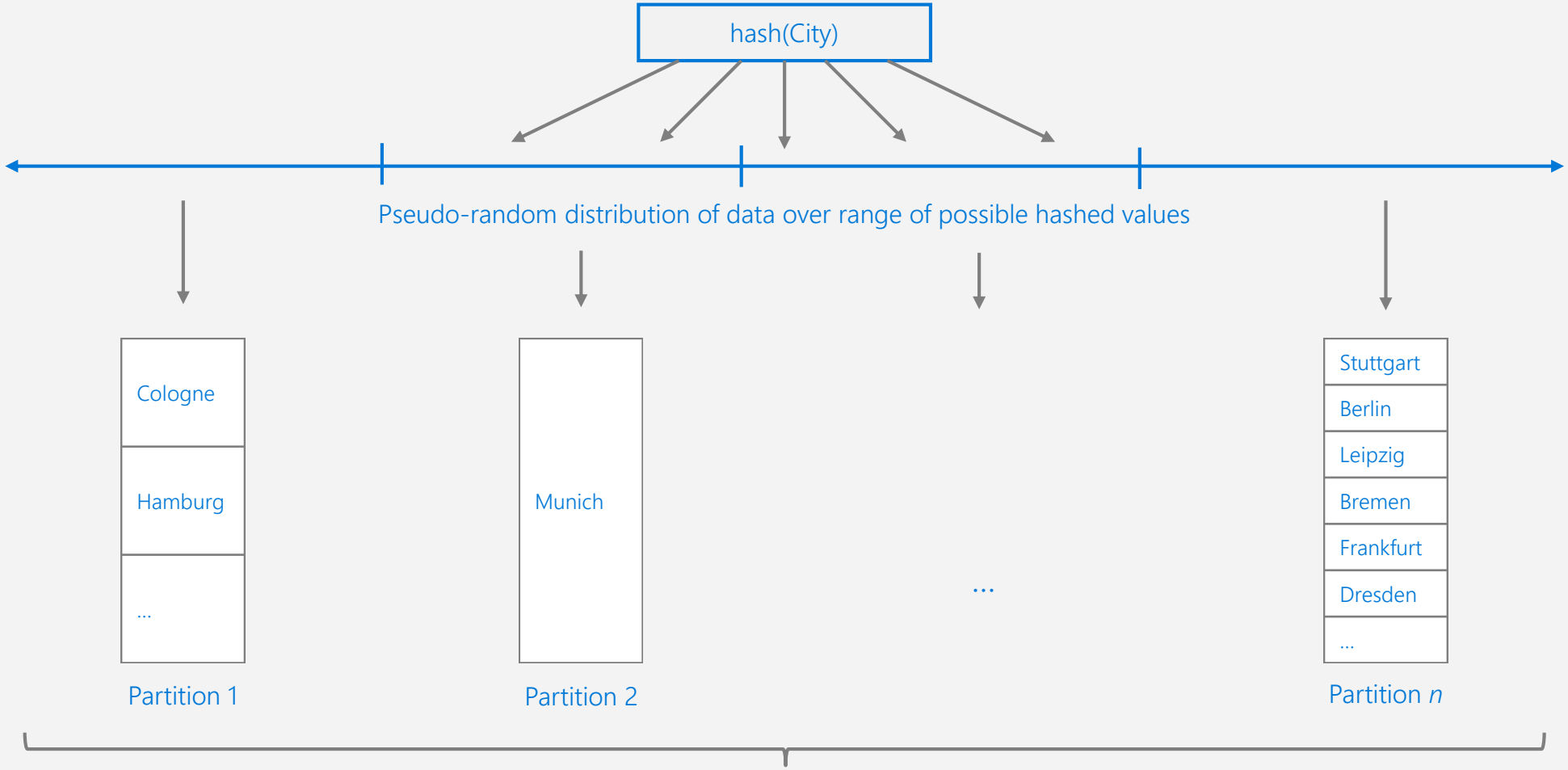
This module will reference partitioning in the context of all Azure Cosmos DB modules and APIs.



PARTITIONS

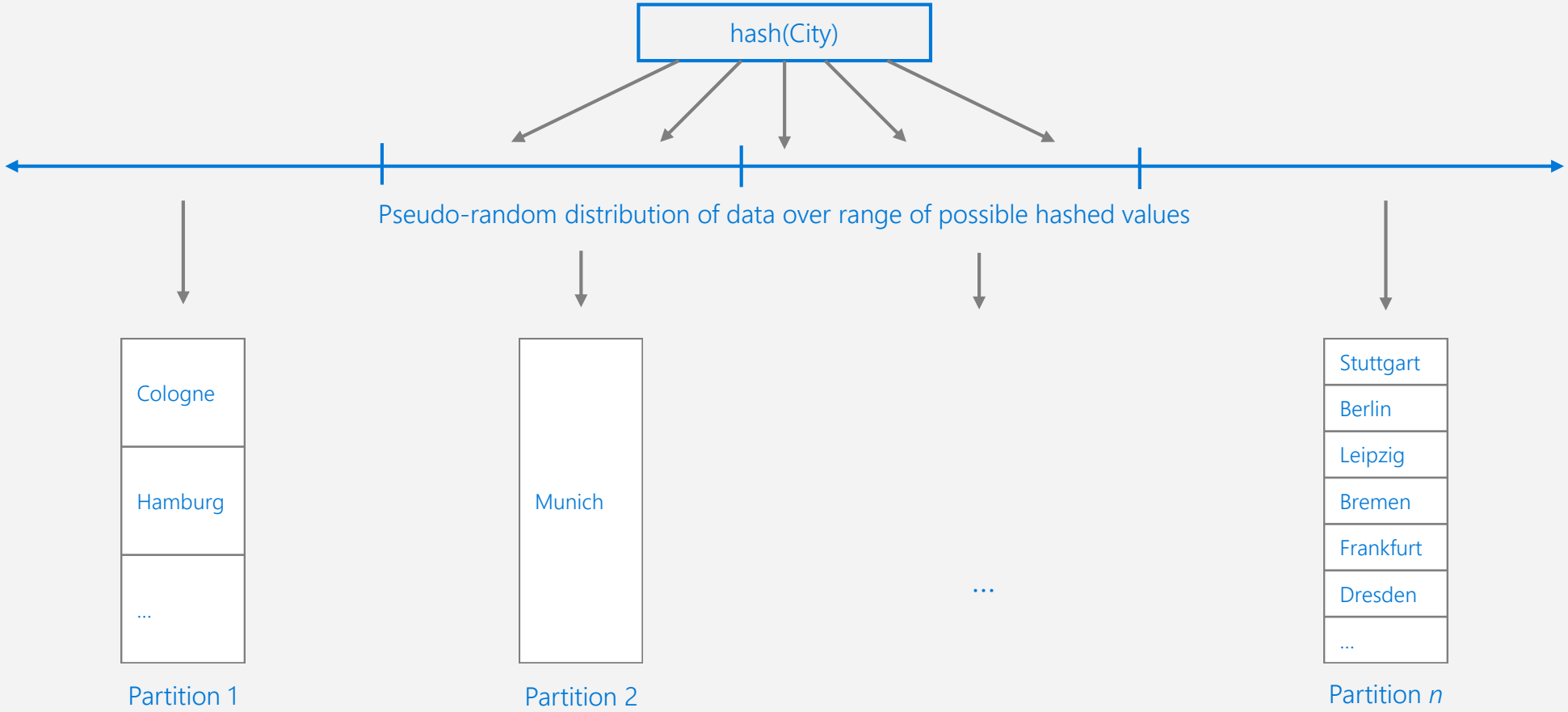

Partition Key: City
Cosmos DB Container
(e.g. Collection)
Logical Partitioning Abstraction
Behind the Scenes:
Physical Partition Sets
hash(City)
Psuedo-random distribution of data over range of possible hashed values

PARTITIONS



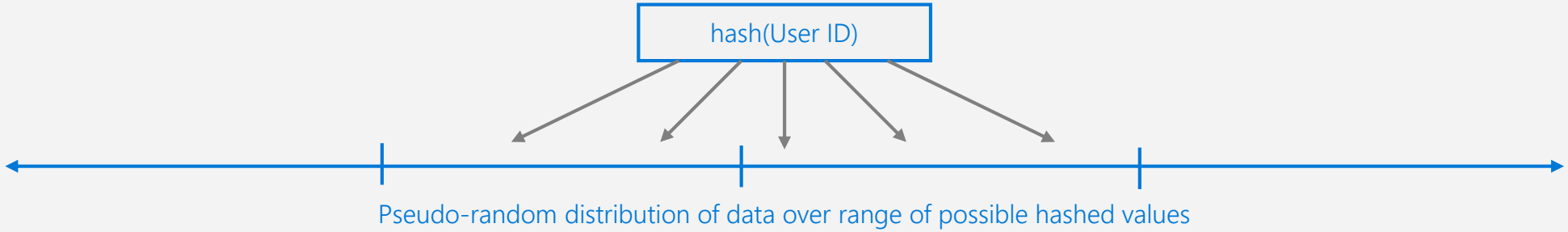
Frugal # of Partitions based on actual storage and throughput needs
(yielding scalability with low total cost of ownership)

PARTITIONS



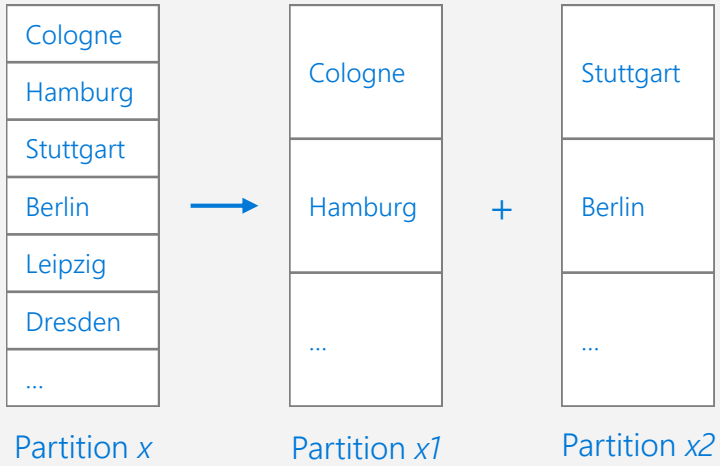
What happens when partitions need to grow?

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.



PARTITIONS

Best Practices: Design Goals for Choosing a Good Partition Key

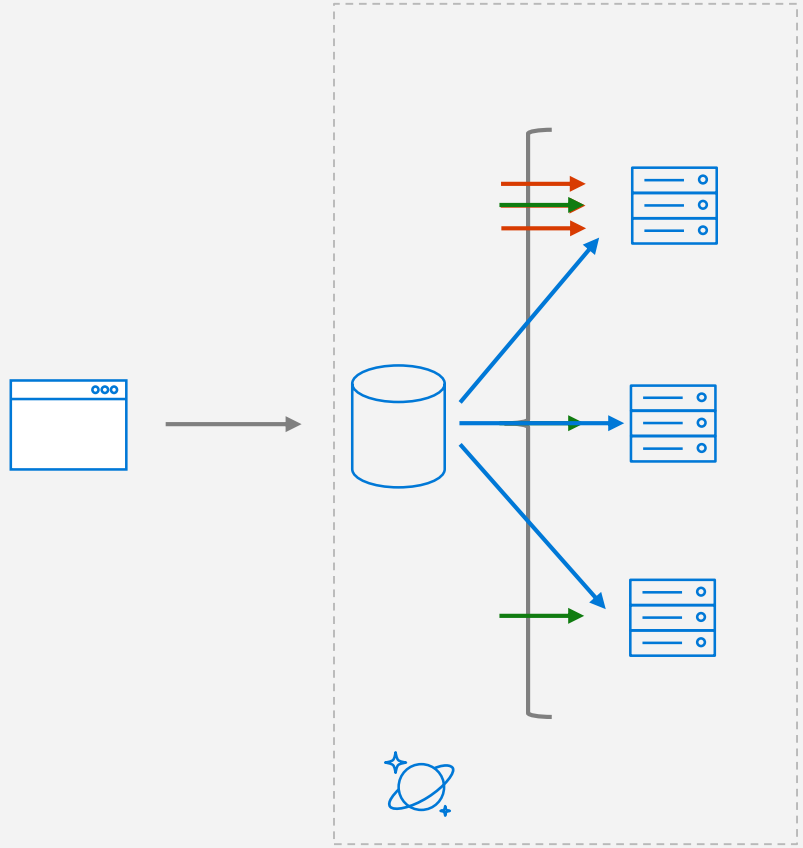
- Distribute the overall request + storage volume
  - Avoid "hot" partition keys
- Partition Key is scope for multi-record transactions and routing queries
  - Queries can be intelligently routed via partition key
  - Omitting partition key on query requires fan-out

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



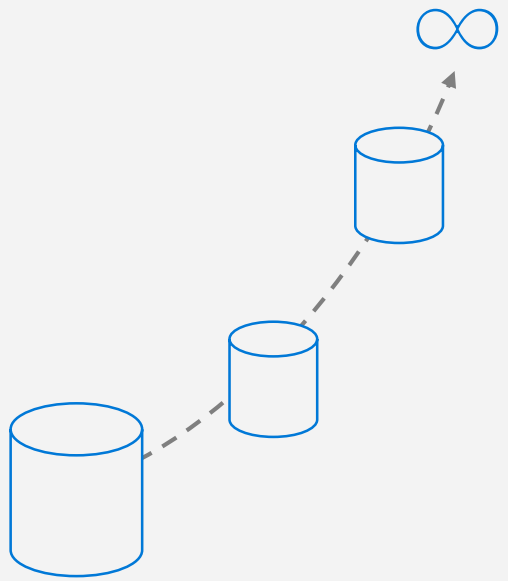
TURNKEY GLOBAL DISTRIBUTION

High Availability

- Automatic and Manual Failover
- Multi-homing API removes need for app redeployment

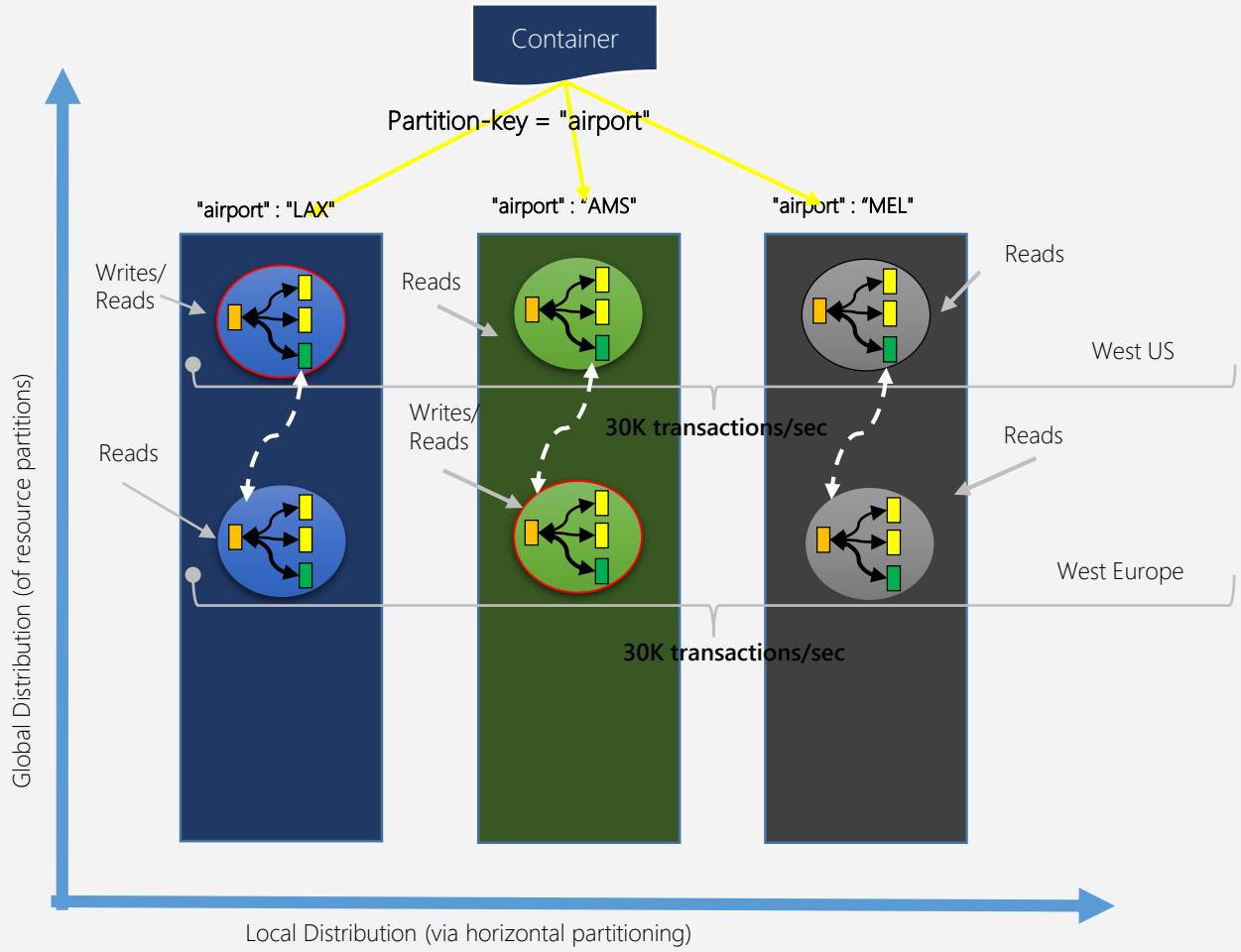
Low Latency (anywhere in the world)

- Packets cannot move fast than the speed of light
- Sending a packet across the world under ideal network conditions takes 100's of milliseconds
- You can cheat the speed of light – using data locality
  - CDN's solved this for static content
  - Azure Cosmos DB solves this for dynamic content

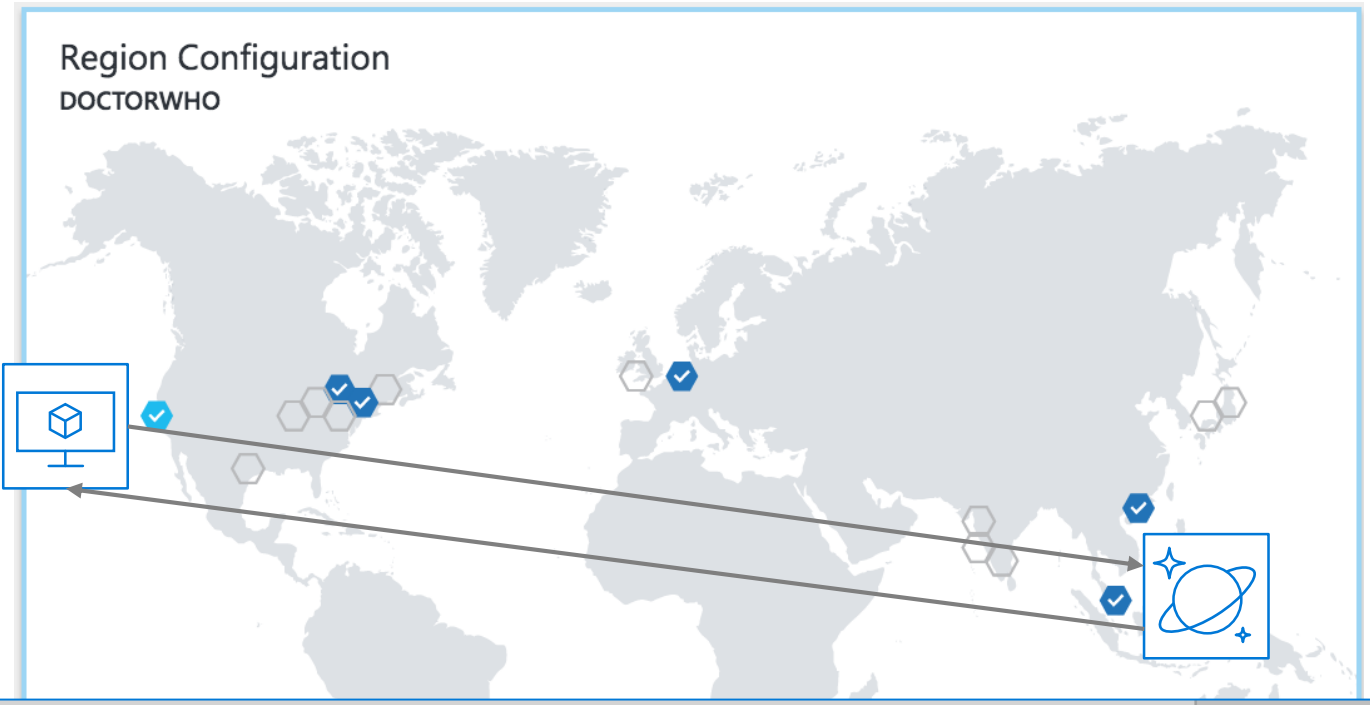


TURNKEY GLOBAL DISTRIBUTION

- Automatic and transparent replication worldwide
- Each partition hosts a replica set per region
- Customers can test end to end application availability by programmatically simulating failovers
- All regions are hidden behind a single global URI with multi-homing capabilities
- Customers can dynamically add / remove additional regions at any time



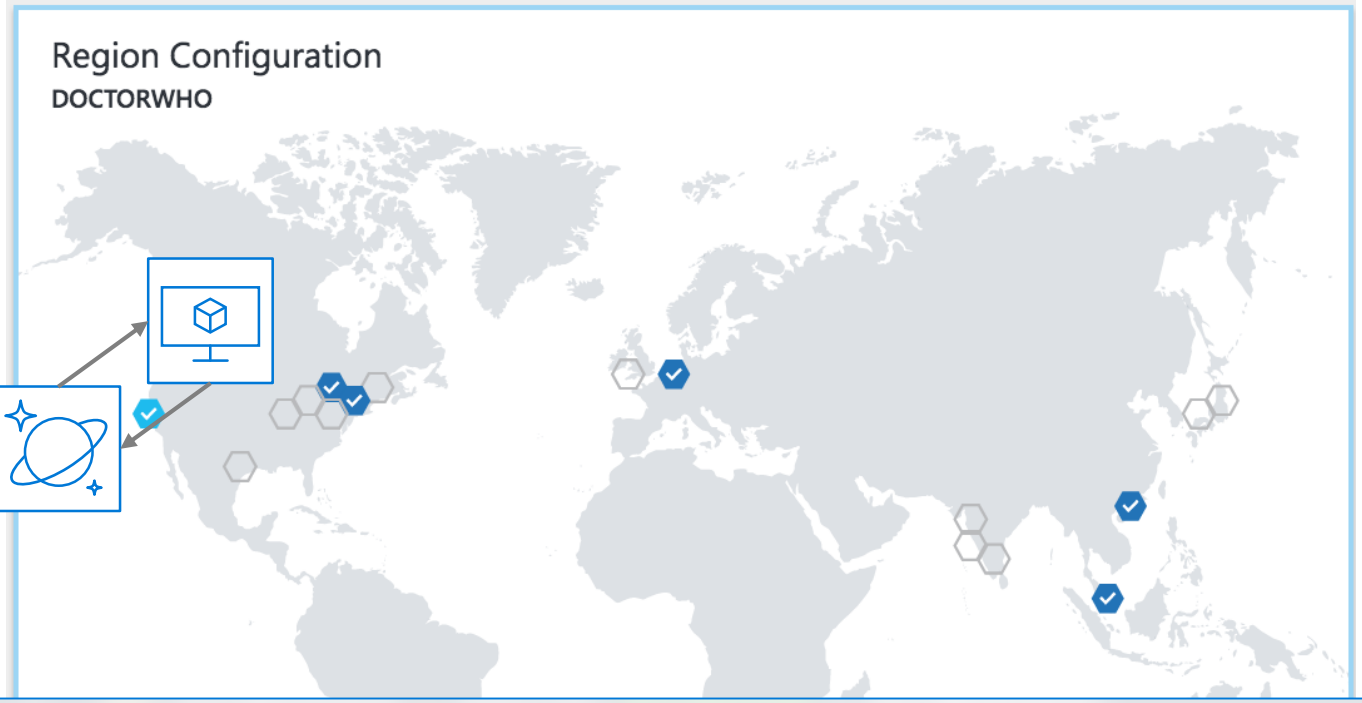
REPLICATING DATA GLOBALLY



...rkspace/docdb/docdb — amypond@blink: ~/docdb — ssh amypond@104.42.108.173 -p 22

```
amypond@blink:~/docdb$ node testQ2.js
210.788804 milliseconds, 2 RUs, ActivityId: 9025eac6-eb74-4a07-94cf-f2383caffbb3
178.825773 milliseconds, 2 RUs, ActivityId: ea53b736-b629-4290-9cdf-cf80c6139461
178.839173 milliseconds, 2 RUs, ActivityId: f143c992-ba67-4c7b-b7b8-0b5db8df4dbf
178.564573 milliseconds, 2 RUs, ActivityId: 1a8d7b5b-42c5-4c39-9160-d1e5ab2200d0
179.229073 milliseconds, 2 RUs, ActivityId: 483b85de-74e0-4f48-9206-70ac1268c60e
178.653772 milliseconds, 2 RUs, ActivityId: 50fbfe91-f41e-4f14-8a15-0b344c894727
178.464572 milliseconds, 2 RUs, ActivityId: cac6446a-79d4-4886-81d8-1dda835daa72
180.708475 milliseconds, 2 RUs, ActivityId: d40af8e4-582b-4479-bb9c-e3582eac6774
```

REPLICATING DATA GLOBALLY




```
amypond@blink:~/docdb$ node testQ2.js
12.736112 milliseconds, 2 RUs, ActivityId: dd3c17b9-1b76-445a-8e27-29b7486bd7e4
4.947605 milliseconds, 2 RUs, ActivityId: e2f4c899-9fb1-4f76-a4ab-e5718fac5742
5.044005 milliseconds, 2 RUs, ActivityId: 0fc5d216-78a0-4d92-a3d0-63efd9dd6552
5.351205 milliseconds, 2 RUs, ActivityId: 861155f0-81ba-4c8a-9933-e50d8708cc21
4.553505 milliseconds, 2 RUs, ActivityId: 3db9641f-70f1-4ef1-84bb-809280bbe1a5
5.427506 milliseconds, 2 RUs, ActivityId: 10d1b2e5-e795-4c77-8655-815f410ba11e
5.900106 milliseconds, 2 RUs, ActivityId: bda1df86-c5ad-45c5-bcfb-93d417c54751
4.895405 milliseconds, 2 RUs, ActivityId: f206d58d-64a2-47f2-9653-145eaf47ff97
5.244306 milliseconds, 2 RUs, ActivityId: 3aa7e177-b1a9-413d-8023-55f5054d1b74
```

FIVE WELL-DEFINED CONSISTENCY MODELS

CHOOSE THE BEST CONSISTENCY MODEL FOR YOUR APP

Five well-defined, consistency models

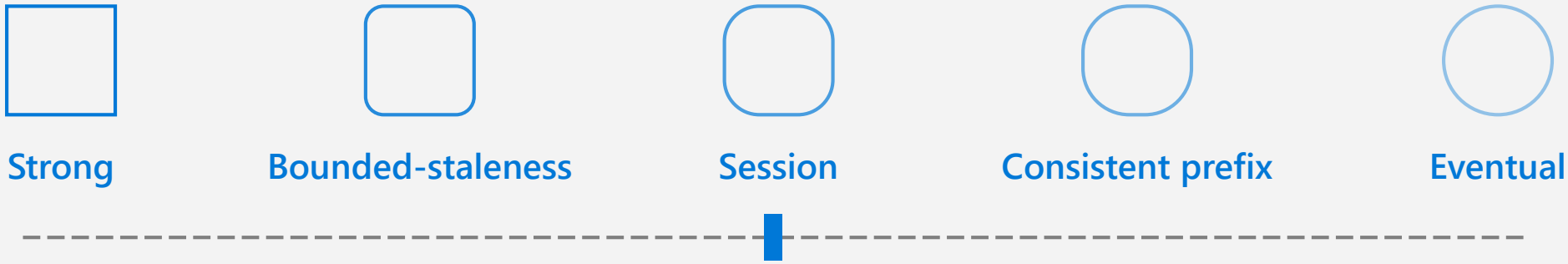
Overridable on a per-request basis

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.

CLEAR TRADEOFFS

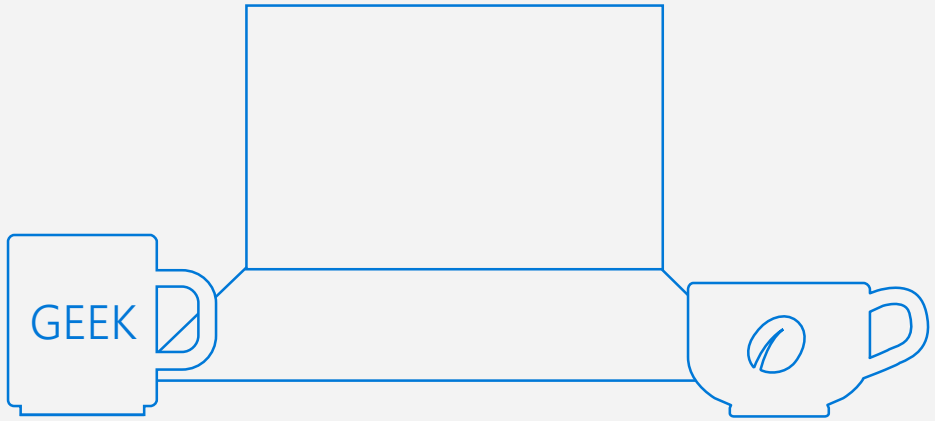
- **Latency**
- **Availability**
- **Throughput**



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.

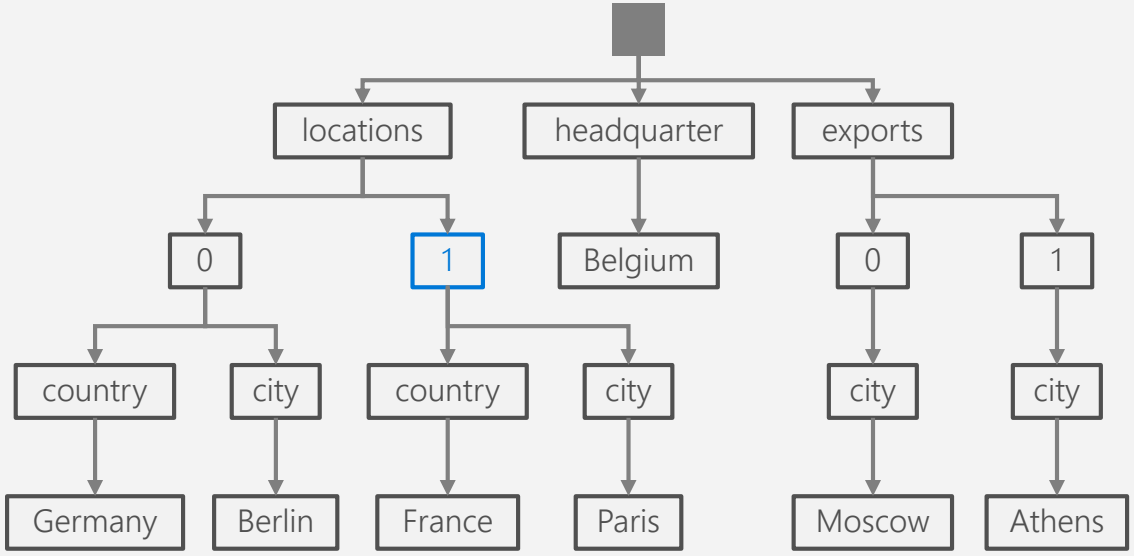
- Automatic index management
- Synchronous auto-indexing
- No schemas or secondary indices needed
- Works across every data model



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

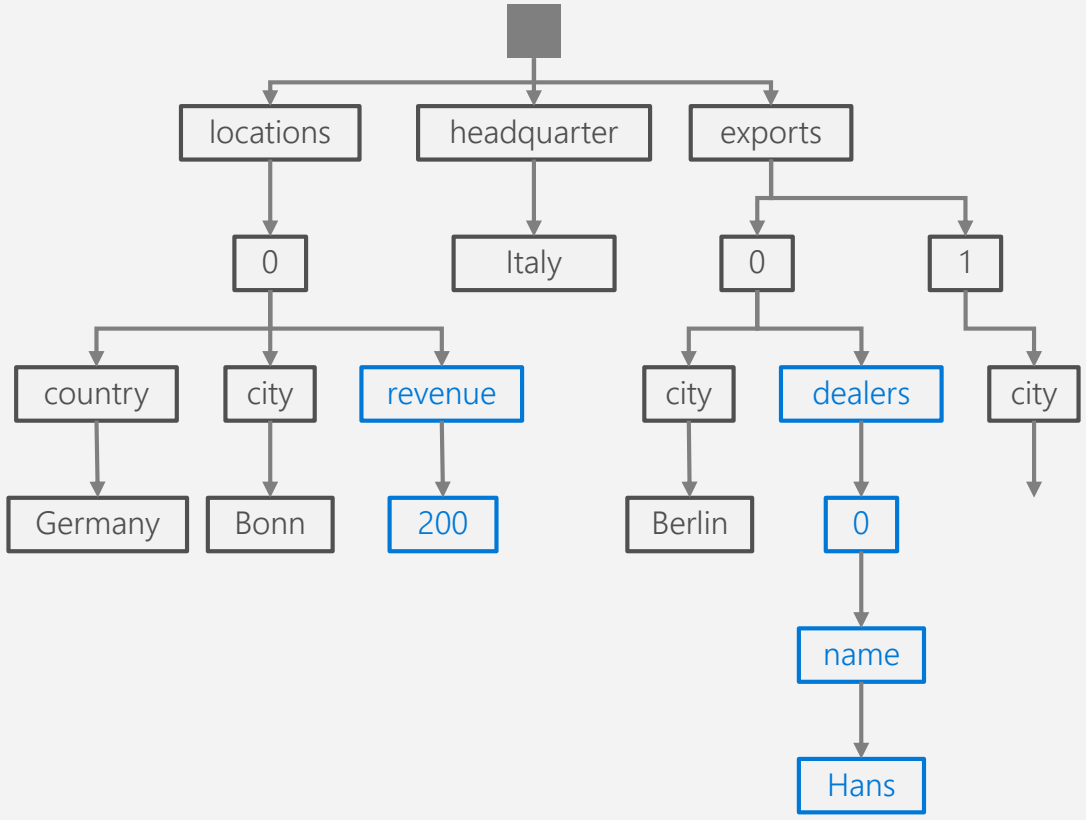
INDEXING JSON DOCUMENTS

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

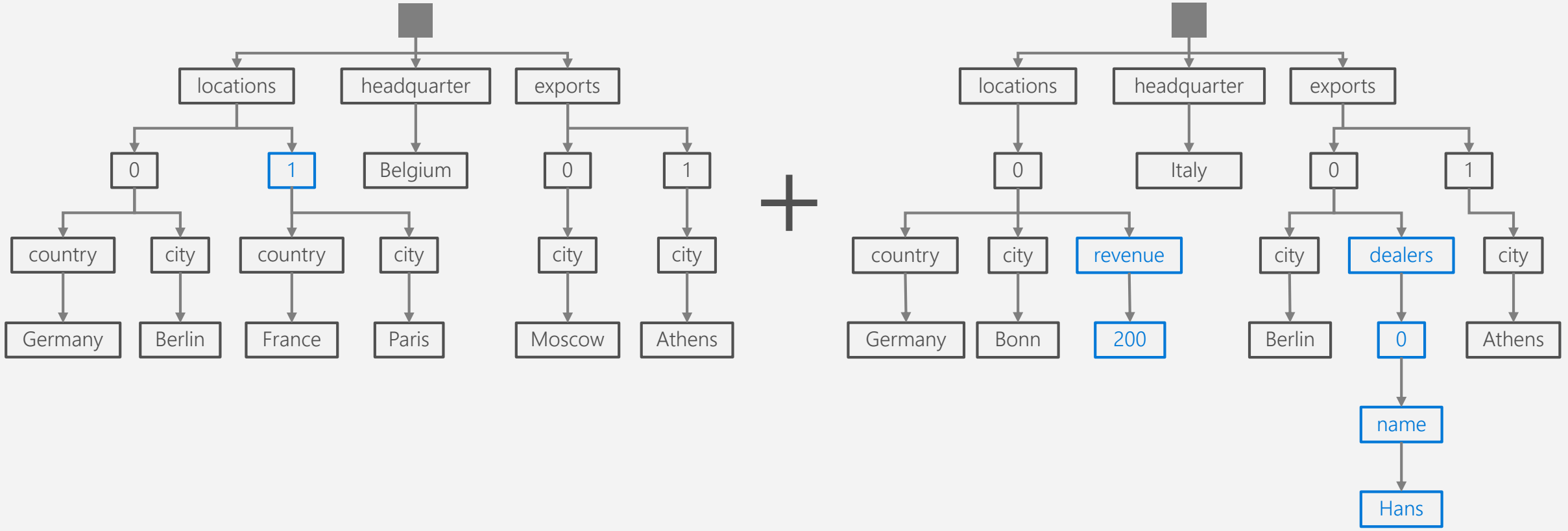


INDEXING JSON DOCUMENTS

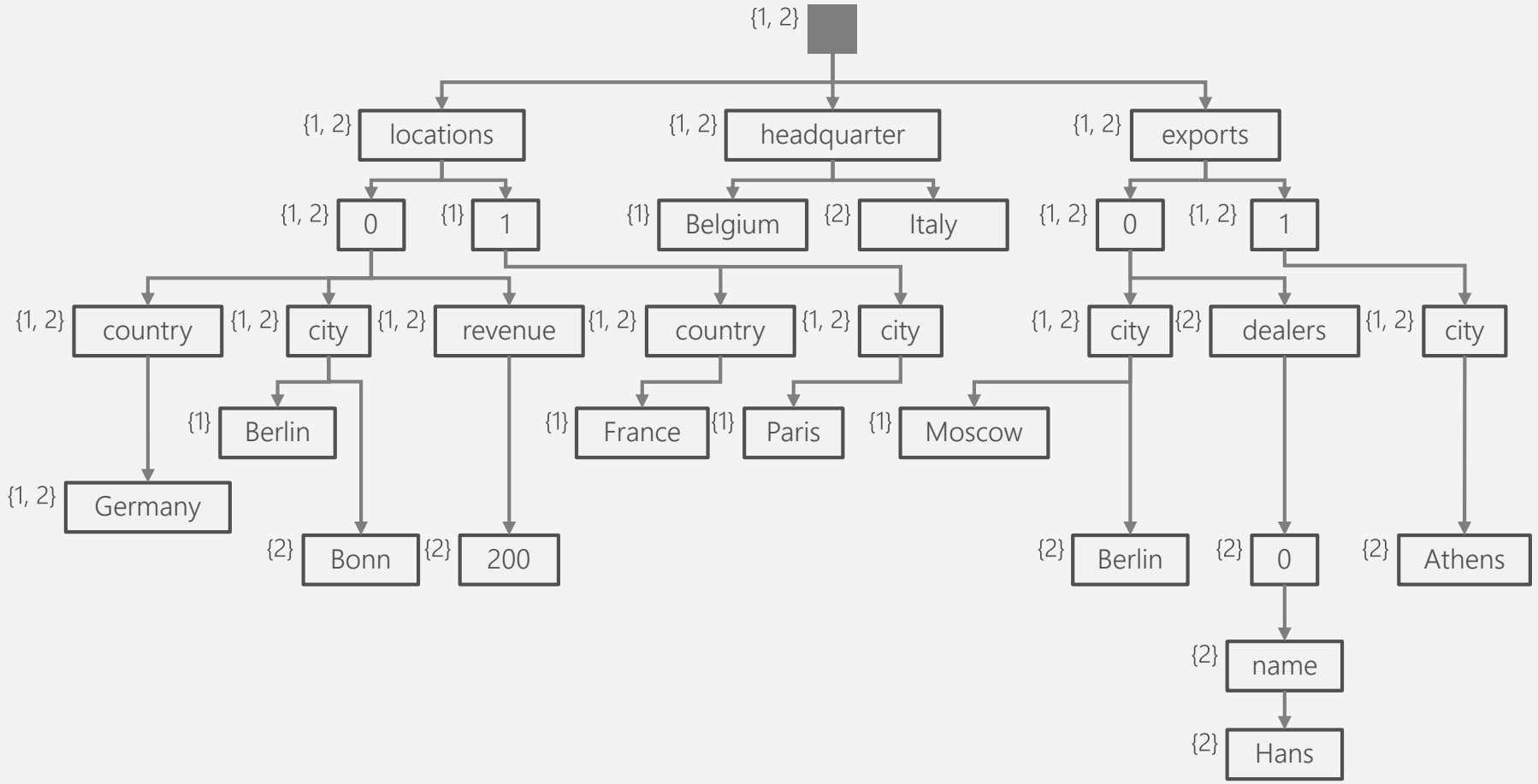
```
{
    "locations": [
        {
            "country": "Germany",
            "city": "Bonn",
            "revenue": 200
        }
    ],
    "headquarter": "Italy",
    "exports": [
        {
            "city": "Berlin",
            "dealers": [
                { "name": "Hans" }
            ]
        },
        { "city": "Athens" }
    ]
}
```



INDEXING JSON DOCUMENTS



INVERTED INDEX



INDEX POLICIES

CUSTOM INDEXING POLICIES

Though all Azure Cosmos DB data is indexed by default, you can specify a custom indexing policy for your collections. Custom indexing policies allow you to design and customize the shape of your index while maintaining schema flexibility.

- Define trade-offs between storage, write and query performance, and query consistency
- Include or exclude documents and paths to and from the index
- Configure various index types

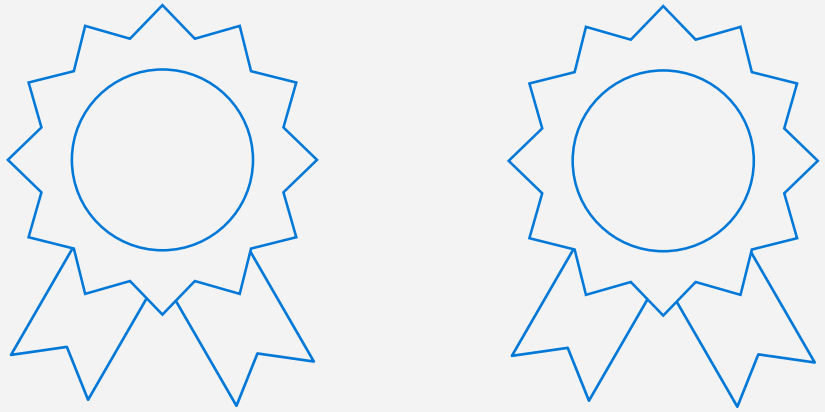
```
{
    "automatic": true,
    "indexingMode": "Consistent",
    "includedPaths": [{
        "path": "/*",
        "indexes": [{
            "kind": "Hash",
            "dataType": "String",
            "precision": -1
        }, {
            "kind": "Range",
            "dataType": "Number",
            "precision": -1
        }, {
            "kind": "Spatial",
            "dataType": "Point"
        }]
    }],
    "excludedPaths": [{
        "path": "/nonIndexedContent/*"
    }]
}
```

SHORT-LIFETIME DATA

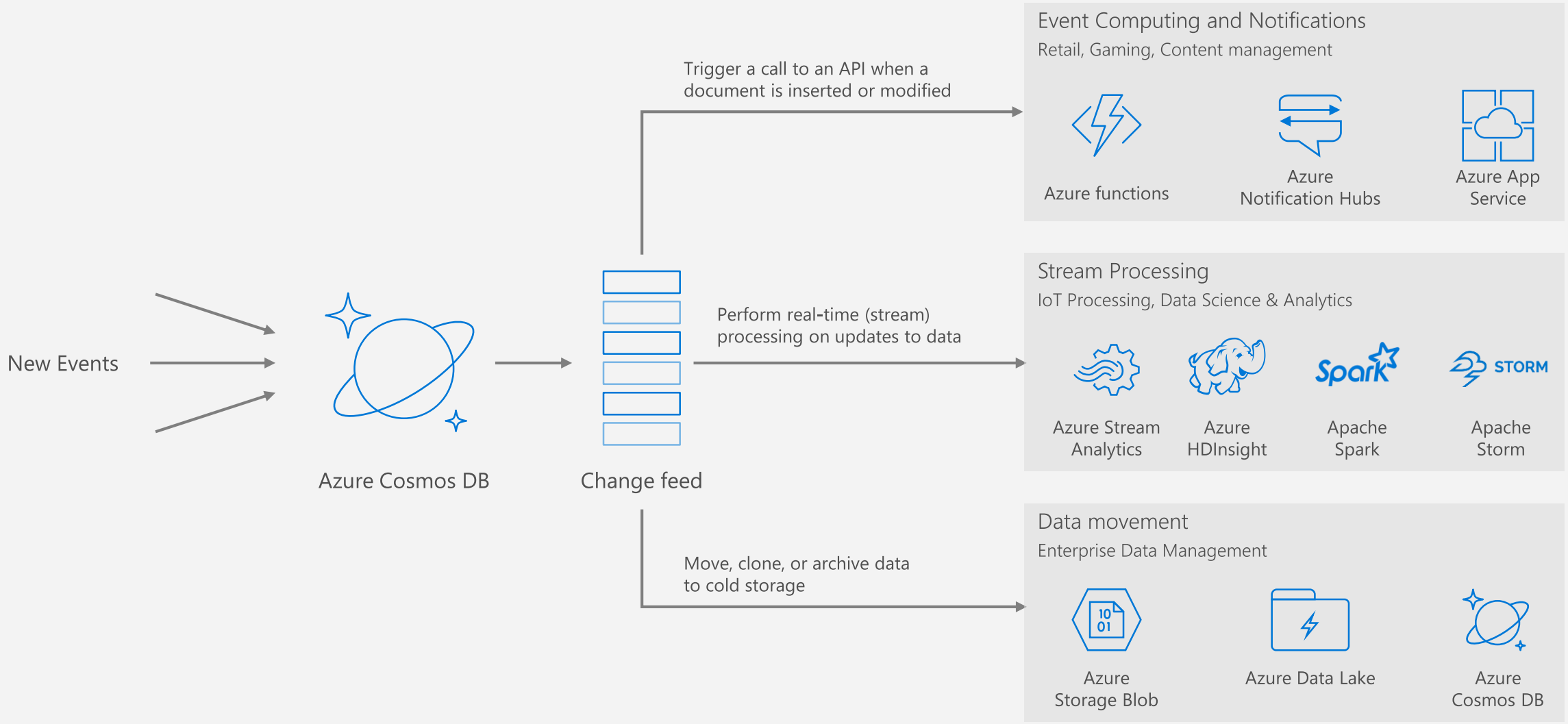
Some data produced by applications are only useful for a finite period of time:

- Machine-generated event data
- Application log data
- User session information

It is important that the database system systematically purges this data at pre-configured intervals.



CHANGE FEED SCENARIOS



Many thanks to all volunteers!

Rafael Dabrowski

Volker Bachmann

Alexander Klein

Tobias Blödt

Ben Kettner

Dirk Hondong

Christian Gräfe

Cornelia Matthesius

Gabi Münster

Dominik Petri

Kai Michael Poppe

Kai Gerlach

Björn Peters

Christa Kurschat

Henrik Schütze

Klaus Betzing

Tanja Salwiczek

Nadine Witthöft

SQLSaturday #772 - Munich

27.10.2018

http://www.sqlsaturday.com/772

SQLSATURDAY

PASS

PASS DEUTSCHLAND e.V.

Microsoft Data Platform Community

PASS Deutschland e.V.

For further information about future events, visit our PASS Deutschland e.V. booth in the exhibitor area.