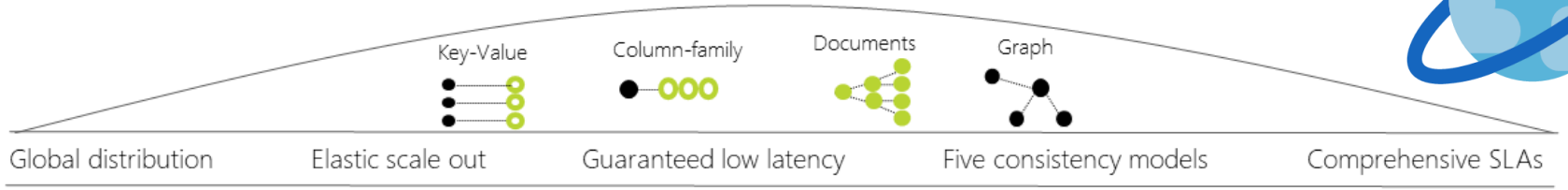
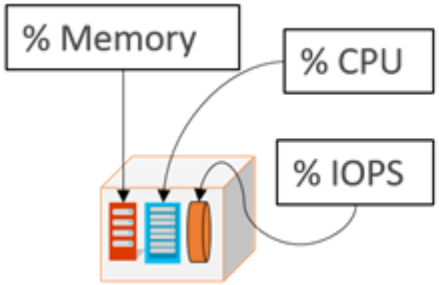
Building scalable applications in Cosmos DB

Ken Seier

Azure Cosmos DB

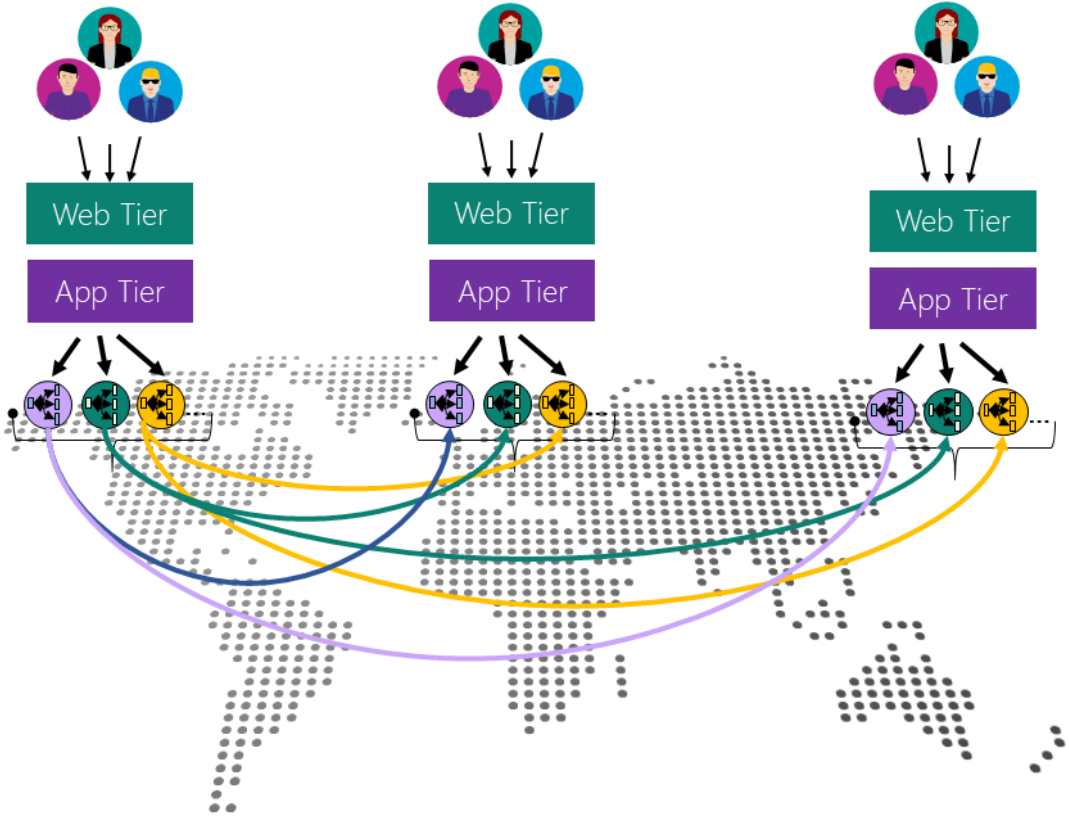


Request units (RU)



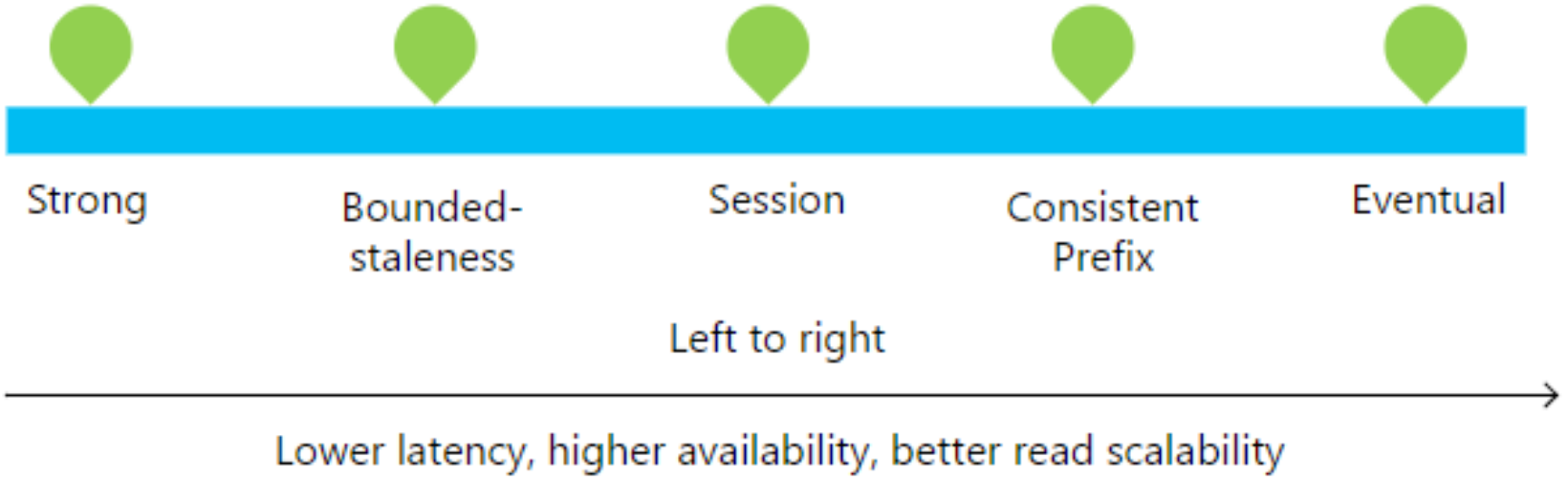
Your results may vary

Global replication



BLUEMETAL
A division of Insight

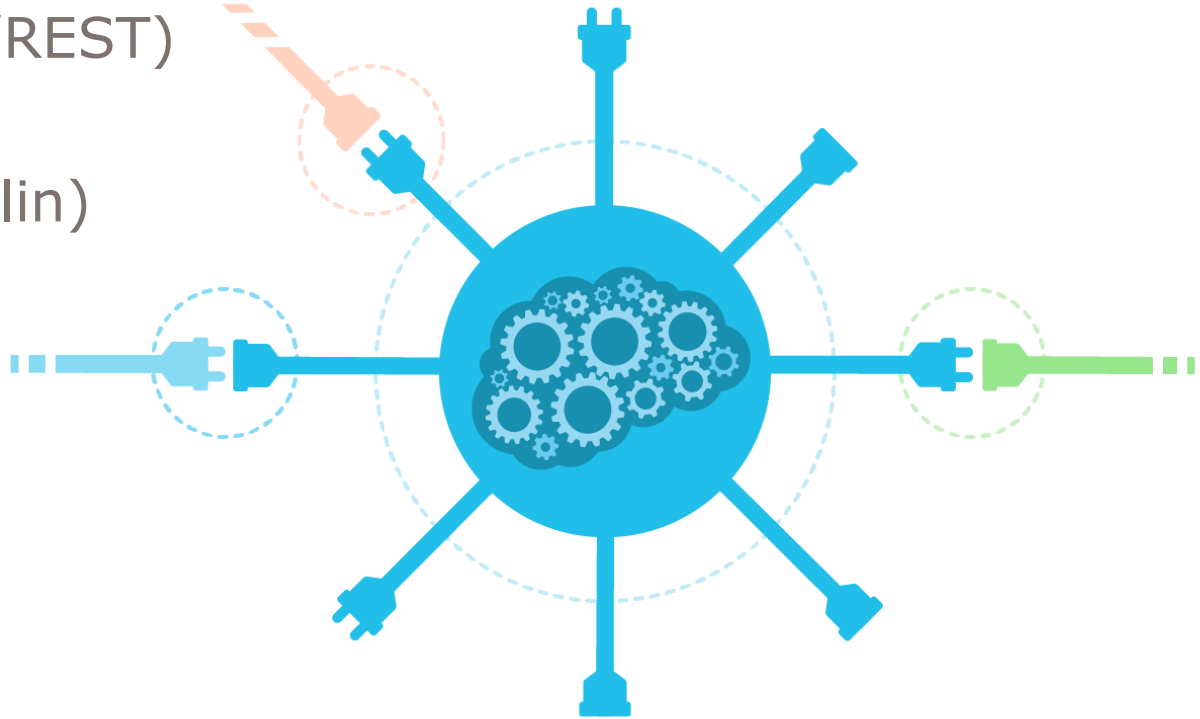
Consistency



Consistency is set at the DB and query levels

APIs

- SQL (was Document/REST)
- MongoDB
- Graph (Apache Gremlin)
- Table (Azure Table)
- Cassandra

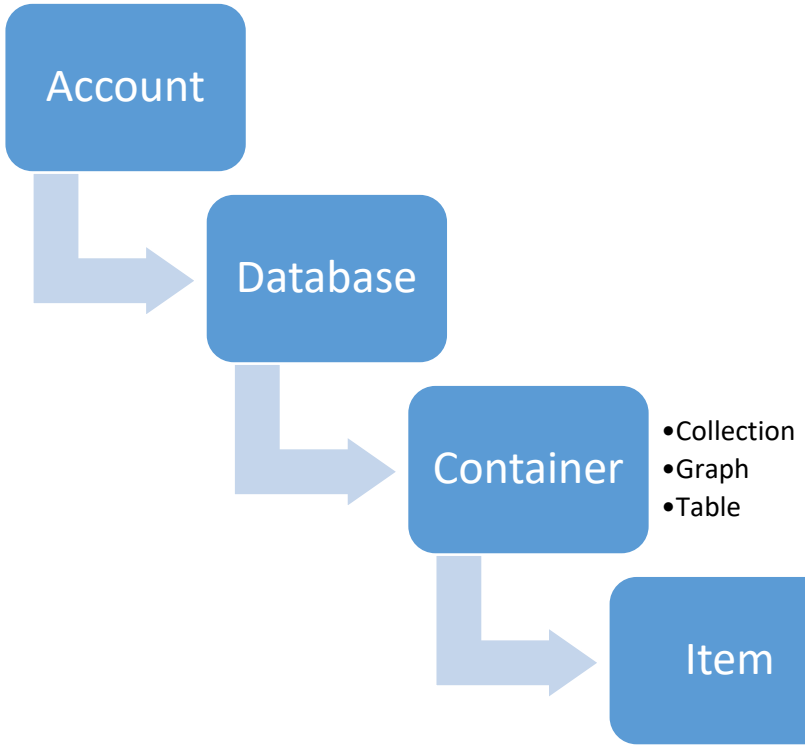


BLUEMETAL
A division of Insight

SDKs

Fall back to REST

Cosmos hierarchy



Also:

- Sproc
- Function
- Trigger


SQL API

SQL API

- SQL-like language over document store
- Formerly REST API, Formerly Document DB
- SELECT, FROM, WHERE
- COUNT, SUM, MIN, MAX, AVERAGE

- SQL API does not behave like SQL

BLUEMETAL
A division of Insight

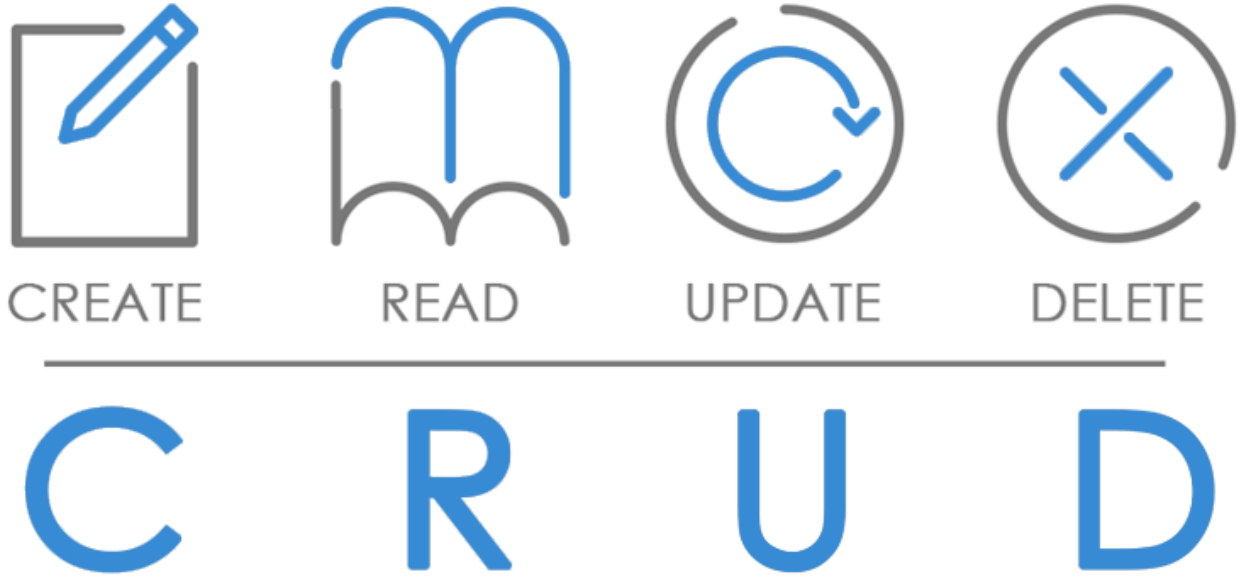
Json document

```
{
 "Name": "Yogurt Depot",
 "Id":  1,
 "Revenue": 2000,
 "Cost": 100,
 "Category": [ "dessert", "food", "yogurt" ],
 "Visits": [
     {
         "day": "Mon",
         "visit_count": 300
     },
     {
         "day": "Tue",
         "visit_count": 700
     }
 ],
```

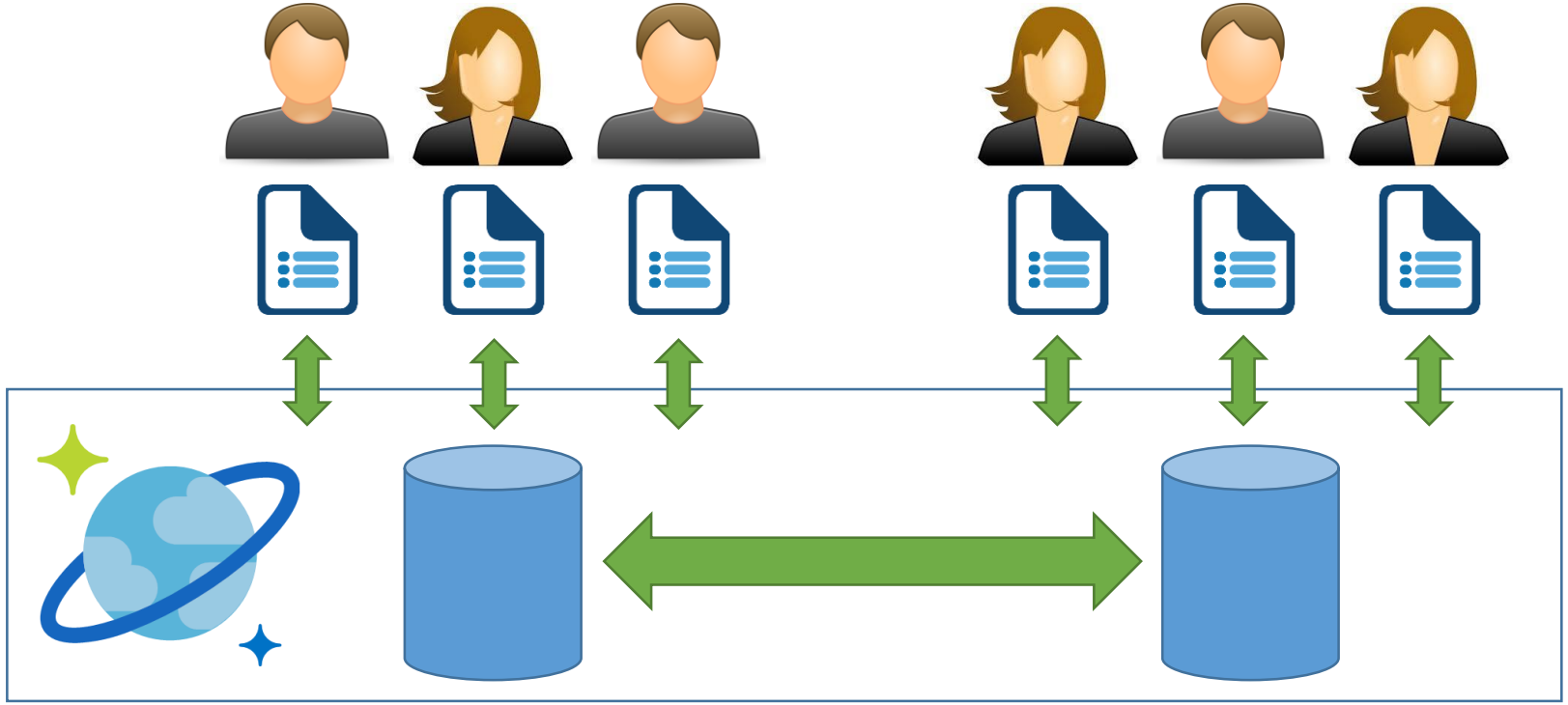
- Hierarchical text format
- Similar to XML except:
  - Lighter weight
  - More human-readable
  - Less annotatable

BLUEMETAL
A division of Insight

Built around CRUD

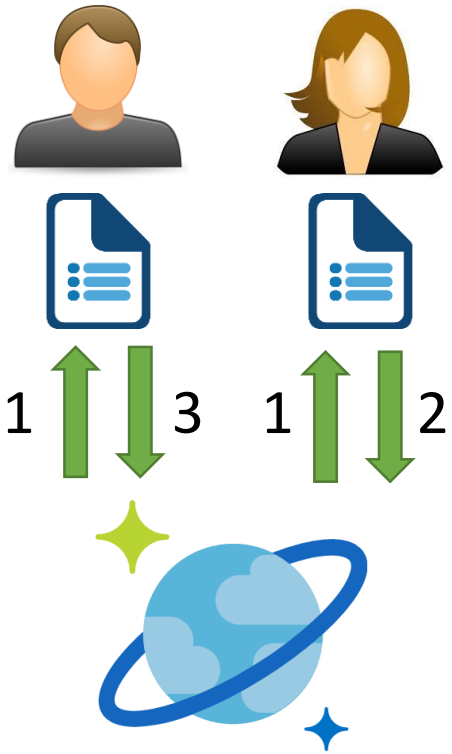


Consistency & concurrency

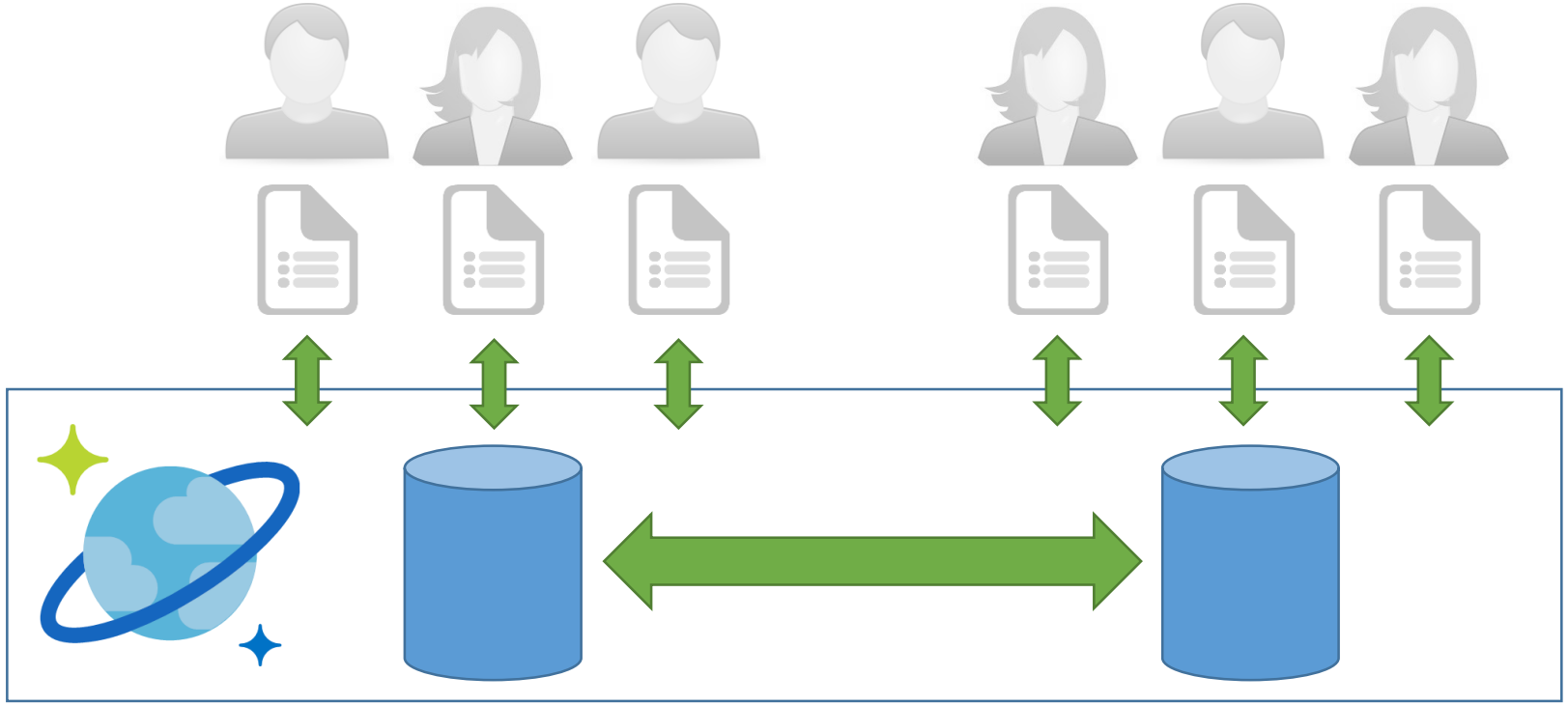


Concurrency

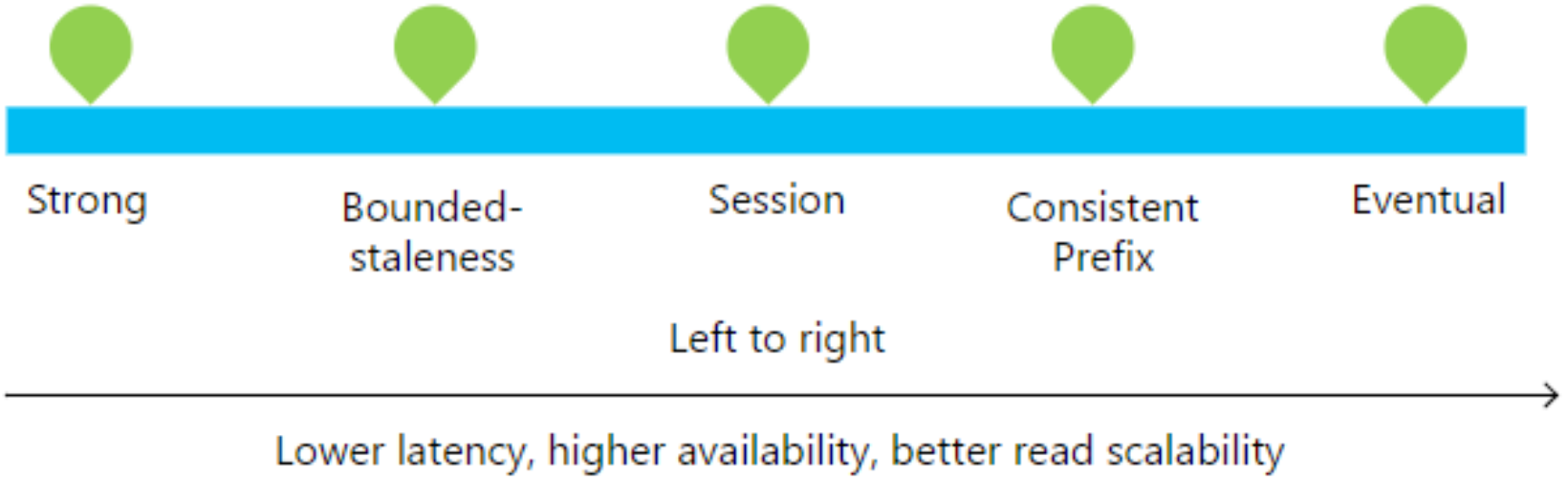
- Optimistic concurrency
- Uses HTTP Etag implemented as collision resistant hash
- HTTP status code result
  - 412 – Precondition failure



Consistency



Consistency levels



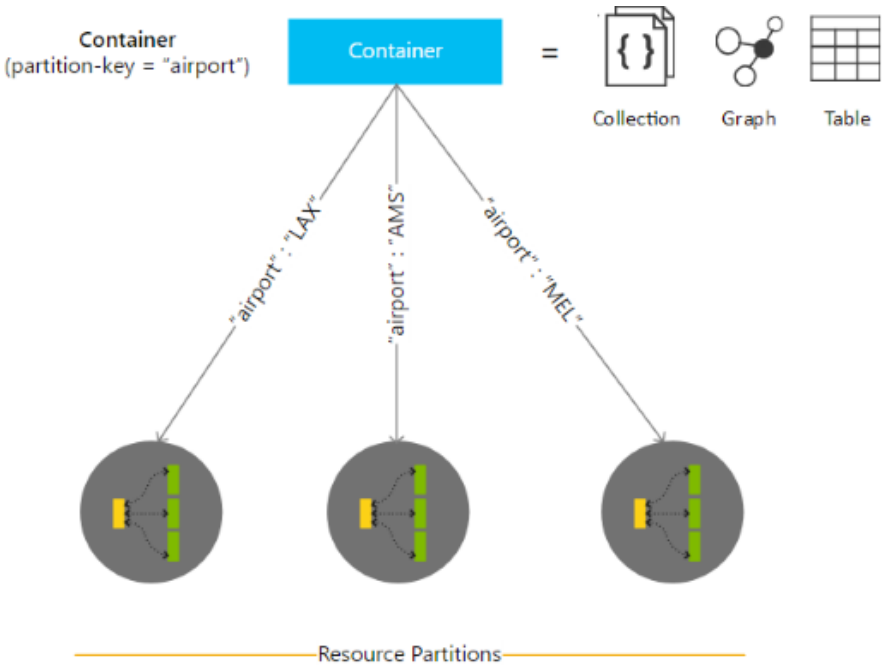
Consistency is set at the DB and query levels

Transactions

- Sprocs, triggers and UDFs
- Written in JavaScript
- Procedural
- ACID - Atomicity, Consistency, Isolation and Durability
- Executed server-side

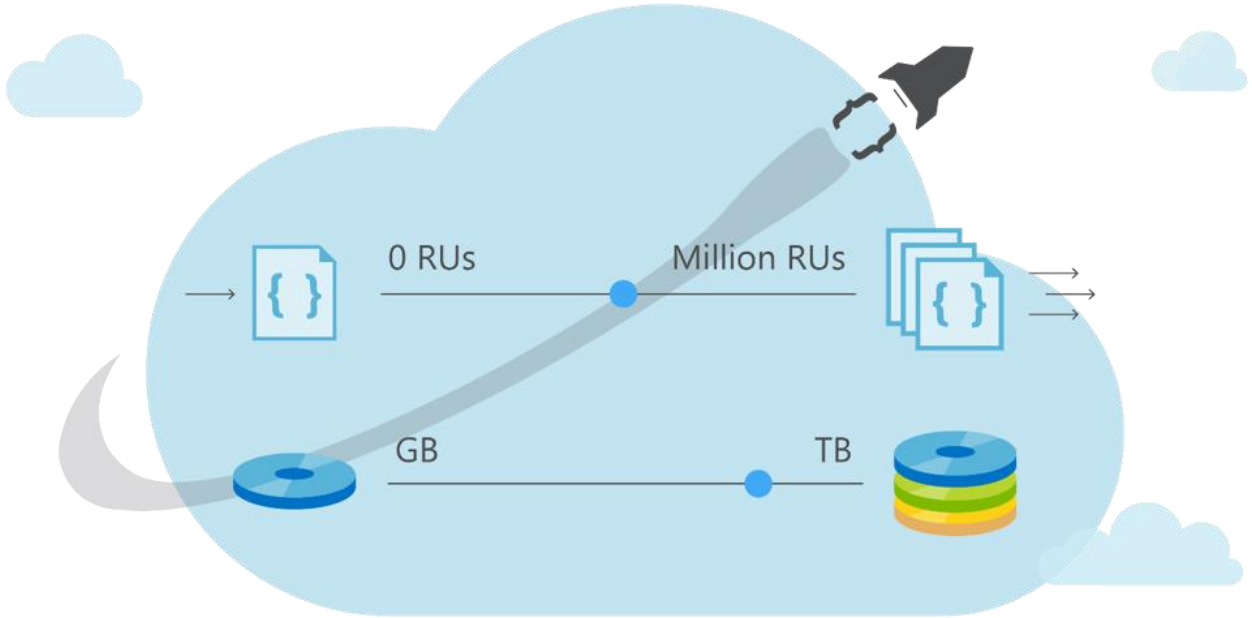
BLUEMETAL
A division of Insight

Horizontal partitioning within a container



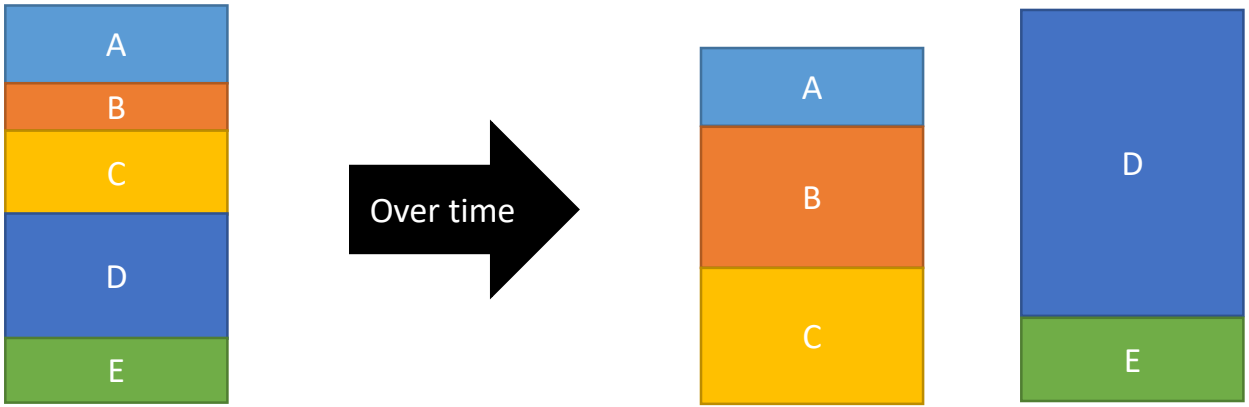
- Logical, horizontal shard identified by a hashed Partition Key
- Each unique hashed Key value is a "partition"
- Partition Key choice is key to low cost scalability

Scaling is defined by collection and implemented by partition



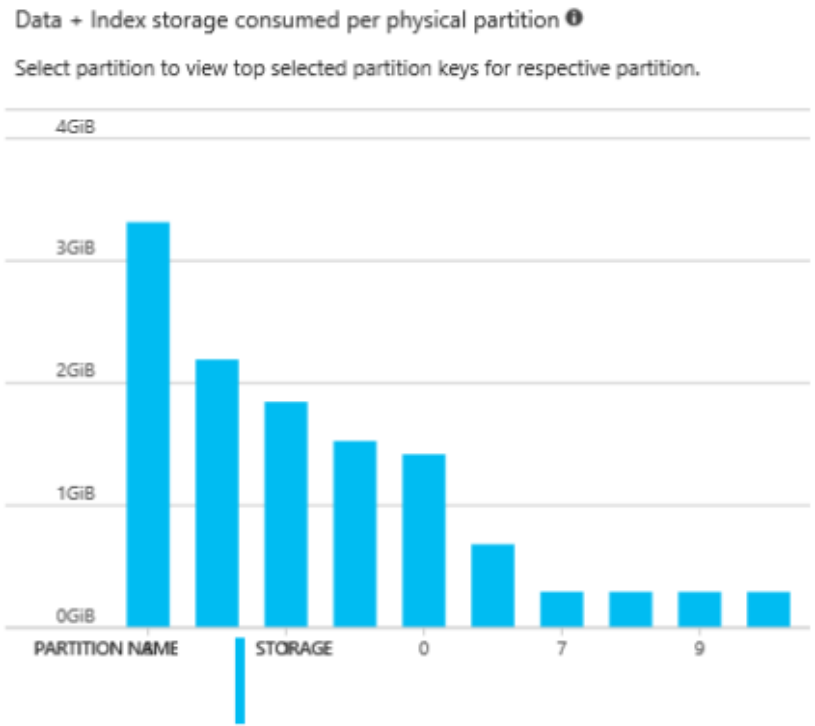
Scaling is not automated… YET

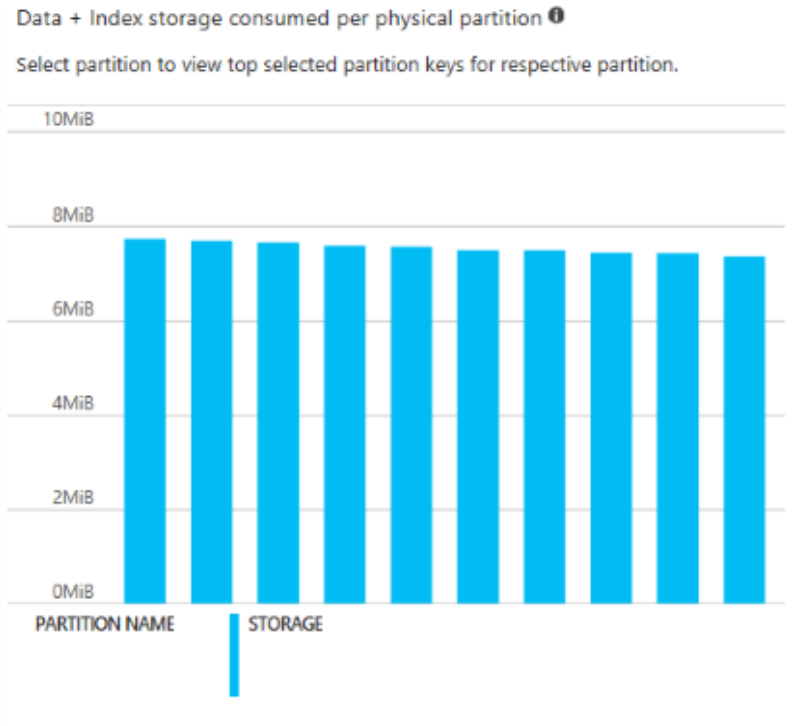
Partition redistribution



Azure implements Partition redistribution automagically to maintain SLA

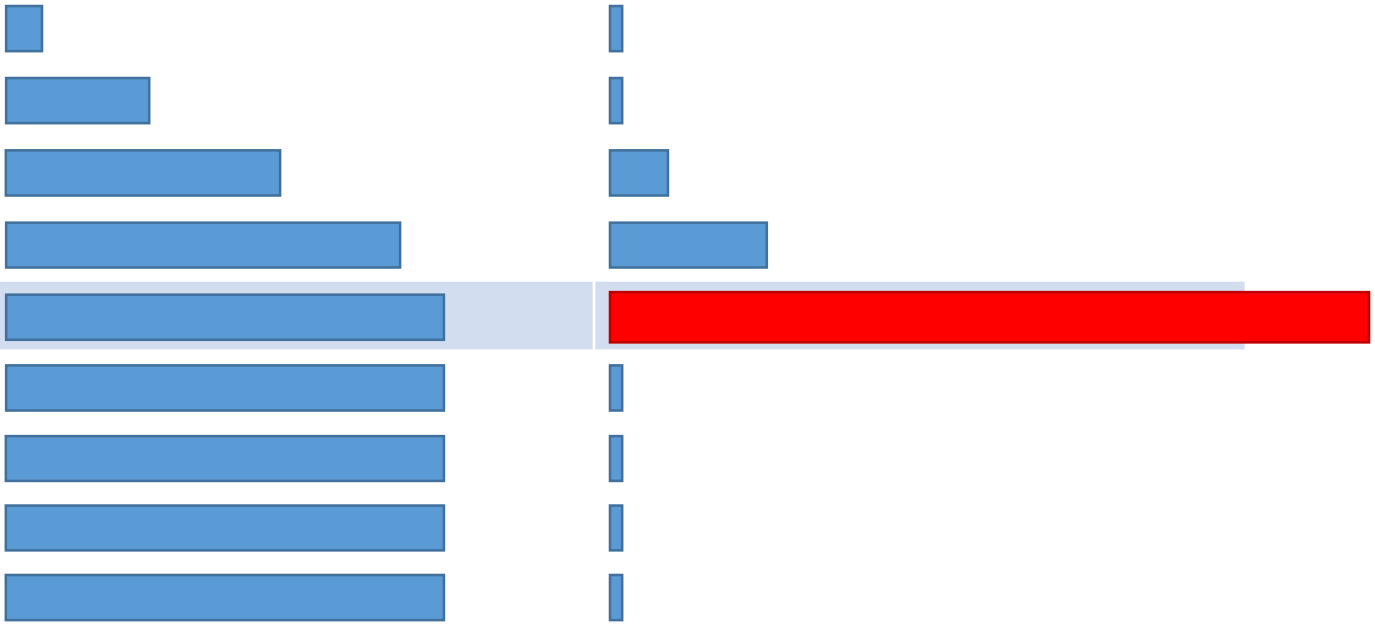
Hotspots





Airline example

Partition Key	Data	RU
2017-04-25		
2017-04-24		
2017-04-23		
2017-04-22		
2017-04-21 (today)		
2017-04-20		
2017-04-19		
2017-04-18		
2017-04-17		

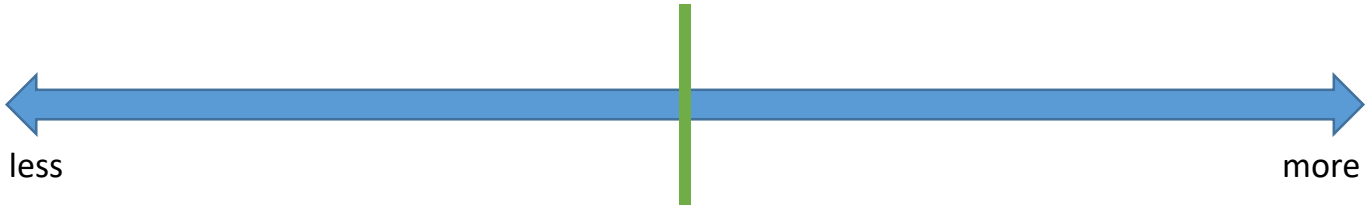


BLUEMETAL
A division of Insight

Partition Key cardinality

Larger partitions allow more efficient transactions

Smaller partitions allow more efficient scaling



Partition Key cardinality

Fan-out

- Queries that cross partitions consume more RUs
  - Each partition hit consumes at least ~1RU, even if no data is affected
- Some queries have to cross partitions
- Smart Partition Key selection can optimize performance against common use cases

```
// Query using partition key
IQueryable<DeviceReading> query =
client.CreateDocumentQuery<DeviceReading>(

UriFactory.CreateDocumentCollectionUri("db"
, "coll"))
    .Where(m => m.MetricType ==
"Temperature" && m.DeviceId == "XMS-0001");
```

```
// Query across partition keys
IQueryable<DeviceReading>
crossPartitionQuery =
client.CreateDocumentQuery<DeviceReading>(

UriFactory.CreateDocumentCollectionUri("db",
"coll"),
    new FeedOptions {
EnableCrossPartitionQuery = true })
    .Where(m => m.MetricType ==
"Temperature" && m.MetricValue > 100);
```

BLUEMETAL
A division of Insight

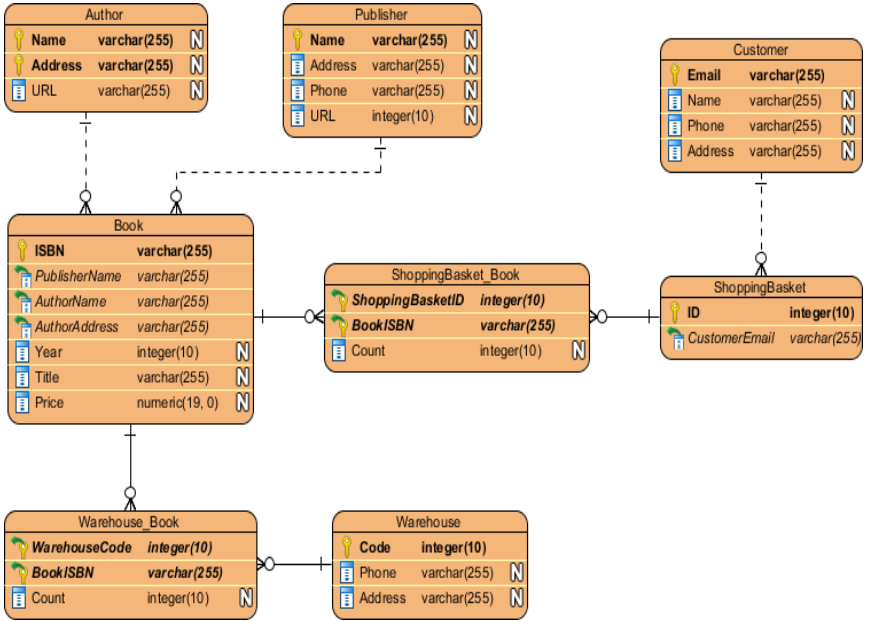
Good Partition Keys

- Evenly distribute data and load…
- …and provide targeting for operations/querys…
- …and define strong transactional bounding…
- …and provide high cardinality for scaling

BLUEMETAL
A division of Insight

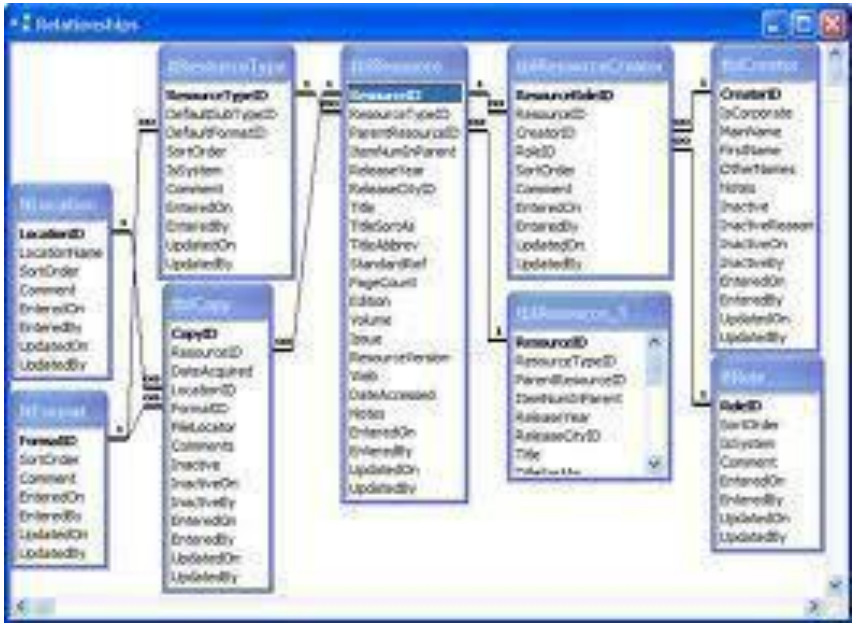
Entity types

- Document types can be identified with the “type” field
- Different types may want to be in the same collection
- Even more, different types may want to use the same Partition Key



BLUEMETAL
A division of Insight

Normalization



- Updates are atomic (whole document)
- Normalization is great for writes, but may require multiple fan-outs for reads
- De-normalization is great for reads, but may require multiple fan-outs for writes
- Be smart

Indexing

- Three modes:
  - Consistent
  - Lazy
  - None
- Set by collection and overridable by collection + path (field)
- Can have significant impact on performance and consistency

Time to Live (TTL)

- Time in seconds after last modification to persist document
- Set at collection and overridable at document level
- Based on _ts value in every document
- Deletion done async and does not impact RUs

Summary

Bringing it together

- ComosDB provides turnkey global scale
  - Replication
  - Consistency
  - SLA

- The SQL API does not behave like SQL
- Containers are limited on partition data and throughput
- Data-design (modeling) strongly affects RU consumption
  - Partition Keys
  - Normalization approach
  - Fan-out scenarios

BLUEMETAL
A division of Insight


BLUEMETAL
A division of Insight
Thank You

2018 Global Azure BOOTCAMP

A BIG thank you to the 2018 Global Sponsors!

BLUEMETAL
A division of Insight