Microsoft

AZ-203.3
Module 02: Develop solutions that use Azure Cosmos DB

Prashanth Kumar

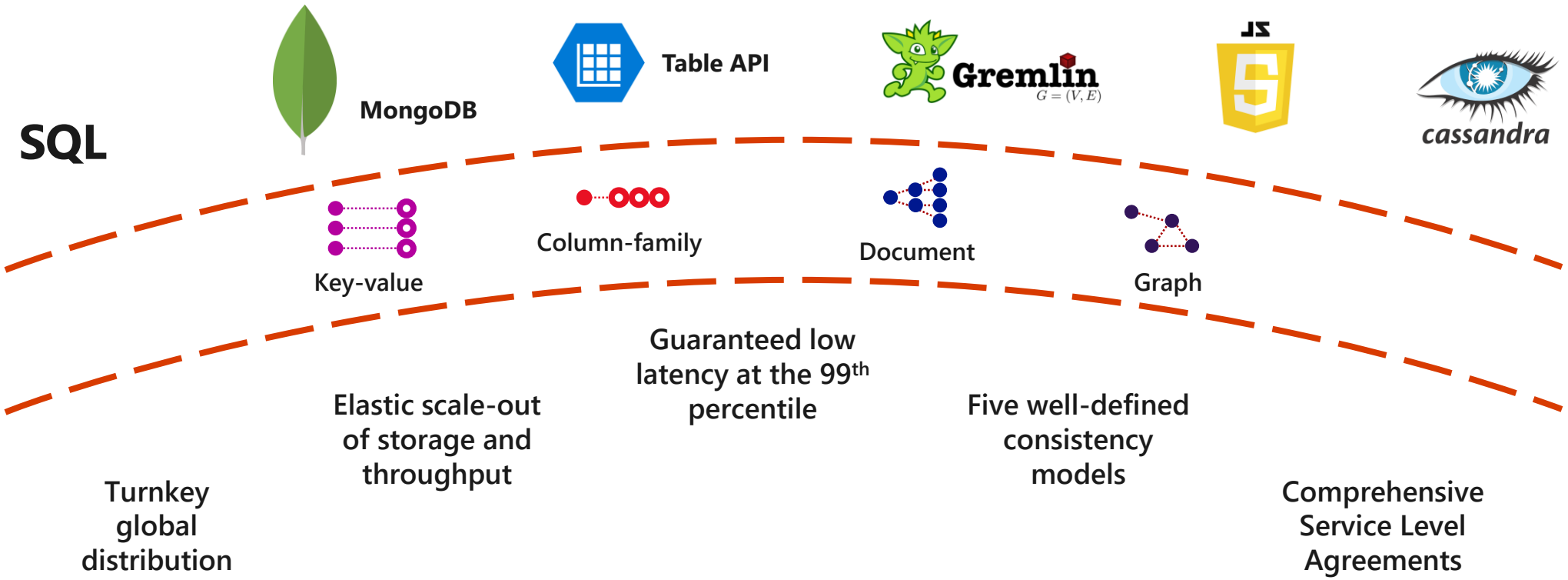
Microsoft

Topics

- Azure Cosmos DB
- Managing containers and items
- Create and update documents by using code
- Server-side programming and features

Lesson 01: Azure Cosmos DB

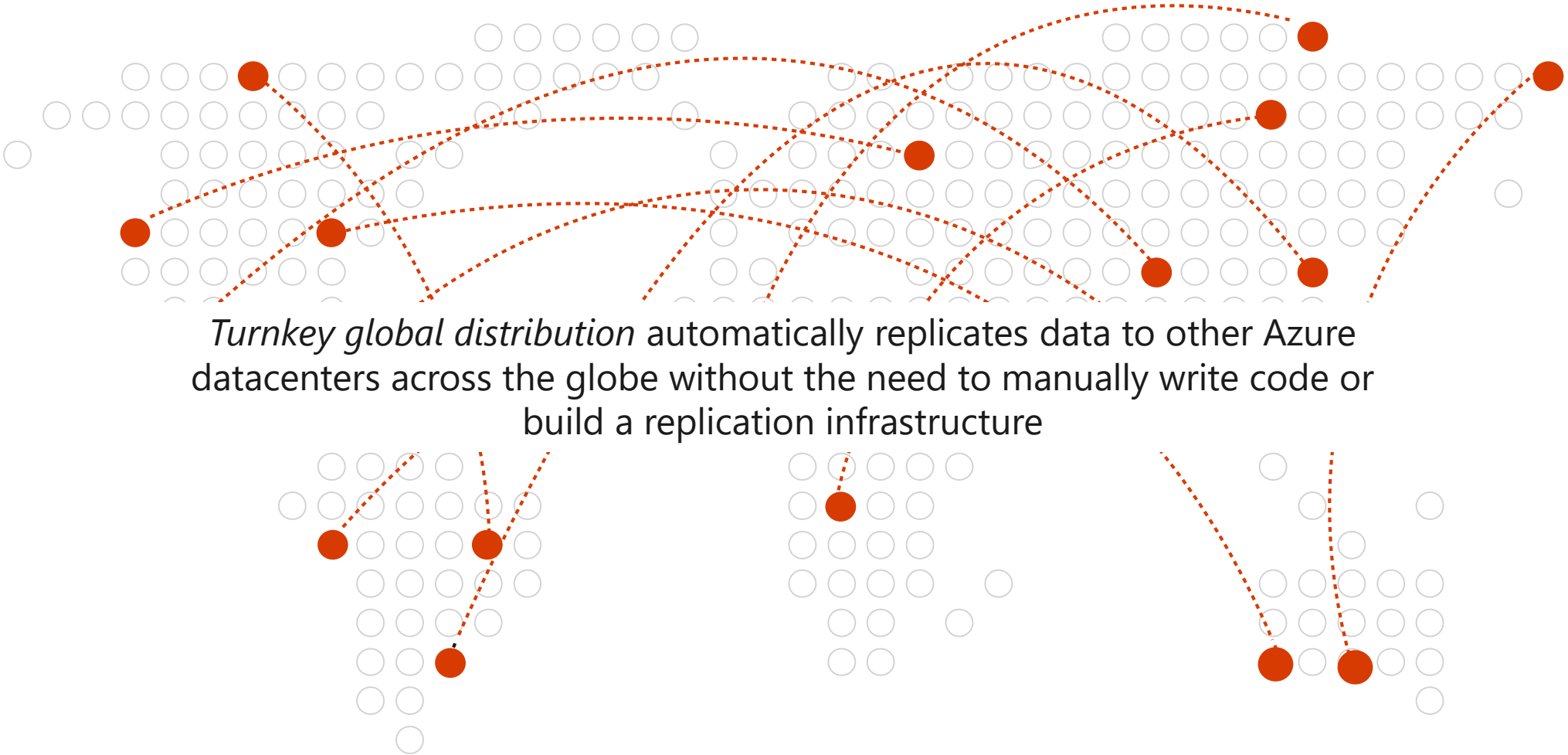
Azure Cosmos DB



Core functionality

- Global replication
  - Automatic and synchronous multi-region replication
  - Supports automatic and manual failover
- Varied consistency levels
  - Offers five consistency models
  - Provides control over performance-consistency tradeoffs, backed by comprehensive SLAs
- Low latency
  - Serve <10 ms read and <15 ms write requests at the 99th percentile
- Elastic scale-out
  - Elastically scale throughput from 10 to 100s of millions of requests/sec across multiple regions
  - Support for requests/sec for different workloads

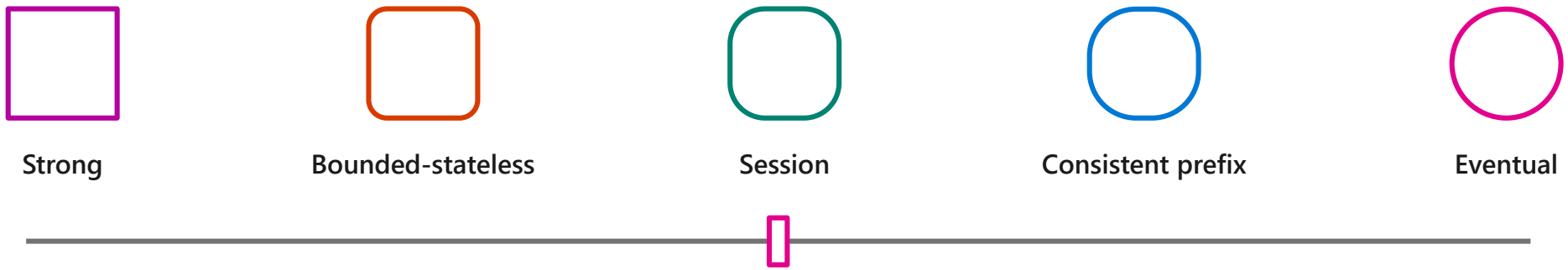
Global Replication



Turnkey global distribution automatically replicates data to other Azure datacenters across the globe without the need to manually write code or build a replication infrastructure

Consistency levels

Azure Cosmos DB provides five consistency levels:



Consistency levels

Consistency Level	Description
Strong	When a write operation is performed on your primary database, the write operation is replicated to the replica instances. The write operation is committed (and visible) on the primary only after it has been committed and confirmed by all replicas.
Bounded Stateless	This level is similar to the Strong level with the major difference that you can configure how stale documents can be within replicas. Staleness refers to the quantity of time (or the version count) a replica document can be behind the primary document.
Session	This level guarantees that all read and write operations are consistent within a user session. Within the user session, all reads and writes are monotonic and guaranteed to be consistent across primary and replica instances.
Consistent Prefix	This level has loose consistency but guarantees that when updates show up in replicas, they will show up in the correct order (that is, as prefixes of other updates) without any gaps.
Eventual	This level has the loosest consistency and essentially commits any write operation against the primary immediately. Replica transactions are asynchronously handled and will eventually (over time) be consistent with the primary. This tier has the best performance, because the primary database does not need to wait for replicas to commit to finalize it's transactions.

APIs

- MongoDB API
  - Acts as a massively scalable MongoDB service powered by the Azure Cosmos DB platform
  - Compatible with existing MongoDB libraries, drivers, tools, and applications

- Table API
  - A key-value database service built to provide premium capabilities to existing Azure Table storage applications without making any app changes

- Gremlin API
  - A fully managed, horizontally scalable graph database service
  - Easy-to-build and run applications that work with highly connected datasets supporting Open Graph APIs (based on the Apache TinkerPop specification, Apache Gremlin)

APIs (cont.)

- Cassandra API
  - Globally distributed Apache Cassandra service powered by the Azure Cosmos DB platform
  - Compatible with existing Apache Cassandra libraries, drivers, tools, and applications
- SQL API
  - JavaScript and JavaScript Object Notation (JSON) native API based on the Azure Cosmos DB database engine
  - Provides query capabilities rooted in SQL
  - Query for documents based on their identifiers or make deeper queries based on properties of the document, complex objects, or the existence of specific properties
  - Supports the execution of JavaScript logic within the database in the form of stored procedures, triggers, and user-defined functions

Demo: Creating an Azure Cosmos DB account

Consistency levels and Azure Cosmos DB APIs

Apache Cassandra

Apache Cassandra 4.x	Azure Cosmos DB (multi-region)	Azure Cosmos DB (single region)
ONE, TWO, THREE	Consistent prefix	Consistent prefix
LOCAL_ONE	Consistent prefix	Consistent prefix
QUORUM, ALL, SERIAL	Bounded stateless	Strong
LOCAL_QUORUM	Bounded stateless	Strong
LOCAL_SERIAL	Bounded stateless	Strong

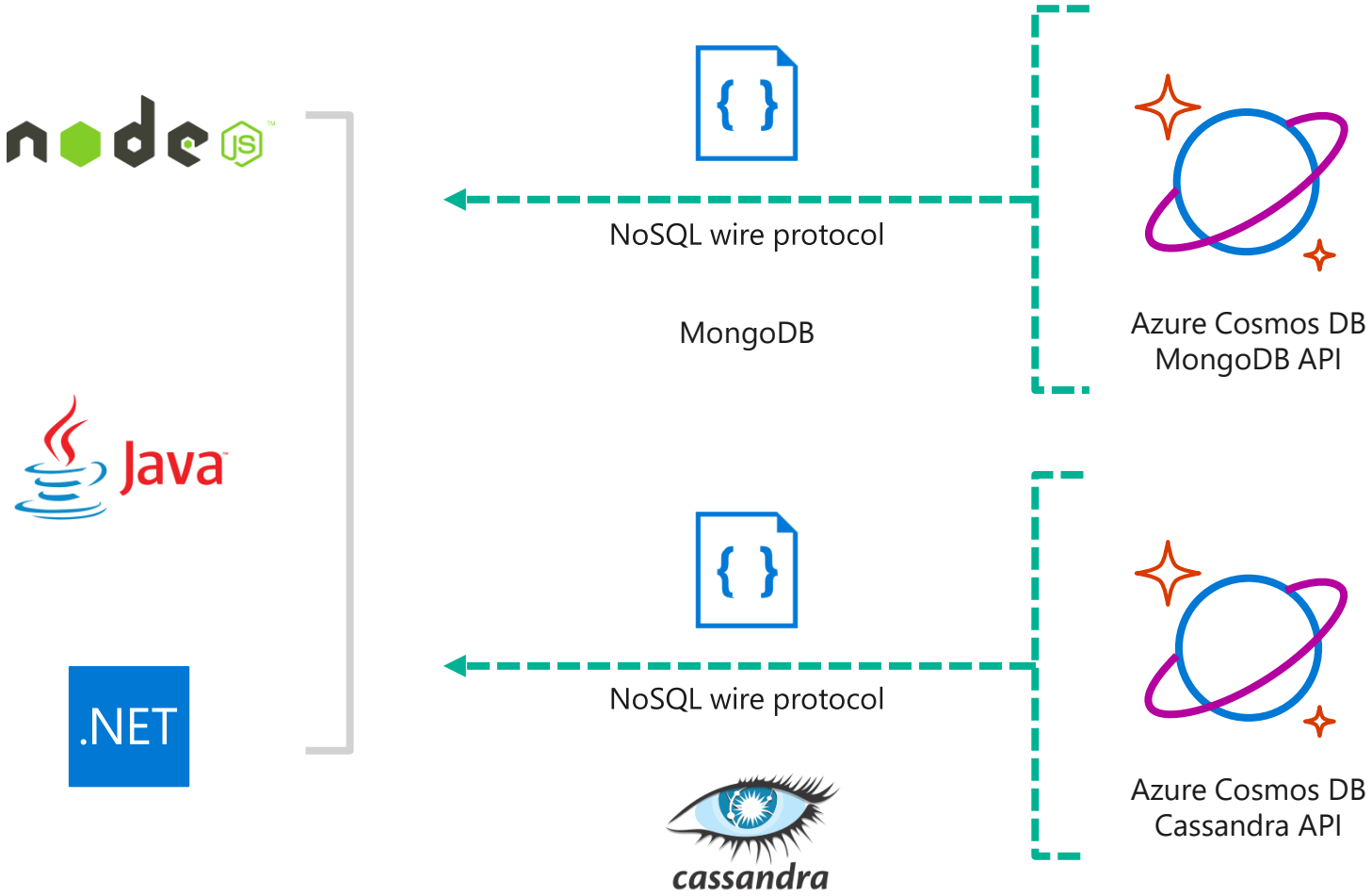
MongoDB (3.4)

MongoDB 3.4	Azure Cosmos DB (multi-region)	Azure Cosmos DB (single region)
Linearizable	Strong	Strong
Majority	Bounded staleness	Strong
Local	Consistent prefix	Consistent prefix

Migrating from NoSQL

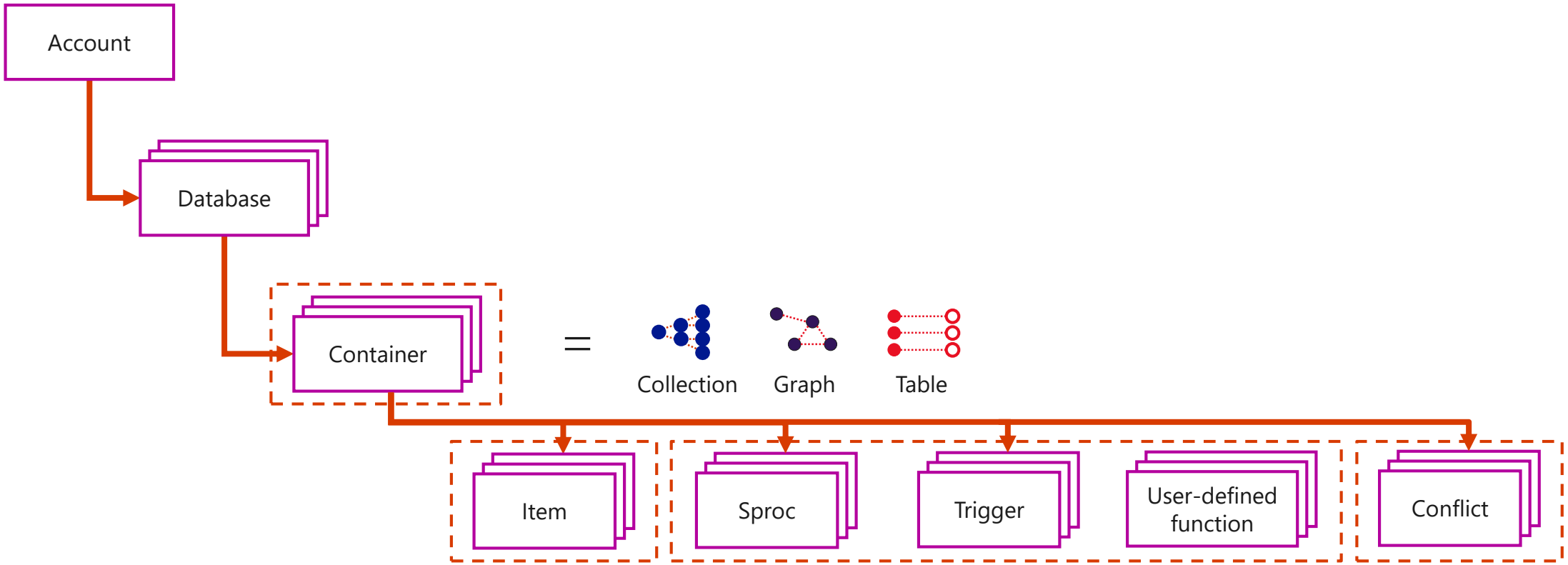
- Many NoSQL database engines are simple to get started with, but they might cause problems as you scale, including:
  - Tedious setup and maintenance requirements for a multiple-server database cluster
  - Expensive and complex high-availability solutions
  - Challenges in achieving end-to-end security, including encryption at rest and in flight
  - Required resource overprovisioning and unpredictable costs to achieve scale
- Azure Cosmos DB provides NoSQL-as-a-service for:
  - MongoDB
  - Cassandra
  - Gremlin

Migrating from NoSQL



Lesson 02: Managing containers and items

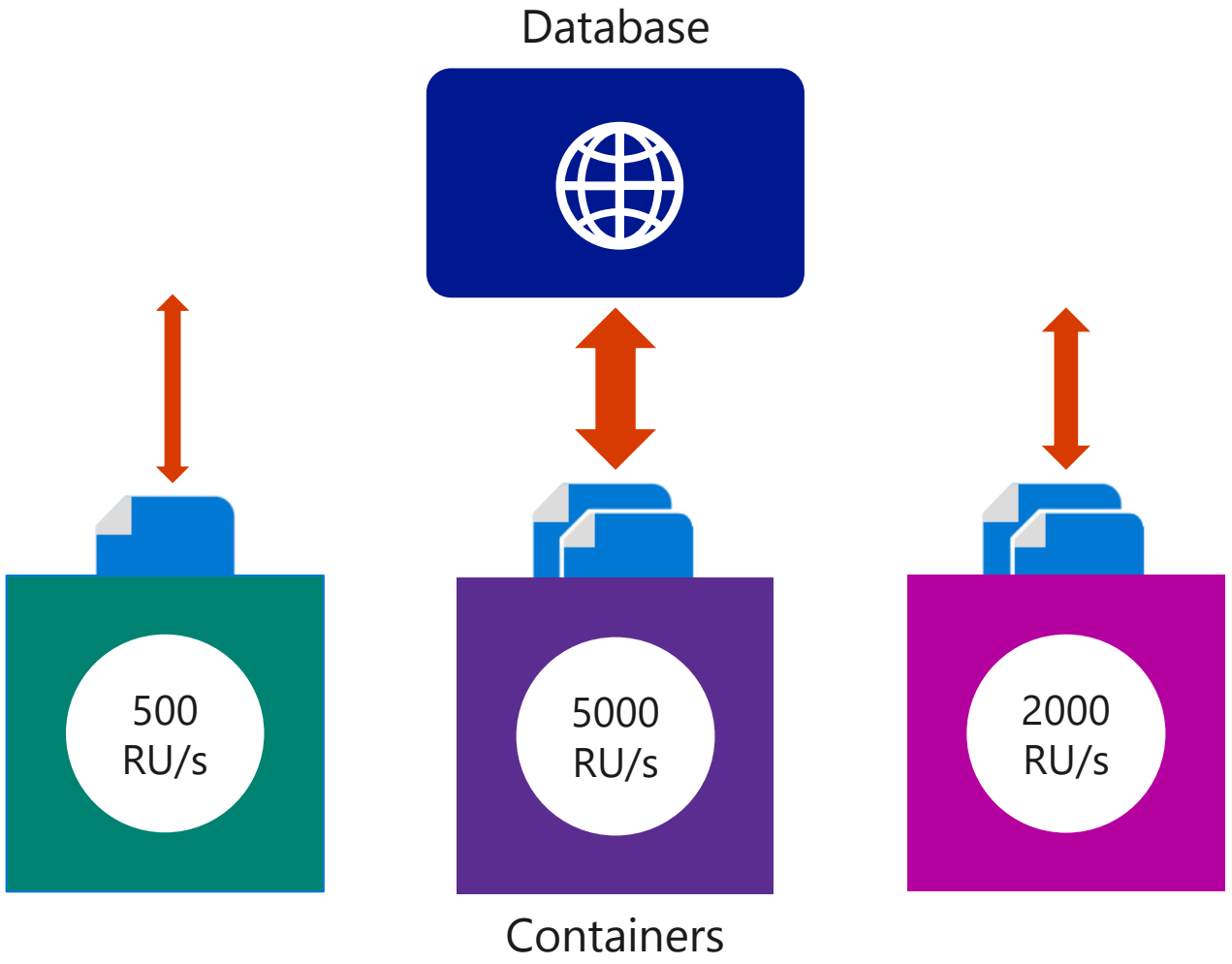
Resource hierarchy



Resource hierarchy (continued)

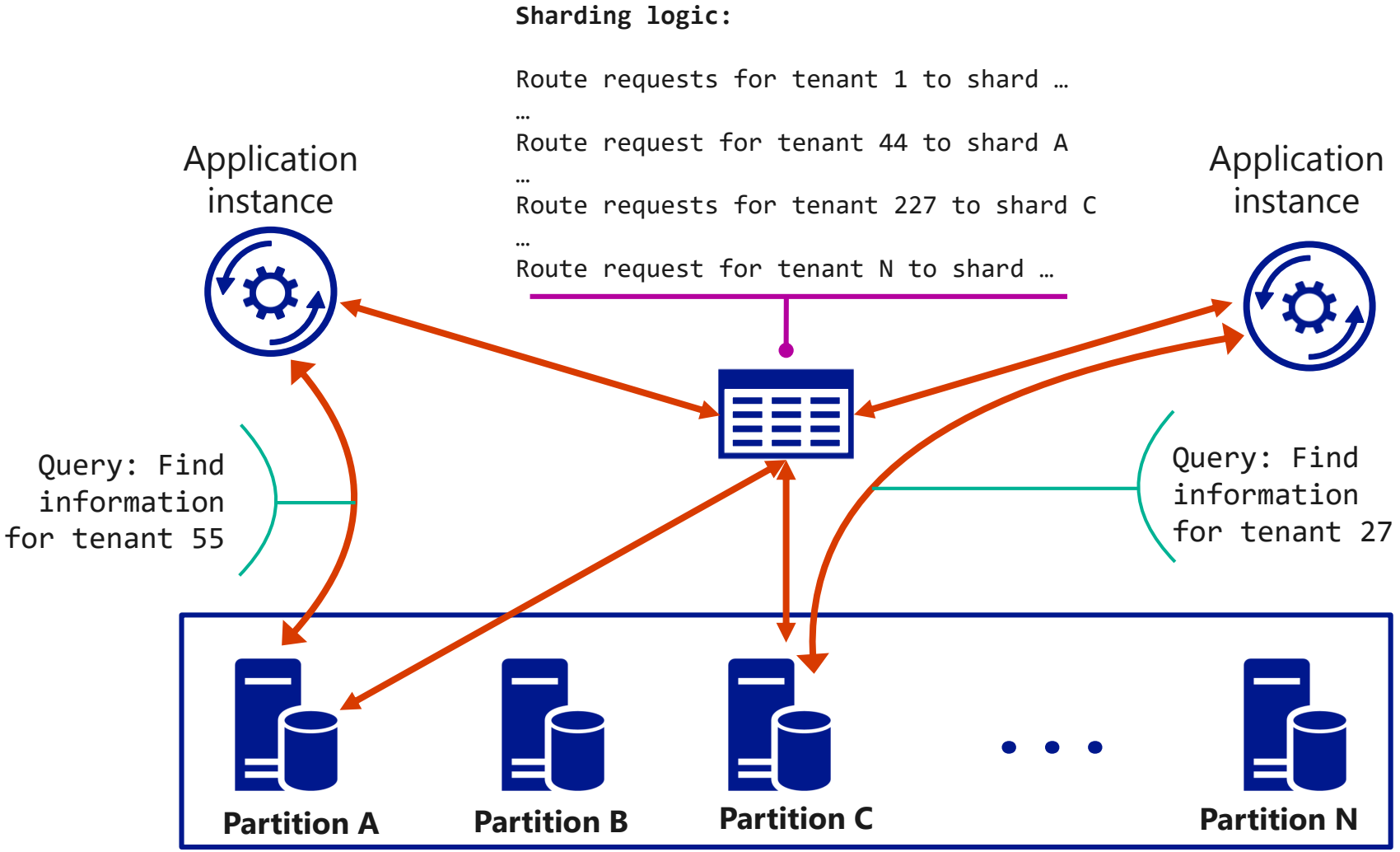
Resource	Description
Account	A set of databases
Database	Logical container for containers that can (optionally) share throughput across the containers
Collection (container)	A group of Items and programmatic resources usually related in some way
Document (item)	An arbitrary unit of content In many cases, this would be a JSON document
Stored procedure (sproc)	Application logic written in JavaScript executed within the database engine as a transaction
Trigger	Application logic written in JavaScript executed before or after either an insert, replace, or delete operation
User-defined function	Application logic written in JavaScript to extend the SQL API query language

Containers

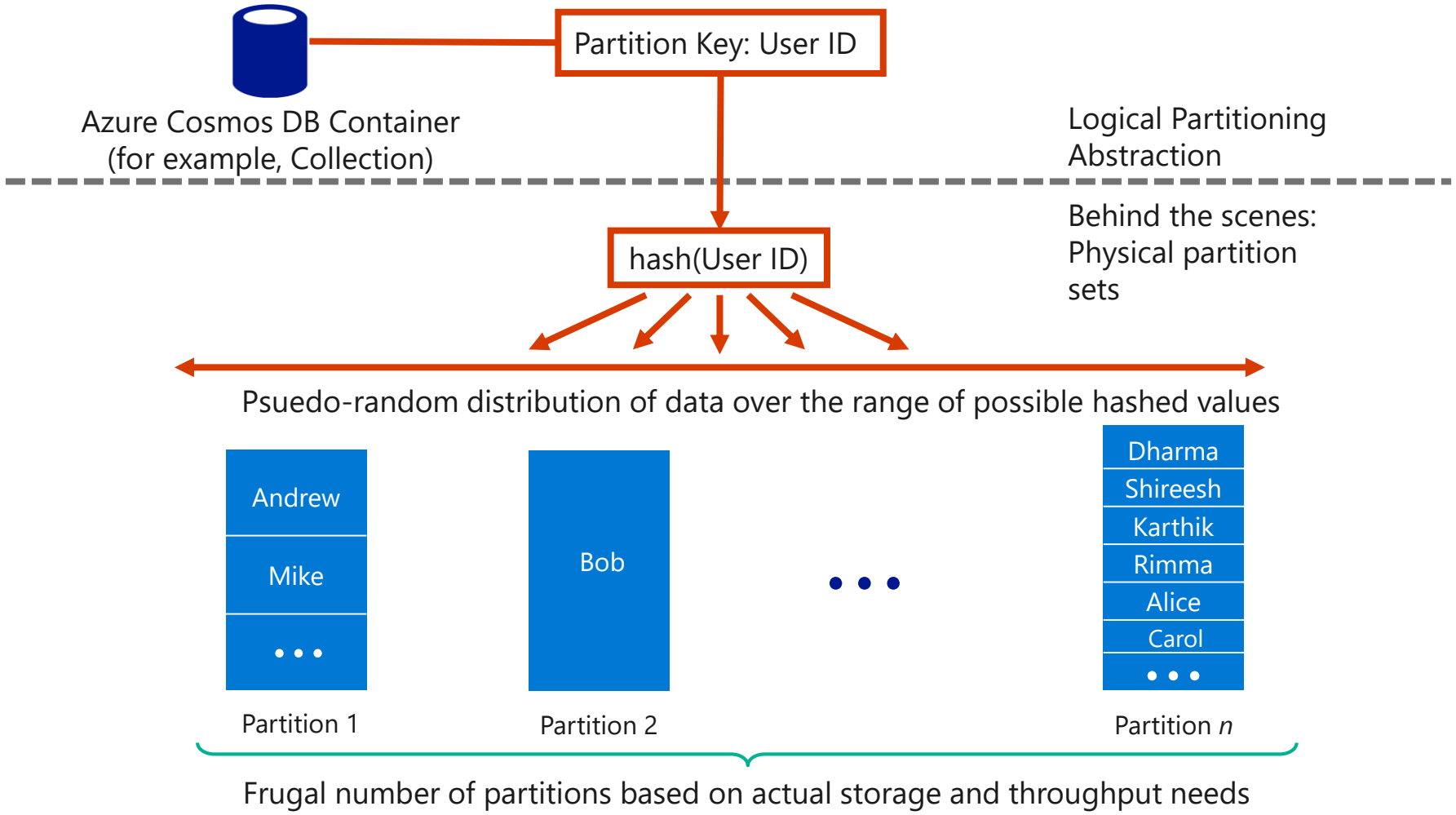


Demo: Dynamically adjusting container throughput

Partitioning



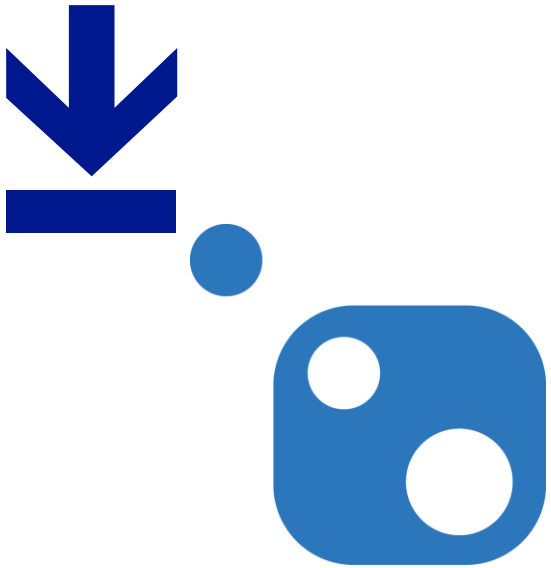
Partitioning implementation



Lesson 03: Create and update documents by using code

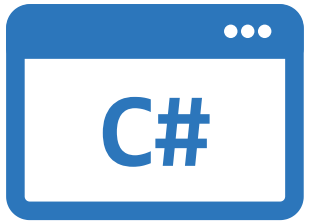
Manage collections and documents

- Install the **Microsoft.Azure.DocumentDB.Core** package from NuGet
- Use the following namespaces
  - Microsoft.Azure.Documents
  - Microsoft.Azure.Documents.Client
- Use the DocumentClient class



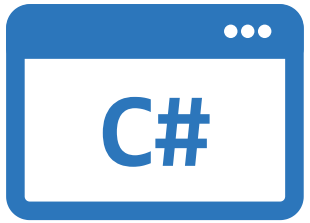
Accessing collections by using .NET

```
using Microsoft.Azure.Documents;
using Microsoft.Azure.Documents.Client;

DocumentClient client = new DocumentClient(new Uri("[endpoint]"), "[key]");

Uri collectionUri = UriFactory.CreateDocumentCollectionUri(databaseName,
collectionName);
```



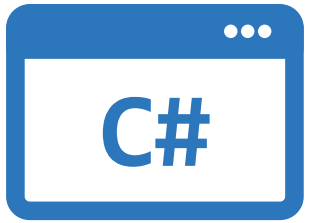
Reading documents by using .NET

```
// create Uri referencing document
Uri documentUri = UriFactory.CreateDocumentUri(databaseName, collectionName,
documentId);

// Use the ReadDocumentAsync method
SerializedType document = await this.client.
ReadDocumentAsync<SerializedType>(documentUri);
```



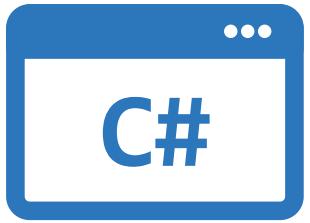
Creating documents by using .NET

```
var document = new {
    firstName = "Alex",
    lastName = "Leh"
}

await this.client.CreateDocumentAsync(collectionUri, document);
```



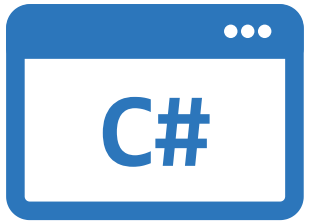
Querying documents by using .NET (continued 1)

```
var query = client.CreateDocumentQuery<Family>(
    collectionUri, new SqlQuerySpec() {
        QueryText = "SELECT * FROM f WHERE (f.surname = @lastName)",
        Parameters = new SqlParameterCollection()
        {
            new SqlParameter("@lastName", "Andt")
        }
    }, DefaultOptions
);

var families = query.ToList();
```



Querying documents by using .NET (continued 2)

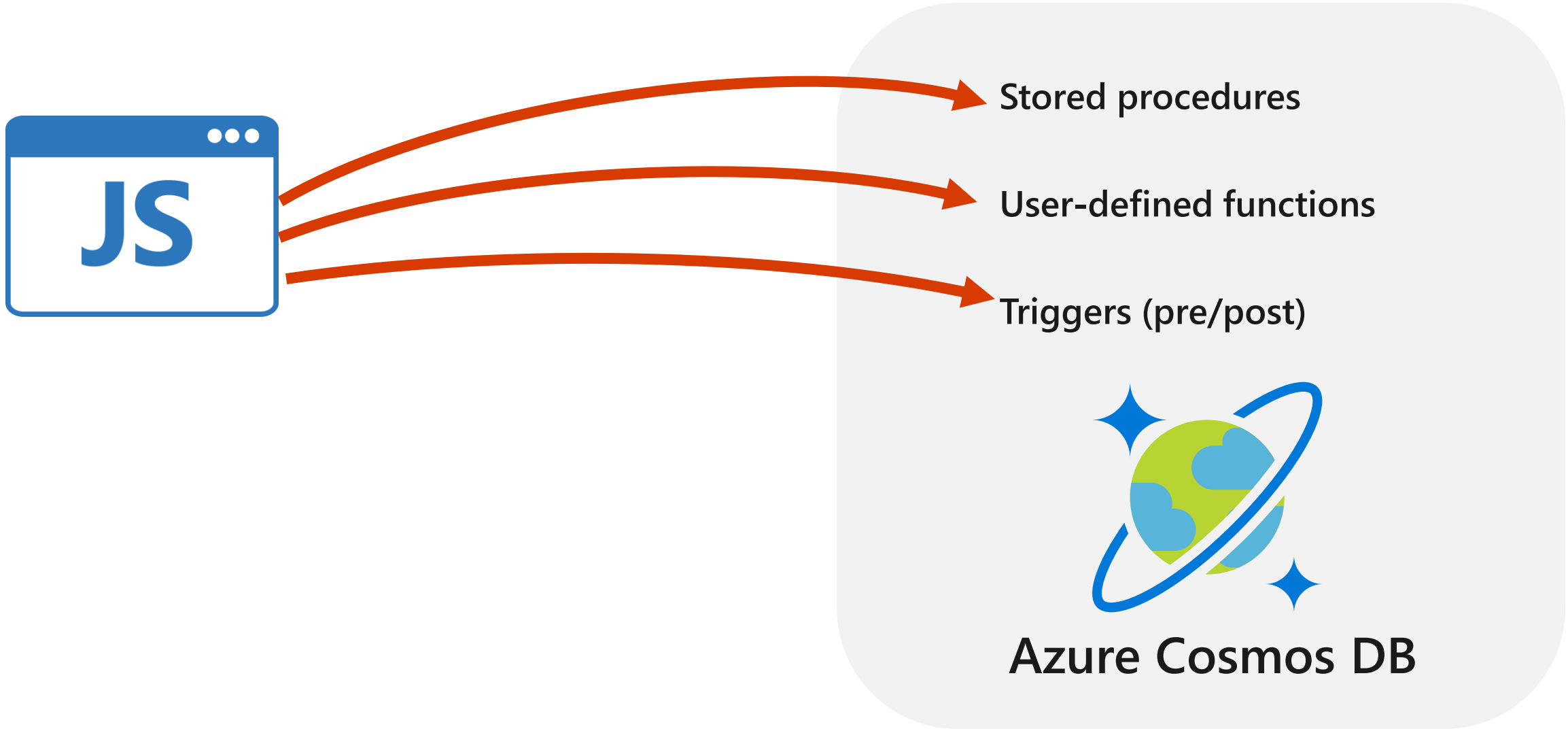
```
var query = client.CreateDocumentQuery<Family>(collectionUri)
    .Where(d => d.Surname = "Andt")
    .Select(d => new { Name = d.Id, City = d.Address?.City)
    .AsDocumentQuery();

var families = query.ToList();
```



Demo: Managing Azure Cosmos DB by using .NET

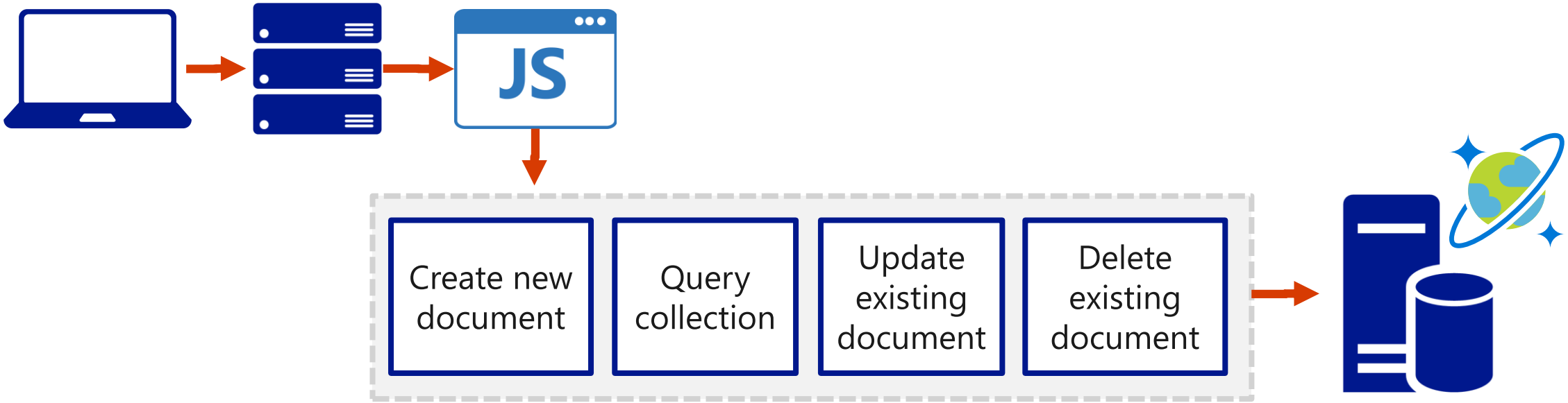
Lesson 04: Server-side programming and features

JavaScript and Azure Cosmos DB



Stored procedures

- In Azure Cosmos DB, JavaScript is hosted in the same memory space as the database
- Requests made within stored procedures and triggers run in the same scope of a database session

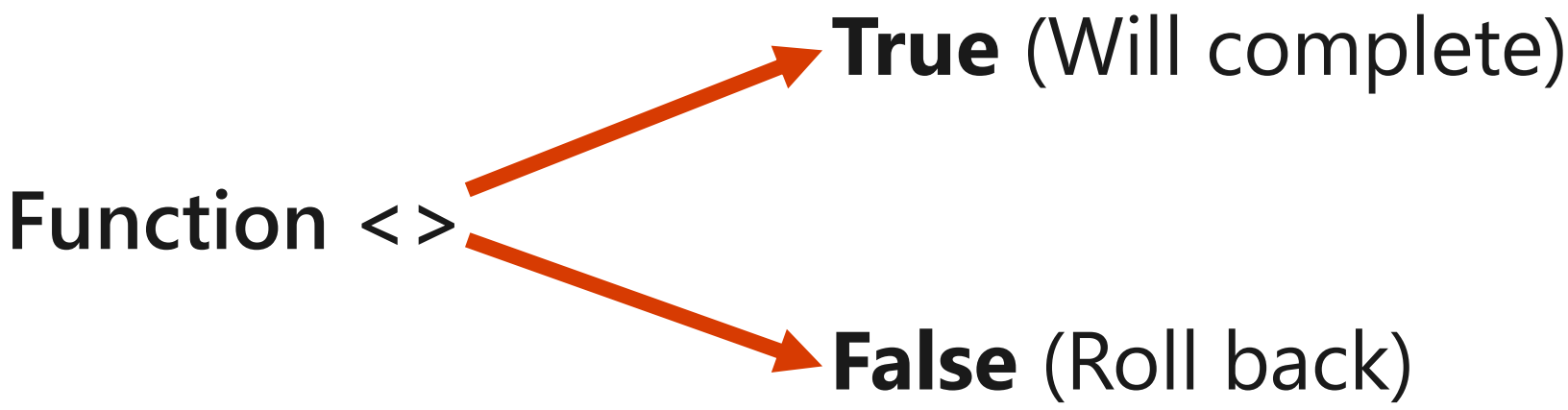


Stored procedure in JavaScript

```
function createSampleDocument(documentToCreate) {
    var context = getContext();
    var collection = context.getCollection();
    var accepted = collection.createDocument(
        collection.getSelfLink(),
        documentToCreate,
        function (error, documentCreated) {
            context.getResponse().setBody(documentCreated.id)
        }
    );
    if (!accepted) return;
}
```

Bounded execution

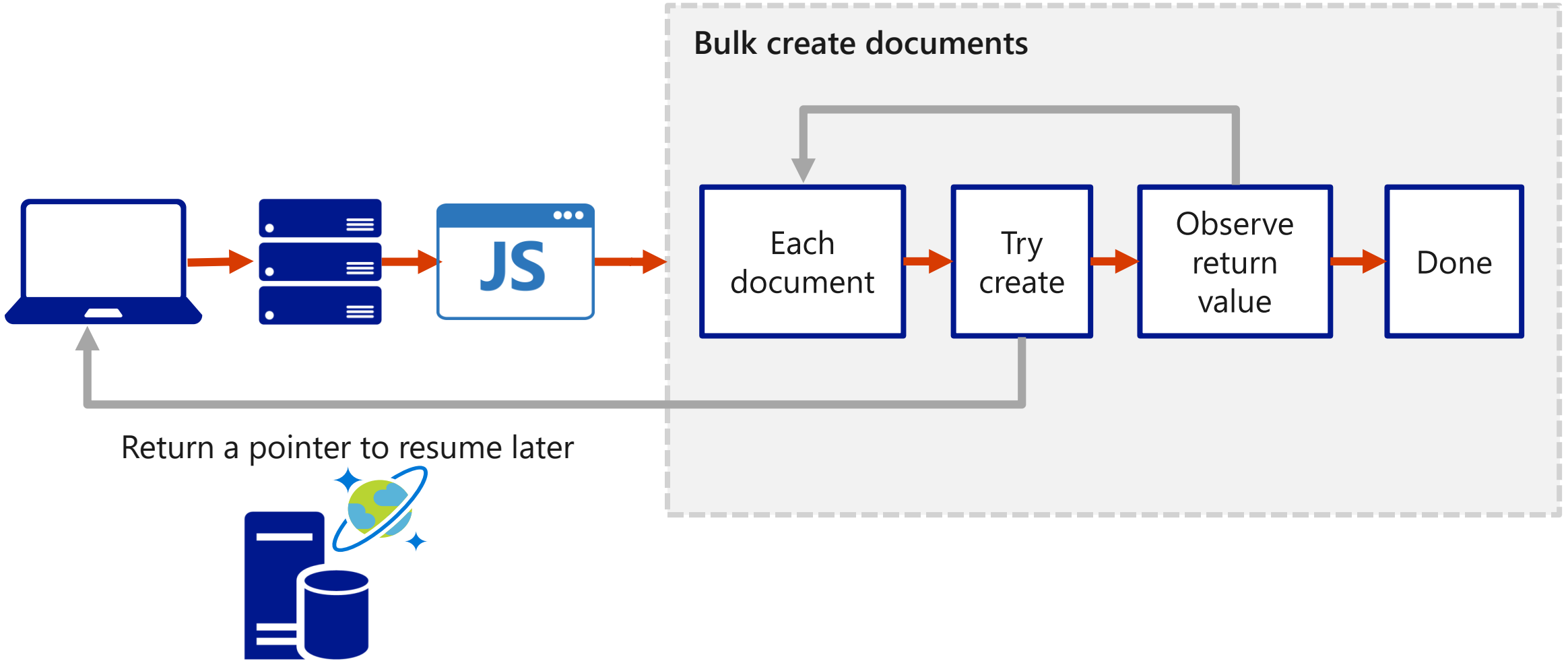
- All Azure Cosmos DB operations must complete within a limited amount of time
  - Specifically, stored procedures have a limited amount of time to run on the server
- All collection functions return a Boolean value that represents whether that operation will complete or not



Transaction continuation

- JavaScript functions can implement a continuation-based model to batch or resume execution
- The continuation value can be any value of your choice
- Your applications can then use this value to resume a transaction from a new starting point

Transaction continuation (cont.)



User-defined functions in JavaScript

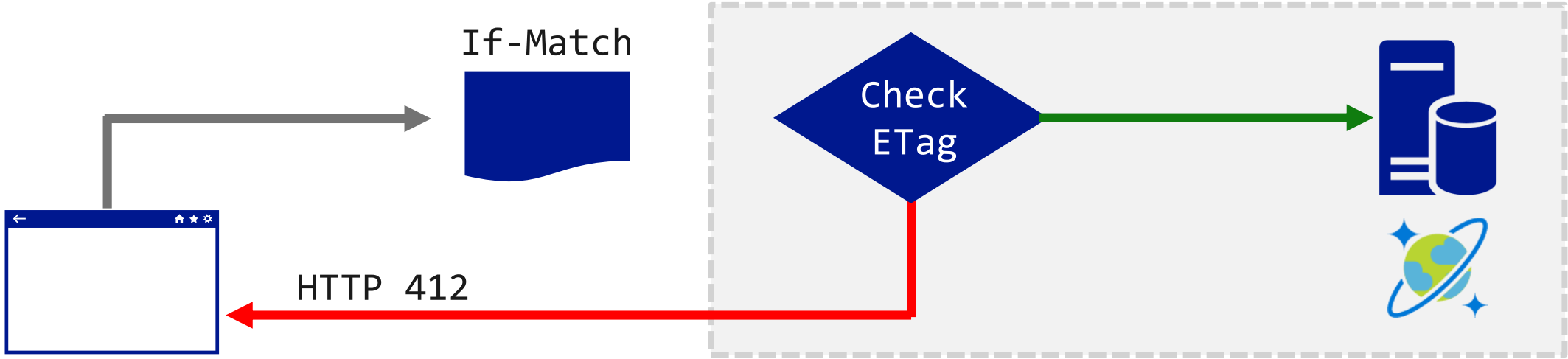
```
var taxUdf = {
    id: "tax",
    serverScript: function tax(income) {
        if (income == undefined)
            throw 'no input';
        if (income < 1000)
            return income * 0.1;
        else if (income < 10000)
            return income * 0.2;
        else
            return income * 0.4;
    }
}
```

User-defined functions in SQL queries

```
SELECT
    *
FROM
    TaxPayers t
WHERE
    udf.tax(t.income) > 20000
```

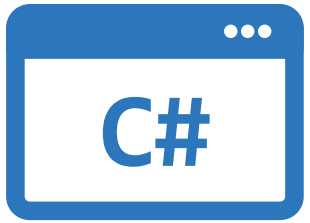
Optimistic concurrency

- The SQL API supports optimistic concurrency control through HTTP ETags
- Every SQL API resource has an ETag system property
- ETags can be used with the If-Match HTTP request header to allow the server to decide whether a resource should be updated



Controlling concurrency in .NET

```
try
{
    var ac = new AccessCondition { Condition = readDoc.ETag, Type =
        AccessConditionType.IfMatch };
    await client.ReplaceDocumentAsync(readDoc, new RequestOptions {
        AccessCondition = ac });
}
catch (DocumentClientException dce)
{
    if (dce.StatusCode == HttpStatusCode.PreconditionFailed)
    {
        Console.WriteLine("Another process has updated the record");
    }
}
```



Microsoft

Review

- Azure Cosmos DB
- Managing containers and items
- Create and update documents by using code
- Server-side programming and features