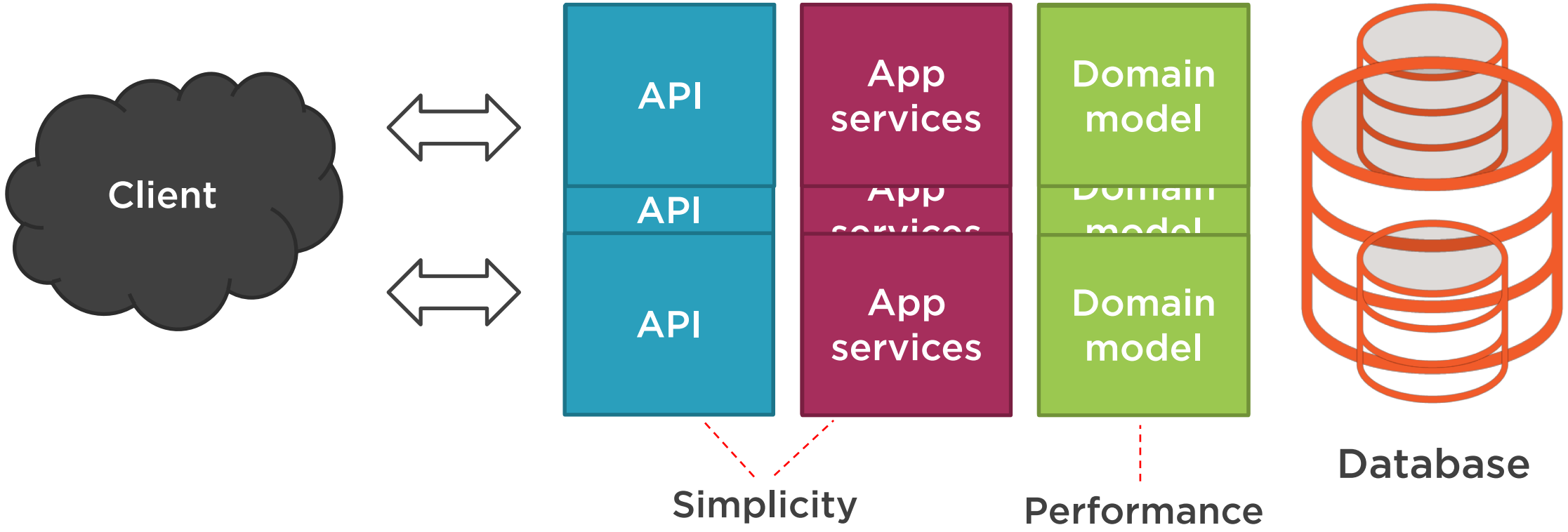
Introducing a Separate Database for Queries

Vladimir Khorikov

@vkhorikov www.enterprisecraftsmanship.com

Meet Scalability



Scalability

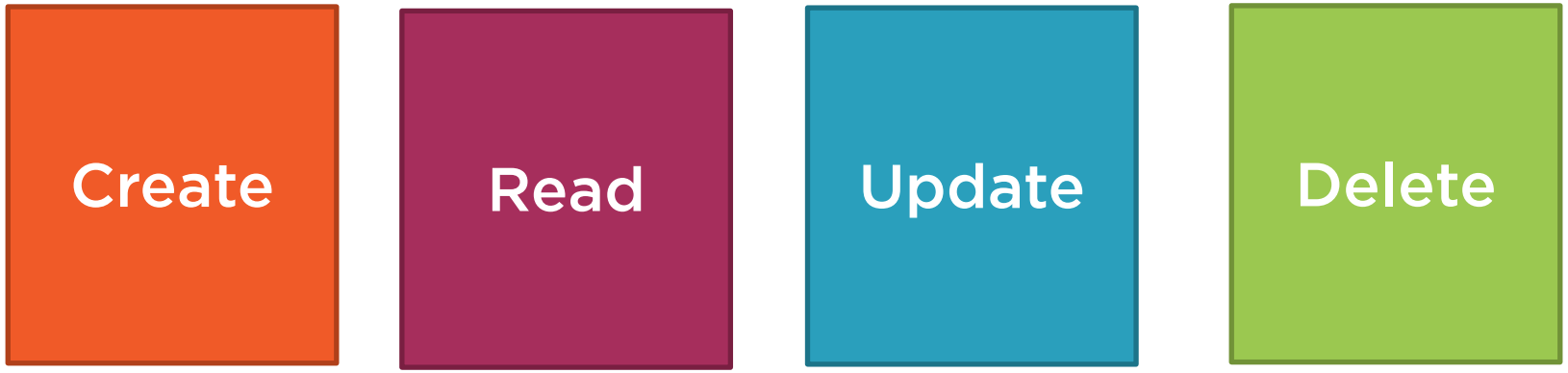
Meet Scalability

Scalability vs. **Performance**

✓ Enables utilization of a number of servers

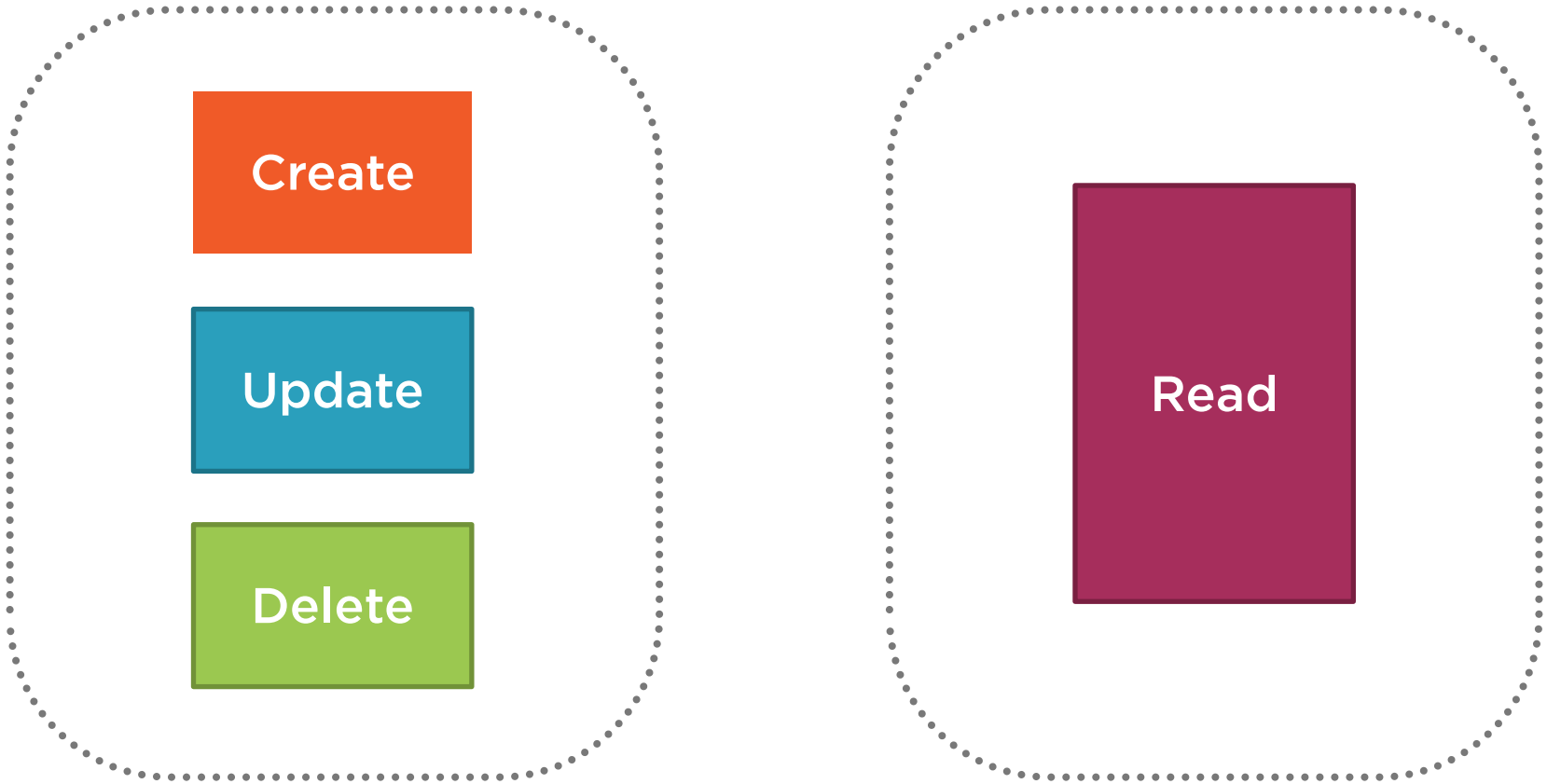
✗ Bound to a single server

Meet Scalability



Independent scaling

Meet Scalability



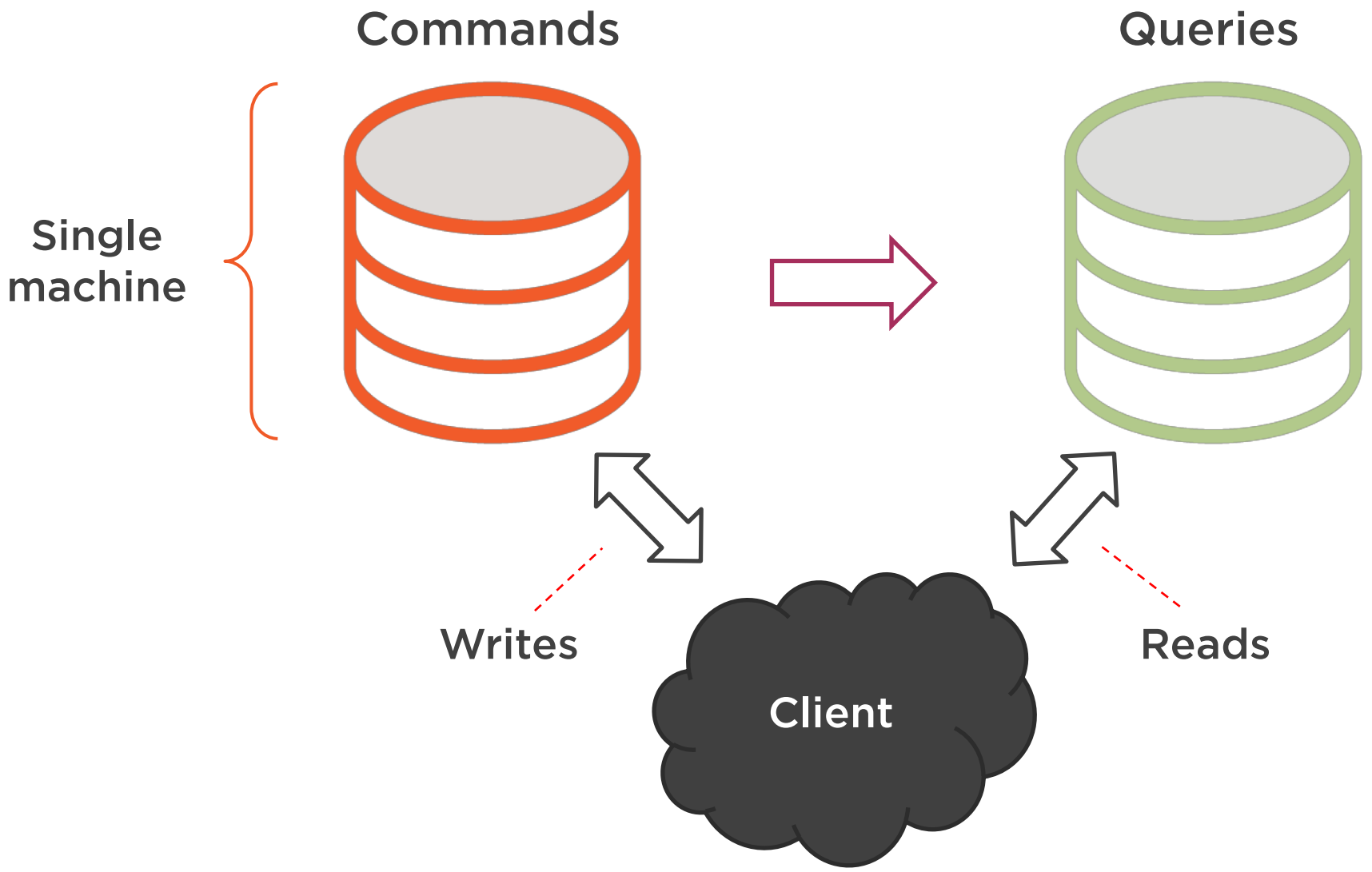
Command side

1 server

Query side

10 servers

Meet Scalability

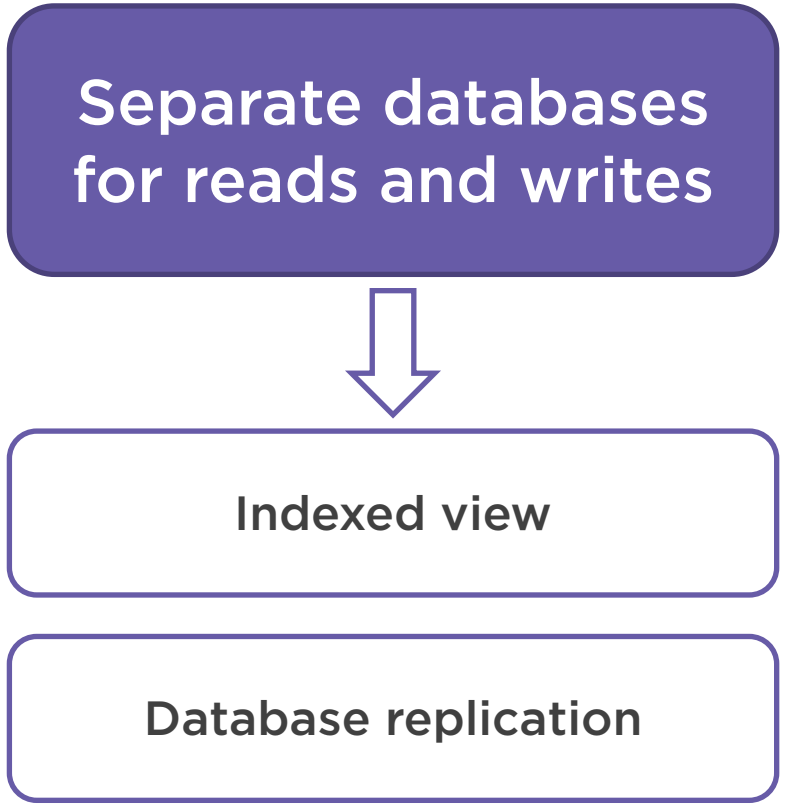


Meet Scalability

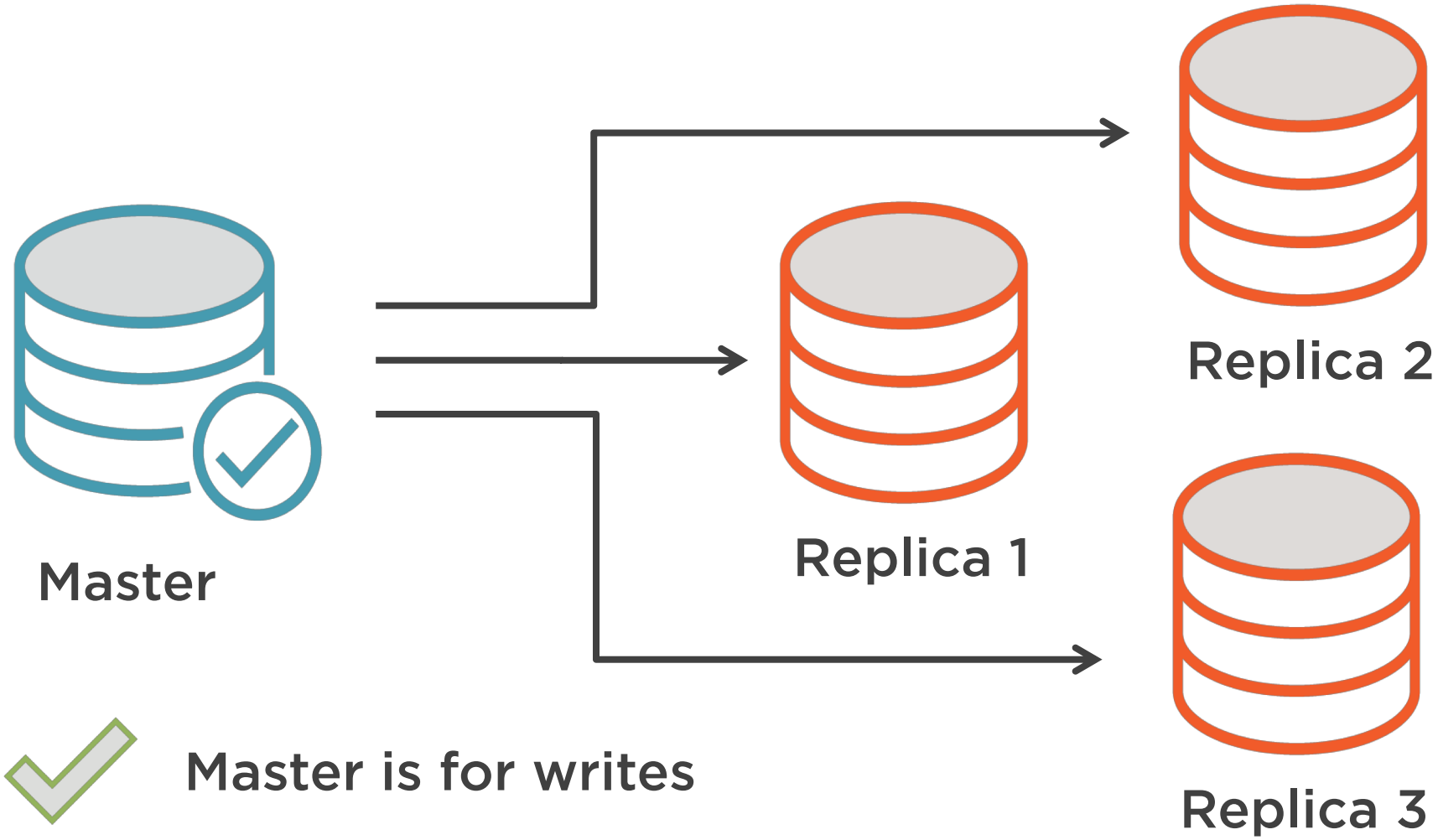
Need sharding to scale commands

It's a rare requirement

Separation at the Data Level in the Real World



Separation at the Data Level in the Real World



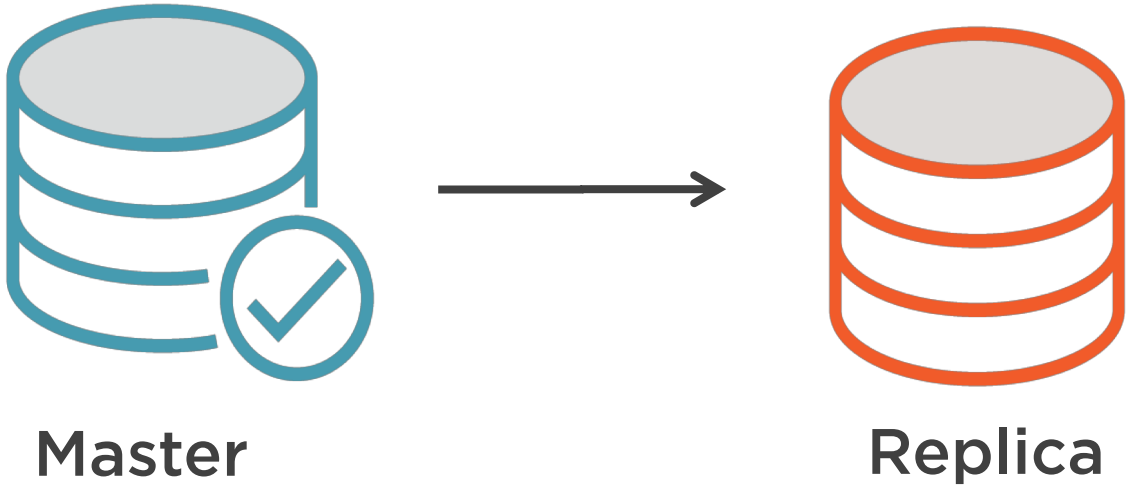
- Master is for writes
- Replicas are for reads

Separation at the Data Level in the Real World

Separate databases for reads and writes

- Indexed view
- Database replication
- elasticsearch

Designing a Database for Queries



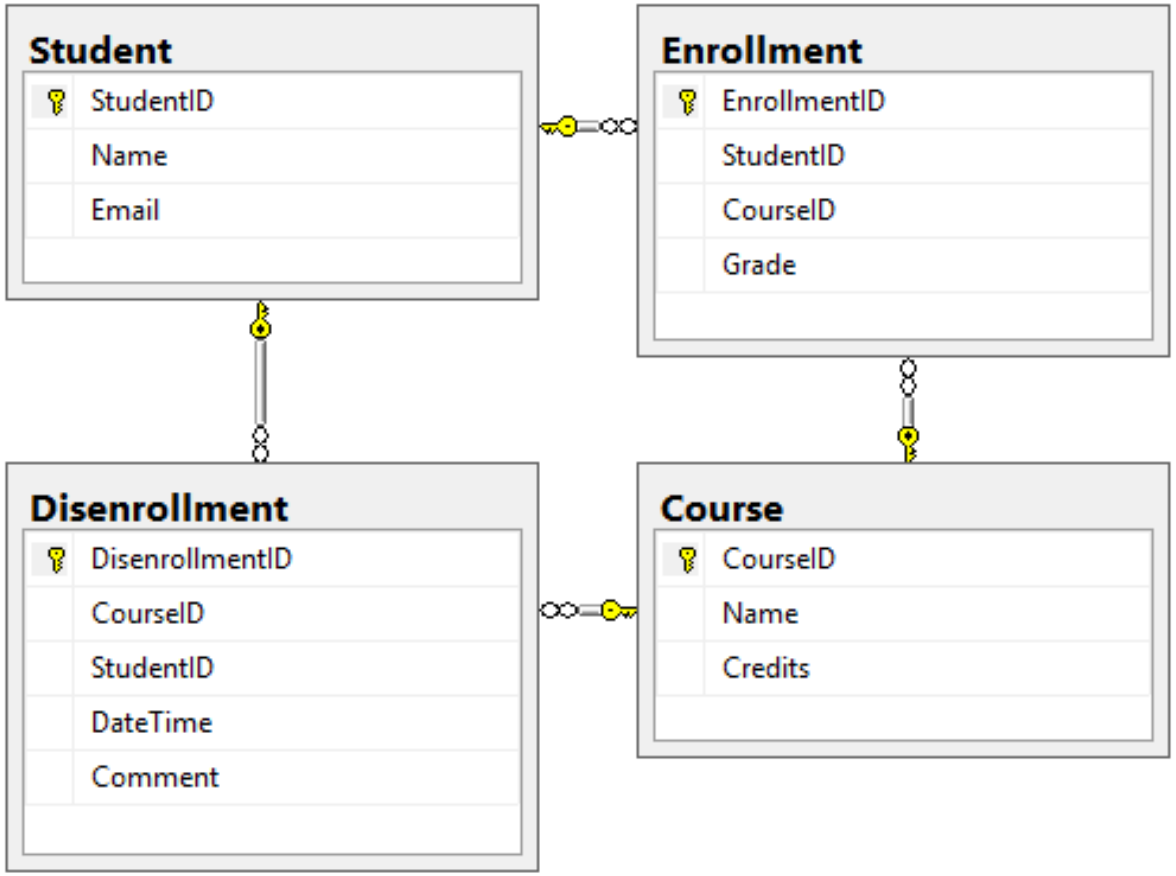
Replicas have the same structure

Designing a Database for Queries

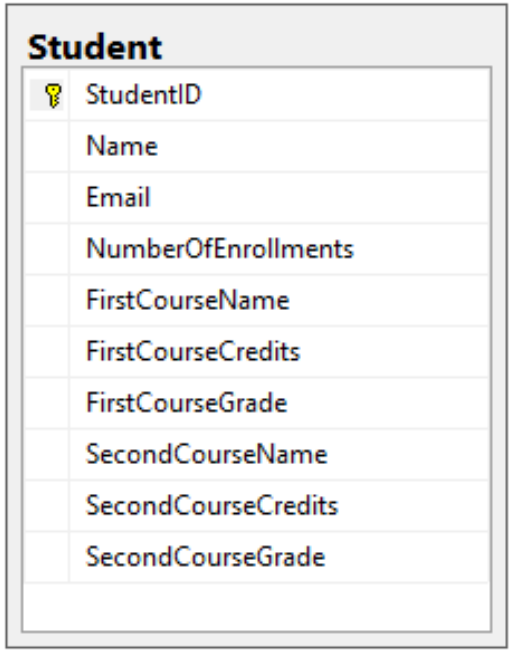
```sql
SELECT s.*, e.Grade, c.Name CourseName, c.Credits
FROM dbo.Student s
LEFT JOIN (
    SELECT e.StudentID, COUNT(*) Number
    FROM dbo.Enrollment e
    GROUP BY e.StudentID) t ON s.StudentID = t.StudentID
LEFT JOIN dbo.Enrollment e ON e.StudentID = s.StudentID
LEFT JOIN dbo.Course c ON e.CourseID = c.CourseID
WHERE (c.Name = @Course OR @Course IS NULL)
    AND (ISNULL(t.Number, 0) = @Number OR @Number IS NULL)
ORDER BY s.StudentID ASC
```

Designing a Database for Queries

Commands



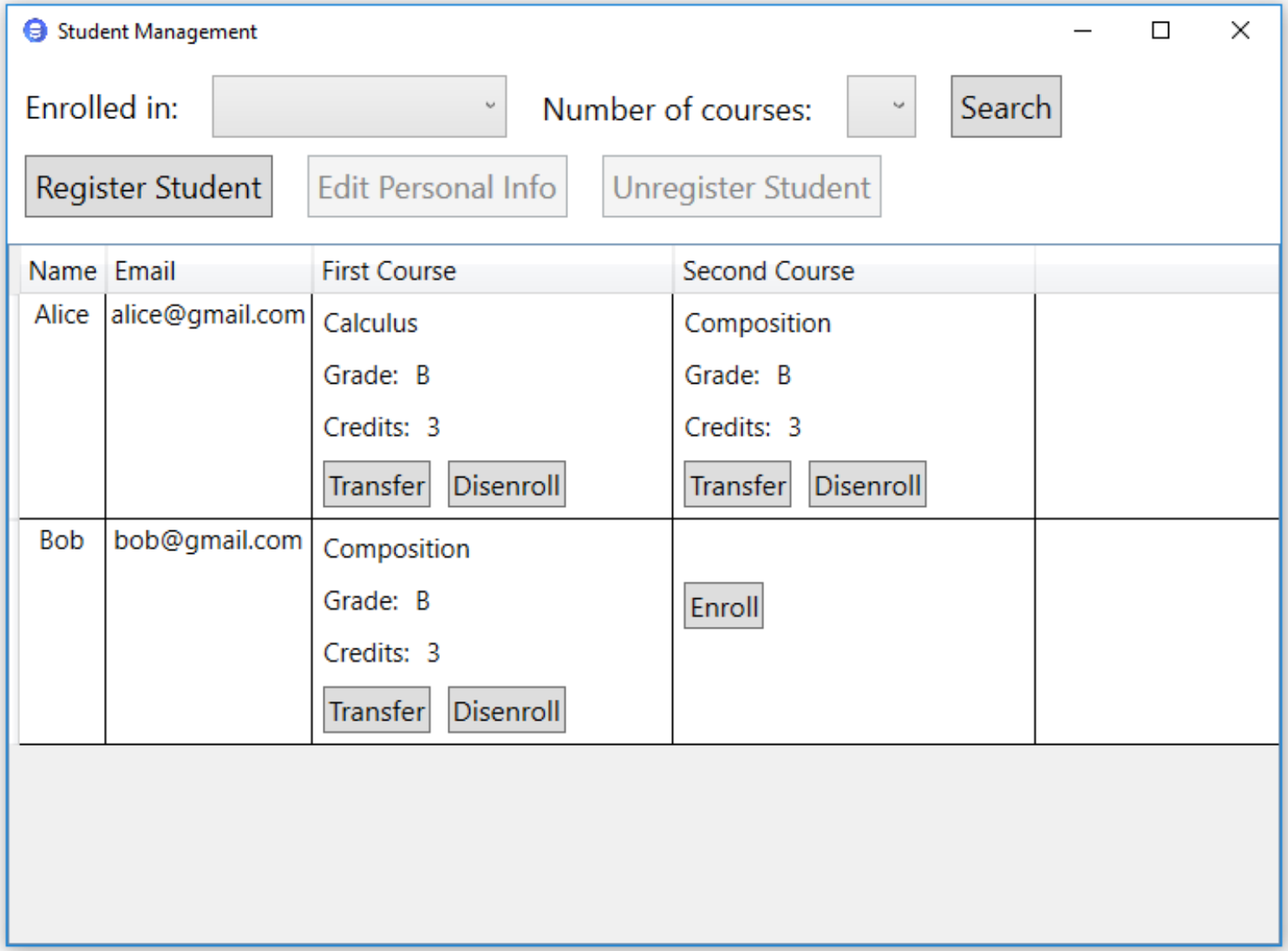
Queries

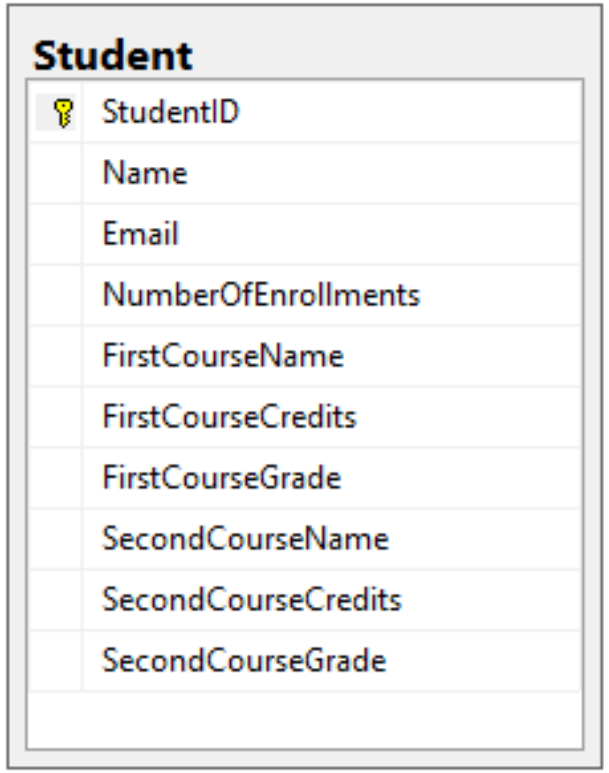


Designing a Database for Queries

Why denormalize the queries database?

Designing a Database for Queries





Designing a Database for Queries

```sql
SELECT s.*, e.Grade, c.Name CourseName, c.Credits
FROM dbo.Student s
LEFT JOIN (
    SELECT e.StudentID, COUNT(*) Number
    FROM dbo.Enrollment e
    GROUP BY e.StudentID) t ON s.StudentID = t.StudentID
LEFT JOIN dbo.Enrollment e ON e.StudentID = s.StudentID
LEFT JOIN dbo.Course c ON e.CourseID = c.CourseID
WHERE (c.Name = @Course OR @Course IS NULL)
    AND (ISNULL(t.Number, 0) = @Number OR @Number IS NULL)
ORDER BY s.StudentID ASC
```

```sql
SELECT s.*
FROM dbo.Student s
WHERE (s.FirstCourseName = @Course
        OR s.SecondCourseName = @Course OR @Course IS NULL)
    AND (s.NumberOfEnrollments = @Number OR @Number IS NULL)
ORDER BY s.StudentID ASC
```

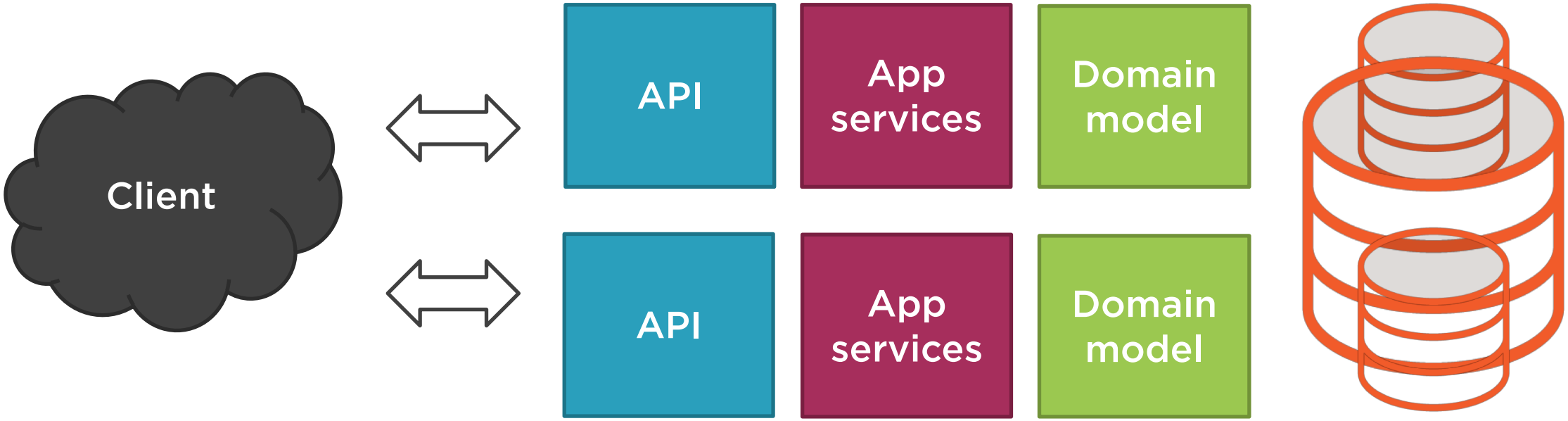
Recap: Creating a Database for Queries

Created a separate database for queries and re-targeted the query handler

Recap: Creating a Database for Queries

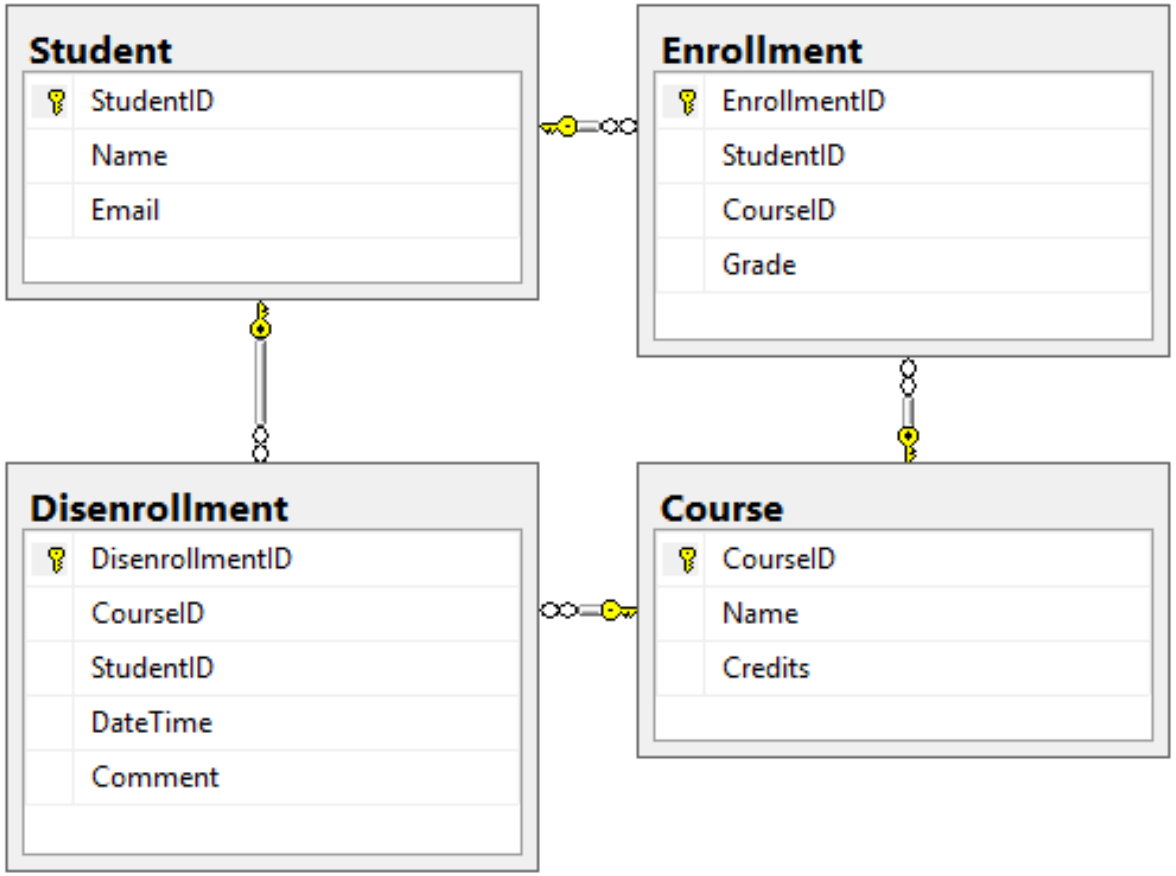
```
public List<StudentDto> Handle(GetListQuery query) {
    string sql = @"
        SELECT *
        FROM dbo.Student s
        WHERE (s.FirstCourseName = @Course
                OR s.SecondCourseName = @Course
                OR @Course IS NULL)
            AND (s.NumberOfEnrollments = @Number
                OR @Number IS NULL)
        ORDER BY s.StudentID ASC";

    using (SqlConnection connection = new SqlConnection(_connectionString.Value)) {
        List<StudentDto> students = connection
            .Query<StudentDto>(sql, new {
                Course = query.EnrolledIn,
                Number = query.NumberOfCourses
            })
            .ToList();

        return students;
    }
}
```

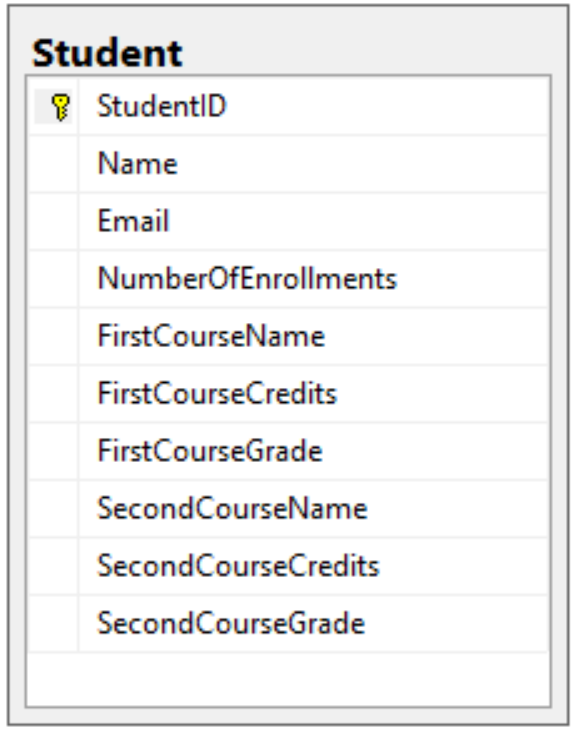
Recap: Creating a Database for Queries



Applied different architectural approaches to the read and write models

Recap: Creating a Database for Queries



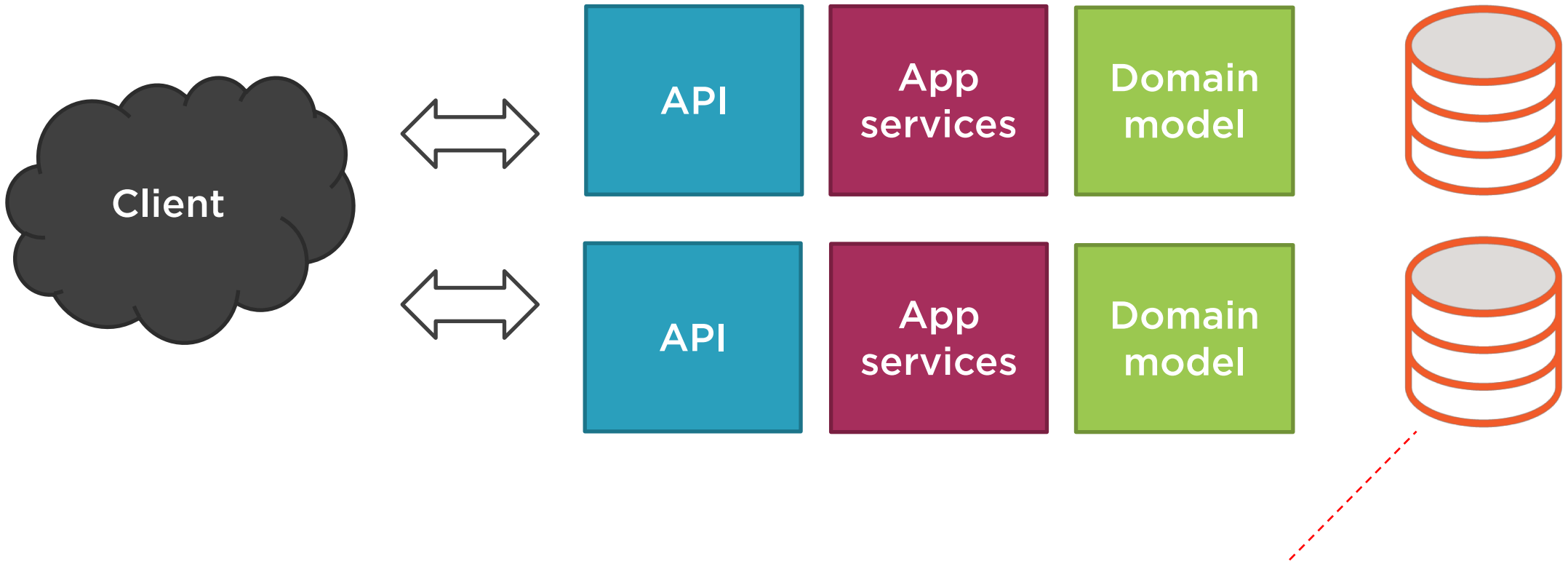


3rd normal form

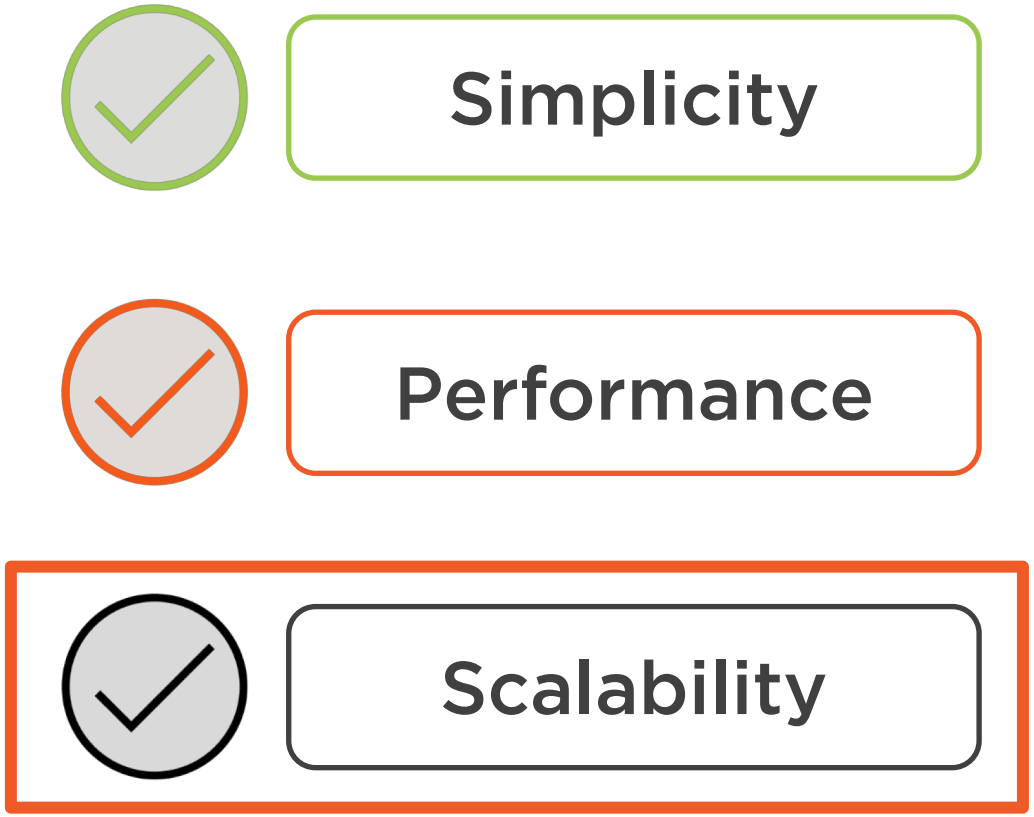
1st normal form

High normal forms are good for commands; low normal forms are good for queries.

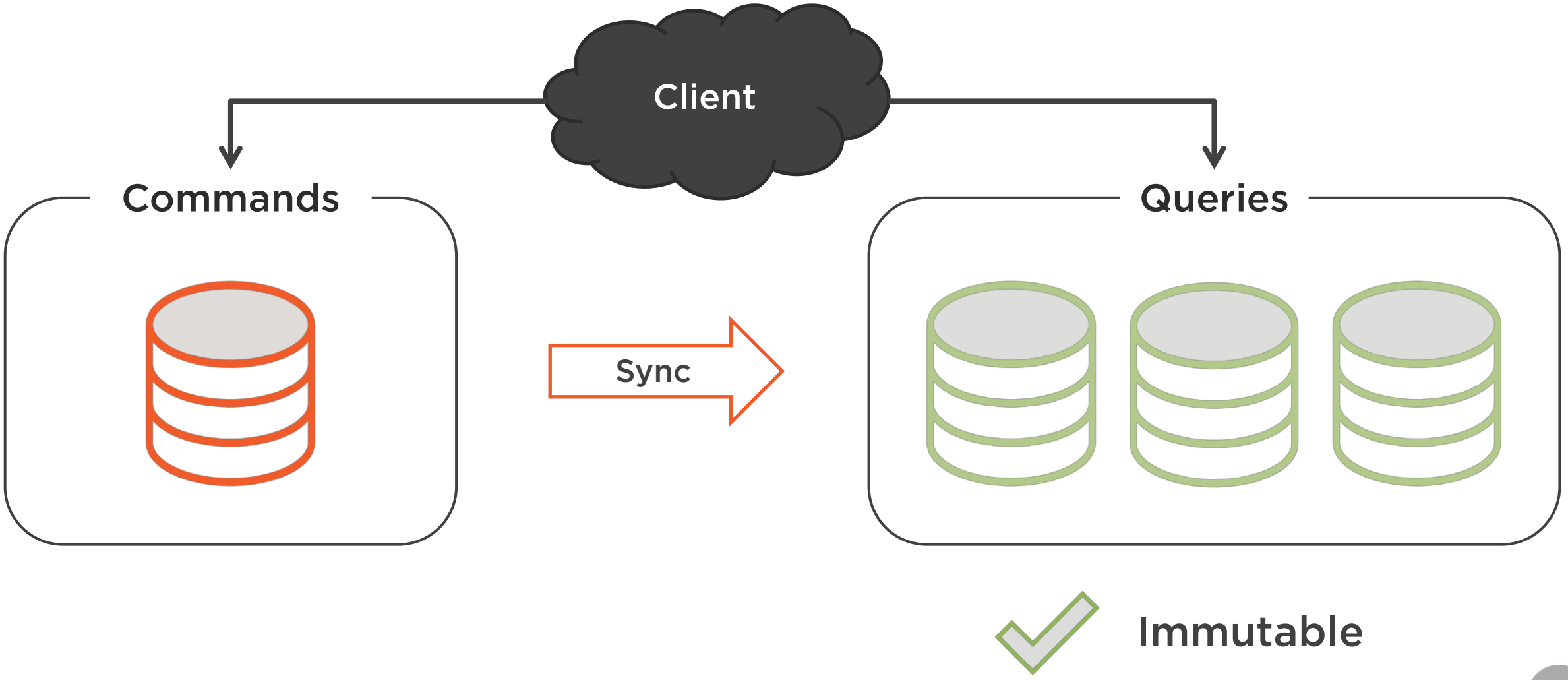
Recap: Creating a Database for Queries



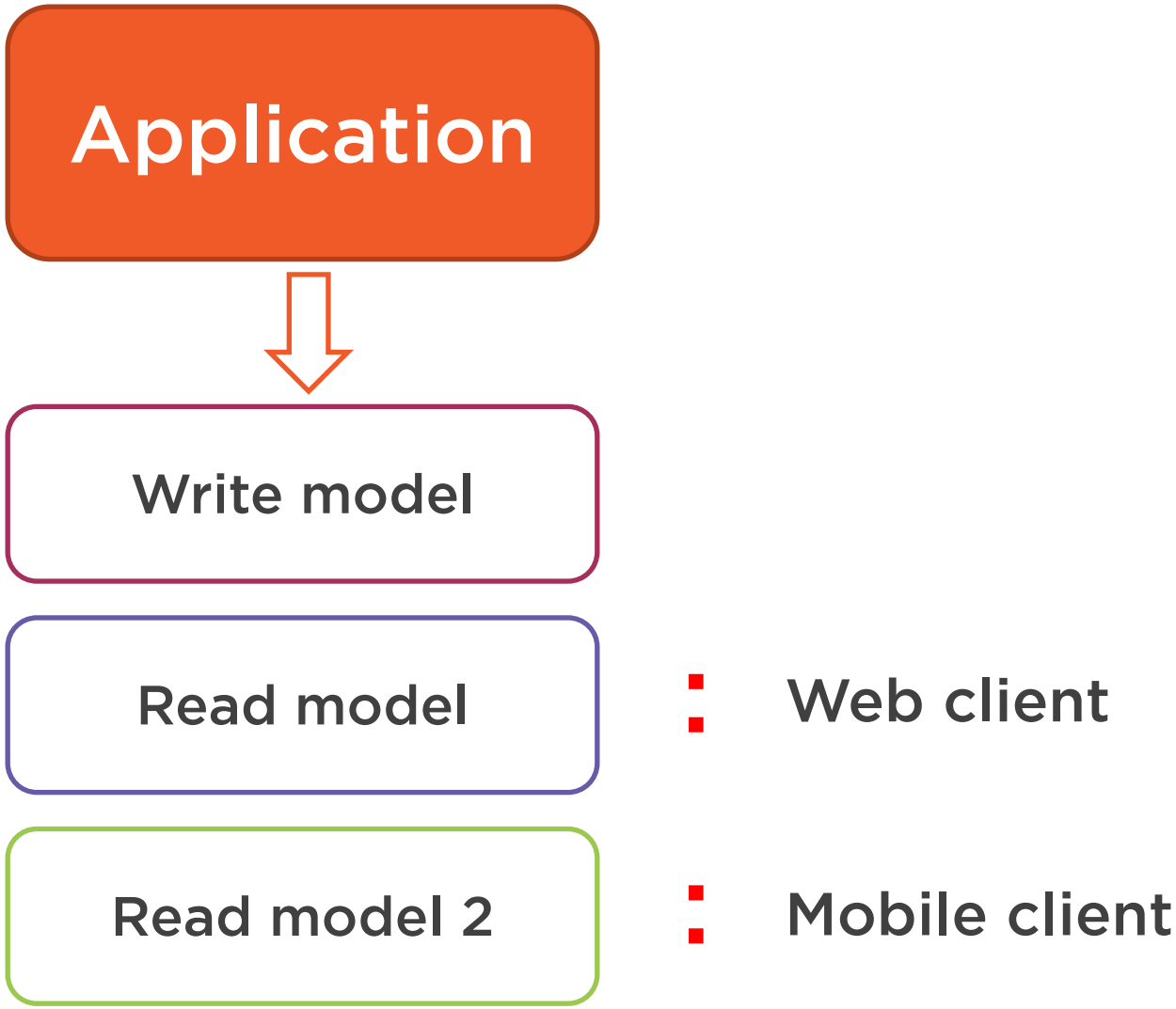
Scalability



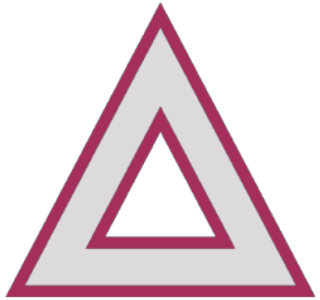
Scalability



Scalability

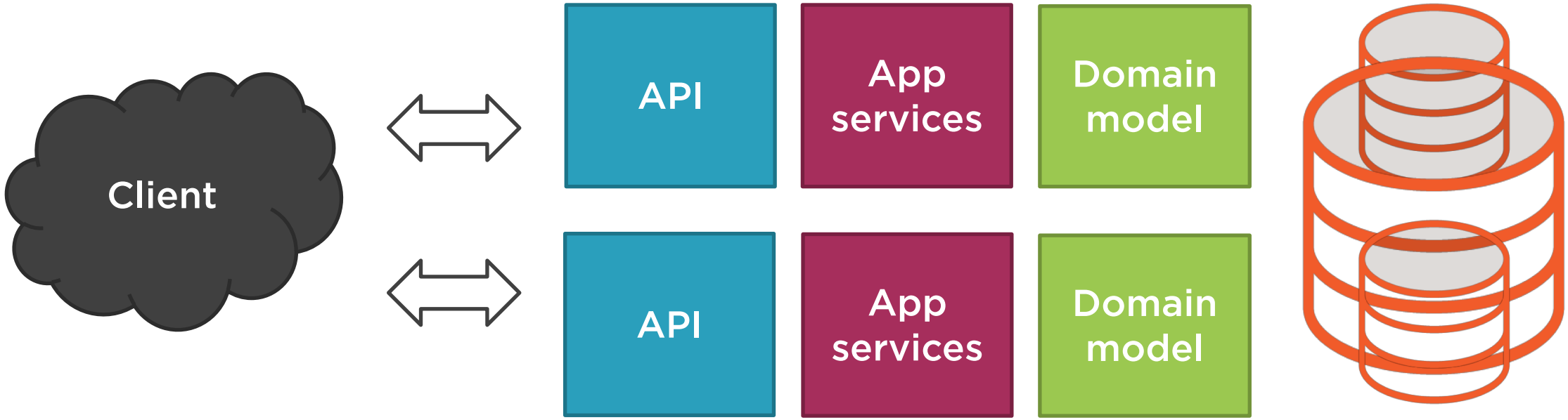


A Word of Caution Regarding the Database for Reads



Be prudent when applying the CQRS pattern

A Word of Caution Regarding the Database for Reads



- ✓ Maintainability costs are manageable
- ✗ Synchronization introduces a lot of complexity
- ✗ Eventual consistency is confusing for users

A Word of Caution Regarding the Database for Reads

Eventual consistency and maintaining a separate database are significant costs

- In most cases, you are just fine without a separate database for reads
- CQRS can be just as effective with a only a single database.

Summary

Introduced a separate database for queries

- Completed implementing the CQRS pattern
- Reads and writes are separated at each level: API, app services, domain model, DB
- Adjusted the read database for the needs of the query model
- Can scale the reads indefinitely

Scalability: utilizing the resources of more than one server

Examples of the separation at the data level:

- Indexed views
- Database replication
- ElasticSearch

Summary

Designed the database for reads

- Denormalized and thus adjusted it to the needs of the read model
- Minimized the number of joins and the amount of post-processing

The 3rd normal relational form is for commands; the 1st form for queries

Might need a separate read database for each client

Maintaining the synchronization is costly; eventual consistency is confusing

- In many cases, a single database is enough

In the Next Module

Synchronizing the commands and queries databases