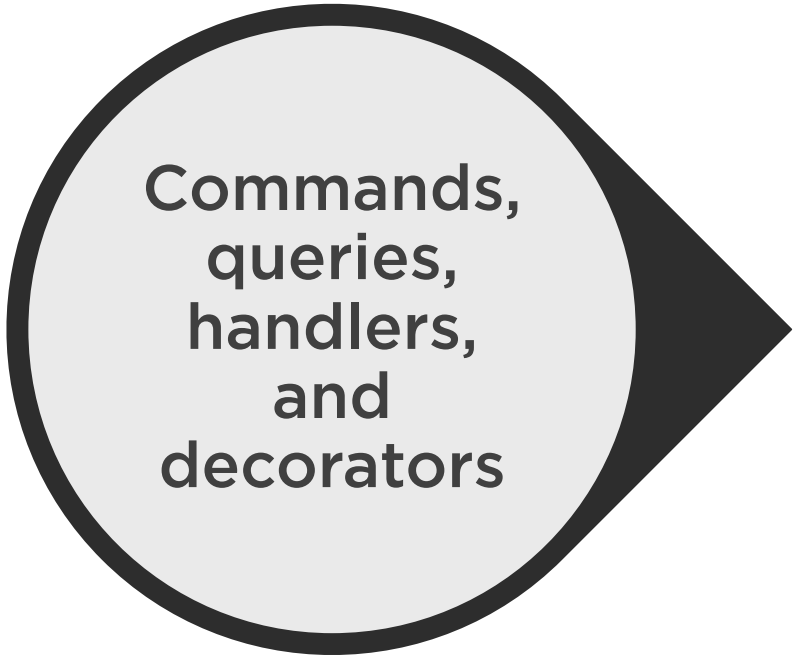
# Simplifying the Read Model

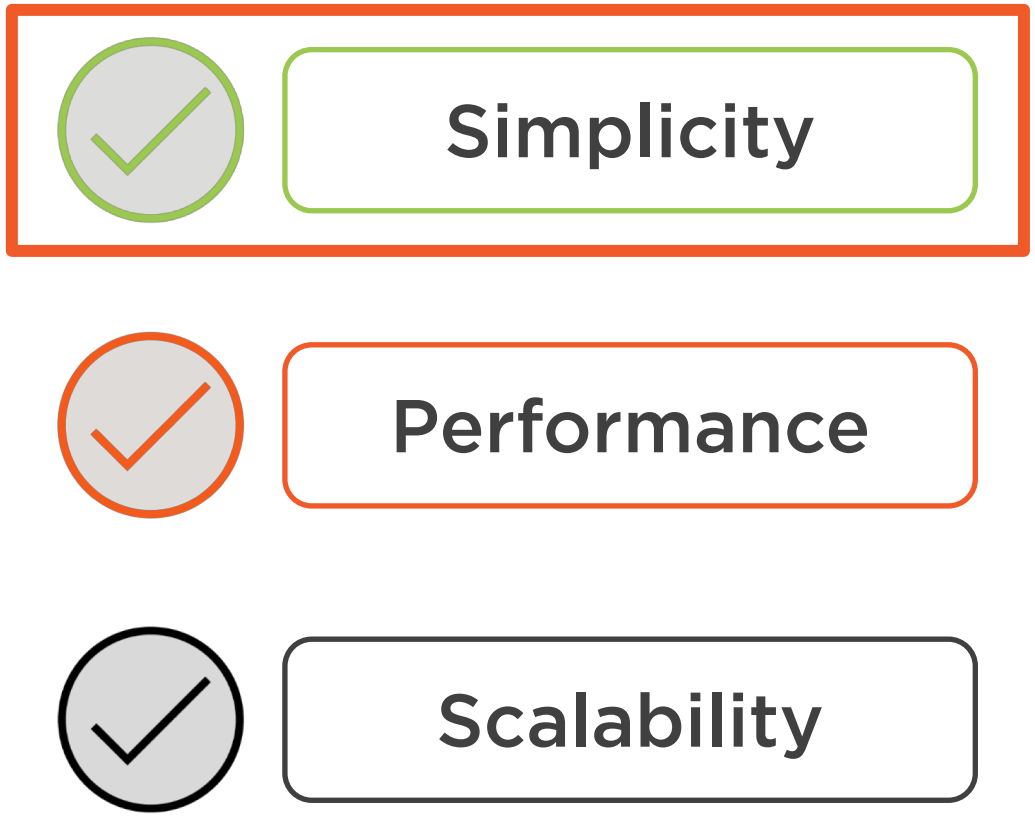

**Vladimir Khorikov**

@vkhorikov www.enterprisecraftsmanship.com

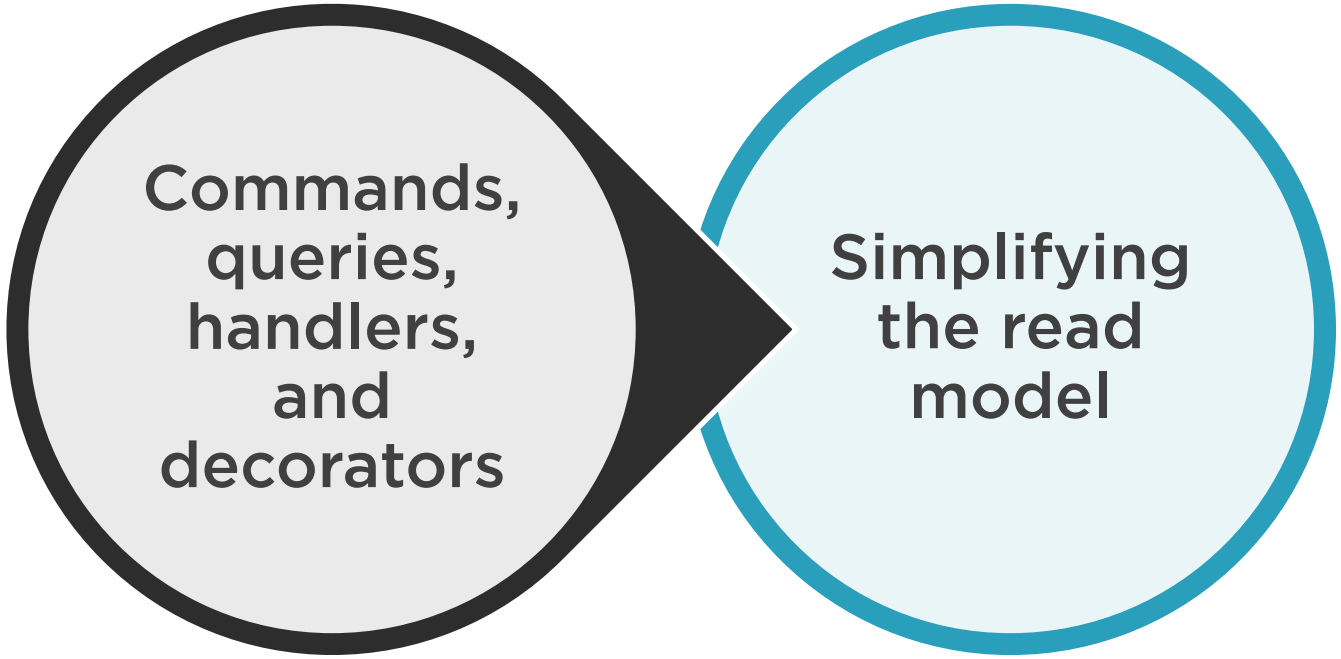
# Agenda



# Agenda



# Agenda


Commands, queries, handlers, and decorators
Simplifying the read model

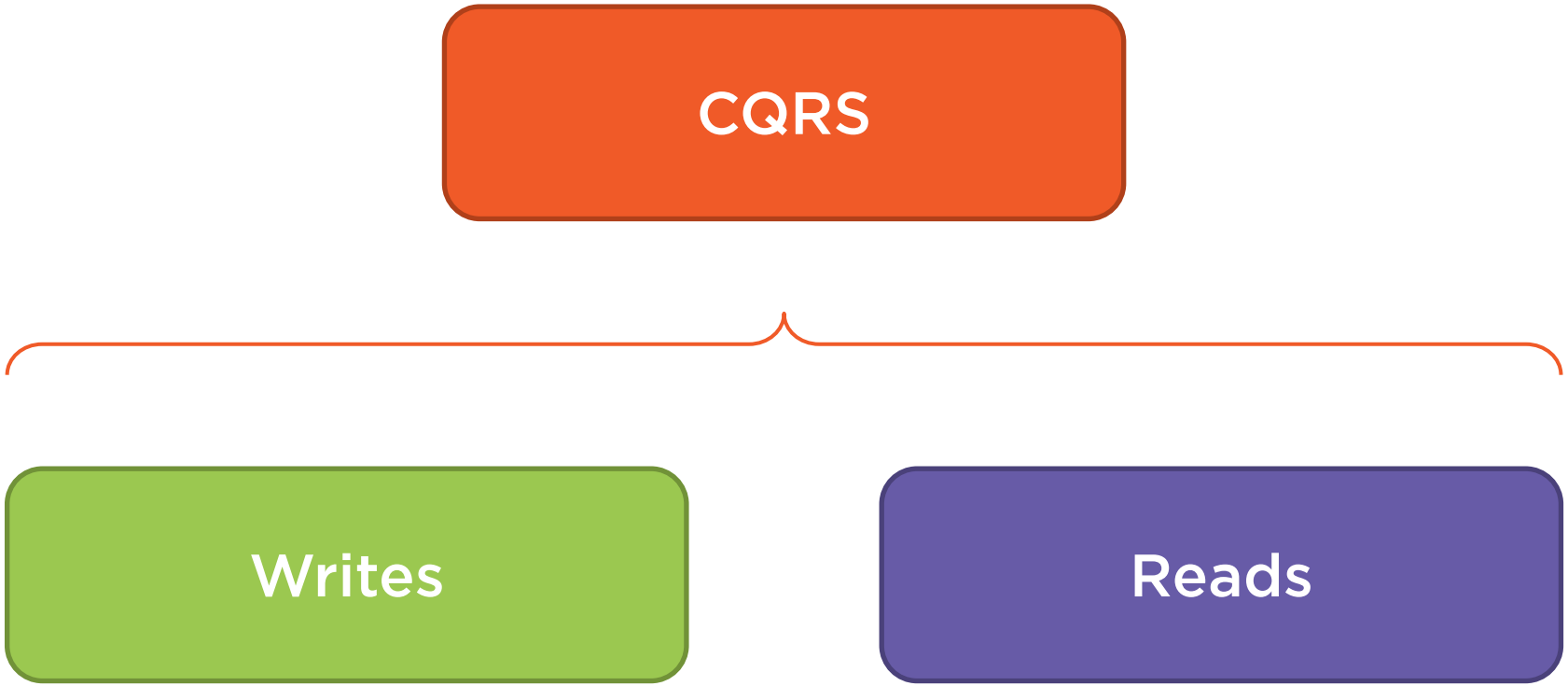
# The State of the Read Model

| Command model | Query model |
| --- | --- |
| Write model | Read model |
| Write side | Read side |
| Command side | Query side |
| Writes | Reads |

# Separation of the Domain Model

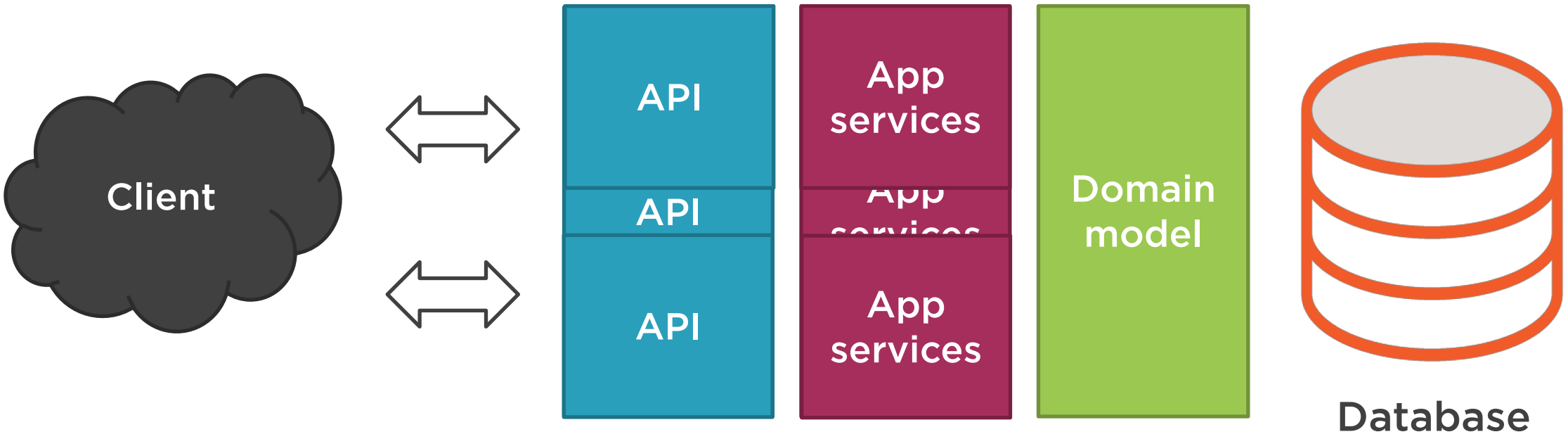
**How to fix the performance issues?**

# Separation of the Domain Model



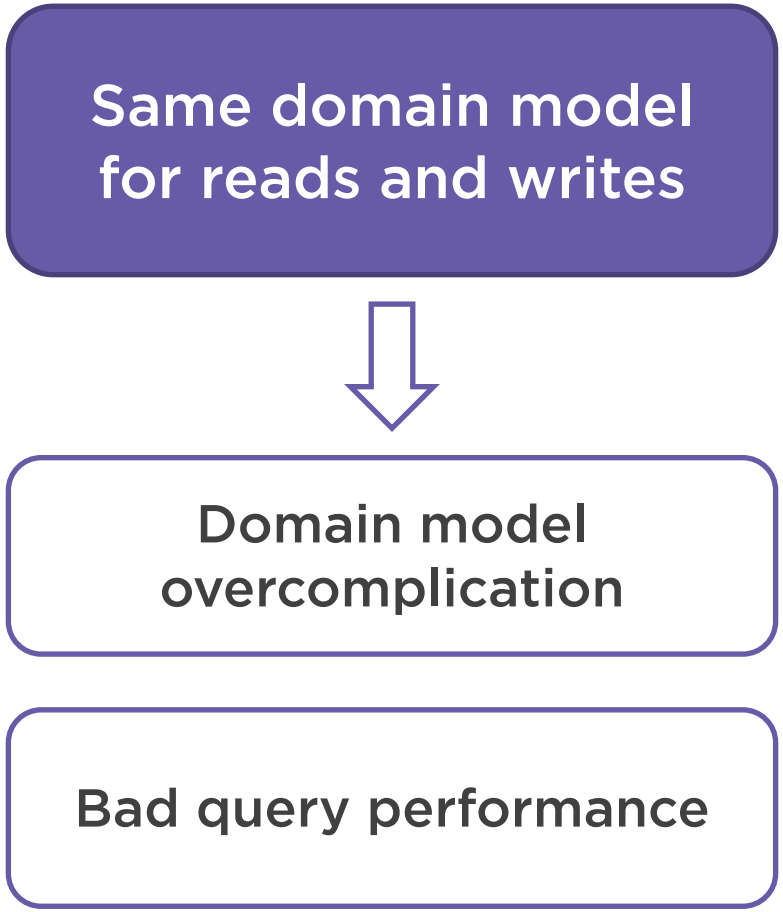

**Optimizing decisions for different situations**

# Separation of the Domain Model

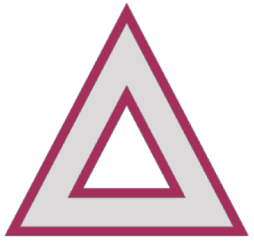



- Split the Update API
- Introduced explicit command and query handlers

# Separation of the Domain Model



# Separation of the Domain Model



**A complex domain model that handles neither reads nor writes well**

**Make the difference between reads and writes explicit**

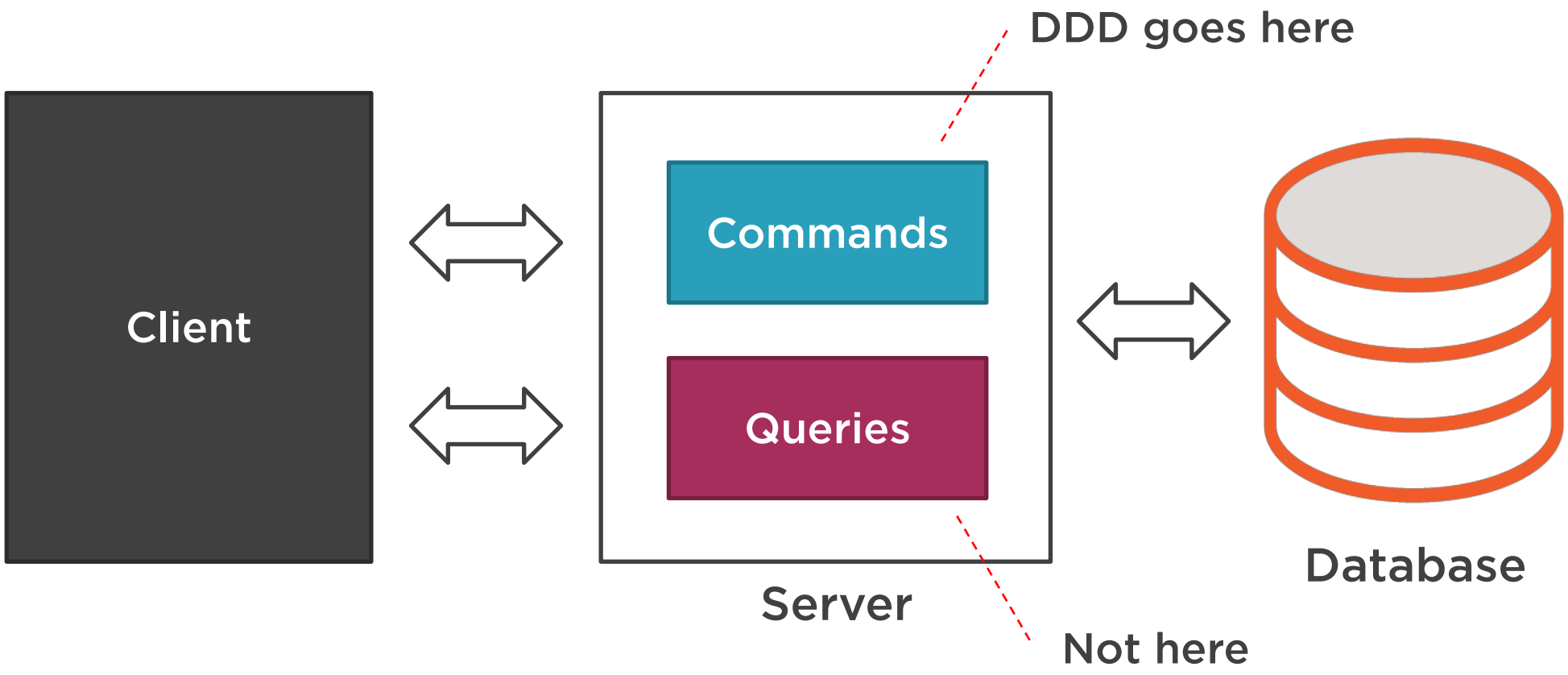
**Split the domain model**

# Separation of the Domain Model

**Are we going to have two domain models now?**

**Take the domain model out of the read side**

# Separation of the Domain Model




No data modifications = { No need in encapsulation; No need in abstractions

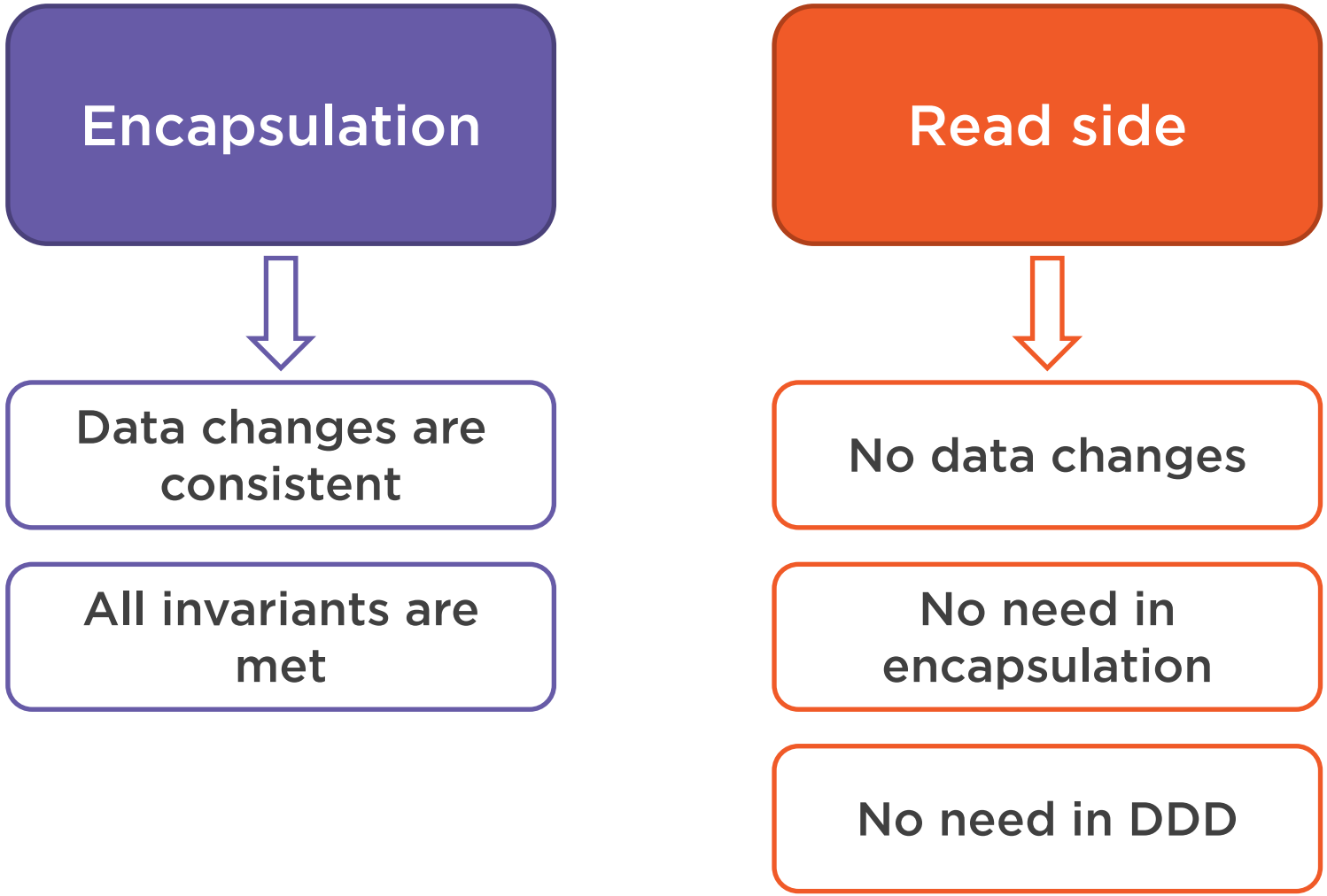
# Recap: Simplifying the Read Model

**Simplified the read side**

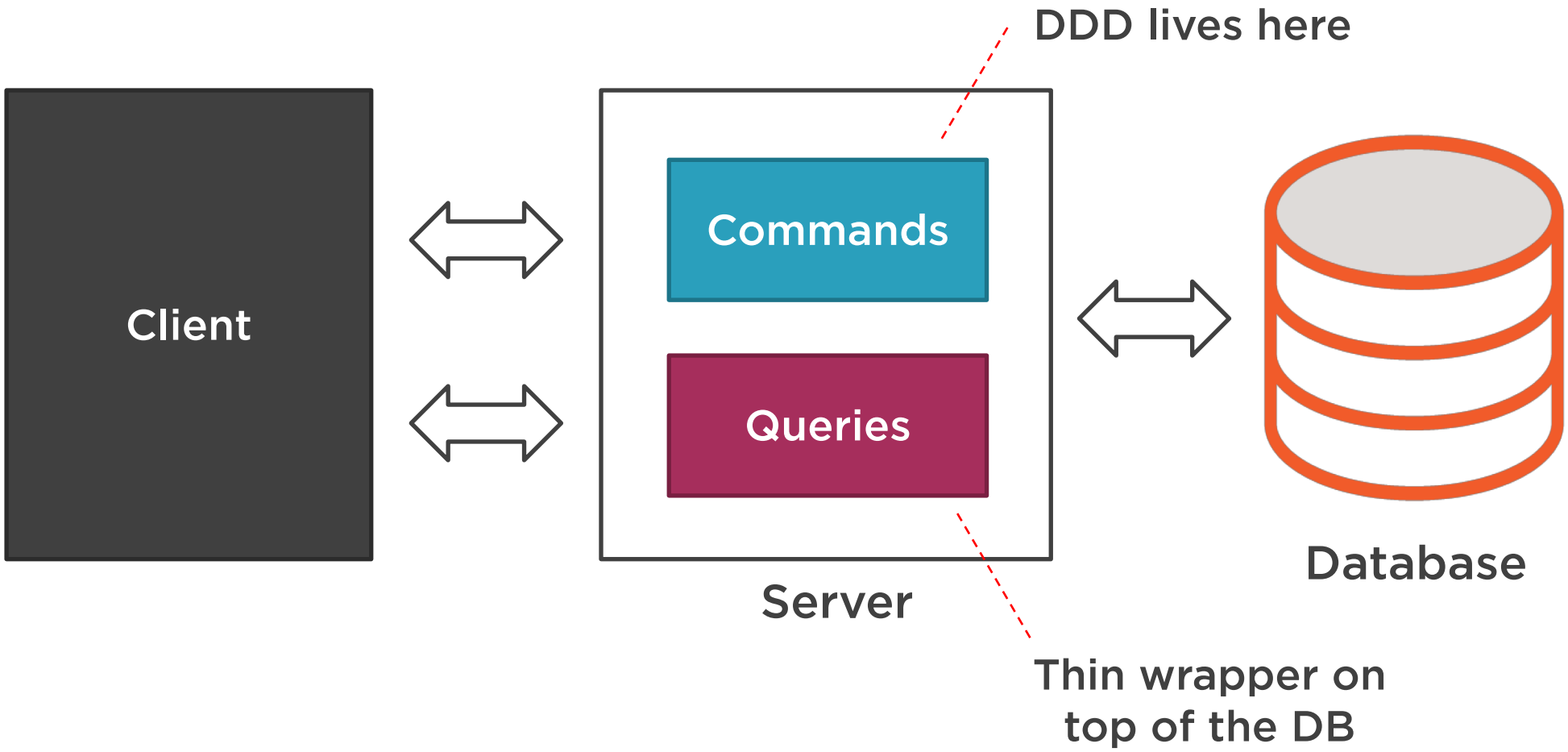
**Read model is a thin wrapper on top of the database**

**Can use database-specific features to optimize the performance**

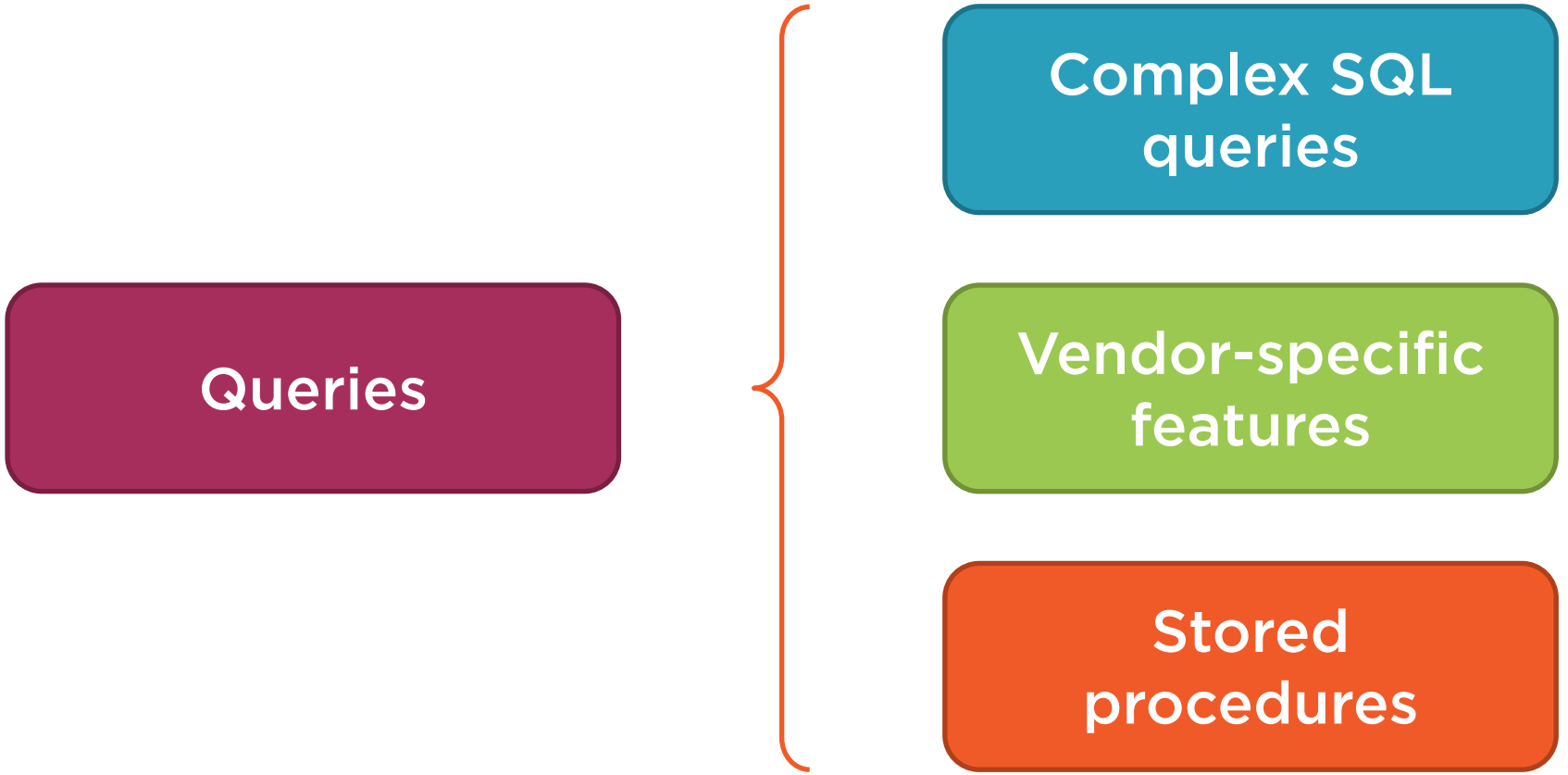
# Recap: Simplifying the Read Model



# Recap: Simplifying the Read Model



# Recap: Simplifying the Read Model



# Recap: Simplifying the Read Model

**Doesn't it make the read model anemic?**

**There's no need in encapsulation if you don't modify any data**

# Refactoring from Anemic Domain Model Towards a Rich One

by Vladimir Khorikov

Building bullet-proof business line applications is a complex task. This course will teach you an in-depth guideline into refactoring from Anemic Domain Model into a rich, highly encapsulated one.

Resume Course | Bookmark | Add to Channel | Live mentoring

Table of contents | Description | Transcript | Exercise files | Discussion | Learning Check | Recommended

Expand all

| | | | |
|---|---|---|---|
| Course Overview | ✓ | 1m 31s | |
| Introduction | ✓ | 22m 24s | |
| Introducing an Anemic Domain Model | | 18m 31s | |
| Decoupling the Domain Model from Data Contracts | | 29m 46s | |

## Course author

Vladimir Khorikov

Vladimir Khorikov is a Microsoft MVP and has been professionally involved in software development for more than 10 years.

## Course info

| | |
|---|---|
| Level | Intermediate |
| Rating | ★★★★★ (73) |
| My rating | ★★★★★ |
| Duration | 3h 36m |
| Released | 13 Nov 2017 |

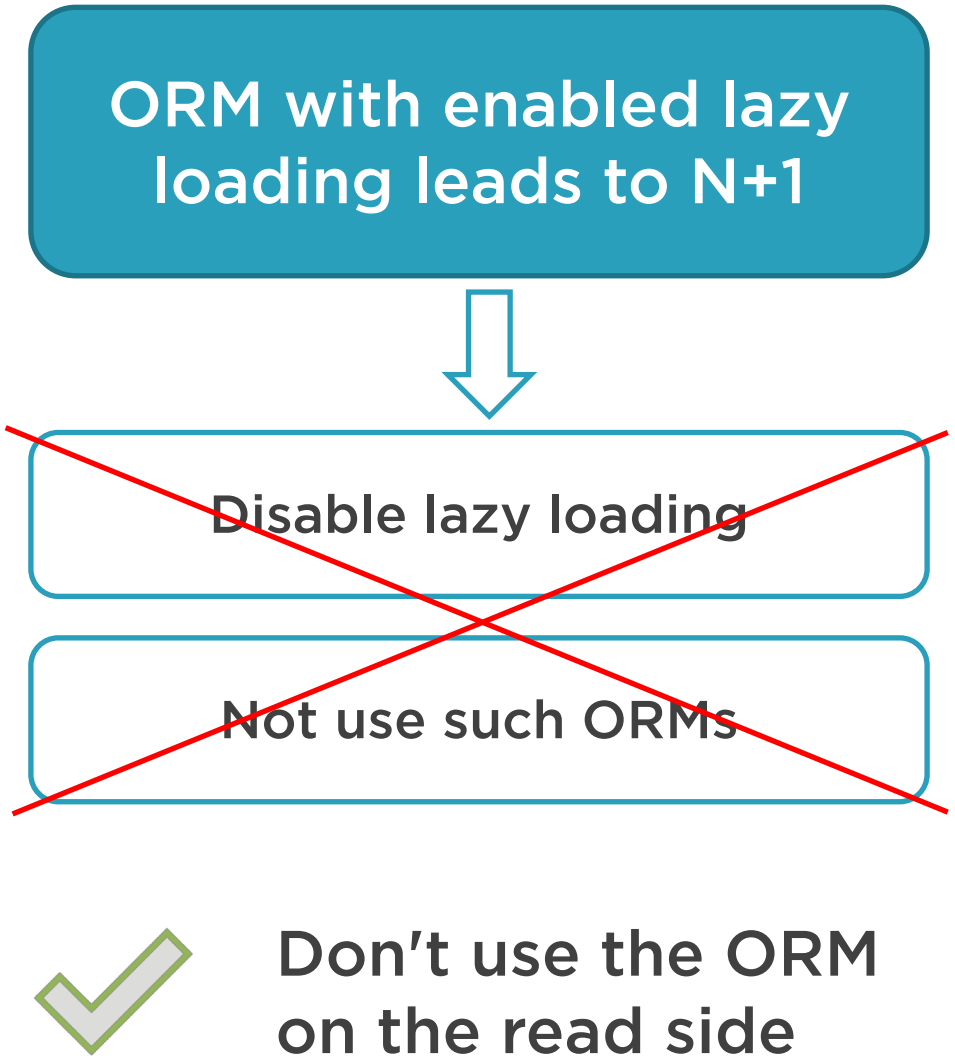
## Share course

# Recap: Simplifying the Read Model

Optimized the data retrieval

Got rid of the N+1 problem

# Recap: Simplifying the Read Model




Don't use the ORM on the read side

CQRS allows you to optimize, read and write models for the different requirements.

# Recap: Simplifying the Read Model

```
public class Student : Entity
{
    public virtual string Name { get; set; }
    public virtual string Email { get; set; }

    private readonly IList<Enrollment> _enrollments = new List<Enrollment>();
    public virtual IReadOnlyList<Enrollment> Enrollments => _enrollments.ToList();
    public virtual Enrollment FirstEnrollment => GetEnrollment(0);
    public virtual Enrollment SecondEnrollment => GetEnrollment(1);
}
```

Simplified the commands side too

The domain model focuses on commands only

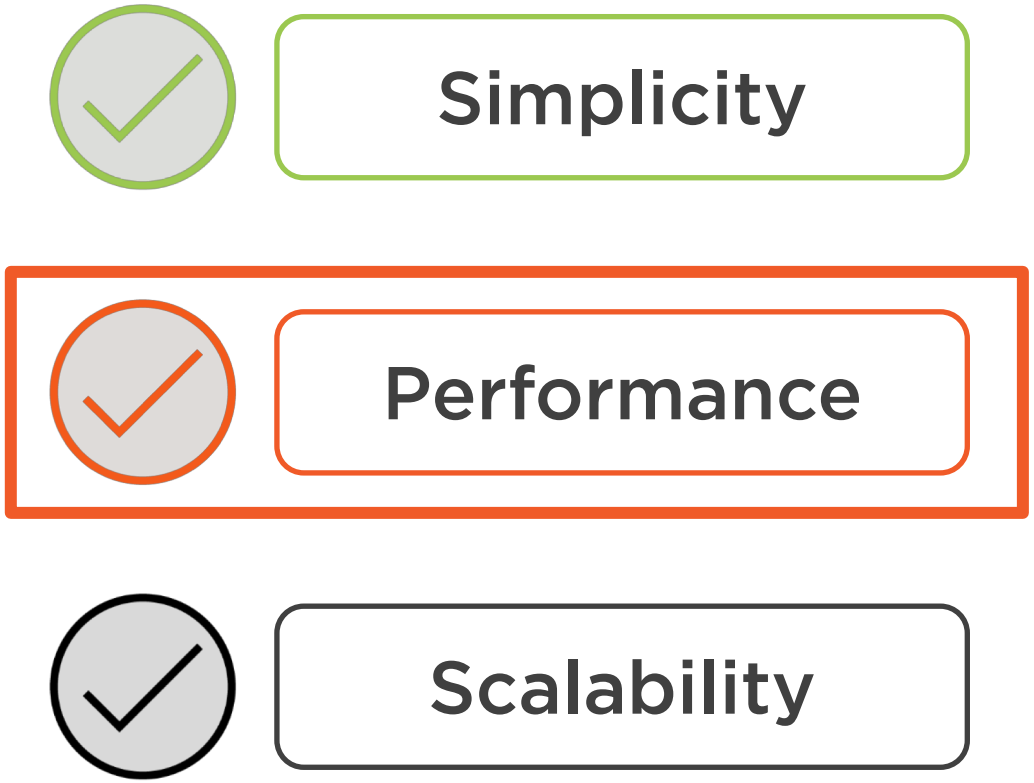
# Recap: Simplifying the Read Model

- It is impossible to create an optimal solution for searching, and processing of transactions utilizing a single model
- Both reads and writes benefit from the separation
- Writes benefit from removing code from the domain model that is not used for data modifications
- Reads benefit because you are able to optimize the data retrieval

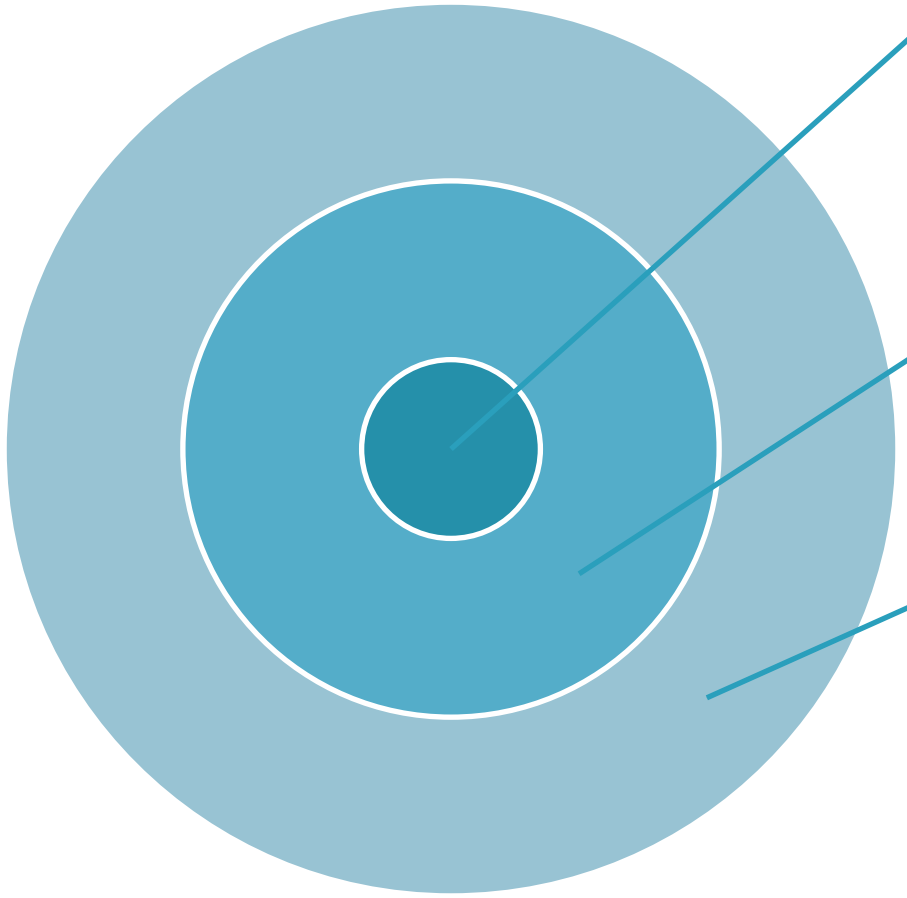
Use a “big” ORM in commands

Use handwritten SQL in queries

# Recap: Simplifying the Read Model



# The Read Model and the Onion Architecture



Entities
Aggregates
Value Objects
Domain Events
Pure Domain Services

+ Commands and queries

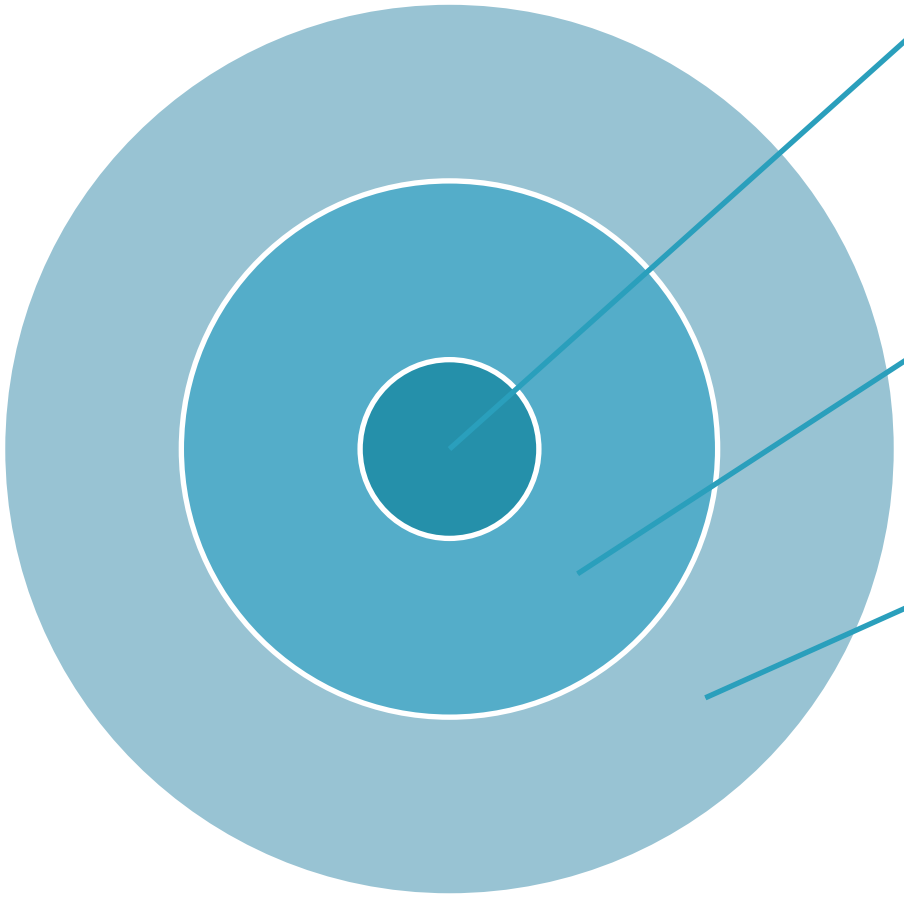
Repositories
Impure Domain Services

Application Services
UI

# The Read Model and the Onion Architecture

```
public sealed class GetListQuery : IQuery<List<StudentDto>>
{
    public string EnrolledIn { get; }
    public int? NumberOfCourses { get; }
}
```

# The Read Model and the Onion Architecture



Entities
Aggregates
Value Objects
Domain Events
Pure Domain Services

The query is here

Repositories
Impure Domain Services

Application Services
UI

DTOs go here

# The Read Model and the Onion Architecture

```
public sealed class GetListQuery : IQuery<List<StudentDto>>
{
    public string EnrolledIn { get; }
    public int? NumberOfCourses { get; }
}
```

```
public sealed class GetListQuery : IQuery<List<Student>>
{
    public string EnrolledIn { get; }
    public int? NumberOfCourses { get; }
}
```

**Queries no longer reside in the onion**

# Summary

**Simplifying the read model**

- It no longer uses the domain model
- Doesn't use NHibernate

**Introduced the separation at the domain model level**

- Simplified the command side
- Optimized the query side
- The domain model no longer contains code used by the queries
- Can even get rid of repositories
- Can use database-specific features in reads

**There's no need for encapsulation in the reads**

**The read model and the onion architecture**

- Queries are no longer part of the onion

# In the Next Module

## Introducing a Separate Database for Queries