TRAILHEAD
TECHNOLOGY PARTNERS

You've Been Organizing Your Code Wrong

A C# Developer's Intro to Vertical Slice Architecture

Jonathan “J.” Tower


"Add a date of birth field for all users"
TRAILHEAD
TECHNOLOGY PARTNERS

A Simple Edit

SQL

```
ALTER TABLE User ADD DateOfBirth DATE;
```

TRAILHEAD
TECHNOLOGY PARTNERS

A Simple Edit

User.cs

```
public class User
{
    public int Id { get; set; }
    public string Name { get; set; }
    public DateTime DateOfBirth { get; set; }
}
```

A Simple Edit

ApplicationDbContext.cs

```
modelBuilder.Entity<User>()
    .Property(u => u.DateOfBirth)
    .IsRequired(true);
```

A Simple Edit

UserModel.cs

```
public class UserModel
{
    public int Id { get; set; }
    public string Name { get; set; }
    public DateTime DateOfBirth { get; set; }
}
```

TRAILHEAD
TECHNOLOGY PARTNERS

A Simple Edit

UserService.cs

```
public void CreateUser(UserModel userModel)
{
    // Business logic involving DateOfBirth
}

public void UpdateUser(UserModel userModel)
{
    // Business logic involving DateOfBirth
}

…
```

A Simple Edit

UserDto.cs

```
public class UserDto
{
    public int Id { get; set; }
    public string Name { get; set; }
    public DateTime DateOfBirth { get; set; }
}
```

TRAILHEAD
TECHNOLOGY PARTNERS

A Simple Edit

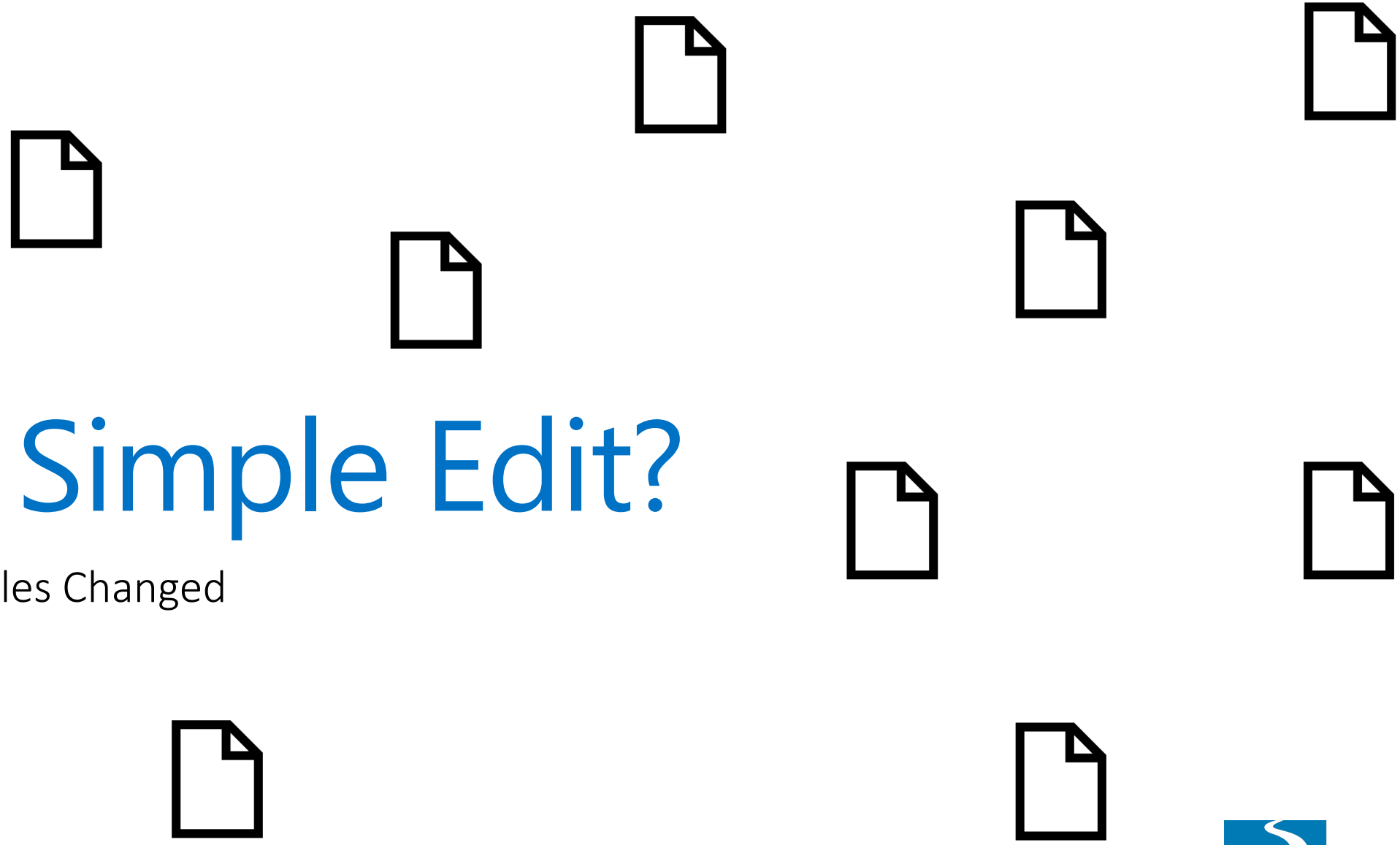
UserViewModel.cs

```
public class UserViewModel
{
    public string Name { get; set; }
    public DateTime DateOfBirth { get; set; }
}
```

A Simple Edit

```[html]
<form ...>
...
    <input type="date" asp-for="DateOfBirth" />
...
</form>
```

A Simple Edit



A Simple Edit?

8+ Files Changed

TRAILHEAD
TECHNOLOGY PARTNERS

Vertical Slice Architecture

The Problem

Previous Approaches

Vertical Slice Architecture

VSA Tools for .NET

Demo

FAQ

Jonathan "J." Tower

Principal Consultant & Partner

TRAILHEAD
TECHNOLOGY PARTNERS

Microsoft MVP in .NET

jtower@trailheadtechnology.com

trailheadtechnology.com/blog

jtowermi

jtower

FREE

CONSULTATION

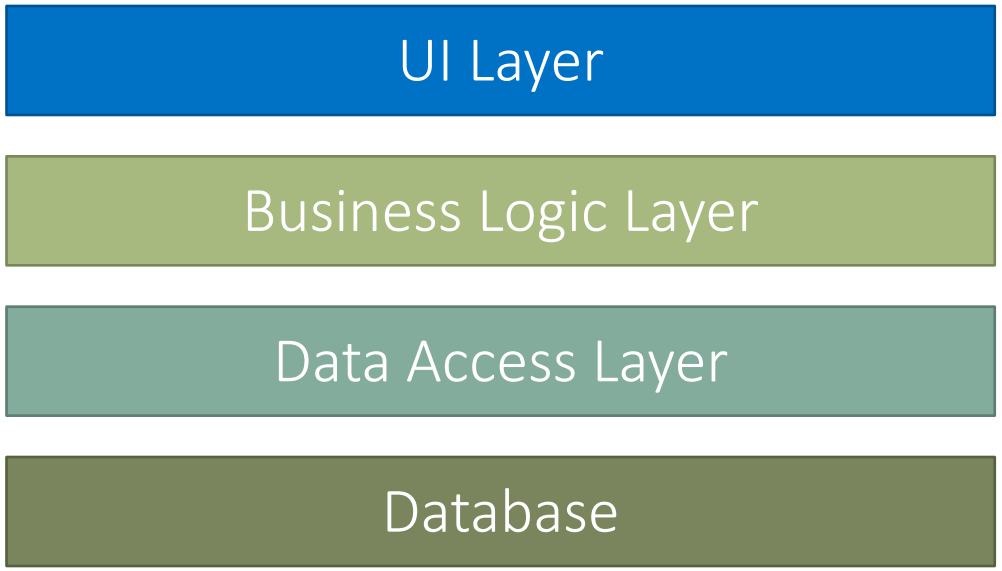
bit.ly/th-offer

github.com/trailheadtechnology/vsa-dotnet

Layered Approaches

How We've Tried Historically

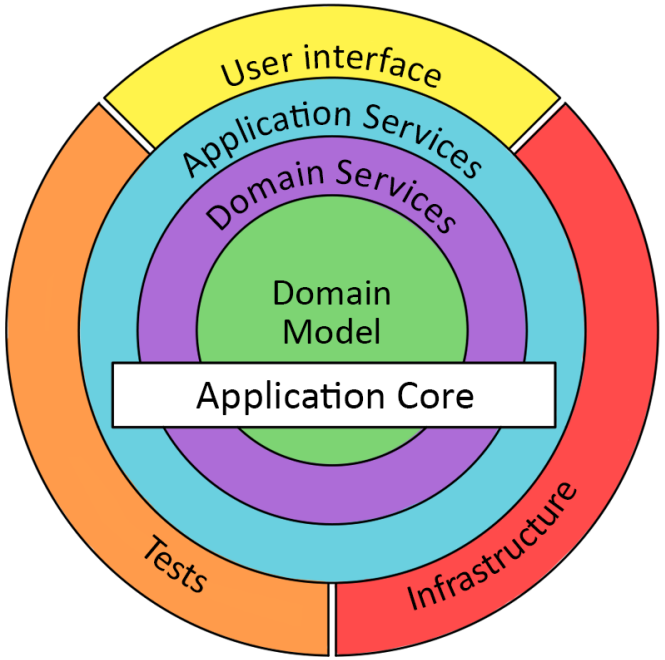
Layered Architectures



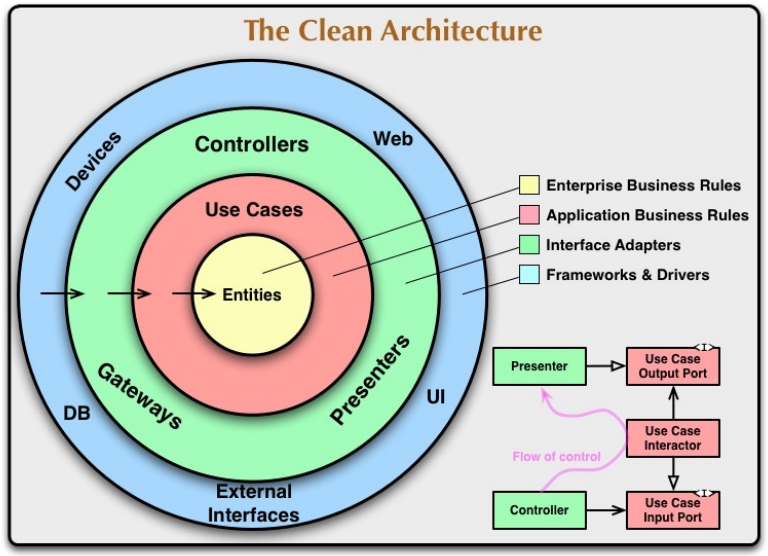
N-Tier

DDD

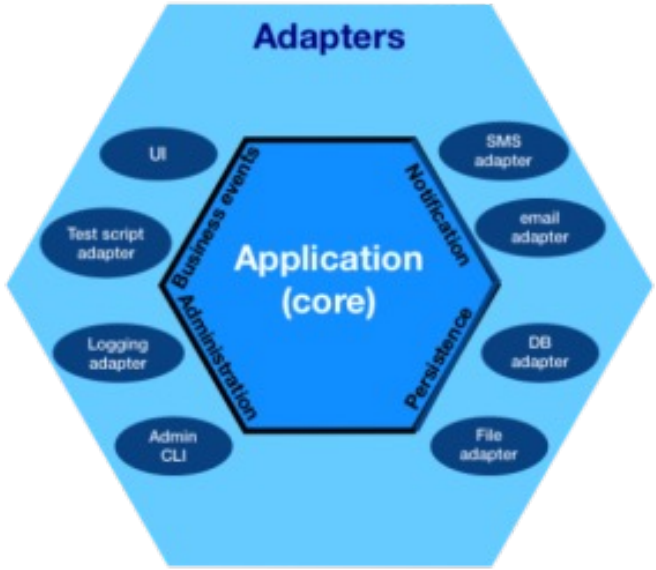
Layered Architectures



Onion

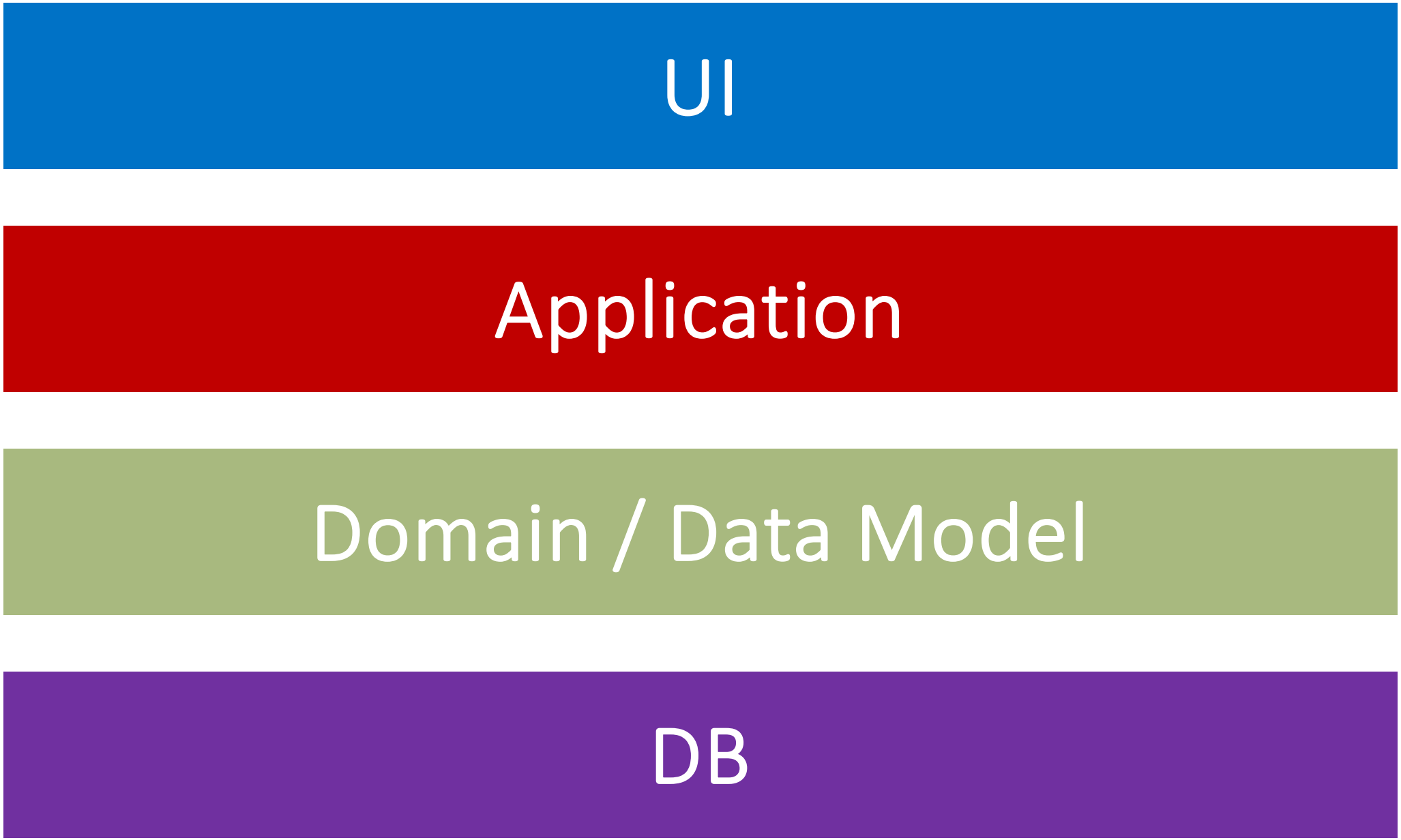


Clean



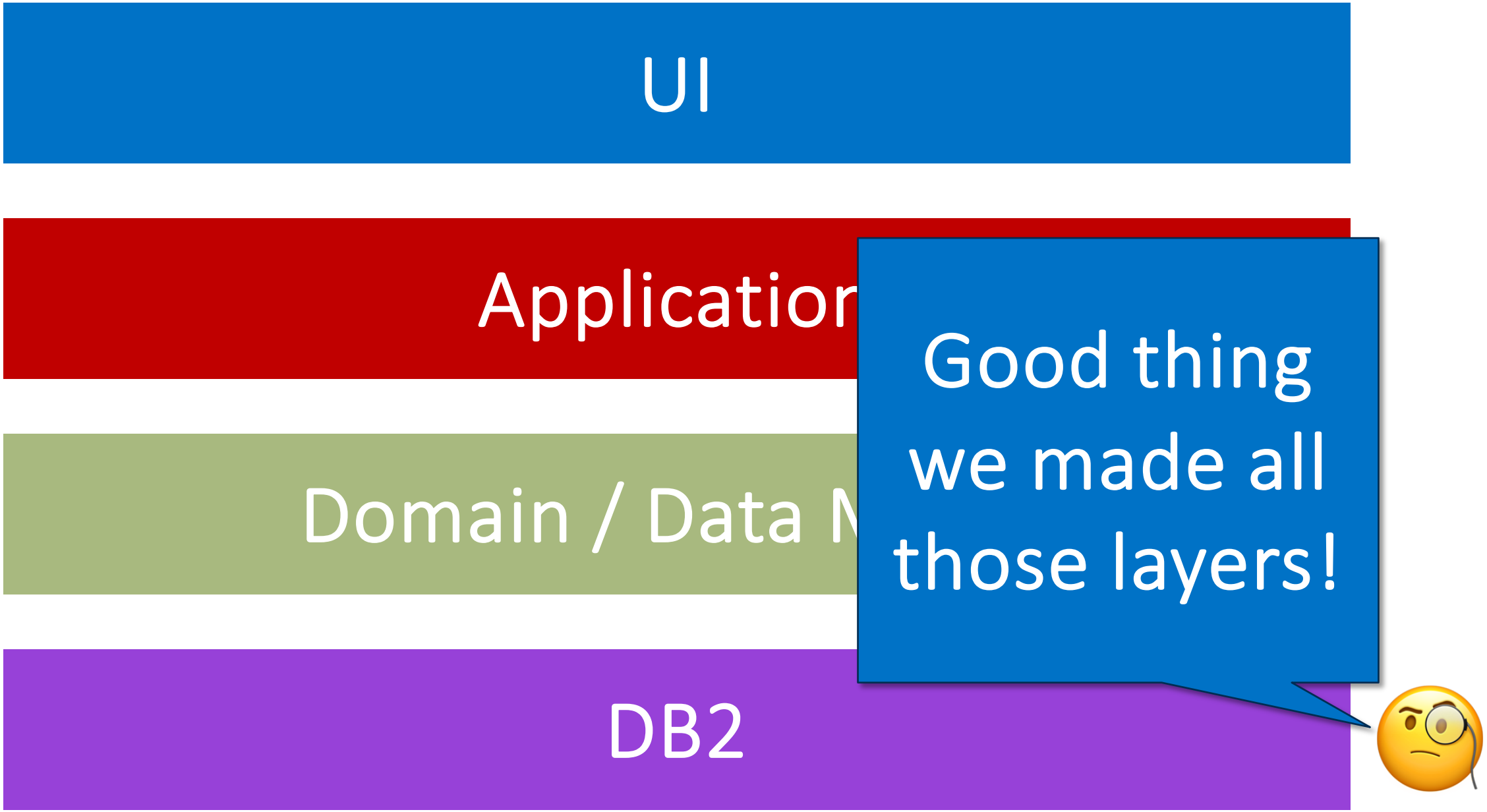
Hexagonal

How Software Typically Changes


UI
Application
Domain / Data Model
DB

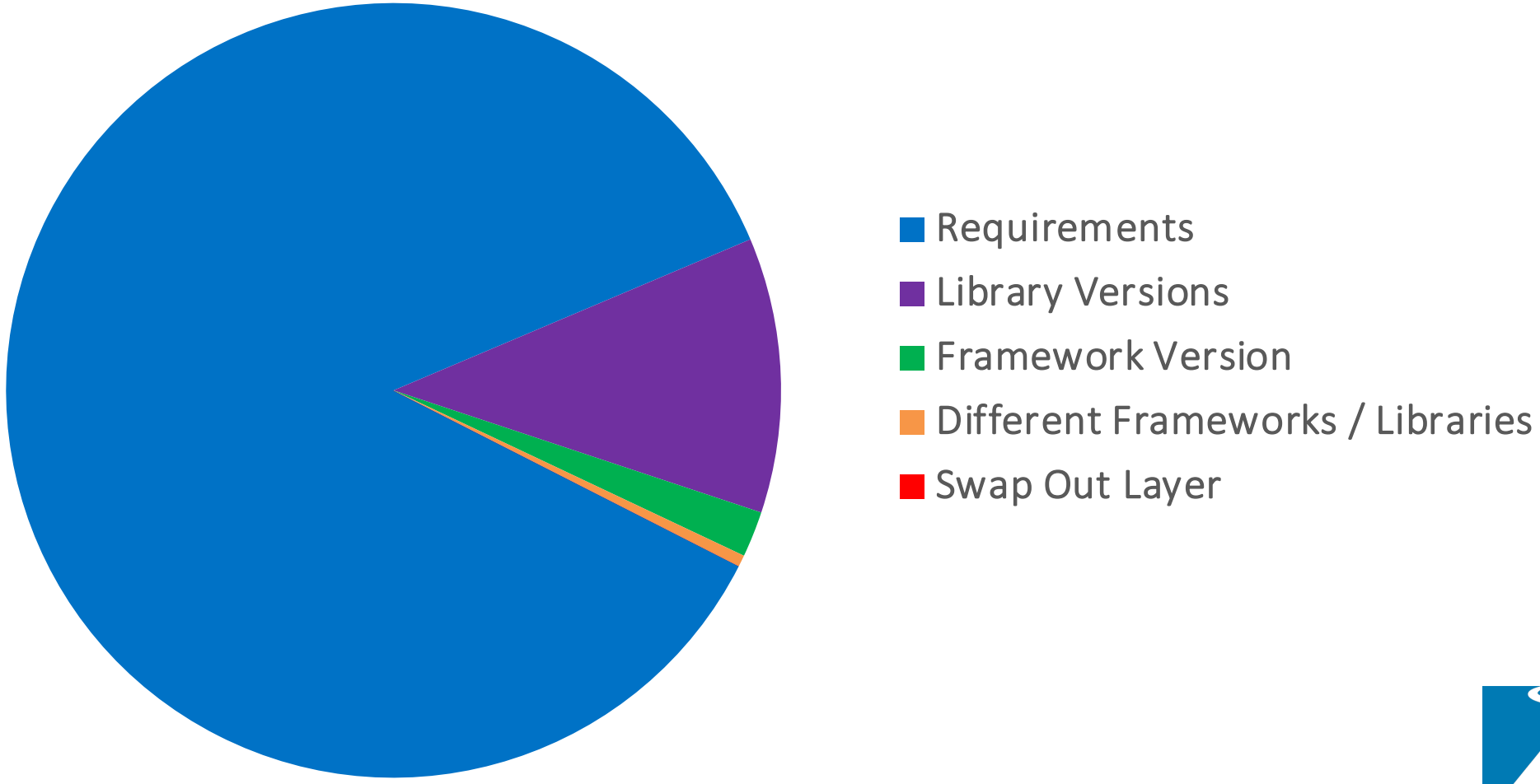
"Change the Data Access Layer"

TRAILHEAD
TECHNOLOGY PARTNERS

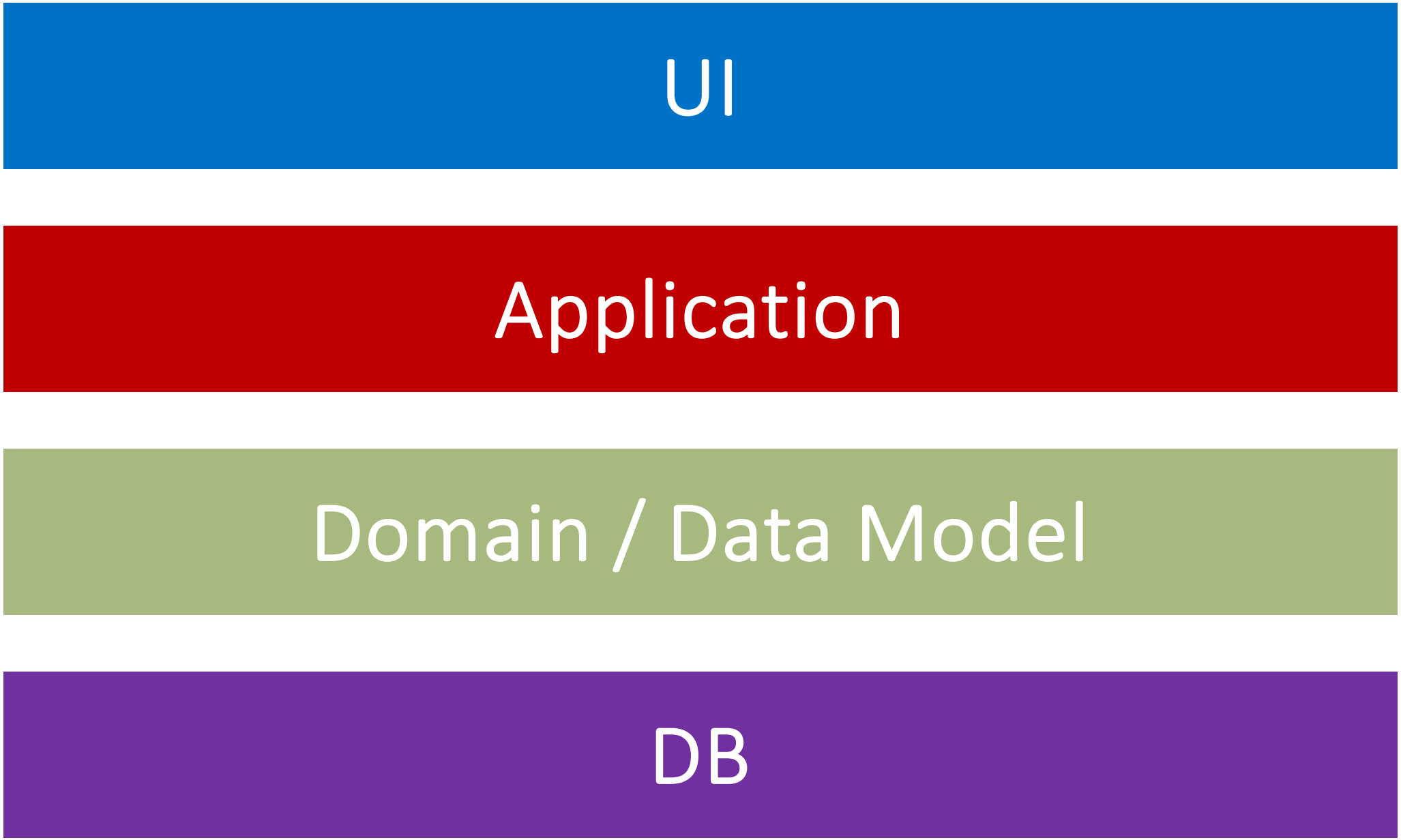

UI
Applicatior
Domain / Data M
DB2
Good thing we made all those layers!

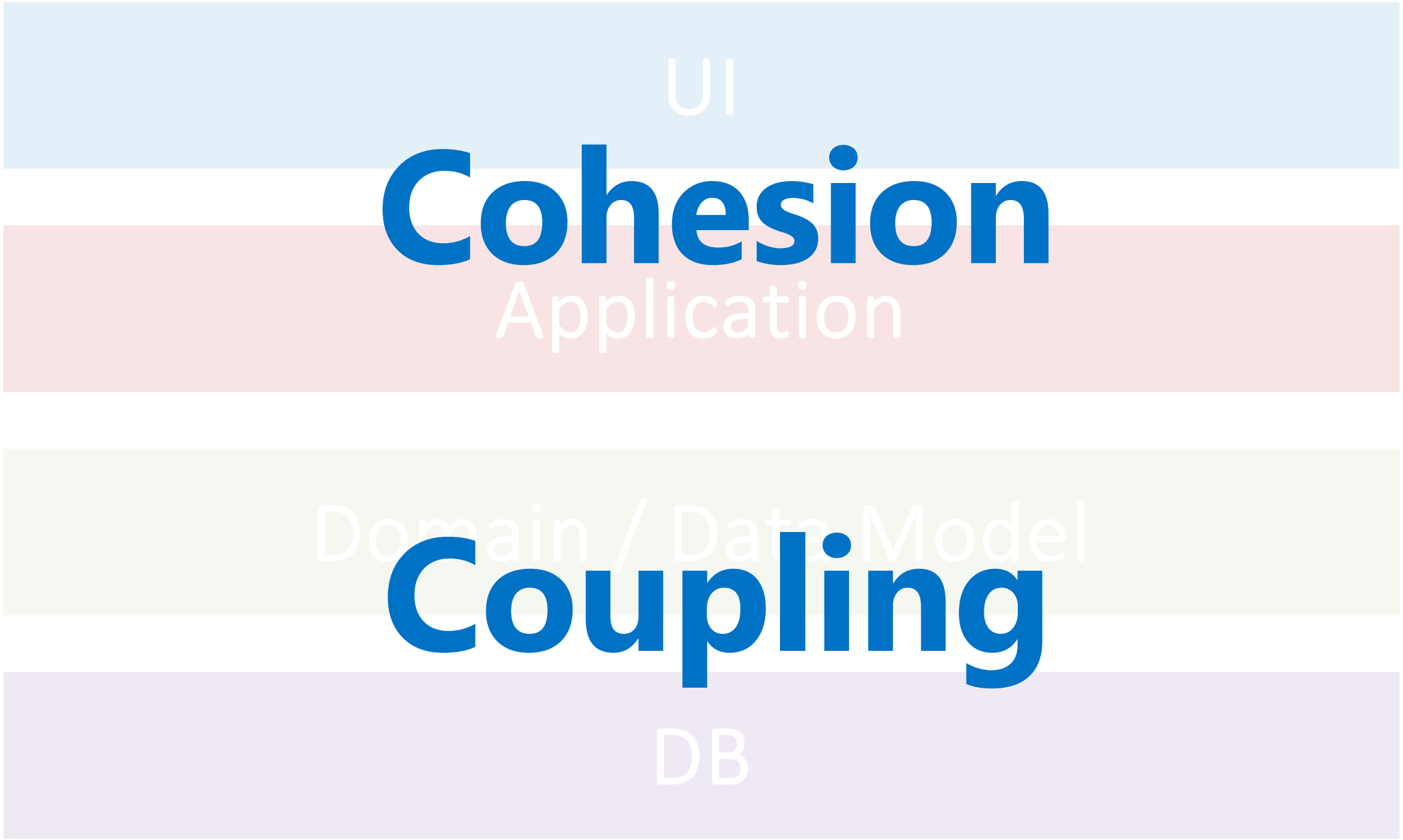

"Change the
Data Access Layer"
TRAILHEAD
TECHNOLOGY PARTNERS

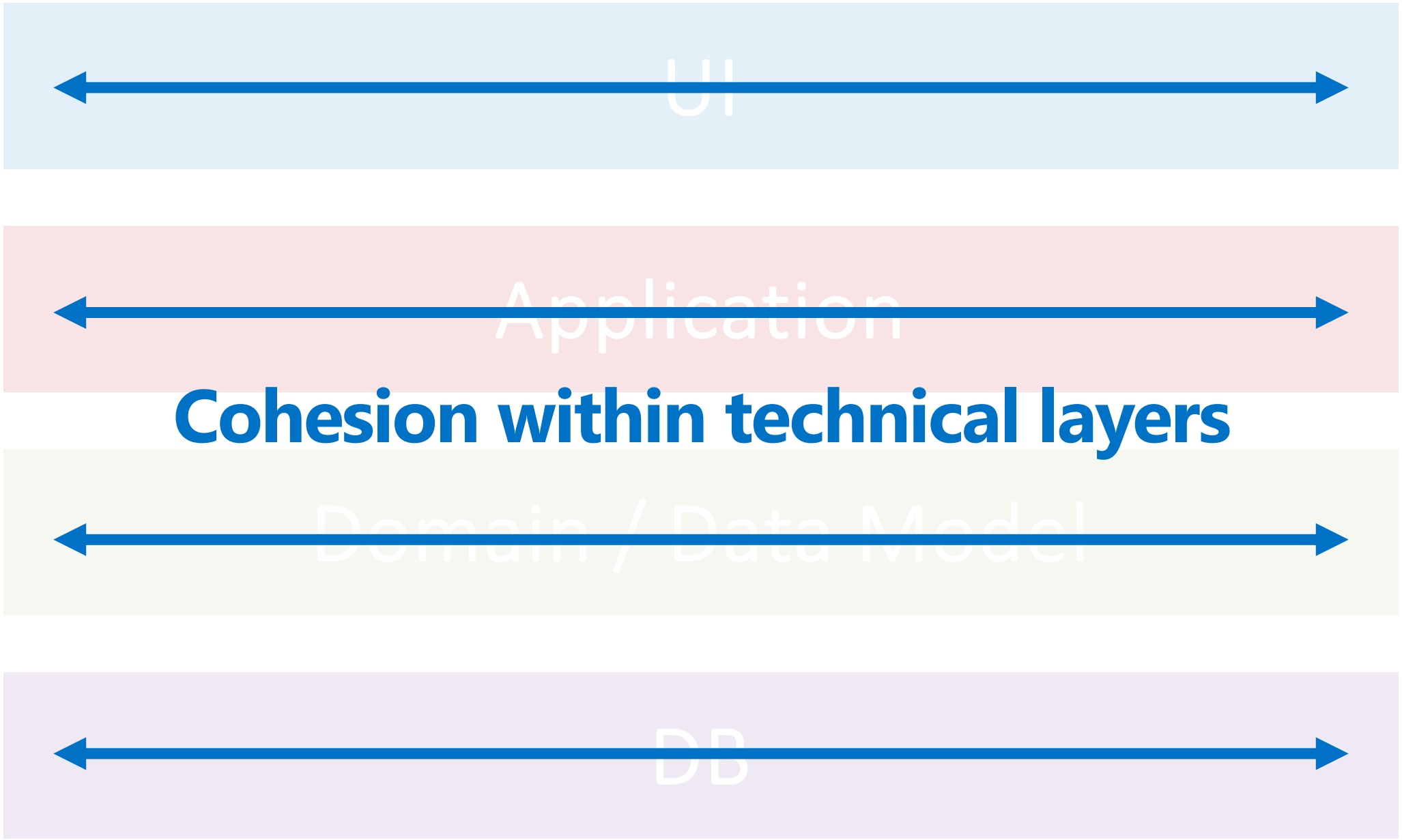
Kinds of Change In a Software Project



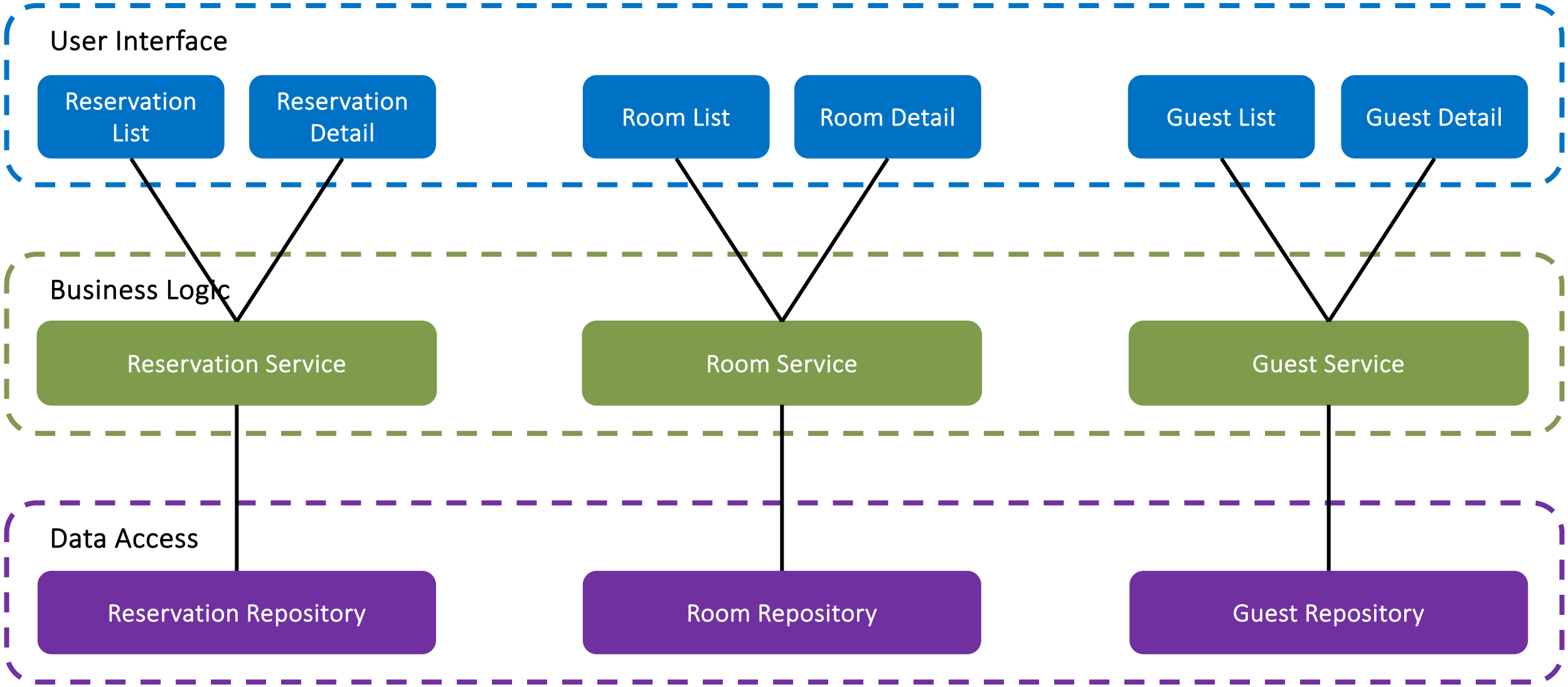
TRAILHEAD
TECHNOLOGY PARTNERS

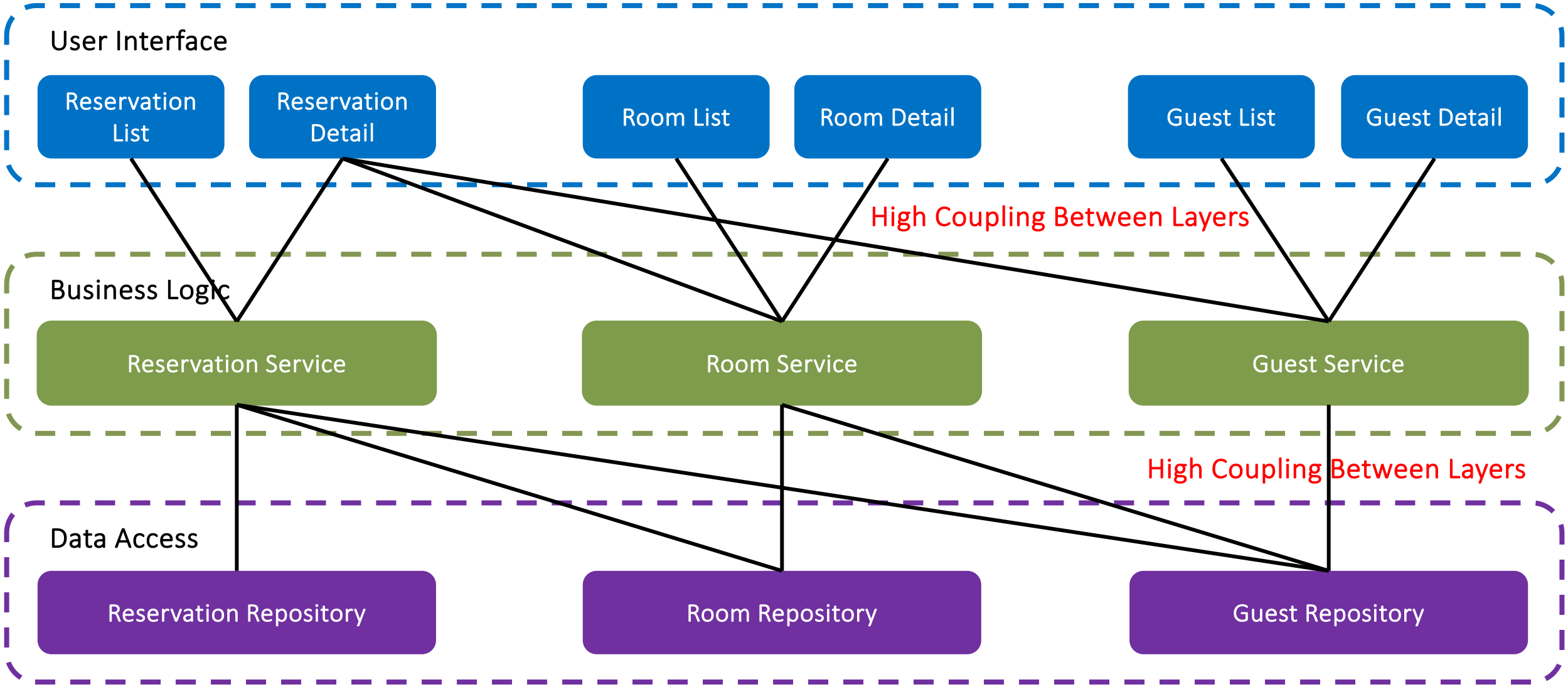

UI
Application
Domain / Data Model
DB


UI
Cohesion
Application
Domain / Data Model
Coupling
DB

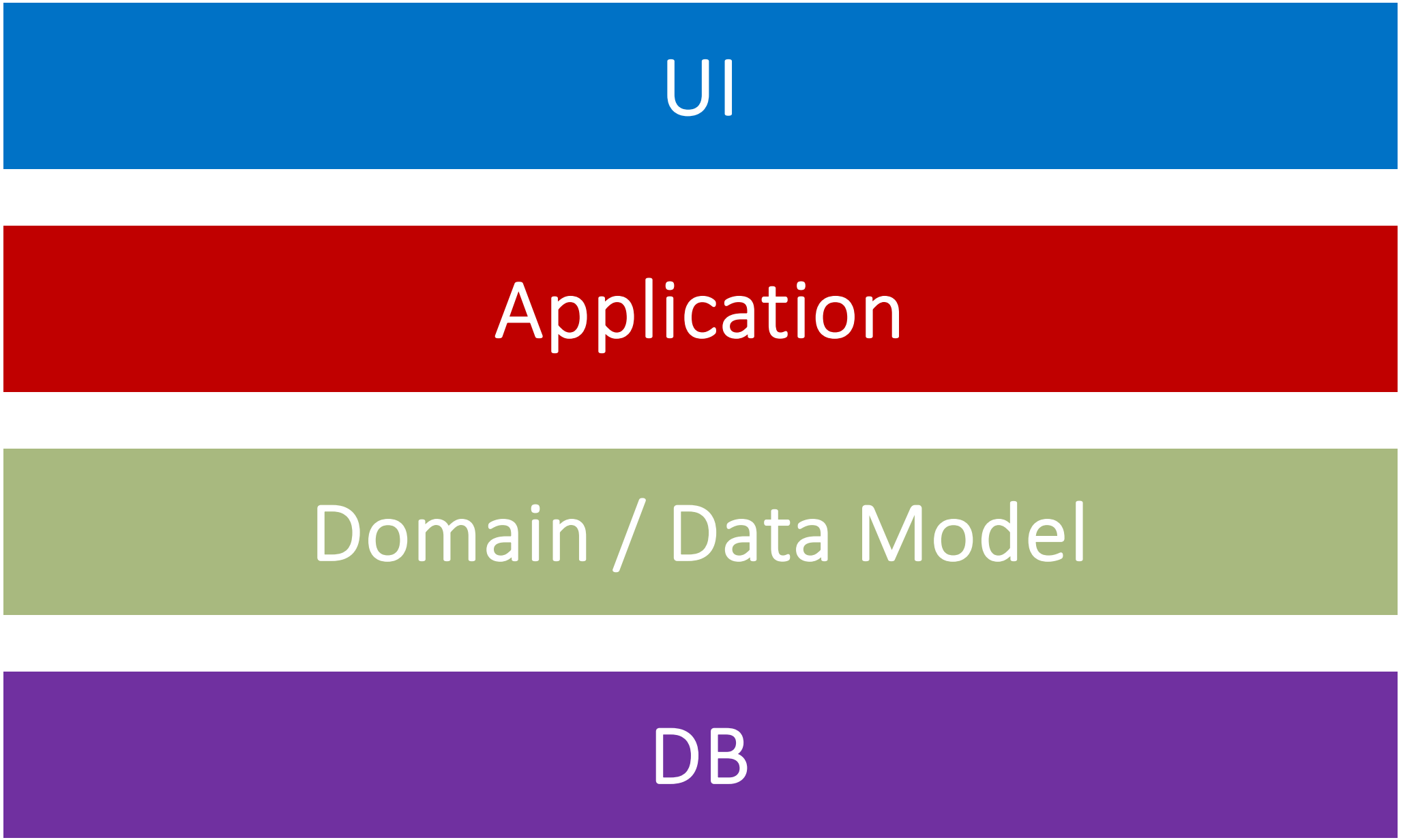


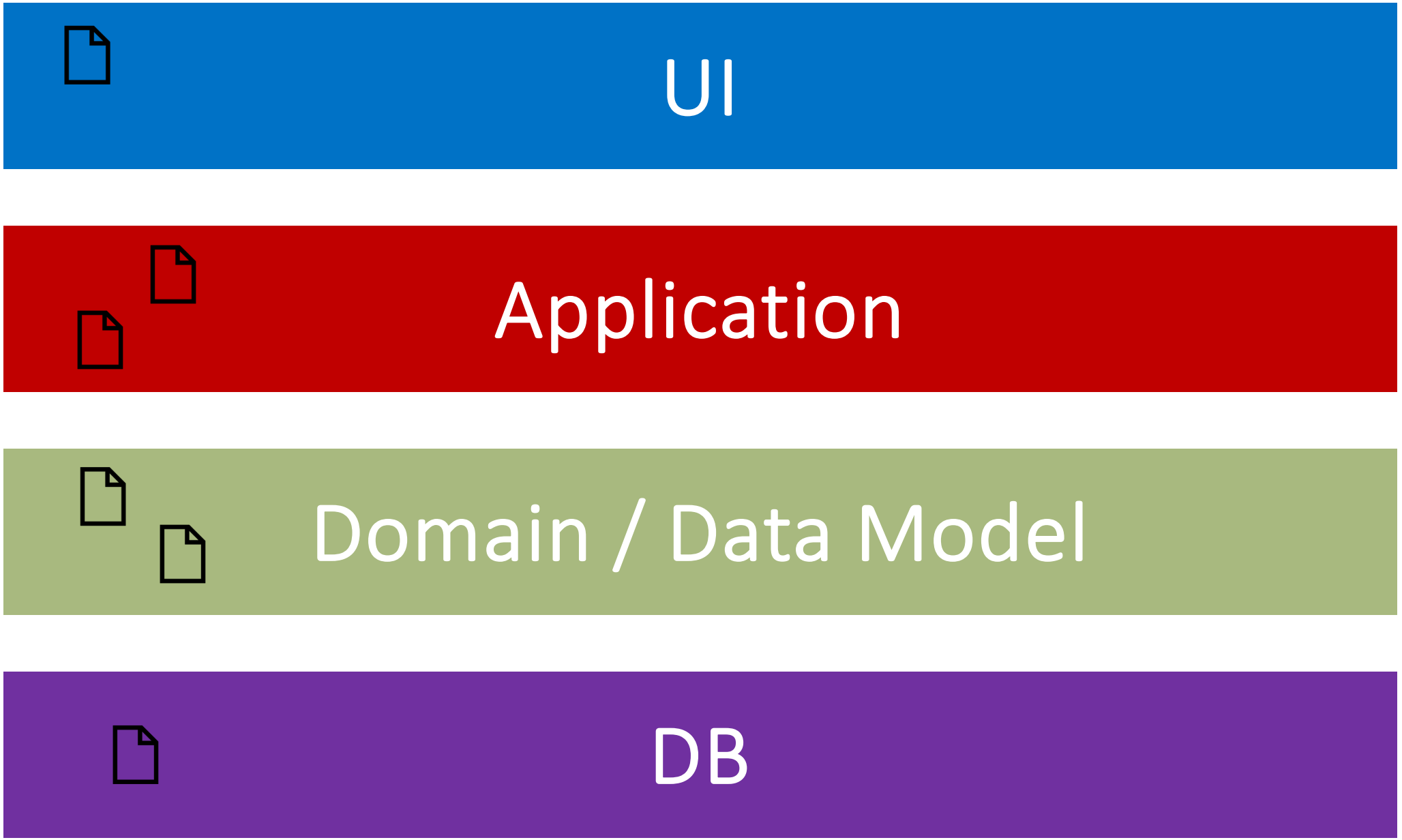
Cohesion within technical layers

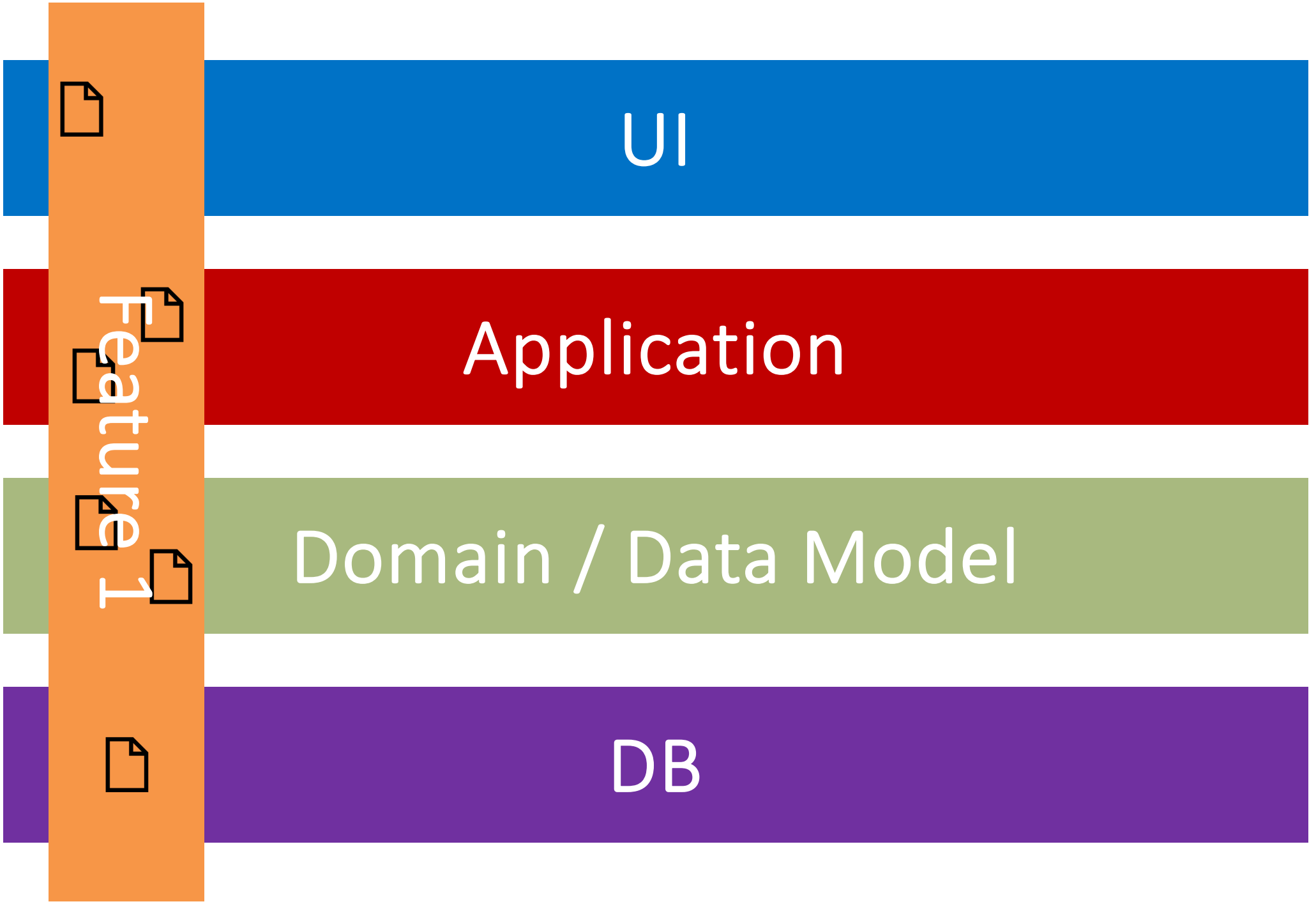

User Interface
Reservation List
Reservation Detail
Room List
Room Detail
Guest List
Guest Detail
Business Logic
Reservation Service
Room Service
Guest Service
Data Access
Reservation Repository
Room Repository
Guest Repository

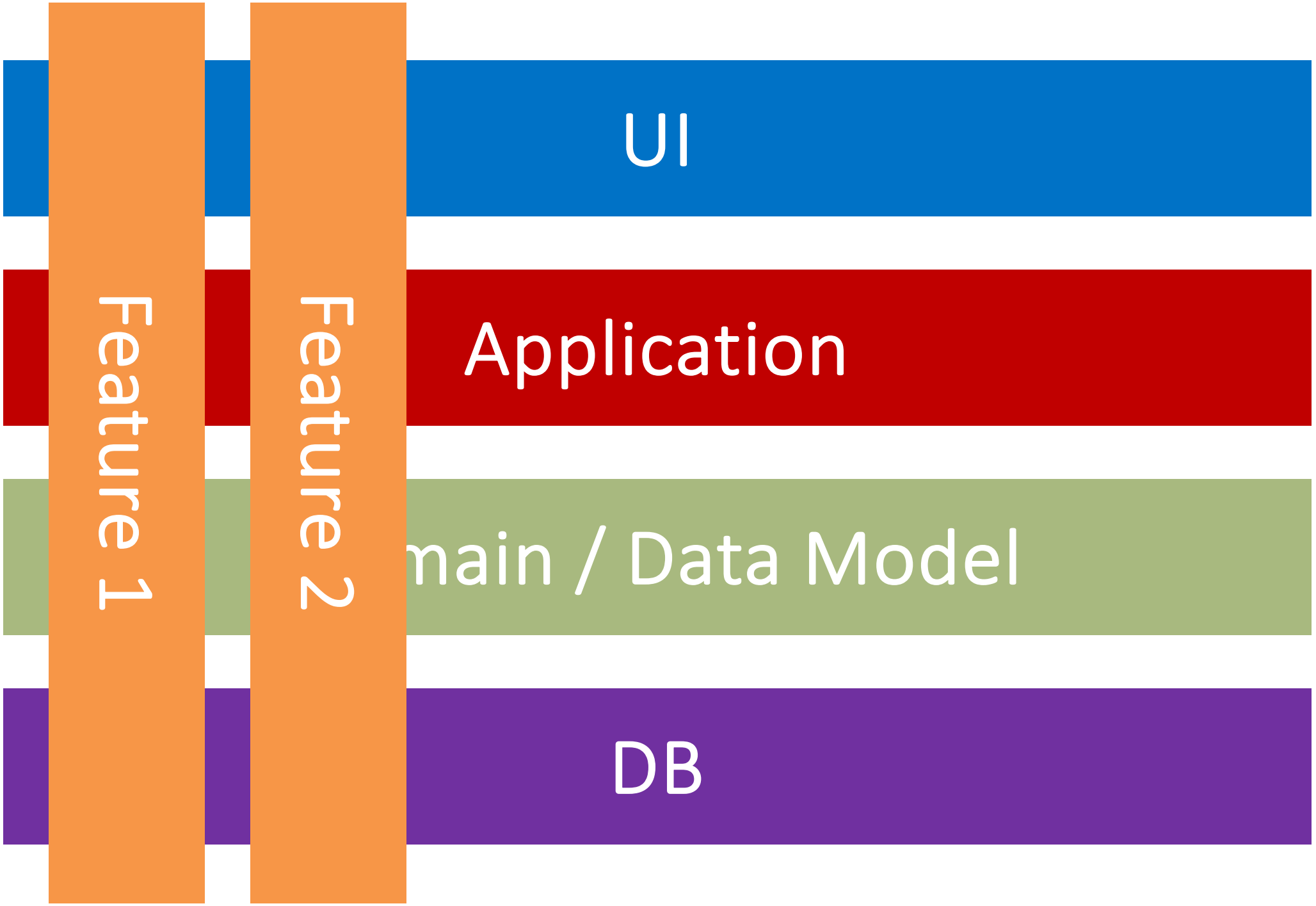

User Interface
Reservation List
Reservation Detail
Room List
Room Detail
Guest List
Guest Detail
High Coupling Between Layers
Business Logic
Reservation Service
Room Service
Guest Service
High Coupling Between Layers
Data Access
Reservation Repository
Room Repository
Guest Repository

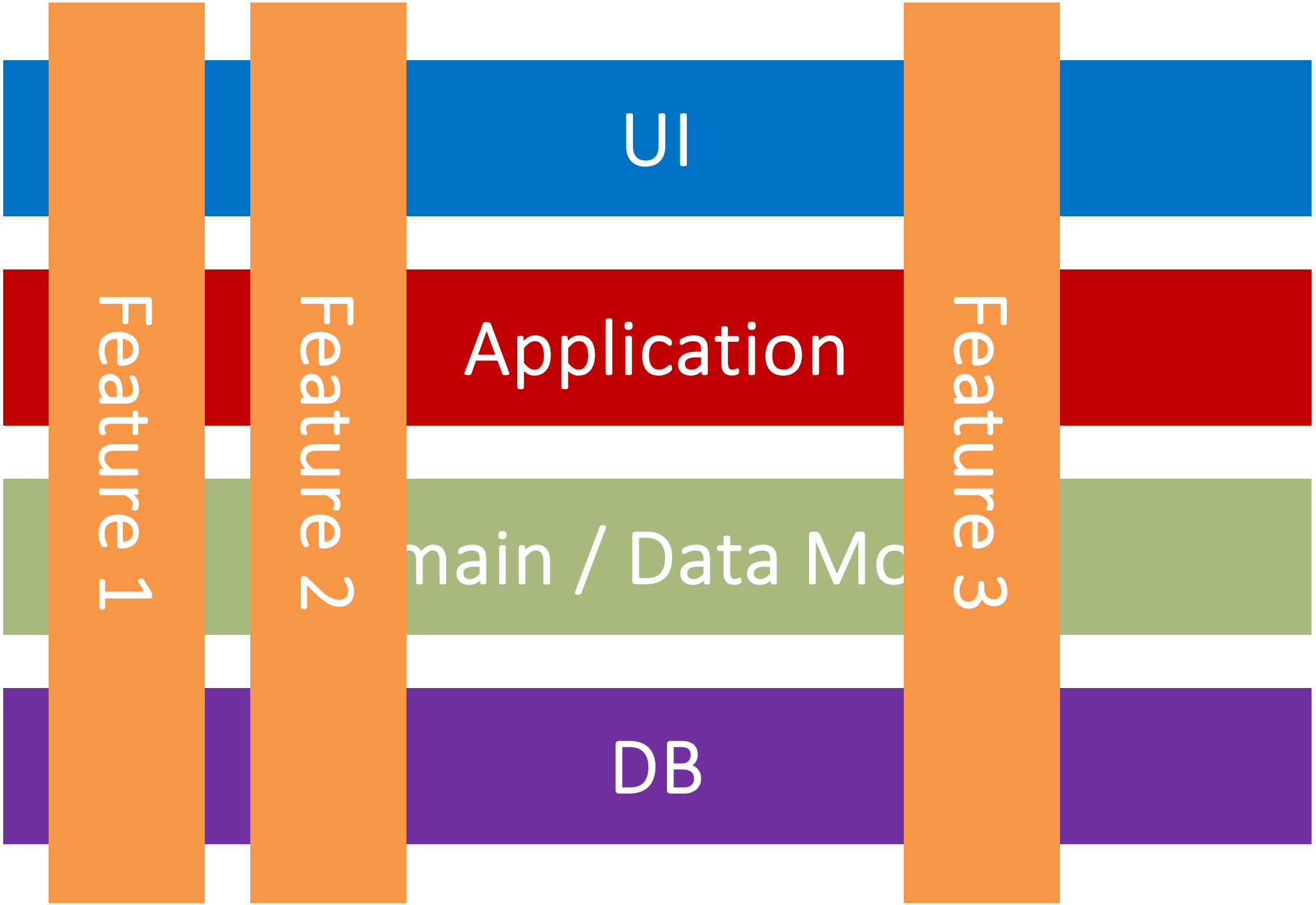
~~Modular~~

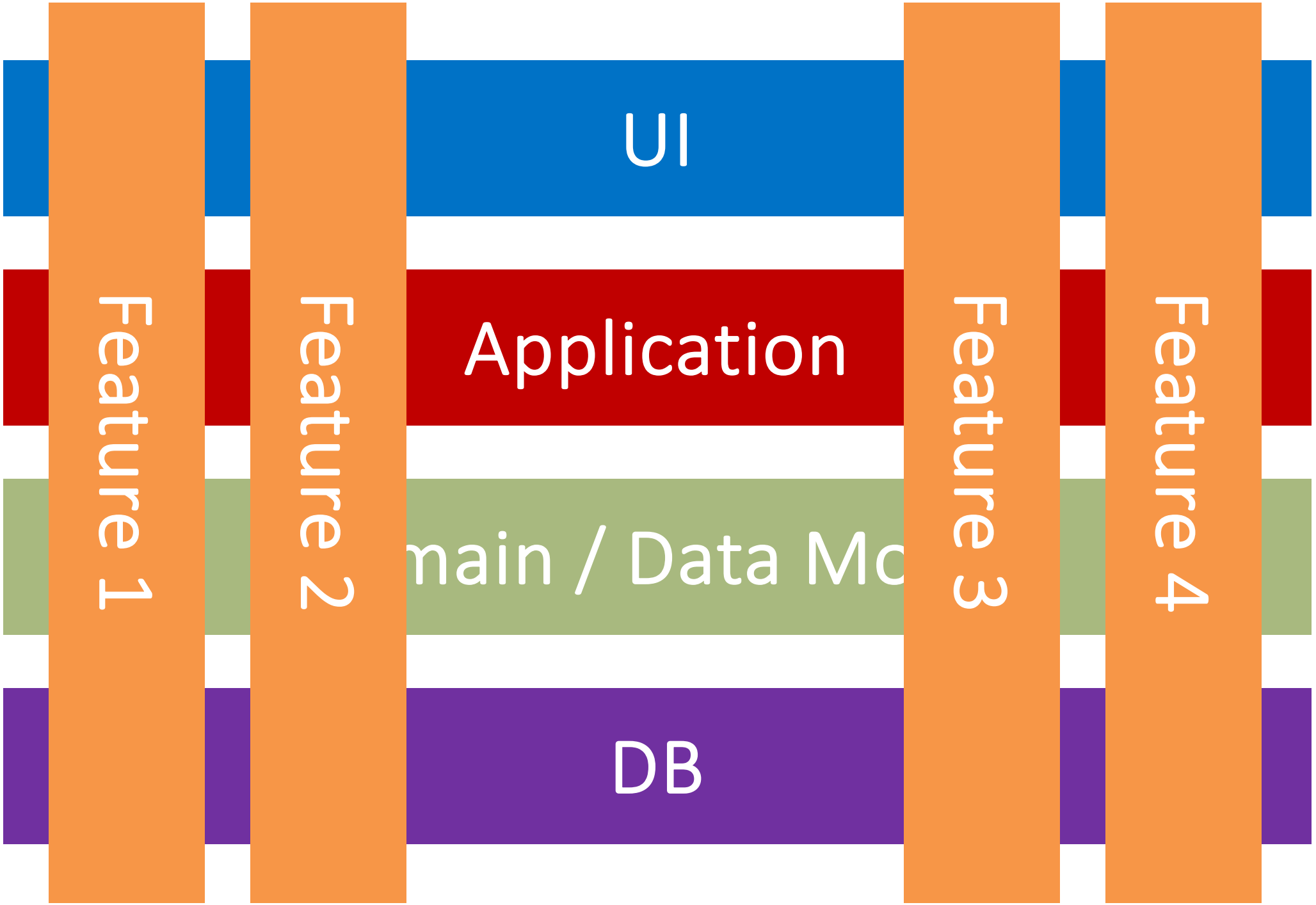

UI
Application
Domain / Data Model
DB

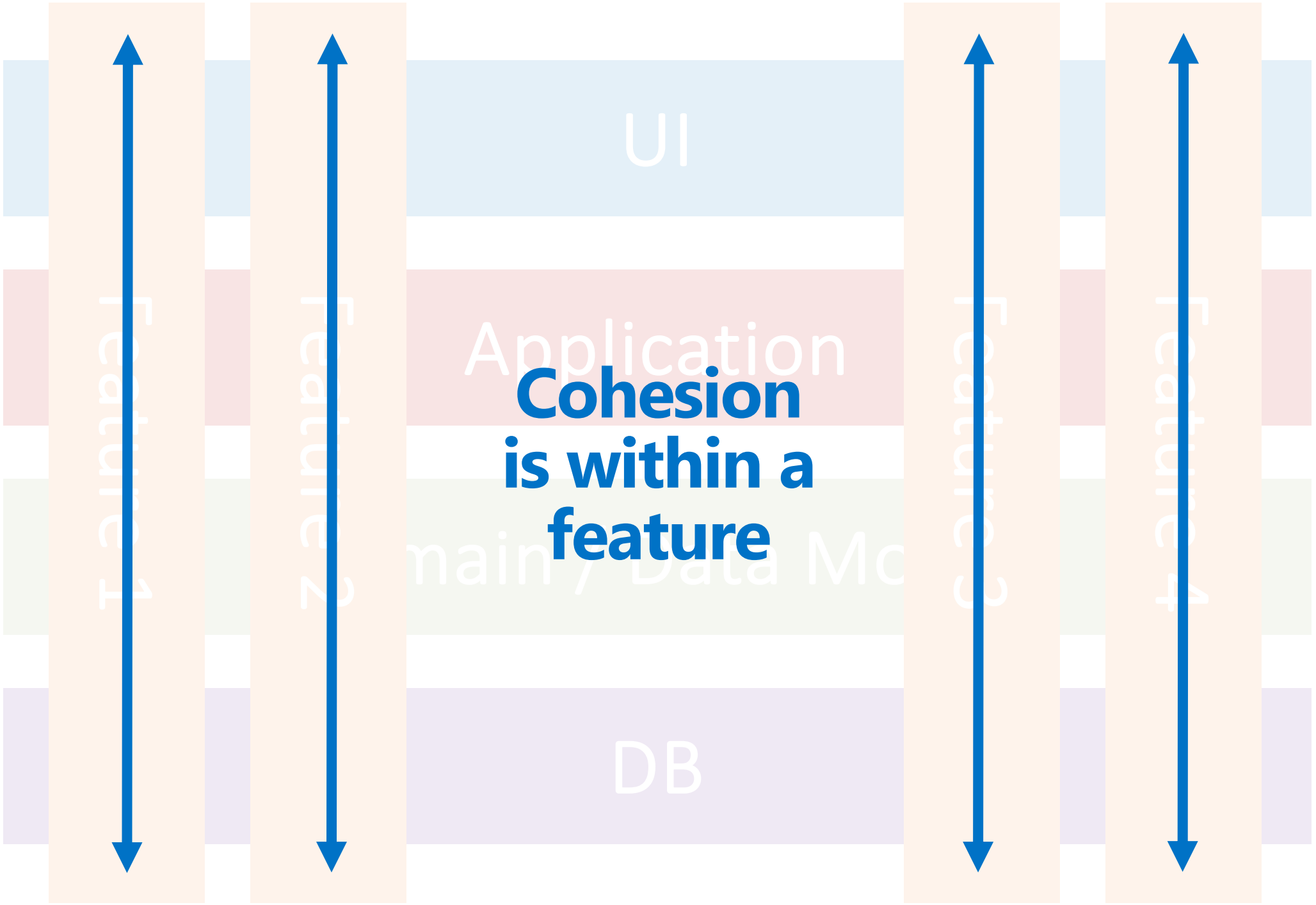

UI
Application
Domain / Data Model
DB

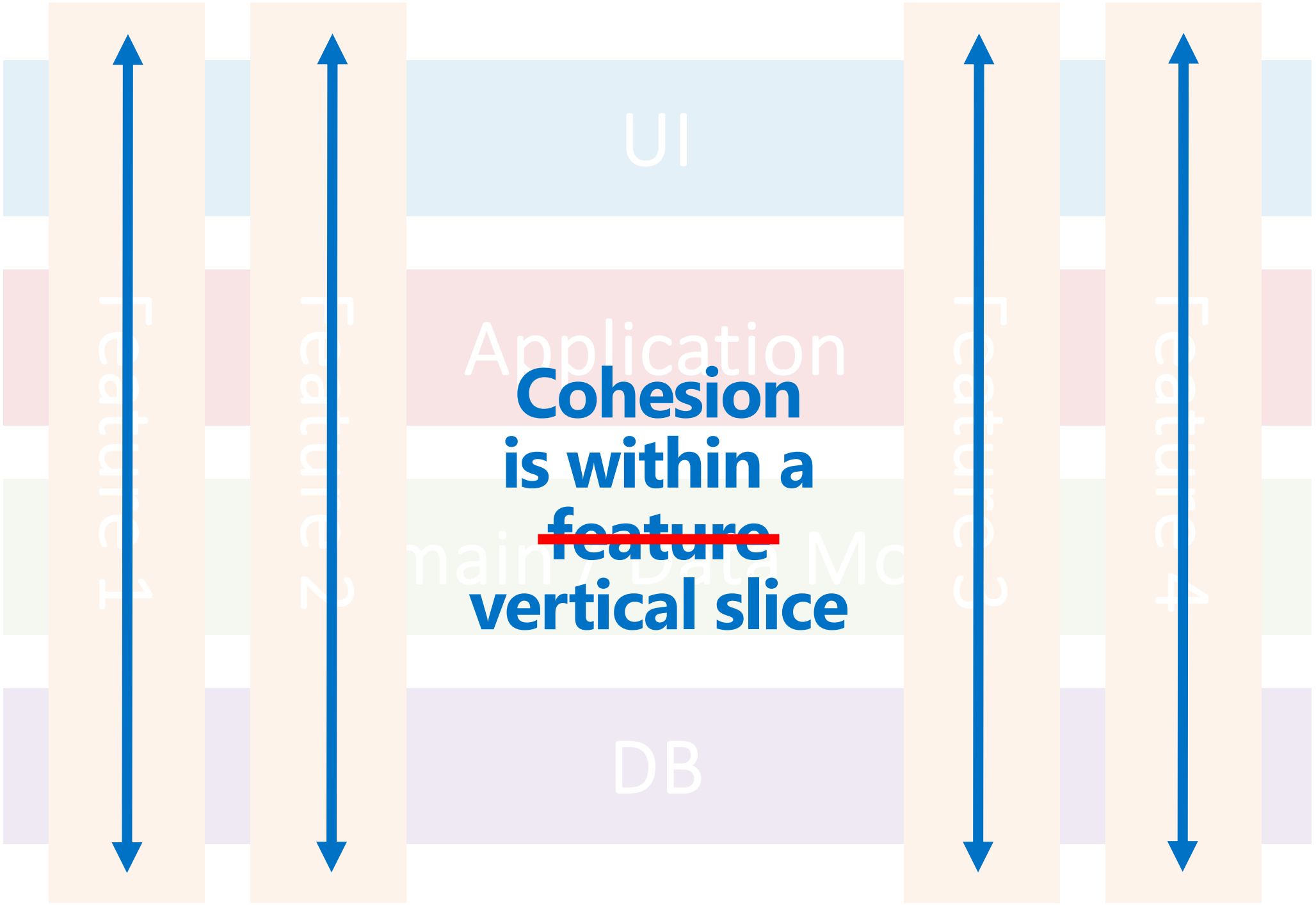

UI
Application
Domain / Data Model
DB
Feature 1


UI
Application
main / Data Model
DB
Feature 1
Feature 2

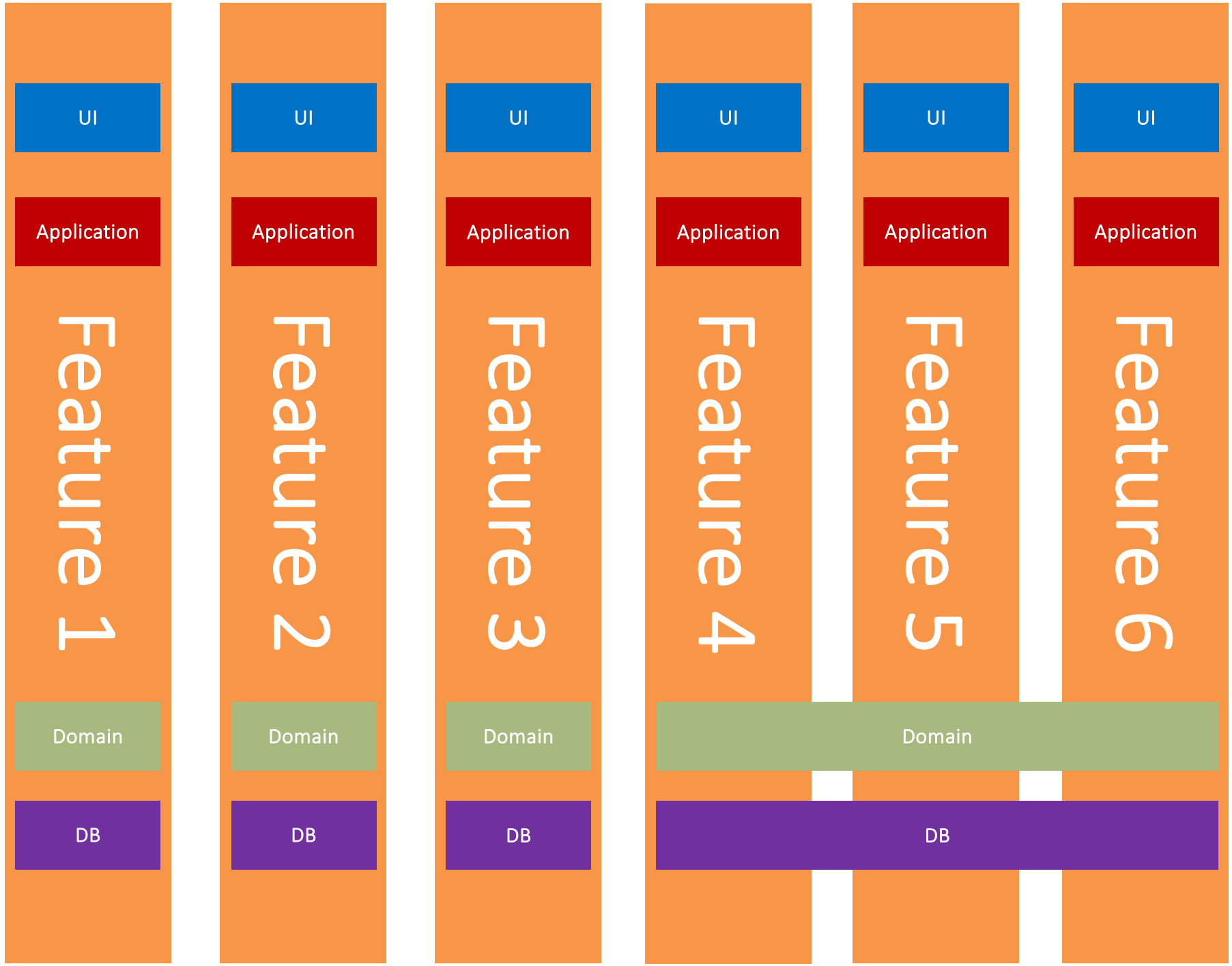

UI
Application
main / Data Mc
DB
Feature 1
Feature 2
Feature 3

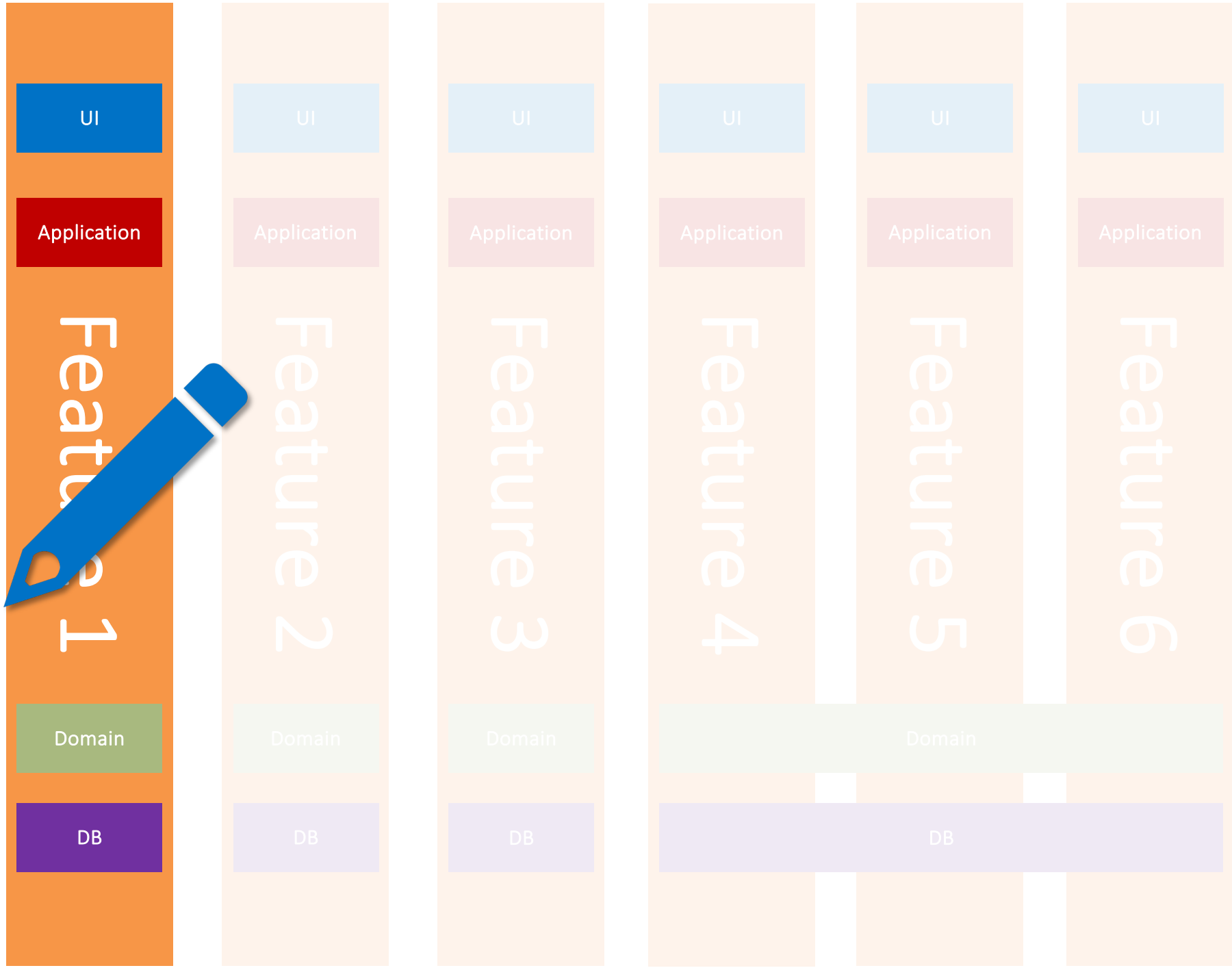

UI
Application
main / Data Mo
DB
Feature 1
Feature 2
Feature 3
Feature 4

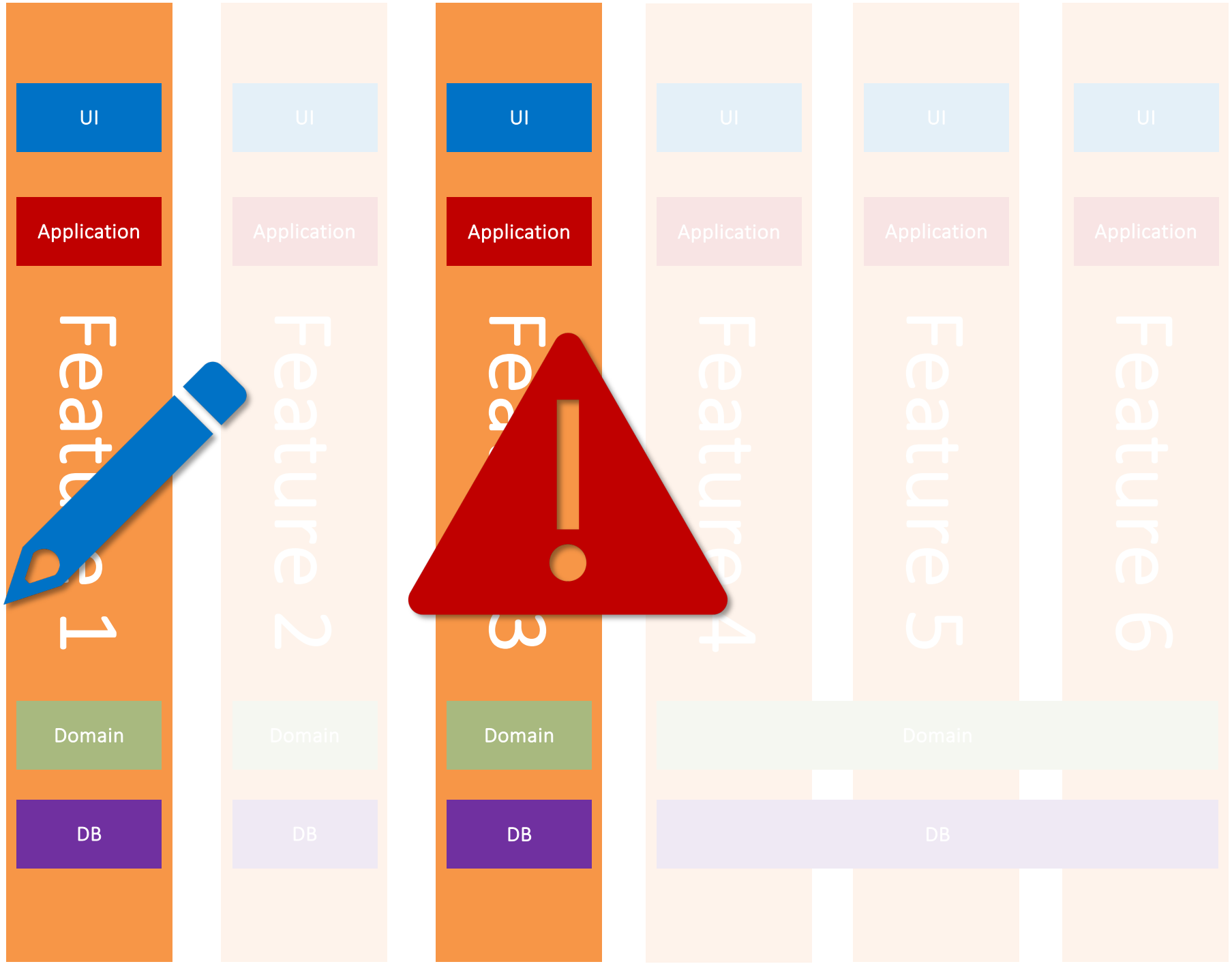

UI
Application
Domain / Data Mo
DB
Feature 1
Feature 2
Feature 3
Feature 4
Cohesion
is within a
feature

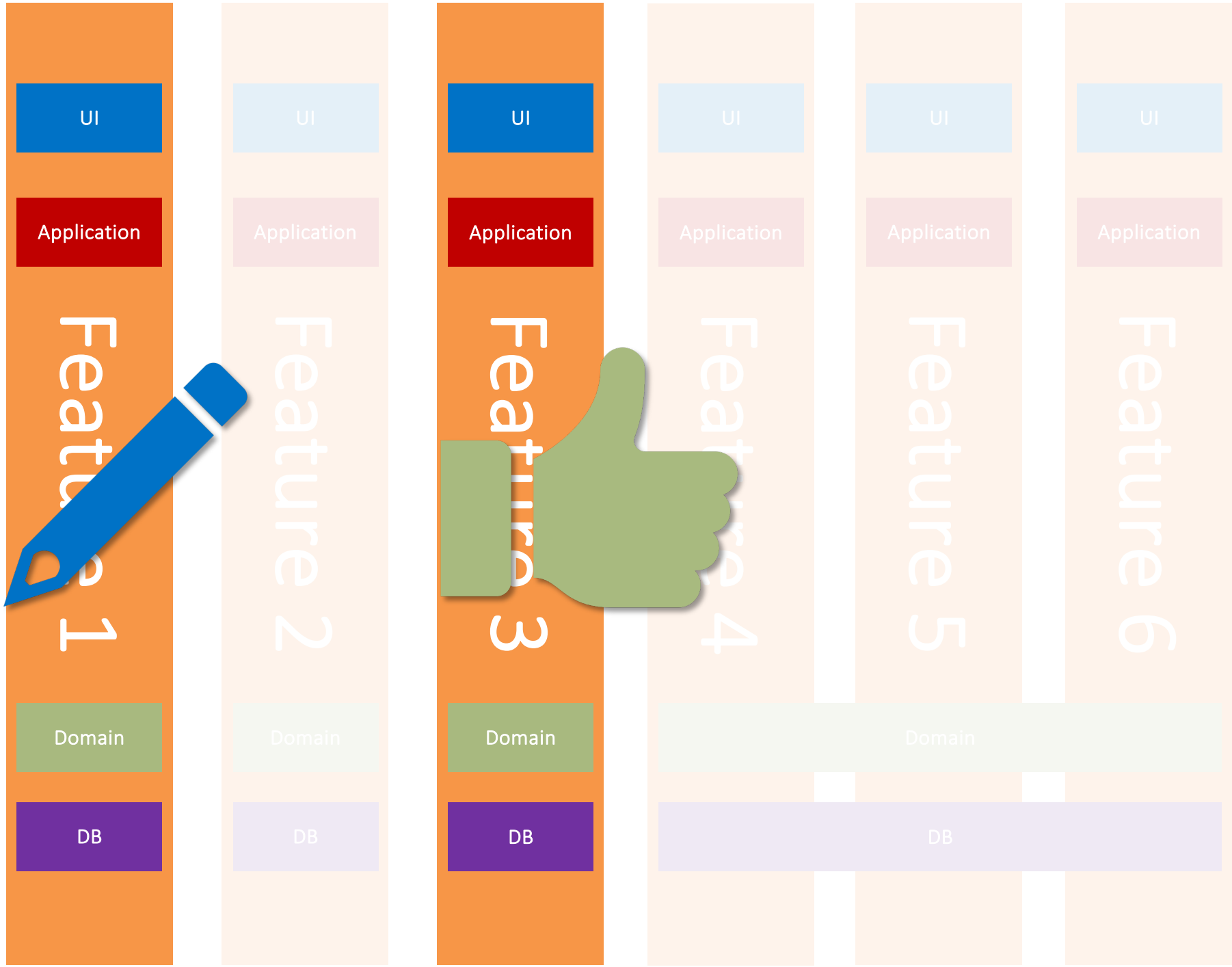


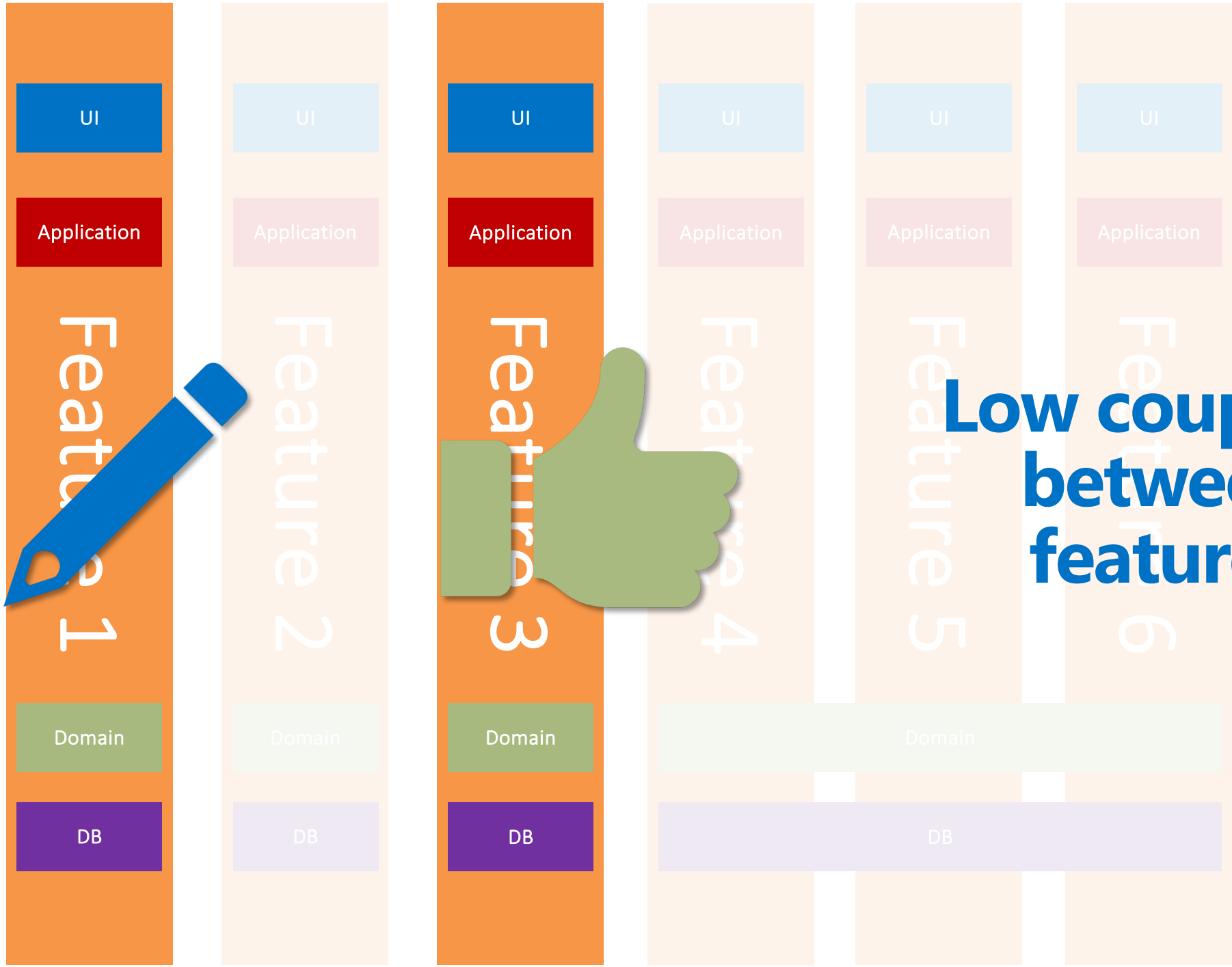
Cohesion is within a ~~feature~~ vertical slice


UI
Application
Feature 1
Domain
DB
UI
Application
Feature 2
Domain
DB
UI
Application
Feature 3
Domain
DB
UI
Application
Feature 4
Domain
DB
UI
Application
Feature 5
UI
Application
Feature 6


UI
Application
Feature 1
Domain
DB
UI
Application
Feature 2
Domain
DB
UI
Application
Feature 3
Domain
DB
UI
Application
Feature 4
Domain
DB
UI
Application
Feature 5
UI
Application
Feature 6


UI
Application
Feature 1
Domain
DB
UI
Application
Feature 2
Domain
DB
UI
Application
Feature 3
Domain
DB
UI
Application
Feature 4
Domain
DB
UI
Application
Feature 5
UI
Application
Feature 6


UI
Application
Feature 1
Domain
DB
UI
Application
Feature 2
Domain
DB
UI
Application
Feature 3
Domain
DB
UI
Application
Feature 4
Domain
DB
UI
Application
Feature 5
UI
Application
Feature 6



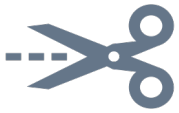
Low coupling between features

Vertical Slice Architecture in .NET

Why Use MediatR

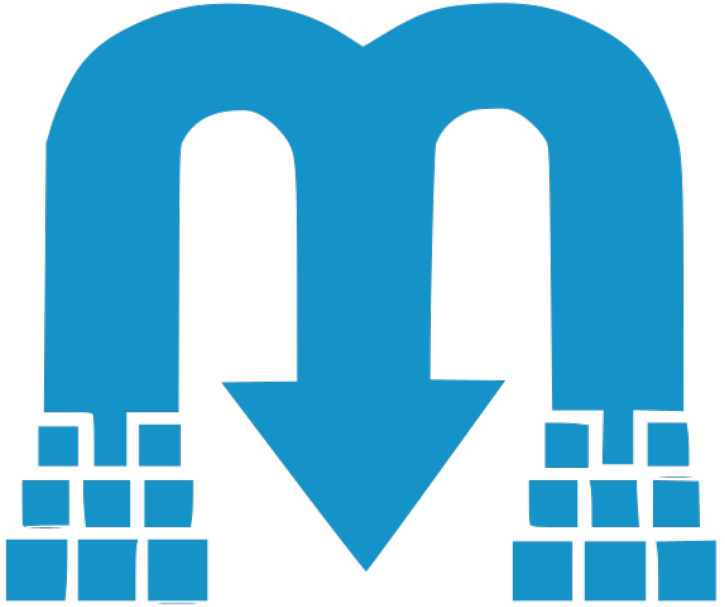
REPR Pattern

DI



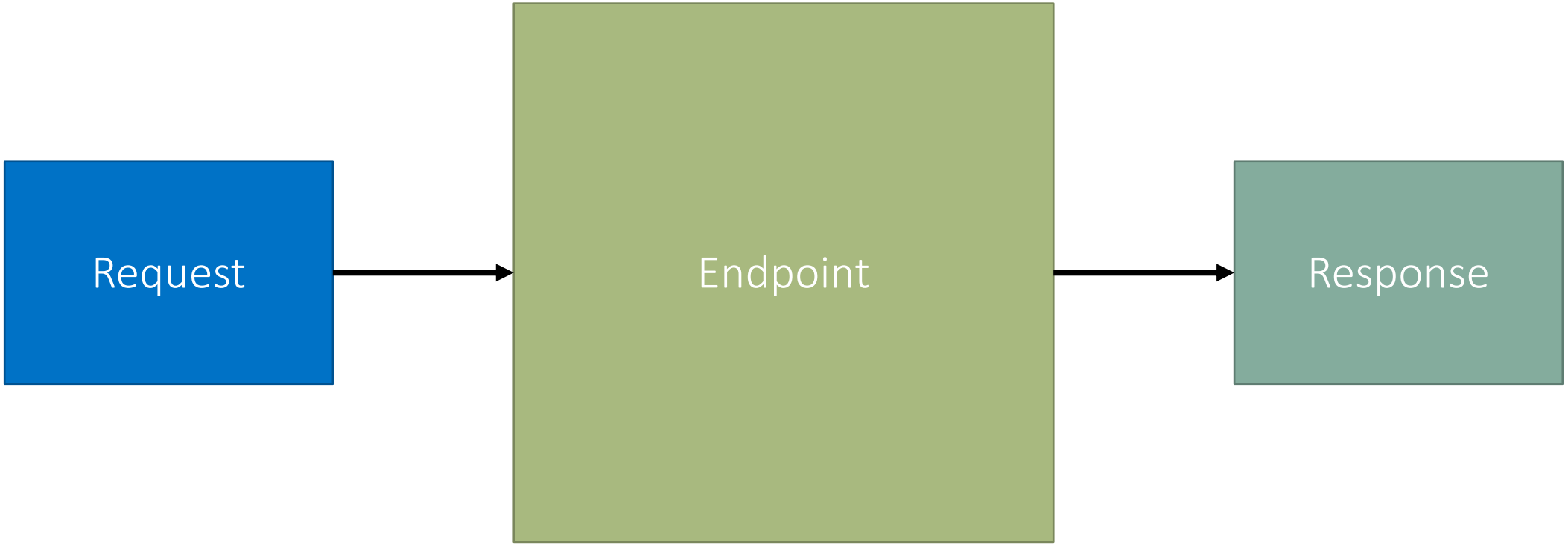
Low Coupling

Cross-Cutting Concerns



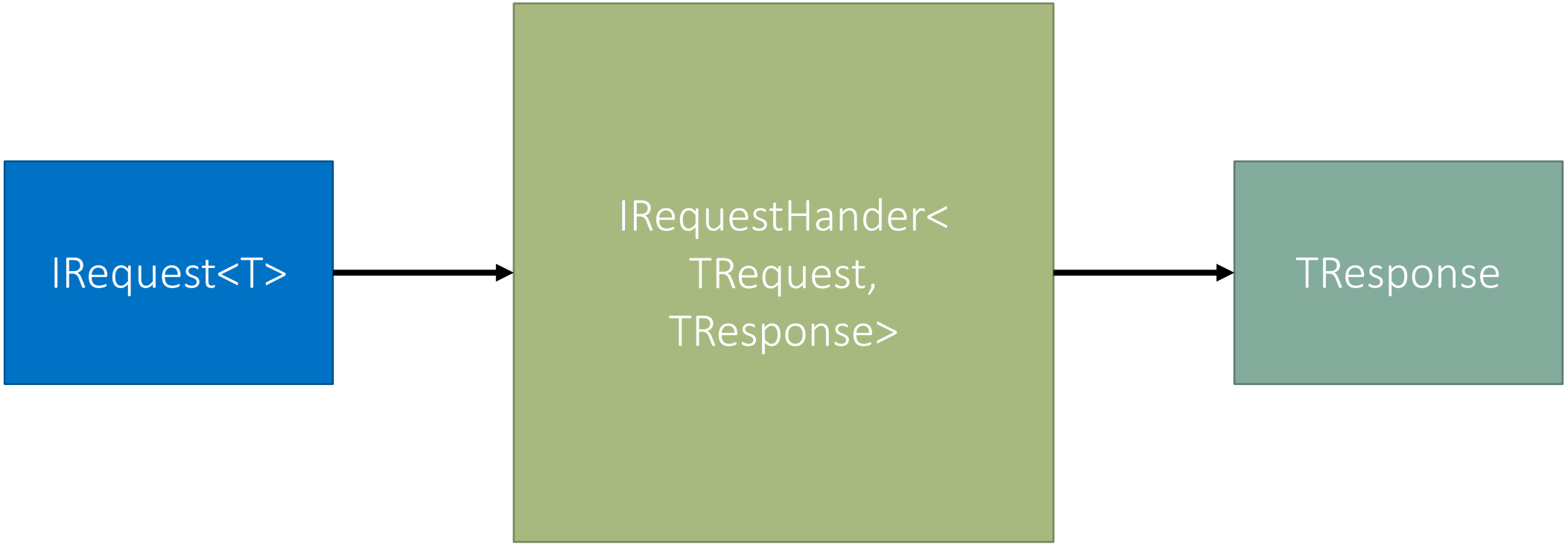
TRAILHEAD
TECHNOLOGY PARTNERS

REPR Pattern



TRAILHEAD
TECHNOLOGY PARTNERS

MediatR



TRAILHEAD
TECHNOLOGY PARTNERS

MediatR

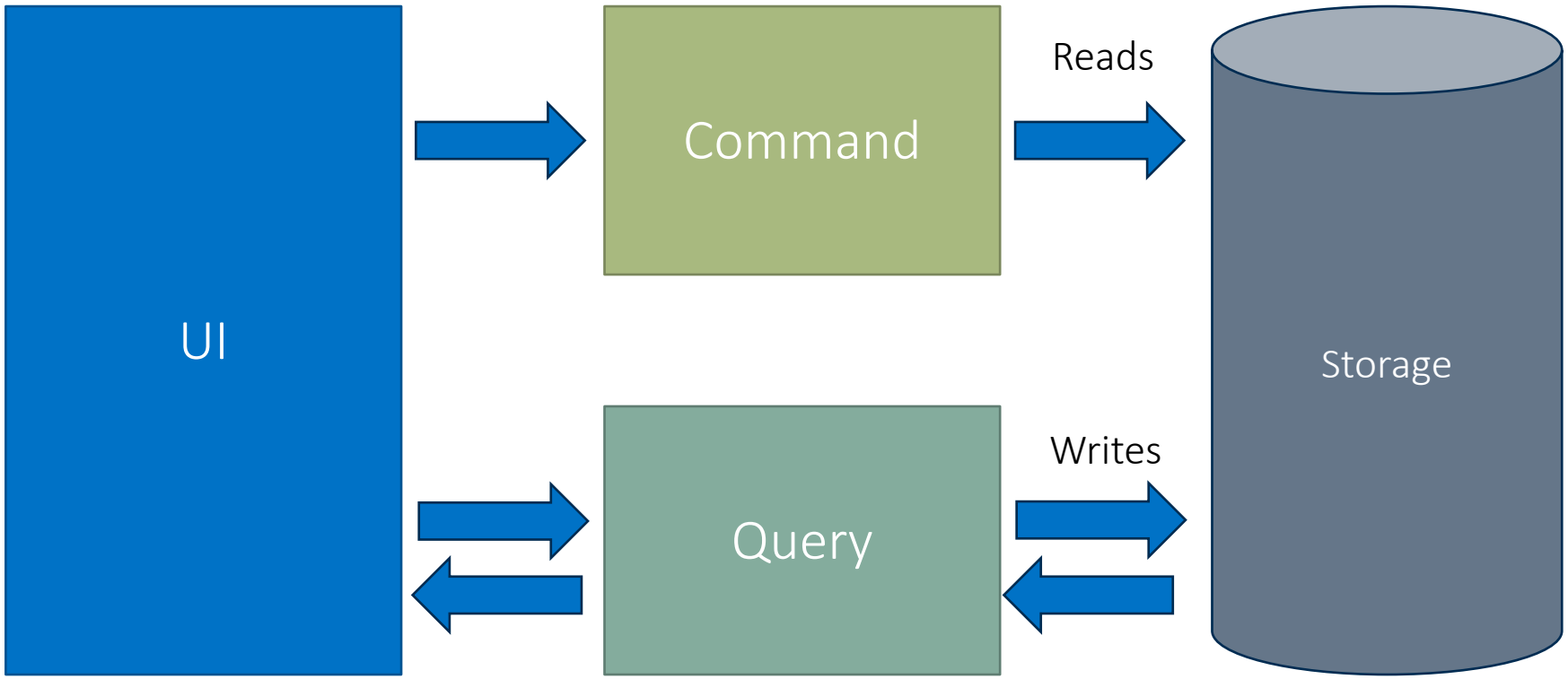
```
public class Ping : IRequest<Pong> { }

public class PingHandler : IRequestHandler<Ping, Pong>
{
    public Task<Pong> Handle(Ping request, CancellationToken cancellationToken)
    {
        return Task.FromResult(new Pong());
    }
}

public class Pong { }
```

TRAILHEAD
TECHNOLOGY PARTNERS

MediatR

```
var pong = await sender.Send(new Ping());
```

CQRS



CQRS

Commands:

Updates data

Task-based vs data-centric

(Optional) May be queued

Return nothing (or just ID)

Queries:

Reads data

Never modifies data

Returns results

Minimal APIs

Program.cs

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/hello", async () => "Hello World!");
app.MapGet("/item/{id}", async (int id) => "You passed in ID {id}.");
app.MapPost("/item", async (Item item) => "Item {item.ID} updated.");

app.Run();
```

Minimal APIs

Program.cs

```
app.MapGet("api/data", async (ISender sender) =>
{
    var result = await sender.Send(new GetData.Query());
    return Results.Ok(result.Value);
})
```

FluentValidation

```
public class Validator : AbstractValidator<Command>
{
    public Validator()
    {
        RuleFor(r => r.RoomId).NotEmpty().GreaterThanOrEqualTo(1);
        RuleFor(r => r.GuestFirstName).NotEmpty();
        RuleFor(r => r.GuestLastName).NotEmpty();
        RuleFor(r => r.CheckInDate)
            .NotEmpty()
            .Must(date => date.Date >= DateTime.Now.Date)
                .WithMessage("Check-in date must be today or a future date.");
        RuleFor(r => r.CheckOutDate)
            .NotEmpty()
            .Must((model, checkOutDate) => checkOutDate.Date > model.CheckInDate.Date)
                .WithMessage("Check-out date must be after the check-in date.");
    }
}
```

FluentValidation

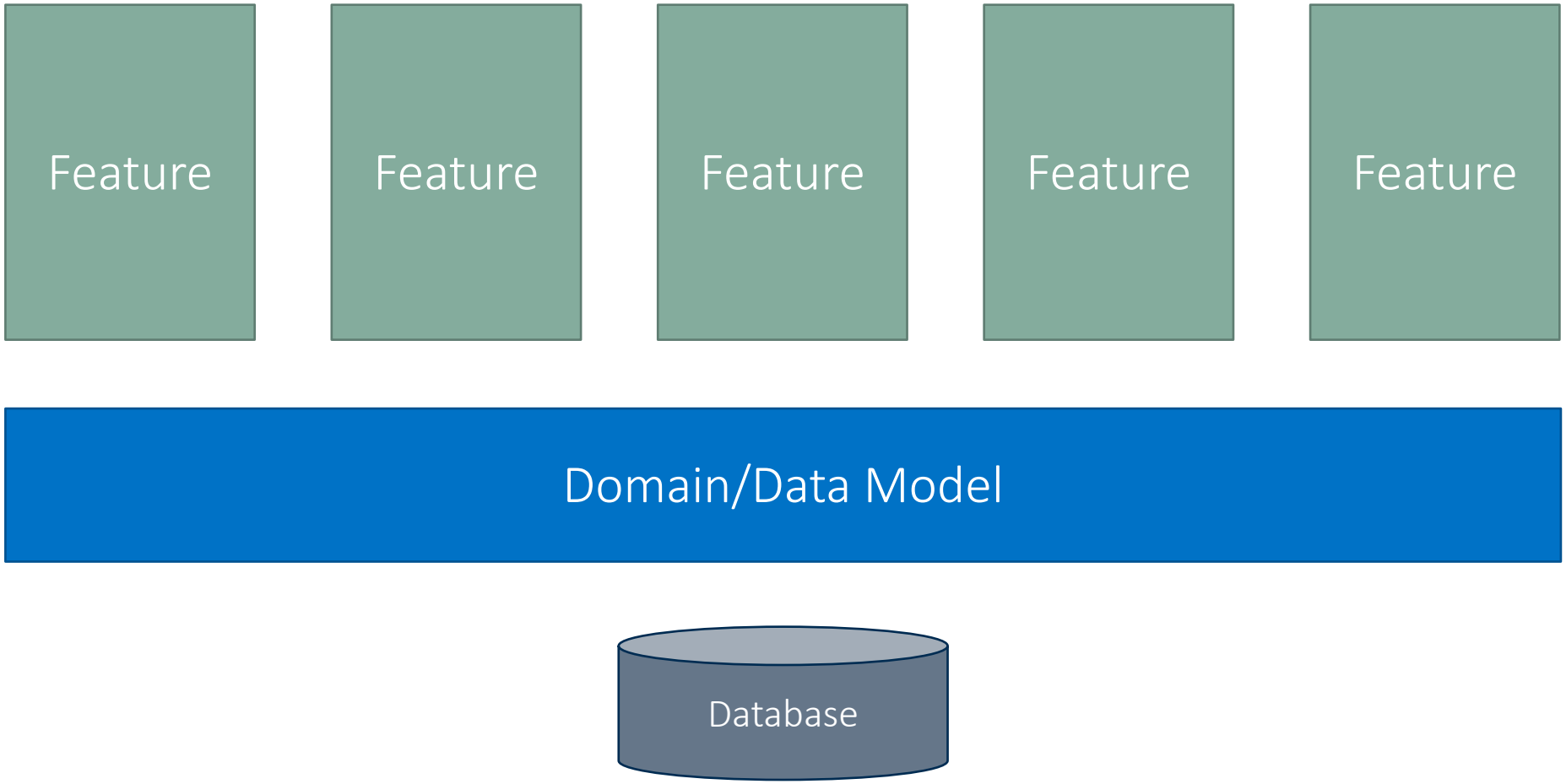
```
var validationResult = _validator.Validate(request);

if (!validationResult.IsValid)
{
    return Result.Failure<Guid>(new Error(
        "CreateReservation.Validation",
        validationResult.ToString()));
}
```

Demo

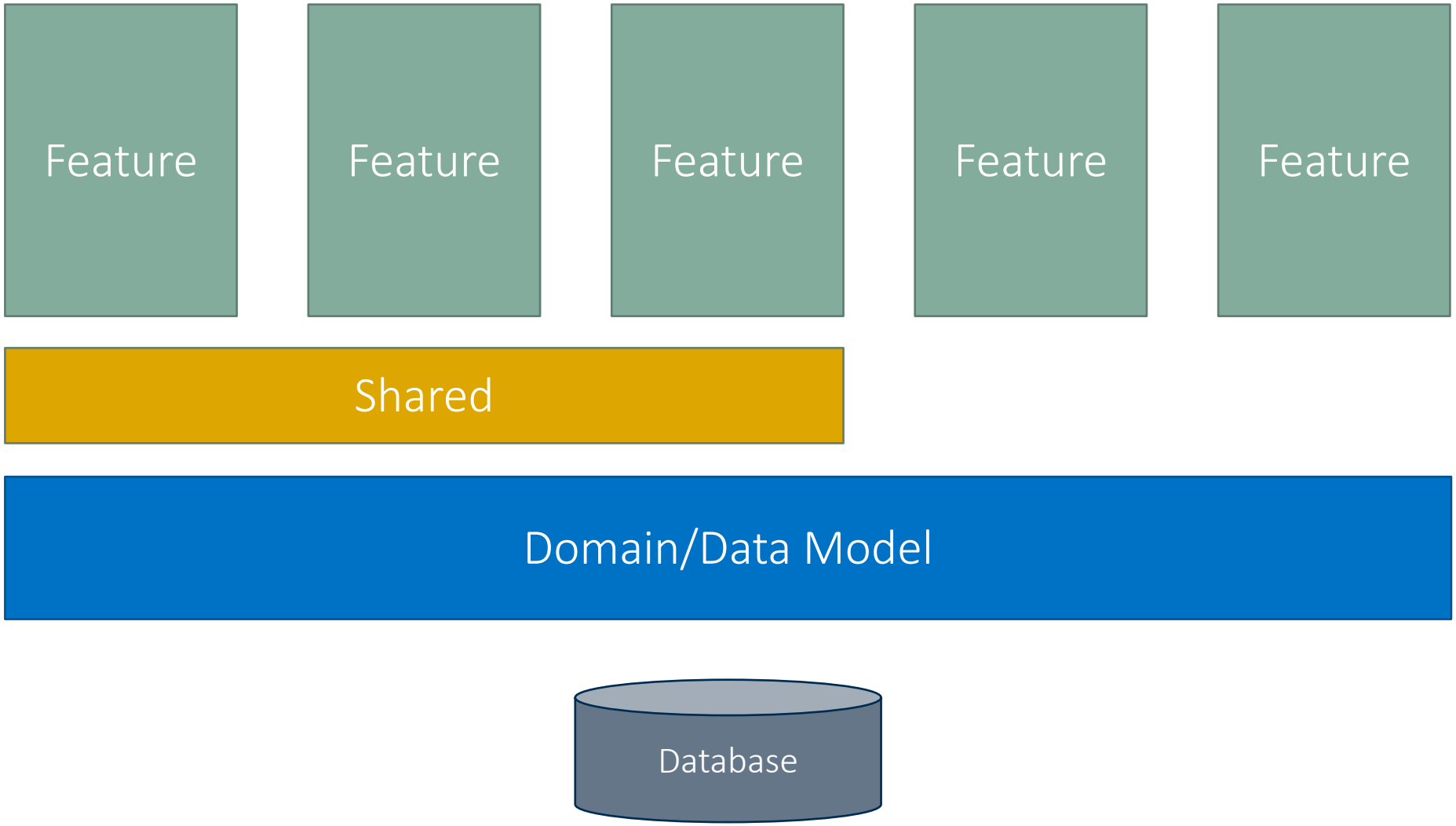
Vertical Slice Architecture

FAQs

Is Sharing Allowed?



Is Sharing Allowed?



Is Sharing Allowed?

Rule of Three:

"Three strikes and you refactor"

- Martin Fowler

Are Their Still Abstractions?

- Database
- Data Model / DAL
- Layers within vertical slice (api endpoint, validator, command, query, handler, etc)
- No tech stack lock-in (like microservices)

Is Messaging Required?

- NO, but it can
  - Event-Drive Subscription
  - Microservices
  - CQRS

Is VSA Compatible with X?

- I think this myth exists because people think of Vertical Slices as the opposite of Layers or as that it's at odds with Clean or Onion Architecture. But that's not the case.
- The point of vertical slices is about cohesion. Clean (or Onion) Architecture is about the direction of dependencies and how you manage coupling. They are orthogonal concerns and not mutually exclusive.
- For example, you can still have a data access layer or some abstraction over data access. The question is do you need that abstraction?
- As an example, if you were using ORM and it had 100's usages all over your application layer. If you want to change that ORM with some other form of data access, that might be a massive undertaking to change all those usages. This is because you're highly coupled from your application layer to the ORM. If you hide the ORM behind some data access layer, it's still the same problem. The problem is the high degree of coupling. Coupling here is the issue.
- However, if you have a dozen usages for a set of related features that use that ORM, and you want to change it to something else or change the underlying database itself, that surely will be easier to manage that change for a dozen usages instead of 100s.
- The point is that you're segregating and deciding how you couple per feature rather than an entire layer.
- This is why I think the misconception exists because once you start getting narrow in features, you start realizing the value of abstractions decreases because you have less coupling to certain dependencies.

Summing Up

1. Typical **axis of change**: FEATURES
2. VSA prioritizes **cohesion within a feature** over within a technical layer
3. VSA **lowers coupling** between features
4. For **VSAs in .NET** consider using:
   - MediatR
   - CQRS
   - MinAPI
   - FluentValidation
5. **Sharing** is ok, **abstraction** still happens, **messaging** not required, is **compatible** with clean/onion/etc architecture

TRAILHEAD
TECHNOLOGY PARTNERS

Thanks! Questions?

Jonathan "J." Tower

- Microsoft MVP in .NET
- jtower@trailheadtechnology.com
- trailheadtechnology.com/blog
- jtowermi
- jtower

FREE

CONSULTATION

bit.ly/th-offer

github.com/trailheadtechnology/vsa-dotnet

Vertical Slice Architecture

What is Vertical Slide Architecture?

2018 by Jimmy Bogard

Slice per feature or request

Code organized by Feature

Makes changes easier