

The DDD Layered Architecture



Dino Esposito

@despos | <http://software2cents.wordpress.com>

Layered Architecture

revisits the classic 3-tier architecture



Presentation

Business

Data Access

Key Points

Layers and Tiers

Segmentation of a
Software System

Domain Layer

Patterns for Organizing
the Business Logic

Other Layers

From Presentation to
Persistence

Spaghetti **vs.** Lasagna



Notably long and thin pasta which requires ad hoc tools to be served and experience to be eaten.

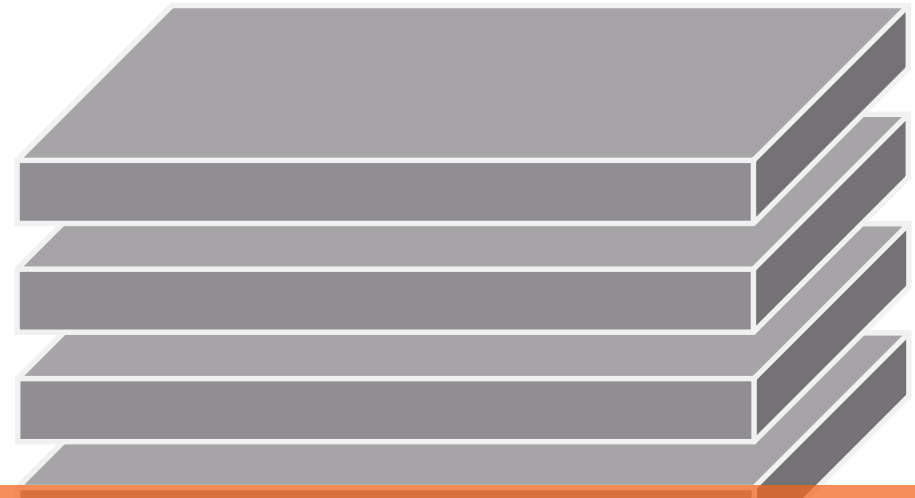


Layered block of noodles and toppings easy to cut in pieces to serve and eat.

Spaghetti-code **vs.** Lasagna-code

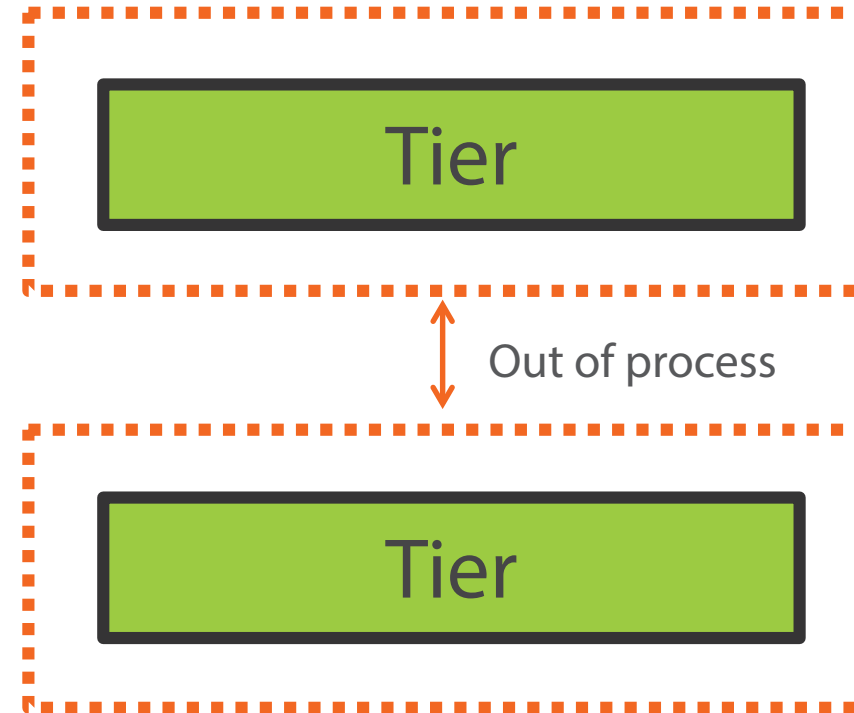
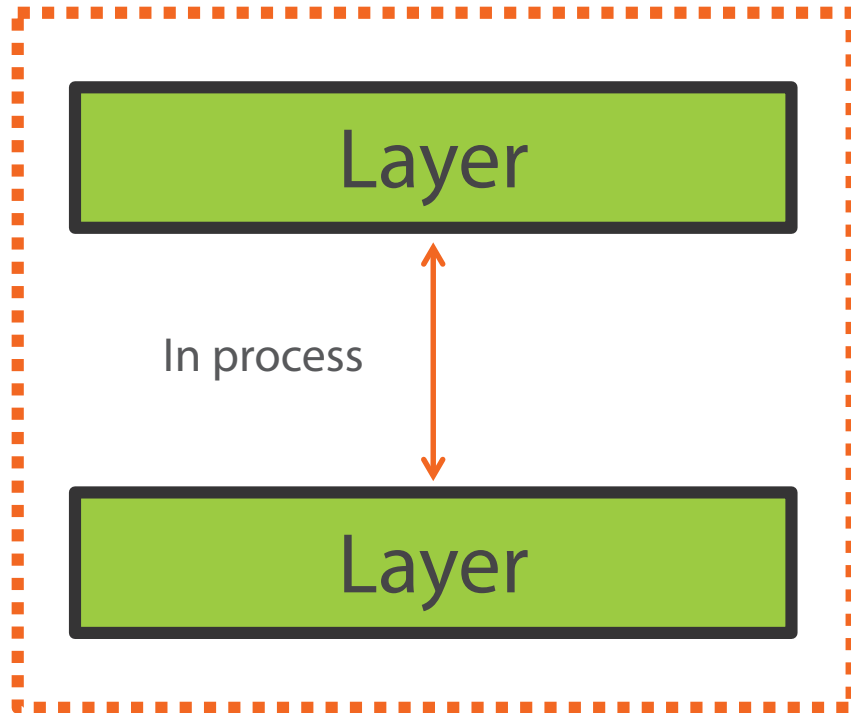


Messy tangle of instructions leading nowhere near to any flicker of solid software.

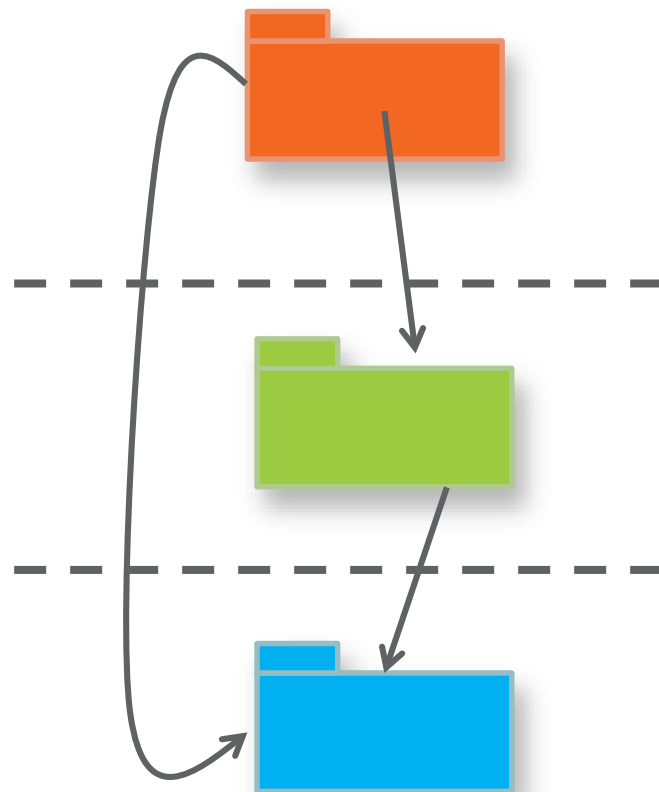


Layered block of modules easy to cut vertically and/or horizontally and easy to deploy.

Segments of Code



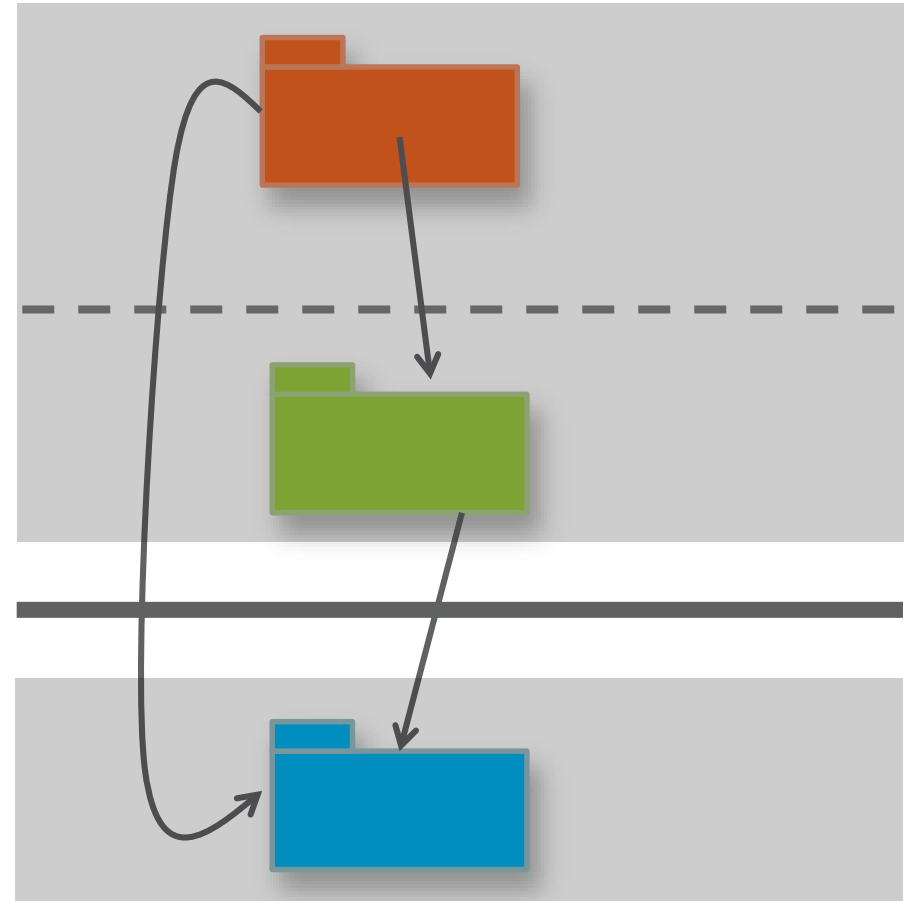
Classic 3-tier

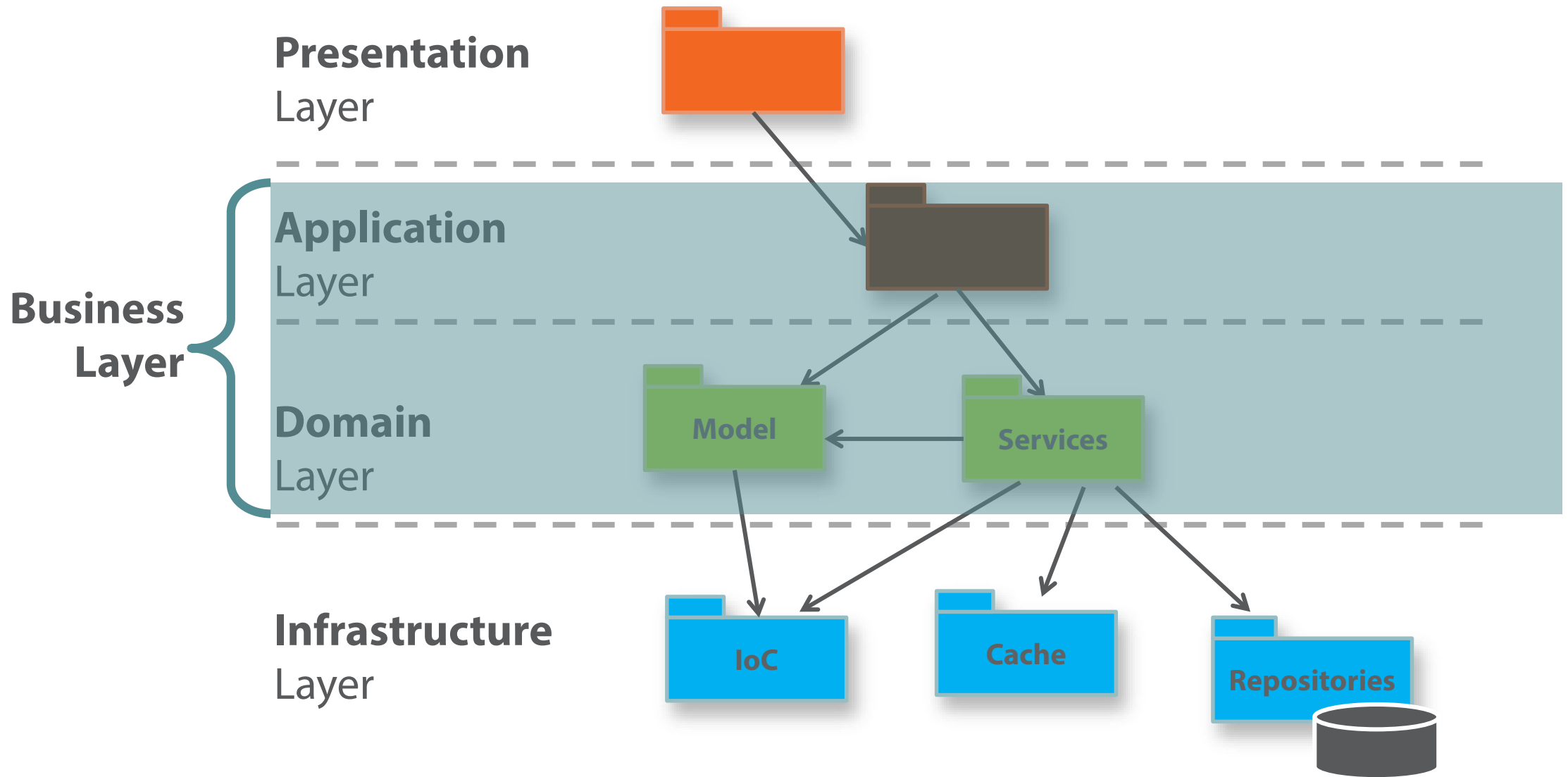


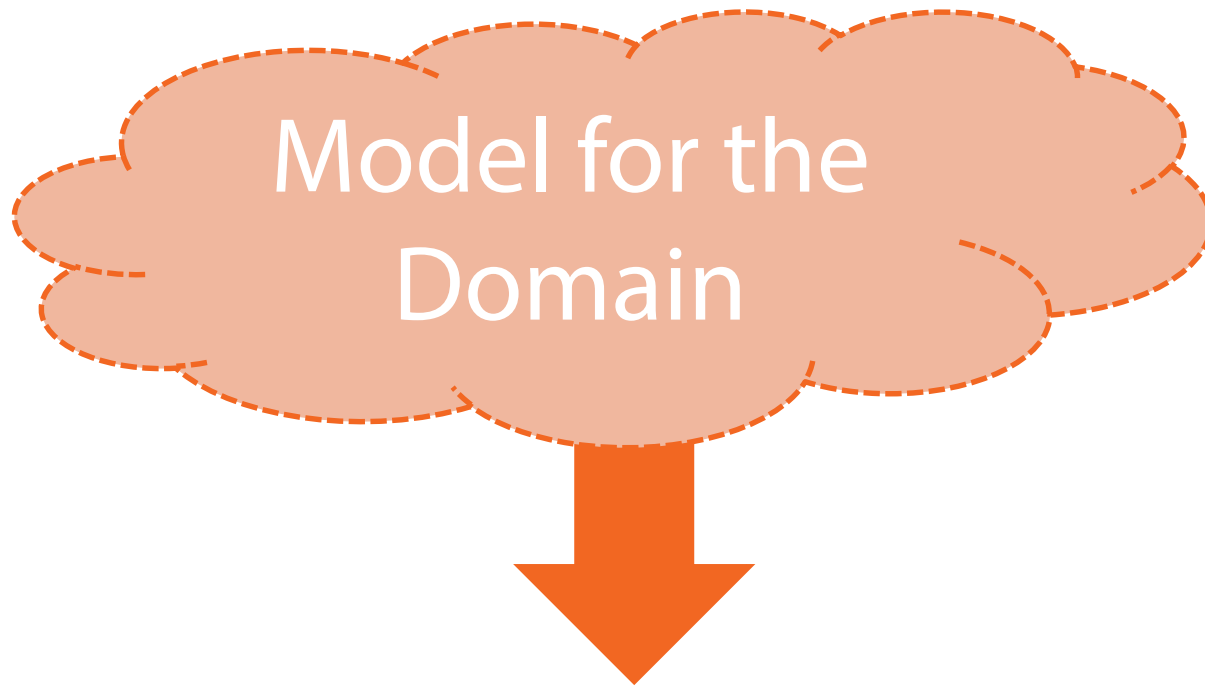
Presentation

Business

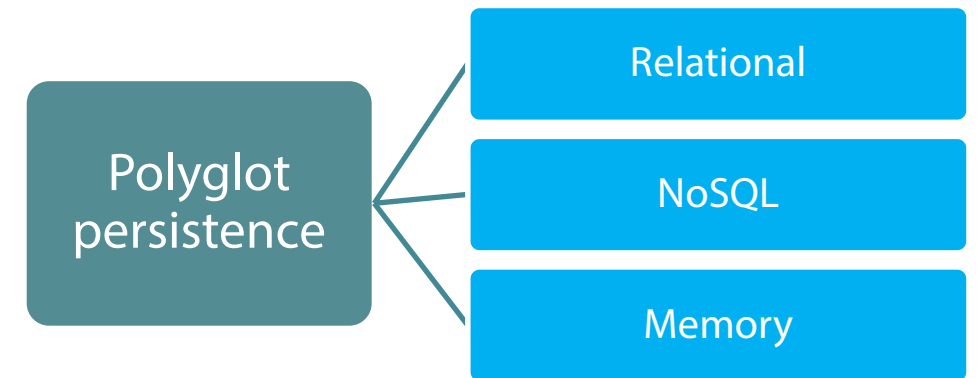
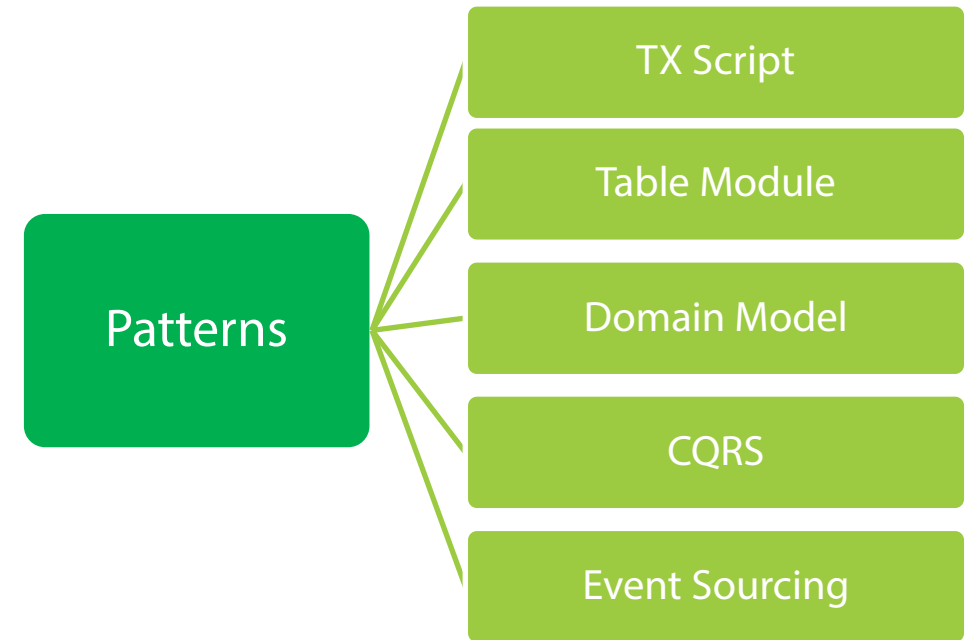
Data







Layered Architecture

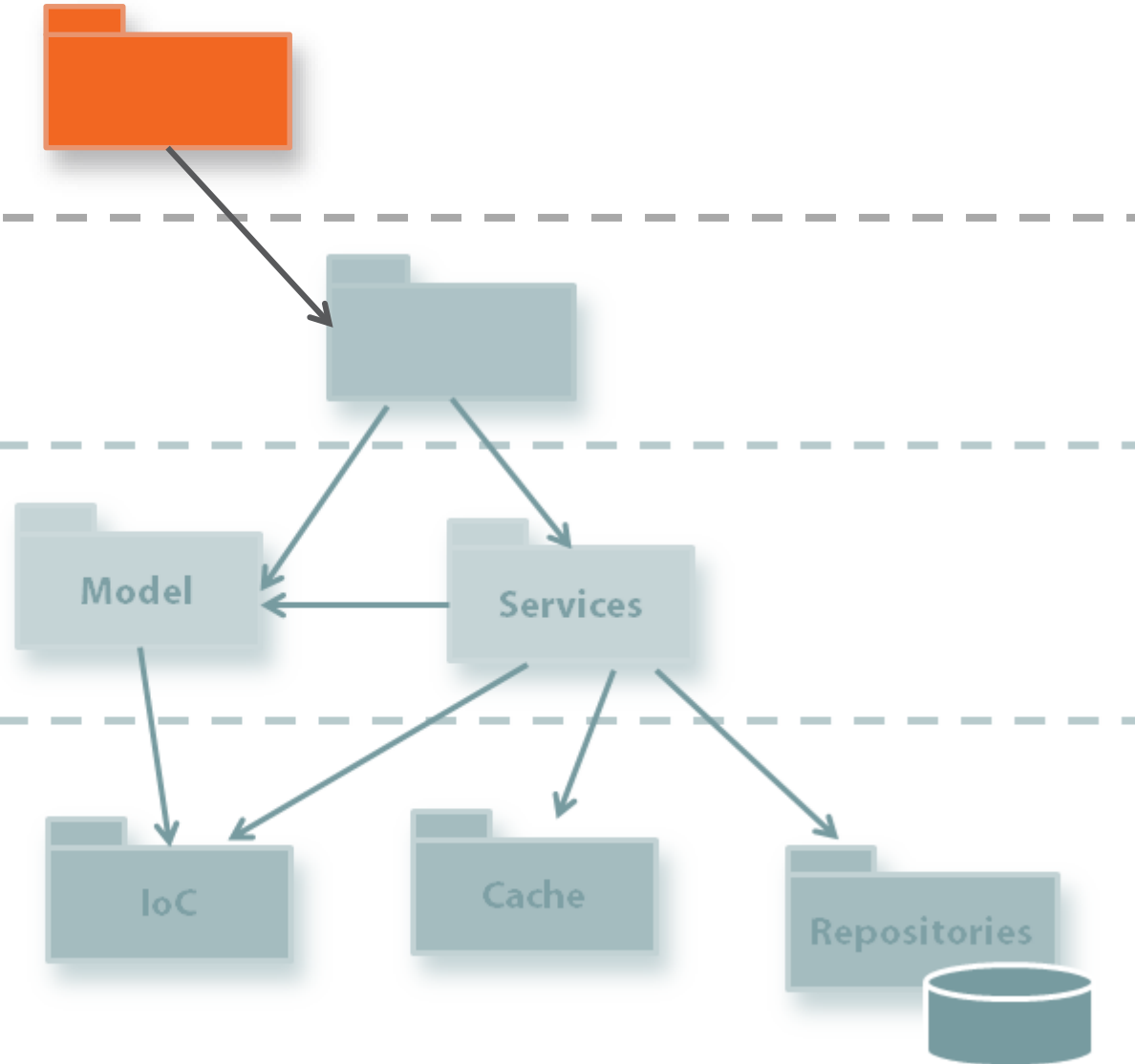


Presentation Layer

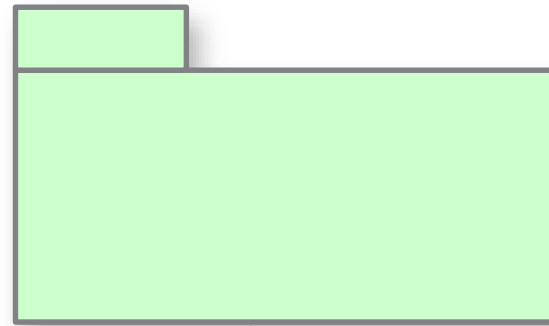
Application Layer

Domain Layer

Infrastructure Layer



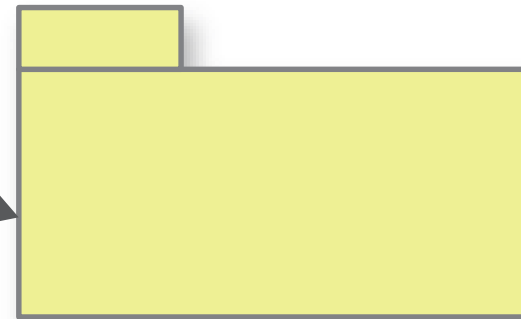
Presentation
Layer



View model

Application
Layer

Input model



Most Critical Part of Modern Applications

Responsible for providing the **user interface** to accomplish any required tasks.

Responsible for providing an effective, smooth and pleasant **user experience**.

Attributes of Presentation Layer



Presentation
Layer



Application
Layer



Domain
Layer



Infrastructure
Layer



Fifty Shades of Gray Areas

Where does code that formats data for presentation purposes really belong?

You said that. It's clearly a presentation concern.

To me, it sounds more like a business logic aspect.

Data is data and the database returns the data.

The Application Layer

Reports to the presentation



Serves ready-to-use data in the required form

Orchestrates tasks triggered by presentation elements



Use-cases of the application's frontend

Doubly-linked with presentation



Possibly extended or duplicated when a new frontend is added


```
public class HomeController
{
    private readonly IHomeApplicationService _service;
    public HomeController(IHomeApplicationService service)
    {
        _service = service;
    }
    public ActionResult Index()
    {
        var model = _service.FillHomePage( /* input model */ );
        return View(model);
    }
    ...
}
```

Orchestrator

```
public class InputModel
{
    string Id {get; set;}
}

public class CustomerService
{
    ViewModel Search(InputModel m);
}
```

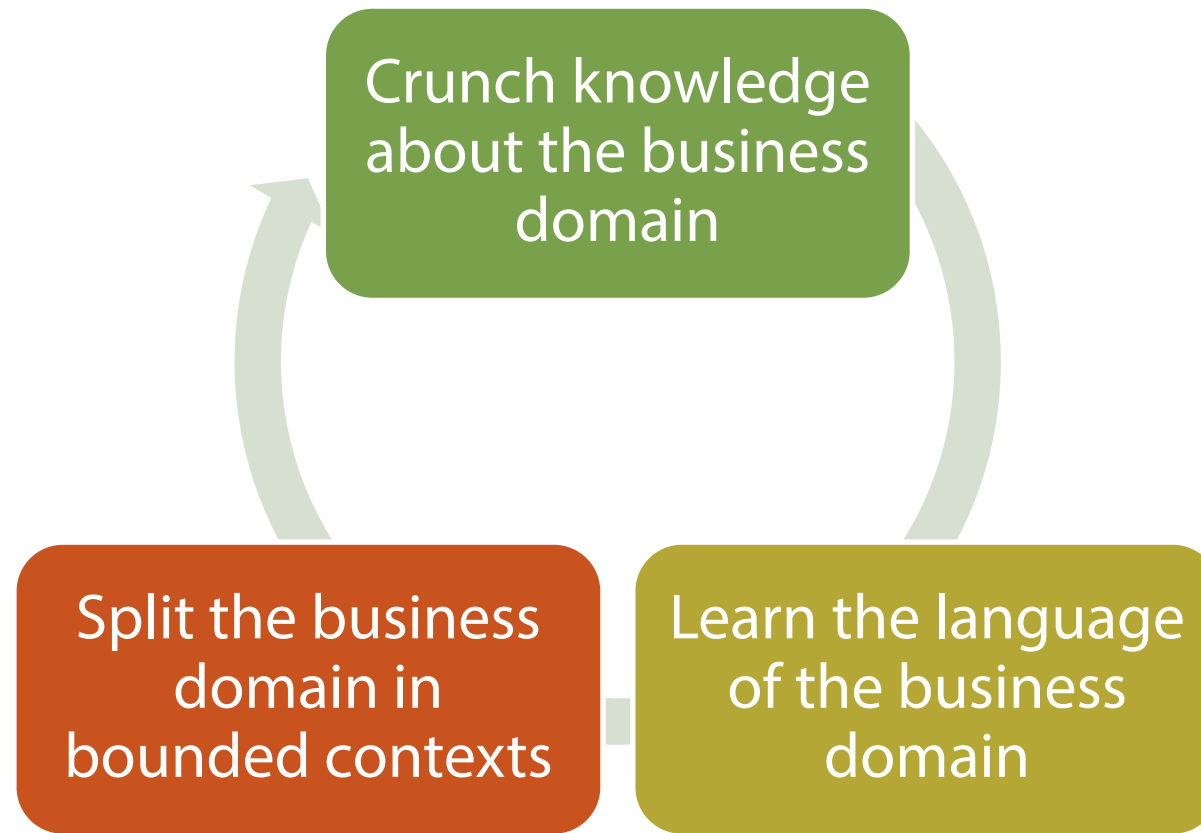
```
public class ViewModel
{
    string Name {get; set;}
    string Address {get; set;}
    string City {get; set;}
    IList<OrderDto> Orders {get; set;}
}
```

Find Customer

Find Customer

Name
Address
City

Order #1
Order #2
Order #3



What's Next?

It's all about implementing all
business rules and organizing
the business logic.

Business Logic—An Abstract Definition

Application Logic

Dependent on use-cases

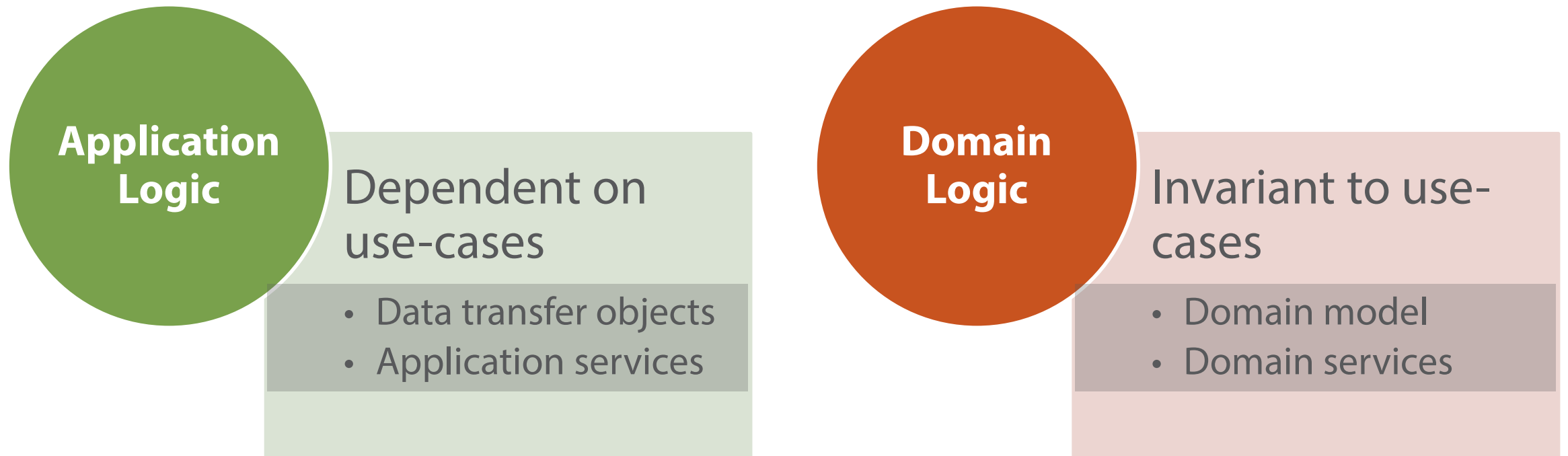
- Application entities
- Application workflow components

Domain Logic

Invariant to use-cases

- Business entities
- Business workflow components

Business Logic—DDD Definition



BUSINESS RULE

Statements that detail the implementation of a business process or describe a business policy to be taken into account.

PATTERNS for organizing the business logic

Domain logic is all about baking business rules into the code

Common Patterns

**TX
SCRIPT**

**TABLE
MODULE**

**DOMAIN
MODEL**

Transaction Script Pattern

System actions

- Each procedure handles a single task



Logical transaction

- end-to-end from presentation to data



Common subtasks

- split into bounded sub-procedures for reuse

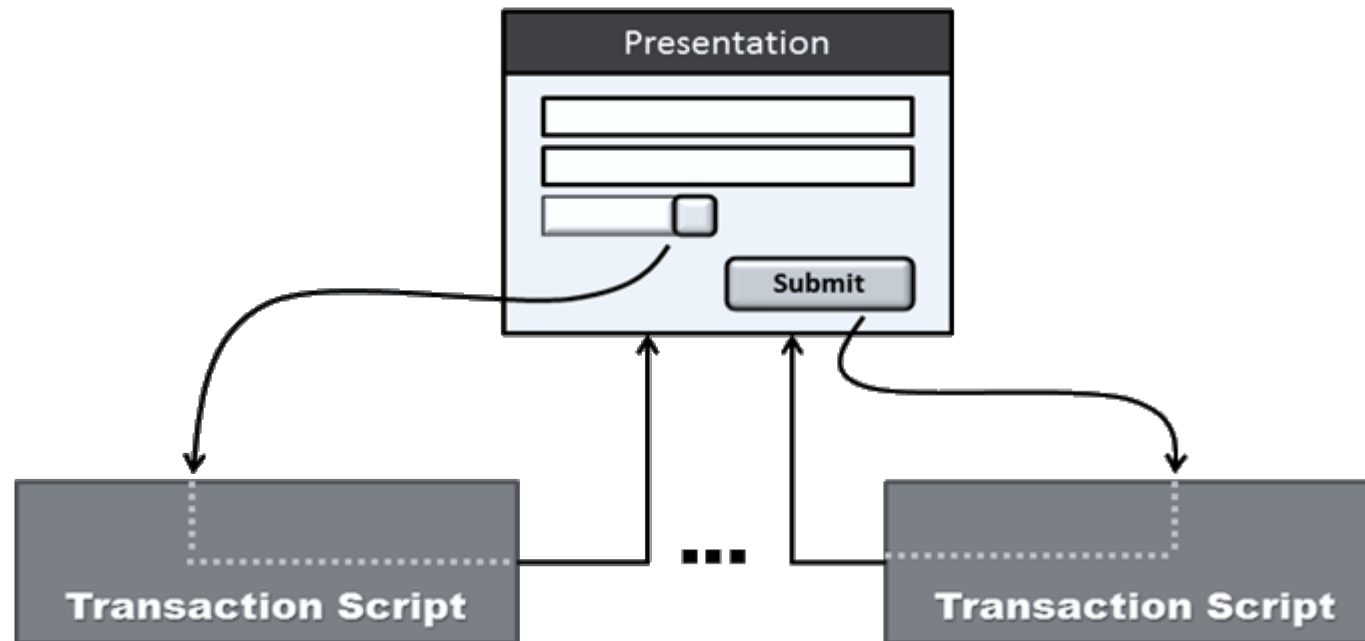
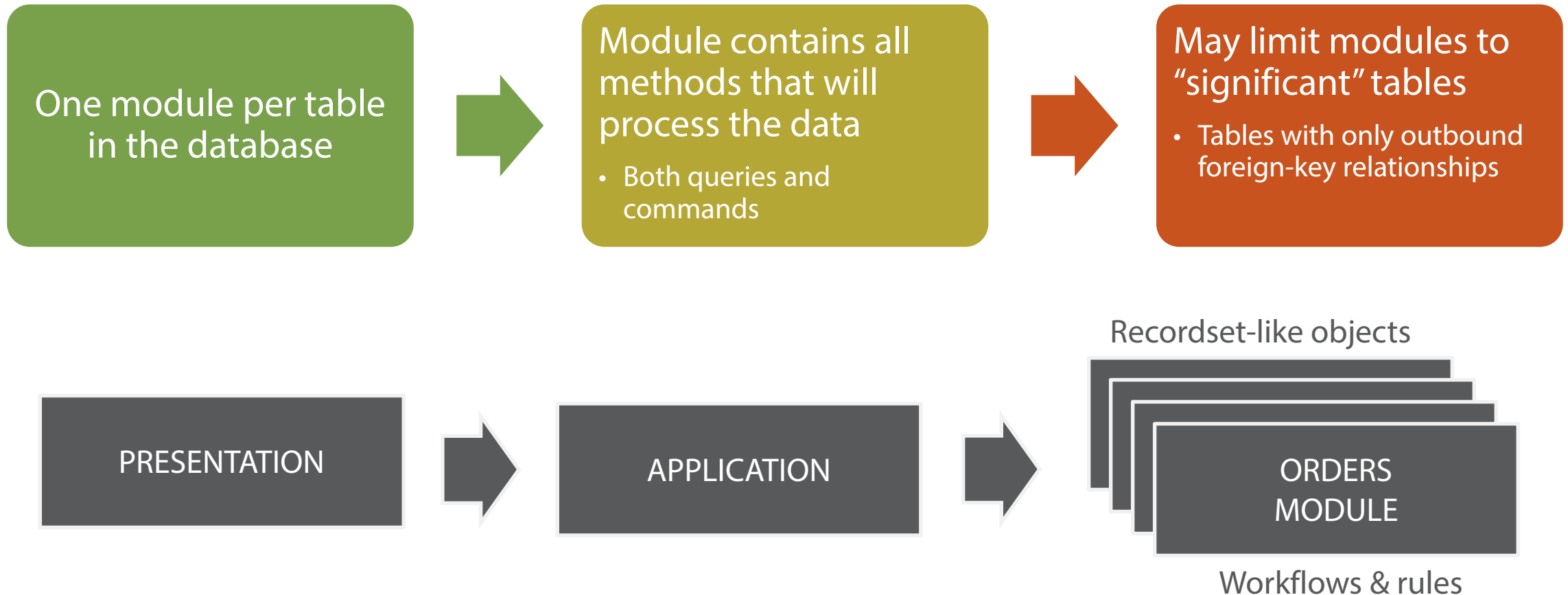
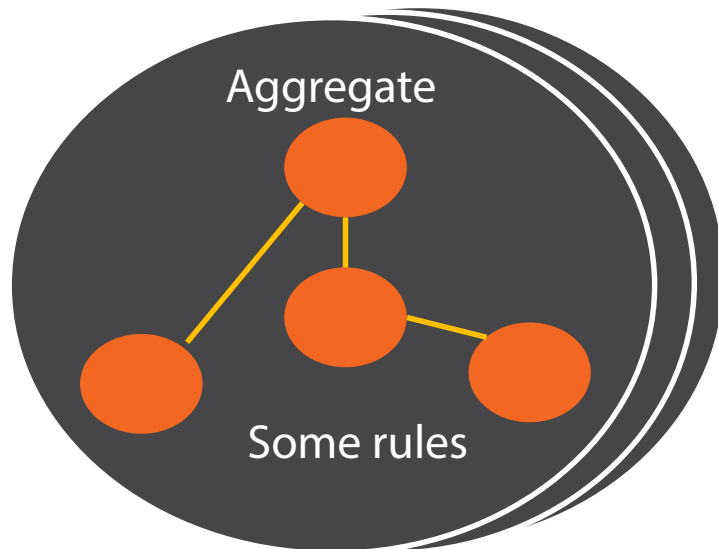


Table Module Pattern



Domain Model Pattern



Domain Layer

Logic invariant to use-cases

- Domain model
- Domain services

Not necessarily

an implementation of the **Domain Model** pattern

Takes care

of persistence tasks

Domain Model

Models for
the business
domain

Object-oriented entity model

Functional model

Guidelines for
classes in an
entity model

DDD conventions (factories, value types, private setters)

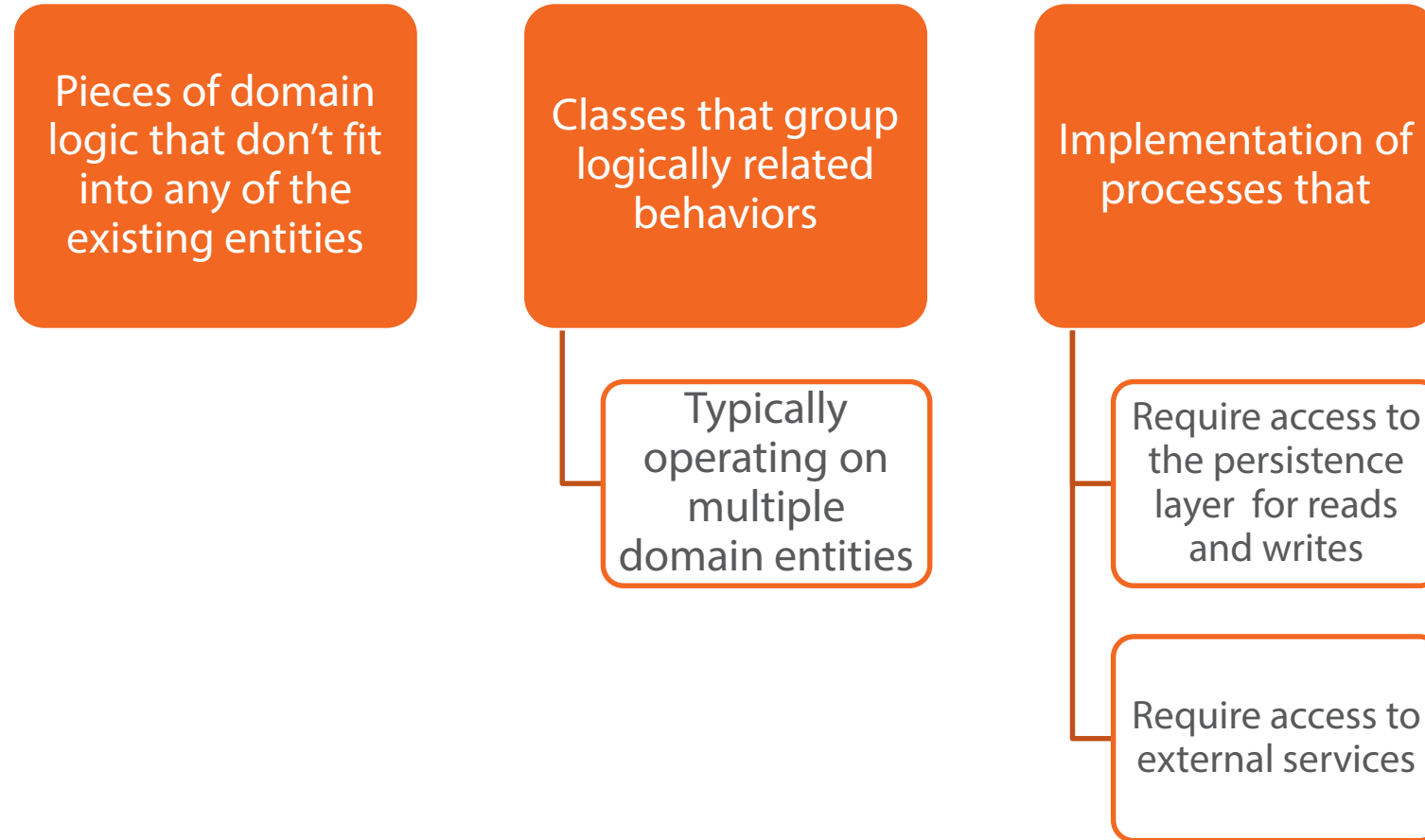
Data and behavior

Anemic
model

Plain data containers

Behavior and rules **moved** to domain services

Domain Services



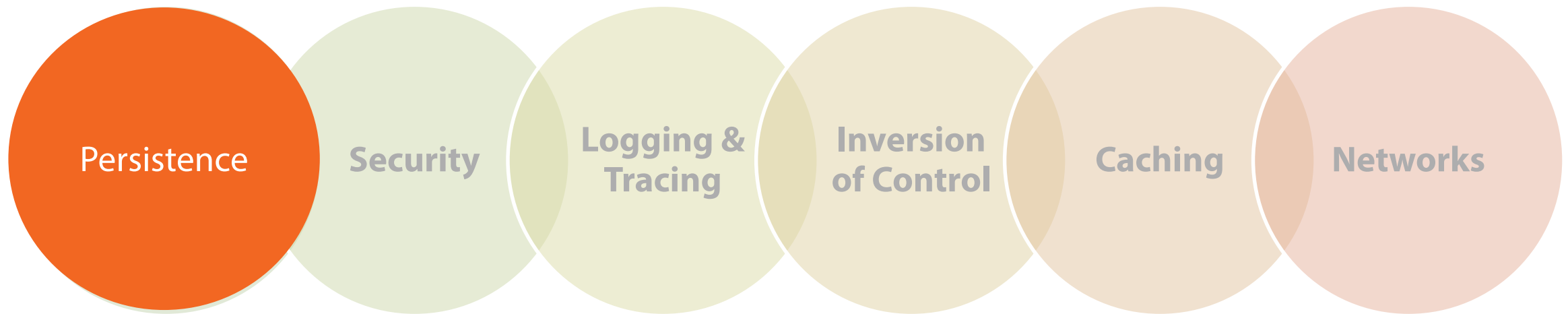


Infrastructure (software)

Set of the fundamental facilities needed for the operation of a software system.

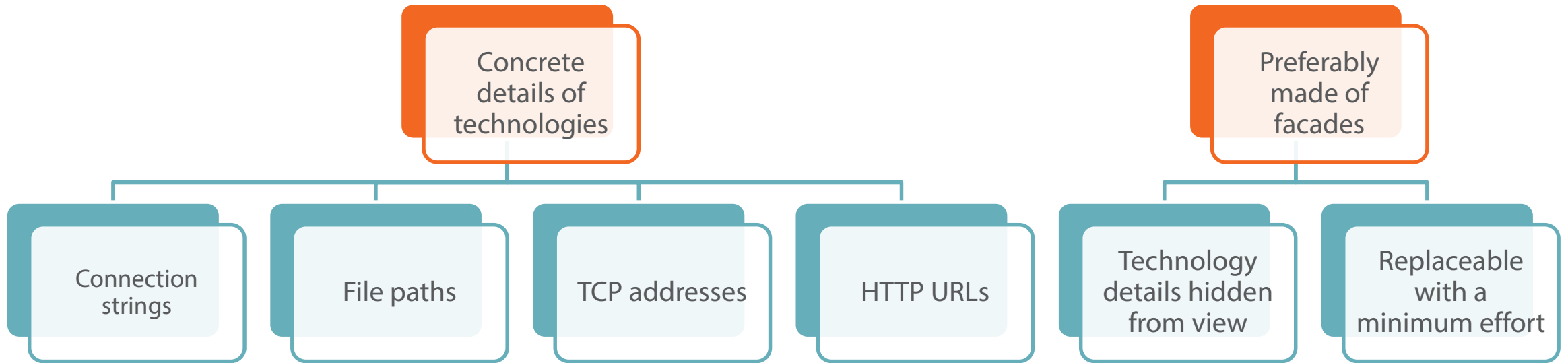


Fundamental Facilities of Software Systems



Down Where Technologies Belong

Necessary to fuel the system



Not binding the system to specific products

What's Next?