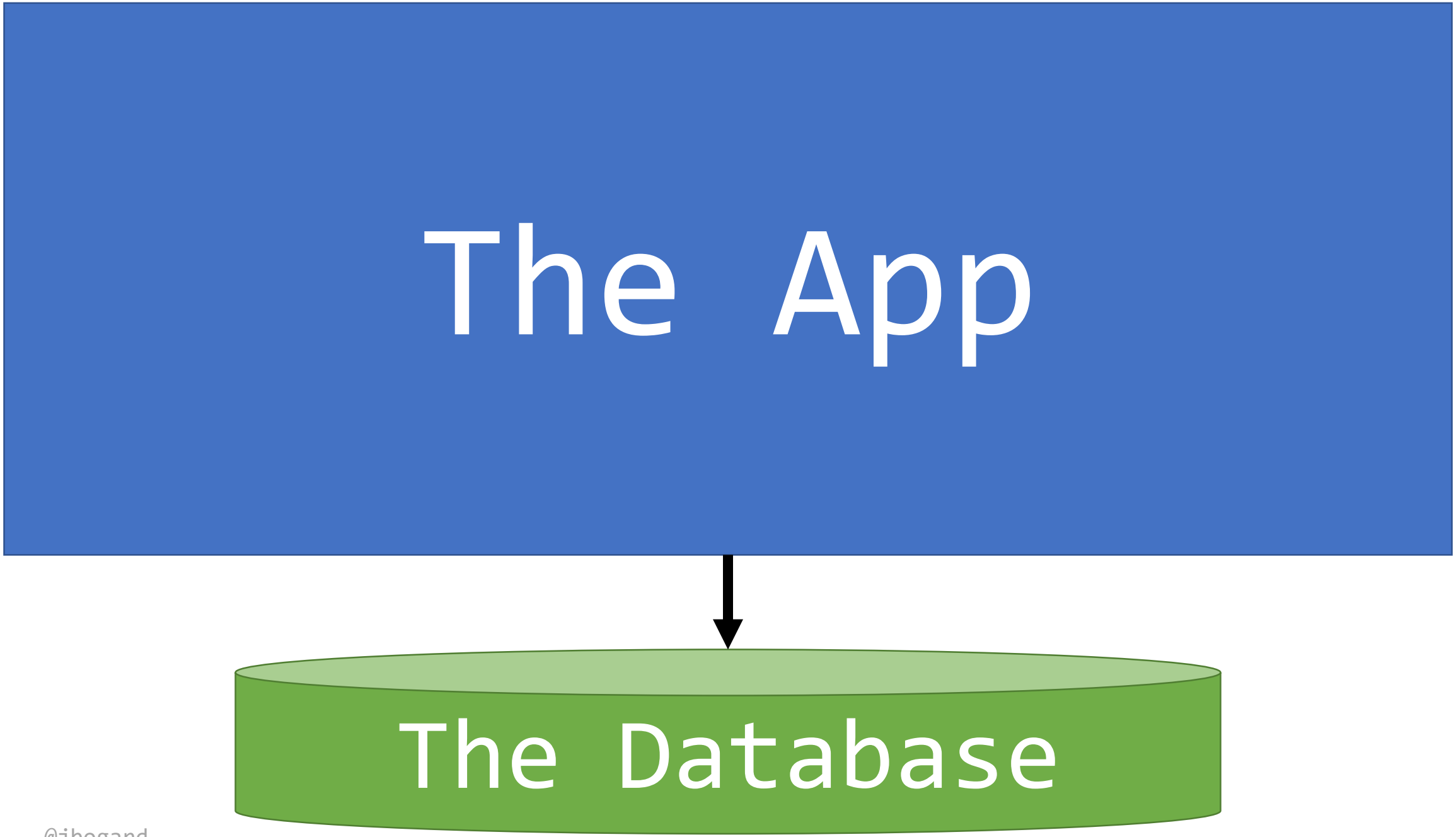
Modularizing the Monolith

Jimmy Bogard
@jbogard
github.com/jbogard
jimmybogard.com

How did we get here?

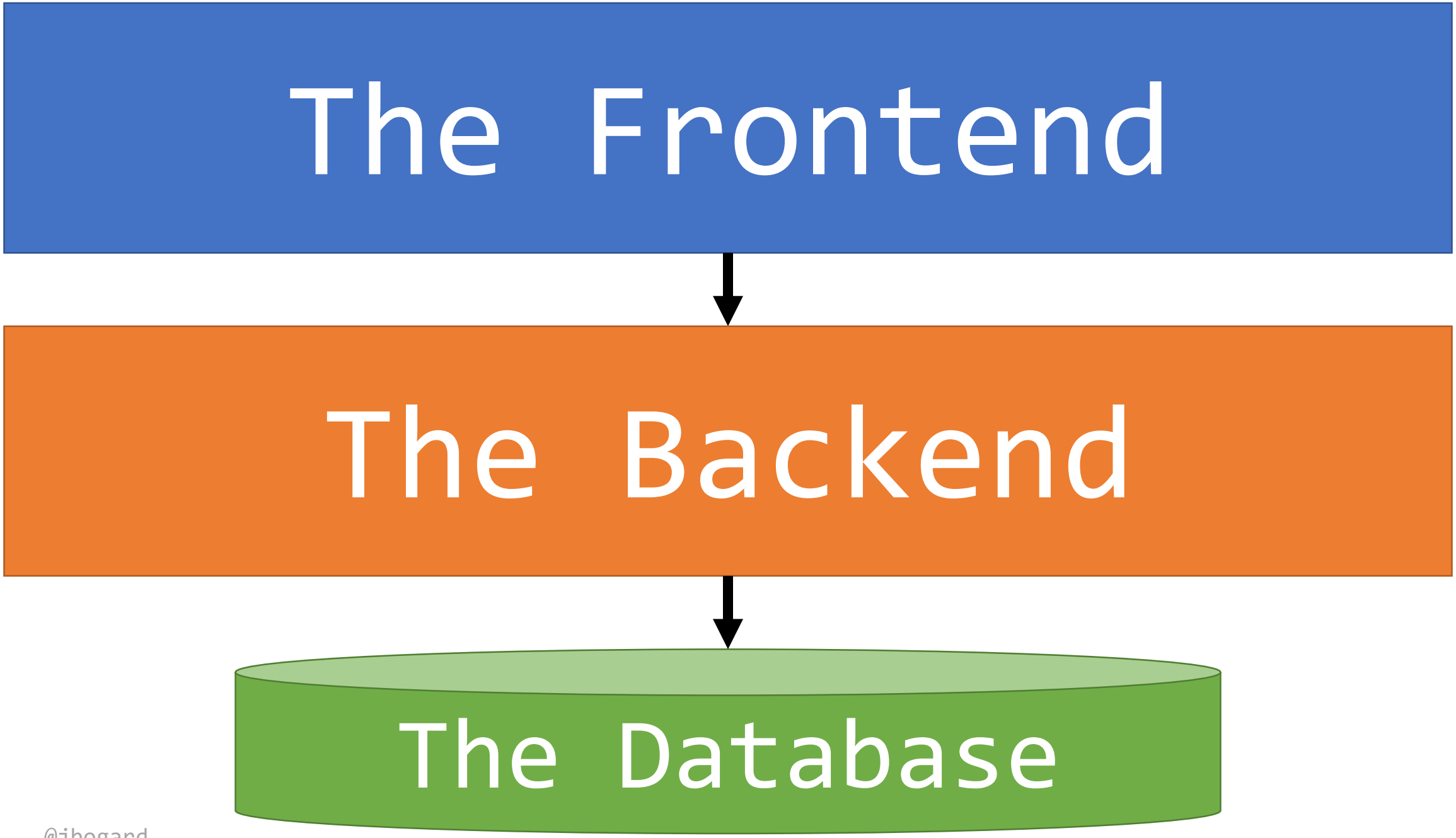

The App
The Database

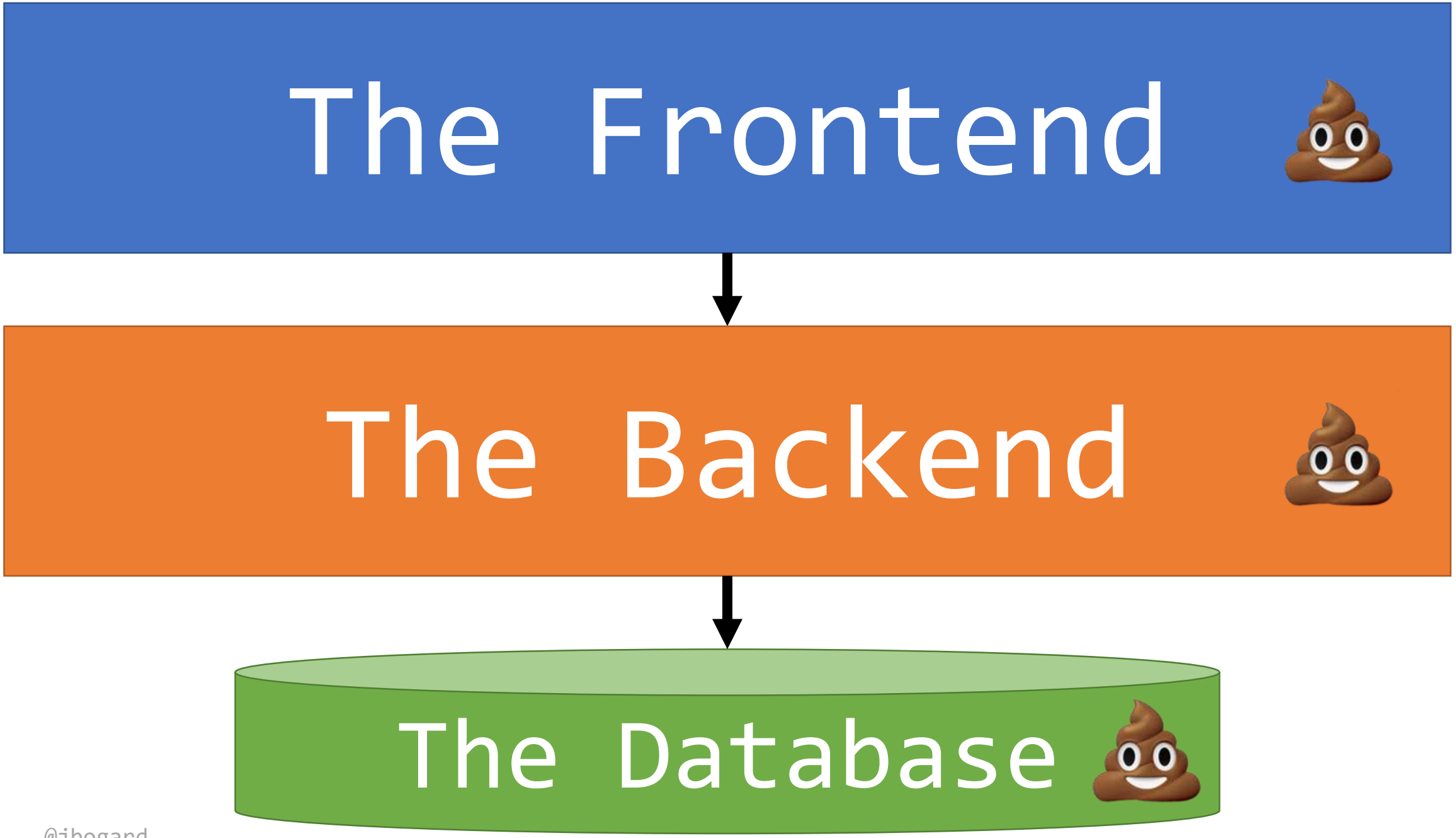
The App

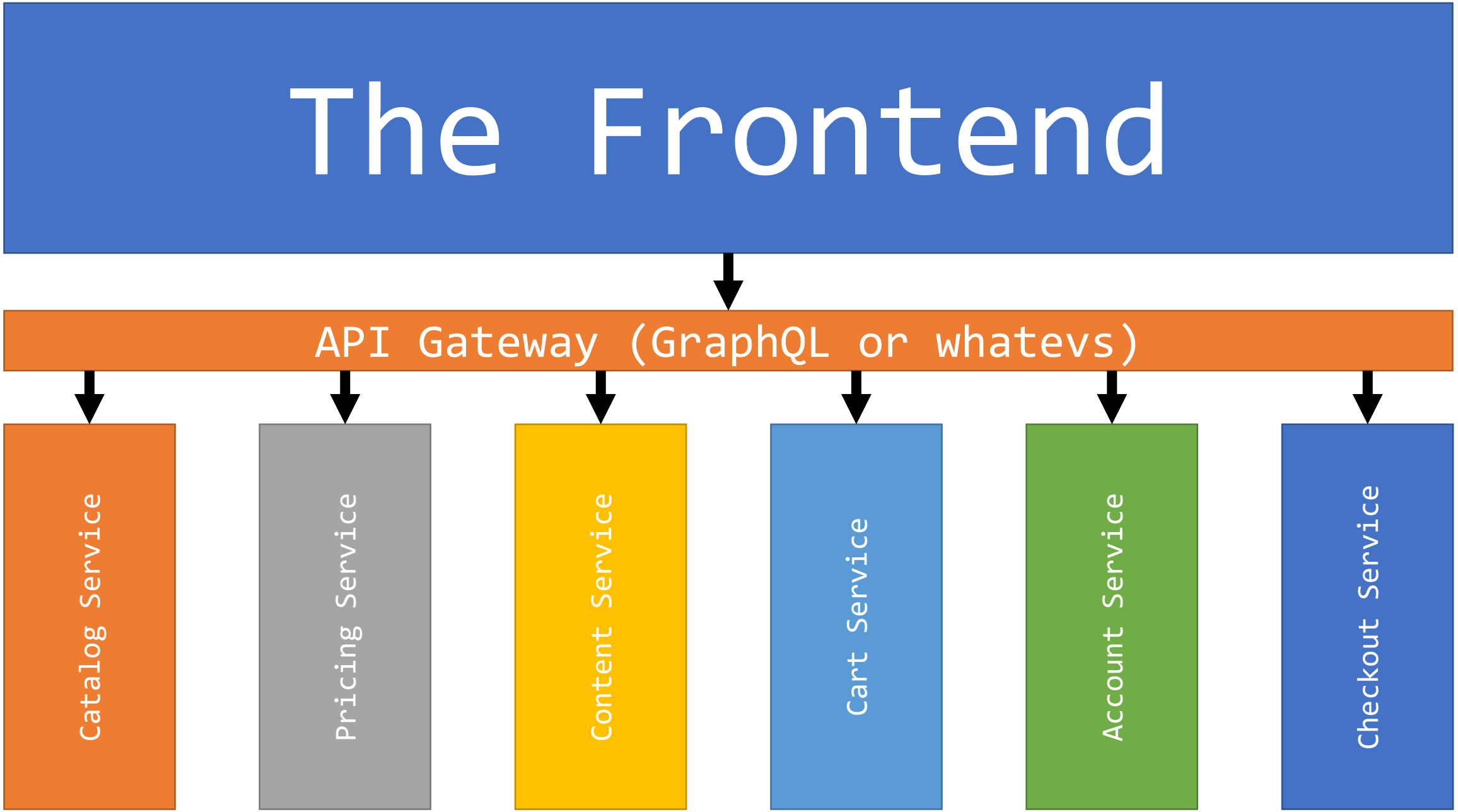
The Database

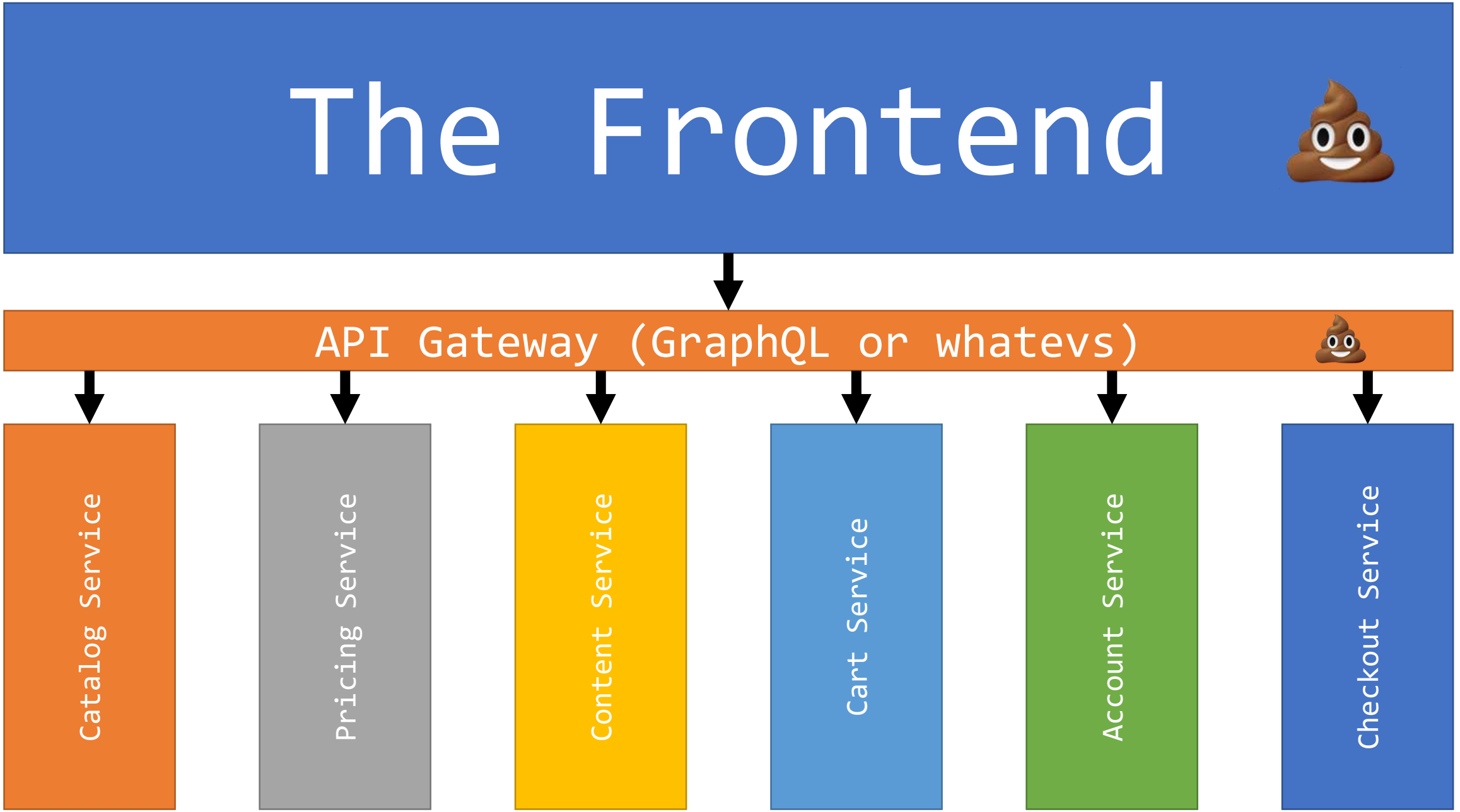
The App

The Database

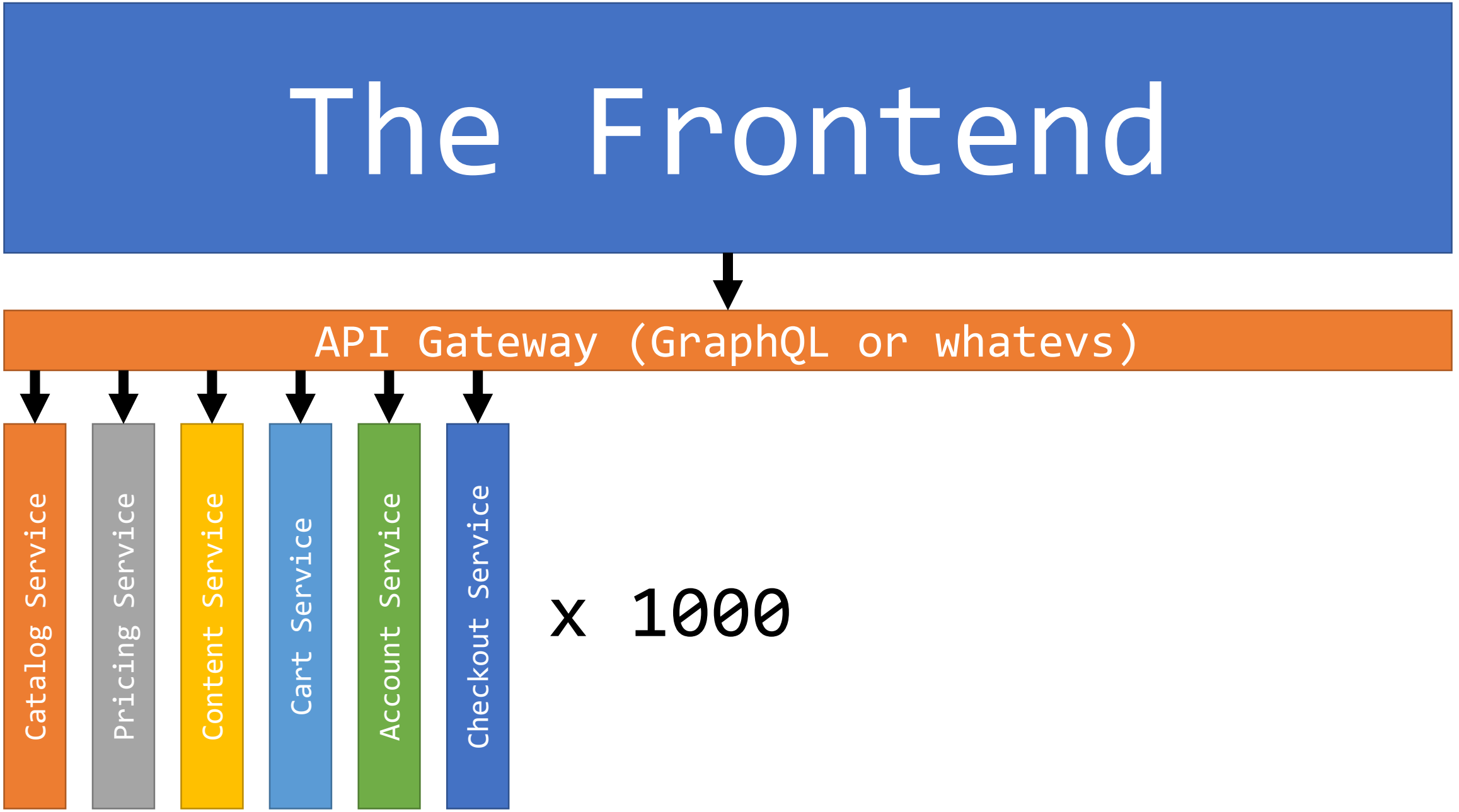

The Frontend
The Backend
The Database


The Frontend
The Backend
The Database
@jbogard


The Frontend
API Gateway (GraphQL or whatevs)
Catalog Service
Pricing Service
Content Service
Cart Service
Account Service
Checkout Service
@jbogard


The Frontend
API Gateway (GraphQL or whatevs)
Catalog Service
Pricing Service
Content Service
Cart Service
Account Service
Checkout Service
@jbogard

But that’s not what happened…

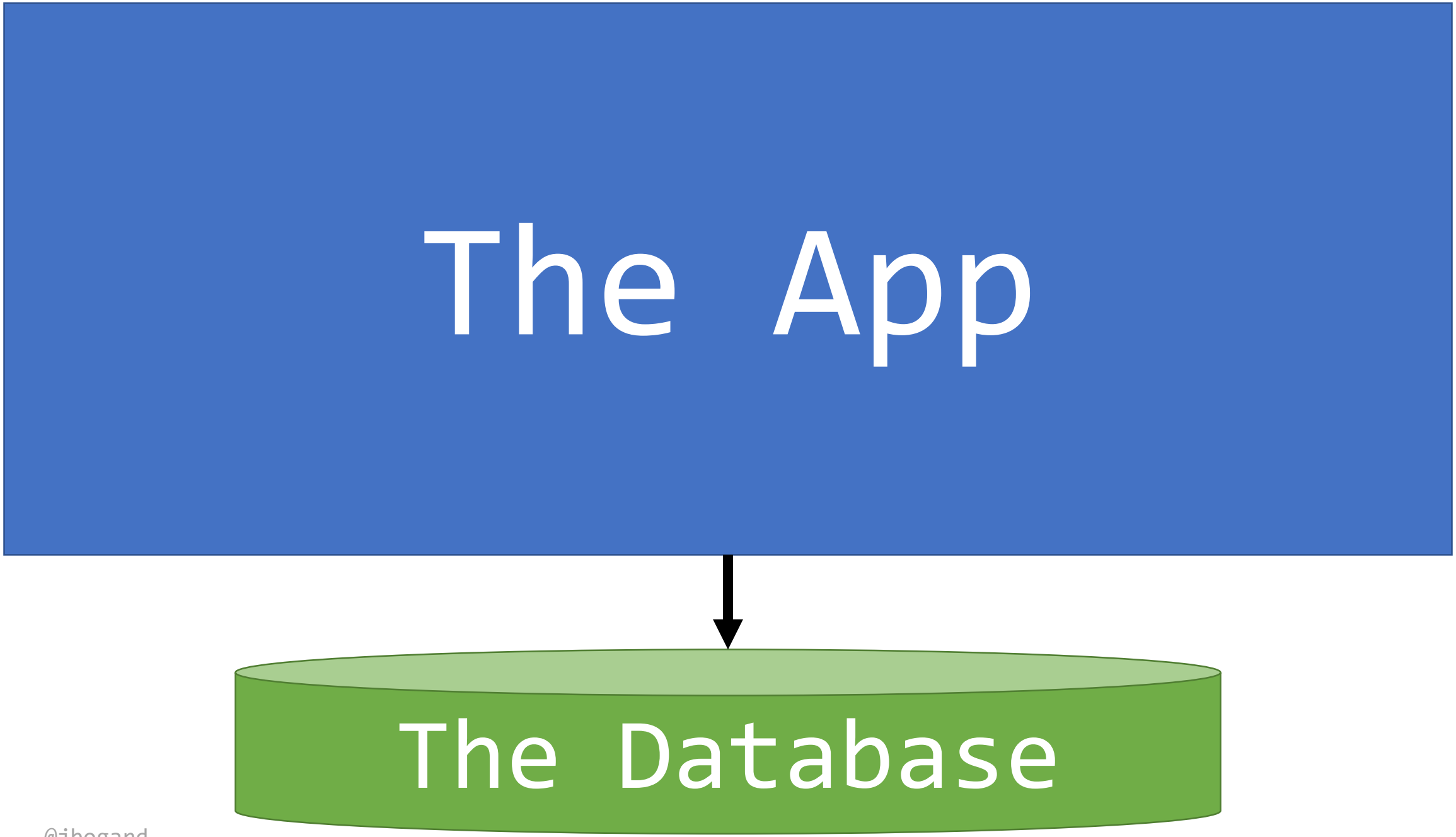

The Frontend
API Gateway (GraphQL or whatevs)
Catalog Service
Pricing Service
Content Service
Cart Service
Account Service
Checkout Service
x 1000
@jbogard

Over-emphasized “micro”
Misunderstood “service”

Boundaries are hard

Distributed systems are harder

Doing both is harder-er


The App
The Database

The App

The Database

Monolith:

System with exactly one unit of deployment

Bad Monolith:

Software whose design, information model, and interface combine multiple competing and interfering domains into one single application and data model.

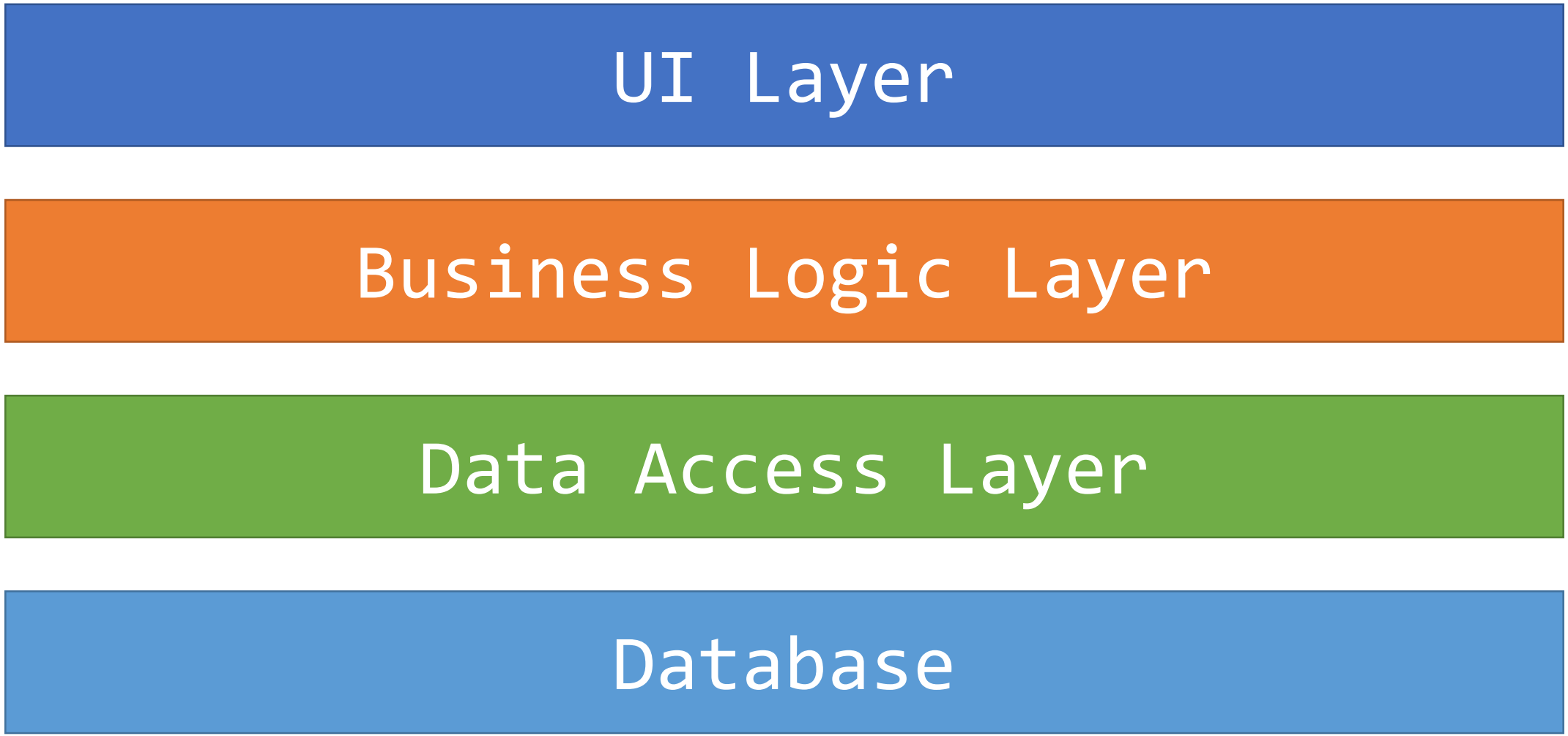
AKA “Big Ball of Mud”

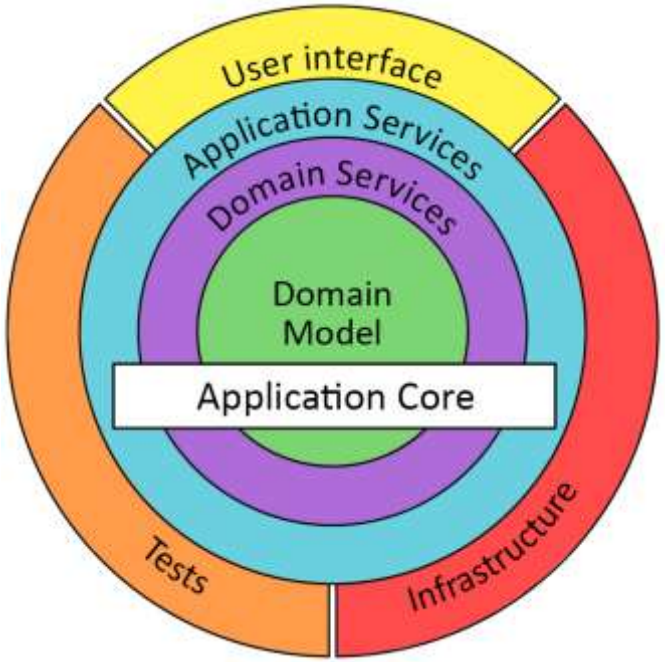
Good Monolith:

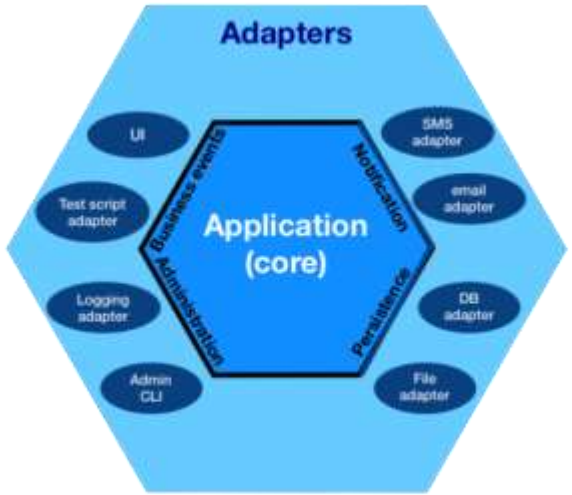
Monolith that is not bad

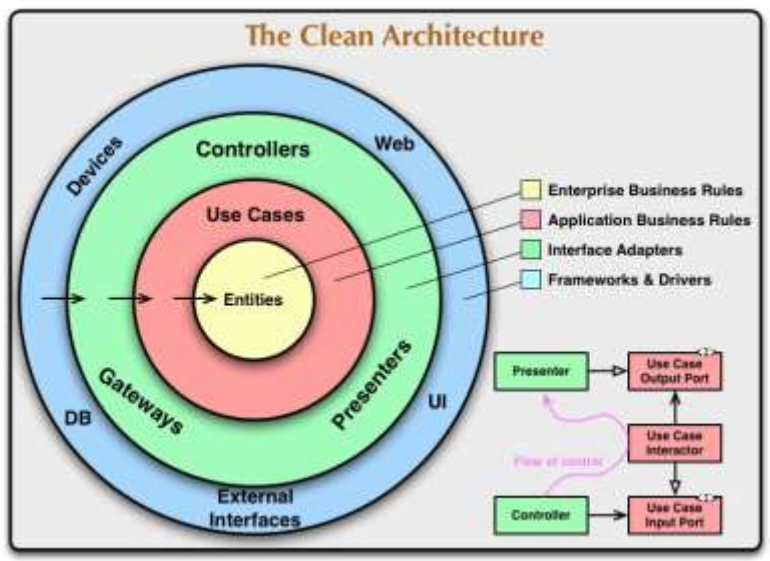
Good Monolith:

Monolith that is easy to change


UI Layer
Business Logic Layer
Data Access Layer
Database

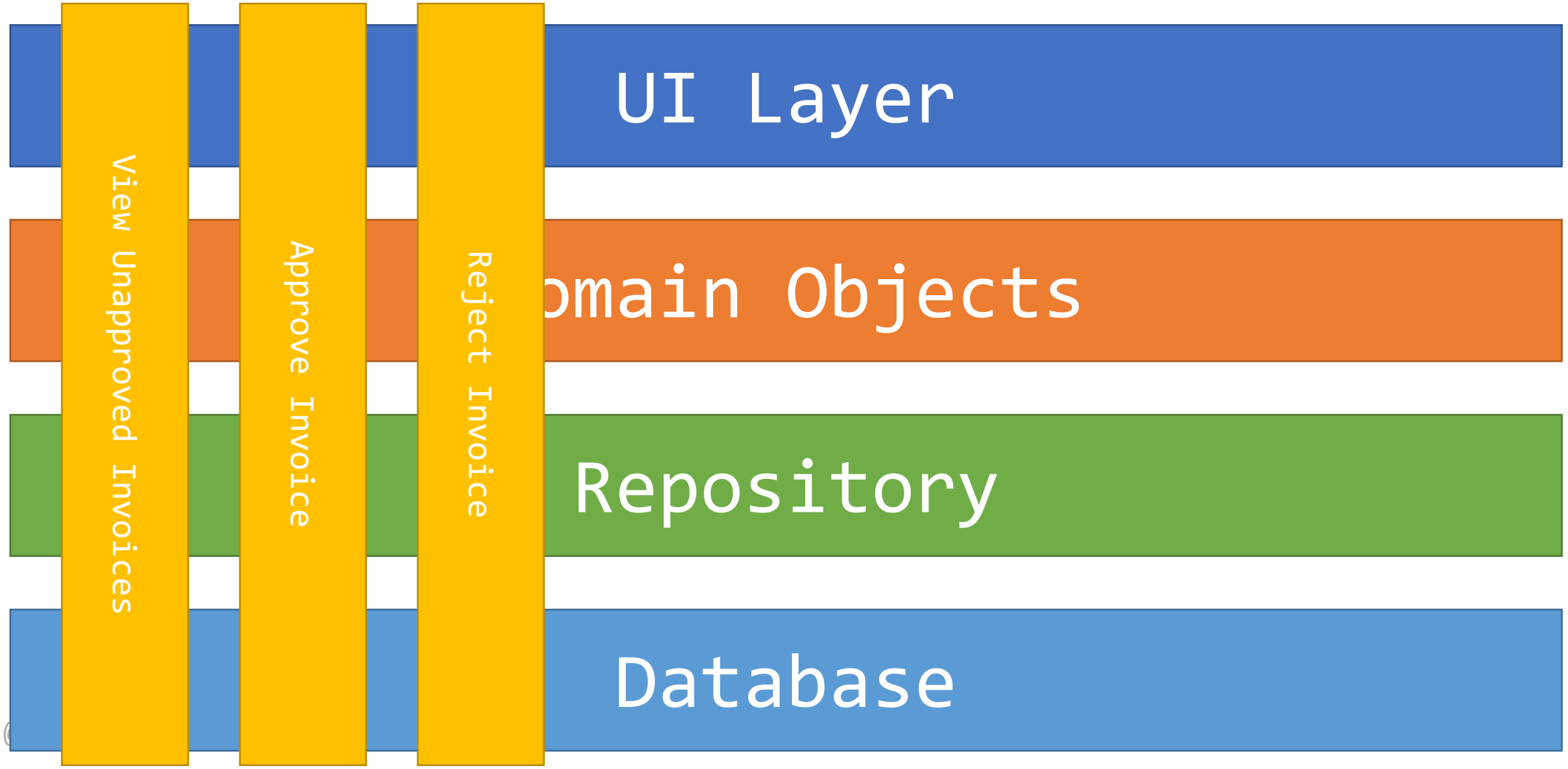

User interface
Application Services
Domain Services
Domain Model
Application Core
Tests
Infrastructure


Adapters
UI
SMS adapter
Test script adapter
email adapter
Application (core)
Business events
Notification
Administration
Persistence
Logging adapter
DB adapter
Admin CLI
File adapter


The Clean Architecture
Devices
Controllers
Web
Use Cases
Entities
Gateways
Presenters
DB
UI
External Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers
Presenter
Use Case Output Port
Use Case Interactor
Controller
Use Case Input Port

Solution: We need **boundaries**

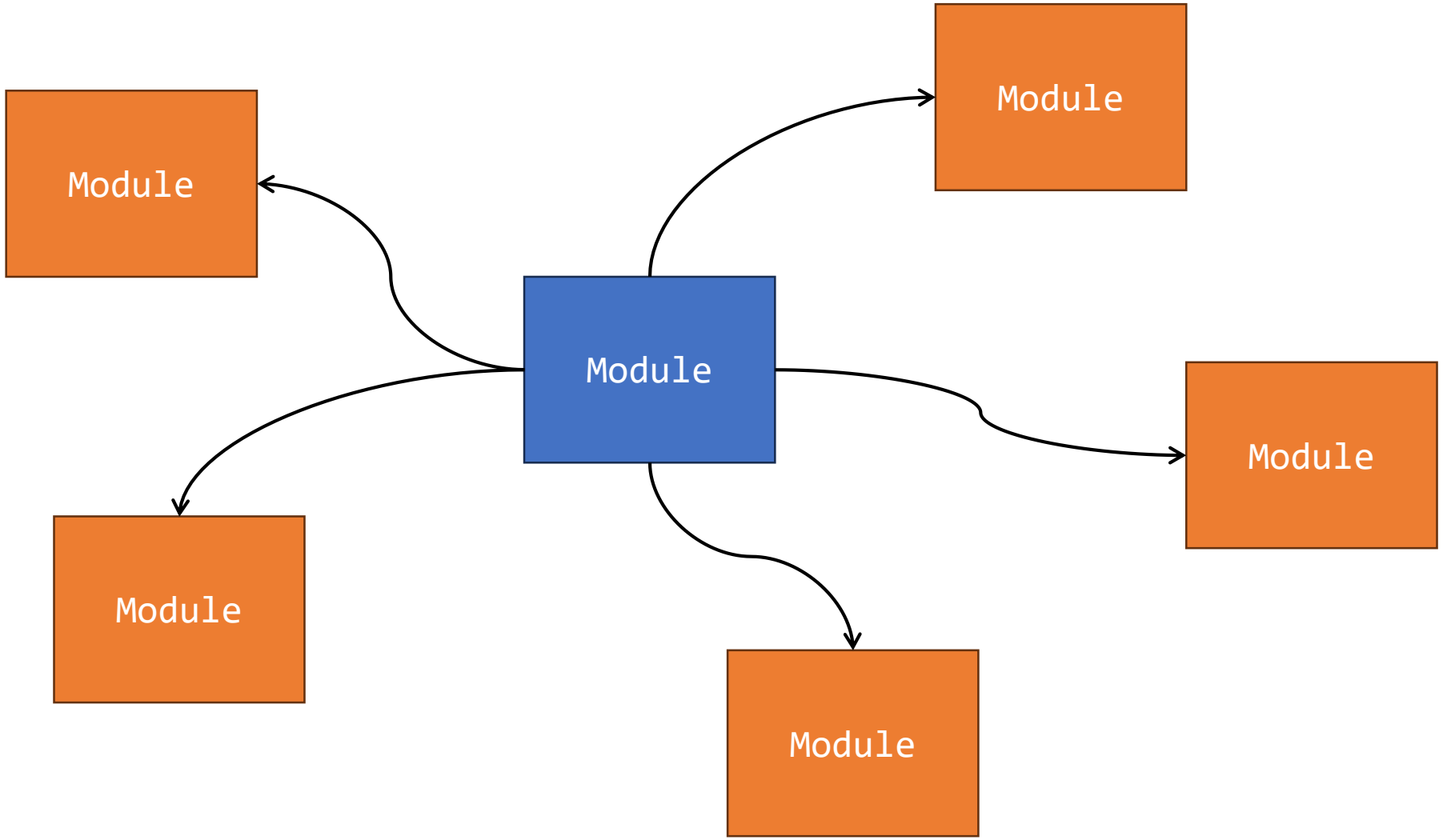
Vertical Slice Architecture



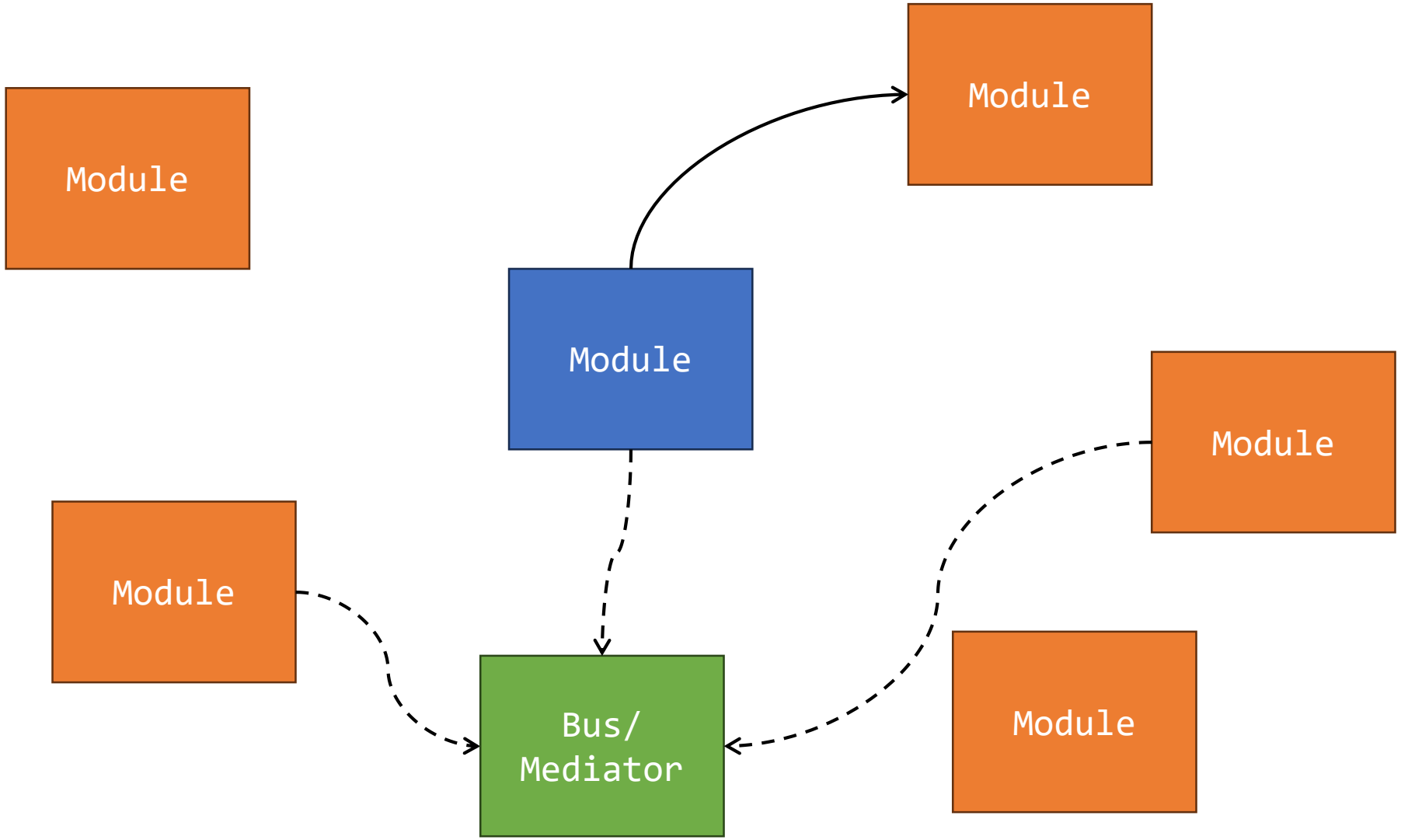
Modules must:

- Be independent and interchangeable
- Contain all logic and data to produce desired functionality
- Have a defined interface

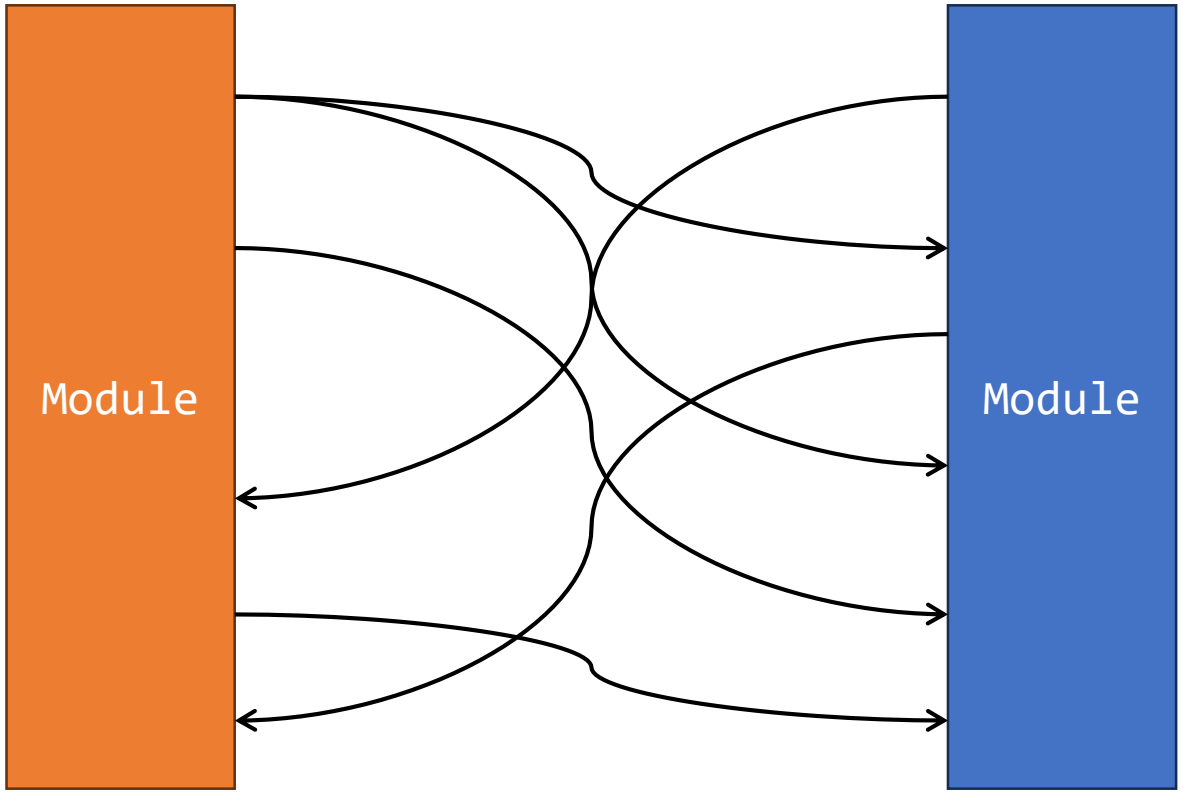
Independent



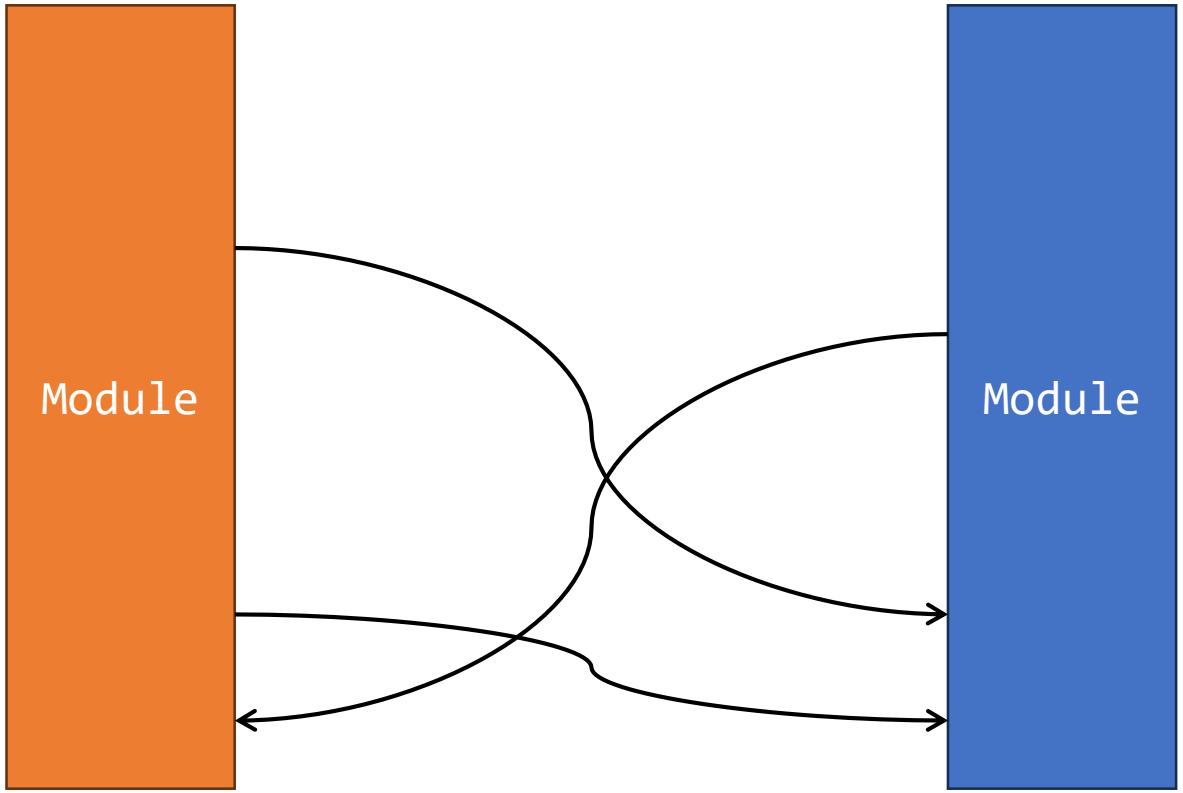
Independent



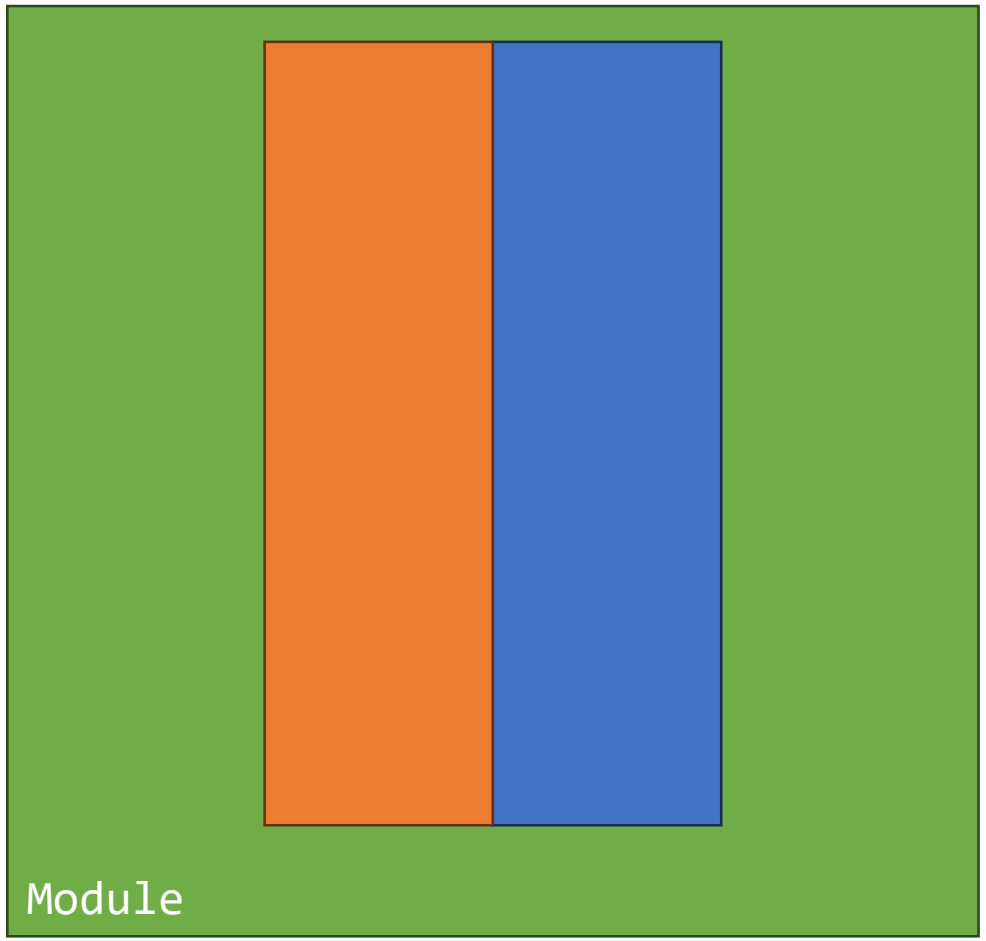
Strong Dependency



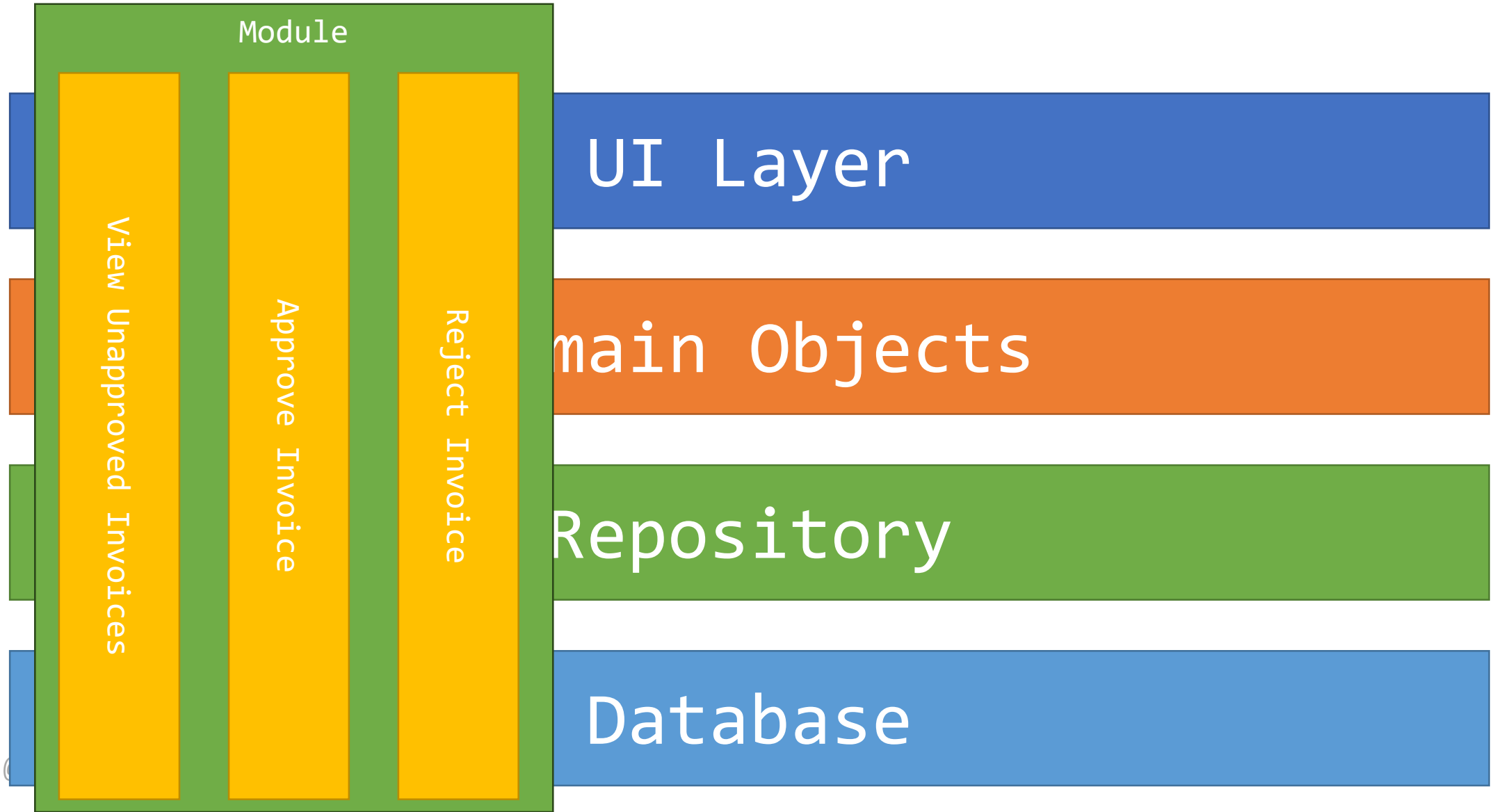
Weak Dependency



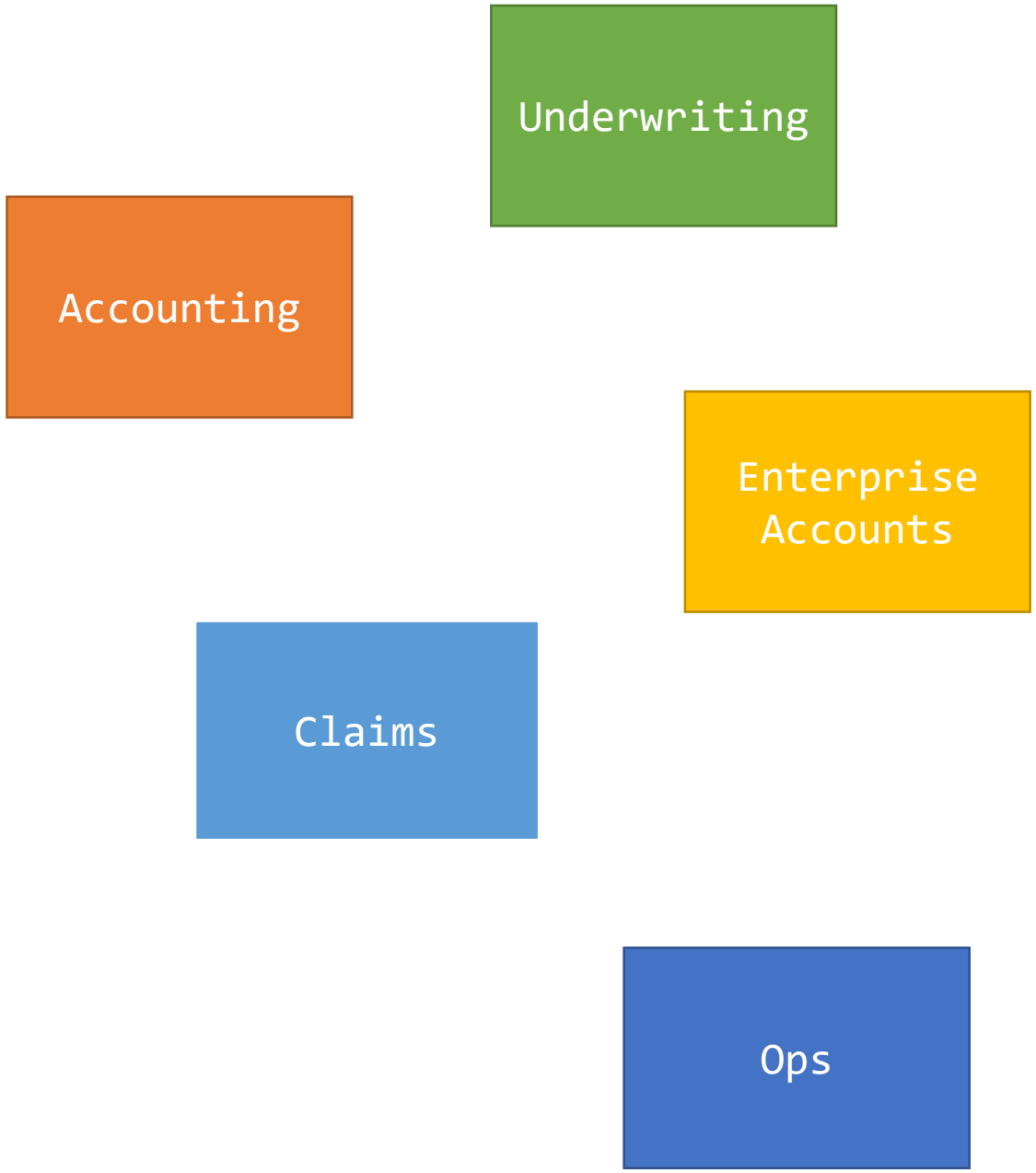
Merged Modules



Slices into Modules



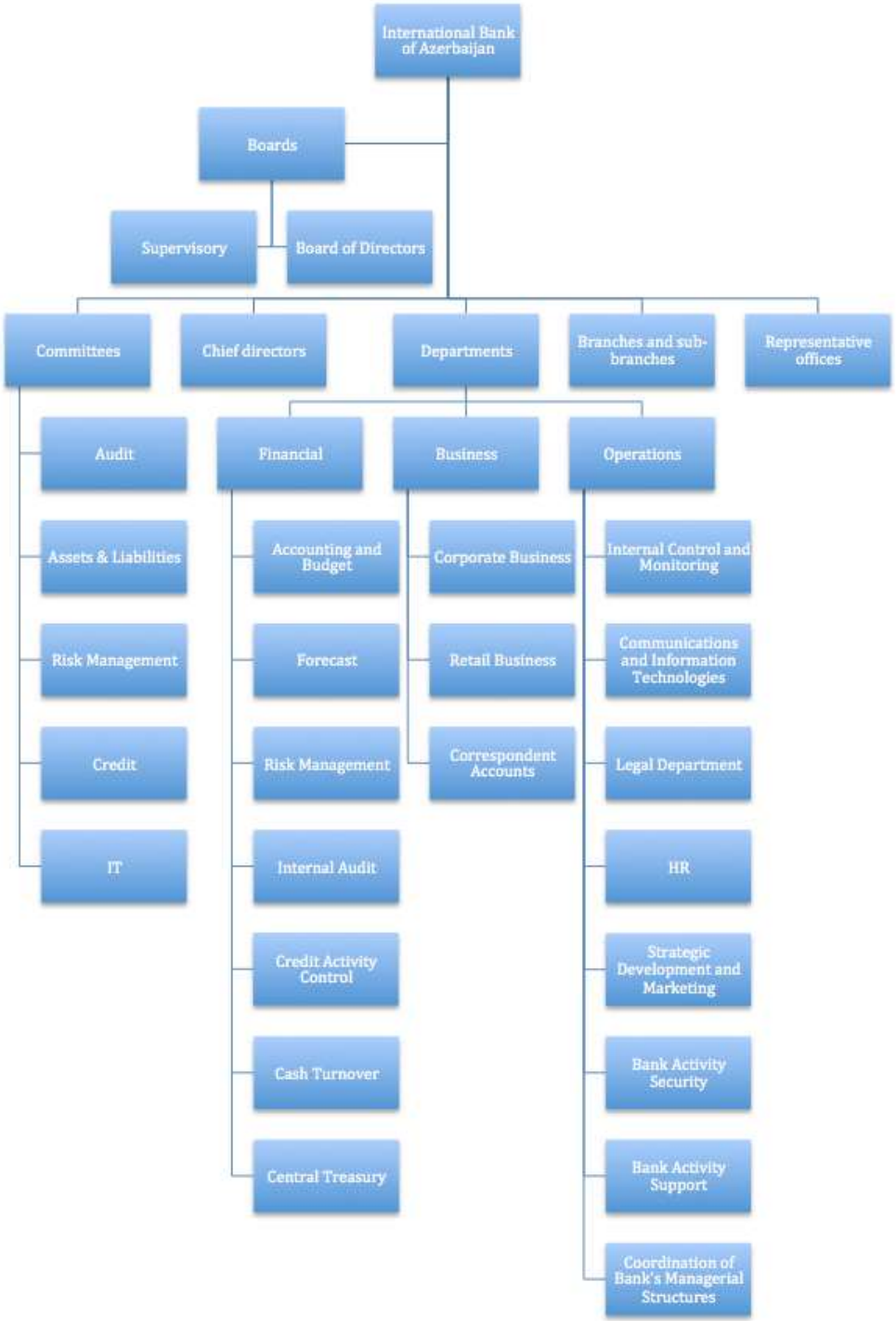
Defining Boundaries



Boundaries are easier to draw when the source is tangible

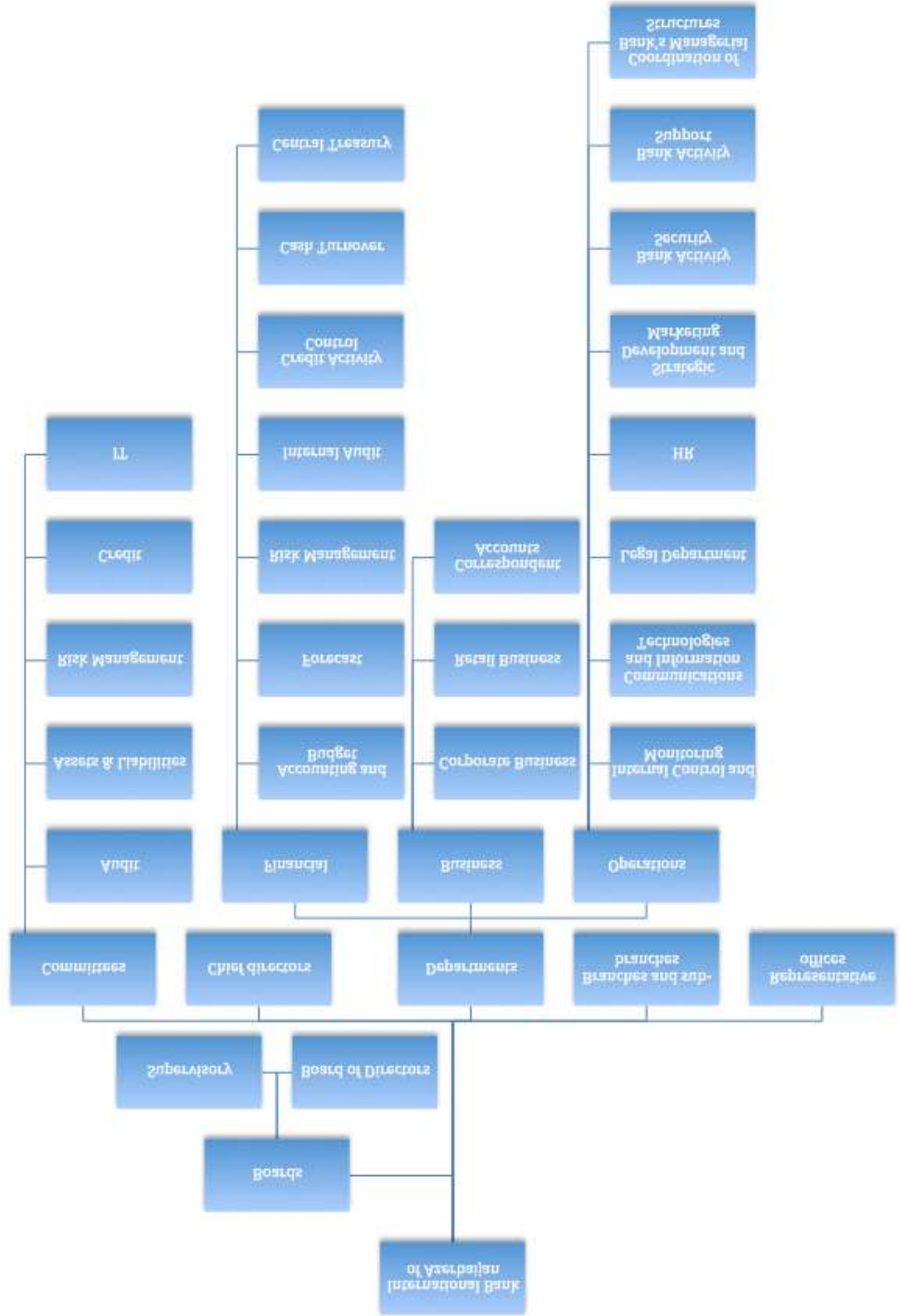
Your boundaries will be wrong!

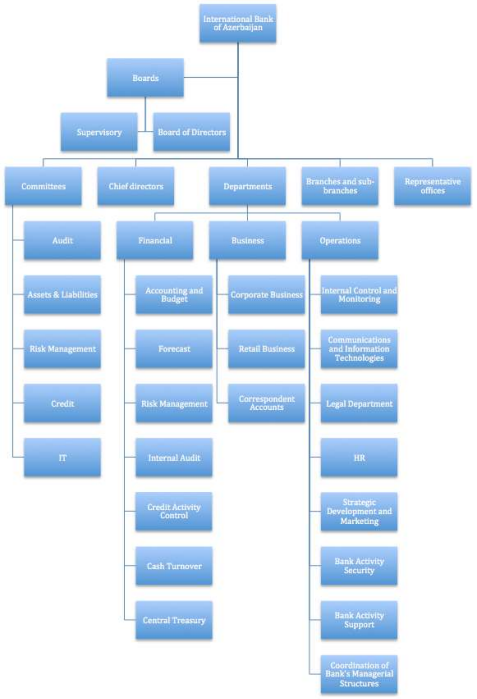
(Plan accordingly)

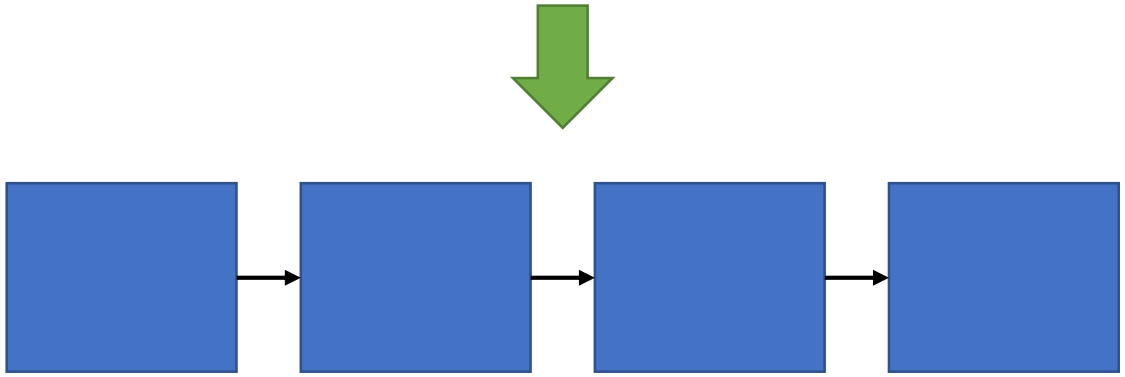


Org Boundary? (Conway's Law)

Reverse Conway?

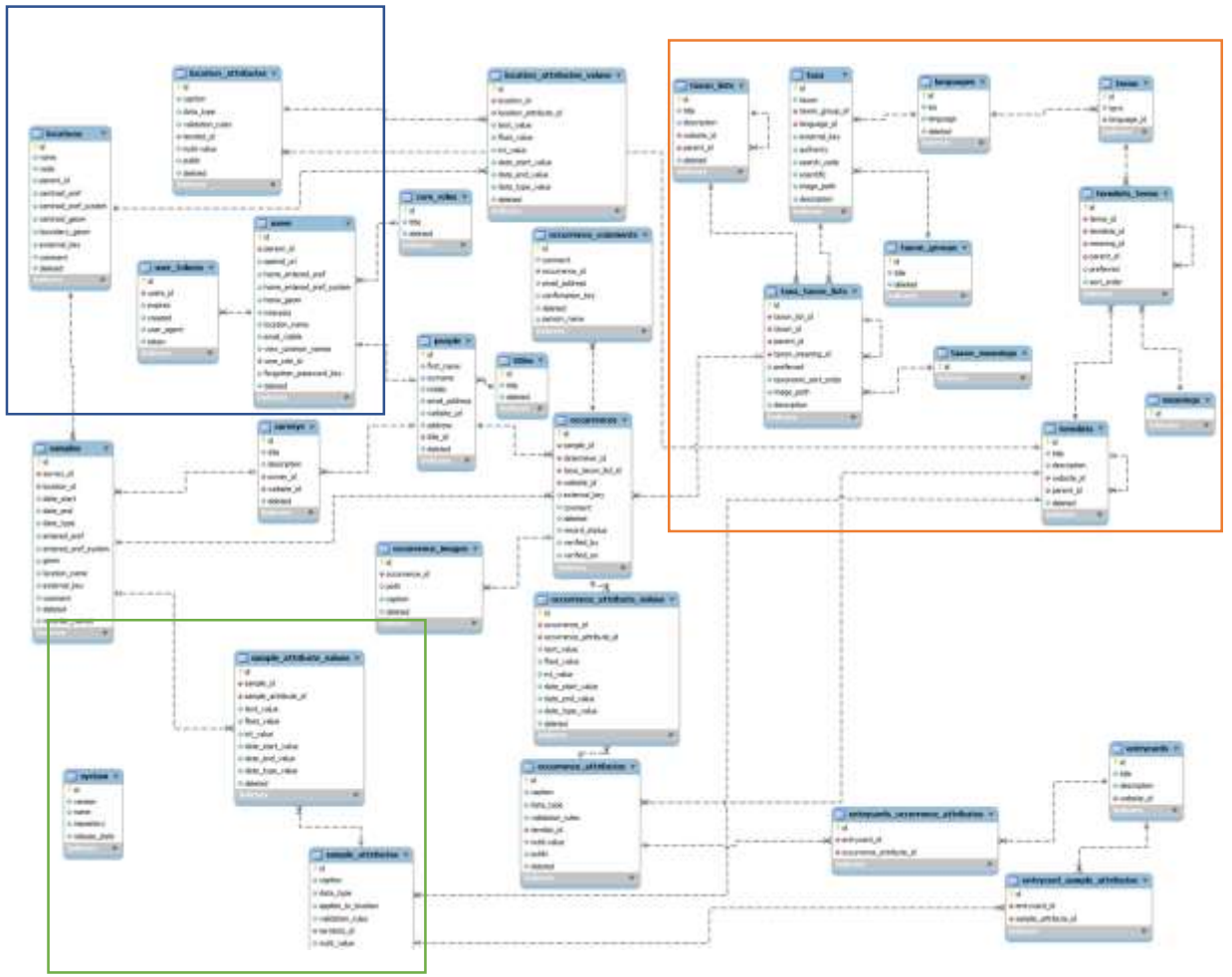


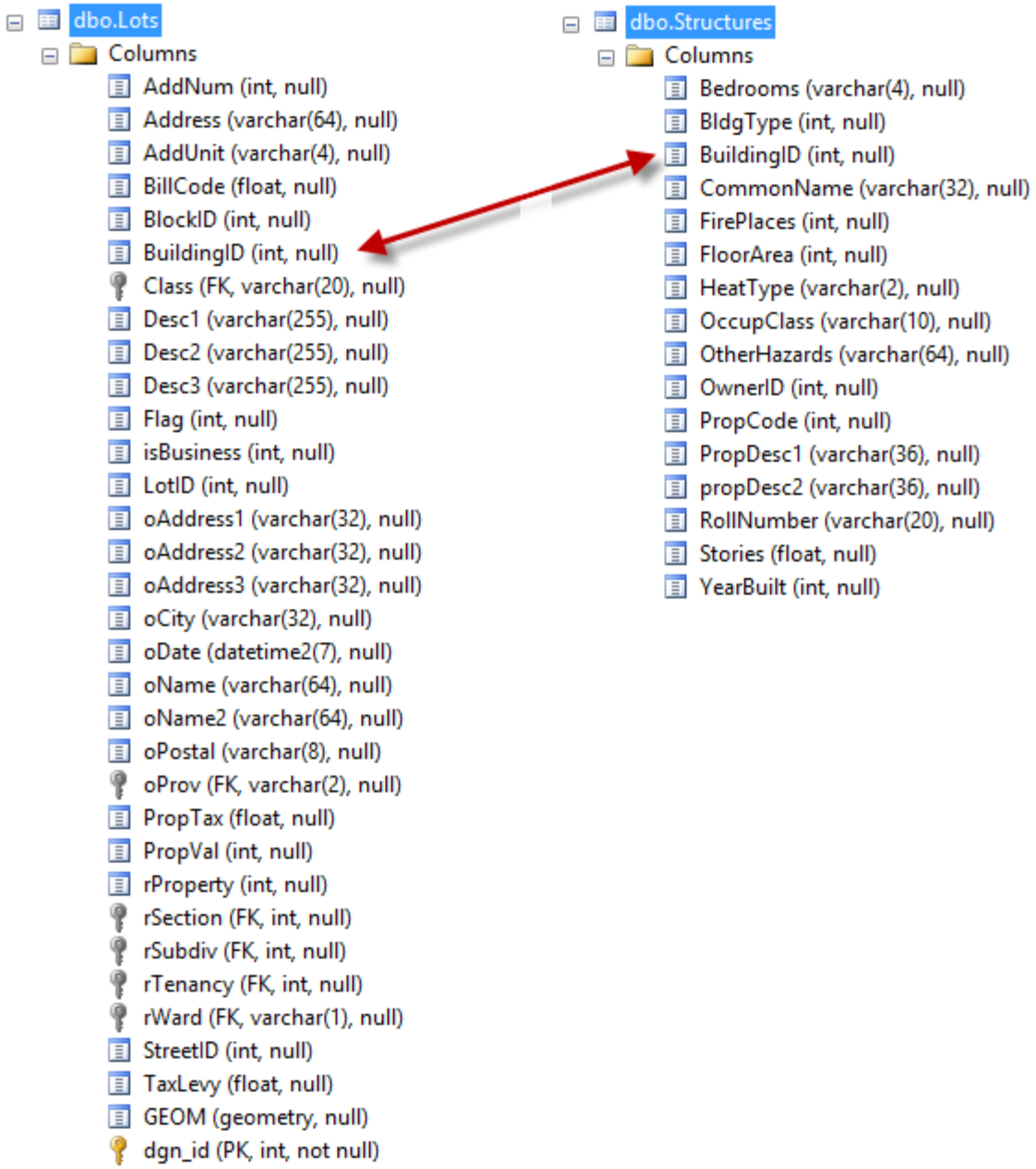




Value Chain Alignment

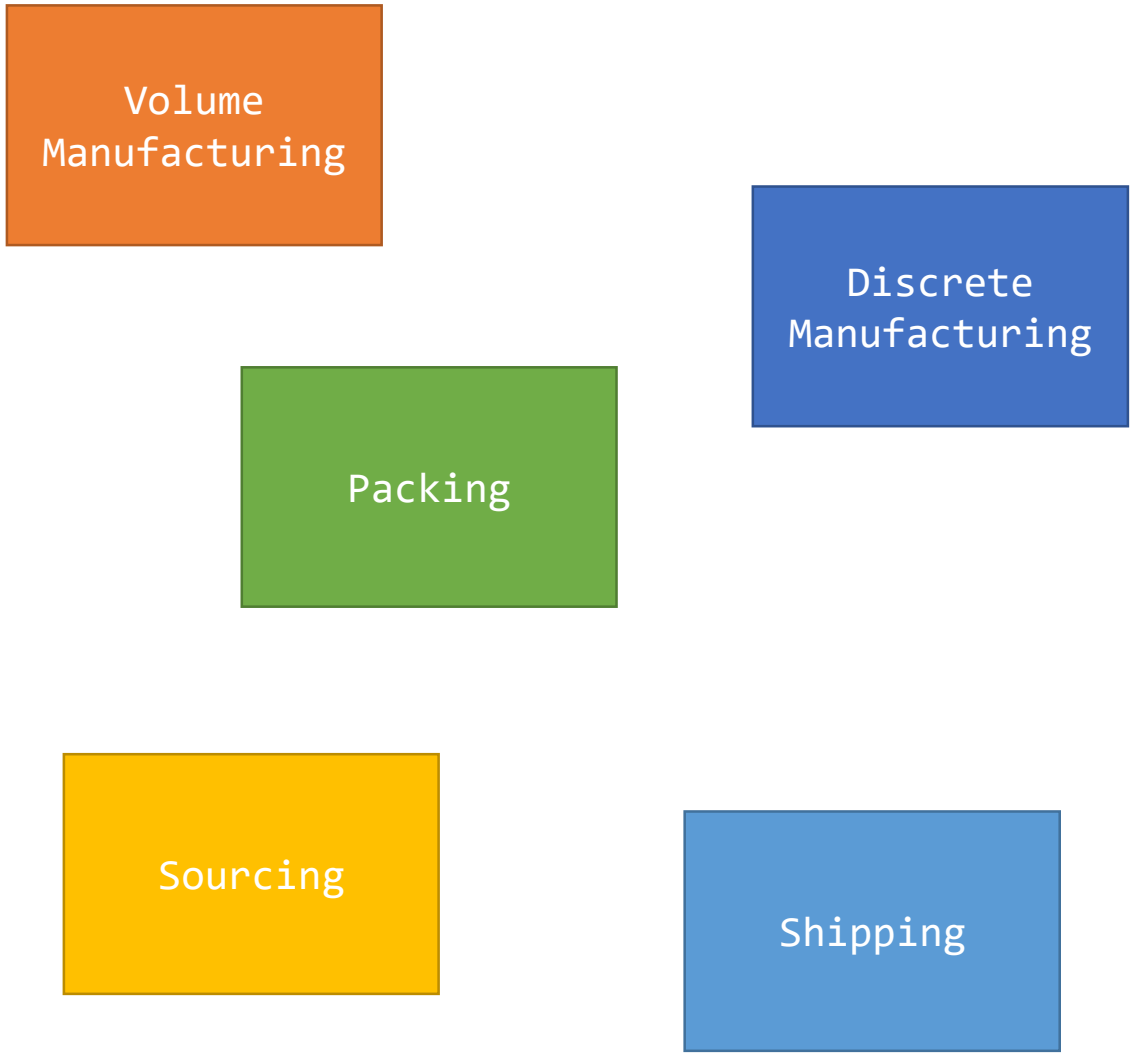
Data Boundaries?





Data Boundary Issues

Business or Functional Areas?



Batch Manufacturing

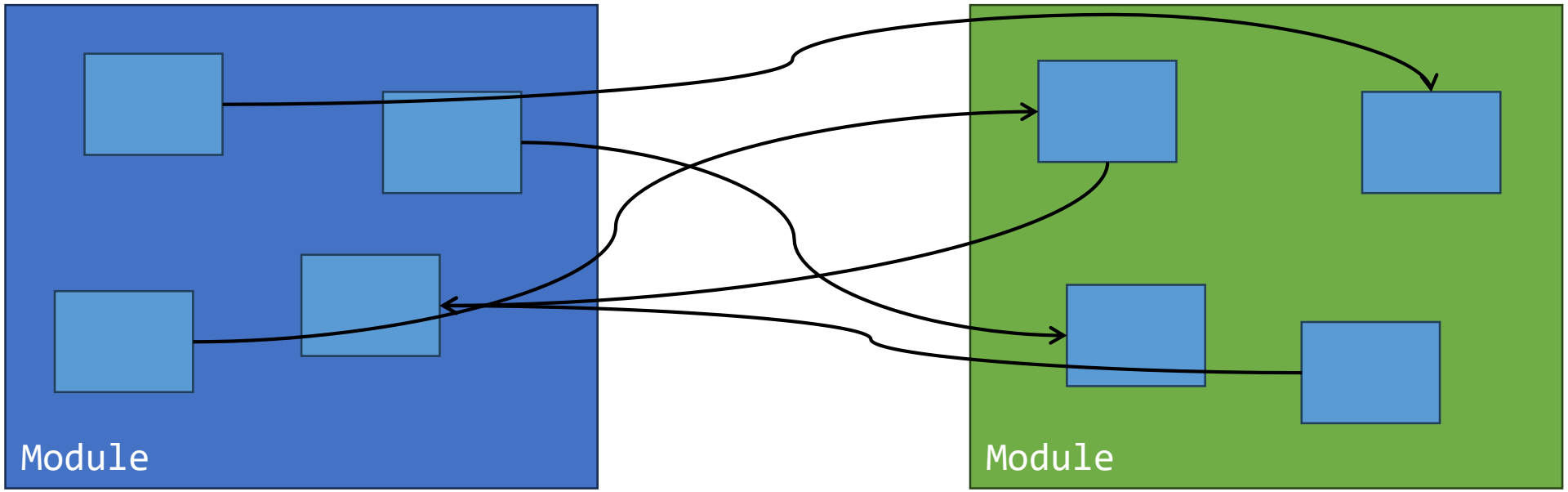
Discrete Manufacturing

Procurement

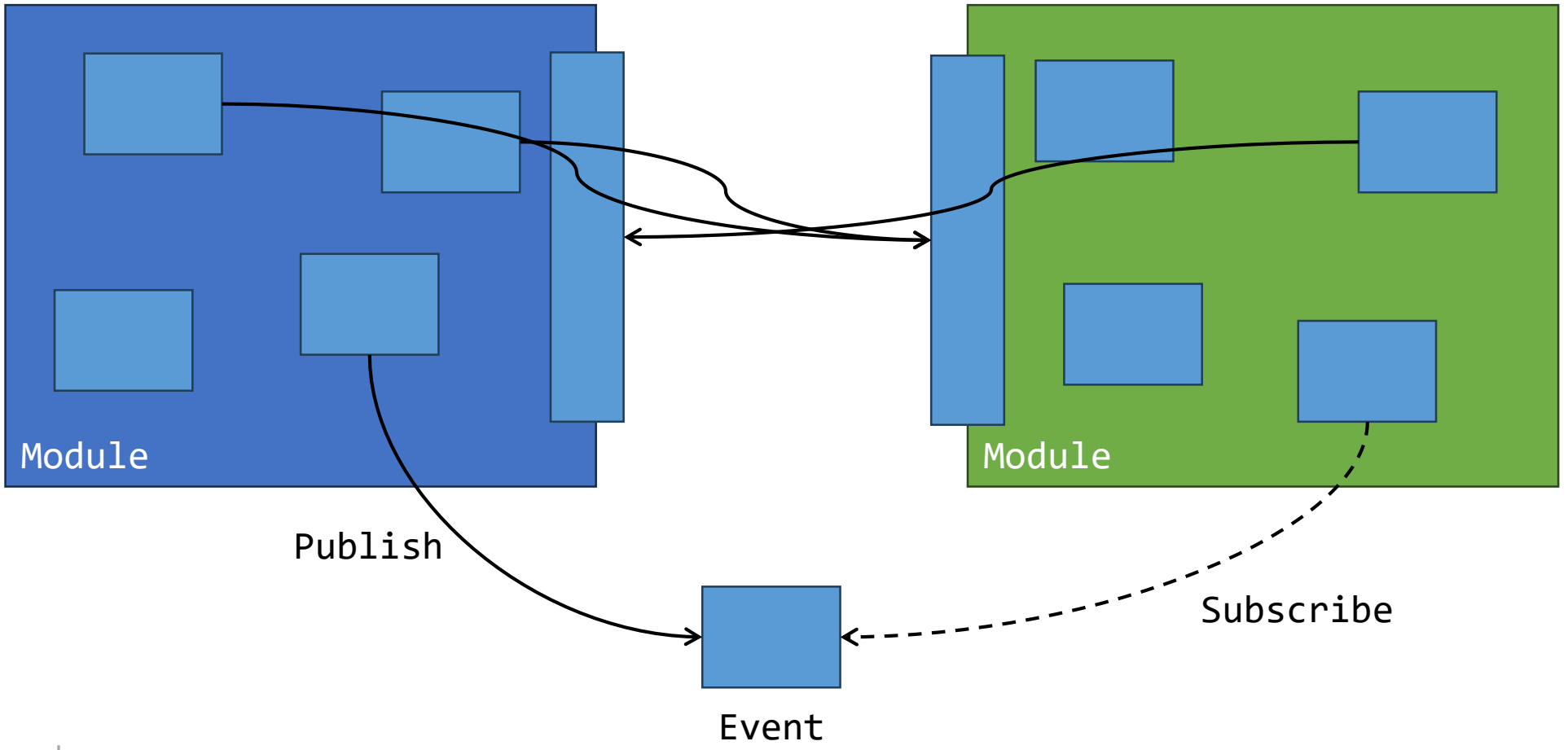
Onboarding

Value Streams?

Modules without contracts or encapsulation

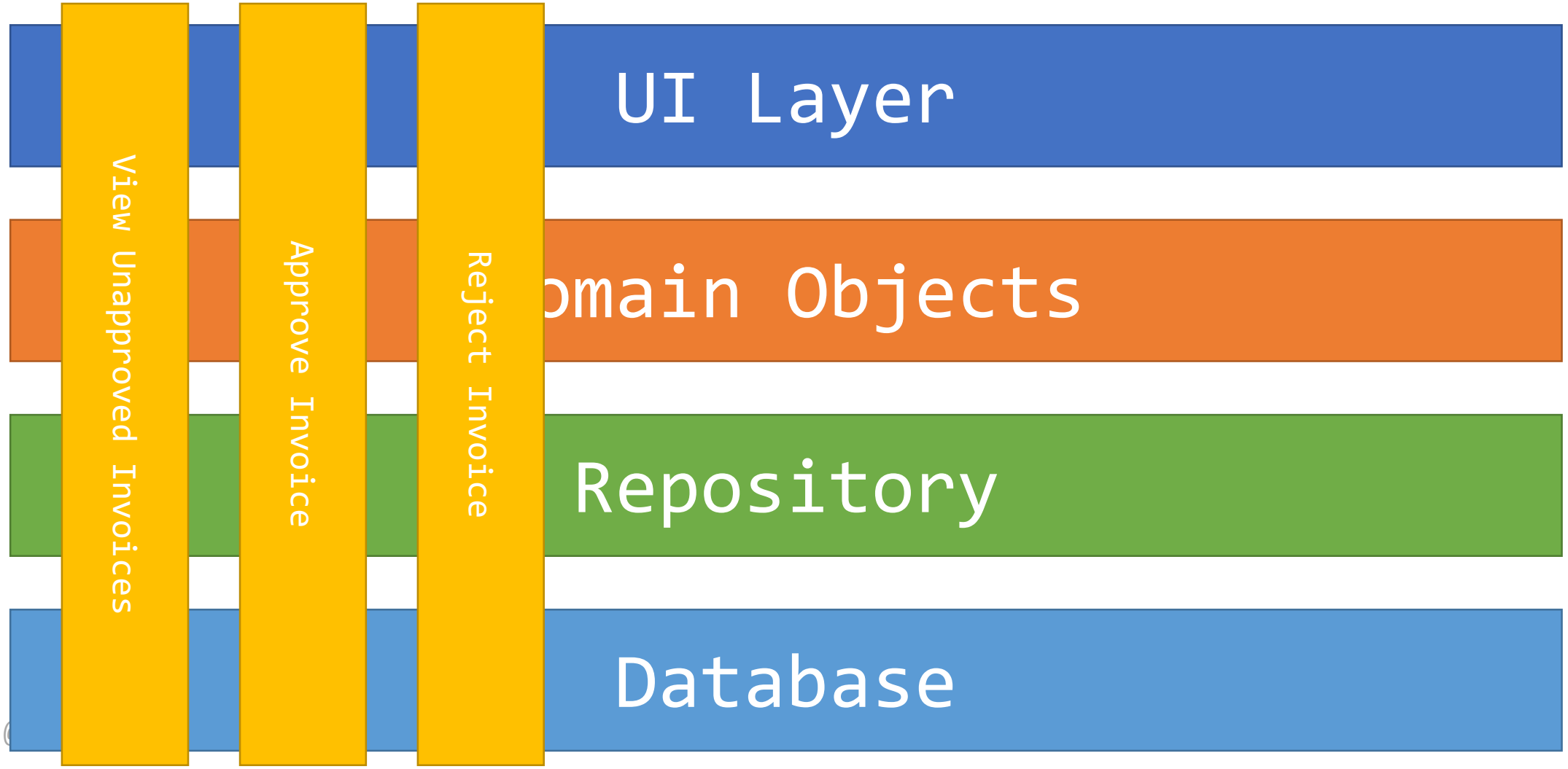


Encapsulating modules with contracts



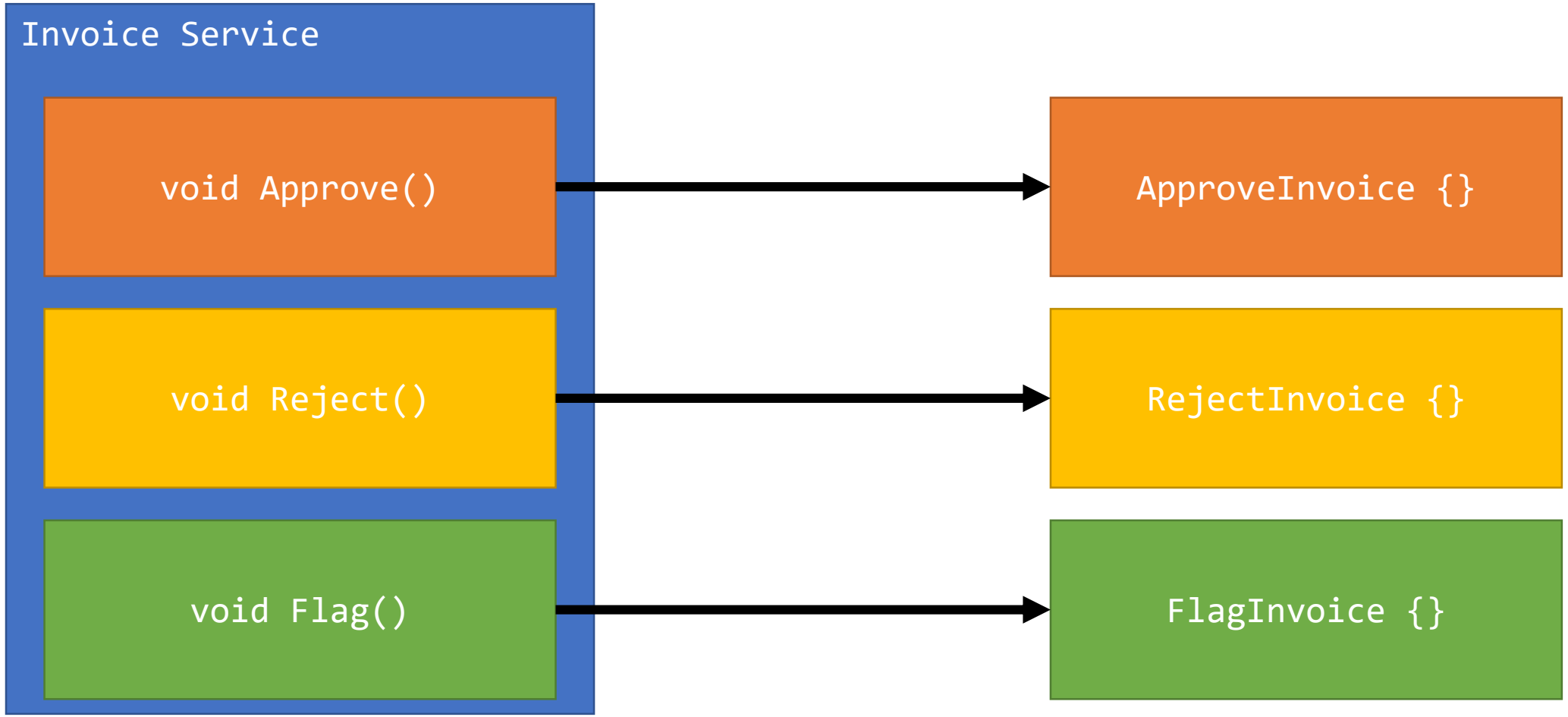
Refactoring to slices

Vertical Slice Architecture

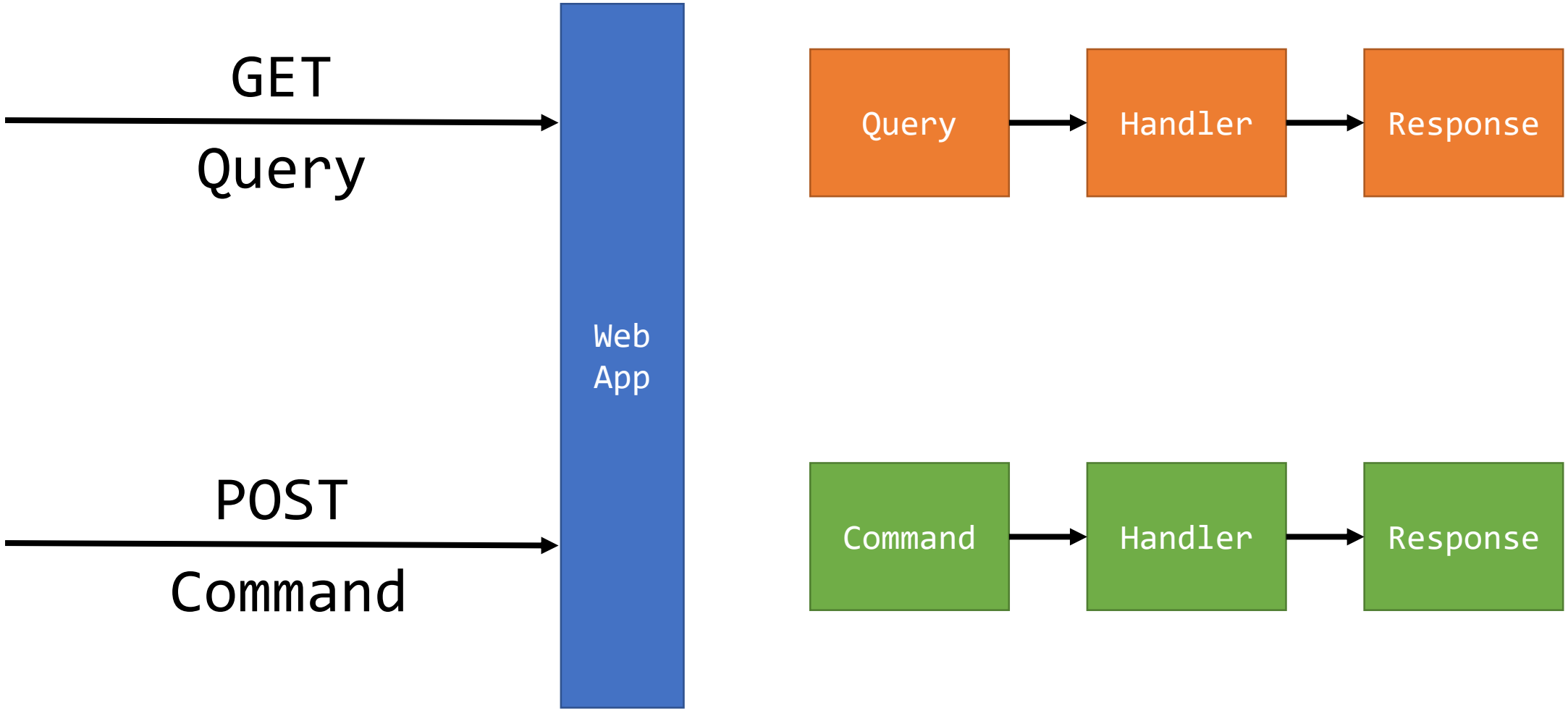


Move code to single place

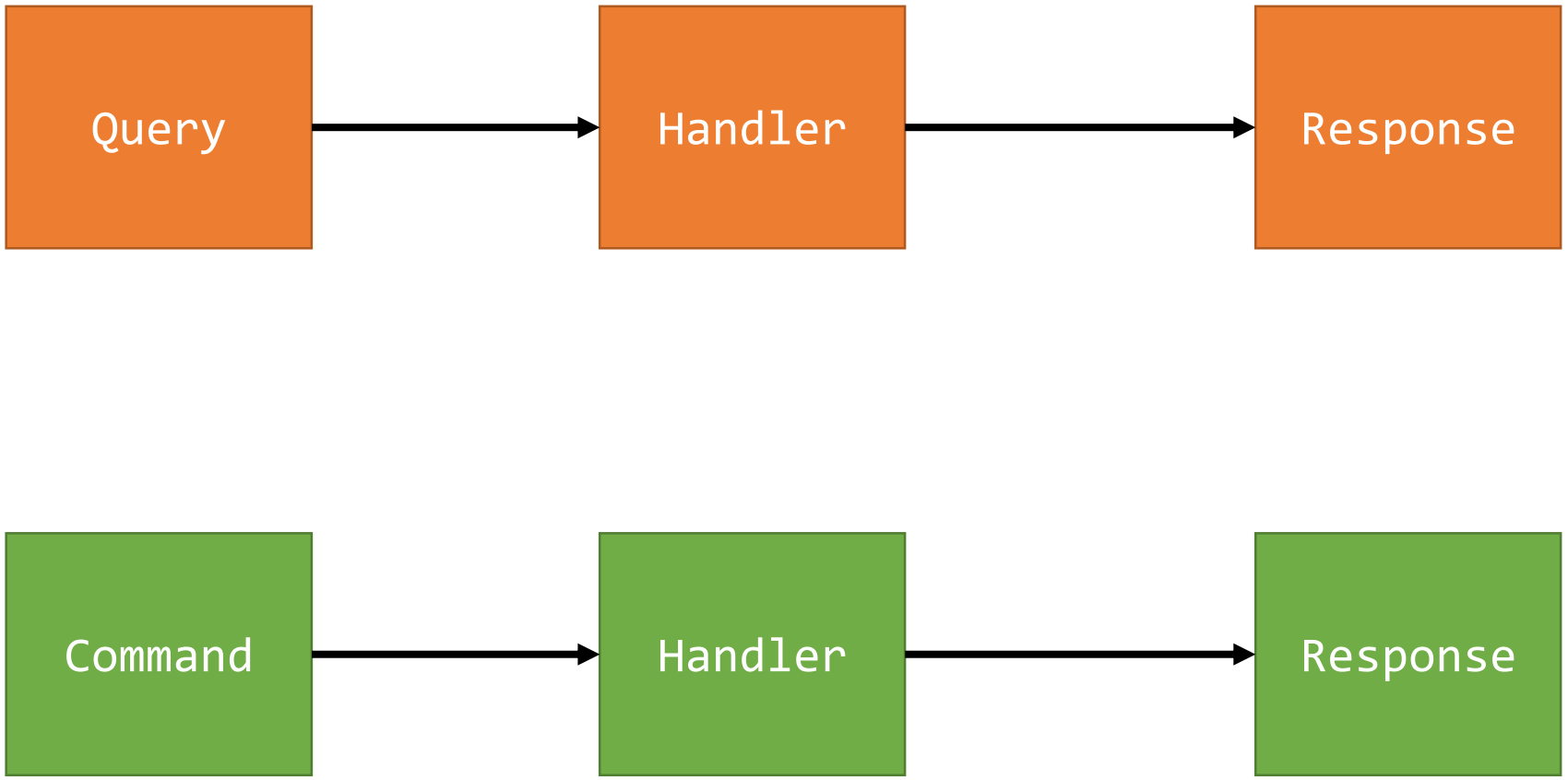
Move service methods to classes



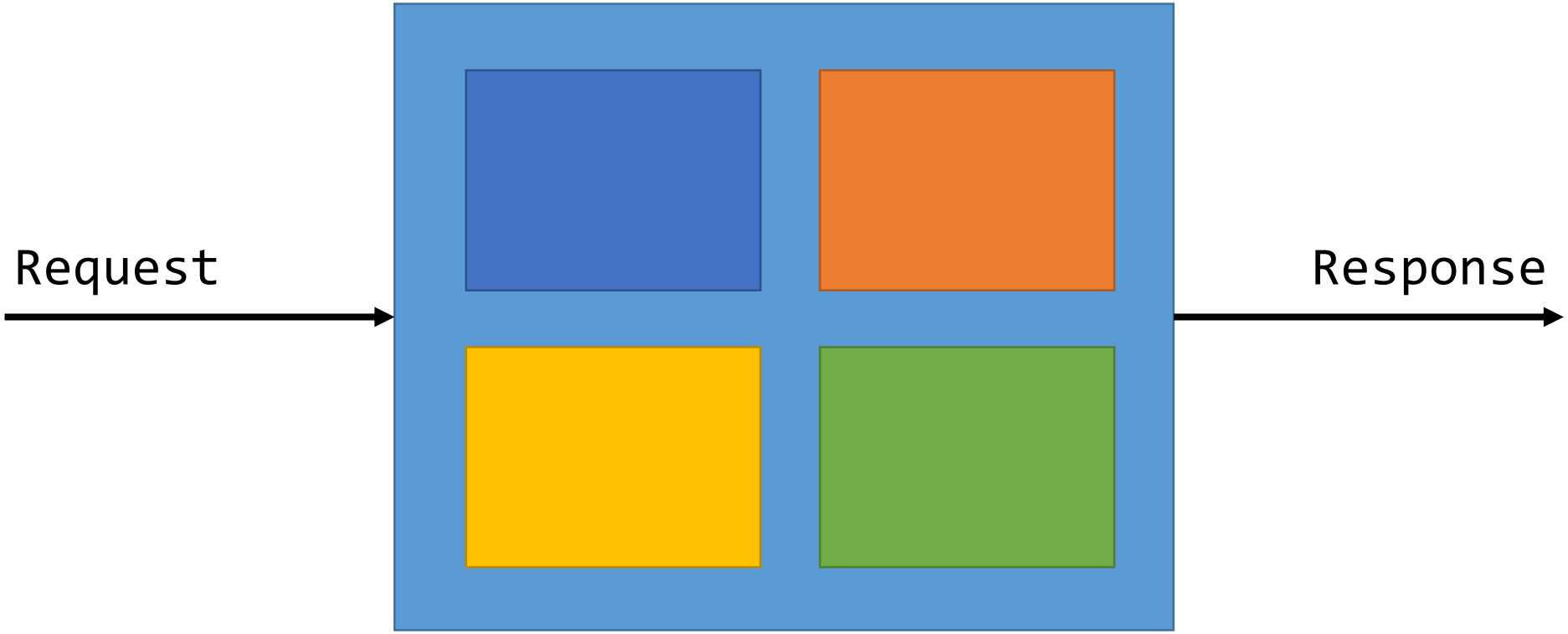
Commands and Queries



One model in, one model out



Complete encapsulation



Modeling Queries

```
public ViewResult Index(string sortOrder, string currentFilter, string searchString, int? page)
{
    ViewBag.CurrentSort = sortOrder;
    ViewBag.NameSortParm = String.IsNullOrEmpty(sortOrder) ? "name_desc" : "";
    ViewBag.DateSortParm = sortOrder == "Date" ? "date_desc" : "Date";

    if (searchString != null)
    {
        page = 1;
    }
    else
    {
        searchString = currentFilter;
    }

    ViewBag.CurrentFilter = searchString;

    var students = from s in db.Students
                   select s;
    if (!String.IsNullOrEmpty(searchString))
    {
        students = students.Where(s => s.LastName.ToUpper().Contains(searchString.ToUpper())
                               || s.FirstMidName.ToUpper().Contains(searchString.ToUpper()));
    }
    switch (sortOrder)
```

Modeling outputs

```
ViewBag.CurrentFilter = searchString;

var students = from s in db.Students
               select s;
if (!String.IsNullOrEmpty(searchString))
{
    students = students.Where(s => s.LastName.ToUpper().Contains(searchString.ToUpper())
                           || s.FirstMidName.ToUpper().Contains(searchString.ToUpper()));
}
switch (sortOrder)
{
    case "name_desc":
        students = students.OrderByDescending(s => s.LastName);
        break;
    case "Date":
        students = students.OrderBy(s => s.EnrollmentDate);
        break;
    case "date_desc":
        students = students.OrderByDescending(s => s.EnrollmentDate);
        break;
    default:  // Name ascending
        students = students.OrderBy(s => s.LastName);
        break;
}

int pageSize = 3;
int pageNumber = (page ?? 1);
return View(students.ToPagedList(pageNumber, pageSize));
```

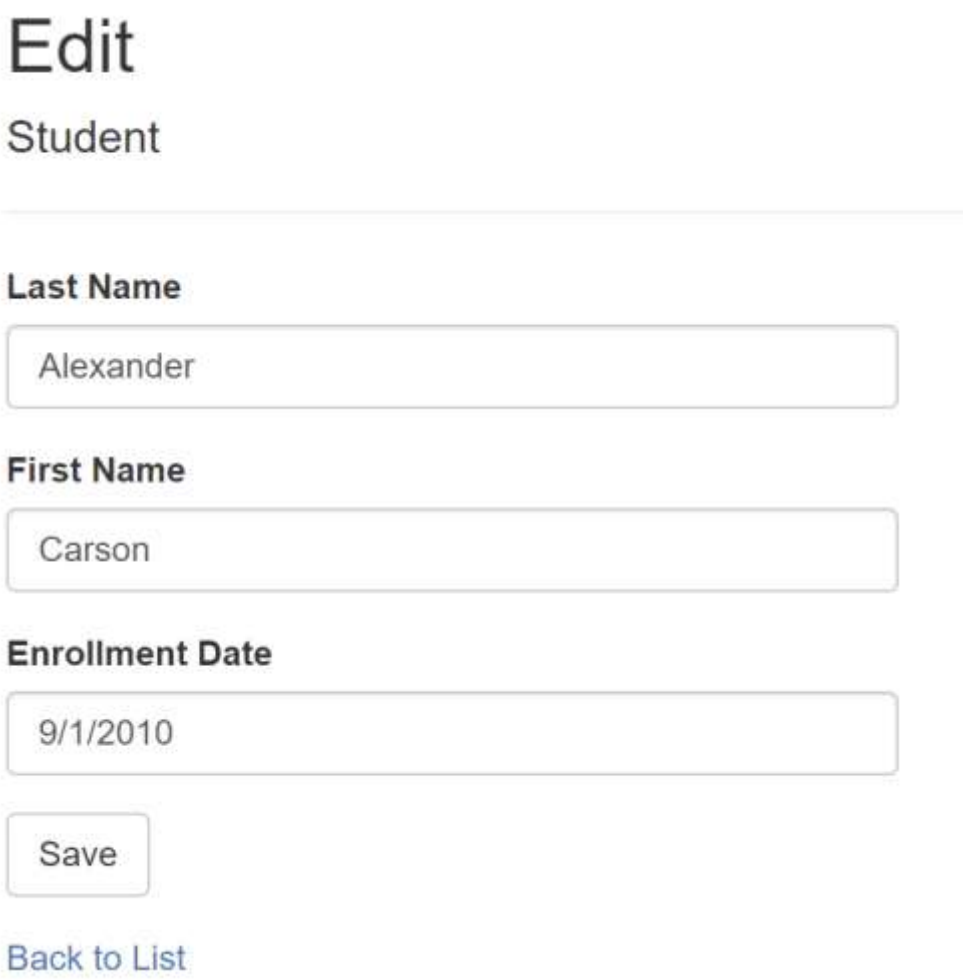
Modeling outputs

```
@foreach (var item in Model)
{
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.LastName)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.FirstMidName)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.EnrollmentDate)
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id = item.ID }) |
            @Html.ActionLink("Details", "Details", new { id = item.ID })
            @Html.ActionLink("Delete", "Delete", new { id = item.ID })
        </td>
    </tr>
```

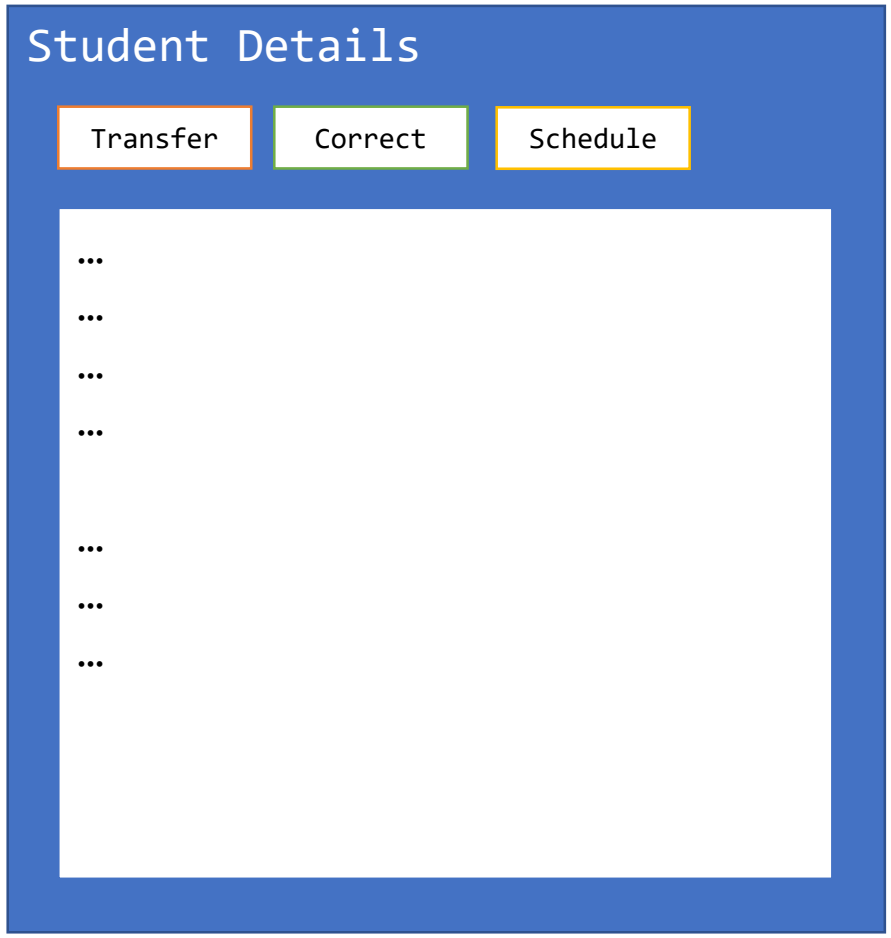
Modeling Commands

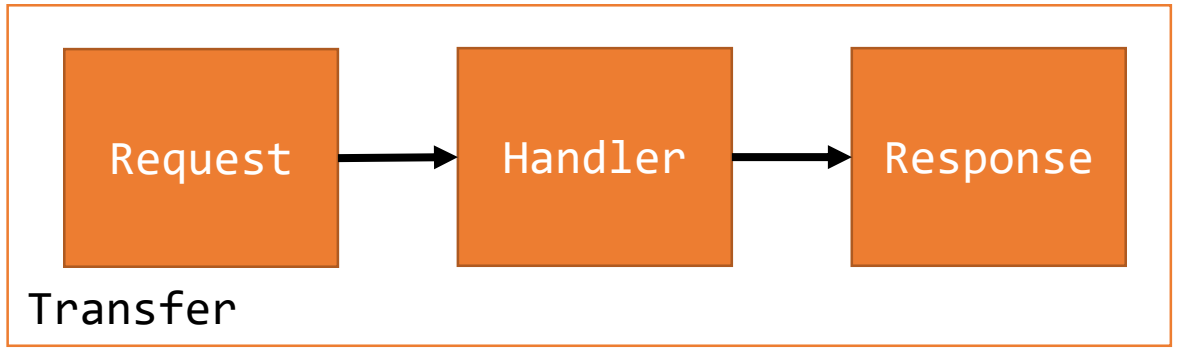
```
public class Edit
{
    public class Command : IRequest
    {
        public int Id { get; set; }
        public string LastName { get; set; }
        public string FirstMidName { get; set; }
        public DateTime? EnrollmentDate { get; set; }
    }
```

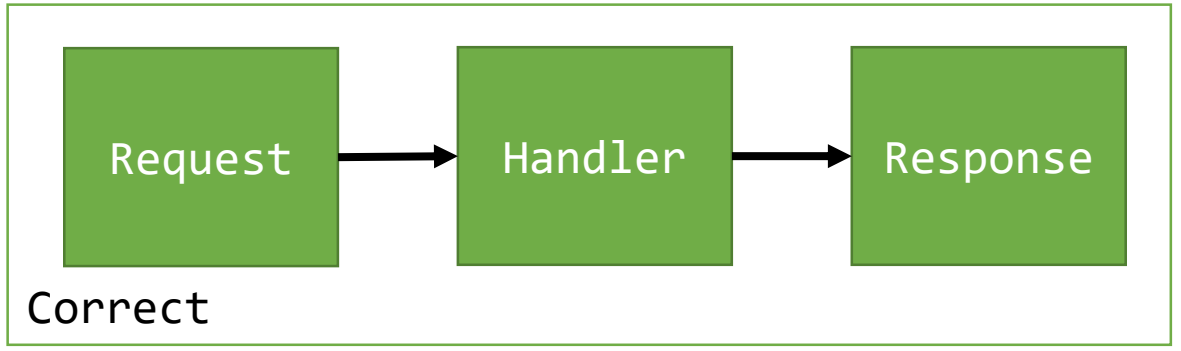
Commands as Forms

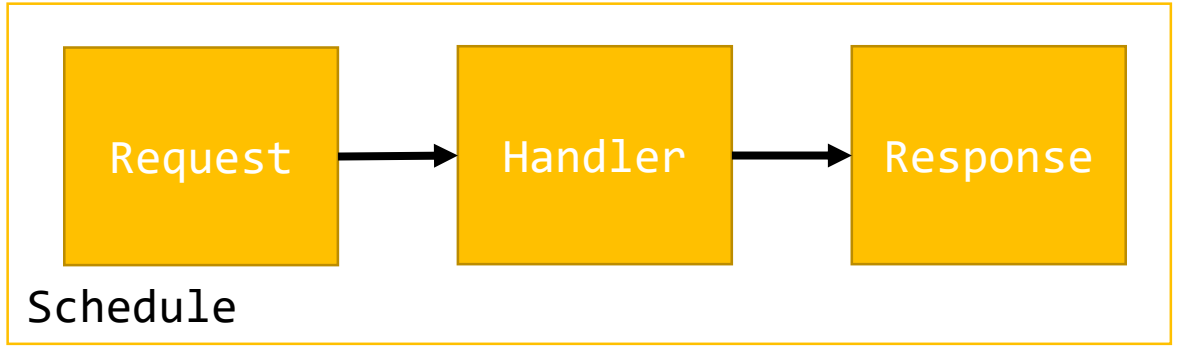


Task-Based UIs

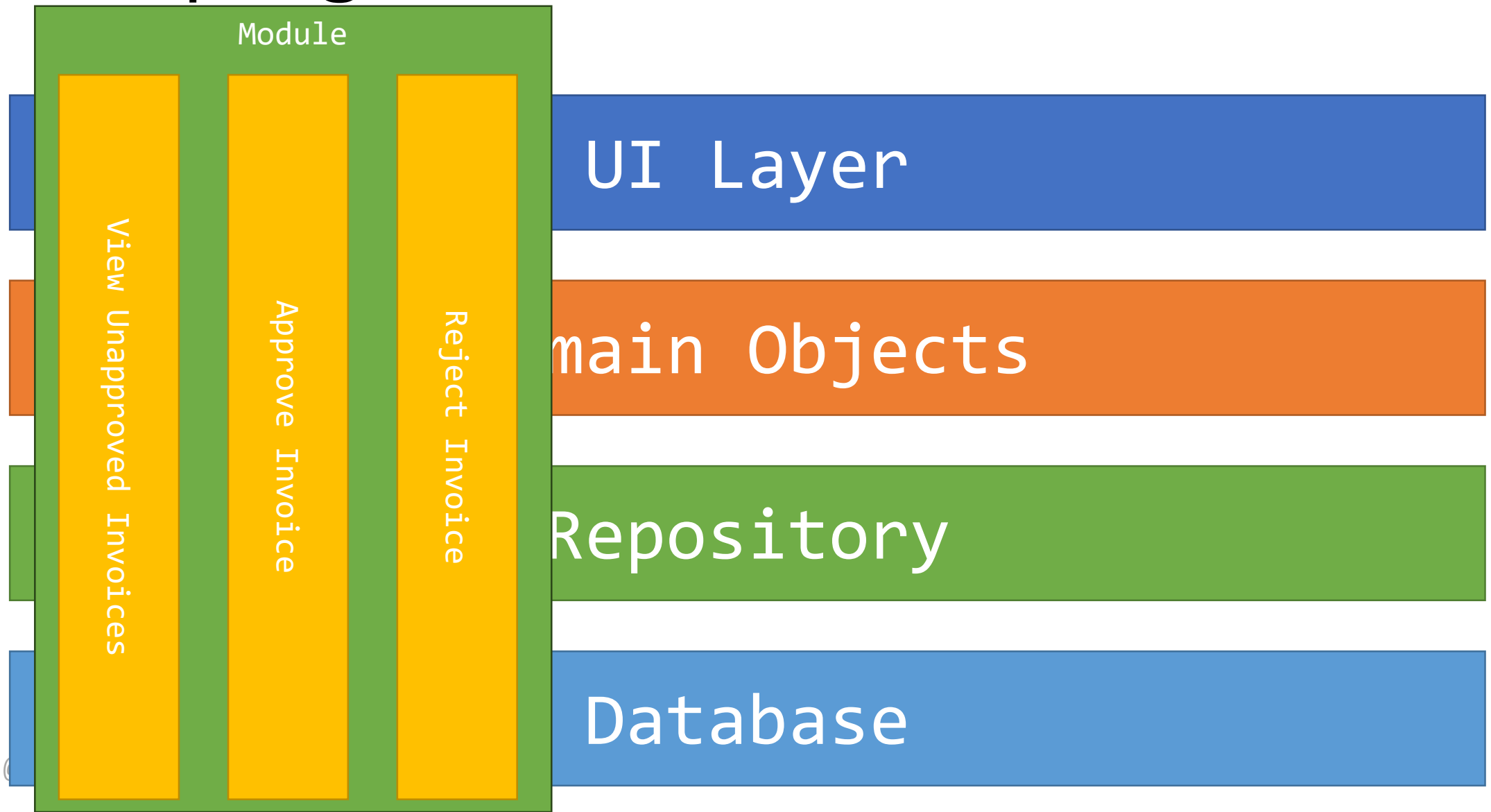






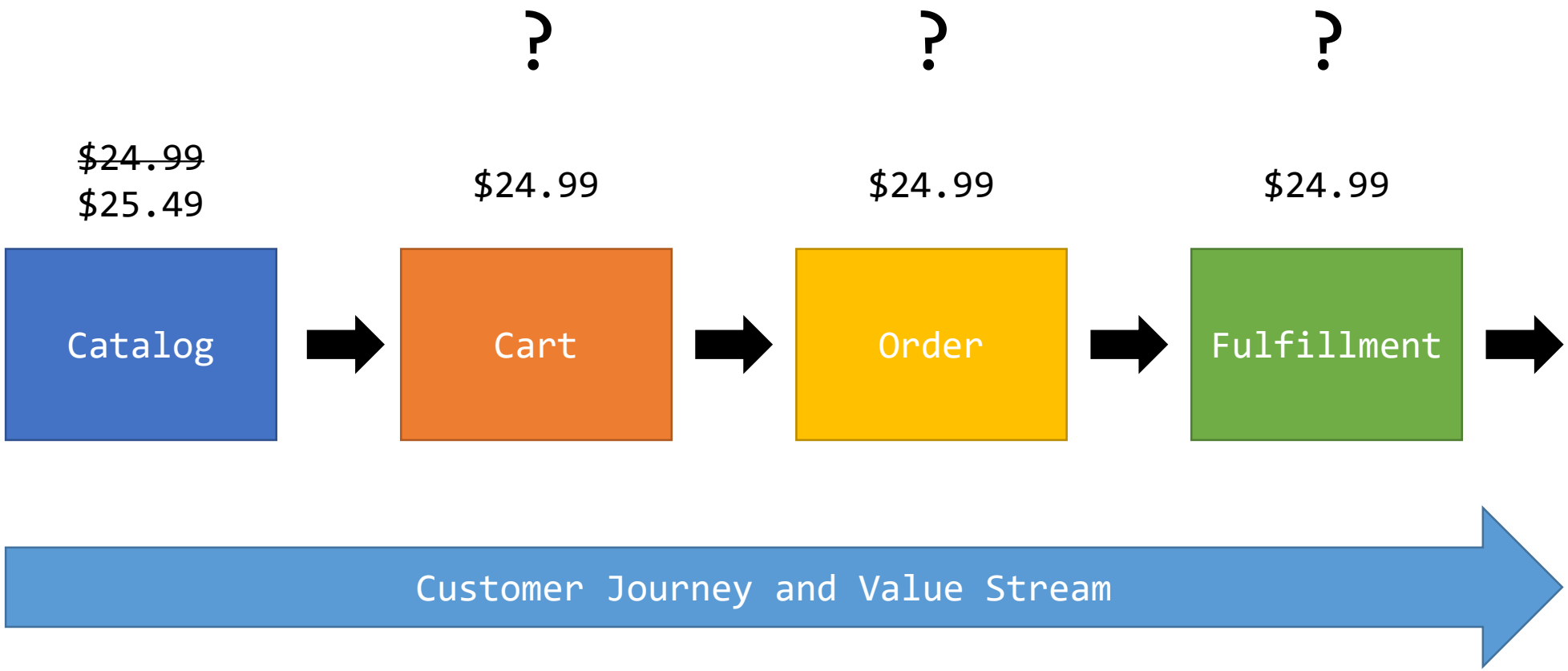


Grouping Slices into Modules

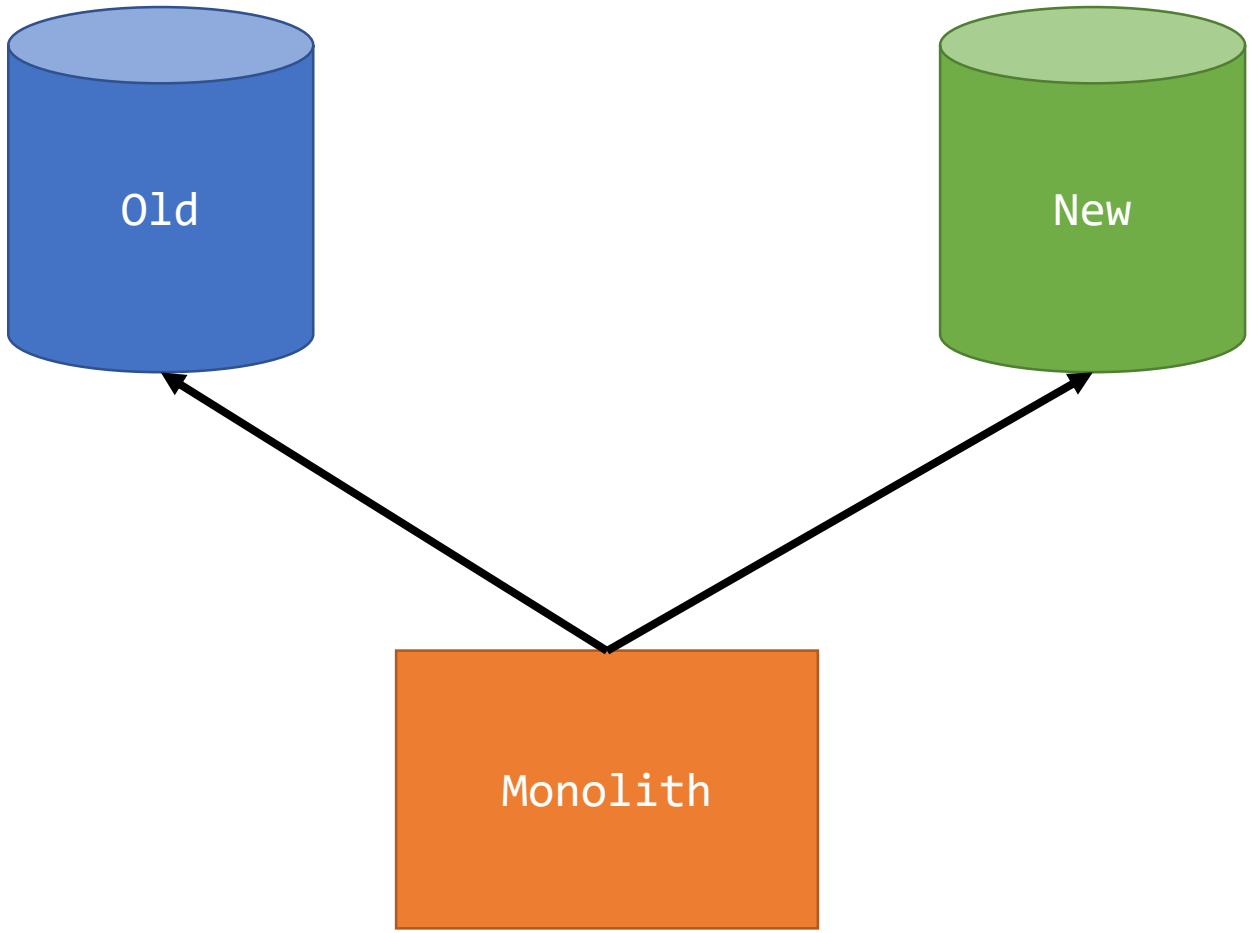


Data Boundaries

Single source of truth

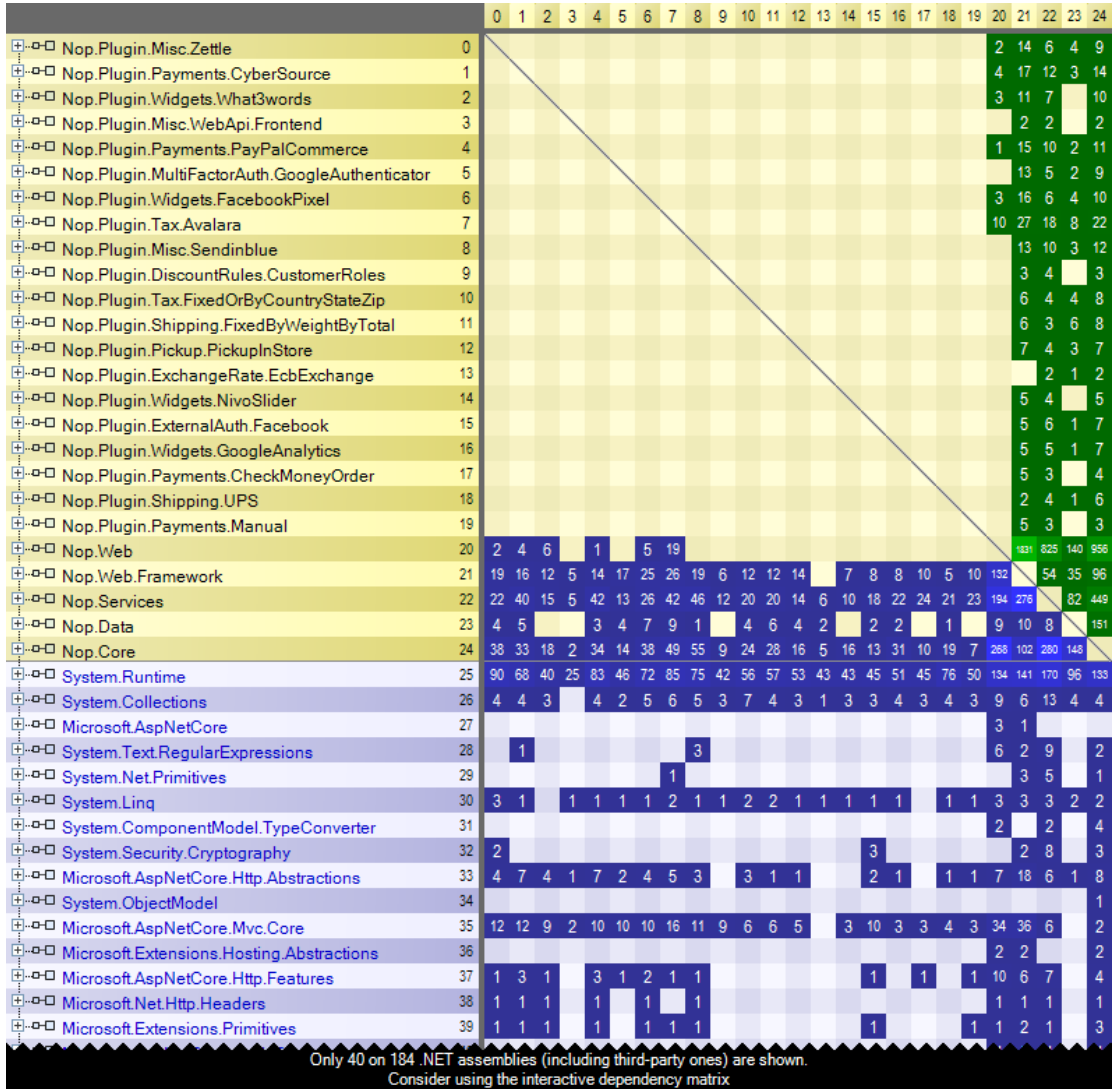

?
?
?
~~$24.99~~
$25.49
$24.99
$24.99
$24.99
Catalog
Cart
Order
Fulfillment
Customer Journey and Value Stream

Migrating Data Paths



Assessing Module Boundaries

Static Analysis



Extracting Modules into (Micro)Services

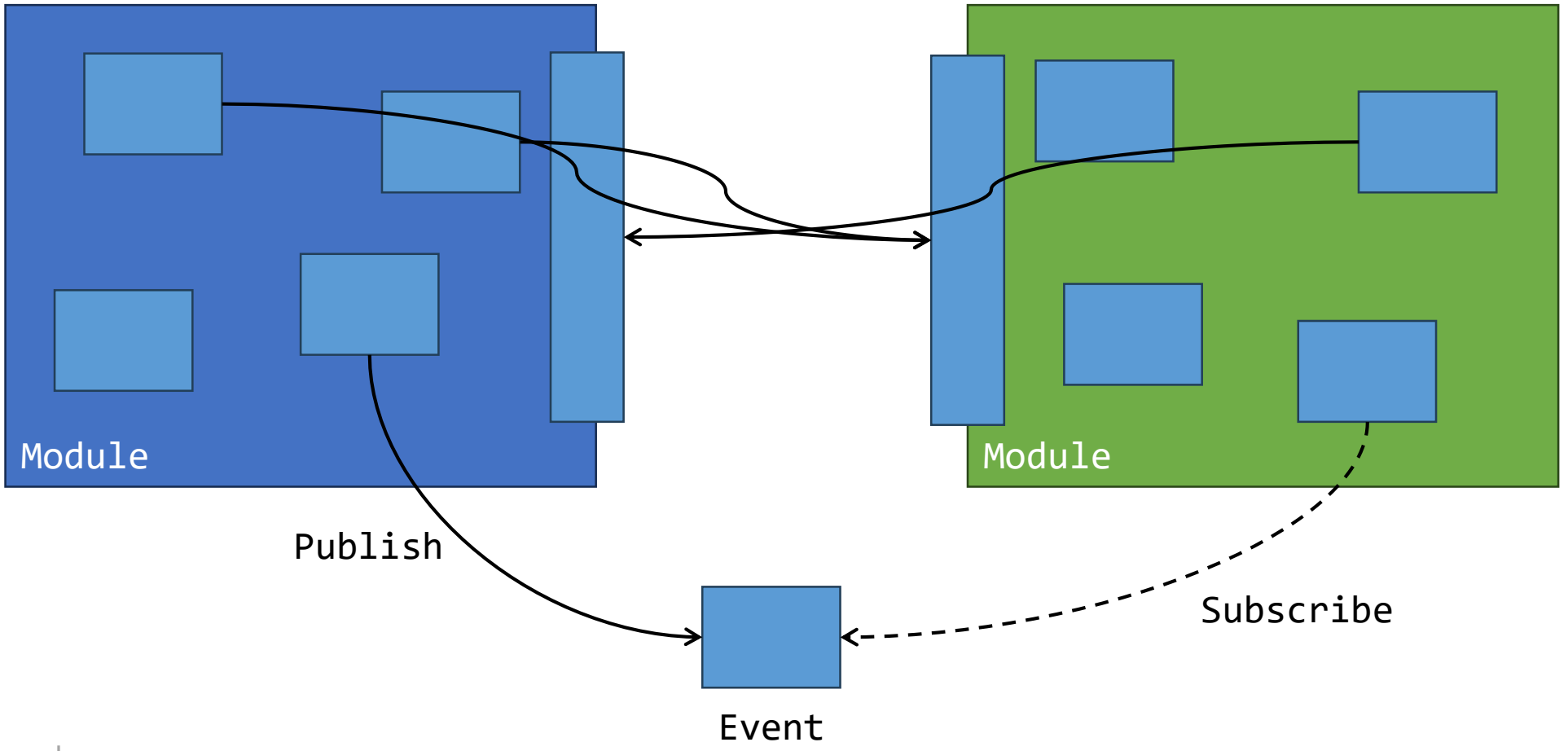
What is a (micro)service?

- Independently deployable
- Loosely coupled
- Autonomous

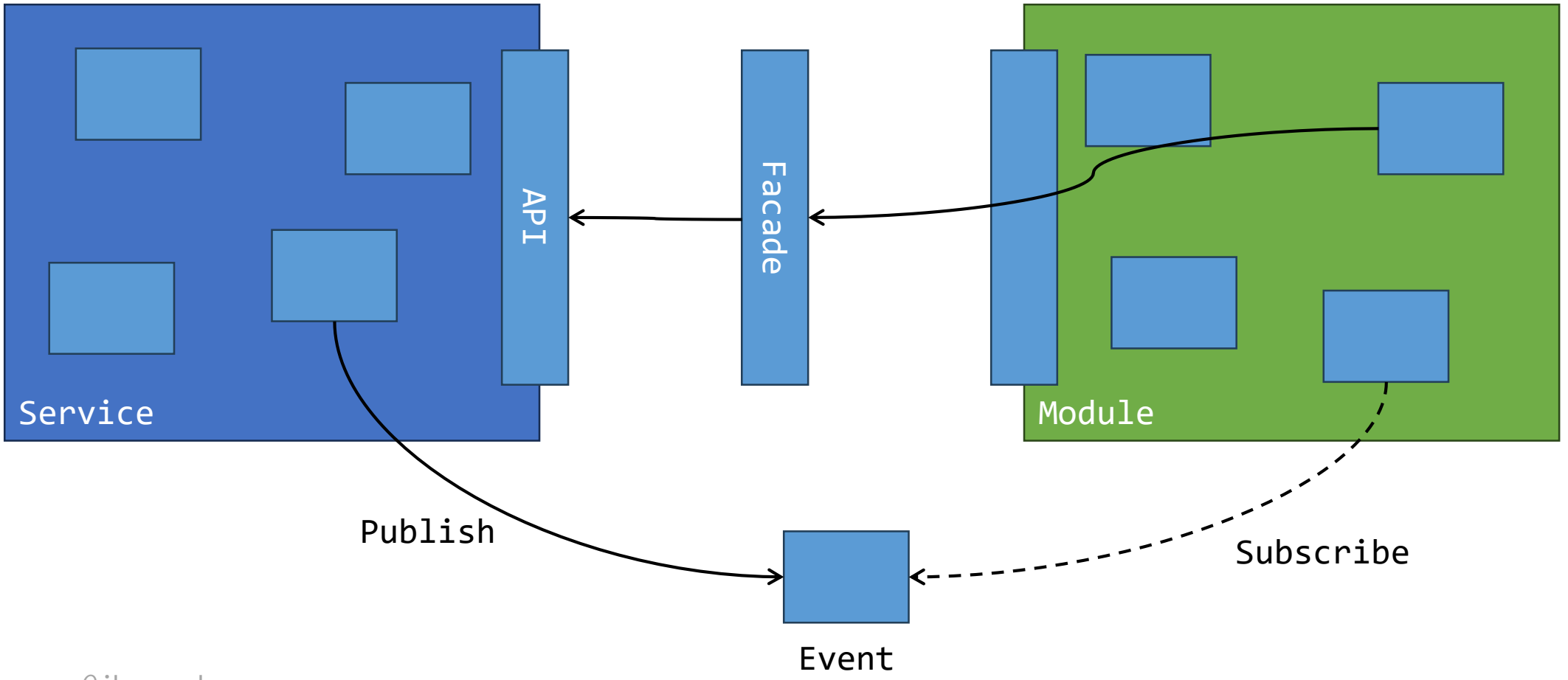
What is autonomy?

- Independent ownership, availability, and delivery
- Manage information and access
- Handle failures
- Maintain contracts

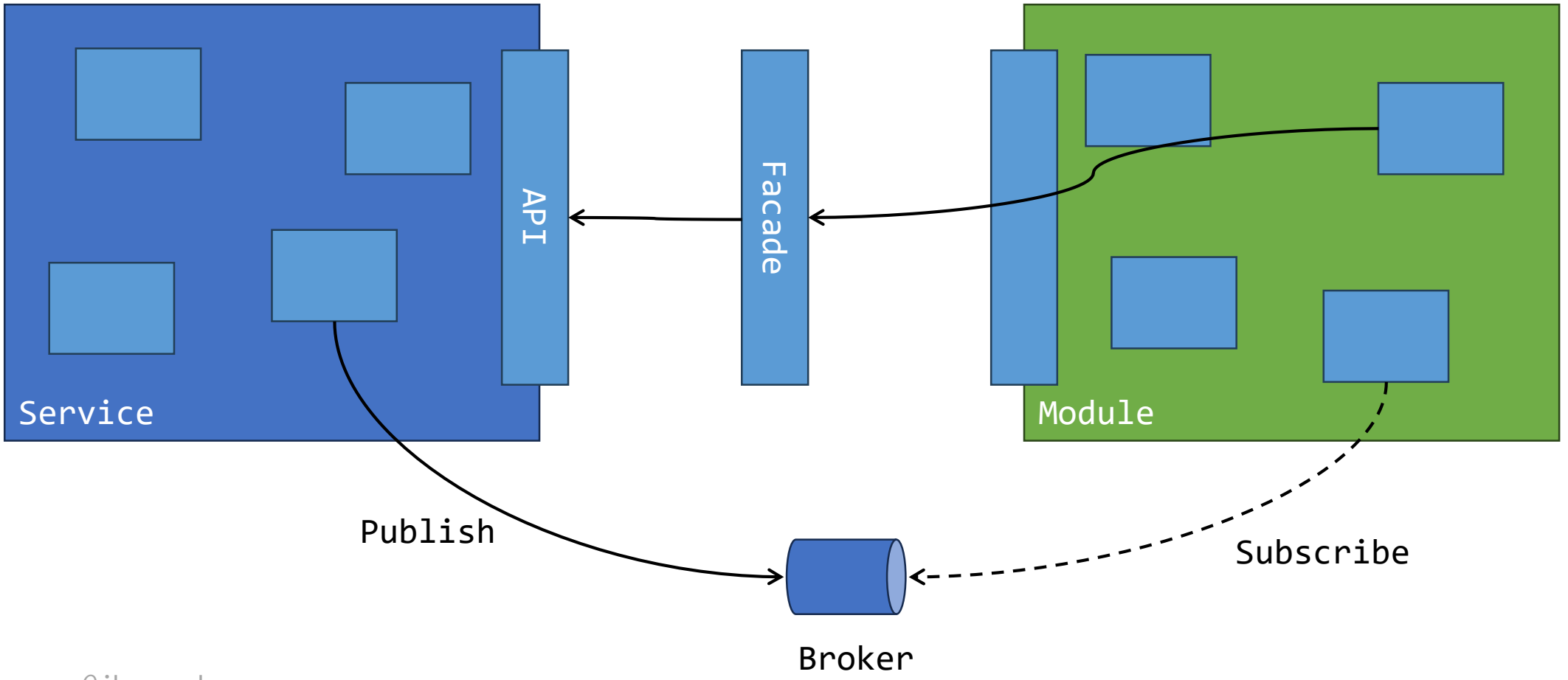
Modules to services



Facades to encapsulate



Sync to async messaging



Key Takeaways

- No one wants a big ball of mud
- Boundaries are critical to facilitate change
- Slices make it easier to move towards modules
- Shaping boundaries is easier in a monolith

Modularizing the Monolith

Jimmy Bogard
@jbogard
github.com/jbogard
jimmybogard.com