الجامعة السورية الخاصة
SYRIAN PRIVATE UNIVERSITY

مقرر بنيان البرمجيات

كلية الهندسة المعلوماتية

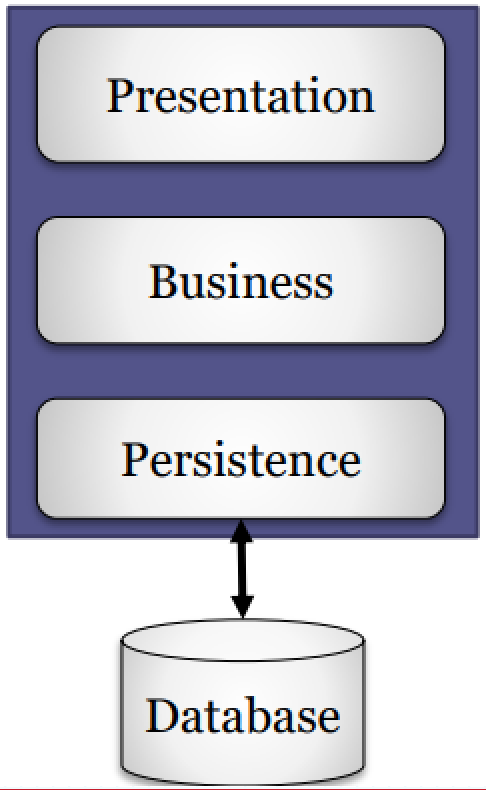
المحاضرة 8

Introduction to Clean Architecture
Hexagon (Ports and Adapters)

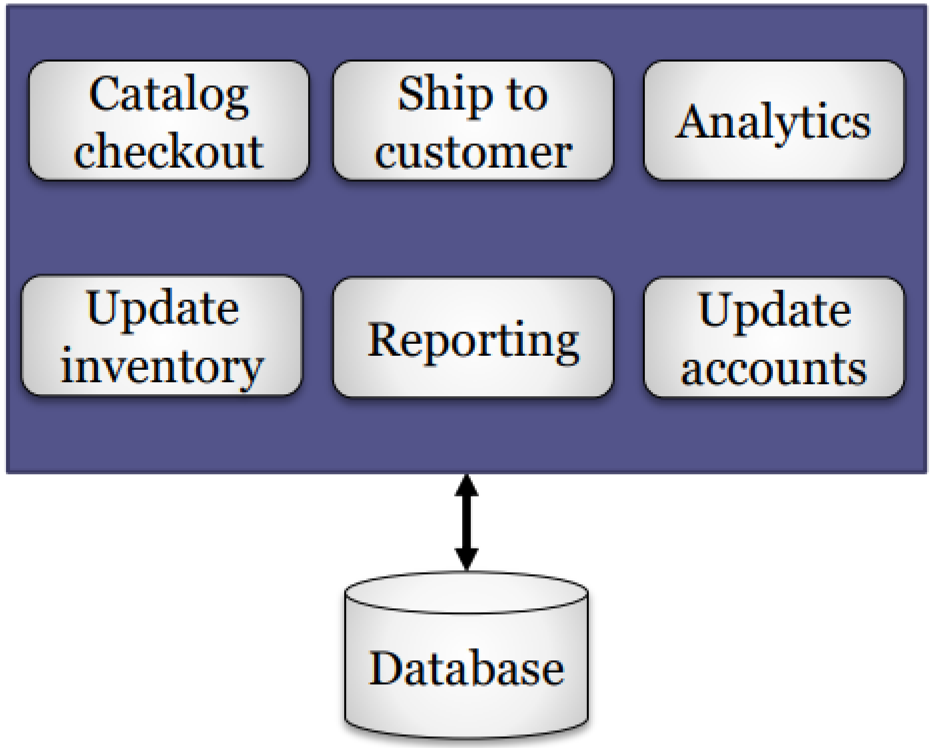
د. رياض سنبل

Technical vs domain partitioning

Technical partitioning
Organize system modules by technical capabilities



Domain partitioning
Organize modules by domain



Domain based

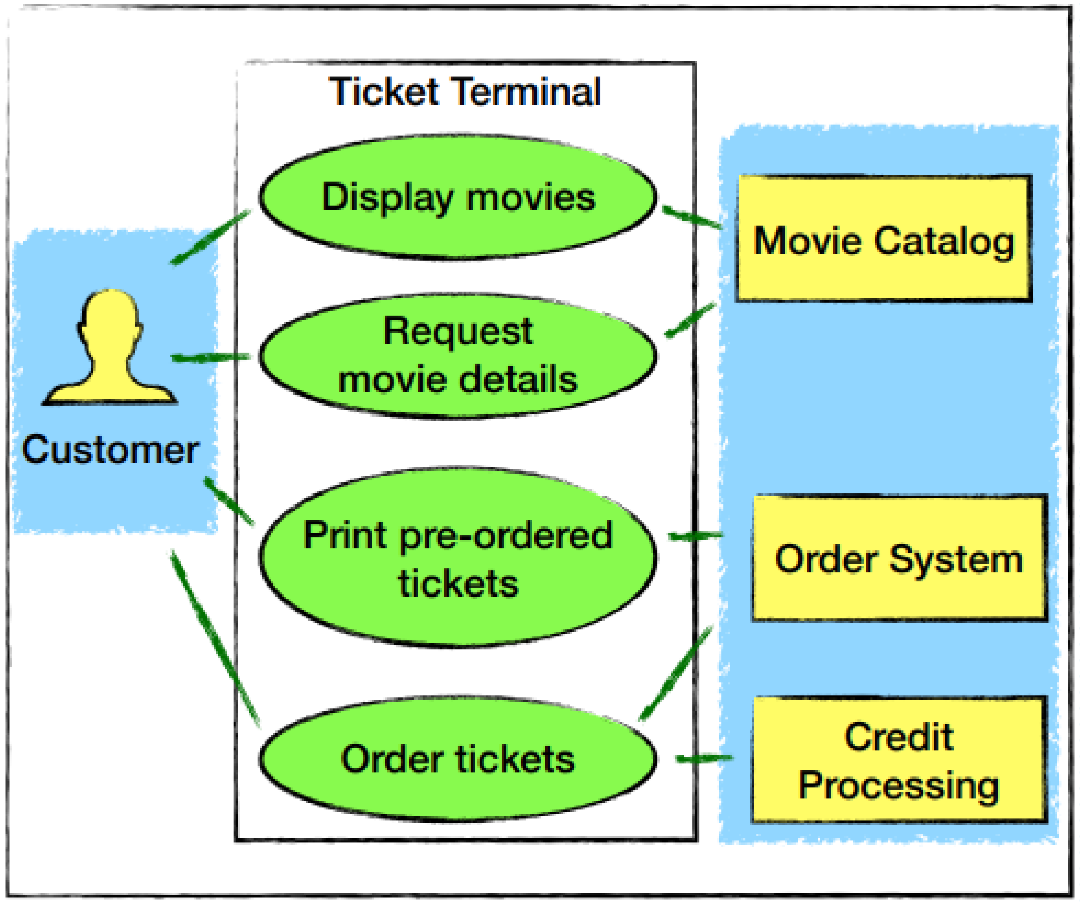
- Centered on the domain and the business logic
  - Goal: Anticipate and handle changes in domain
  - Collaboration between developers and domain experts
- Elements
  - Domain model: formed by: Context, Entities, Relationships
  - Application: Manipulates domain elements
- Variants
  - DDD - Domain driven design
  - Hexagonal style
  - Data centered
  - N-Layered Domain Driven Design
  - Naked Objects

Clean Architecture

- Use Cases as central organizing structure.
- Follows the Ports and Adapters pattern (Hexagonal Architecture).
  - Implementation is guided by tests.
  - Decoupled from technology details.
- Lots of Principles (SAP, SDP, SOLID..)
- Pluggable User Interface

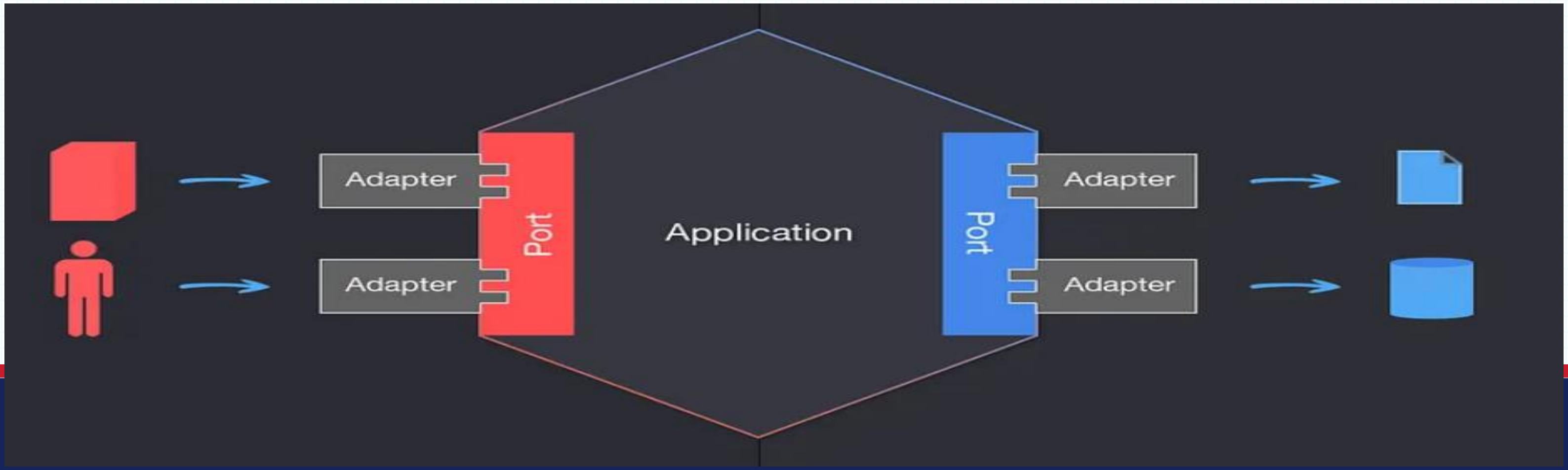
Use Cases

- Use Cases are **delivery independent**.
- Show the **intent** of a system.
- Use Cases are **algorithms** that interpret the input to generate the output data.
- **Primary** and **secondary** actors



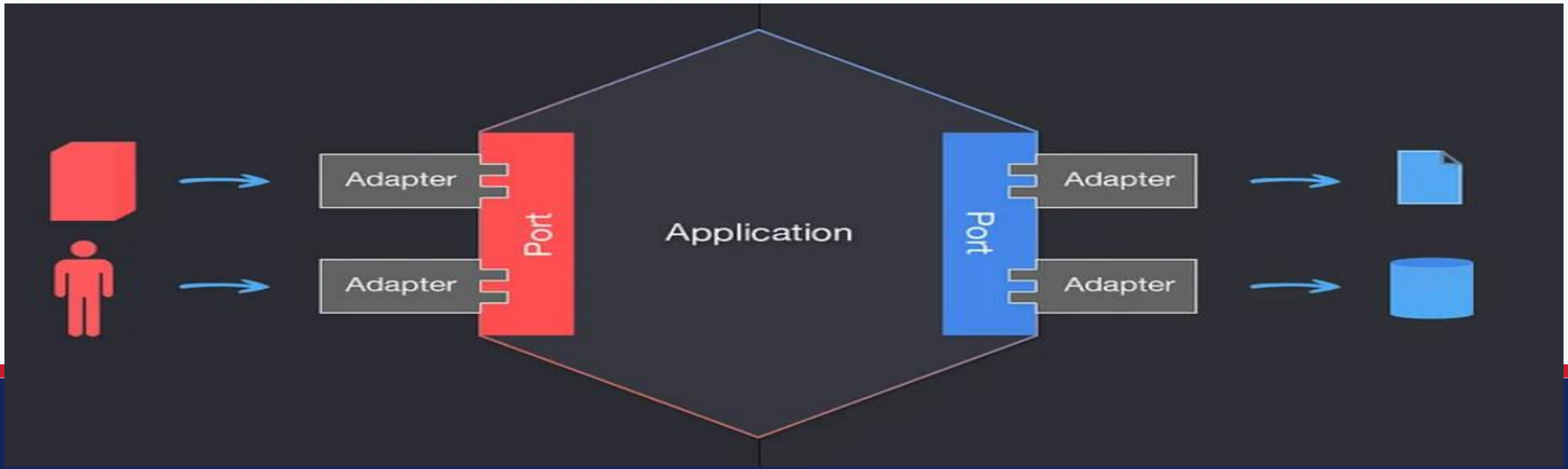
The Hexagonal Architecture

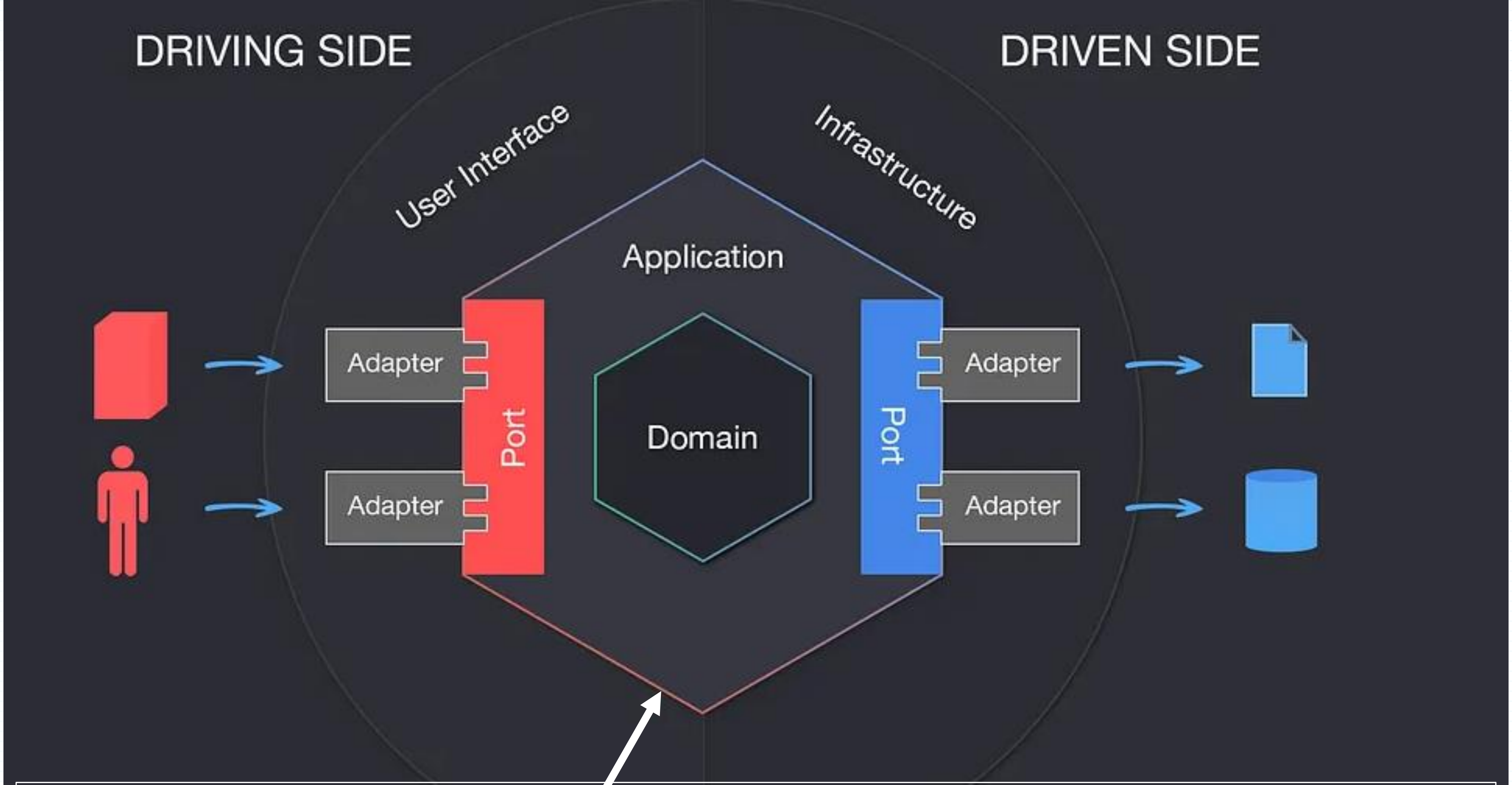
- The Hexagonal Architecture, also referred to as Ports and Adapters, is an architectural pattern that:
  - allows input by users or external systems to **arrive** into the Application at a Port via an Adapter, and allows output to be **sent** out from the Application through a Port to an Adapter.
- This creates an abstraction layer that protects the core of an application and isolates it from external — and somehow irrelevant — tools and technologies.



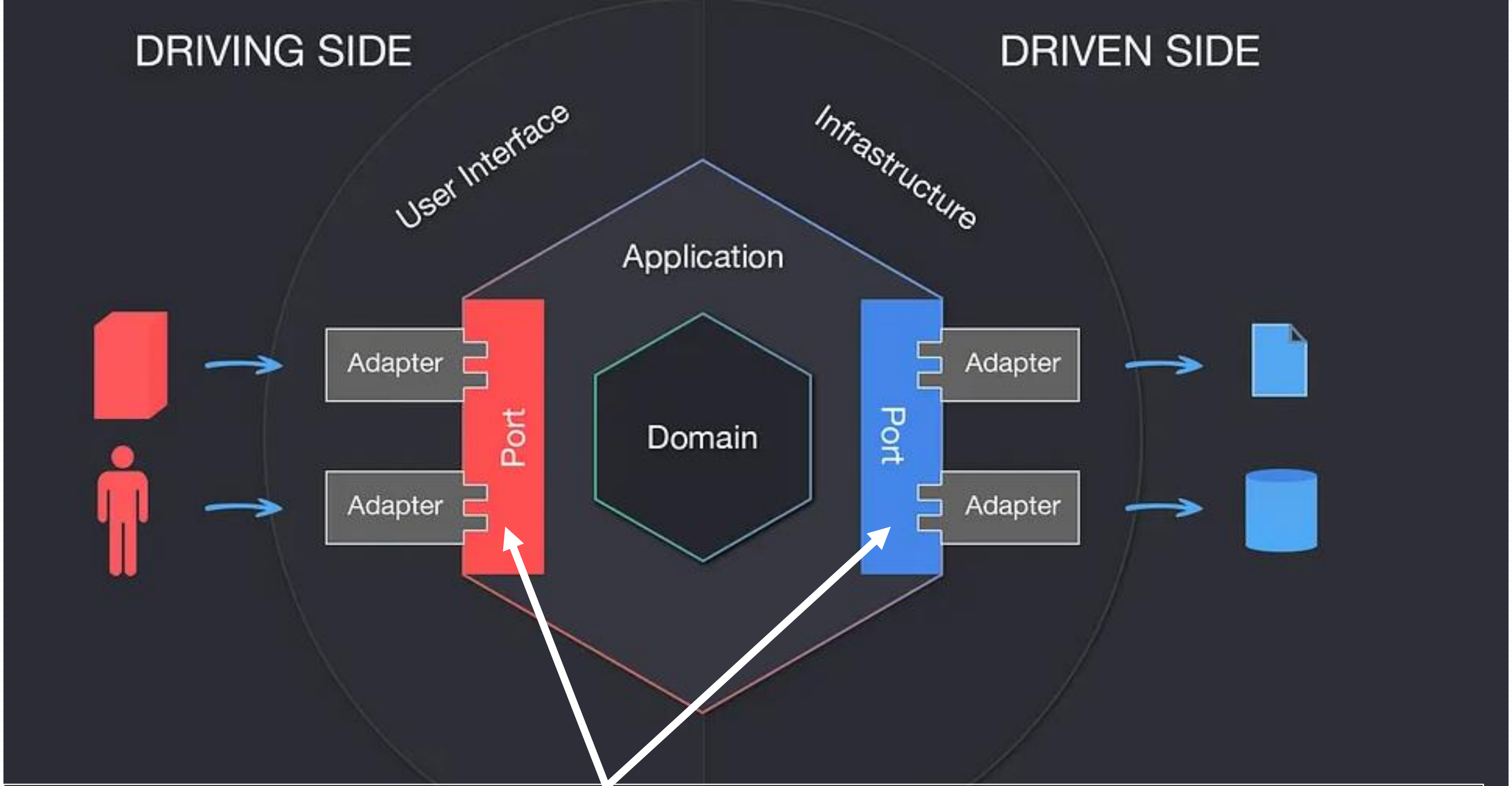
The Hexagonal Architecture

- **Ports**: We can see a Port as a technology-agnostic entry point, it determines the interface which will **allow foreign actors to communicate with the Application.**
- **Adapters:** An Adapter will **initiate the interaction** with the Application through a Port, using a specific technology,
  - for example, a REST controller would represent an adapter that allows a client to communicate with the Application.

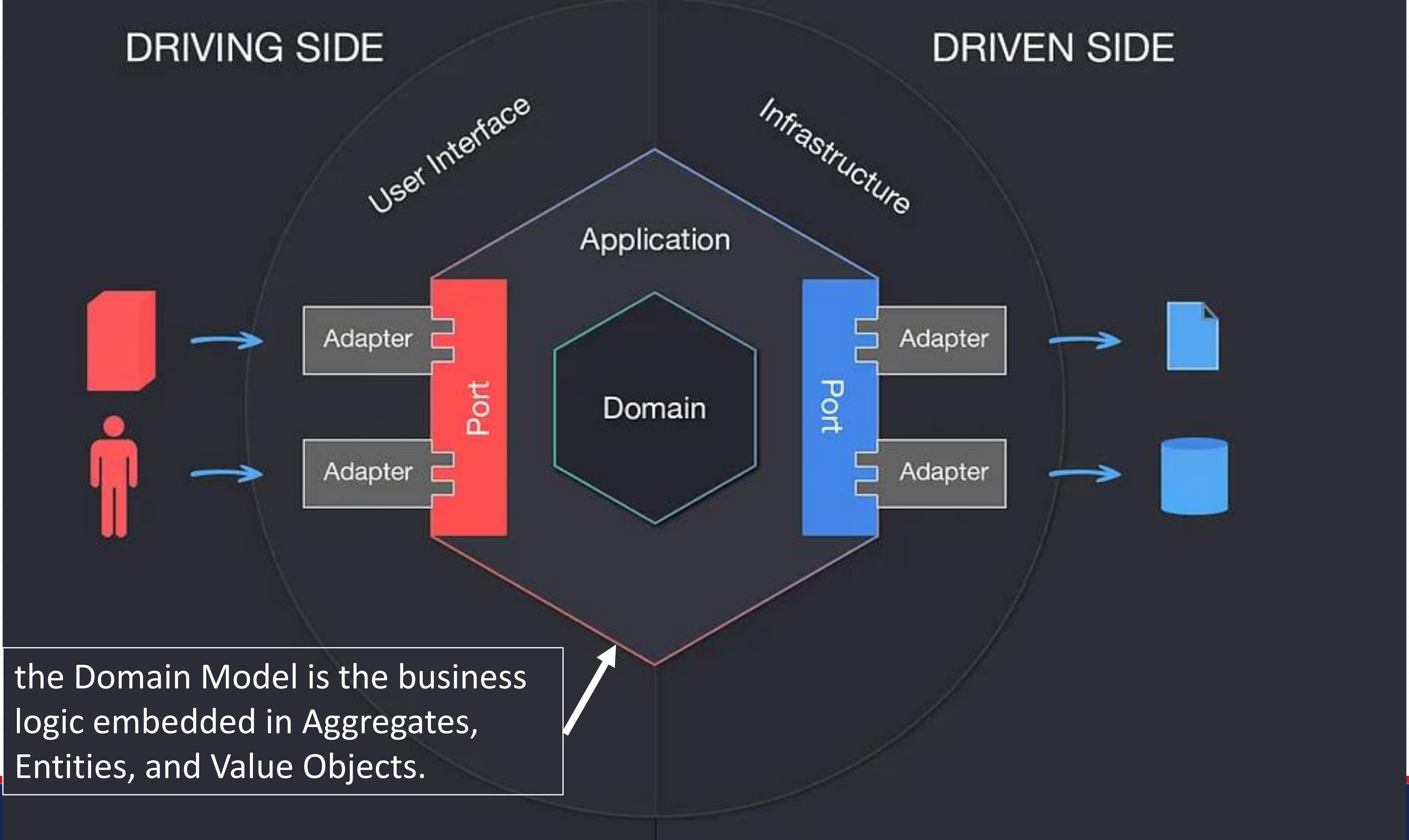




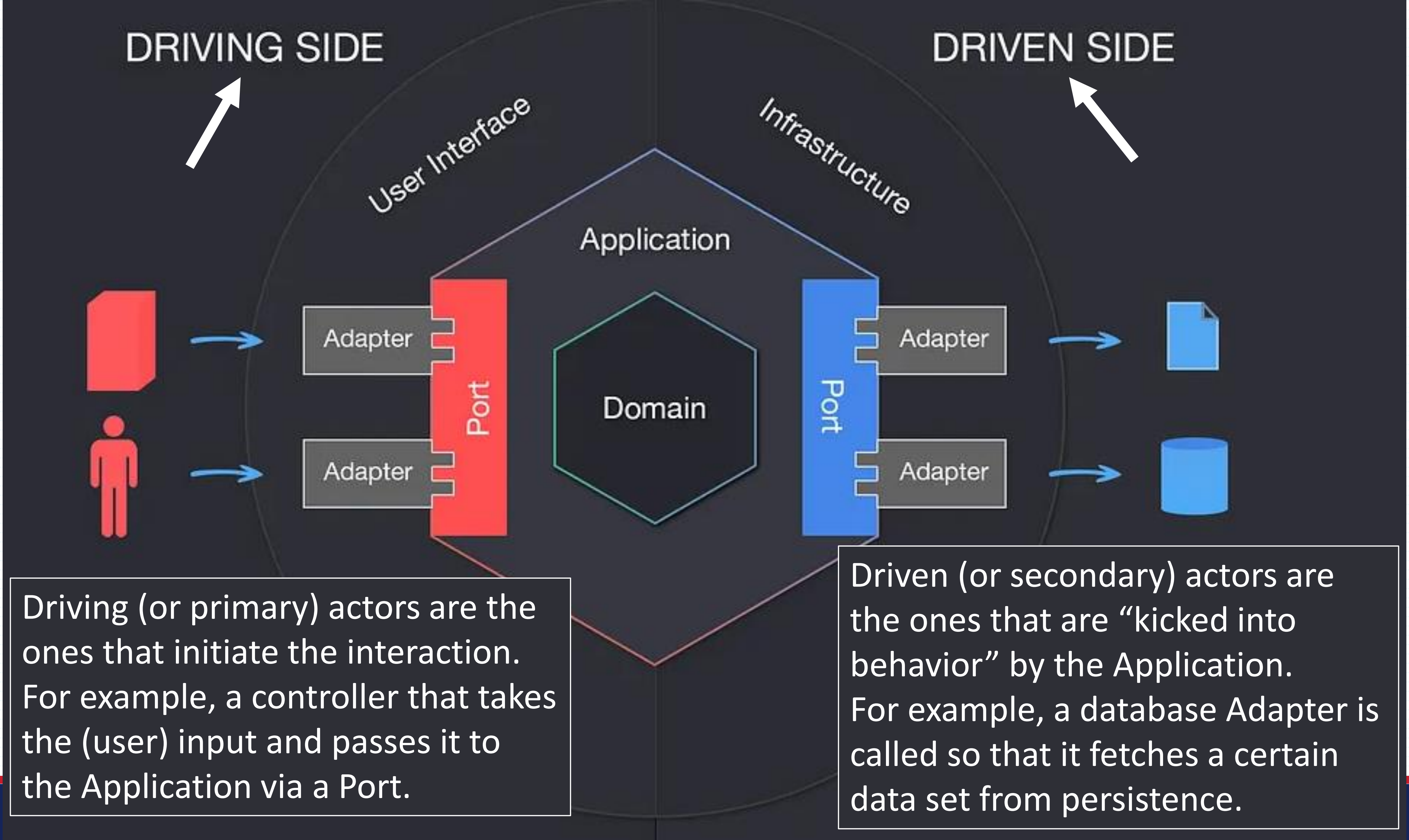
The Application is the core of the system, it contains the Application Services which orchestrate the functionality or the use cases.



The Application receives commands or queries from the Ports, and sends requests out to other external actors, like databases, via Ports as well.



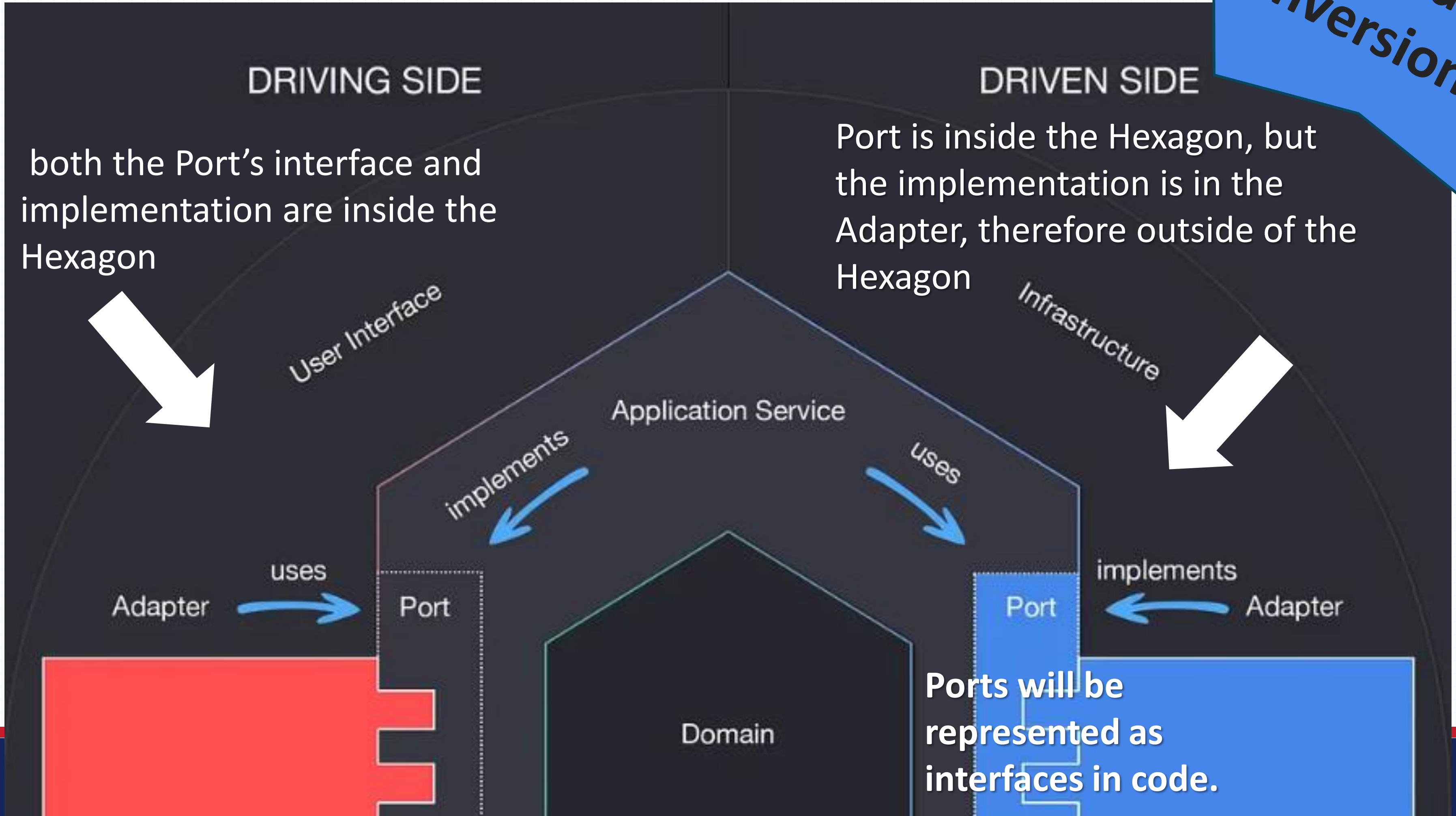
the Domain Model is the business logic embedded in Aggregates, Entities, and Value Objects.



Driving (or primary) actors are the ones that initiate the interaction. For example, a controller that takes the (user) input and passes it to the Application via a Port.

Driven (or secondary) actors are the ones that are “kicked into behavior” by the Application. For example, a database Adapter is called so that it fetches a certain data set from persistence.

The Flow



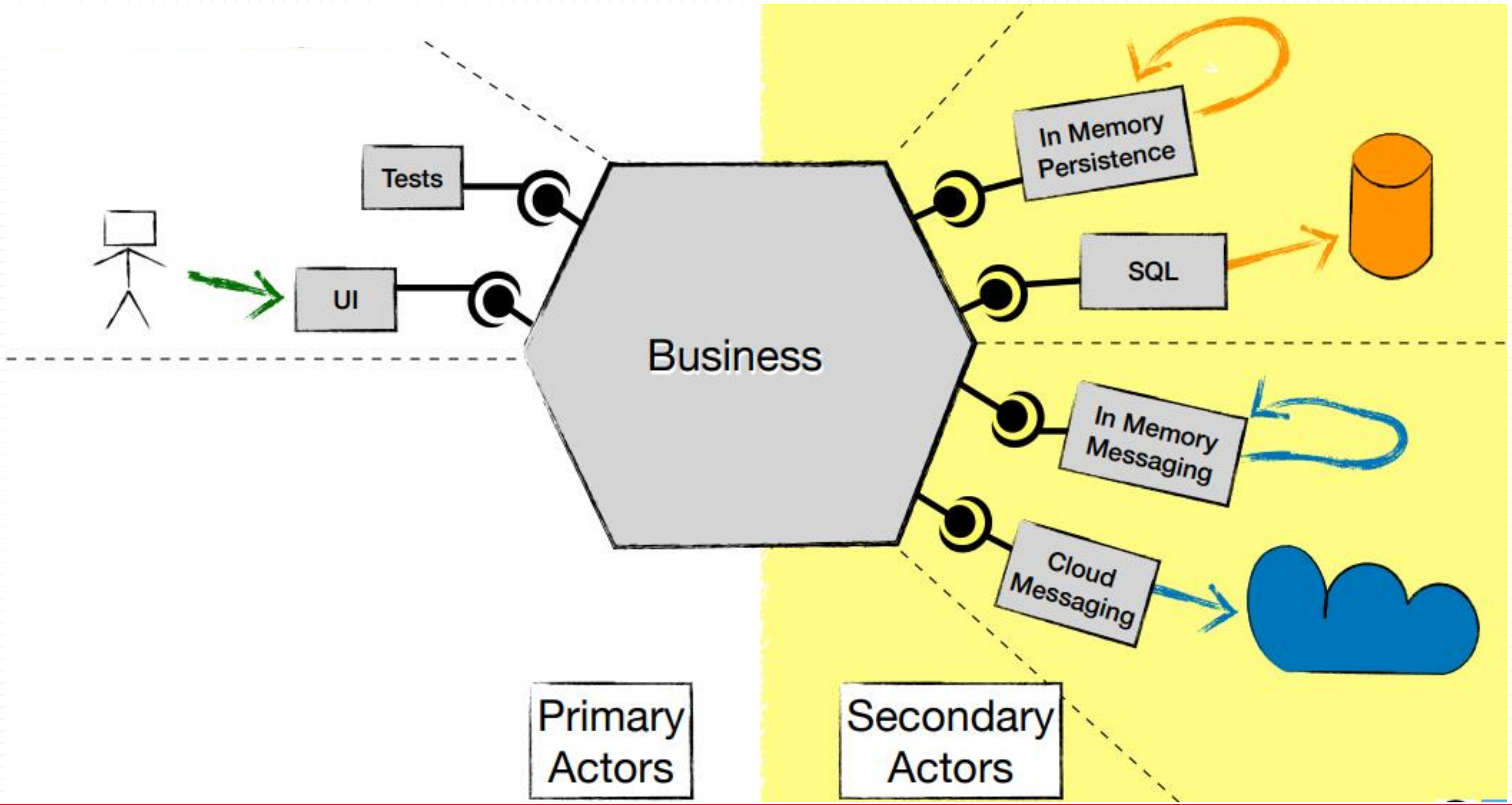
both the Port's interface and implementation are inside the Hexagon

Port is inside the Hexagon, but the implementation is in the Adapter, therefore outside of the Hexagon

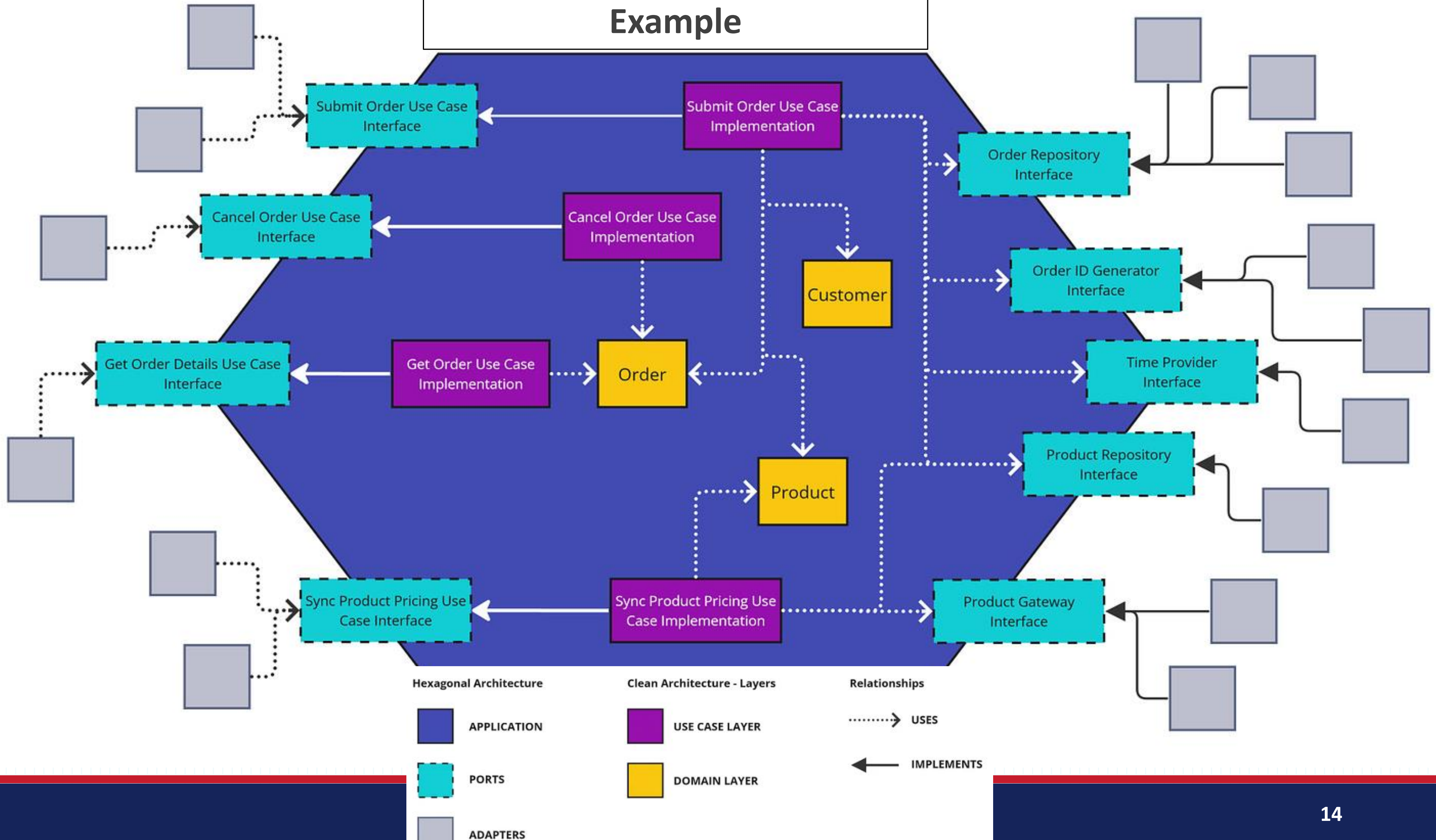
Dependency Inversion ☺

Ports will be represented as interfaces in code.

Ports and Adapters

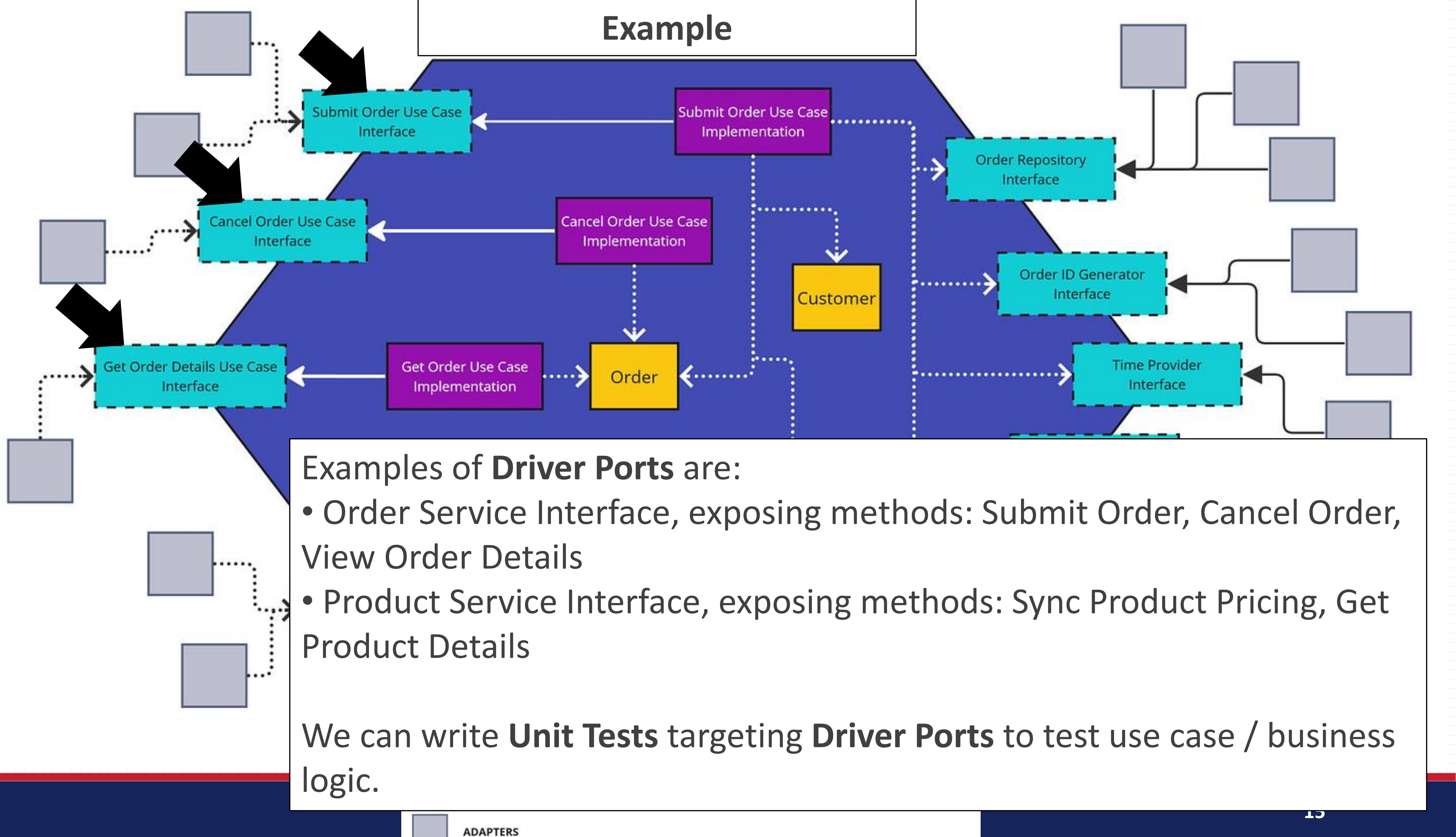


Example



Submit Order Use Case Interface
Submit Order Use Case Implementation
Order Repository Interface
Cancel Order Use Case Interface
Cancel Order Use Case Implementation
Customer
Order ID Generator Interface
Get Order Details Use Case Interface
Get Order Use Case Implementation
Order
Time Provider Interface
Product Repository Interface
Product
Sync Product Pricing Use Case Interface
Sync Product Pricing Use Case Implementation
Product Gateway Interface
Hexagonal Architecture
APPLICATION
PORTS
ADAPTERS
Clean Architecture - Layers
USE CASE LAYER
DOMAIN LAYER
Relationships
USES
IMPLEMENTS

Example



Submit Order Use Case Interface
Submit Order Use Case Implementation
Cancel Order Use Case Interface
Cancel Order Use Case Implementation
Get Order Details Use Case Interface
Get Order Use Case Implementation
Order
Customer
Order Repository Interface
Order ID Generator Interface
Time Provider Interface
ADAPTERS

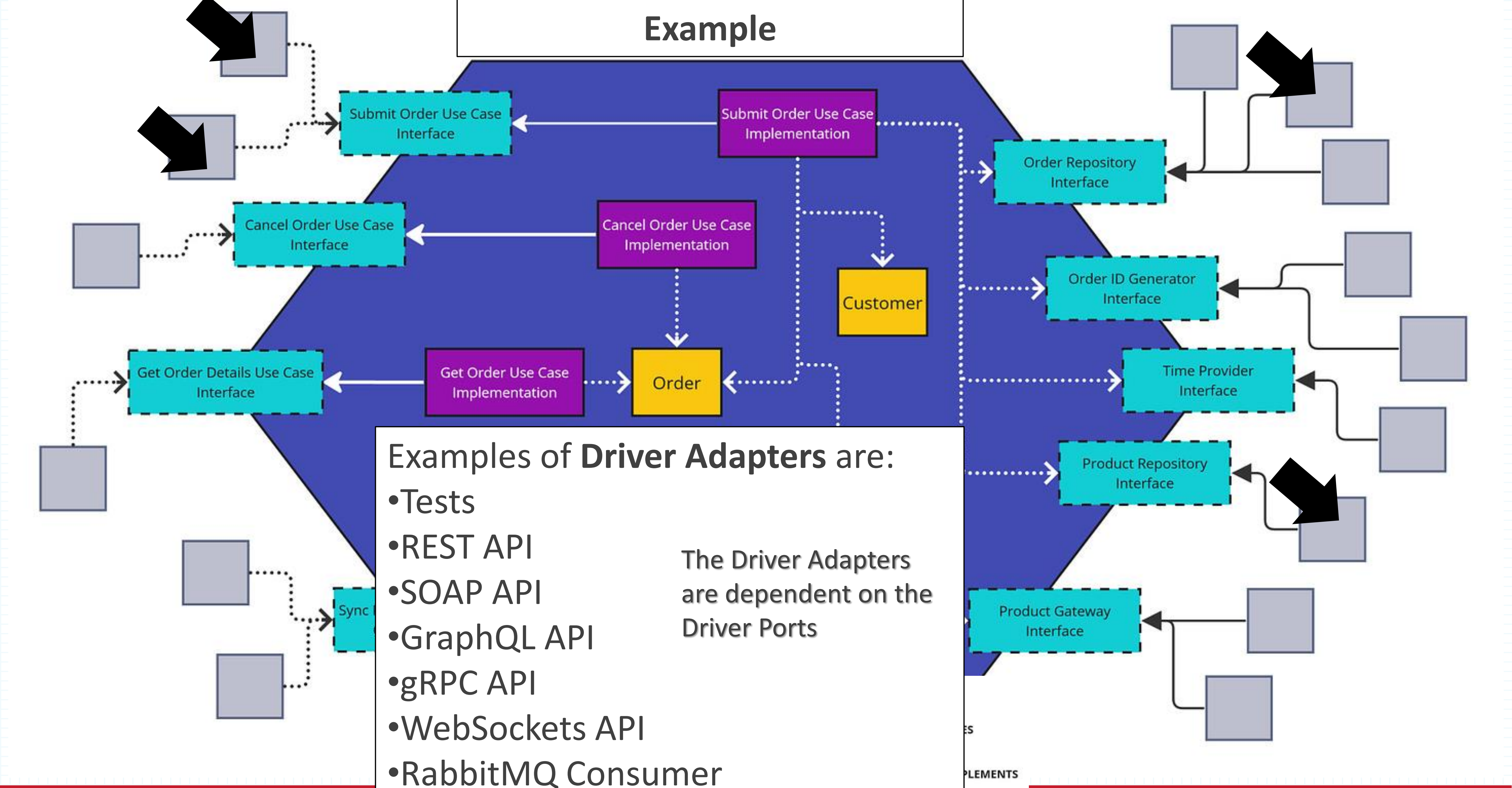
Examples of **Driver Ports** are:

• Order Service Interface, exposing methods: Submit Order, Cancel Order, View Order Details

• Product Service Interface, exposing methods: Sync Product Pricing, Get Product Details

We can write **Unit Tests** targeting **Driver Ports** to test use case / business logic.

Example

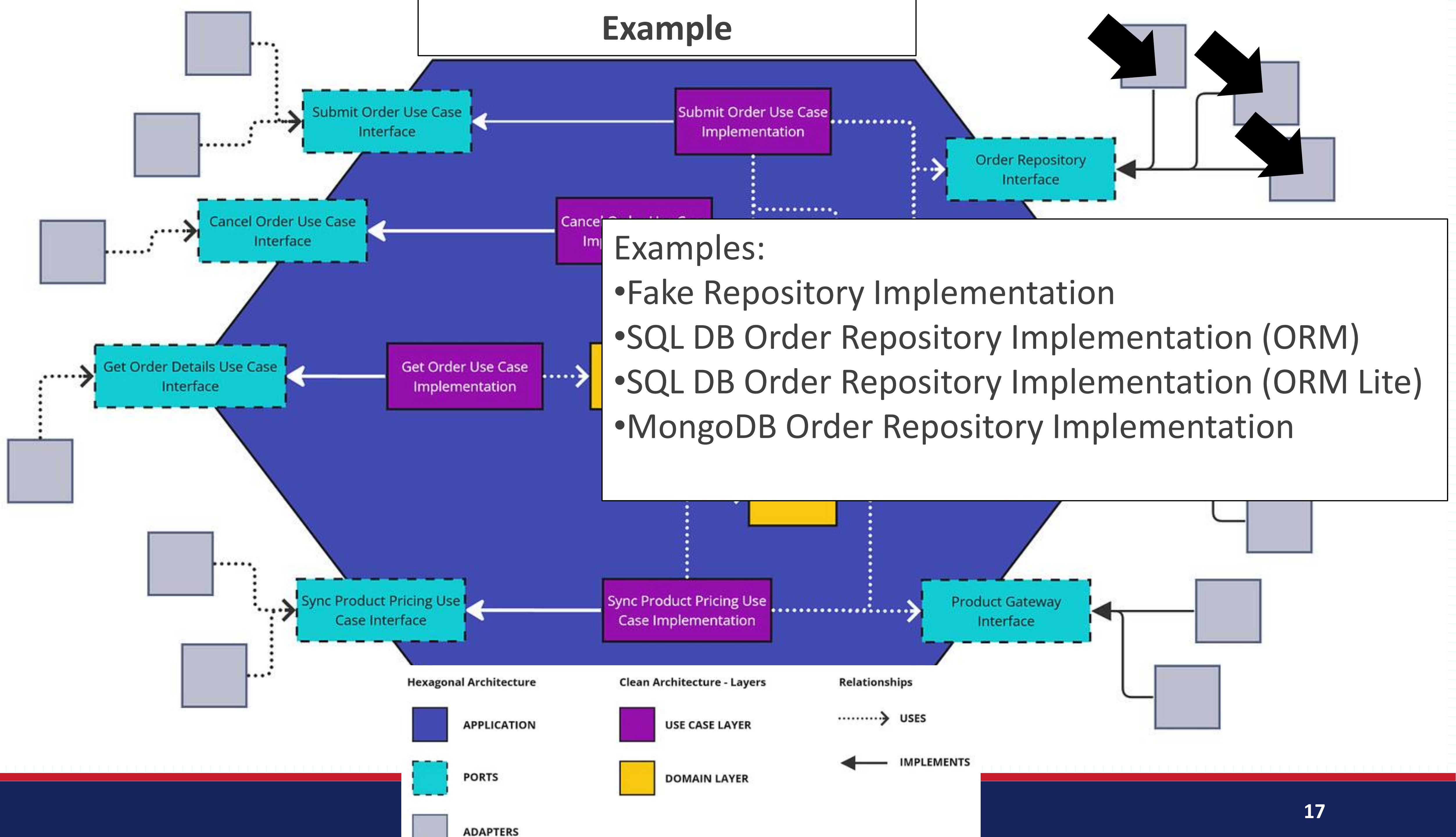

Submit Order Use Case Interface
Submit Order Use Case Implementation
Cancel Order Use Case Interface
Cancel Order Use Case Implementation
Get Order Details Use Case Interface
Get Order Use Case Implementation
Order
Customer
Order Repository Interface
Order ID Generator Interface
Time Provider Interface
Product Repository Interface
Product Gateway Interface
Sync
ES
PLEMENTS

Examples of **Driver Adapters** are:

- •Tests
- •REST API
- •SOAP API
- •GraphQL API
- •gRPC API
- •WebSockets API
- •RabbitMQ Consumer
- •Kafka Consumer

The Driver Adapters are dependent on the Driver Ports

Example

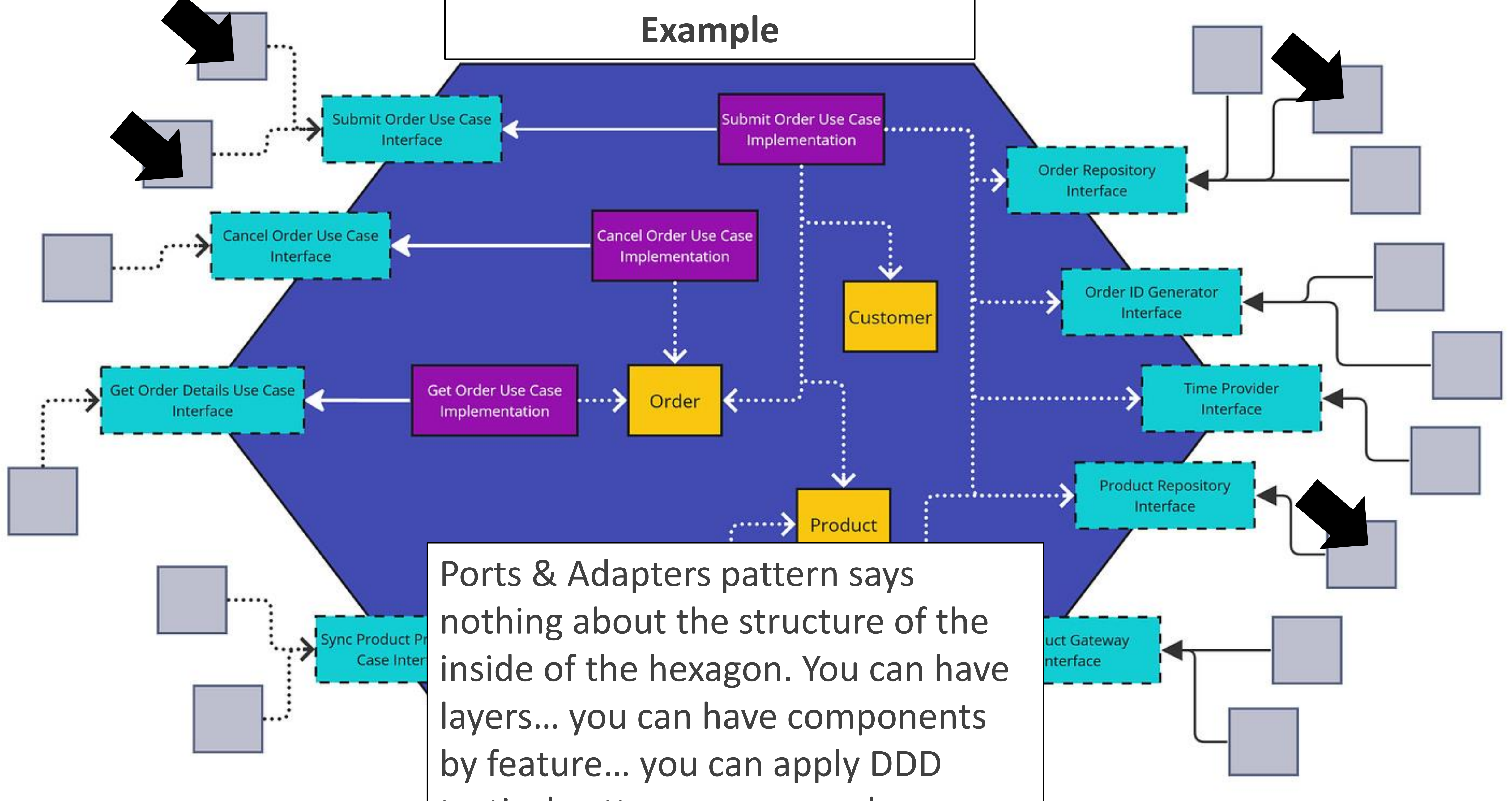


Submit Order Use Case Interface
Submit Order Use Case Implementation
Order Repository Interface
Cancel Order Use Case Interface
Cancel
Imp
Get Order Details Use Case Interface
Get Order Use Case Implementation
Sync Product Pricing Use Case Interface
Sync Product Pricing Use Case Implementation
Product Gateway Interface
Hexagonal Architecture
APPLICATION
PORTS
ADAPTERS
Clean Architecture - Layers
USE CASE LAYER
DOMAIN LAYER
Relationships
USES
IMPLEMENTS

Examples:

- •Fake Repository Implementation
- •SQL DB Order Repository Implementation (ORM)
- •SQL DB Order Repository Implementation (ORM Lite)
- •MongoDB Order Repository Implementation

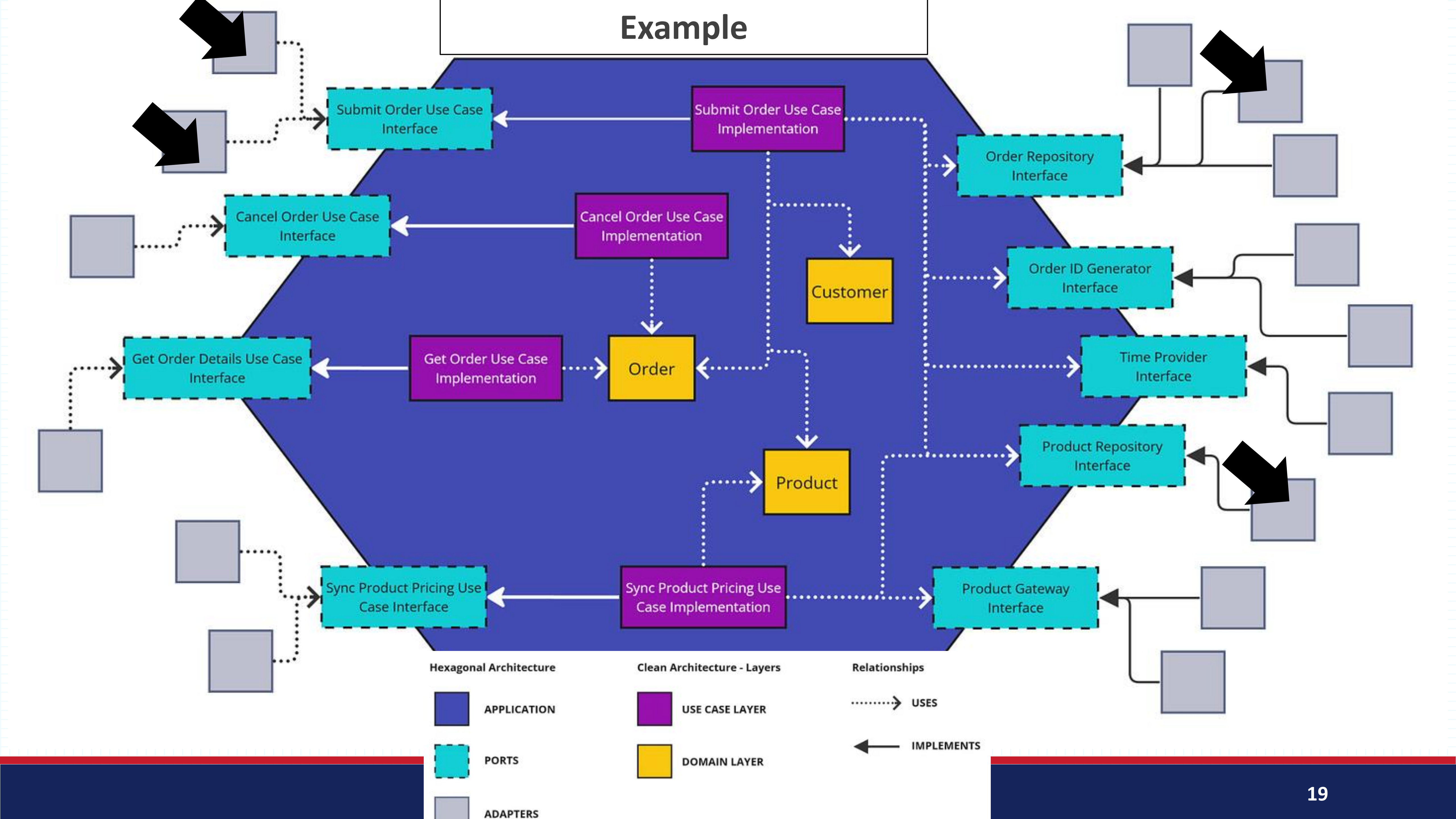
Example

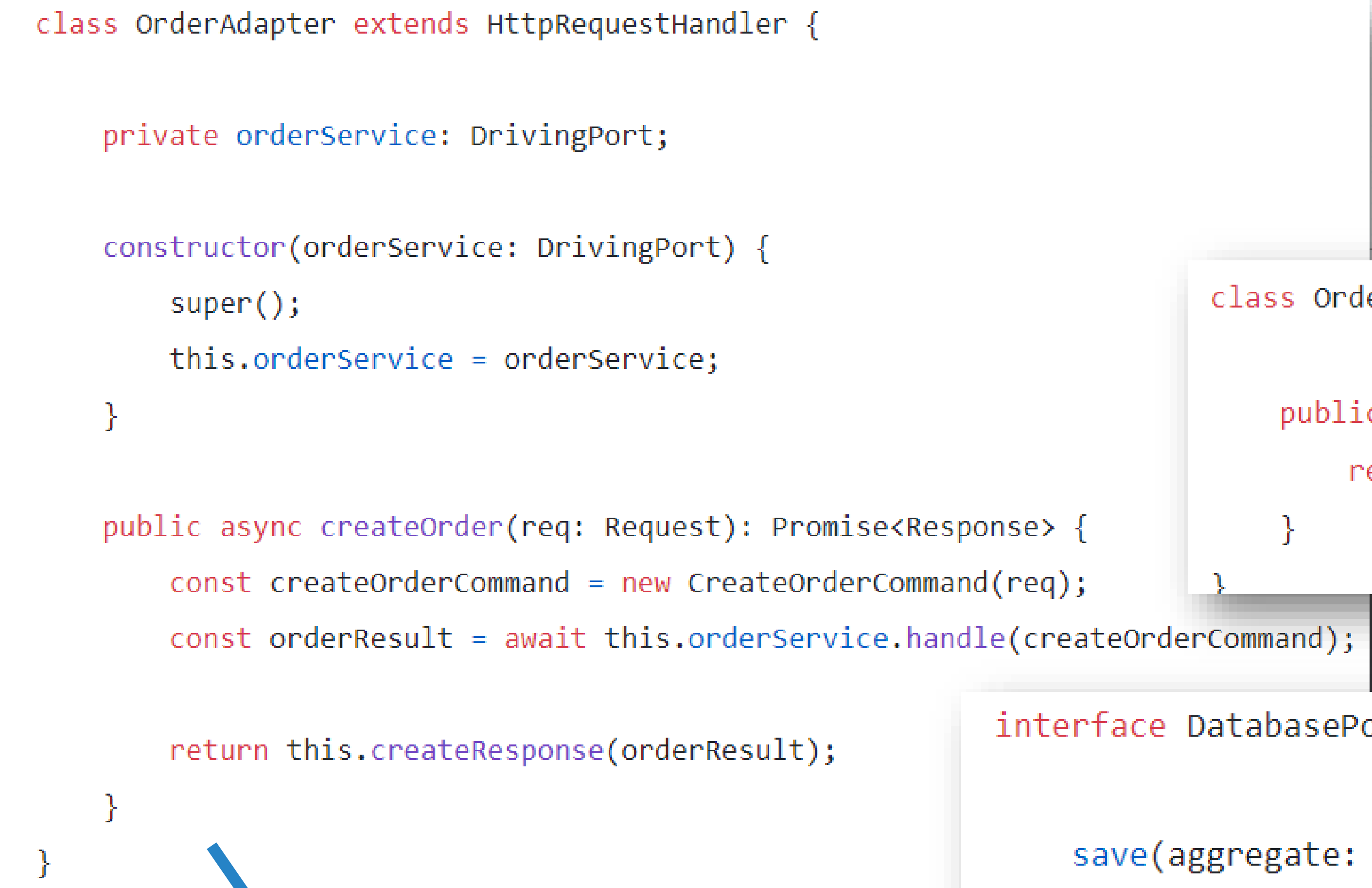


Submit Order Use Case Interface
Submit Order Use Case Implementation
Cancel Order Use Case Interface
Cancel Order Use Case Implementation
Get Order Details Use Case Interface
Get Order Use Case Implementation
Order
Customer
Product
Order Repository Interface
Order ID Generator Interface
Time Provider Interface
Product Repository Interface
Sync Product Pr
Case Inter
uct Gateway
nterface

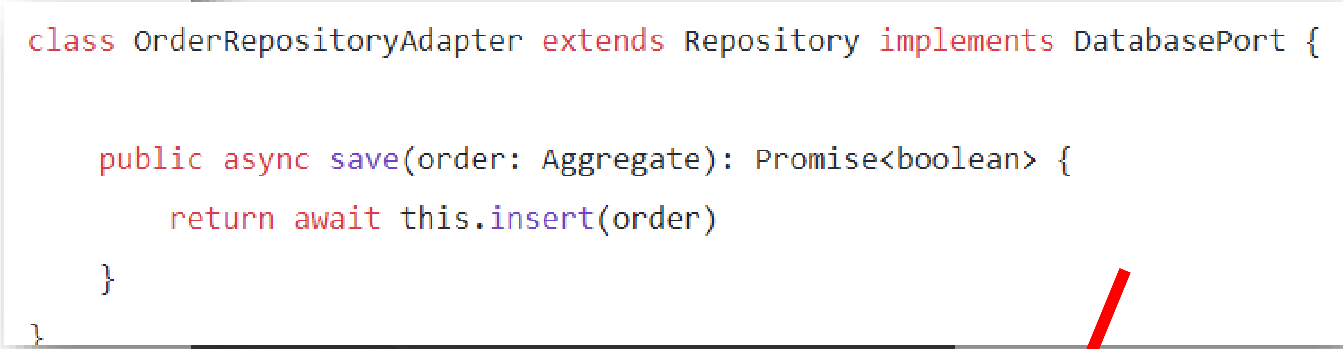
Ports & Adapters pattern says nothing about the structure of the inside of the hexagon. You can have layers… you can have components by feature… you can apply DDD tactical patterns… you can have a single CRUD… it's up to you.

Example

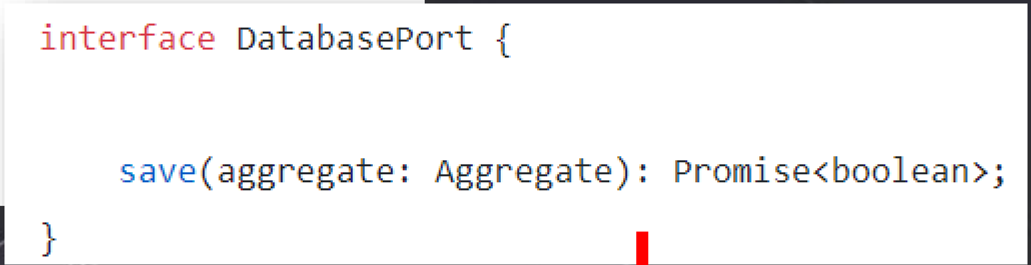

Submit Order Use Case Interface
Submit Order Use Case Implementation
Cancel Order Use Case Interface
Cancel Order Use Case Implementation
Order Repository Interface
Customer
Order ID Generator Interface
Get Order Details Use Case Interface
Get Order Use Case Implementation
Order
Time Provider Interface
Product Repository Interface
Product
Sync Product Pricing Use Case Interface
Sync Product Pricing Use Case Implementation
Product Gateway Interface
Hexagonal Architecture
APPLICATION
PORTS
ADAPTERS
Clean Architecture - Layers
USE CASE LAYER
DOMAIN LAYER
Relationships
USES
IMPLEMENTS



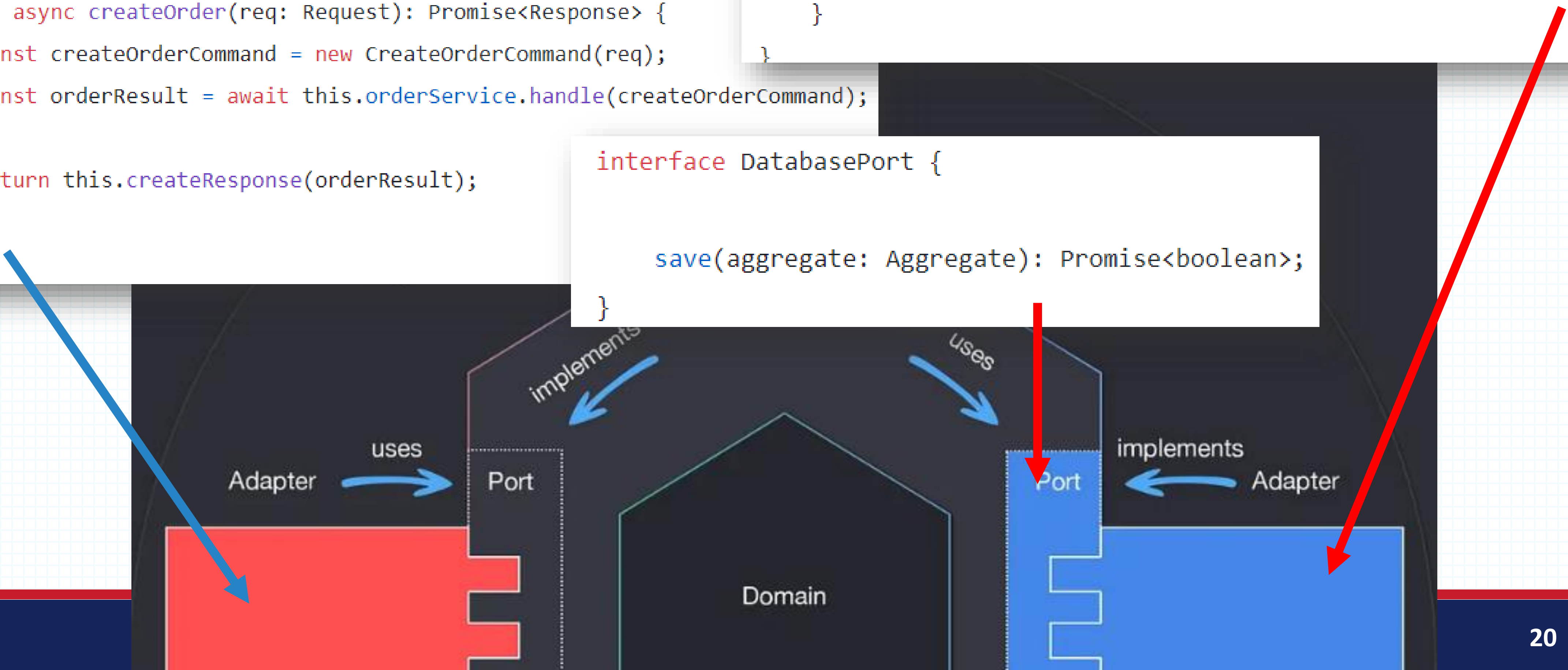
```
class OrderAdapter extends HttpRequestHandler {

    private orderService: DrivingPort;

    constructor(orderService: DrivingPort) {
        super();
        this.orderService = orderService;
    }

    public async createOrder(req: Request): Promise<Response> {
        const createOrderCommand = new CreateOrderCommand(req);
        const orderResult = await this.orderService.handle(createOrderCommand);

        return this.createResponse(orderResult);
    }
}
```



```
class OrderRepositoryAdapter extends Repository implements DatabasePort {

    public async save(order: Aggregate): Promise<boolean> {
        return await this.insert(order)
    }
}
```



```
interface DatabasePort {

    save(aggregate: Aggregate): Promise<boolean>;
}
```


implements
uses
Adapter
uses
Port
Domain
Port
implements
Adapter