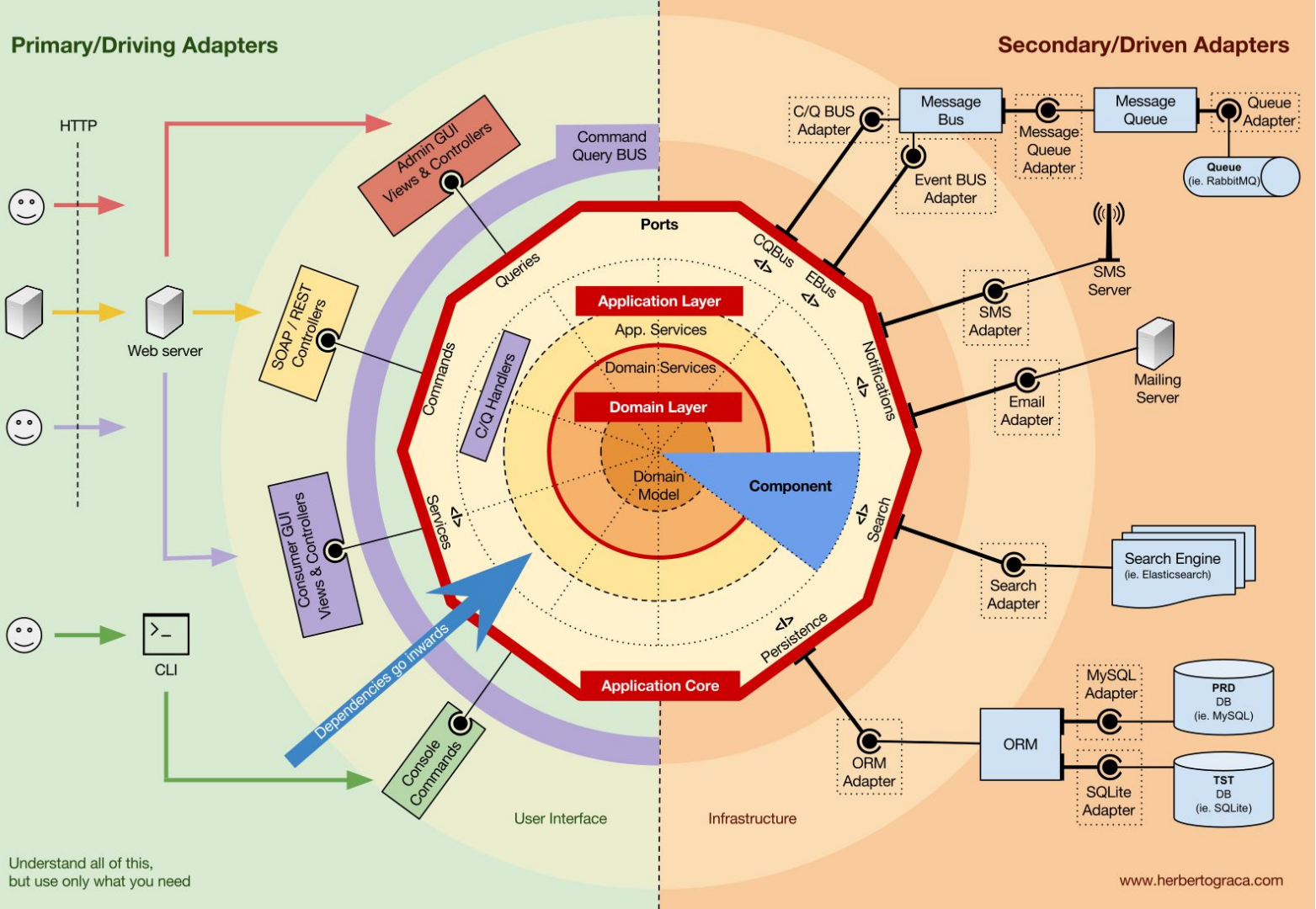
Why not all hexagons have eight sides?

sli.do: #geecon
Room 11


Primary/Driving Adapters
Secondary/Driven Adapters
HTTP
Web server
CLI
Admin GUI
Views & Controllers
Command
Query BUS
SOAP / REST
Controllers
Consumer GUI
Views & Controllers
Console
Commands
Dependencies go inwards
Ports
Queries
Commands
C/Q Handlers
Application Layer
App. Services
Domain Services
Domain Layer
Domain
Model
Component
Application Core
CQBus
EBus
Notifications
Search
Persistence
Services
C/Q BUS
Adapter
Message
Bus
Message
Queue
Adapter
Message
Queue
Queue
Adapter
Queue
(ie. RabbitMQ)
Event BUS
Adapter
SMS
Server
SMS
Adapter
Mailing
Server
Email
Adapter
Search Engine
(ie. Elasticsearch)
Search
Adapter
MySQL
Adapter
PRD
DB
(ie. MySQL)
ORM
ORM
Adapter
SQLite
Adapter
TST
DB
(ie. SQLite)
User Interface
Infrastructure
Understand all of this,
but use only what you need
www.herbertograca.com

$ whoami

Krzysiek Przygudzki

Software developer

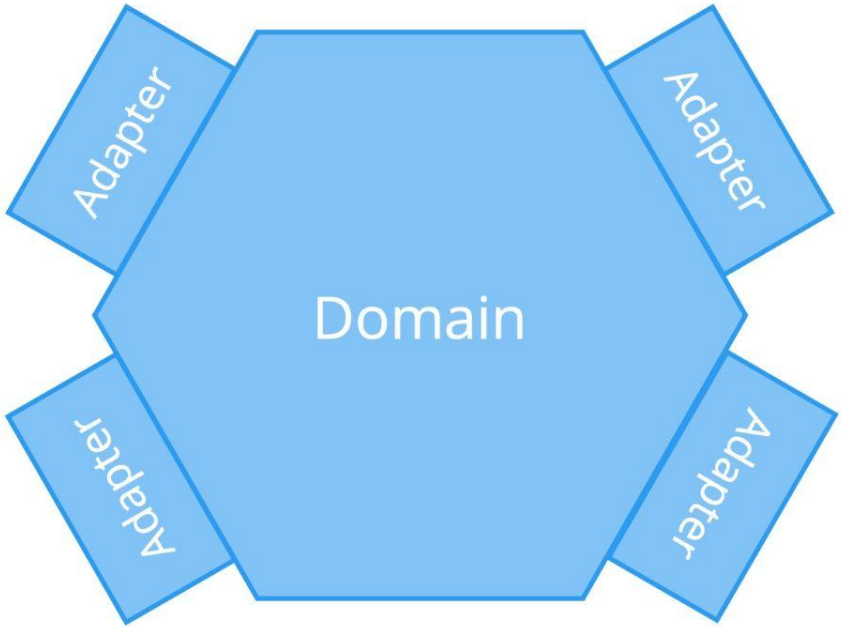
ex- Hazelcast, TouK, Allegro Smart

~~twitter~~ X: @kprzygudzki

Caveat auditor

The information contained within this presentation represents the views and opinions of the author and does not necessarily represent the views or opinions of any company or organisation.

Hexagonal architecture



Extracting functions

Decomposition, cognitive load reduction

Abstraction

Code reuse instead of duplication

Code that changes together is located together, cohesion

Extracting classes

OOP => objects

Co-locate state and logic, co-locate changes

Encapsulate state, expose behaviours, reduce coupling

Abstraction, reduction in cognitive load

Extracting packages

Vertical division – based on the domain context

Logical order, easy navigation

Co-locating changes, cohesion

Horizontal division

Horizontal division – based on the technical context

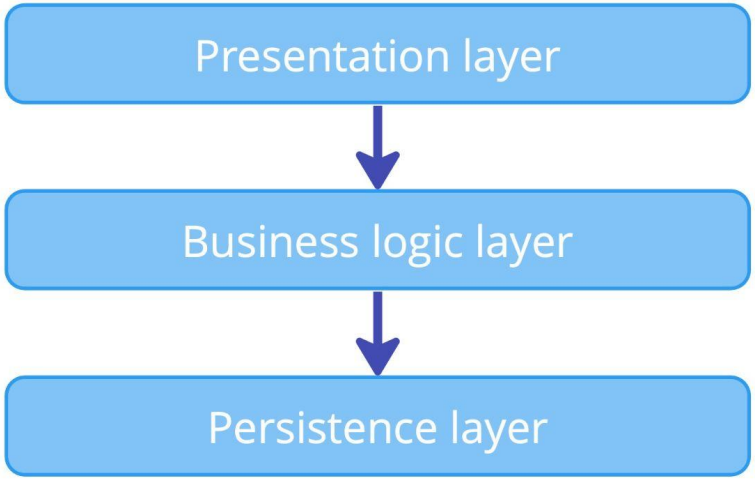
Logical order, easy navigation

Layered architecture

Classic (outdated) layered architecture

Consistent direction of dependency

Useful (?)

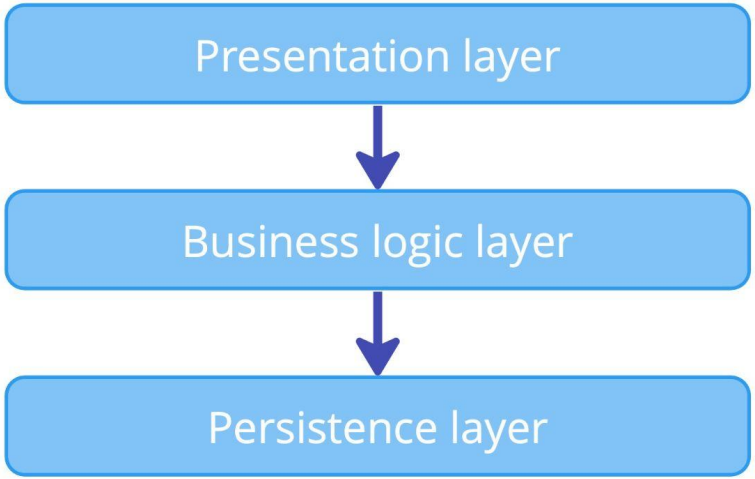


Database-centric architecture

Everything depends on persistence layer

Anaemic domain model

Complexity at the Heart of Software



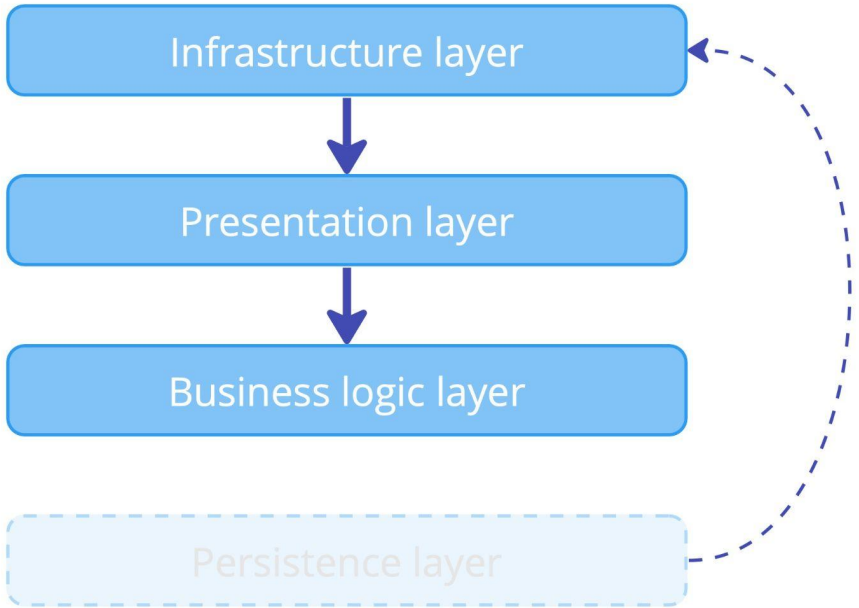
Layered architecture – a new start

Dependency inversion

~~persistence~~ infrastructure layer to the top

Modern layered architecture

Business logic perfectly isolated

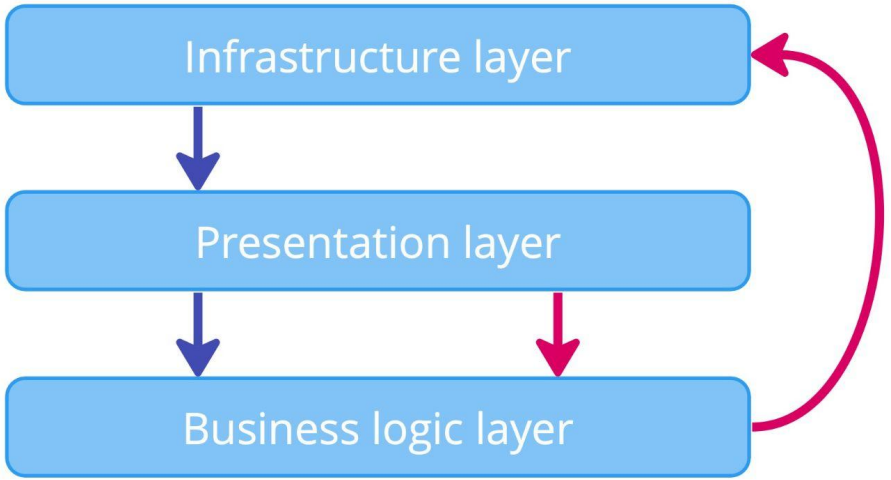


Why hexagonal architecture?

Direction of dependency vs direction of control

Direction of dependency: top-to-bottom

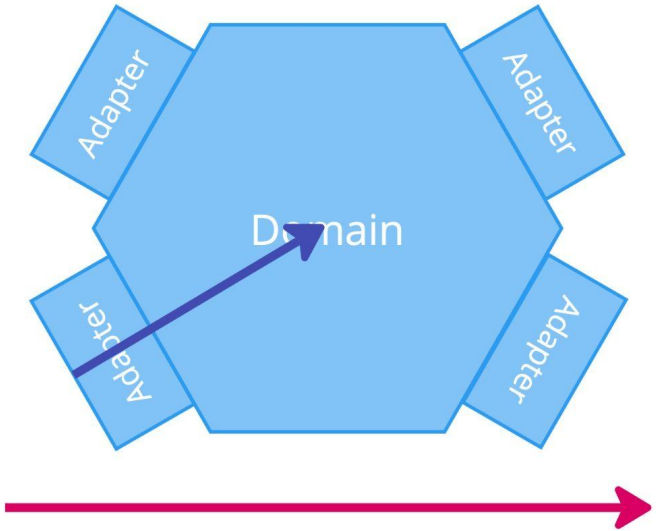
Direction of control: *inconsistent*



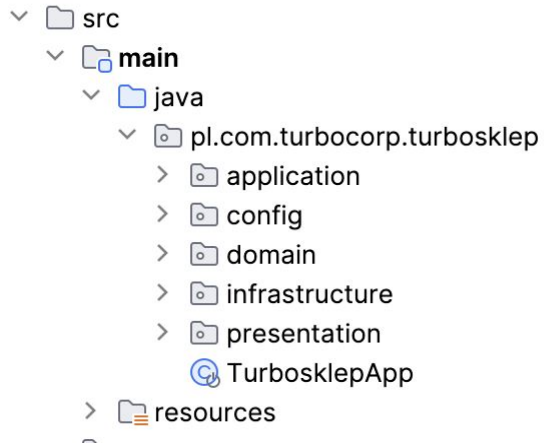
Direction of dependency vs direction of control

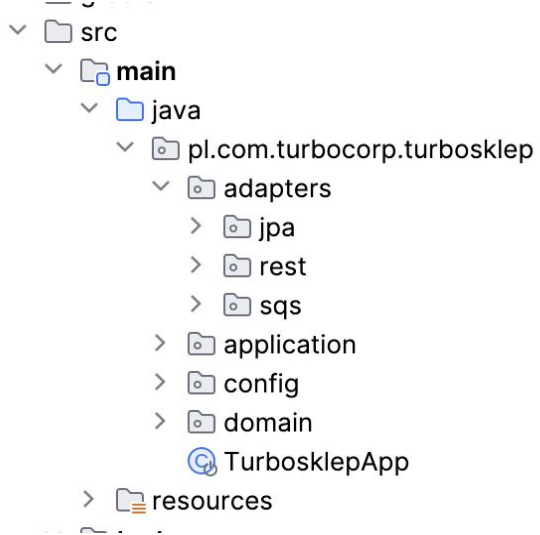
Direction of dependency: outside-in

Direction of control: left-to-right

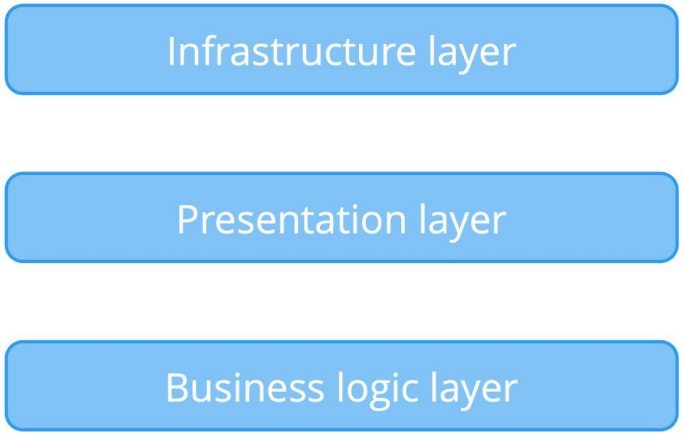


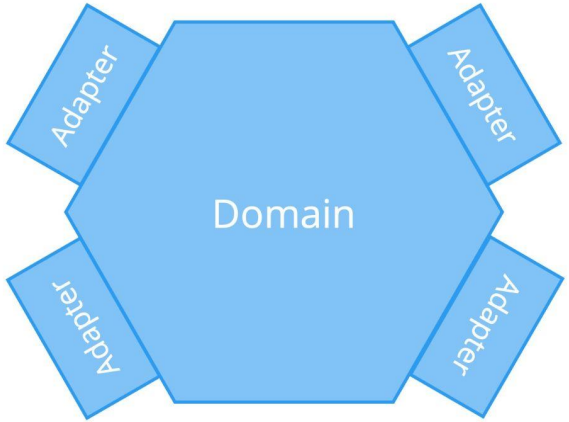
Project structure





Project structure





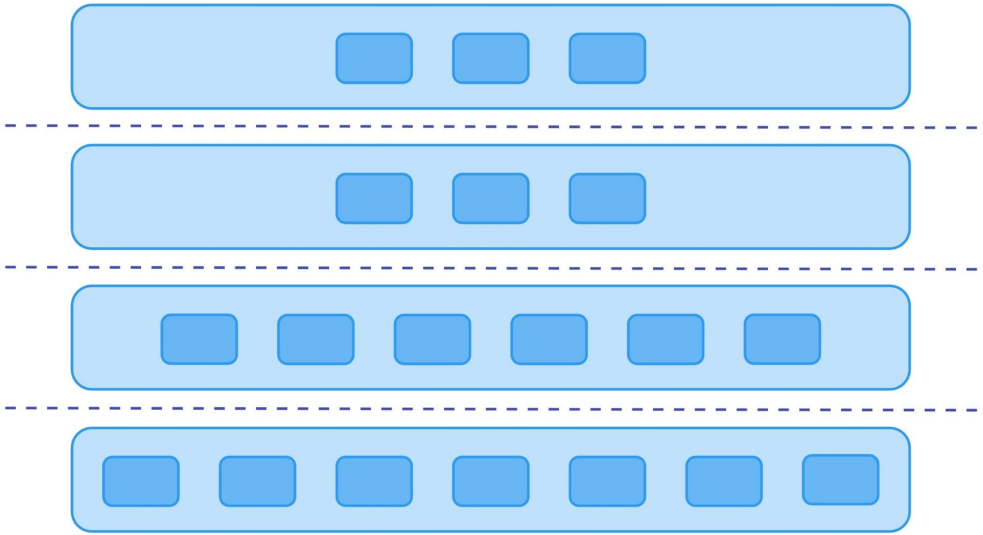
What if the layers grow too large?

Horizontal division

More layers!

Horizontal division *within* existing layers

Cohesion?

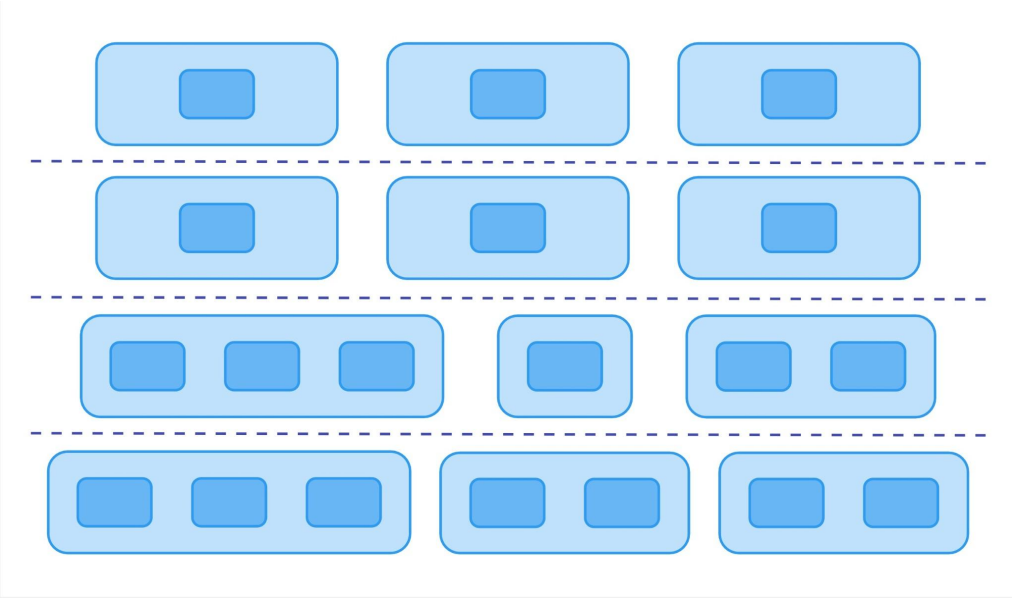


Vertical division

Vertical division within existing layers

Cohesion

Coupling?

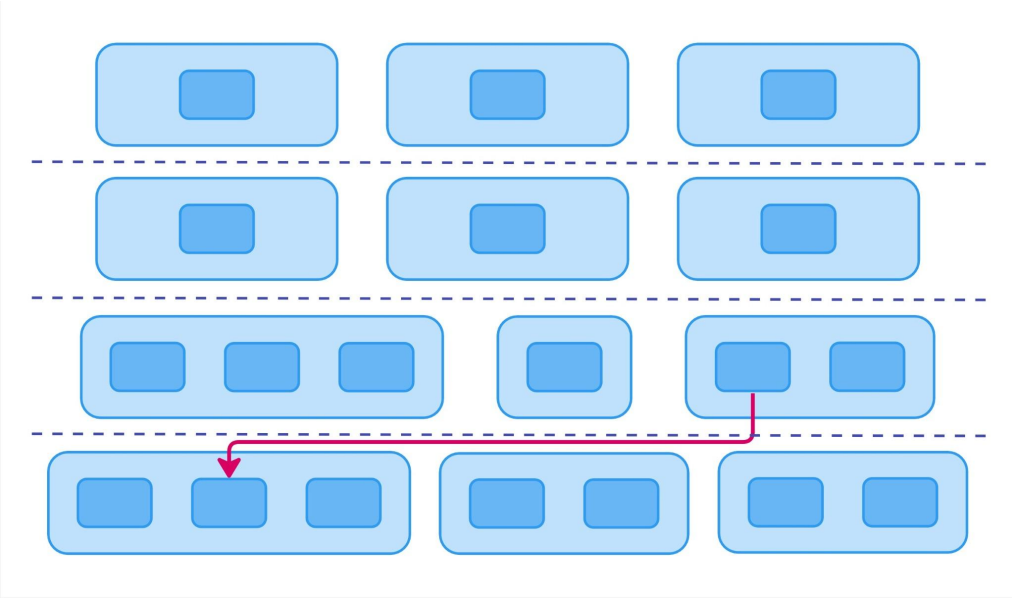


Vertical division

Vertical division within existing layers

Cohesion

Coupling?

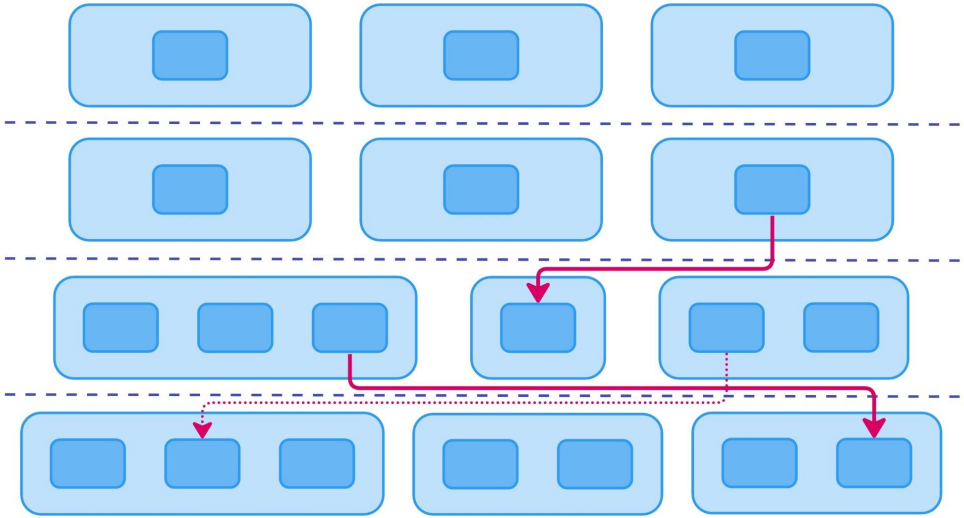


Vertical division

Vertical division within existing layers

Cohesion

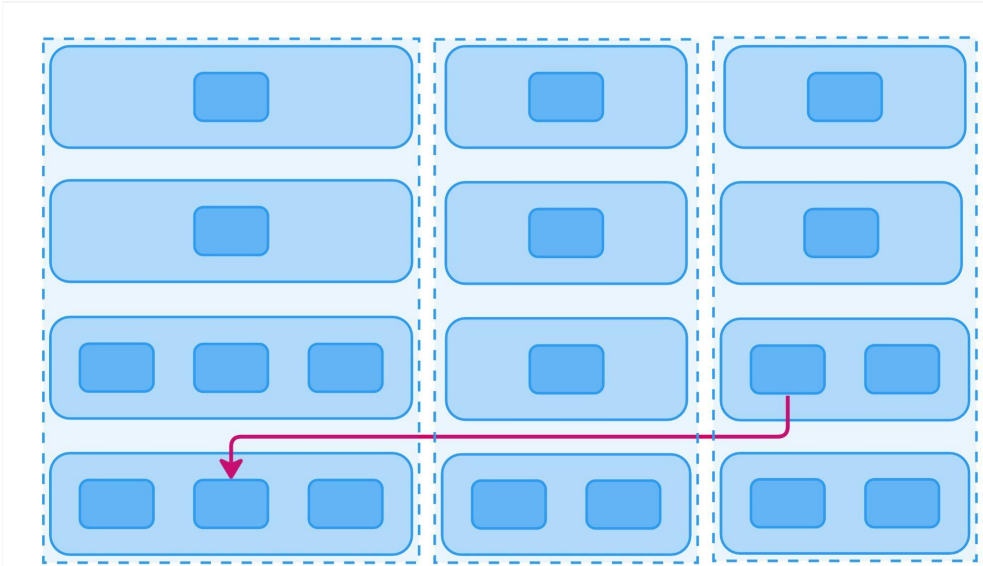
Coupling?



Vertical-first

Modules

Layers *within* modules

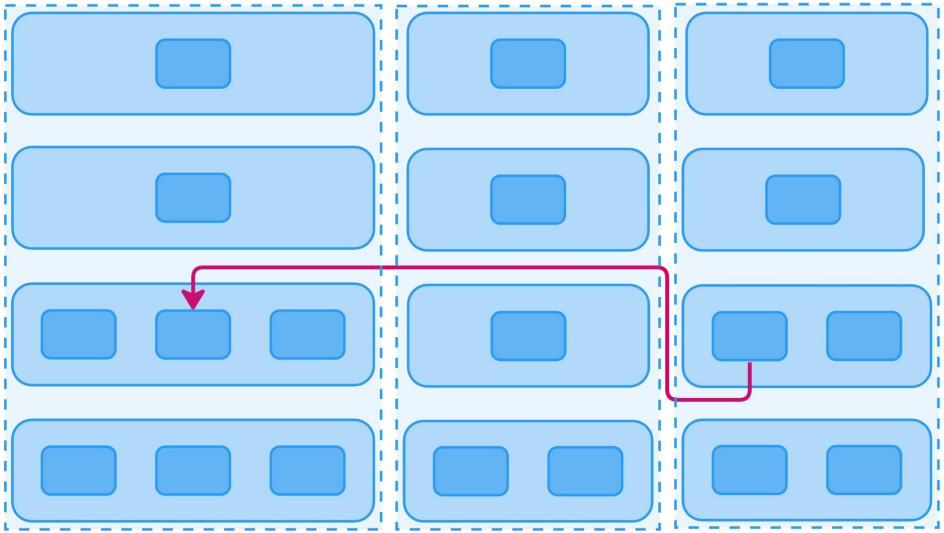


Vertical-first

Modules

Layers *within* modules

Modules with APIs



Vertical-first

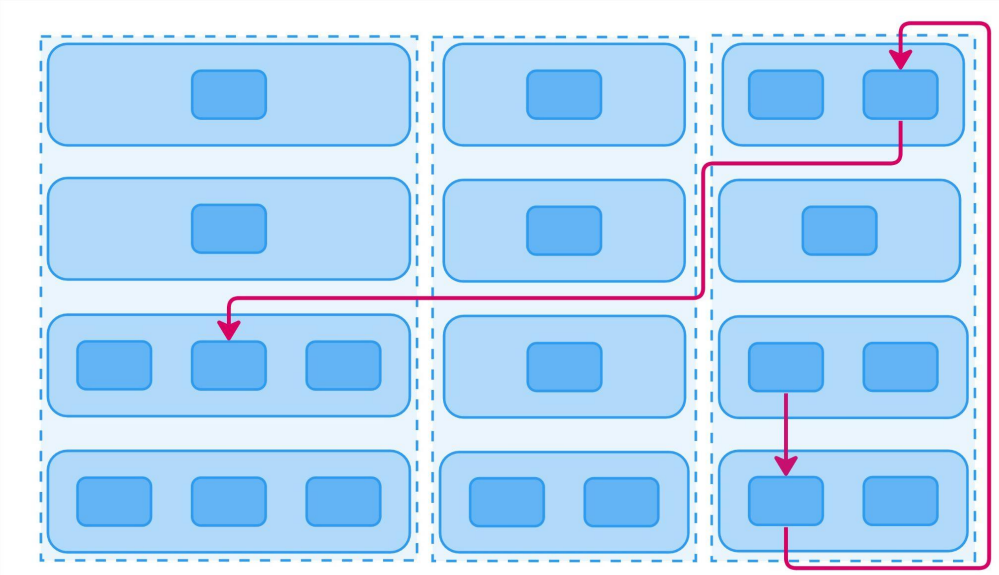
Modules

Layers *within* modules

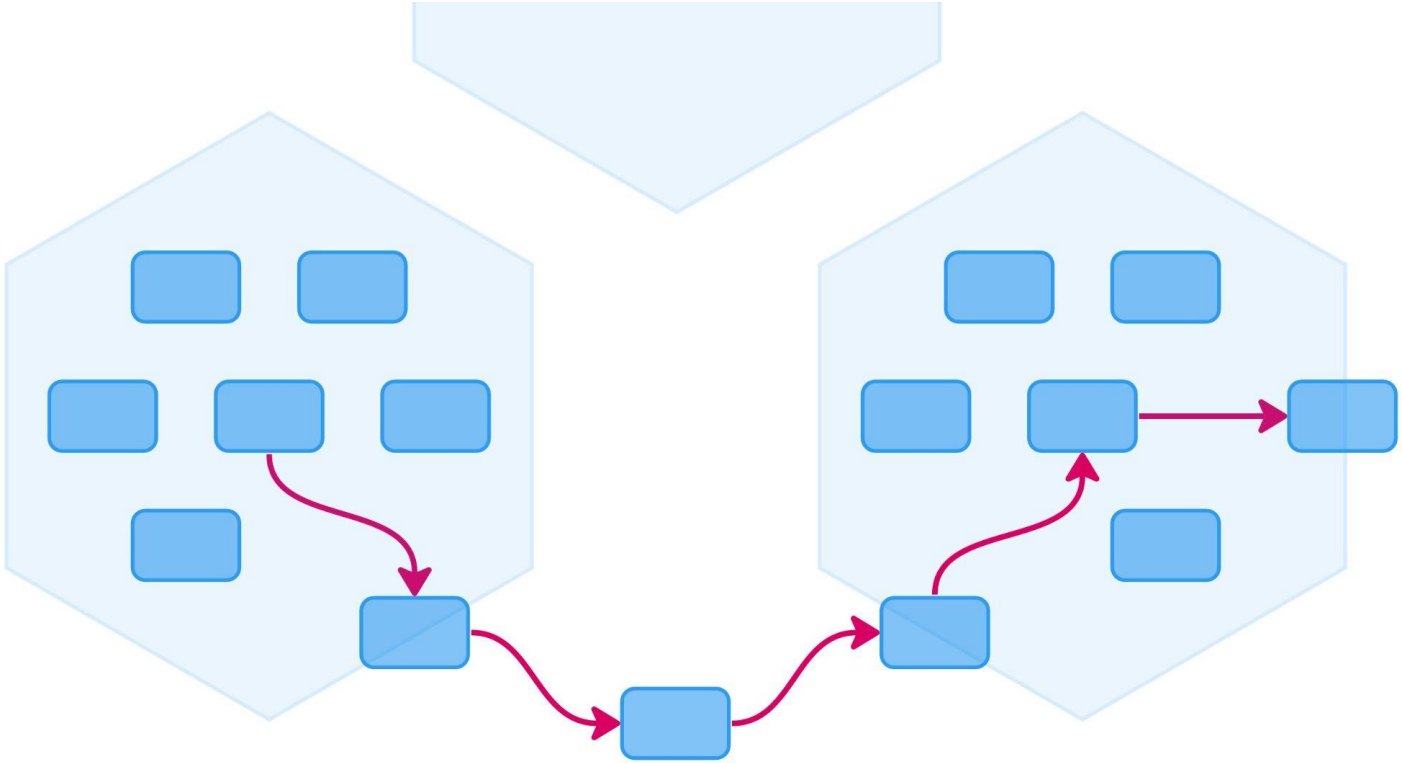
Modules with APIs

Modules with SPIs

Low coupling

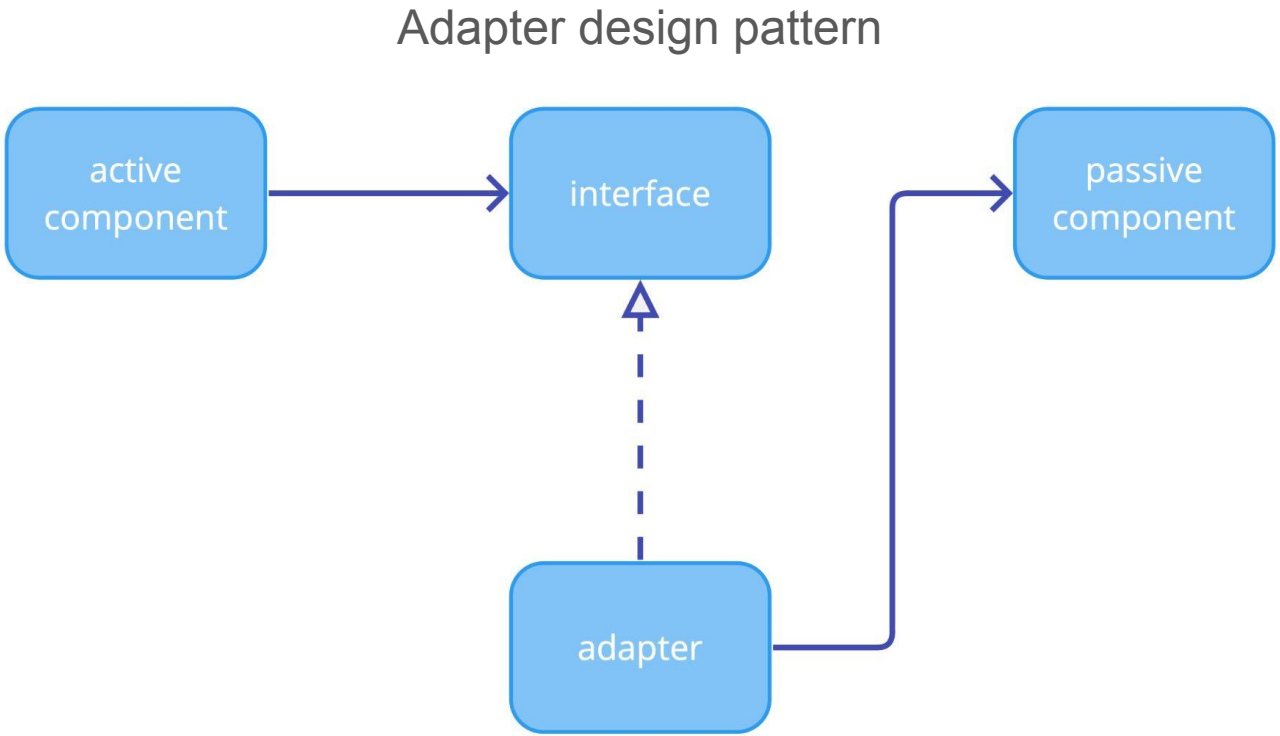


Hexagonal modules

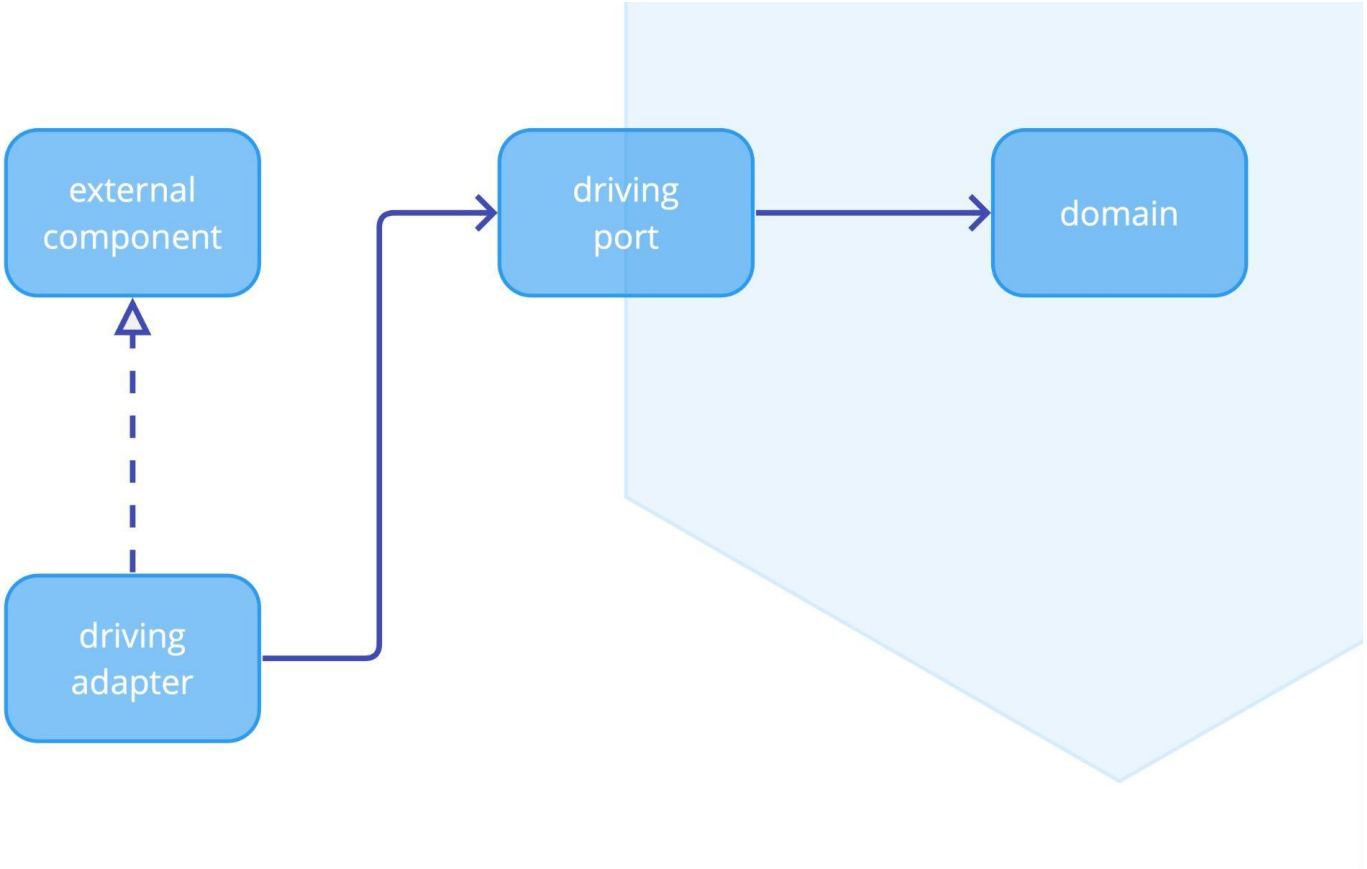


Architecture of many hexagons

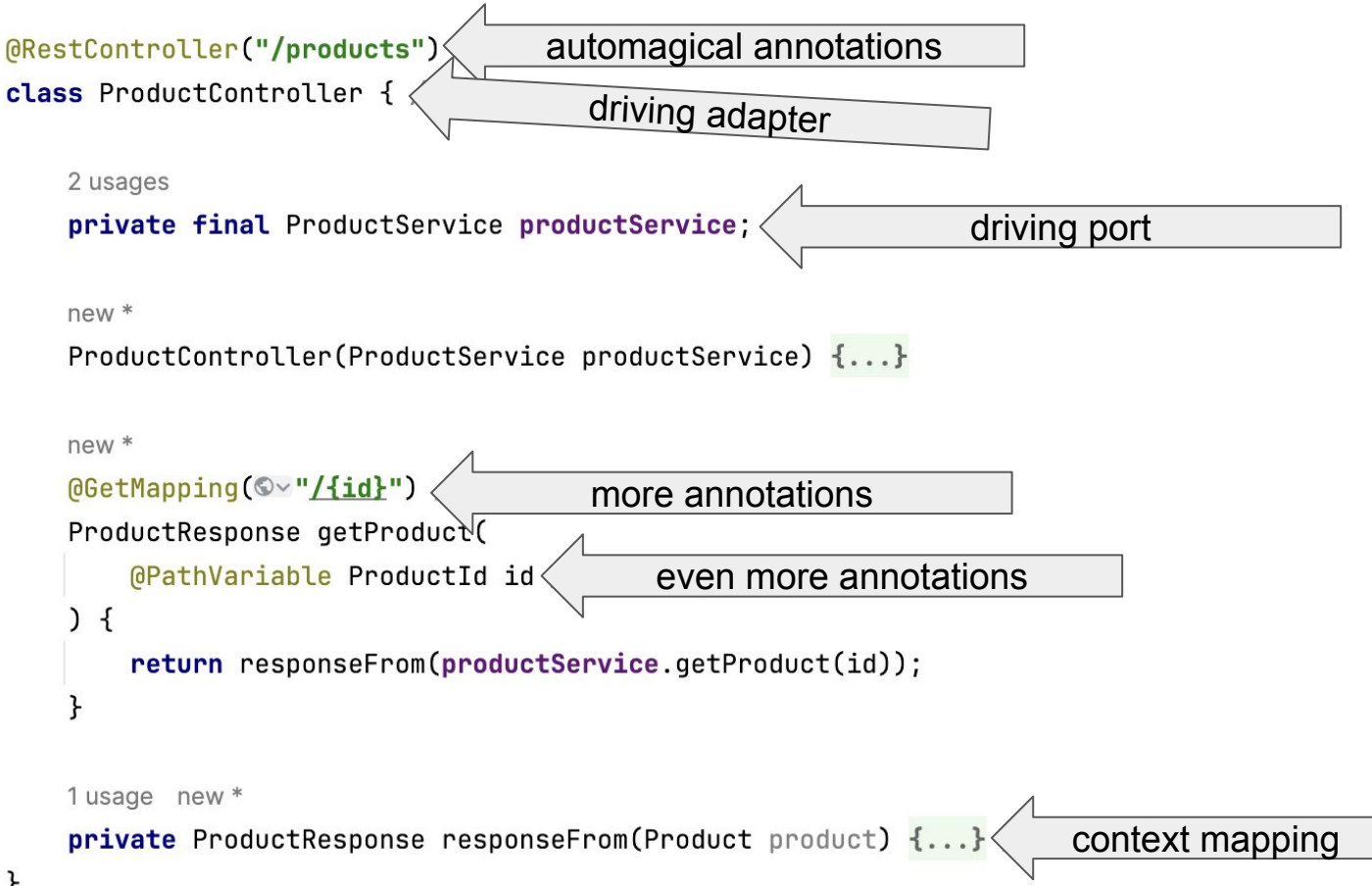
Anatomy of an adapter



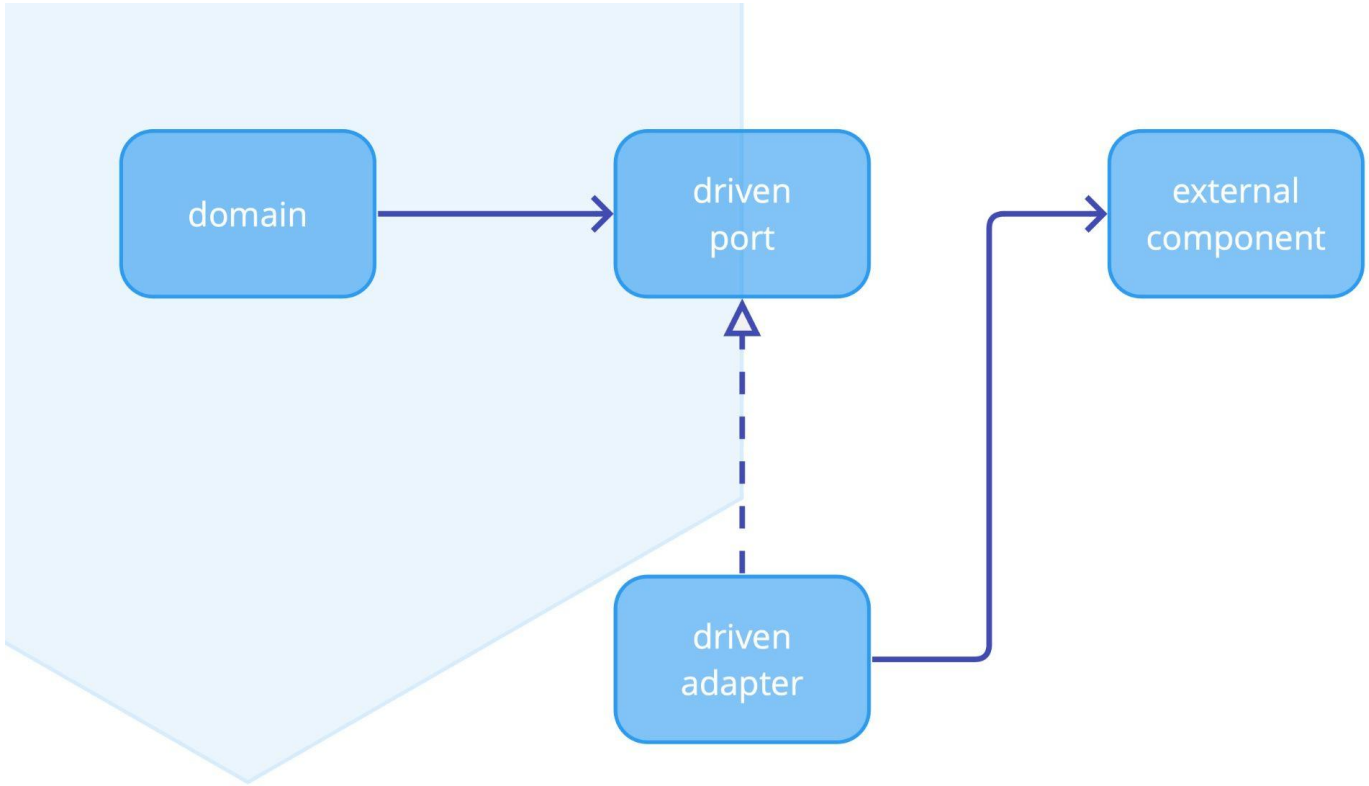
Primary adapter



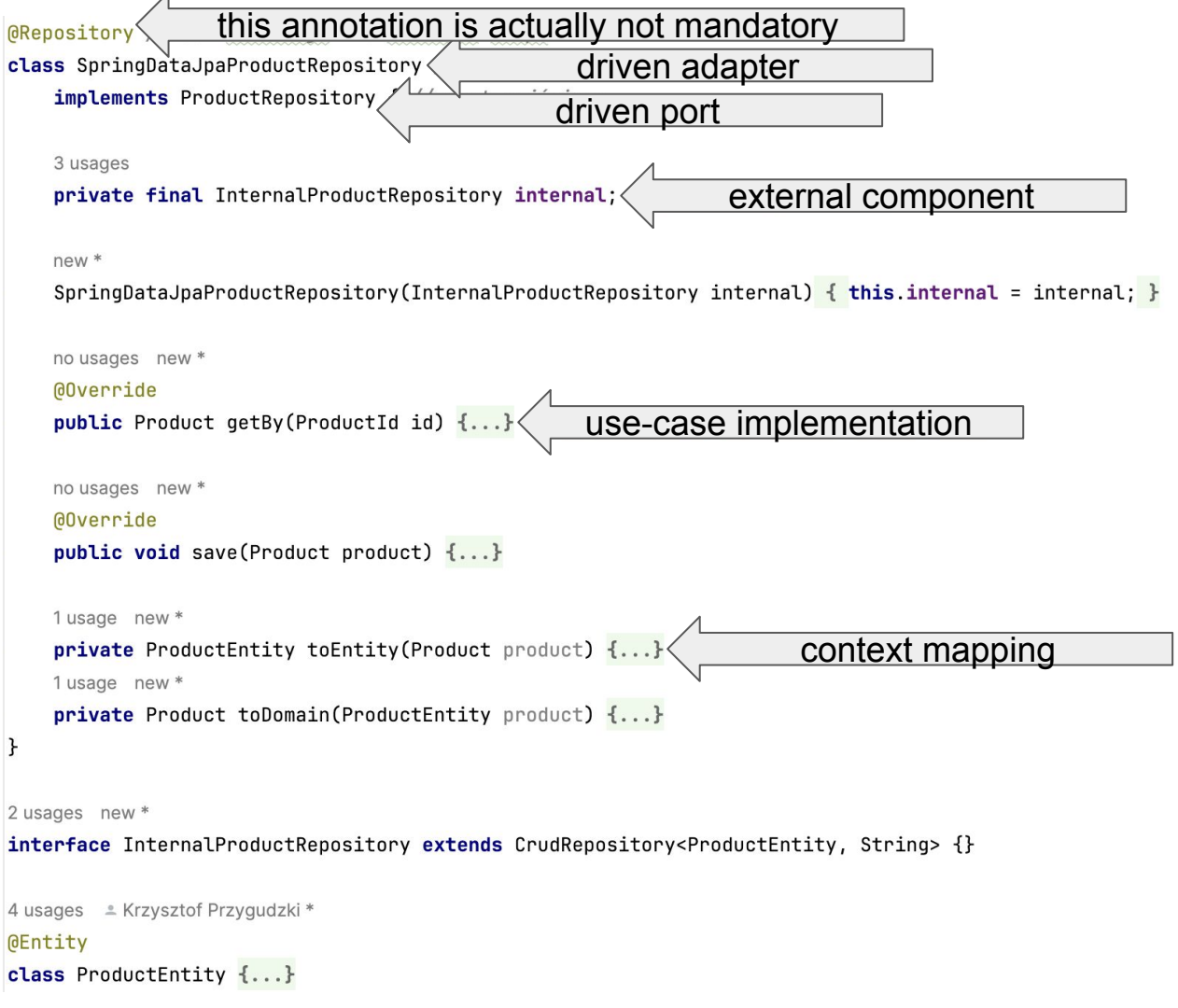
Driving (primary) adapter



Secondary adapter

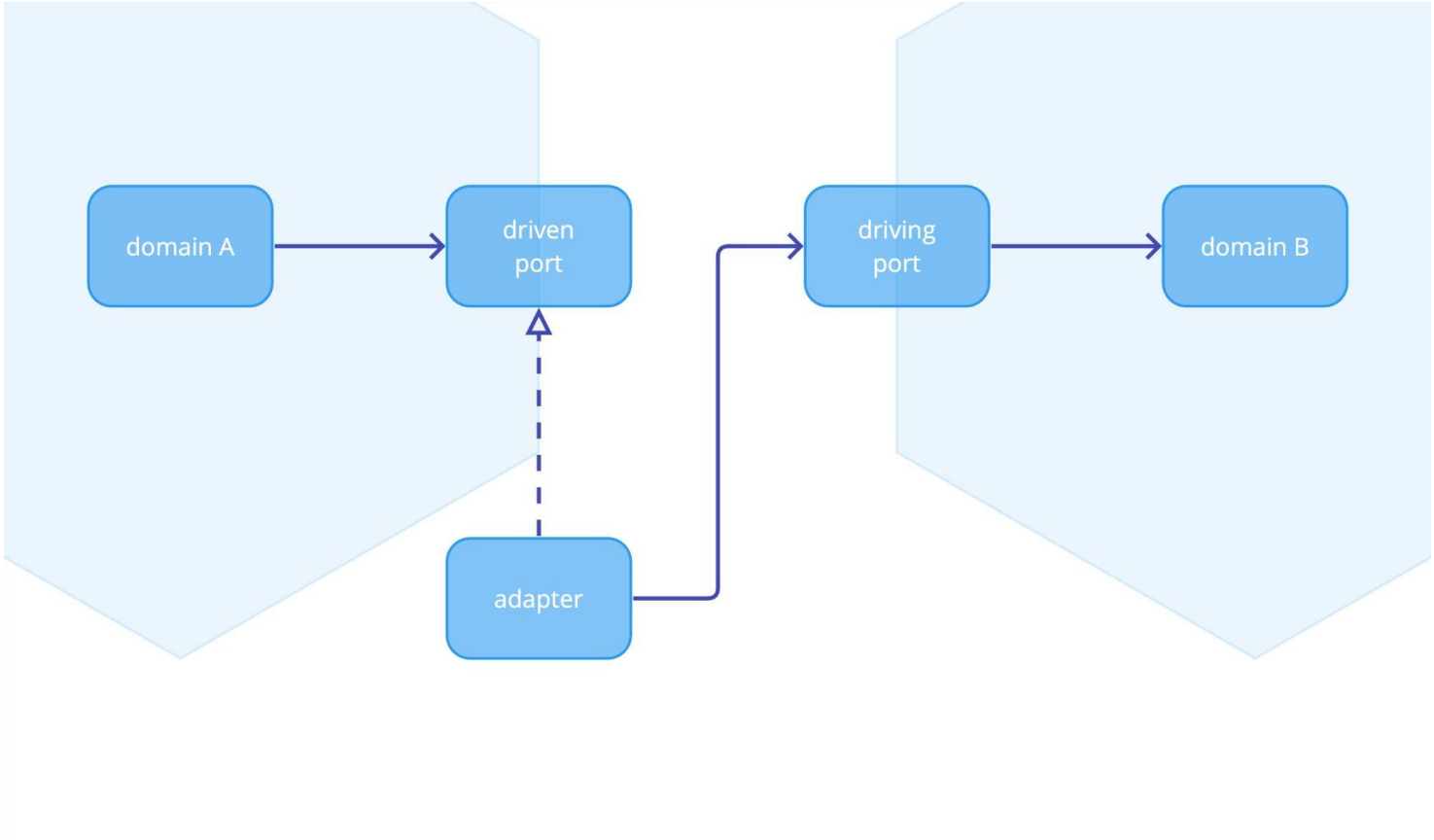


Driven (secondary) adapter




```
@Repository
class SpringDataJpaProductRepository
    implements ProductRepository {

    3 usages
    private final InternalProductRepository internal;

    new *
    SpringDataJpaProductRepository(InternalProductRepository internal) { this.internal = internal; }

    no usages  new *
    @Override
    public Product getBy(ProductId id) {...}

    no usages  new *
    @Override
    public void save(Product product) {...}

    1 usage  new *
    private ProductEntity toEntity(Product product) {...}
    1 usage  new *
    private Product toDomain(ProductEntity product) {...}
}

2 usages  new *
interface InternalProductRepository extends CrudRepository<ProductEntity, String> {}

4 usages  Krzysztof Przygudzki *
@Entity
class ProductEntity {...}
```

Inter-modular adapter



Dependency inversion principle

Communication between components

should happen on the highest mutual level of abstraction

Component of higher level of abstraction

should not depend on the implementation details

of the component of lower level of abstraction

Adapter interchangeability

Limits of hexagons

Can’t always do it clean

Transactions

Our code is an implementation of use-cases

Technical debt

What is architecture really?

Packages structure? Design patterns?

Mental model

Structure of the code directs our way of thinking about it

Different levels of abstraction

High cohesion, low coupling

geecon 2024
Kraków, 15-17 May

Thank you!

geecon

Questions?

Links

https://herbertograca.files.wordpress.com/2018/11/080-explicit-architecture-svg.png