

# SPEAKER: MATT STINE

- Executive Director - Software Engineering  
JPMorgan Chase
- Managed Languages Lead  
Engineers' Channel/Blueprints Product Line Architect
- 22-year veteran Enterprise IT engineer/architect
- Host of Software Architecture Radio  
<http://softwarearchitecturerad.io>
- Author: Migrating to Cloud-Native Application Architectures  
<https://www.oreilly.com/programming/free/migrating-cloud-native-application-architectures.csp>
- Coming Soon: Humility-Driven Development  
<https://leanpub.com/humility-driven-development>



## Humility-Driven Development

*Practicing the Timeless Wisdom of Egoless Programming*



Matt Stine

O'REILLY®

Migrating to  
Cloud-Native  
Application  
Architectures

Compliments of  
Pivotal



Matt Stine

---

WHAT IS THE  
STATE OF MICROSERVICES  
IN 2022?

---



WE BUILT THE **WRONG** MICROSERVICES!



WE BUILT TOO MANY MICROSERVICES!



**WE DIDN'T REALLY NEED MICROSERVICES!**



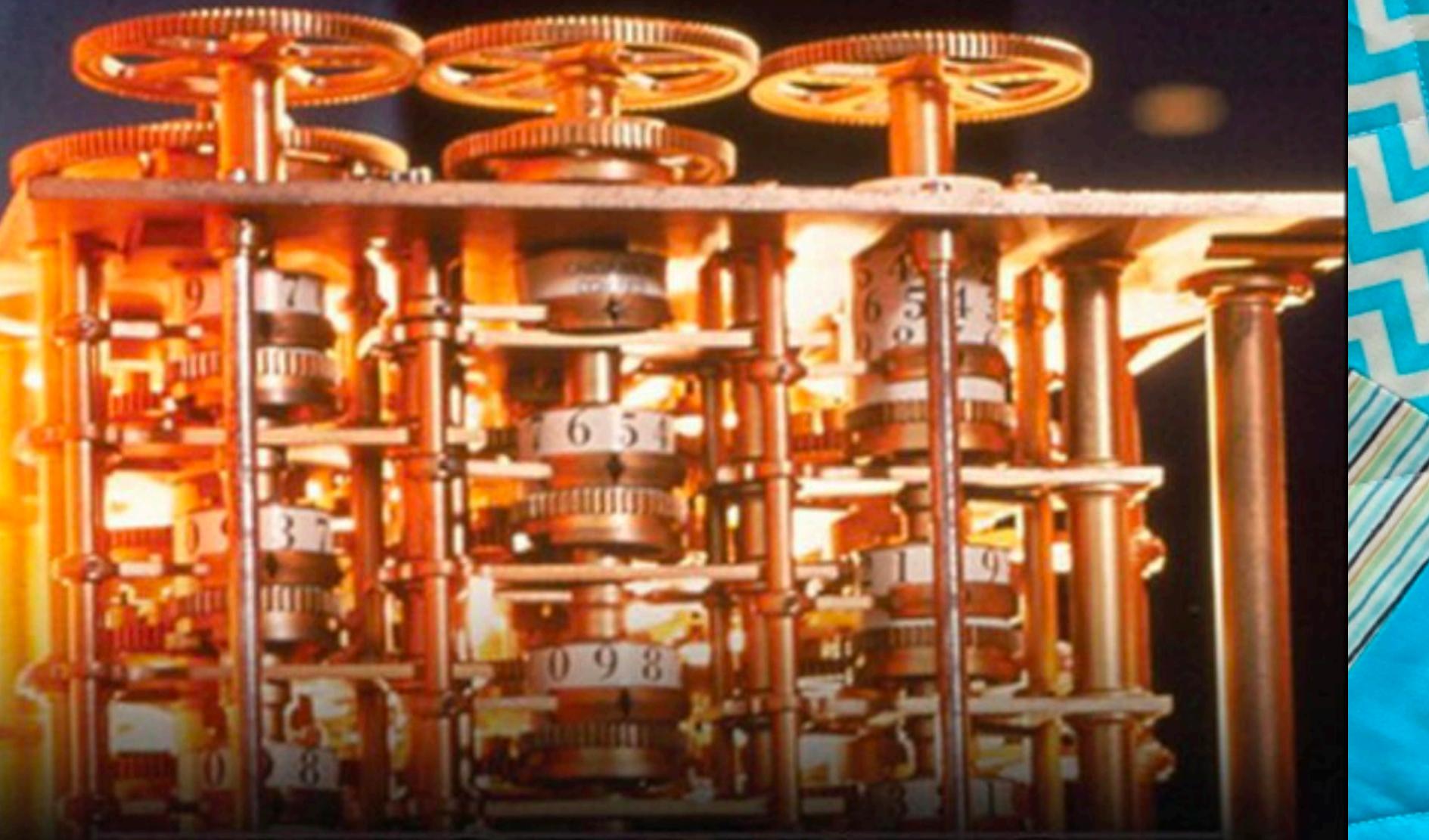
WE CREATED THE SAME OLD ARCHITECTURE WITH  
MICROSERVICES!

WHAT WE'VE GOT  
HERE IS FAILURE TO  
**DESIGN**





Robert C. Martin Series



# WORKING EFFECTIVELY WITH LEGACY CODE

Michael C. Feathers

WHERE ARE THE  
**SEAMS?**



Applause from Pivotal, Mark Carlson, and 125 others



Matt Stine

Cloud Native Polymath and Product Manager @Pivotal. Subtweets are clearly my own.

Jun 2, 2017 · 6 min read

# What's Your Decomposition Strategy?

It's a simple question really, but very few people have an answer.

The most frequent and most important question I get from the developers and architects with whom I work is this:

*| What microservices should we have?*

If you stop and ponder this question for a minute, you'll quickly come to realize that if you get the answer to this question wrong, it doesn't really matter what else you get right.

<https://builttoadapt.io/whats-your-decomposition-strategy-e19b8e72ac8f>

ON THE CRITERIA TO BE USED  
IN DECOMPOSING SYSTEMS INTO MODULES

D. L. Parnas

Department of Computer Science  
Carnegie-Mellon University  
Pittsburgh, Pa.

August, 1971

---

This paper discusses modularization as a mechanism for improving the flexibility and comprehensibility of a system while allowing the shortening of its development time. The effectiveness of a "modularization" is dependent upon the criteria used in dividing the system into modules.

---

---

The major progress in the area of modular programming has been the development of coding techniques and assemblers which (1) allow one module to be written with little knowledge of the code used in another module and, (2) allow modules to be reassembled and replaced without reassembly of the whole system.

---

Decomposition Techniques

---

**LET'S BUILD AN ONLINE STORE!**

**Product  
Catalog**

**Orders**

**Inventory**

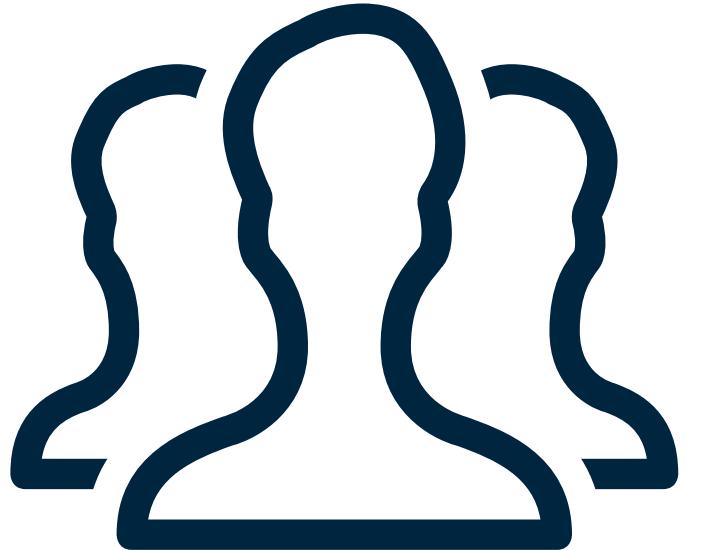
**Payment**

**Notifications**

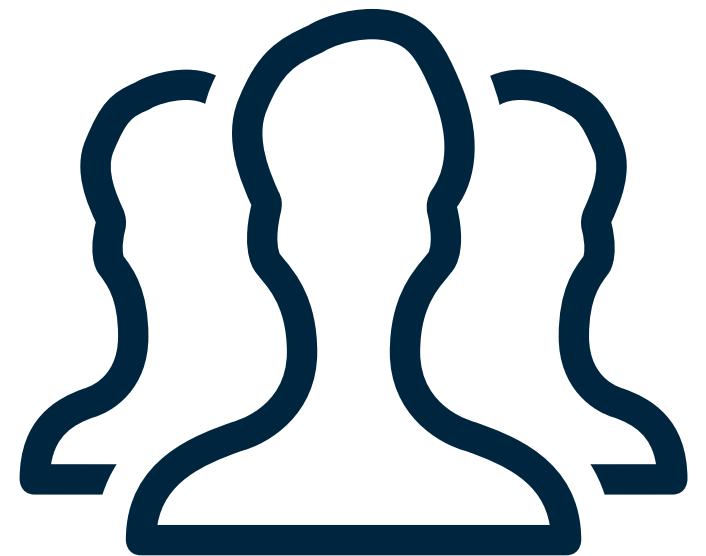
**Fulfillment**

**Shipping**





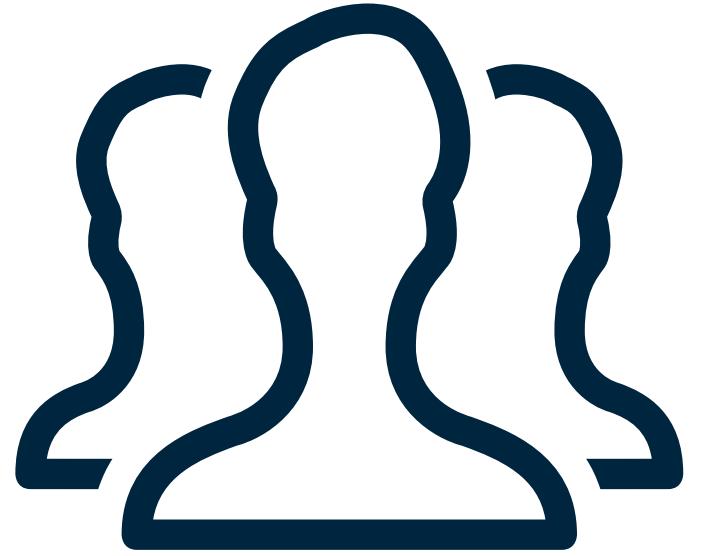
UI



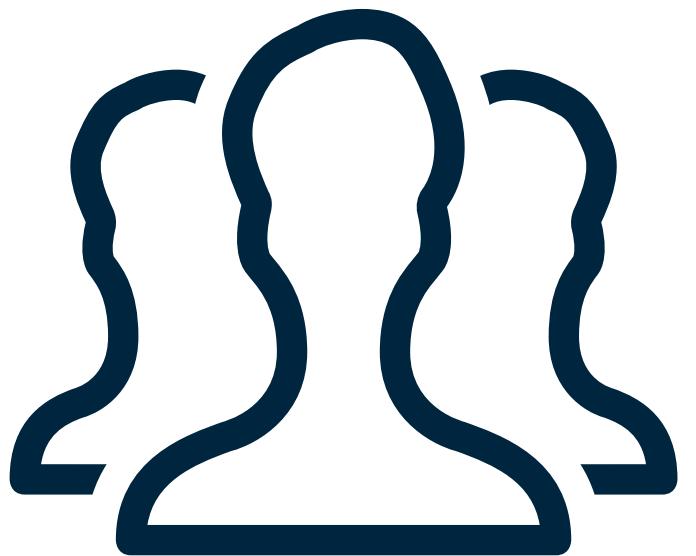
Services



Data



UI

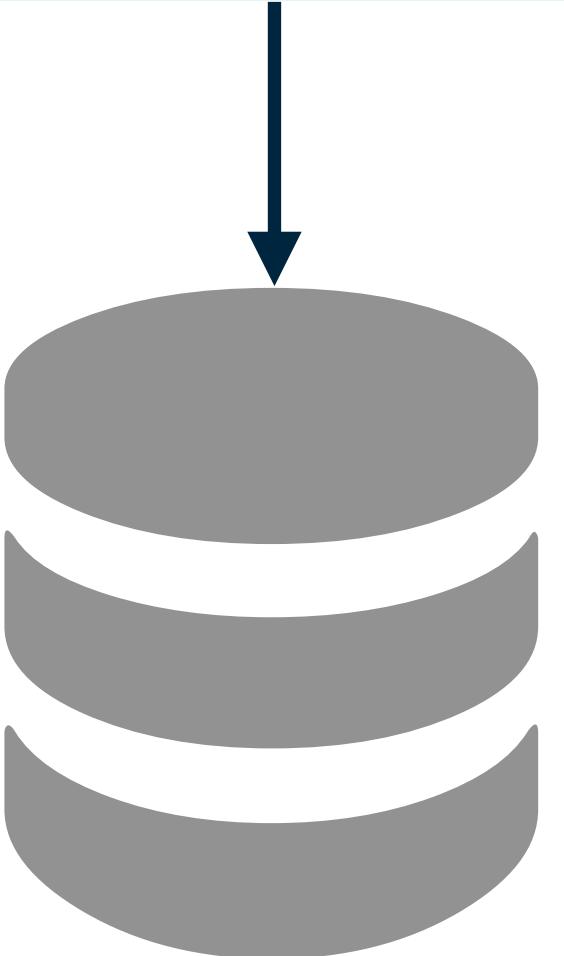


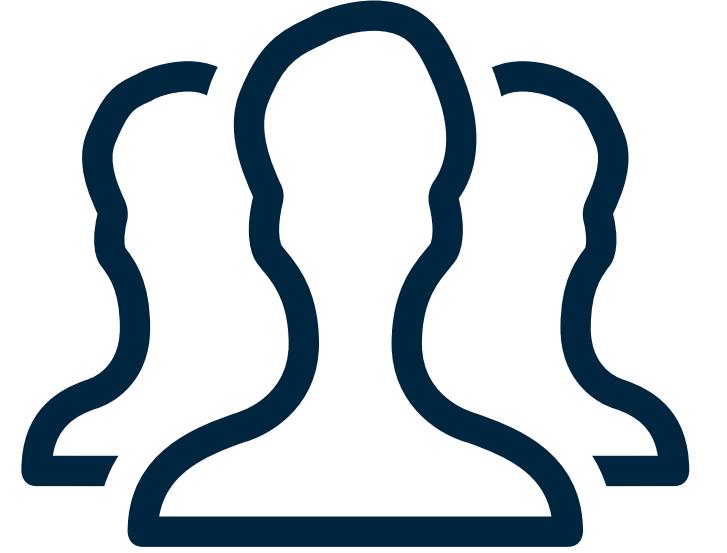
Services



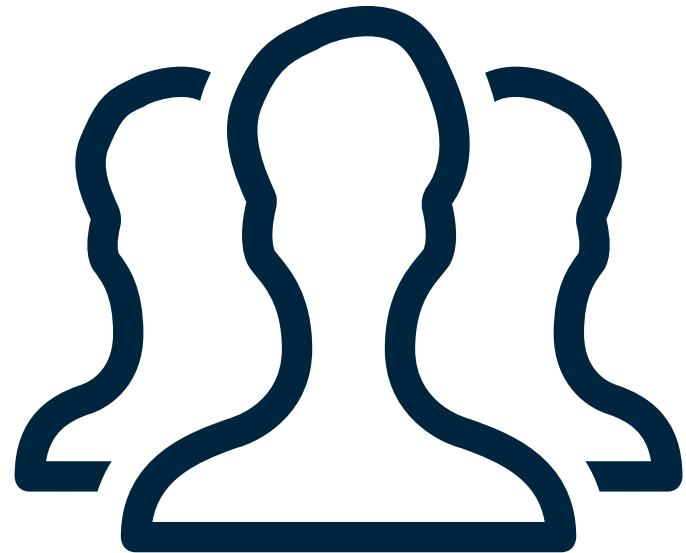
Data

Domain Model





UI

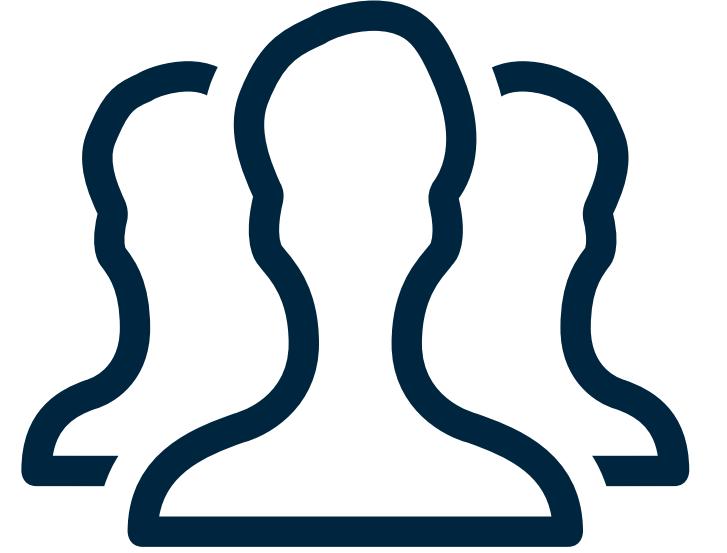


Services



Data





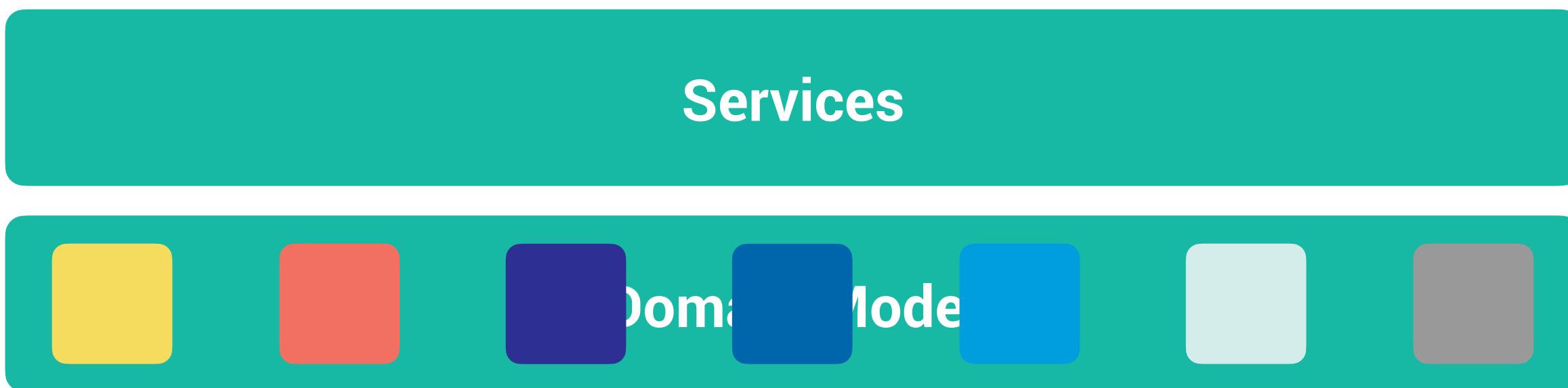
UI

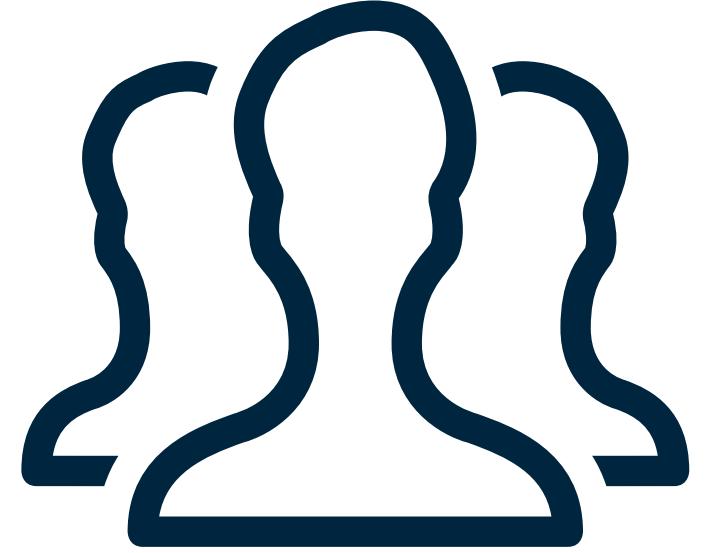


Services

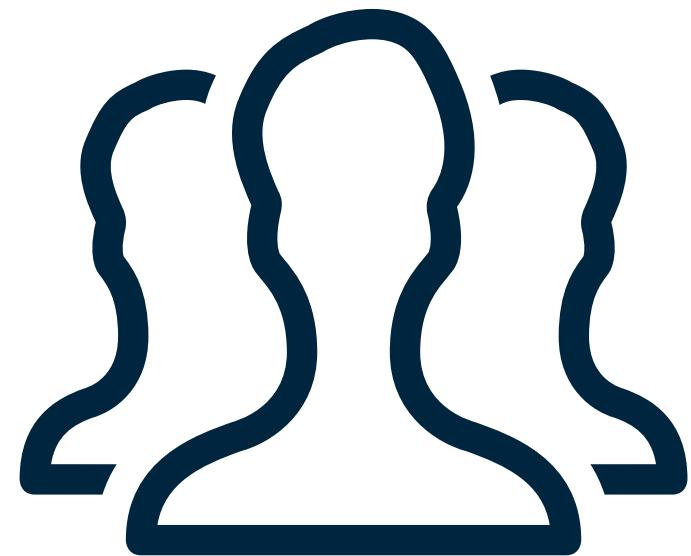


Data





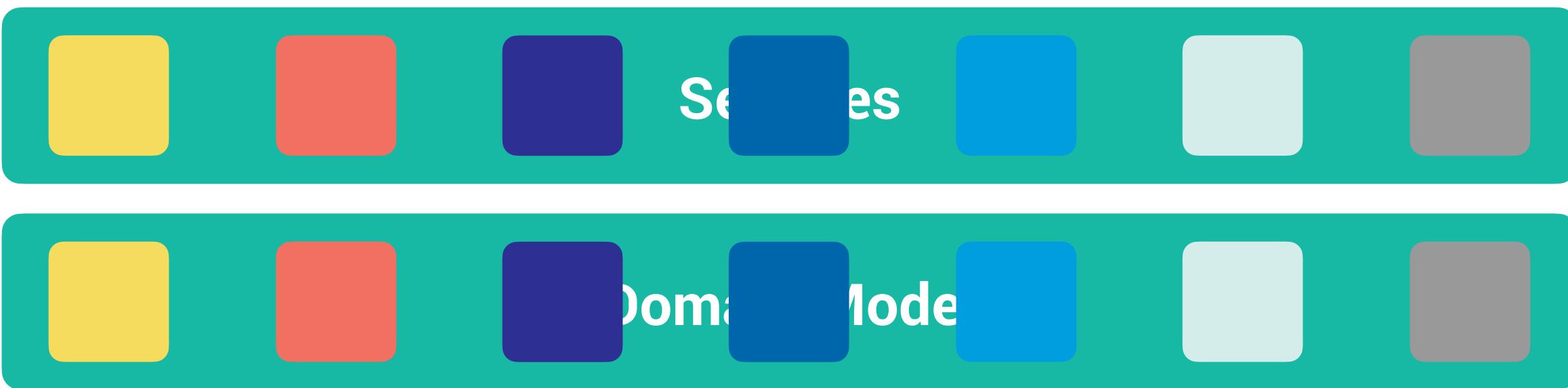
UI

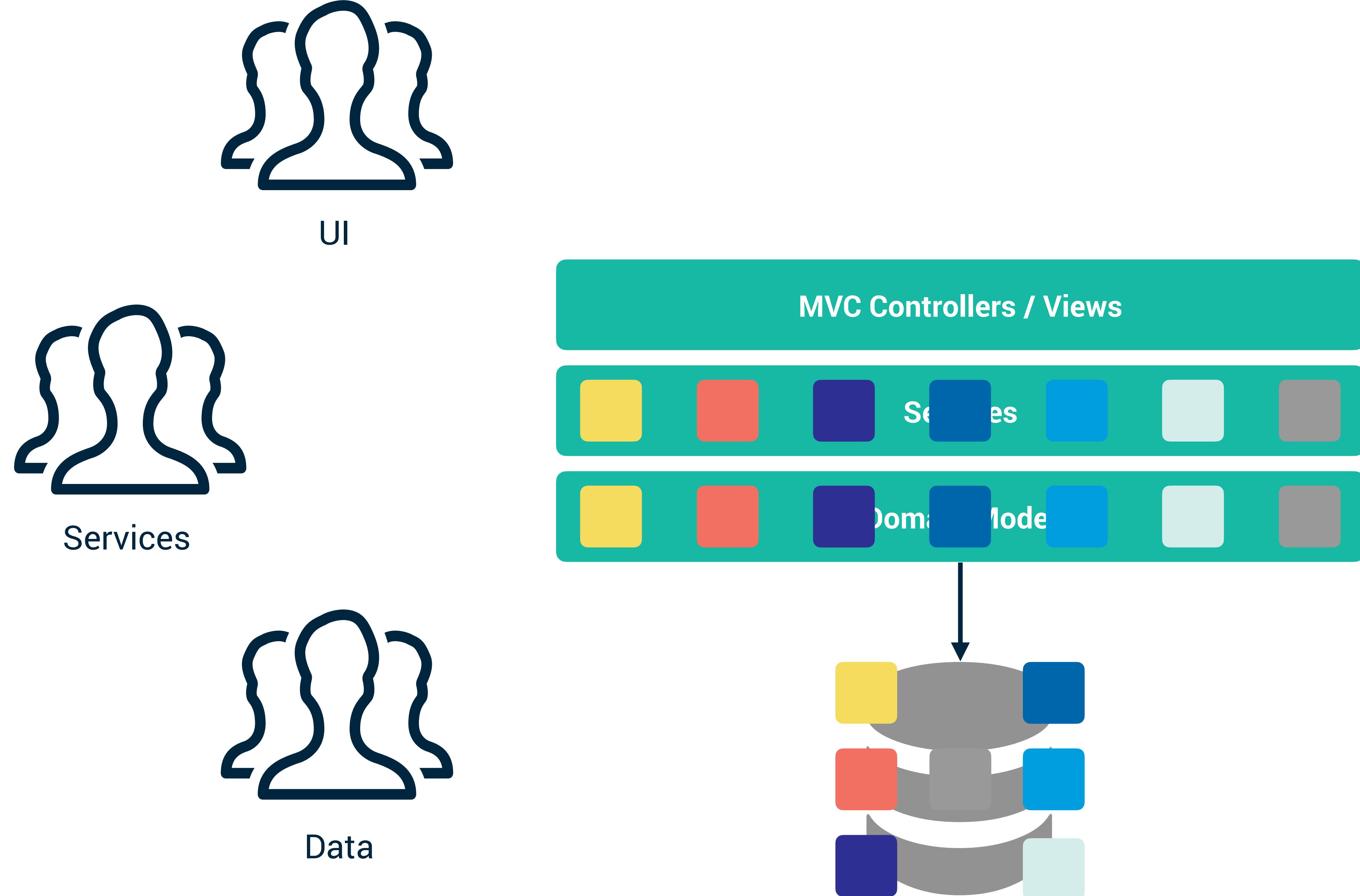


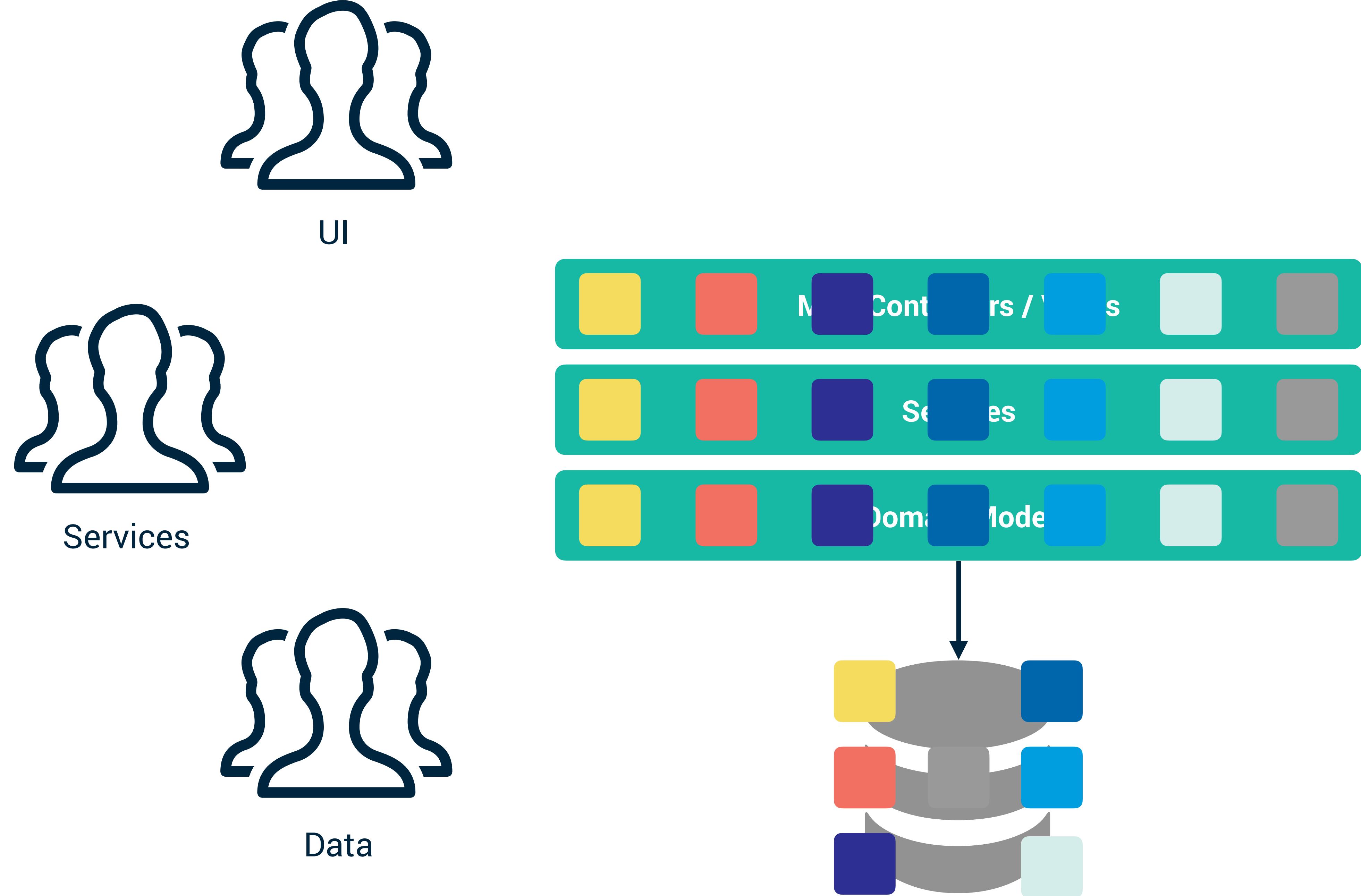
Services

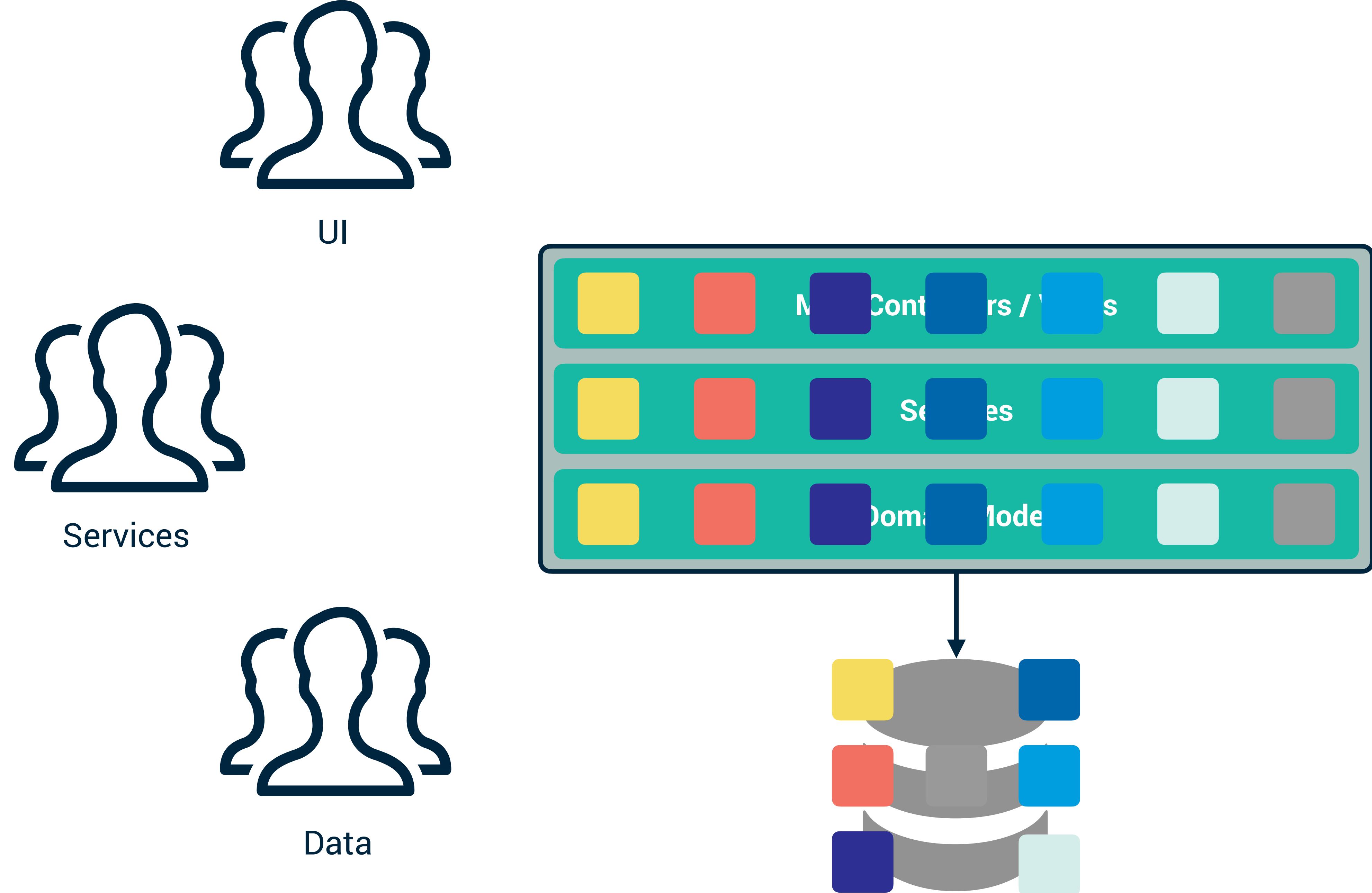


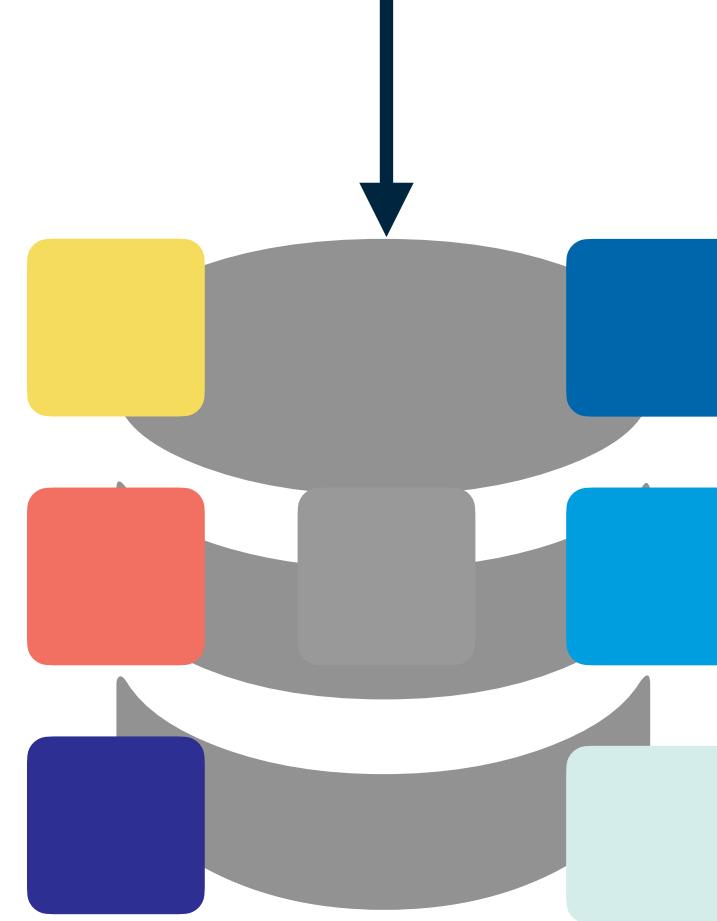
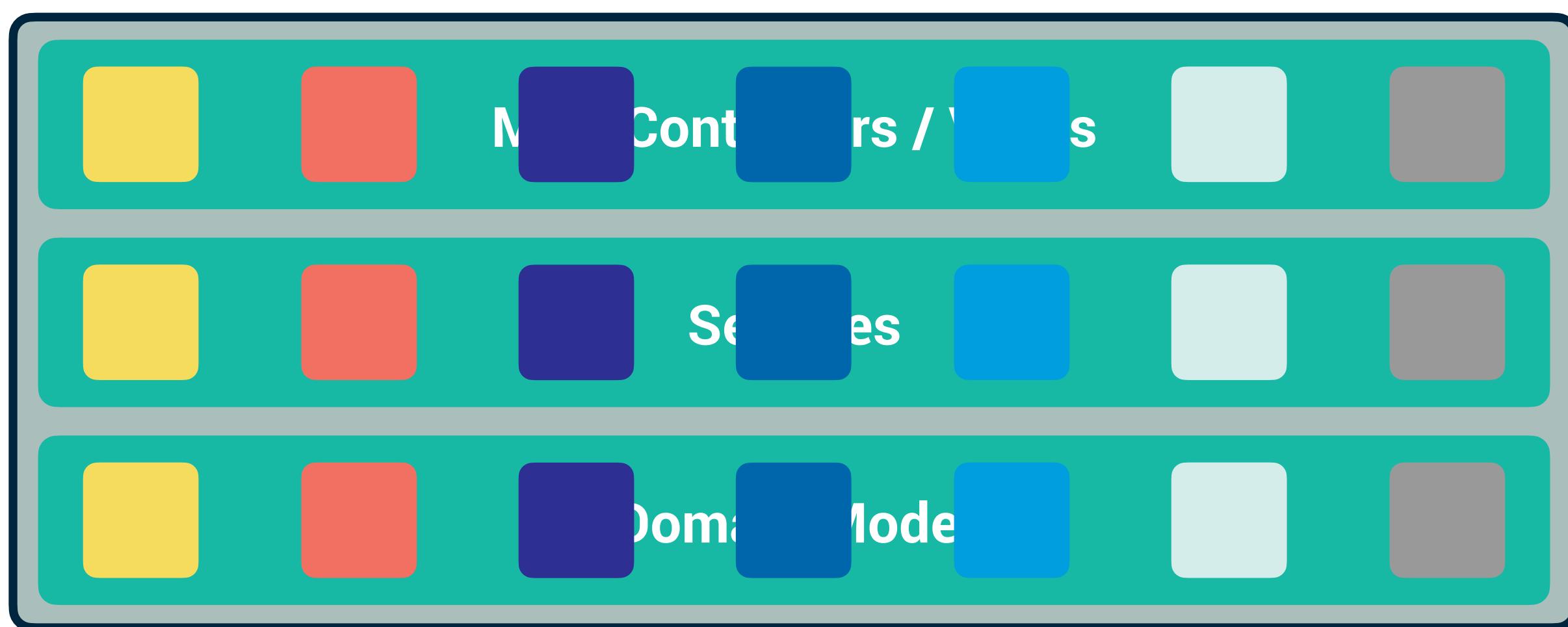
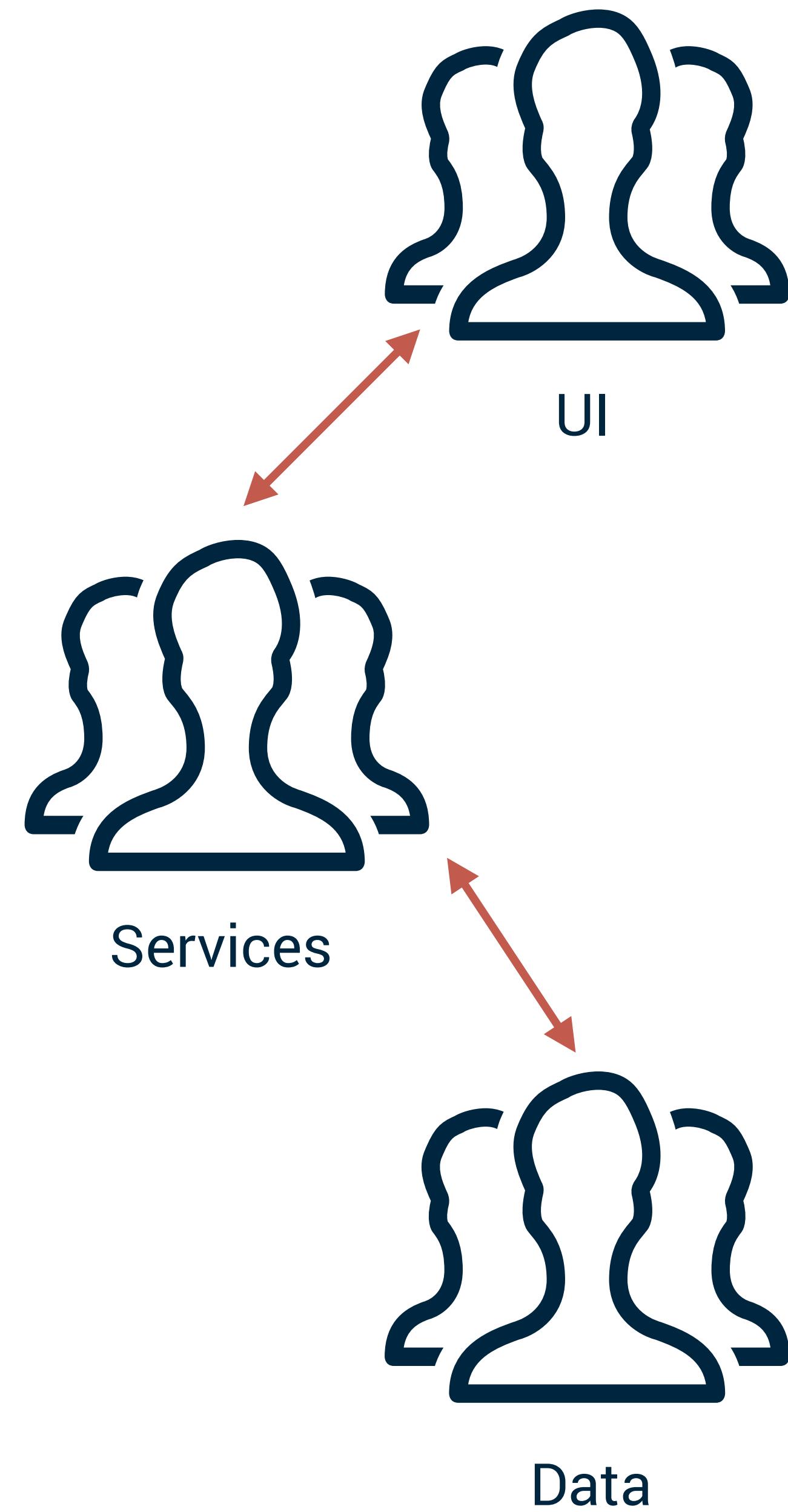
Data











# FUNCTIONAL DECOMPOSITION

## Changes

Struggles with knowledge spread across all modules, so often each module has to change in response to a desired functional change.

## Independent Development

Dependent on shared data formats and schema. Must be jointly defined and agreed upon across multiple groups.

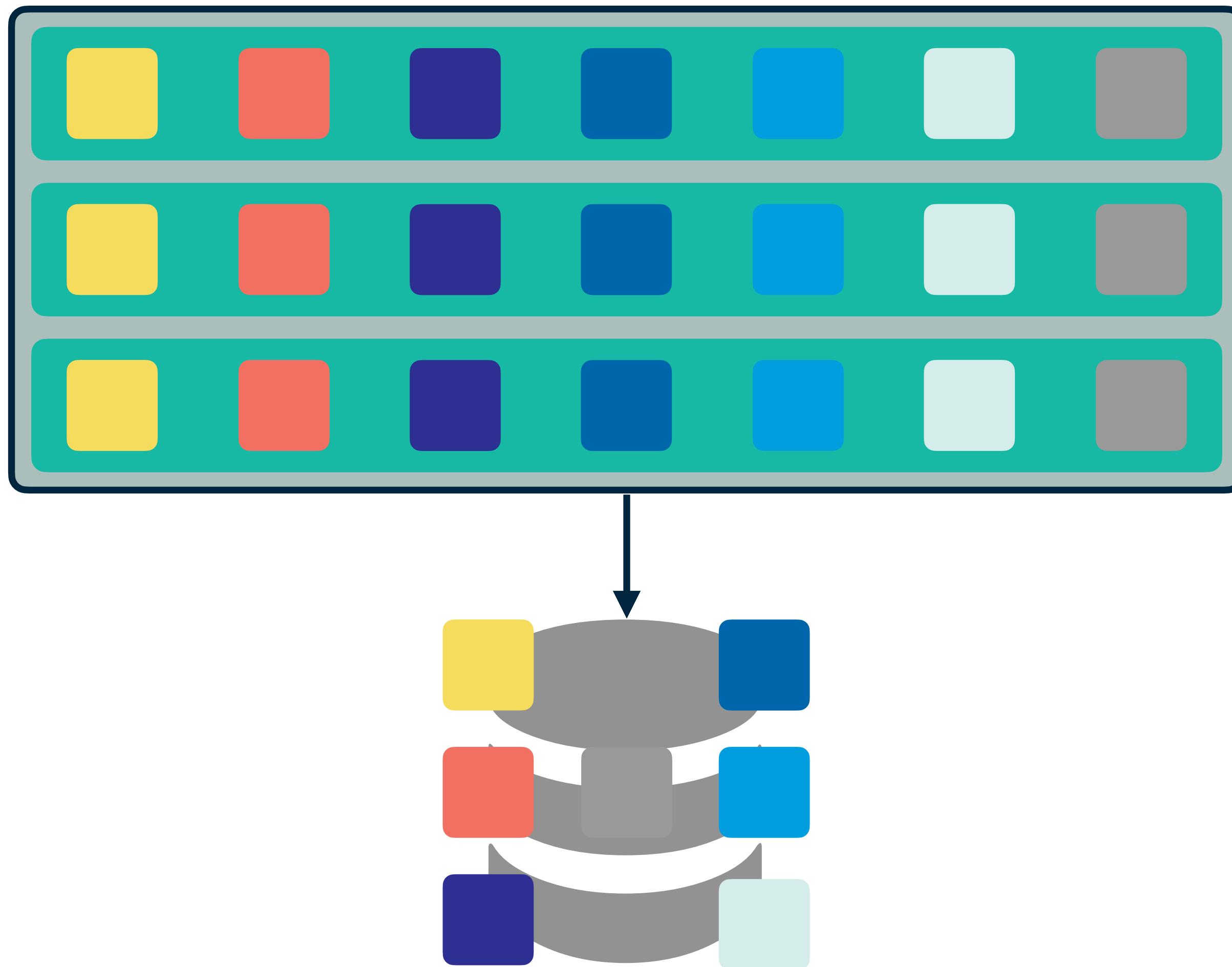
## Comprehensibility

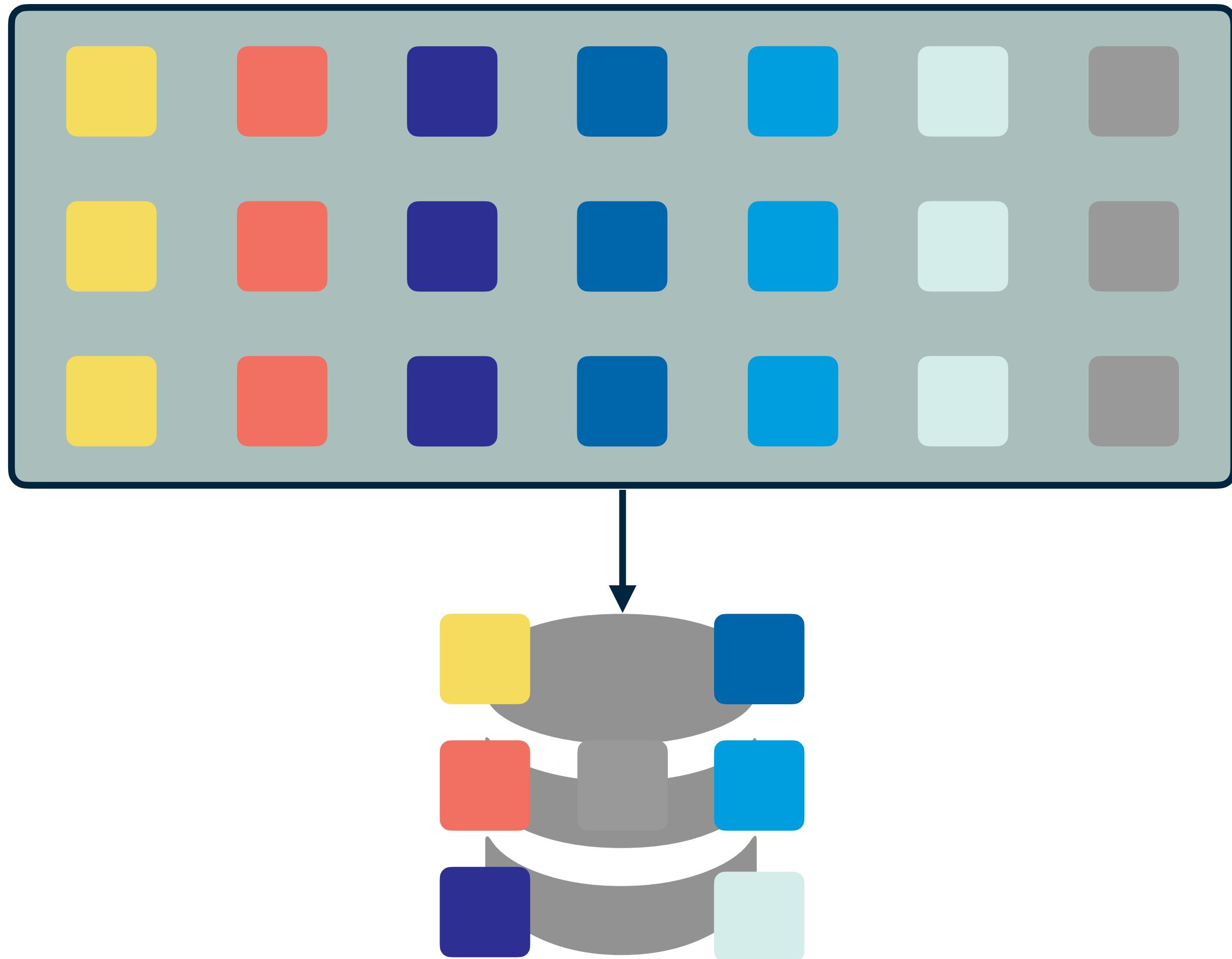
You need to know something about how all of the modules work to understand the whole system.

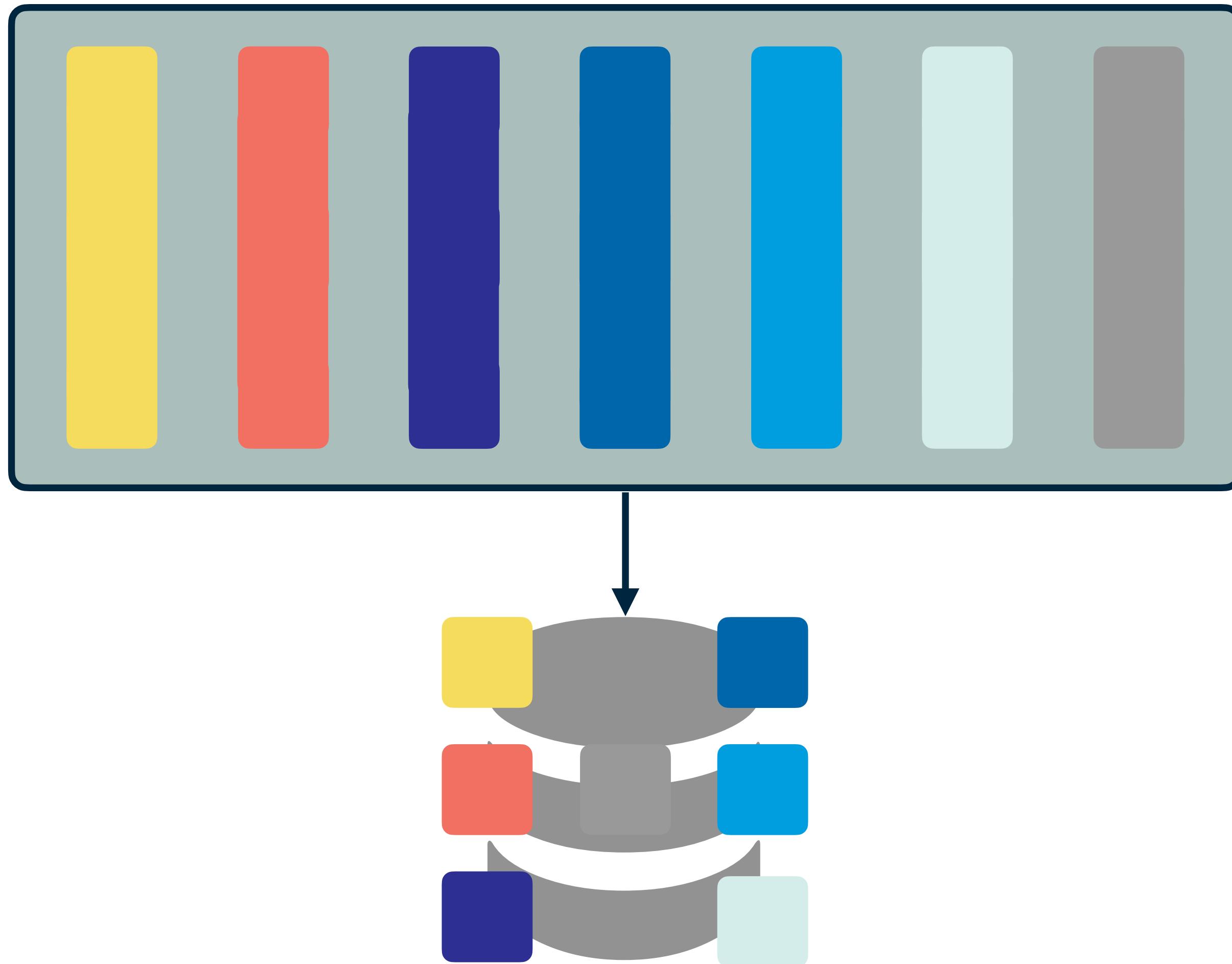
Decomposition Techniques

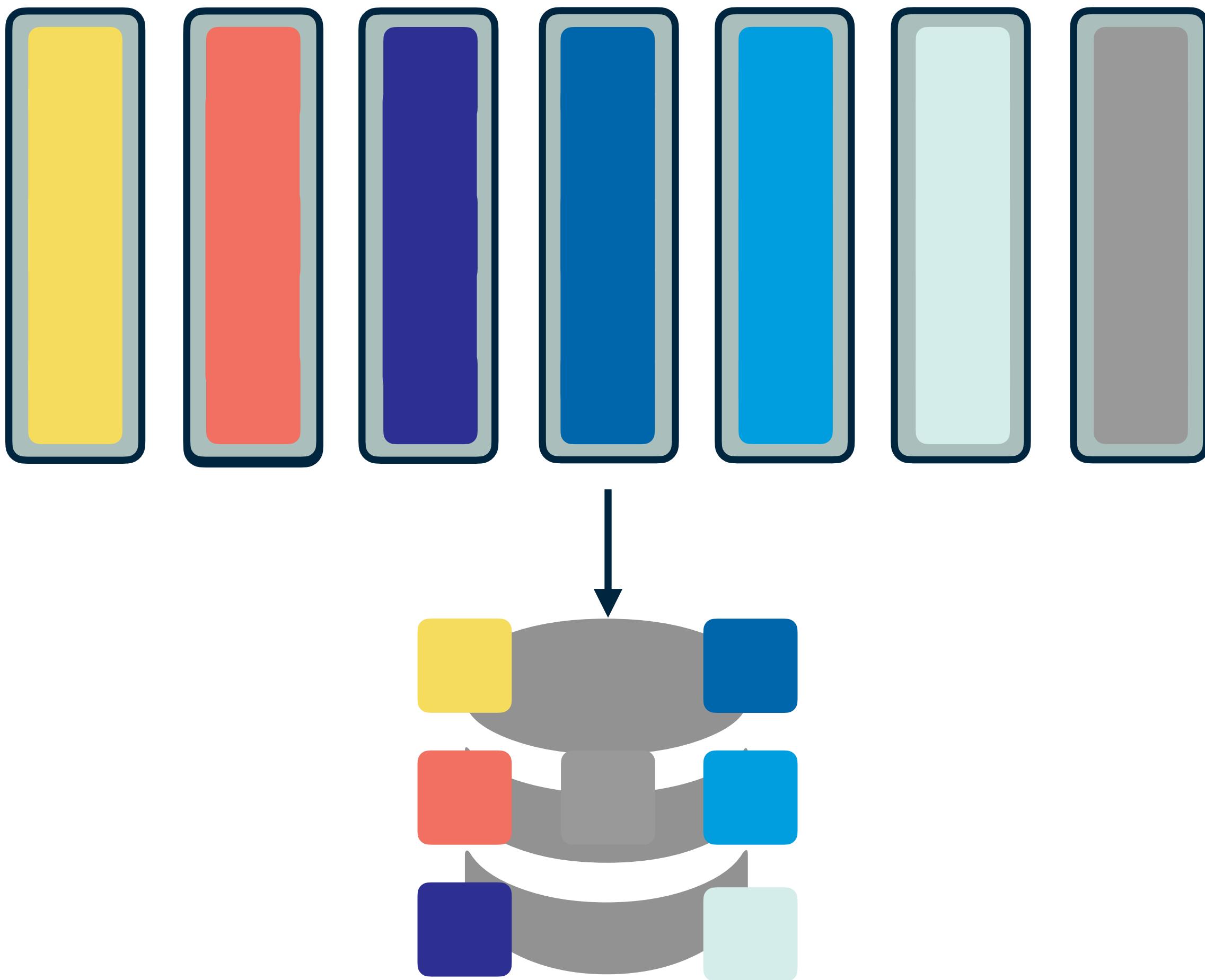
---

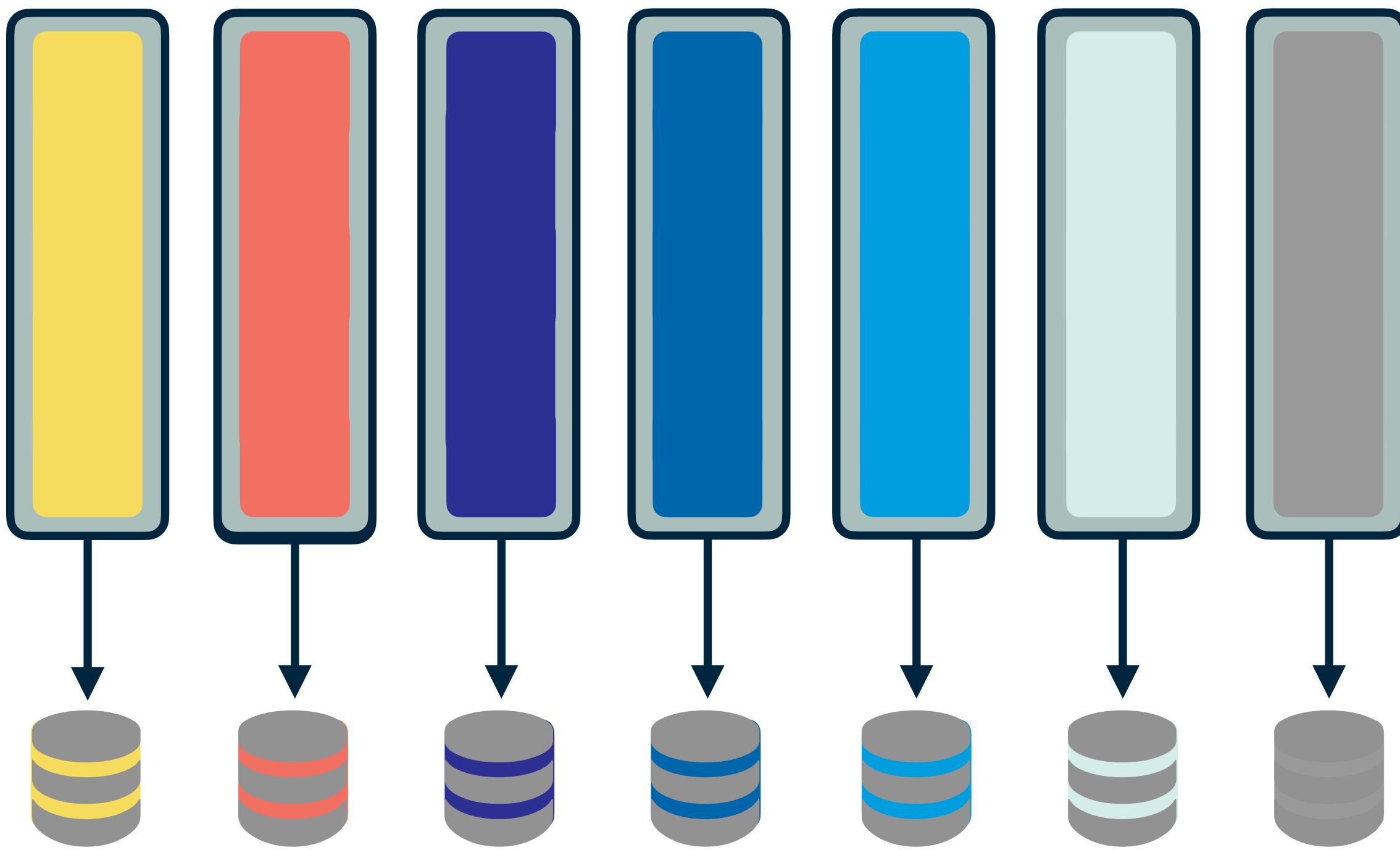
LET'S REFACTOR TO A  
CAPABILITY DECOMPOSITION

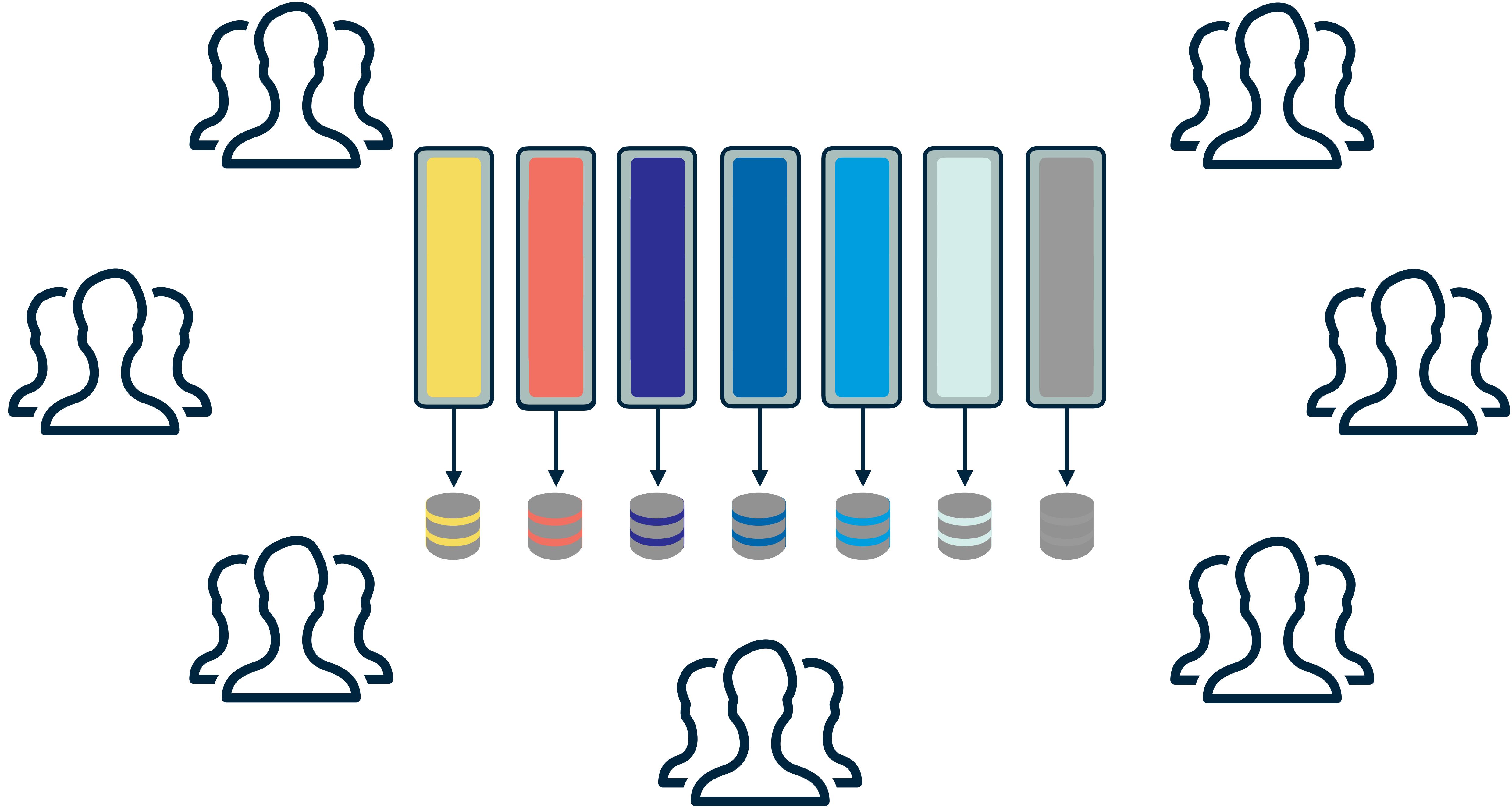


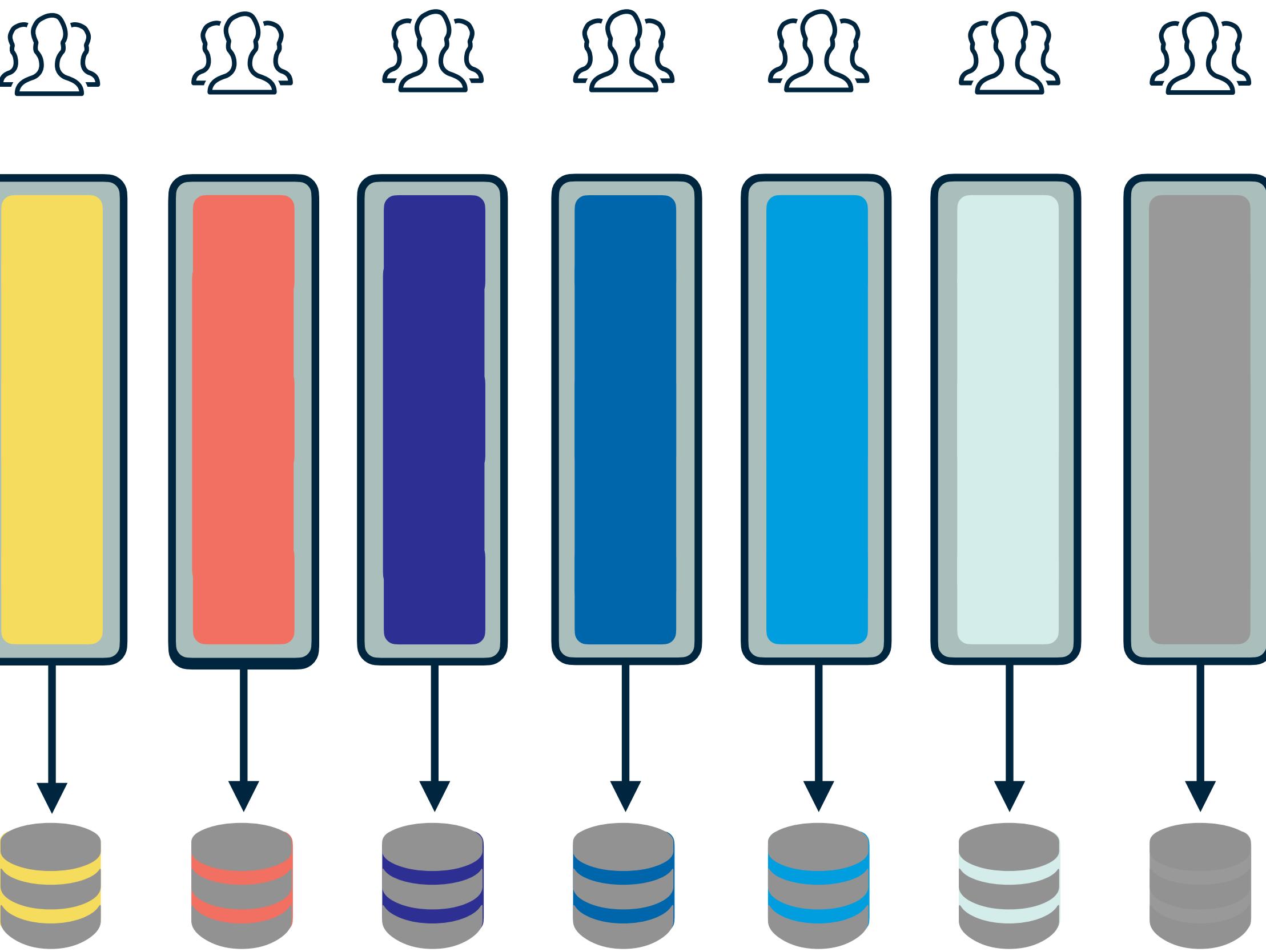












# CAPABILITY DECOMPOSITION

## Changes

Usually isolates a change to a single module.

## Independent Development

Has abstract interfaces that encapsulate the work to be done..

## Comprehensibility

You can understand modules independently.

---

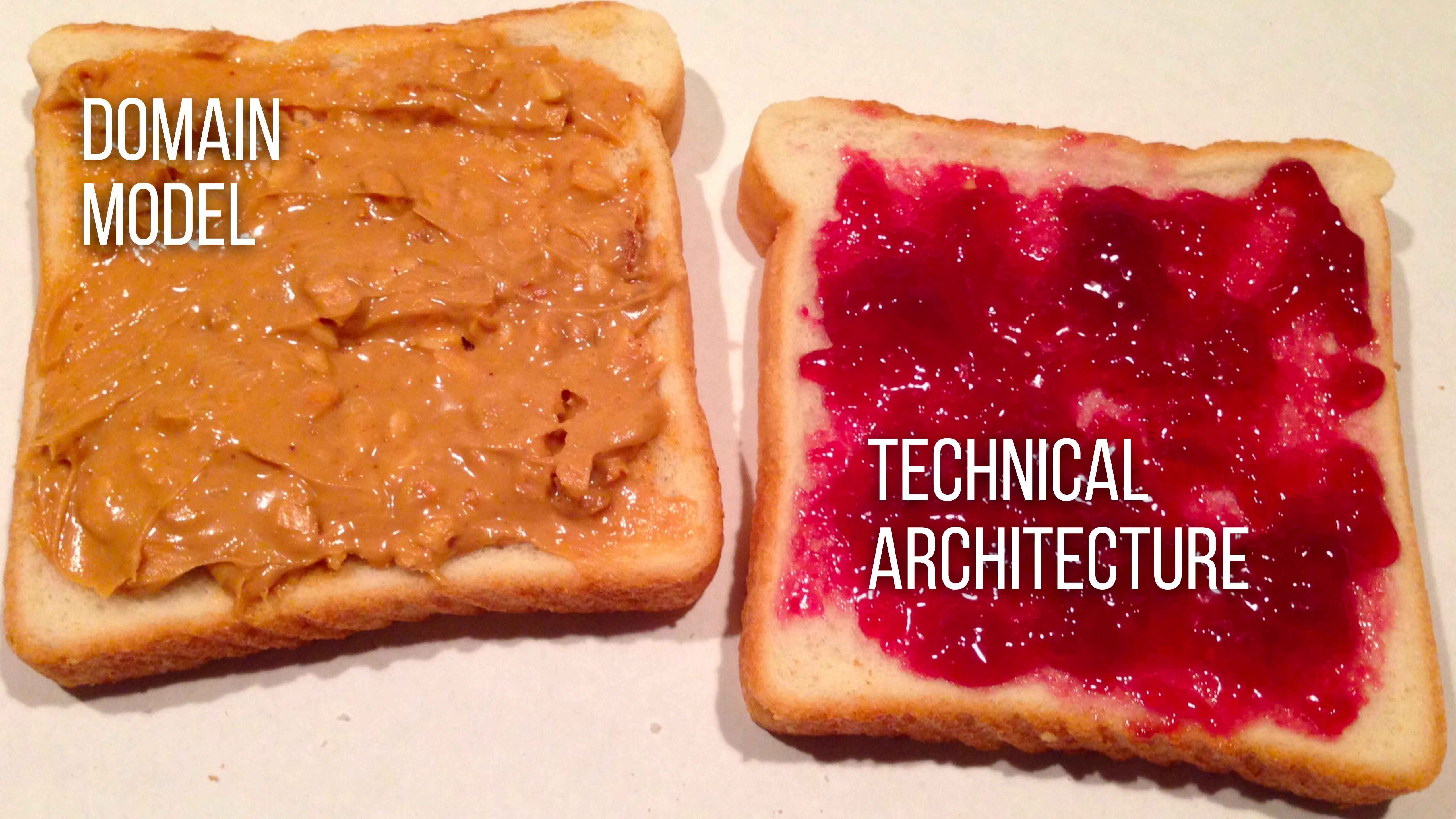
We have tried to demonstrate...it is almost always incorrect to begin the decomposition of a system into modules on the basis of a flowchart...instead that one begins with a list of difficult design decisions or design decisions which are likely to change. Each module is then designed to hide such a decision from the others. Since, in most cases, design decisions transcend time of execution, modules will not correspond to steps in the processing...we must abandon the assumption that a module is one or more subroutines, and instead allow subroutines and programs to be assembled collections of code from various modules.

---

Decomposition Techniques

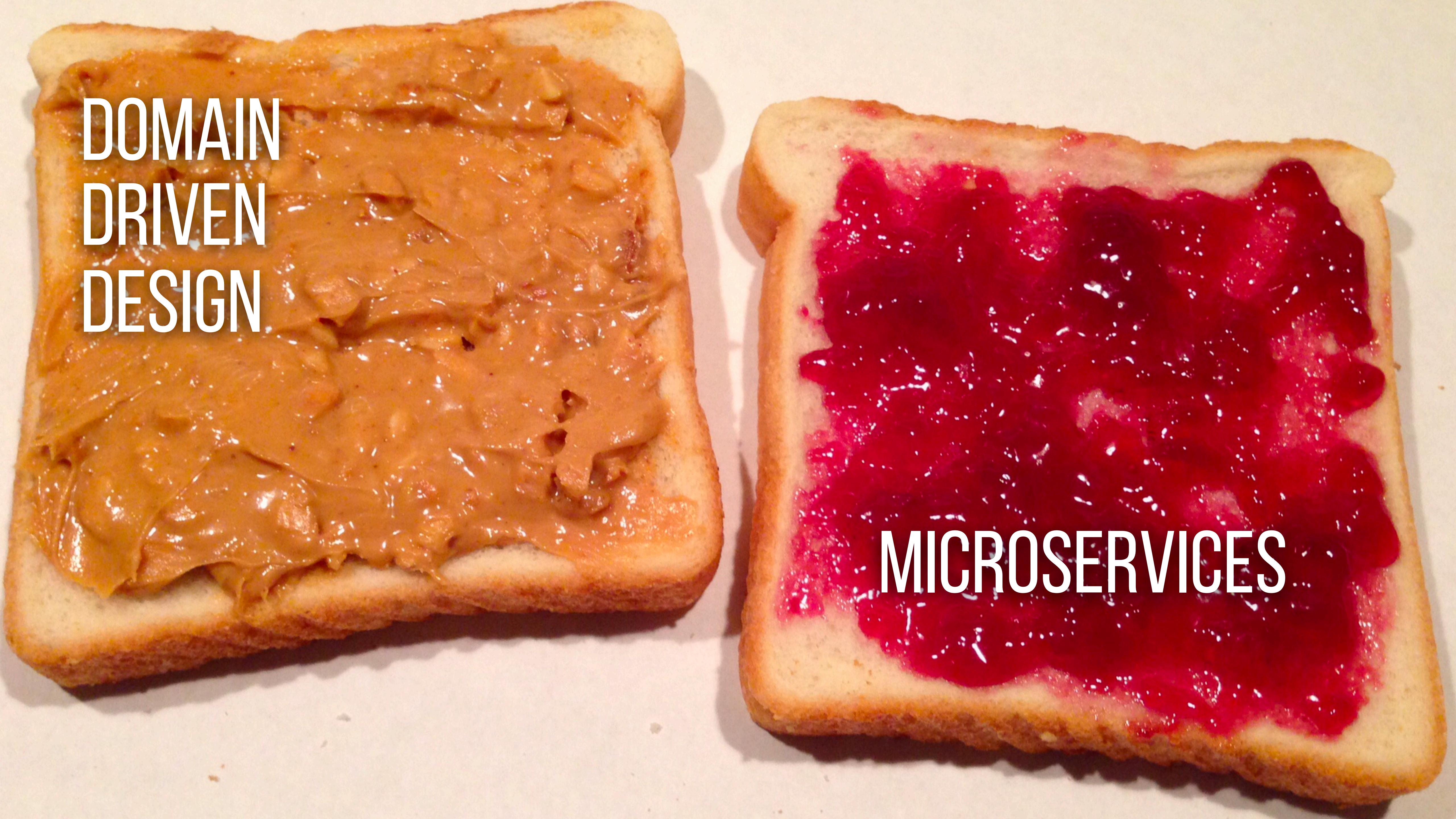
---

THE FUNDAMENTAL WAYS IN WHICH WE DESIGN  
MODULES EFFECTIVELY HAS NOT CHANGED IN THE  
LAST 50 YEARS.

A photograph of two slices of white bread. The left slice is spread with a thick layer of peanut butter, while the right slice is spread with a thick layer of red jam. Both spreads are uneven and runny, dripping down the sides of the bread. The bread is set against a plain, light-colored background.

DOMAIN  
MODEL

TECHNICAL  
ARCHITECTURE

A photograph of two slices of white bread. The slice on the left is spread with a thick layer of peanut butter, which has been partially smeared onto the adjacent slice. The slice on the right is spread with a thick layer of red jam. Both slices are resting on a light-colored surface.

DOMAIN  
DRIVEN  
DESIGN

MICROSERVICES

# THE UBIQUITOUS LANGUAGE

---

When discussing the **domain**, and I use a **term**,  
and you use **exactly the same** term, we mean  
**precisely the same** thing.

---



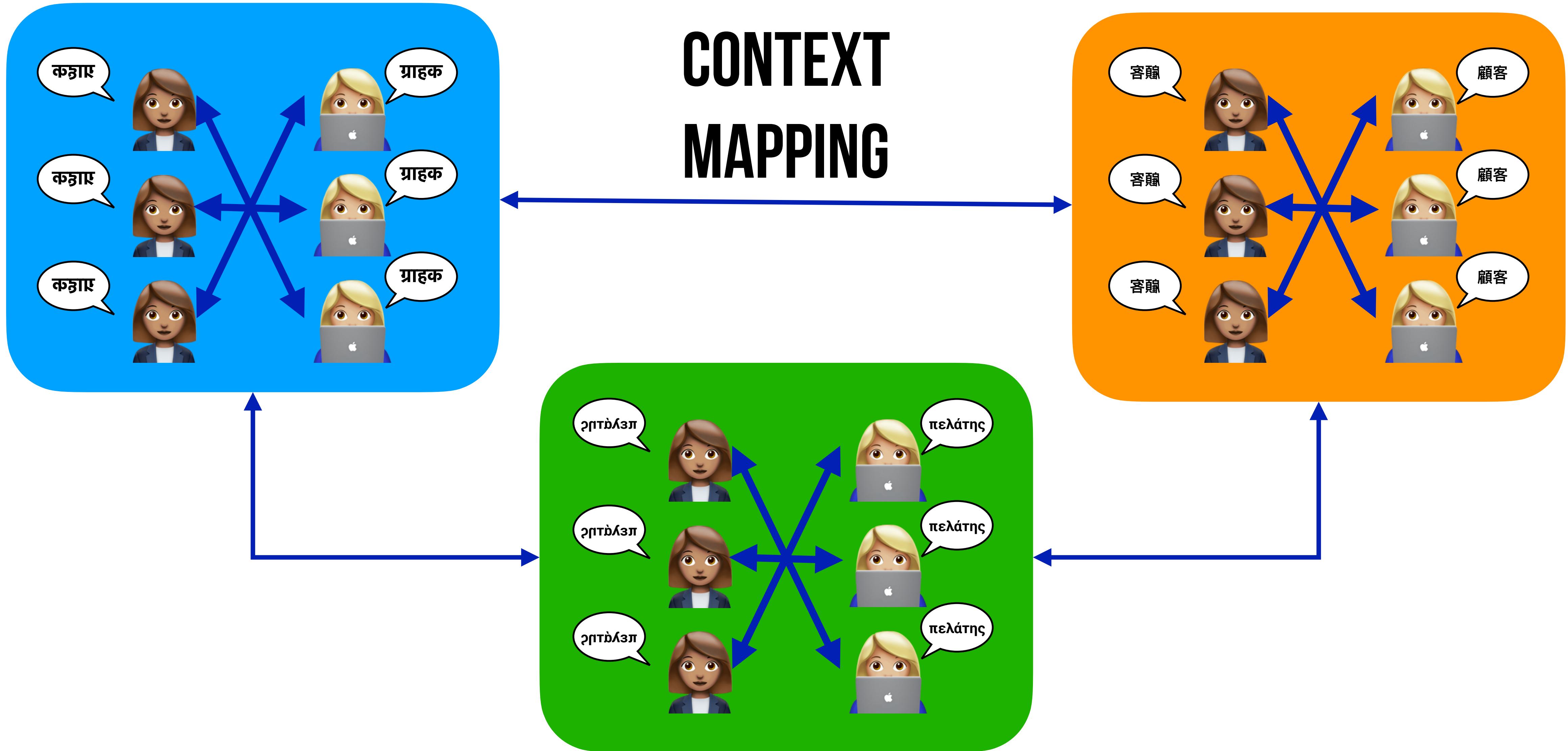
**STRATEGIC DESIGN - THE VIEW FROM 35,000 FEET**



# BOUNDED CONTEXTS

FINDING THE

# INTEGRATING WITH CONTEXT MAPPING



# TACTICAL DESIGN?

Now it's time to start speaking the  
Ubiquitous Language within each Bounded Context.



- Business Invariants
- Policies
- Transactions
- State
- Persistence

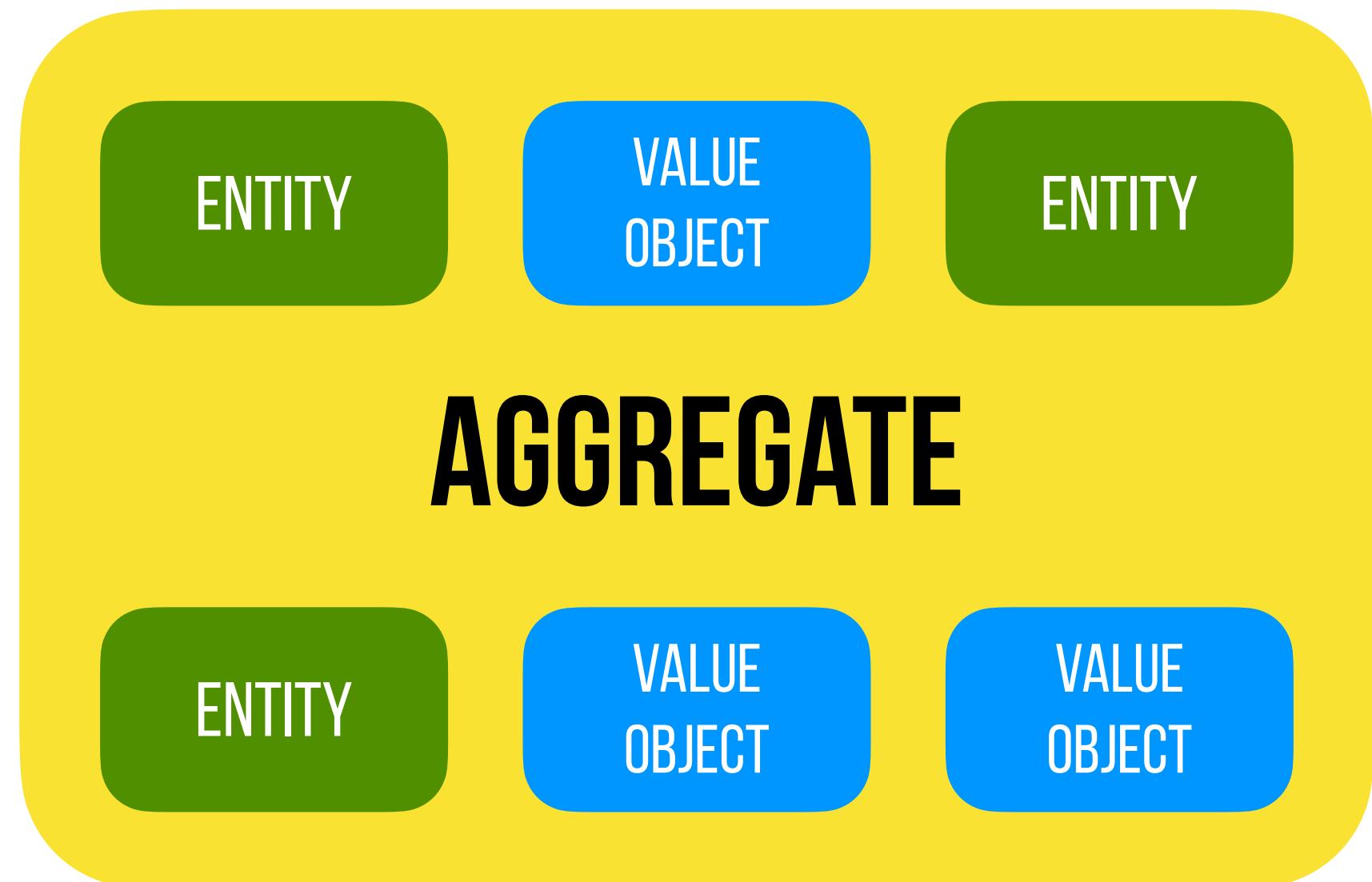
# THE CENTRAL CONCEPT

A cluster of objects  
treated as a single unit.

Only accessed through its  
Root Entity.

Often modeled as a  
state machine.

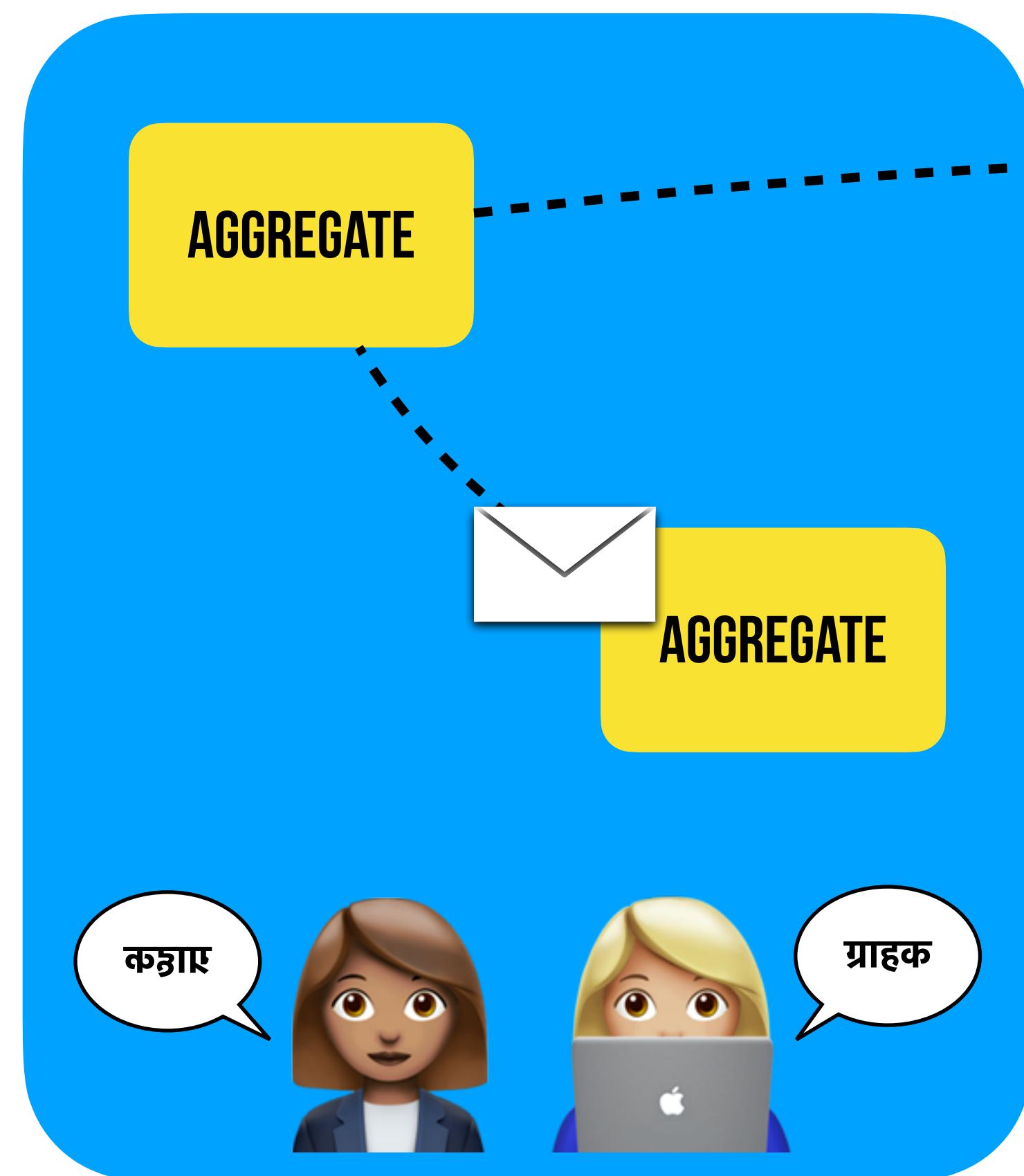
The atomic unit for any  
transactional  
behavior.



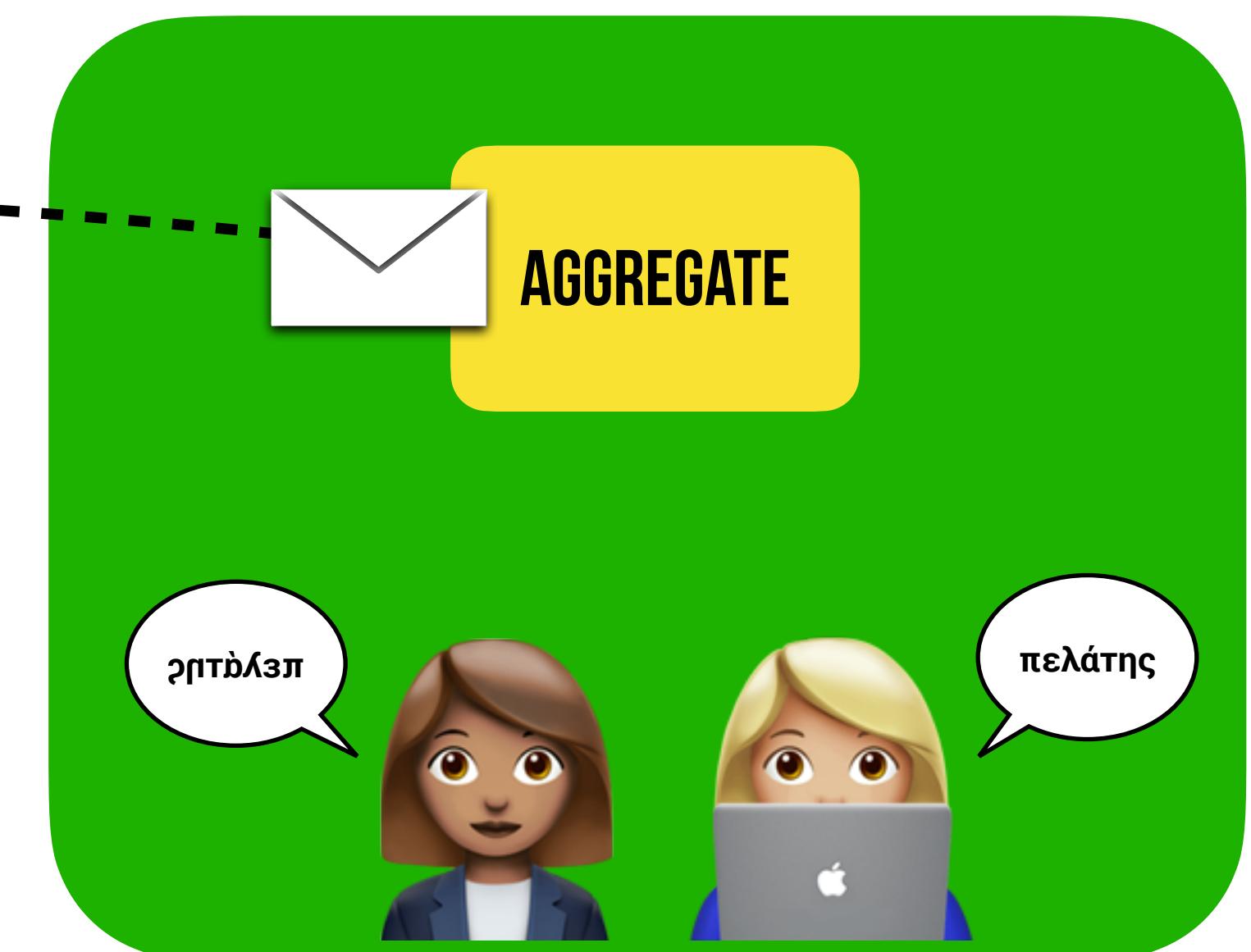
Responsible for maintaining  
any/all business invariants.



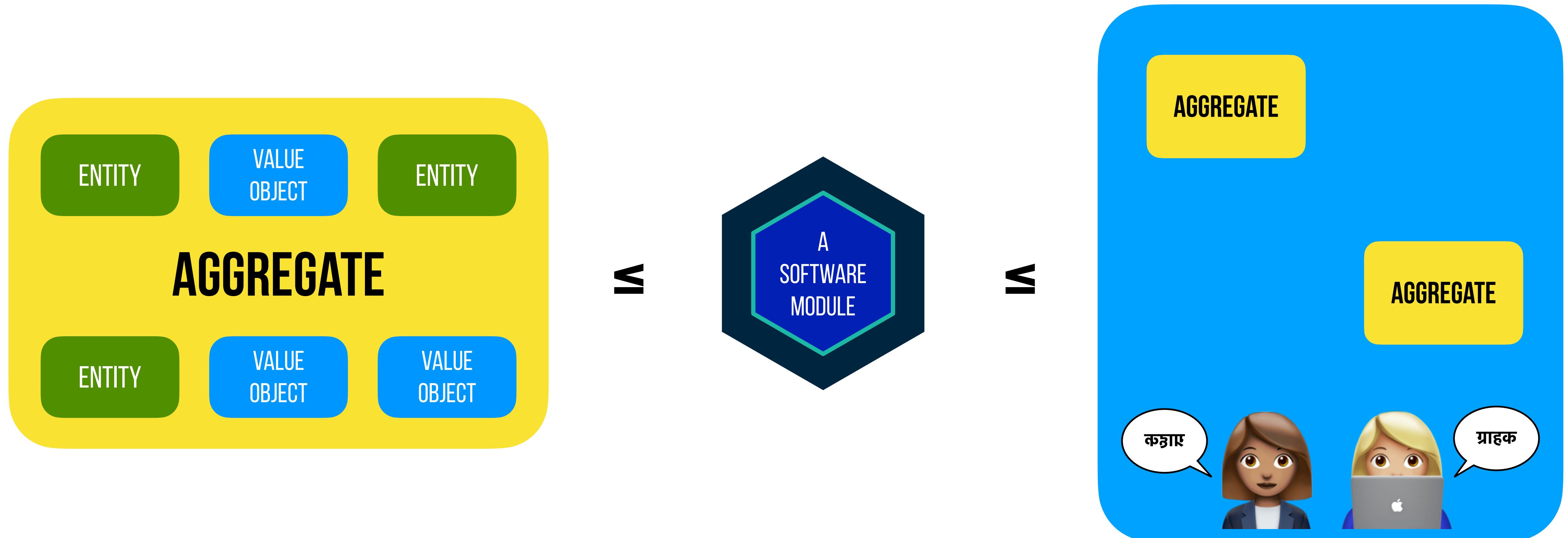
A record of a business-significant occurrence  
within a Bounded Context.



- OrderPlacedEvent
- ItemAddedToOrderEvent
- OrderCanceledEvent
- OrderPaidEvent
- OrderFulfilledEvent
- OrderDeliveredEvent

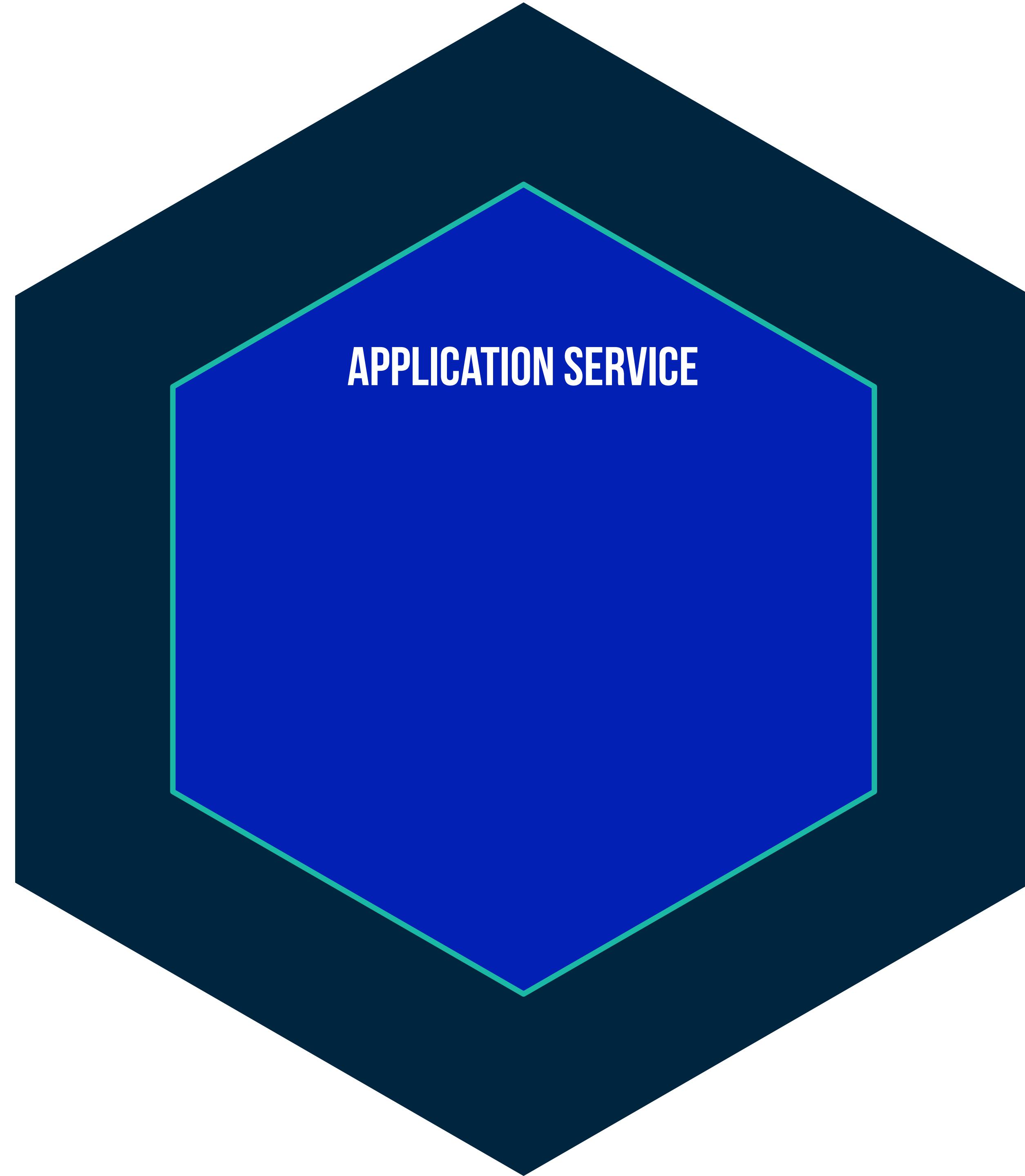


# THE GOLDEN RULE



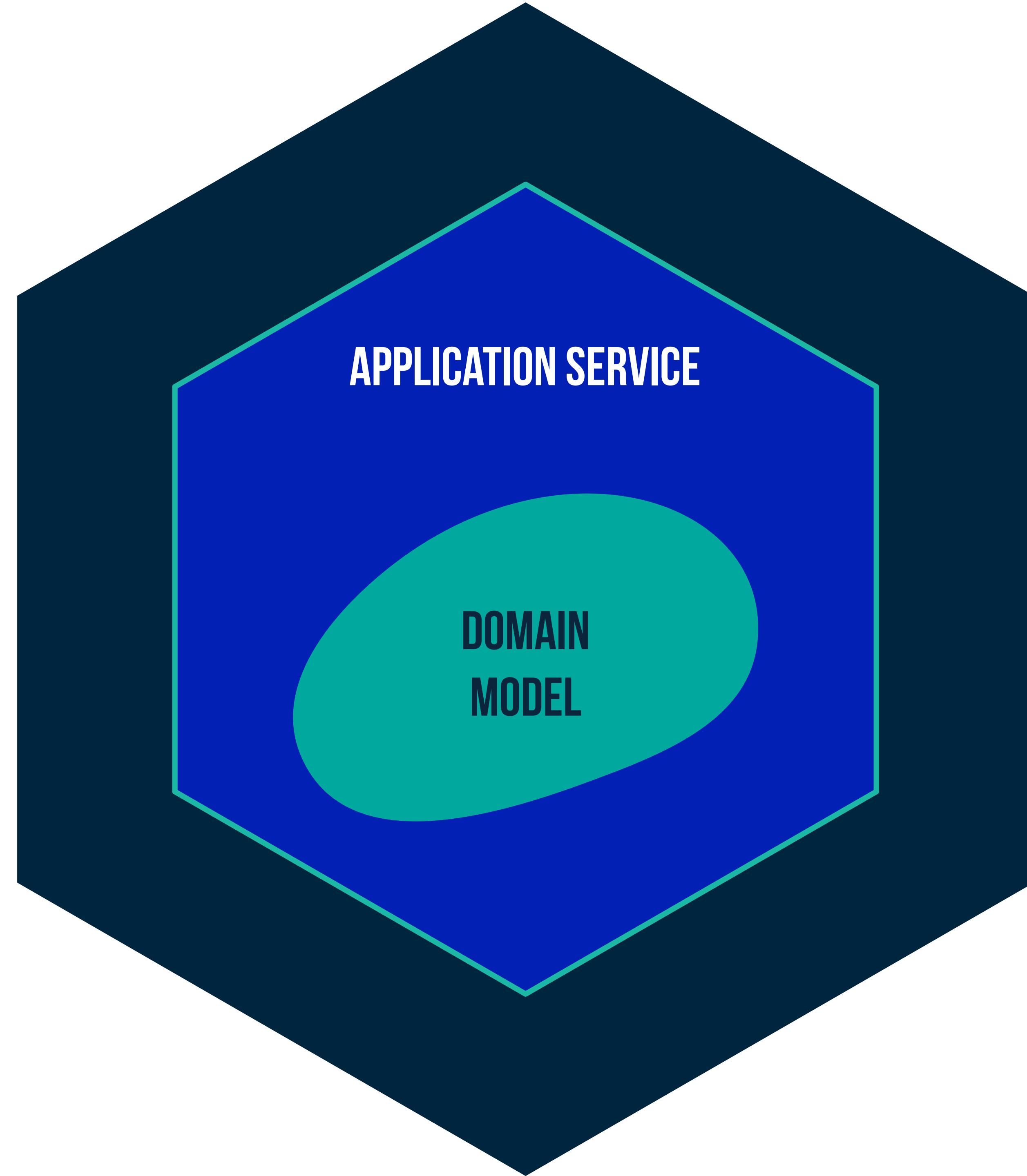
Aggregate  $\leq$  SizeOf(Module)  $\leq$  Bounded Context

# HEXAGONAL ARCHITECTURE



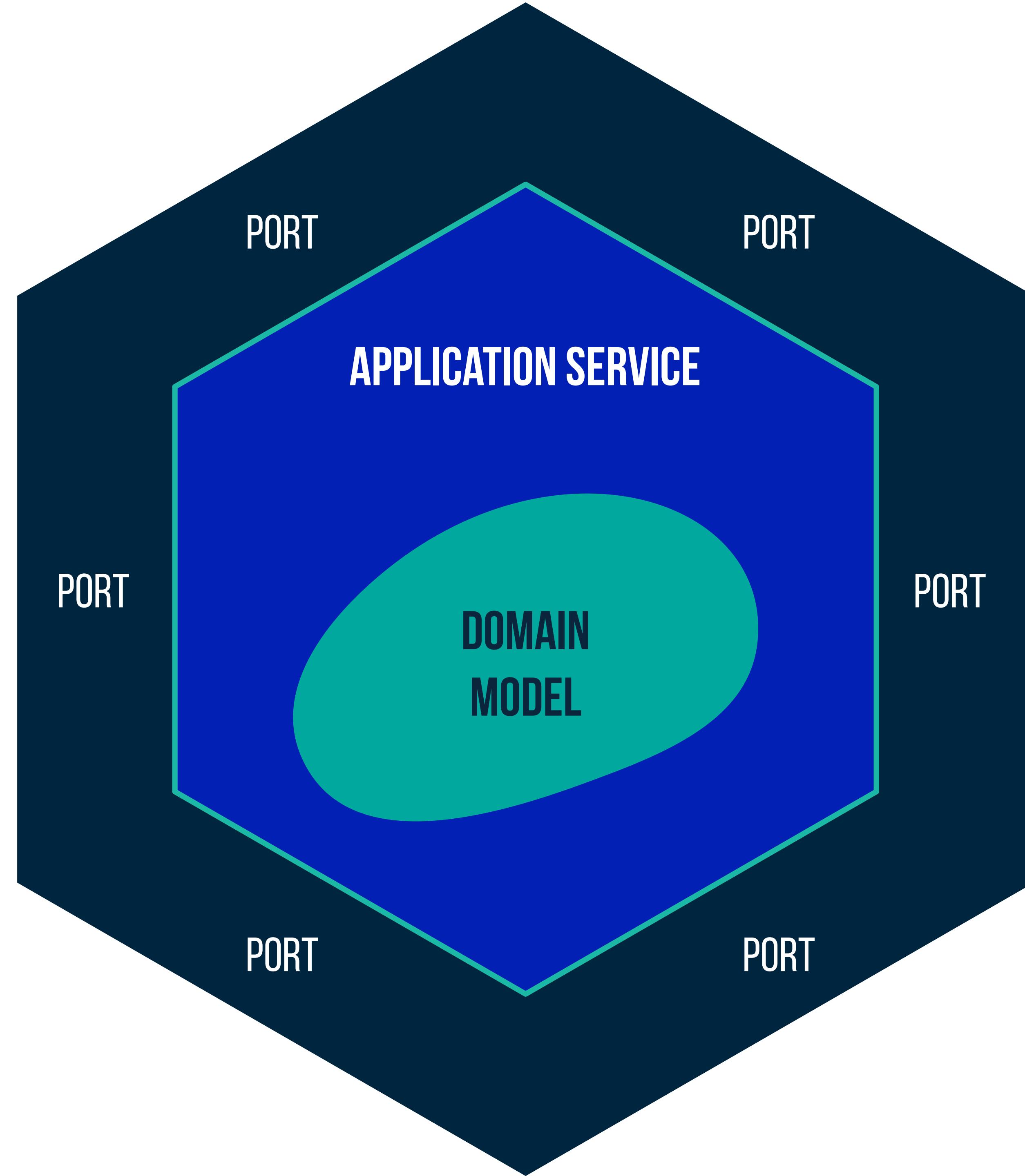
# HEXAGONAL ARCHITECTURE

- Domain Model (Aggregates, etc.) wrapped by an Application Service



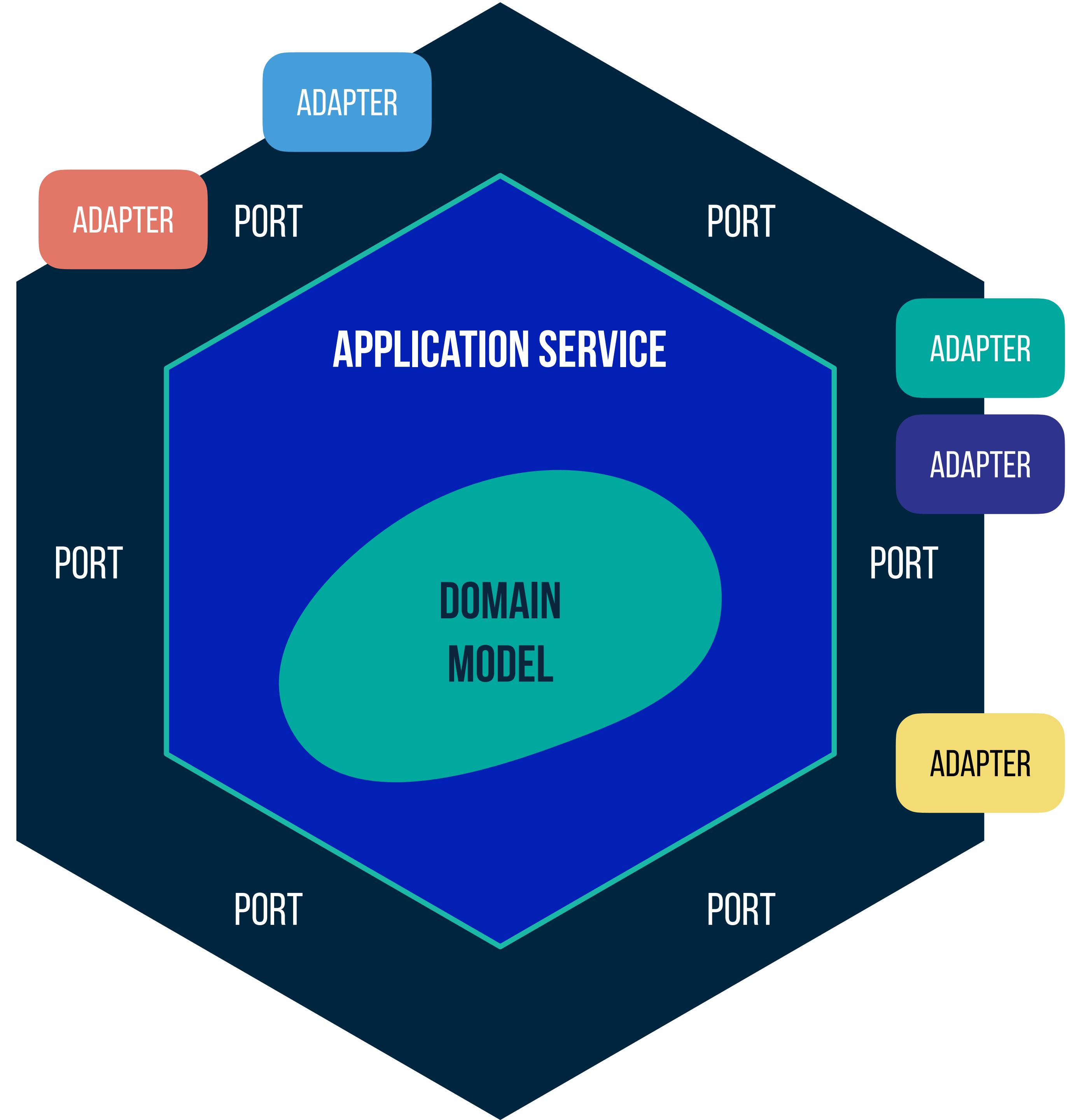
# HEXAGONAL ARCHITECTURE

- Domain Model (Aggregates, etc.) wrapped by an Application Service
- Ports represent generic entry/exit points ([interface](#))



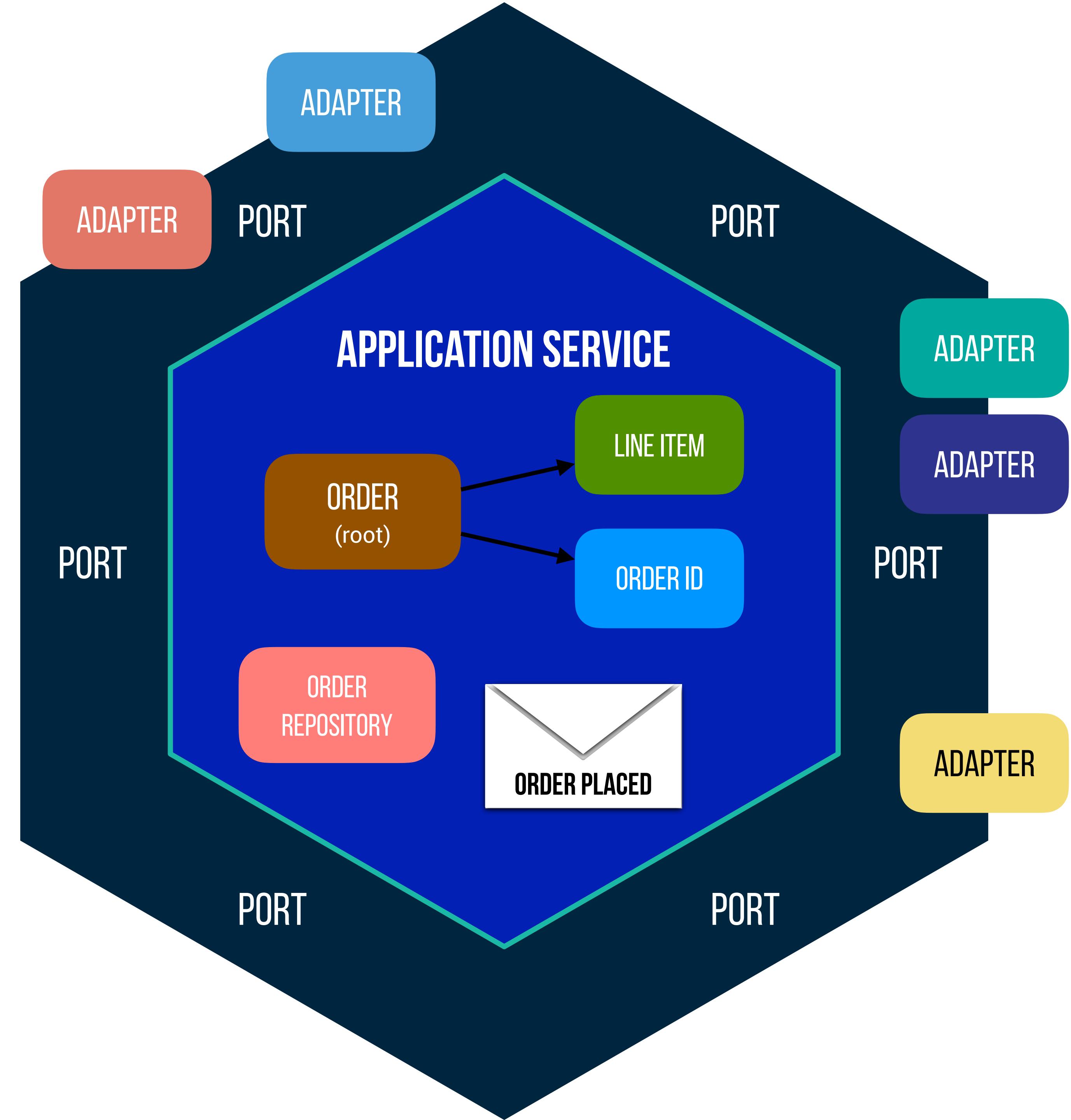
# HEXAGONAL ARCHITECTURE

- Domain Model (Aggregates, etc.) wrapped by an Application Service
- Ports represent generic entry/exit points ([interface](#))
- Adapters represent specific entry/exit implementations ([class](#))



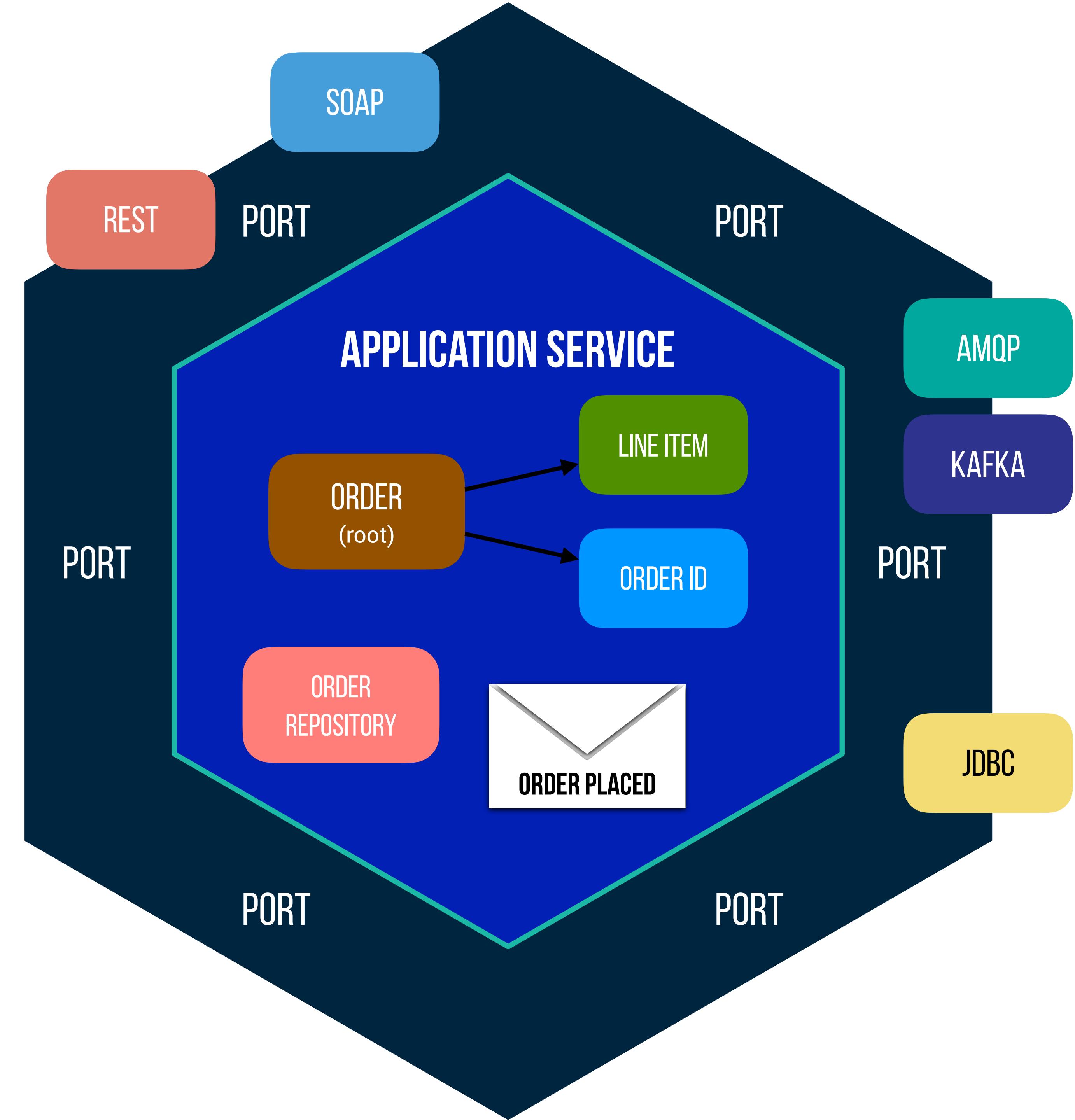
# HEXAGONAL ARCHITECTURE

- Domain Model (Aggregates, etc.) wrapped by an Application Service
- Ports represent generic entry/exit points ([interface](#))
- Adapters represent specific entry/exit implementations ([class](#))

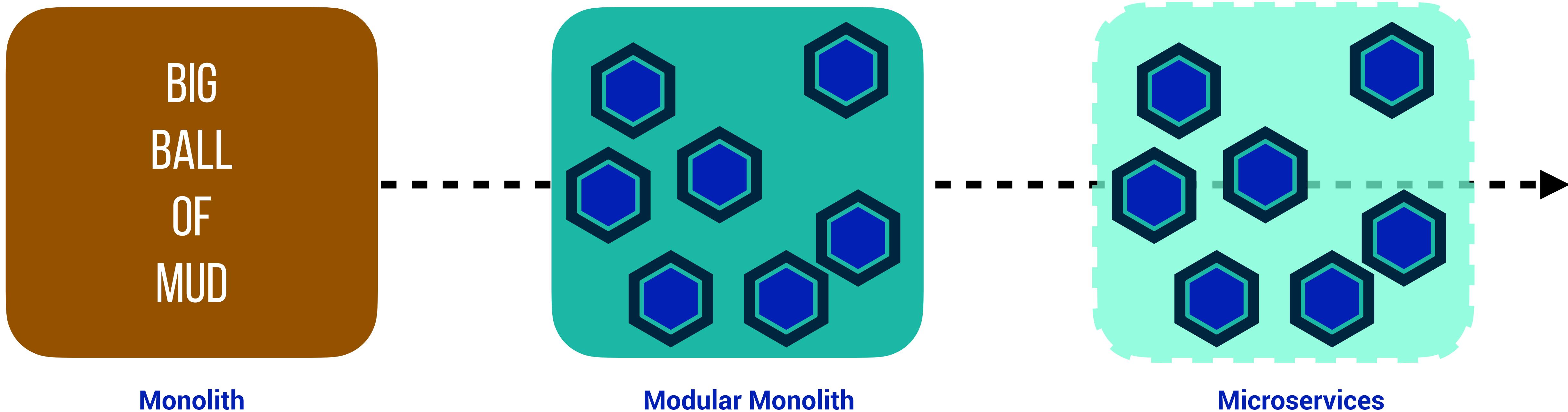


# HEXAGONAL ARCHITECTURE

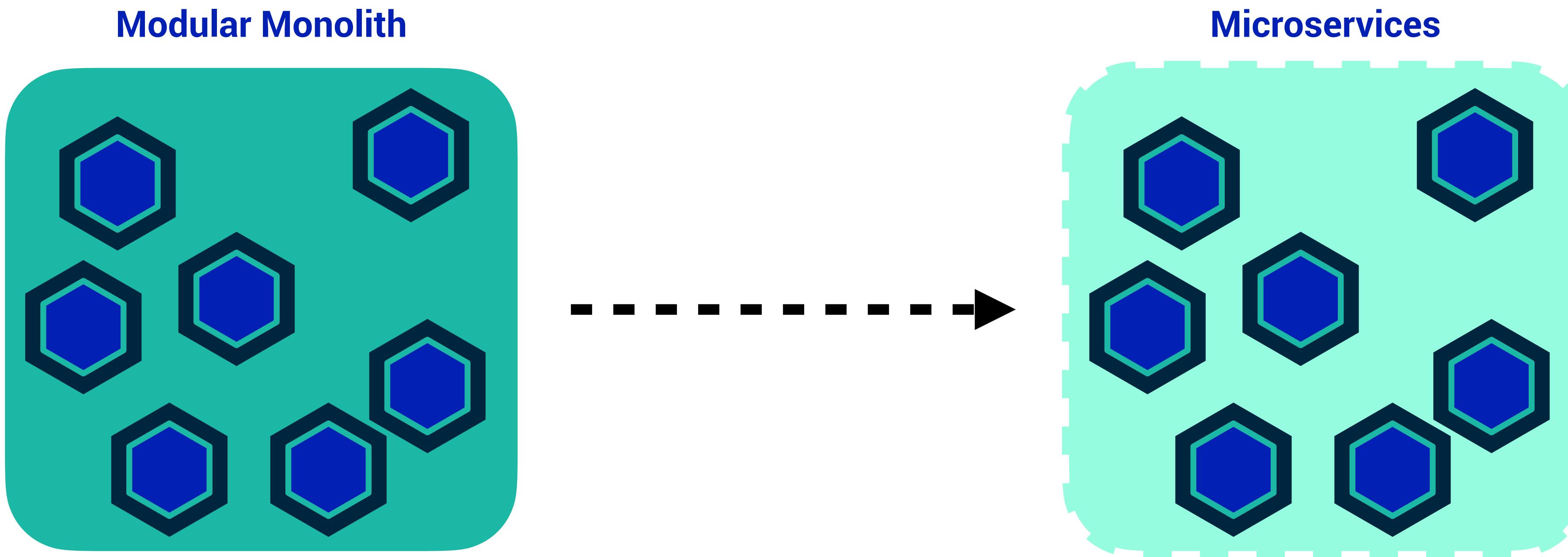
- Domain Model (Aggregates, etc.) wrapped by an Application Service
- Ports represent generic entry/exit points ([interface](#))
- Adapters represent specific entry/exit implementations ([class](#))



# DEPLOYMENT TOPOLOGY SPECTRUM



# MODULES ARE MODULES



- High Cohesion
  - Low Coupling
  - Business Capability Focus
  - Bounded Contexts / Aggregates
  - Data Encapsulation
  - Substitutable
  - Composable
- 
- Individually Deployable
  - Individually Upgradeable
  - Individually Replaceable
  - Individually Scalable
  - Heterogeneous Tech Stacks

---

The **criteria** for effectively creating modules  
inside of a **monolithic** codebase translates  
into effective criteria for microservice  
**modularity.**

---

---

We want to maintain a modular structure  
that can exist in either a monolithic or  
microservices architecture.

---

