Software Engineering

Part (VII)- System Design (II): DDD & Microservices

By: Mehran Alidoost Nia
Shahid Beheshti University, Fall 2023

MICROSERVICES
LIGHTYEAR
MICROSERVICES EVERYWHERE

What is the right size of a service in the microservice architecture?

SO YOU WANT TO SPLIT EACH FUNCTION INTO ITS OWN PROCESS
AND CALL THEM MICRO SERVICES?
memegenerator.net

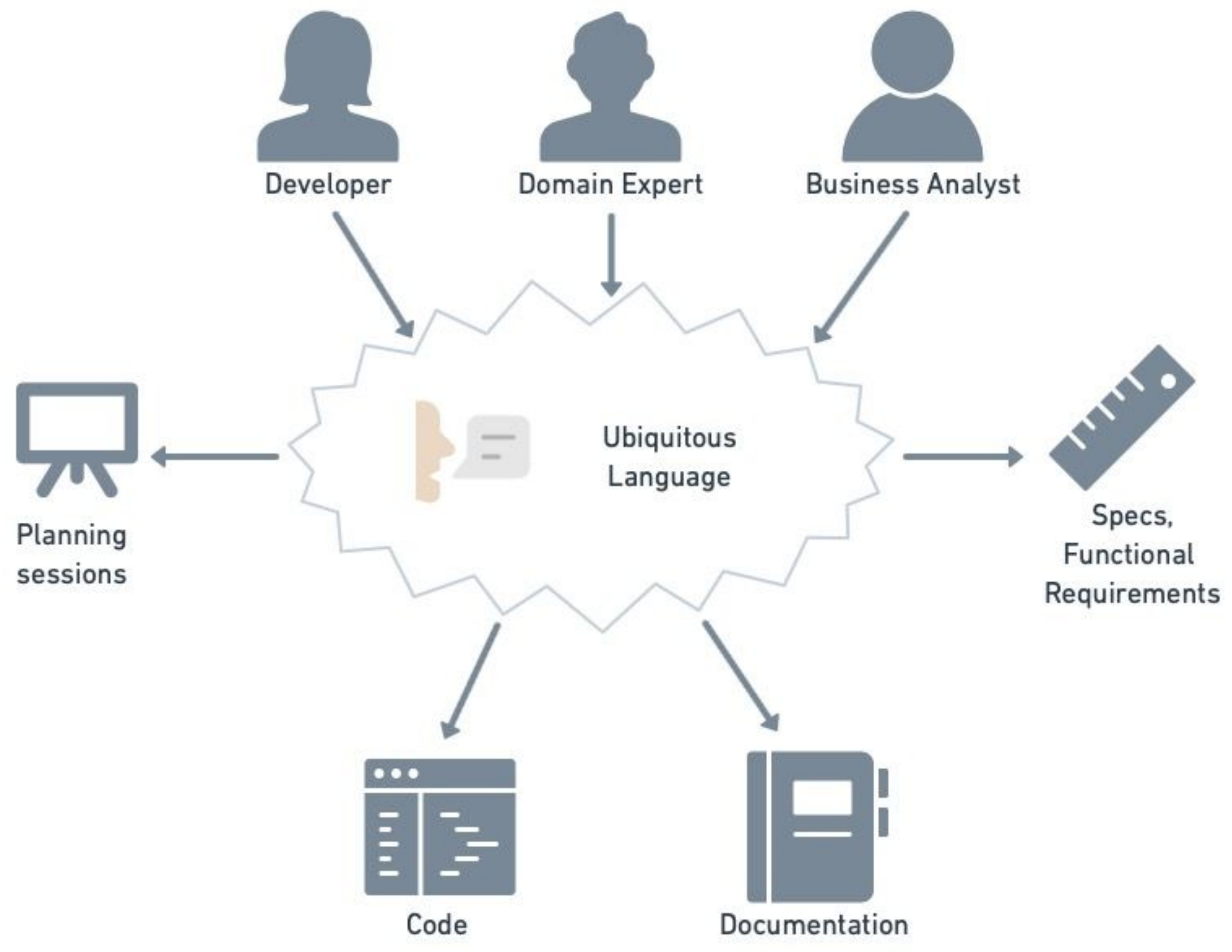
Domain

- A sphere of knowledge, influence, or activity.
- The subject area to which the user applies a program is the domain of the software.

Domain-Driven Design

- Domain-Driven Design (DDD) is a **software design method** wherein developers construct models to understand the business requirements of a domain.
- These models serve as the **conceptual foundation** for developing software.

A Perfect Overview of DDD



DDD is an approach for building complex software applications that is centered on the development of an object-oriented domain model.

Designing a city analogy

Unplanned

Big Ball Of Mud

Planned

Domain Driven Design

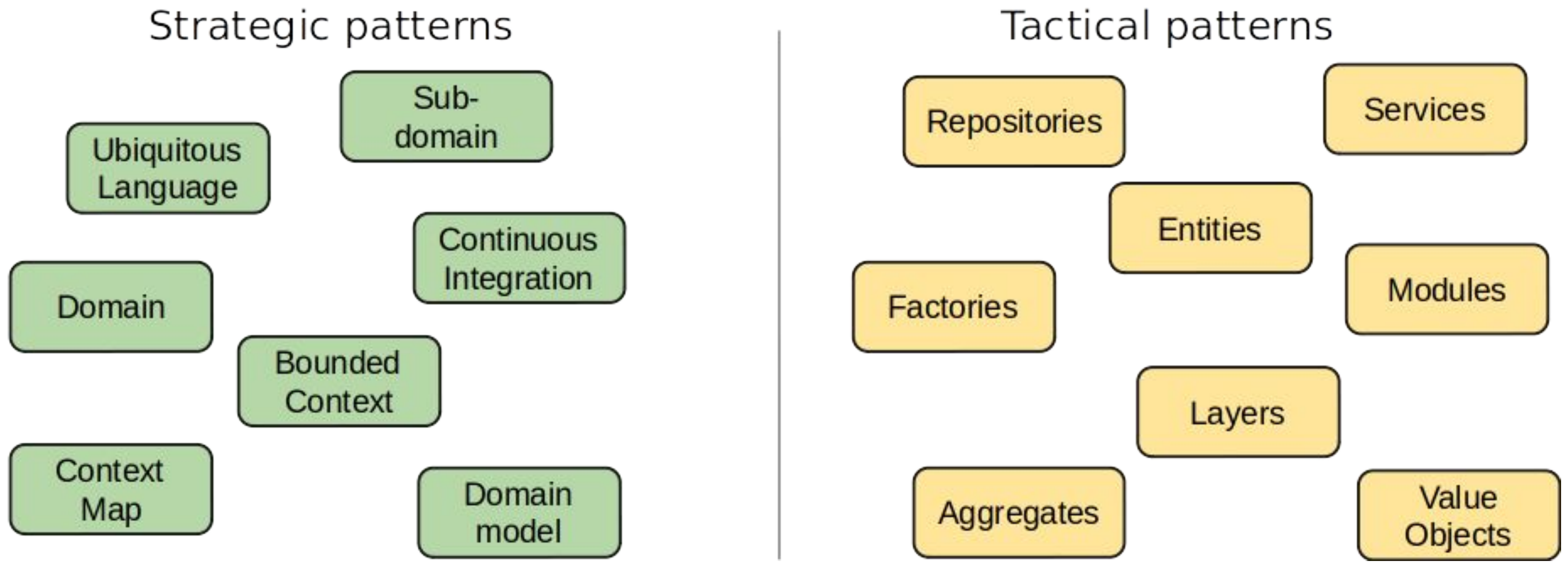
Advantages

- Simpler communication
- More flexibility
- The domain is more important than UI/UX

Difficulties

- Deep domain knowledge is needed
- Contains repetitive practices

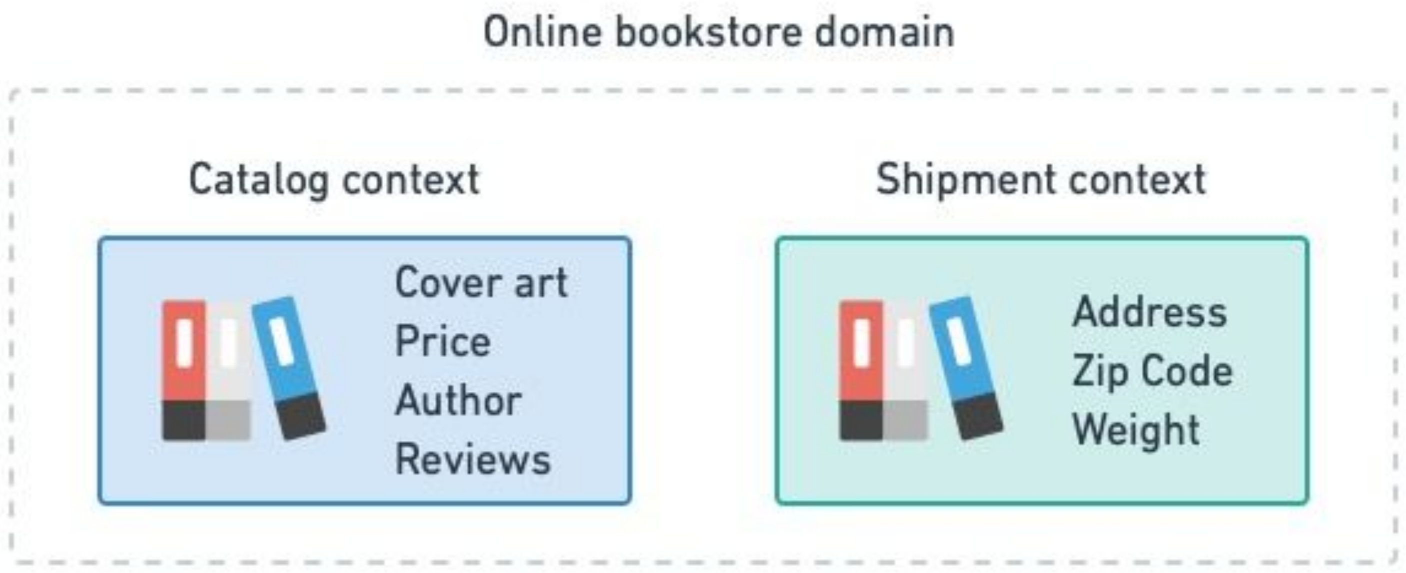
DDD Patterns



Strategic Patterns

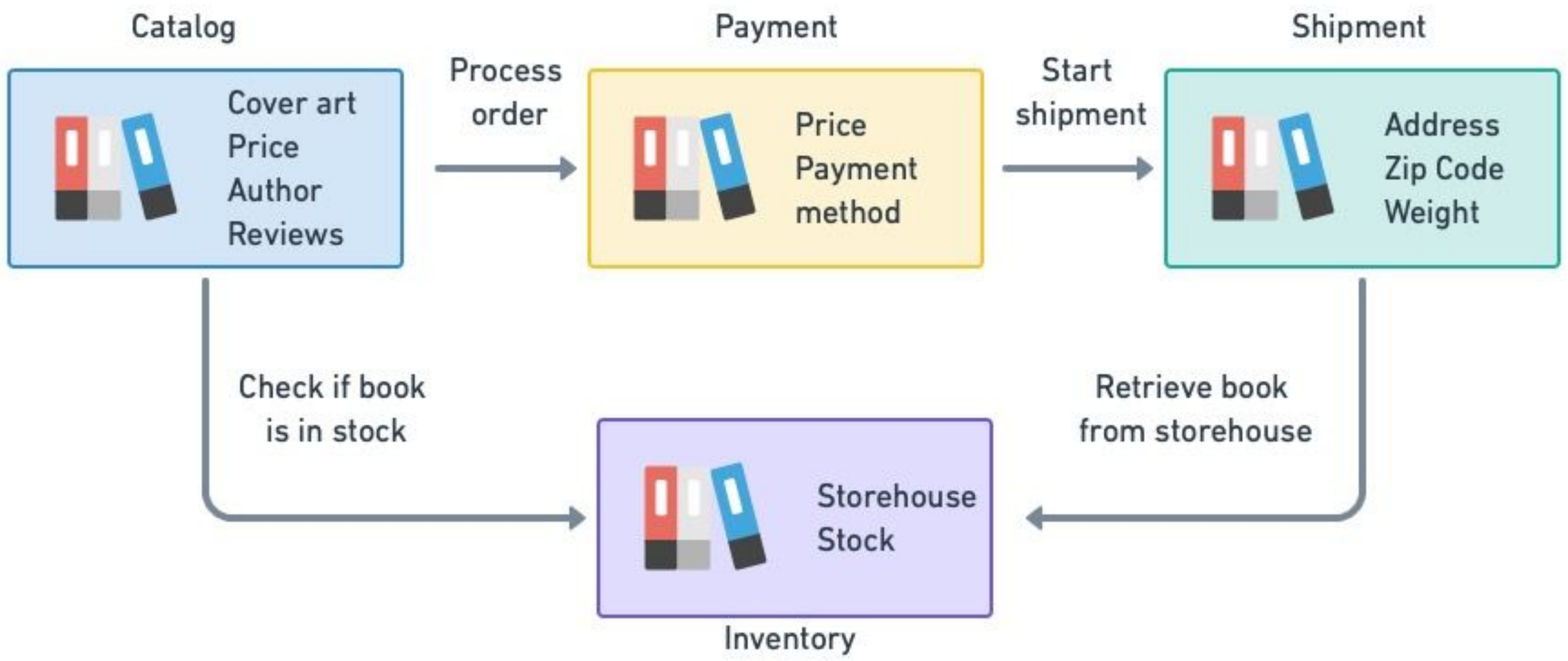
Bounded Context

- A bounded context (BC) is the space in which a term has a definite and unambiguous meaning.

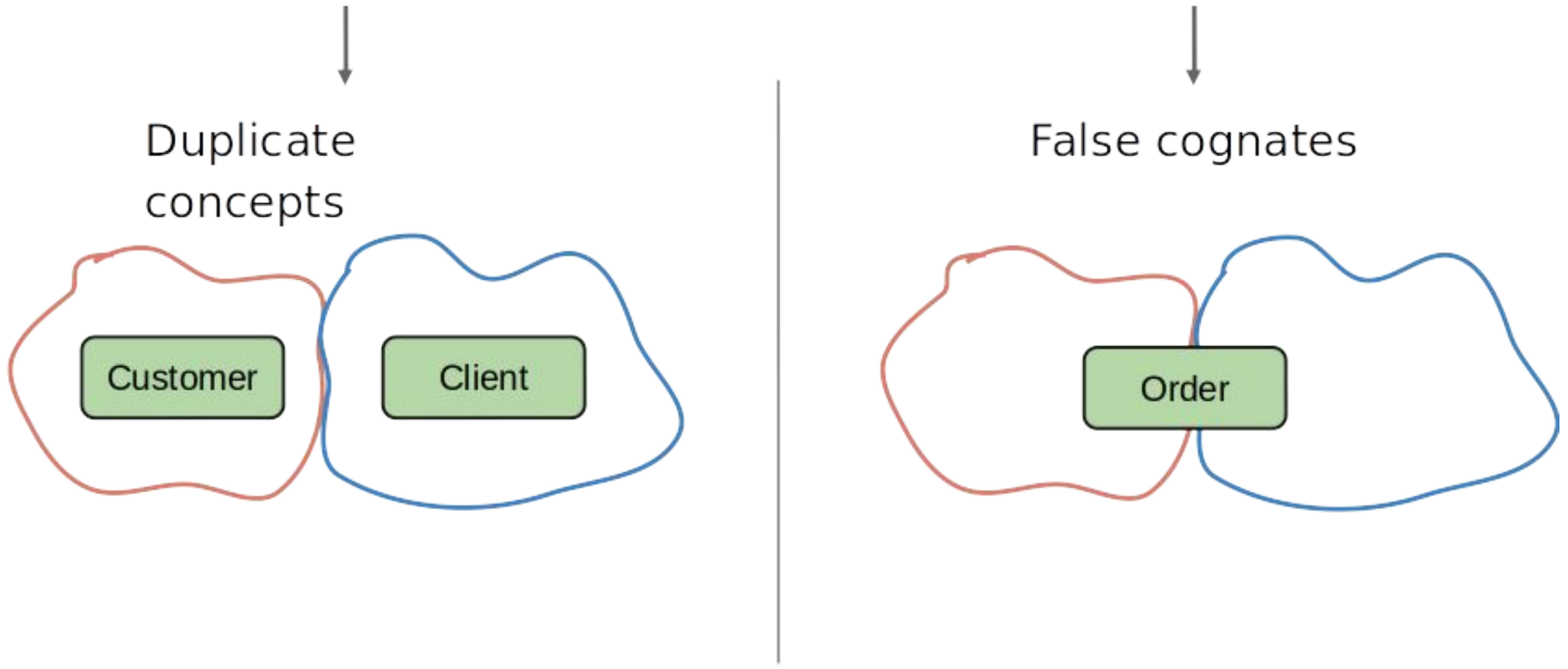


Context Map

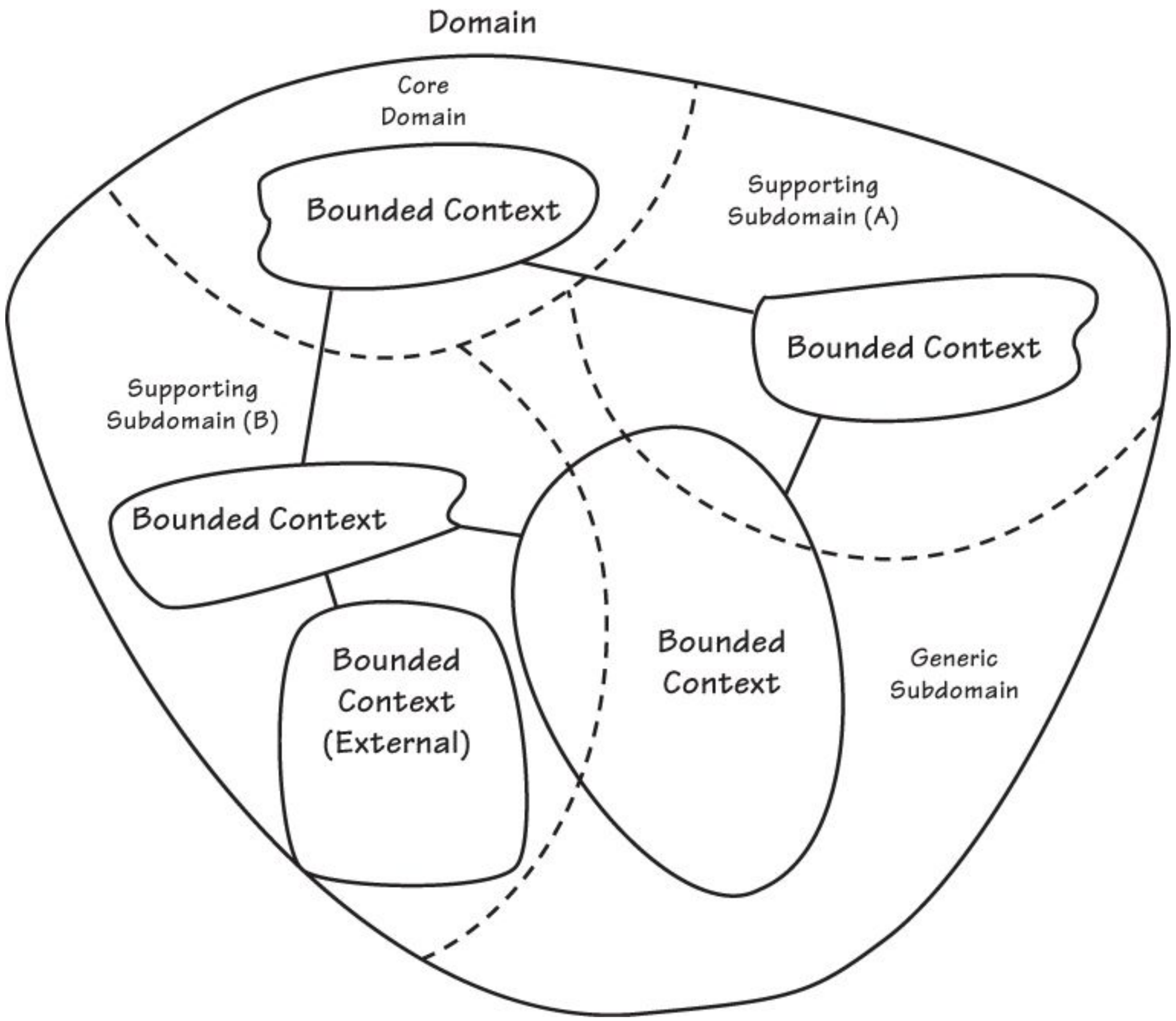
- The relationships among the BCs are depicted in the form of a context map.



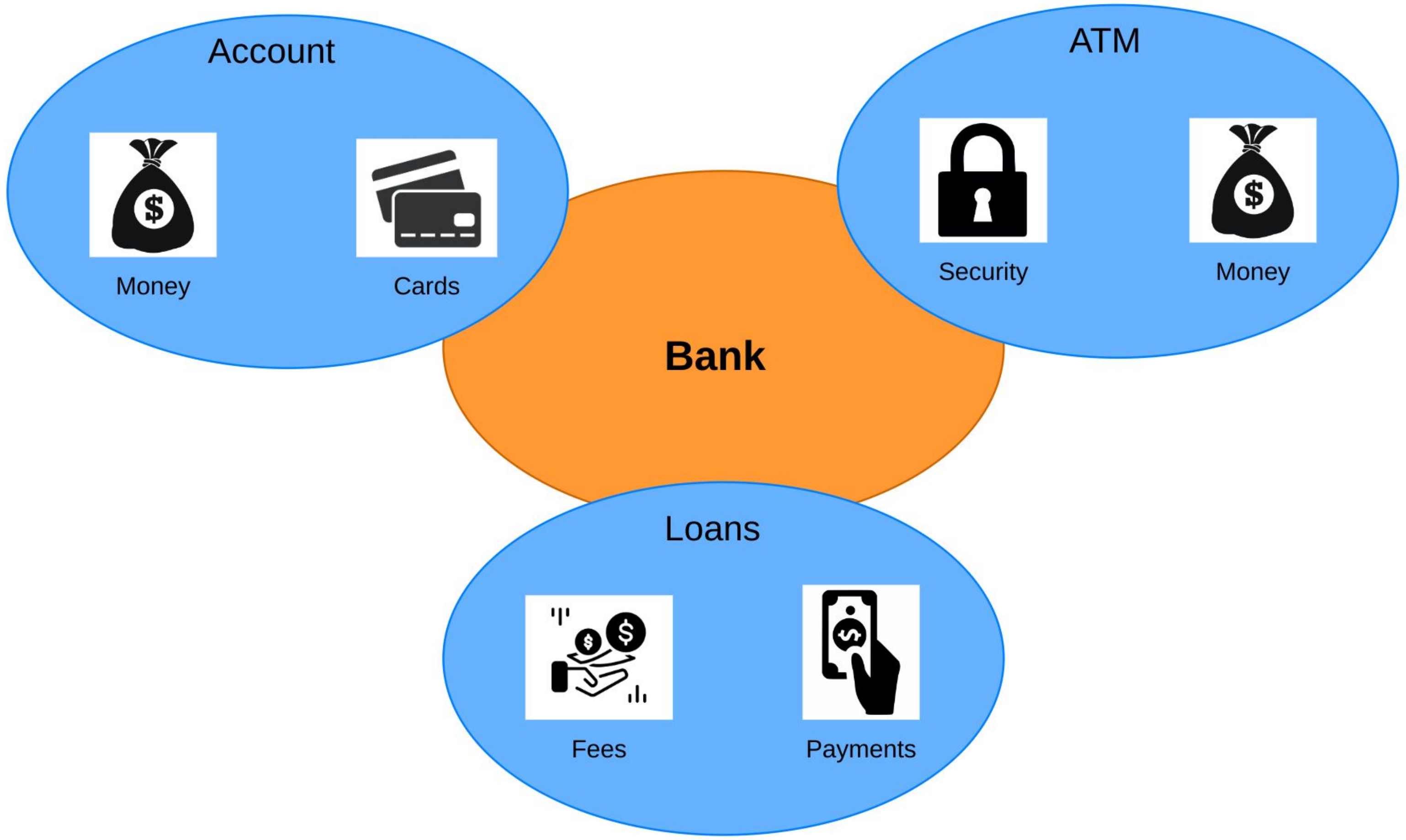
Bounded Context: possible problems



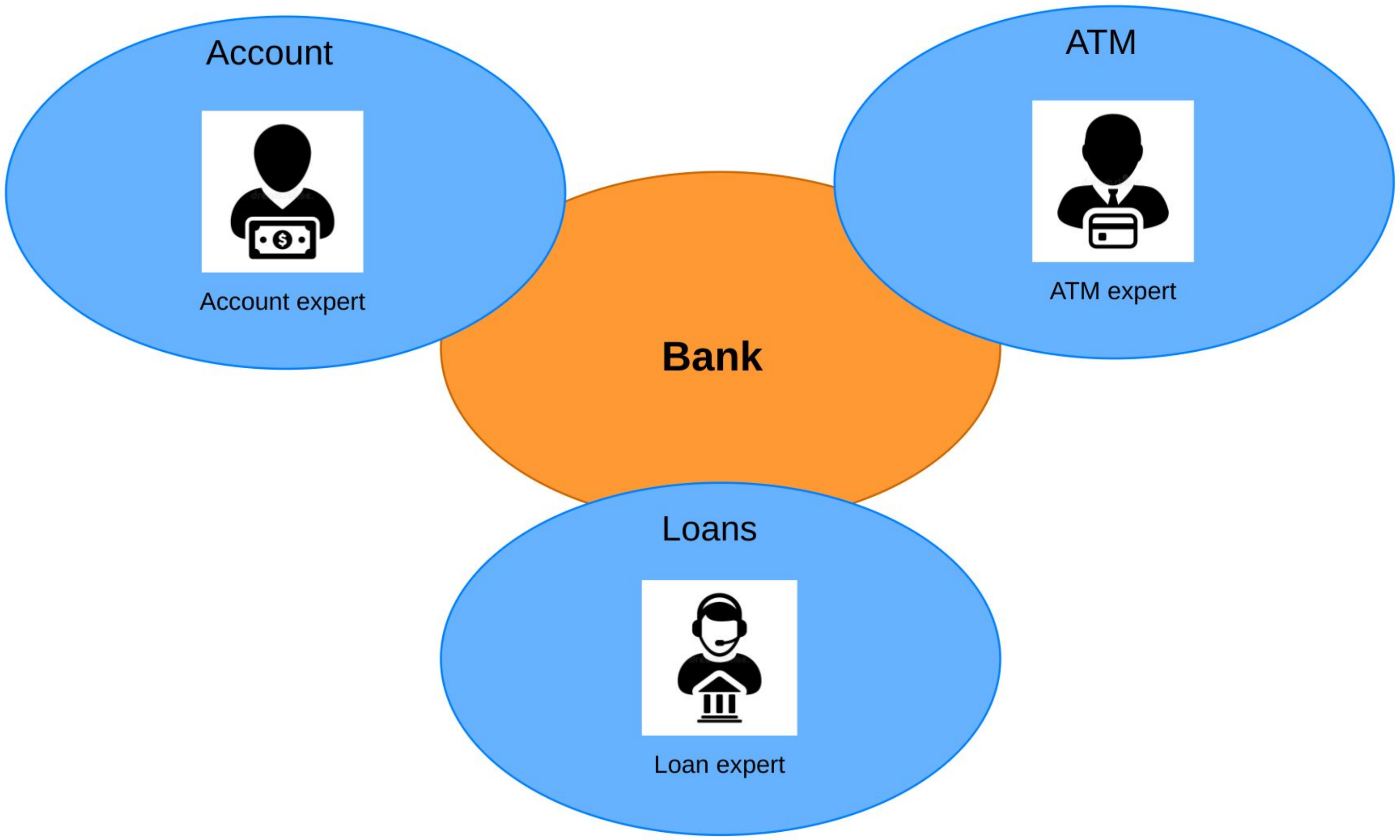
DDD toolbox: Domain, Subdomain



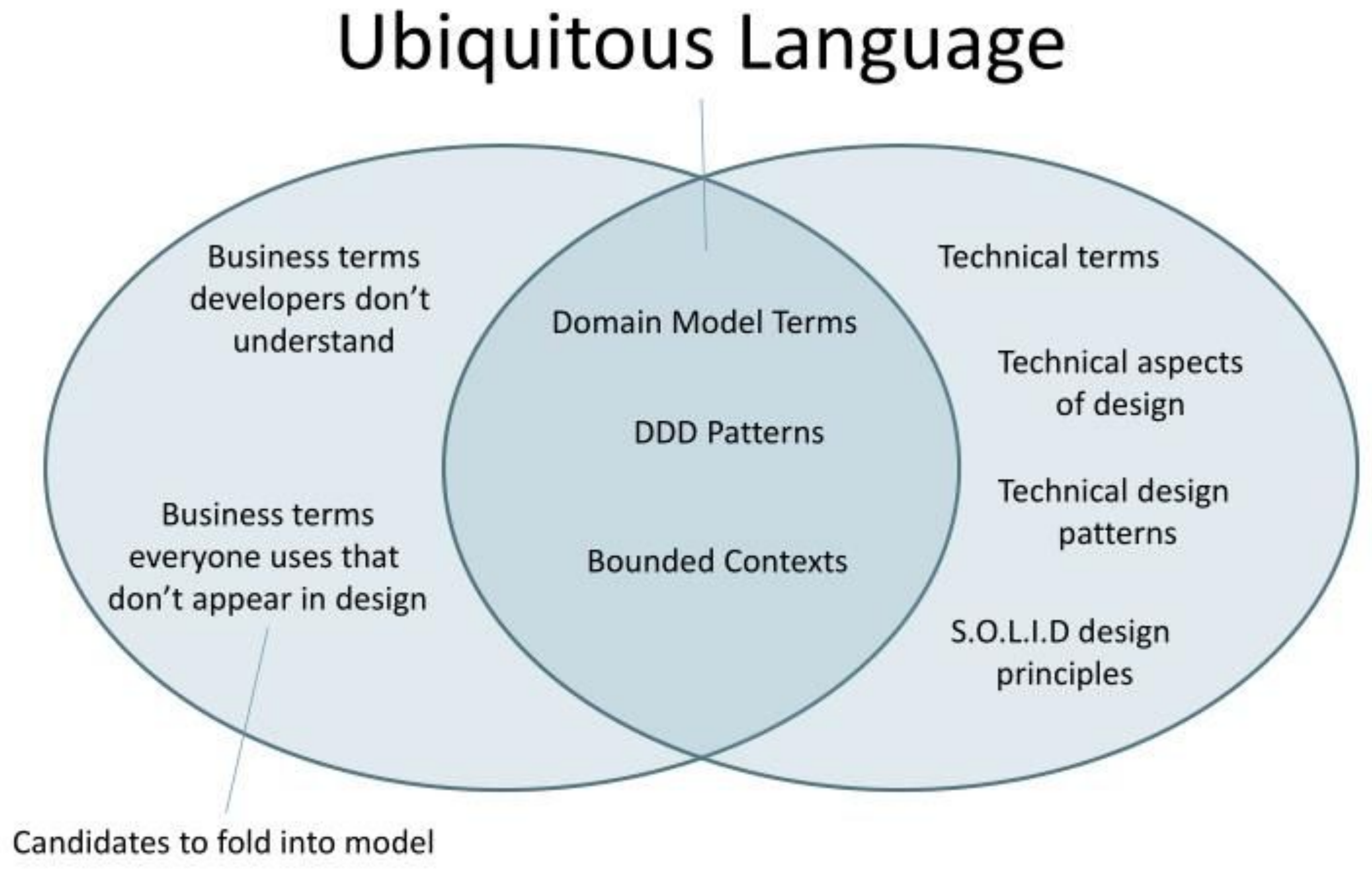
Domain/Subdomains: An Example



Domain Experts

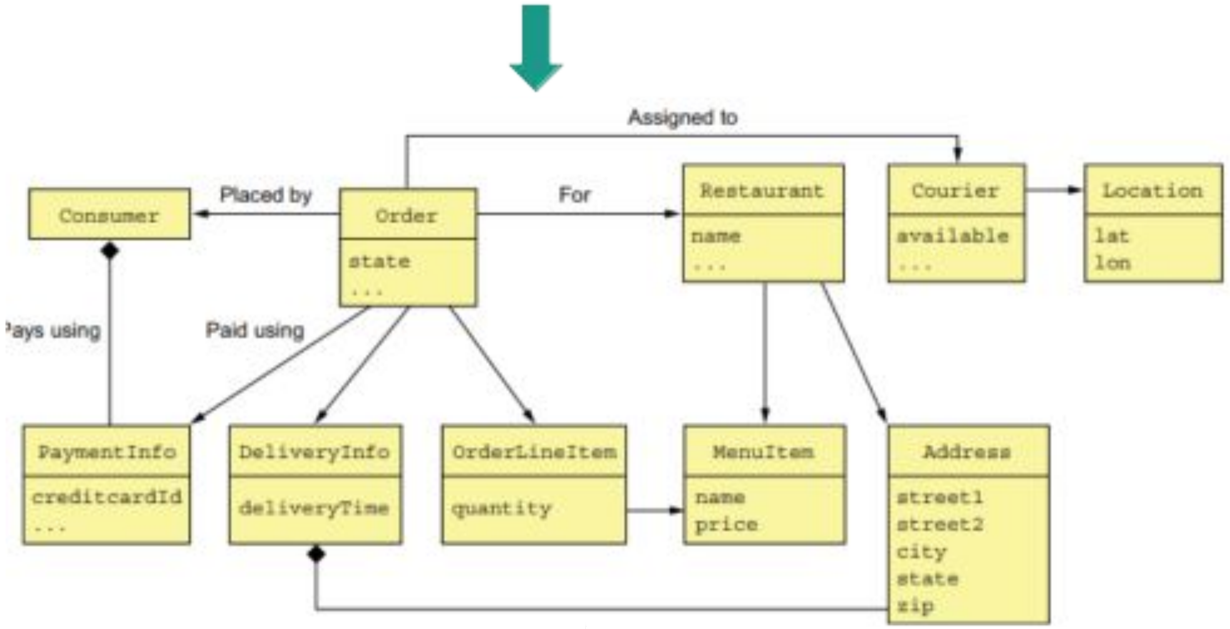


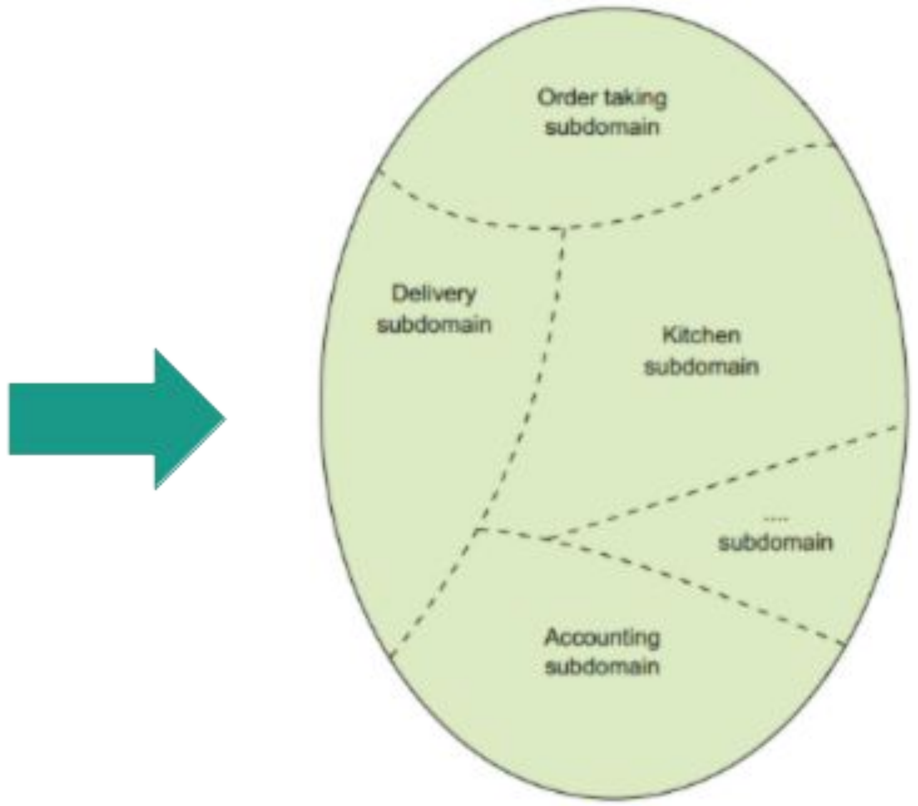
DDD toolbox: Ubiquitous Language


Ubiquitous Language
Business terms developers don't understand
Business terms everyone uses that don't appear in design
Candidates to fold into model
Domain Model Terms
DDD Patterns
Bounded Contexts
Technical terms
Technical aspects of design
Technical design patterns
S.O.L.I.D design principles

Result of using Ubiquitous Language, Domain and Subdomain

```
Given a consumer
  And a restaurant
  And a delivery address/time that can be served by that restaurant
  And an order total that meets the restaurant's order minimum
When the consumer places an order for the restaurant
Then consumer's credit card is authorized
  And an order is created in the PENDING_ACCEPTANCE state
  And the order is associated with the consumer
  And the order is associated with the restaurant
```





Tactical Patterns

Value Object

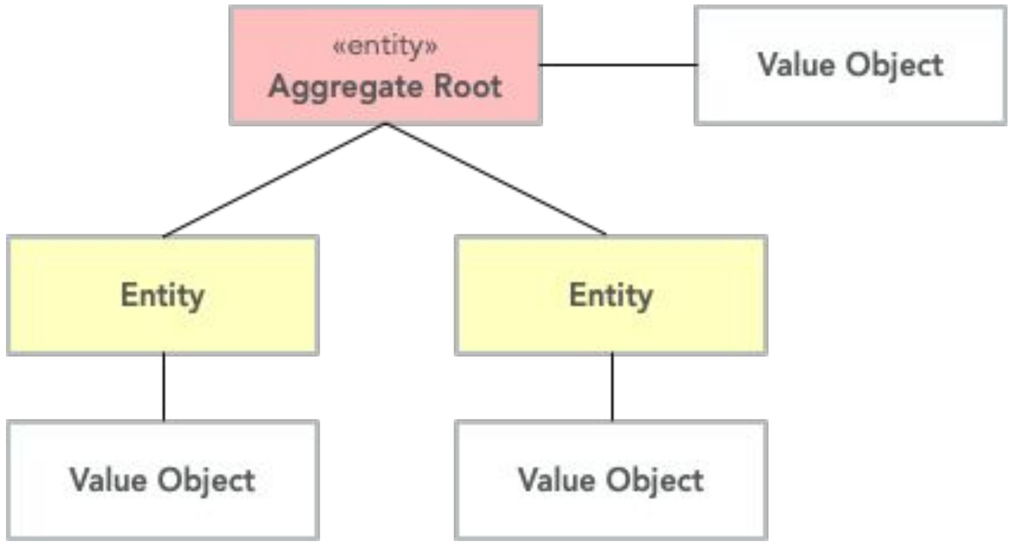
- A value object is an object whose value is of importance.
- Two value objects with the exact same value can be considered the same value object and are thus interchangeable.
- Value objects should always be made immutable.
- For complex value objects, consider using the builder or essence pattern.

Entities

- An entity is an object whose identity is of importance.
- Every entity has a unique ID that is assigned when the entity is created and remains unchanged.
- As opposed to value objects, entities are mutable.
- However, that does not mean you should create setter methods for every property.

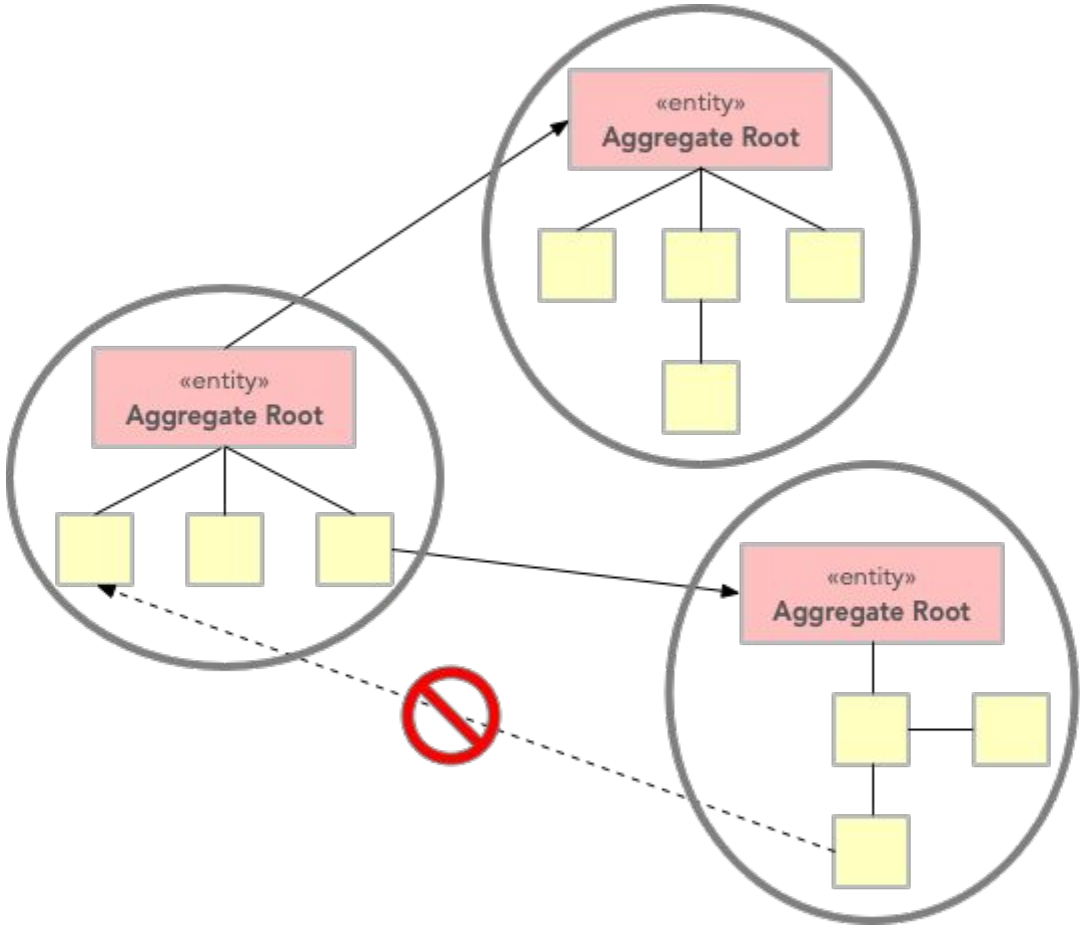
DDD toolbox: Aggregate

- An aggregate is a group of entities and value objects that have certain characteristics:
  - It is created, retrieved, and stored as a whole.
  - The aggregate is always in a consistent state.
  - It owned by an entity called the aggregate root, whose ID is used to identify the aggregate itself.

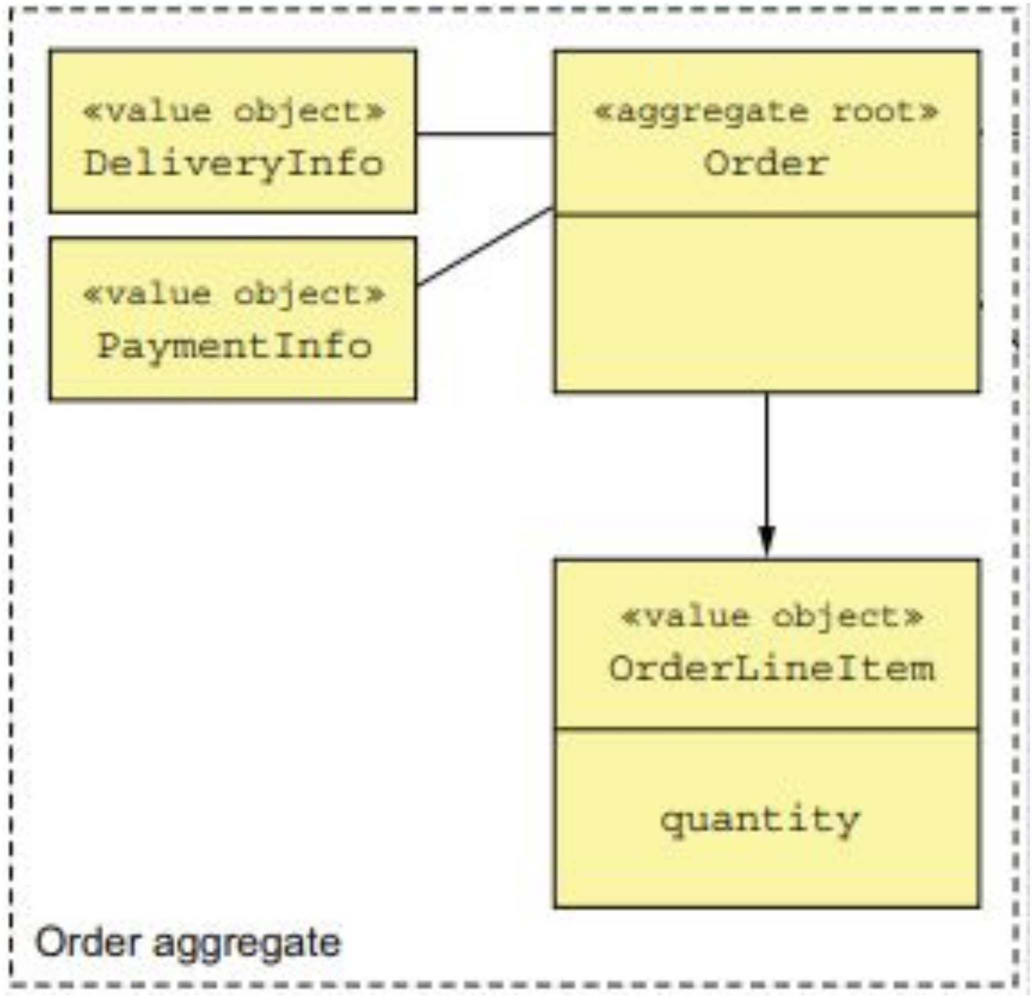


Aggregate Constraints

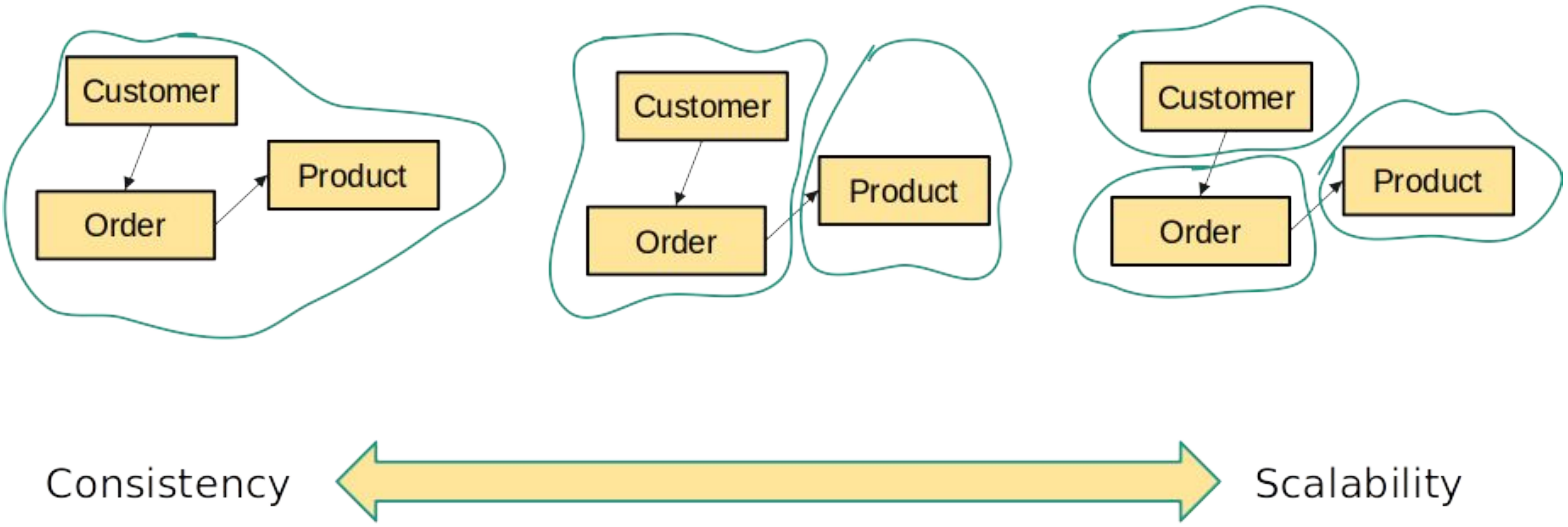
- An aggregate can be referenced from the outside through its root only. Objects outside of the aggregate may not reference any other entities inside the aggregate.



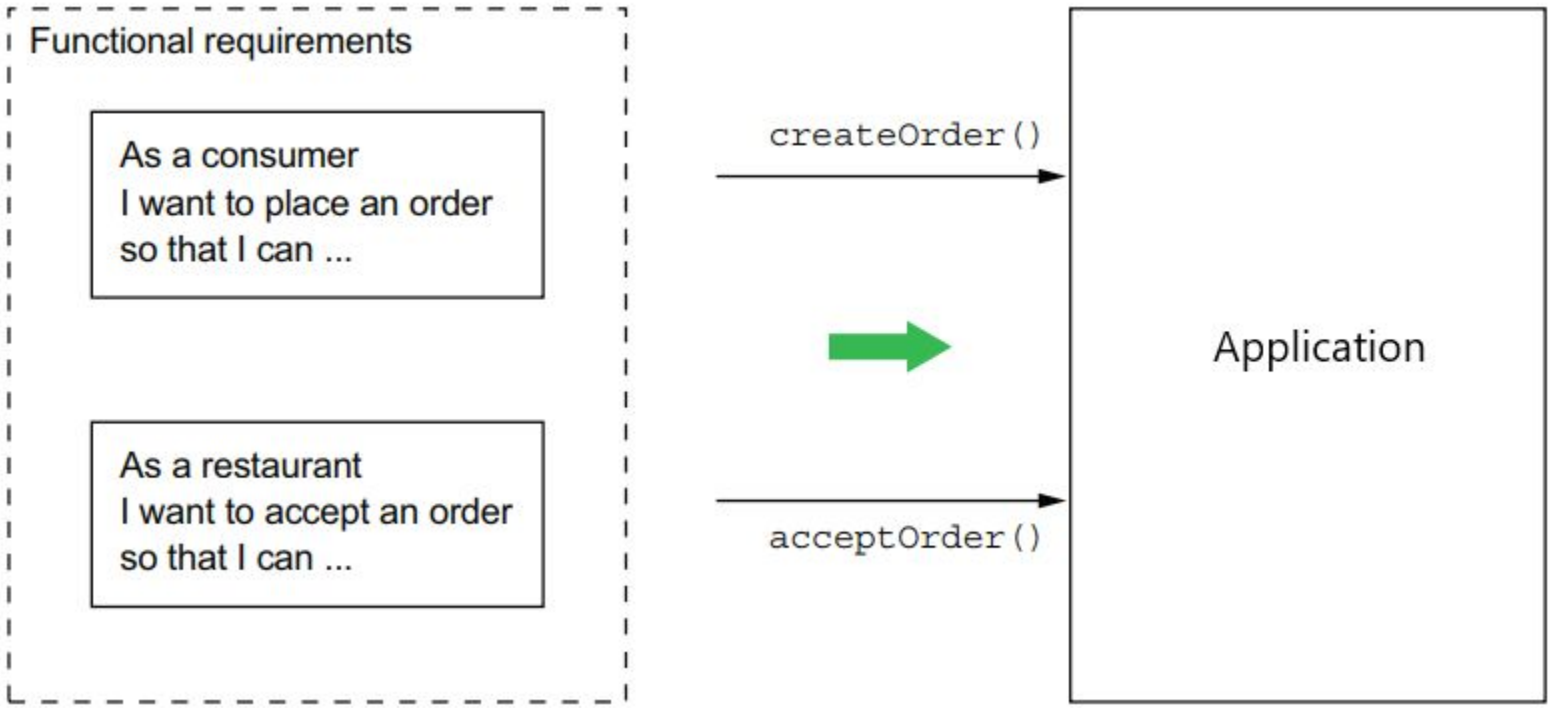
Aggregate: An Example



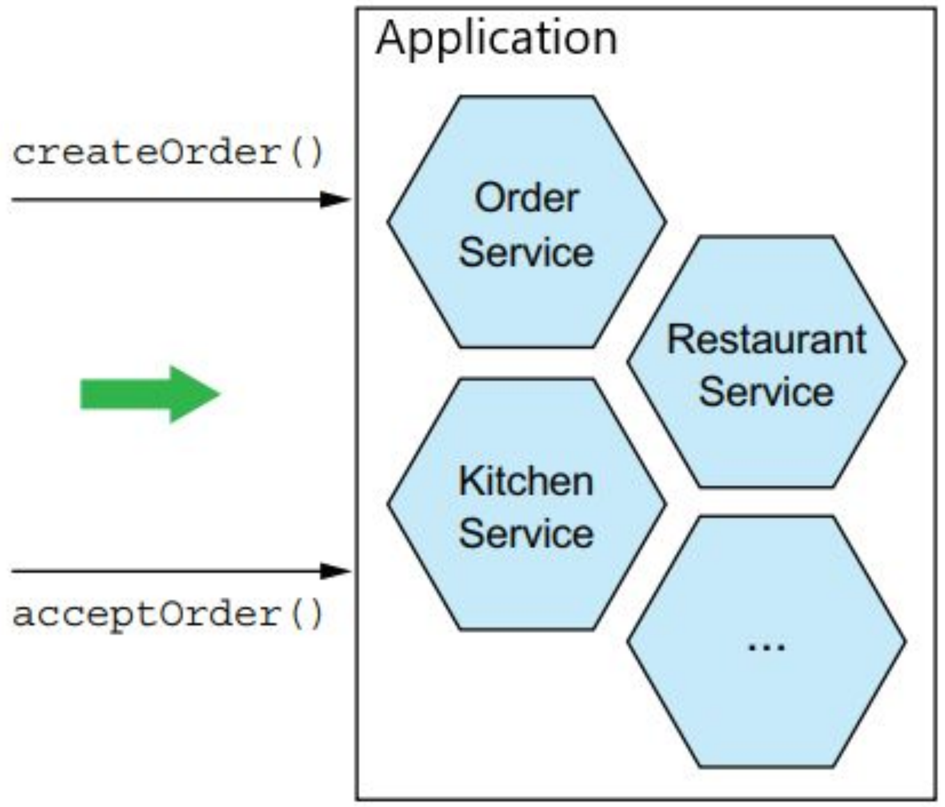
Aggregate granularity



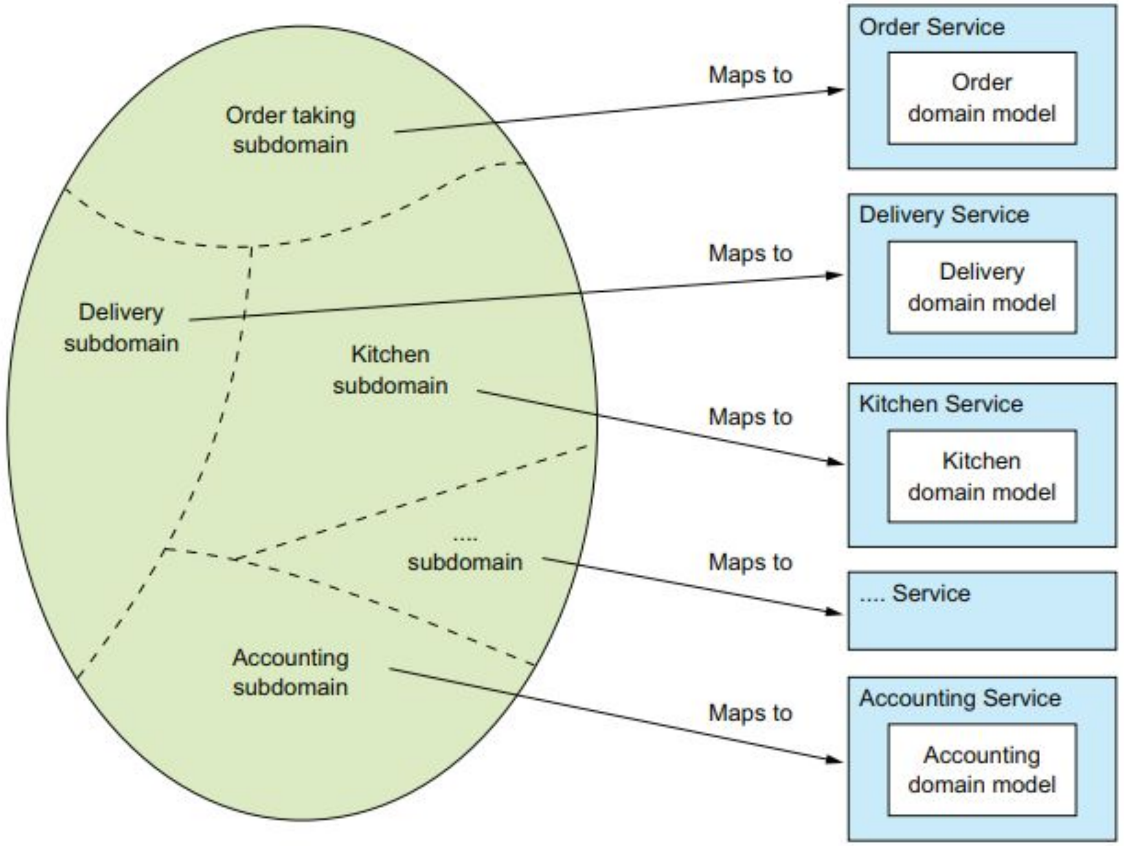
Service Creation



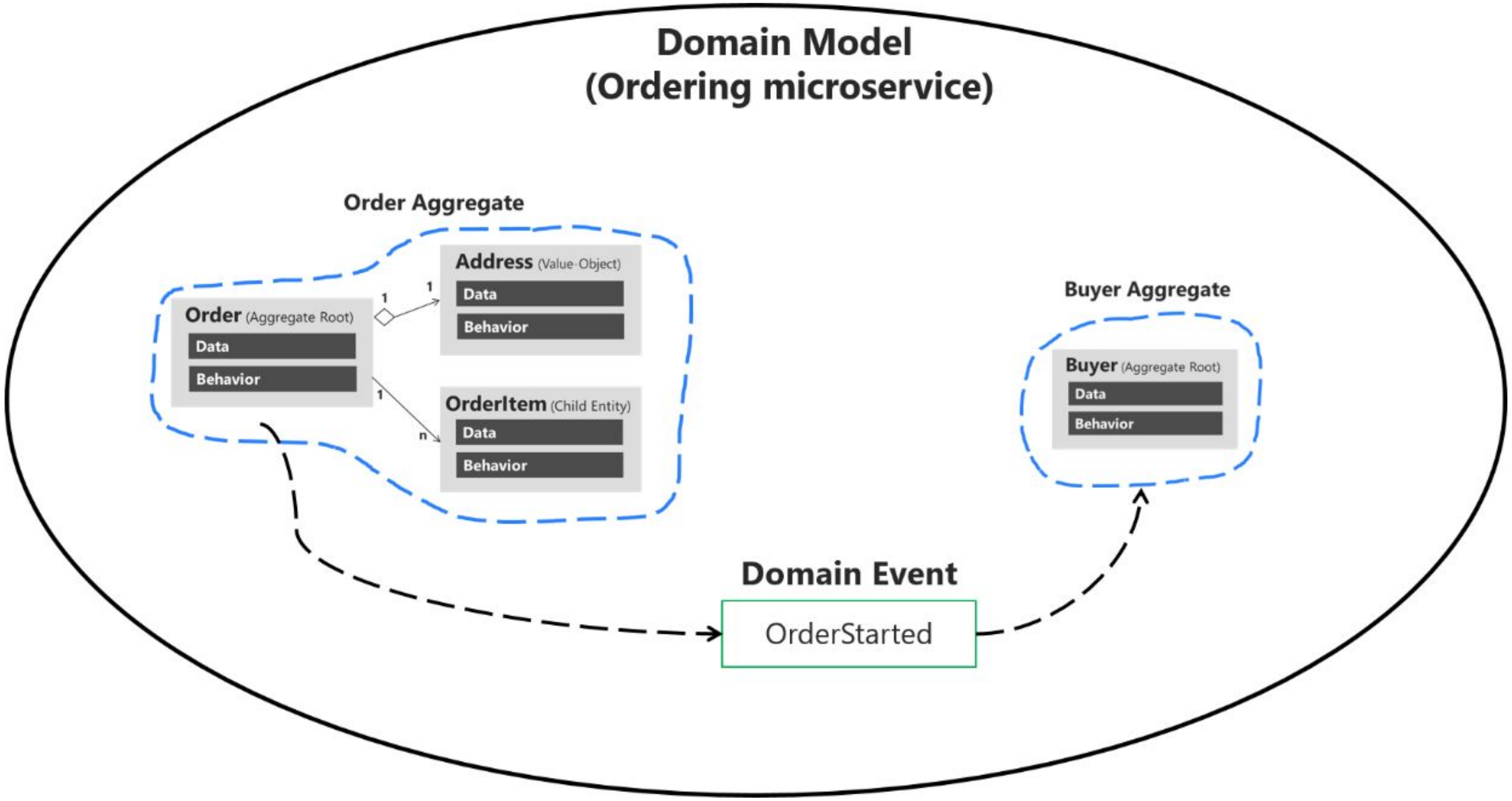
Application Services



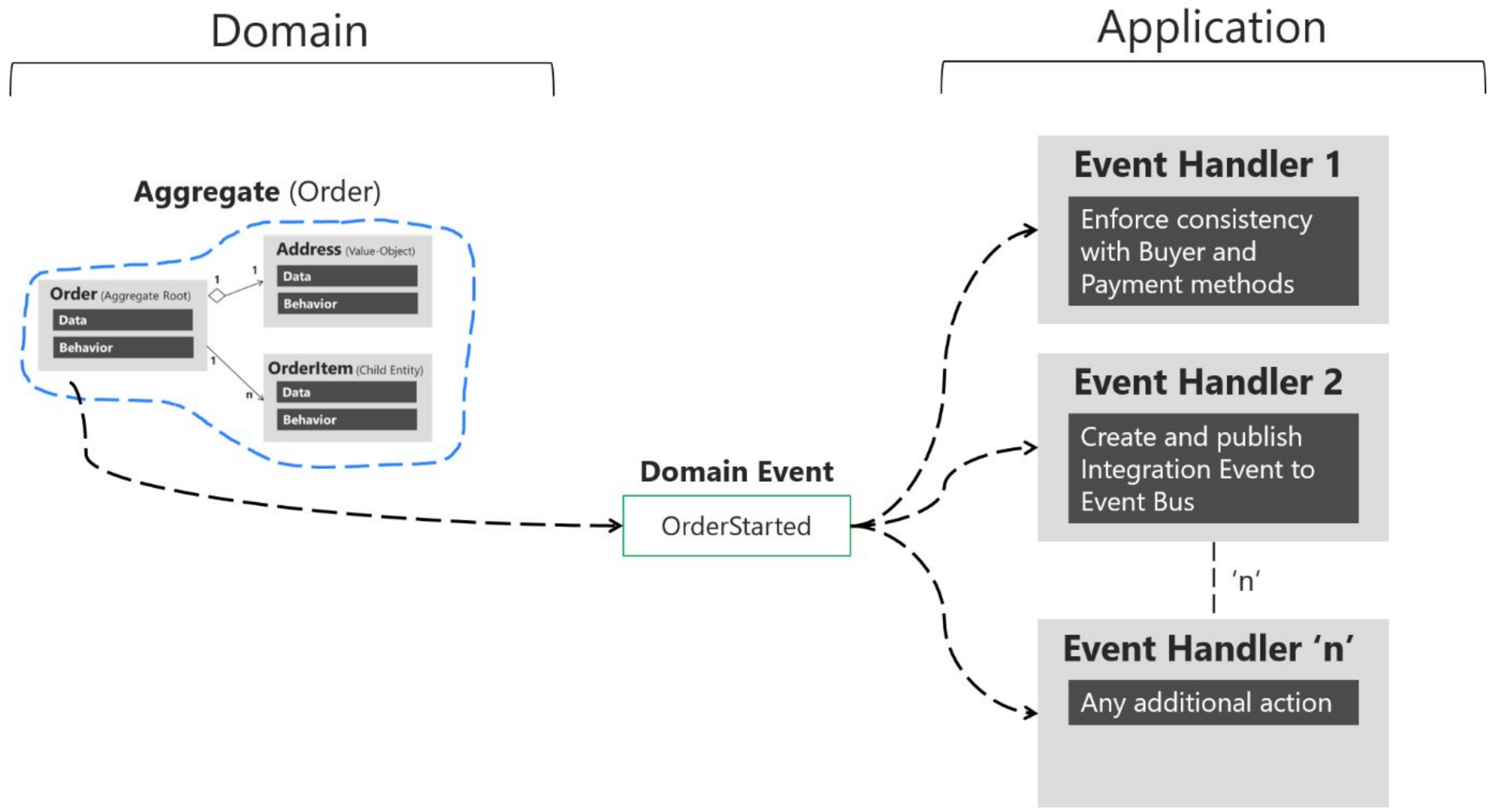
Patterns for decomposing an application into services



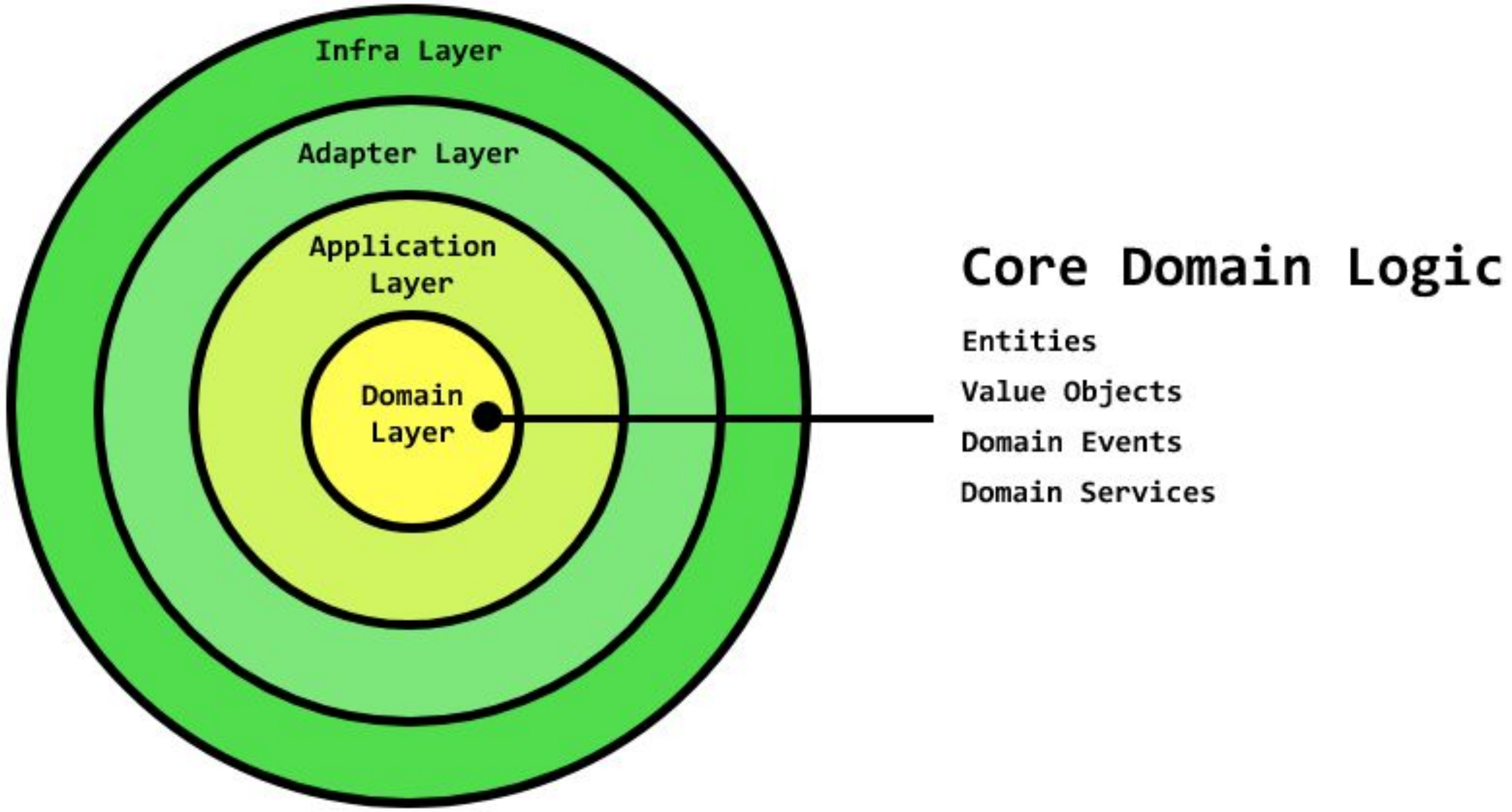
Domain Events



Domain Events



DDD Layers

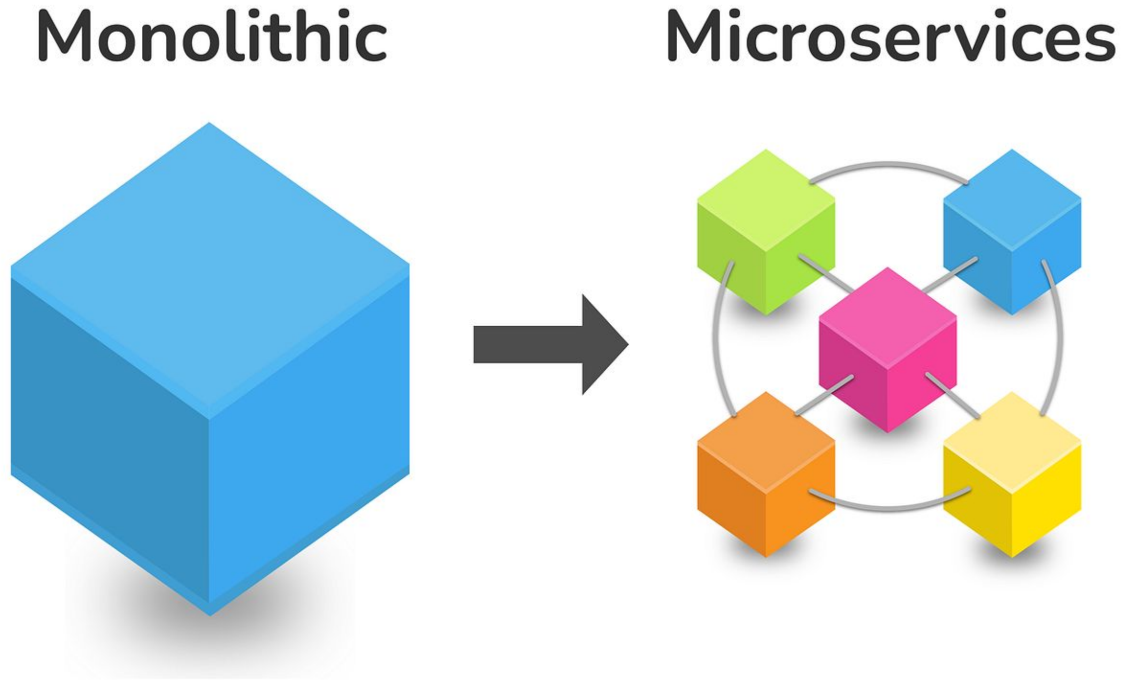

Infra Layer
Adapter Layer
Application Layer
Domain Layer
Core Domain Logic
Entities
Value Objects
Domain Events
Domain Services

Three steps to defining an application's microservice architecture

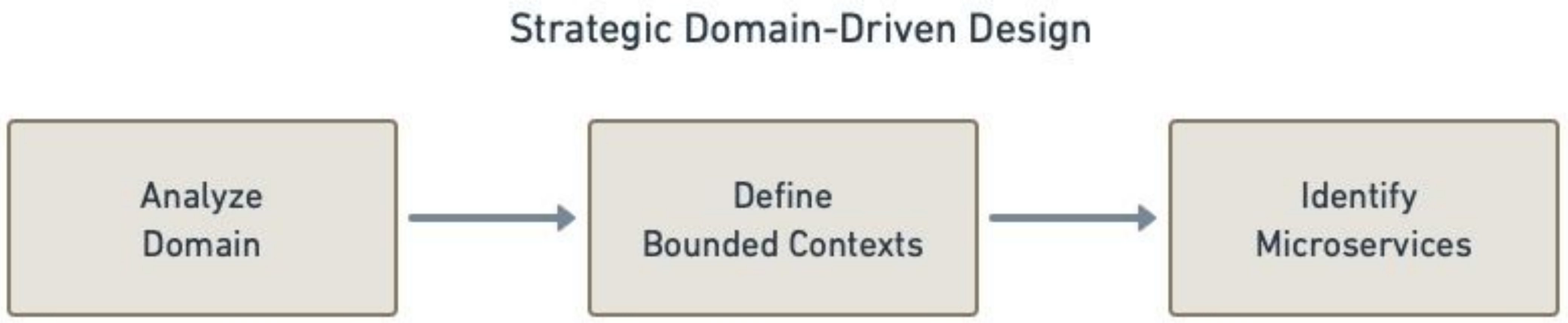
- Identify system operations
- Identify services
- Define service APIs and collaborations

Microservices Dilemma

- Monolith first vs. Microservices first



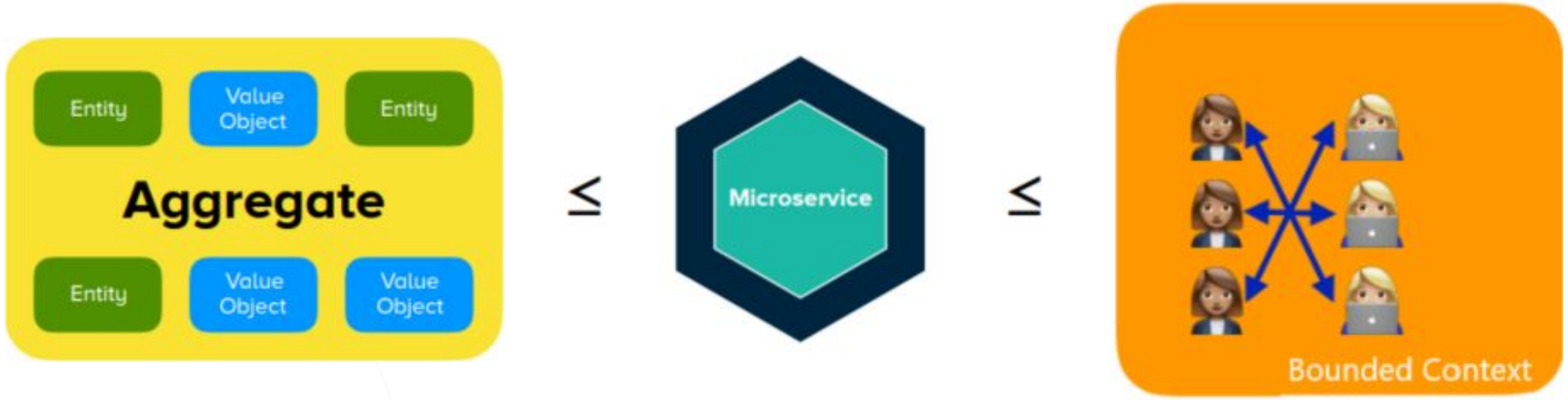
Strategic DDD



DDD & Microservices

- Apply strategic DDD **to identify microservices** (bounded context, ubiquitous language, context map)
- Apply tactical DDD **to design individual services** (aggregator, value object, service)

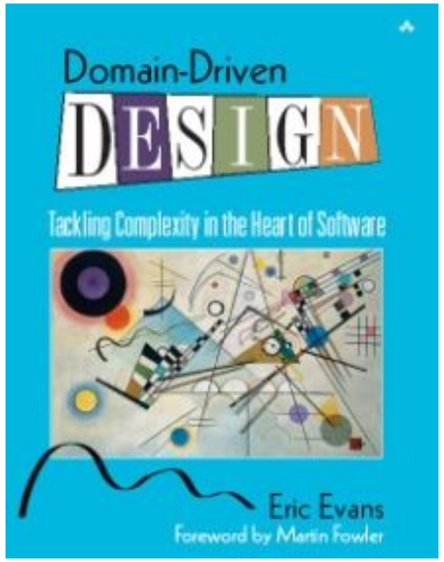
What is the right size of a service in the microservice architecture?



The Quote of the Day

The heart of software is its ability to solve domain-related problems for its user.

Eric Evans


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

Readings

- ➢ Domain-Driven Design: Tackling Complexity in the Heart of Software, Eric Evans, 1st Edition, 2003.
- ➢ Complementary video at the following address: https://www.aparat.com/v/VGuTK