Container Bootcamp

Microservice Bounded Context

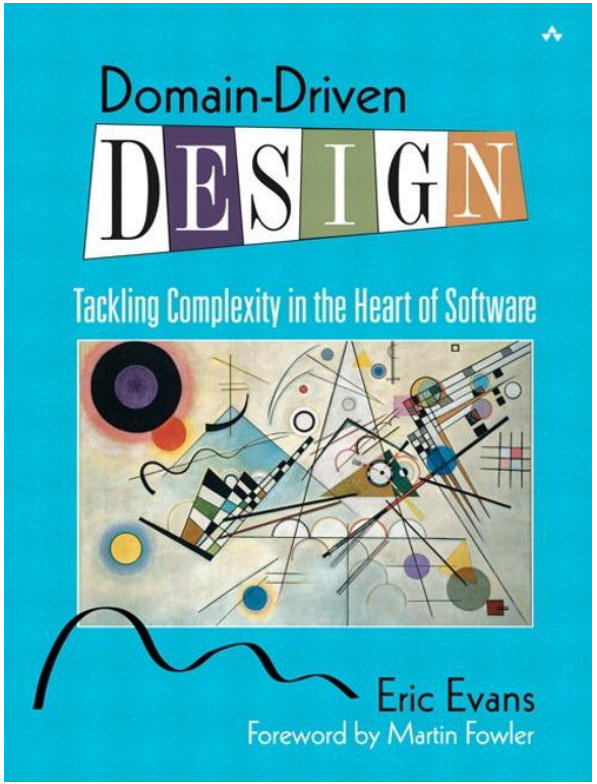
INNOQ

Domain-Driven Design

2004

Still very relevant

By Eric Evans



Free reference: http://domainlanguage.com/ddd/reference/

Domain Driven Design

- System decomposition along business domains
- Advantages
  - Better communication between business experts and developers
  - Business Experts do not have to grasp technical details
  - Congruence of business and technical model

Domain Driven Design

- Ubiquitous Language
- Common lingo for all business and technical staff
- Code / database / user should use the same terms

Entity

- Has its own identity i.e. customer
- Has its own lifecycle

Value Object

- Derive identity from their values
- Inherit lifecycle from Entities that reference it
- No identity on its own i.e. address
- Should be immutable

Aggregate

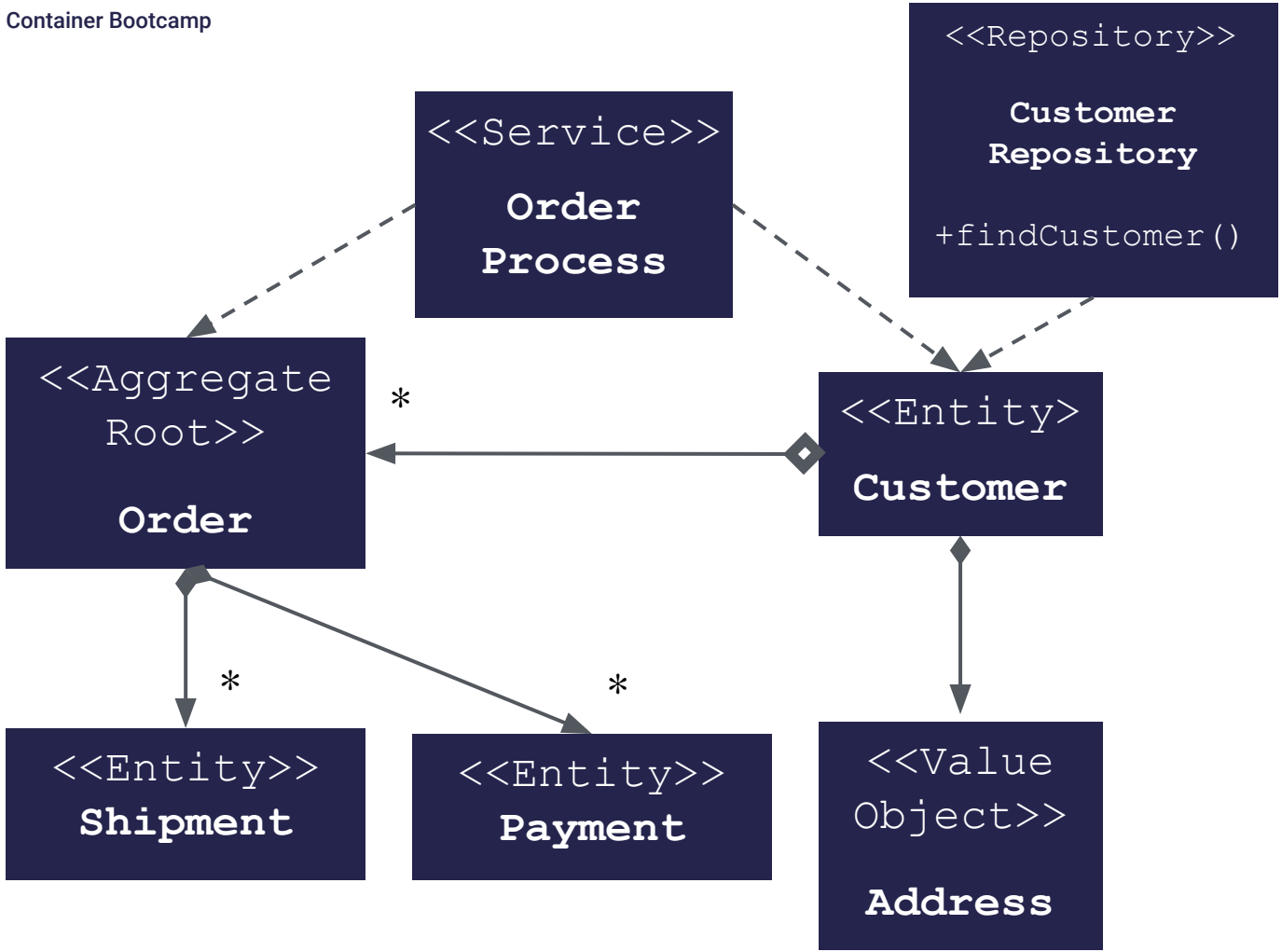
- Cluster of Entities or Value Objects
- Can enforce invariants and consistency
- Access only through the root (Entity)
- References only to the root
- Common lifecycle
- Example: Order (including order positions, addresses etc)

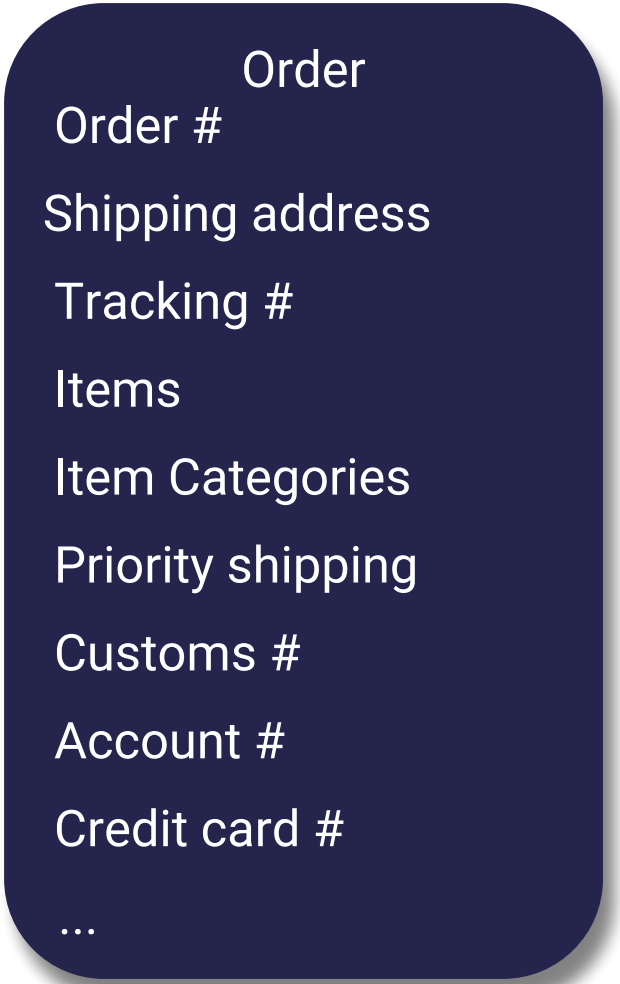
Service

- Process or transformation
- Not a natural responsibility of an Entity or Value Object
- Stateless

Repository

- Only for Aggregates that need global access
- Illusion of in-memory store
- Add & remove objects
- Queries based on criteria


<<Repository>>
Customer Repository
+findCustomer()
<<Service>>
Order Process
<<Aggregate Root>>
Order
*
<<Entity>>
Customer
*
*
<<Entity>>
Shipment
<<Entity>>
Payment
<<Value Object>>
Address


Order
Order #
Shipping address
Tracking #
Items
Item Categories
Priority shipping
Customs #
Account #
Credit card #
...

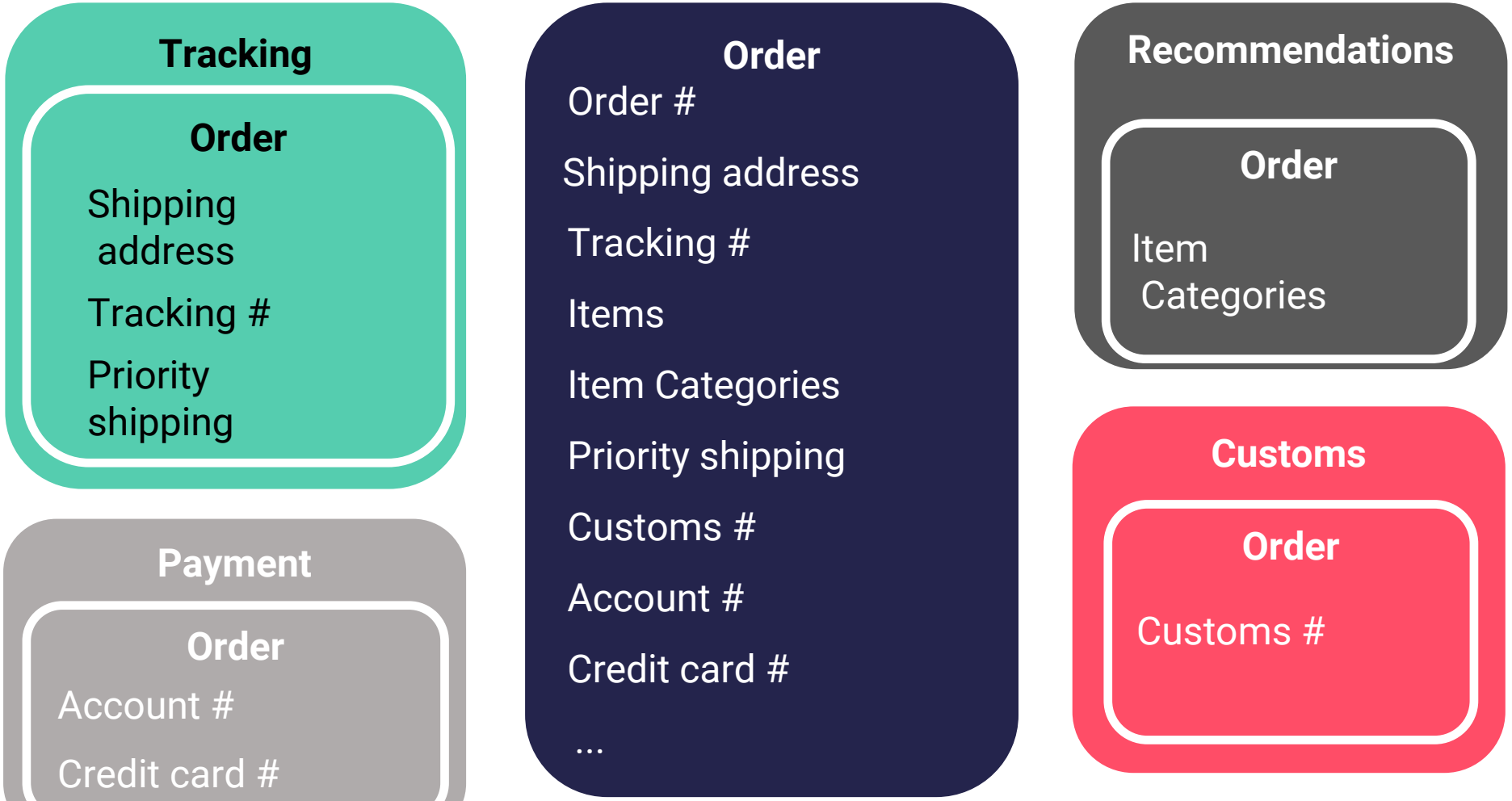
My Domain Model is a mess!

Order is the core of the business – should be possible to come up with a proper model!

Bounded Context

- Domain model is only valid for one context

- There is no universal data model!
- See all failed SOA attempts


Tracking
Order
Shipping address
Tracking #
Priority shipping
Payment
Order
Account #
Credit card #
Order
Order #
Shipping address
Tracking #
Items
Item Categories
Priority shipping
Customs #
Account #
Credit card #
...
Recommendations
Order
Item Categories
Customs
Order
Customs #

Integration: Strategic Design

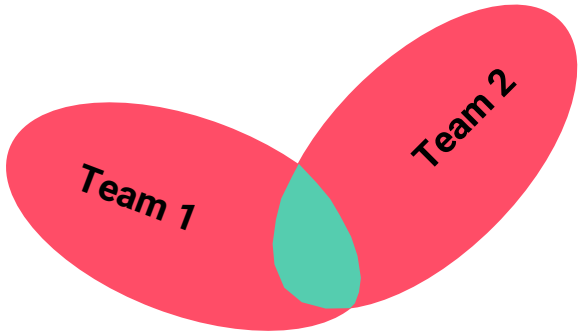
Strategic Design

- A domain model is only valid in a Bounded Context.
- A Bounded Context should be implemented in a deployable unit.

- How do Bounded Contexts relate to each other?
- Context can have relationships

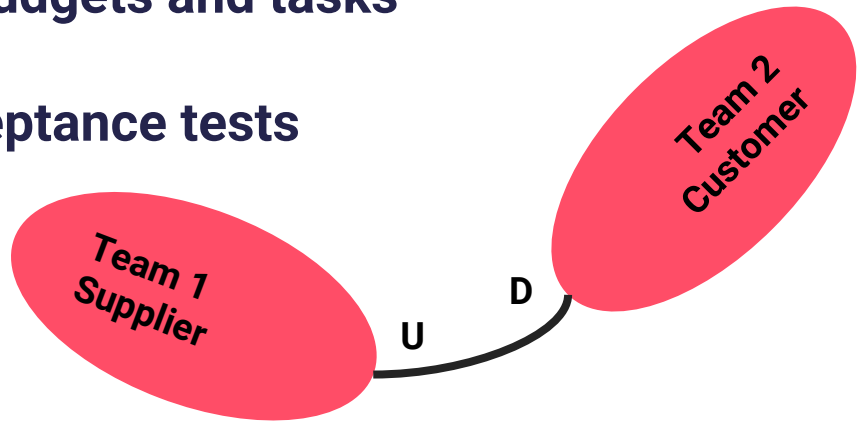
Shared Kernel

- Subset of a model
- …that two teams share
- Including code and database
- Tests by both teams



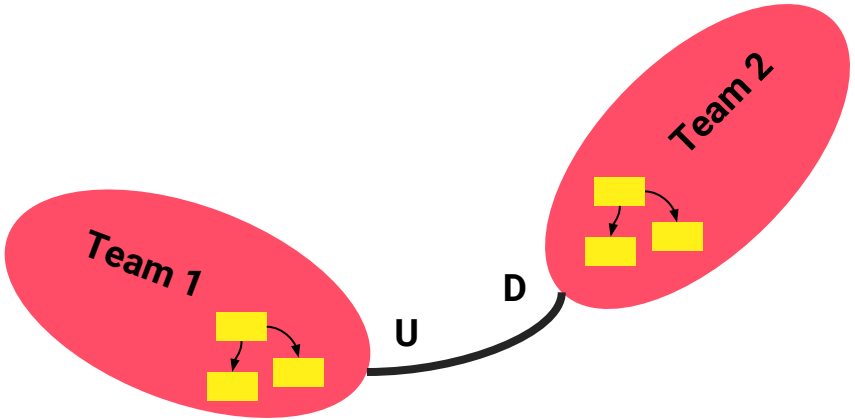
Customer / Supplier

- Upstream team provides model
- Downstream team = customer for upstream team
- Downstream team negotiates budgets and tasks
- Jointly develop automated acceptance tests



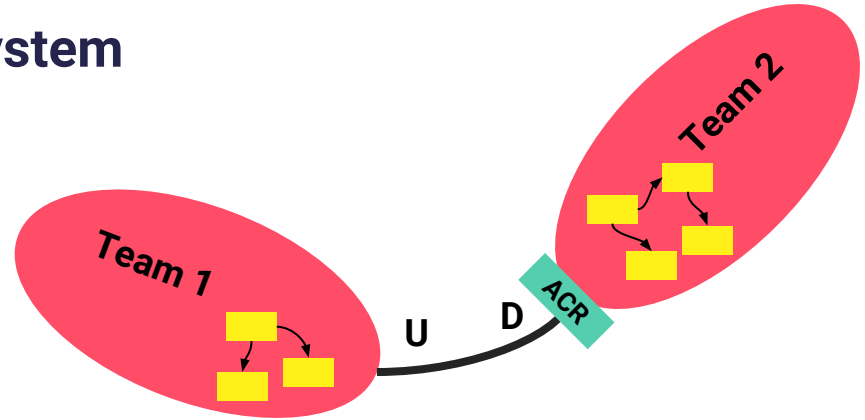
Conformist

- Follow the upstream team
- Simplifies integration
- Easier communication



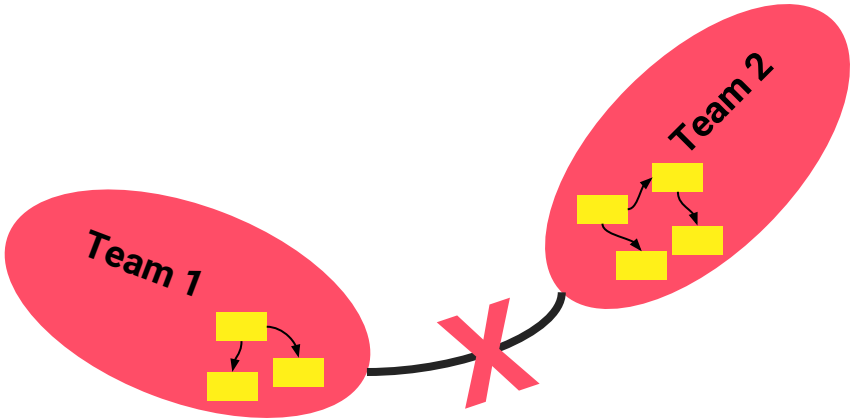
Anticorruption Layer

- Don’t let e.g. a legacy model influence a new model
- Isolate model by additional layer
- No need to modify the old system



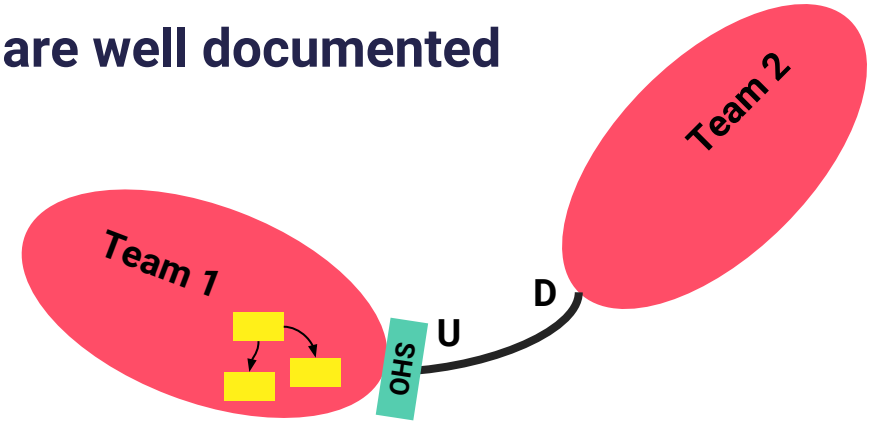
Separate Ways

- Bounded Context has no connection to the others
- Allows for simple, specialized solutions in a small scope



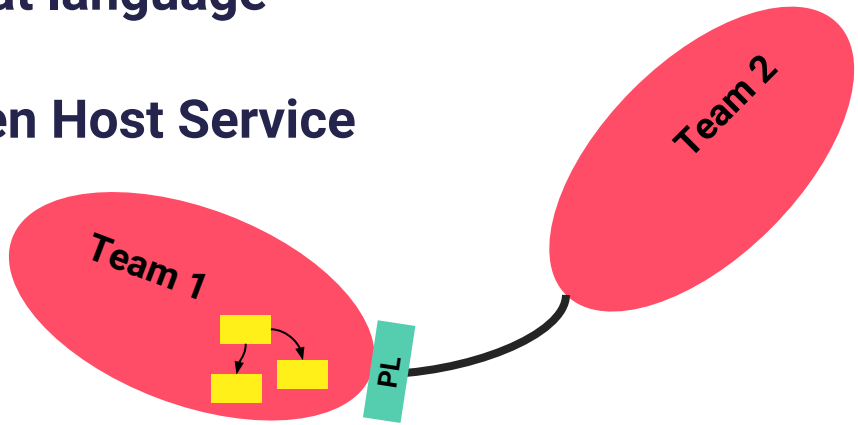
Open Host Service

- Define protocol or interface as a set of services
- Build integrations using the Services
- The services offered by API are well documented



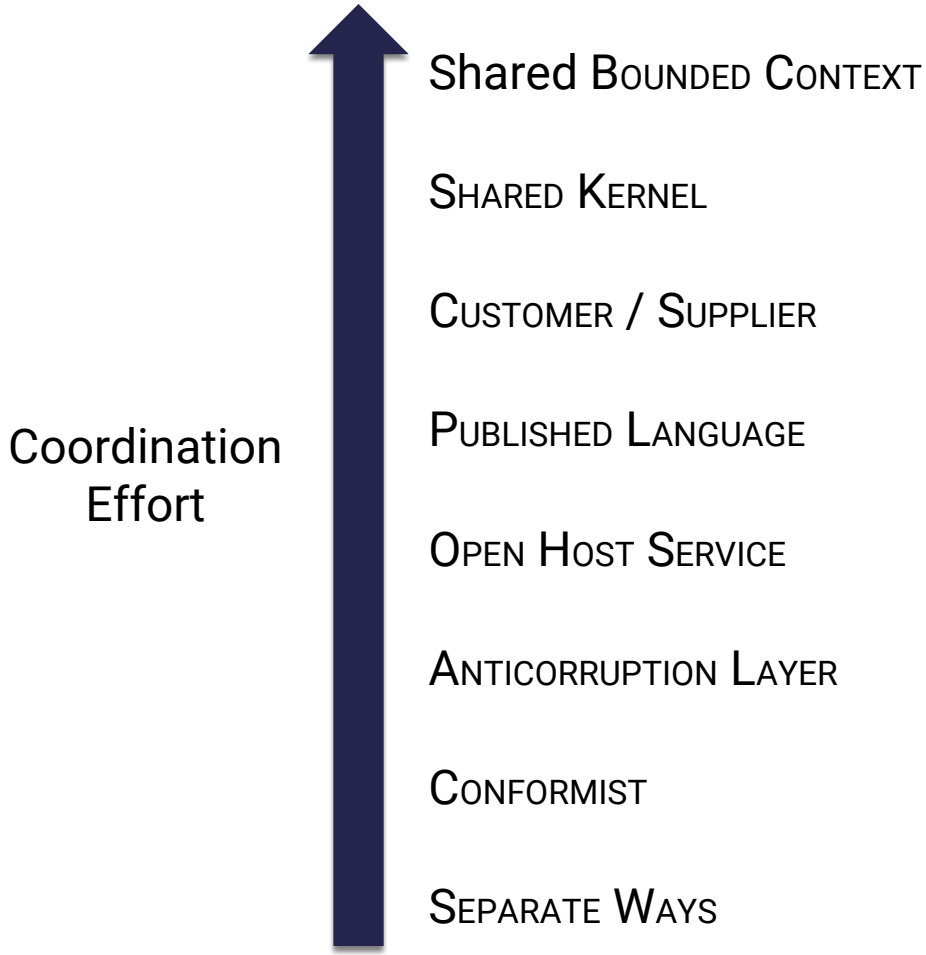
Published Language

- Well-documented shared language (e.g. XML Schema, Avro)
- Accessible to anyone
- Can translate into and out of that language
- Not just used internally like Open Host Service



Context Relationships

- Team = Deployment Unit = Bounded Context
- Context Relationships define how Bounded Context are used…
- …and how much teams need to collaborate

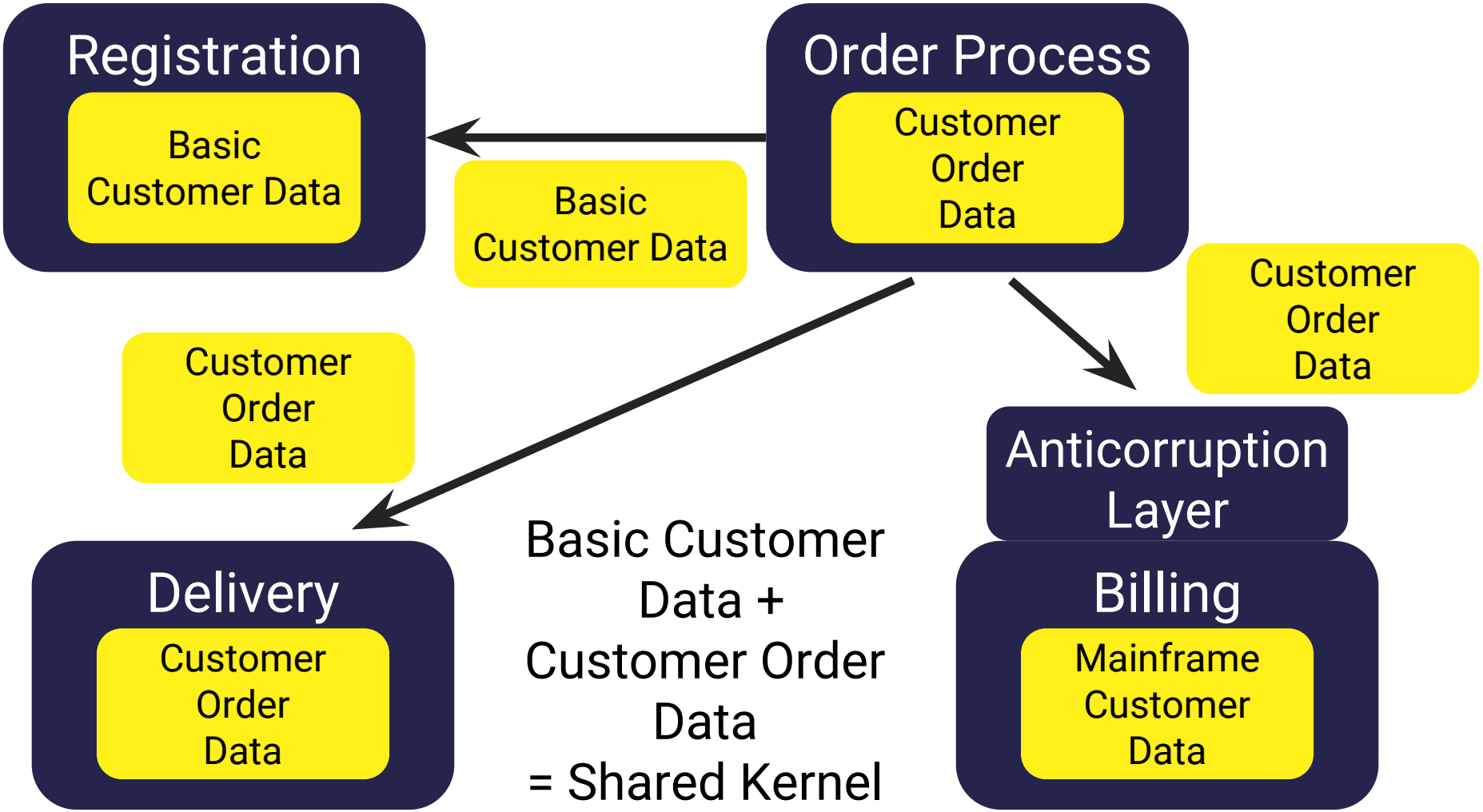

Coordination Effort
Shared Bounded Context
Shared Kernel
Customer / Supplier
Published Language
Open Host Service
Anticorruption Layer
Conformist
Separate Ways

Context Map

Context Map

- Show the different Bounded Context
- …and the relation to each other

- Bounded Context might be Services
- ...or communication links


Registration
Basic Customer Data
Order Process
Customer Order Data
Basic Customer Data
Customer Order Data
Customer Order Data
Anticorruption Layer
Delivery
Customer Order Data
Billing
Mainframe Customer Data
Basic Customer Data + Customer Order Data = Shared Kernel

Large Scale Structure

- Ideas for the overall shape of the system
- Optional

- System Metaphor: For the overall system
- E.g. customer journey for E-commerce

Large Scale Structure

- Responsibility Layer: Upper layer may call lower layer
- Not technical
- i.e. Catalog -> Order Process -> Billing

- Evolving Order: Let the overall structure evolve
- …as you learn about the domain

Bounded Context & Microservice

- Domain architecture important
- Influences organization (Conway’s Law)
- …and efficiency / communication
- Bounded Context helps to get it right
- Helps to define large-scale architecture, too