Microservices

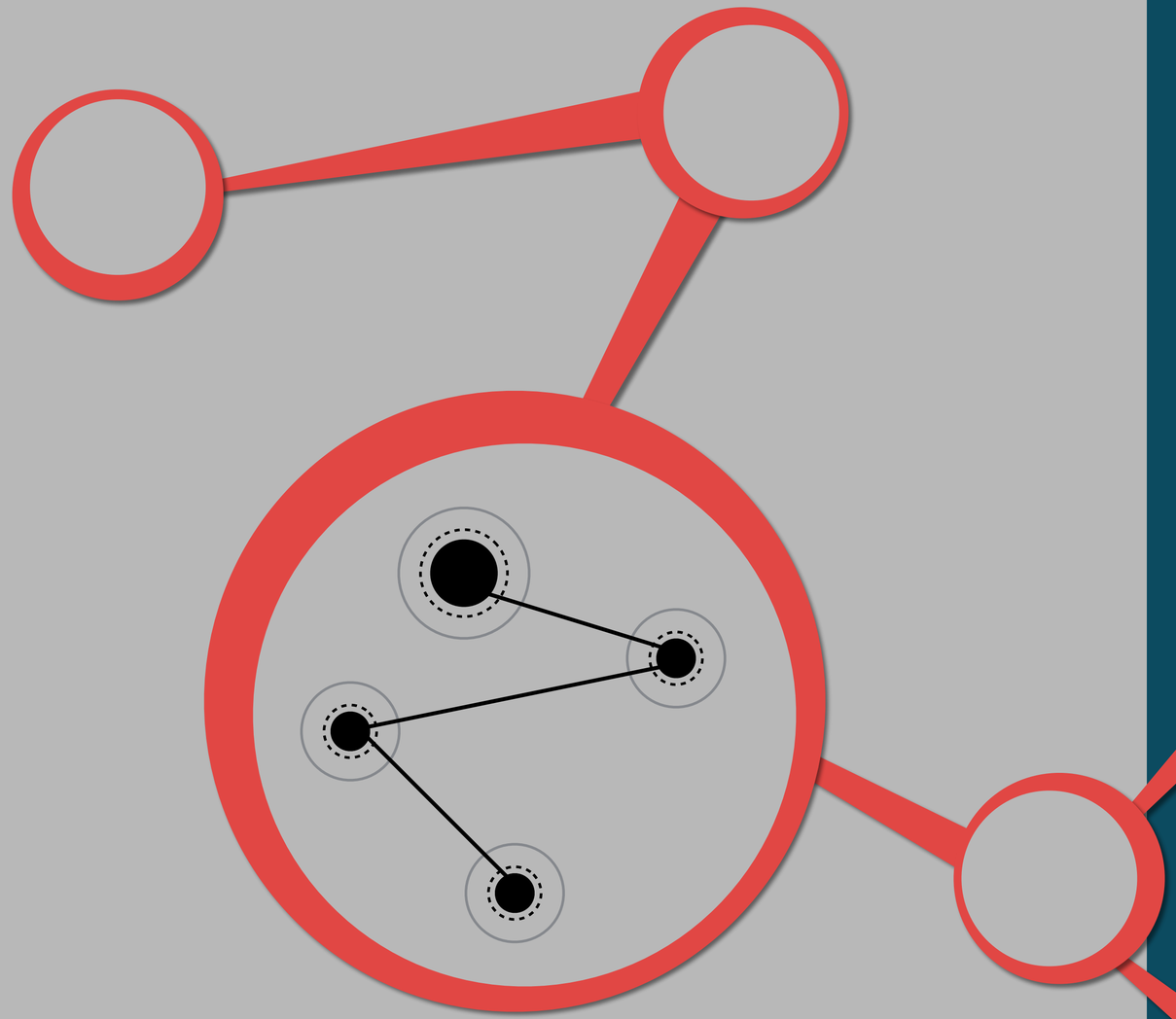
Domain Driven Design

Michael Plöd - innoQ
@bitboss

Disclaimer

Most of these ideas do not come from me personally. I have to thank Eric Evans and Vaughn Vernon for all the inspiration / ideas. If you haven’t: go out and get their amazing books!

Michael Plöd - innoQ
@bitboss



D D D

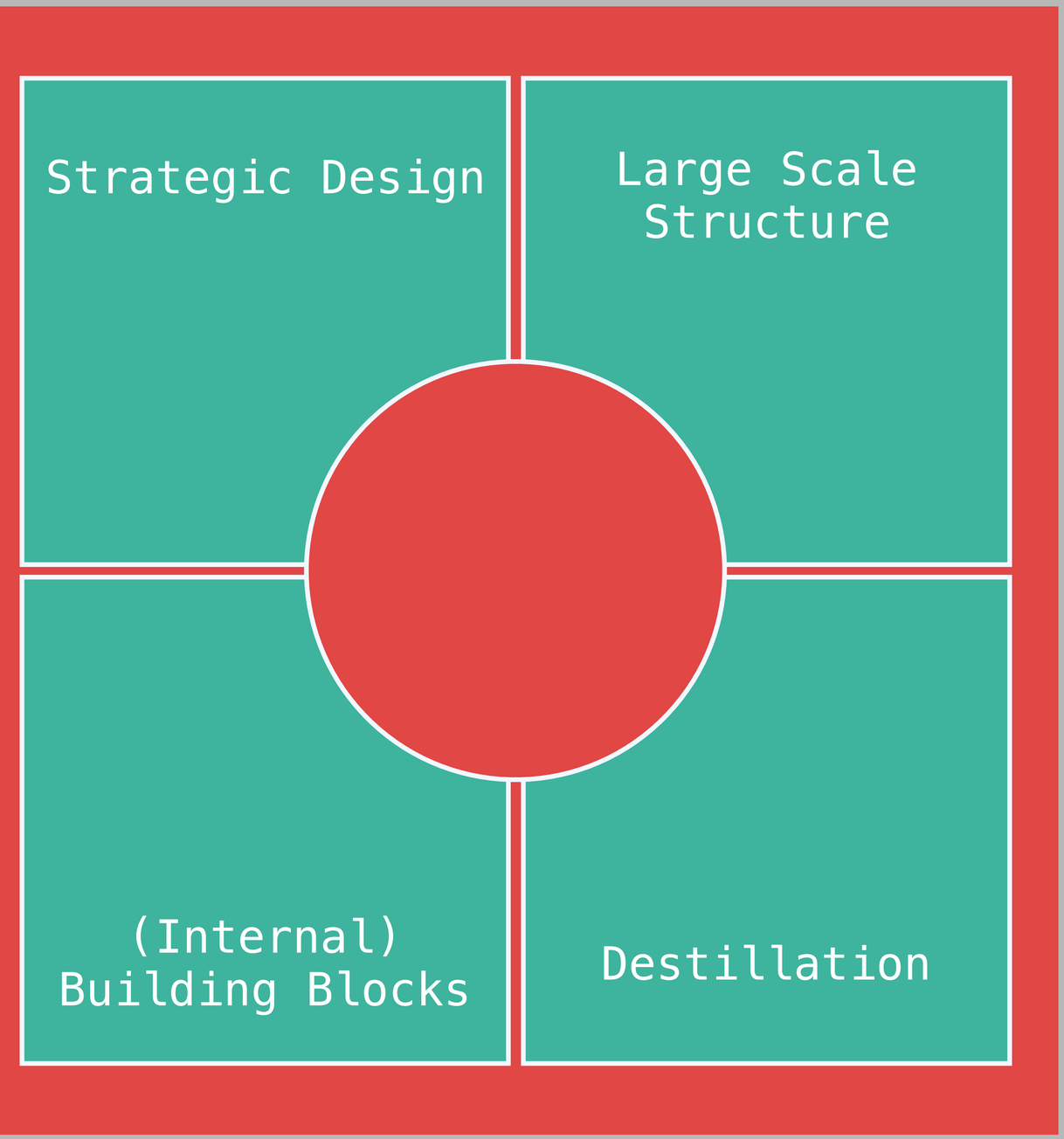
in Microservices

DDD and Microservices are not just about Bounded Contexts

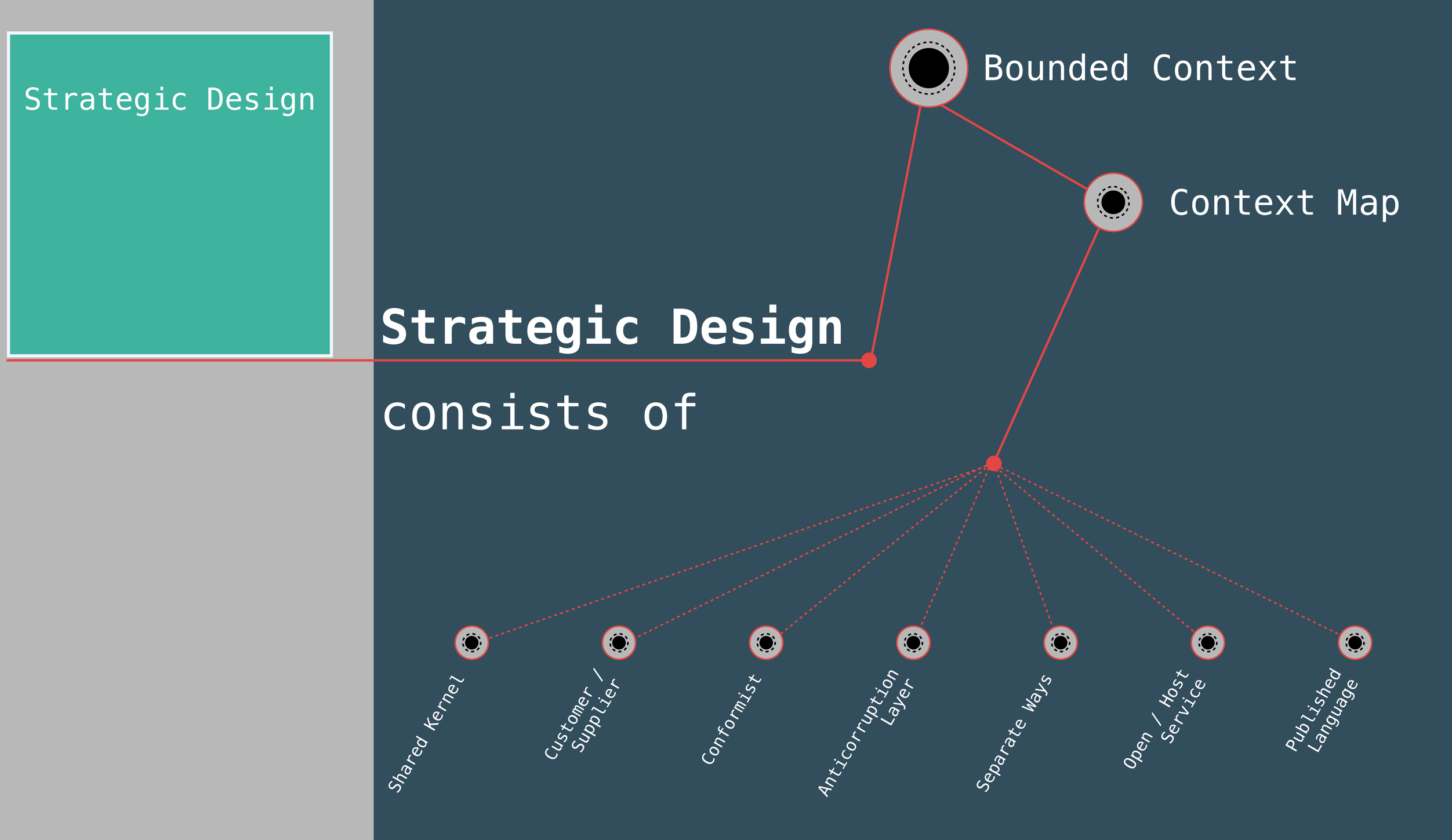
DDD itself is not just about Aggregates, Entities and Services

Domain Driven Design

helps us with Microservices in four areas

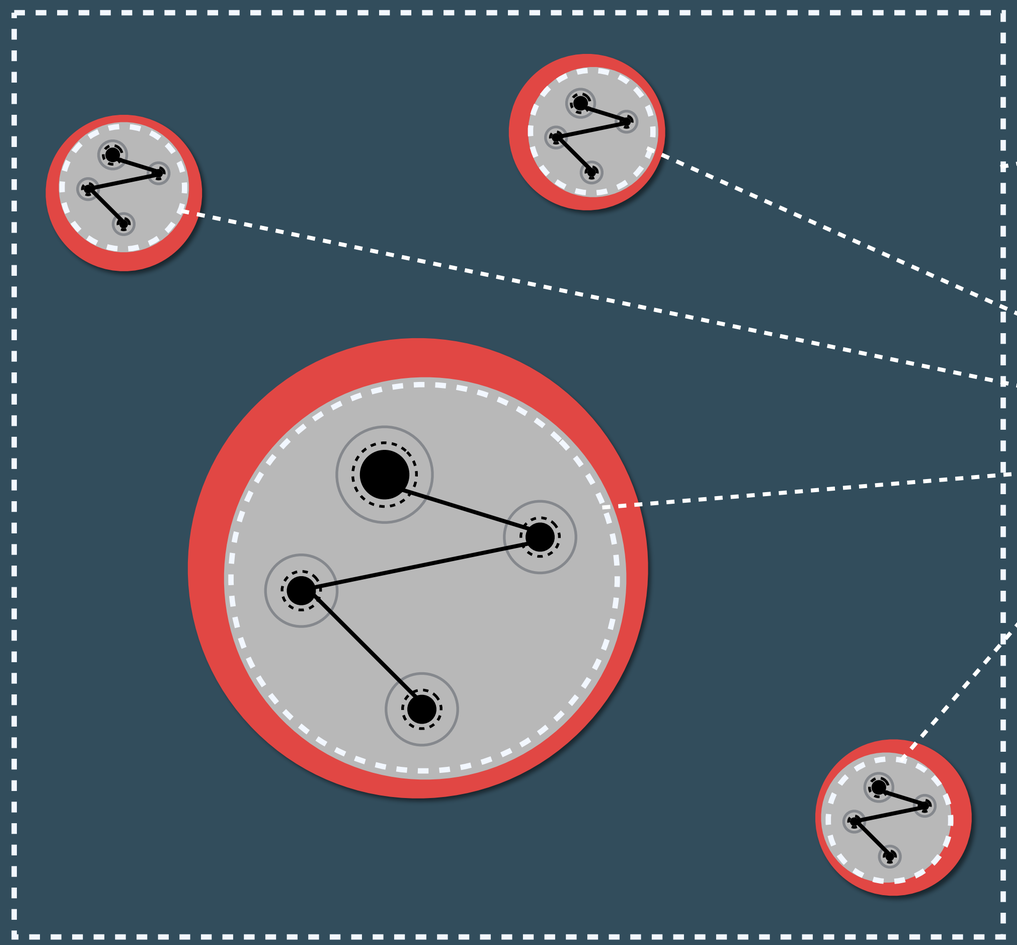


Strategic Design



Strategic Design

Bounded Context



Every sophisticated business domain consists of a bunch of **Bounded Contexts**

Each **Bounded Context** contains models and maybe other contexts

The **Bounded Context** is also a boundary for the meaning of a given model

Strategic Design

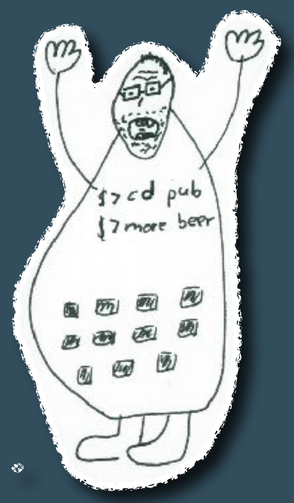
Bounded Context Example

Reservations

Event Management

Badges

Customer



Name

Payment Details

Address

Company

Session Registrations

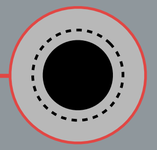
Lunch Preferences

Name

Job Description

Twitter Handle

Strategic Design



Bounded Context Example

Each Bounded Context has its own model of a customer

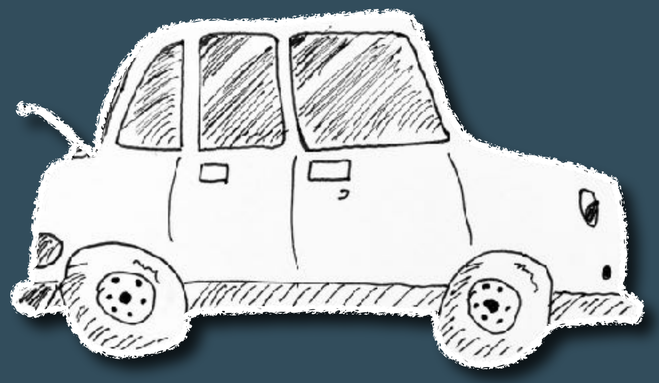
This is a major enabler for independent Microservices

Take a look at the name of the customer? Maybe we want some shared data?

Strategic Design

Bounded Context Example

Car



Personal Driving

Service Center

Think about the differences in starting the car or simple components such as ABS, ESP, engine or infotainment

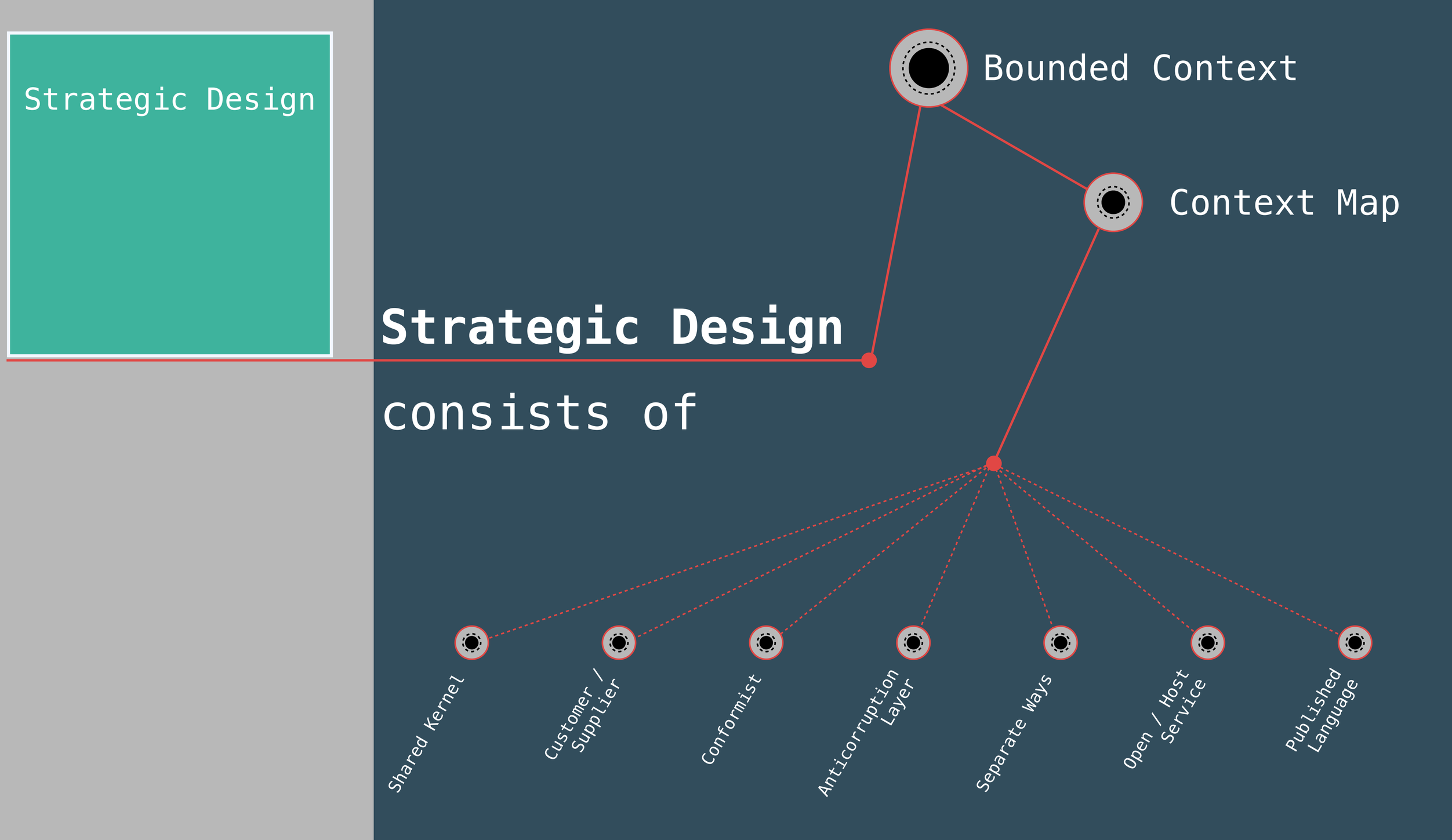
Strategic Design

How to identify Bounded ContextS?

Factors

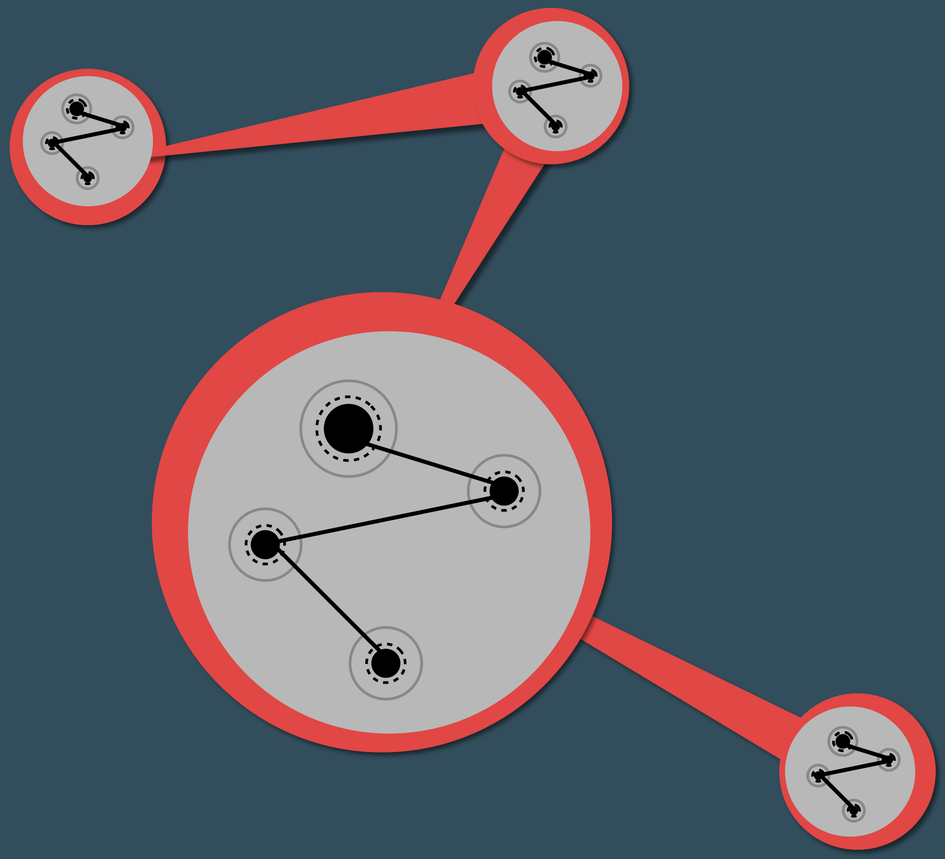
One Team	If a Bounded Context must be managed or implemented by more than one team it is probably too big and should be split up.
Meaningful Model	Try to identify models that make sense and that are meaningful in one specific context. Also think about decoupling of models.
Cohesion	Take a look at your (sub-) domain and think about which parts of that domain are strongly related or in other words highly cohesive.
Language	Models act as an Ubiquitous Language, therefore it is necessary to draw a line between Contexts when the project langeuage changes.

Strategic Design



Strategic Design

Context Map



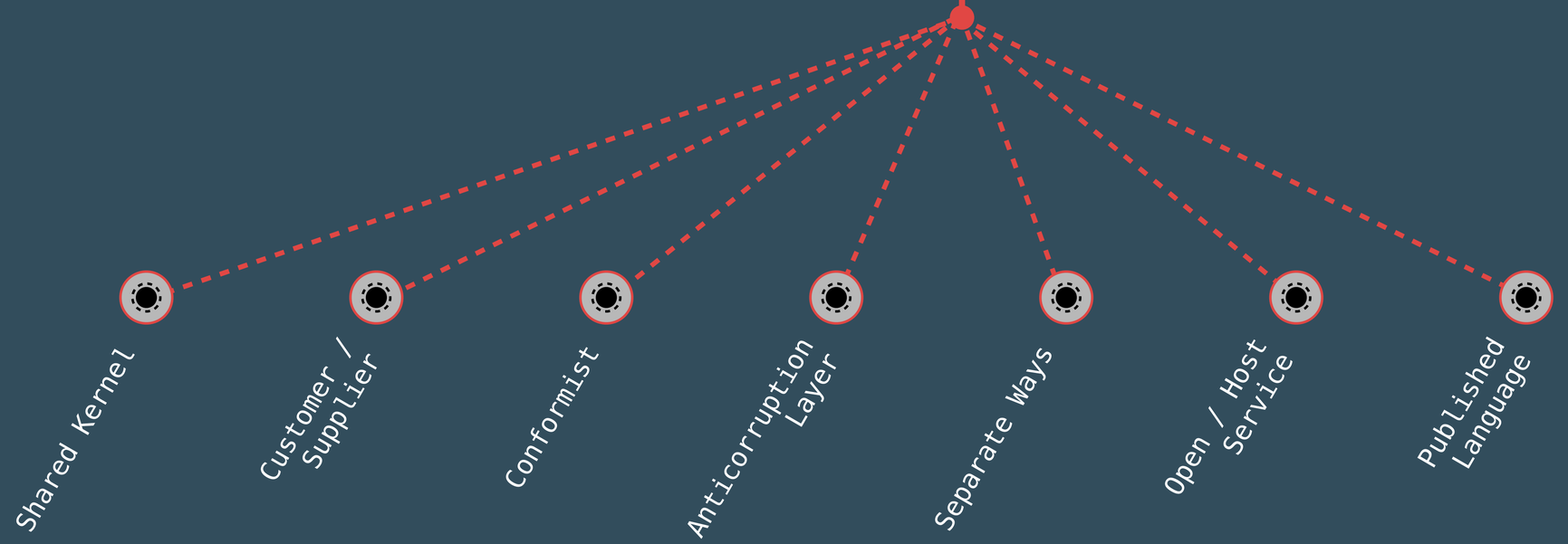
The Bounded Context by itself does not deliver an overview of the system

By introducing a **Context Map** we describe the contact between models / contexts

The **Context Map** is also a great starting point for future transformations

Strategic Design

Context Map – Patterns

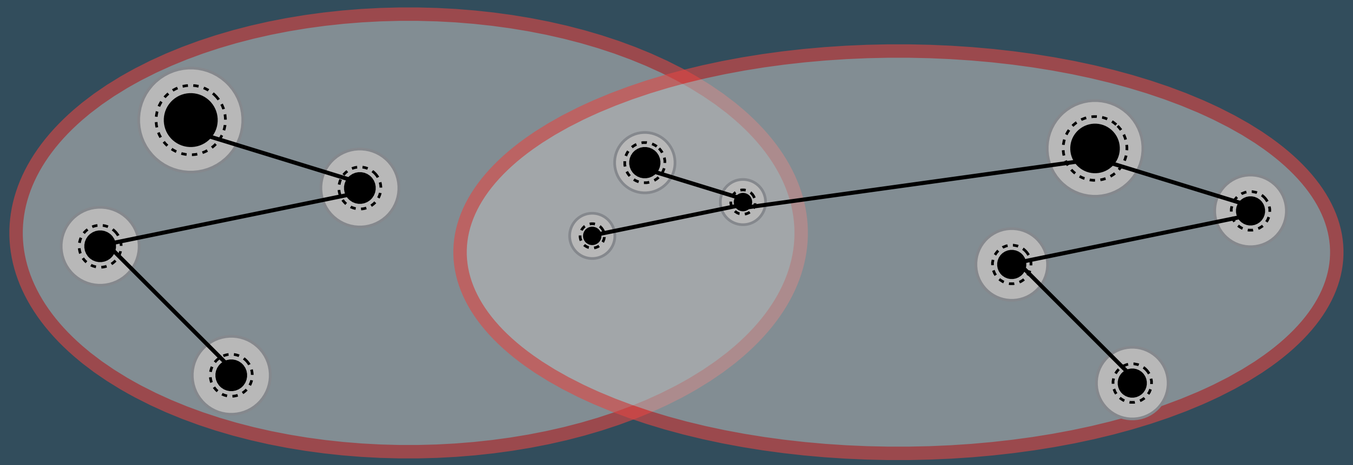


Strategic Design

Context Map – Patterns

- Shared Kernel
- Customer / Supplier
- Conformist
- Anticorruption Layer
- Separate Ways
- Open / Host Service
- Published Language

Two teams share a subset of the domain model including code and maybe the database. The shared kernel is often refered to as the core domain.

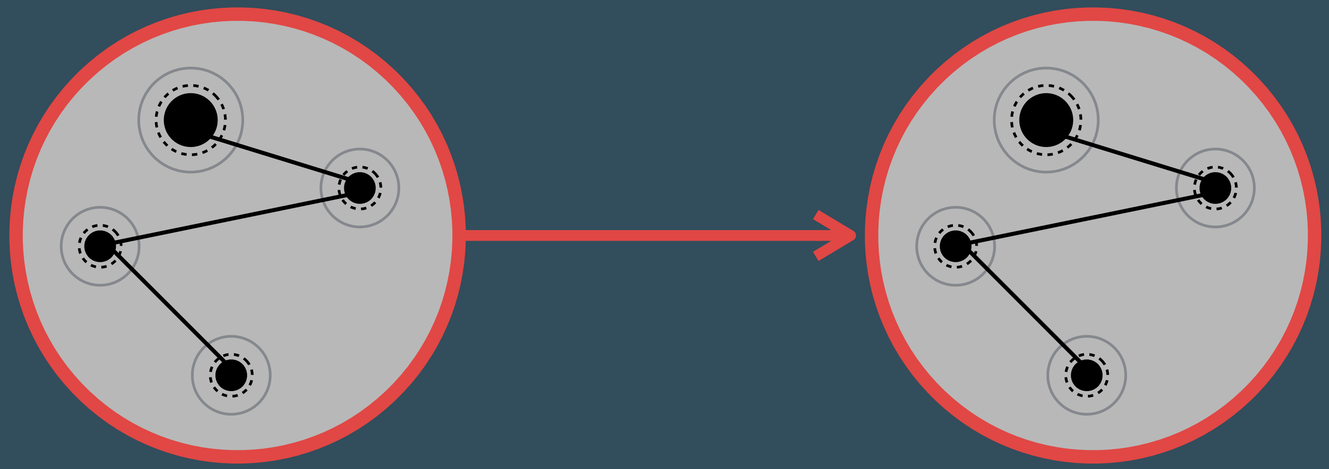


Strategic Design

Context Map – Patterns

- Shared Kernel
- Customer / Supplier
- Conformist
- Anticorruption Layer
- Separate Ways
- Open / Host Service
- Published Language

There is a customer / supplier relation ship between two teams. The downstream team is considered to be the customer, sometimes with veto rights.

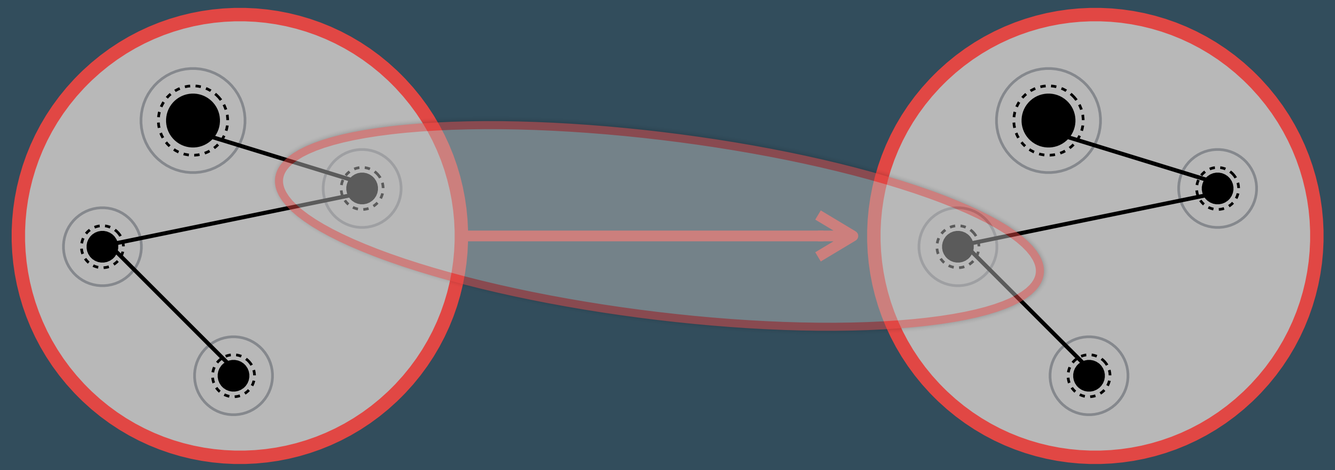


Strategic Design

Context Map – Patterns

- Shared Kernel
- Customer / Supplier
- Conformist
- Anticorruption Layer
- Separate Ways
- Open / Host Service
- Published Language

The downstream team conforms to the model of the upstream team. There is no translation of models and no vetoing. If the upstream model is a mess, it propagates to the downstream model.

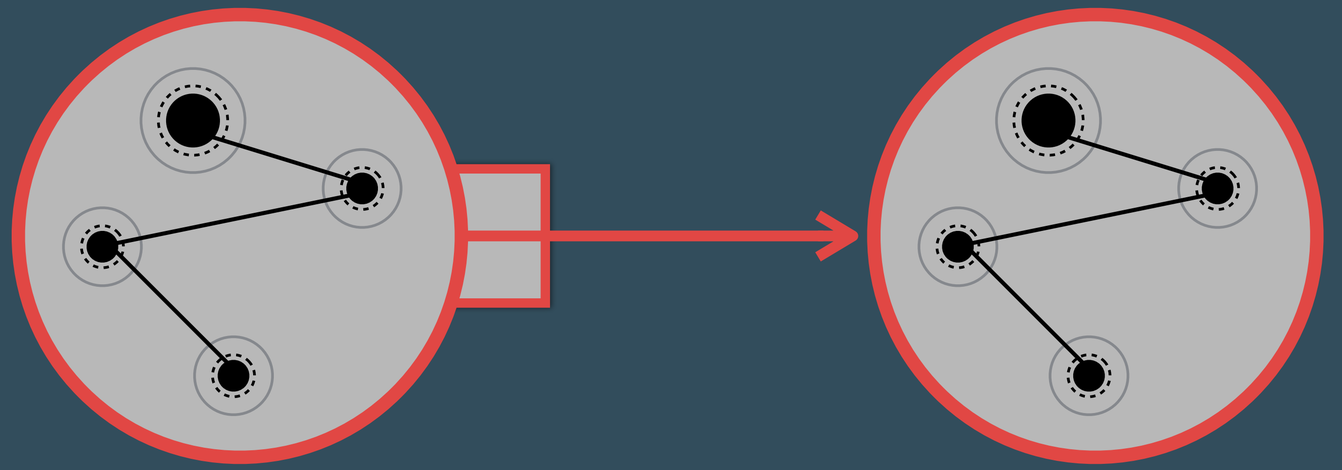


Strategic Design

Context Map – Patterns

- Shared Kernel
- Customer / Supplier
- Conformist
- Anticorruption Layer
- Separate Ways
- Open / Host Service
- Published Language

The anticorruption layer is a layer that isolates a client’s model from another system’s model by translation.



Strategic Design

Context Map – Patterns

- Shared Kernel
- Customer / Supplier
- Conformist
- Anticorruption Layer
- Separate Ways
- Open / Host Service
- Published Language

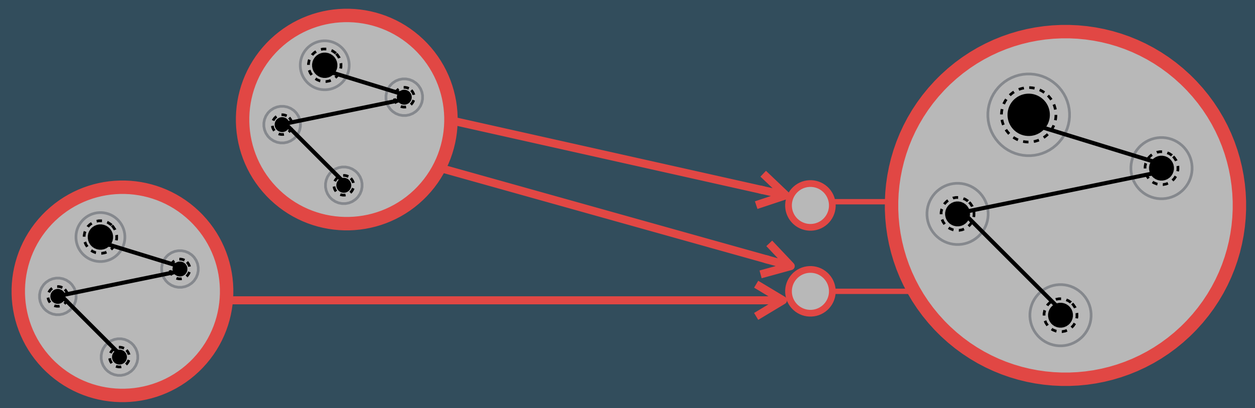
There is no connection between the bounded contexts of a system. This allows teams to find their own solutions in their domain.

Strategic Design

Context Map – Patterns

- Shared Kernel
- Customer / Supplier
- Conformist
- Anticorruption Layer
- Separate Ways
- Open / Host Service
- Published Language

Each Bounded Context offers a defined set of services that expose functionality for other systems. Any downstream system can then implement their own integration. This is especially useful for integration requirements with many other systems.

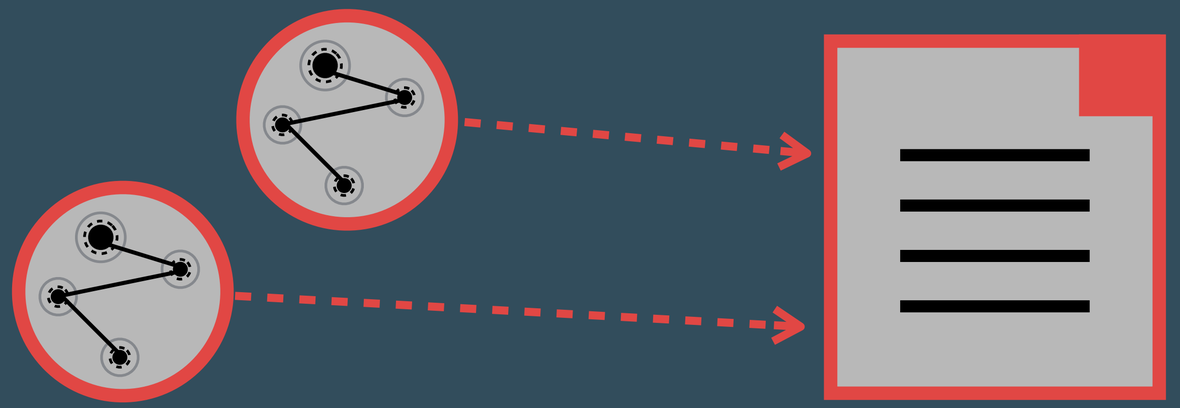


Strategic Design

Context Map – Patterns

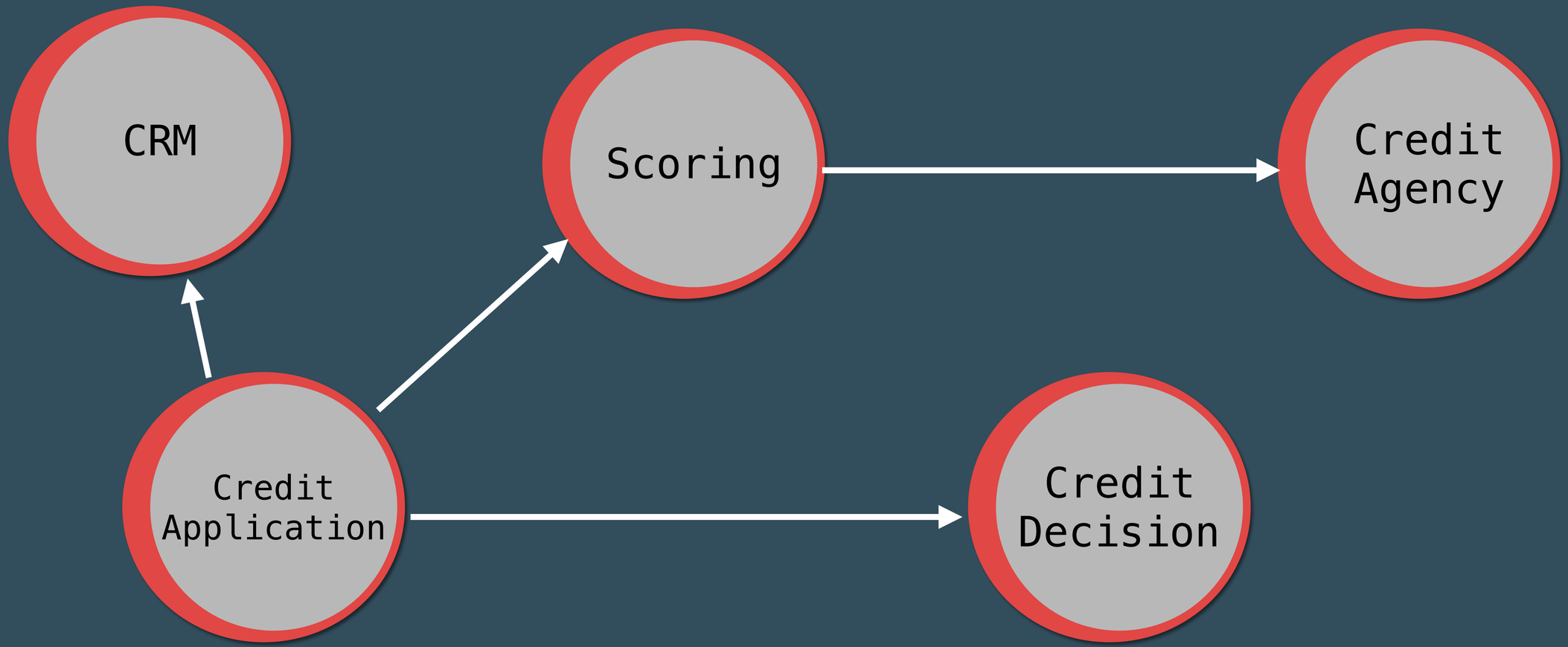
- Shared Kernel
- Customer / Supplier
- Conformist
- Anticorruption Layer
- Separate Ways
- Open / Host Service
- Published Language

Published Language is quite similar to Open / Host Service. However it goes as far as to model a Domain as a common language between bounded contexts.



Strategic Design

Context Map – Why?



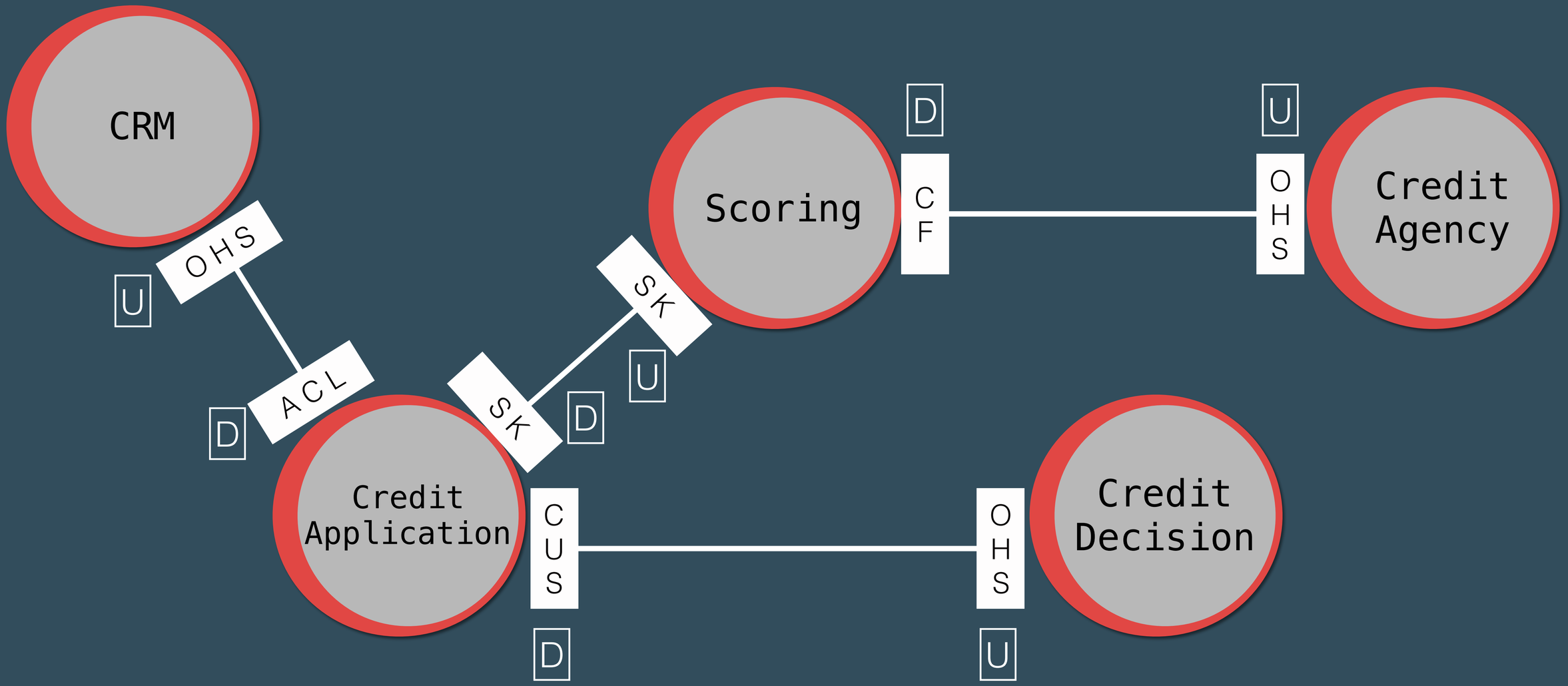
Strategic Design

Context Map – Why?

Currently we only see call stacks

Strategic Design

Context Map



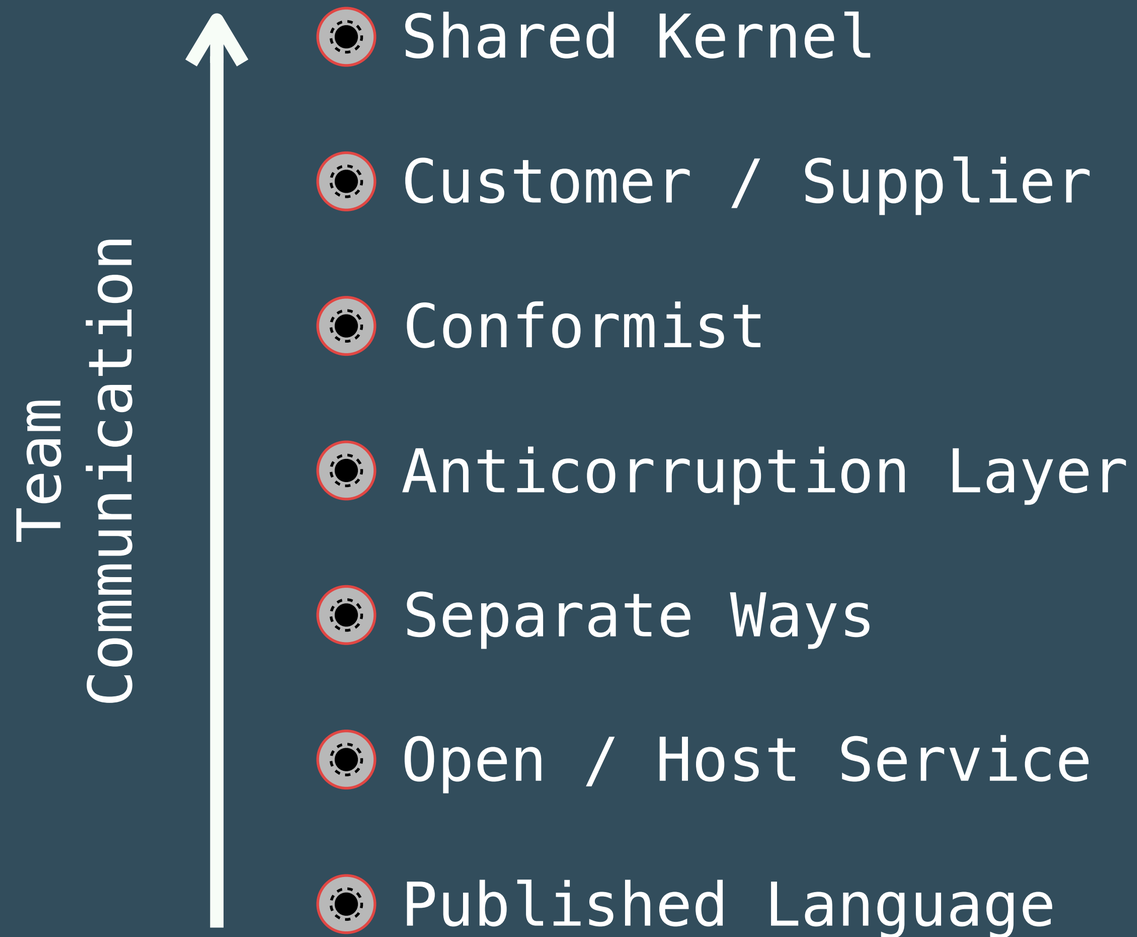
Strategic Design

and Conway’s Law

- Shared Kernel
- Customer / Supplier
- Conformist
- Anticorruption Layer
- Separate Ways
- Open / Host Service
- Published Language

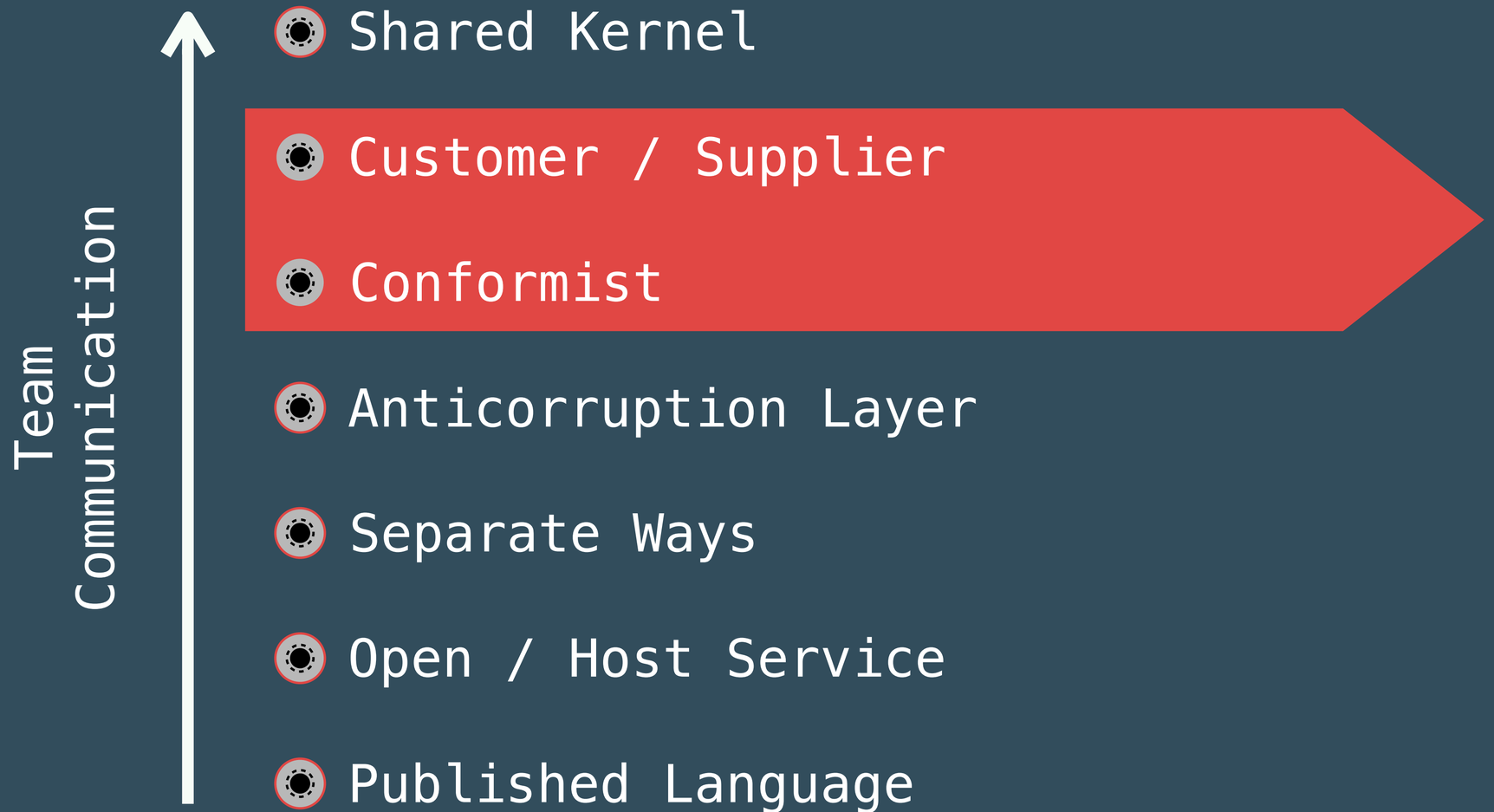
Strategic Design

and Conway's Law



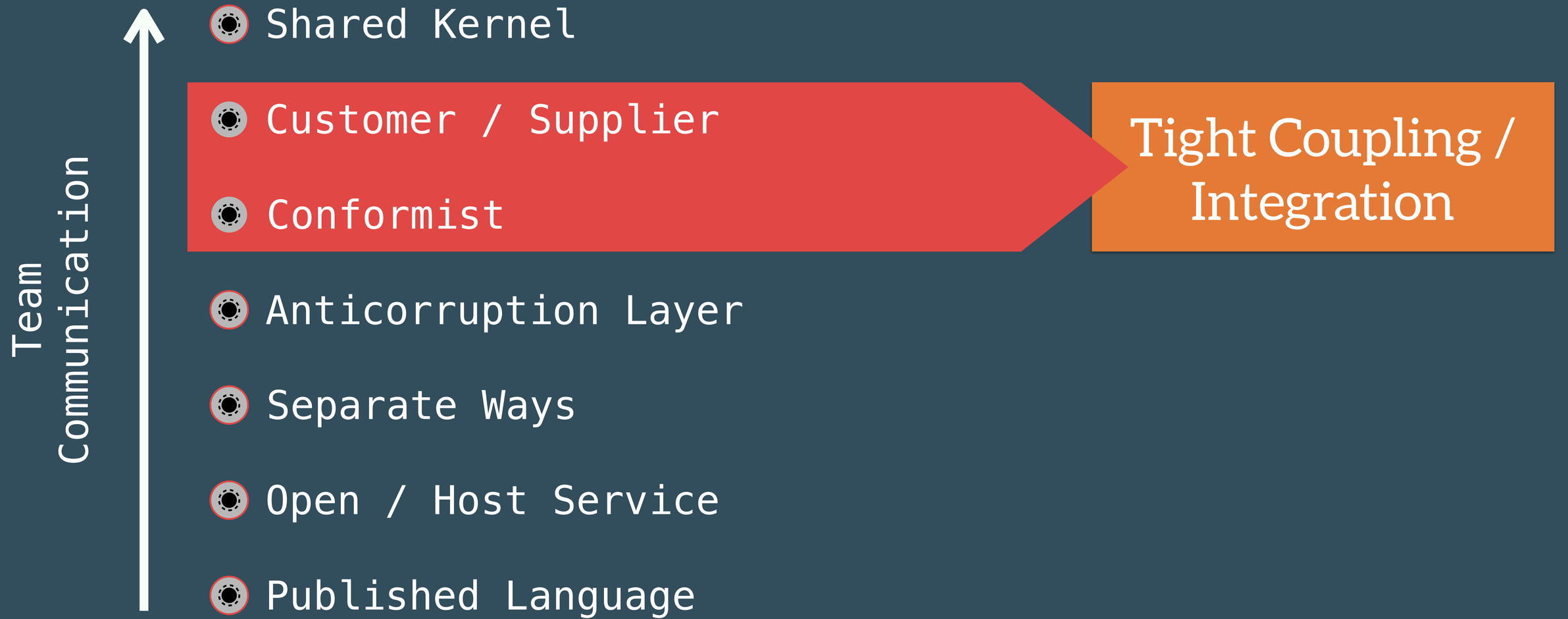
Strategic Design

and Conway's Law



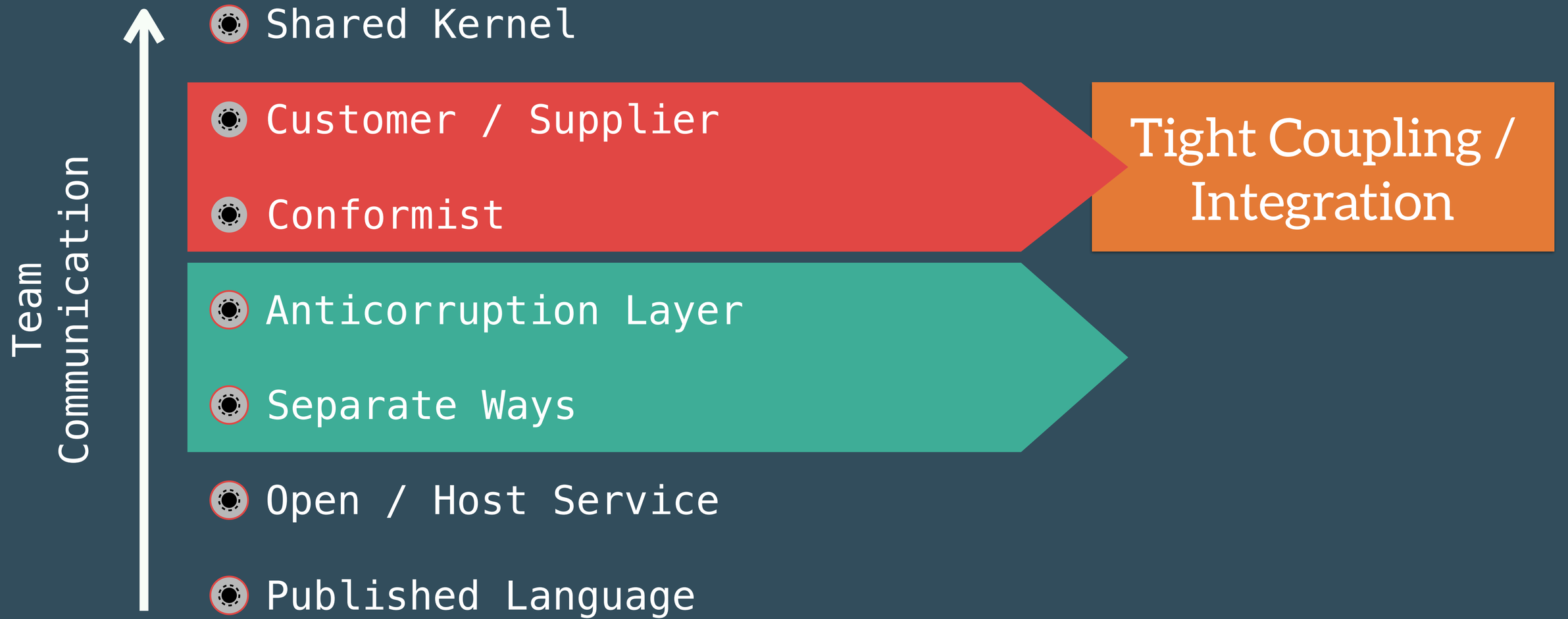
Strategic Design

and Conway's Law



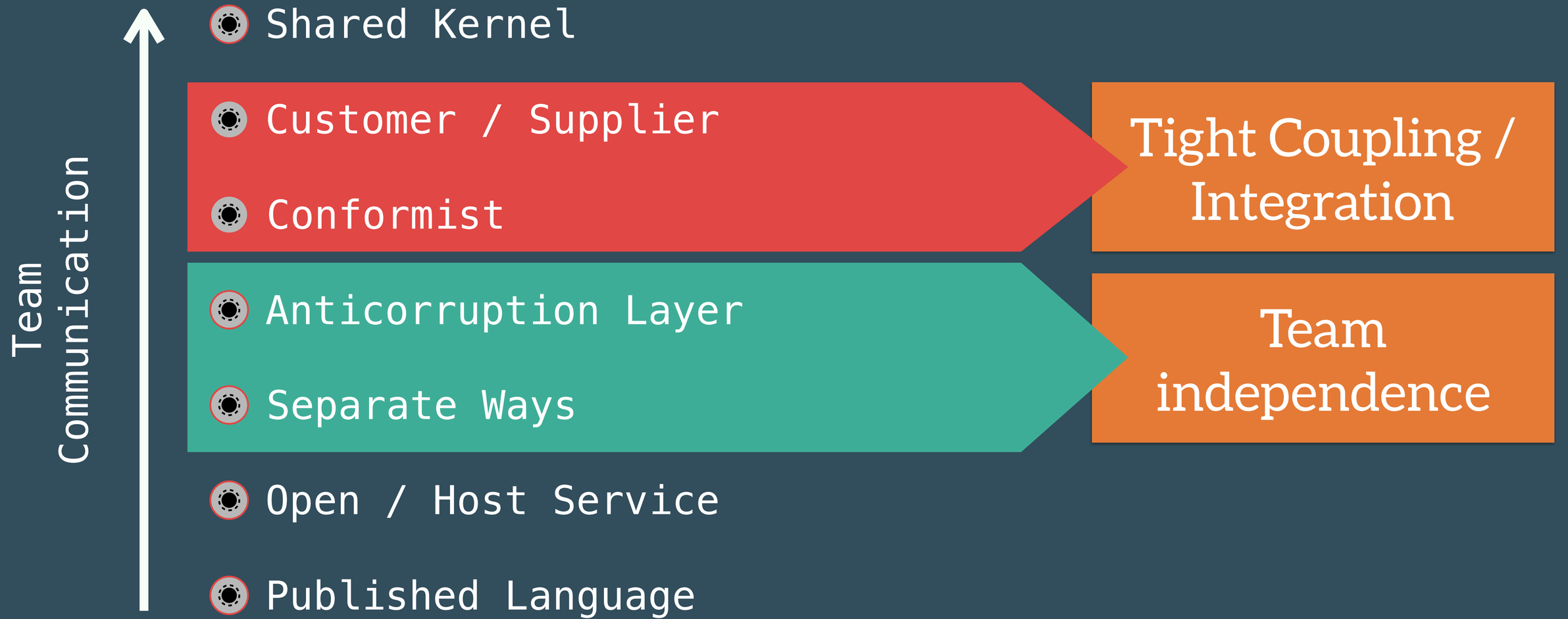
Strategic Design

and Conway's Law



Strategic Design

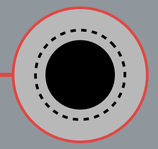
and Conway's Law



Strategic Design

Where do Context Maps help?

Strategic Design

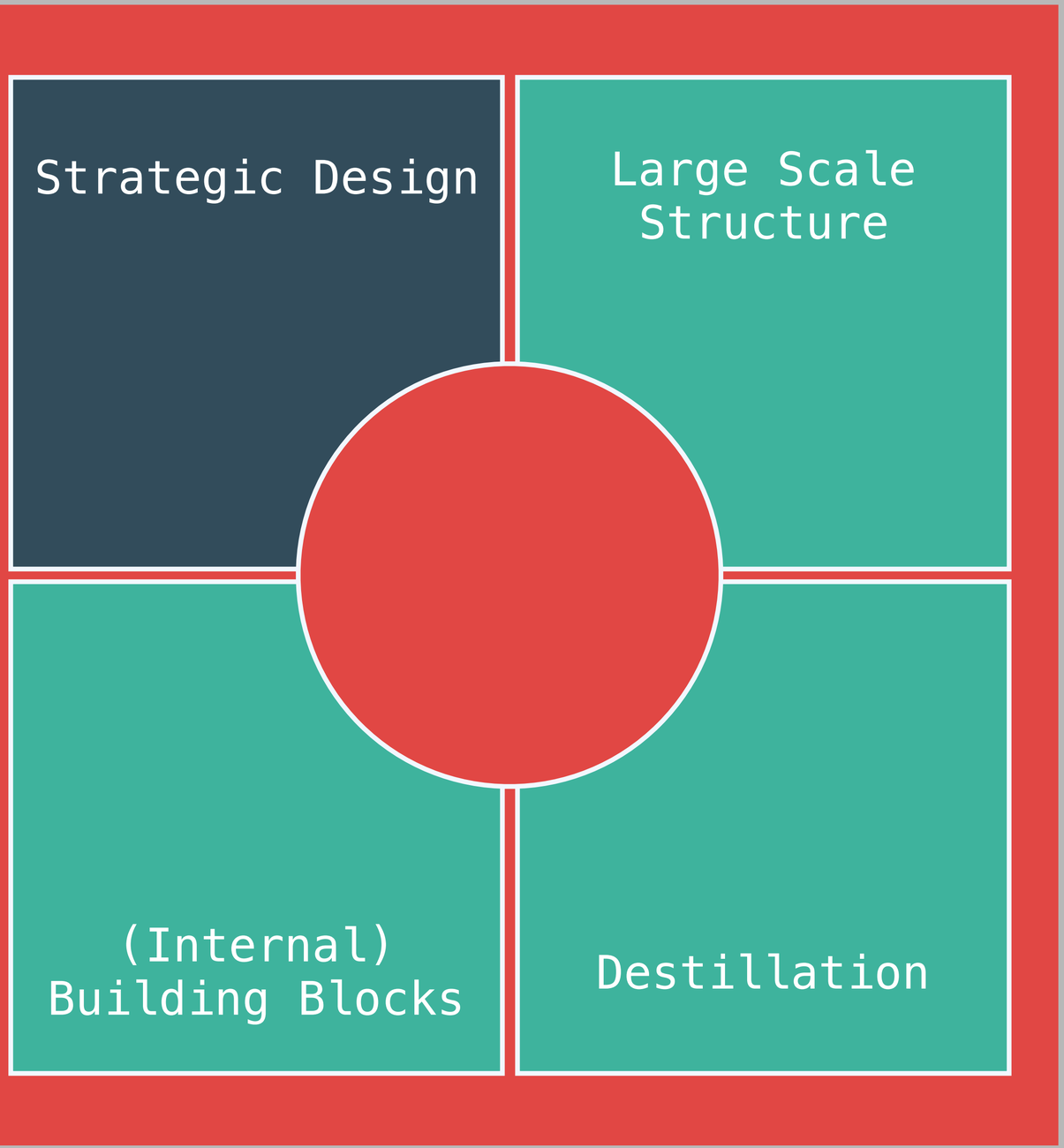


Where do Context Maps help?

Governance	A Context Map helps to identify governance issues between applications and teams.
Politics	By not just looking at technical integration aspects the Context Map also helps us in seeing how teams communicate and „play politics“.
Bad Models	By introducing a Context Map we get a clear view on where and how bad models propagate through application landscapes
Transformation	A Context Map is an excellent starting point for future transformations: it gives an in-depth insight into integration aspects and subdomain / context mathesxw

Domain Driven Design

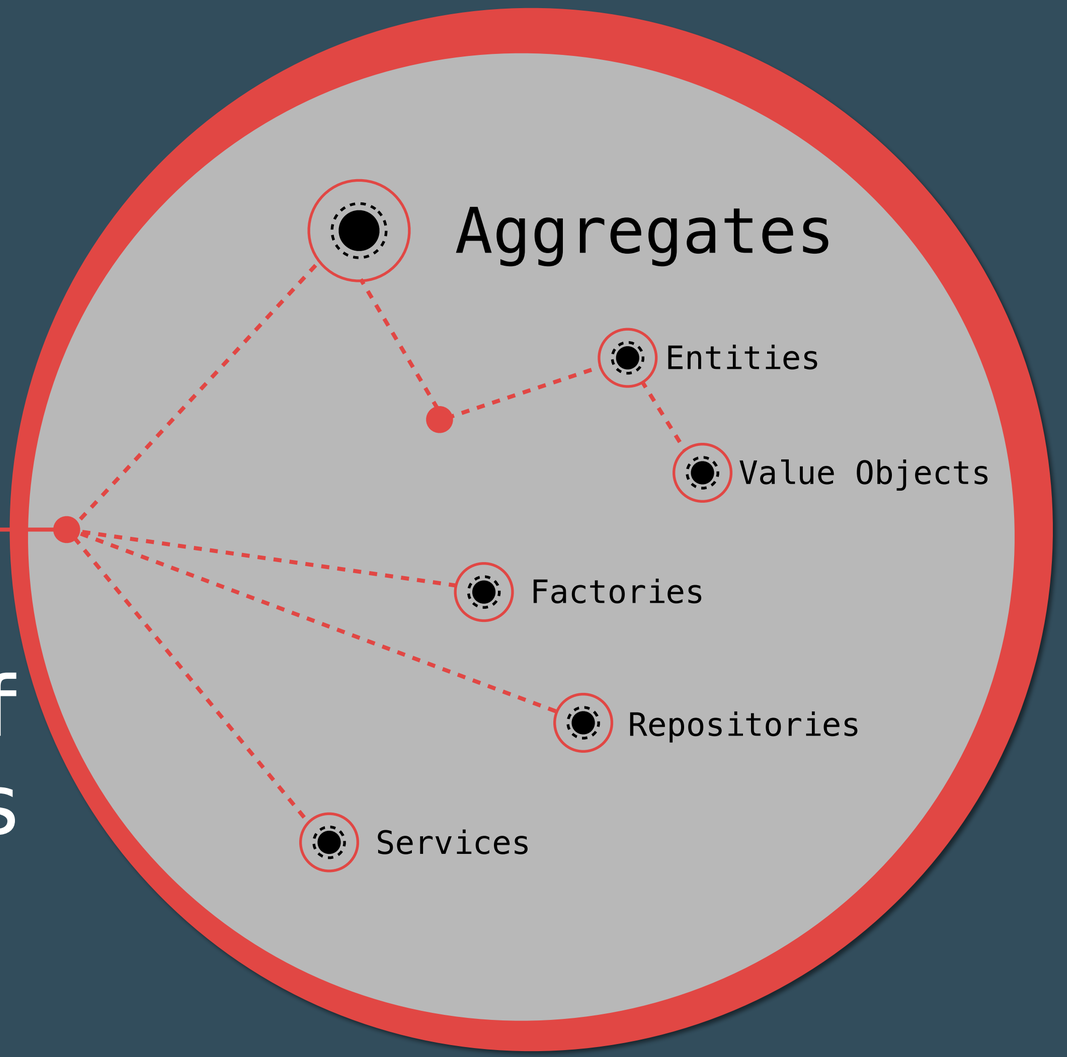
helps us with Microservices in four areas



(Internal)
Building Blocks

Building Blocks

help designing
the internals of
Bounded Contexts



Building Blocks

Entities

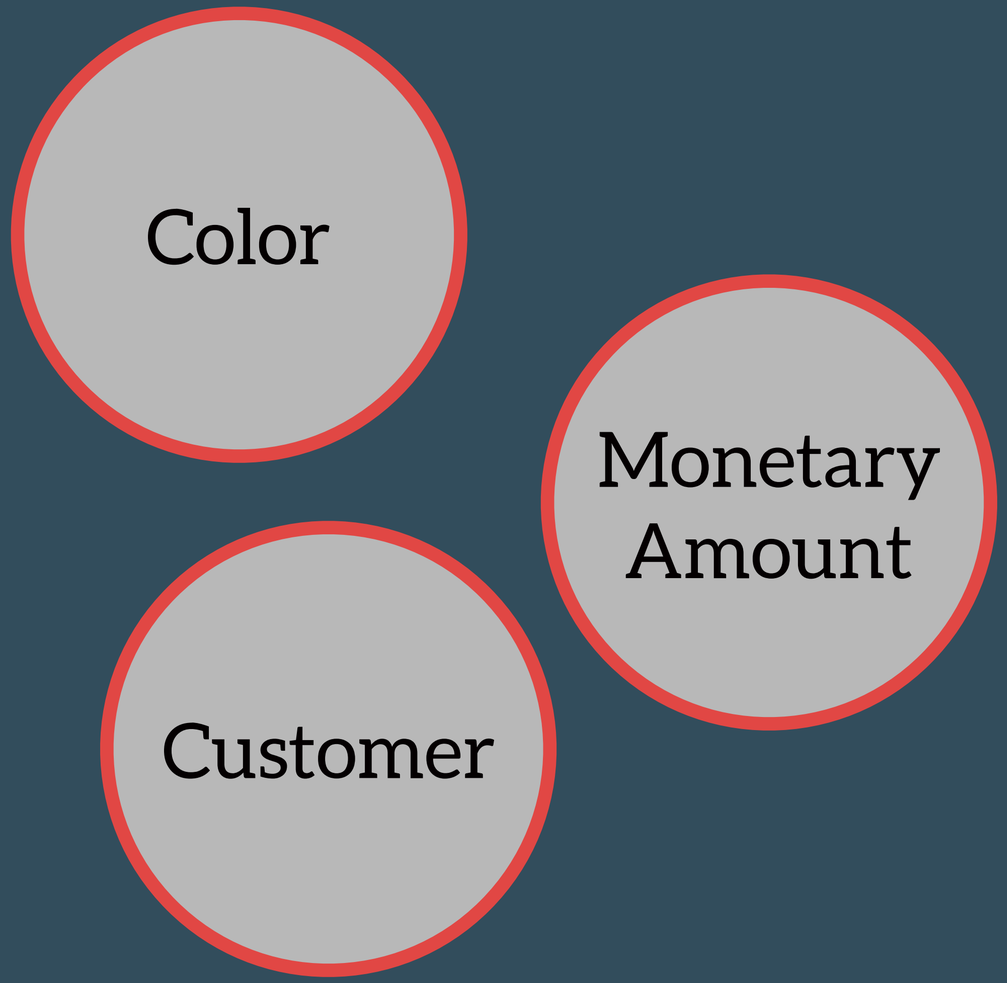
Entities represent the core business objects of a bounded context's model

Each **Entity** has a constant identity

Each **Entity** has its own lifecycle

Building Blocks

Value Objects



Value Objects derive their identity from their values

Value Objects do not have their own lifecycle, they inherit it from Entities that are referencing them

You should always consider value objects for your domain model

Building Blocks

Is „Customer“ an Entity or a Value Object

If an object can be considered an Entity or a Value Object always depends on the (Bounded Context) it resides in.

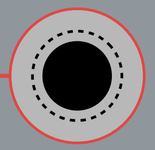
Example: A customer is an entity in a CRM-like microservice but not in a microservice that prints badges, there we are just interested in name, job description and Twitter handle

Building Blocks

Aggregates

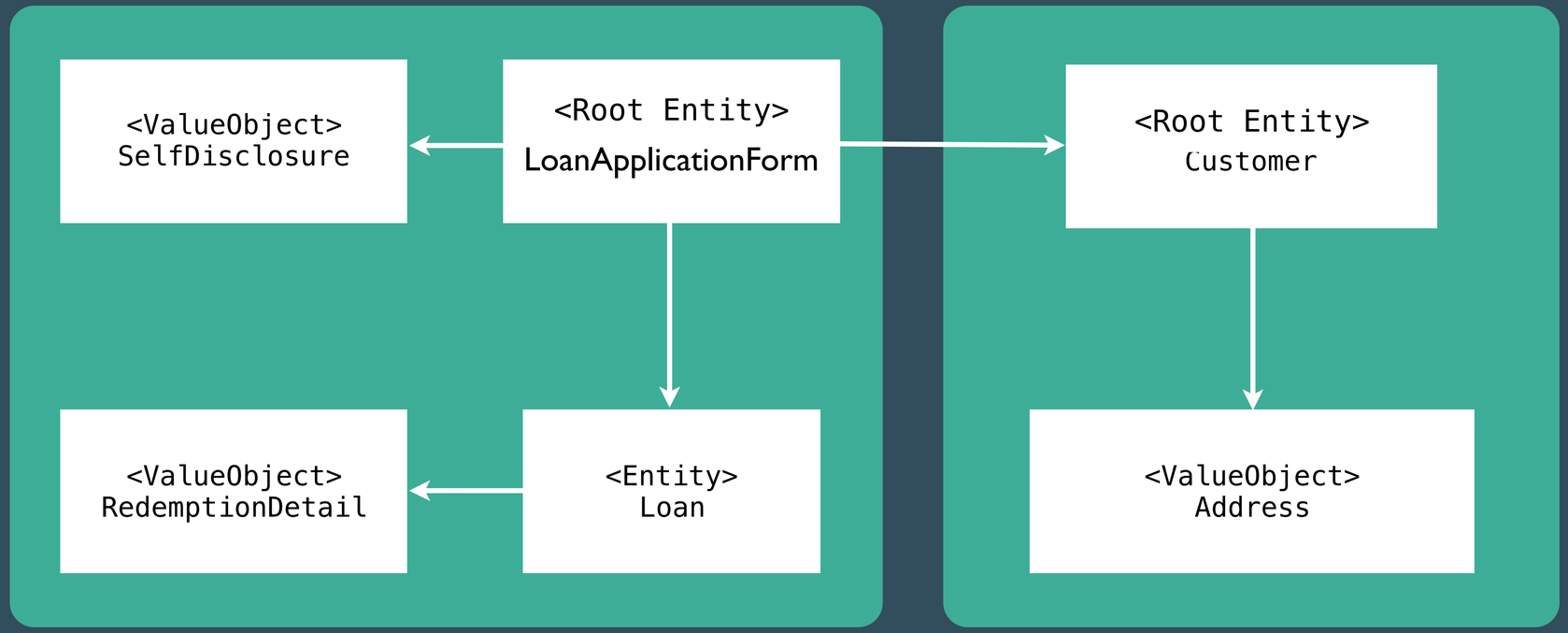
Do not underestimate the **power of the Aggregate**

Building Blocks



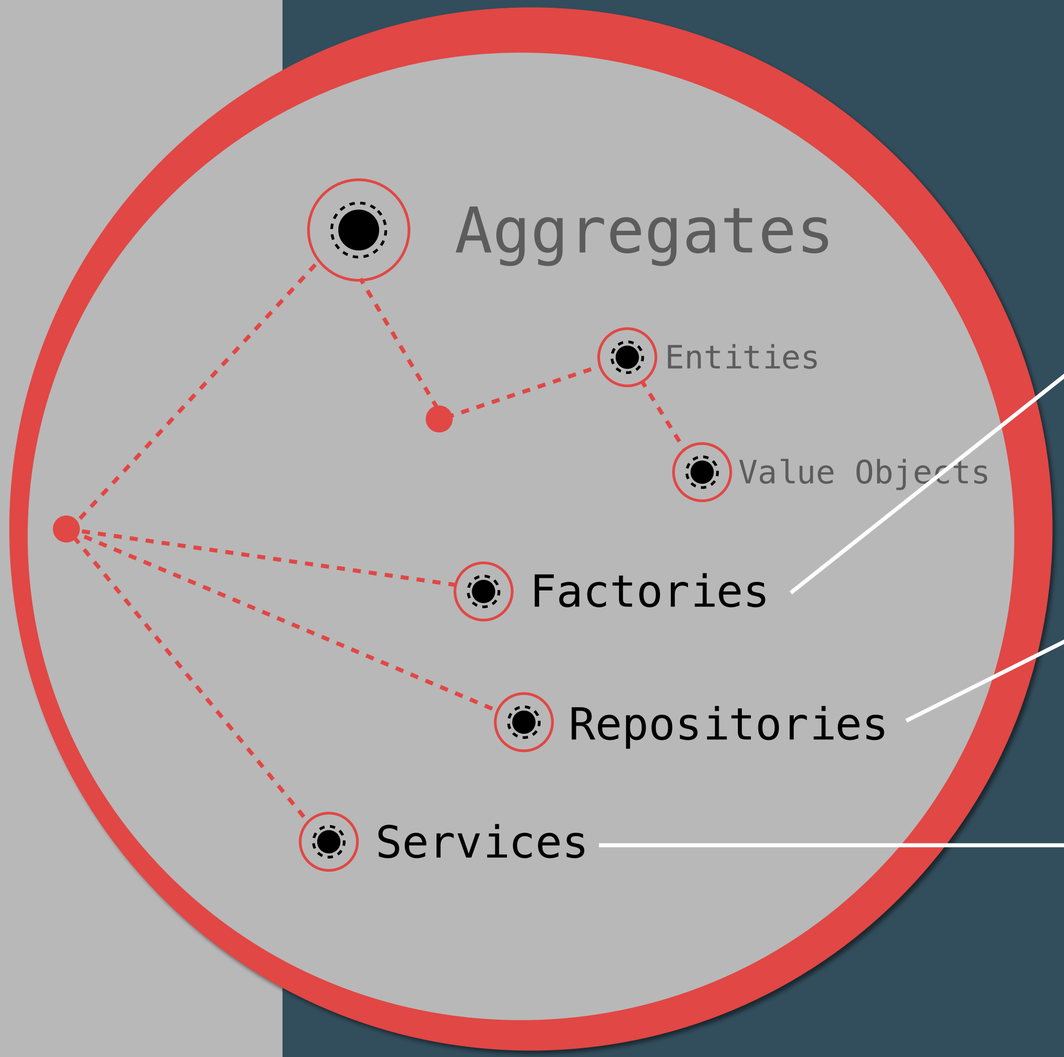
Aggregates

Aggregates group Entities. The Root Entity is the lead in terms of access to the object graph and lifecycle.



Building Blocks

Factories, Services, Repositories



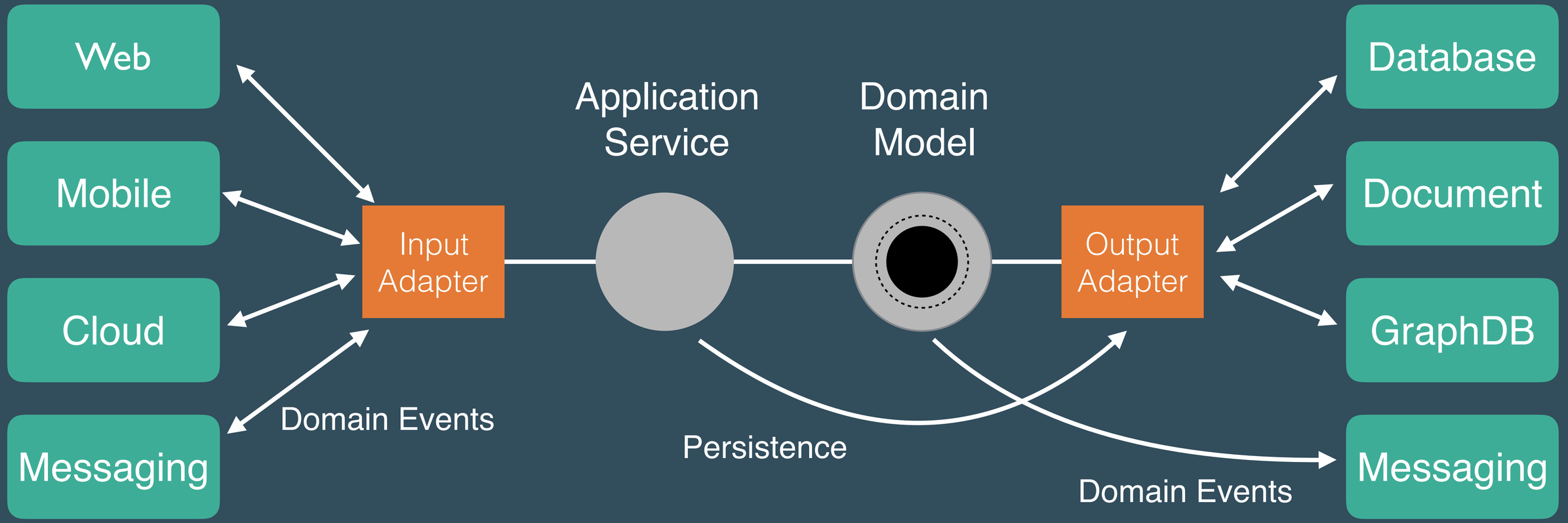
Factories take care of Entity- / Aggregate-Instantiations

Repositories encapsulate and represent data access

Services implement business logic that relates to multiple Entities / Aggreates

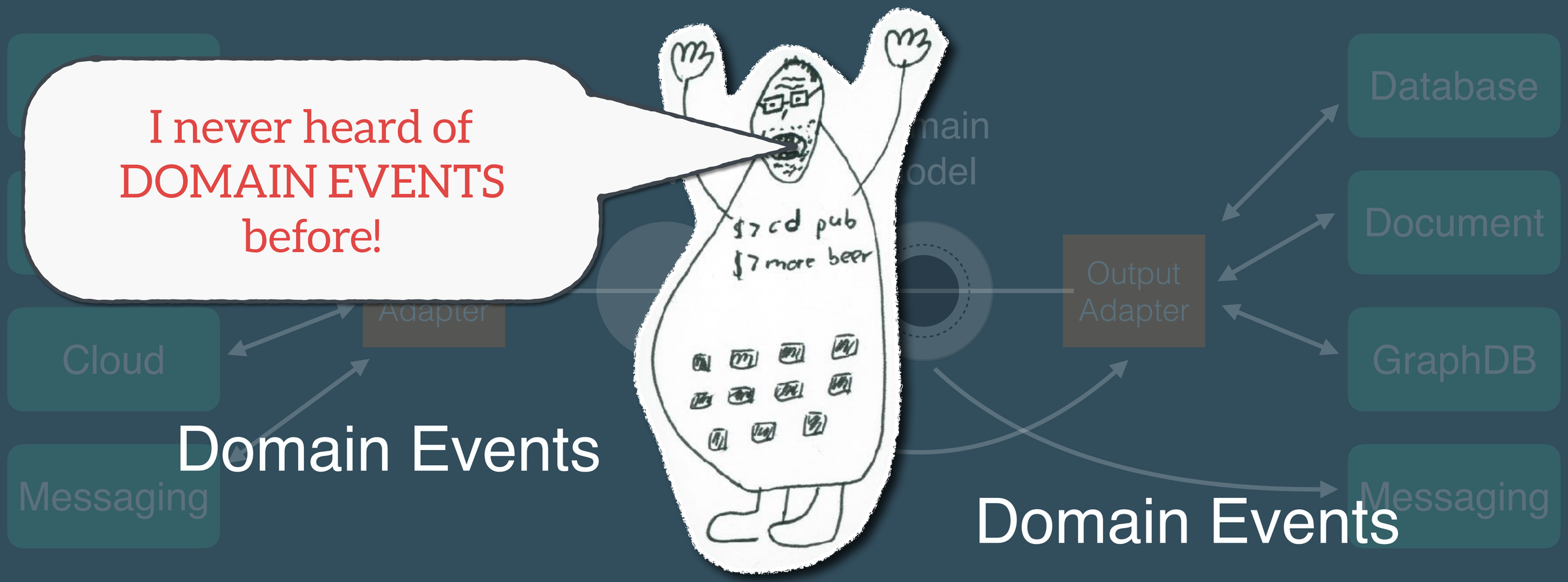
Building Blocks

Align the internal building blocks along Application Services and the Domain Model



Building Blocks

Align the internal building blocks along Application Services and the Domain Model



„If that happens“

„After the customer has“

„Notify me if“

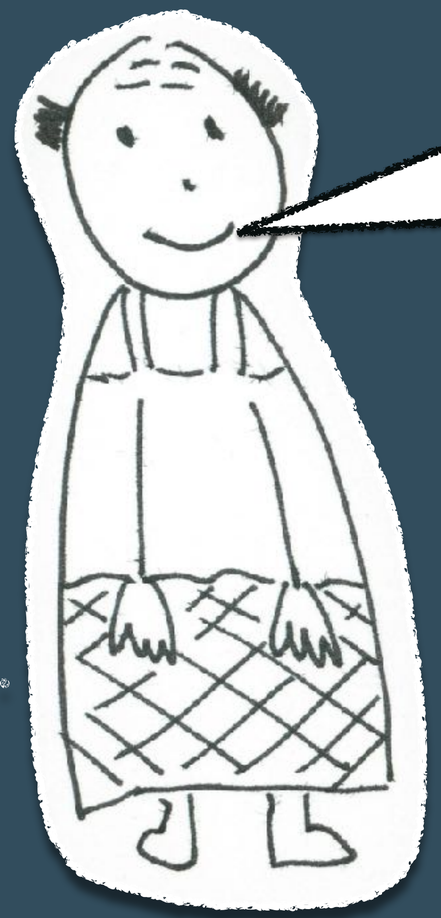
„When ..“

„After inserting data into“

„We need to check the status of“

„When we have called System X“

Ubiquitous Language anyone?

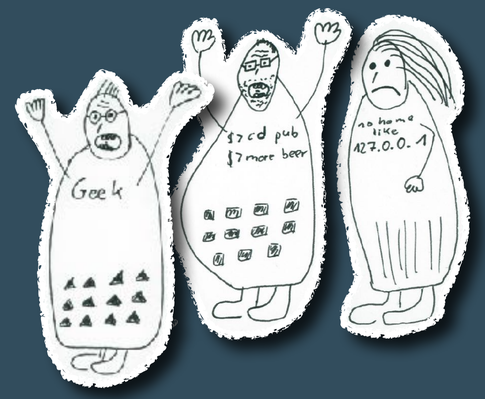


Domain Events are something that happened that Domain Experts care about

Model information about activity in the domain as a series of discrete events.

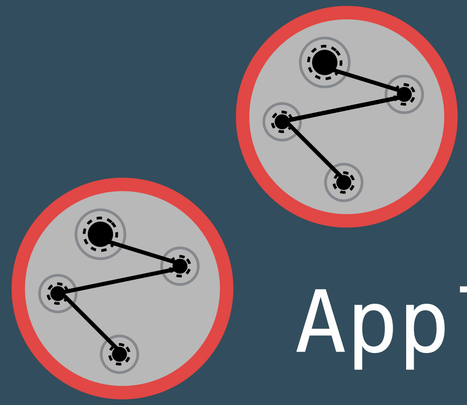
Triggers of Events

Time

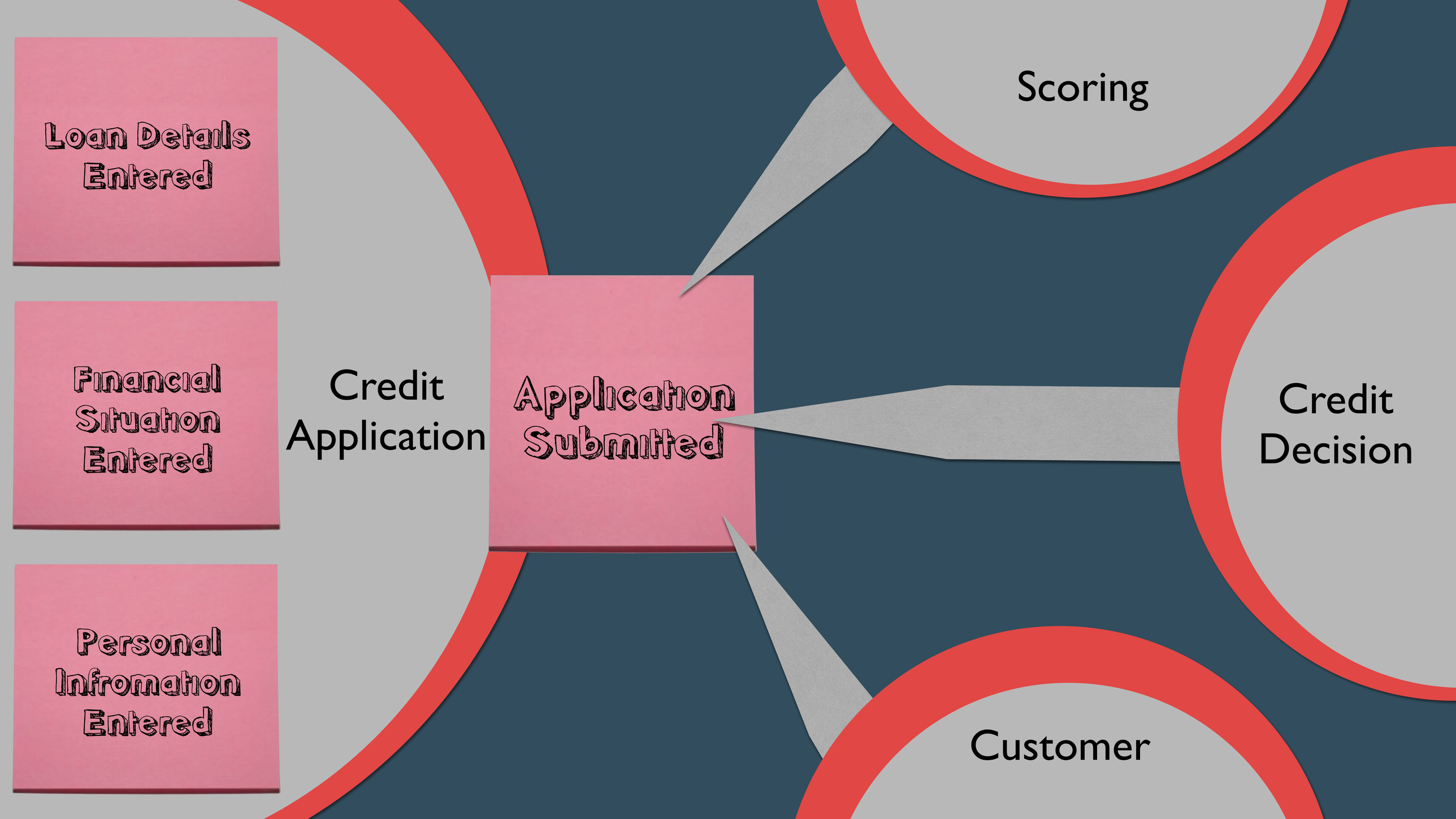


User Actions

Documents

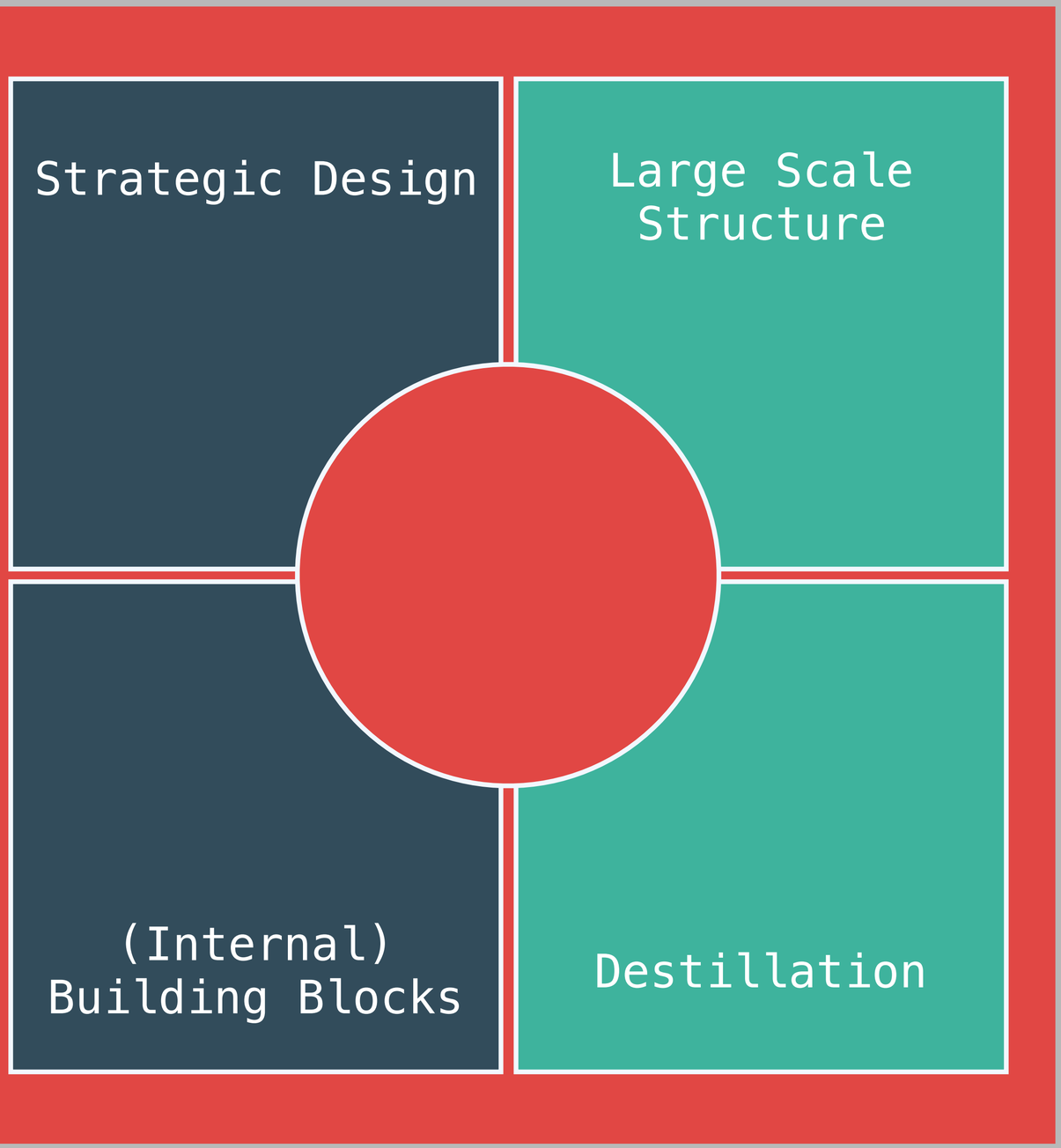


Applications


Loan Details Entered
Financial Situation Entered
Personal Infromation Entered
Credit Application
Application Submitted
Scoring
Credit Decision
Customer

Domain Driven Design

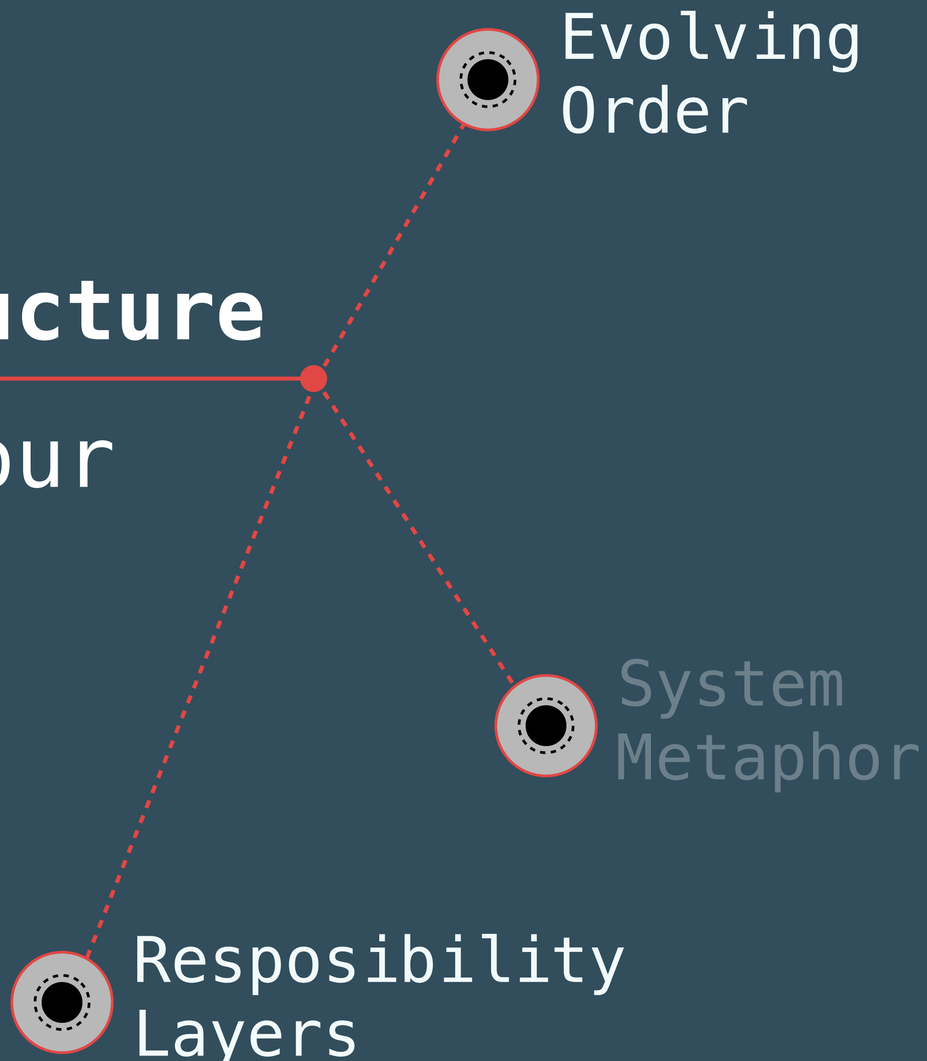
helps us with Microservices in four areas



Large Scale Structure

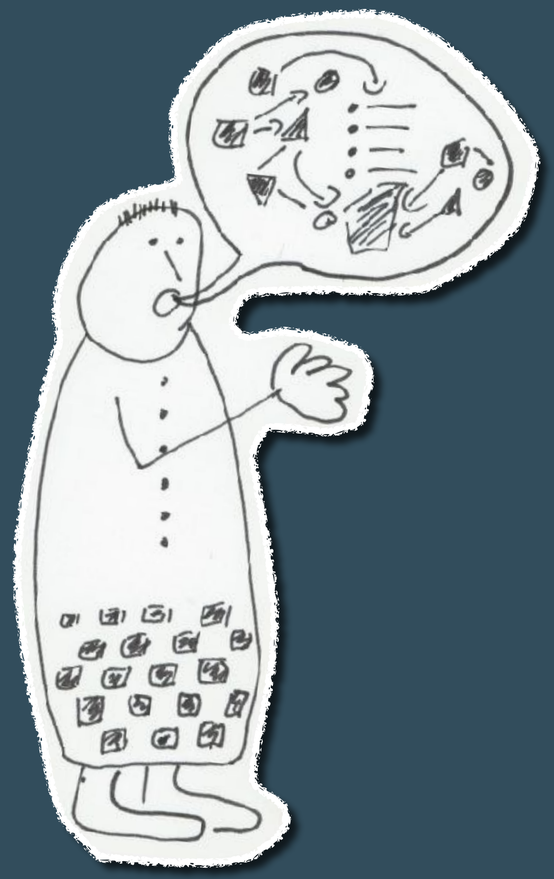
Large Scale Structure

helps evolving our Microservice landscapes



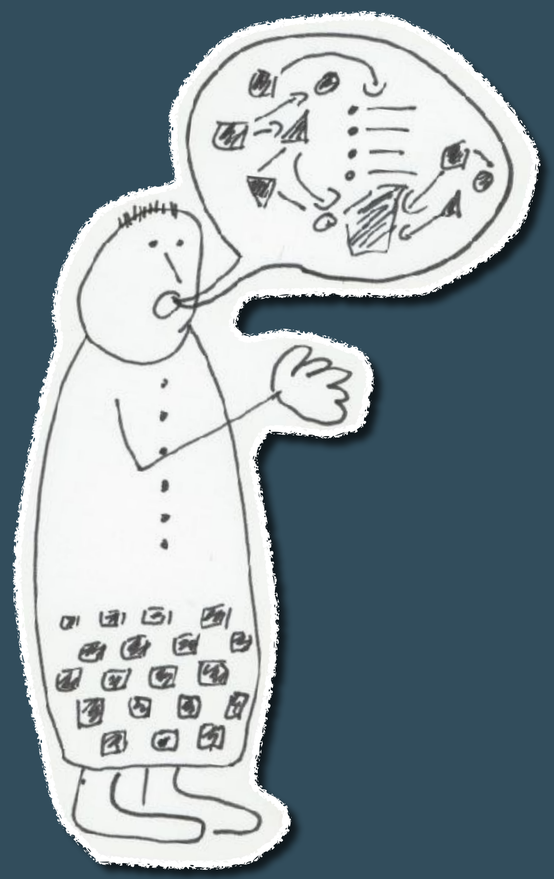
Large Scale Structure

Evolving Order



Large Scale Structure

Evolving Order



Job Title:
Chief Ivory Tower Architect

Large Scale Structure

Evolving Order

Job Title:
Chief Ivory Tower Architect

Rigid Development Guidelines

Inflexible Architecture

Clear rules and conventions for everything

„I don’t need expensive developers, I prefer cheap ones and I do the thinking for them“

Large Scale Structure

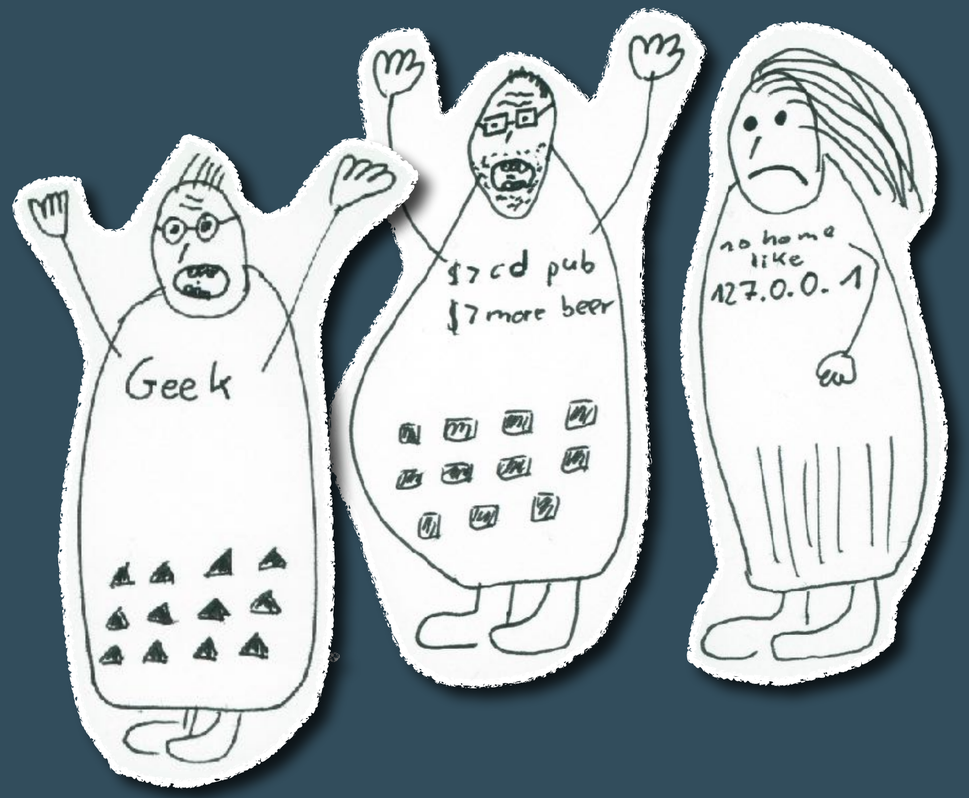
Evolving Order

Development Team

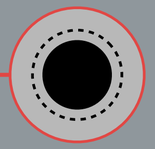
System is too complex

Let's dumb down the system to fit the rules

We need a workaround to undermine some rules



Large Scale Structure



Evolving Order

Evolving Order

Let large structures evolve, don't overconstrain design principles

These large structures should be applicable across bounded contexts

However there should be some **practical** constraints

Sounds familiar in a microservice environment?

Large Scale Structure

Responsibility Layers

Large Scale Structure

Responsibility Layers

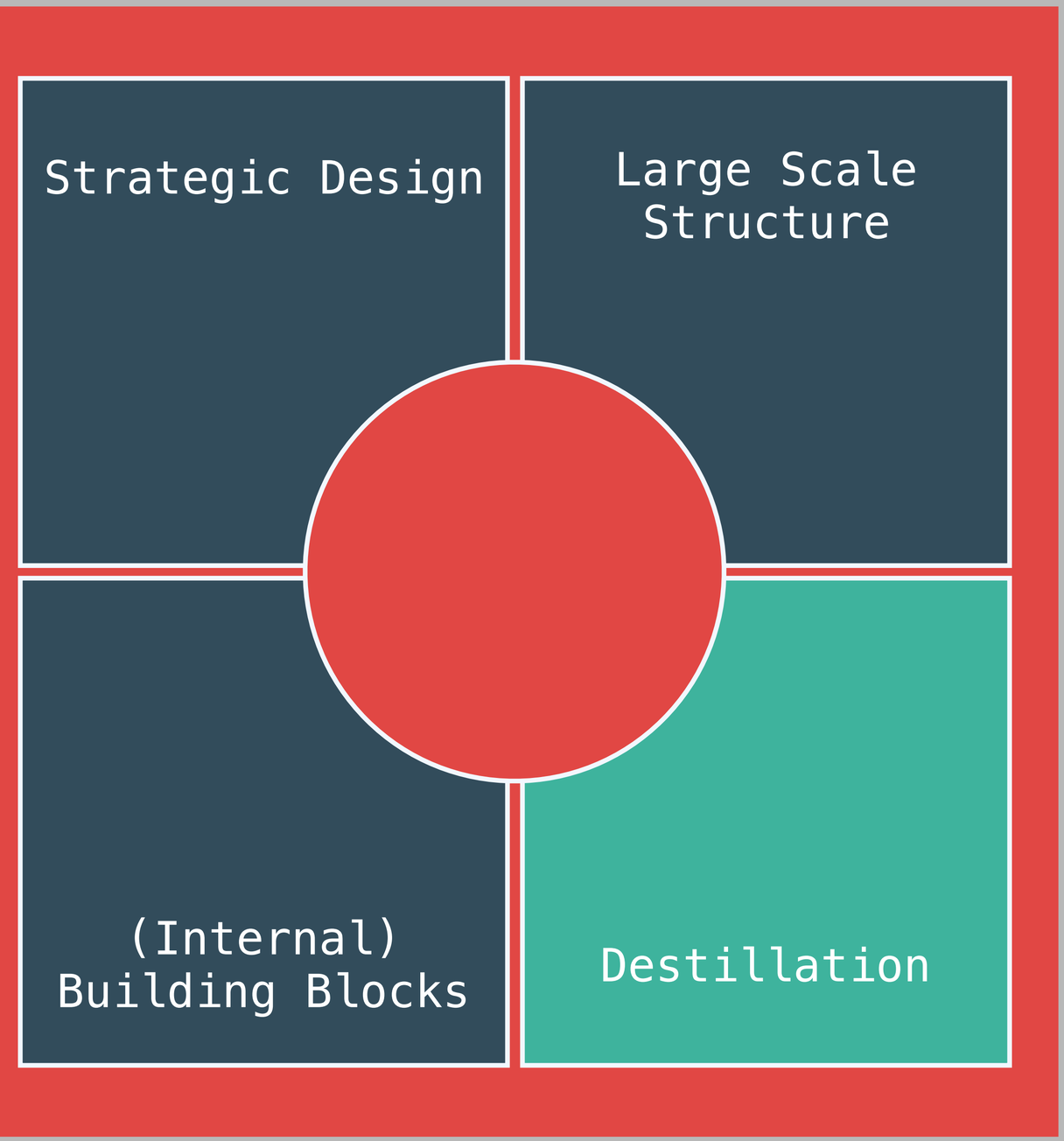
Each of these Microservices is structures according to a bounded context

Inside these context developers have the chance to use building blocks

However we could structure our bounded context according to responsibilities

Domain Driven Design

helps us with Microservices in four areas



Destillation

Destillation

helps extracting Microservices out of an existing monolithic application

Destilla-
tion

Identify the core domain

Vision Statement

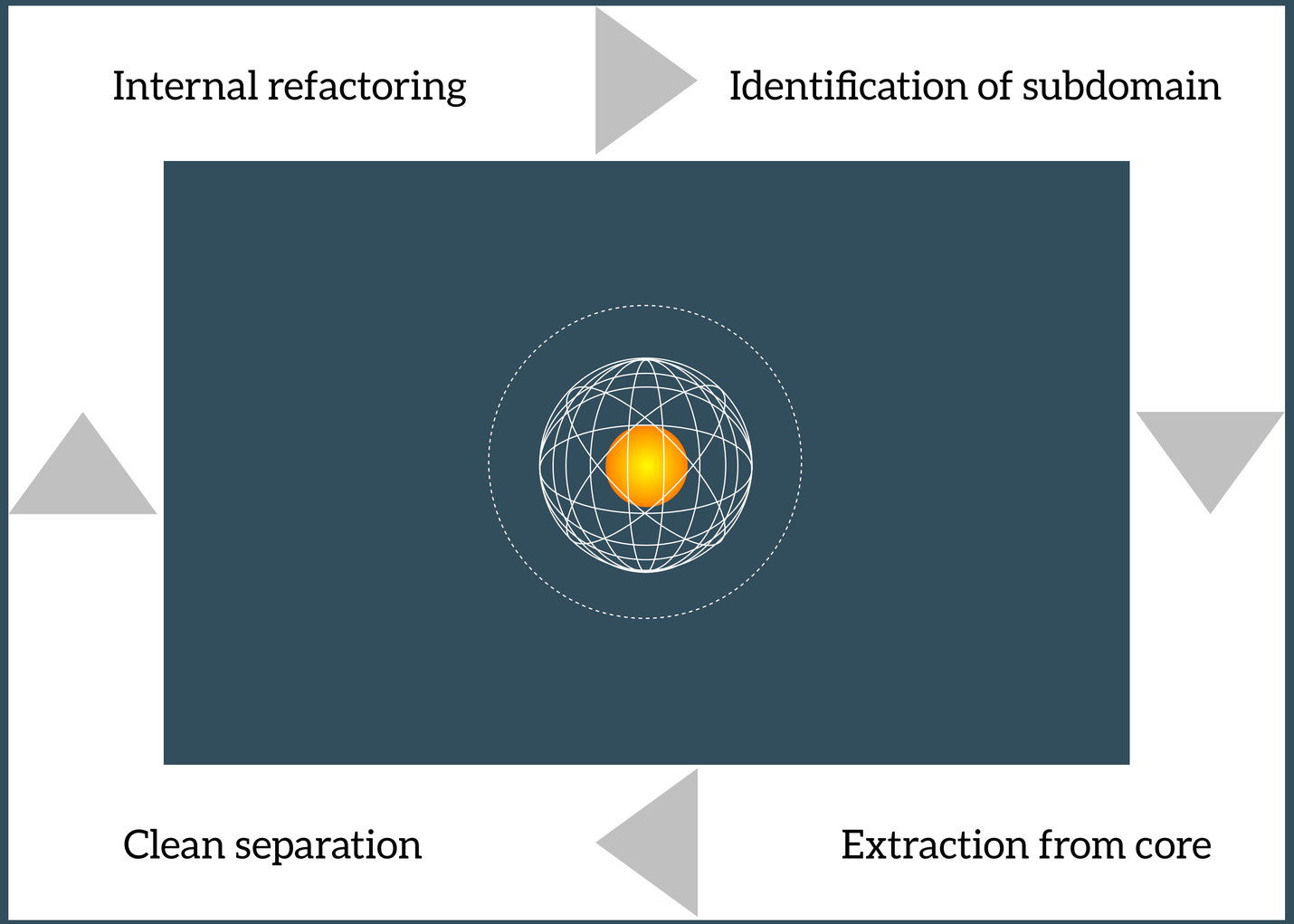
Defines what is in the core domain and what is not in the core domain

Destillation Document

Describes all the details of the core domain

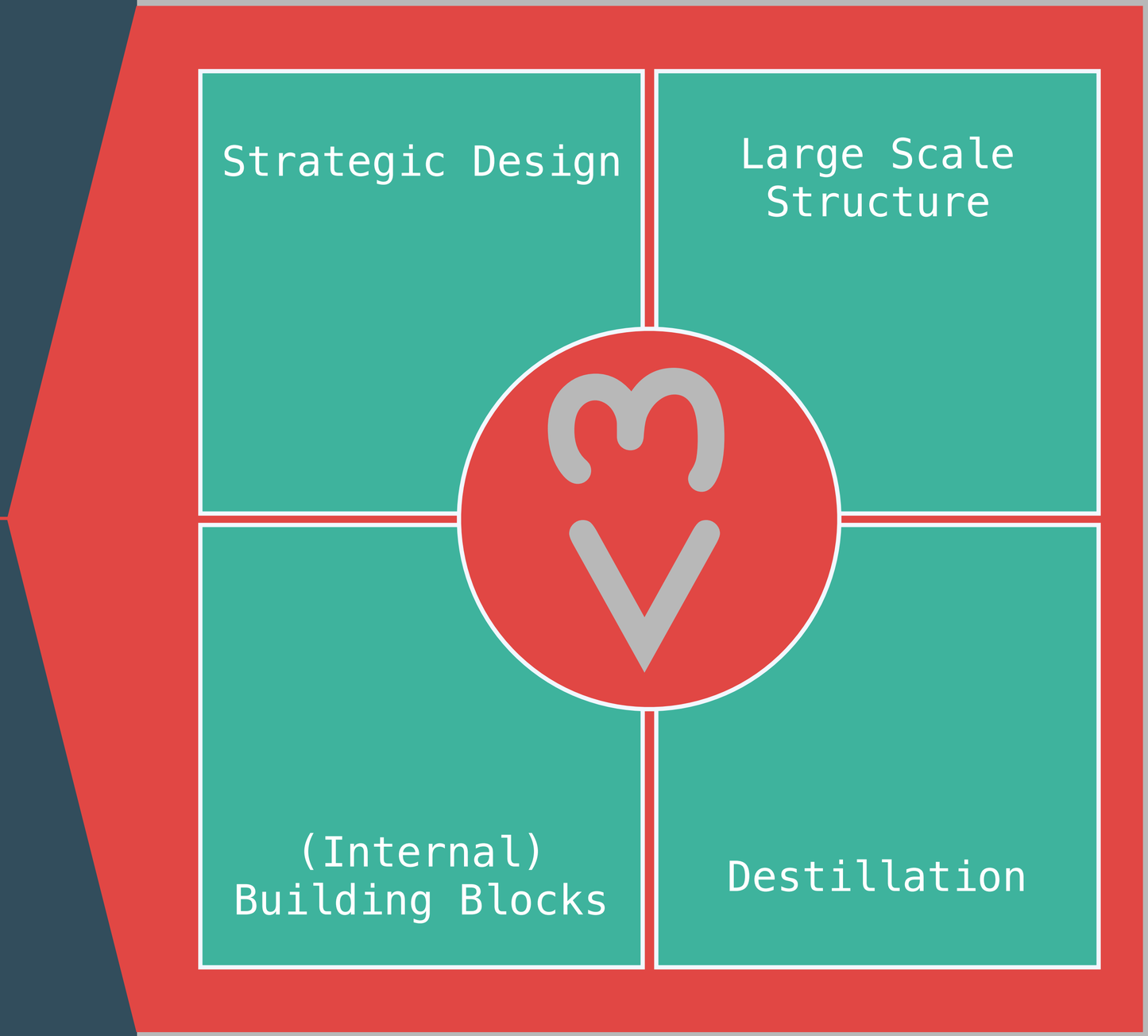
Destilla-
tion

Extract subdomains



Microservices

Domain Driven Design



THANK YOU!

Michael Plöd - innoQ
@bitboss

Shameless plug: we offer DDD trainings and consulting