Microservices

Domain Driven Design

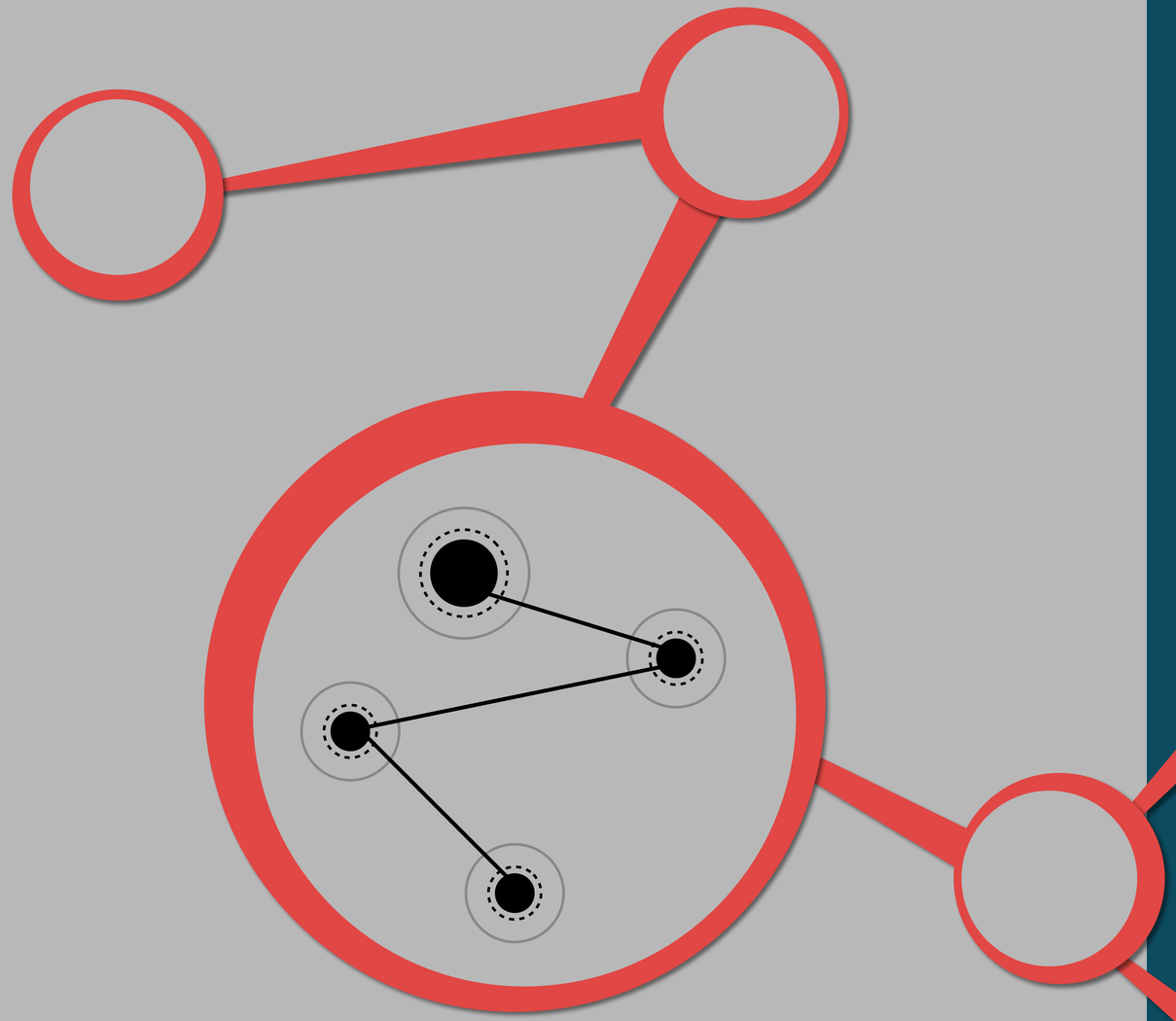
Michael Plöd - innoQ
@bitboss

Disclaimer

Die meisten dieser Ideen stammen nicht von mir sondern aus Eric Evans herausragendem Buch Domain Driven Design. Dieses Buch möchte ich jedem Zuhörer meines Vortrags empfehlen.

Ich übernehme die englischen Bezeichnungen aus Erics Buch im Sinne der Ubiquitous Language (Stichwort: Anglizismen)

Michael Plöd - innoQ
@bitboss



D D D

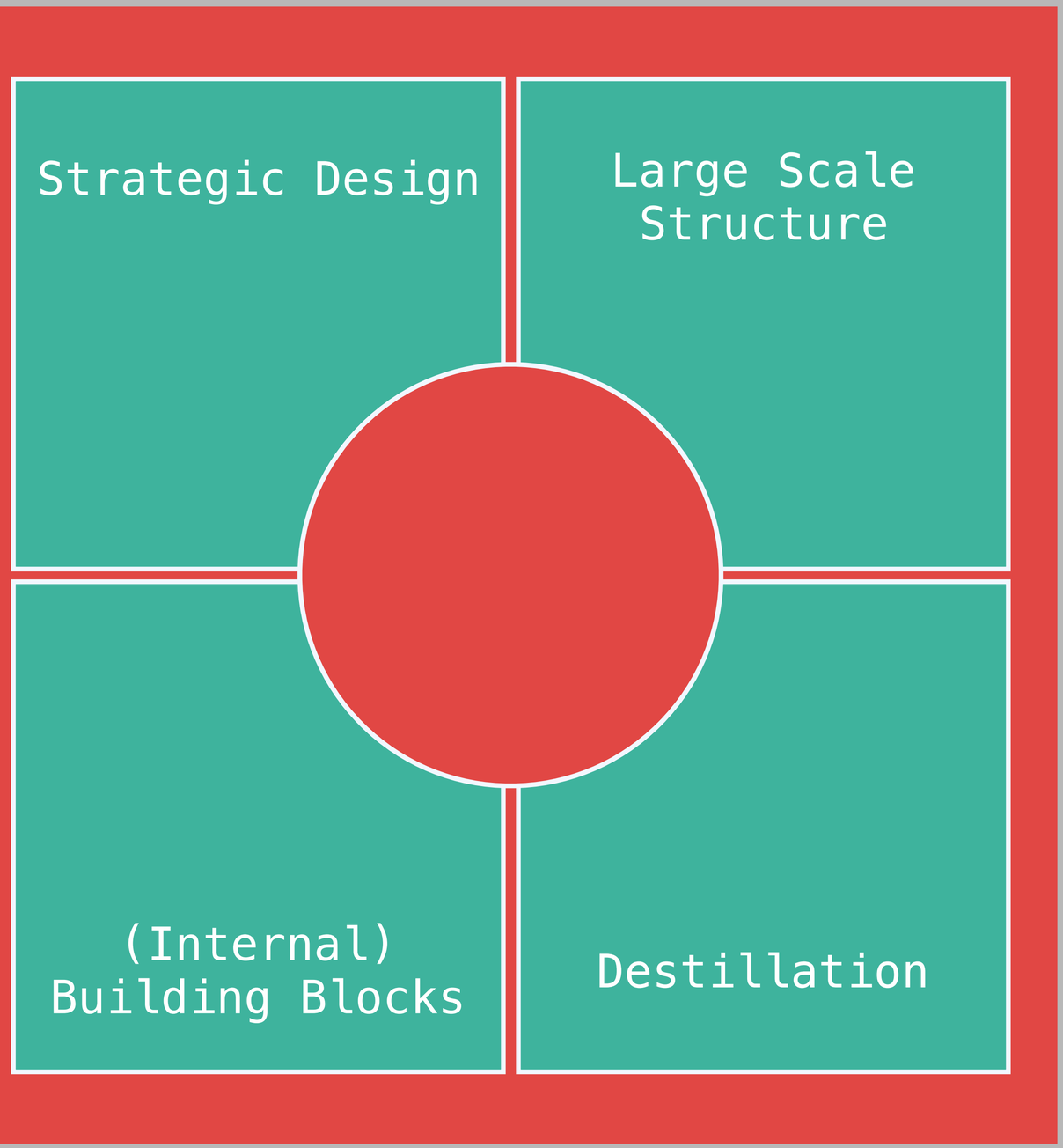
in Microservices

Der Zusammenhang von DDD und Microservices geht über Bounded Contexts hinaus

Bei DDD geht es um mehr als nur um Aggregate, Entitäten und Services

Domain Driven Design

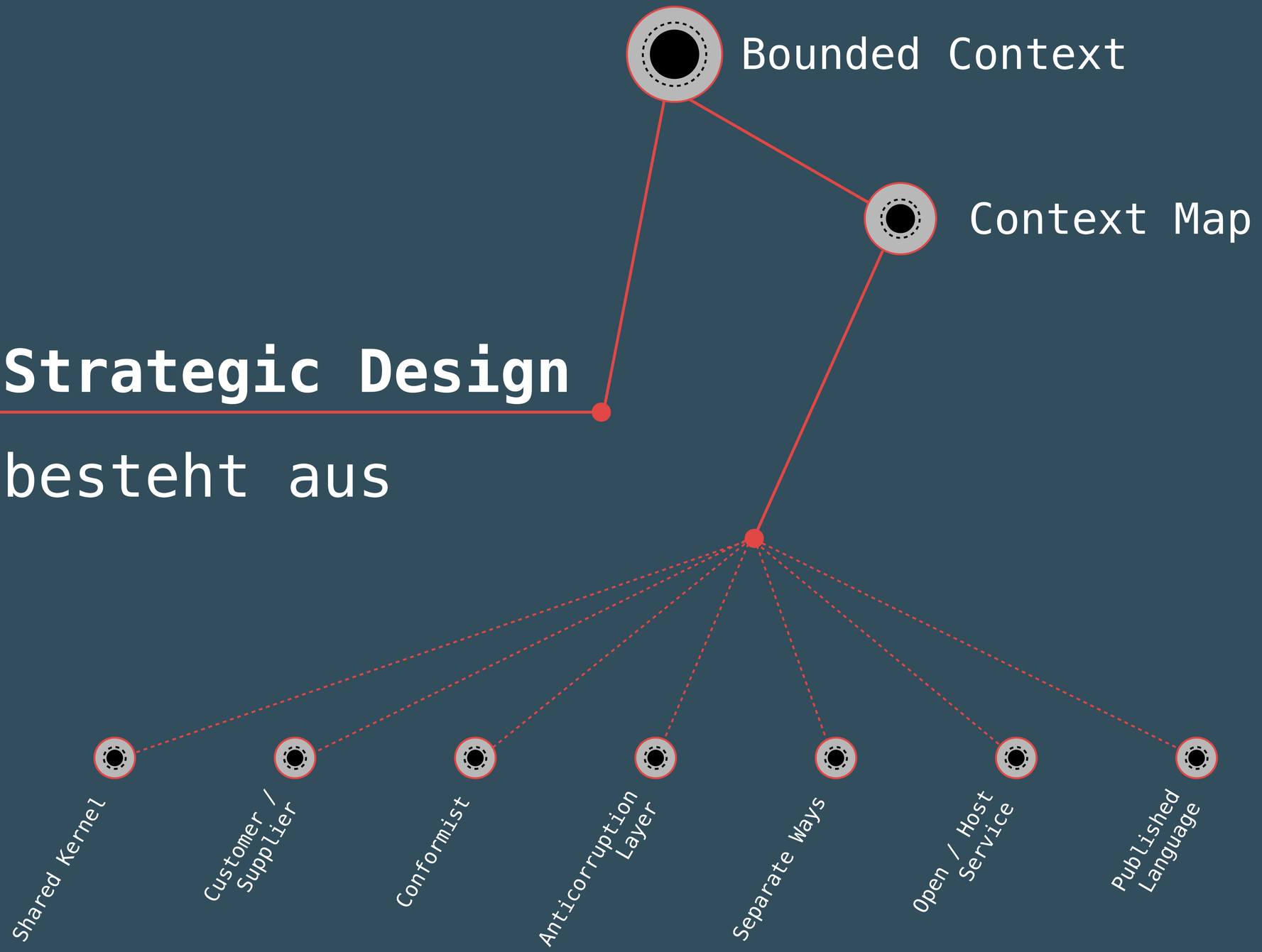
hilft uns im Hinblick auf Microservices in vier Bereichen



Strategic Design

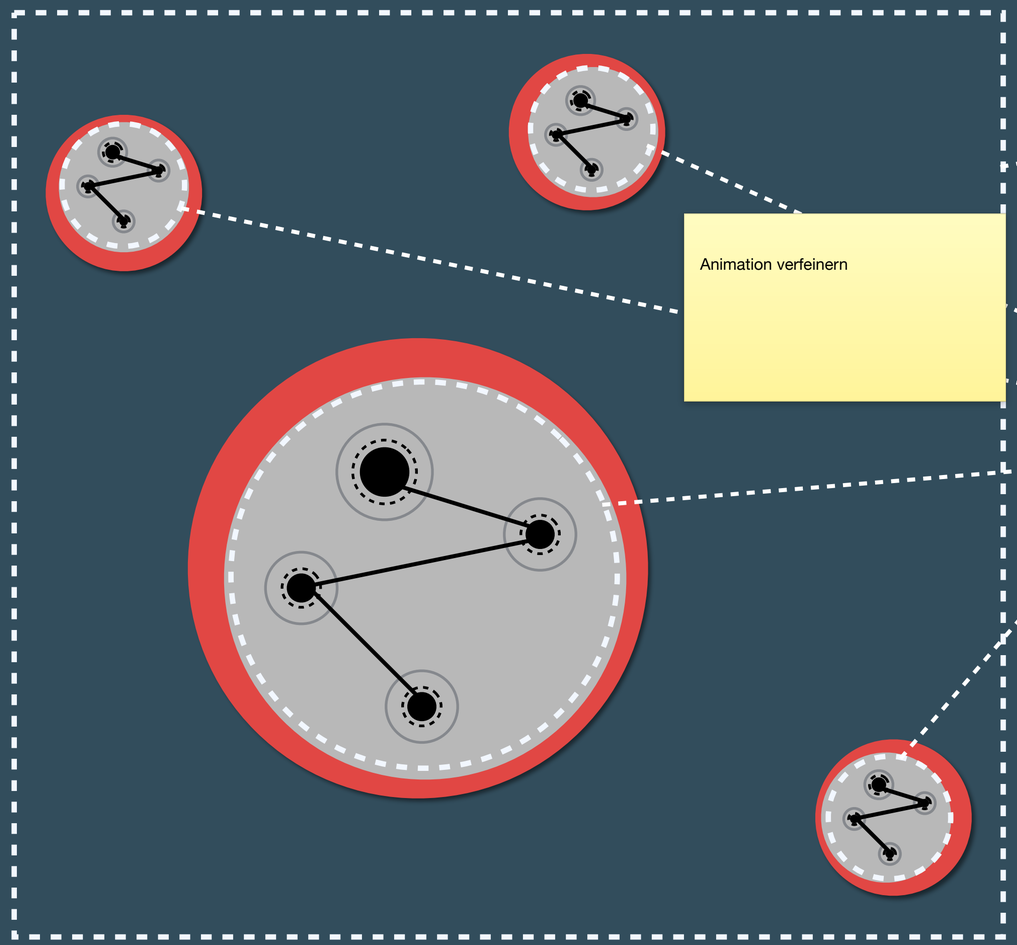
Strategic Design

besteht aus



Strategic Design

Bounded Context



Jede anspruchsvolle Domäne besteht aus mehreren **Bounded Contexts**

Jeder **Bounded Context** enthält sein Model und ggf weitere Contexts

Der **Bounded Context** ist zudem eine Grenze um die Gültigkeit eines Models

Strategic Design

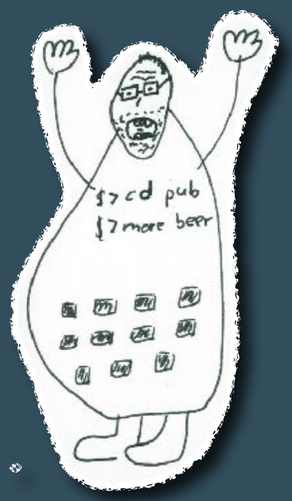
Bounded Context Beispiel

Anmeldung

Event Management

Badges

Kunde



Name

Zahlungsart

Adresse

Firma

Session Registrierungen

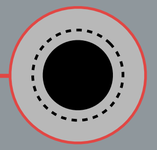
Essenspräferenzen

Name

Job Titel

Twitter Handle

Strategic Design



Bounded Context Example

Jeder Bounded Context hat sein eigenes Model eines Kunden

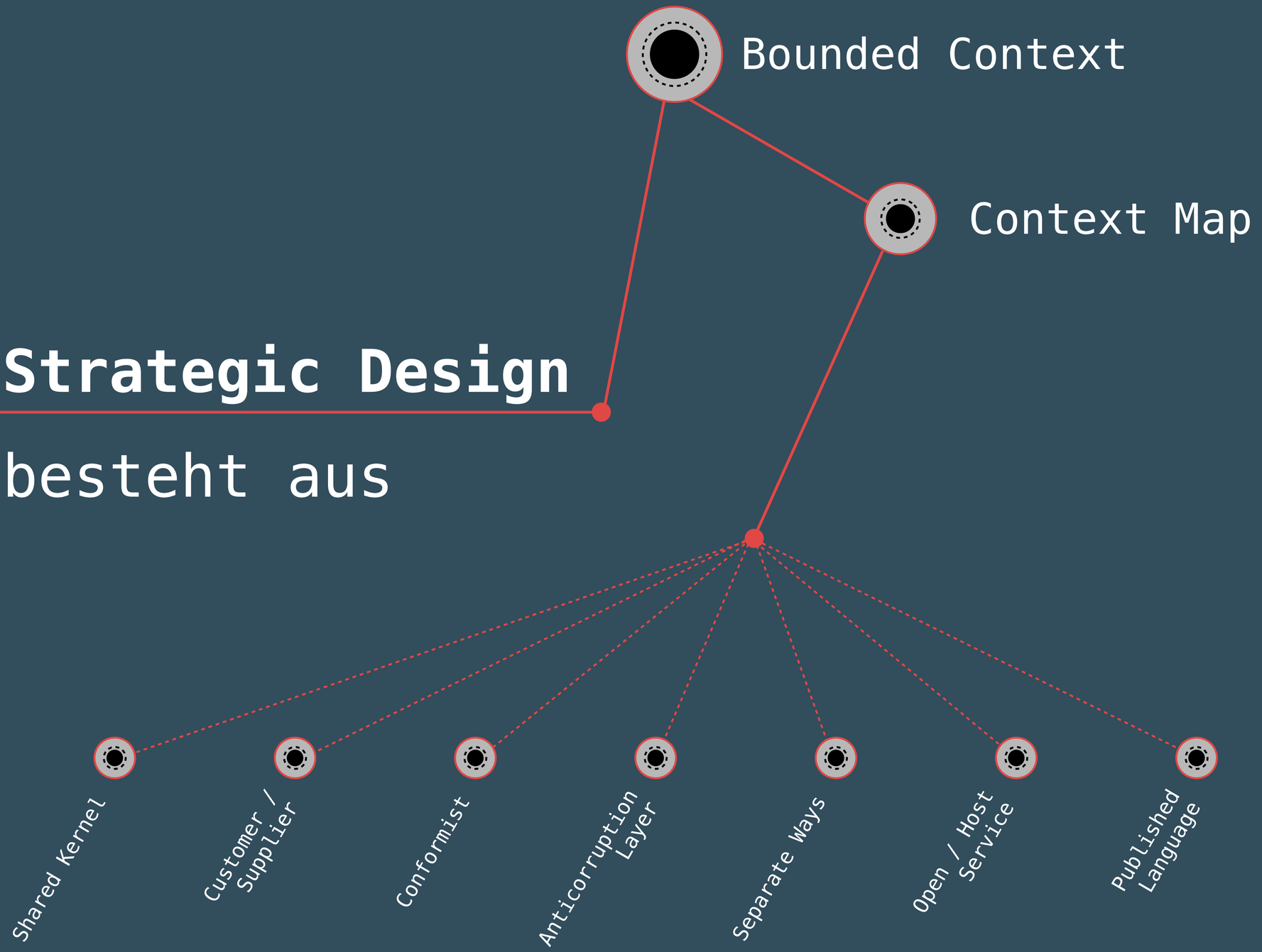
Dies ist ein wichtiger Enabler für unabhängige Microservices

Bitte beachten Sie den Namen des Kunden, vielleicht ein Anlass für geteilte Daten

Strategic Design

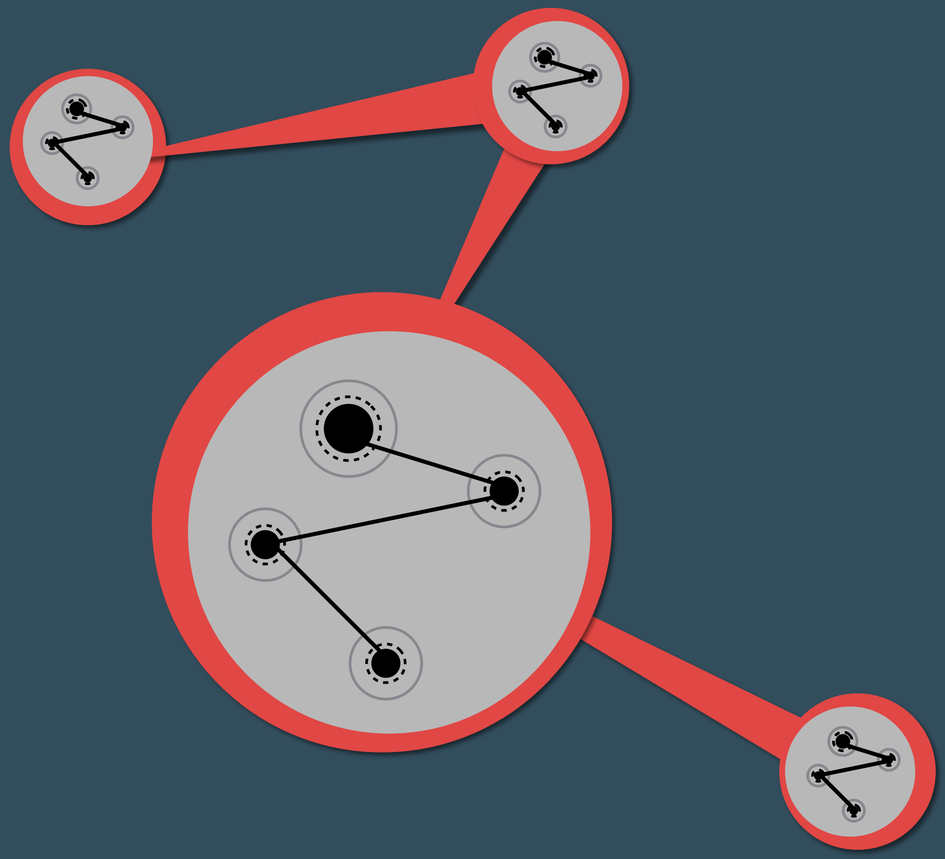
Strategic Design

besteht aus



Strategic Design

Context Map



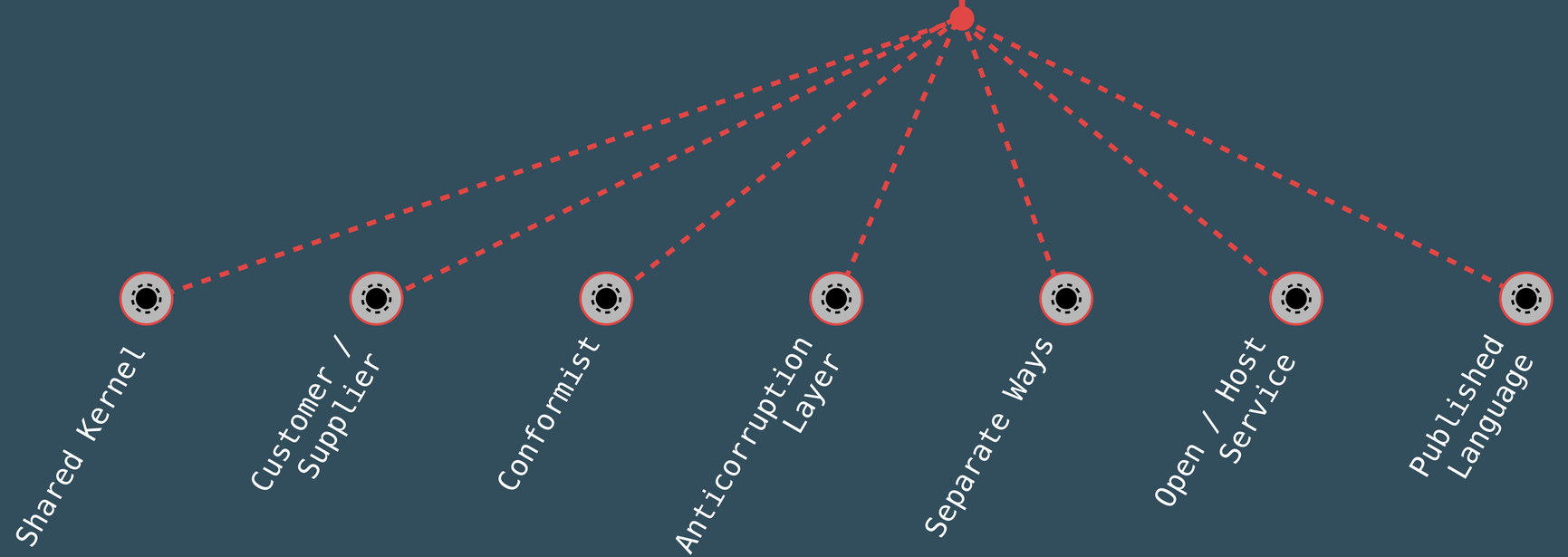
Der Bounded Context an sich liefert noch keinen Überblick über ein System

Durch die Einführung einer **Context Map** beschreiben wir, wie sich Models propagieren

Die **Context Map** ist somit eine gute Ausgangsbasis für künftige Transformationen

Strategic Design

Context Map – Patterns

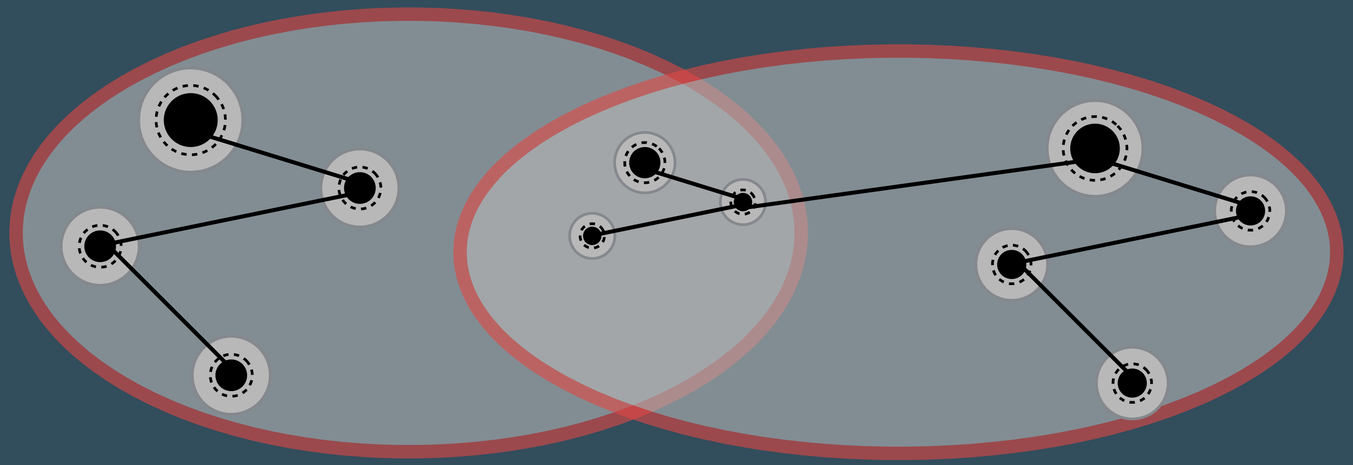


Strategic Design

Context Map – Patterns

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

Zwei Teams teilen sich ein Subset eines Domain Models inklusive Code und ggf. der Datenbank. Der Shared Kernel wird häufig auch als Kerndomäne bezeichnet.

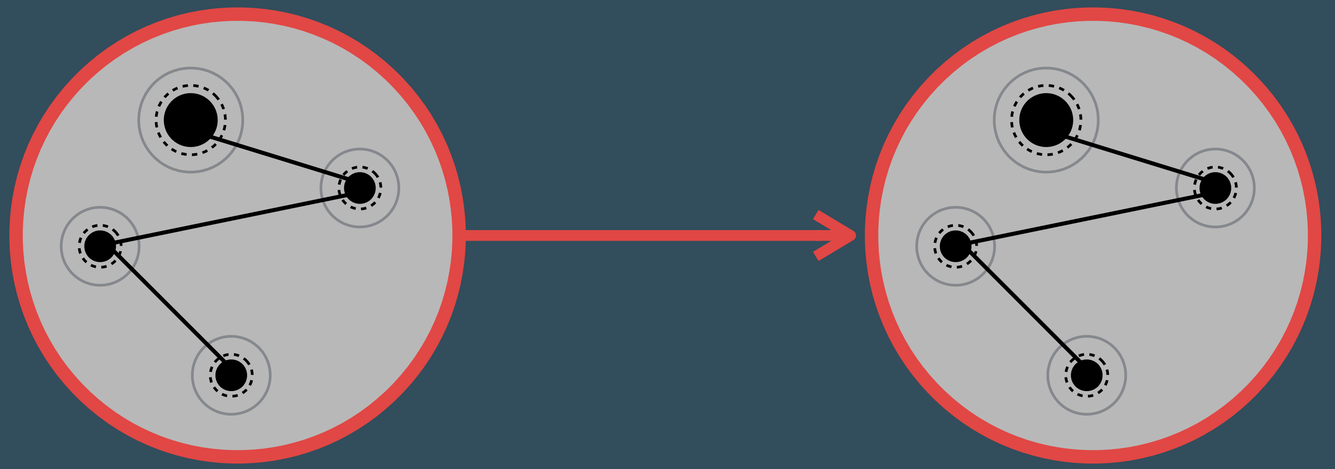


Strategic Design

Context Map – Patterns

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

Es gibt eine Kundenbeziehung zwischen zwei Teams. Das konsumierende Team ist Kunde mit häufig weitreichendem Einfluss (z.B. Vetorecht).

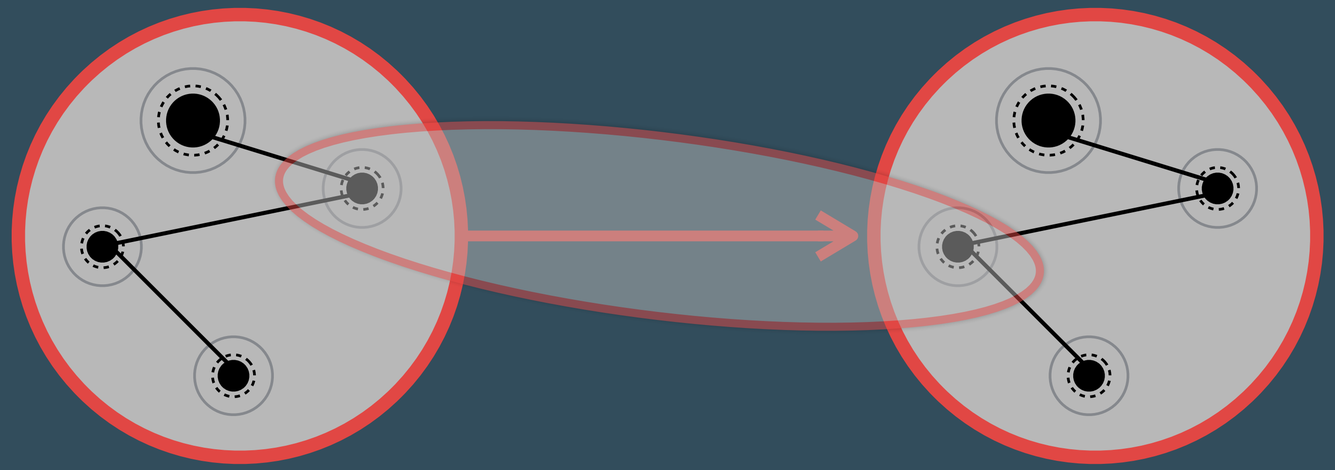


Strategic Design

Context Map – Patterns

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

Das konsumierende Team passt sein Model dem Model des Dienstanbieters an. Es findet keine Übersetzung des Models statt. Das konsumierende Team hat zudem kein Vetorecht. Achtung vor Propagation von schlechten Models!

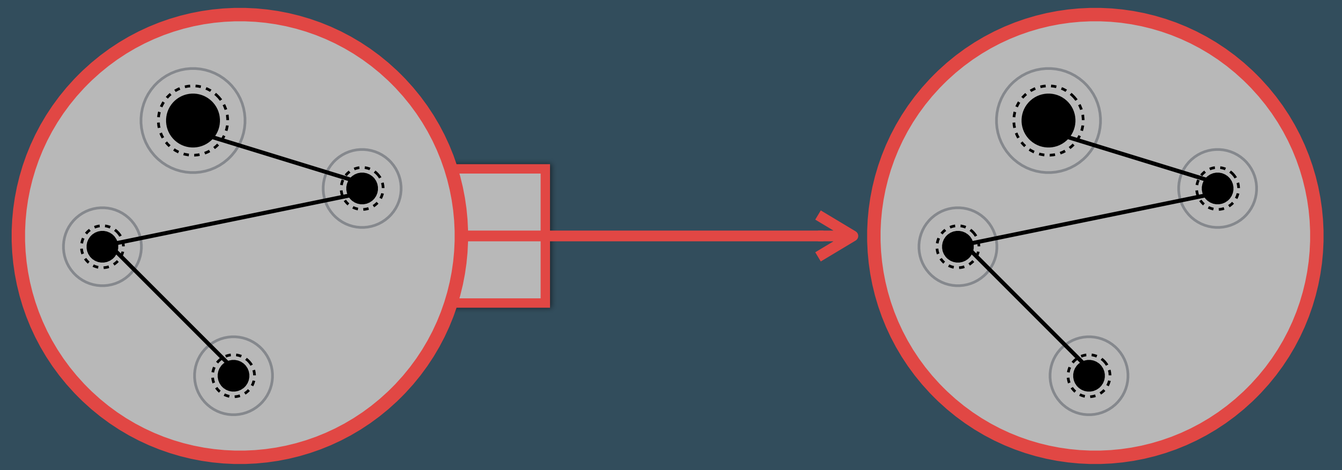


Strategic Design

Context Map – Patterns

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

Der Anticorruption Layer ist eine Komponente, die das Model eines Clients von dem eines anderen Systems isoliert. Üblicherweise geschieht dies über eine Model-Transformation.



Strategic Design

Context Map – Patterns

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

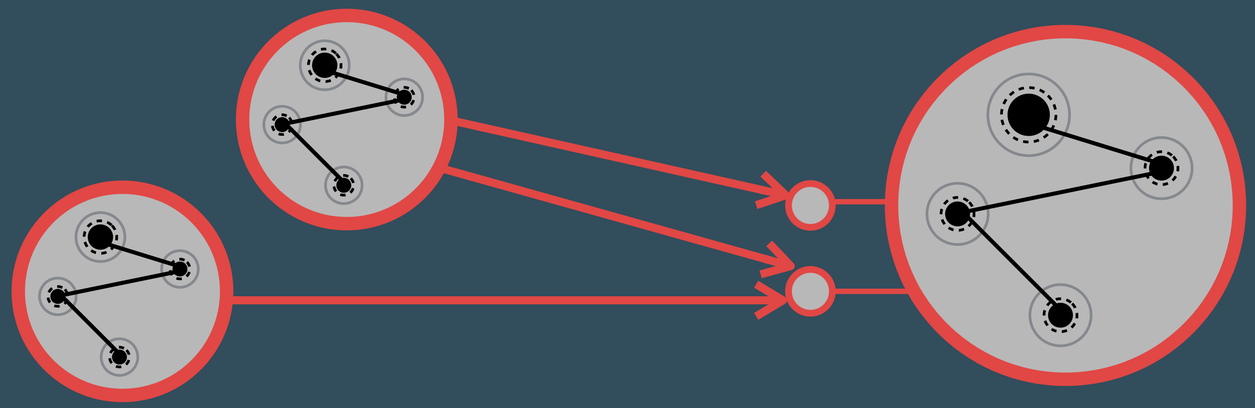
Es gibt keine Verbindung zwischen den Bounded Contexts von Systemen. Dies erlaubt den Teams unabhängig voneinander Lösungen zu finden und auszurollen.

Strategic Design

Context Map – Patterns

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

Jeder Bounded Context bietet ein definiertes Set an Services / Funktionalität nach aussen hin an. Jedes konsumierende System kann über diese integrieren.

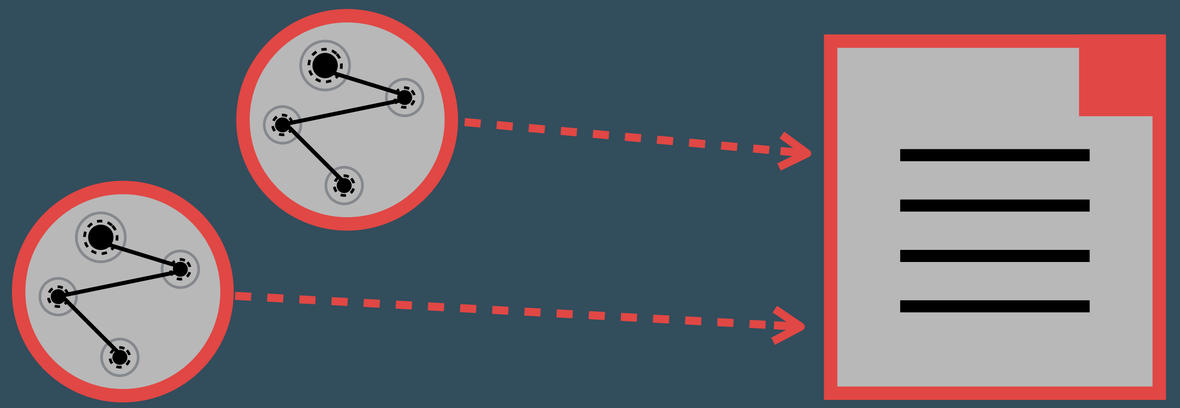


Strategic Design

Context Map – Patterns

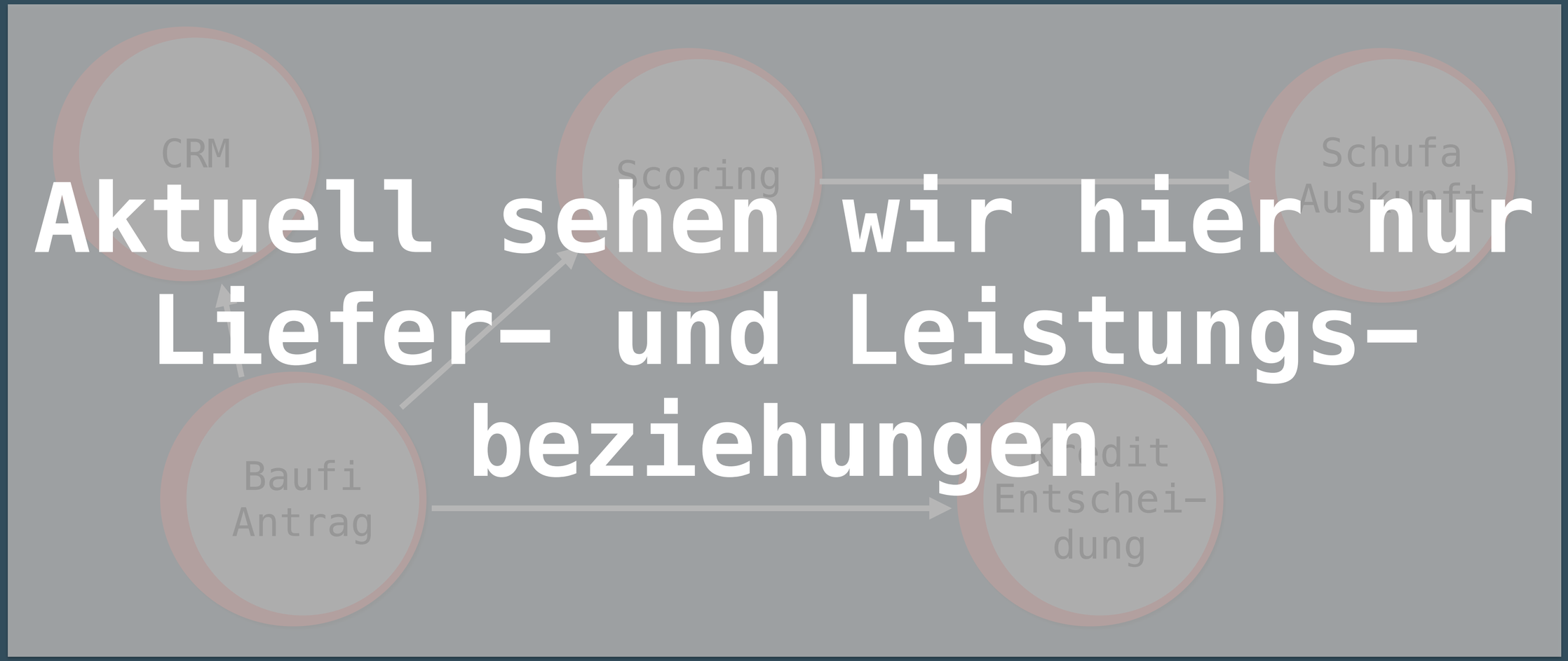
- Shared Kernel
- Customer / Supplier
- Conformist
- Anticorruption Layer
- Separate Ways
- Open / Host Service
- Published Language

Die Published Language is relativ ähnlich zum Open / Host Service. Allerdings erfolgt die Model Abstraktion über die Ubiquitous Language.



Strategic Design

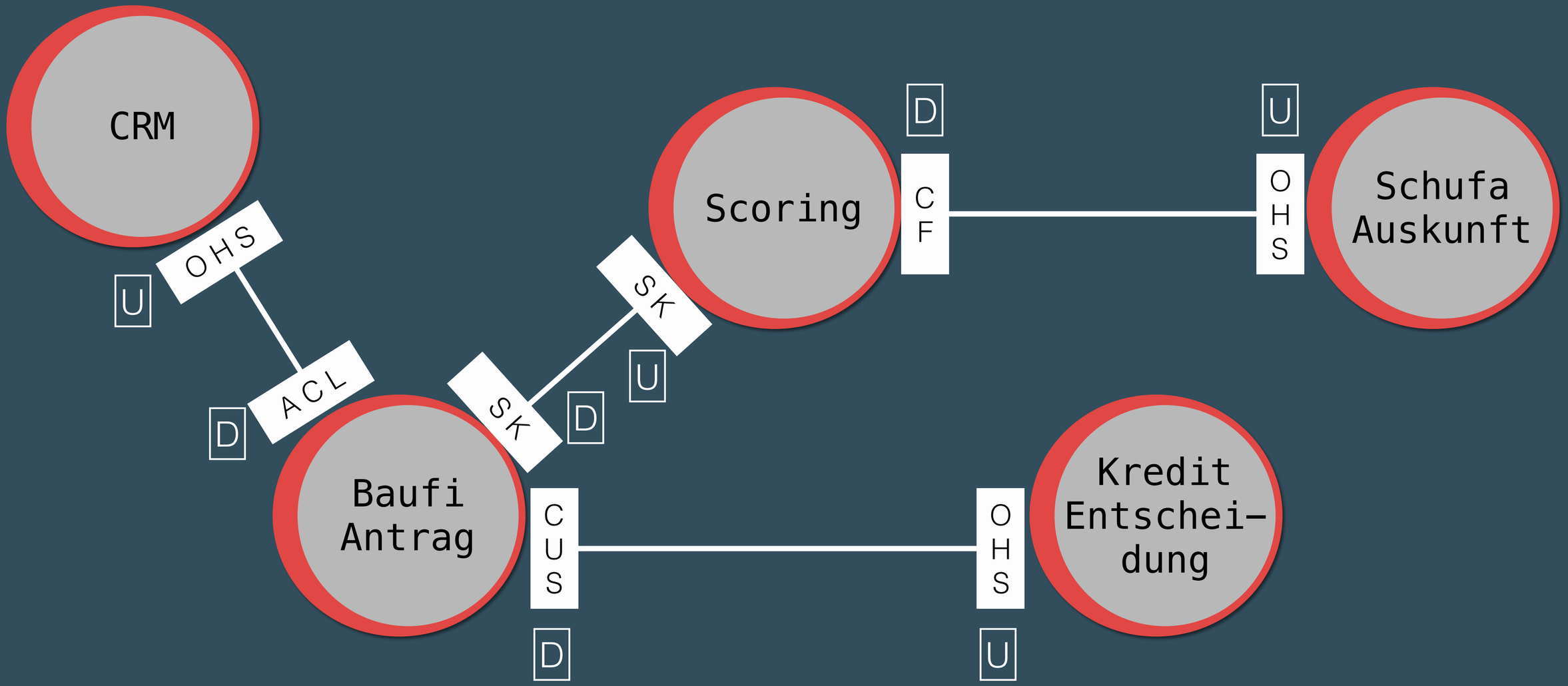
Context Map – Warum?



Aktuell sehen wir hier nur Liefer– und Leistungs–beziehungen

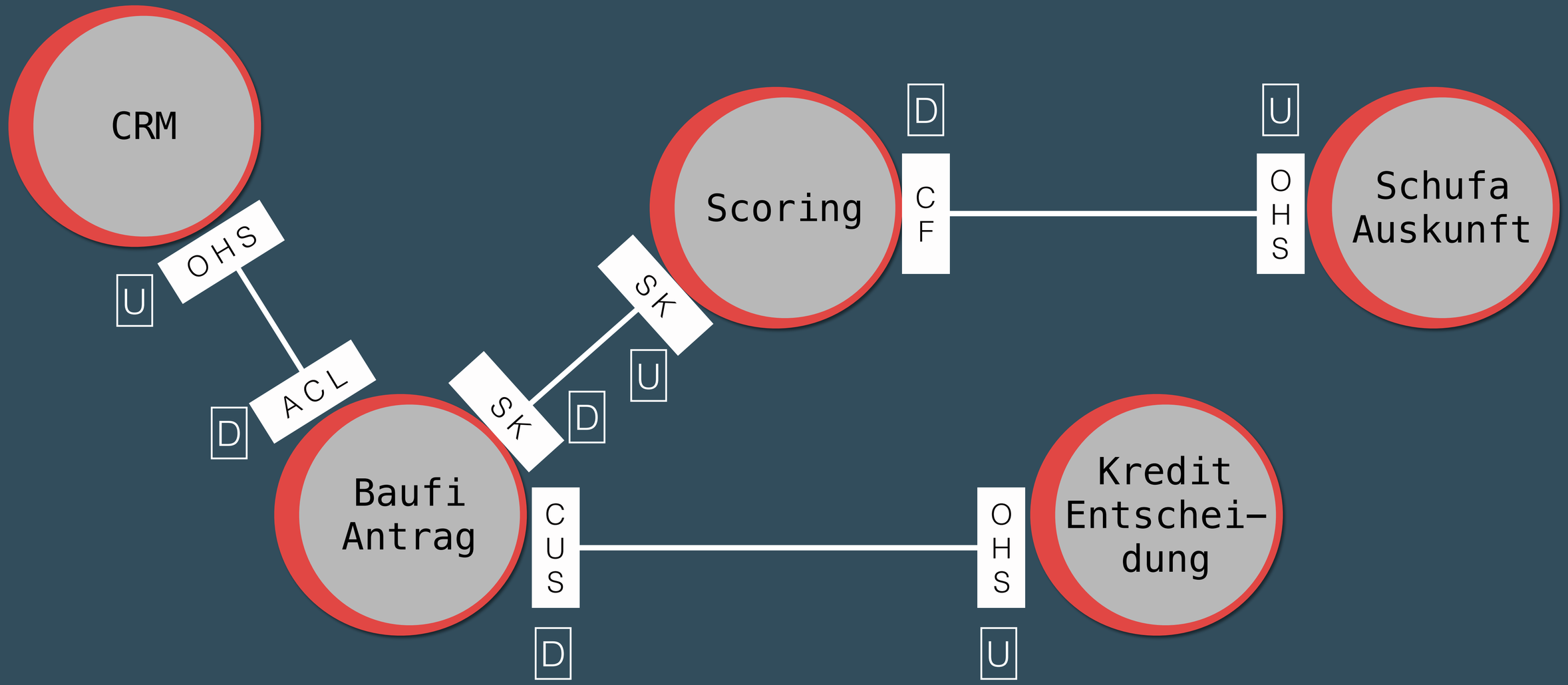
Strategic Design

Context Map



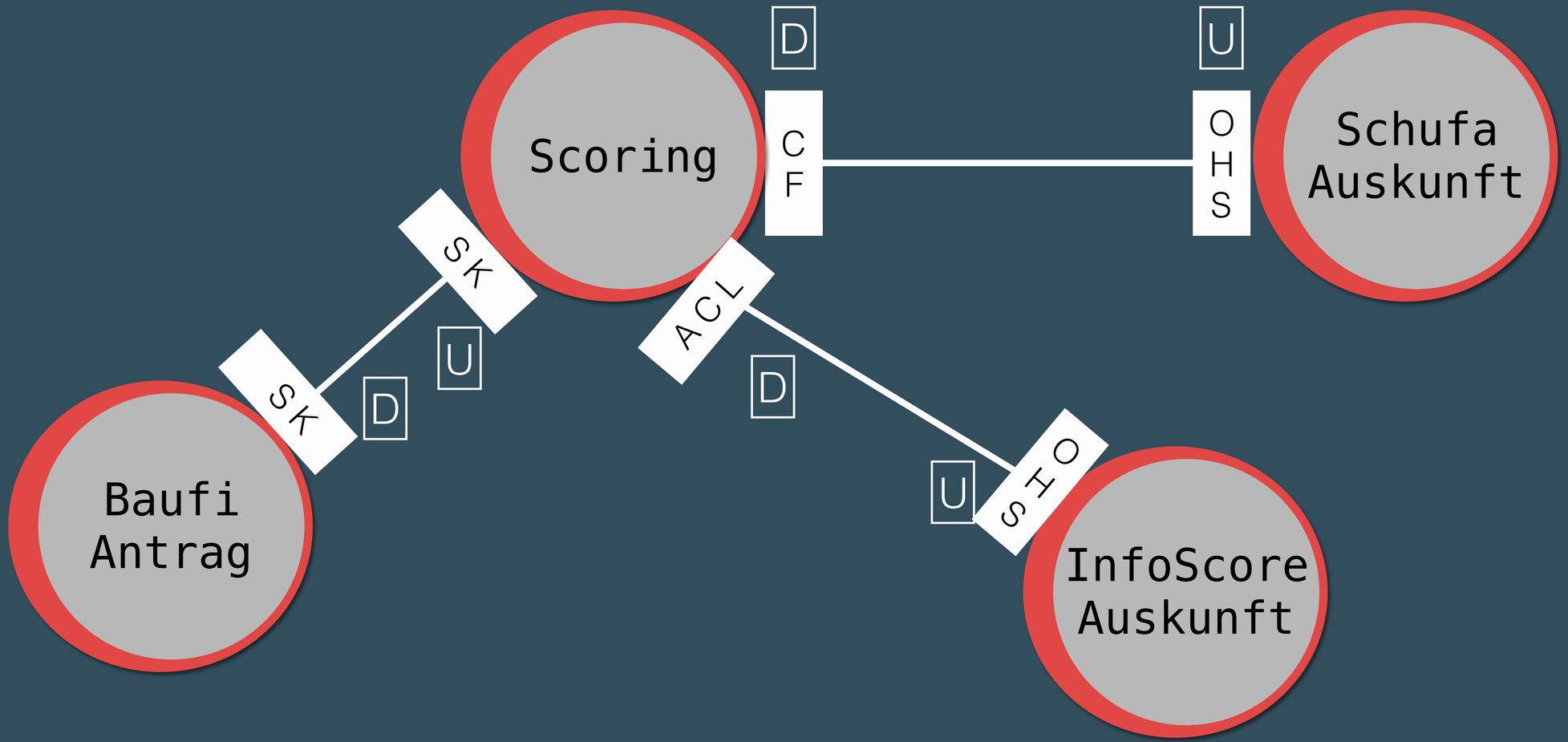
Strategic Design

Context Map: Propagation



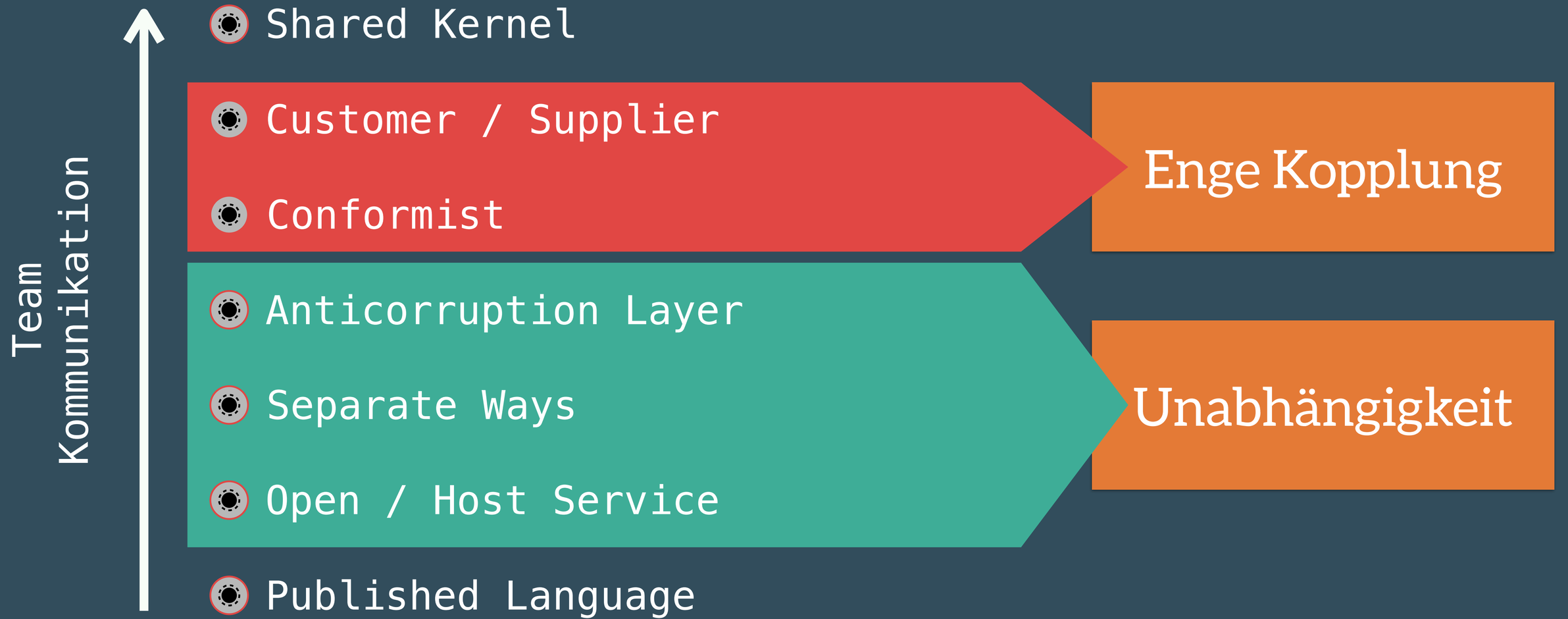
Strategic Design

Context Map: Propagation



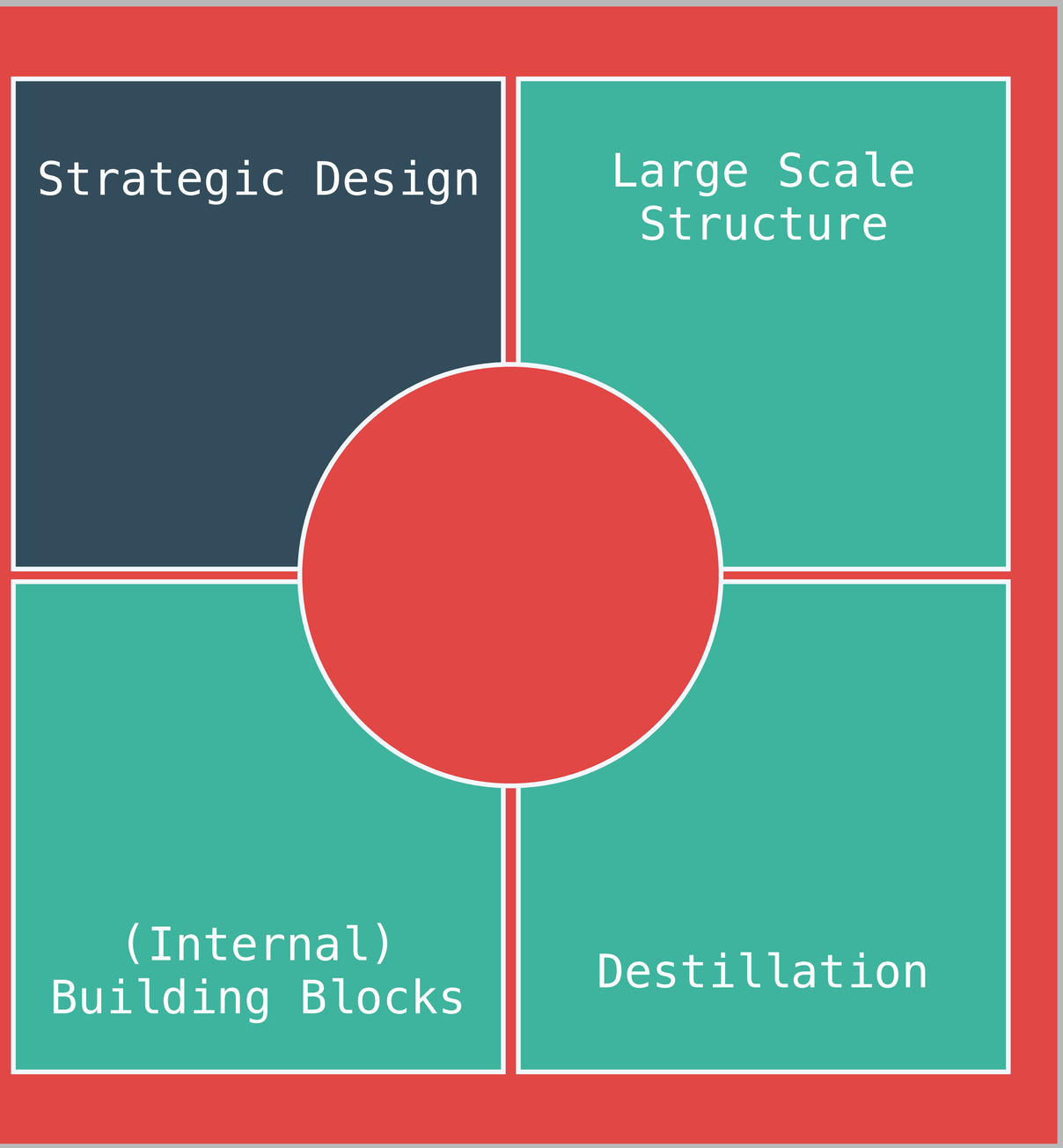
Strategic Design

und Conway’s Law



Domain Driven Design

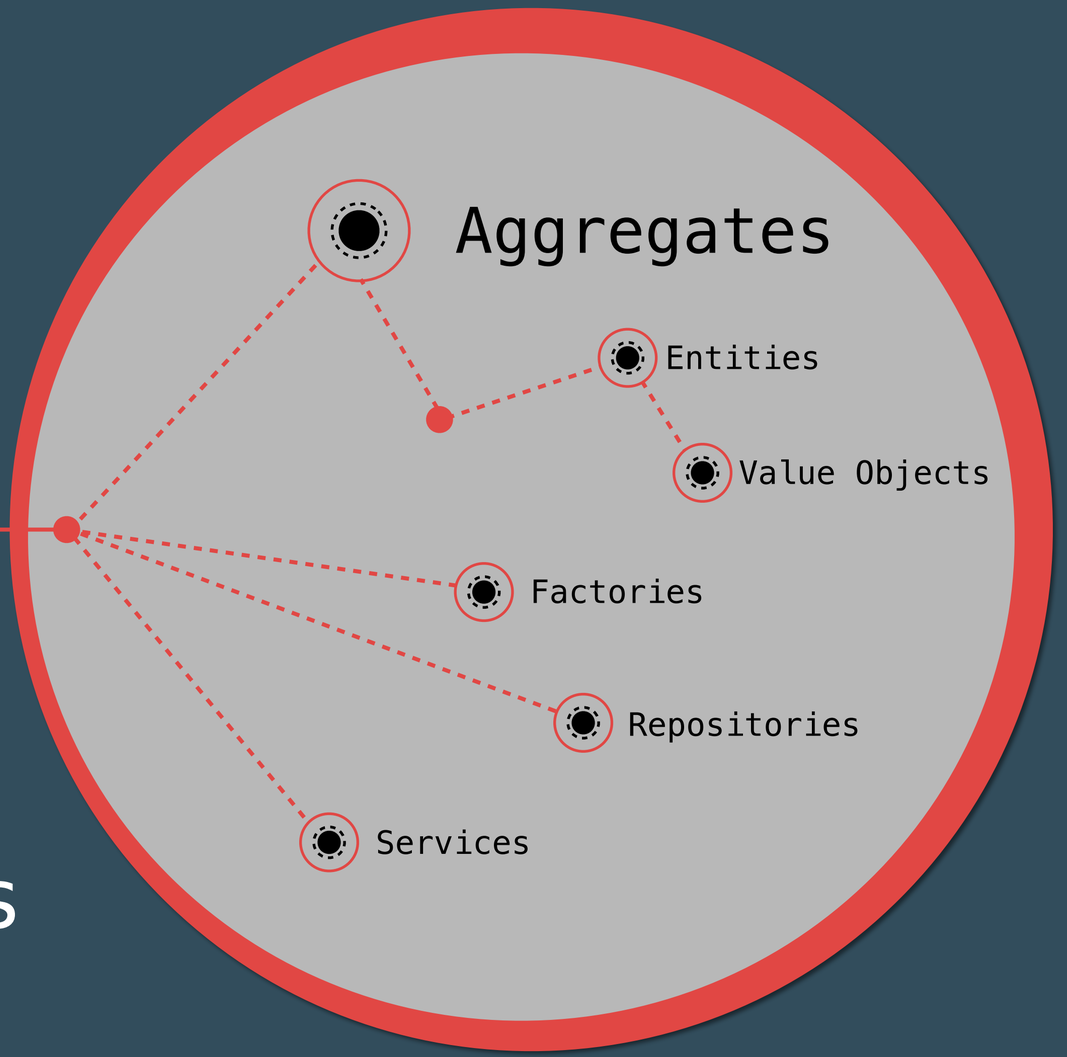
hilft uns im Hinblick auf Microservices in vier Bereichen



(Internal)
Building Blocks

Building Blocks

helfen uns beim Entwurf der Interna eines Bounded Contexts



Building Blocks

Entities

Entities stellen die Kern Business Objekte einer Domäne dar

Jede **Entity** hat eine konstante Identität

Jede **Entity** hat einen eigenen Lebenszyklus

Building Blocks

Value Objects

Value Objects haben keine eigenen Identität, es zählt die Ausprägung der Attribute

Value Objects haben keinen Lebenszyklus, er hängt von der referenzierenden Entität ab

Value Objects sind sehr wertvoll für das Domänen Modell

Building Blocks

Ist „Kunde“ eine Entity der ein Value Object

Grundsätzlich hängt diese Entscheidung vom Bounded Context ab.

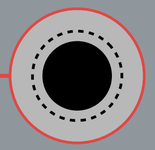
Beispiel: Ein Kunde ist in einer CRM-Anwendung sicherlich eine Entität wohingegen er in der Anwendung für die Badges eher ein Value Object ist.

Building Blocks

Aggregates

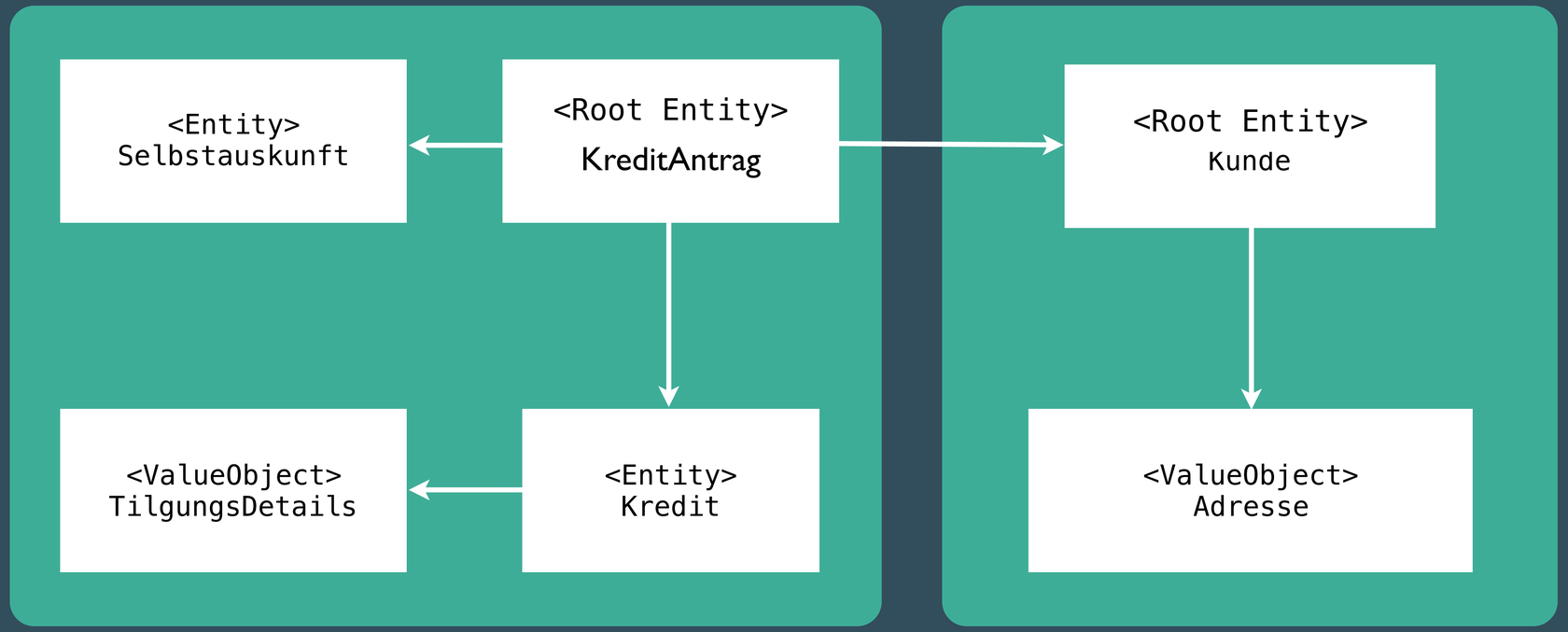
Unterschätze niemals die **Macht des Aggregates**

Building Blocks



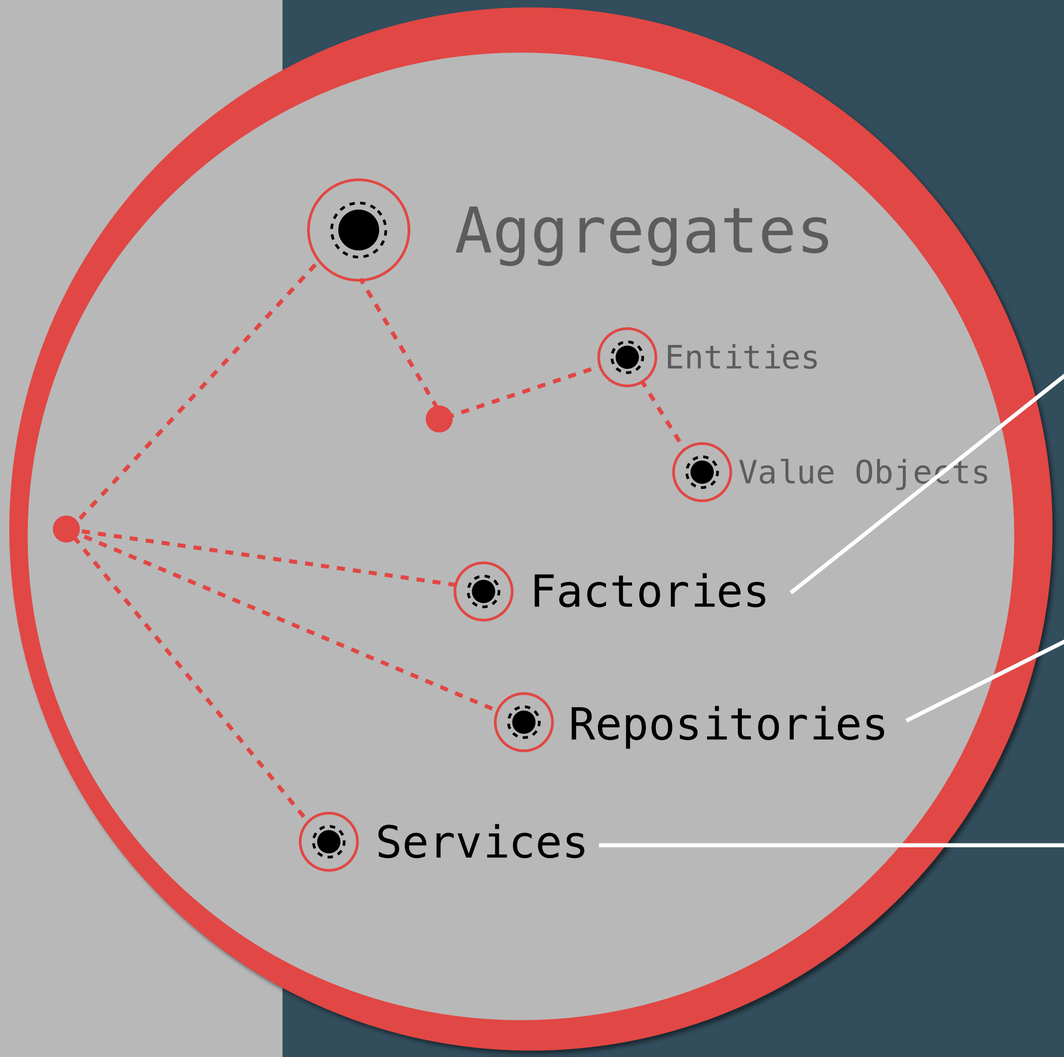
Aggregates

Aggregate gruppieren Entitäten. Die Root-Entity steuert den Zugriff und den Lebenszyklus des Objektgraphen.



Building Blocks

Factories, Services, Repositories



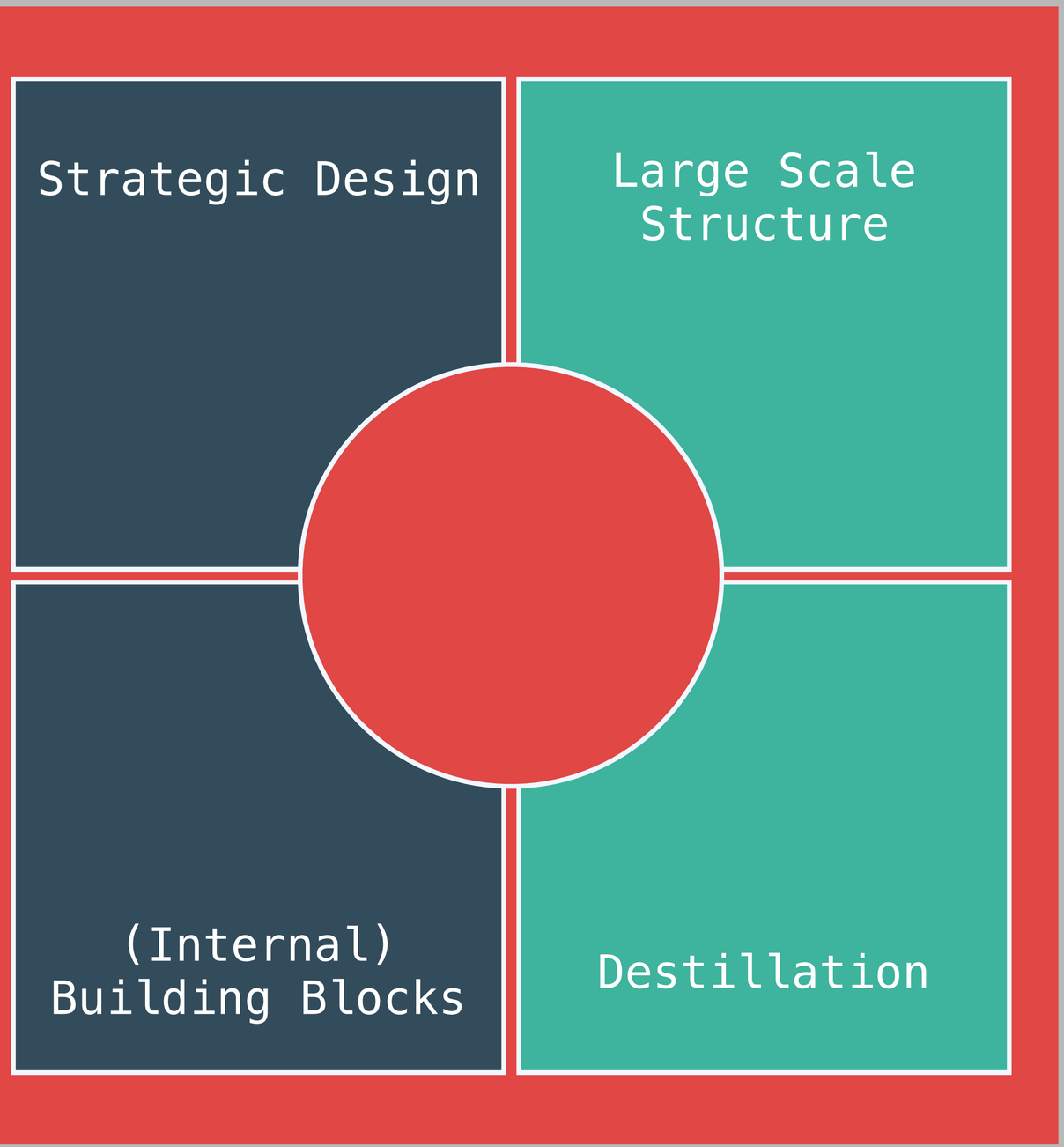
Factories übernehmen die Instantiierung von Entities und Aggregates

Repositories sind für Datenzugriff zuständig

Services implementieren Fachlogik, die mehrere Entities und Aggregate betrifft

Domain Driven Design

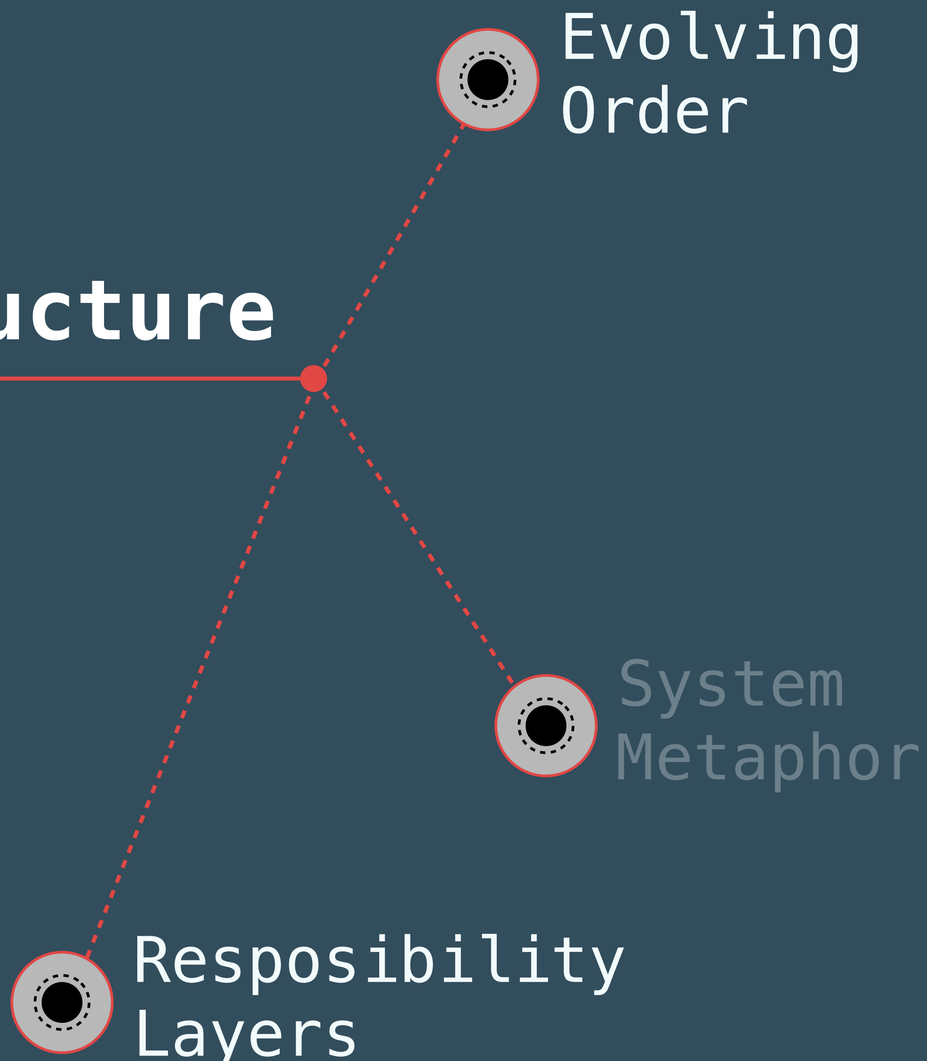
hilft uns im Hinblick auf Microservices in vier Bereichen



Large Scale Structure

Large Scale Structure

hilft bei der Evolution der Microservice Landschaft



Large Scale Structure

Evolving Order

Job Title:
Chief Ivory Tower Architect

Rigide Development Guidelines

Unflexible Architektur

Klare Regeln für jedes kleine Detail

„Ich brauche nur billige Entwickler und übernehme das Denken für Sie“

Large Scale Structure

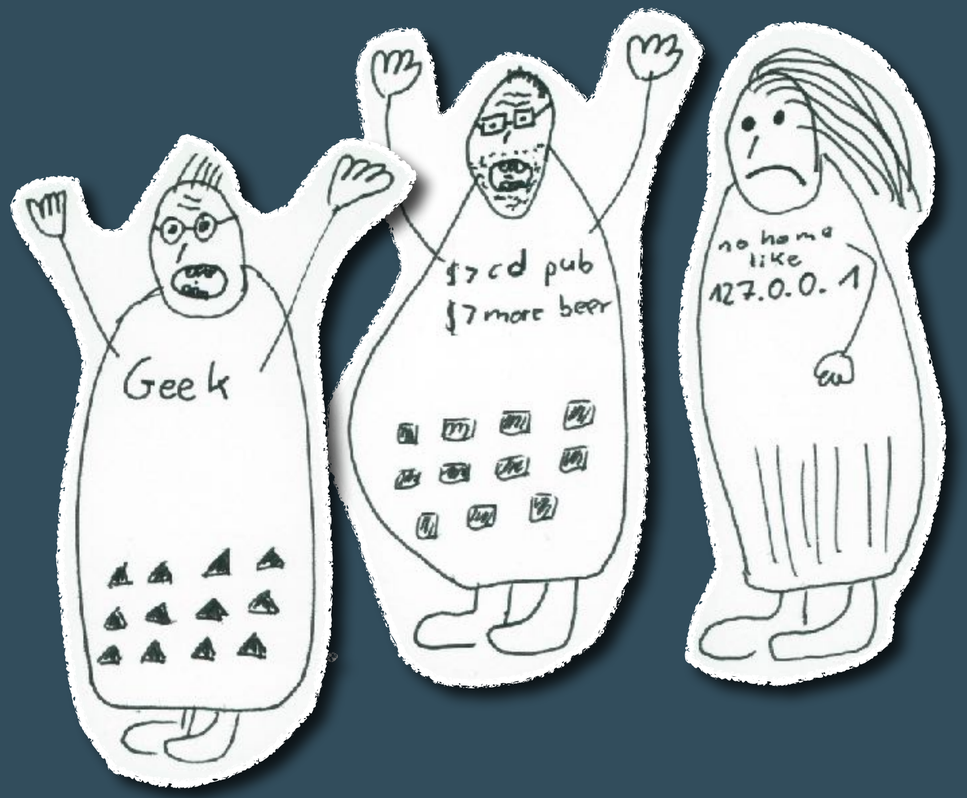
Evolving Order

Entwickler Team

System is zu komplex

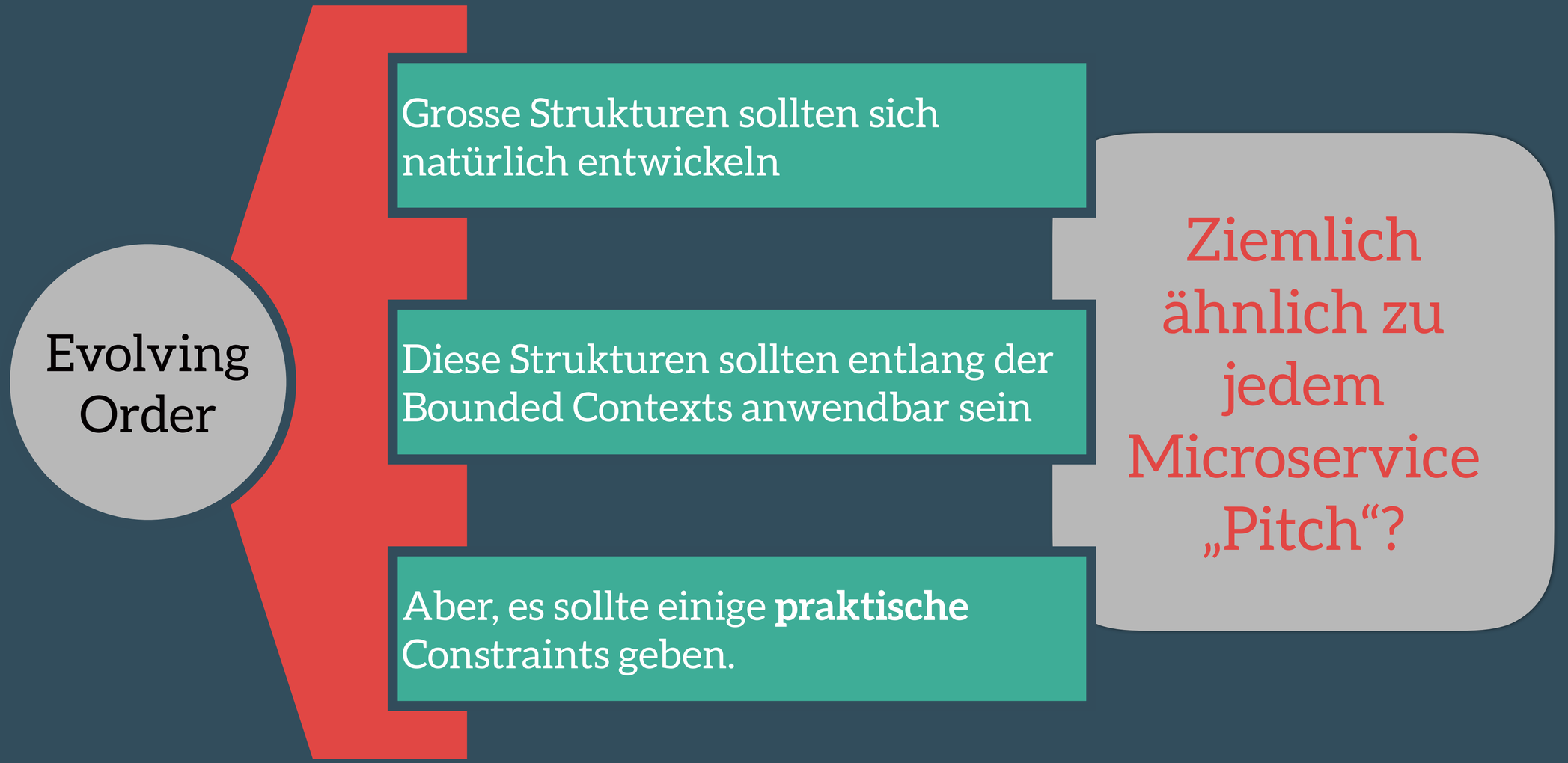
Wir müssen uns darauf fokussieren, die Regeln einzuhalten

Wir benötigen Workarounds um die Regeln zu umgehen



Large Scale Structure

Evolving Order



Large Scale Structure

Responsibility Layers

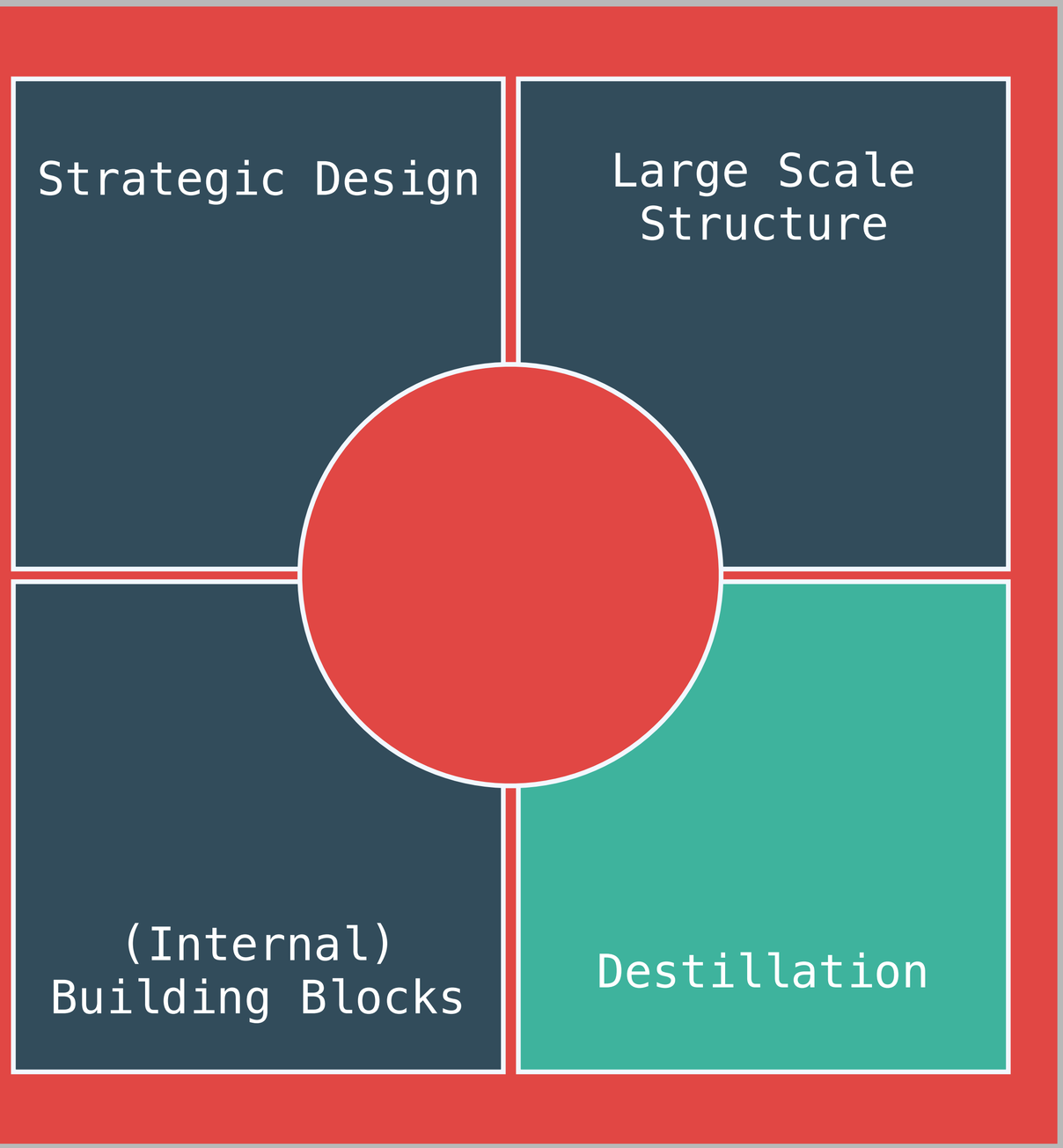
Jeder Microservice ist entlang eines Bounded Context entworfen

Innerhalb der Microservices strukturiert man mit den Building Blocks

Allerdings sollte man zudem Microservices entlang ihrer Zuständigkeiten anordnen

Domain Driven Design

hilft uns im Hinblick auf Microservices in vier Bereichen



Destillation

Destillation

hilft uns bei der Extraktion von Microservices aus einem Monolithen

Destilla-
tion

Identifikation der Kerndomäne

Vision Statement

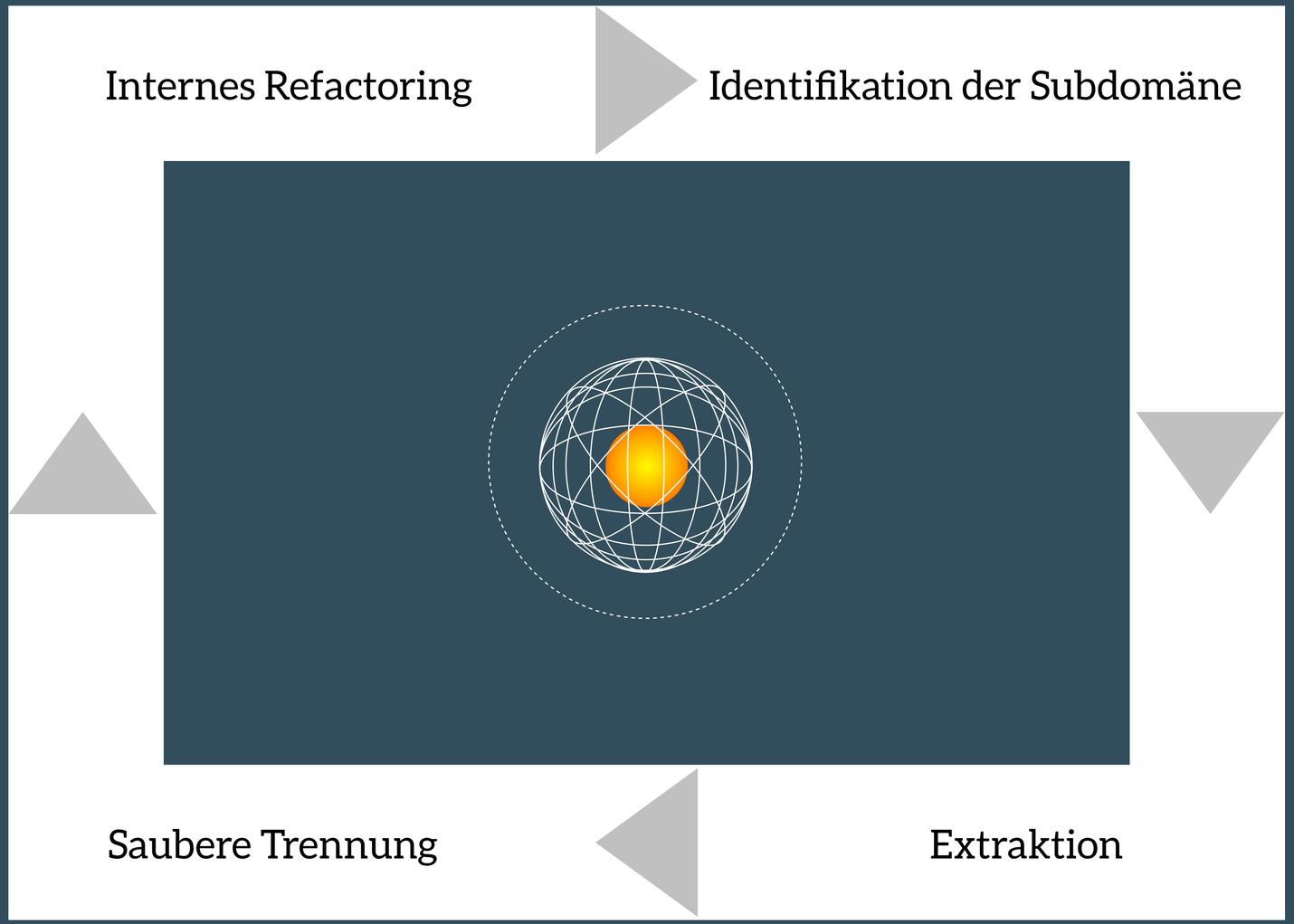
Definiert was Bestandteil der Kerndomänen ist und was nicht

Destillation Document

Beschreibt den Extraktionsvorgang

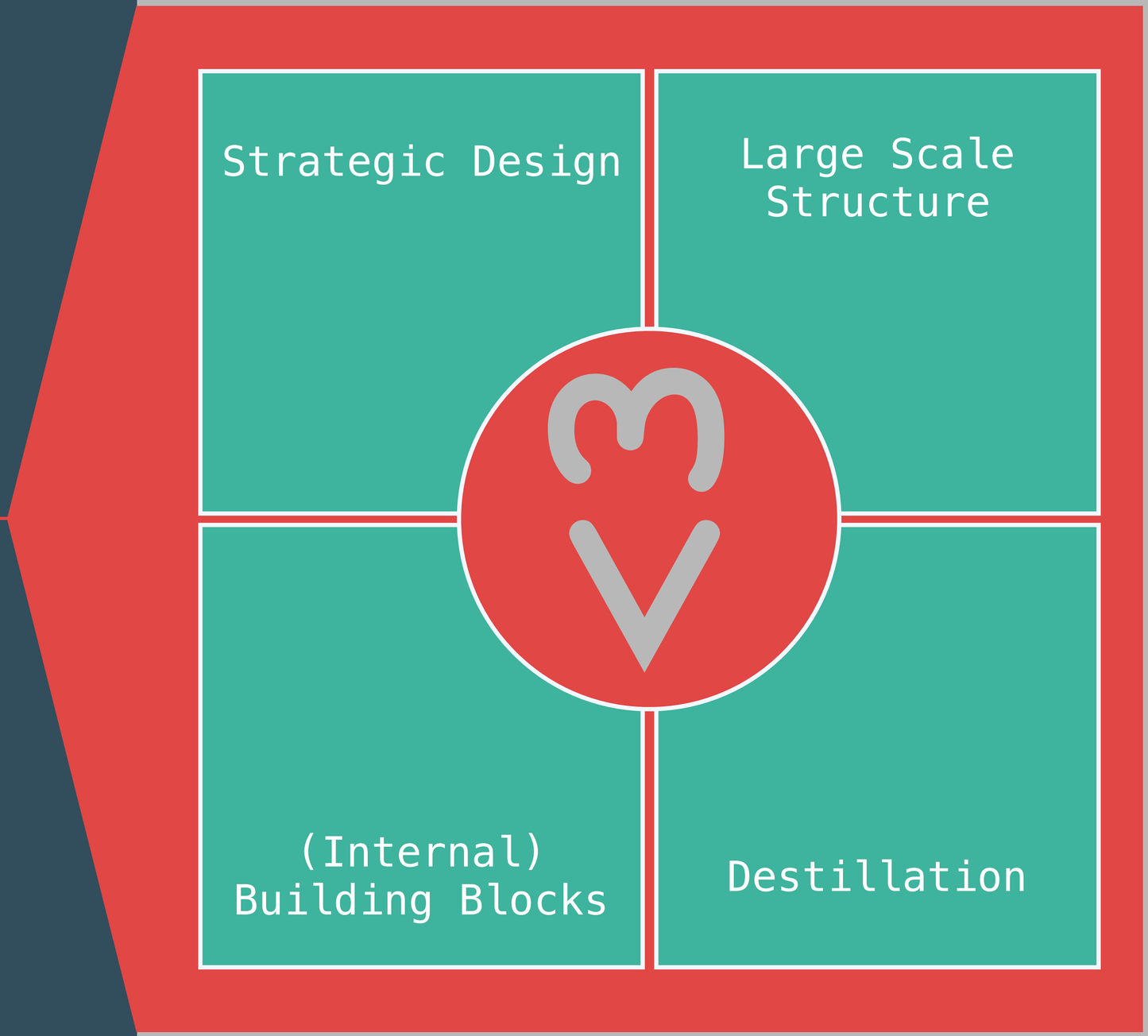
Destilla-
tion

Extraktion von Subdomänen



Microservices

Domain Driven Design



Vielen Dank!

Michael Plöd - innoQ
@bitboss