Visualizing Sociotechnical Architectures with Context Maps

Michael Plöd

INNOQ

Speaker

Michael Plöd
Fellow at INNOQ

Twitter: @bitboss

My assuptions / prerequisites

You know about (Sub)domains, Bounded Contexts, Domain Models and Integration. You don't need to be an expert.

I will talk about

Context Maps: their motivation, the patterns, their use cases, a few heuristics and how to apply them. In addition to that I will also look at other options.

I will NOT talk about

REST, WebServices, Apache Kafka, Microservices or other technical integration stuff.

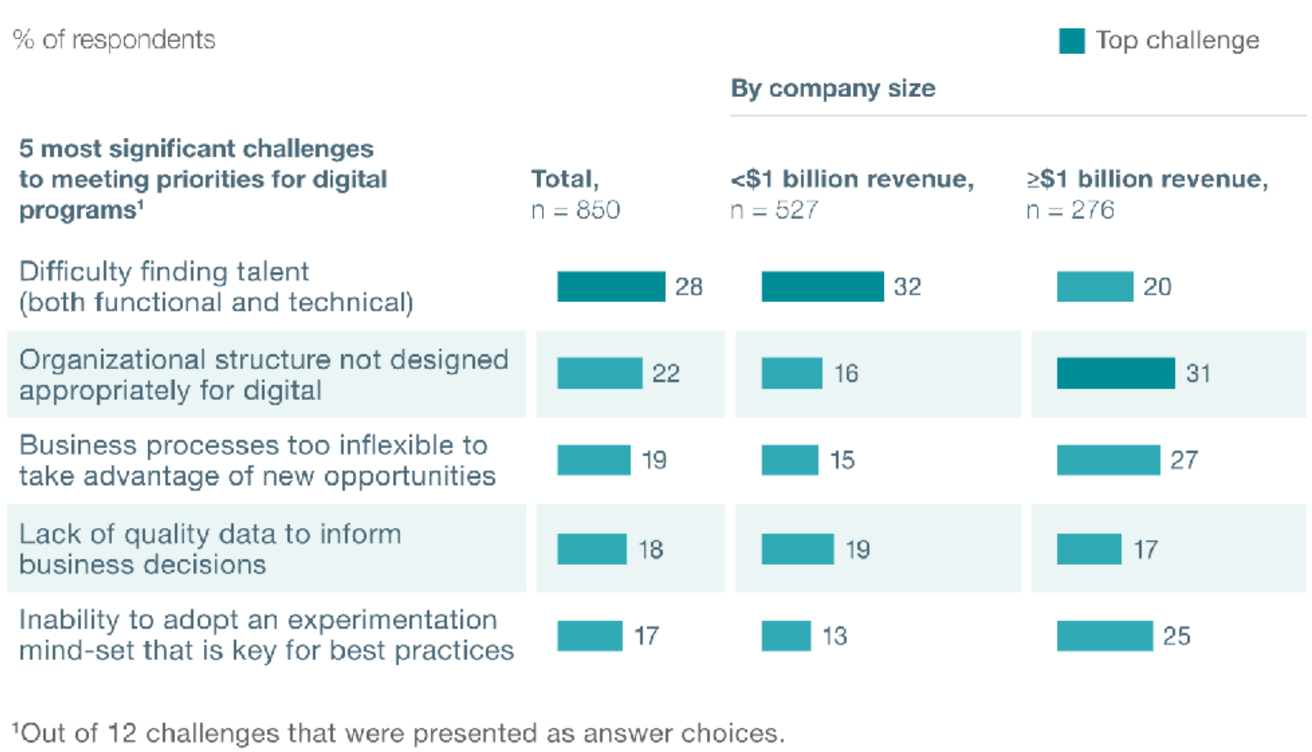
Get my DDD book cheaper

Book Voucher: 7.99 instead of (min) 9.99
http://leanpub.com/ddd-by-example/c/speakerdeck

McKinsey: „At larger companies, structural issues are the top hurdle to meeting digital goals“

Exhibit 5

At larger companies, structural issues are the top hurdle to meeting digital goals.



[1]Out of 12 challenges that were presented as answer choices.

„Focus on organization-wide impact.

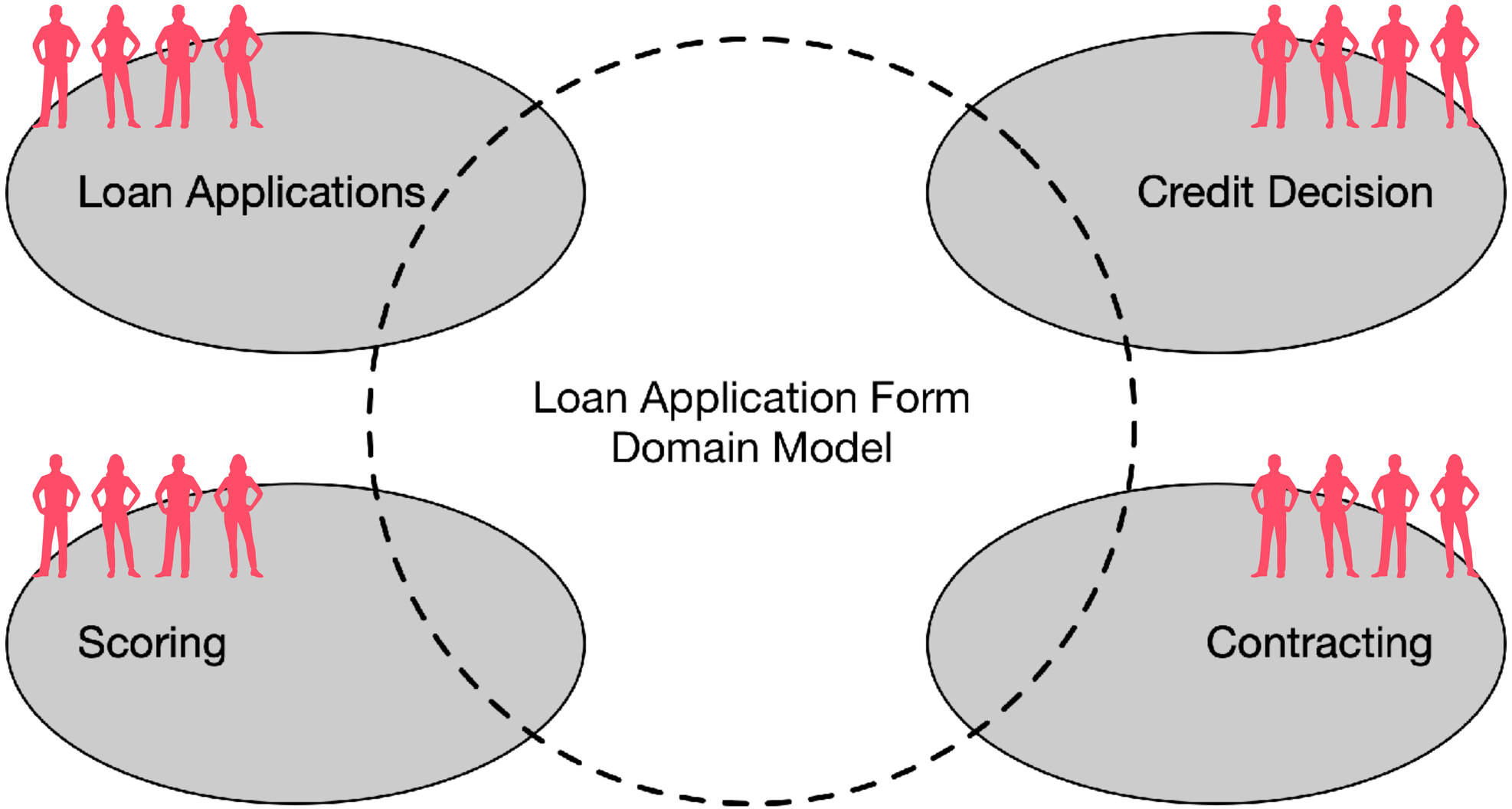
Companies expect much of their near-term growth to be driven by digital, but impact remains elusive. What's more, effective organizational structures, accountability, and meaningful metrics and incentives are largely lacking. As executives become more involved in digital efforts, they must work to ensure that their structures and business processes are set up to take full advantage of the opportunities that digital efforts offer.“

Quelle: https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/the-digital-tipping-point-mckinsey-global-survey-results

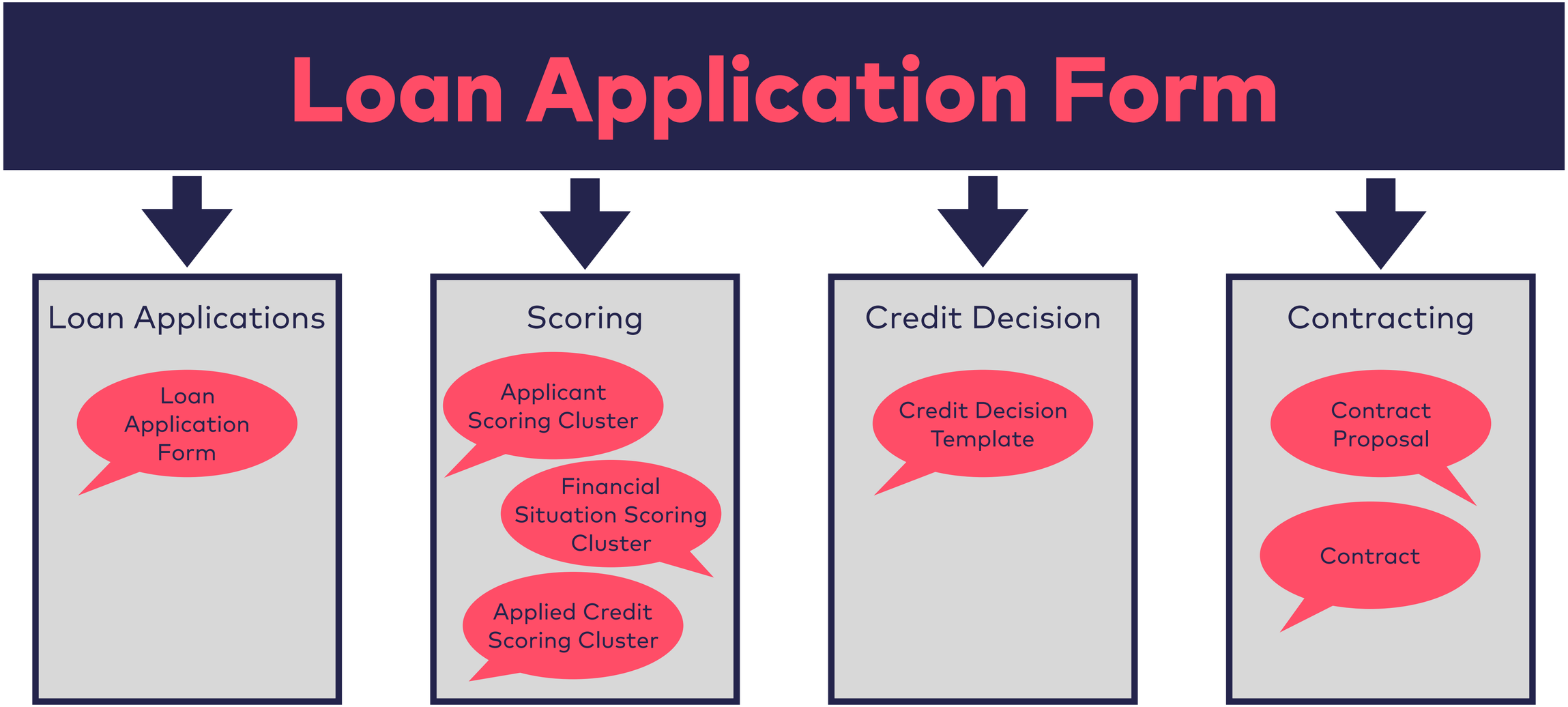
Domain-driven Design

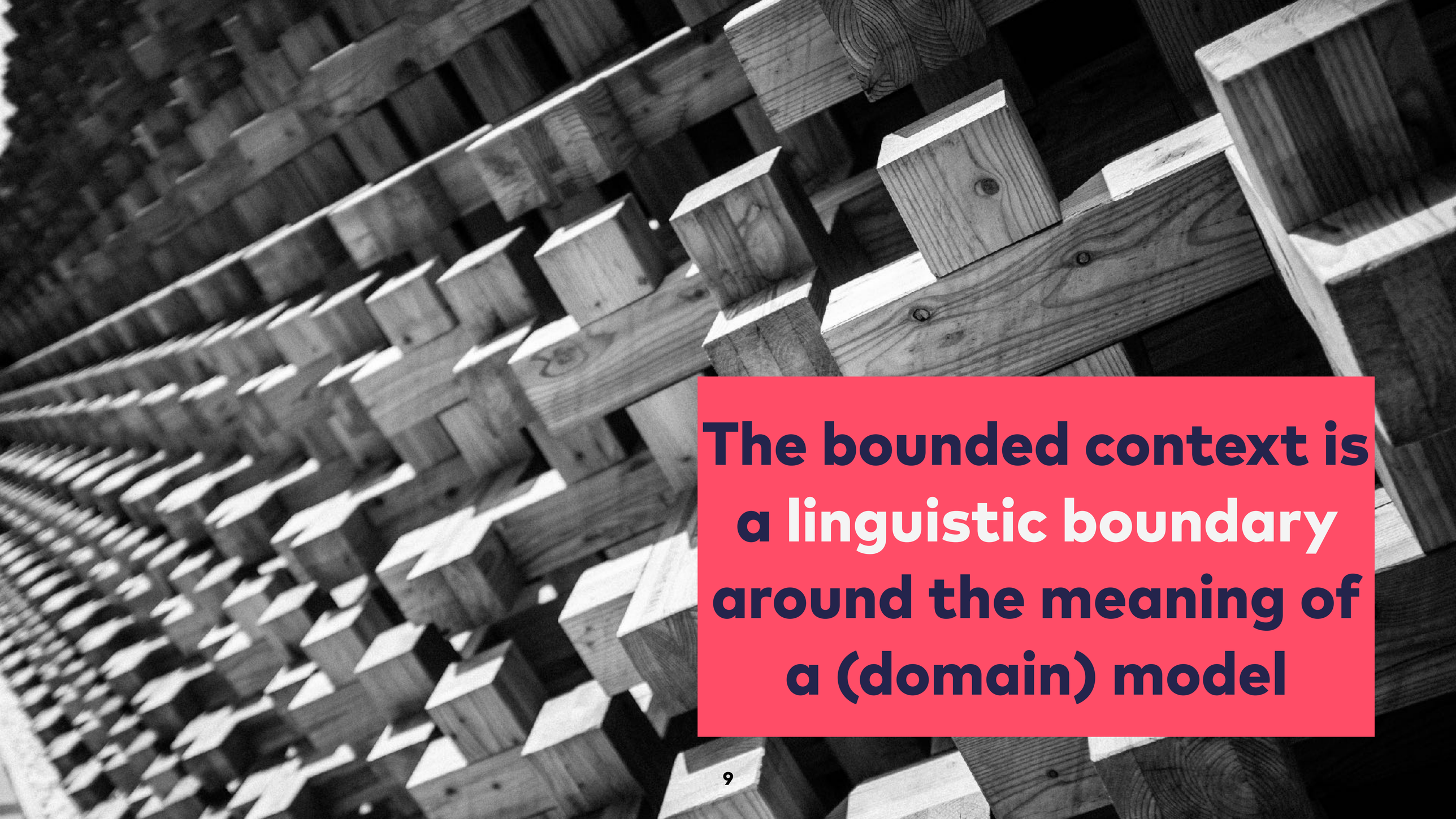
contains sociotechnical aspects which can enable teams to be autonomous and to deliver value at speed by decentralizing governance aspects

Centralized models lead to many dependencies



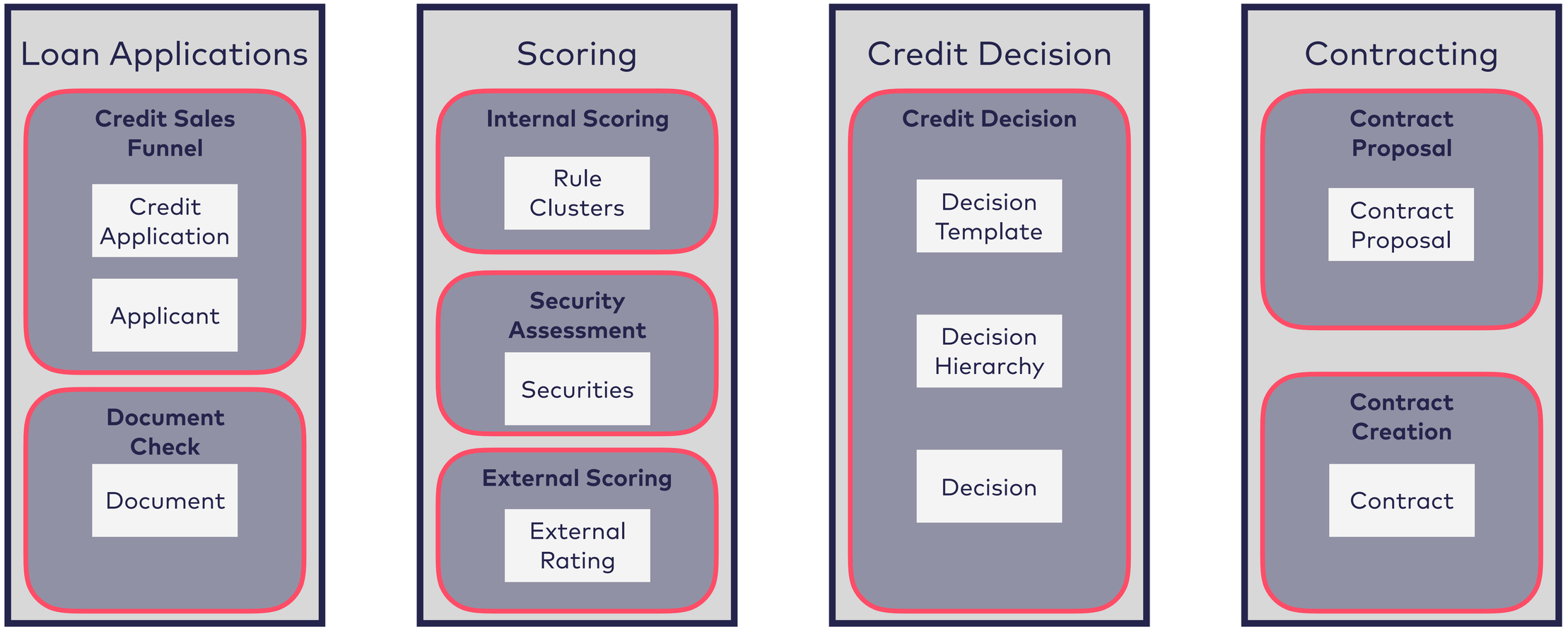
The language for loan application form may differ





The bounded context is a linguistic boundary around the meaning of a (domain) model

In an ideal world we want to align Subdomains and Bounded Contexts

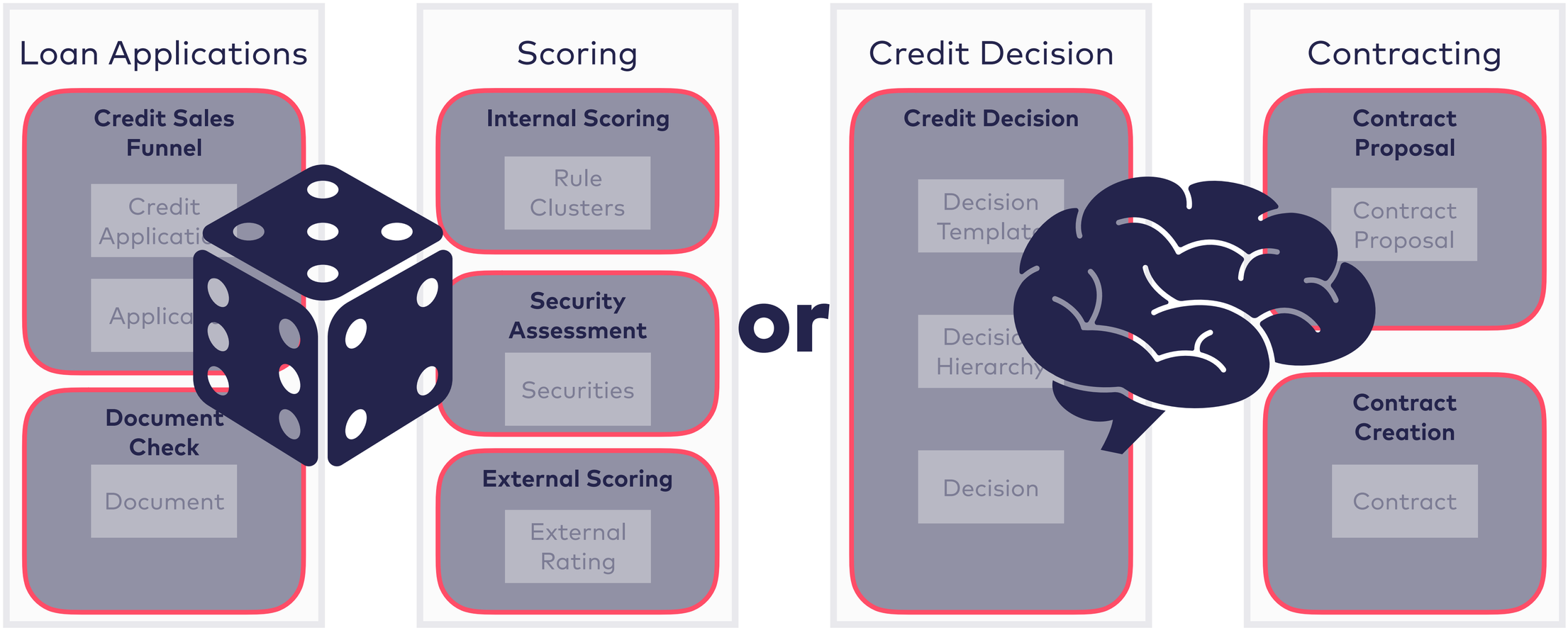


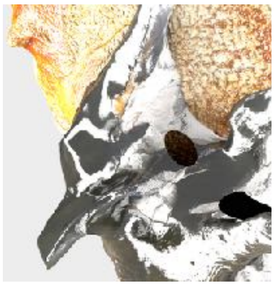
„Team assignments are the first draft of the architecture"

Michael Nygard

Author of „Release It"

Let's say we have a total of 50 people to build teams? How do we distribute them?





Things to keep in mind

Heuristics for team distribution

- 5-9 people per team
- Mind cognitive load of a team
- „You design it, you build it, you run it“
- Mind communication bandwith
- Take subdomains into account

Categories for Subdomains

Core (Sub)domain

- Most important subdomain
- The heart of an organization's business
- Differentiation against competitors and complexity

Supporting Subdomain

- Vital for the operating of core (sub)domains
- Lack of strategic relevance with regards to competition
- Can be complex or simple

Generic Subdomain

- Needed functionality, which is not critical at all
- „We need some solution of problem x"
- No way to differentiate

The categorization of subdomains should have an impact on staffing, make or buy decisions and even IT-procurement

Core (Sub)domain

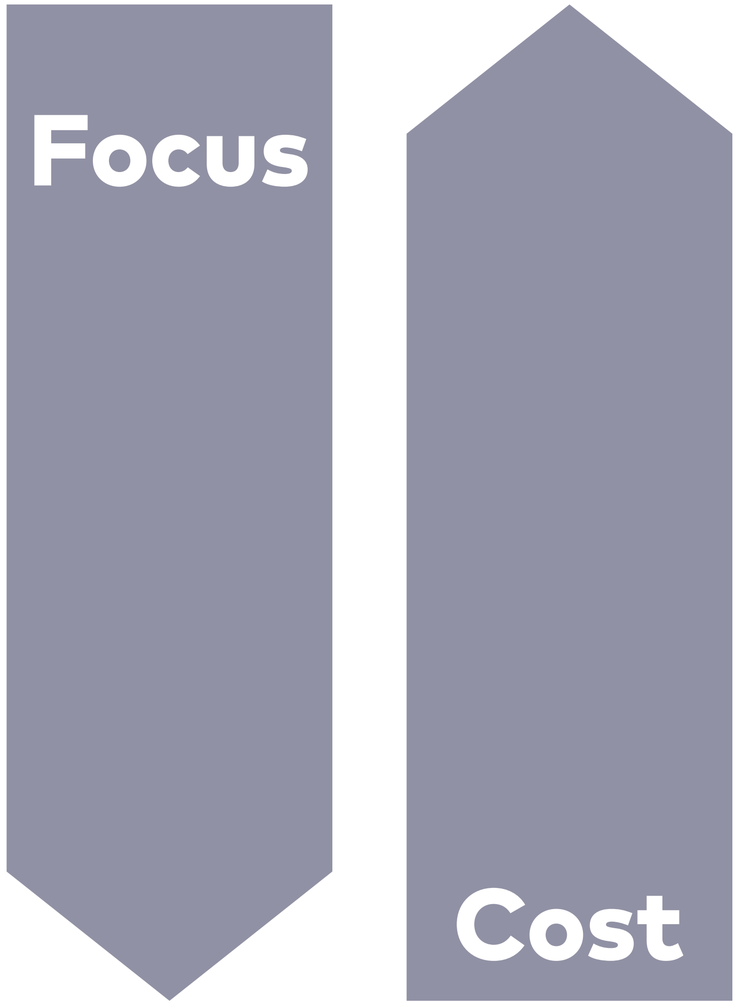
- In-House development
- Excellent teams with excellent working conditions
- Staffing strategy should be „insourcing“
- No Near- / Off-Shore development
- No „Custom Off The Shelf“ (COTS / Standard) Software

Supporting Subdomain

- In-House development with external support is possible
- Some Near-Shore development is an option
- „Custom Off The Shelf“ (COTS / Standard Software) products that fits your processes with minor customizations
- Avoid complete outsourcing and / or major customizations of COTS products

Generic Subdomain

- Look of COTS or SaaS Products that get „the task done“
- Staffing strategy can be „outsourcing“
- Look for cost saving opportunities
- Don’t put your best employees into generic subdomains
- Vendor Management is a key skill in this area



Wardley Maps can also provide valuable insights

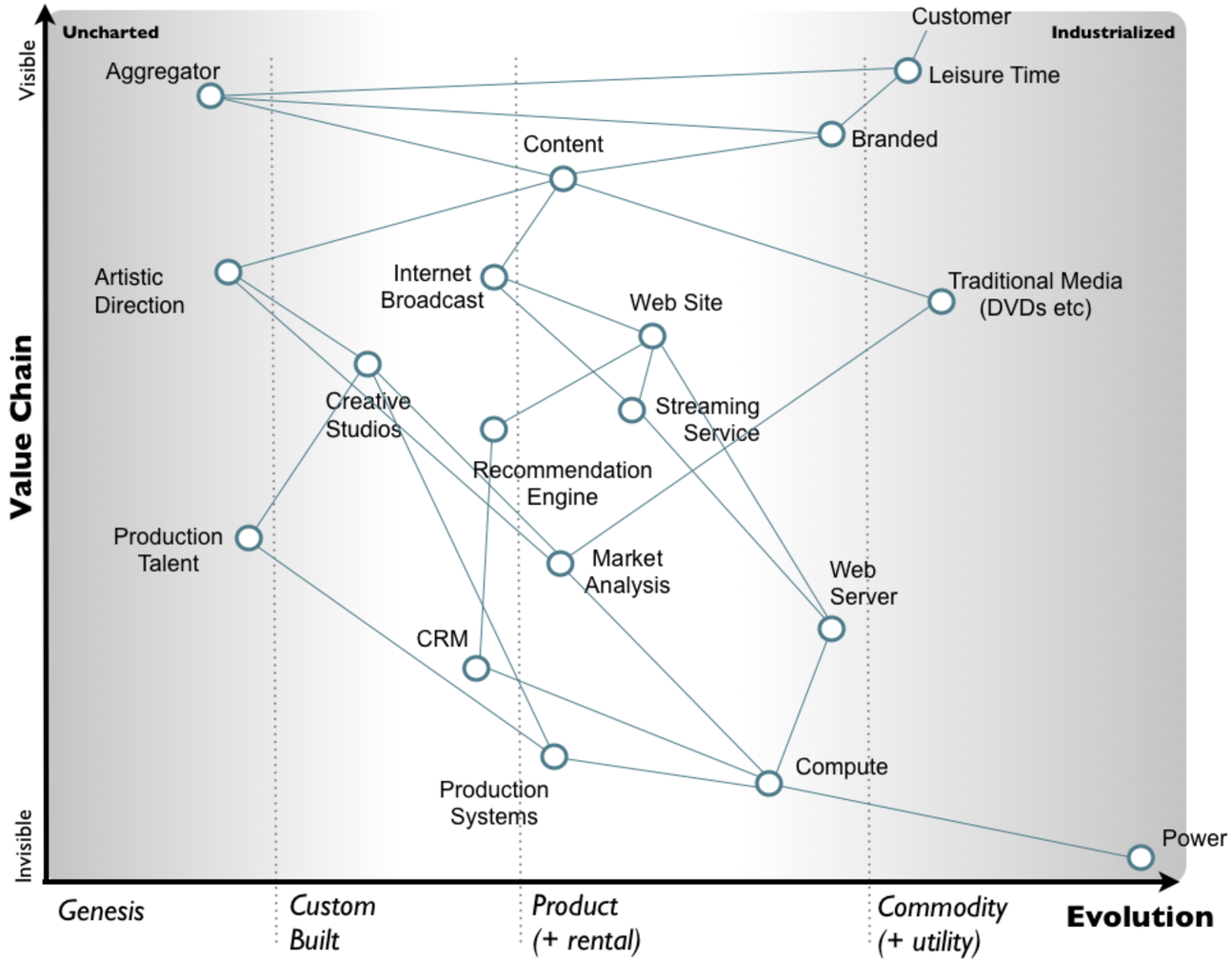
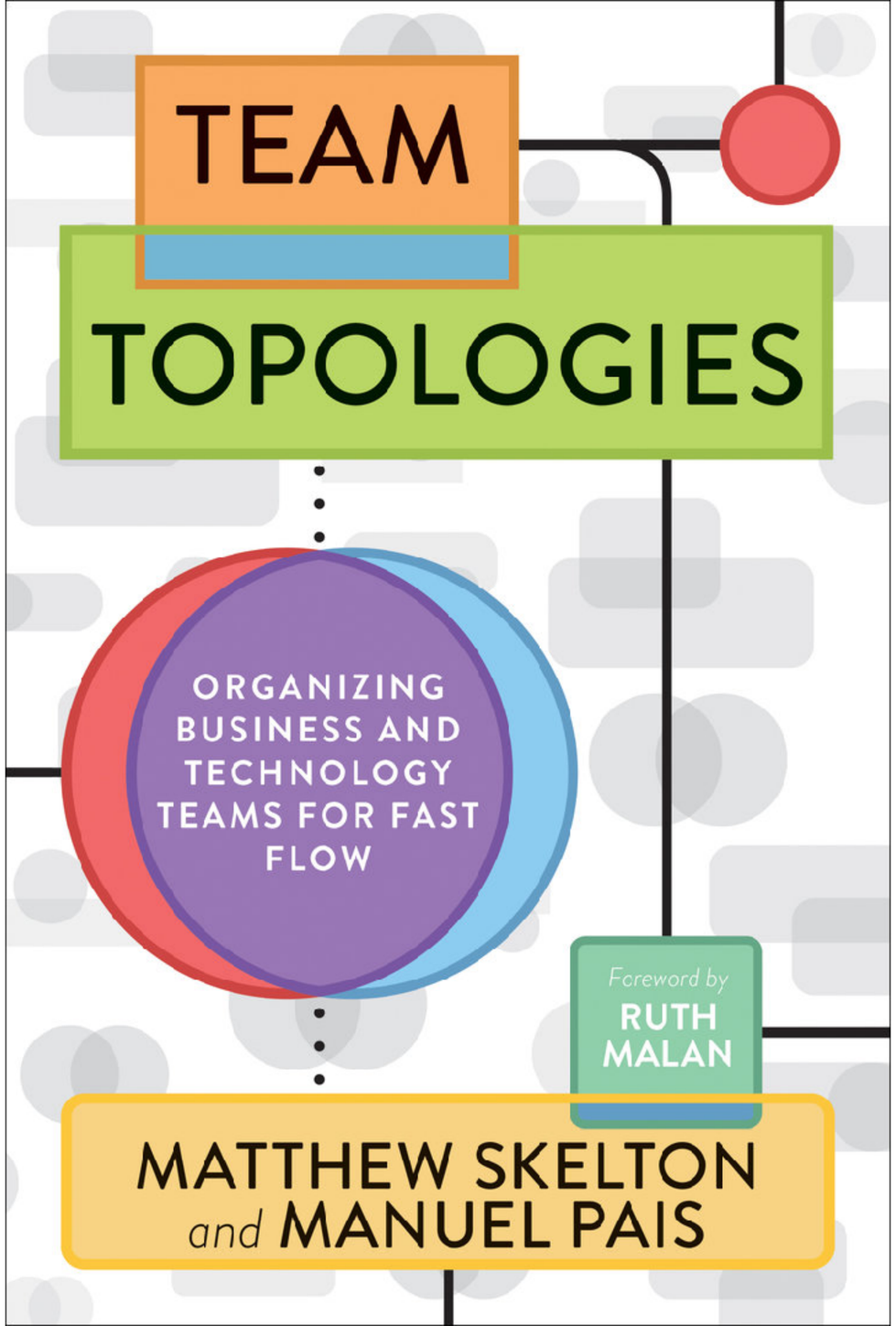
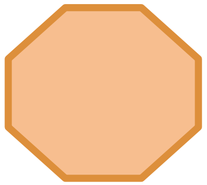


Image Credit: https://medium.com/@rbouma/lean-agile-scotland-2016-c3756adcb18d

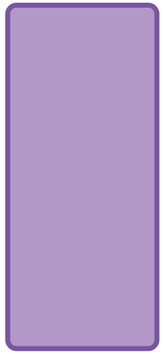
Make sure that your core (sub)domains and / or your most valuable contexts in genesis and custom built phases are perfectly staffed



Fundamental Team Topologies



Complicated Subsystem



Enabling

Platform

Stream-aligned

Subdomains + Team Topologies

Core (Sub)domain

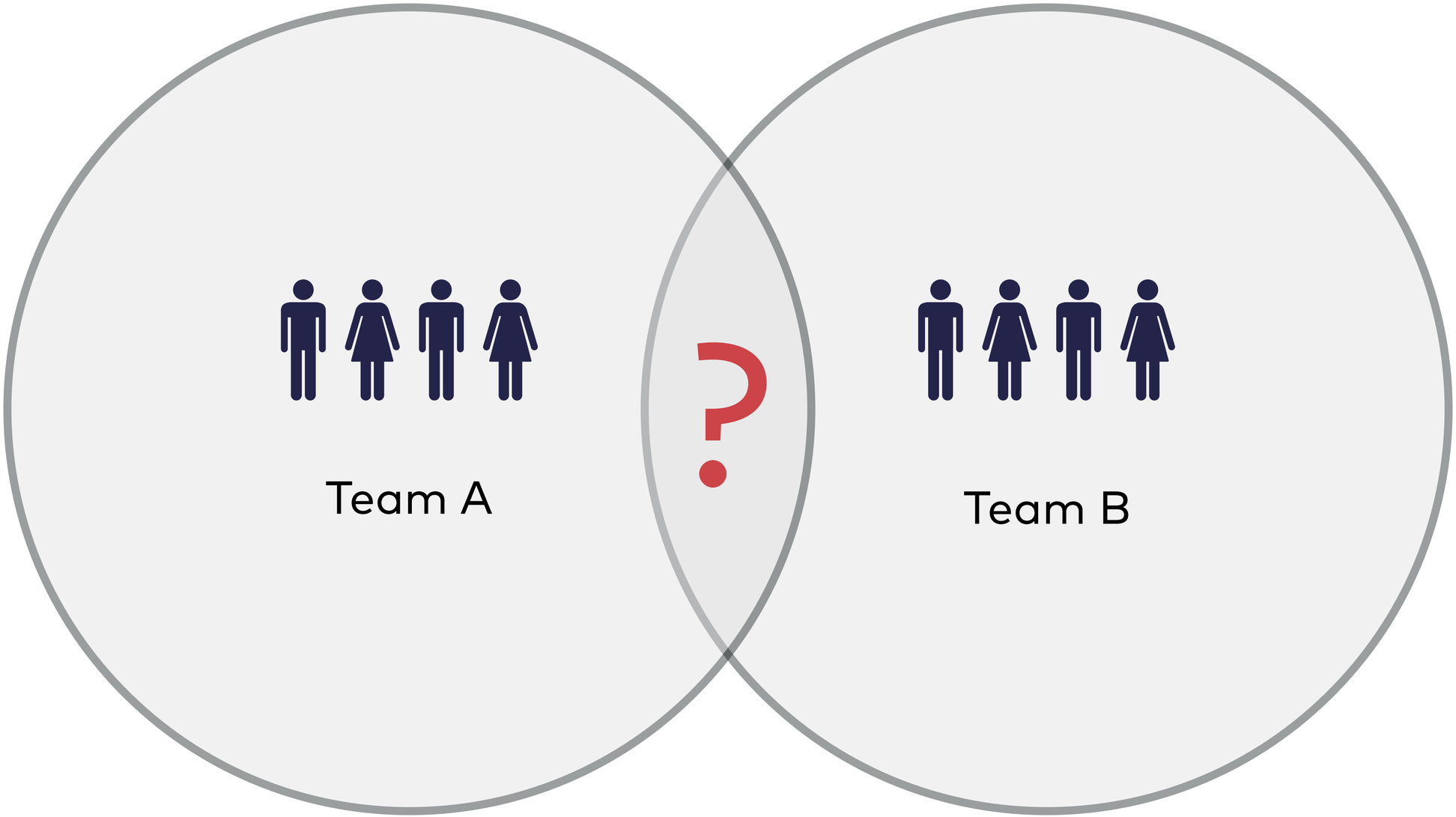
Supporting Subdomain

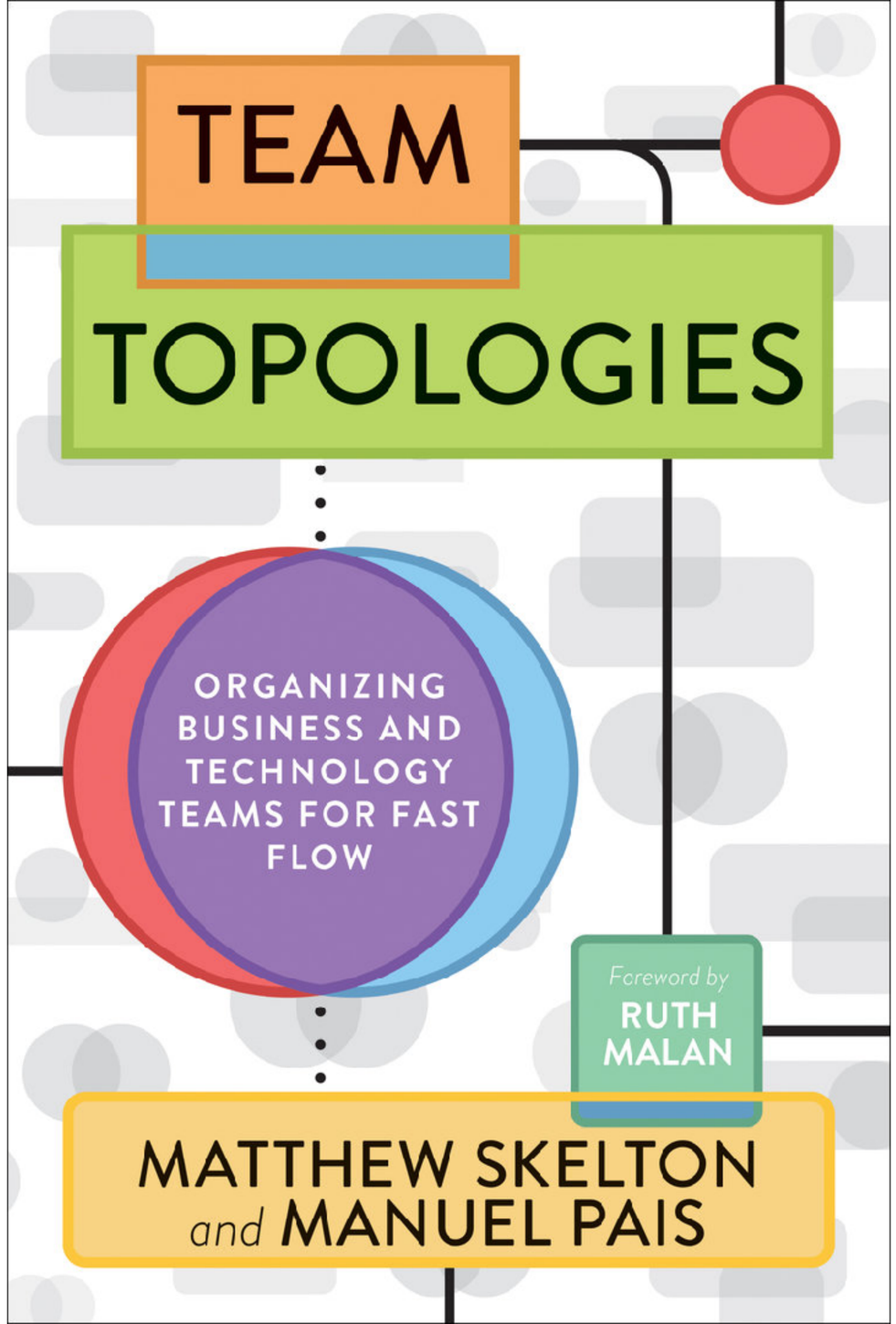
Generic Subdomain

Coupling of bounded contexts

Although bounded contexts aim for a high degree of decoupling there needs to be some kind of connection between:

- The teams
- The software
- The business capabilities

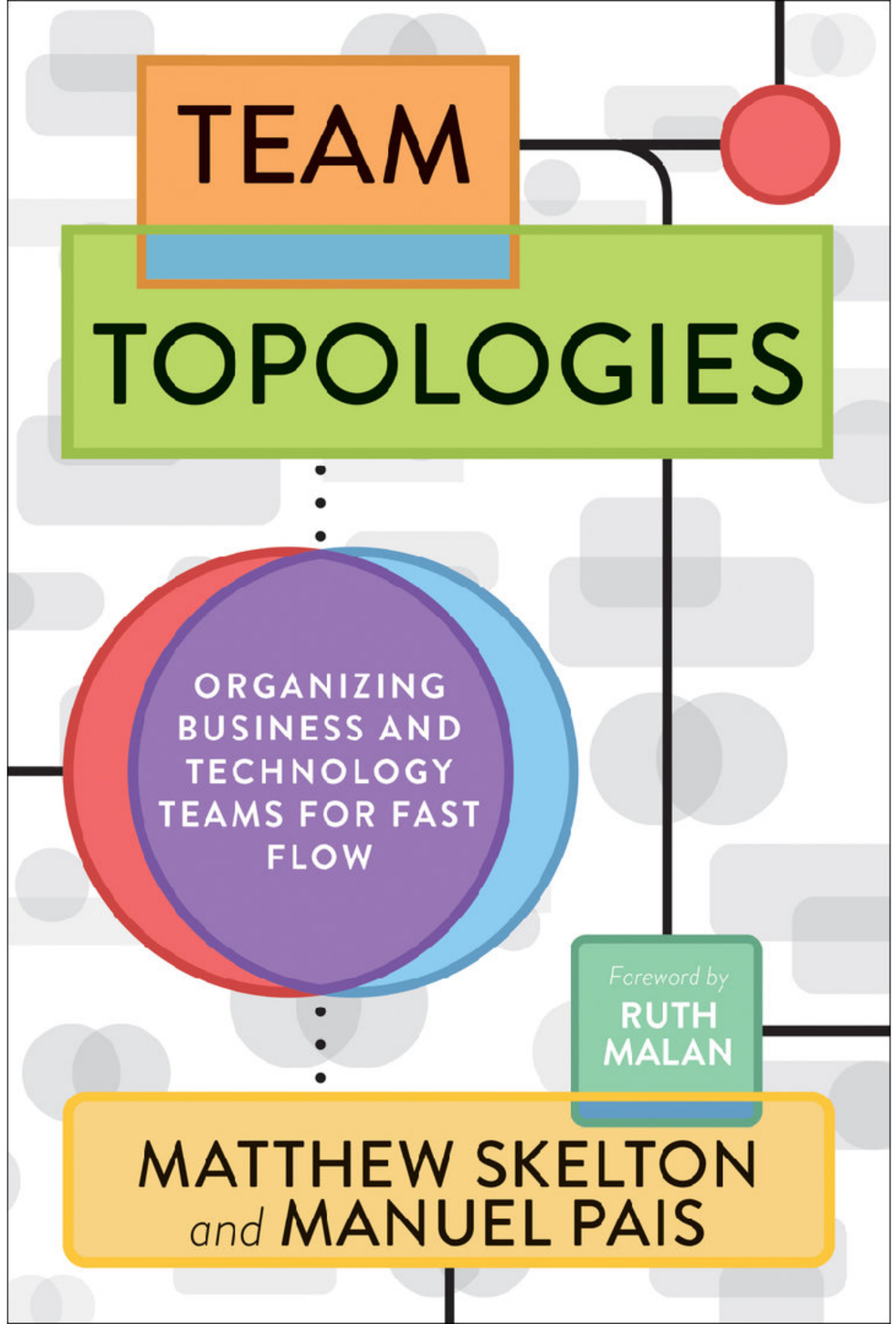

Team A
?
Team B



"An architect should be thinking:

Which team interaction modes are appropriate for these two teams?

What kind of communication do we need between these two parts of the system, between these two teams?**"**



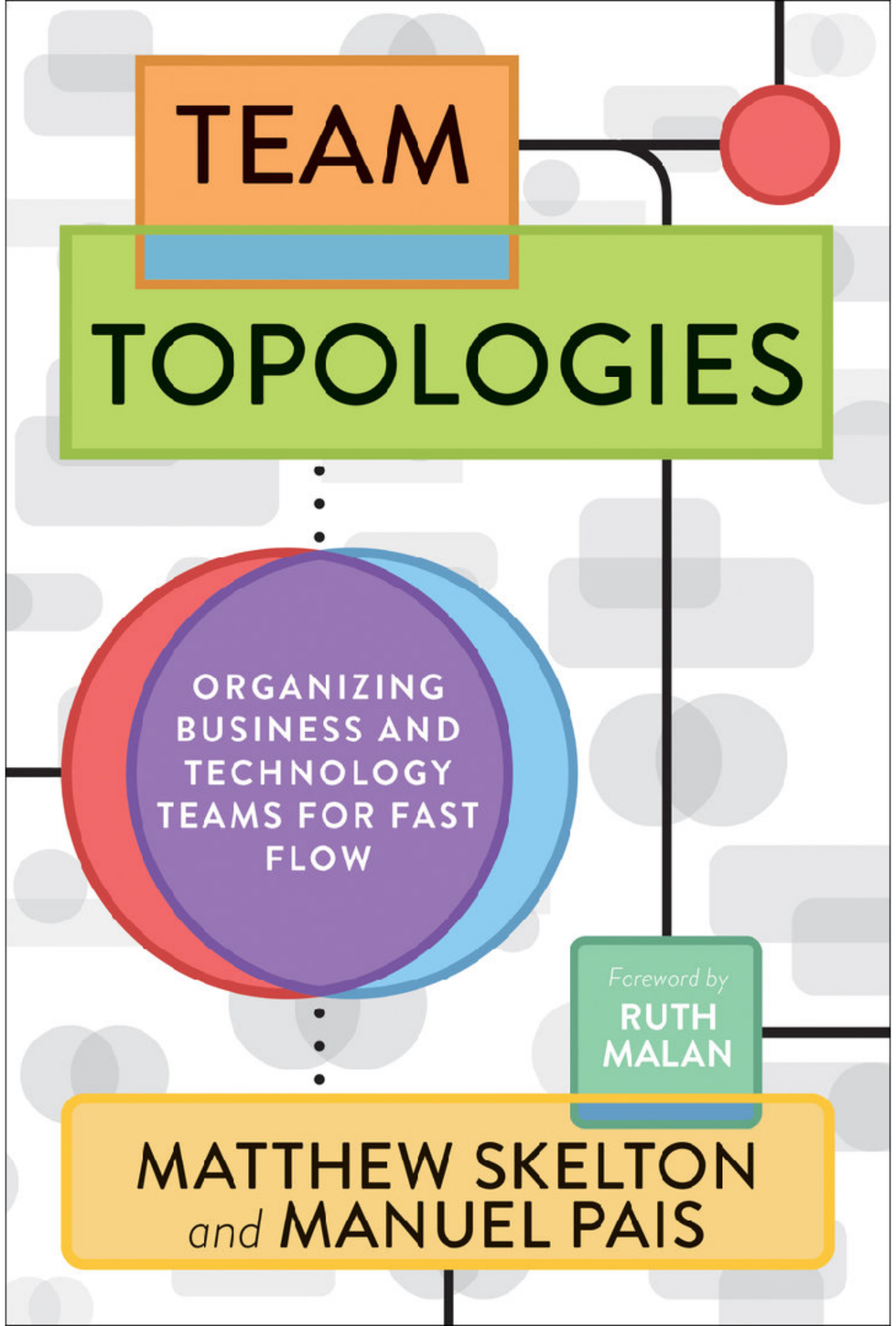
Team Interaction Modes

Collaboration

X-as-a-Service

Facilitating

Image taken from the Team Topologies book



Team Interaction Modes

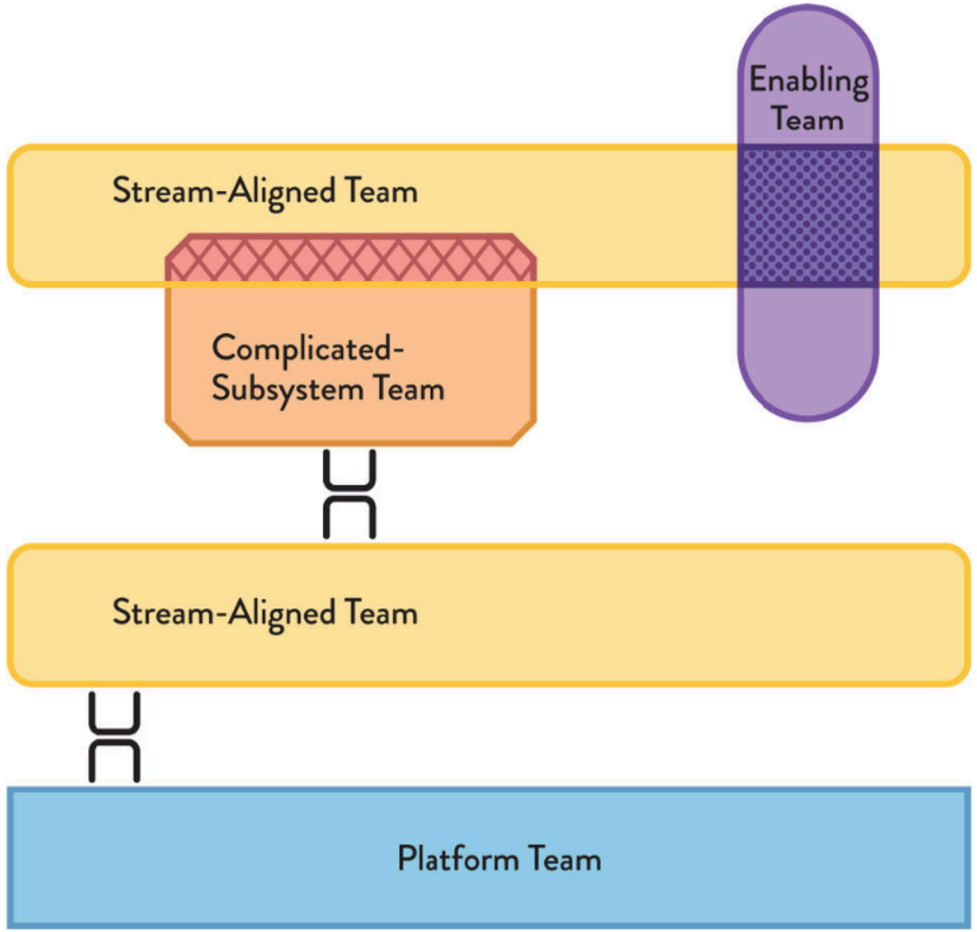
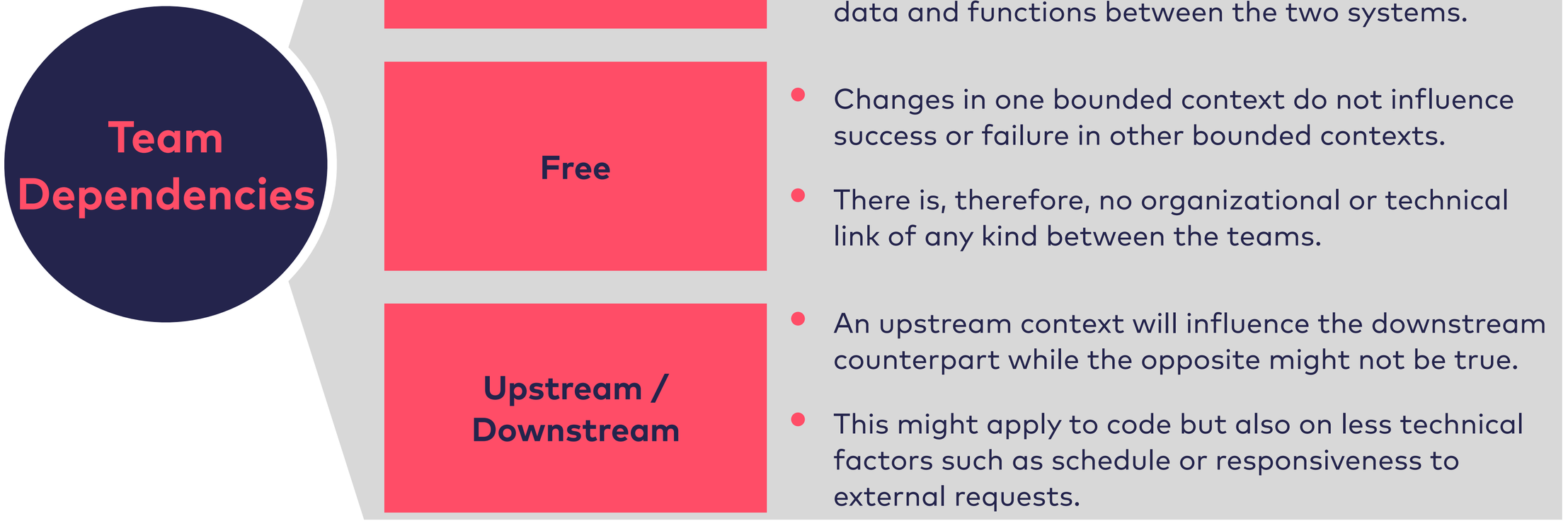


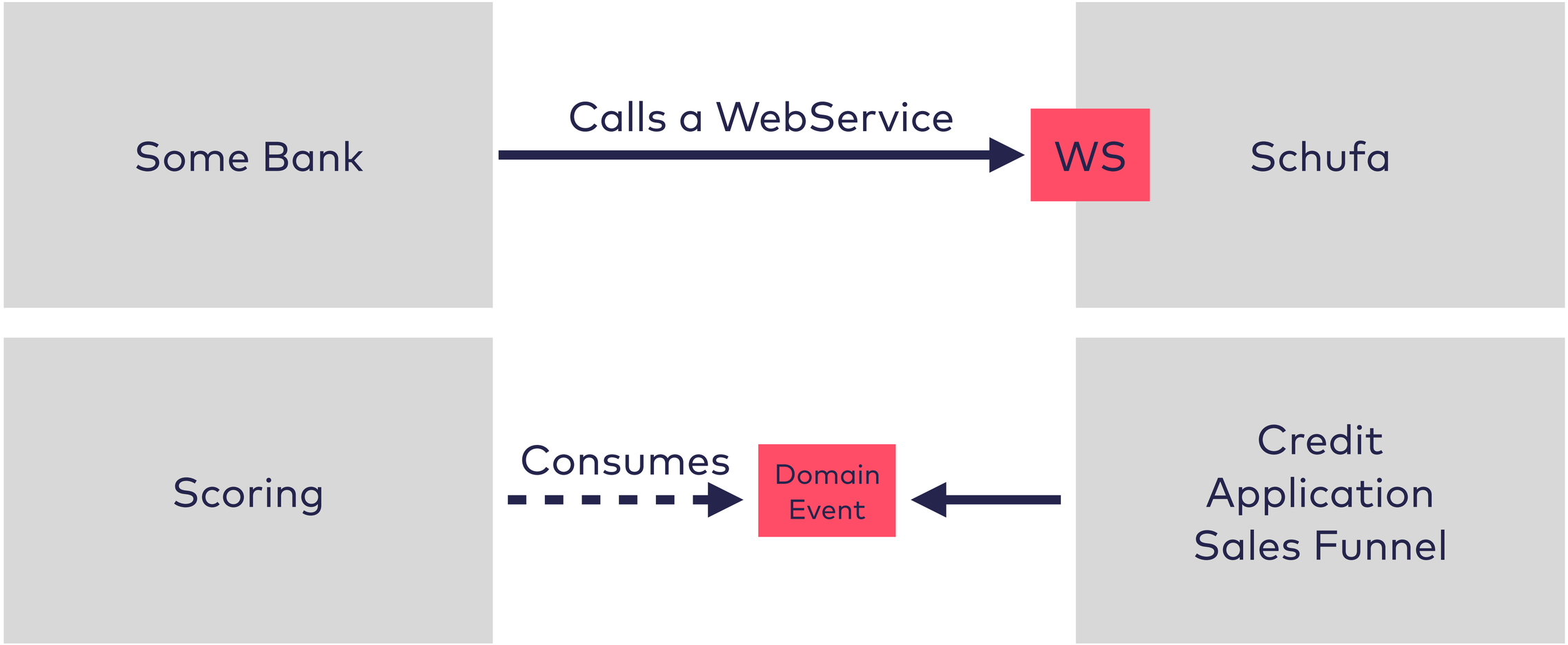
Image taken from the Team Topologies book

Context Maps aim to deliver a holistic overview with regards to coupling of bounded contexts

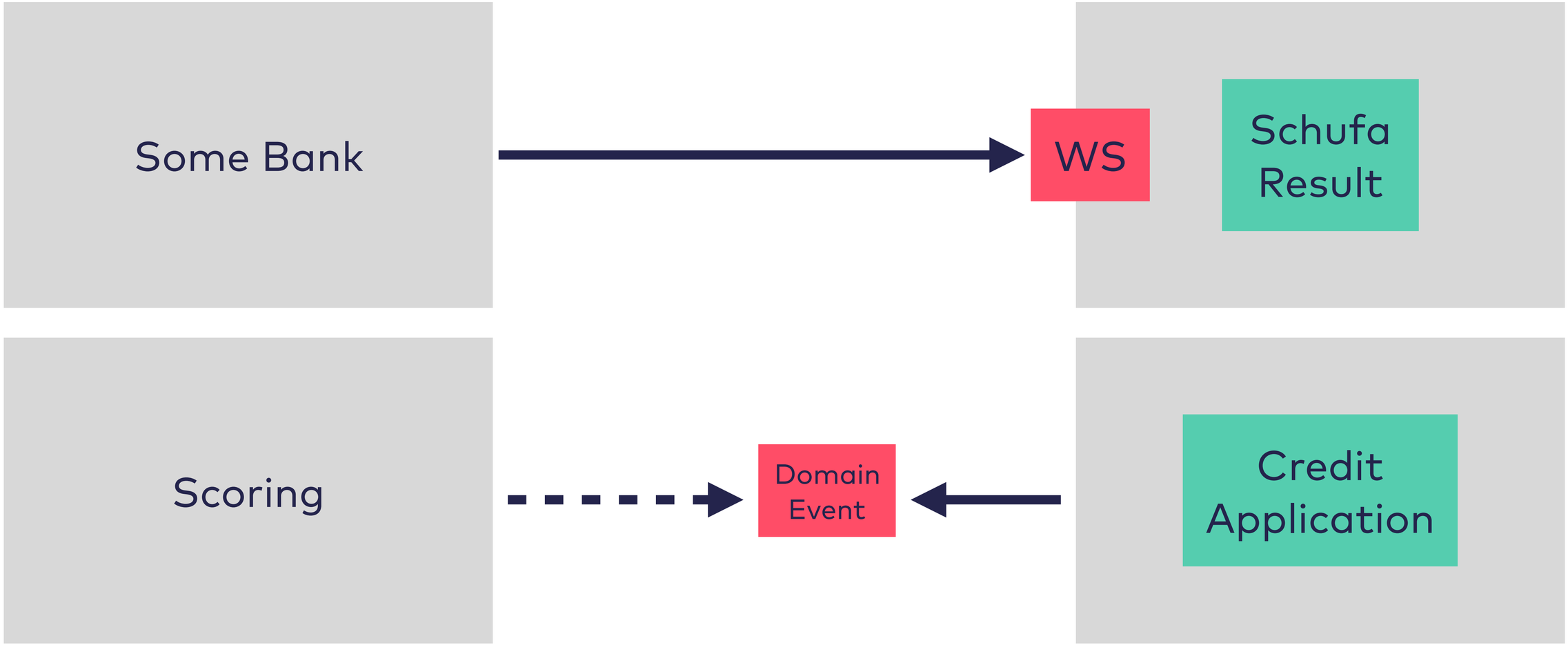
Dependencies between teams



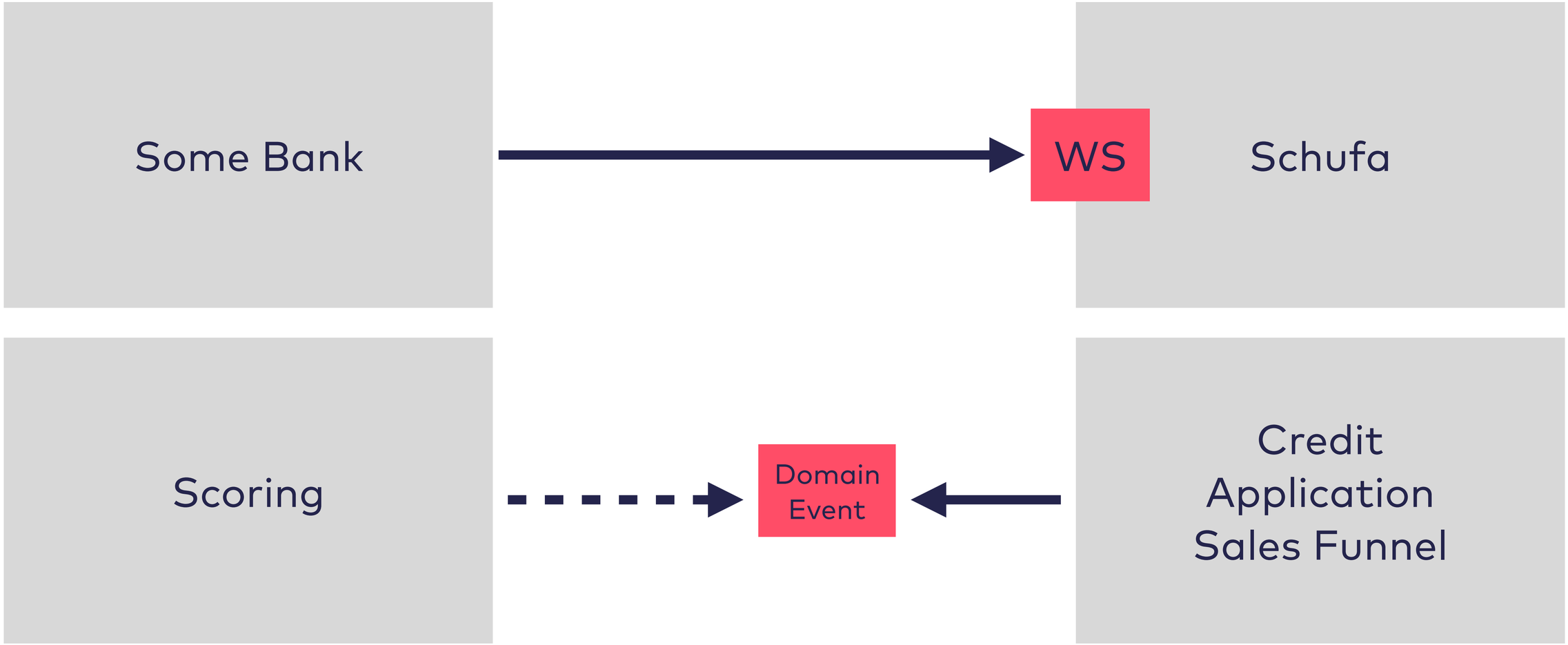
Systems calling each other or consuming events



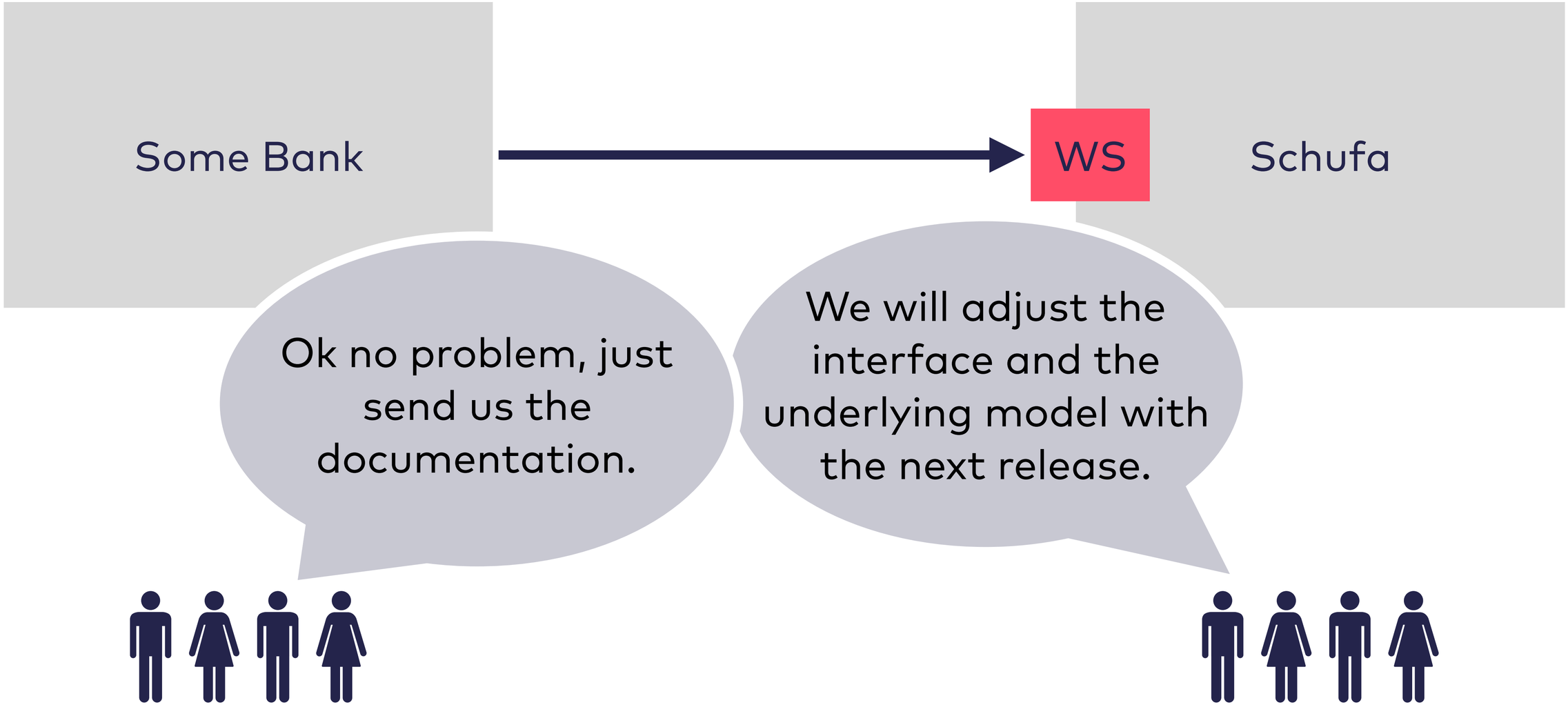
The propagation of models



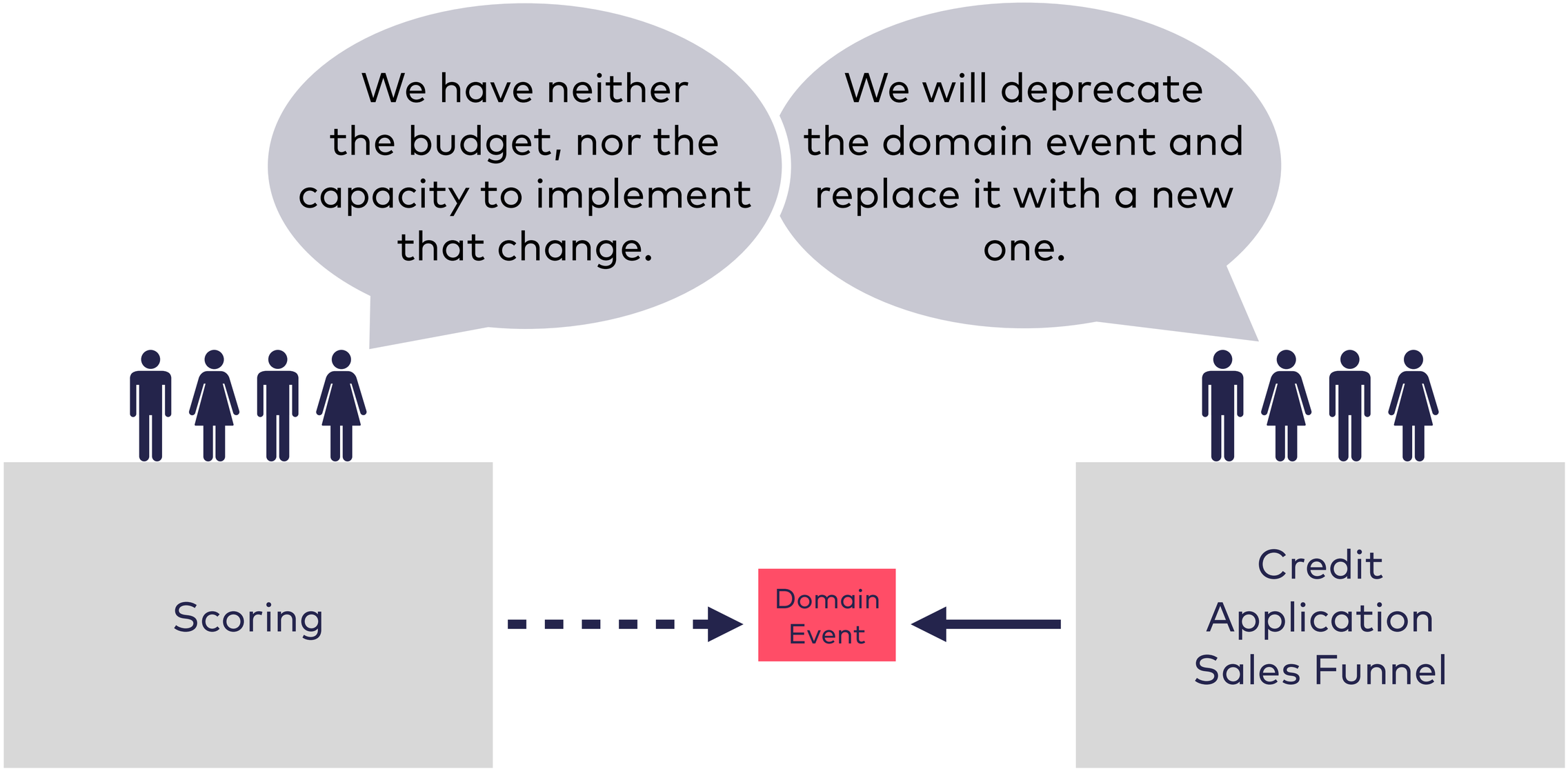
Teams communicating with each other



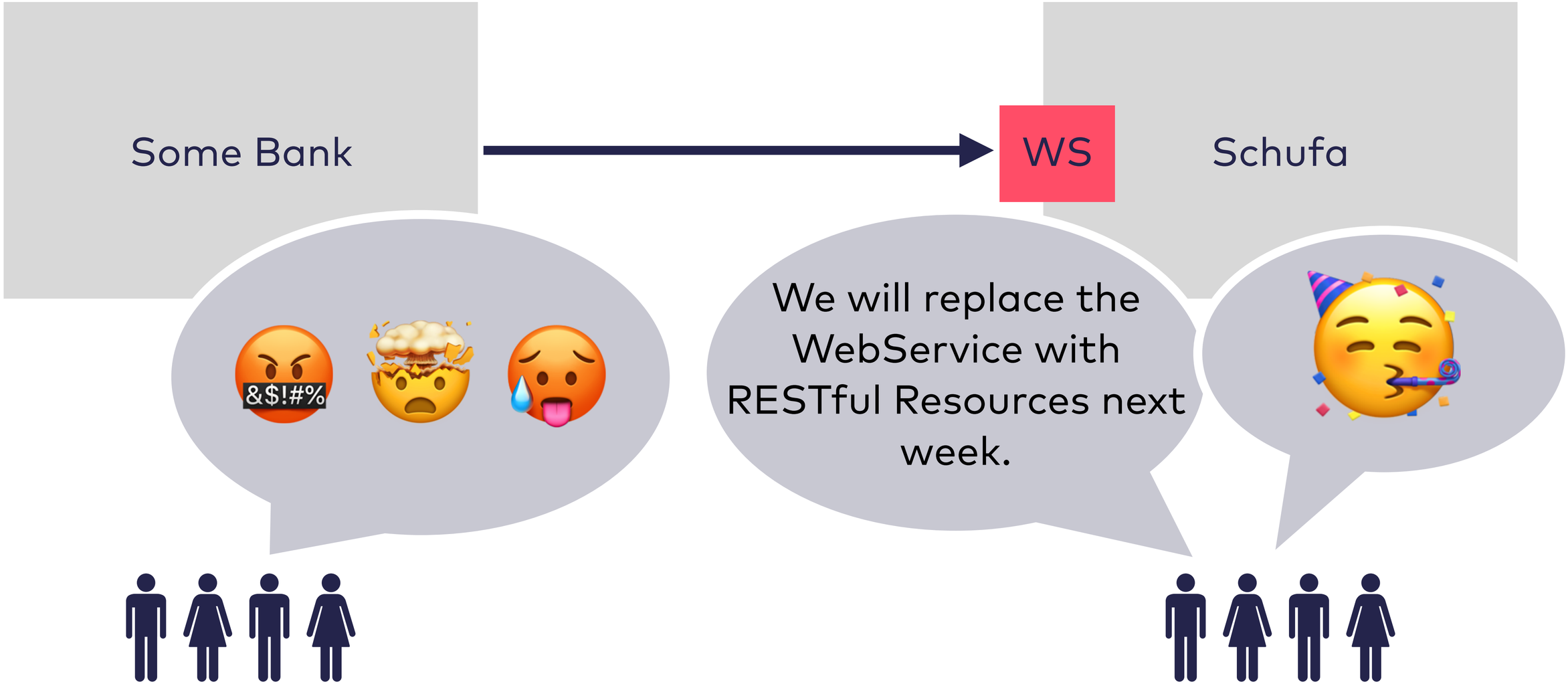
Teams communicating with each other



Teams communicating with each other



Actions of one team have an impact on others



The context map uses patterns to describe the contact between bounded contexts and teams

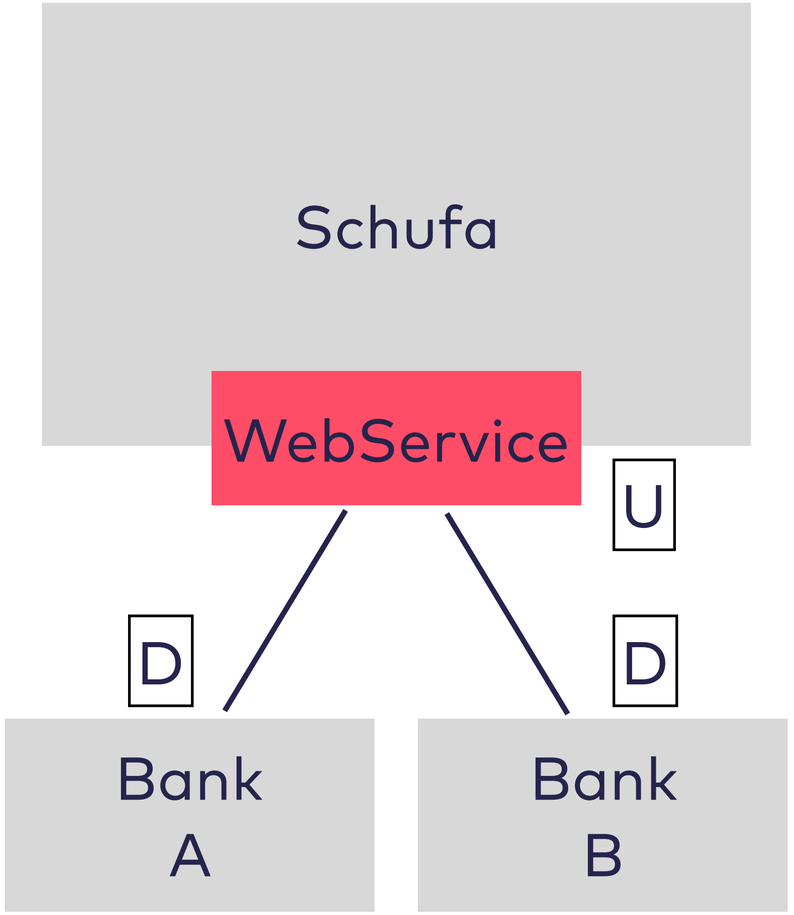
- Partnership
- Shared Kernel
- Customer / Supplier
- Conformist
- Anticorruption Layer
- Separate Ways
- Open / Host Service
- Published Language
- Big Ball Of Mud

These patterns address a diverse variety of perspectives

Open-host Service

The Open-host Service is a public API

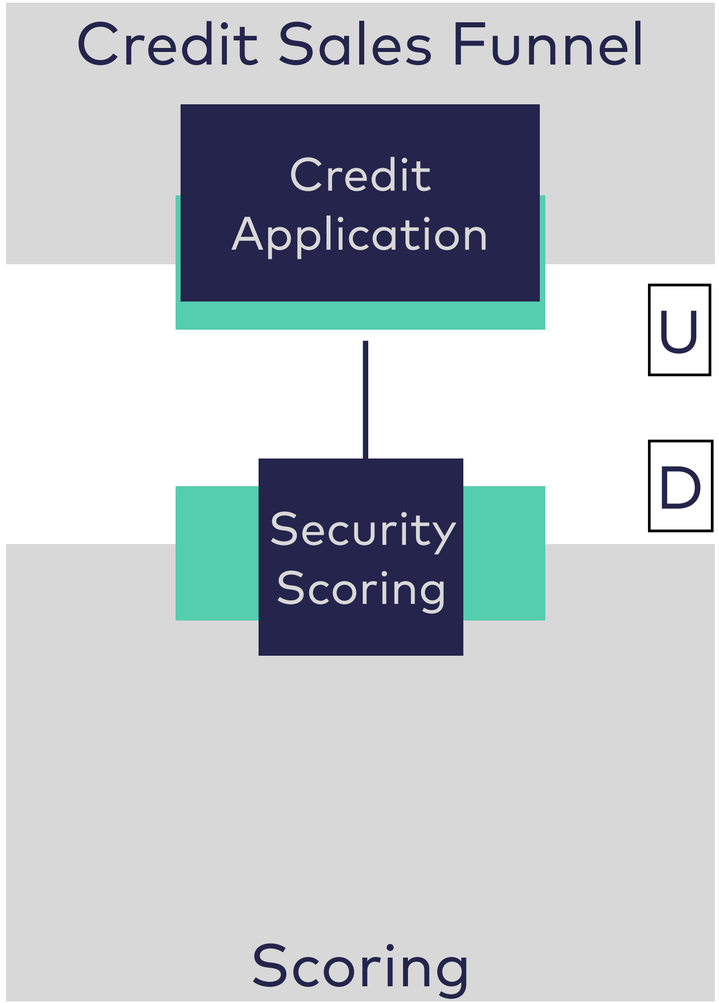
- One API for several consumers
- No point-to-point API
- Has a common, general purpose model and functionality
- The team providing the Open-host Service is an upstream team



Anticorruption Layer

The Anticorruption Layer translates one model to another one

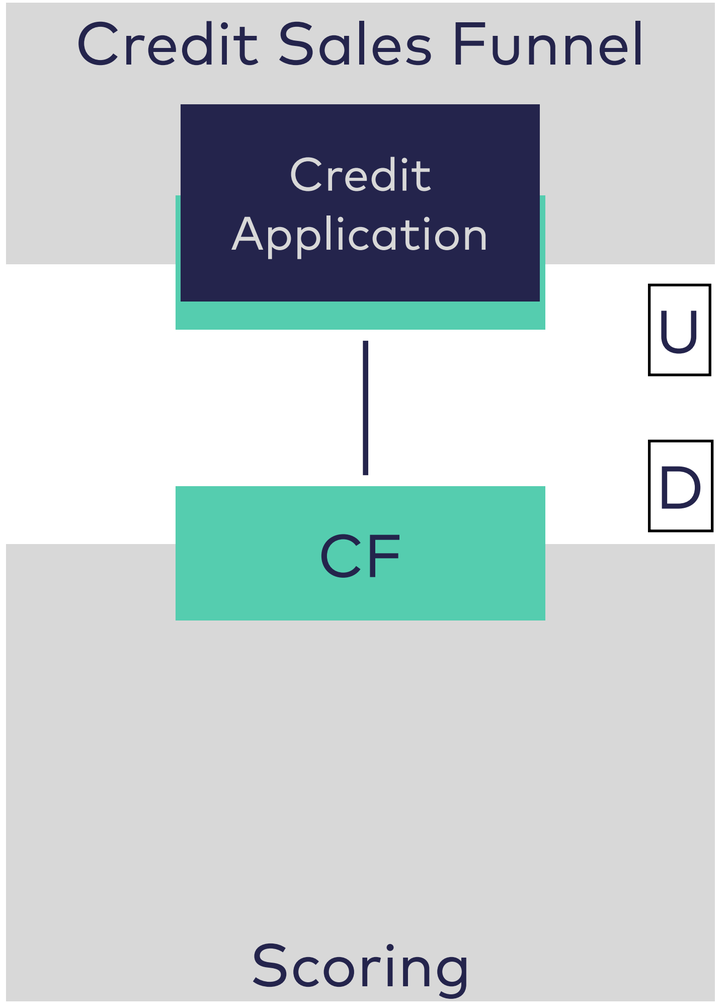
- Transforms an external model from another team / bounded context / system to another internal one
- Reduces the amount of coupling to a single layer
- The team implementing an Anticorruption Layer is always downstream

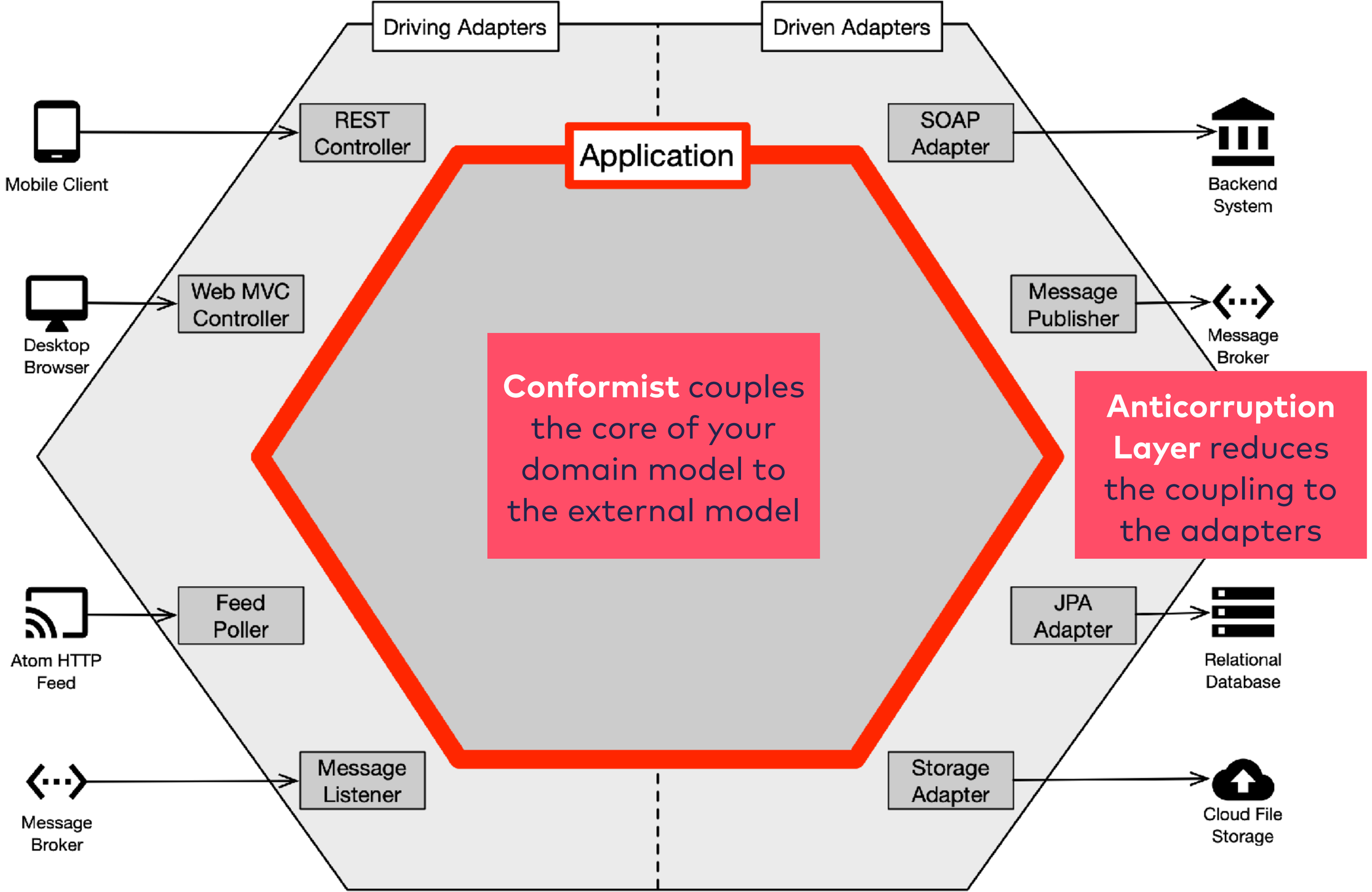


Conformist

The Conformist slavishly adheres to the upstream model

- There is no model-to-model transformation
- Motivation: Simplicity, contracts, force or delight (for the upstream model)
- The team implementing a Conformist is always downstream




Driving Adapters
Driven Adapters
Application
Mobile Client
REST Controller
SOAP Adapter
Backend System
Desktop Browser
Web MVC Controller
Message Publisher
Message Broker
Conformist couples the core of your domain model to the external model
Anticorruption Layer reduces the coupling to the adapters
Atom HTTP Feed
Feed Poller
JPA Adapter
Relational Database
Message Broker
Message Listener
Storage Adapter
Cloud File Storage

Heuristics for choosing a Conformist

The degree of coupling and also connascence of a Conformist is higher compared to an Anticorruption Layer but there are a few situations in which a Conformist may still be the better choice.

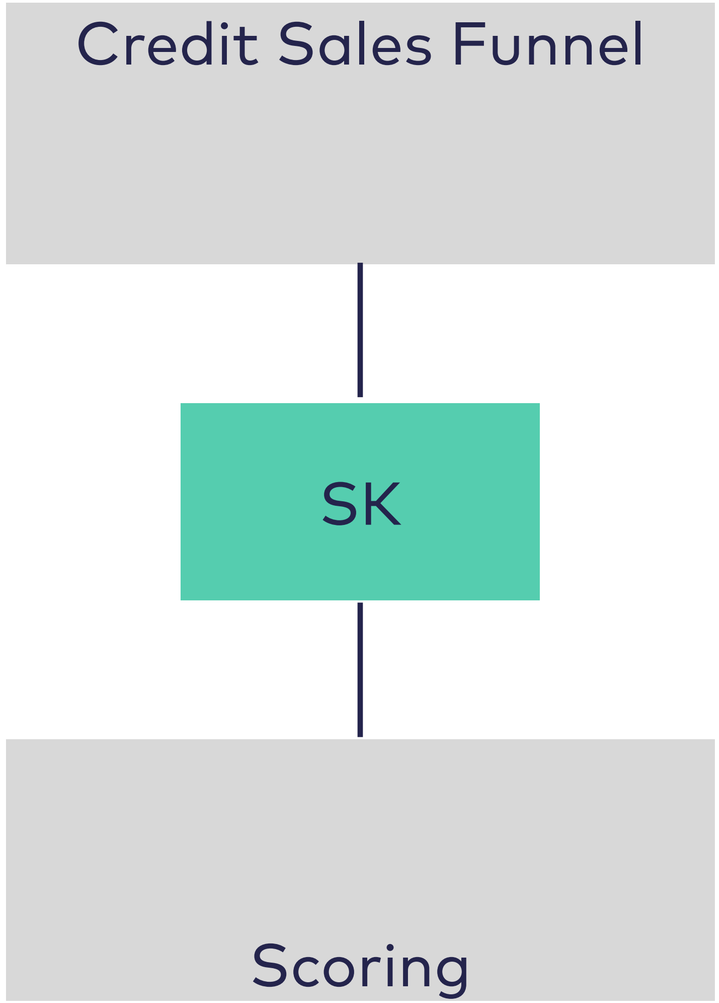
- One Bounded Context provides computations, that are highly regulated by legal authorities (collateral value of a real estate for example)

- One team / Bounded Context is considered to be a specialist in aggregations or computations and we don't want others to alter their results.

Shared Kernel

Shared Kernel is a subset of a domain model that two teams share

- „Physically“ shared artifact between two teams
- Examples: shared JARs or database
- High degree of coupling requires a high amount of coordination between the involved teams
- Shared Kernel is no Anti-Pattern but use with caution



Heuristics for Shared Kernels

A Shared Kernel introduces a high degree of coupling between the teams and their software and is, therefore, often considered not to be a good option. However, there are situations in which a Shared Kernel may be a good idea:

- One team is responsible for two or more bounded contexts which have an overlap in terms of language
- Strictly avoid a Shared Kernel when two teams are in a competitive situation or where a lot of backdoor politics are being played
- When two teams have a Shared Kernel, they should form a Partnership…

Partnership

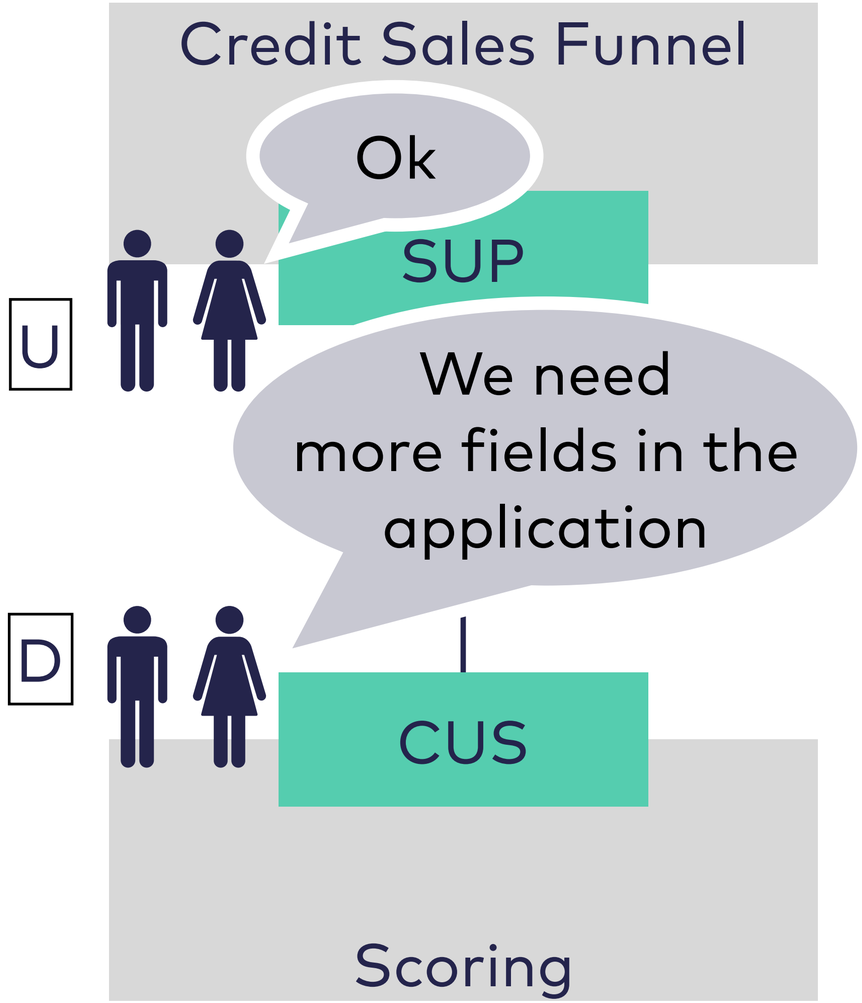
Partnership is about cooperative relationships between teams

- Establishes a process for coordinated planning of development and joint management of integration
- Not technical at all, Partnership is plain organizational
- Recommended for teams which depend on a Shared Kernel

Customer-Supplier

A Customer-Supplier development gives the downstream team some influence

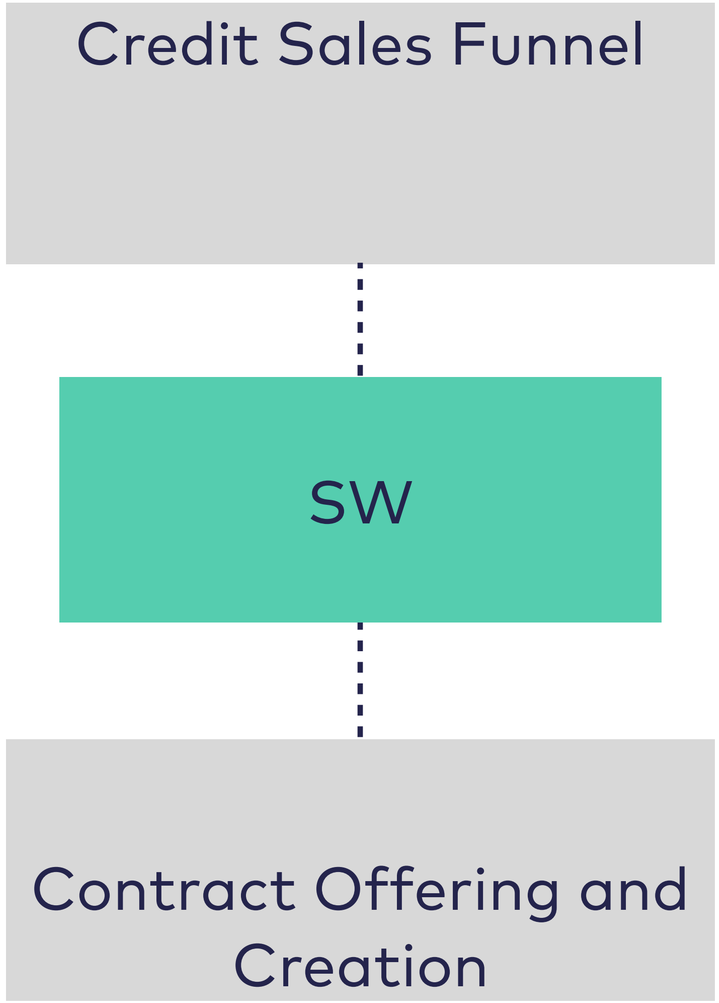
- Downstream requirements factor into upstream planning. Therefore, the downstream team gains some influence over the priorities and tasks of the upstream team
- Customer-Supplier is organizational
- Mind „vetoing customer“ and customer against an OHS as anti-patterns



Separate Ways

A bounded context has no connections to others

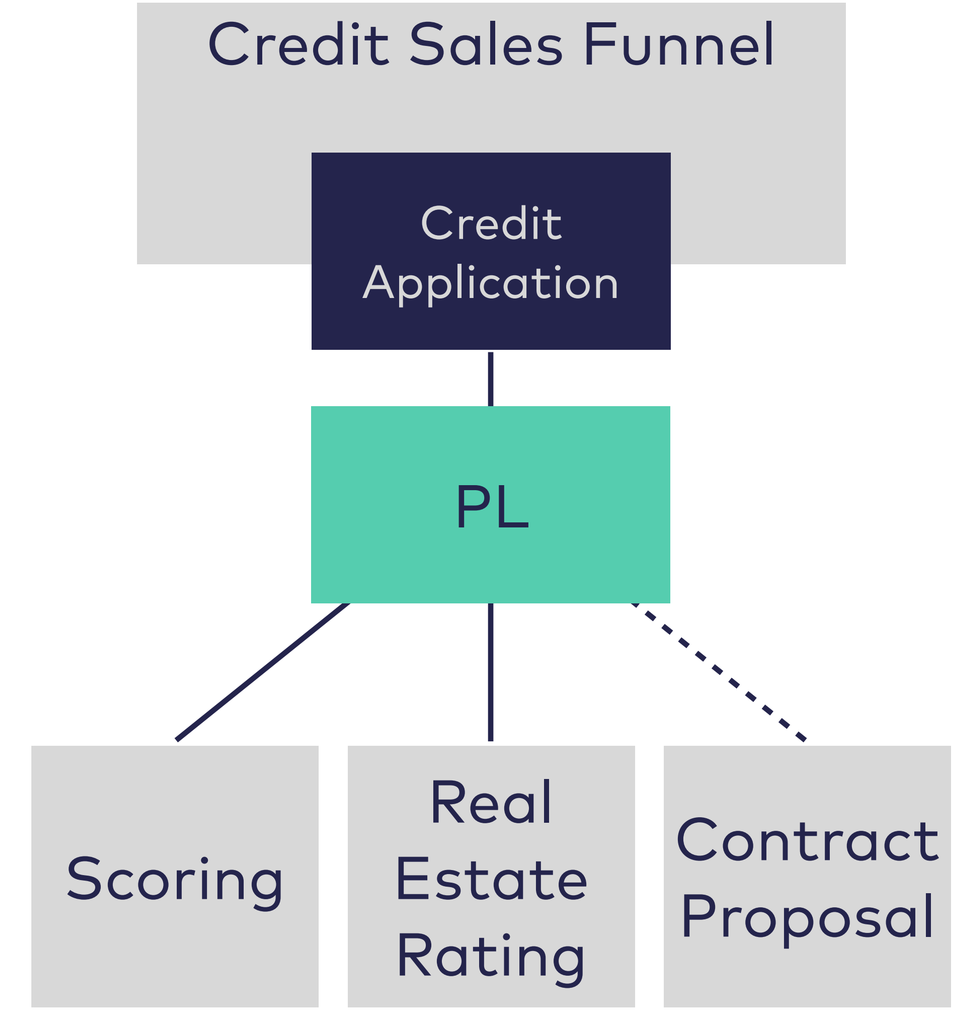
- Sometimes integration is too expensive or takes very long
- The teams choose separate ways in order to focus on specialized solutions
- Interesting pattern for minimum viable products or organizational solutions in uncertain market conditions



Published Language

A well documented language shared between bounded contexts

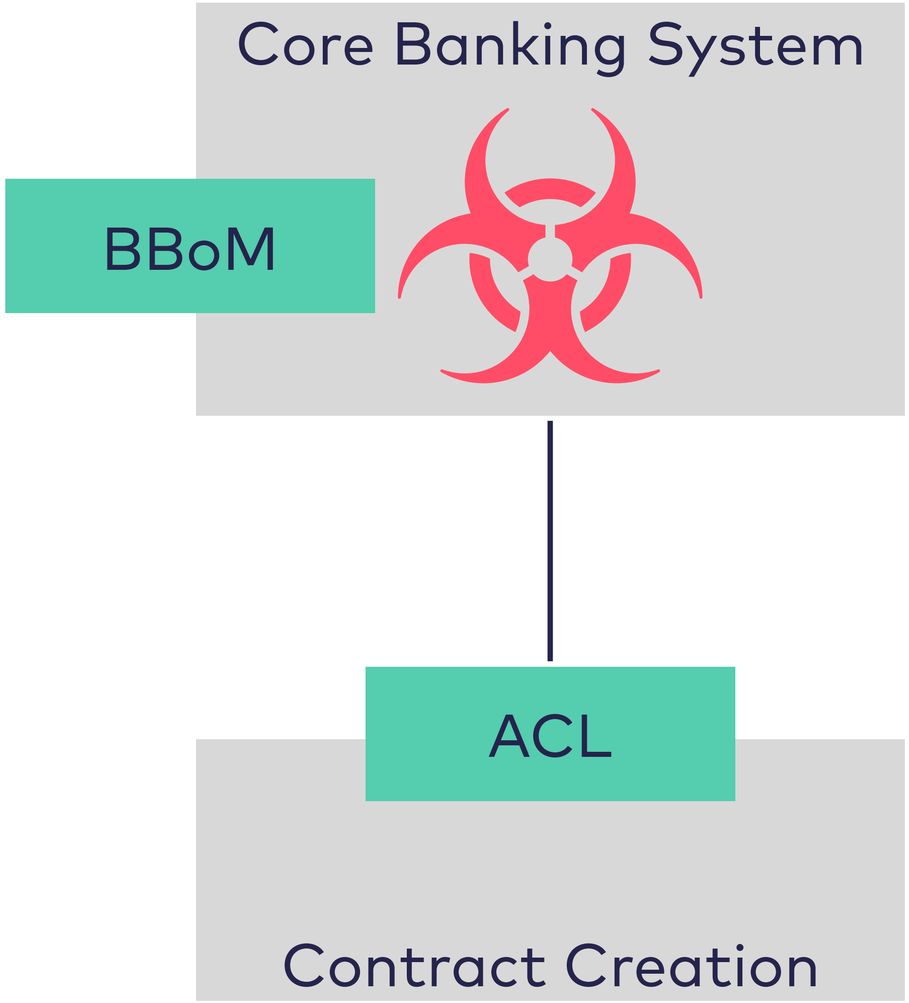
- Every bounded context can translate in and out from that language
- Sometimes defined by a consortium of the most important stakeholders / teams
- Often combined with Open-host Service
- Examples: iCalendar, vCard, ZugFerd



Big Ball Of Mud

A (part of a) system which is a mess by having mixed models and inconsistent boundaries

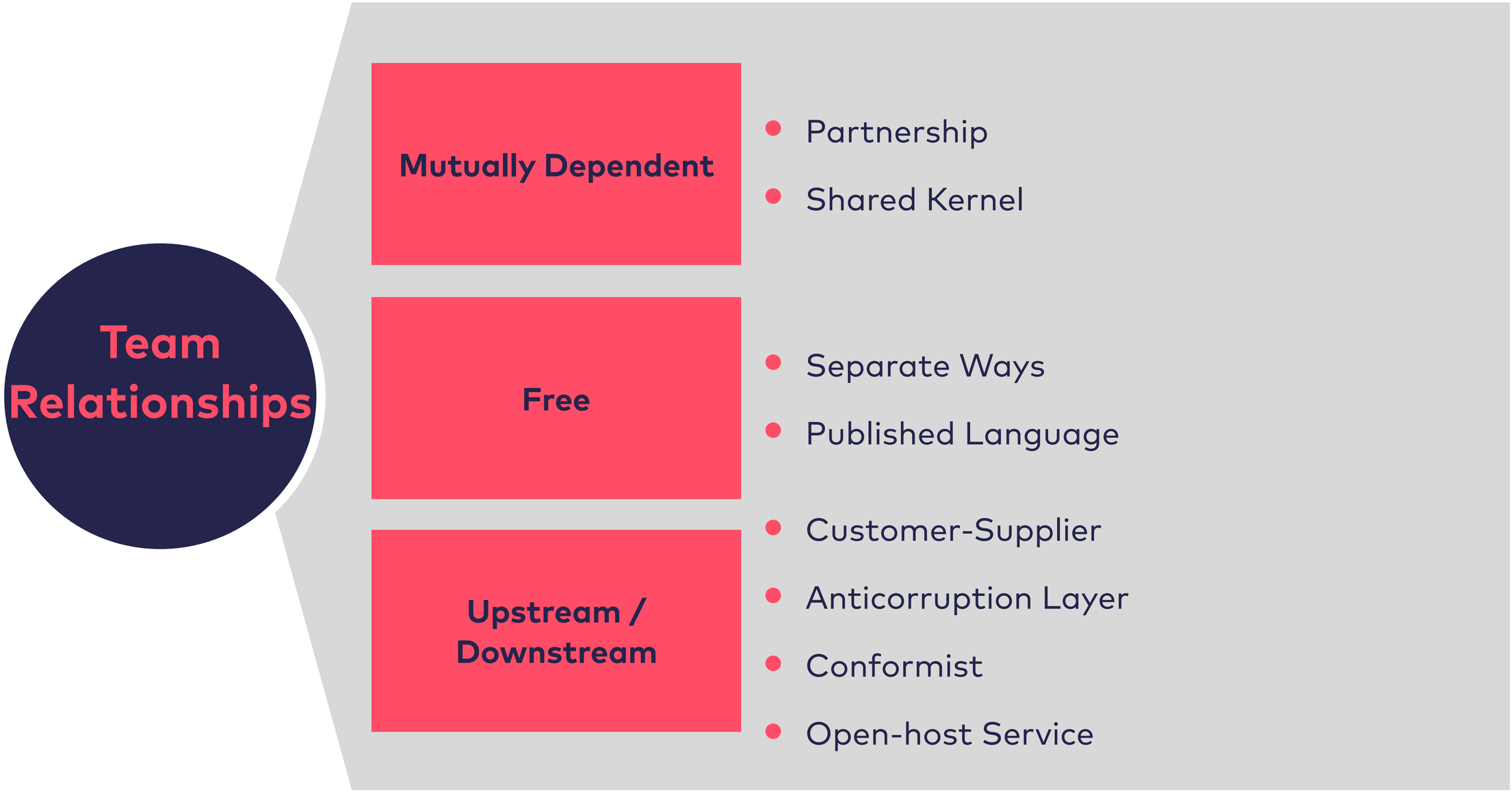
- Don't let the (lousy) model of the Big Ball Of Mud propagate into your context
- Anticorruption Layer is the pattern of choice on the downstream
- Demarcation of bad model or system quality



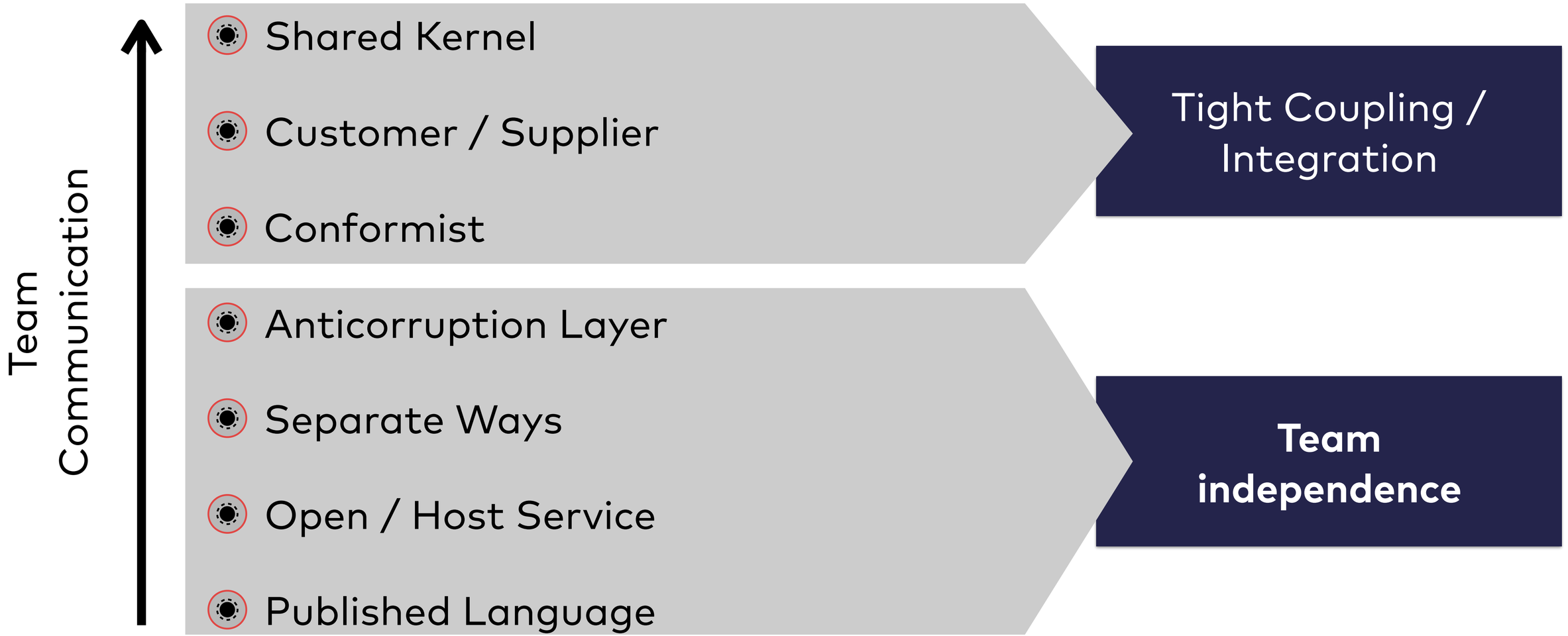
The patterns address various aspects

	Team Relationships	Model Propagation	API / „technical“
Open-host Service	(✅)		✅
Anticorruption Layer		✅	
Conformist		✅	
Shared Kernel		✅	(✅)
Partnership	✅		
Customer-Supplier	✅		
Separate Ways	✅	(✅)	
Published Language	✅	(✅)	
Big Ball Of Mud			✅

Some of the patterns reflect team relationships

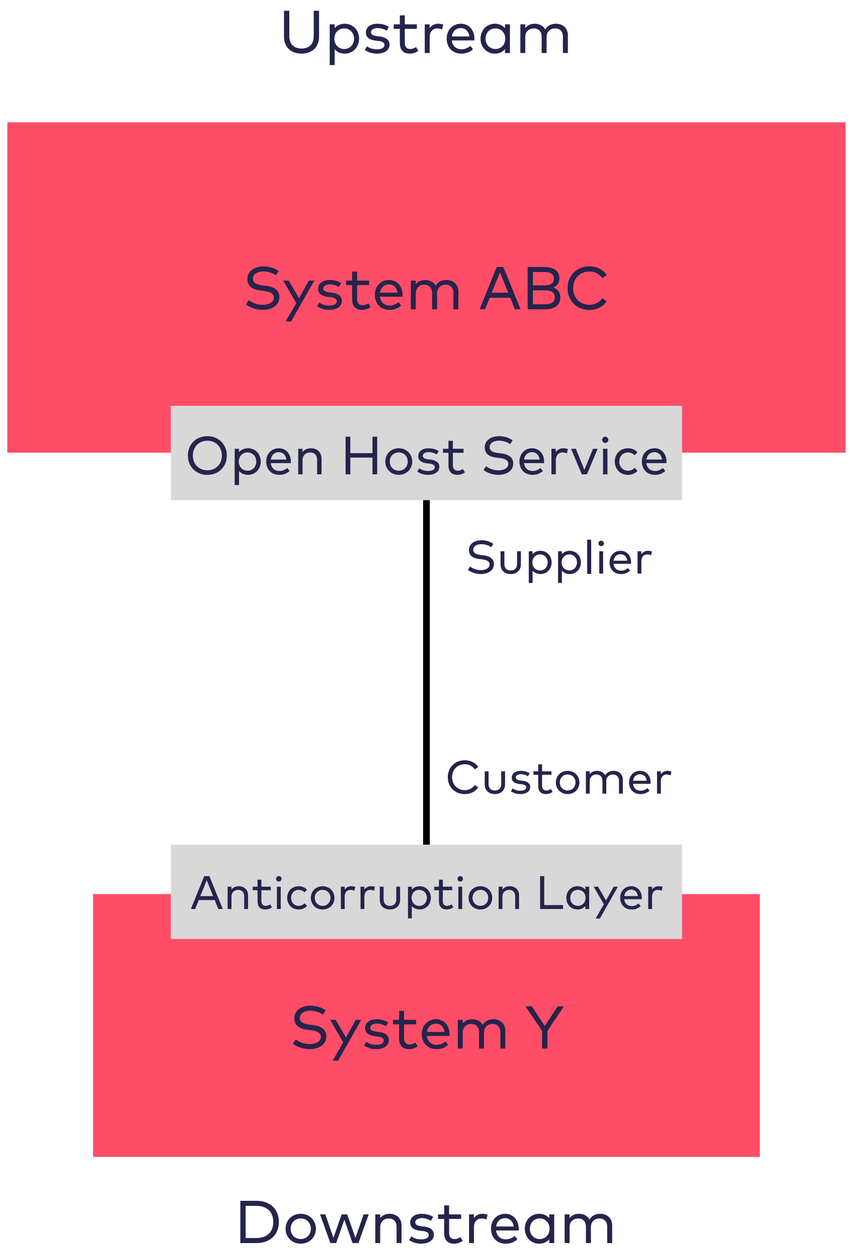


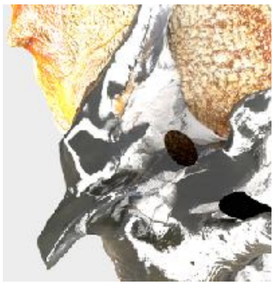
Mind team communication



You can visualize different perspectives

- Call Relationship
- Team Relationship - Level 1
- API Level
- Model Propagation
- Team Relationship - Level 2





Subdomain categories and the context map

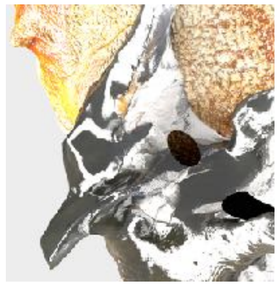
Ask those questions

Should a generic or supporting subdomain be a customer for a supplier in a core domain?

Should a core domain conform to the model of a generic or supporting subdomain?

How do we deal with a Big Ball Of Mud in a core domain? Conform to it in the other categories?

Do we want Partnerships or mutually dependent teams between core and the other subdomains?



Also look at bounded context model traits

Name

Model Traits
draft, execute, audit, enforcer, interchange, gateway, etc.

Description
Summary of purpose and responsibilities

Information and Services Provided

Queryable Information

Invokable Commands

Published Events

Reactive Jobs

Strategic Classification

Domain	Business Model	Evolution
- Core - Supporting - Generic - Other	- Revenue - Engagement - Compliance - Cost reduction	- Genesis - Custom built - Product - Commodity

Business Decisions
Key business rules, policies, and decisions

Dependencies and Relationships

Suppliers		Consumers	
Name	Relationship	Name	Relationship

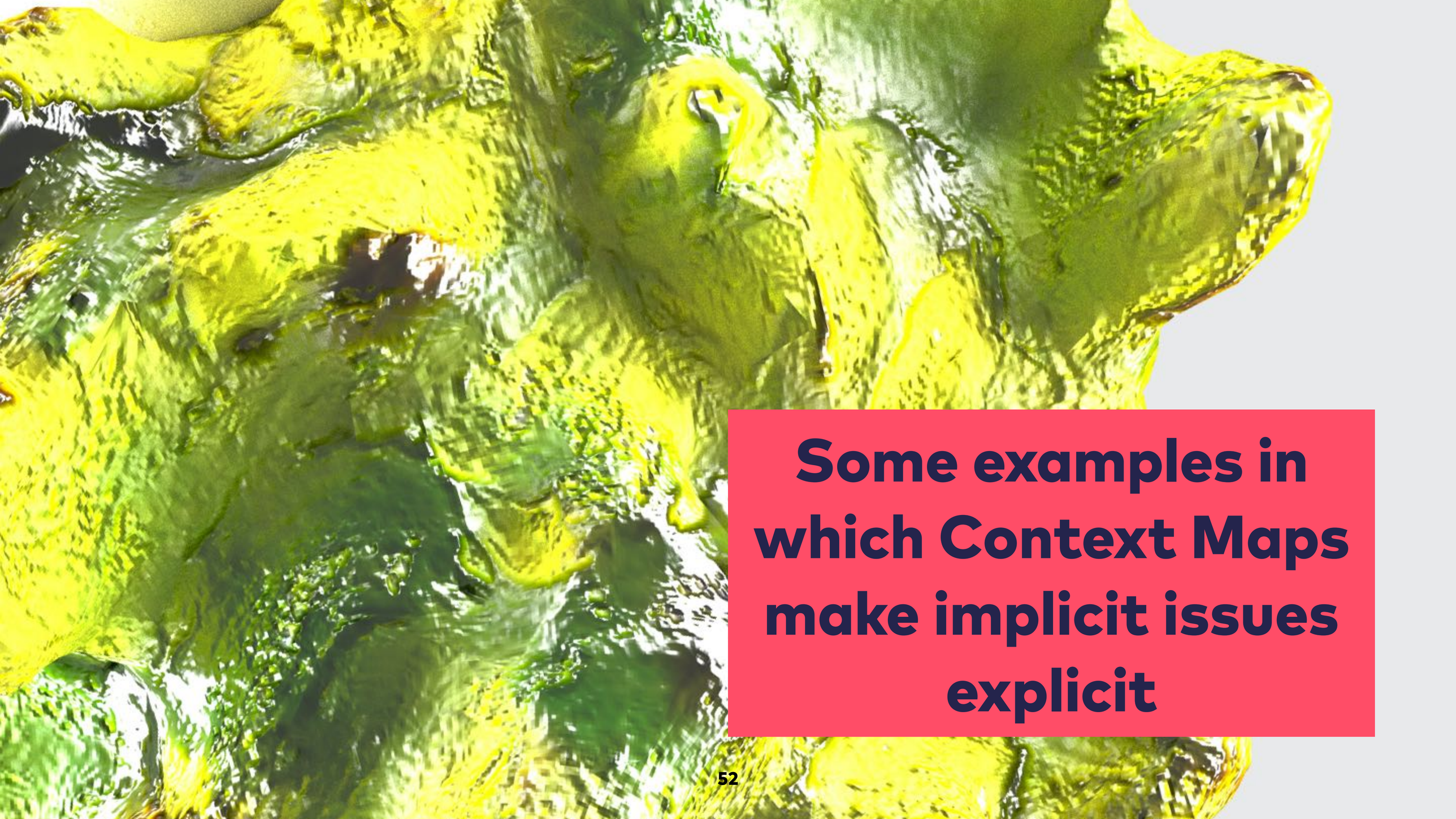
Ubiquitous Language
Key domain terminology

Bounded Context Canvas V2 Nick Tune | ntcoding.co.uk

Influence of an Octopus Enforcer?

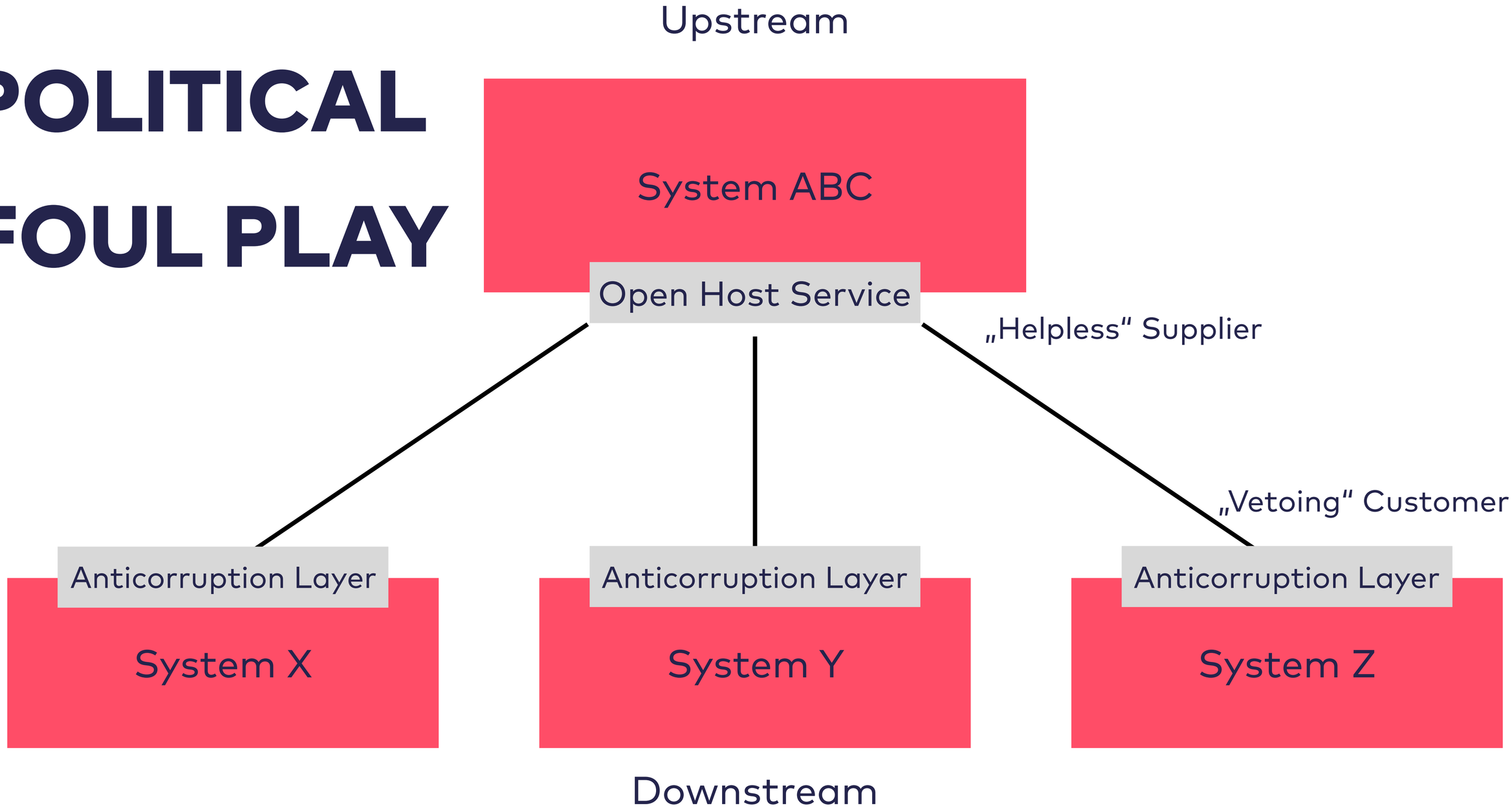
Bubble Context for Bubble Context?

Mind the Enterprise Integration Patterns!

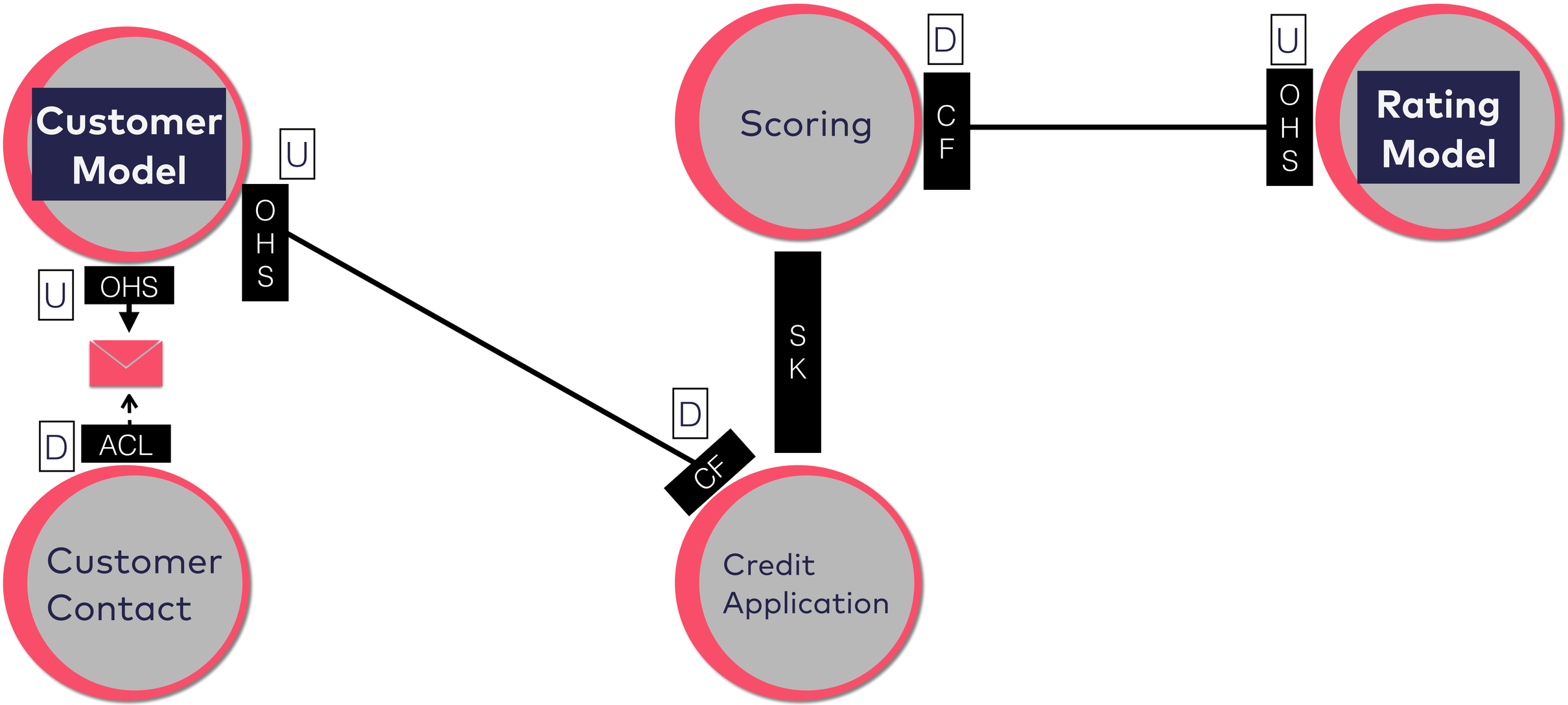


Some examples in which Context Maps make implicit issues explicit

POLITICAL FOUL PLAY



The model propagation from hell

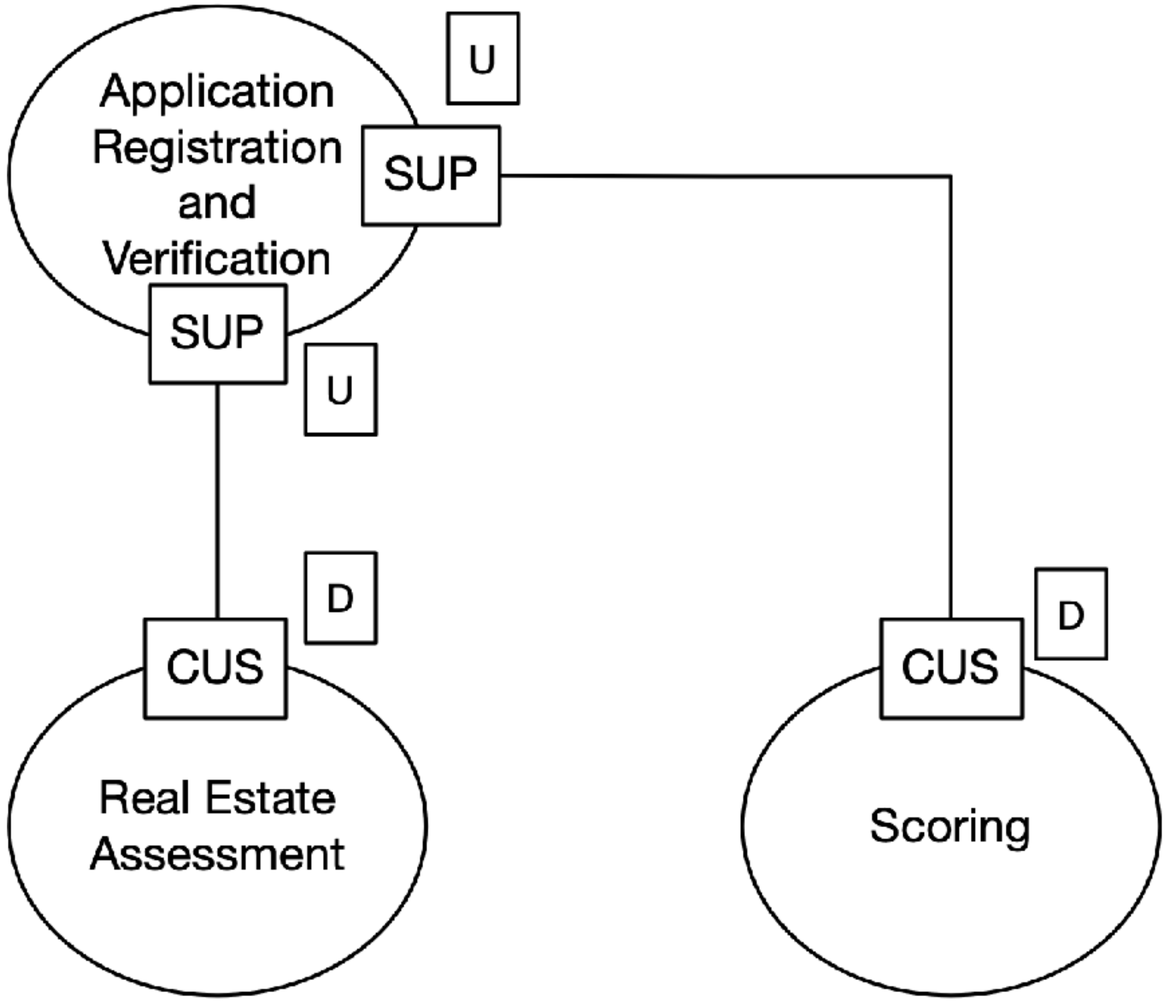


Context Map Scenarios

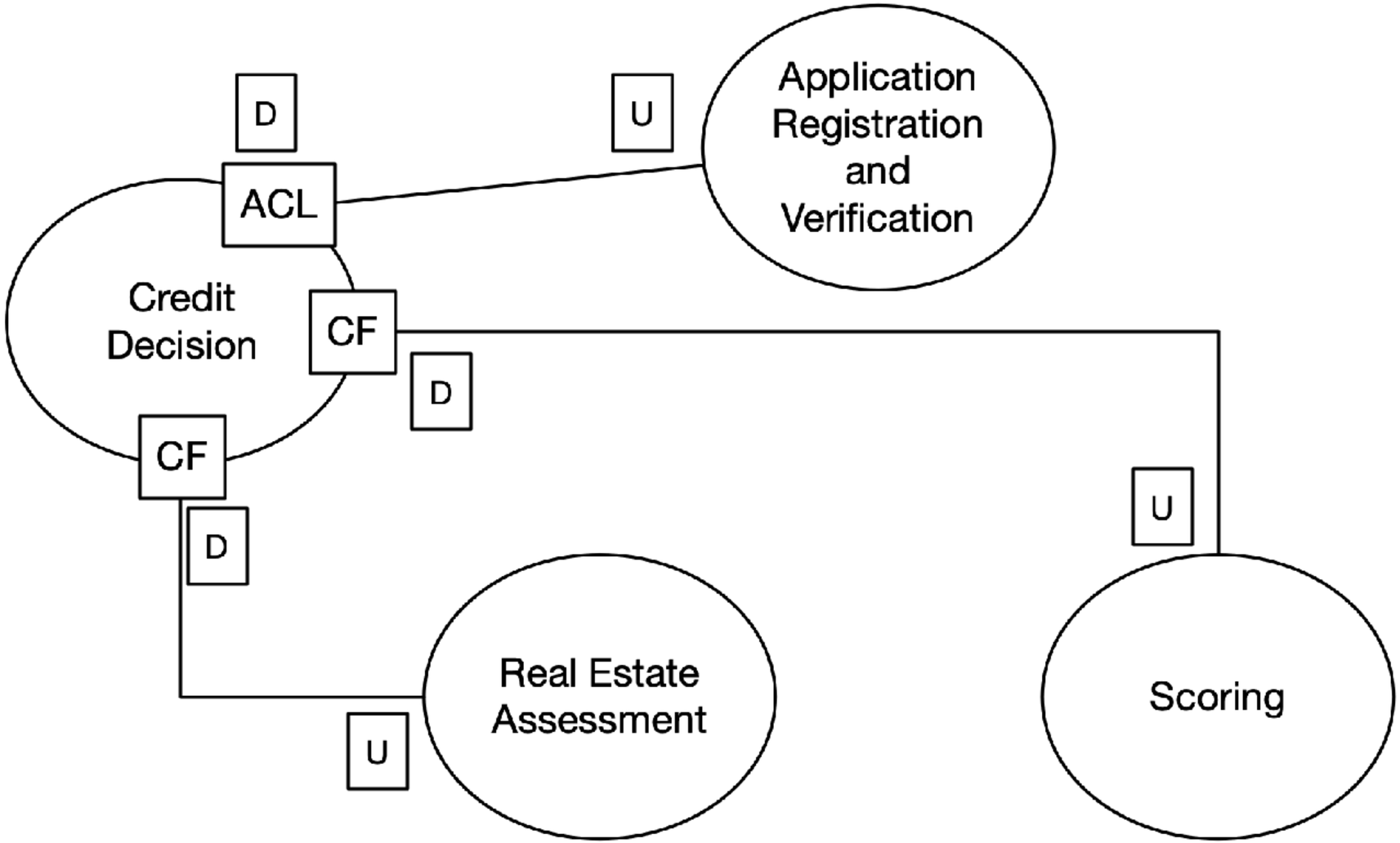
Taken from my Domain-driven Design book

- These scenarios showcase various options / alternatives
- Domain: retail mortgage loans
- https://www.leanpub.com/ddd-by-example

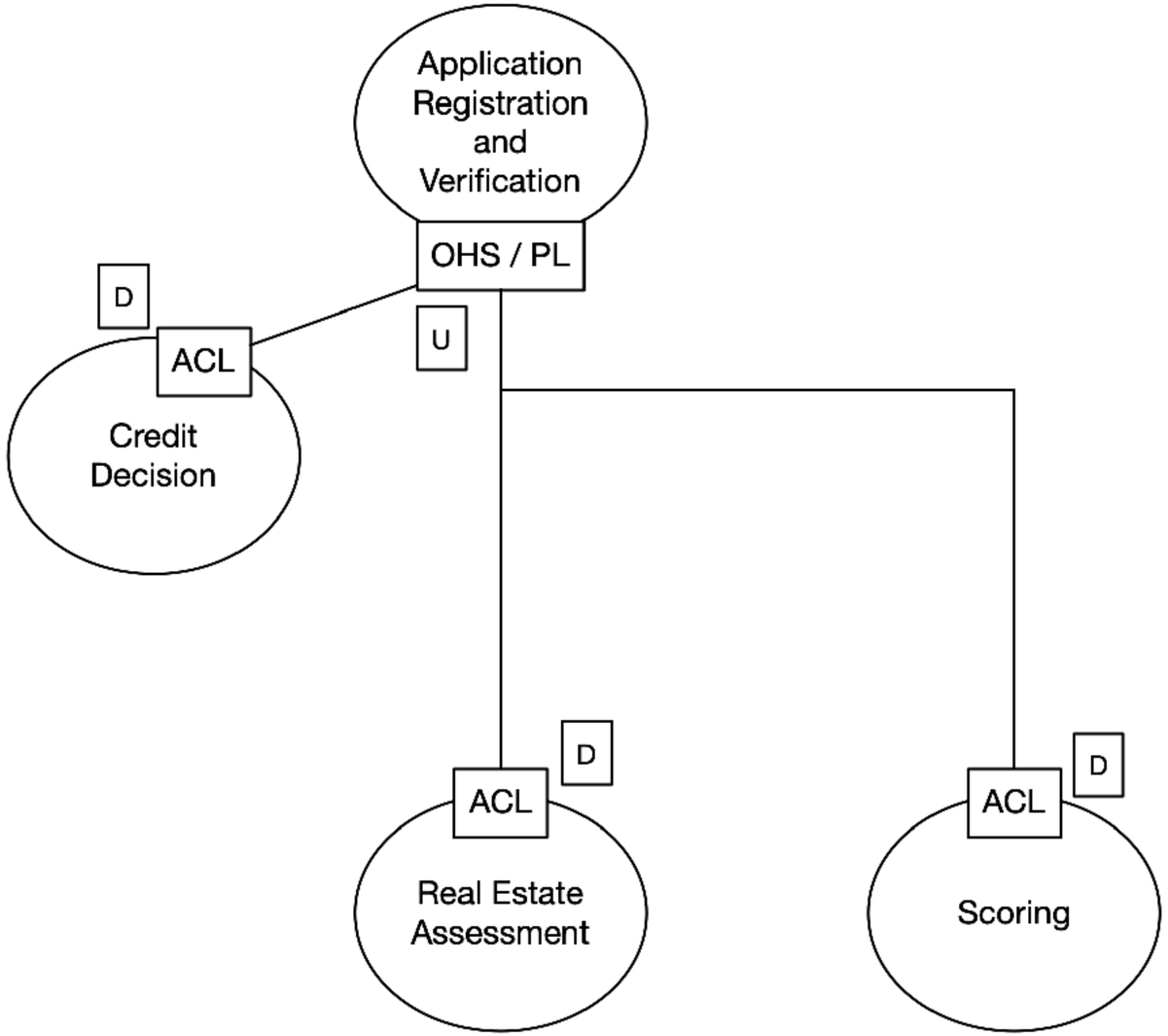
How teams gain influence



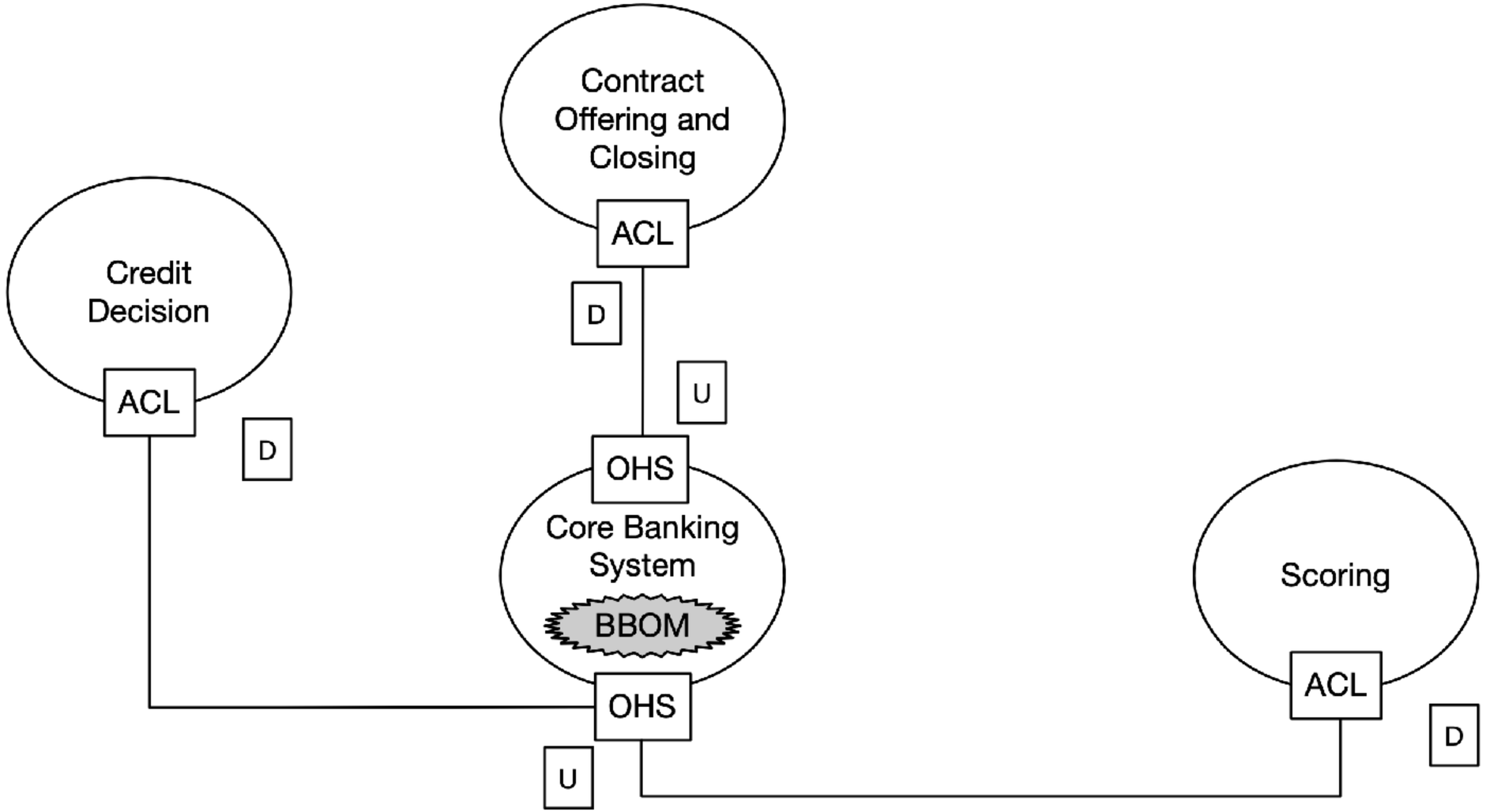
Degrading influence of a team



Using a Published Language



Dealing with a Big Ball Of Mud



Where should 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 relations

Context Map

The context map is a powerful tool for

- making implicitly hidden organizational processes visible
- identify the flow of bad domain models
- establishing a decentralized governance model
- PS: managers LOVE context maps ;-)

Get my DDD book cheaper

Book Voucher: 7.99 instead of (min) 9.99
http://leanpub.com/ddd-by-example/c/speakerdeck

Thank you!

Michael Plöd

Follow me on Twitter: @bitboss