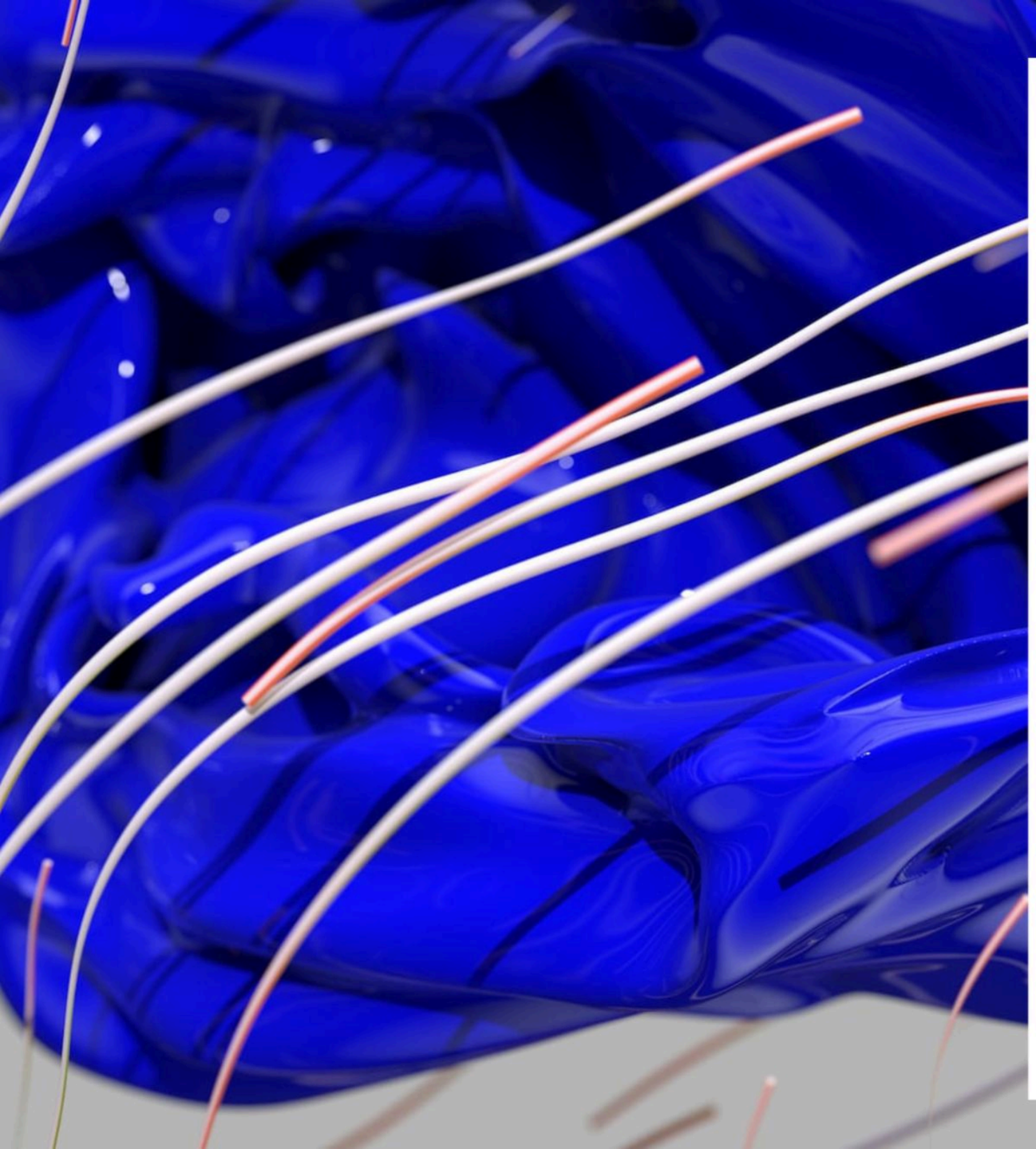


Domain Driven Design Europe 2022 - Amsterdam

Introduction to Context Mapping

INNOQ

MICHAEL PLÖD
FELLOW

168 194 136 59 43

Get my DDD book cheaper

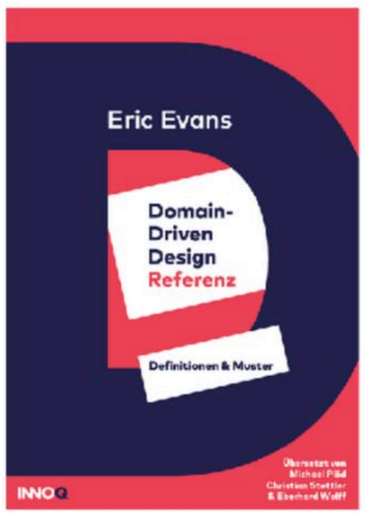
Book Voucher: 7.99 instead of (min) 9.99
http://leanpub.com/ddd-by-example/c/speakerdeck

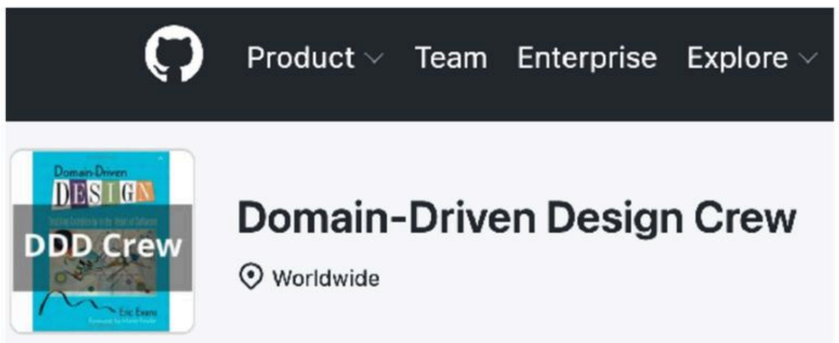
Michael Plöd

Fellow at INNOQ

Current consulting and training activities:

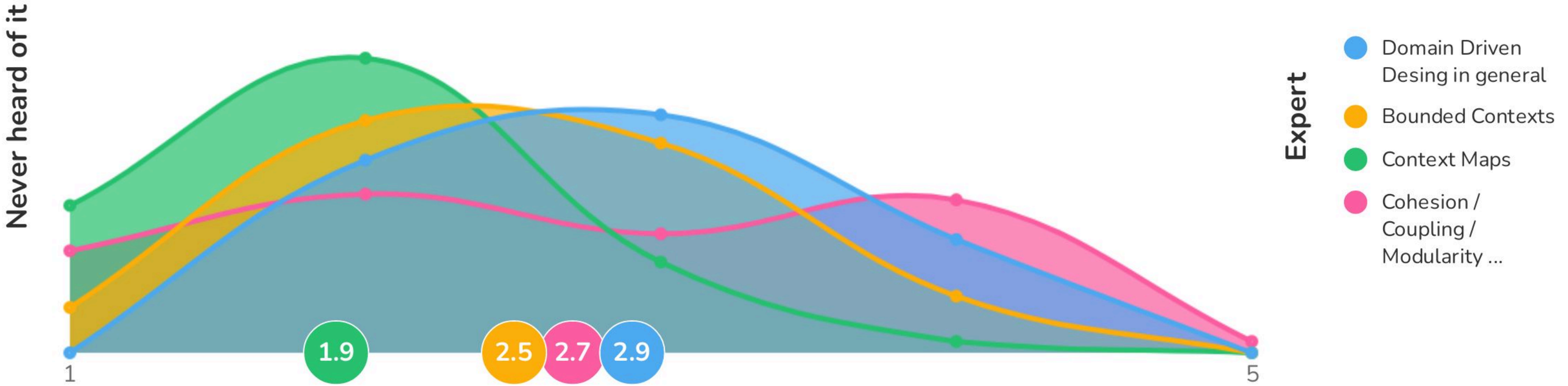
- Domain-Driven Design
- Team Topologies
- Software Architecture (and often environment) transformation





168 194 136 59 43

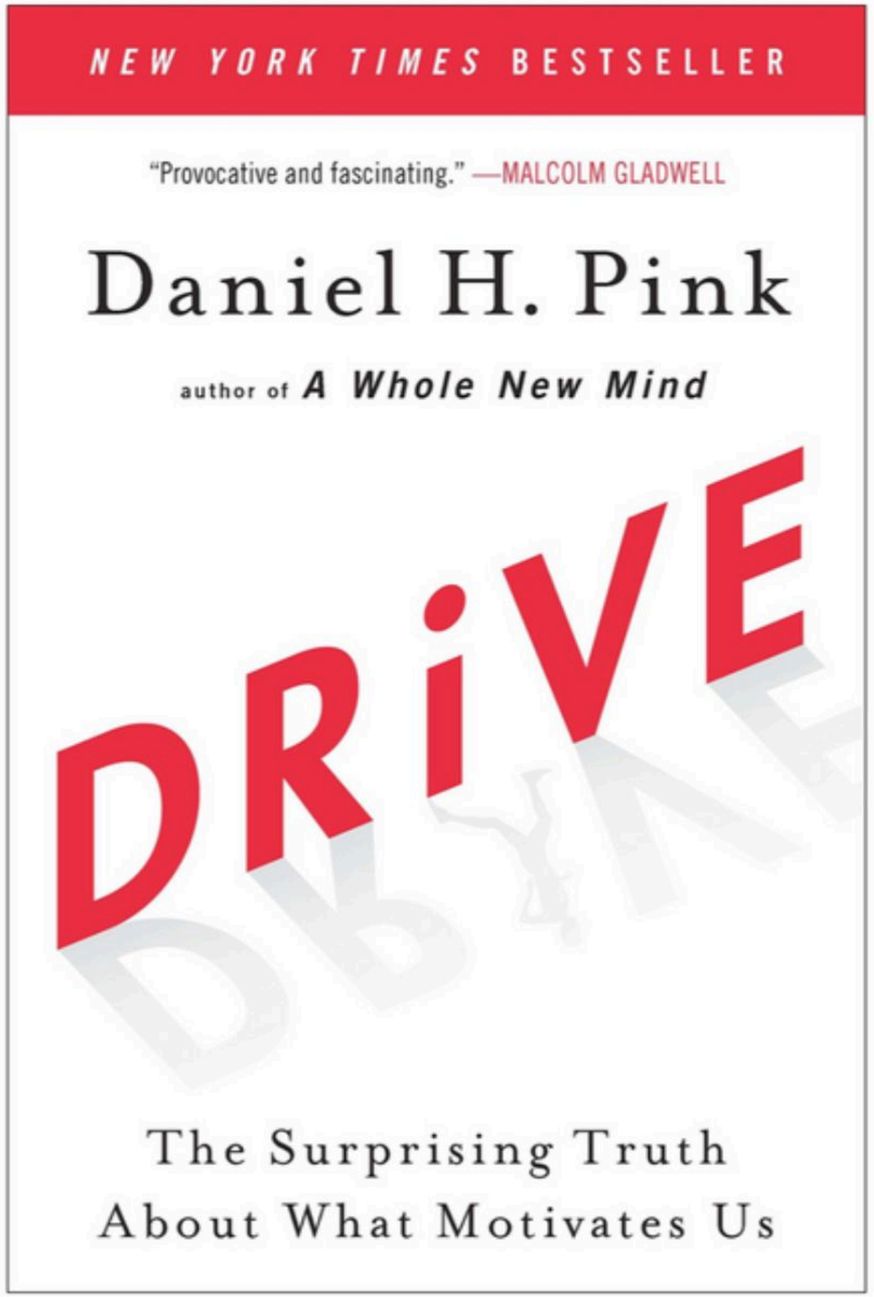
Rate your knowledge



168 194 136 59 43 96

„Good fences make good neighbors“

Robert Frost



Autonomy

Mastery

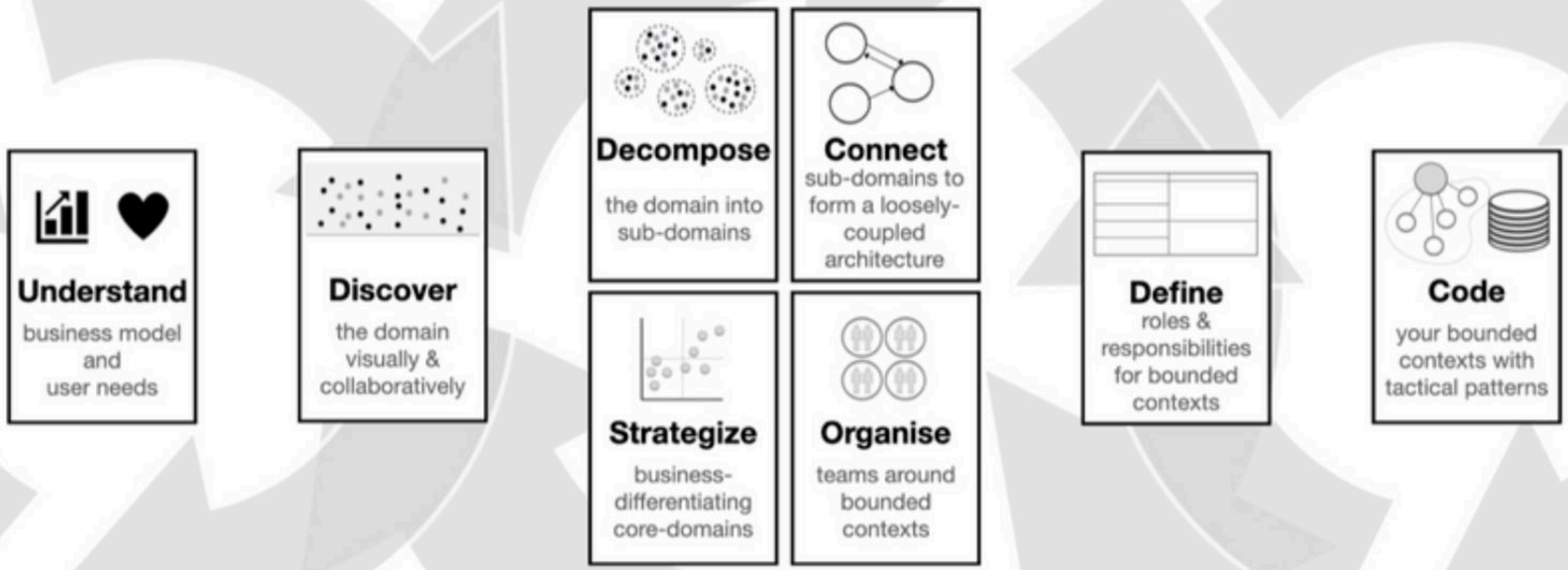
Purpose

We need good boundaries in which teams can achieve

Autonomy - Mastery - Purpose

Domain-Driven Design Starter Modelling Process

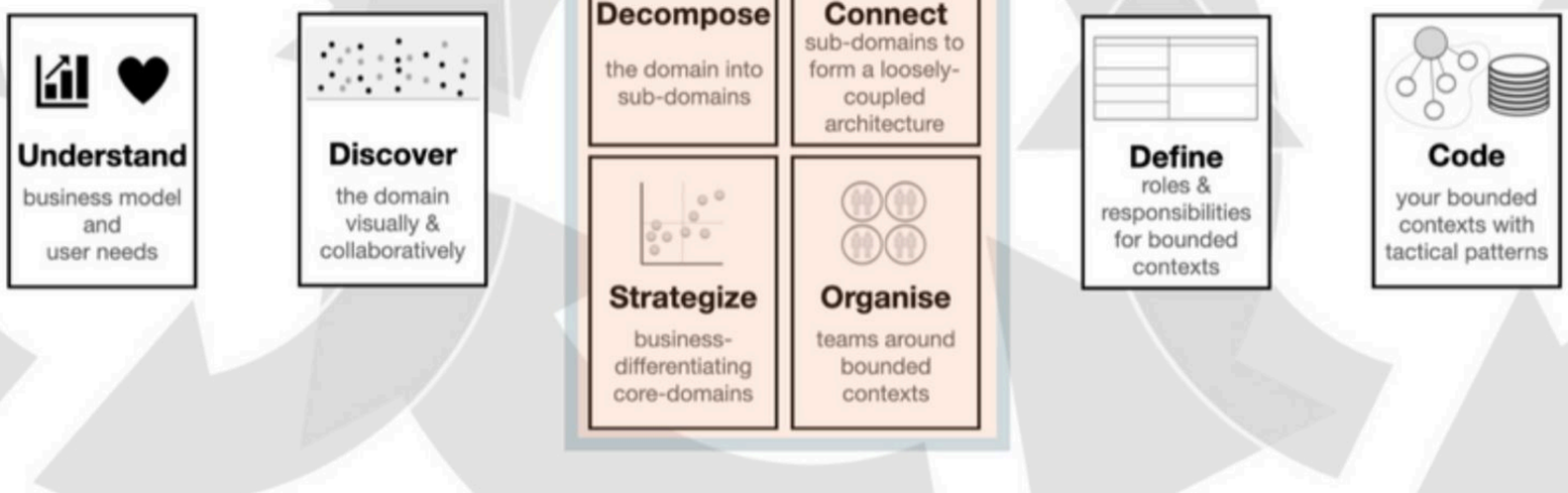
A starter process for beginners, not a rigid best-practice. DDD is continuous, evolutionary, and iterative design.



https://github.com/ddd-crew/ddd-starter-modelling-proces

Domain-Driven Design Starter Modelling Process

A starter process for beginners, not a rigid best-practice. DDD is continuous, evolutionary, and iterative design.



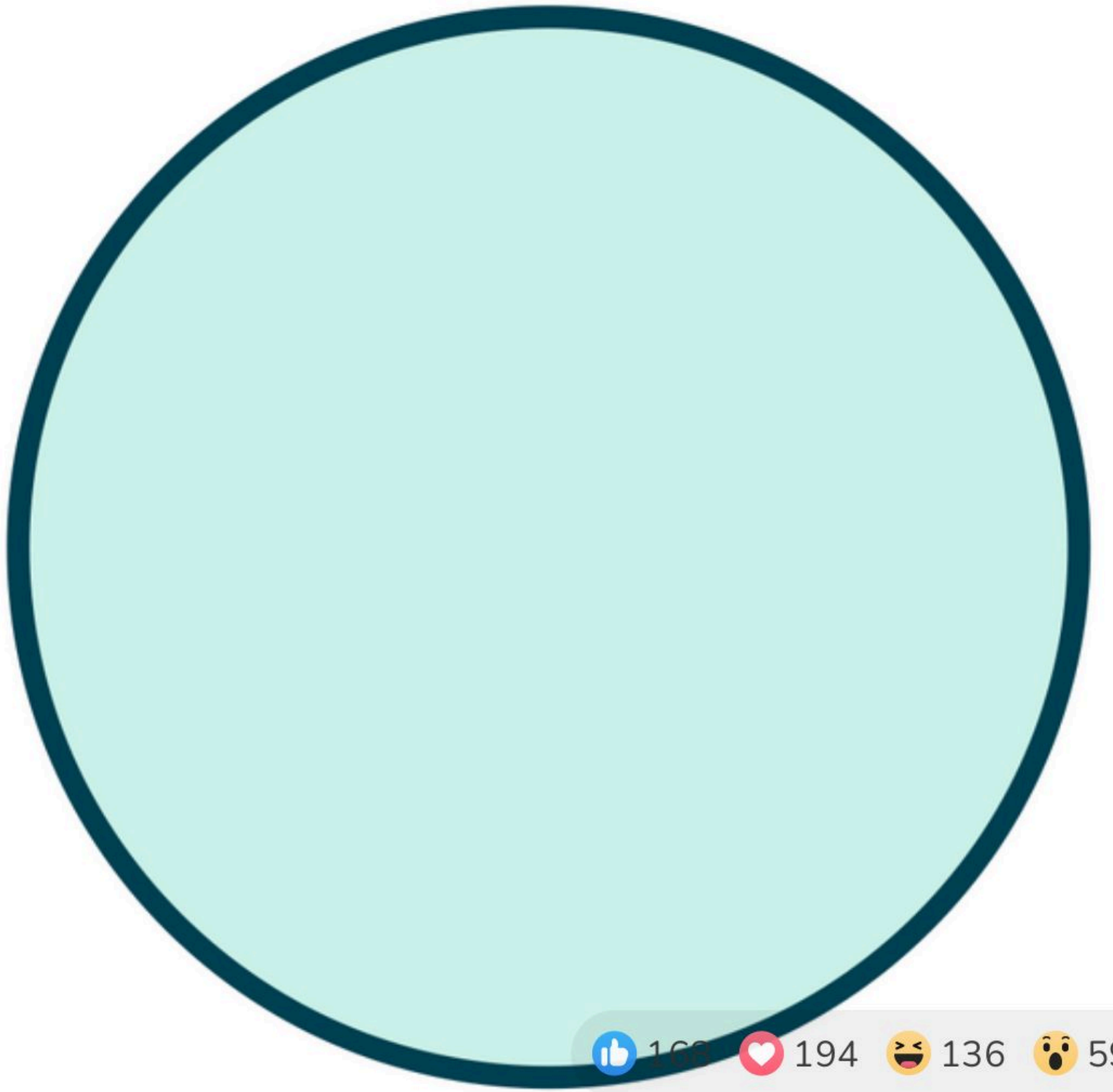
https://github.com/ddd-crew/ddd-starter-modelling-proces

A Bounded Context is a boundary for a model expressed in a consistent (ubiquitous) language tailored around a specific purpose

Bounded Context

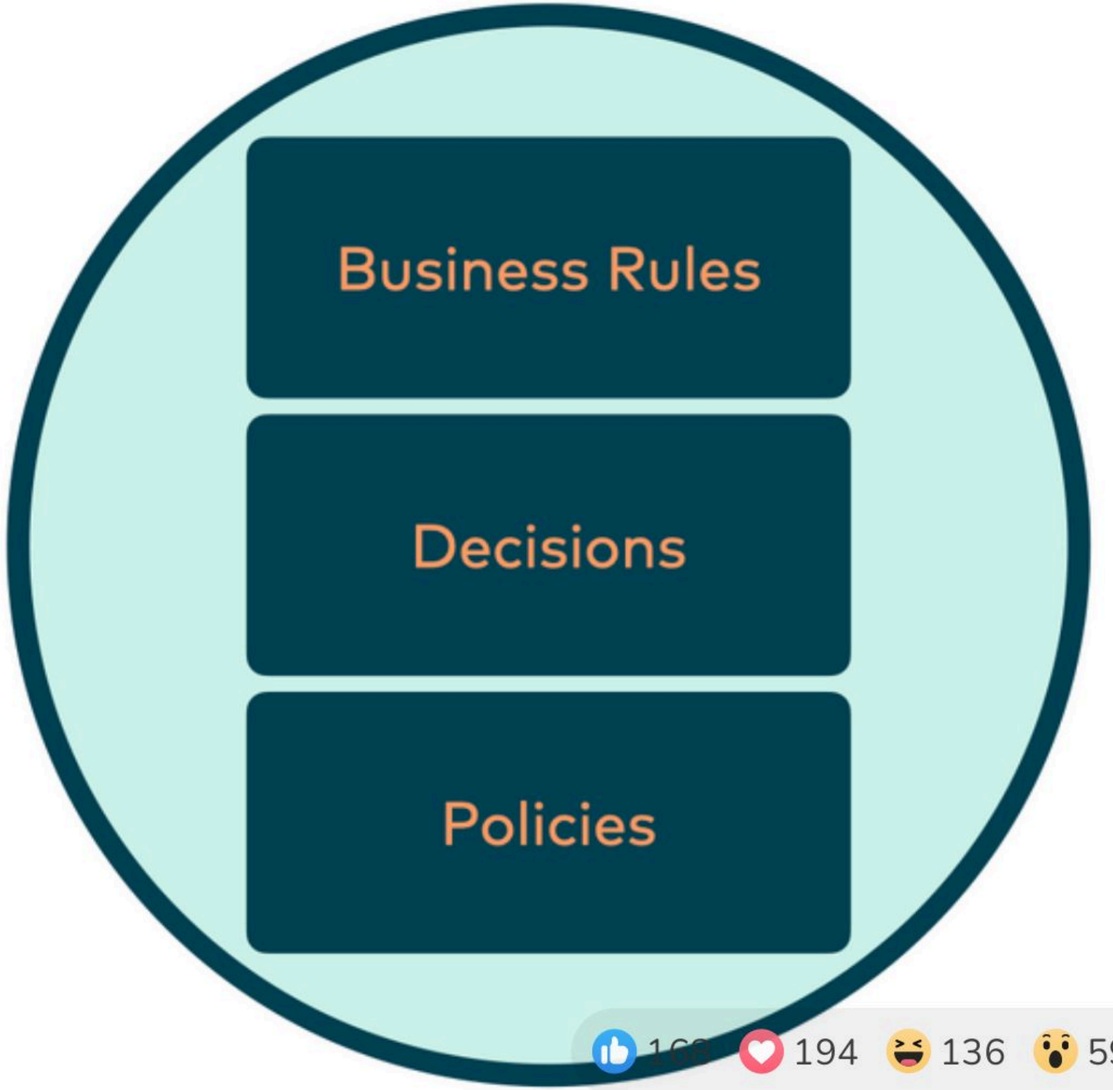
A Bounded Context is a **boundary** for a model expressed in a consistent language tailored around a specific purpose

Boundary

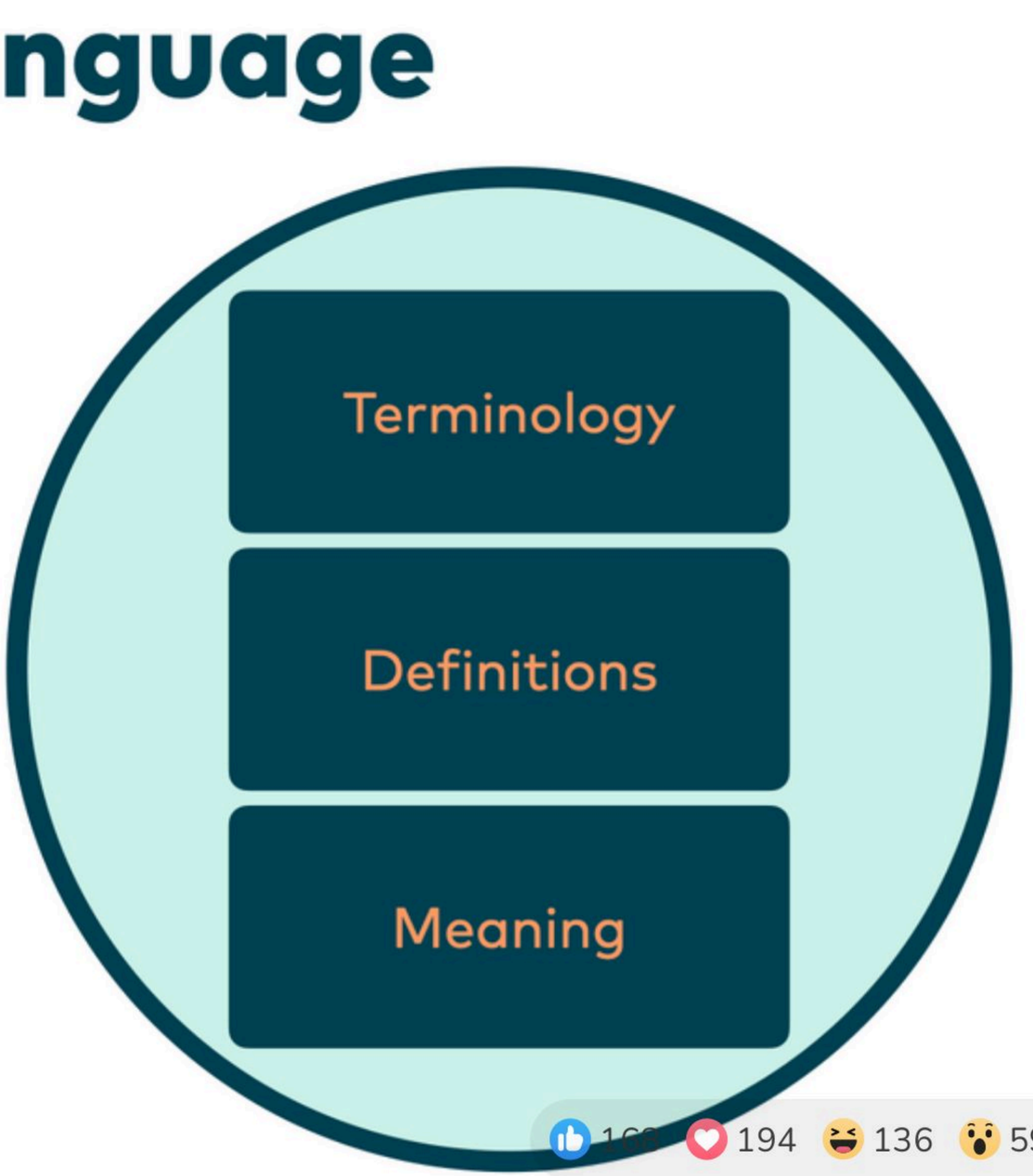


A Bounded Context is a boundary for a **model** expressed in a consistent language tailored around a specific purpose

Boundary for a **model**

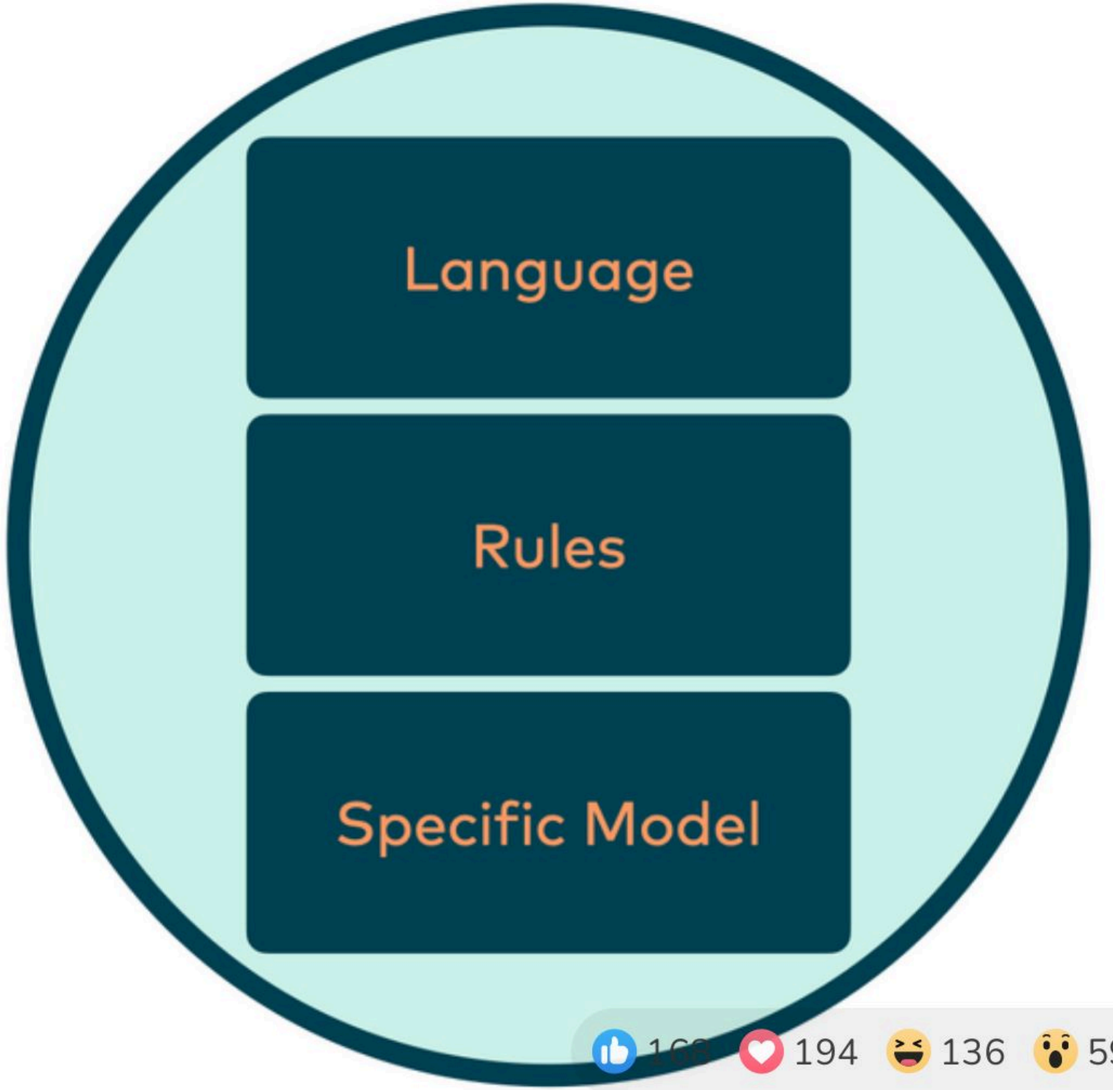


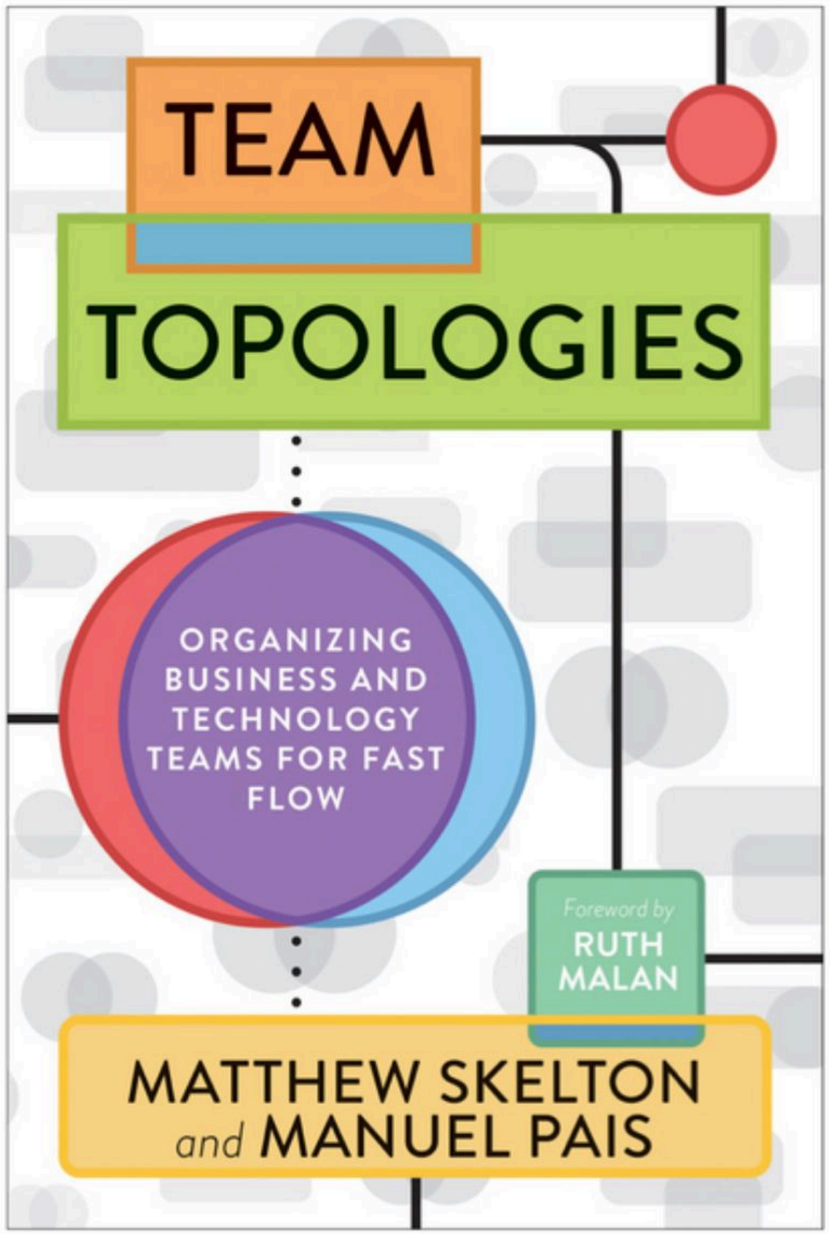
A Bounded Context is a boundary for a model expressed in a consistent **language** tailored around a specific purpose



A Bounded Context is a boundary for a model expressed in a consistent language tailored around a specific **purpose**

Purpose





The
Bounded Context
is a

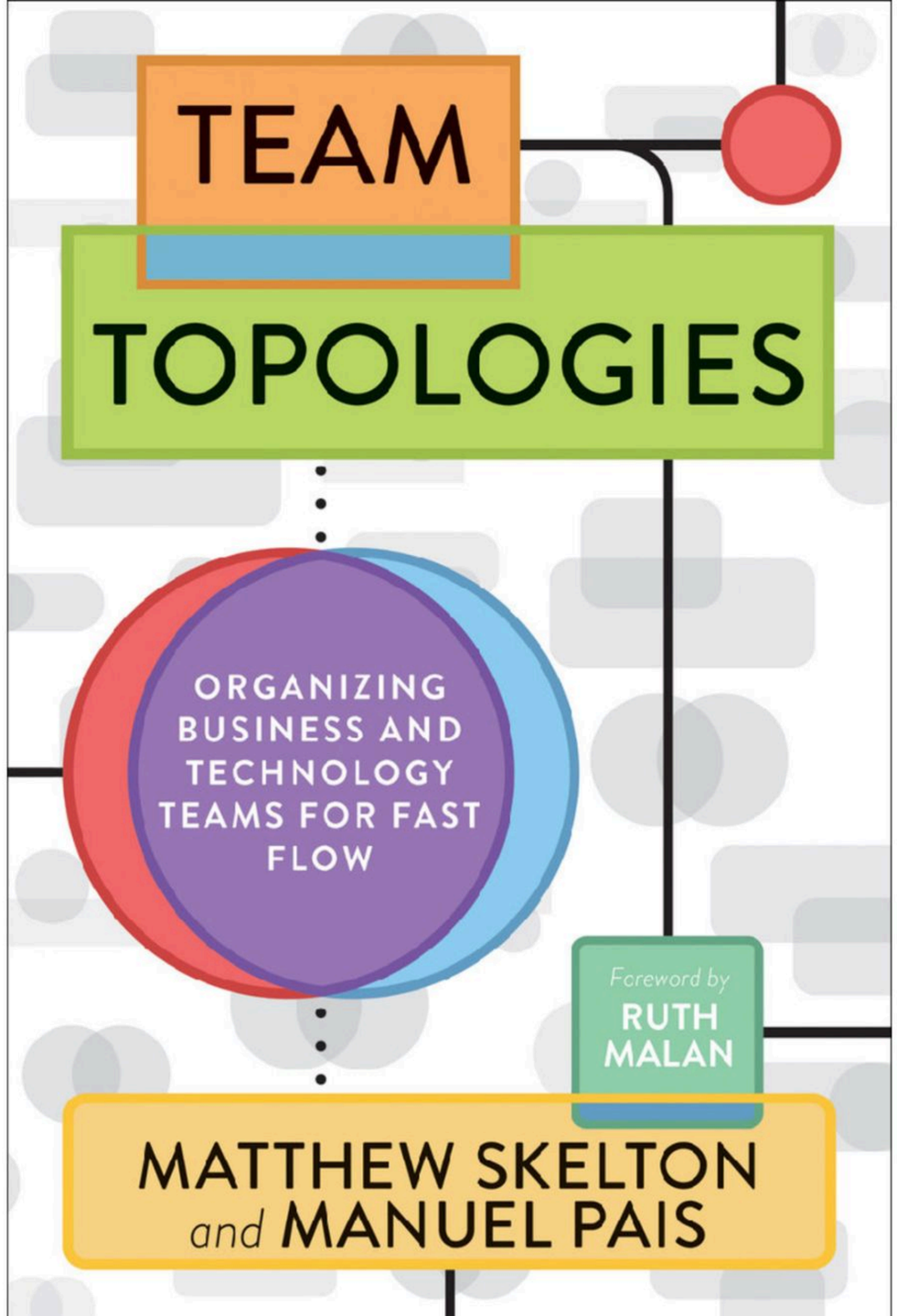
team first

boundary

168 194 136 59 43

„Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.“

Melvin Conway



"An architect should be thinking:

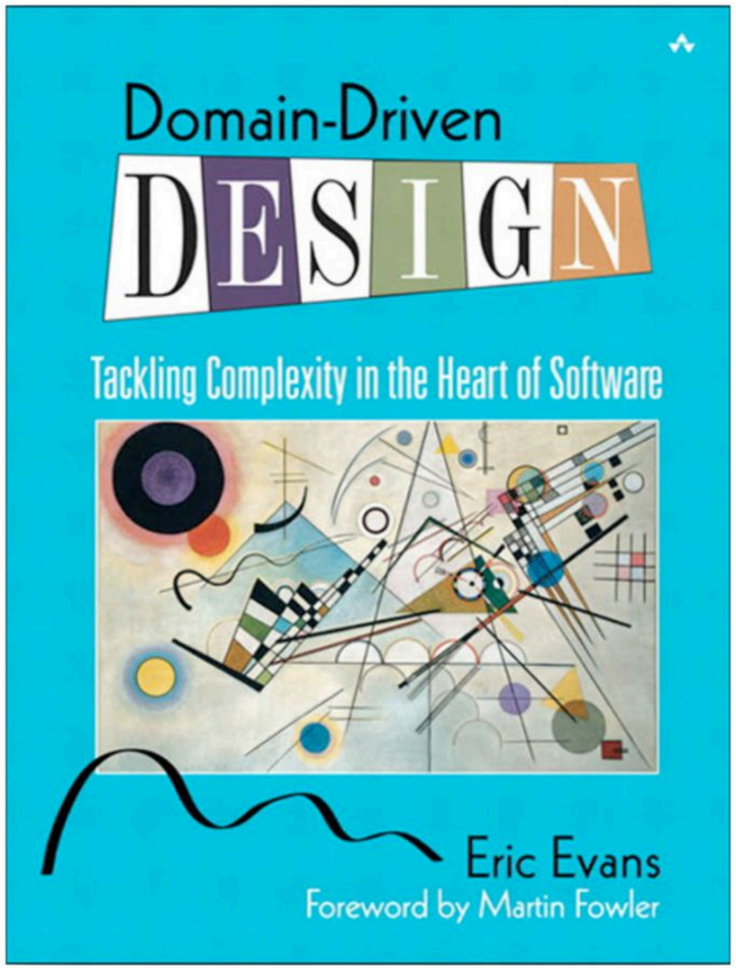
Which team interaction modes are appropriate for these two teams?

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

168 194 136 59 43

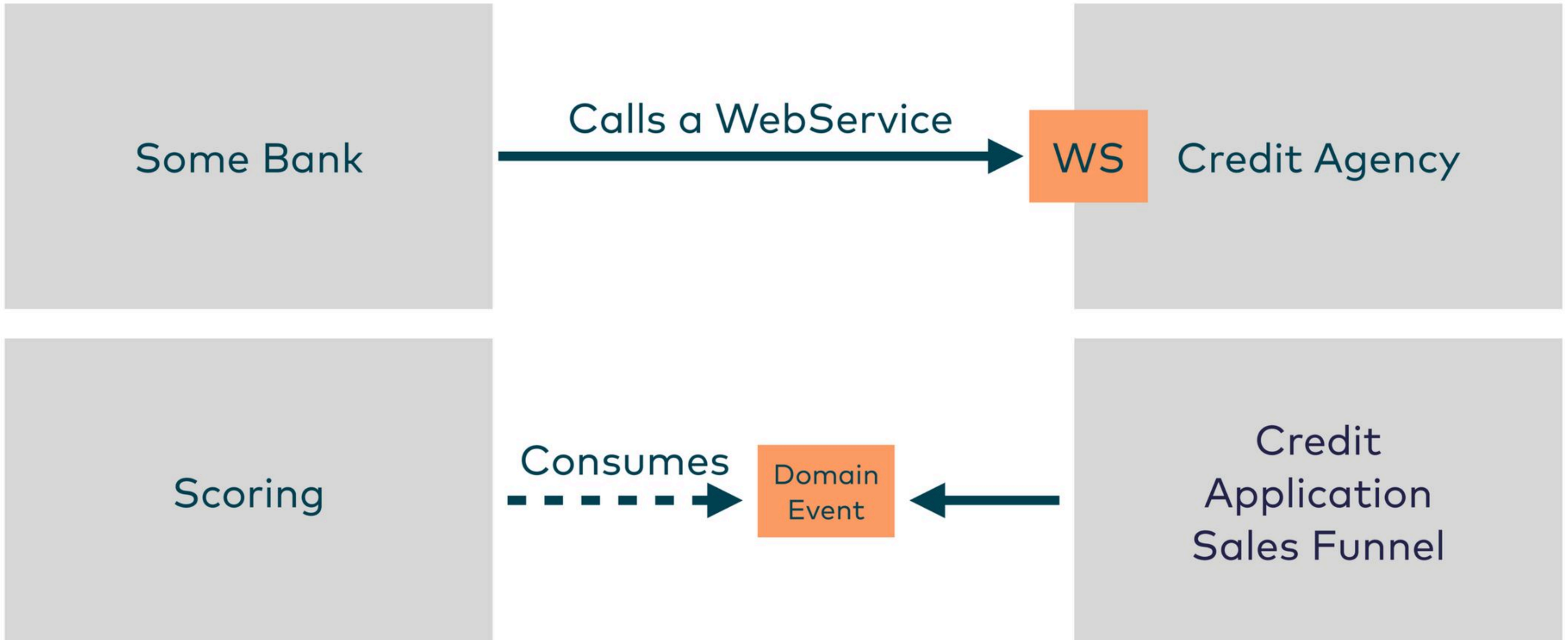
Strategic Domain Driven Design has a technique to visualize relationships between Bounded Contexts and teams:

CONTEXT MAPS

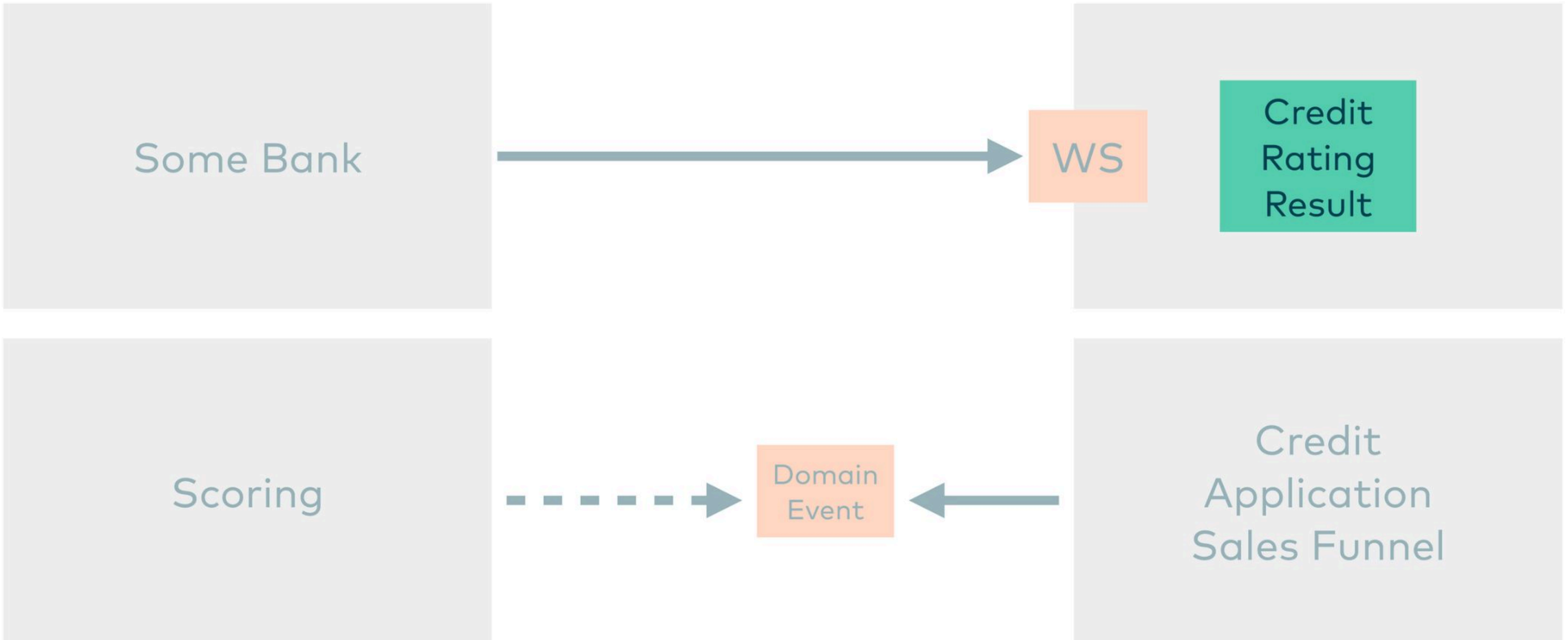


Which kinds of relationships between systems and teams do context maps address?

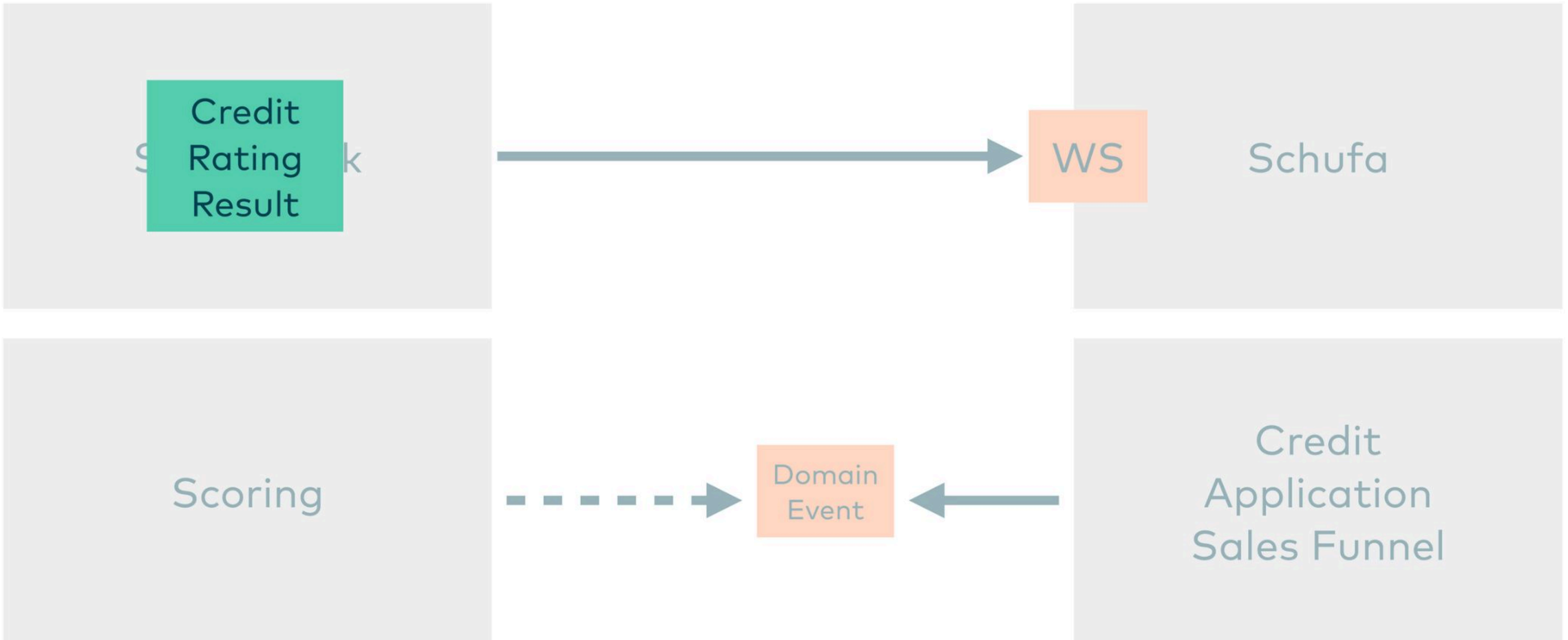
Systems calling each other or consuming events



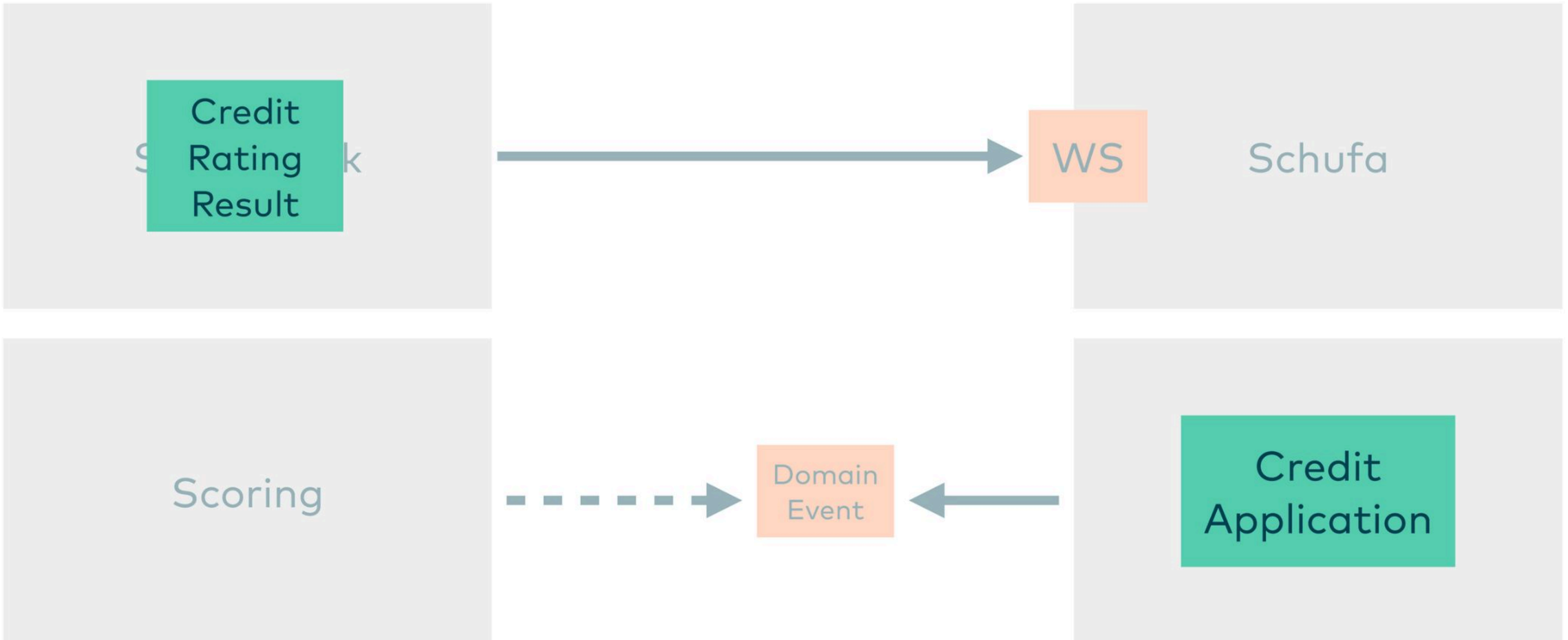
The propagation of models



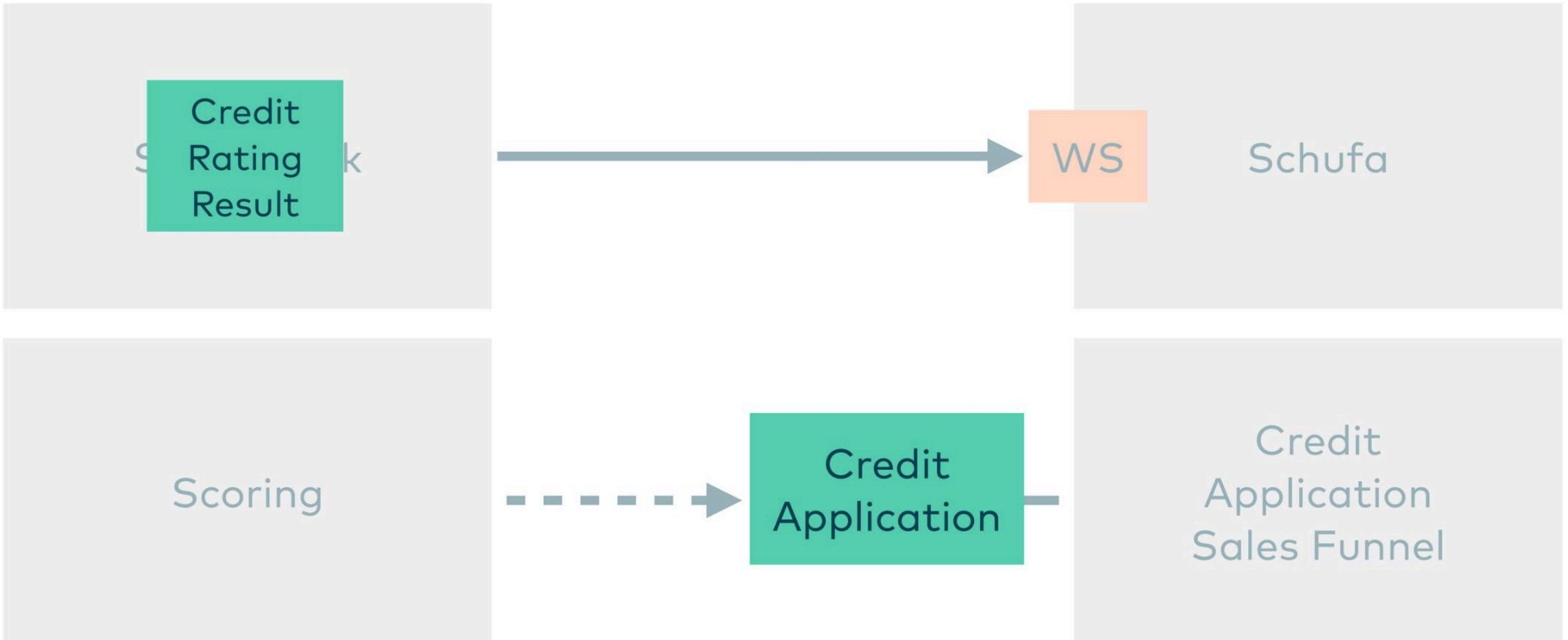
The propagation of models



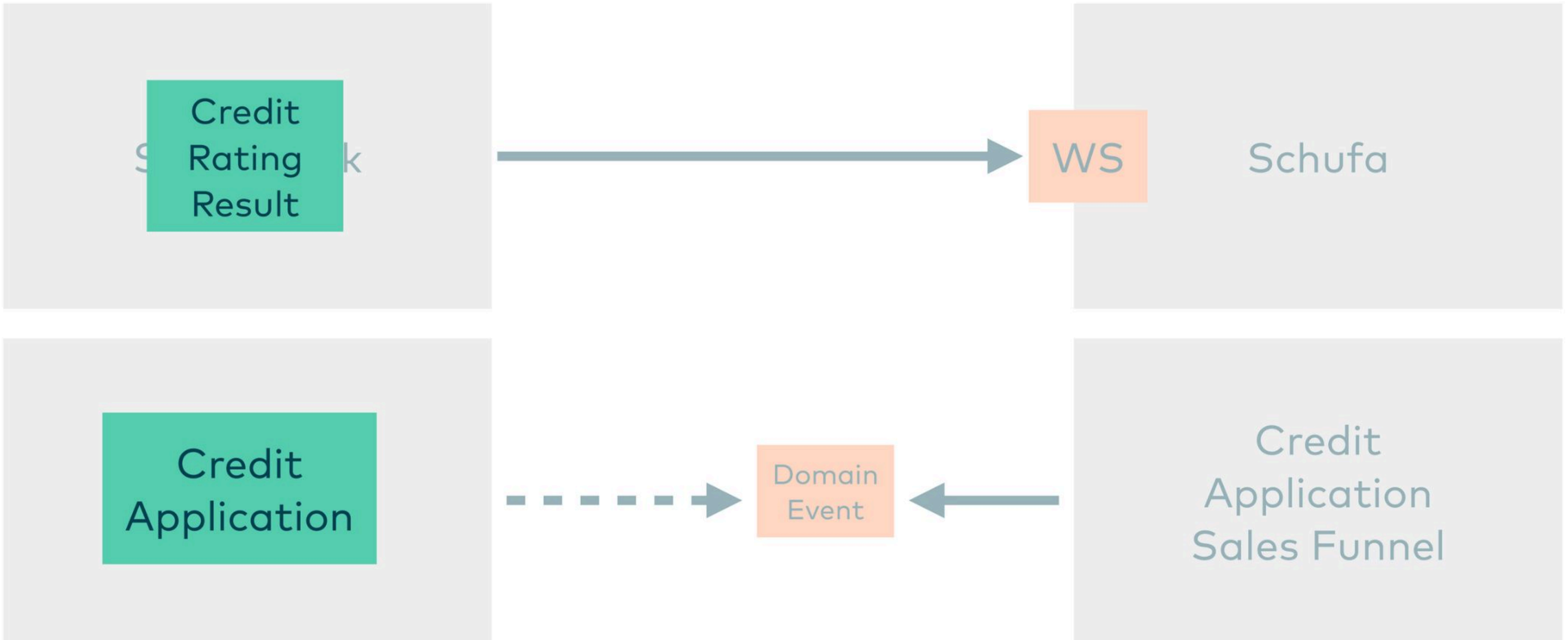
The propagation of models



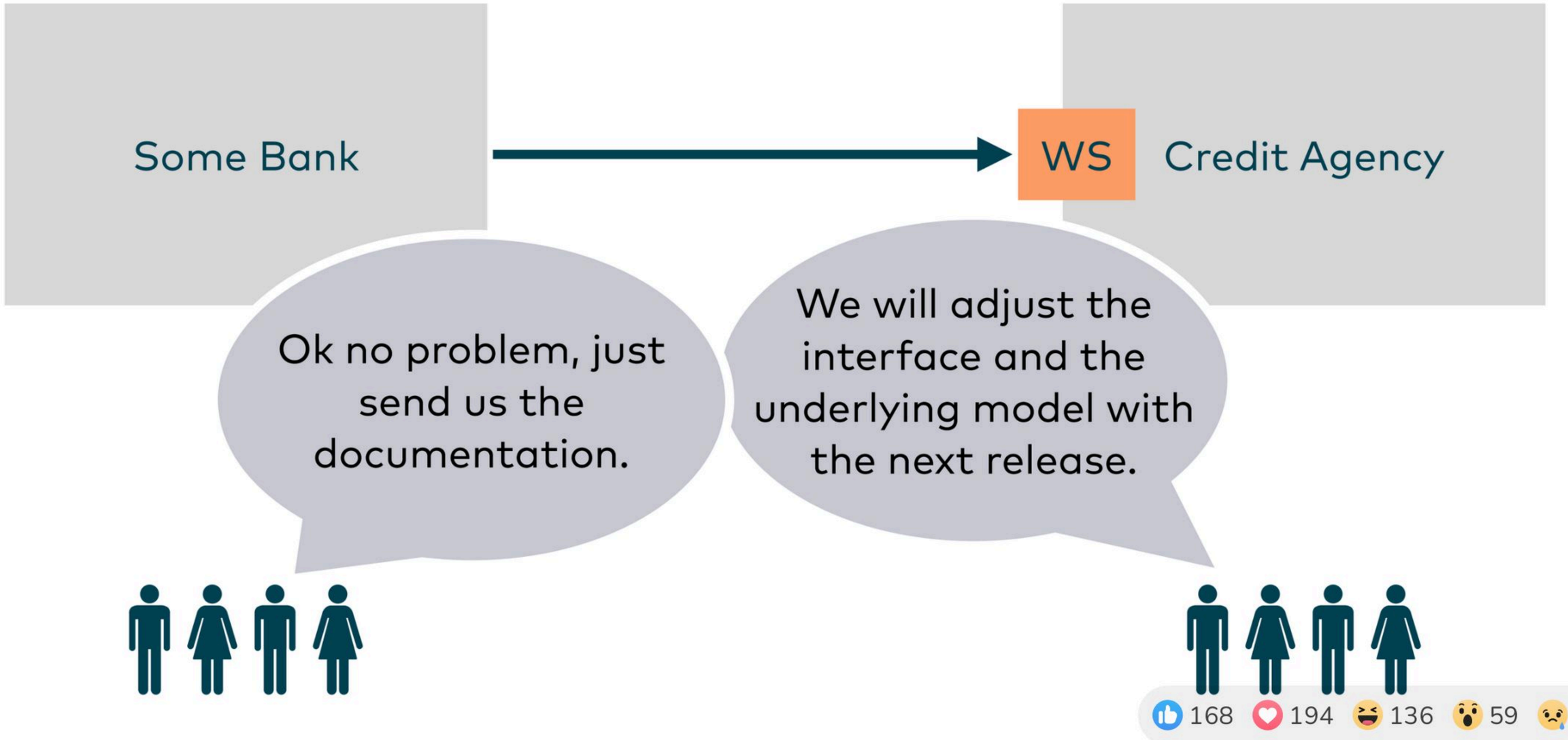
The propagation of models



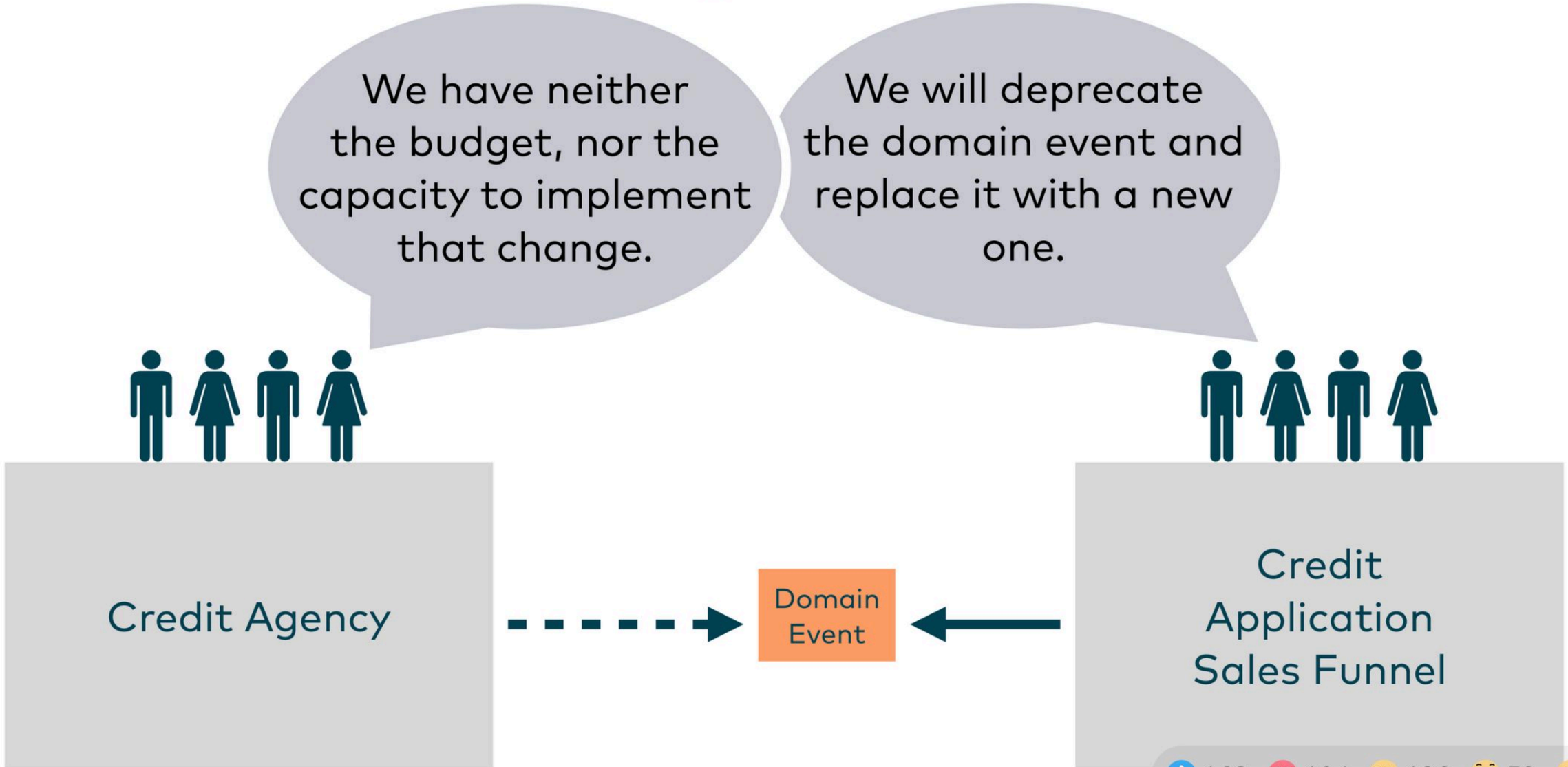
The propagation of models



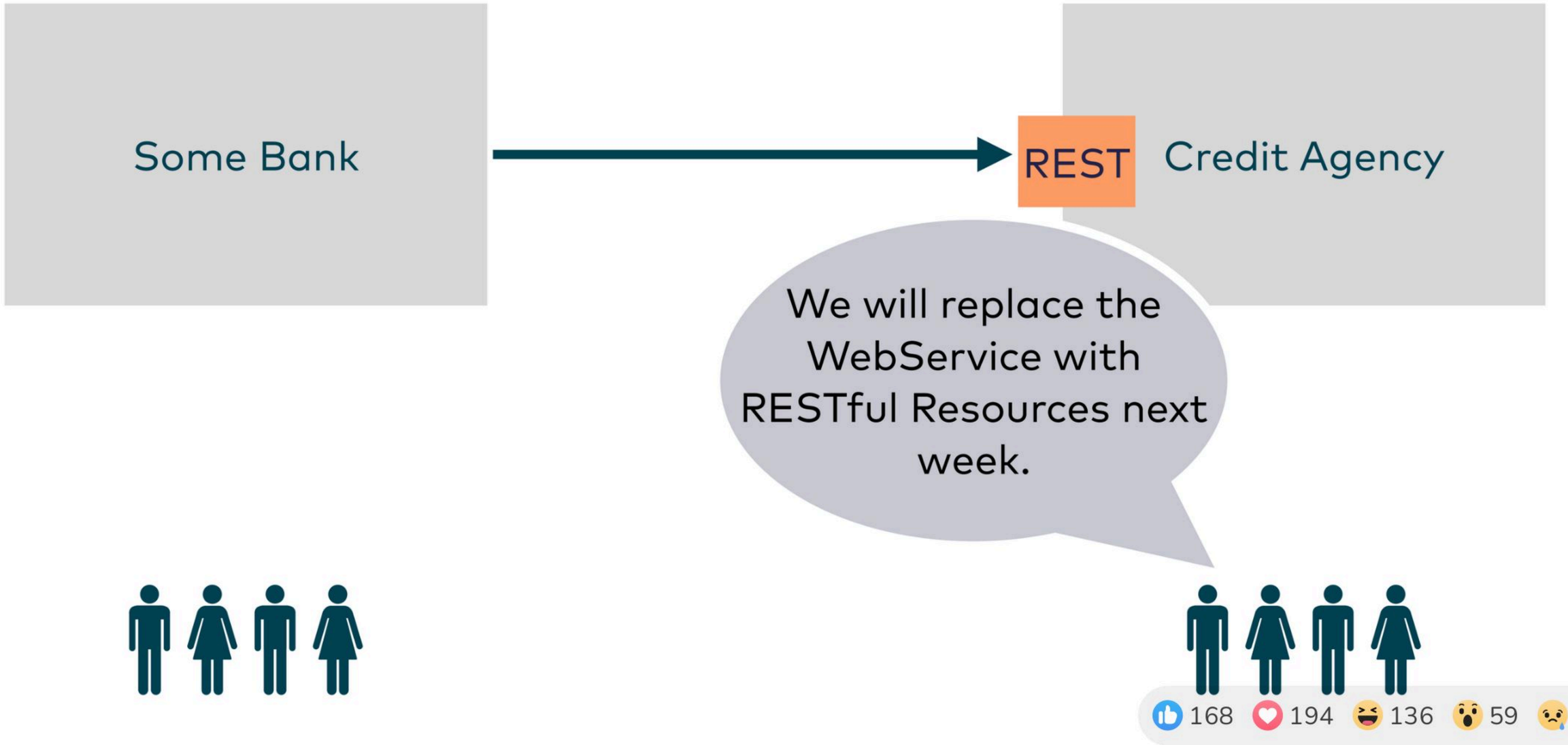
Teams communicating with each other



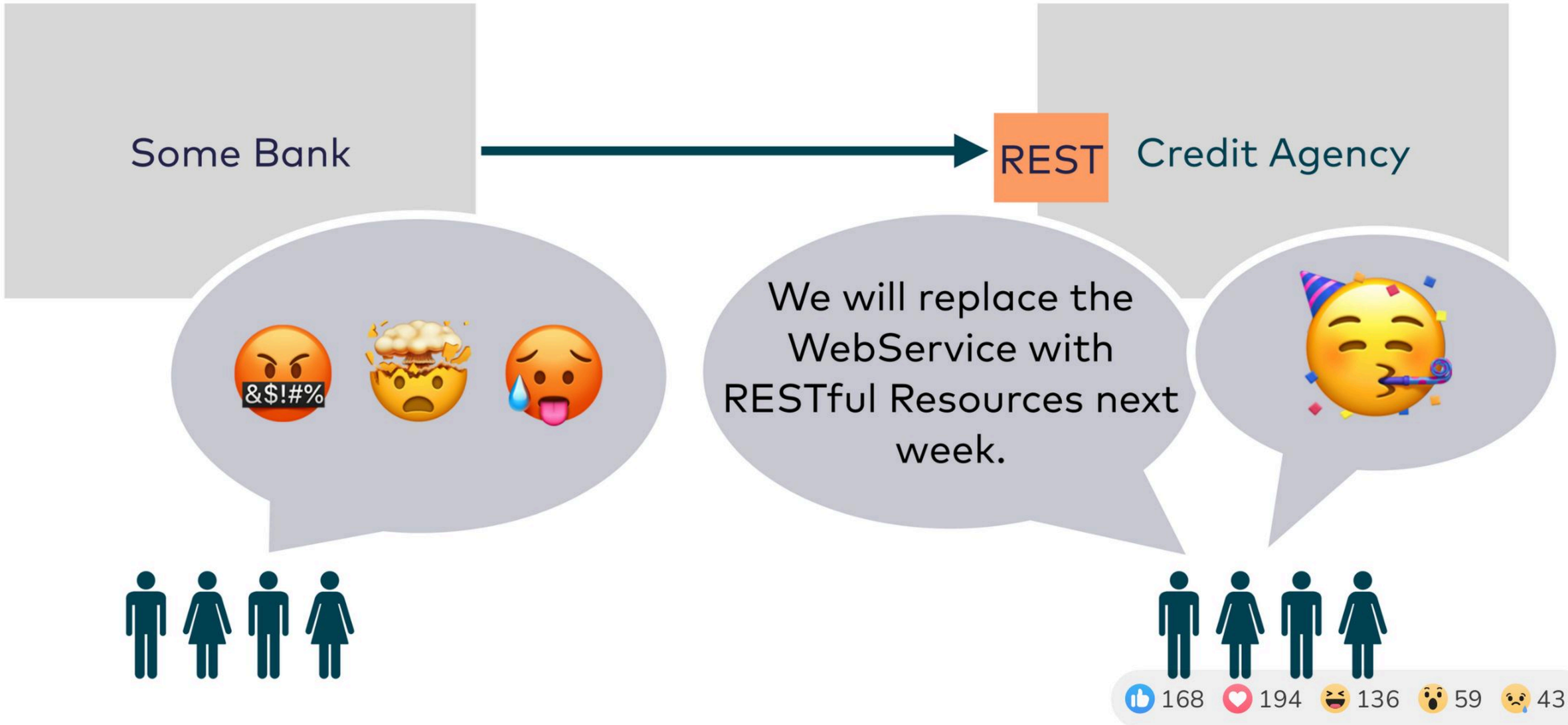
Teams communicating with each other



Actions of one team have an impact on others



Actions of one team have an impact on others



Dependencies between teams

Team Dependencies

Mutually Dependent

- Two software artifacts or systems in two bounded contexts need to be delivered together to be successful and work.
- There is often a close, reciprocal link between data and functions between the two systems.

Dependencies between teams

Team Dependencies

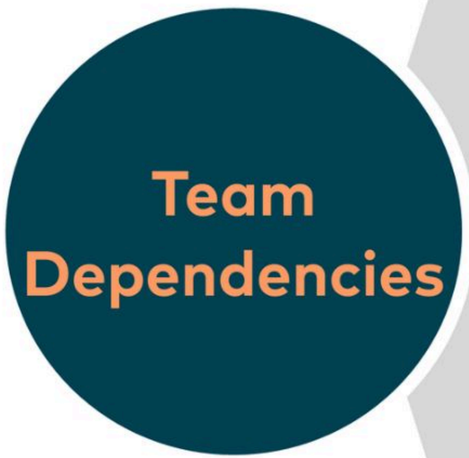
Mutually Dependent

- Two software artifacts or systems in two bounded contexts need to be delivered together to be successful and work.
- There is often a close, reciprocal link between data and functions between the two systems.

Free

- Changes in one bounded context do not influence success or failure in other bounded contexts.
- There is, therefore, no organizational or technical link of any kind between the teams.

Dependencies between teams



Mutually Dependent

- Two software artifacts or systems in two bounded contexts need to be delivered together to be successful and work.
- There is often a close, reciprocal link between data and functions between the two systems.

Free

- Changes in one bounded context do not influence success or failure in other bounded contexts.
- There is, therefore, no organizational or technical link of any kind between the teams.

Upstream / Downstream

- An upstream context will influence the downstream counterpart while the opposite might not be true.
- This might apply to code but also on less technical factors such as schedule or responsiveness to external requests.

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

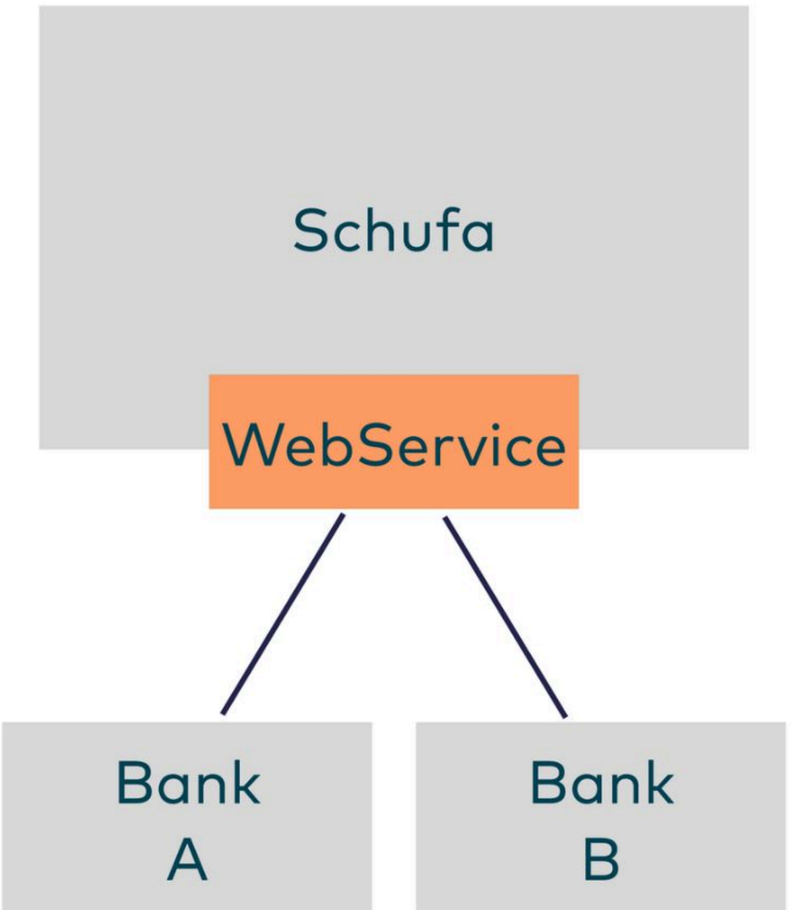
- Partnership
- Shared Kernel
- Customer / Supplier
- Conformist
- Anticorruption Layer
- Separate Ways
- Open / Host Service
- Published Language
- Big Ball Of Mud

These patterns address a diverse variety of perspectives

Open-host Service

The Open-host Service is a public API

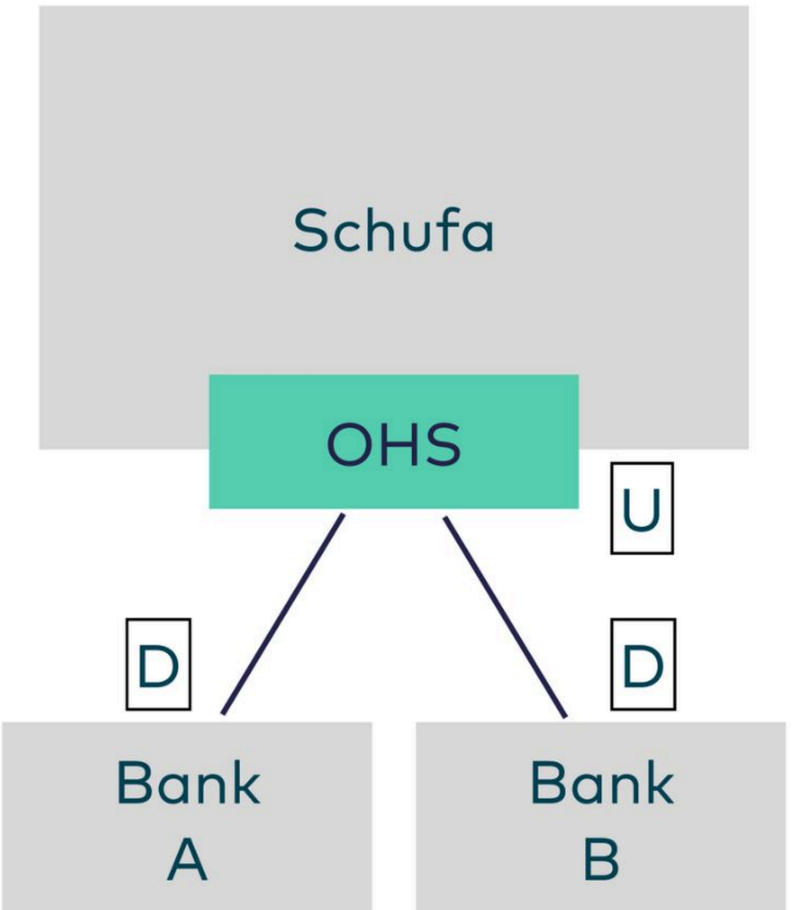
- One API for several consumers
- No point-to-point API
- Has a common, general purpose model and functionality
- The team providing the Open-host Service is an upstream team



Open-host Service

The Open-host Service is a public API

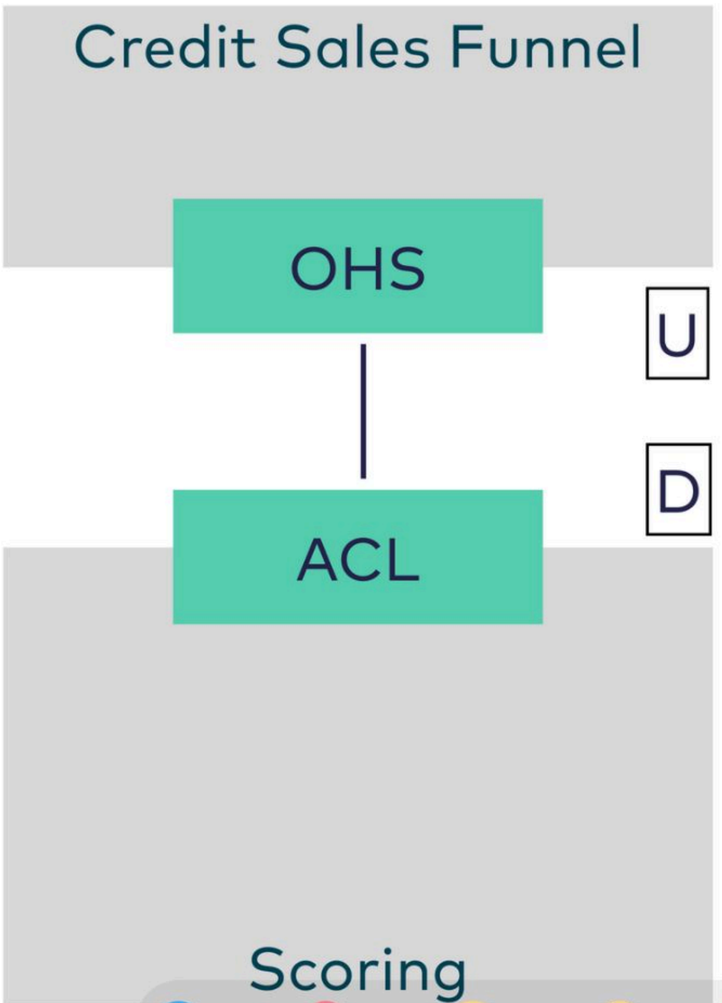
- One API for several consumers
- No point-to-point API
- Has a common, general purpose model and functionality
- The team providing the Open-host Service is an upstream team



Anticorruption Layer

The Anticorruption Layer translates one model to another one

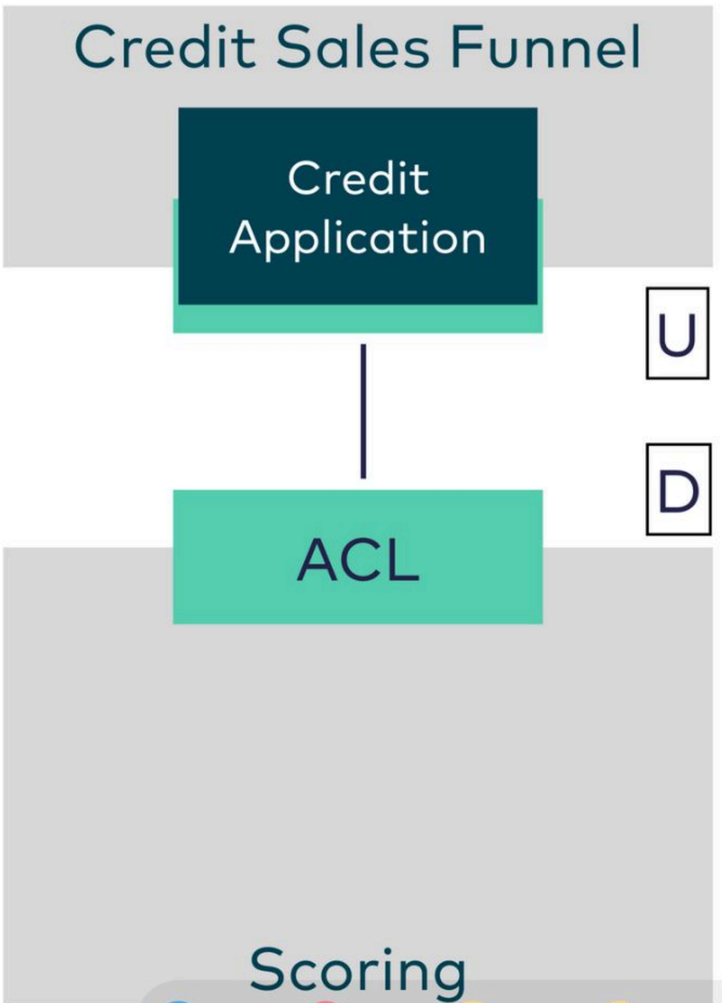
- Transforms an external model from another team / bounded context / system to another internal one
- Reduces the amount of coupling to a single layer
- The team implementing an Anticorruption Layer is always downstream



Anticorruption Layer

The Anticorruption Layer translates one model to another one

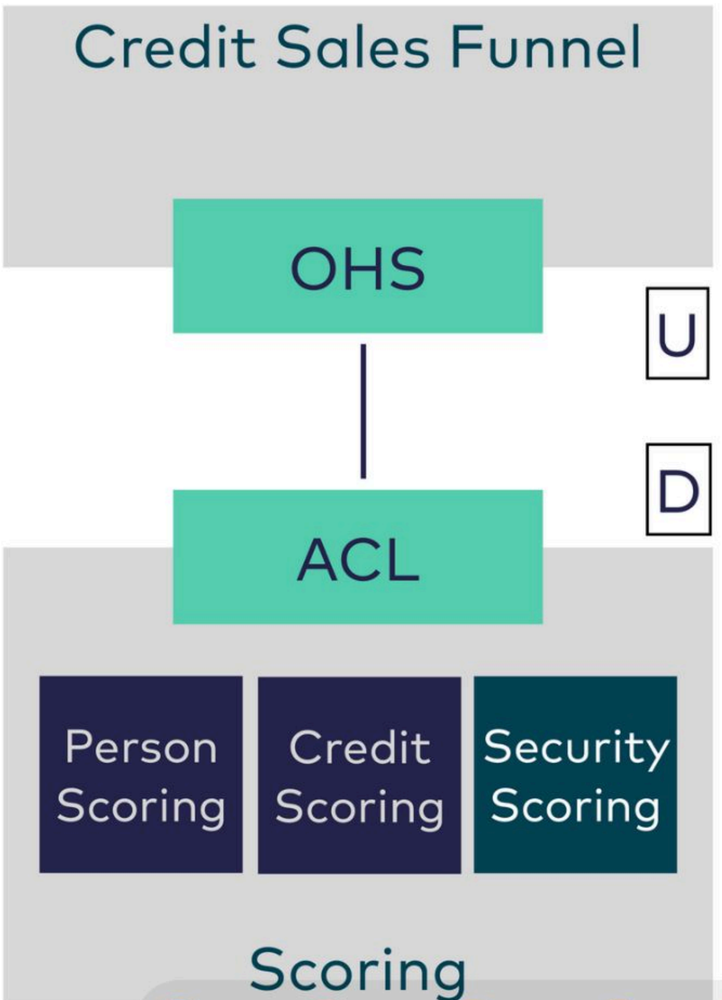
- Transforms an external model from another team / bounded context / system to another internal one
- Reduces the amount of coupling to a single layer
- The team implementing an Anticorruption Layer is always downstream



Anticorruption Layer

The Anticorruption Layer translates one model to another one

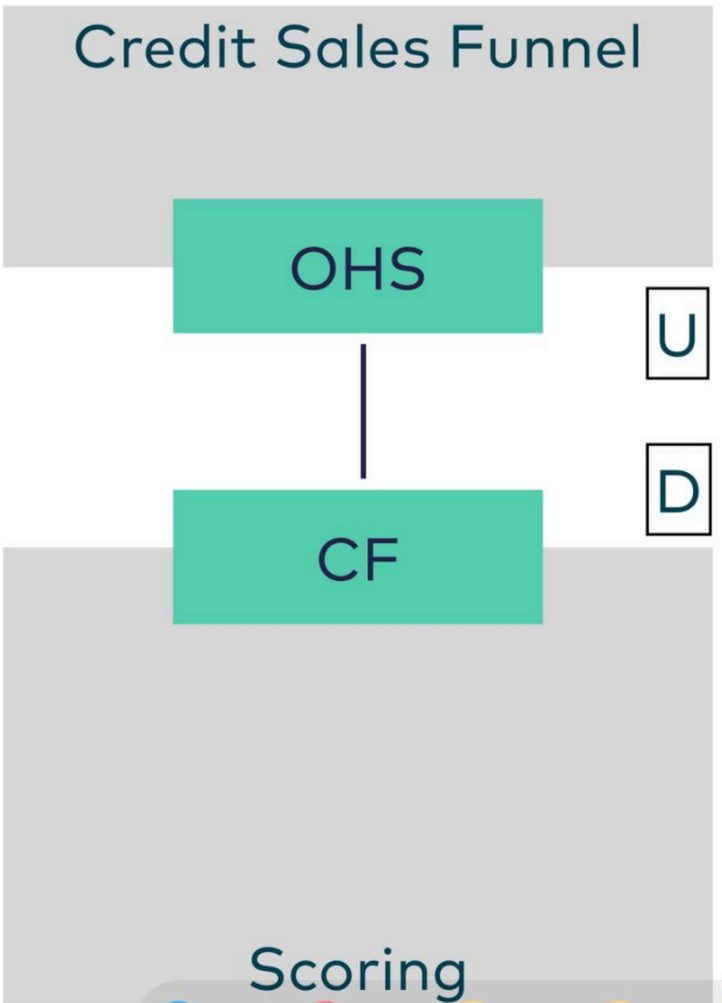
- Transforms an external model from another team / bounded context / system to another internal one
- Reduces the amount of coupling to a single layer
- The team implementing an Anticorruption Layer is always downstream



Conformist

The Conformist slavishly adheres to the upstream model

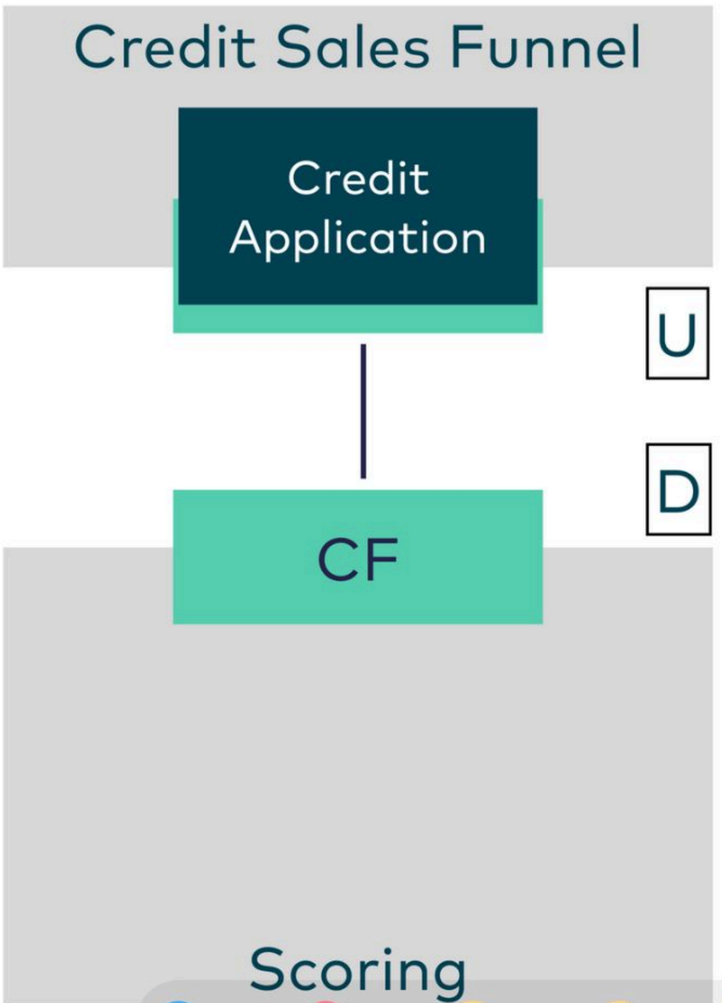
- There is no model-to-model transformation
- Motivation: Simplicity, contracts, force or delight (for the upstream model)
- The team implementing a Conformist is always downstream



Conformist

The Conformist slavishly adheres to the upstream model

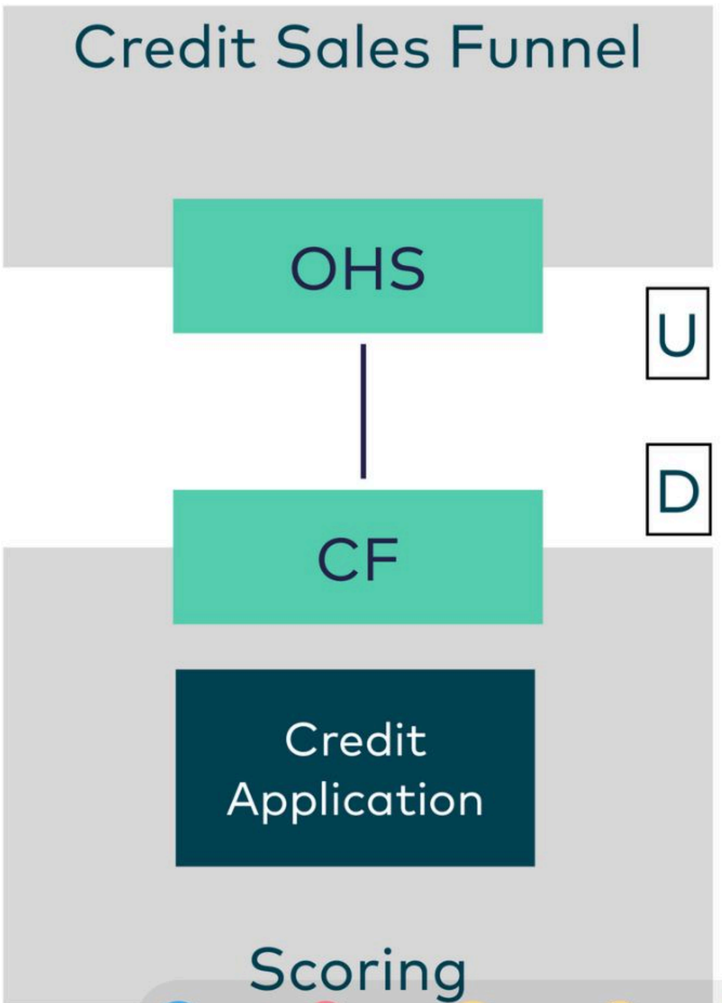
- There is no model-to-model transformation
- Motivation: Simplicity, contracts, force or delight (for the upstream model)
- The team implementing a Conformist is always downstream

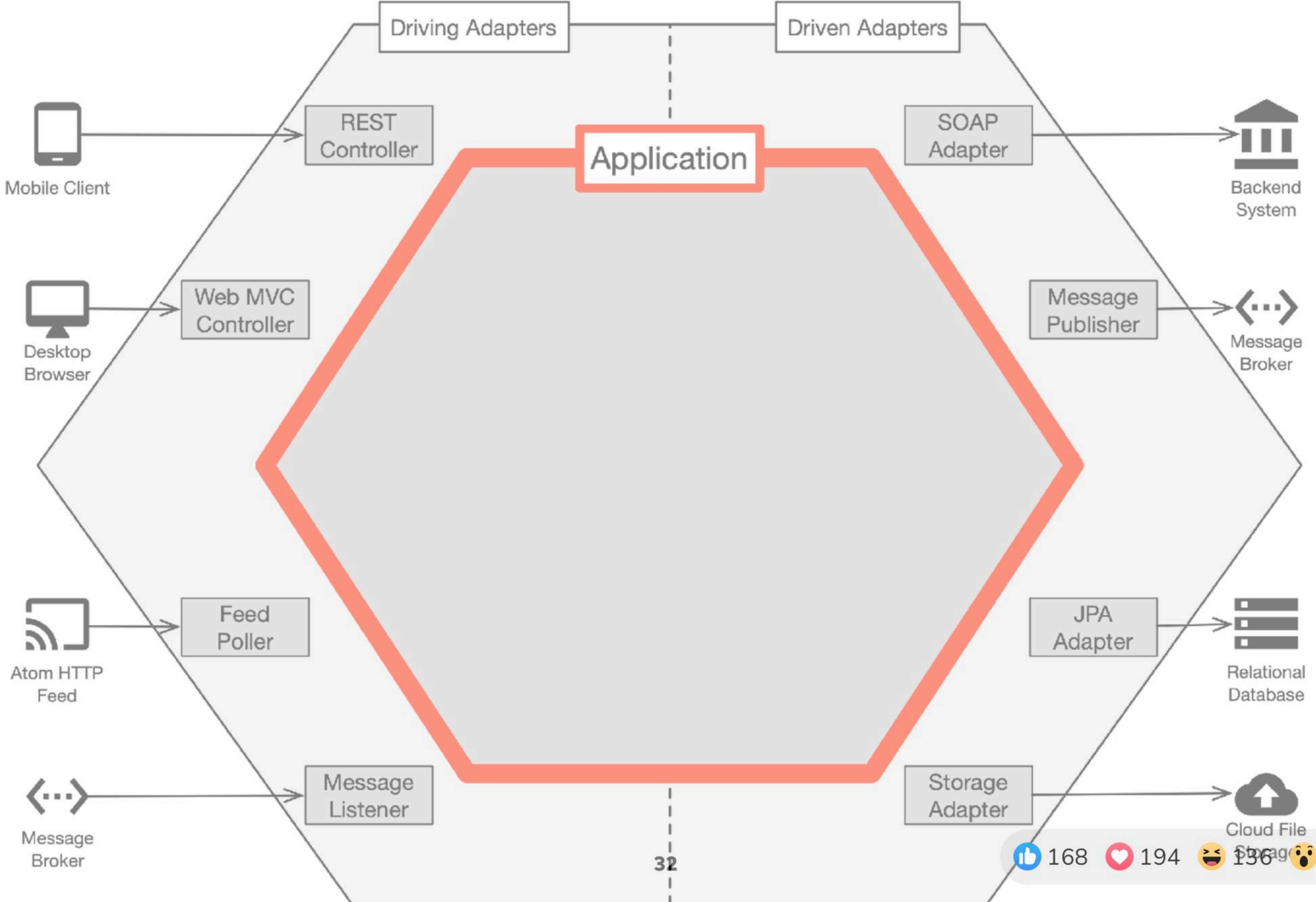


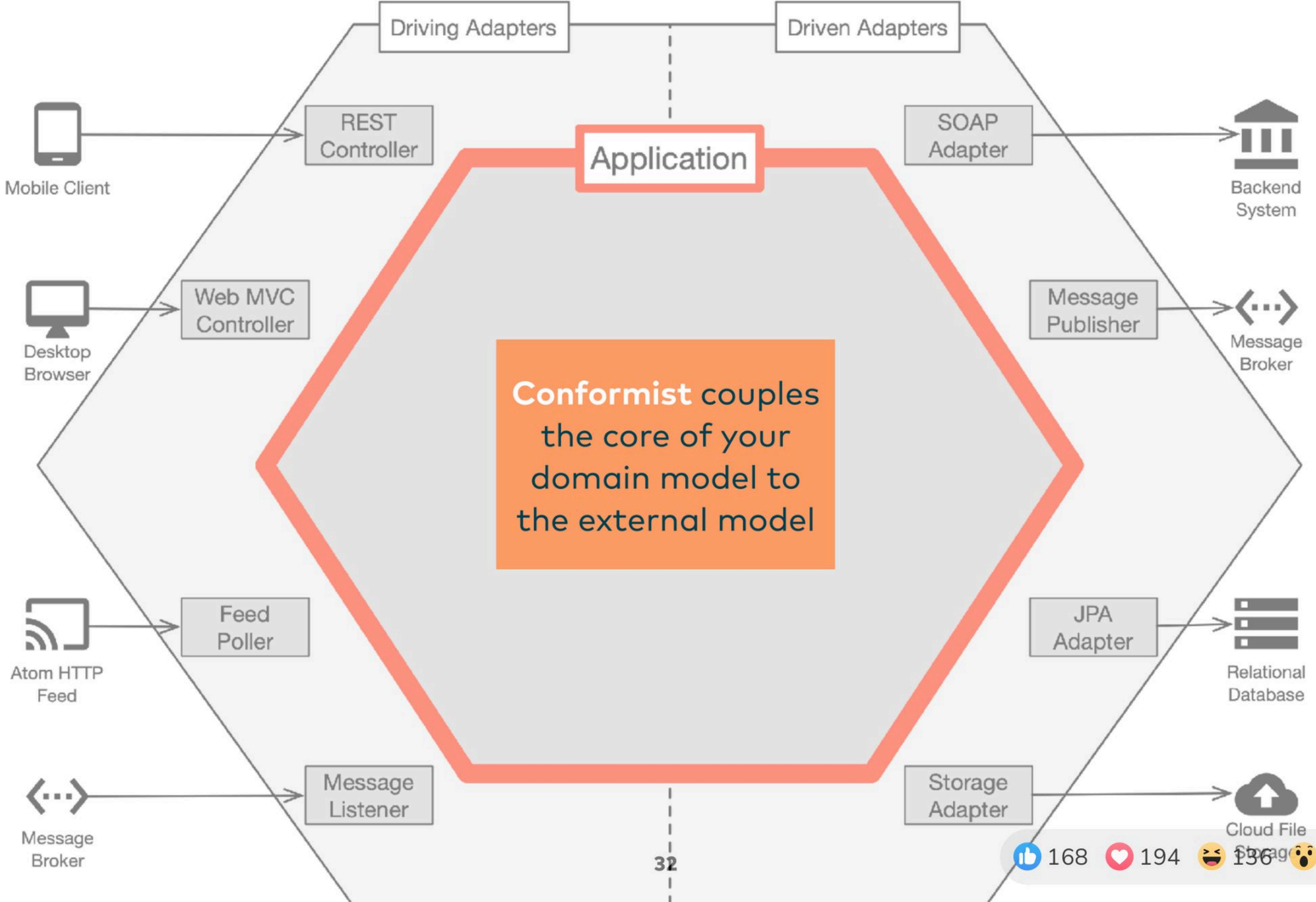
Conformist

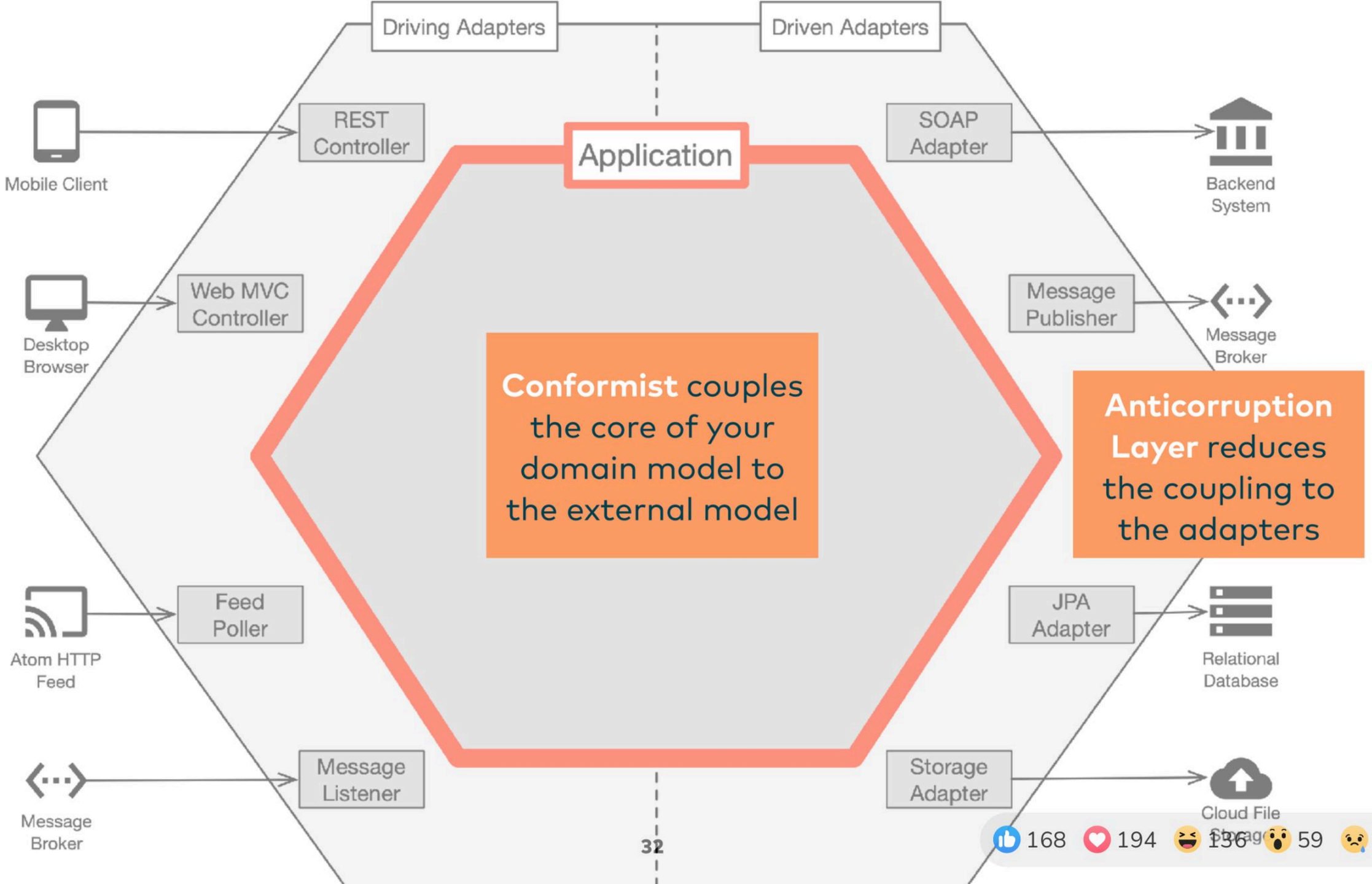
The Conformist slavishly adheres to the upstream model

- 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









Heuristics for choosing a Conformist

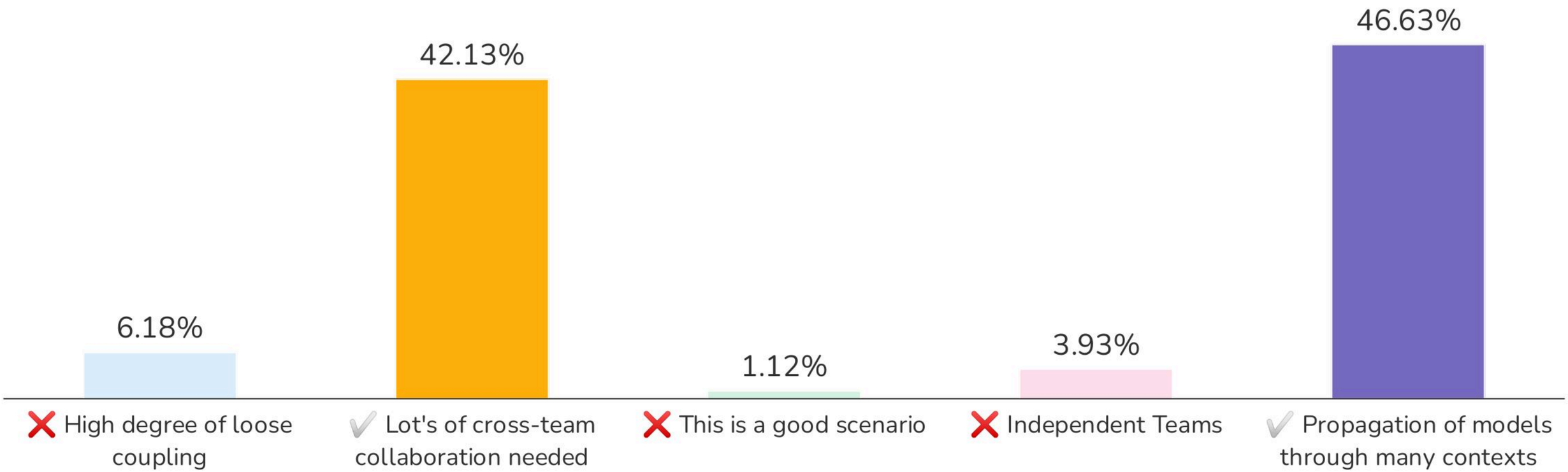
The degree of coupling and also connascence of a Conformist is higher compared to an Anticorruption Layer but there are a few situations in which a Conformist may still be the better choice.

- 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.

Scan this QR code to join →

168 194 136 59 43

Which comments apply best to this Context Map?

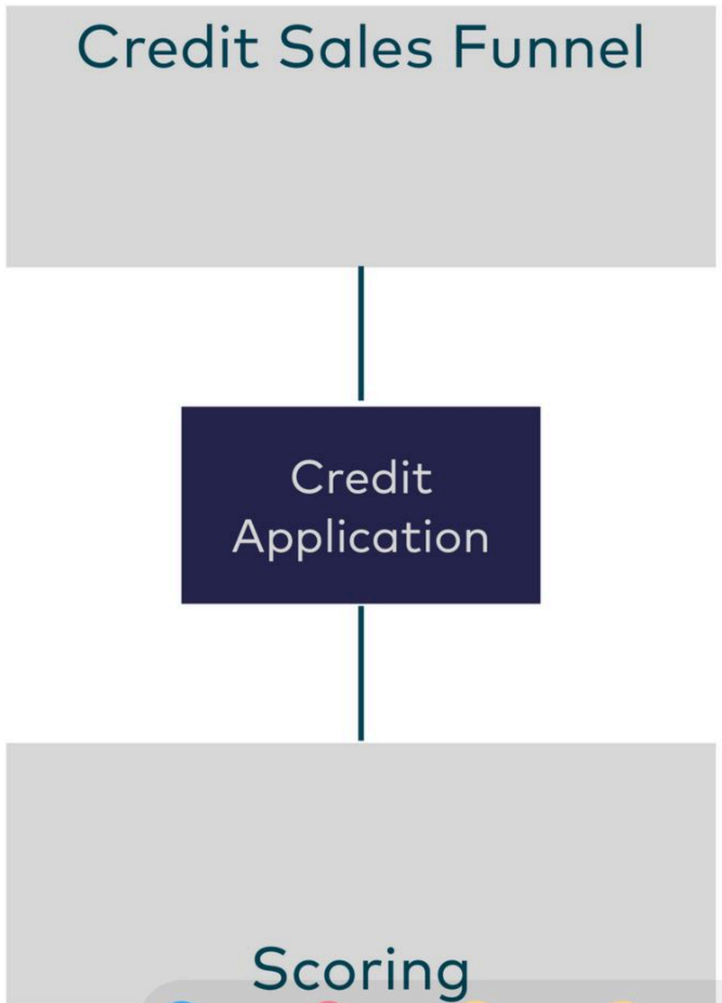


168 194 136 59 43 102

Shared Kernel

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

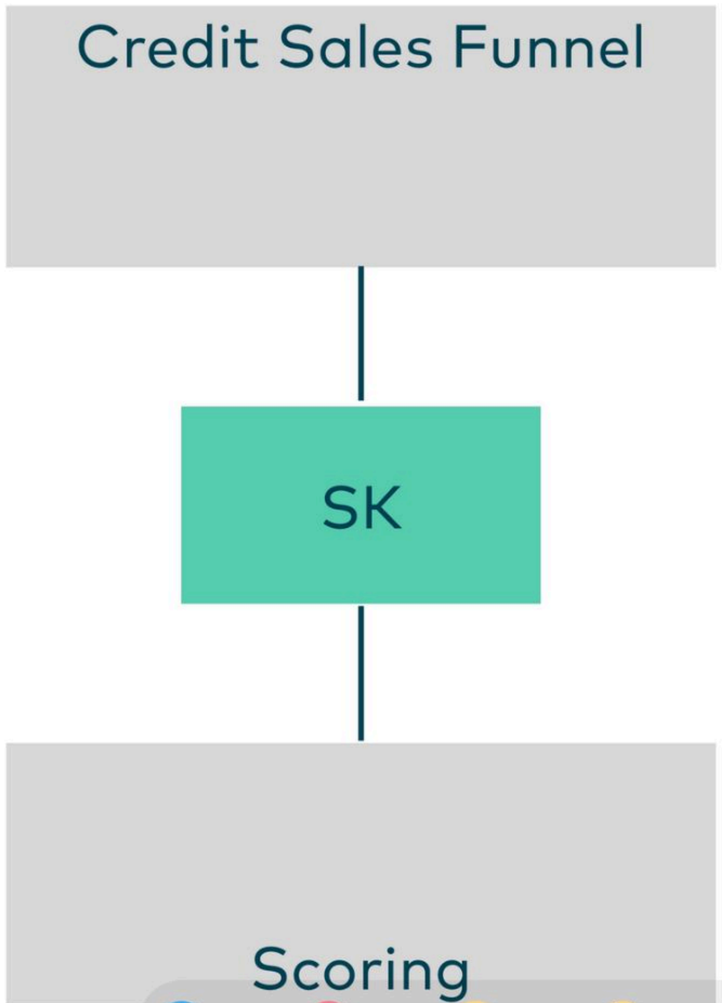
- „Physically“ shared artifact between two teams
- Examples: shared JARs or database
- High degree of coupling requires a high amount of coordination between the involved teams
- Shared Kernel is no Anti-Pattern but use with caution



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:

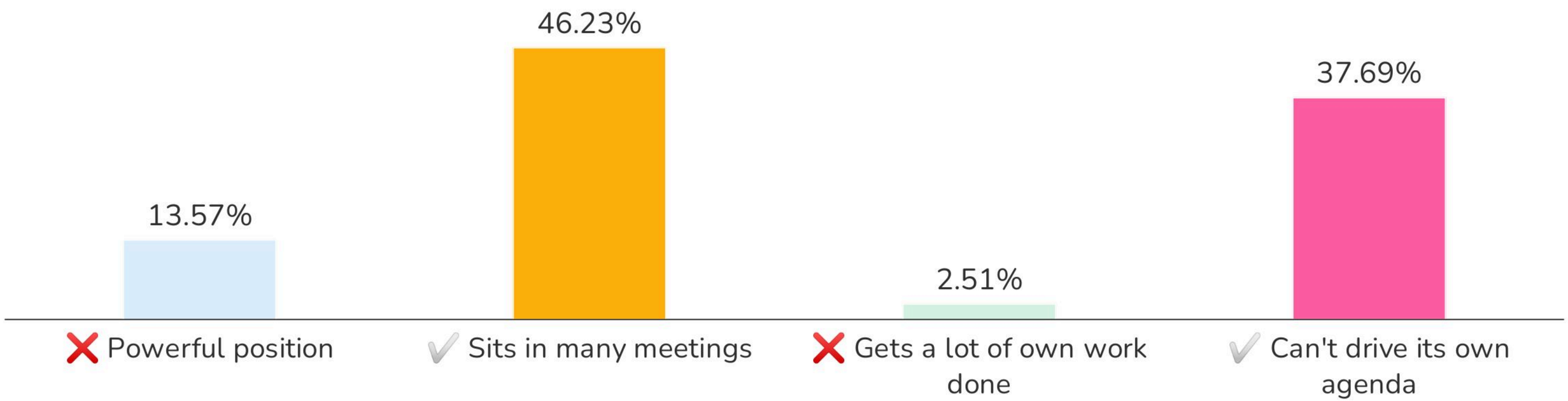
- One team is responsible for two or more bounded contexts which have an overlap in terms of language
- Strictly avoid a Shared Kernel when two teams are in a competitive situation or where a lot of backdoor politics are being played
- When two teams have a Shared Kernel, they should form a Partnership...

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

Which comments apply best regarding to the team in the middle

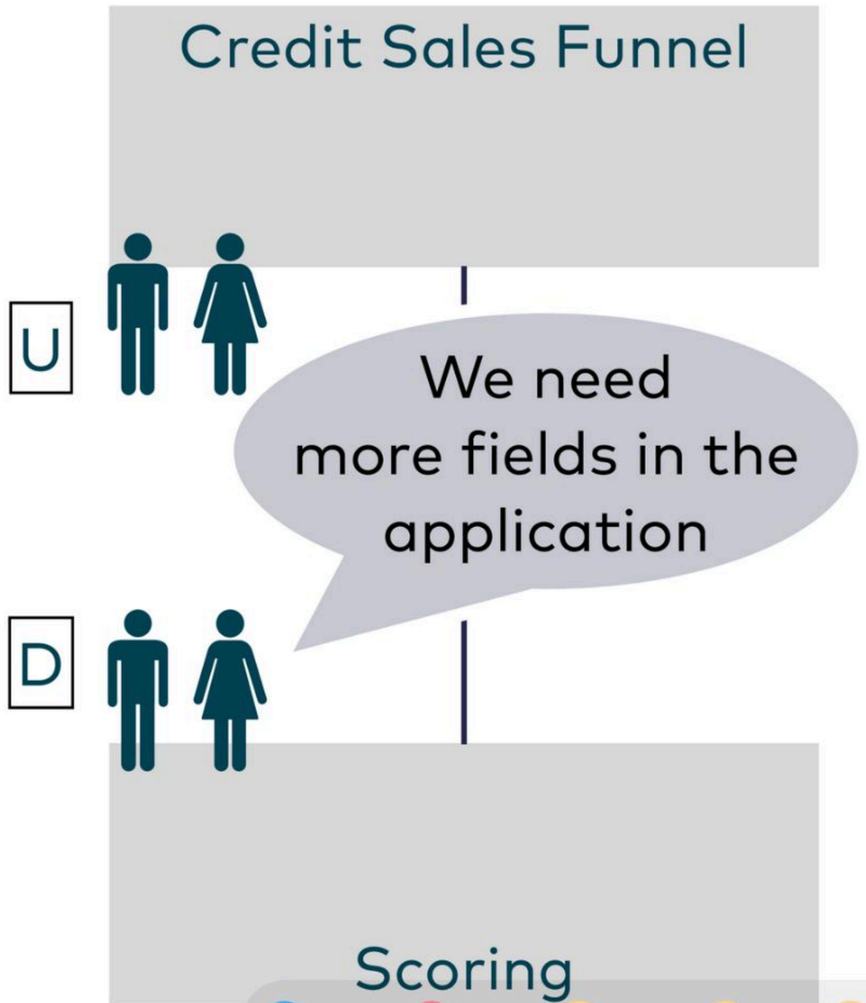


168 194 136 59 43 99

Customer-Supplier

A Customer-Supplier development gives the downstream team some influence

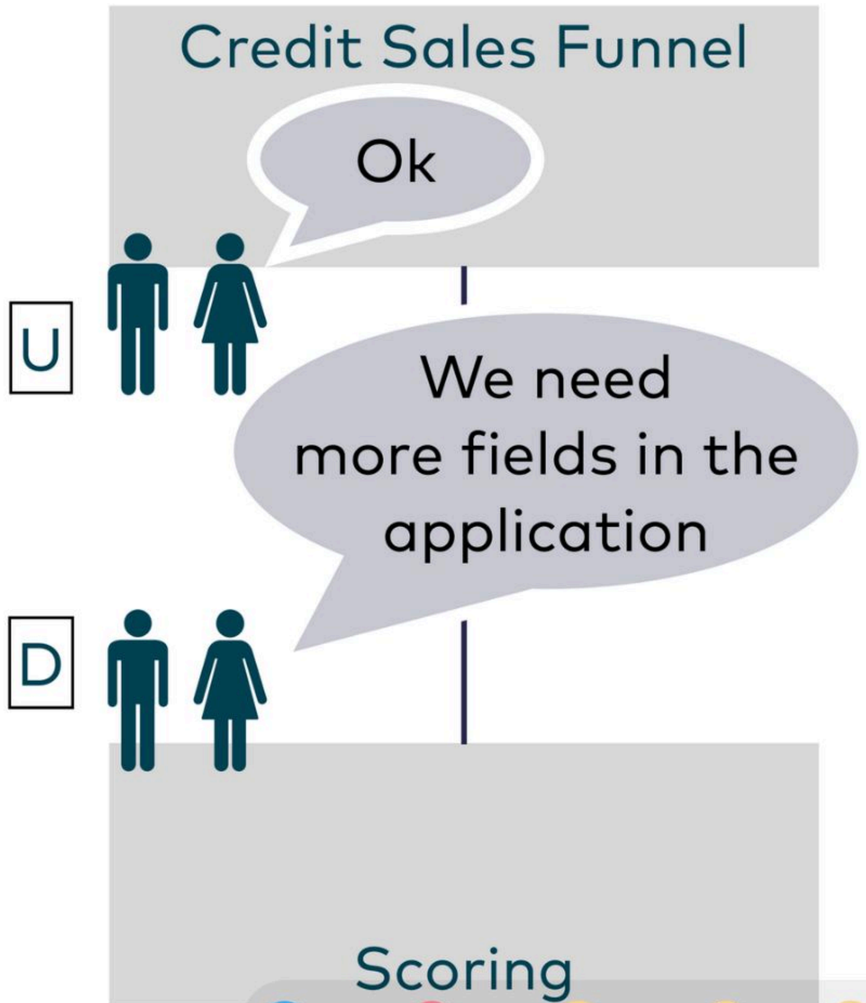
- Downstream requirements factor into upstream planning. Therefore, the downstream team gains some influence over the priorities and tasks of the upstream team
- Customer-Supplier is organizational
- Mind „vetoing customer“ and customer against an OHS as anti-patterns



Customer-Supplier

A Customer-Supplier development gives the downstream team some influence

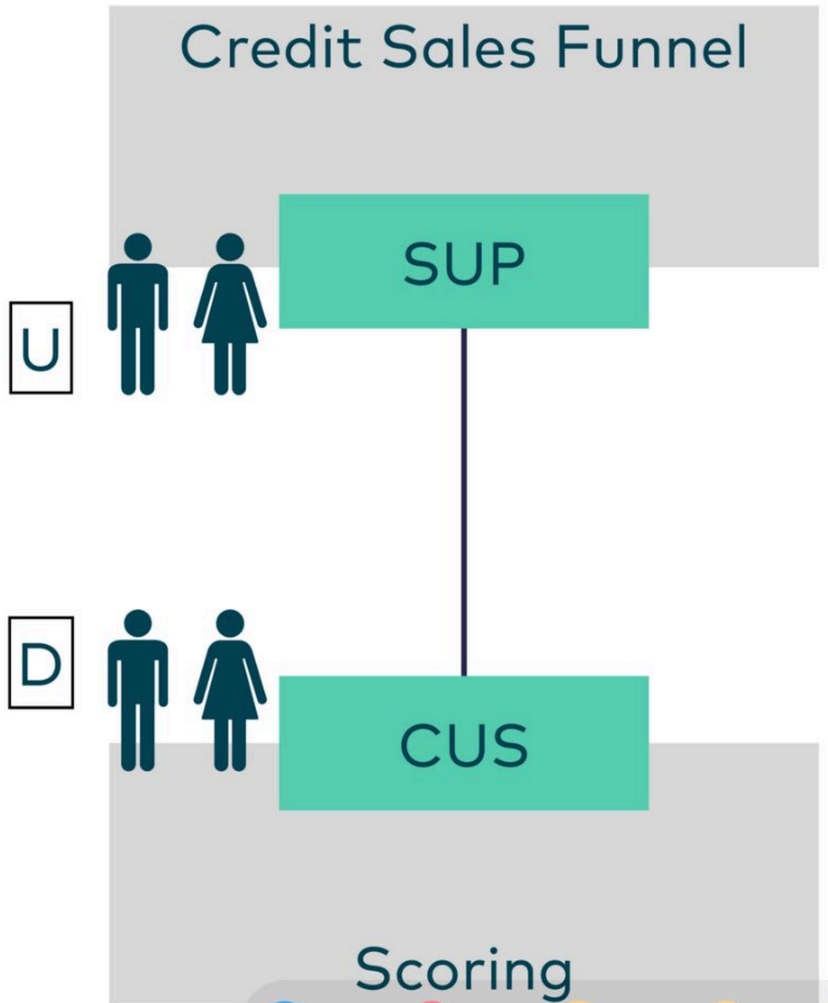
- Downstream requirements factor into upstream planning. Therefore, the downstream team gains some influence over the priorities and tasks of the upstream team
- Customer-Supplier is organizational
- Mind „vetoing customer“ and customer against an OHS as anti-patterns



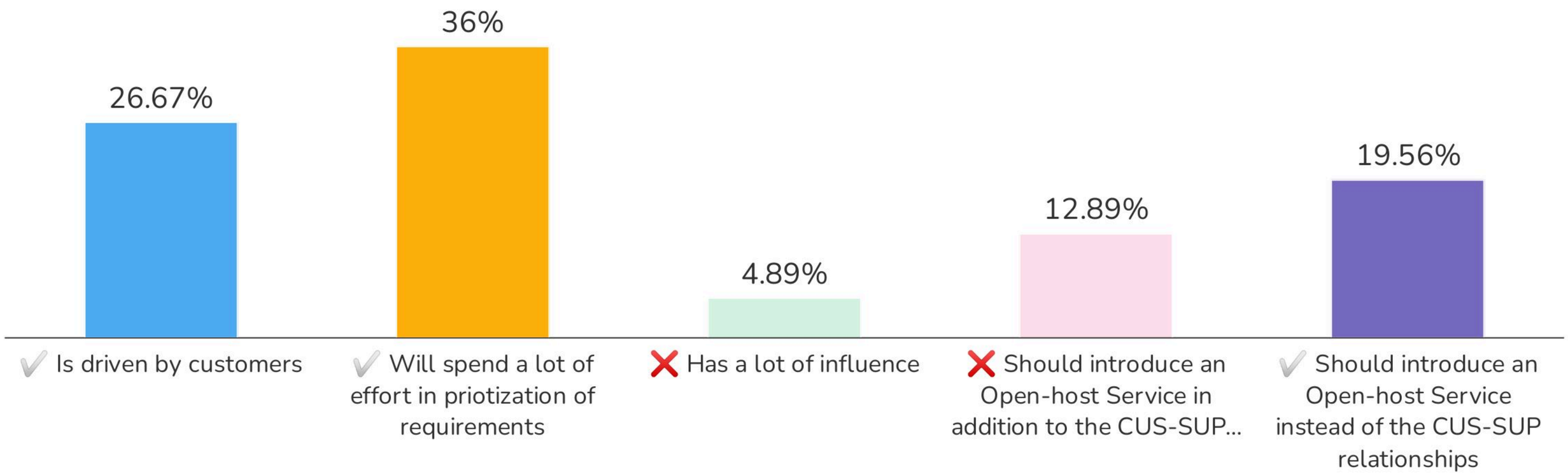
Customer-Supplier

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



Which statements are valid for the supplier?



Separate Ways

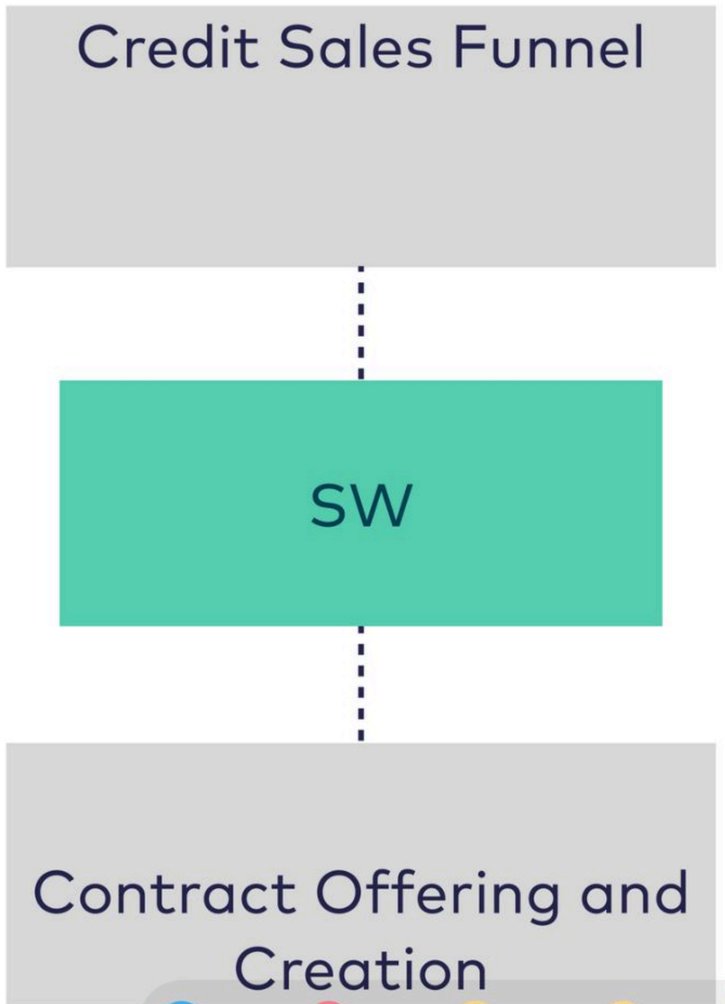
A bounded context has no connections to others

- 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

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

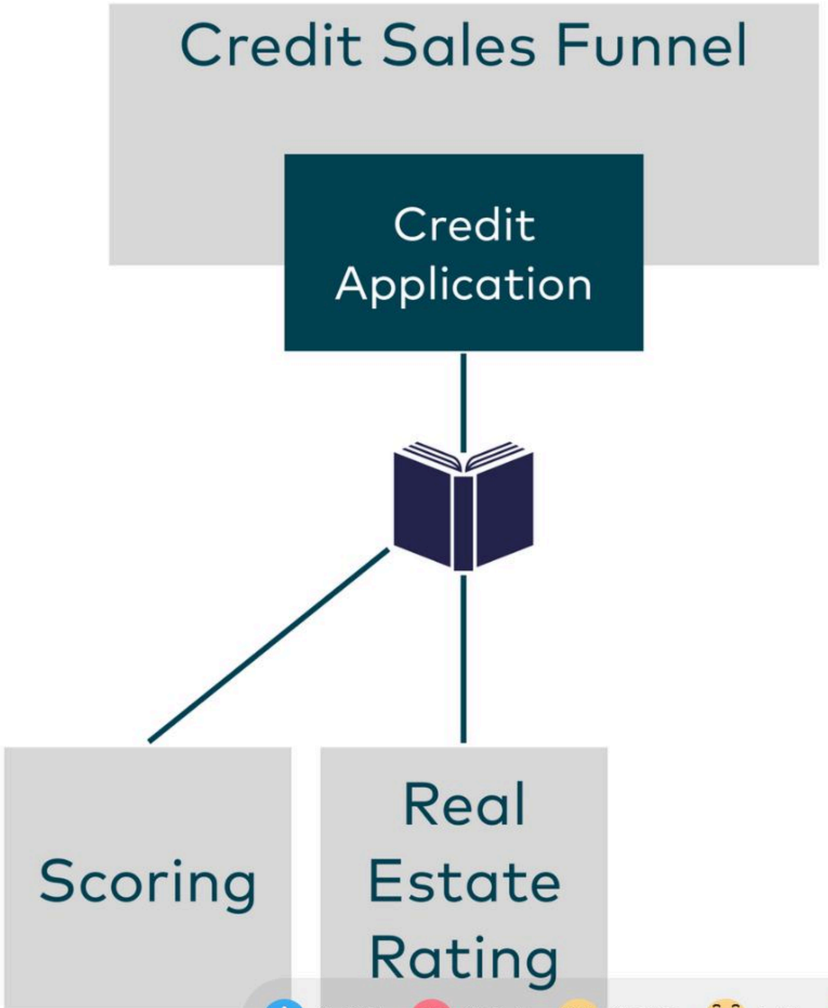
A well documented language shared between bounded contexts

- 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

Published Language

A well documented language shared between bounded contexts

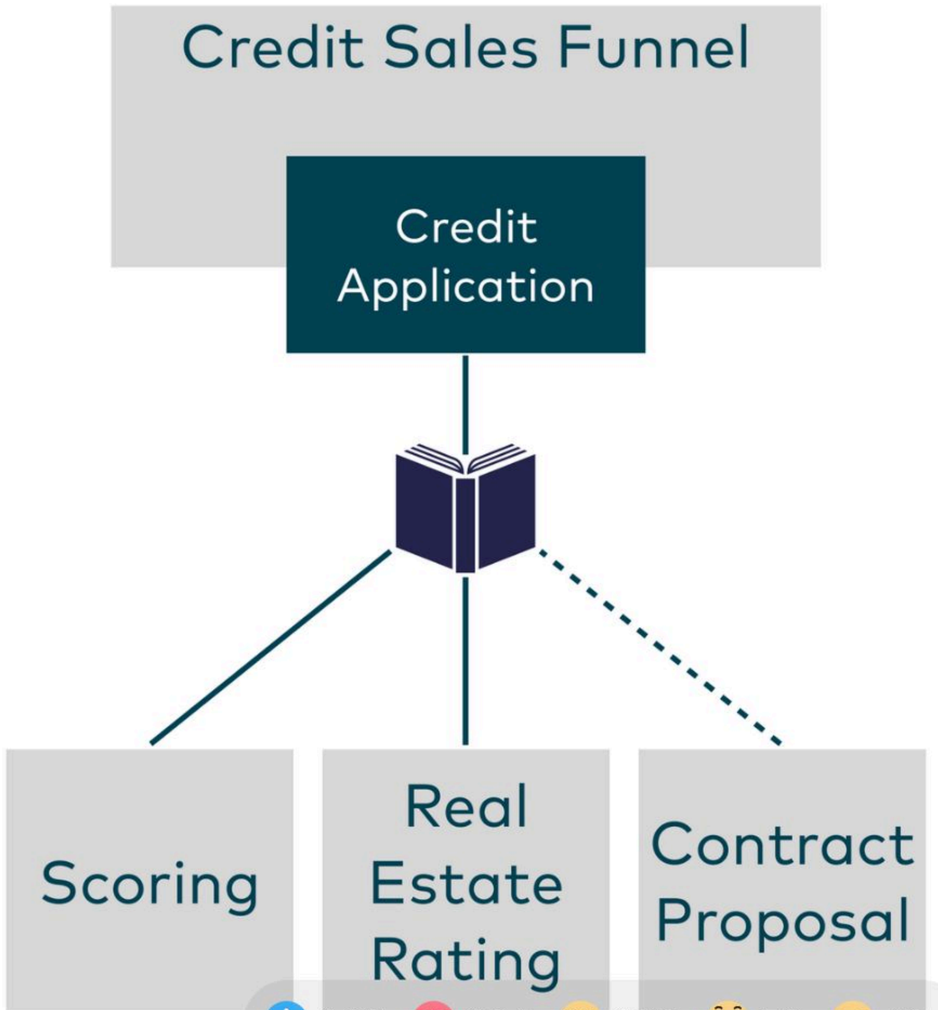
- Every bounded context can translate in and out from that language
- Sometimes defined by a consortium of the most important stakeholders / teams
- Often combined with Open-host Service
- Examples: iCalendar, vCard, ZugFerd



Published Language

A well documented language shared between bounded contexts

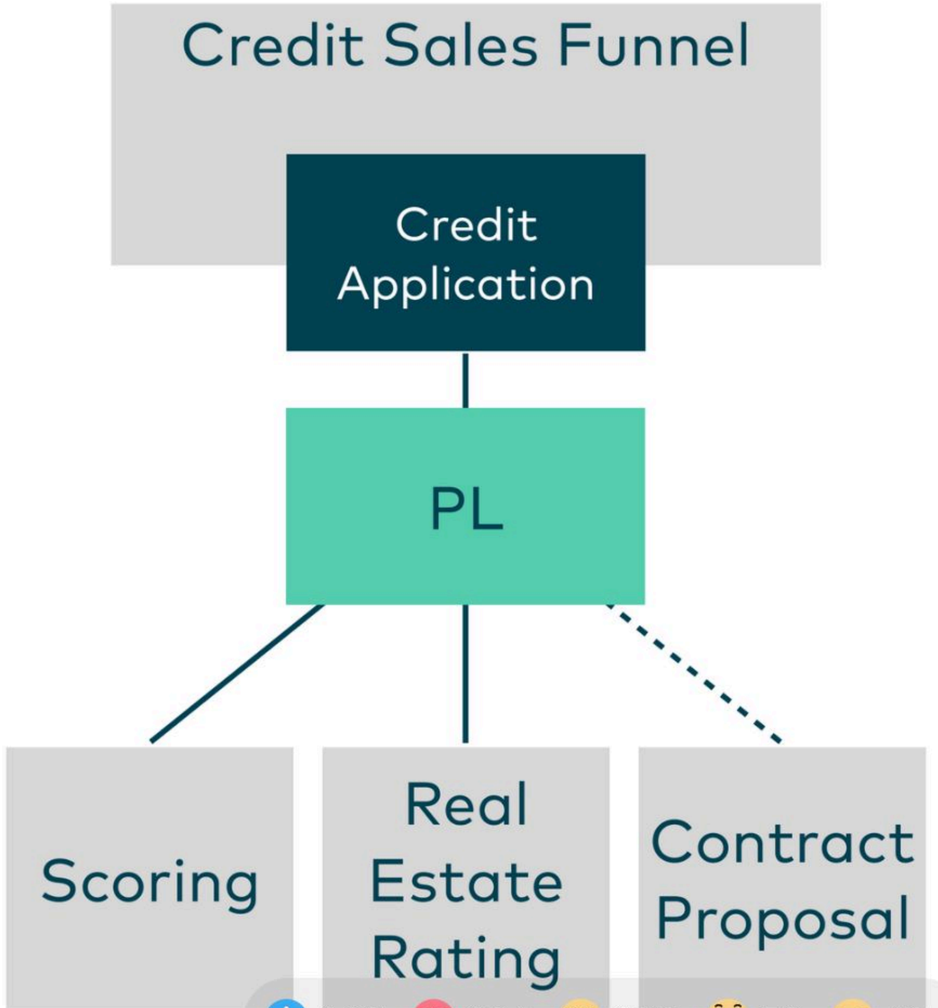
- Every bounded context can translate in and out from that language
- Sometimes defined by a consortium of the most important stakeholders / teams
- Often combined with Open-host Service
- Examples: iCalendar, vCard, ZugFerd



Published Language

A well documented language shared between bounded contexts

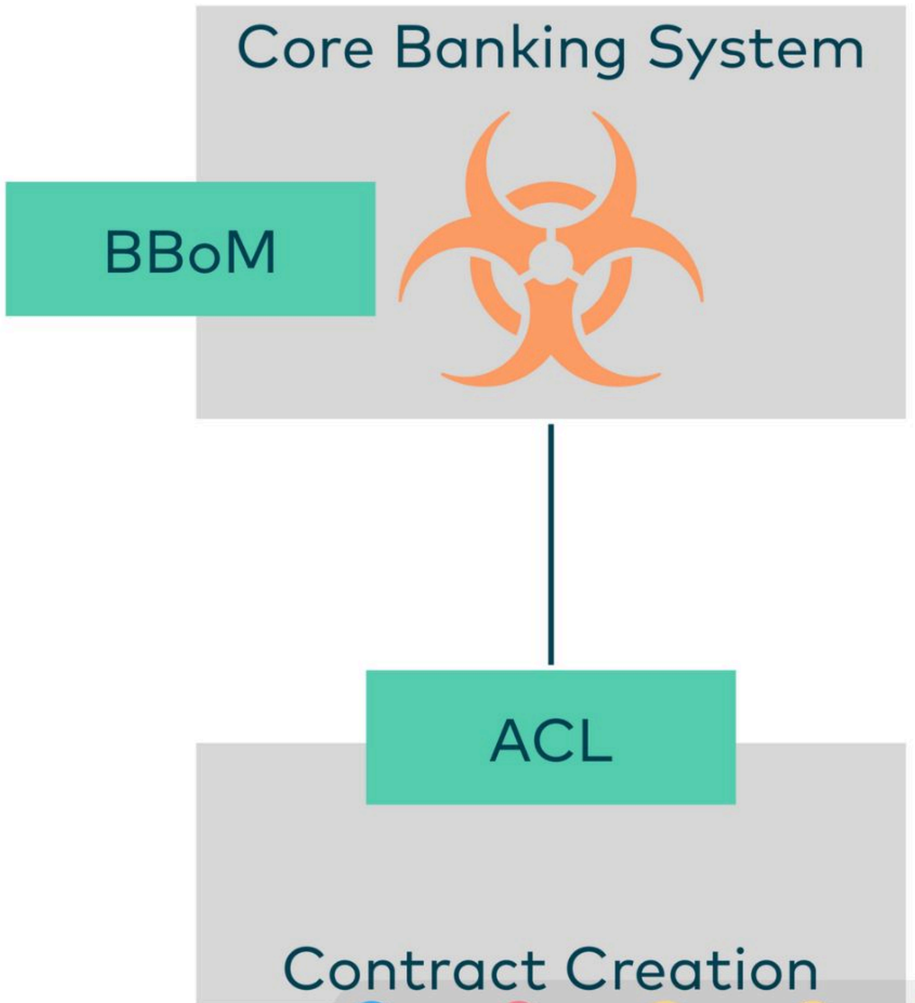
- Every bounded context can translate in and out from that language
- Sometimes defined by a consortium of the most important stakeholders / teams
- Often combined with Open-host Service
- Examples: iCalendar, vCard, ZugFerd



Big Ball Of Mud

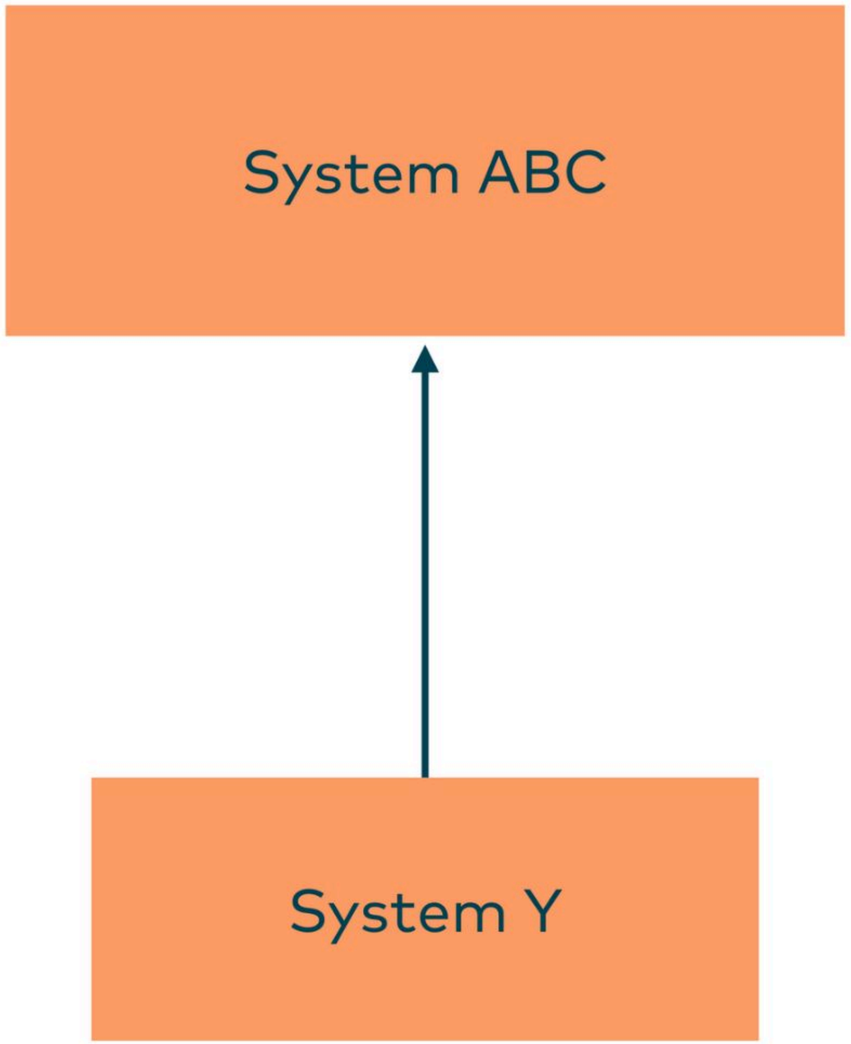
A (part of a) system which is a mess by having mixed models and inconsistent boundaries

- 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



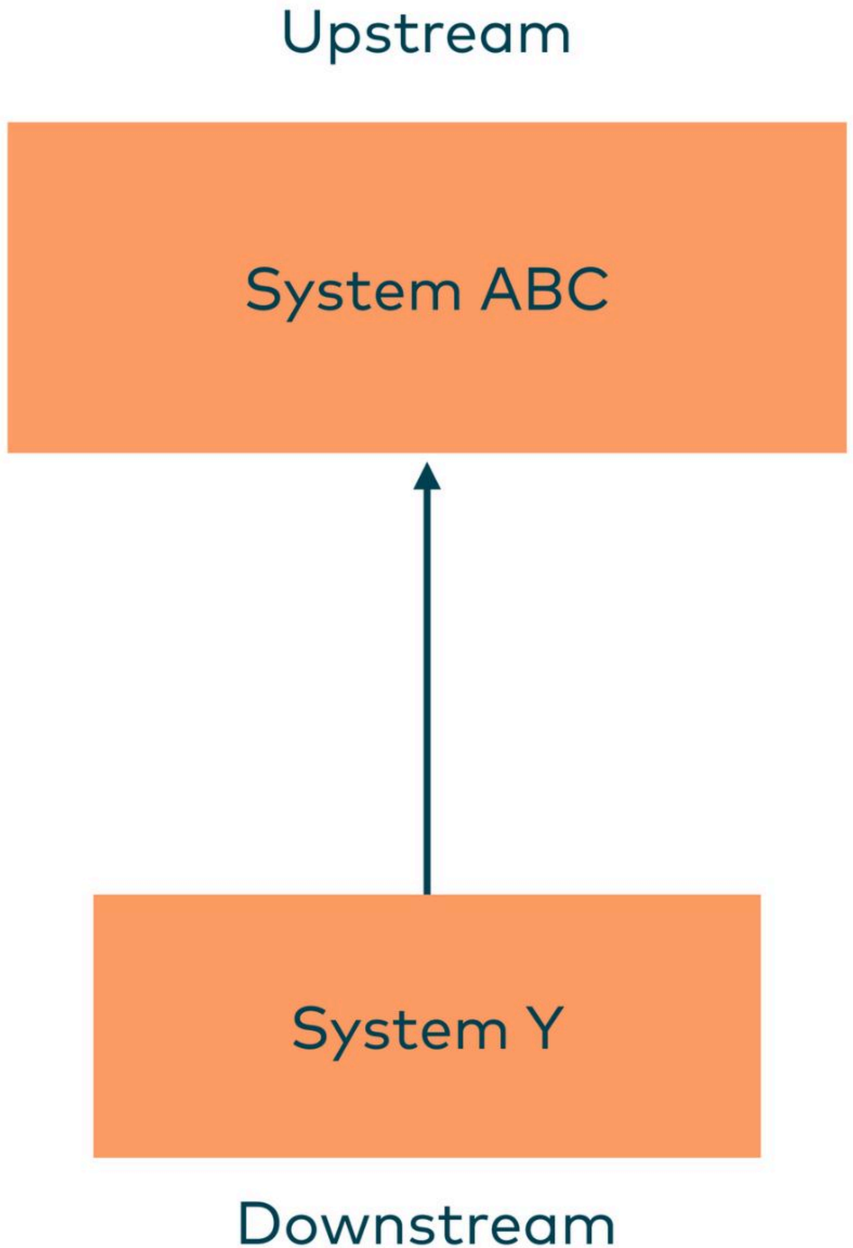
You can visualize different perspectives

- Call Relationship



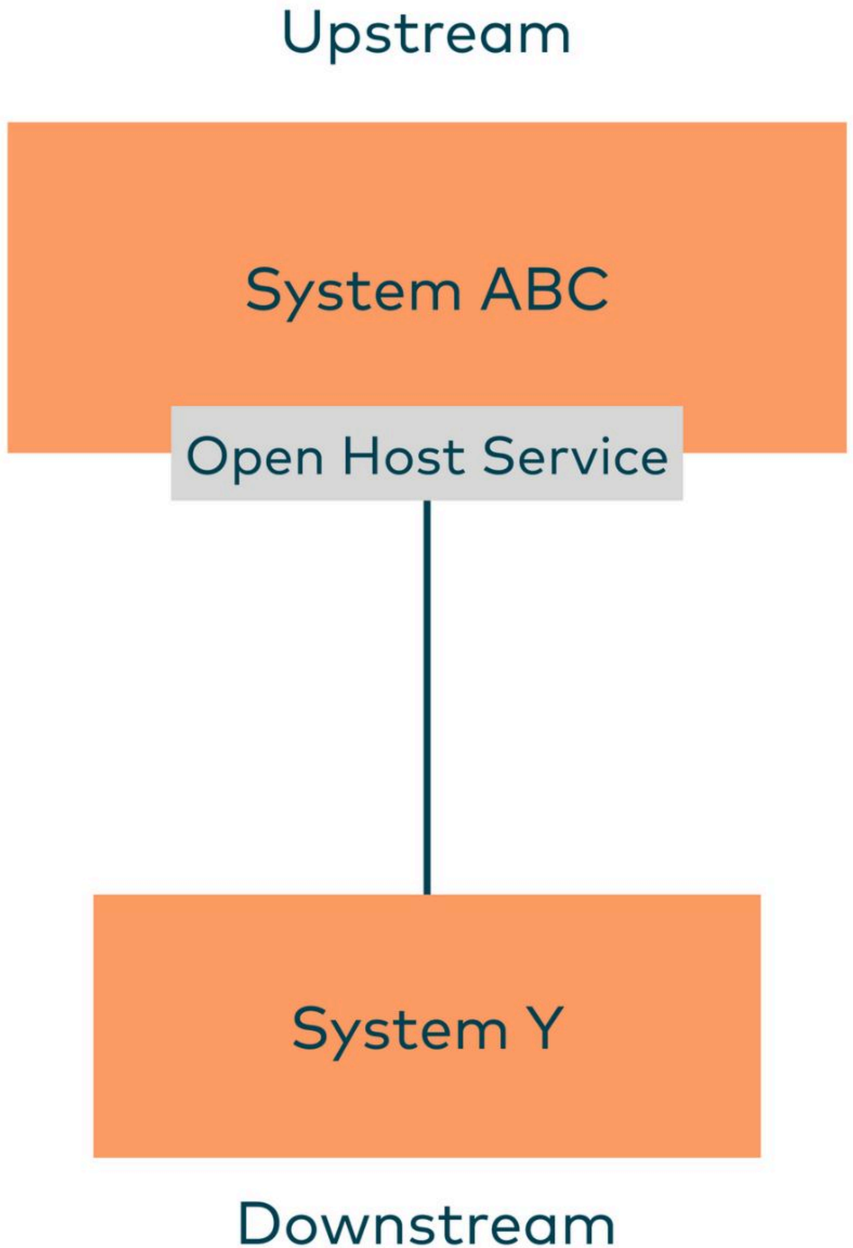
You can visualize different perspectives

- Call Relationship
- Team Relationship - Level 1



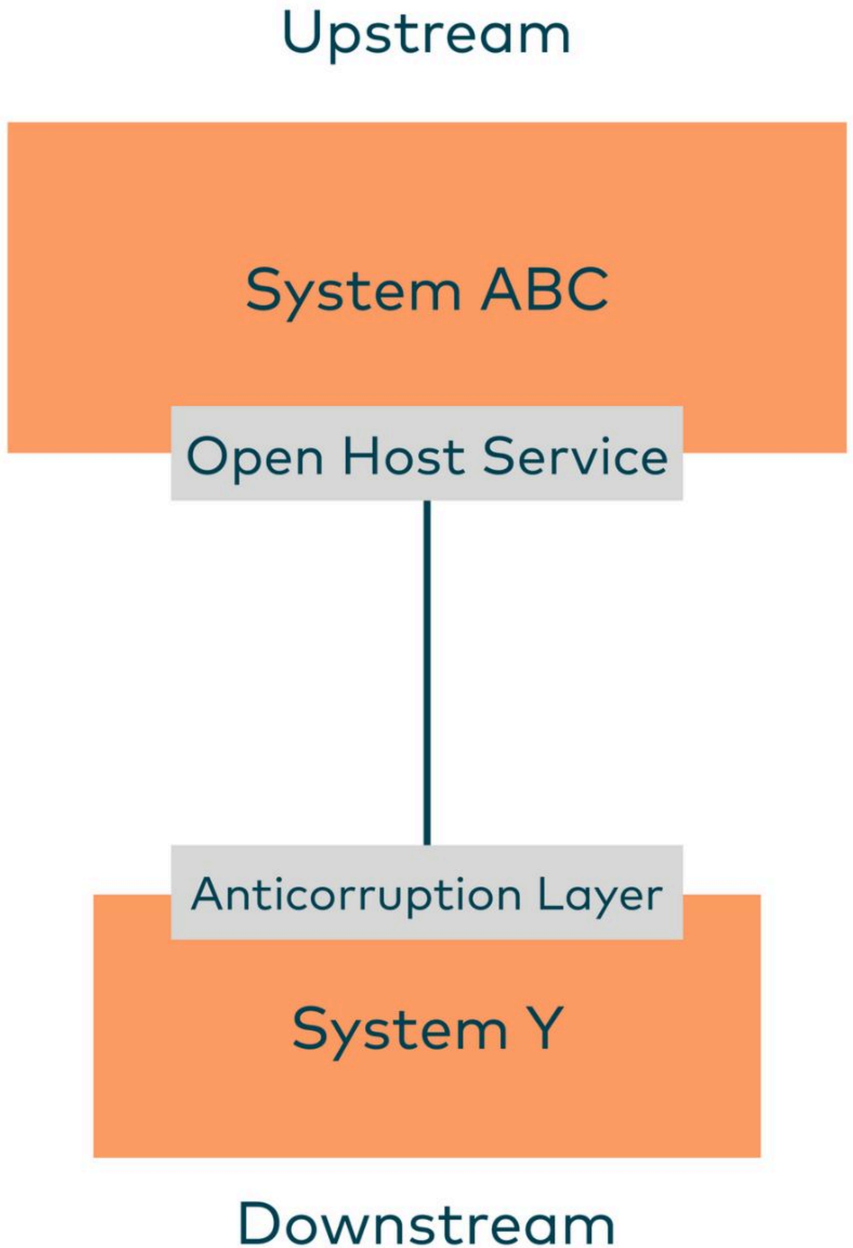
You can visualize different perspectives

- Call Relationship
- Team Relationship - Level 1
- API Level



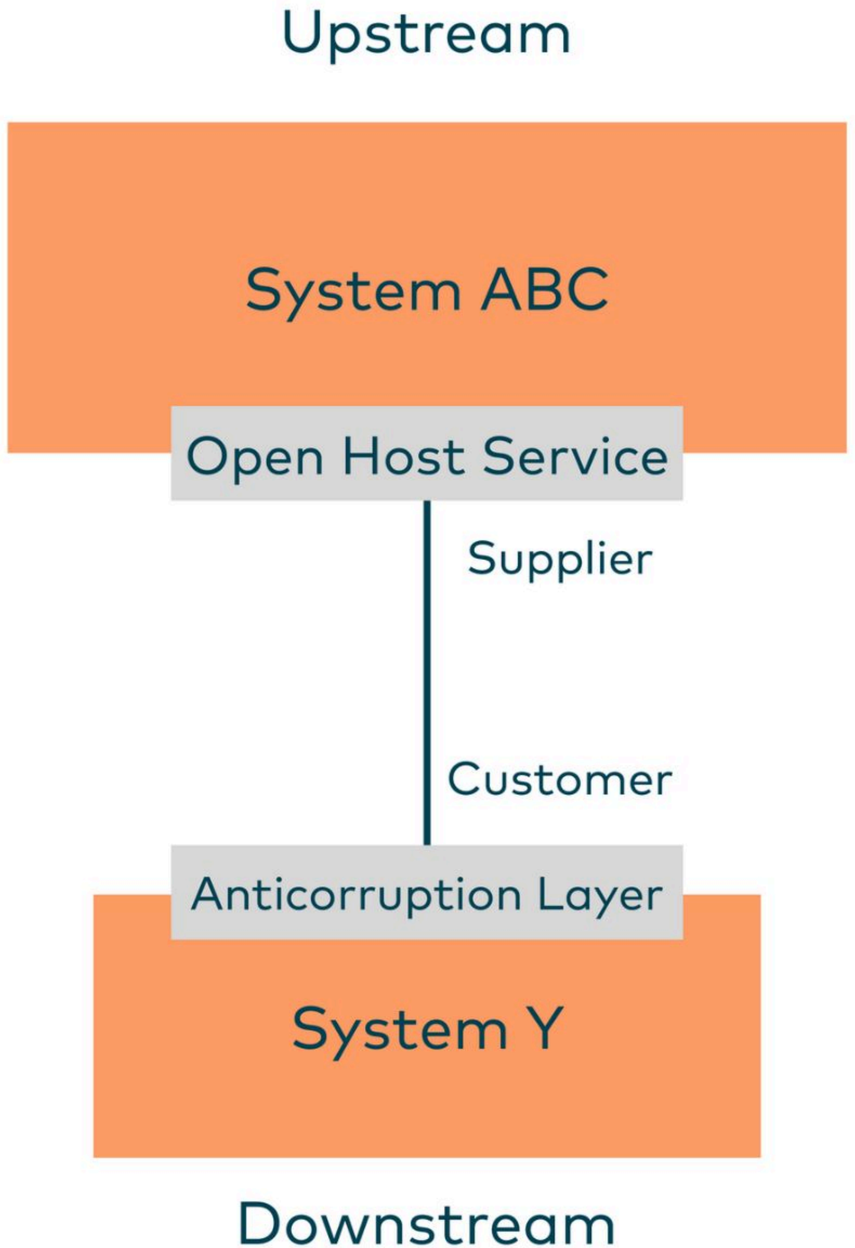
You can visualize different perspectives

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



You can visualize different perspectives

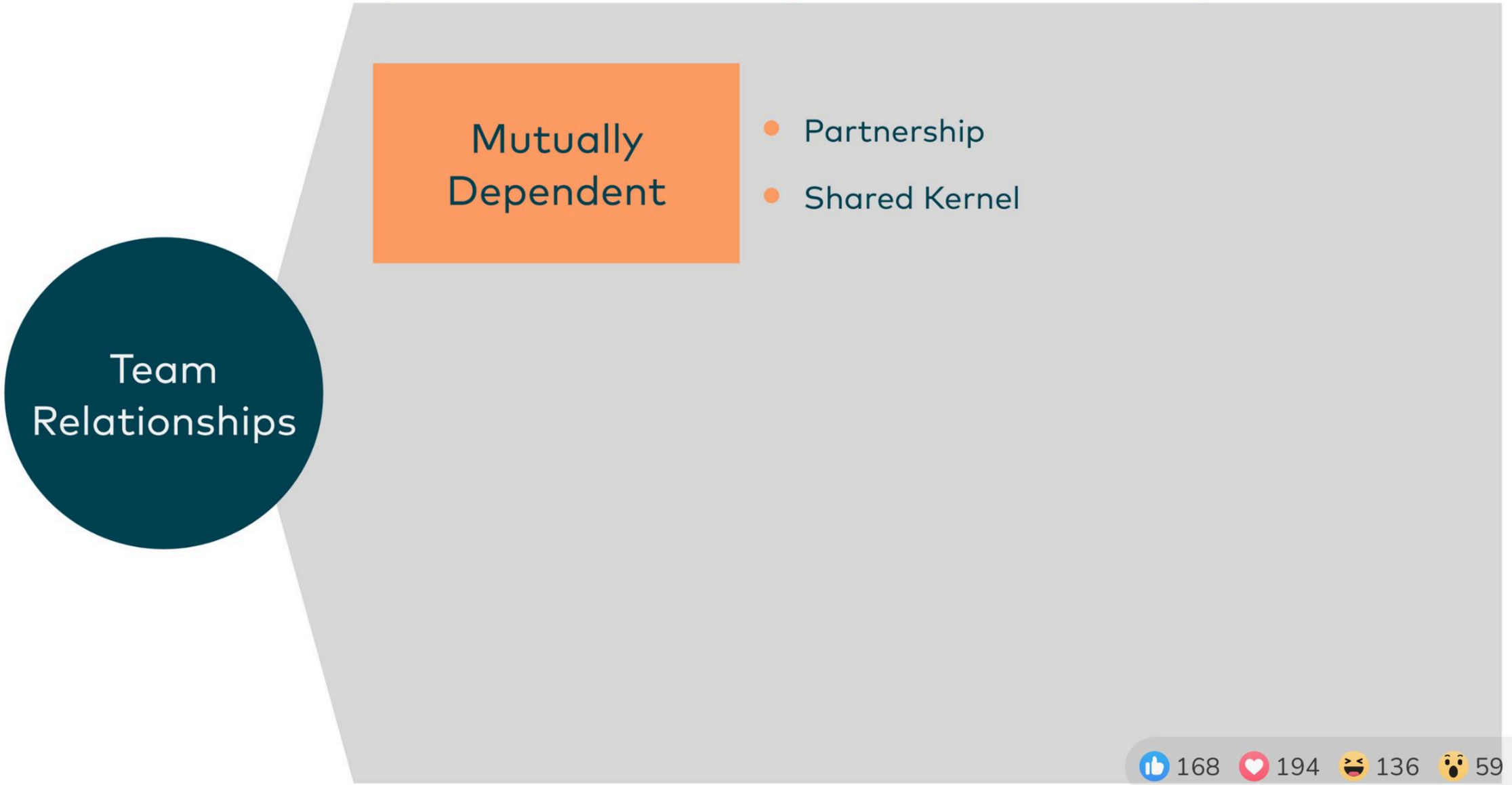
- Call Relationship
- Team Relationship - Level 1
- API Level
- Model Propagation
- Team Relationship - Level 2



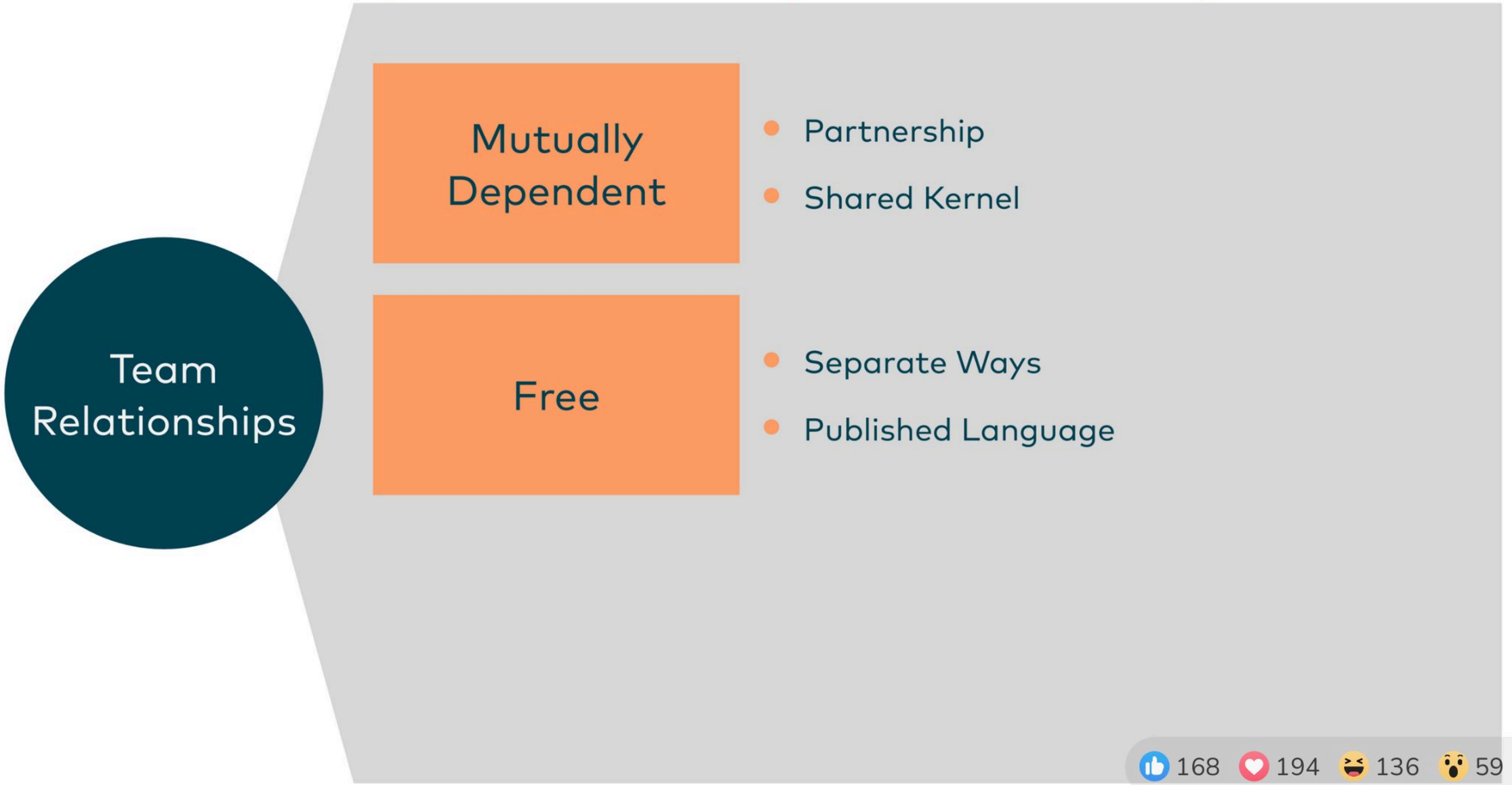
Patterns & perspectives

	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			✅

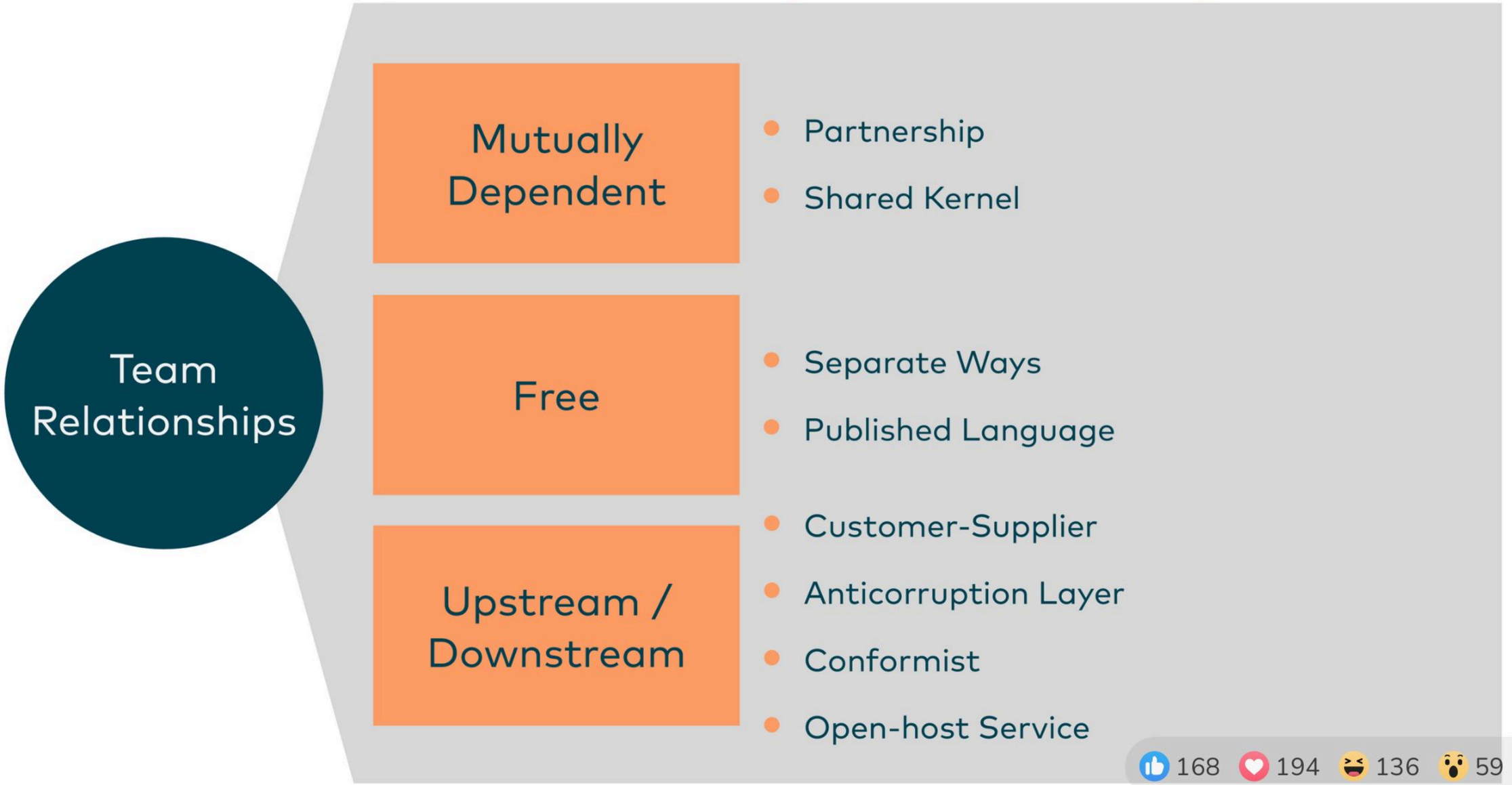
Some of the patterns map to team dependencies



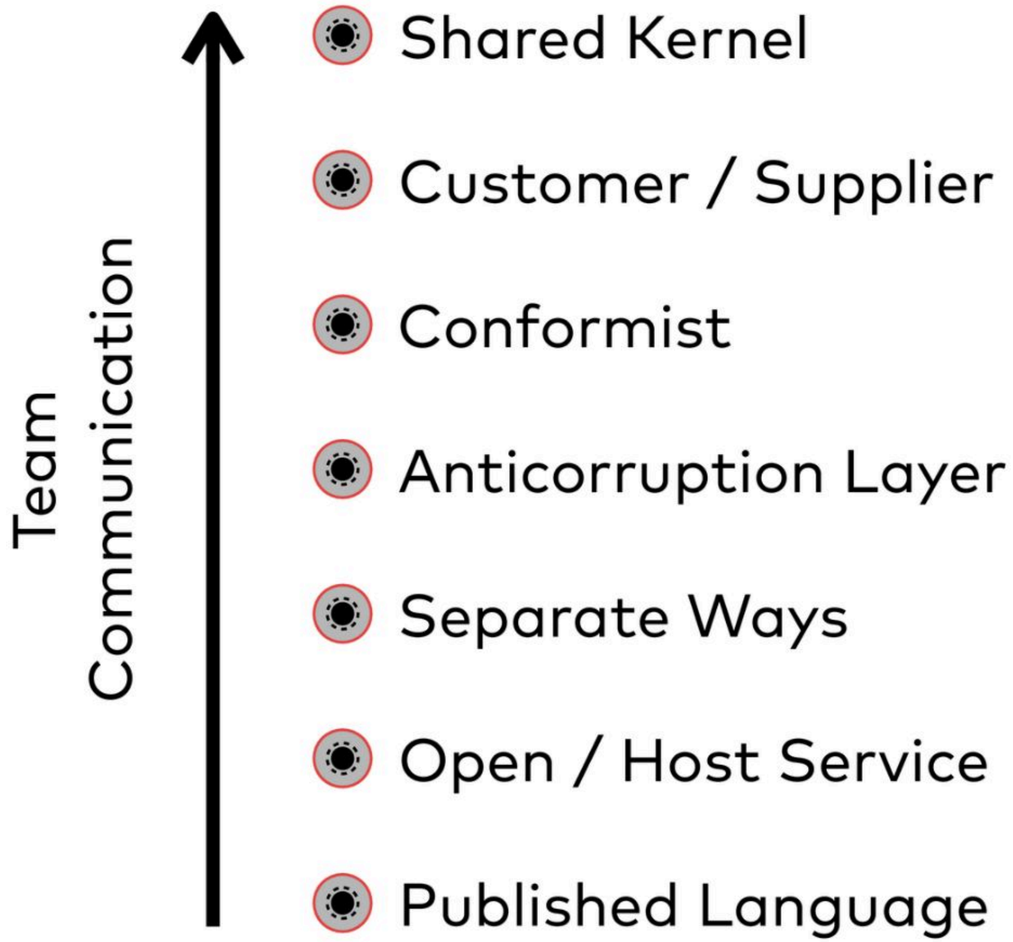
Some of the patterns map to team dependencies



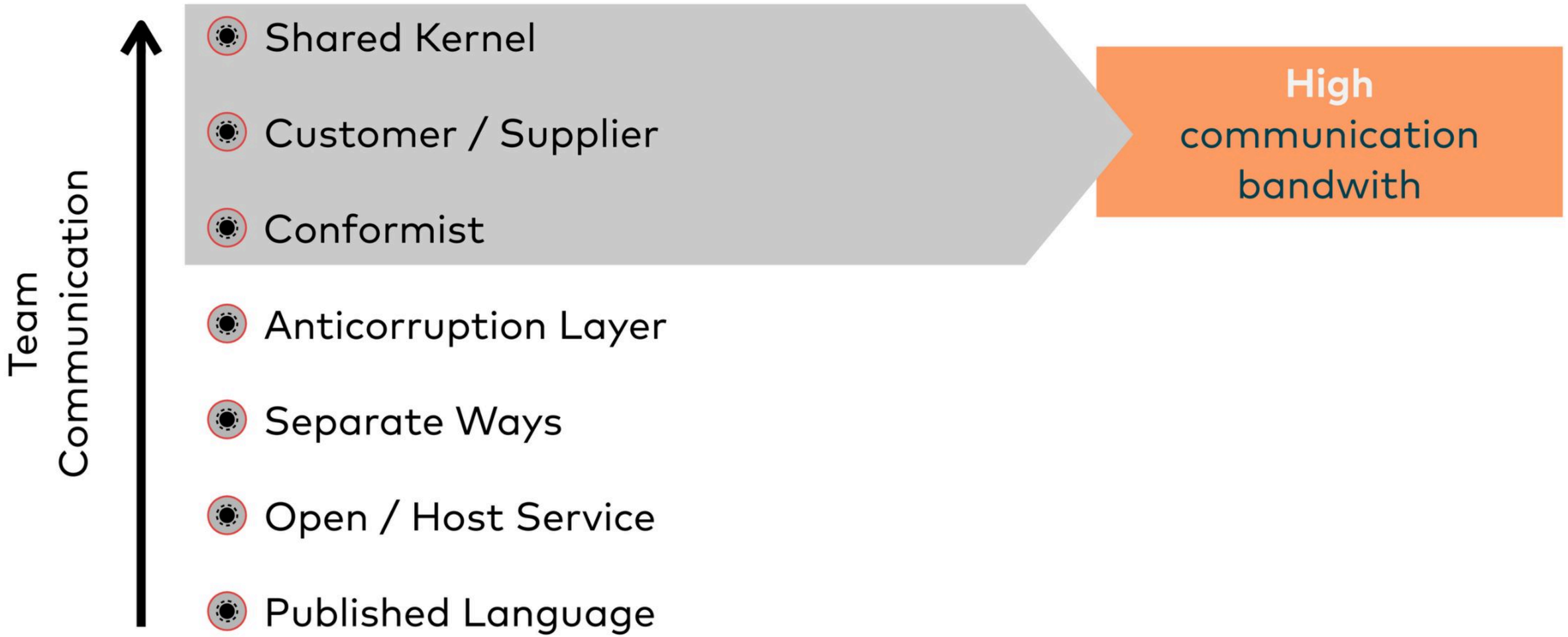
Some of the patterns map to team dependencies



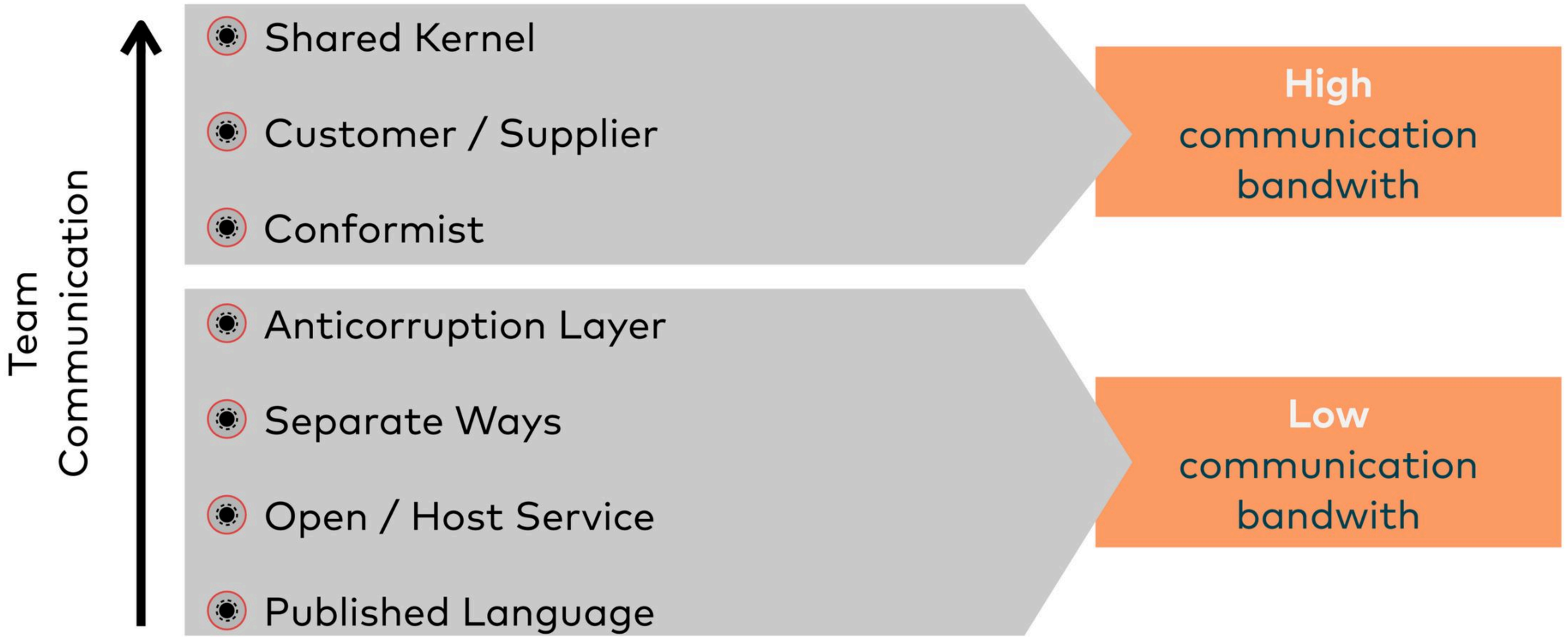
Mind team communication

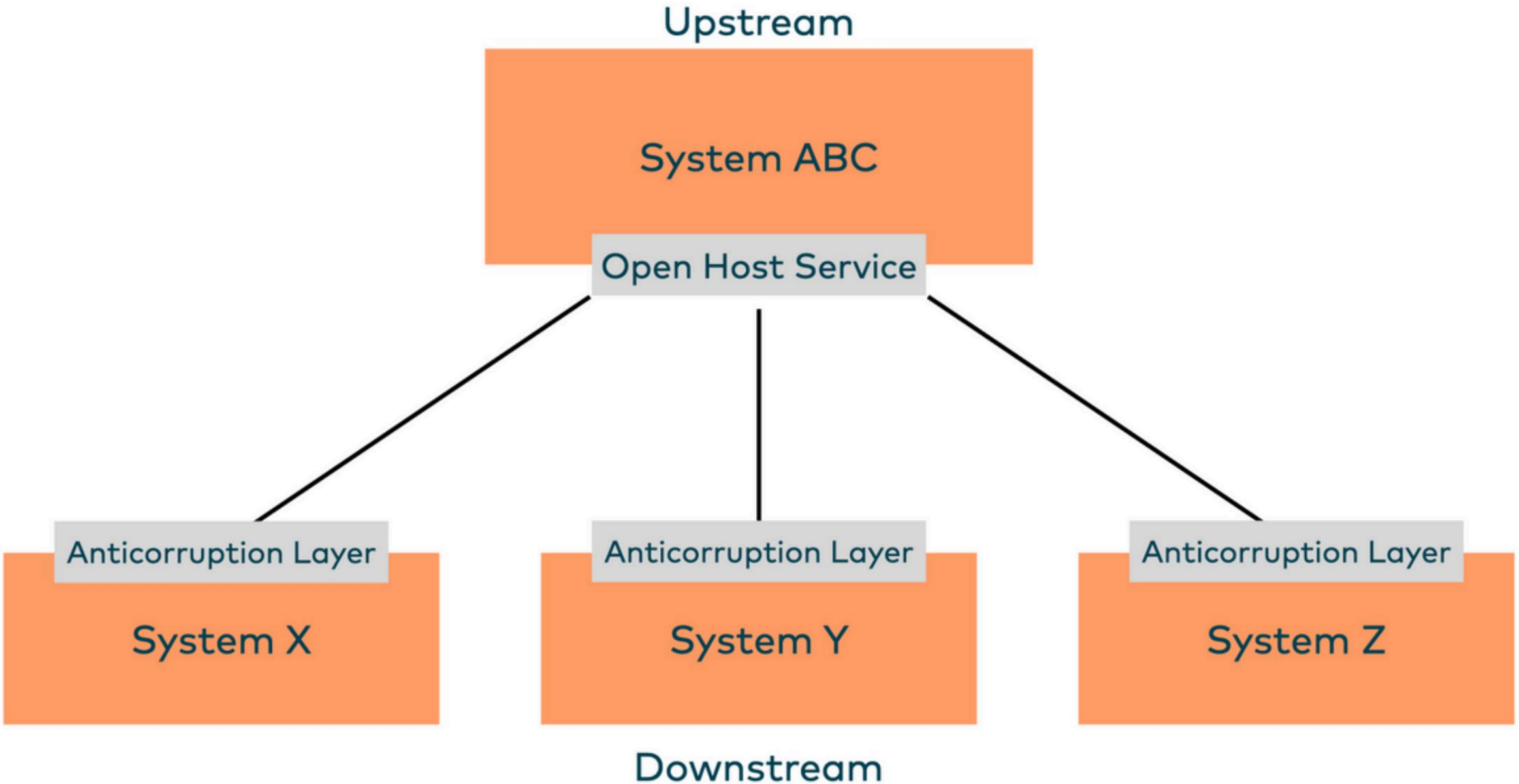


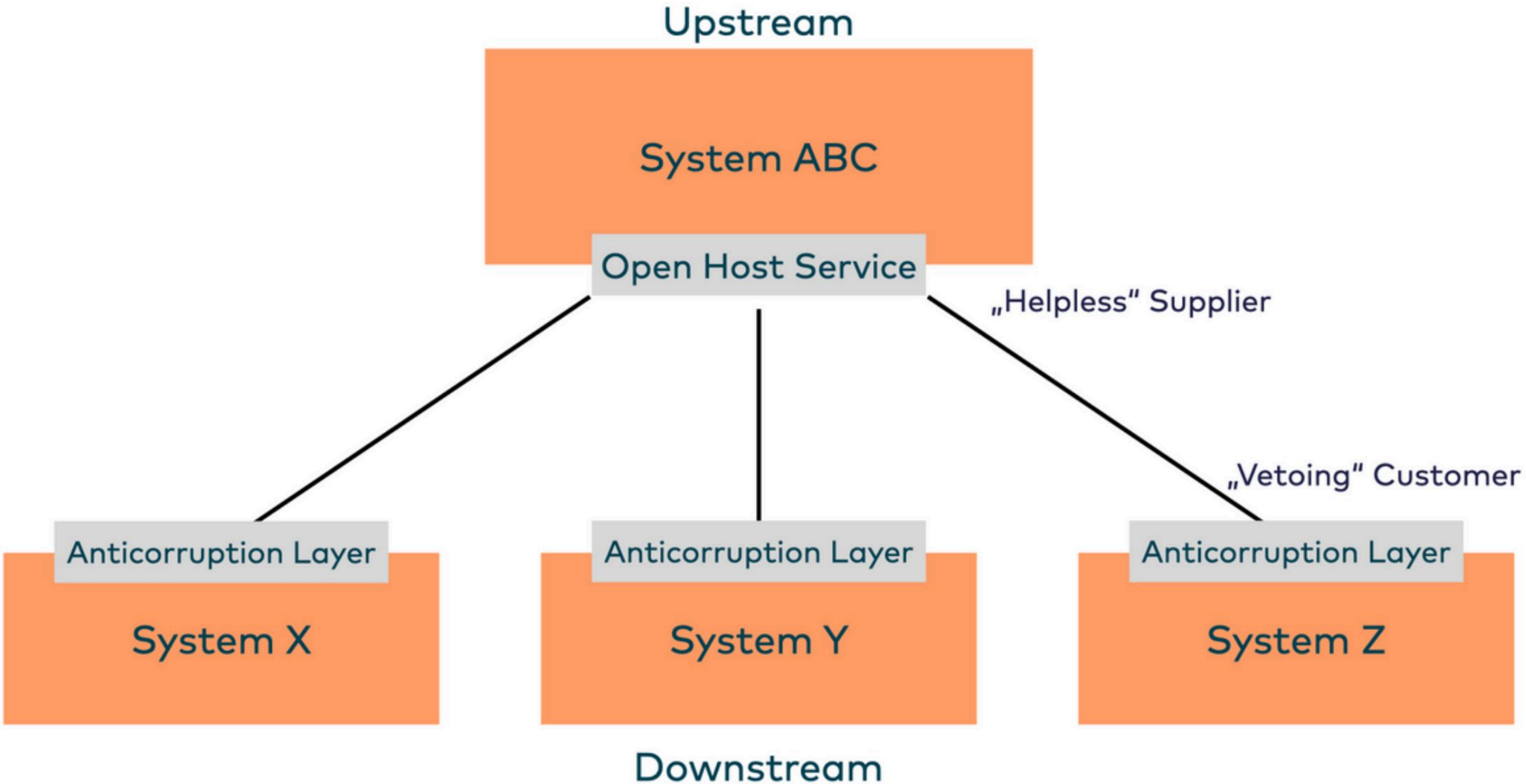
Mind team communication

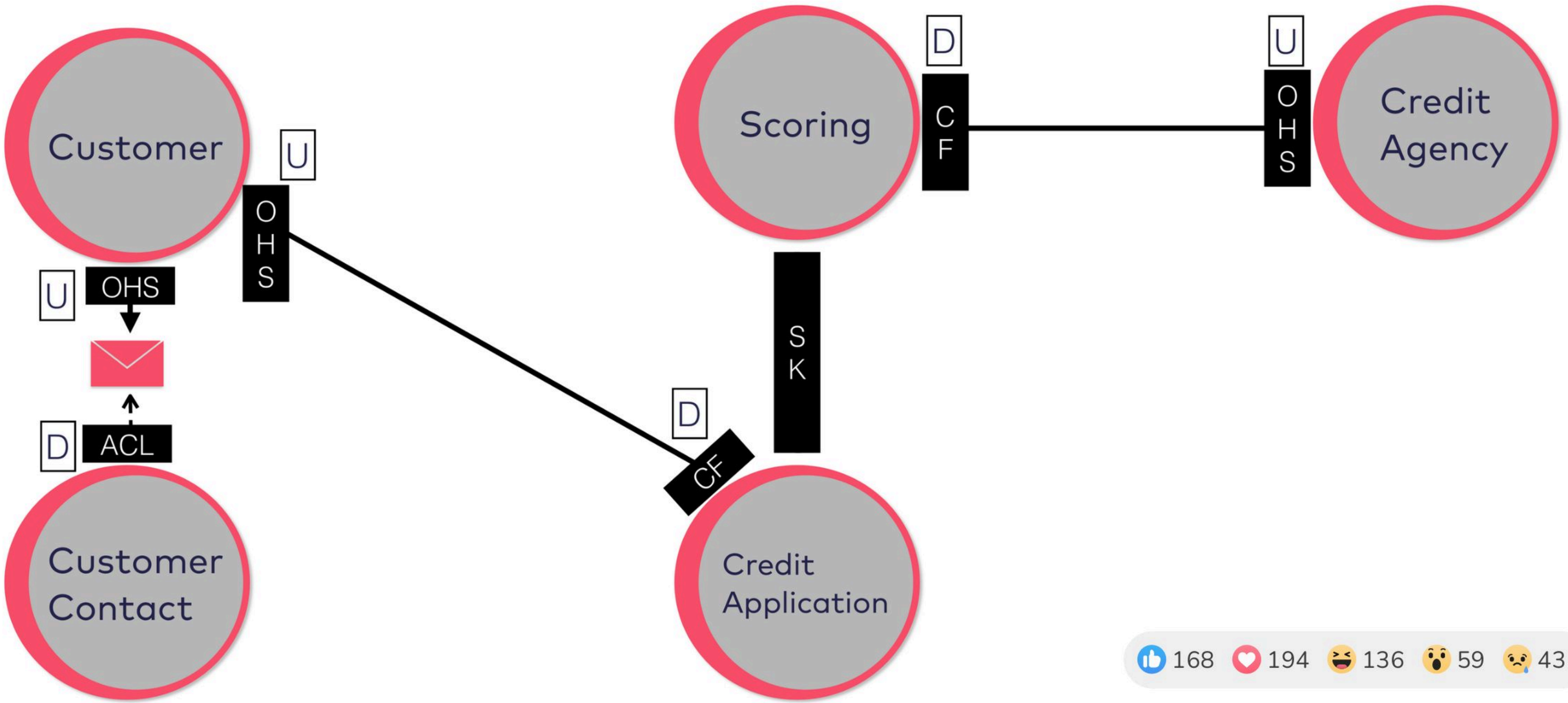


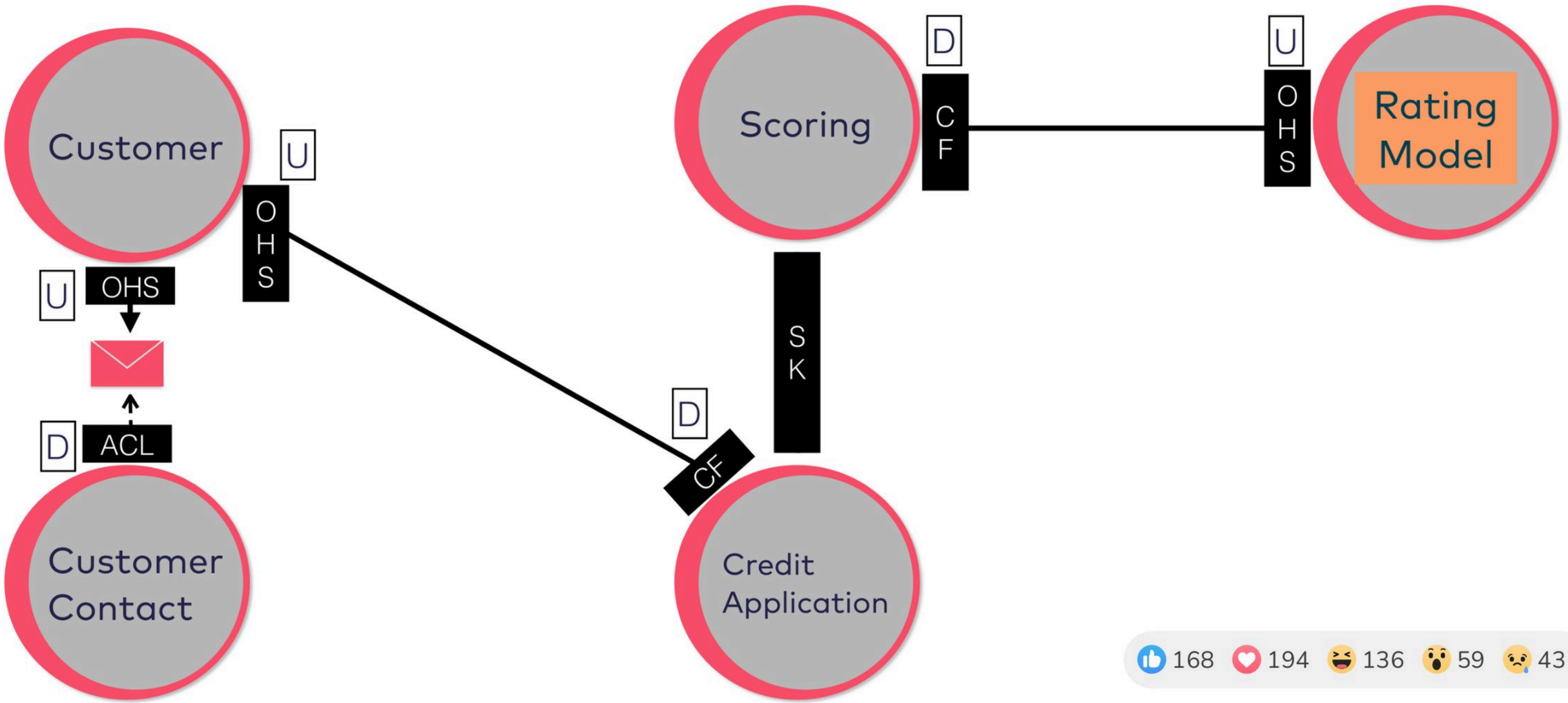
Mind team communication

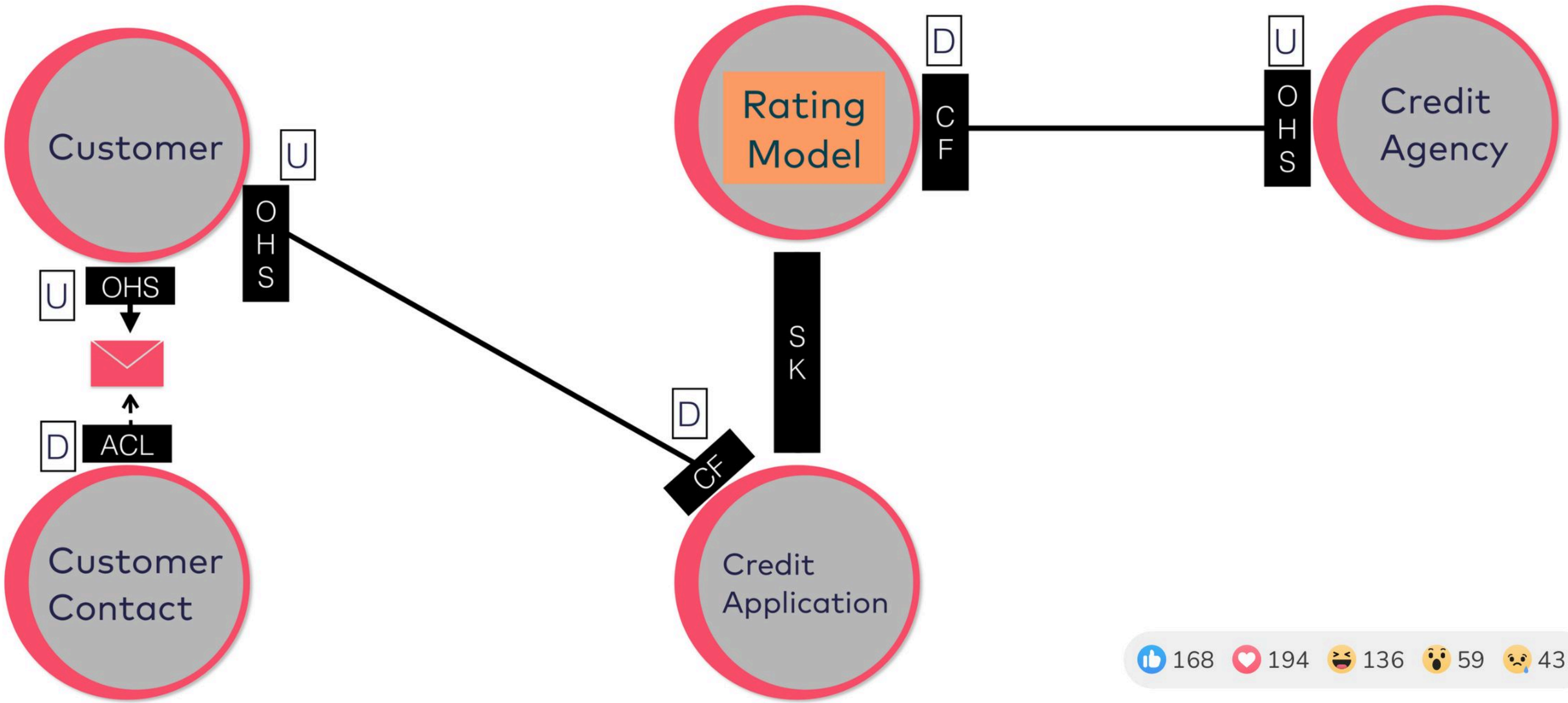


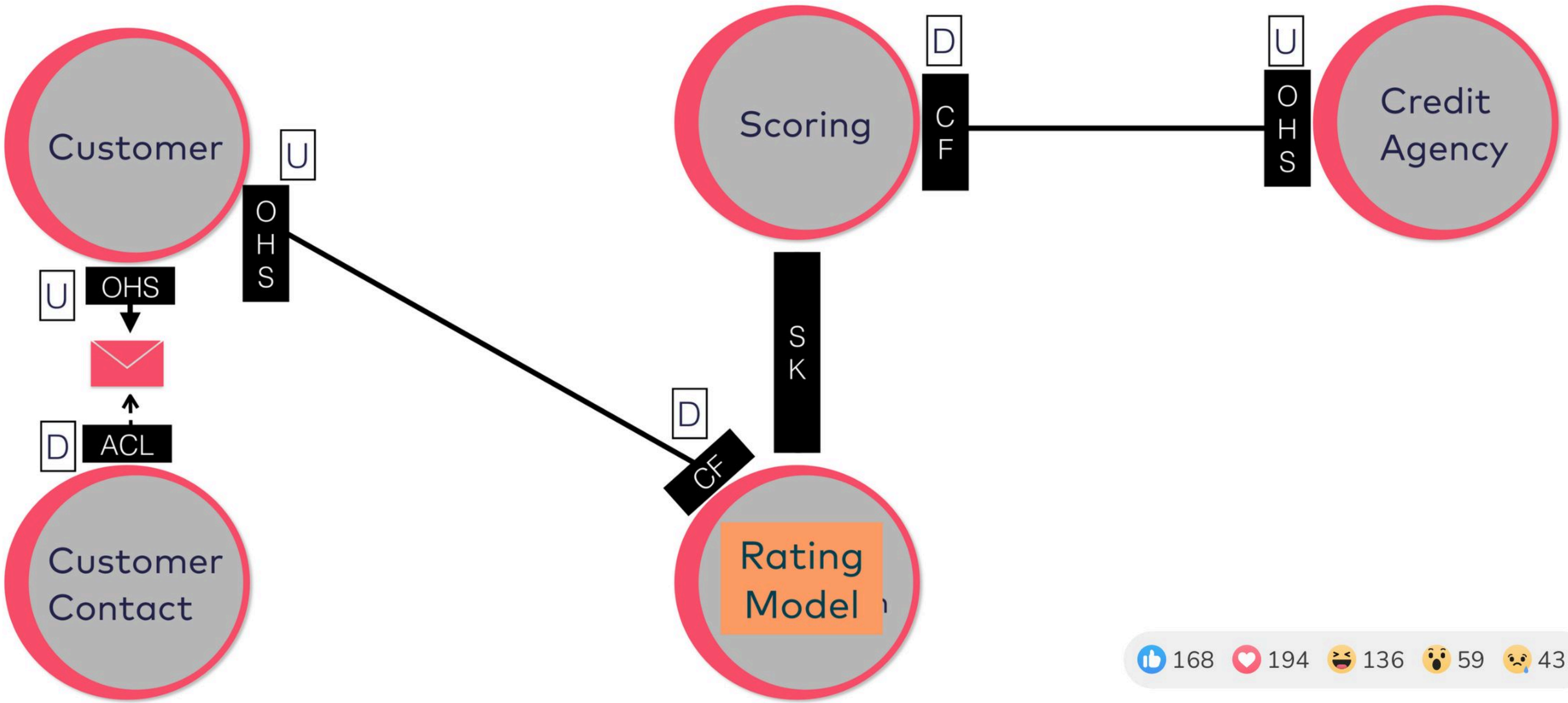


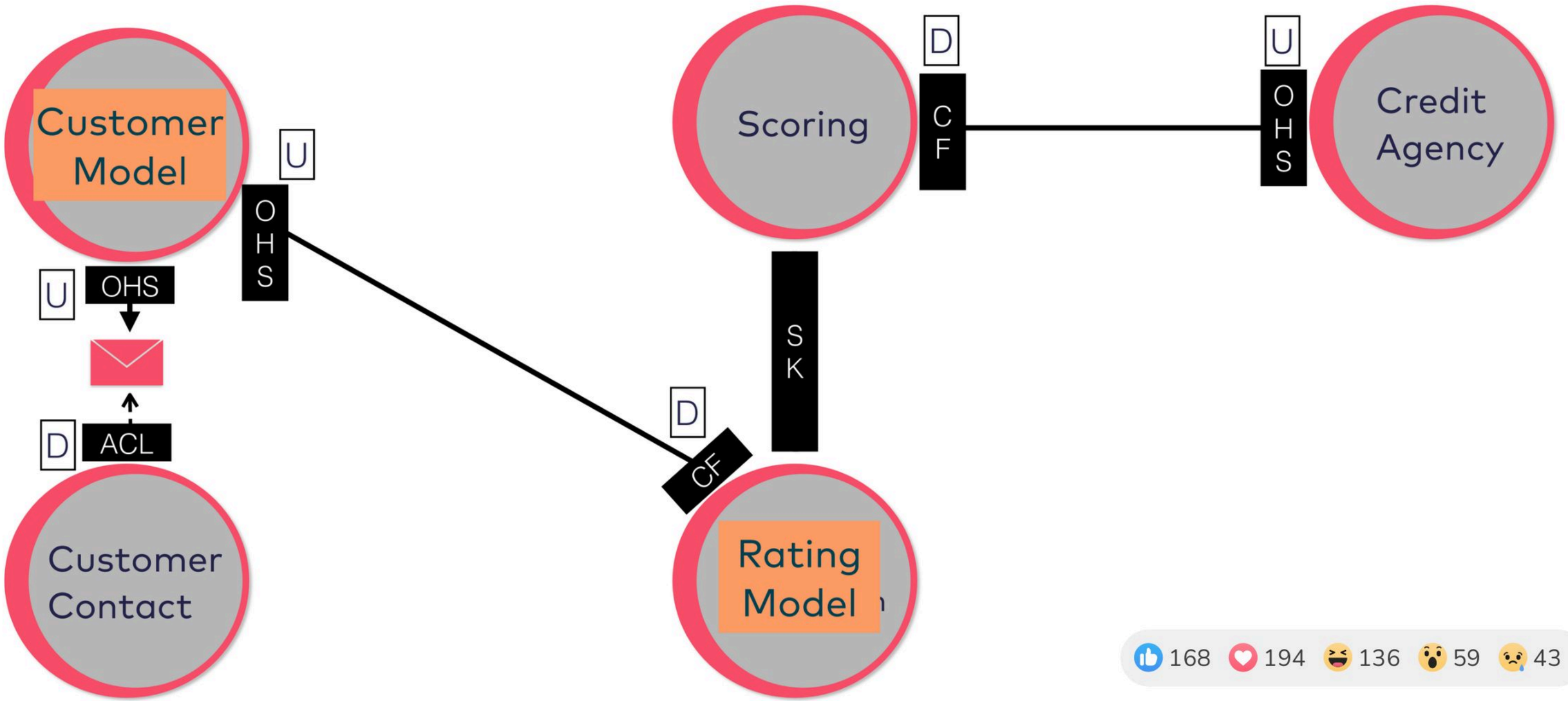


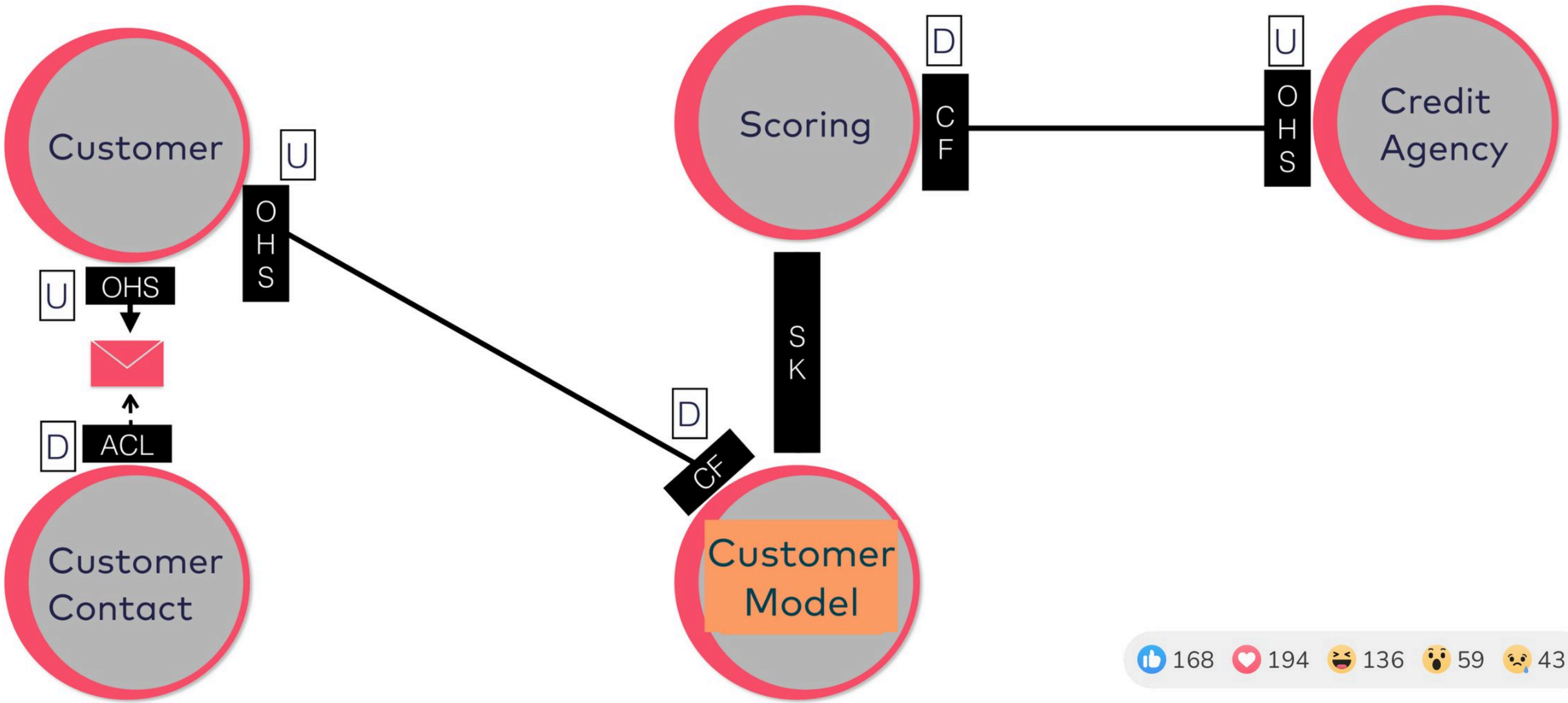


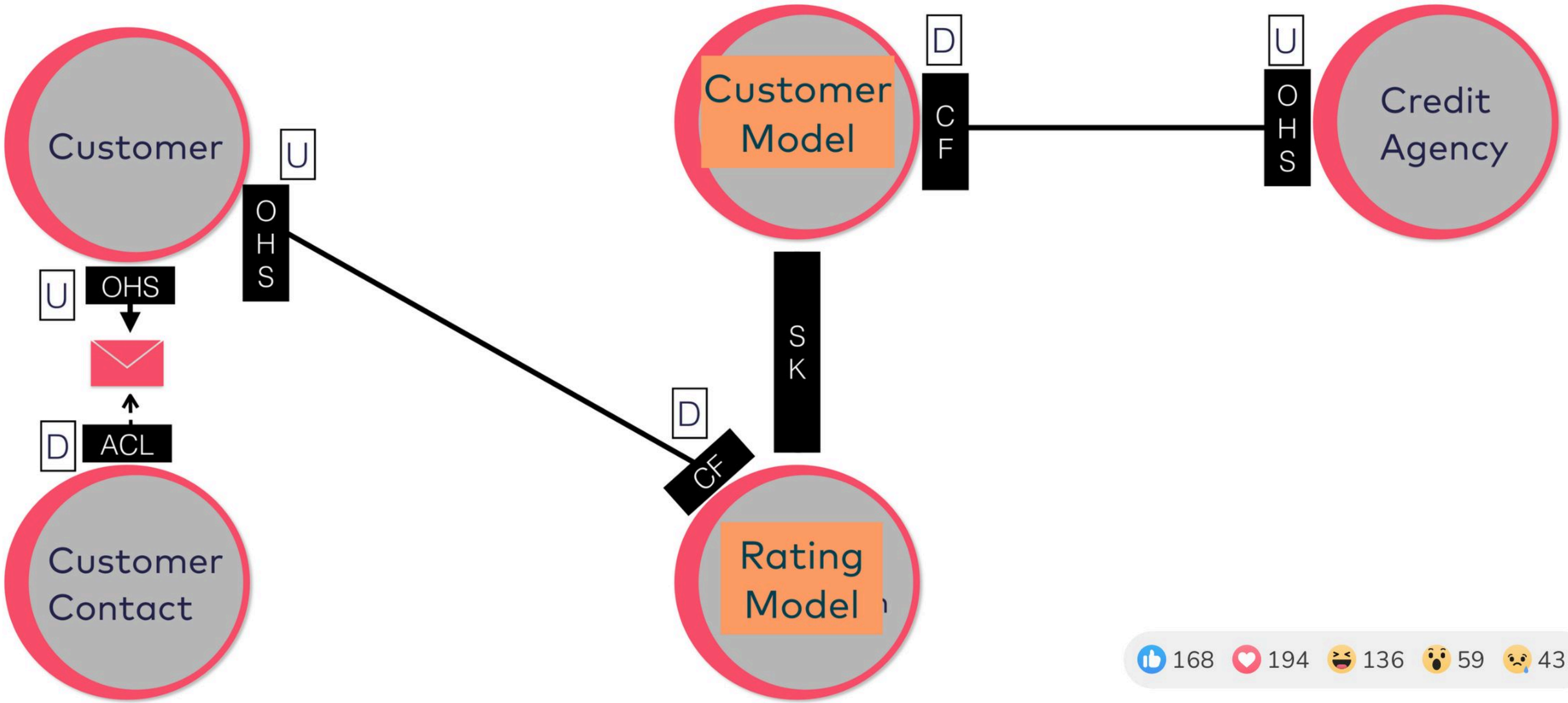




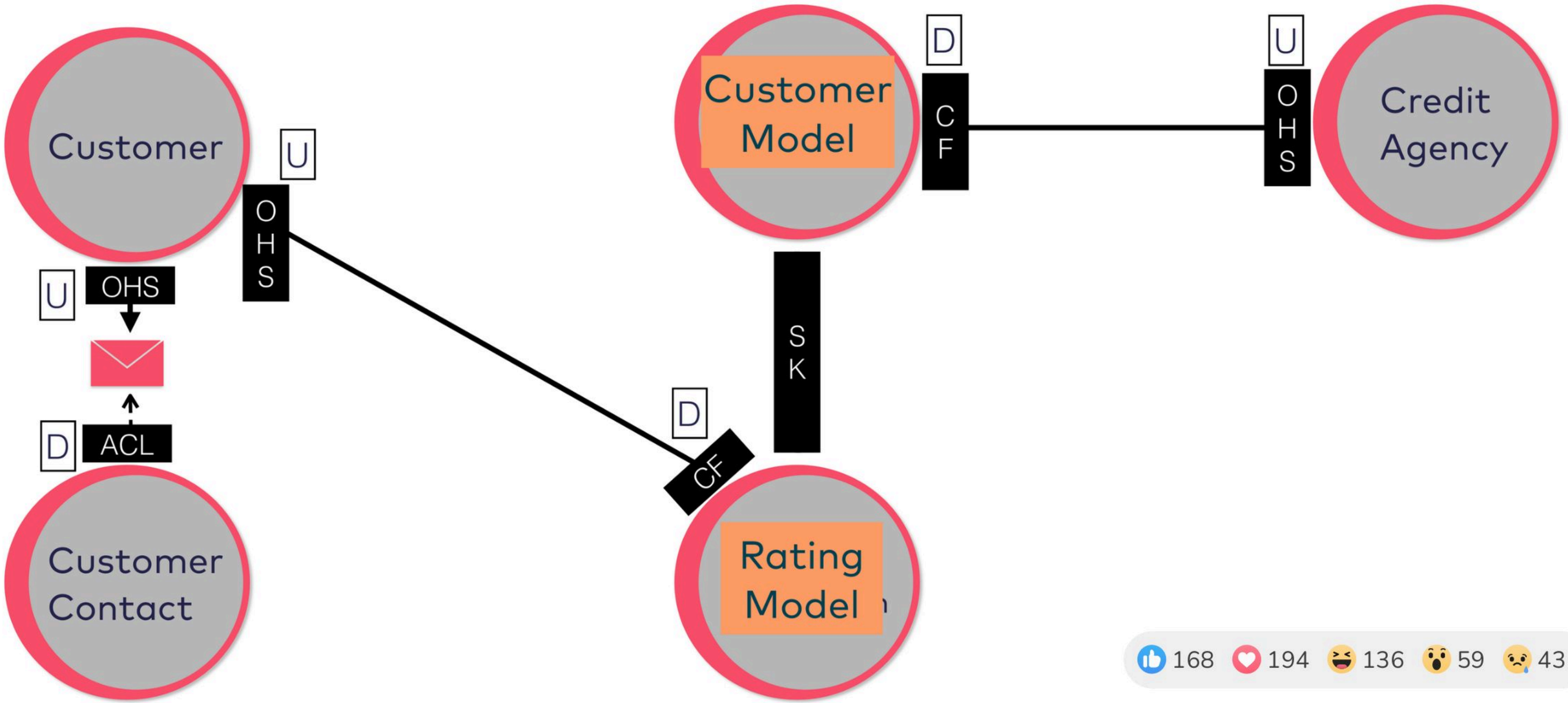




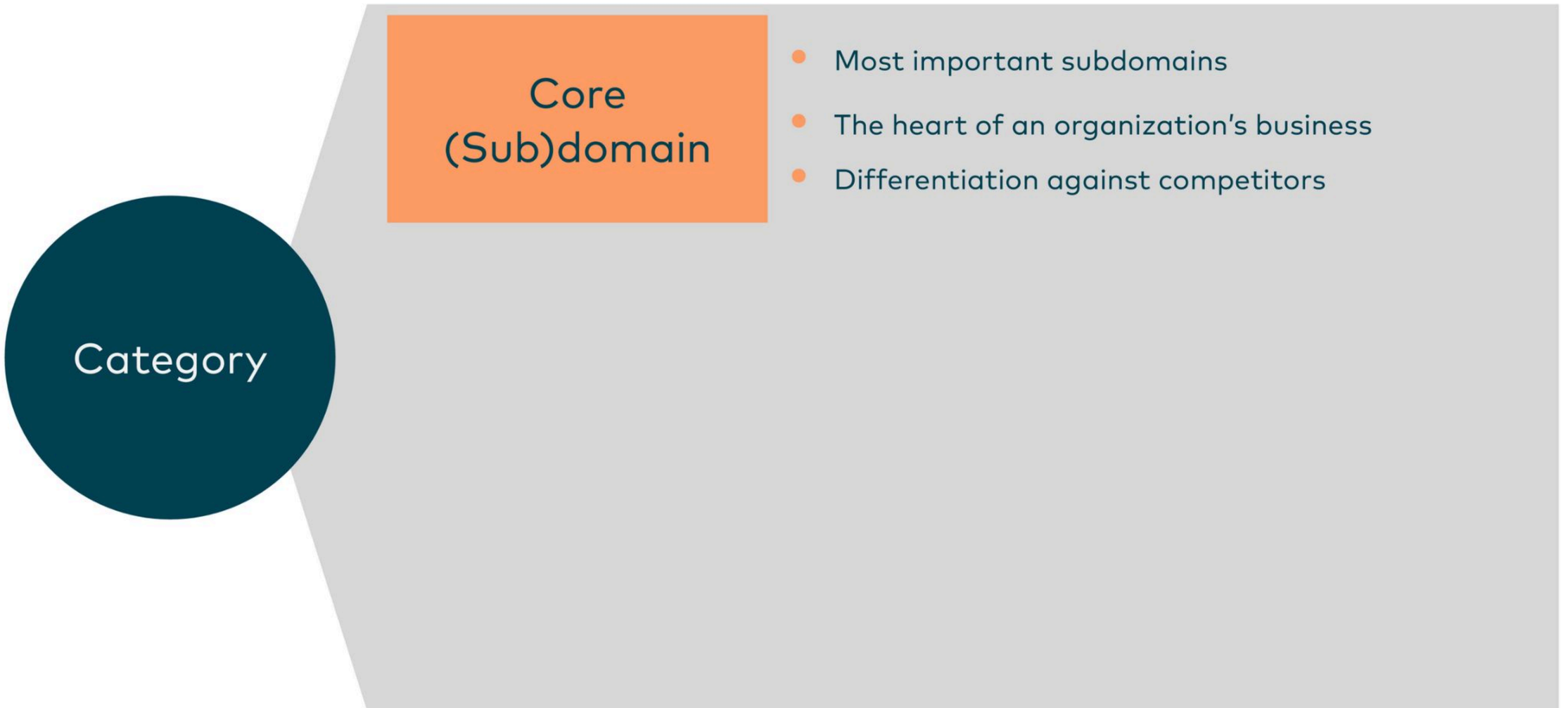




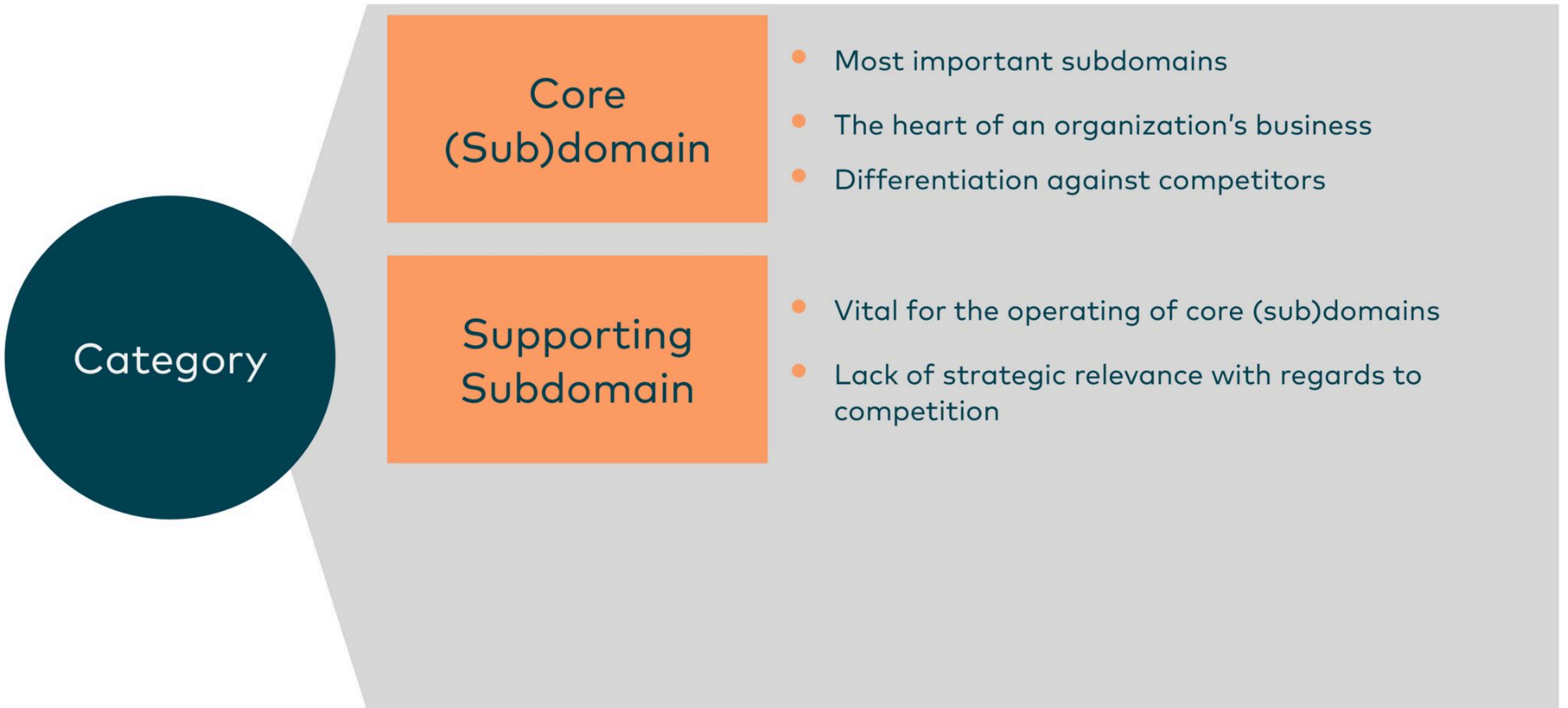
The model propagation from hell



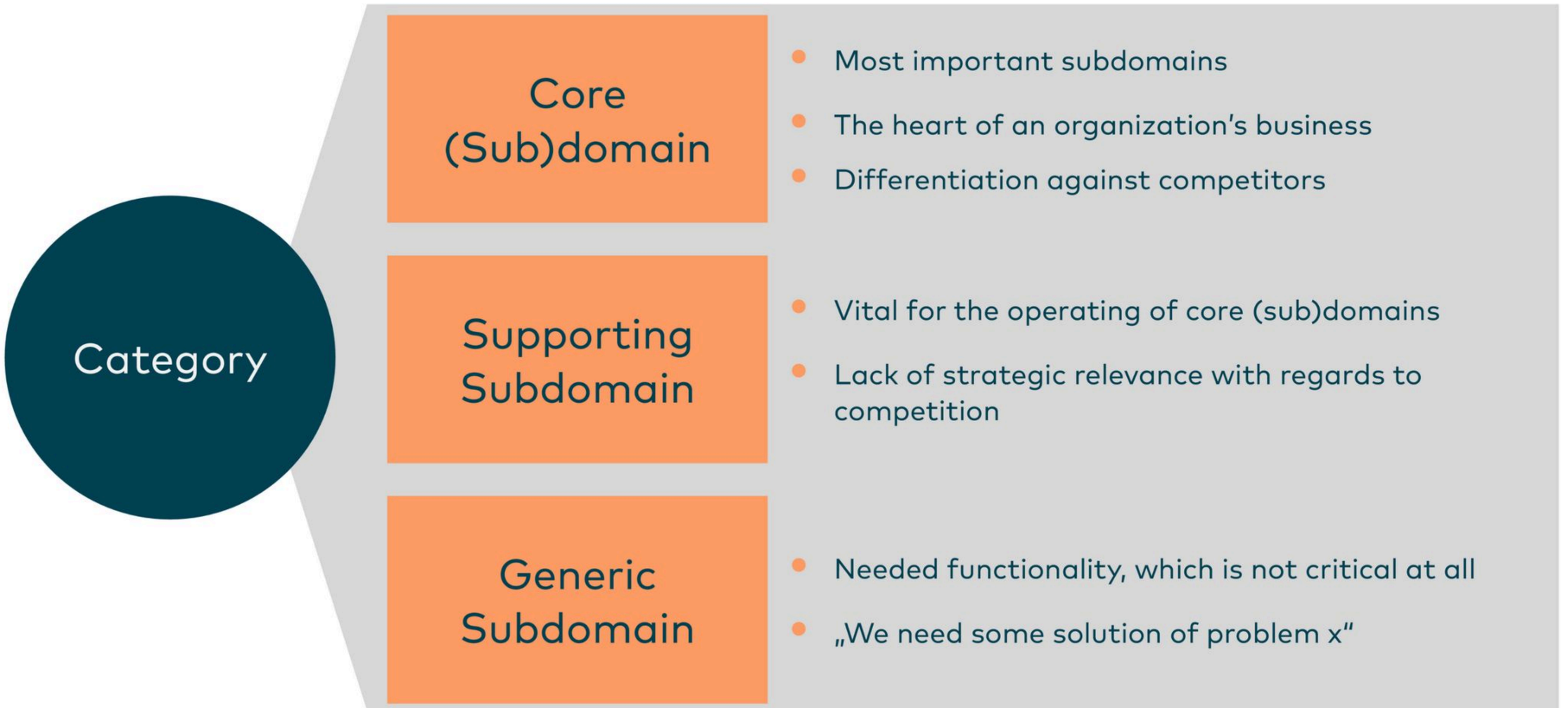
Domain categories can help as well

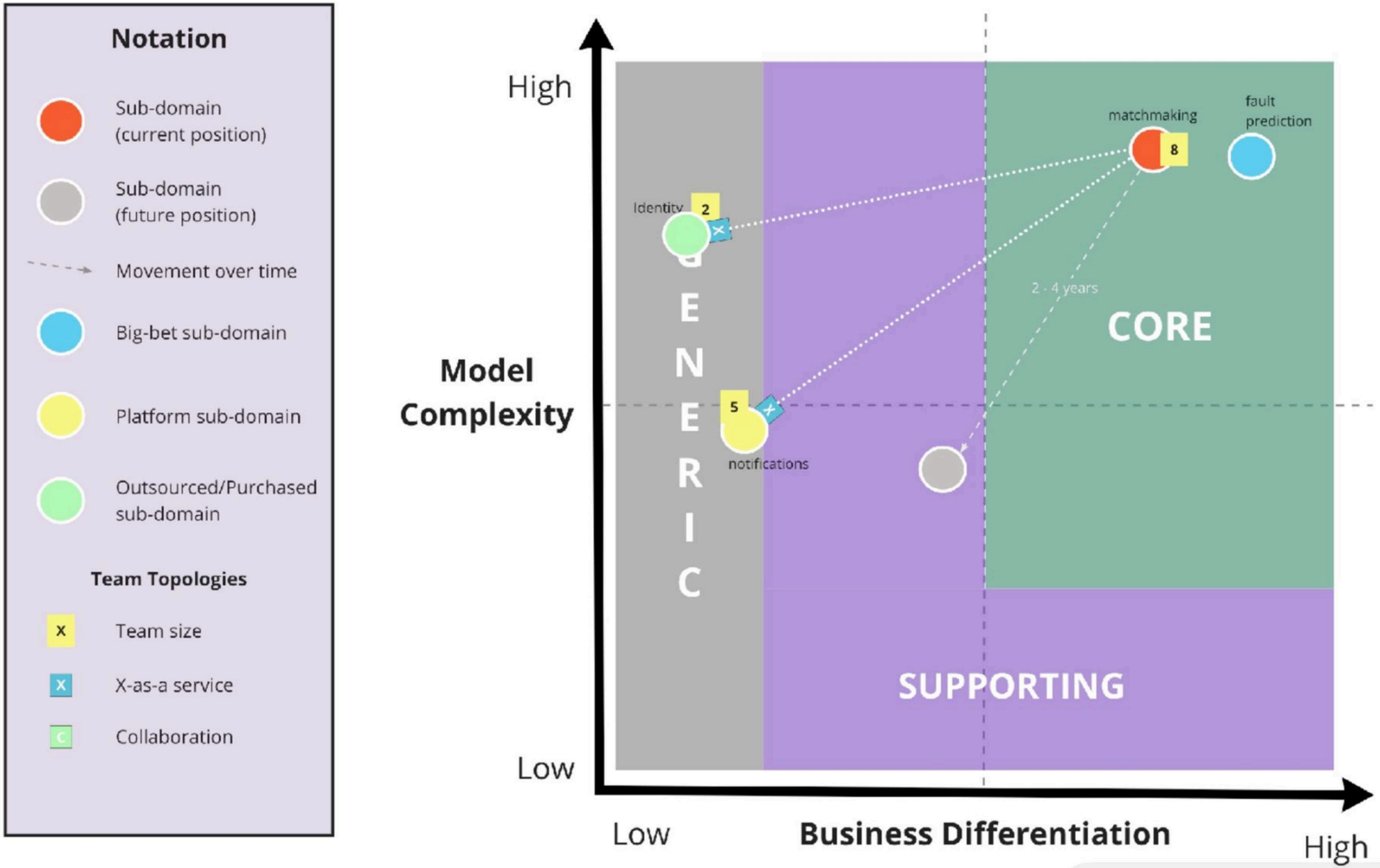


Domain categories can help as well



Domain categories can help as well





https://github.com/ddd-crew/core-domain-charts

Subdomain categories & Context Maps

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?

What are your thoughts on a core domain conforming to an external system?

Bad idea

Bad Idea

Bad idea

Risky

Not a good idea

Been there .. it’s hell..

Risky

Core should define its own modem

Grts hard to maintain over time.

Lot of risk

It depends

Bye bye core domain! Bad idea!

StupidMcstupidface

No good idea

Then you don‘t have a core domain ...

If the external system is specialized and well: might be feasible

Very bad idea

Bad idea

Very dangerous

Ew

Risky

That is no core domain then.

Need to implement acl

Other domains bleeding into your own core domain.

Mostly bad

You're not in charge of your business

A lot of changes can come in any time, cannot plan, can get blocked. Not a good idea :D

This can make a company fail

Dependance from the outside World where you have no influence

Too risky, an ACL would be better.
This is bad idea!

Depends on external system

bad idea, domain should be independent only on itself

Hi Mom

💣💣💣💣

Hell

Bad Idea

A major risk, which might deemed acceptable, but should be considered.

Sounds like a bad idea

Weak position to be in.

Bad idea

Bad idea

stupid idea

Depends, but generally bad

It depends

Terrible idea

Totally ok for a young business to prove there is a market

Bad Idea, better use acl

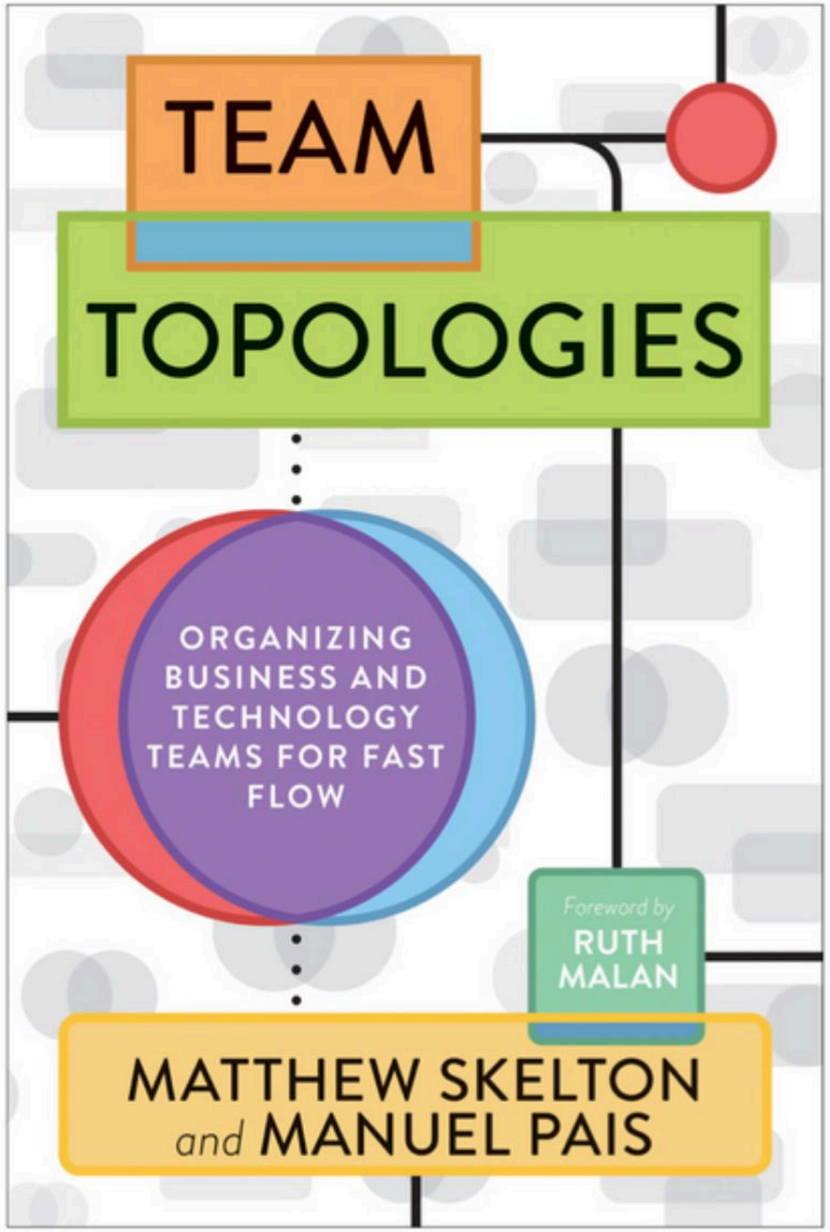
There’s a risk that our core domain may not be innovative as we think.

It depends but an ACL is recommended

You want to take control of core domains

I think, if the external system matches our needs we can use it.

If it changes we will do the ACL later



Team Interaction Modes

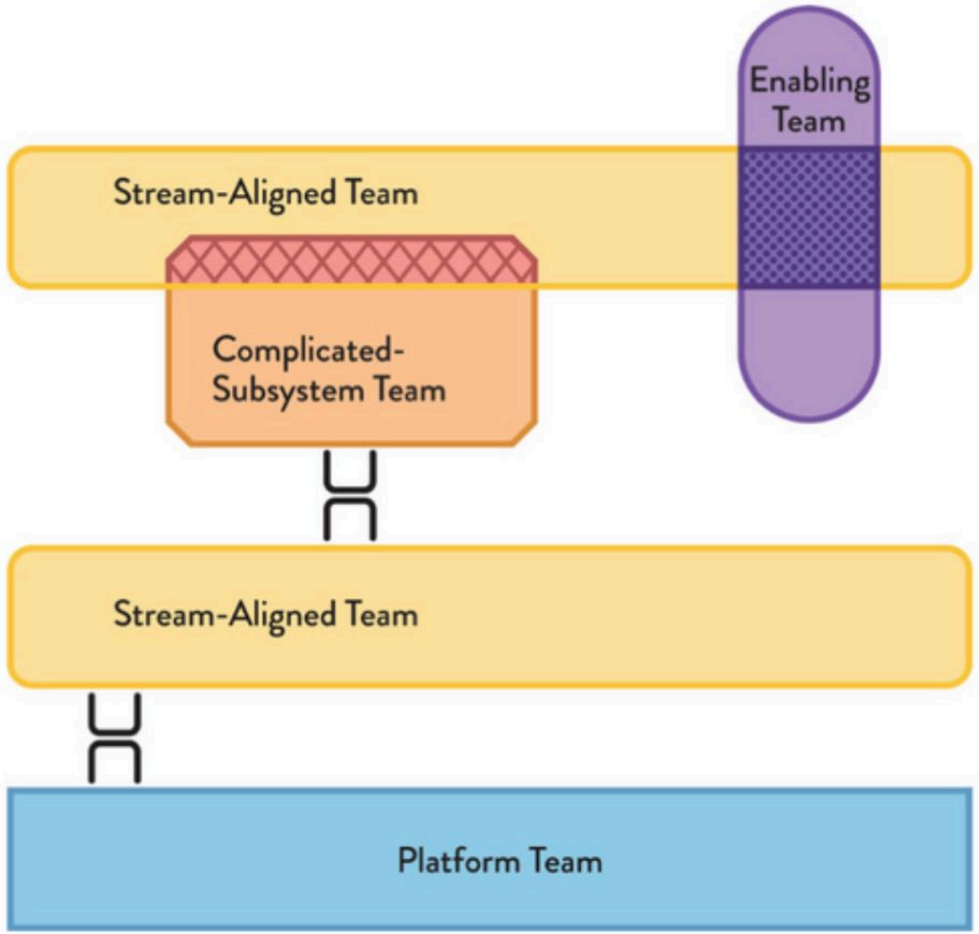
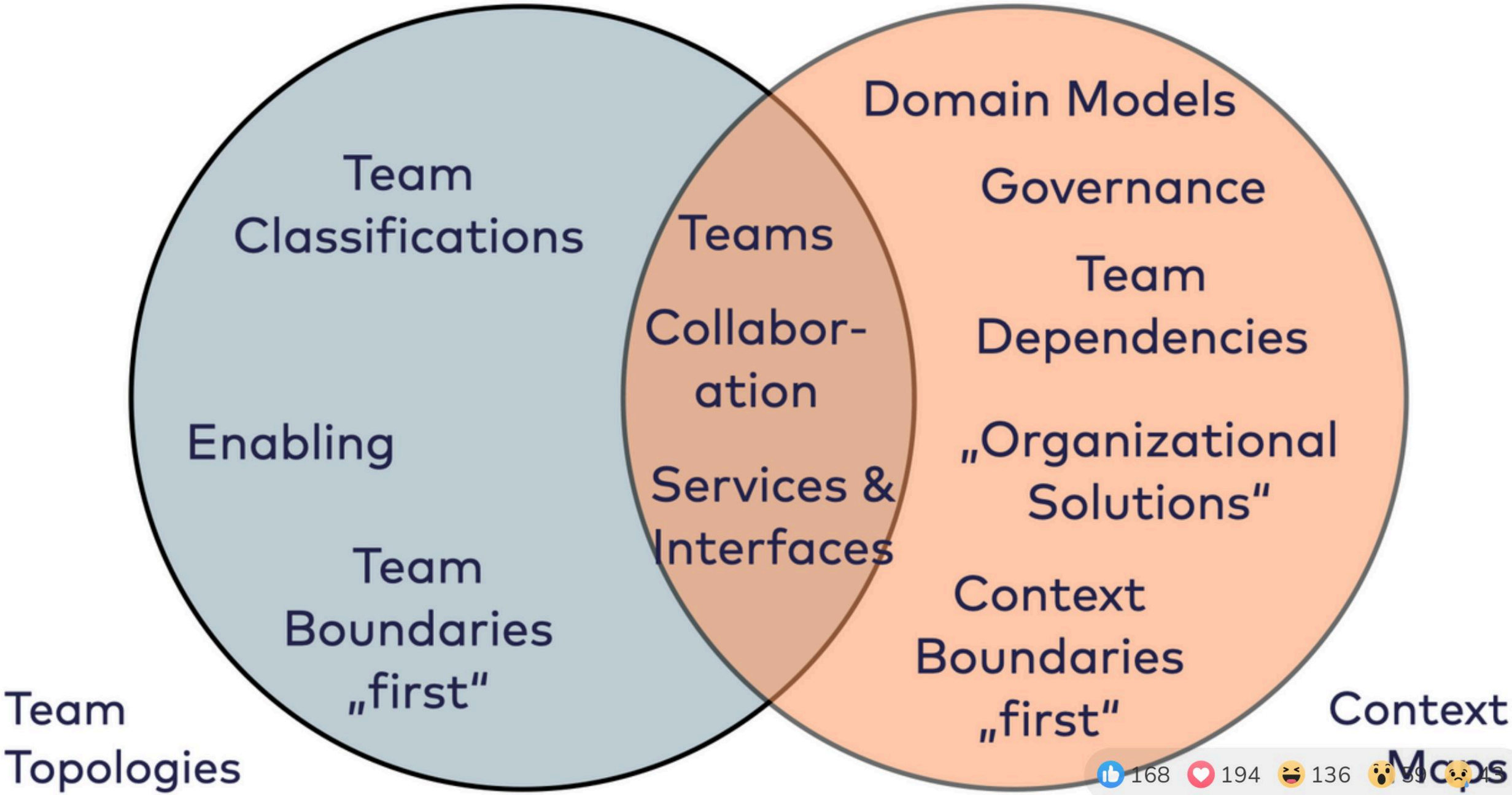
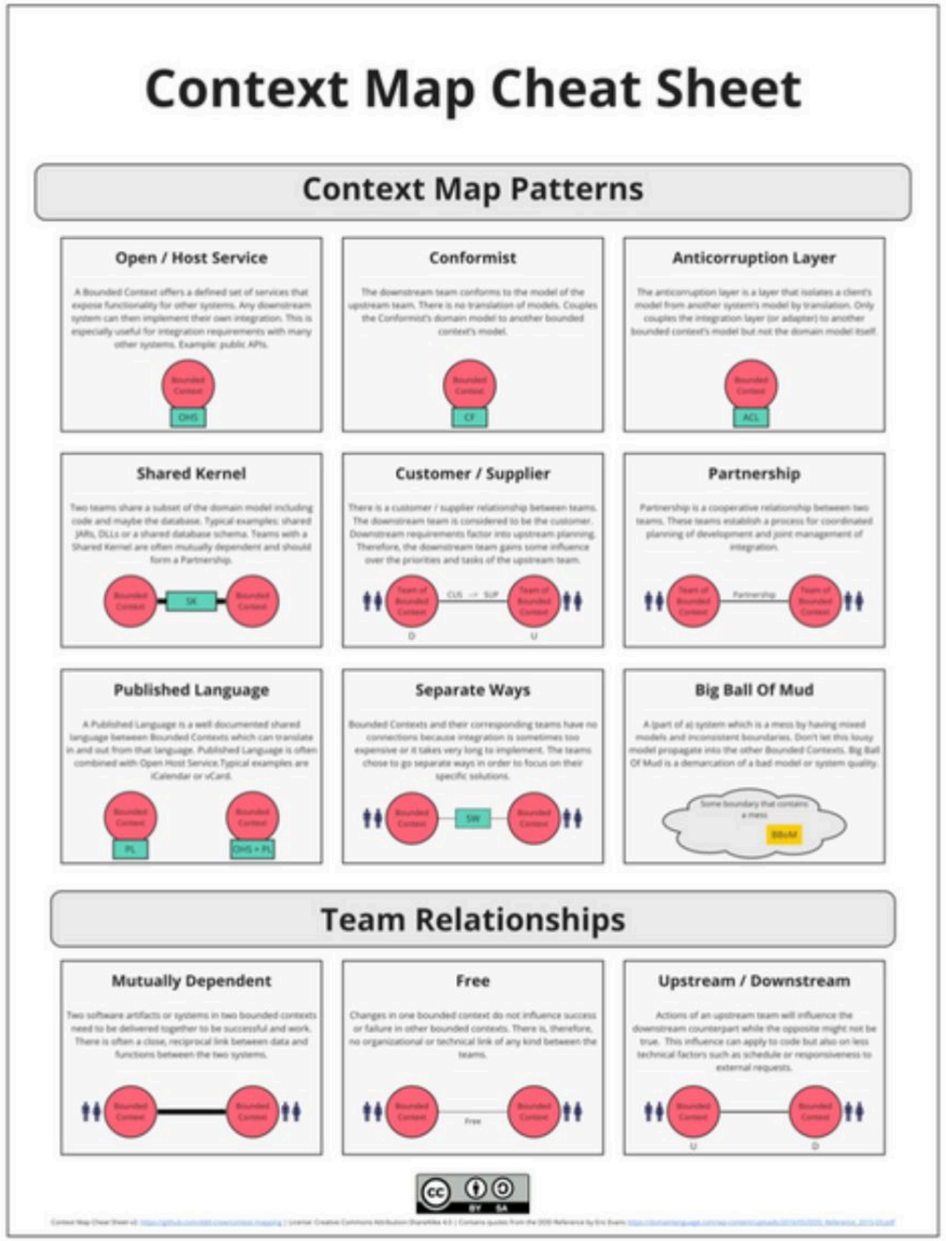


Image taken from the Team Topologies book





Check out DDD Crew on GitHub

- Cheat Sheet for all of the patterns and Team Relationships
- Context Mapping Starter Kit for Miro (as a downloadable Board Backup)
- Creative Commons

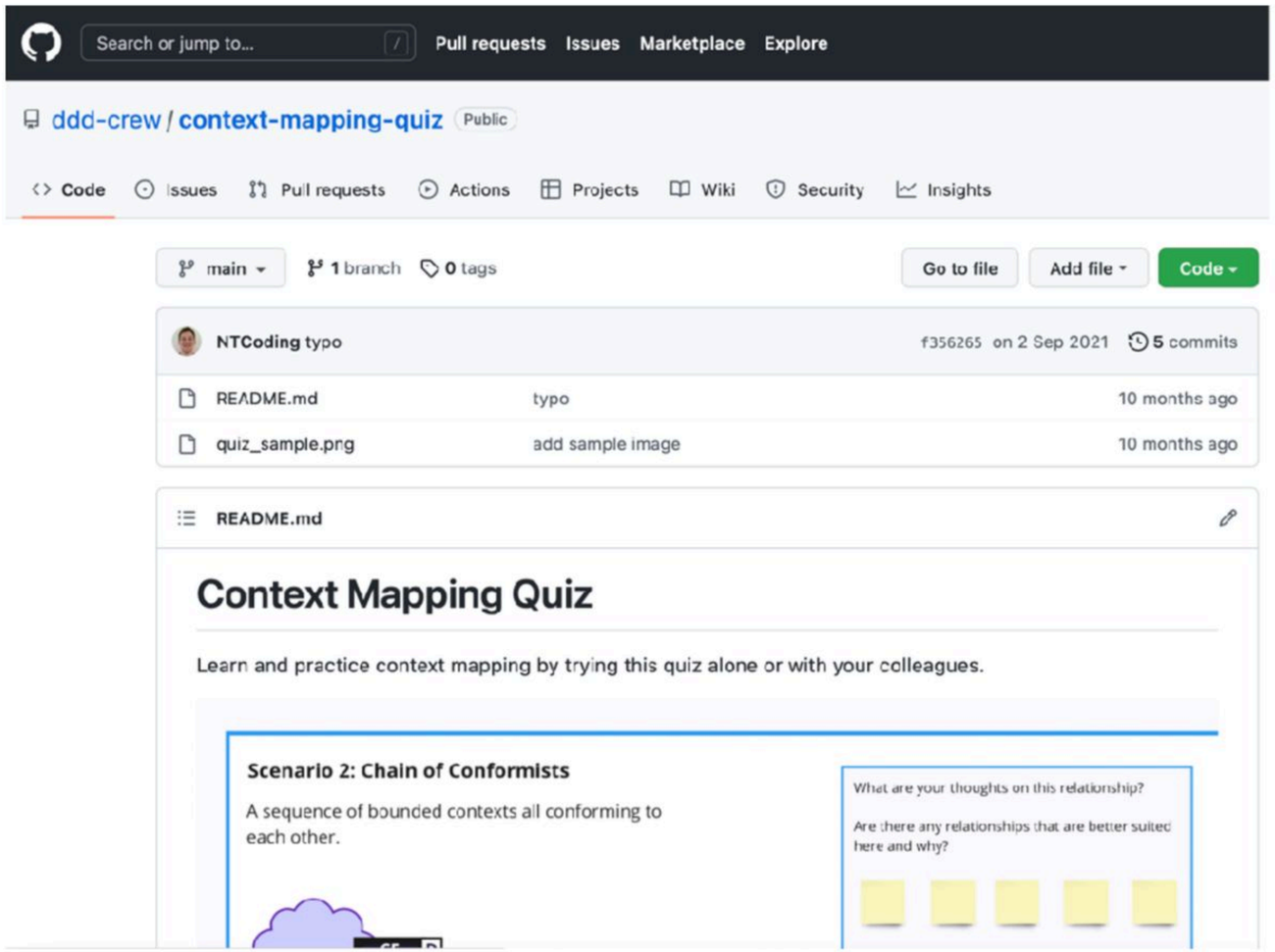
https://github.com/ddd-crew/context-mapping

Get my DDD book cheaper

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

Context Mapping Quiz

- Miro based quiz game on various scenarios
- Creative Commons



https://github.com/ddd-crew/context-mapping-quiz

168 194 136 59 43

Thanks! Questions?

INNOQ
www.innoq.com

Michael Plöd
michael.ploed@innoq.com

Follow me on Twitter: @bitboss

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim
+49 2173 3366-0

Ohlauer Str. 43
10999 Berlin

Ludwigstr. 180E
63067 Offenbach

Kreuzstr. 16
80331 München

Hermannstrasse 13
20095 Hamburg

Erftstr. 15-17
50672 Köln

Königstorgraben 11
90402 Nürnberg