

Combining Team Topologies with Context Maps

INNOQ



MICHAEL PLÖD
FELLOW

Michael Plöd

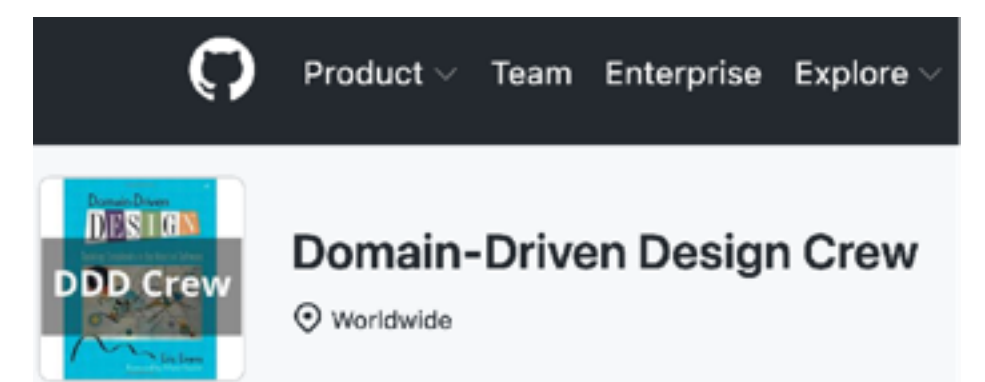
Fellow at INNOQ

Follow me on Twitter under @bitboss

Current consulting topics:

- Domain-Driven Design
- Team Topologies
- Transformation from IT Delivery to digital product orgs

Regular speaker at (inter-)national conferences and author of a book + various articles



Hands On
**DOMAIN-
DRIVEN
DESIGN**
by example

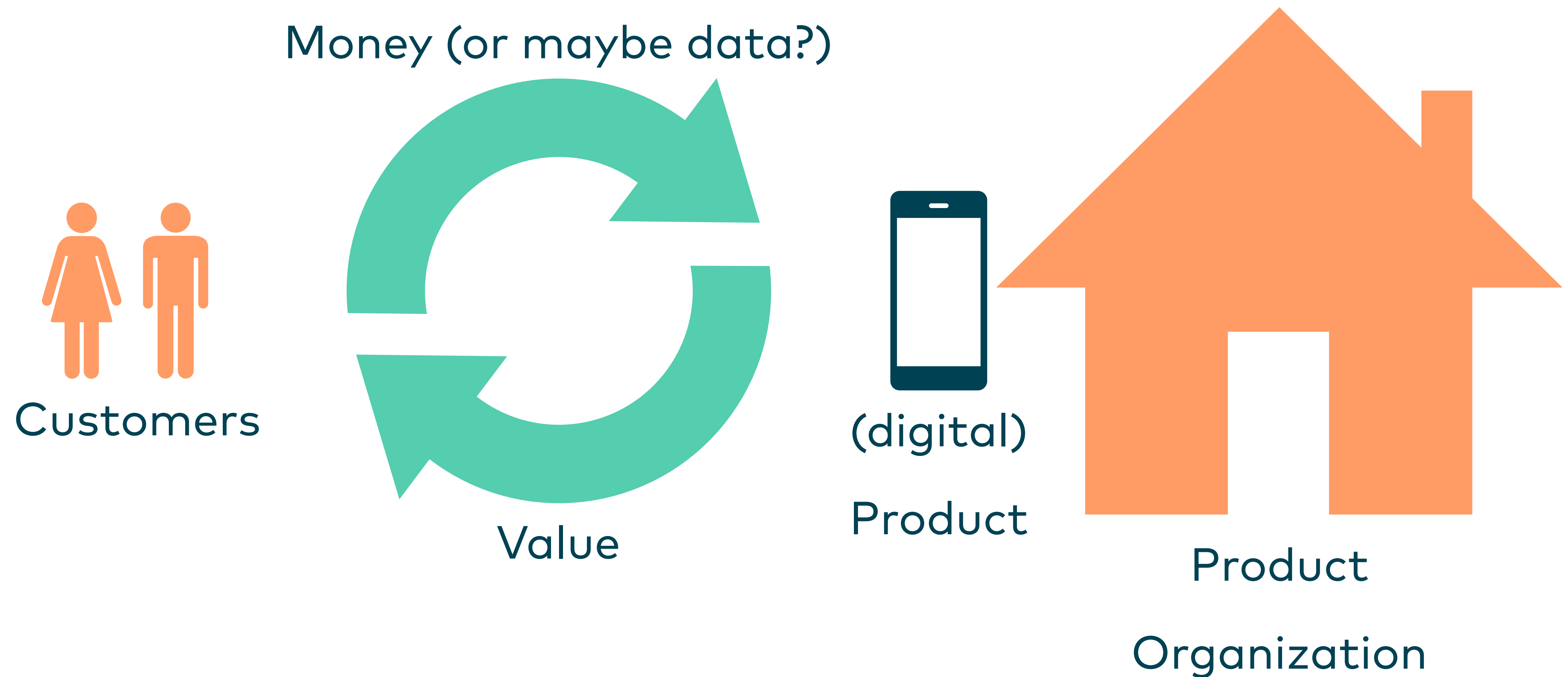
Michael Plöd

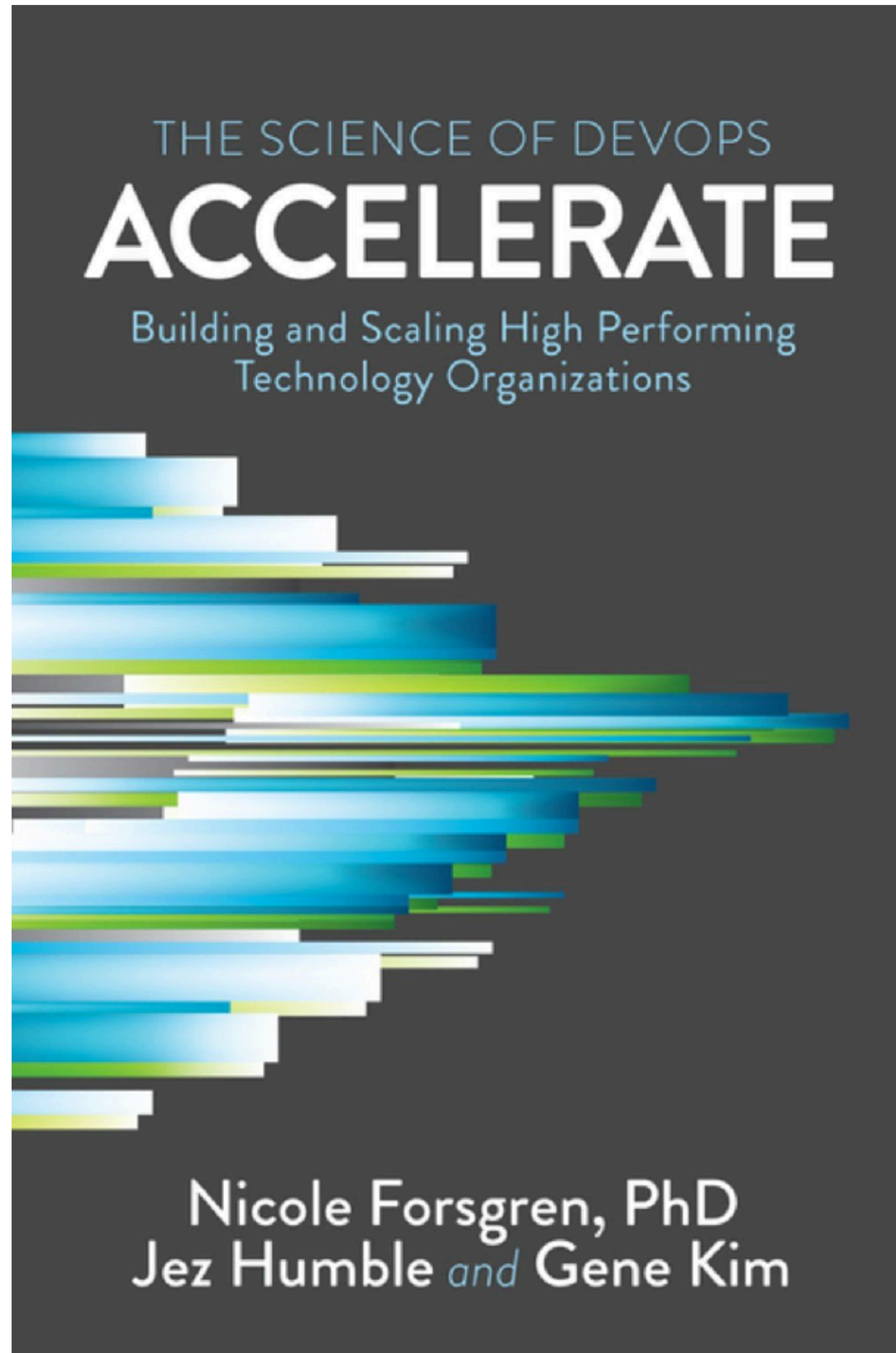
**Get my DDD book
cheaper**



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

How can we maximize the value exchange with the customer in a continuous fashion at high velocity?



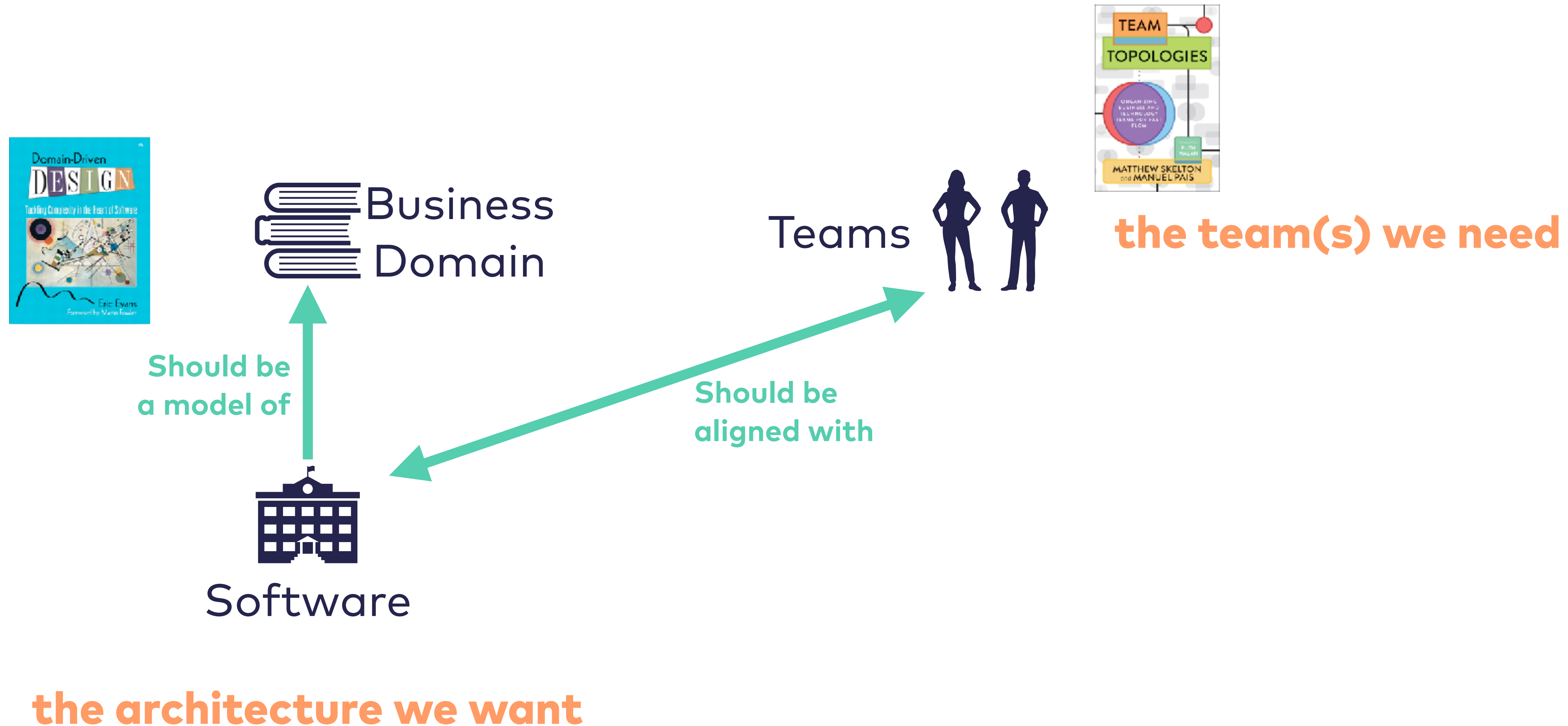


"A loosely coupled software architecture and org structure to match" is a key predictor of:

- Continuous Delivery Performance
- Ability to scale organization and increase performance linearly



Some basic ideas...



"Sociotechnical Architecture is about taking an holistic co-design approach to technical and organizational systems, given the inherent impact they have on each other."

Eduardo Da Silva

<https://esilva.net>



**„Team assignments
are the first draft
of the
architecture“**



Michael Nygard

Author of „Release It“

**There are two boundaries to this
and we should align them**



**Team
Boundaries**

**Software
Boundaries**

„Good fences make good neighbors“

Robert Frost

NEW YORK TIMES BESTSELLER

"Provocative and fascinating." —MALCOLM GLADWELL

Daniel H. Pink

author of *A Whole New Mind*

DRIVE

The Surprising Truth
About What Motivates Us

Autonomy

Mastery

Purpose



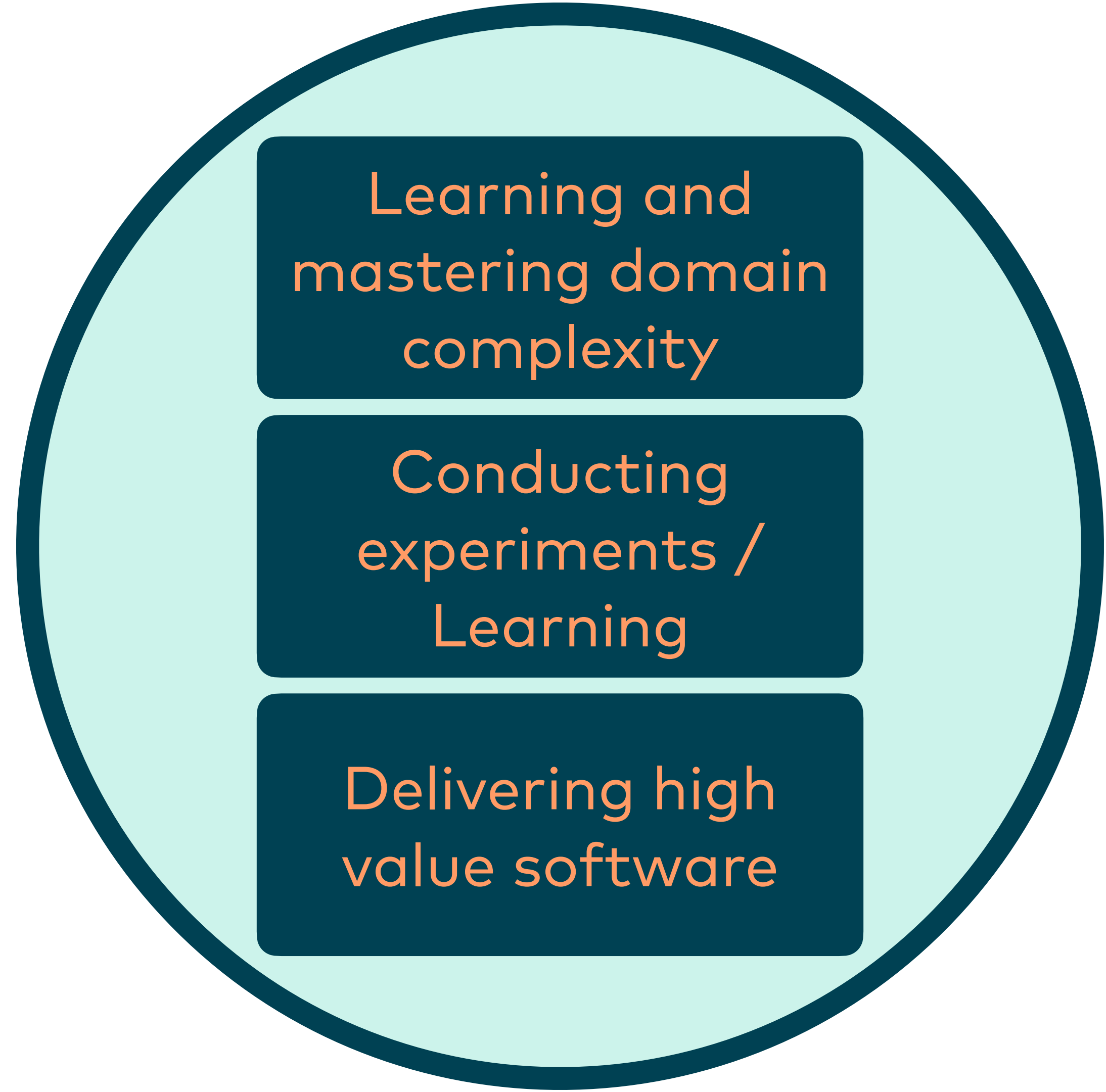
**We need good boundaries in which teams
can achieve**

Autonomy - Mastery - Purpose

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

Bounded Context

What we want
to achieve in a
**Bounded
Context**



NEW YORK TIMES BESTSELLER

"Provocative and fascinating." —MALCOLM GLADWELL

Daniel H. Pink

author of *A Whole New Mind*

DRIVE

The Surprising Truth
About What Motivates Us

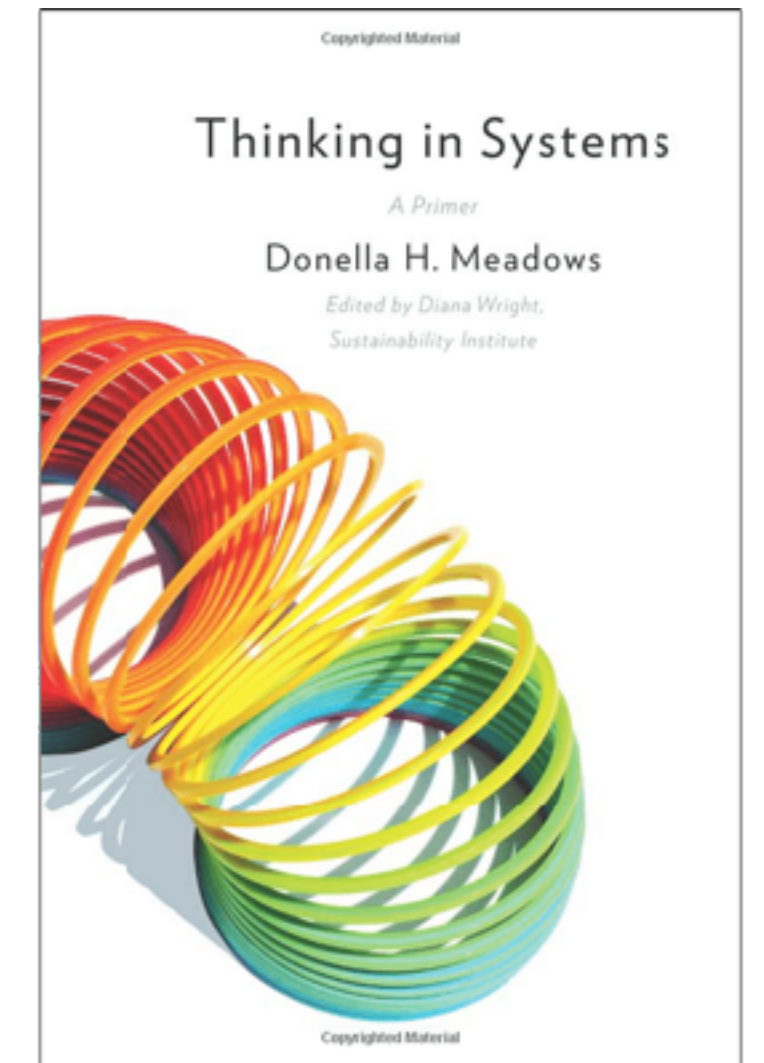
Autonomy

Mastery

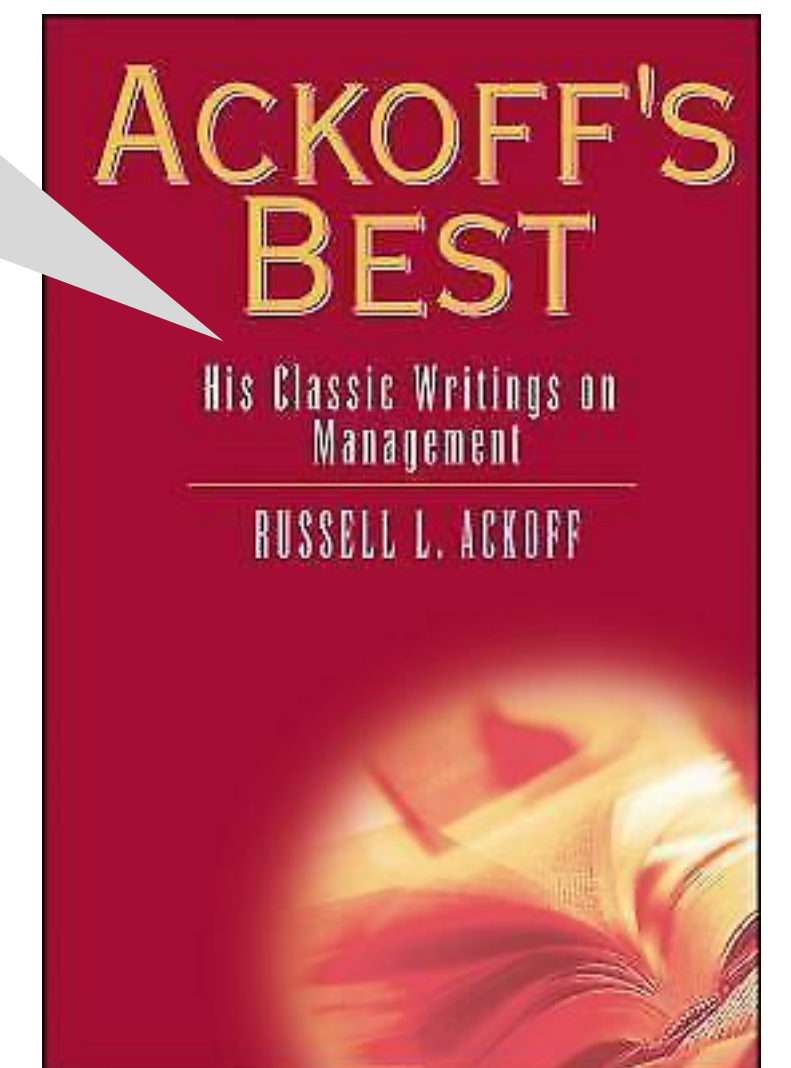
Purpose



Sociotechnical Architectures are a lot about **Systems Thinking** as well



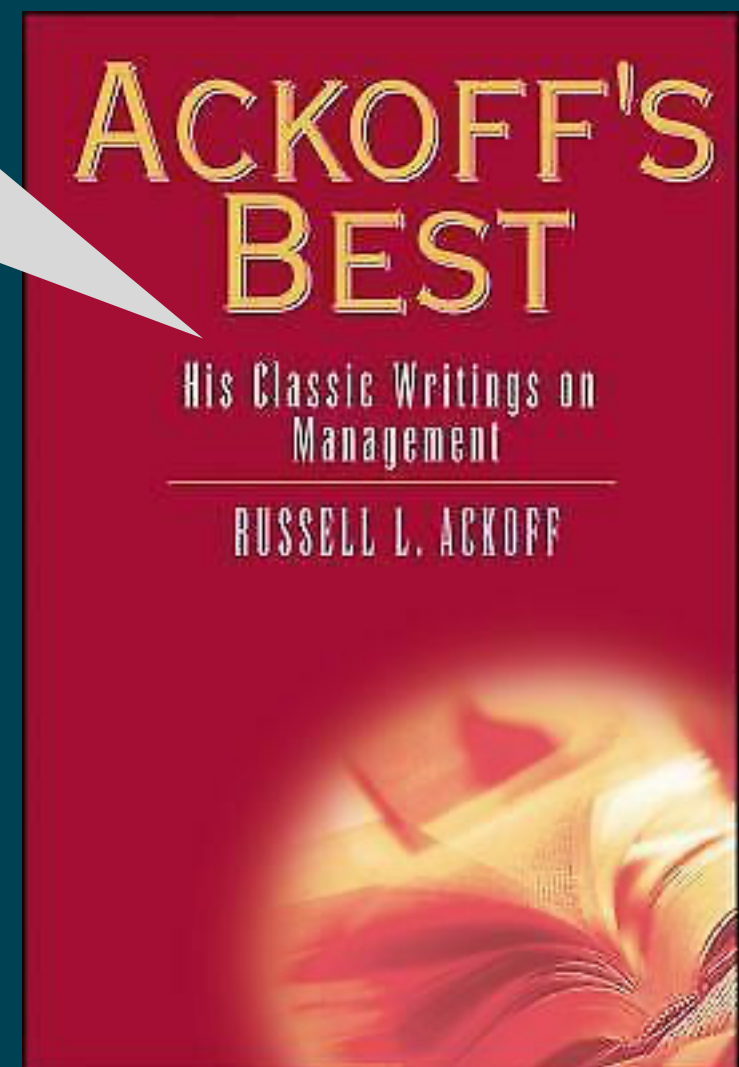
*To manage a
system effectively, you
might **focus on the
interactions of the parts**
rather than their behavior
taken separately*



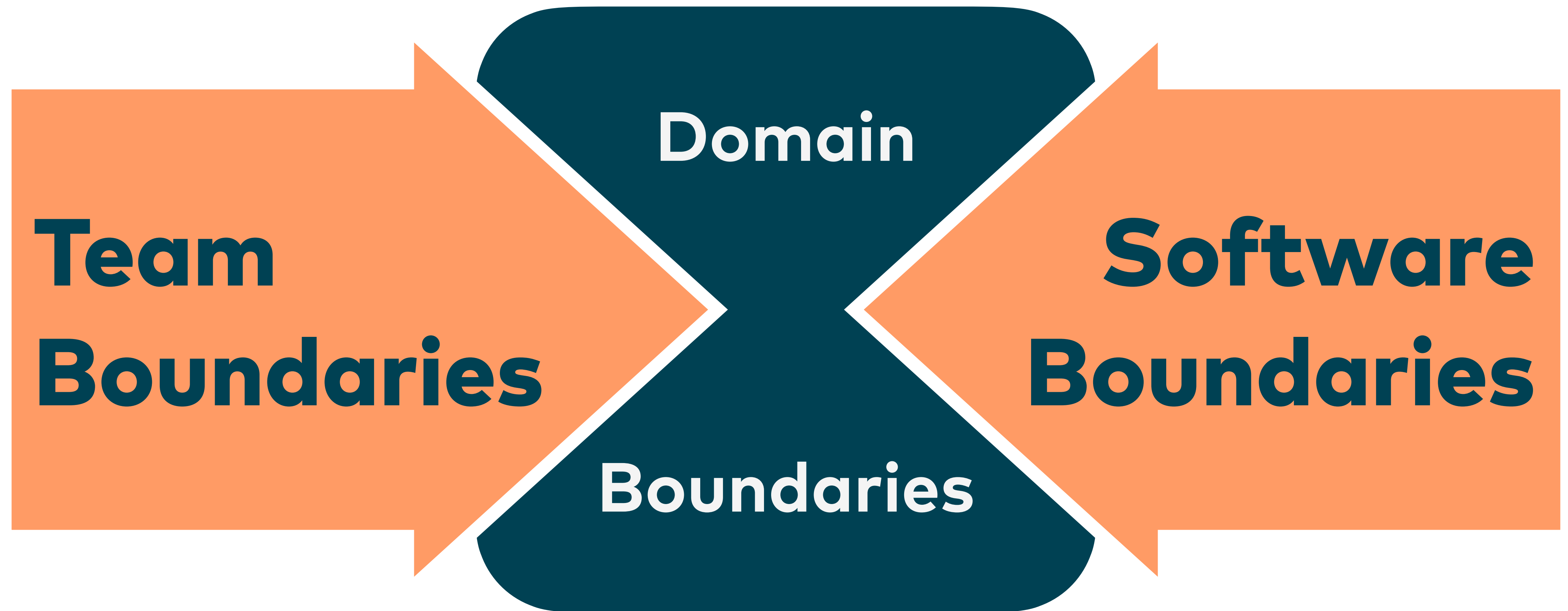
Autonomy

Isn't it about (maybe a reduction / lack of) interactions?

*To manage a system effectively, you might **focus on the interactions of the parts** rather than their behavior taken separately*



Align along domain boundaries



TEAM

TOPOLOGIES

ORGANIZING
BUSINESS AND
TECHNOLOGY
TEAMS FOR FAST
FLOW

Foreword by
RUTH
MALAN

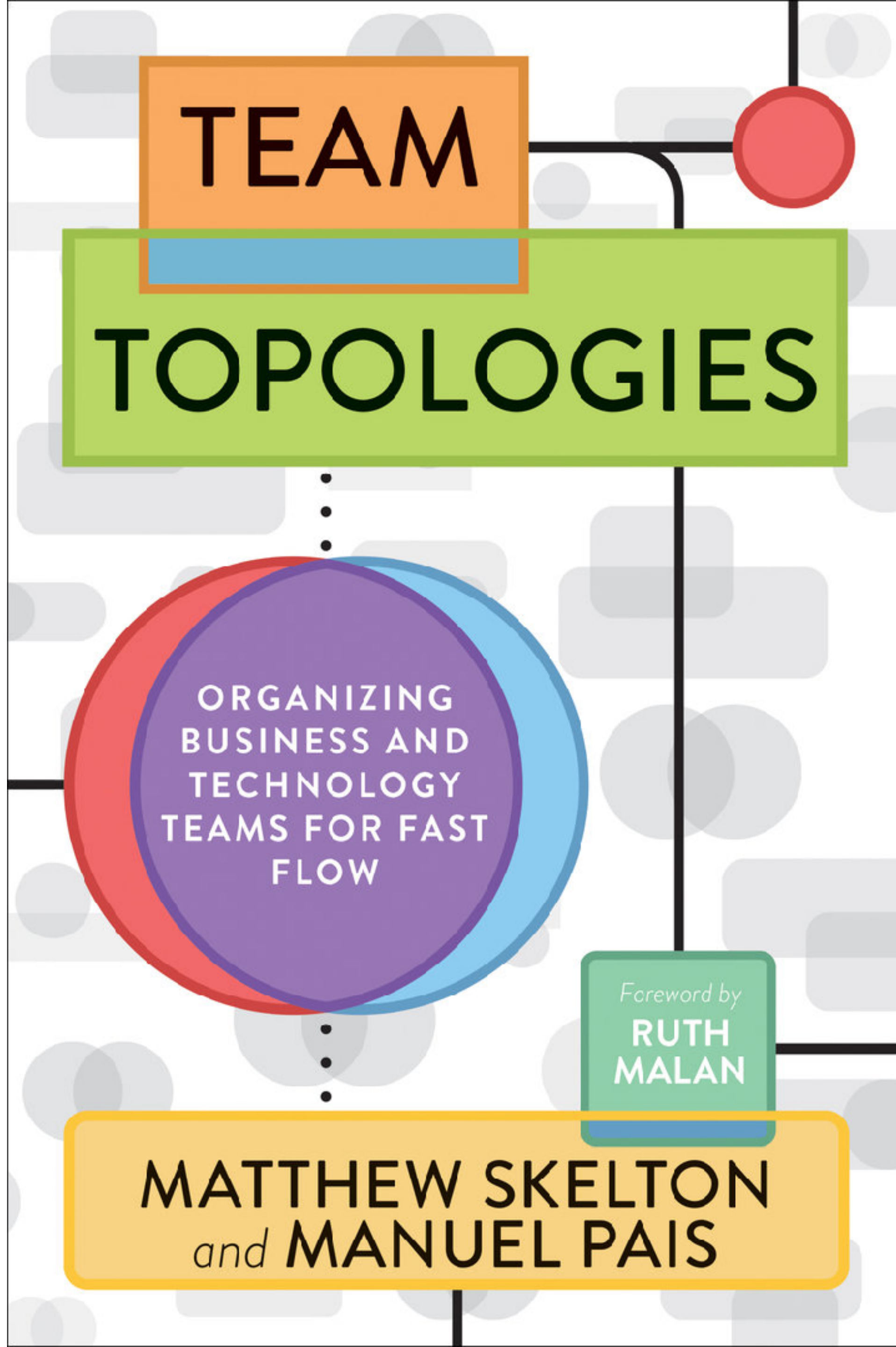
MATTHEW SKELTON
and MANUEL PAIS

"An architect should be thinking:

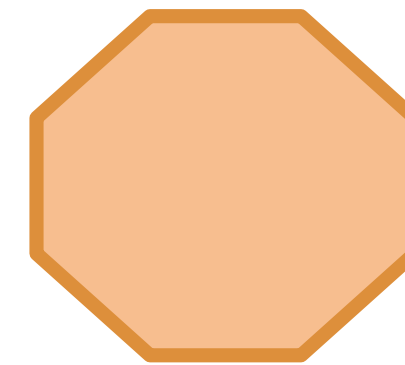
Which team interaction modes
are appropriate for these two
teams?

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

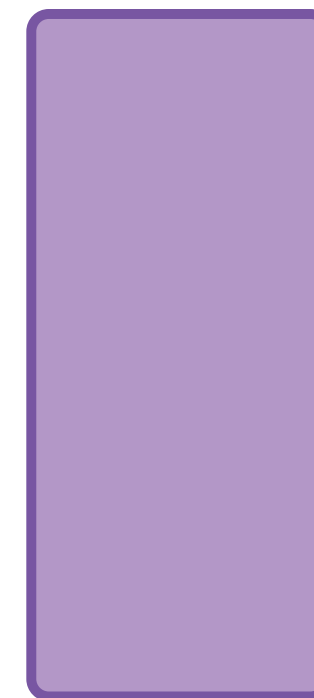




Fundamental Team Topologies



Complicated Subsystem



Enabling



Platform



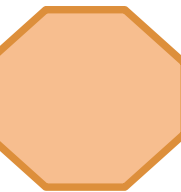
Stream-aligned

Stream-aligned Team



- Tailored to a business area or organizational capability (Bounded Context)
- Is intended to create customer value quickly, safely and autonomously without having to delegate parts of the work to other teams.

Complicated Subsystem Team



- Responsible for building and maintaining a part of the system that is highly dependent on specialist expertise
- Team manages the complexity of the subsystem using specific skills and expertise that are usually difficult to find or recruit.

Platform Team



- Should give stream-aligned teams the possibility to do their work with a high degree of autonomy,
- Platform provides self-service APIs, tools and services as an internal product

Enabling Team



- Work alongside the stream-aligned teams and support them in the area of knowledge building and empowerment.
- Have a strong collaborative nature and strive to understand the problems and shortcomings of the other teams
- Inhouse consulting team

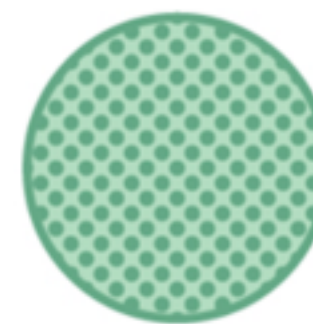
Team Interaction Modes



Collaboration



X-as-a-Service



Facilitating

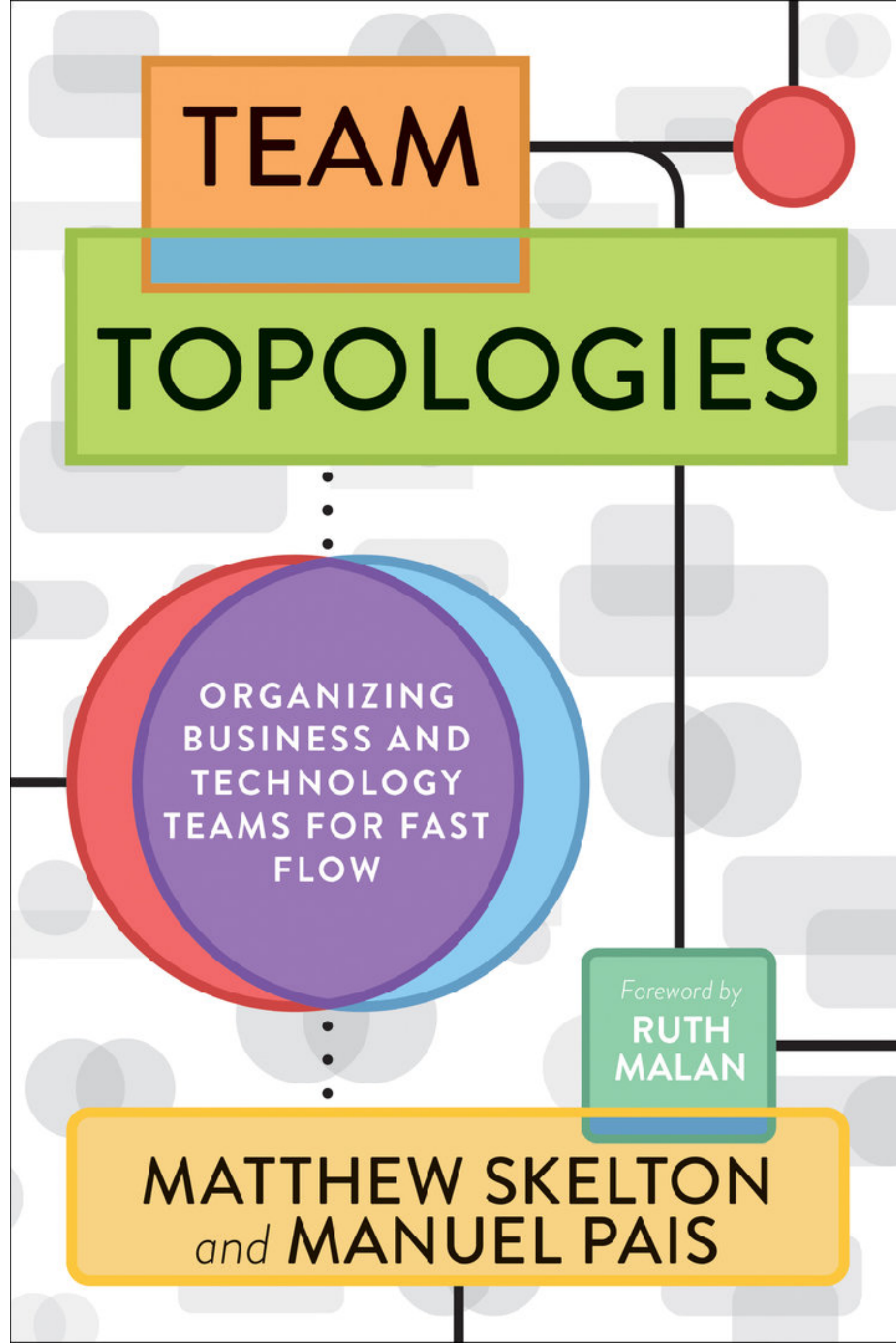


Image taken from the Team Topologies book

Team Interaction Modes

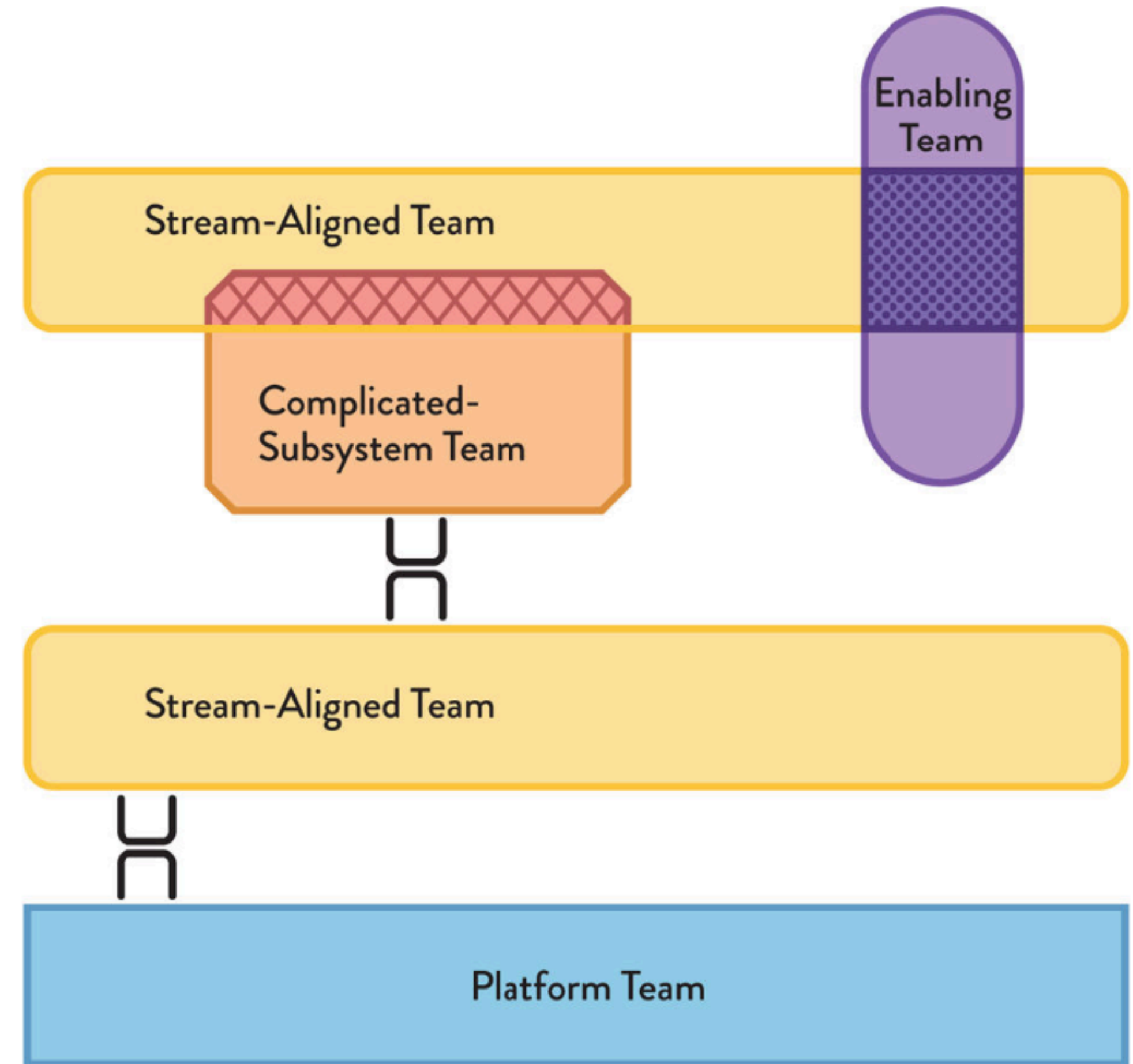
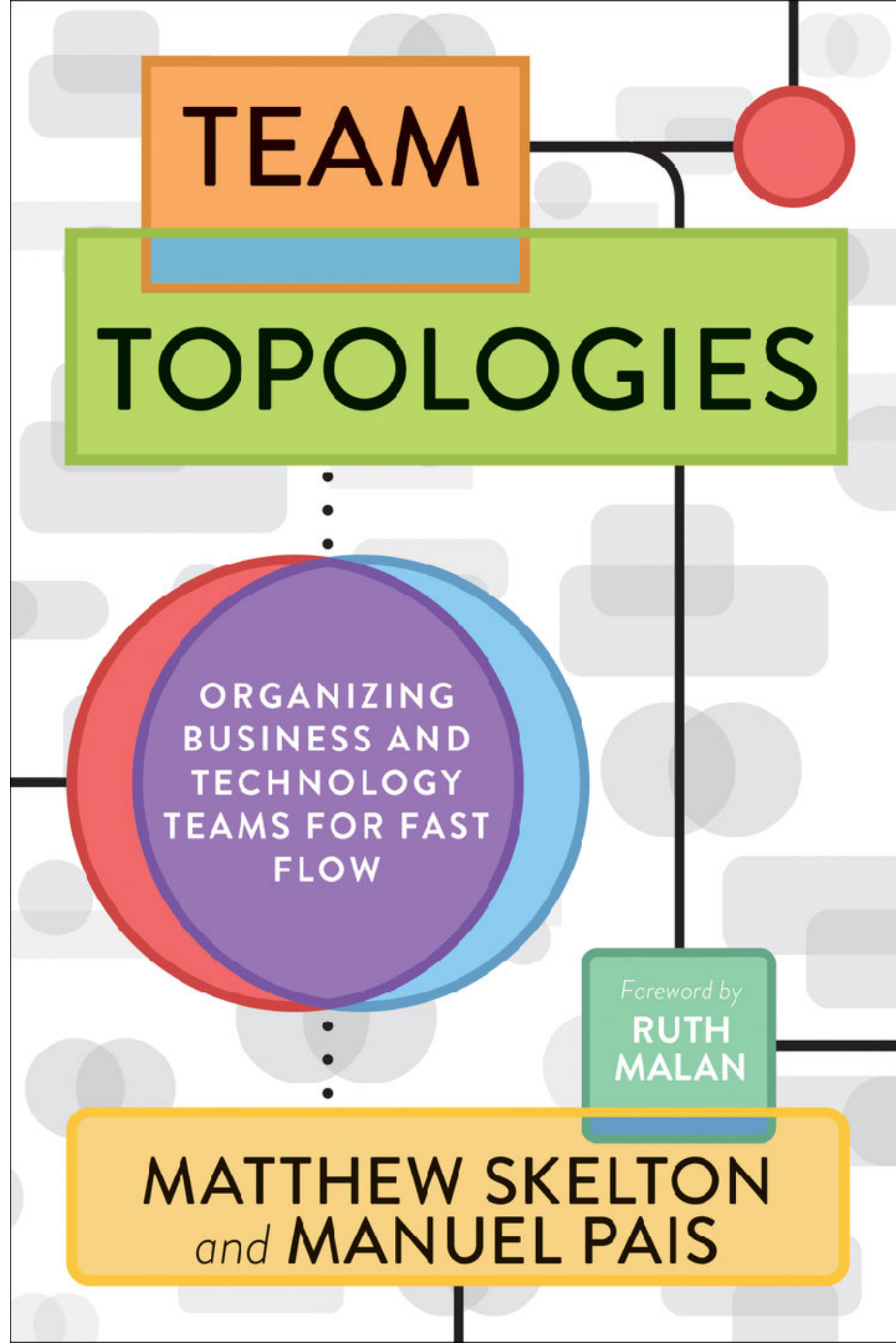
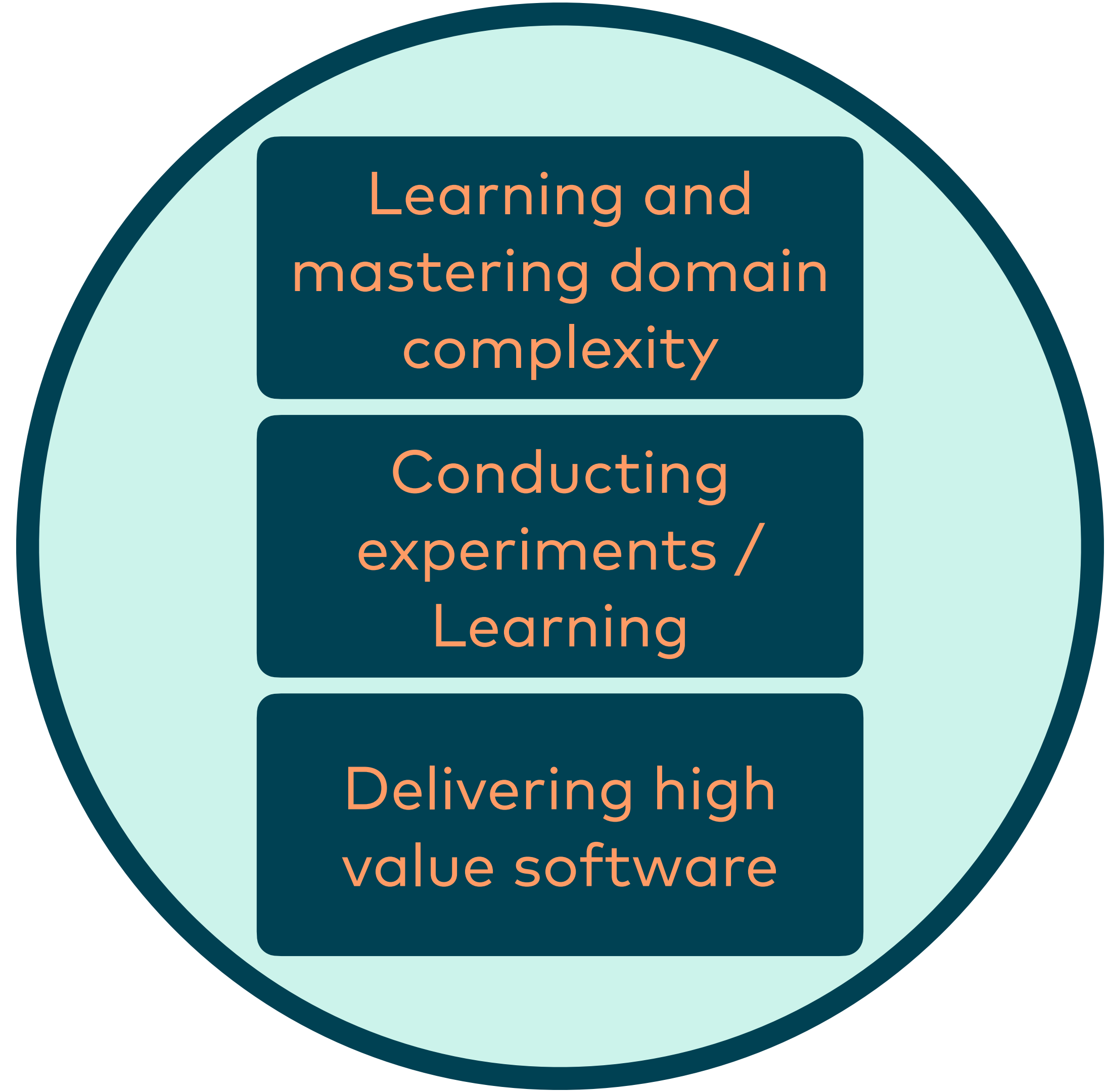
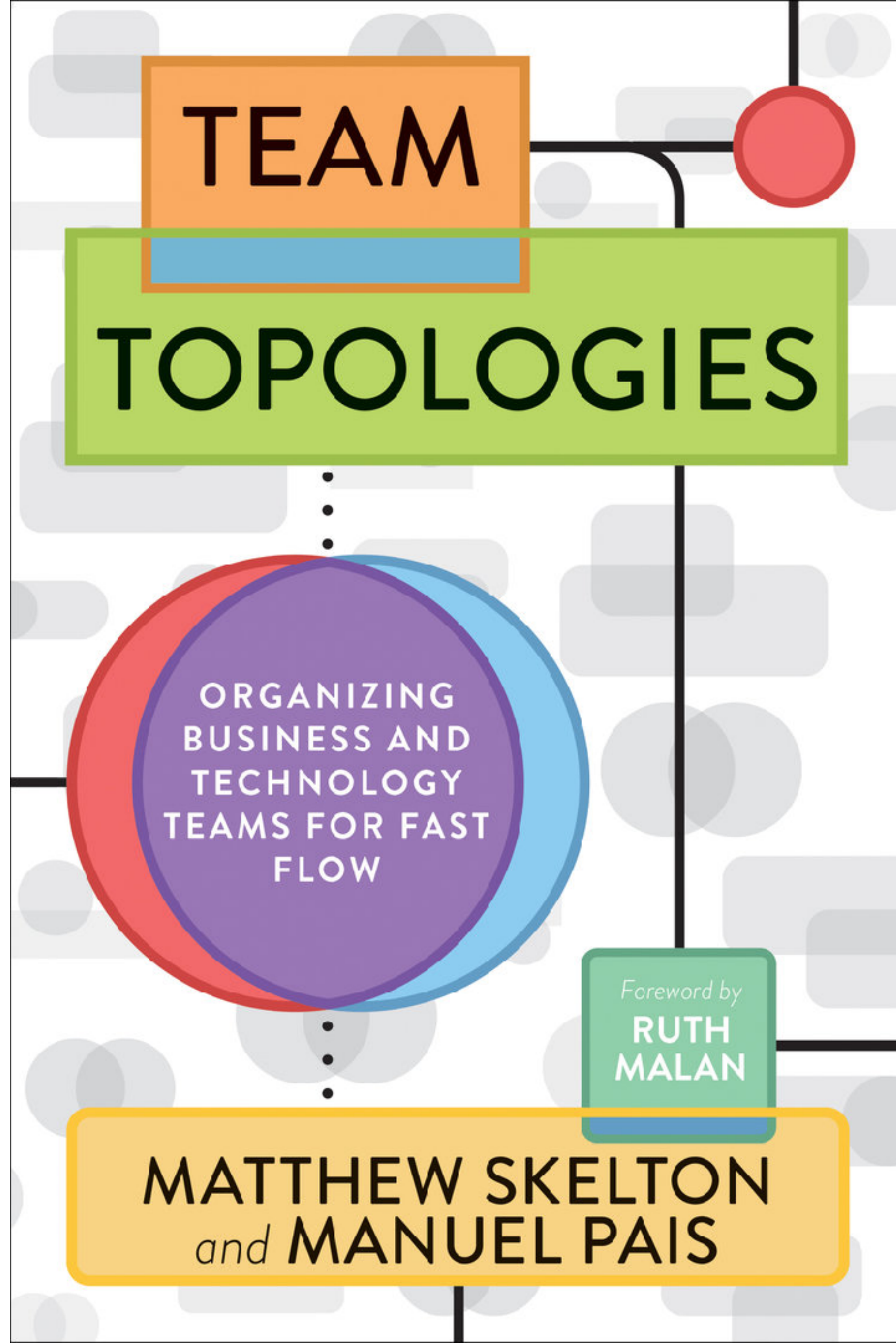


Image taken from the Team Topologies book

Mind the **COGNITIVE LOAD**

of the teams.
We need a
boundary for
this!





The
Bounded Context
(as a fracture plan)
is a

team first
boundary



Team
Classifications Teams

Collaboration

Enabling

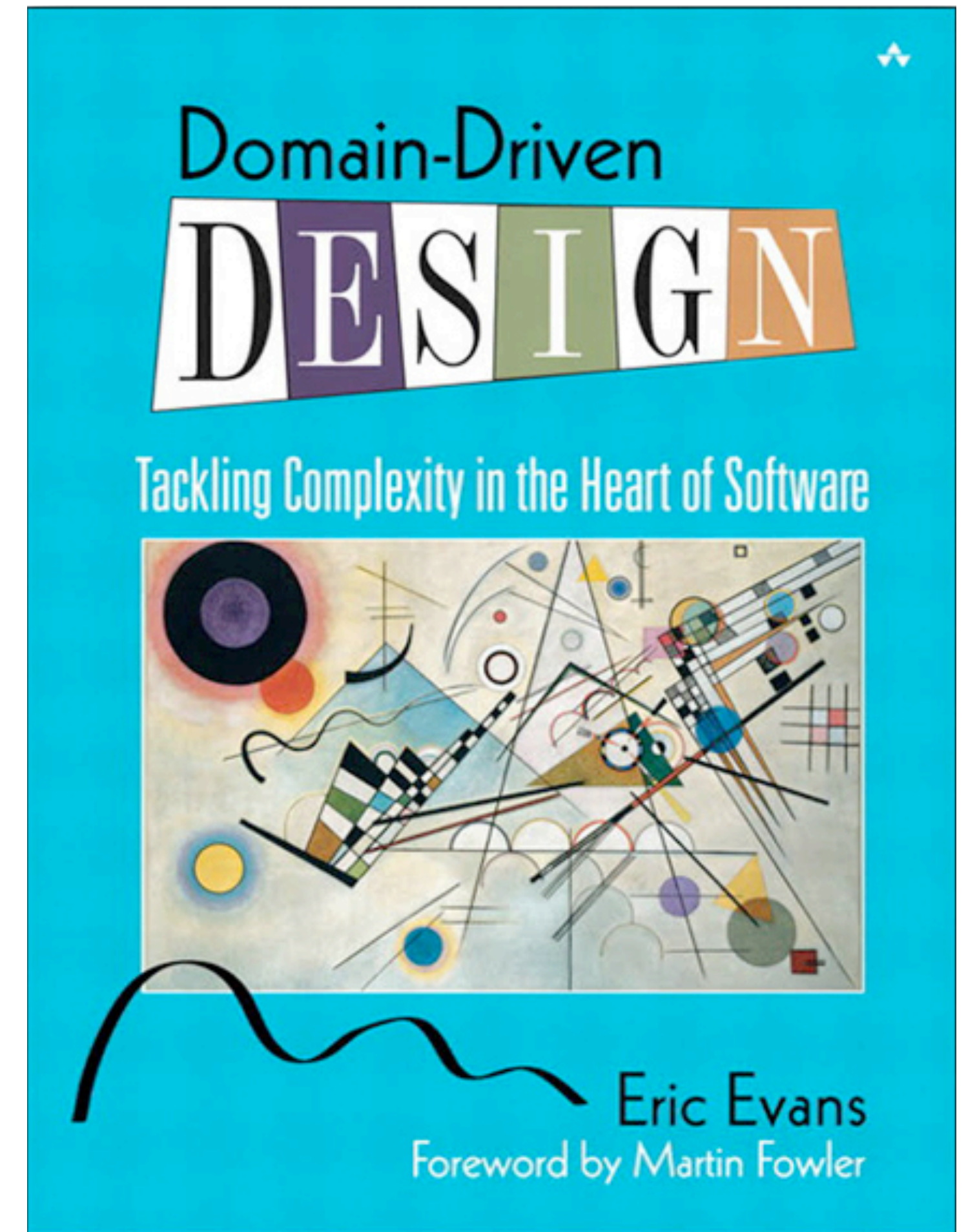
Services &
Interfaces

Team
Boundaries
„first“

Team
Topologies

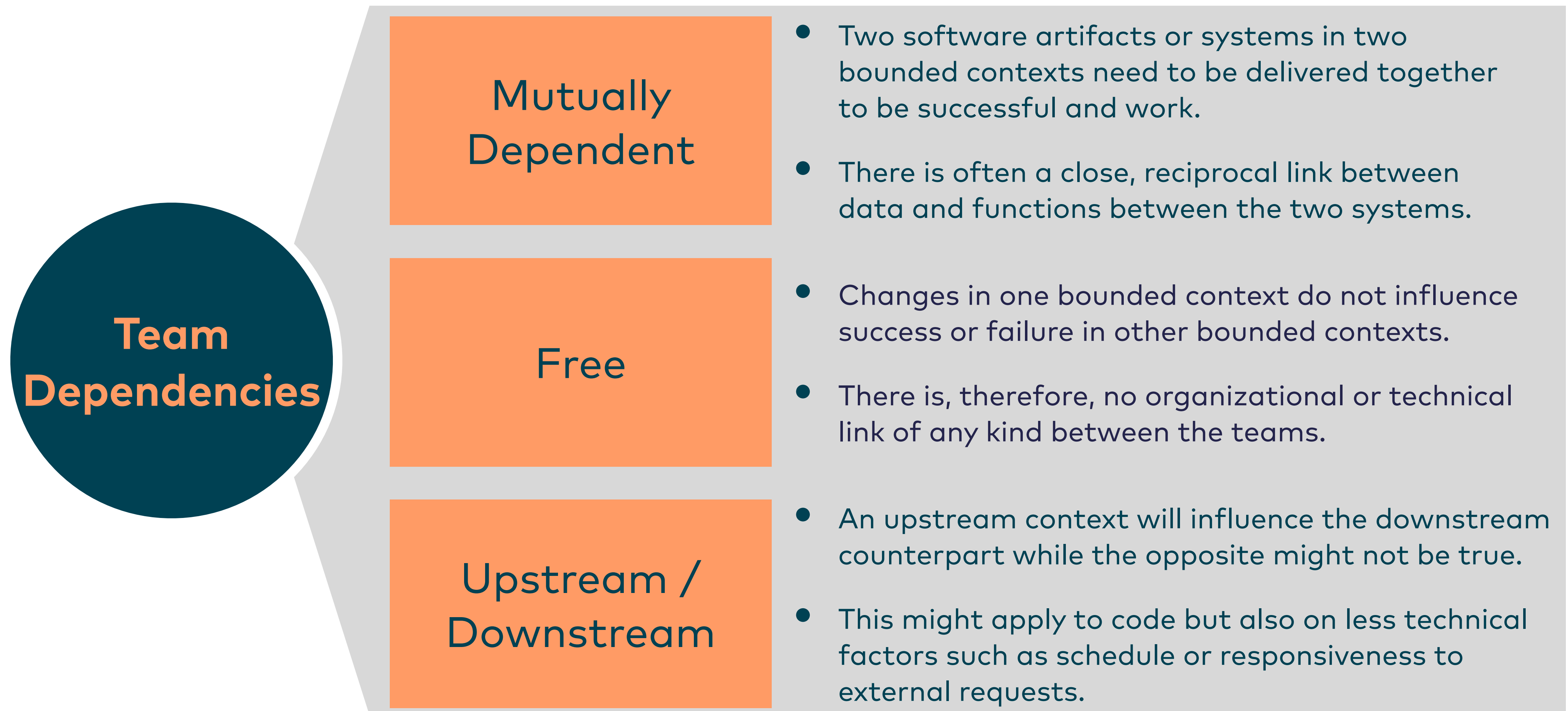
**Strategic Domain
Driven Design**
also has a technique
which can be used
to visualize
sociotechnical
relationships:

CONTEXT MAPS



Context Maps in Domain Driven Design address relationships between Bounded Contexts and teams. They start „bounded context first“.

Dependencies between teams



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

Context Map Cheat Sheet

Context Map Patterns

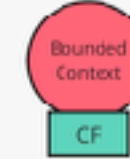
Open / Host Service

A 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. Example: public APIs.



Conformist

The downstream team conforms to the model of the upstream team. There is no translation of models. Couples the Conformist's domain model to another bounded context's model.



Anticorruption Layer

The anticorruption layer is a layer that isolates a client's model from another system's model by translation. Only couples the integration layer (or adapter) to another bounded context's model but not the domain model itself.



Shared Kernel

Two teams share a subset of the domain model including code and maybe the database. Typical examples: shared JARs, DLLs or a shared database schema. Teams with a Shared Kernel are often mutually dependent and should form a Partnership.



Customer / Supplier

There is a customer / supplier relationship between teams. The downstream team is considered to be the customer. Downstream requirements factor into upstream planning. Therefore, the downstream team gains some influence over the priorities and tasks of the upstream team.



Partnership

Partnership is a cooperative relationship between two teams. These teams establish a process for coordinated planning of development and joint management of integration.



Published Language

A Published Language is a well documented shared language between Bounded Contexts which can translate in and out from that language. Published Language is often combined with Open Host Service. Typical examples are iCalendar or vCard.



Separate Ways

Bounded Contexts and their corresponding teams have no connections because integration is sometimes too expensive or it takes very long to implement. The teams chose to go separate ways in order to focus on their specific solutions.



Big Ball Of Mud

A (part of a) system which is a mess by having mixed models and inconsistent boundaries. Don't let this lousy model propagate into the other Bounded Contexts. Big Ball Of Mud is a demarcation of a bad model or system quality.



Team Relationships

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

Actions of an upstream team will influence the downstream counterpart while the opposite might not be true. This influence can apply to code but also on less technical factors such as schedule or responsiveness to external requests.



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>

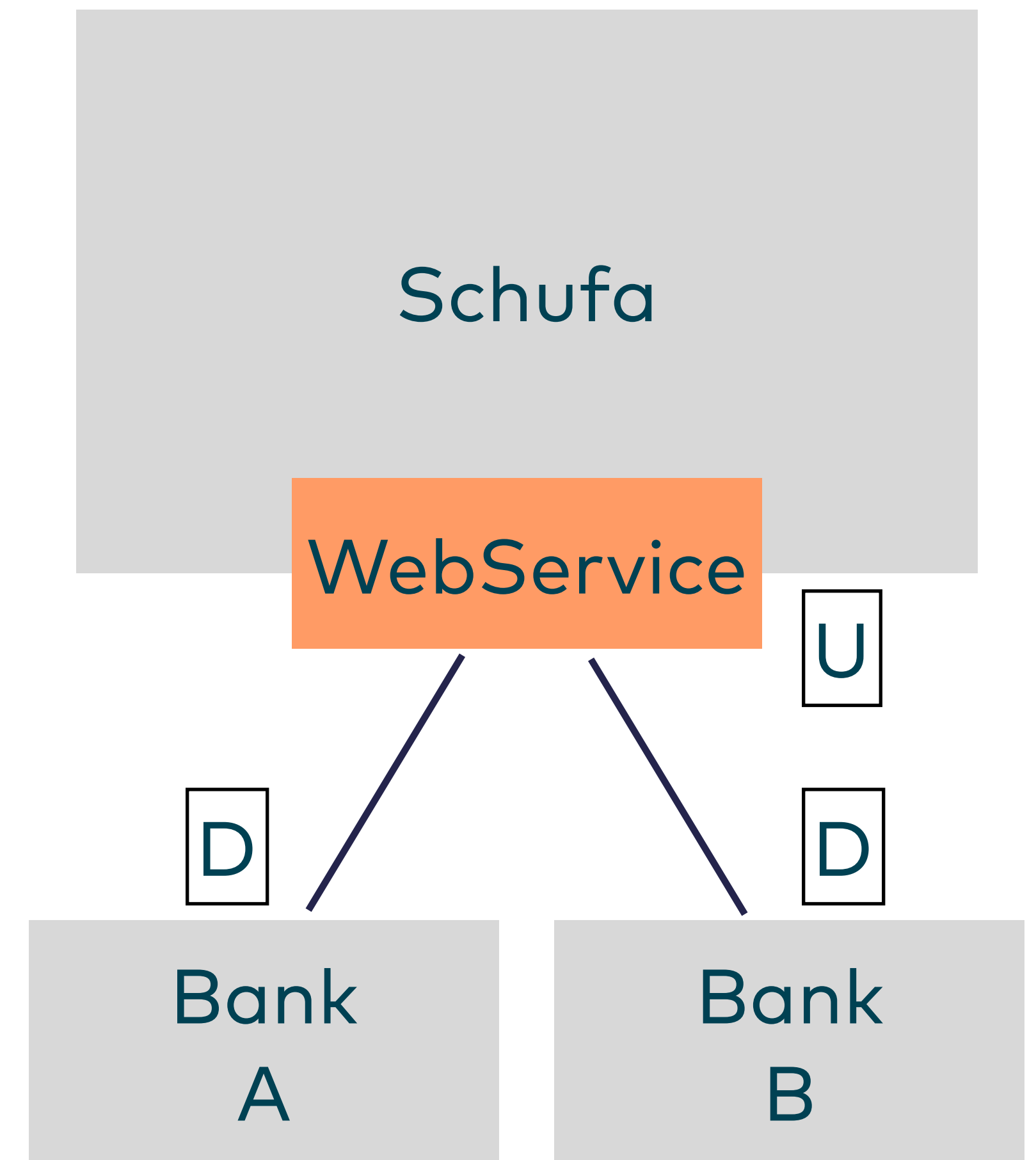


I'll just mention a few of the patterns here which we will later pick up for the combination with Team Topologies.

Open-host Service

The Open-host Service is a public API

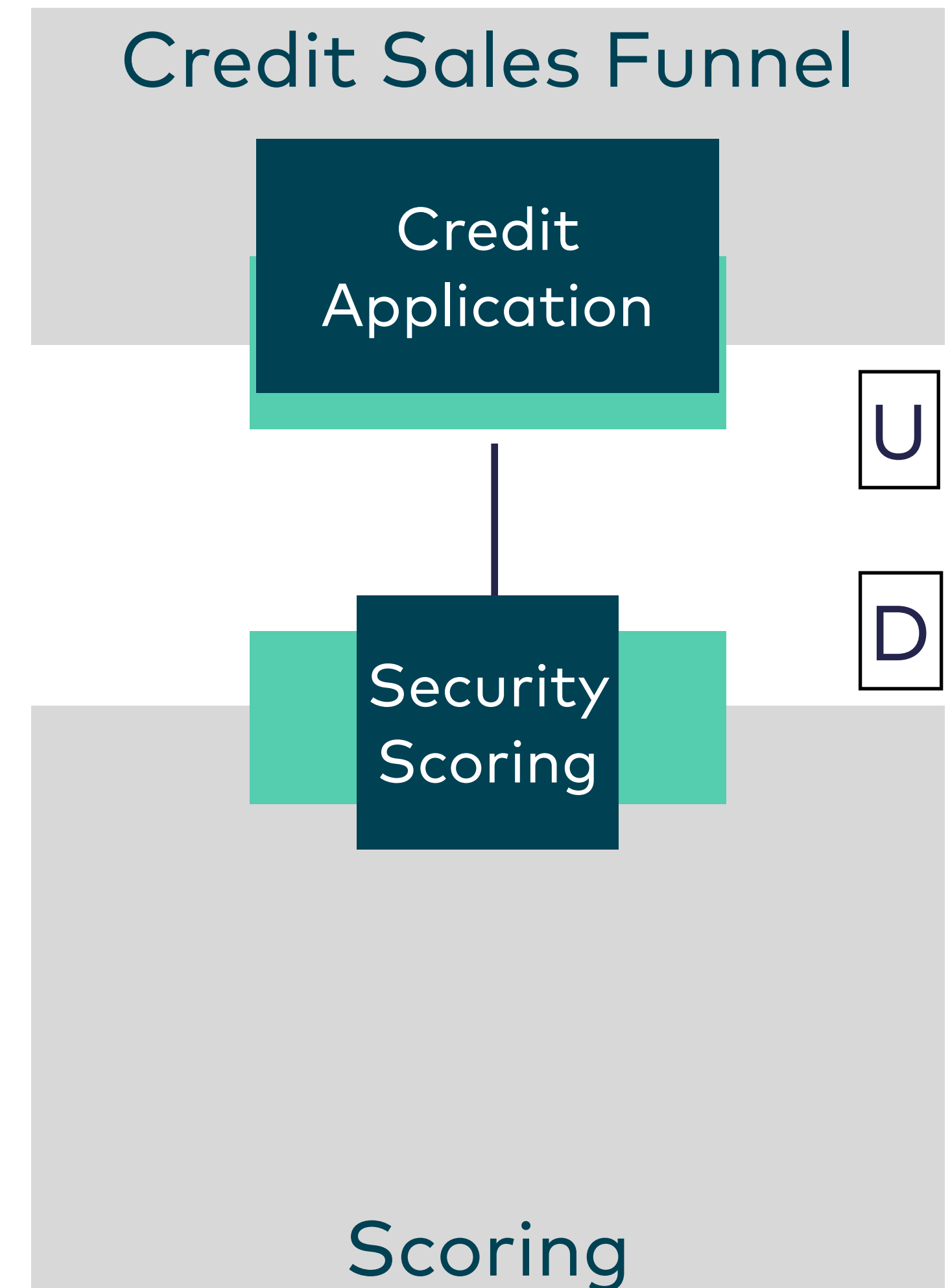
- One API for several consumers
- No point-to-point API
- Has a common, general purpose model and functionality
- The team providing the Open-host Service is an upstream team



Anticorruption Layer

The Anticorruption Layer translates one model to another one

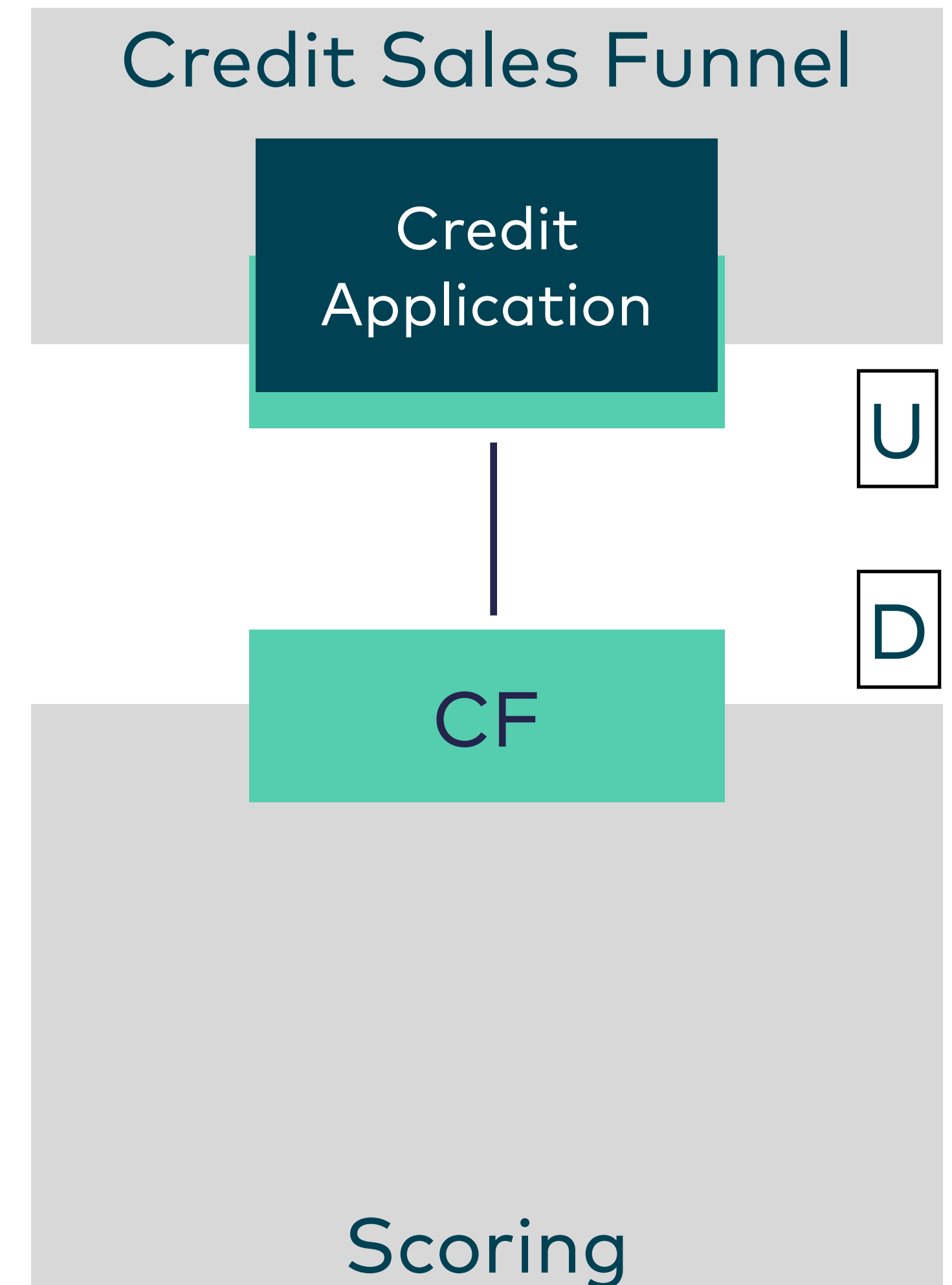
- Transforms an external model from another team / bounded context / system to another internal one
- Reduces the amount of coupling to a single layer
- The team implementing an Anticorruption Layer is always downstream



Conformist

The Conformist slavishly adheres to the upstream model

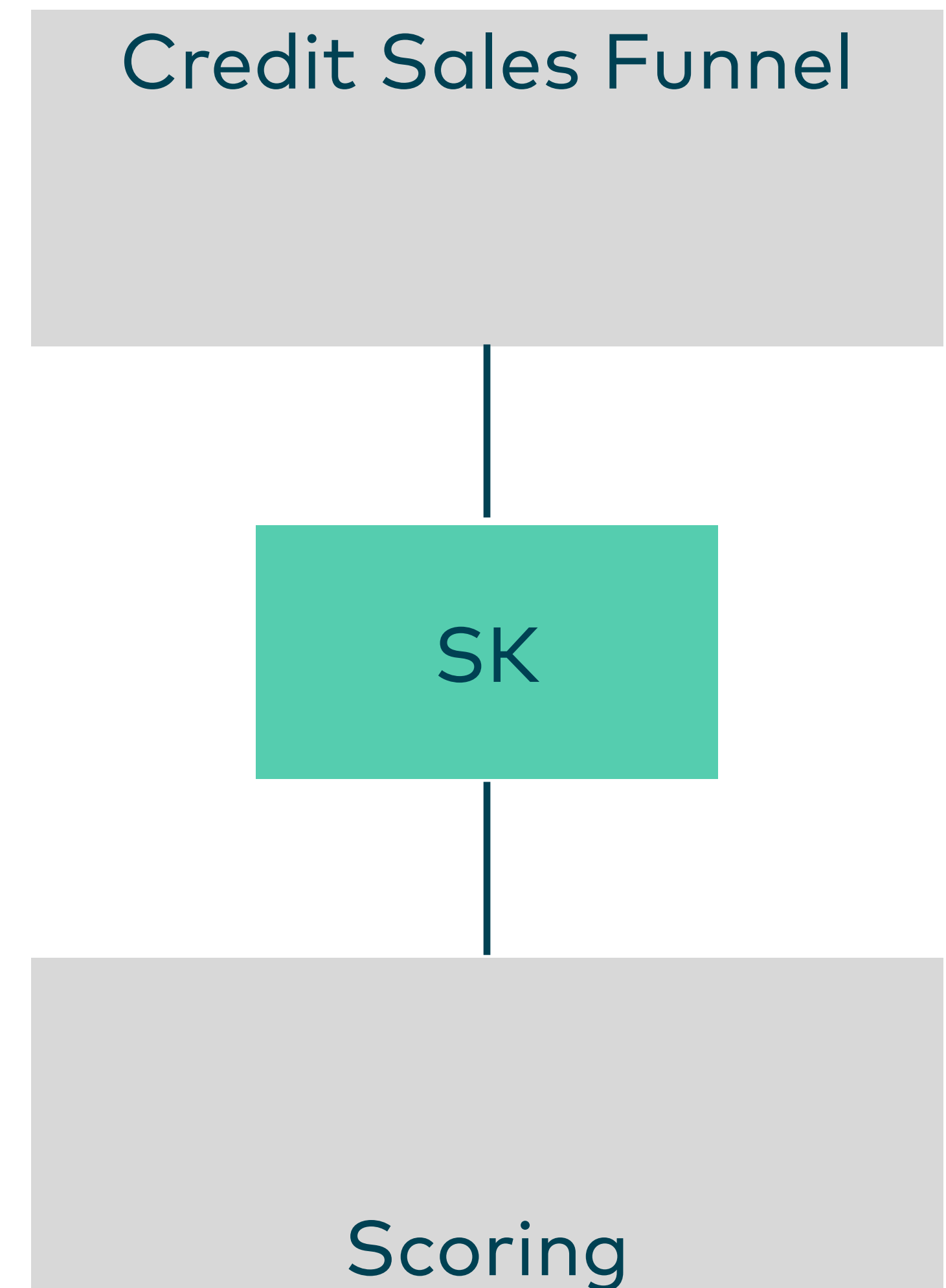
- There is no model-to-model transformation
- Motivation: Simplicity, contracts, force or delight (for the upstream model)
- The team implementing a Conformist is always downstream



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



Partnership

Partnership is about cooperative relationships between teams

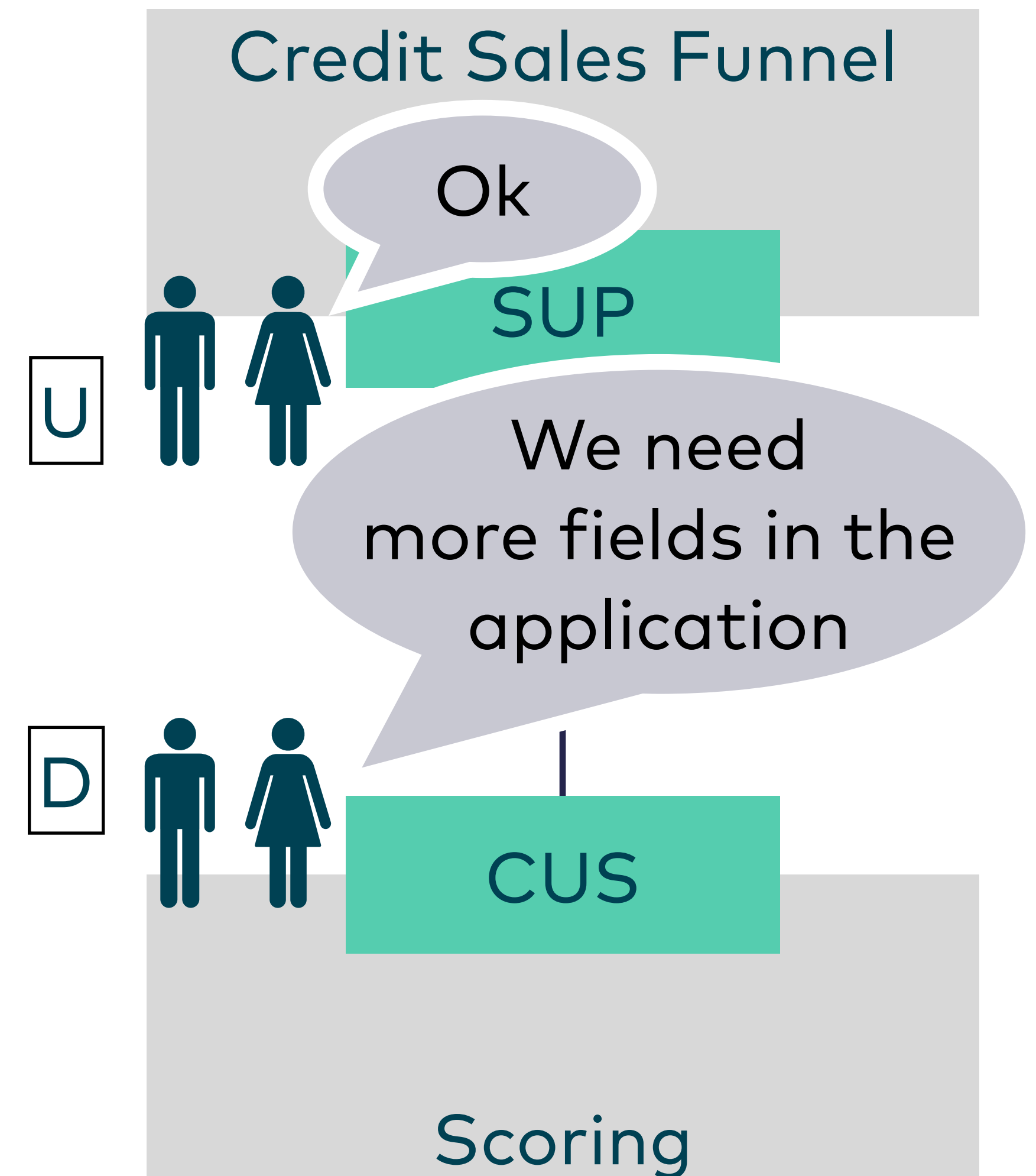
- Establishes a process for coordinated planning of development and joint management of integration
- Not technical at all, Partnership is plain organizational
- Recommended for teams which depend on a Shared Kernel



Customer-Supplier

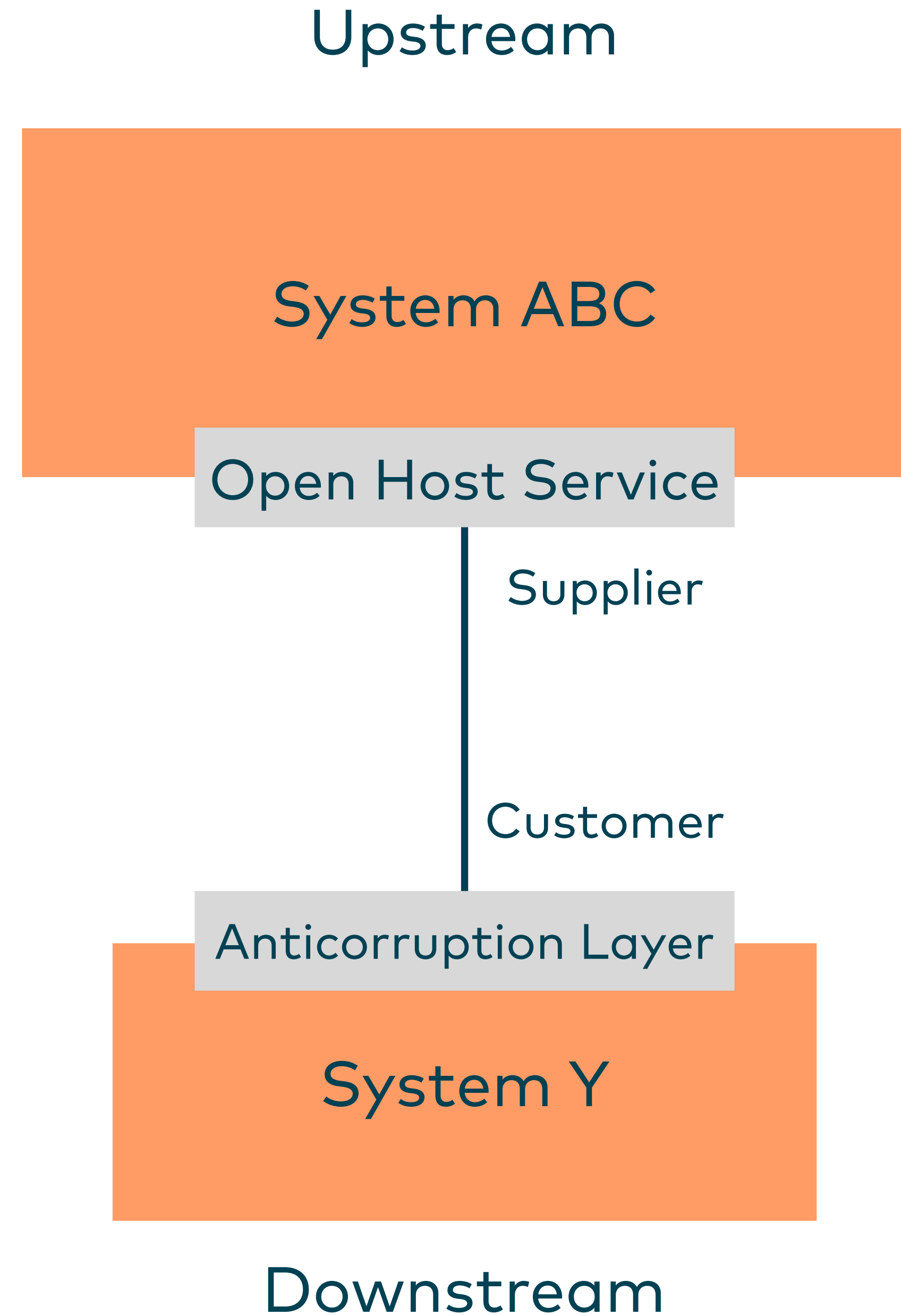
A Customer-Supplier development gives the downstream team some influence

- 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



You can visualize different perspectives

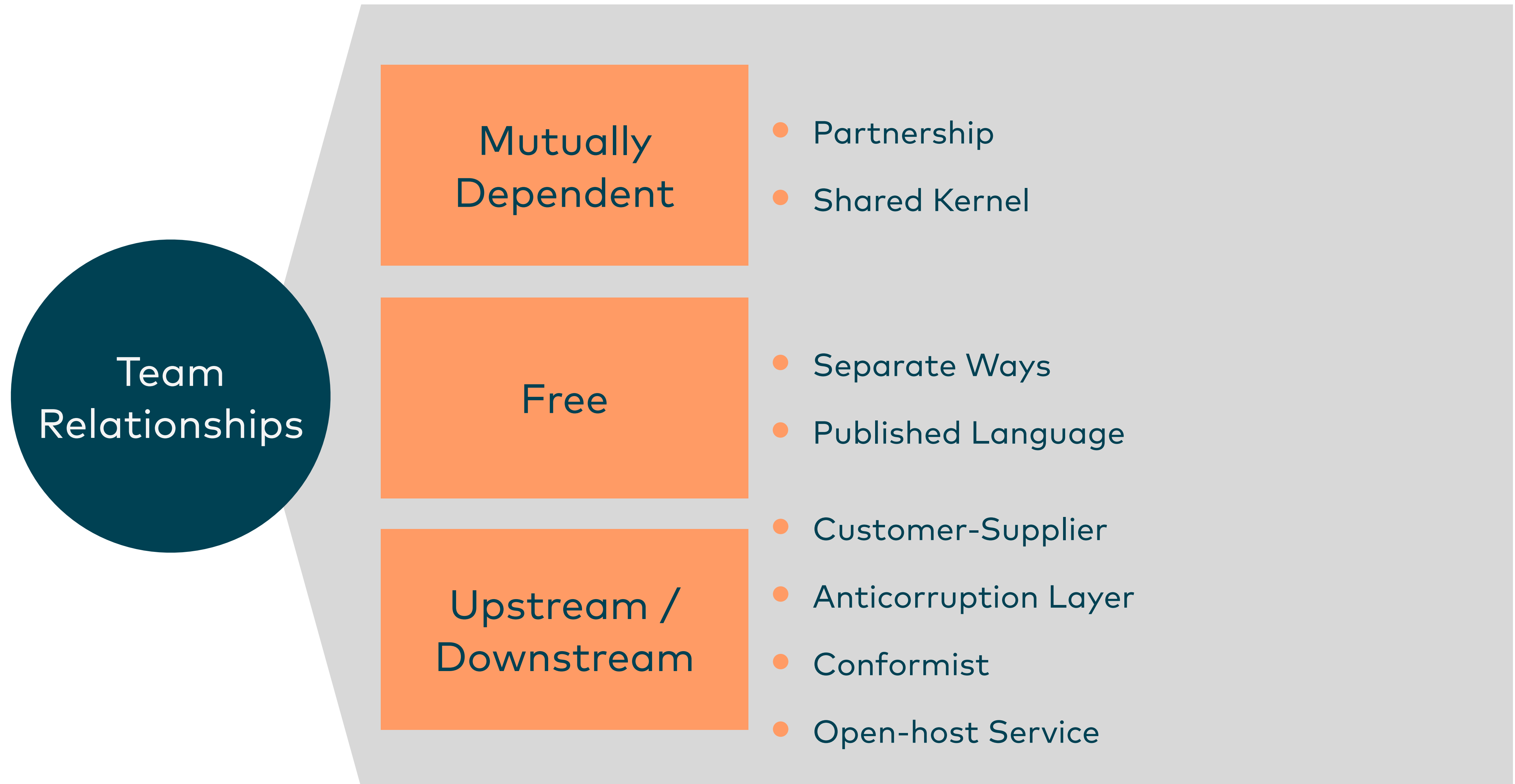
- Call Relationship
- Team Relationship - Level 1
- API Level
- Model Propagation
- Team Relationship - Level 2

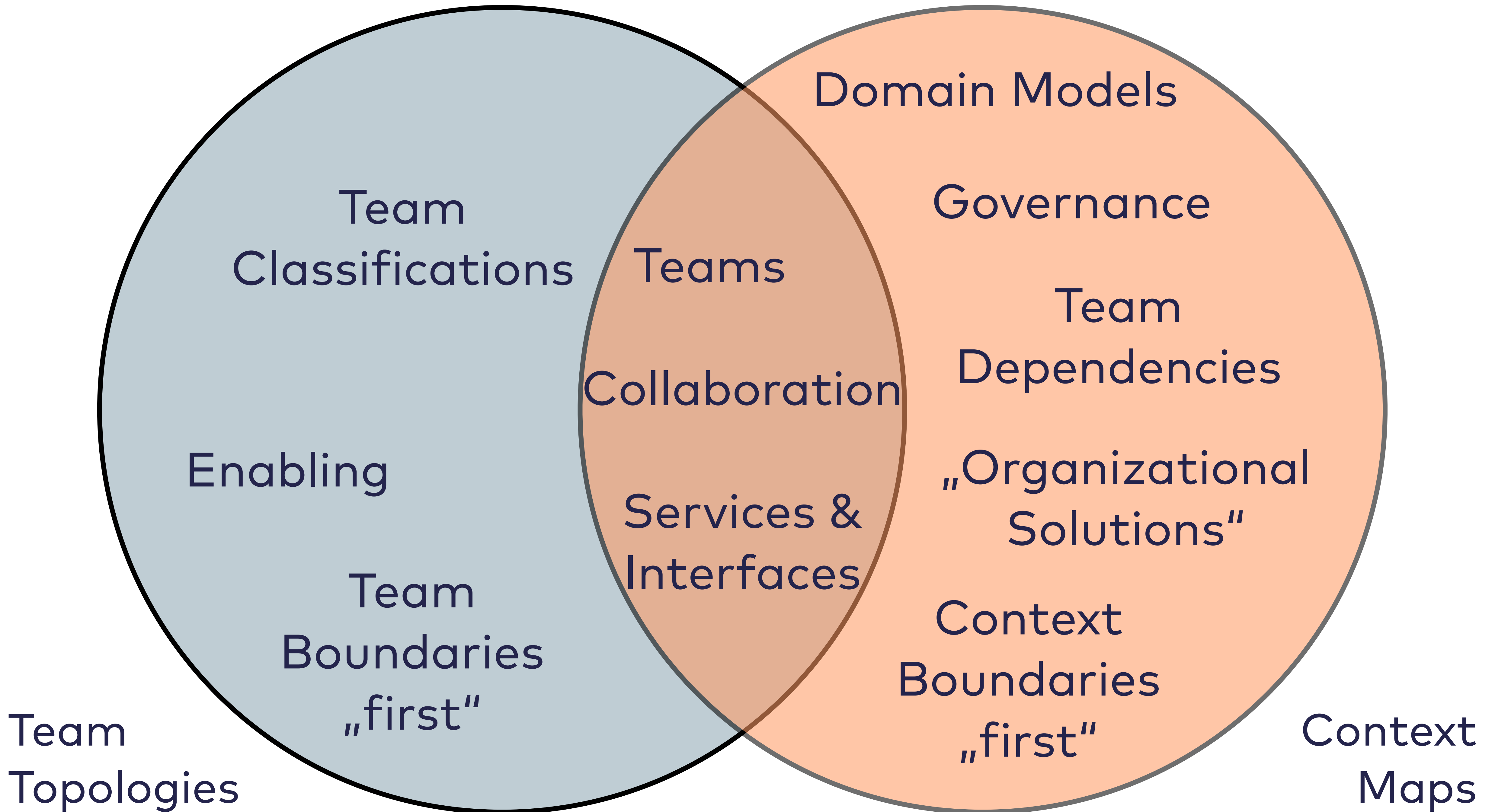


The patterns address various aspects

	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 map to team dependencies

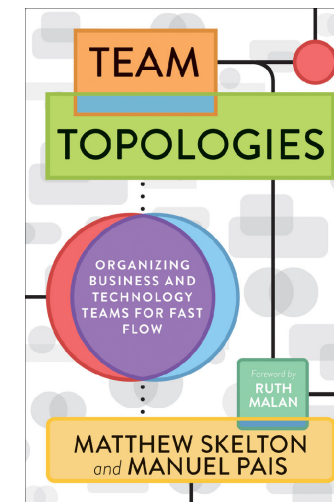




YES

you can combine Team Topologies and Context Maps

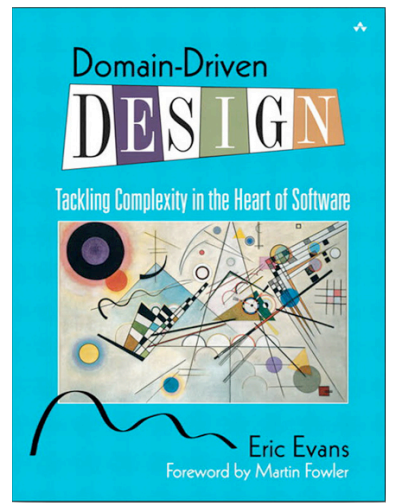
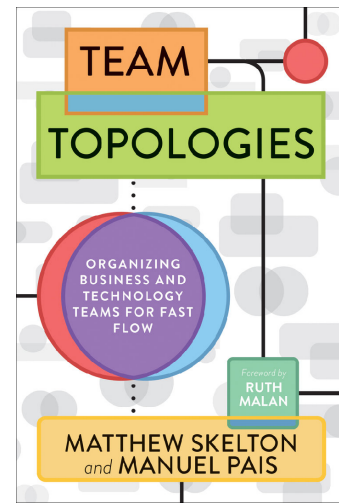
How „aligned“ are stream-aligned teams?



Stream-aligned team

How „aligned“ are stream-aligned teams?

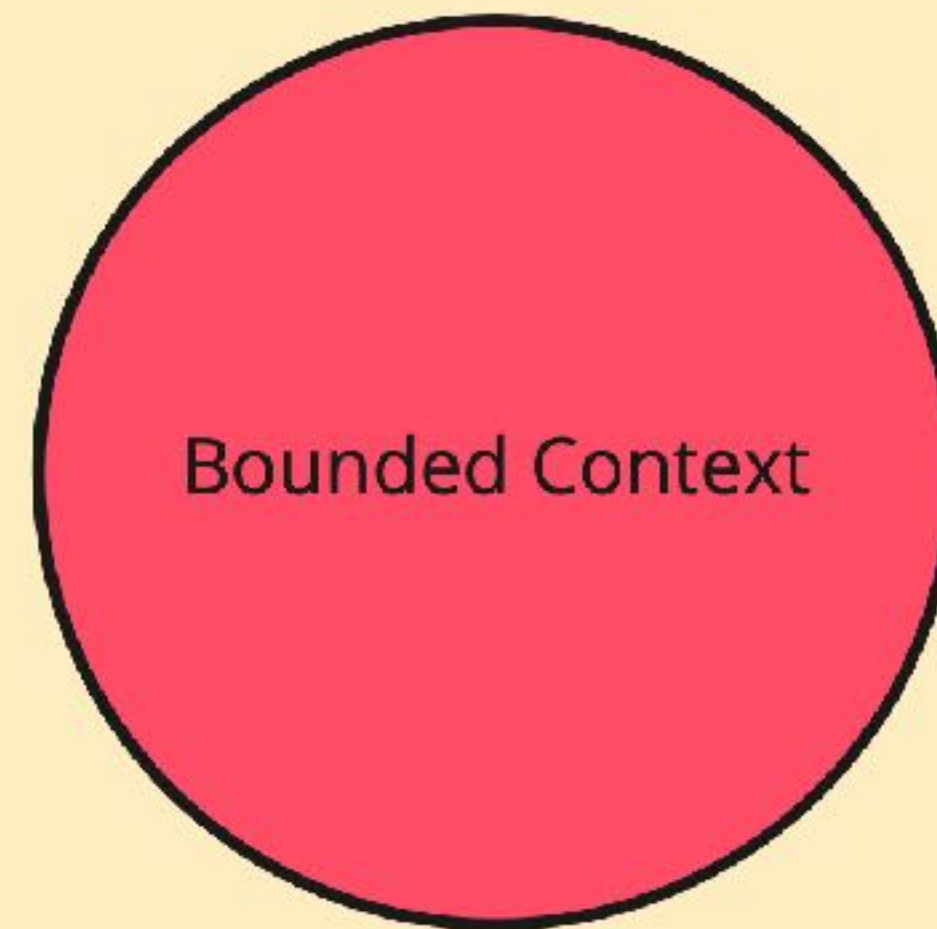
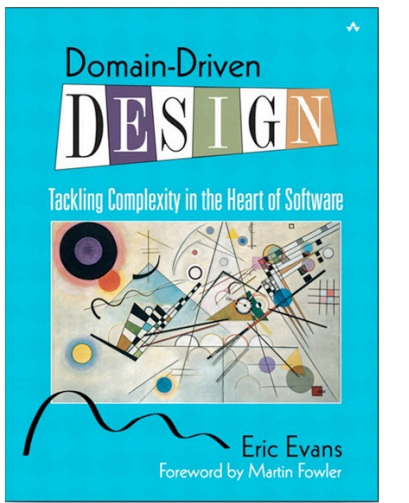
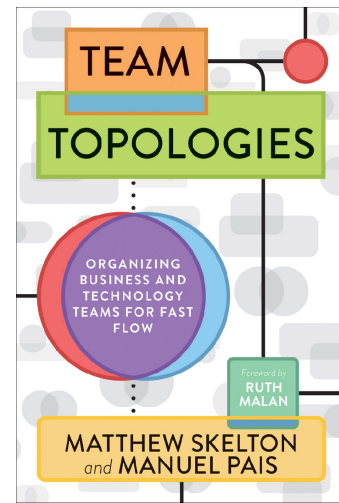
Example: not so aligned



Stream-aligned team

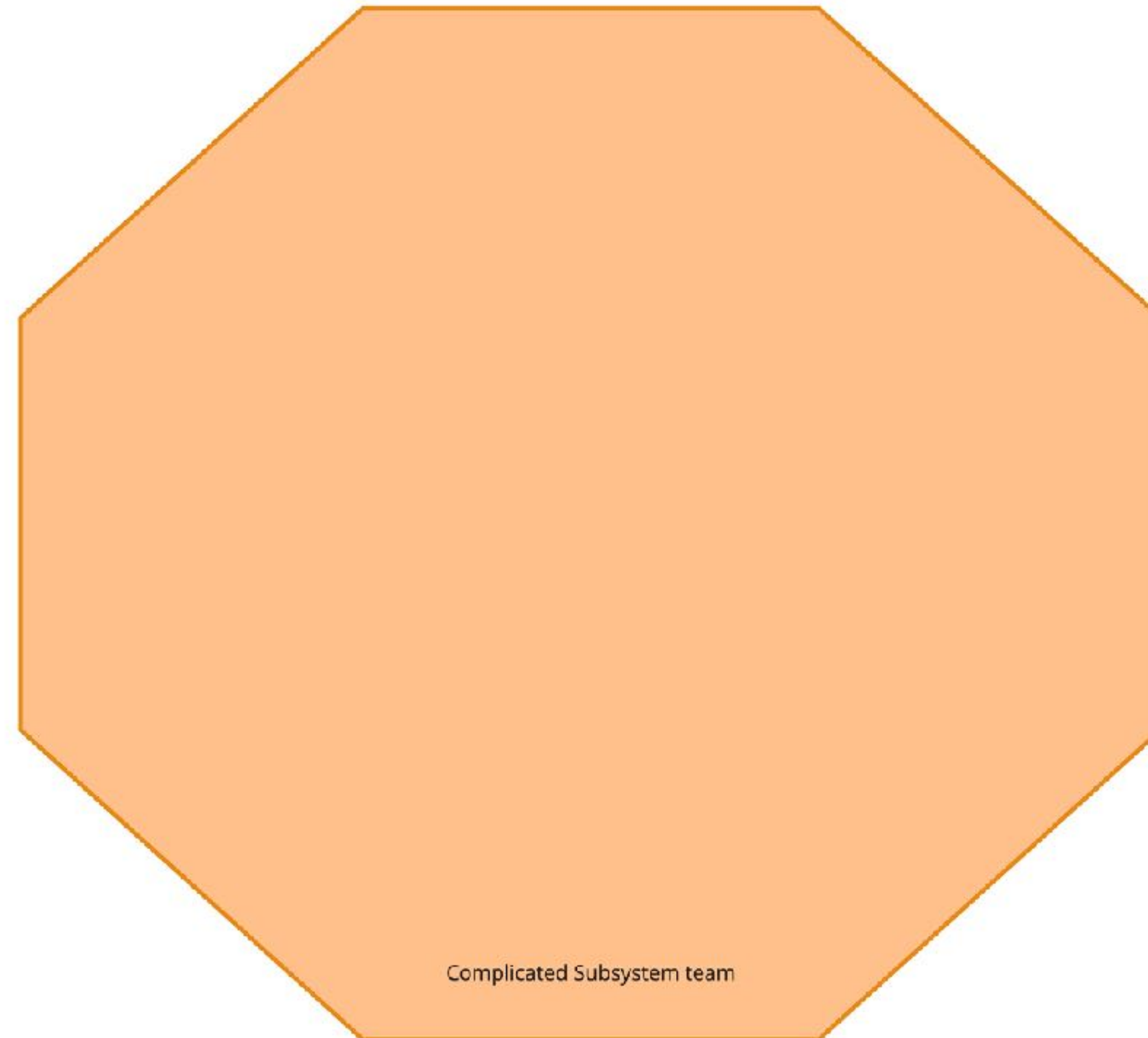
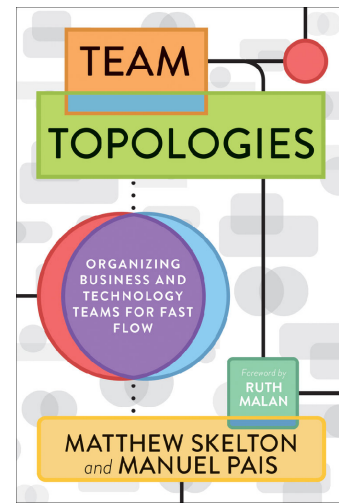
How „aligned“ are stream-aligned teams?

Example: aligned

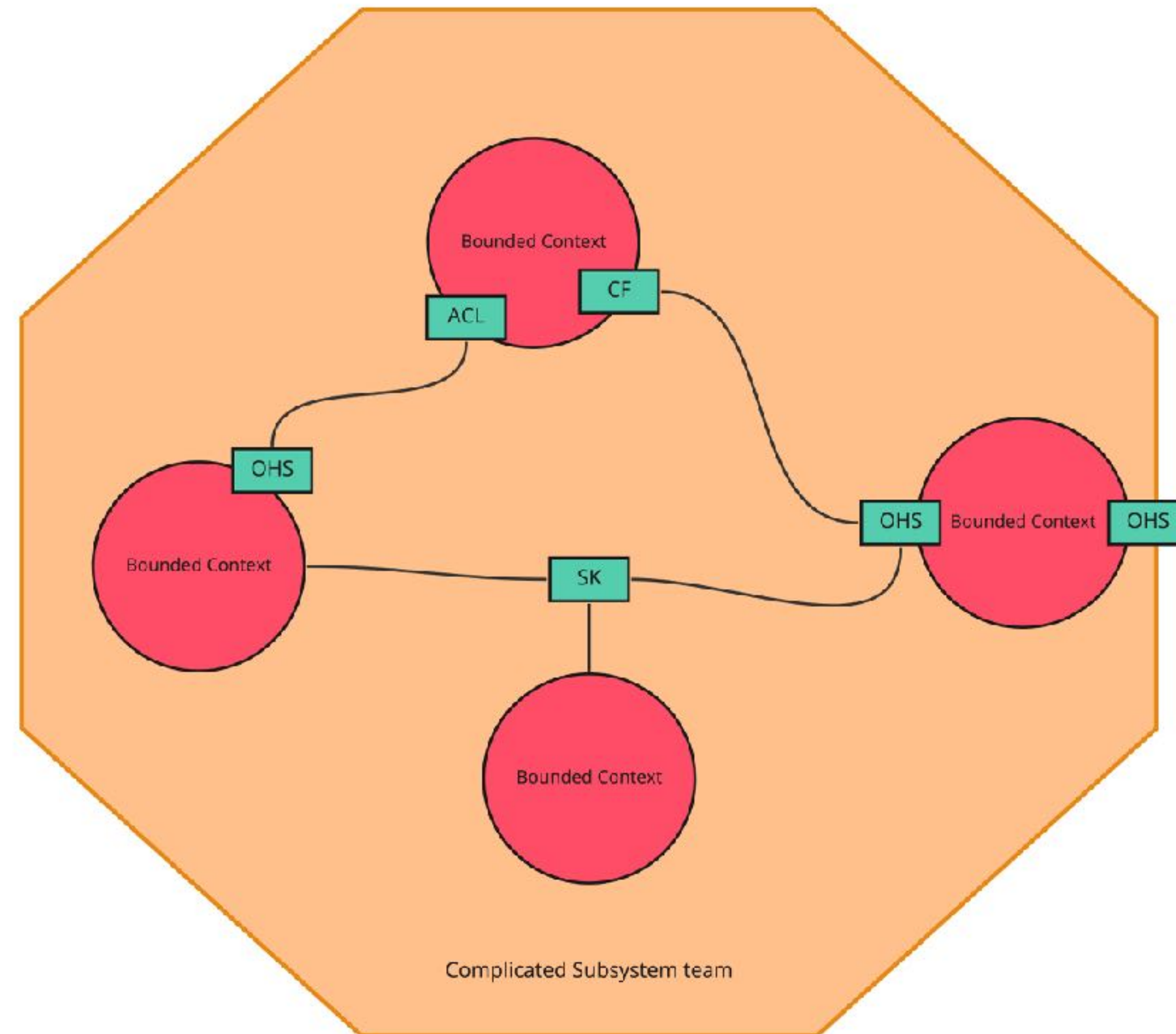
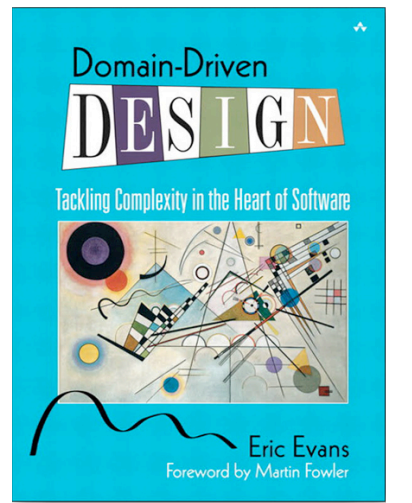
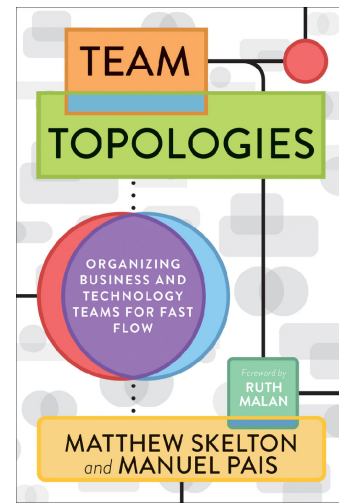


Stream-aligned team

How „complicated“ is the responsibility of a complicated subsystem team?



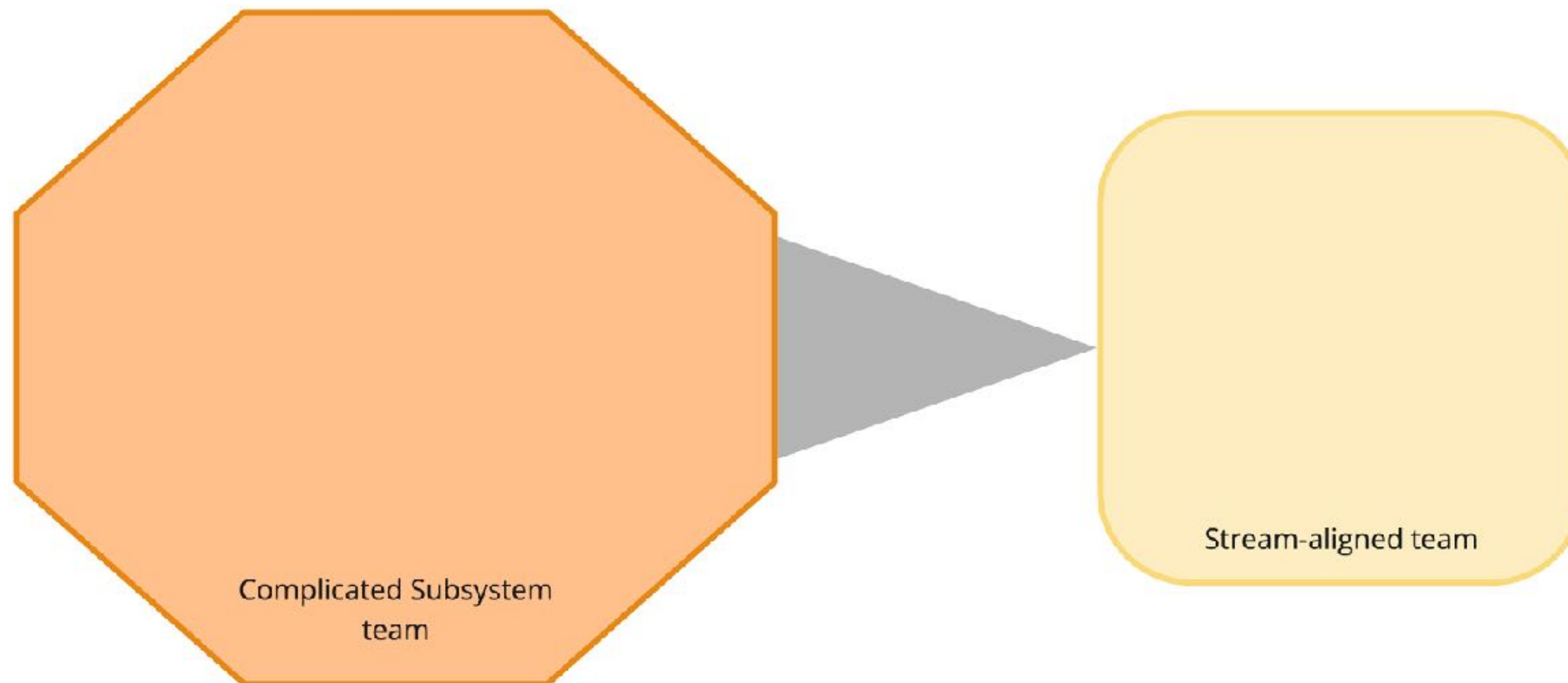
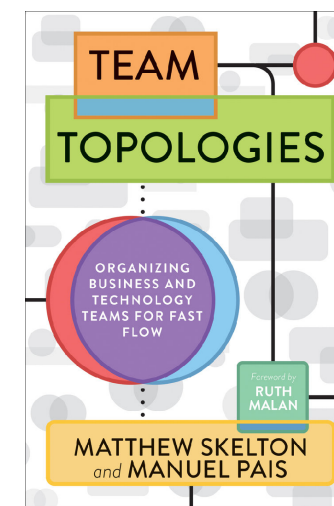
How „complicated“ is the responsibility of a complicated subsystem team?



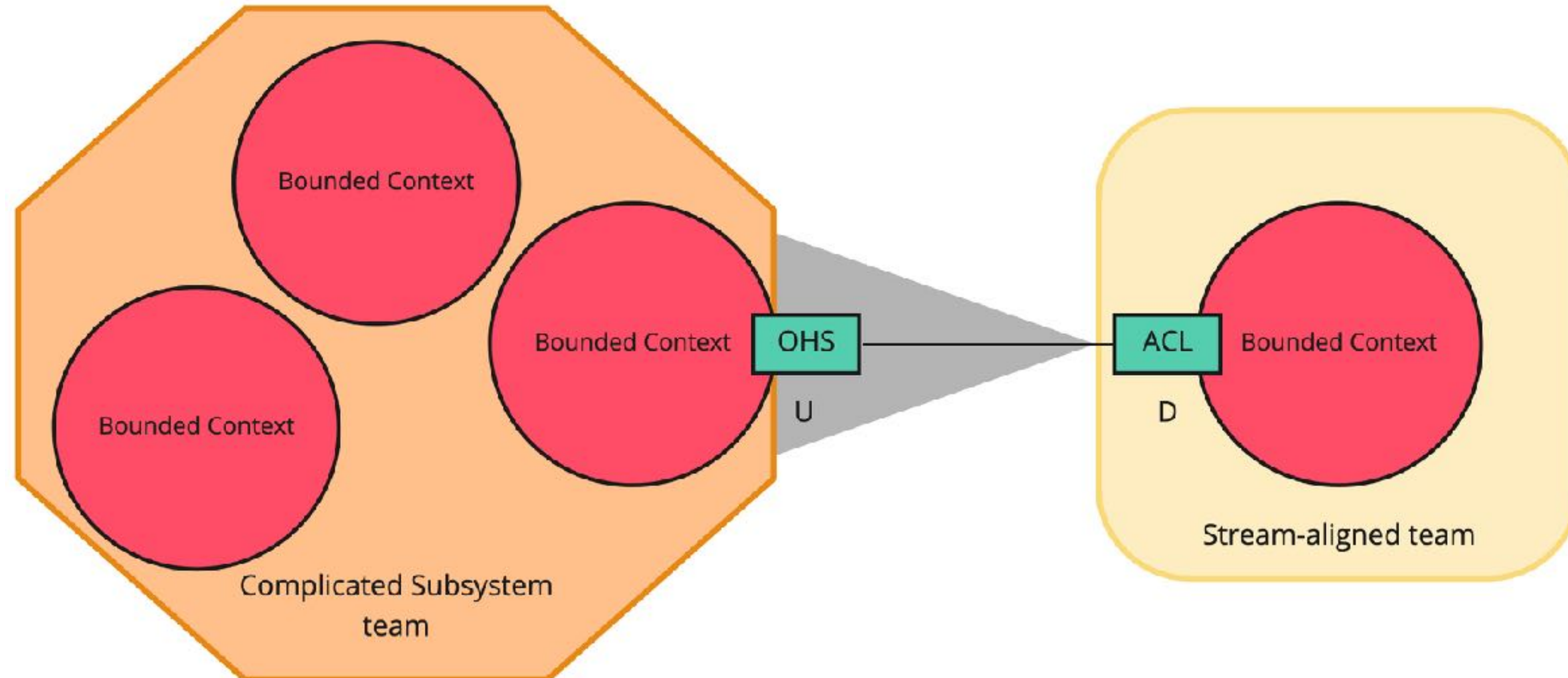
Learning

with context maps we can dig into the boundaries of teams in order to see how they are mapped to their internal responsibilities / software boundaries and if this suits the type of team (stream-aligned, ...)

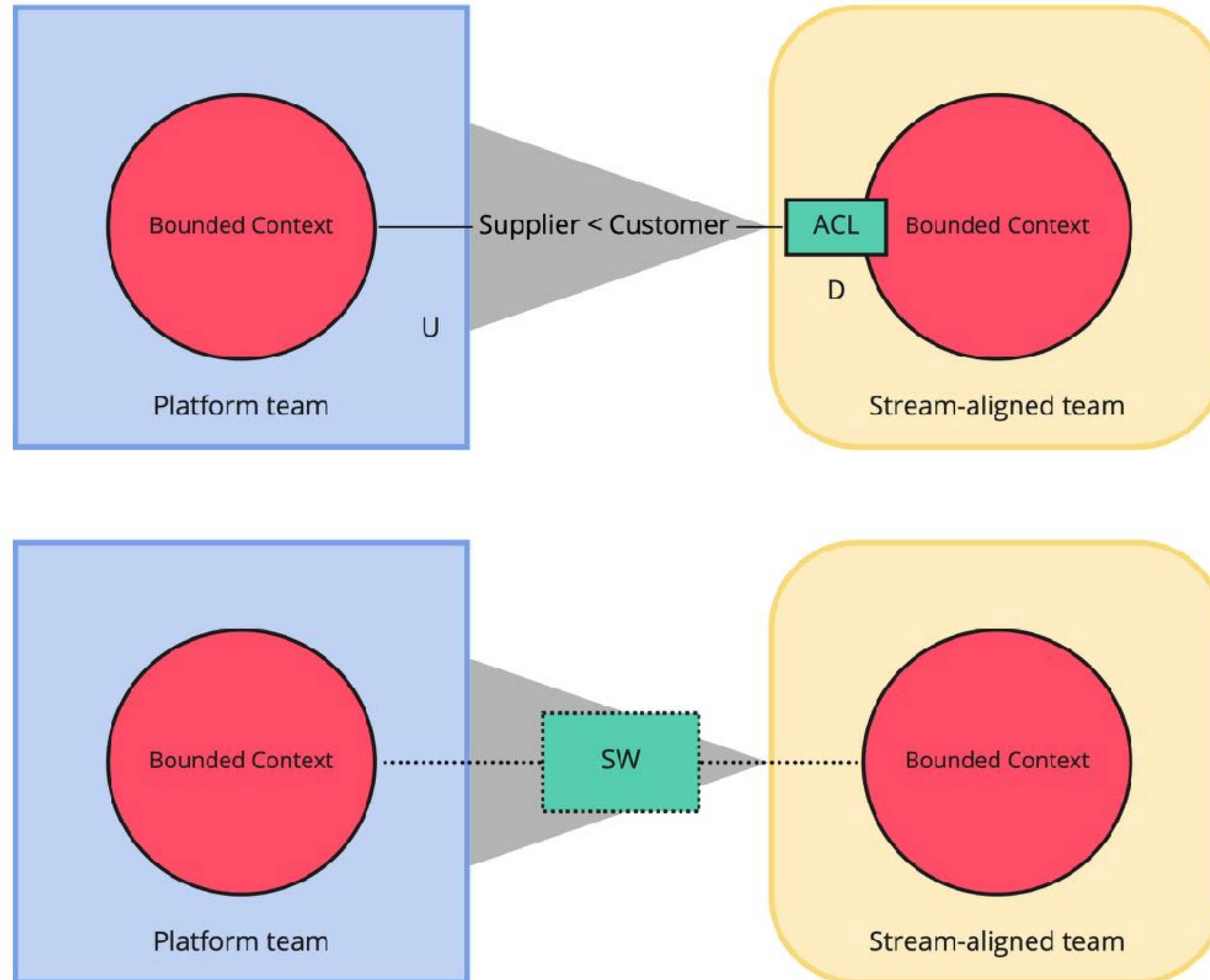
A Complicated Subsystem Team providing a service (X-aa-S) to a Stream-aligned team



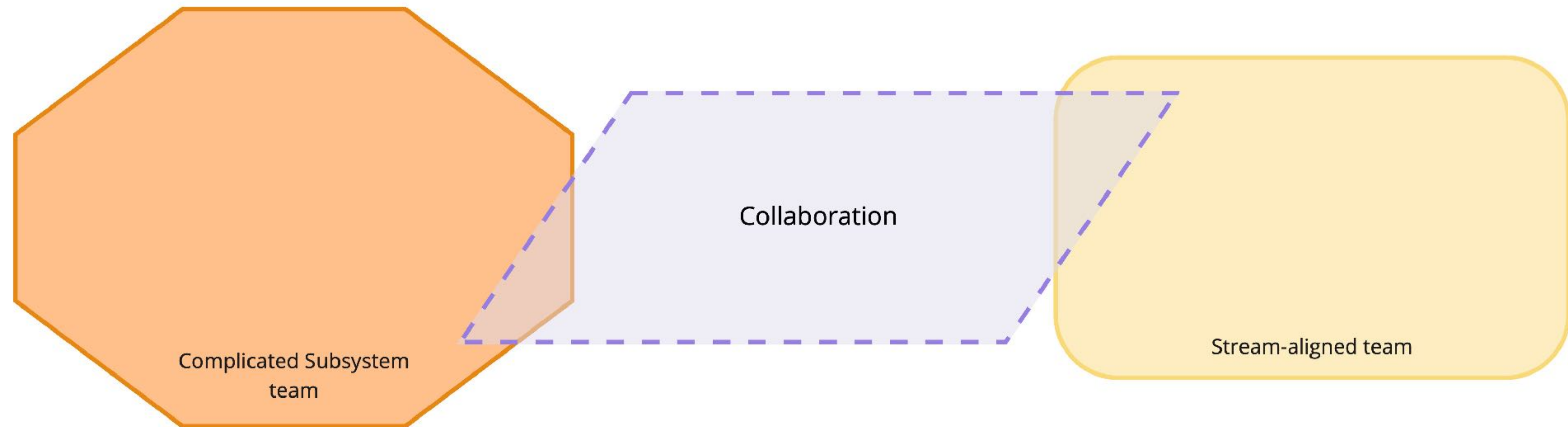
Let's dig deeper into this relationship with DDD's Context Maps



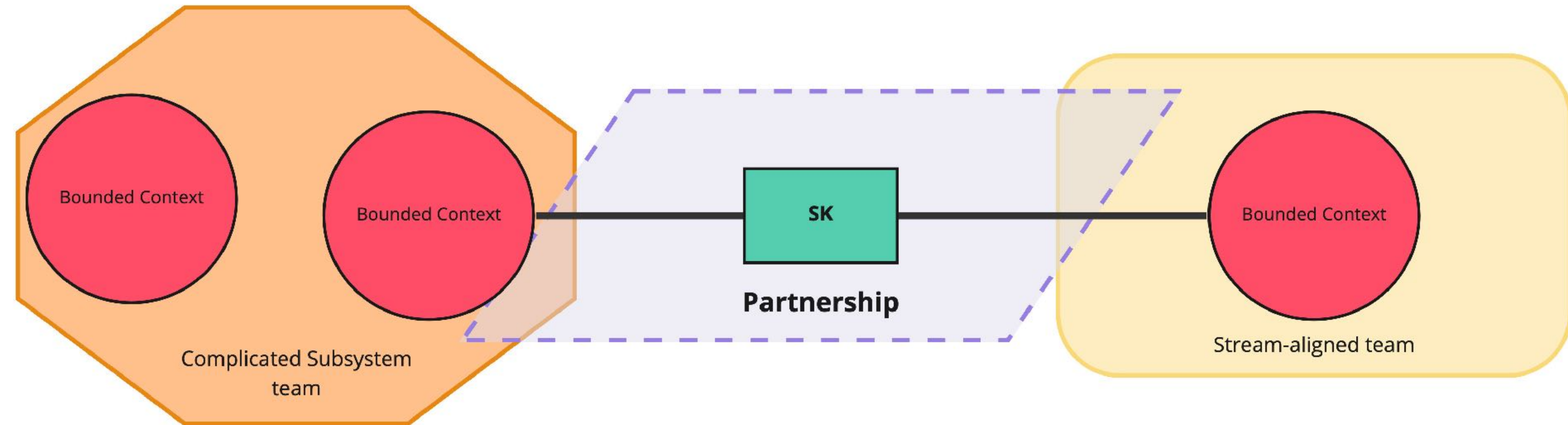
Other examples



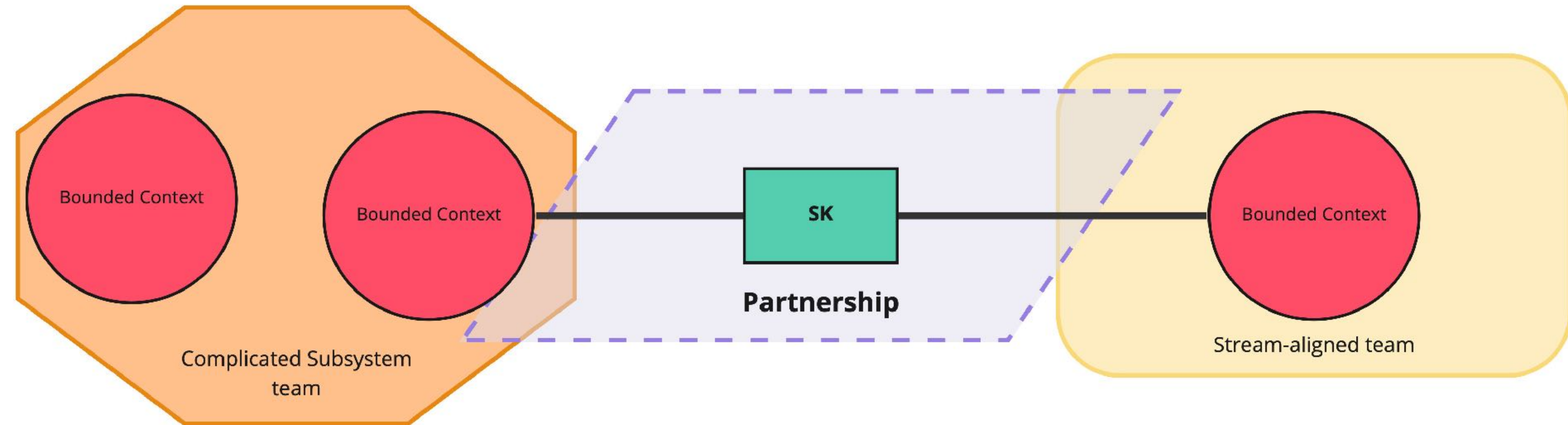
How does a Team Topologies collaboration look like in detail?



Let's drill down into the collaboration and detect something really ugly

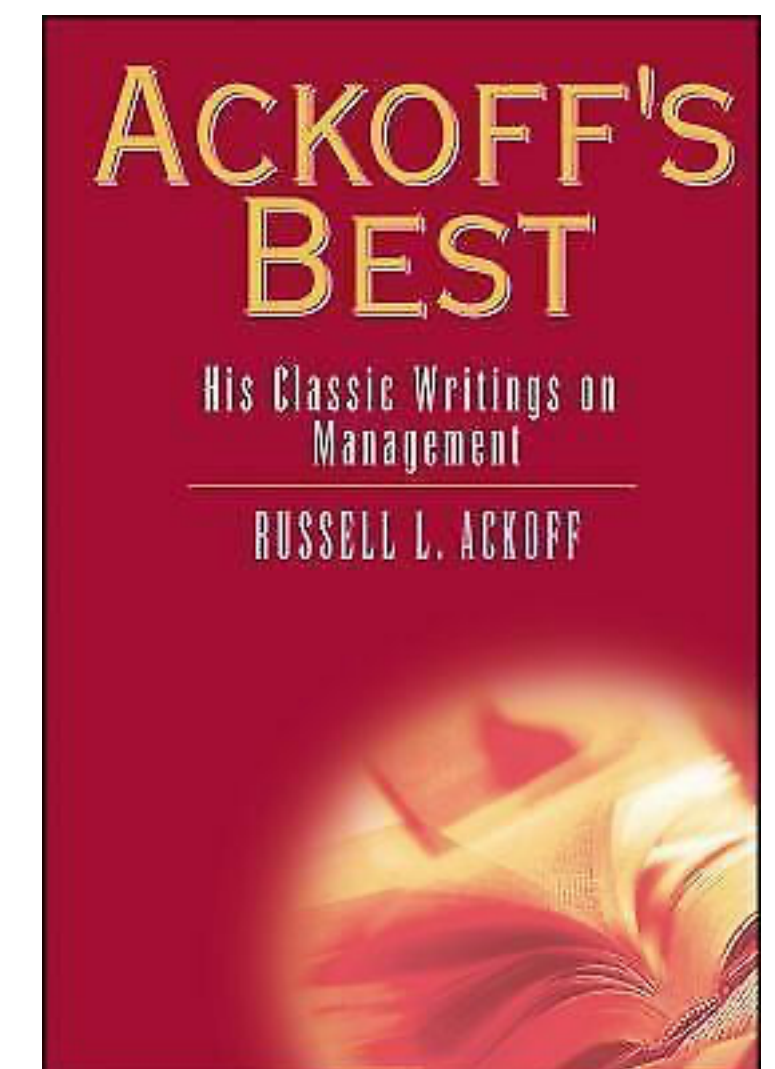
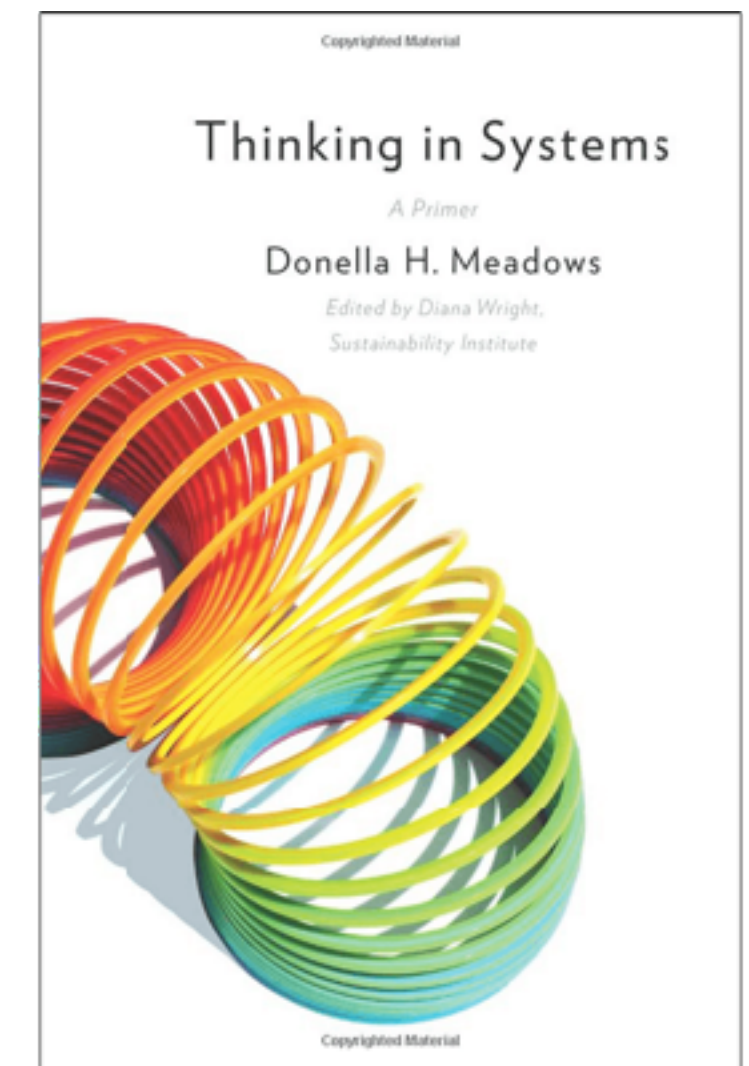


YOU DON'T WANT THIS!

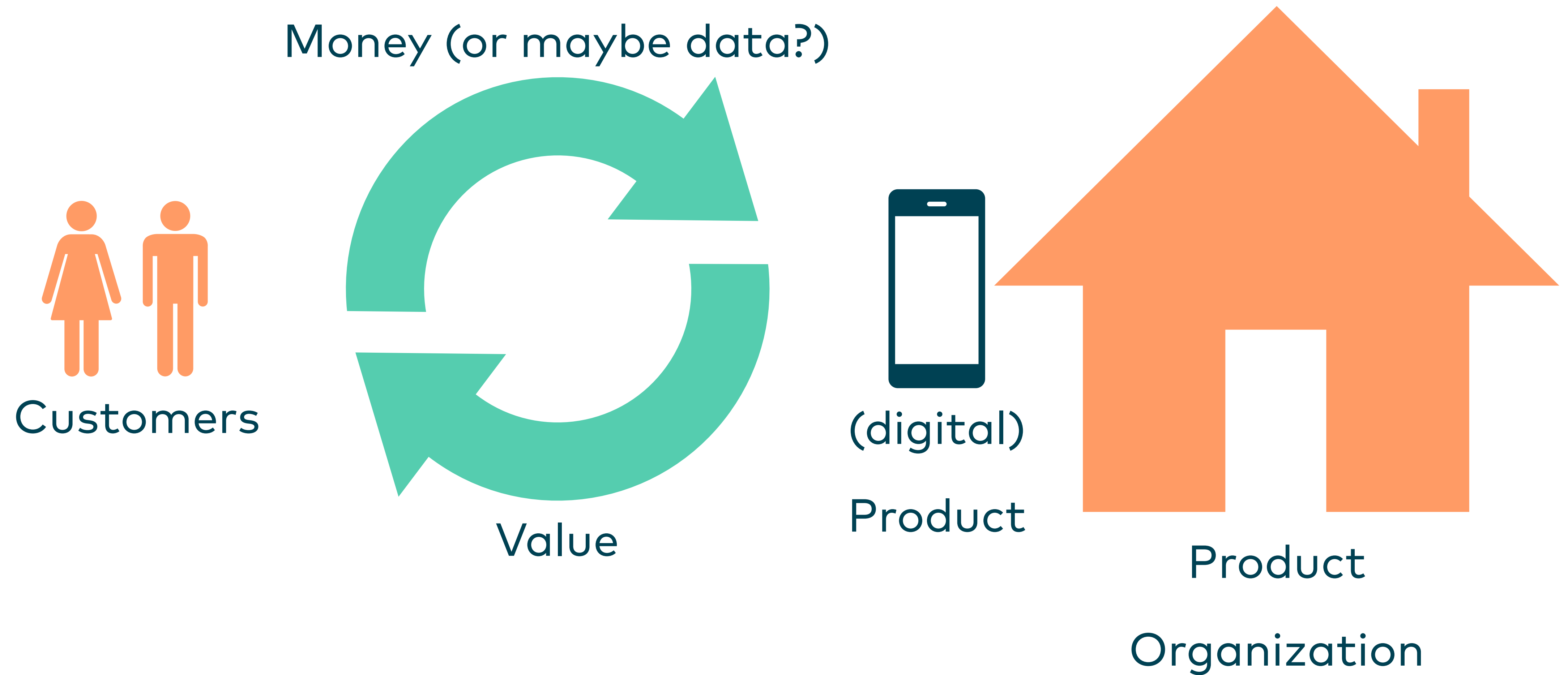


From a **Systems Thinking** perspective which aims at understanding a system as a whole **combining Team Topologies with Context Maps** makes sense

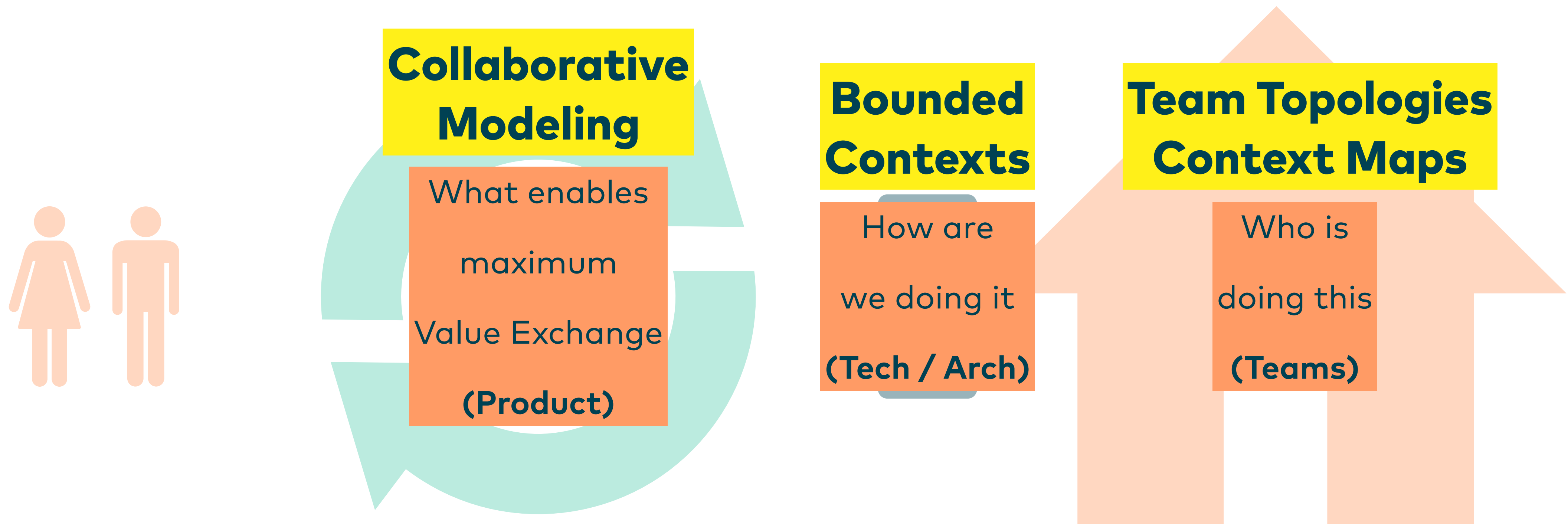
- Team Topologies give us a great starting point by focussing on teams and their core relationships
- Context Maps allow us to dig deeper into the interactions of those relationships and add another perspective with their focus on Bounded Contexts
- Combining both allows us to really understand a system as a whole



Value Exchange with Customer



Value Exchange with Customer



Thanks!



Michael Plöd

Twitter: @bitboss

LinkedIn: <https://www.linkedin.com/in/michael-ploed/>



Book Voucher: 7.99 instead of (min) 9.99

<http://leanpub.com/ddd-by-example/c/speakerdeck>

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