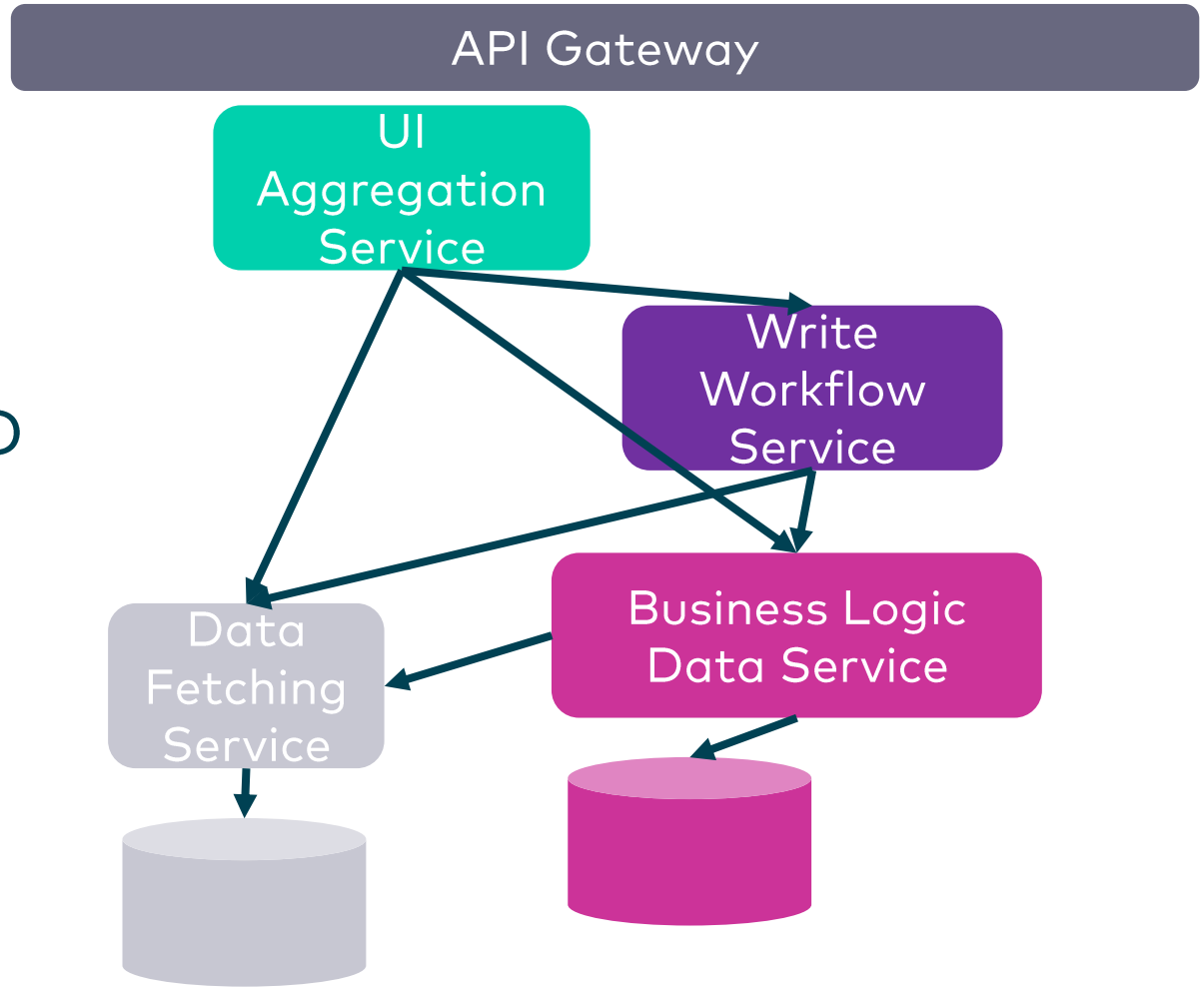
Why You Might Fail with Domain-driven Design

INNOQ

Eberhard Wolff
Fellow @ewolff

Is This a Great Architecture?

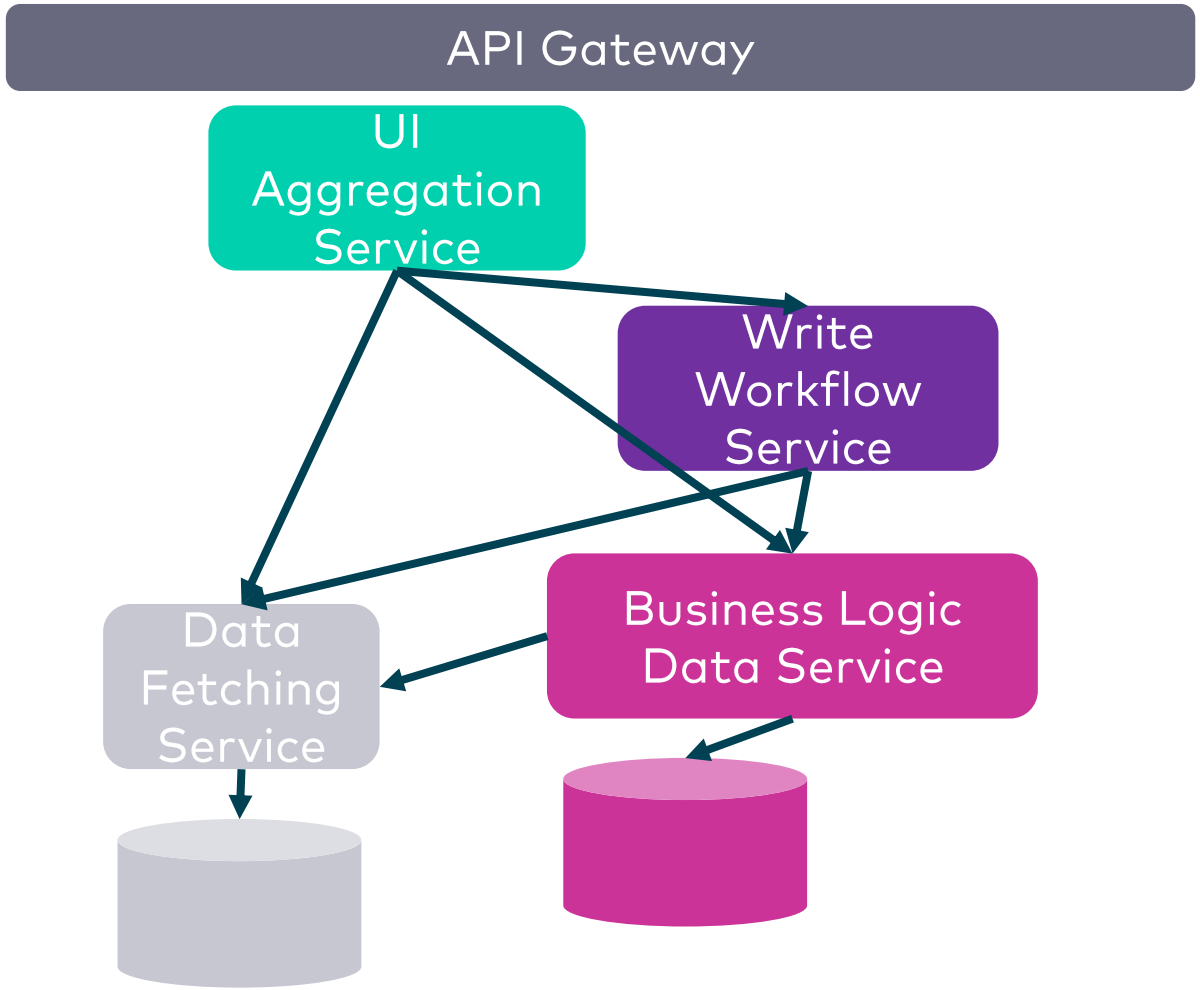
We are using all the tactical DDD pattern like Service, Repository, ...



DDD Domain-driven Design

- Software should provide business value.
- Software should support business processes.
- Typical changes are to business logic.
- Therefore:
  Let the domain drive the design!

What is Even the Domain?



nDDD

DDD vs nDDD

- DDD Domain-driven Design

Domain drives the design

- nDDD Non-domain-driven Design

Something else drives the design

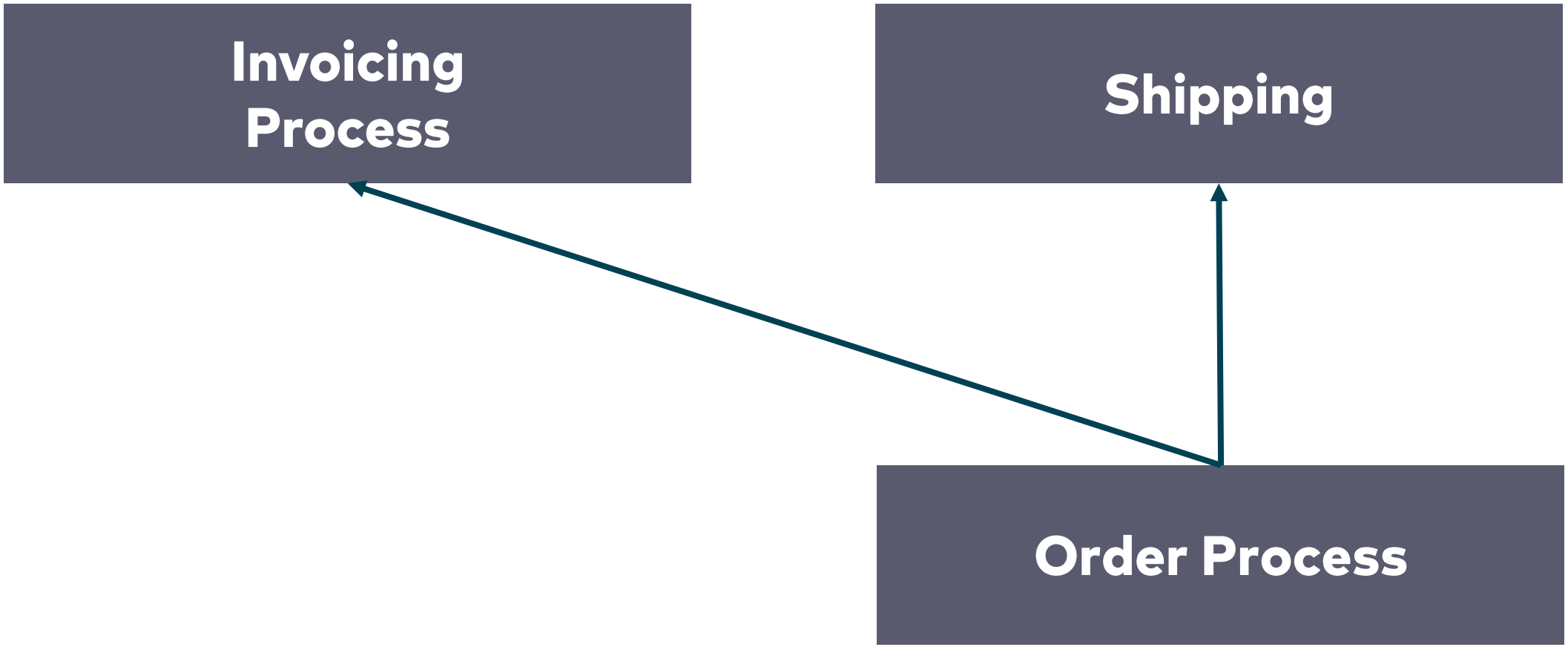
How to Detect nDDD

- Can you tell which domain the architecture is for?
- Can you use the architecture for a self-driving car or a video game?
- My experience:
  Technical architecture much too common

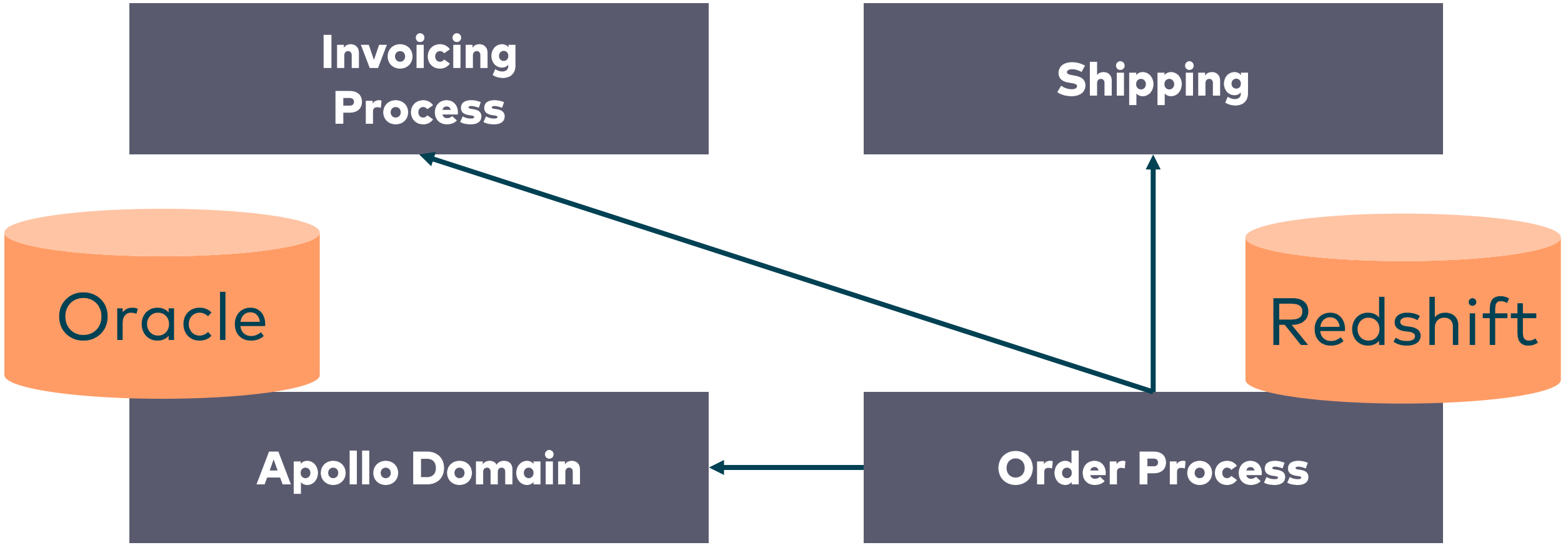
Would you rather show / discuss something technical or business-related if asked for the architecture?

Usually, I'm presented with technical architecture diagrams.

Better



Even Worse: Tech + Business Chaos



DDD vs nDDD

- Can the team execute the business process the application implements?
- When was the last time the team talked to a user / customer?
- Can you explain the business purpose of the application to your partner?
- How does the architecture structure the business logic?

Why would I care?

There are requirements, right?

Domain-driven Design

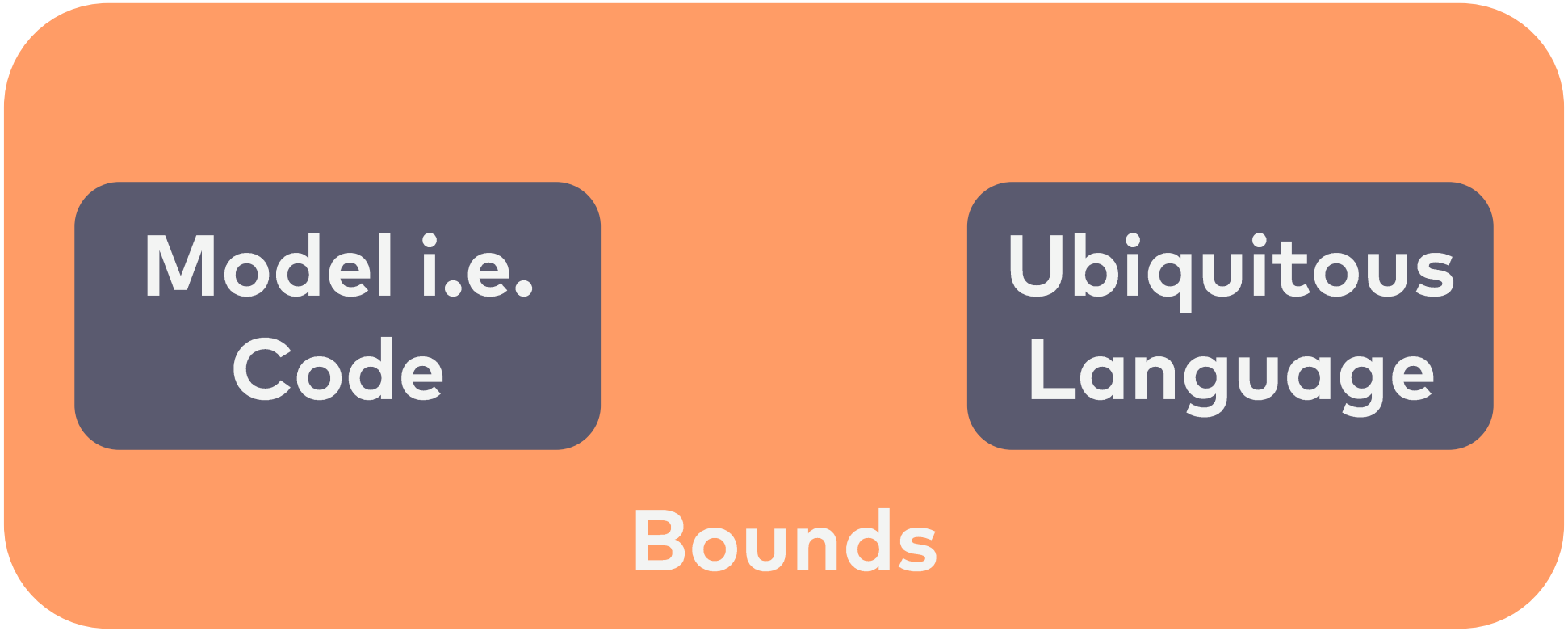
- Domain-driven Design:
  software should structure domain logic
- DDD's aim is to support the business as well as possible
- So: Must understand the domain

DDD = Collaboration

- Technical people can't define the business purpose by themselves.
- So: Ask & support businesspeople
- Might be hard
- Sometimes, you might fail
- Collaborative Modeling e.g. event storming / domain story telling can help

Bounded Context

Bounded Context



- Usually handled by one team.
- Example: Order process, delivery process

Bounded Context

- Bounded context = module
- No other concept is so poorly understood.

Bounded Context Example

Invoicing Process

Invoicing

VAT

Shipping

Tracking

Delivery

Order Process

Shopping Cart

Accept order

Example Non Bounded Contexts

- Customer
- Product
- Very likely data-driven,
  not domain-driven

Module

Public
Information

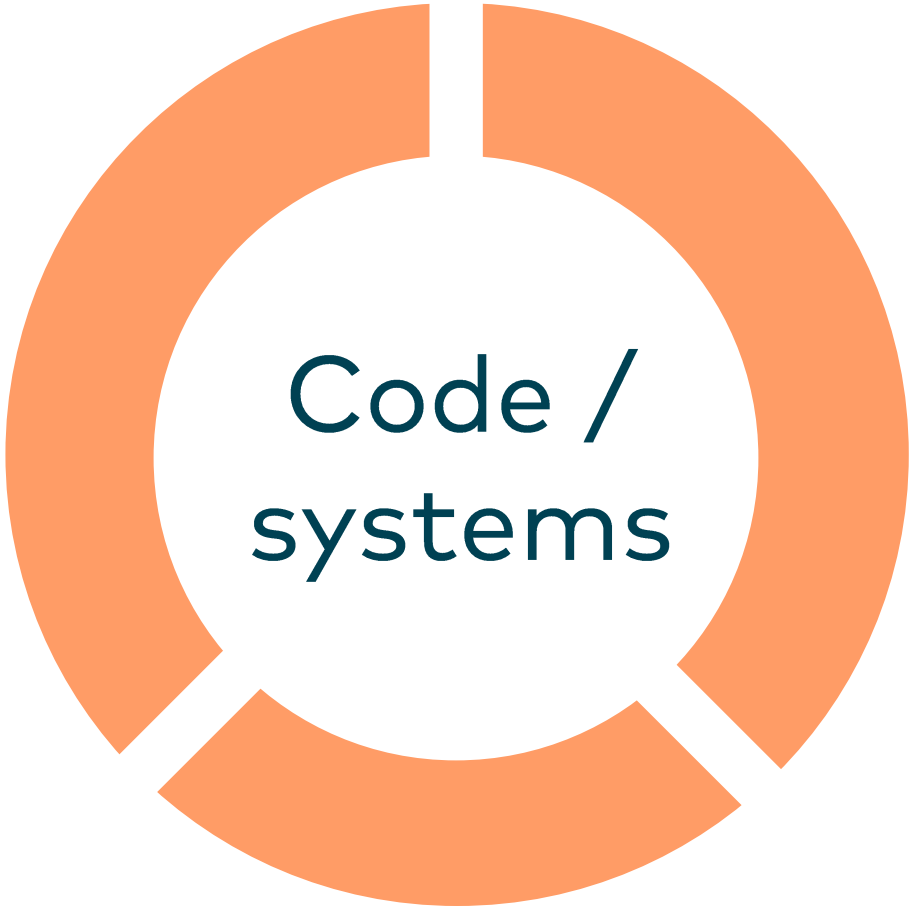
Class

Public
Methods

CRC Cards for Classes

Class Order Service	**Responsibility** Accepting Orders
Collaboration Order Repository Invoice Service Shipment Service Statistics Service	

Bounded Context



Interfaces

Bounded Context Canvas

Name Payment	**Core**	**Description** Processing Payments
Inbound Communication Order Processing	**Ubiquitous Language** Receipt Payment	**Outbound Communication** Payment Provider Book keeping Order Processing

Bounded Context Canvas

Collaboration

Name Payment	**Core**	**Description** Processing Payments
Inbound Communication Order Processing	**Ubiquitous Language** Receipt Payment	**Outbound Communication** Payment Provider Book keeping Order Processing

Bounded Context Canvas

Responsibility

Name Payment	**Core**	**Description** Processing Payments

Inbound Communication

Order Processing

Ubiquitous Language

Receipt
Payment

Outbound Communication

Payment Provider
Book keeping
Order Processing

BUILD MODULES BY FUNCTIONALITY NOT DATA!

Seriously:

BUILD MODULES BY FUNCTIONALITY NOT DATA!

CRC Cards for Classes: No Data!

Class Order Service	**Responsibility** Accepting Orders
Collaboration Order Repository Invoice Service Shipment Service Statistics Service	

Bounded Context Canvas: No Data!

Name Payment	**Core**	**Desciption** Processing Payments
Inbound Communication Order Processing	**Ubiquitous Language** Receipt Payment	**Outbound Communication** Payment Provider Book keeping Order Processing

Bounded Context Example

Invoicing Process

Customer e.g. billing address

Product e.g. price

Shipping

Customer e.g. shipping address

Product e.g. size

Order Process

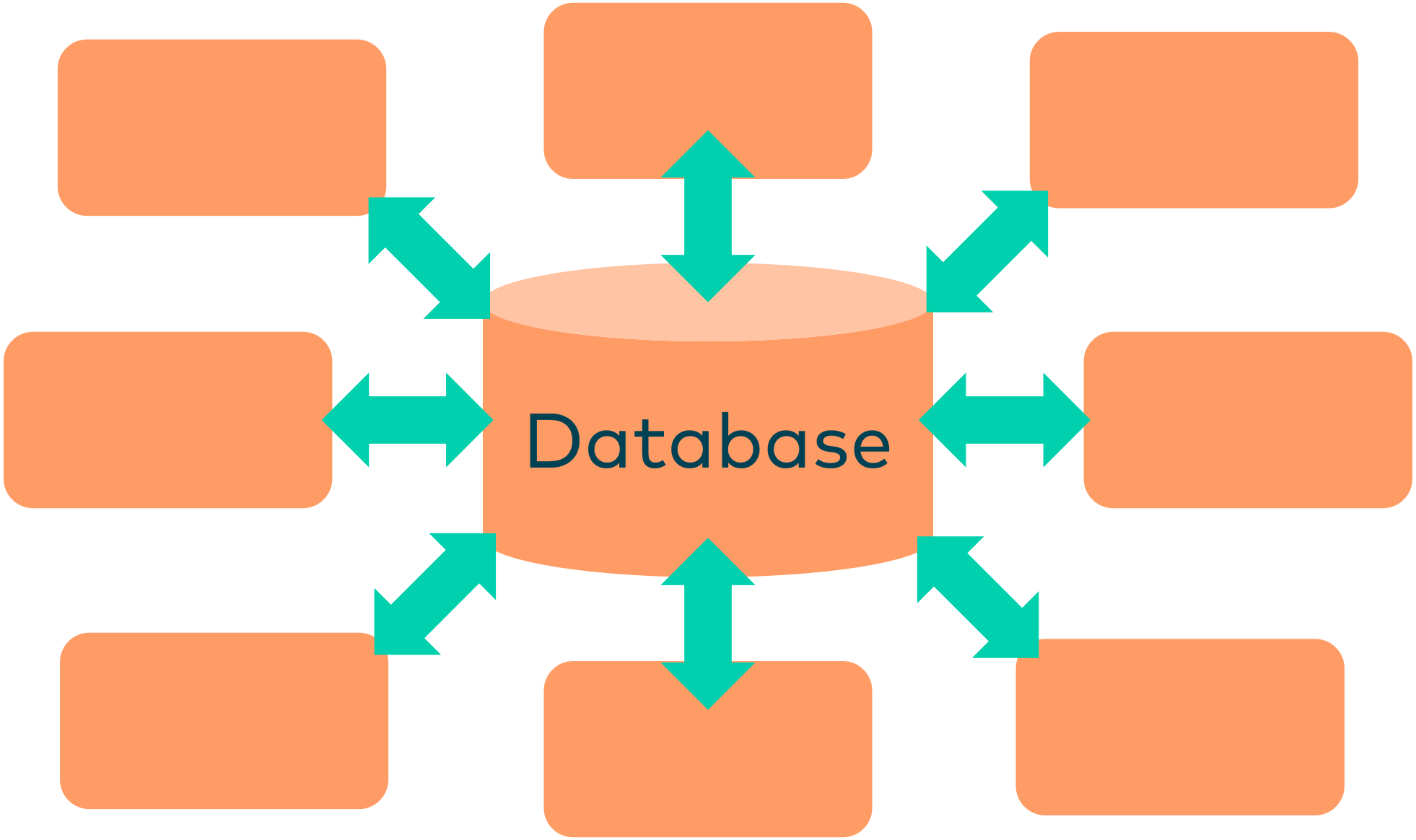
Customer e.g. product preferences

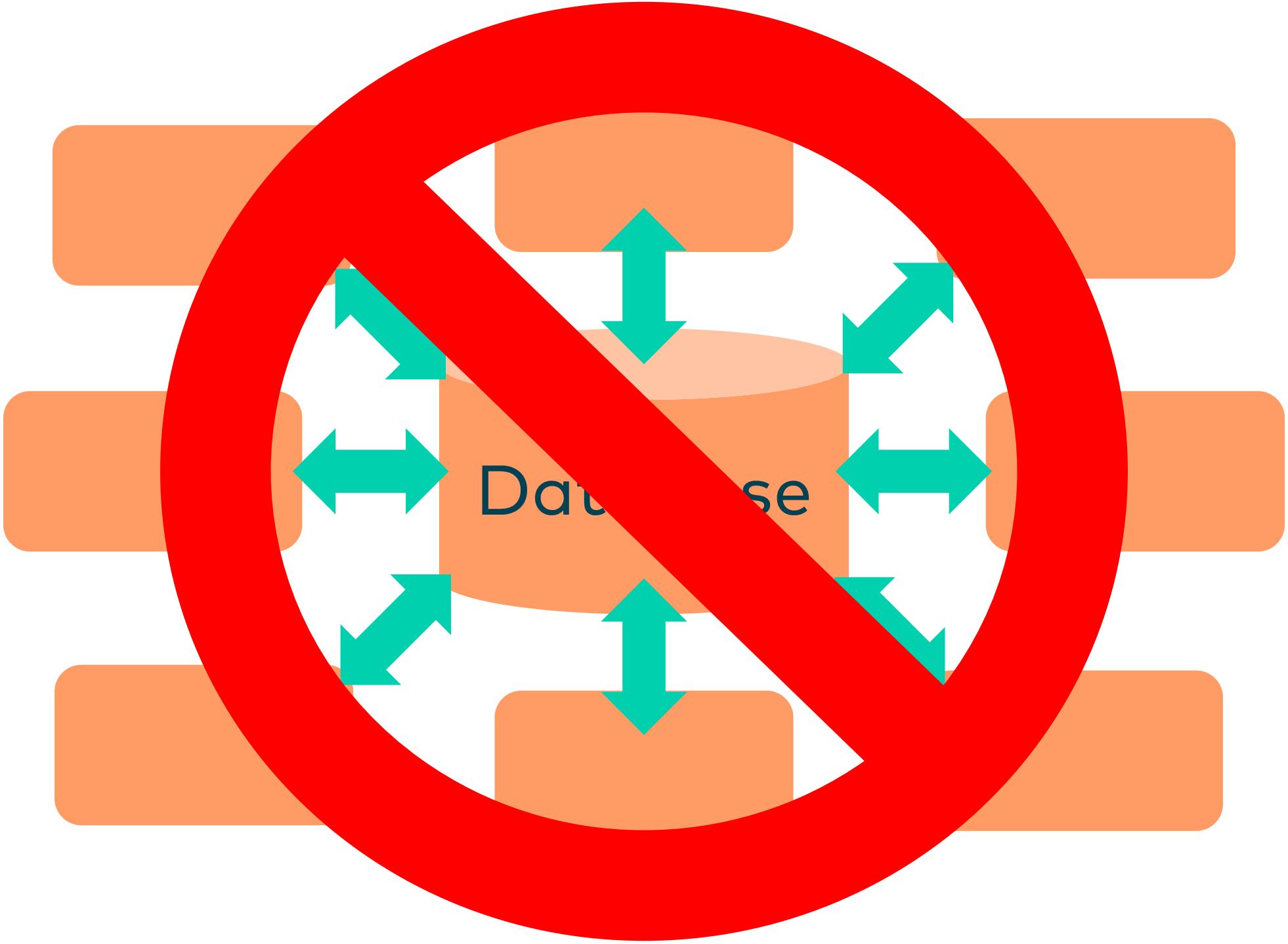
Product e.g. marketing information

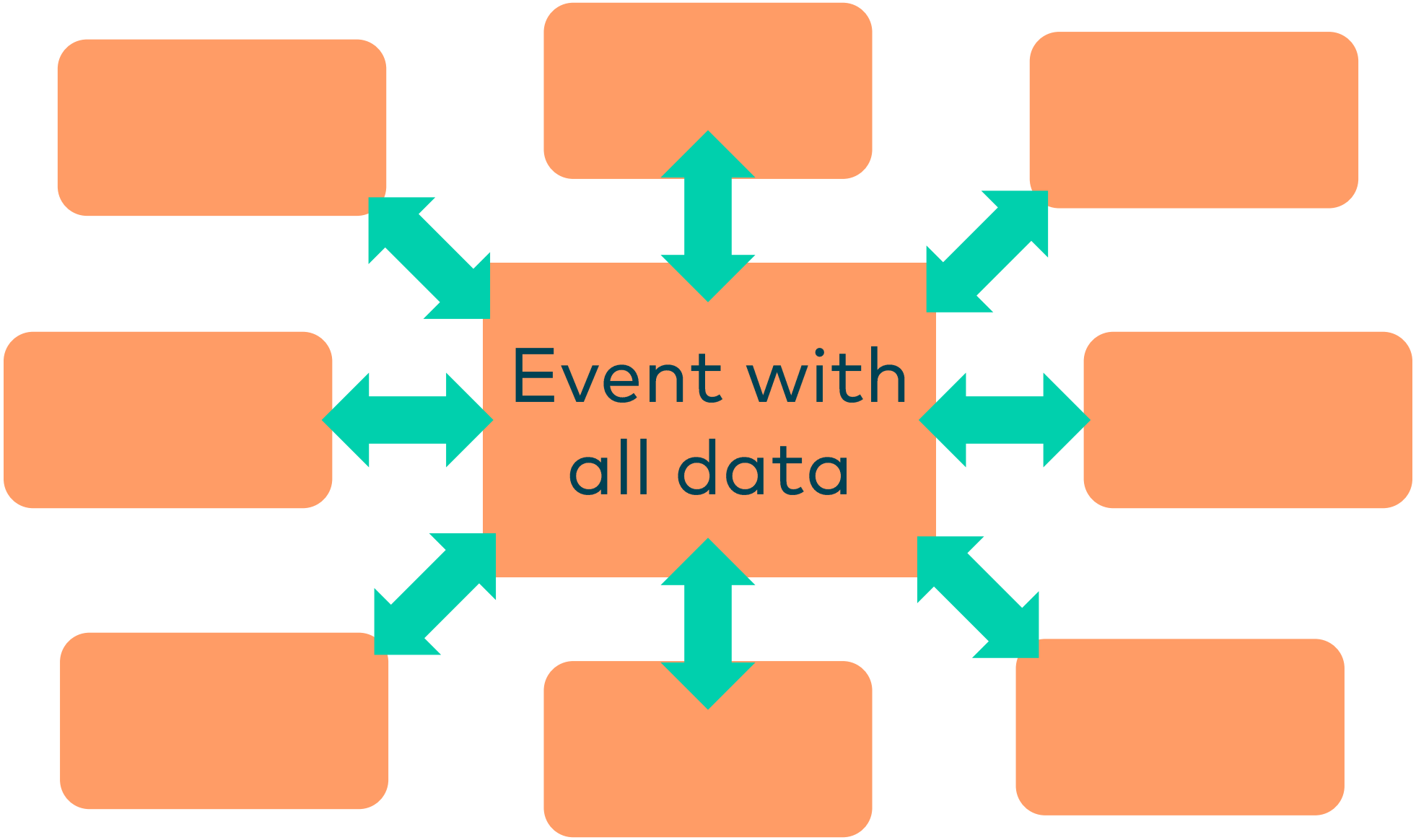
Bounded Context & Modules

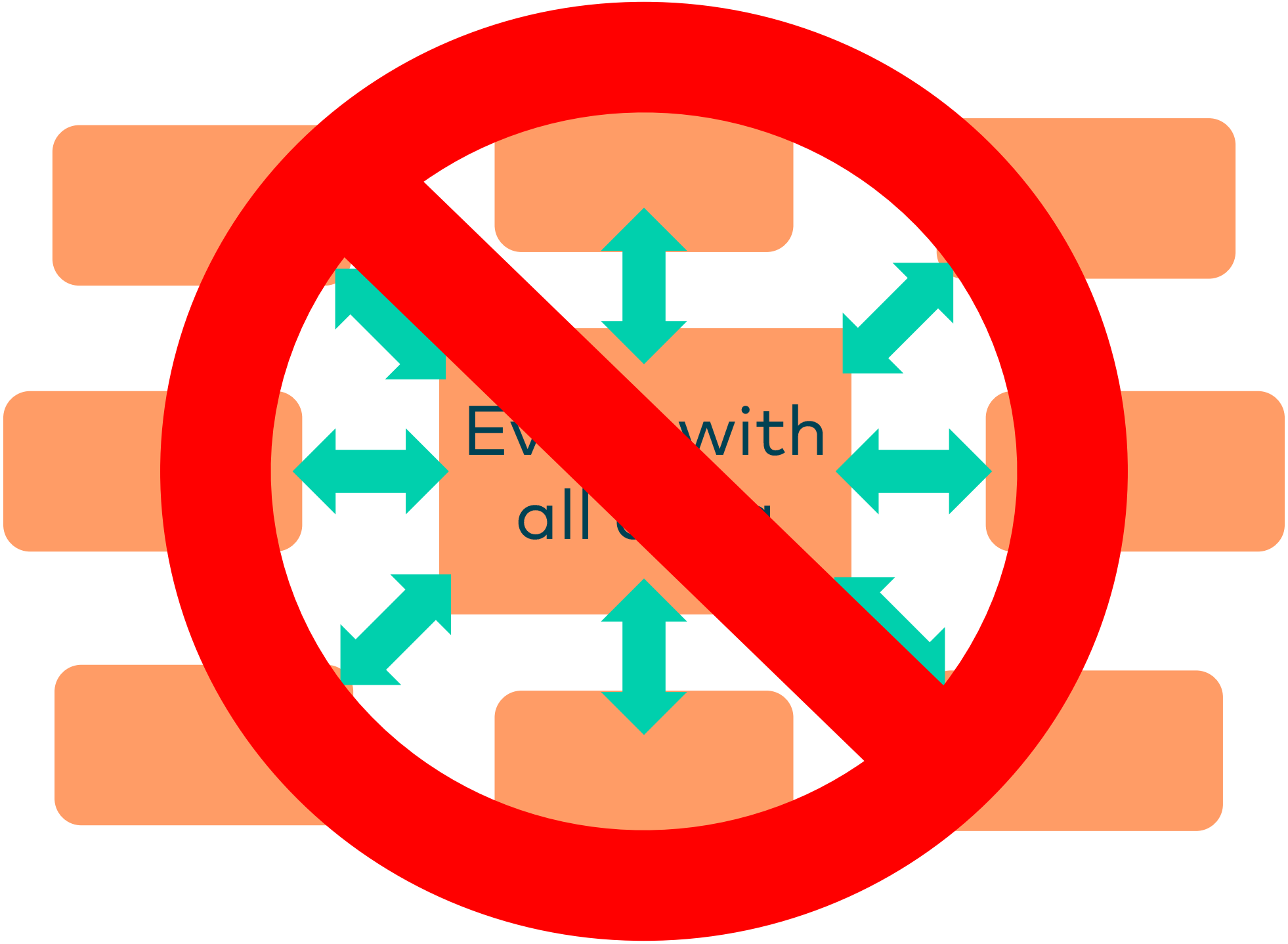
- Data model internal
- i.e. hides most design decisions.
- E.g. how data is stored

- Bounded Contexts are naturally great modules!


Database


Dat se


Event with
all data


Ev with
all

Quality

Highest quality everywhere!

eXtreme Programming

Software Craftsmanship

"The harsh reality is that not all parts of the design are going to be equally refined. Priorities must be set."

Core Domain Pattern, Eric Evans

What part has the highest business value?

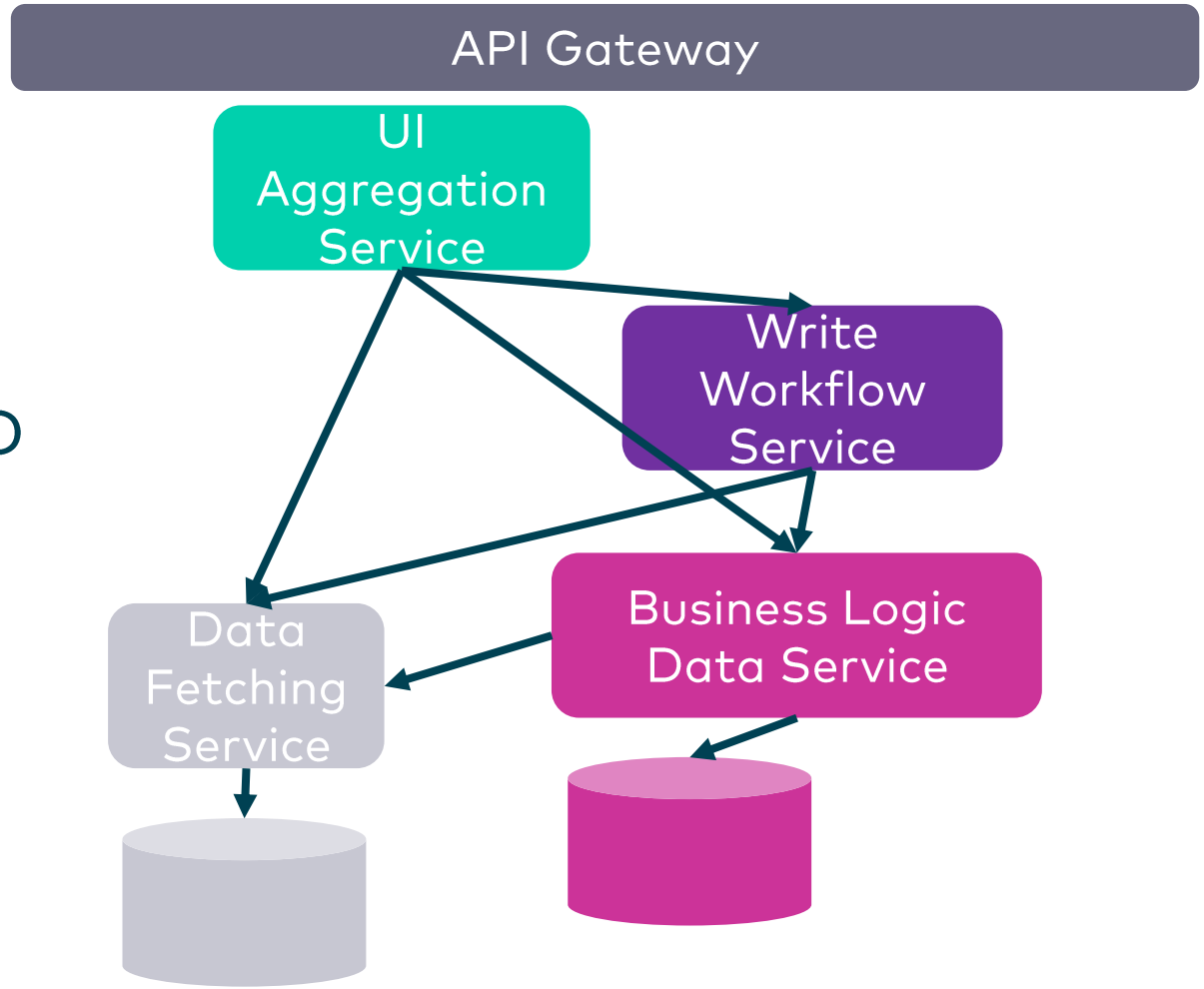
How can you focus if you don't know which part has the highest business value?

~~Quality~~

No Focus

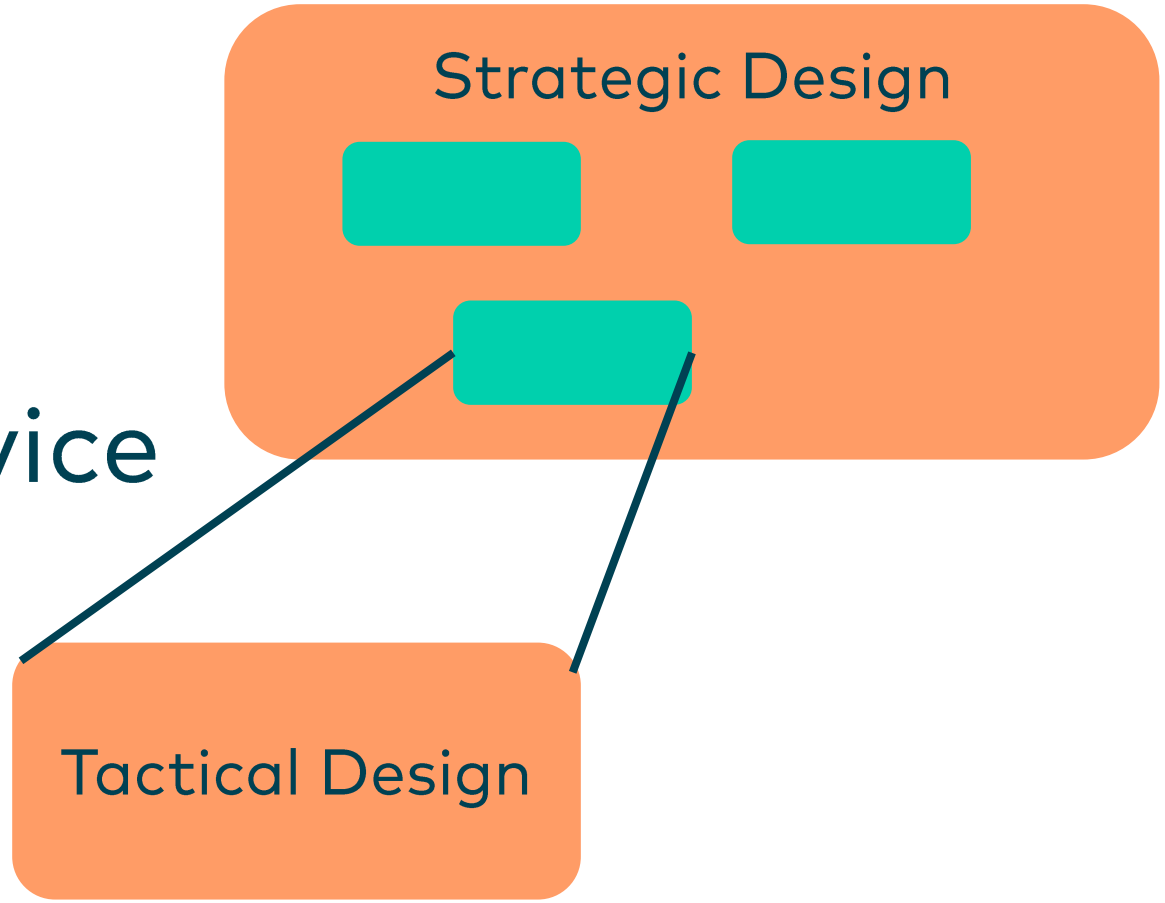
Is This a Great Architecture?

We are using all the tactical DDD pattern like Service, Repository, ...



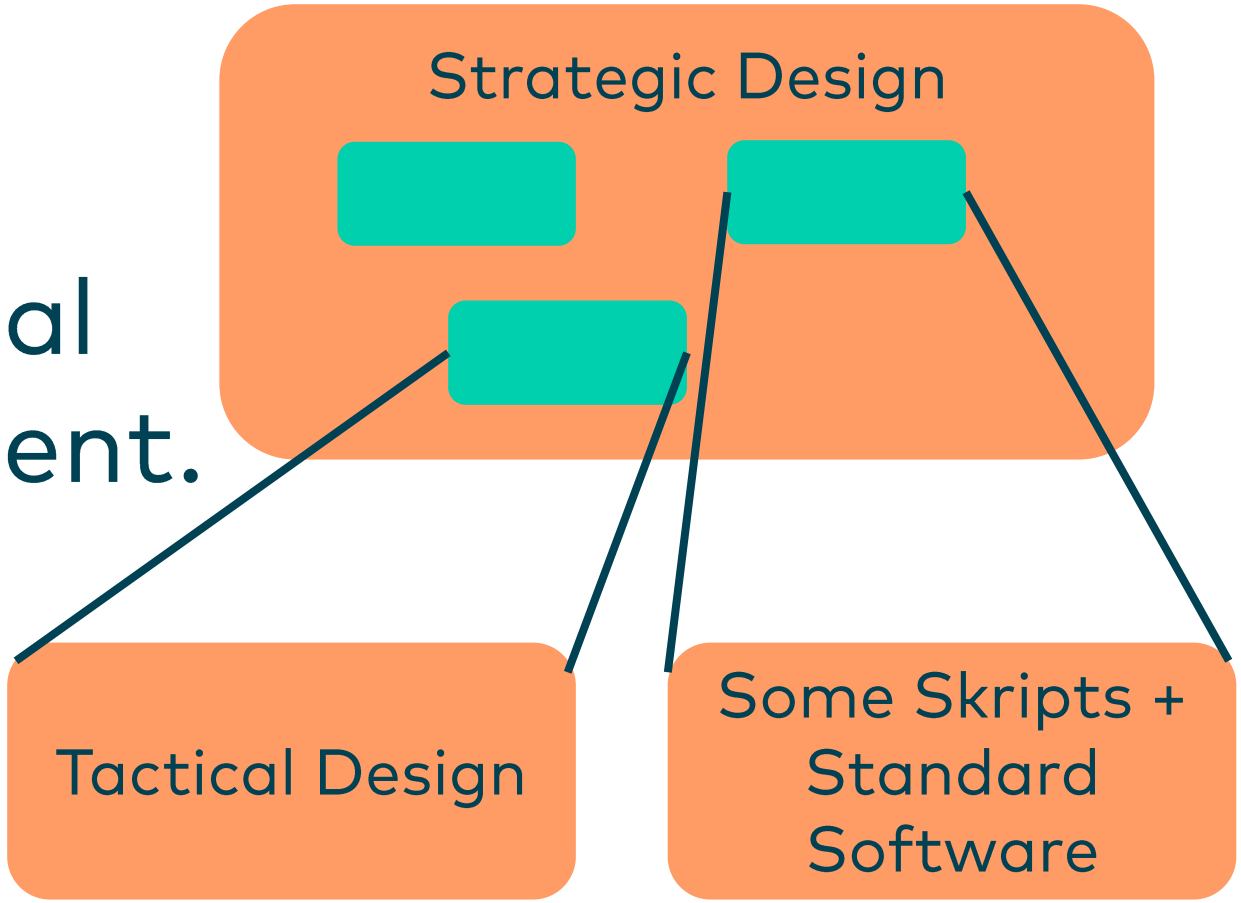
DDD: Strategic / Tactical

- Strategic Design:
  Coarse grained
  Modules / microservice
- Tactical Design:
  Fine grained
  Classes



DDD: Strategic / Tactical

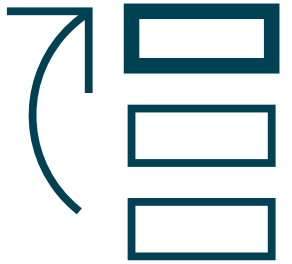
- Strategic and tactical design are independent.
- You can use one of them or both.



Strategic Design

- More important for architecture.
- More impact
- Sets priorities (Core domain)
- Solves problems between teams

Priorities



- No need to use tactical design to just move data around.
- Data by itself is not valuable
- What happens with data is valuable
- So: Simple business logic might be a missed opportunity to implement valuable business logic.

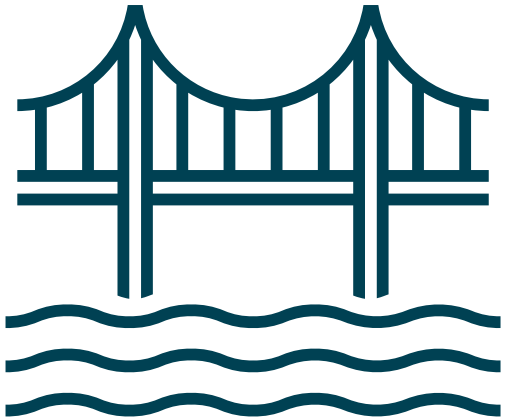
Architecture & Migration

- No more green fields
- Brown fields
- So: Migration
- Hardly discussed and very complex

Migrate

What is the next step?

How will it provide value?



How do we overcome
the next obstacles?

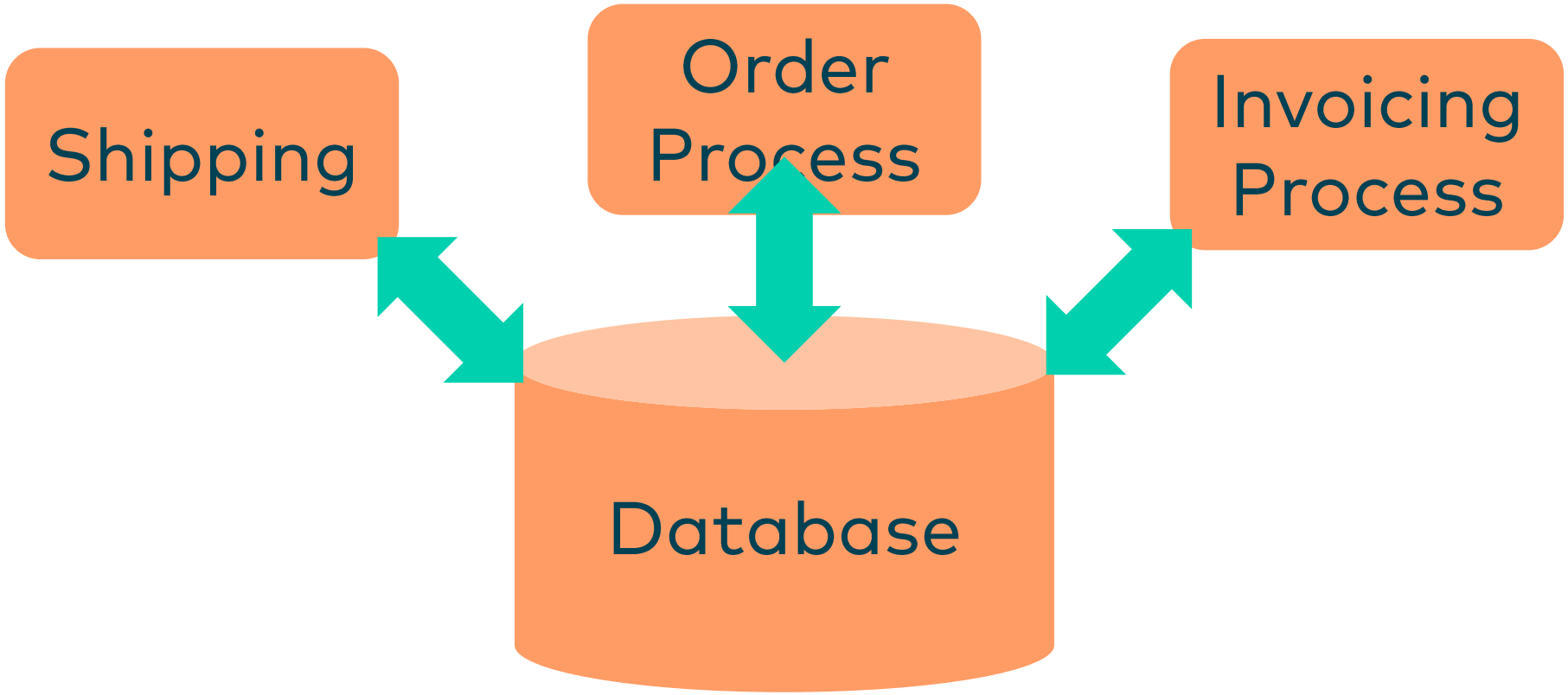
Goal: Full migration
in a few years

Bounded contexts

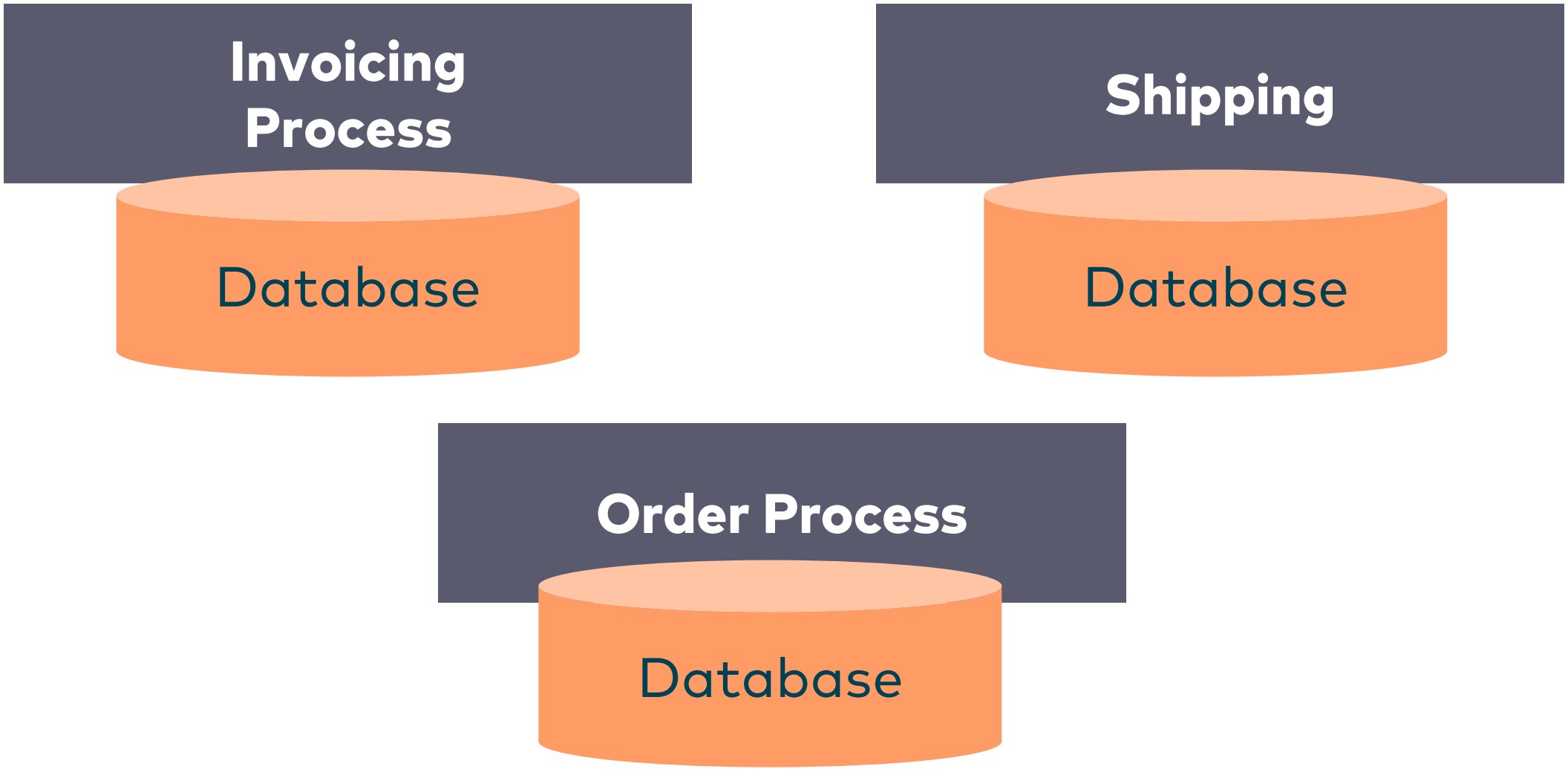
Migrating to Bounded Context

State before Migration

- Modules might share data



Goal: Bounded Contexts



Results

- Independent modules
- Less coordination
- More productivity

Migration

- Lots of effort to fully migrate – often years
- Business value? Just better productivity?
- First step?
- Value of first step?

Domain-driven Migration

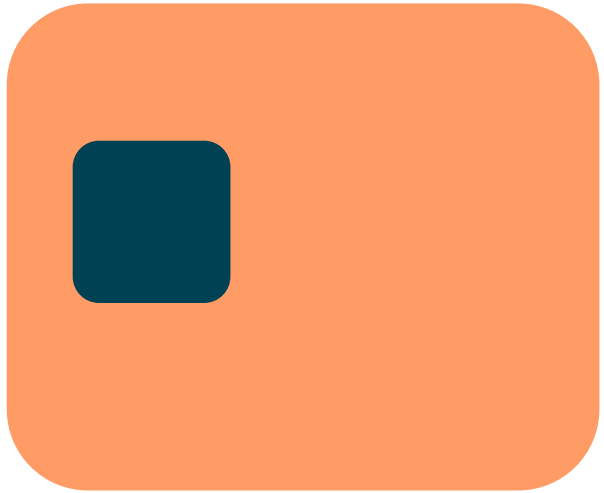
- The domain should drive the design.
- The domain should drive the migration.
- Where is the business value?
- Why are we doing this migration now?

Domain-driven Migration

- Might build new, separate bounded context for new features

Domain-driven Migration

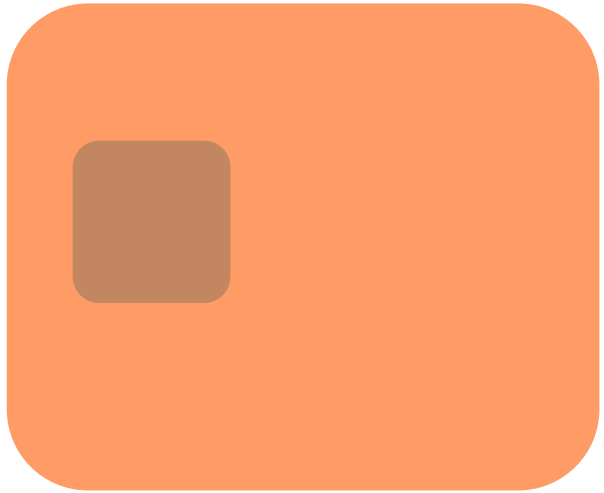
- Might build transient "bubble context" inside existing systems



https://www.domainlanguage.com/ddd/surrounded-by-legacy-software/
https://software-architektur.tv/2020/07/14/folge006.html

Domain-driven Migration

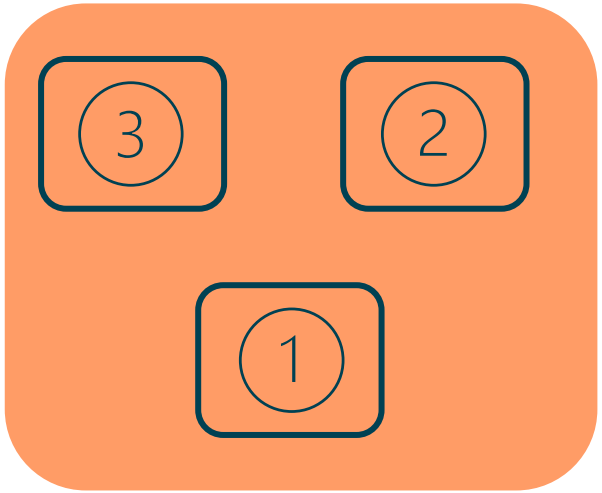
- Might build transient “bubble context” inside existing systems



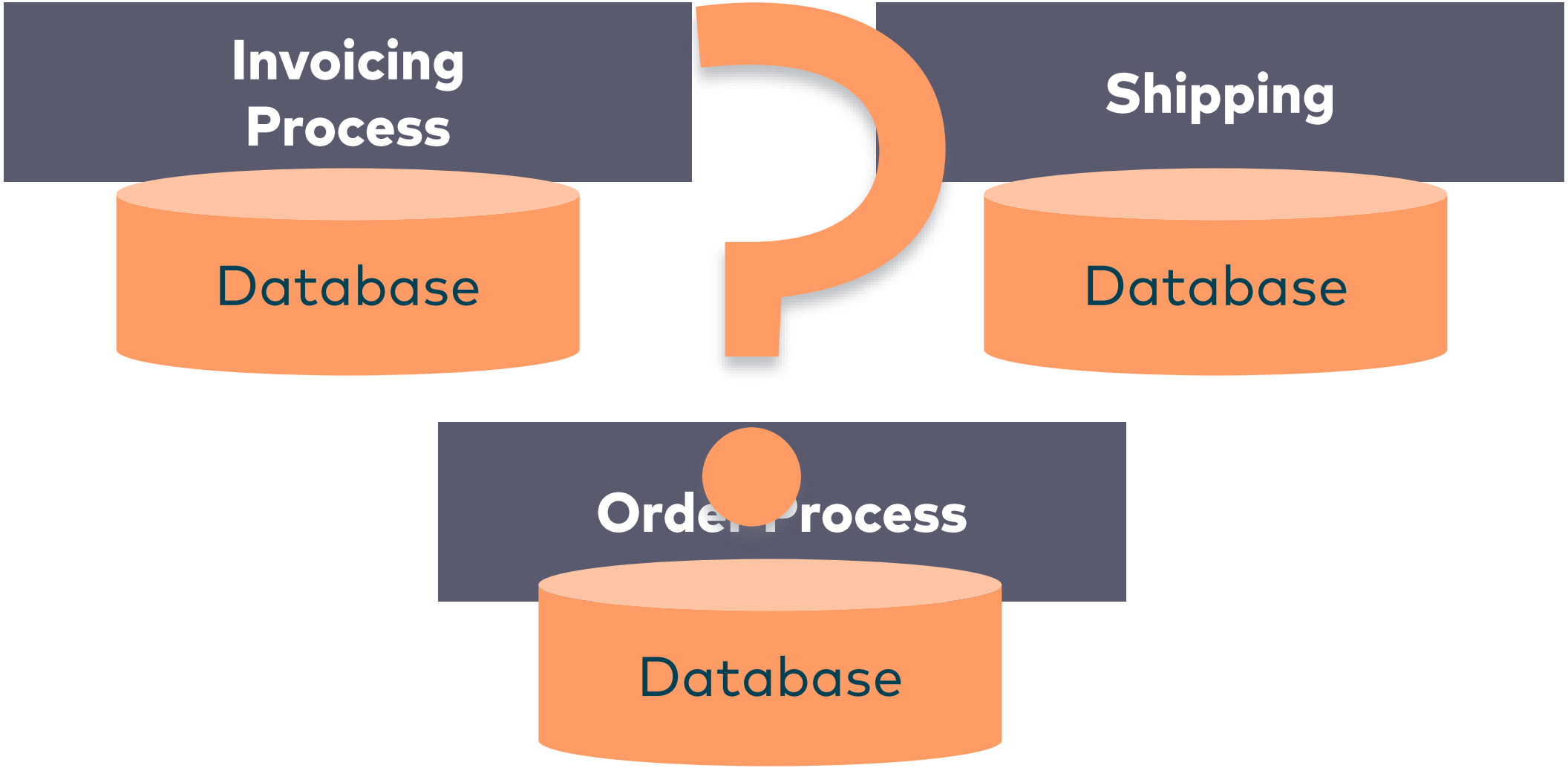
https://www.domainlanguage.com/ddd/surrounded-by-legacy-software/
https://software-architektur.tv/2020/07/14/folge006.html

Domain-driven Migration

- Define a core domain
- Might prioritize modules differently
  – not change them



Bounded Contexts: Really the Goal?



Domain-driven Migration

- Understanding bounded context is hard.
- Not actually implementing them is even harder

DDD: Migration

- Ask questions:
- Why is the migration done now?
- What are the next planned changes to the system?
- What has business given up asking for?

Strategic Domain-driven Design

Strategic Domain-driven Design

- A bounded context might be more or less important.
- A team might need support from other teams to be successful.
- Only solution: Patterns for collaboration between teams

Strategic Domain-driven Design

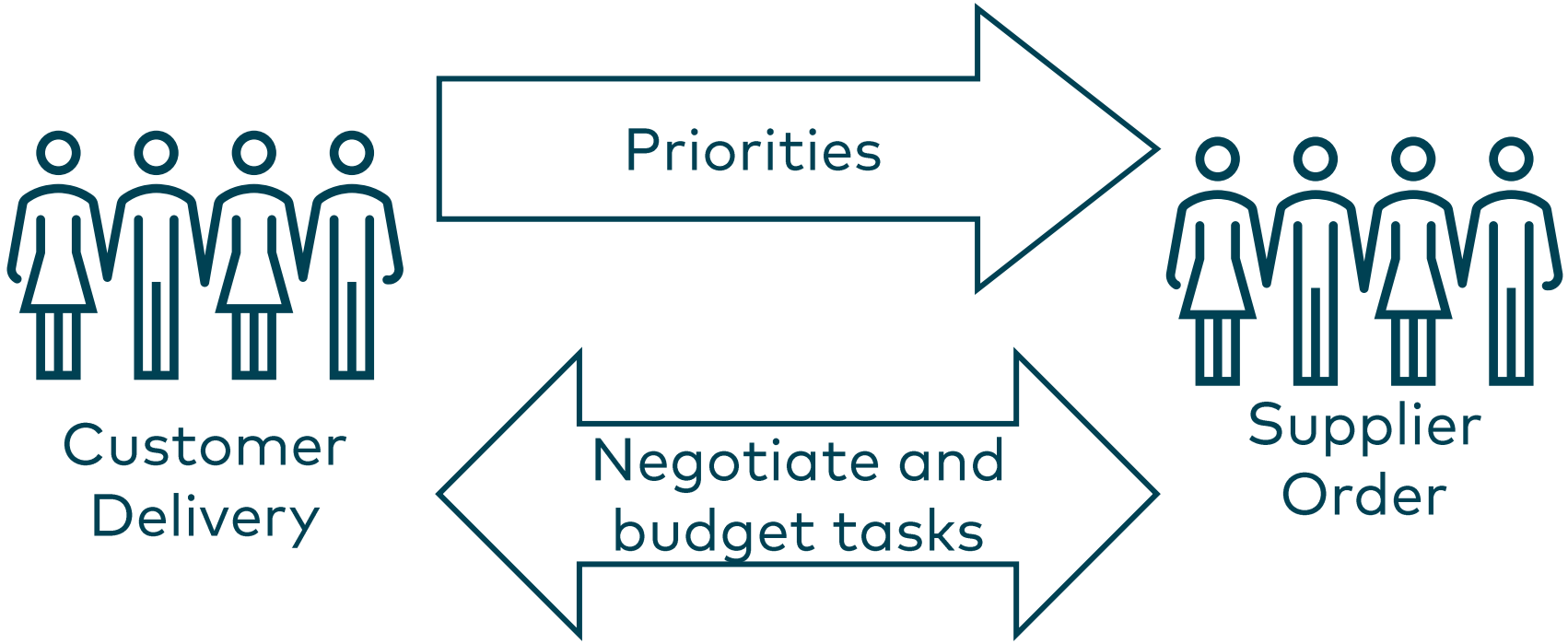
- Downstream team depends on upstream team to be successful.

Customer / Supplier

- Factor downstream priorities (customer) into planning of upstream team (supplier)!
- Negotiate and budget tasks!

Strategic Domain-driven Design

Customer / Supplier



Why do I see so little adoption of strategic domain-driven design?

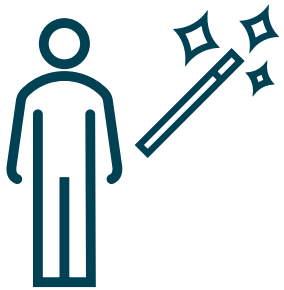
Can you set up a Customer / Supplier relationship?

Customer / Supplier

- Customer / supplier is about who does what.
- Probably not what an architect decides.

Setting Up Customer / Supplier

Supplier



Customer

Customer / Supplier in Practise

Customer / Supplier

- Managers can set up a customer / supplier relationship.
- In real life, teams can circumvent or fight such rules.

Is this Customer / Supplier?

At the end, she and her team got the help they needed...

Even if you don't have formal control over a team, you can still try to influence it.

Organisation

- Folge 163 - Kommunikation im Entwicklungsprozess mit Rebecca Temme
- Folge 147 - Wie reißt man den Elfenbeinturm ein? mit Anja Kammer
- Folge 141 - Auftragstaktik - Agilität beim Militär? mit Sönke Marahrens
- Folge 125 - Organisation und Architektur - ein Beispiel
- Folge 115 - Data Mesh - nur ein neuer Datenanalyse-Hype?
- Folge 110 - Conway's Law
- Folge 106 - Anne Herwanger, Alexandra Hoitz, Stefan Link - Resiliente Organisation und resiliente Software Architektur - live von der OOP
- Episode 101 - Kenny Baas-Schwegler, Gien Verschatse, Evelyn Van Kelle - Facilitating Collaborative Design Decisions - Live from OOP
- Folge 96 - Organisation, Architektur - Was ich im Stream gelernt habe
- Folge 91 - Sven Johann - Cross-funktionale Teams zielgerichtet in den Abgrund stürzen
- Episode 82 - Avraham Poupko & Kenny Baas-Schwegler - The Influence of Culture on Software Design
- Epsiode 80 - Microservices, Inverse Conway Maneuver, and Flow with James Lewis - Live from Software Architecture Gathering
- Folge 73 - Das Spotify-Modell gibt es gar nicht!
- Folge 63 - Kim Nena Duggen zu Soft Skills für Software-Architekt:innen
- Folge 16 - Gerrit Beine zu Sozialwissenschaften und Software-Architektur
- Folge 2 - Organisation und Architektur

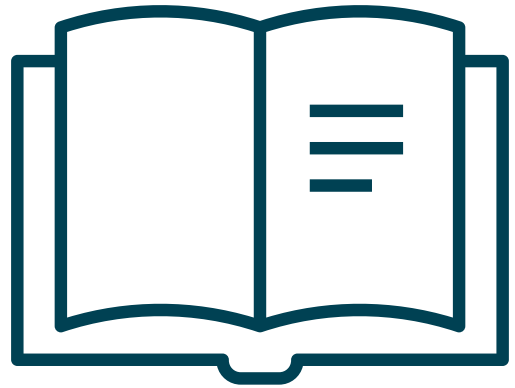
https://software-architektur.tv/tags.html#Organisation

Iterations

These [domain] models are never perfect; they evolve.

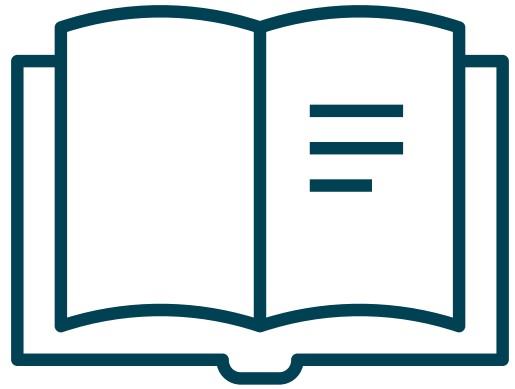
Eric Evans

A True Story



- Plan at start:
  Migrate the system module-by-module
- Prototype to validate migration.

A True Story: The Start

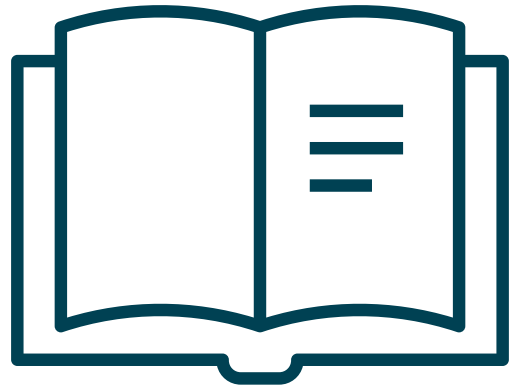


- Project start
- Learn more about the domain
- Migration by module makes it impossible

...to improve support for business.

...to improve automation

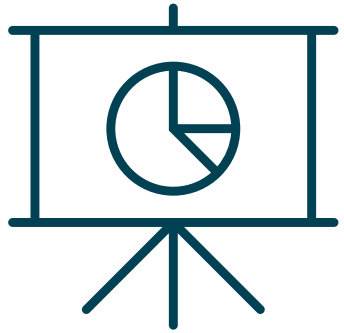
A True Story: Result



- There were other issues, too.
- Project cancelled

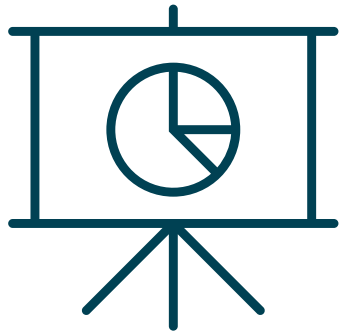
...and considered a failure.

Intuitive Lesson Learned



- Do more research up front!
- Be more restrict in approving projects!
- IMHO this is wrong.
- You will always learn about the domain!
- i.e. there will always be something wrong.
- Not just at the start.

Recommended Lesson Learned



- Consider dropping the technical validation of an architecture.
- It might need to be changed.
- You might be too (emotionally) invested.
- Be prepared to change the architecture.
- But: don’t be intentionally stupid!

Conclusion

Conclusion: DDD vs nDDD

- Domain-driven Design means the domain drives the design.
- Actually learn and understand the domain!

Conclusion: Quality & Focus

- Focus on the parts with the highest business value!
- Don’t invest too much in other parts.

Strategic Design & Iterations

- You can do strategic design.

...even if you don't have the formal authority.

Iterations

- You will learn about the domain.
- So: Work in iterations.

Domain-driven Design

- Folge 134 - Domain Prototyping - Iterative Entwicklung mit Domain-driven Design & User Experience mit Tobias Goeschel
- Folge 116 - Events, Event Sourcing und CQRS
- Folge 90 - Michael Plöd - Wie steigt man in Domain-driven Design ein?
- Folge 72 - Strategisches Domain-driven Design - Grundlegende Patterns unter der Lupe
- Folge 22 - Markus Völter zu Fachliche Architekturen mit DSL (Domain Specific Languages)
- Folge 21 - Domain Story Telling mit Henning Schwentner und Stefan Hofer
- Folge 17 - Nicole Rauch zu DDD, Event Storming & Specification by Example
- Folge 11 - Nick Tune - Legacy Architecture Modernisation With Strategic Domain-Driven Design
- Folge 6 - Eric Evans "Getting Started with DDD When Surrounded by Legacy Systems"
- Folge 4 - Fachliche Architektur - Warum und wie?

https://software-architektur.tv/tags.html#Domain-driven%20Design

Send email to jax2023@ewolff.com
Slides
+ Service Mesh Primer EN
+ Microservices Primer DE / EN
+ Microservices Recipes DE / EN
+ Sample Microservices Book DE / EN
+ Sample Practical Microservices DE/EN
+ Sample of Continuous Delivery Book DE

Powered by Amazon Lambda
& Microservices

EMail address logged for 14 days,
wrong addressed emails handled manually