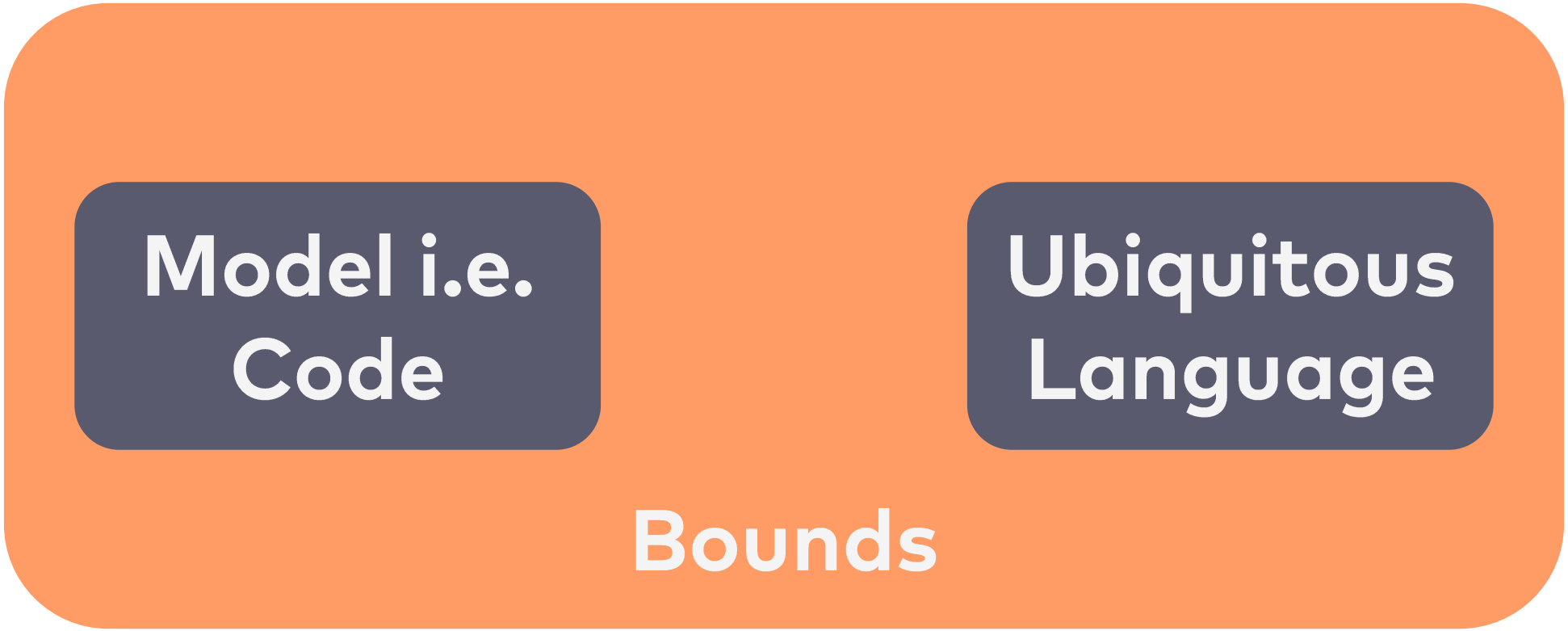
Why You Might Fail with Domain-driven Design

INNOQ

Eberhard Wolff
Fellow @ewolff

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

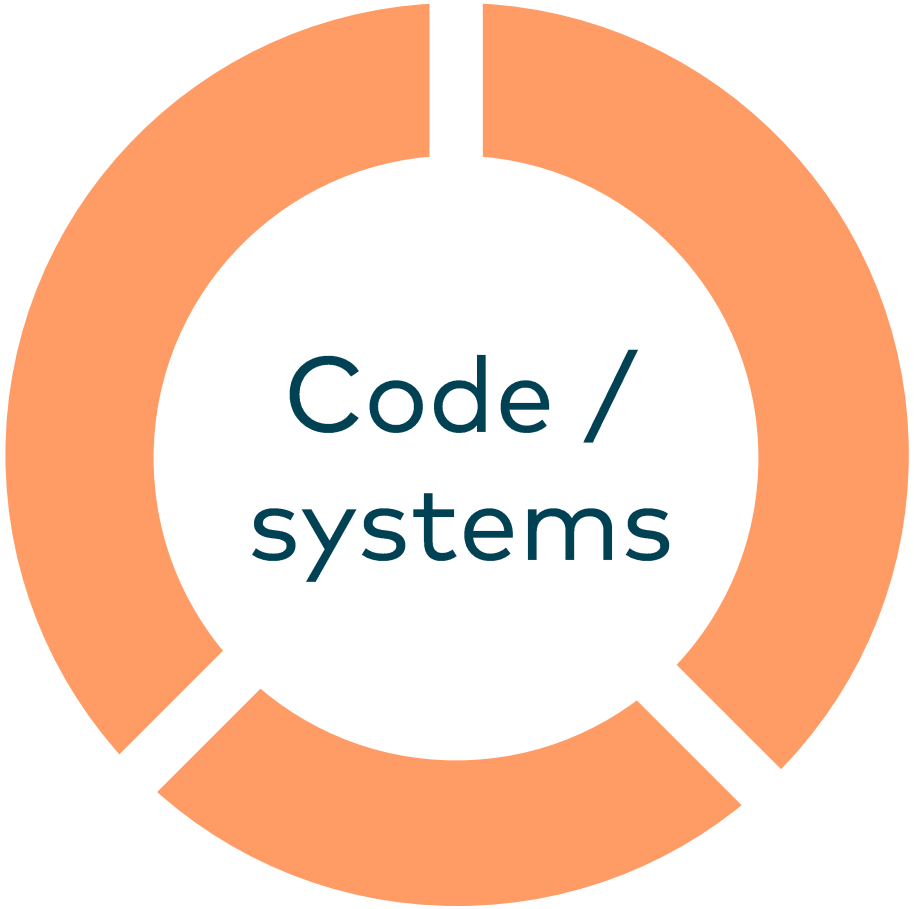
Class

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

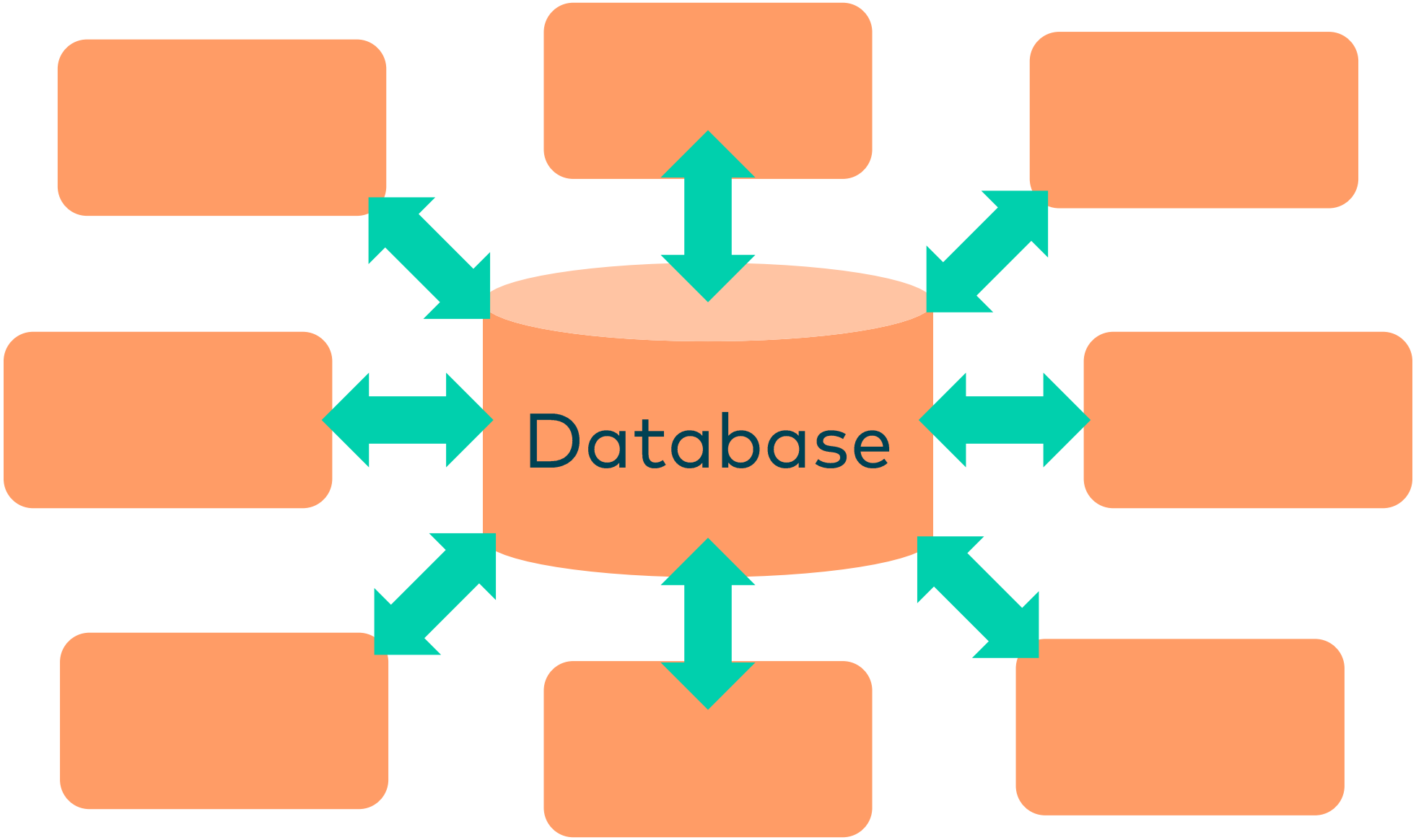
Customer e.g. product preferences

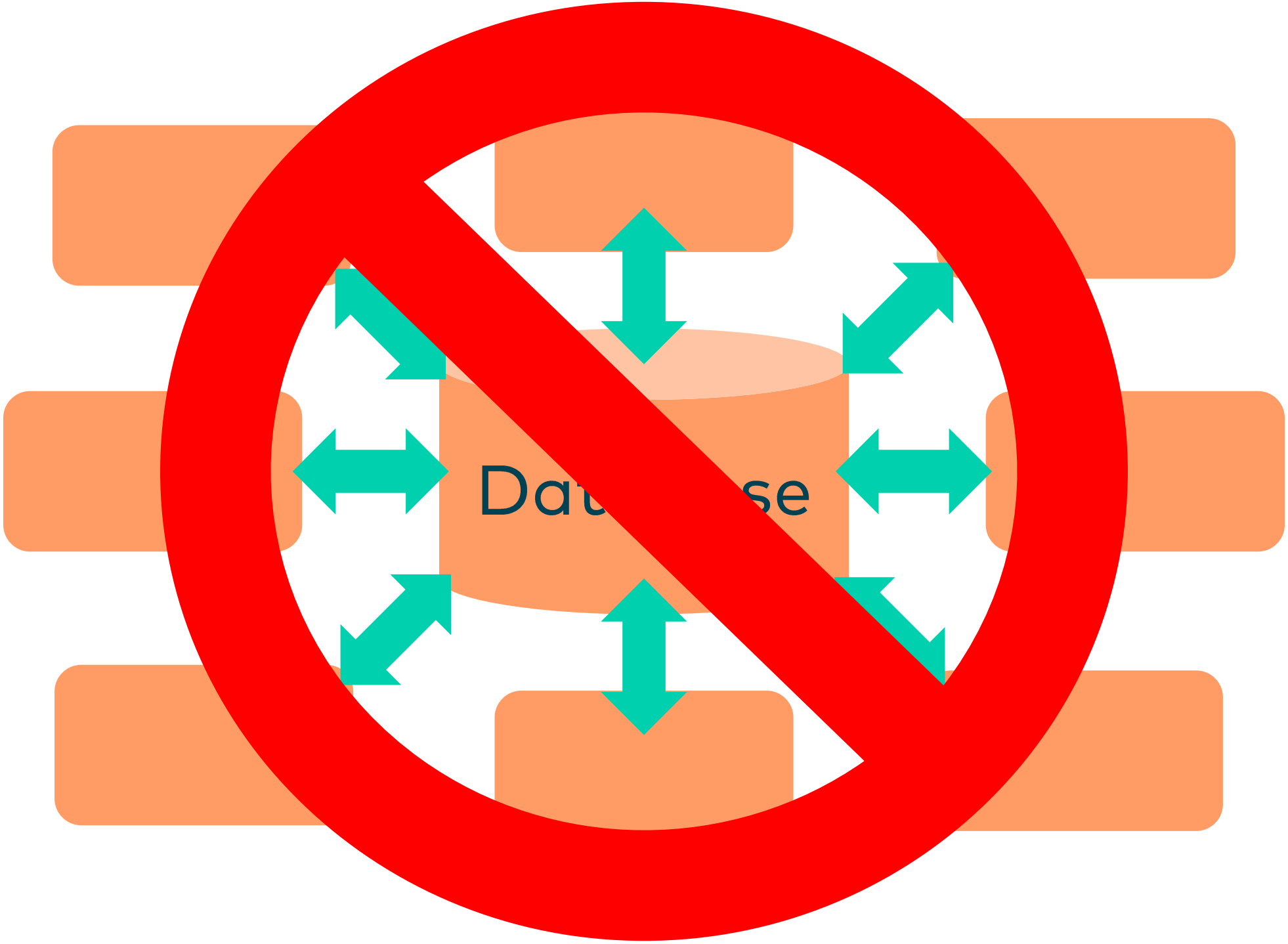
Product e.g. marketing information

Bounded Context & Modules

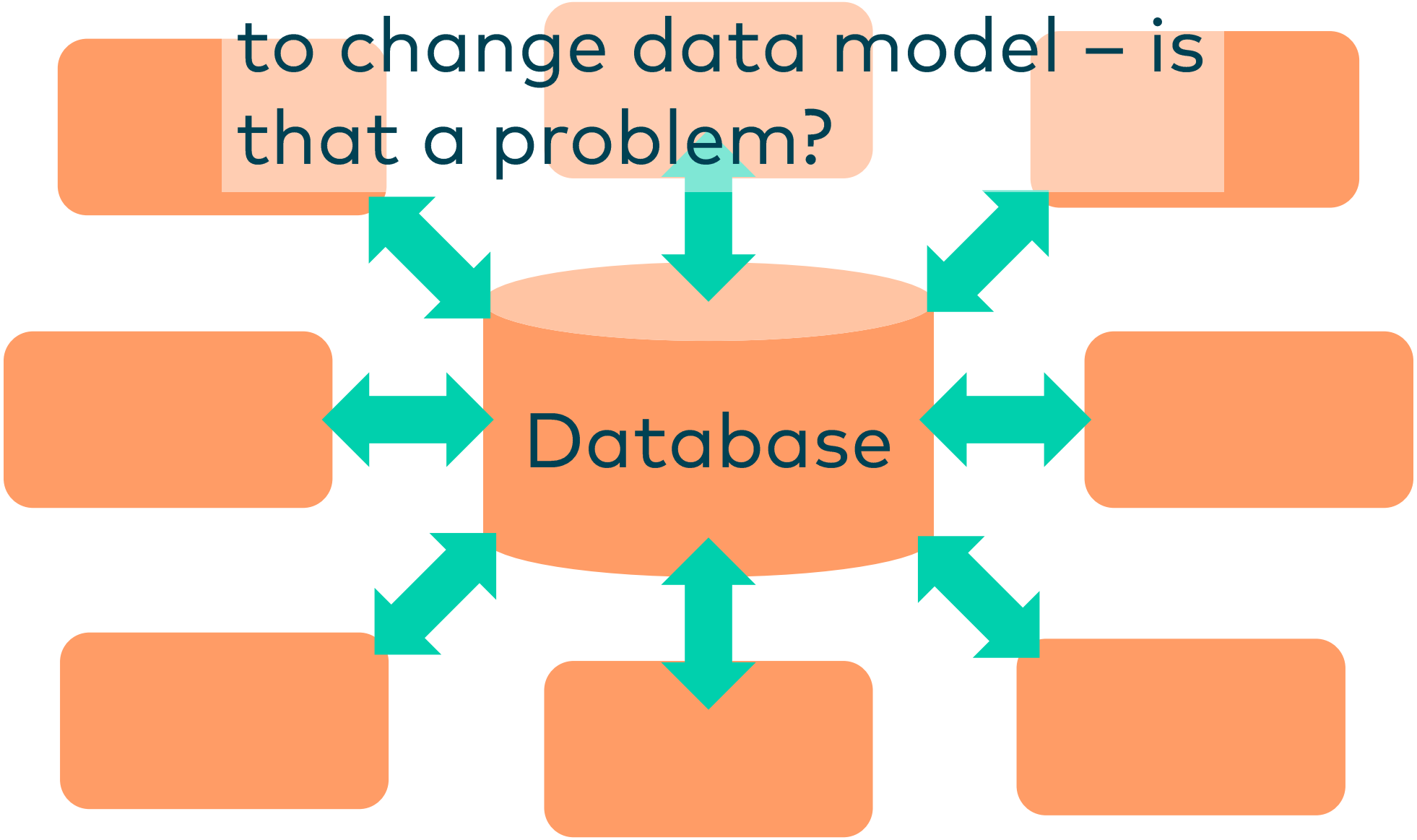
- Data model internal
- i.e. hides most design decisions.
- E.g. how data is stored

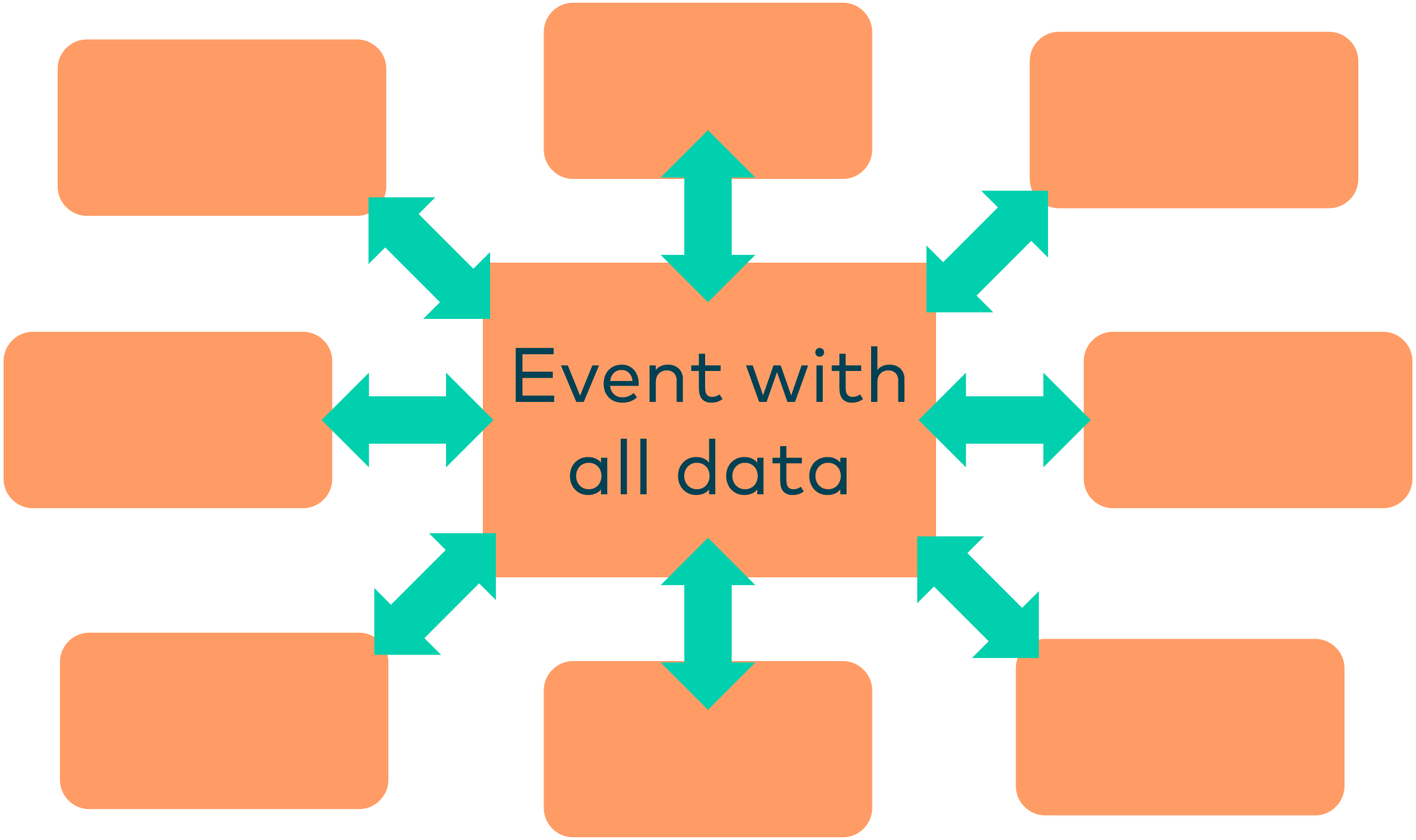
- Bounded Contexts are naturally great modules!

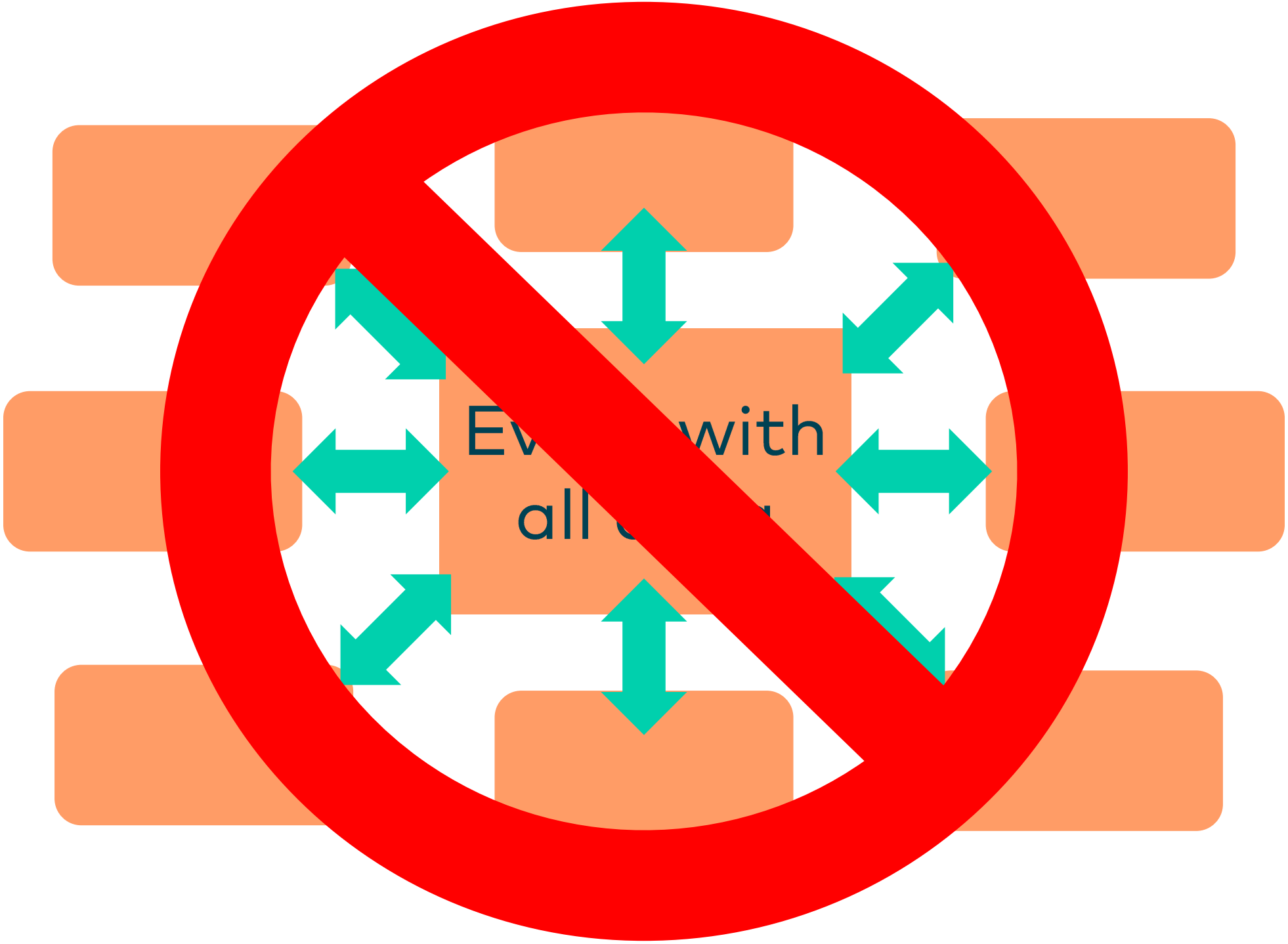

Database


Dat se

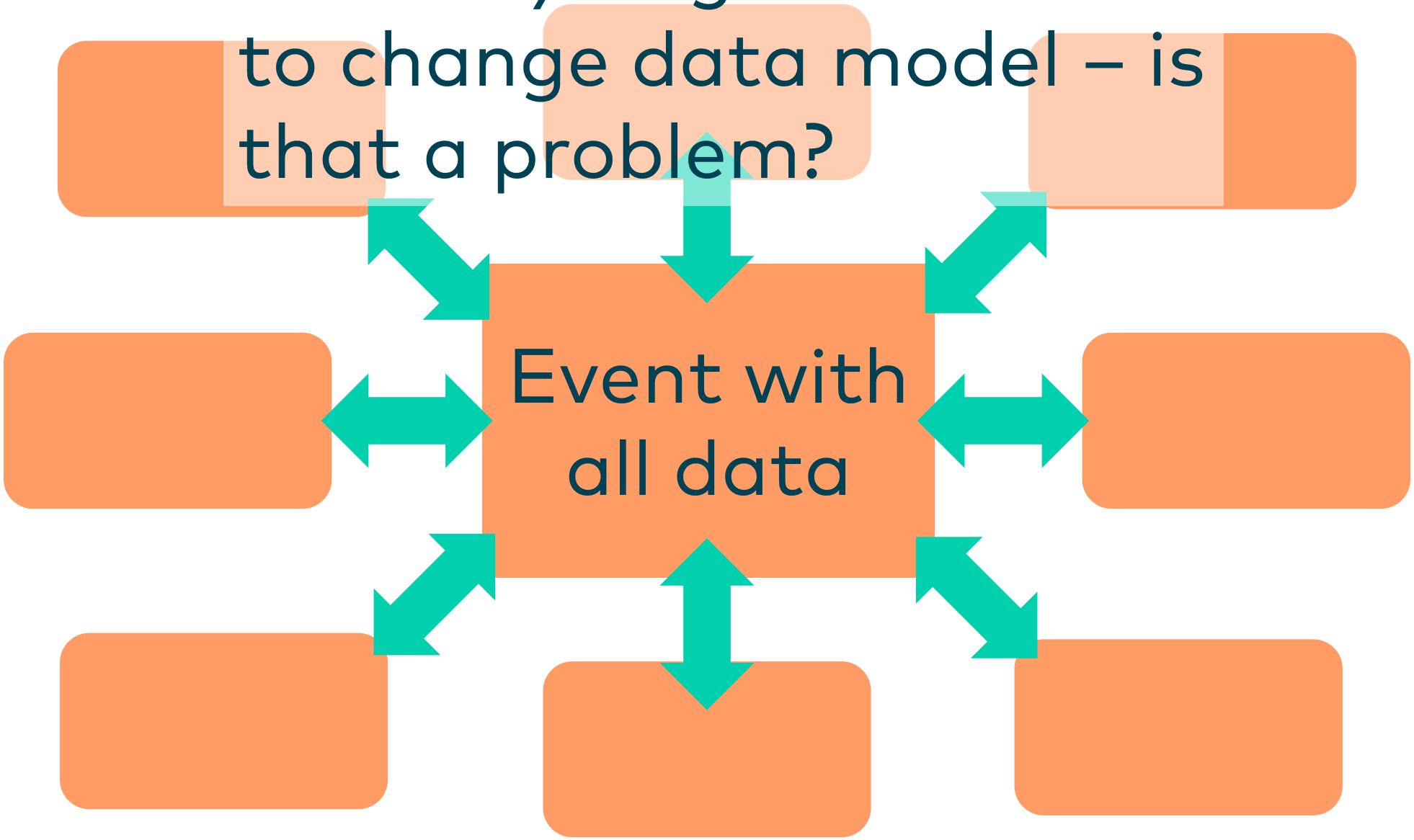
Probably large and hard to change data model – is that a problem?




Event with
all data


Ev with
all

Probably large and hard to change data model – is that a problem?



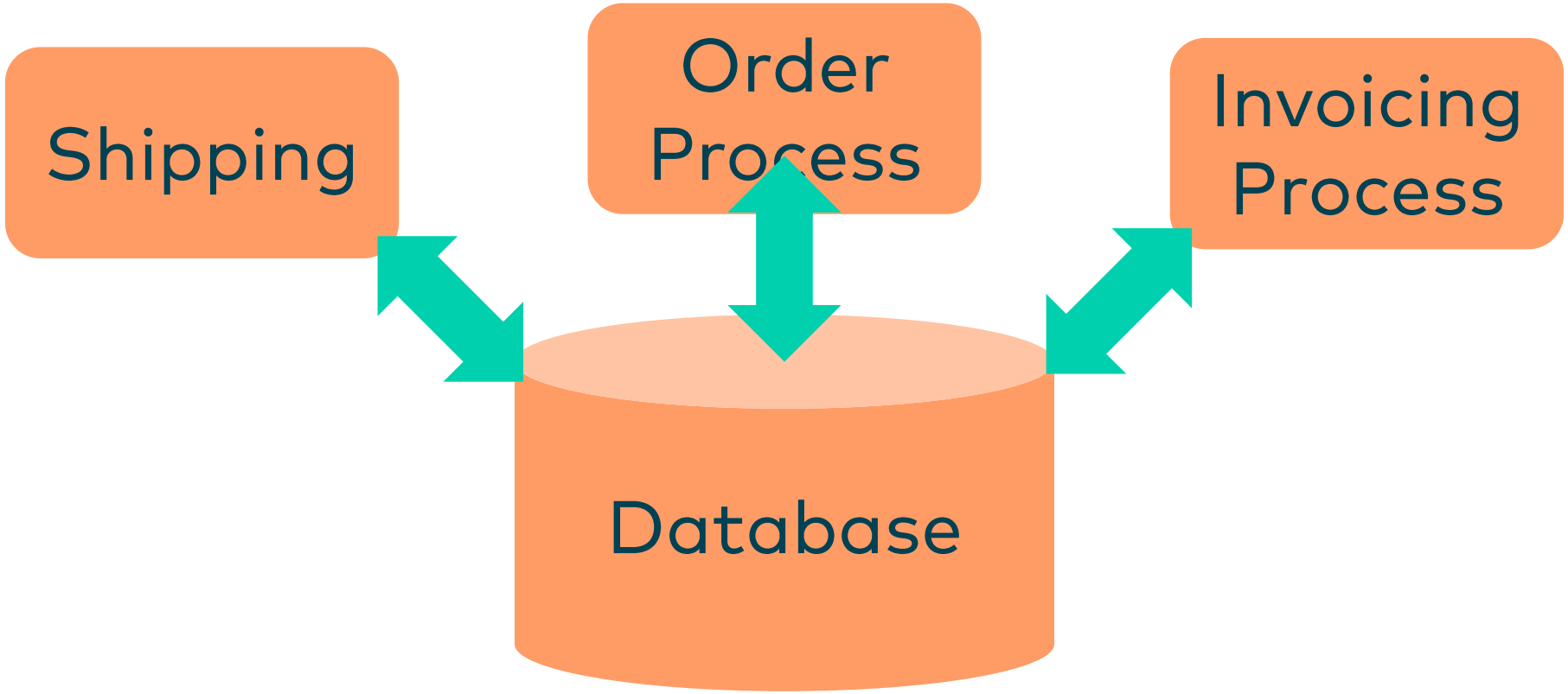
software-architektur.tv/shorts

https://www.youtube.com/watch?v=RCHZ6oCNZvU

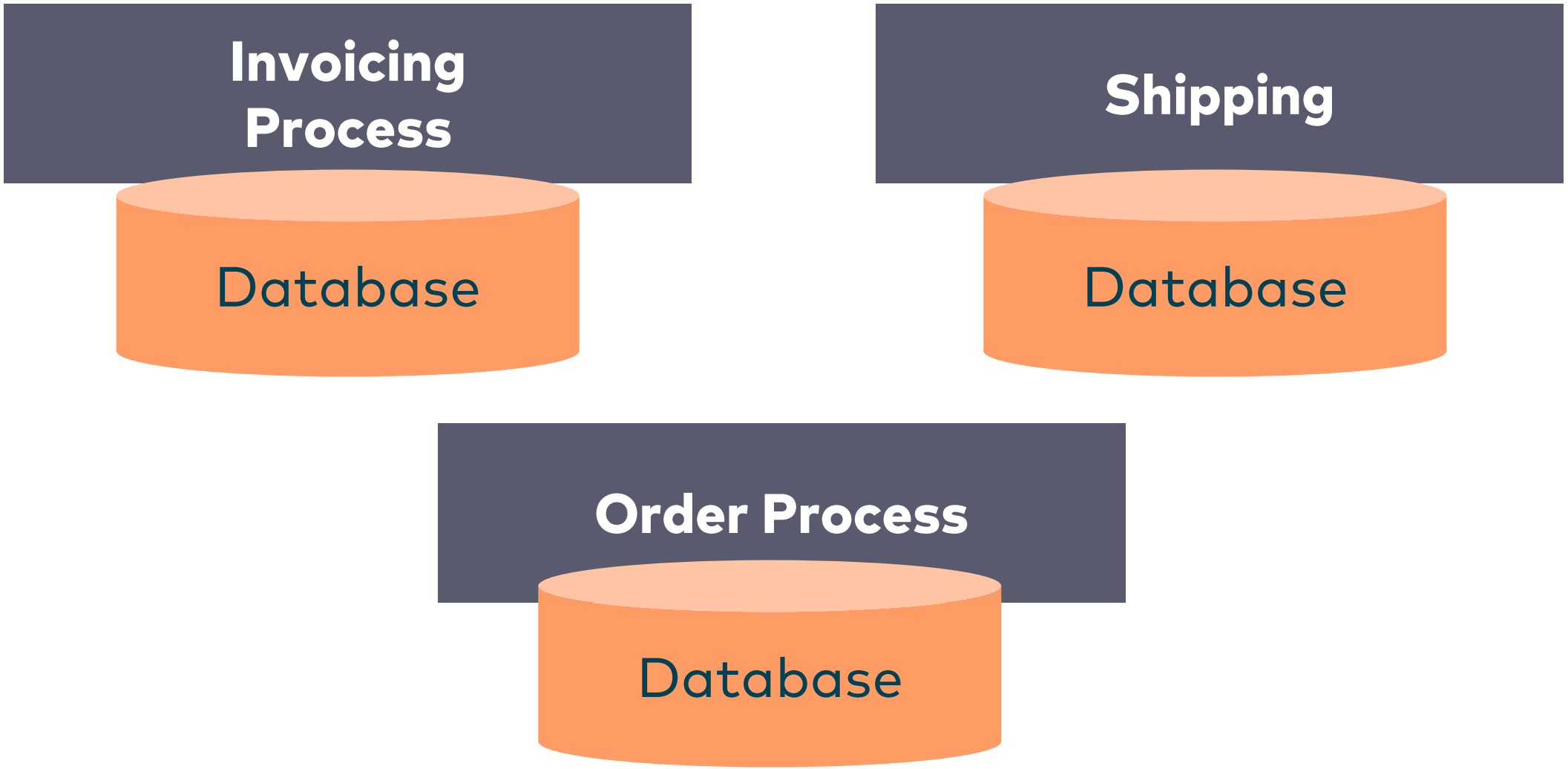
Migrating to Bounded Context

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

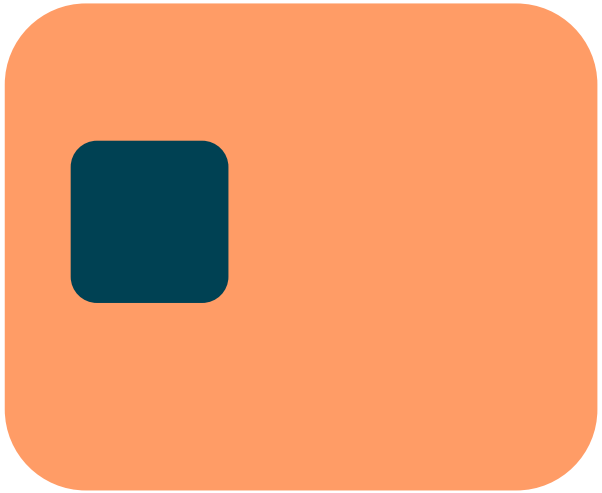
- The domain should drive the design.
- The domain should drive the migration.
- Where is the business value?
- Why are we doing this migration now?

Domain-driven Migration

- Might build new, separate bounded context for new features

Domain-driven Migration

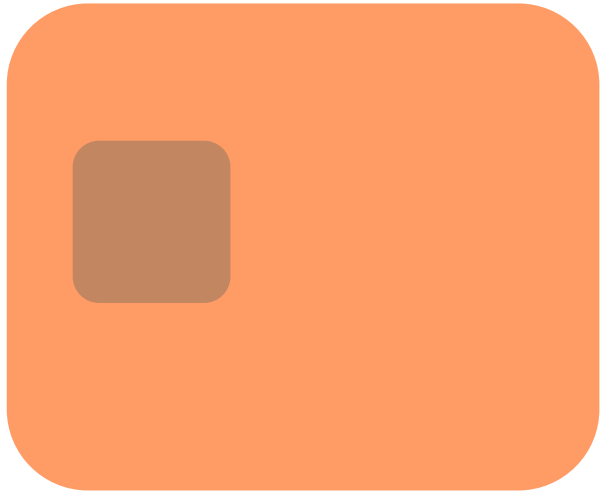
- Might build transient “bubble context” inside existing systems



https://www.domainlanguage.com/ddd/surrounded-by-legacy-software/

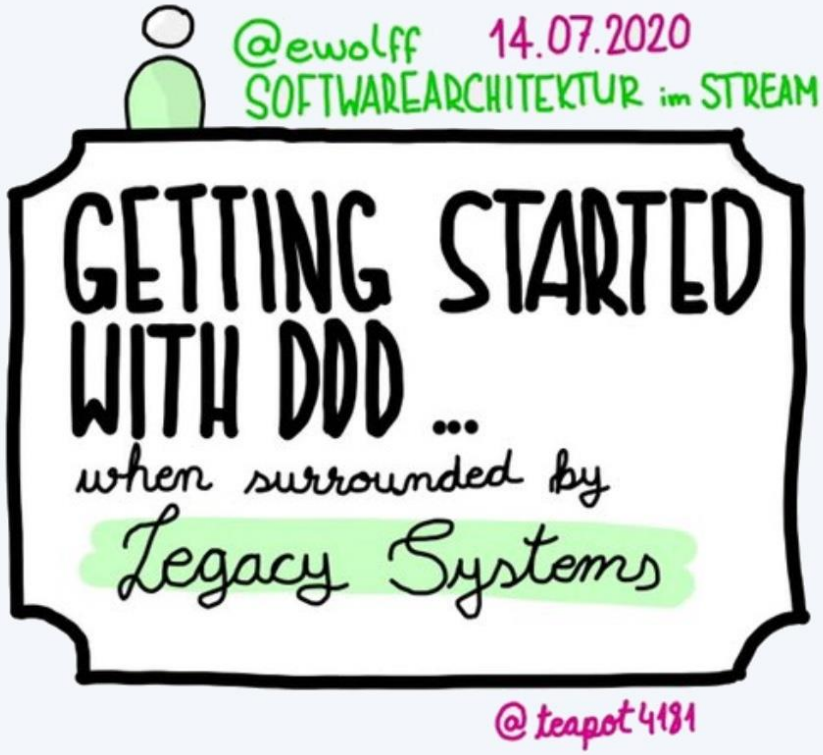
Domain-driven Migration

- Might build transient "bubble context" inside existing systems



https://www.domainlanguage.com/ddd/surrounded-by-legacy-software/

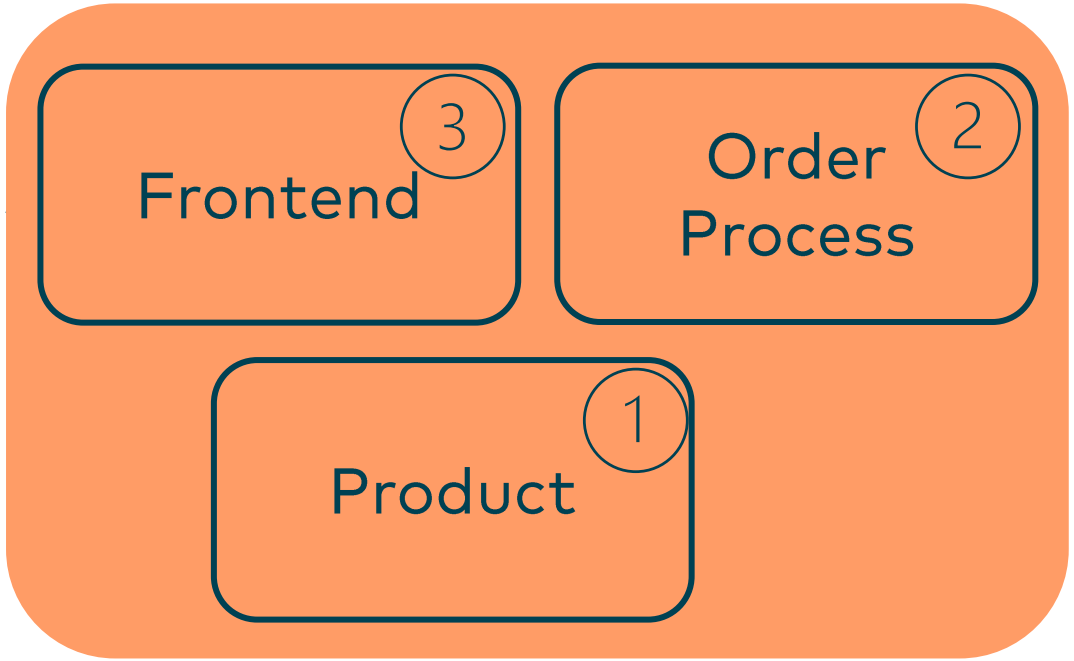
#SoftwareArchitektur



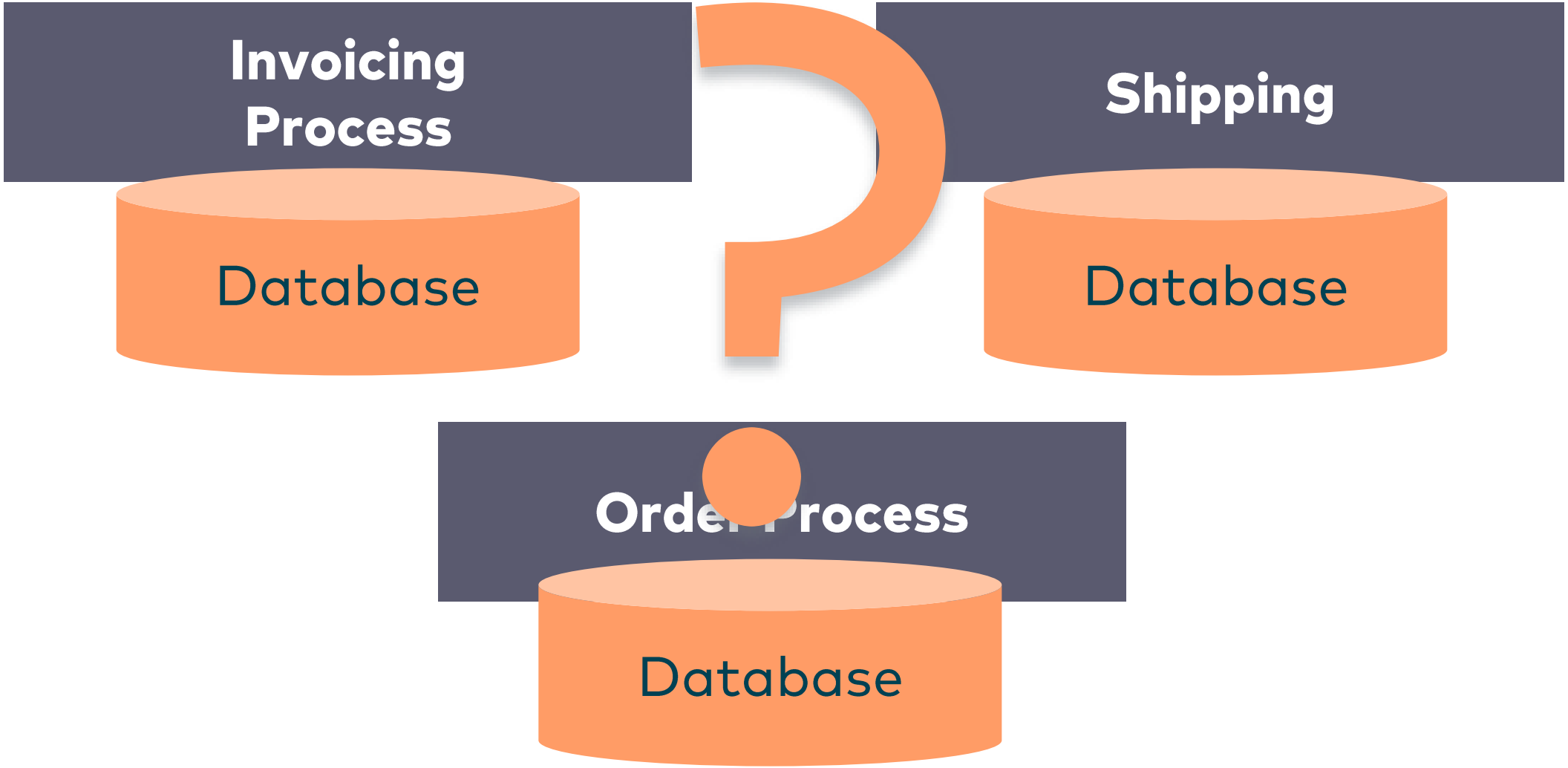
https://software-architektur.tv/2020/07/14/folge006.html

Domain-driven Migration

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



Bounded Contexts: Really the Goal?



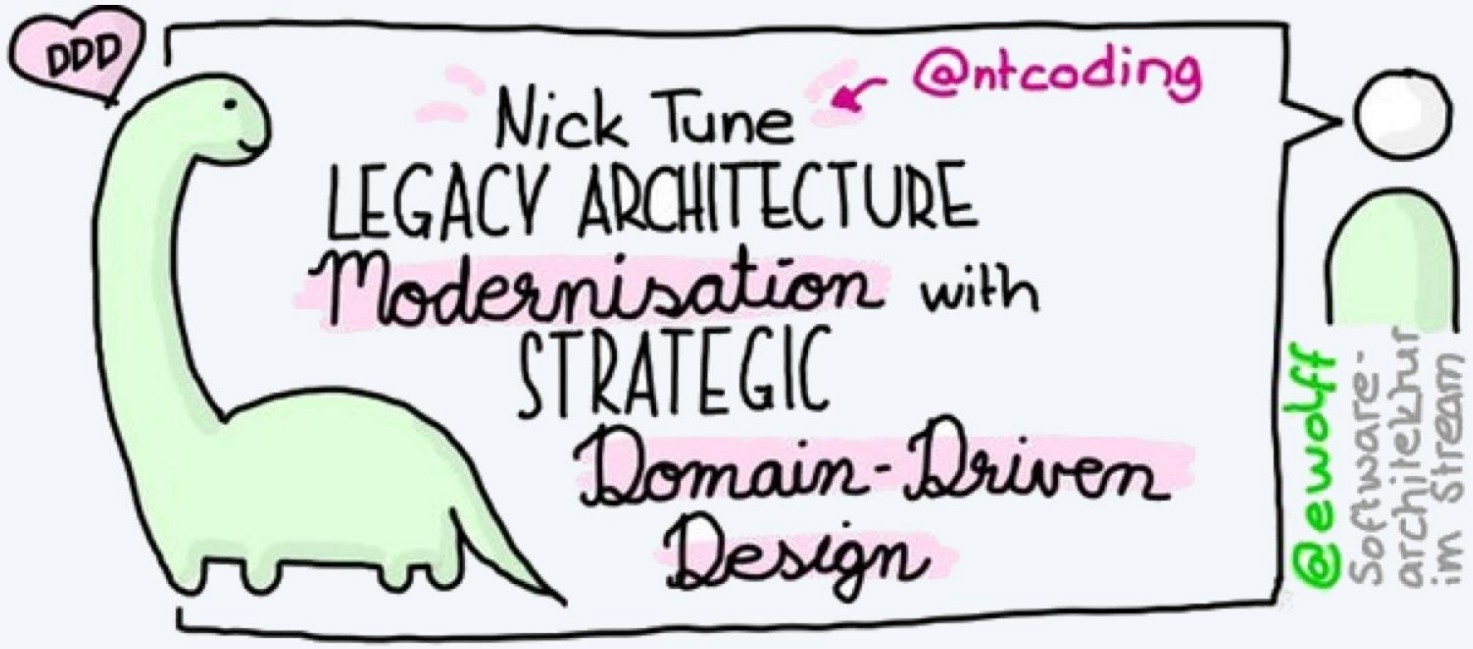
Domain-driven Migration

- 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?

#SoftwareArchitektur



https://software-architektur.tv/2020/08/07/folge011.html

Migration

- Folge 149 - Das Strangler Fig Pattern
- Folge 143 - Architektur-Migration (nicht nur) zu Microservices
- Folge 99 - Sam Newman - Monolith to Microservices
- 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"

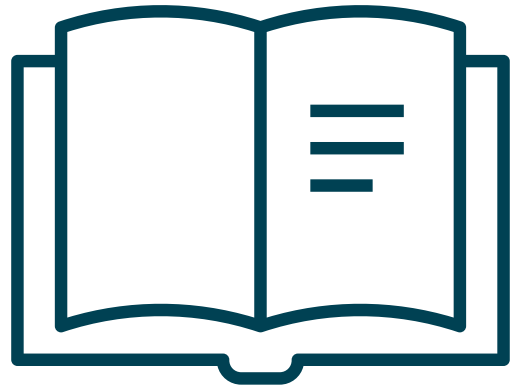
https://software-architektur.tv/tags.html#Migration

Iterations

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

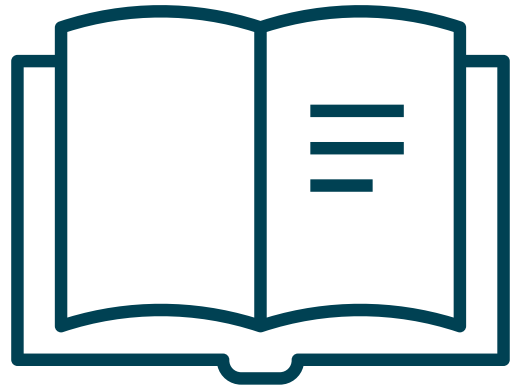
Eric Evans

A True Story



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

A True Story: The Start

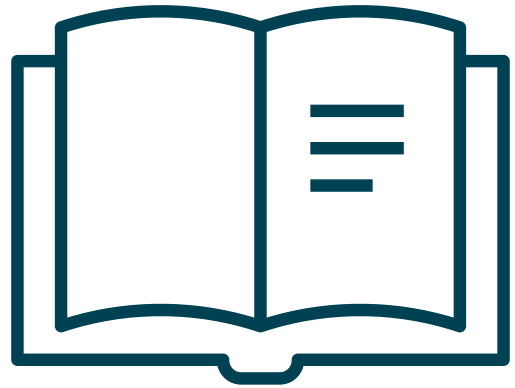


- Project start
- Learn more about the domain
- Migration by module makes it impossible

...to improve support for business.

...to improve automation

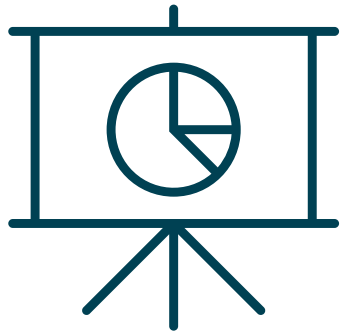
A True Story: Result



- There were other issues, too.
- Project cancelled

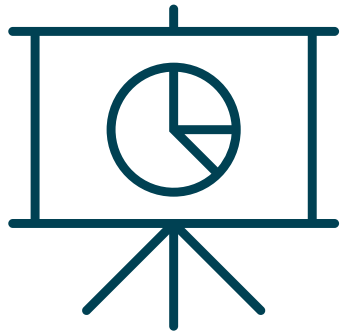
...and considered a failure.

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.

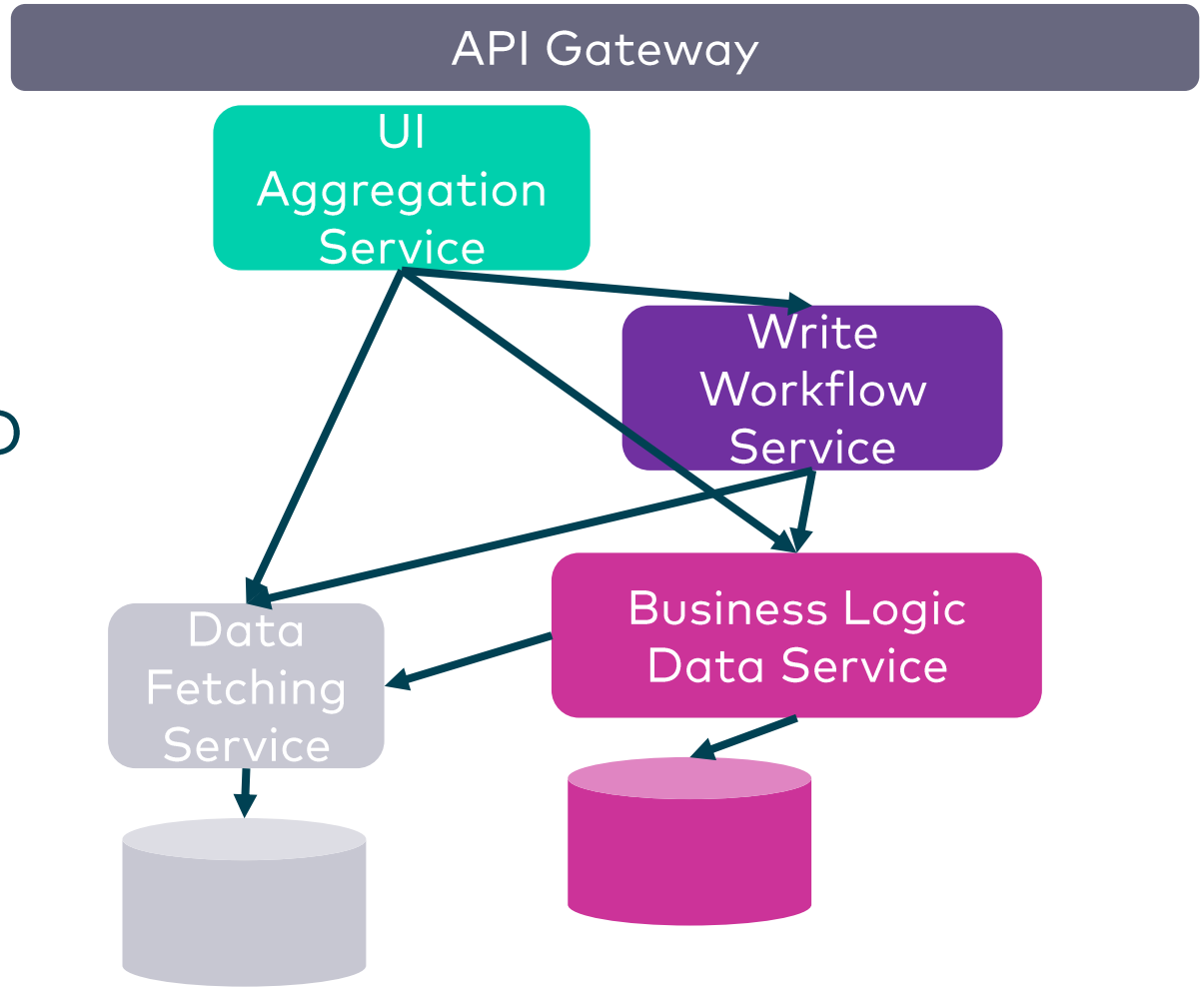
Recommended Lesson Learned



- 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!

Is This a Great Architecture?

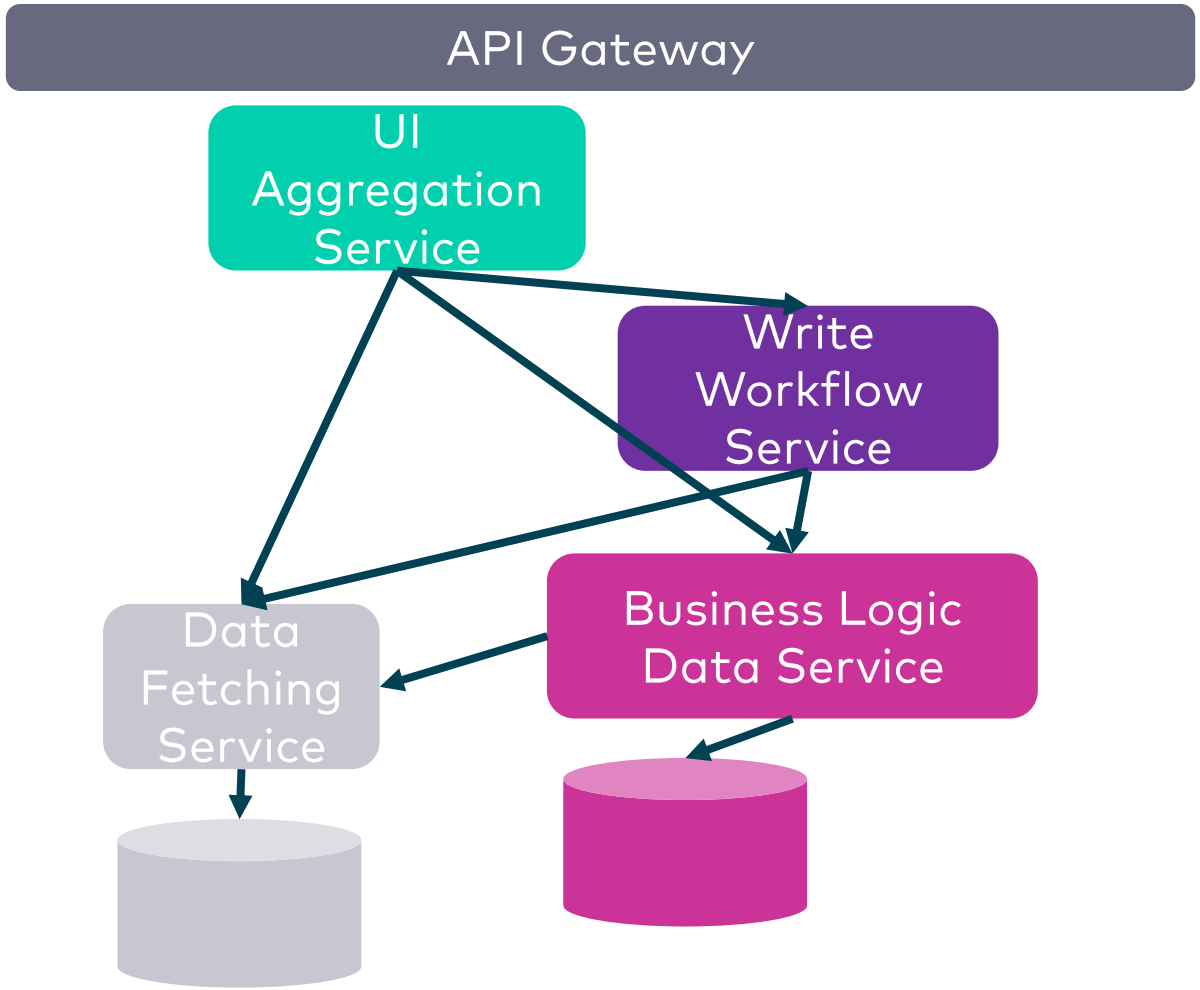
We are using all the tactical DDD pattern like Service, Repository, ...



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

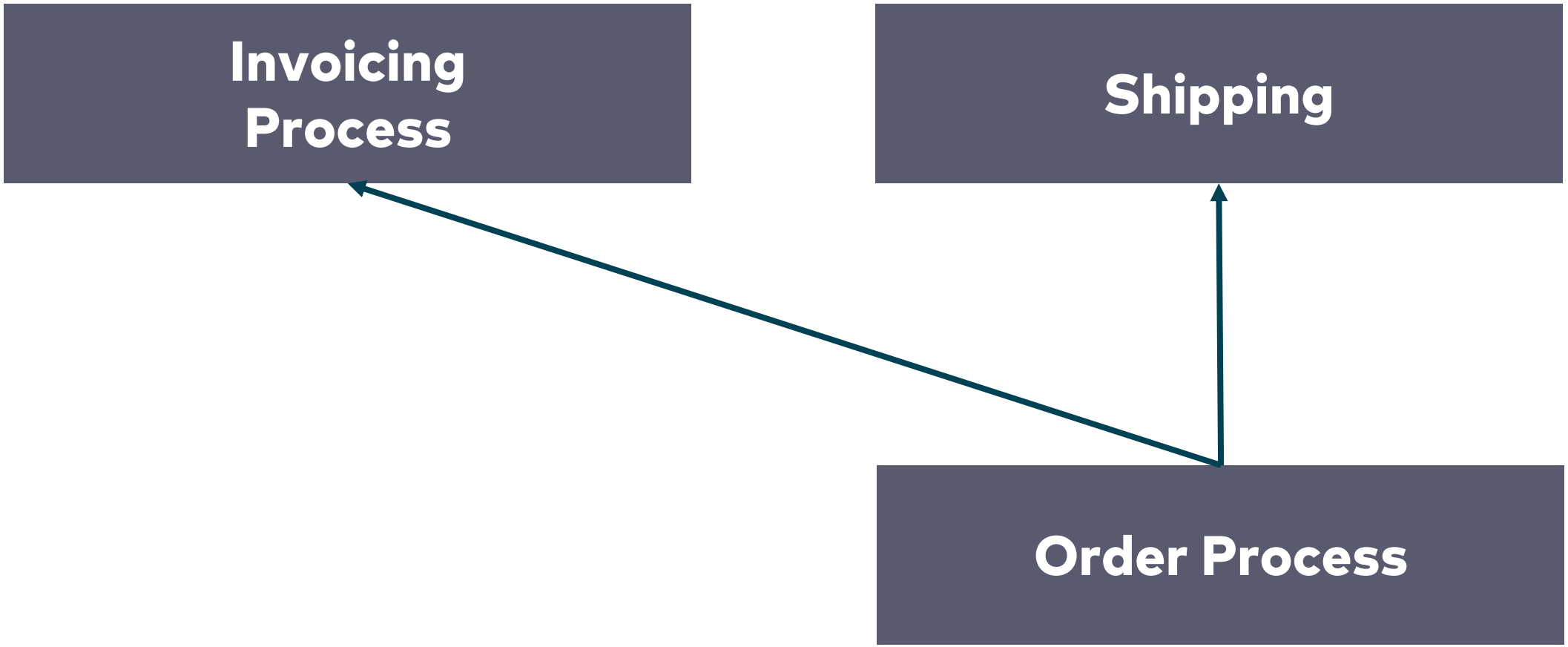
How to Detect nDDD

- 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



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

Conclusion

Iterations

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

Conclusion: DDD vs nDDD

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

Send email to itn2023@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