Please

# Ask questions through the app

*Rate Session*

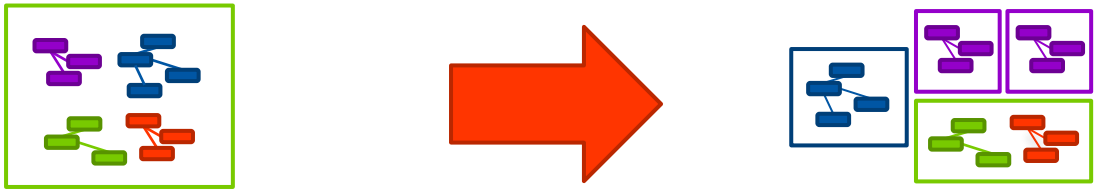
Thank you!

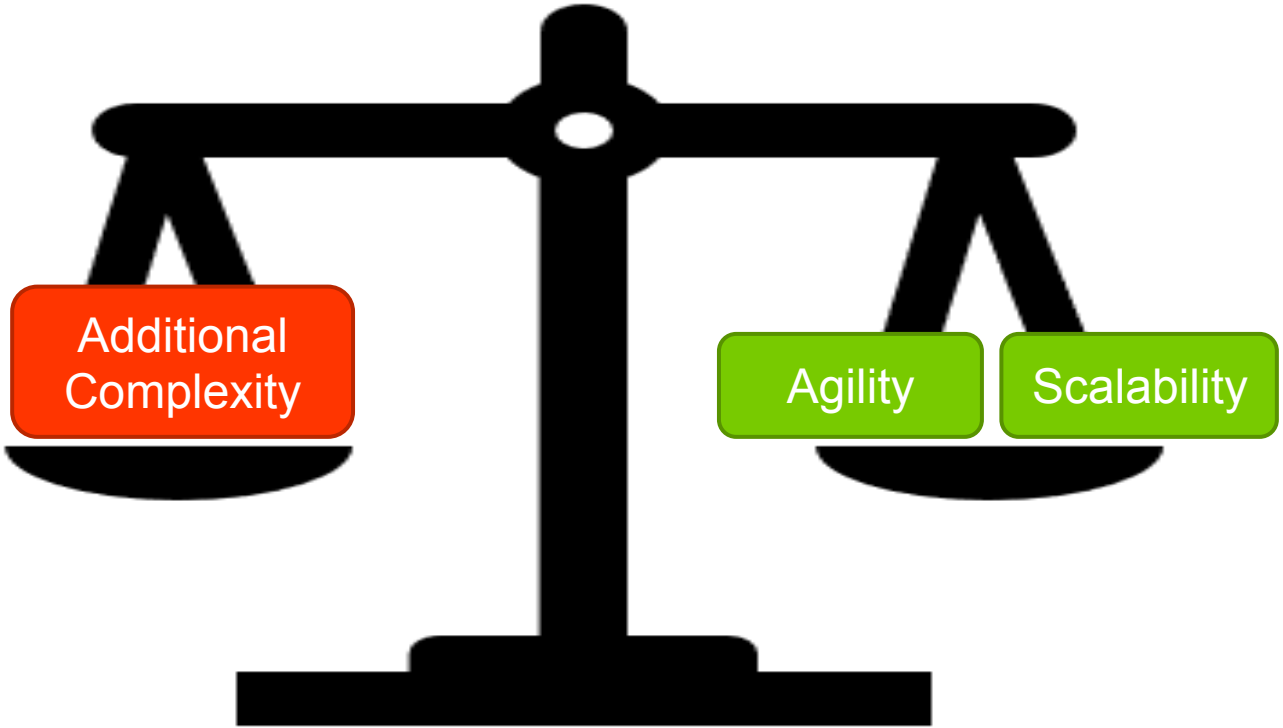
# Building microservices systems with the Axon platform

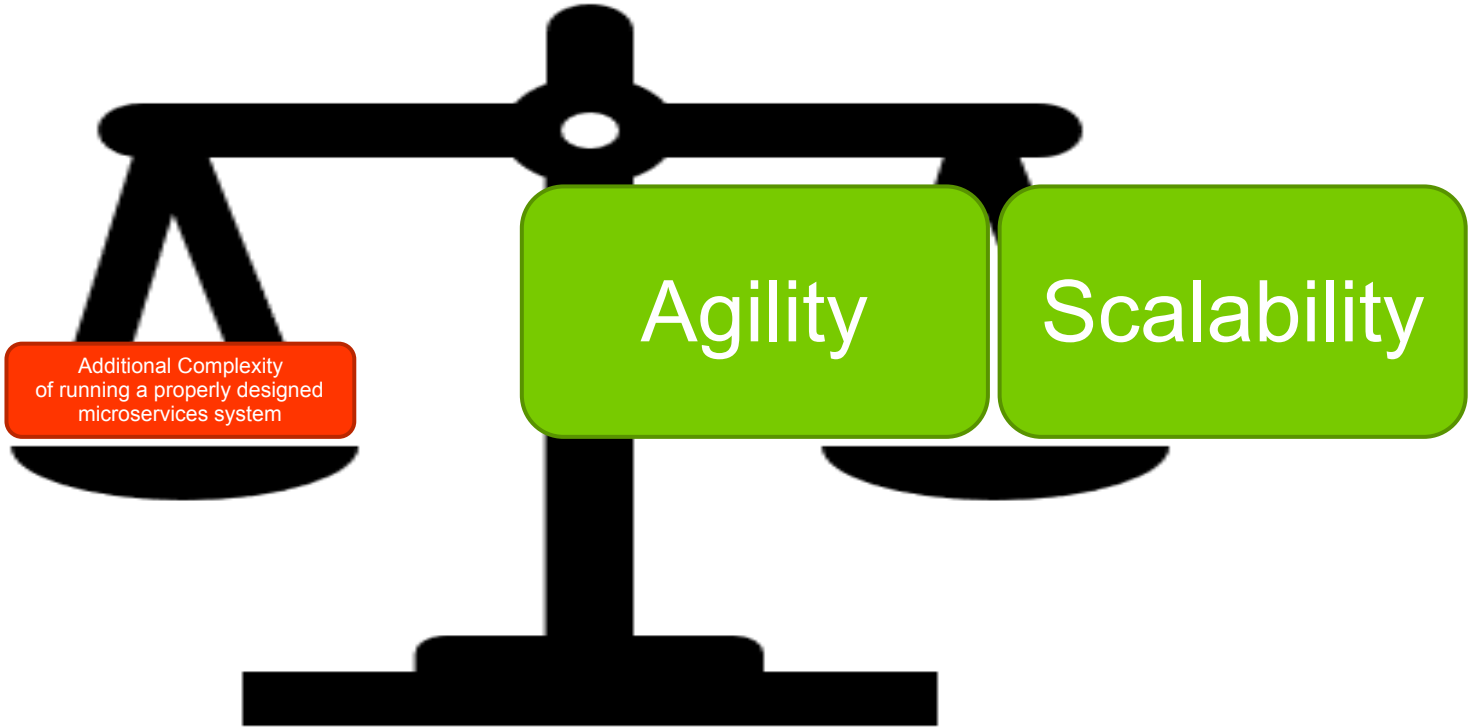
Frans van Buul
Evangelist @ AxonIQ

AxonIQ frans.vanbuul@axoniq.io @Frans_vanBuul

# Setting the stage: Microservices




Additional Complexity
Agility
Scalability

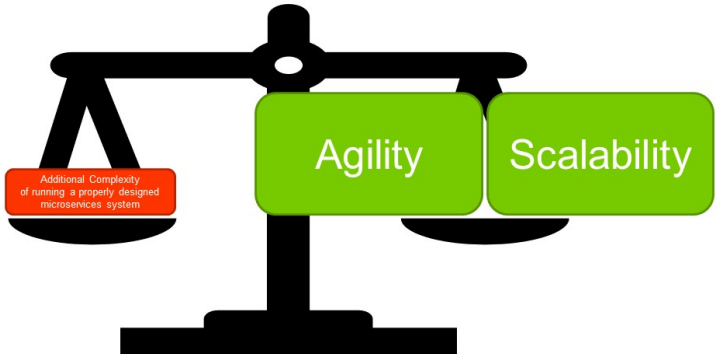

Additional Complexity
of running a properly designed
microservices system
Agility
Scalability


Additional
complexity
of running a
**"distributed monolith"**


frans.vanbuul@axoniq.io
@Frans_vanBuul

# Getting to to good place

- Ensure you cut the right boundaries between microservices.
- Start developing a structured monolith, evolve towards microservices.

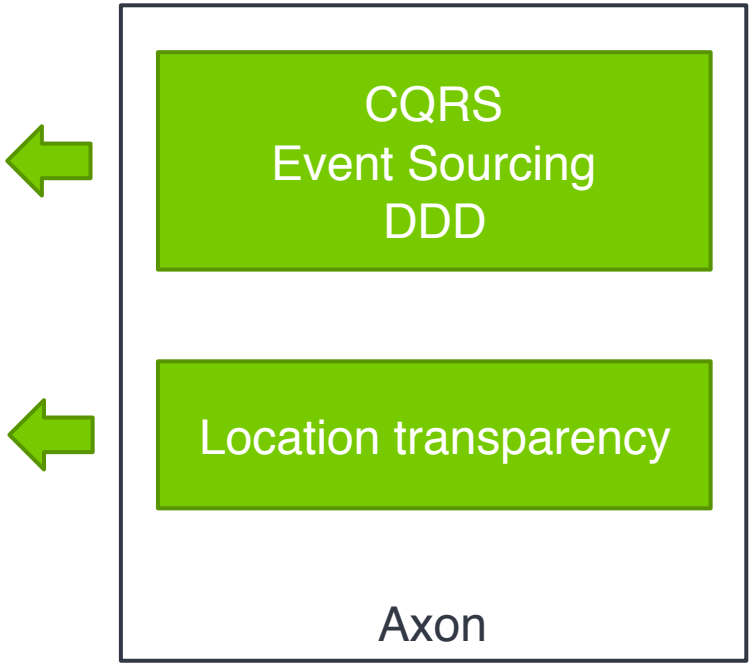




frans.vanbuul@axoniq.io
@Frans_vanBuul

# Getting to the good place

- Ensure you cut the right boundaries between microservices.
- Start developing a structured monolith, evolve towards microservices.

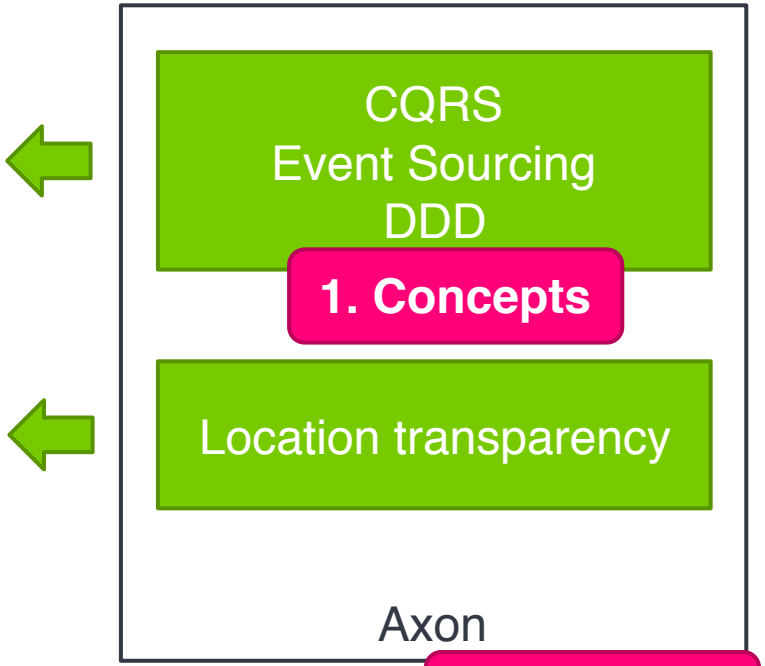




frans.vanbuul@axoniq.io
@Frans_vanBuul

# Agenda

- Ensure you cut the right boundaries between microservices.
- Start developing a structured monolith, evolve towards microservices.



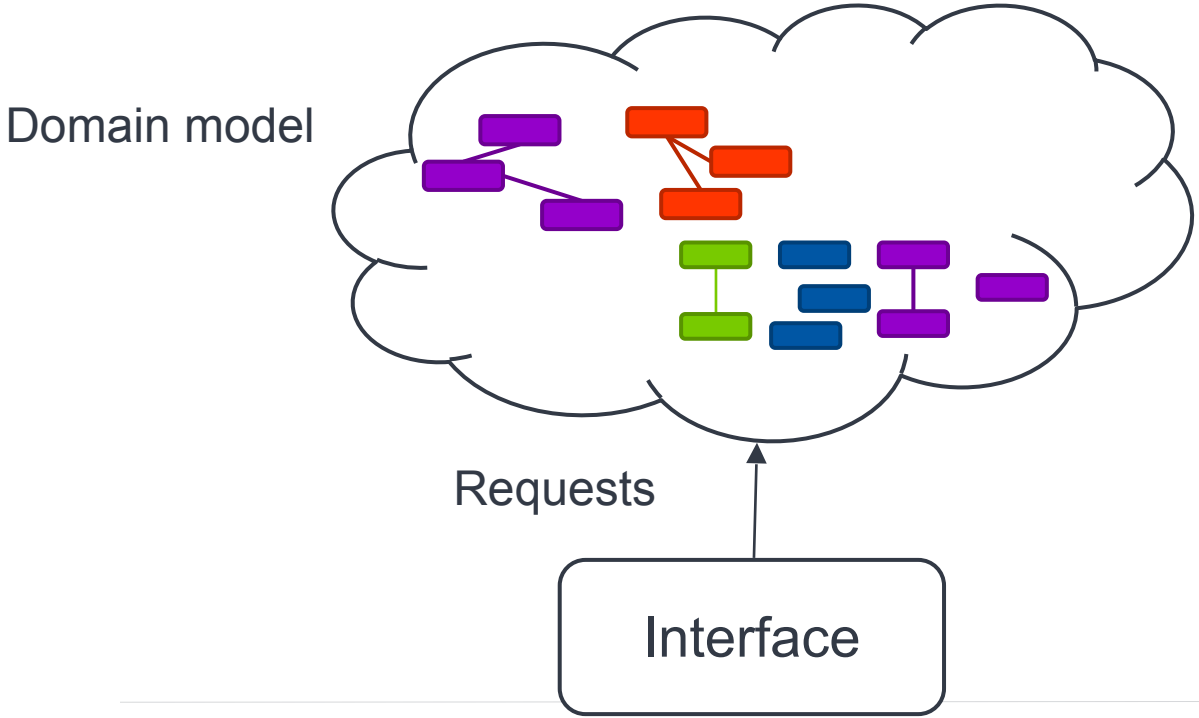
# Concepts

# CQRS

## Command Query Responsibility Segregation

AxonIQ frans.vanbuul@axoniq.io @Frans_vanBuul

# Non-CQRS

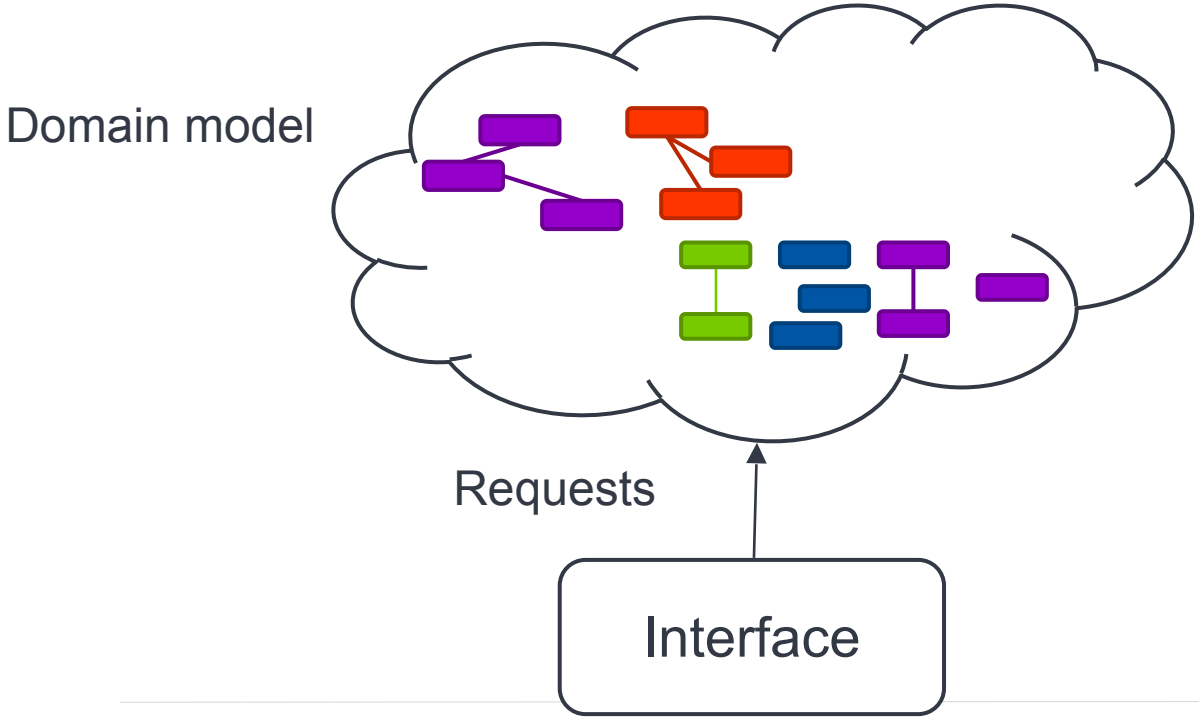



Request may express

- An intent to change/do something
- A inquiry for information
- A mixture


frans.vanbuul@axoniq.io @Frans_vanBuul

# Non-CQRS




Request may express:

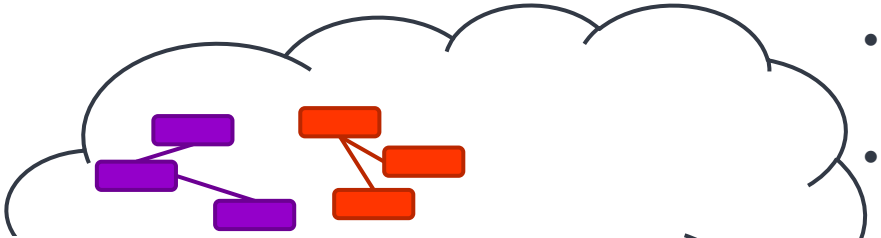
- An intent to change/do something
- A inquiry for information
- A mixture

Model tends to be:

- Optimized for writing/ maintaining integrity
- By keeping it normalized and avoiding redundancy

# Non-CQRS

Domain model



```
SELECT TaxonName2.FullTaxonName, Locality.Latitude,
Locality.Longitude FROM (CollectionObject) INNER JOIN
(((CollectionObject AS CollectionObject2) INNER JOIN ((Determination
AS Determination2) INNER JOIN (TaxonName AS TaxonName2) ON
TaxonName2.TaxonNameID = Determination2.TaxonNameID) ON
Determination2.BiologicalObjectID =
CollectionObject2.CollectionObjectID) INNER JOIN ((CollectingEvent
AS CollectingEvent2) INNER JOIN (Locality) ON Locality.LocalityID =
CollectingEvent2.LocalityID) ON CollectingEvent2.CollectingEventID =
CollectionObject2.CollectingEventID) ON
CollectionObject2.CollectionObjectID =
CollectionObject.CollectionObjectID  WHERE
(((((TaxonName2.FullTaxonName IN('Rana blairi', 'Rana
catesbeiana'))))) AND (((Locality.Latitude BETWEEN 37.8 AND 39.4
AND Locality.Longitude BETWEEN 94.8 AND 96.0))))
```

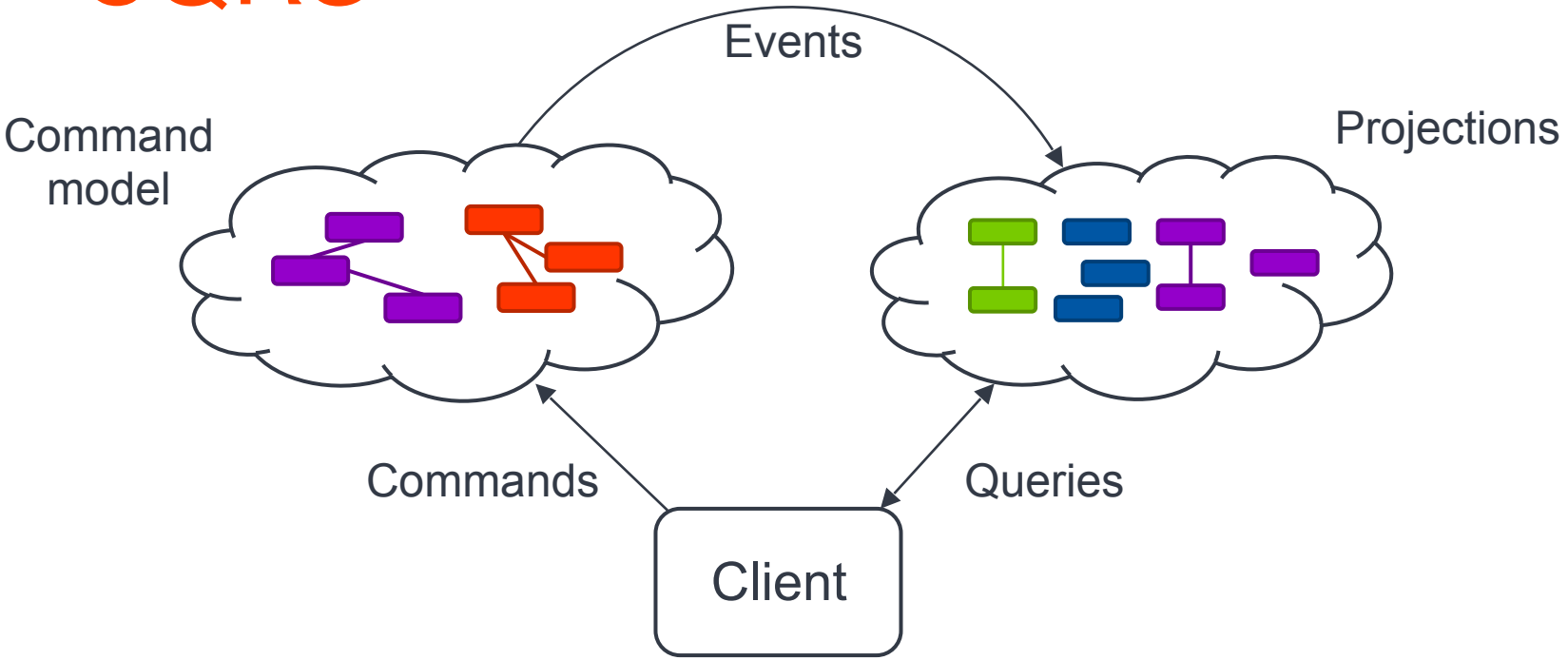
Queries (just retrieving info)

- need to combine data from many objects
- as a result, are likely to become complex
  - expensive to execute
  - hard to maintain

# Core CQRS ideas

- Queries could be simplified by storing a copy of the data in a form easily queryable ("materialized view" or "projection").
- In many cases, updating the query models can happen asynchronously from processing the transaction ("eventually consistent").

# CQRS

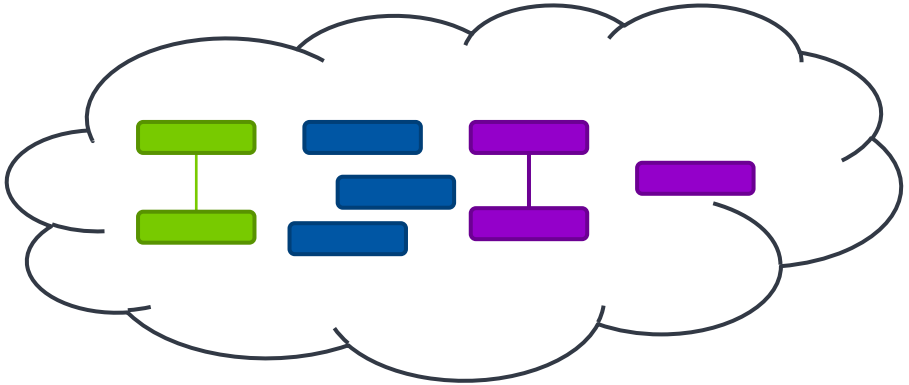

Events
Command model
Projections
Commands
Queries
Client

# Read models (a.k.a. "projections")

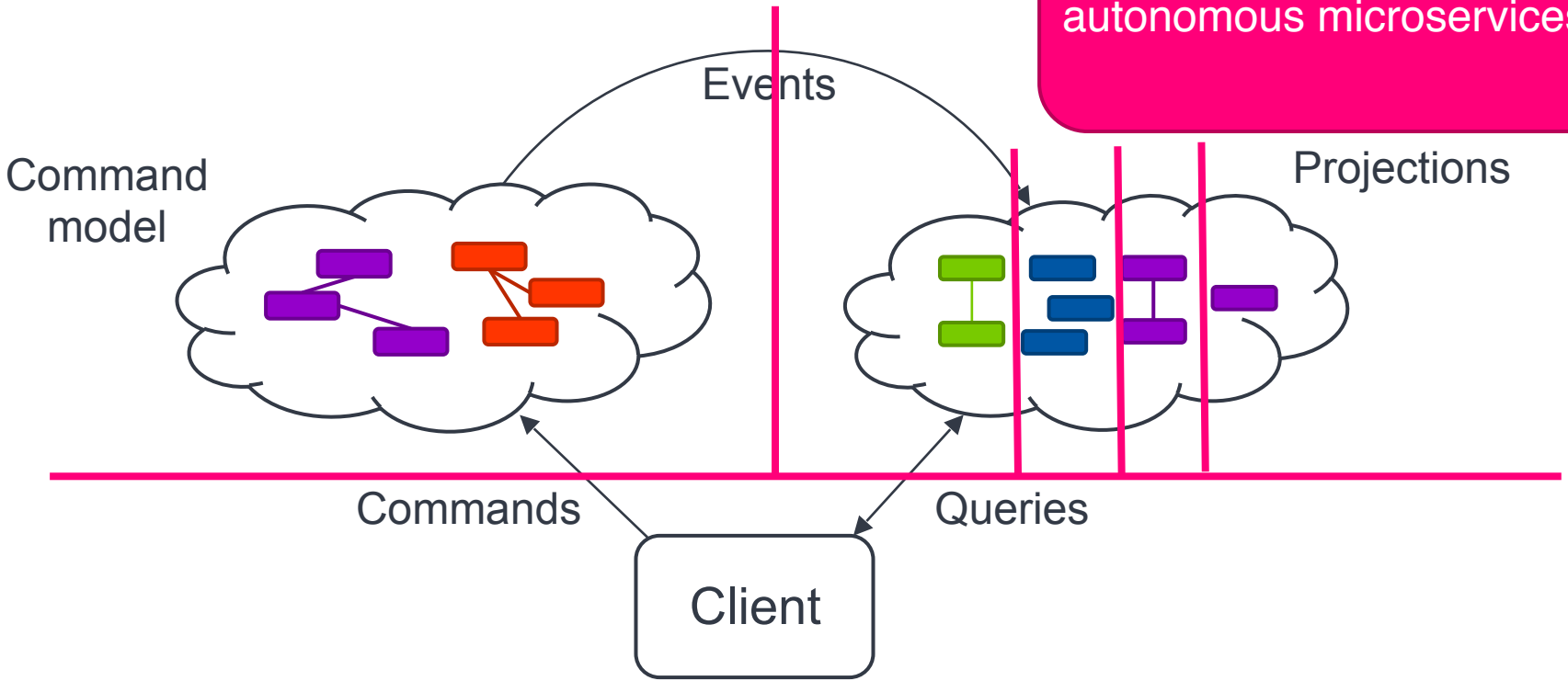
Optimized for the specific read use-cases (e.g. screens, API methods)

Many separated ones instead of one big one.

Use various technologies (RDBMS, Elastic, NoSQL etc.)



Projections are natural, autonomous microservices!



# Domain-Driven Design (DDD)
# "Aggregates"

AxonIQ frans.vanbuul@axoniq.io @Frans_vanBuul

# Domain-Driven Design

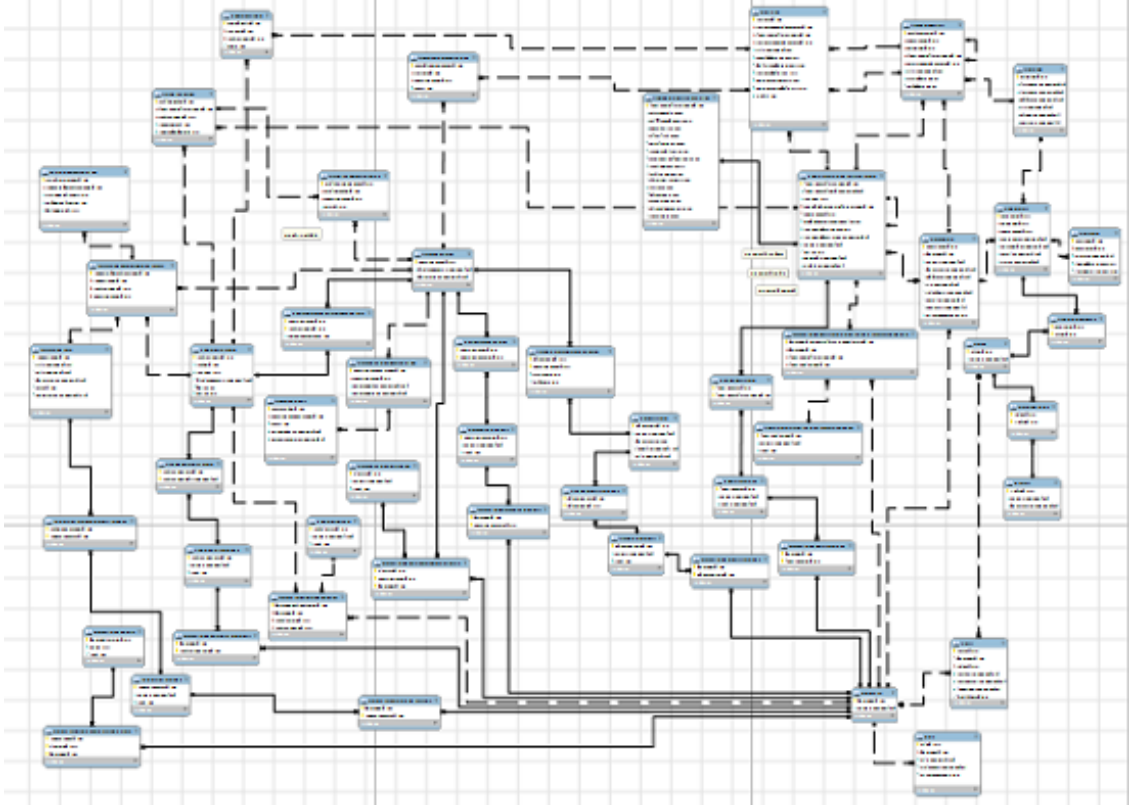
**Strategic level**

- Make software containing explicit models that represent domains.
- Tackle complex business domains.

**Building blocks**

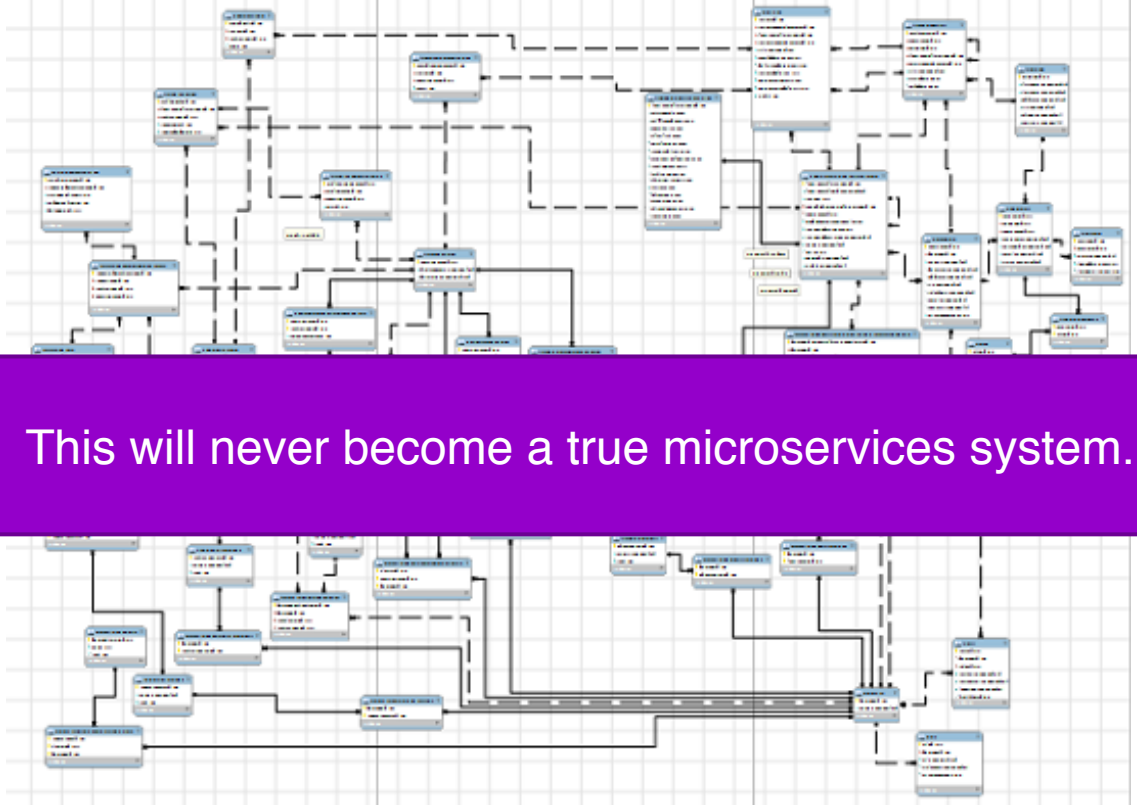
- **Aggregate**
- Entity vs. Value Object
- Repository
- Service
- Anti-Corruption Layer
- ……


frans.vanbuul@axoniq.io @Frans_vanBuul



An entity-relationship diagram can get pretty large if you are building a complex application.
Some contain hundreds or even thousands of tables.

*Source: http://en.tekstenuitleg.net/articles/software/database-design-tutorial/creating-an-entity-relationship-diagram-(erd)*



This will never become a true microservices system.

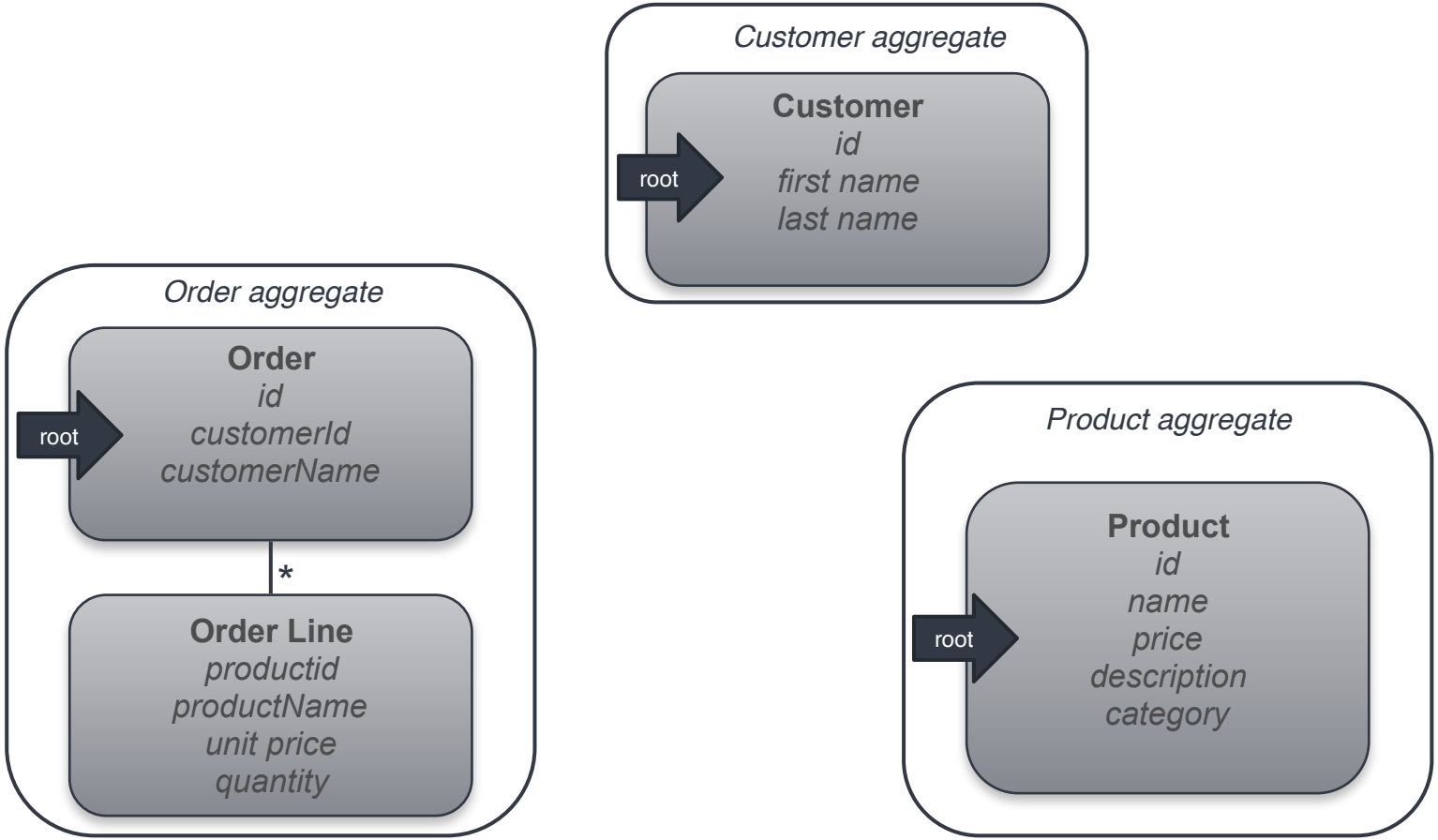
An entity-relationship diagram can get pretty large if you are building a complex application.
Some contain hundreds or even thousands of tables.

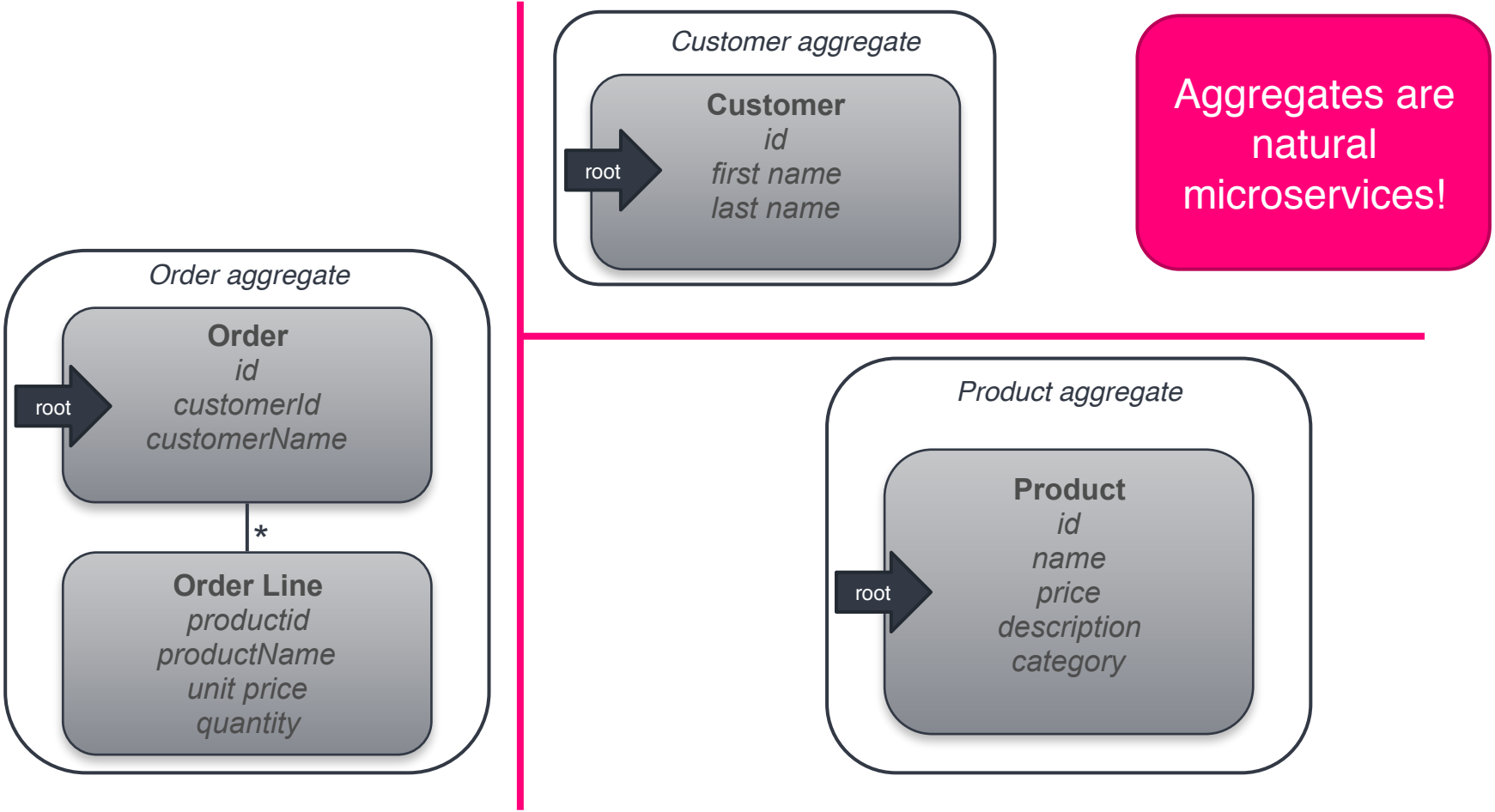
*Source: http://en.tekstenuitleg.net/articles/software/database-design-tutorial/creating-an-entity-relationship-diagram-(erd)*

# Thinking in *aggregates*

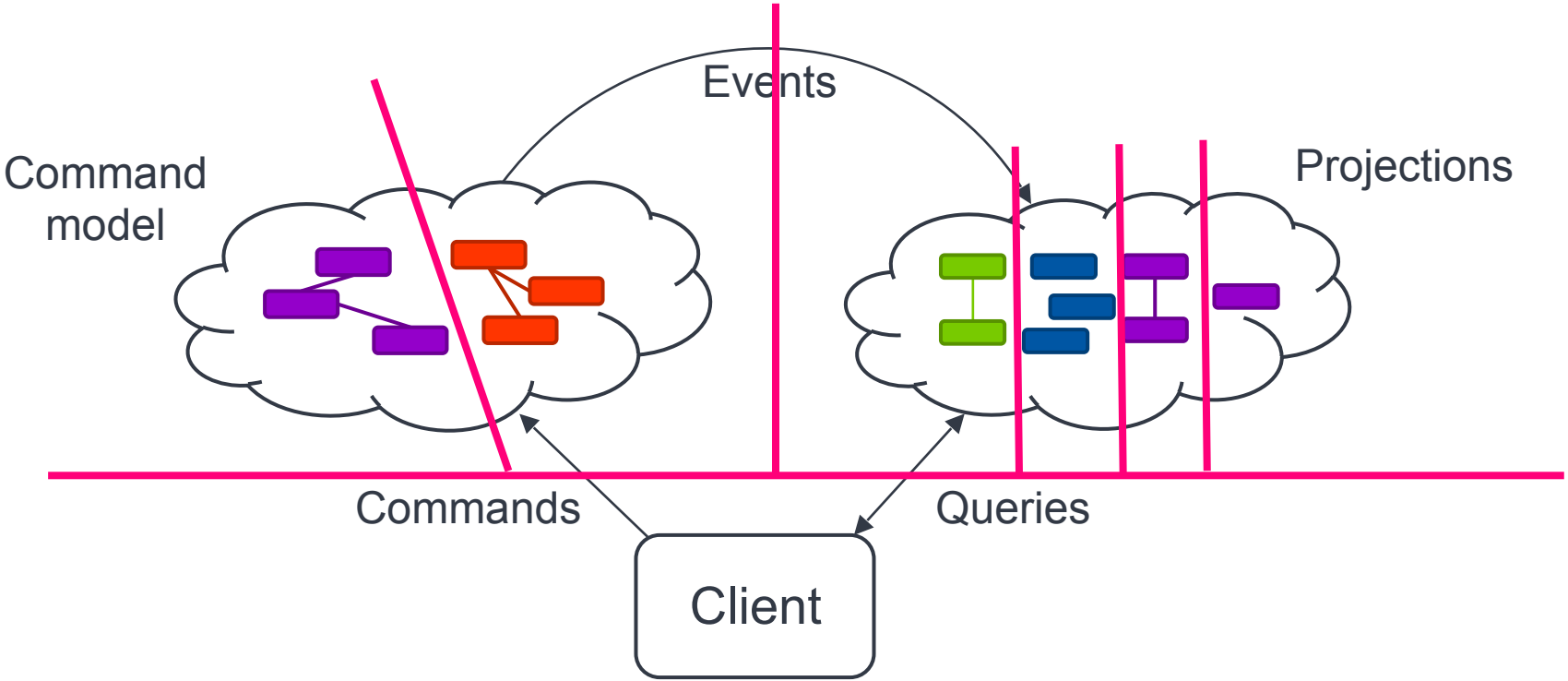
- Start thinking about your domain model in Java (not tables).
- Do *not*
  - connect everything to everything
  - create classes with just getters/setters
- Instead, design aggregates:
  - Smallish object graphs considered a unit for persistence/consistency.
  - Responsible for their own consistency, and able to enforce it. Have business logic!
  - Often contain copies of data stored in other aggregates.
  - May indirectly refer to other aggregates.


frans.vanbuul@axoniq.io @Frans_vanBuul


Customer aggregate
Customer
id
first name
last name
root
Order aggregate
Order
id
customerId
customerName
root
*
Order Line
productid
productName
unit price
quantity
Product aggregate
Product
id
name
price
description
category
root


Customer aggregate
Customer
id
first name
last name
root
Aggregates are natural microservices!
Order aggregate
Order
id
customerId
customerName
root
*
Order Line
productid
productName
unit price
quantity
Product aggregate
Product
id
name
price
description
category
root

DDD aggregates + CQRS give great guidance for potential microservices boundaries





frans.vanbuul@axoniq.io
@Frans_vanBuul

# Event Sourcing

AxonIQ frans.vanbuul@axoniq.io @Frans_vanBuul

# How to store aggregates?

**Regular way**

- Persist state in database (CRUD).
- Often using done using an ORM framework.

**Event sourcing way**

- Store a history of everything that happened.
- Calculate 'current state' based on that history when needed.


frans.vanbuul@axoniq.io @Frans_vanBuul

# Sounds weird?
# Think about bank accounts

**"Regular" way**

- UPDATE the balance of an account when a payment is made.
- Overwrite the previous values.
- Want to know what happened? Check the logs.

**Event sourcing way**

- When a payment is made, store the details of that transaction as such.
- Calculate the balance as a *derived value*

***This is the "normal" way!***

Event Sourcing

=

*treating any aggregate like a bank account (storing events rather than outcome), even when it models something else*


frans.vanbuul@axoniq.io @Frans_vanBuul

# Why use event sourcing?

## Business reasons

- Auditing / compliance / transparency
- Data mining, analytics: value from data

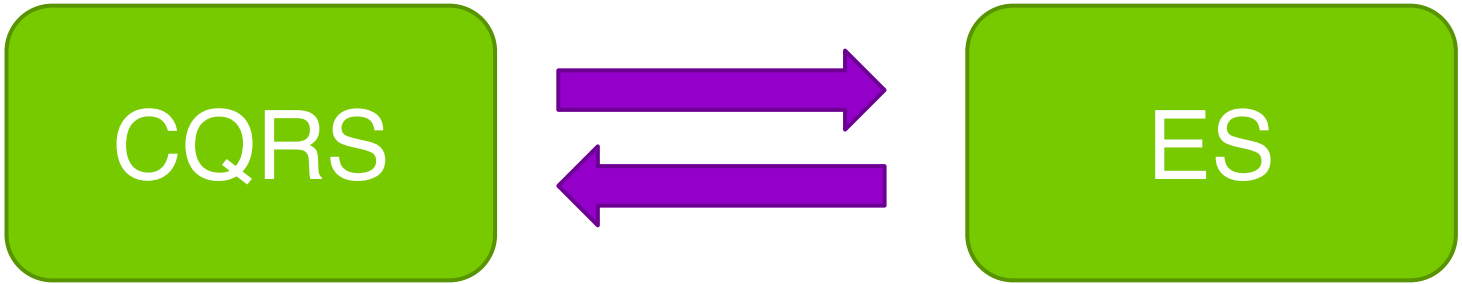

## Modeling+technical reasons

- Single source of truth
- Easily capturing intent
- Natural match with business processes
- Guaranteed completeness of raised events
- No need for separate log statements
- Replay into new read models (CQRS)

ns.vanbuul@axoniq.io @Frans_vanBuul

# Better together!

CQRS provides a way to efficiently read information from (multiple) event-sourced aggregates




ES provides

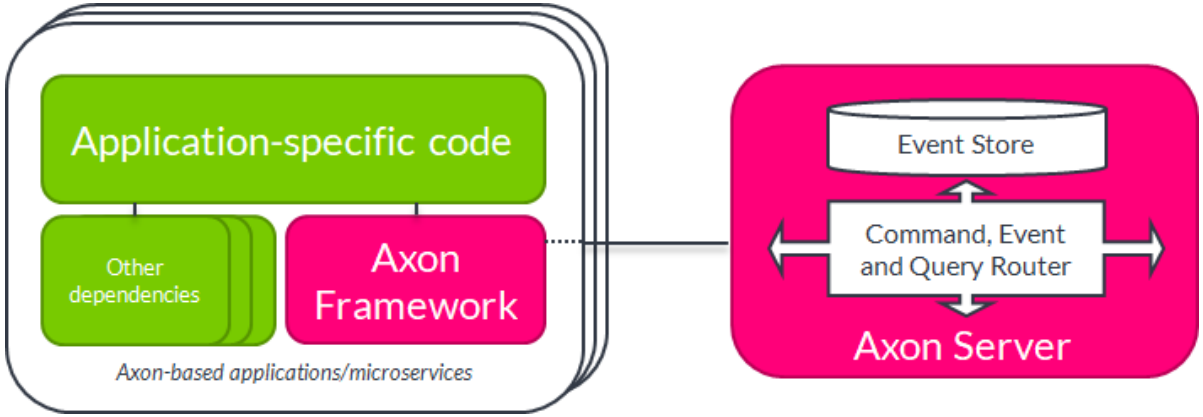
- A standard way to asynchronously update CQRS query models.
- An easy way to initialize new query models

# Axon

platform to create microservices using CQRS, ES and DDD in Java

AxonIQ frans.vanbuul@axoniq.io @Frans_vanBuul

# Axon = Axon Framework + Axon Server




Open source Java framework (Apache2)

Implements DDD, CQRS, ES, location transparency

Built-for-purpose event store and messaging platform

Open core

Optional for Axon Framework


frans.vanbuul@axoniq.io @Frans_vanBuul

# Live coding domain: Gift Cards

Source: https://en.wikipedia.org/wiki/Gift_card

- Gift cards get **issued** at a certain initial value.
- Each time they are used to buy something **("redeemed")** the remaining value is adjusted.

Do it yourself:

# https://axoniq.io/download

I'll push my code to:

# https://github.com
# /fransvanbuul/gotober2018

Please

# Remember to rate this session

Thank you!