DEVelopers Köln Meetup 20.11.2019

5 years of food retail e-commerce

A microservice success story

Sebastian Gauder
Software Architect
@rattakresch

REWE digital


Our history

Details REWE Group

Turnover
>61 bn

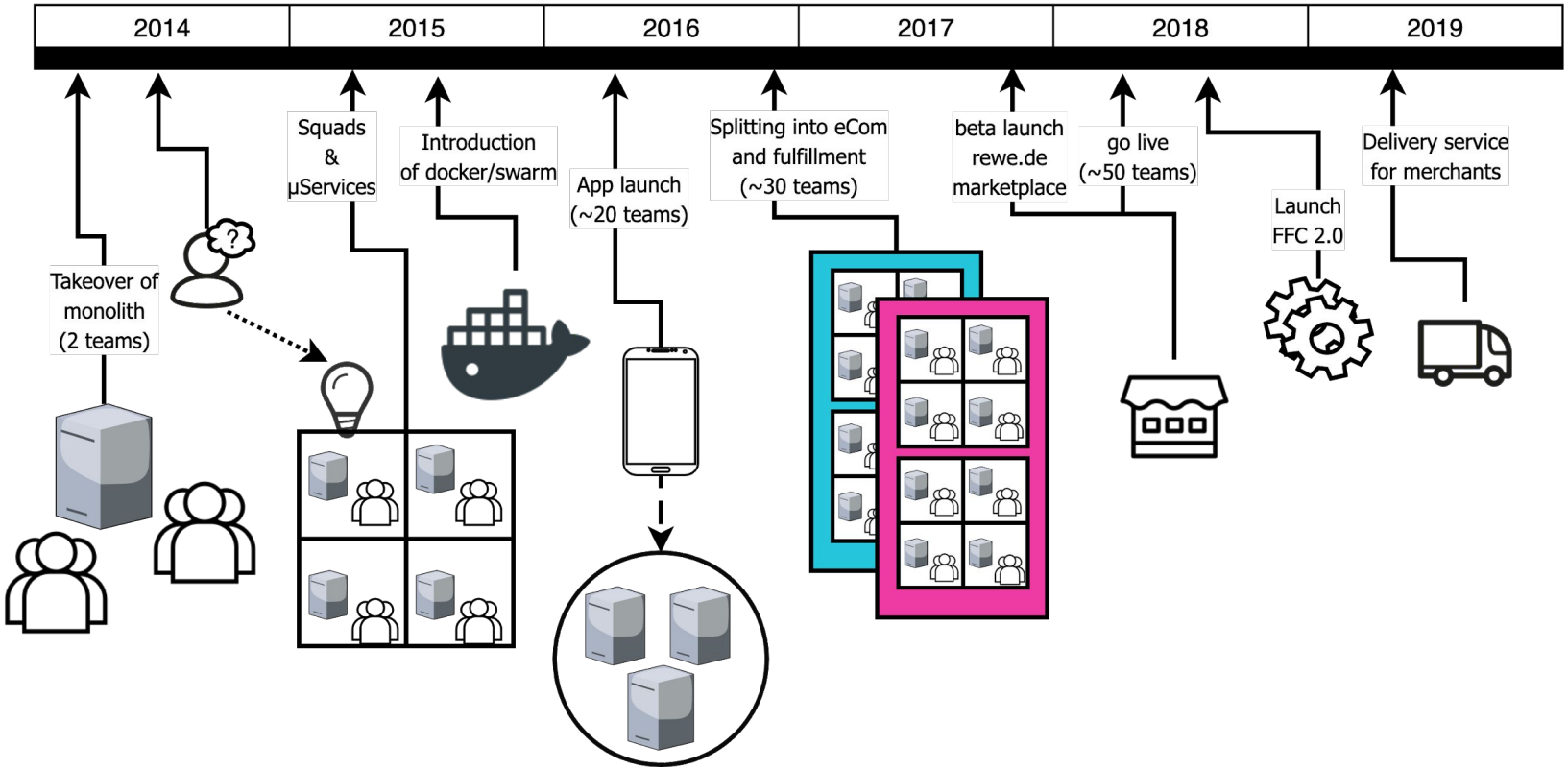
Employees
>360.000

Shops
>15.000

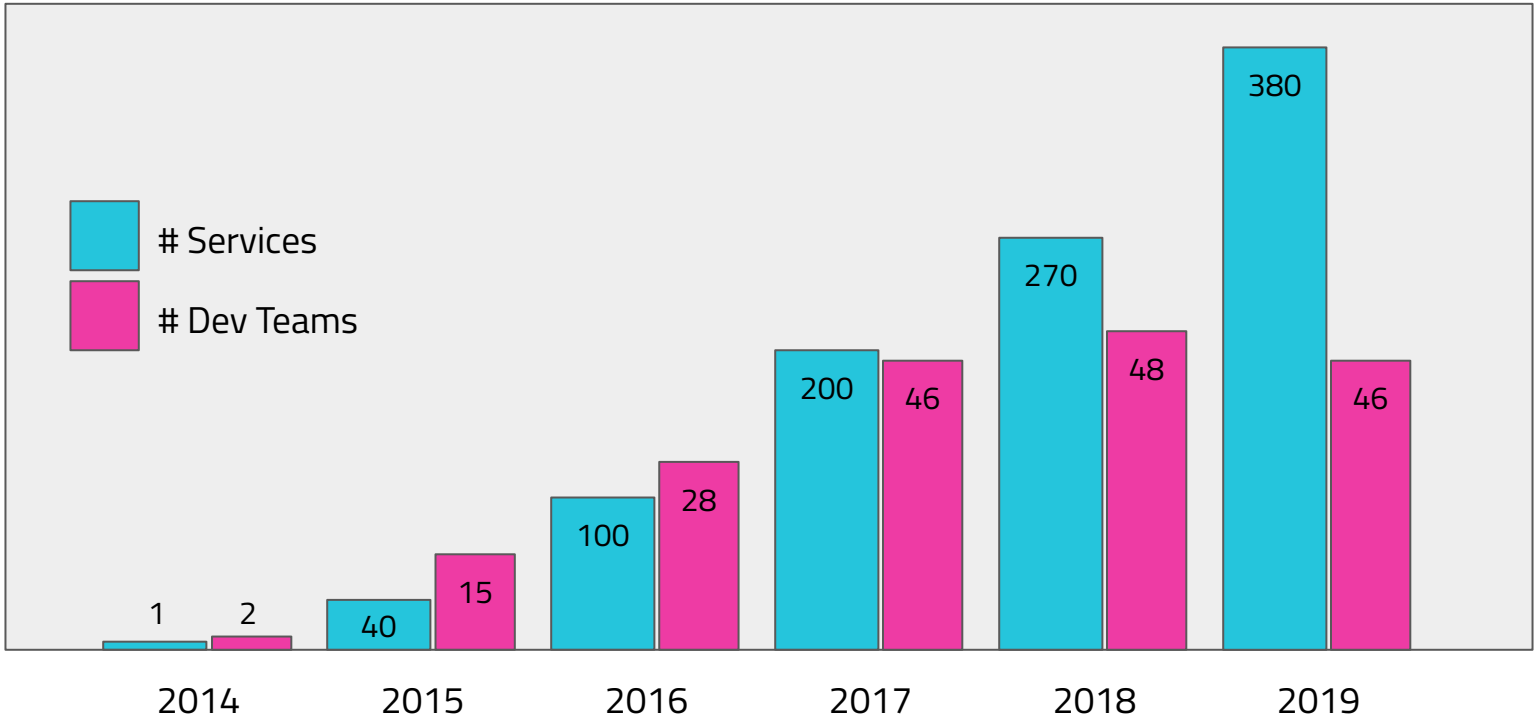
Industries
Food Retail, Tourism, DIY

History
>90 years

Our history - the overview



Service/team growth



Service/team growth


How to scale monoliths?

Design Goals

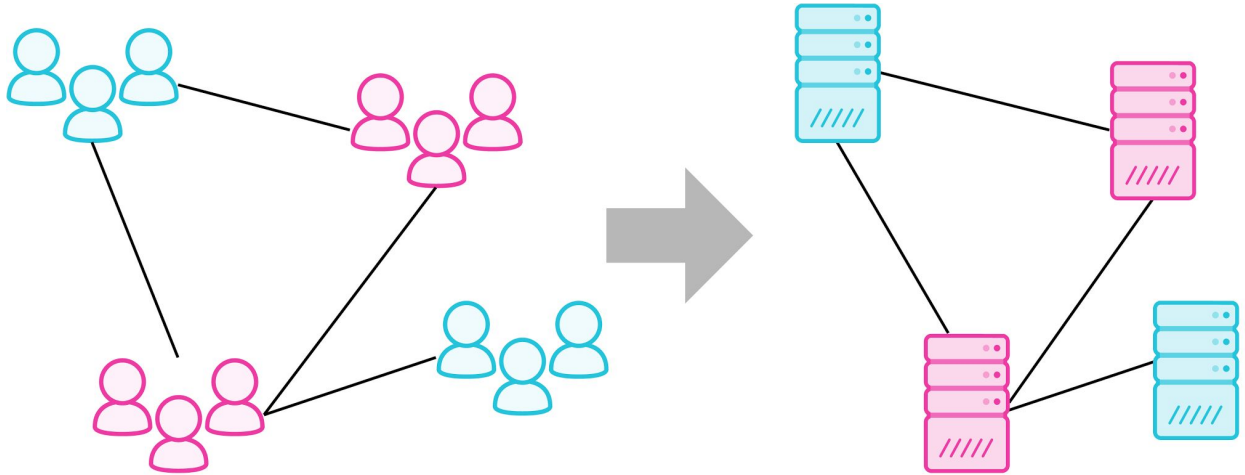
Decentralization

Vertical Boundaries

Conway's law

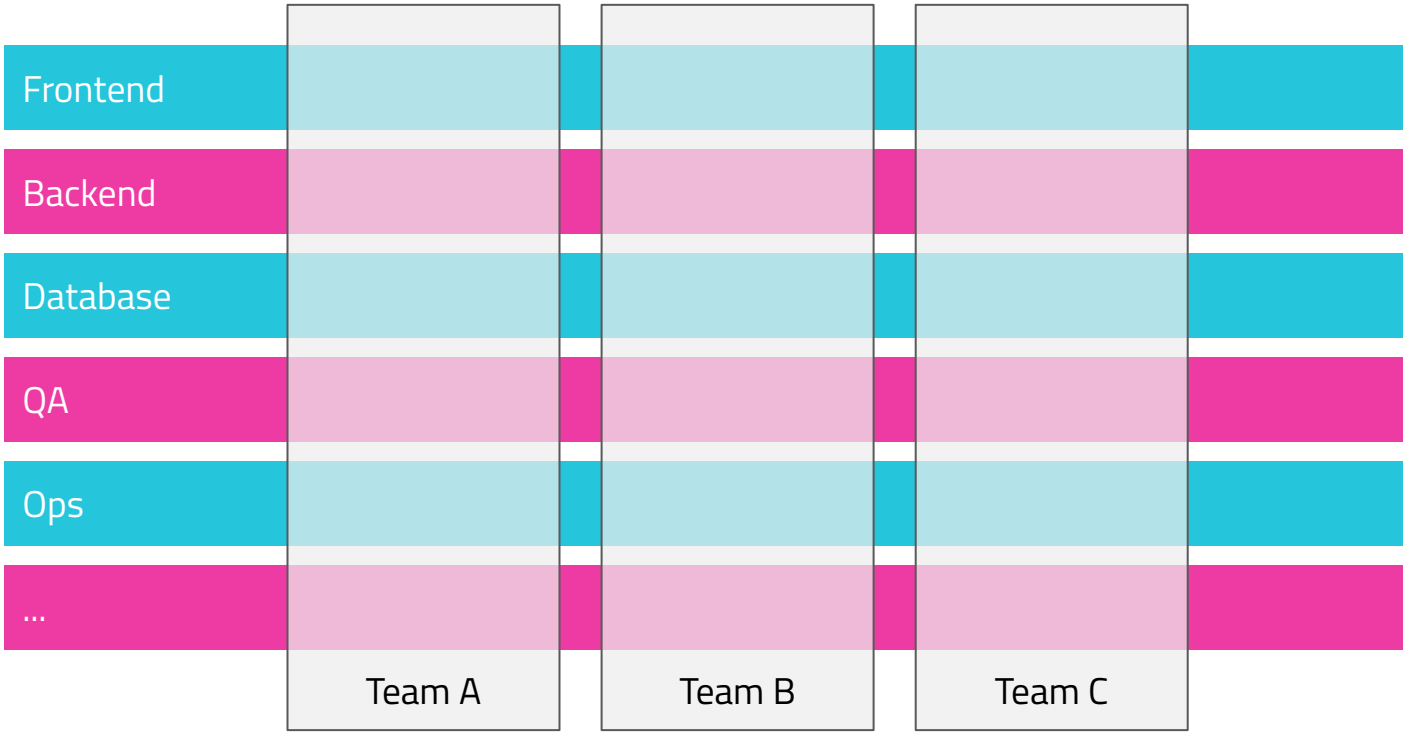
"Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure."

Melvin Conway (1967)



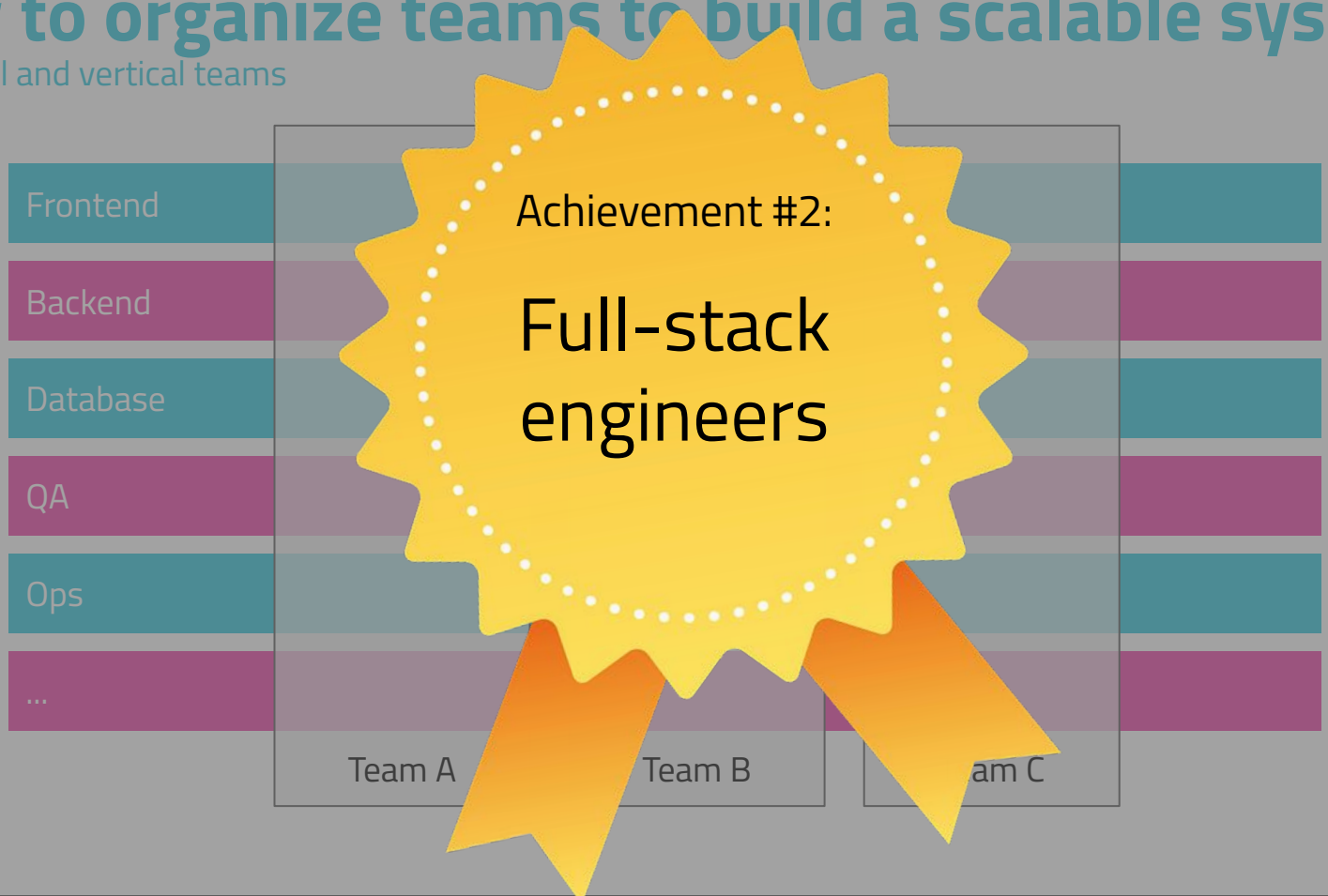
How to organize teams to build a scalable system?

Functional and vertical teams



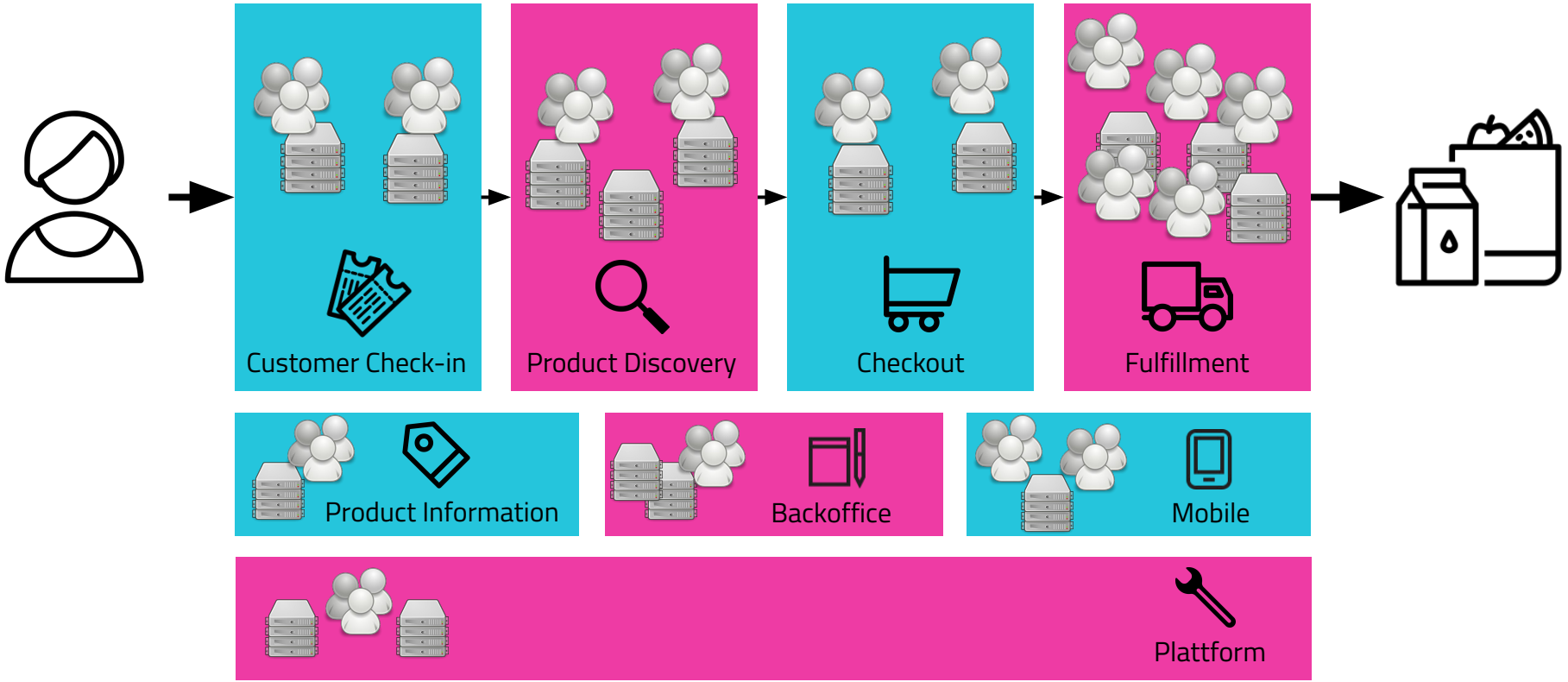
How to organize teams to build a scalable system?

Functional and vertical teams

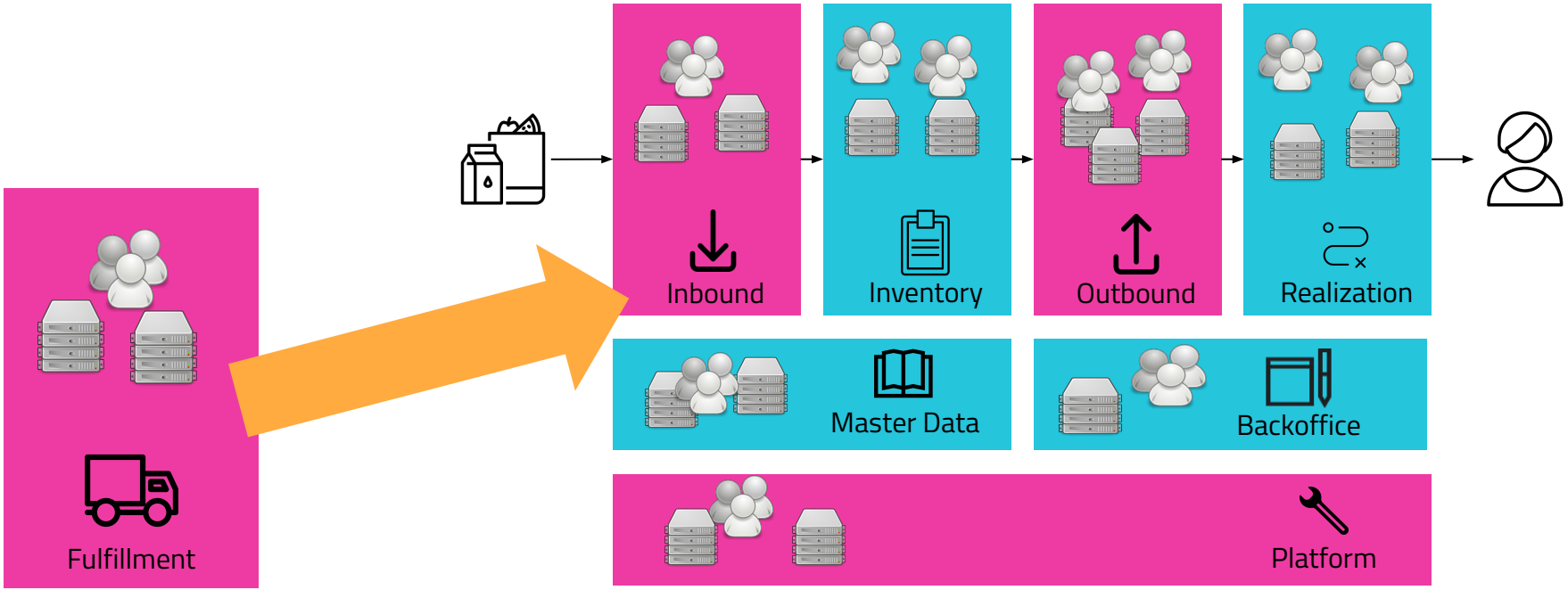


How to **determine boundaries?**

Customer journey defines subdomains



Fullfilment domain transformation



Fullfilment domain transformation


How to guide 150+ developers?

Guarding rails for developers

Design Goals

- Vertical boundaries
- Decentralization

Architectural principles

- Collection of 9 basic ‘laws’
- Autonomy, Automation and Communication

Guides

- Practical manual for common tasks (RFC 2119)
- e.g. Eventing, REST, Authentication

MUST
SHOULD
COULD

Guarding rails for developers

Design Goals

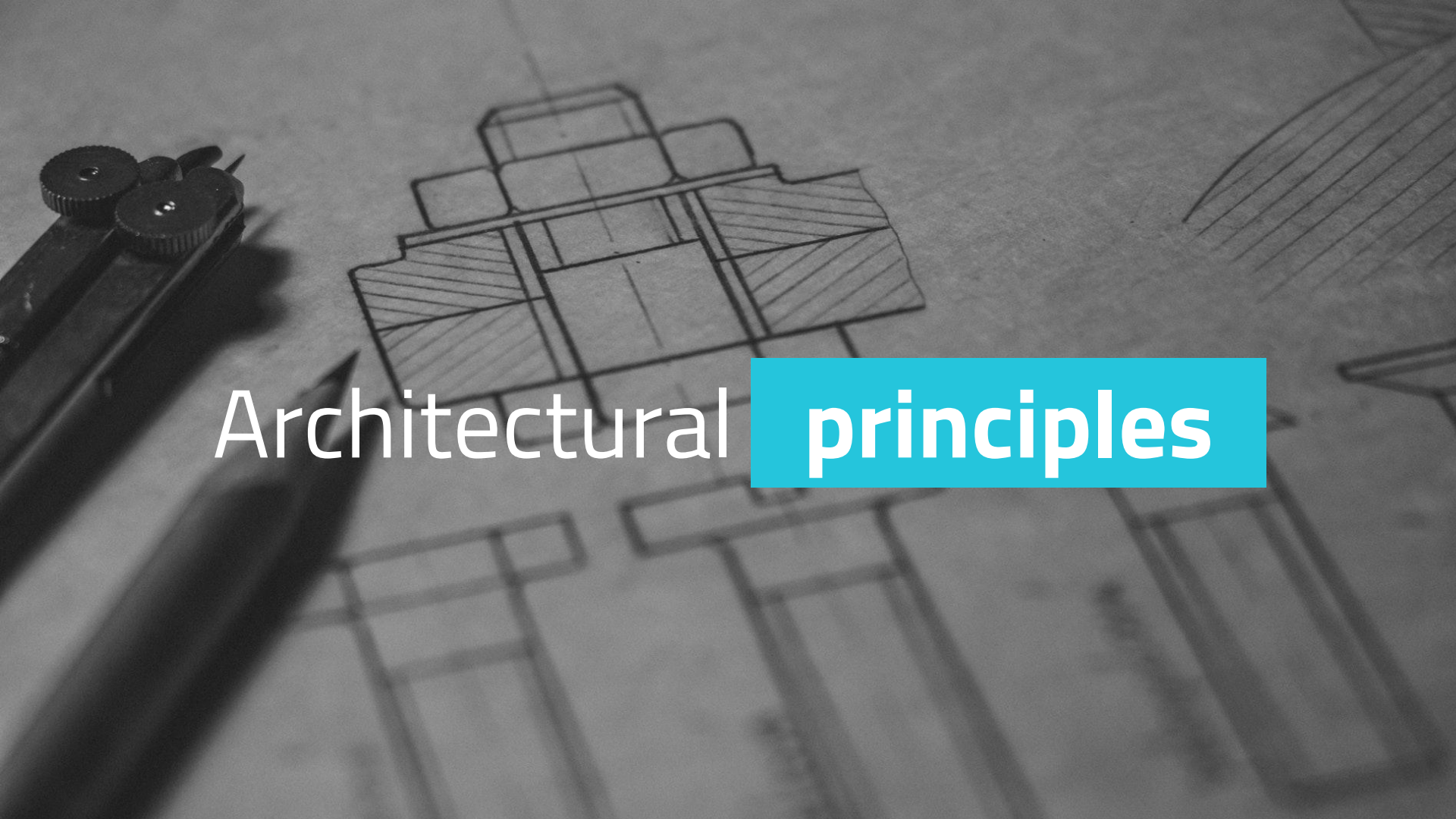
- Vertical boundaries
- Decentralization

Architectural principles

- Collection of 9 basic ‘laws’
- Autonomy, Automation and Comm

Guides

- Practical manual for common tasks (RFC
- e.g. Eventing, REST, Authentication


Architectural principles

autonomy

Autonomy principles

Deploy independently

Ensure that the services can and are deployed by the teams themselves.

Isolate failure

Make the services as resilient as possible.

Hide implementation Details

Different verticals must be stateless and must not share state.

Encapsulate Data Storage

For any data resource exactly one service is responsible. If possible, the data supply should proceed asynchronously in the background

Autonomy principles

The classic enemies of autonomy

Bloated Features

- Smaller features (MVP) -> fewer teams, lesser synchronization overhead

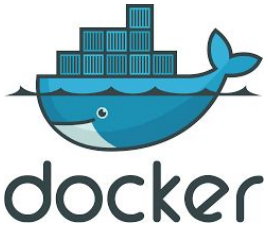
Big Bang Releases

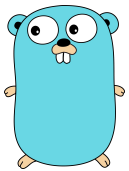
- Appropriate patterns -> smaller risk, lesser need for synchronization

Urge for Reutilization and Shared Libraries

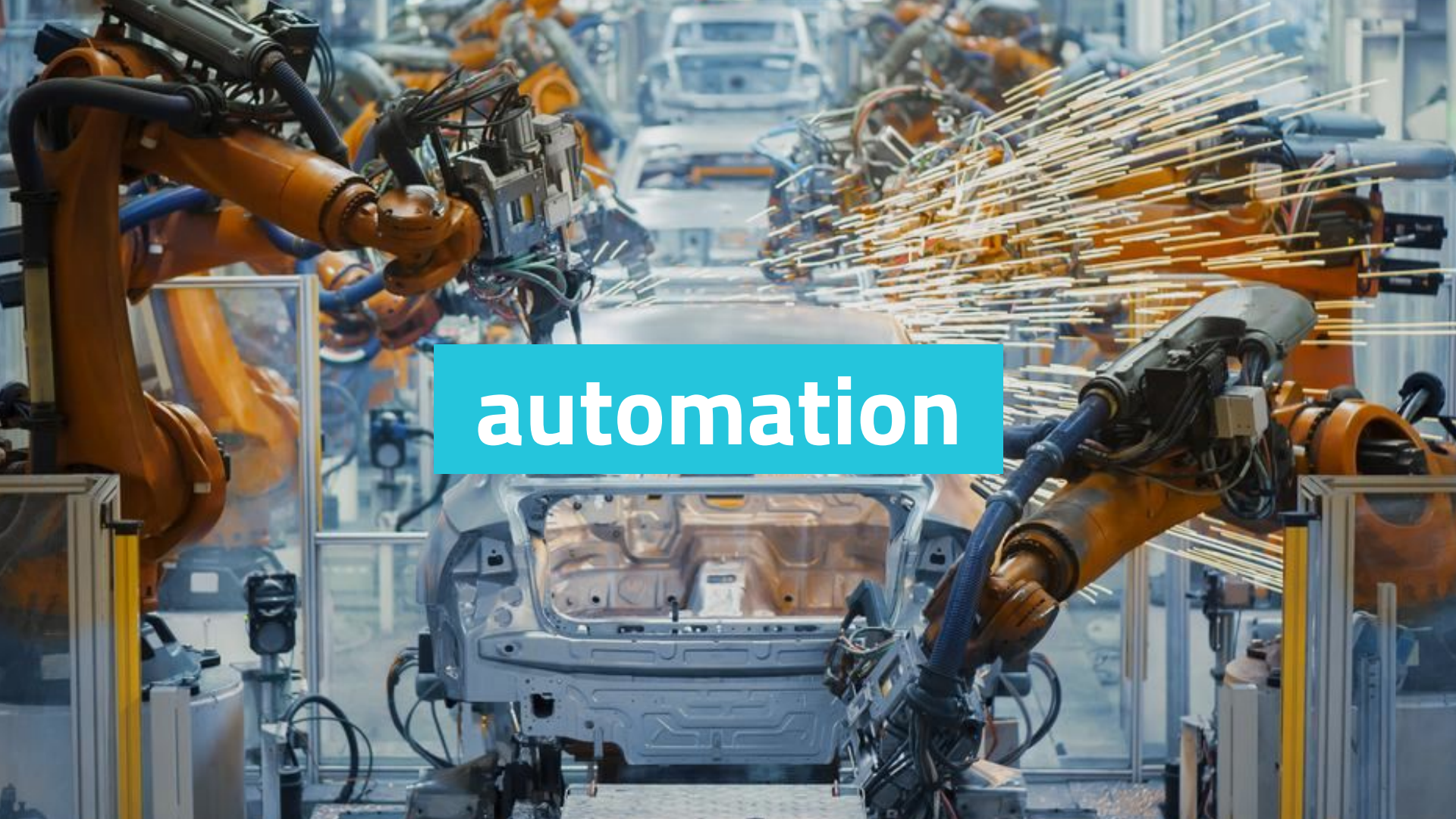
- “WET is the new DRY”

Autonomy opens your tech stacks

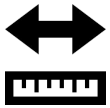




Autonomy opens your tech stacks


automation

Automation principles



Be Scalable

Services are scaled horizontally

Embrace a Culture of Automation

Test, deployment and operations are completely automated

Be Highly Observable

Use of semantic monitoring to see if the whole platform works correctly.

Automation principles

Be Scalable

Services are scaled horizon

Embrace a Culture of A

Test, deployment and ope

Be Highly Observable

Use of semantic monitoring to see if t ly.


communication

Communication principles

Standardize Service communication

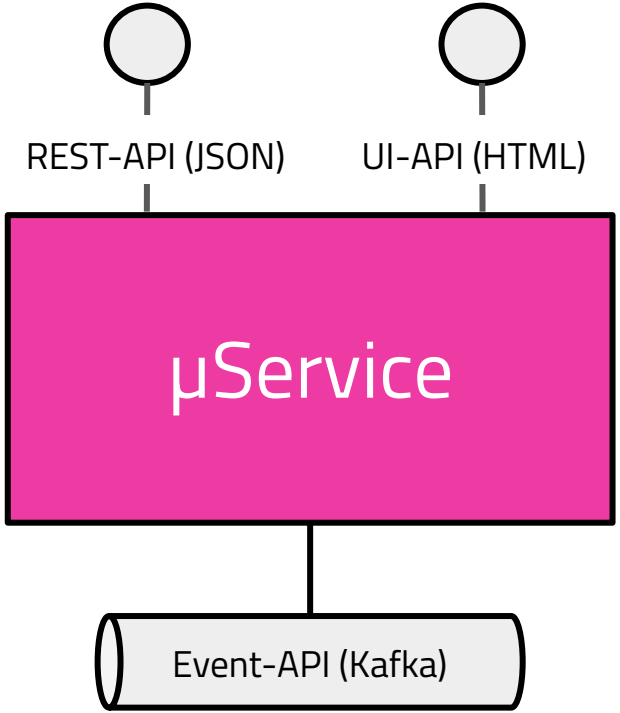
Inter-service communication is standardized and if possible asynchronous

Follow REST Principles

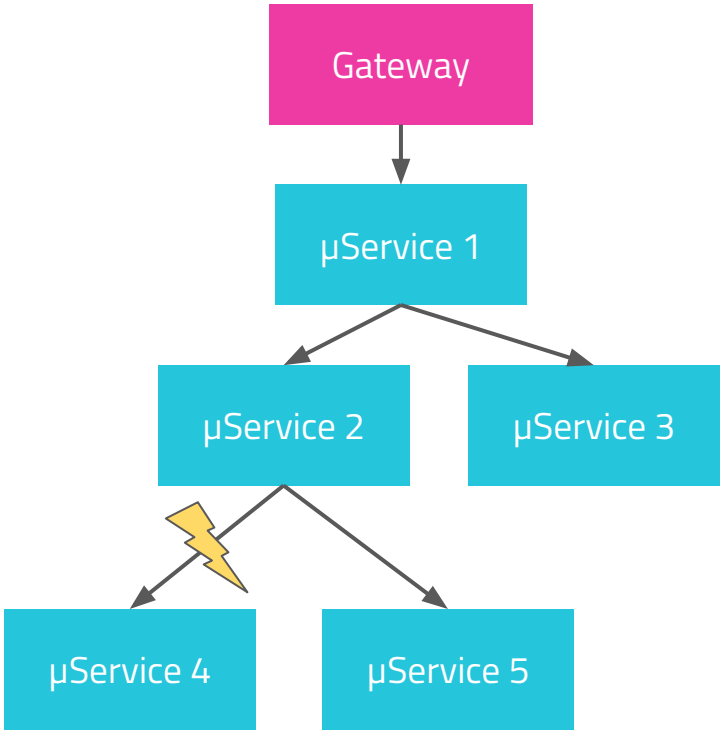
The API of a service follows the RESTful paradigm (REST maturity level 2)

Communication in microservice worlds

Having data is better than fetching data



Problems in HTTP/REST-only architectures



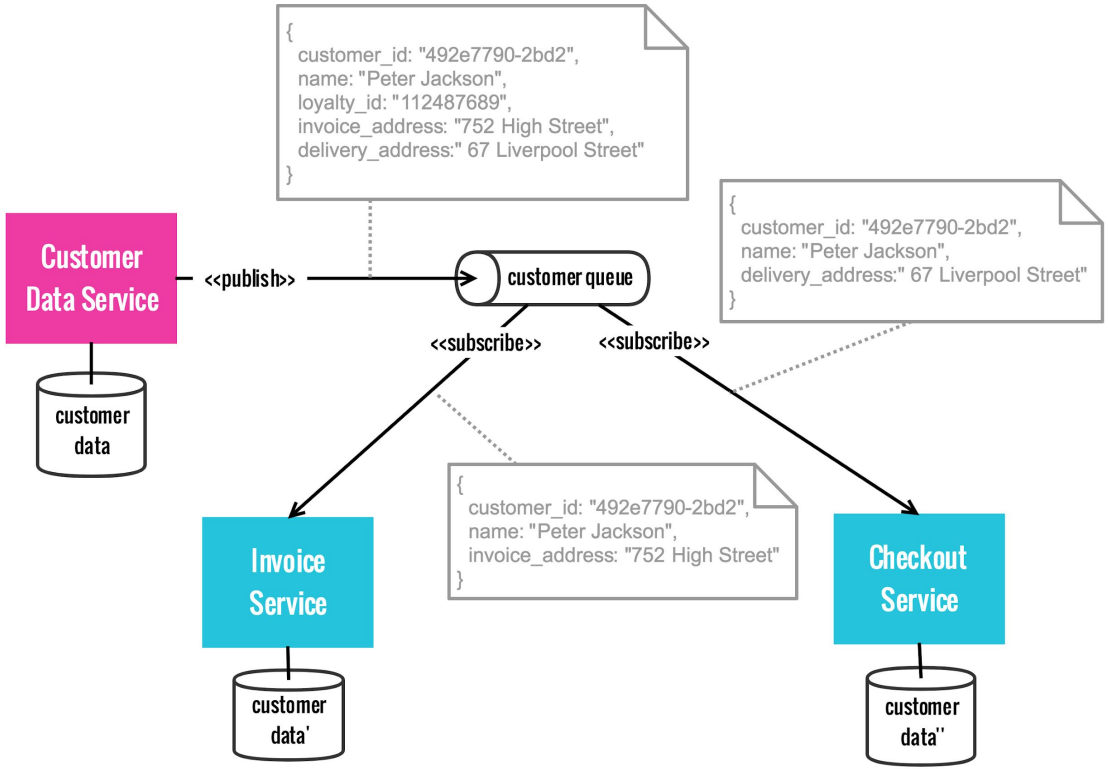
Things that mitigate:

- Timeouts
- Fallbacks
- Circuit Breakers

Things that actually help:

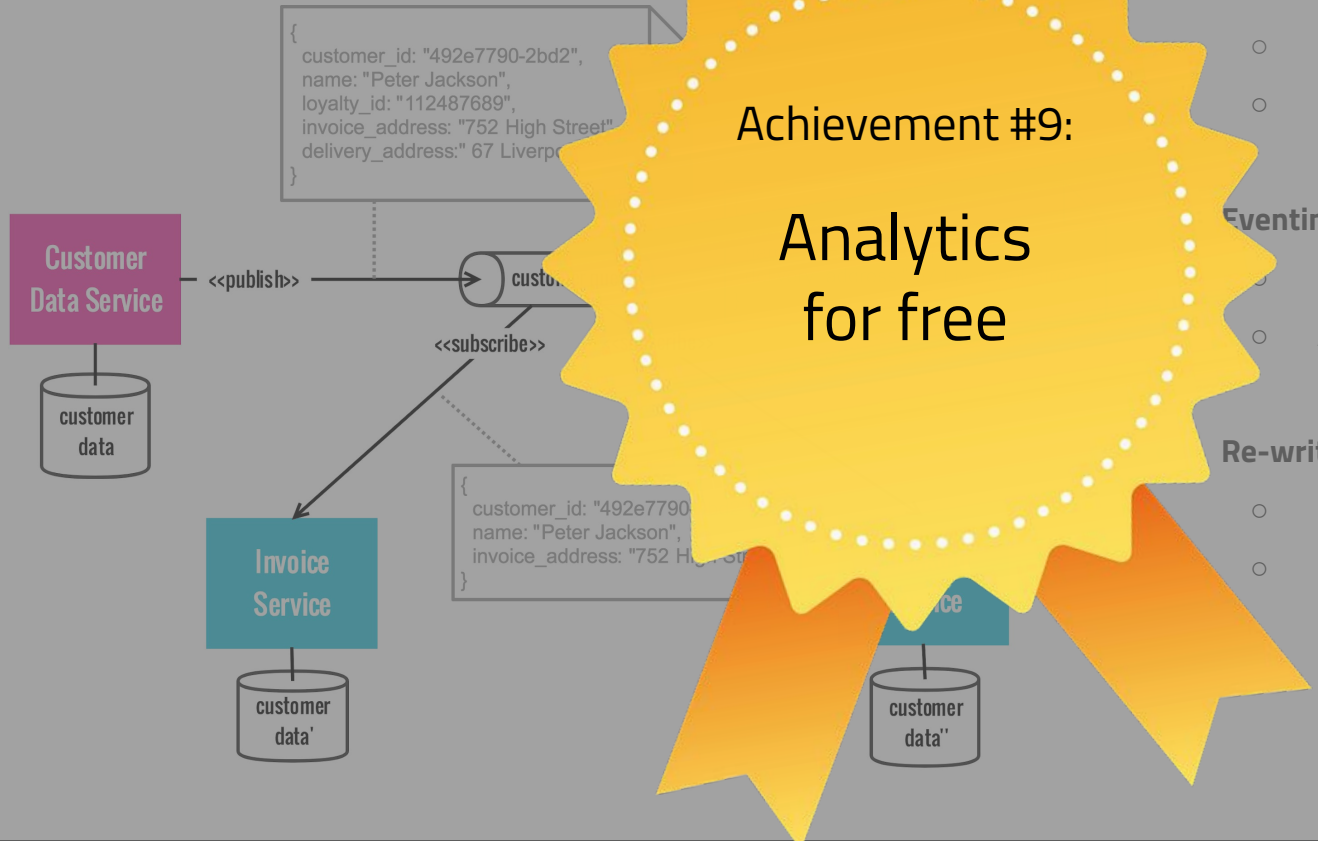
- **Eventing**

Eventing with Apache Kafka



- **Eventing != Messaging**
  - Publish events that already **happened**
  - One owning service per queue/topic
- **Eventing != Event Sourcing**
  - Complete entities - not deltas
  - Allows log compaction
- **Re-writing and Re-reading**
  - needed in case of additional data
  - useful in case of data loss

Eventing with Apache Kafka



Achievement #9:

Analytics for free

- **Eventing != Messaging**
  - Publish events that already **happened**
  - One owning service per queue/topic
- **Eventing != Event Sourcing**
  - Complete entities - not deltas
  - Allows log compaction
- **Re-writing and Re-reading**
  - needed in case of additional data
  - useful in case of data loss

Lessons learned with microservice communication

Eventing is an API as well

- Events have to behave like APIs and avoid breaking changes.

Writing (generic) APIs is hard

- Teams tend to write APIs for special clients (e.g. mobile apps)

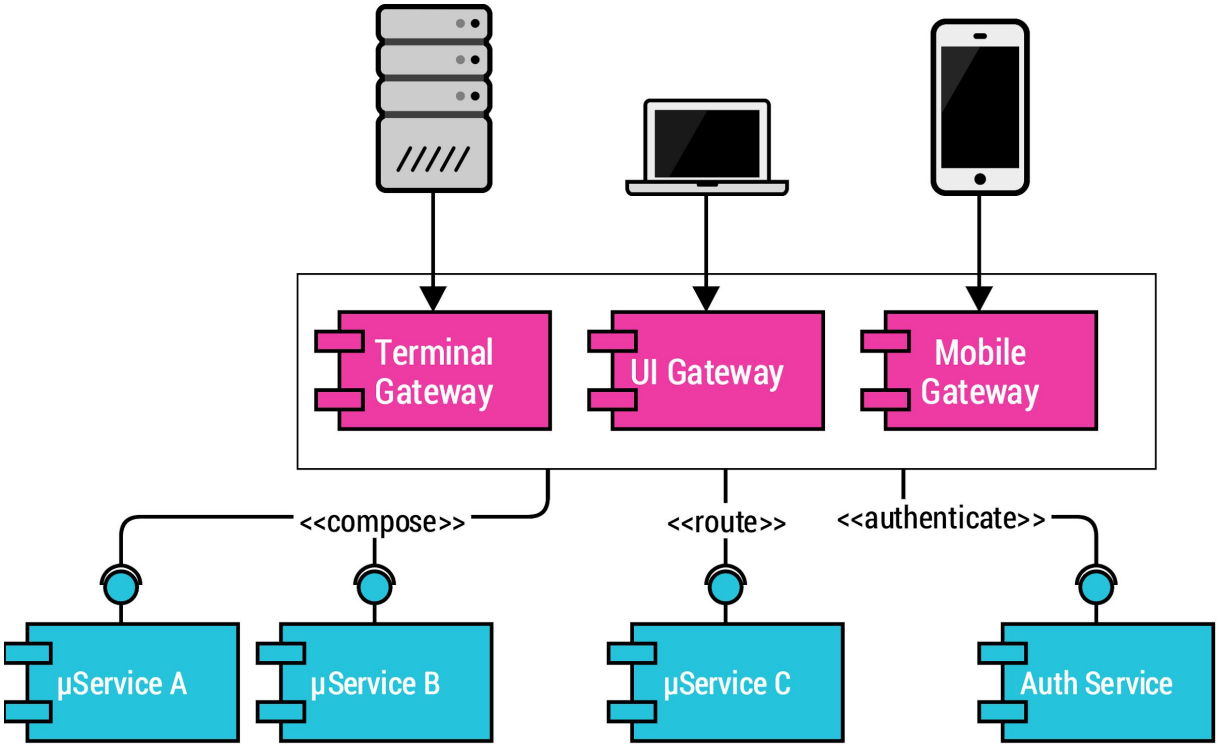
Breaking changes require a strict procedure

- Must be solved by introducing new endpoints / new topics


How to access?

Usage of API gateways as entry points

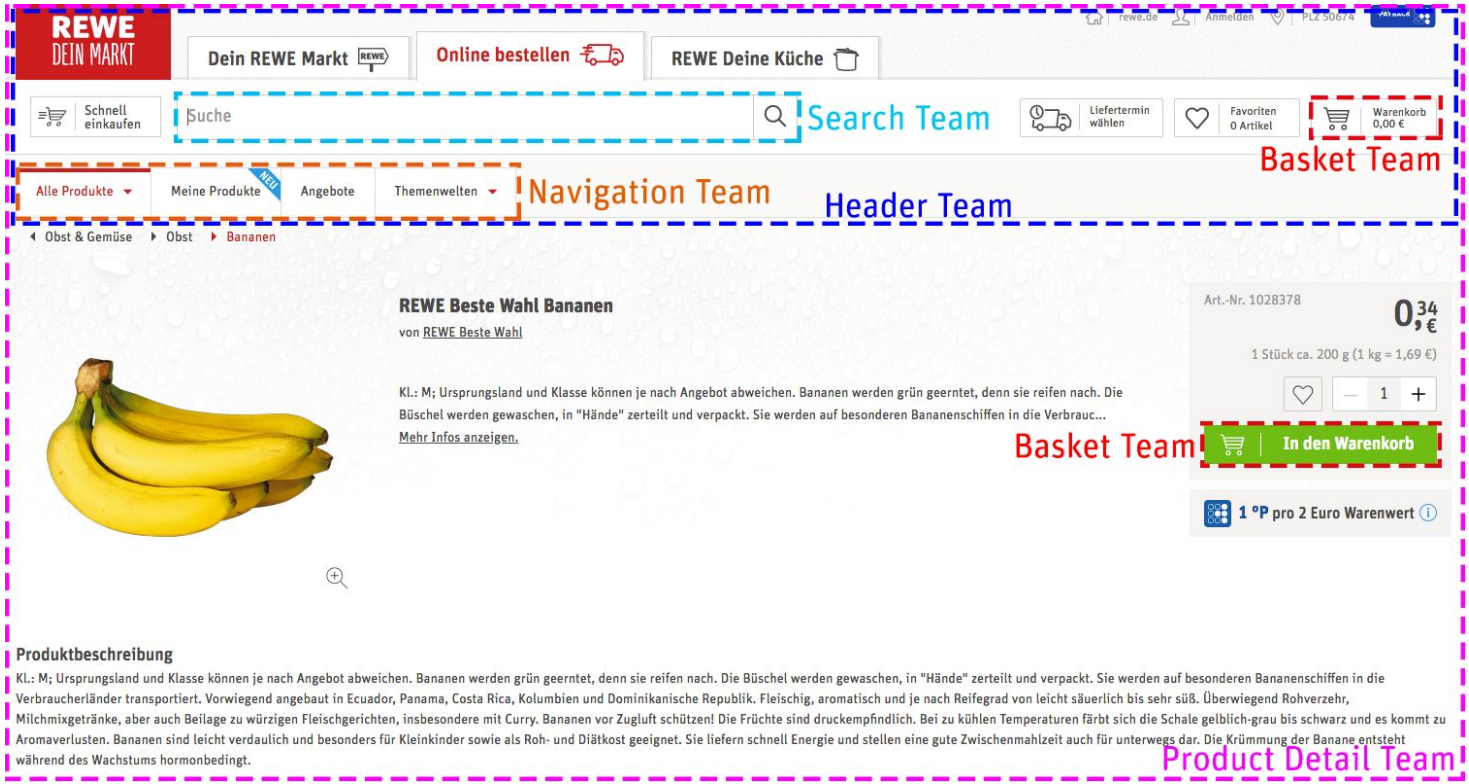
How to access Microservices?



What about **frontend?**

Dynamic frontend composition

How UIs in microservice architectures should be composed



Dynamic frontend composition

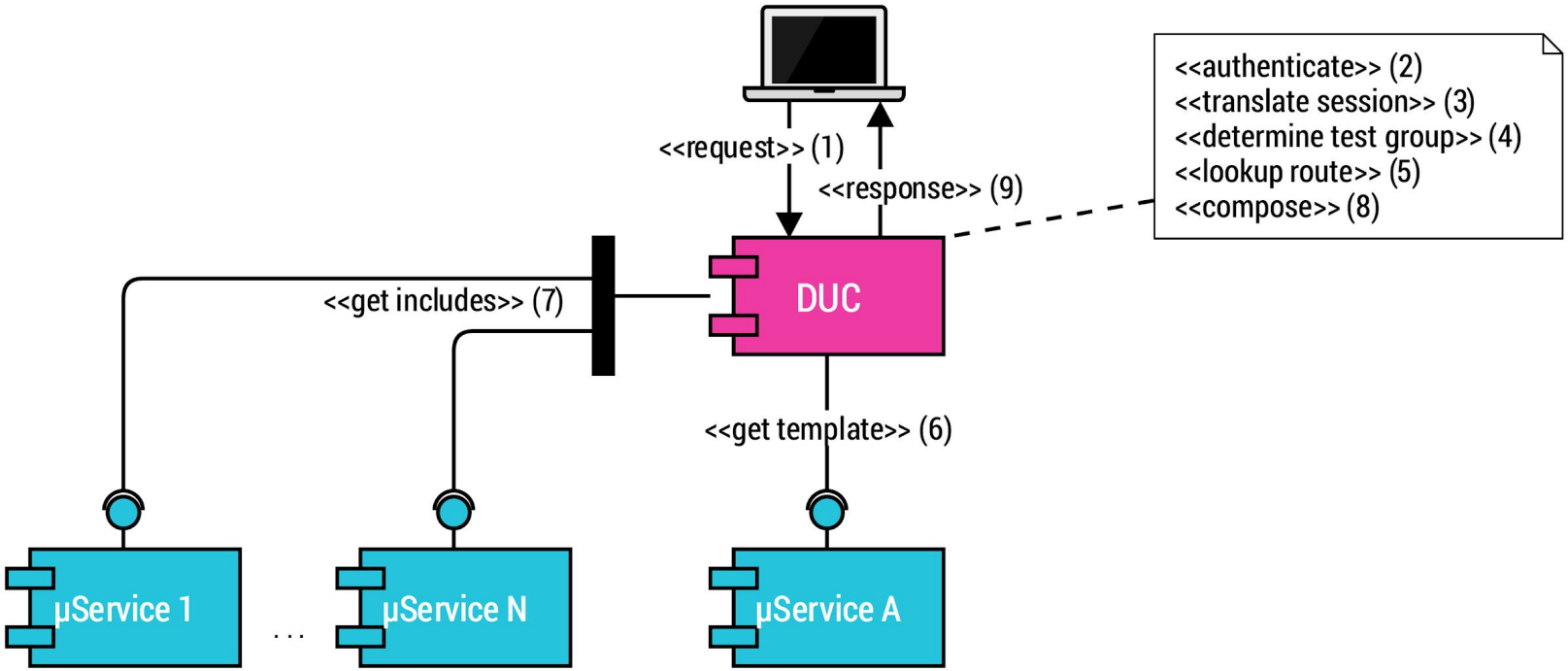
How UIs in microservice architectures should be

Achievement #10:

Verticle boundaries all the way

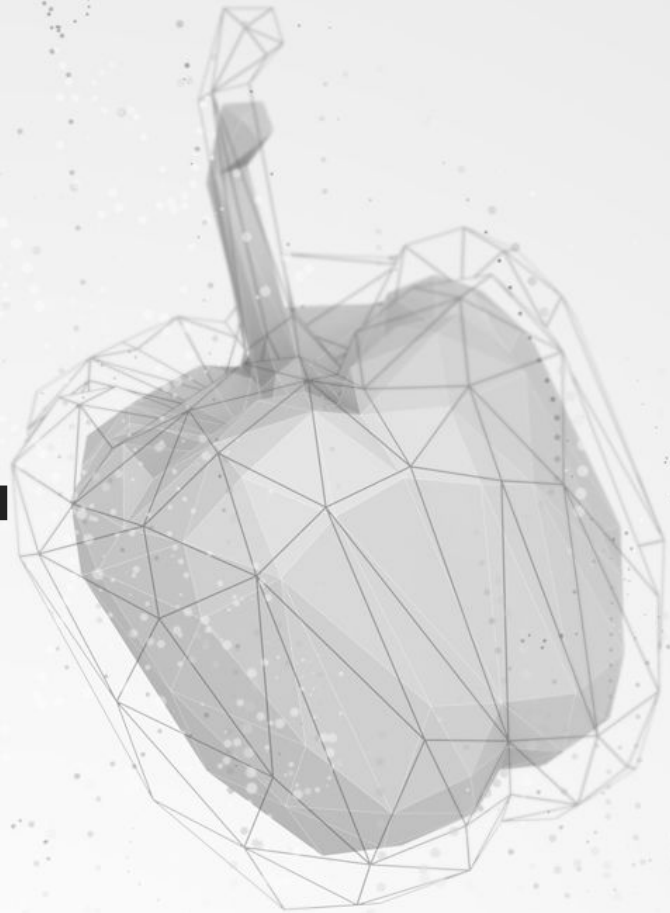
Dynamic UI composition with DUC

Where does my UI come from?



Achievements - Summary

1. Fast growth
2. Full-Stack-Engineers
3. Flexible architecture
4. Motivation by freedom
5. Fast movement
6. Clear data ownership
7. Flexible choice of technology
8. Flexible scaling
9. Analytics for free
10. Vertical boundaries all the ways



Thank you

Questions ?

REWE digital

DEVelopers Köln Meetup

5 years of food retail e-commerce

A microservice success story

Sebastian Gauder
Software Architect
@rattakresch

REWE digital