A Microservices Journey

Susanne Kaiser

@suksr

EX-CTO at Just Software

@JustSocialApps

Each Journey is Different

"People try to copy Netflix,
but they can only copy what they see.
They copy the results, not the process."

Adrian Cockcroft, AWS VP Cloud Architect,
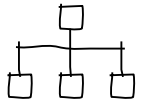former Netflix Chief Cloud Architect
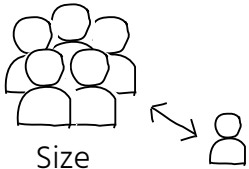
Each Journey is Different
Affecting Circumstances
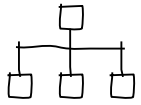
Team

Structure

Skillset

Size

Journey
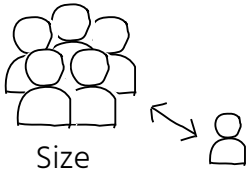
Each Journey is Different
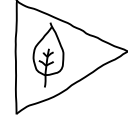## Affecting Circumstances

Team

Structure

Skillset

Size

Journey

Legacy

Maintenance effort

Runtime environment

Each Journey is Different
Affecting Circumstances

Team
- Structure
- Skillset
- Size

Journey

Legacy
- Maintenance effort
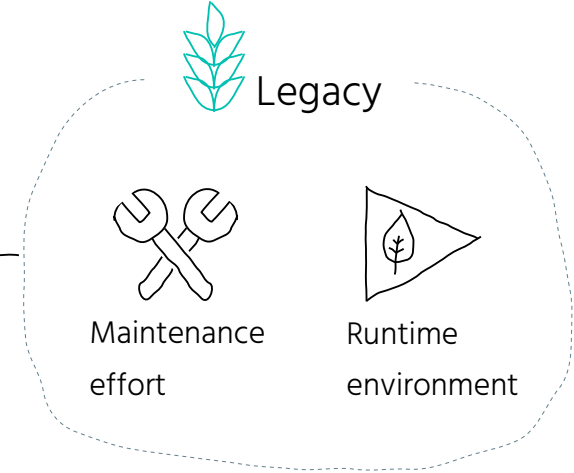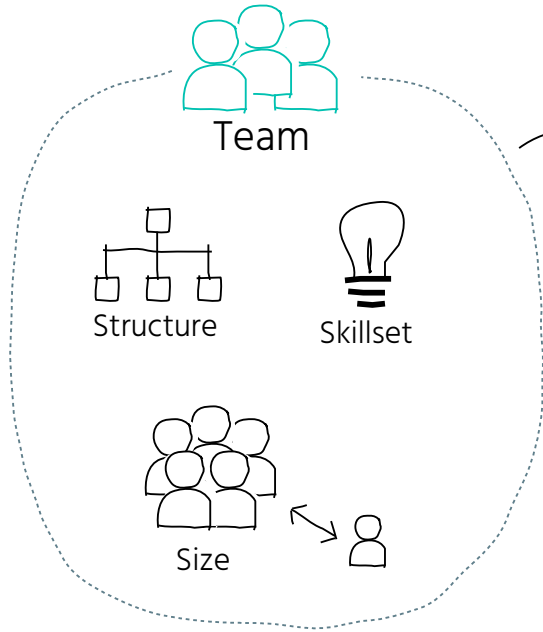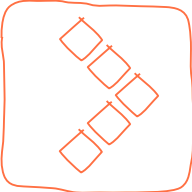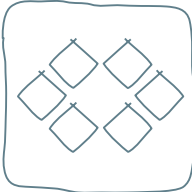- Runtime environment
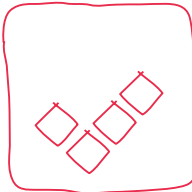
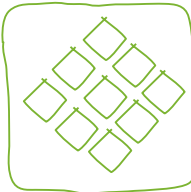**Strategy**
- New Features
- Timeline / Milestones
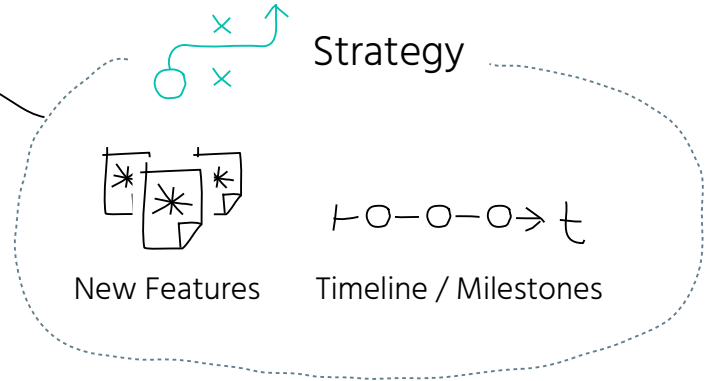
Background

JUST SOCIAL
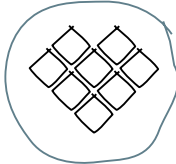
JUST DRIVE JUST CONNECT JUST LIST JUST WIKI

JUST PEOPLE JUST NEWS

At the Beginning

A Monolith in Every Aspect

One team

Single Unit

One technology stack
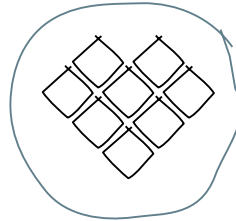
One collaboration product

Background
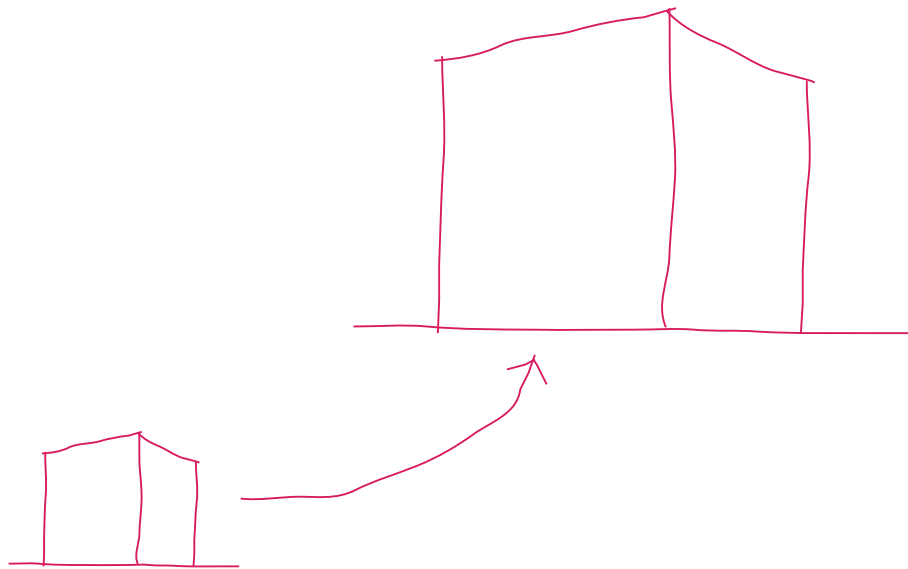
After an Evolving Time

Background
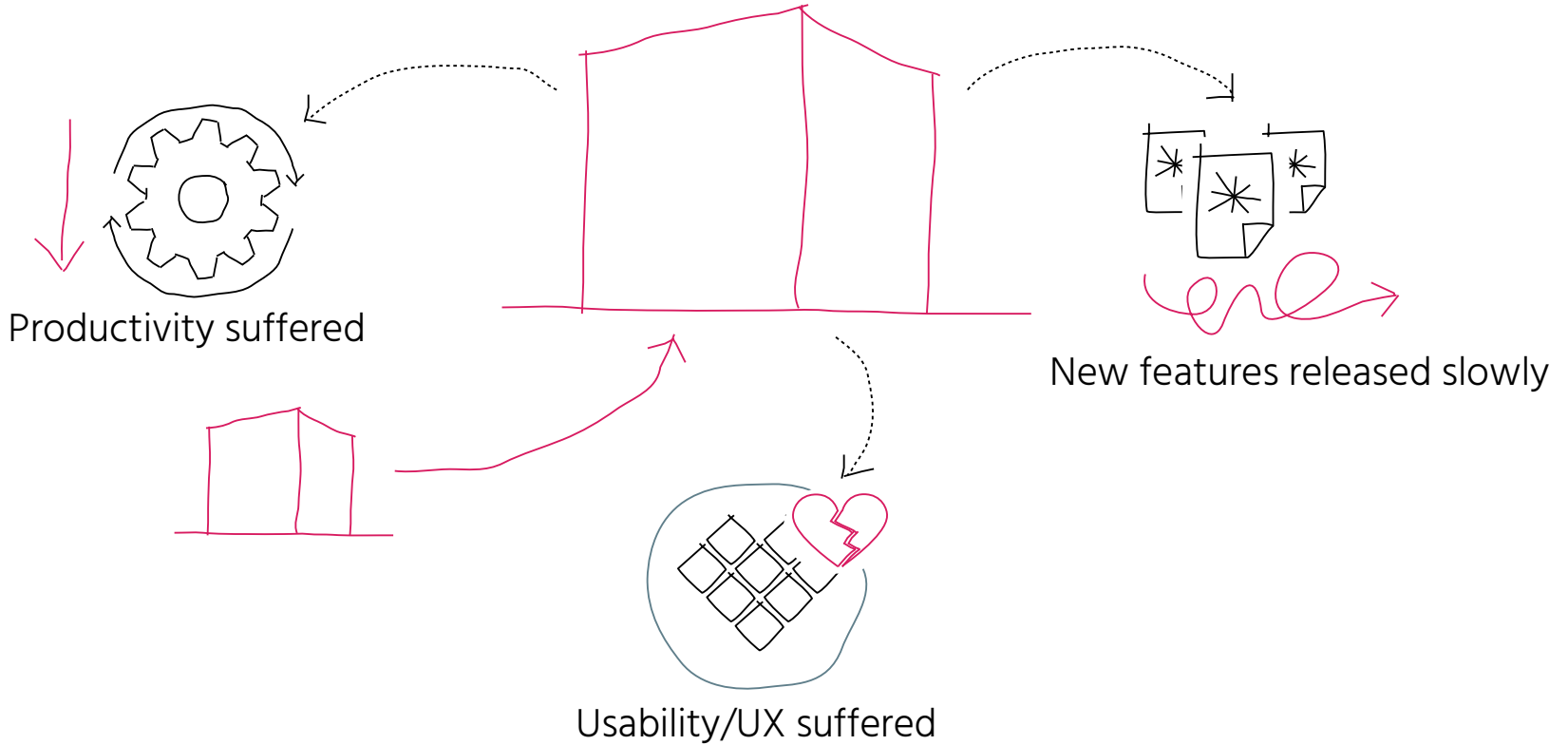
After an Evolving Time

Productivity suffered

Background

After an Evolving Time

Productivity suffered

Usability/UX suffered

Background

Productivity suffered

Usability/UX suffered
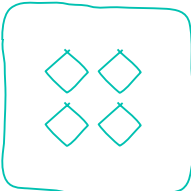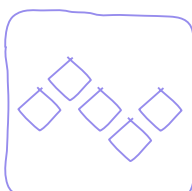
New features released slowly
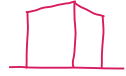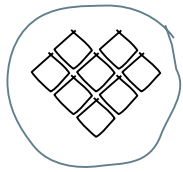
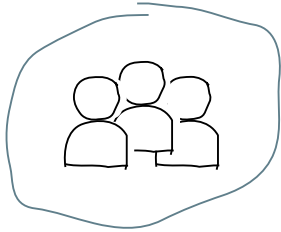# Background

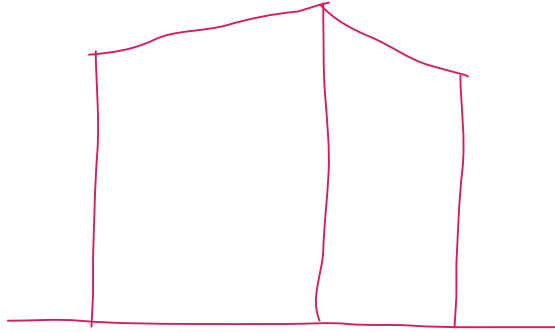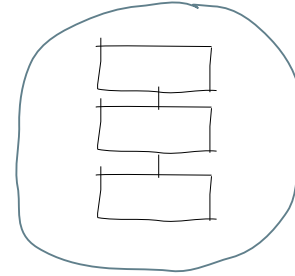JUST SOCIAL
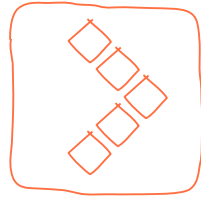
JUST DRIVE
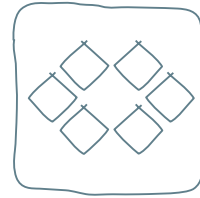
JUST CONNECT

JUST LIST

JUST WIKI

JUST PEOPLE
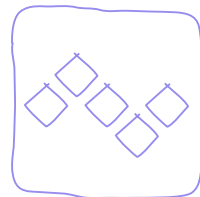
JUST NEWS

Background

Separate, Autonomous Teams

Well-defined responsibilites
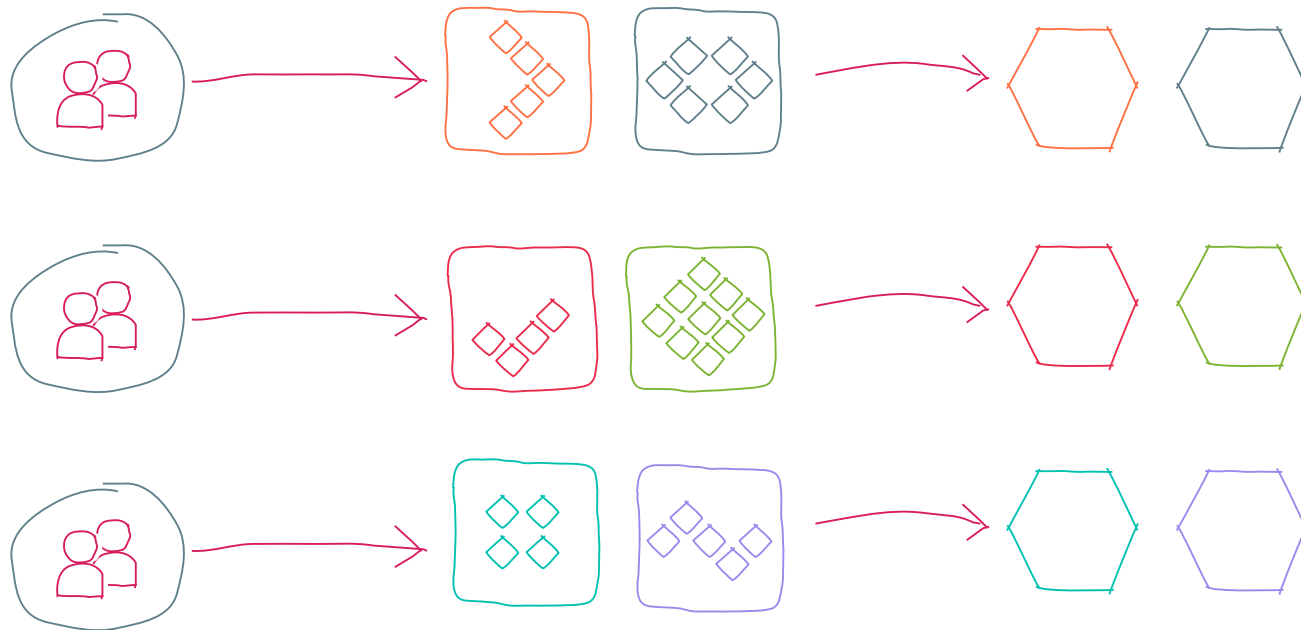
Organisation   Product   Software

Background
Our Motivation for Microservices
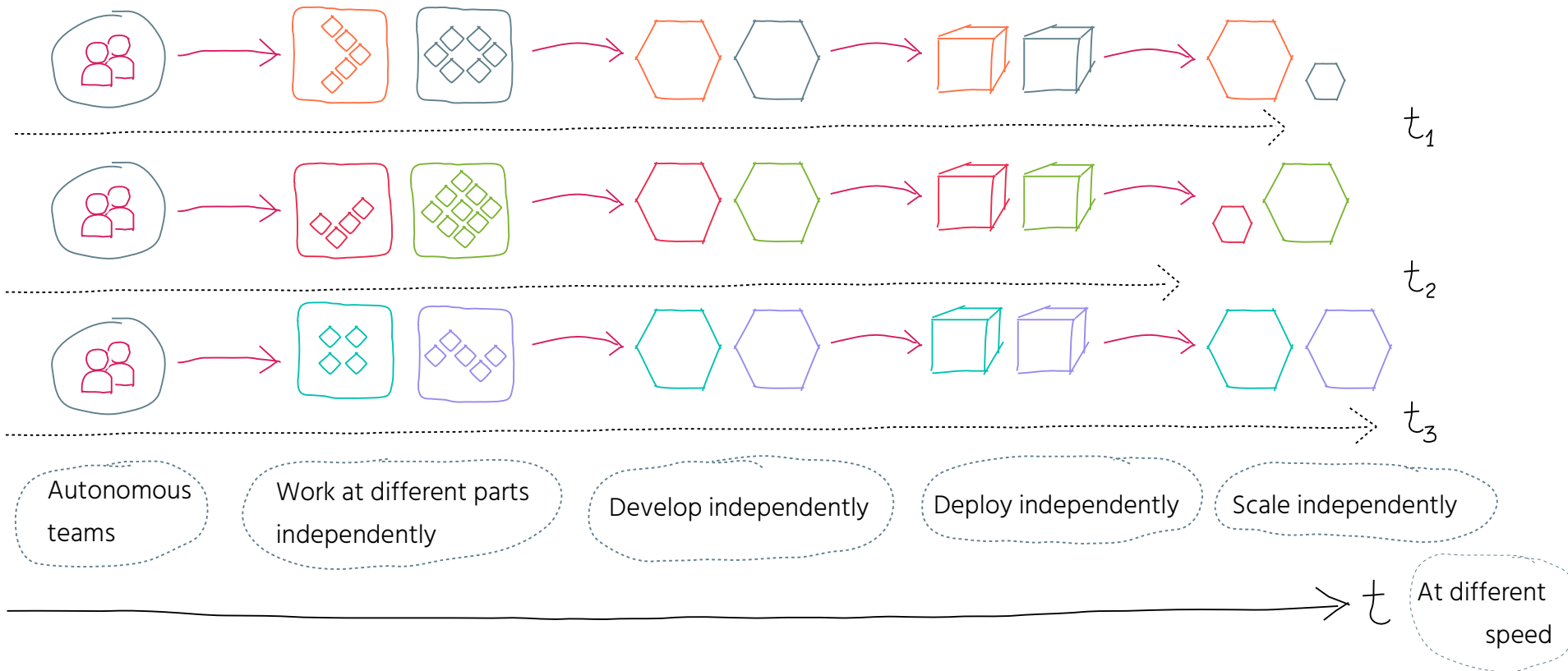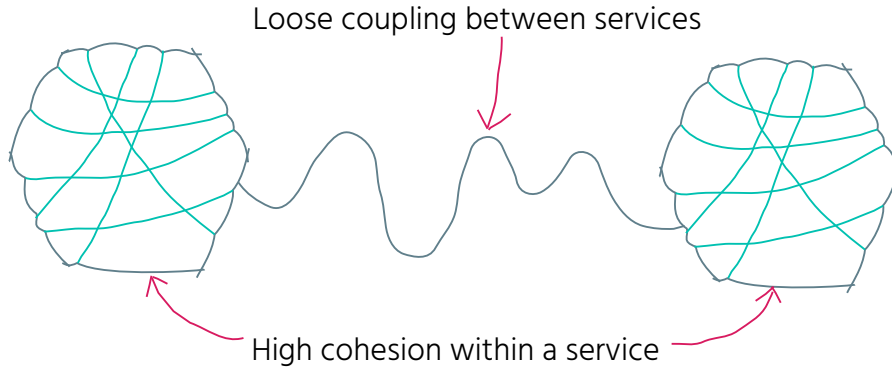
t_1

t_2

t_3

Autonomous teams

Work at different parts independently

Develop independently

Deploy independently

Scale independently

t

At different speed

Decomposition Strategy

Identify Bounded Contexts

Loose coupling between services

High cohesion within a service

Decomposition Strategy

Identify Bounded Contexts

Loose coupling between services

High cohesion within a service

Related behaviour

Bounded Context

Well-defined business function

Semantic boundary around domain model
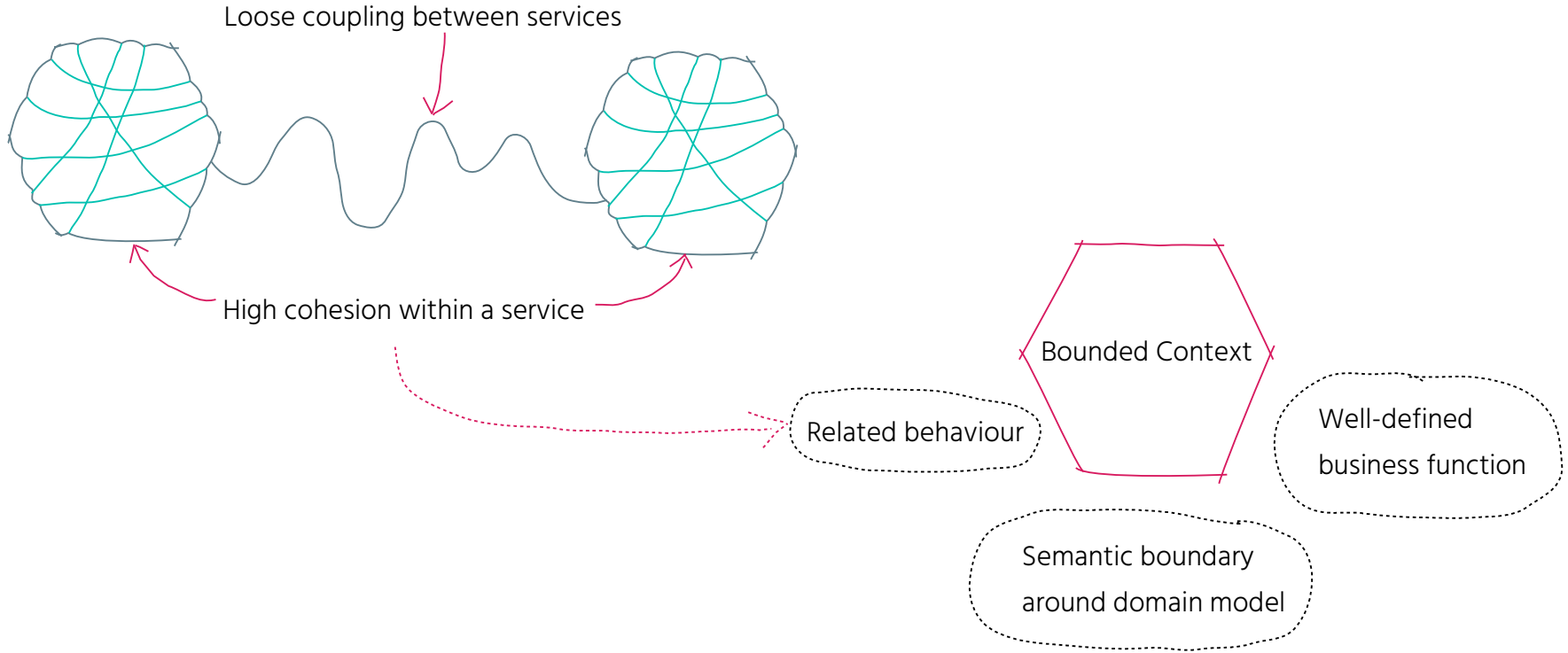
Decomposition Strategy

⬡ Identify Bounded Contexts

Bounded Contexts
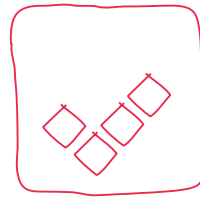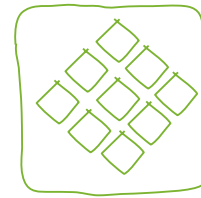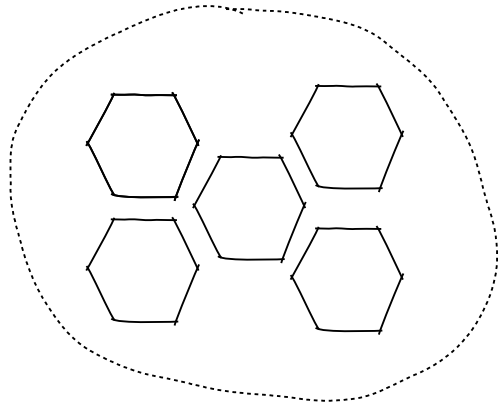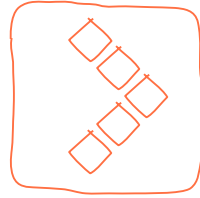

JUST DRIVE
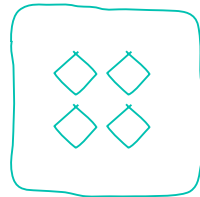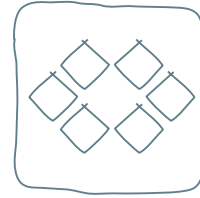
JUST CONNECT
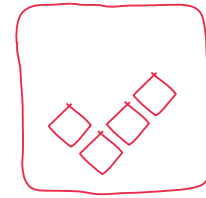
JUST LIST
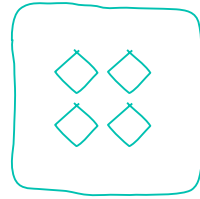
JUST PEOPLE

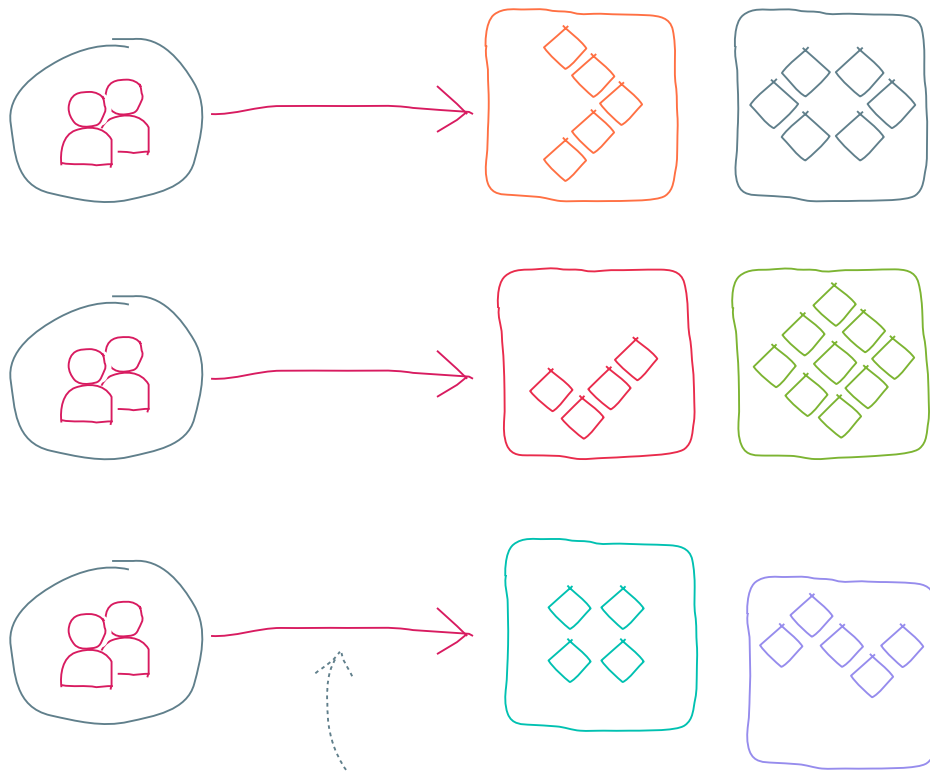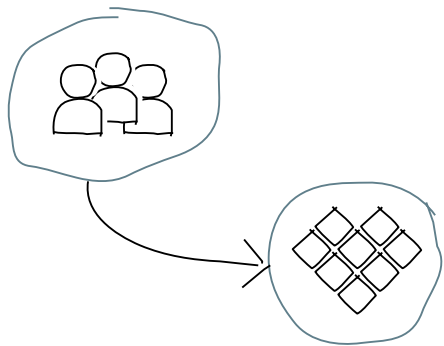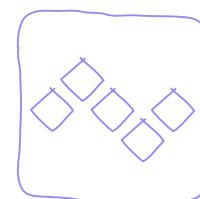JUST NEWS
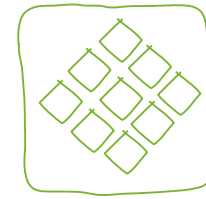
JUST WIKI

Decomposition Strategy
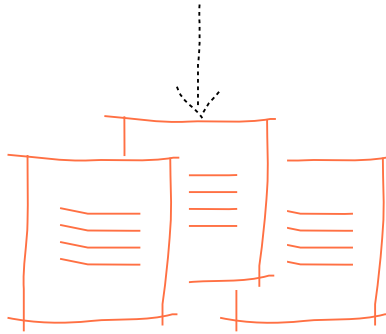
Co-Existing Service From Scratch

JUST DRIVE

Decomposition Strategy

Co-Existing Service From Scratch

JUST DRIVE

JUST PEOPLE

Decomposition Strategy

Co-Existing Service From Scratch

JUST DRIVE

REST API

Application-Service

Domain-Model

DB Adapter

Monolith

owns document state

Decomposition Strategy

Co-Existing Service From Scratch

Monolith

owns profile state

REST API

Application-Service

Domain-Model

DB Adapter

document created by author

owns document state

Decomposition Strategy

Co-Existing Service From Scratch

REST API

Message Broker Adapter

Application-Service

Domain-Model

Monolith

owns profile state

publish

Message Broker

Events

subscribe

DB Adapter

owns document state

local copy of author
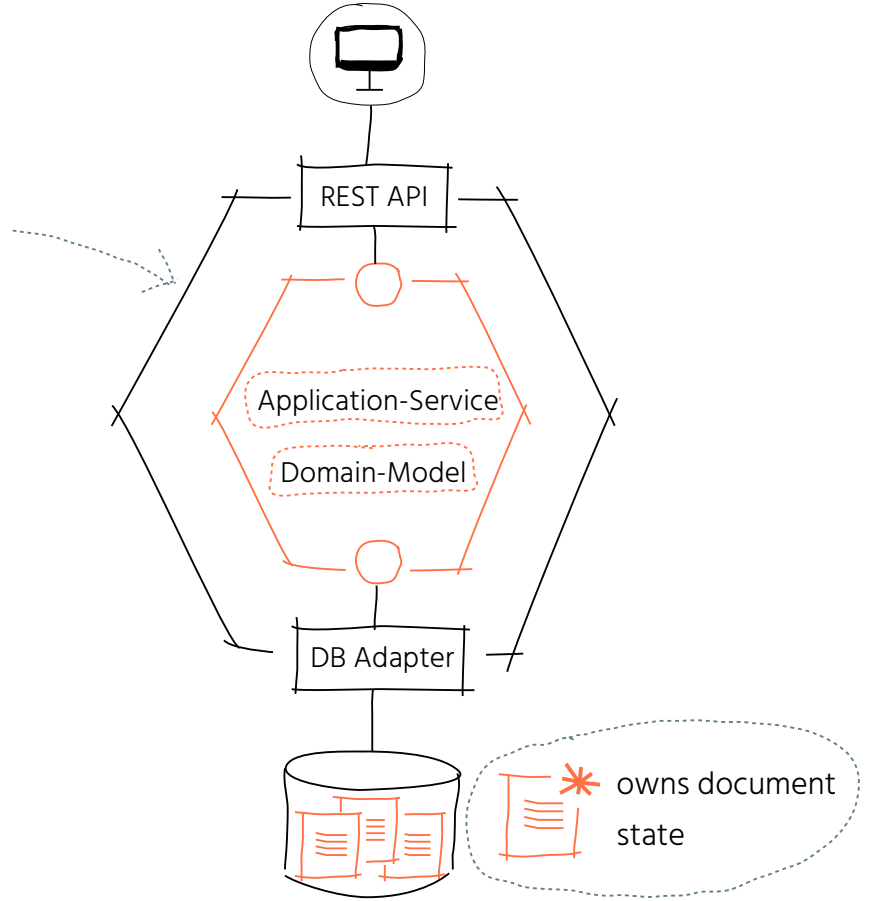
Decomposition Strategy

Co-Existing Service From Scratch

Good approach in general,

but we did too many steps at once

=> Not optimal to start with

vs.

New UI

New Business Logic

REST API

Application-Service

Domain-Model

Domain-Event

Message Broker Adapter

DB Adapter

New Data Structure

Decomposition Strategy

Incremental Top Down

Monolith

REST API

REST Client

REST API
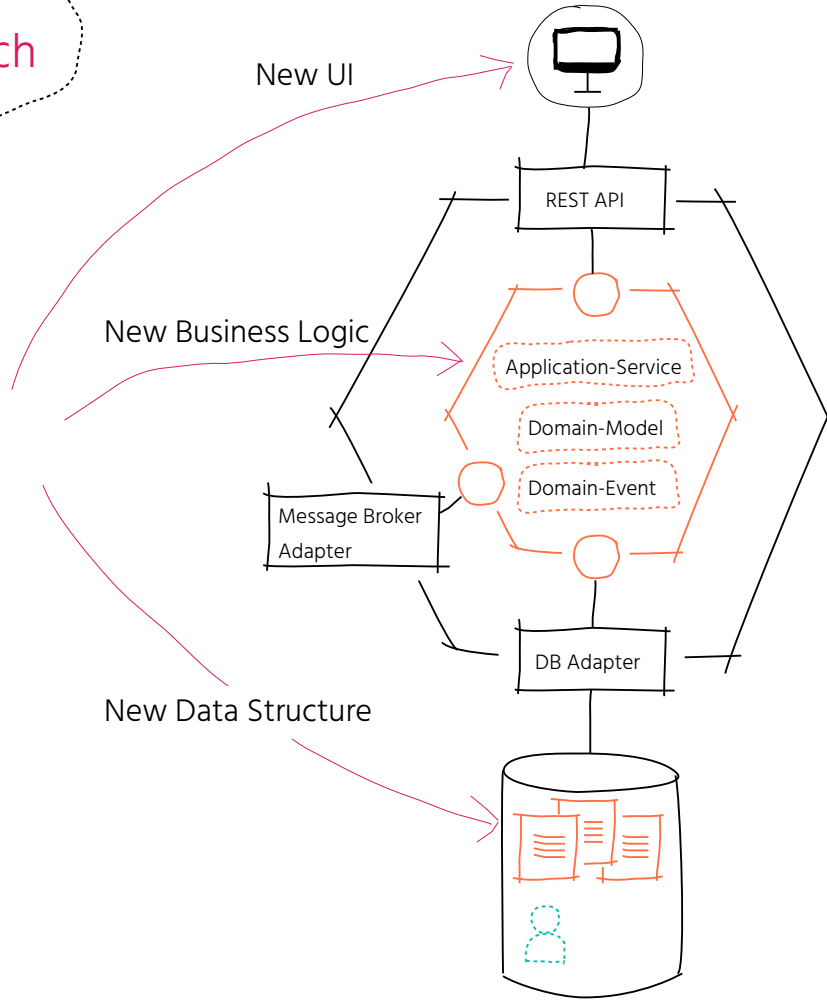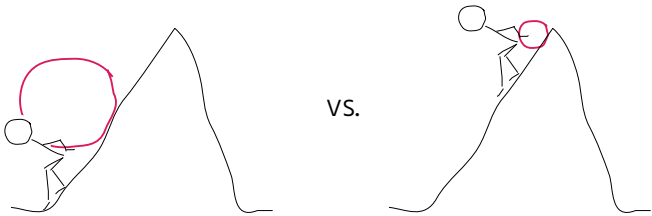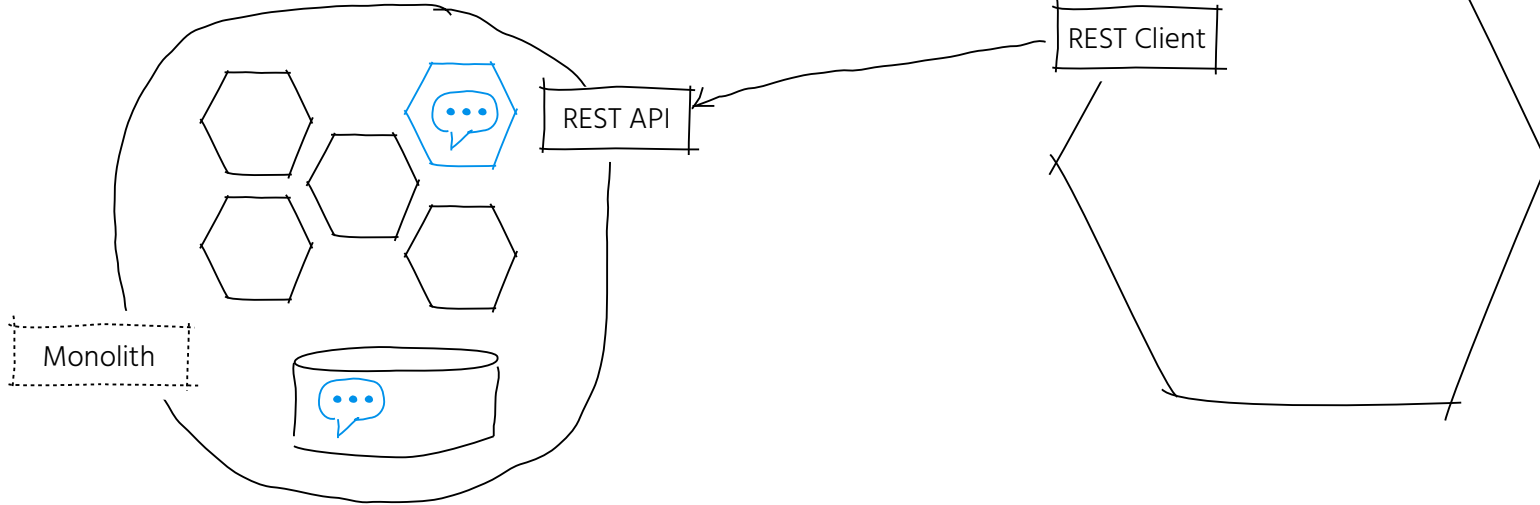
1. Extracting Web App

Decomposition Strategy

Incremental Top Down

Monolith uses
extracted business logic

REST Client

Monolith

REST API

Application-Service

Domain-Model

DB Adapter
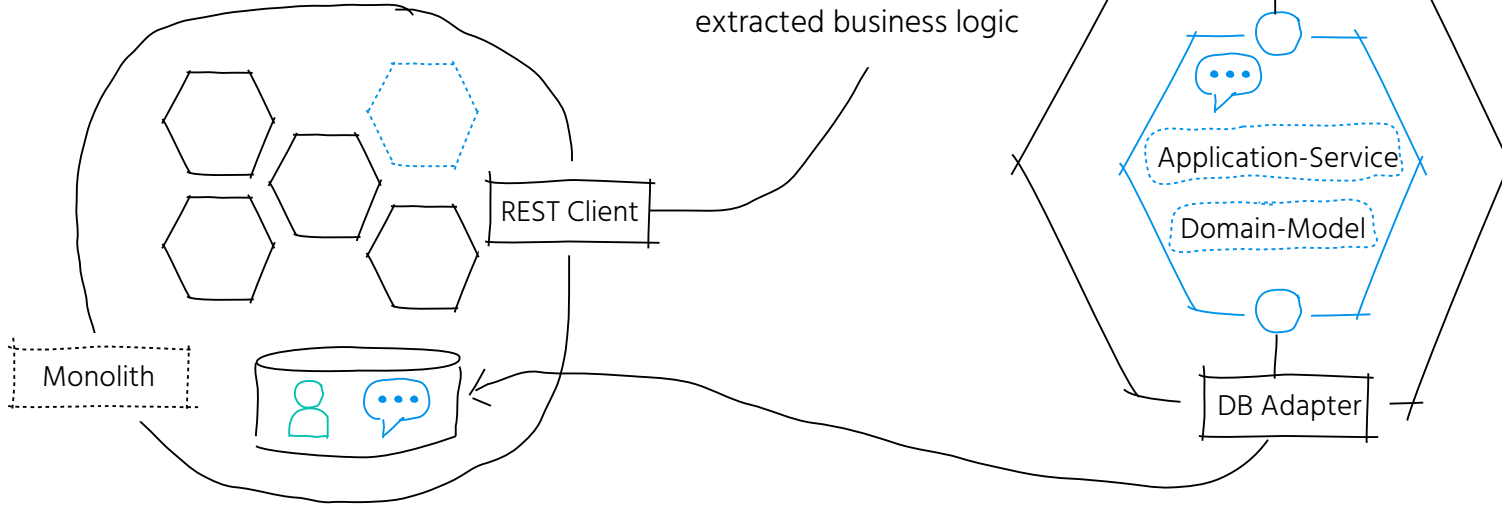
2. Extracting Business Logic

Decomposition Strategy

Incremental Top Down

Monolith

REST API

Application-Service

Domain-Model

Message Broker
Adapter

publish

subscribe

Message Broker

Events

DB Adapter
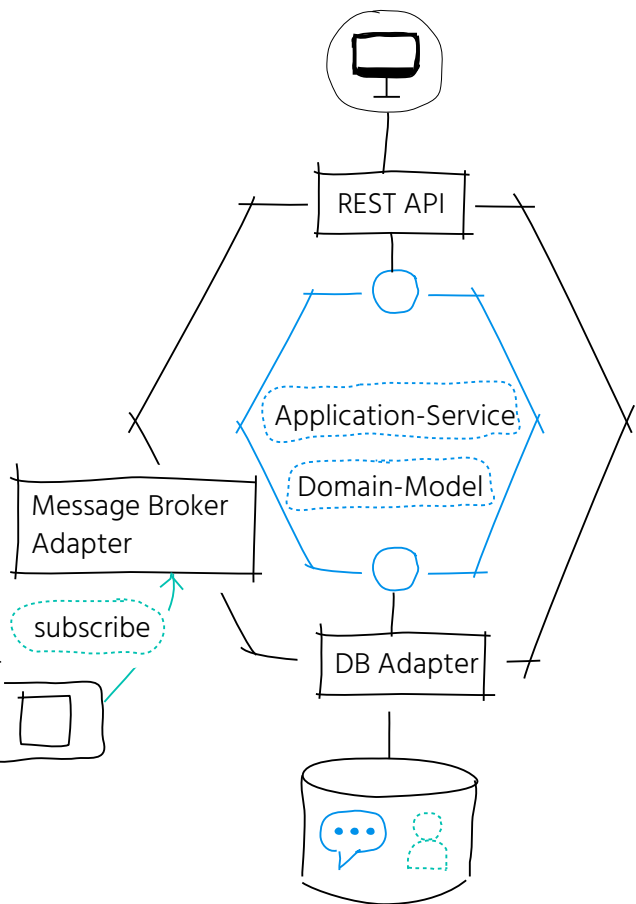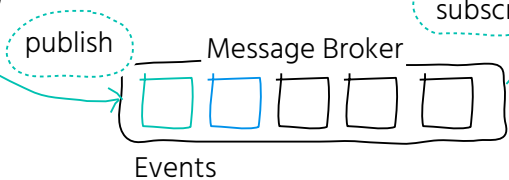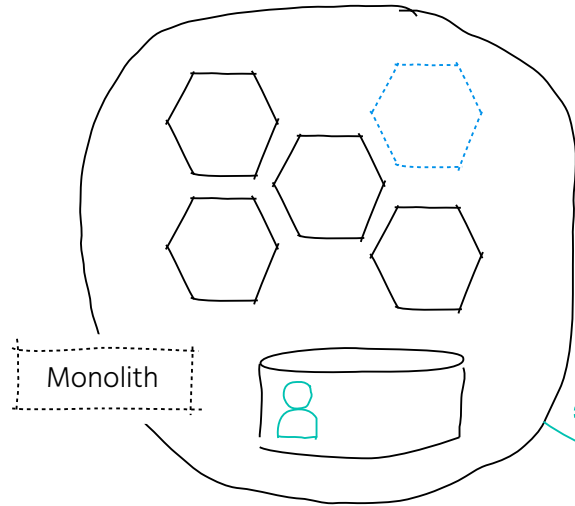
3. Splitting Data Storage

Decomposition Strategy

Which One First?

vs.

Easy to Extract

Early experiences w/ Microservices

Changing Frequently

Greatest benefit after extraction
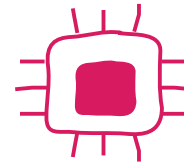
Different Resource Consumption

Cross-Cutting Concerns
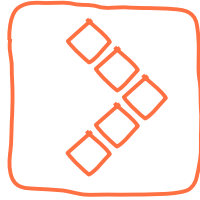
Authorization

JUST DRIVE
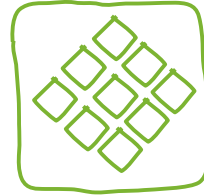
JUST WIKI
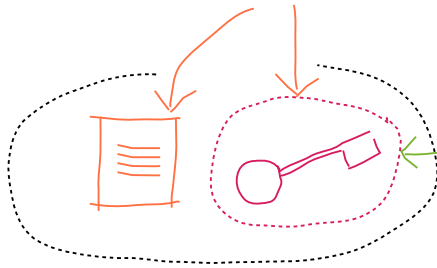
Fine-grained authorization

Inter-service dependency

Cross-Cutting Concerns
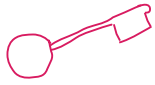
🔑 Authorization

I have a new service that needs authorization. Where is the authz service I could use?

Not there, yet. Sorry!

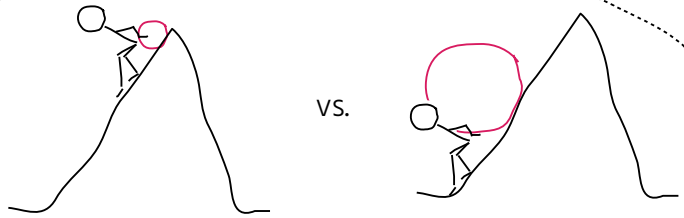Ok, then I am putting my code to the place where authz handling exists … to the monolith.

Ok, then I am implementing authz in my local service.

Feeding the monolith

Re-implementing authz w/ every new service

Cross-Cutting Concerns

Handle Them Early

⚡ Feeding the monolith

⚡ Re-implementing authz w/ every new service

↩ Handle Cross-Cutting Concerns Early

Cross-Cutting Concerns

Avoid A Distributed Monolith

Authz Service

Cross-Cutting Concerns

Avoid A Distributed Monolith

translate

translate

translate
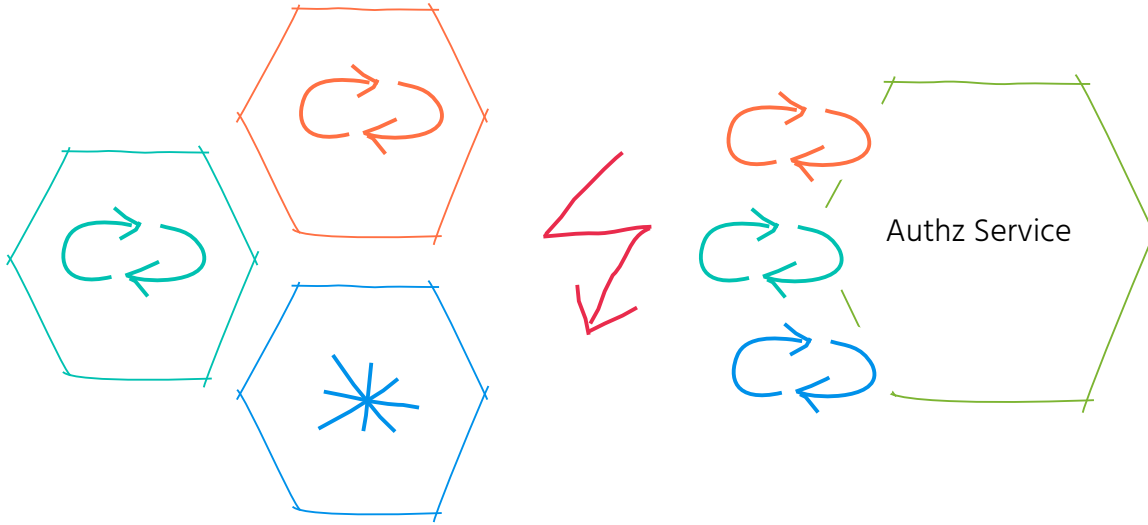
One stable
common contract

Authz Service
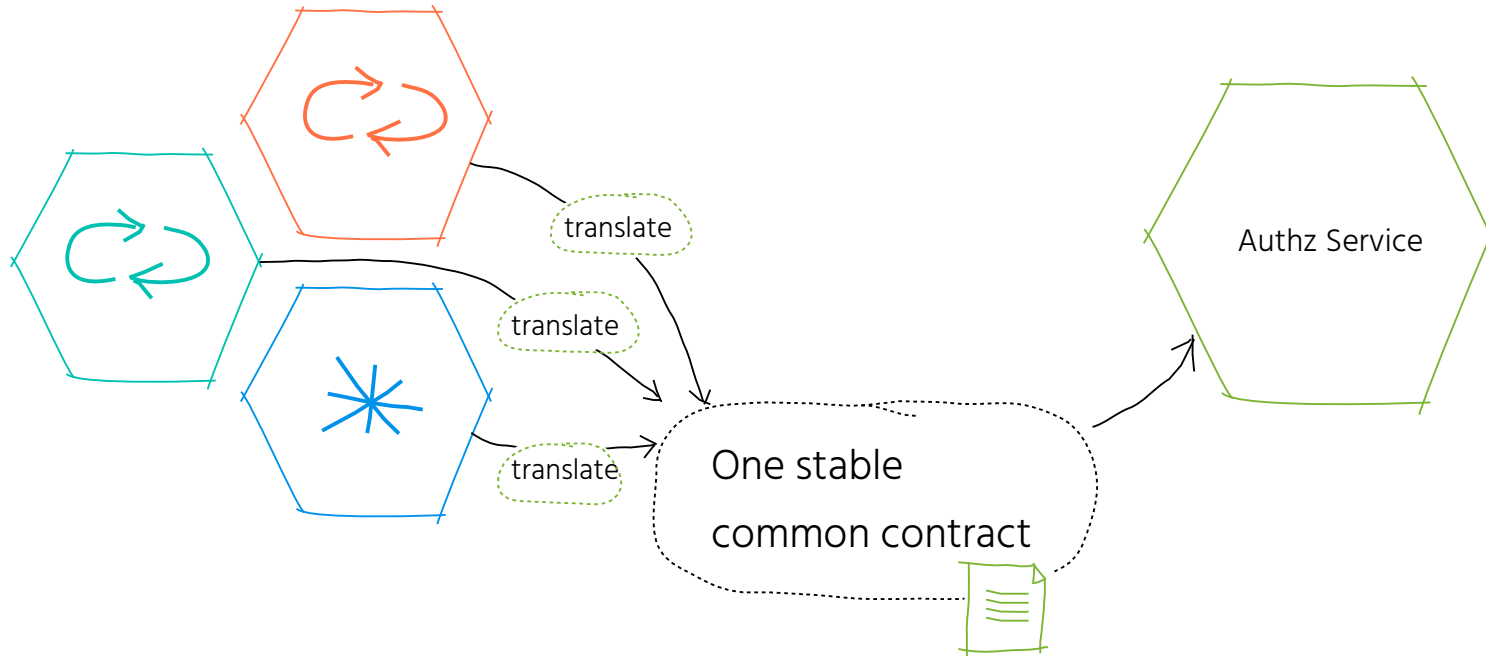
Service Interaction
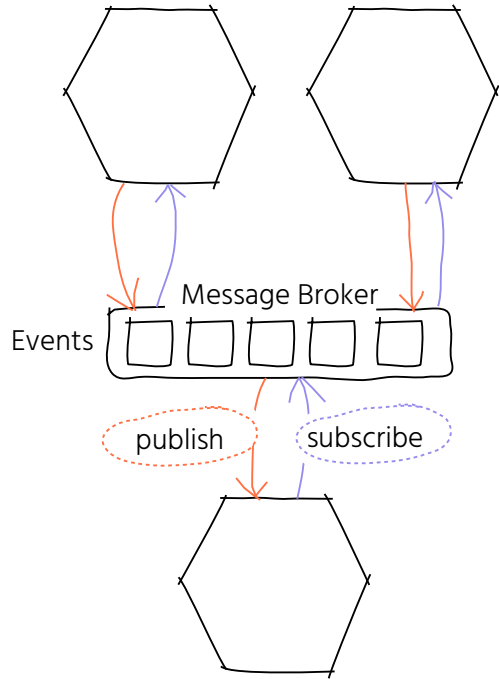
Event-Driven / Request-Driven

Event-Driven

Message Broker

Events

publish   subscribe

Request-Driven

command

query

Hybrid

command

query

Message Broker

Events

publish   subscribe
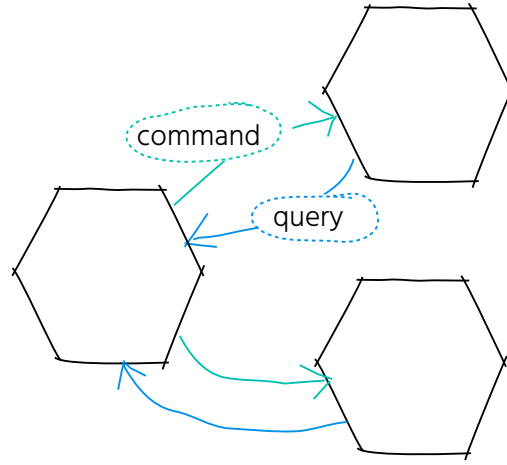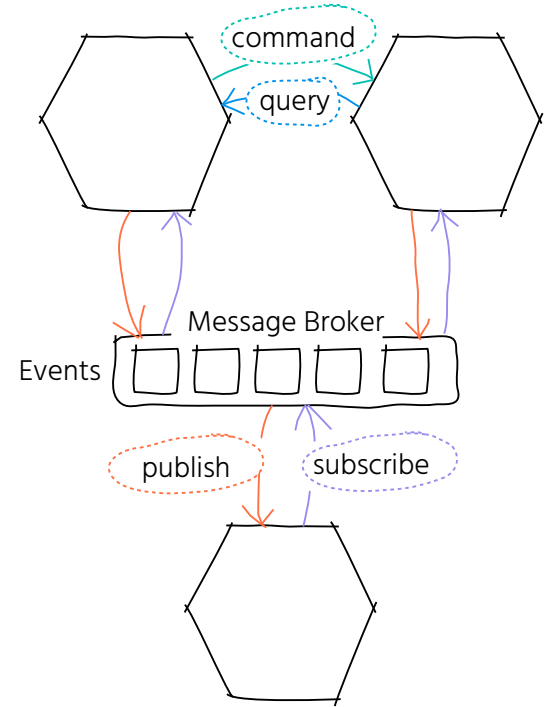
How To Manage Shared Data?

Hybrid Model

Remote query directly to source

REST API

Message Broker

Events for notification
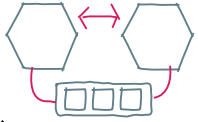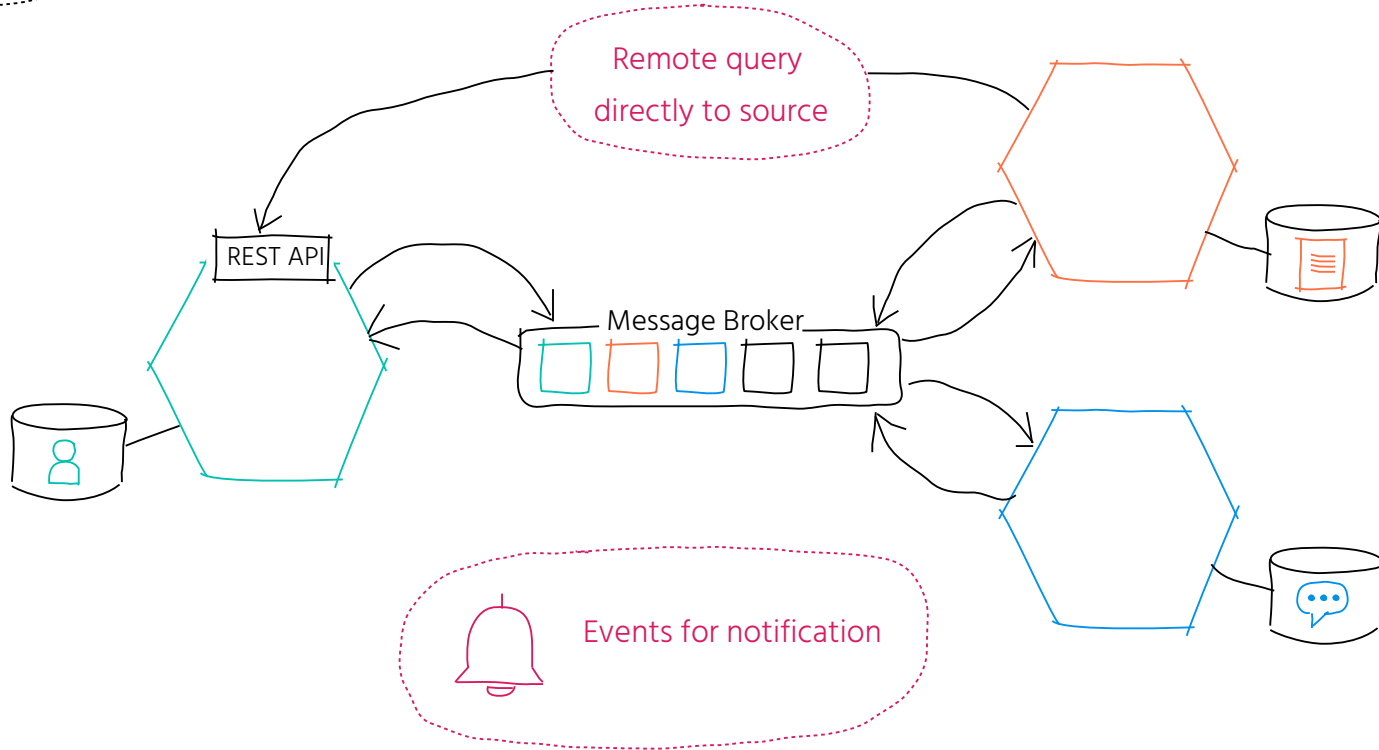
How To Manage Shared Data?
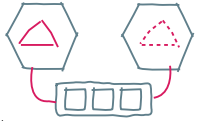
Event Driven State Transfer

ProfileUpdatedEvent

Message Broker

Local copy of profile data

Events for data duplication
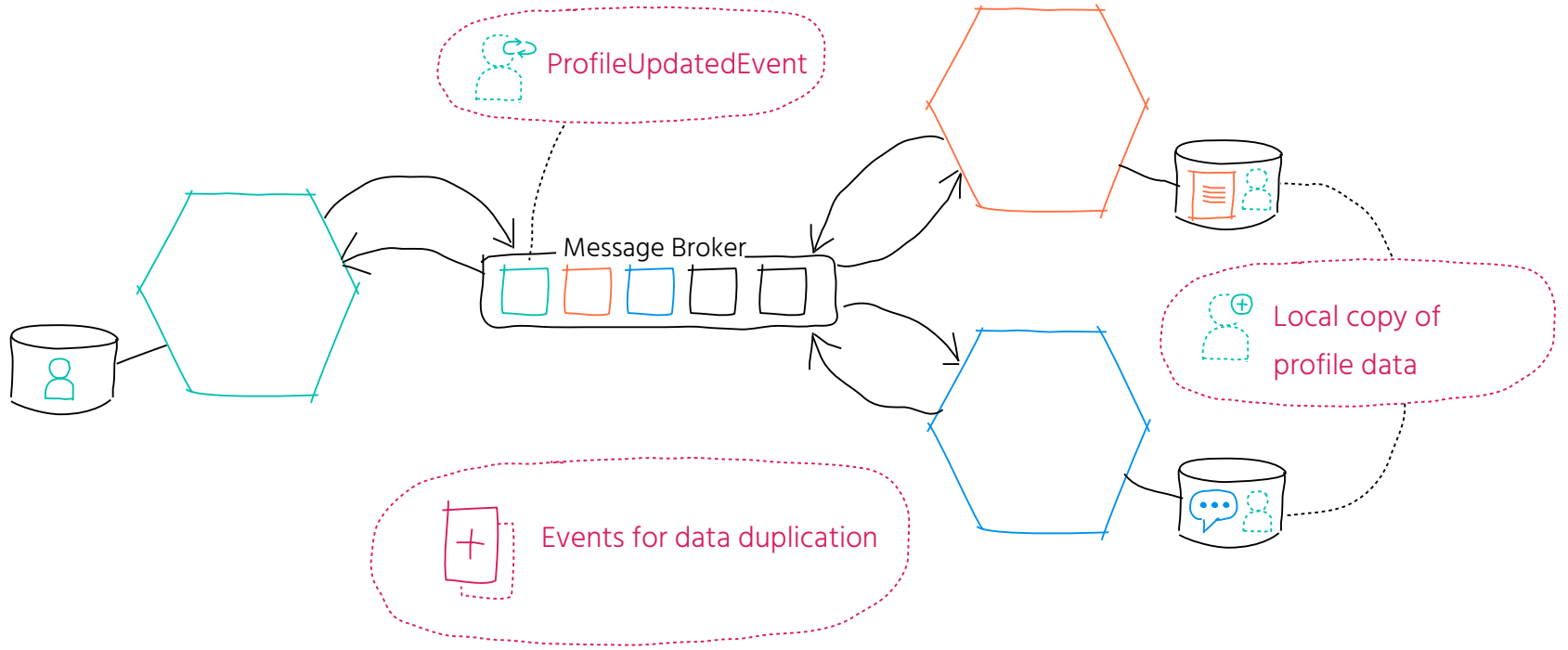
How To Manage Shared Data?
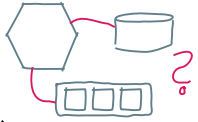
Source Of Truth

Multiple sources of truth

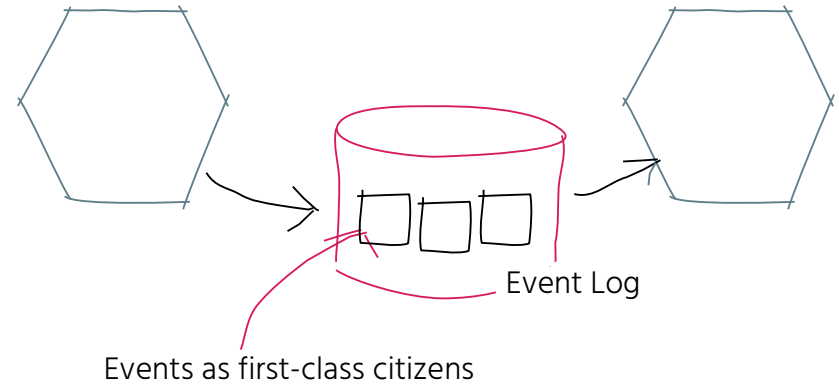Internal source of truth

External source of truth
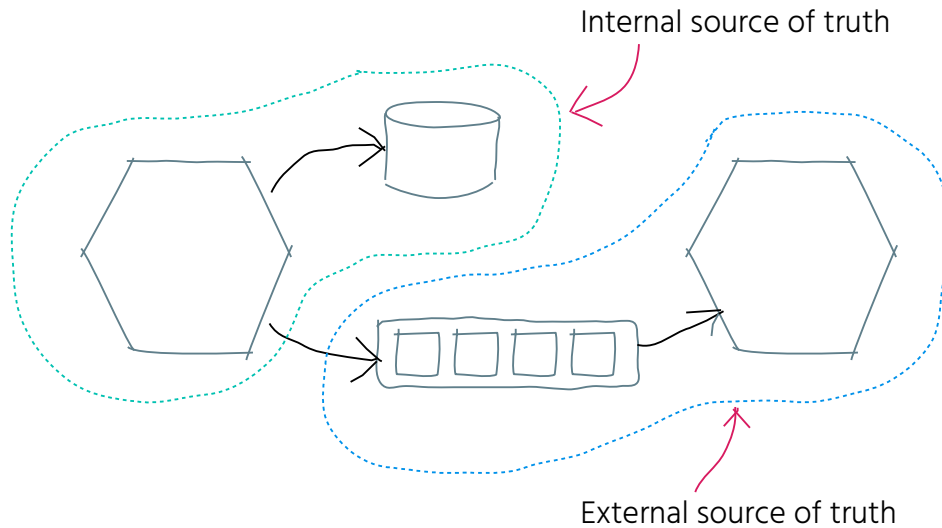
"Traditional" Event-Driven System

Single source of truth

Event Log

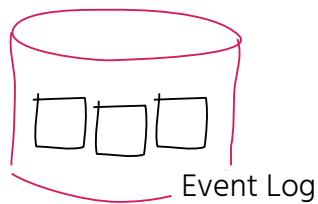Events as first-class citizens

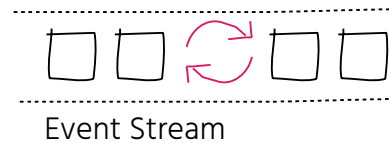Event Store

APACHE kafka®

Message Broker

Event Log

Event Stream
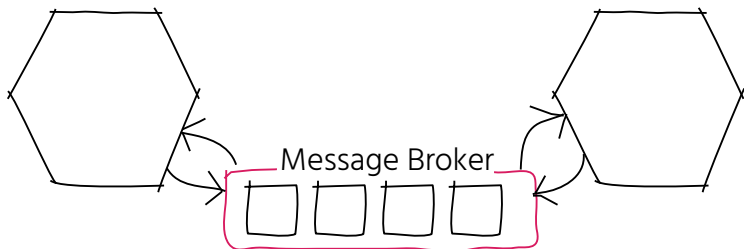
Messaging System

Storage System
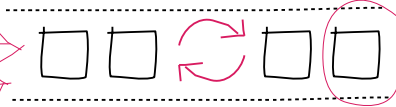
Streaming Platform

How To Manage Shared Data?

Kafka Streams

Unbounded, ordered sequence
of data records

Continuously
updating

Key-value pair

How To Manage Shared Data?

Kafka Streams

Loaded on startup of
Microservice

Running in the same process
of Microservice

Service

Kafka Topic

Lightweight embedded state store (disk-backed)
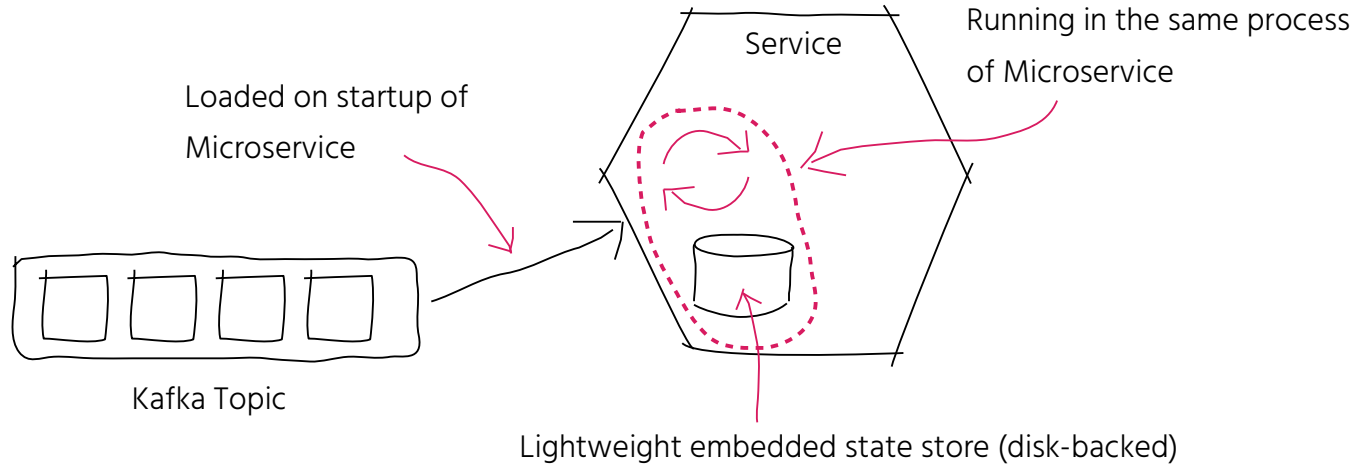
Streams make data available wherever it's needed

How To Manage Shared Data?
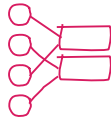
Kafka Streams

Kafka Streams API

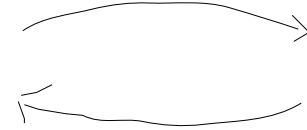→ join

filter

group by

aggregate

etc.

Kafka Stream-Table Duality

KStream

Changelog of state changes

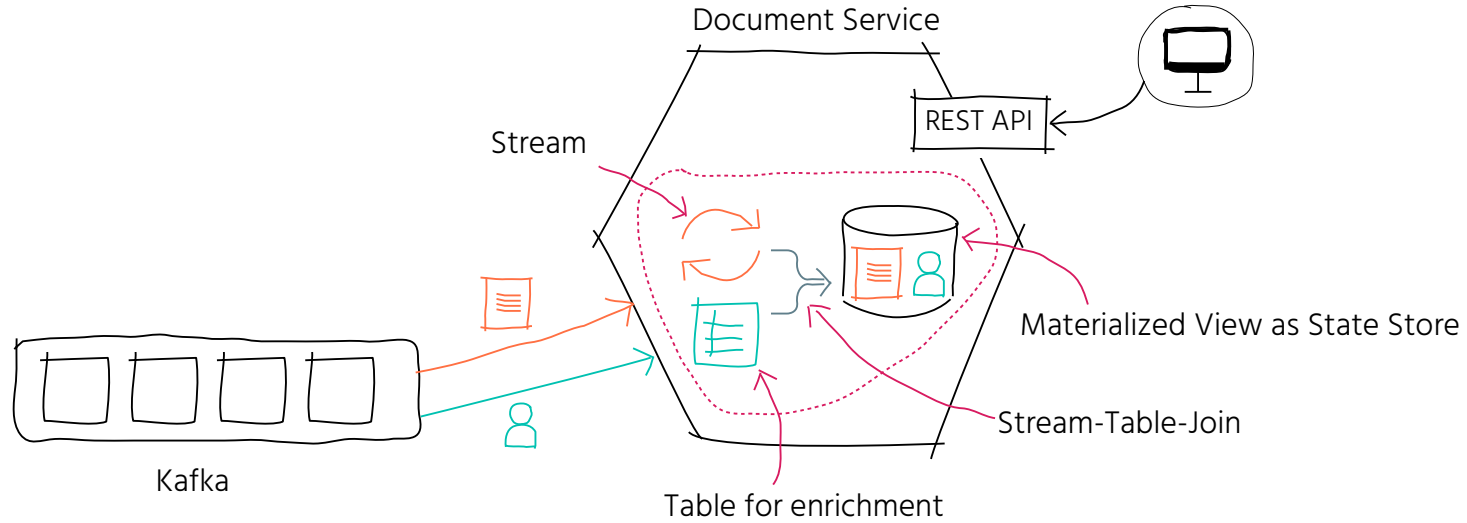(key1, value1), (key2, value2), (key 1, value 3)

KTable

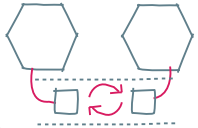Snapshot of the latest
value for each key

key1 → value3

key2 → value2

How To Manage Data?
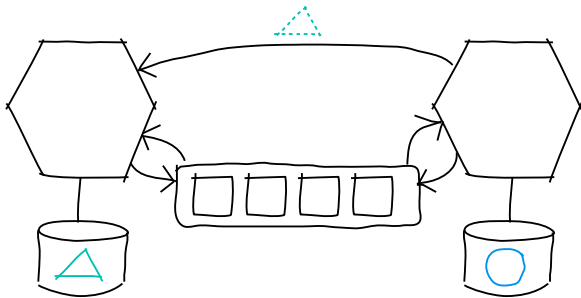
Materialized Views w/ Kafka Streams

Document Service

Stream

REST API

Materialized View as State Store

Kafka

Stream-Table-Join

Table for enrichment

How To Manage Shared Data?

Event Streams as a Shared Source of Truth

Events for notification

Events for data duplication

Event streams as a shared source of truth

- Simple integration
- Remote query => increasing coupling
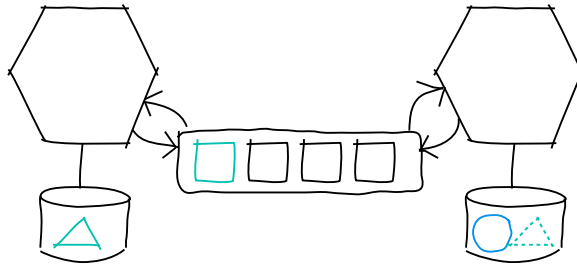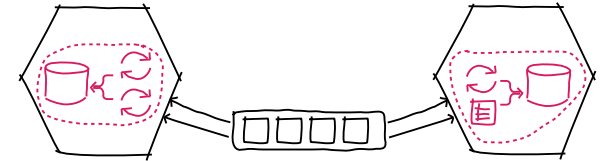
- Eliminating remote query => better decoupling
- Local copy => better autonomy
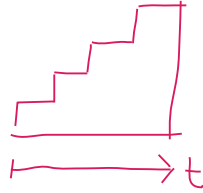- Duplicating effort to maintain local dataset

- Eliminating local copy => reduces duplicating effort
- Pushes data to where it's needed
- Increases pluggability
- Low barrier to entry for new service
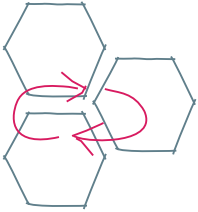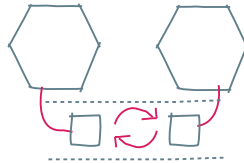
Lessons Learned

Start small

One manageable step at a time
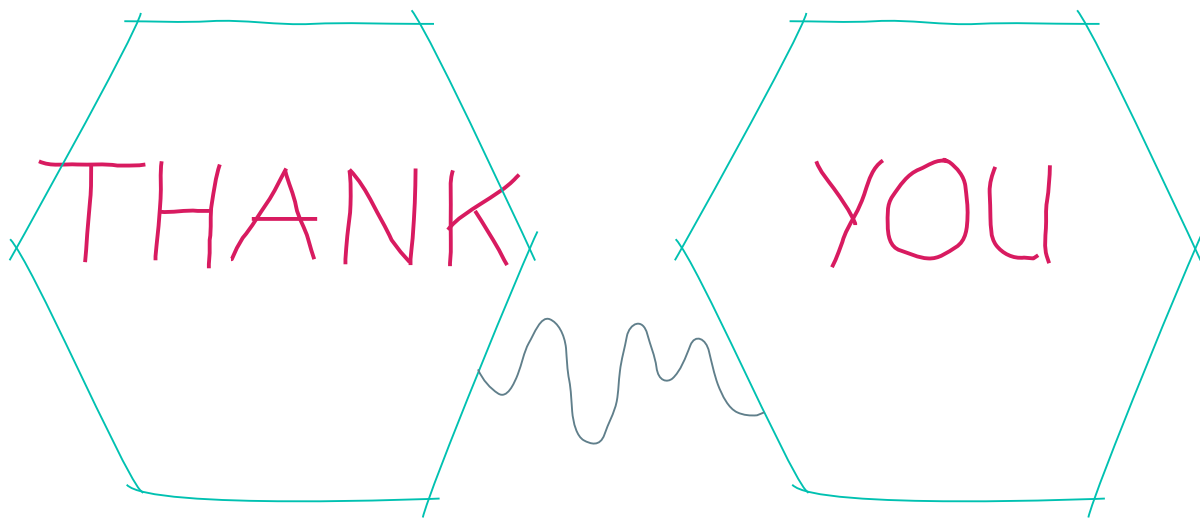
Handle cross-cutting concerns early

Avoid a distributed monolith

Design event-driven & consider
event streams as shared source of truth

Be aware of affecting circumstances
&
Each journey is different :)

THANK YOU

Susanne Kaiser

@suksr

EX-CTO at Just Software

@JustSocialApps