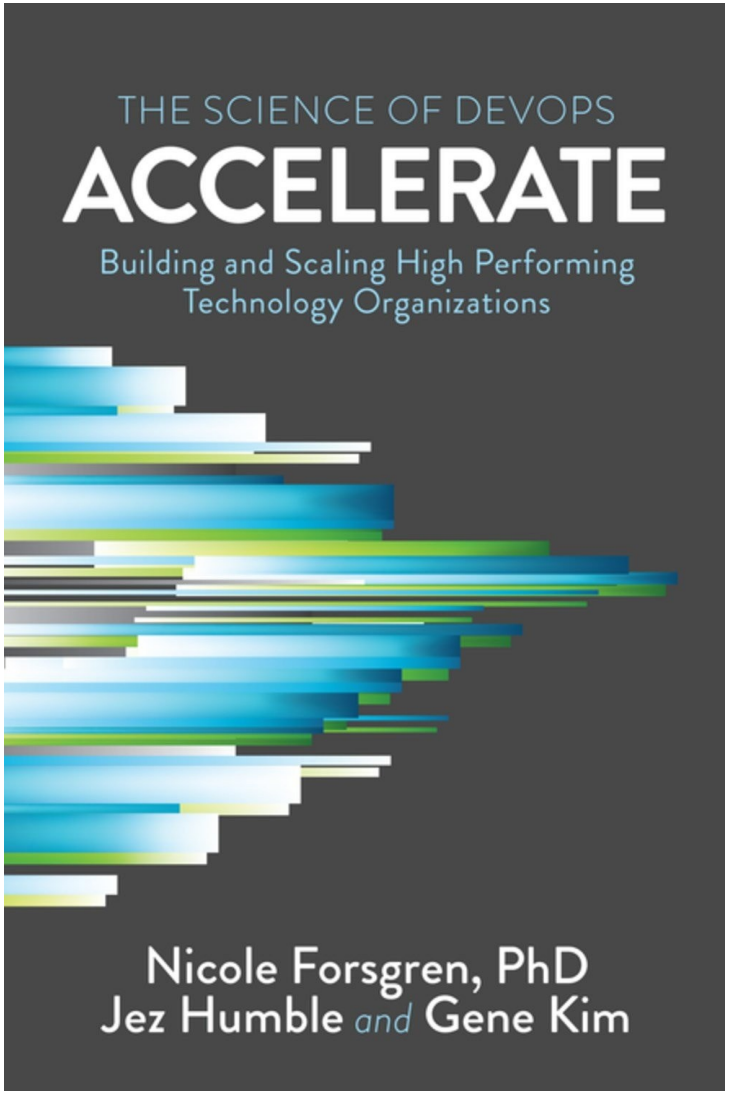
Microservices Lessons Learned

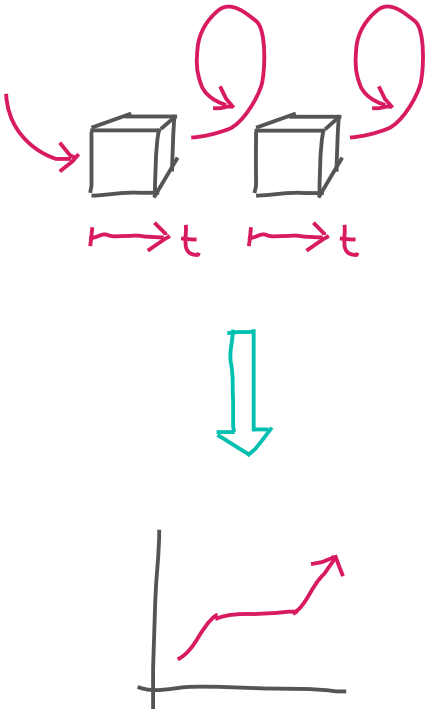
Susanne Kaiser

Independent Tech Consultant

@suksr



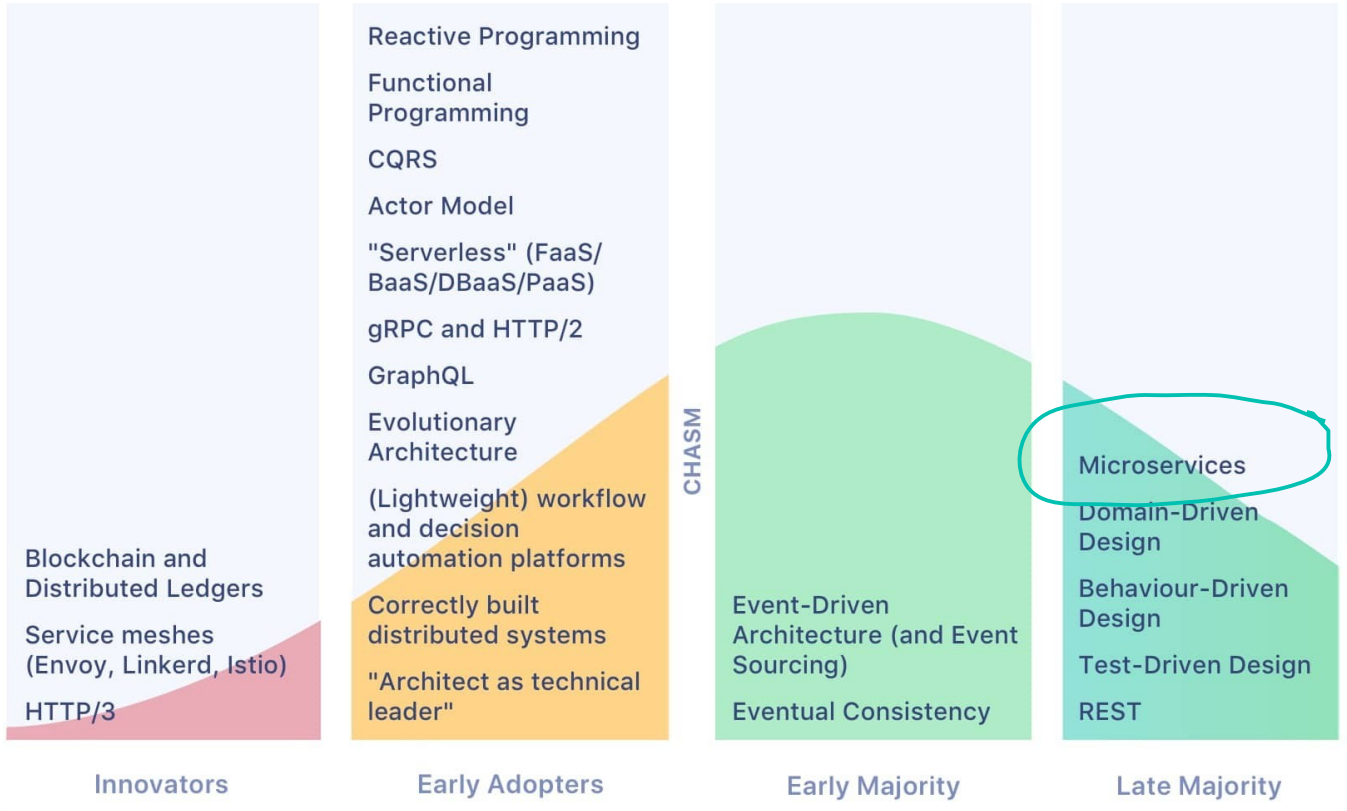
Software Delivery Performance



Profitability, Productivity & Market Share

Software Development
Architecture and Design 2019 Q1 Graph
http://infoq.link/architecture-trends-2019

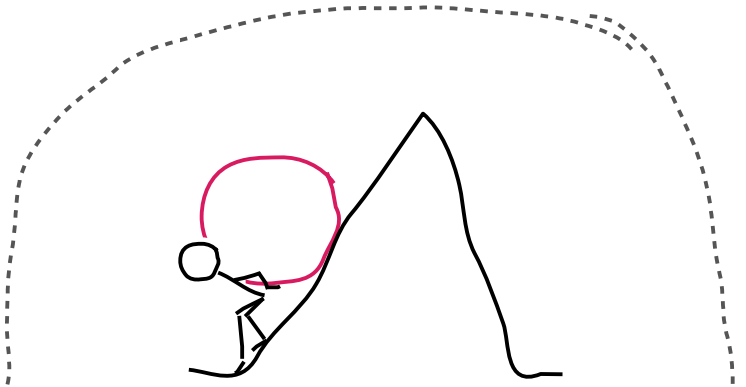
InfoQ



Software Development
Architecture and Design 2019 Q1 Graph
http://infoq.link/architecture-trends-2019

InfoQ

@suksr



Challenges Of Microservices

Background

Susanne Kaiser ... who?

Independent Tech Consultant
@suksr

EX- CTO
at Just Software
@JustSocialApps

Co-Organizer @microXchg, Berlin

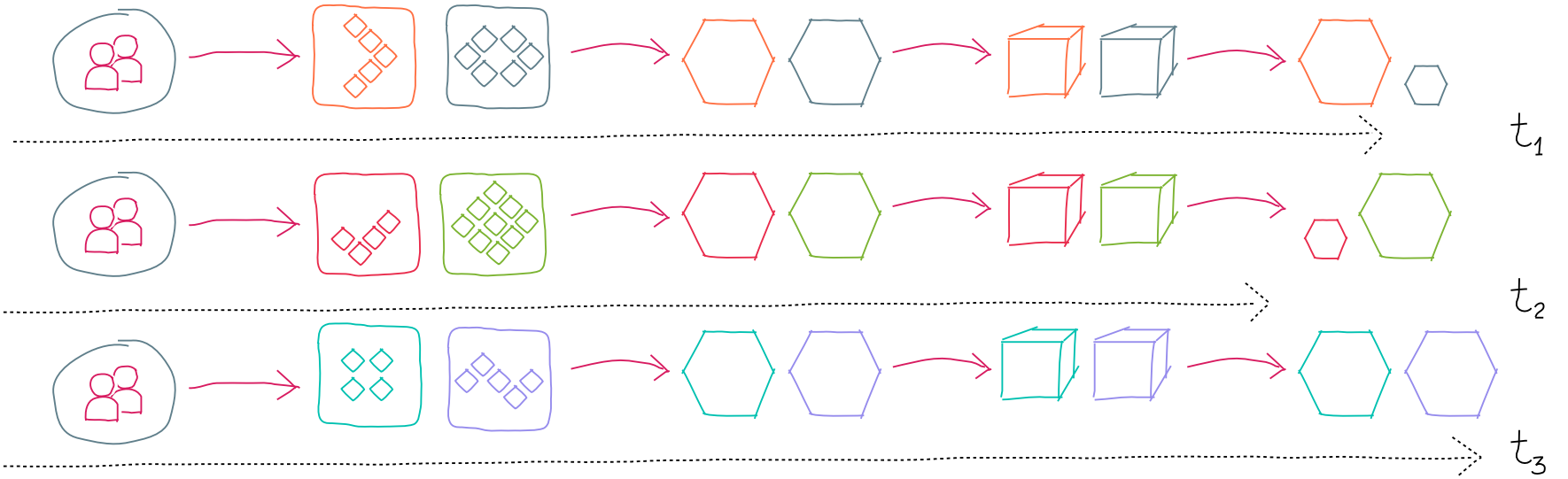
Program committee &
track host
QCon, New York
@qconnewyork

Program committee
MicroCPH, Copenhagen
@MicroCPH

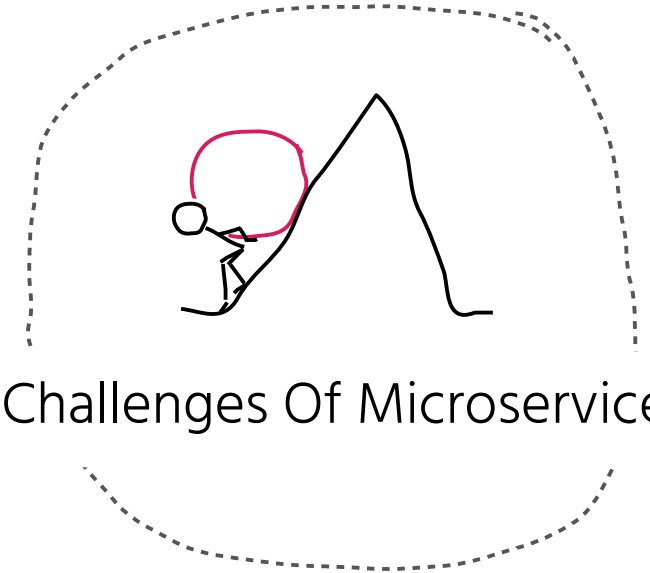
Program committee
ServerlessDays, Hamburg
@ServerlessHAM

Background

Motivation for Microservices

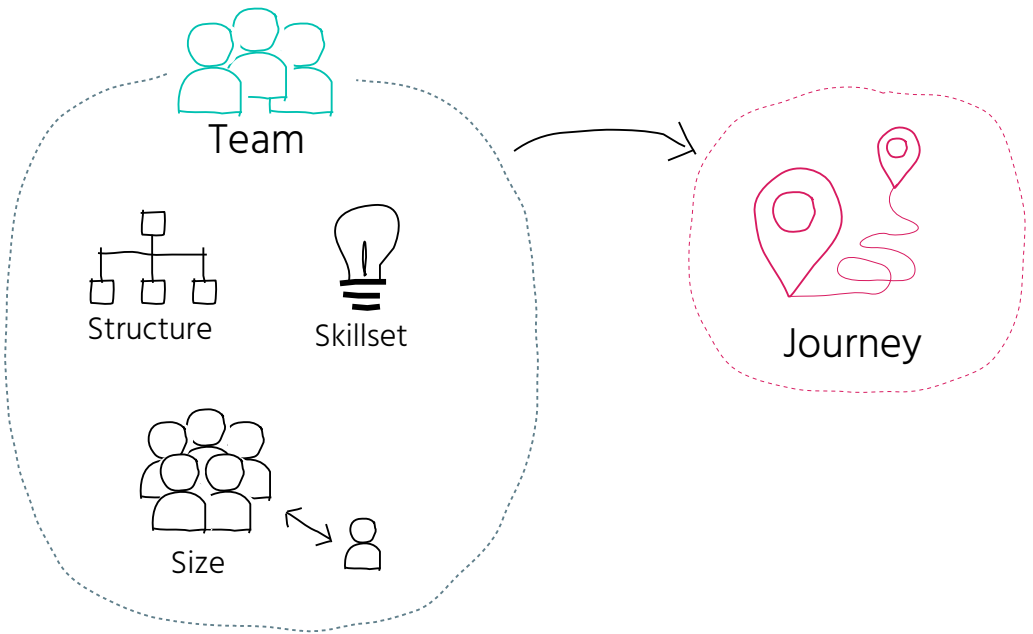


1. Organizational Circumstances

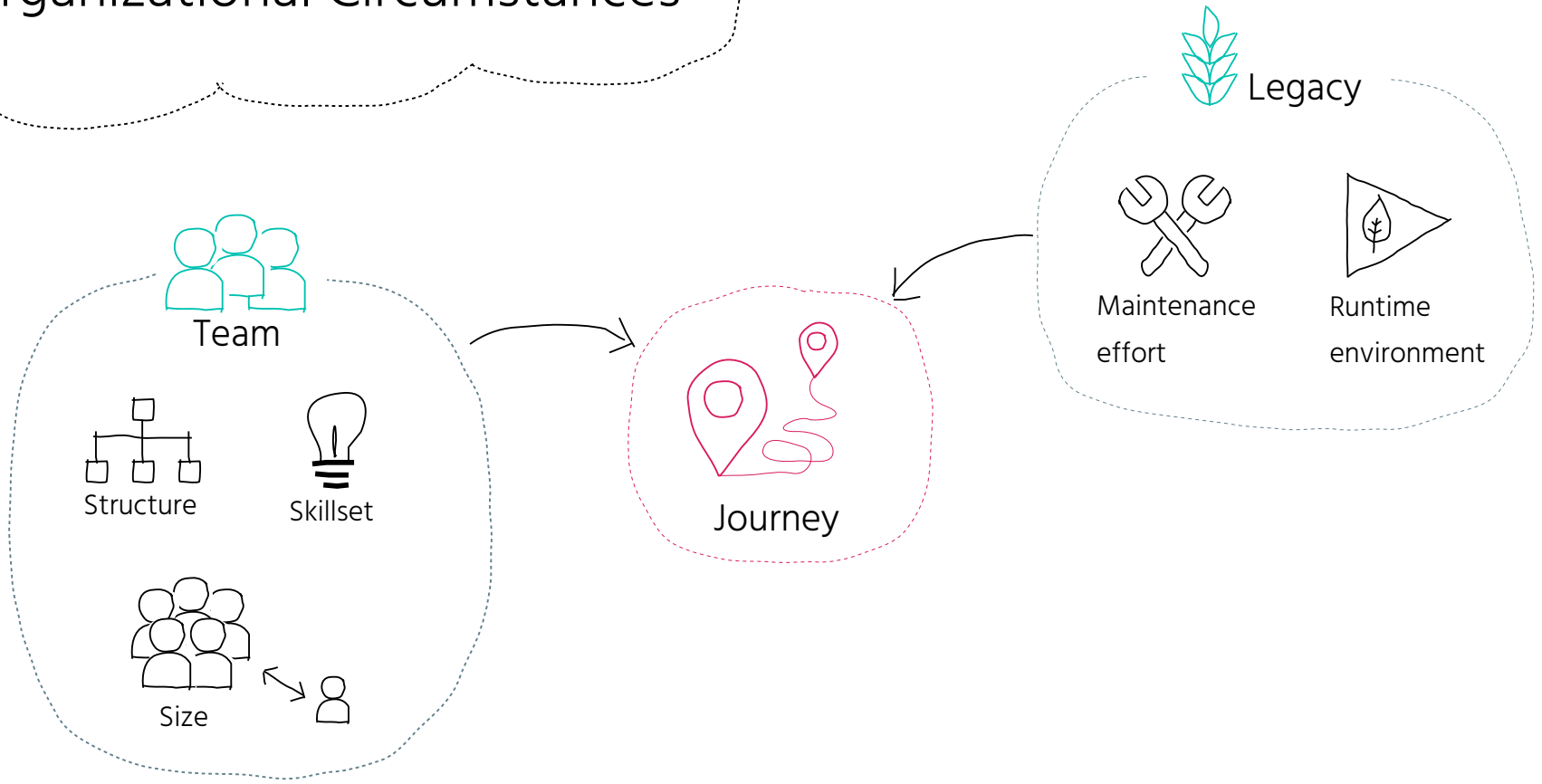


Challenges Of Microservices

Organizational Circumstances

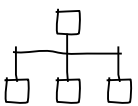


Organizational Circumstances



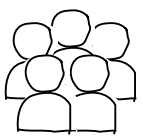
Organizational Circumstances

Team



Structure

Skillset

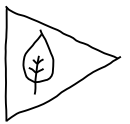


Size

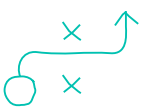
Journey

Legacy

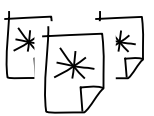
Maintenance effort



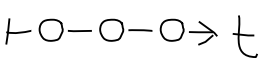
Runtime environment



Strategy



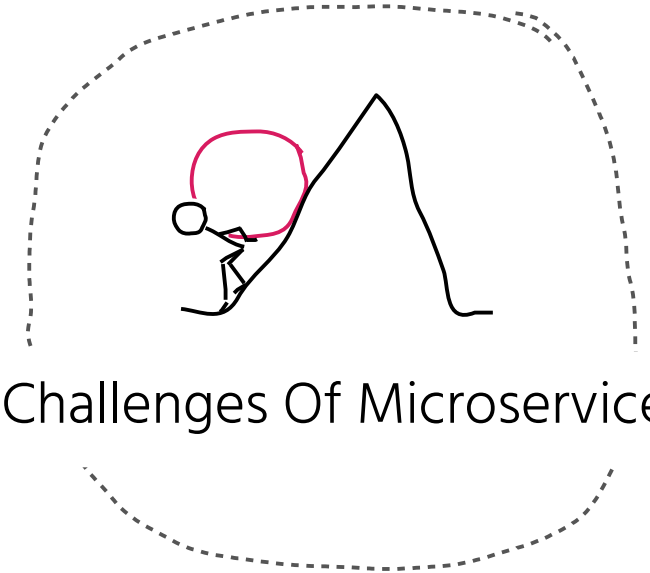
New Features



Timeline / Milestones

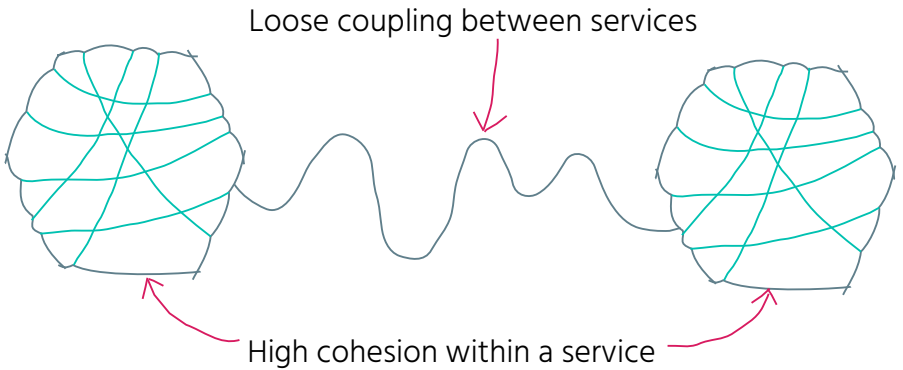
2. Manageable Steps

1. Organizational Circumstances

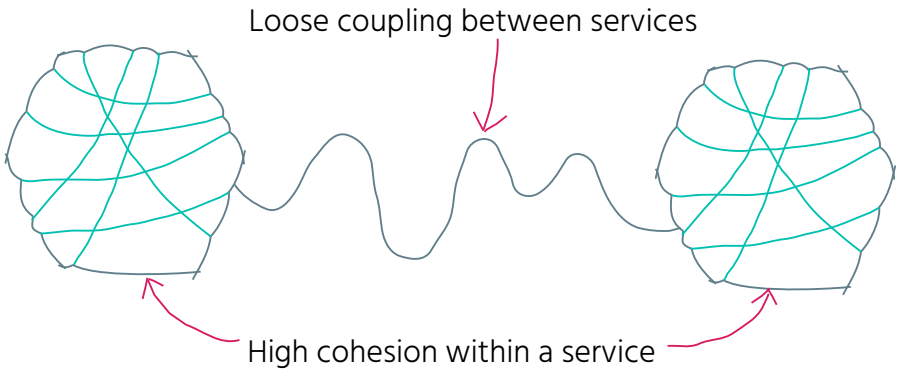


Challenges Of Microservices

Identify Bounded Contexts



Identify Bounded Contexts



Bounded Context

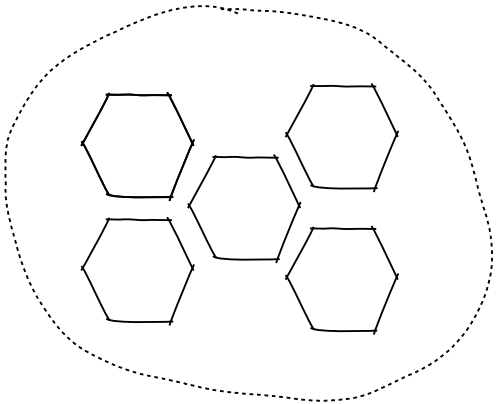
Related behaviour

Well-defined business function

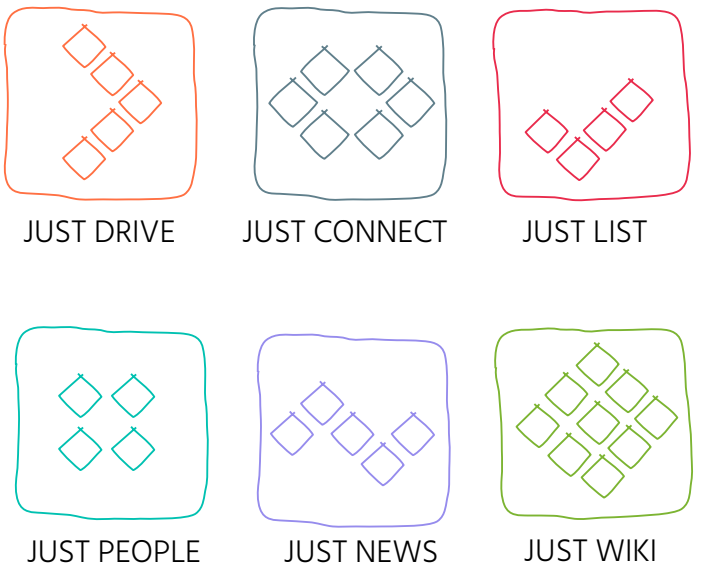
Semantic boundary around domain model

Examples for Bounded Contexts

Bounded Contexts

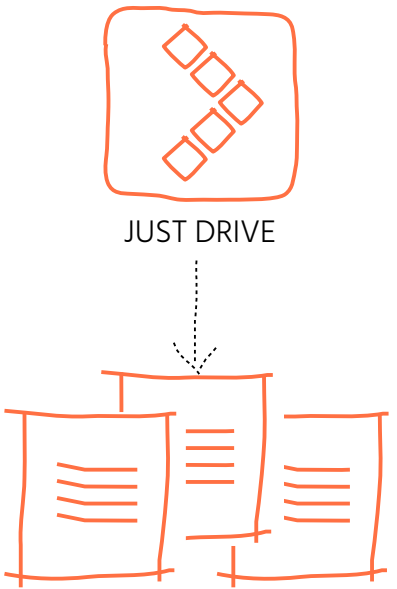


=



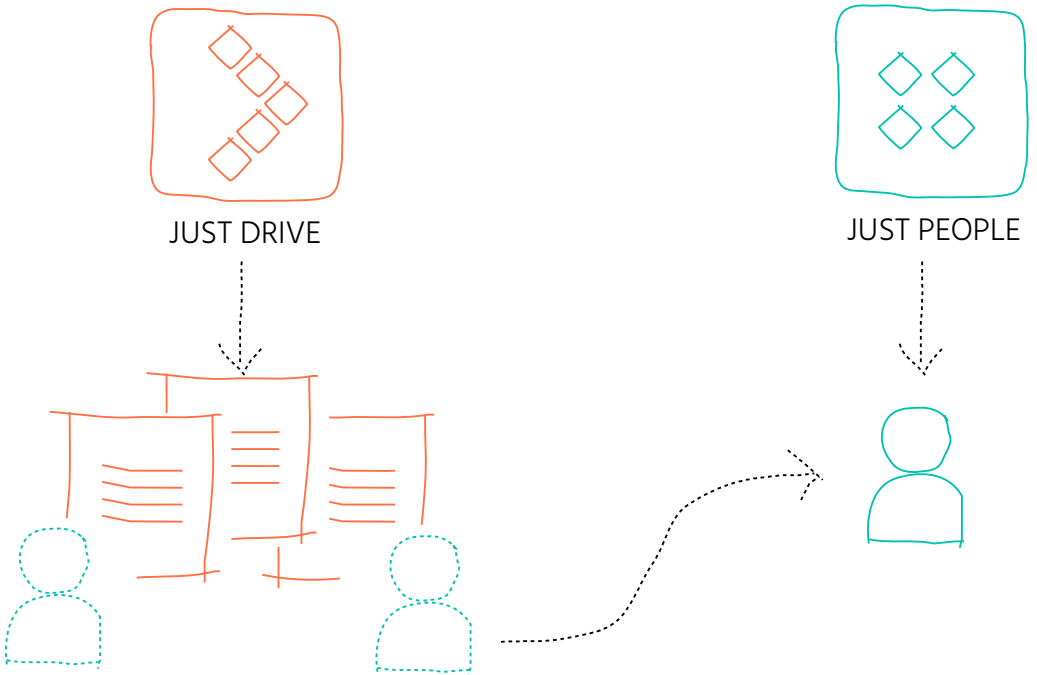
Decomposition Strategy

Co-Existing Service From Scratch



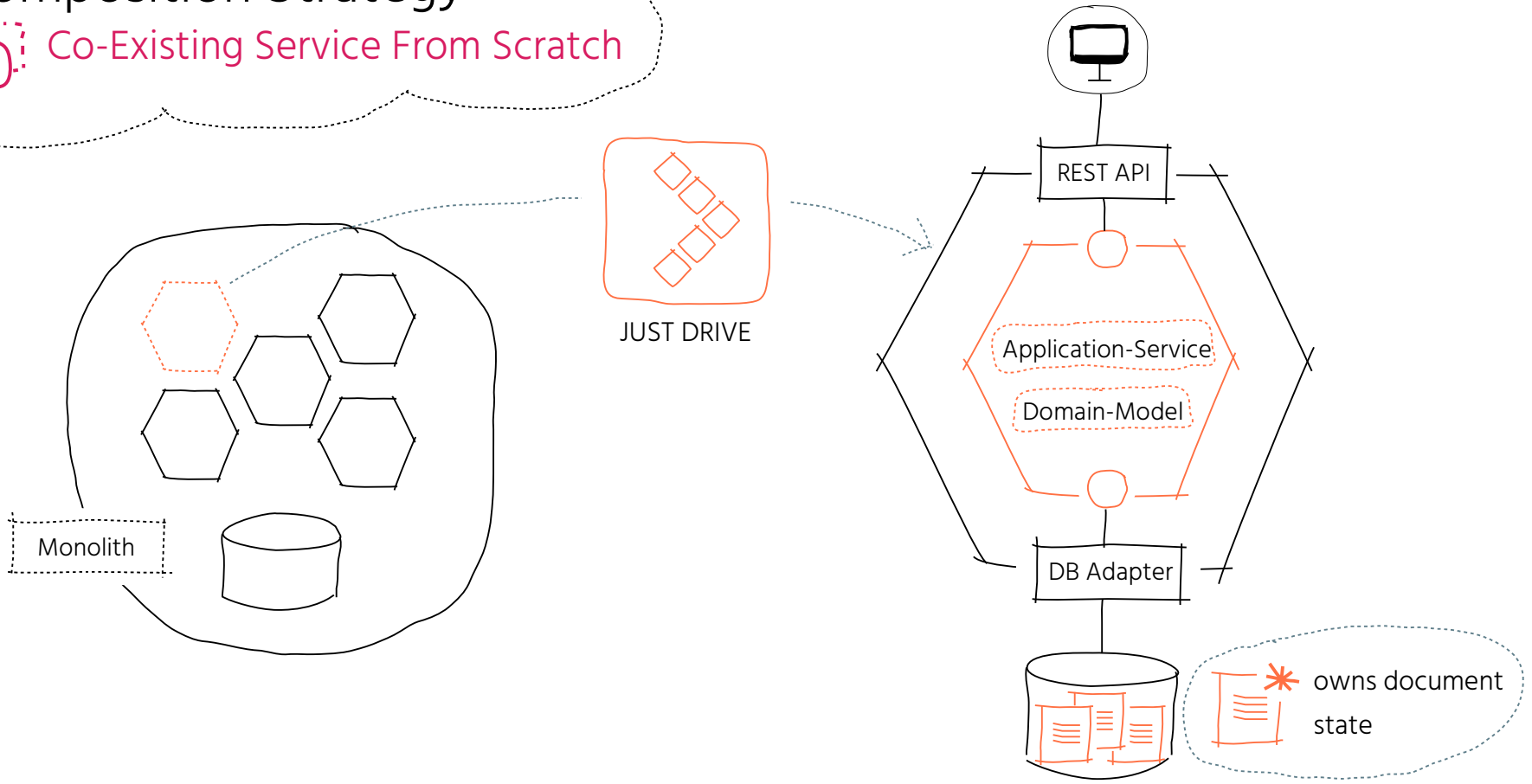
Decomposition Strategy

Co-Existing Service From Scratch



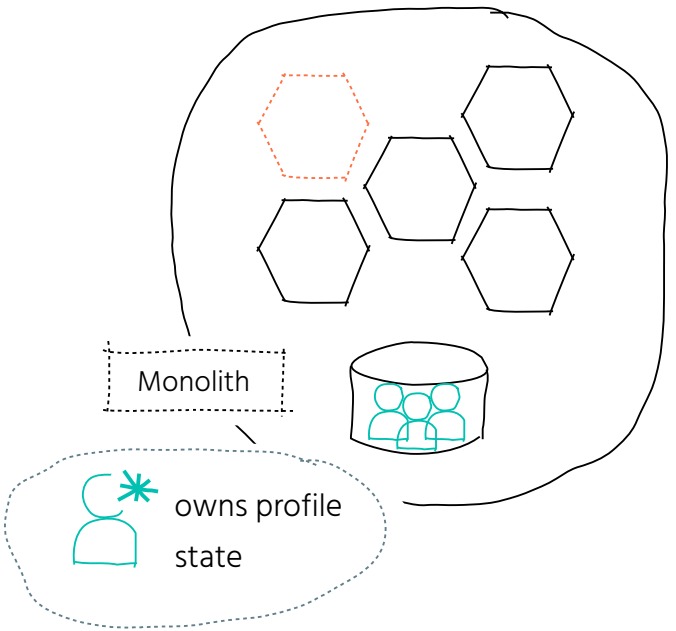
Decomposition Strategy

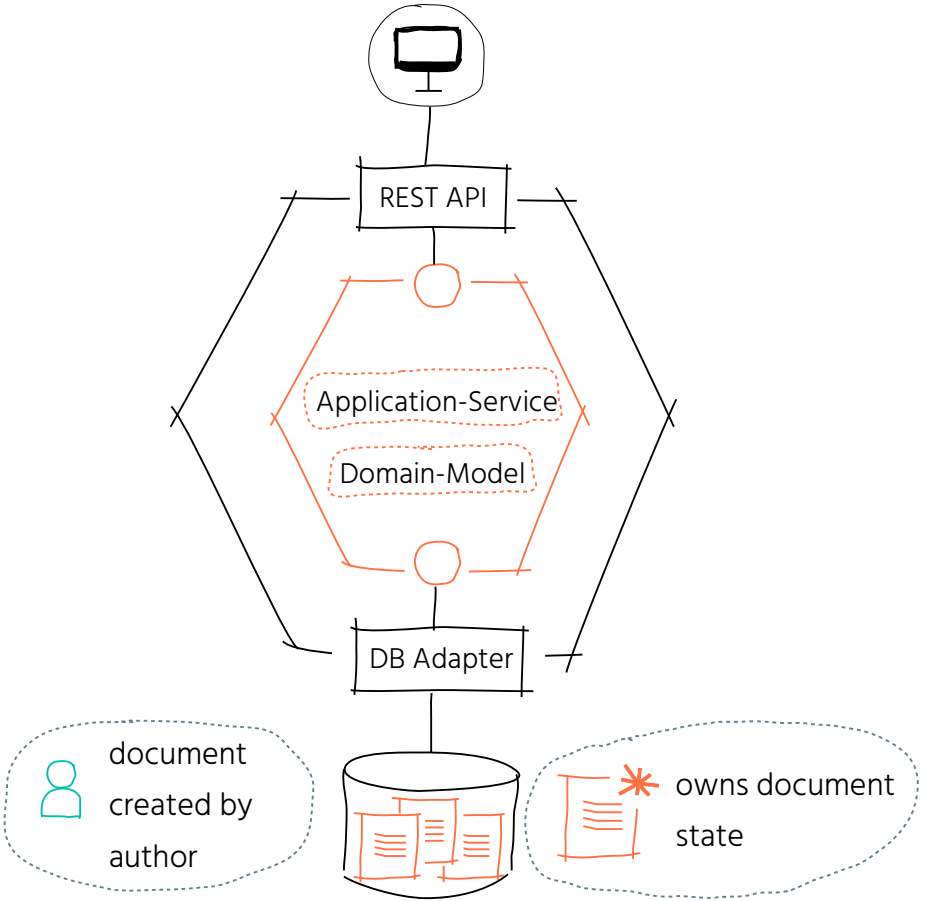
Co-Existing Service From Scratch



Decomposition Strategy

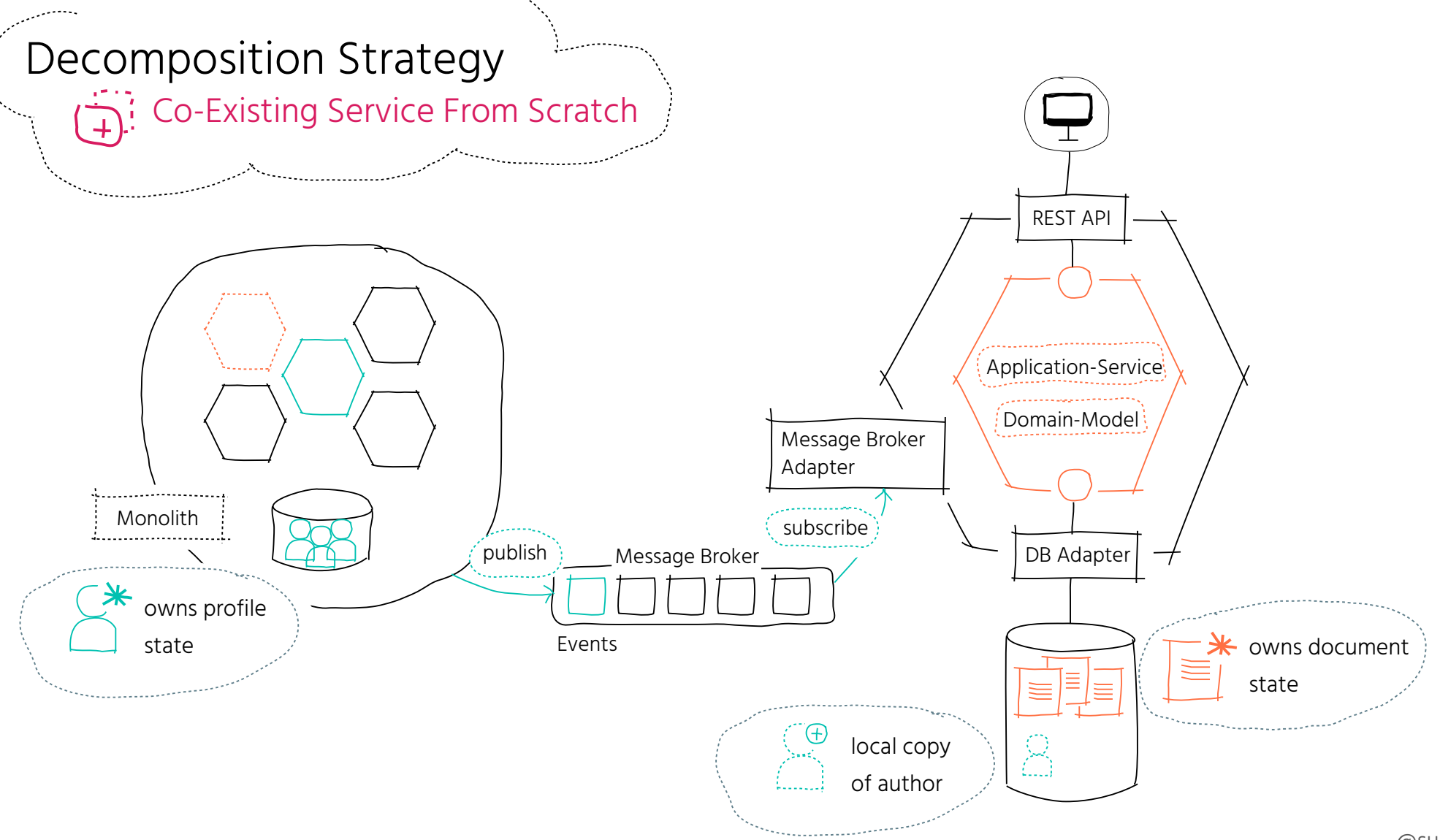
Co-Existing Service From Scratch





Decomposition Strategy

Co-Existing Service From Scratch



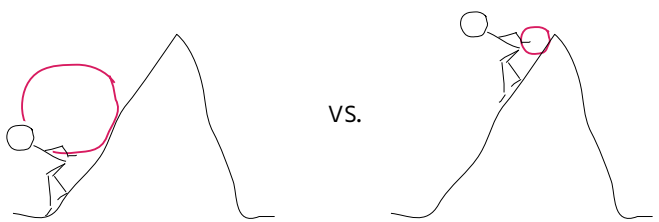
Decomposition Strategy

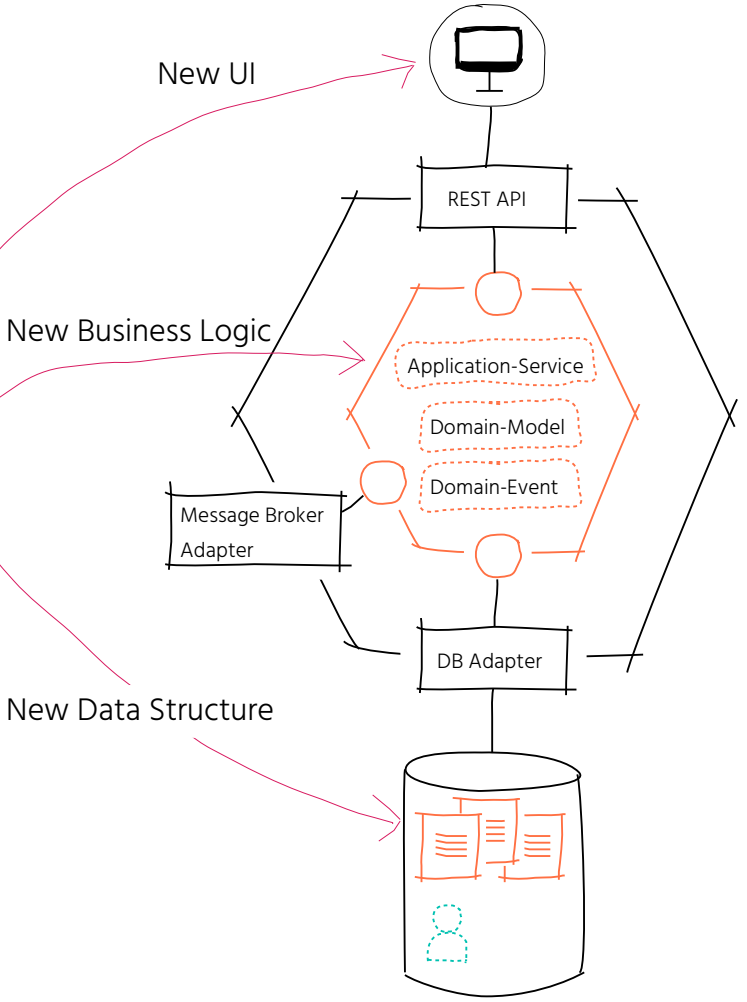
Co-Existing Service From Scratch

Good approach in general,

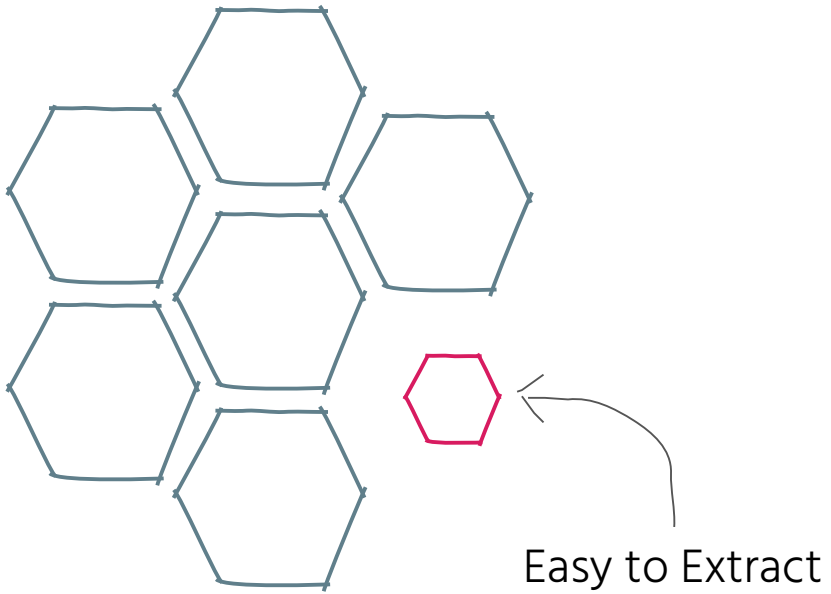
but we did too many steps at once

=> Not optimal to start with

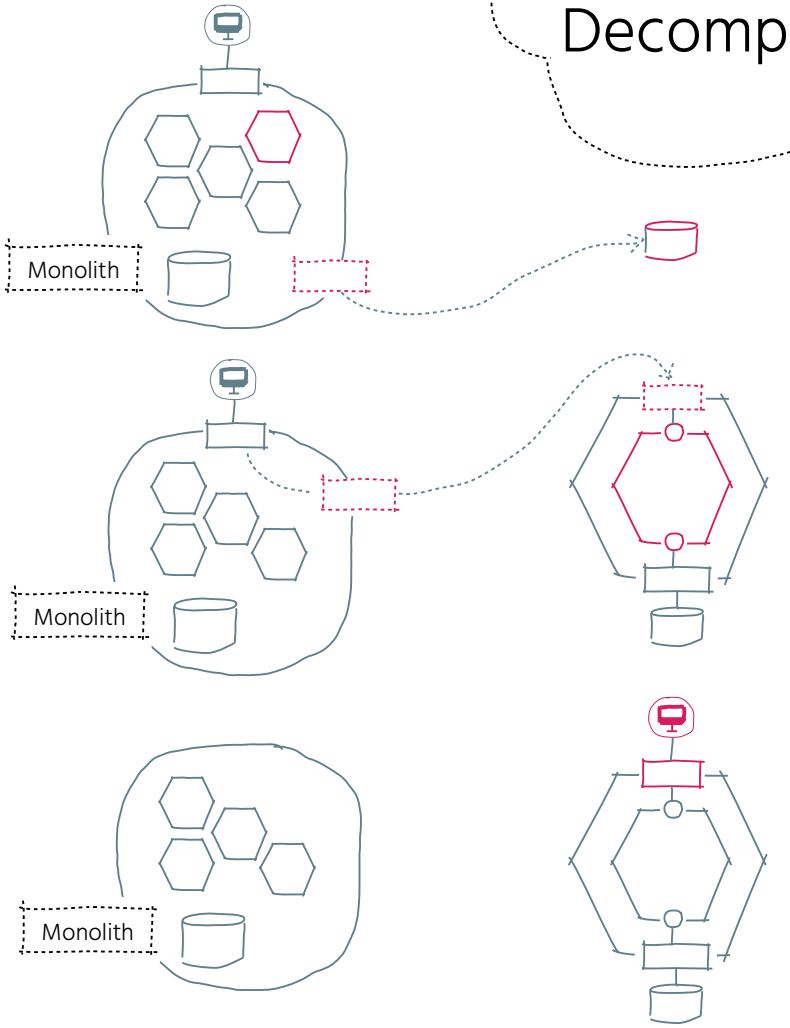




Start Small

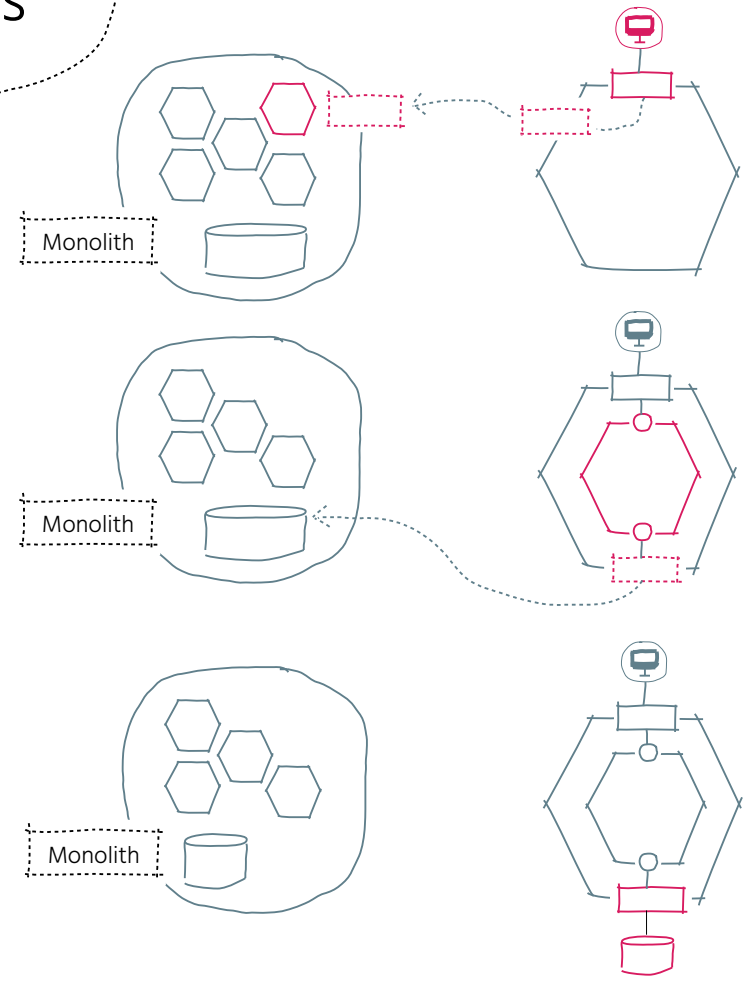


Decompose in Steps



Incremental Decomposition → Bottom-Up

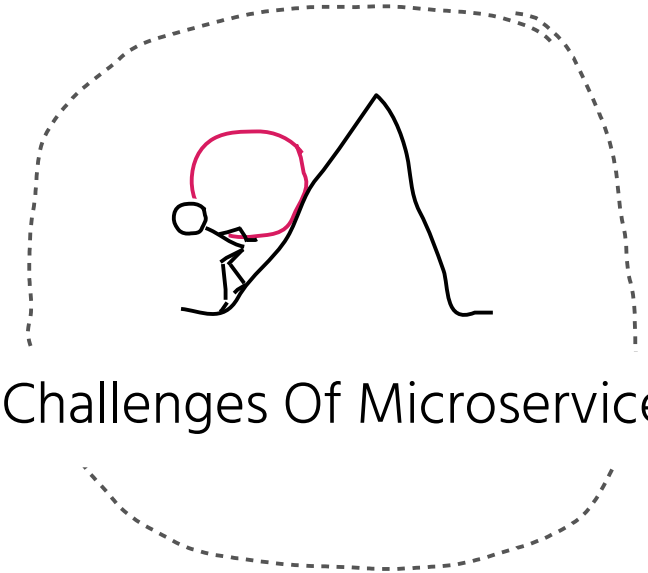
- or -



Incremental Decomposition → Top-Down

2. Manageable Steps

1. Organizational Circumstances

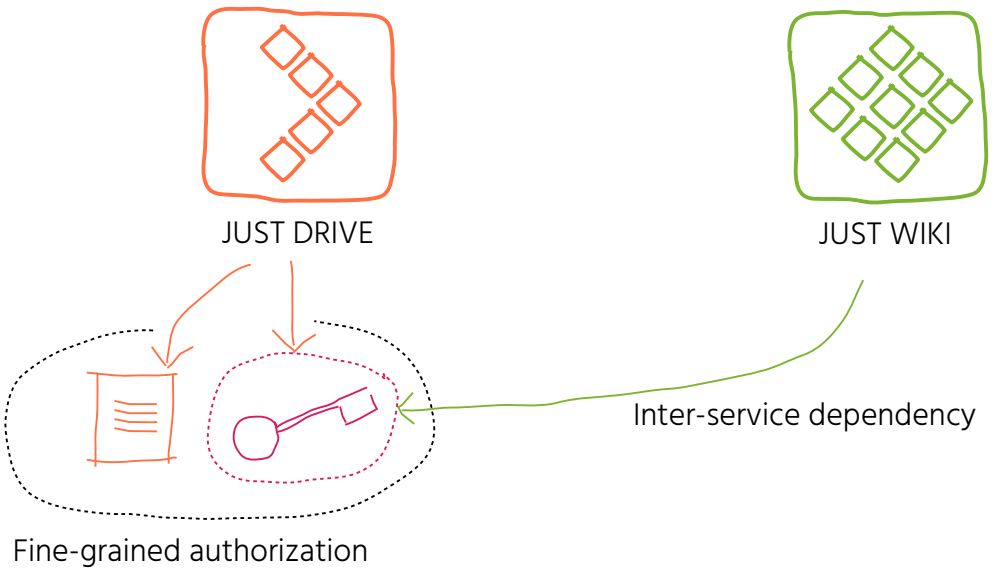


3. Cross-Cutting Concerns

Challenges Of Microservices

Cross-Cutting Concerns

Authorization



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.

Feeding the monolith

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

Re-implementing authz w/ every new service

Cross-Cutting Concerns

Handle Them Early

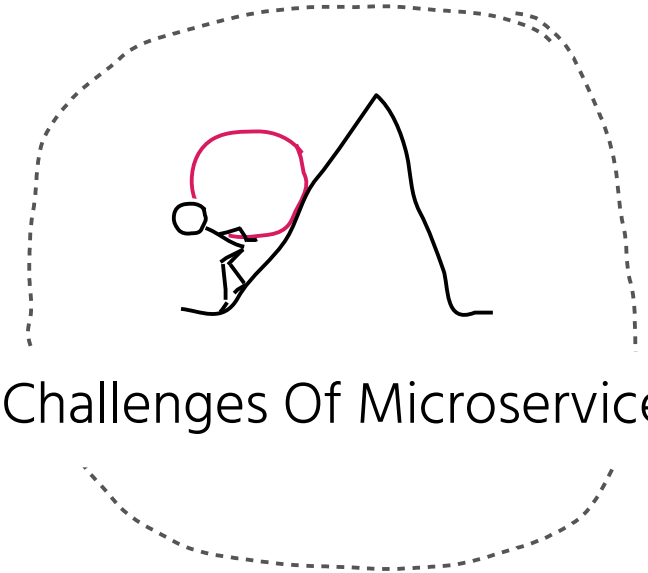
Feeding the monolith

Re-implementing authz w/ every new service

Handle Cross-Cutting Concerns Early

2. Manageable Steps

1. Organizational Circumstances



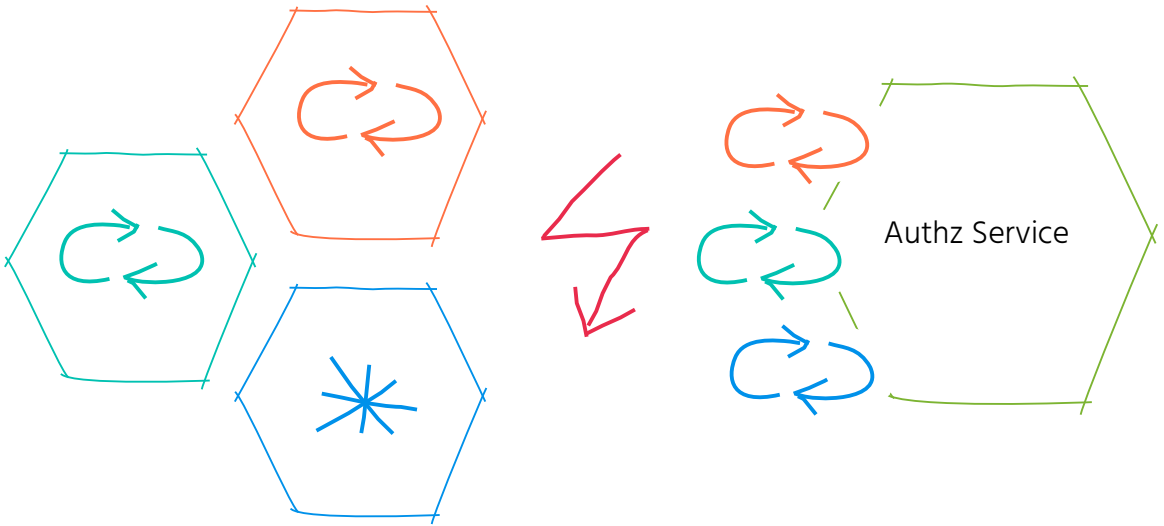
3. Cross-Cutting Concerns

Challenges Of Microservices

4. Distributed Monolith

Cross-Cutting Concerns

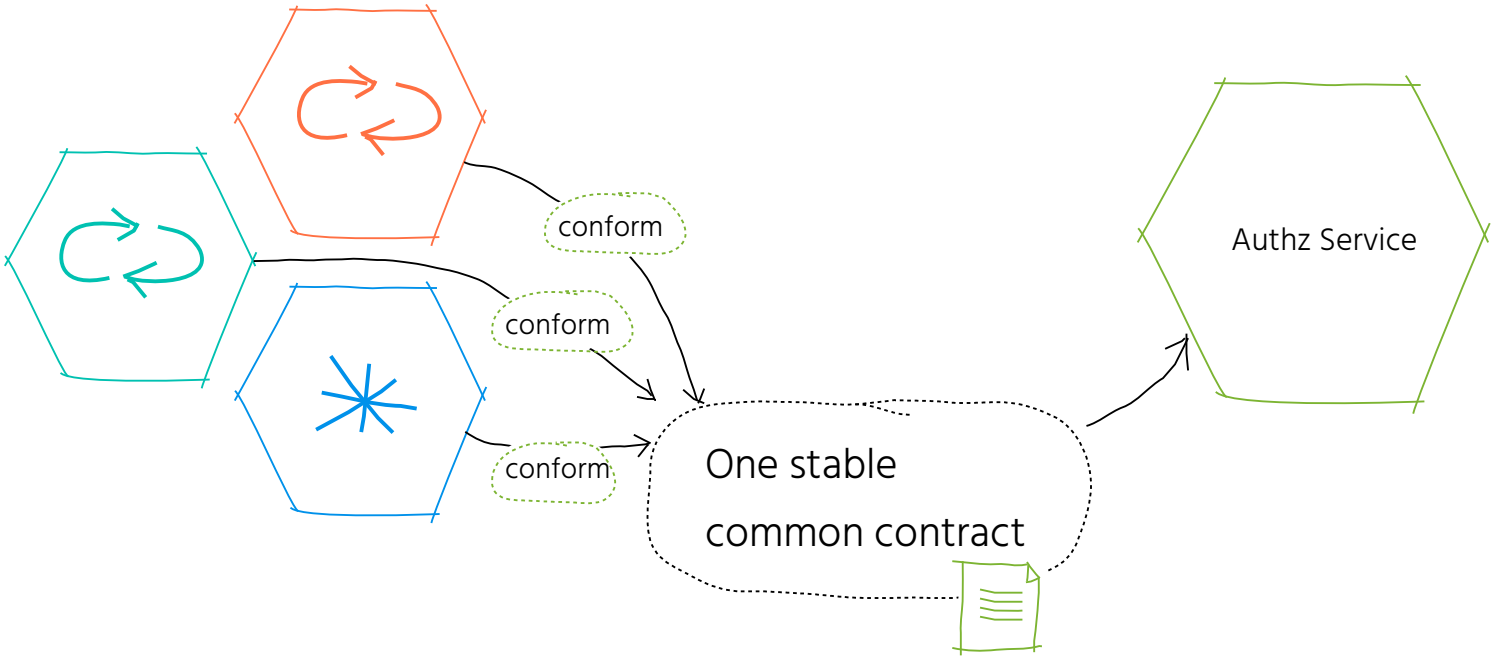
Avoid A Distributed Monolith



Does a change to one microservice require changes to or deployments of other microservices?

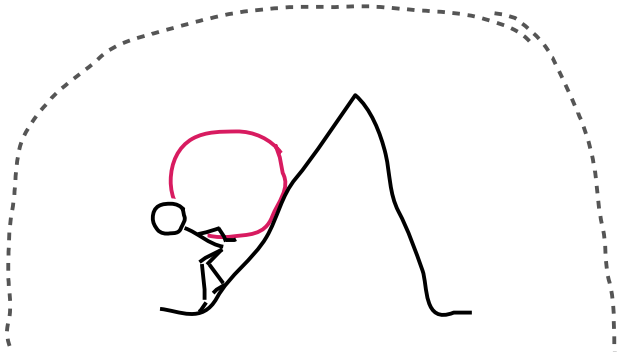
Cross-Cutting Concerns

Avoid A Distributed Monolith



2. Manageable Steps

1. Organizational Circumstances



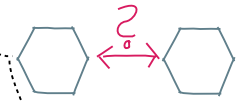
3. Cross-Cutting Concerns

Challenges Of Microservices

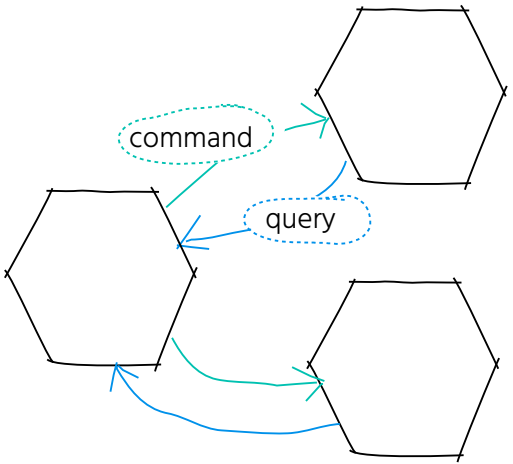
4. Distributed Monolith

5. Service-Interaction, Shared Data & Event-Patterns

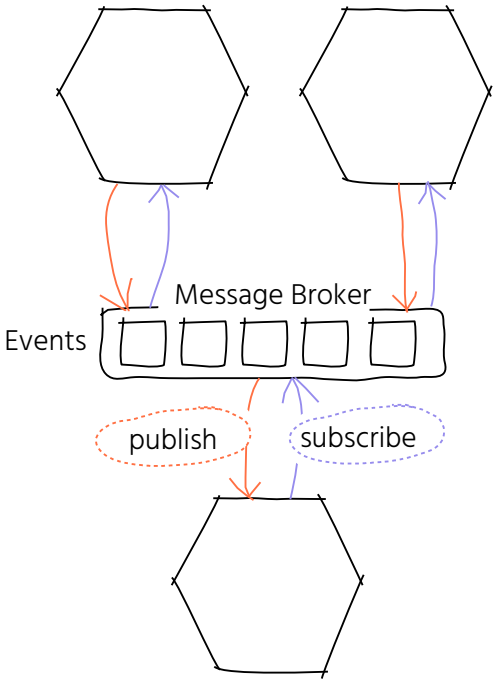
Service Interaction



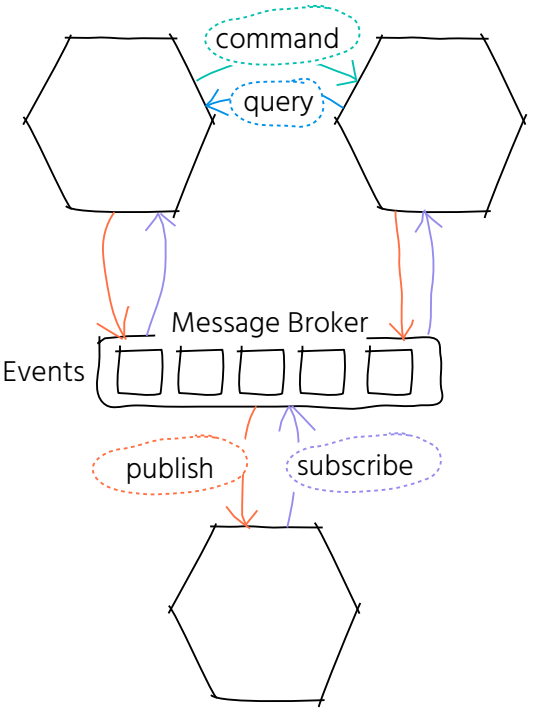
Request-Driven / Event-Driven



Request-Driven

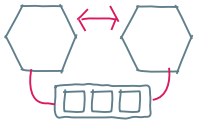


Event-Driven

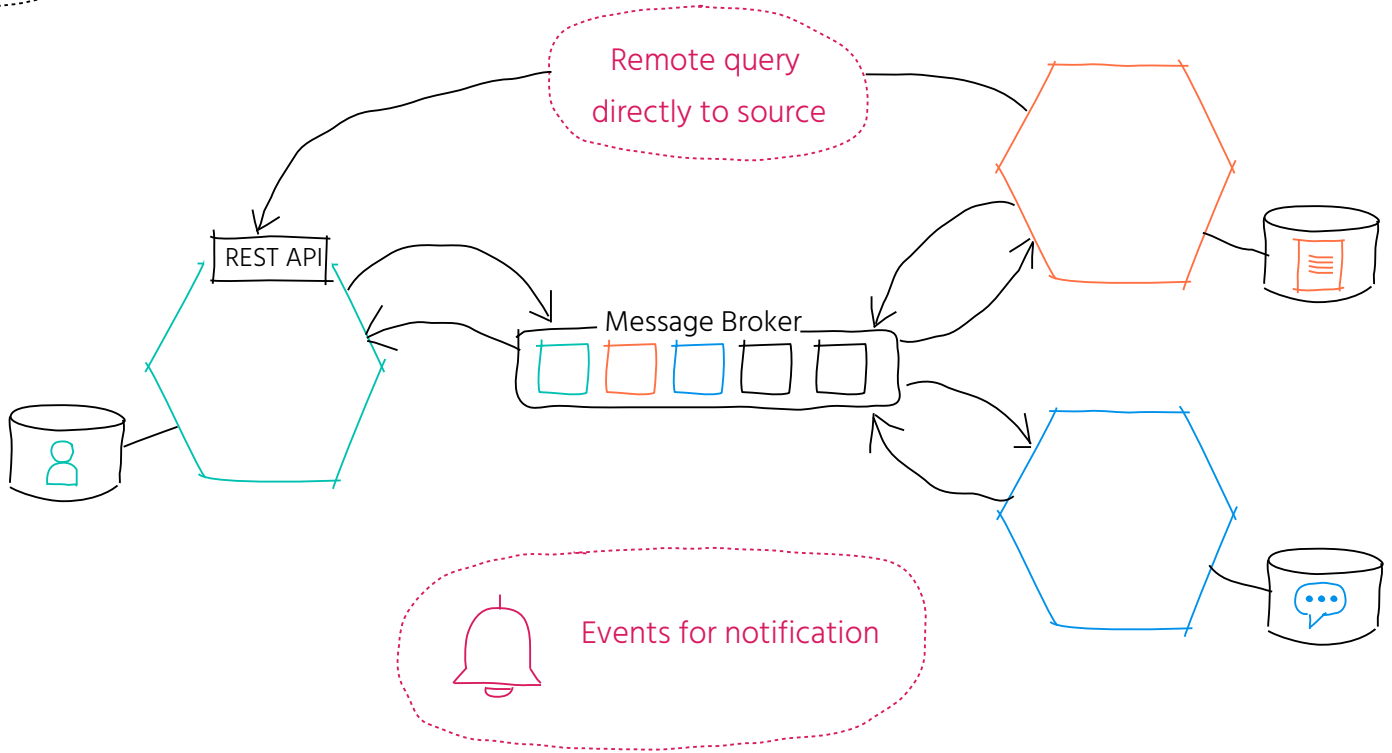


Hybrid

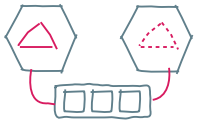
How To Manage Shared Data?



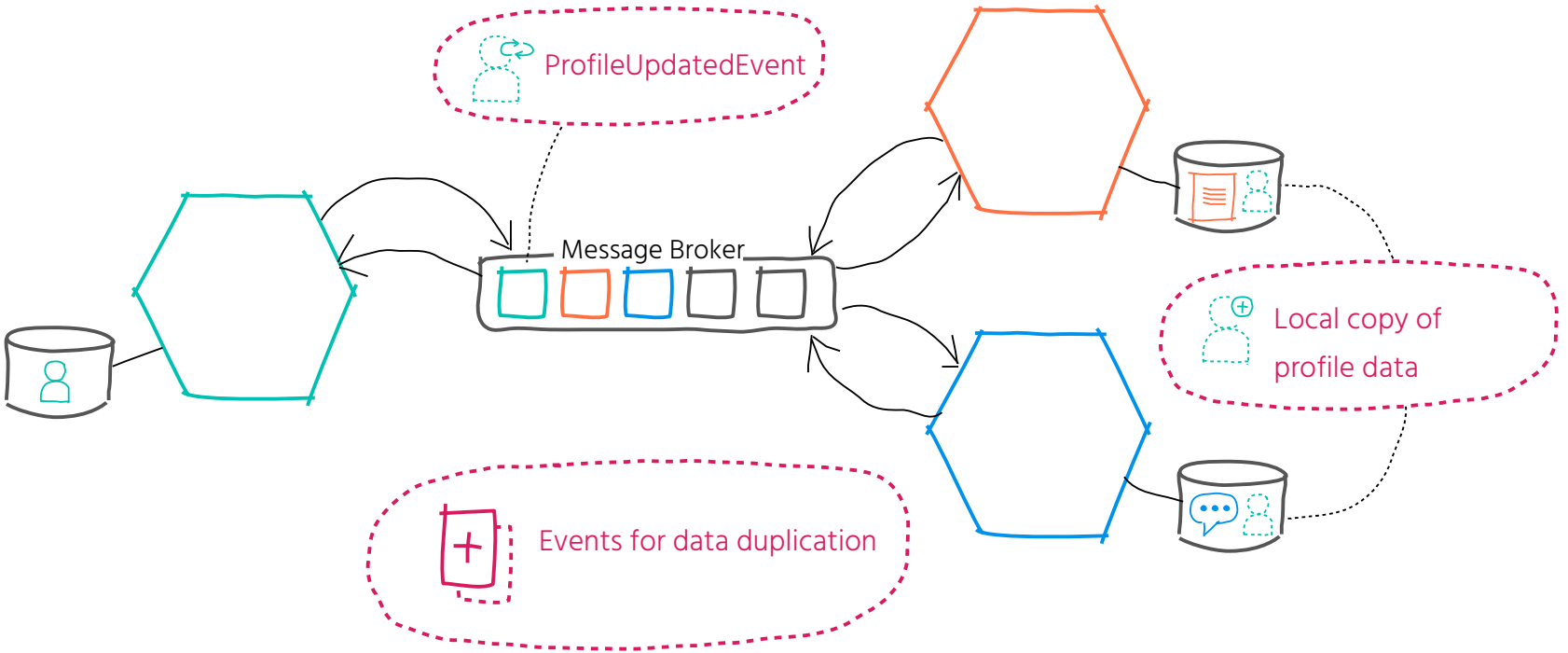
Hybrid Model



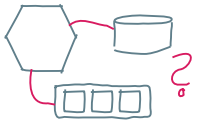
How To Manage Shared Data?



Event Driven State Transfer

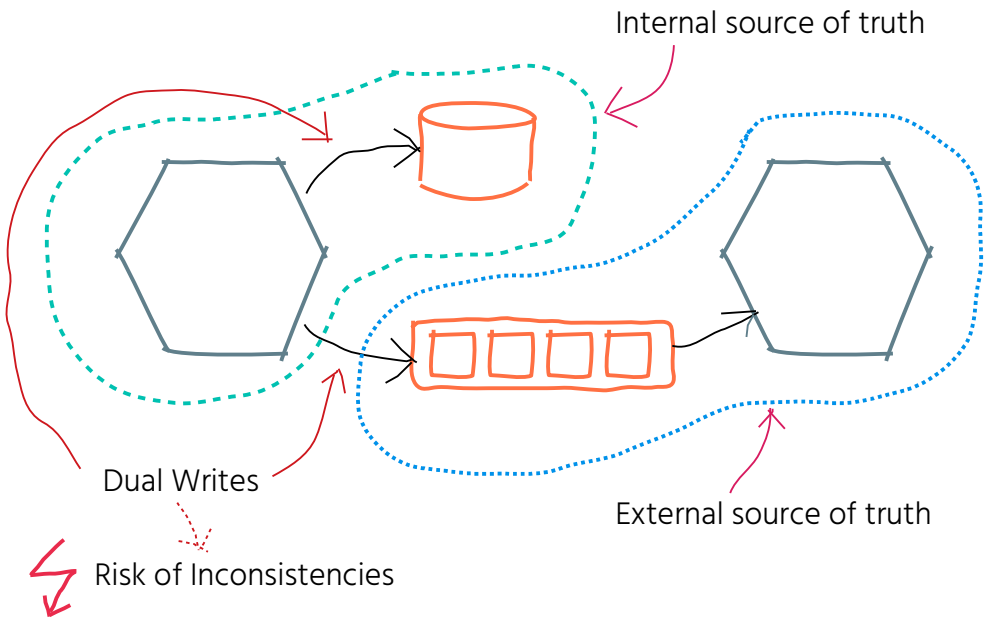


How To Manage Shared Data?



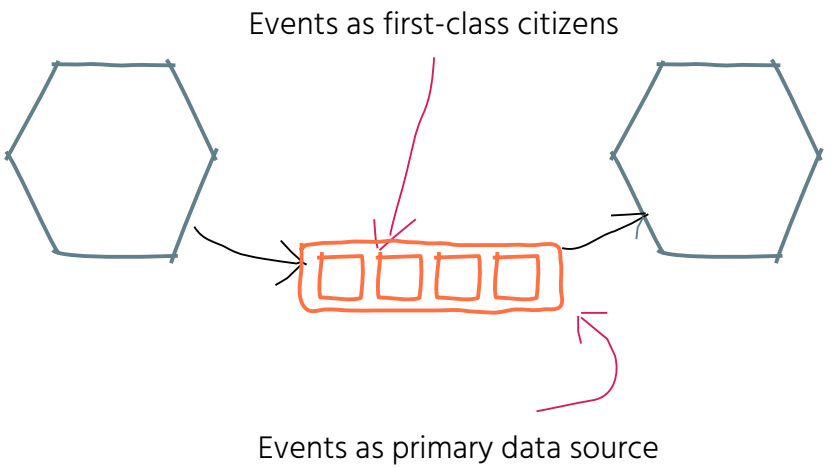
Source Of Truth

Multiple sources of truth



"Traditional" Event-Driven System

Single source of truth



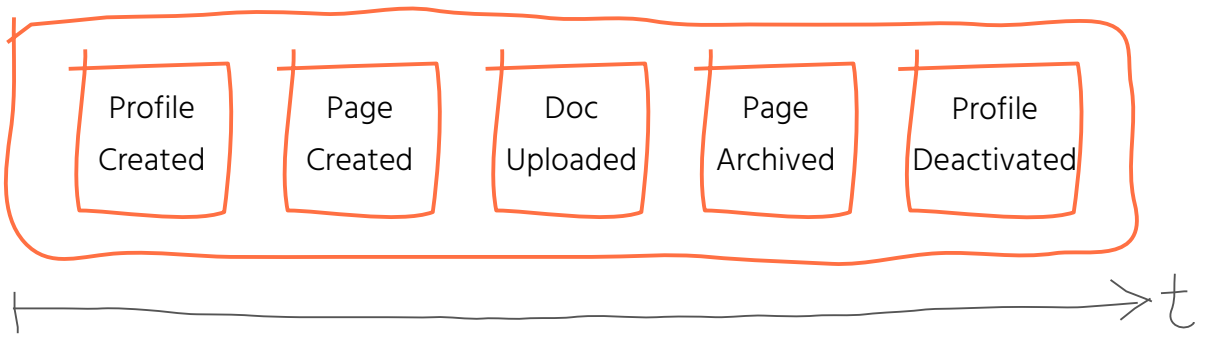
Event Log

Event Log

Immutable

Persisted

Append-Only



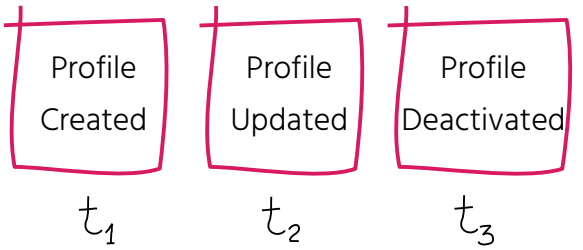
Ordered

Sequence of Events

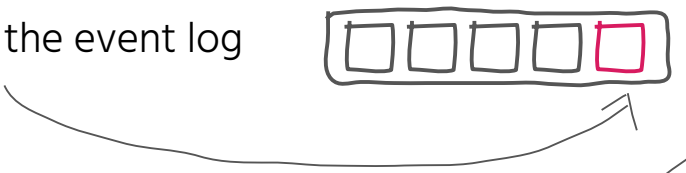
Event = A fact that has happened in the past

Event Sourcing

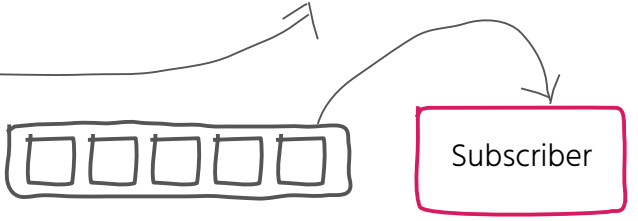
1. State changes modeled as series of events



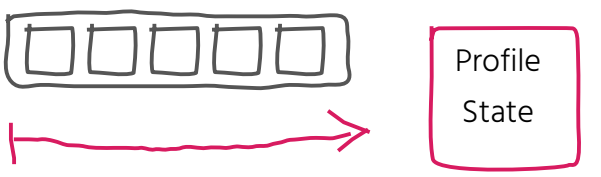
2. Events are persisted & appended to the event log



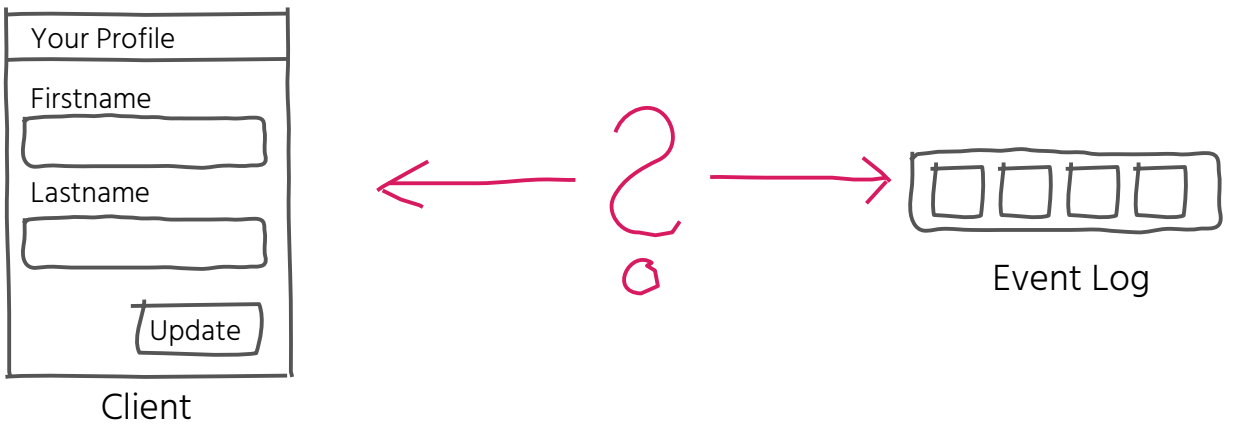
3. Services can subscribe to the event log



4. Current state is reconstructed by replaying events

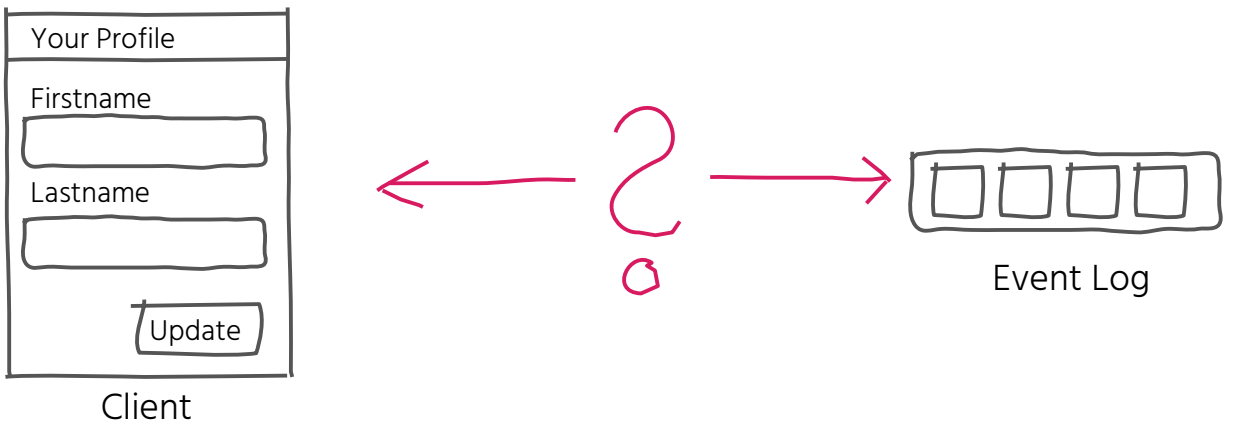


Event Sourcing



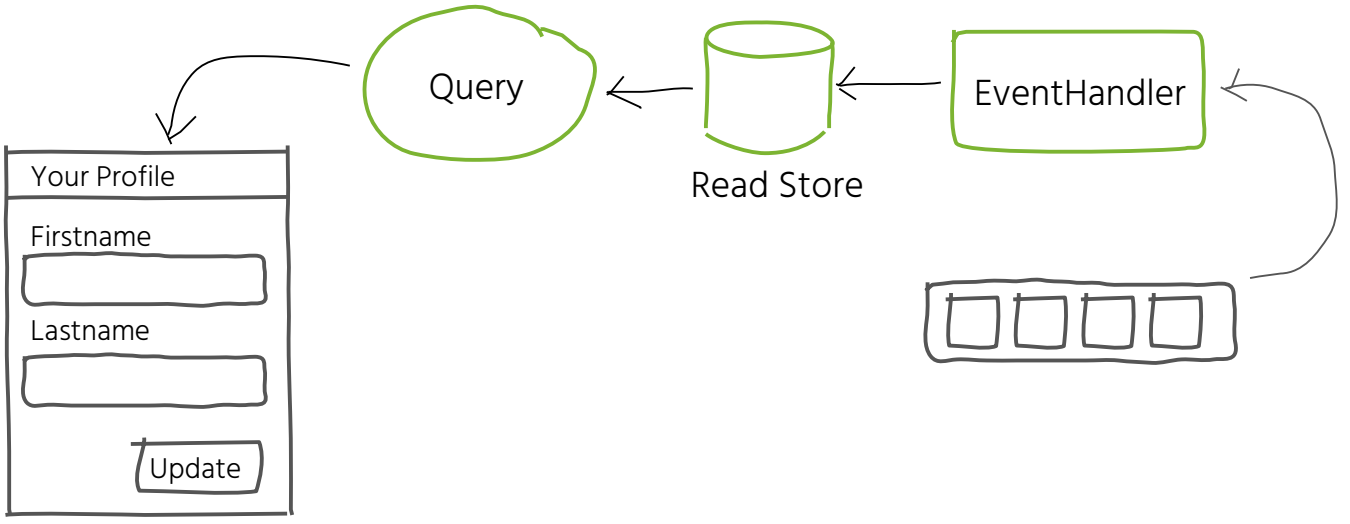
Event Sourcing

1. How to derive materialized views?



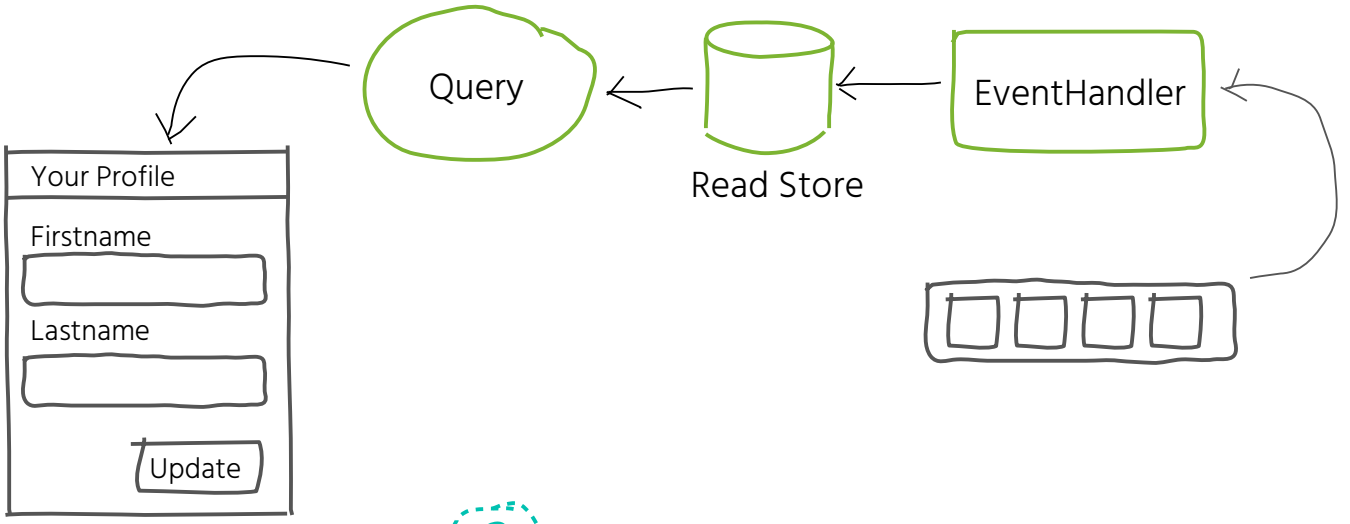
Event-Sourcing

Materialized Views



Event-Sourcing

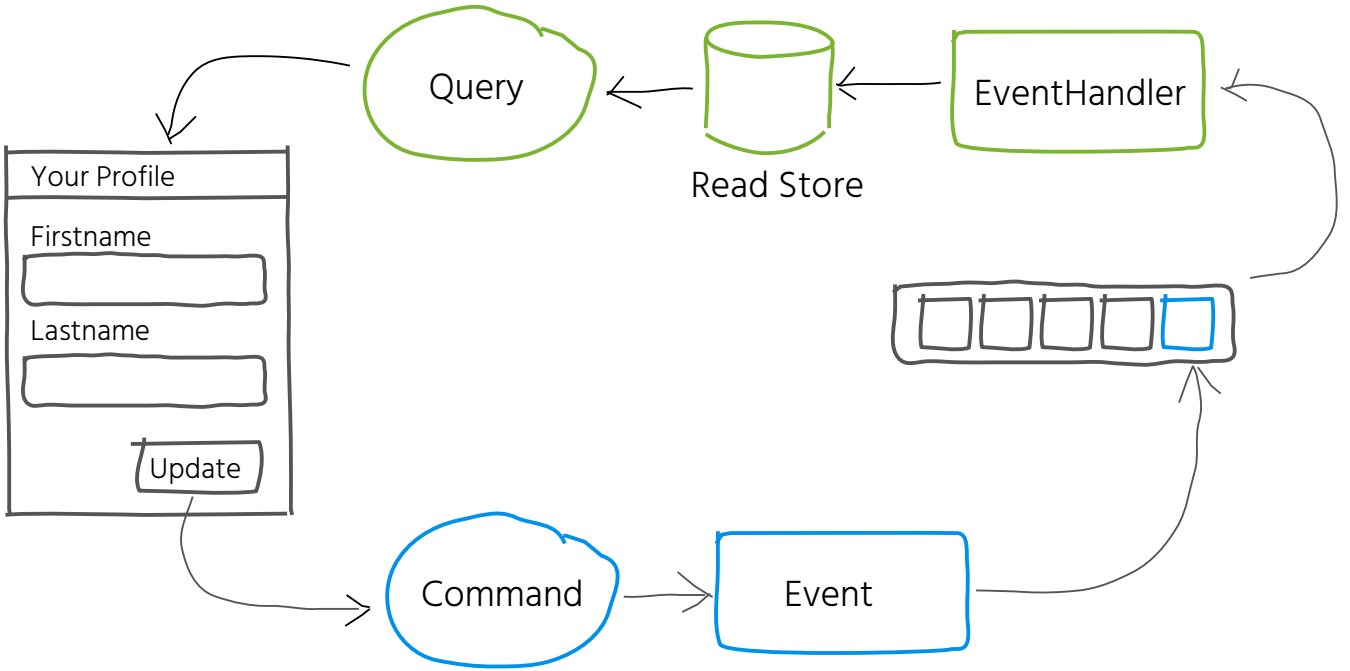
Materialized Views



2. How to update state?

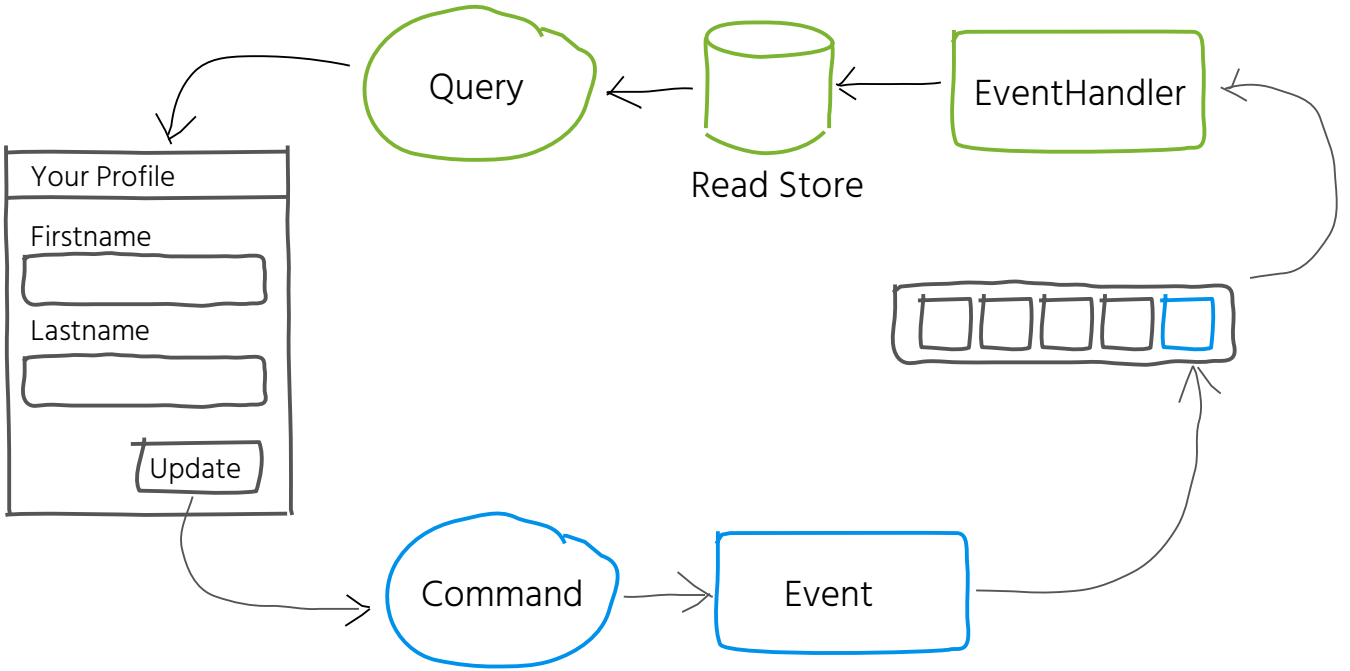
Event-Sourcing

State Changes w/ Commands & Events



Event-Sourcing

State Changes w/ Commands & Events



Read events of profile

replay events to build internal state

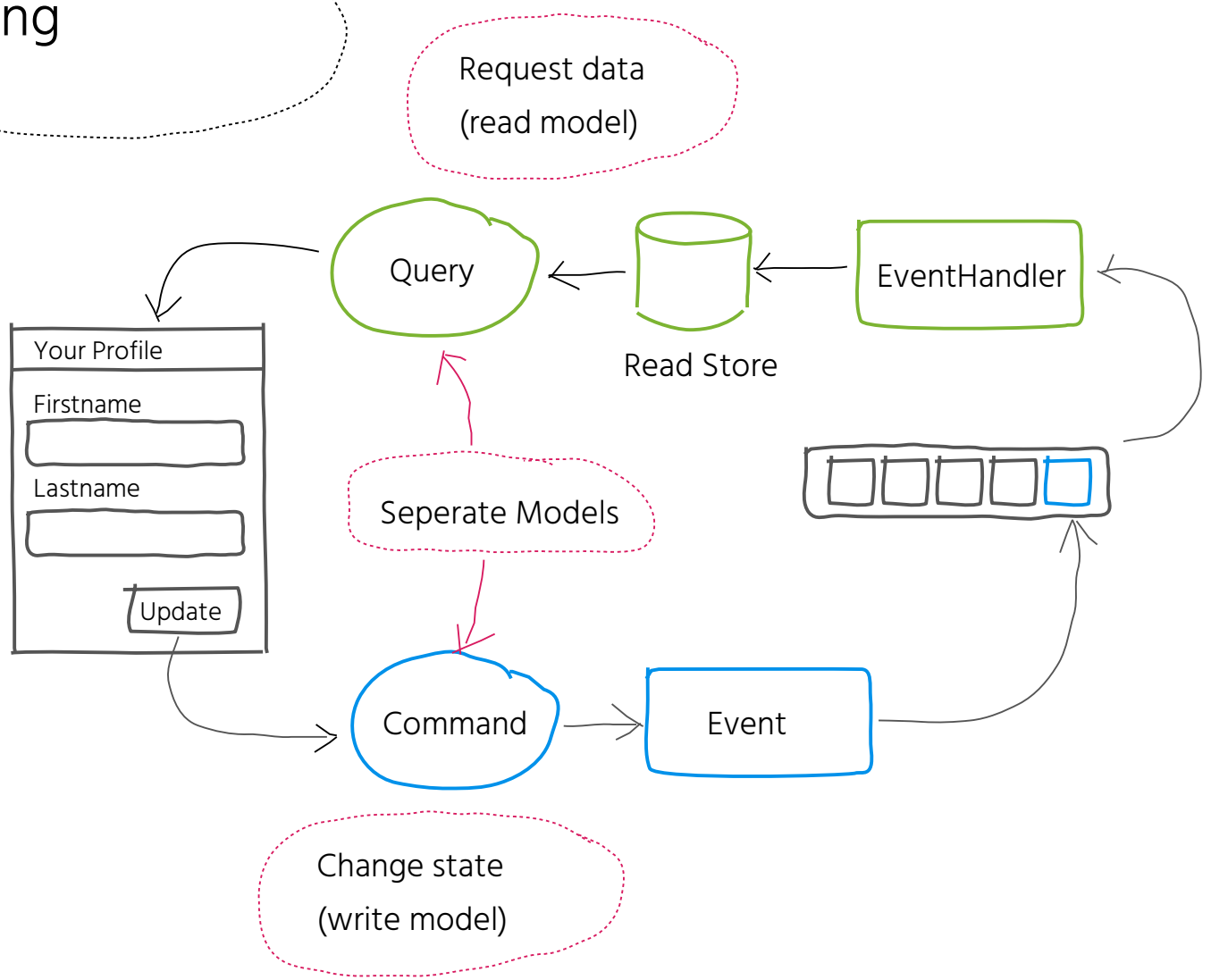
check invariants on internal state

generate event & update internal state

save event

Event-Sourcing

CQRS

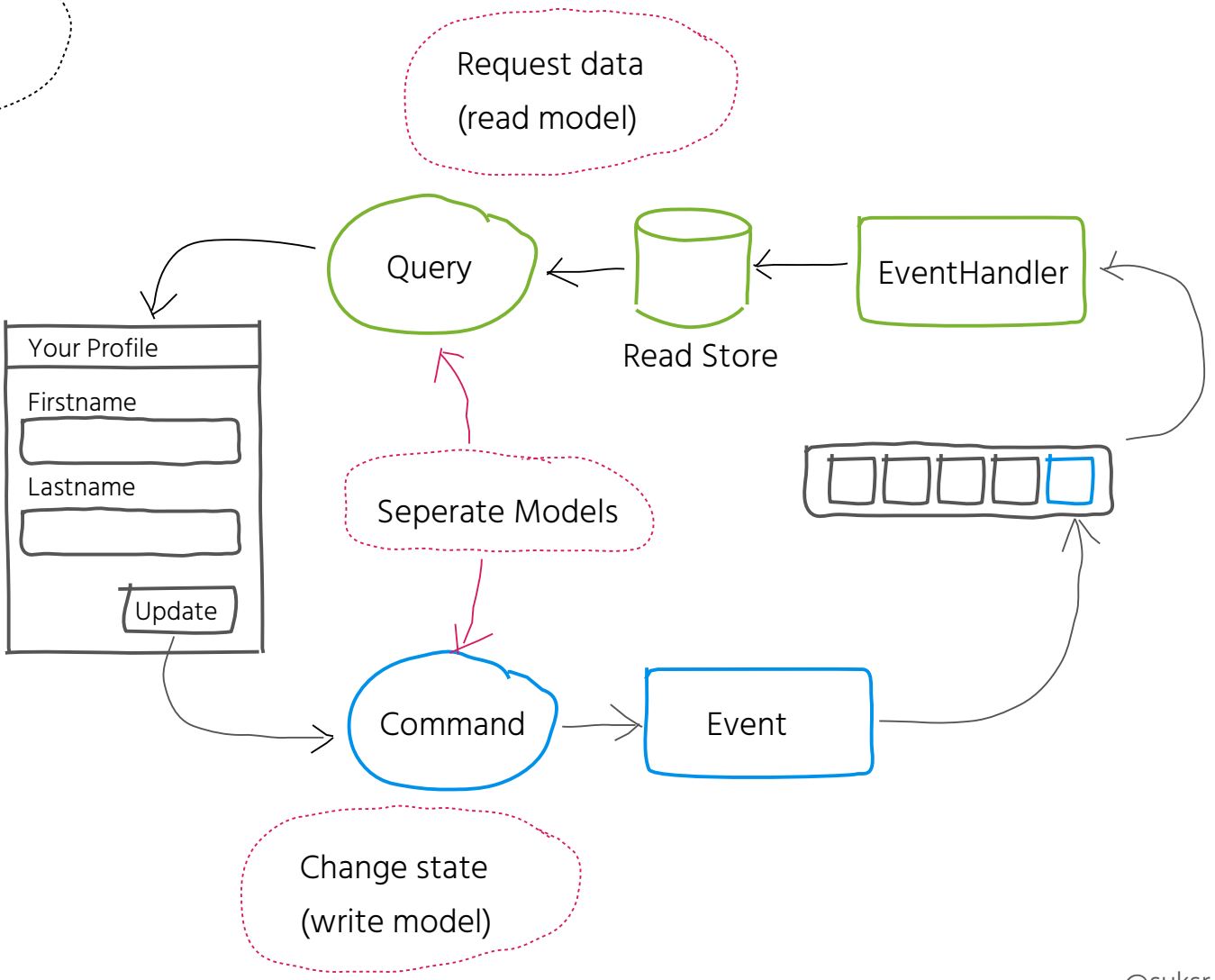


Event-Sourcing

CQRS

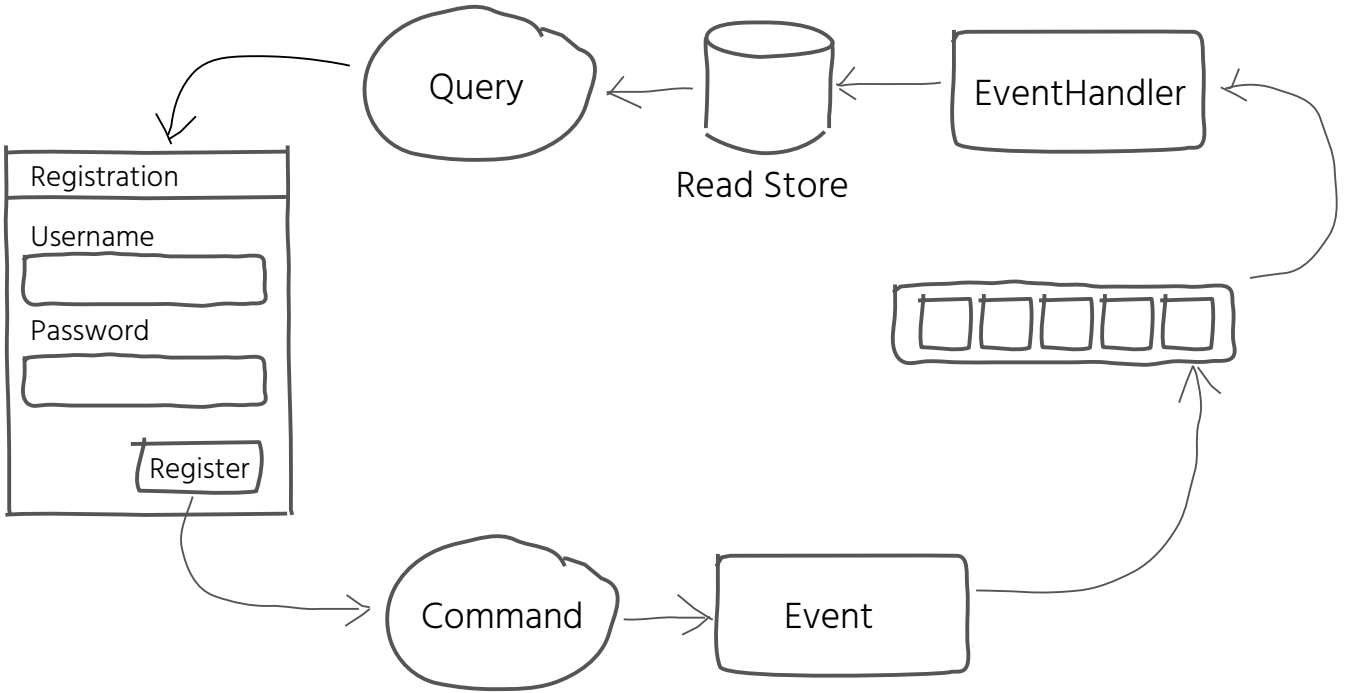
Commands & Queries

- Can be scaled independently
- Can be deployed separately
- Read model can be optimized to make queries fast & efficient
- Might involve more work due to transforming events to a read model
- Might have a higher learning curve



Event-Sourcing

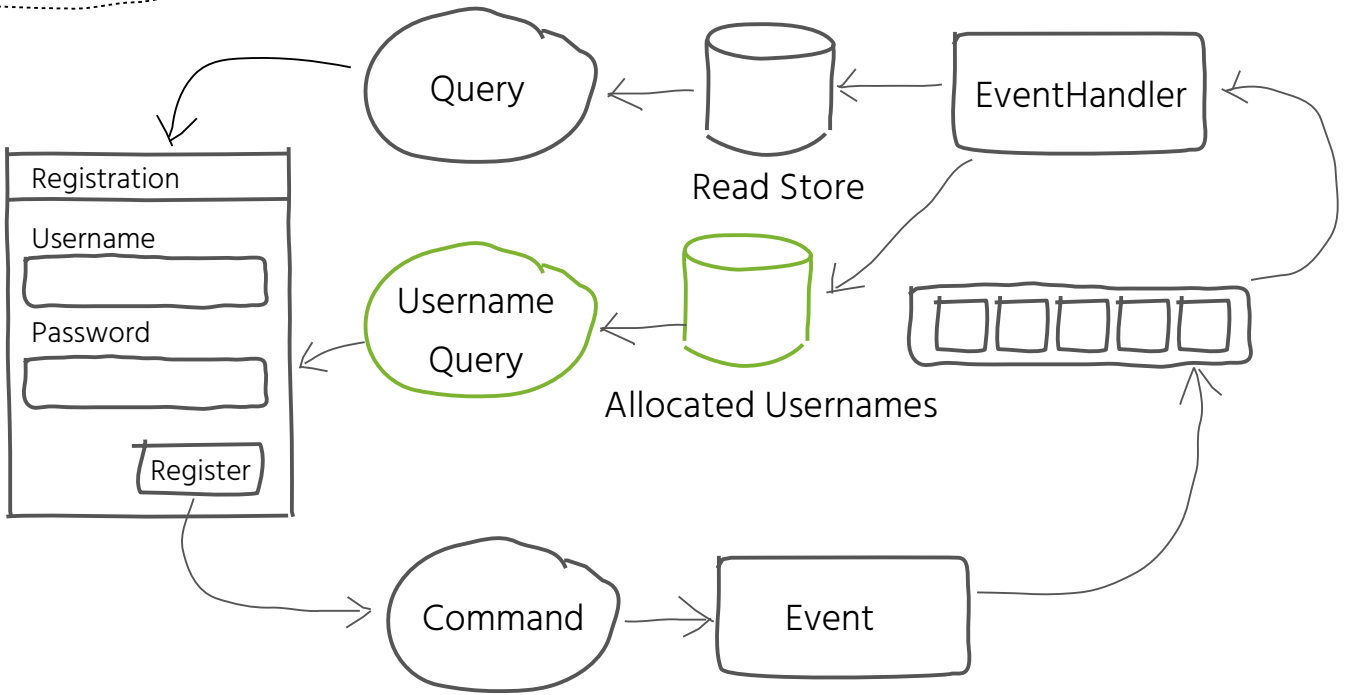
Validation



3. How to preserve business constraints among domain models, e.g. unique usernames?

Event-Sourcing

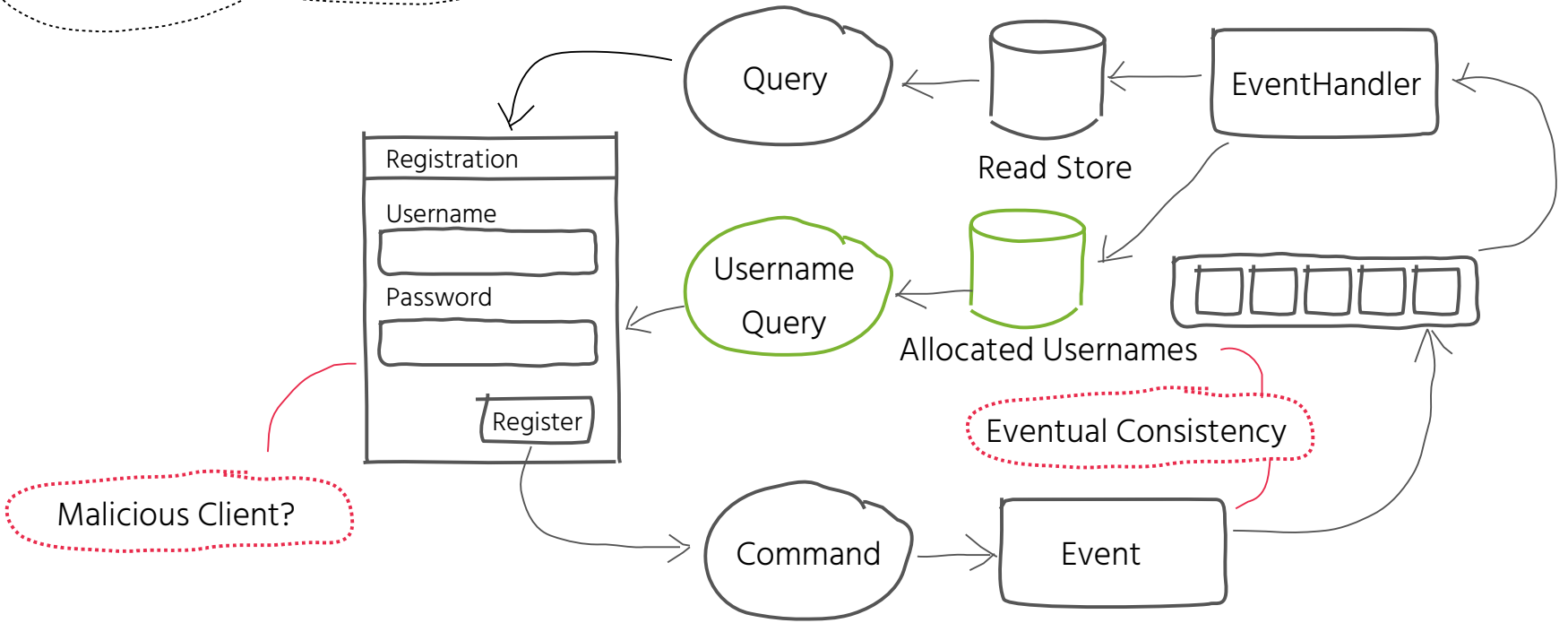
Validation



3. How to preserve business constraints among domain models, e.g. unique usernames?

Event-Sourcing

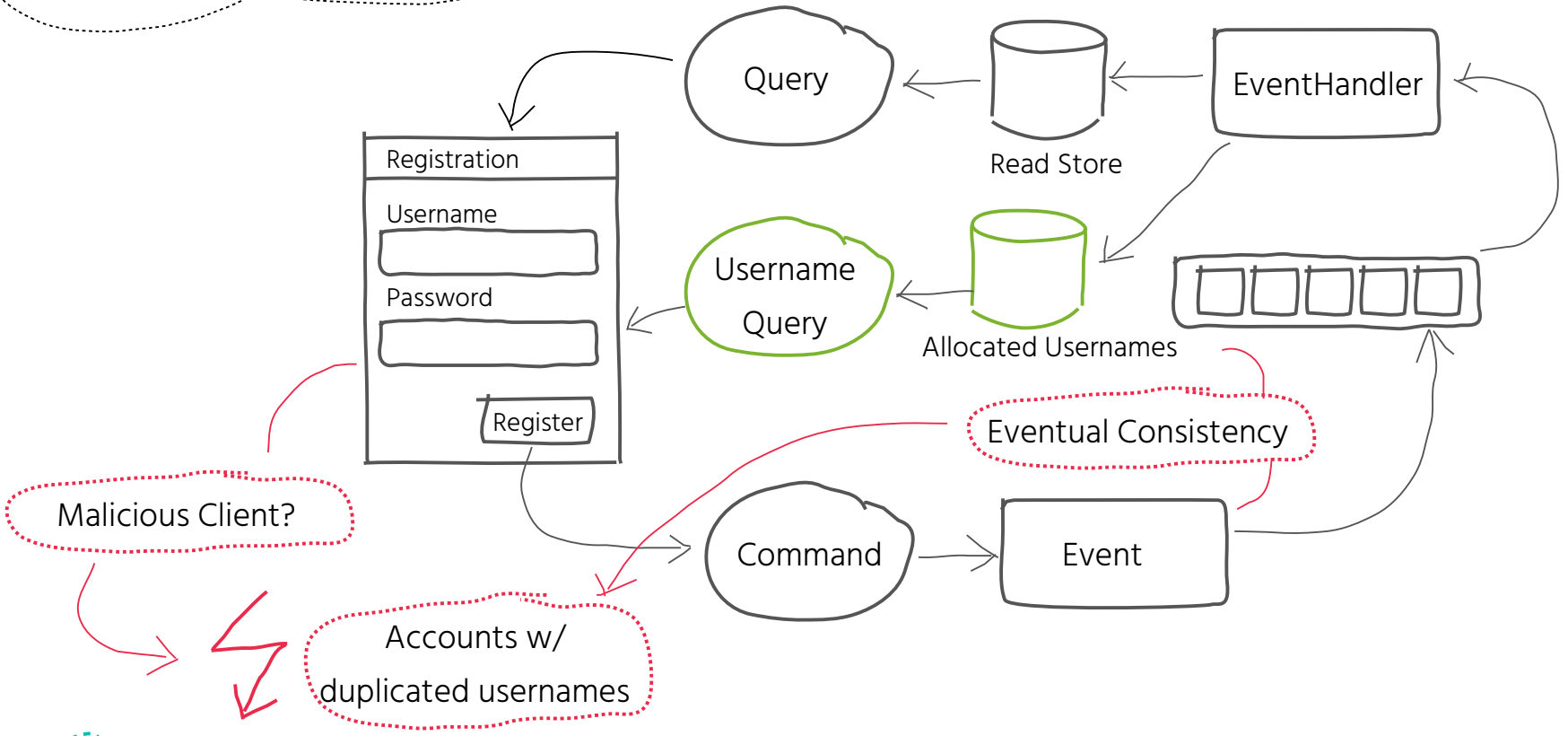
Validation: New Read Store & Client-side Query Execution



3. How to preserve business constraints among domain models, e.g. unique usernames?

Event-Sourcing

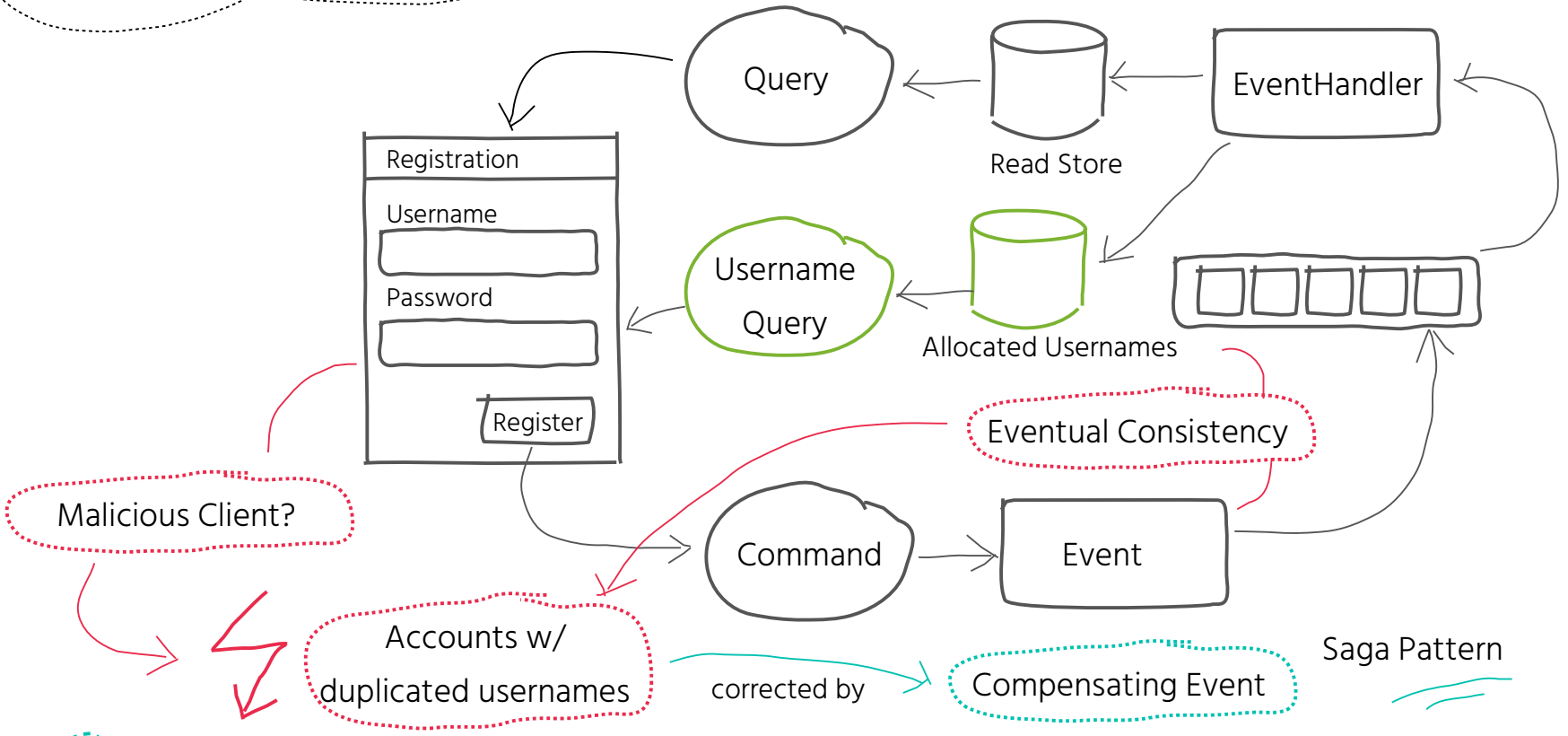
Validation: New Read Store & Client-side Query Execution



3. How to preserve business constraints among domain models, e.g. unique usernames?

Event-Sourcing

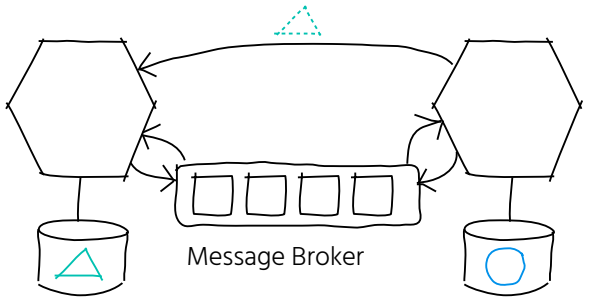
Validation: New Read Store & Client-side Query Execution + Saga Pattern



3. How to preserve business constraints among domain models, e.g. unique usernames?

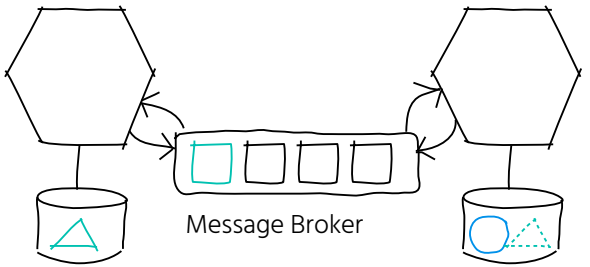
Event-Patterns

Events for notification



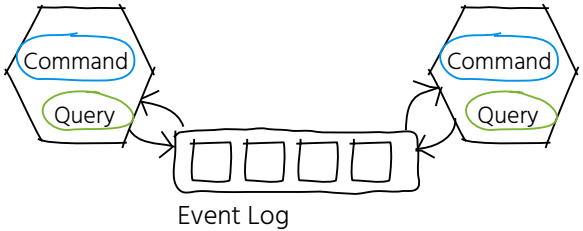
- Simple integration
- No local datasets to maintain
- Remote query => increasing coupling

Event Driven State Transfer



- Eliminating remote query by introducing local copy => better decoupling
- Local copy => better autonomy
- Duplicating effort to maintain local dataset

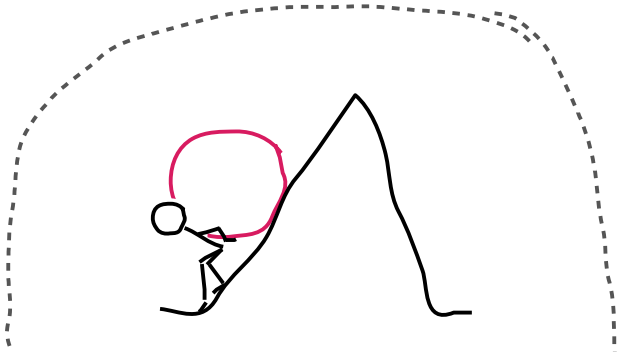
Event Sourcing w/ CQRS



- Series of events make activities in business domain explicit
- Complete log of state changes => eases troubleshooting
- Independant scaling of read & writes
- Read store can be optimized to queries
- Enables audit logging
- Might involve more work due to transforming events to a read model
- Preserving business contraints among domain objects could be tricky

2. Manageable Steps

1. Organizational Circumstances



3. Cross-Cutting Concerns

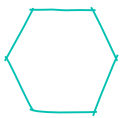
Challenges Of Microservices

6. Infrastructure & Operational Complexities

4. Distributed Monolith

5. Service-Interaction, Shared Data & Event-Patterns

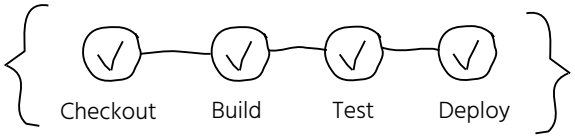
Complexities



µService

Complexities


SCM



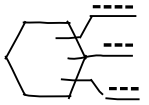
CI/CD Pipeline

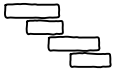

Backup


Recovery


Monitoring


Log Aggreation


Metrics


Distributed Tracing

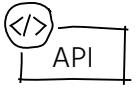

Health Checks

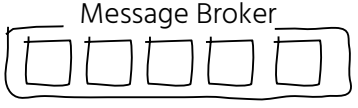

Timeout-Handling

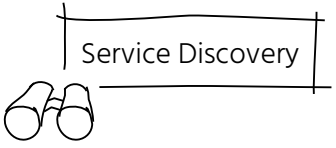

Retries

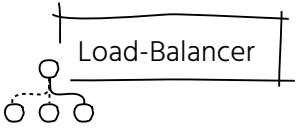
Idempotency

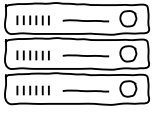

Bulkheads

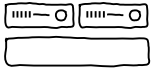


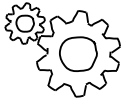


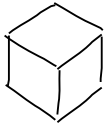





Hardware

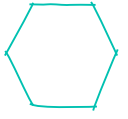

Virtualization


O/S

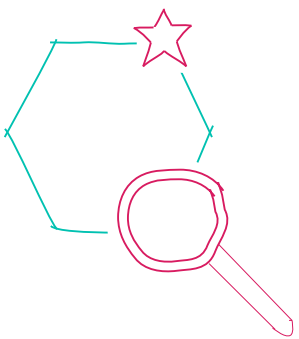

Container

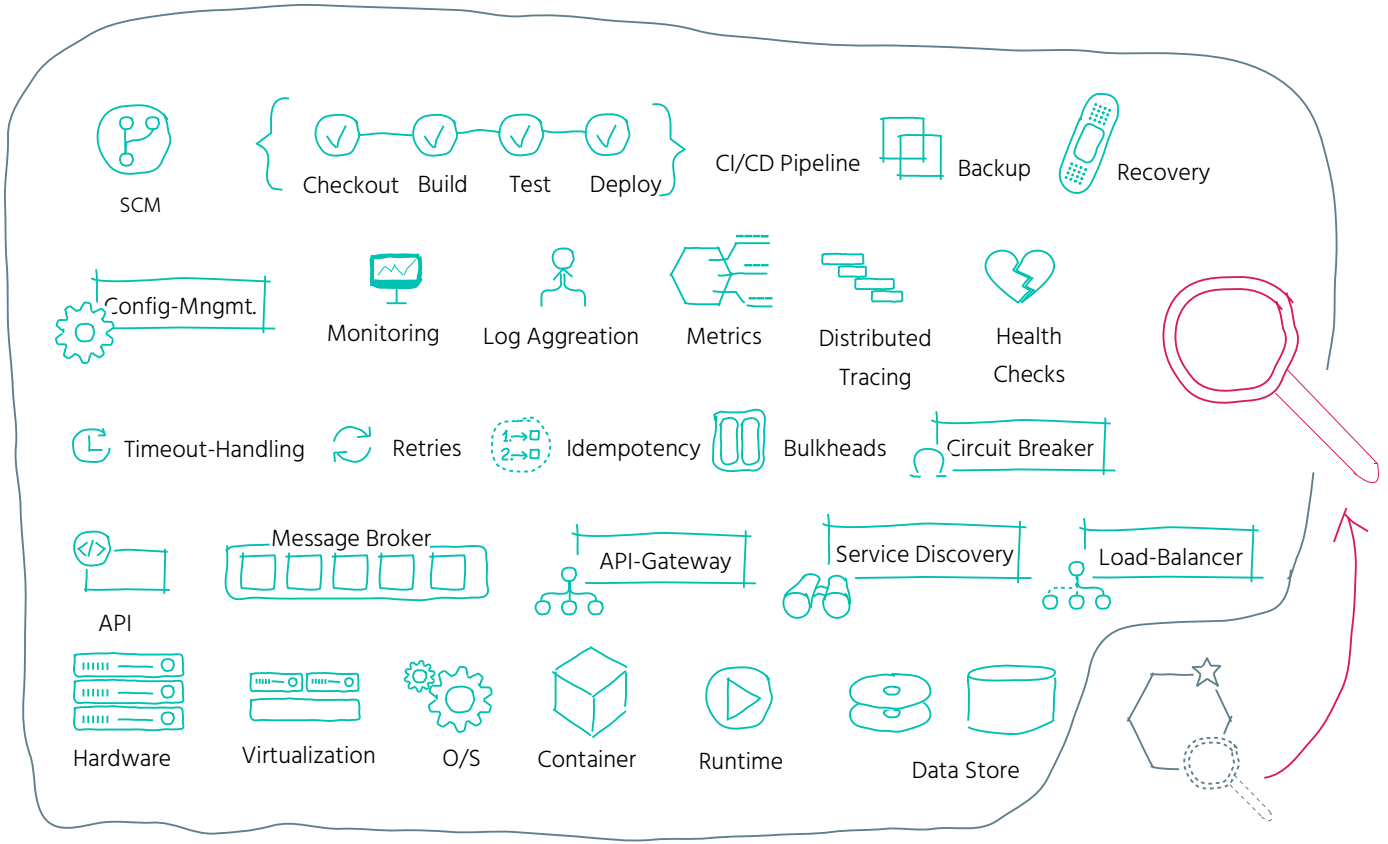

Runtime


Data Store

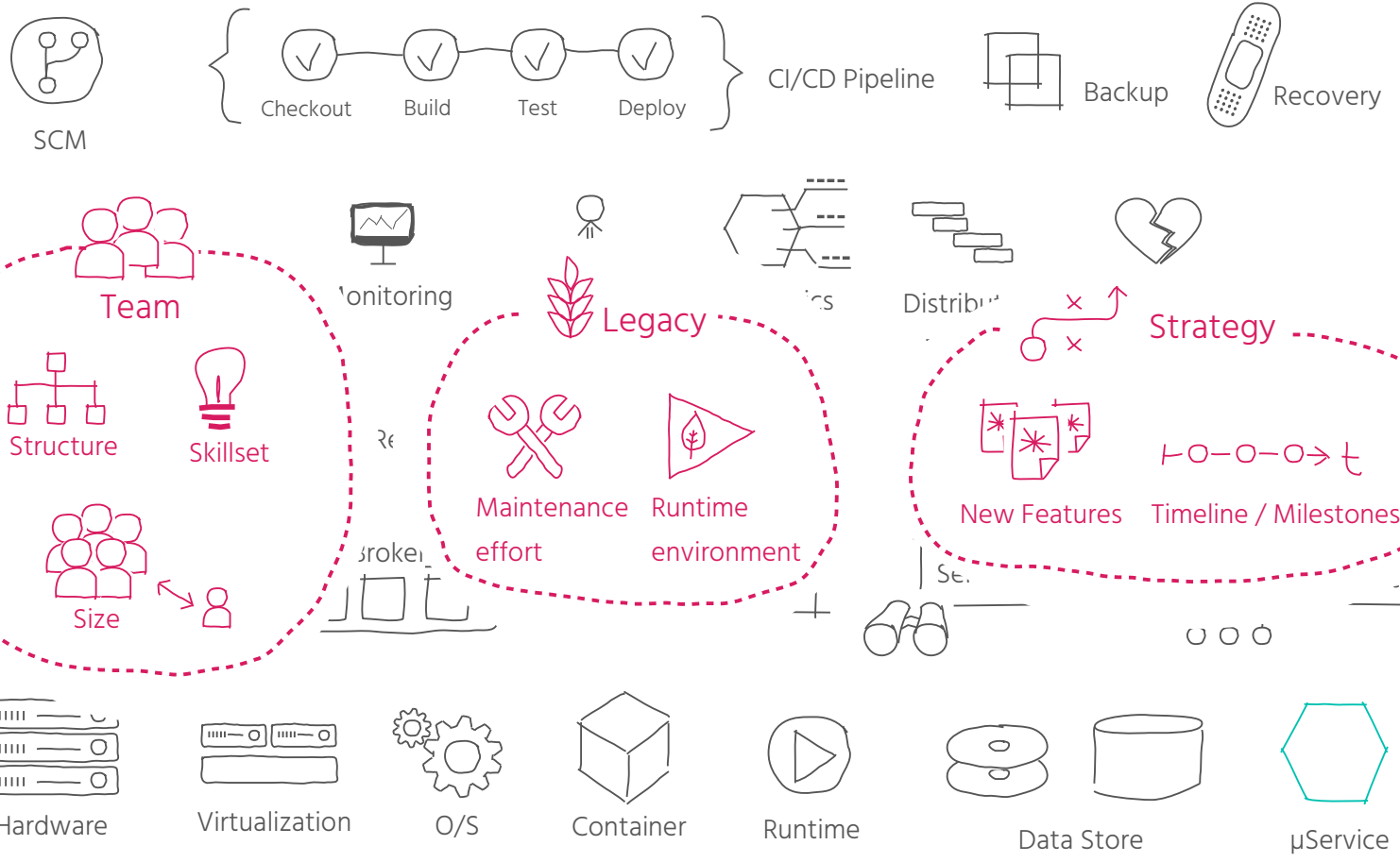

µService

Complexities

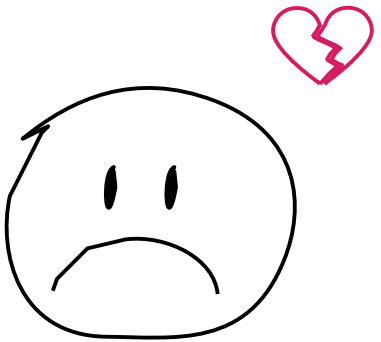




Complexities

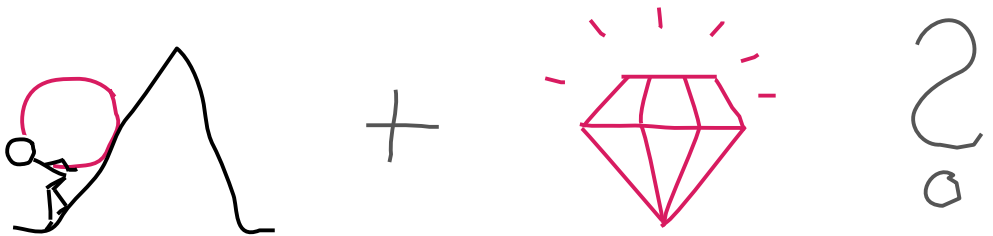


Complexities

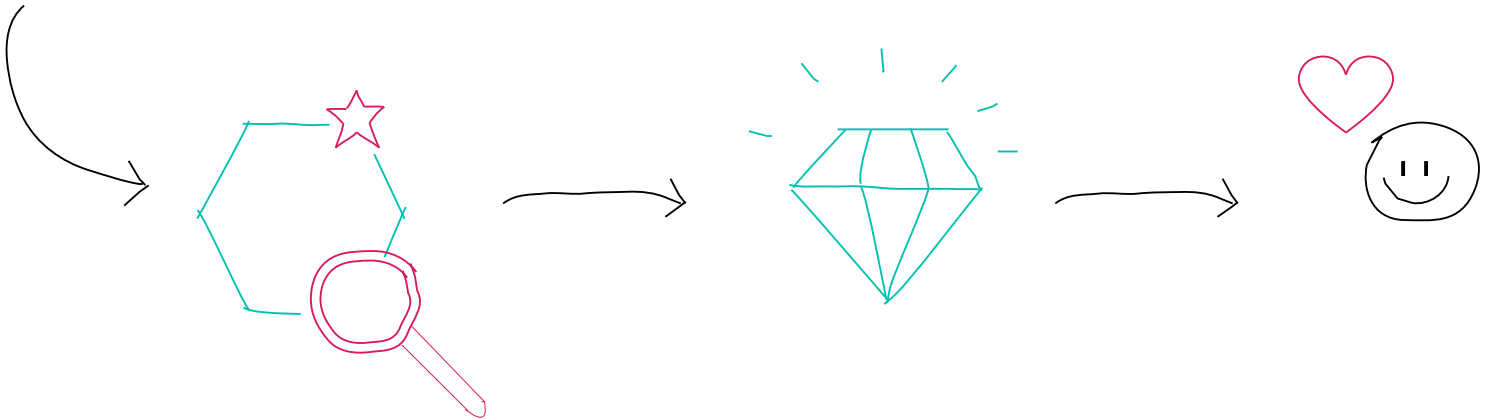


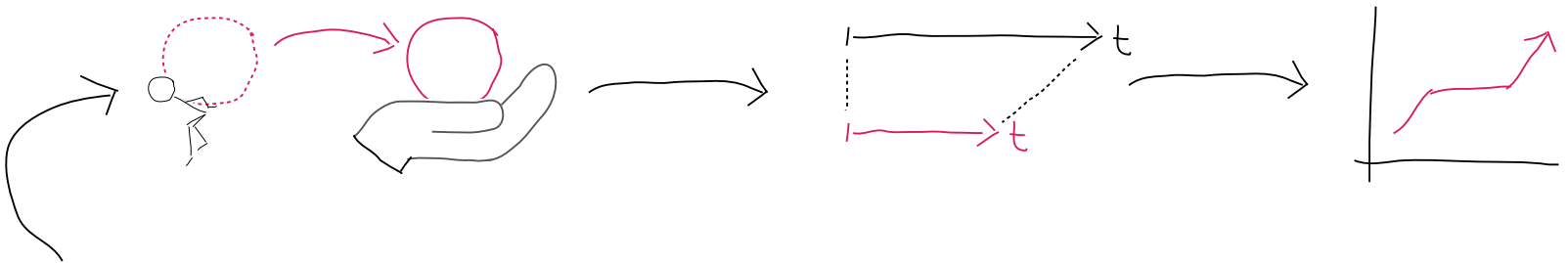
Complexities

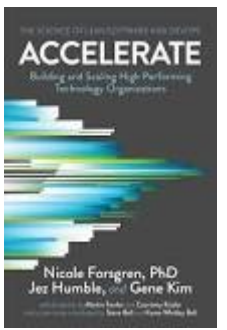
How can a small team handle infrastructure complexities and deliver business value?



Build the things that differentiate you

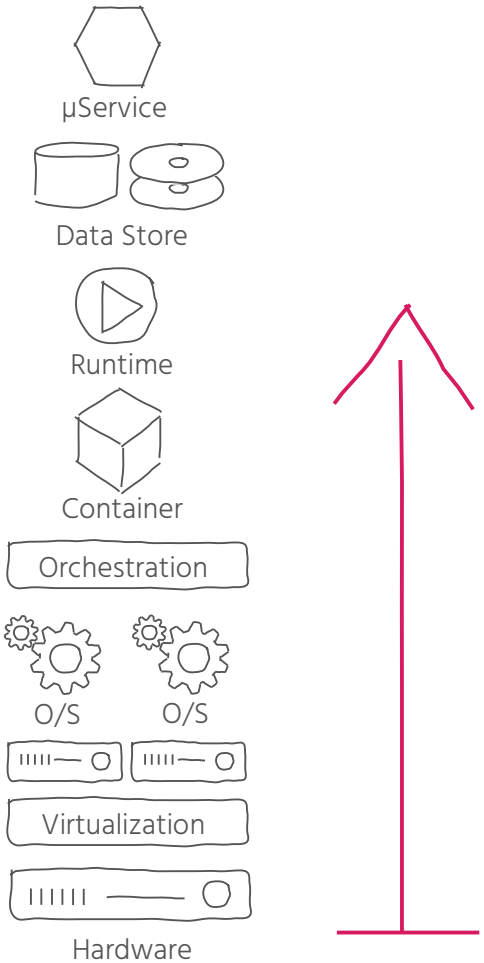






Offload the things that don't

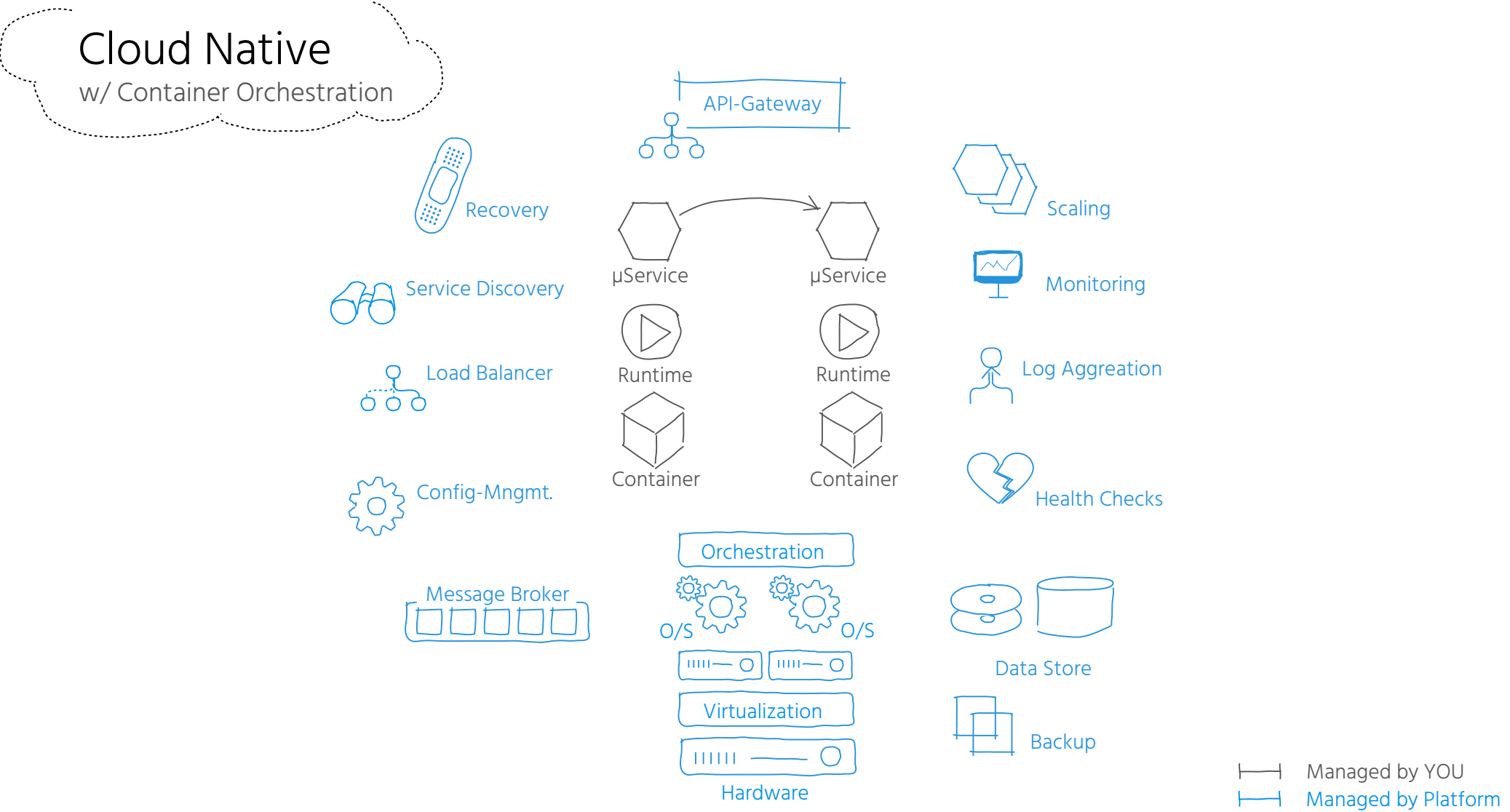
Managed Services



Offload by getting common building blocks managed

Cloud Native

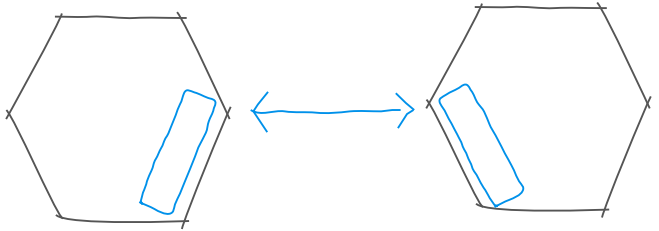
w/ Container Orchestration

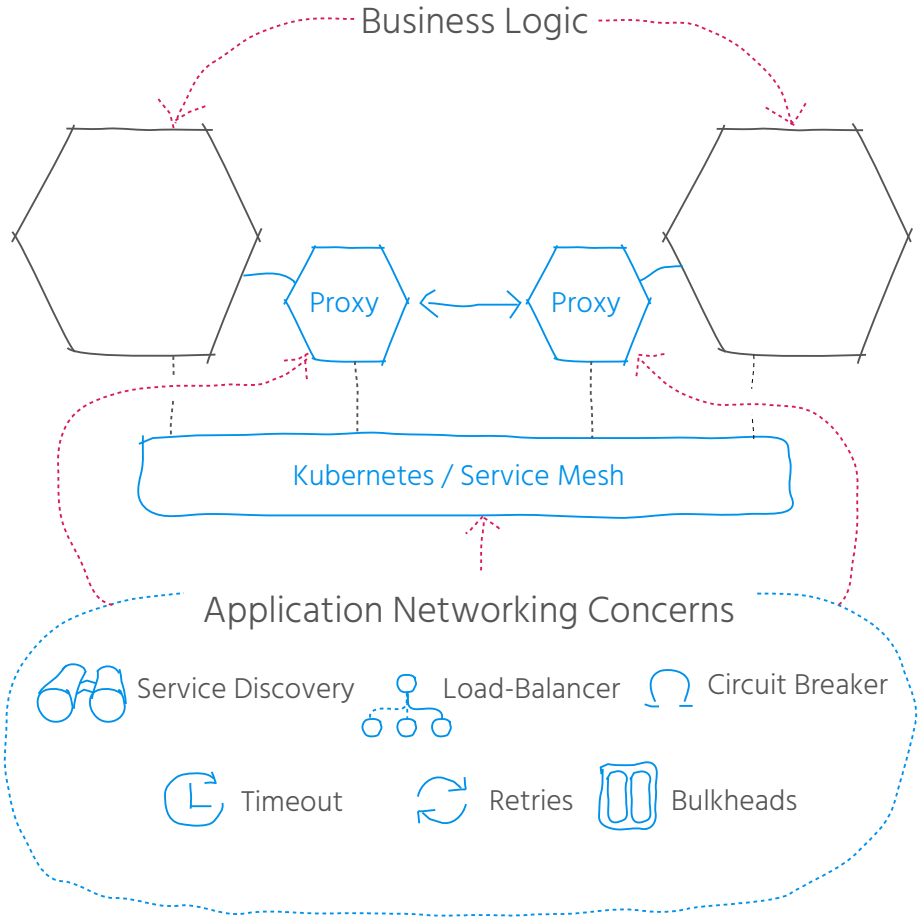


@suksr

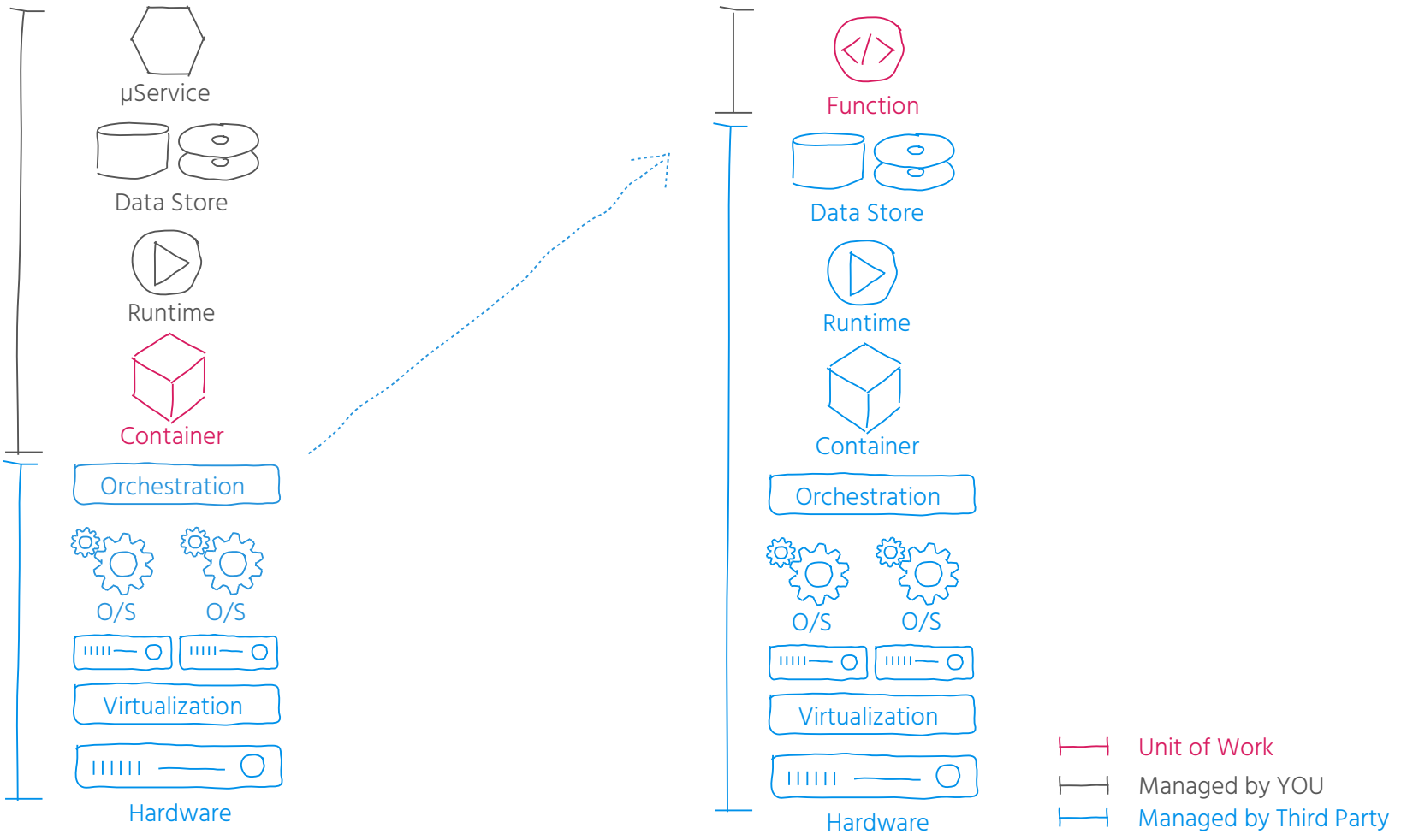
Separation Of Concerns

Service Mesh



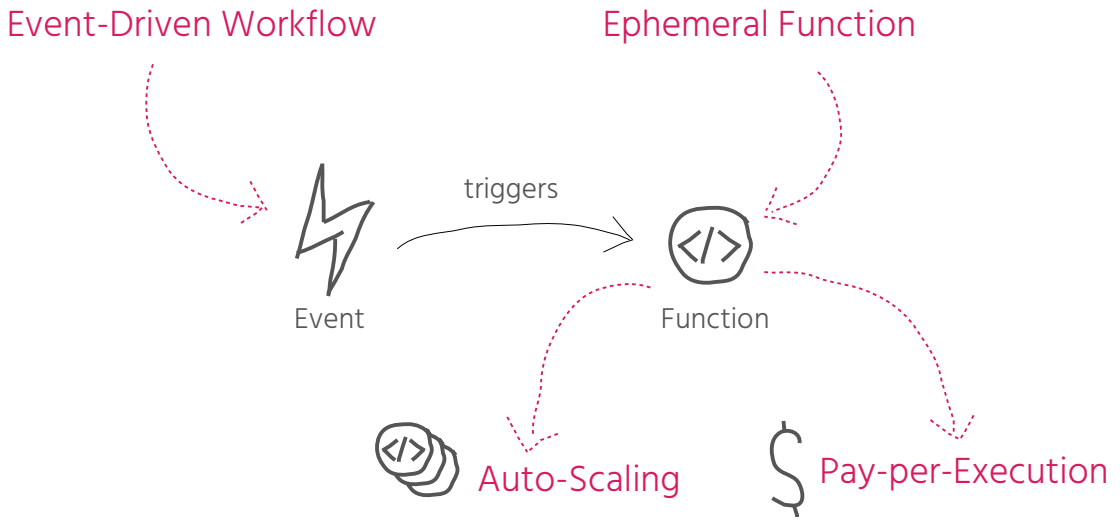


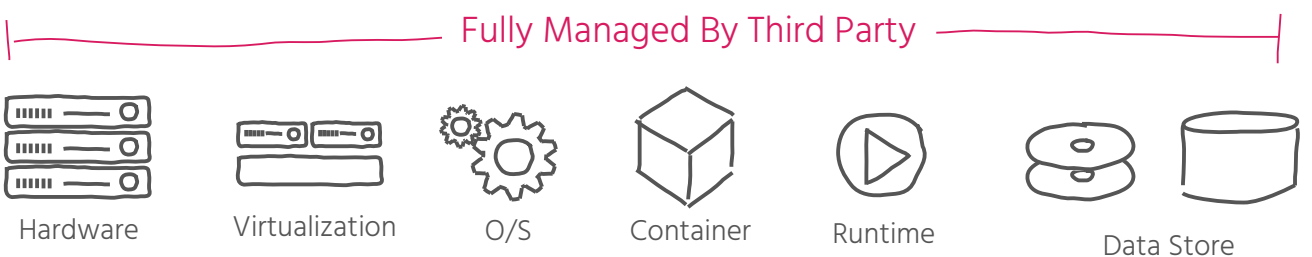
Serverless



@suksr

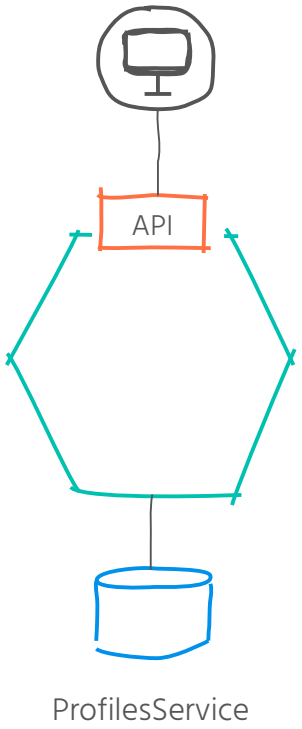
Serverless
Characteristics

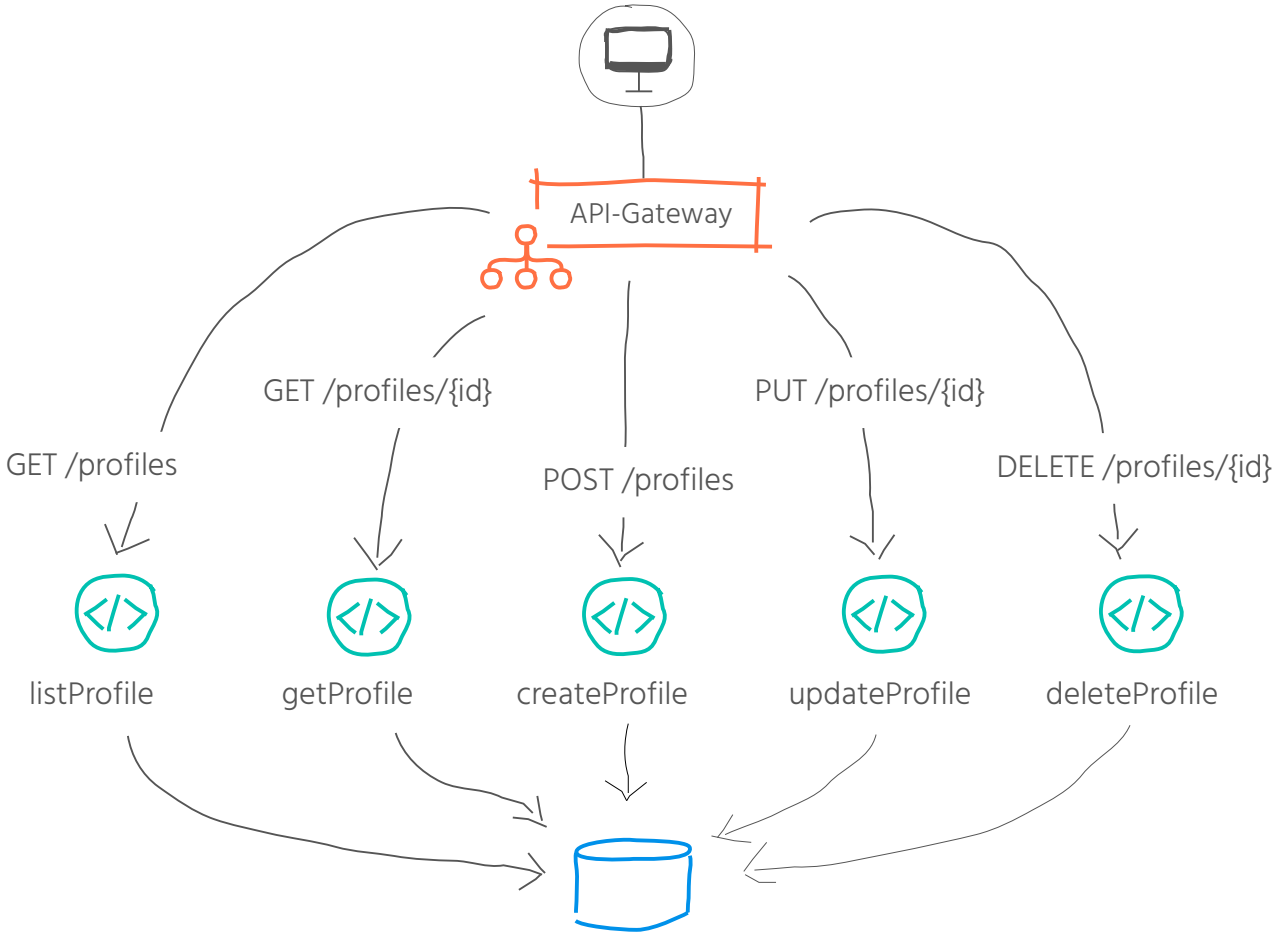




Serverless

Example Backend API





One function per endpoint and action

Serverless

Benefits

Low Maintenance

Low Cost (Total Cost of Ownership)

Easy to Scale

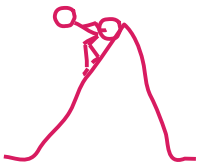
Focus on Code => Focus on Core Domain

Serverless

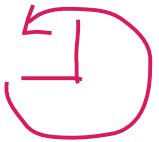
Constraints

- Limitation in programming languages and runtimes
- Latency at initial requests (cold start)
- Limits of RAM, deployment package size, number of parallel executions
- (Maximum Execution Time)
- Tooling for distributed tracing
- Vendor Lock-In

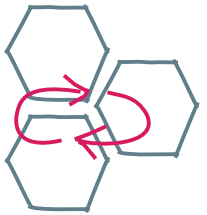
Lessons Learned



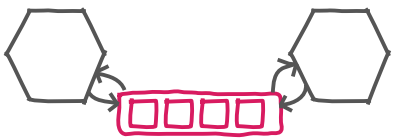
Start small



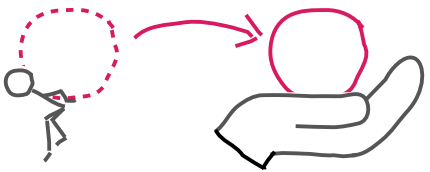
Handle cross-cutting concerns early



Avoid a distributed monolith



Design event-driven to be easy to evolve



Consider managed services to offload infrastructure complexities

Be aware of affecting circumstances
&
Distributed Systems are Complex :)

THANK YOU

Susanne Kaiser

Independent Tech Consultant

@suksr