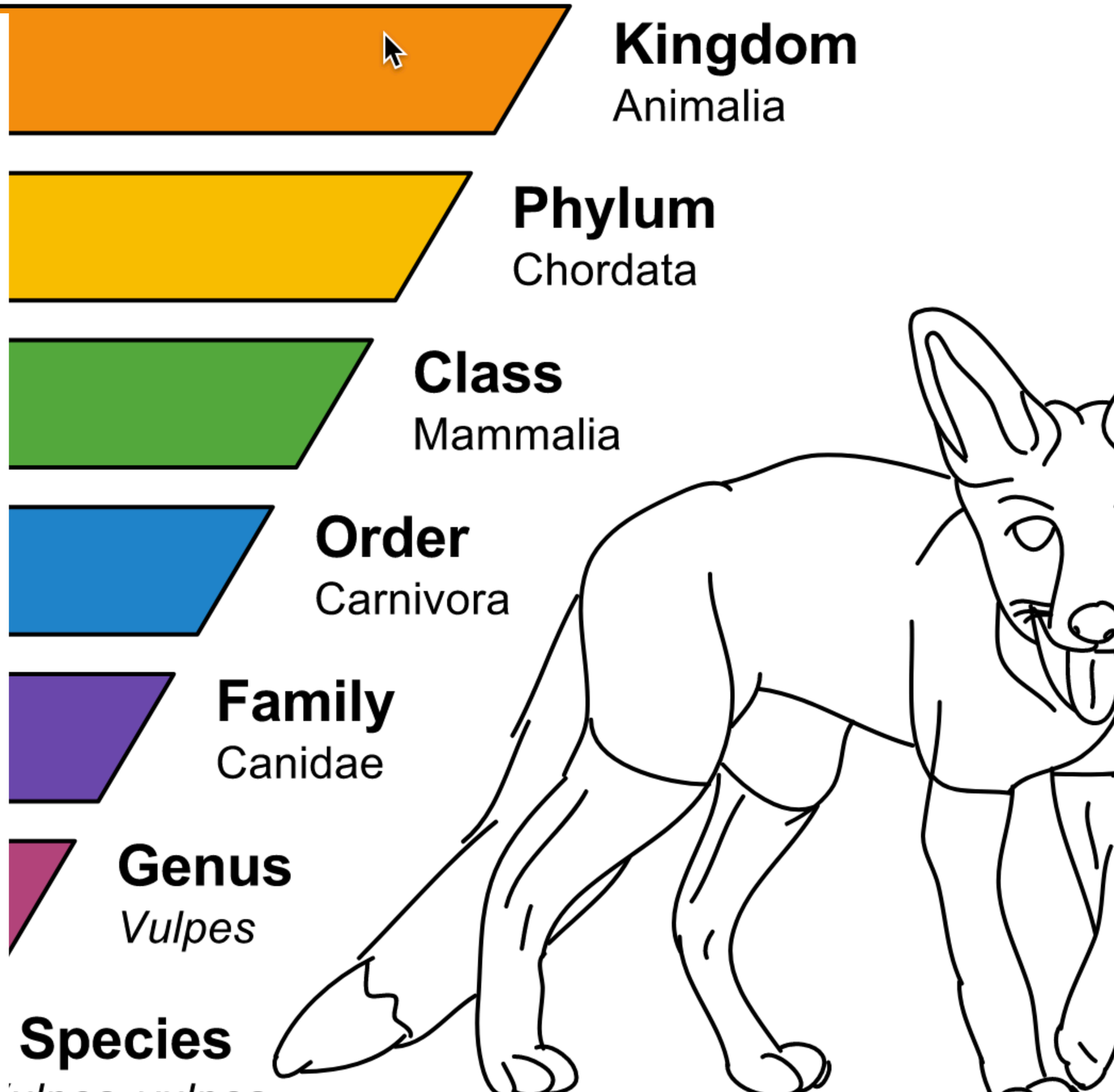
Microservices 2019

# Microservices: A Taxonomy

Stefan Tilkov
stefan.tilkov@innoq.com
@stilkov

INNOQ

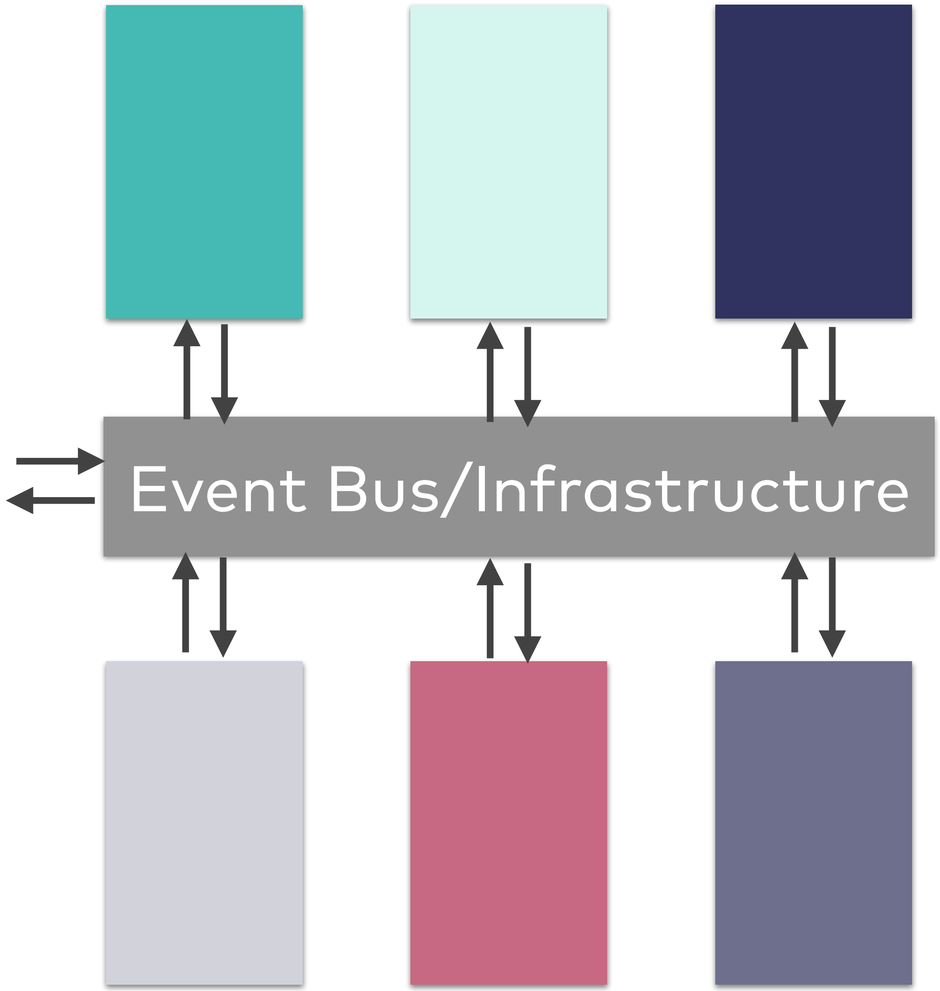


# Microservices – Common Traits

- Focused on “one thing”
- Autonomous operation
- Isolated development
- Independent deployment
- Localized decisions

# Example: Device Event Handling

- Incoming event validation
- Format transformation
- Fan-out event generation
- Aggregation
- Storage




→FaaS

# Pattern: FaaS (Function as a Service)

## Description:

- As small as possible
- A few hundred lines of code or less
- Triggered by events
- Communicating asynchronously

## As seen on:

- Any recent Fred George talk
- Serverless Architecture
- AWS Lambda

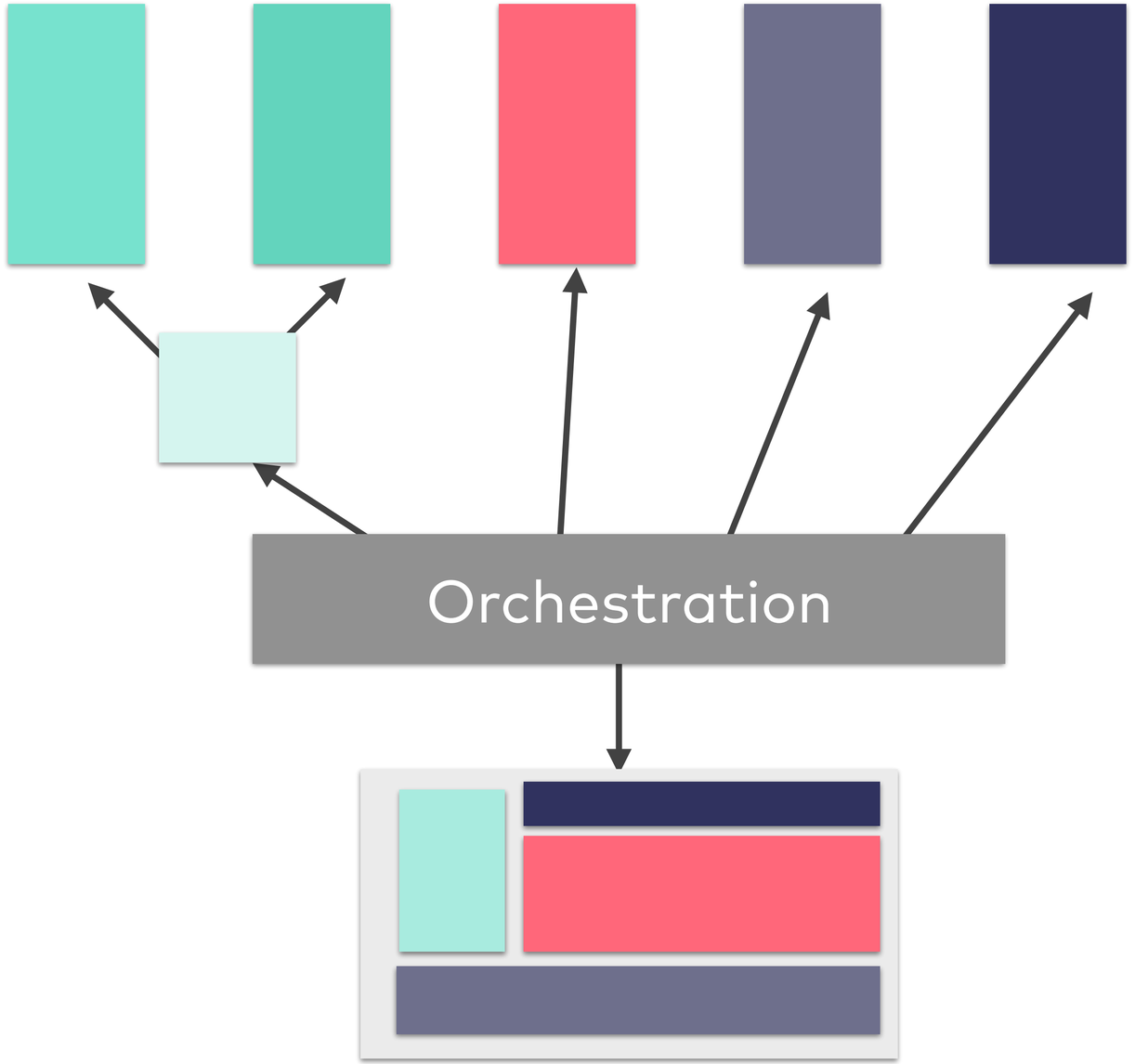
# Pattern: FaaS (Function as a Service)

## Consequences:

- Shared strong infrastructure dependency
- Common interfaces, multiple invocations
- Application logic in event handler configuration
- Emerging behavior (a.k.a. “what the hell just happened?”)
- (Possibly) billed per request
- (Possibly) unpredictable response times

# Example: Product Detail Page

- Core product data
- Prose description
- Images
- Reviews
- Related content




→µSOA

# Pattern: μSOA (Microservice-SOA)

## Description:

- Small, self-hosted
- Communicating synchronously
- Cascaded/streaming
- Containerized

## As seen on:

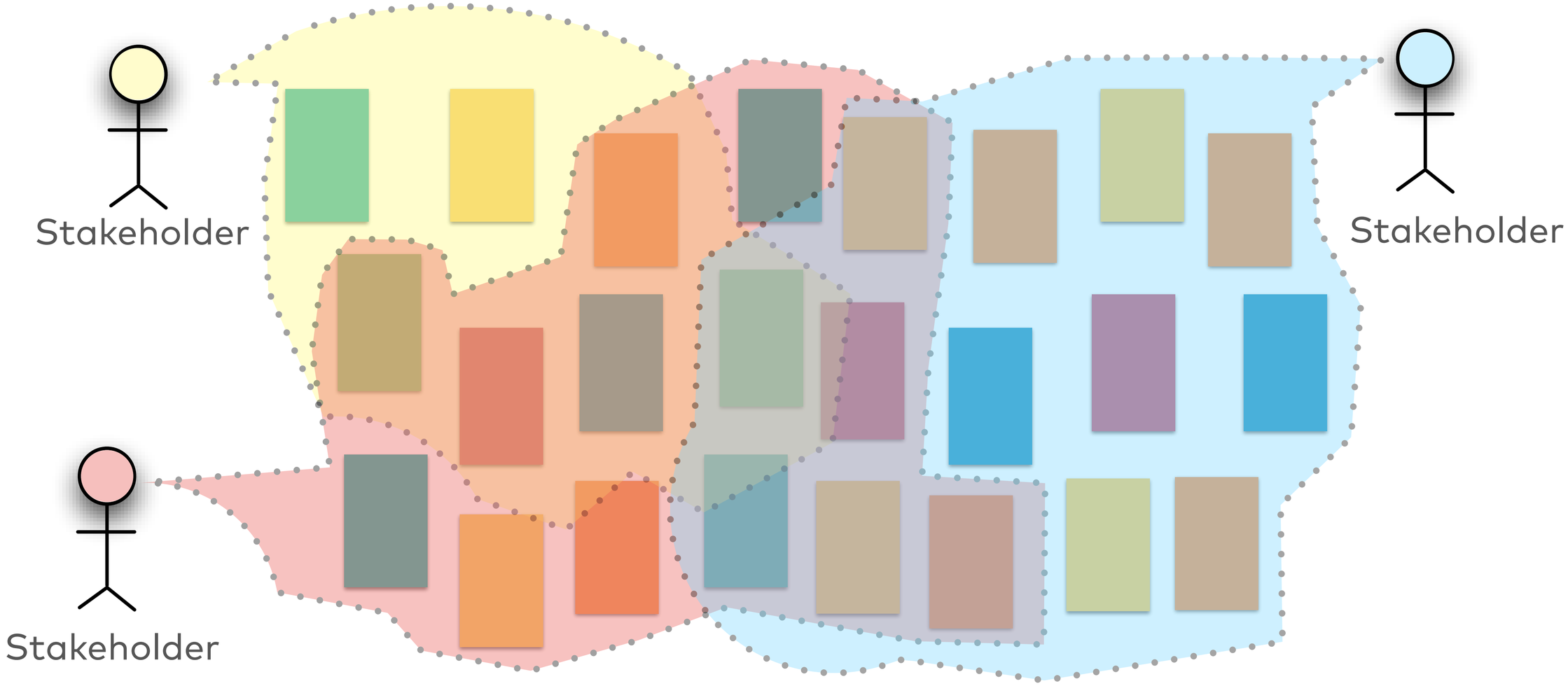
- Netflix
- Twitter
- Gilt

# Pattern: µSOA (Microservice-SOA)

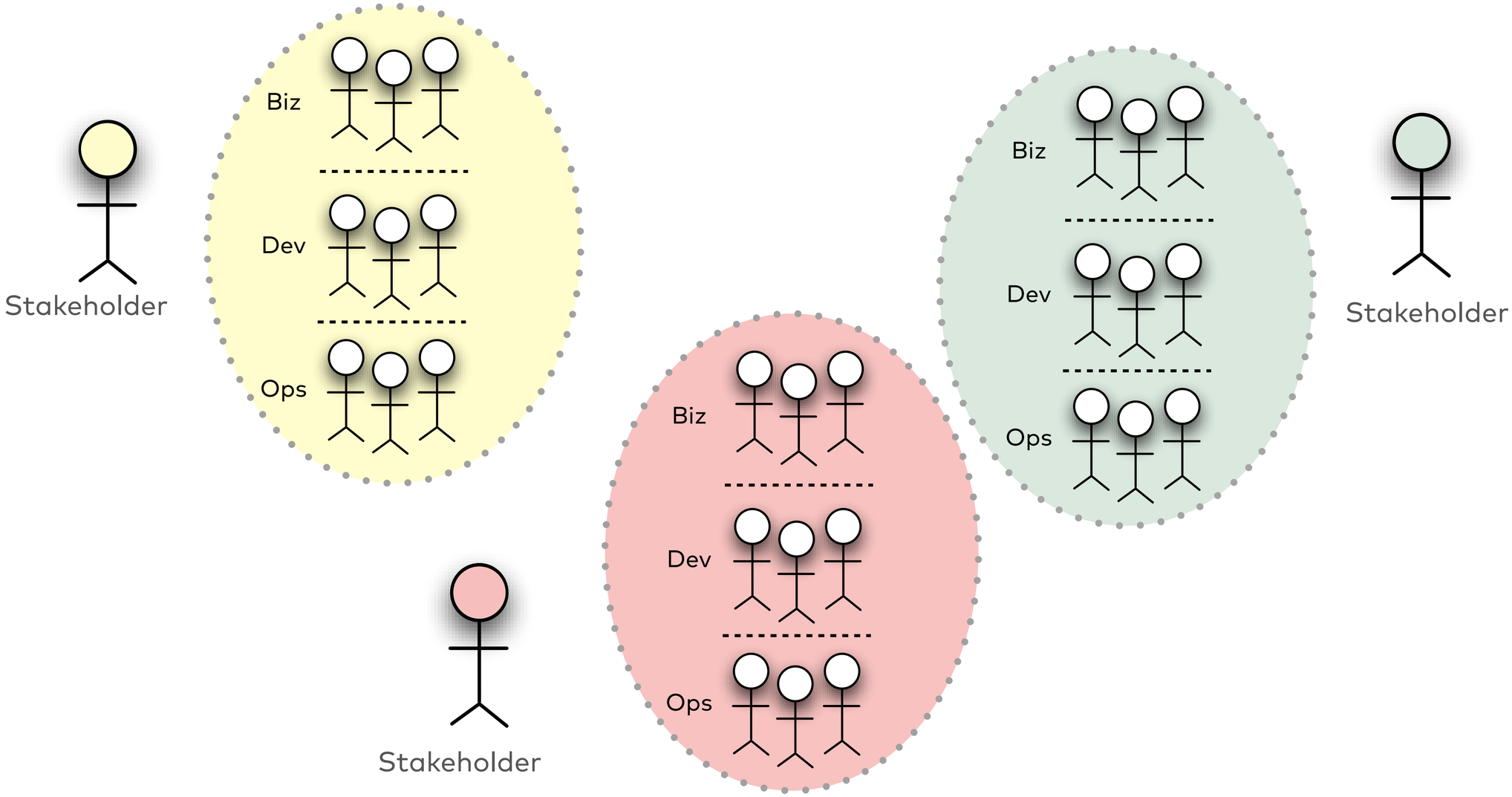
## Consequences:

- Close collaboration – common goal
- Need for resilience/stability patterns for invocations
- High cost of coordination (versioning, compatibility, ...)
- High infrastructure demand
- Often combined with parallel/streaming approach
- Well suited to environments with extreme scalability requirements

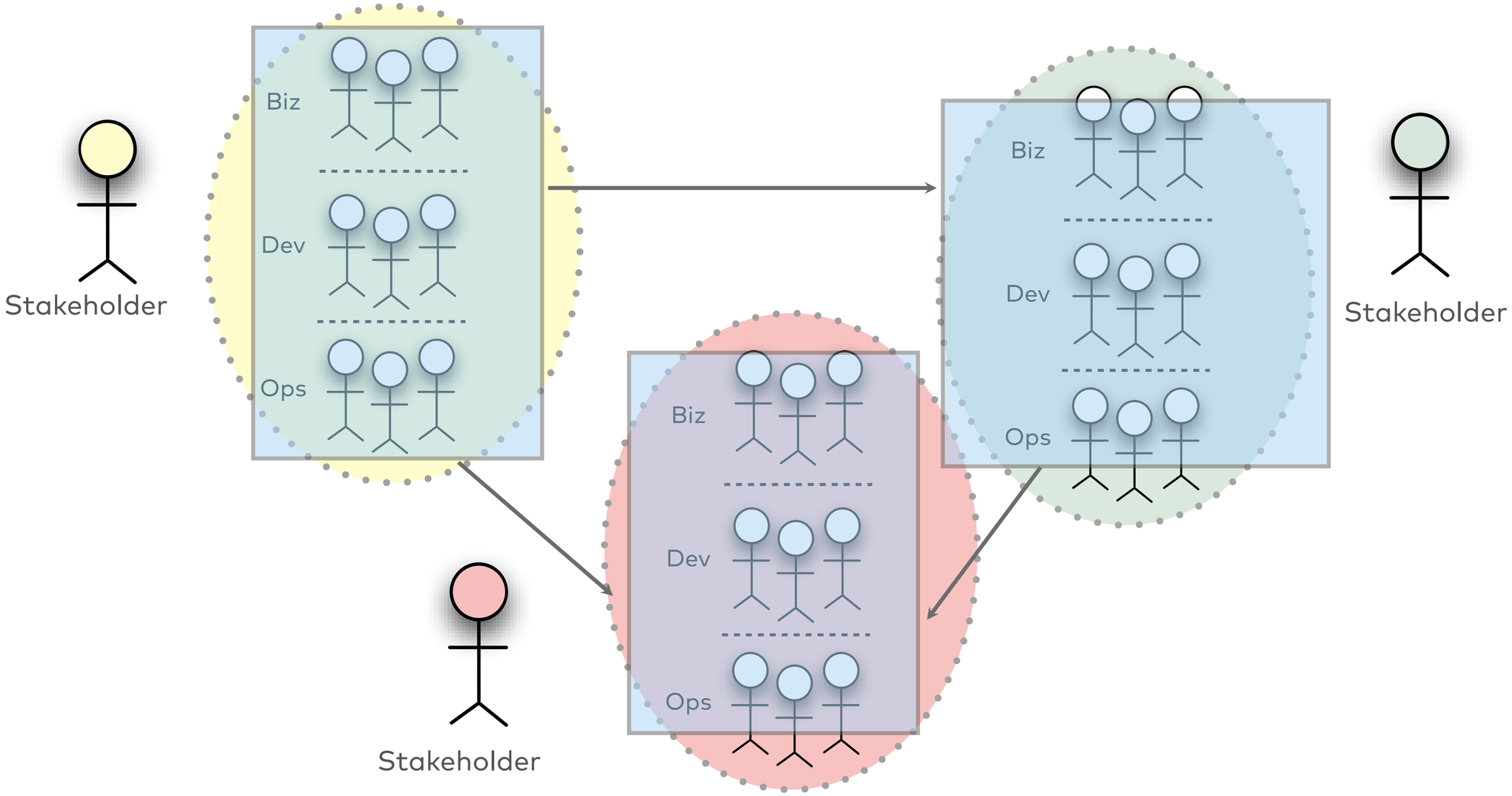
# Antipattern: Decoupling Illusion



# Pattern: Autonomous Cells



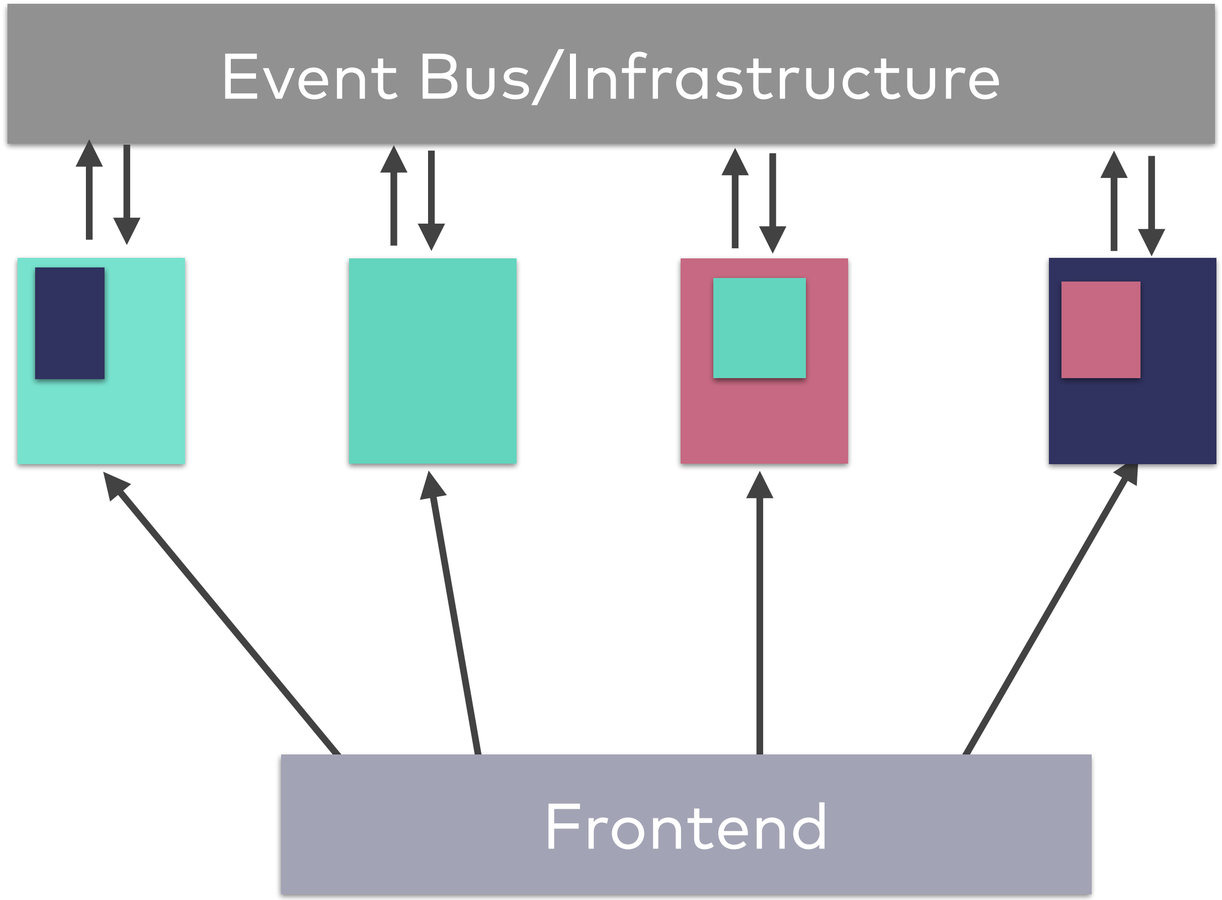
# Pattern: Autonomous Cells



# Example: Logistics Application

- Order management
- Shipping
- Route planning
- Invoicing

→DDDD



# Pattern: DDDD (Distributed Domain-driven Design)

## Description:

- Small, self-hosted
- Bounded contexts
- Redundant data/CQRS
- Business events
- Containerized

## As seen on:

- (undisclosed)

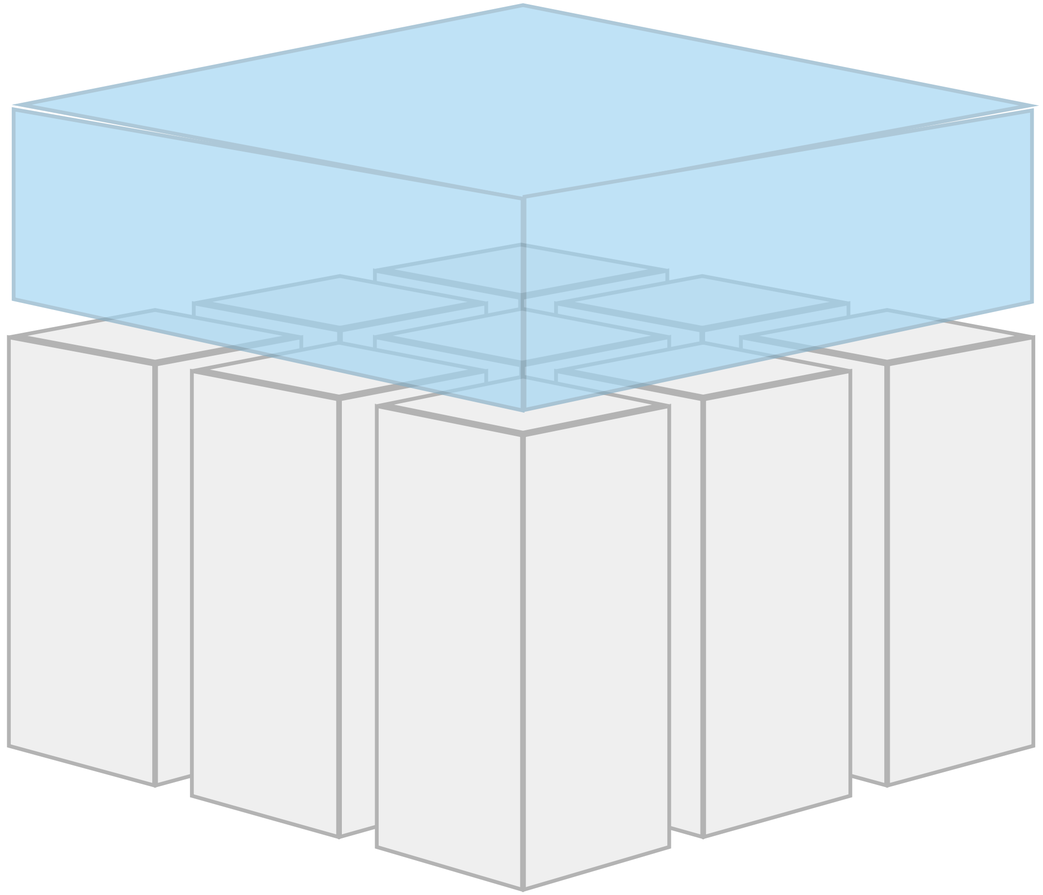
# Pattern: DDDD (Distributed Domain-driven Design)

## Consequences:

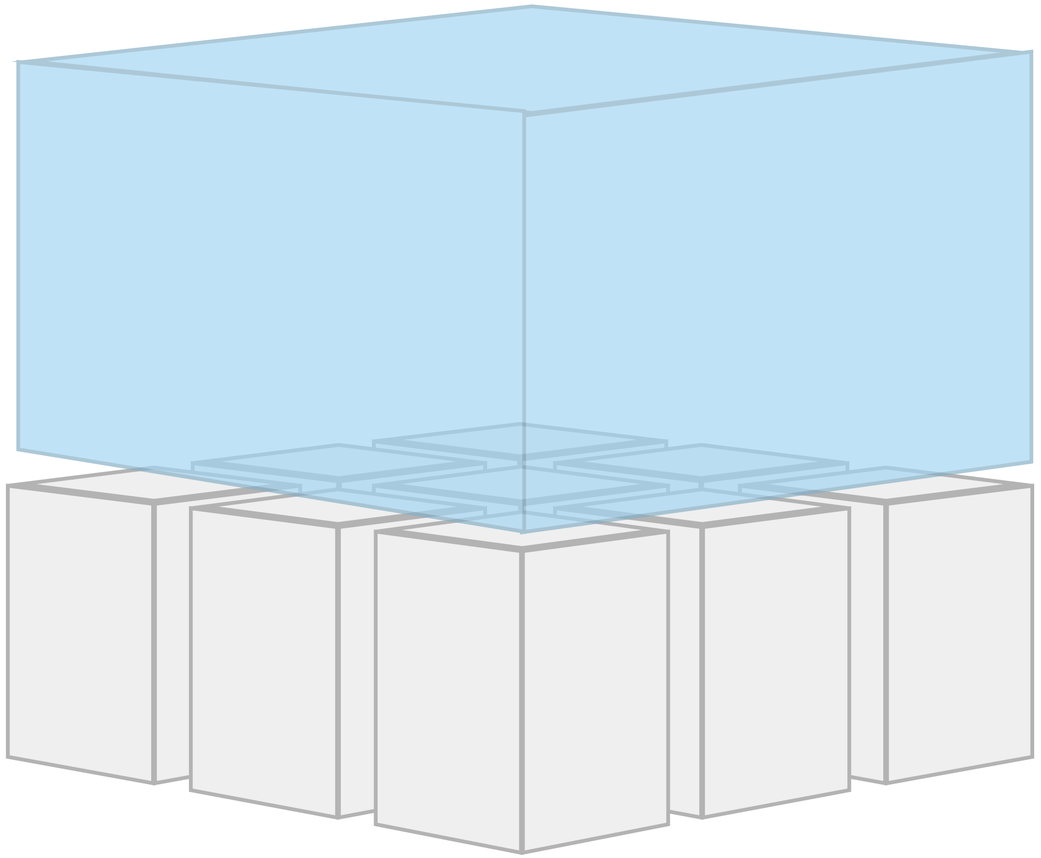
- Loose coupling between context
- Acknowledges separate evolution of contexts
- Asynchronicity increases stability
- Well-suited for to support parallel development

# That UI thing? Easy!

# Assumption



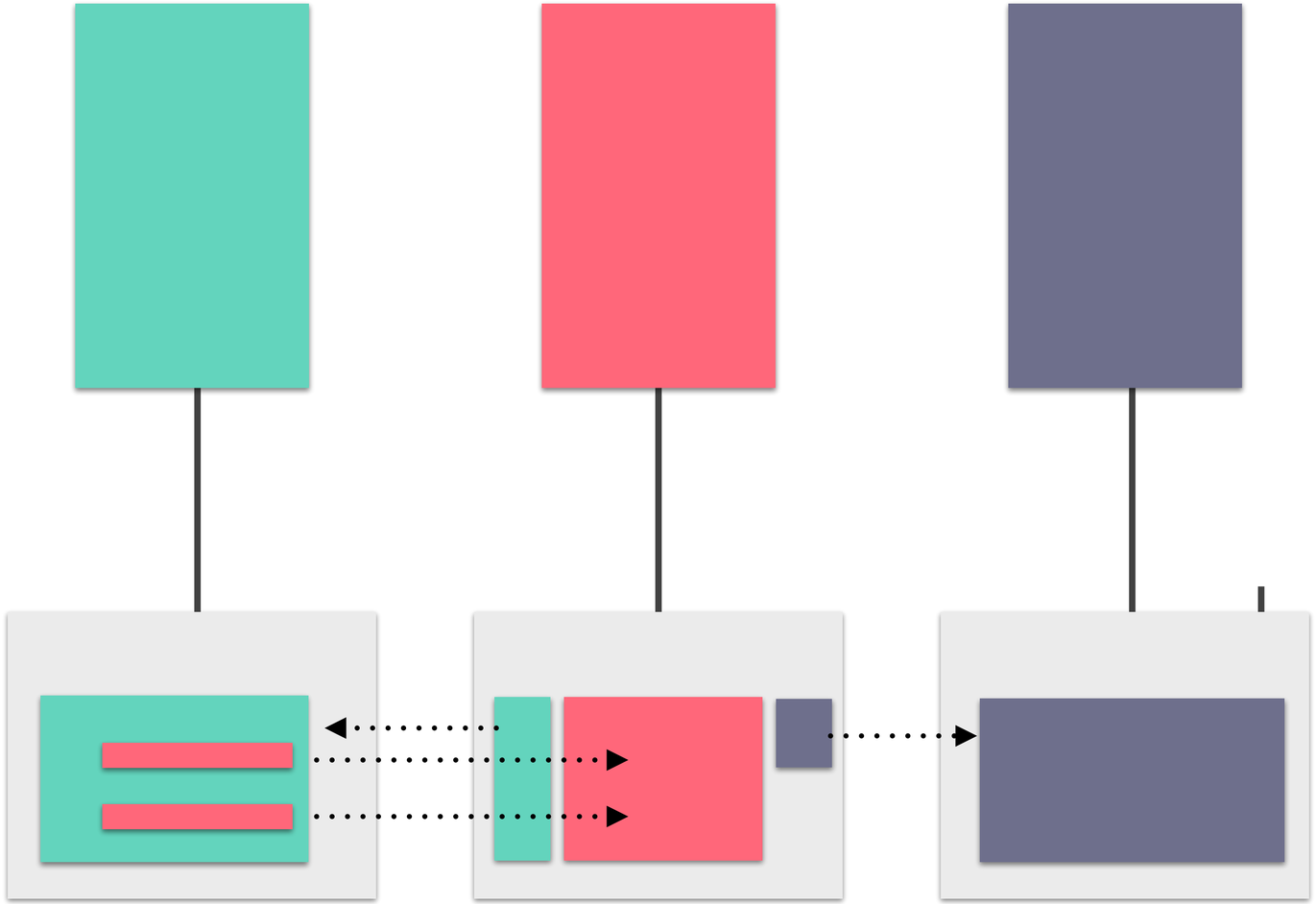
# Reality – Antipattern: Frontend Monolith



# Example: E-Commerce Site

- Register & maintain account
- Browse catalog
- See product details
- Checkout
- Track status

→SCS



# Pattern: SCS (Self-contained Systems)

## Description:

- Self-contained, autonomous
- Including UI + DB
- Possibly composed of smaller microservices

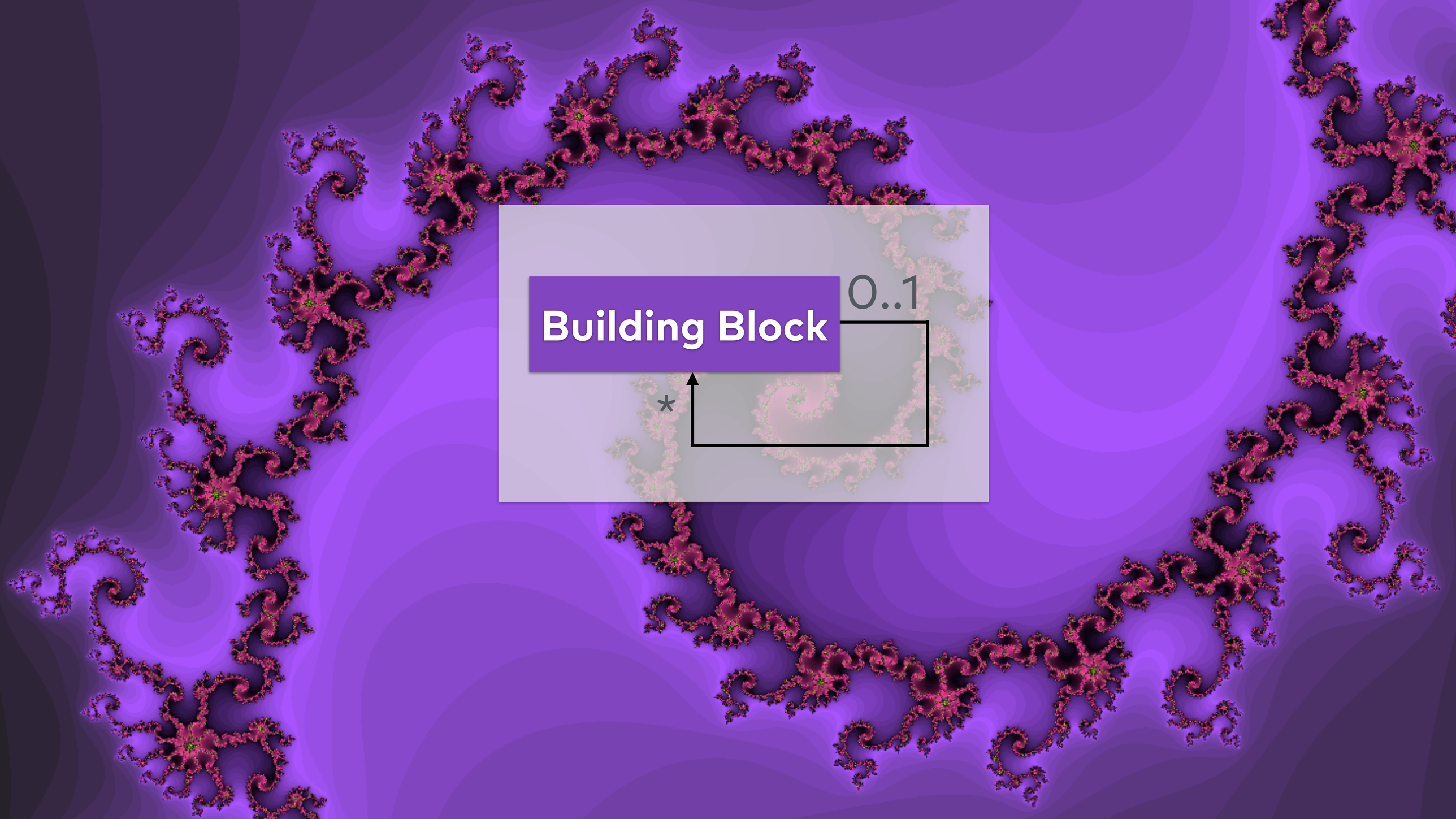
## As seen on:

- Amazon
- Groupon
- Otto.de
- https://scs-architecture.org

# Pattern: SCS (Self-contained Systems)

## Consequences:

- Larger, independent systems,
  including data + UI (if present)
- Able to autonomously serve requests
- Light-weight integration, ideally via front-end
- No extra infrastructure needed
- Well suited if goal is decoupling of development teams


Building Block
0..1
*

# One more thing ...


We love monoliths –
so let's build a lot of them!

Separate separate things

Join things that belong together

# That's all I have.
# Thanks for listening!

Stefan Tilkov
@stilkov
stefan.tilkov@innoq.com
Phone: +49 170 471 2625

www.innoq.com

## SERVICES

Strategy & technology consulting
Digital business models
Software architecture & development
Digital platforms & infrastructures
Knowledge transfer, coaching & trainings

## FACTS

~125 employees
Privately owned
Vendor-independent

## OFFICES

Monheim
Berlin
Offenbach
Munich
Zurich

## CLIENTS

Finance
Telecommunications
Logistics
E-commerce
Fortune 500
SMBs
Startups