A Question of Size

Stefan Tilkov, stefan.tilkov@innoq.com, @stilkov

Java Forum Nord 2017

innoQ

Reviewing architectures

Generic Architecture Review Results

Building features takes too long

Technical debt is well-known and not addressed

Deployment is way too complicated and slow

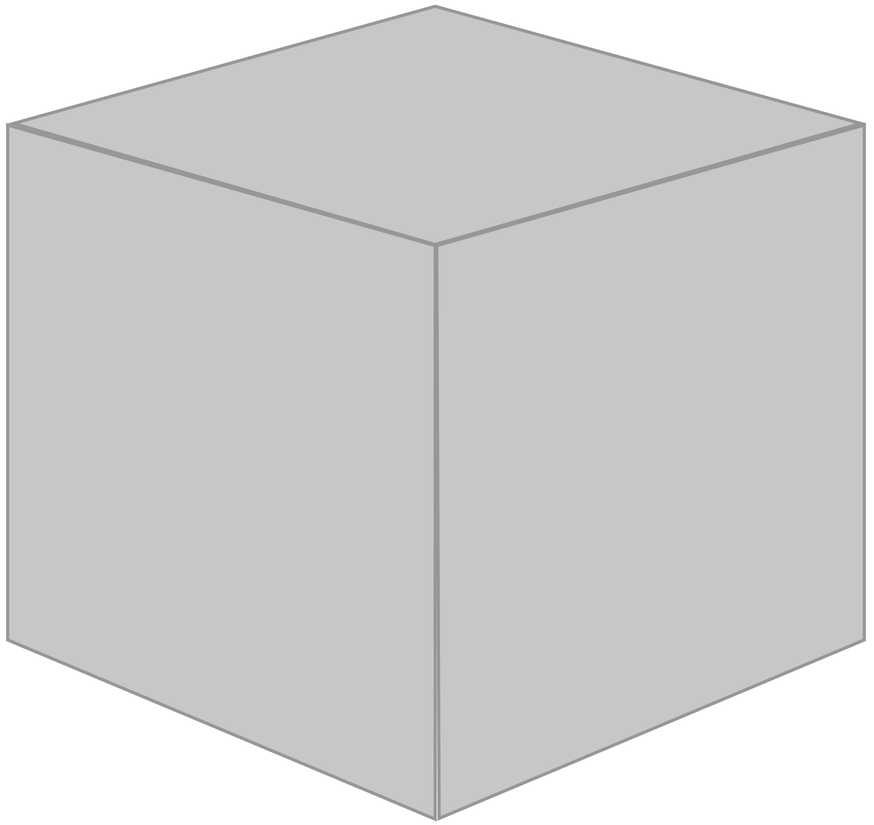
Architectural quality has degraded

Scalability has reached its limit

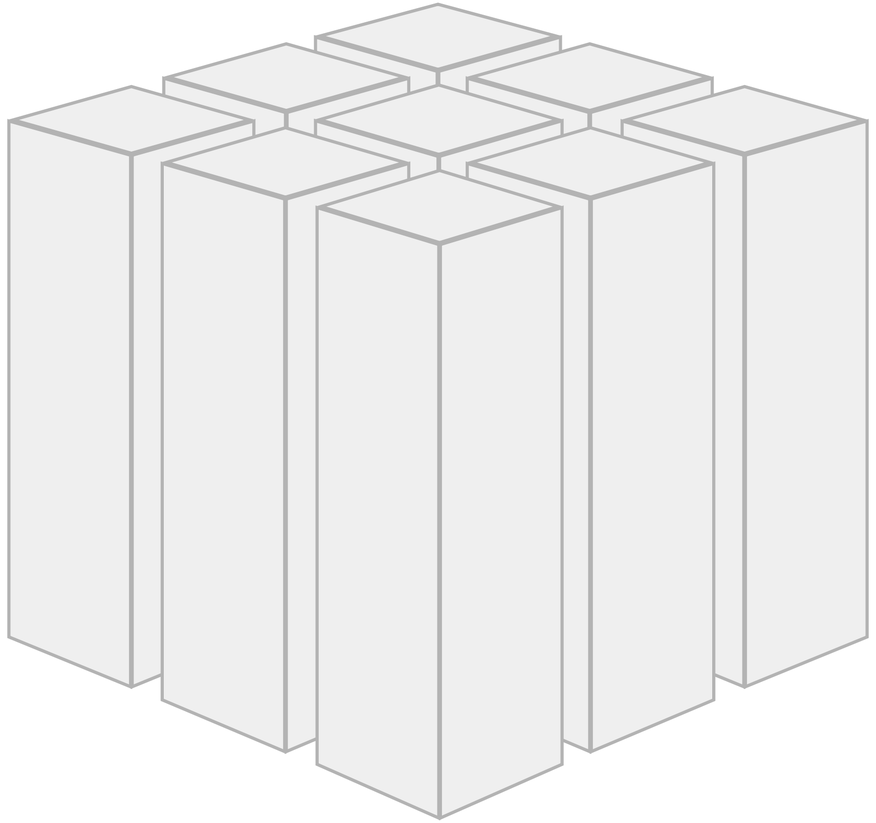
“-ility” problems abound

Replacement would be way too expensive

So let’s start with this …



… and cut it apart: *Voilà, Microservices!*



“Microservices” are building blocks of an architectural style that uses deployment boundaries as a first-class software architecture principle

How big shall each individual service be?

Just make things the *right* size

High Cohesion
Loose Coupling

Vocabulary

adhesive: able to stick fast to a surface or object; sticky:

cohesive: characterized by or causing cohesion

cohesion: the action or fact of forming a united whole;
in physics: the sticking together of particles of the same substance

inherent: existing in something as a permanent, essential, or characteristic attribute

http://vanderburg.org/blog/Software/Development/cohesion.rdoc

Separate separate things

Join things that belong together

Building blocks

lambdas

components

functions

services

containers

dynamic libraries

VMs

units

objects

libraries

images

classes

procedures

shared objects

modules

microservices

Commonalities

boundary

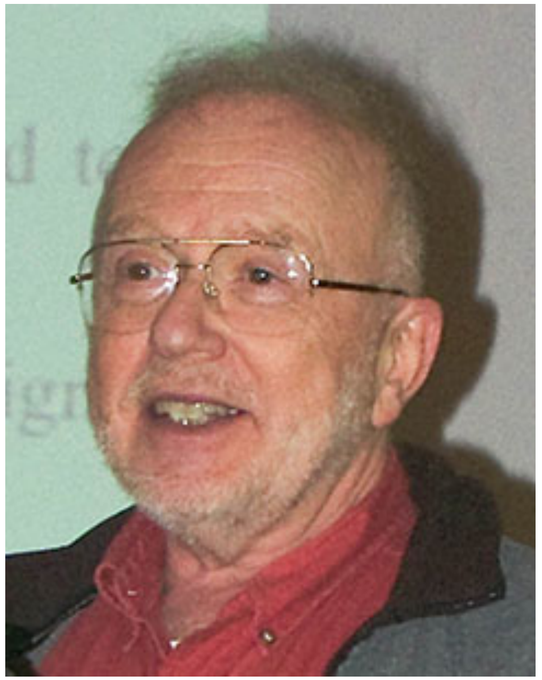
environment

implementation

dependencies

interface

Information Hiding



"[I]t is almost always incorrect to begin the decomposition of a system into modules on the basis of a flowchart. We propose instead that one begins with a list of difficult design decisions or design decisions which are likely to change. ***Each module is then designed to hide such a decision from the others.****"*

David L. Parnas, 1971

http://www.cs.umd.edu/class/spring2003/cmsc838p/Design/criteria.pdf

Single Responsibility Principle

*"**A class [or module] should only have one reason to change. [...]** The SRP is one of the simplest of the principles, and one of the hardest to get right. Finding and separating those responsibilities from one another is much of what software design is really about."*

*"There is a corrolary here. **An axis of change is only an axis of change if the changes actually occur."***

Robert C. Martin, 1995/2003

http://www.butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod

Indicators of *strong* cohesion

simple to understand

simple to explain

difficult to split

one stakeholder

one reason to change

(re-)used as a whole

Indicators of *weak* cohesion

hard to understand

difficult to explain

obviously divisible

multiple stakeholders

partially re-used

many reasons to change

Forces for separation

Different environments (scale, performance, security, …)

Frequency of change

Weight

Need for reuse

Crosscutting concerns

Technical dependencies

Domain dependencies

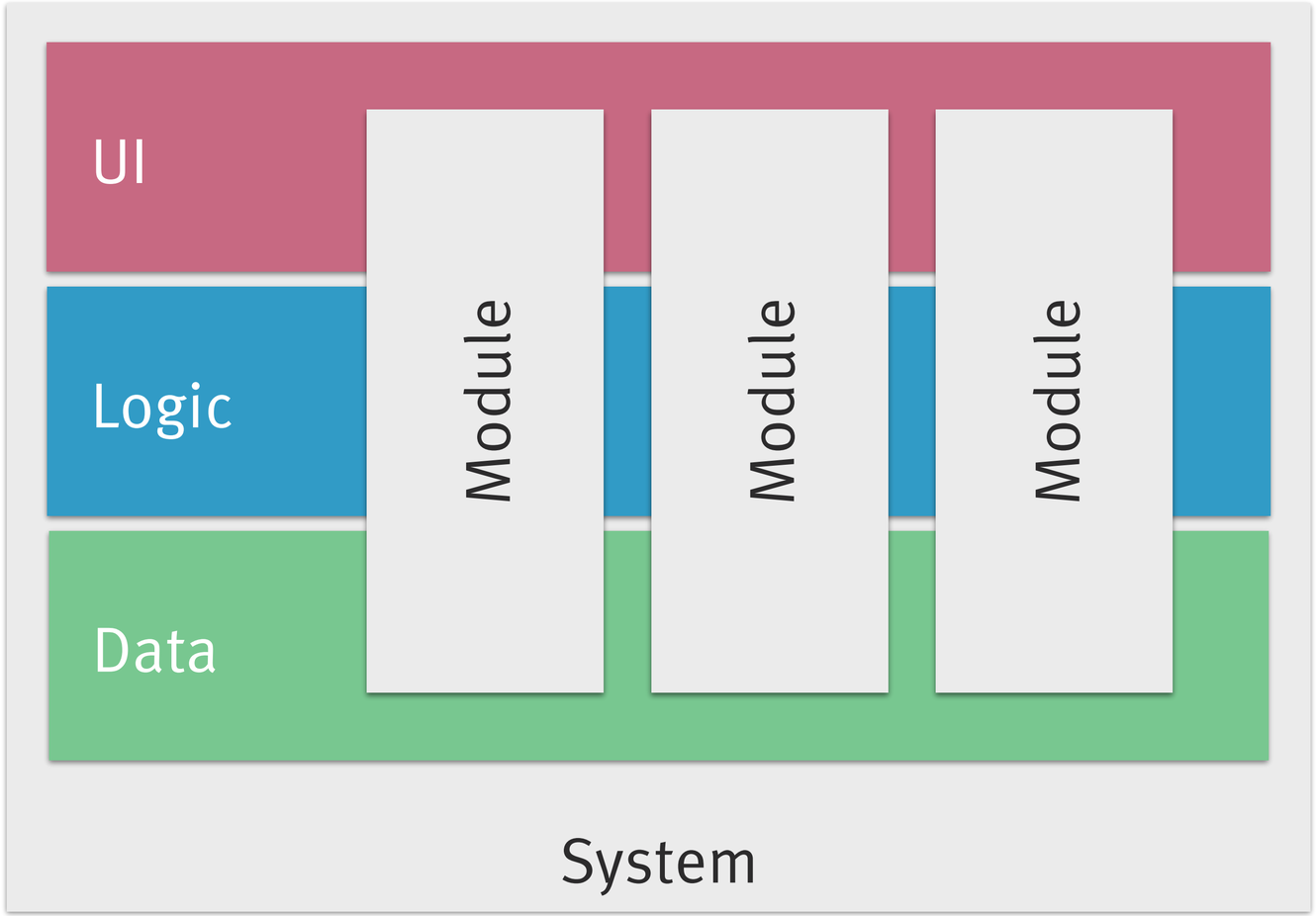
Parallel/isolated runtime

Implementation

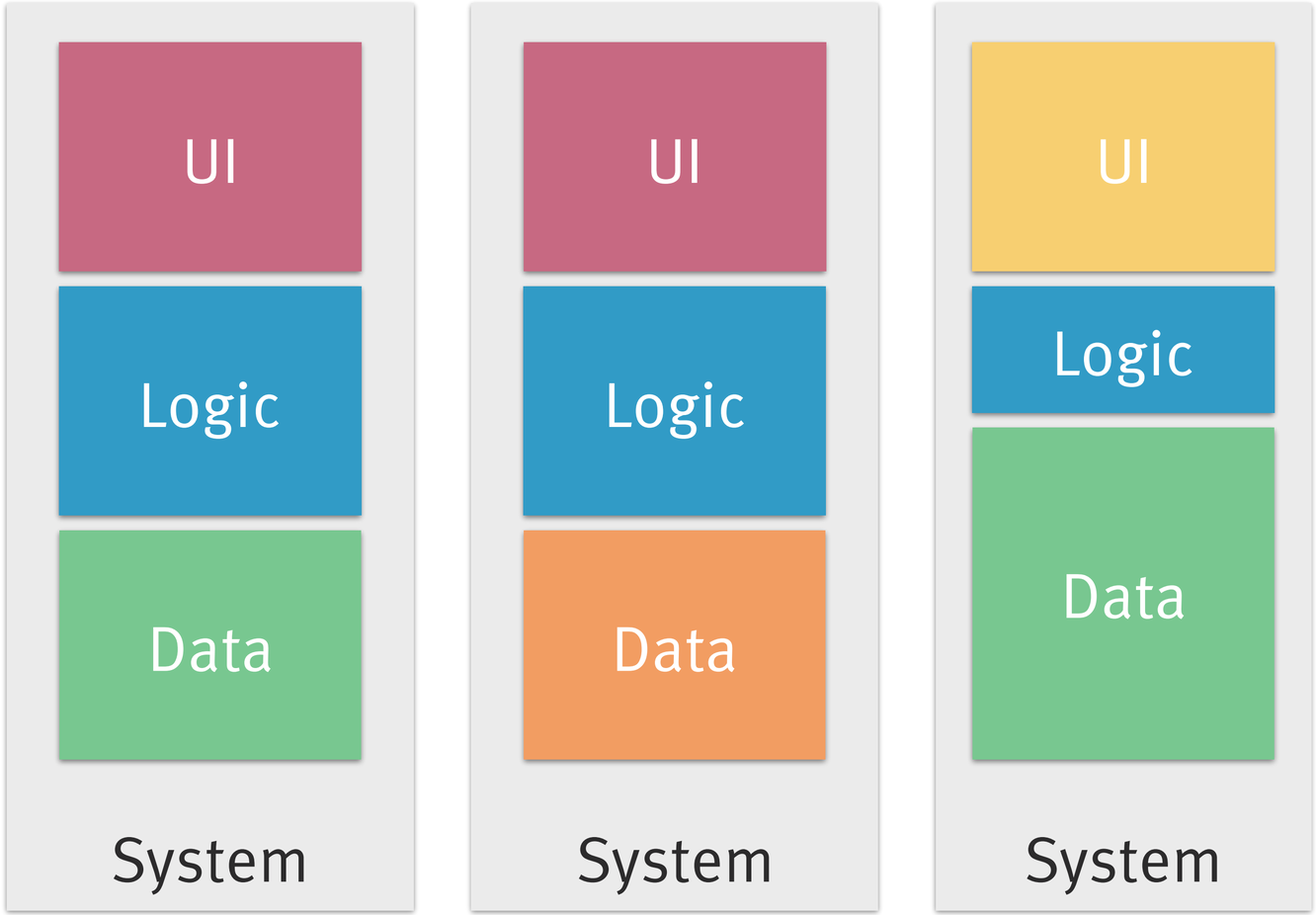
Parallel/isolated development

Multiple Dimensions
Different Priorities

Layered system



System of systems



Let’s talk about Microservices

Microservices – Common Traits

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

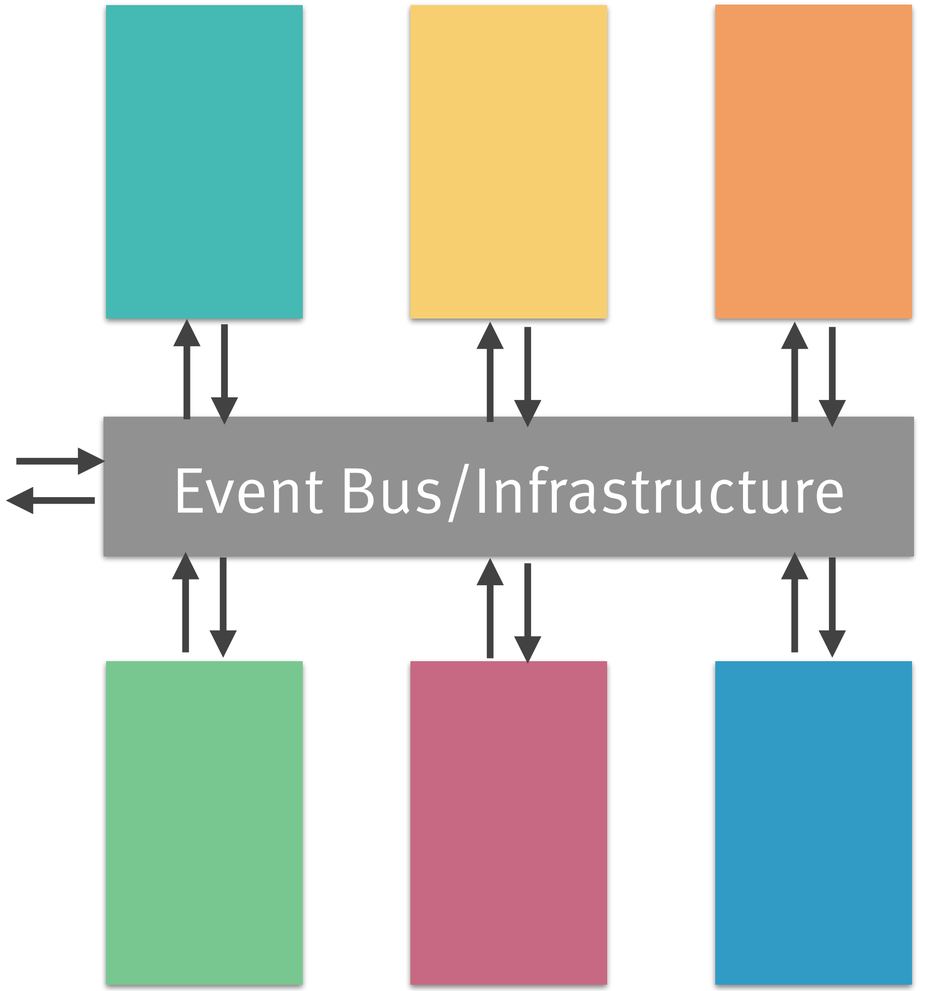
Benefits

1. Isolation
2. Autonomy
3. Indidual Scalability
4. Resilience
5. Speed
6. Experimentation
7. Rapid Feedback
8. Flexibility
9. Replaceability
10. Ecosystem

Example: Pricing Engine

- Default product prices
- General discounts
- Customer-specific discounts
- Campaign-related rebates

→FaaS



FaaS – Function as a Service

Characteristics:

- 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

(*) https://leanpub.com/serverless

FaaS – Function as a Service

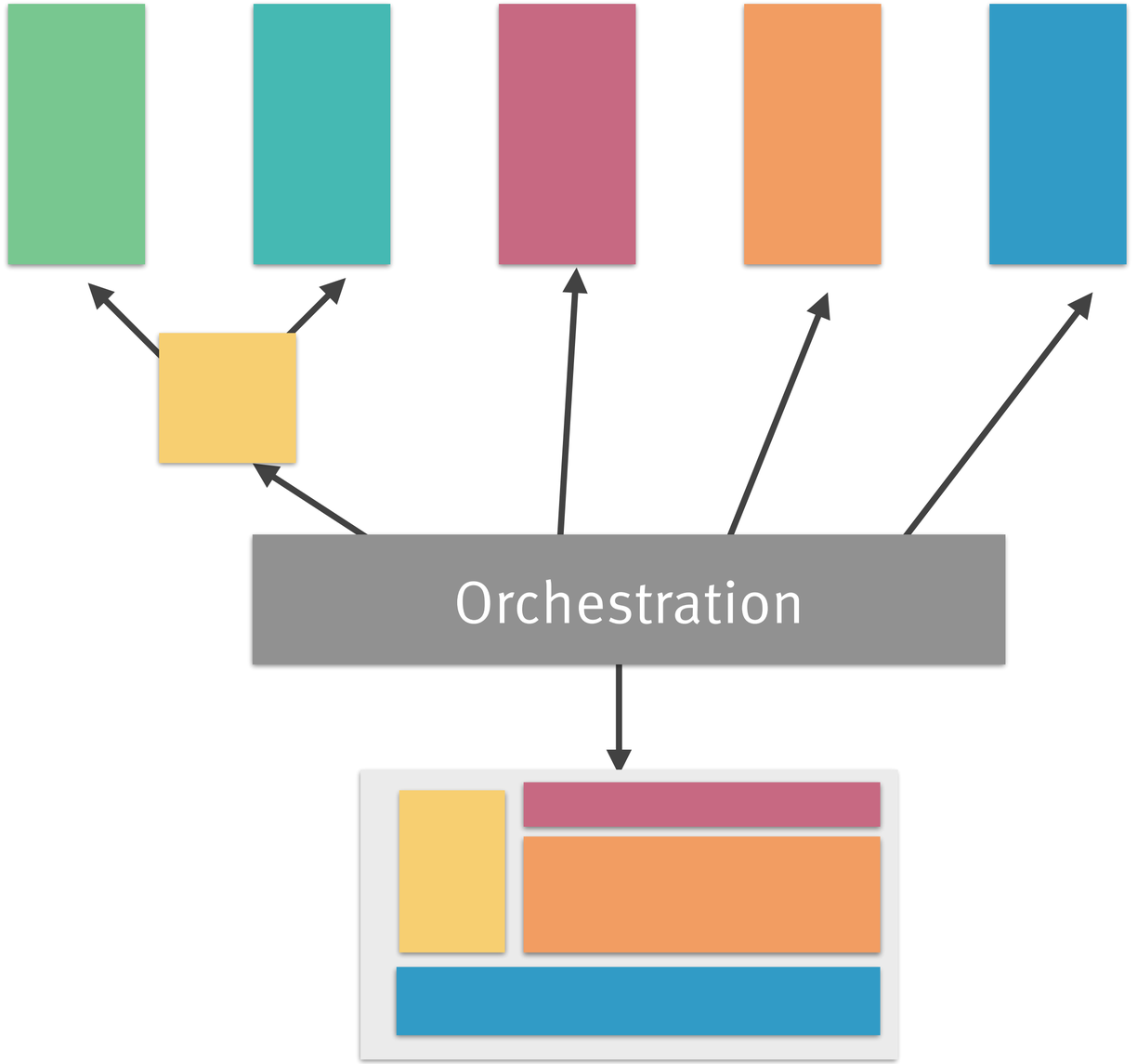
Consequences:

- Close collaboration – common goal
- Shared strong infrastructure dependency
- Common interfaces, multiple invocations
- Close similarity to actor-based environments
- Well suited to decomposable/“fuzzy” business problems

Example: Product Detail Page

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

→ µSOA



µSOA – Microservice-oriented Architecture

Characteristics:

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

As seen on:

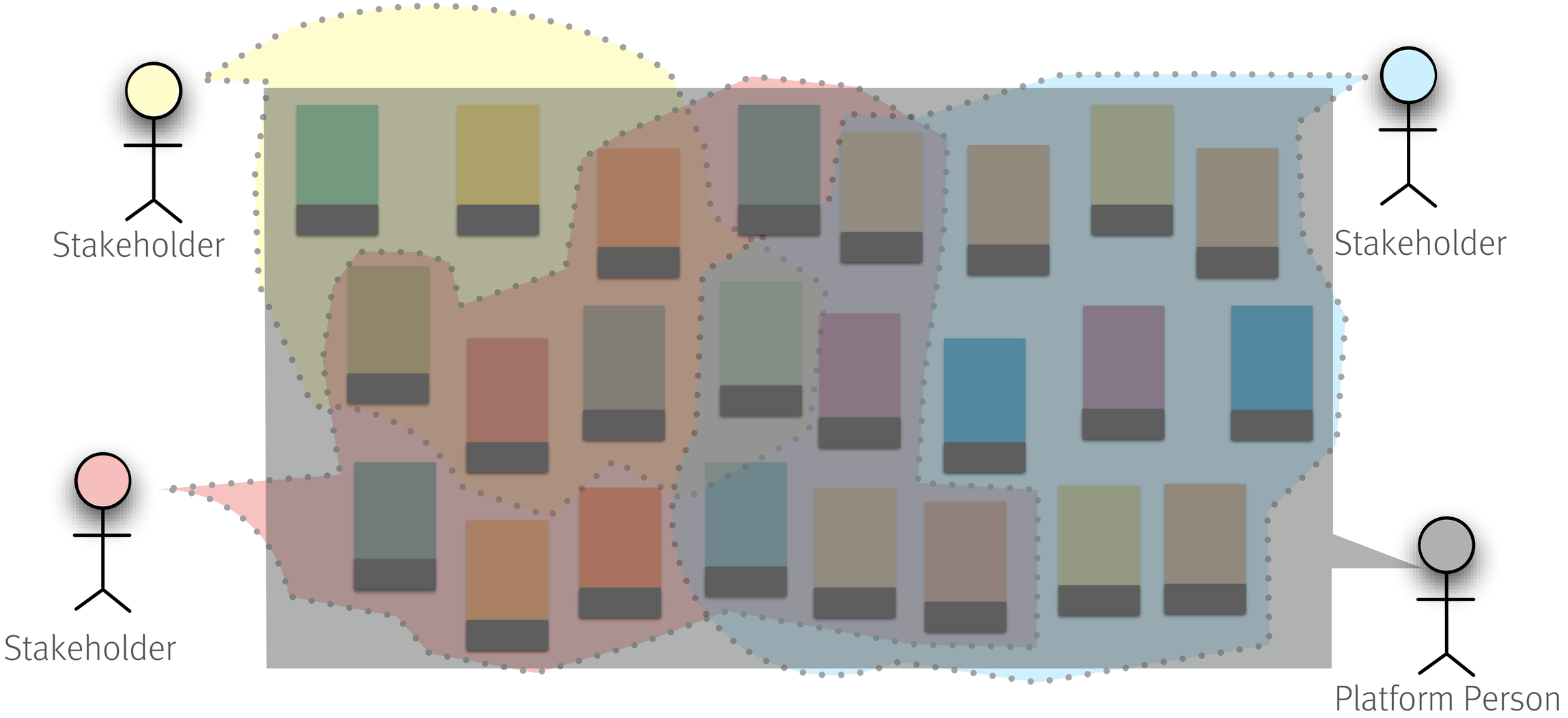
- Netflix
- Twitter
- Gilt

µSOA – Microservice-oriented Architecture

Consequences:

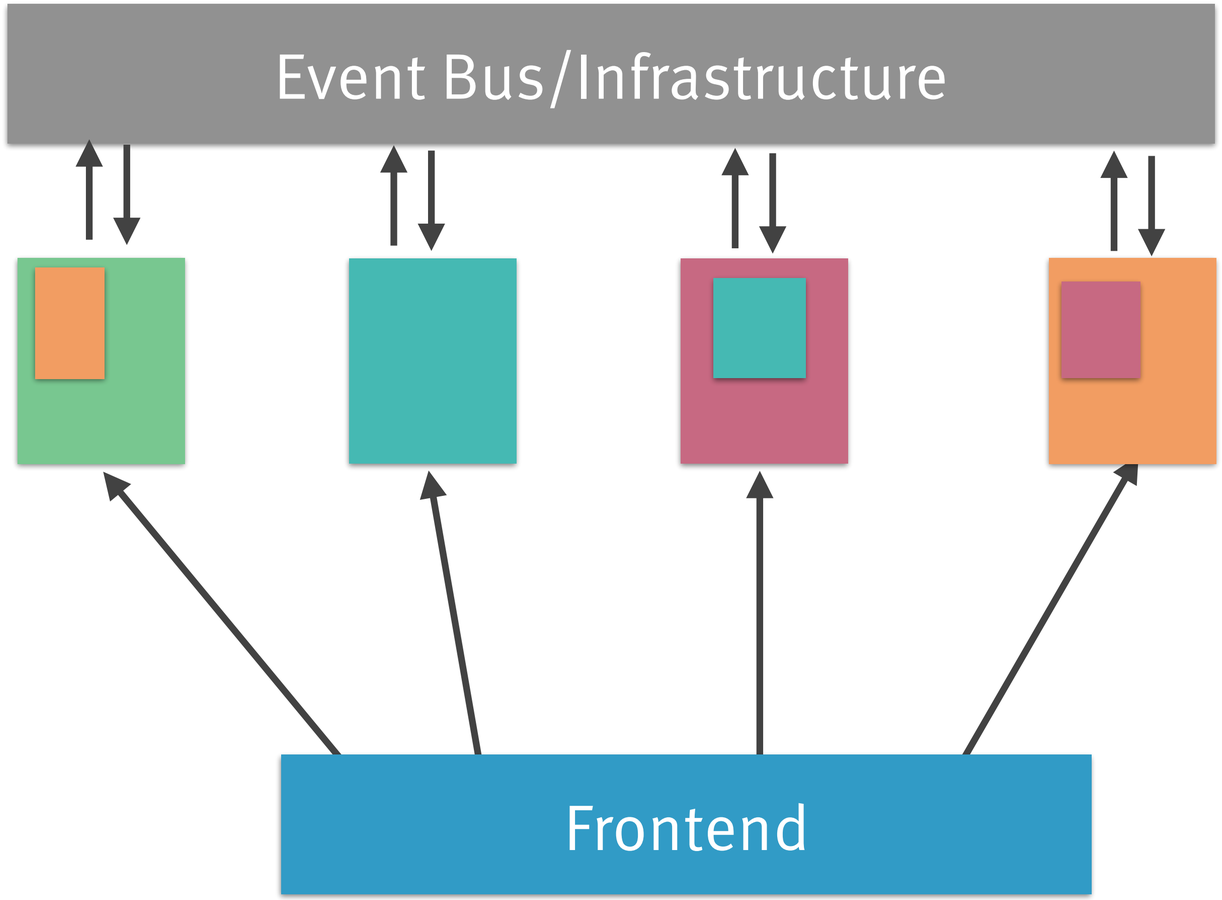
- Close collaboration – common goal
- Need for resilience/stability patterns for invocations
- Often combined with parallel/streaming approach
- Well suited to environments with extreme scalability requirements

Antipattern: Decoupling Illusion



Example: Logistics Application

- Order management
- Shipping
- Route planning
- Invoicing



→DDDD

DDDD – Distributed Domain-driven Design

Characteristics:

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

As seen on:

- (undisclosed)

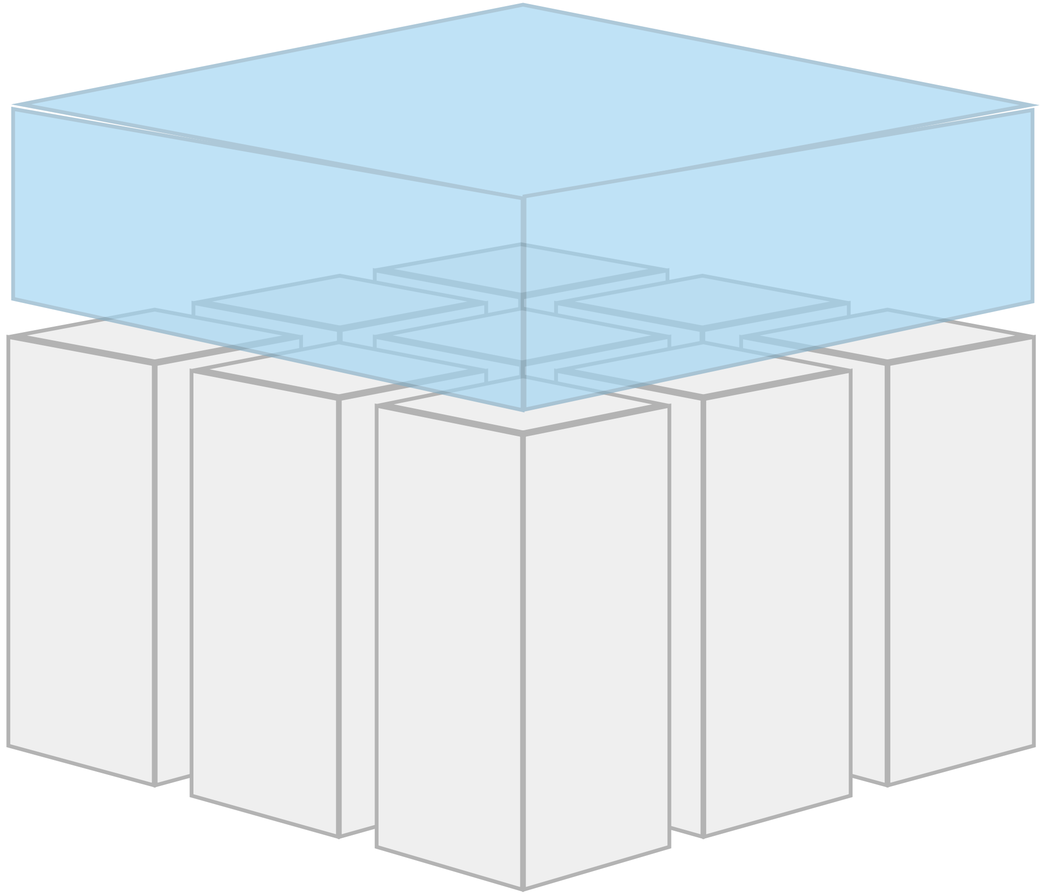
DDDD – Distributed Domain-driven Design

Consequences:

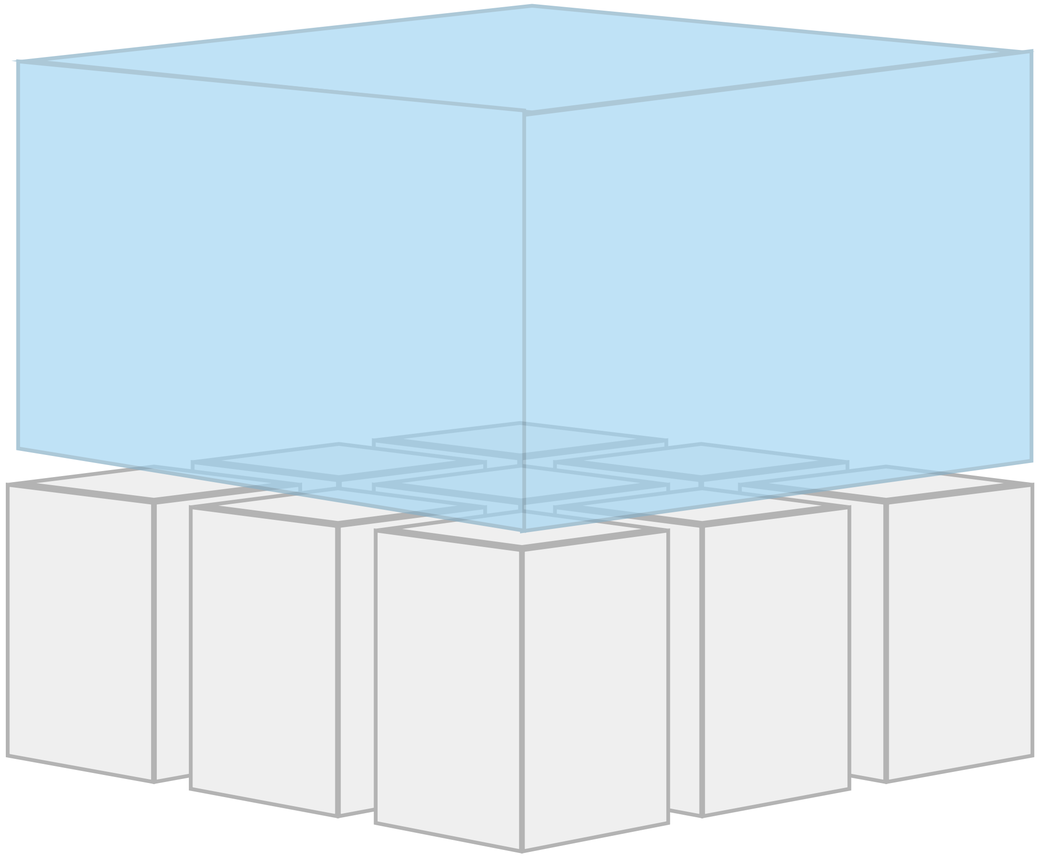
- Loose coupling between context
- Acknowledges separate evolution of contexts
- Asynchronicity increases stability
- Well-suited for to support parallel development

That UI thing? Easy!

Assumption



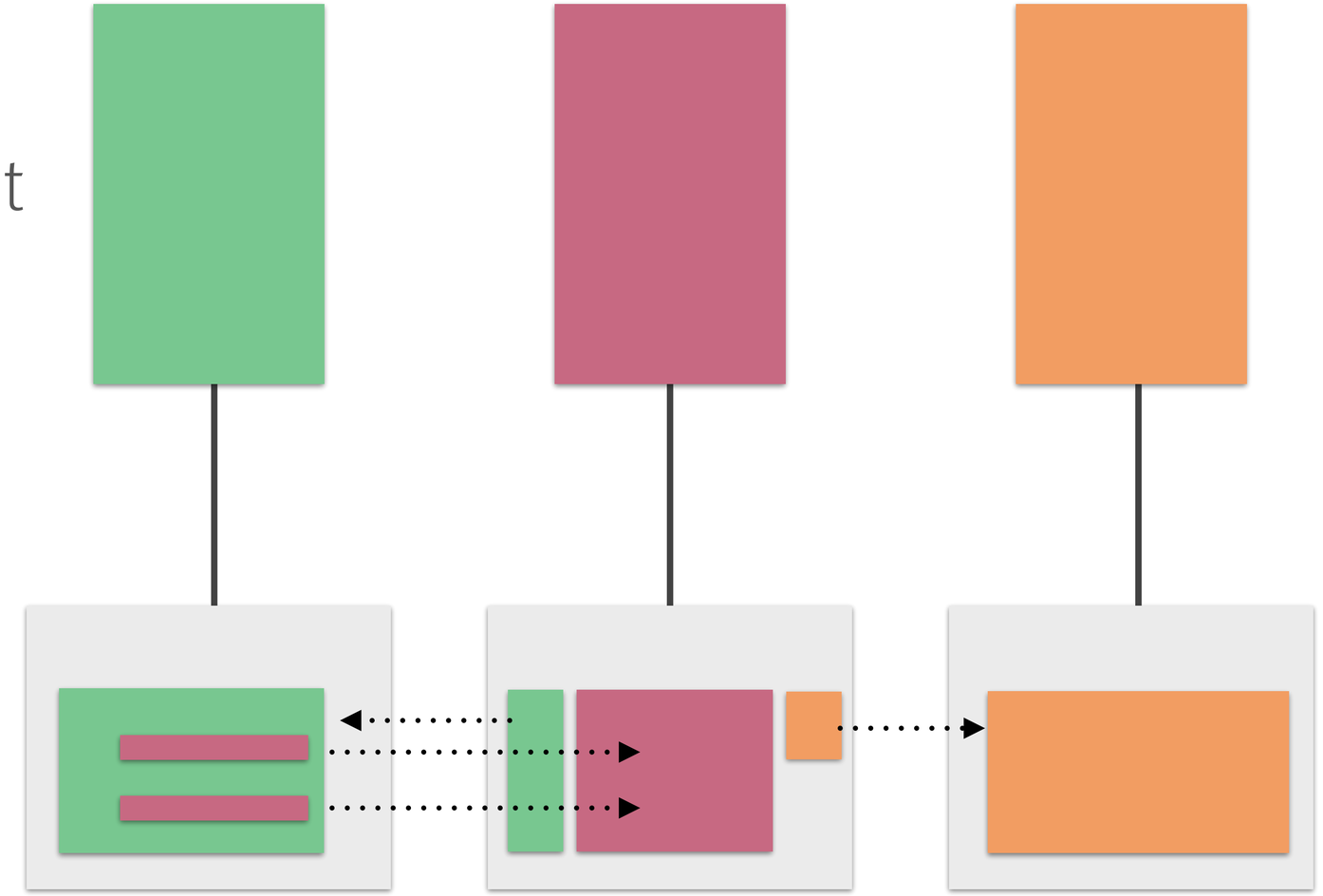
Reality



Example: E-Commerce Site

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

→SCS



SCS – Self-contained Systems

Characteristics:

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

As seen on:

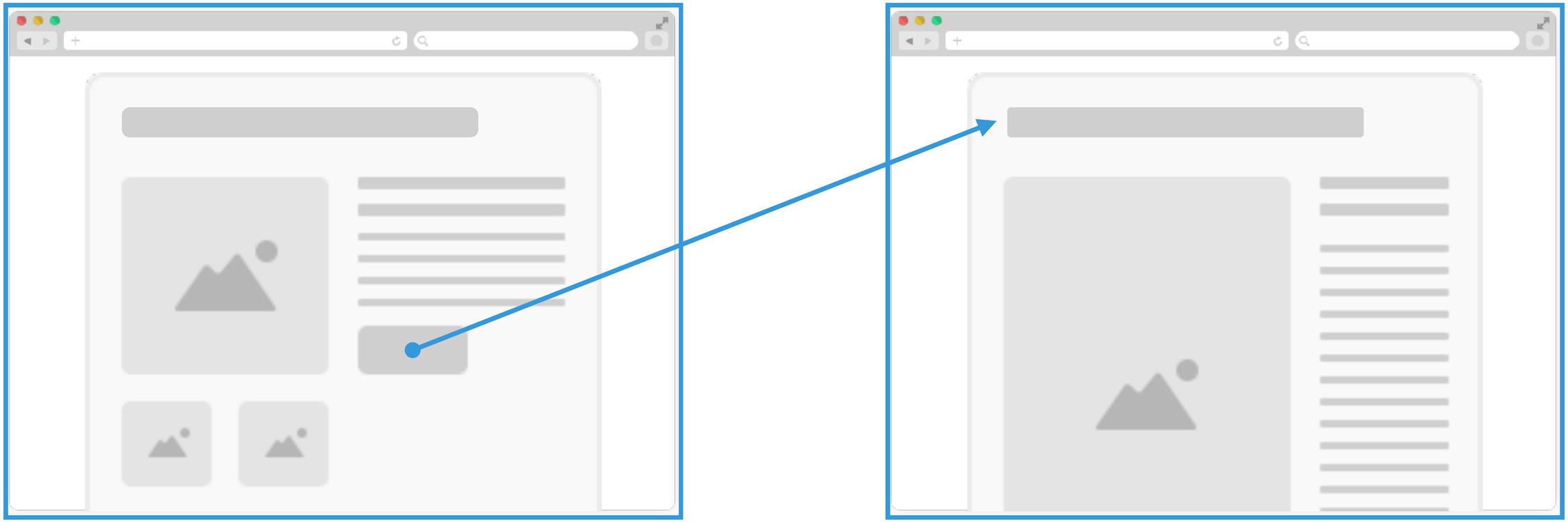
- Amazon
- Groupon
- Otto.de
- https://scs-architecture.org

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

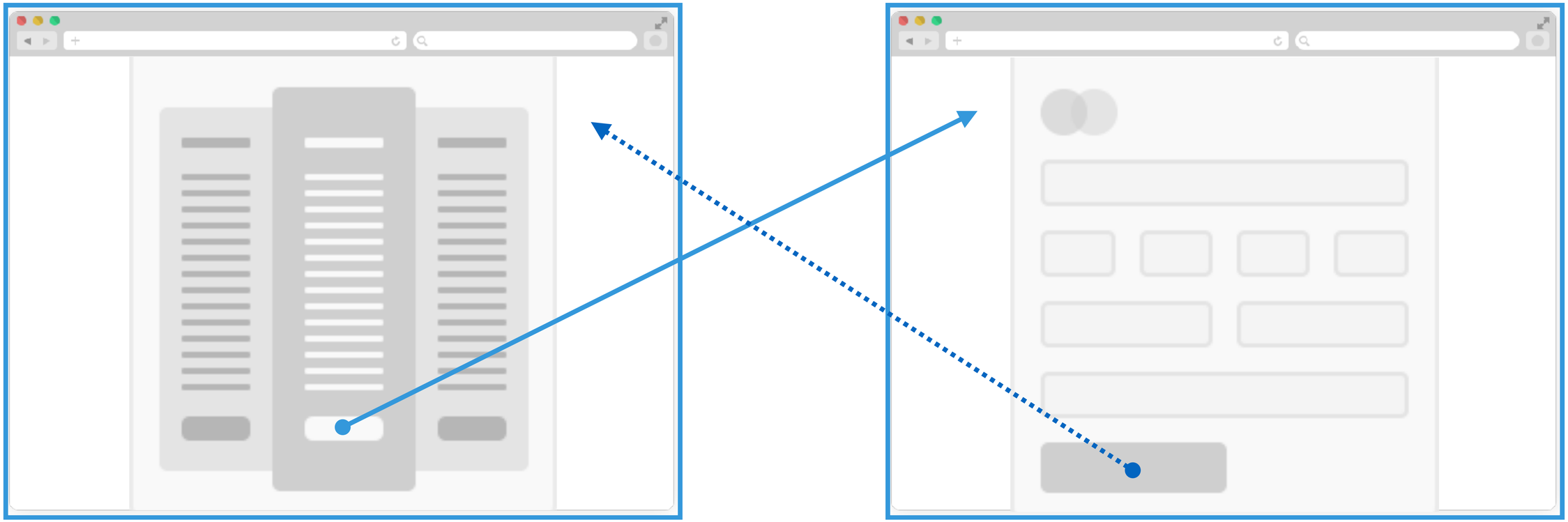
Web UI Integration: Links



System 1

System 2

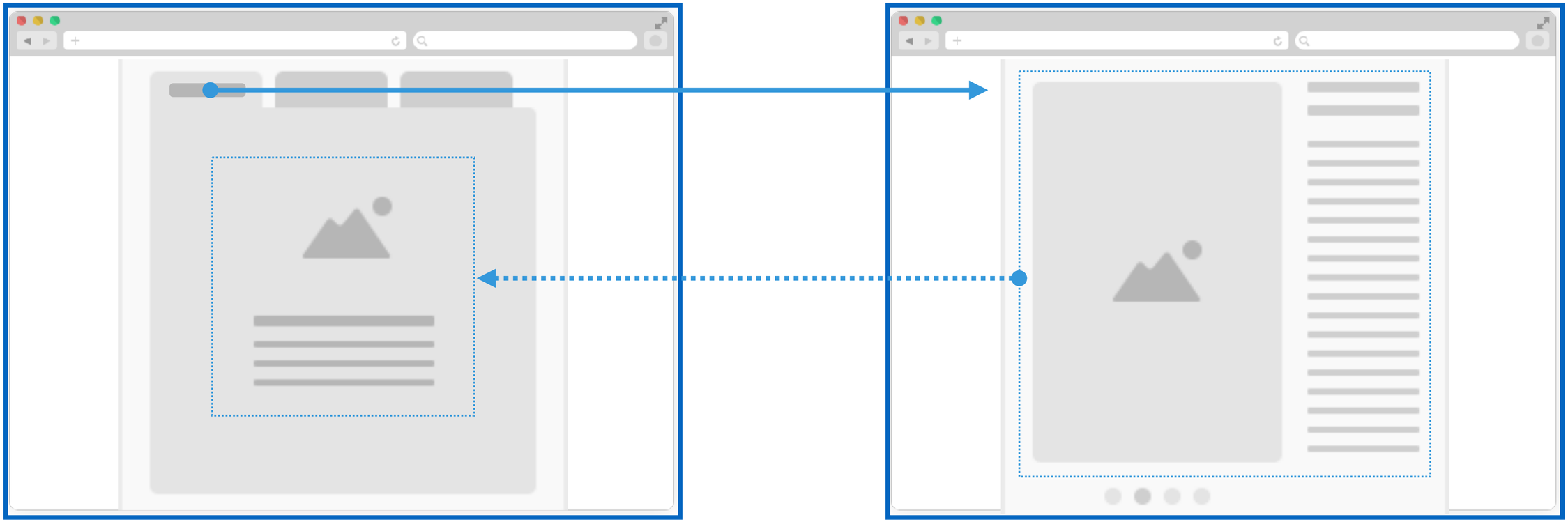
Web UI Integration: Redirection



System 1

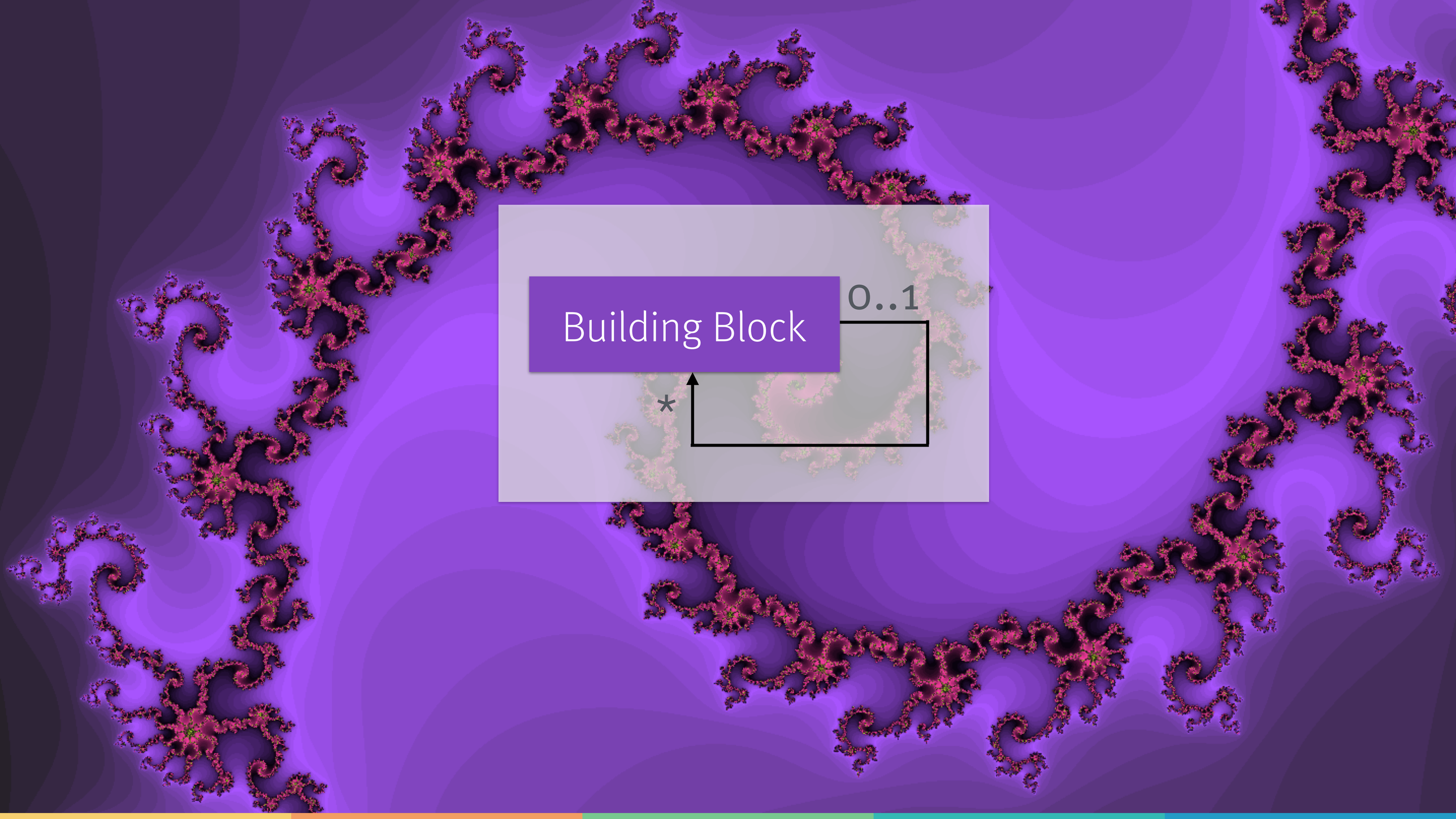
System 2

Web UI Integration: Transclusion



System 1

System 2


Building Block
0..1
*

So what?

Summary & Recommendations

1.

Explicitly design system boundaries

2.

Start front-to-back instead of top-down or bottom-up

3.

Modularize systems using the appropriate approach, including monoliths

That’s all I have.

thanks for listening!

@stilkov

Stefan Tilkov

stefan.tilkov@innoq.com

Phone: +49 170 471 2625

innoQ

www.innoq.com

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim am Rhein
Germany
Phone: +49 2173 3366-0

Ohlauer Straße 43
10999 Berlin
Germany
Phone: +49 2173 3366-0

Ludwigstr. 180E
63067 Offenbach
Germany
Phone: +49 2173 3366-0

Kreuzstraße 16
80331 München
Germany
Phone: +49 2173 3366-0

innoQ Schweiz GmbH

Gewerbestr. 11
CH-6330 Cham
Switzerland
Phone: +41 41 743 0116

About Stefan Tilkov

- CEO/Co-founder & principal consultant at innoQ
- Focus on architecture, REST, Web
- Messing with technology since 1993
- stefan.tilkov@innoq.com, @stilkov

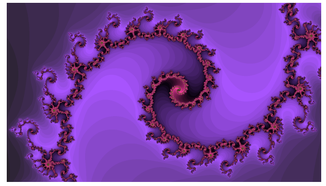
About innoQ

- Offices in Monheim (near Cologne), Berlin, Offenbach, Munich, Zurich
- ~125 employees
- Core competencies: software architecture consulting and software development
- Privately owned, vendor-independent
- Clients in finance, telecommunications, logistics, e-commerce; Fortune 500, SMBs, startups

www.innoq.com

Image Credit

Alice Popkorn, https://flic.kr/p/5NsmsK



hairchaser, https://flic.kr/p/aqNWyV