Microservices: Architecture to Scale Agile

Eberhard Wolff
Fellow, innoQ
@ewolff

http://continuous-delivery-buch.de/

Eberhard Wolff

Microservices

Grundlagen flexibler Softwarearchitekturen

dpunkt.verlag

http://microservices-buch.de/

Microservices

Flexible Software Architectures

Eberhard Wolff

http://microservices-book.com/

Eberhard Wolff

Microservices Primer

A Short Overview

FREE!!!!

innoQ

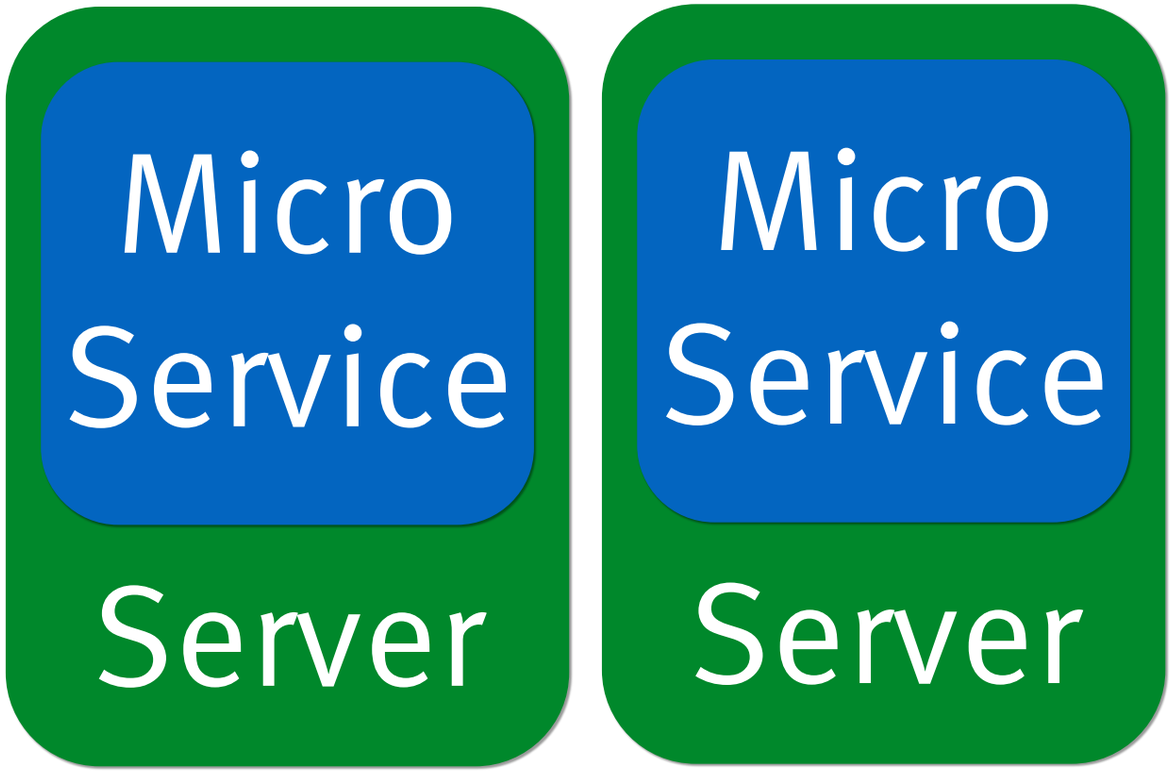
http://microservices-book.com/primer.html

- Microservices
- Agility
- Self-contained Systems
- SCS to scale agile

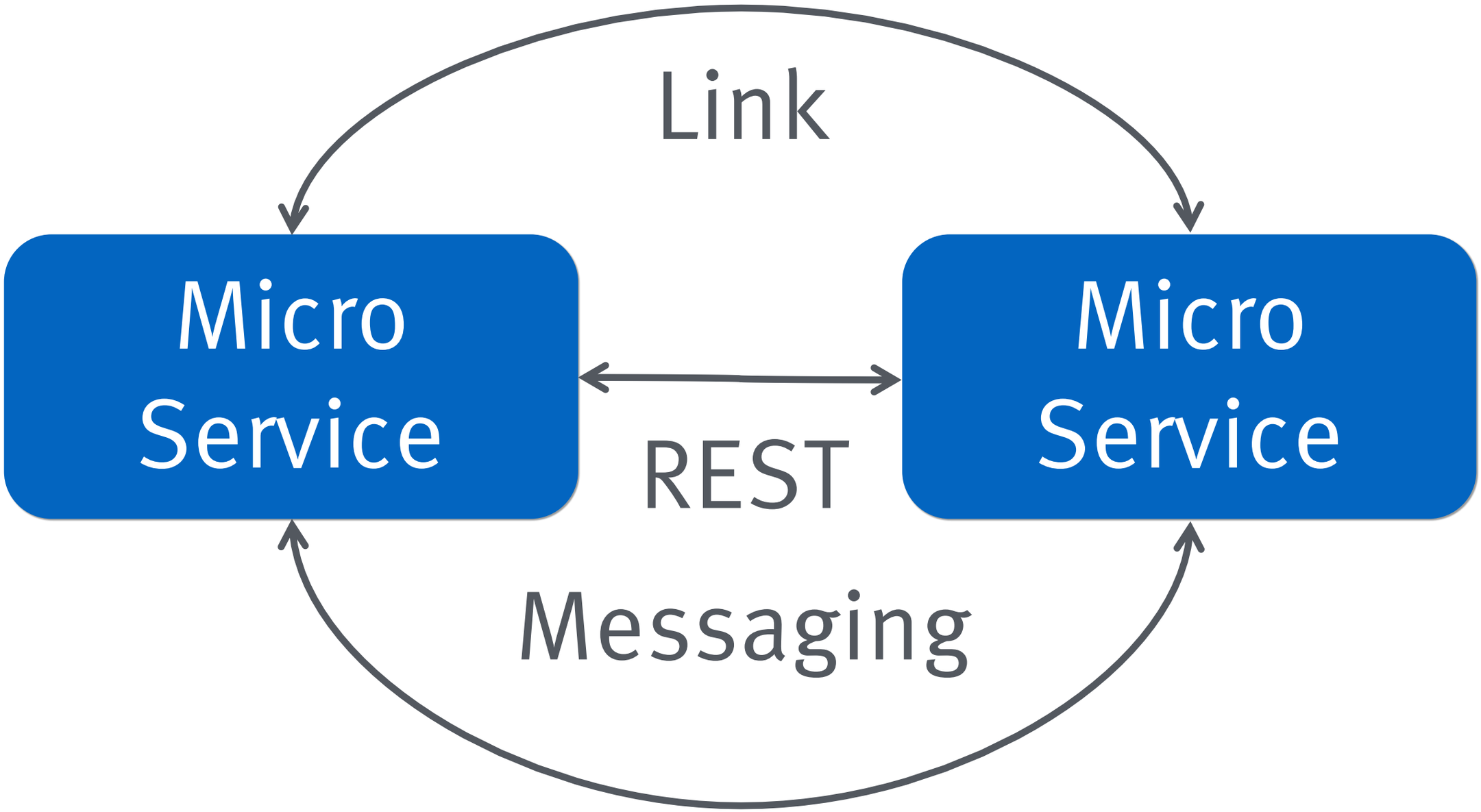
Microservices

Microservices: Definition

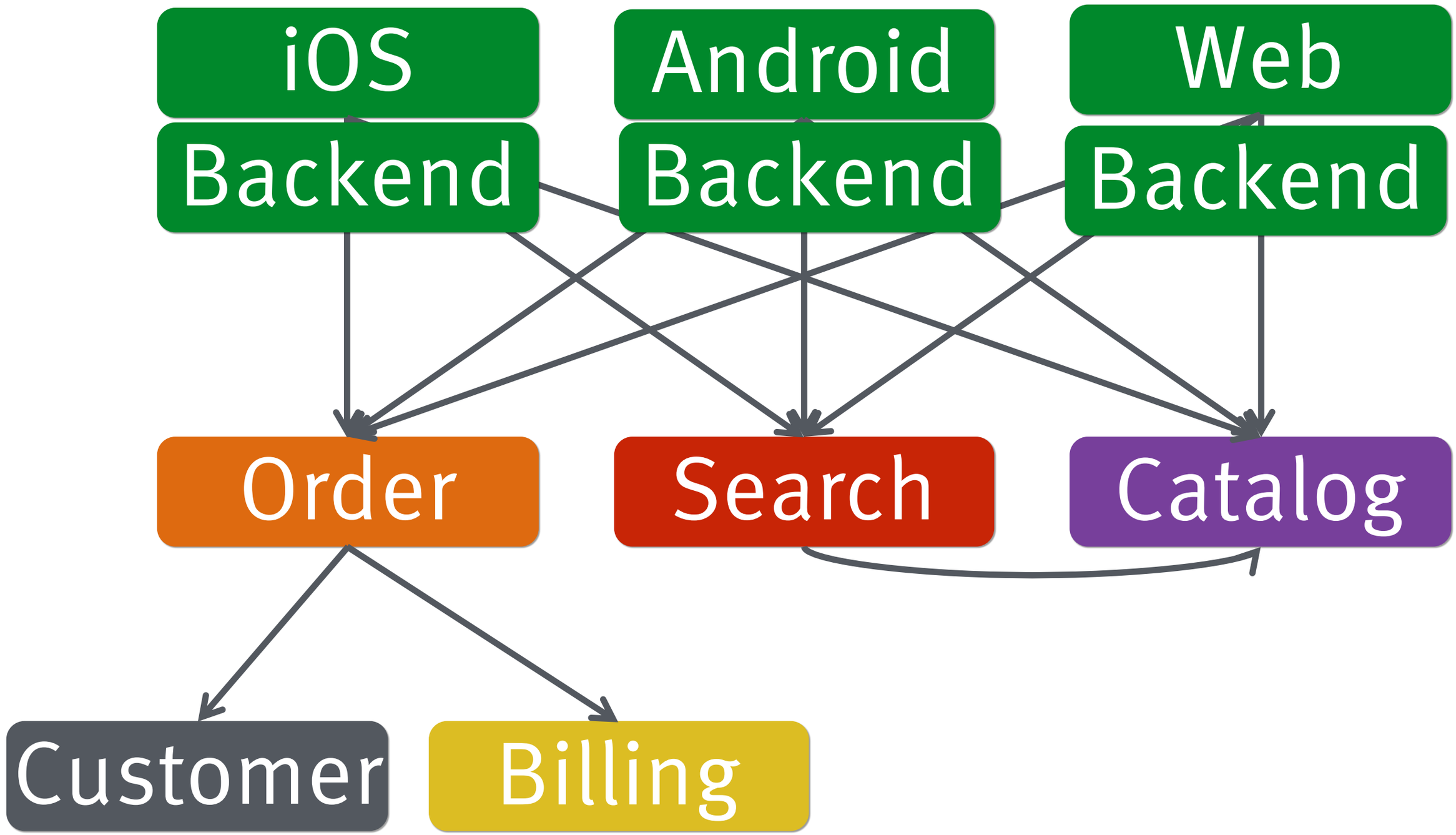
- Independent deployment units
- E.g. process, VMs, Docker containers
- Any technology
- Any infrastructure



Components Collaborate



Possible Microservices Architecture



Real-World Example

- Team = one person
- Microservices-based system

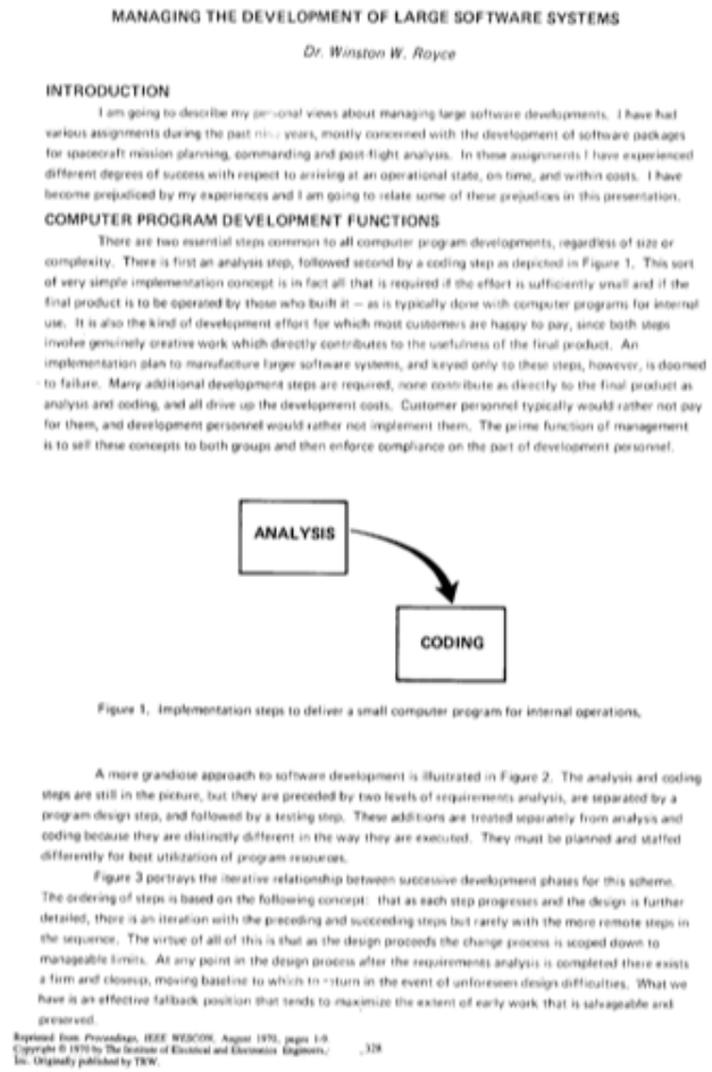
- Reasons
- Fast and easy deployment
- Clear separation

Microservices can do more than scaling agile!

Agility

Iterations

- Work in iterations
- Not following a fixed plan
- “Waterfall” paper 1970 (Royce): Do it twice!
- I.e. iterations are a well-known idea for very long



MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS

Dr. Winston W. Royce

INTRODUCTION

I am going to describe my personal views about managing large software developments. I have had various assignments during the past nine years, mostly concerned with the development of software packages for spacecraft mission planning, commanding and post-flight analysis. In these assignments I have experienced different degrees of success with respect to arriving at an operational state, on time, and within costs. I have become prejudiced by my experiences and I am going to relate some of these prejudices in this presentation.

COMPUTER PROGRAM DEVELOPMENT FUNCTIONS

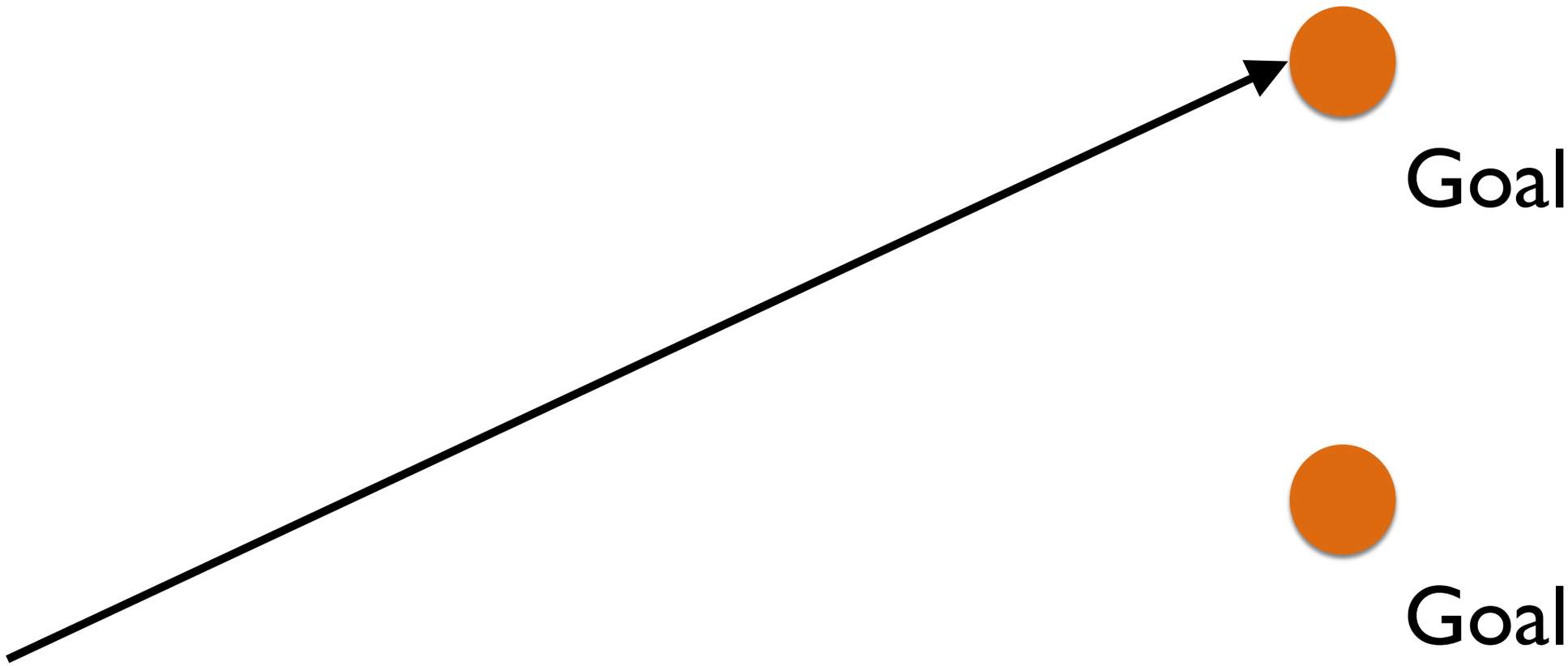
There are two essential steps common to all computer program developments, regardless of size or complexity. There is first an analysis step, followed second by a coding step as depicted in Figure 1. This sort of very simple implementation concept is in fact all that is required if the effort is sufficiently small and if the final product is to be operated by those who built it — as is typically done with computer programs for internal use. It is also the kind of development effort for which most customers are happy to pay, since both steps involve genuinely creative work which directly contributes to the usefulness of the final product. An implementation plan to manufacture larger software systems, and keyed only to these steps, however, is doomed to failure. Many additional development steps are required, none contribute as directly to the final product as analysis and coding, and all drive up the development costs. Customer personnel typically would rather not pay for them, and development personnel would rather not implement them. The prime function of management is to sell these concepts to both groups and then enforce compliance on the part of development personnel.

Figure 1. Implementation steps to deliver a small computer program for internal operations.

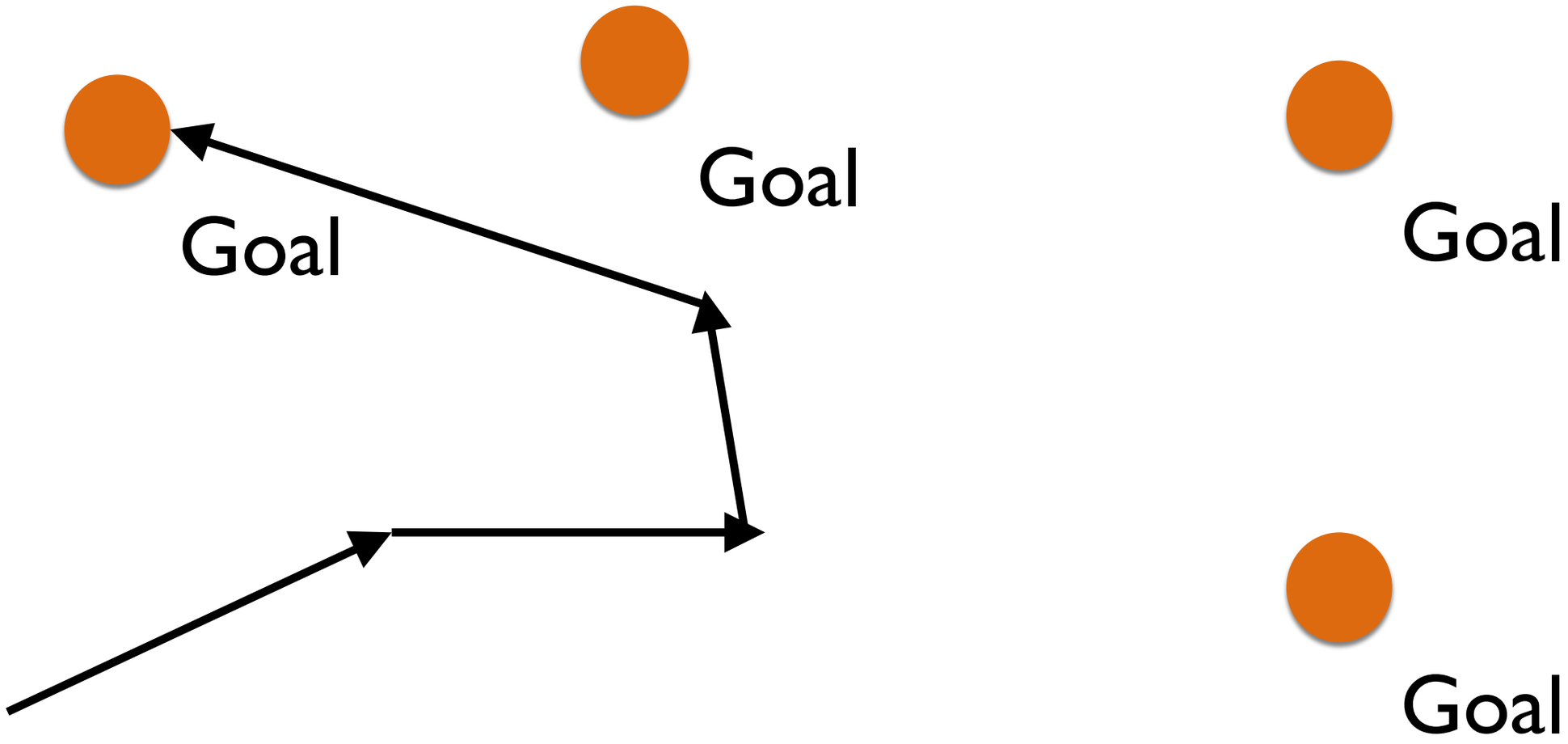
A more grandiose approach to software development is illustrated in Figure 2. The analysis and coding steps are still in the picture, but they are preceded by two levels of requirements analysis, are separated by a program design step, and followed by a testing step. These additions are treated separately from analysis and coding because they are distinctly different in the way they are executed. They must be planned and staffed differently for best utilization of program resources.

Figure 3 portrays the iterative relationship between successive development phases for this scheme. The ordering of steps is based on the following concept: that as each step progresses and the design is further detailed, there is an iteration with the preceding and succeeding steps but rarely with the more remote steps in the sequence. The virtue of all of this is that as the design proceeds the change process is scoped down to manageable limits. At any point in the design process after the requirements analysis is completed there exists a firm and closeup, moving baseline to which to return in the event of unforeseen design difficulties. What we have is an effective fallback position that tends to maximize the extent of early work that is salvageable and preserved.

No Iterations



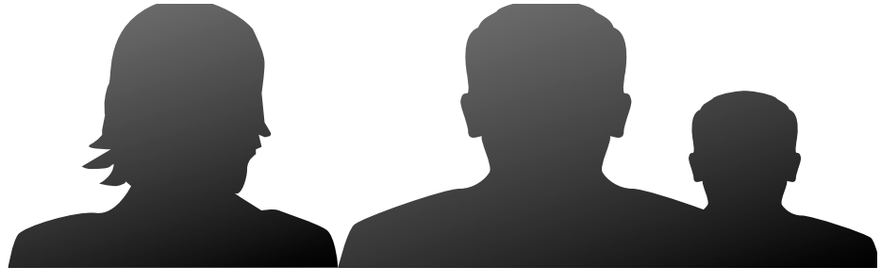
Iterations



Cross-functional Team

- Need lots of skill to develop features

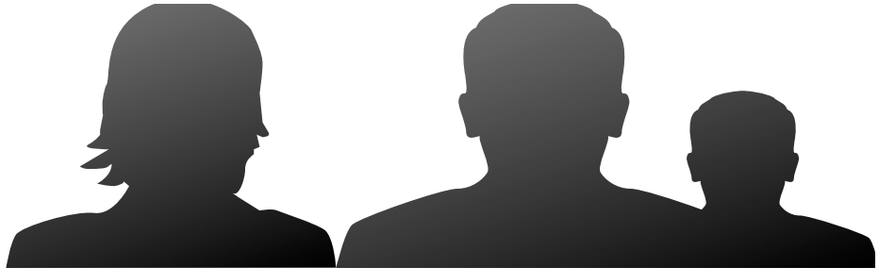
- Have all skills in the team



- Direct communication is better

Self Organization

- Team knows best how to solve problems

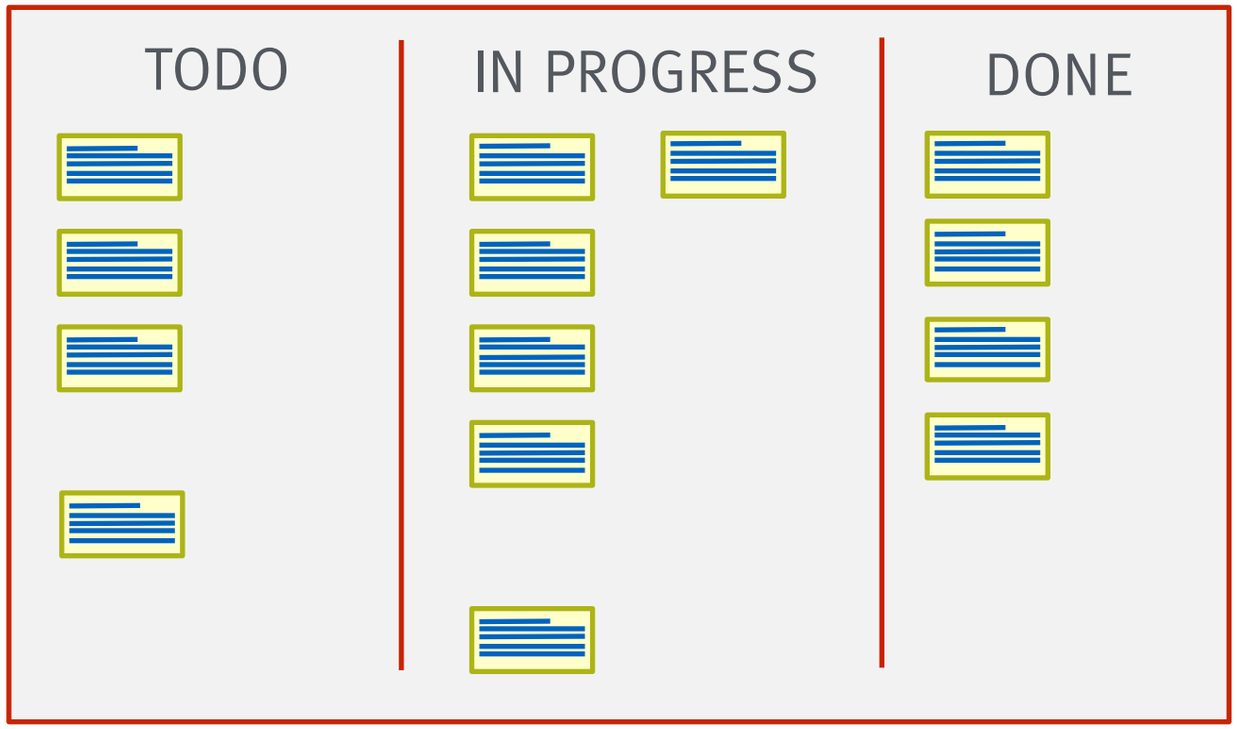


- Let them decide

Scaling

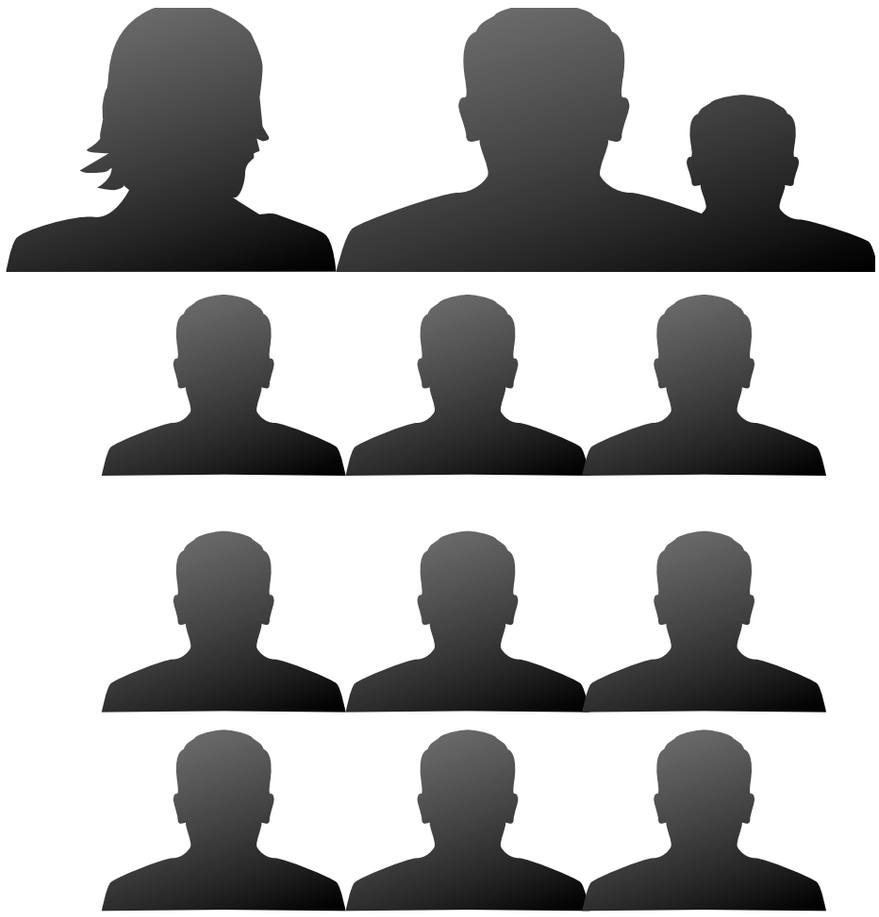
Scaling

- Do more
- Get more stories implemented
- ...and running in production



Scaling

- Do more
- Add more people
- Let the work in parallel
- Build more teams



Organizational Approaches

Scrum of Scrums

Scaled Agile Framework (SAFe)

Large Scale Scrum(LeSS)

Agile Path

Organizational Approaches

Scrum of Scrums

Scaled Agile Framework (SAFe)

Large-Scale Scrum (LeSS)

Agile Path

Not my expertise

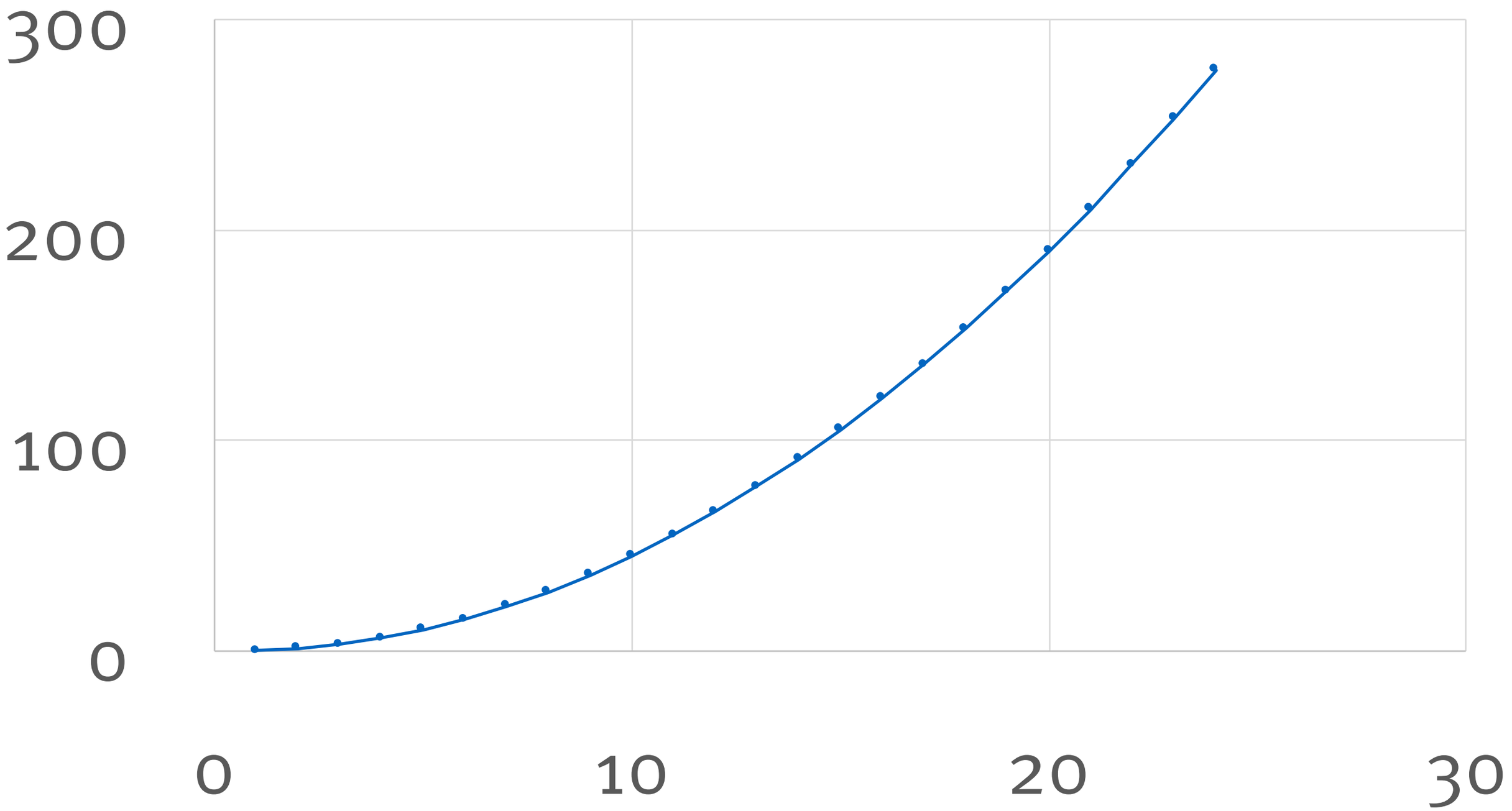
Communication

- Agile means communication
- Instead of written requirements
- ...story cards
- + direct communication

Persons vs. Potential Links

$$\frac{n * (n - 1)}{2}$$

Persons vs. Potential Links



Communication is great!

Need more persons

Depedencies and Coordination are Problems!

Dependencies

- Feature across teams
- Team must wait for other teams‘ results

3 sprints

Team Sprint

Team Sprint

Team Sprint

time

What Do You Communicate About?

Functionalities

Releases

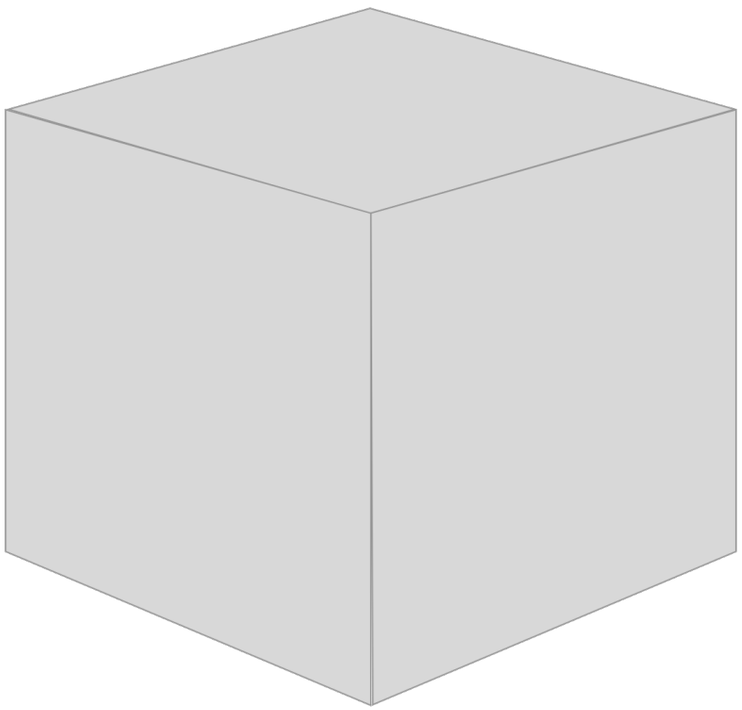
Technology and Architecture

Can we limit the need for communication?

Challenges for Scaling Agile

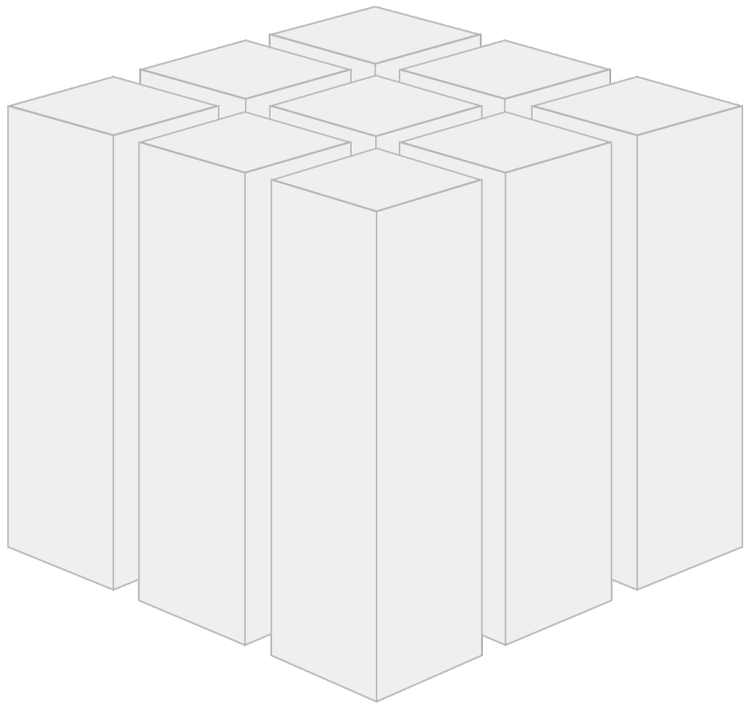
- Dependencies cause delays
- Too much communication about functionalities…
- …releasing software,
- …and technologies

Self-contained System

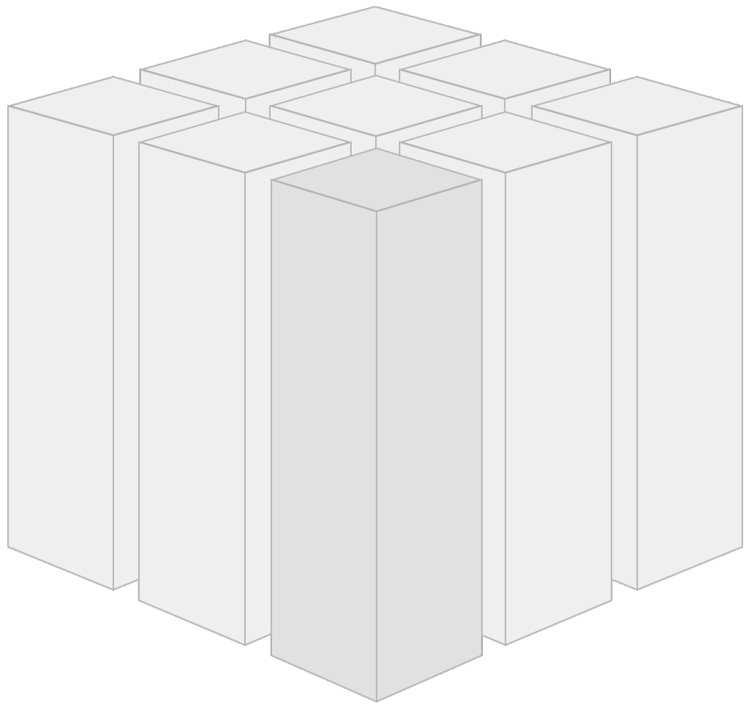


Deployment monolith

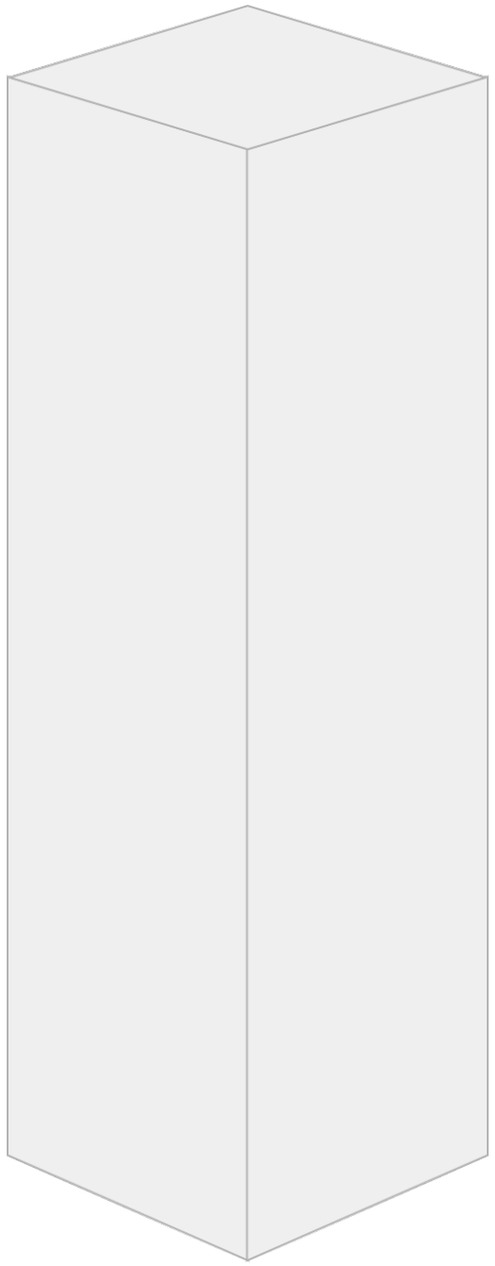
Graphics by Roman Stranghöhner, innoQ
http://scs-architecture.org



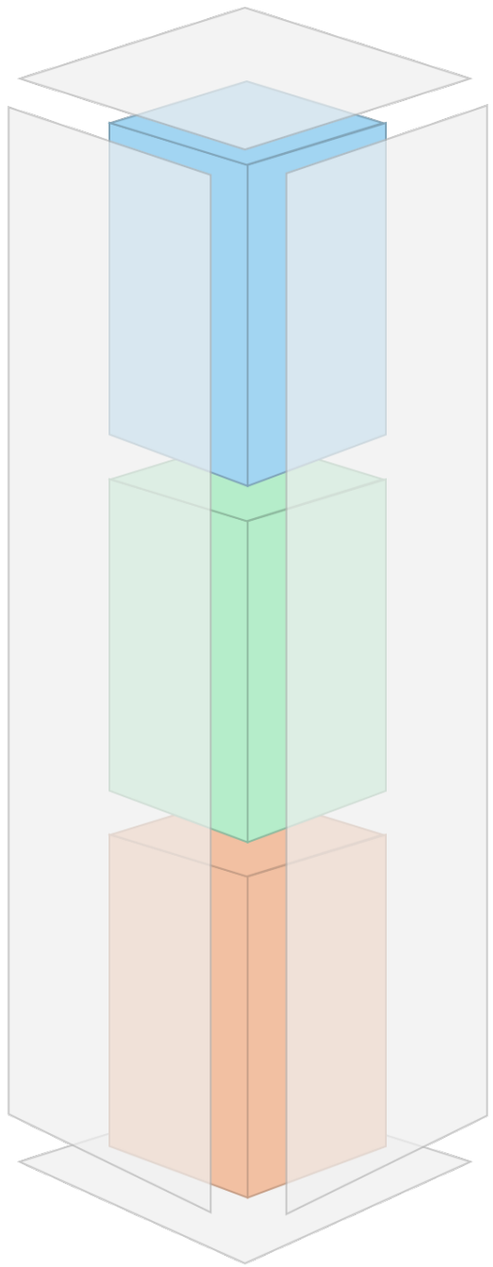
Cut Deployment monolith along domains …



... wrap domain in **separate** web application ...



Self-contained System (SCS) – individually deployable

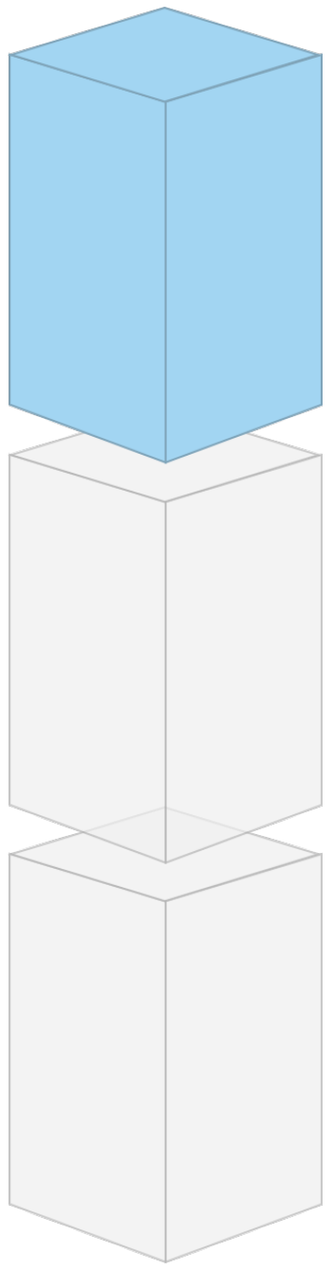


SCS =

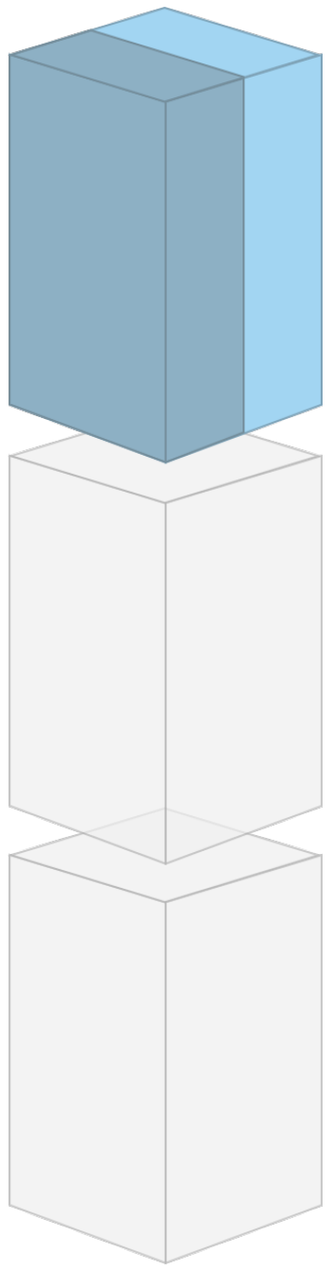
user interface+

business logic+

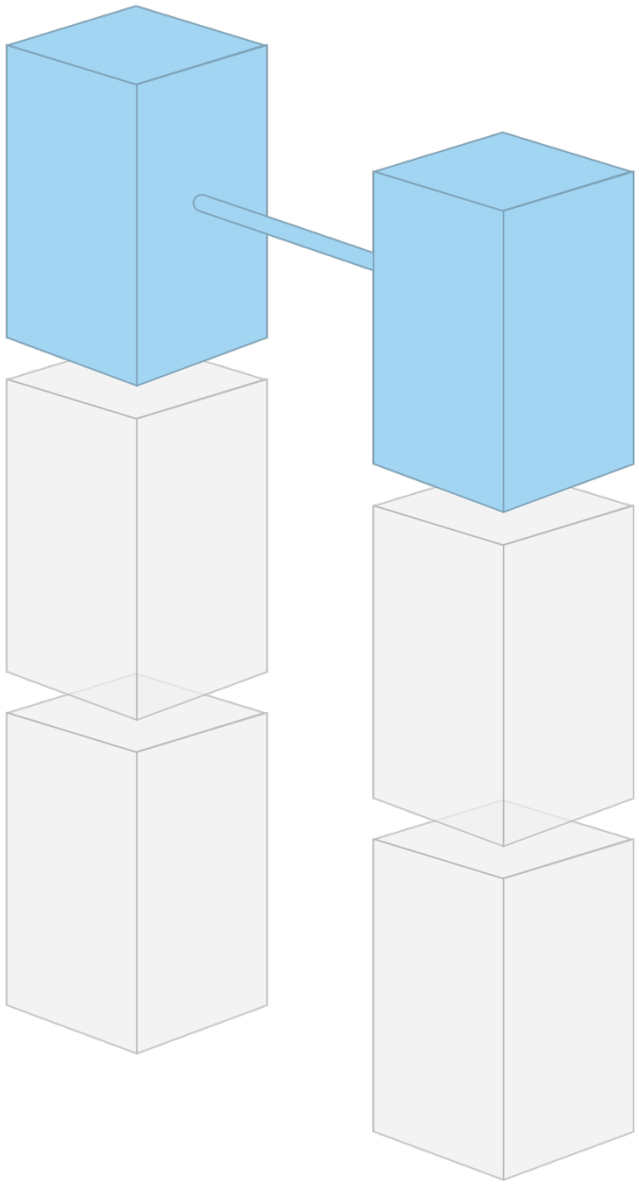
data storage



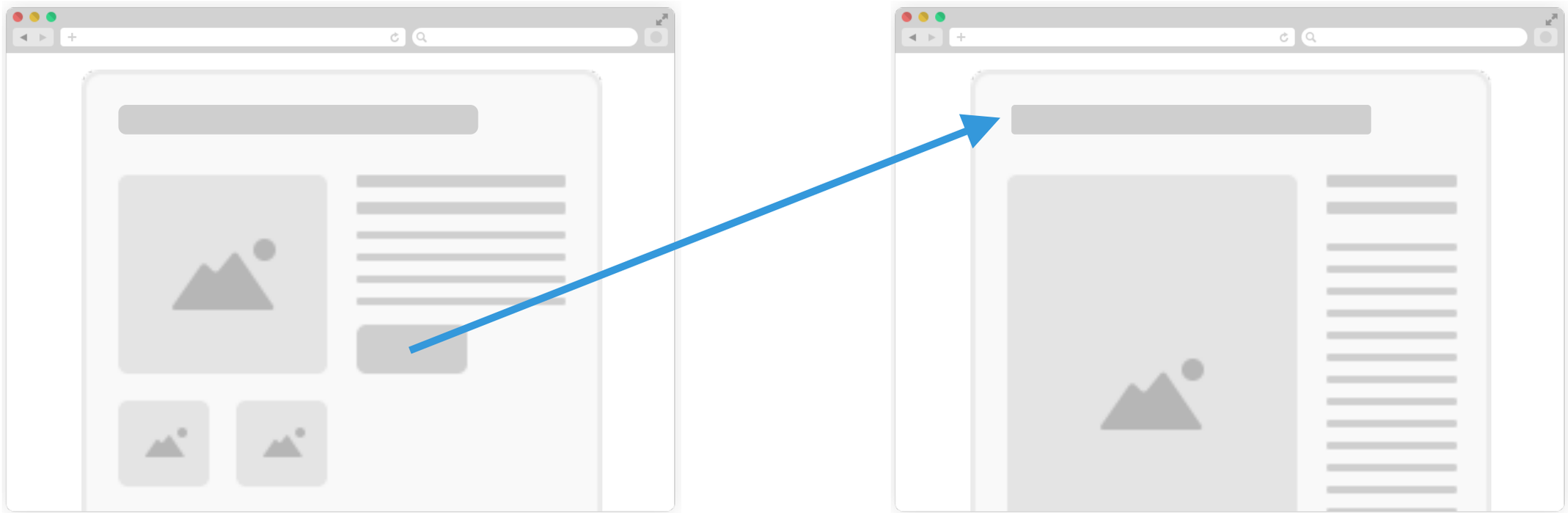
Web user interface



optional API e.g. for mobile

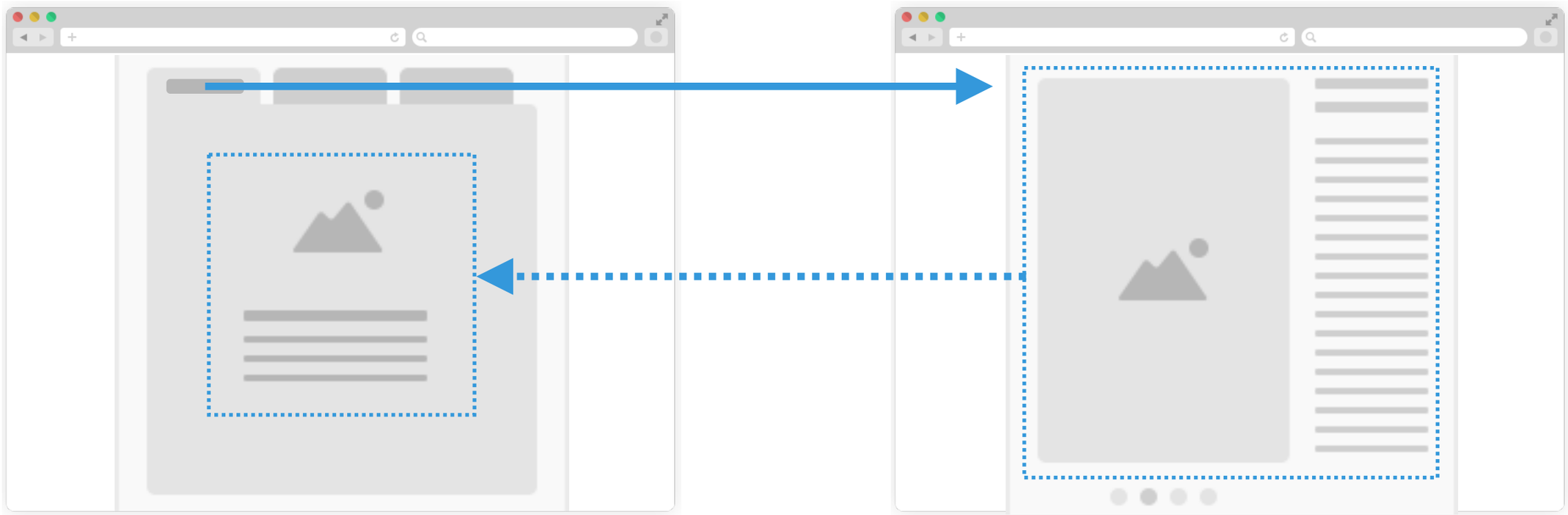


Self-contained Systems should be integrated in the **web interface**

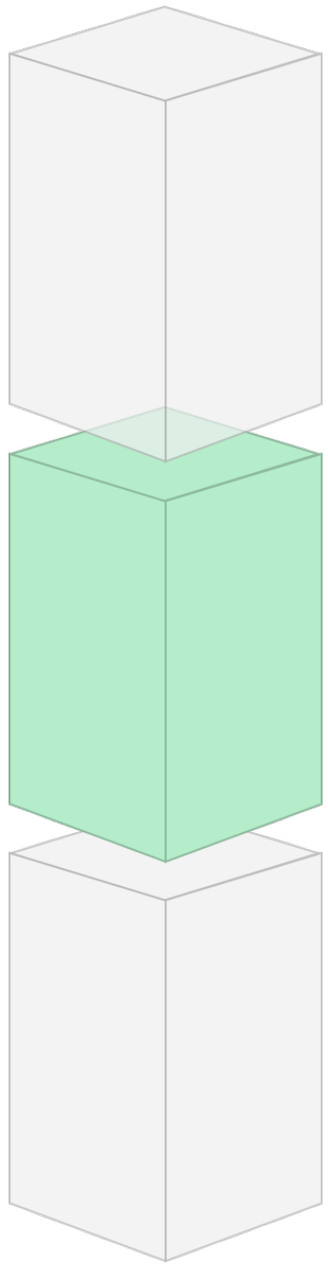


System 1 System 2

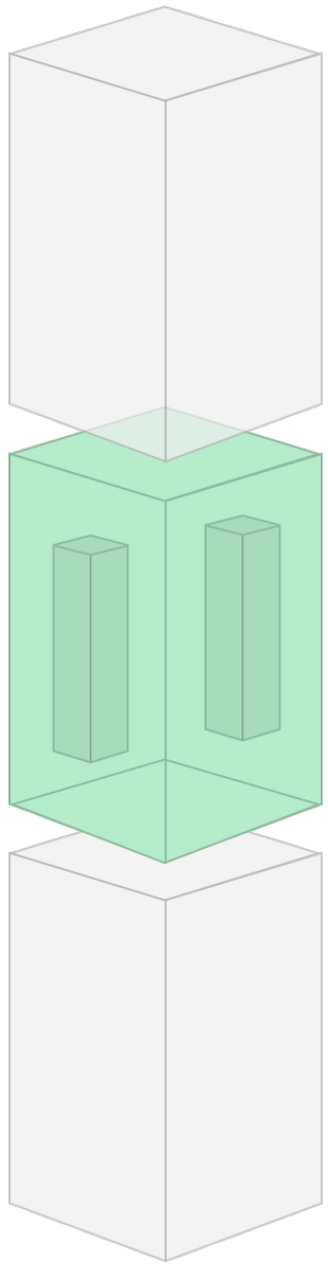
Hyperlinks to navigate between systems.



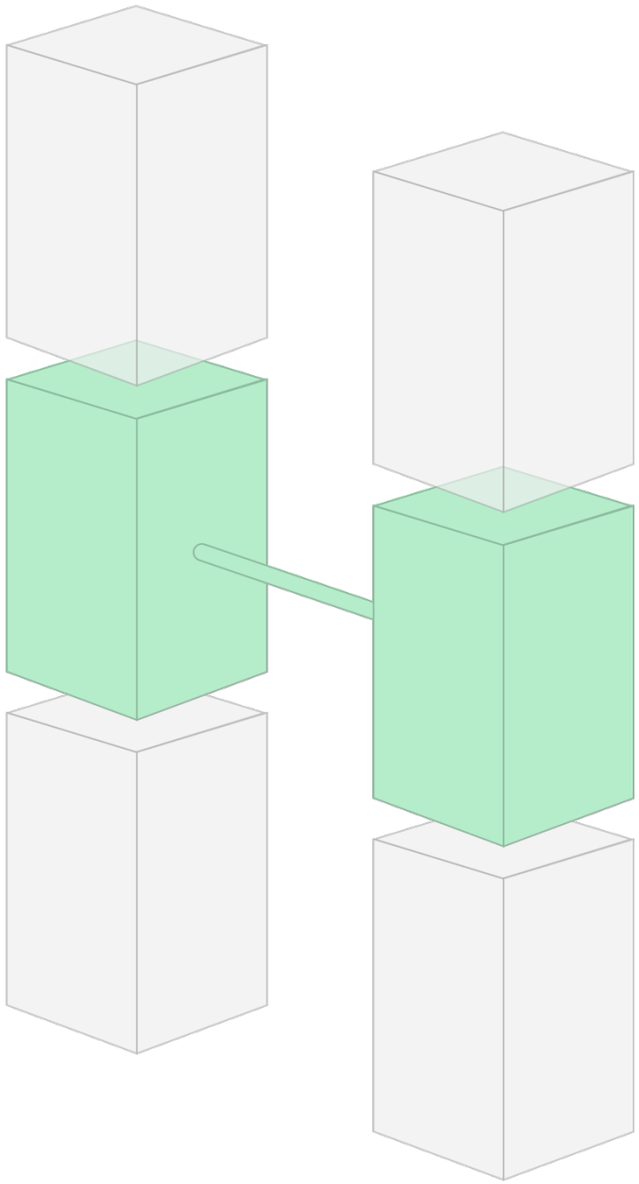
Dynamic inclusion of content served by another application



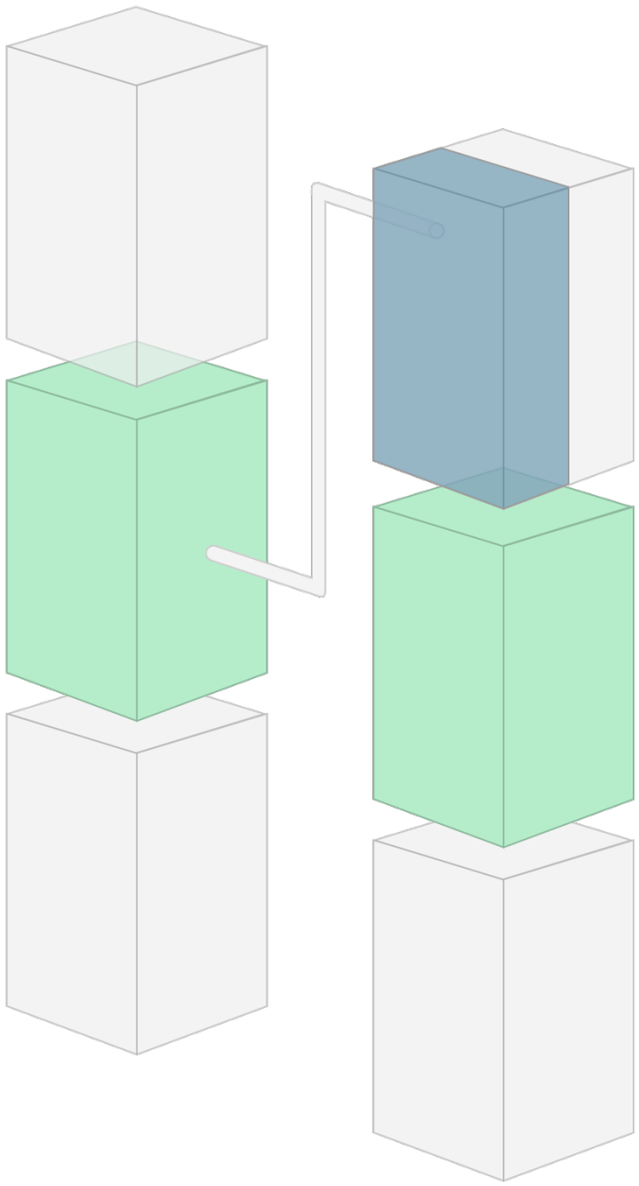
Logic only shared over a **well defined** interface.



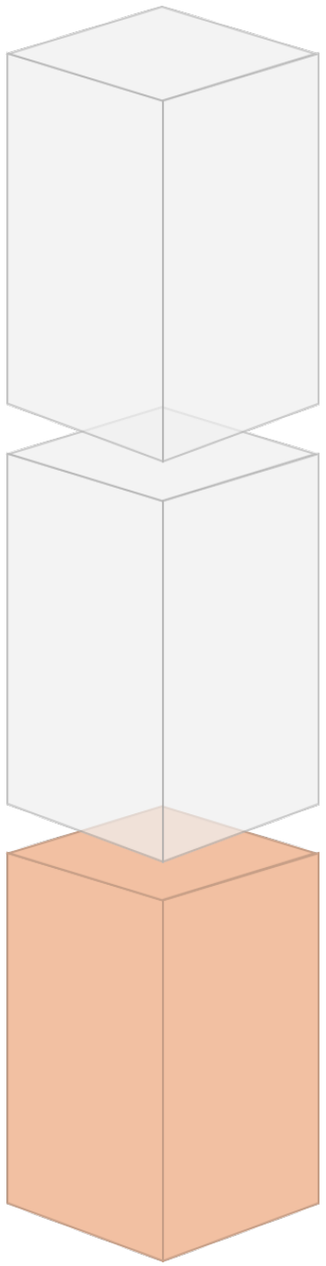
Business logic can consist of microservices



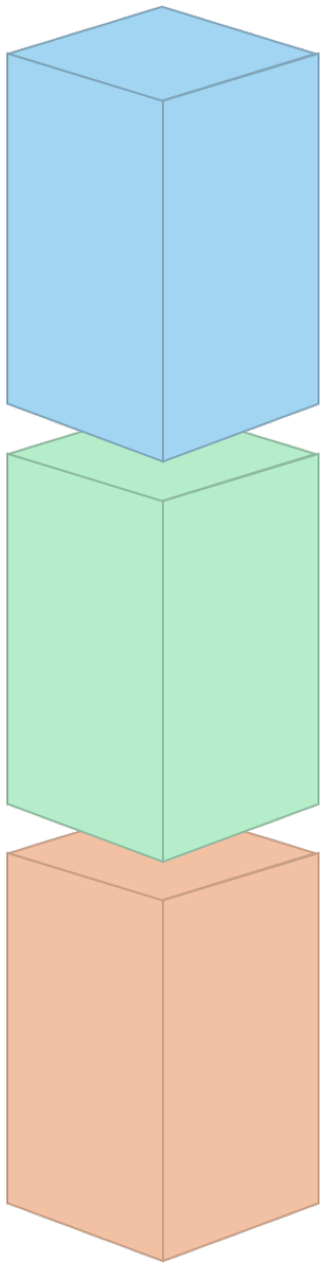
Avoid synchronous remote calls



Remote API calls should be asynchronous



Every SCS brings its **own data storage** with its own (potentially redundant) data



Technical decisions can be made independently from other systems (programming language, frameworks, tooling, platform)

Domained scoped SCS enables the development, operation and maintenance of an domain by a **single team**.

1 SCS

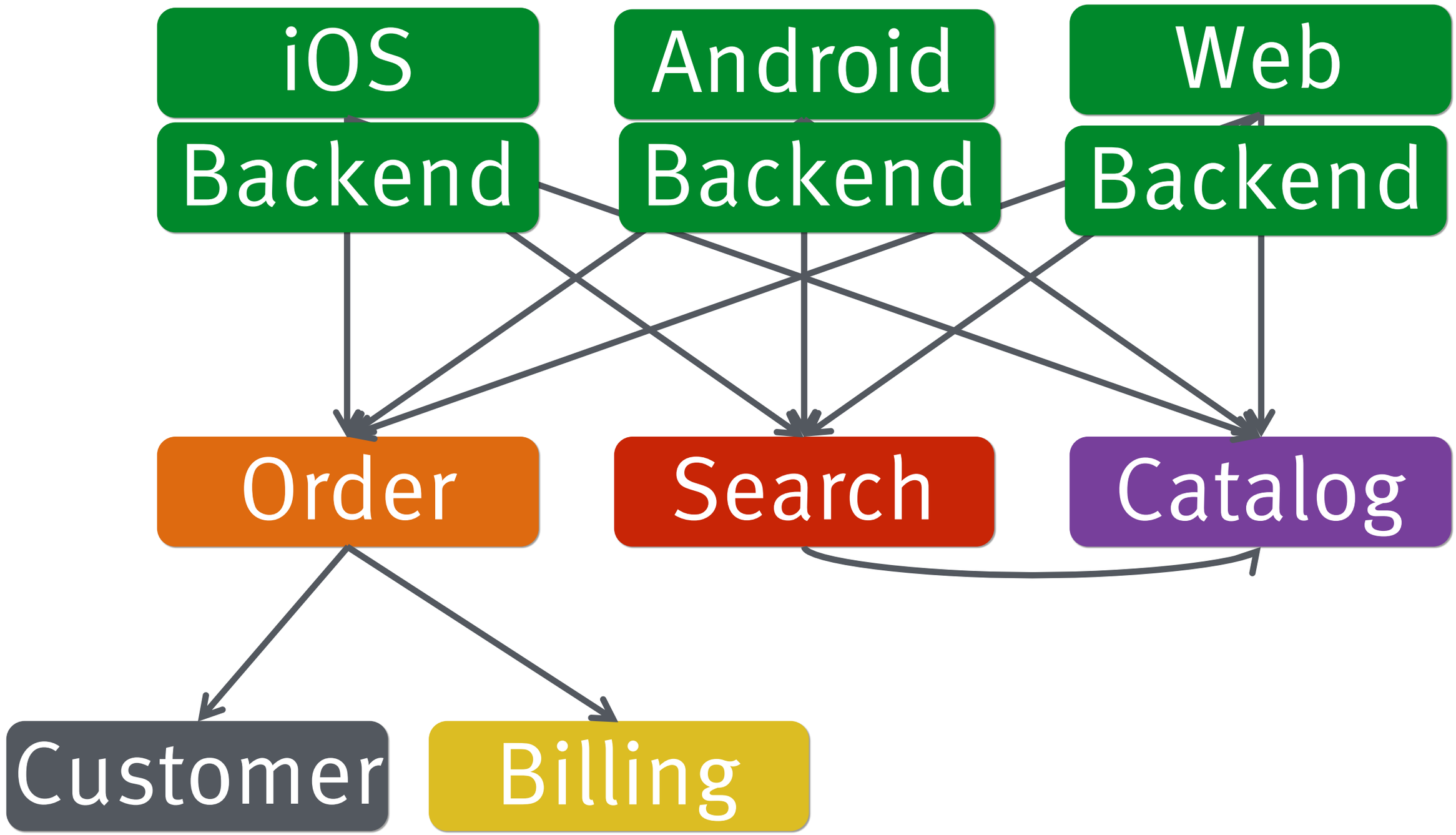
= 1 Domain

= 1 Web App

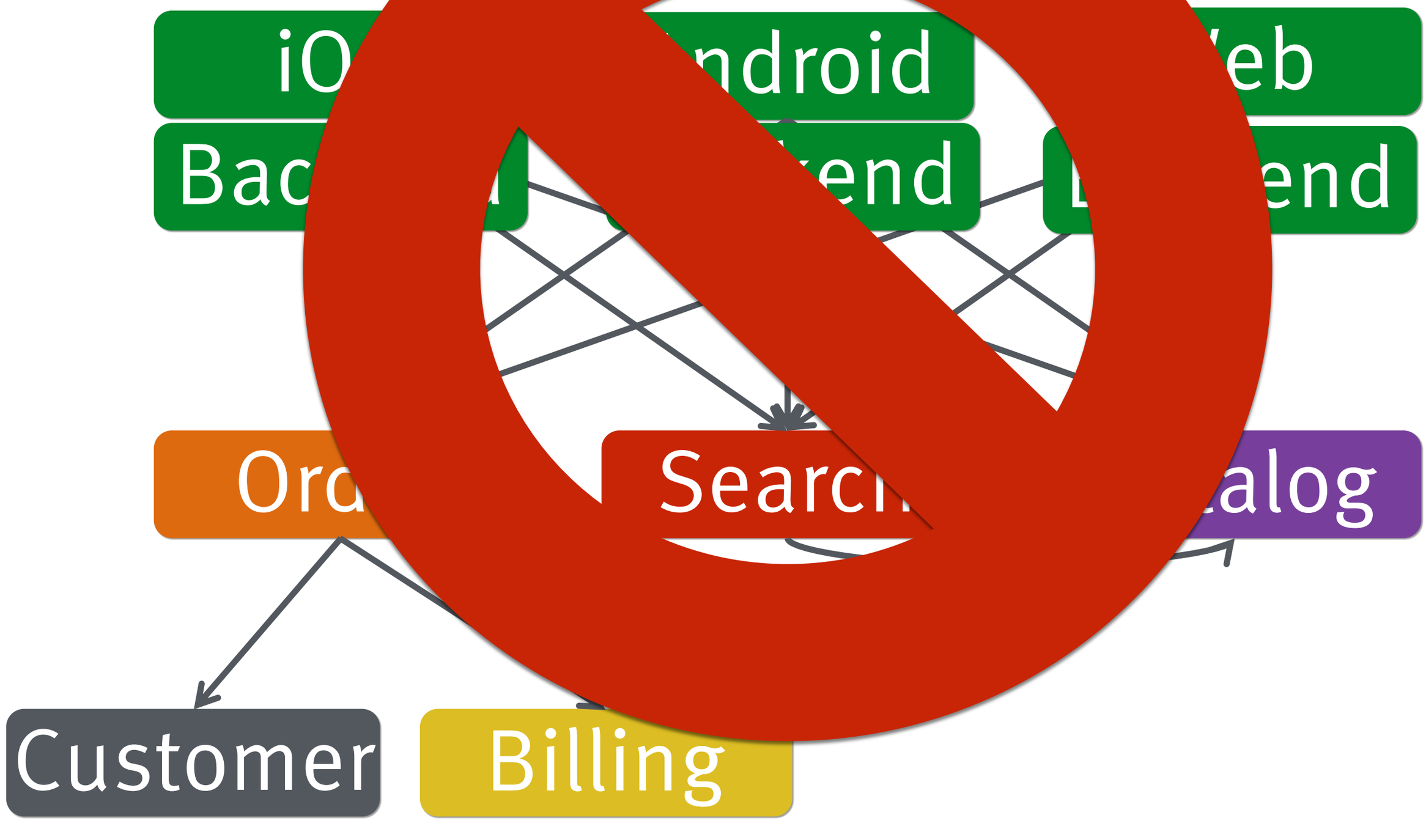
= 1 Team

= 1-n Microservices

SCS?


iOS
Backend
Android
Backend
Web
Backend
Order
Search
Catalog
Customer
Billing

SCS?



SCS to Scale Agile

Challenges for Scaling Agile

- Dependencies cause delays
- Too much communication about functionalities…
- …releasing software,
- …and technologies

Challenges for Scaling Agile

- Dependencies cause delays
- Too much communication about functionalities…
- …releasing software,
- …and technologies

Conway‘s Law

Architecture

copies

communication structures

of the organization

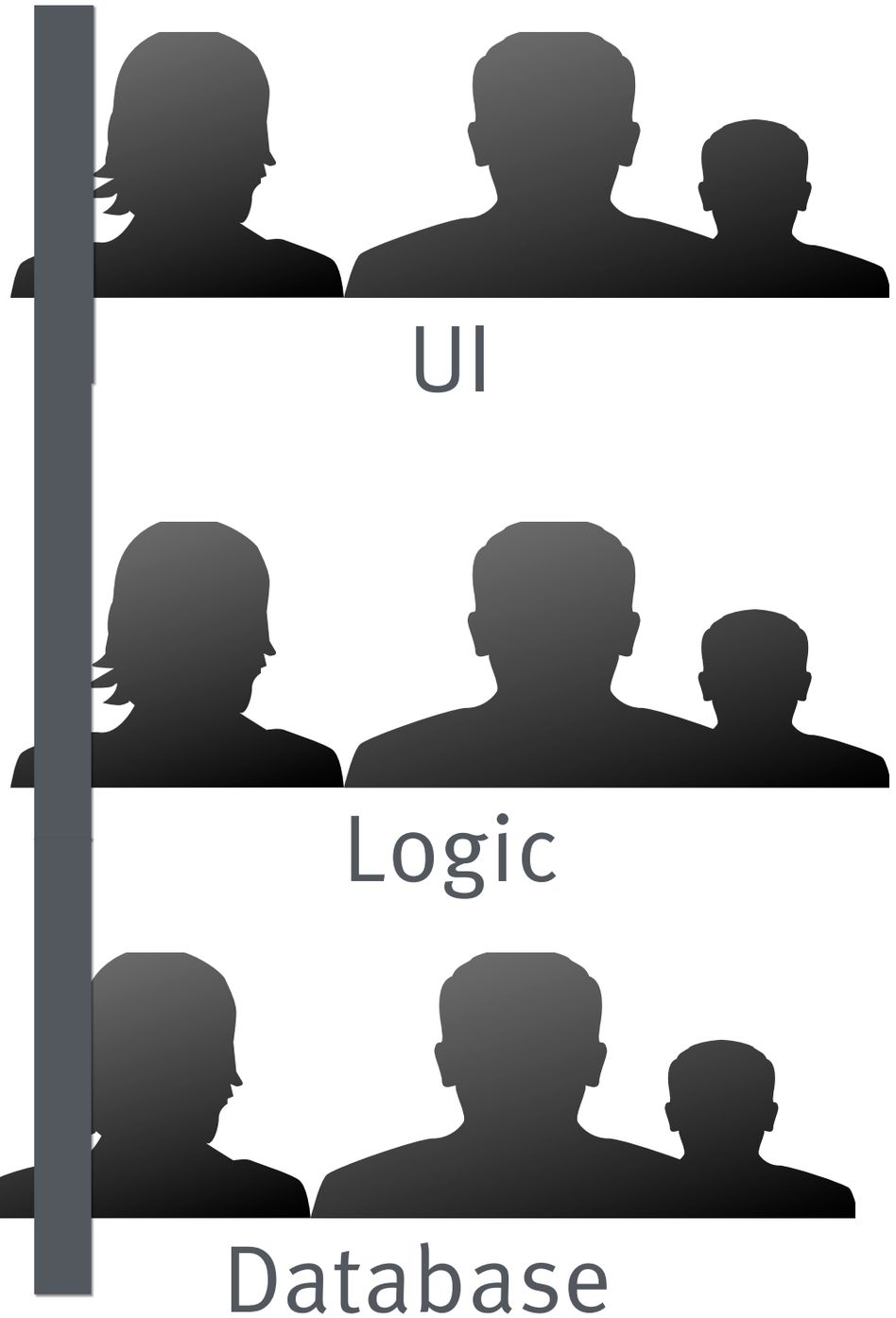
Conway‘s Law: Impact

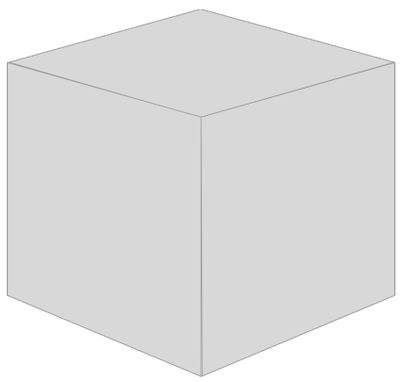
Architecture
and
communication structures
in the organization
are the same thing.

Conway's Law as a Limit

- Organization drives architecture
- Teams of experts
- i.e. UI, logic & database team
- Three technical artifacts

Change Order Process!





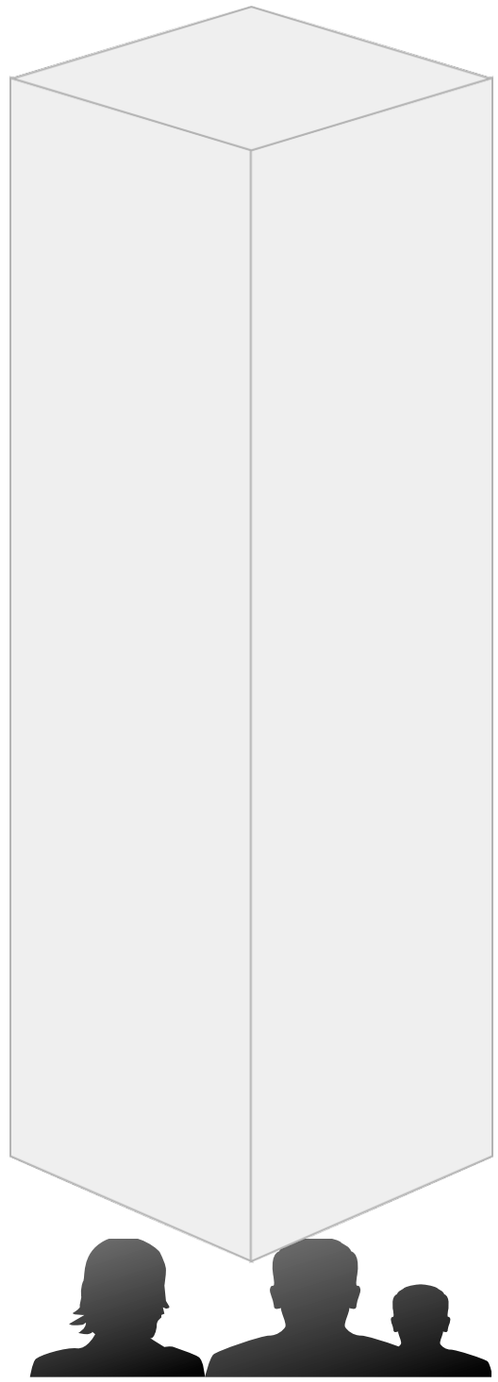
3 sprints

DB
Team Sprint

Logic
Team Sprint

GUI
Team Sprint

time



Order Team =
UI+Logic+Database

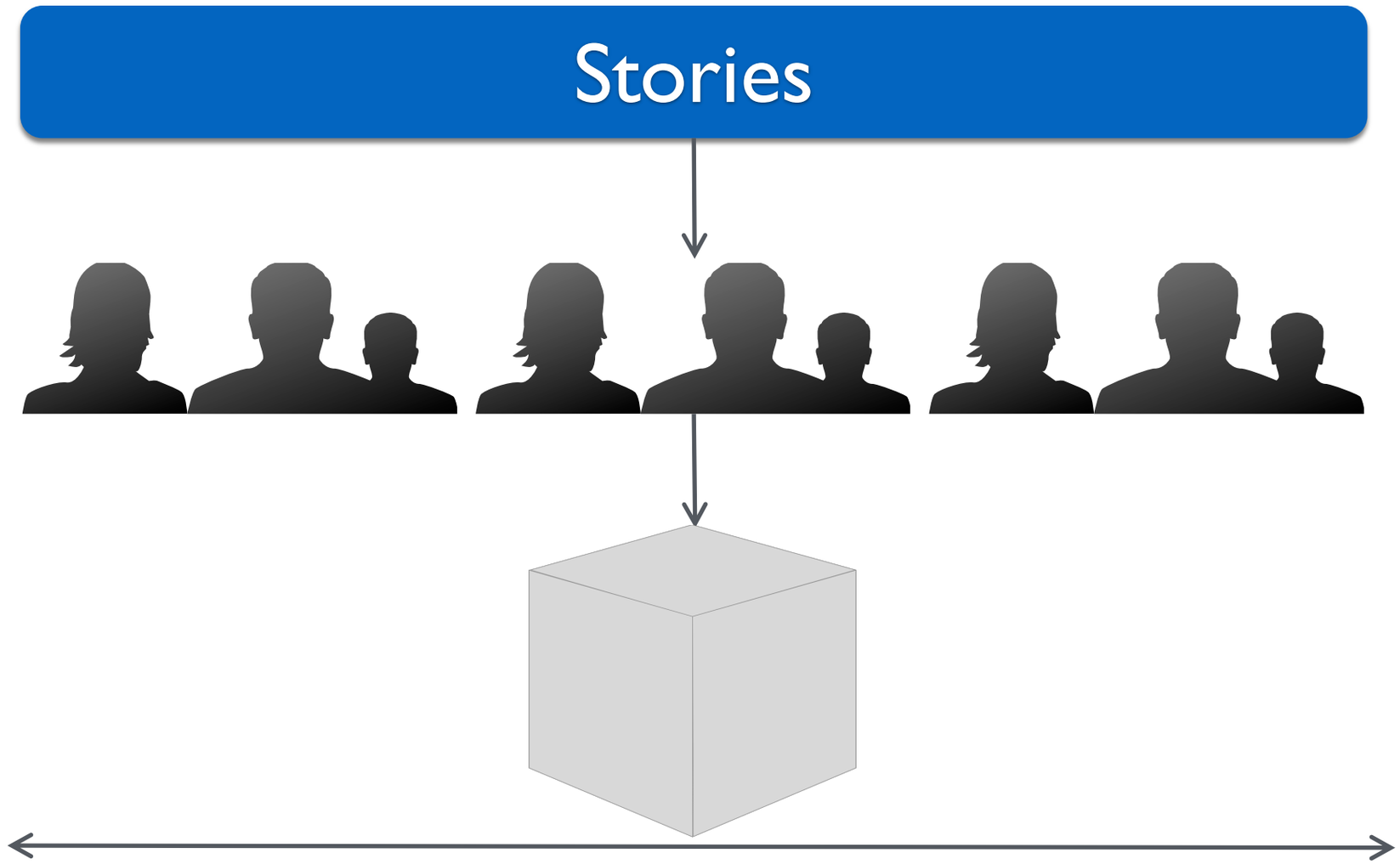
Domained scoped SCS enables the development of a domain by a **single team**

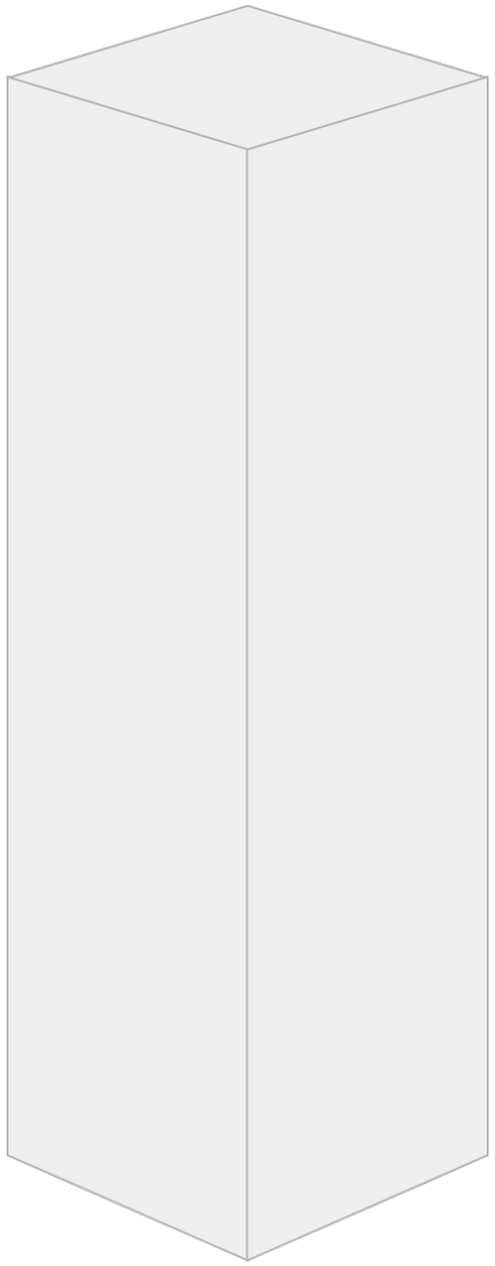
– no coordination

Challenges for Scaling Agile

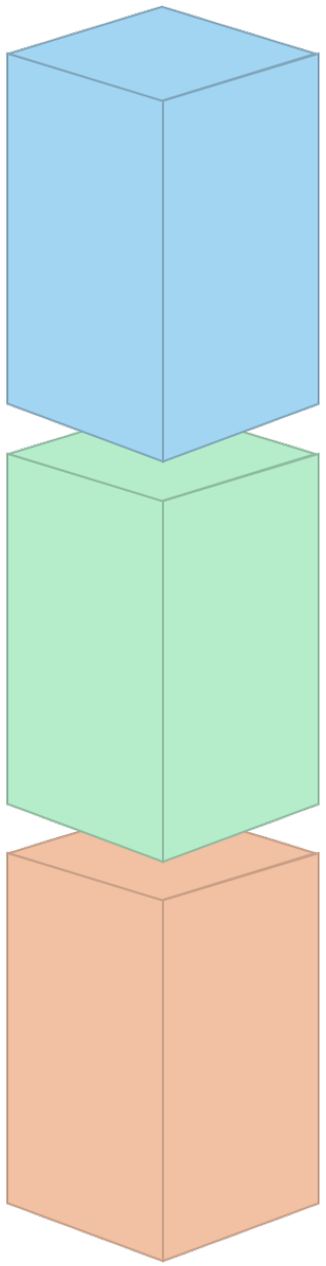
- Dependencies cause delays
- Too much communication about functionalities…
- …releasing software,
- …and technologies

Deployment Monolith



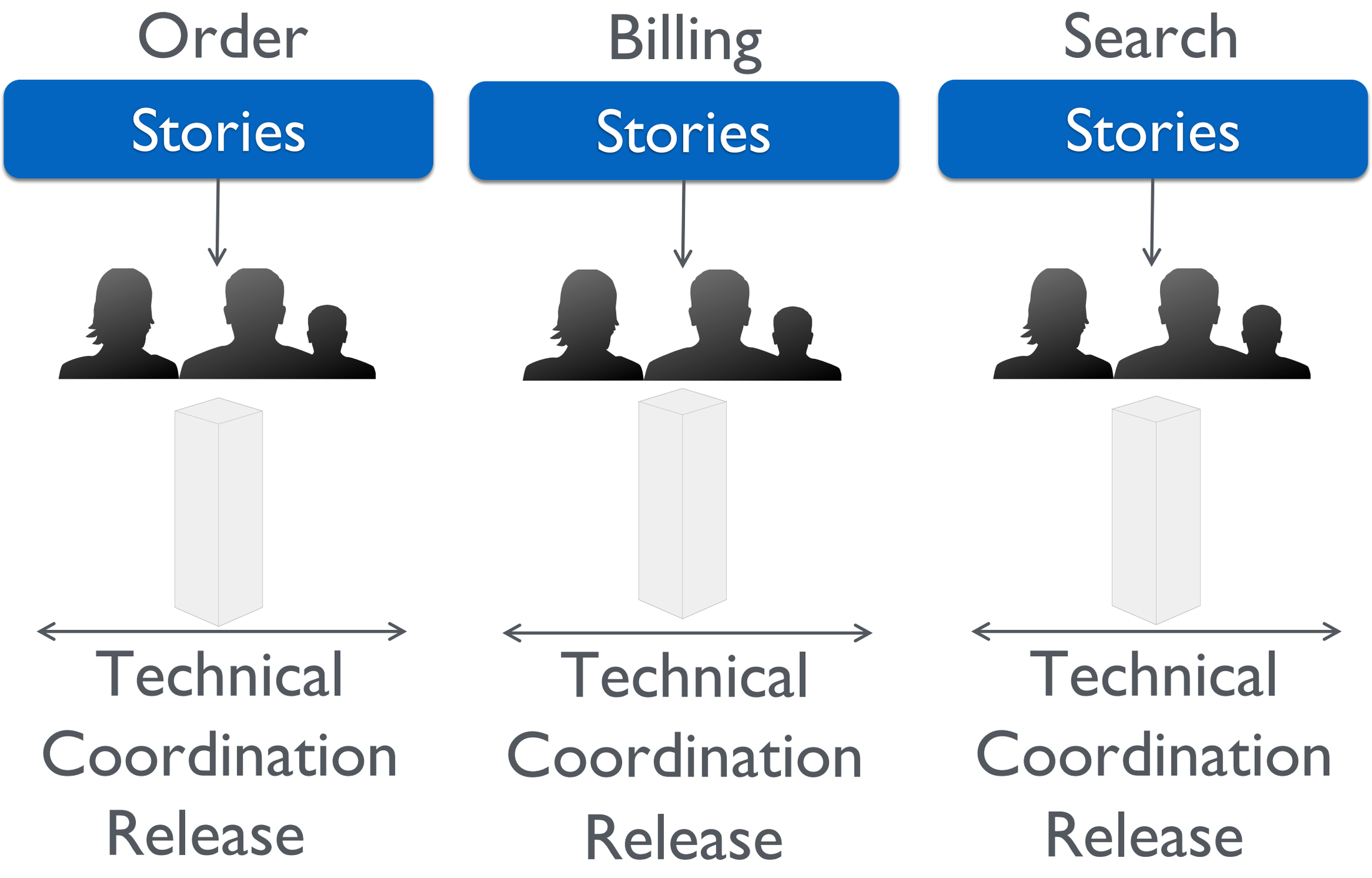


Self-contained System (SCS) – individually deployable



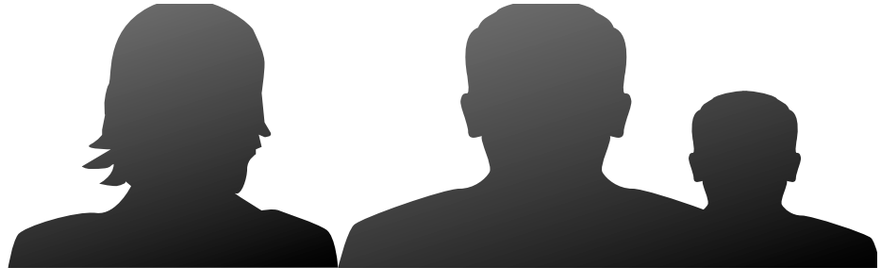
Technical decisions can be made independently from other systems (programming language, frameworks, tooling, platform)

SCS


Order
Billing
Search
Stories
Stories
Stories
Technical Coordination Release
Technical Coordination Release
Technical Coordination Release

Impact on Teams

- More self-organization
- Decide about technologies, releases etc.



- Need to be more cross-functional
- E.g. ops skills become more important

Conclusion

How to scale agile?

Define architecture to limit communication needs

Conclusion: SCS & Microservices

- Microservices have many advantages

- SCS are a way to use Microservices
- ...for large projects
- ...to scale agile

Conclusion: SCS & Agility

- Domain in one SCS
  - less dependencies
  - less delays
  - less communication about functionalities
- Technological freedom
  - less communication about technologies
- Independent releases
  - no need to coordinate releases

Challenges

Challenges

- UI integration
- ...in particular for mobile / Single Page App
- Architecture more important
- Architecture = organization

Meta-Conclusion

How To Think About Architecture

- Process has an impact on architecture
- Software architecture & organization are the same

Thank You!

@ewolff

http://scs-architecture.org

(Creative Commons)