LONDON
INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE

goto; conference

One size does not fit all

Stefan Tilkov

@stilkov

innoQ

GOTO London 2016

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

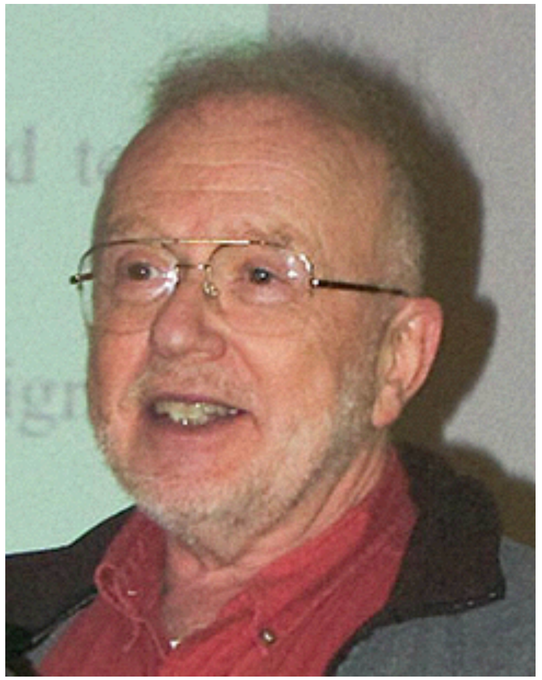
How big shall each individual piece be?

Just make things the *right* size

Separate separate things

Join things that belong together

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

Separation of concerns

“Let me try to explain to you, what to my taste is characteristic for all intelligent thinking. It is, that one is willing to study in depth an aspect of one's subject matter in isolation for the sake of its own consistency, all the time knowing that one is occupying oneself only with one of the aspects. […] ***It is what I sometimes have called "the separation of concerns", which, even if not perfectly possible, is yet the only available technique for effective ordering of one's thoughts, that I know of. This is what I mean by "focussing one's attention upon some aspect": it does not mean ignoring the other aspects, it is just doing justice to the fact that from this aspect's point of view, the other is irrelevant.*** *It is being one- and multiple-track minded simultaneously.”*

Edsger W. Dijkstra, 1974

http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD447.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

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

Cohesion in OO: Object Calisthenics

1. One level of indentation per method
2. Don’t use the ELSE keyword
3. Wrap all primitives and strings
4. First class collections
5. One dot per line
6. Don’t abbreviate
7. Keep all entities small
8. No classes with more than two instance variables.
9. No getters/setters/properties
10. No static methods other than factory methods

Jeff Bay, 2008 – http://www.cs.helsinki.fi/u/luontola/tdd-2009/ext/ObjectCalisthenics.pdf

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

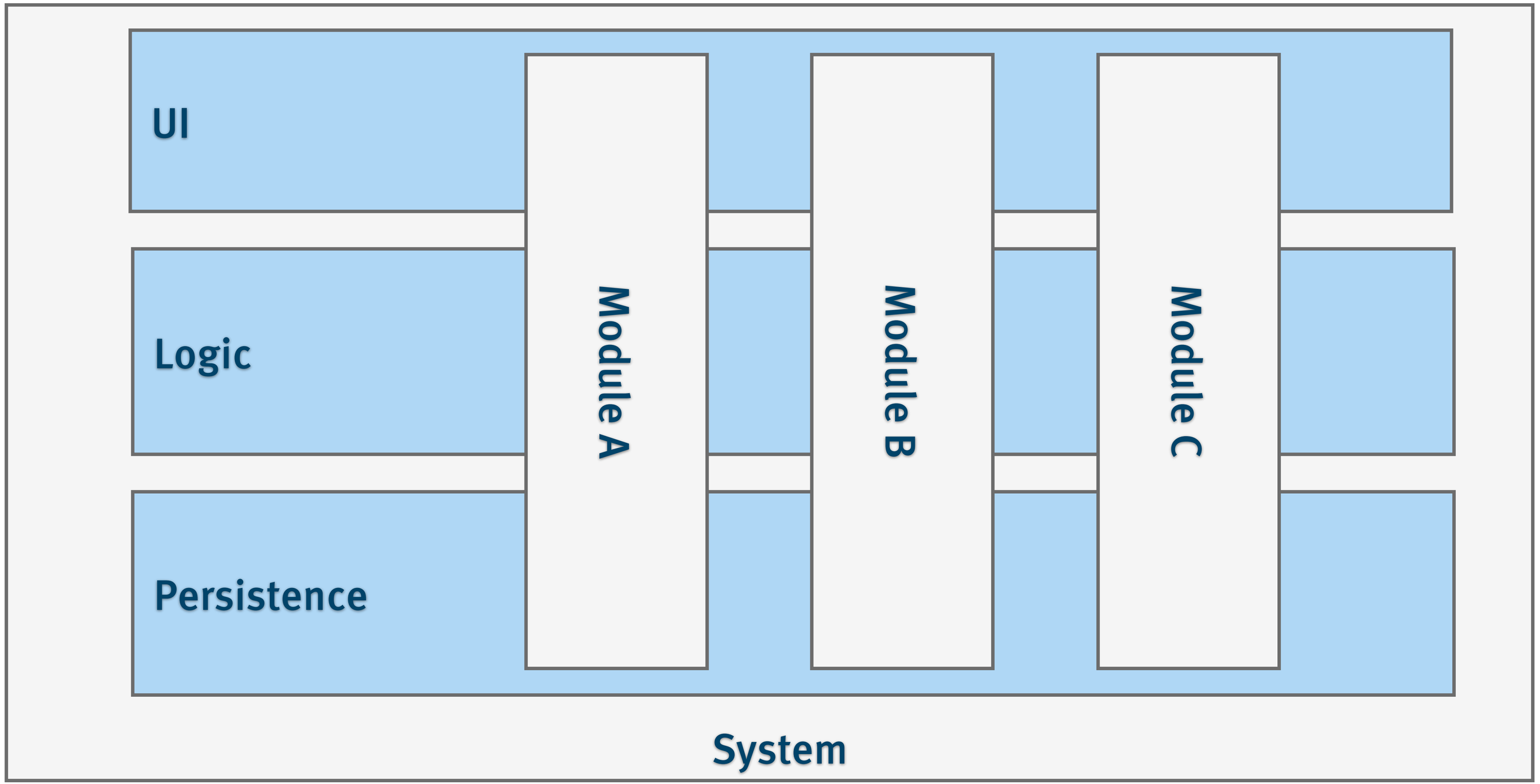
Domain dependencies

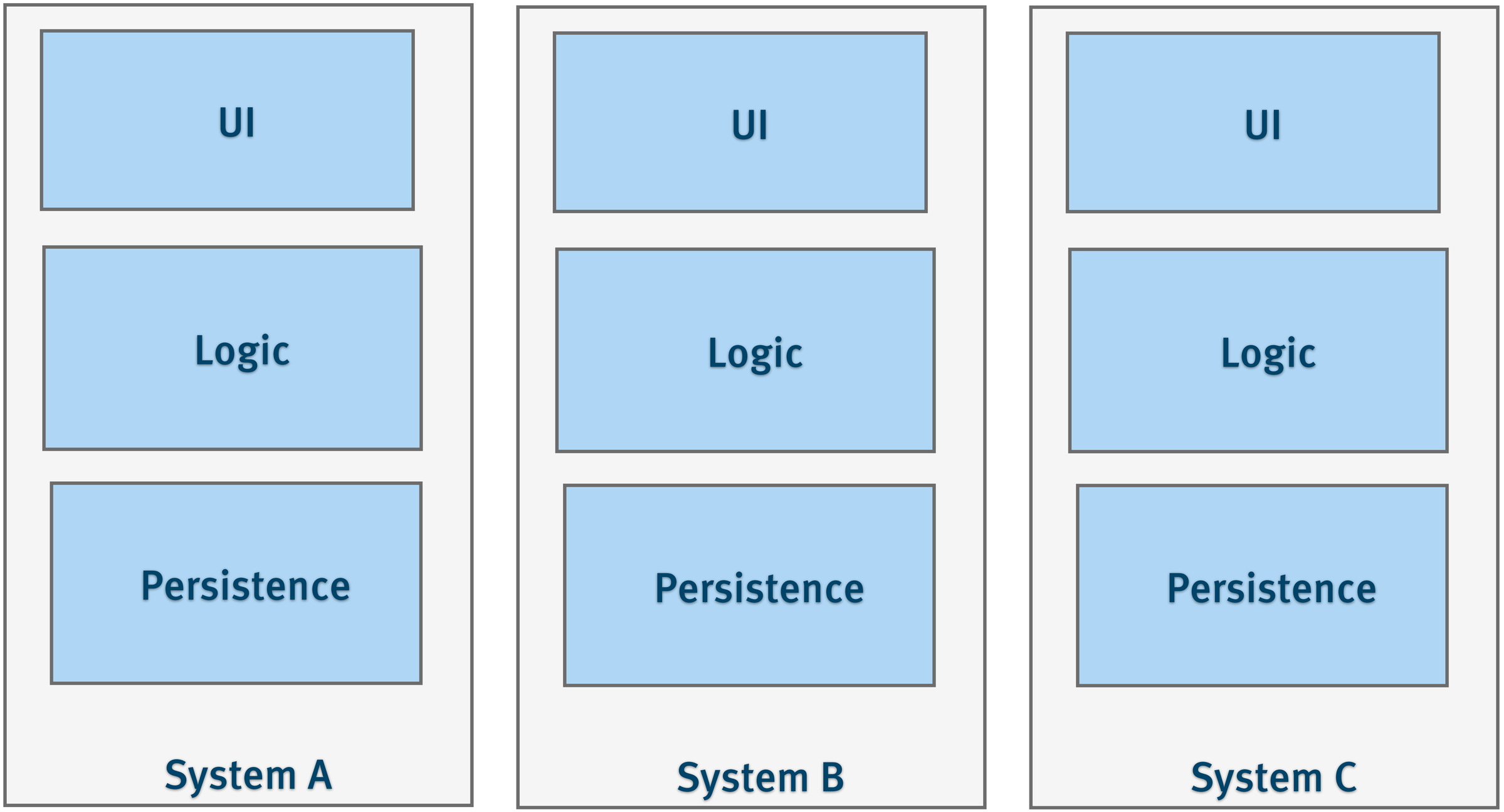
Parallel/isolated runtime

Implementation

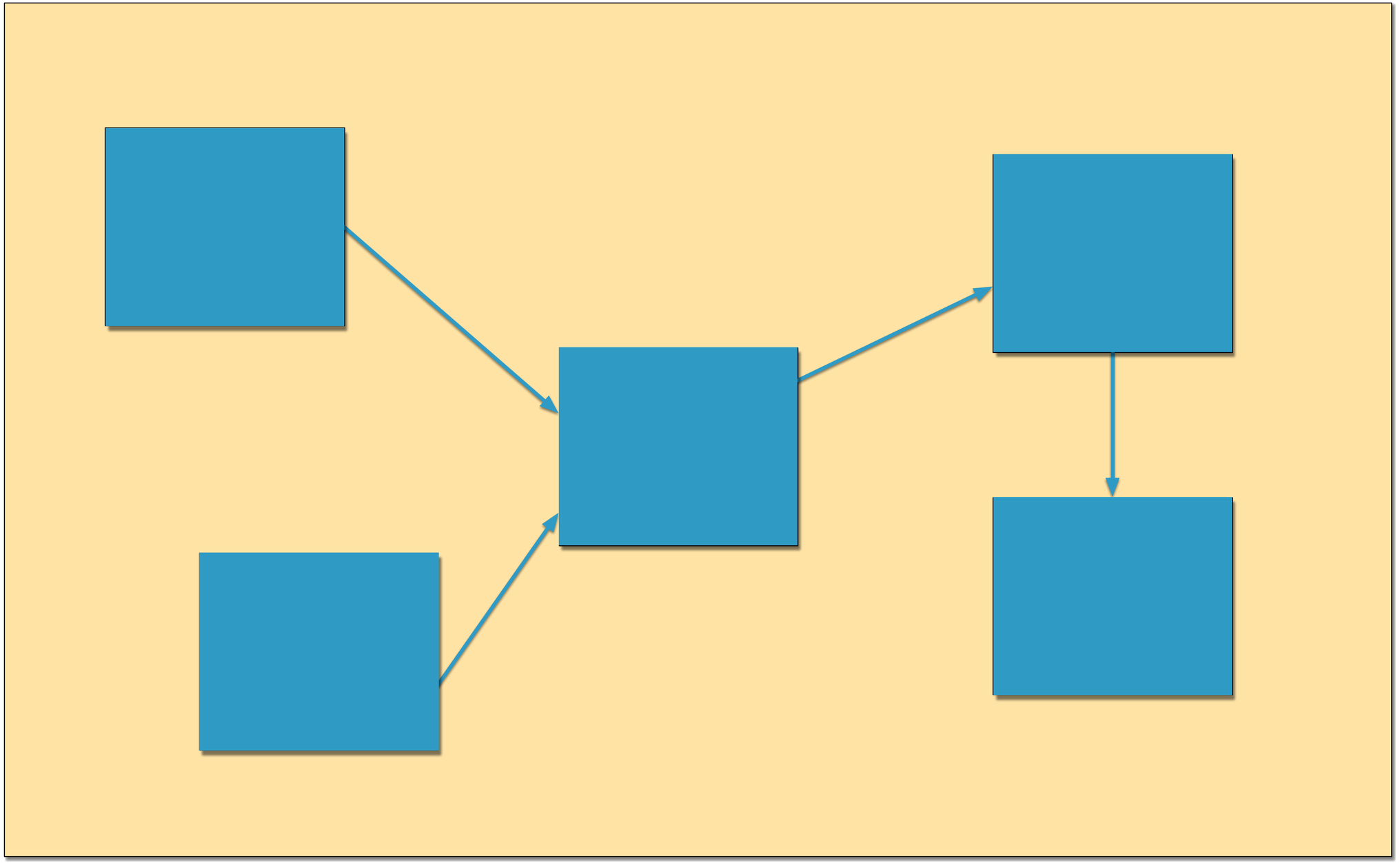
Parallel/isolated development

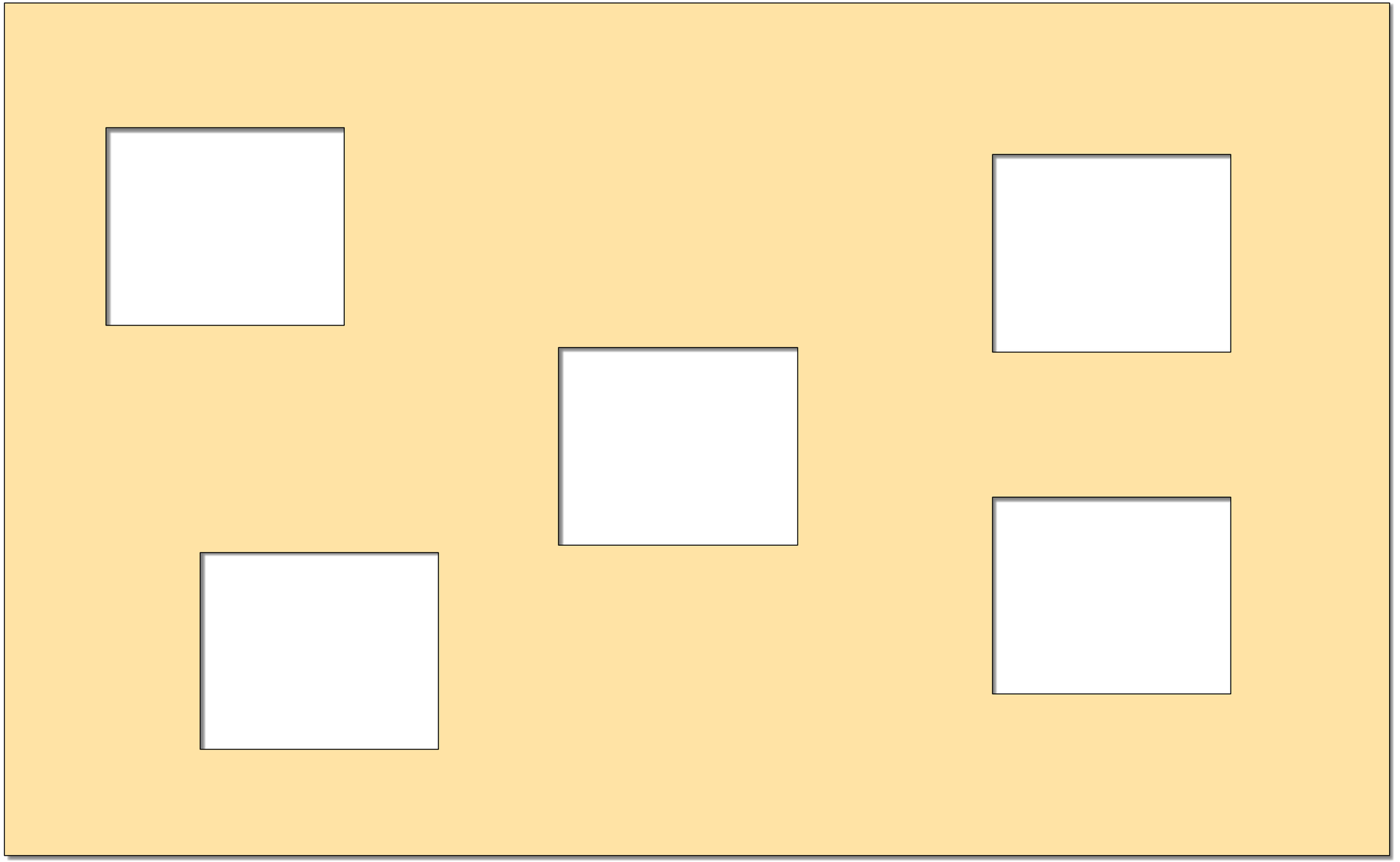
Multiple Dimensions
Different Priorities


UI
Logic
Persistence
Module A
Module B
Module C
System


UI
Logic
Persistence
System A
UI
Logic
Persistence
System B
UI
Logic
Persistence
System C

Environments





Environments

Language runtimes

Supervisors

Operating Systems

Container Hosts

Hardware

Application servers

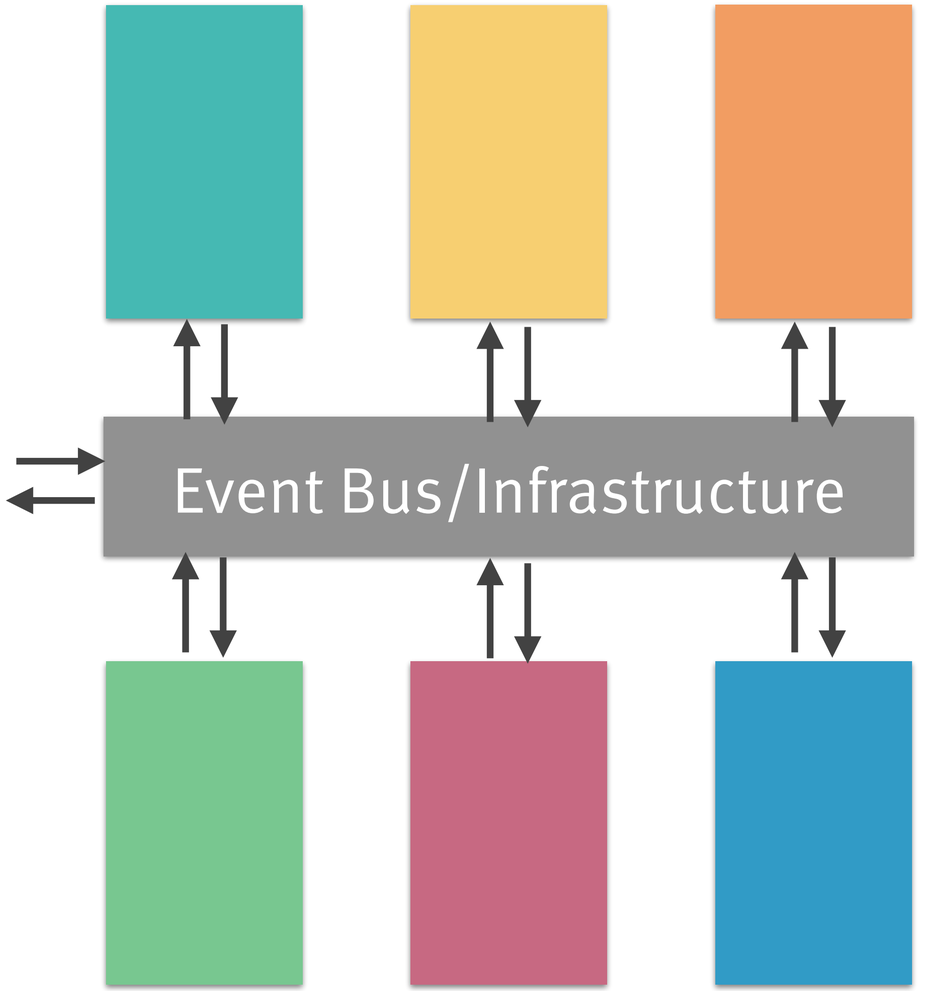
Let’s talk about Microservices

Microservices – Common Traits

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

Example: Pricing Engine

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



Super-small, really micro, nano

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

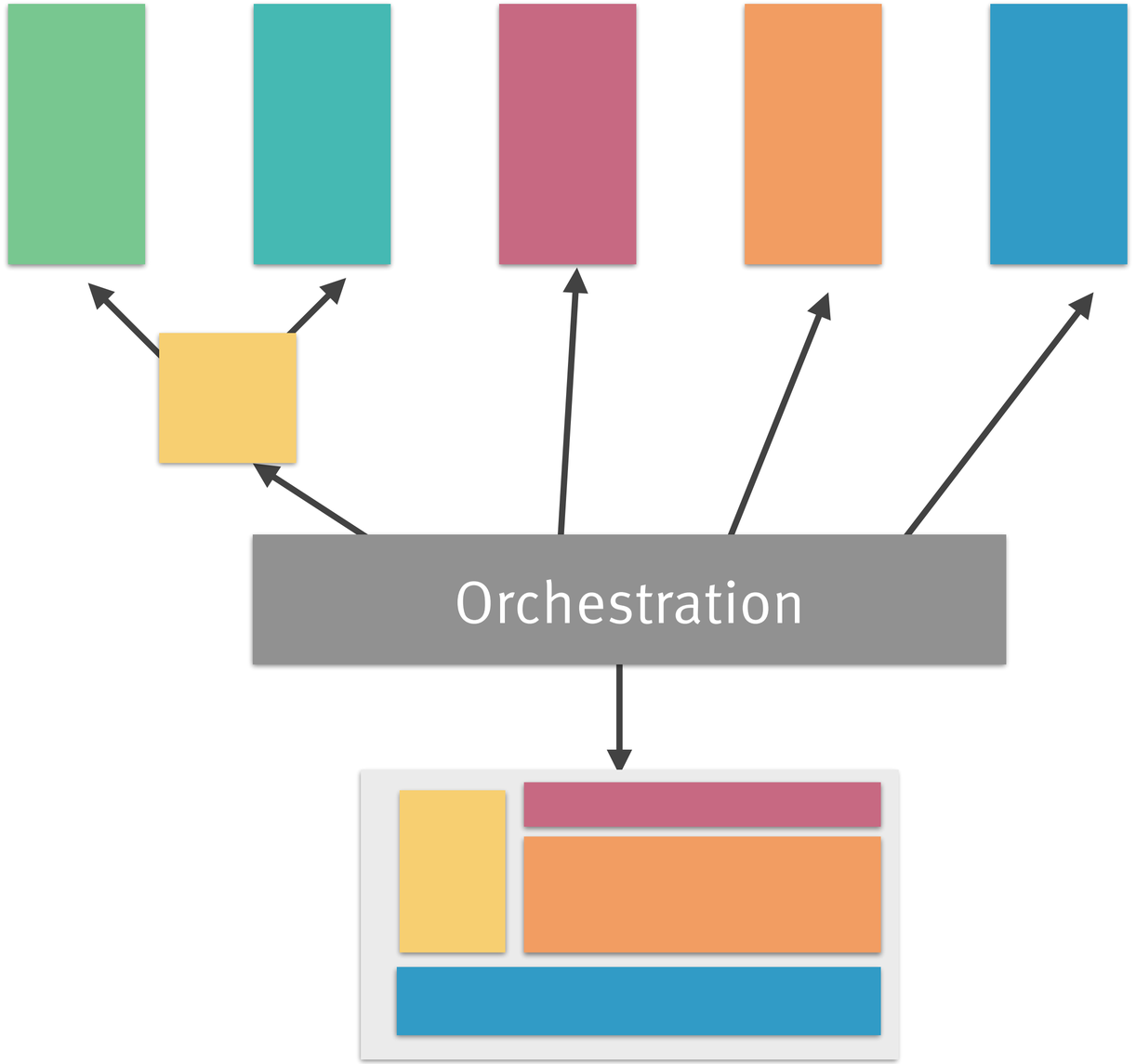
Super-small, really micro, nano

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



Small, micro

Characteristics:

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

As seen on:

- Netflix
- Twitter
- Gilt

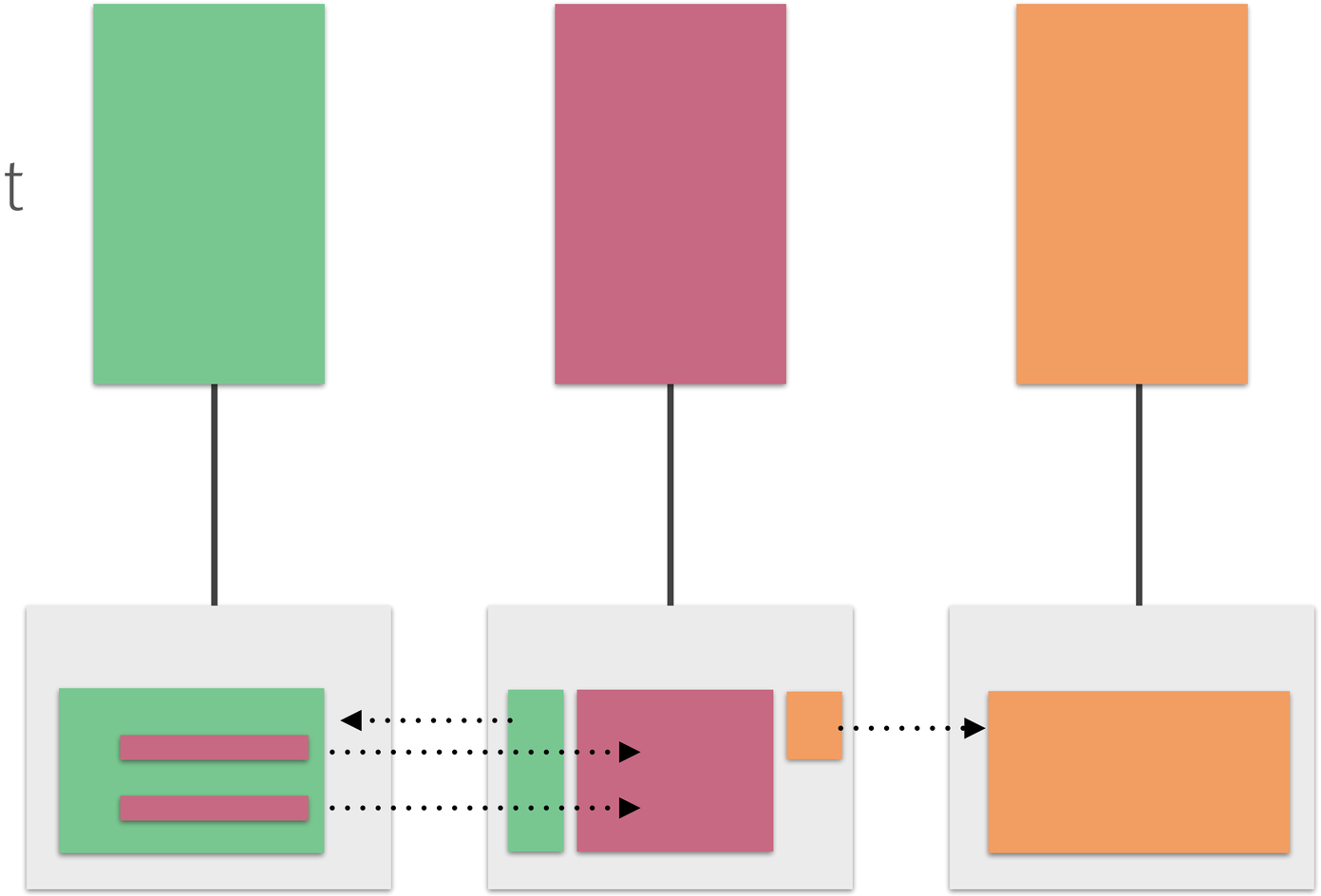
Small, micro

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

Example: E-Commerce Site

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



Medium-sized

Characteristics:

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

As seen on:

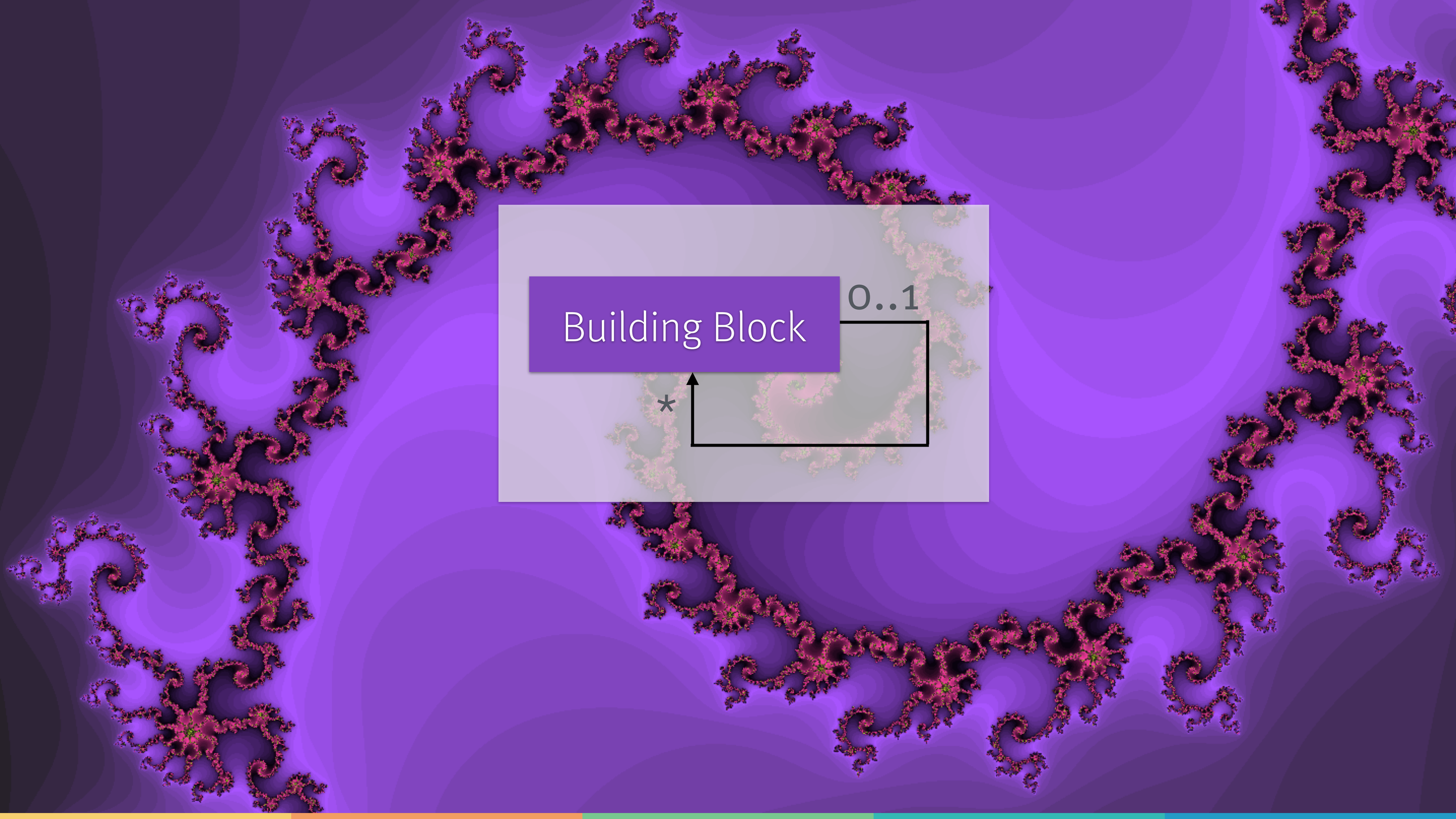
- Amazon
- Groupon
- Otto.de
- Self-contained systems (SCS)(*)

(*) https://scs-architecture.org

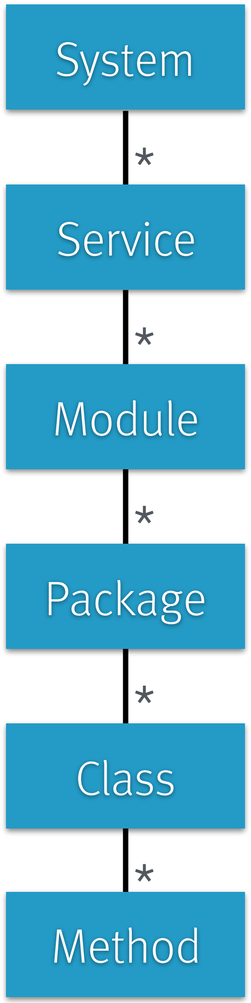
Medium-sized

Consequences:

- Larger, independent systems
- Including data + UI (if present)
- Able to autonomously serve requests
- Light-weight integration, ideally via front-end
- Well suited if goal is decoupling of development teams


Building Block
0..1
*

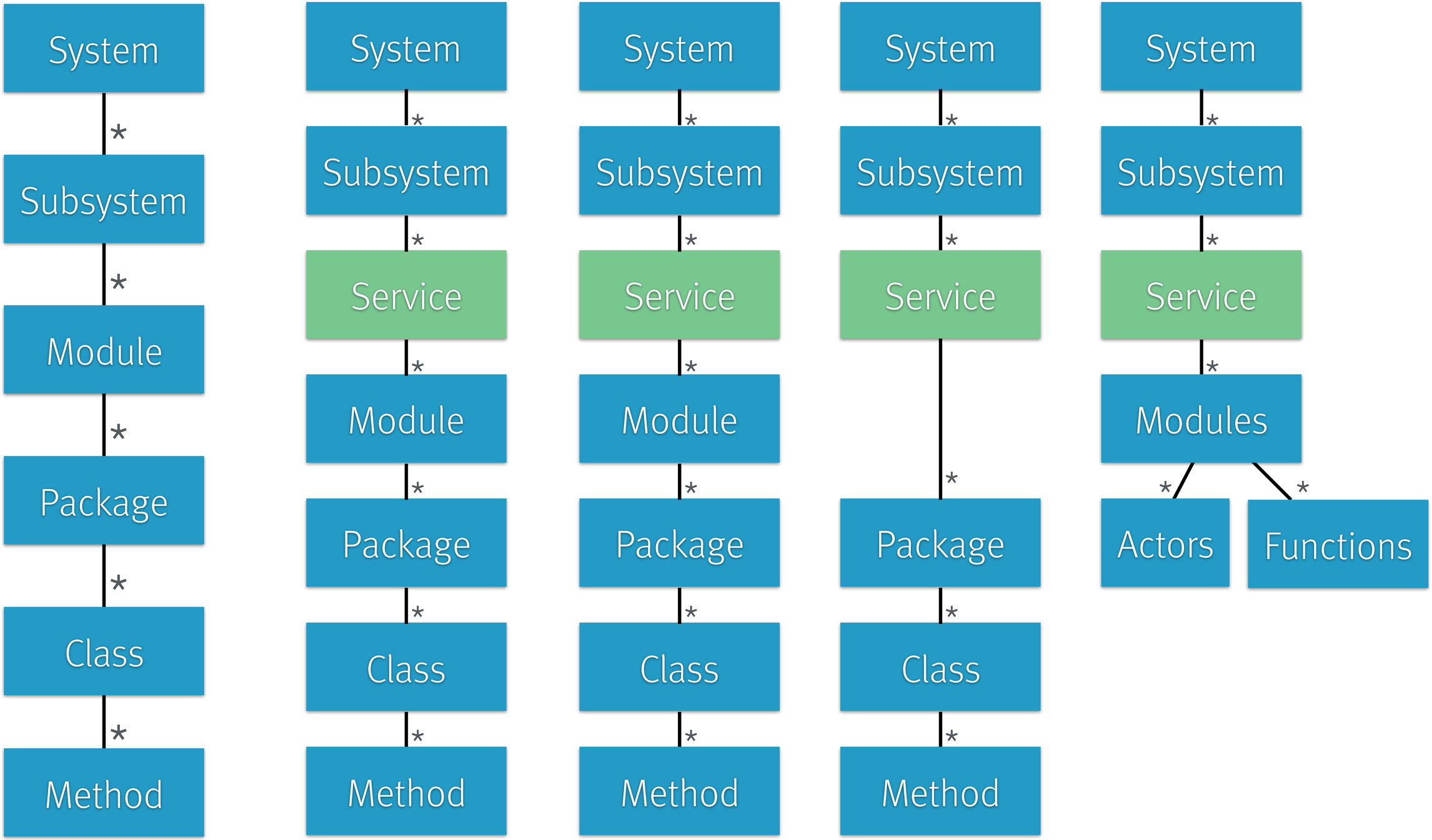
Hierarchy & Rule Example



- Systems communicate async, use front-end integration
- Subsystems can use sync calls via facades
- Modules only depend on modules of lower layers
- Packages must not have circular dependencies
- Classes within a package can collaborate closely
- Methods must not call beyond depth 2

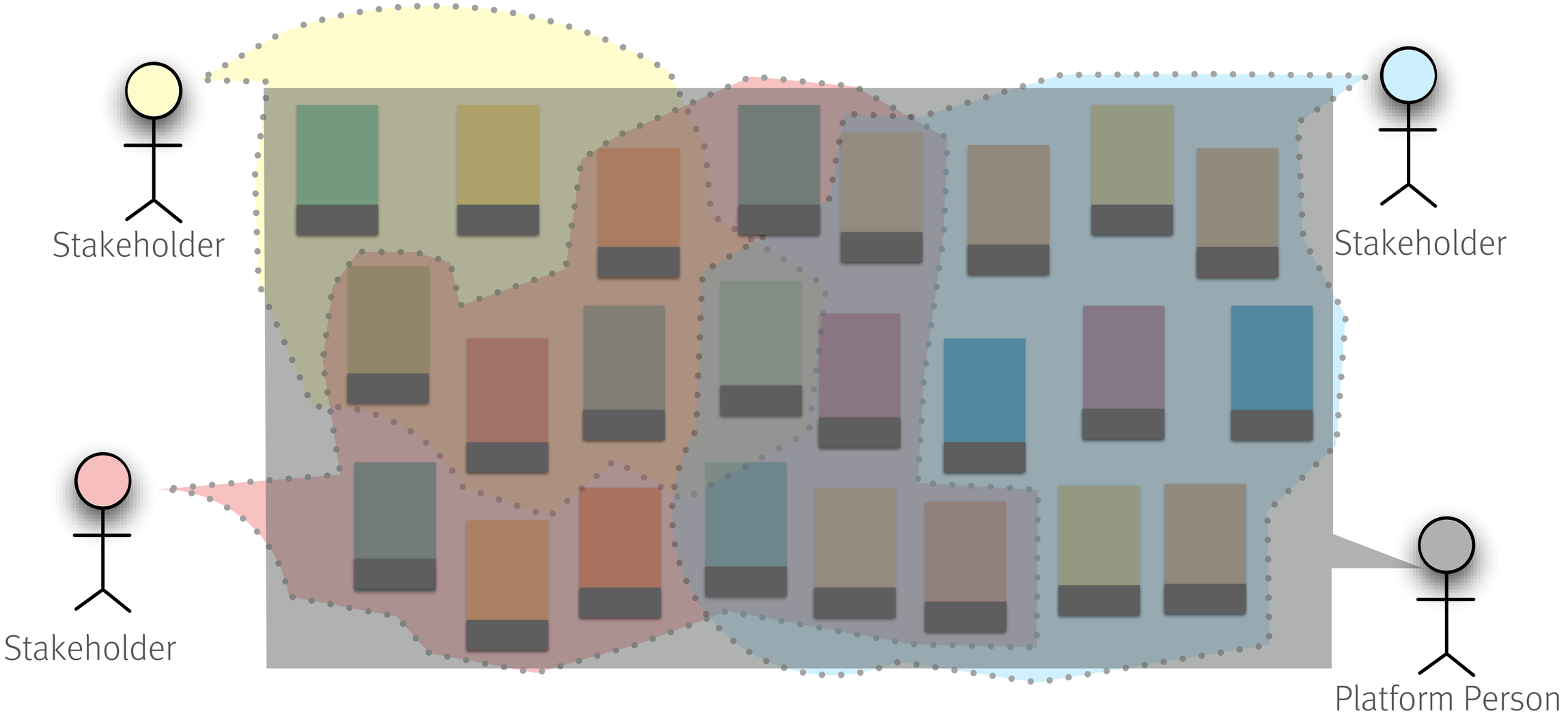
Different modularization levels

Different rules & strategies

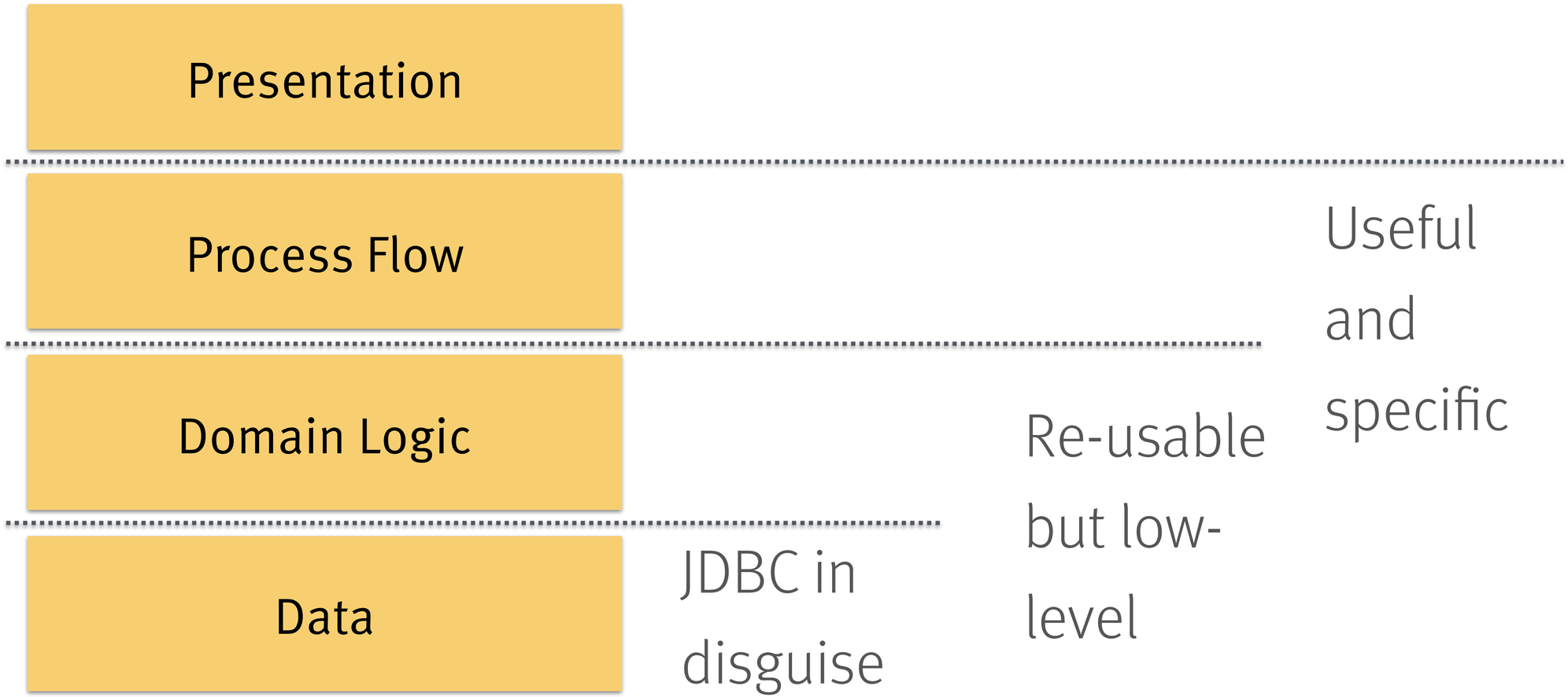

System
Subsystem
Module
Package
Class
Method
System
Subsystem
Service
Module
Package
Class
Method
System
Subsystem
Service
Module
Package
Class
Method
System
Subsystem
Service
Package
Class
Method
System
Subsystem
Service
Modules
Actors
Functions

Microservice Sizing Antipatterns

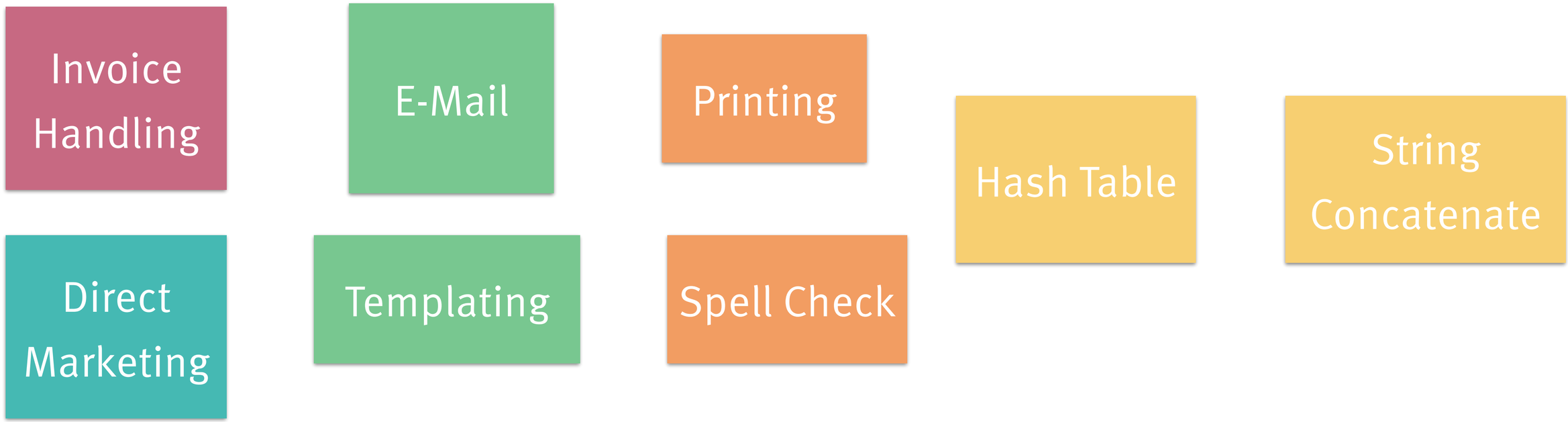
Antipattern: Decoupling Illusion



Antipattern: Anemic Service



Antipattern: Unjustified Re-Use



Lessons learned

What works:

Being explicit about your meta-model

What doesn't:

Mentioning the word "meta-model"

What works:

Separating macro and micro decisions

What doesn't:

Over-regulating everything

What works:

Trusting your gut and making a good guess

What doesn't:

Fleeing into technicalities

What works:

Use organization and its

use cases as level `0` driver

What doesn't:

Center around technical

commonality

What works:

Prepare to be wrong on every level

What doesn't:

Aim for perfection and stubbornly stick to it

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

LONDON
INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE

goto; conference

Please

Remember to rate this session

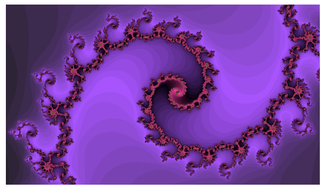
Thank you!

Join the conversation #gotoldn

Image Credit

David Mellor Kitchen Knives, https://flic.kr/p/pyW8xB

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



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