



Eberhard Wolff
@ewolff

Microservices – After the Hype

INNOQ

EBERHARD WOLFF

Fellow at INNOQ Deutschland GmbH

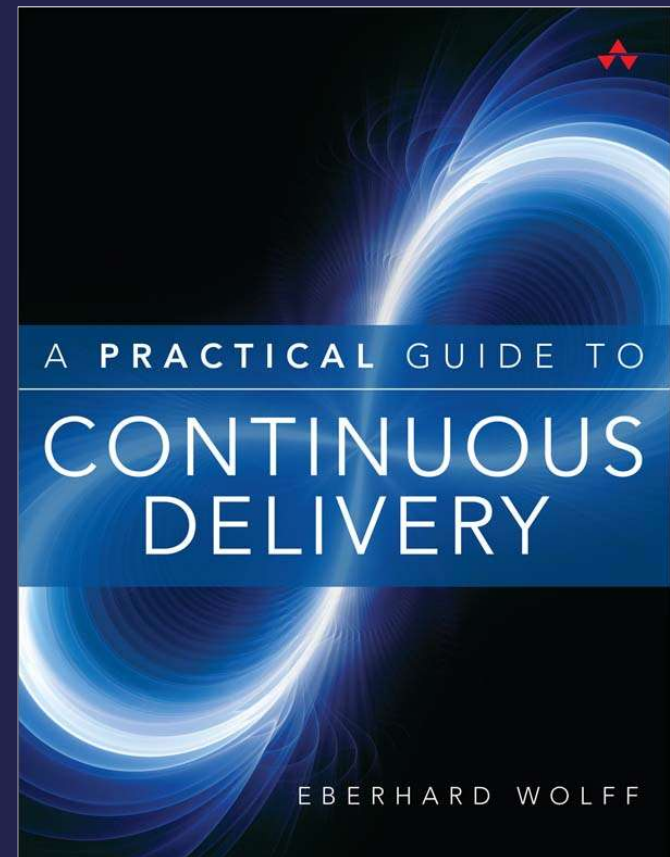
@ewolff

www.ewolff.com



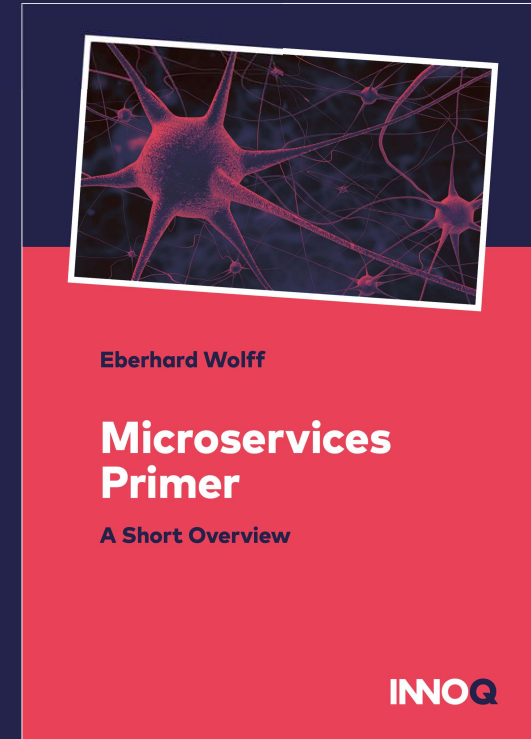
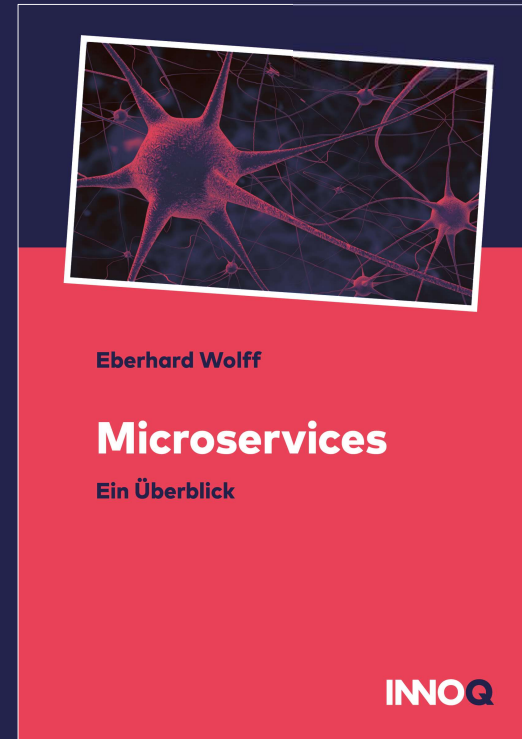
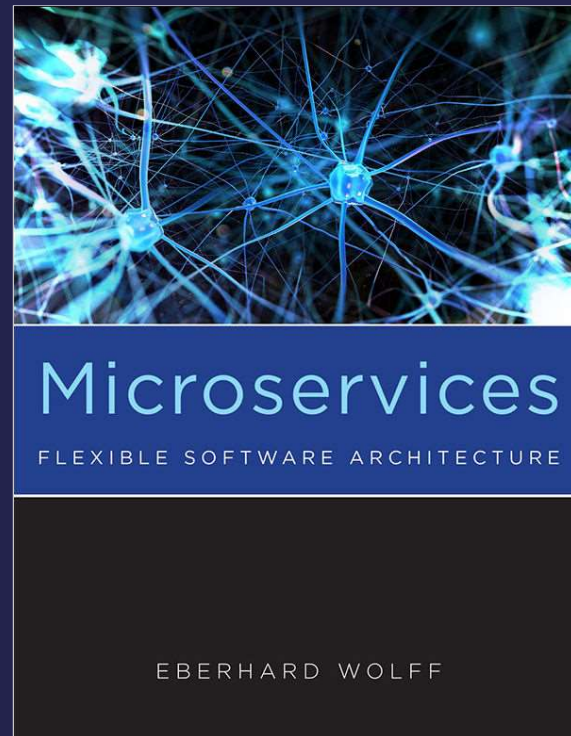
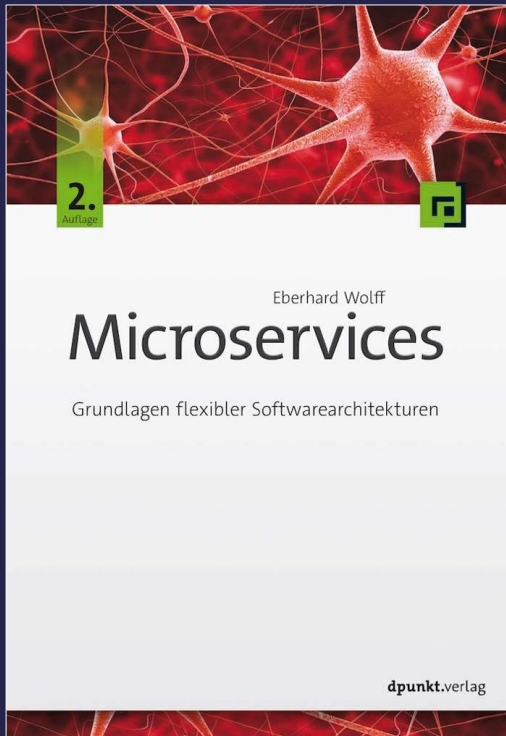


www.continuous-delivery-buch.de



www.continuous-delivery-buch.de

FREE



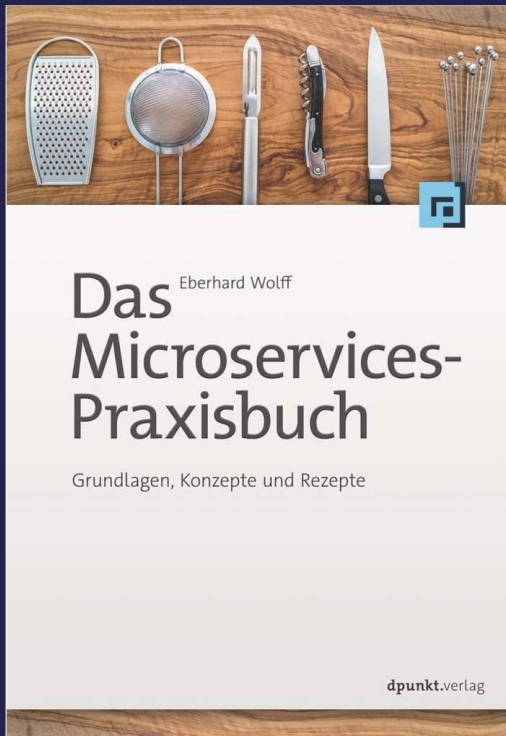
microservices-buch.de

microservices-book.com

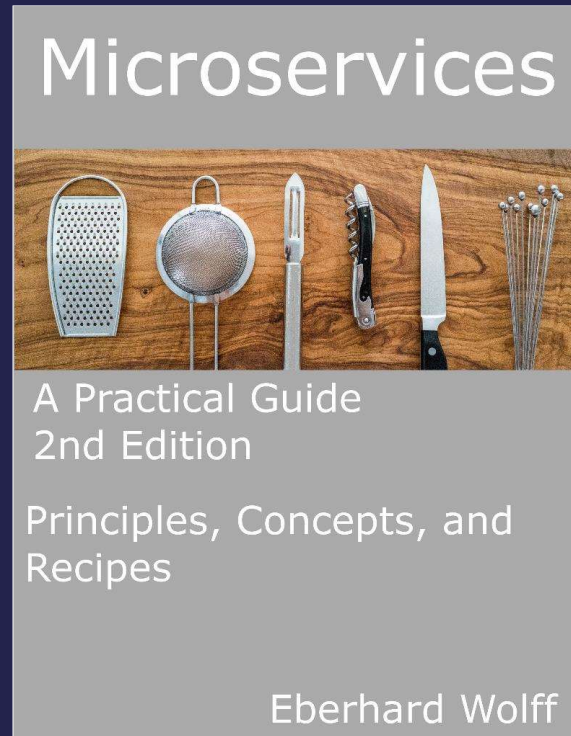
[microservices-buch.de/
ueberblick.html](http://microservices-buch.de/ueberblick.html)

[microservices-book.com/
primer.html](http://microservices-book.com/primer.html)

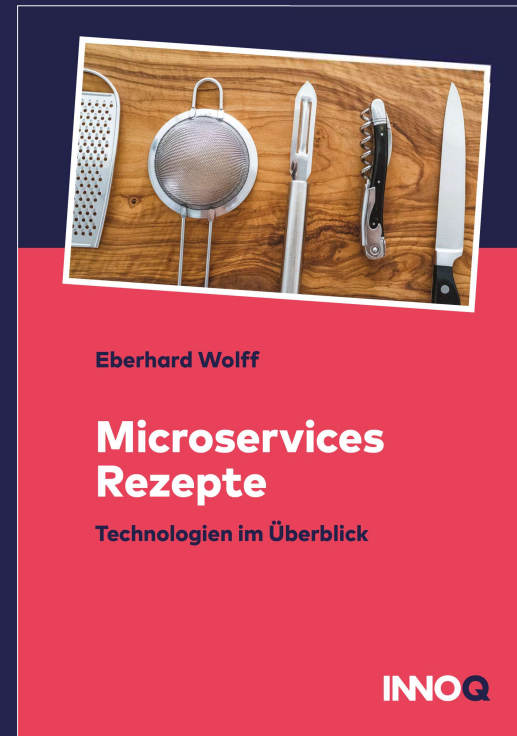
FREE



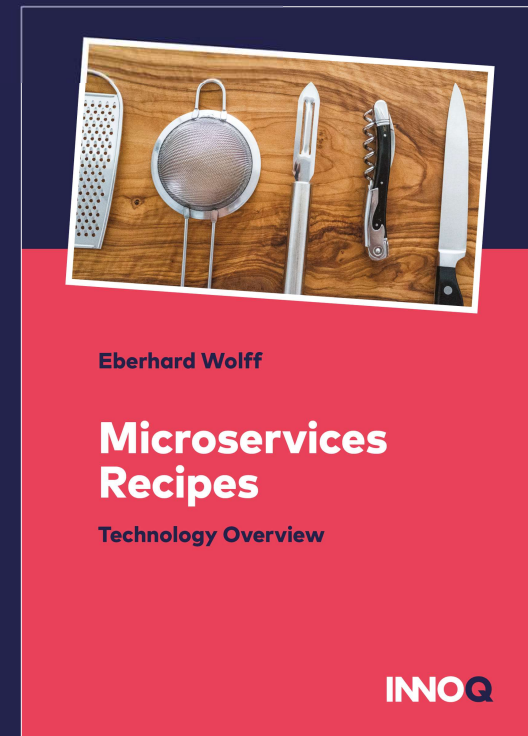
microservices-praxisbuch.de



practical-microservices.com



[microservices-praxisbuch.de/
rezepte.html](https://microservices-praxisbuch.de/rezepte.html)



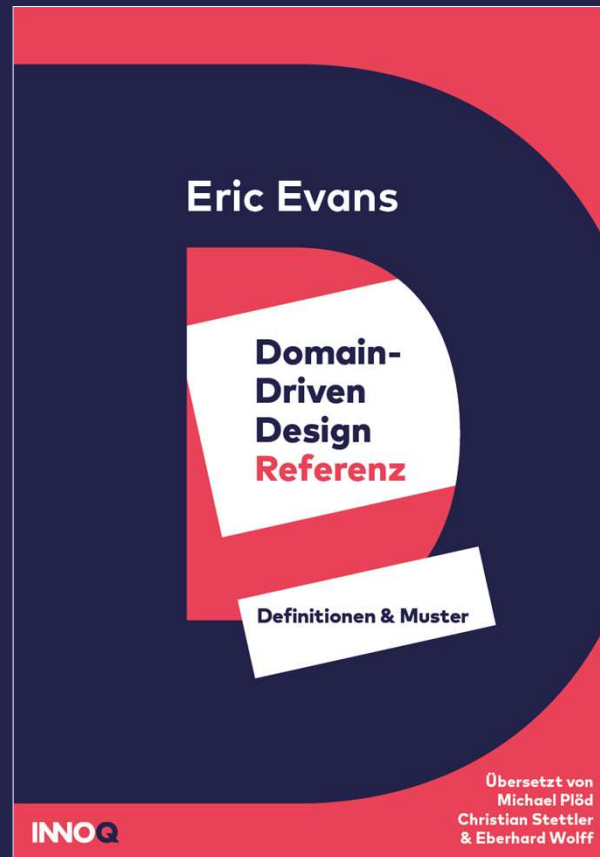
[practical-microservices.com/
recipes.html](https://practical-microservices.com/recipes.html)

FREE



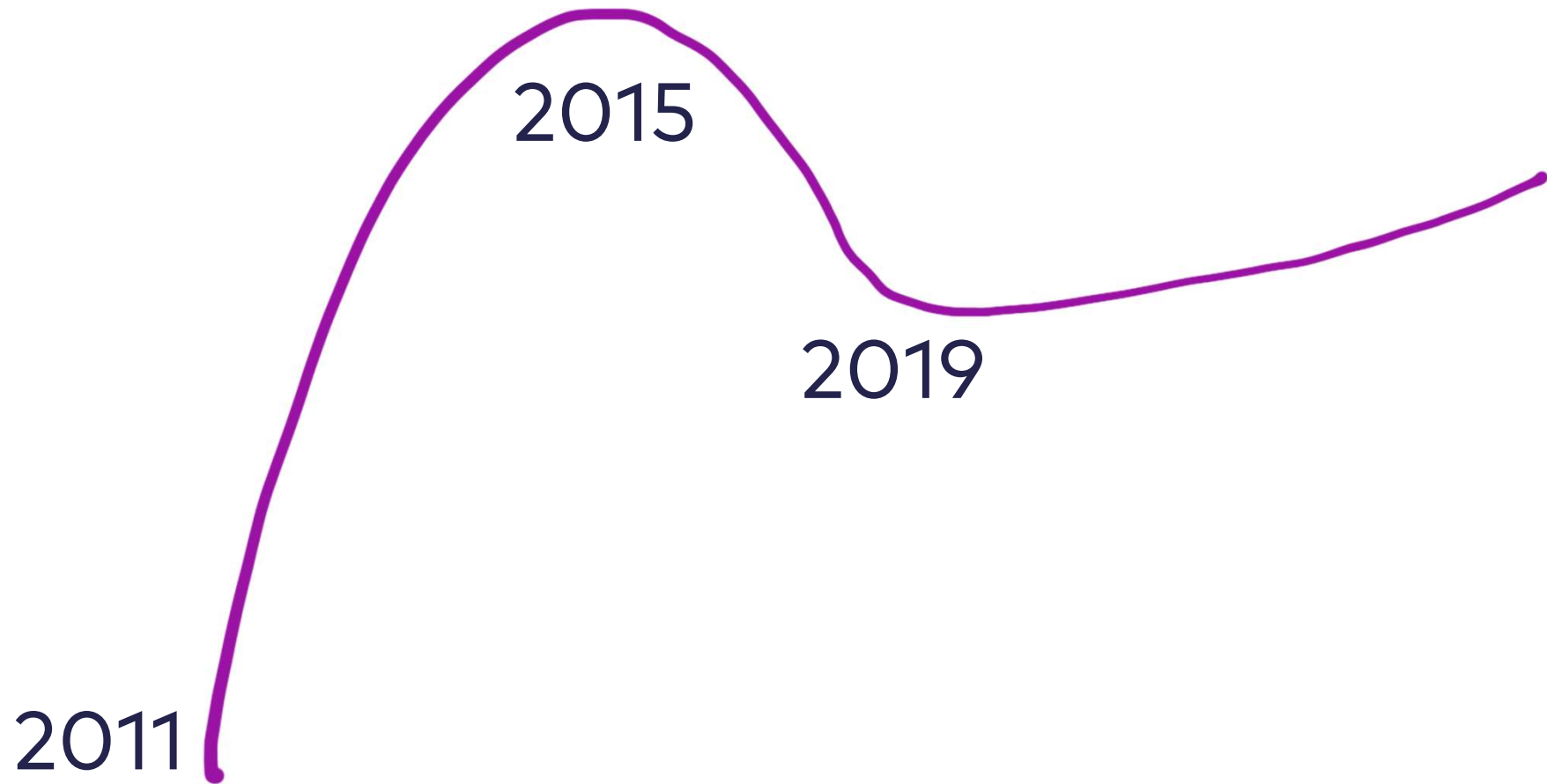
leanpub.com/service-mesh-primer/

FREE



www.ddd-referenz.de www.domainlanguage.com/ddd/reference

Hype Cycle



Hype Cycle & Projects

- There are just advantageous decisions
...and not so great decisions
...for a specific project

Hype Cycle & Projects

- Why care about hype cycles?
- Why use cool stuff if it doesn't help your project?
- Why skip uncool stuff if it solves your problem?

PUBLIC



ENEMY

Def
Jam
recordings



DON'T BELIEVE THE HYPE

652833 7

Why is there even a hype?

How Did Microservices Start ?

Life Before Microservices

- Deployment monoliths
- Several million lines of code
- Compilation time: hours
- Start-up time: many minutes
- Complex integration with other systems
- Votes: Who ever had too large / small deployment units?

Wikipedia: History of Microservices

Software
architecture
workshop

2011: common
architectural style

2012 Coined the
term "microservice"



Adrian Cockcraft

- Architecture at Netflix
- "Fine grained SOA"
- Cloud
- High scalability
- Independent teams



James Lewis

- 2012 Presentation "Java the Unix Way"
- Scale to many developers
- Tough non-functional requirements (performance, scalability)
- Implement Unix services in Java, REST



Fred George

- Diverse projects
- Tiny services
- Asynchronous communication
- Developer anarchy
- Delivered many projects



Microservices: The Beginning

- Different, senior people
- Significant projects
- Solve similar problems
(e.g. scaling projects)
- ...but also specific problems
(i.e. functional requirements, cloud)
- Diverse technologies & principles
(sync/async, small/large microservices)

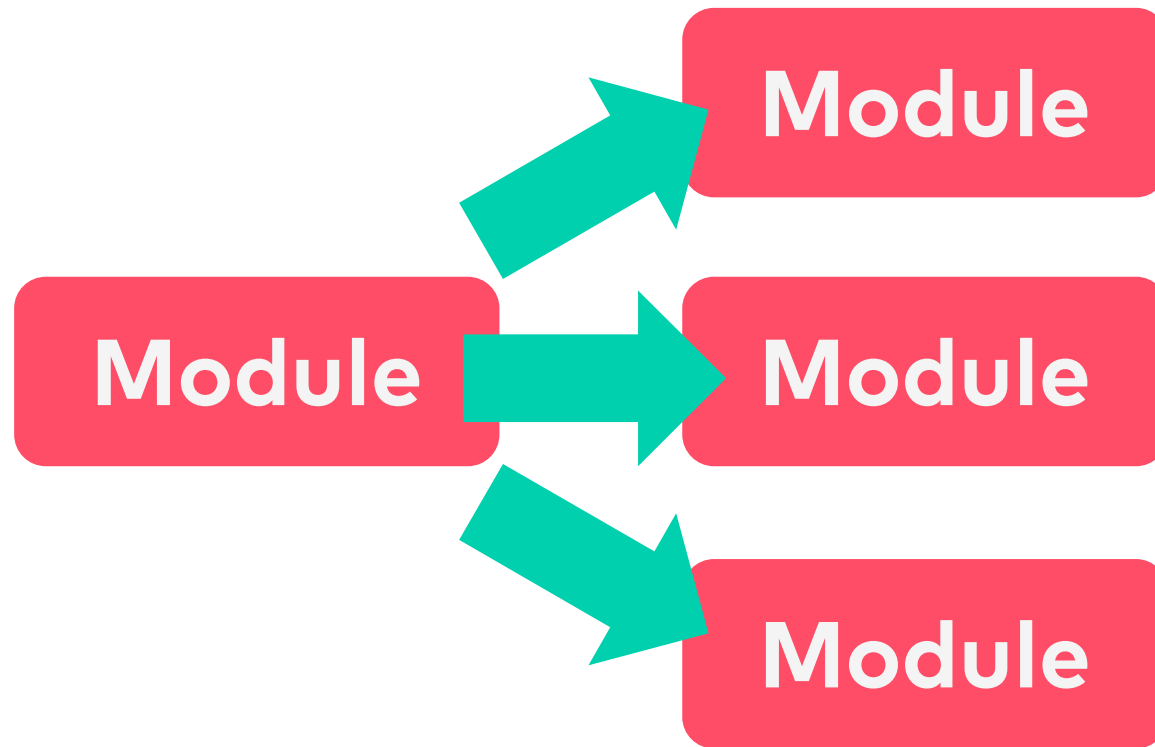
**There was never a clear
"Microservices" concept.**

**Microservices started as the
(simplest / only?) solution to real
problems!**

Microservices: not just an over-hyped solution!

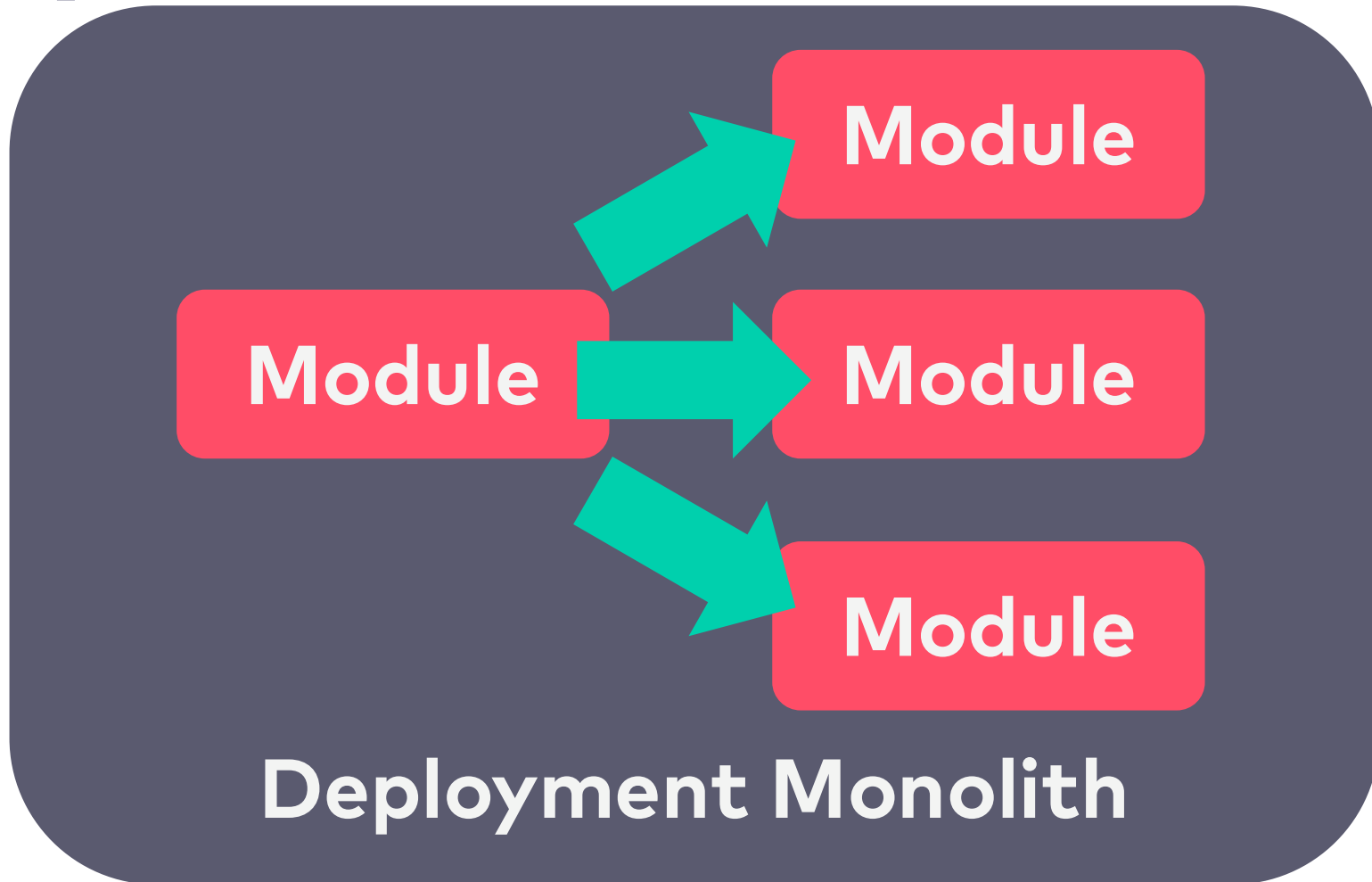
Modules

Deployment Monolith



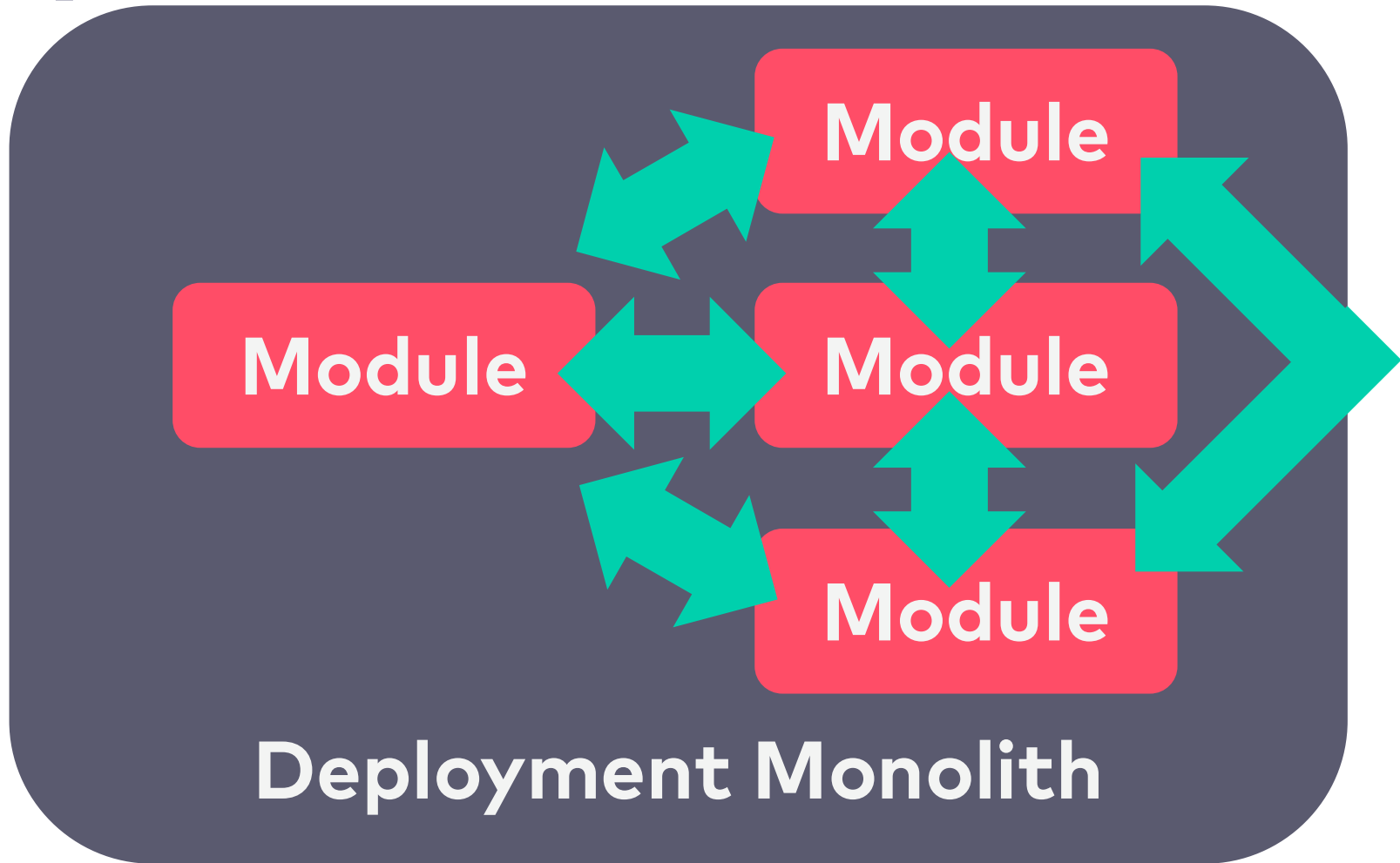
Desired Architecture

Deployment Monolith

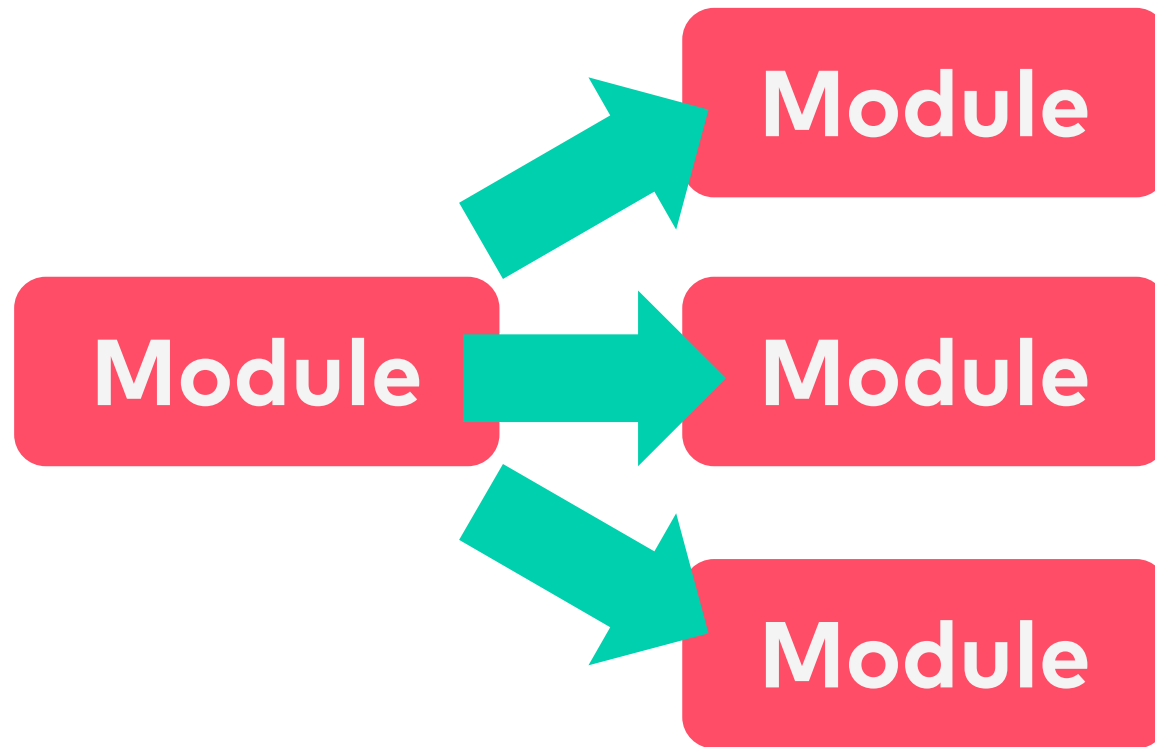


Deployment Monolith

Dependencies
sneak in

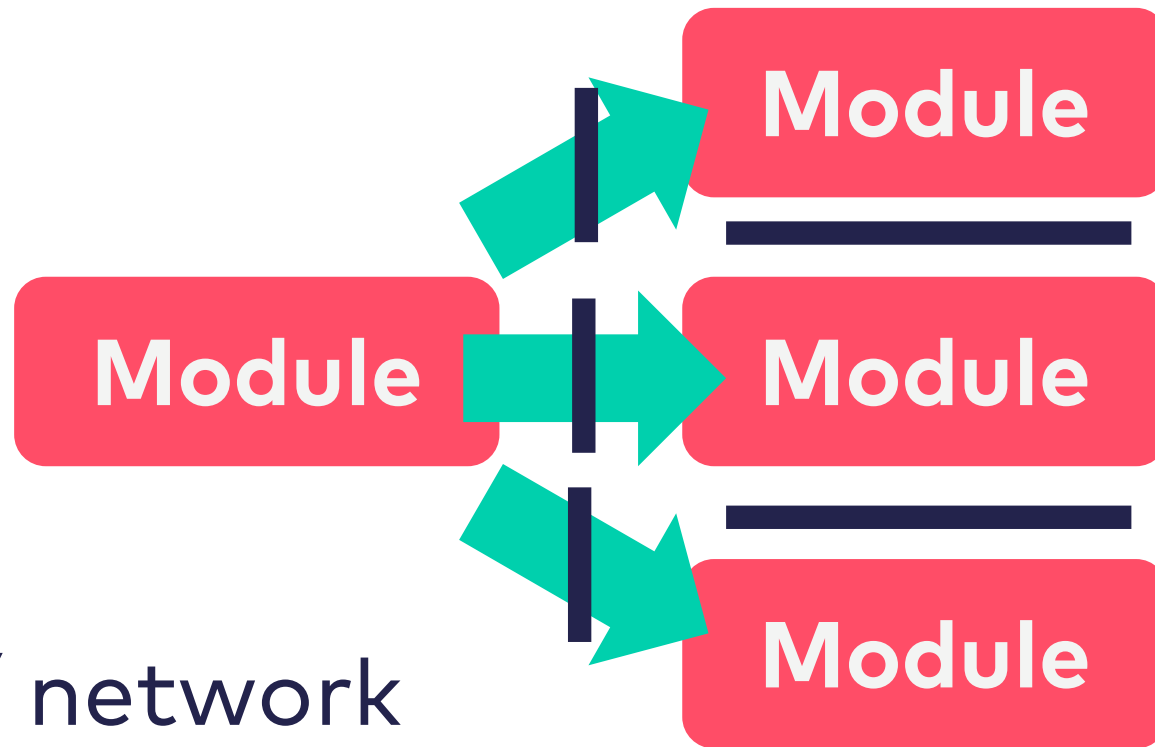


Microservices



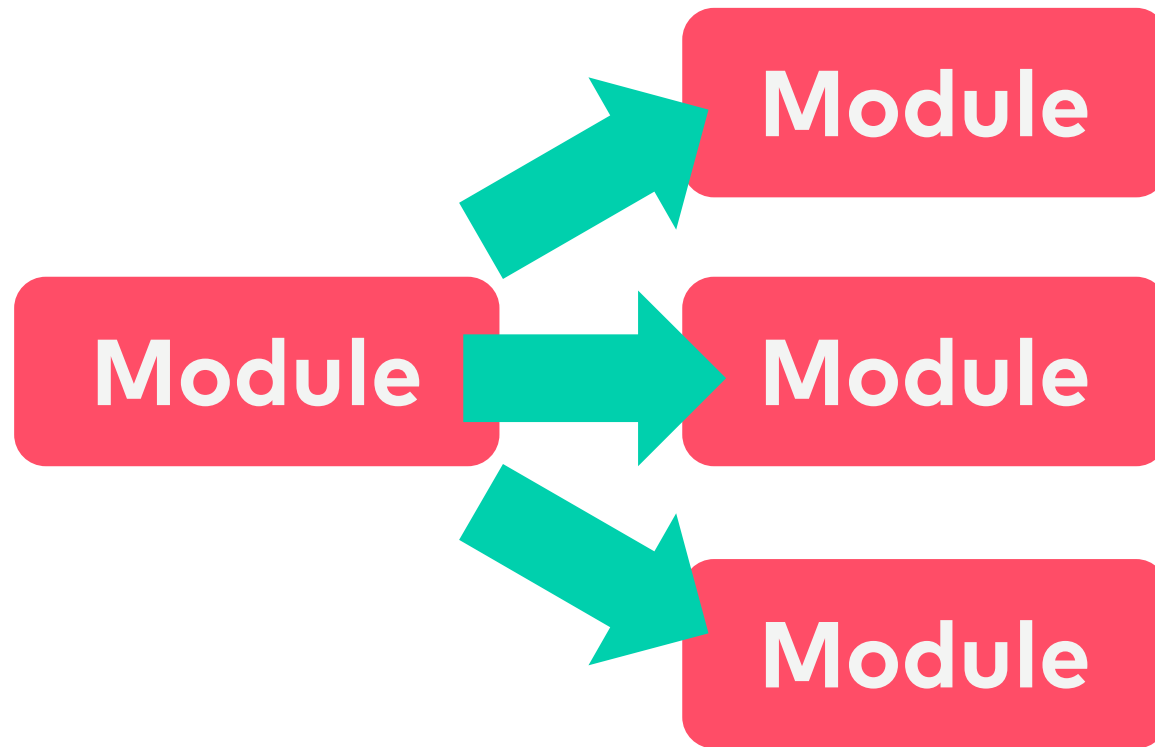
Desired Architecture

Microservices



| Process / network
| boundaries

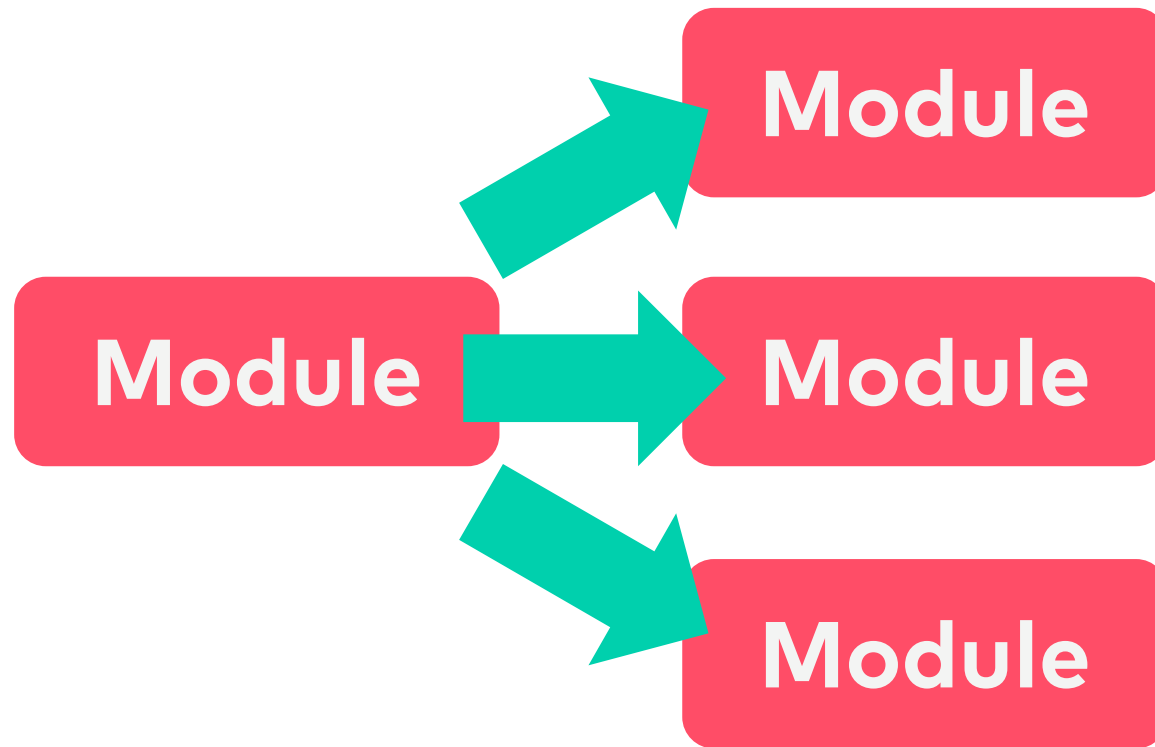
Deployment Monolith: Dev



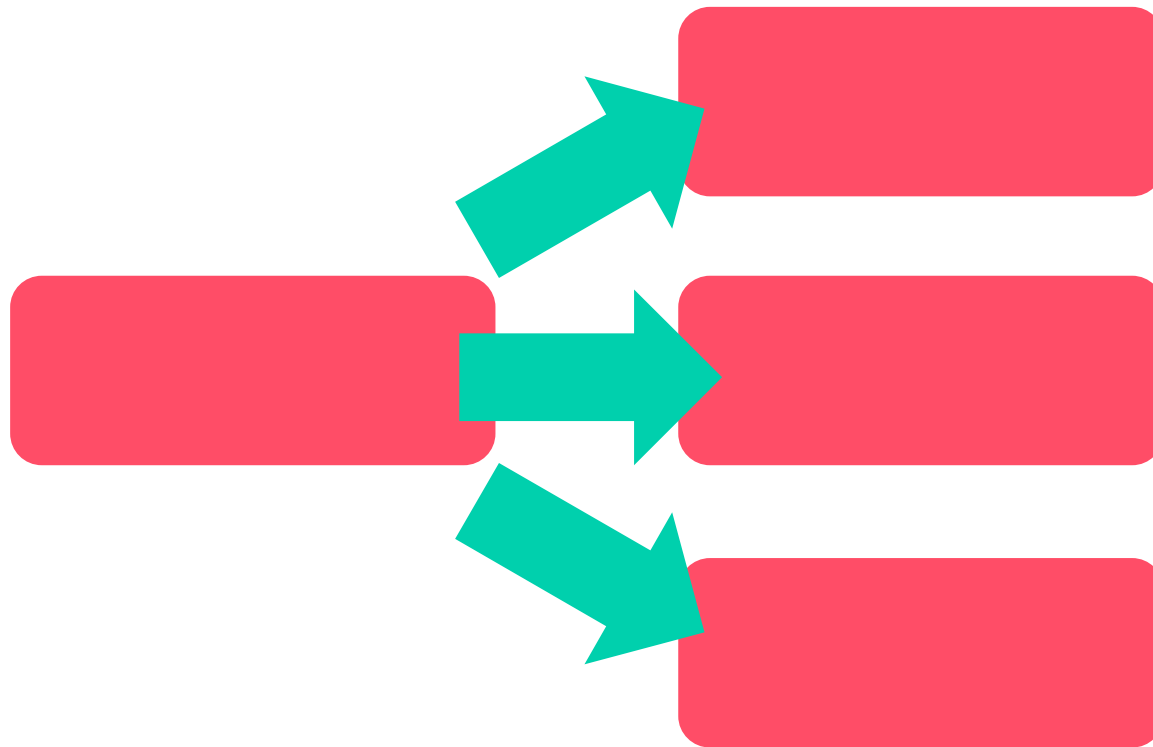
Deployment Monolith: Ops

Deployment Monolith

Microservices: Dev




Microservices: Ops



Modules & Microservices

- Strong boundaries between modules
- Modules not just for developers
- ...but also for communication and deployment

Modules & Microservices

- Modules more obvious
- Modules harder to change
- Modules became important again! 
- Modular Monoliths

Modular Monoliths

- Architecture monolith: no modules
- Deployment monoliths may have modules
- Deployment monoliths should have modules!
- Hardly surprising

Modular Monoliths

- If microservices hype helps to understand modules, great!
- Why are so many deployment monoliths badly structured???

**If all your deployment monoliths
are badly structure – why would
the next one be better?**

Architecture

Again: Architecture

- Find a solution for the problem at hand!
- Microservices: just one tool
- Microservices: just one part of a solution
- Why exclude microservices up-front?
- Why microservices in a specific case?

Domain-driven Design

Understanding DDD 2005

- An OO paradigm
- Classes will be repositories, services, entities, aggregates
- I.e. fine-grained concepts

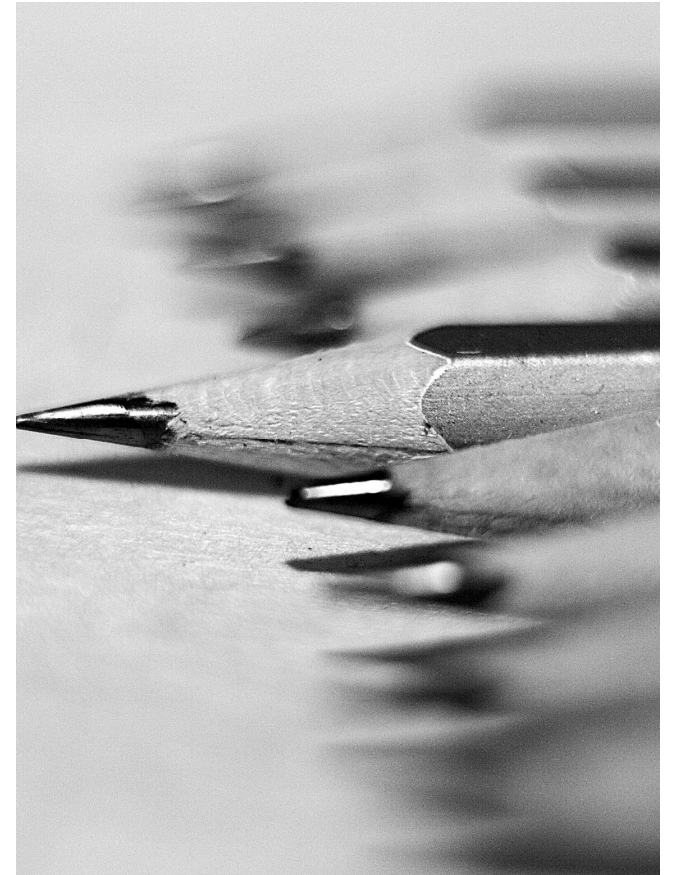


Photo: Konstantin Stepanov

Understanding DDD 2019

- Coarse-grained modules
- Bounded context
- Strategic design
- Core domain / generic subdomain



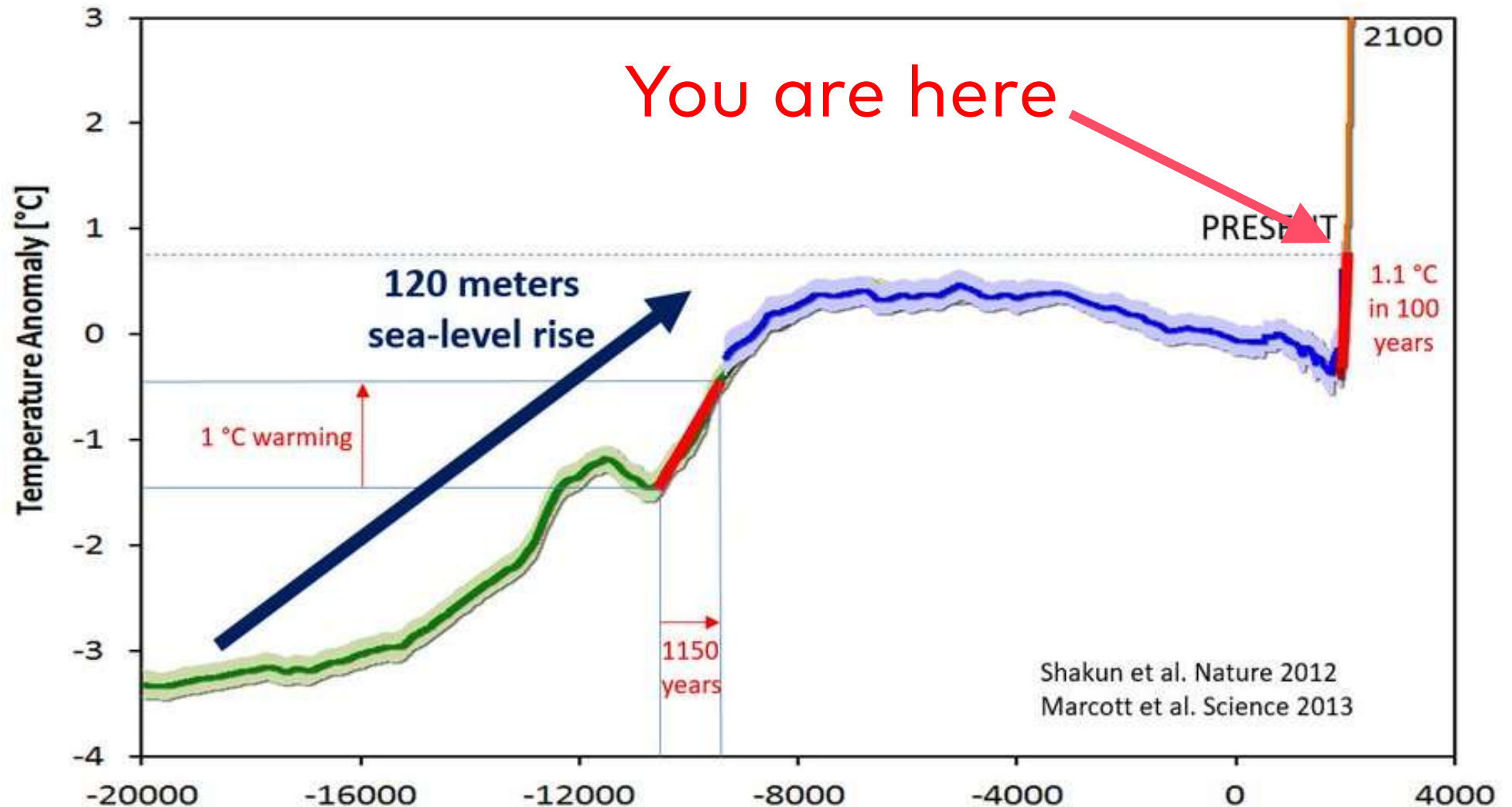
Photo: Bill Smith

Architecture

- Modules important again 
 - DDD renaissance 
 - Coarse grained decoupling 
 - Microservices: new type of modules
 - Fundamental!
-
- But: microservices just one option

Long-term Architecture

GLOBAL TEMPERATURE SINCE THE LAST ICE AGE



Shakun et al. Nature 2012
Marcott et al. Science 2013

<http://www.realclimate.org/index.php/archives/2013/09/paleoclimate-the-end-of-the-holocene/>

<https://twitter.com/rahmstorf/status/1186963006640545792>

Long-term Architecture

- Many systems survive very long.
- Is "clean" architecture enough?
- How to avoid architecture rot?
- How many successfully avoided it?
- Enforce architecture strictly?
- With self-organization?

Long-term Architecture

- Which decision survive many years?
- How can you predict the future?
- Crystal ball?



Photo: Carlos Ebert

Long-term Architecture: Structure

- DDD's Bounded Context might be stable
- Capture essence of business
- Probably the same in a few years
- If essence of business changes, software is probably your least concern.

Long-term Architecture: Technologies

- Today's fancy new technology
... will be outdated sooner or later.
- Security fixes often only in latest releases
- Need to be up-to-date
- Large, risky technology updates 😞

**Are there other examples
of successful long-term
architecture?**

Modular Synthesizer

- Communication: control, trigger
- Deployment: Module sizes, power

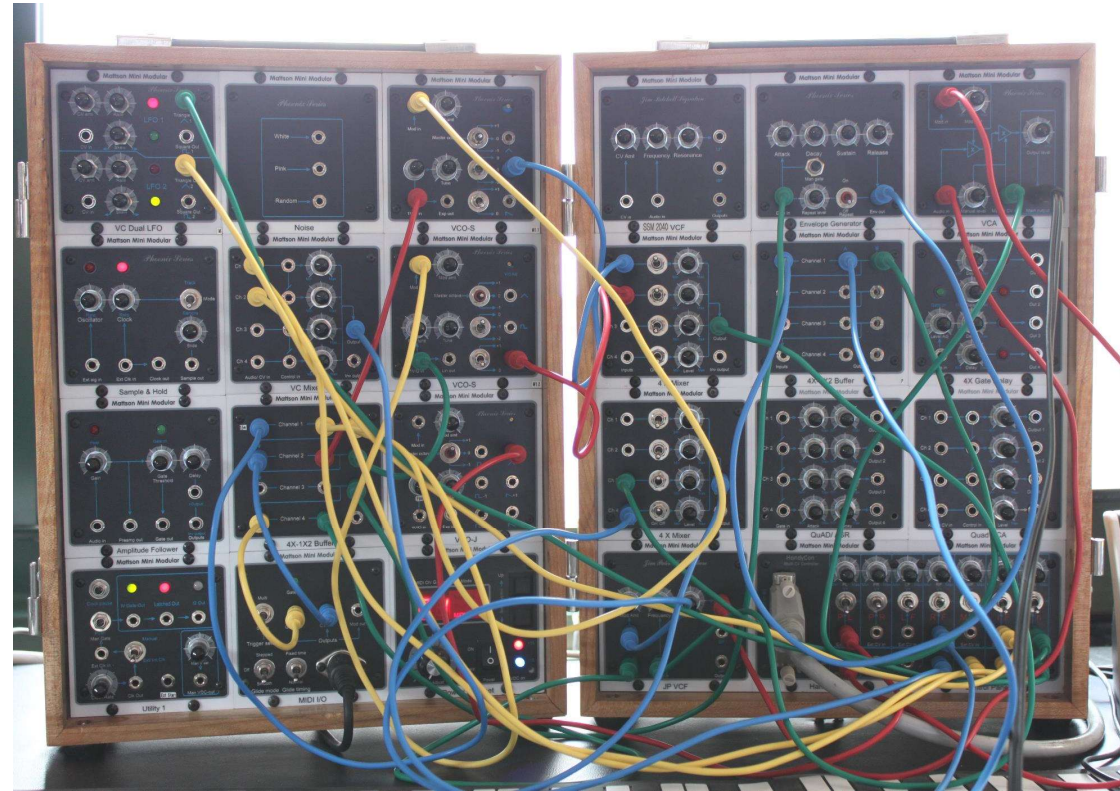


Photo: Muff, Flickr

Modular Synthesizer

- Since 1996
- >5.000 module
- Many, very different modules

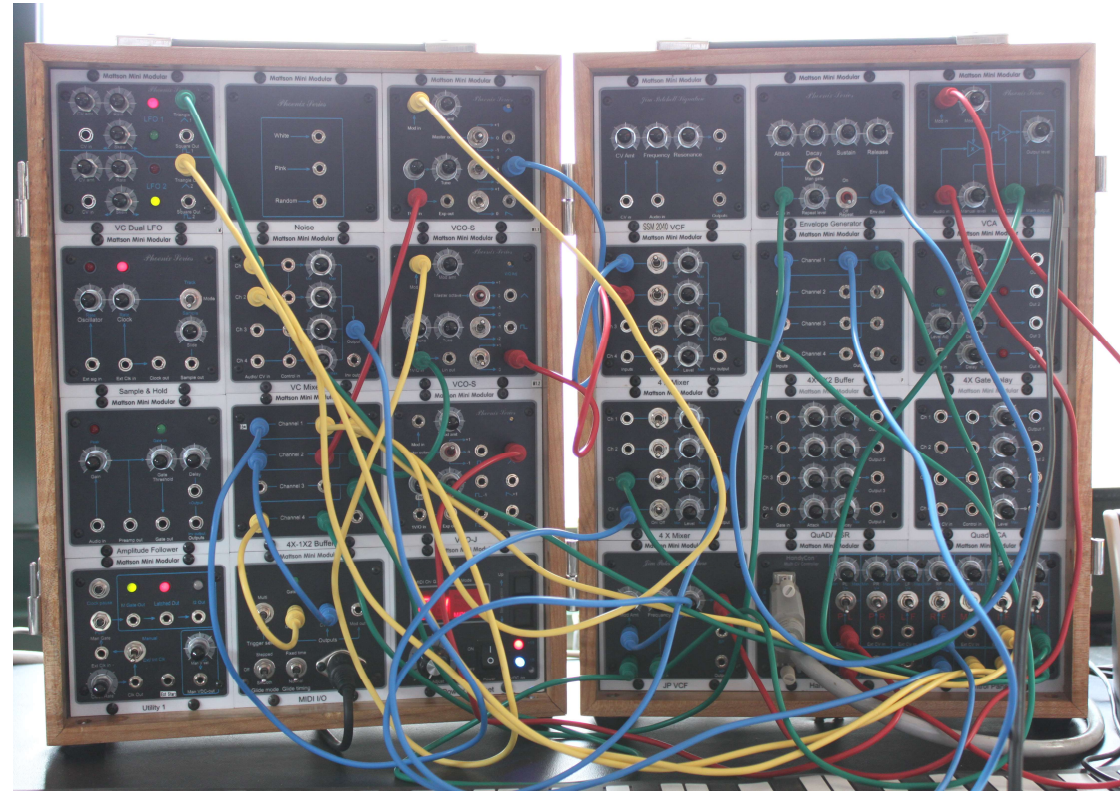


Photo: Muff, Flickr



Microservices

- Communication:
REST, messaging etc.
- Deployment:
Container, ops interface
- Hide internal technologies
- Information hiding

Long-term Architecture: Technologies

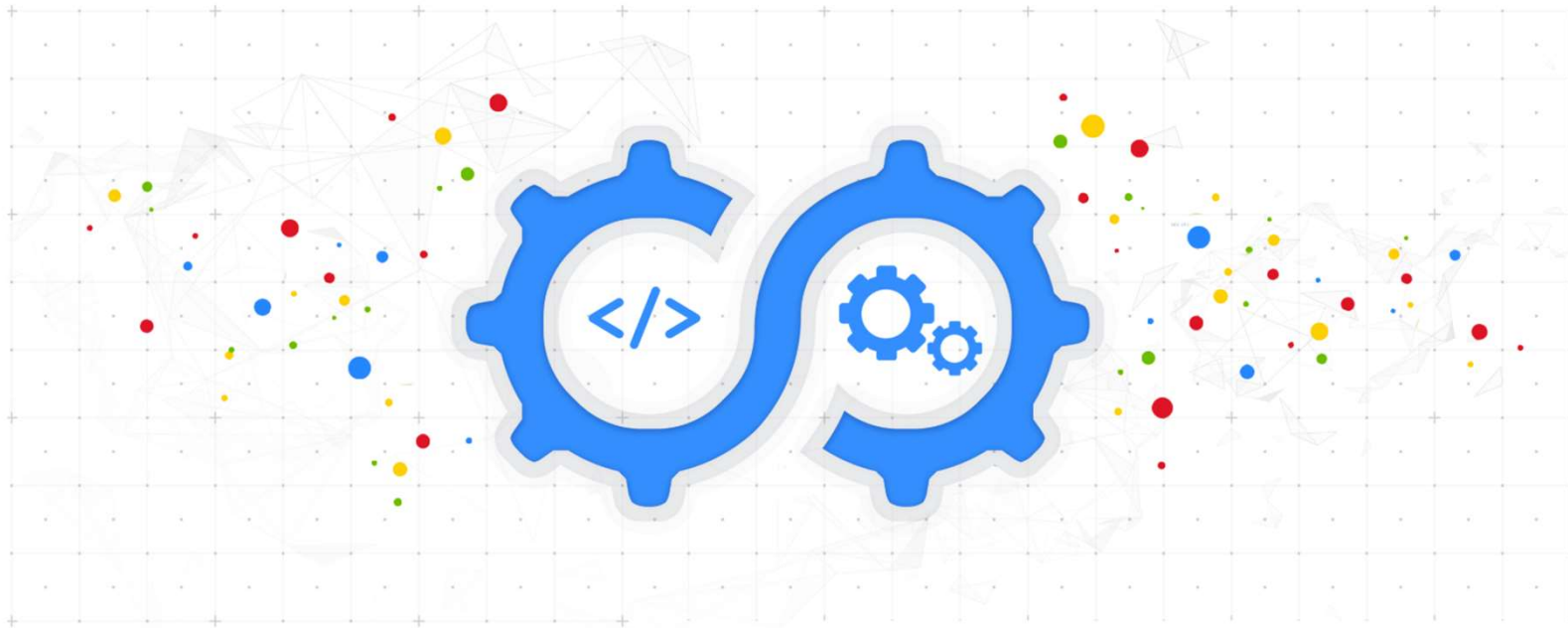
- Microservices allow heterogeneous technology stacks
- Support updates to new technologies
 - ...stepwise by microservice
 - ...to mitigate risk
- Microservices only option for long-term architecture?

Long-term Architecture: Conclusion

- DDD: stable coarse-grained split 
 - Architecture rot is hard to avoid.
 - Clean architecture less important
-
- Plan for heterogeneous technologies 
 - Microservices only option to do this?

Continuous Delivery

The 2019 Accelerate State of DevOps: Elite performance, productivity, and scaling

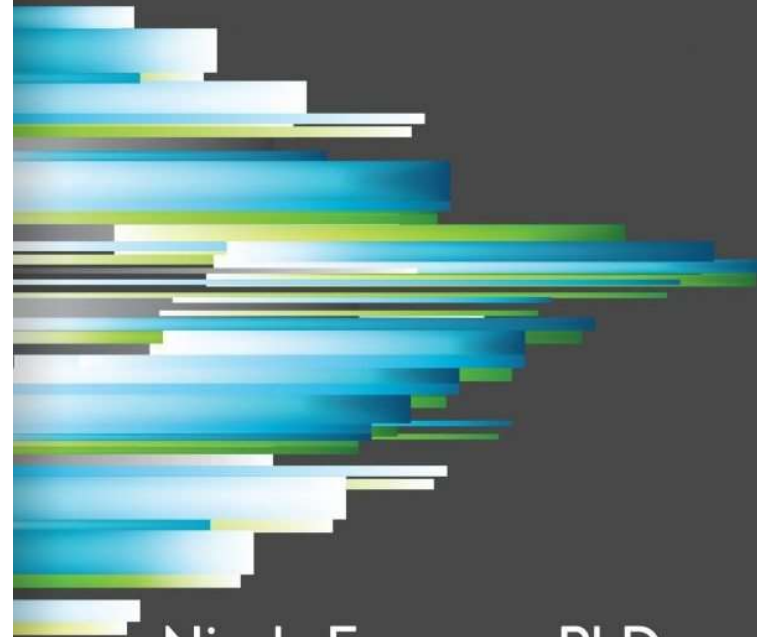


<https://cloud.google.com/devops/state-of-devops/>

THE SCIENCE OF LEAN SOFTWARE AND DEVOPS

ACCELERATE

Building and Scaling High Performing
Technology Organizations



Nicole Forsgren, PhD
Jez Humble, *and* Gene Kim

*with forewords by Martin Fowler and Courtney Kissler
and a case study contributed by Steve Bell and Karen Whitley Bell*

Deployment Frequency: Results

- Elite Performers vs. Low Performers
- Multiple times per day vs. once per month /6 months
- 106x better lead time for change
- 2.604x better time to restore service
- 7x better change failure rate
- 50% vs 30% time spent on new work (2018 report)
- Less work on security issues, bugs, end user support (2018 report)

Higher Deployment Frequency Improves

Lead Time
Reliability
New Work %

dramatically

Customer Scenario

- Quarterly releases
- 10 weeks of testing
- Release two days over the weekend



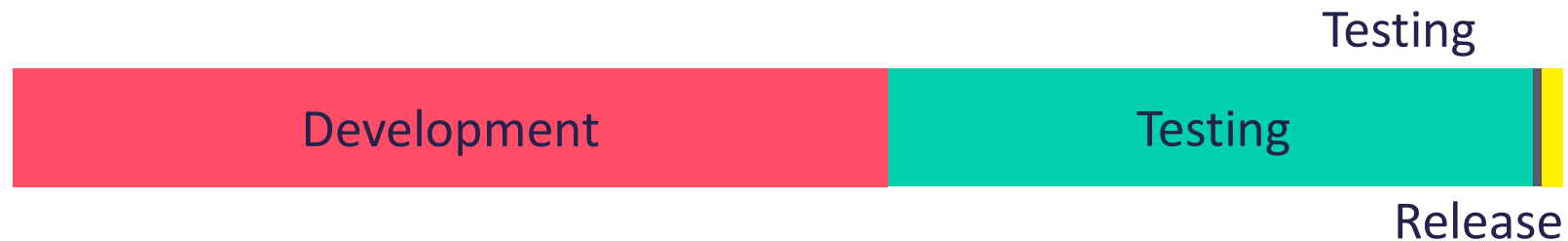
- Goal: Several deployments a day

Release



Testing + Release + Development

Customer Scenario: Automation



- Development faster – smaller batches
- Assumption: Test automation = 100x speedup for test
- 4h Tests

Customer Scenario: Automation



- Assumption: Deployment automation = 4x speedup
- 2h Deployment

Customer Scenario: Automation



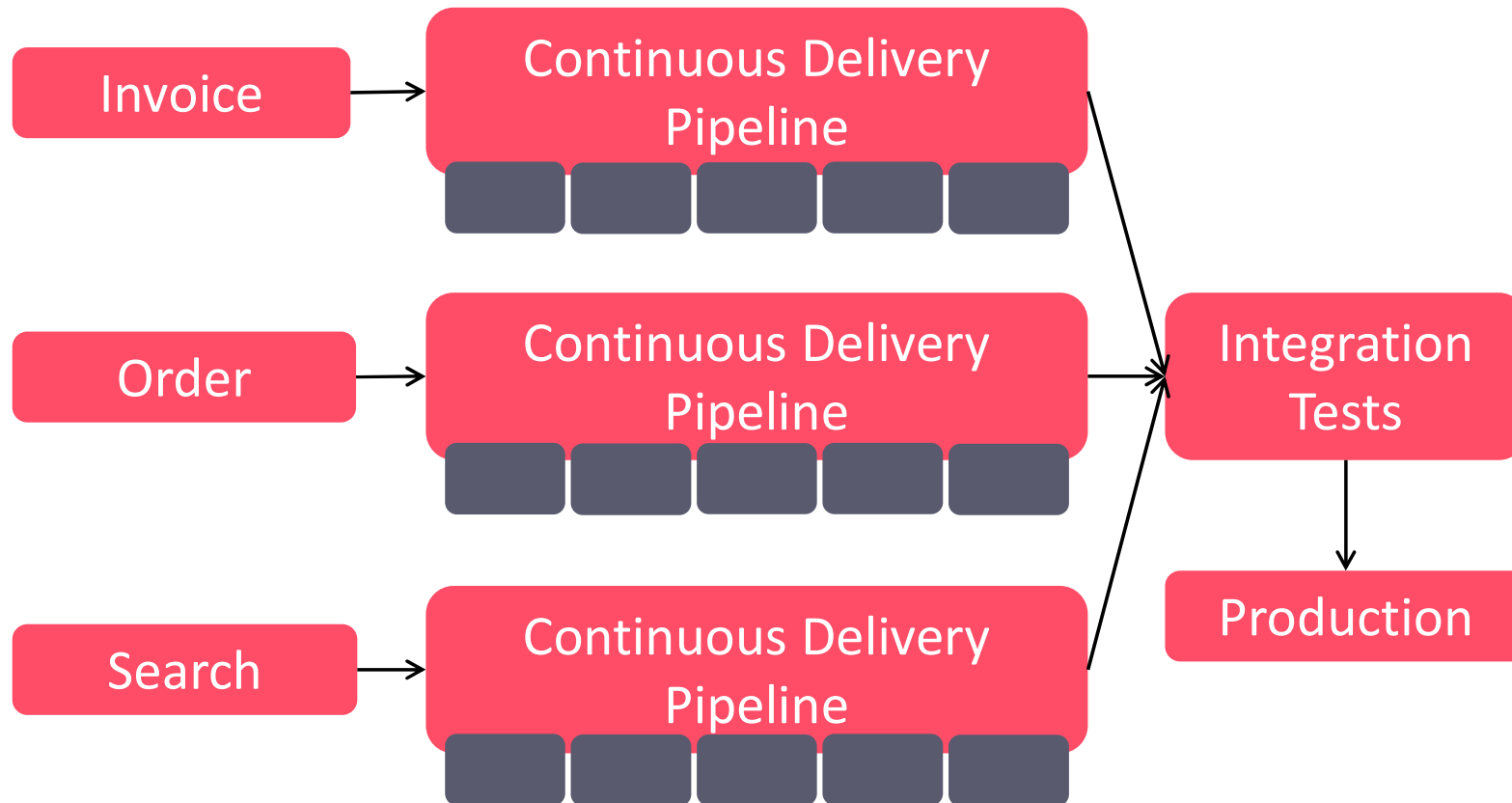
- 2h Deployment + 4h tests
= 6 hours
= 1 deployment per day
Need another threefold improvement

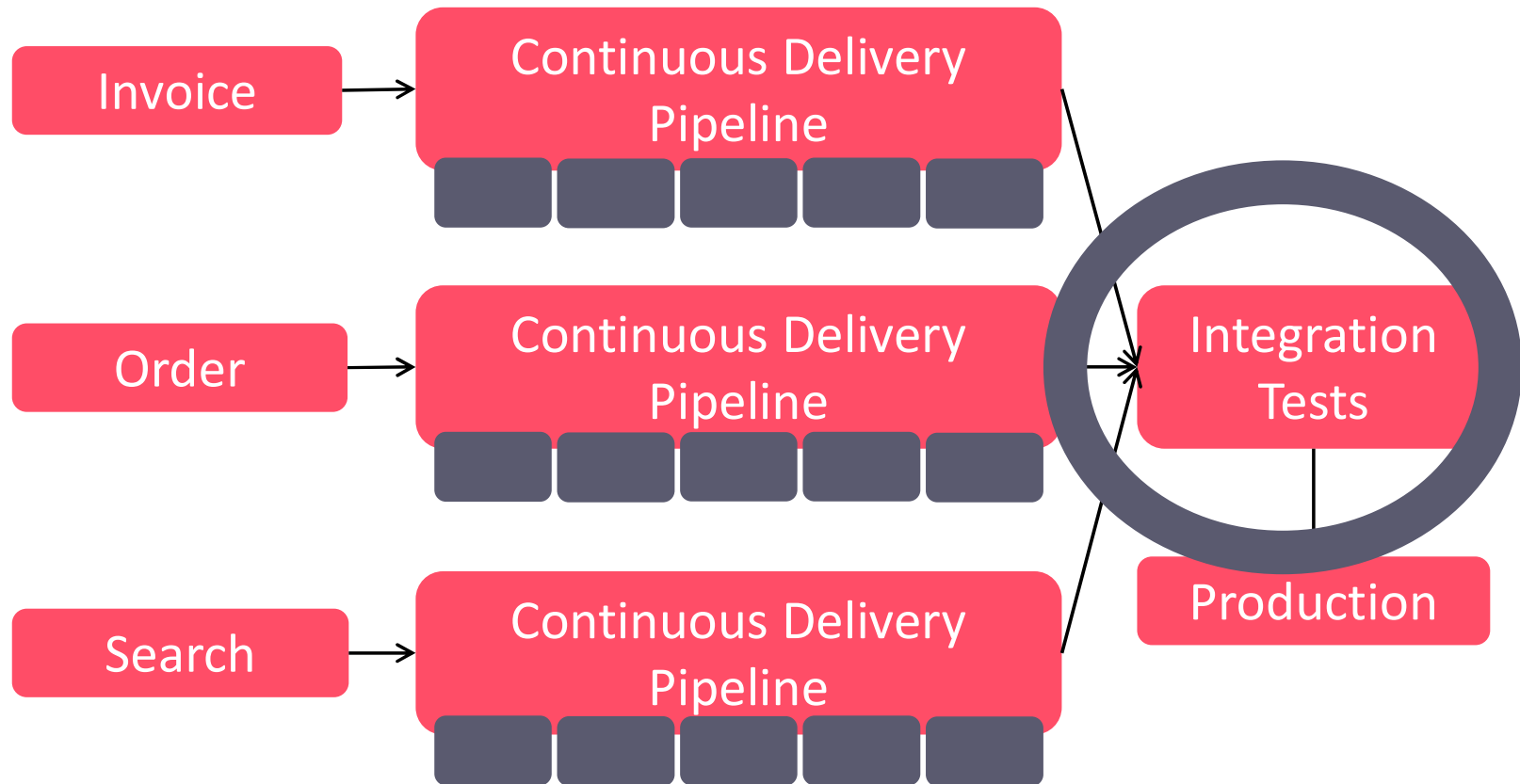
Solution

- Independently deployable modules
aka microservices
- Other solutions?

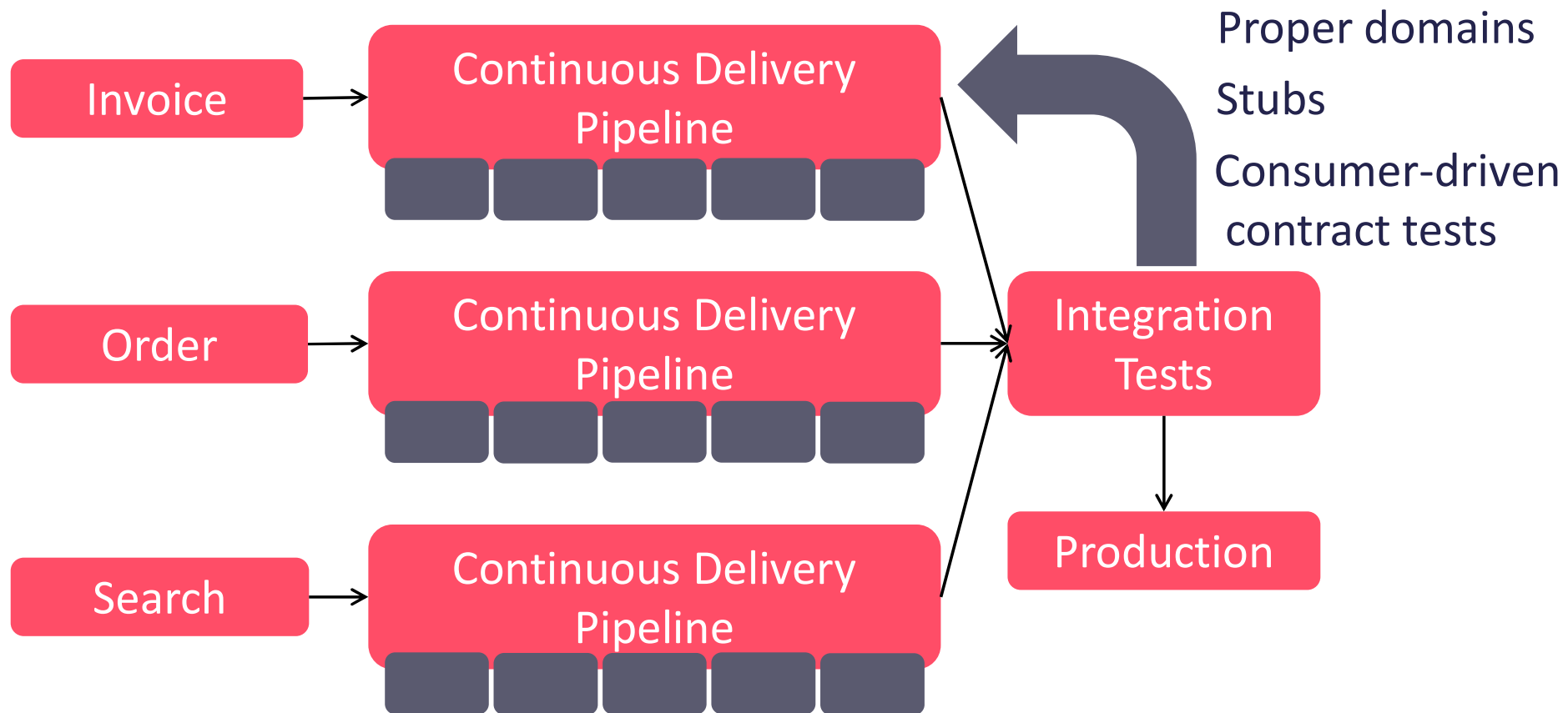
Open Challenges

- Complete environment for each dev
- Too hard to keep updated
- Proper modularization: not necessary





**Integration Test =
Bottleneck**



Continuous Delivery: Conclusion

- Continuous delivery gives many benefits
- Microservices
 - = independently deployable modules
- Microservices solve some continuous delivery challenges
- Should be considered
- CD: important reason for microservices

Culture & Organization

Conway's Law:
Architecture copies
communication structure



**Change
Order
Process!**



UI

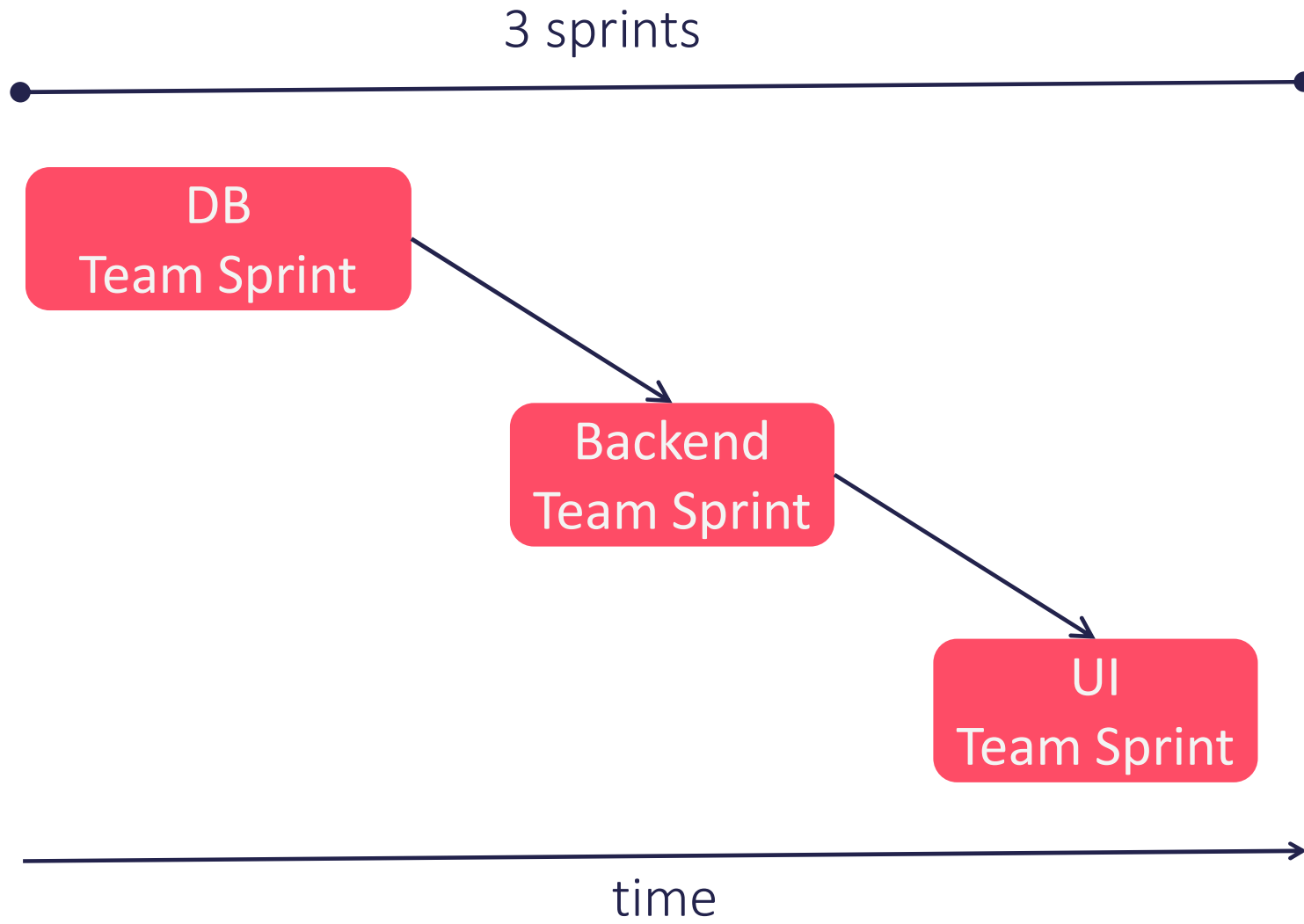


Backend



DB

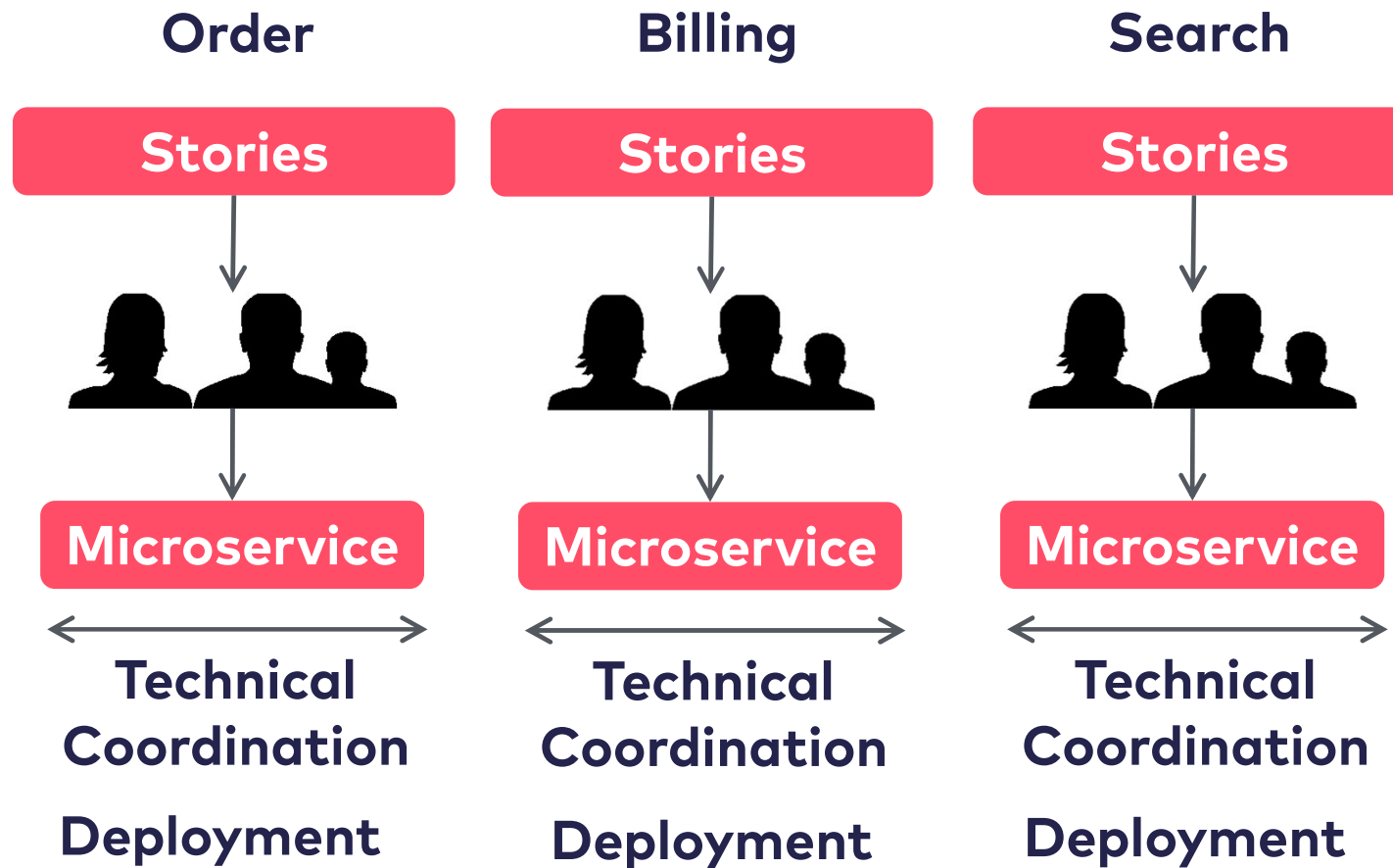
**E Commerce
Shop**



Inverse Conway Maneuver

- Let architecture drive the organization
- Team for each bounded context

Microservices & Inverse Conway



Independent Teams

- Architecture *supports* independent teams
- Do you trust teams to make the right decisions?
- Must teams communicate decisions?
- When do you step in?
- But really: culture / org problem

**Scaling a development process
means changing the organization
and the architecture!**

Org & Architecture: DDD

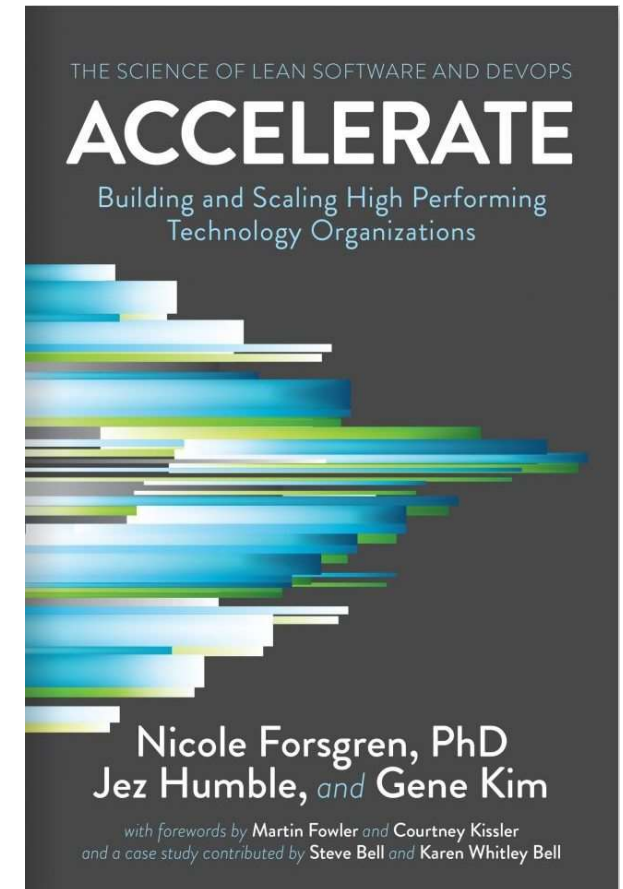
- Bounded context = team
- Strategic design: How *teams* interact

Org & Architecture: Agile

- Cross-functional teams
- Self-organizing teams
- Architecture?

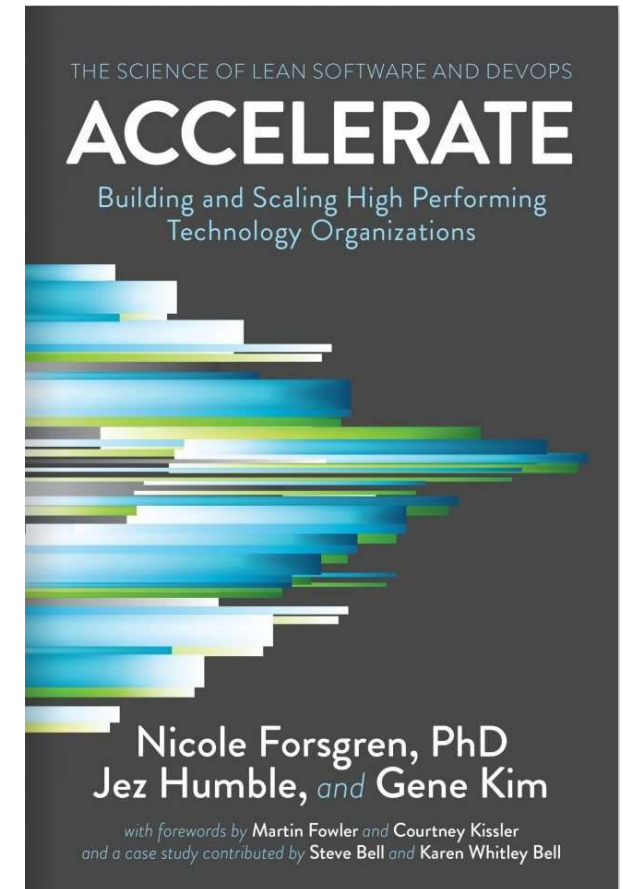
Org & Architecture: Accelerate

- Recommendation based on empiric study
 - Loose coupling to scale team
 - Allow teams to choose tools
 - Focus on developers and results
- ...not tools or technology



The Importance of Org: Accelerate


- Microservices support independence
- There might be alternatives, though



Conclusion: Culture & Organization

- Organization: architecture tool
- Architecture should drive organization
- Conway's Law 1968
- Domain-driven Design 2005
- Agile
- Accelerate 2019

Conclusion: Culture & Organization

- Microservices made idea popular. 
- Microservices: more independence. 
- No Focus on tools or technology
...but developers and results.
- Microservice: Just another (great!) tool

Are you a manager or a software architect?

**Are microservices only useful for
large teams?**

Technology & Additional Complexity

Challenges with Microservices

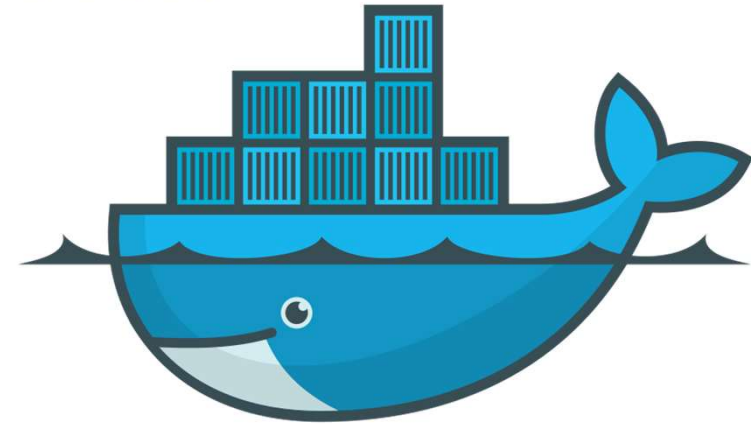
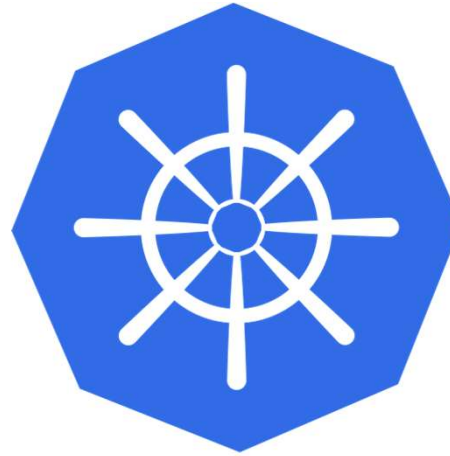
- Operations
- Infrastructure
- Visibility

When I Started with Microservices

- Virtual machine per service
- Lots of resource
- Hard to run locally
- Monitoring: not dynamic enough
- Log collection: only in prod due to cost

New Technologies: Platform

- Docker
- Kubernetes
- Service Mesh

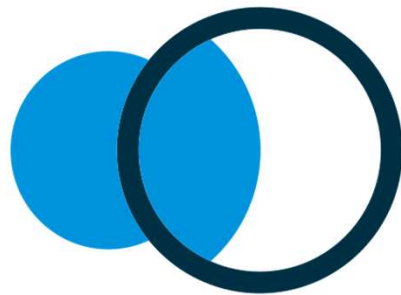


- PaaS
- Serverless



New Technologies: Operation

- Monitoring (e.g. Prometheus)
- Logging (Elastic, too many to mention)
- Visualization (Kiali)
- ...



New Technologies

- "But these are complex!"
- Are the old virtualization and monitoring systems trivial?
- Operations remains a challenge
might block you from microservices.

Rely on Public Cloud!
Infrastructure Managed for You!

Technologies: Developer Tools

- GraalVM
- Spring Boot
- ...

GraalVM™



- Too many microservices frameworks to mention all...

Technology & Complexity

- Lots of innovation
 - Containers
 - Serverless, PaaS
 - Microservices tools
-
- Will technology eliminate the complexity?

Conclusion

Conclusion

- Microservices = Hype
- Renaissance of modules and DDD
- Long-term architecture
- Help with continuous delivery
- Established org as architecture tool
- Profit from innovation

**Even if you don't do
microservices – some ideas are
still important!**

**Migration might mean that we
will have the old model for quite
some time**