Karlsruhe Institute of Technology

Software Engineering II

Prof. Dr. Ralf H. Reussner

Topic 06
Microservices

DSIS – DEPENDABILITY OF SOFTWARE-INTENSIVE SYSTEMS
KASTEL – INSTITUTE OF INFORMATION SECURITY AND DEPENDABILITY

dsis.kastel.kit.edu

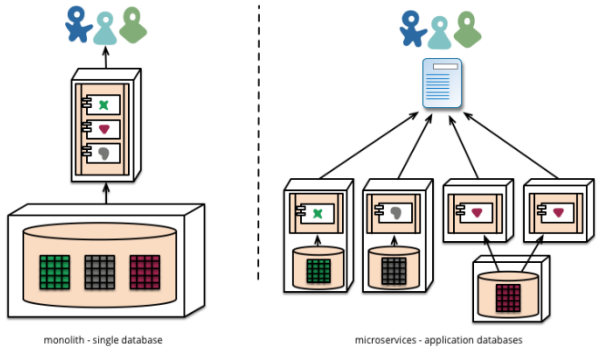
Overview on Next Lectures

Content: Microservices

- … as an architectural style
- … for developing a single application as suite of services
- … as independently deployable, scalable, manageable services

Learning goals: Students are able to

- characterize microservices
- know means to Componentize and decentralize an application
- understand internals of microservices
- know benefits and costs of applying the microservice architectural style



[https://martinfowler.com/articles/microservices.html]

Recap & Motivation

- Desire to build systems by plugging together components
- A component is a contractually specified building block for software which can be composed, deployed, and adapted without understanding its internals. [Reussner2016]
  - not necessarily black-box: information on component's internals may be provided for tools
  - effort for composition, deployment, or adaptation should be as low as possible.
- Microservice: separate components into user processes to maximize decoupling and independent deployability

What are Microservices?

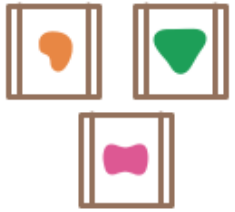
- **Microservice** is an architectural style.
- Microservice is an approach to develop a single application as a suite of small services
- **Primarily it is about scaling your development team,** secondarily your application.
- Each application
  - runs in its own process
  - communicates with lightweight mechanisms (e.g., HTTP resource API, REST principles)
- Services
  - are independently deployable
  - are independently scalable
  - can be written in different programming languages
  - can be managed by different teams

[https://martinfowler.com/articles/microservices.html]

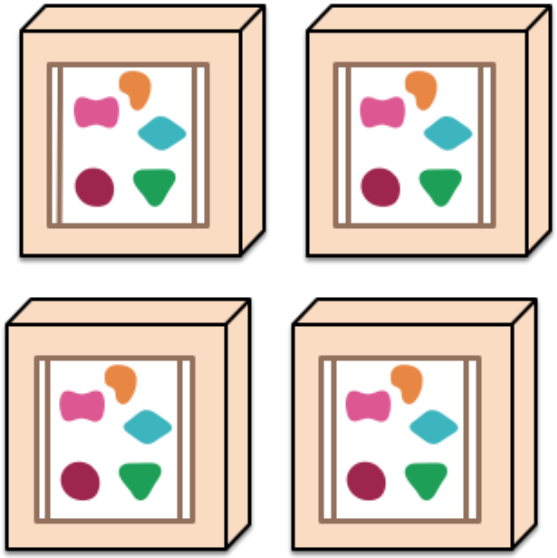
Monolith vs. Microservice

A monolithic application puts all its functionality into a single process...

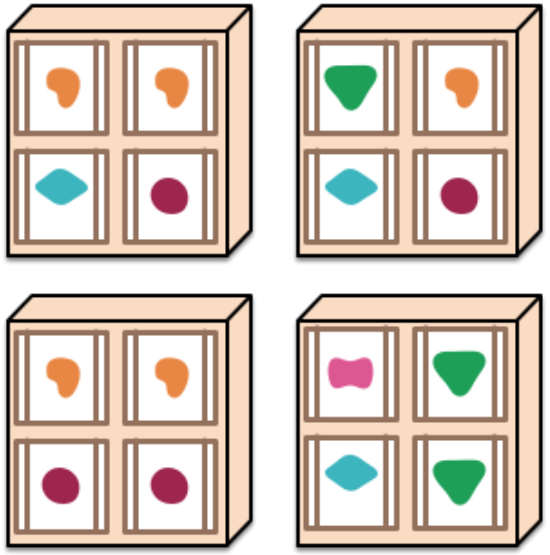
A microservices architecture puts each element of functionality into a separate service...



... and scales by replicating the monolith on multiple servers



... and scales by distributing these services across servers, replicating as needed.



https://martinfowler.com/articles/microservices.html

Goal: Decoupling for independent development and deployment

But: That can also be achieved with monoliths [Martin 2017]

Characteristics of a Microservice Architecture

- **What are characteristics of microservice architectures?**
  - _____
  - _____
  - _____
  - _____
  - _____
  - _____
  - _____
  - _____
  - _____

Characteristics of a Microservice Architecture

- **Note**: Not all microservice architectures have all the characteristics!
- But, most microservice architectures exhibit most characteristics:
  - Componentization via services
  - Organized around business capabilities
  - Products not projects
  - Smart endpoints and dumb pipes
  - Decentralized governance
  - Decentralized data management
  - Infrastructure automation
  - Design for failure
  - Evolutionary design

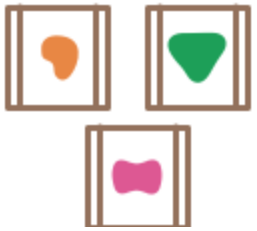
[https://martinfowler.com/articles/microservices.html]

Componentization via Services

- Microservice architectures can be broken down into services which **implement a user recognizable part of the functionality (“product”)** (i,e., no infrastructure or just GUI)
- Services are deployed components who communicate with a mechanism, such as a *web service request* or *remote procedure call*.
- Consequences from actually using components:
  - independently deployable
  - explicit component interface by using explicit remote call mechanisms

A monolithic application puts all its functionality into a single process...

A microservices architecture puts each element of functionality into a separate service...



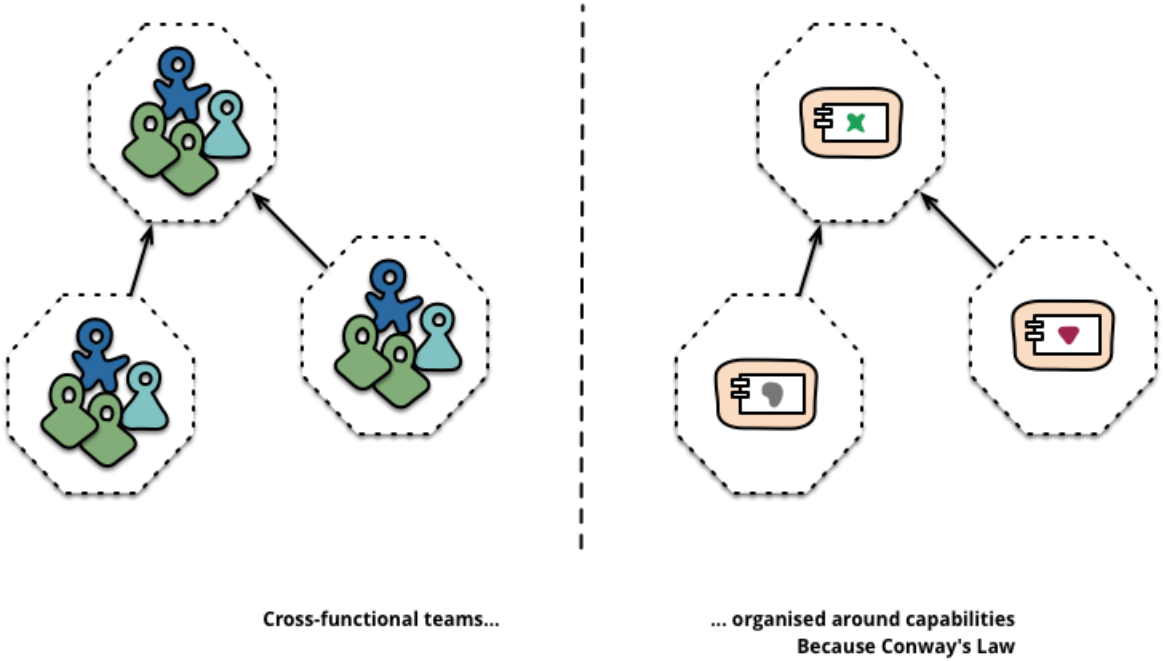
[https://martinfowler.com/articles/microservices.html]

Organized around Business Capabilities

- The microservice architecture splits up systems into services organized around business capability

➔ Thus, conforms to Conway's Law

“Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.” [Conway67]



[https://martinfowler.com/articles/microservices.html]

Products not Projects

- How can software assist its users to enhance the business capability?
- The product mentality ties in with the linkage to business capabilities
- Rather than the software as a set of functionality to be completed
- The small granularity of services makes it easier to create the personal relationships between service developers and their users

- Often: Teams are also responsible to run a service
  - DevOps

[https://martinfowler.com/articles/microservices.html]

amazon

"You build it, you run it. This brings **developers** … into day-to-day **contact with the customer**. This customer feedback loop is essential for improving the quality of the service.",
Werner Vogels, CTO Amazon

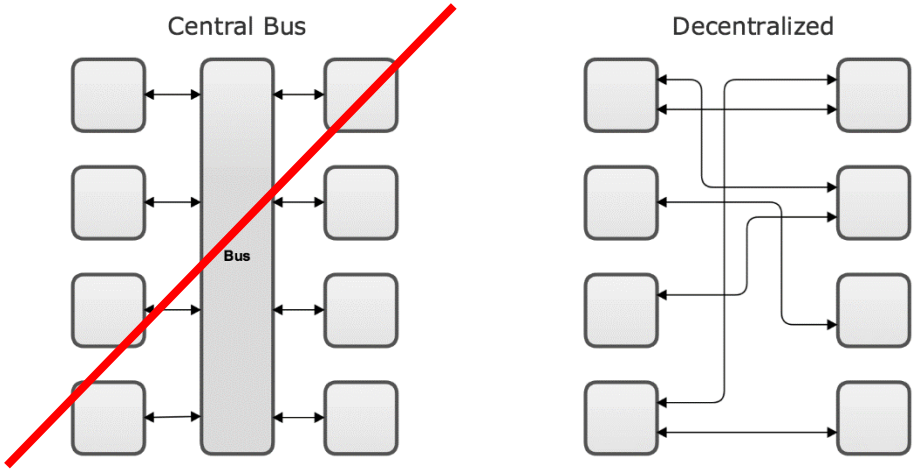
Source: ACM Interview with Werner Vogels, June 2006

Alex Popov, at https://www.slideshare.net/CodeVoyagersSofia/you-build-it-you-run-it

Smart Endpoints and Dumb Pipes

- Applications built from microservices aim to be as decoupled and as cohesive as possible
- They own their own domain logic and act more as filters in the classical Unix sense
  - receiving a request
  - applying logic as appropriate
  - producing a response
- These are choreographed using simple RESTish protocols

[https://martinfowler.com/articles/microservices.html]



https://medium.com/@nathankpeck/microservice-principles-smart-endpoints-and-dumb-pipes-5691d410700f

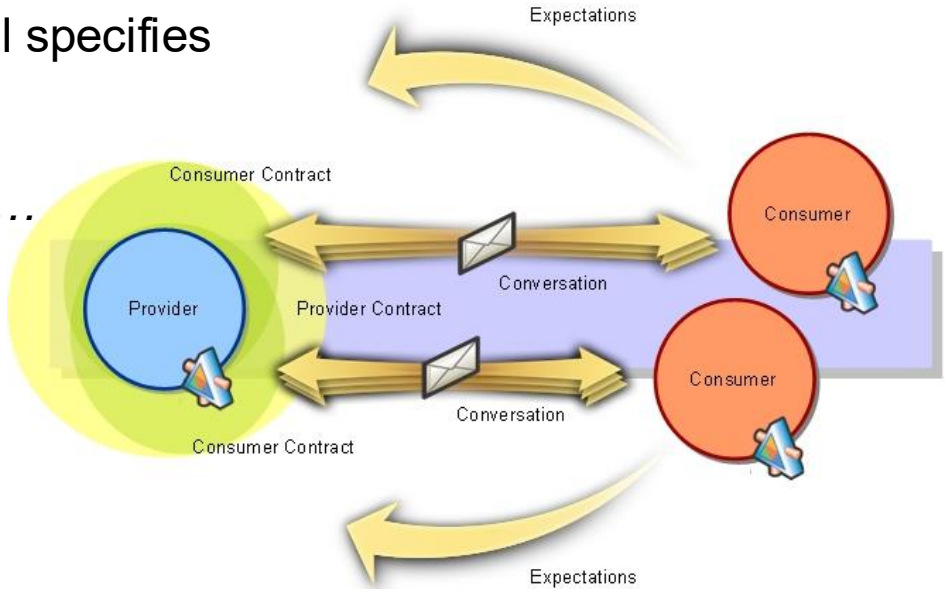
Decentralized Governance

- Developers of microservices prefer idea of producing useful tools that other developers can use to solve similar problems
- These tools are usually harvested from implementations and shared with a wider group
- Patterns like *Tolerant Reader* and *Consumer-Driven Contracts* are often applied to microservices
- These patterns aid service contracts in evolving independently.

[https://martinfowler.com/articles/microservices.html]

Tolerant Reader and Consumer-Driven Contracts

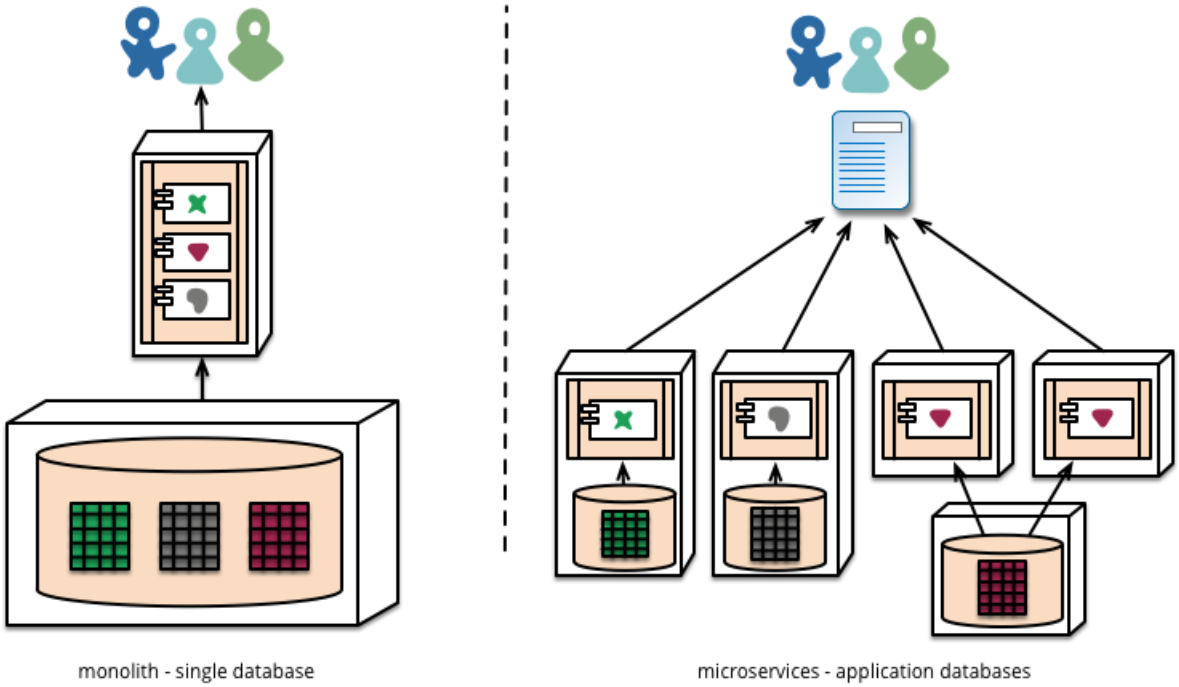
- Tolerant Reader:
  Ignore unknown elements and make minimum assumptions to increase robustness.
- Consumer-Driven Contracts:
  - Service implements union of all consumers needs the provider has to satisfy
  - Call before actual service call specifies the actually needed service
  - Contract: *Data Schema, Service interfaces, Policies, …*



[https://martinfowler.com/bliki/TolerantReader.html]
[https://martinfowler.com/articles/consumerDrivenContracts.html]

Decentralized Data Management

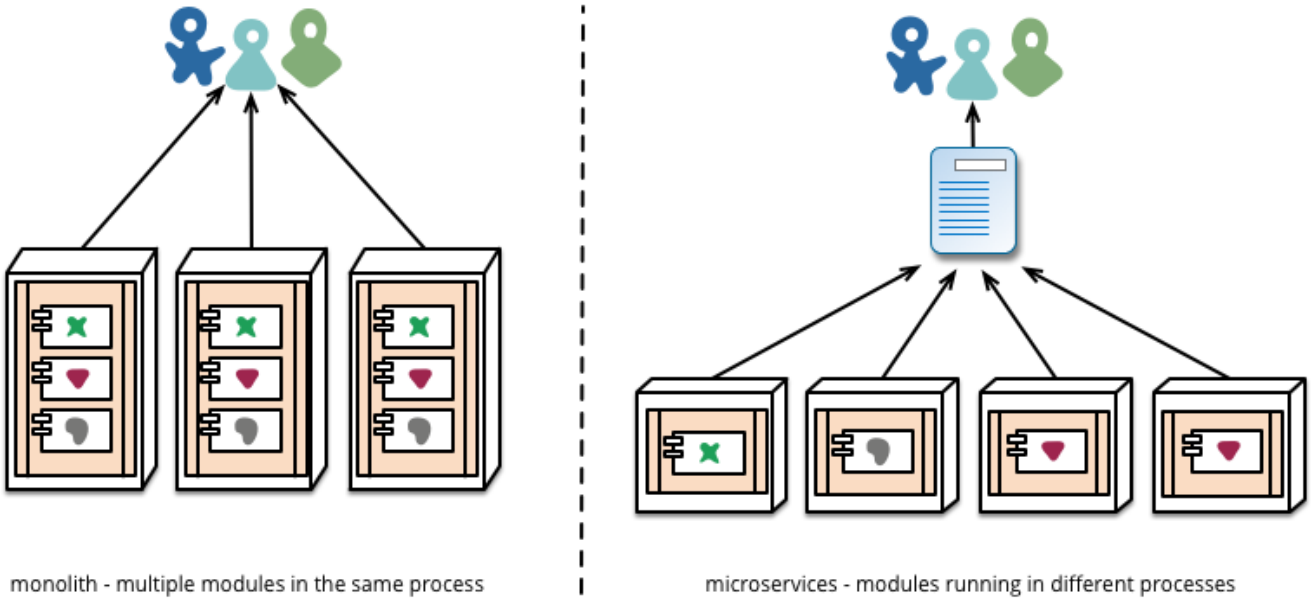
- Microservices prefer letting each service manage its own database
  - either different instances of the same database technology
  - or entirely different database systems
- Yet, managing consistency becomes challenging



[https://martinfowler.com/articles/microservices.html]

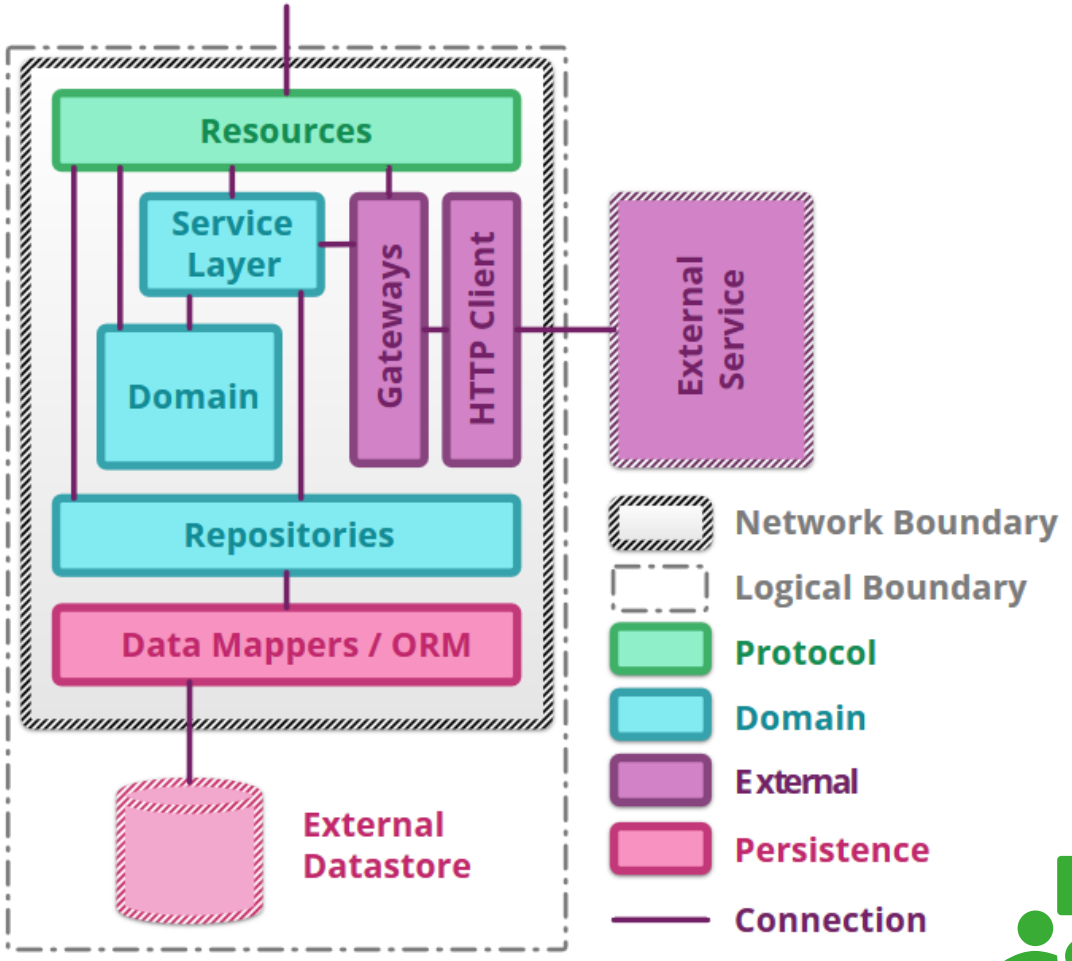
Infrastructure Automation

- Many of the products or systems being build with microservices are being built by teams with extensive experience of *Continuous Delivery* and *Continuous Integration*
- Teams, building software this way, make extensive use of infrastructure automation techniques



[https://martinfowler.com/articles/microservices.html]

Internals of Microservices



By Toby Clemson, from https://martinfowler.com/articles/microservice-testing/#anatomy-connections
Consider reading his description of microservices at https://martinfowler.com/articles/microservice-testing/#definition

Clean Architecture and Microservices

- „Component“ here could be a Microservice

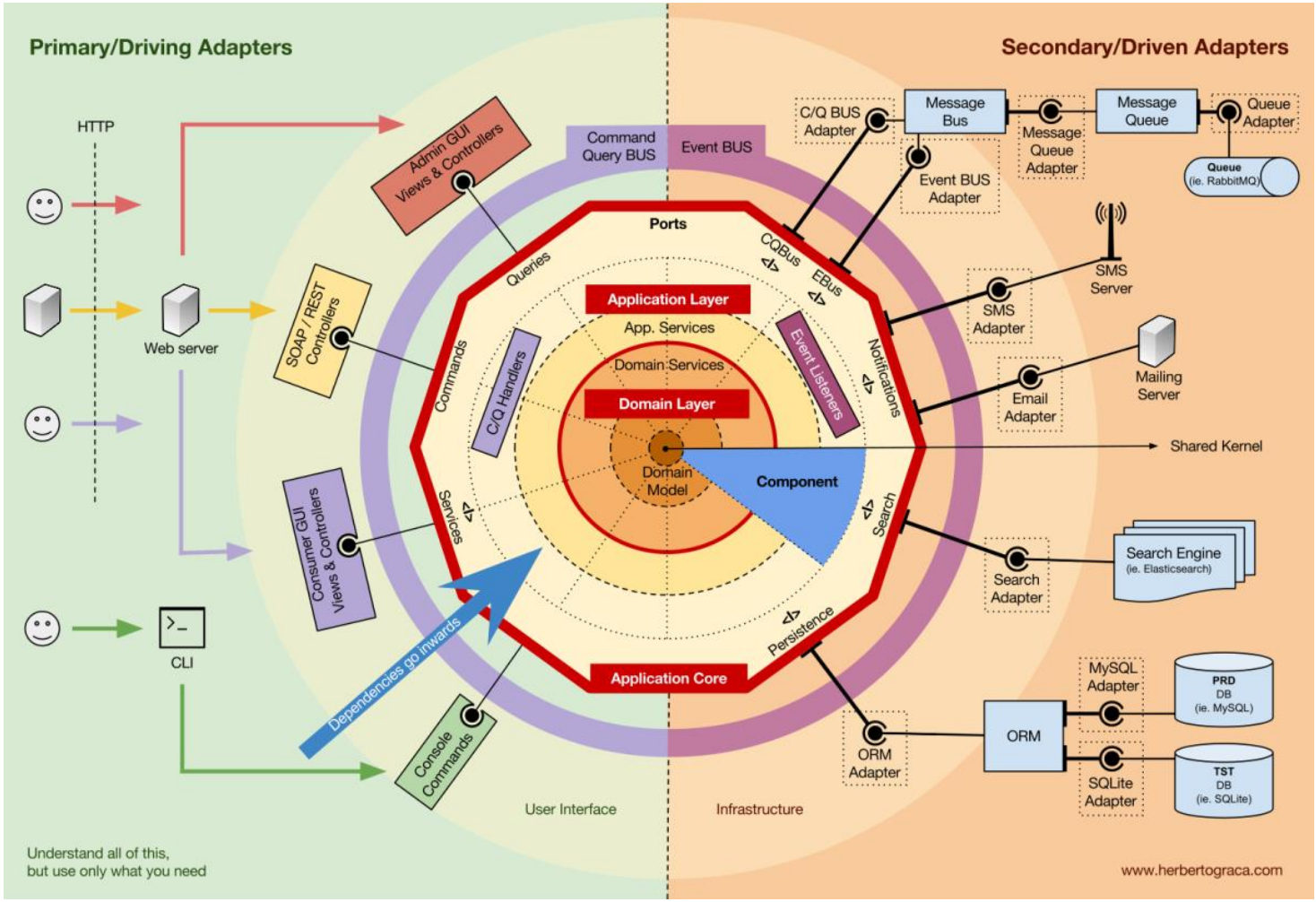


Image from @herbertograca
https://herbertograca.com/2017/11/16/explicit-architecture-01-ddd-hexagonal-onion-clean-cqrs-how-i-put-it-all-together/

Design for Failure

- Using services as components ➔ applications need to be designed so that they can tolerate the failure of services
- Since services can fail at any time, it's important to be able to detect the failures quickly and, if possible, automatically restore service
- Microservice applications put a lot of emphasis on online monitoring of the application, checking both architectural elements (how many requests per second is the database getting) and business relevant metrics (such as how many orders per minute are received).

- Tools to test for resilience, *e.g. Netflix' Simian Army*
- Patterns for resilience, *e.g. Circuit Breaker*

[https://martinfowler.com/articles/microservices.html]
image: https://en.wikipedia.org/wiki/Chaos_engineering#/media/File:Netflix_simianarmy-768x797.jpg]

Evolutionary Design

- Evolutionary design background:
  service decomposition as further tool to enable application developers to control changes in their application
- Whenever trying to break software system into components, developers need to decide how to divide the pieces
- What are the principles used to decide how to slice up an application?
  - The key property of a component is the notion of independent replacement and upgradeability
  - Looking for points where rewriting a component would not affect its collaborators.
  - User functionality as dimension, not technology (as in n-tier architecture)
- Try to avoid manual versioning

[https://martinfowler.com/articles/microservices.html]

When to use what?

Microservices provide benefits…

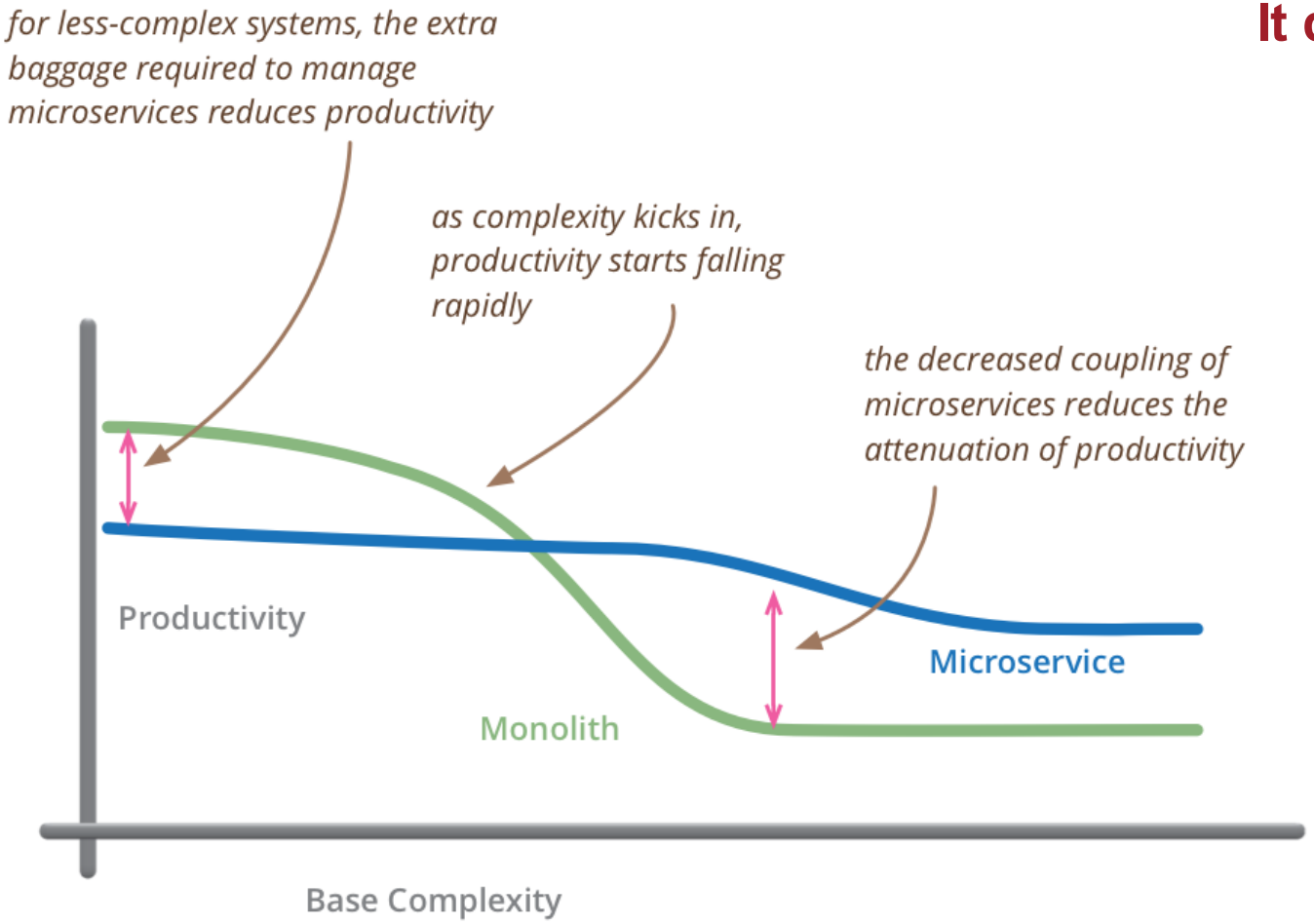
- **Strong Module Boundaries:**
  Microservices reinforce modular structure, which is particularly important for larger teams.
- **Independent Deployment:**
  Simple services are easier to deploy, and since they are autonomous, are less likely to cause system failures when they go wrong.
- **Technology Diversity:**
  With microservices you can mix multiple languages, development frameworks and data-storage technologies.

…but come with costs

- **Distribution:**
  Distributed systems harder to program, since remote calls are slow and are might always fail.
- **Eventual Consistency:**
  Maintaining strong consistency is extremely difficult for distributed system, which means everyone has to manage eventual consistency.
- **Operational Complexity:**
  You need a mature operations team to manage lots of services, which are being redeployed regularly.

When to use what?

It depends!



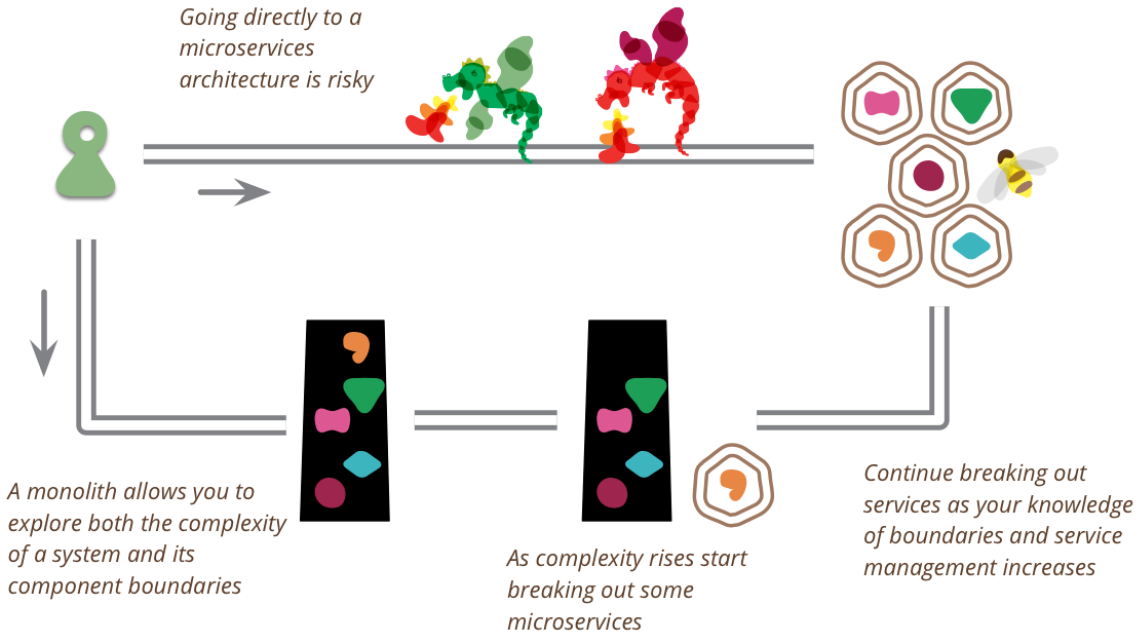
[https://martinfowler.com/bliki/MicroservicePremium.html]

but remember the skill of the team will outweigh any monolith/microservice choice

How to begin? Different opinions

Option 1: Start with a clean monolith, separate later

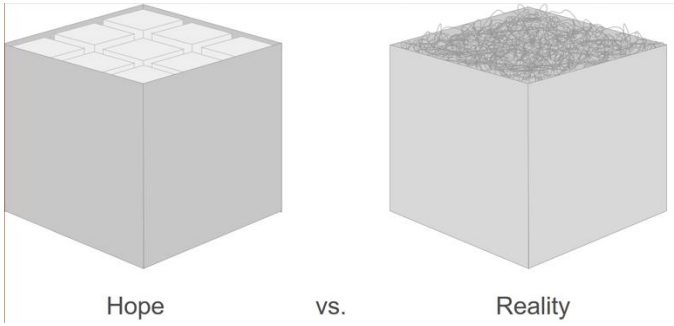
- YAGNI
- Service boundaries may be unclear at the beginning
- Prepare with clean architecture



Option 2: Start with Microservices

- Enforces modularity
- Monolith will get messy

[https://martinfowler.com/bliki/MonolithFirst.html]

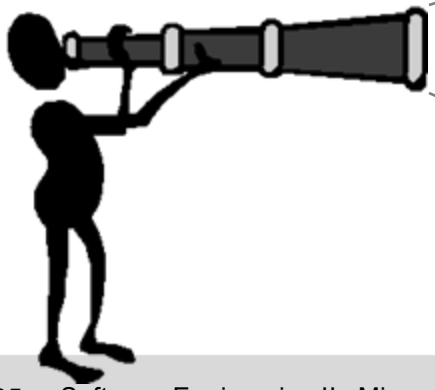


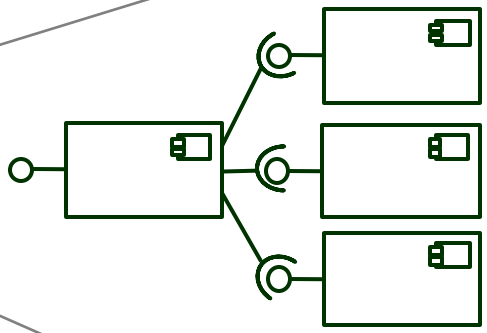
Conclusion

- Microservice is an architectural style.
- Microservice is an approach to developing a single application as a suite of small services.
- Microservices are independently deployable, scalable, and developable services

Students know

- how microservices are characterized
- means to componentize and decentralize an application
- the internal structure of a microservice
- benefits and costs





Enterprise Application Patterns

References

- Web resources from Martin Fowler
  https://martinfowler.com/articles/microservices.html
  (see links on individual slides)

- [Martin 2017] Robert C. Martin, *Clean architecture: a craftsman's guide to software structure and design*. Prentice Hall Press, 2017.