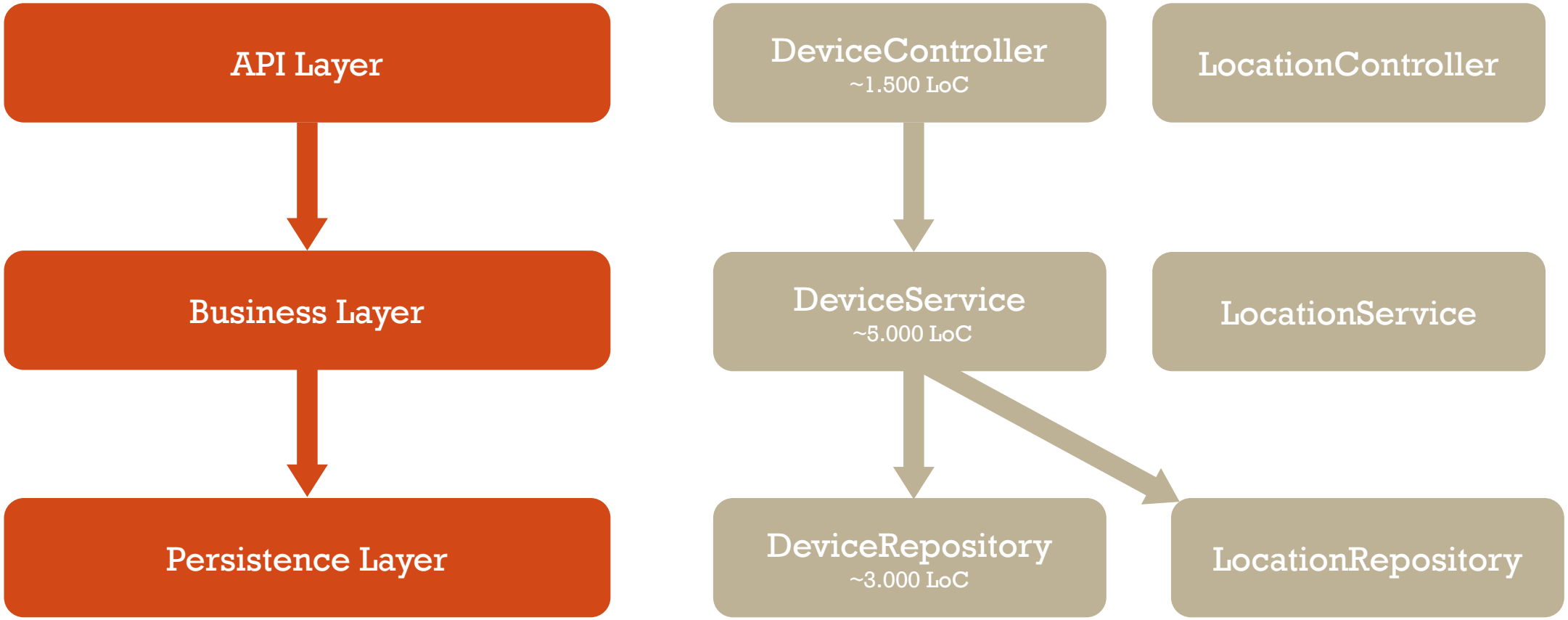
# VERTICAL SLICES

https://etc.ch/ZJj8

- Show sample app with Controller, Service and Repository


API Layer
Business Layer
Persistence Layer
DeviceController
~1.500 LoC
LocationController
DeviceService
~5.000 LoC
LocationService
DeviceRepository
~3.000 LoC
LocationRepository

*If you have the feeling that something will break at one end if you change something elsewhere…*

***…then maybe it's time to think about refactoring your architecture.***

ce*

t touch anything.

# ALEXANDER PABINGER

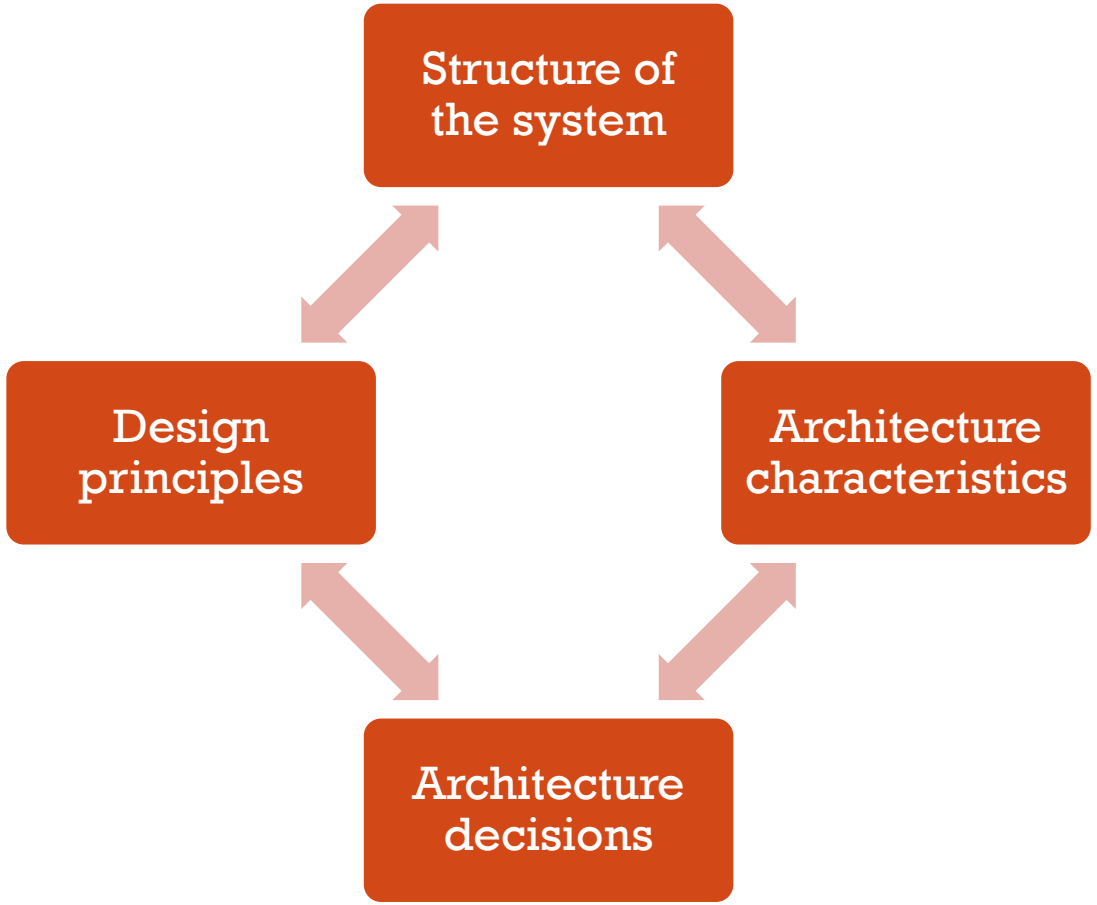
# WHAT IS SOFTWARE ARCHITECTURE?

- Hard to define
- Structure of the system

*What architecture do you use?*

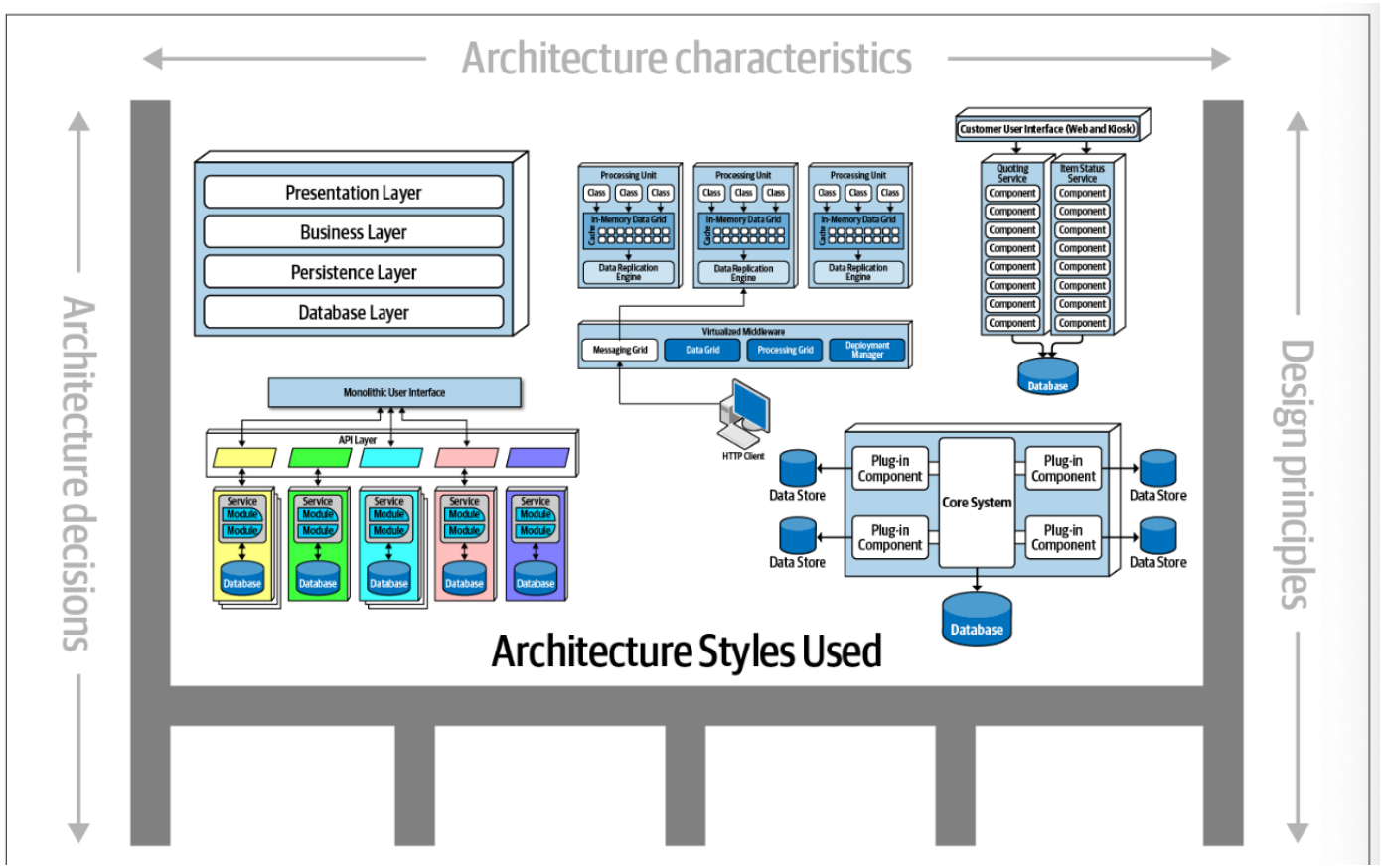
- Clean Architecture, Hexagonal, Layered, Microservices

...but




Feature Driven Architecture by Alexander Pabinger

*source: Fundamentals of Software Architecture (Mark Richards & Neal Ford)*



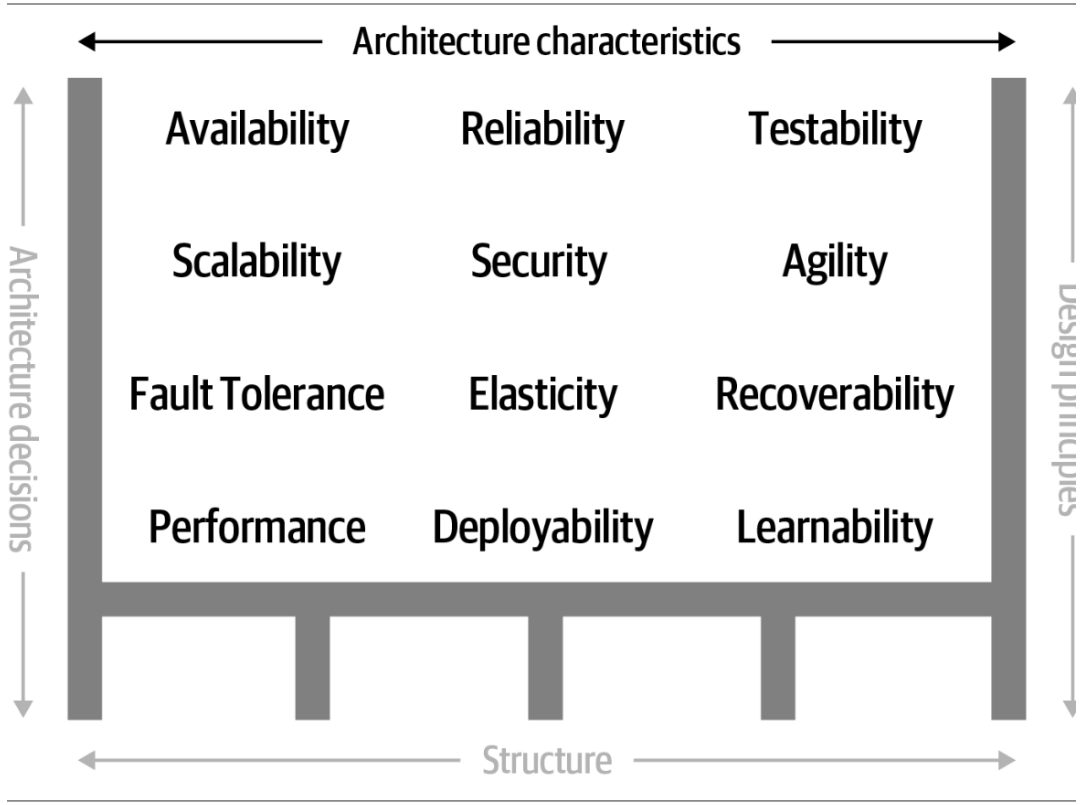

# STRUCTURE OF THE SYSTEM

e.g. Microservices, Layered, Microkernel

Feature Driven Architecture by Alexander Pabinger

*source: Fundamentals of Software Architecture (Mark Richards & Neal Ford)*

# ARCHITECTURE CHARACTERISTICS



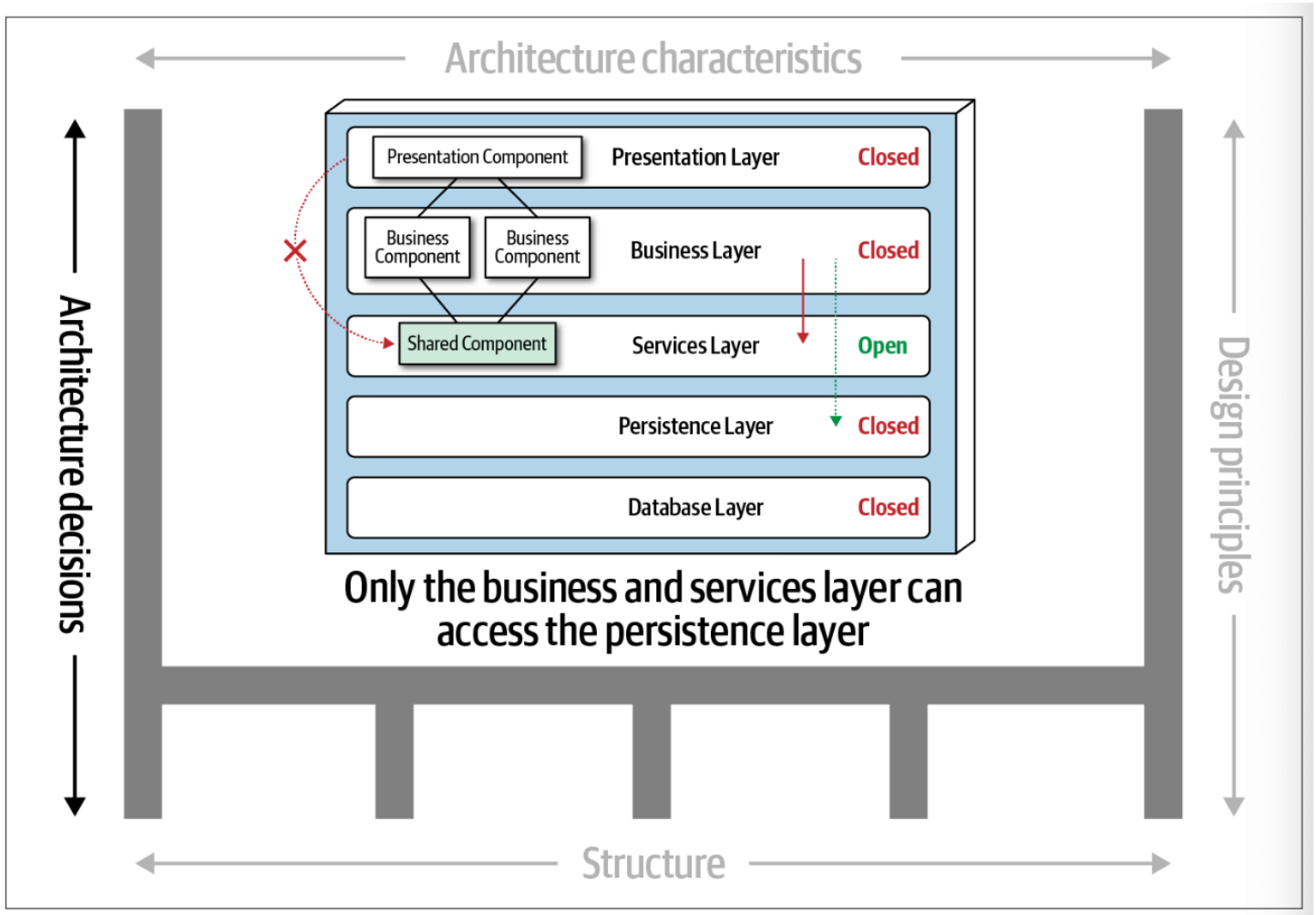

- Success criteria of a system
- Required for the system to function properly

Feature Driven Architecture by Alexander Pabinger

*source: Fundamentals of Software Architecture (Mark Richards & Neal Ford)*

# ARCHITECTURE DECISIONS

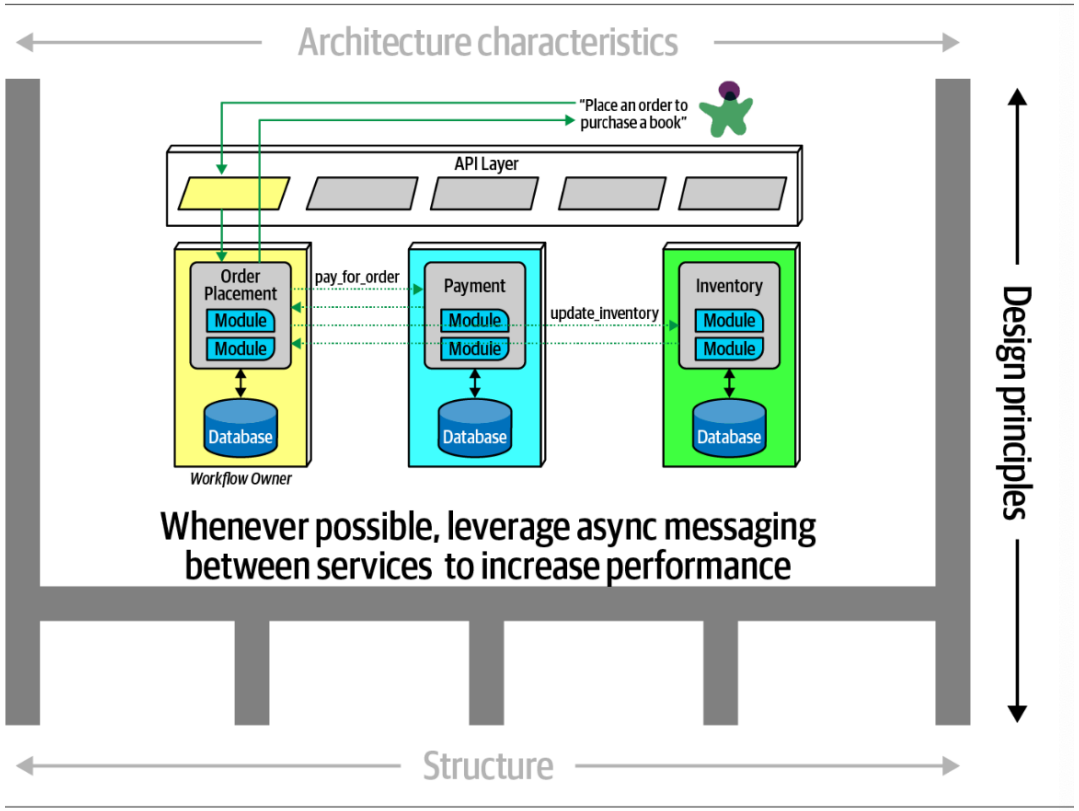
- Strict Rules
- Can be enforced with tests




Feature Driven Architecture by Alexander Pabinger

*source: Fundamentals of Software Architecture (Mark Richards & Neal Ford)*

# DESIGN PRINCIPLES




- Rather a guideline than a rule
- e.g. use Async if possible
- Ensure in pull requests

Feature Driven Architecture by Alexander Pabinger

*source: Fundamentals of Software Architecture (Mark Richards & Neal Ford)*

# EXPECTATIONS OF AN ARCHITECT

- Lead developer? Expert programmer? …
- Core expectations
  - Make architecture decisions
  - Continually analyze the architecture
  - Keep current with latest trends
  - Have business domain knowledge
  - Possess interpersonal skills
  - Understand and navigate politics

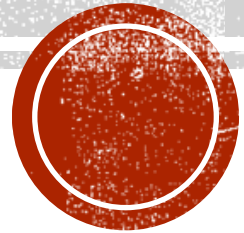


*source: Fundamentals of Software Architecture (Mark Richards & Neal Ford)*

# LAWS OF SOFTWARE ARCHITECTURE

- *"You always die some kind of death" – Philipp Pendelin, softaware*
- *"Everything in software architecture is a trade-off."*
- *"If an architect thinks they have discovered something that isn't a trade-off, more likely they just haven't identified the trade-off yet."*
- *"Why is more important than how."*

# COHESION

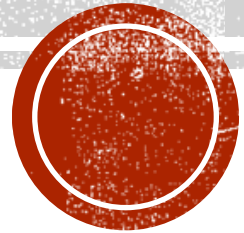


# COHESION

- “Degree to which the elements inside a module belong together” (Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design)

- Form building blocks that have a stronger relationship to each other
- Code that belongs together, stays together

# COUPLING

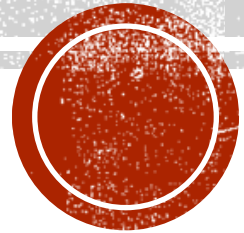


# COUPLING

- Degree of interdependence between software modules (ISO/IEC/IEEE 24765:2010 systems and software engineering – Vocabulary)

- Higher dependency means more likely destabilizing effects

# CQS & CQRS



# CQS VS CQRS

## CQS

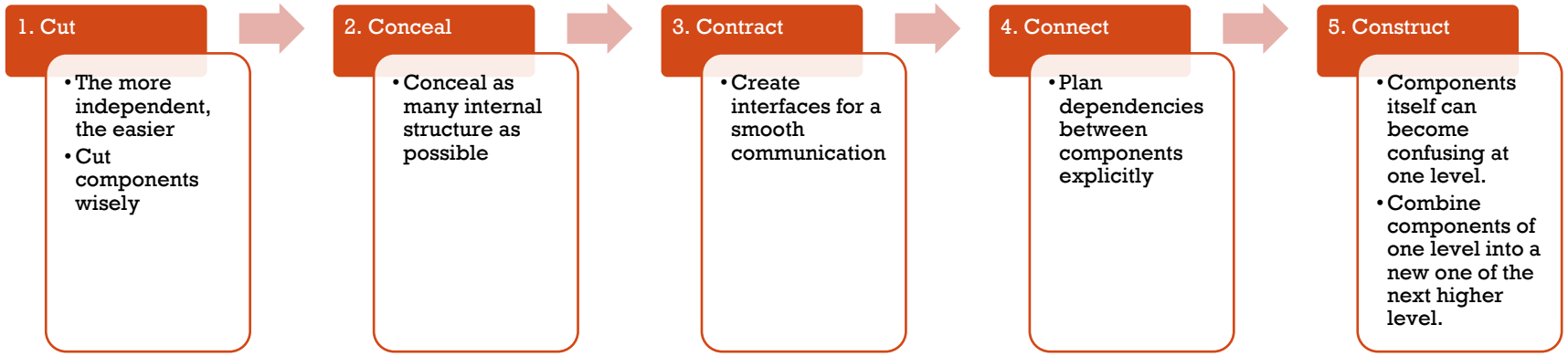
- Command-Query-Separation
- Split methods into mutations and queries
- Same data store

## CQRS

- Command-Query-Responsibility-Segregation
- Split the entire system into mutations and queries
- Different data stores or read models

# 5C DESIGN

- Not for compilers or interpreters, they don't care.
- Aimed to split software in small parts for human beings to understand the system and keep it maintainable

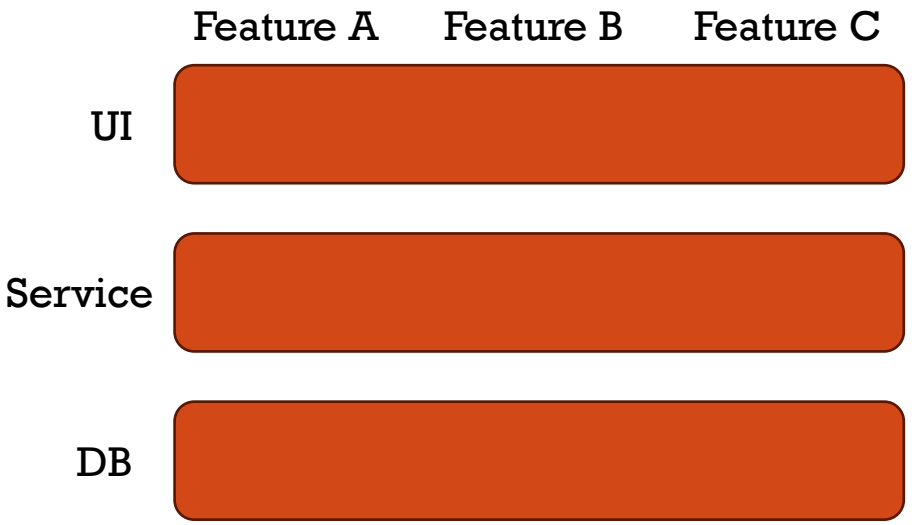




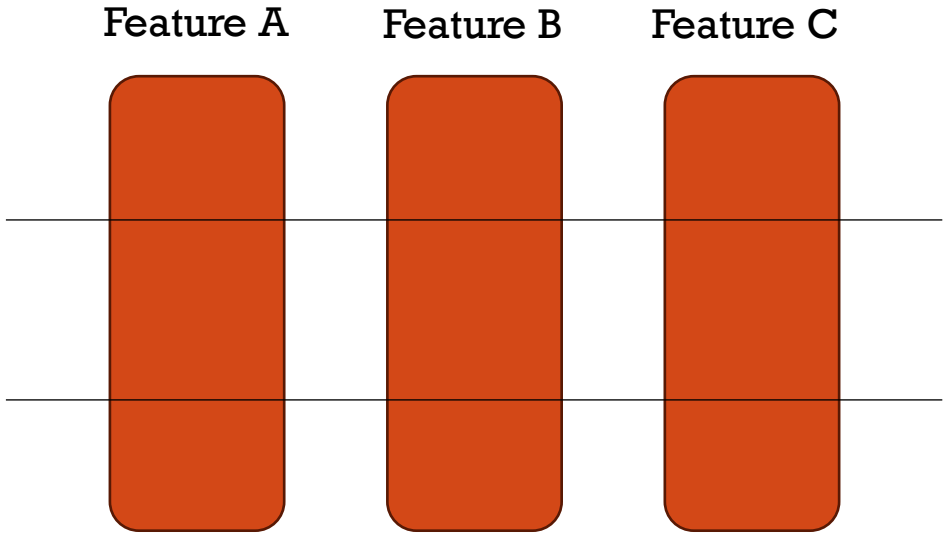

*source: Modulare Softwarearchitektur (Herbert Dowalil) - https://5c-design.info/*

# 5C DESIGN | CUT

## Horizontal




## Vertical






*source: Modulare Softwarearchitektur (Herbert Dowalil) - https://5c-design.info/*

# VERTICAL CUT = VERTICAL SLICES

- Vertical slice = Use Case
- Build business capabilities
- Group classes, methods, files per feature
- Contains all code to fulfill a feature
- May share code: Services, Domain, DTOs
- Not to confuse with Clean or Onion Architecture: Coupling vs Cohesion

# PROS AND CONS

## PROS

- Improved cohesion
- Easy to extend or update features
- Less likely that features impact other features
- Use the same programming model as always (no pub/sub, messaging, etc. needed)
- Infrequent Merge Conflicts
- Dev Experience

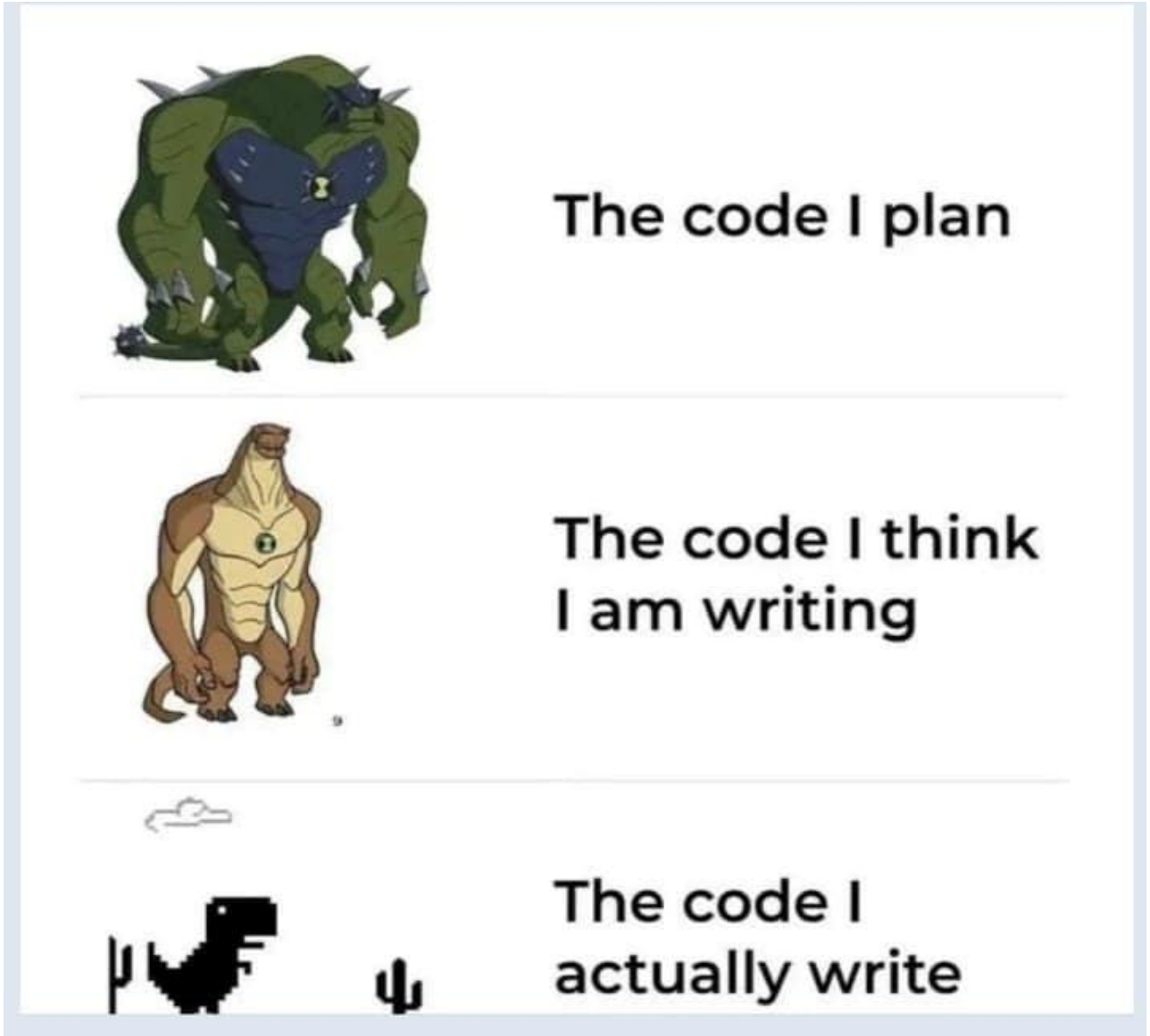
## CONS

- You don’t ”get rid” of coupling
- Code duplication
- …

# SAMPLE

- Show sample application
- Create Decorator for Cross-Cutting-Concern

source:
https://economistwritingeveryday.com/2024/02/17/programmer-pain-in-memes/



# TECHNICAL ASPECTS OF VSA

- It's about business capabilities, not about Architecture structure like Microservices, Monolith, Modulith, etc.
- Business capability = the WHAT? or the capacity to perform a unique business activity
- VSA can be used in a monolith, in a modulith, etc
- VSA can be used to change a Monolith to a Modulith or Microservice if needed

# USE REPOSITORY?

- It depends
- Decide if you need an abstraction for DbContext
- How we did it?
- Inject and use DbContext in all Query Handlers
- What about command handlers? How to unit test them?
- Data access service per Command Handler (e.g. UpdateNameCommandHandler, UpdateNameDataAccessor)

# It's all about ETC!

## easy to change

*(The pragmatic programmer – David Thomas and Andrew Hunt)*

# THANK YOU!

Don’t forget to connect on LinkedIn

Link to Github