PROFESSIONAL SOFTWARE ENGINEERING

PSE SWE LE 4 und 5 - Domain Driven Design

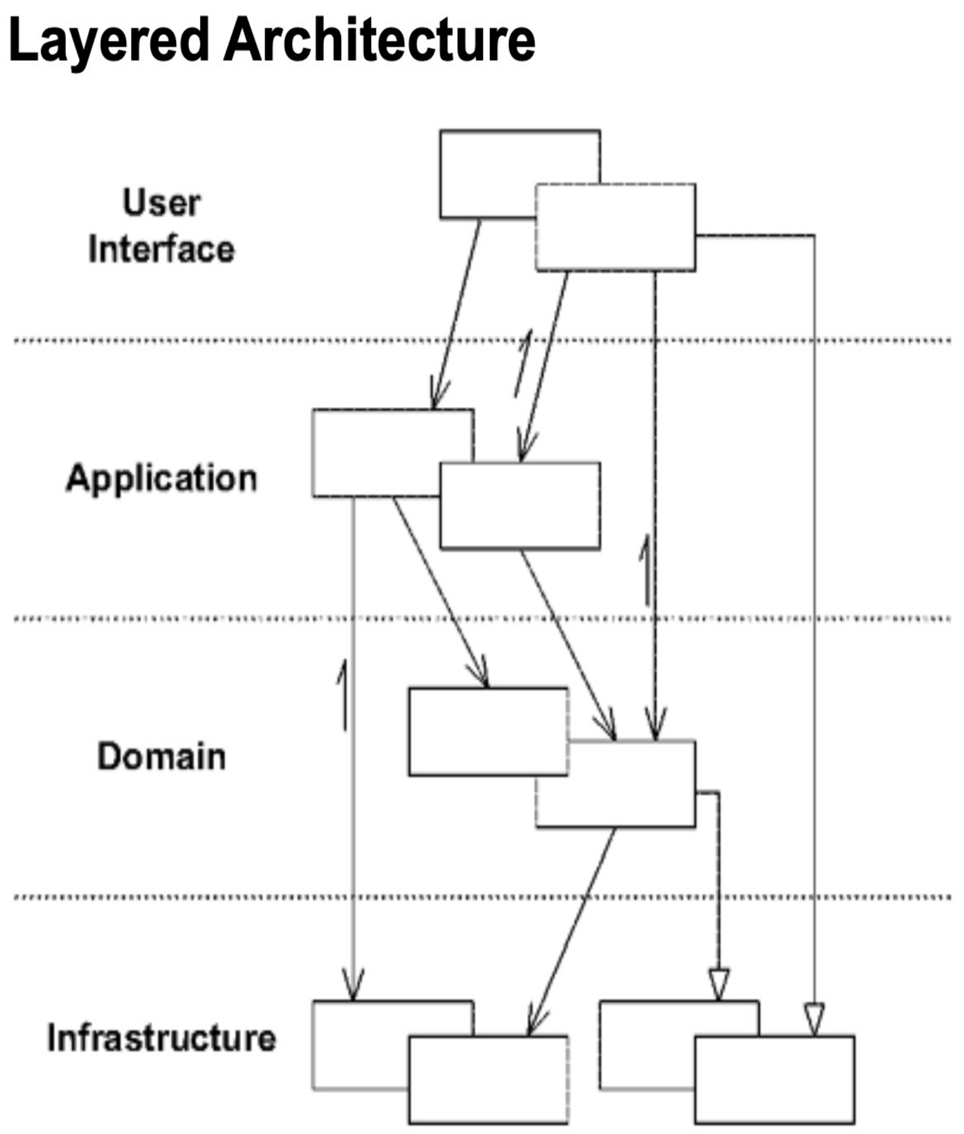
Architekturmuster

Dominik Neumann

Knowledge Foundation
@ Reutlingen University

LAYERED ARCHITECTURES

LAYERED ARCHITECTURE

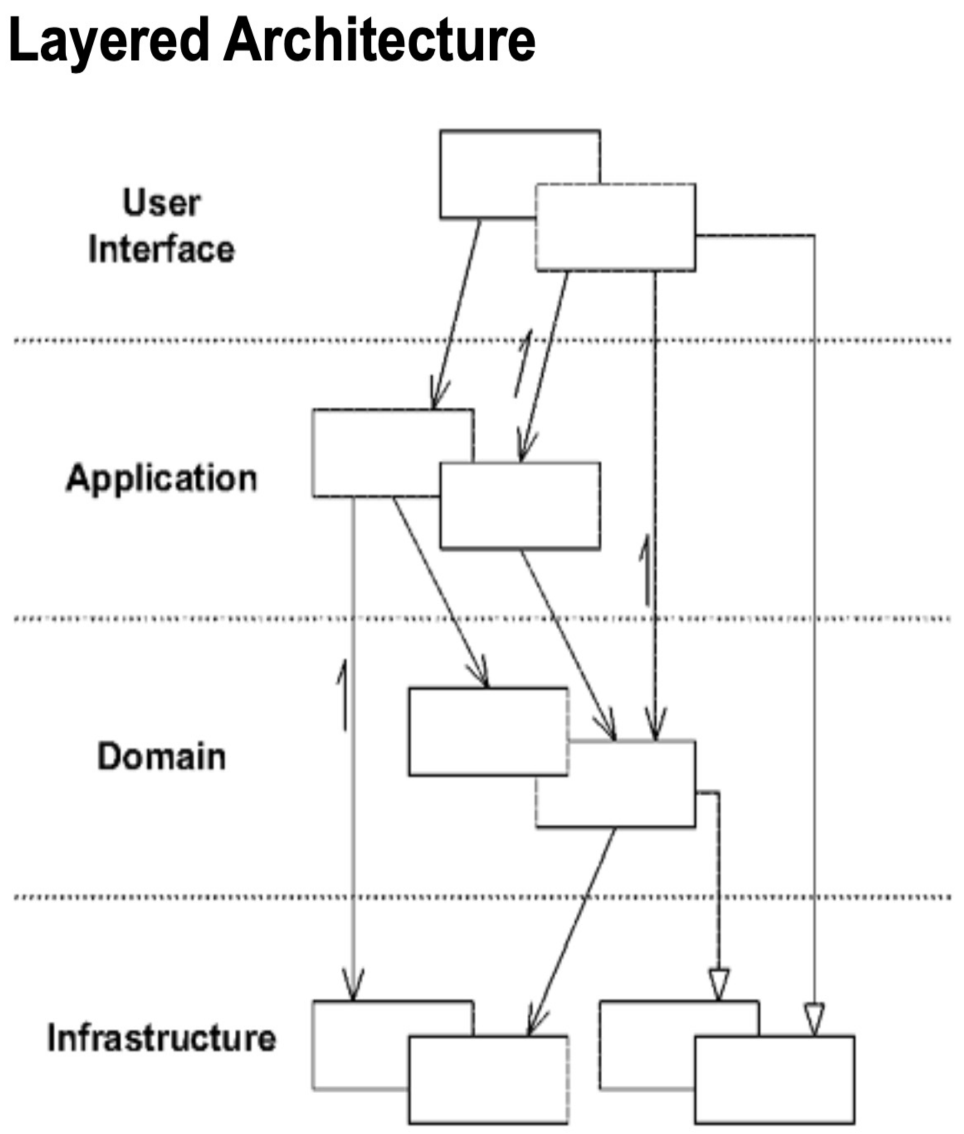


User Interface (or Presentation Layer):

Responsible for showing information to the user and interpreting the user's commands. The external actor might sometimes be another computer system rather than a human user.

Quelle: Evans, p.68 Layered Architecture

LAYERED ARCHITECTURE



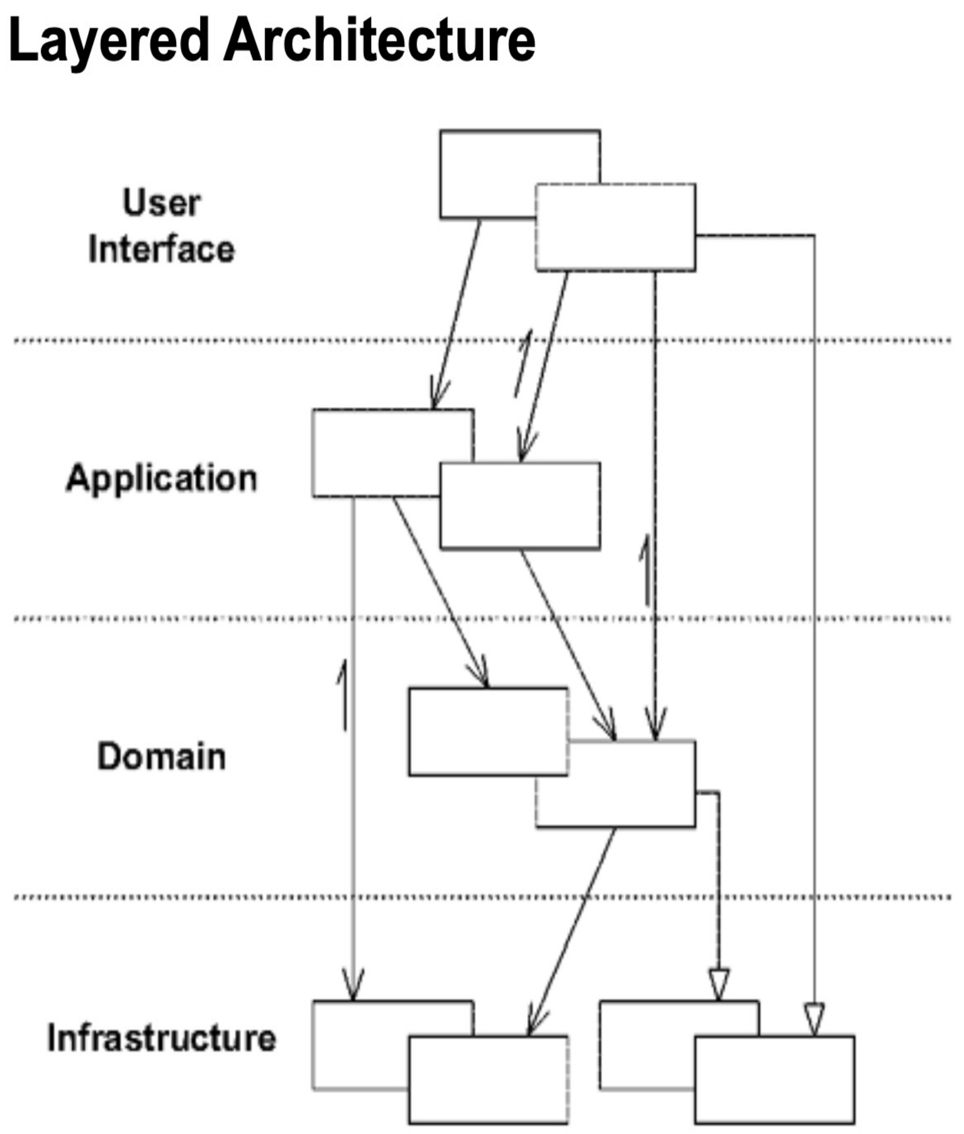
Application Layer:

Defines the jobs the software is supposed to do and directs the expressive domain objects to work out problems. This layer is kept thin:

- It does not contain business rules or knowledge, but only coordinates tasks and delegates work to collaborations of domain objects in the next layer down.
- It does not have state reflecting the business situation, but it can have state that reflects the progress of a task for the user or the program.

Quelle: Evans, p.68 Layered Architecture

LAYERED ARCHITECTURE



<u>Domain Layer (or Model Layer):</u>

Responsible for representing concepts of the business, information about the business situation, and business rules. State that reflects the business situation is controlled and used here, even though the technical details of storing it are delegated to the infrastructure.

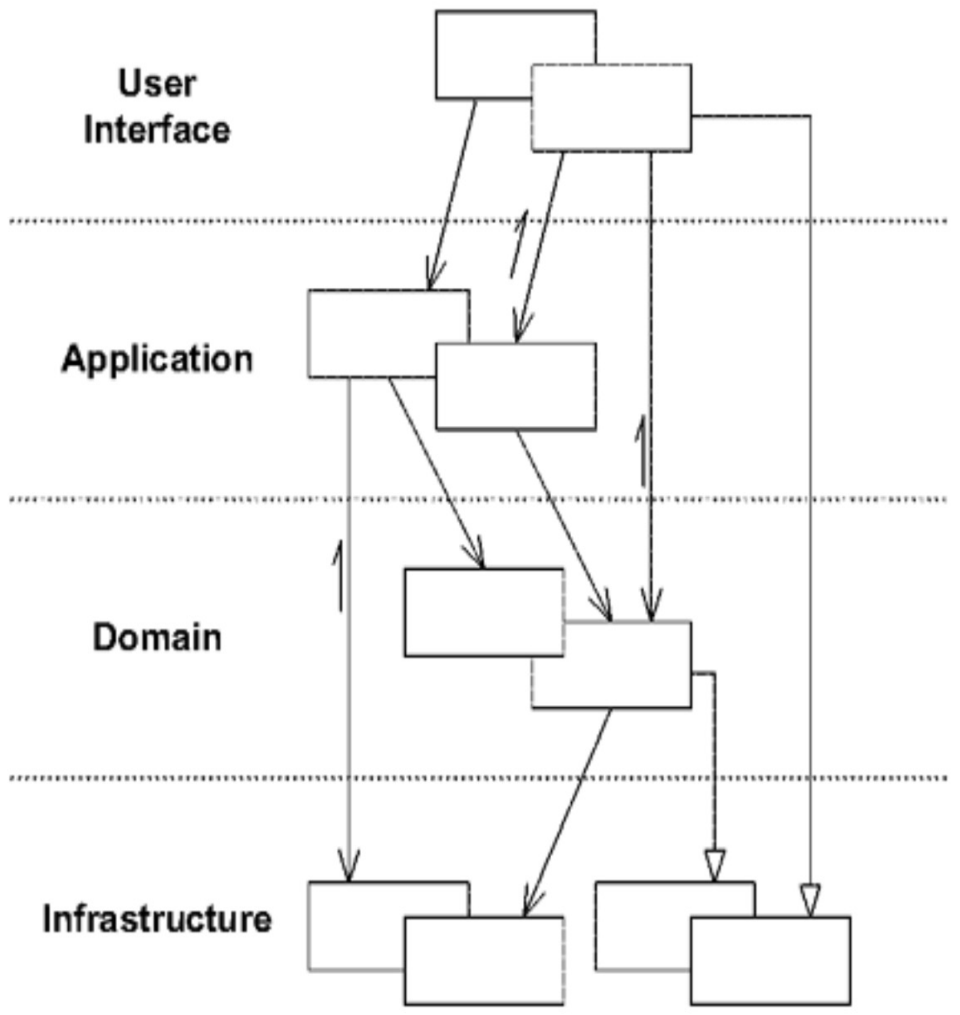
This layer is the heart of business software.

Quelle: Evans, p.68 Layered Architecture

LAYERED ARCHITECTURE

Knowledge Foundation
@ Reutlingen University

Layered Architecture



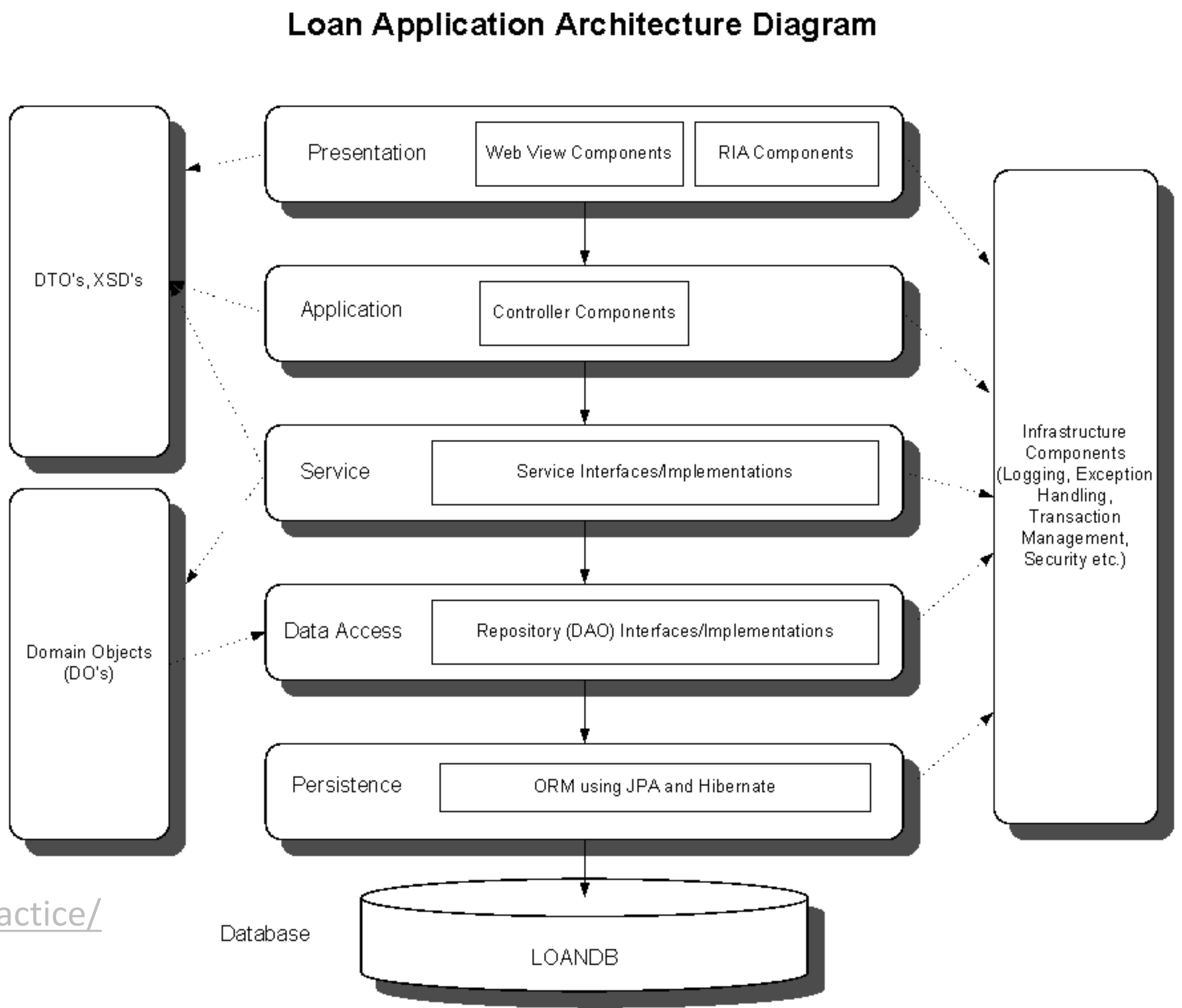
Infrastructure Layer:

Provides generic technical capabilities that support the higher layers: message sending for the application, persistence for the domain….

Quelle: Evans, p.68 Layered Architecture

LAYERED ARCHITECTURE

- Beispiel

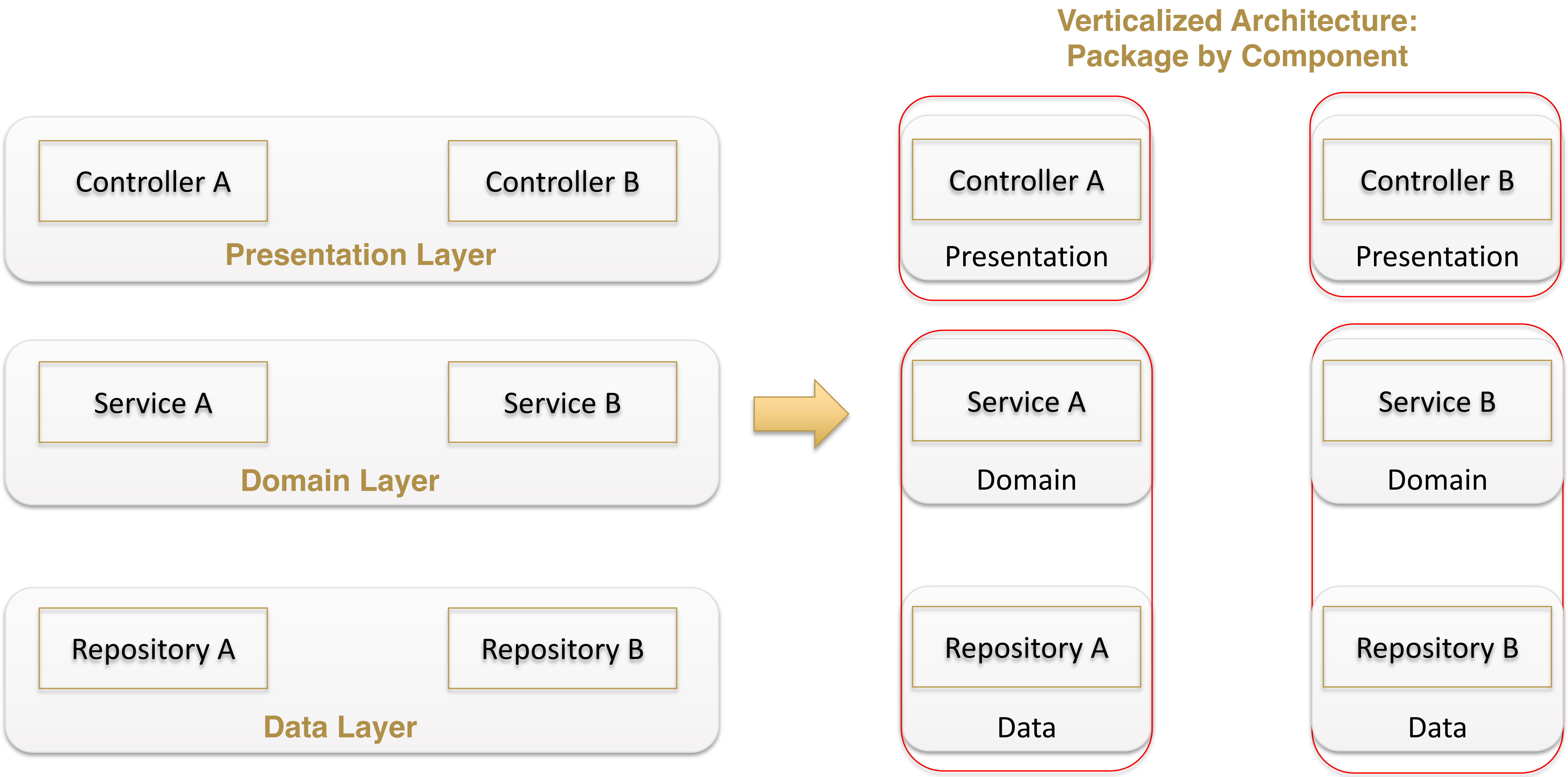

Loan Application Architecture Diagram
DTO's, XSD's
Domain Objects (DO's)
Presentation
Web View Components
RIA Components
Application
Controller Components
Service
Service Interfaces/Implementations
Data Access
Repository (DAO) Interfaces/Implementations
Persistence
ORM using JPA and Hibernate
Infrastructure Components (Logging, Exception Handling, Transaction Management, Security etc.)
Database
LOANDB

https://www.infoq.com/articles/ddd-in-practice/

ANTI-PATTERN:
PACKAGE BY LAYER

PACKAGE BY LAYER

Knowledge Foundation @ Reutlingen University

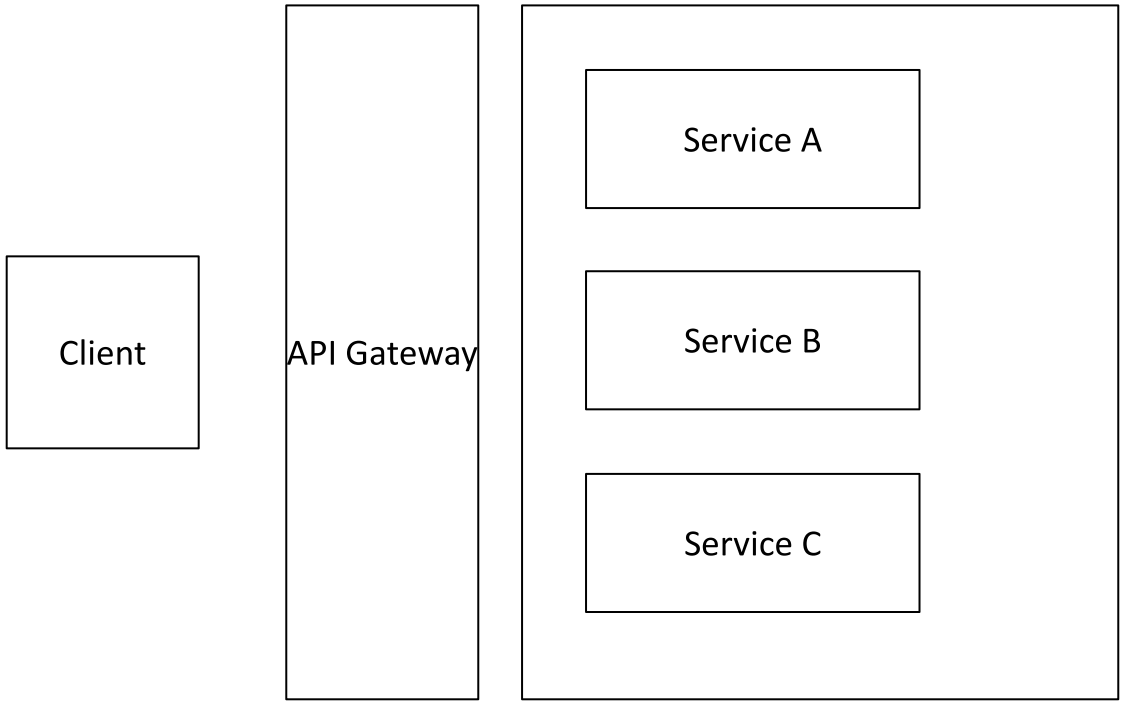


Knowledge Foundation
@ Reutlingen University

MICROSERVICES

MICROSERVICES

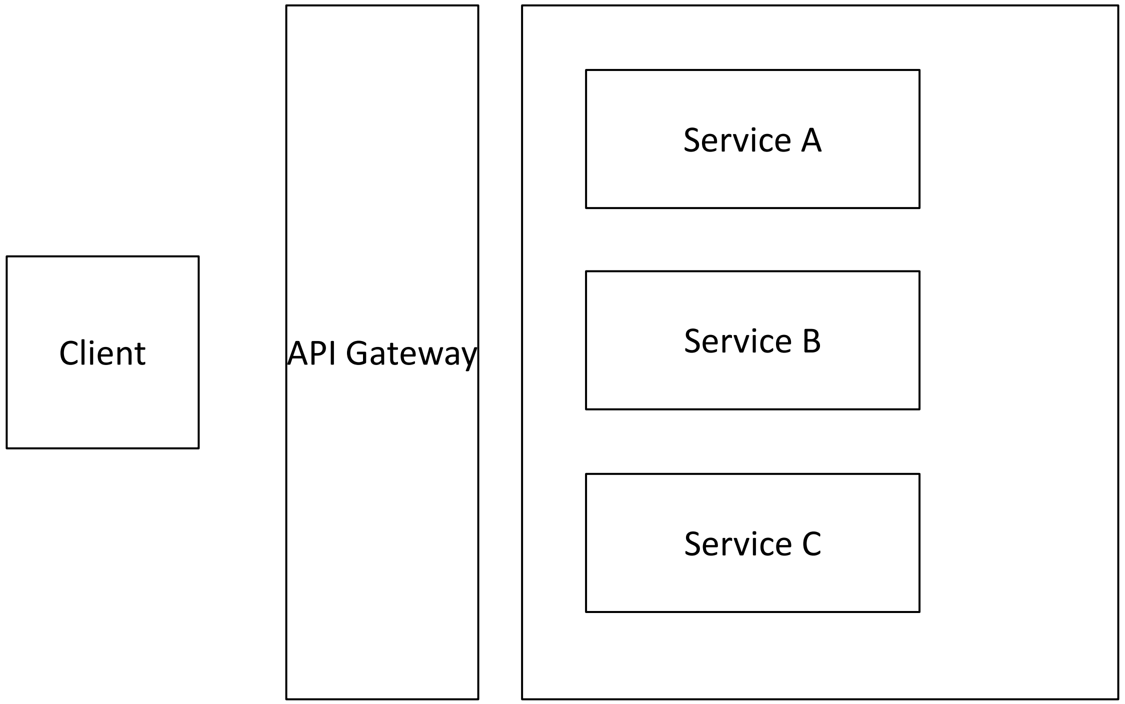
- Microservices sind unabhängig und lose gekoppelte Komponenten.
- Sind so “klein”, dass ein einzelnes Team die Verantwortung übernehmen kann.
- Jeder Service hat eine isolierte Codebasis.
- Jeder Services kann unabhängig von allen anderen Services deployed werden.



MICROSERVICES

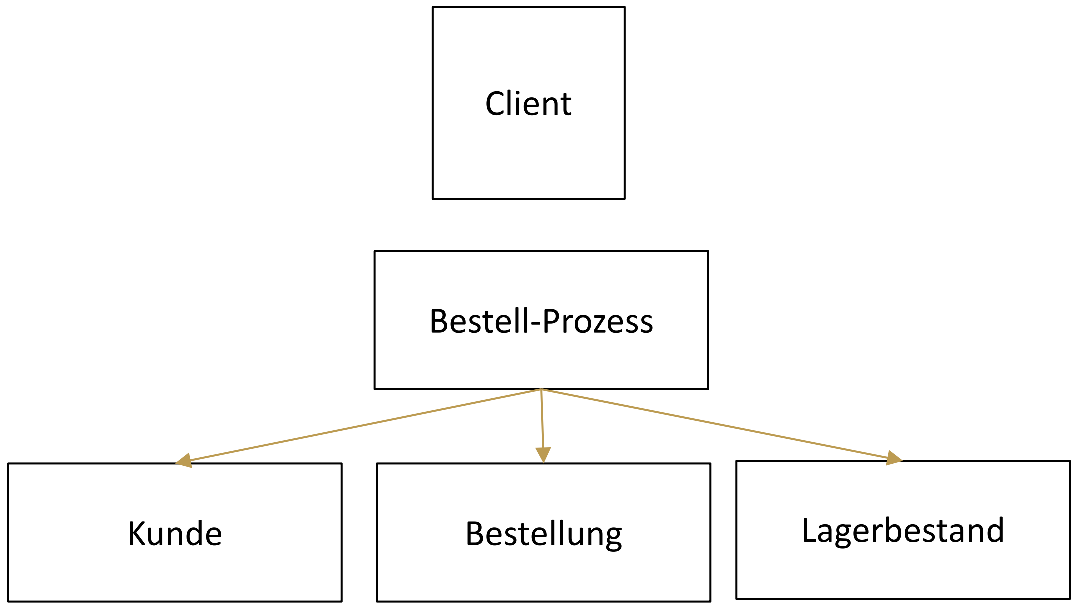
Knowledge Foundation
@ Reutlingen University

- Services sind für die Persistenz ihrer eigenen Daten / ihres Zustandes verantwortlich.
- Services kommunizieren miteinander über klar definierte API.
- Implementierungs-Details bleiben anderen Services verborgen.
- Jeder Service kann seinen eigenen Technologiestack haben.



MICROSERVICES

- Der anämische Microservice (blutleere Microservice)



← **Business Logic**

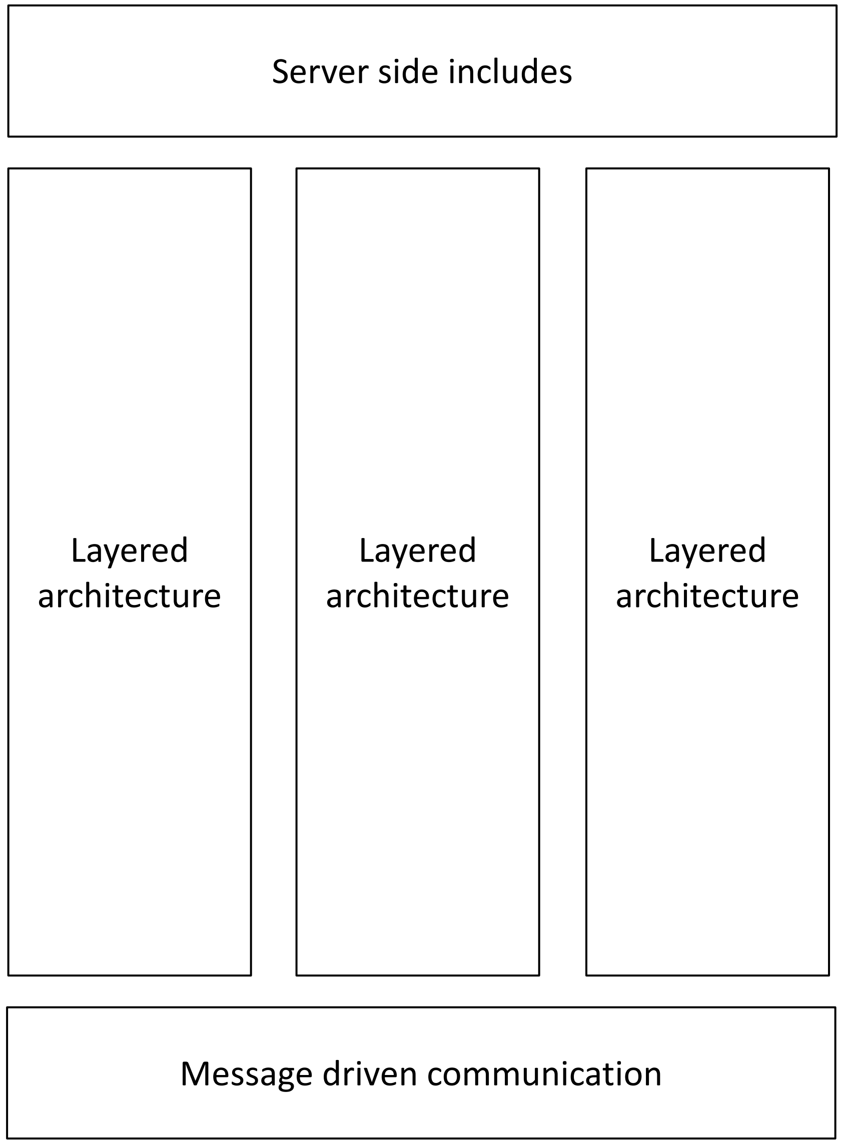
← **CRUD Operations**

SELF-CONTAINED SYSTEMS

SCS sind autonome Web-Anwendungen.

Autonomie bedeutet:

- eigenes (Micro-)Frontend (no shared UI)
- eigene Geschäftslogik und
- eigene Datenhaltung
- Logik und Datenhaltung kann als Microservice implementiert werden
- Können optional auch einen Service als API anbieten.
- Jedes SCS hat genau ein Team als Owner
- SCS kommunizieren asynchrony, wenn immer möglich.



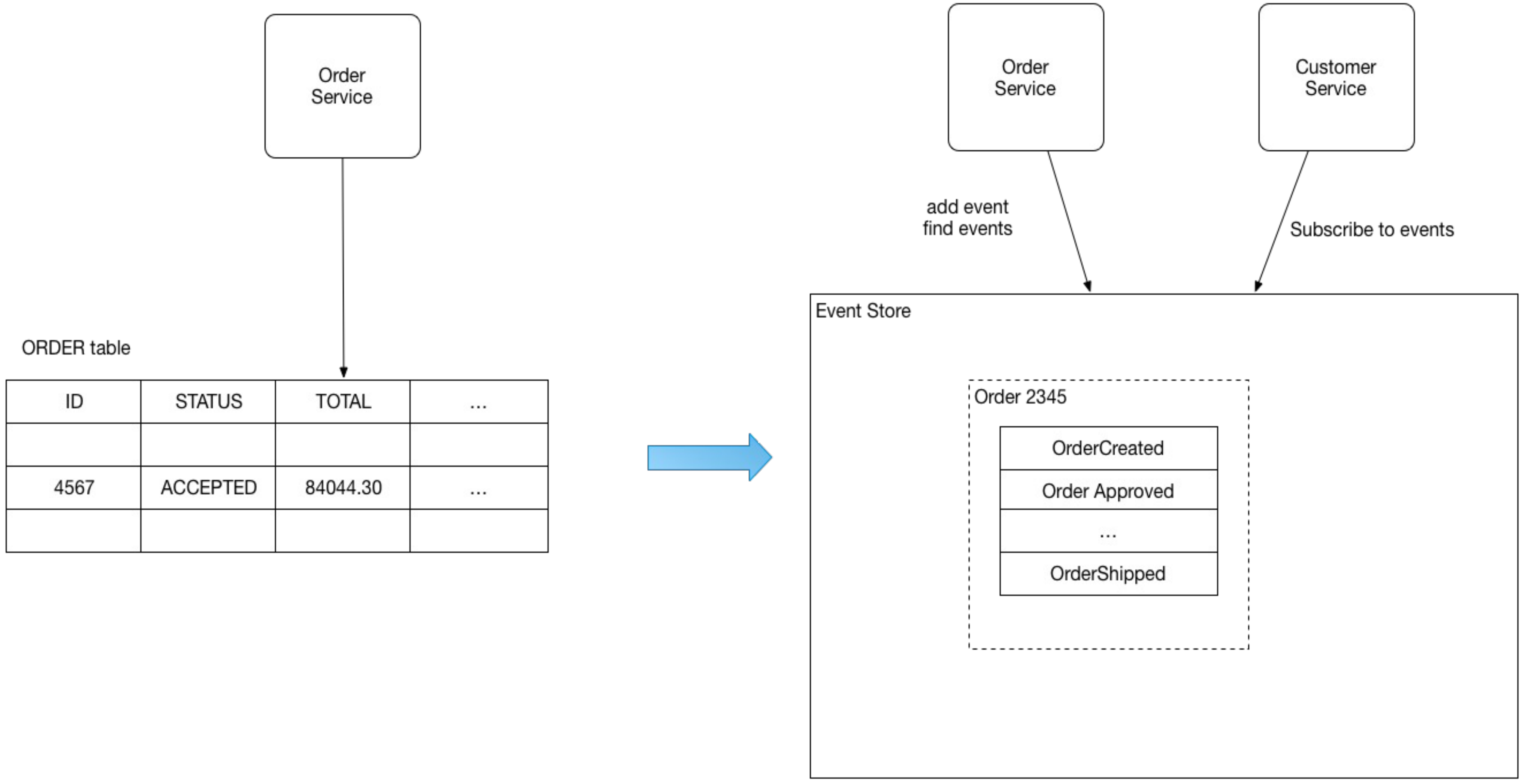
EVENT-SOURCING (1/2)

Definition:

“The fundamental idea of Event Sourcing is that of ensuring every change to the state of an application is captured in an event object, and that these event objects are themselves stored in the sequence they were applied for the same lifetime as the application”

https://martinfowler.com/eaaDev/EventSourcing.html

EVENT-SOURCING (2/2)



ID	STATUS	TOTAL	...
4567	ACCEPTED	84044.30	...

https://microservices.io/patterns/data/event-sourcing.html

CQRS (1/2)

CQRS = Command-Query-Responsibility-Segregation

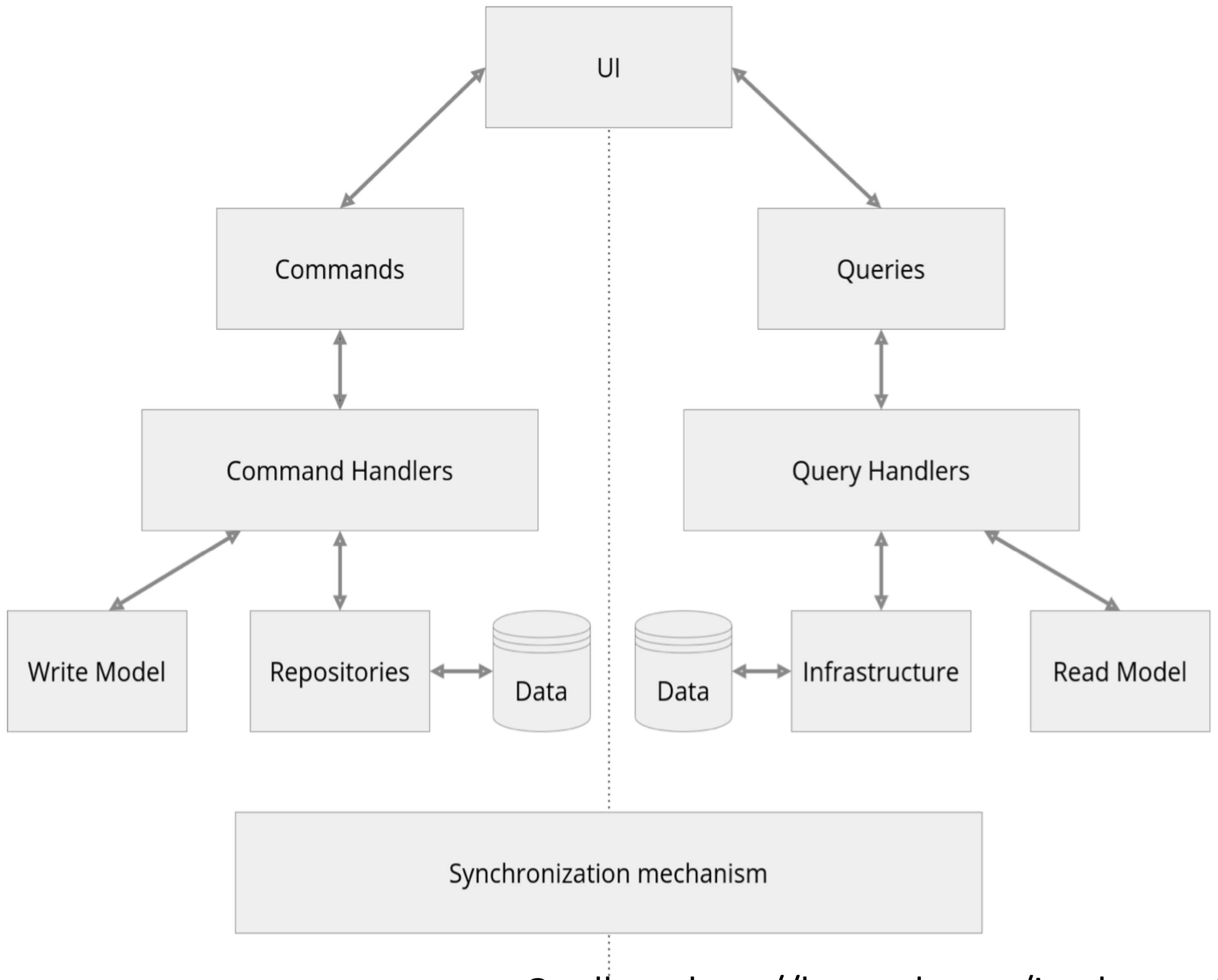
In CQRS wird das Modell in zwei Teile getrennt:

- Transaktionales Modell (Command) → Update information
- Performantes Anfrage-Modell (Query) → Read information

Da die Datenhaltung ebenfalls getrennt erfolgt wird kann nur Eventual Consistency gewährleistet

Quelle http://leanpub.com/implementing-ddd-cqrs-and-event-sourcing

CQRS (2/2)



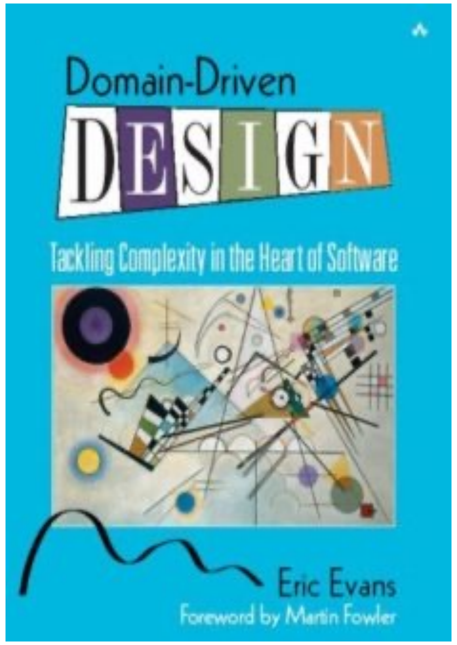
Quelle http://leanpub.com/implementing-ddd-cqrs-and-event-sourcing

Knowledge Foundation
@ Reutlingen University

FURTHER READING

DDD LITERATURE

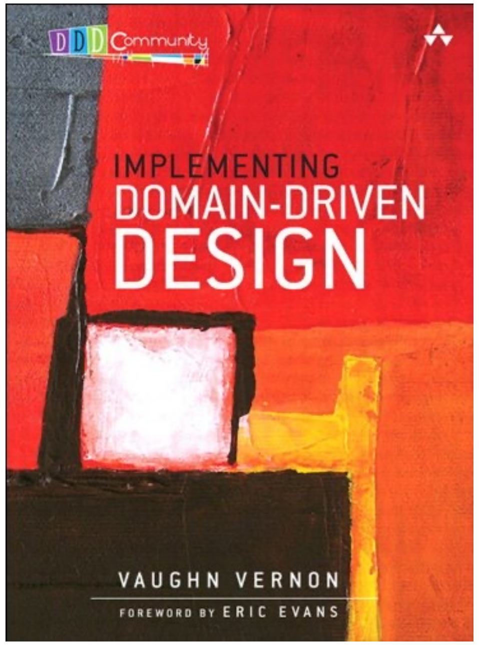
The Big Blue Book


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

2003 - Eric Evans
“Domain Driven Design”

“not easy to read”

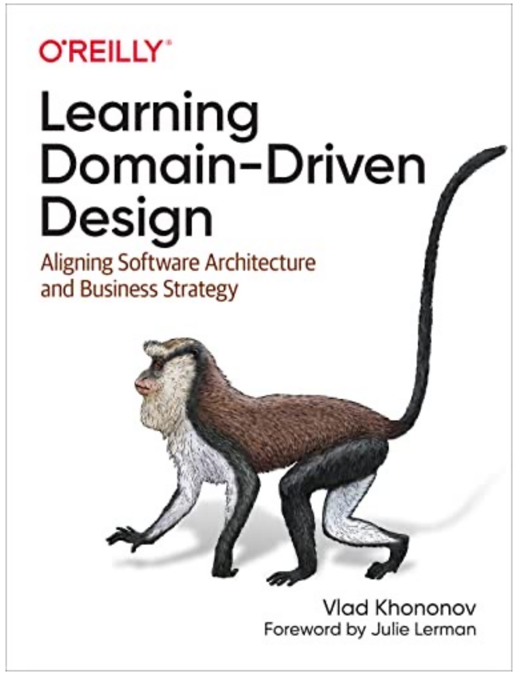
The Big Red Book



2013 - Vernon Vaughn
“Implementing Domain Driven Design”
“Fokus auf State of the Art (2013) architecture styles”

Vaughn Vernon
Domain-Driven
Design
kompakt
→ Aus dem Englischen übersetzt
von Carola Lilienthal und Henning Schwentner
dpunkt.verlag

2017 - Vernon Vaughn
“Domain Driven Design - kompakt”
“deutsche Übersetzung von C. Lilienthal – kompakte Einführung”

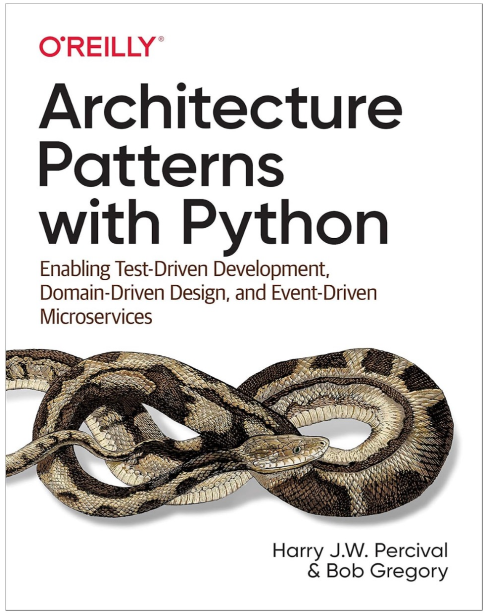


2022 – Vlad Khononov
“Learning Domain Driven Design”

“adopted DDD to modern design principles”

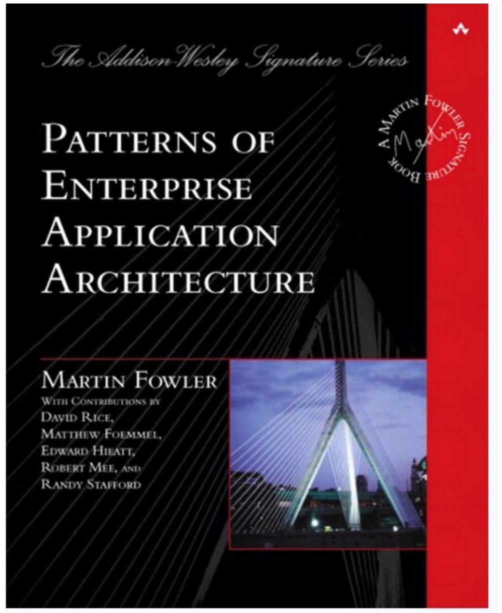
DDD LITERATURE

Focus on Python


O'REILLY
Architecture Patterns with Python
Enabling Test-Driven Development, Domain-Driven Design, and Event-Driven Microservices
Harry J.W. Percival & Bob Gregory

2020 – Percival & Gregory
“Architecture Patterns with Python”

“covers a lot of modern architecture pattern on top of DDD”


The Addison-Wesley Signature Series
Patterns of Enterprise Application Architecture
Martin Fowler
With Contributions by David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee, and Randy Stafford
A Martin Fowler Signature Book

2002 – Martin Fowler
“Patterns of Enterprise Application Architecture”