

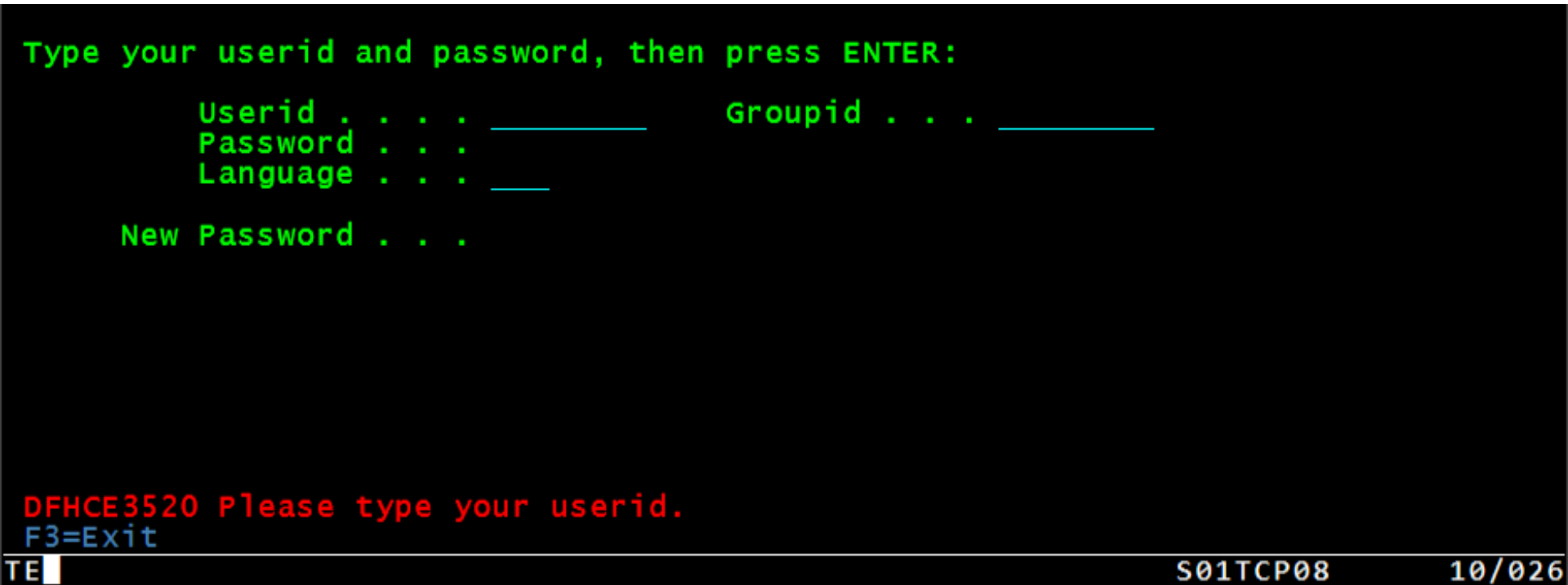
COCO Application Design 2025

Einführung Schichten-Architektur

Technology
Arts Sciences
TH Köln

Das „Schweizer Taschenmesser“: **Hintergrund und Charakteristik von Schichtenarchitekturen**

1950er / 1960er: Mainframes mit Terminals (1)



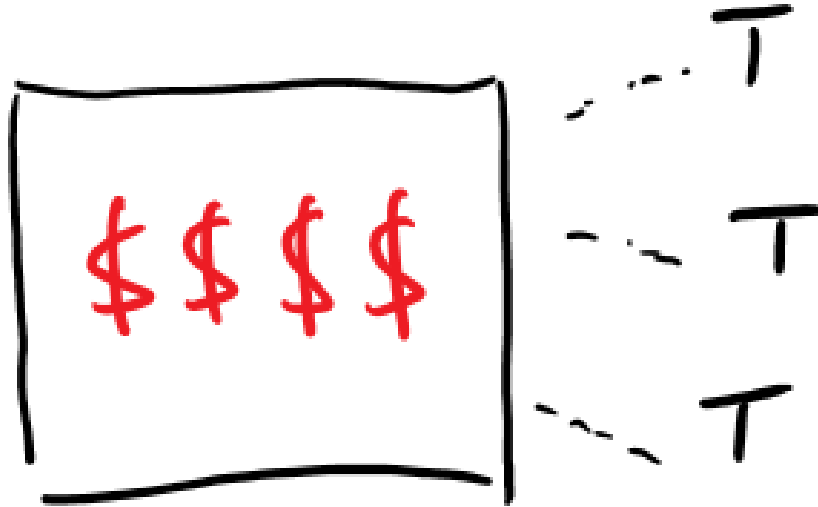
Das „Schweizer Taschenmesser“: Hintergrund und Charakteristik von Schichtenarchitekturen

Viele arbeitsteilige Zugriffe auf die gleiche Software ...



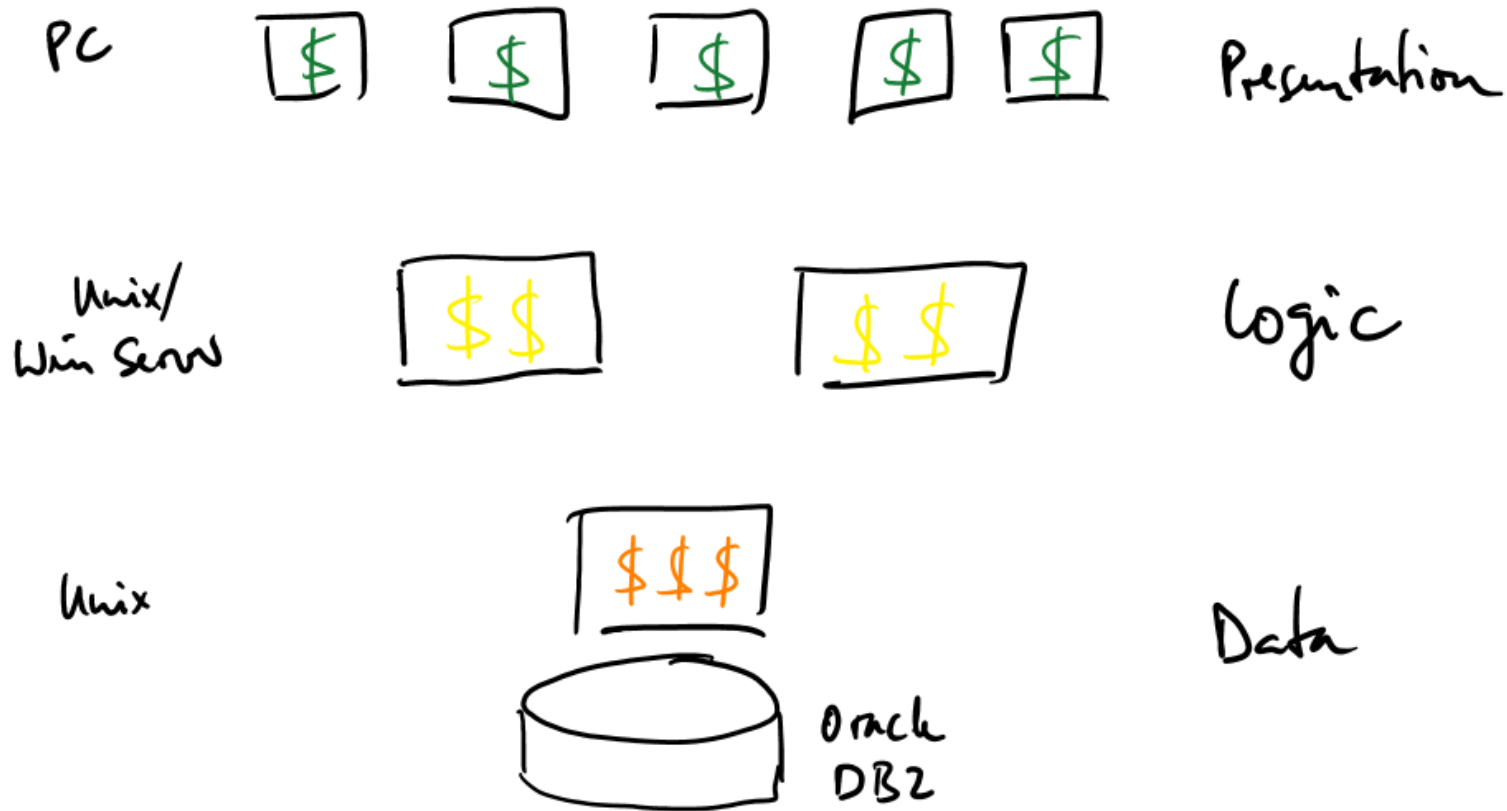
Bild: Petiatil (talk) on Wikipedia-en. - Eigenes Werk, Gemeinfrei, <https://commons.wikimedia.org/wiki/index.php?curid=821965>

1950er / 1960er: Mainframes mit Terminals (2)



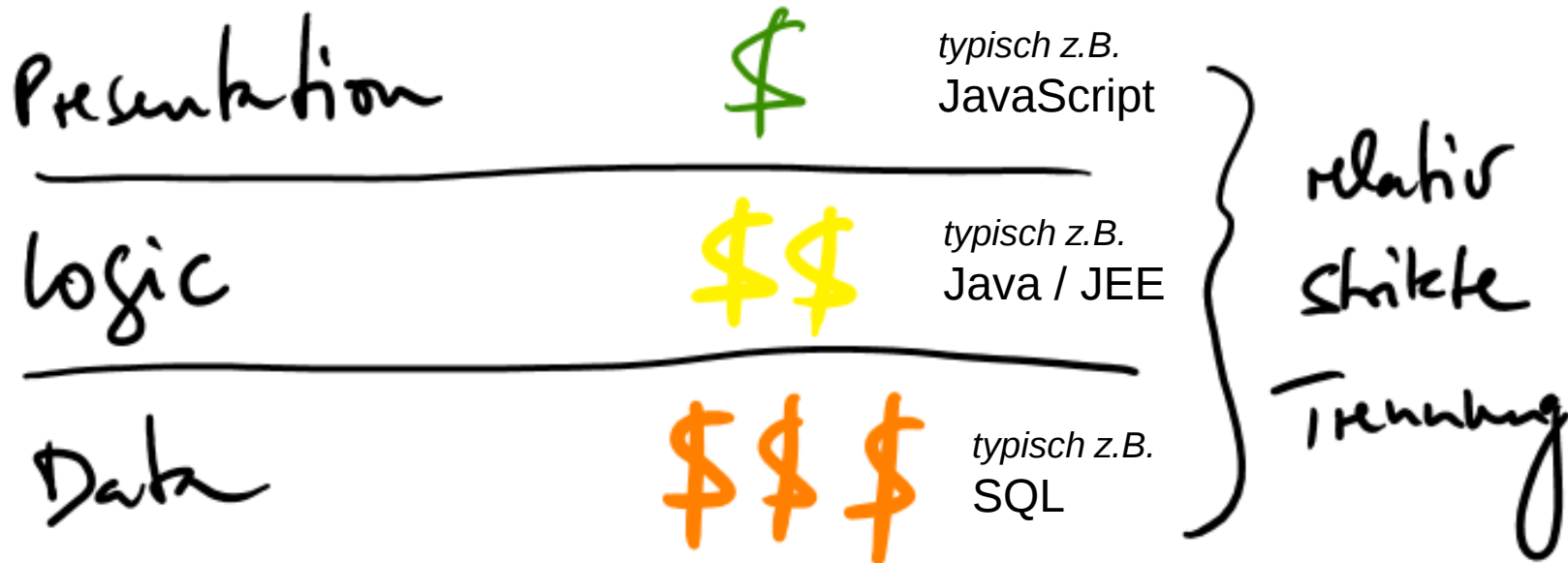
- Sehr teure, zentrale Hardware (Mainframe)
- HW und SW aus derselben Hand
- Verteilte Terminals
 - Terminal = nur ein Netzwerkclient, der **Masken** anzeigt
 - Maske = digitales Formular, keine Mausbedienung, Navigation mit Tabs
 - Masken werden mit IDs (3-4stellige Zahlen) direkt angesprungen

Ende 1970er ... frühe 2000er: UNIX-Servers, PCs als Clients



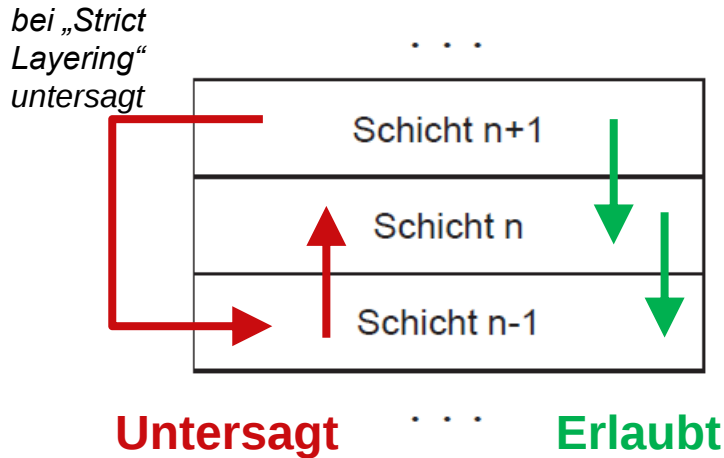
- Typische Konfiguration hat im Wesentlichen **drei Arten** / Schichten von Rechnern.
- 1. **Data:** Zentrale relationale Datenbank (Oracle, DB2, ...), i.d.R. eine zentrale Instanz für das ganze Unternehmen, sehr teure HW und SW-Lizenzen. I.d.R. vertikale Skalierung (mächtige CPU und viel Memory) für den DB-Server.
- 2. **Logic:** Mittelmäßig teure UNIX- / Linux-Server für die Geschäftslogik. Verteilt, mit horizontaler Skalierung (mehr Server zu einem Netzwerk zusammenschließen)
- 3. **Presentation:** relativ preiswerte PCs für die UI Clients (erst Rich Clients, später hauptsächlich Web Clients)

Klassische 3-Schichtenarchitektur: **Data / Logic / Presentation**



- Aus dieser im Grunde technischen Konfiguration hat sich die klassische 3-Schichten-Architektur entwickelt, mit relativ strikter Trennung:
 1. **Data**
 2. **Logic**
 3. **Presentation**

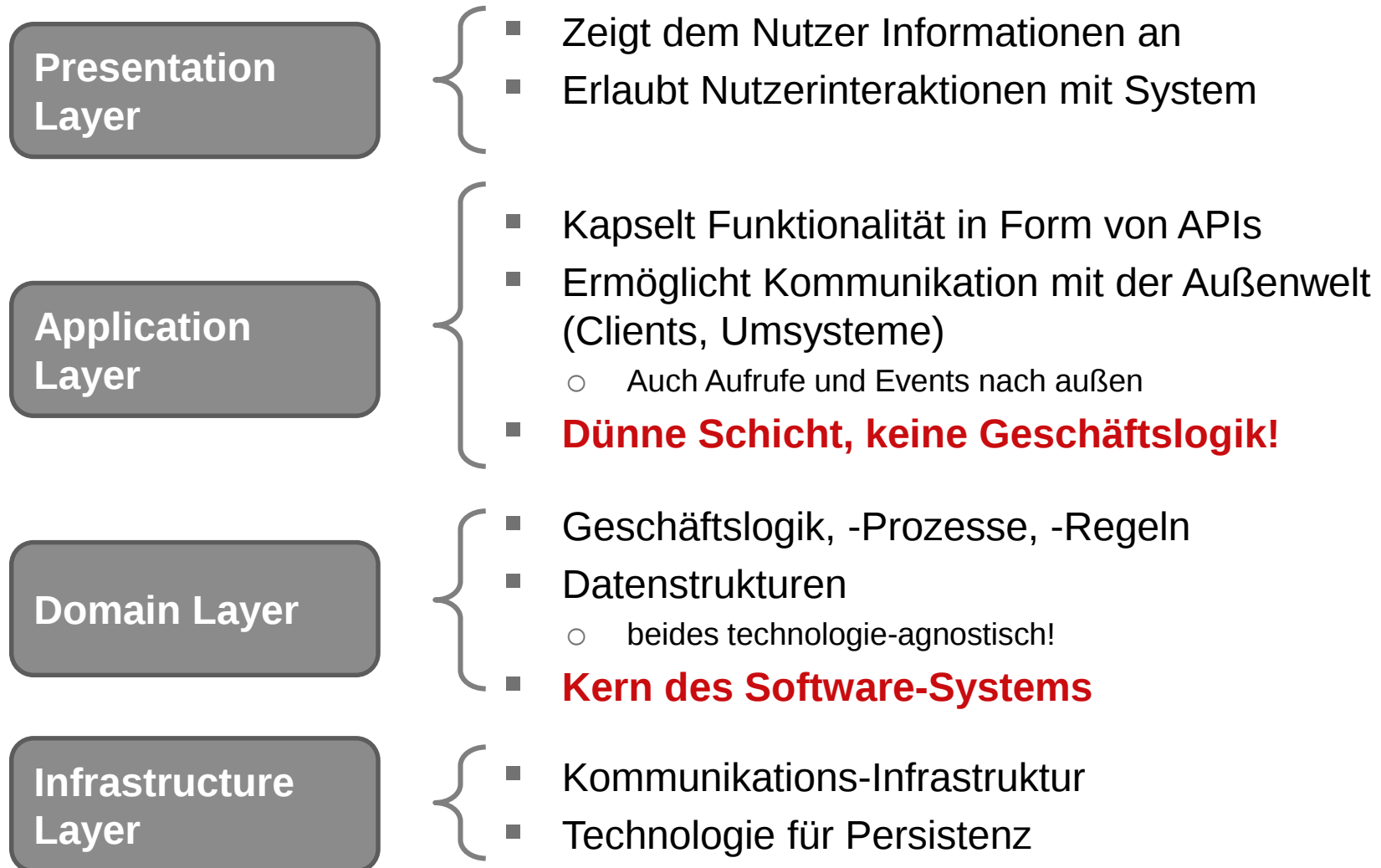
Regeln für Schichtenarchitektur



- Schicht **n** hängt nur von der tiefer liegenden Schicht **n-1** ab
 - Keine Schicht hängt von höheren Schichten ab
 - Schicht **n-1** bietet Dienste für Schicht **n** an
 - Zugriff von **n** auf **n-1** über Schnittstellen
 - Schnittstellen können formal oder informell sein
-
- Zugriff "nach oben" **n** auf **n+1, n+2, ...** untersagt
 - *Strict Layering*: Zugriff **n** auf **n-2, n-3, ...** untersagt

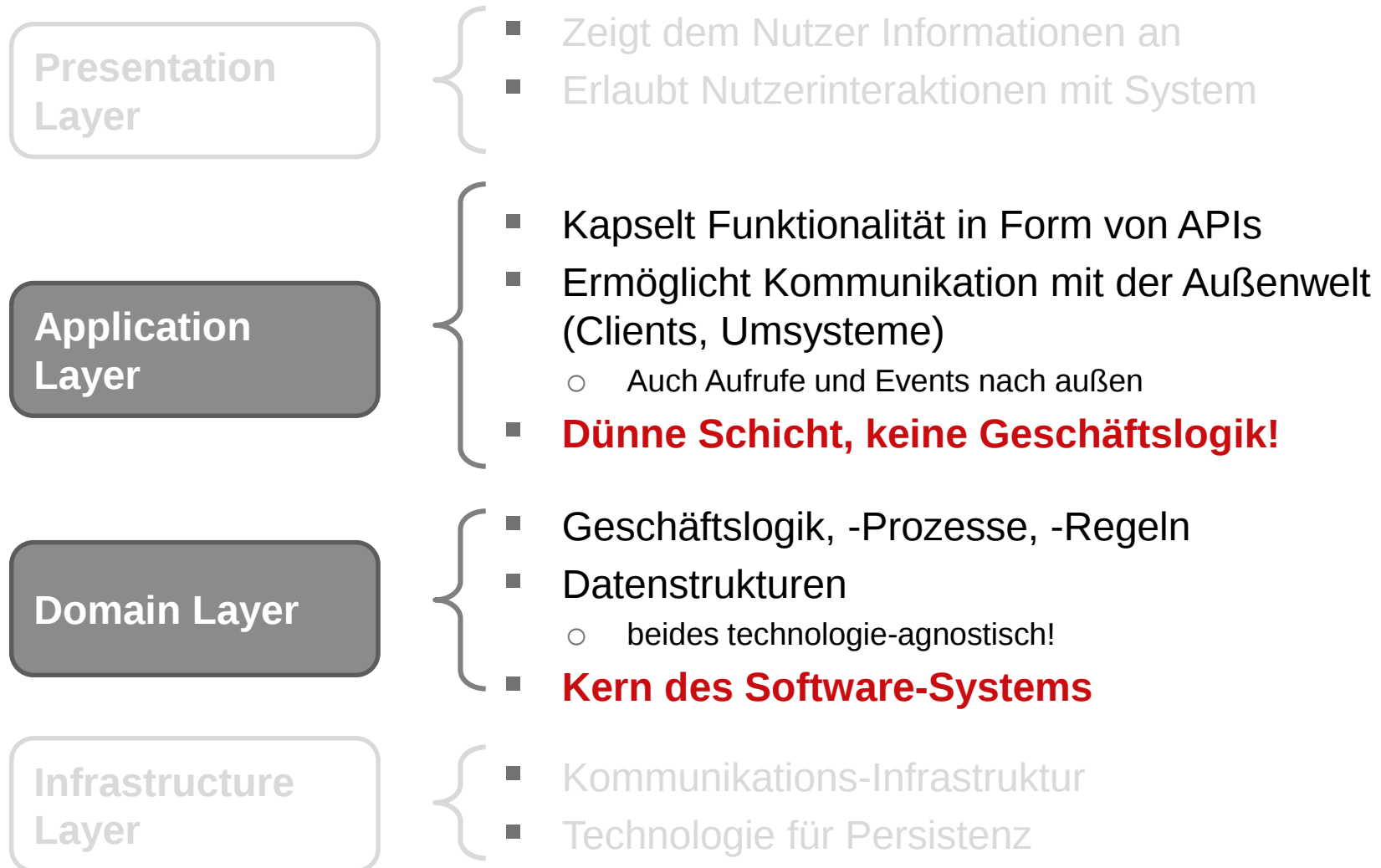
Das Schichtenmodell nach Domain-Driven Design

Schichtenaufteilung nach Evans (Domain-Driven-Design)



Quelle: [Evans], S. 53, zusammengefasst durch den Vortragenden

Schichtenaufteilung nach Evans (Domain-Driven-Design)



Quelle: [Evans], S. 53, zusammengefasst durch den Vortragenden

Services und Domain-Klassen

Application Layer

(Application)Service

- zustandslos
- regelt Domain Lifecycle (CRUD)
- ermöglicht Interaktionen & Workflows

nutzt für CRUD

nutzt für Workflows

Domain Layer

Repository

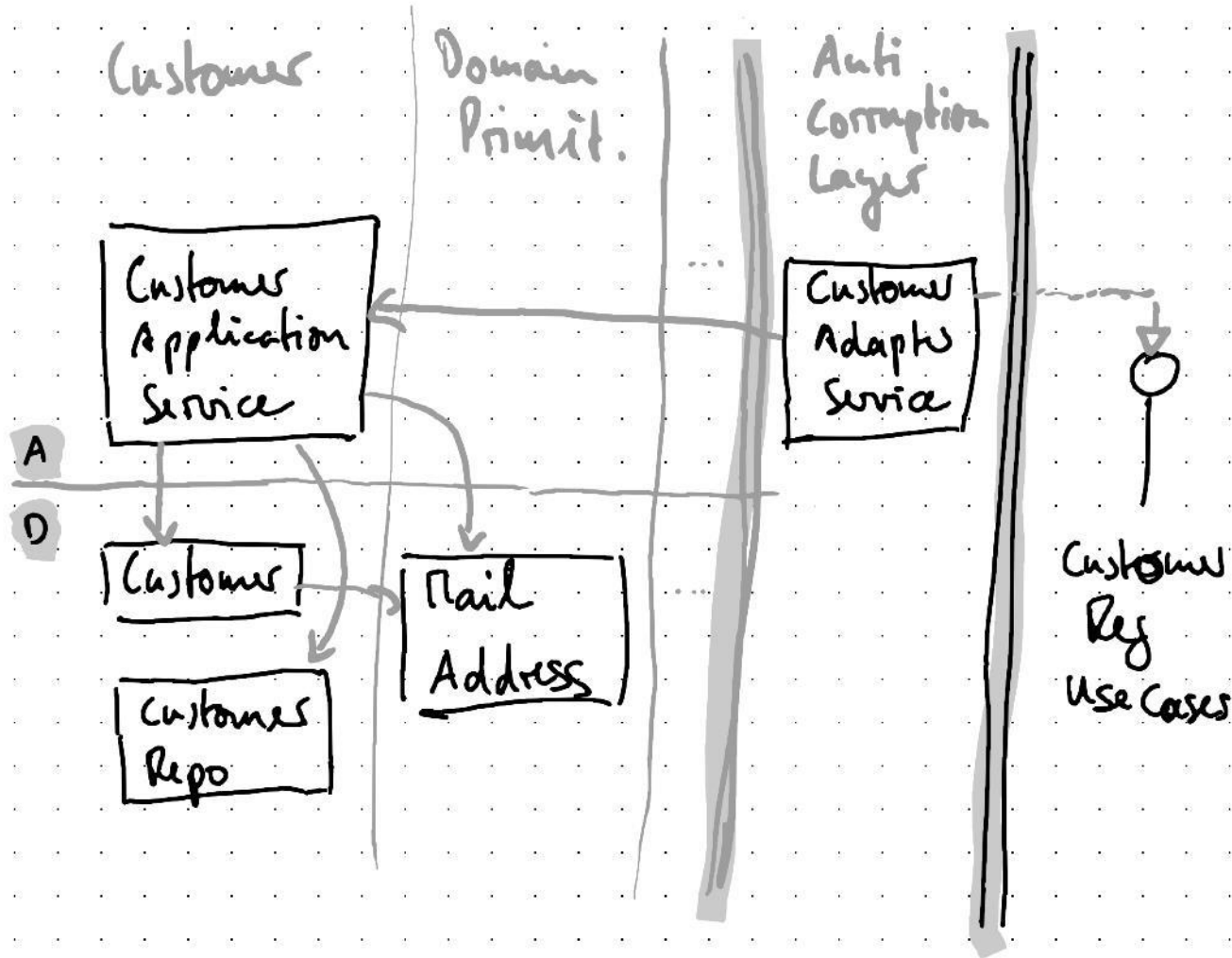
- stellt Standard-Operationen für CRUD zur Verfügung
- definiert Queries

Entity

- haben Zustand
- werden auf Datenbank abgebildet

Value Object

... und das für das Beispiel „Kunde“ ...



*Alternative Formen der DDD-
Schichtung:*

Hexagonal Architecture

Onion Architecture

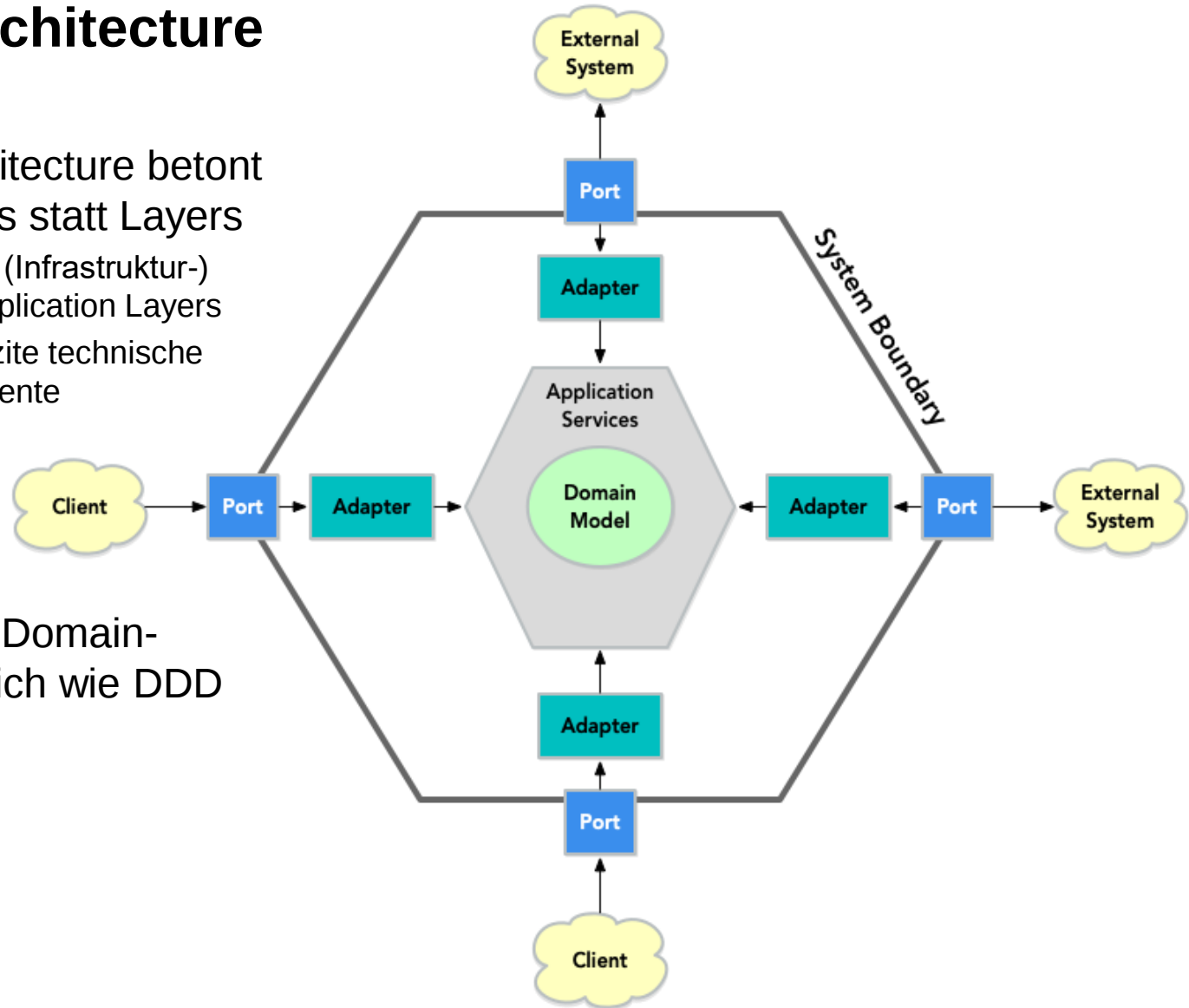
Clean Architecture

Hexagonal Architecture, Onion Architecture

- Hexagonal Architecture ist ein Konzept von Alistair Cockburn ⁽¹⁾
- Onion Architecture stammt von Jeffrey Palermo ⁽²⁾
- Beide Konzepte sind in gewisser Weise „mehrdimensionale“ Varianten des DDD-Schichten-Konzepts
 - Sie stammen auch aus ähnlicher Zeit (DDD: 2003, HA: 2005, OA: 2008)
- Gemeinsamkeiten:
 - ringförmige Darstellung, statt gestapelter Schichten
- Haupt-Unterschiede:
 - Onion Architecture kennt Layers, Hexagonal Architecture nur Ports
- Im Weiteren stelle ich die Konzepte anhand eines Blogs von Herberto Graca vor, der eine sehr gute Infografik produziert hat (siehe nächste Folie)
- Quellen
 - (1) Cockburn, A. (2005). Hexagonal Architecture.
<https://web.archive.org/web/20180822100852/http://alistair.cockburn.us/Hexagonal+architecture>, abgerufen 12.4.20
 - (2) Palermo, J. (2008). The Onion Architecture.
<https://web.archive.org/web/20180822100852/http://alistair.cockburn.us/Hexagonal+architecture>, abgerufen 12.4.20

Hexagonal Architecture

- Hexagonal Architecture betont Ports & Adapters statt Layers
 - =~ „technische“ (Infrastruktur-) Aspekte des Application Layers
 - Adapter = explizite technische Controller-Elemente
- Application und Domain-Kern sonst ähnlich wie DDD



DDD / Hexagonal Architecture / Onion Architecture

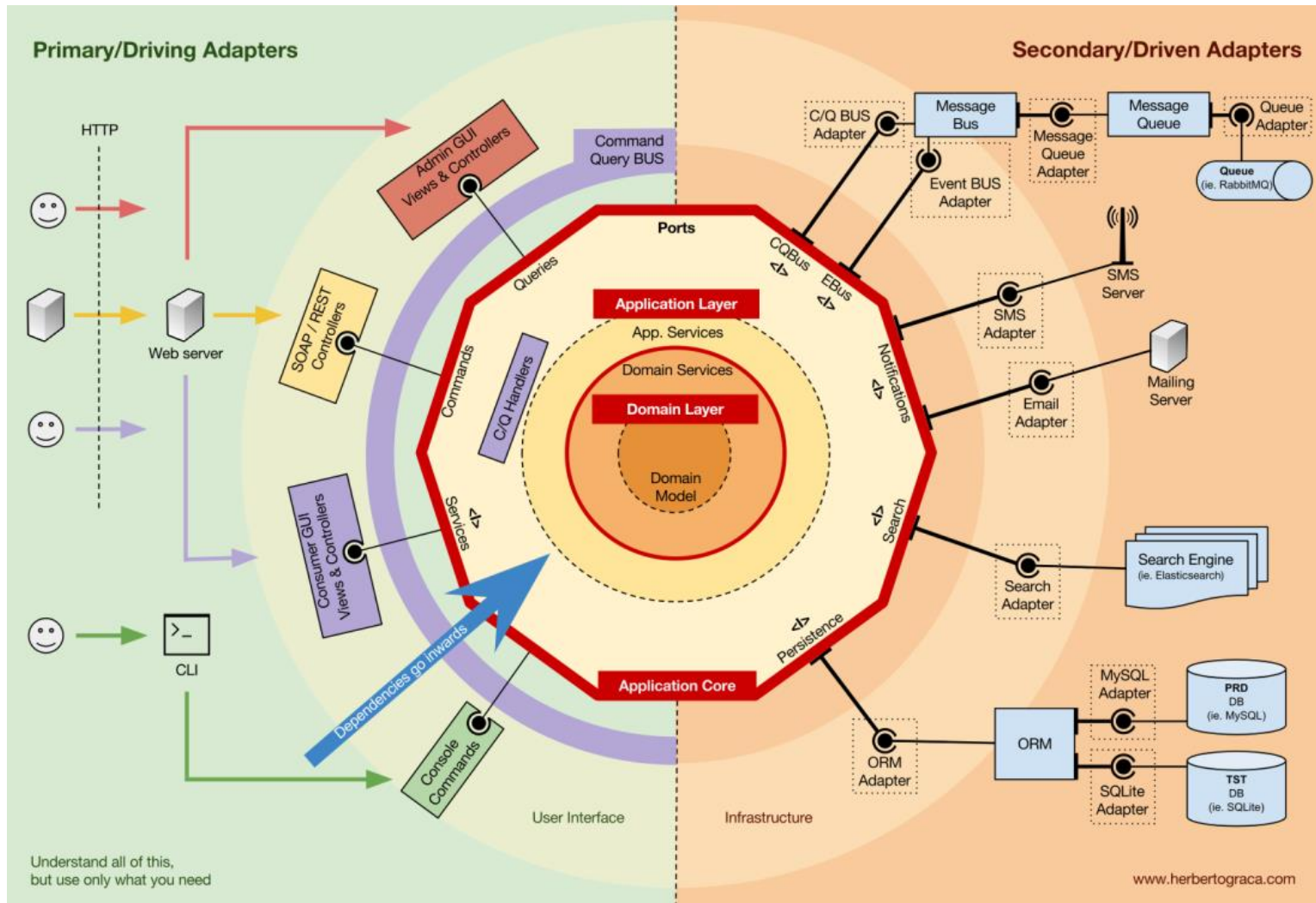


Bild: Graca, H. (2017, November 16). DDD, Hexagonal, Onion, Clean, CQRS, ... How I put it all together. @hgraca.
<https://herbertograca.com/2017/11/16/explicit-architecture-01-ddd-hexagonal-onion-clean-cqrs-how-i-put-it-all-together/>

Clean Architecture (Bob Martin)

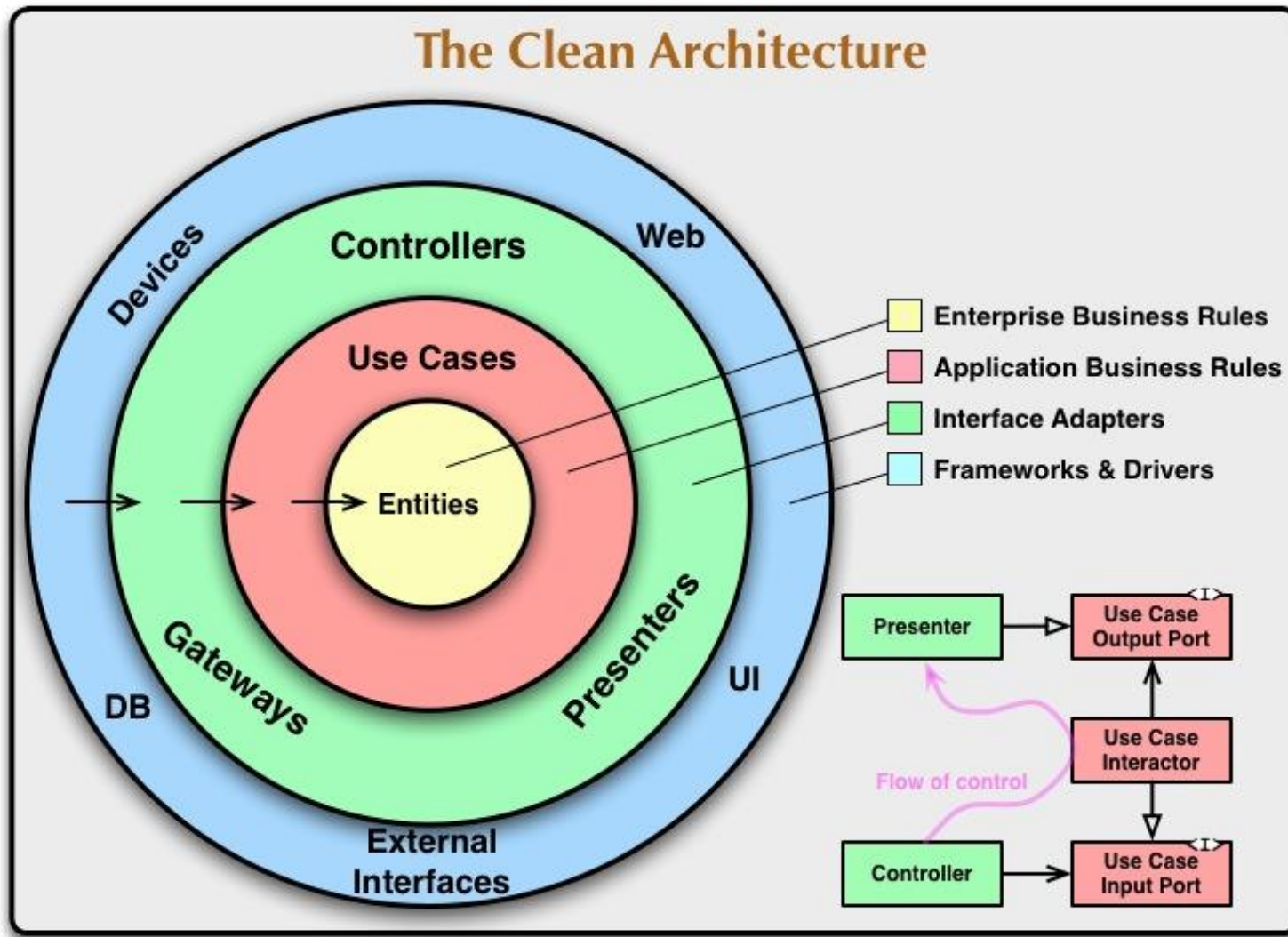


Bild: Martin, R. C. (2012, August 13). The Clean Architecture. *Clean Coder Blog*. <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>

Clean Architecture (Bob Martin)

- Clean Architecture hat eine ähnliche ringförmige Struktur wie Hexagonal Architecture und Onion Architecture
- Vergleich zu DDD:
 - Der **äußere blaue** Ring repräsentiert (wie bei HA und OA) eine Mischung aus Presentation und Infrastructure Layer
 - Der **grüne** und der **rote** Ring entsprechen dem Application Layer
 - Use Cases entsprechen Application Layer Services (siehe später in dieser Vorlesung)
 - Der **gelbe** Kern entspricht dem Domain Layer

