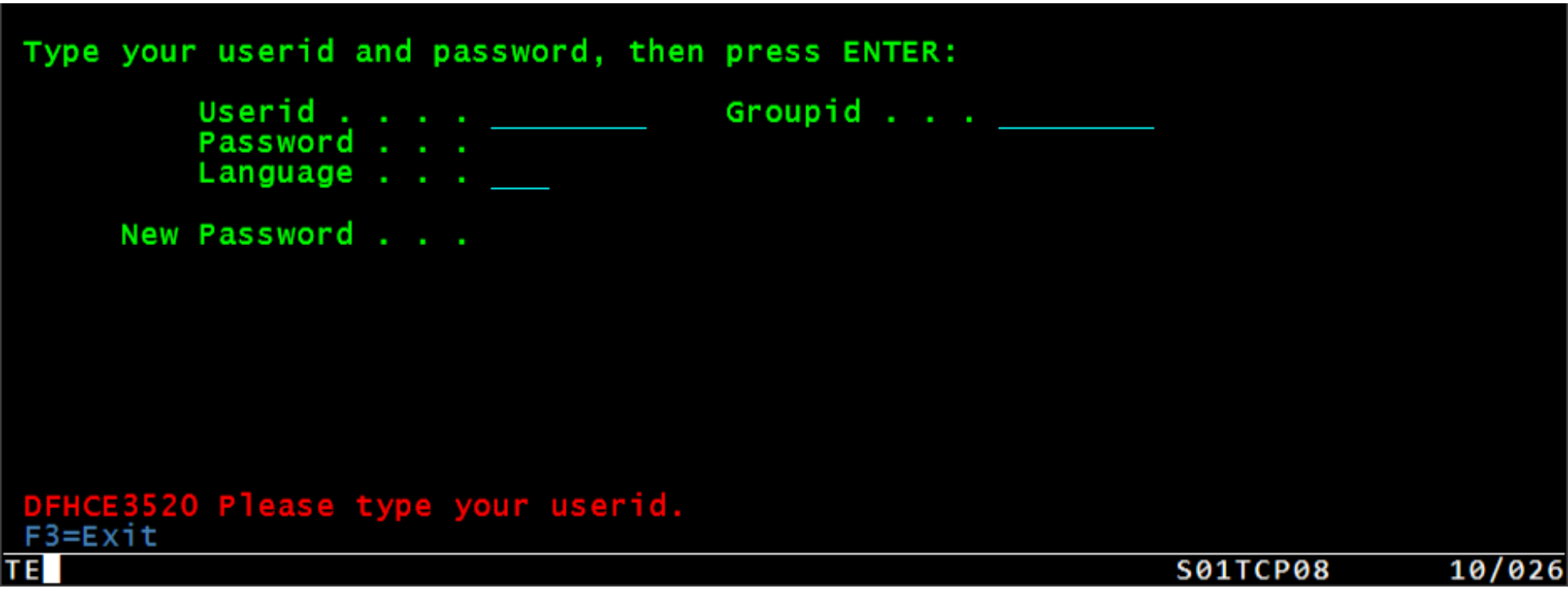
Das „Schweizer Taschenmesser“: **Schichtenarchitektur und Package-Struktur in der Praxis**

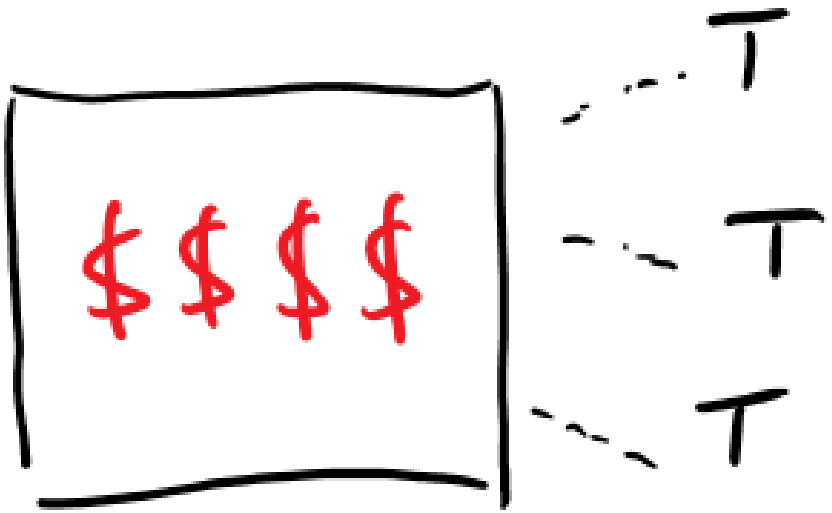
Viele arbeitsteilige Zugriffe auf die gleiche Software …

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

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

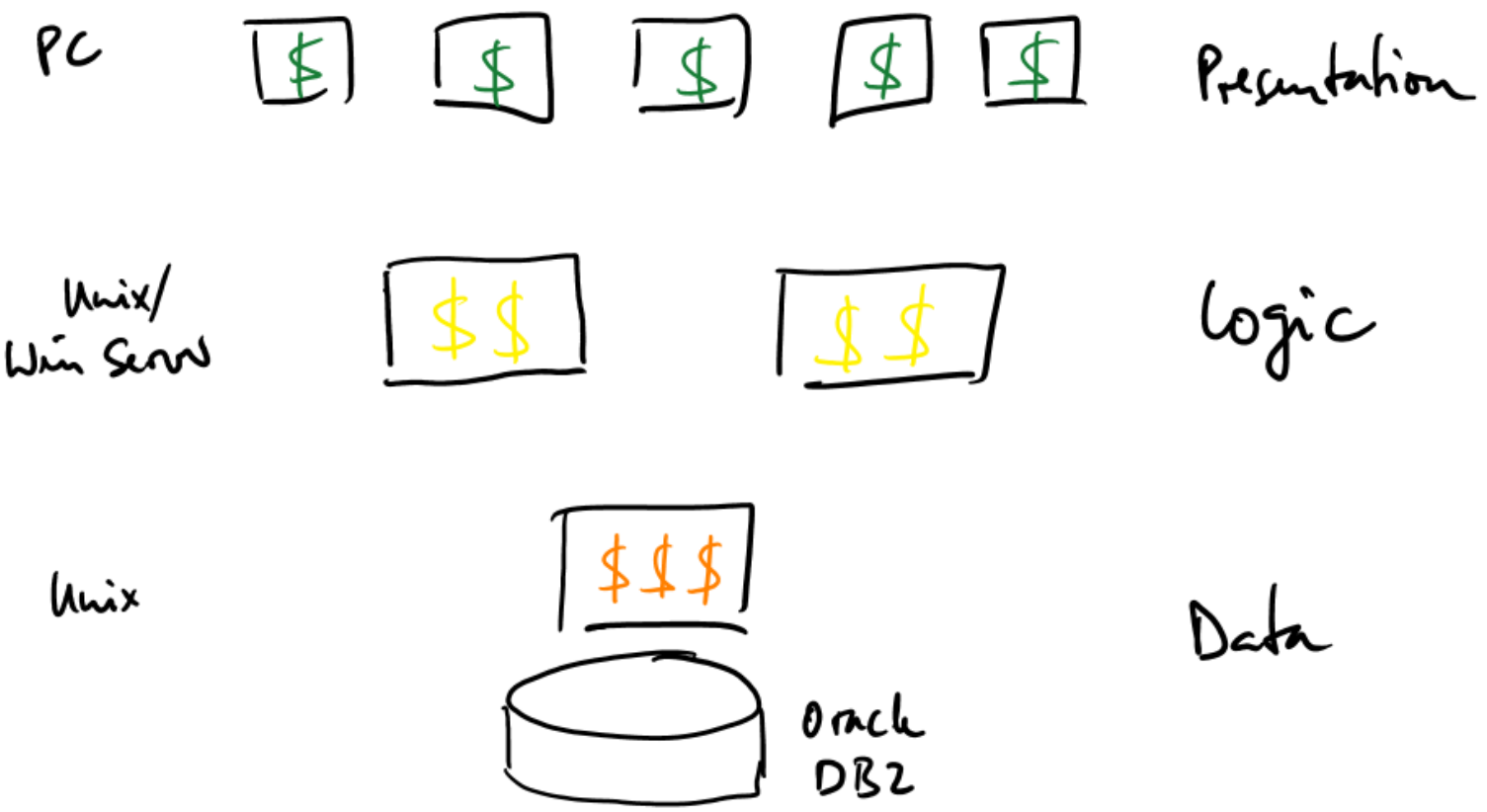


1950er / 1960er: Mainframes mit Terminals (2)



- 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

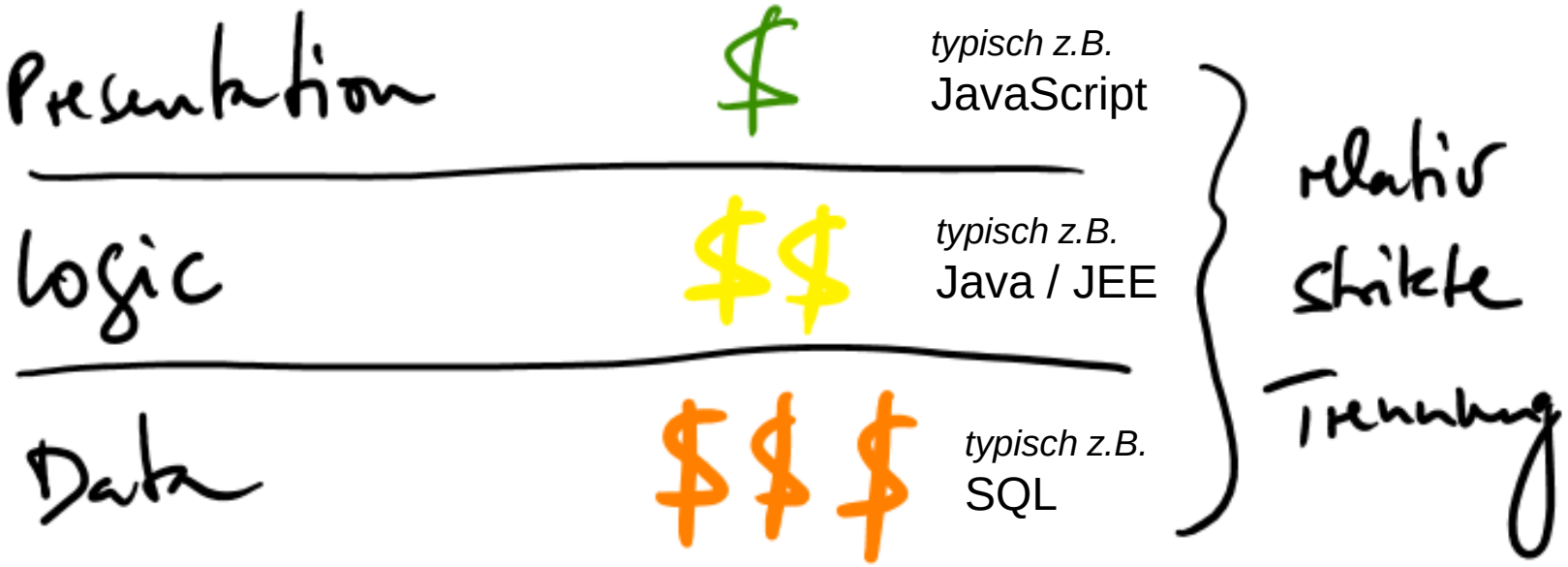
Ende 1970er … frühe 2000er: UNIX-Servers, PCs als Clients



- 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 ganz 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**

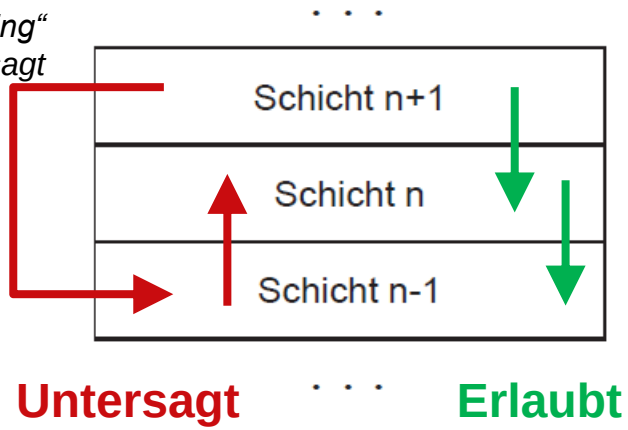


- Aus dieser im Grunde technischen Konfiguration hat sich die klassische 3-Schichten-Architektur entwickelt, mit relativ strikter Trennung:

1. **Data**
2. **Logic**
3. **Presentation**

Regeln für Schichtenarchitektur

bei „Strict Layering“ untersagt



- 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

Quelle: nach [Goll], S. 290ff.

Das Schichtenmodell nach Domain-Driven Design

Schichtenaufteilung nach Evans (Domain-Driven-Design)

Presentation Layer

- Zeigt dem Nutzer Informationen an
- Erlaubt Nutzerinteraktionen mit System

Application Layer

- Kapselt Funktionalität in Form von APIs
- Ermöglicht Kommunikation mit der Außenwelt (Clients, Umsysteme)
  - Auch Aufrufe und Events nach außen
- **Dünne Schicht, keine Geschäftslogik!**

Domain Layer

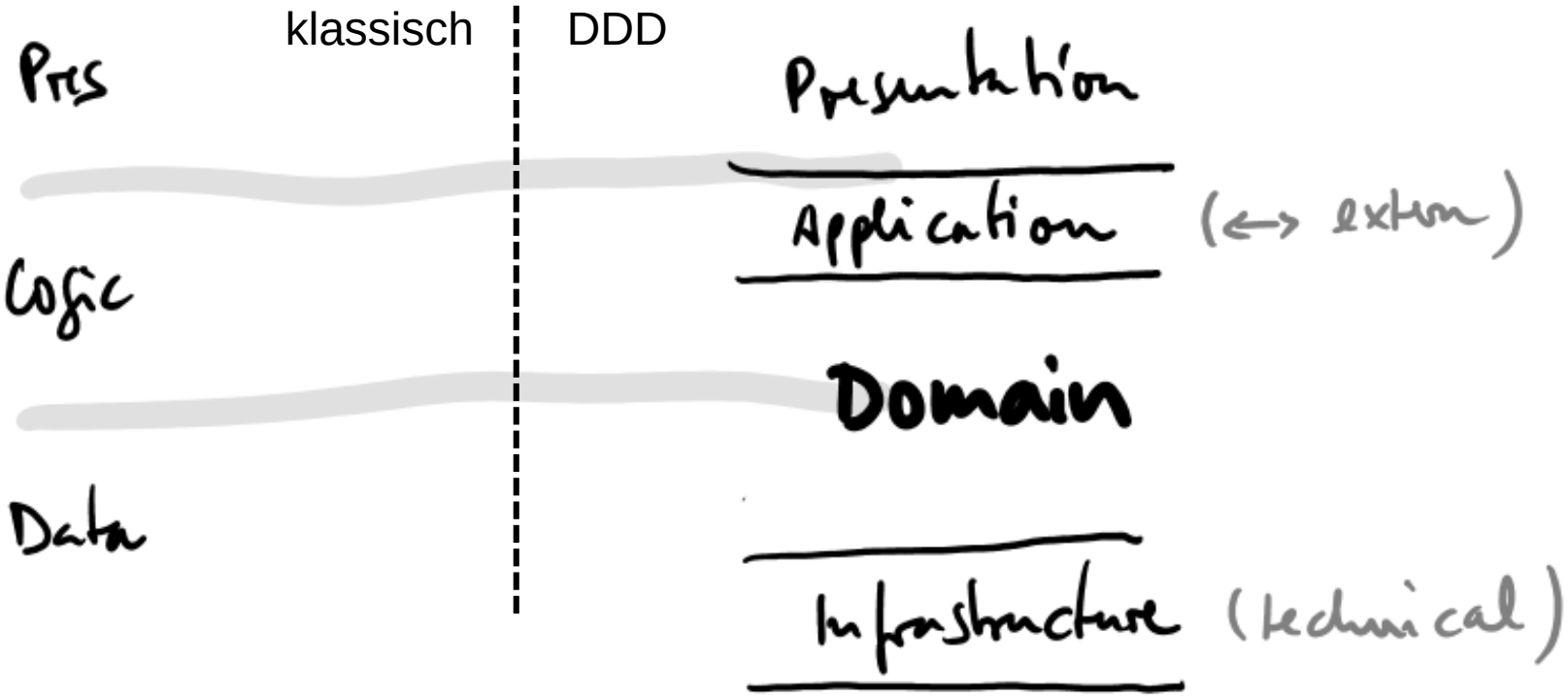
- Geschäftslogik, -Prozesse, -Regeln
- Datenstrukturen
  - beides technologie-agnostisch!
- **Kern des Software-Systems**

Infrastructure Layer

- Kommunikations-Infrastruktur
- Technologie für Persistenz

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

Unterschied DDD vs. „klassische“ Schichtenarchitektur



- Der **Presentation Layer** ist ähnlich in beiden Modellen.
- Ein Teil der klassischen Logikschicht wird als **Application Layer** abgetrennt – genau der Teil, der wesentliche Funktionen nach außen (als Application) zur Verfügung stellt. Enthält APIs und Kommunikation mit der Außenwelt oder anderen Teilen des Systems. Enthält ausdrücklich **keine** Geschäftslogik.
- Die Teile der Logic- und Data-Schichten, die Geschäftslogik enthalten, werden als **Domain Layer** zusammengeführt. Dies ist der Kern des Software-Systems.
- Der **Infrastructure Layer** enthält die technischen Infrastrukturen, z.B. für Kommunikation oder Persistenz.

Schichtenaufteilung nach Evans (DDD) - Originalzitate

Presentation Layer

- Responsible for showing information to the user and interpreting the user's commands. (…)

Application Layer

- Defines the jobs the software is supposed to do and directs the expressive domain objects to work out problems.
- The tasks this layer is responsible for are meaningful to the business or necessary for interaction with the application layers of other systems.
- 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.

Domain Layer

- 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.*** (**)

Infrastructure Layer

- Provide generic technical capabilities that support the higher layers: message sending for the application, persistence for the domain, drawing widgets for the UI, etc.
- The infrastructure layer may also support the pattern of interactions between the four layers through an architectural framework.

Quelle: [Evans], S. 53
(*) Hervorhebung durch Stefan Bente (**) Hervorhebung im Originaltext durch Eric Evans

Was gehört wohin?

Presentation Layer

Application Layer

- Application Service
  - Lifecycle Management
  - Workflows für externe Systeme
- REST Controller

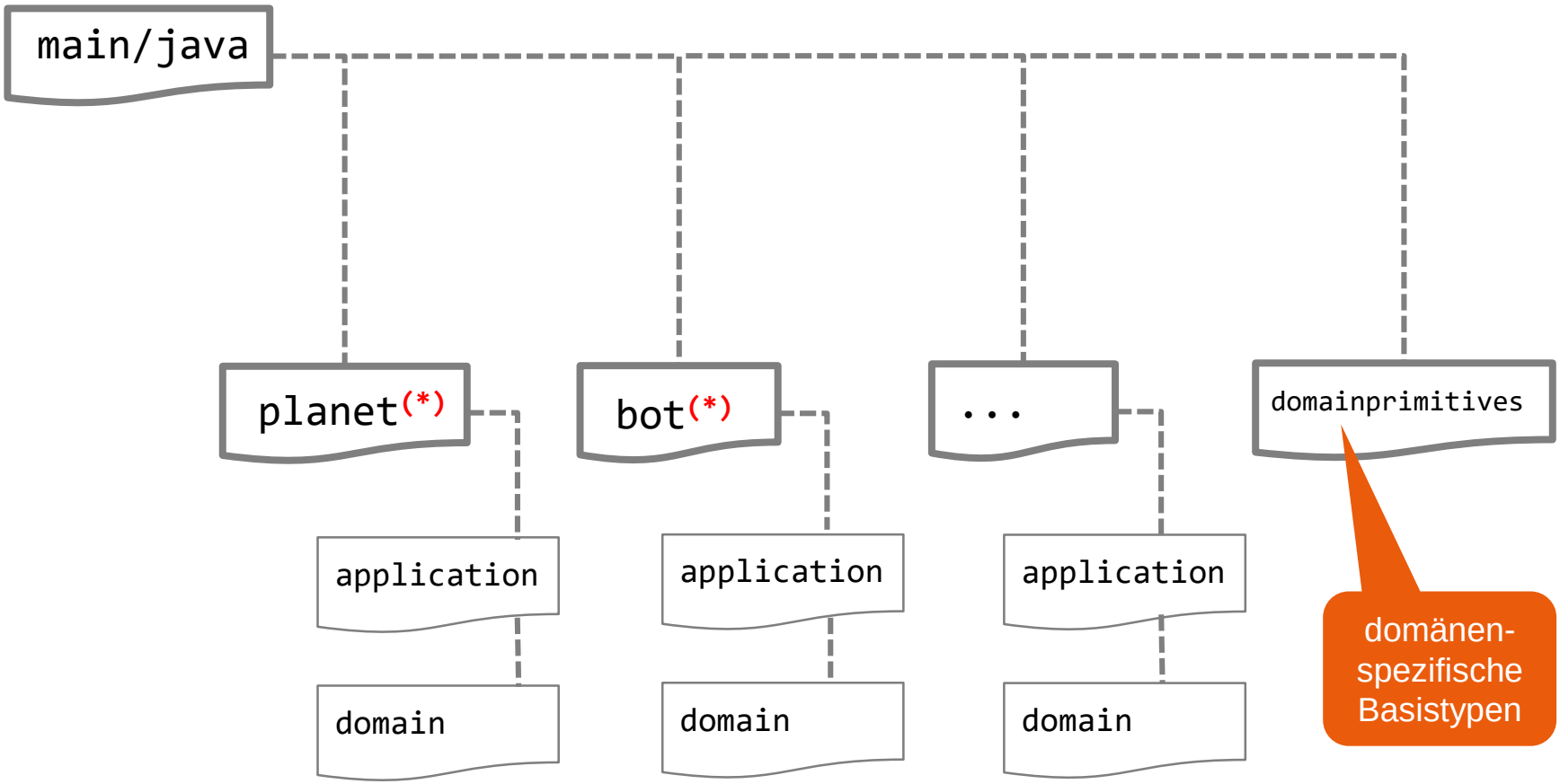
Domain Layer

- Entity
- Repository
- Value Object
- Domain Primitive
- Domain Service
  - Transaktionen zwischen *mehreren* Entities

Infrastructure Layer

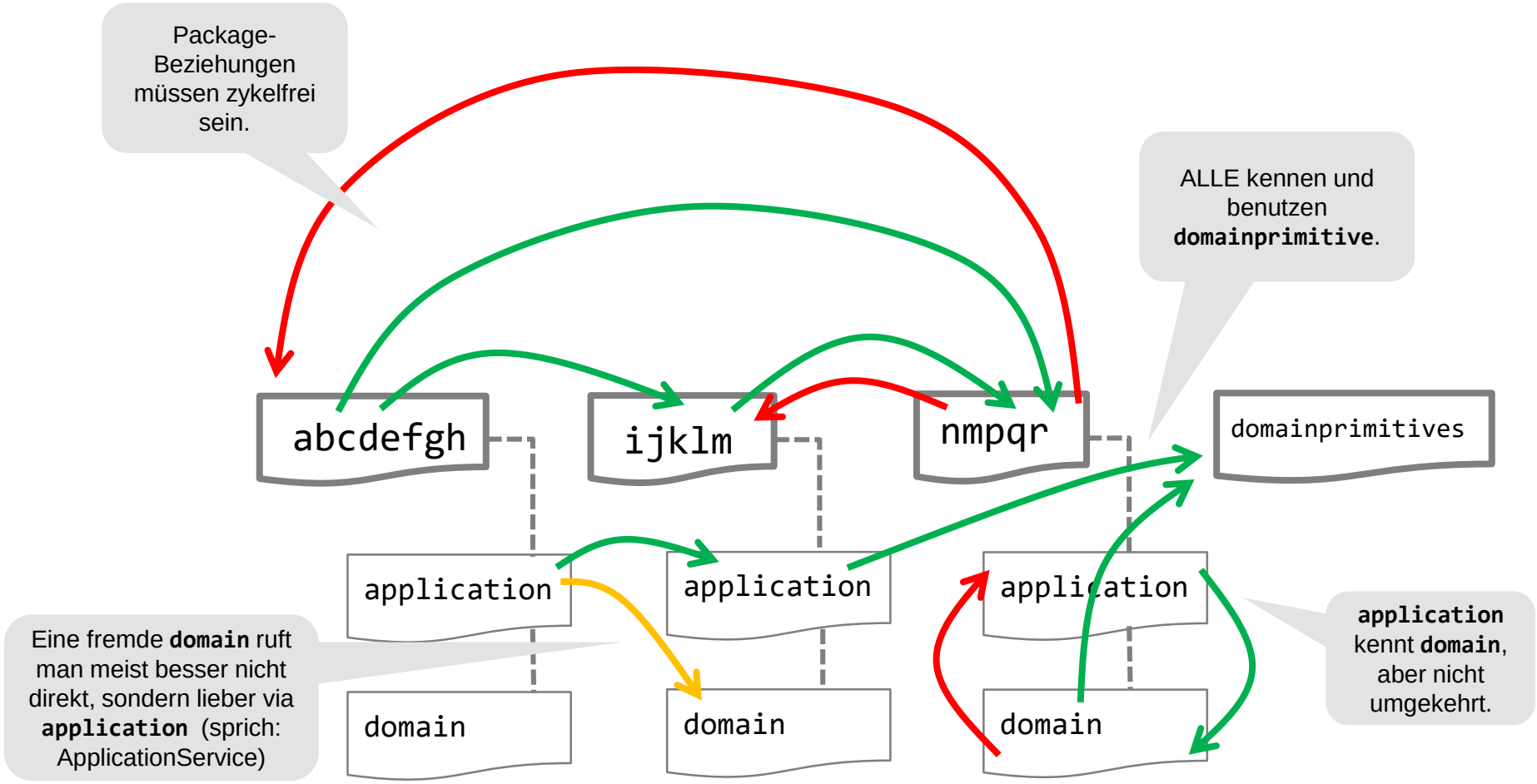
elle: [Evans], S. 53, zusammengefasst durch den Vortragenden

Sinnvolle Package-Strukturen



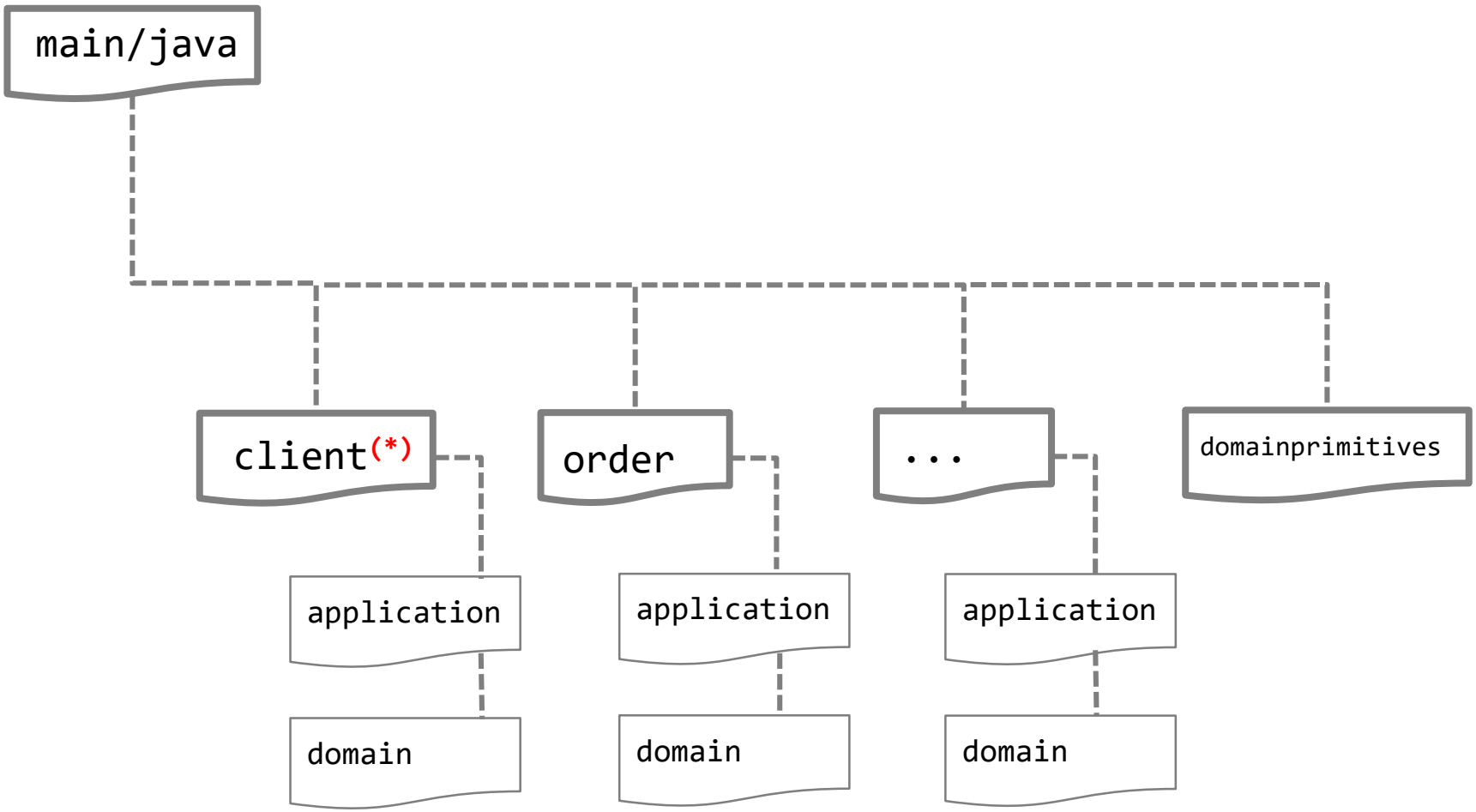
(*) *Heißt in Ihrer Individualisierung natürlich wahrscheinlich anders …*

Sinnvolle Package-Strukturen



(*) *Heißt in Ihrer Individualisierung natürlich wahrscheinlich anders …*

In unserem Fall



(*) *Heißt in Ihrer Individualisierung natürlich wahrscheinlich anders …*

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

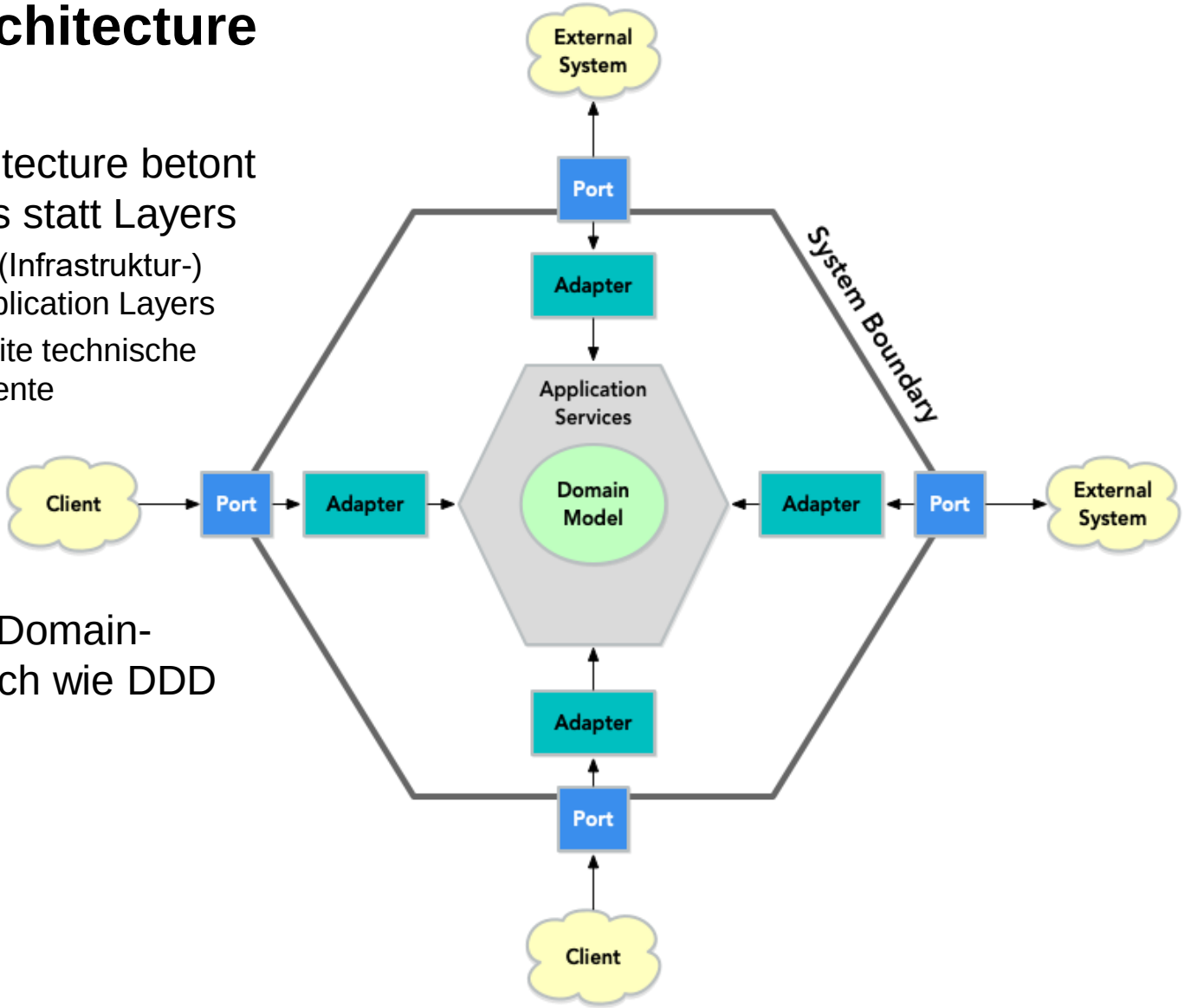
- Im Weiteren stelle ich die Konzepte anhand eines Blogs von Herberto Graca vor, der eine sehr gute Infografik produziert hat (siehe nächste Folie)
- Quellen

(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

Bild: Holmström, P. (2020). *Domain-Driven Design and the Hexagonal Architecture*. Vaadin. https://vaadin.com/learn/tutorials/ddd/ddd_and_hexagonal

DDD / Hexagonal Architecture / Onion Architecture

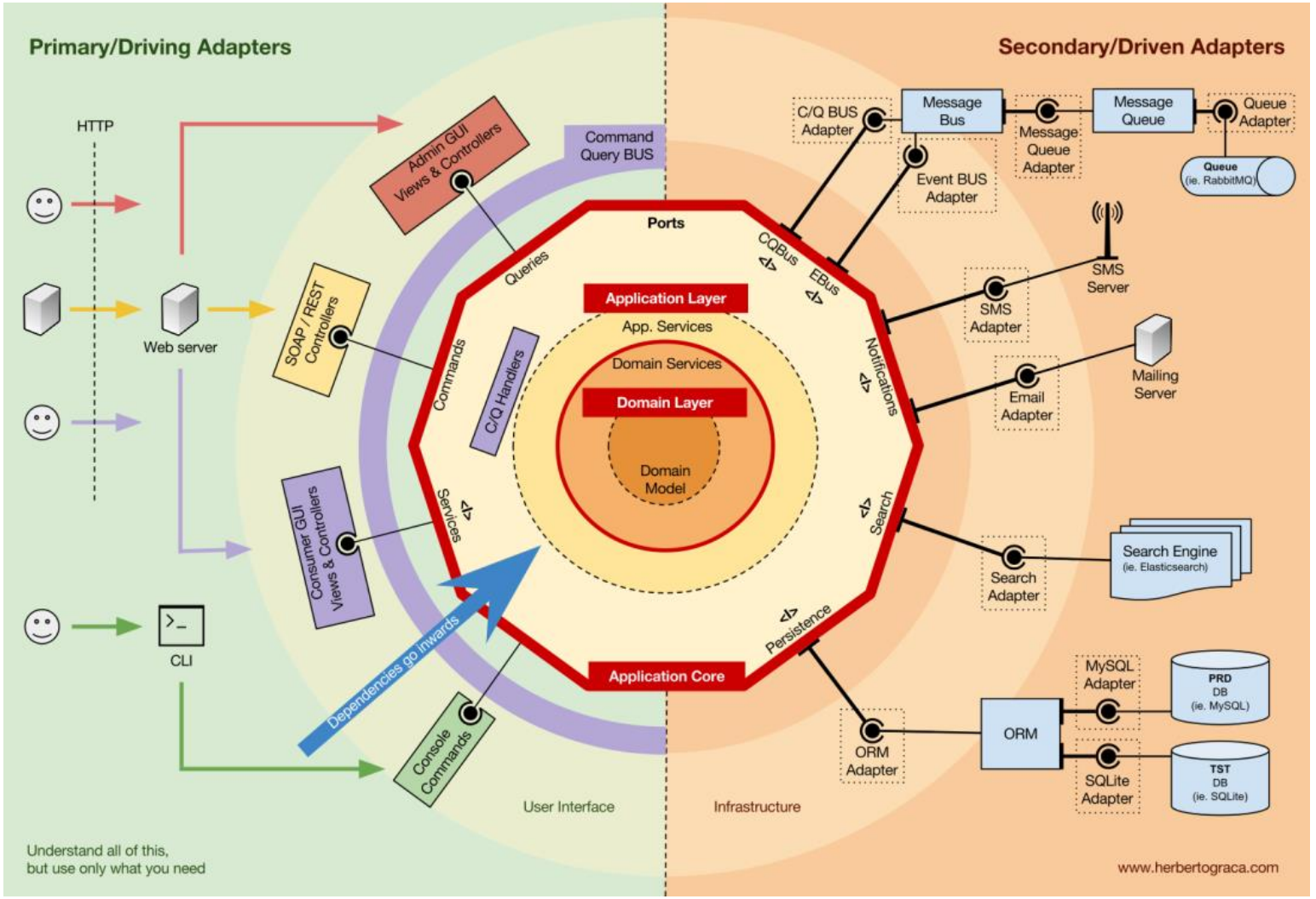


Bild: Graca, H. (2017, November 16). DDD, Hexagonal, Onion, Clean, CQRS, … How I put it all together. *@hgraca*. https://herbertograca.com/2017/11/16/explicit-architecture-01-ddd-hexagonal-onion-clean-cqrs-how-i-put-it-all-together/

Clean Architecture (Bob Martin)

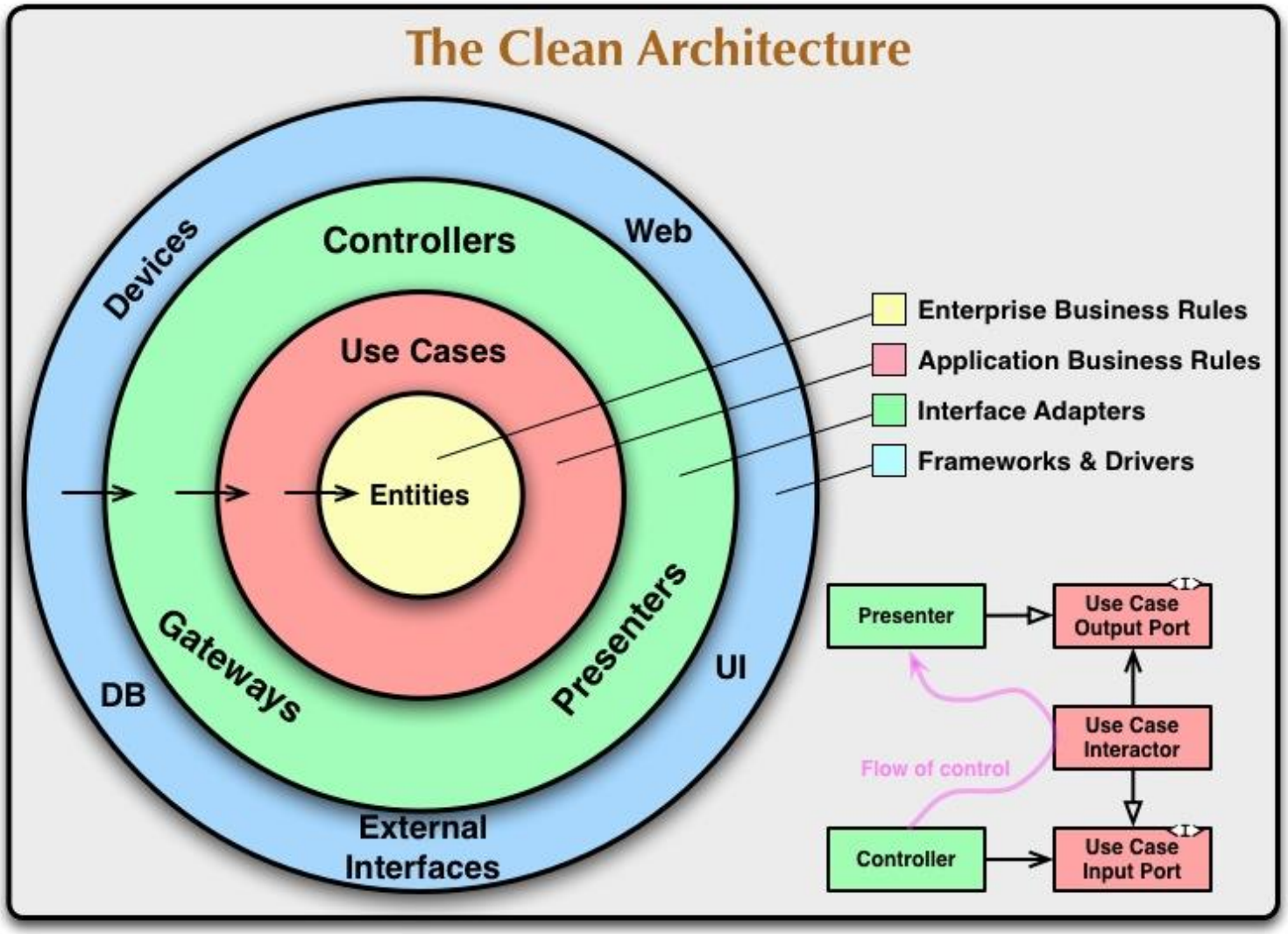


Bild: Martin, R. C. (2012, August 13). The Clean Architecture. *Clean Coder Blog*. https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

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