

Mehrschichten- Architektur

Matthias Wallnöfer
(*) Einige Inhalte: Ruth Breu

Motivation

- **Entwurfsmuster:** Verständnis für Umsetzung von Aufgabenstellungen nach Kochbuch
 - Vorgefertigte Ideen → Verlässliche Lösungen (Objektbeobachtung ↔ *Observer*, Schnittstellenanpassung ↔ *Adapter* etc. etc.)
- Bei kleineren Programmen sind wir fertig

Motivation

- **Entwurfsmuster:** Verständnis für Umsetzung von Aufgabenstellungen nach Kochbuch
 - Vorgefertigte Ideen → Verlässliche Lösungen (Objektbeobachtung ↔ *Observer*, Schnittstellenanpassung ↔ *Adapter* etc. etc.)
- Bei kleineren Programmen sind wir fertig

...Oder doch nicht ?



Motivation

- Die **Anwendungsfälle** (Use-Cases) unserer Programme fallen in unterschiedliche *Aufgabenbereiche*:
 - Eingabe & Ausgabe
 - Algorithmen, Verarbeitungslogik
 - Kommunikation
 - Datenablage

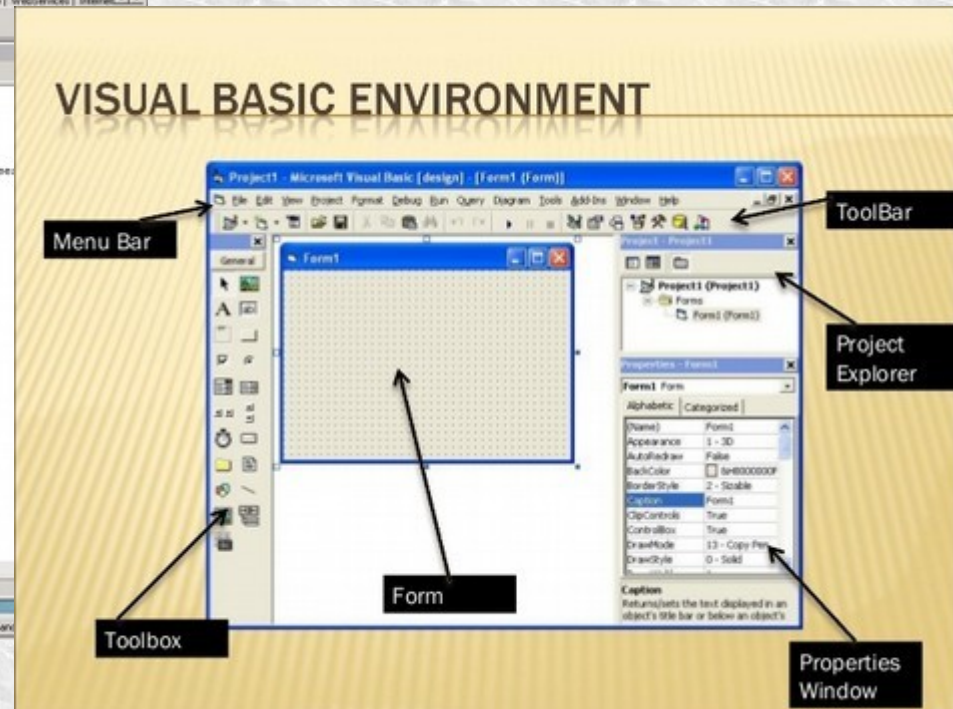
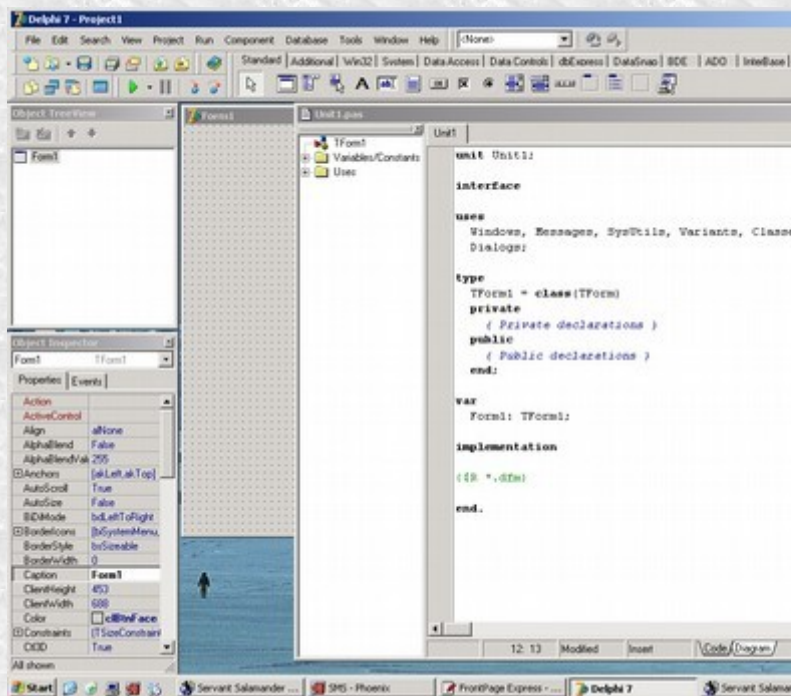
Motivation

- Die **Anwendungsfälle** (Use-Cases) unserer Programme fallen in unterschiedliche *Aufgabenbereiche*:
 - Eingabe & Ausgabe
 - Algorithmen, Verarbeitungslogik
 - Kommunikation
 - Datenablage
- **Diese Aspekte sollten in der Planungsphase Berücksichtigung finden!**



Rückblick

- In den 90er und den 2000er RAD- (*Rapid Application Development*) bzw. 4GL- (*Fourth Generation Language*) IDEs äußerst populär



RAD - Pros

- Entwicklung erfolgt in drei Schritten
 - 1)Form zusammenklicken**
 - 2)Komponenten reinziehen**
 - 3)In den Handlern Code entwickeln**
- Relativ schnelle Fertigstellung
- Kleine Änderungen einfach umsetzbar
- In gewissen Umfeldern ideal:
Prototyping, Ansteuerungen (Elektronik)

RAD - Contras

- Größere Änderungen aufwendig
- Code meist nicht ausreichend strukturiert, alles ineinander verwoben (UI, DB etc.)
- Man ist von Hilfsmitteln der IDE abhängig
 - IDE/Compiler nicht leicht austauschbar
 - → Vendor Lock-In

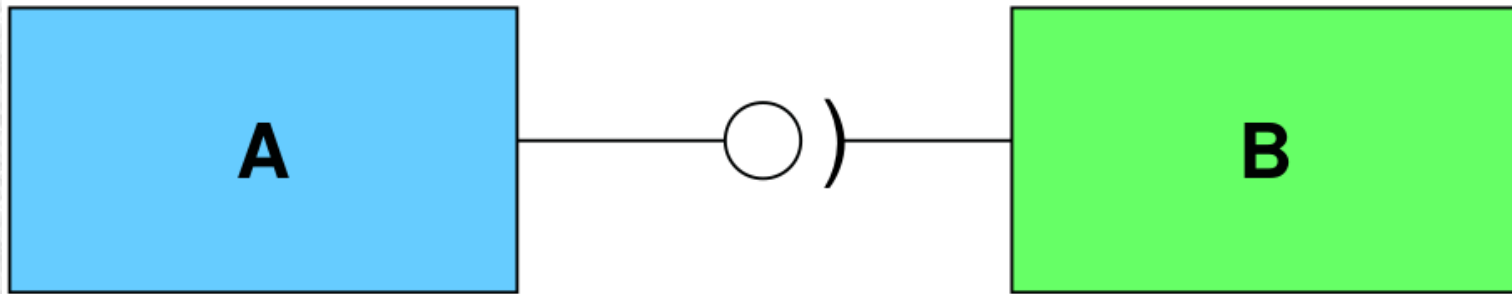
RAD - Contras

- Größere Änderungen aufwendig
- Code meist nicht ausreichend strukturiert, alles ineinander verwoben (UI, DB etc.)
- Man ist von Hilfsmitteln der IDE abhängig
 - IDE/Compiler nicht leicht austauschbar
 - → Vendor Lock-In
- **Schlechte Wartbarkeit**



Konzept der Komponente

- Klare Schnittstelle zwischen Teilsystemen (*)



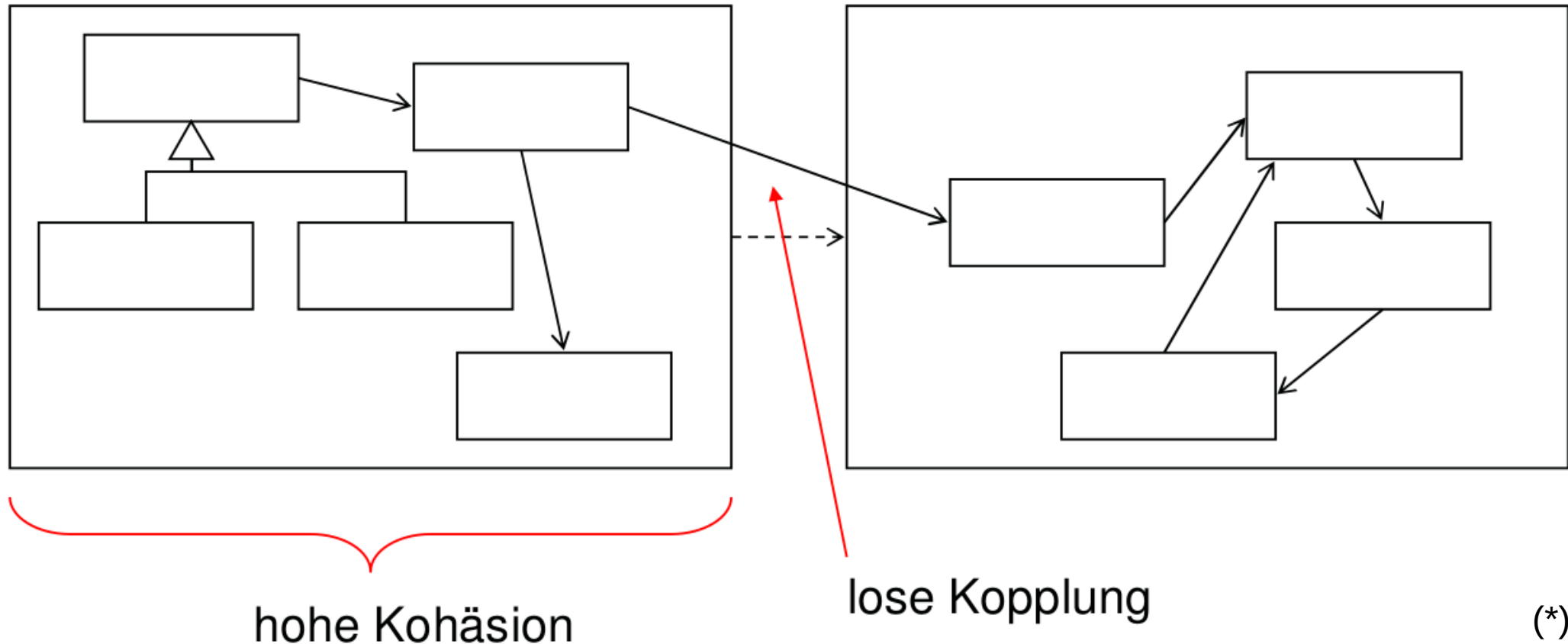
D.h. **B** greift auf **A** mittels Schnittstelle zu

- **Vertragsprinzip** (rely-guarantee)
 - **A** garantiert die Implementierung der SS
 - **B** verlässt sich, dass **A** SS zur Verfügung stellt

Komponentenbildung

- **Unabhängigkeit:** Komponenten können unabhängig entwickelt werden
- **Austauschbarkeit:** Komponenten können unter Einhaltung der Schnittstellen getauscht werden
 - Neue UI, neue DB
- **Erweiterbarkeit:** Einfache Integration neuer Bestandteile
- **Testbarkeit:** Unabhängige Tests möglich

Kohäsion-Kopplung



- **Kohäsion:** Klassen innerhalb einer Komponente bilden ein enges Beziehungsgeflecht (*hohe Kohäsion*)
- **Kopplung:** Klassen in unterschiedlichen Komponenten sind nur *lose gekoppelt*

Tests

- Zwei Prinzipien:
 - **Black-Box** (ohne Code)
 - Test über Schnittstellen wie APIs (Frameworks wie [Fitnesse](https://fitnesse.org) fitnesse.org)
 - Automatisierte UI Tests (Frameworks wie [SeleniumHQ](https://seleniumhq.org) seleniumhq.org)
 - **White-Box/Glass-Box** (mit Code)
 - Assertions im Code (Instruktion **assert** u.a. in Java)
 - Unit-Testframeworks wie [JUnit](https://junit.org) (junit.org), [nunit](https://nunit.org) (nunit.org)
- In beiden Fällen:
 - Automatisierte Integration in VCS ([Continuous Integration](https://jenkins.io)).
Beispiel: [Jenkins](https://jenkins.io) (www.jenkins.io)
- Auch vor der Implementierung möglich/erwünscht

Prototypen

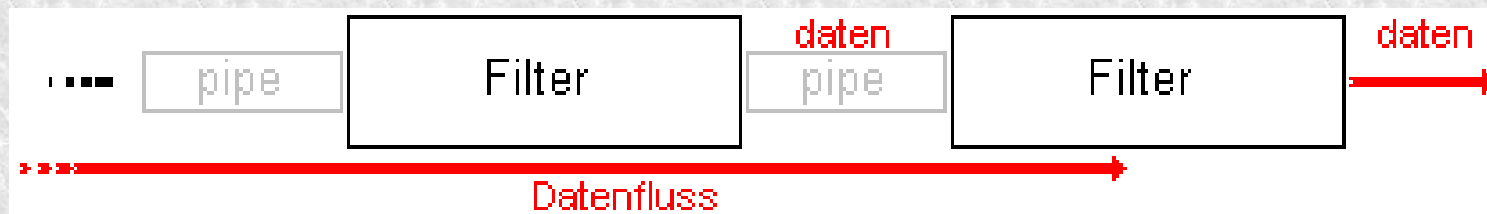
- Voraussetzung für **Frühzeitige Tests** und **Abstimmung der Schnittstellen**
- Varianten:
 - **Oberfläche/UI**
 - Abnehmer: Erster Überblick, Änderungswünsche
 - Entwickler: Interaktive Tests
 - **Code (= Ersatz für Komponenten)**
 - **Stub** (Minimaler Code für erfolgreichen Build, sonst nichts)
 - **Dummy** (wie Stub, aber mit einfacher Testlogik)
 - **Mock** (wie Stub, aber mit komplexer Testlogik)
- Prototypen auch in anderer Programmiersprache möglich

Architekturmuster

- Analog zu Entwurfsmustern gibt es auch „Best Practices“ für Architekturen
- Drei Makrokategorien
 - **Chaos-zu-Struktur** („Mud-to-structure“)
Bsp. *Pipes-Filter, Schichtenarchitektur*
 - **Verteilte Systeme** (Verteilte Ressourcen und Dienste)
Bsp. *Client-Server, Peer-to-Peer*
 - **Interaktive Systeme** (Umgang Mensch mit Maschine)
Bsp. *Model-View-Controller*

Pipes-Filter

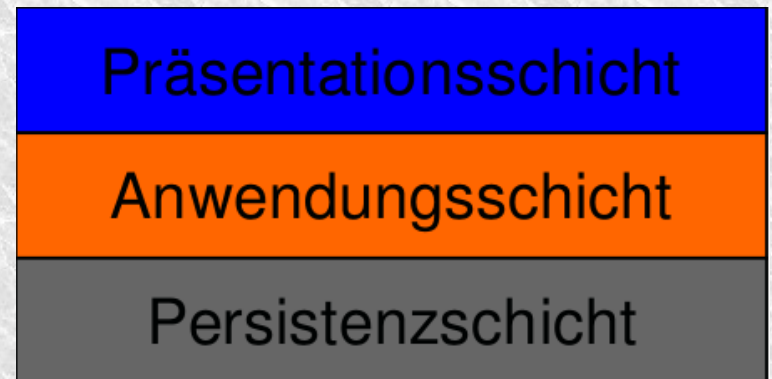
- Auch als **Kette** bezeichnet, wie Produktionskette
- Pipes gibt es unter allen OS (GNU/Linux, Unix, Windows)
- Streams in Programmiersprache wie C++, C#, Java
- Interpreter und Übersetzer (Compiler)



https://de.wikipedia.org/wiki/Pipes_und_Filter#/media/File:PipesFilters.png

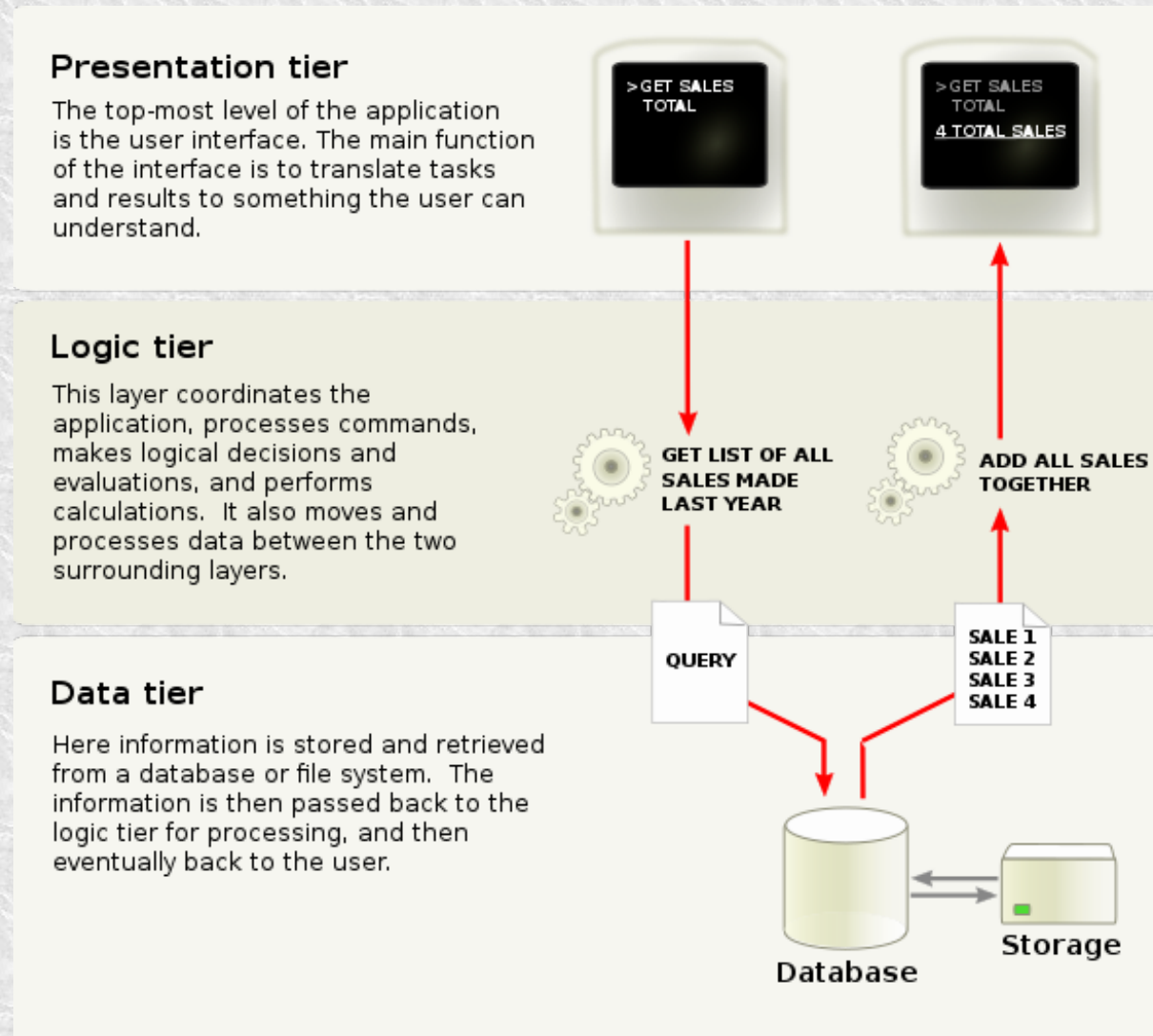
Dreischichten-Architektur

- Der Klassiker
- **Code** wird in **Komponenten** strukturiert
- Jede **Komponente** gehört einer **Schicht** (*Layer* bzw. *Tier*) an
- Es kann sein, dass es keine Persistenz braucht, dann werden es einfach zwei Schichten



(*)

Drei-Schichten-Architektur



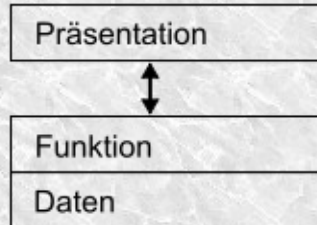
https://en.wikipedia.org/wiki/Multitier_architecture#/media/File:Overview_of_a_three-tier_application_vectorVersion.svg

Mehrschichtenarchitektur

- Drei Schichten können beliebig erweitert werden → **Multi-Tier**
- Haben wir zwischen Persistenz und Applikation/Geschäftslogik einen **Cache**?
- Die UI läuft im Browser (MVC) und kommuniziert mit der Geschäftslogik auf dem Server, d.h. dazwischen existiert eine **API/Middleware**
- Auch bei der Persistenz muss es sich nicht um eine DB handeln, auch hier **Middleware** bzw. **Services** möglich

Mehrschichtenarchitektur

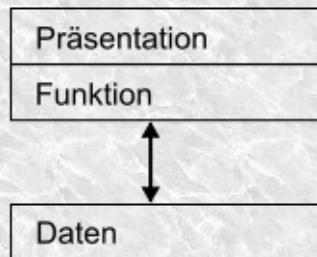
- unterschiedliche Aufteilung



THIN-CLIENT, AKTIVER SERVER

- + zentrale Administration & Wartung
- + Kostenersparnis
- + Sicherheit (nur 1 Server muss geschützt werden)
- + Flexibilität

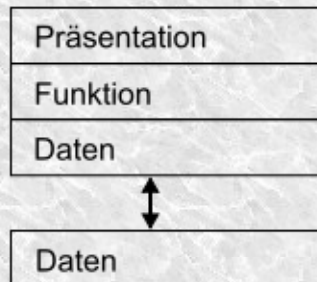
- hohe Belastung für Server



DATEN-SERVER

- Server liefert nur benötigte Daten
- + jeder Client kann selber rechnen → höhere Performance
- + zentrale Datenhaltung bleibt bestehen → Sicherheit

- hoher Installationsaufwand, da jeder Client alle Applikationen braucht



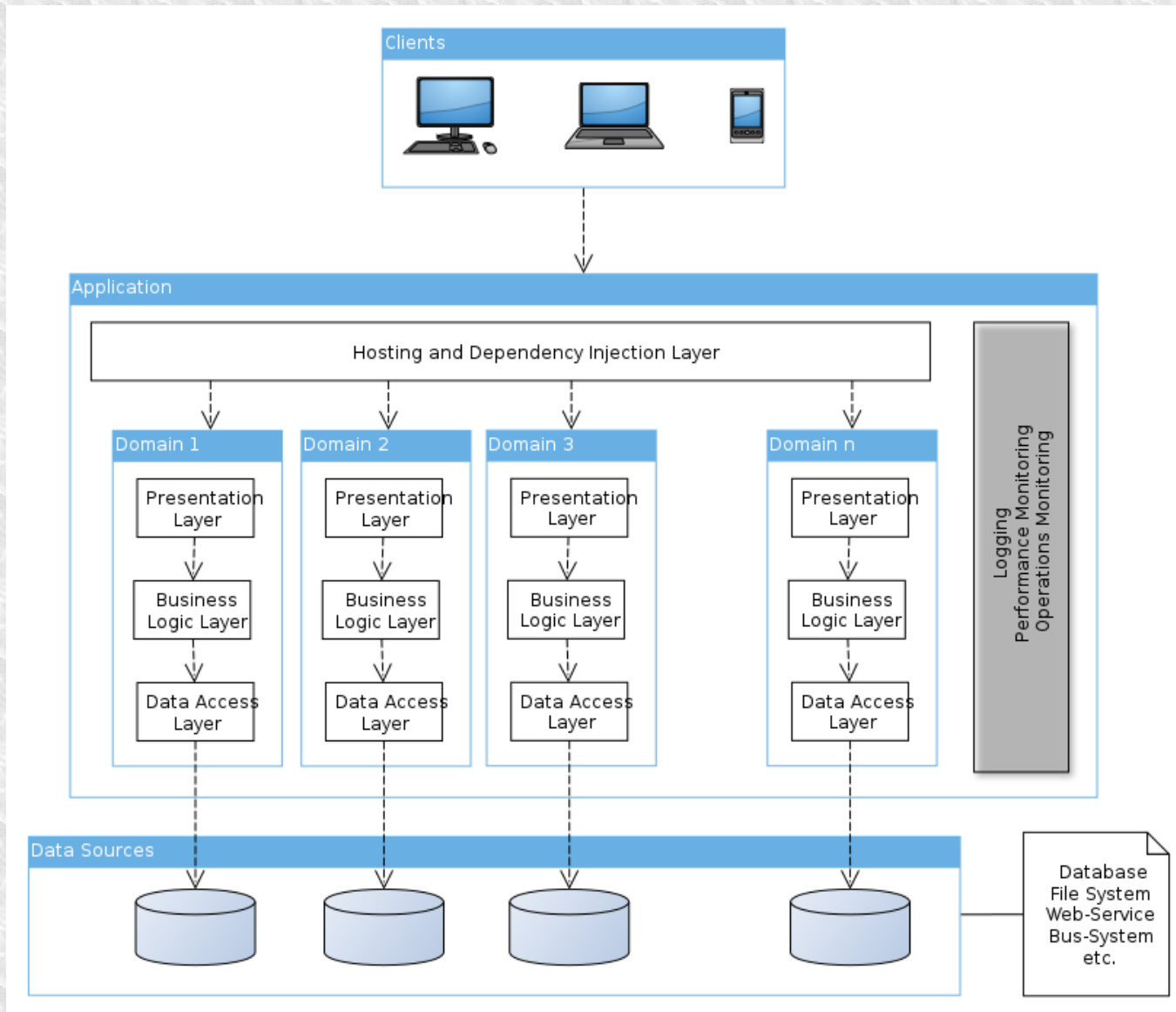
FAT-CLIENT

- + Entlastung Server & Datenleitung durch Plug-Ins
- + Weiterarbeiten möglich, wenn Kommunikation zum Server gestört

- lange Dauer, bis alle Clients Updates haben
- hohe Wartungskosten

<https://de.wikipedia.org/wiki/Schichtenarchitektur#/media/File:Tier-aufteilung.svg>

Mehrschichtenarchitektur



<https://de.wikipedia.org/wiki/Schichtenarchitektur#/media/File:Schichtenarchitektur.svg>

ISO/OSI-Stack

OSI-7-Layer-Model (Open Systems Interconnection Reference Model)

Begriffe: Englisch - Deutsch

7 Application Layer	- Anwendungsschicht
6 Presentation Layer	- Darstellungsschicht
5 Session Layer	- Sitzungs- bzw. Kommunikationsschicht
4 Transport Layer	- Transportschicht
3 Network Layer	- Netzwerk- bzw. Vermittlungsschicht
2 Data Link Layer	- Sicherungsschicht
1 Physical Layer	- Bitübertragungsschicht

PC im Netzwerk
A



W http://www.wikipedia.org



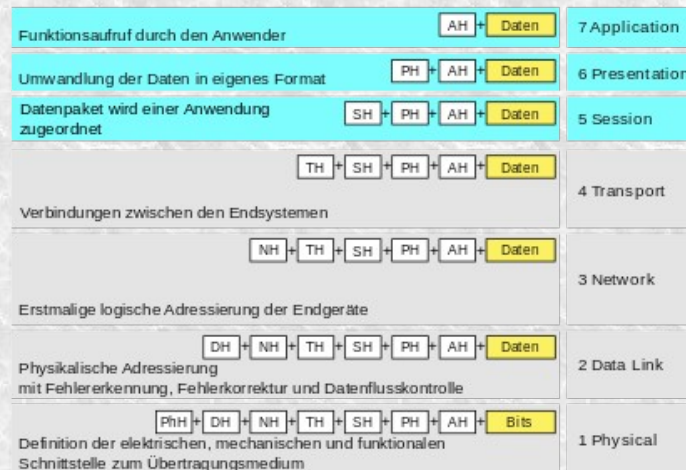
Der Benutzer empfängt lediglich die Antwort des Servers ("wikipedia.org"-Startseite). Im Allgemeinen bekommt er von der Schachtelung seines Seitenaufrufs durch die Ebenen seines PCs (abwärts) und vom Parsen der Antwort des Servers zurück durch die Ebenen seines PCs (aufwärts) nichts mit!

Server schickt die entsprechenden Daten über die selbe Methode zurück. (s.u.)

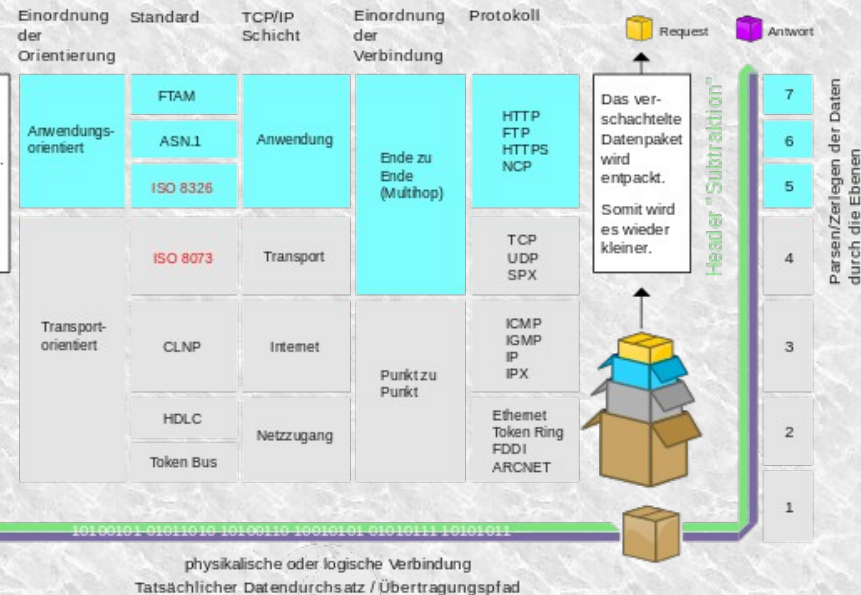
Server im Netzwerk
B



Zusammenbau des Pakets:
(Package Assembling/Formatting)



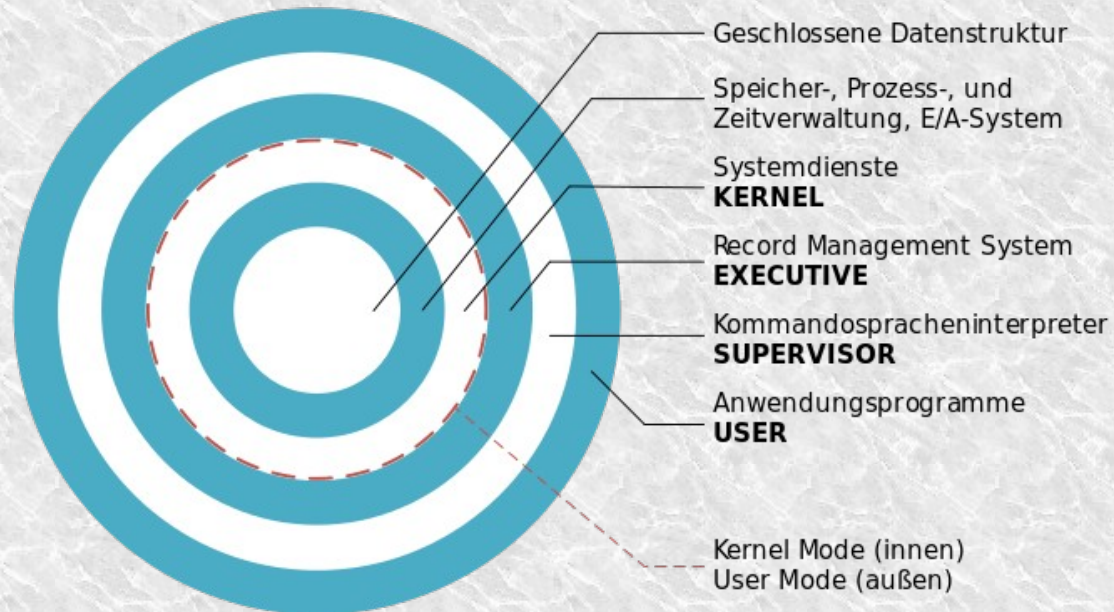
Zusammensetzung der Abkürzungen oben:
Anfangsbuchstabe der Schicht und "H" für Header.
z.B. Application Header = AH



Autor: gob (www.godofbytes.de)
Bildversion 3.0 (14. Okt. 2006)
SVG-Neuaufbau + Korrektur: der_Fahrer
Dateiversion 3.1 (21. Juni 2010)

https://upload.wikimedia.org/wikipedia/commons/thumb/1/14/OSI7Layer_model_3.1.svg/1024px-OSI7Layer_model_3.1.svg.png

Betriebssystem



<https://de.wikipedia.org/wiki/Schichtenarchitektur#/media/File:Schalenmodell.svg>

Case Study

- Wir vertiefen unser Wissen anhand eines größeren Projekts
- Dazu bietet sich „**Open Data Hub Südtirol**“ an
 - <https://www.opendatahub.com/>
 - Entwickelt unter der Leitung vom NOI techpark
 - Dient als Datensammler und -bereitsteller von Öffentlichem Gemeingut (= *Open Data*)
 - Plattform selbst quelloffen bzw. Freie Software

Open Data Hub Südtirol

Open Data Hub Südtirol - Alto Adige

Open Data Hub is your access point to South Tyrol's relevant data. You and your service providers can use Open Data Hub for all channels of digital communication. The data is updated on a regular basis and easily accessible.

Open Data Hub Südtirol

- **Datenquellen**

- Bike-Sharing
- Car-Pooling
- Meteo
- Parking
- Tourismusdaten
- Und viele andere mehr

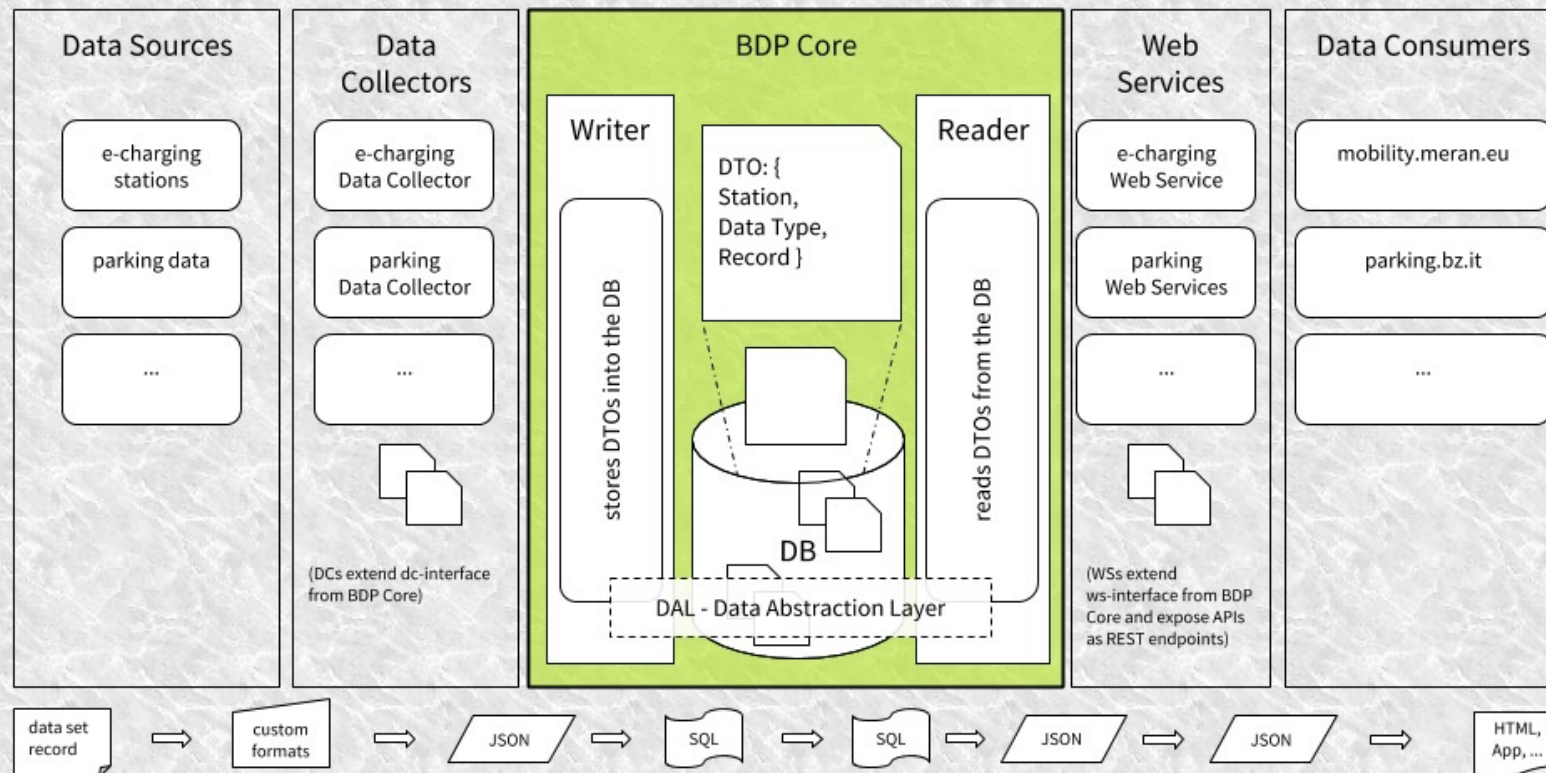


- **Mock-Ups**

- Katalog: <https://webcomponents.opendatahub.com/>
- Parkplätze Bozen/Merano: <https://parking.bz.it/>
- Busse in Bozen: <http://bus.bz.it>
- Mobility Merano: <https://mobility.merano.eu/>

Open Data Hub Südtirol Architektur

Open Data Hub Elements for Data Flow and Data Transformation v1.0



<https://opendatahub.readthedocs.io/en/latest/intro.html#odh-architecture>

Open Data Hub Südtirol

Dokumentation

- Wie entwickle ich meine eigene App?
→ *API-Dokumentation*,
<https://opendatahub.readthedocs.io/en/latest/datasets.html>
- Wie ist Open Data Hub Südtirol intern aufgebaut?
Kann ich Open Data Hub Südtirol erweitern?
→ *Technische Dokumentation*,
<https://opendatahub.readthedocs.io/en/latest/guidelines.html>
- Allgemein:
<https://opendatahub.readthedocs.io/en/latest/index.html>