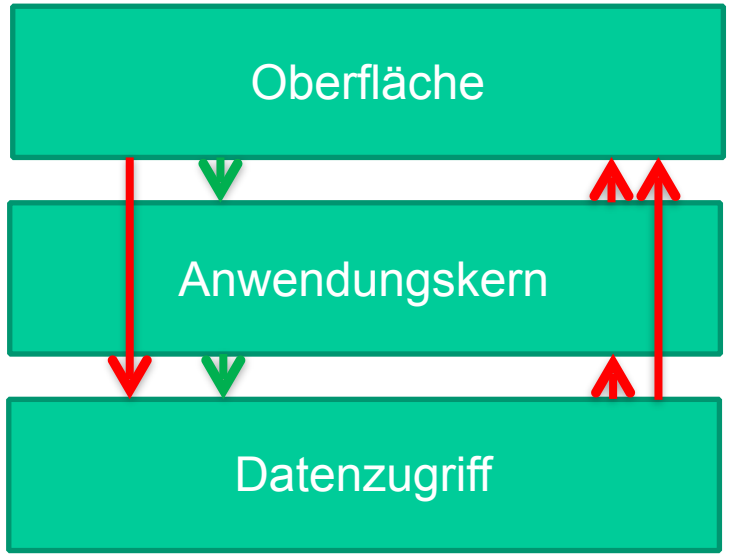
UI Architektur

Hochschule **Rosenheim**
University of Applied Sciences

Die klassische Schichten-Architektur

- Strikt: Eine Schicht kennt nur die direkt drunterliegende
- Non-Strikt: Eine Schicht kennt alle drunterliegenden (Achtung: Steigende Komplexität pro Schicht!



Wie realisiert man eine strikte Schichtenarchitektur an der UI?

- Service-orientierte Schnittstelle zwischen UI und Anwendungskern (Anwendungskern API)
  - Möglichst geringe Abhängigkeiten (Anzahl Methoden und Parameterklassen)
  - Zustandslos
  - KIS (Keep it Simple): Einfache Schnittstellen mit wenig Methoden, einfachen Datentypen bzw. Datentransferobjekten (POJO) ermöglichen Zwei- und Drei-tier Betrieb sowie Mock-Ups für den Testbetrieb

  - http://martinfowler.com/eaaCatalog/dataTransferObject.html

Hochschule Rosenheim
University of Applied Sciences

Objektorientierte vs. serviceorientierte Schnittstellen

- Objektorientiert

```
Kunde k = new Kunde();
k.setName(„“)
k.setAddress(„“)
Bestellung b = new Bestellung(k);
b.setProduct(…);
BestellungDAO dao = new BestellungDAO();
dao.persist(k);
dao.persist(b);
```

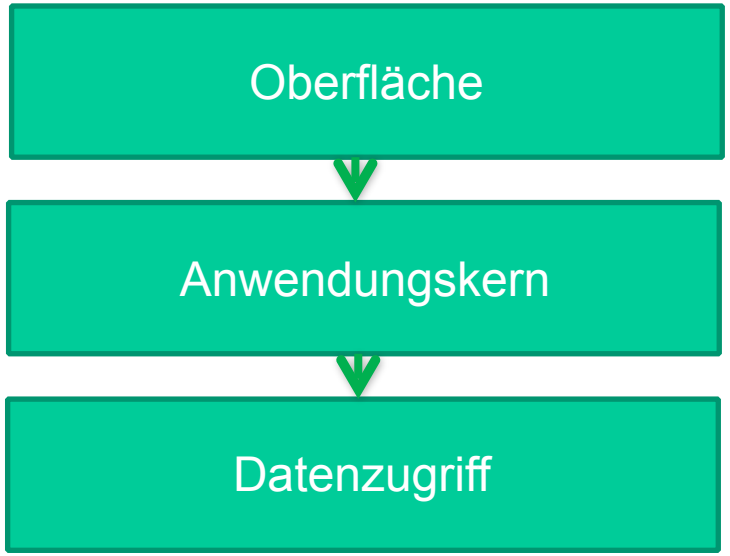
- Serviceorientiert

```
BestellungAPI service
   = new BestellungAPI();

service.orderForCustomer(
   new CustomerVO(id,name,
      address),
   new ProductVO(id, amount)
   );
```

Hochschule Rosenheim
University of Applied Sciences

Nur eine strikte Schichten-Architektur entkoppelt Technologiewissen!



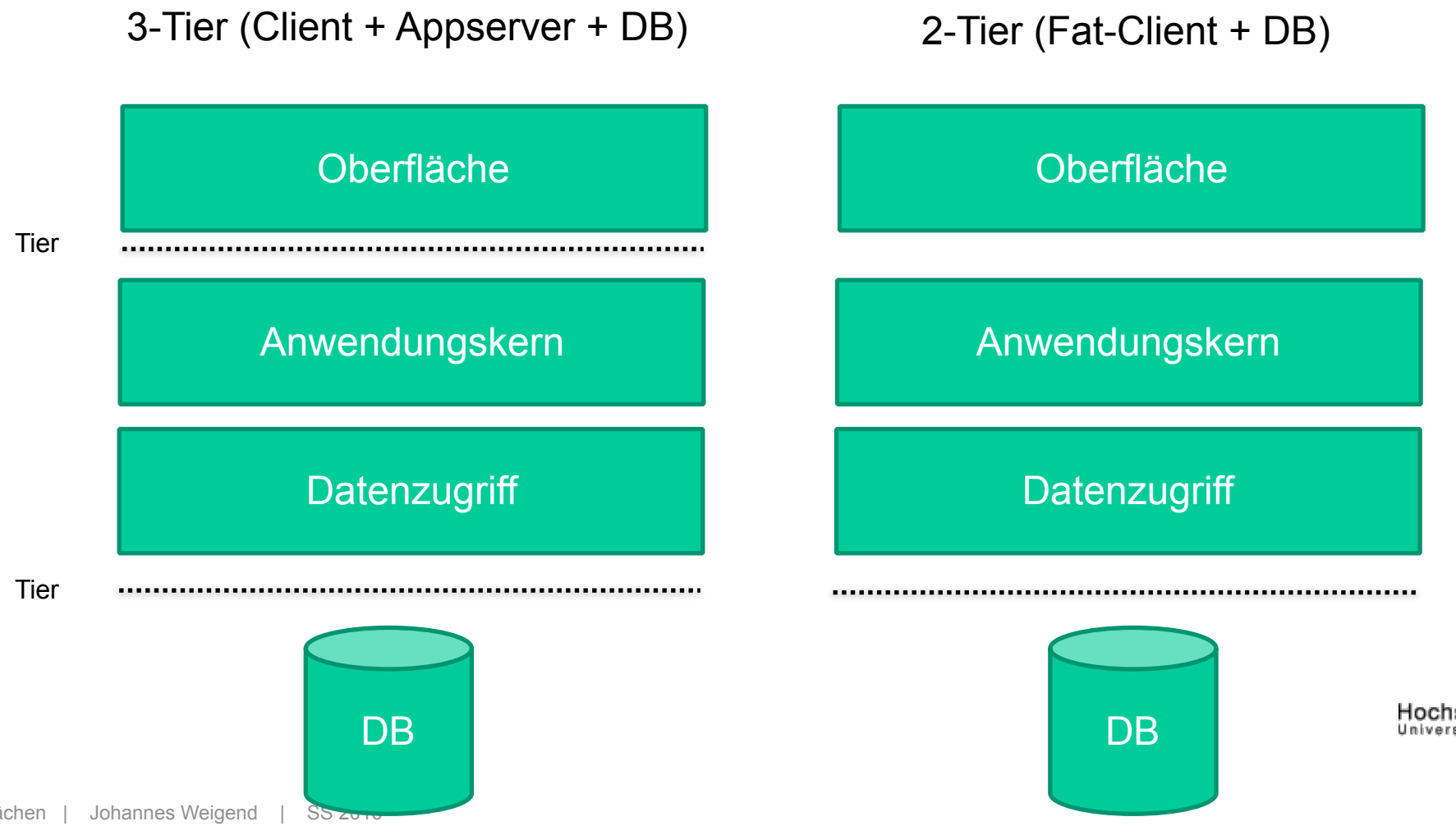
z.B. JavaFX

z.B. Bibliothek für Finanzmathematik

z.B. JPA

Hochschule Rosenheim
University of Applied Sciences

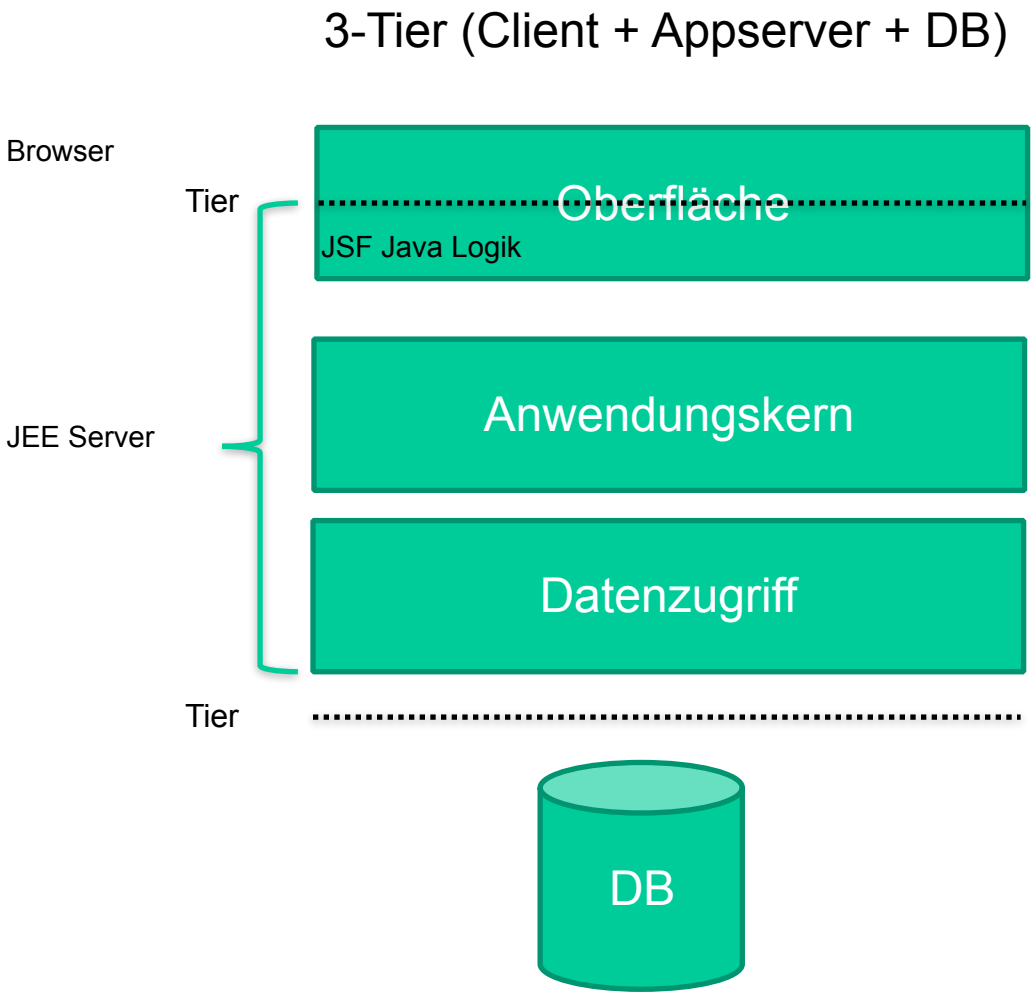
Die Schichten-Architektur ermöglich Zwei- und Dreitier Betrieb



Hochschule Rosenheim
University of Applied Sciences

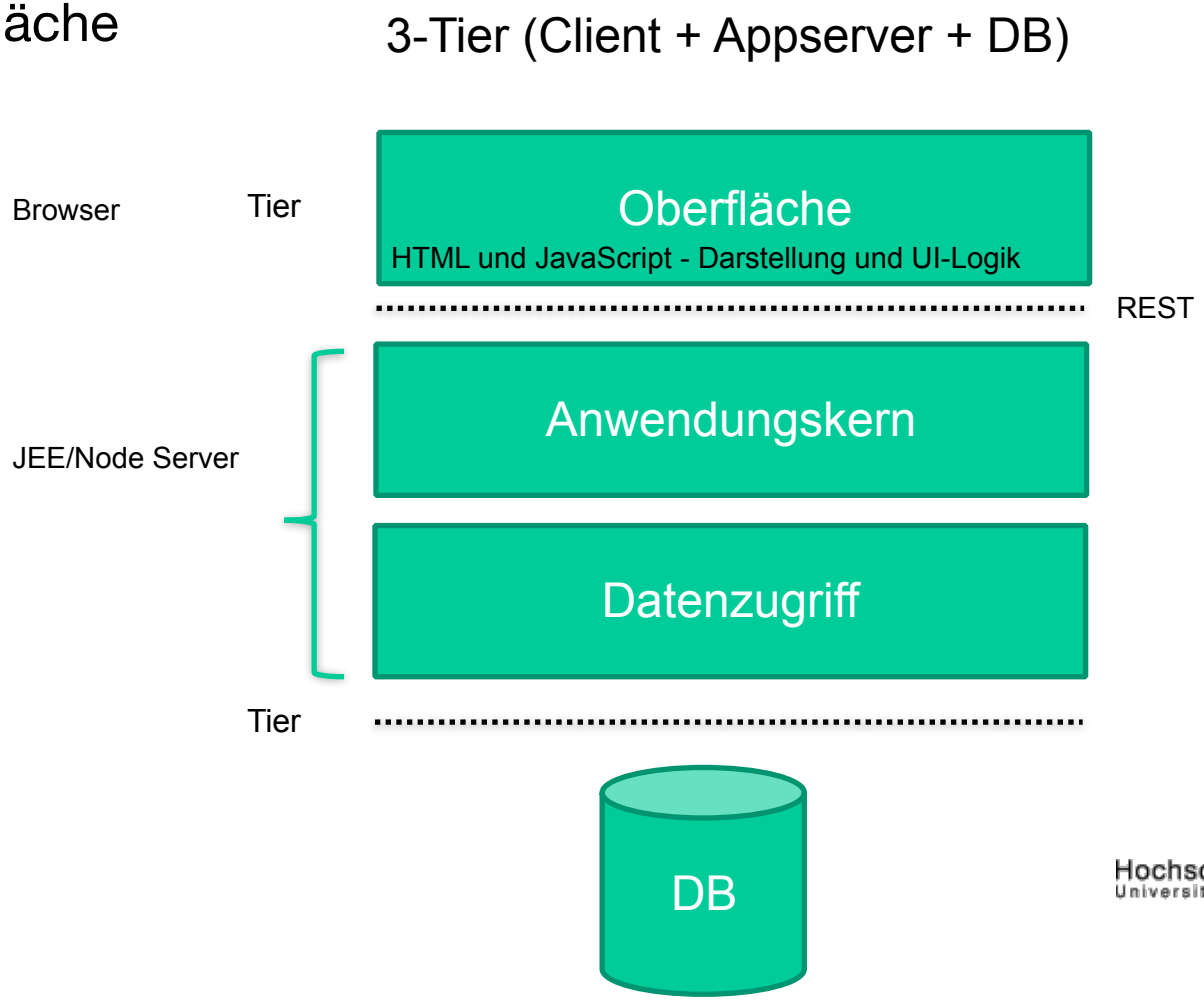
Bei Weboberflächen verschiebt sich die Tier-Grenze oft in den Oberflächen-Layer

- Beispiel Java Server Faces



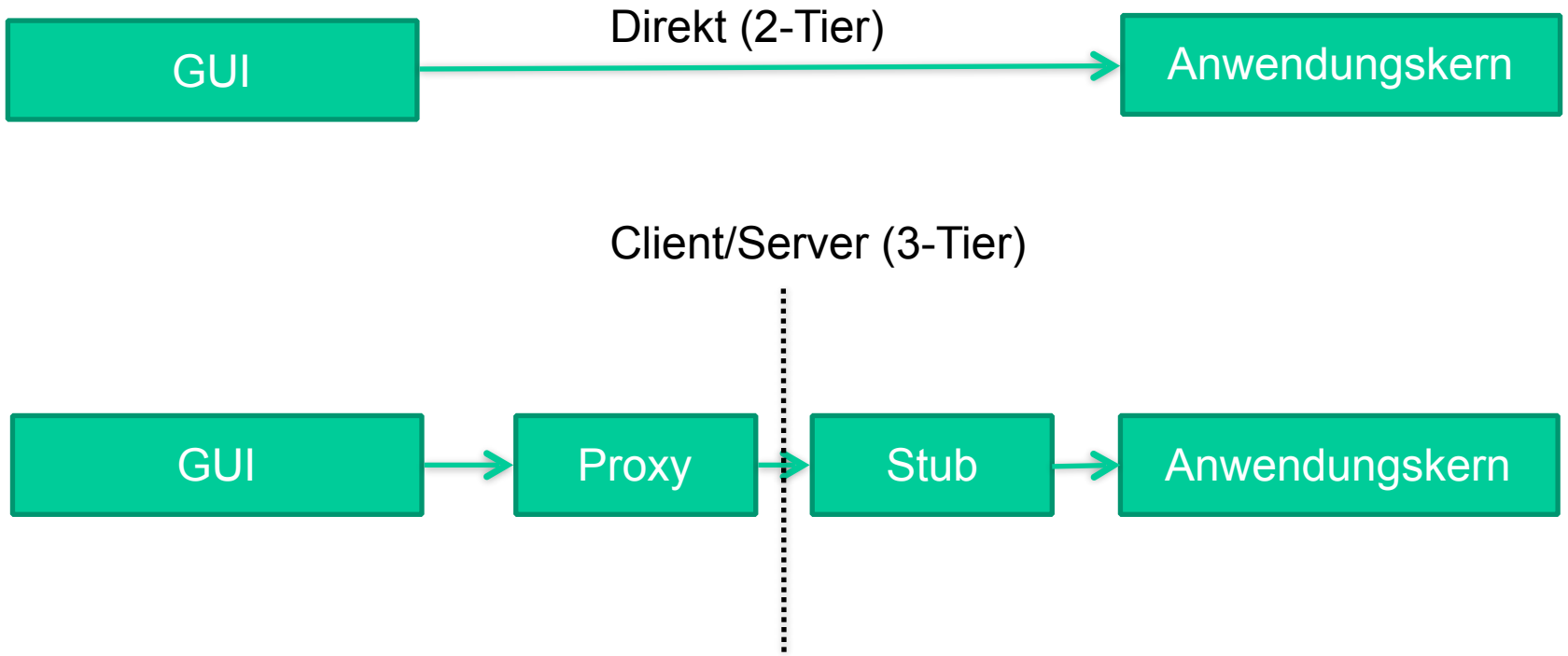
Moderne Rich-Clients mit JavaScript haben diesen Problem nicht.

- Beispiel: HTML5 / JS Oberfläche



Hochschule Rosenheim
University of Applied Sciences

Lokationstransparenz ist zwischen UI und Anwendungskern Standard



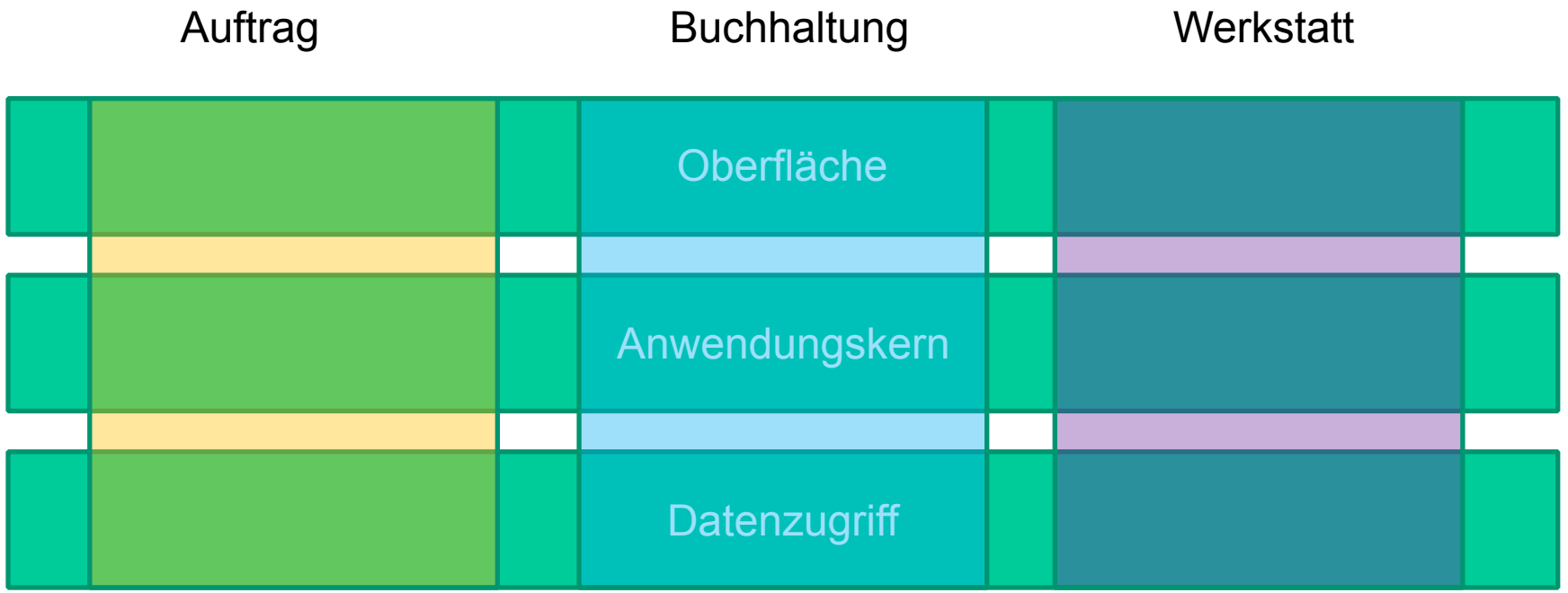
Typische Technologien
- REST (XML/JSON)
- SOAP
- Binärprotokolle (RMI, Corba …)

Hochschule Rosenheim
University of Applied Sciences

Eine Säulenarchitektur ist für fachlich eigenständige Teilanwendungen sinnvoll

Hochschule **Rosenheim**
University of Applied Sciences

Oft werden beide Ansätze kombiniert



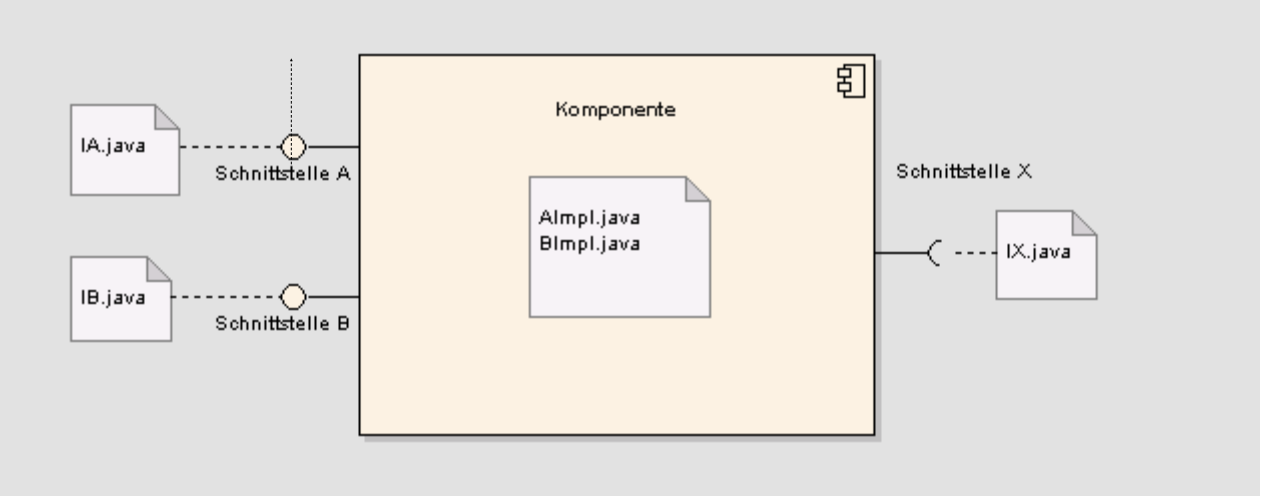
Hochschule Rosenheim
University of Applied Sciences

Von der Schichtenarchitektur zur Komponentenarchitektur

- Eine Schichten- / Säulen-Architektur lässt Fragen offen
  - Keine Aussage über erlaubte Querbeziehungen (Buchhaltung -> Auftrag, Auftrag -> Buchhaltung ?)
  - In der Regel sehr technisch motiviert. Fachliche Schichtenbildung ist eher die Ausnahme

Hochschule **Rosenheim**
University of Applied Sciences

Was sind Komponenten?



- Komponenten gruppieren Klassen und Schnittstellen zu einer logischen Einheit
- Komponenten exportieren und importieren Schnittstellen
- Schnittstellen exportieren Schnittstellen
- Komponenten / Schnittstellen können auf Artefakte der Implementierung (Klassen, Interfaces) abgebildet werden.
  - Komponente → n Klassen/Interfaces
  - Schnittstelle → n Klassen/Interfaces

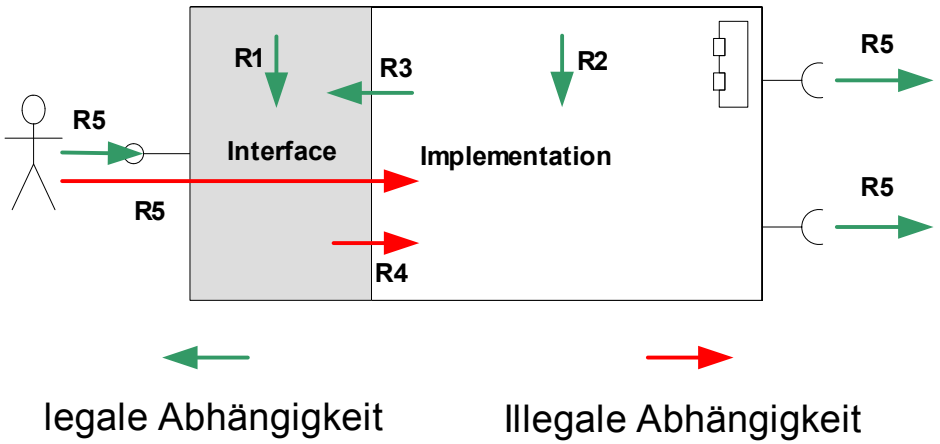
Hochschule Rosenheim
University of Applied Sciences

Komponenten, Schnittstellen und Konfiguration

- Schnittstelle definiert Operationen:
  - mit Syntax, Semantik, Protokoll (synchron, asynchron)
  - kann nicht-funktionale Eigenschaften fordern: Perfomance, Robustheit, Rechengenauigkeit
  - beschreibt das beobachtbare Verhalten einer Komponente
  - zur Schnittstelle gehören die verwendeten Typen und ggfs. abstrakte Hilfsklassen
- Schnittstellen sind nicht alleine lauffähig.
- Jede Komponente exportiert (implementiert) eine oder mehrere Schnittstellen.
- Jede Schnittstelle kann durch beliebig viele Komponenten implementiert werden.
- Jede Komponente importiert beliebig viele Schnittstellen (nicht Komponenten).
- Konfiguration versieht eine Komponente mit Implementierungen der importierten Schnittstellen.

Hochschule Rosenheim
University of Applied Sciences

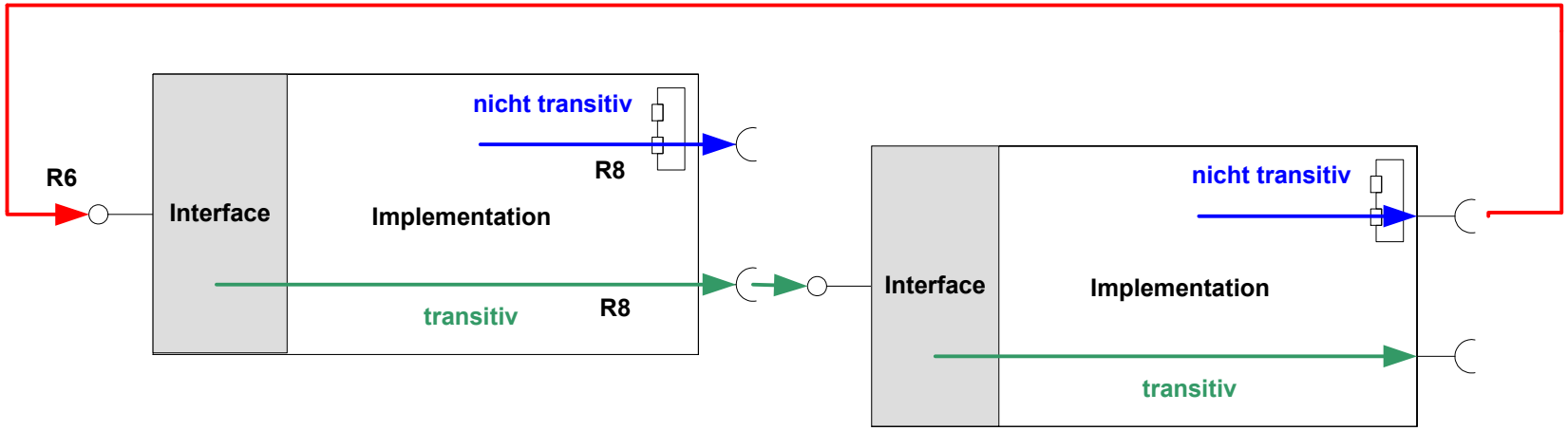
Legale und illegale Abhängigkeiten in Komponenten



- **(R1)** Ein Artefakt (Methode, Attribut, Klasse) der Schnittstelle darf andere Artefakte der identischen Schnittstelle benutzen.
- **(R2)** Ein Artefakt der Implementierung darf andere Artefakte der identischen Komponente benutzen.
- **(R3)** Ein Artefakt der Implementierung einer Komponente C darf alle Artefakte der Schnittstellen von C benutzen.
- **(R4)** Artefakte der Schnittstellen dürfen nur Artefakte der Implementierung nutzen, wenn keine transitive Abhängigkeit zum Schnittstellennutzer entsteht.
- **(R5)** Komponenten dürfen nur über Ihre Schnittstellen verwendet werden.

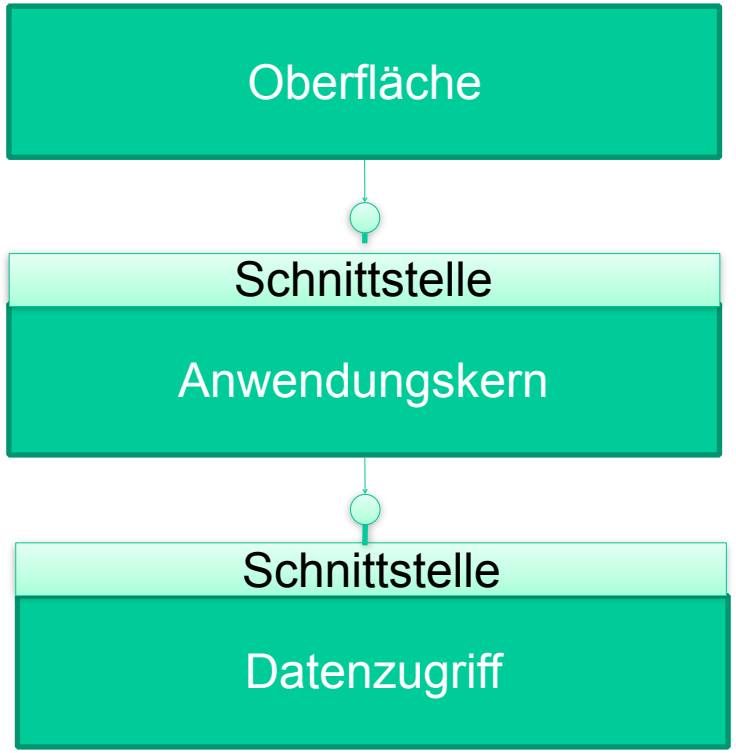
Hochschule **Rosenheim**
University of Applied Sciences

Legale und illegale Abhängigkeiten zwischen Komponenten



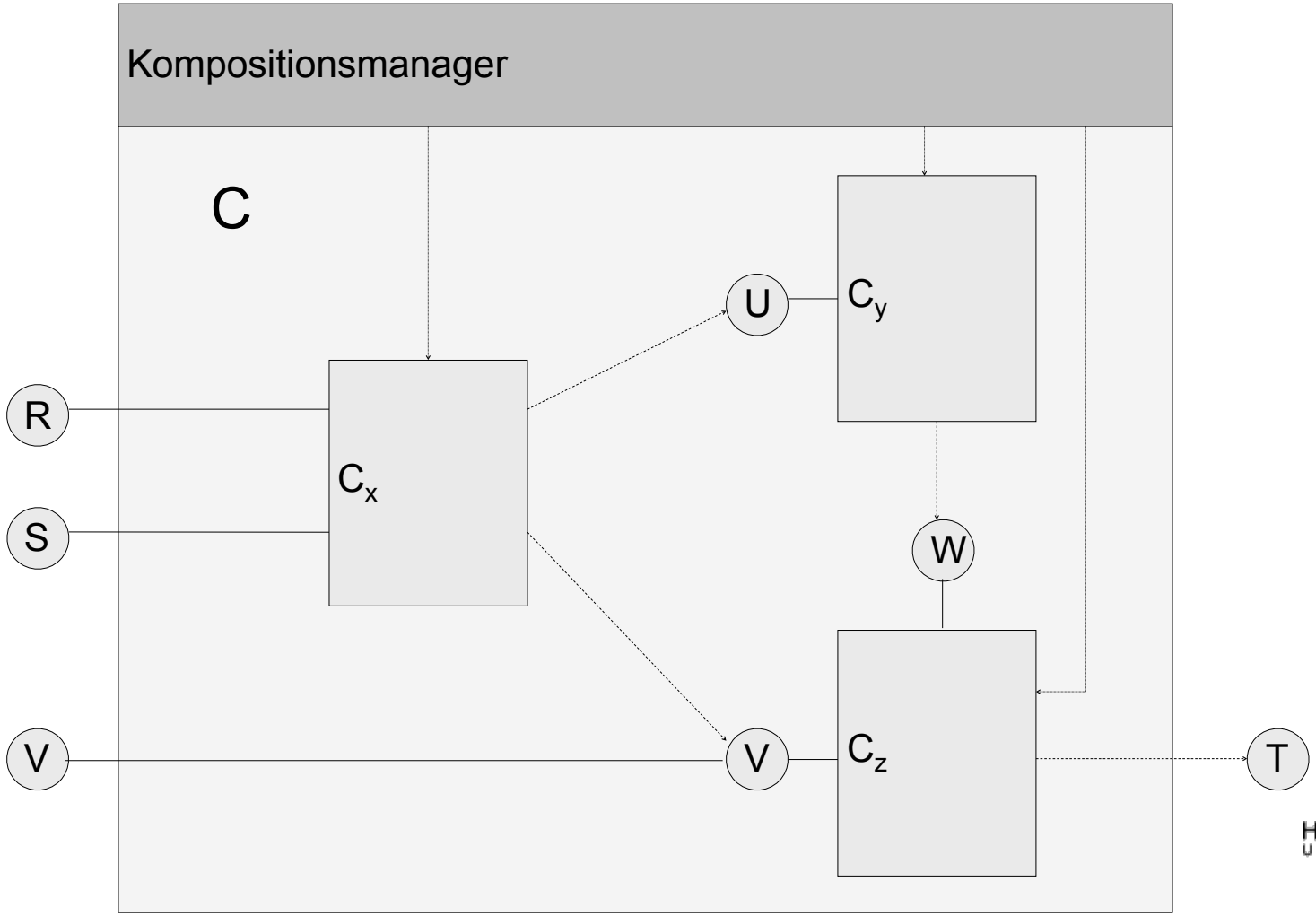
- **(R6)** Der Beziehungsgraph von Komponenten muss zyklenfrei sein.
- **(R8)** Transitive Abhängigkeiten zwischen Schnittstellen sollten minimal sein (hohe Kopplung).
- → Es gibt diverse technische Möglichkeiten die Regeln zur Compilezeit (Maven) und/oder zur Laufzeit sicherzustellen (OSGi)

Die klassische Schichtenarchitektur als Komponentenarchitektur



Hochschule Rosenheim
University of Applied Sciences

Komponenten unterstützten Komposition



Hochschule Rosenheim
University of Applied Sciences

Wie baut man Komponenten?

- Trenne Schnittstelle und Implementierung
  - Getrennte Pakete für Schnittstelle und Implementierung
- Drei Varianten
  - Ein JAR pro Anwendung
    - Nachteil: Alles ist sichtbar – Regeln sind nur per Konvention abgebildet (oder durch ein Codeanalysewerkzeug (z.B. S101)
  - Ein JAR pro Komponente (Schnittstelle und Implementierung)
    - Abhängigkeiten zwischen Komponenten können per Maven abgebildet werden (vermeidet zyklische Abhängigkeiten)
    - Illegale Abhängigkeiten zwischen Schnittstelle und Implementierung werden mit Maven nicht erkannt
    - OSGi löst das Problem durch public/private packages
  - Getrennte JARs für Schnittstelle und Implementierung
    - Abhängigkeiten lassen sich mit Maven vollständig modellieren
    - Kann sehr aufwendig werden!

Hochschule Rosenheim
University of Applied Sciences

Java 9 JIGSAW

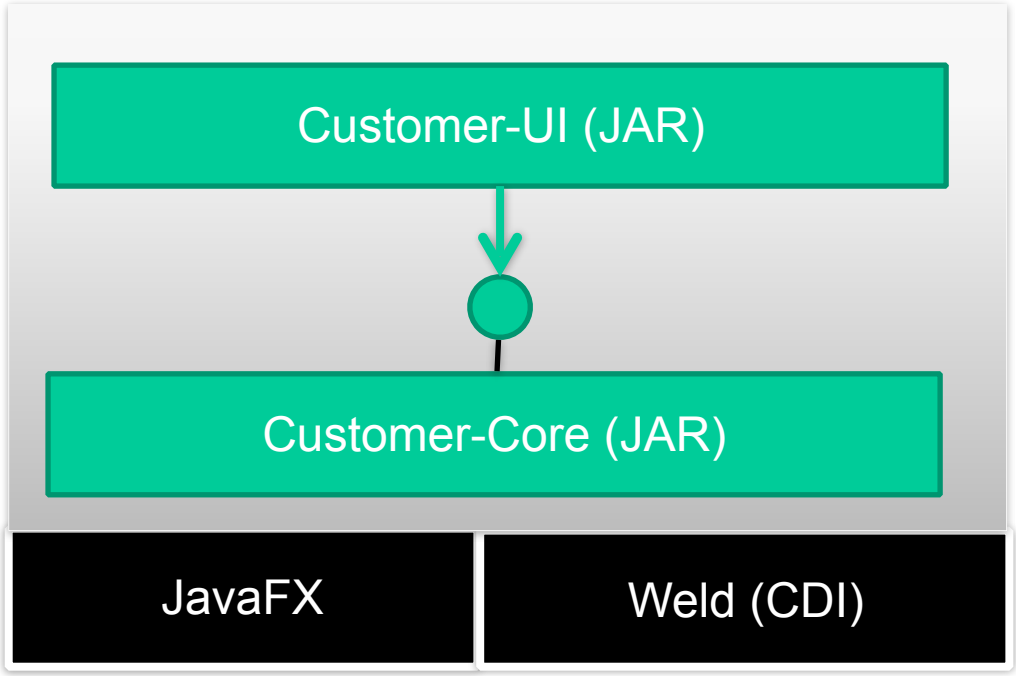
- Java 9 unterstützt den Bau von Komponenten
  - Eine Komponente ist ein normales JAR-File mit einem speziellen Moduldeskriptor (module-info.java / module-info.class)
  - Der Moduldeskriptor definiert exportierte Schnittstellen
  - Der Moduldeskriptor definiert importierte Schnittstellen
  - Der Java Compiler stellt sicher, dass:
    - Nutzer nur die Schnittstelle sehen, nicht die Implementierung
    - Es keine zyklischen Abhängigkeiten zwischen Moduln gibt
    - Externe Klassen die in der Implementierung verwendet werden nicht nach aussen sichtbar sein (Transitive Dependencies)

    - Siehe http://qaware.blogspot.de/2015/10/java-9-jigsaw-long-awaited-java-module.html

Hochschule Rosenheim
University of Applied Sciences

Beispiel: Eine Anwendung zur Kundensuche

Kundensuche



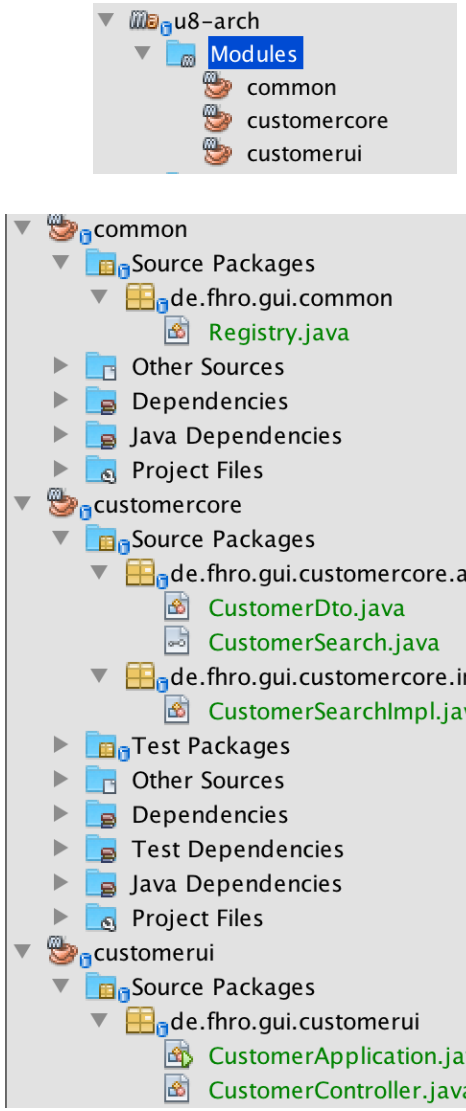
Die Komponente CustomerUI

- Separates Maven Modul
- Verwendet die API der CustomerCore Komponente
  - interface Search
    - List<CustomerDto> searchForCustomers(String expression)
  - class CustomerDto
    - POJO für Kundendaten (Name, Vorname)

Die Komponente CustomerCore

- Enthält zwei Implementierung
  - SearchMock – Eine einfache Mock-Implementierung
  - SearchImpl – Die Kundensuche (REST, DB, …)
- Verwendet Weld als Factory
  - Die Klasse Registry kapselt den Weld-Zugriff
  - Hierzu gibt es viele Alternativen
    - Spring, CDI, OSGi, Google Guice, Afterburner.FX …
    - Am besten kapselt man das in einer separaten Klassen (Registry)

Hochschule Rosenheim
University of Applied Sciences



Klammerprojekt (keine Quellen)

Factory

Komponente *CustomerCore*

Schnittstelle

Implementierung

Komponente *CustomerUI*

CustomerCore Nutzung