jsf-academy.com

www.jsf-academy.com

UI-Architekturen mit JSF

- JSF ist mehr als nur Syntax -

Agenda

- Warum reden wir überhaupt über UI-Architektur?
- Technologien und Architekturen – wie passt das zusammen?
- Design Prinzipien und Design Patterns
- Verschiedene UI-Architektur-Ansätze (mit JSF)
- Fazit

Unsere Wunschvorstellung

- Realisierung des UI-Layers auf einer tragfähige Gesamtarchitektur
- Das UI-Layer soll nahtlos in die Gesamtarchitektur passen
- Aussagen wie Wiederverwendbarkeit, Erweiterbarkeit und Testbarkeit gelten auch für das UI

Das Problem der weißen Seite

- Wie baue ich eine gute (UI-) Architektur?
- Wo fange ich an?

→ Frühzeitig darüber nachdenken, nicht erst dann, wenn es zu spät ist (wie im rechten Bild)

Software Design Prinzipien als Grundlage

- Separation of Concerns
  - Jede Klasse hat ihre spezifische Zuständigkeit und ist damit klar von anderen Klassen abgrenzbar
- DRY / DIE
  - Keine Code-Duplizierung
- Lose Kopplung
  - Verschiedene Komponenten sollten so gestaltet sein, dass Änderungen einfach möglich sind und keine harte Verdrahtung zu anderen Komponenten vorhanden ist

Design Patterns – die nächste Stufe

- MVC - Model View Controller
  MVP - Model View Presenter
- Sandbox-Pattern
- Facade
- Blackboard-Pattern

UI und Architektur?

- „UI, das sind doch nur bunte Pixel !“
- Im UI wird doch nichts programmiert, sondern nur gemalt. Also brauche ich keine Architektur.
- Hey ihr Frontend‘ler, lasst die Architektur die richtigen Entwickler machen. Bleibt ihr lieber bei den JSP-Seiten.

UI und Architektur? Sehr wohl!

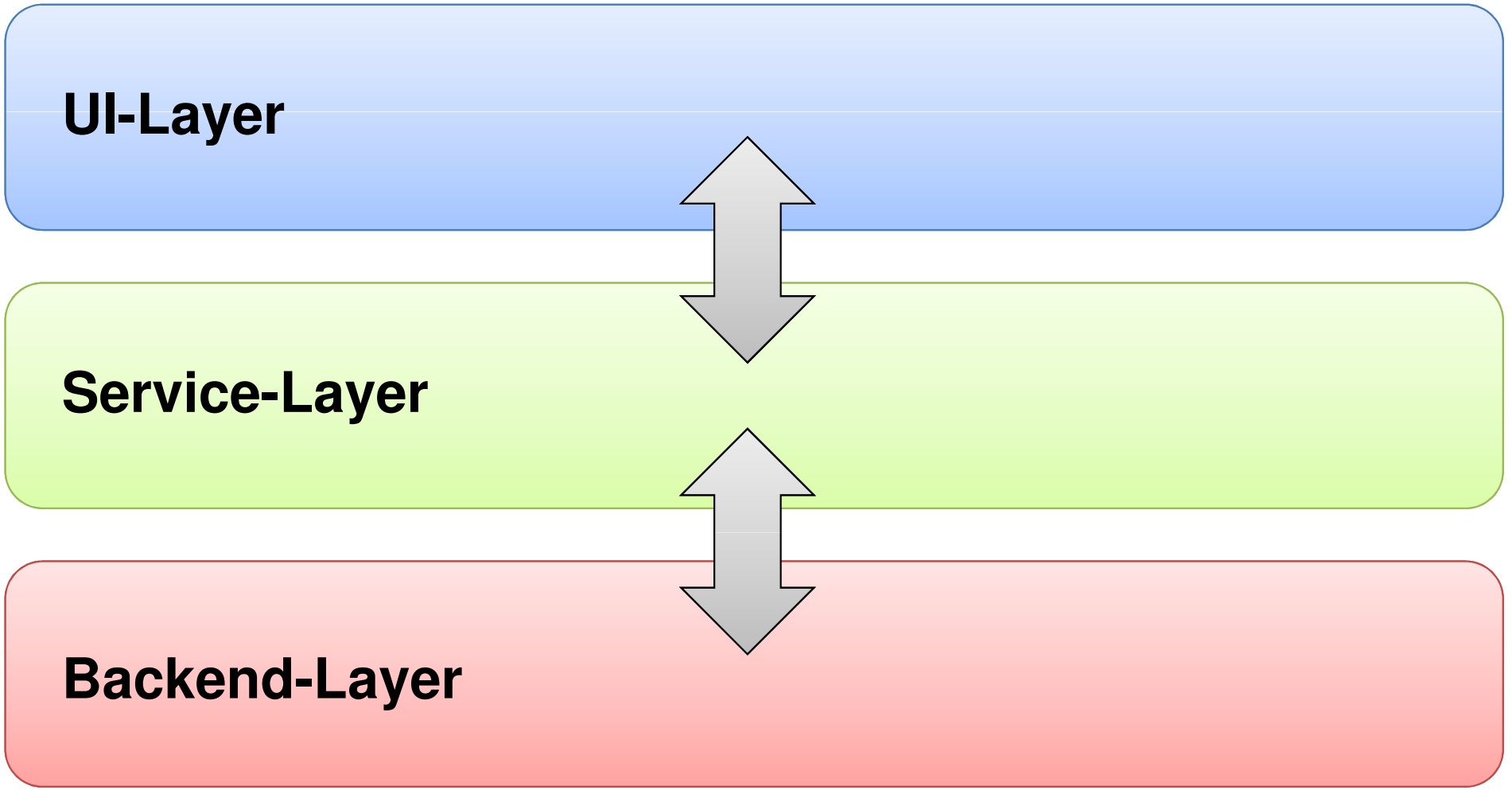
- Das UI ist eine eigene Schicht in einem Gesamt-Schichtenmodell, folglich Bestandteil der Gesamt-Architektur

- Im UI-Layer passiert mehr, als manch einer denkt:
  - Bean-Handling
  - Page-Flow
  - Konvertierung und Validierung
  - Visualisierung von Fehlern
  - Aufruf des Service-Layers
  - Rückantwort des Service-Layers

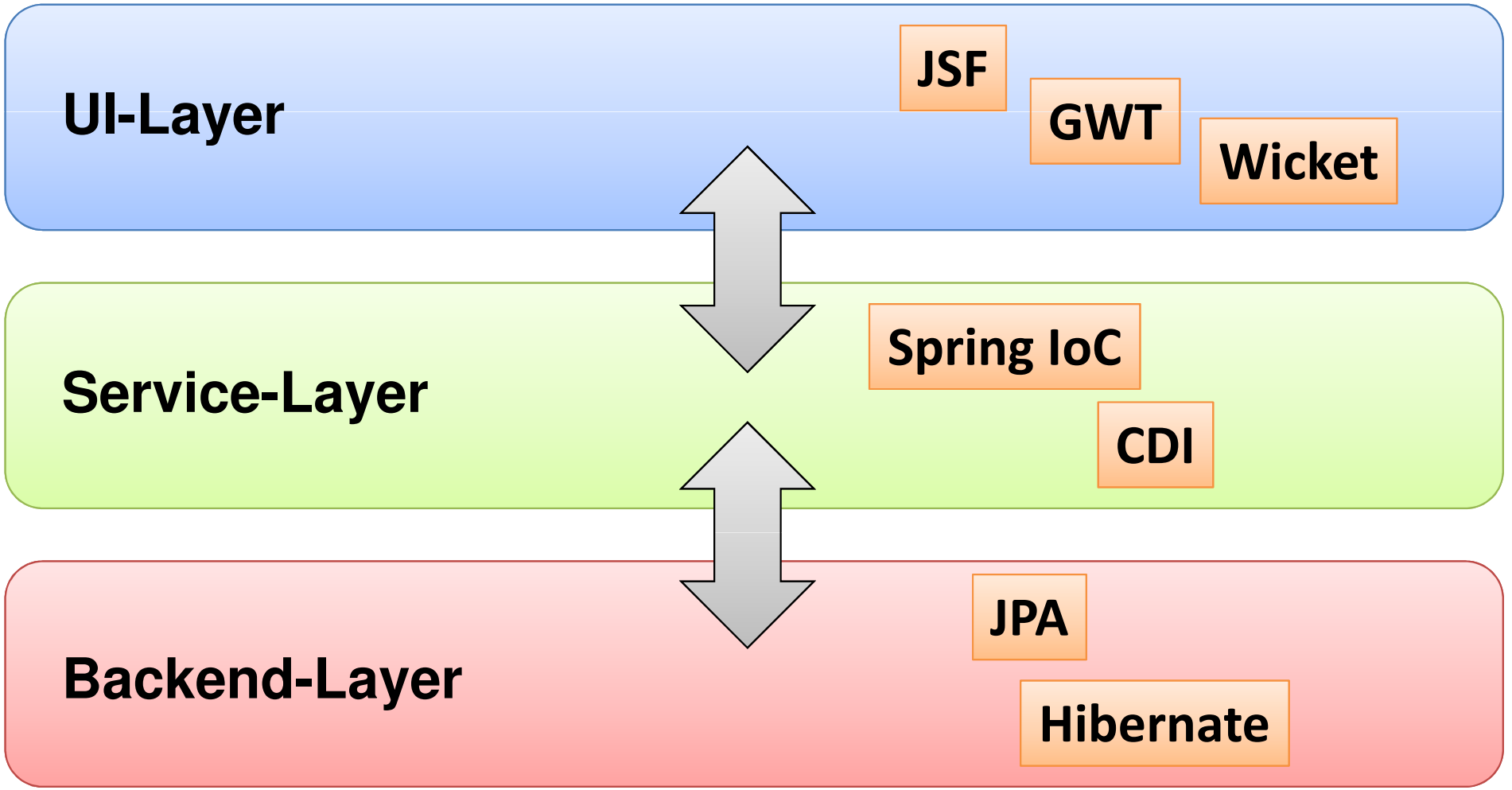
Architektur und Technologien

- Technologien sind keine Architektur!
- Allerdings helfen Technologien, eine Architektur zu realisieren.
- Spezielle Technologien lassen sich eindeutig einem Bestandteil einer Architektur zuordnen.

Gesamt-Architektur



Gesamt-Architektur mit Technologien



Aufgaben des UI-Layers

- Darstellung der View (natürlich ;-))
- Datenhaltung für die Sitzung / Conversation
- Verarbeitung von Request-Parametern, dazu Konvertierung und ggf. Validierung
- Aufruf des Service-Layers
- Verarbeitung der Rückgabe des Service-Layers

Patterns im UI-Layer

- MVC
  - Model-View-Controller Pattern
  - Realisiert das Separation-of-Concerns Prinzip
  - Spezialisierte Artefakte für das Modell, die View und den Controller

- MVP
  - Model View Presenter Pattern
  - Basis ist MVC, nur dass M und V komplett getrennt sind
  - Der Presentor regelt sämtliche Verarbeitungslogik (Modell steuern, View befüllen, …)

JSF und MVC

- JSF kennt lediglich Managed Beans
- Die Spezifikation sagt nichts über Model oder Controller

→ Dann eben selbst gemacht: Model Managed Beans und Controller Managed Beans

→ MVC lässt sich mit JSF hervorragend umsetzen

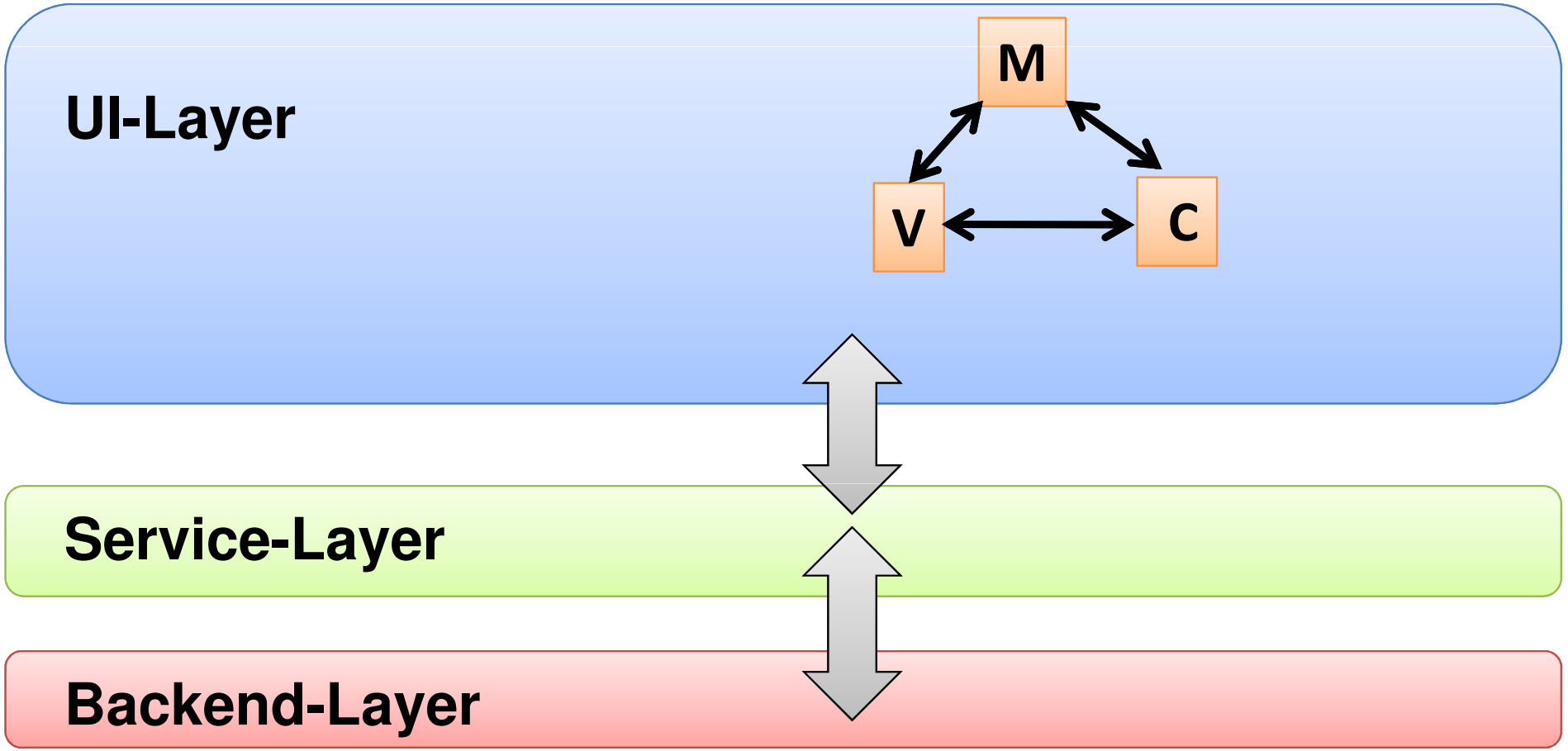
Funktionalität im Modell?

Warum sollte ich überhaupt zwischen Modell und Controller unterscheiden?

- Separation of Concerns
- Übersichtlichkeit, Wartbarkeit

- **Aber auch: Zentrales Exception Handling (Sandboxing) möglich**

Empfehlung: MVC in JSF



Exception Handling in JSF

- JSF 1.x: Kein zentrales Ansatzpunkt möglich.

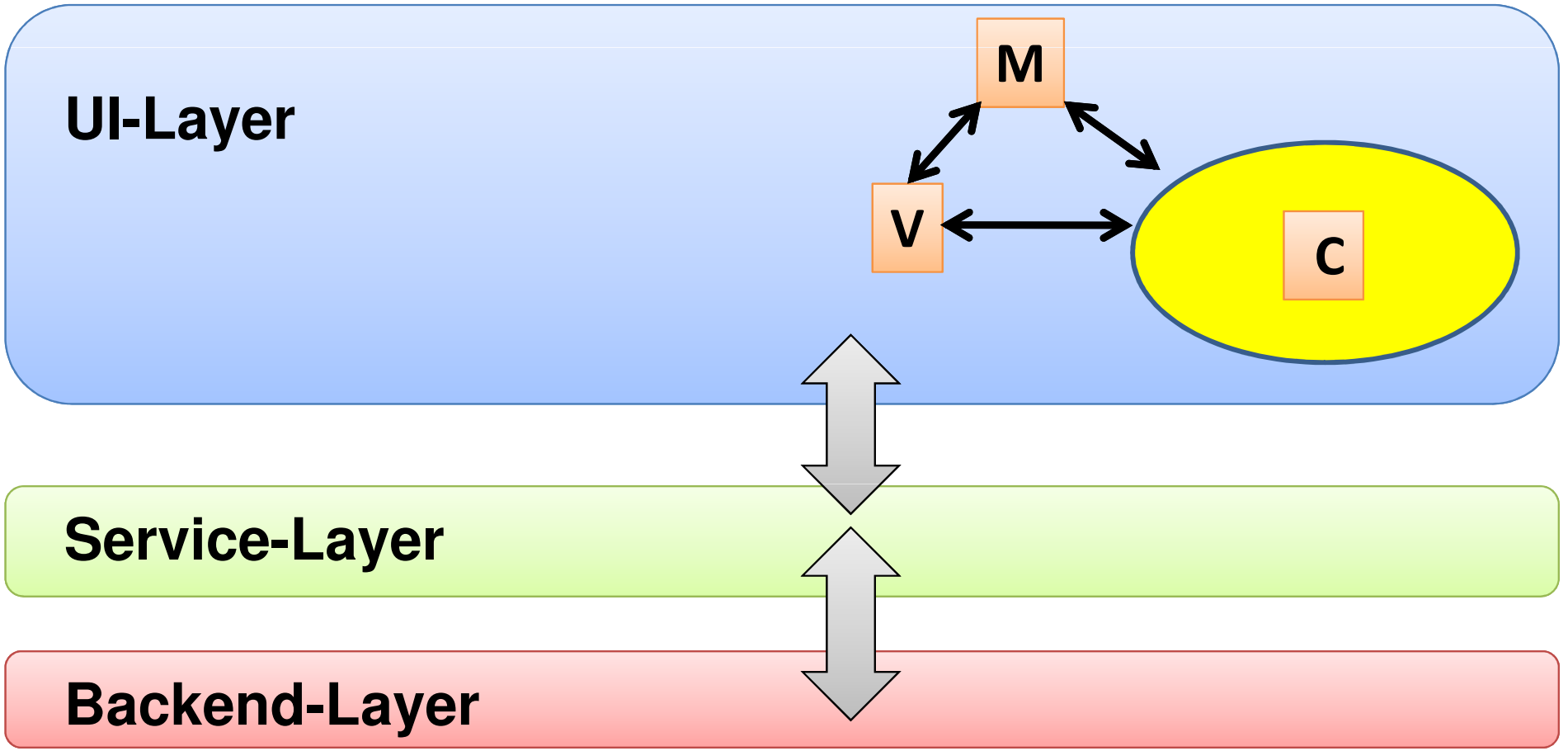
→ Try / catch in Phase 5 (Invoke Application)

→ **Sandboxing** Verfahren

→ **Never ever** Funktionalität (Stichwort Lazy Loading) im Modell; wer soll hier eine Exception abfangen?

- JSF 2.x: Zentraler **Exception Handler Mechanismus**

Empfehlung: Sandboxing bei „C“



Aufruf des Service-Layers

- Fachlichkeit gehört nicht (Betonung auf „nicht“) in den UI-Layer, sondern in den Service-Layer

- Aufruf aus dem UI-Layer mittels
  - Einfacher Methoden-Call
  - Webservice-Call (SOAP)
  - REST
  - Remoting (RMI, HttpInvoker, …)

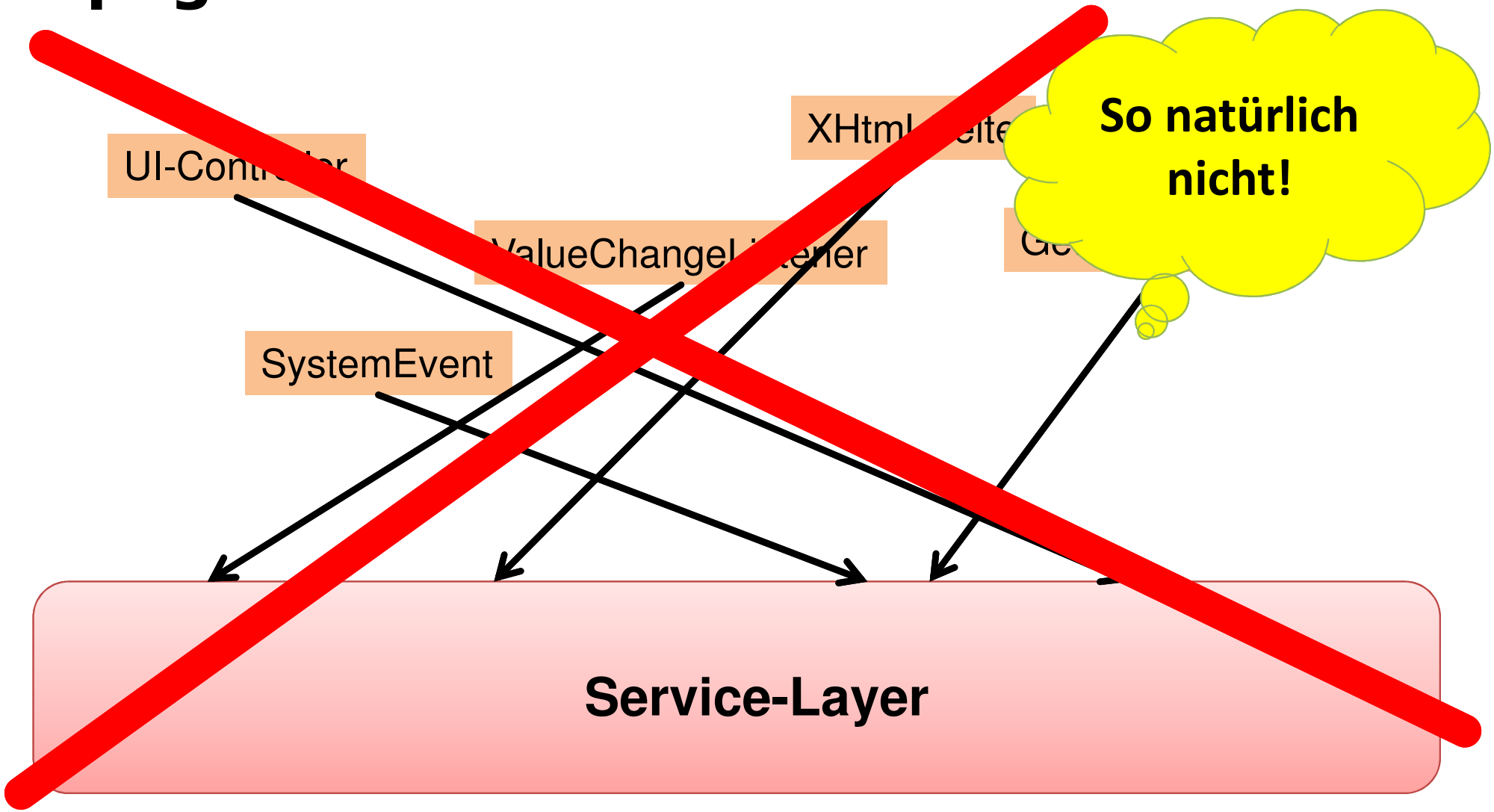
- Rückgabe
  - Exceptions? Checked? Unchecked?
  - Return-Objekt
  - Nur ein reines Datenobjekt?

Über JSF zum Service-Layer

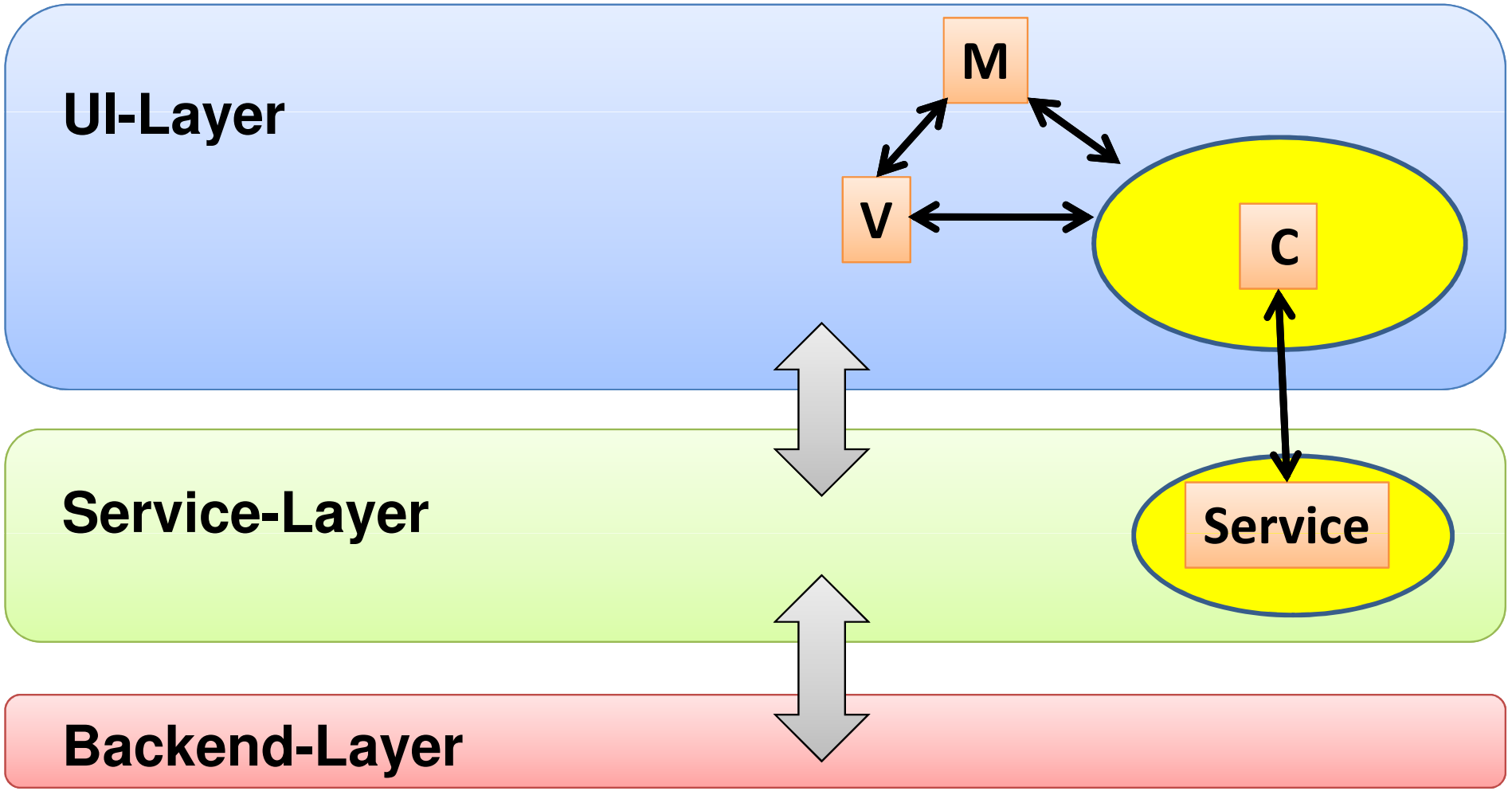
- Aufruf nur in Phase 5 (Invoke Application)
- Oder doch nicht? Geht doch auch lazy in Getter-Methoden in Beans (bäh).
- Zugriff in ValueChangelistener? Angetriggert durch SystemEvents?

→ **Empfehlung: Klare Übergänge regeln, kein wilder Durchgriff aus dem UI-Layer heraus.**

Spaghetti von UI nach Service



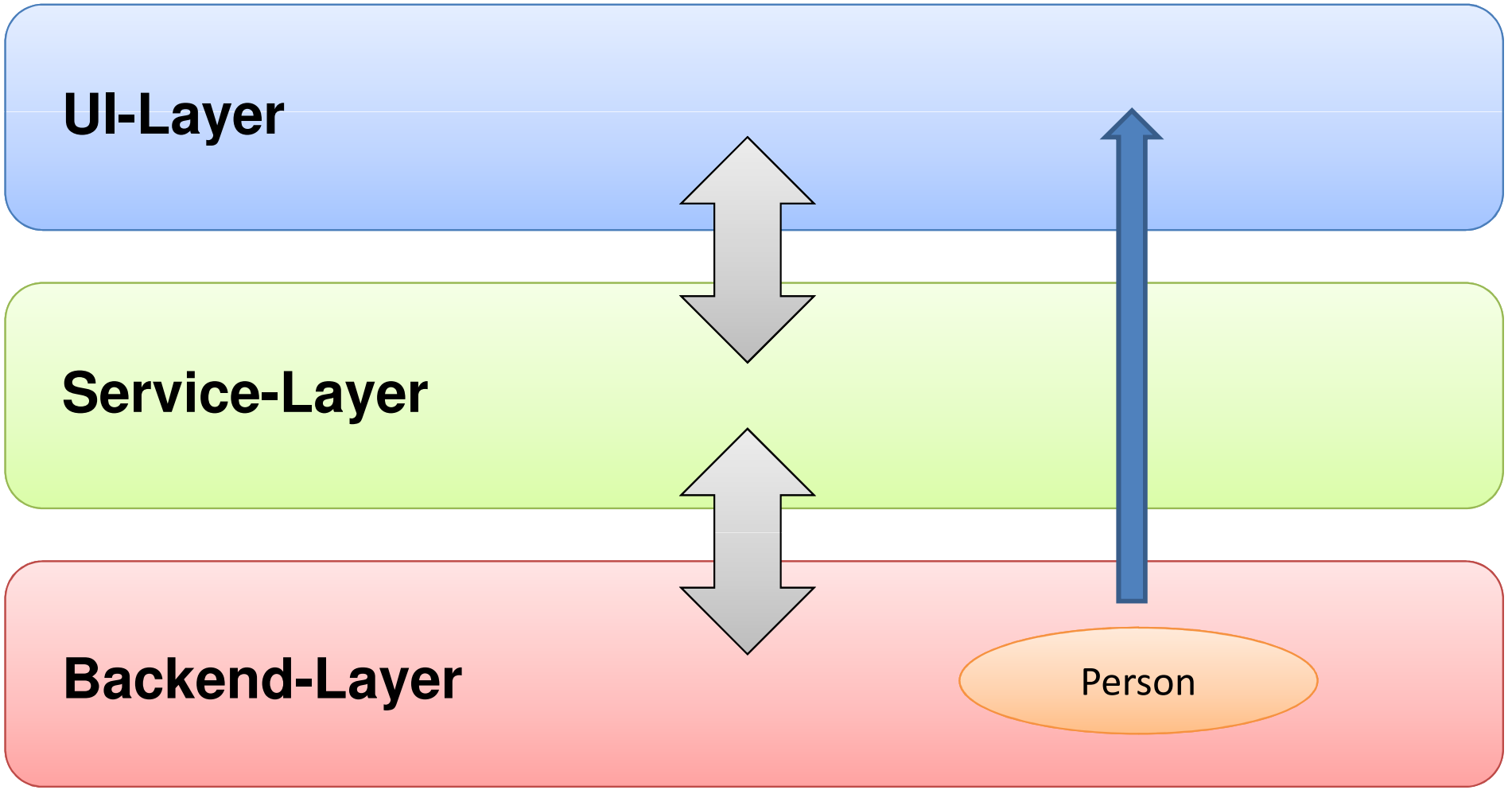
Empfehlung: Service-Aufruf im „C“



Anderes Thema: Das Modell

- Ist das M in MVC ein reines View-Modell?
- Nehme ich die Datenbank-Entity und reiche sie nach oben?
- DTO‘s?
- Detached oder attached Objekte ins UI?

Wo liegt das Modell?



Arbeiten auf detached Objekten

- Per JPA laden
- In Invoke Application laden (in DB-Layer) und detachen
- Von DB nach JSF hochliefern

→ Entitäten können Relationen aufweise. Sollen diese gleich mitgeladen werden?
→ Komplett tief laden?
→ Oder dann einfach „null“ zurückliefern?

Arbeiten auf attached Objekten

- Servlet Filter

  Öffnen der Datenbank-Session in einem Filter,
  am besten VOR dem JSF-Request und DANACH

→ OpenSessionInView-Pattern

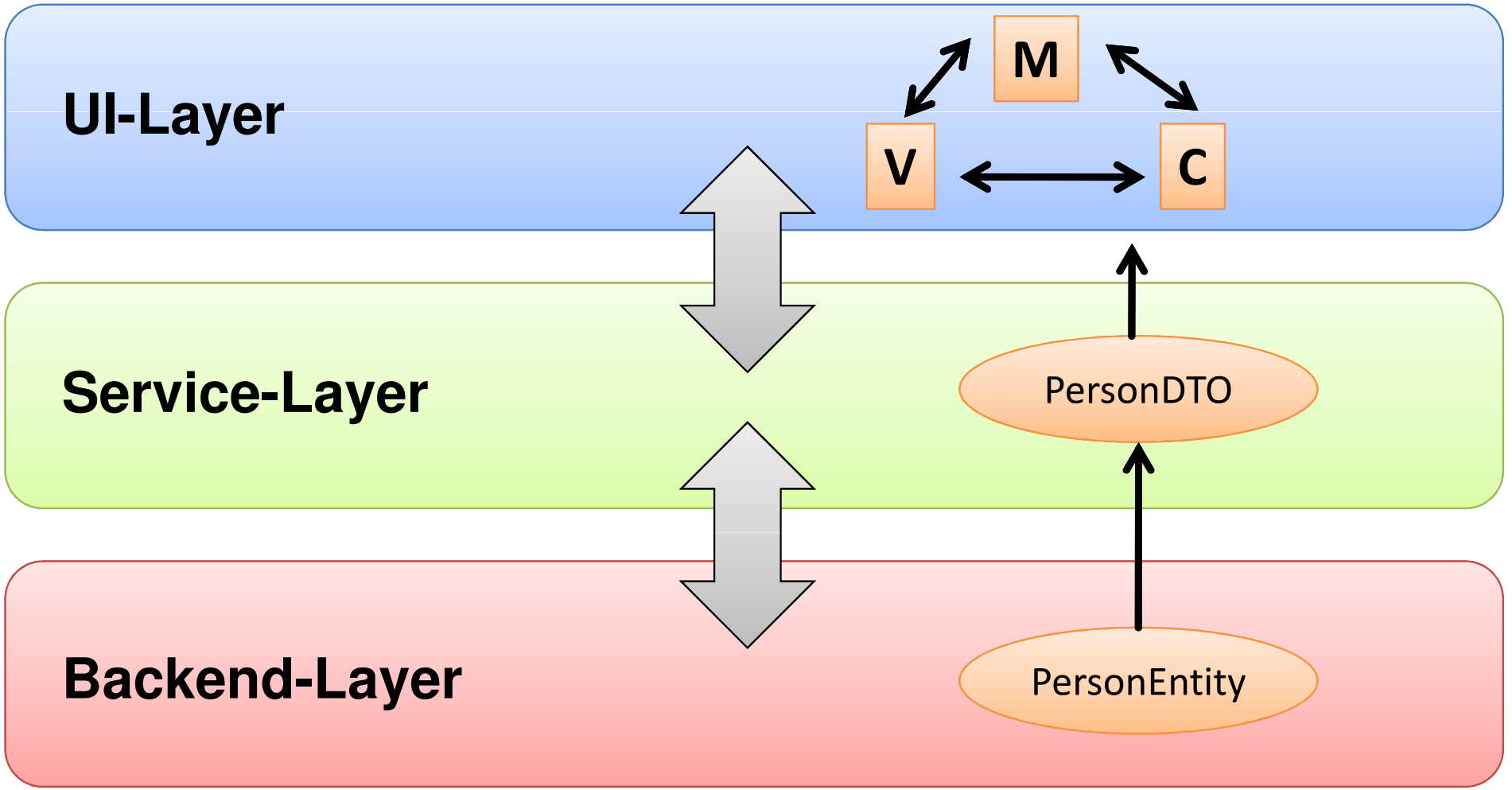
→ **Hat irgendjemand noch eine Kontrolle über die SQLs?**
→ **Schichtentrennung?**

Klassisch: DTOs

- Eine tiefere Schicht lädt alle notwendigen Daten
- Die Daten werden in eine gesonderte Struktur gepackt
- Die Struktur (das DTO) wird nach oben gereicht
- Martin Fowler spricht z.T. von Domain Objects und Presentation Objects

Konsequenzen
- Super verständlich :-)
- Viel (unnützes?) mappen :-(

Ein mögliches Ergebnis



Fazit

- UI-Architektur ist ein wichtiges Thema.
- Und wir wissen jetzt: Es gibt überhaupt eine UI-Architektur, nicht nur bunte Bilder im UI-Layer
- Es gibt verschiedene Ansätze für eine UI-Architektur, die jeweils ihre Vor- und Nachteile haben. Eine genaue Konzeption im Vorfeld ist daher unbedingt ratsam.

Fragen?

Gerne E-Mail an mich:
andy.bosch@jsf-academy.com

Twitter
@andybosch

Oder einfach mal ein paar JSF- und CDI-Tutorials austesten auf:
www.jsf-academy.com