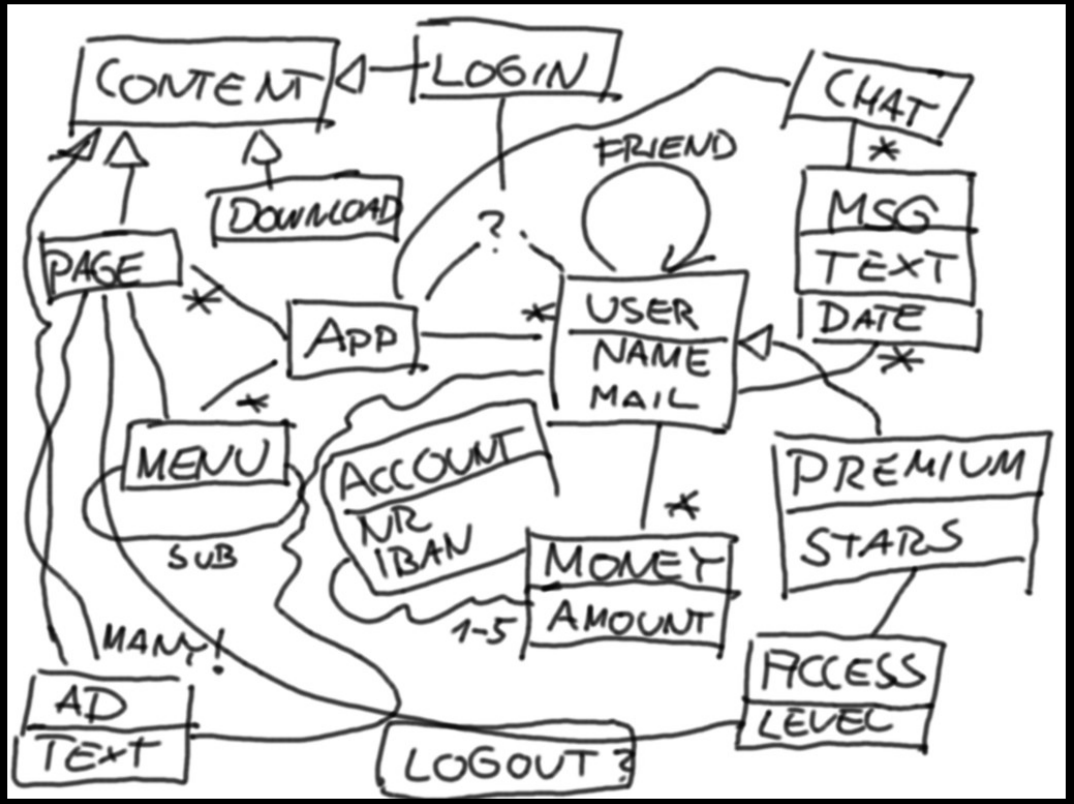
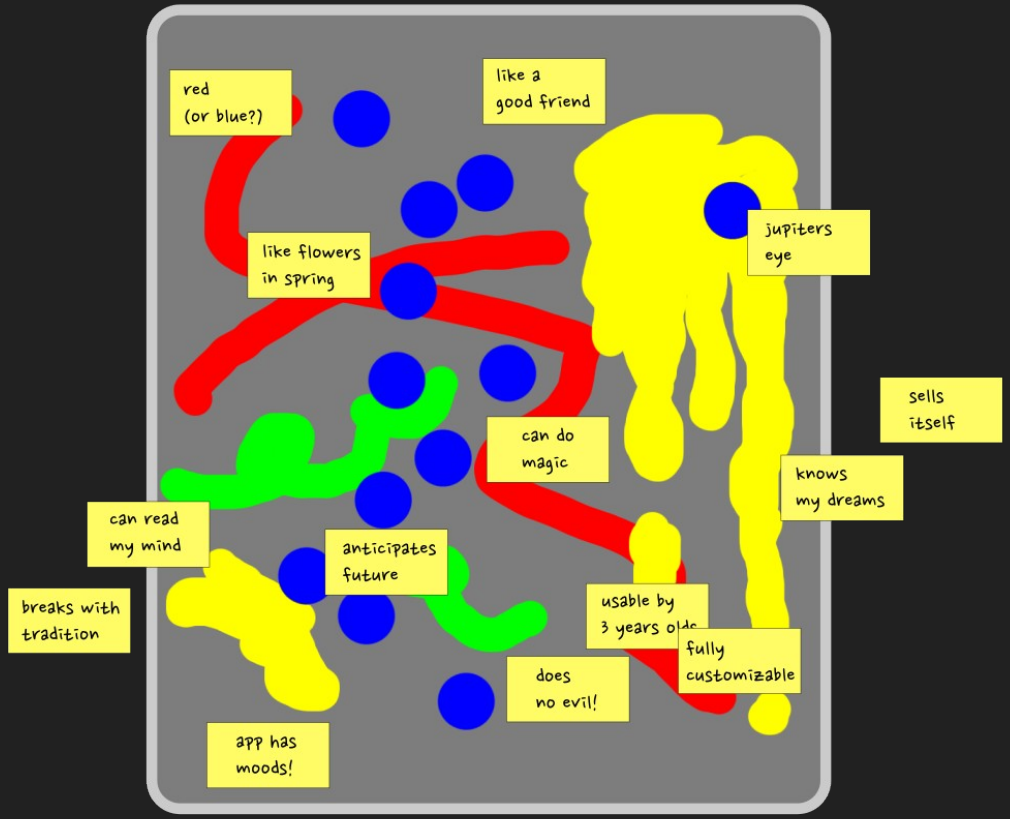
Das neue Frontend

Automation



Tom Technik

Art



Carl Cünstler

Modern Frontends - Art, Architecture and Automation

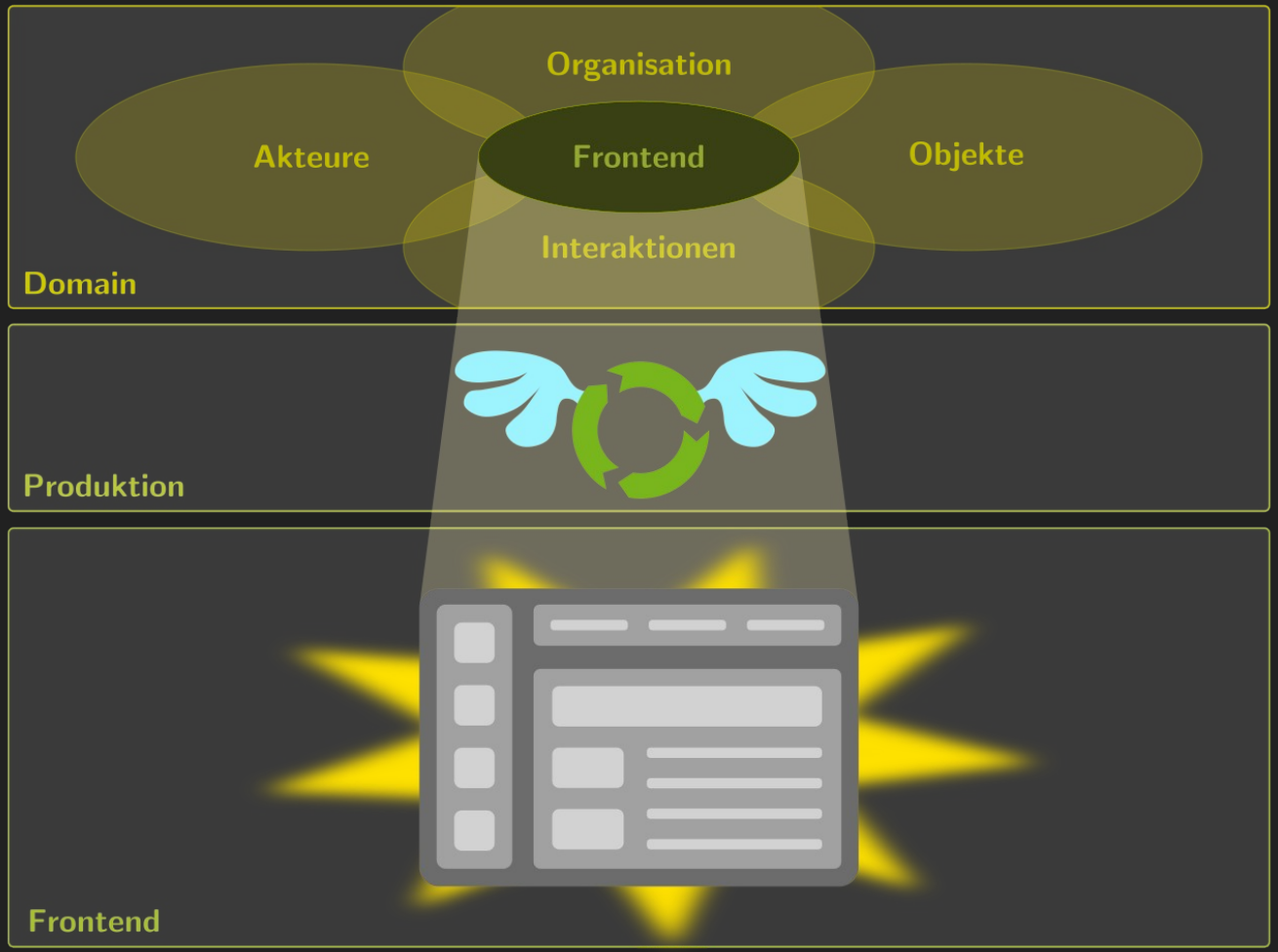
https://modernfrontend.design

Dr. Arne Bosien
Colamda GmbH

Andreas Leue
Sphenon GmbH

Einleitung

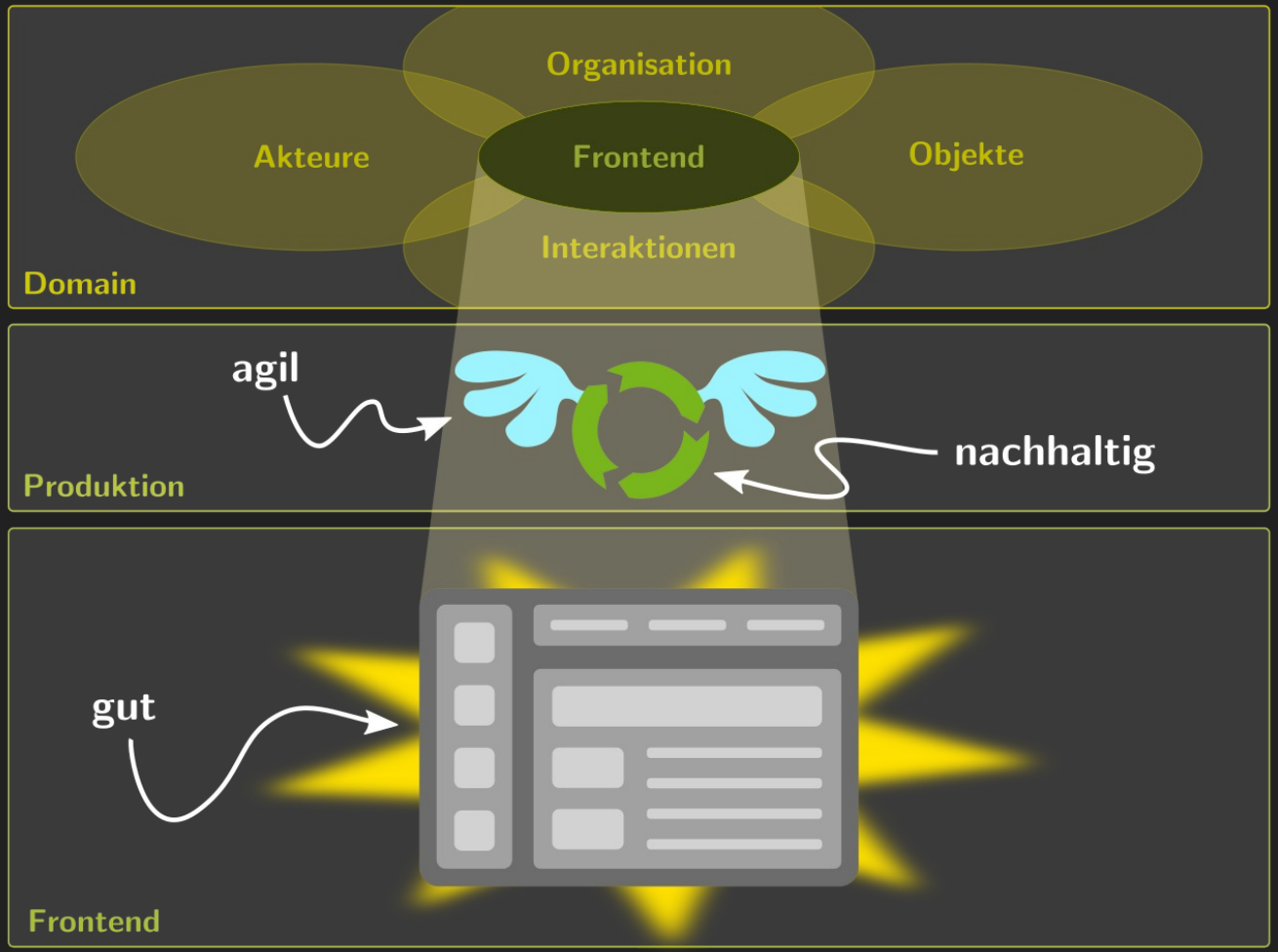
Was ist das (augenscheinliche) Problem?



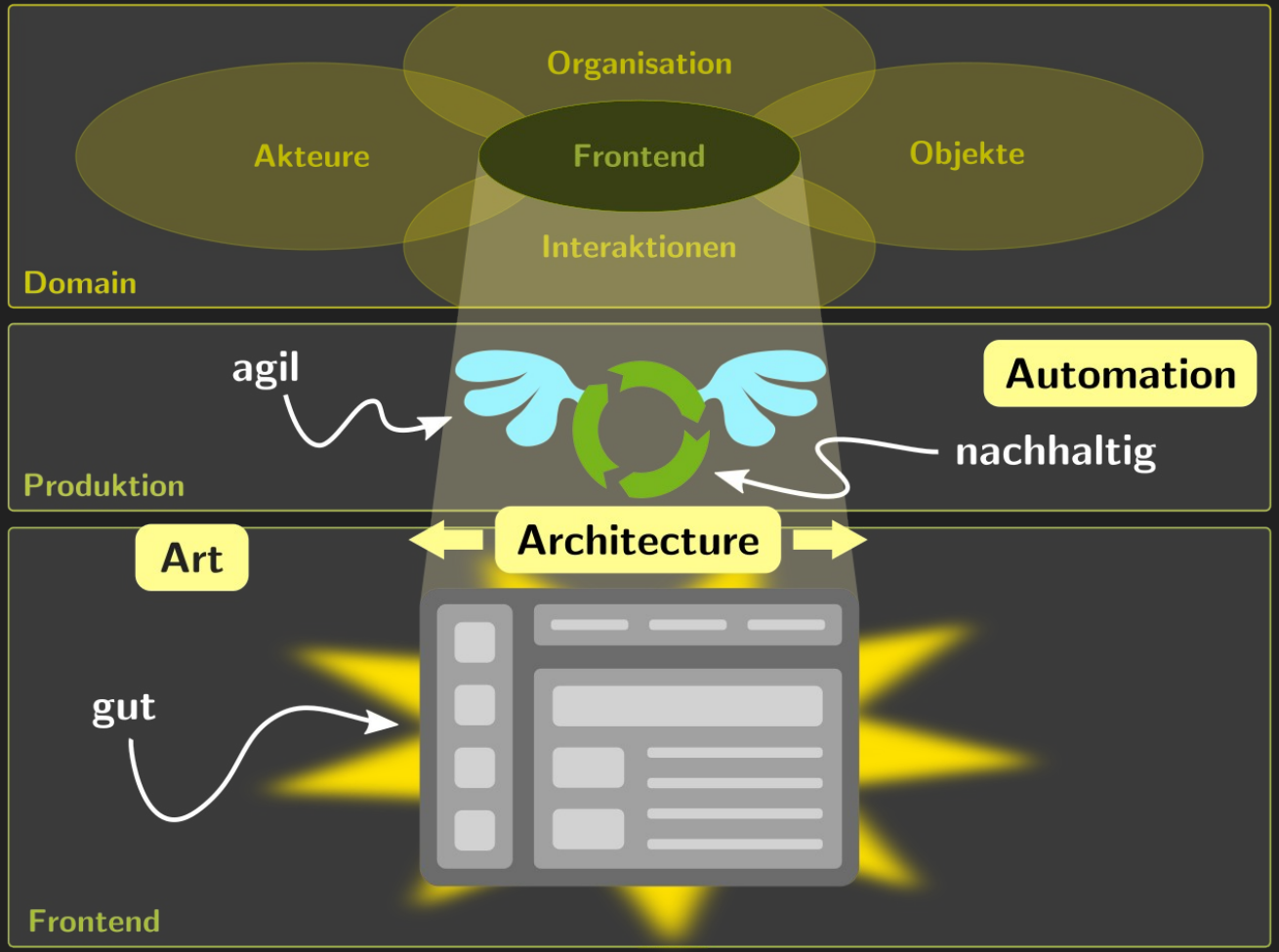
Herausforderung UI Entwicklung

- gute UIs sind herausfordernd zu erzeugen und weiterzuentwickeln

Herausforderung UI Entwicklung



- gute UIs sind herausfordernd zu erzeugen und weiterzuentwickeln
- **nachhaltige** und **agile** Produktion **guter** Ergebnisse passen nicht richtig unter einen Hut



Art, Architecture, Automation

- Frontend-Entwicklung ist eine Kunst
- kreatives Kunstwerk
- Automation als maximal effiziente Produktion
- Noch besser: Fertigteile
- Architektur als Gegensatz zu Kunst
- Gute Architektur als harmonisierendes Element

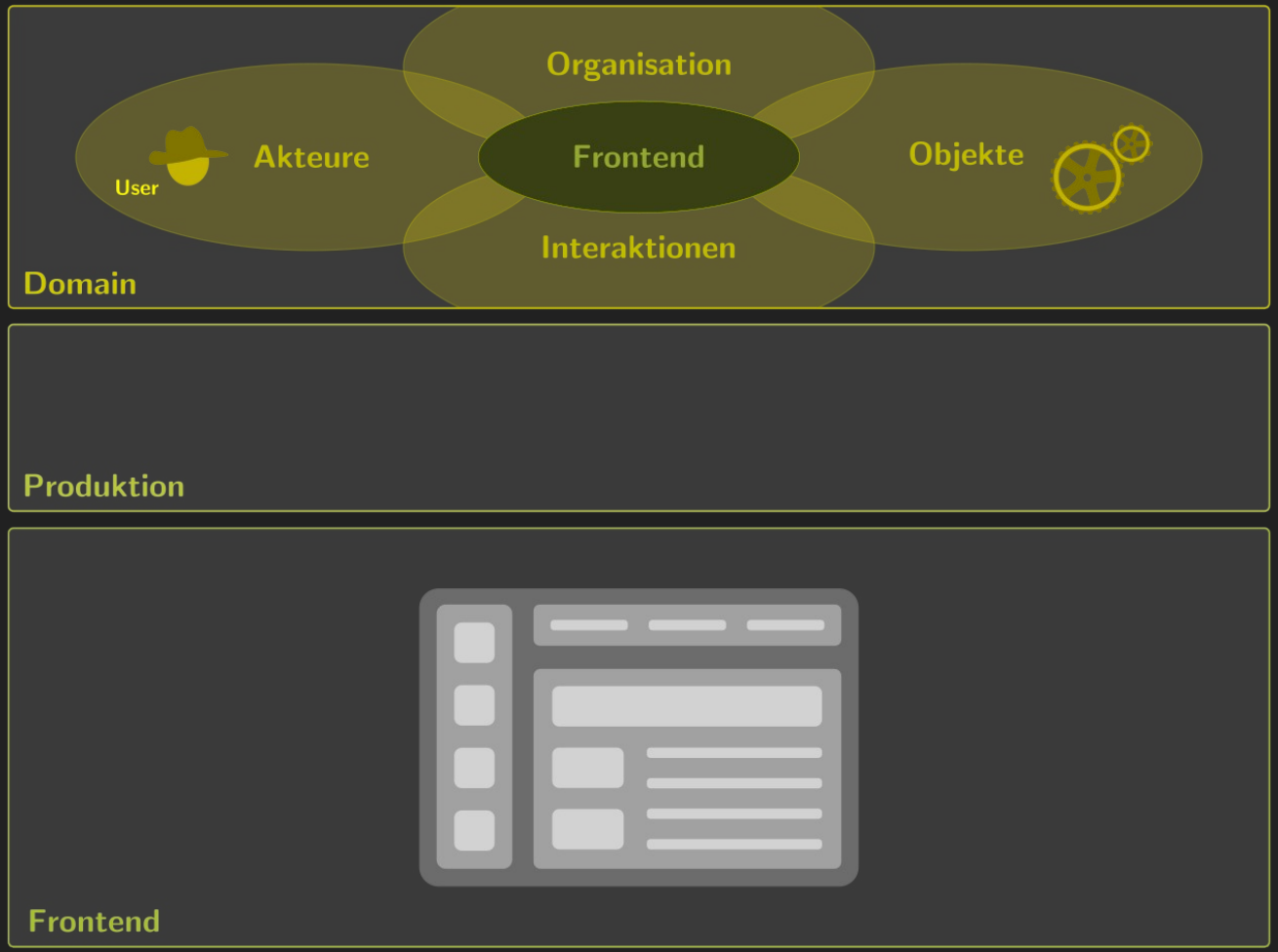
Agenda

- Wo sind Reibungszonen?
- Was steckt dahinter?
- Art vs. Automation

- Frameworks
- Architektur
- Sustainable Components

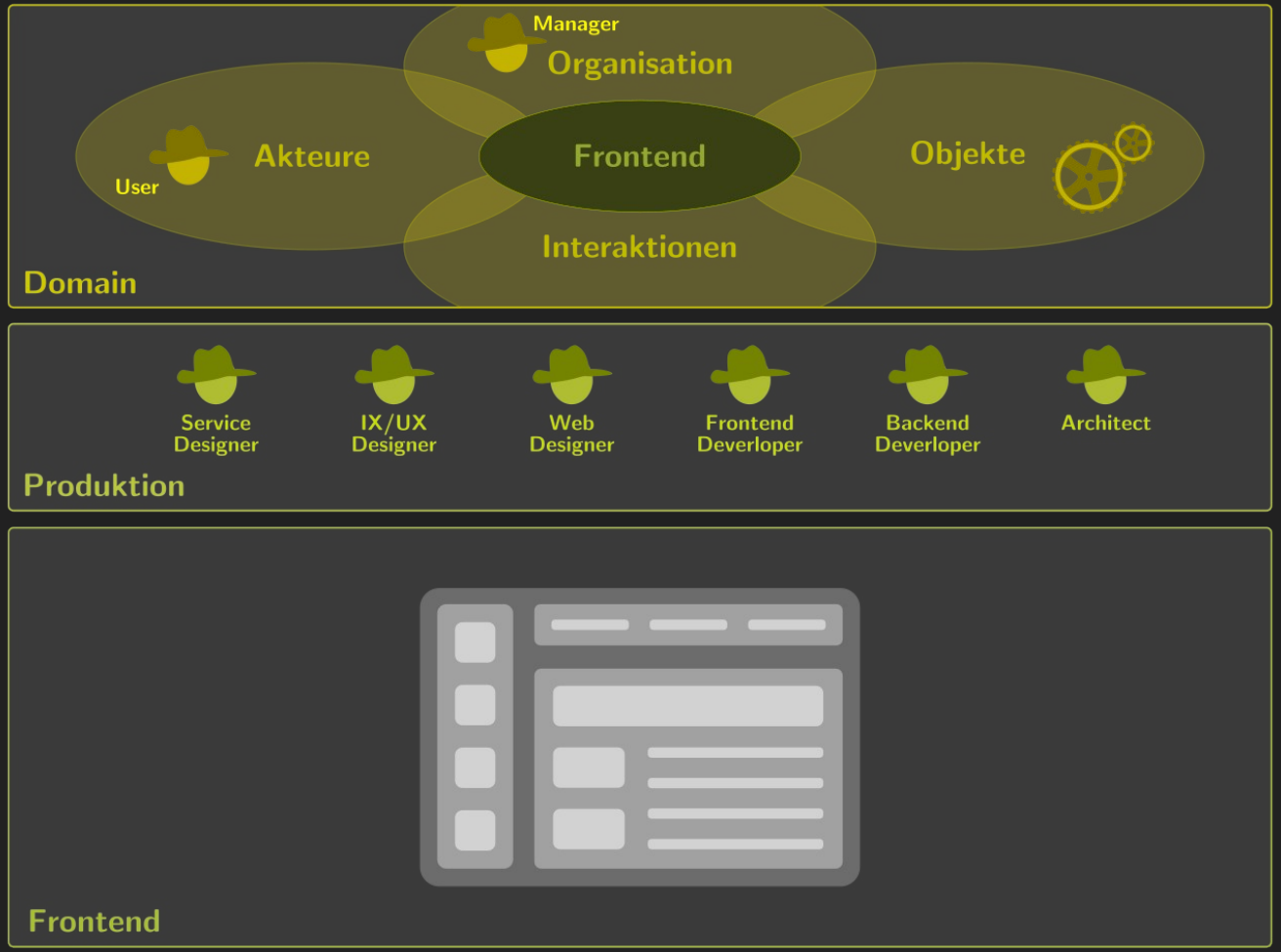
- Zusammenfassung

Was macht die (Zusammen-) Arbeit schwierig?



Zielgruppe: Mensch trifft auf Maschine

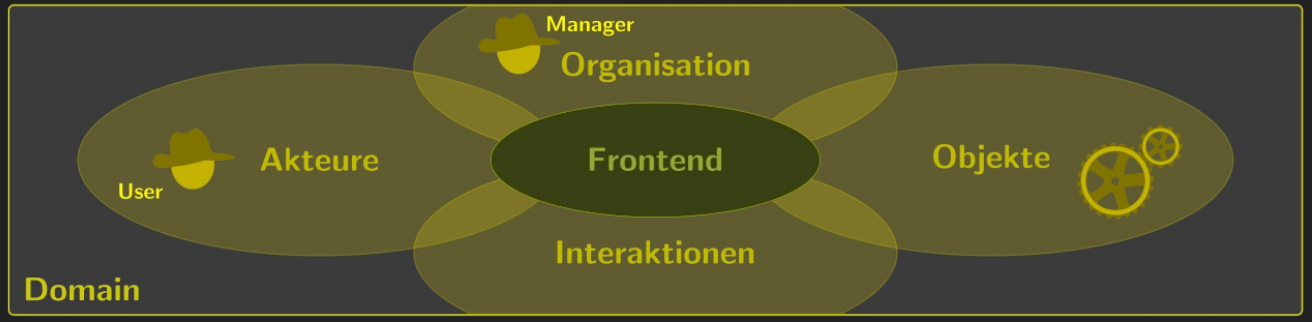
- Das Medium “UI” soll Interaktionen zwischen Mensch und Maschine vermitteln
- Zwei sehr unterschiedliche Domänen!

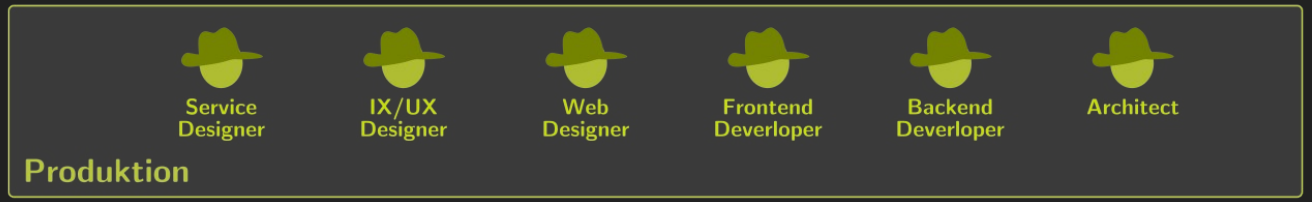


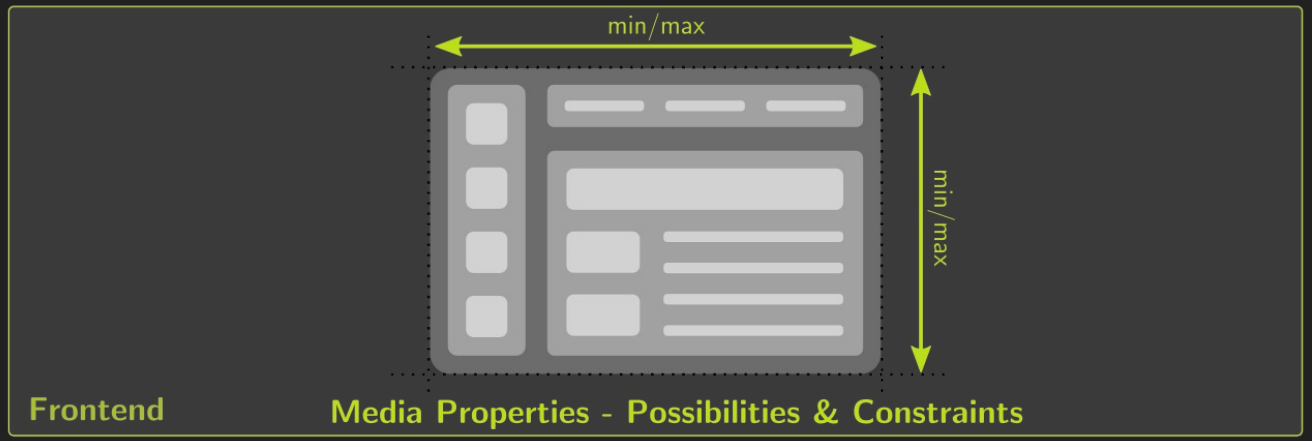
Produktion: Interessen, Kultur und Kooperation

Viele verschiedene Stakeholder sollen zusammen das Artefakt “UI” produzieren:

- CIO
- Sponsor/PoC dev
- Product Owner
- Designer
- Design-Ops
- (Design/UX) frontend dev
- Architekt
- Endnutzer







Komplexes, sich entwickelndes **Medium**

UI Technologie

- fantastische Möglichkeiten
- auch Einschränkungen
- will beherrscht sein
- ständigem Wandel unterworfen

Warum ist das relevant?

Ubiquitous Frontends

Frontends sind allgegenwärtig

- Device Evolution: embedded/IoT, Displays überall
- Heterogenität, agile Business, experimenting, fast
- Hohe Nutzerzahlen

Anforderungen

Kunden wollen zunehmend bessere UIs

- Hohe Lade-Geschwindigkeit
- Autonomie / Offline-Fähigkeit
- Responsive UI
- i18n/l10n
- a11y
- Ergonomie
- Ästhetik
- geringe Lernkurve
- Aktualität der App
- bookmarkable URLs

- Daten-Konsistenz im Frontend
- Daten-Konsistenz zwischen Usern
- Validierung
- Großen Datenmengen
- Undo/Redo/Abort/Submit
- CRUD-Operationen
- Multi-Entity-Operationen
- Paginierte Tabellen und Einzel-Entitäten
- Sortieren und Filtern von Tabellen
- Optimistic locking
- Idempotenz

Produktion

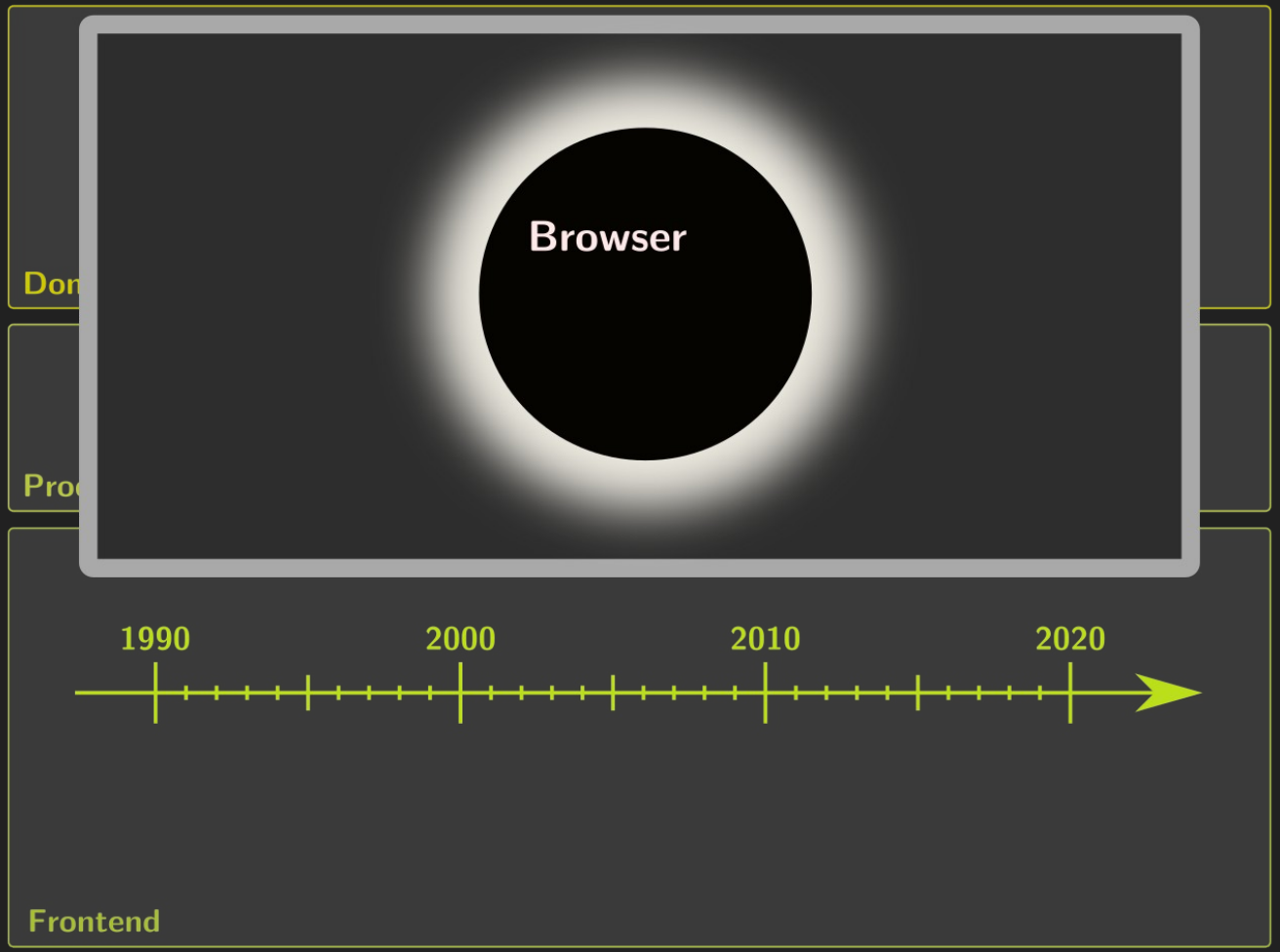
Unternehmen müssen zunehmend schnell sein

- Prozess
  - hohe Taktung
  - Agiles, iteratives Vorgehen
  - DevOps, CX, Kontinuierliche Verbesserung
- schnelle Reaktion auf Anforderungen
  - von seiten Interaktion
  - von seiten Domänen/Business-Logik/-Struktur
- unabhängige Teams
- Investitionssicherheit / Nachhaltigkeit
  - Übergabe/Übernahme-Fähigkeit
  - Dokumentation
  - Testbarkeit
  - DRY/Wiederverwendbarkeit, einheitliche Design-Prinzipien

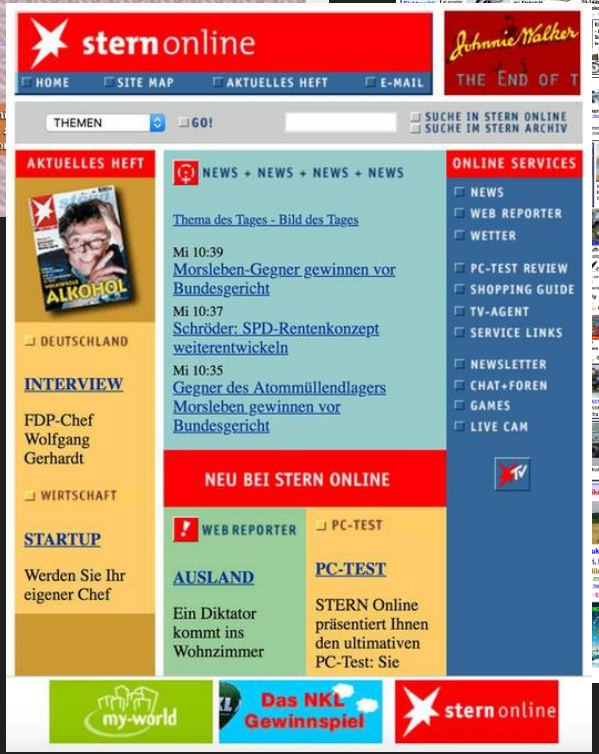
- Freiheit
  - UI Change
  - Design-Freiheit
  - Refactoring
  - Plattform-Agnostik
  - Plattform-Gnosis
  - Technologie-Wechsel
  - Integration
  - Skalierbarkeit
  - Logic Change
  - Flexibilität / Erweiterbarkeit für jeden Stakeholder
  - Skalierbarkeit

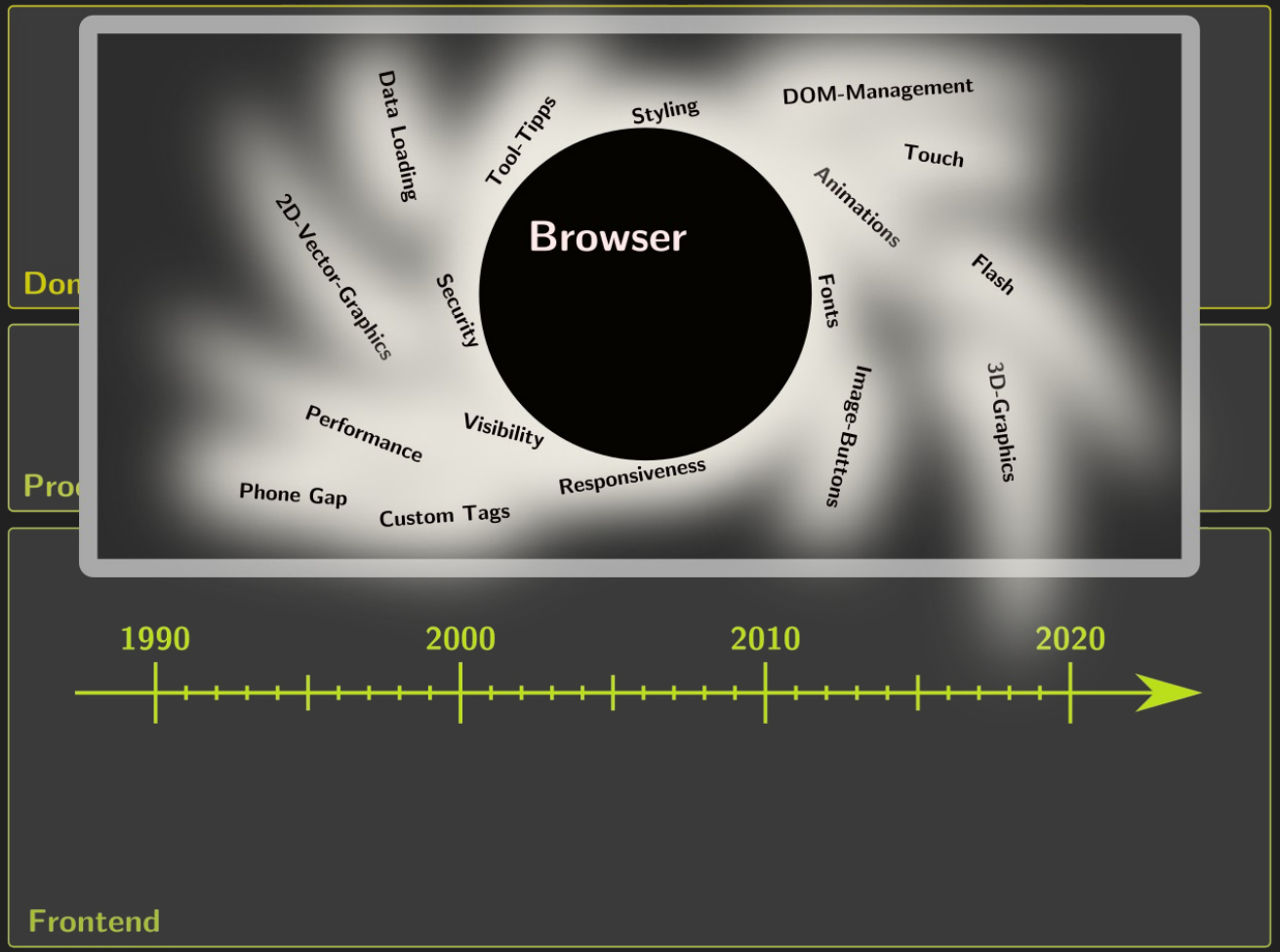
Was steckt hinter den Reibungszonen?

Der Browser - drei Jahrzehnte IT-Geschichte



Der Browser - drei Jahrzehnte IT-Geschichte

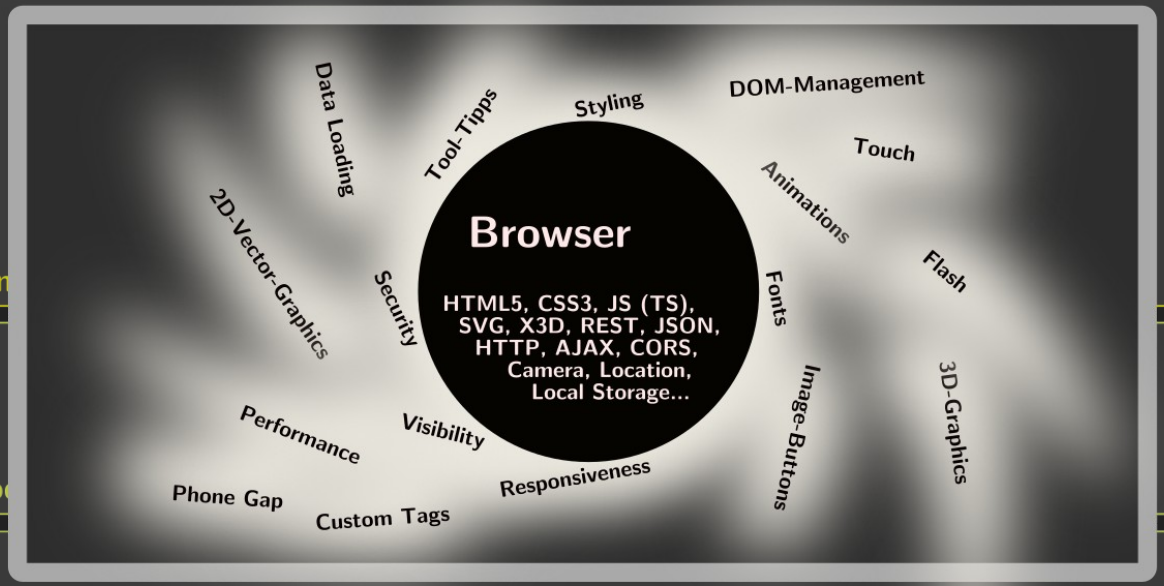




Der Browser - drei Jahrzehnte IT-Geschichte

- Immer wieder Experimente, neue Konzepte, Workarounds, Paradigmen
- Hypes und Moden
- Fun & Fails
- caniuse.com

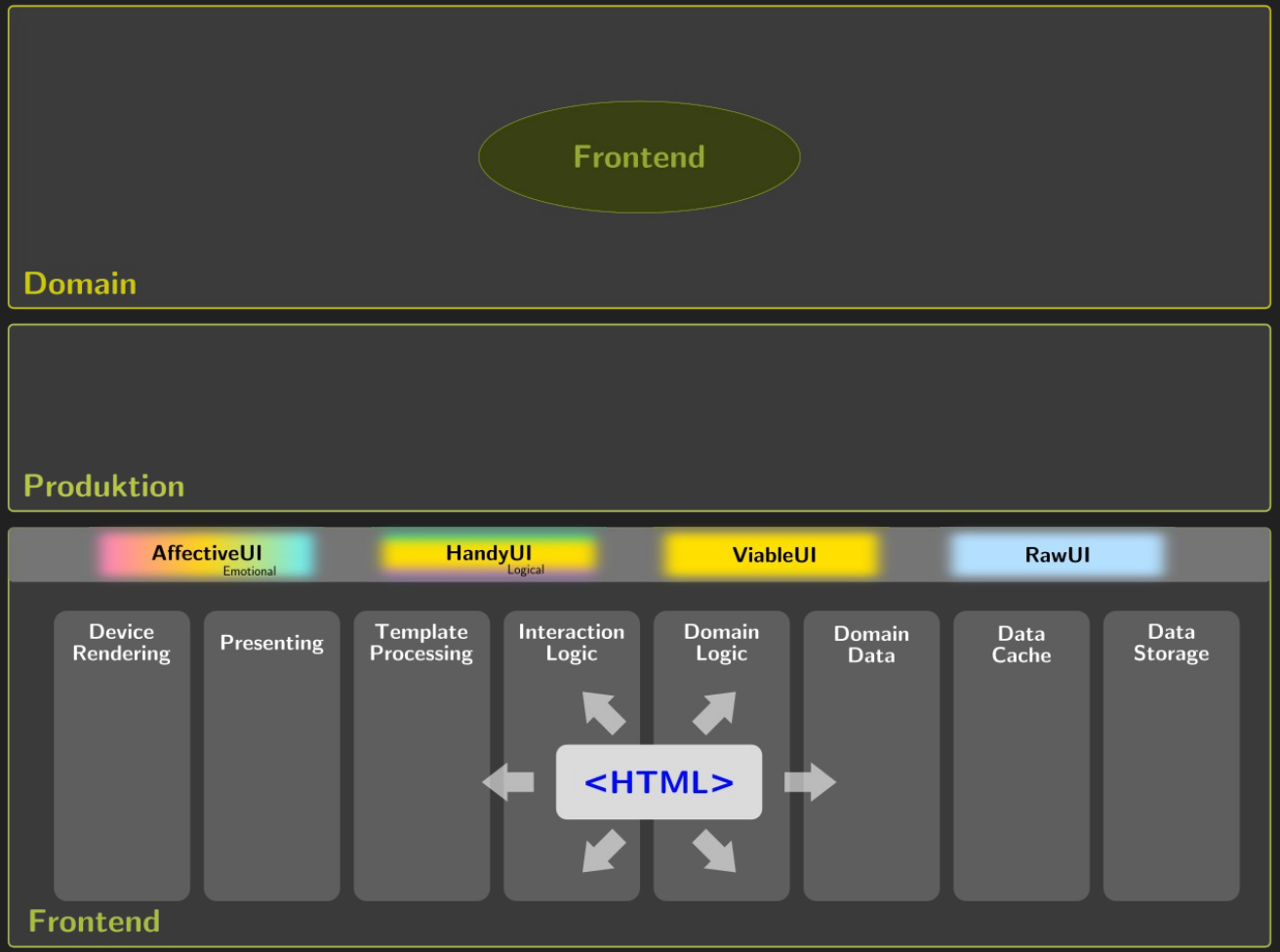
Der Browser - drei Jahrzehnte IT-Geschichte



- sedimentierte Technologie
- Standardisierung durch das W3C
- sehr gute Implementierungen
- **battle-tested**
- aber auch: **moving target**

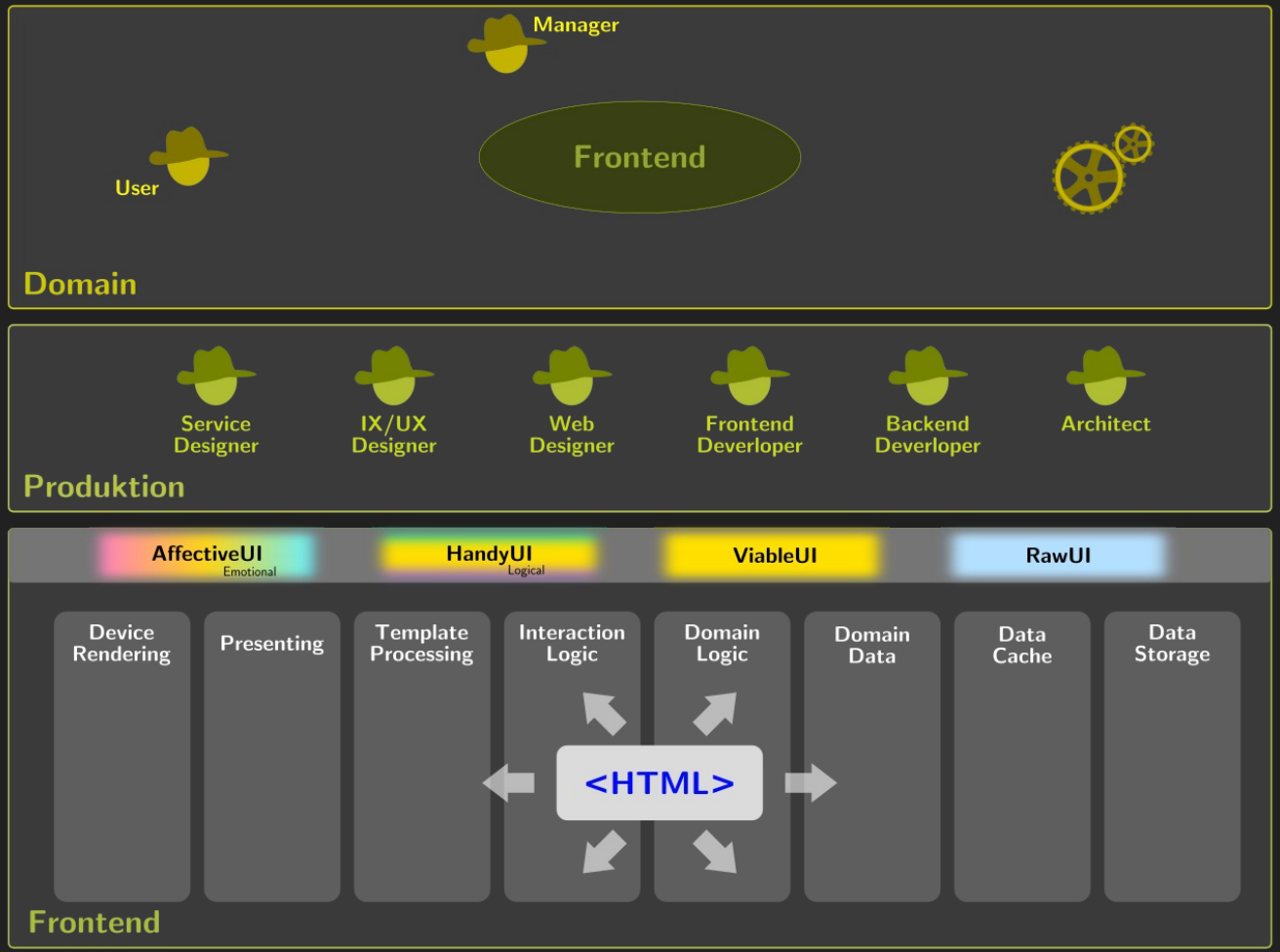
Frontend-Koevolution mit Browser

- von der HTML-Seite...



Frontend-Koevolution mit Browser

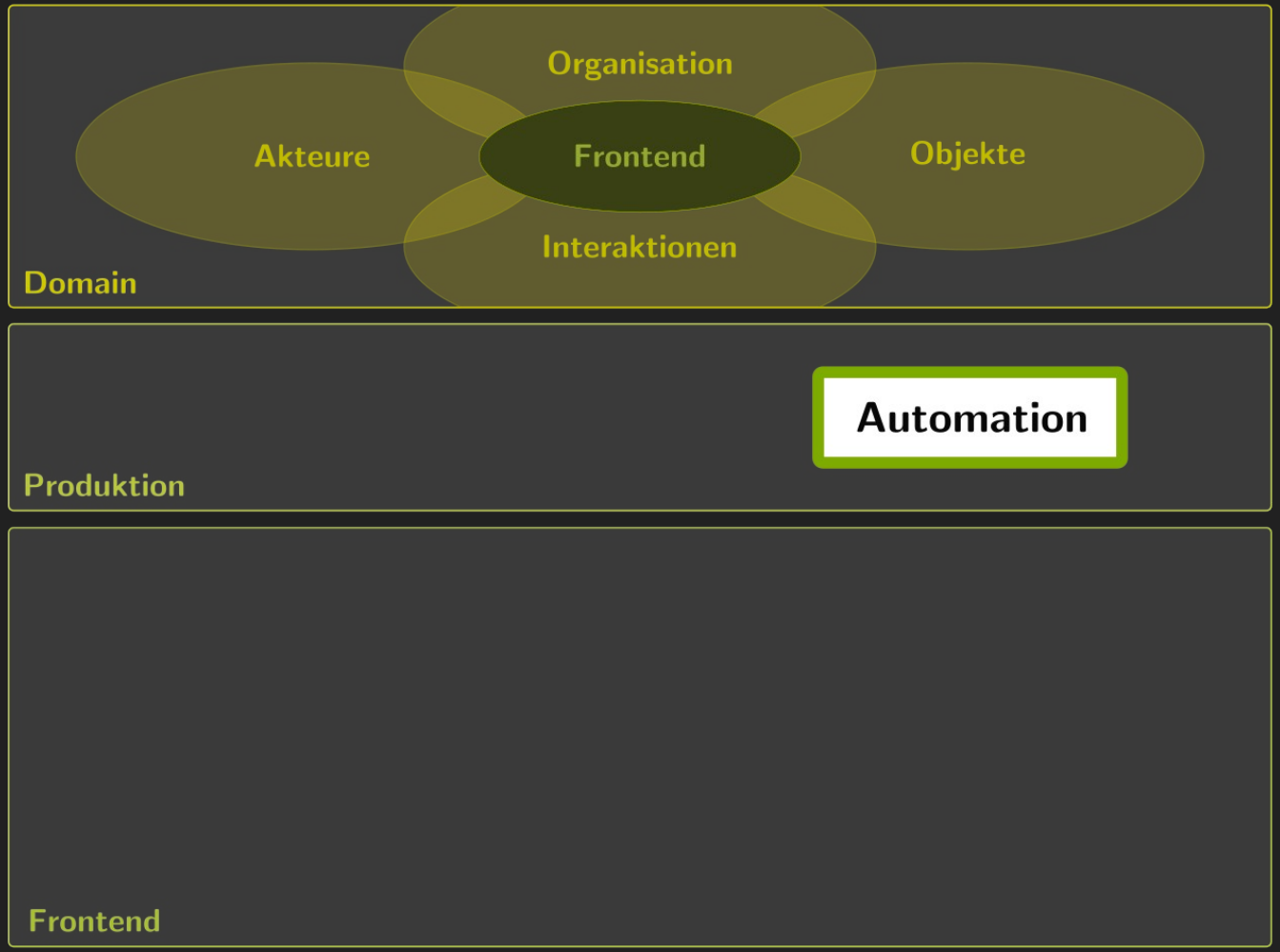
- von der HTML-Seite…
- ...zur komplexen App mit
  - vielfältigen Funktionen
  - und differenzierter Architektur



UIs und ihre Herstellung sind anspruchsvoll

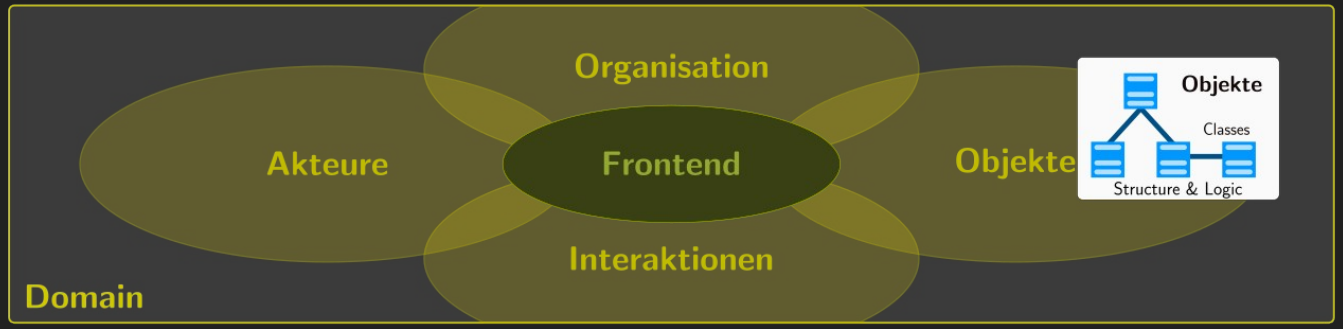
- Komplexe Plattform
- Allgegenwärtigkeit
- Multi-Stakeholder
- Produktions-Prozess
- Endnutzer: Vielfalt, Anspruch, Benutzbarkeit

Zwei (einseitige) Ansätze



Automation

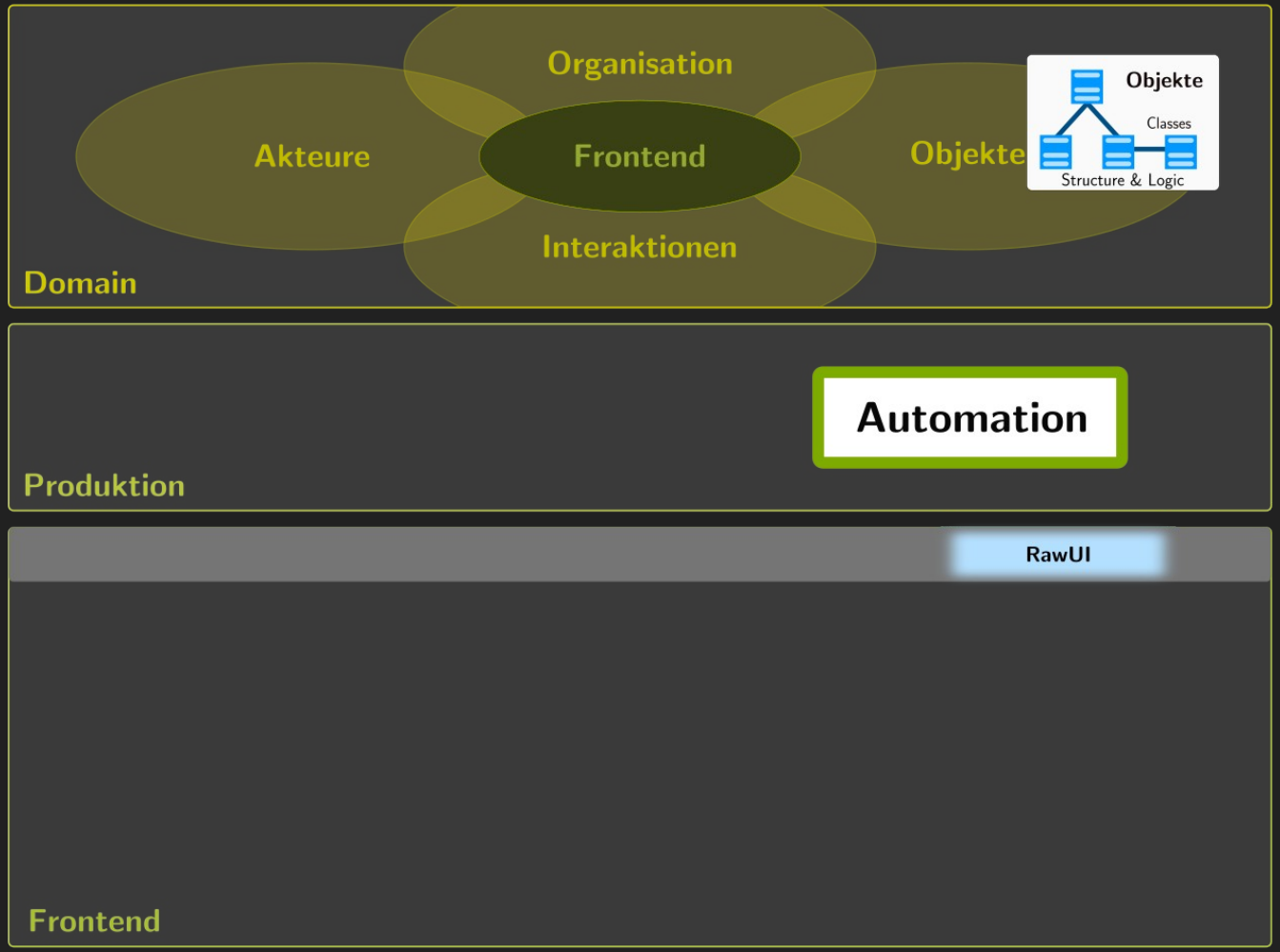
- Coding ist aufwendig, nicht DRY, eher technisch
- daher Wunsch, Code automatisch zu erzeugen
- (sofern nicht als Fertigteil bereits nutzbar)



Automation

Naheliegende Quelle:

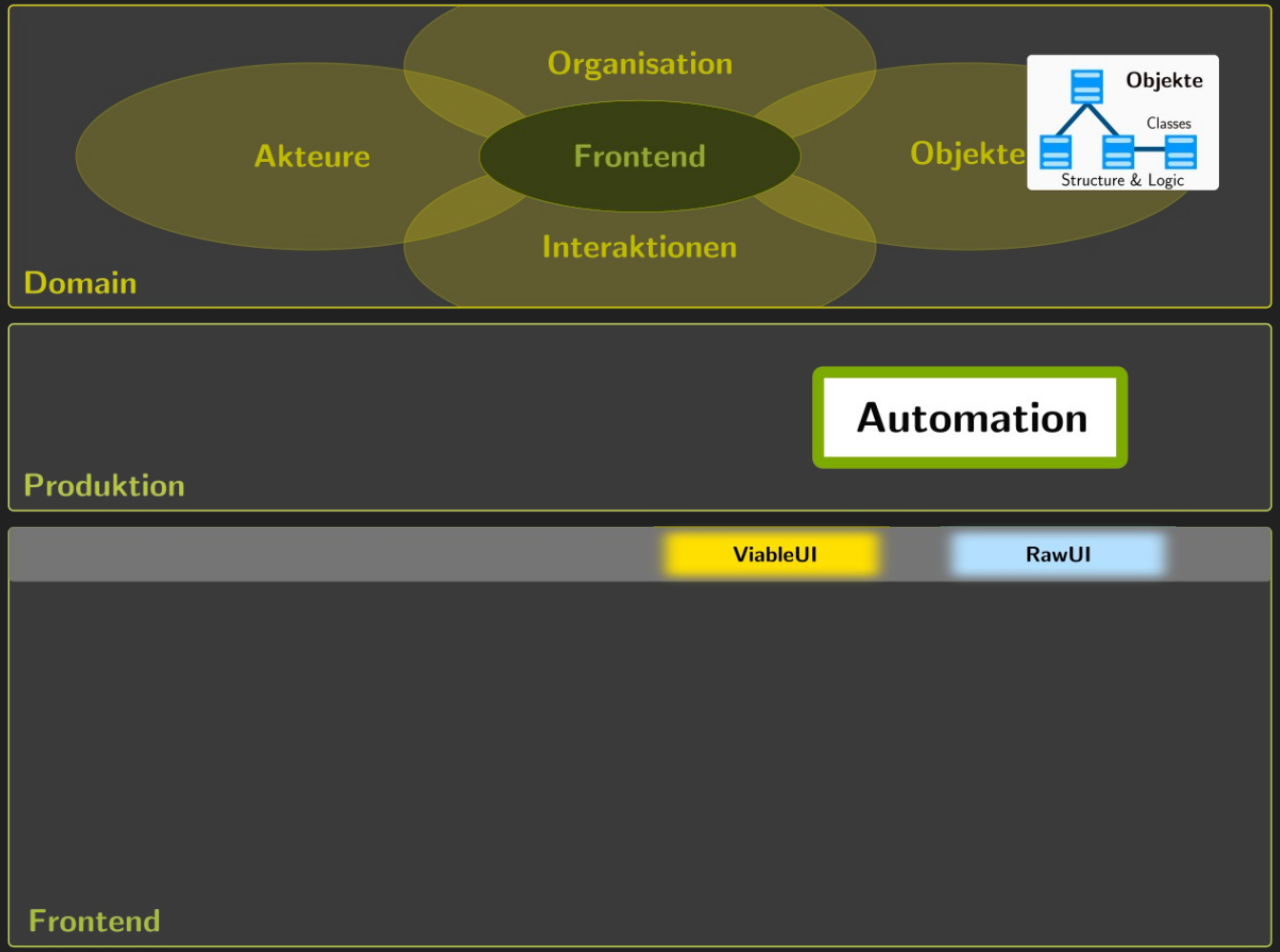
- ein oder mehrere Klassen-Modelle



Automation

“Raw UI”

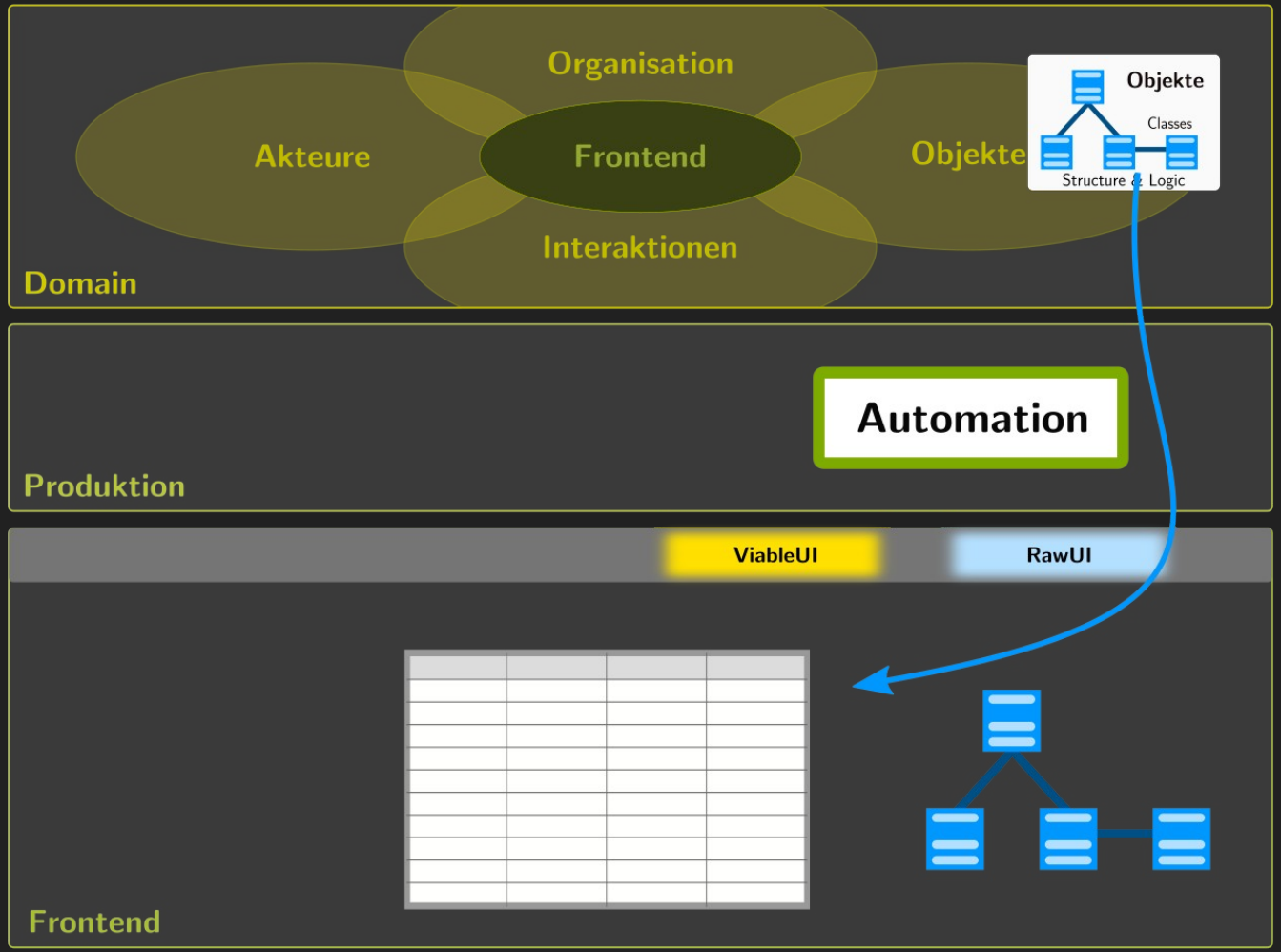
- CRUD-Operationen lassen sich ohne weiteres generieren
- ein einfaches Modell mit Klassen, Attributen und Typen reicht dazu aus



Automation

“Viable UI”

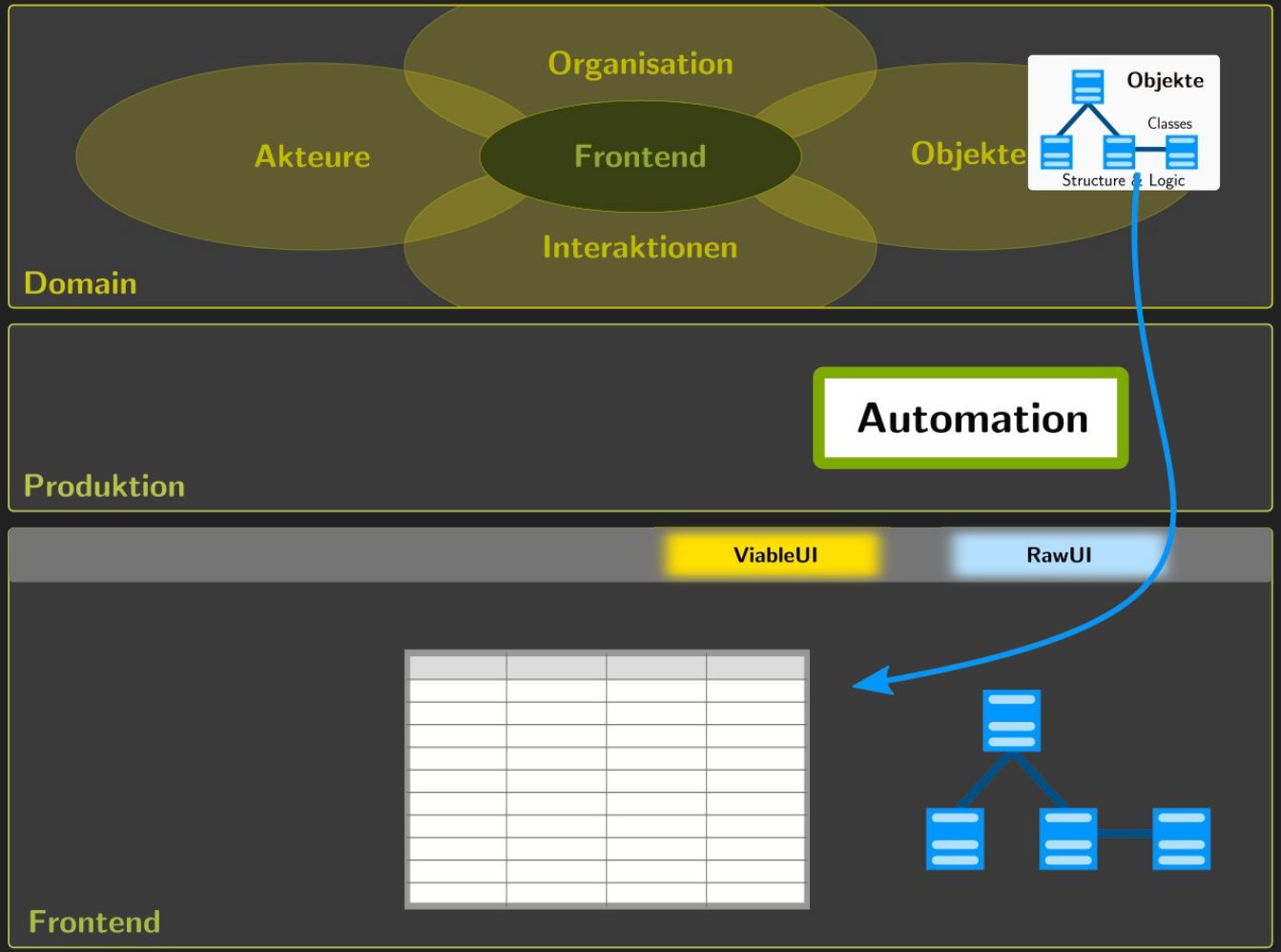
- mit einem gut gepflegten Modell können auch weitgehend ungültige oder inkonsistente Aktionen verhindert werden
- das erfordert die Angabe von Constraints, Berechtigungen usw.



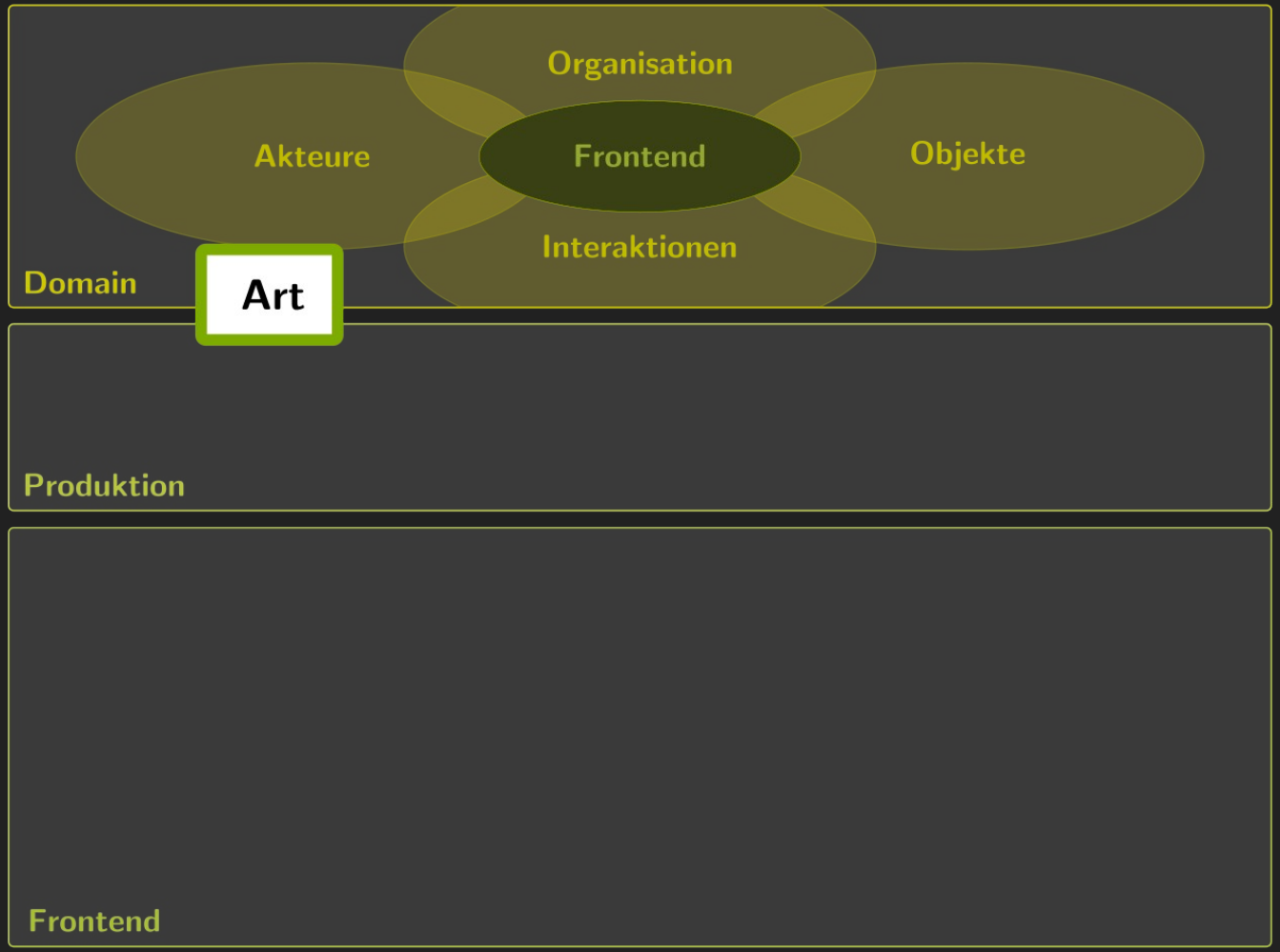
Automation

- das funktioniert, sieht aber ein bisschen aus wie ein Spreadsheet oder ein Datenbank-Editor
- den Anforderungen an moderne, nutzerfreundliche, ansprechende UIs genügt das hinten und vorne nicht

Automation

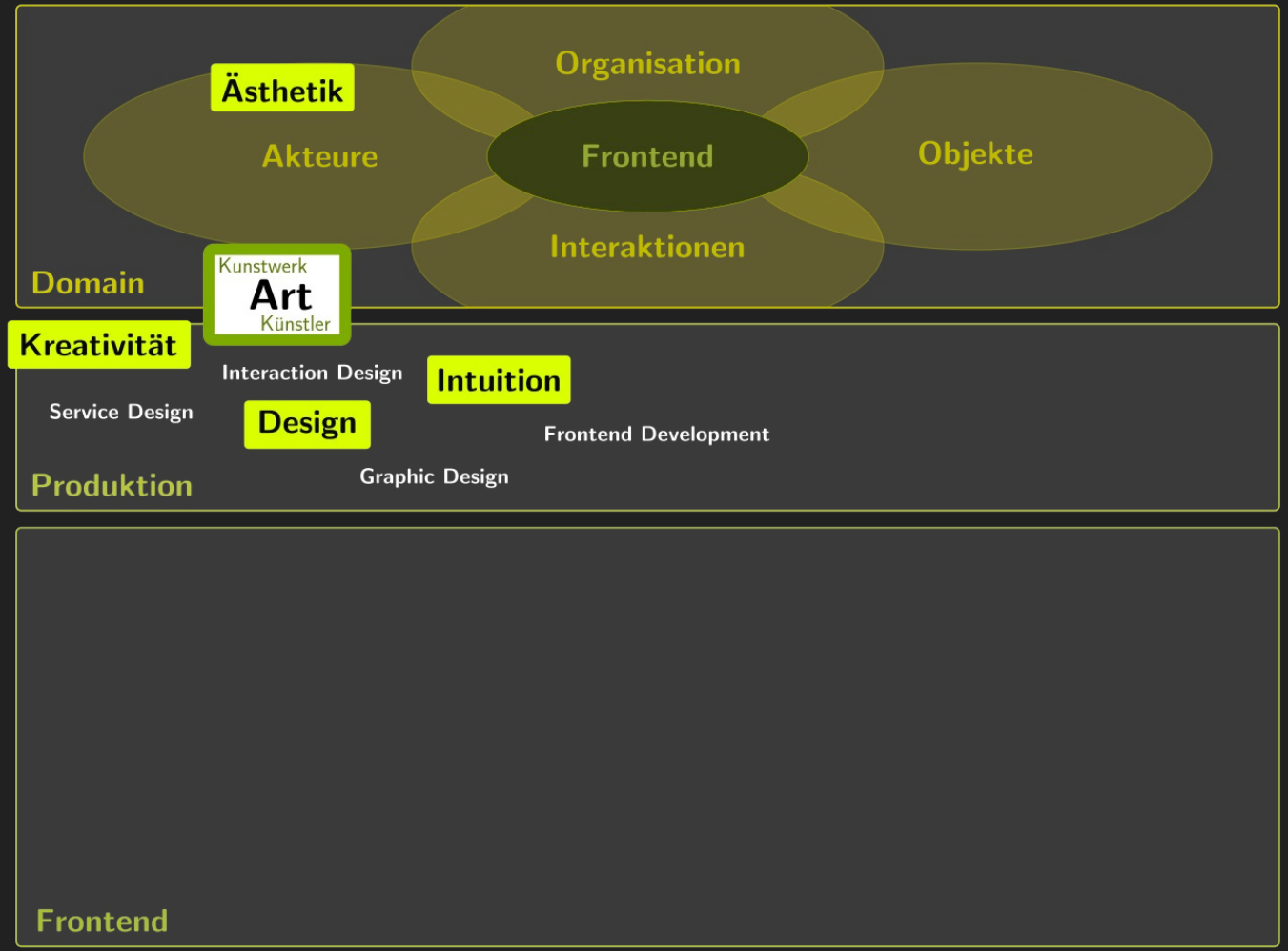


- auch die Gestaltbarkeit ist stark beschränkt
- manuelle “Nachbearbeitung” ist in großem Umfang nötig
- damit wird die Re-Generierbarkeit verhindert
- taugt, wenn überhaupt, nur zum Scaffolding



Art

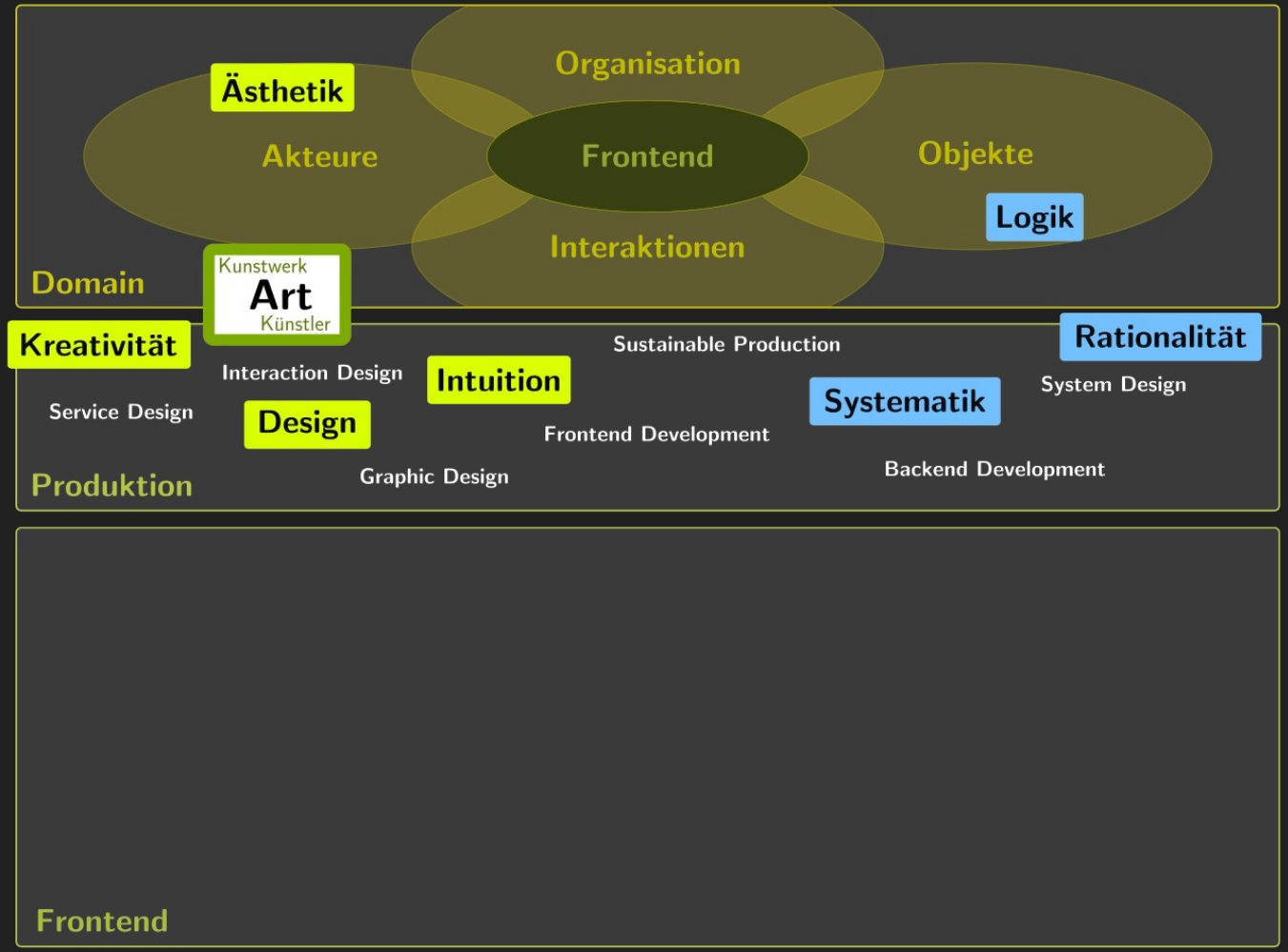
- Gestaltung aus Nutzungs-Sicht, von “außen”
- Technik zweitrangig, zunächst nicht beachtet



Art

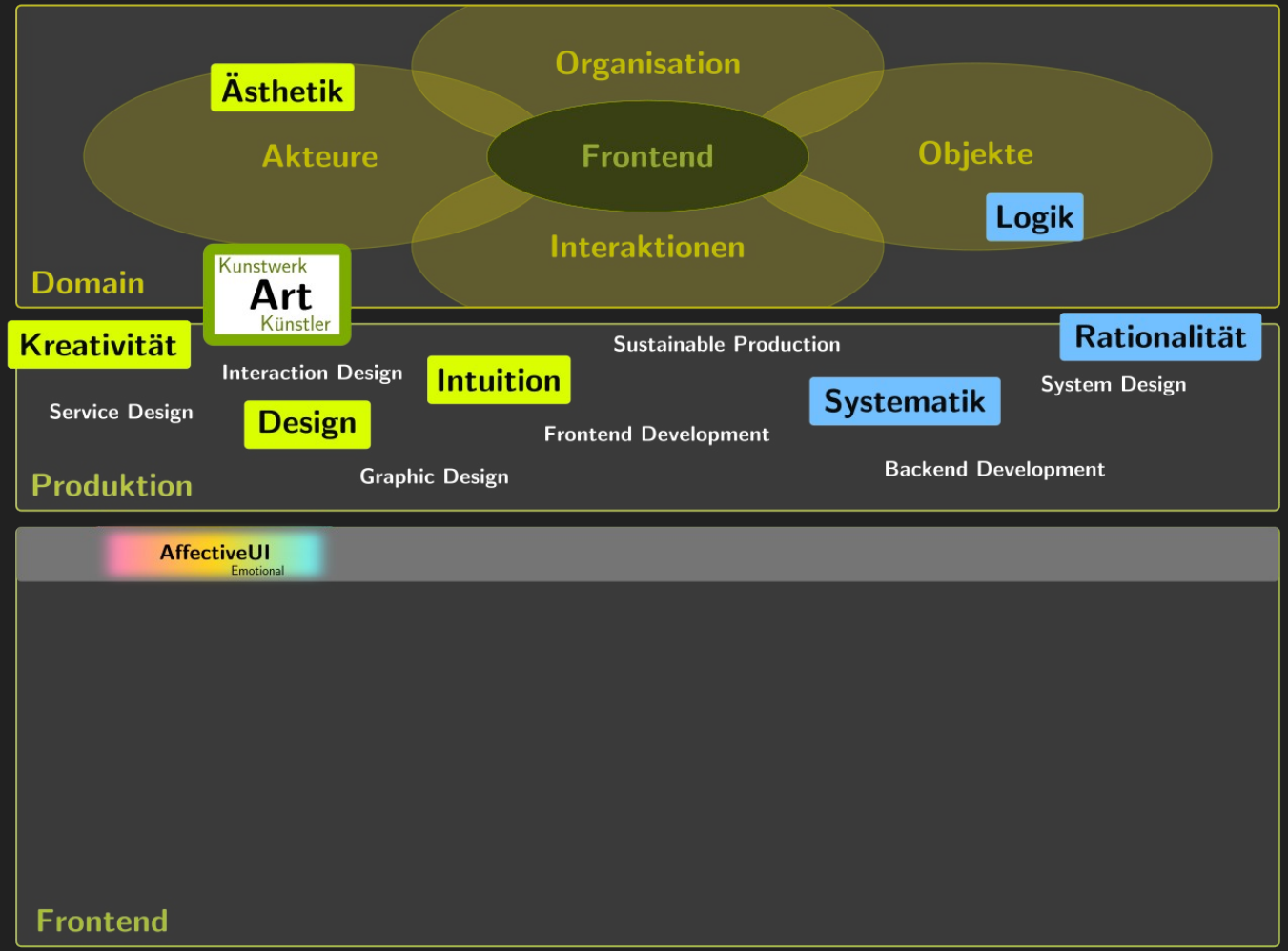
- gekennzeichnet durch verschiedene Design-Prozesse
- enthalten “künstlerische” Elemente: Kreativität und Intuition bei den Akteuren, Ästhetik im Produkt

Art



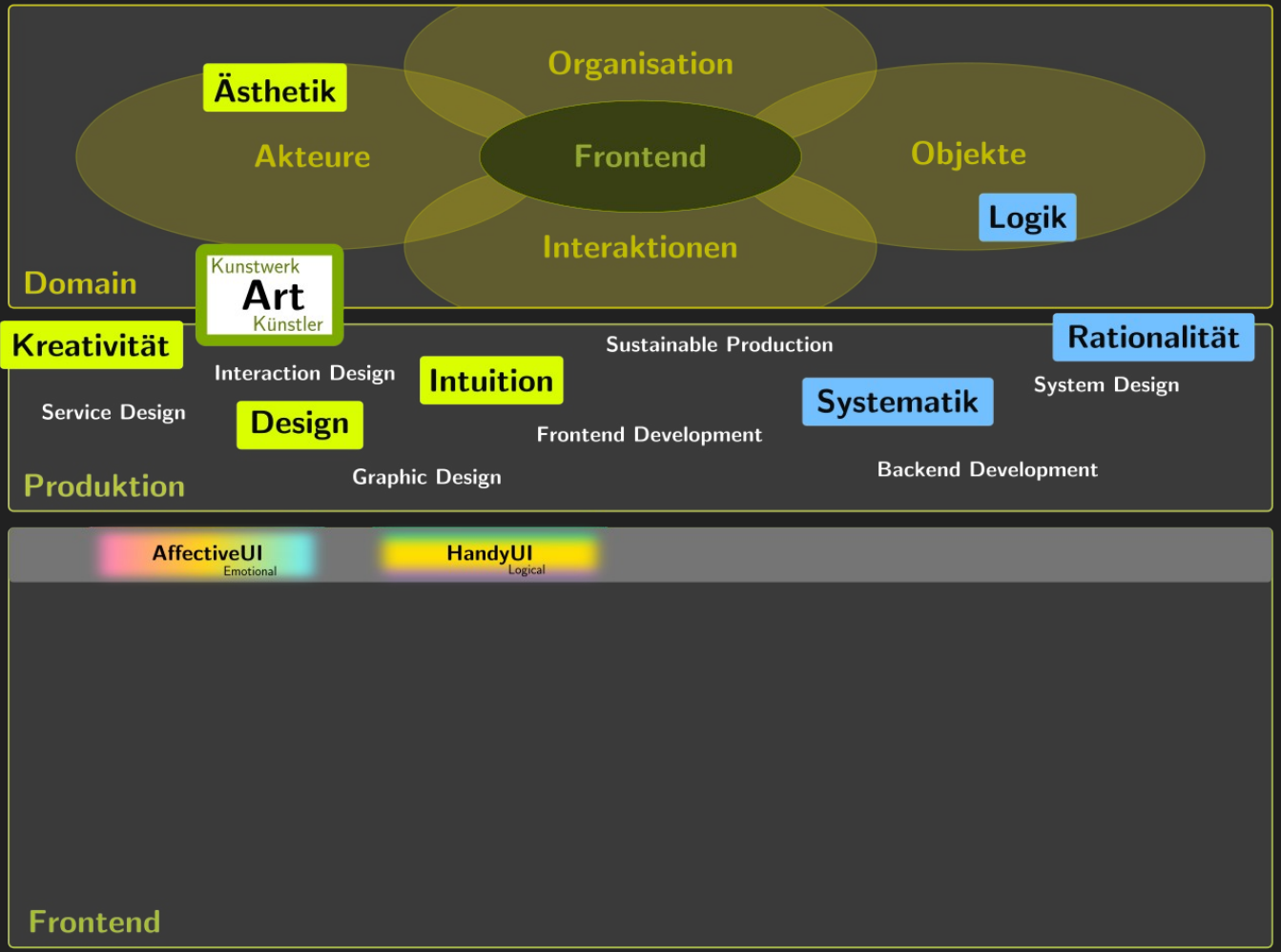
- steht im Kontrast zur “kalten Technik”
- jedoch auch Berührungspunkte, vor allem bedingt durch Usability Design: Konsistenz passt zu Systematik, Verstehbarkeit passt zu Logik

Art



“Affective UI”

- emotional ansprechend
- ästhetische Ausgestaltung

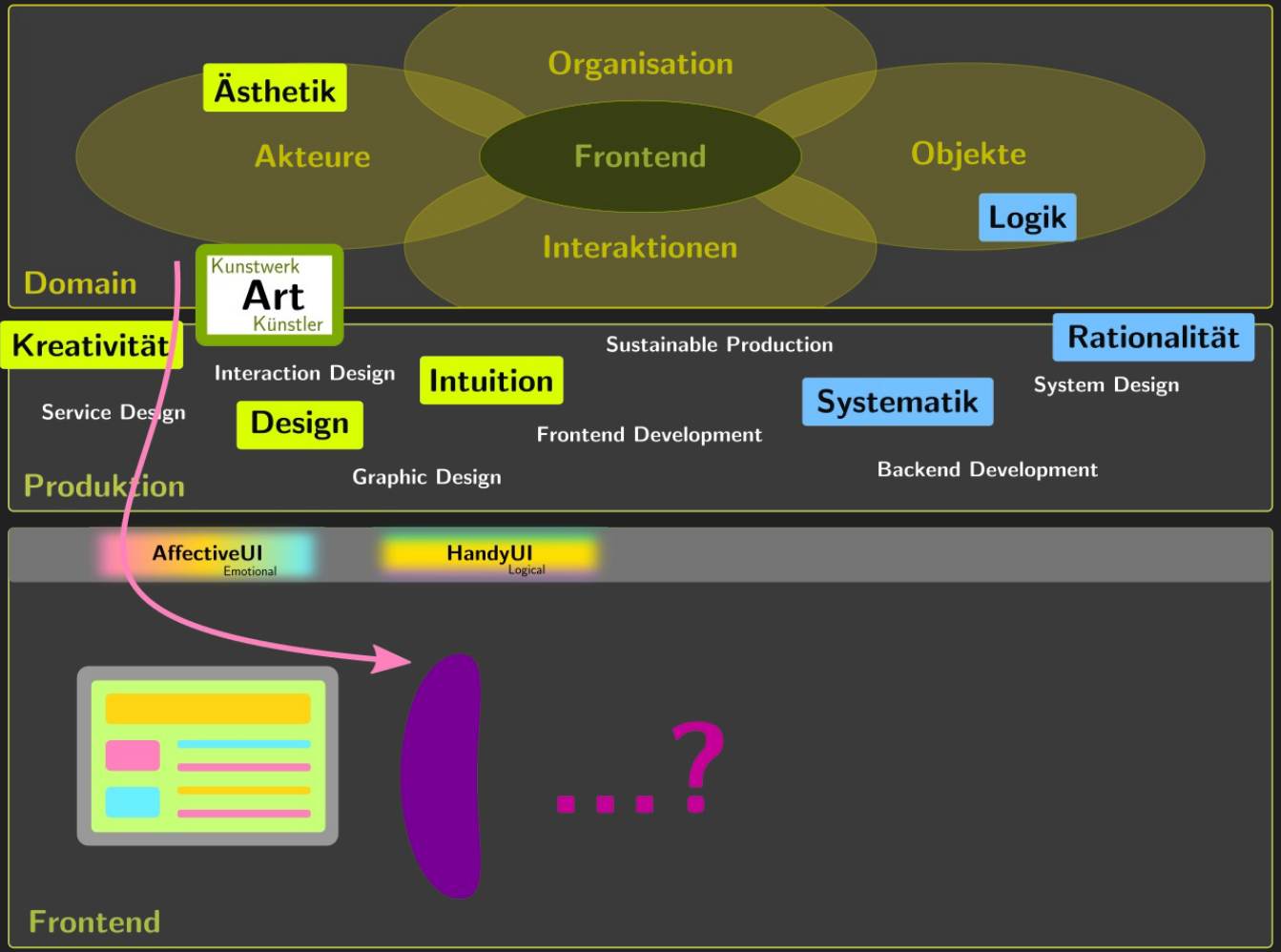


Art

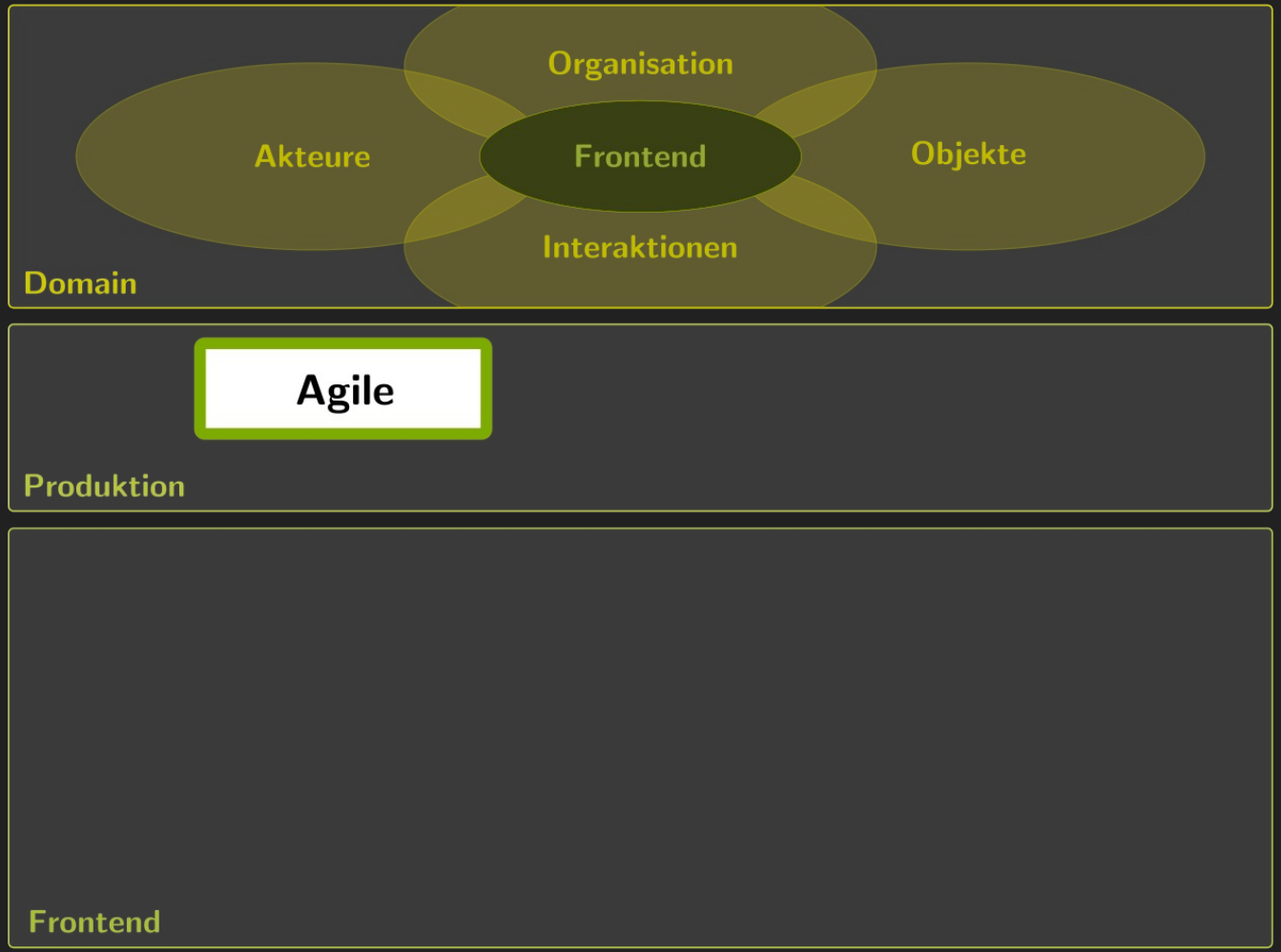
"Handy UI" (*"nahegelegt"*)

- freie, zweckmäßige Gestaltung der räumlichen und zeitlichen Anordnung von UI-Elementen

Art

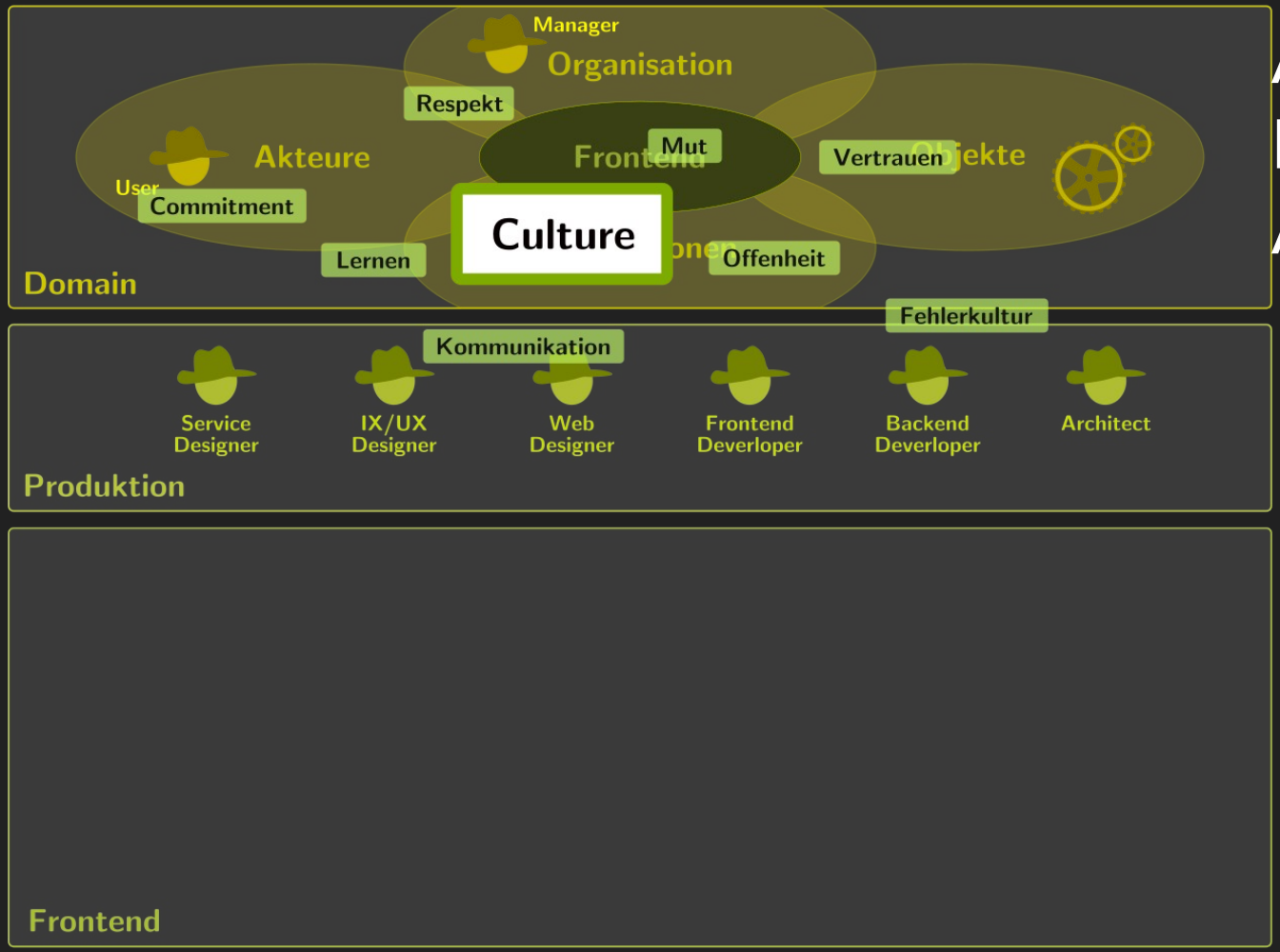


- zumindest anfangs und kurzfristig gute, bedarfsgerechte Ergebnisse
- ohne zusätzliche Maßnahmen entsteht oft *Technische Schuld*



Agilität

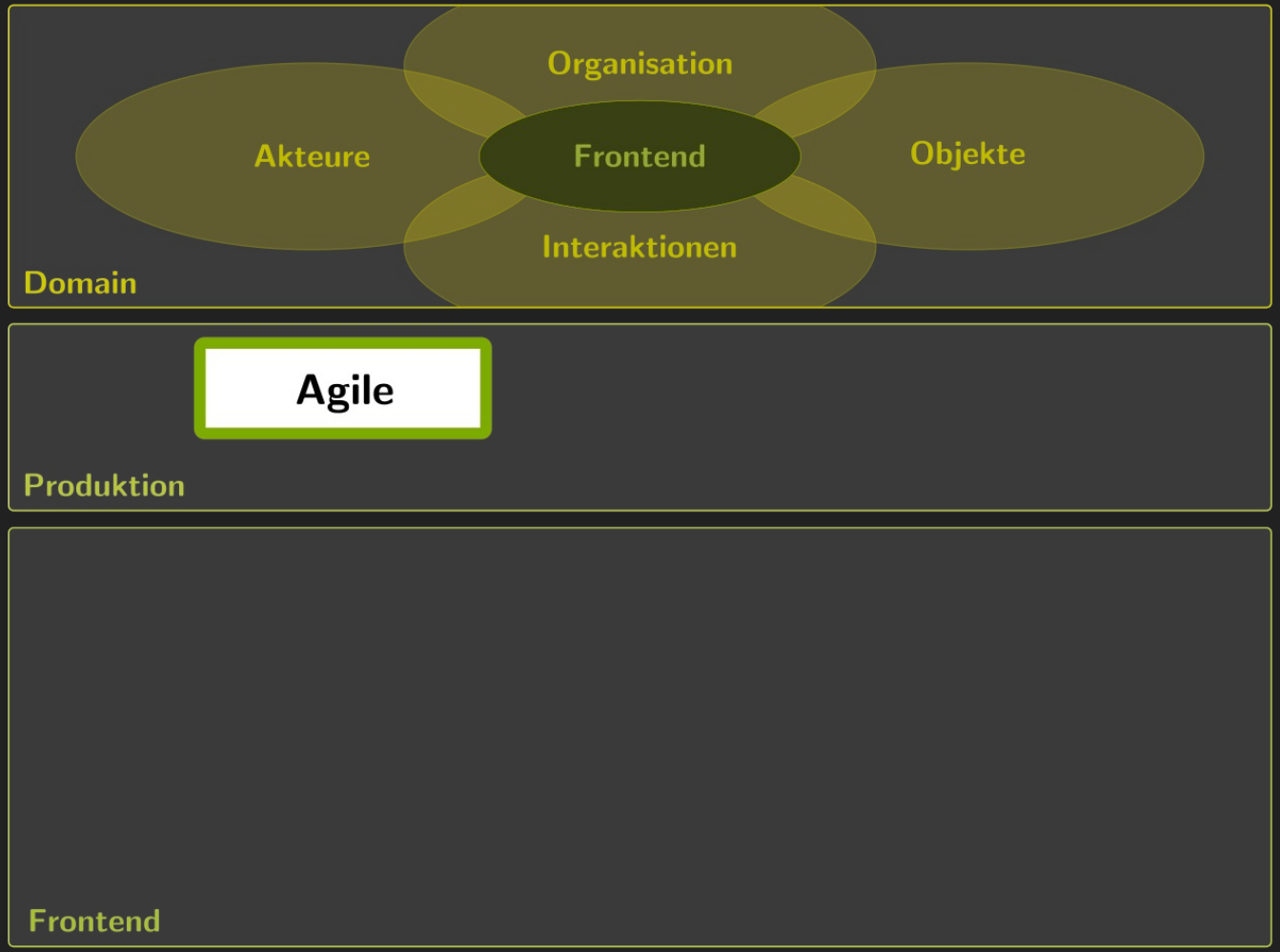
- Agile Kultur
- Agiler Prozess



Agile Kultur: Kommunikation, Arbeitsweise

- Feedback, Lernfähigkeit, guter Umgang mit-einander
- Ist, und bleibt wichtig für erfolgreiche Zusammen-Arbeit
- Bezüge zum Bauhaus “kooperierende Kunsthandwerker”

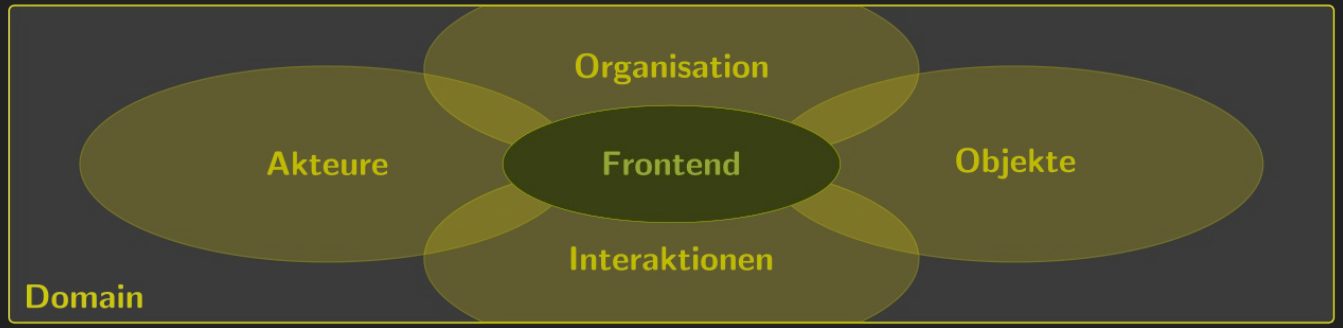
Agilität: mehr als Kultur...



Agiler Prozess

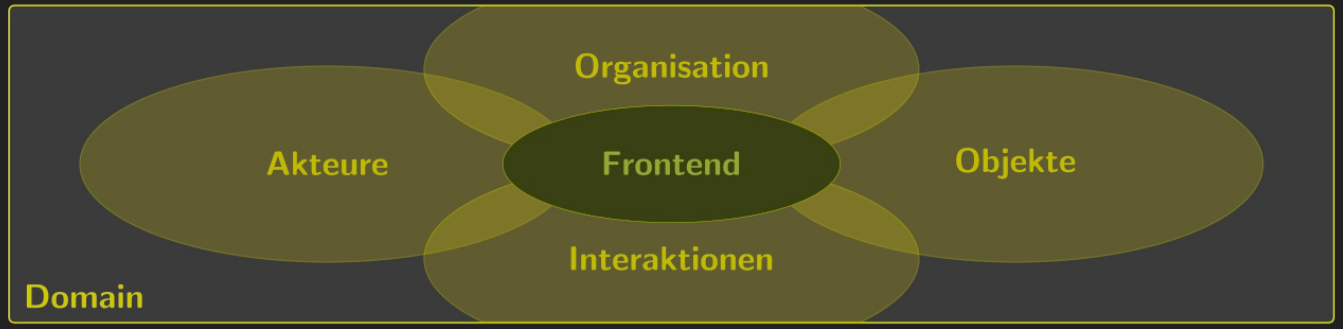
- Ziel: kontinuierliche, iterative Erzeugung von geschäftlichem Nutzen

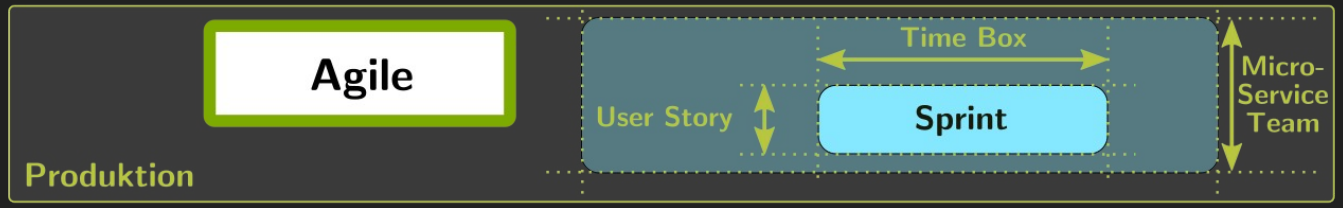
Agiler Prozess



- Iterationen sind zeitlich begrenzt
- Iterations-Inhalte müssen folglich im Umfang begrenzt sein
- Kurzfristigkeit ermöglicht Flexibilität des Prozesses

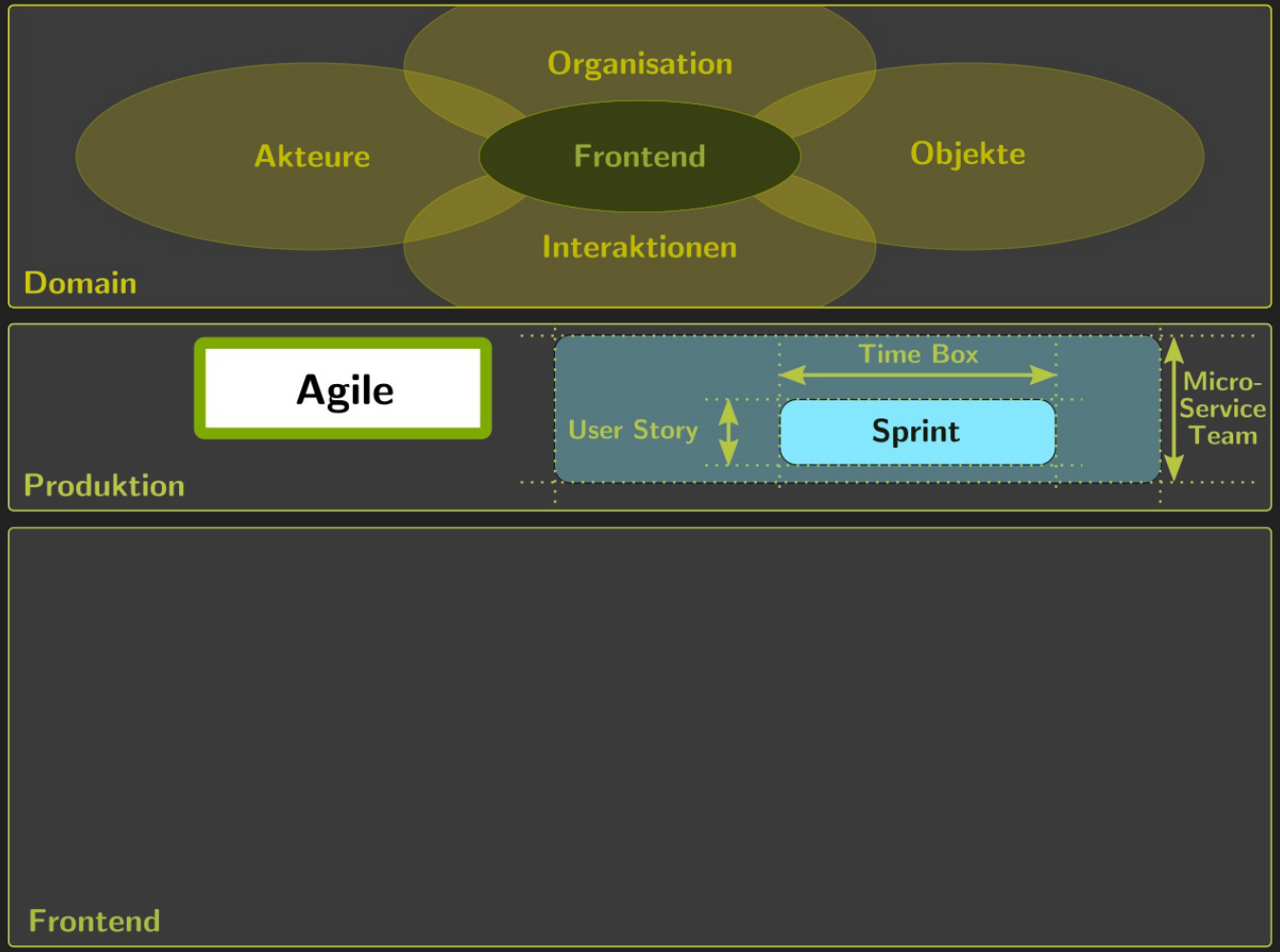
Agiler Prozess





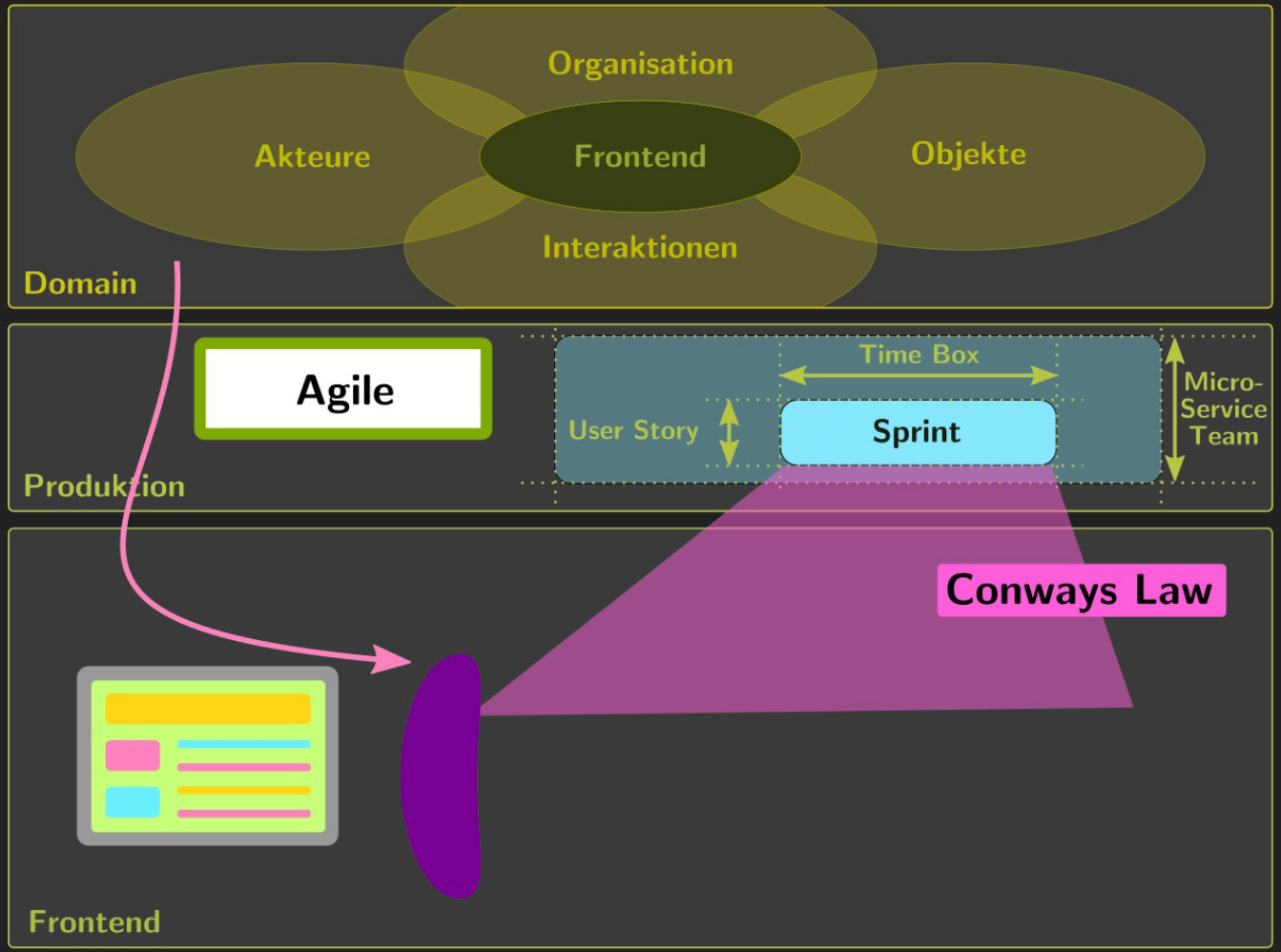
- Einteilung in Microservice-Teams
- In sich abgeschlossen - Entkopplung
- orientiert an Bounded Context

Agiler Prozess



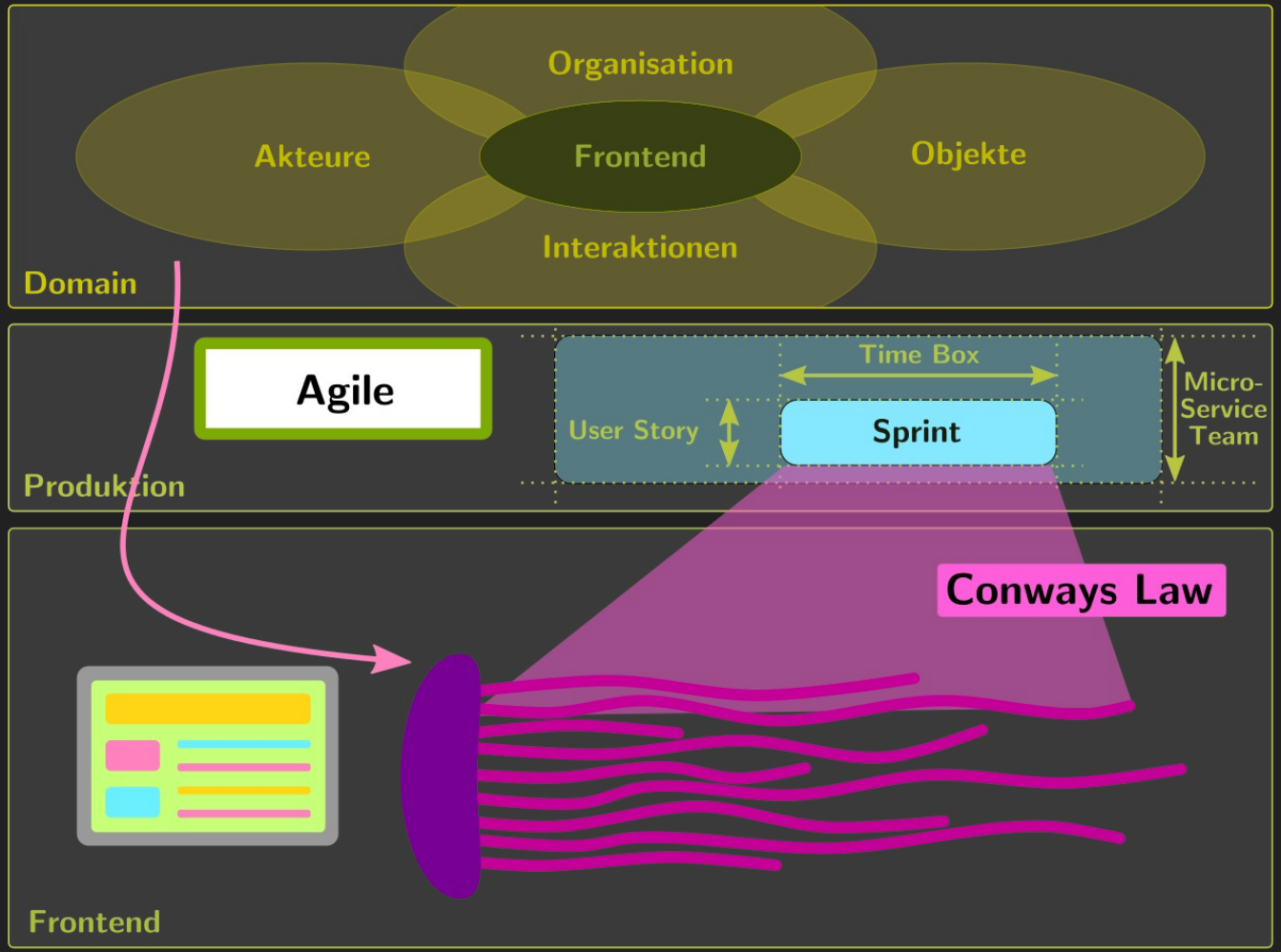
- “Sozio-technisches Produktionssystem”
- **Pro:** selbststeuernd, anpassungsfähig, lernfähig
- weniger abhängig
- **Contra:** Kurzfristigkeit belohnt kurzfristige Ziele, Langfristiges nicht fest verankert
- Systemqualität
- Manpower/Aufwand

Conways Law



- Widerspiegelung der Arbeitsweise in der Strukturierung des Produkts

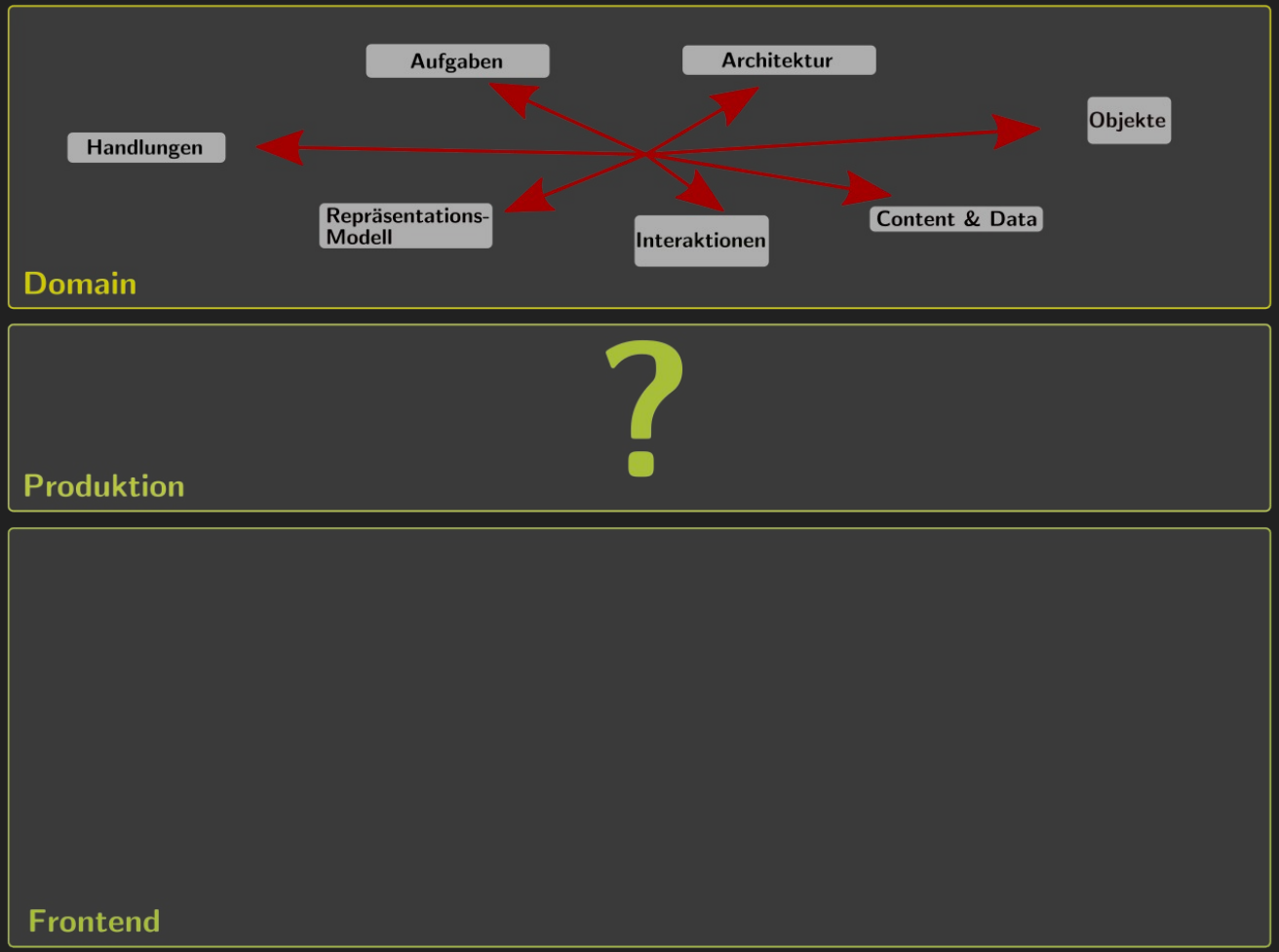
Conways Law



- Frontend-Qualle: Features werden Tentakel-artig implementiert
- Zwei Gründe: Kurzfristigkeit und primäre Nutzer-/Nutzen-Orientierung
- Tentakel: Form (Conway), Giftigkeit (Qualität)

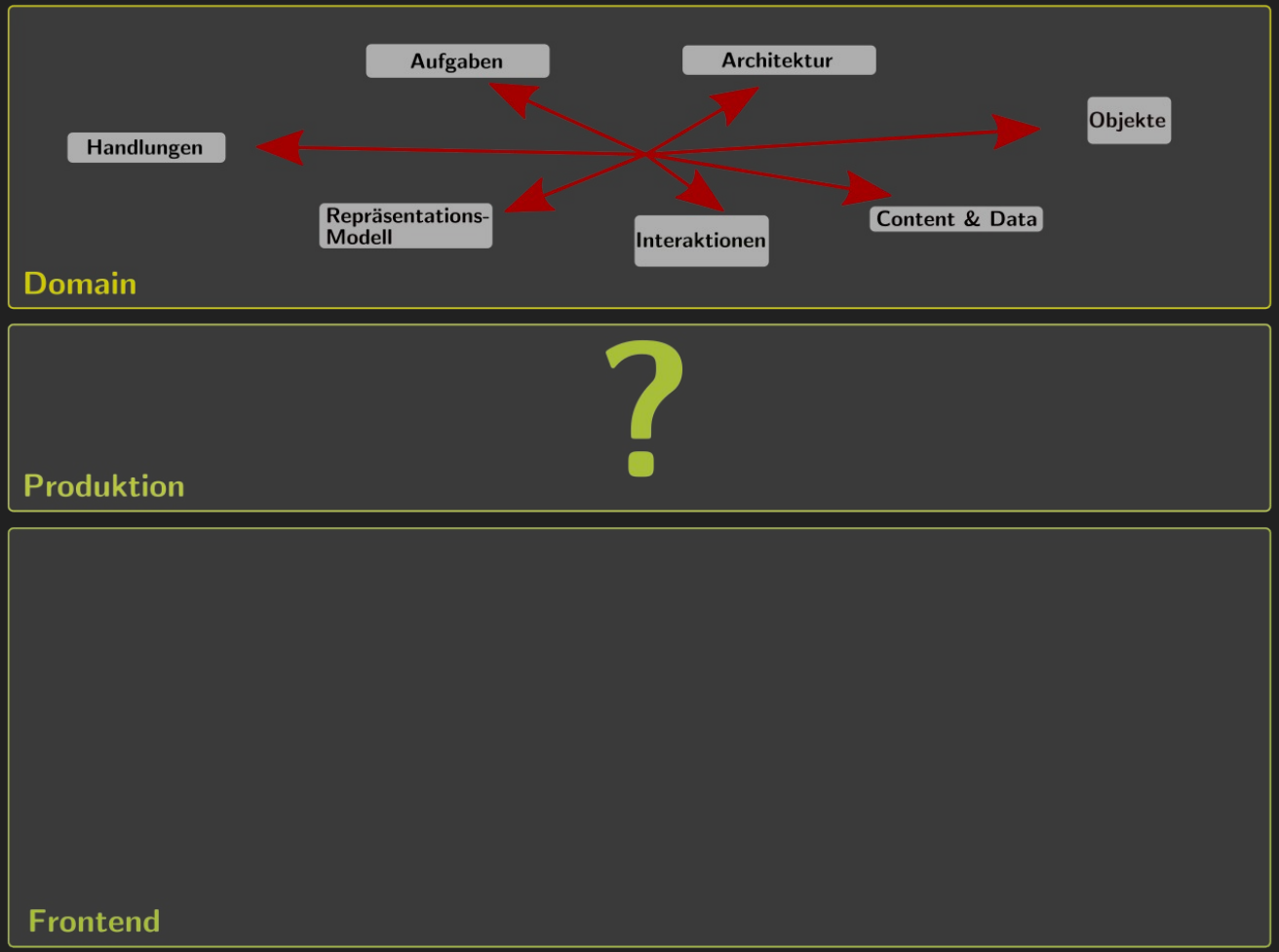
Frontend

Backend



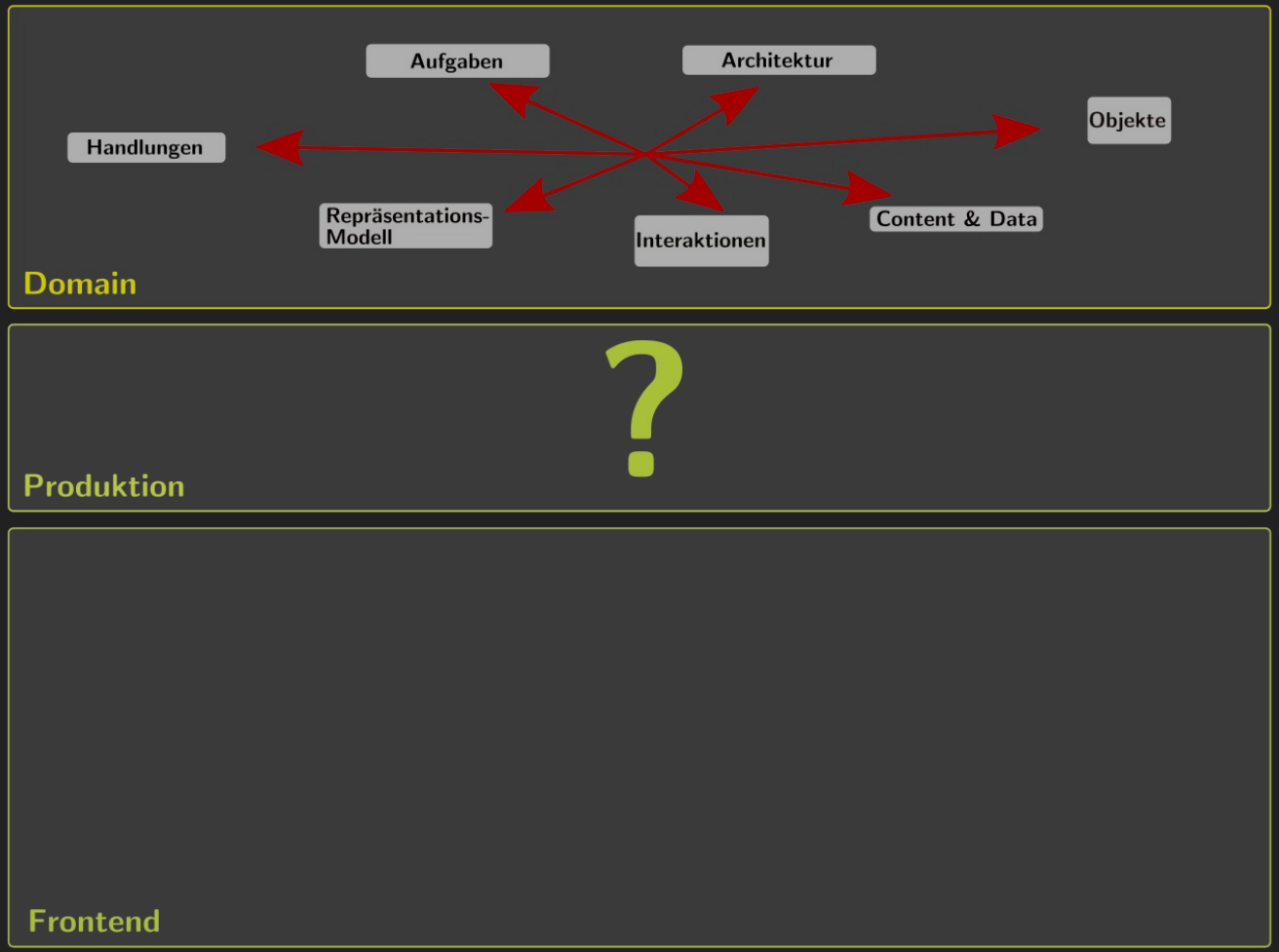
Entstehung der Reibung

- Agile Kultur, agiler Prozess **mindern** Folgen von Reibung, oder verhindern durch **radikale Trennung** (zu einem Preis)
- Was ist eigentlich die **Ursache** für die Entstehung?



Entstehung der Reibung

- Offensichtlich: Umgang mit Artefakten anderer Stakeholder problematisch
- Gibt es einen guten Mediator?



Anforderungen an Artefakte

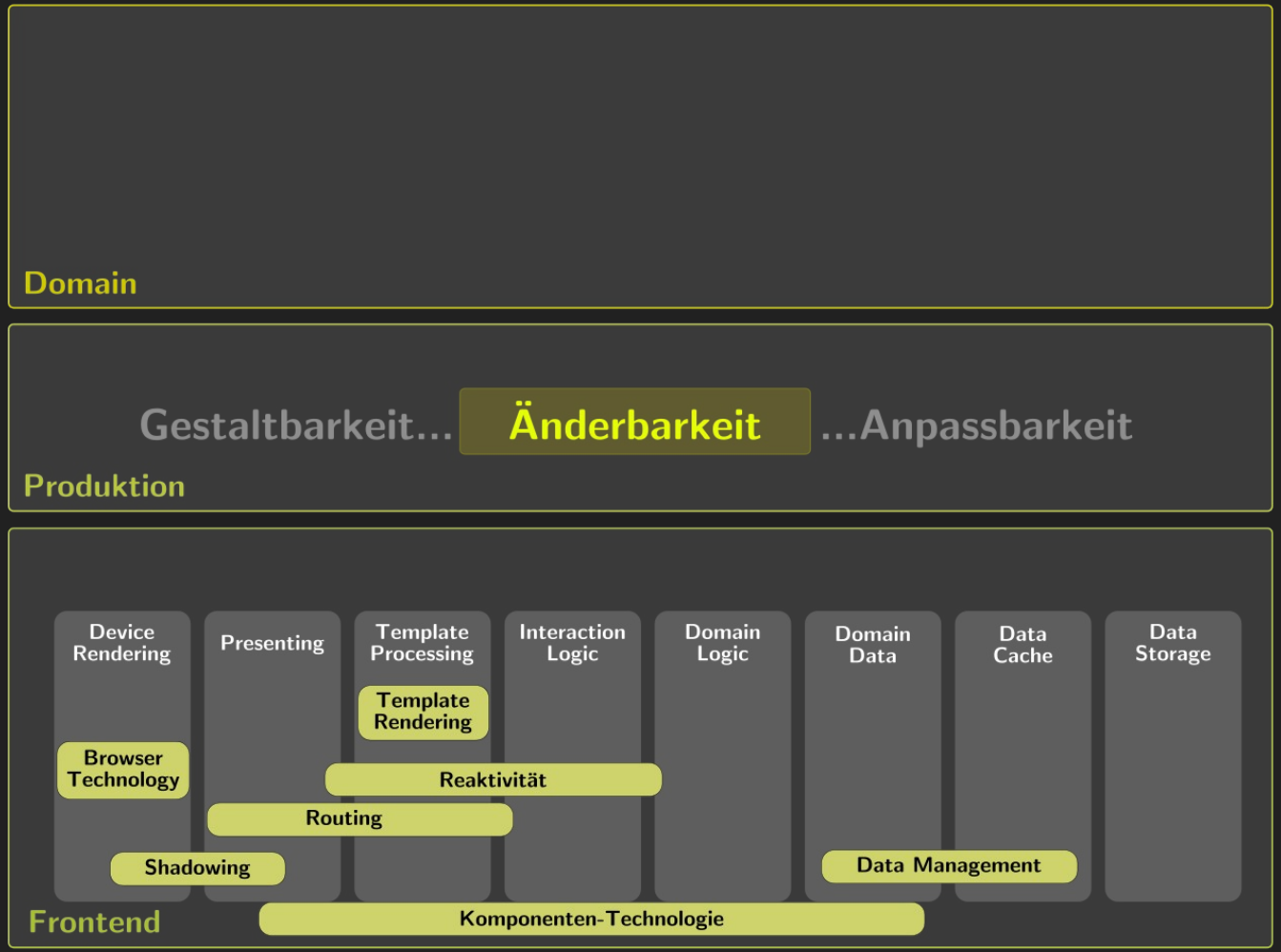
Änderbar

- überhaupt (Zugänglichkeit) und verständlich (Deklarativität)
- ordentlich (Modularität) und nachhaltig (Architektur)
- einfacher (Funktionalität & Automatismen)

Wie helfen Frameworks?

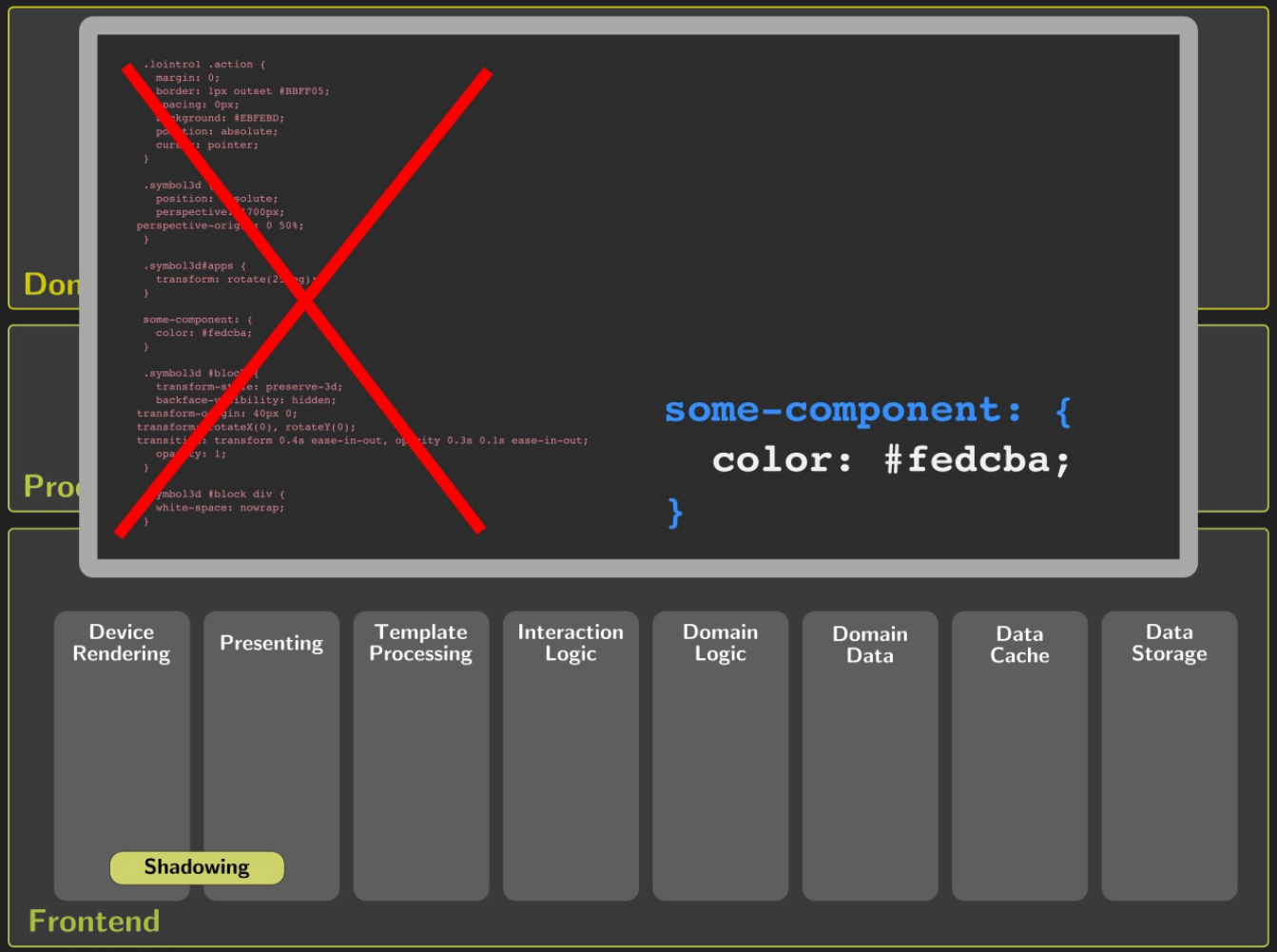
Fähigkeiten

- Zugänglichkeit
- Deklarativität
- Schichten
- (Komfort)funktionen
- (Opinionated)



Templates

- deklarative Beschreibung des Seitenlayouts



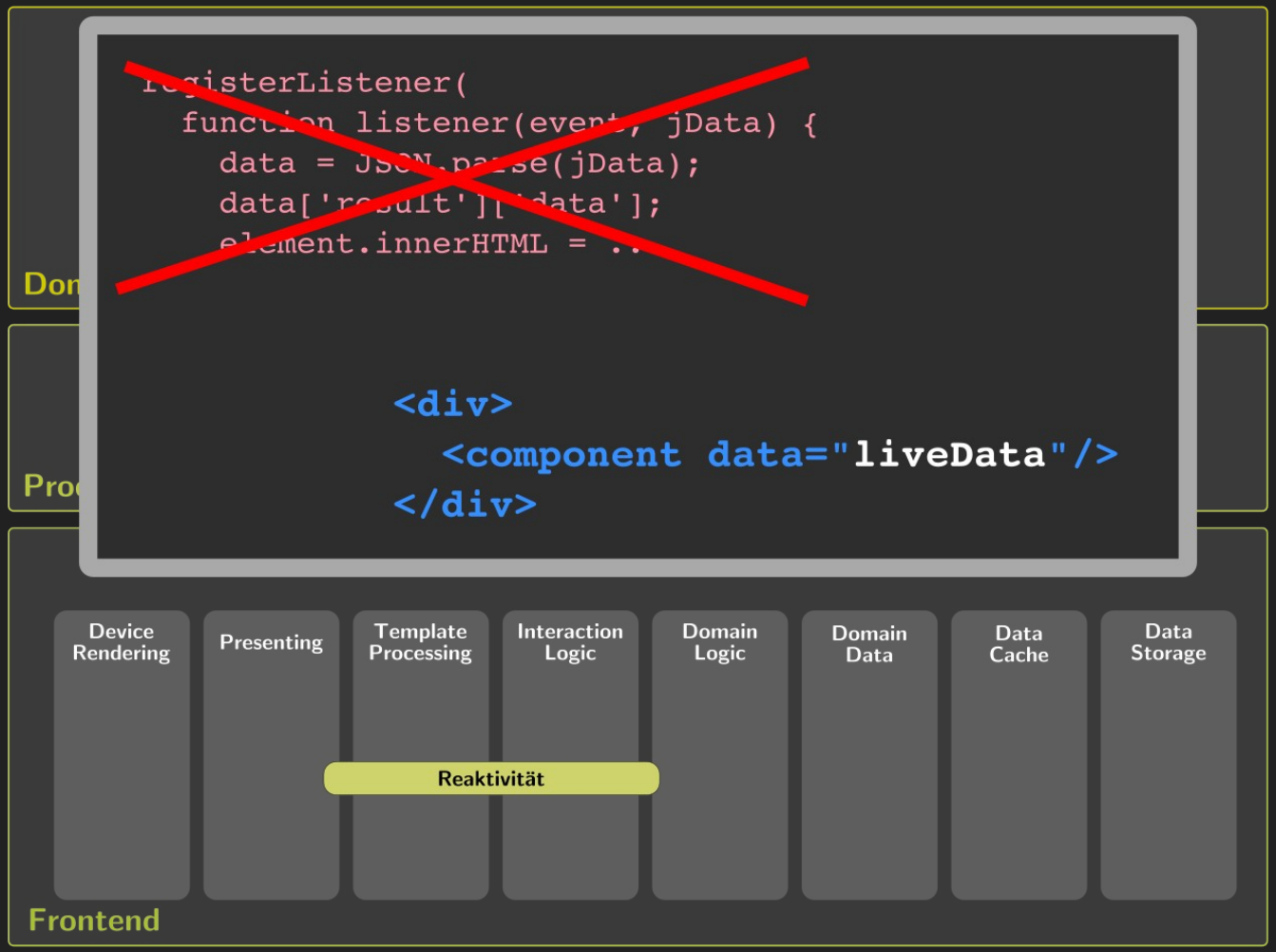
Shadowing / Encapsulation

- schützt Komponenten im Browser vor unerwünschten Wechselwirkungen (CSS, DOM)

Reaktivität

```
registerListener(
  function listener(event, jData) {
    data = JSON.parse(jData);
    data['result']['data'];
    element.innerHTML = ..
```

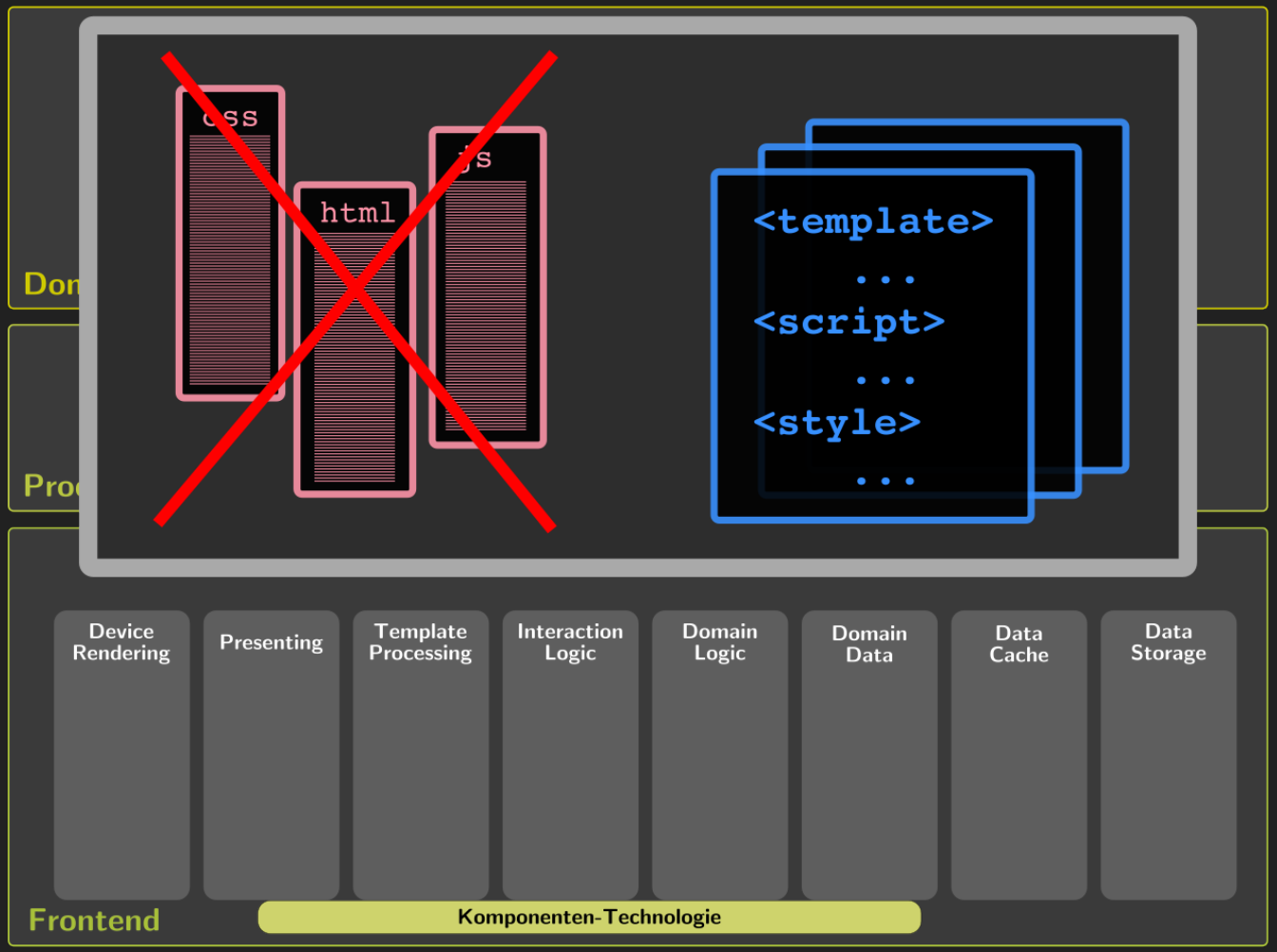
```
<div>
  <component data="liveData"/>
</div>
```



- versteckt und gewährleistet Update-Verarbeitung

Routing

- vereinfacht navigationsbedingte Anzeige von Komponenten
- durch deklarative Beschreibung in Routing-Tabelle



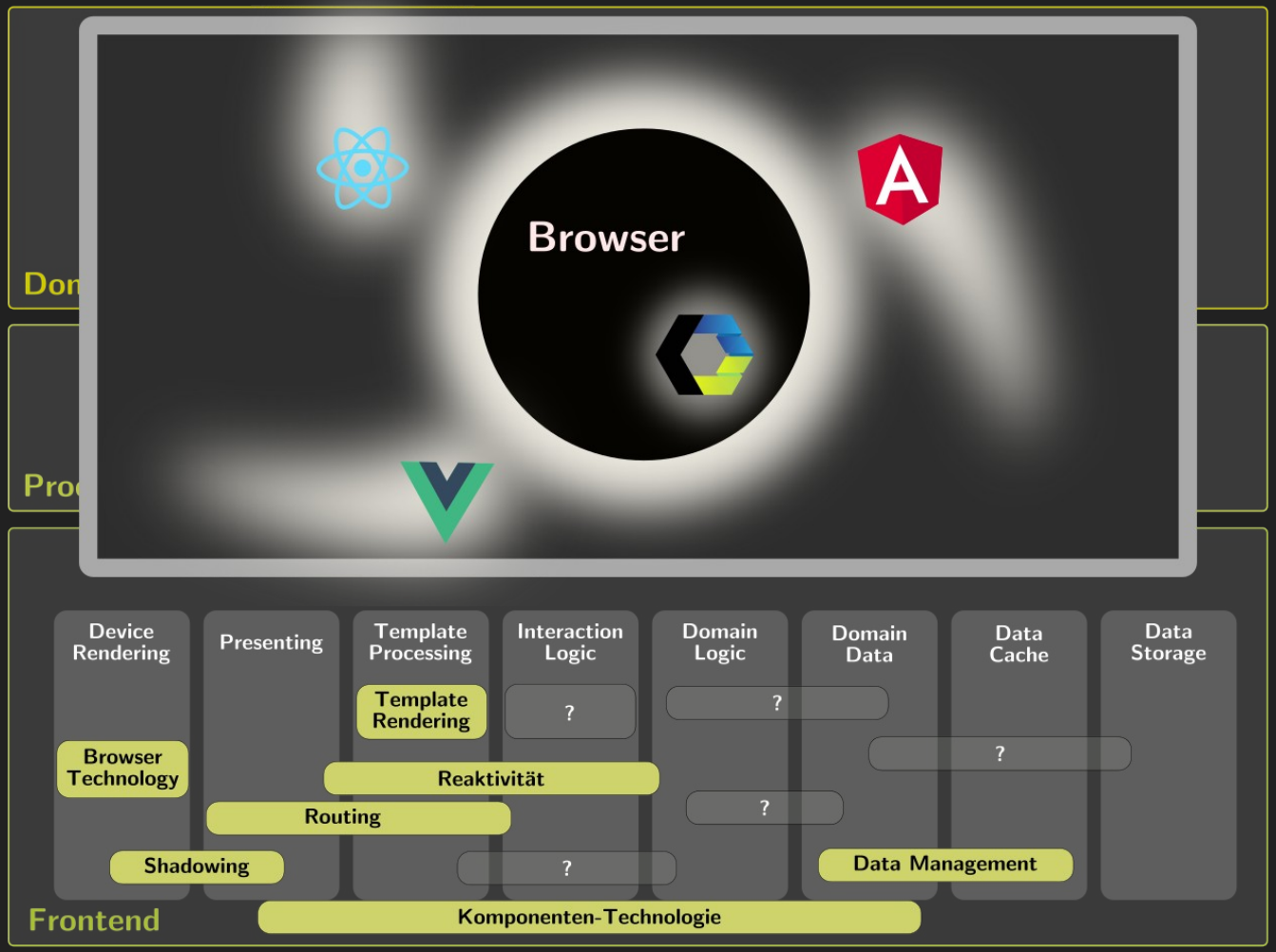
Code Organization

- hilft Code übersichtlicher und nebeneffektfreier zu organisieren

Data Store

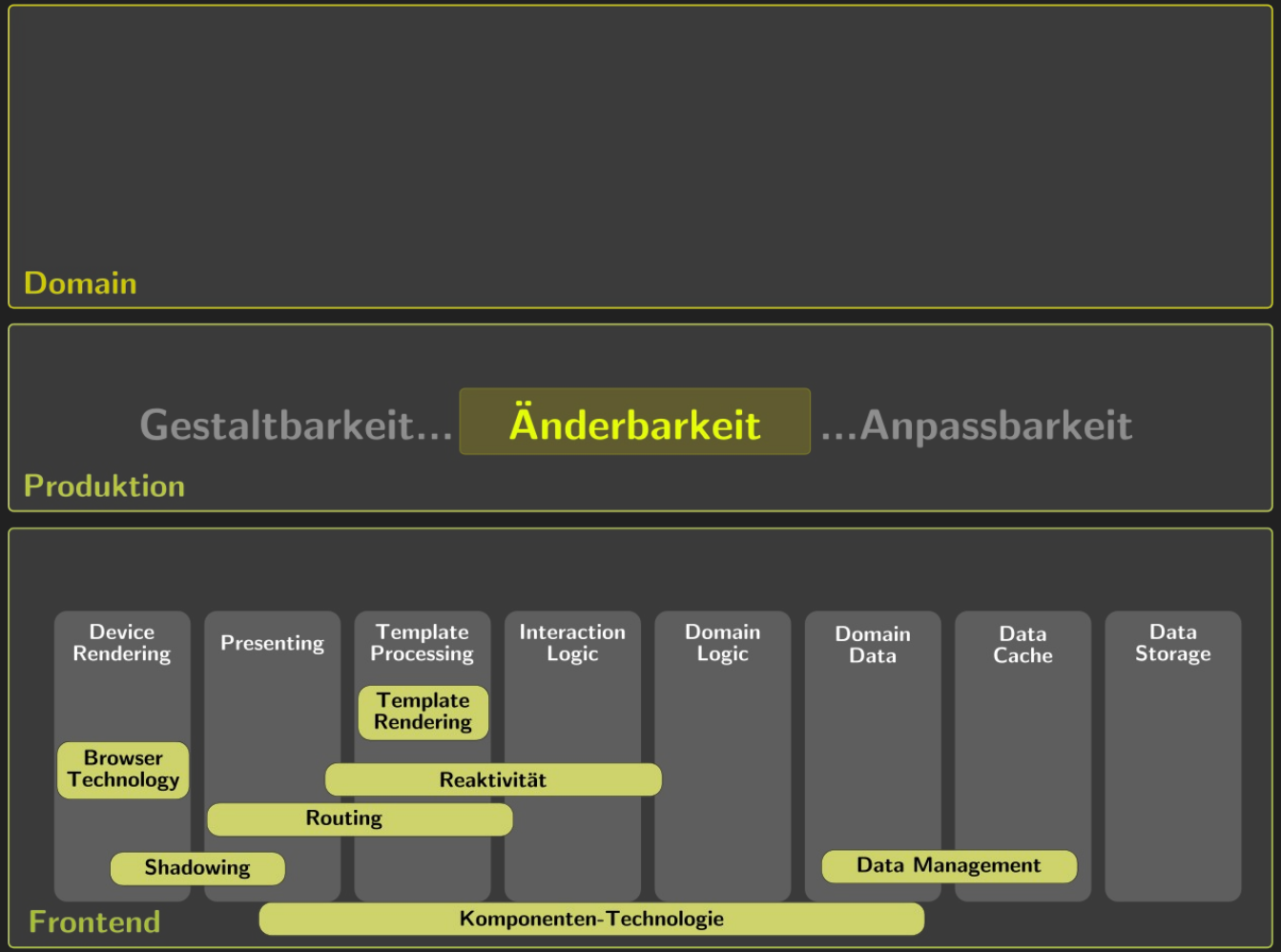
- koordiniert Zugriffe auf die Daten

Evolution



- Web-Components: bewährte Framework-Features werden vom Browser assimiliert
- Was fehlt?
- Was kommt?

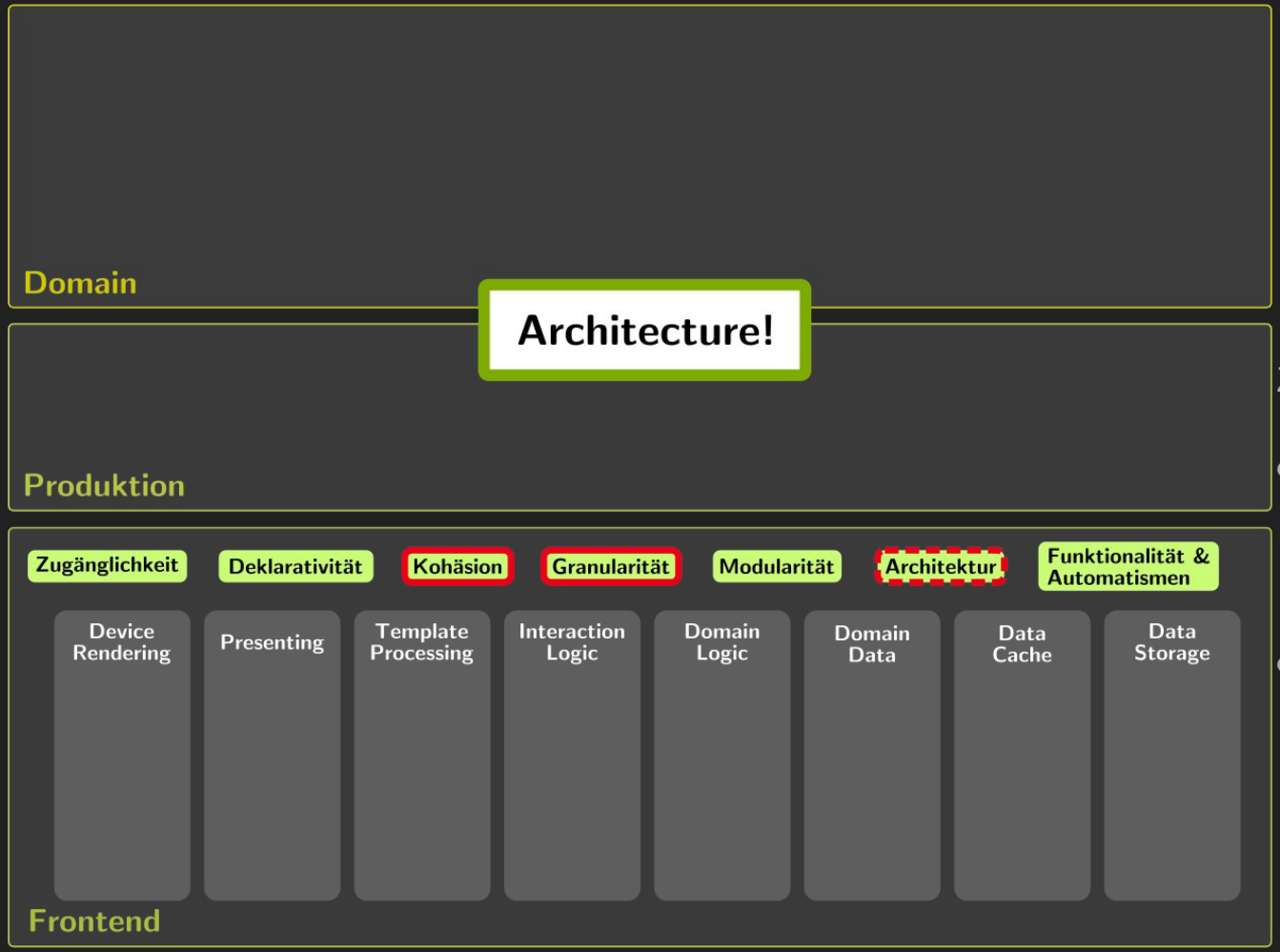
Was ist offen?



- Zuschnitt der Komponenten
- Konkrete Architektur
- Ordentlich (Modularität) und nachhaltig (Architektur)

Architektur

Architektur



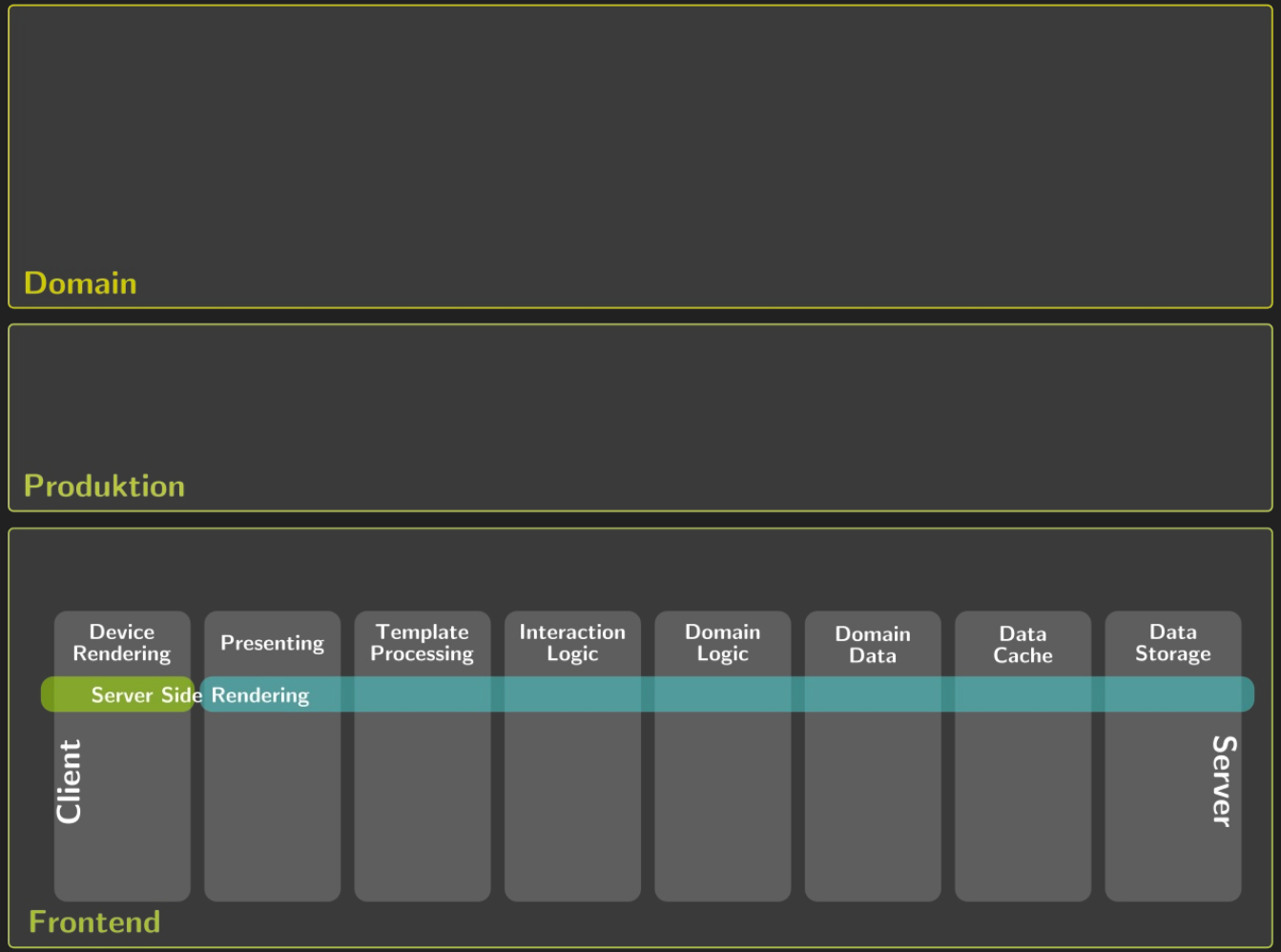
Zwei Ebenen:

- **Anwendungs-Architektur**: Zuschnitt und Verteilung von System-Komponenten (grob, welche Schichten)
- **Komponenten-Architektur**: innere Struktur der System-Komponenten (kleinteilig, welche Bauteile)

Anwendungs-Architektur
ist hier Frontend-Architektur

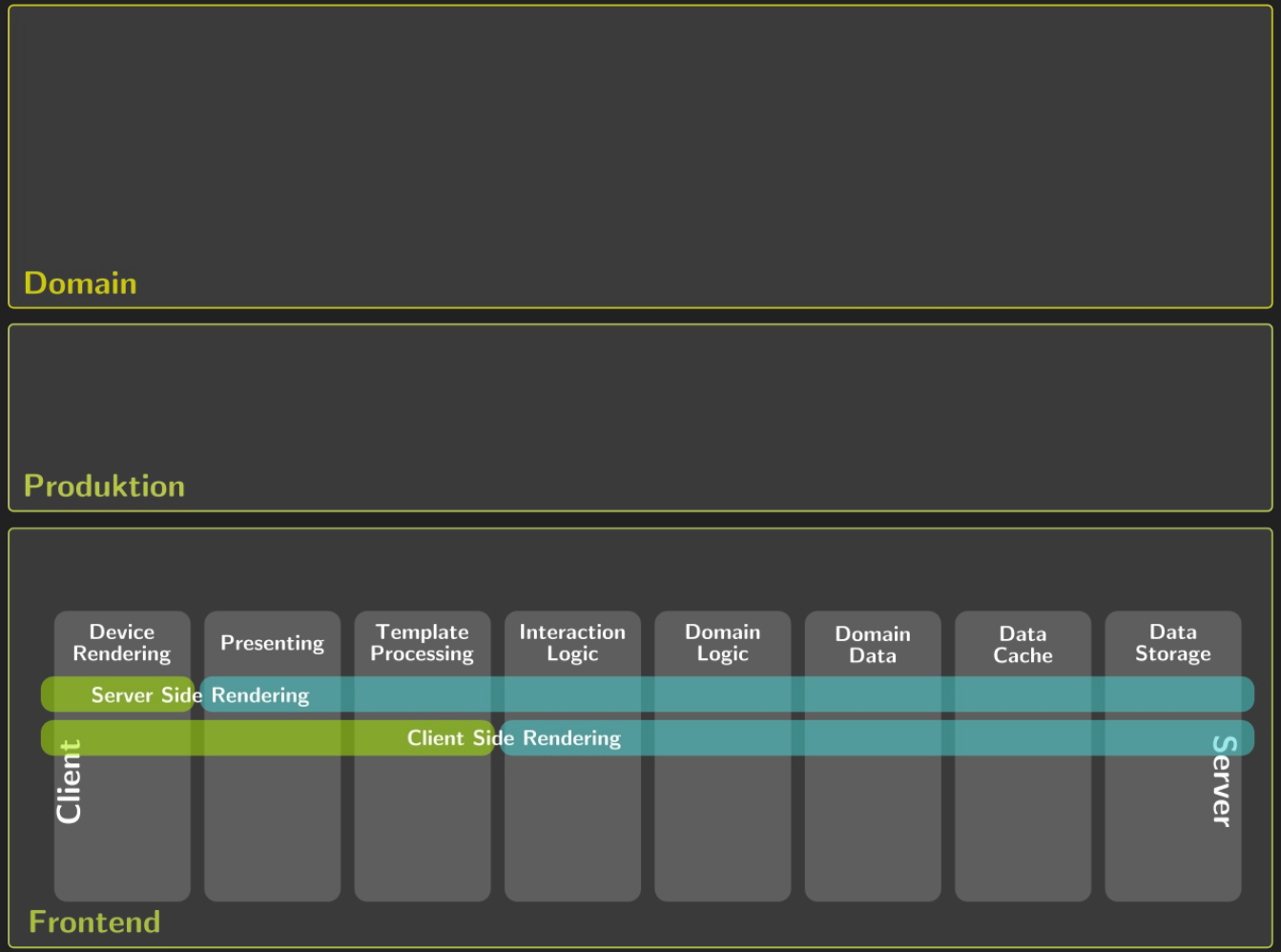
...oder?

Shifting Layers



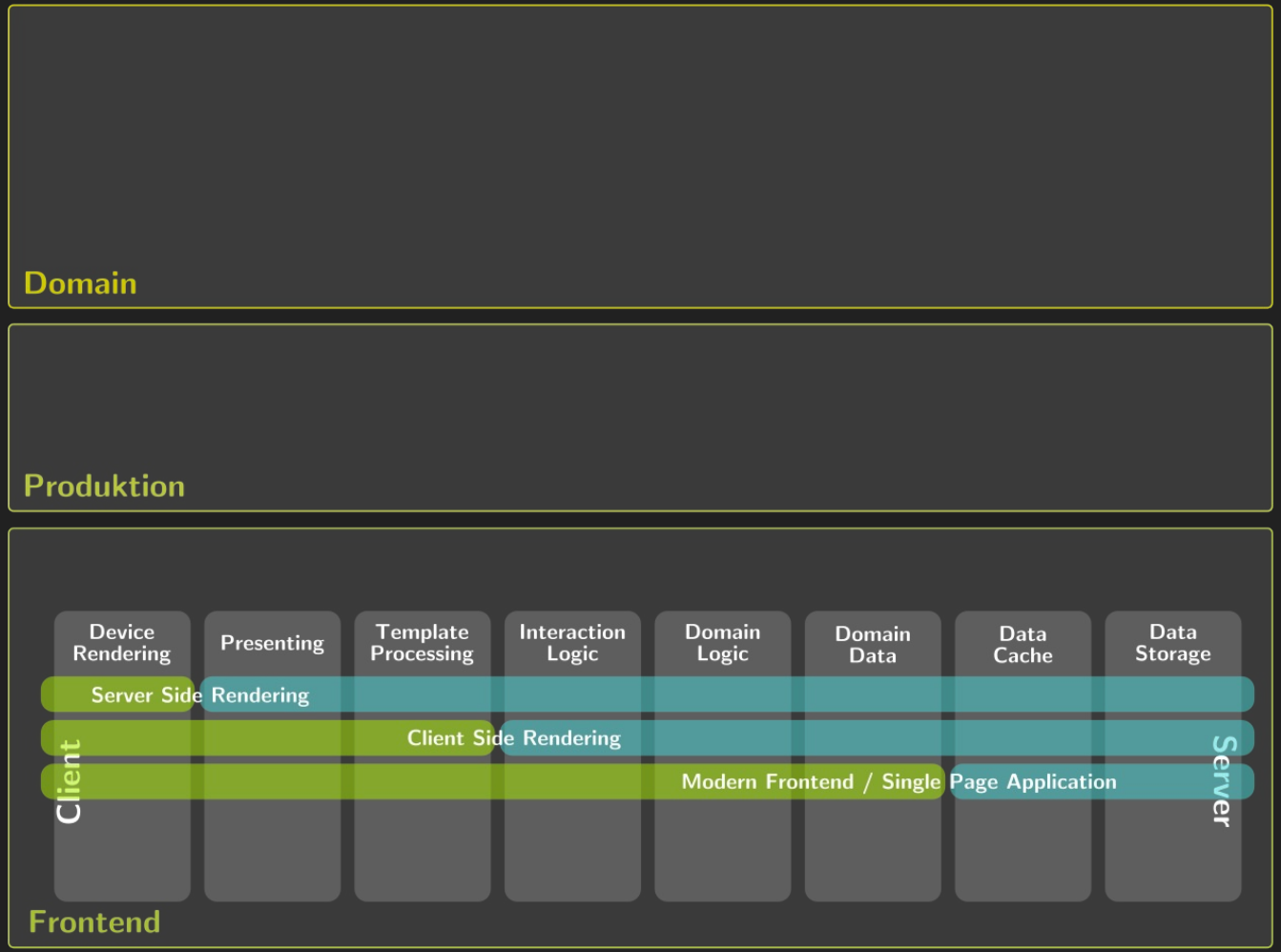
Server Side Rendering

Shifting Layers



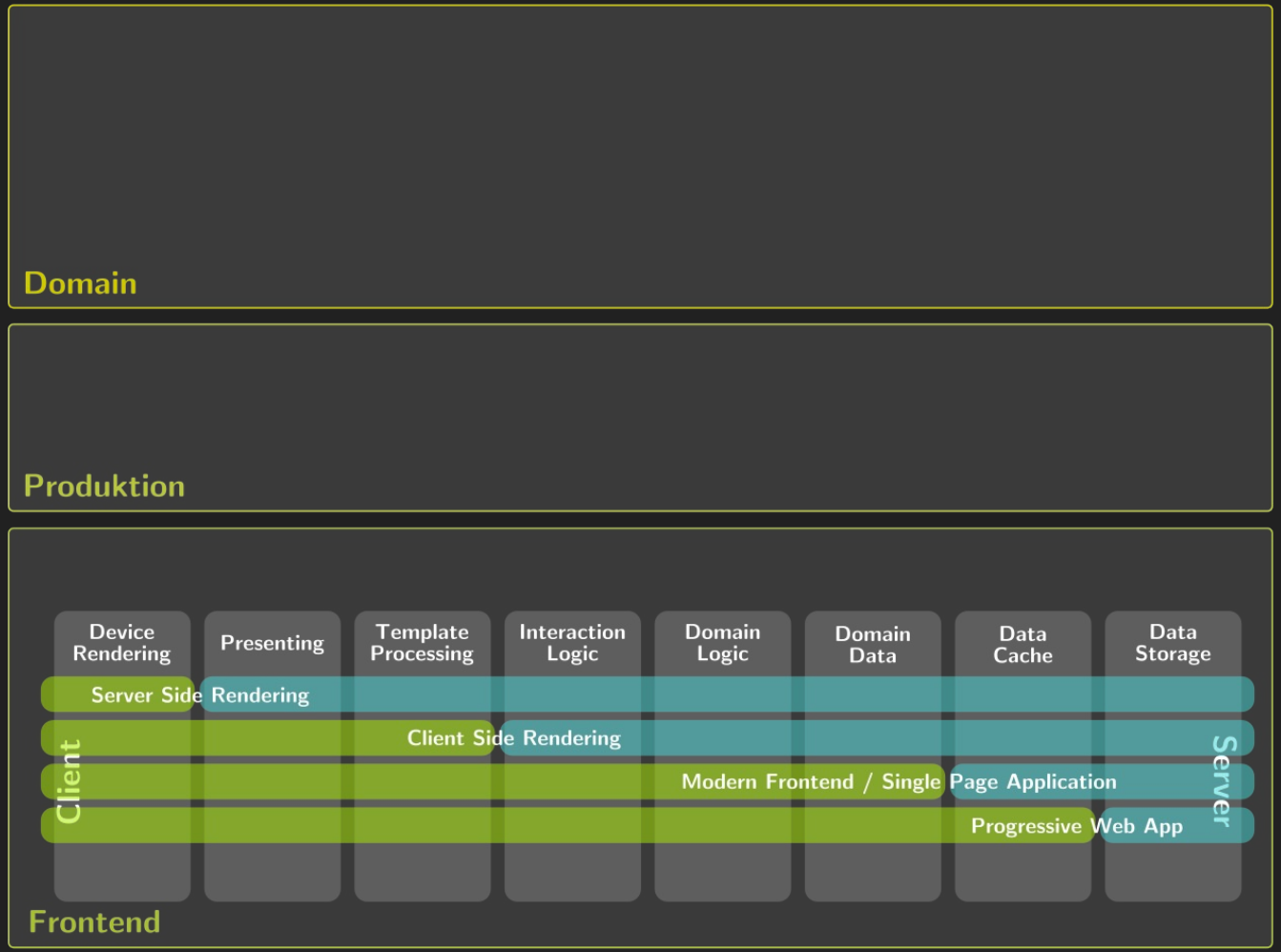
Client Side Rendering

Shifting Layers



Single Page Application

Shifting Layers

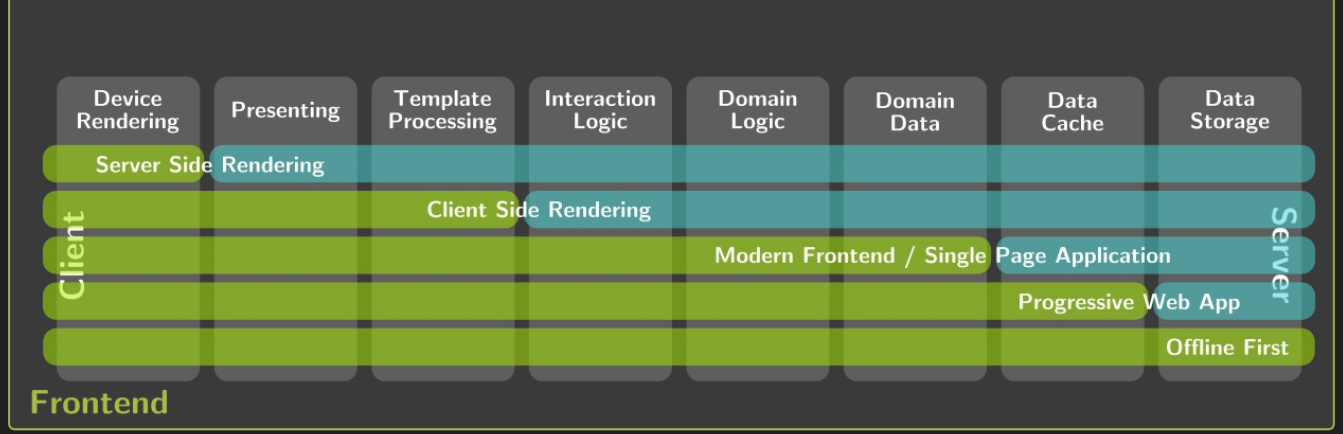


Progressive Web App

Shifting Layers

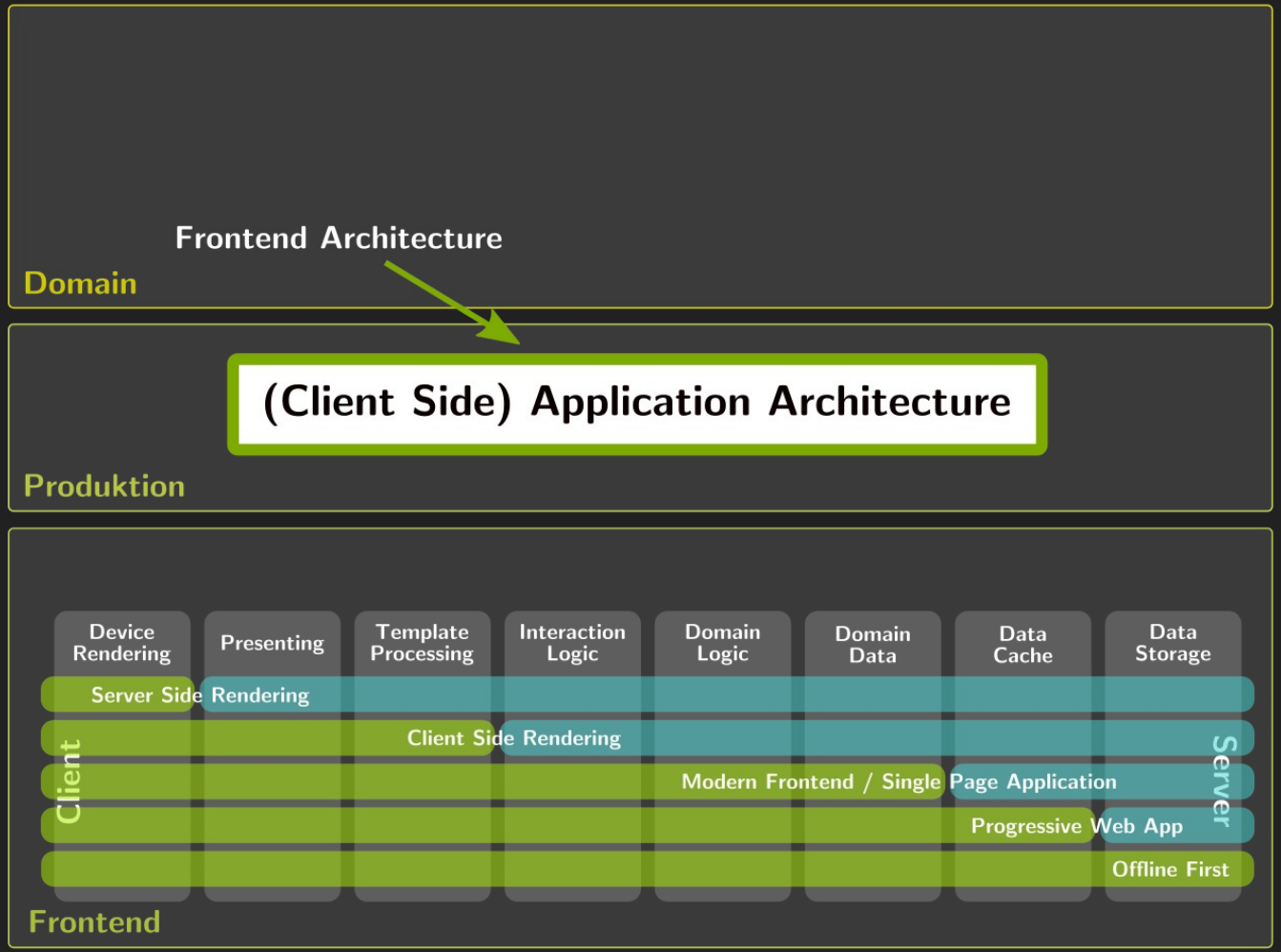
Domain

Produktion



Offline First

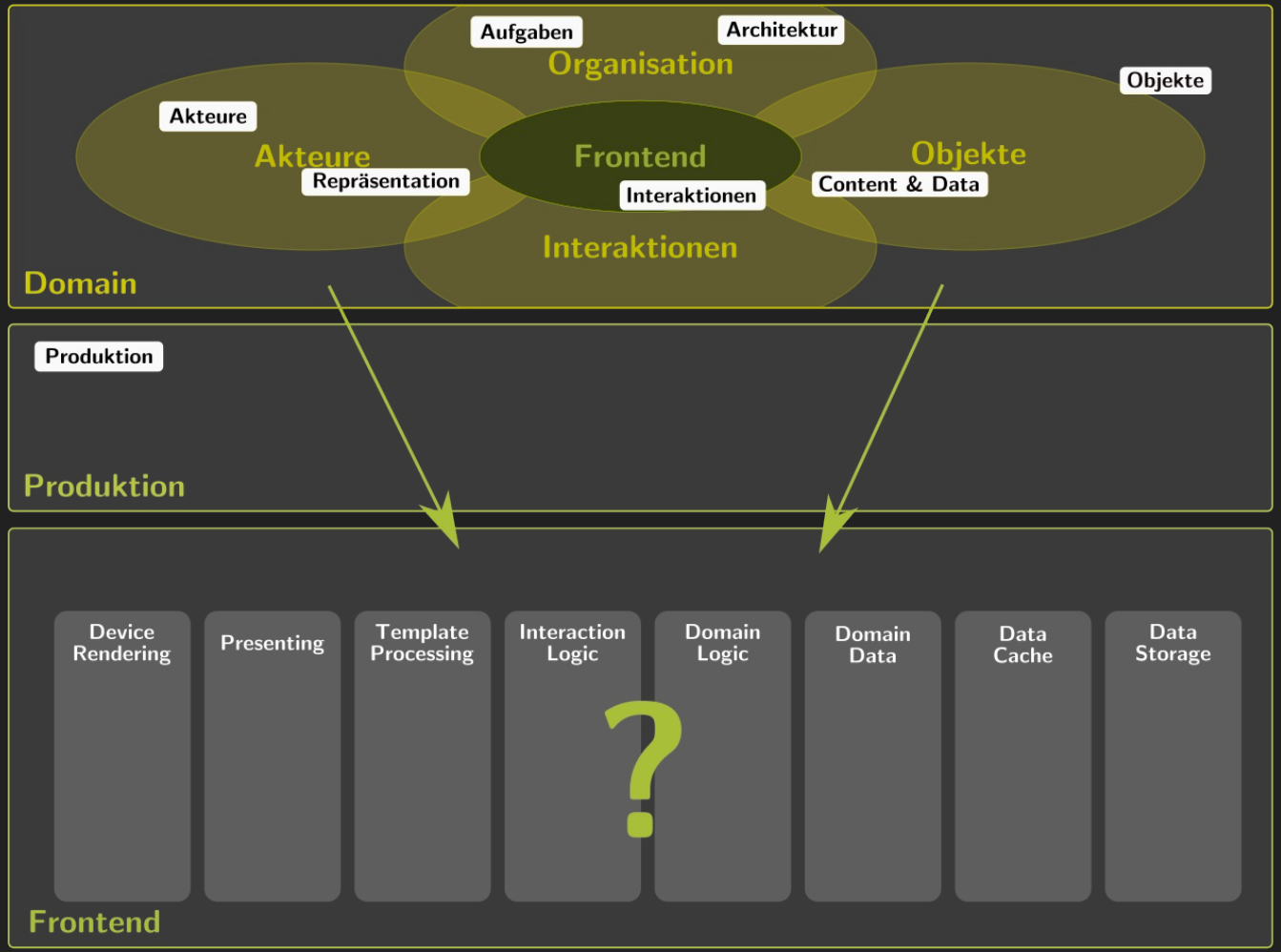
Shifting Layers



Frontend-Architektur ?

Fallweise vollständige Anwendungs-Architektur !

Komponenten-Architektur



Komponenten-Architektur

Auf der Suche nach:

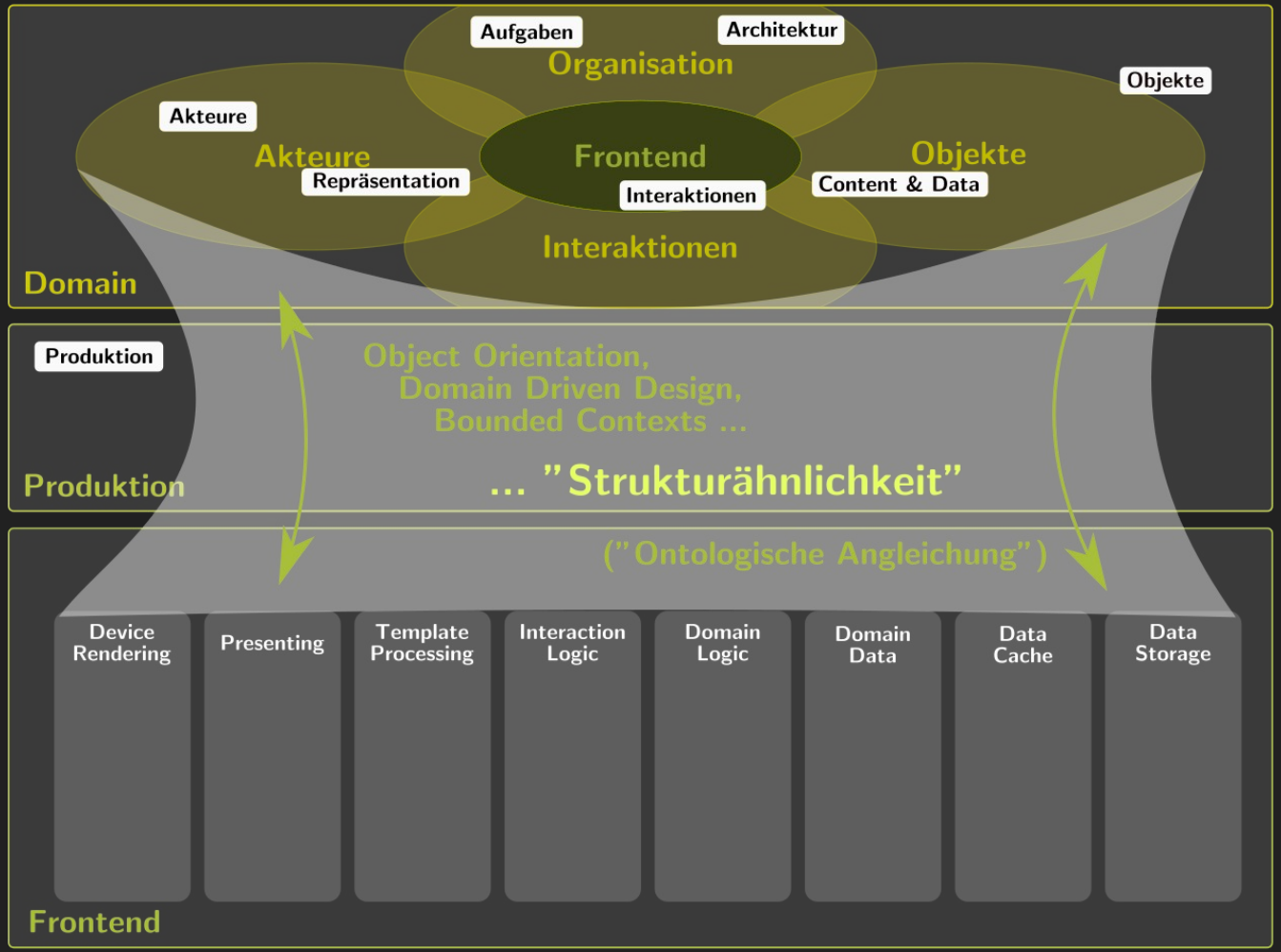
- **Struktur** und **Bauteile**
- **leicht** ändern können, aber nicht müssen
- **nachhaltig** produzieren
- aus relevanten **Perspektiven**
- ...die aber nicht deckungsgleich sind

Gute Architektur

Bewährte **Gestaltungsprinzipien** für die Entwicklung guter Architektur

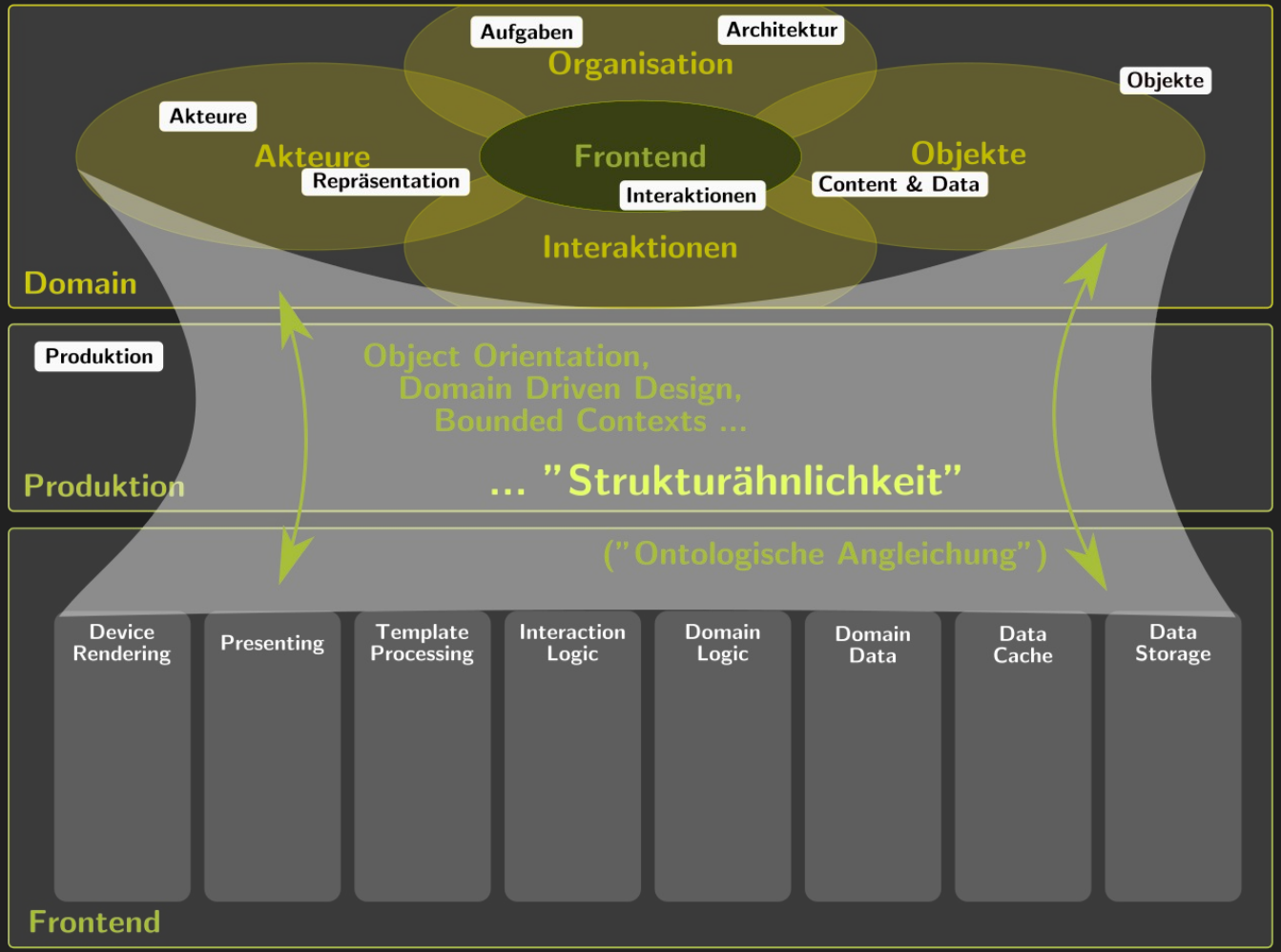
- **Strukturähnlichkeit** zwischen Domäne und System
  - Domänen-Orientierung (DDD, BC)
  - Objekt-Orientierung (OO, != OOP)
- **Ausgewogenheit**:
  - Geringe äußere Kopplung
  - Gute innere Kohäsion
  - Separation of Concerns
  - Don’t Repeat Yourself (redundanzfrei)
  - Single Responsibility
  - Angemessene Komplexität

Komponenten-Architektur



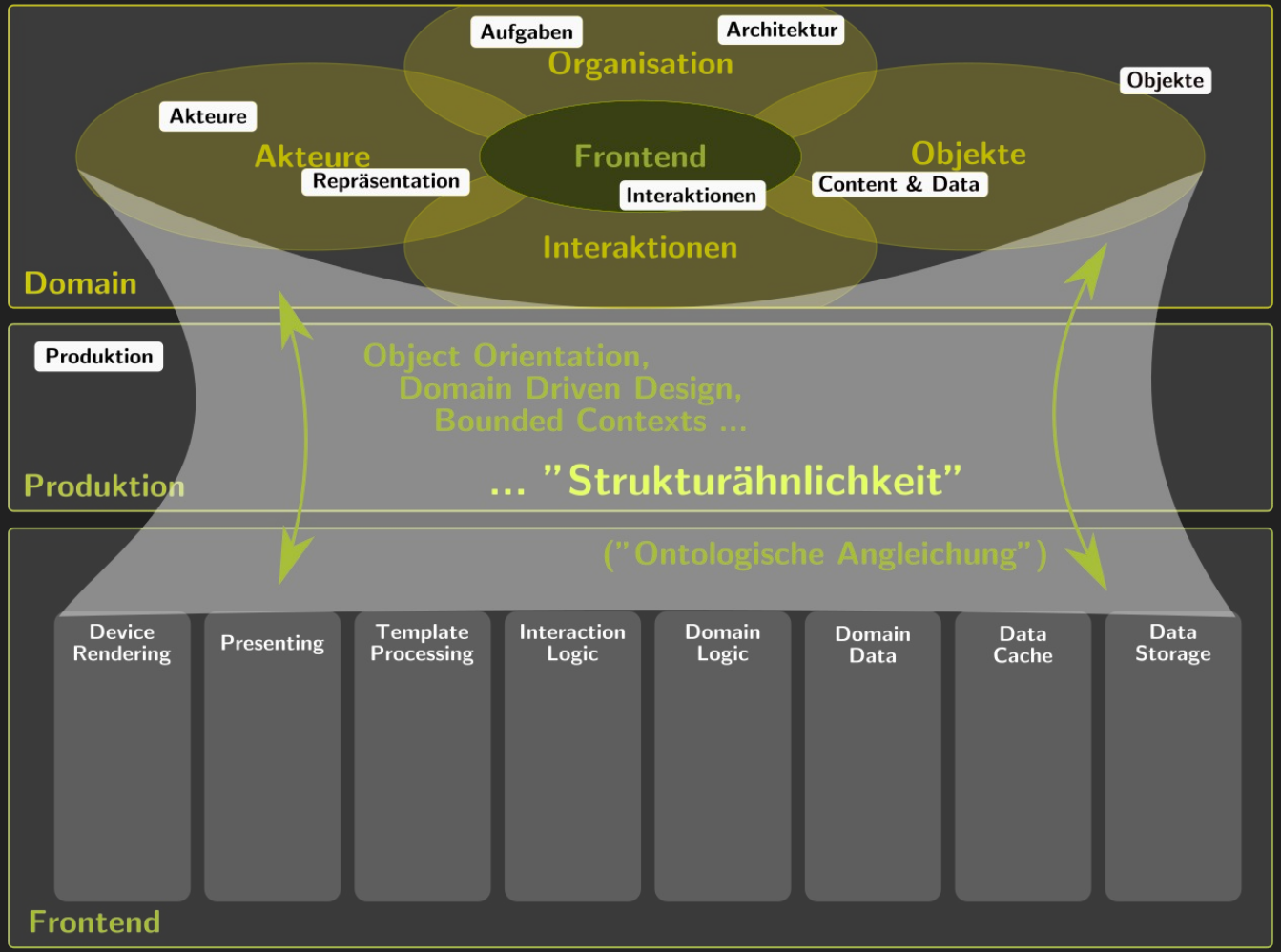
- Strukturähnlichkeit zwischen Domäne und System
- Was genau ist die Domäne?

Komponenten-Architektur



- Strukturähnlichkeit zwischen Domäne und System
- Was genau ist die Domäne?
- Offensichtlich können all diese Aspekte (fallweise) relevant sein
- naives Scaffolding muss (i.d.R.) scheitern

Komponenten-Architektur



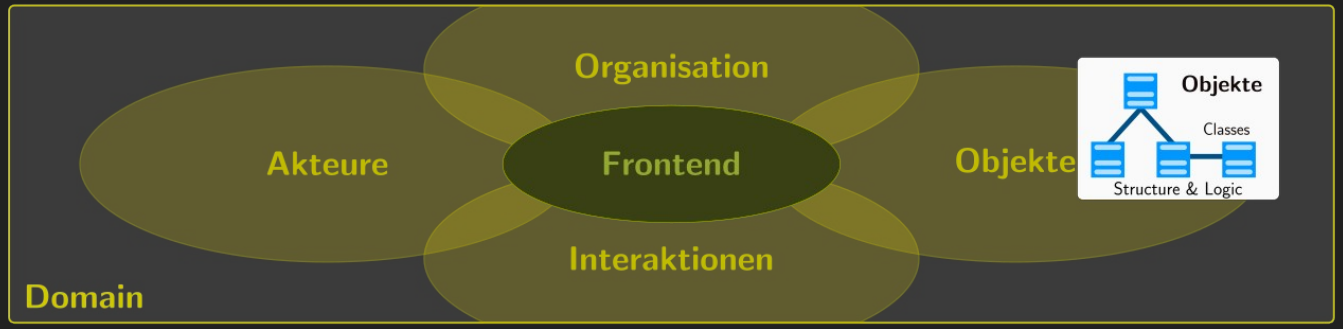
- Ziel im Auge behalten: Usability der Komponenten für betroffene Stakeholder
- keineswegs in jedem Projekt immer alle

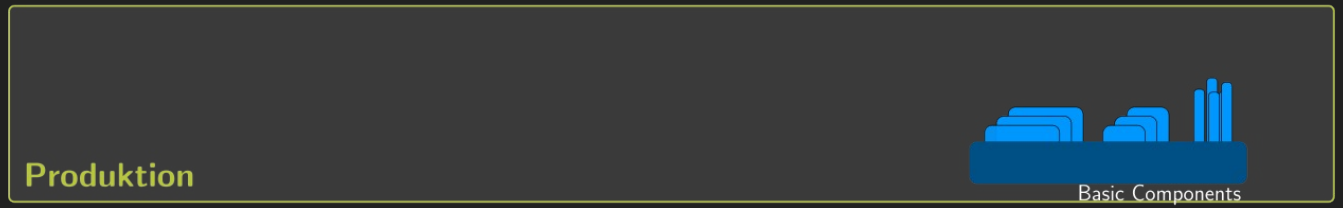
Sustainable Components

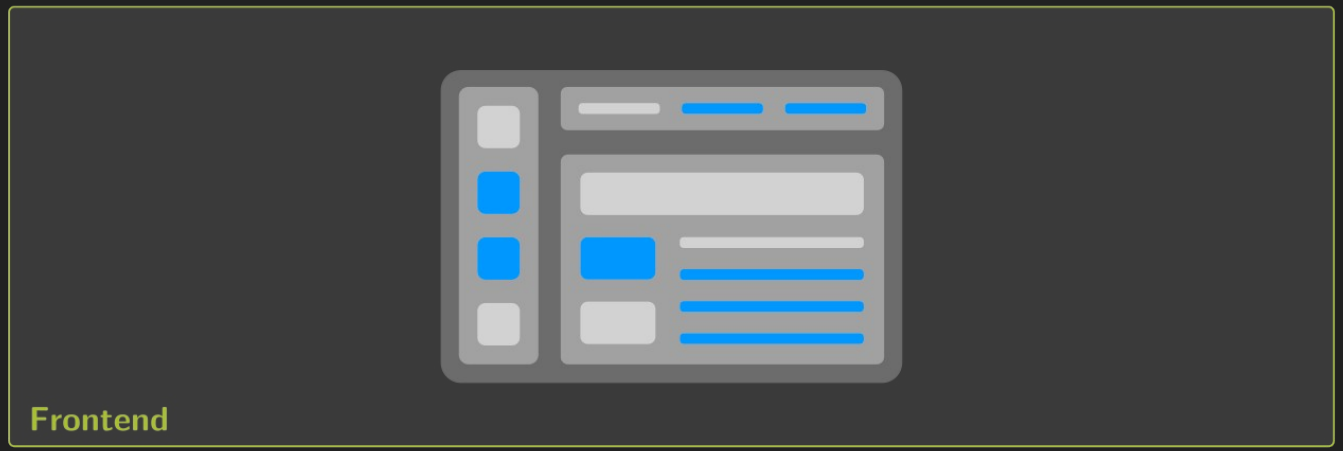
Sustainable Components

- Konsolidierte Zusammenstellung anhand Vorüberlegungen (Strukturähnlichkeit usw.)

  “Best Of” aus mehreren Projekten
  Entertainment/Hospitality Systeme (Angular)
  Intranet/Mitarbeiter-Portal (Angular)
  Versicherungs-Portal, Produkt-Katalog, WaWi (ExtJS)
  Generisches PWA-Frontend (Angular/Vue.js)



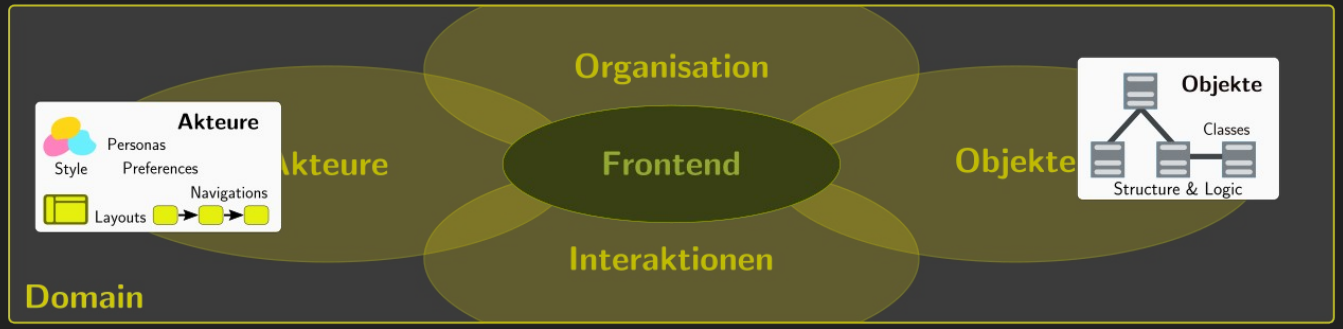


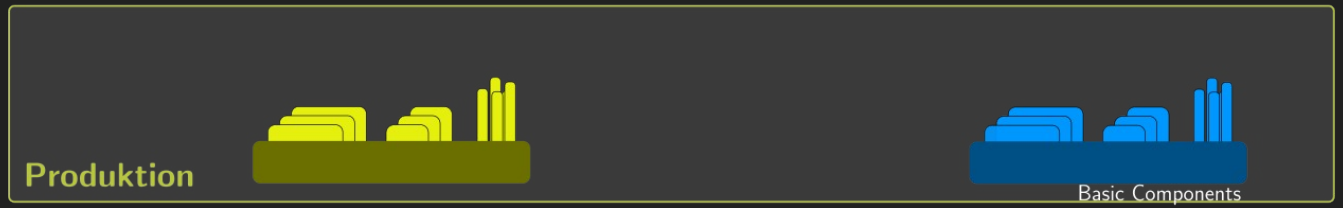


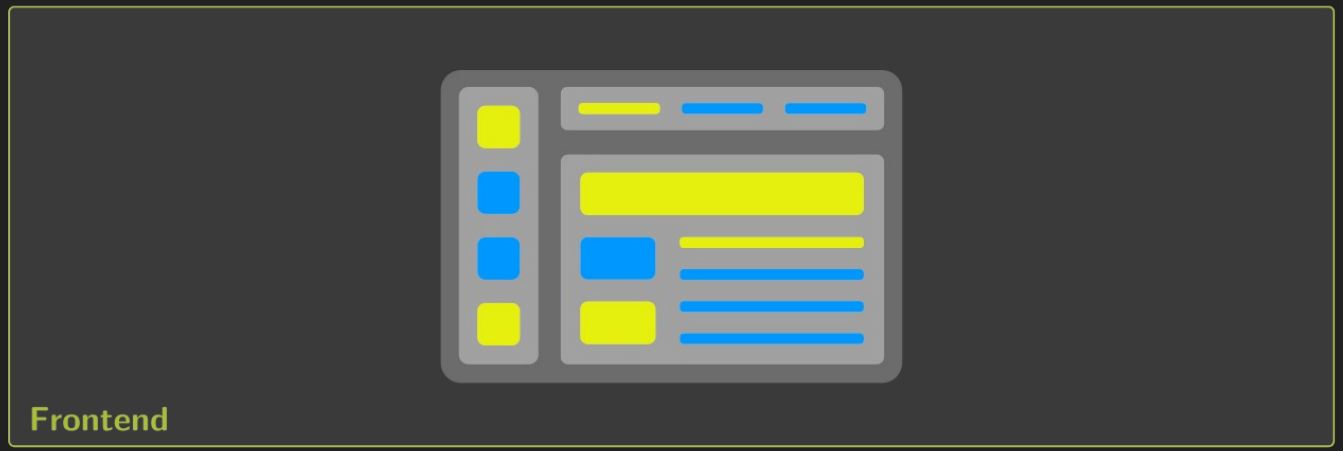
Grund-Elemente

- Bauteile für CRUD-Dialoge
- Präsentation
- Selektor
- Validierung

- lassen sich durchaus automatisiert erzeugen
- mittels Scaffolding oder Generierung



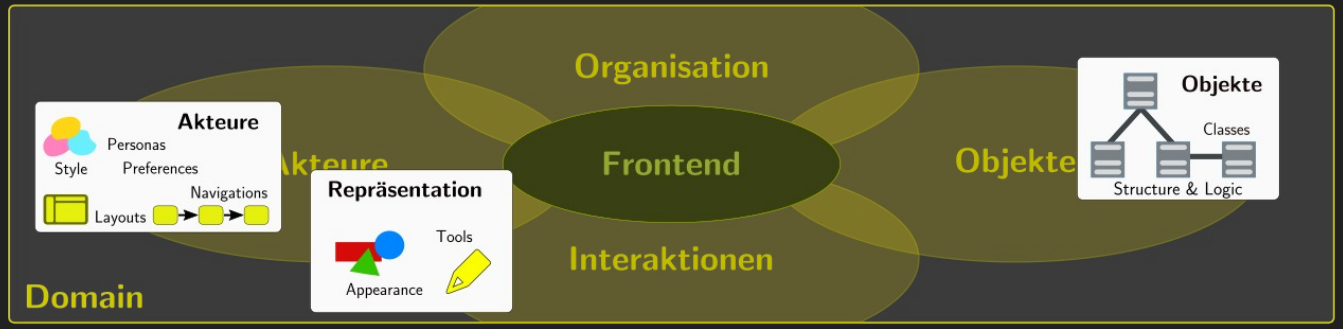


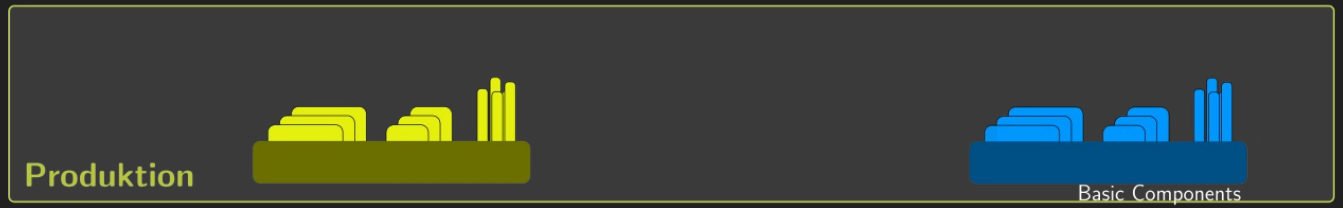


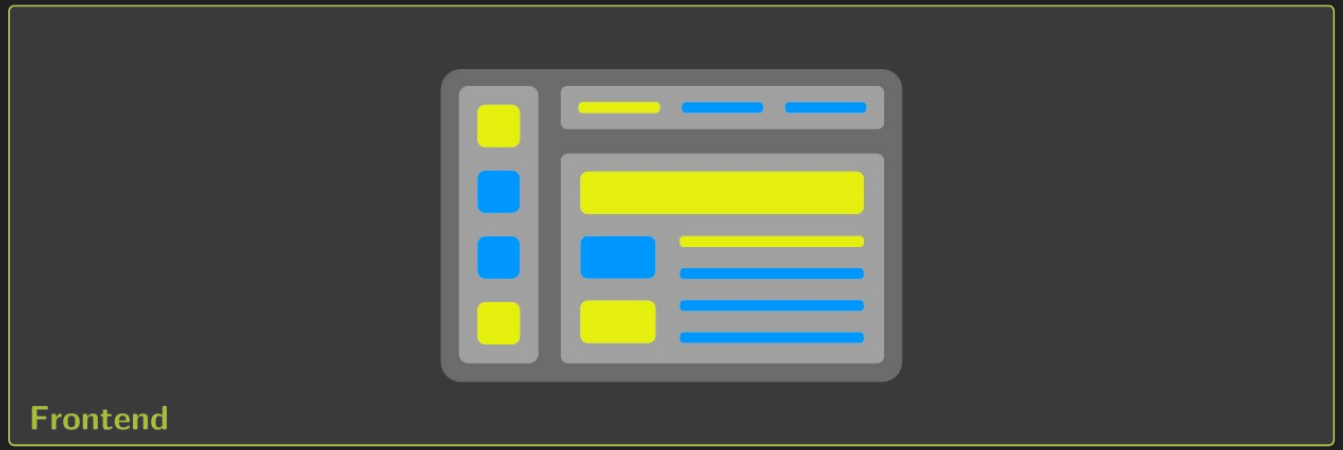
Grund-Elemente

Individuelle Komponenten

- müssen manuell gebaut werden
- teilweise aber wiederverwendbar (Suche, Selektion, u.ä.)
- leichtgewichtiges “Akteur-Modell” kann hilfreich sein



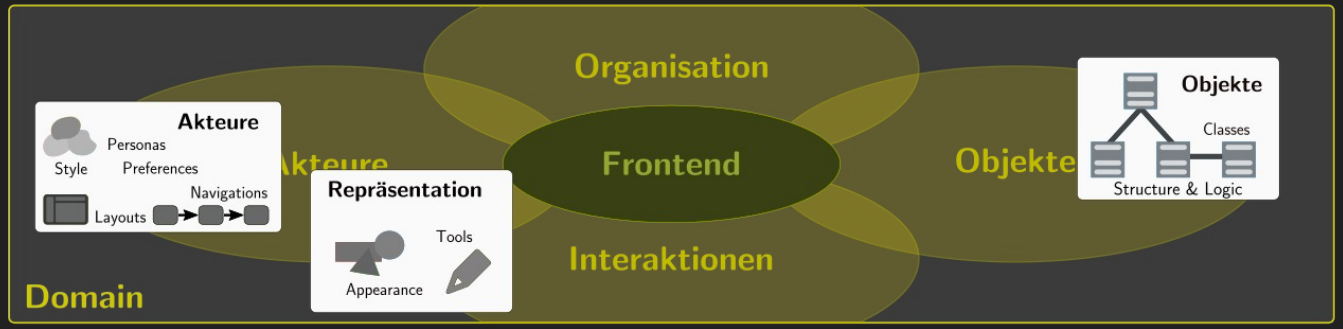


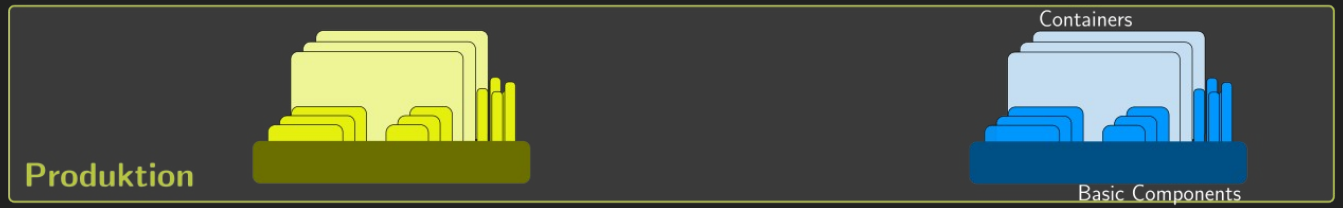


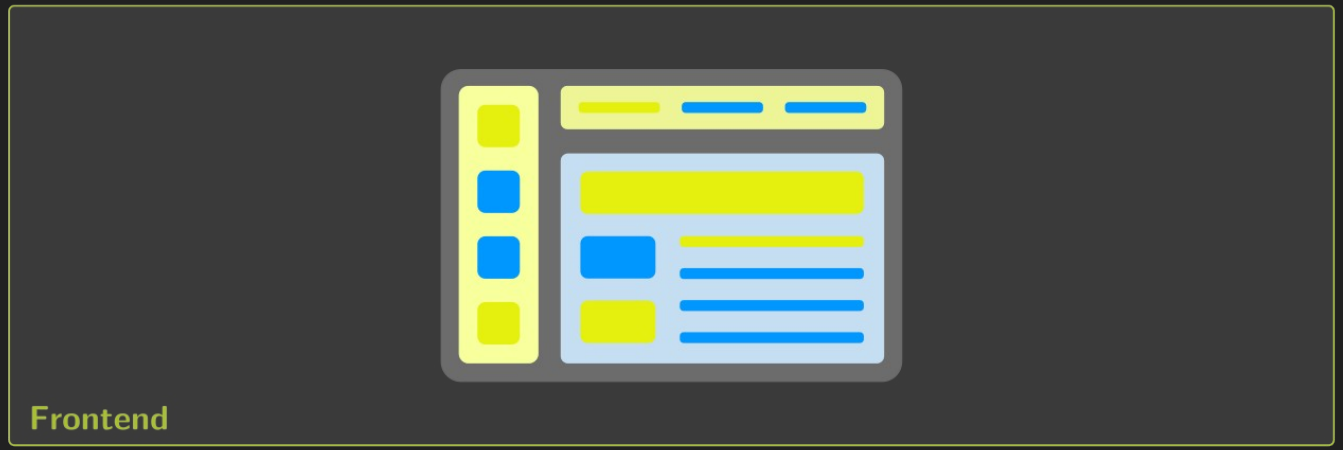
Grund-Elemente

Über primitive Felder hinausgehend:

- Verbreitete Elemente wie Color-Chooser
- Kreative Visualisierungen von Objekten und Werkzeugen
- fallweise unterschiedlich wiederverwendbar





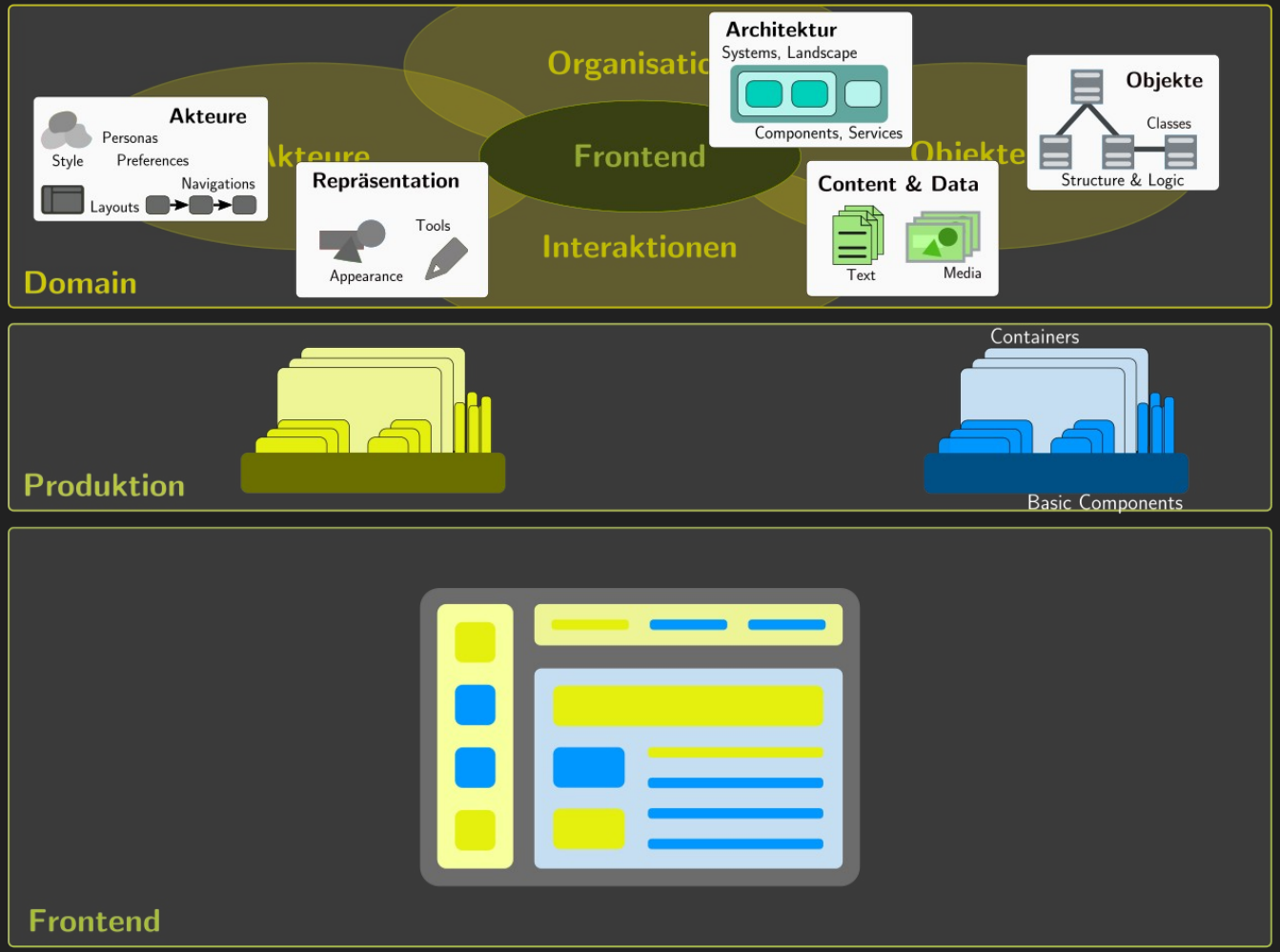


Grund-Elemente

Container

- hierarchisch zusammengesetzte Seiten, Container oder Abläufe
- häufig manuell z.B. Auscheck-Vorgang im Shop

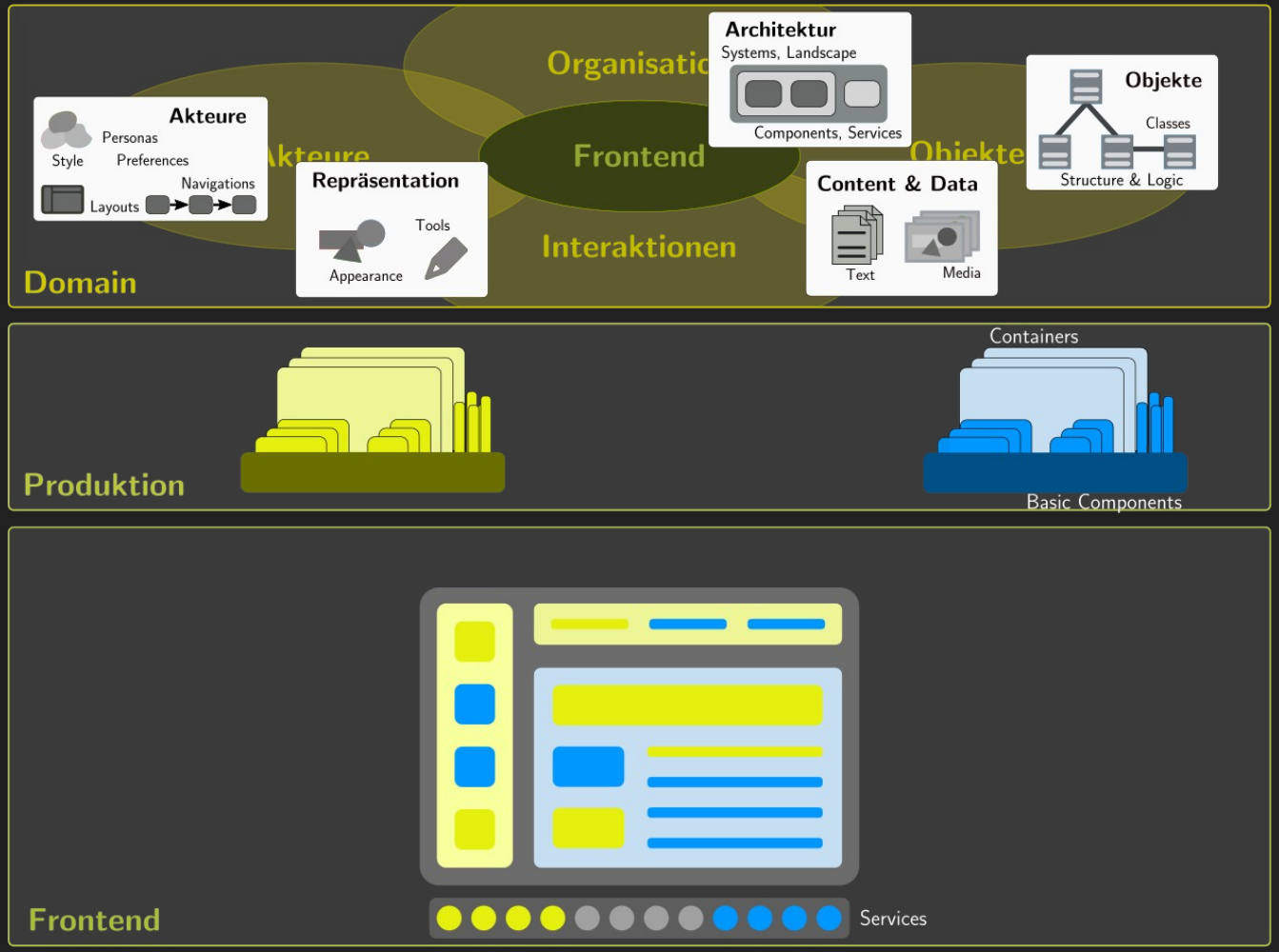
Grund-Elemente



Weitere Quellen neben Objektmodell:

- CMS (Template-basierte Seiten)
- Architektur-Modell, z.B. Navigations-Elemente lassen sich ableiten

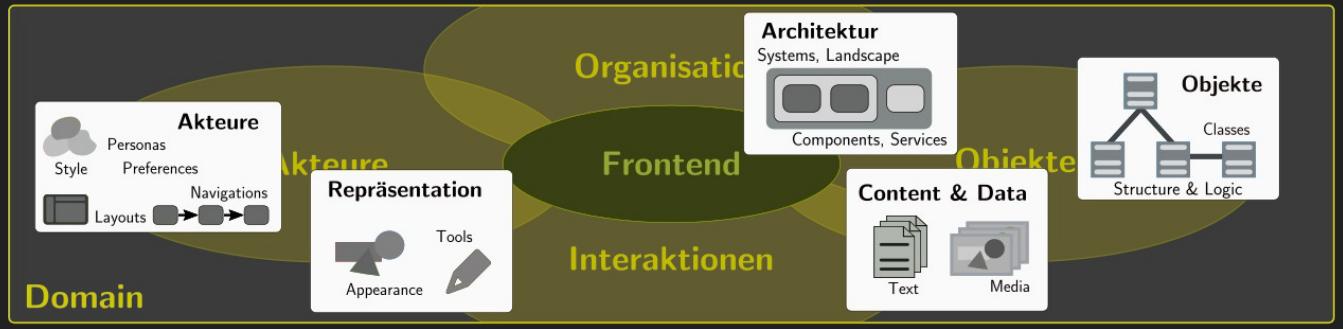
Grund-Elemente

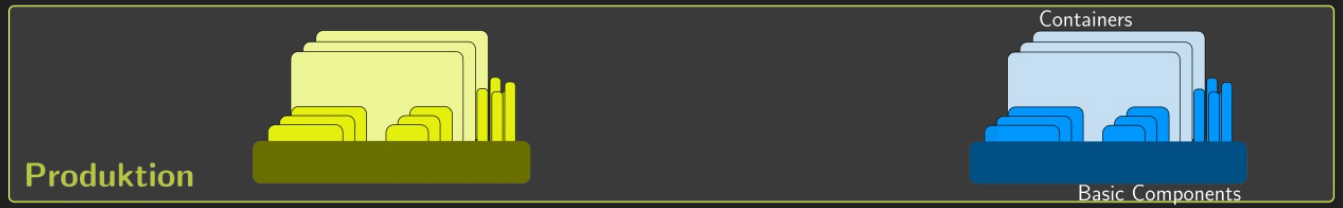


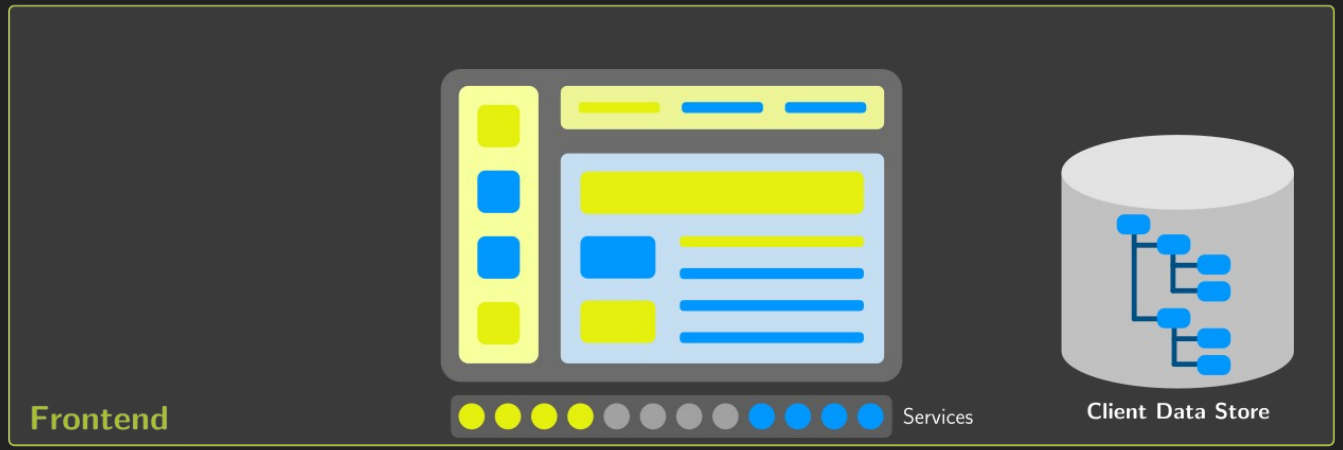
Services:

- CRUD-Adapter, erzeugbar aus Objekt-Modell
- Individuell, manuell programmiert
- Infrastruktur, manuell erzeugt aber universell, z.B. Logging, Session, ...

Daten & Logik

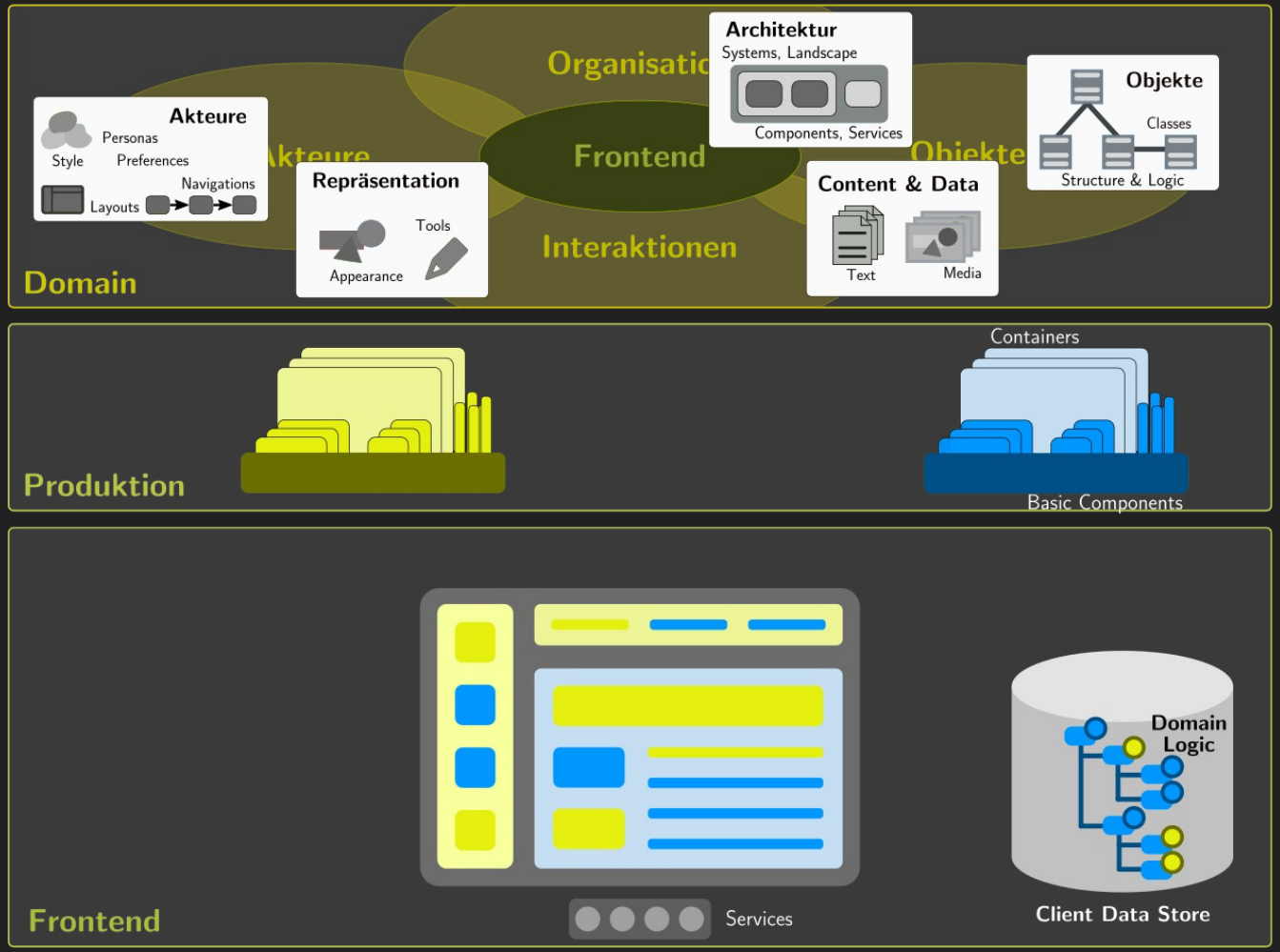






- Services getrennt von Data Store
- Auffindbarkeit, getrennter Life-Cycle, keine Kapselung, keine Vererbung

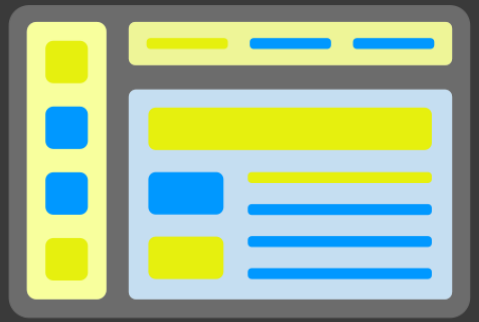
Daten & Logik

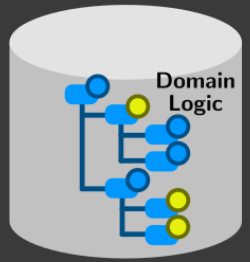


- Zusammenführen in “Domain Layer”
- aus DTOs Objekte machen
- mit Prototypes elegant lösbar

Daten & Logik

```
// define logic
... logic : { get X () { ... methods ... },
              get Y () { ... methods ... }
            }

// make data item an instance of
Object.setPrototypeOf(item, logic[item.type]);

// or
Object.assign(new logic[item.type](), item);
```





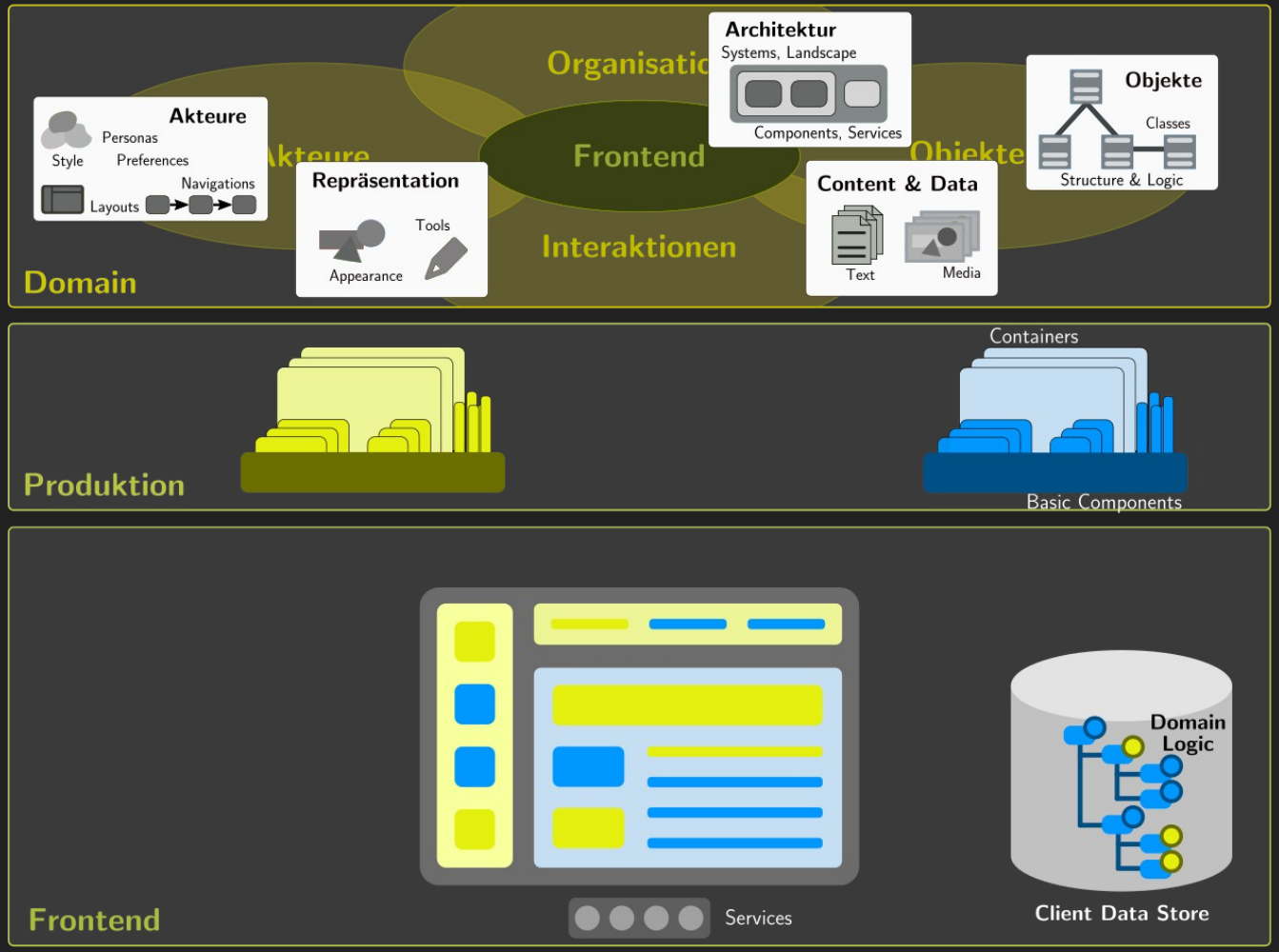
Services

Client Data Store

Frontend

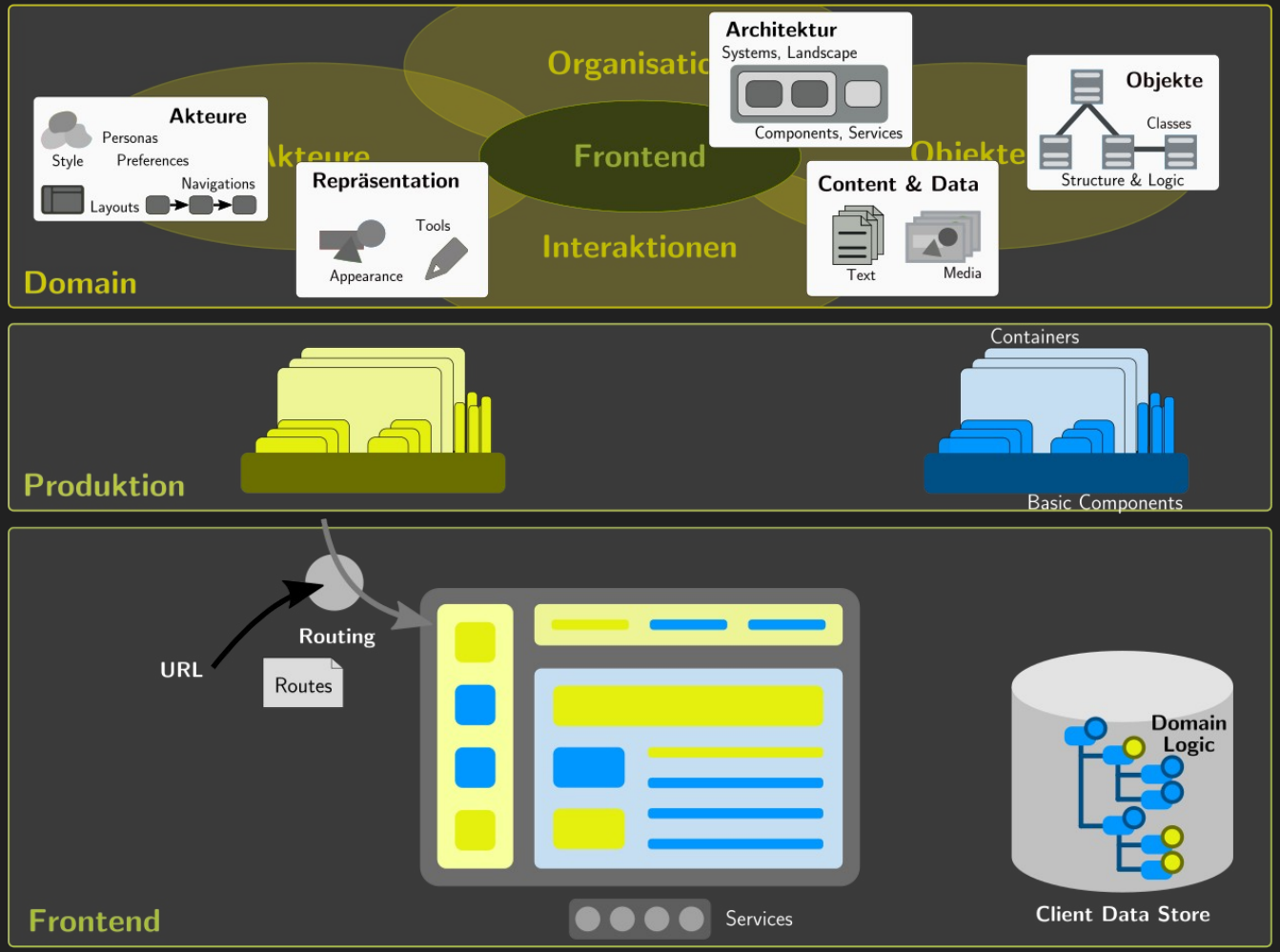
- Zusammenführen in “Domain Layer”
- aus DTOs Objekte machen
- mit Prototypes elegant lösbar

Grund-Baukasten



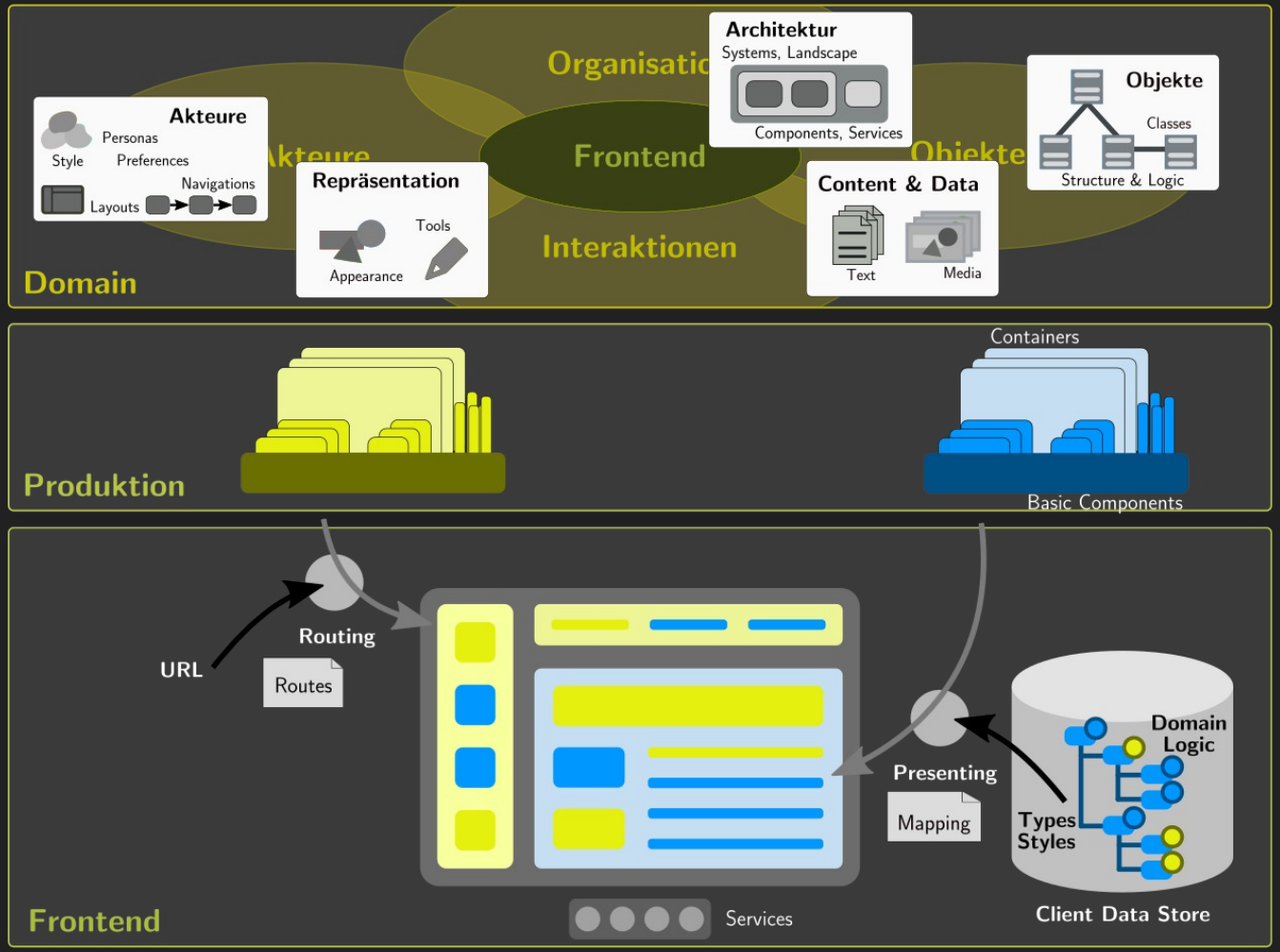
- Grund-Elemente als Baukasten für Frontend-Entwicklung
- u.U. wenig generierbar
- Zusammenbauen von Containern weiterhin mühsam
- Grundelemente nur unterster Ebene des Komponentenbaums
- abgeschlossen, geringe Parametrisierung

Router



- wählt Komponenten anhand der URL aus
- nutzt Routing-Tabelle

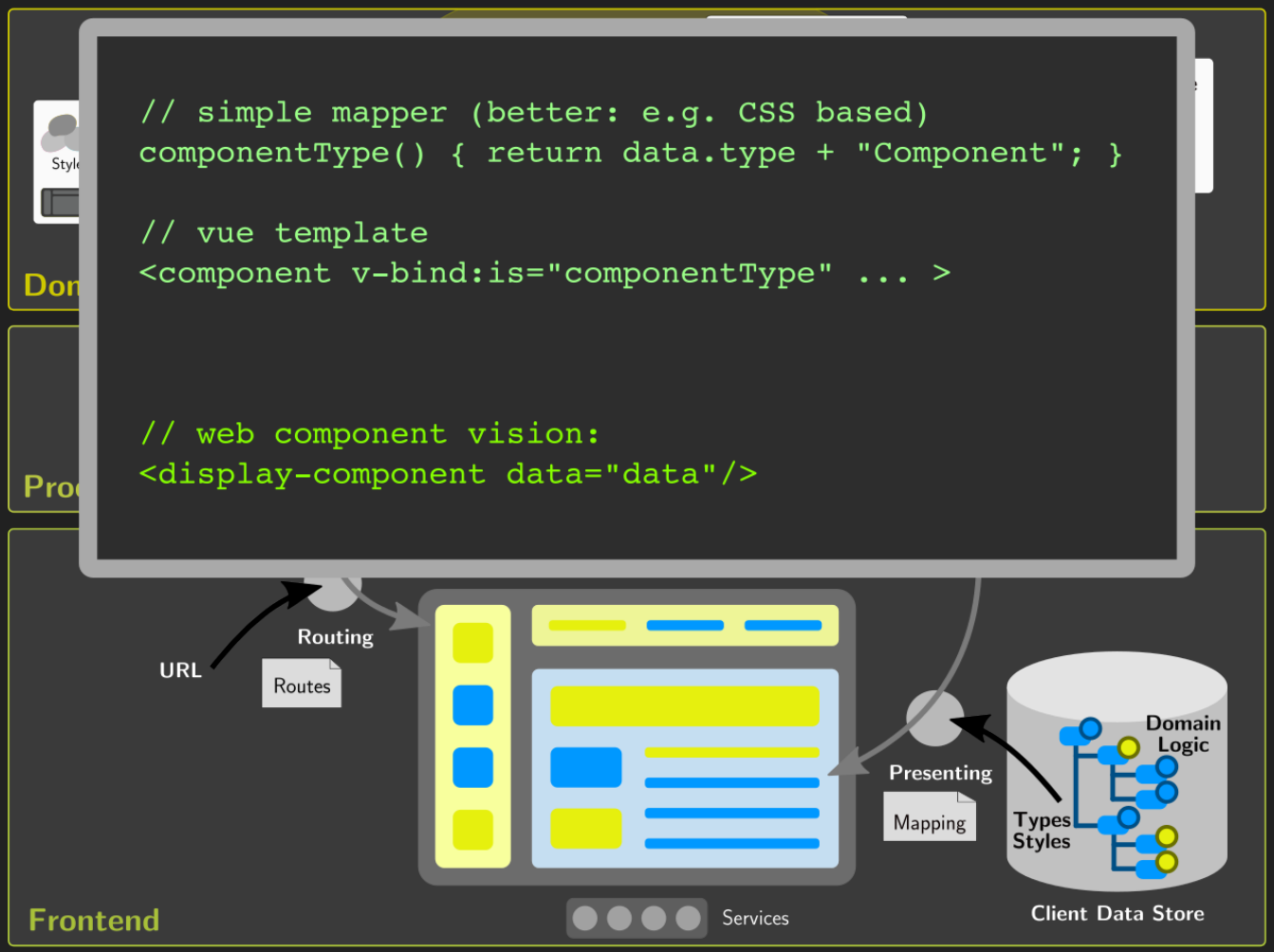
Wiring



- “Daten-Router” wählt Komponenten anhand der empfangenen Daten aus
- in Container dann für Childs verwenden

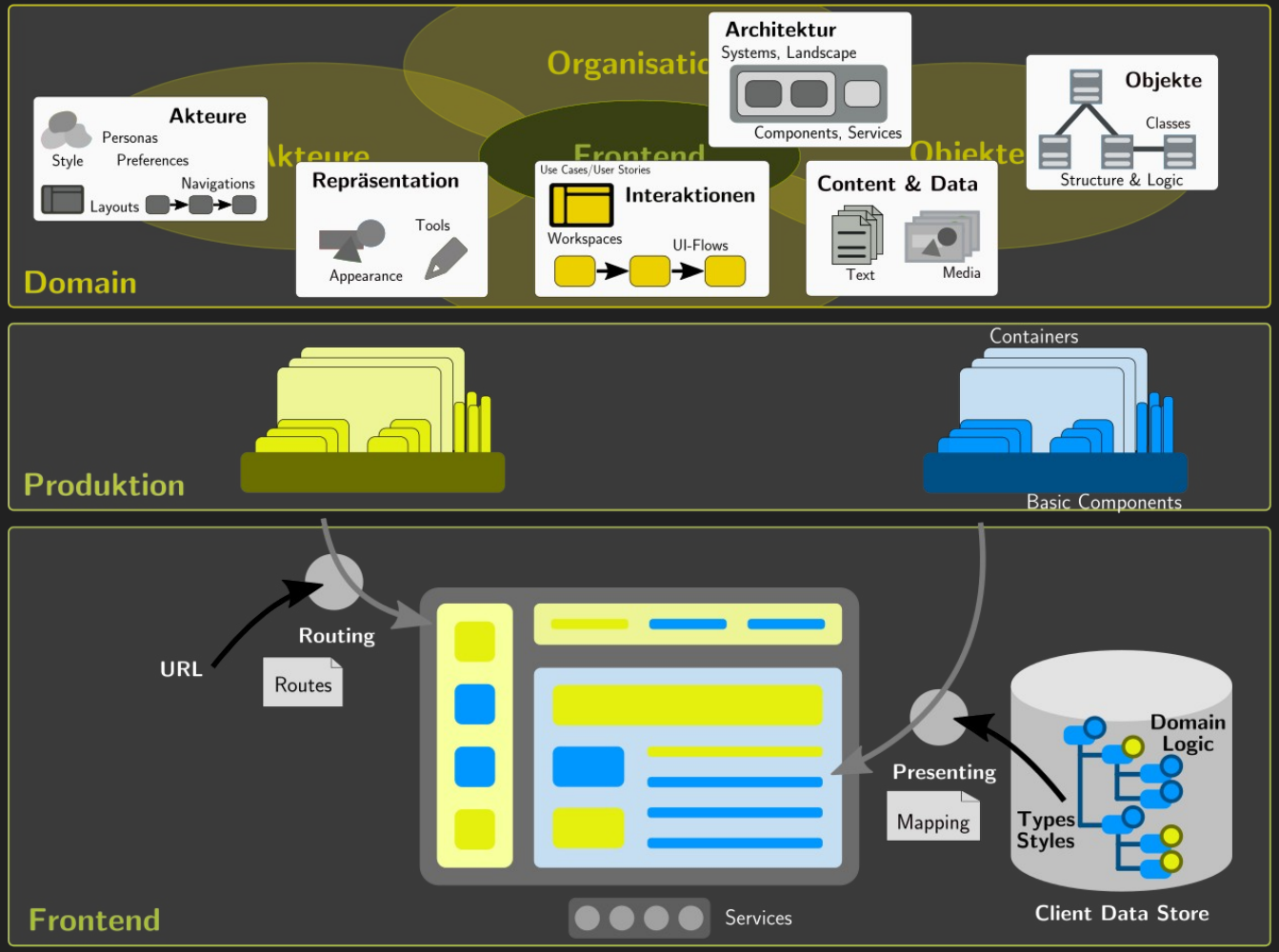
Wiring

```
// simple mapper (better: e.g. CSS based)
componentType() { return data.type + "Component"; }

// vue template
<component v-bind:is="componentType" ... >


// web component vision:
<display-component data="data"/>
```



- “Daten-Router” wählt Komponenten anhand der empfangenen Daten aus
- in Container dann für Childs verwenden
- einfache Mapping-Tabelle, oder sinnvoll bspw. CSS...

Interaktionen

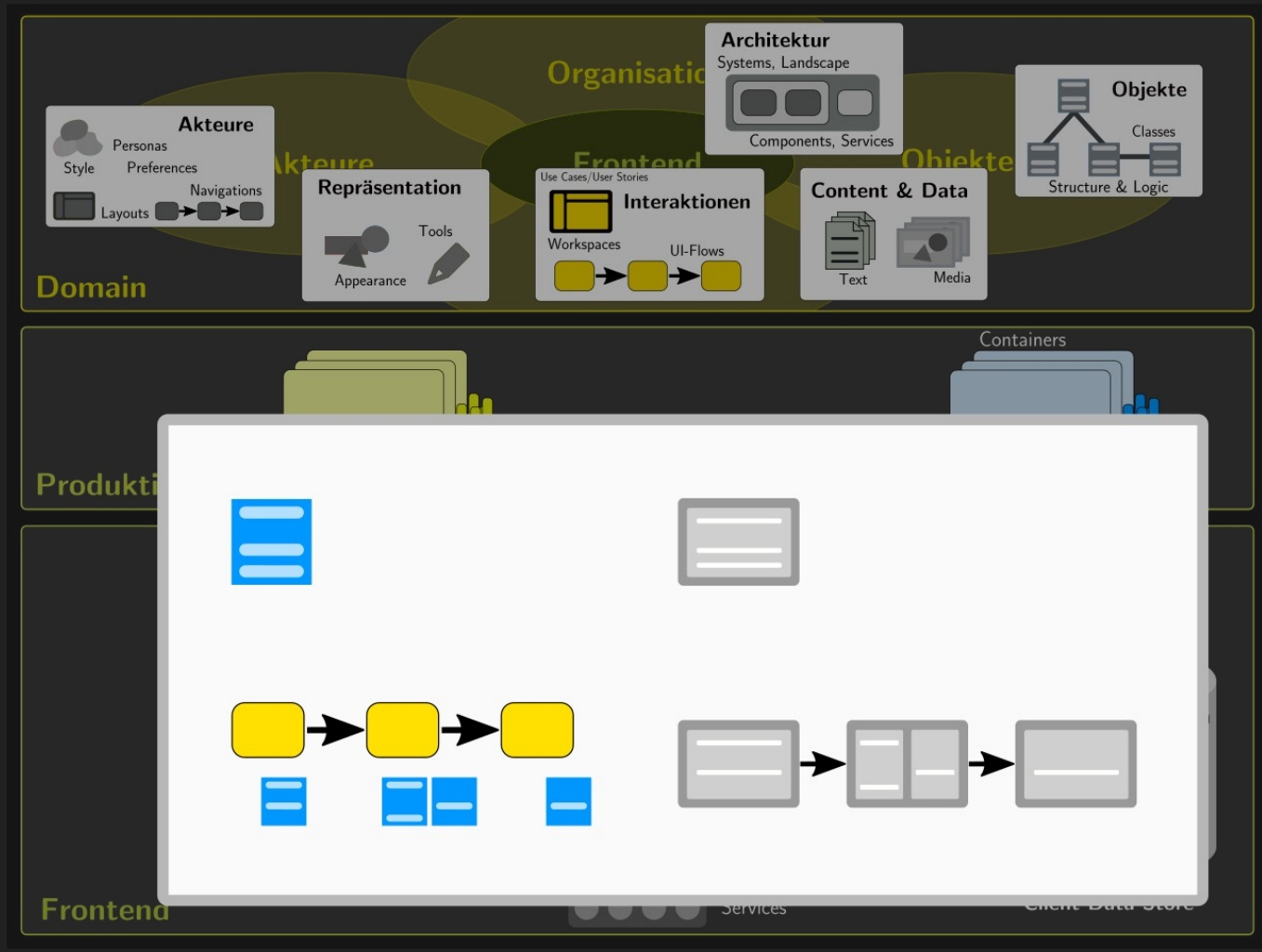


- weitere Quellen aus UI/UX-Design: Page-Flows, Site-Maps, POC-UI
- daraus sinnvolle Grund-Elemente erzeugen
- Interaktionen: verknüpfendes Konzept zwischen Nutzung und Daten

Interaktionen

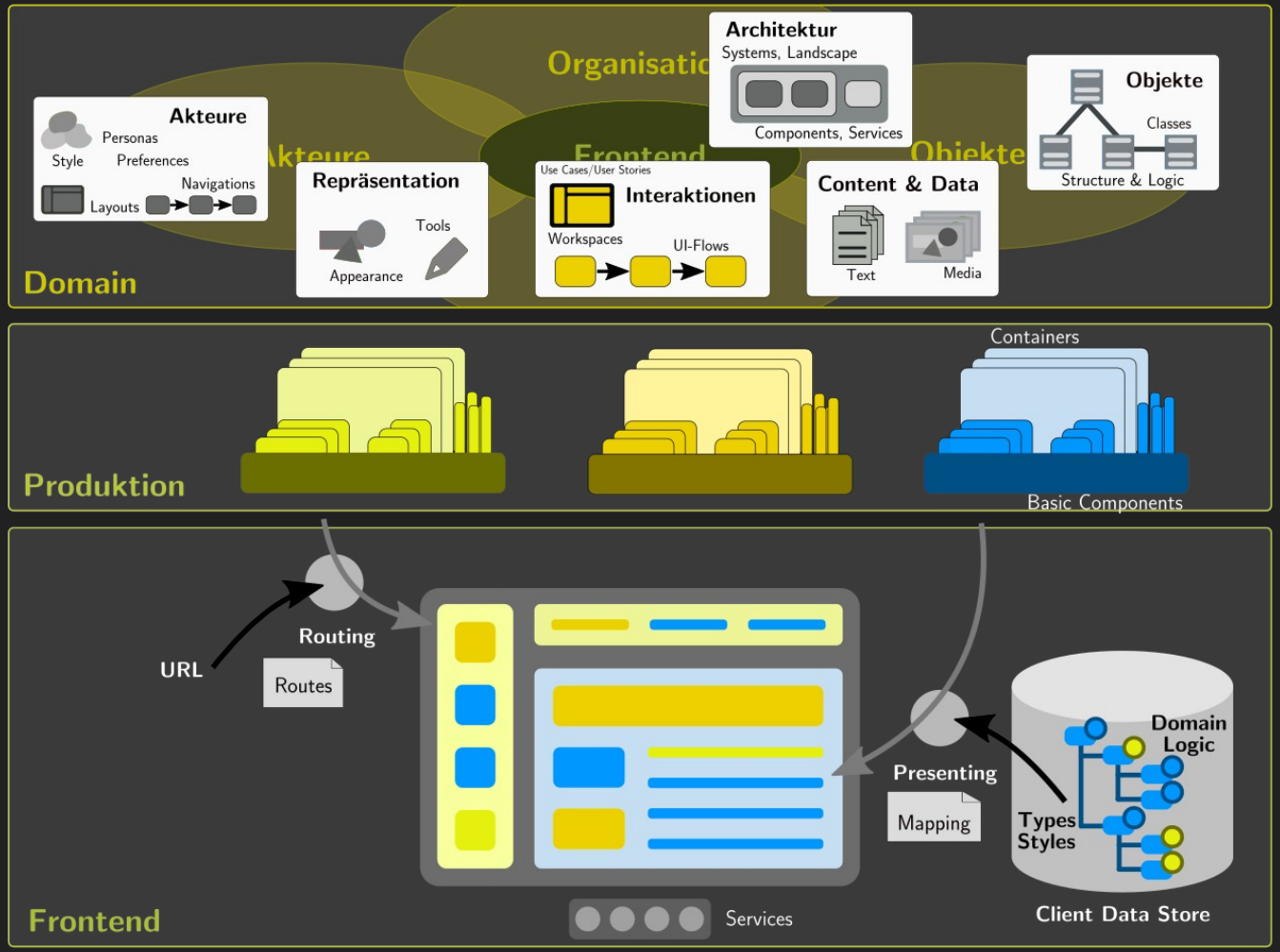
- aus einer Klassen-Beschreibung lassen sich einzelne Dialoge erzeugen
- “pro Attribut ein Feld”

Interaktionen



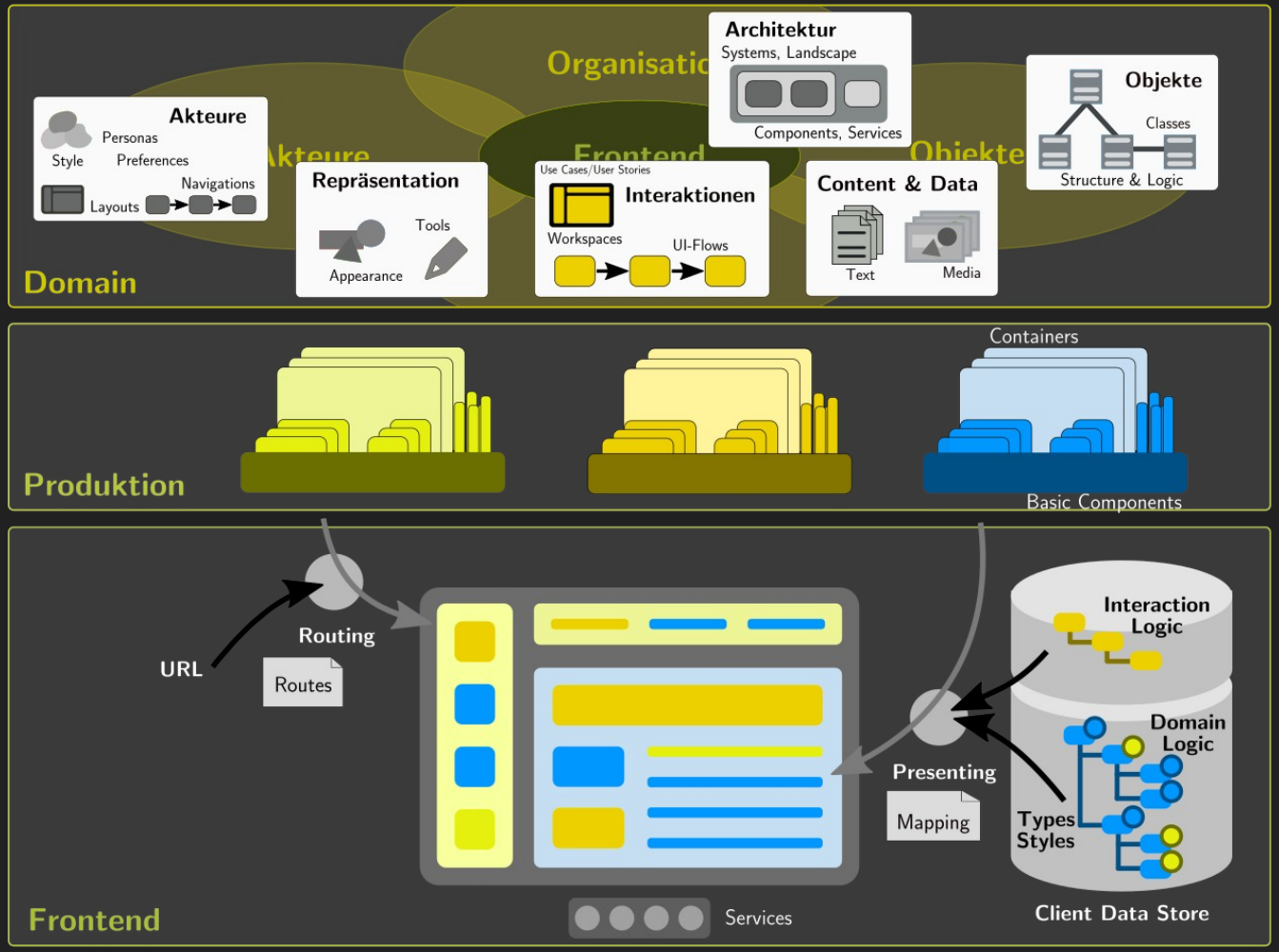
- aus einer Interaktions-Beschreibung lässt sich eine Folge von Dialogen erzeugen
- “pro Step ein Dialog”
- Dialoge sind gefiltert

Interaktionen



- aus Interaktionen können weitere Grund-Elemente erzeugt werden

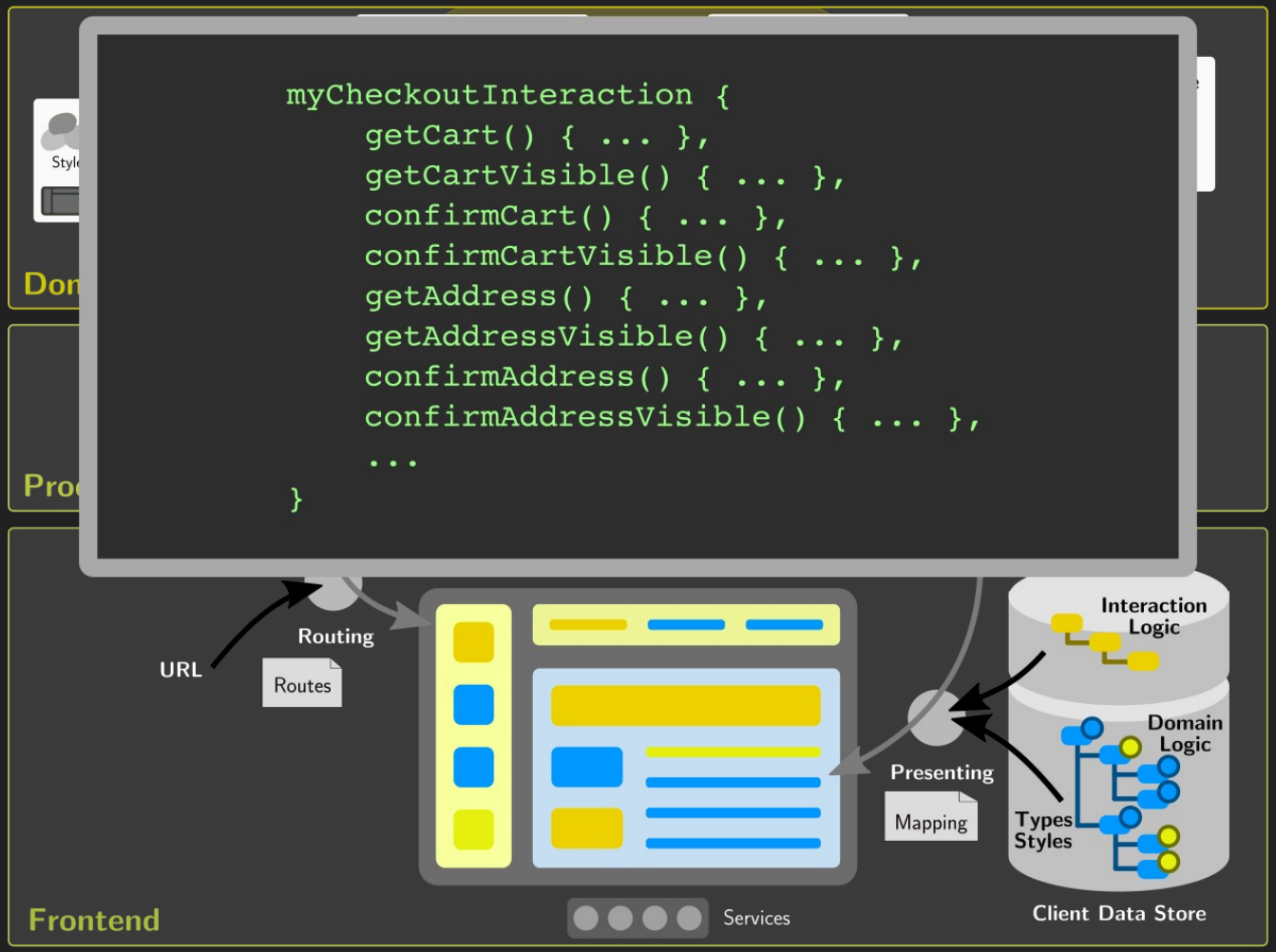
Interaktionen



- Logik und Zustand von Präsentations-Komponente separieren

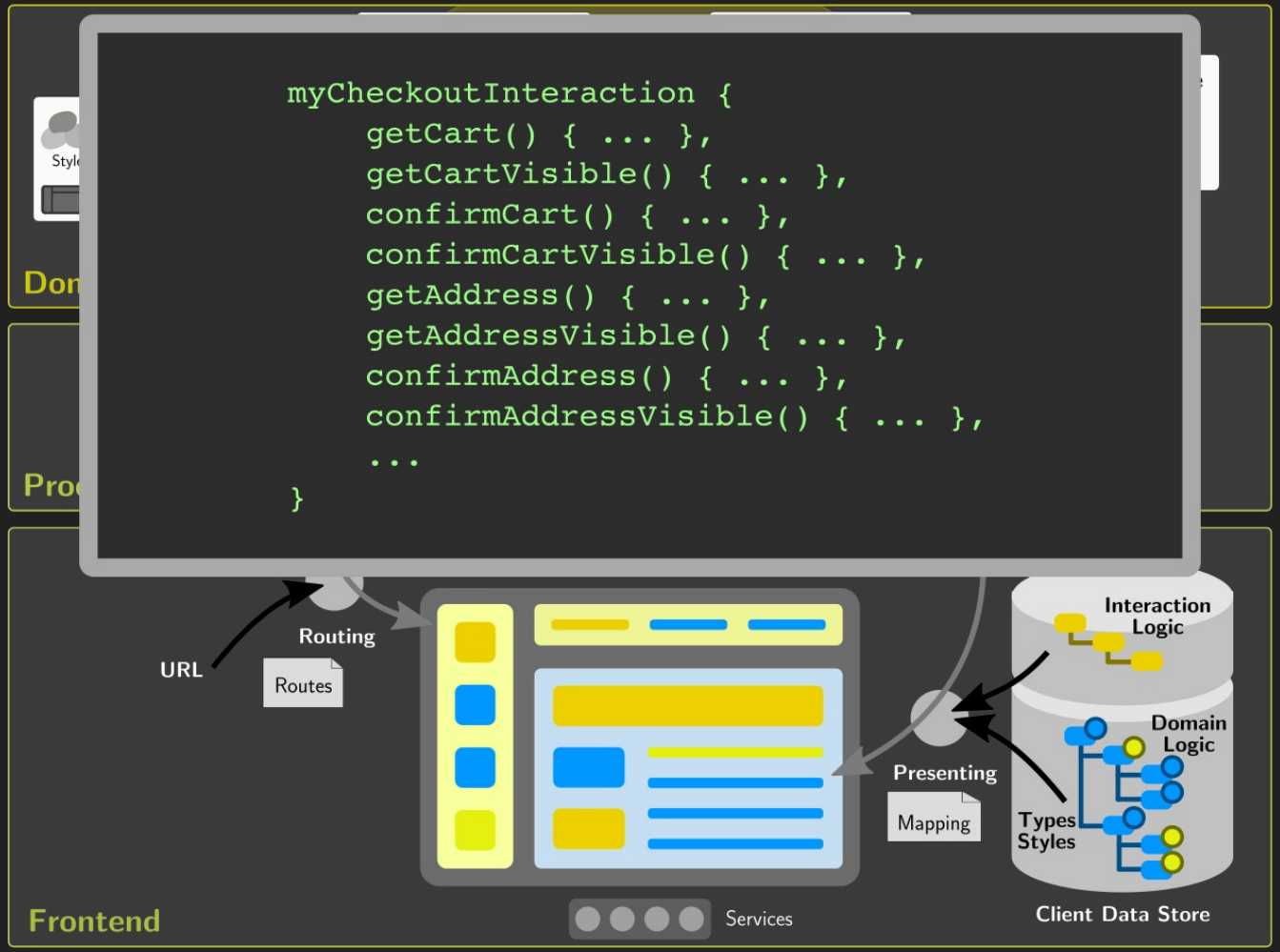
Interaktionen

```
myCheckoutInteraction {
    getCart() { ... },
    getCartVisible() { ... },
    confirmCart() { ... },
    confirmCartVisible() { ... },
    getAddress() { ... },
    getAddressVisible() { ... },
    confirmAddress() { ... },
    confirmAddressVisible() { ... },
    ...
}
```



- Logik und Zustand von Präsentations-Komponente separieren

Interaktionen



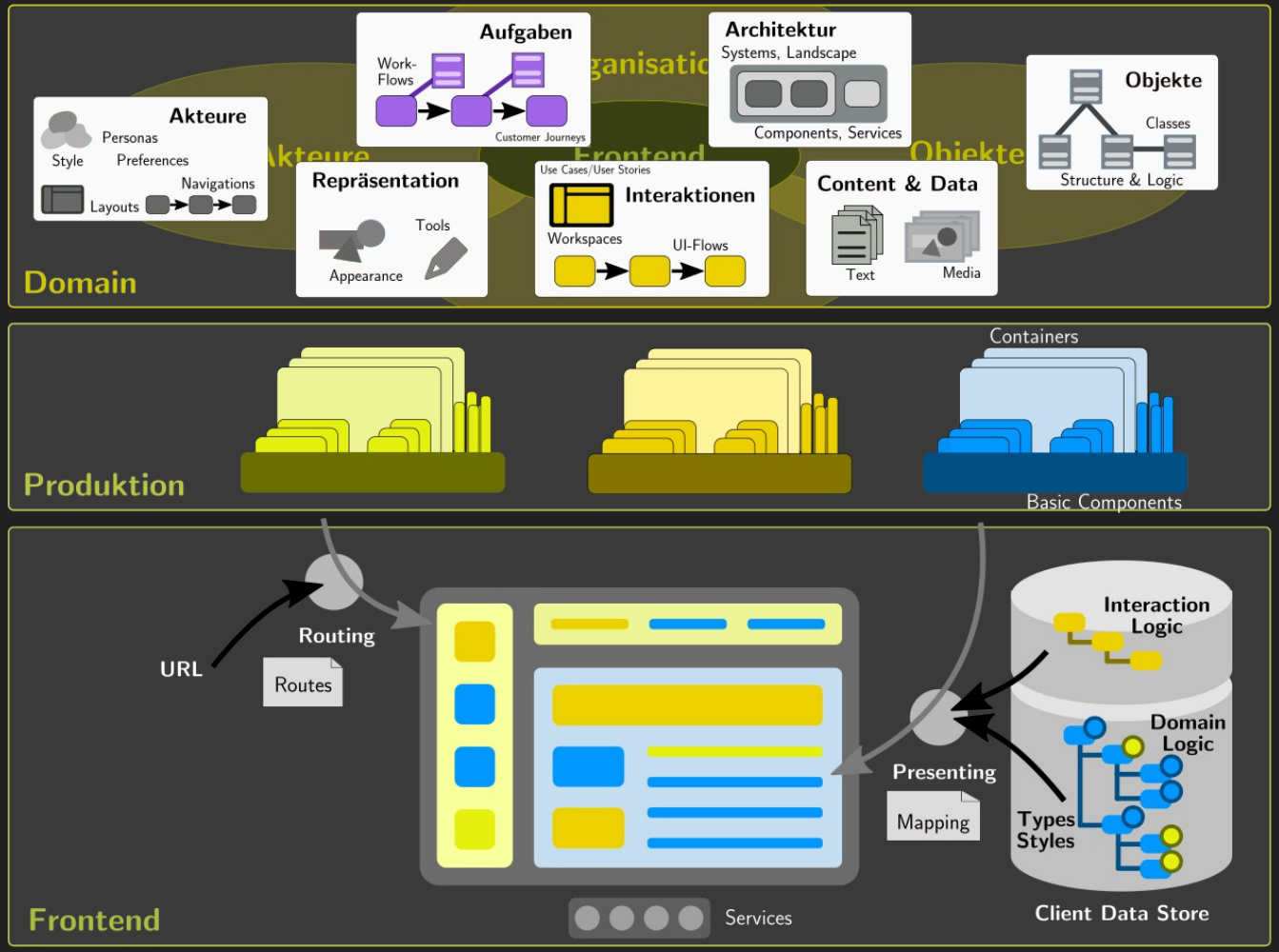
- Logik und Zustand von Präsentations-Komponente separieren
- recht **weitgehende Trennung** von UX/Darstellung und UI-/Domänen-Logik
- Option für **Micro-Frontends** da, hohe Einheitlichkeit
- geringere Framework-Abhängigkeit

Generische Komponenten

```
// get all visible, regular members
Object.keys(item)
      .filter(
        (key) =>  ! /.*Visible$/.test(key)
                  && item[key + 'Visible']()
      )
```

- noch weiter trennen
- keine festen Referenzen zwischen Präsentation und Domänen-Logik
- **generische** Komponenten, bspw.
  - Navigation
  - Abfolgen
  - strukturierte Daten

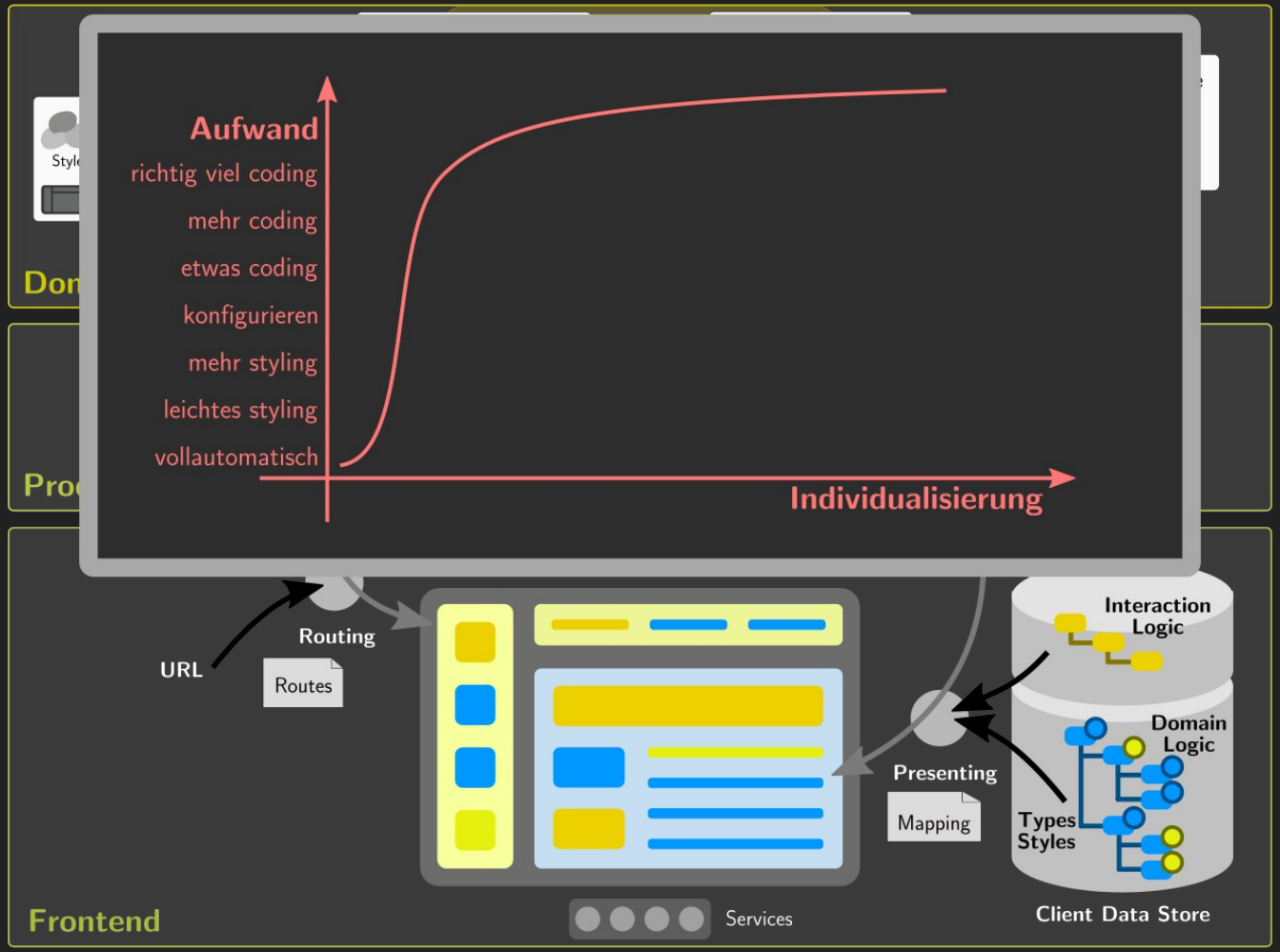
Interaktionen



Weitere mögliche Quelle neben Interaktions-Modell:

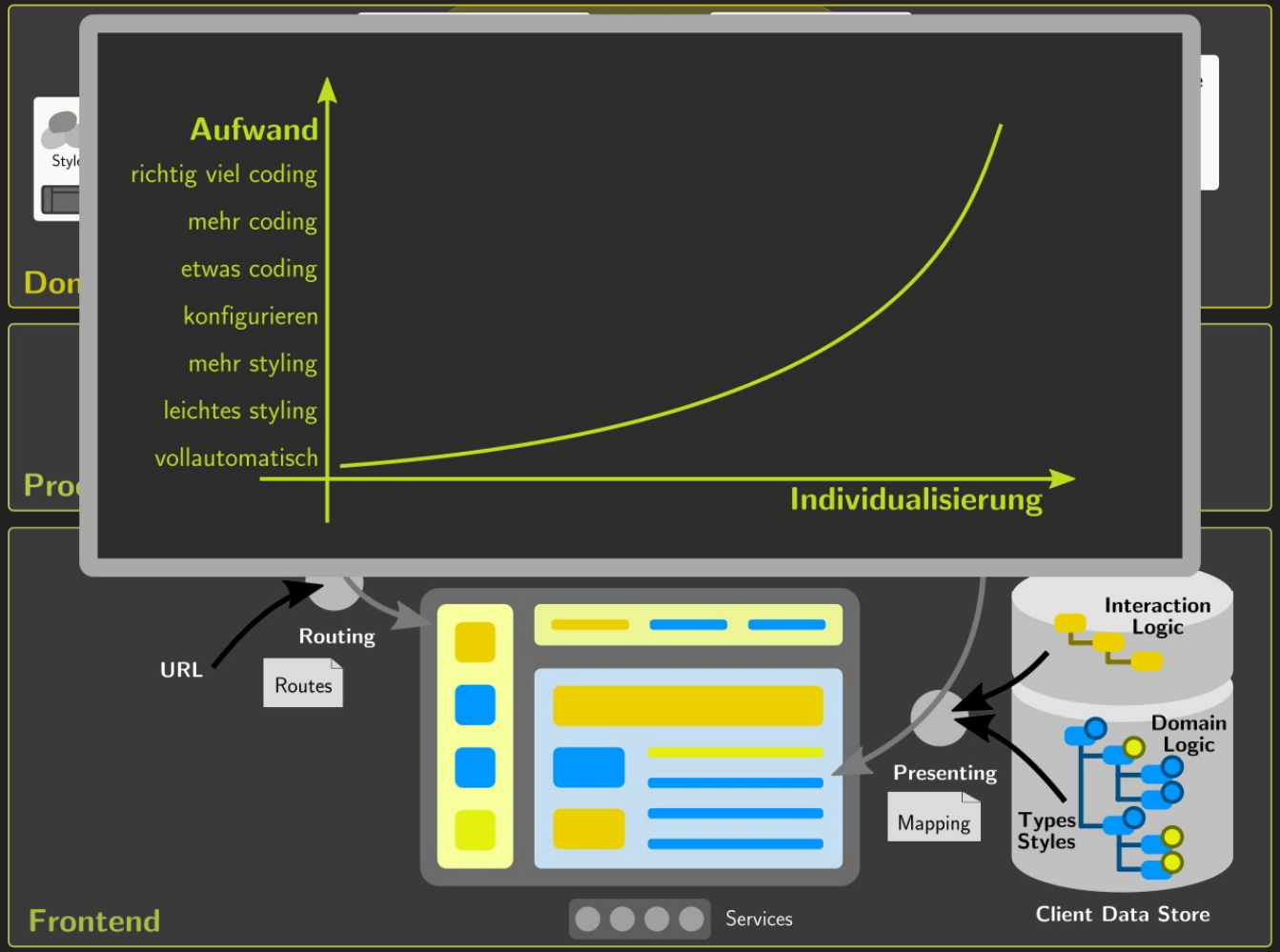
- Aufgaben-Modelle (Workflow, Customer Journeys, Use Cases)

Reibung I - Harter Aufprall



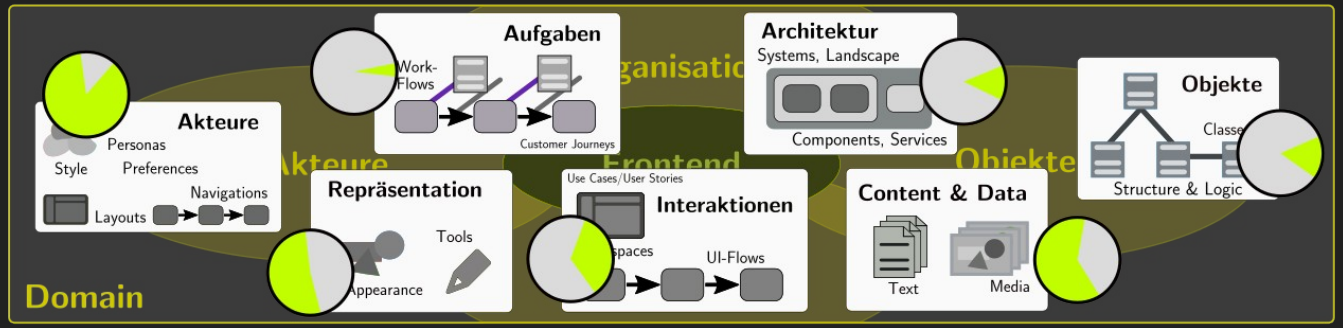
- am Anfang Scaffolding möglich
- danach alles mühsam

Reibung II - Sanfte Landung



- je nach benötigter Individualisierung leicht zunehmender Aufwand
- in der Regel aber wenig
- Recursive Pareto:
  80/20% …
  80/20% …
  80/20% …

Zusammenfassung



Produktion

Frontend

Steckbrief: Welchen Projekttyp?

- Nicht alle Projekte sind gleich!
- Welche Teile der Domäne sind wie wichtig und wie komplex?
- Welche Aspekte (Teilmodelle) sind wie relevant und wie umfangreich?

Verständnis entwickeln - gute Kultur schaffen

- Unter Umständen: nichts tun, alles passt - done.
- Bewusstsein: “Modern Web is Bauhaus”

also Kunst nicht naiv, sondern

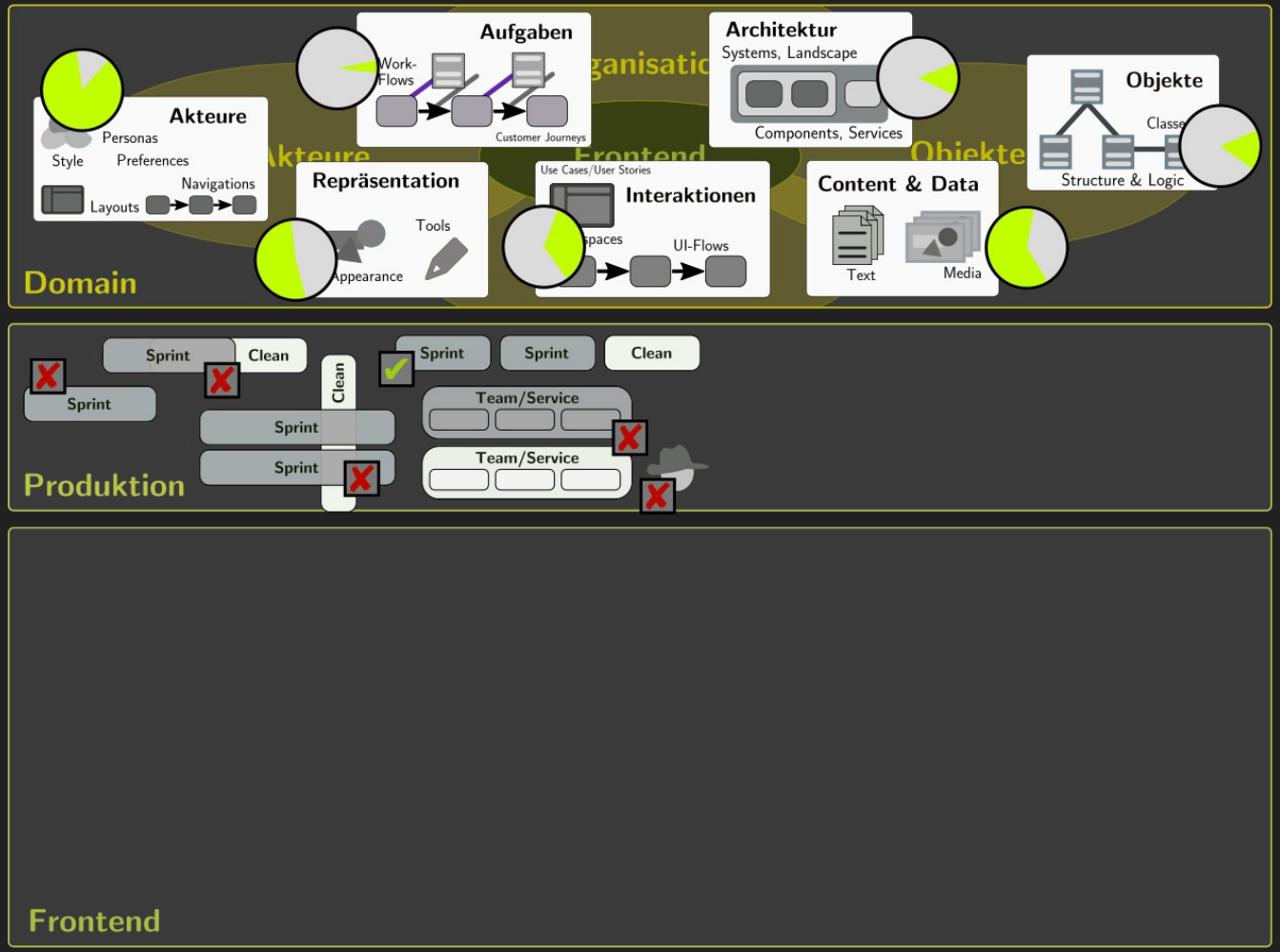
a) Kunst als gemeistertes Handwerk

- "Der Künstler ist die Steigerung des Handwerkers"

b) Spielraum für kreative Kunst

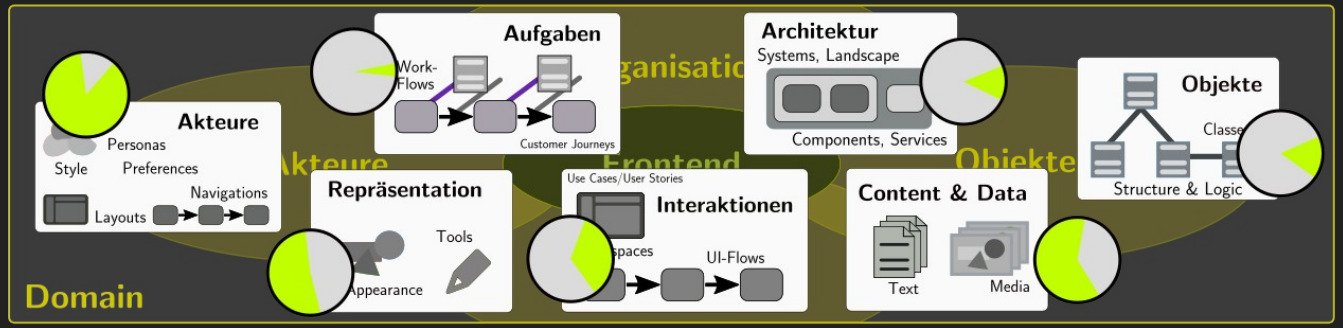
- Regeln brechen vs. Wiedererkennung (Shneiderman)
- Bewusste Wahl ermöglichen durch Recursive Pareto

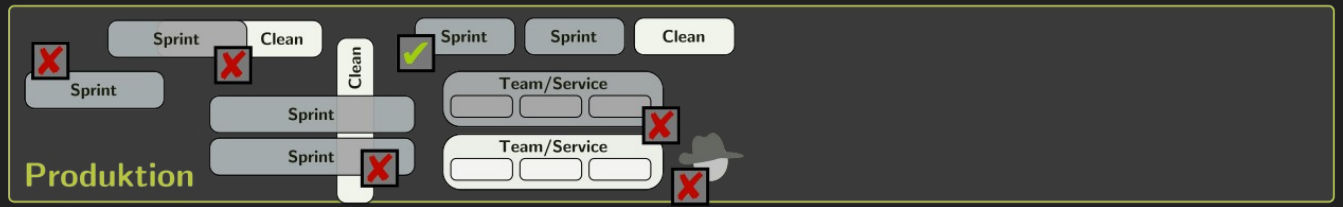
Passender Prozess

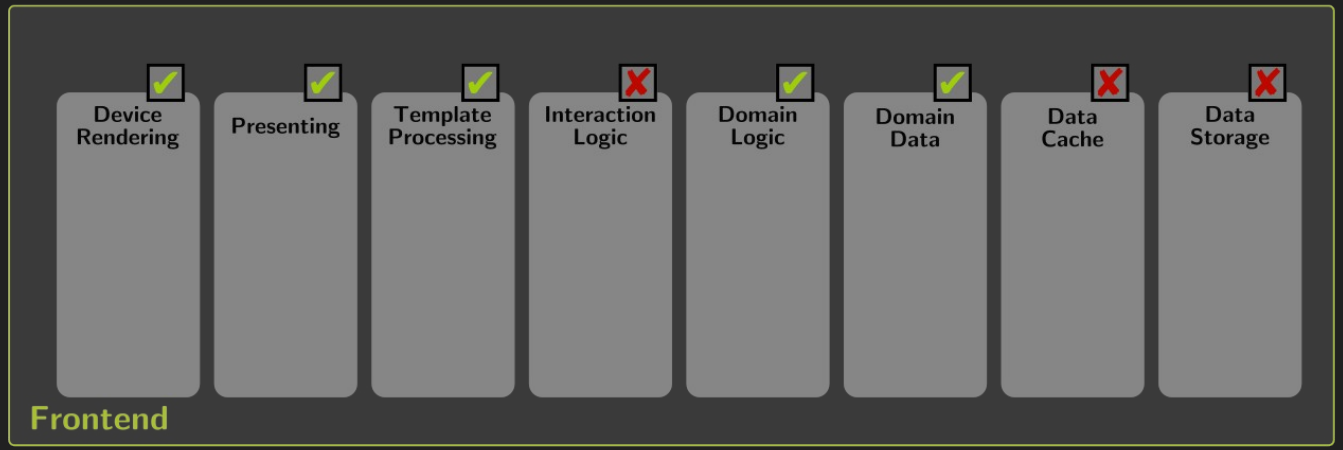


- Nachhaltigkeit gemäß Conways Law im agilen Prozess verankern
- Qualitäts-Pflege belohnen
- z.B.: jede 2. Story Code-Qualität
- pro Sprint ein konkreter Schritt zu Code-Qualität
- Querschnitts-Teams
- Architekt als Stakeholder
- u.ä.

Passende Architektur

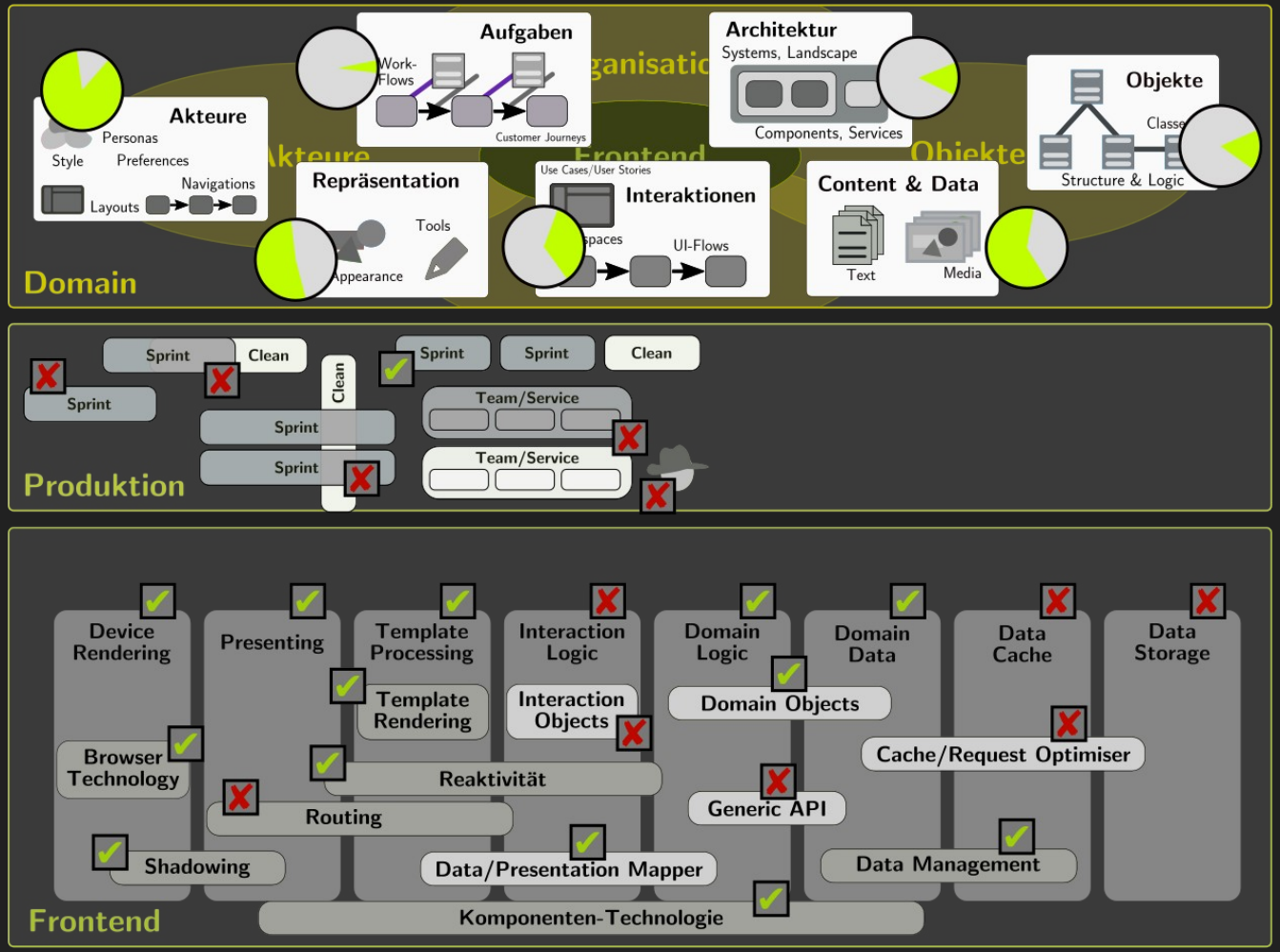






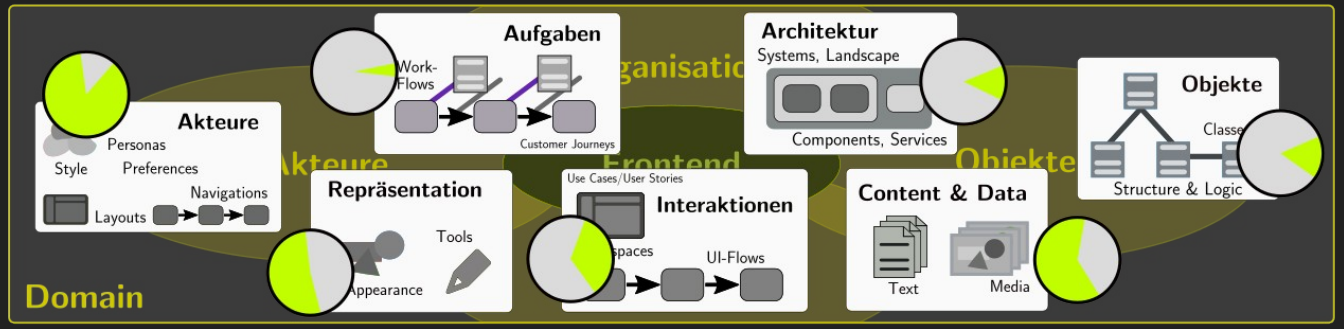
- Welchen Typ App baut man?
- Welche Strukturierung lohnt?

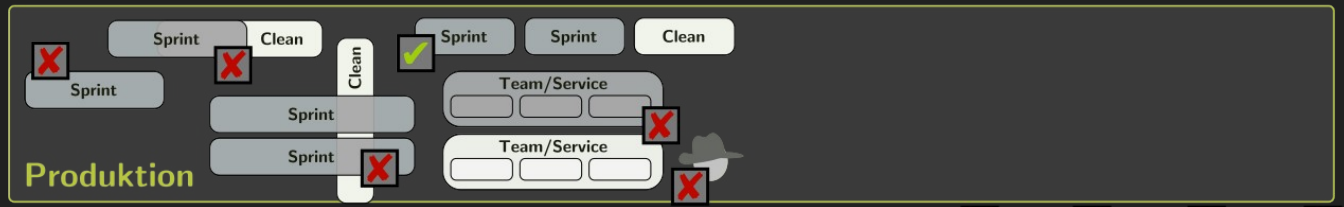
Passende Technologien

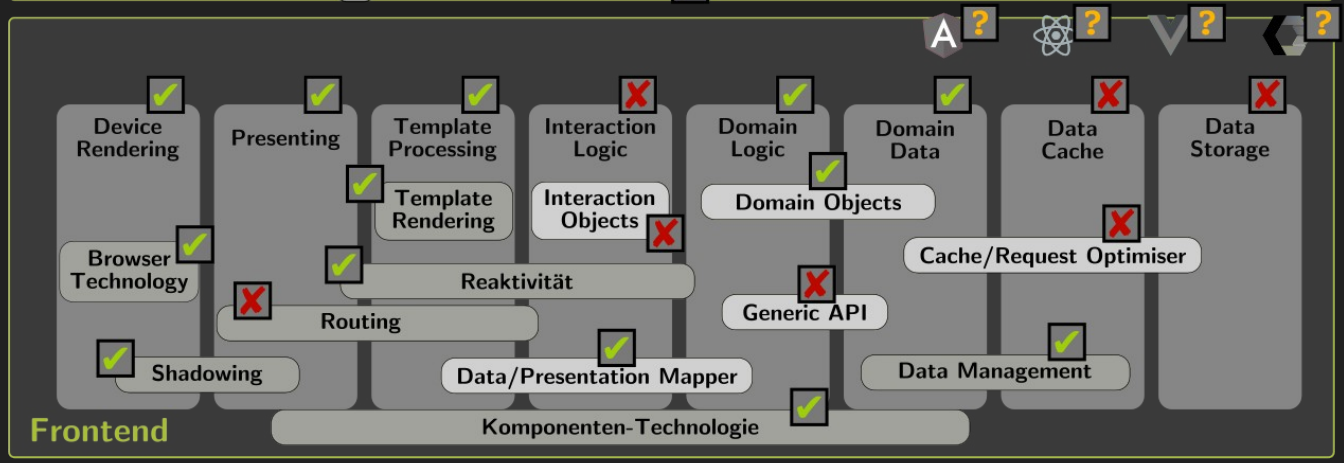


- Ggf. zusätzliche Technologie auswählen

Passendes Framework

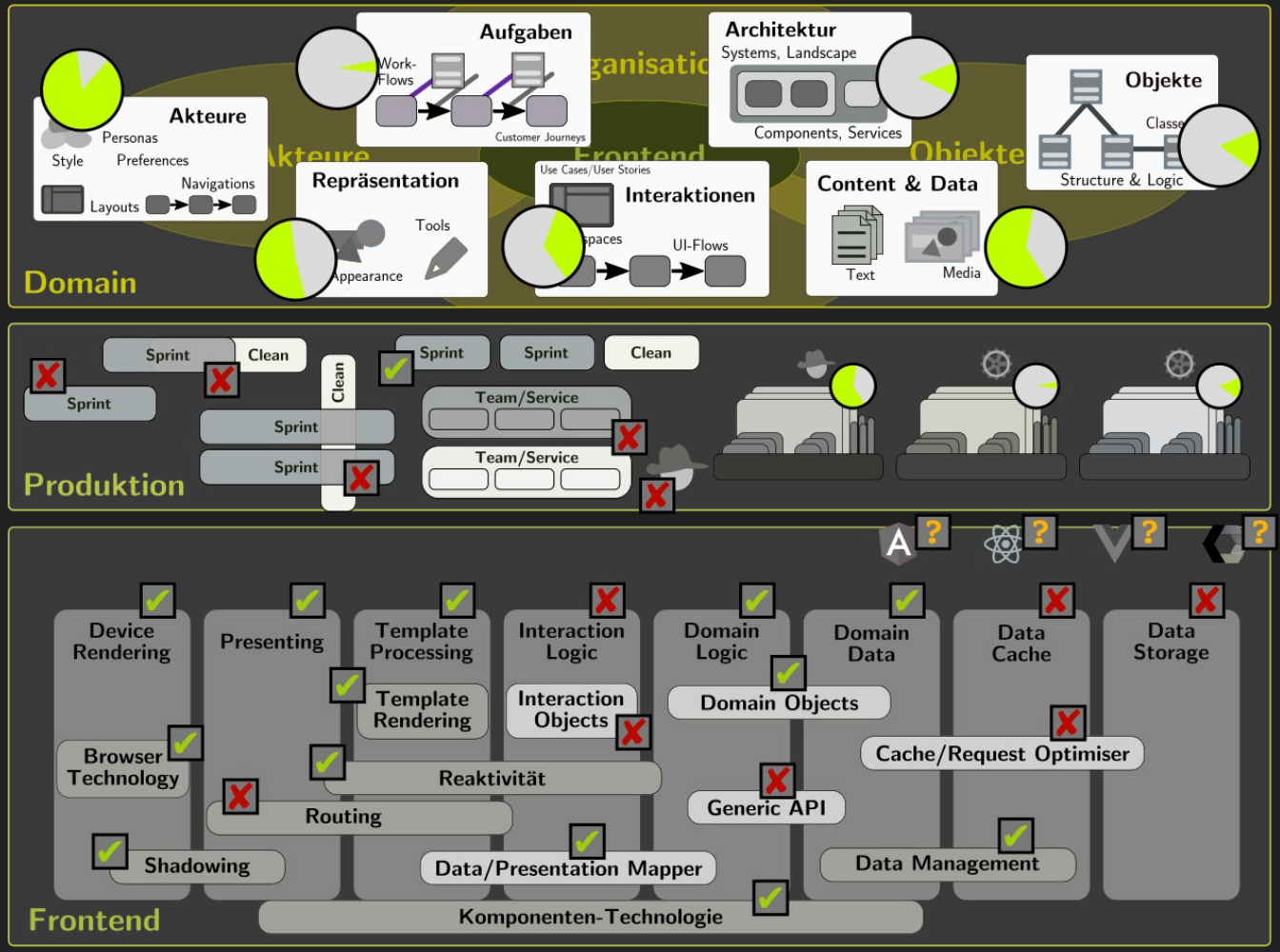




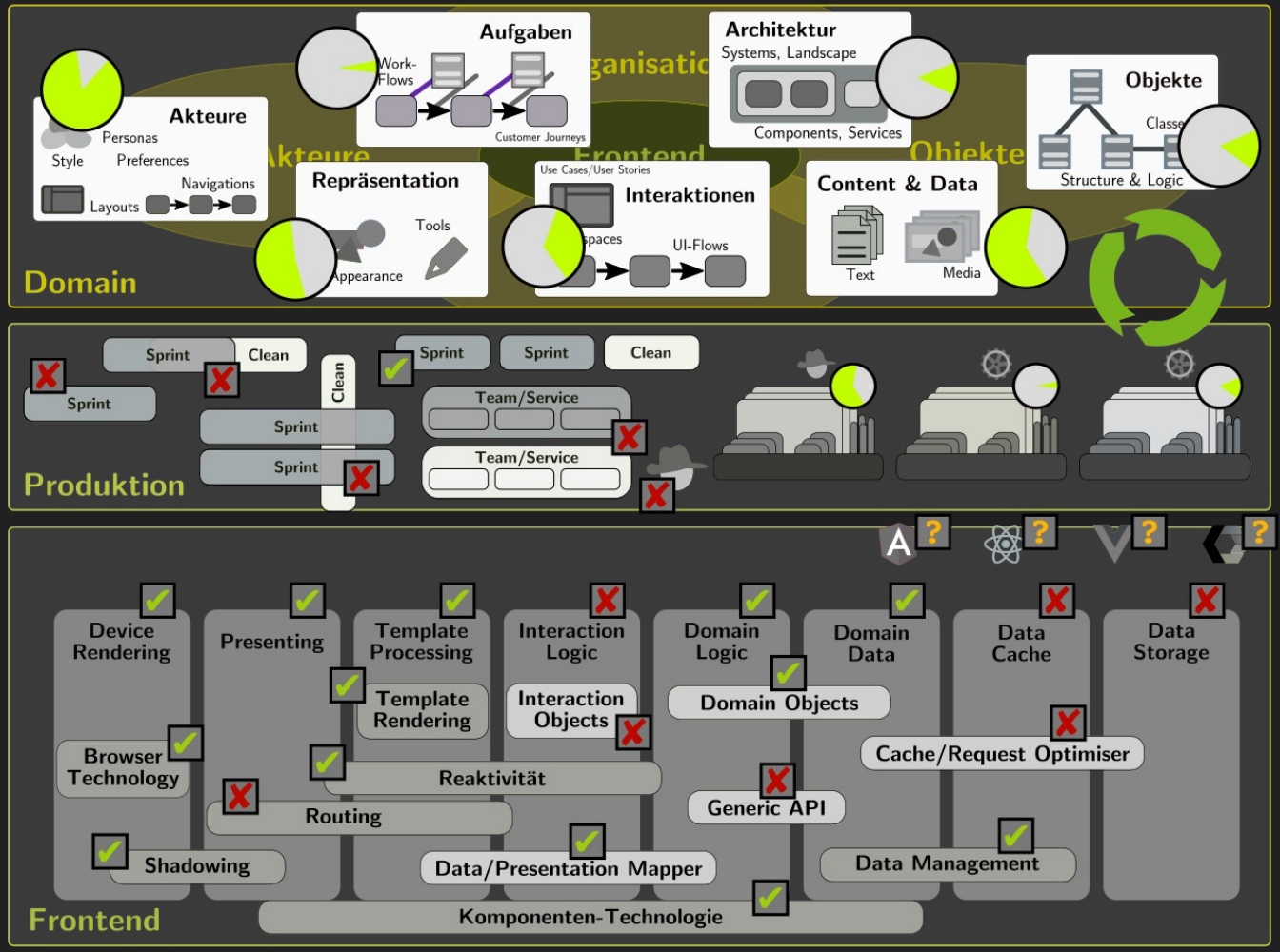


- Auf Basis der Vorüberlegungen (Architektur, Technologien) geeignetes Framework wählen
- Wechsel des Frameworks nicht unmöglich

Passende Komponenten



- Zuschnitt überlegen
- Fallweise entscheiden, ob manuelle Fertigung, Copy & Paste-Reuse, Scaffolding, Generierung oder Generizität besser ist



Sustainable Frontend Toolbox

- Umfangreicher Werkzeugkasten, der viele Optionen zur nachhaltigen Frontend-Entwicklung bietet

https://modernfrontend.design

Quellen / Lizenzen

Vue Logo	https://de.wikipedia.org/wiki/Datei:Vue.js_Logo_2.svg
Angular Logo	https://de.wikipedia.org/wiki/Datei:Angular_full_color_logo.svg
React Logo	https://de.wikipedia.org/wiki/Datei:React-icon.svg
Frontend/Backend	https://diaspodon.fr/@jaywink/101466550027090670