UI/UX ENTWICKLUNG MIT

# REACTJS

METHODEN
& KONZEPTE

REACTJS

PRAXIS

# UI/UX ENTWICKLUNG MIT REACTJS

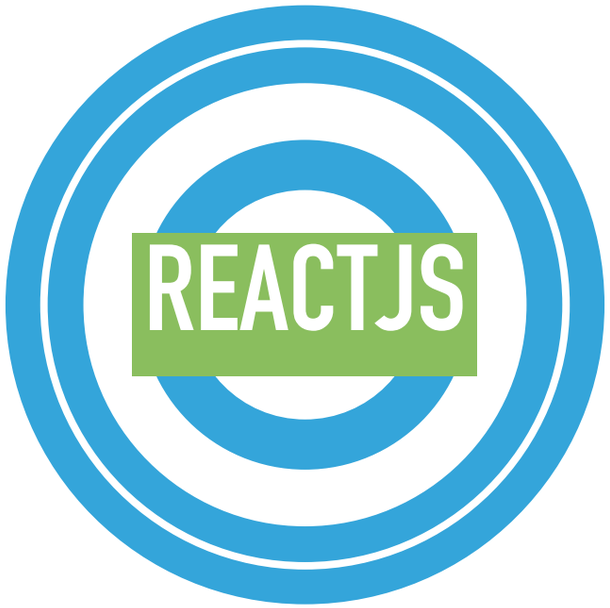

STRUKTUR

KOMPLEXITÄT

WIEDERVERWENDUNG

SOLL/IST

# UI/UX ENTWICKLUNG MIT REACTJS




INFORMATIONS ARCHITEKTUR

DOMAIN DRIVEN DESIGN

ATOMIC DESIGN

TESTING

# ZUSAMMENFASSUNG

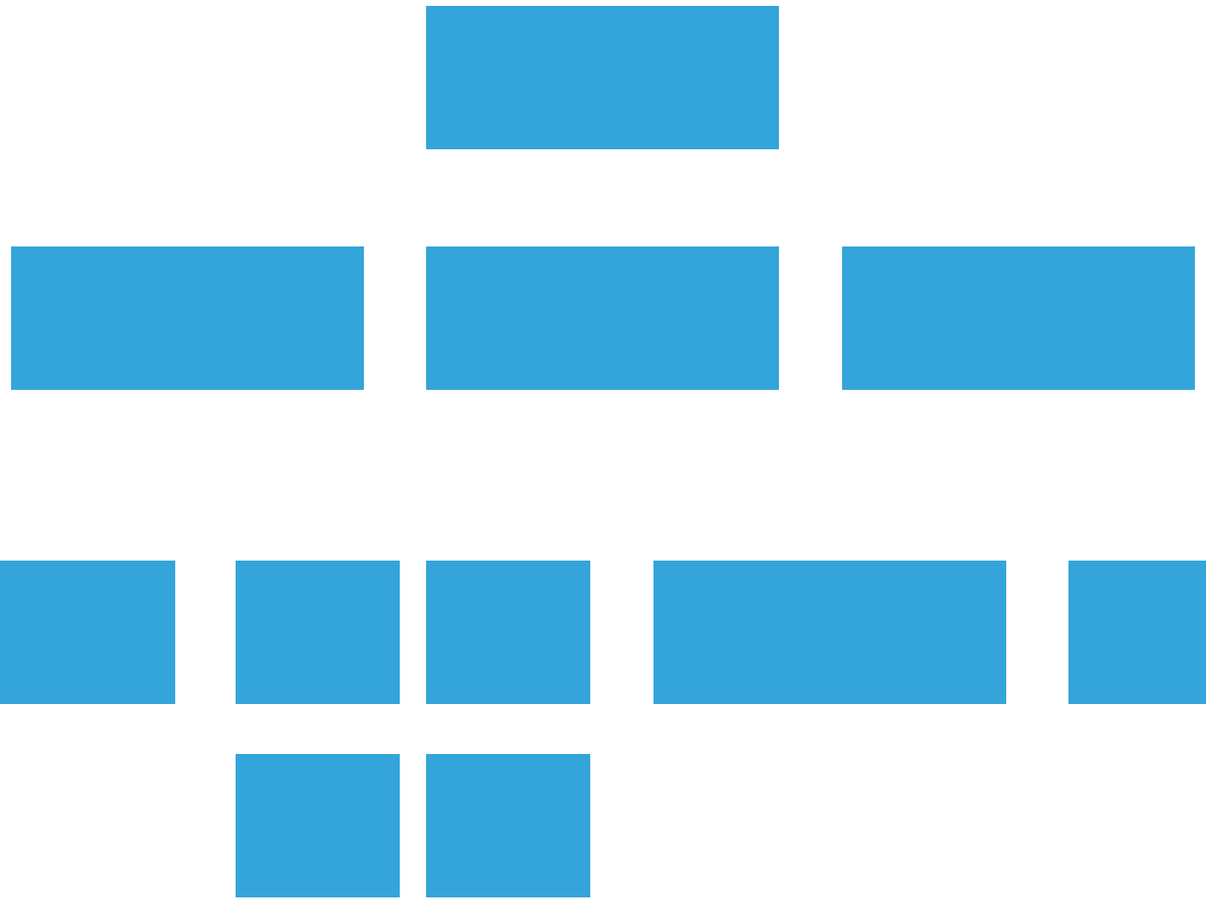
- Interdisziplinäre Zusammenarbeit
- Verbinden von Nutzererwartung und fachlich/technischer Lösung
- Aufbau und Festigung von explizitem Wissen
- Prägen möglichst exakter Begriffe und deren Bedeutung
- Festlegen von Kontext und Struktur durch Abgrenzung und Gruppierung
- Organisation von Abhängigkeiten
- Sicherung der Integrität zwischen Anwendungskomponenten

# INFORMATION ARCHITECTURE

# WAS IST EIGENTLICH INFORMATIONSARCHITEKTUR?

> EINE EFFEKTIVE IA WAR UND IST ALSO EINE STRUKTUR, DIE SICH DEM NUTZER NICHT IN DEN WEG STELLT, SONDERN HILFT INNERHALB EINES (MEIST DIGITALEN) INFORMATIONSSYSTEMS EIN GEWÜNSCHTES ZIEL ZU ERREICHEN.
>
> **Jan Jursa**

# STRUKTUR



# WAS BRAUCHT DER BENUTZER?

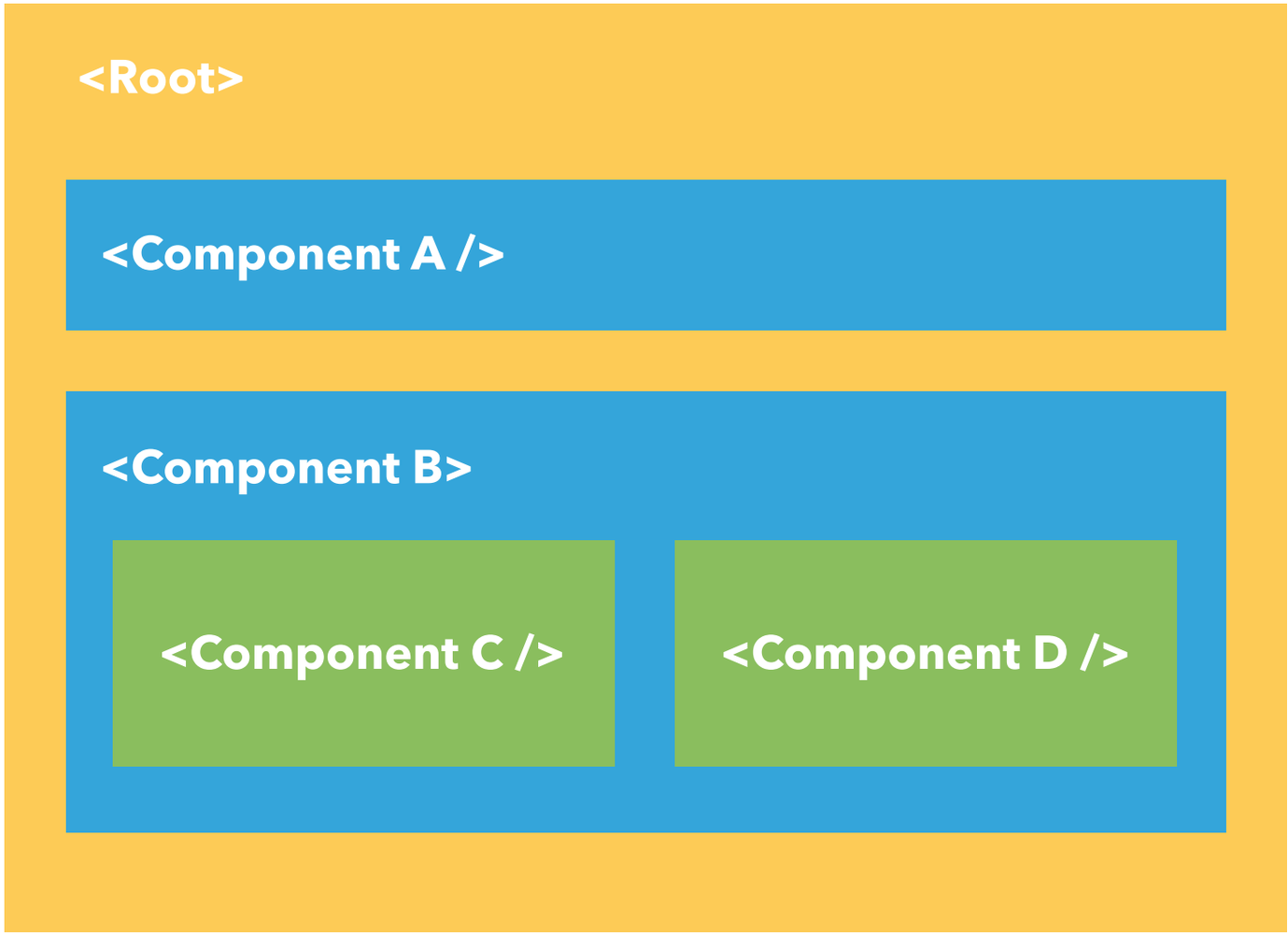
- **Organization**
  Gruppierung von Informationen nach z.B. Zeit, Thema, Funktion, Aufgabe, Prozess, Publikum, Attribut, Facette, Standort

- **Benennung**
  Erarbeiten von Begriffen und Gruppierungen

- **Priorisieren**
  Anordnen der Inhalte

- **Verbindungen**
  Verknüpfen von Aufgaben und Lösung

# REACTJS HIERARCHIE



# REACTJS KONZEPTE UND ARCHITEKTUR

- DOM vs. VDOM
- Redering
- Eltern-Kind Strukturen
- Unidirektionale Datenbindungen
- Ereignisse

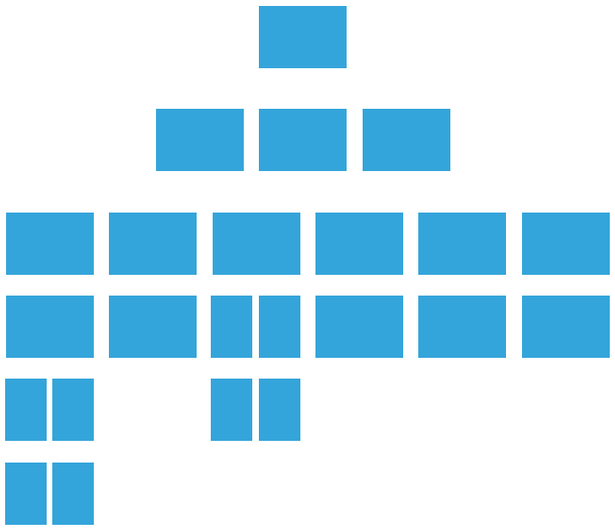
# REACTJS UNIDIREKTIONALE DATENBINDUNG

DATEN

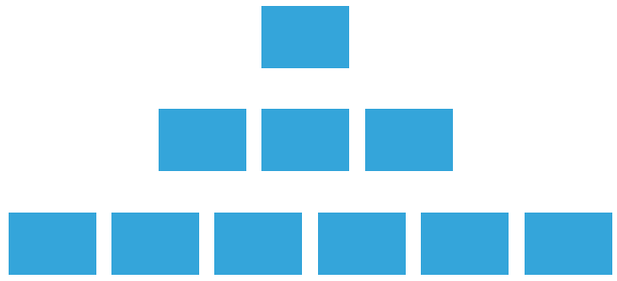
EREIGNISSE

# REACTJS ARCHITEKTUR

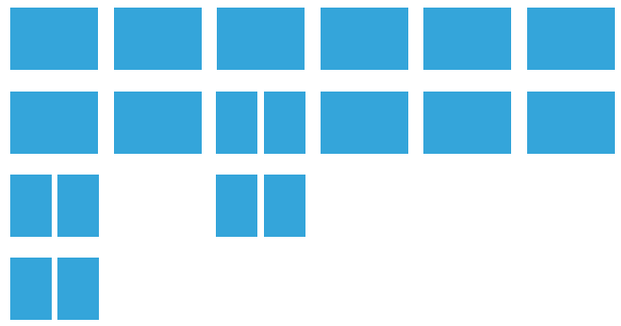
APP



APP A



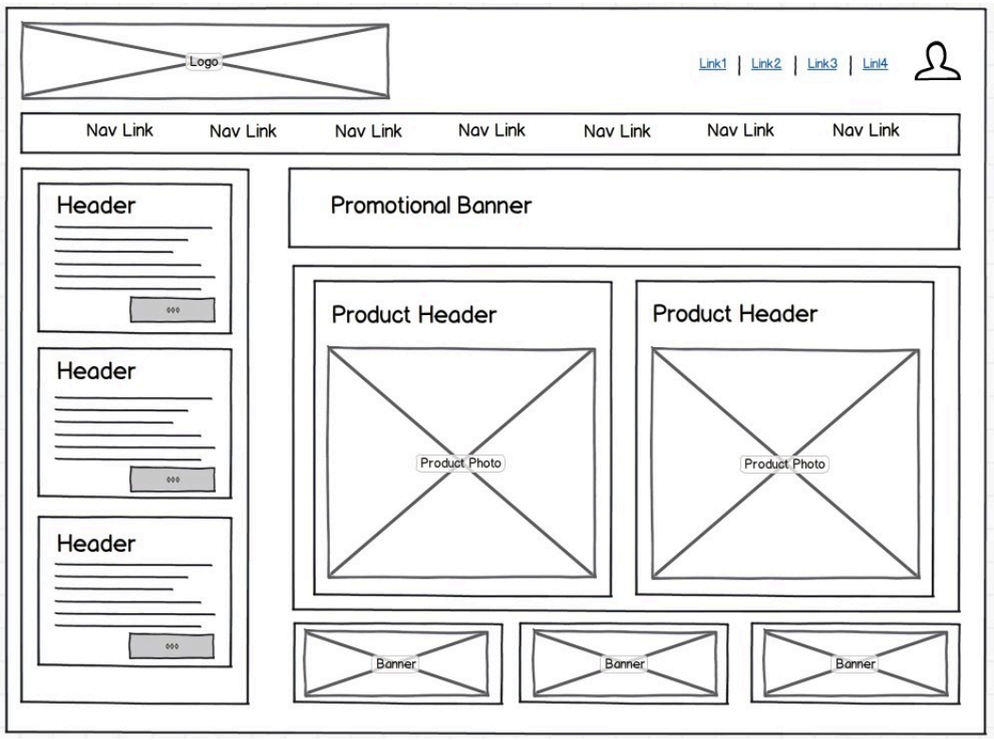
SHARED



# TOOLS

- Storyboard & Wireframes - **Balsamiq**
- Projekte/Workspaces - **Yarn / Lerna**
- App/Pages/Component Hierarchie - **React Router**

# STORYBOARD & WIREFRAMES




- Benutzerführung - Gruppierung, Benennung und Interaktion
- Verknüpfung von Daten und Aufgaben
- Navigation und UI Mockups

# PROJEKTE/WORKSPACES

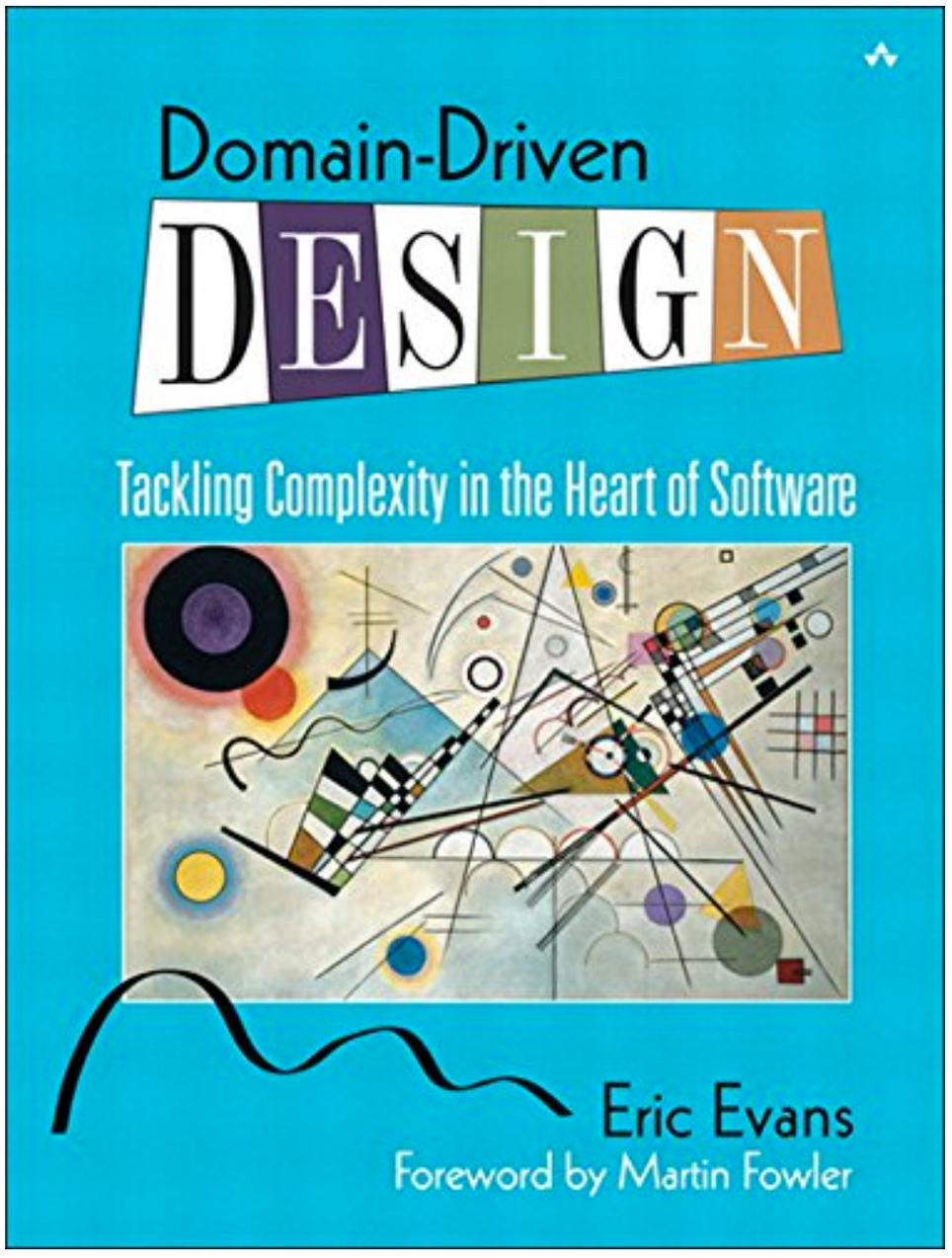
- Isolieren und Gruppieren von fachlichen und technischen Arbeitsbereichen
- Verwaltung von Abhängigkeiten
- Explizite APIs, Versionierung und Dokumentation

# ZUSAMMENFASSUNG

- Fachliche Begriffe und Zusammenhänge erkennen, prägen, anordnen und verknüpfen
- Gewährleisten einer hohen Informationsintegrität
- Strukturierung von komplexen Informationen, Aufgaben und deren Beziehungen in ein Wireframes/Storyboard
- Erkennen, isolieren, benennen und etablieren von UI/UX Mustern
- Designgrundlage für ReactJS Komponenten-APIs, Datenbindung und Ereignisse

# FRAGEN?

# DOMAIN DRIVEN DESIGN

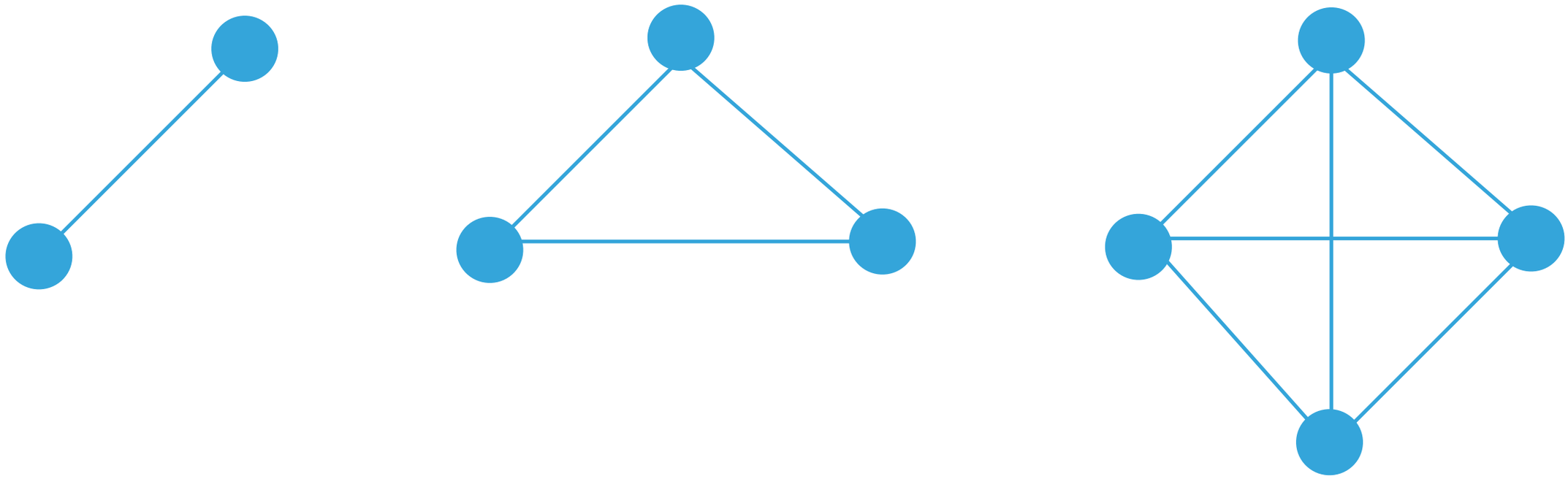


# WAS IST EIGENTLICH DDD?

DOMAIN-DRIVEN DESIGN, KURZ DDD, BESCHREIBT VORGEHENSWEISEN, DIE KOMPLEXE SOFTWARE-PROJEKTE TRANSPARENTER FÜR ALLE BETEILIGTEN MACHEN SOLLEN. GLEICHZEITIG DEFINIERT ES EINE REIHE VON TECHNIKEN UND ELEMENTEN, MIT DENEN EIN OPTIMIERTES DOMÄNENMODELL ERREICHT WERDEN SOLL.

**Eric Evans**

# KOMPLEXITÄT



. . . KOMMUNIKATION, ABHÄNGIGKEITEN, ZUSTÄNDE, KOMPONENTEN, ETC.

# DOMAIN DRIVEN DESIGN

- Etablieren einer allgegenwärtigen fachlichen und technischen Sprache
- Förden von interdisziplinärer Kommunikation
- Schwerpunkt ist das modellieren fachlicher Zusammenhänge
- Kapseln von fachlicher Logik und technischen Abhängigkeiten

# DOMÄNENMODELL (VEREINFACHT)

- Wertobjekte, Entitäten und Aggregate
- Assoziationen und fachlich/technische Ereignisse
- Serviceobjekte, Repositories, etc.

# „FUNCTIONAL“ DDD & REACTJS

| | | | |
|---|---|---|---|
| VALUE OBJECT<br>ENTITY | AGGREGATE | EVENTS | SERVICES |
| COMPONENT<br>(FUNCTION/CLASS) | COMPONENT<br>COMPOSITION | FUNCTION<br>(HANDLER) | CONTEXT<br>(PROVIDER) |

# REACTJS KONZEPTE UND ARCHITEKTUR

- Zustand vs. Zustandslos mit Smart- und Dump-Components
- Komponentenorientierung und Komposition mittels Higher-Order-Components & Render-Props-Pattern
- Kontext und das Provider-Consumer-Pattern
- Asynchrone Kommunikation mit RESTful APIs, GraphQL, etc.
- Event-getrieben mit globalem Zustand durch Flex/Redux

# STATELESS VS. STATEFULL COMPONENT

<Component text=„ABC“ />

RENDER

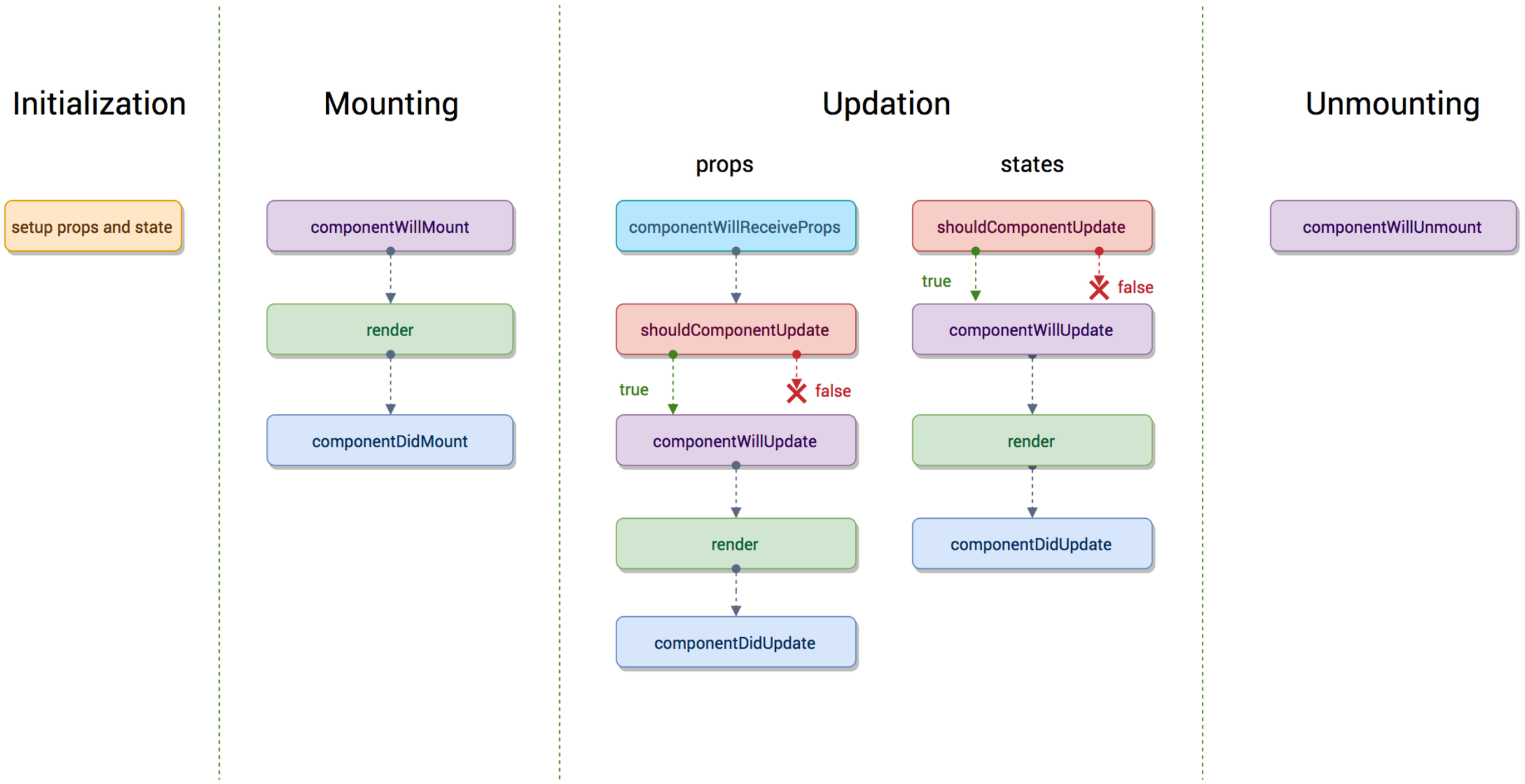
<Component text=„ABC“ />

RENDER
USESTATE
RERENDER

```
const [isLoading, setIsLoading] = useState(true);
```

USEEFFECT

# COMPONENT LIFECYCLE



# COMPONENT LIFECYCLE MIT HOOKS

> **Tip**
>
> If you're familiar with React class lifecycle methods, you can think of `useEffect` Hook as `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount` combined.

```
useEffect(() => {
  doSomething();
}, []);
```

https://github.com/streamich/react-use

# RENDER-PROPS PATTERN

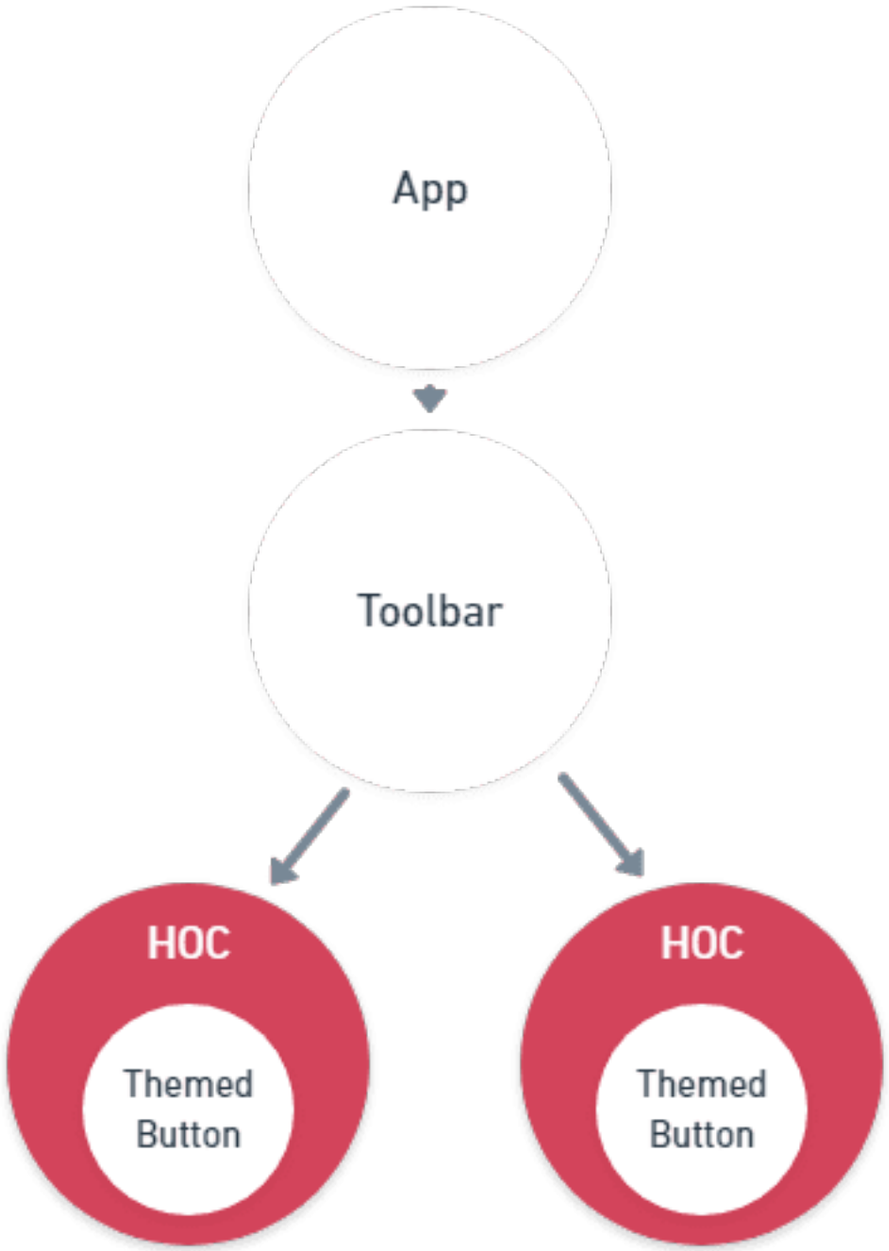
```
<Component
    renderSomething={() => <CompontentA />
/>
```

```
<Parent>
    {() => return <ComponentA />}
</Parent>
```

# HIGHER-ORDER-COMPONENTS

## COMPONENT

## OWN STATE

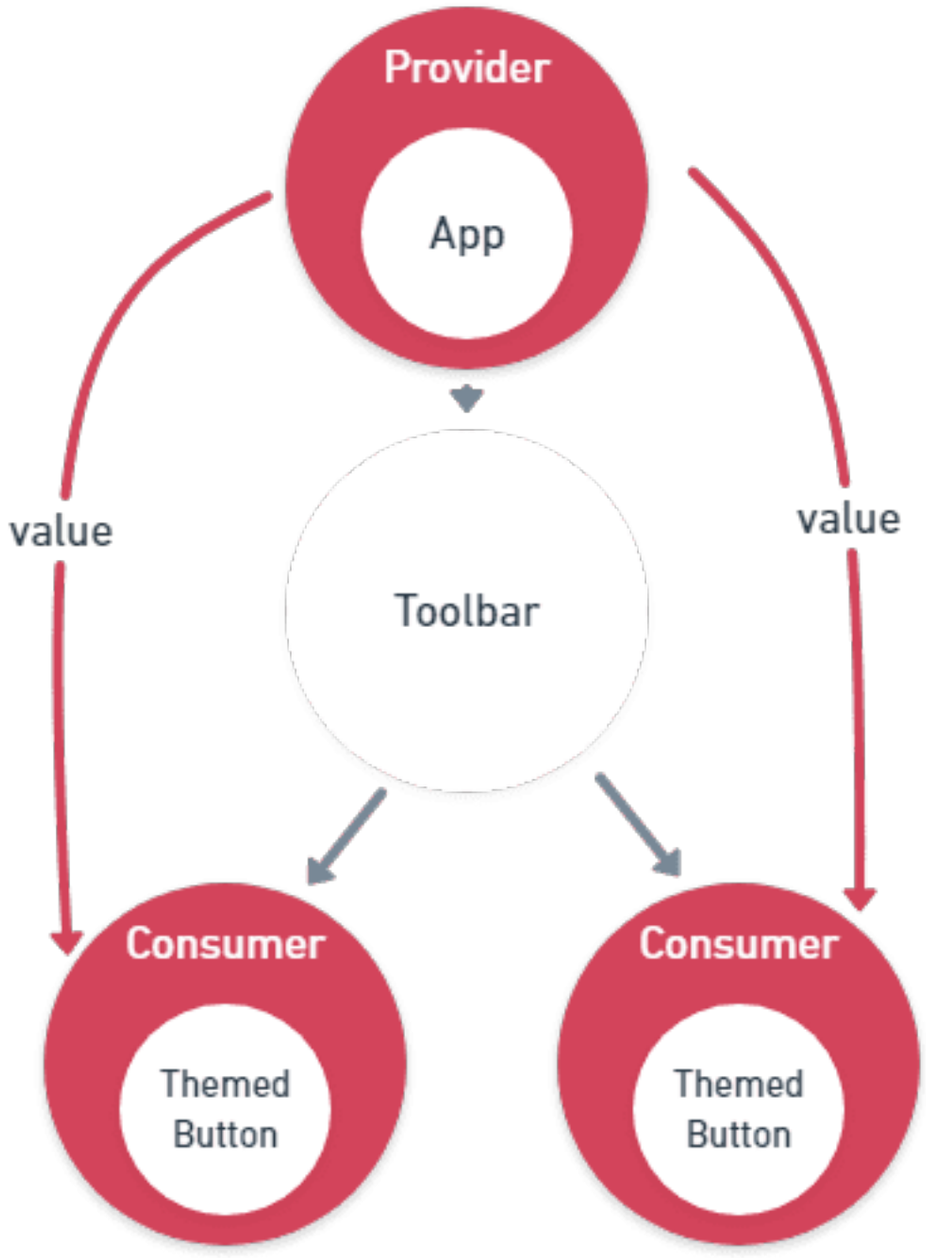



https://github.com/acdlite/recompose

# CONSUMER-PROVIDER PATTERN

# KONTEXT

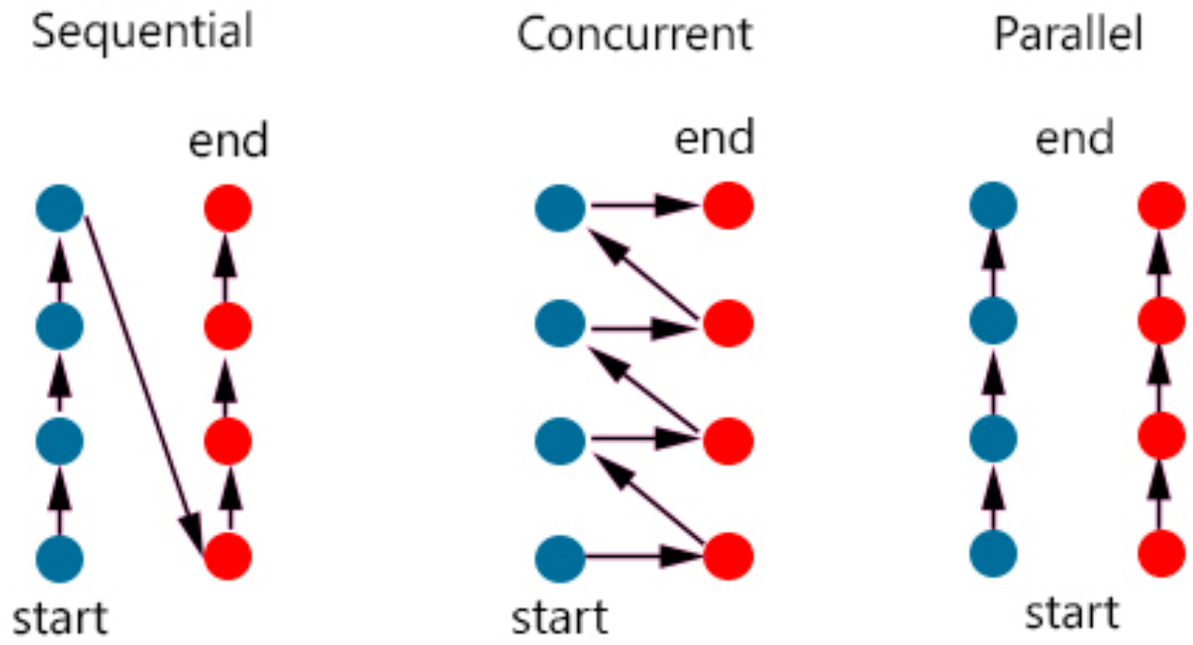
# SHARED STATE



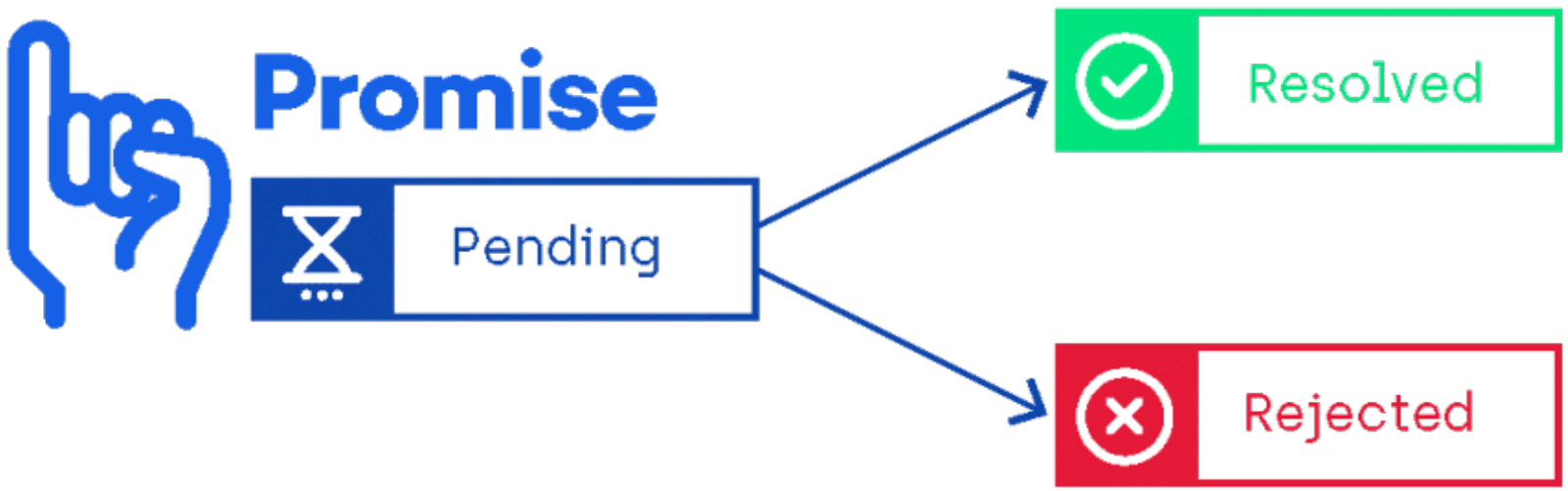
# HOCS VS. PROVIDER

- Sind Daten/Zustand global? Provider
- Ist der Zugriff dynamisch? Provider
- Handelt es sich um statischen Zugriff? HOC / Hooks

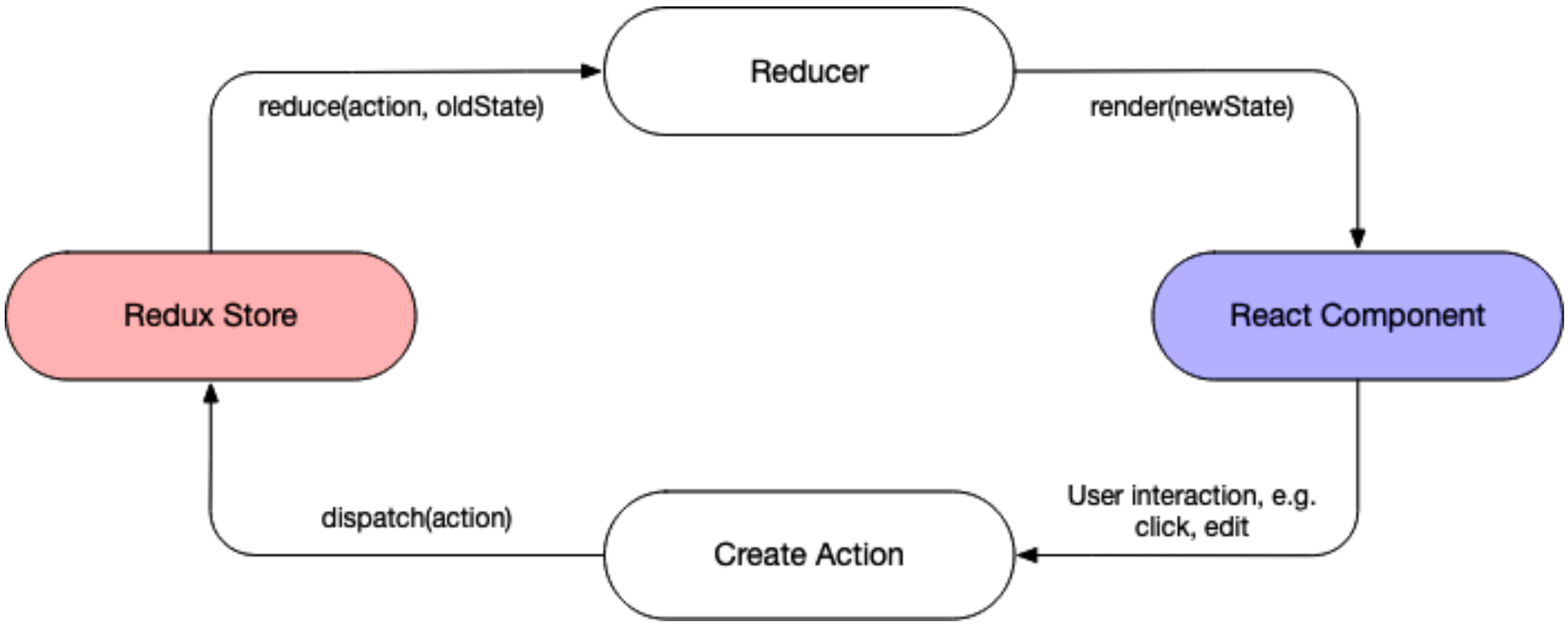
# ASYNCHRONE KOMMUNIKATION



# ASYNCHRONE PROGRAMMIERUNG

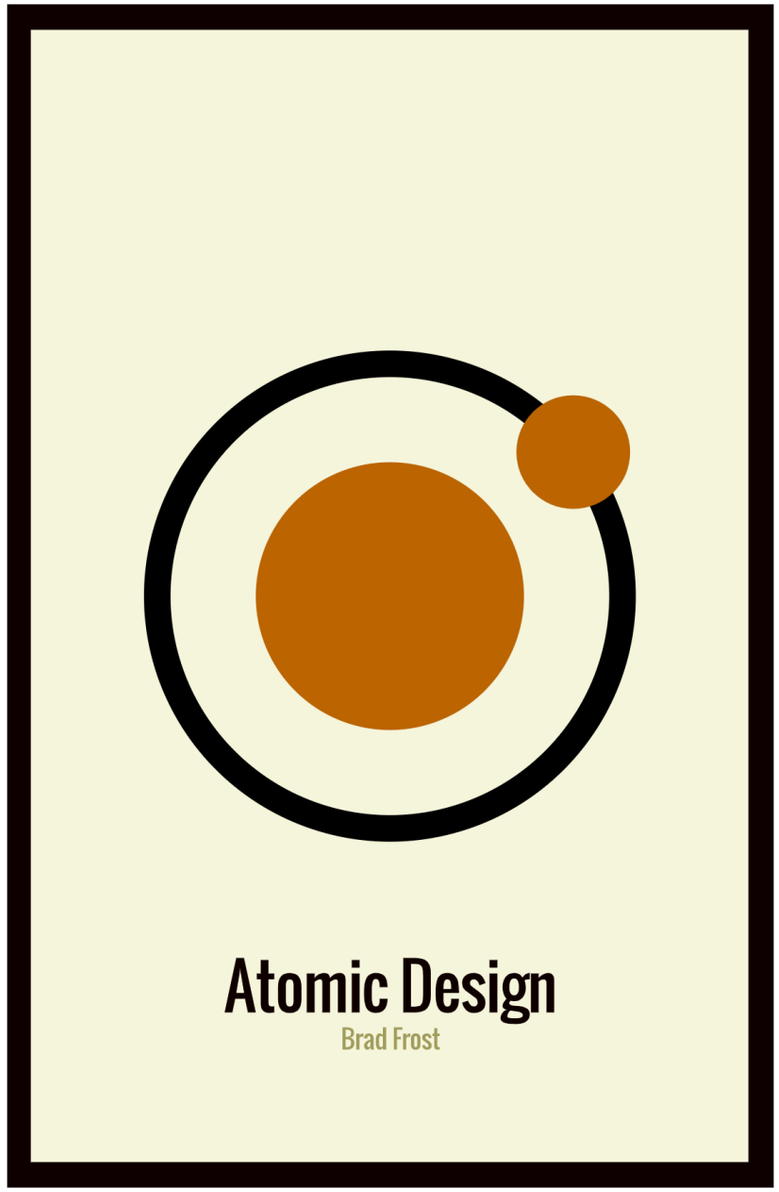


# EVENT-DRIVEN MIT FLUX/REDUX



# FRAGEN?

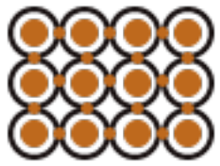
# ATOMIC DESIGN



# KOMPONENTEN

ATOMS

MOLECULES

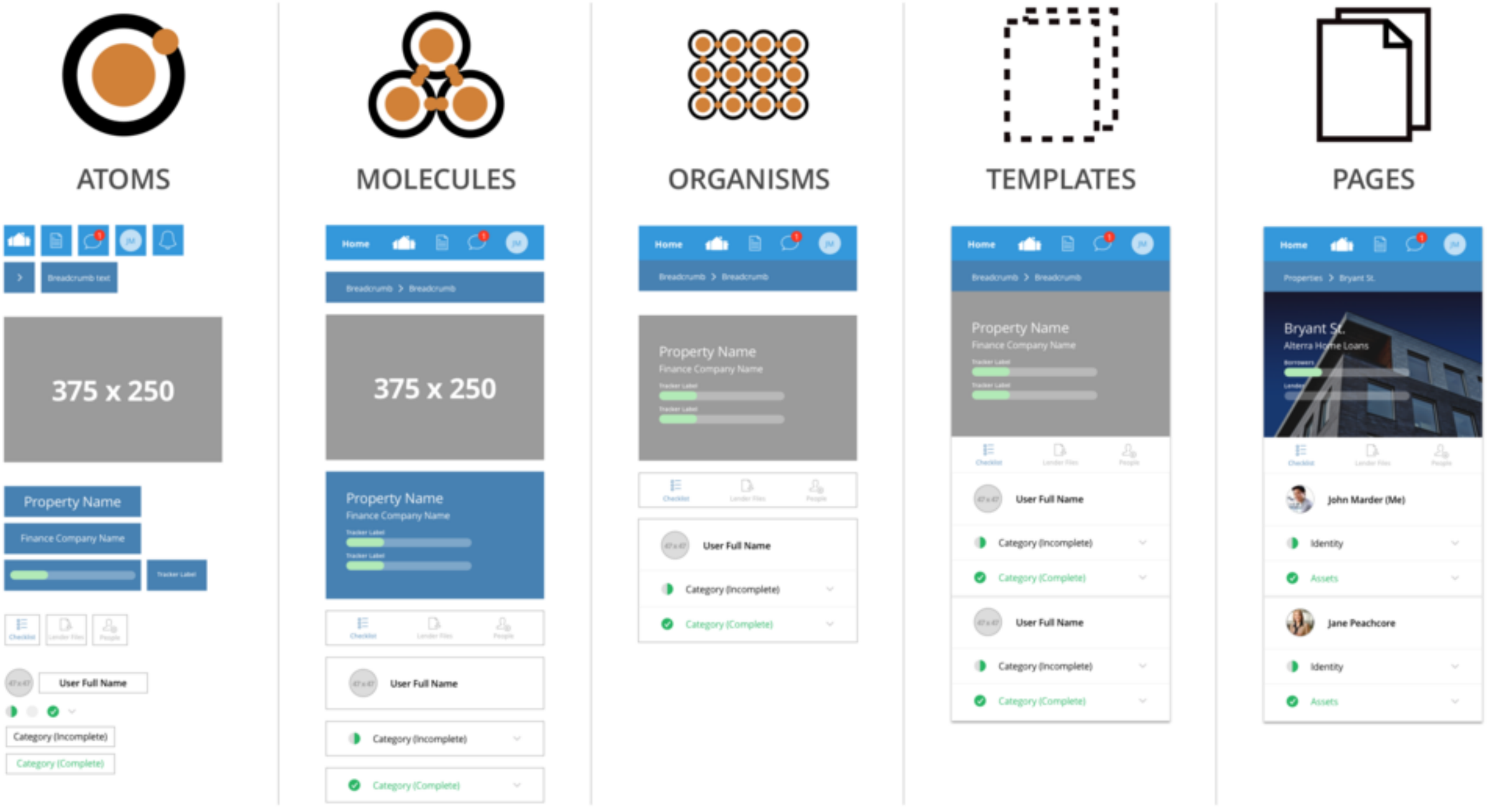


ORGANISMS

TEMPLATES

PAGES

# KOMPOSITION



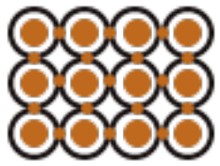
# KOMPONENTEN

ATOMS

ELEMENT

MOLECULES

KOMPOSITION



ORGANISMS

KONTEXT
STATE
SEITENEFFEKTE

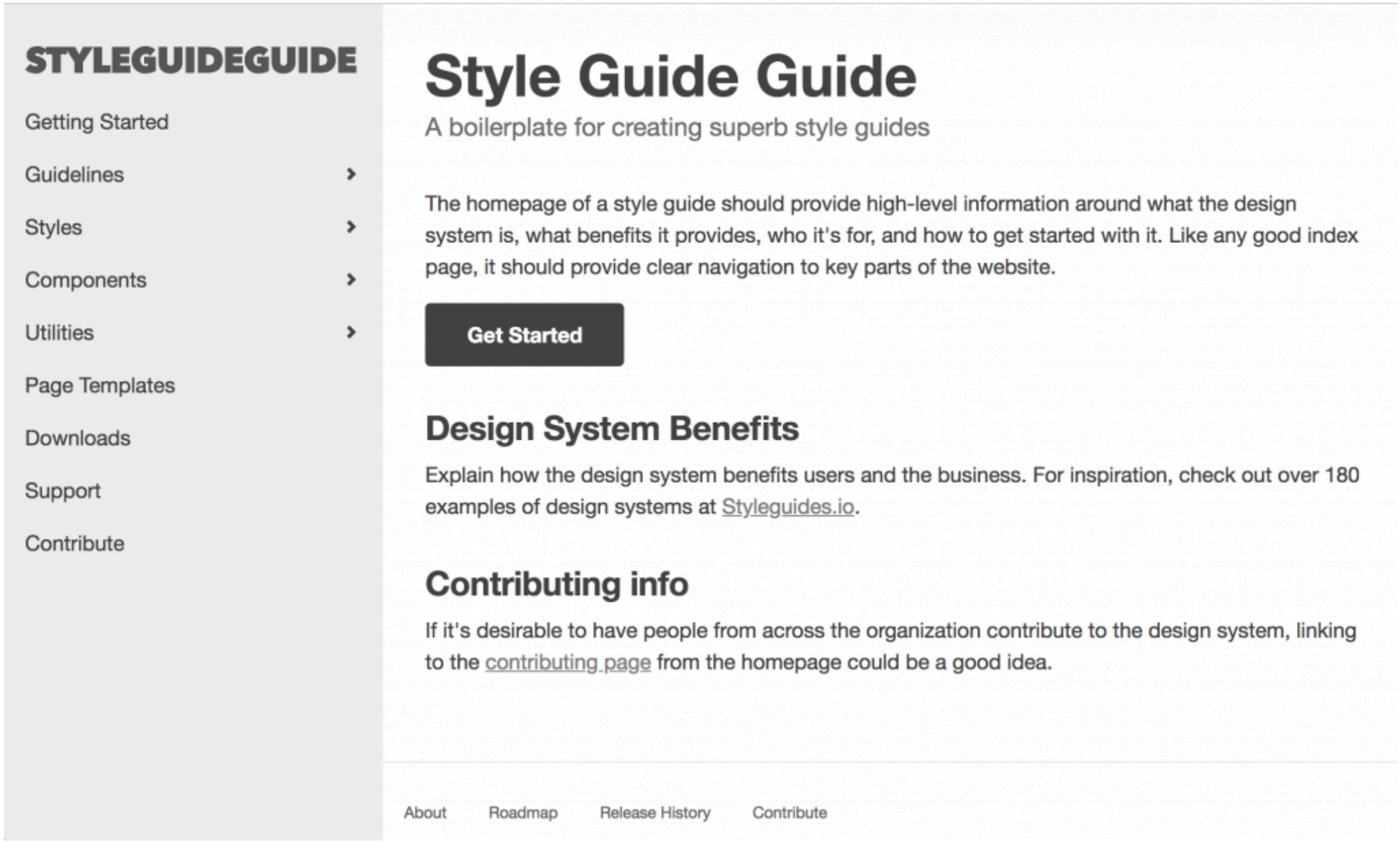
TEMPLATES

LAYOUT & SKELETON

PAGES

INTEGRATION

# STYLE GUIDE GUIDE



# TOOLS

- UI Environment/Playground - **Storybook** (https://storybook.js.org/)
- Design Systeme wie **Material-UI, Fabric-UI, Bootstrap, etc.**
- Erstellen von Style Guides - z.B. https://bradfrost.github.io/gatsby-style-guide-guide

# ZUSAMMENFASSUNG

- Schwerpunkt und Ziel liegt auf UI/UX Komponentenorientierung
- Isolation, Struktur und Organisation von Komponenten
- Modularer Aufbau von Komponenten
- An den Grundbegriffen der Naturwissenschaften orientiert
- Entwurf und Umsetzung von Komponenten-APIs

# BEISPIELE

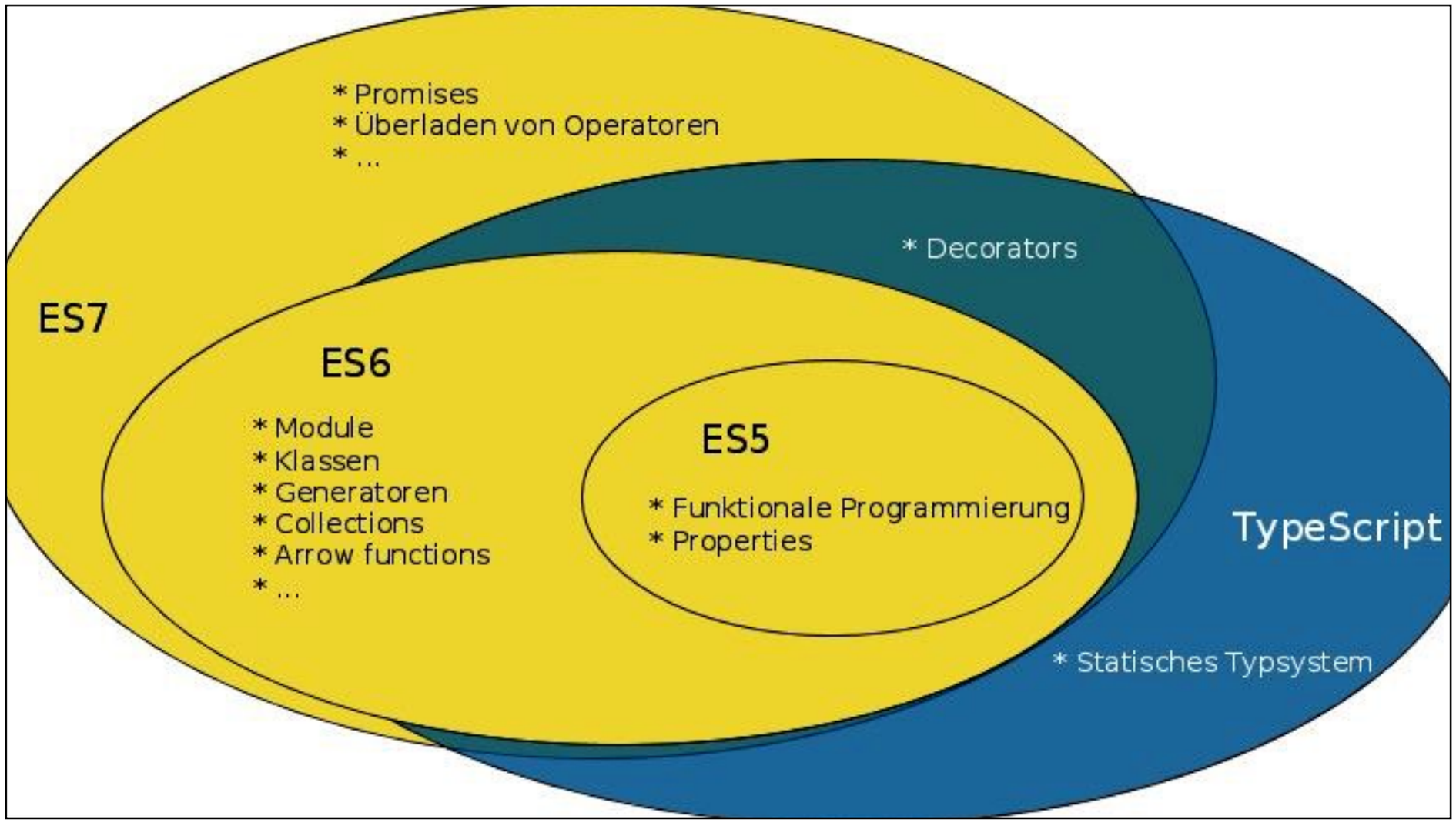
- Material-UI (https://material-ui.com)
- Fabric-UI (https://developer.microsoft.com/en-us/fabric)

# FRAGEN?

# JAVASCRIPT

# ECMASCRIPT

# TYPESCRIPT

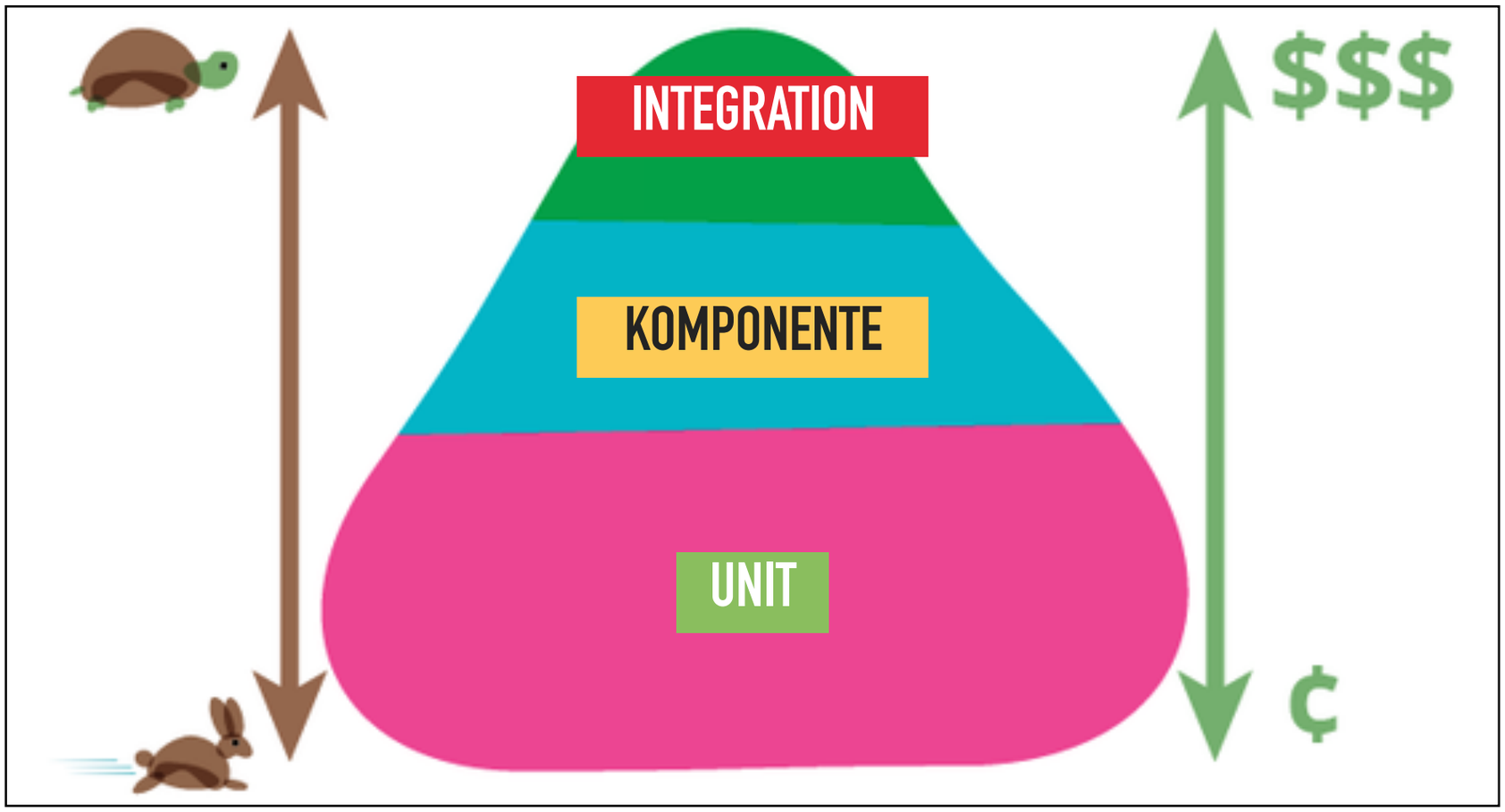

* Promises
* Überladen von Operatoren
* ...
* Decorators
ES7
ES6
* Module
* Klassen
* Generatoren
* Collections
* Arrow functions
* ...
ES5
* Funktionale Programmierung
* Properties
TypeScript
* Statisches Typsystem

# FRAGEN?

# TESTING

GESCHWINDIGKEIT
AUFWAND

KOSTEN
WERT



# REACTJS TEST-TOOLS

## UNIT TESTS

JEST

## COMPONENT TESTS

JEST

## UI/UX TESTS

PUPPETEER