

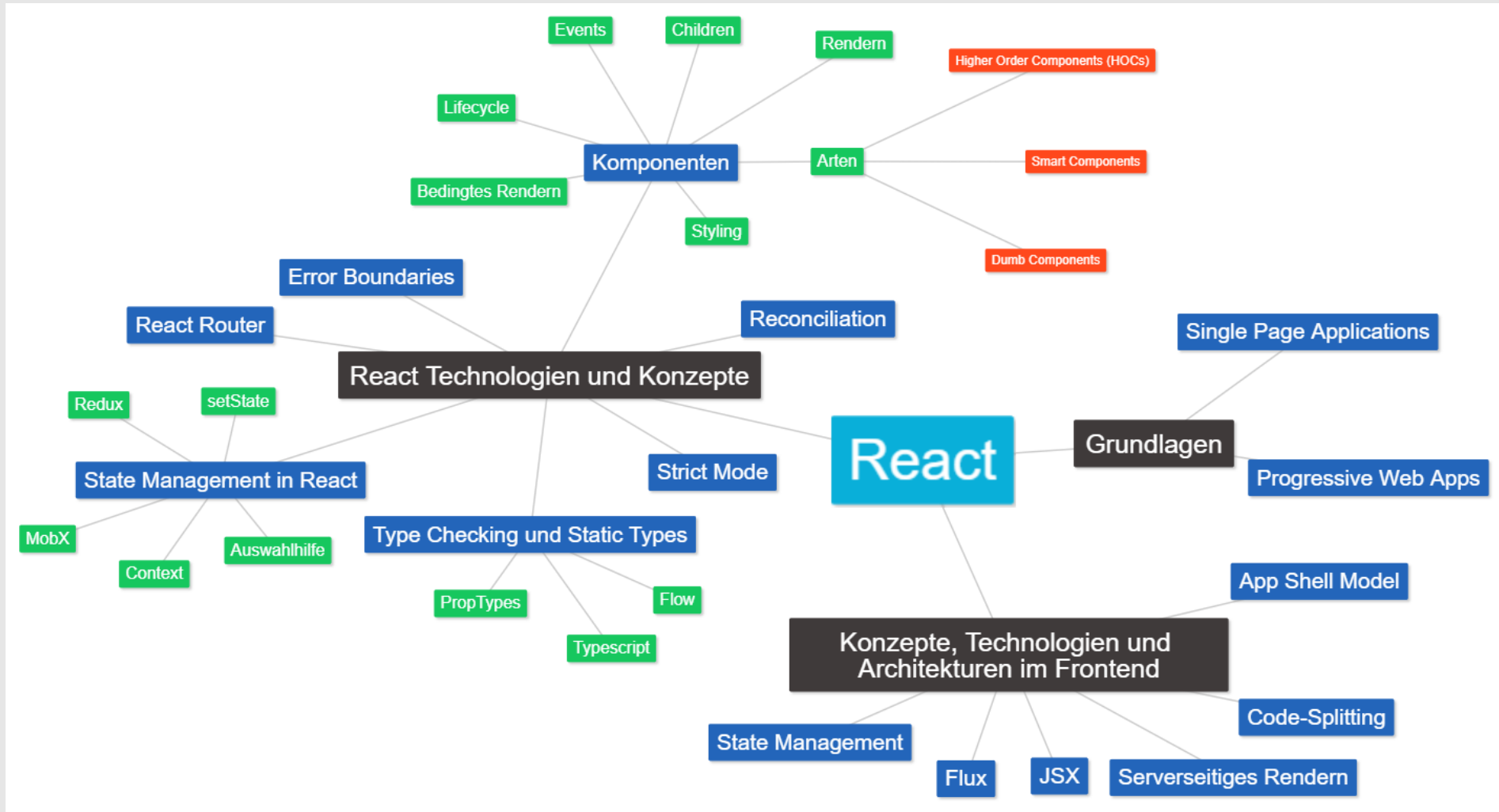
# Spezielle Gebiete zum Software Engineering

## React

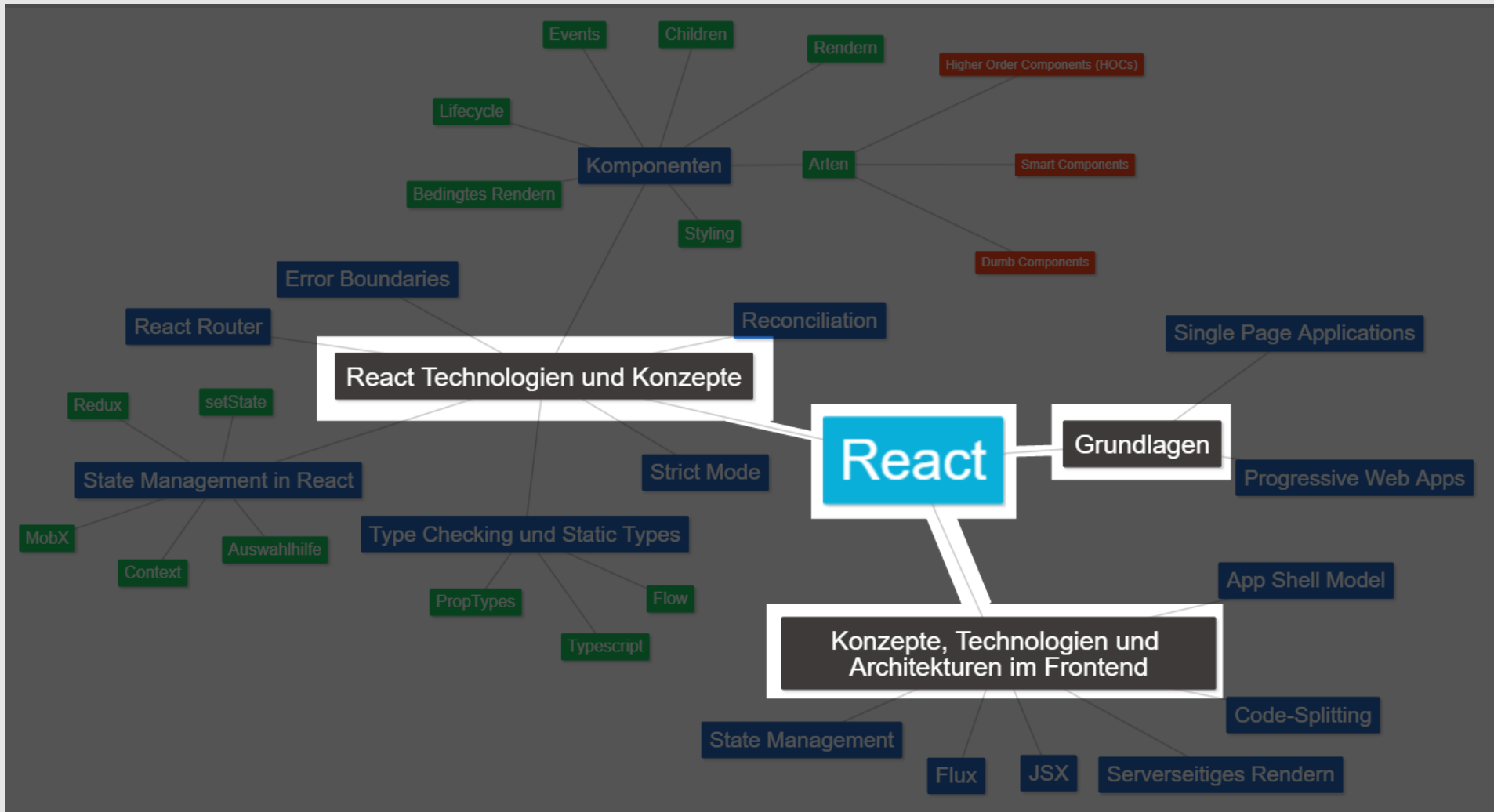
Patrick Vogt

09.07.2018

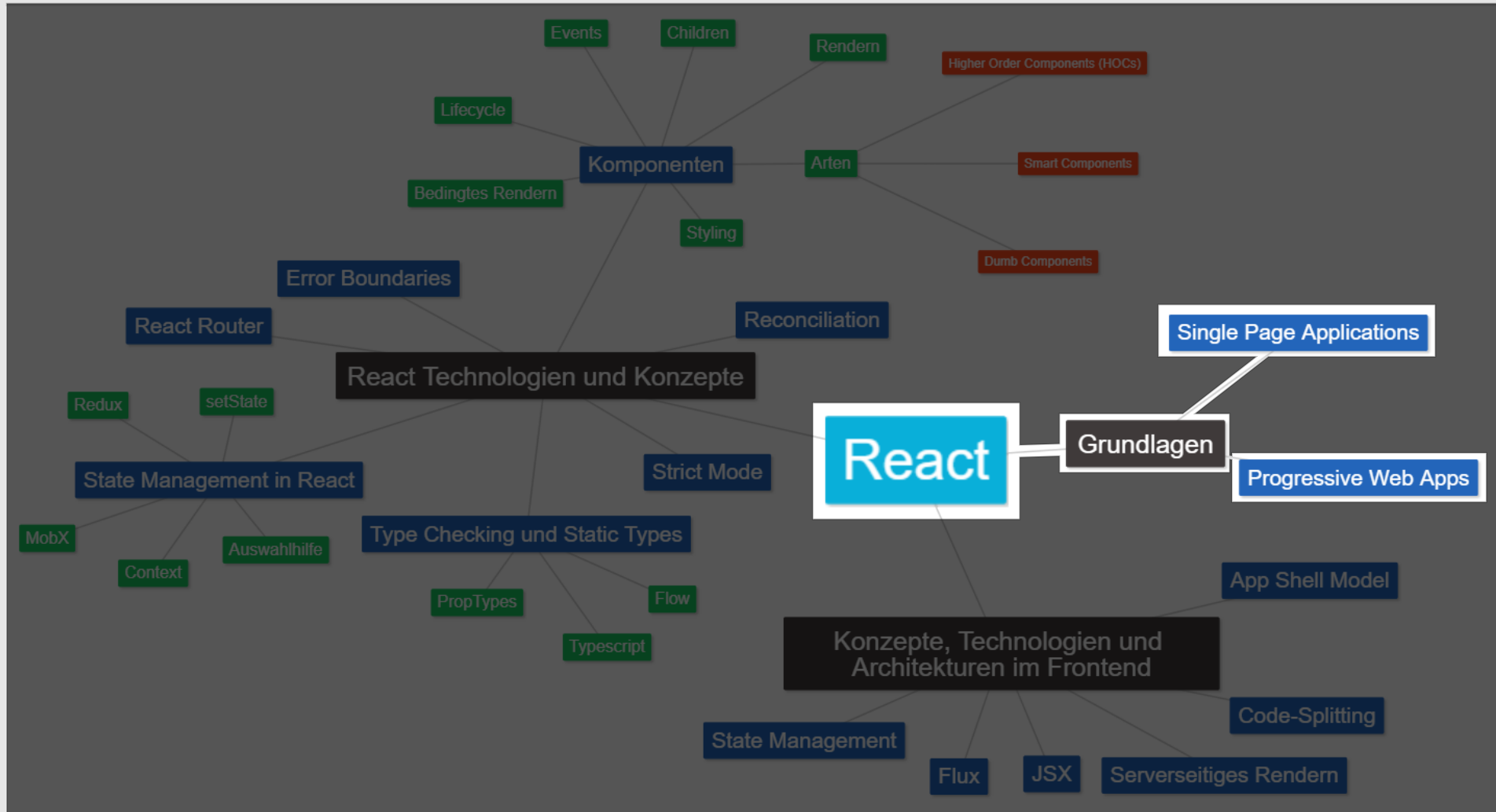
# Agenda



# Agenda

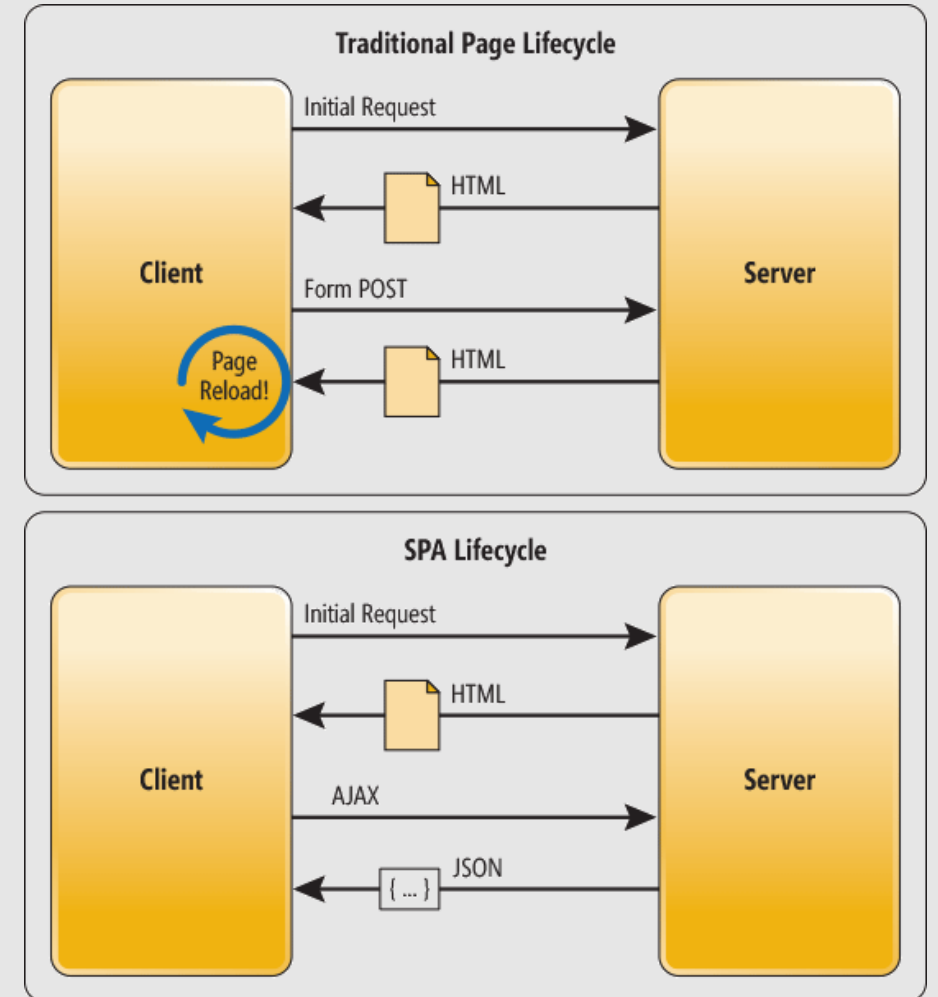


# Agenda



# Single Page Applications

- AJAX → (meist) JSON statt HTML Anfragen
- Kein erneutes Laden der Seite
- UI-Interaktionen beim Client (JS & CSS)



AJAX: Asynchronous JavaScript and XML

Quelle: <https://msdn.microsoft.com/en-us/magazine/dn463786.aspx>

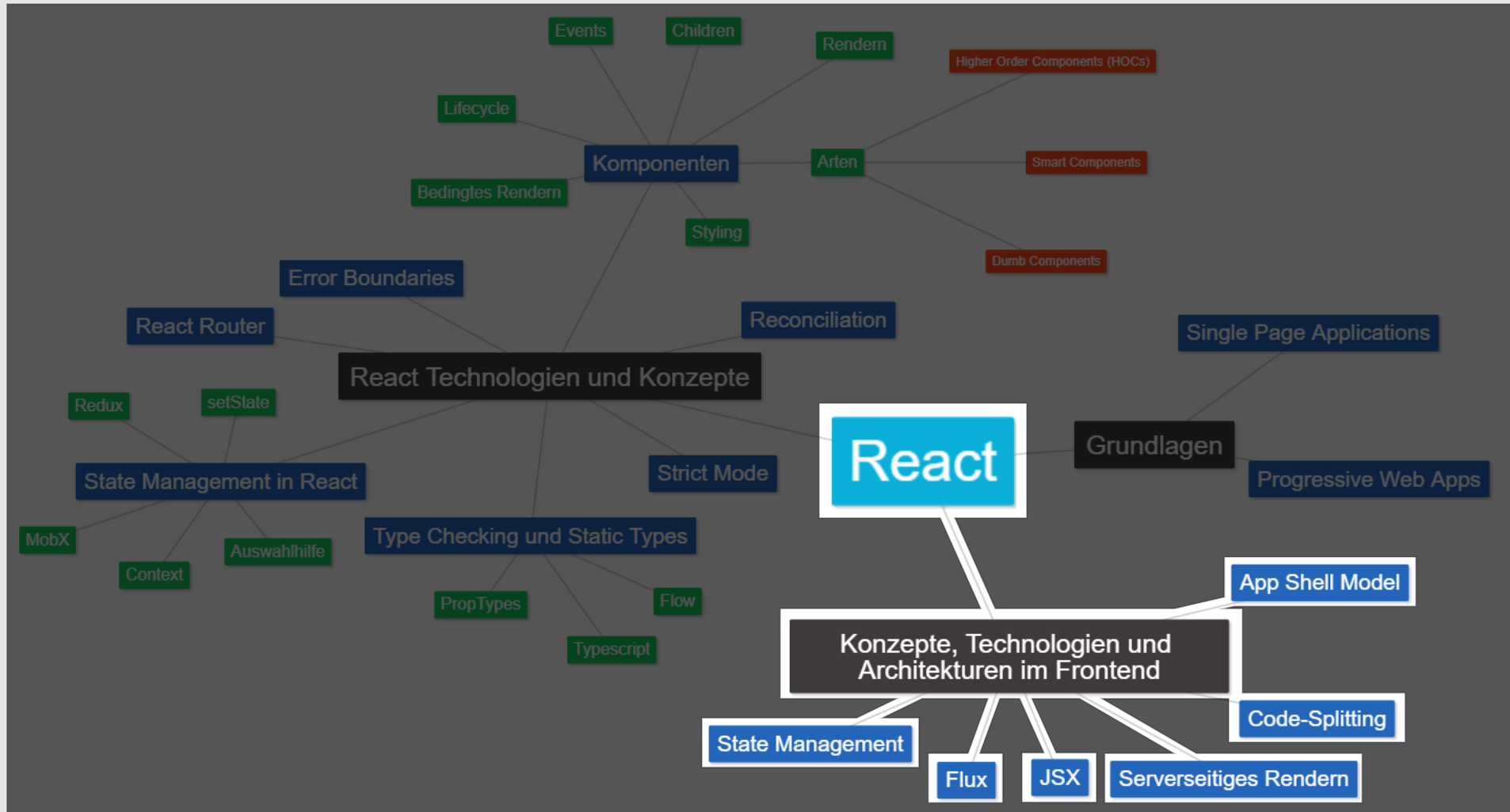
# Progressive Web App (PWA)

- Erscheinen dem Nutzer als native Anwendung
- **Reliable**: Laden unabhängig vom Netzwerkstatus
- **Fast**: Schnelle Ladezeiten
- **Engaging**:
  - Ohne App Store installierbar
  - Vollbild
  - Können Push Notifications versenden
- Checkliste/Tool: <https://developers.google.com/web/progressive-web-apps/checklist>



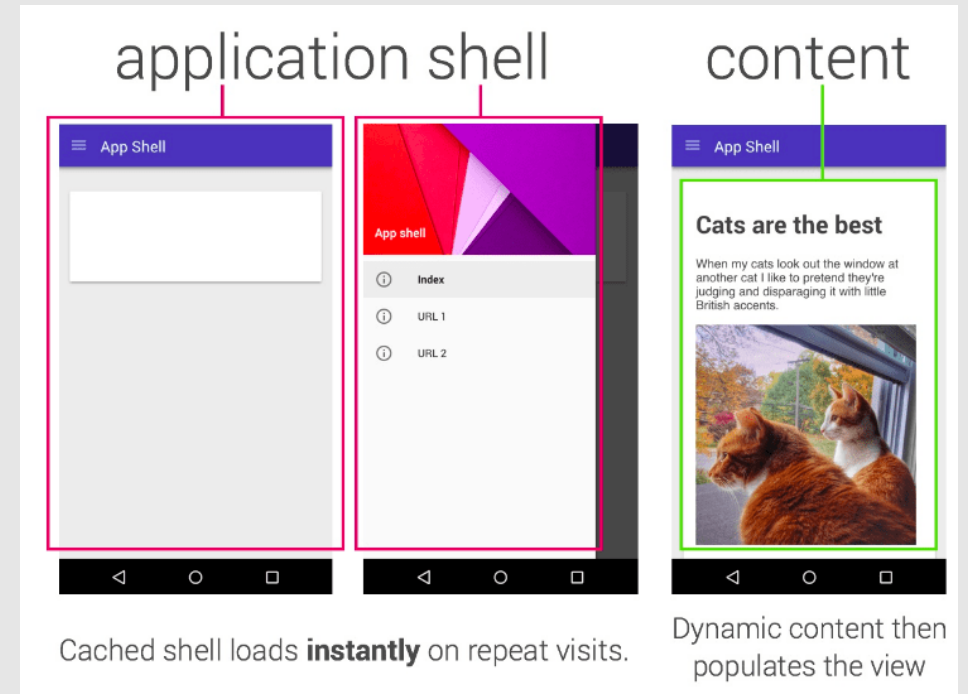
Quelle: <https://hackernoon.com/progressive-web-apps-the-next-step-in-web-app-development-372235bf9a99>

# Agenda



# App(lication) Shell Model

- Architektur zur Erstellung von zuverlässigen und sofort ladenden PWAs
- **Shell**: Grundgerüst (Offline-Cache)
- **Content**: Seiteninhalt, wird nachgeladen
- Z.B. Service Worker für das Caching
- Gut geeignet, wenn Grundgerüst gleich bleibt



Quelle: <https://developers.google.com/web/fundamentals/architecture/app-shell>

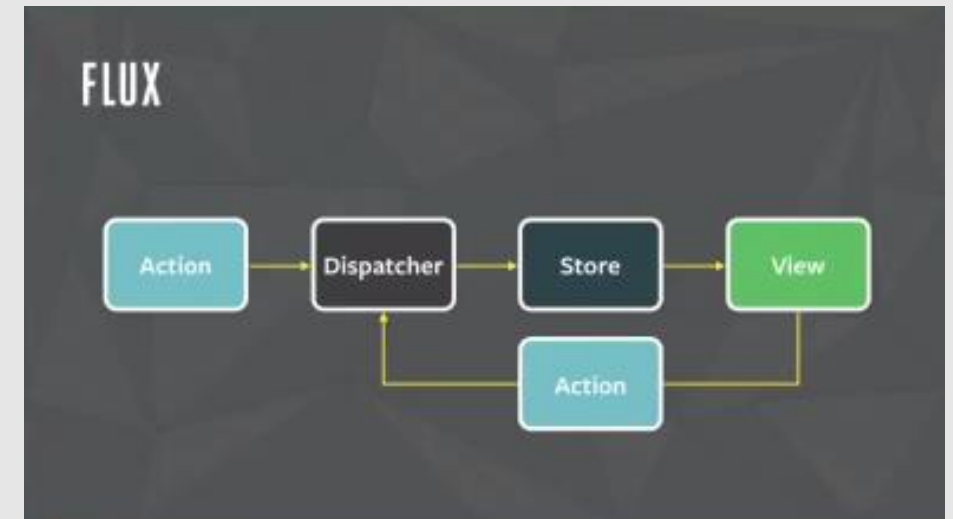
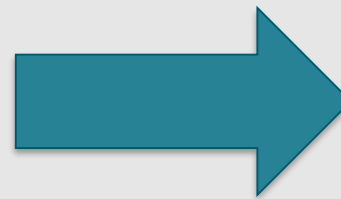
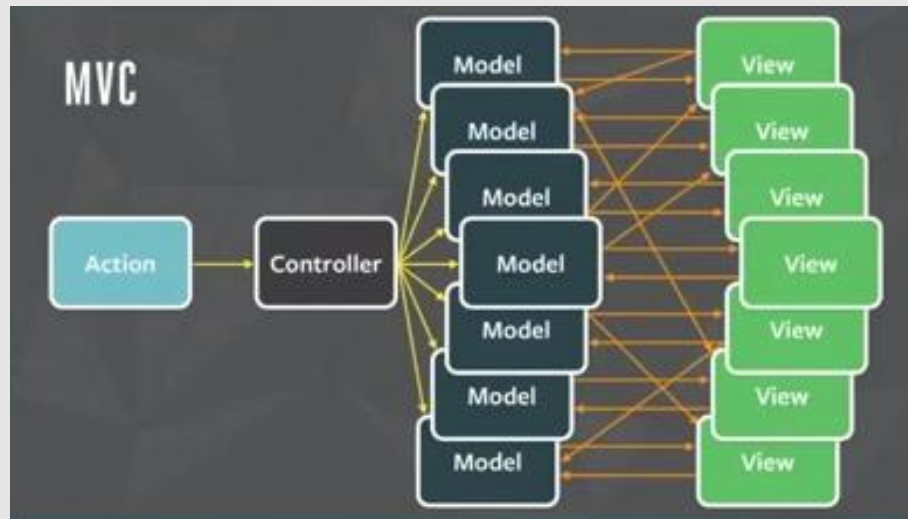


# State Management

- Komplexe Anwendungen → viele Zustände → schwer handhabbar
  - (Fast) Jede Interaktion mit der GUI kann den Zustand ändern
- Sehr wichtiges Thema im Bereich der modernen Frontendentwicklung
- Gängige Frameworks bieten vielerlei Hilfe bei der Bewältigung

# Flux

- Unidirektionales Architektur Pattern
- Erleichtert Überblick über Zustandsänderungen
- 2014 veröffentlicht 
- Große Projekte → MVC Pattern schwer zu beherrschen



Quelle: <https://www.infoq.com/news/2014/05/facebook-mvc-flux>

# Flux

Action: Ereignis/Benachrichtigung

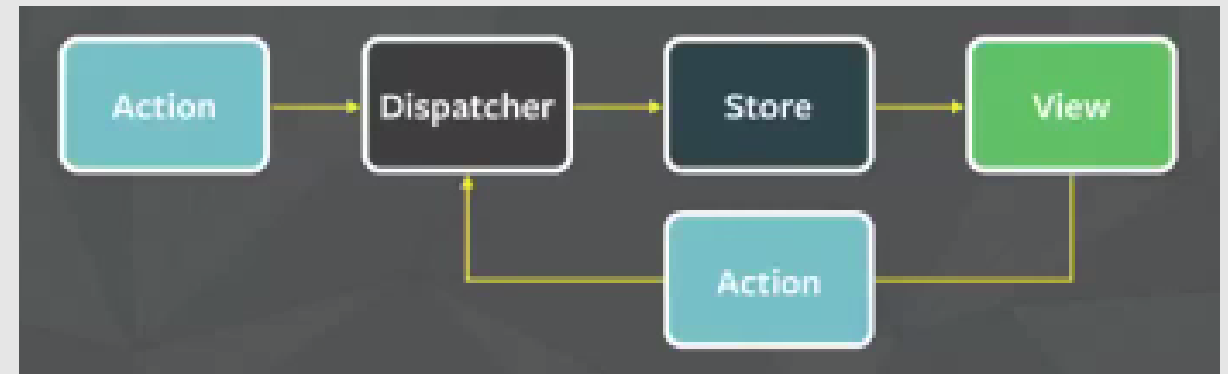


```
js
{
  type: 'ADD_PERSON',
  payload: {
    forename: 'Max'
    surname: 'Mustermann'
  }
}
```

Dispatcher: Verteilt Actions auf Stores

Store: Das „Model“ (aus MVC)

View: Anzeige, abonnieren Stores, lösen Actions aus



Quelle: <https://www.infoq.com/news/2014/05/facebook-mvc-flux>

# JSX

- XML/HTML-artige Struktur in JavaScript
- Beschreibt die Struktur von GUI-Elementen
- Wird in JavaScript übersetzt (z.B. Babel) → Variablen als „Expressions“ einsetzbar: `{var}`

```
function render() {  
  return (  
    <div>  
      <h1>Master Informatik</h1>  
      <h2>Campus Minden</h2>  
      <h3>Modulname:</h3>  
      <input type="text" autoFocus name="moduleName"/>  
    </div>  
  );  
};
```

JSX



```
function render() {  
  return React.createElement("div", null,  
    React.createElement("h1", null, "Master Informatik"),  
    React.createElement("h2", null, "Campus Minden"),  
    React.createElement("h3", null, "Modulname:"),  
    React.createElement("input", {  
      type: "text",  
      autoFocus: true,  
      name: "moduleNameInput"  
    })  
  );  
};
```

js

# Code Splitting

- Tools wie webpack bündeln Web Apps zu einer einzelnen (großen) Datei  
→ Stets alle Daten laden
- Lade nur das, was benötigt wird → **Code-Splitting** (quasi „lazy loading“)
- Unterschiedliche Arten
  - z.B. komponentenbasiert oder routenbasiert

*Beispiel React mit webpack:*

```
import { SubMenu } from './SubMenu';  
  
SubMenu.open();
```

js



```
// Webpack führt hier ein Code-Splitting durch  
import("./SubMenu").then(module => {  
  module.SubMenu.open();  
});
```

js

# Serverseitiges Rendern (SSR)

- **Clientseitig:** Browser lädt einfache HTML Datei, Befüllung durch JS
- **Serverseitig:** **Startinhalt** (HTML) auf Server generieren, Update-Handling im Browser

+ Search Engine Optimization (SEO)

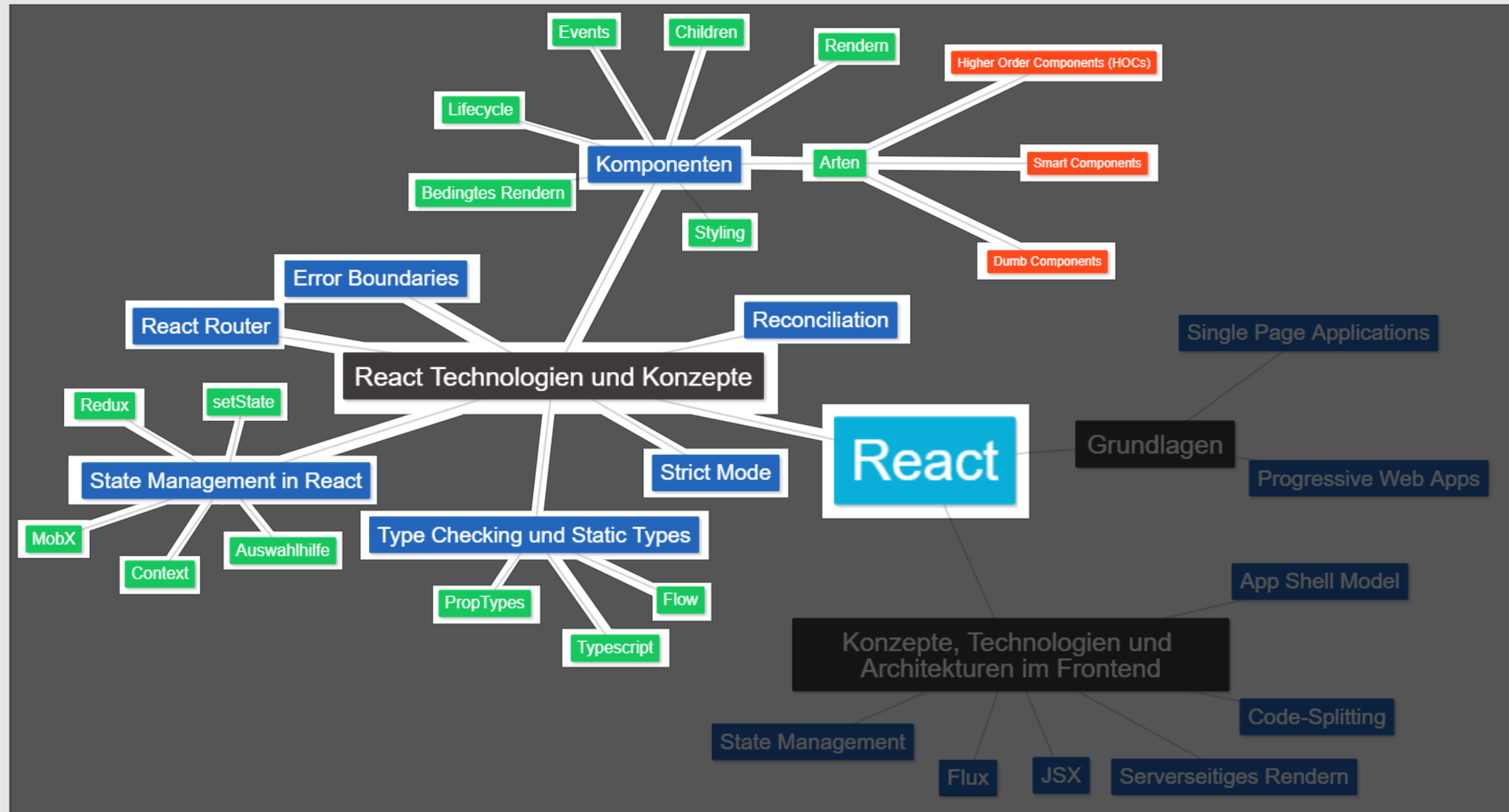
+ (Meist) Bessere clientseitige Performance

- Server stärker beansprucht

- Größe der initialen HTML-Datei steigt (i.d.R. vernachlässigbar)

- Komplexität der Anwendung nimmt zu

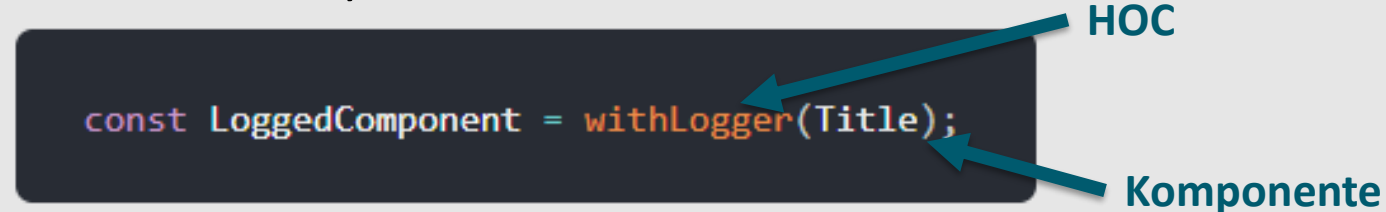
# Agenda



# Komponenten

## Arten

- Dumb Components
  - Zustandslos
  - Ändern Aussehen auf Basis von (konstanten) Properties (*props*)
  - Arrow Function
- Smart Components
  - Verfügen über einen (änderbaren) Zustand (*this.state*) und Properties (*this.props*)
  - Klasse
- Higher Order Components (HOCs)
  - Pattern (vgl. Decorator von GoF)
  - Erweitern Komponenten um zusätzliche Funktionalitäten/Properties
  - Verwendung wie eine Factory Methode



The diagram shows a code snippet in a dark box: `const LoggedComponent = withLogger(Title);`. Two teal arrows point from the text labels to the code. One arrow points from the label 'HOC' to the `withLogger` function, and the other points from the label 'Komponente' to the `Title` argument.

```
const LoggedComponent = withLogger(Title);
```

HOC

Komponente



# Komponenten

## Children

- Komponenten wie gewohnt hierarchisch Strukturiert
- Direkte Nachfahren → *(this.)props.children*

```
const List = (props) => {  
  return (  
    <div>  
      <p>Anzahl an Items: {React.Children.count(props.children)}</p>  
      <ul>  
        {props.children}  
      </ul>  
    </div>  
  )  
}
```

Aufruf

```
<List>  
  <li>Item 1</li>  
  <li>Item 2</li>  
</List>  
  
//=====  
// Ausgabe:  
//=====  
// Anzahl an Items: 2  
//   • Item 1  
//   • Item 2  
//=====
```

# Komponenten

## (Bedingtes) Rendern

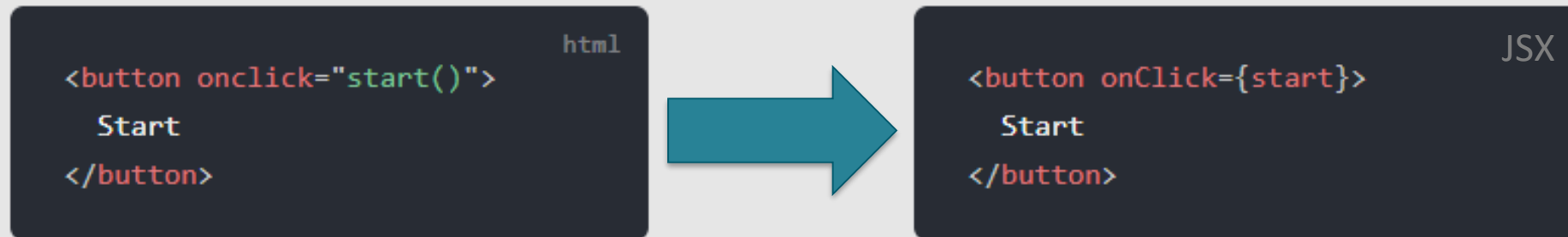
- *render()* dient zum Bestimmen des Aussehens von Smart Components
- Bedingtes Rendern mittels JavaScript möglich

```
...
render() {
  return (
    this.state.isRunning
    ?
    // wird gerendert, falls this.state.isRunning true ist
    <button onClick={this.stop}>
      Stopp
    </button>
    :
    // wird gerendert, falls this.state.isRunning false ist
    <button onClick={this.start}>
      Start
    </button>
  );
}
```

# Komponenten

## Events

- Gewohnte Events vorhanden, jedoch camelCase
- Handler direkt als Expression (JavaScript-Code), statt String
- Achtung mit „Bindung an Klassenkomponente“ → siehe Ausarbeitung



# Komponenten Styling

- CSS
- Standardmäßig global
- In Verbindung mit z.B. webpack auch lokal möglich ([\*CSS-Modules\*](#))

```
CSS

.red-text {
  color: red;
}

#some-id {
  font-size: 12px;
}
```

```
JSX

function someComponent() {
  return <div className="red-text" id="some-id">Roter
    Text</div>;
}
```

# Komponenten Styling

- *style*-Attribut
- JavaScript-Objekt
- Fast wie CSS

camelCase!


```
const redTextStyle = {  
  color: 'red',  
  fontSize: '12px',  
};  
  
// ...  
  
function someComponent() {  
  return <div style={redTextStyle}>Roter Text</div>;  
}
```

JSX

# Komponenten Styling

- „Styled Components“
- JS-Bibliothek
- Erzeugt Komponente mit dem gewünschten Style
- String Interpolation ([\*Tagged Template Literal\*](#))
- CSS-in-JS → Style an Komponente gebunden
- Themes, Style auf Basis von Props, ...

„Tagged Template Literal“

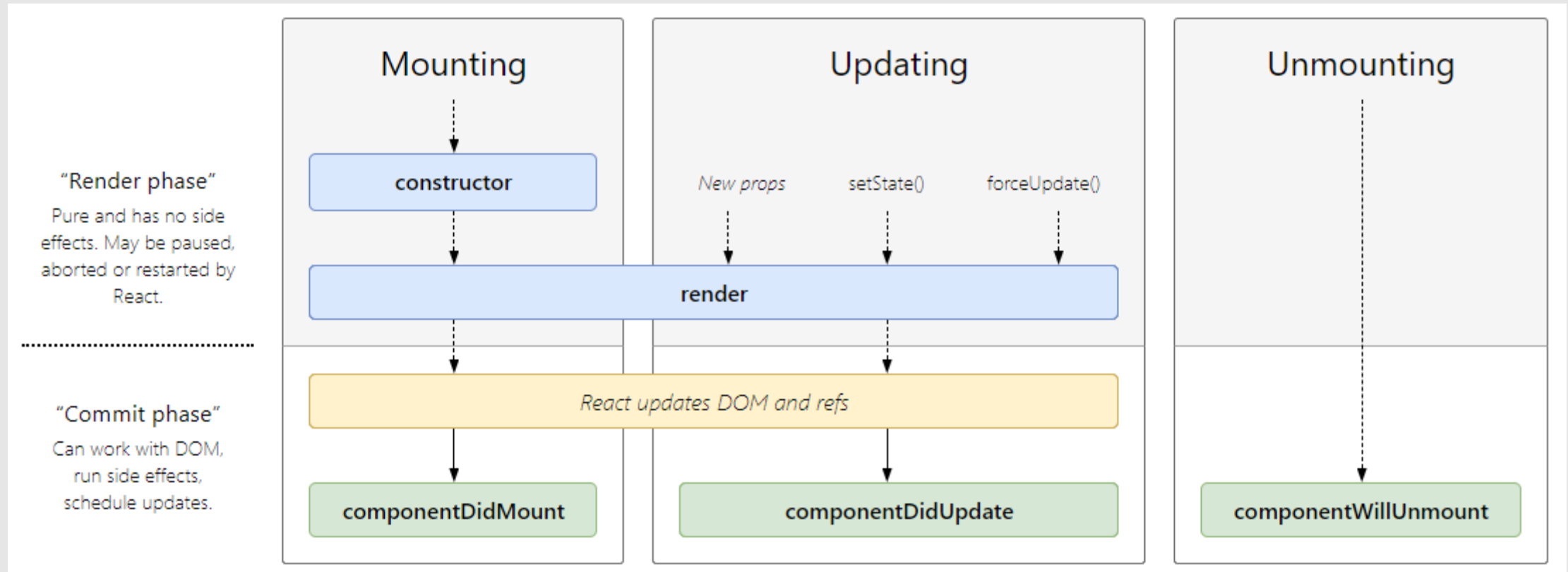


```
JSX
const RedText = styled.div`
  color: red;
  font-size: 12px;
`;

// ...

function someComponent() {
  return <RedText>Roter Text</RedText>;
}
```

# Komponenten Lifecycle (von Smart Components)



Auszug → siehe Ausarbeitung für weitere

Quelle: <http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/>

# State Management in React

- `this.setState()` → verwendet direkt *this.state*, triggert *render()*
- Context → kontextbasierter Provider & Consumer Ansatz
  1. Erzeuge Kontextpaar
  2. a) Erzeuge Provider, der diesen Kontext verwaltet  
b) Abonniere den Kontext mit Komponente

In React enthalten

```
// Erzeugt ein Kontextpaar: {Provider, Consumer}  
// defaultValue wird von Consumern verwendet, die keinen  
// passenden Provider im JSX-Baum "über" sich haben  
const MyContext = React.createContext(defaultValue);
```

js

```
<div>  
  // Kontext setzen  
  <MyContext.Provider value={someContextValue}>  
    ...  
    <MyContext.Consumer> {  
      contextValue => (  
        // Hier das Element rendern, das auf den Wert des  
        // Kontextes angewiesen ist.  
        // Der Kontext kann über "contextValue" gelesen  
        // werden  
      )  
    }  
  </MyContext.Consumer>  
  ...  
</MyContext.Provider>  
</div>
```

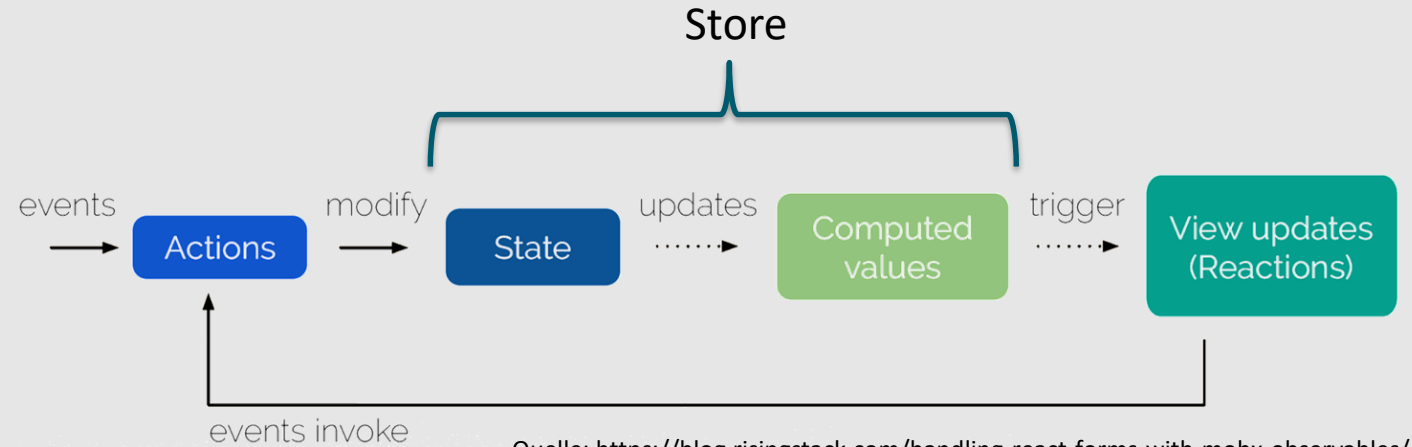
JSX



# State Management in React

## ■ MobX

- JS-Bibliothek
  - Observer & Observables
  - Relativ einfach
- *Siehe SW-Demo*

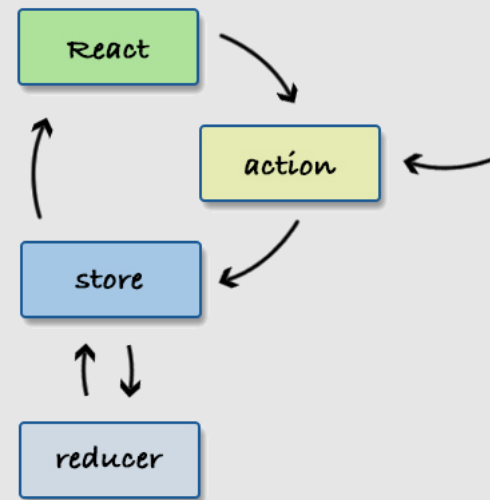


Quelle: <https://blog.risingstack.com/handling-react-forms-with-mobx-observables/>

## ■ Redux

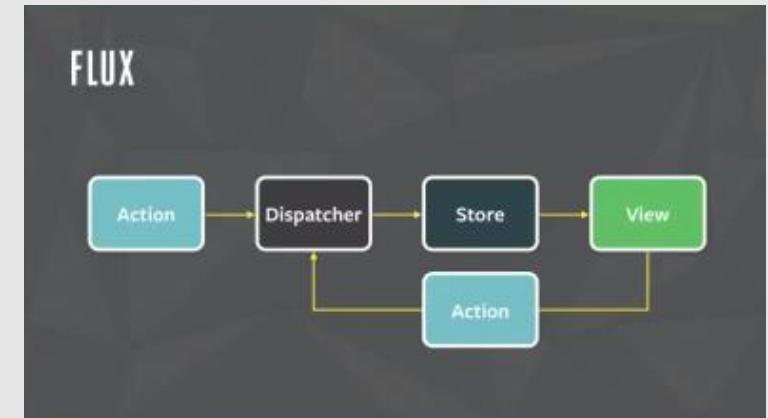
- JS-Bibliothek
- Für komplexe Anwendungen
- Typ. Single-Store
- Immutable state

Ändern Zustand auf Basis  
von akt. Zustand & Action



Quelle: <https://legacy.gitbook.com/book/krasimir/react-in-patterns/details>

Zur Erinnerung:



Quelle: <https://www.infoq.com/news/2014/05/facebook-mvc-flux>

# State Management in React

Soll es **keine externen Abhängigkeiten** geben?

**useState** oder **Context** verwenden



Steht die **Einfachheit** im Vordergrund?

**MobX** verwenden

Ist die **Skalierbarkeit** wichtig?

**Redux** verwenden

# Type Checking und Static Types

- **PropTypes**
  - In React enthalten
  - Prüft Properties
  - Fehlermeldung in Browser-Konsole
- **Flow** 
  - Static Type Checker für JavaScript Applikationen
  - Nicht nur Properties, auch z.B. Typen von Rückgabewerten
- **TypeScript**  **Microsoft**
  - Open Source Programmiersprache
  - Kompiliert zu JavaScript
    - sehr gute Kompatibilität (nur Erweiterung)
  - Generics, Vererbung, Interfaces, Enumerationen uvm.
  - Zugriffsmodifizierer (public, protected, private)

Komponente

```
Header.propTypes = {  
  title: PropTypes.string.isRequired, // zwingend notwendig  
  children: PropTypes.element        // optional  
};
```

```
// @flow  
function myLoggerFunction(text: string): void {  
  console.log(text)  
}  
  
myLoggerFunction(2); // Fehler
```

```
interface TimerInterface {  
  setTimer(value: number): void;  
  start(): void;  
  stop(): void;  
  onElapsed(): void;  
  isRunning(): boolean;  
}  
  
/// Implementierung analog wie z.B. in Java  
class Timer implements TimerInterface {  
  setTimer(value: number): void {  
    // ...  
  }  
}
```

# Reconciliation

- Update des nativen DOMs relativ ineffizient → virtuelles DOM
- JavaScript Objekt, das den DOM repräsentiert (in-memory)
- *setState()* erstellt virtuellen DOM neu
- Möglichst nur geänderte Elemente austauschen
- Vergleich mit alten DOM → mind.  $O(n^3)$
- Heuristik:  $O(n)$  → **Reconciliation**

# Reconciliation

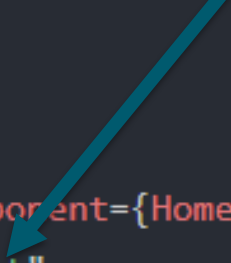
## 2 Vereinbarungen:

1. Annahme: Elemente unterschiedlichen Typs erzeugen unterschiedliche DOM-Trees
  - Breitensuche (ebenenweise)
  - Vergleiche Elemente (neu vs. alt)
  - Unterschiedlicher Typ? Markiere als „dirty“ (werden neu gerendert)
2. Elemente einer Liste können einen „Key“ erhalten:  
Vergleiche die alte Liste mit der neuen Liste:
  - Durchlaufe alle Elemente
    - Key in alter, aber nicht in neuer Liste vorhanden?  
→ rufe `unmount` für die Komponente auf
    - Key in neuer, aber nicht in alter Liste vorhanden?  
→ rufe `mount` Komponente auf
    - Key in beiden Listen vorhanden?  
→ rufe `shouldComponentUpdate` auf, um zu entscheiden, was passieren soll

# React Router

- SPA mit mehreren Seiten → Lesezeichen, Navigation über „Browserpfeile“, SSR  
→ *BrowserRouter*
- Statischer Content → *HashRouter*, verwendet Text hinter „#“ in URL als Parameter

Wird der Komponente als Property „text“ übergeben



```
ReactDOM.render((  
  <BrowserRouter>  
    <Switch>  
      <Route exact path="/" component={Home} />  
      <Route path="/Display/:text" component={DisplayText} />  
      <Route component={Default} />  
    </Switch>  
  </BrowserRouter>  
, document.getElementById('root'))
```

# Error Boundaries & Strict Mode

## Error Boundaries

- Fangen von Fehlern innerhalb der Kindelemente
- Konstruktor + Lifecycle-Methoden
- Einfaches Einbinden über Lifecycle Methode

```
componentDidCatch(error, errorInfo) {}js
```

## Strict Mode

- Einhalten von Best Practices
- Prüft Kindelemente

```
<StrictMode>  
  Kindelemente  
</StrictMode>
```

```
✖ Warning: Unsafe lifecycle methods were found within a strict-mode tree: warning.js:33  
  in Application  
  
  componentWillMount: Please update the following components to use componentDidMount  
  instead: Main  
  
  Learn more about this warning here:  
  https://fb.me/react-strict-mode-warnings
```

Quelle: <https://medium.com/@baphemot/whats-new-in-react-16-3-d2c9b7b6193b>

# SW-DEMO



# Fragen