



NILS HARTMANN

Single-Page-Anwendungen mit

# React

Eine praktische Einführung

Slides: <https://nils.buzz/conceptpeople-react>

# NILS HARTMANN

nils@nilshartmann.net

**Freiberuflischer Entwickler, Architekt, Trainer aus Hamburg**

Trainings, Workshops und Beratung

Java

JavaScript, TypeScript

React

GraphQL

**[HTTPS://NILSHARTMANN.NET](https://nilshartmann.net)**

# React

Grundlagen, fortgeschrittene Techniken und Praxistipps

**2. Auflage, Dezember 2019**



**[HTTPS://REACTBUCH.DE](https://reactbuch.de)**

<https://reactjs.org>

- Minimales API (?)
- Minimales Feature Set

<https://reactjs.org>

- Minimales API (?)
  - Minimales Feature Set
- 👉 Man kann (muss ?) viele Entscheidungen selber treffen
- 👉 Sehr gut integrierbar in vorhandenen Technologie Stack

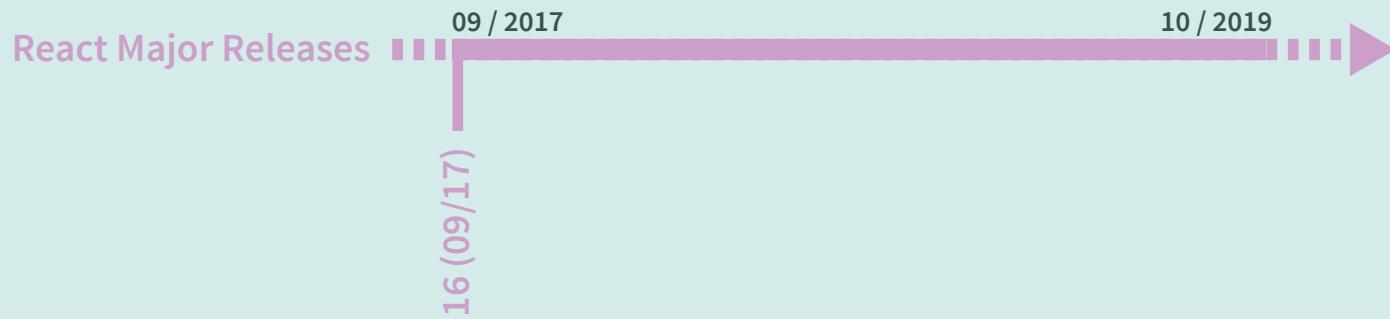
## <https://reactjs.org>

- Minimales API (?)
- Minimales Feature Set
- 👉 Man kann (muss ?) viele Entscheidungen selber treffen
- 👉 Sehr gut integrierbar in vorhandenen Technologie Stack
- Bewusste Verstöße gegen Best Practices

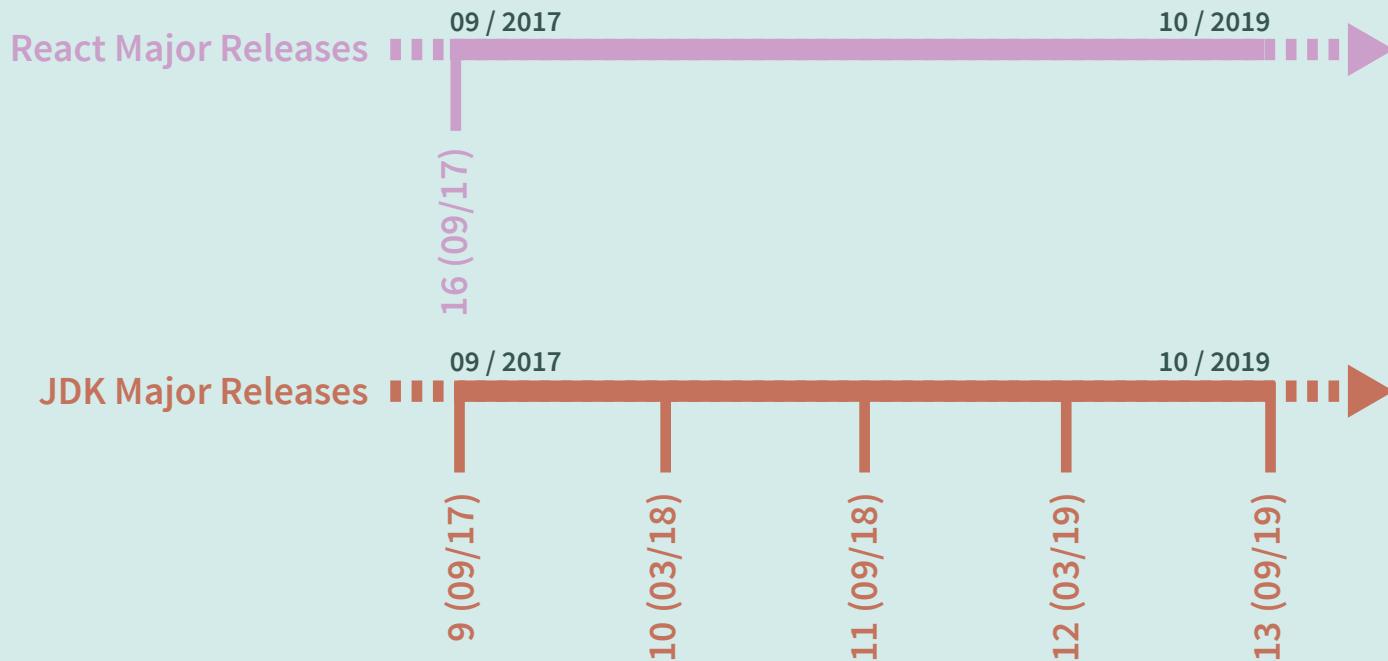
## React ist sehr stabil

- Erste 16er-Version im September 2017
  - Seitdem nur Minor-Versionen
  - Trotzdem sehr viele neue Features, inklusive neuer Hooks API
- 👉 Gut geeignet für langlaufende Anwendungen

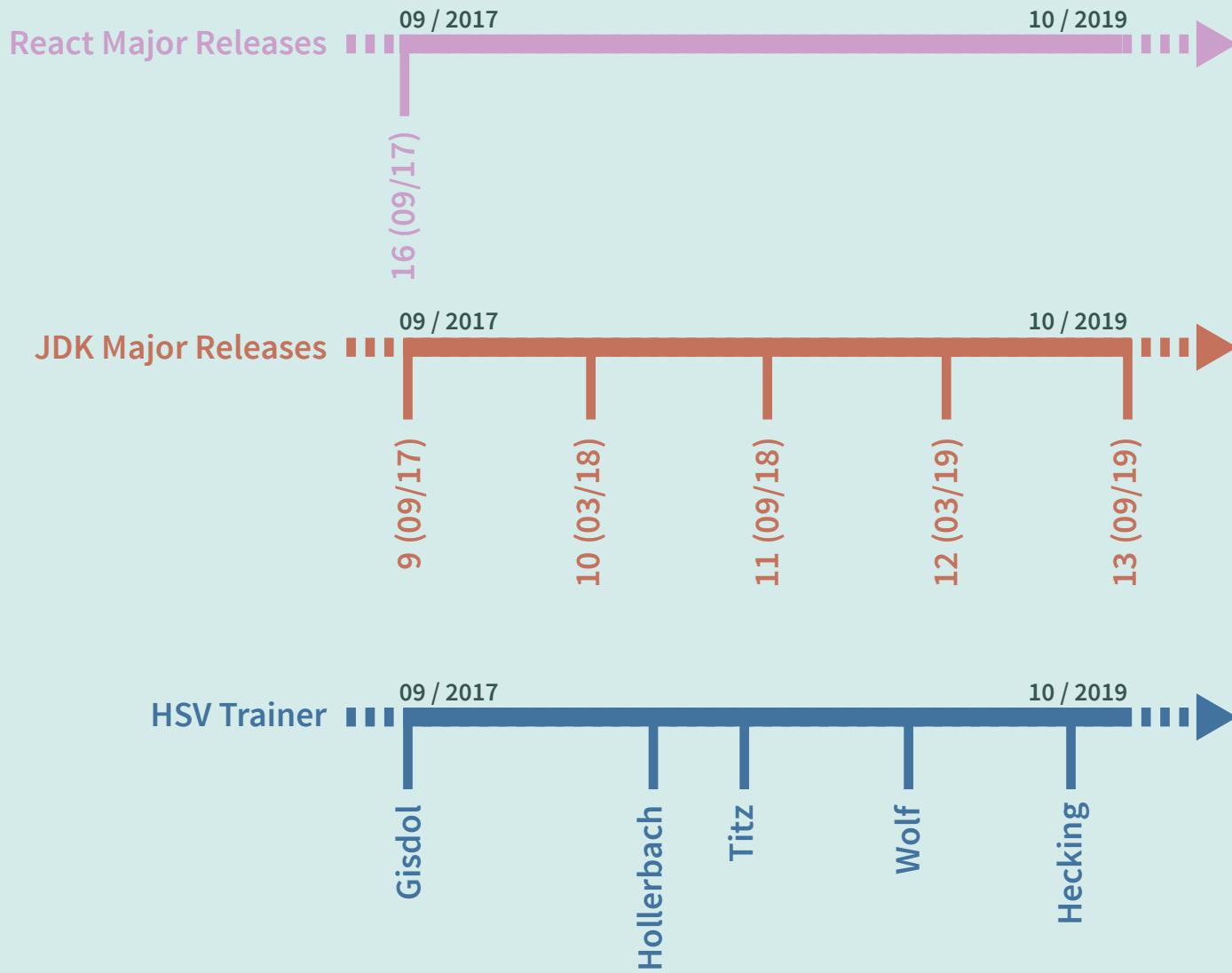
# REACT - STABILITÄT



# REACT - STABILITÄT



# REACT - STABILITÄT



# Greeting App

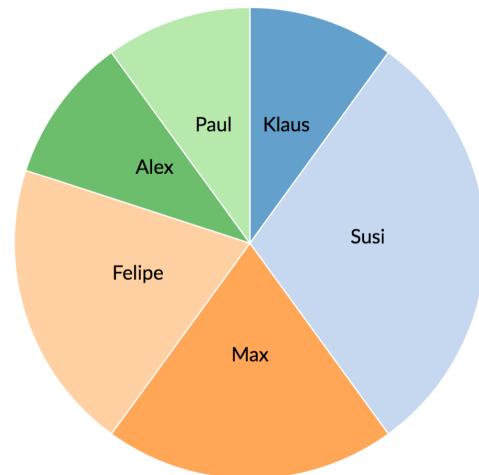
Showing 10 of 10 Greetings

Name	Greeting
Klaus	Moin
Susi	Hello!
Max	Bonjour
Susi	How are you?
Max	Bon soir
Felipe	Hola, ¿qué tal?
Alex	Happy Birthday
Felipe	¡buenos días
Paul	Wie gehts?
Susi	Have a nice day

(All greetings are shown. Click a row to filter)

Add

● Klaus   ● Susi   ● Max   ● Felipe   ● Alex   ● Paul



## BEISPIEL: DIE GREETING APP

# RETHINKING BEST PRACTICES

Klassische Aufteilung

Logik, Model  
(JS)



View  
(HTML, Template)



Gestaltung  
(CSS)



Aufteilung in Komponenten

Button



Eingabefeld



Label



Grafik Inspiriert von: [https://pbs.twimg.com/media/DCXJ\\_tjXoAAoBbu.jpg](https://pbs.twimg.com/media/DCXJ_tjXoAAoBbu.jpg)

## KOMPONENTEN

## React-Komponenten

- bestehen aus Logik und UI
- keine Templatesprache
- werden deklarativ beschrieben
- werden immer komplett gerendert
- können auf dem Server gerendert werden



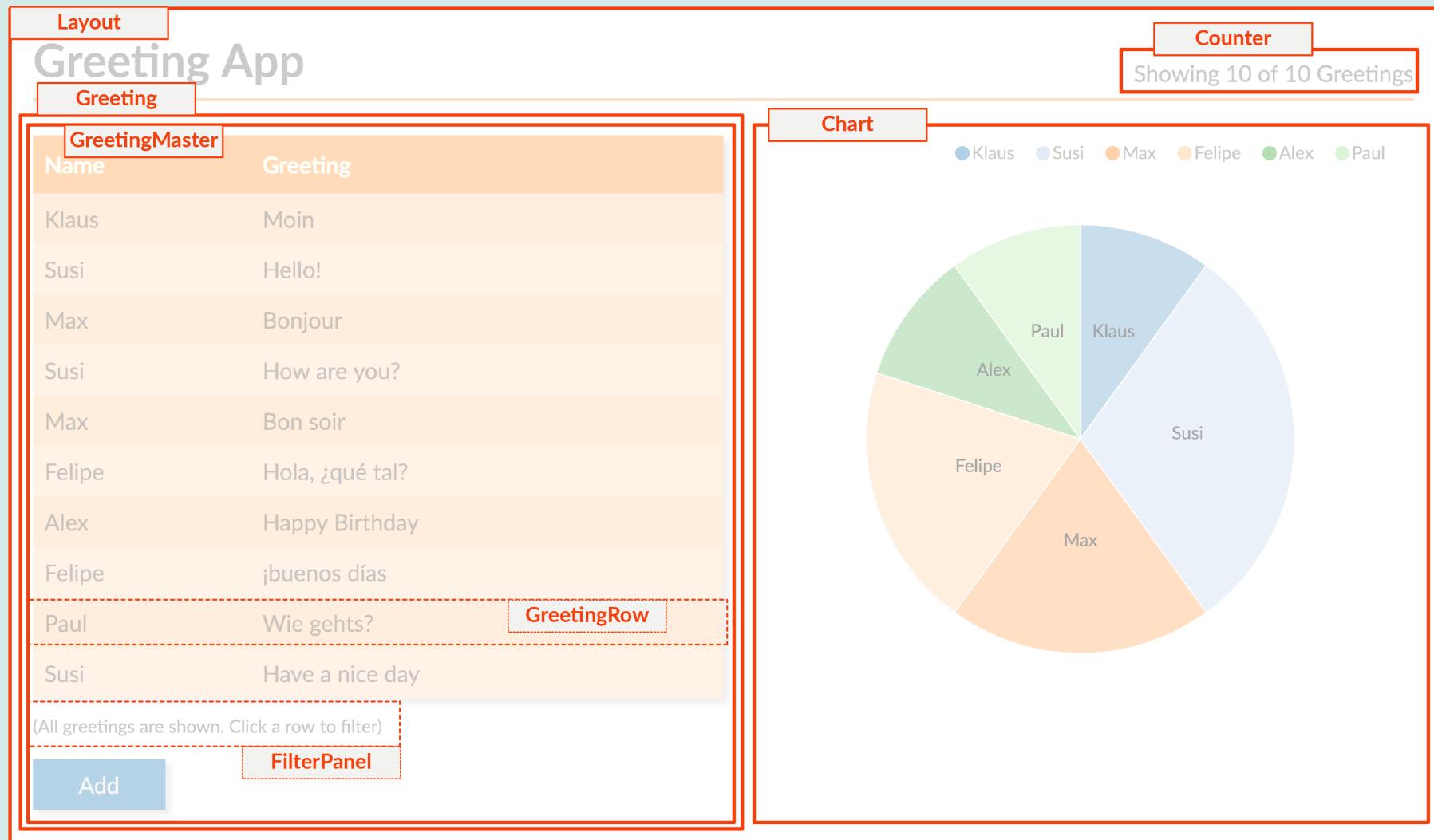
Button



Eingabefeld



Label



## GREETING APP: KOMPONENTEN

# React

PRAKTISCH

# EINE EINFACHE REACT KOMPONENTE

Komponente

Showing 3 of 11 Greetings

# EINE EINFACHE REACT KOMPONENTE

Komponente

Showing 3 of 11 Greetings

Komponenten sind JavaScript-Funktionen

```
function Counter() {
```

```
}
```

# EINE EINFACHE REACT KOMPONENTE

Komponente

Showing 3 of 11 Greetings

```
function Counter() {  
  return <div>Showing 3 of 11 Greetings</div>  
}  
          ^  
          |  
          +-----+  
          | UI in JavaScript ("JSX") 😱  
          +-----+
```

# EINE EINFACHE REACT KOMPONENTE

Komponente

Showing 3 of 11 Greetings

Properties

```
function Counter({filtered, total}) {
```

```
}
```

# EINE EINFACHE REACT KOMPONENTE

Komponente

Showing 3 of 11 Greetings

```
function Counter({filtered, total}) {  
  return filtered === total ?  
    <div>Showing all {total} Greetings</div>  
  :  
    <div>Showing {filtered} of {total} Greetings</div>  
}
```

# EINE EINFACHE REACT KOMPONENTE

Komponente

Showing 3 of 11 Greetings

Counter.js

```
function Counter({filtered, total}) {  
  return filtered === total ?  
    <div>Showing all {total} Greetings</div>  
  :  
    <div>Showing {filtered} of {total} Greetings</div>  
}
```

Verwendung

```
<Counter filtered={3} total={11} />
```

# KOMPONENTEN WERDEN ZU APPLIKATIONEN AGGREGIERT

App.js

```
import Counter from './Counter';
import Greeting from './Greeting';
import Chart from './Chart';

function App() {
  return
    <div className="Main">
      <div className="Title">
        <Counter filtered={3} total={11} />
      </div>
      <div className="Left">
        <Greeting />
      </div>
      <div className="Right">
        <Chart />
      </div>
    </div>
}
```

# KOMPONENTE EINBINDEN

index.html

```
<html>
  <head> . . . </head>
  <body>
    <div id="root"></div>
  </body>
  <script src="dist/index.js"></script>
</html>
```

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';

import App from './App';

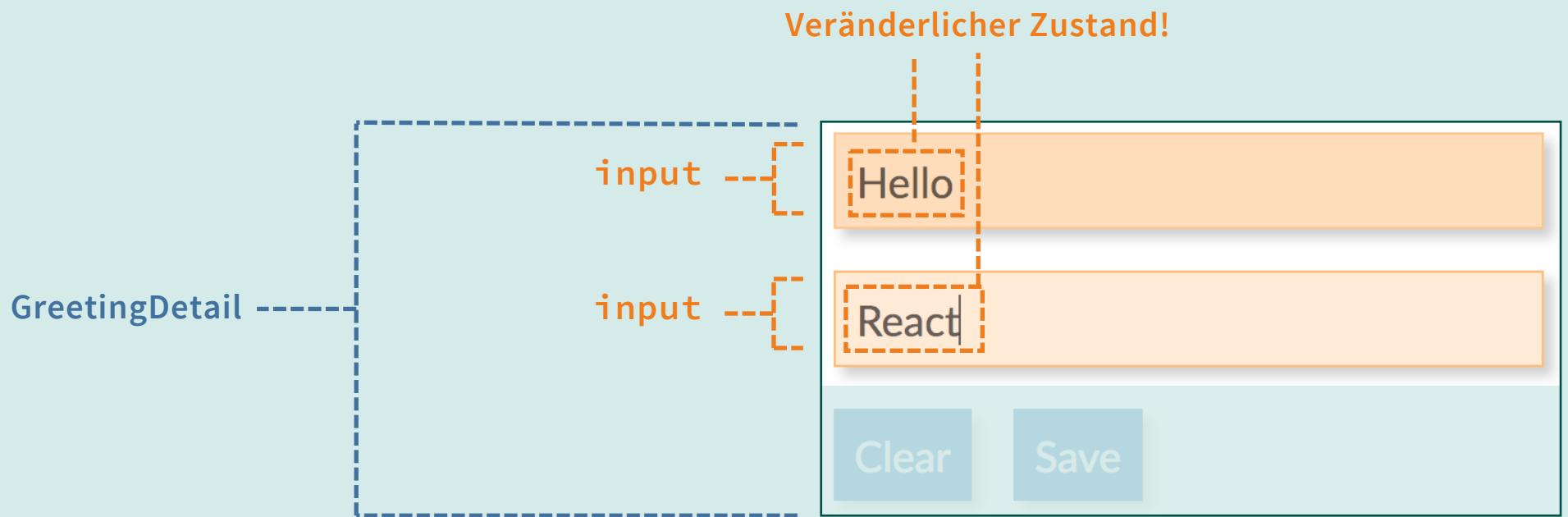
ReactDOM.render(
  <App />,
  document.getElementById('root')
);
```

Model a.k.a

State

ARBEITEN MIT VERÄNDERLICHEN DATEN

## BEISPIEL: EINGABEFELD



# REACT HOOKS API

**Hooks:** Neue React API seit Anfang 2019

- Zugriff auf Zustand, Lebenszyklus etc. aus Funktionskomponenten

**Hooks sind reguläre Funktionen, aber...**

- **nur in** Funktionskomponenten (oder anderen Hooks) erlaubt
- **müssen** auf Top-Level-Ebene stehen (nicht in Schleife, if, ...)
- **müssen** mit "use" benannt werden

## BEISPIEL: EINGABEFELD

Komponente

Hello

-- input

"Hook"-Funktion

```
function GreetingDetail() {  
  const [ phrase, setPhrase ] = React.useState("Hello");
```

|

Aktueller State

|

Setter

|

Initialer Wert

```
}
```

- Mit useState wird ein "Model" erzeugt (State in React)
- Es gibt kein zwei-Wege-Databinding

## BEISPIEL: EINGABEFELD

Komponente

Hello

]}-- input

"Hook"-Funktion

1. Input mit Wert aus State befüllen

```
function GreetingDetail() {  
  const [ phrase, setPhrase ] = React.useState("Hello");  
  
  return <div>  
    <input  
      value={phrase}  
    />  
  </div>  
}
```

## BEISPIEL: EINGABEFELD

Komponente

Hello

]}-- input

"Hook"-Funktion

1. Input mit Wert aus State befüllen

2. Zustand neu setzen

```
function GreetingDetail() {  
  const [ phrase, setPhrase ] = React.useState("Hello");  
  
  return <div>  
    <input  
      value={phrase}  
      onChange={e => setPhrase(e.target.value)}  
    />  
  </div>  
}
```

# BEISPIEL: EINGABEFELD

Komponente

Hello

]-- input

"Hook"-Funktion

1. Input mit Wert aus State befüllen

2. Zustand neu setzen

```
function GreetingDetail() { ←----- Neu rendern
  const [ phrase, setPhrase ] = React.useState("Hello");
```

```
return <div>
  <input
    value={phrase}
    onChange={e => setPhrase(e.target.value)}
  />
</div>
}
```

Event

Zustand ändern

# REACT: UNI DIRECTIONAL DATAFLOW

```
class PasswordForm extends React.Component {  
  onPasswordChange(newPassword) { this.setState({password: newPassword}); }  
  ...  
  render() {  
    const password = this.state.password;  
    const checks = this.checkPassword(password);  
    const failedChecks = ...;  
    const isValidPassword = failedChecks === 0;  
  
    return <div>  
      <input type='password'  
             value={password}  
             onChange={event => this.onPasswordChange(event.target.value)} />  
  
      <CheckLabelList checks={checks}>  
        {failedChecks > 0 ?  
          <div className='Label'>{failedChecks} checks failed</div>  
          :  
          <div className='Label Label-success'>All checks passed!</div>  
        }  
      </CheckLabelList>  
      <Button label='Set Password' enabled={isValidPassword} />  
    </div>;  
  }  
}
```

The diagram illustrates the unidirectional data flow in React. It features three main components: **Zustand** (green), **Event** (orange), and **Rendern** (red). A blue dashed arrow points from **Zustand** to **Event**, representing the flow of state changes. Another blue dashed arrow points from **Event** to **Rendern**, representing the flow of events triggering re-renders. The code in the central column shows how an event (onChange) triggers a state update (onPasswordChange) in **Zustand**, which then drives the rendering logic in **Rendern**.

**Event**

**Zustand**

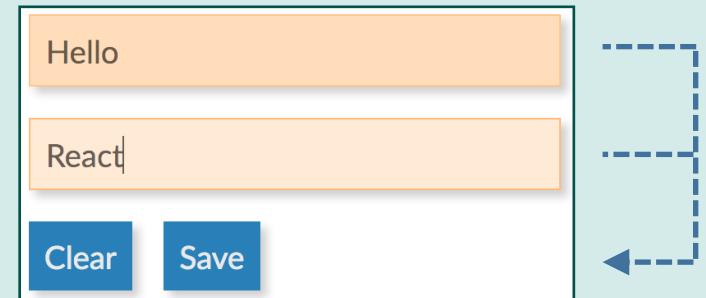
**Rendern**

**RESPOND TO EVENTS & RENDER UI**

# RENDERING VON KOMPONENTEN

## Gerendert wird immer die **ganze** Komponente

- Inklusive aller Unterkomponenten
- Bei Zustandsänderung
- **Verhindert Inkonsistenzen**
- "UI as a Function"



```
function GreetingDetail() {  
  const [ phrase, setPhrase ] = React.useState("Hello");  
  const [ name, setName ] = React.useState("React");  
  const saveDisabled = !(phrase && name);  
  
  return <div>  
    <input value={phrase} />  
    <input value={name} />  
    <button disabled={saveDisabled}>Save</button>  
  </div>  
}
```

# RENDERING VON KOMPONENTEN

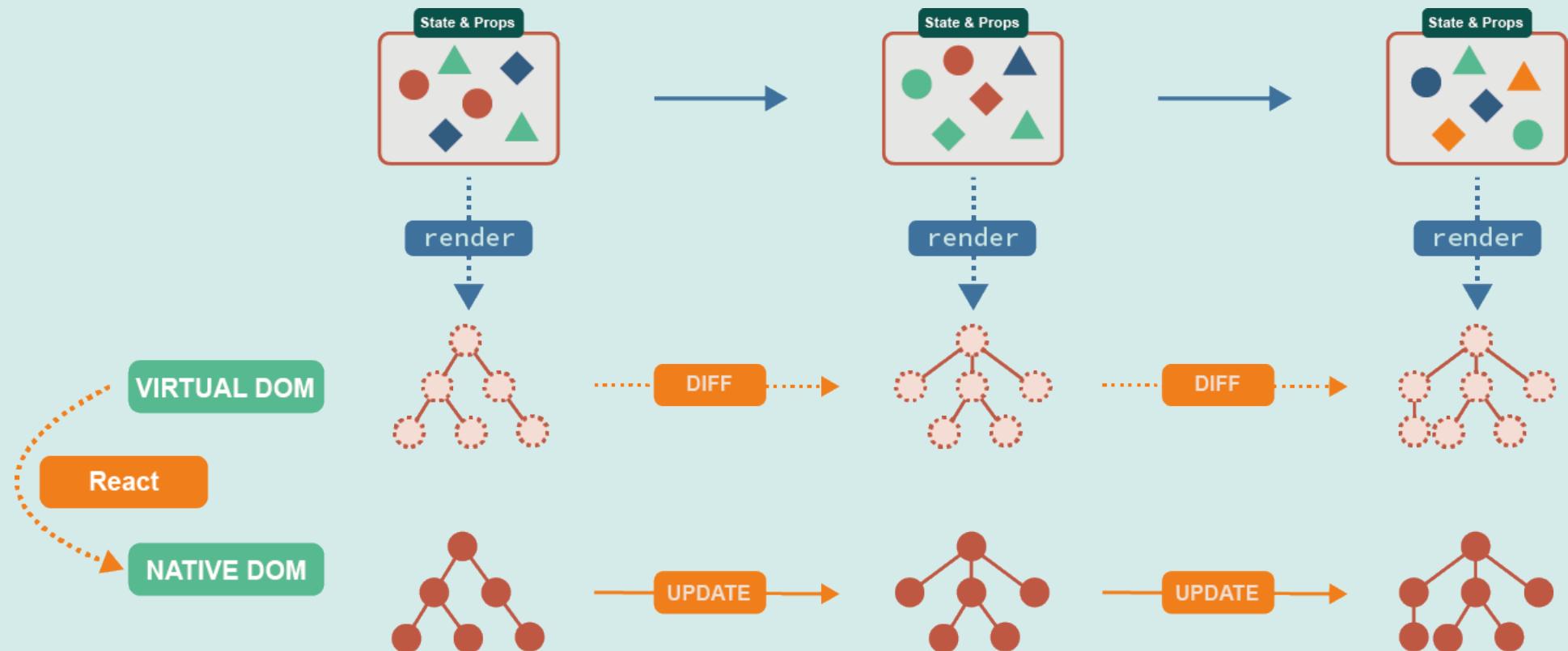
## Beispiel

👉 DevTools: GreetingDetail mit Input-Komponente

# RENDERING VON KOMPONENTEN

## Virtual DOM

👉 Rendering hat doppelte Bedeutung!



## BEISPIEL: DATEN VOM SERVER LADEN

- Gleiches Prinzip: geladene Daten werden in den State gesetzt
- Besonderheit: React Komponenten müssen Seiteneffekt-frei sein

```
function GreetingController() {  
  // 1. Model (State) für geladene Daten erzeugen  
  const [greetings, setGreetings] = React.useState(null);  
  
  // 2. Leere Liste (oder Platzhalter) rendern  
  return greetings <GreetingMaster greetings={greetings} />  
    ? : <LoadingIndicator />;  
}
```

## BEISPIEL: DATEN VOM SERVER LADEN

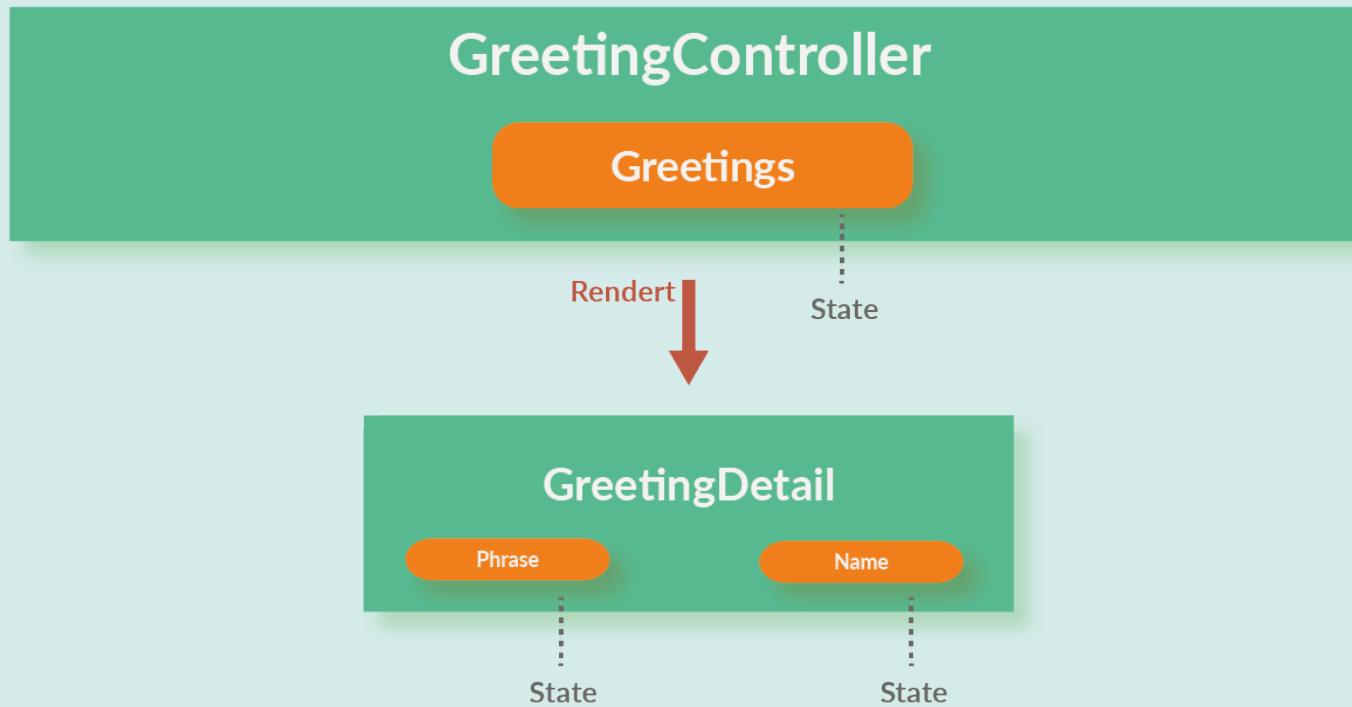
- Gleiches Prinzip: geladene Daten werden in den State gesetzt
- Besonderheit: React Komponenten müssen Seiteneffekt-frei sein

```
function GreetingController() {  
    // 1. Model (State) für geladene Daten erzeugen  
    const [greetings, setGreetings] = React.useState(null);  
  
    // 2. Seiteneffekt registrieren  
    React.useEffect( () => {  
        fetch("/api/greetings");  
        .then(response => response.json())  
        .then(greetingsAsJson => {  
            // 3. Daten sind geladen => State setzen => neu rendern  
            setGreetings(greetingsAsJson);  
        })  
    }, []);  
  
    // 4. Liste mit Grüßen (oder Platzhalter) rendern  
    return greetings <GreetingMaster greetings={greetings} />  
        ? : <LoadingIndicator />;  
}
```

**von der Komponente zur Anwendung**

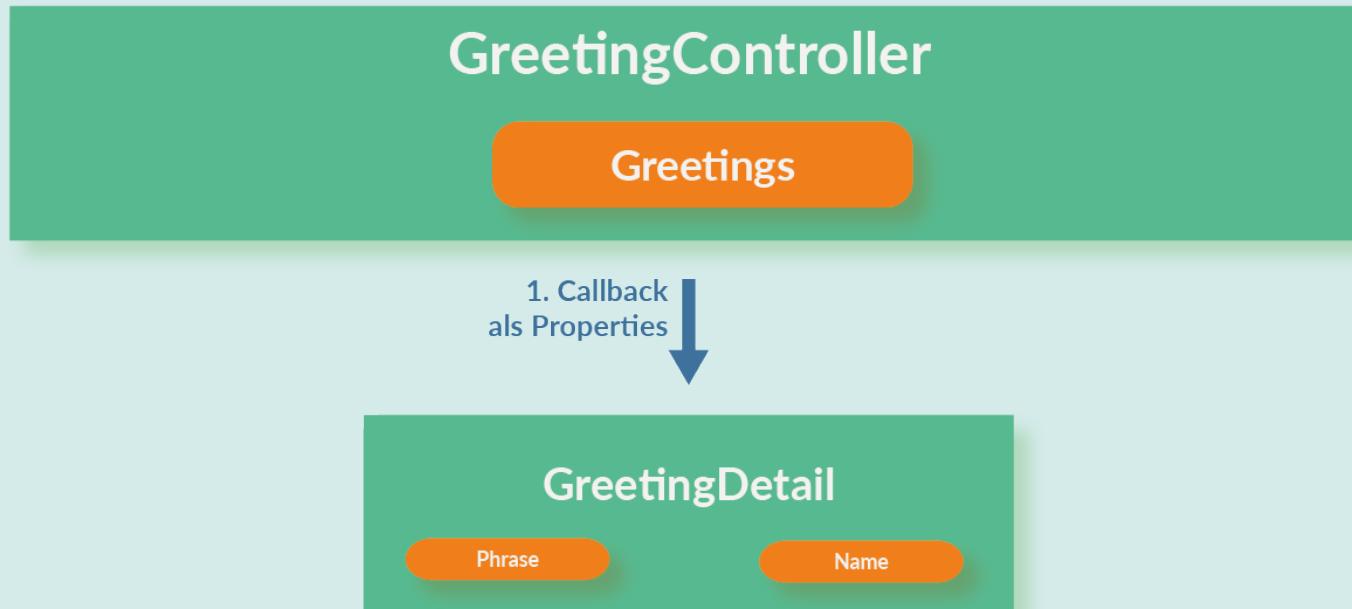
# KOMPONENTENHIERARCHIEN

- Komponenten werden in Hierarchien aggregiert



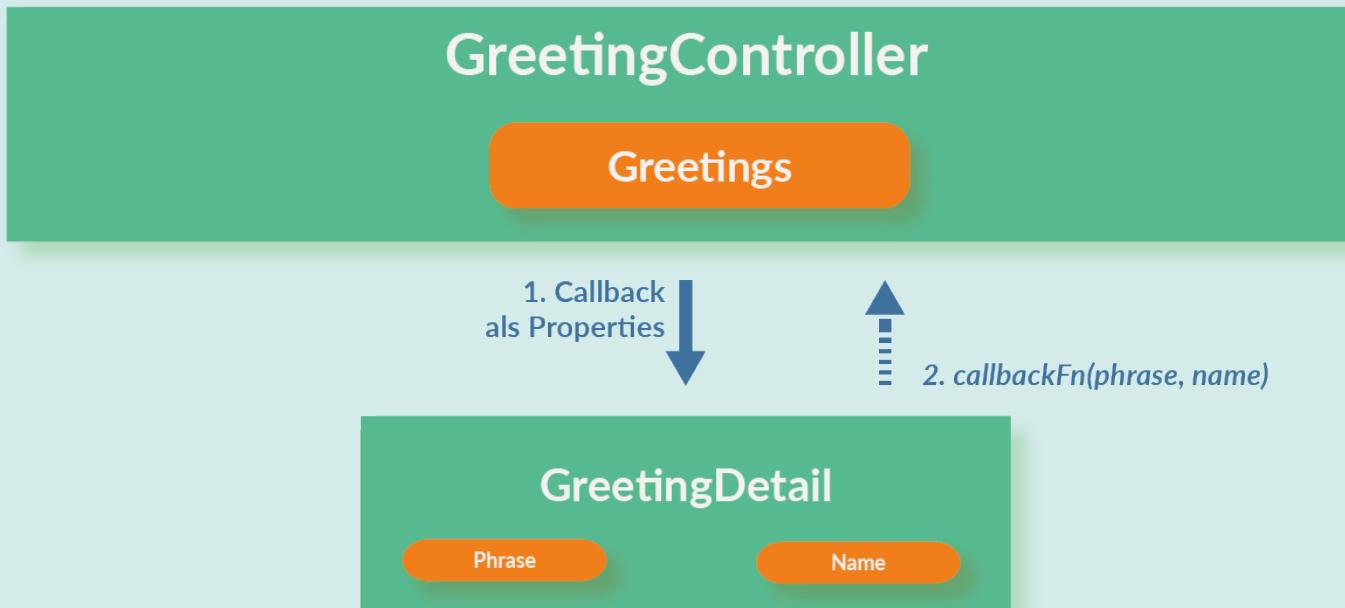
# KOMPONENTENHIERARCHIEN

- Kommunikation per Callback-Funktionen



# KOMPONENTENHIERARCHIEN

- **Kommunikation per Callback-Funktionen**
- Unterkomponente kann Daten per Callback nach oben kommunizieren



## BEISPIEL: KOMMUNIKATION PER CALLBACK-FUNKTION

```
function GreetingDetail({onSave}) {  
  const [ phrase, setPhrase ] = React.useState("Hello");  
  const [ name, setName ] = React.useState("React");  
  
  return <div>  
    <input value={phrase} ... />  
    <input value={name} ... />  
    <button onClick={() => onSave(phrase, name)}>Save</button>  
  </div>  
}
```

## BEISPIEL: KOMMUNIKATION PER CALLBACK-FUNKTION

```
function GreetingDetail({onSave}) {
  const [ phrase, setPhrase ] = React.useState("Hello");
  const [ name, setName ] = React.useState("React");

  return <div>
    <input value={phrase} ... />
    <input value={name} ... />
    <button onClick={() => onSave(phrase, name)}>Save</button>
  </div>
}

function GreetingController() {
  function onNewGreeting(phrase, name) {
    // ...
  }

  return ... <GreetingDetail onSave={onNewGreeting} /> ...
}
```

## BEISPIEL: ROUTING UND DEEP LINKS

### React hat keinen eigenen Router

- De-facto-Standard: React Router

```
import GreetingMaster from "./GreetingMaster";
import GreetingDisplay from "./GreetingDisplay";
import NotFound from "./NotFound";
import {Route, Switch} from "react-router-dom";

function App() {
  return (
    <div>
      <h1>Greetings</h1>
      <Switch>
        <Route path="/greet/:greetingId"><GreetingDisplayPage/></Route>
        <Route exact path="/">          <GreetingMaster />      </Route>
        <Route>                      <NotFound />          </Route>
      </Switch>
    </div>
  );
}
```

## BEISPIEL: TESTEN

### Testen einer Komponente

Testen ohne Browser möglich, headless im CI-Build

```
import React from "react";
import { render, fireEvent } from "@testing-library/react";
import GreetingDetails from "./GreetingDetails";

test("it should behave fine", () => {

  // Mock für Event-Handler
  const onSaveHandler = jest.fn();

  // Komponente Rendern
  const { getByText, container } = render(
    <GreetingDetails initialPhrase="Hello" initialGreeting="React"
      onSave={onSaveHandler} />
  );
  
  // Ereignis simulieren
  fireEvent.click(getByText("Save"));

  // Ergebnis überprüfen
  expect(onSaveHandler).toHaveBeenCalledWith("Hello", "React");
});
```

# **GLOBALE DATEN**

**ZUSTANDSMANAGEMENT**

# GLOBALE DATEN

- Beispiel: Globale Daten in der Anwendung

Screenshot of a web application titled "Greeting App" running on localhost. The application displays a table of greetings and a pie chart.

The table shows three entries for Susi:

Name	Greeting
Susi	Hello!
Susi	How are you?
Susi	Have a nice day

(Shown are greetings for Susi. Reset Filter)

Add

2-hierarchy

localhost 80% Search

Showing 3 of 10 Greetings

Klaus Susi Max Felipe Alex Paul

Paul  
Klaus  
Susi  
Felipe  
Max  
Alex

A pie chart illustrating the distribution of greetings. The slices represent different names: Klaus (blue), Susi (light blue), Max (orange), Felipe (light orange), Alex (green), and Paul (light green). The slice for Susi is highlighted with a purple oval.

# GLOBALE DATEN

- Beispiel: Globale Aktionen in der Anwendung

Screenshot of a web application titled "Greeting App" showing a list of greetings and a pie chart.

The application interface includes:

- Header bar with tabs: "2-hierarchy" (active), "localhost" (address bar), "80%", "Search", and various browser icons.
- Title: "Greeting App".
- Text: "Showing 3 of 10 Greetings".
- Table:

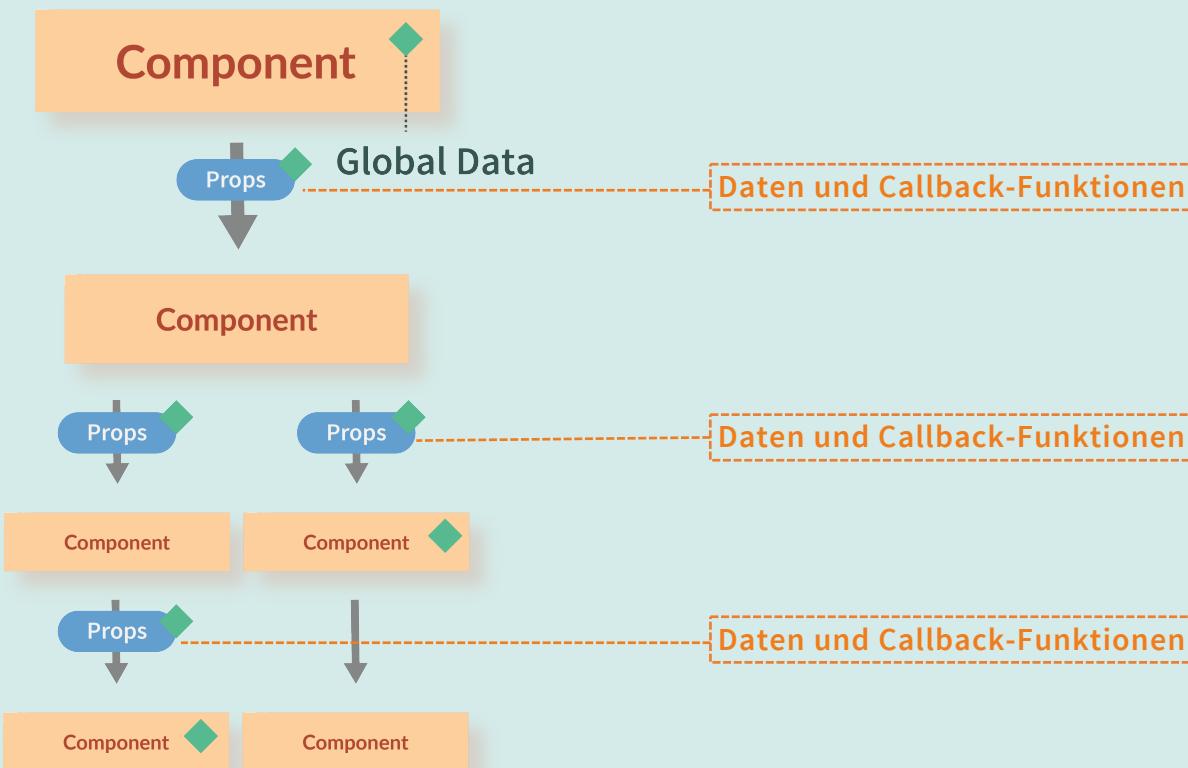
Name	Greeting
Susi	Hello!
Susi	How are you?
Susi	Have a nice day
- Text: "(Shown are greetings for Susi. Reset Filter)".
- Button: "Add".
- Pie chart legend: Klaus (blue), Susi (light blue), Max (orange), Felipe (orange), Alex (green), Paul (light green).
- Pie chart data (approximate proportions): Susi (~35%), Max (~15%), Felipe (~20%), Klaus (~10%), Alex (~10%), Paul (~10%).

Annotations:

- A purple oval highlights the "Susi" entry in the table.
- A purple oval highlights the "Reset Filter" link in the text below the table.
- A purple oval highlights the "Susi" slice in the pie chart.

# OPTIONEN

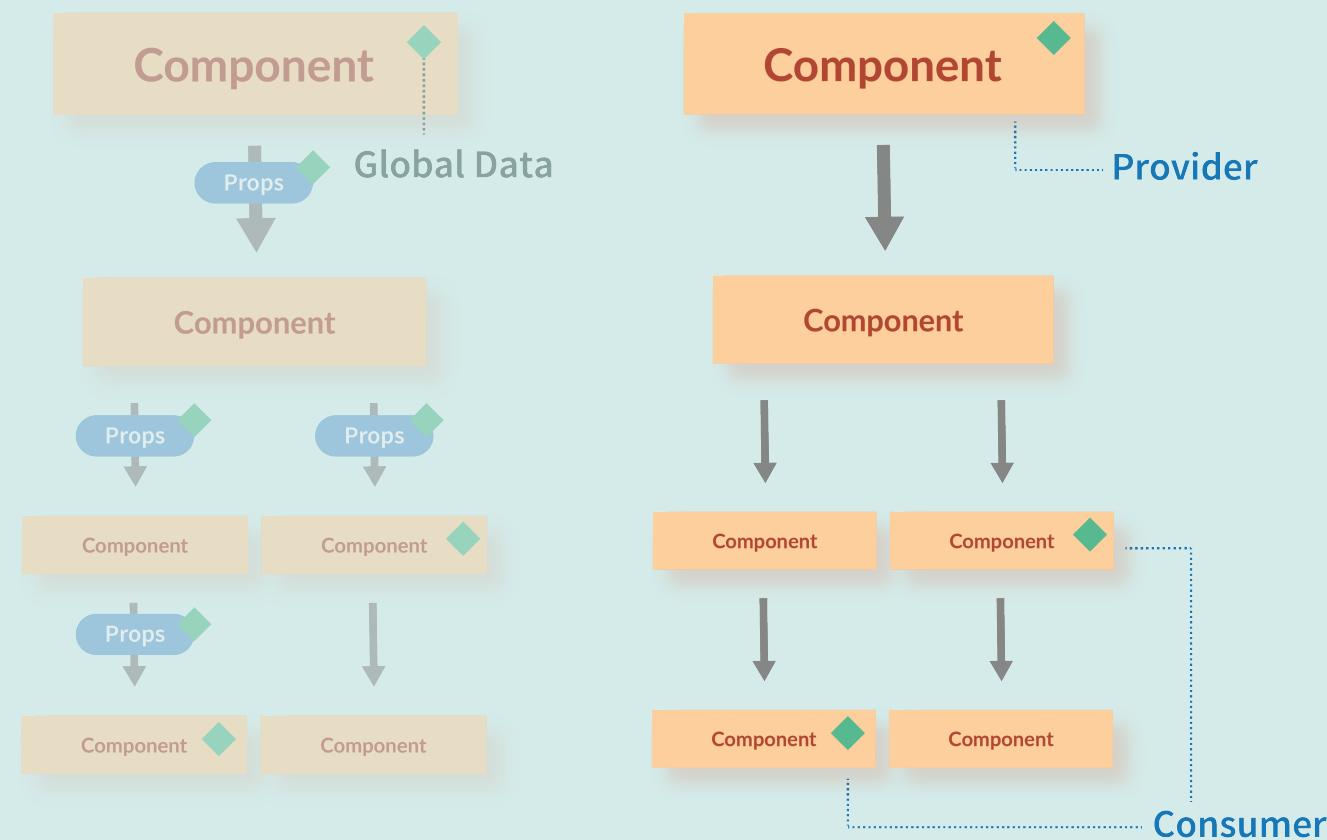
## Globale Daten: Mit Properties (Durchreichen)



# OPTIONEN

## Globale Daten: Mittels Context API

- Provider bietet Daten an
- Consumer kann auf Daten zugreifen



## BEISPIEL: REACT CONTEXT

**Provider stellt globale Daten und Aktionen zur Verfügung**

```
const GreetingContext = React.createContext();

function GreetingProvider({children}) {
  const [greetings, setGreetings] = React.useState([]);
  const [filter, setFilter] = React.useState("");

  return <GreetingContext.Provider
    value={{greetings, setGreetings, filter, setFilter}}>
    {children}
  </GreetingContext.Provider>;
}
```

## BEISPIEL: REACT CONTEXT

### Einbinden des Providers in die Komponenten-Hierarchie

```
import GreetingProvider from "./GreetingProvider";

function App() {
  return <GreetingProvider>
    <div className="Main">
      <div className="Title">
        <Counter filtered={3} total={11} /> -----
      </div>
      <div className="Left">
        <Greeting /> -----
      </div>
      <div className="Right">
        <Chart /> -----
      </div>
    </div>
  </GreetingProvider>
}
```

Zugriff auf Context möglich

# BEISPIEL: REACT CONTEXT

## Verwenden des Contexts

```
import GreetingContext from "./GreetingProvider";

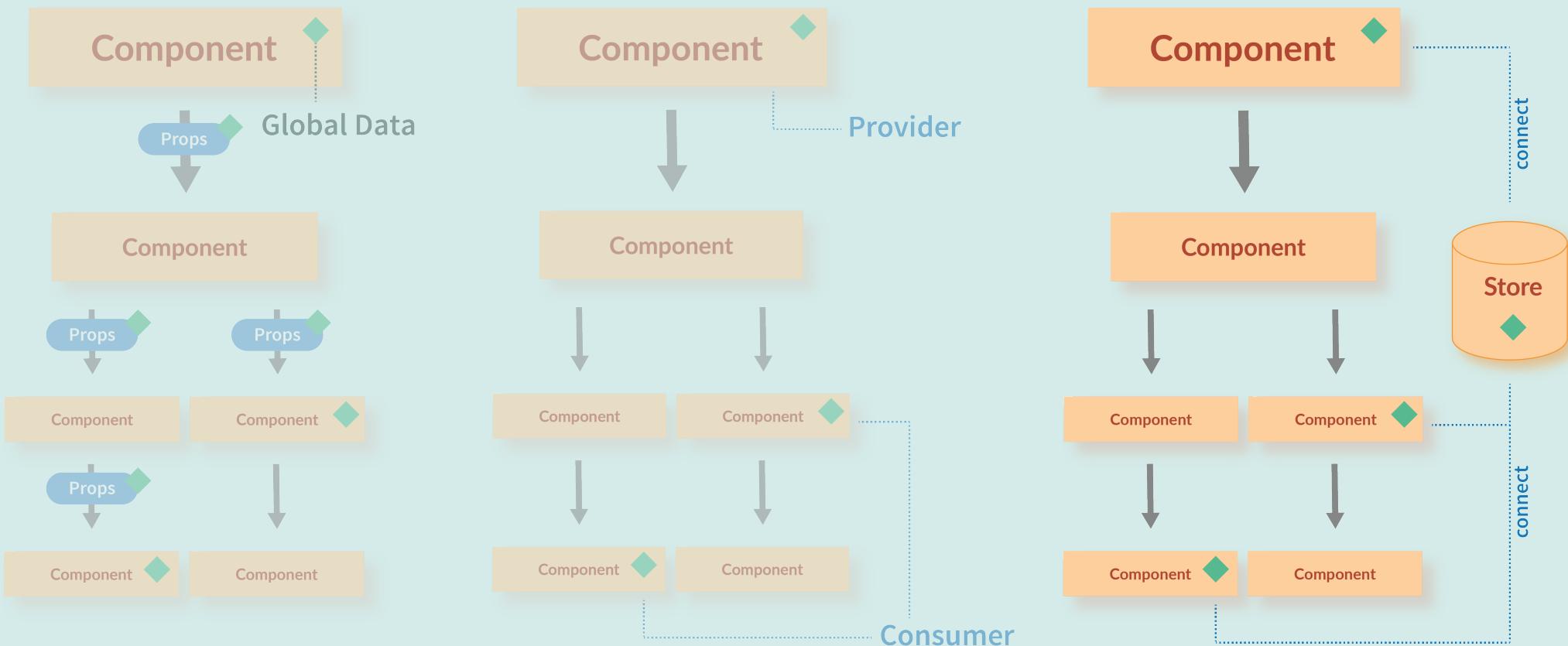
function GreetingMaster() {
  const {greetings, filter, setFilter} = React.useContext(GreetingContext);
  const filteredGreetings = filterGreetings(greetings, filter);

  return <table>
    {filteredGreetings.map(greeting =>
      <tr onClick={() => setFilter(greeting.name)}>
        <td>{greeting.name}</td><td>{greeting.phrase}</td>
      </tr>
    )}
  </table>
}
```

# OPTIONEN

## Globale Daten: Mit externem State Management

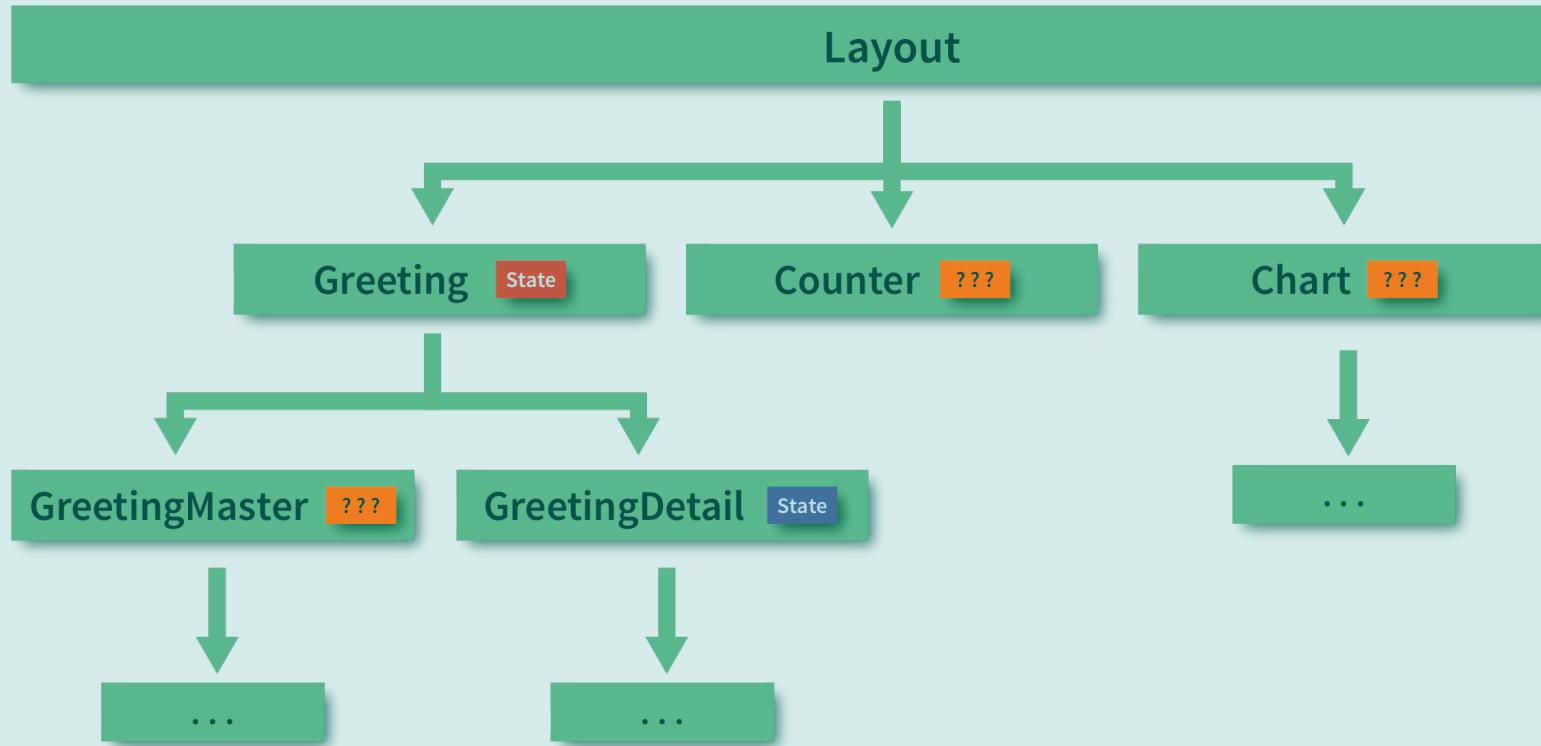
- MobX oder Redux



# **REDUX**

**EXTERNES STATEMANAGEMENT**

# KOMPONENTENHIERARCHIEN



## Probleme:

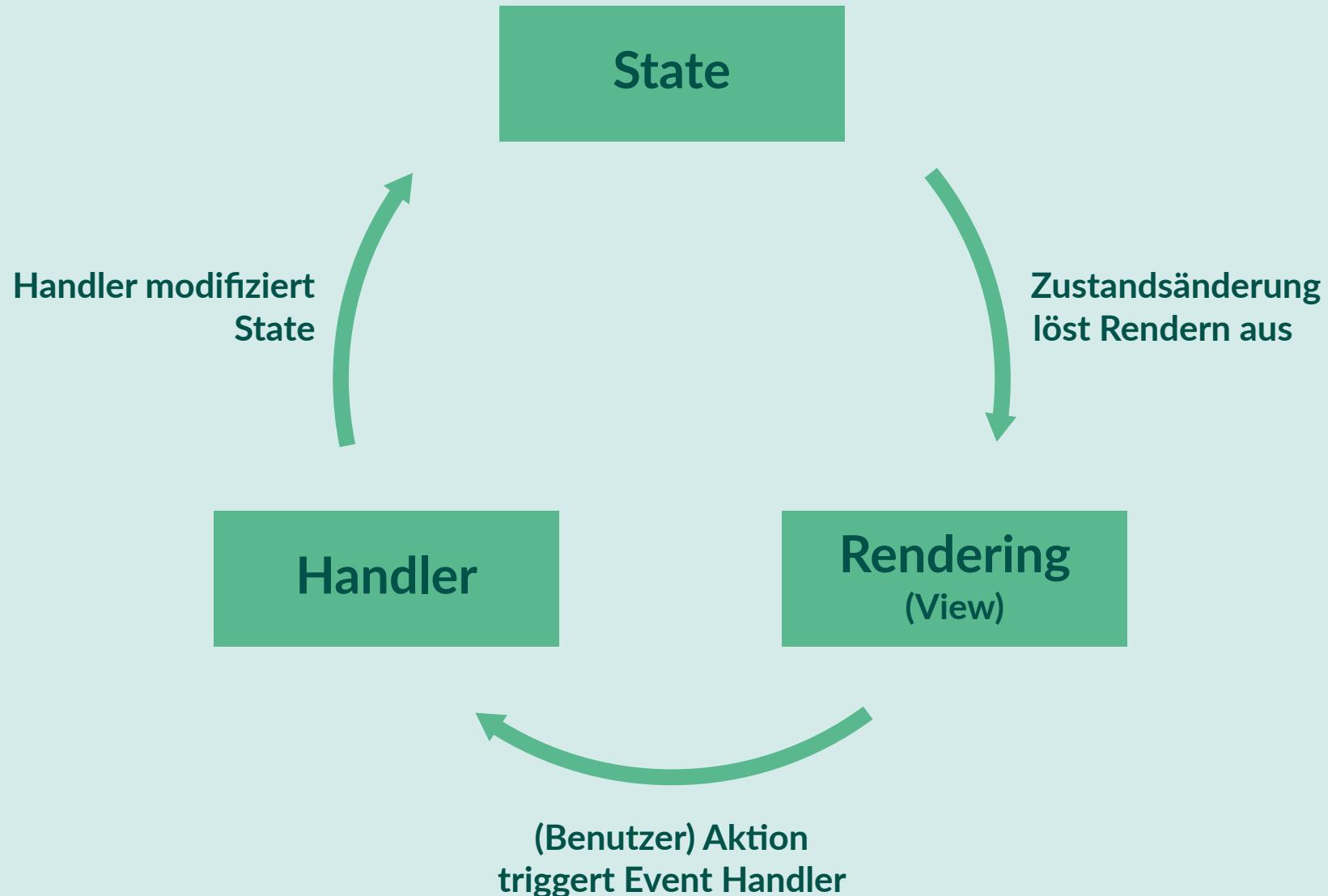
- Wohin mit gemeinsamen Zustand? (Greetings in 3 Komponenten!)
- Verteilter Zustand vs "Gott-Komponenten"
- Kopplung UI und Fach-Logik

# EXTERNES STATE MANAGEMENT

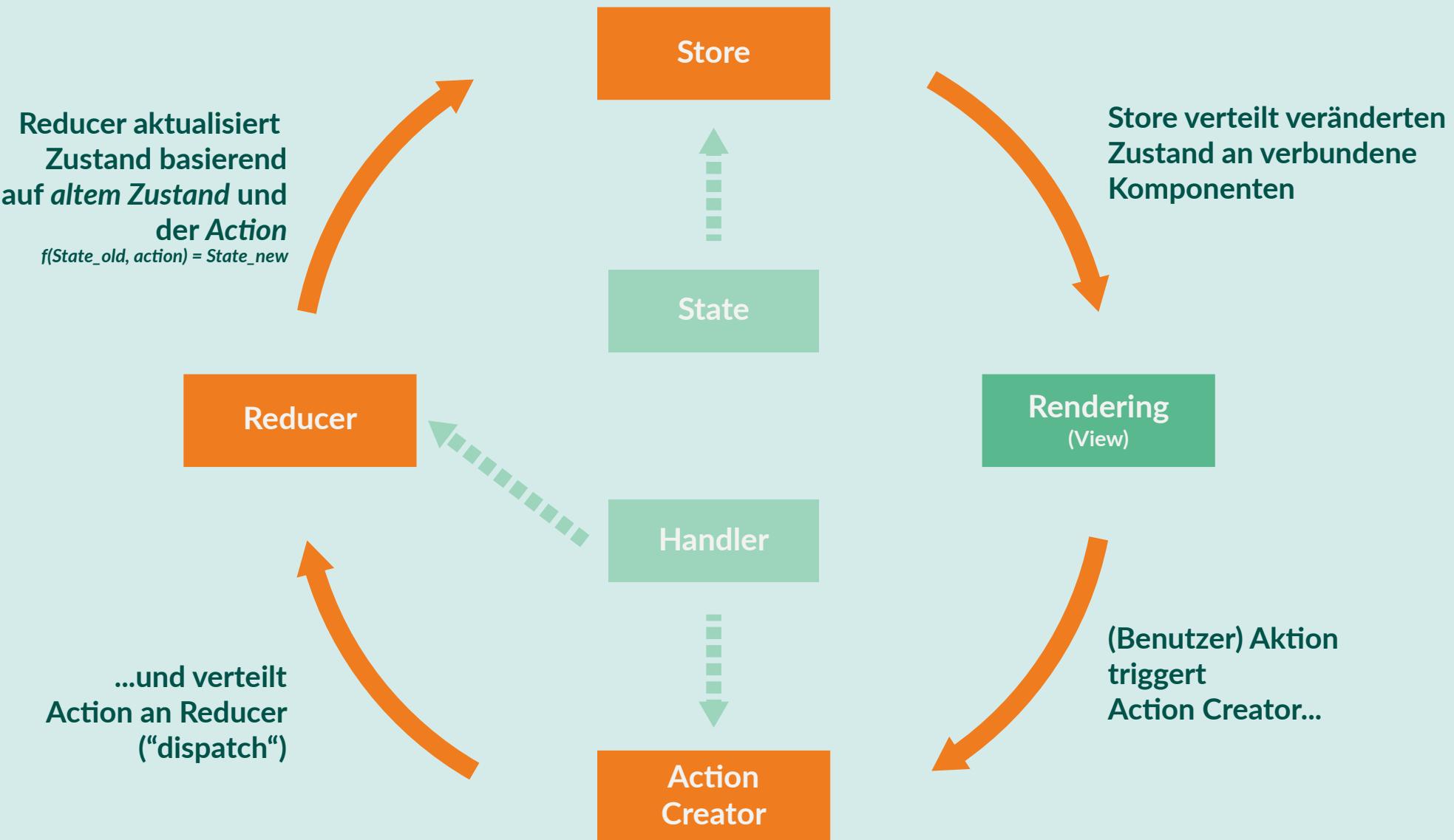
## Beispiel: Redux

- Extern: Zustand wandert aus den UI Komponenten
- Zentral: Zustand wandert in einen Store
- Architektur Pattern und Implementierung
- React-unabhängig
  - Bindings u.a. auch für Angular und Vue

# REACT RENDER ZYKLUS



# REDUX-BASIERTE REACT-ANWENDUNG



## BEISPIEL: REDUX

### Zugriff auf globalen Zustand mit useSelector-Hook

```
import {useSelector} from "react-redux";

function GreetingMaster() {
  const { greetings, filter } = useSelector(state => ({
    greetings: state.greetings,
    filter: state.filter
 )));
  return ... // Tabelle mit gefilterten Greetings
}
```

## BEISPIEL: REDUX

### Auslösen von Actions mit useDispatch

```
import {useSelector, useDispatch} from "react-redux";

function GreetingMaster() {
  const {greetings, filter} = useSelector(...);
  const dispatch = useDispatch();

  function rowClick(greeting) {
    dispatch({
      type: "FILTER_SELECTED", payload: { filter: greeting.filter }
    })
  }

  return ...
  <tr onClick={() => rowClick(greeting)}>...Greeting...</tr>
}
```

# BEISPIEL: REDUX

## Reducer-Funktion

```
function filterReducer(filter = "", action) {  
  switch (action.type) {  
    case "FILTER_SELECTED":  
      return action.payload;  
    case "FILTER_CLEARED":  
      return "";  
    default:  
      return filter;  
  }  
}
```

# TYPESCRIPT

TYPSICHERES REACT

## TypeScript: Obermenge von JavaScript

- Jeder gültige JavaScript Code ist gültiger TypeScript Code (theoretisch...)
- Ergänzt JS um Typ-System, Sichtbarkeiten, Enums und Dekoratoren
- Compiler erzeugt aus TypeScript-Code JavaScript (ES3, ES5, ES6)-Code
- Entwickelt von Microsoft
  - <http://www.typescriptlang.org/>
- Sehr guter IDE Support
  - IntelliJ IDEA, Visual Studio Code

## Typ-Angaben für typsichere Artefakte

- State (kann abgeleitet werden)
- Komponenten Properties
- Redux State und Actions

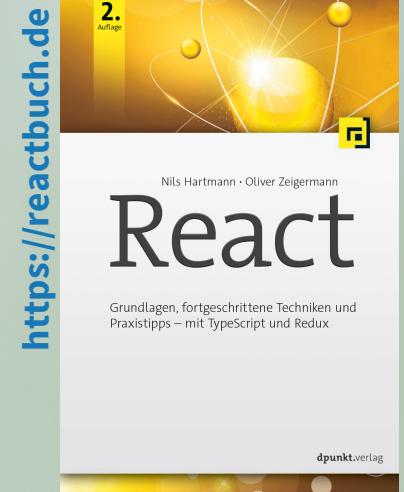
# TYPESCRIPT DEMO

## Typen für React Komponenten Fehlermeldung und Code Completion in der IDE

The screenshot shows the VS Code interface with the following details:

- File:** GreetingDetail.tsx
- Content:**

```
8
9  interface GreetingDetailState {
10    name: string;
11    greeting: string;
12  }
13
14  export default class GreetingDetail extends React.Component<
15    GreetingDetailProps,
16    GreetingDetailState
17  > {
18    input?: HTMLInputElement | [ts] Type 'Readonly<GreetingDetailState>' has no property
19    'nothere' and no string index signature.
20    render() {
21      const { name, greeting, nothere } = this.state;
22      const saveDisabled = !(name && greeting);
23
24      return (
25        <div>
26          <input
```
- Code Completion Tooltip:** A tooltip appears over the `nothere` variable in line 21, stating: "[ts] Type 'Readonly<GreetingDetailState>' has no property 'nothere' and no string index signature." It also shows a suggestion: "const nothere: any".
- Problems Bar:** Shows one error: "[ts] Type 'Readonly<GreetingDetailState>' has no property 'nothere' and no string index signature. (21, 29)".
- Status Bar:** Shows the current line and column: Ln 21, Col 36.



# vielen Dank!

Slides: <https://nils.buzz/conceptpeople-react>

Source Code: <https://github.com/nilshartmann/react-training>

Fragen & Kontakt: [nils@nilshartmann.net](mailto:nils@nilshartmann.net)