

NILS HARTMANN

**Context API, Suspense, Time Slicing & mehr**

# React 2018

Slides: <https://bit.ly/wjax2018-react>

# **NILS HARTMANN**

## **Programmierer aus Hamburg**

**JavaScript, TypeScript, React  
Java**

**Trainings, Workshops**  
 [nils@nilshartmann.net](mailto:nils@nilshartmann.net)

**@NILSHARTMANN**

# NILS HARTMANN



gaearon commented 5 days ago

Member



...

[REDACTED]

[REDACTED]

@nilshartmann

[REDACTED]

I do think you're a bit confused.

[REDACTED]

[REDACTED]



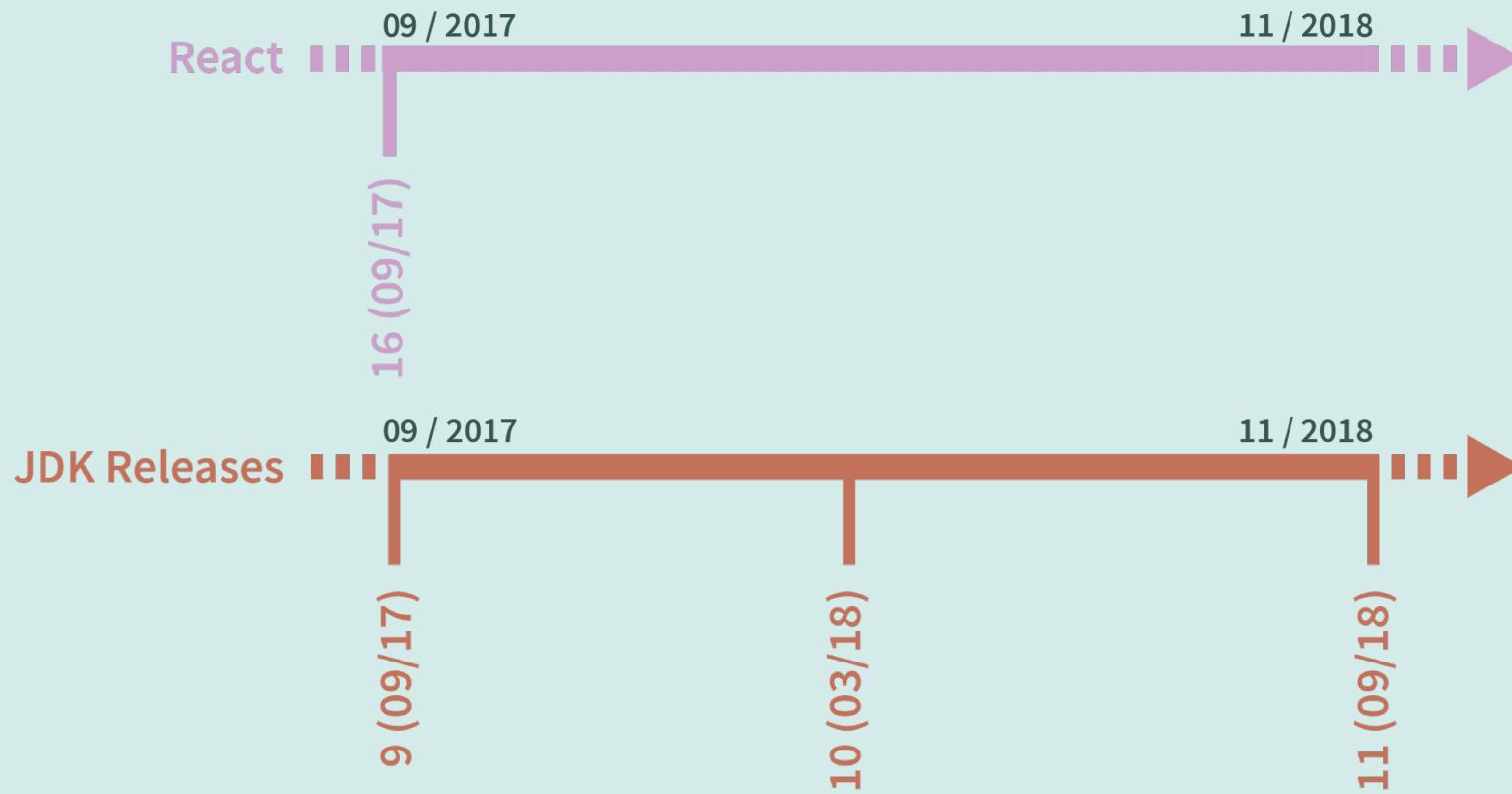
2

"A BIT CONFUSED"

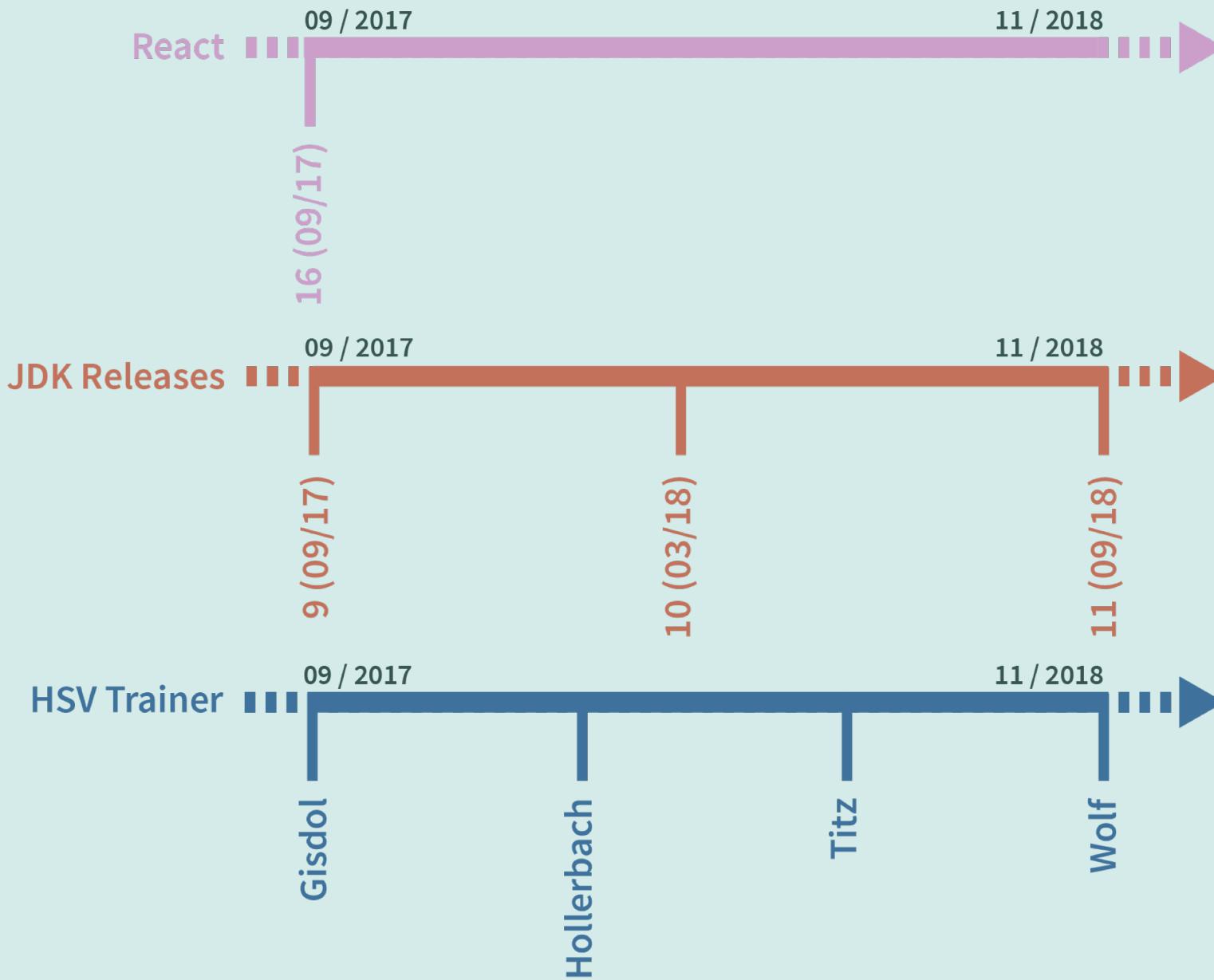
## ZUM VERGLEICH...



## ZUM VERGLEICH...



## ZUM VERGLEICH...



**React 16**

# **Fiber**

**ALLES NEU?**

The image displays two side-by-side screenshots of a web application titled "React Chat Example".

**Left Screenshot (React 16.6.0):**

- Header:** "React Chat Example" and "16.6.0".
- Top Bar:** Buttons for "In the Office...", "Philosophy", and "Coffee".
- Message List:** A series of messages from users like Harry, Peter, Maja, and Sue.
- Anonymous User Alert:** A message box for "Anonymous-620" stating they are not logged in and need to log in to send messages.
- Login Button:** A red "Login" button.
- Footer:** "Please login to post messages" and a "Login" button.
- Bottom Bar:** "Exit" button.
- Page URL:** <https://github.com/nilshartmann/react-chat-example>

**Right Screenshot (React 16.7.0-alpha.0):**

- Header:** "React Chat Example" and "React 16.7.0-alpha.0".
- Top Bar:** Buttons for "In the Office...", "Philosophy", and "Coffee".
- Message List:** A series of messages from users like Harry, Peter, Maja, and Sue.
- Anonymous User Alert:** A message box for "Anonymous-620" stating they are not logged in and need to log in to send messages.
- User Profile:** A message box for "Klaus" stating "You're logged in as Klaus".
- Input Field:** "Add Message" input field and a "Send" button.
- Side Panel:** Buttons for "Dashboard (Effects)", "Dashboard (Suspense)", and "Exit".
- Bottom Bar:** "Exit" button.
- Page URL:** <https://github.com/nilshartmann/react-chat-example>

<https://github.com/nilshartmann/react-chat-example>

EIN BEISPIEL...

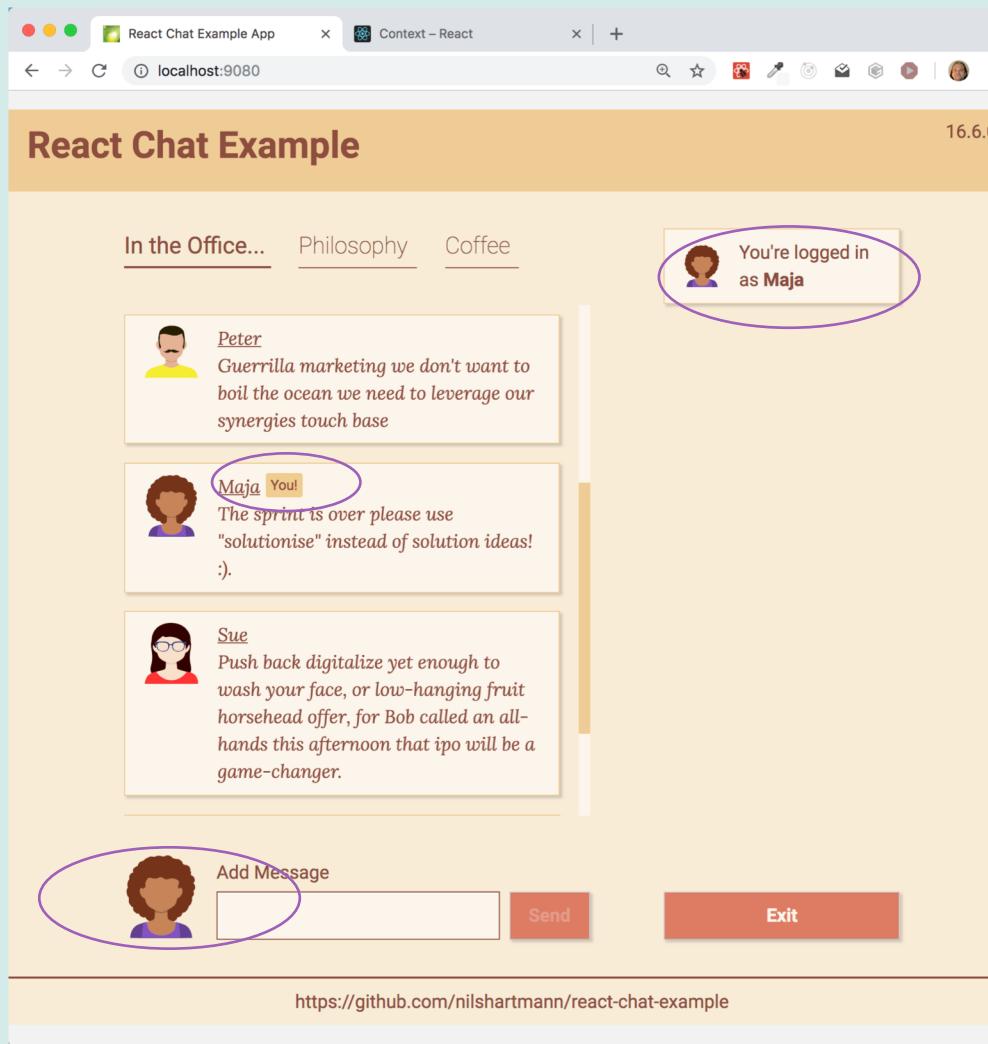
16.3

# Context

GLOBALE DATEN IN DER ANWENDUNG

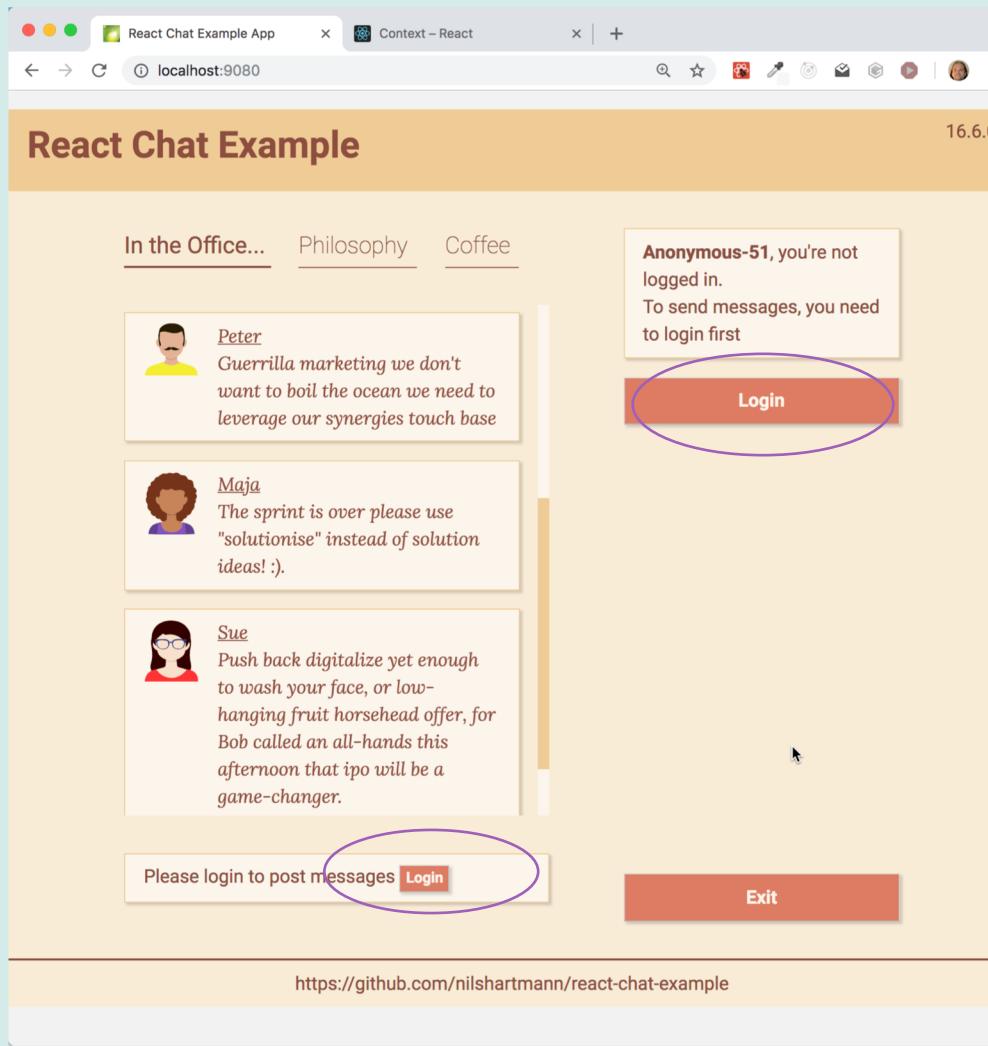
# REACT CONTEXT

- **Problem: Globale Daten in der Anwendung**



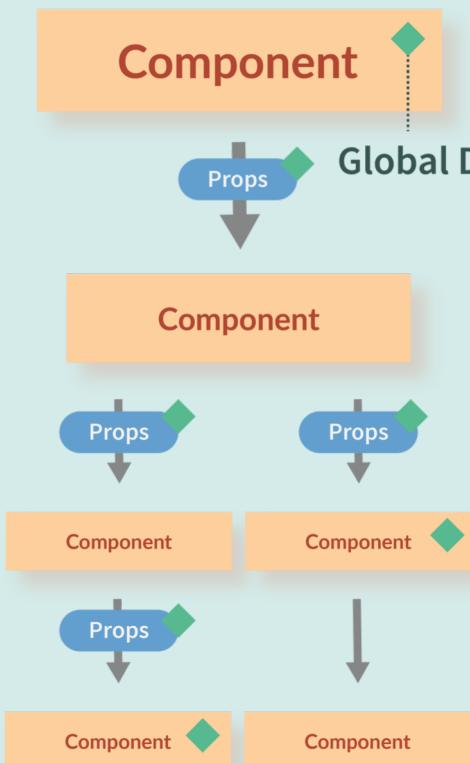
# REACT CONTEXT

- **Problem: Globale Aktionen in der Anwendung**



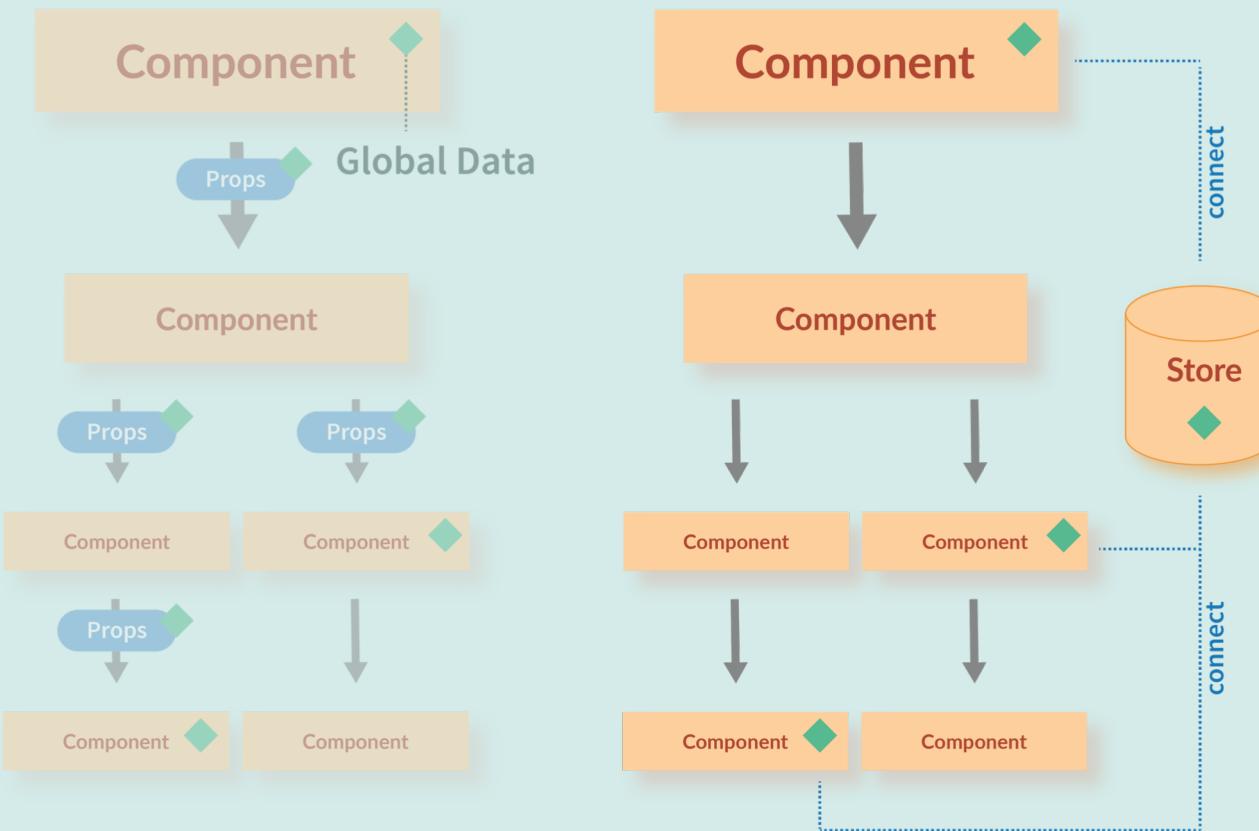
# REACT CONTEXT

## Globale Daten: Mit Properties (Durchreichen)



# REACT CONTEXT

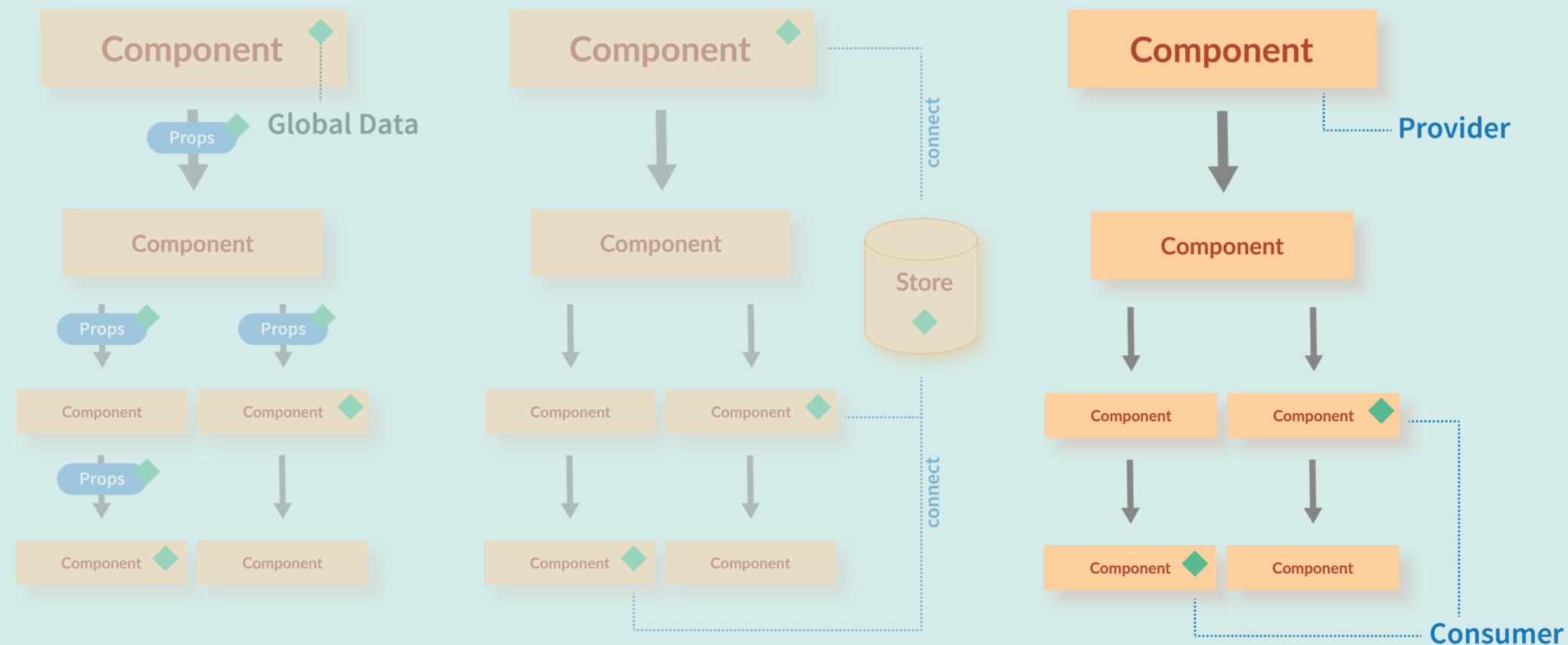
**Globale Daten:** Mit externem State Management (Redux/Mobx)



# REACT CONTEXT

## Globale Daten: Mittels Context API

- Provider bietet Daten an
- Consumer kann auf Daten zugreifen



# REACT CONTEXT VERWENDEN

## Context Erzeugen

```
// Context erzeugen:  
  
const UserContext = React.createContext({ .... });  
  
// Provider-Komponente:  
UserContext.Provider;  
  
// Consumer-Komponente:  
UserContext.Consumer;
```

## React Context: Verwenden des Providers

- Daten werden mit "value"-Property angegeben
- Daten können dann in Unterkomponenten konsumiert werden

```
class UserDataManager extends React.Component {  
  render() {  
    const value = {  
  
    }  
  
    return <UserContext.Provider value={value}>  
      {this.props.children}  
    </UserContext.Provider>  
  }  
}
```

## React Context: value-Objekt

- Stellt die eigentlichen Daten zur Verfügung

```
class UserDataManager extends React.Component {  
  render() {  
    const value = {  
      user: this.state.user,  
  
    }  
  
    return <UserContext.Provider value={value}>  
      {this.props.children}  
    </UserContext.Provider>  
  }  
}
```

## React Context: value-Objekt

- Stellt die eigentlichen Daten zur Verfügung
- Callback führt zu State-Änderung, Provider wird neu gerendert, Konsumenten erhalten neue Werte (analog zu Properties)

```
class UserDataManager extends React.Component {  
  render() {  
    const value = {  
      user: this.state.user,  
      login = userId => { const newUser = loginViaApi(userId);  
                            this.setState({user: newUser}) }  
    }  
  
    return <UserContext.Provider value={value}>  
      {this.props.children}  
    </UserContext.Provider>  
  }  
}
```

# REACT CONTEXT API

## Consumer: Daten aus Context verwenden

```
import UserContext from "...";  
  
class UserProfile extends React.Component {  
  render() {  
    return <UserContext.Consumer>  
      { values => {  
        // ... work with data from context ...  
      } }  
    </UserContext.Consumer>  
  }  
}
```

**Function as Children** ----- 

## HINTERGRUND: RENDER PROPS

### Render Properties: Alternative zu HOCs

- Über ein Property wird einer Komponente ein Callback übergeben
- Dieses Callback liefert aber keine Daten, sondern rendert eine Komponente
- Populär z.B. in React Router und Apollo GraphQL

## HINTERGRUND: RENDER PROPS

### Render Properties: Alternative zu HOCs

```
class ChatMessageView extends React.Component {  
  render() {  
    return <>  
      <h1>Latest Chat Message</h1>  
      <ChatMessageLoader  
        onLoading={ () => <p>Please wait, Data is loading</p> }  
        onMsg={ (msg) => <p>{msg.title} {msg.body}</p> }  
      />  
    </>;  
  }  
}
```

## HINTERGRUND: RENDER PROPS

### Spezialfall: Function as Children

- Callback wird als Children übergeben

```
class ChatMessageView extends React.Component {  
  render() {  
    return <>  
      <h1>Latest Chat Message</h1>  
      <ChatMessageLoader>  
        { (load, msg) => {  
          if (load) {<p>Please wait, Data is loading</p>}  
          return <p>{msg.title} {msg.body}</p>  
        } }  
      />  
    </>;  
  }  
}
```

Function as Children ---

# REACT CONTEXT API

## Consumer: Daten aus Context verwenden

```
import ChatContext from "...";  
  
class UserProfile extends React.Component {  
  render() {  
    return <ChatContext.Consumer>  
      { values => {  
        } }  
    </ChatContext.Consumer>  
  }  
}
```

# REACT CONTEXT API

## Consumer: Daten aus Context verwenden

```
import ChatContext from "...";  
  
class UserProfile extends React.Component {  
  render() {  
    return <ChatContext.Consumer>  
      { values => {  
        if (values.user) { return <p>Hello, {values.user}</p> }  
        return <button onClick={()=>values.login()}>Login</button>  
      } }  
    </ChatContext.Consumer>  
  }  
}
```

Auf Daten zugreifen  
"Action" auslösen

# REACT CONTEXT API

## Consumer: Mehrere Kontexte verwenden

- Kontexte können z.B. fachlich aufgeteilt werden

```
import UserContext from "...";
import ChatContext from "...";
```

```
class Chatroom extends React.Component {
  render() {
    return <UserContext.Consumer>
      { user => {
        <ChatContext.Consumer>
          { chat => {
            return <p>Hello {user.name}, you're in {chat.room}</p>
          } }
        </ChatContext.Consumer>
      } }
    </UserContext.Consumer>
  }
}
```

## REACT CONTEXT API

- **Consumer:** contextType [16.6]
- Alternativer Zugriff (nur in Klassen, nur bei einem Kontexttype)

```
import ChatContext from "...";

class UserProfile extends React.Component {
  static contextType = ChatContext;

  render() {
    if (this.context.user) { return <p>Hello, {this.context.user}</p> }

    return <button onClick={() => this.context.login()}>Login</button>
  }
}
```

16.6

# Suspense

RENDERN UNTERBRECHEN

## SUSPENSE

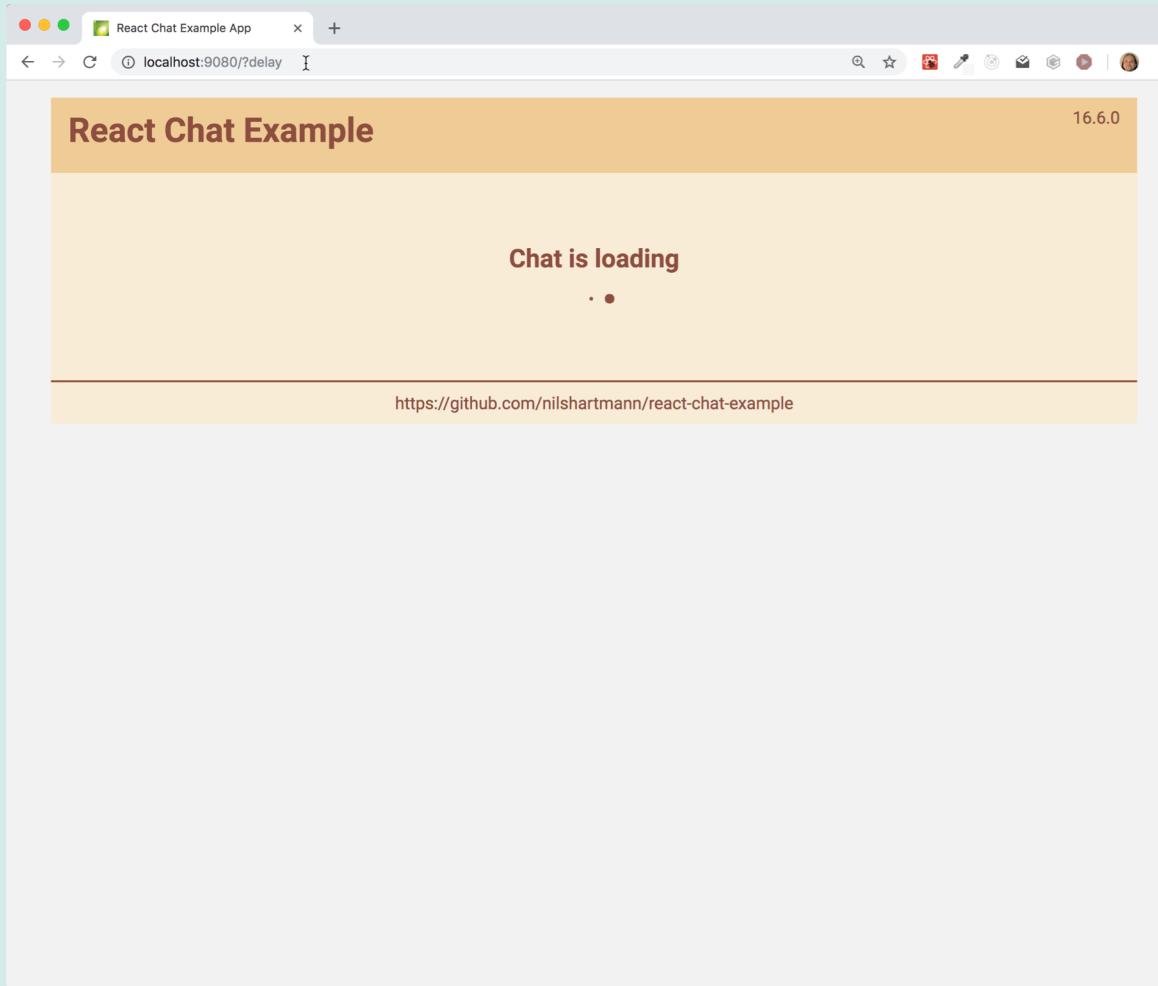
**Suspense:** Rendern unterbrechen, während (asynchron) Daten geladen werden [16.6]

- Funktioniert aktuell für Code Splitting

# DEMO: LAZY UND SUSPENSE

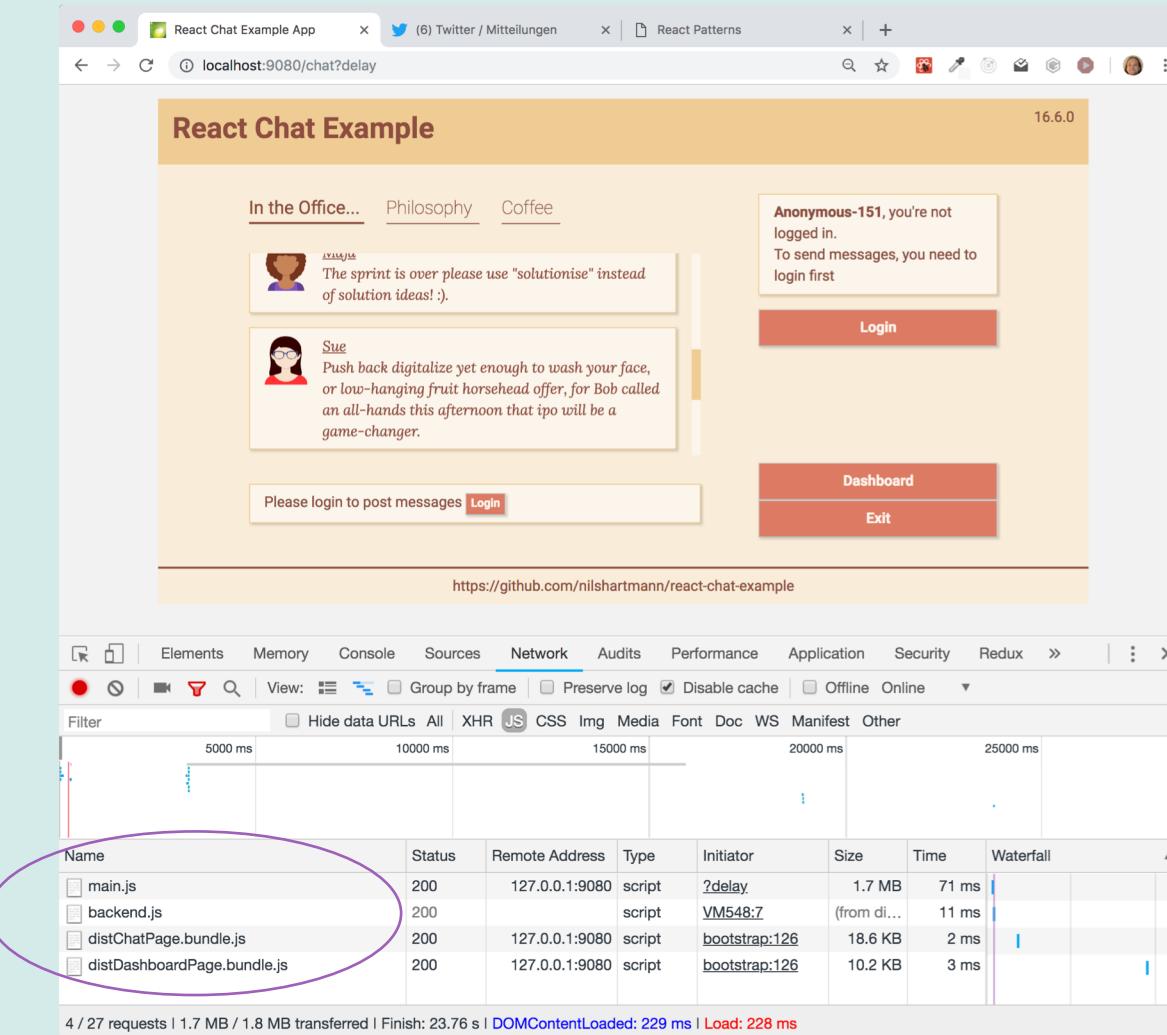
- **Demo: Fallback Komponente**

<http://localhost:9080/?delay>



# DEMO: LAZY UND SUSPENSE

- **Netzwerk Requests**  
<http://localhost:9080/?delay>



React Chat Example 16.6.0

In the Office... Philosophy Coffee

AYAMAJM The sprint is over please use "solutionise" instead of solution ideas! .

Sue Push back digitalize yet enough to wash your face, or low-hanging fruit horsehead offer, for Bob called an all-hands this afternoon that ipo will be a game-changer.

Anonymous-151, you're not logged in. To send messages, you need to login first

Login

Please login to post messages [Login](#)

Dashboard

Exit

<https://github.com/nilshartmann/react-chat-example>

Elements Memory Console Sources Network Audits Performance Application Security Redux ▾

View: Group by frame Preserve log Disable cache Offline Online

Filter Hide data URLs All XHR JS CSS Img Media Font Doc WS Manifest Other

Name	Status	Remote Address	Type	Initiator	Size	Time	Waterfall
main.js	200	127.0.0.1:9080	script	?delay	1.7 MB	71 ms	
backend.js	200	127.0.0.1:9080	script	VM548:7	(from di...	11 ms	
distChatPage.bundle.js	200	127.0.0.1:9080	script	bootstrap:126	18.6 KB	2 ms	
distDashboardPage.bundle.js	200	127.0.0.1:9080	script	bootstrap:126	10.2 KB	3 ms	

4 / 27 requests | 1.7 MB / 1.8 MB transferred | Finish: 23.76 s | DOMContentLoaded: 229 ms | Load: 228 ms

# SUSPENSE

## React.lazy: Code splitting with Suspense [16.6]

```
const ChatPage = React.lazy(() => import("./chat/ChatPage"));  
class App {  
  render() {  
    return <>  
      <ChatPage />  
      // more pages...  
    </>  
  }  
}
```

**Dynamic Import**

# SUSPENSE

**React.Suspense:** Zeigt Fallback Komponente an [16.6]

- Bis Komponente geladen ist, muss Spinner o.ä. angezeigt werden

```
const ChatPage = React.lazy(() => import("./chat/ChatPage"));

class App {
  render() {
    return <>
      <React.Suspense fallback={<h1>Loading...</h1>}>
        <ChatPage />
        // more pages...
      </React.Suspense>
    <>
  }
}
```

## DEMO: LAZY UND SUSPENSE

### Problem: "Flickern"

- Entsteht, wenn Ladezeiten eher schnell sind

16.7-alpha unstable!

# Concurrent React

AUSBLICK

## SUSPENSE MIT CONCURRENT MODE

**Suspense:** Flickern verhindern mit Concurrent Mode

- `maxDuration` legt eine Zeit fest, bis `fallback` gerendert wird
- Bis dahin wird bestehende Komponente angezeigt
- Vor React 16.7 nicht / nur schwer möglich

```
const ChatPage = React.lazy(() => import("./chat/ChatPage"));

class App {
  render() {
    return <>
      <React.Suspense maxDuration={100} fallback={<h1>...</h1>}>
        <ChatPage />
        // more pages...
      </React.Suspense>
    <>
  }
}
```

## LAZY UND SUSPENSE

- **Demo: Fallback Komponente mit maxDuration [16.7-alpha]**

<http://localhost:9081>

## SUSPENSE

**Ausblick [16.7]:** Weitere Anwendungsfälle

- *Alle gezeigten Beispiele verwenden unstable API!!*

# BEISPIEL: DATEN LADEN MIT SUSPENSE

- REST Aufrufe mit fetch

The screenshot shows a web browser window titled "React Chat Example App" at "localhost:9081". The page is titled "React Chat Example" and displays "React 16.7.0-alpha.0". On the left, there's a sidebar with two sections: "Logs" and "User". The "Logs" section contains a list of log entries:

- [Anonymous-361] client disconnected
- [Anonymous-365] Assigned User id 'Anonymous-365'
- [Anonymous-365] Client registered
- [Anonymous-365] join chatroom with id 'r1'
- [Anonymous-366] Assigned User id 'Anonymous-366'
- [Anonymous-366] Client registered
- [Anonymous-366] join chatroom with id 'r1'
- [Anonymous-364] client disconnected
- [Anonymous-367] Assigned User id 'Anonymous-367'
- [Anonymous-367] Client registered

The "User" section displays a table of user data:

Id	Name
u1	Susi
u2	Klaus
u3	Harry
u4	Peter
u5	Maja
u6	Sue
u7	Olivia
u8	Cathy

At the bottom of the page is the URL: <https://github.com/nilshartmann/react-chat-example>.

On the left side of the screenshot, there are two labels with dashed lines pointing to specific parts of the interface:

- /api/logs ----- points to the "Logs" section in the sidebar.
- /api/users ----- points to the "User" section in the sidebar.

# ASYNCHRONES DATEN LADEN

- **"Klassisches" Daten laden**

- In componentDidMount Daten das Laden anstoßen
- In der Zwischenzeit Loading Indicator anzeigen

```
class LogsView extends React.Component {  
  state = {};  
  
  async componentDidMount() {  
    const response = await fetch("/api/logs");  
    const logs = await response.json();  
    this.setState({ logs })  
  }  
  
  render() {  
    if (!this.state.logs) { return <h1>Loading...</h1> }  
    return <> // render logs </>;  
  }  
}
```

# DATEN LADEN MIT SUSPENSE - 1

- **Daten laden mit Suspense**

- Vor dem Rendern wird Funktion aufgerufen die Daten liefert – oder auch nicht
- Sobald die Funktion (später) Daten liefert, wird die Komponente gerendert

```
function LogsView() {  
  const logs = LogsResource.read(); // kehrt nur mit Daten zurück  
  
  return <> ...geladene logs hier anzeigen... </>;  
}
```

## DATEN LADEN MIT SUSPENSE - 2

- **Daten laden mit Suspense**

- Vor dem Rendern wird Funktion aufgerufen die Daten liefert – oder auch nicht
- Sobald die Funktion (später) Daten liefert, wird die Komponente gerendert
- Komponente wird irgendwo im Tree mit **Suspense** umschlossen

```
function LogsView() {  
  const logs = LogsResource.read(); // kehrt nur mit Daten zurück  
  
  return <> ...geladene Logs hier anzeigen... </>;  
}  
  
function DashboardPage() {  
  return <Suspense maxDuration={...} fallback={...}>  
    <LogsView />  
  </Suspense>  
}
```

## DATEN LADEN MIT SUSPENSE - 3

- **react-cache** (zzt 2.0.0-alpha): Noch experimentell!
  - Geladene Daten (**Resourcen**) können gecached werden
  - Wenn Daten noch nicht vorhanden, werden sie vom Server gelesen

```
import { unstable_createResource } from "react-cache";

// Liefert Promise zurück
async function loadLogsFromApi() {

  const response = await fetch("http://localhost:9000/api/logs");
  return await response.json();

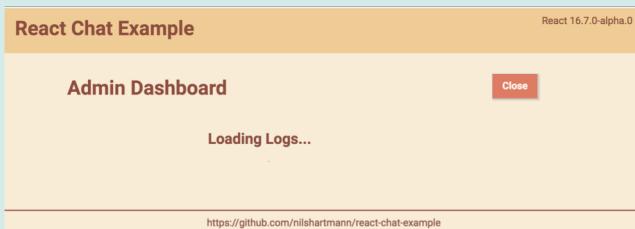
}

const LogsResource = unstable_createResource(loadLogsFromApi);
```

# DATEN LADEN MIT SUSPENSE

- Demo: Suspense an diversen Stellen

<http://localhost:9081/dashboard?delayfetch>



## HINTERGRUND: SUSPENSE

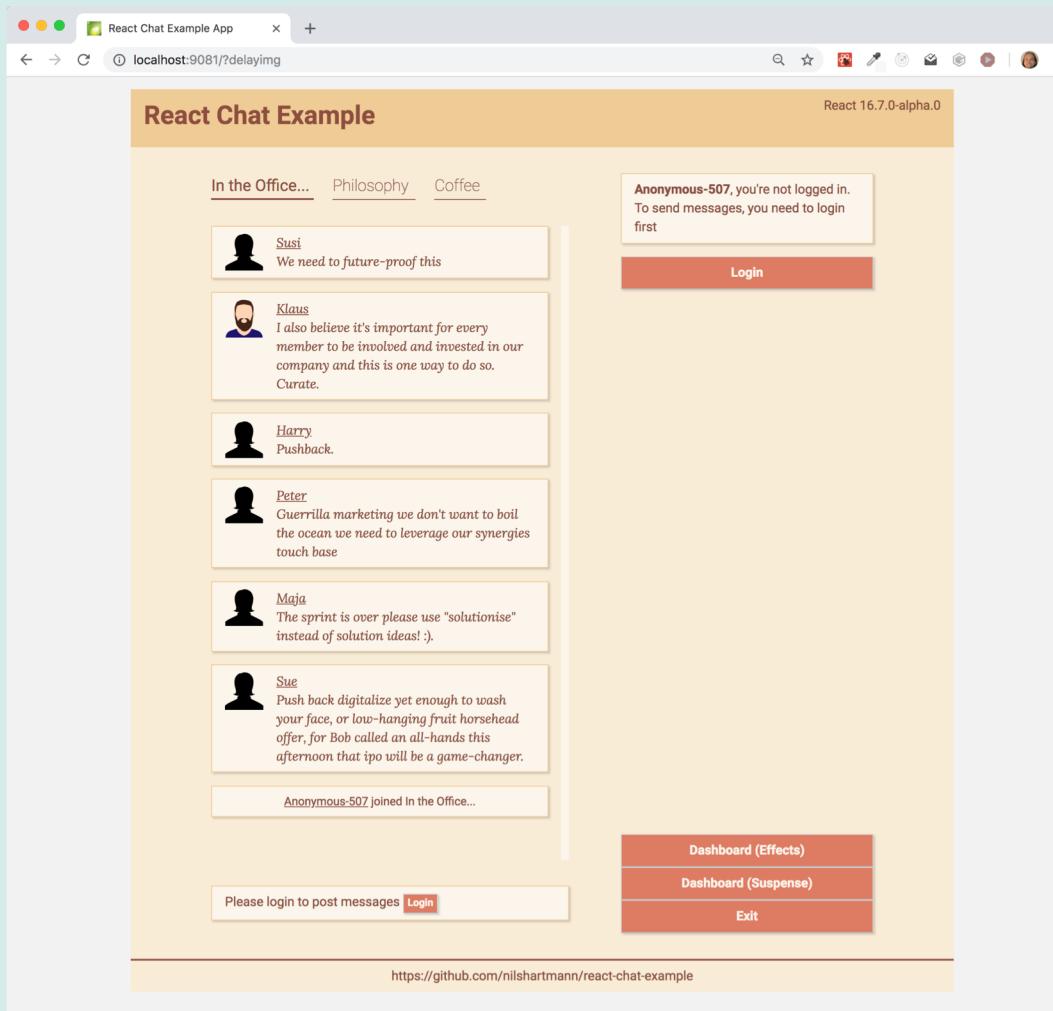
- Wie funktioniert das eigentlich?

```
function LogsView(props) {  
  const logs = LogsResource.read();  
  
  // ⏵ wird nur ausgeführt, wenn logs zurückgeliefert wird: 🤔🤔  
  return <> ... Logs hier anzeigen ... </>;  
}
```

# BEISPIEL: VORSCHAUEN MIT RESPONSE

- **Suspense nutzen, um Vorschauen zu laden**

Demo: <http://localhost:9081/?delayimg>



## BEISPIEL: VORSCHAUEN MIT SUSPENSE

### Vorher – ohne Vorschau, so wie gewohnt

```
function Avatar(props) {  
  const src = `/avatars/${props.userId}.svg`;  
  return <img className="Avatar" src={src} />;  
}  
  
function ChatMessage(props) {  
  
  return (  
    <div className="Message">  
      <Avatar userId={message.user.id} />  
      { props.message.text}  
      ...  
    </div>  
  );  
}
```

## BEISPIEL: VORSCHAUEN MIT SUSPENSE

### Avatar Komponente mit Suspense

```
function Avatar(props) {  
  const src = `/avatars/${props.userId}.svg`;  
  ImageResource.read(src); // <-- "Wartet" auf Image  
  return <img className="Avatar" src={src} />;  
}
```

credits: @jaredpalmer

<https://github.com/jaredpalmer/react-conf-2018/blob/master/full-suspense/src/components/ArtistDetails.js>

## BEISPIEL: VORSCHAUEN MIT SUSPENSE

### Image Resource

```
function Avatar(props) {  
  const src = `/avatars/${props.userId}.svg`;  
  ImageResource.read(src); // <-- "Wartet" auf Image  
  return <img className="Avatar" src={src} />;  
}  
  
const ImageResource = unstable_createResource(  
  source =>  
    new Promise(resolve => {  
      const img = new Image();  
      img.src = source;  
      img.onload = resolve;  
    })  
)
```

"Trick", um zu warten, bis der Browser ein Image geladen hat

credits: @jaredpalmer

<https://github.com/jaredpalmer/react-conf-2018/blob/master/full-suspense/src/components/ArtistDetails.js>

## BEISPIEL: VORSCHAUEN MIT SUSPENSE

### Einbinden

```
function ChatMessage(...) {  
  
  return (  
    <div className="Message">  
      <React.Suspense fallback={}>  
        <Avatar userId={message.user.id} />  
      </React.Suspense>  
  
      { props.message.text}  
      ...  
    </div>  
  );  
}
```

**16.7-alpha**

# Hooks

**FUNCTIONS EVERYWHERE**

[https://medium.com/@dan\\_abramov/making-sense-of-react-hooks-fdbde8803889](https://medium.com/@dan_abramov/making-sense-of-react-hooks-fdbde8803889)

If you create any content about Hooks while they're unstable, please mention prominently that they are an experimental proposal, and include a link to the official documentation.

# Hooks

FUNCTIONS EVERYWHERE

Unstable!  
Experimental!  
Proposal!

# Hooks

FUNCTIONS EVERYWHERE

Official Documentation

<https://reactjs.org/docs/hooks-intro.html>

# Hooks

FUNCTIONS EVERYWHERE

## HINTERGRUND

**Hooks:** State, Context etc auch in Funktionskomponenten

### Motivation:

- Bessere Wiederverwendbarkeit von Code
- Logik in Klassen nicht immer einfach verständlich (insb Lifecycles)

**Hooks sind reguläre Funktionen**

# USESTATE HOOK

## useState: State in Funktionskomponenten

### Beispiel: Tab Bar

In the Office...    Philosophy    Coffee

```
function Tabs(props) {  
  return <div>...</div>  
}
```

# USESTATE HOOK

## useState: State erzeugen

```
function Tabs(props) {  
  const [activeTabId, setActiveTabId] = React.useState(0);  
  |  
  | Aktueller State  
  |  
  | Setter  
  |  
  | Default Wert  
}  
|
```

# USESTATE HOOK

## useState: Aktuellen State verwenden

```
function Tabs(props) {  
  const [activeTabId, setActiveTabId] = React.useState(0);  
  
  return (  
    <div>  
      {props.tabs.map(tab => {  
        return <Tab  
          classname={tab.id === activeTabId ? "active" : ""}>  
            />  
        })}  
    </div>  
  );  
}
```

Zugreifen auf State

# USESTATE HOOK

## useState: State verändern

```
function Tabs(props) {  
  const [activeTabId, setActiveTabId] = React.useState(0);  
  
  return (  
    <div>  
      {props.tabs.map(tab => {  
        return <Tab  
          classname={tab.id === activeTabId ? "active" : ""}  
          onClick={() => setActiveTabId(tab.id)}  
        />  
      })}  
    </div>  
  );  
}
```

Setzen von State  
(kein Objekt mehr!)

## USESTATE HOOK

**useState**: Mehrere States in einer Komponente möglich

- Kein "mergen" von State mehr!

```
function LoginForm(props) {  
  const [username, setUsername] = React.useState("klaus");  
  const [password, setPassword] = React.useState("");  
  
  return (<>  
    <input value={username}  
          onChange={e => setUsername(e.target.value)} />  
  
    <input value={password}  
          onChange={e => setPassword(e.target.value)} />  
  </>);  
}
```

## ARBEITEN MIT SEITENEFFEKTEN

**Server-Zugriffe, Subscriptions etc sind Seiteneffekte**

## ARBEITEN MIT SEITENEFFEKTEN

### Server-Zugriffe, Subscriptions etc sind Seiteneffekte

- Bislang nur in Klassen-Komponenten

```
class ChatPage extends React.Component {  
  componentDidMount() {  
    this.disconnectFromApi = ChatApi.subscribe(this.props.apiKey);  
  }  
}
```

```
  render() { return <div><h1>Chat</h1>...</div> }  
}
```

## ARBEITEN MIT SEITENEFFEKTEN

### Server-Zugriffe, Subscriptions etc sind Seiteneffekte

- Bislang nur in Klassen-Komponenten

```
class ChatPage extends React.Component {  
  componentDidMount() {  
    this.disconnectFromApi = ChatApi.subscribe(this.props.apiKey);  
  }  
  
  componentWillUnmount() {  
    this.disconnectFromApi()  
  }  
  
  render() { return <div><h1>Chat</h1>...</div> }  
}
```

# ARBEITEN MIT SEITENEFFEKTEN

## Server-Zugriffe, Subscriptions etc sind Seiteneffekte

- Bislang nur in Klassen-Komponenten

```
class ChatPage extends React.Component {  
  componentDidMount() {  
    this.disconnectFromApi = ChatApi.subscribe(this.props.apiKey);  
  }  
  
  componentWillUnmount() {  
    this.disconnectFromApi()  
  }  
  
  componentDidUpdate(prevProps) {  
    if (prevProps.apiKey !== this.props.apiKey) {  
      ChatApi.subscribe(this.props.apiKey);  
    }  
  }  
  
  render() { return <div><h1>Chat</h1>...</div> }  
}
```

Nur ausführen, wenn Properties sich geändert haben

# ARBEITEN MIT SEITENEFFEKTEN

## useEffect: Seiteneffekte in Funktionskomponenten

```
function ChatPage(props) {  
  React.useEffect(  
    () => { ----- Ersetzt componentDidMount & componentDidUpdate  
            const disconnectFromApi = ChatApi.subscribe(props.apiKey);  
  
    },  
  );  
  
  return <div><h1>Chat</h1>...</div>  
}
```

# ARBEITEN MIT SEITENEFFEKTEN

## useEffect: Seiteneffekte in Funktionskomponenten

Aufräumen in Rückgabe-Funktion

```
function ChatPage(props) {  
  React.useEffect(  
    () => {  
      const disconnectFromApi = ChatApi.subscribe(props.apiKey);  
      return () => disconnectFromApi();  
    },  
    |  
  );  
  
  return <div><h1>Chat</h1>...</div>  
}
```

# ARBEITEN MIT SEITENEFFEKTEN

## useEffect: Seiteneffekte in Funktionskomponenten

### Bedingte Ausführung

```
function ChatPage(props) {  
  React.useEffect(  
    () => {  
      const disconnectFromApi = ChatApi.subscribe(props.apiKey);  
      return () => disconnectFromApi();  
    },  
    [props.apiKey] ----- Ersetzt Property-Vergleich in componentDidUpdate  
  );  
  
  return <div><h1>Chat</h1>...</div>  
}
```

## WEITERE HOOKS

### (Fast) alles geht jetzt mit Hook

- useState
- useEffect
- useContext
- useRef
- useReducer
- (Noch offen: Error Boundaries)

## CUSTOM HOOKS

**Eigene Hooks sind möglich** und können wiederverwendet werden

- Beispiel: Handler für Input-Felder

## CUSTOM HOOKS

Eigene Hooks sind möglich und können wiederverwendet werden

- Beispiel: Handler für Input-Felder

```
function useFormInput(initialValue, onEnter) {  
  const [value, setValue] = React.useState(initialValue);  
  
  function onEnterHandler(e) {  
    const keyCode = e.which || e.keyCode;  
    if (keyCode === 13) {  
      onEnter(value);  
    }  
  }  
  
  return {  
    value,  
    onChange: e => setValue(e.target.value),  
    onKeyPress: onEnterHandler  
  };  
}
```

## CUSTOM HOOKS

Eigene Hooks sind möglich und können wiederverwendet werden

- Beispiel: Handler für Input-Felder

```
function useFormInput(initialValue, onEnter) { ... }

// Verwendung:
function LoginDialog(props) {
  const usernameInput = useFormInput("", ChatApi.login);

  return <form>
    <input {...usernameInput} />
  </form>
}
```

## CUSTOM HOOKS

Eigene Hooks sind möglich und können wiederverwendet werden

- Beispiel: Generischer "fetch hook"
- Alle Hooks können verwendet werden

```
function useApi(path, initData) {  
  const [data, setData] = React.useState(initialData);  
  
  React.useEffect(async () => {  
    const response = await fetch(`http://localhost:9000/${path}`);  
    const data = await response.json();  
  
    setData(data);  
  }, [path]);  
  
  return data;  
}
```

## CUSTOM HOOKS

Eigene Hooks sind möglich und können wiederverwendet werden

- Beispiel: Generischer "fetch hook"

```
function useApi(path, initialValue) { ... }

// Verwendung
function Dashboard(props) {
  const logs = useApi("/logs", []);
  const users = useApi("/users", []);

  return <>
    <LogViewer logs={logs} />
    <UsersViewer users={users} />
  </>;
}
```

# HOOKS

- **Müssen wir jetzt alle Hooks verwenden? 😱**
- **Was ist mit unseren Klassen? 😱**

# HOOKS

- **Müssen wir jetzt alle Hooks verwenden?** 😱
- **Was ist mit unseren Klassen?** 😱
- **Zunächst:**
  - Hooks sind "opt-in"
  - Hooks sind abwärtskompatibel
  - Eingeführt in Minor-Version (!)

# HOOKS

- Müssen wir jetzt alle Hooks verwenden? 😱
- Was ist mit unseren Klassen? 😱
- ...also: keine Panik! React bleibt stabil! 😊

Finally, there is no rush to migrate to Hooks. We recommend avoiding any “big rewrites”, especially for existing, complex class components. It takes a bit of a mindshift to start “thinking in Hooks”. In our experience, it’s best to practice using Hooks in new and non-critical components first, and ensure that everybody on your team feels comfortable with them. After you give Hooks a try, please feel free to send us feedback, positive or negative.

We intend for Hooks to cover all existing use cases for classes, but **we will keep supporting class components for the foreseeable future**. At Facebook, we have tens of thousands of components written as classes, and we have absolutely no plans to rewrite them. Instead, we are starting to use Hooks in the new code side by side with classes.

@NILSHARTMANN

# vielen Dank!

Slides: <https://bit.ly/wjax2018-react>

Beispiel-Code: <http://bit.ly/wjax2018-react-example>

KONTAKT@NILSHARTMANN.NET