



NILS HARTMANN
<https://nilshartmann.net>

Neues in

React 18

Slides: <https://react.schule/ejs-2022>

NILS HARTMANN

nils@nilshartmann.net

Freiberuflicher Entwickler, Architekt, Trainer aus Hamburg

Java, TypeScript, React



<https://graphql.schule/video-kurs>



<https://reactbuch.de>

HTTPS://NILSHARTMANN.NET

NEUE DOKUMENTATION

[https://beta.reactjs.org:](https://beta.reactjs.org) Komplett neue Dokumentation

- "Hooks-first"

React

18

React

Auto Batching

Veränderter Strict-Mode

Neue Hooks

Concurrent React

Kein Support für Internet Explorer mehr!

In this release, React is dropping support for Internet Explorer, which is going out of support on June 15, 2022. We're making this change now because new features introduced in React 18 are built using modern browser features such as microtasks which cannot be adequately polyfilled in IE.

If you need to support Internet Explorer we recommend you stay with React 17.

Vor einer Woche!

<https://reactjs.org/blog/2022/03/08/react-18-upgrade-guide.html#dropping-support-for-internet-explorer>

AUTO BATCHING

Renderings in Promises

- Ab React 18: Set-State-Aufrufe in Promises werden zusammengefasst

```
function Blog() {  
  const [post, setPost] = useState(null);  
  const [loading, setLoading] = useState("");  
  
  function loadPost() {  
    fetch("...")  
      .then(data => {  
        setPost(data);  
        setLoading(false);  
      })  
  }  
  
  return <div>  
    <button onClick={loadPost}>Load Blog Post</button>  
  </div>;  
}
```

Ein Renderzyklus
(vor React 18 zwei Renderzyklen)

VERÄNDERTER STRICT-MODE

Strict-Mode: Komponenten werden zweimal gemounted

```
// index.js

render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

- Bisher: React rendert Komponenten zweimal

VERÄNDERTER STRICT-MODE

Strict-Mode: Komponenten werden zweimal gemounted

```
// index.js

render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

- Bisher: React rendert Komponenten zweimal
- React 18: **Auch Effekte werden doppelt ausgeführt**
 - Dadurch kann (manuell) geprüft werden, ob alle Clean-Up-Funktion in Effekten richtig funktionieren

VERÄNDERTER STRICT-MODE

Strict-Mode: Potentielle Ressourcen-Leaks erkennen

```
function Chat() {  
  React.useEffect(() => {  
    subscribeToApi();  
  });  
  return <div>. . .</div>;  
}
```

VERÄNDERTER STRICT-MODE

Strict-Mode: Potentielle Ressourcen-Leaks erkennen

```
function Chat() {  
  React.useEffect(() => {  
    subscribeToApi(); ----- Problem: Subscription  
    ----- wird nicht abgebrochen  
  });  
  
  return <div>. . .</div>;  
}
```

VERÄNDERTER STRICT-MODE

Strict-Mode: Potentielle Ressourcen-Leaks erkennen

```
function Chat() {  
  React.useEffect(() => {  
    subscribeToApi(); // Problem: Subscription wird nicht abgebrochen  
  });  
  
  return <div>...</div>;  
}
```

Problem: Subscription wird nicht abgebrochen

```
function Chat() {  
  React.useEffect(() => {  
    const id = subscribeToApi();  
    return () => cancelSubscription(id); // Richtig: Subscription in Clean-up-Funktion beenden  
  });  
  
  return <div>...</div>;  
}
```

Richtig: Subscription in Clean-up-Funktion beenden

NEUE HOOKS

useId: Generiert eindeutige Ids, die bei Server- und Client-seitig identisch sind (wichtig für SSR)

```
function Form() {
  const infoId = React.useId();

  return <form>
    <label>
      Title:
      <input aria-describedby={infoId} value="..." />
    </label>
    <span id={infoId} className="info">Enter new Blog Title</span>
  </form>;
}
```

NEUE HOOKS

- **useSyncExternalStore** und **useInsertionEffect**: nur für Bibliotheken
- **useTransition/useDeferredValue**: Updates als "nicht-dringend" markieren
(Concurrent React)

React

Auto Batching

Veränderter Strict-Mode

Neue Hooks

Concurrent React

REACT 18: WHAT'S IN THE BOX?

Ziel: Bessere Ausnutzung der CPU Ressourcen

- Nicht bessere "Performance"
- Flüssigerer Eindruck für User

DEMO

Demo 

- <http://localhost:3002>

USETRANSITION HOOK

useTransition: Markiert Update als "weniger wichtig"

```
function ExampleChart() {
  const [input, setInput] = React.useState("");
  const [data, setData] = React.useState(initialData);

  const [isPending, startTransition] = React.useTransition();
  
    Läuft Transition gerade ?
    (ist 'data' aktuell)
    Transition starten
}

}
```

USETRANSITION HOOK

useTransition: Markiert Update als "weniger wichtig"

```
function ExampleChart() {
  const [input, setInput] = React.useState("");
  const [data, setData] = React.useState(initialData);

  const [isPending, startTransition] = React.useTransition();

  function handleInputChange(e) {
    setInput(e.target.value);
```

"Normale" Updates sind
"dringend" (wie bisher)

```
}
```

USETRANSITION HOOK

useTransition: Markiert Update als "weniger wichtig"

```
function ExampleChart() {
  const [input, setInput] = React.useState("");
  const [data, setData] = React.useState(initialData);

  const [isPending, startTransition] = React.useTransition();

  function handleInputChange(e) {
    setInput(e.target.value);

    startTransition(() => {
      setData(/* ... */);
    });
  }
}
```

Updates innerhalb der Callback-Funktion sind "nicht dringend"

USETRANSITION HOOK

useTransition: Markiert Update als "weniger wichtig"

```
function ExampleChart() {
  const [input, setInput] = React.useState("");
  const [data, setData] = React.useState(initialData);

  const [isPending, startTransition] = React.useTransition();

  function handleInput(e) {
    setInput(e.target.value);

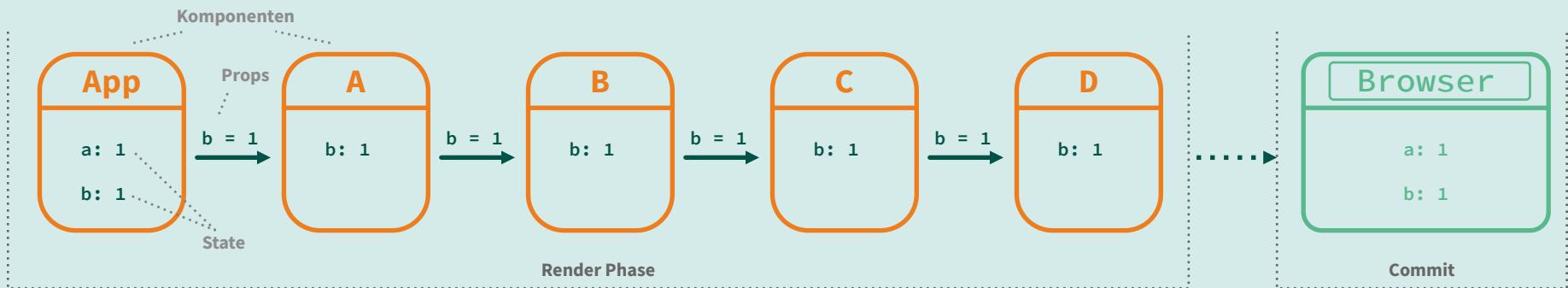
    startTransition(() => {
      setData(/* ... */);
    });
  }

  return (<div>
    <input value={input} onChange={handleInput} />
    <p>{isPending ? "Pending!" : "Not Pending"}</p>
    <TheChart data={data} />
  </div>
);
}
```

isPending zeigt an, das nicht dringende
Updates noch ausstehen
(data ist nicht aktuell)

RENDER UND COMMIT (KLASSISCH)

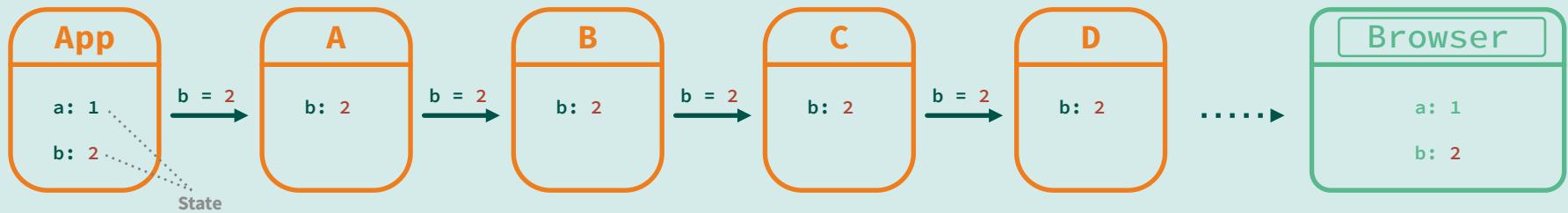
Klassisches React: Rendern ist ein Vorgang



RENDER UND COMMIT (KLASSISCH)

Klassisches React: Nach State-Update wird alles neu gerendert und committed

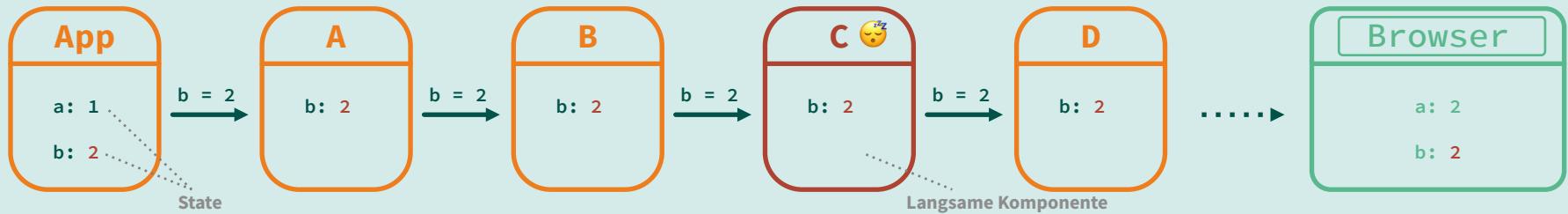
setB(2);



RENDER UND COMMIT (KLASSISCH)

Klassisches React: Nach State-Update wird alles neu gerendert und committed

```
setB(2);
```

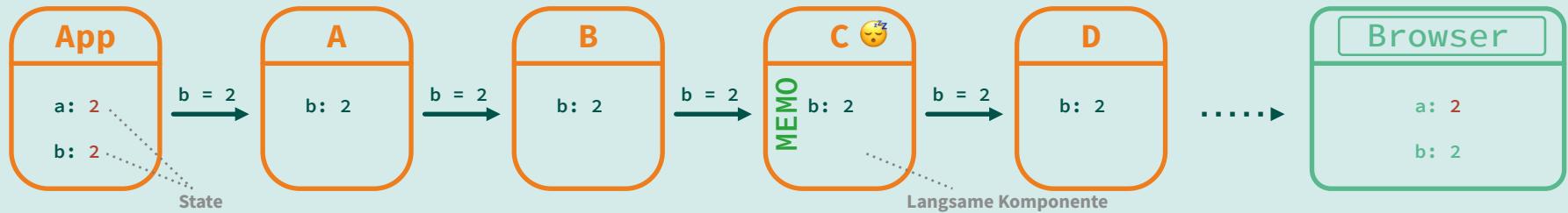


Rendern dauert lange, kann aber nicht unterbrochen werden 😴

RENDER UND COMMIT (KLASSISCH)

Klassisches React: Rendern ist ein Vorgang

```
setA(2); setB(2);
```

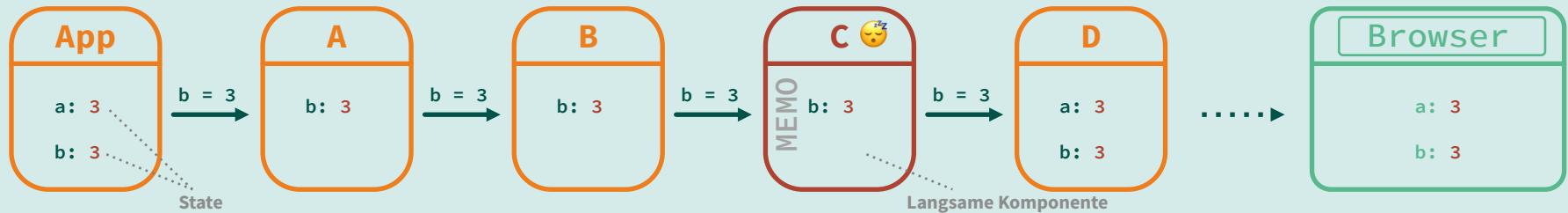


Rendern geht schnell(er), C kommt aus dem "Cache" 👍

RENDER UND COMMIT (KLASSISCH)

Klassisches React: Rendern ist ein Vorgang

```
setA(3); setB(3);
```



Rendern wieder langsam 🤐

RENDER UND COMMIT (CONCURRENT)

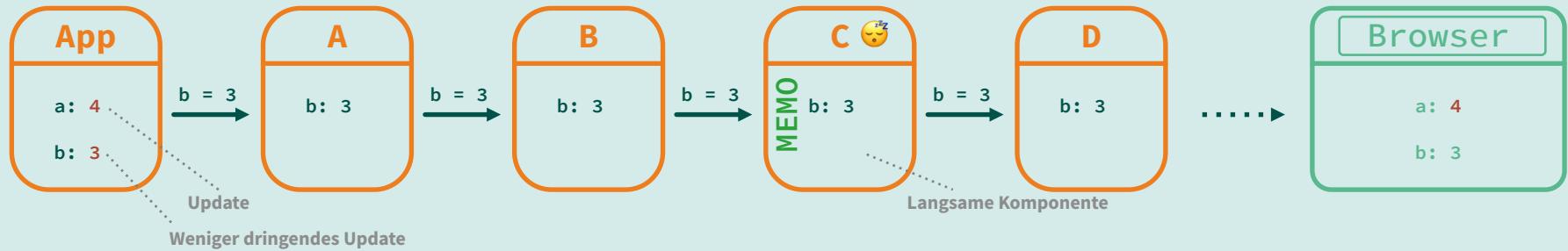
Concurrent React: Dringende und weniger dringende Updates

```
setA(4); startTransition( () => setB(4) );
```

RENDER UND COMMIT (CONCURRENT)

Concurrent React: Updates können priorisiert werden

```
setA(4); startTransition( () => setB(4) );
```

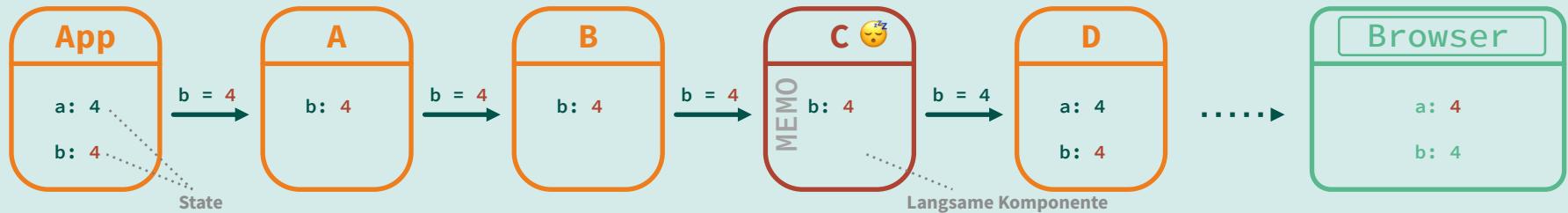


Erst wird "dringendes" Update für **a** durchgeführt...

RENDER UND COMMIT (CONCURRENT)

Concurrent React: Updates können priorisiert werden

```
setA(4); startTransition( () => setB(4) );
```

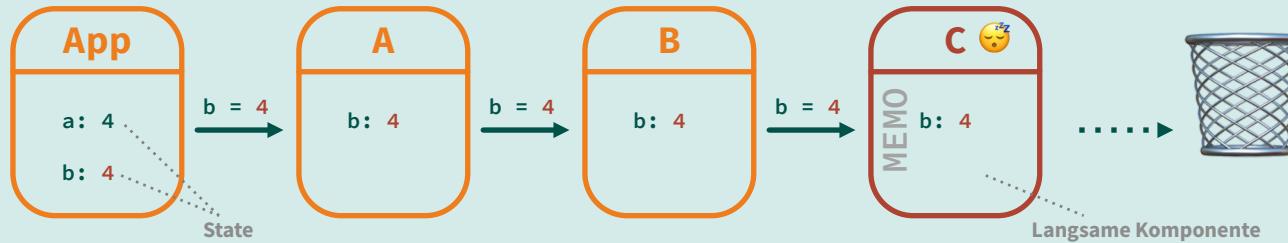


...danach wird das "nicht dringende" Update für **b** durchgeführt

RENDER UND COMMIT (CONCURRENT)

Concurrent React: Updates können abgebrochen werden

```
setA(5); startTransition( () => setB(4) );
```



...danach wird das "nicht dringende" Update für **b** durchgeführt

CONCURRENT REACT - DEMO

Demo: Rendering ab- und unterbrechen



- <http://localhost:3001>

CONCURRENT REACT - DEMO

- Demo

```
function App() {  
  const [count, setCount] = useState(0);  
  const [notSoUrgentValue,  
        setNotSoUrgentValue] = useState(0);  
  const [, startTransition] = useTransition();  
  
  function handleClick() {  
    // urgent update ("count")  
    setCount(c => c + 1);  
  
    startTransition()  
      () => setNotSoUrgentValue(x => x + 1)  
  };  
  
  return (  
    <div className="App">  
      <button onClick={handleClick}>Go!</button>  
  
      <p>Count: {count}</p>  
      <A notSoUrgentValue={notSoUrgentValue} />  
    </div>  
  );  
}
```

15:54:23.574	Commit	App count: 15, notSoUrgentValue: 14
15:54:23.575	Render	App count: 15, notSoUrgentValue: 15
15:54:23.575	Render	A notSoUrgentValue: 15
15:54:23.575	Render	B notSoUrgentValue: 15
15:54:23.576	Render	C notSoUrgentValue: 15
15:54:24.527	Render	App count: 16, notSoUrgentValue: 14
15:54:24.528	Render	A notSoUrgentValue: 14
15:54:24.528	Render	B notSoUrgentValue: 14
15:54:24.528	Render	C notSoUrgentValue: 14
15:54:24.528	Render	D notSoUrgentValue: 14
15:54:24.528	Commit	D notSoUrgentValue: 14
15:54:24.529	Commit	C notSoUrgentValue: 14
15:54:24.529	Commit	B notSoUrgentValue: 14
15:54:24.529	Commit	A notSoUrgentValue: 14
15:54:24.529	Commit	App count: 16, notSoUrgentValue: 14

USE DEFERRED VALUE

useDeferredValue: Aufschieben von Aktualisierungen

Beispiel: vorübergehendes Cachen der UI

USE DEFERRED VALUE

useDeferredValue: Aufschieben von Aktualisierungen

Beispiel: vorrübergehendes Cachen der UI

```
function TheChart({data}) {  
  
    const ui = useMemo(() => (  
        <div>  
            {data.map((d) => (  
                <Bar key={d.id} {...d} />  
            ))}  
        </div>  
    ),  
    [data]  
);  
  
    return ui;  
}
```

USE DEFERRED VALUE

useDeferredValue: Aufschieben von Aktualisierungen

|

```
function TheChart({data}) {  
  const ui = useMemo(() => (  
    <div>  
      {data.map((d) => (  
        <Bar key={d.id} {...d} />  
      ))}  
    </div>  
  ),  
  [data]  
);  
  
  return ui;  
}
```

Problem: data ist hier immer aktuell...

USE DEFERRED VALUE

useDeferredValue: Aufschieben von Aktualisierungen

```
function TheChart({data}) {  
  const ui = useMemo(() => (  
    <div>  
      {data.map((d) => (  
        <Bar key={d.id} {...d} />  
      ))}  
    </div>  
  ),  
  [data]  
);  
  
  return ui;  
}
```

Problem: data ist immer aktuell...

...UI wird immer aktualisiert 😢

USE DEFERRED VALUE

useDeferredValue: Liefert vorherigen Wert zurück, wenn gerade "wichtiges" Update gerendert wird

```
function TheChart({data}) {
  const deferredData = React.useDeferredValue(data);

  const ui = useMemo(() => (
    <div>
      {deferredData.map((d) => (
        <Bar key={d.id} {...d} />
      ))}
    </div>
  ),
  [deferredData]
);

return ui;
}
```

kann 'data' sein oder 'data' von
vorherigem Rendern

DEFERRED VALUE DEMO

Demo 

- <http://localhost:3002>

DEFERRED VALUE DEMO

Demo

- <http://localhost:3002>

17:36:32.166	Render	TheChart barData from props: '1', deferredValue: '1'
17:36:37.965	Render	TheChart barData from props: '2', deferredValue: '1'
17:36:38.115	Render	TheChart barData from props: '3', deferredValue: '1'
17:36:38.206	Render	TheChart barData from props: '4', deferredValue: '1'
17:36:38.297	Render	TheChart barData from props: '4', deferredValue: '4'
17:36:39.454	Commit	TheChart barData from props: '4', deferredValue: '4'

React

10

Ausblick (experimentell)

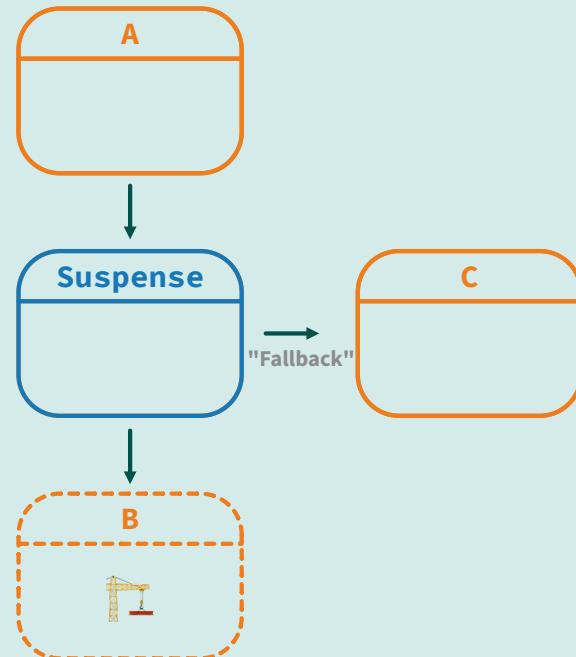
REACT: WHAT'S NEXT?

Suspense for Data Detching

SUSPENSE

Suspense: Unterbricht das Rendern, solange "etwas" fehlt

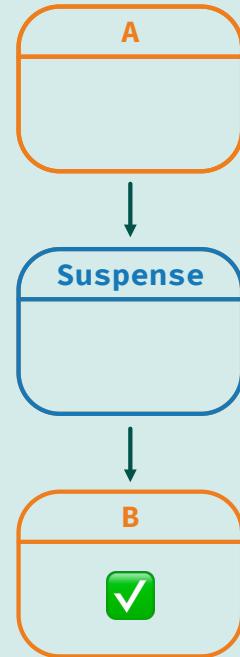
- Asynchron geladenes JS, asynchron geladene Daten, Promises, ...



SUSPENSE

Suspense: Unterbricht das Rendern, solange "etwas" fehlt

- Asynchron geladenes JS, asynchron geladene Daten, Promises, ...



SUSPENSE

Suspense: Unterbricht das Rendern, solange "etwas" fehlt

- Asynchron geladenes JS, asynchron geladene Daten, Promises, ...
- Funktioniert seit React 16.6 für dynamische Imports stabil

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

function App() {
  return <Router>
    <Route path="/user">
      <React.Suspense fallback={<h1>Stay tuned! </h1>}>
        <UserProfile />
      </React.Suspense>
    </Route>
  <Router>;
}
```

SUSPENSE

Suspense for Data Fetching

- Experimentell z.B. in SWR, React Query
- Insbesondere die interne API ist noch nicht stabil

SUSPENSE

Suspense for Data Fetching: Demo 🕵️

- localhost:3003

- Homepage / BlogList
- PostPage
- BlogTeaserList, useTransition

Zero-Bundle-Size

Server Components

Idee: Neue Art von Komponenten

- "Server Components" werden nur auf dem Server ausgeführt
- Sie stehen nicht auf dem Client zur Verfügung
- Der Server schickt lediglich eine *Repräsentation der UI, aber keinen Code*

👉 "Zero-Bundle-Size"

Demo: Server Components

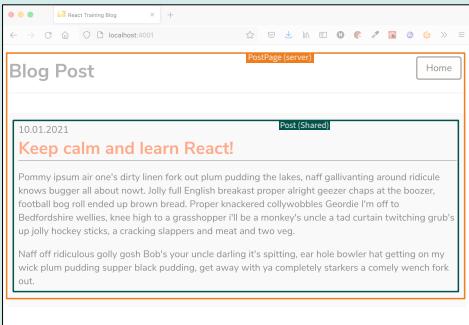


- <http://localhost:4001>
- PostListPage.server.js
 - "Normale" Komponente (JSX etc.)
 - Läuft aber nur auf dem Server
 - Verwendet **PostList.server.js** (=> Server Resourcen)
 - In den Dev Tools finden wir die Komponente nicht
 - Aber im Netzwerk-Tab wird dafür "irgendwas" übertragen
- Code: <https://github.com/nilshartmann/server-components-blogexample>

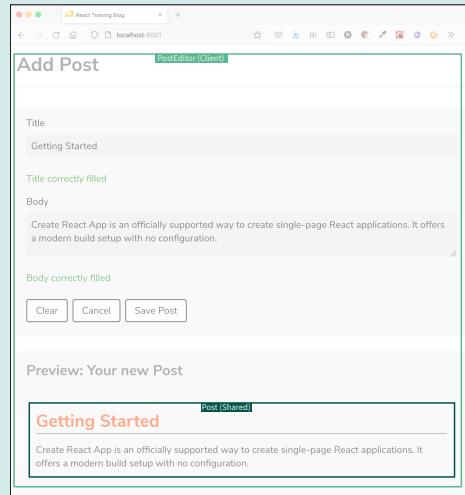
SERVER COMPONENTS

Demo: Shared Components

- JS-Code wird erst bei Bedarf auf den Client geladen (ansonsten nur UI)



Verwendung "Post"-Komponente 1:
innerhalb einer Server-Komponente



Verwendung "Post"-Komponente 2:
innerhalb einer Client-Komponente



Demo

- Post-Seite: keine "Post-Komponente"
- PostEditor: Post-Komponente wird geladen (-> Netzwerk-Tab) und als Komponente gerendert (-> Dev Tools)
- Netzwerk-Tab: unten ist der JS-Code der Komponente



NILS HARTMANN
<https://nilshartmann.net>

vielen Dank!

Slides: <https://react.schule/ejs-2022>

Kontakt: nils@nilshartmann.net