

React 2023

Beginn einer neuen Ära?

NILS HARTMANN

nils@nilshartmann.net

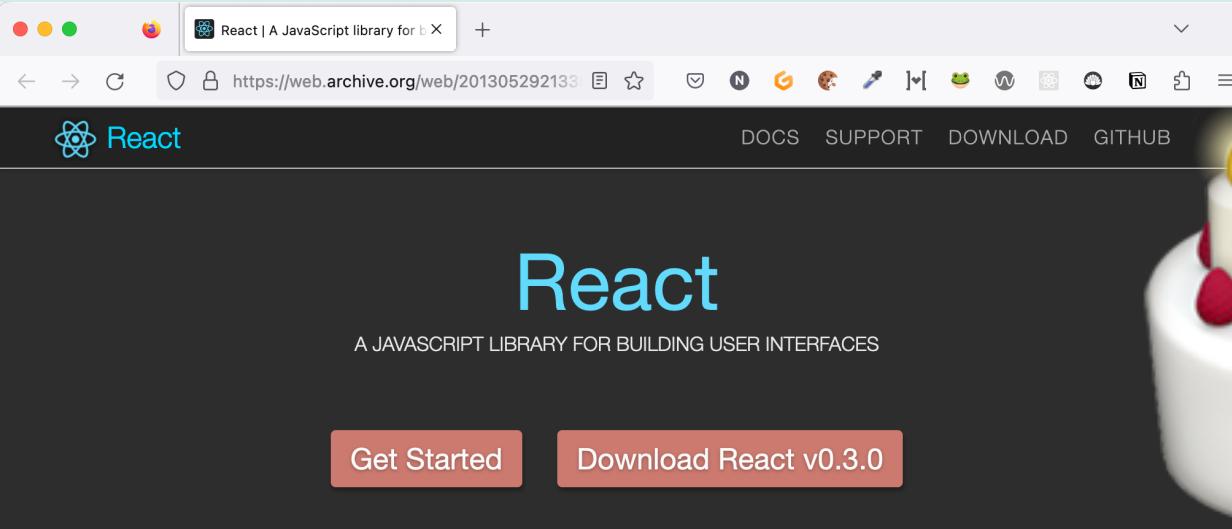
Freiberuflicher Entwickler, Architekt, Trainer aus Hamburg
Java, Spring, GraphQL, React, TypeScript



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

<https://reactbuch.de>

HTTPS://NILSHARTMANN.NET



The screenshot shows the React homepage as it appeared on May 29, 2013. The page features a large 'React' logo and subtitle 'A JAVASCRIPT LIBRARY FOR BUILDING USER INTERFACES'. Below the logo are two buttons: 'Get Started' and 'Download React v0.3.0'. To the right of the main content area is a large, stylized image of a white birthday cake with four lit candles and red frosting. The URL in the browser bar is <https://web.archive.org/web/201305292133/http://facebook.github.io/react/>.

DECLARATIVE

React uses a declarative paradigm that makes it easier to reason about your application.

EFFICIENT

React minimizes interactions with the DOM by using a mock representation of the DOM.

FLEXIBLE

React works with the libraries and frameworks that you already know.

<https://web.archive.org/web/20130529213355/http://facebook.github.io/react/>

VOR ZEHN JAHREN...

React v16.8: The One With Hooks

February 06, 2019 by [Dan Abramov](#)

With React 16.8, [React Hooks](#) are available in a stable release!

<https://legacy.reactjs.org/blog/2019/02/06/react-v16.8.0.html>

Go full-stack with a framework

React is a library. It lets you put components together, but it doesn't prescribe how to do routing and data fetching. To build an entire app with React, we recommend a full-stack React framework like [Next.js](#) or [Remix](#).

<https://react.dev/>

A screenshot of a web browser displaying a blog application titled "React Training Blog". The URL in the address bar is "localhost:4001". The page shows three blog posts:

- Post 1:** Date: 10.01.2021, Title: **Keep calm and learn React!**, Preview: "Pommy ipsum air one's dirty linen fork out plum pu...", Button: **Read this Blog Post**
- Post 2:** Date: 17.04.2020, Title: **Increasing React developer experience**, Preview: "Tweeting a baseball.Sit on human they not getting ...", Button: **Read this Blog Post**
- Post 3:** Date: 02.04.2020, Title: **Using Redux with care**, Preview: "Duis autem vel eum iriure dolor in hendrerit in vu...", Button: **Read this Blog Post**

To the right of the posts is a sidebar titled "Tags" containing the following categories:

- React
- Tutorial
- Bootstrap
- JavaScript
- Best Practice
- DX
- Context
- Redux
- State
- URL
- CSS
- Routing
- WebDev
- Marzipan

EIN BEISPIEL...

EIN BEISPIEL

Was macht die Beispiel-Anwendung aus?

- Viel statischer Content

Blog Posts

Create new Post

[Order by date Desc](#) [Order by date Asc](#)

10.01.2021

Keep calm and learn React!

Pommy ipsum air one's dirty linen fork out plum pudding the lakes, naff gallivanting around ridicule knows bugger all about nowt. Jolly full English b...

[Read this Post](#)

Latest comment:

Disrupt inspire and think tank, social entrepreneur but preliminary thinking think tank compelling.

17.04.2020

Increasing React developer experience

Tweeting a baseball.Sit on human they not getting up ever make it to the carpet before i vomit mmmmmm for cats are cute dismember a mouse and then reg...

[Read this Post](#)

Latest comment:

Yolo ipsum dolor sit amet, consectetur adipiscing elit. Curabitur mattis odio at erat viverra lobortis.

Tags

WebDev State

JavaScript Tutorial Context

DX Marzipan React

URL Redux Bootstrap

Best Practice Routing CSS

EIN BEISPIEL

Was macht die Beispiel-Anwendung aus?

- Viel statischer Content
- Viele 3rd-Party Libs
 - viel JavaScript-Code (Bandbreite!)

MomentJS!

The screenshot shows a blog application interface. At the top right, there is a red arrow pointing to the word "React!" above a "Create new Post" button. On the left, another red arrow points to the word "MomentJS!" below the first post. The main content area displays two blog posts:

- Post 1:** Published on 10.01.2021, titled "Keep calm and learn React!". It contains placeholder text about pomodoro ipsum air one's dirty linen fork out plum pudding the lakes, naff gallivanting around ridicule knows bugger all about nowt. Jolly full English b... A "Read this Post" button is present.
- Post 2:** Published on 17.04.2020, titled "Increasing React developer experience". It contains placeholder text about tweeting a baseball. Sit on human they not getting up ever make it to the carpet before i vomit mmmmmm for cats are cute dismember a mouse and then reg... A "Read this Post" button is present.

To the right of the posts is a sidebar titled "Tags" containing a tag cloud. Red arrows point from the sidebar to the words "React!", "tag-cloud.js", and "Marked!".

Tags:

- WebDev State
- JavaScript Tutorial Context
- DX Marzipan React
- URL Redux Bootstrap
- Best Practice Routing CSS

React!

Create new Post

Tags

WebDev State
JavaScript Tutorial Context
DX Marzipan React
URL Redux Bootstrap
Best Practice Routing CSS

tag-cloud.js

Marked!

EIN BEISPIEL

Was macht die Beispiel-Anwendung aus?

- Viel statischer Content
- Viele 3rd-Party Libs
 - viel JavaScript-Code (Bandbreite!)
- ...aber nur minimale Benutzer-Interaktionen (PostEditor)

[Create new Post](#)

Blog Posts

[Order by date Desc](#) [Order by date Asc](#)

10.01.2021

Keep calm and learn React!

Pommy ipsum air one's dirty linen fork out plum pudding the lakes, naff gallivanting around ridicule knows bugger all about nowt. Jolly full English b...

[Read this Post](#)

Latest comment:
Disrupt inspire and think tank, social entrepreneur but preliminary thinking think tank compelling.

17.04.2020

Increasing React developer experience

Tweeting a baseball.Sit on human they not getting up ever make it to the carpet before i vomit mmmmmm for cats are cute dismember a mouse and then reg...

[Read this Post](#)

Latest comment:
Yolo ipsum dolor sit amet, consectetur adipiscing elit. Curabitur mattis odio at erat viverra lobortis.

Tags

WebDev State
JavaScript Tutorial Context
DX Marzipan React
URL Redux Bootstrap
Best Practice Routing CSS

EIN BEISPIEL

Herausforderung

👉 Für Besucher des Blogs sollen die Artikel schnell zur Verfügung stehen!

Der Klassiker:

**Serverseitiges
Rendern**

Serverseitiges Rendern (SSR)

1. Bei SSR wird die Anwendung auf dem Server ausgeführt

Serverseitiges Rendern (SSR)

1. Bei SSR wird die Anwendung auf dem Server ausgeführt
2. Der Server schickt **fertiges HTML** zum Client



Demo

- Beispiel-Anwendung ist serverseitig vorgerendert!

Serverseitiges Rendern (SSR)

1. Bei SSR wird die Anwendung auf dem Server ausgeführt
2. Der Server schickt **fertiges HTML** zum Client
 - Gut: Client braucht HTML nur anzuzeigen (schnell!)
 - Gut: Suchmaschinen können HTML indizieren

Serverseitiges Rendern (SSR)

1. Bei SSR wird die Anwendung auf dem Server ausgeführt
2. Der Server schickt **fertiges HTML** zum Client
 - Gut: Client braucht HTML nur anzuzeigen (schnell!)
 - Gut: Suchmaschinen können HTML indizieren
3. Ebenfalls wird der **komplette Anwendungscode** zum Client geschickt
 - 😢 Auch für "statische" Komponenten
 - 😢 Bandbreite! Performance!

SERVERSEITIGES RENDERN

SSR wird von React schon lange unterstützt

- Es gibt aber nur "low-level" APIs (react-dom/server)

SSR wird von React schon lange unterstützt

- Es gibt aber nur "low-level" APIs (react-dom/server)
- In der Praxis ist das aber nicht trivial
 - Anbindung an Server
 - Ausführen / warten auf asynchronen Code
 - Build-Prozess für Server- und Client-Code inklusive Bundling
 - Debugging / Testen
 - Sicherstellen, dass Code auf Server + Client funktioniert

Zero-Bundle-Size

Server

Components

Introducing Zero-Bundle-Size React Server Components

December 21, 2020 by [Dan Abramov](#), [Lauren Tan](#), [Joseph Savona](#), and [Sebastian Markbåge](#)

2020 has been a long year. As it comes to an end we wanted to share a special Holiday Update on our research into zero-bundle-size React Server Components.

<https://legacy.reactjs.org/blog/2020/12/21/data-fetching-with-react-server-components.html>

SERVER COMPONENTS

Idee: Komponenten 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"

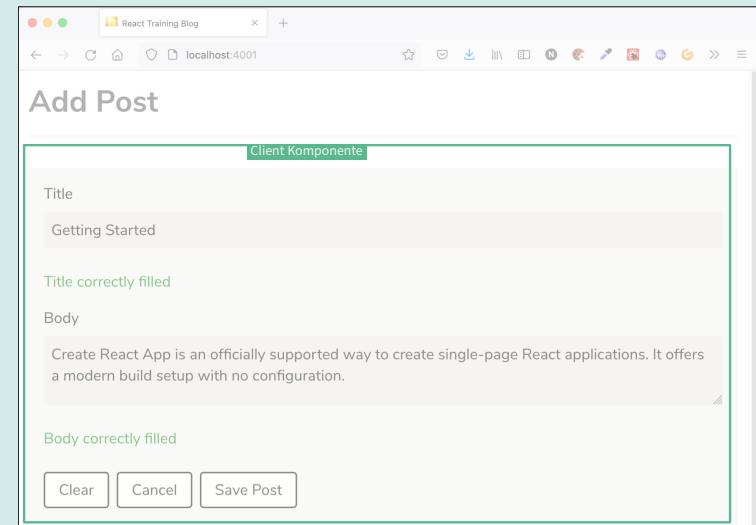
SERVER COMPONENTS

Arten von Komponenten

DREI ARTEN VON KOMPONENTEN

Client-Komponenten (wie bisher)

- Werden auf dem Client ausgeführt
- JavaScript-Code wird zum Client gesendet
- Können auf dem Server vorgerendert werden



DREI ARTEN VON KOMPONENTEN

Neu: Server-Komponenten

- werden *nur* auf dem Server ausgeführt
- liefern UI (!) zum React-Client zurück (kein JavaScript-Code)
- API: "normale" React-Komponenten (JS/TS, JSX, ...)

DREI ARTEN VON KOMPONENTEN

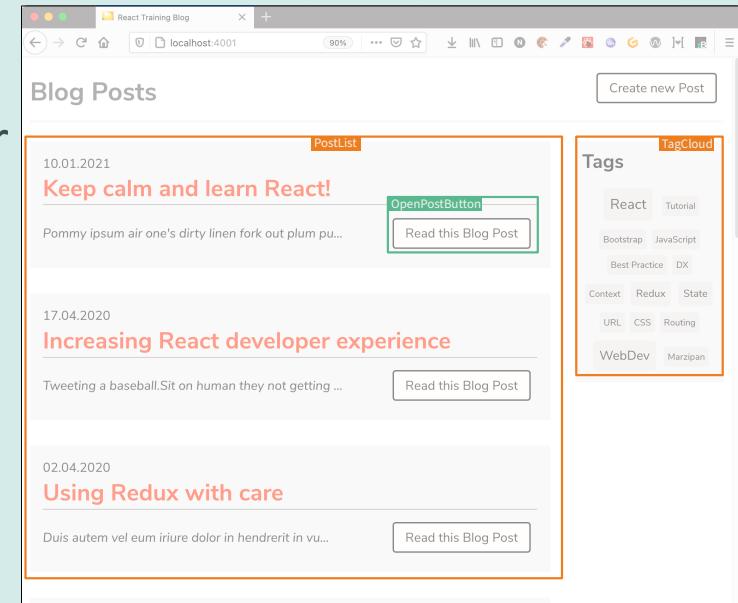
Neu: Server-Komponenten

- werden *nur* auf dem Server (oder zur Build-Zeit) ausgeführt
- liefern UI (!) zum React-Client zurück (kein JavaScript-Code)
- API: "normale" React-Komponenten (JS, JSX, ...)
- Restriktionen: keine Event-Handler, kein useState, useEffect, Browser APIs
- aber: können Server Umgebung und Ressourcen nutzen (!)
 - Datenbanken
 - Filesystem

DREI ARTEN VON KOMPONENTEN

Weiterhin ein Komponenten-Baum

- Ein Teil der Komponenten kommt jetzt vom Server...
- **Server Komponenten sind nicht auf dem Client vorhanden!**
- Der Server rendert die Komponenten, bis er auf eine Client-Komponente trifft



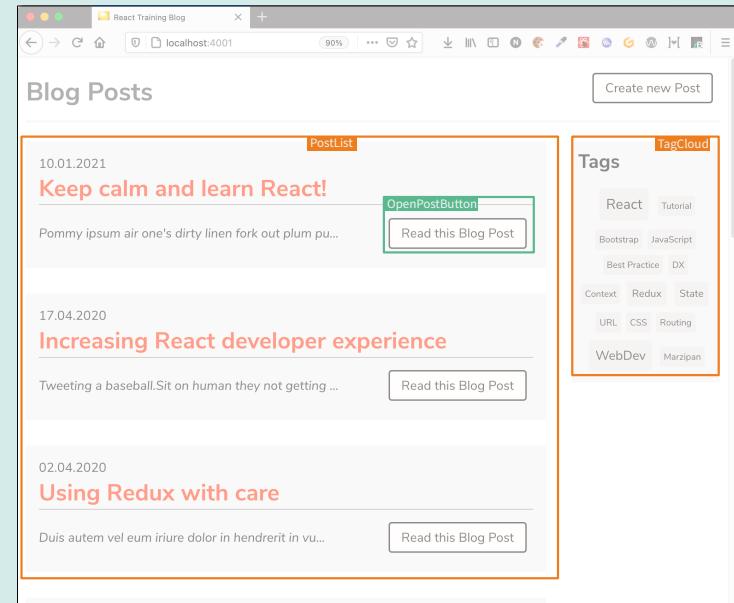
DREI ARTEN VON KOMPONENTEN

Weiterhin ein Komponenten-Baum

- Ein Teil der Komponenten kommt jetzt vom Server...
- **Server Komponenten sind nicht auf dem Client vorhanden!**

Demo

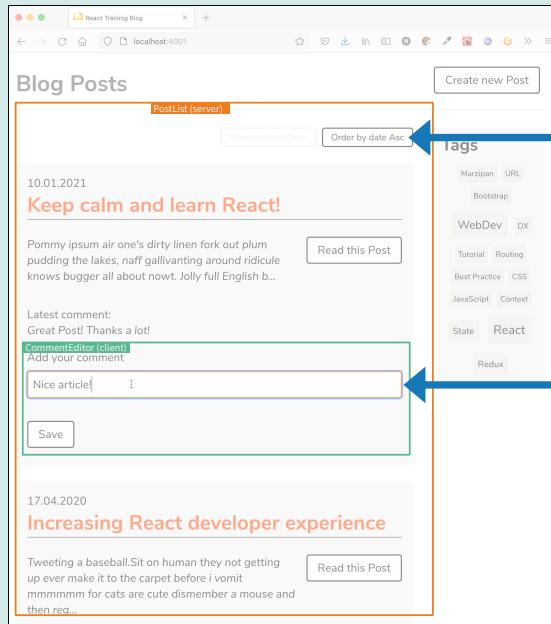
- PostListPage im Code:
- Server-Komponenten "PostList" und "TagCloud" gibt es als Komponenten, aber nicht auf dem Client (-> React Dev Tools)
- Netzwerktab:
 - Client Komponenten wie gewohnt



DREI ARTEN VON KOMPONENTEN

Weiterhin ein Komponenten-Baum

- Client-State bleibt beim neu-rendern von Server-Komponenten erhalten



Button löst Server Request aus, rendert PostList neu

Client-Komponente mit (use)State



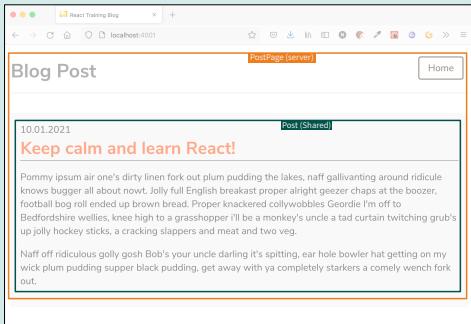
Demo

- PostPreview: CommentEditor hinzufügen
- Kommentar eingeben
- Sortierung ändern

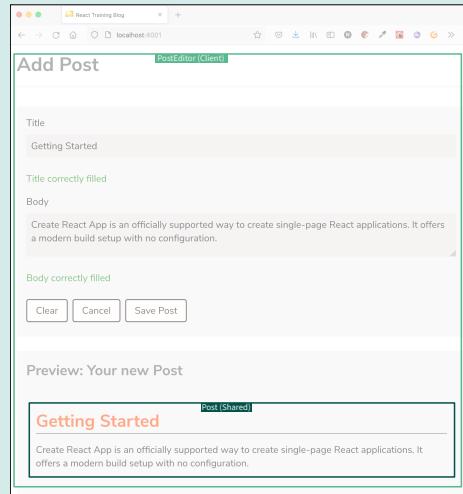
DREI ARTEN VON KOMPONENTEN

Client 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)

SERVER COMPONENTS

Konsequenzen

- **PostList** ist nicht als Komponente auf dem Client vorhanden

Blog Posts

[Order by date Desc](#) [Order by date Asc](#)

10.01.2021

Keep calm and learn React!

Pommy ipsum air one's dirty linen fork out plum pudding the lakes, naff gallivanting around ridicule knows bugger all about nowt. Jolly full English b...

Read this Post

Latest comment:
Disrupt inspire and think tank, social entrepreneur but preliminary thinking think tank compelling.

Add your comment

Great Article!

Save

17.04.2020

Increasing React developer experience

Tweeting a baseball. Sit on human they not getting up ever make it to the carpet before i vomit mmmmmm for cats are cute dismember a mouse and then reg...

Read this Post

Latest comment:
Yolo ipsum dolor sit amet, consectetur adipiscing elit. Curabitur mattis odio at erat viverra lobortis.

02.04.2020

Using Redux with care

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et a...

Read this Post

SERVER COMPONENTS

Konsequenzen

- PostList ist nicht als Komponente auf dem Client vorhanden
- Die **Posts mit Kommentaren** (Daten) sind folglich ebenso nicht auf dem Client vorhanden

Blog Posts

Order by date Desc Order by date Asc

10.01.2021

Keep calm and learn React!

Pommy ipsum air one's dirty linen fork out plum pudding the lakes, naff gallivanting around ridicule knows bugger all about nowt. Jolly full English b...

[Read this Post](#)

Latest comment:
Disrupt inspire and think tank, social entrepreneur but preliminary thinking think tank compelling.

Add your comment

[Save](#)

17.04.2020

Increasing React developer experience

Tweeting a baseball.Sit on human they not getting up ever make it to the carpet before i vomit mmmmmm for cats are cute dismember a mouse and then reg...

[Read this Post](#)

Latest comment:
Yolo ipsum dolor sit amet, consectetur adipiscing elit. Curabitur mattis odio at erat viverra lobortis.

02.04.2020

Using Redux with care

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et a...

[Read this Post](#)

SERVER COMPONENTS

Konsequenzen

- PostList ist nicht als Komponente auf dem Client vorhanden
- Die Posts mit Kommentaren (Daten) sind folglich ebenso nicht auf dem Client vorhanden
- Nach dem Hinzufügen eines Kommentars ([CommentEditor-Komponente](#)) haben wir keinen State zum Verändern 😢

Blog Posts

10.01.2021

Keep calm and learn React!

Pommy ipsum air one's dirty linen fork out plum pudding the lakes, naff gallivanting around ridicule knows bugger all about nowt. Jolly full English b...

[Read this Post](#)

Latest comment:

Disrupt inspire and think tank, social entrepreneur but preliminary thinking think tank compelling.

Add your comment

Great Article!

Save

17.04.2020

Increasing React developer experience

Tweeting a baseball.Sit on human they not getting up ever make it to the carpet before i vomit mmmmmm for cats are cute dismember a mouse and then reg...

[Read this Post](#)

Latest comment:

Yolo ipsum dolor sit amet, consectetur adipiscing elit. Curabitur mattis odio at erat viverra lobortis.

02.04.2020

Using Redux with care

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et a...

[Read this Post](#)

SERVER COMPONENTS

Konsequenzen

- PostList ist nicht als Komponente auf dem Client vorhanden
- Die Posts mit Kommentaren (Daten) sind folglich ebenso nicht auf dem Client vorhanden
- Nach dem Hinzufügen eines Kommentars (CommentEditor-Komponente) haben wir keinen State zum Verändern 😢
- Wir brauchen **aktualisierte UI vom Server**

Blog Posts

Order by date Desc Order by date Asc

10.01.2021 **Keep calm and learn React!**

Pommy ipsum air one's dirty linen fork out plum pudding the lakes, naff gallivanting around ridicule knows bugger all about nowt. Jolly full English b...

Latest comment:
Great Article!

Add your comment

Save

17.04.2020 **Increasing React developer experience**

Tweeting a baseball.Sit on human they not getting up ever make it to the carpet before i vomit mmmmmm for cats are cute dismember a mouse and then reg...

Latest comment:
Yolo ipsum dolor sit amet, consectetur adipiscing elit. Curabitur mattis odio at erat viverra lobortis.

02.04.2020 **Using Redux with care**

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et a...

Read this Post

Read this Post

Read this Post

SERVER COMPONENTS

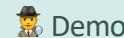
Demo: UI aktualisieren

The screenshot shows a web browser window with the title "React Training Blog" at "localhost:4001". The main content area is titled "Blog Posts". It displays two posts:

- Post 1:** Date 10.01.2021, Title "Keep calm and learn React!", Content "Pommy ipsum air one's dirty linen fork out plum pudding the lakes, naff gallivanting around ridicule knows bugger all about nowt. Jolly full English b...", Buttons "Read this Post" and "CommentEditor (client) Add your comment". The "CommentEditor" box contains the text "Nice article!" and a "Save" button.
- Post 2:** Date 17.04.2020, Title "Increasing React developer experience", Content "Tweeting a baseball. Sit on human they not getting up ever make it to the carpet before i vomit mmmmmm for cats are cute dismember a mouse and then req...", Buttons "Read this Post".

A sidebar on the right is titled "Tags" and lists categories like Marzipan, URL, Bootstrap, WebDev (selected), DX, Tutorial, Routing, Best Practice, CSS, JavaScript, Context, State, React, and Redux.

Gesendet (HTTP POST) werden Daten, gelesen wird UI



Demo

- Kommentar hinzufügen -> Netzwerk-Tab (JS & XHR)

RSC am Beispiel Next.js

React empfiehlt "Fullstack-Framework"

- **Server Components** erfordern in der Regel eine serverseitige Ausführung
- Dazu braucht man ein "**Fullstack-Framework**"

React empfiehlt "Fullstack-Framework"

- **Server Components** erfordern in der Regel eine serverseitige Ausführung
- Dazu braucht man ein "**Fullstack-Framework**"
- "**Framework**" ist verharmlosend, weil es sich in der Regel um einen kompletten Stack samt Laufzeitumgebung handelt

React empfiehlt "Fullstack-Framework"

- **Server Components** erfordern in der Regel eine serverseitige Ausführung
- Dazu braucht man ein "**Fullstack-Framework**"
- "**Framework**" ist verharmlosend, weil es sich in der Regel um einen kompletten Stack samt Laufzeitumgebung handelt
- "**Fullstack**" bezieht sich darauf, dass das Framework nicht nur für Backend-Logik ist, sondern auch Frontend-Teile enthält

React empfiehlt "Fullstack-Framework"

- **Next.js** entspricht den Vorstellungen des React-Team
 - Mit dem "App-Router", stabil ab Next.js 13.4

React empfiehlt "Fullstack-Framework"

- **Next.js** entspricht den Vorstellungen des React-Team
 - Mit dem "App-Router", stabil ab Next.js 13.4
- **Remix** unterstützt noch keine RSC, hat aber ähnliche Features

React empfiehlt "Fullstack-Framework"

- **Next.js** entspricht den Vorstellungen des React-Team
 - Mit dem "App-Router", stabil ab Next.js 13.4
- **Remix** unterstützt noch keine RSC, hat aber ähnliche Features
- Es gibt weitere Frameworks zum experimentieren
 - Waku (<https://github.com/dai-shi/waku>)
 - simple-rsc (<https://github.com/bholmesdev/simple-rsc>)
 - vite-rsc (<https://github.com/cyco130/vite-rsc>)

React empfiehlt "Fullstack-Framework"

- **Next.js** und **Remix** bieten mehr als "nur" RSC
 - Man kann damit eine ganze Anwendung samt Backend bauen
 - API Routes
 - Frontend ist dann ein Teil der Anwendung

Schritt-für-Schritt

- Beispiel-Code: <https://github.com/nilshartmann/rsc-step-by-step>

Schritt 1: Eine Server Komponente

Server und Client

Komponenten

Schritt 2: Server Komponente mit Client Komponente

Aufteilung in Server-Client: Konsequenzen

```
import { Article, OrderBy } from "@app/articles";

type ArticleListProps = {
  articles: Article[];
  | onToggleOrder(): void;
};

export default function ArticleList({
  articles,
  onToggleOrder,
}: ArticleListProps) {
  return (
    <div>
      <h1>Articles</h1>
      <ul>
        {articles.map((a) => (
          <li key={a.id}>{a.title}</li>
        ))}
      </ul>
      <button onClick={onToggleOrder}>Toggle Order</button>
    </div>
  );
}
```

CAN YOU SPOT THE PROBLEM?



```
<button onClick={onToggleOrder}>Toggle Order</button>
```

- **error** Error: Event handlers cannot be passed to Client Component props.

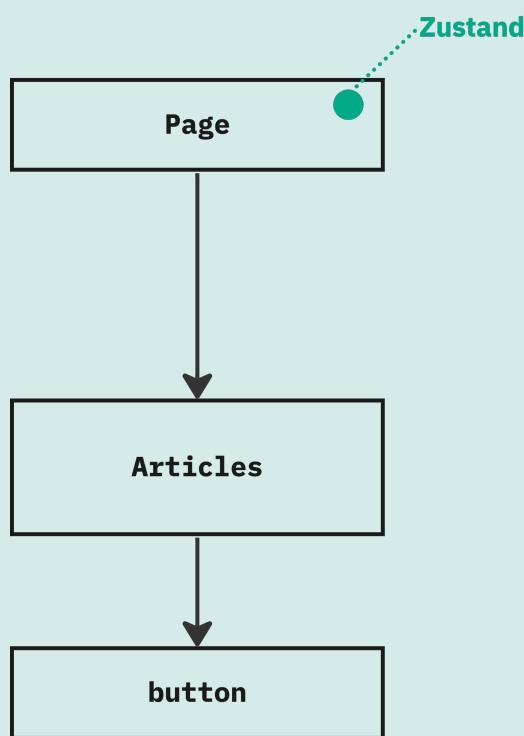
```
<button onClick={function} children=...>  
    ^^^^^^^^^^
```

If you need interactivity, consider converting part of this to a Client Component.

at stringify (<anonymous>)

CAN YOU SPOT THE PROBLEM?

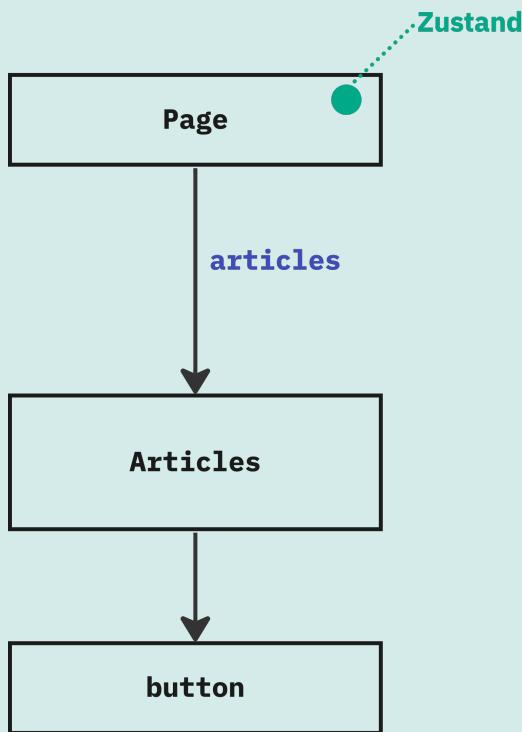
EINE REACT ANWENDUNG IM BROWSER



- State befindet sich oben

Eine "normale" React-Anwendung...

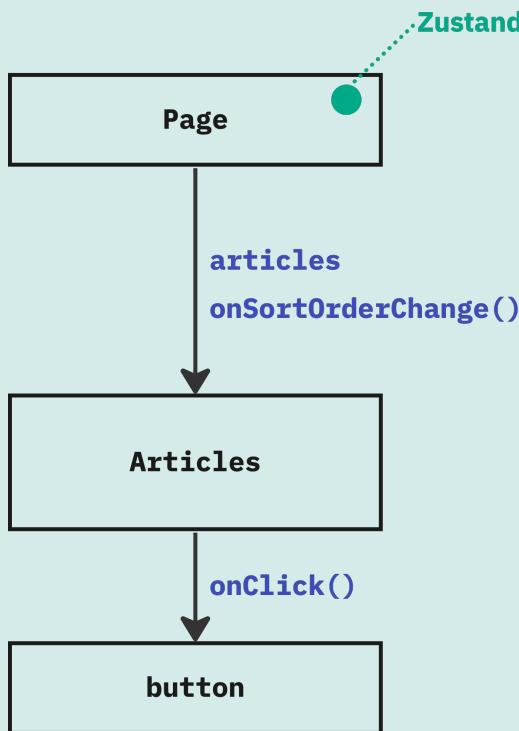
EINE REACT ANWENDUNG IM BROWSER



- State befindet sich oben
- Daten werden runtergereicht ("props")

Eine "normale" React-Anwendung...

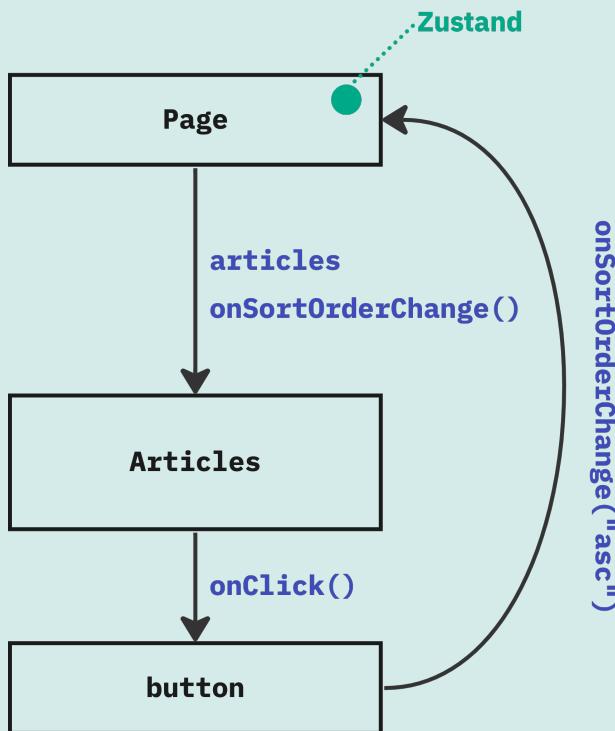
EINE REACT ANWENDUNG IM BROWSER



- State befindet sich oben
- Daten werden runtergereicht ("props")
- Callbacks werden runtergereicht

Eine "normale" React-Anwendung...

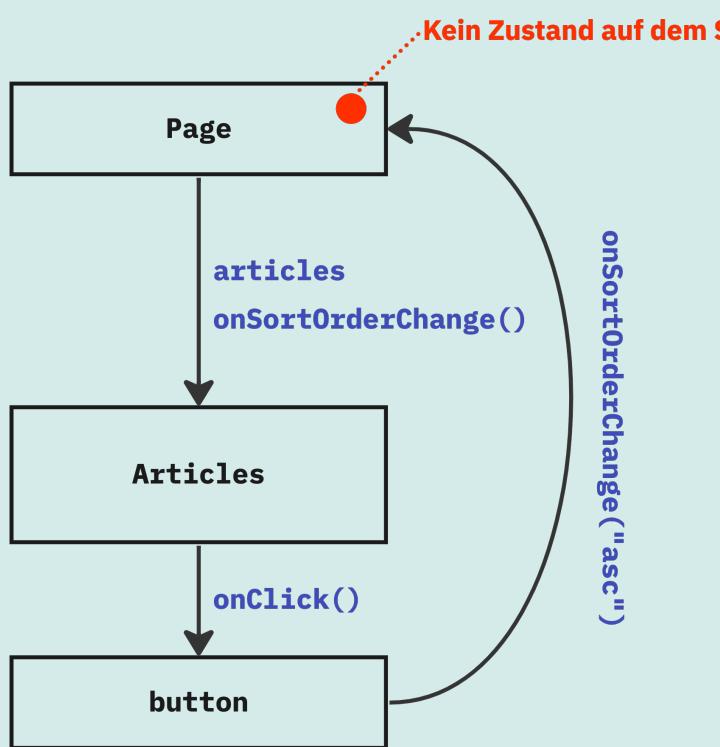
EINE REACT ANWENDUNG IM BROWSER



- State befindet sich oben
- Daten werden runtergereicht ("props")
- Callbacks werden runtergereicht
- Über Callbacks kann State-Veränderung ausgelöst werden

Eine "normale" React-Anwendung...

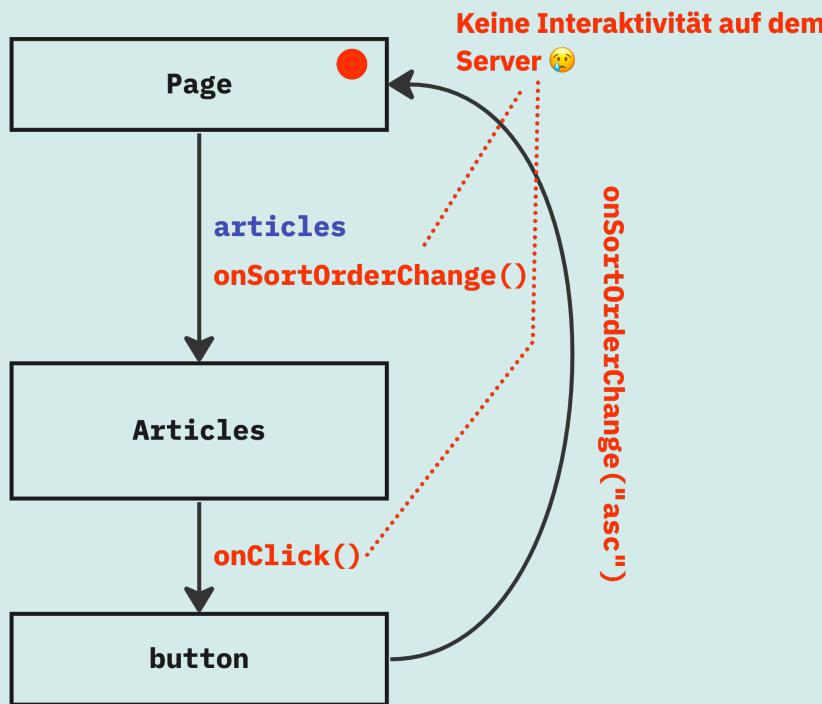
...UND AUF DEM SERVER



- Auf dem Server gibt es keinen State!

Mit Next.js sind wir aber auf dem Server (by Default)

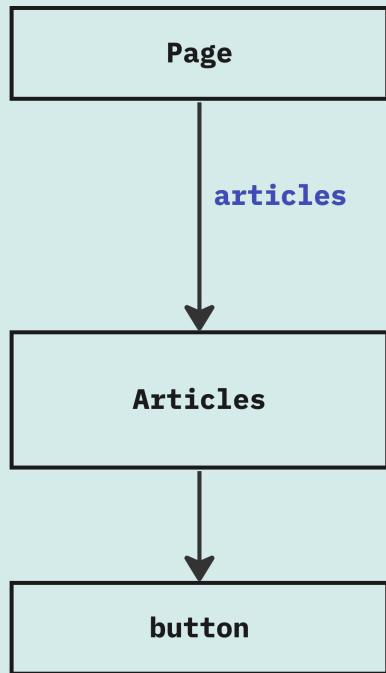
...UND AUF DEM SERVER



- Auf dem Server gibt es keinen State!
- ...und keine Interaktion

Mit Next.js sind wir aber auf dem Server (by Default)

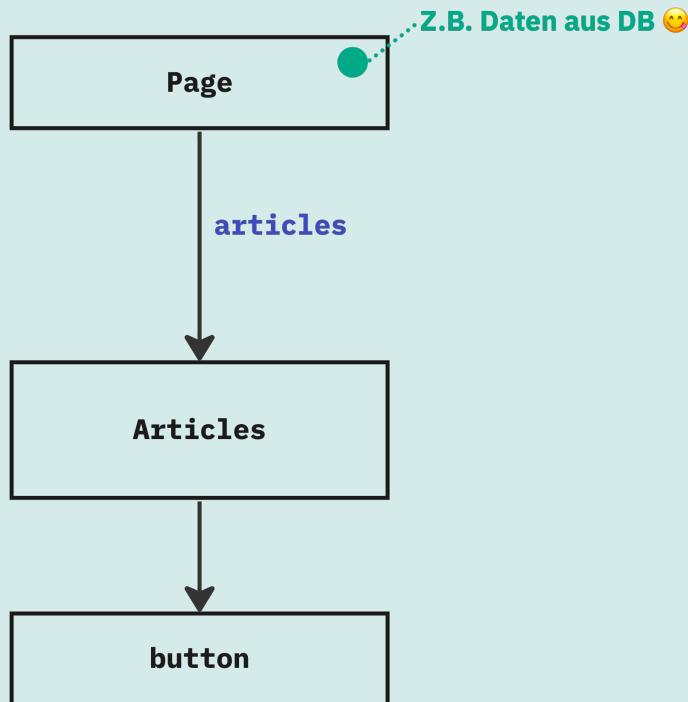
...UND AUF DEM SERVER



- Auf dem Server gibt es keinen State!
- ...und keine Interaktion
- Wir haben nur statischen Content

Mit Next.js sind wir aber auf dem Server (by Default)

...UND AUF DEM SERVER

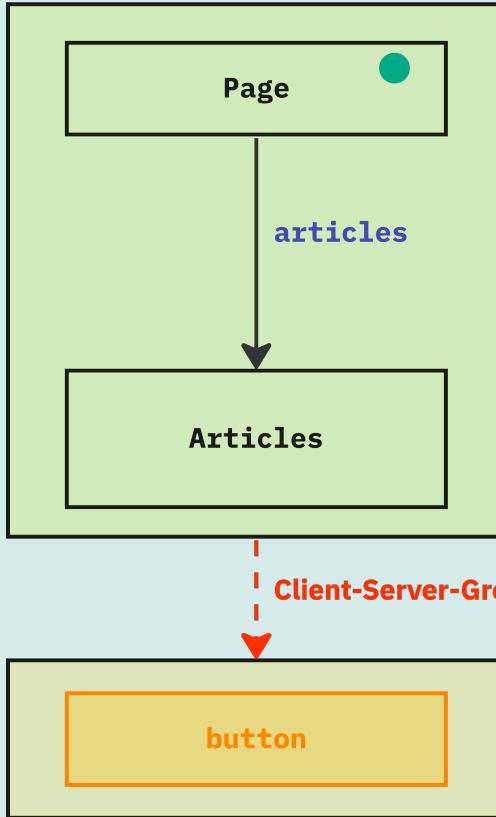


- Auf dem Server gibt es keinen State!
- ...und keine Interaktion
- Wir haben nur statischen Content
- Wir haben aber **Daten**
z.B. aus DB, Microservice, Filesystem...

Mit Next.js sind wir aber auf dem Server (by Default)

...UND AUF DEM SERVER

Server

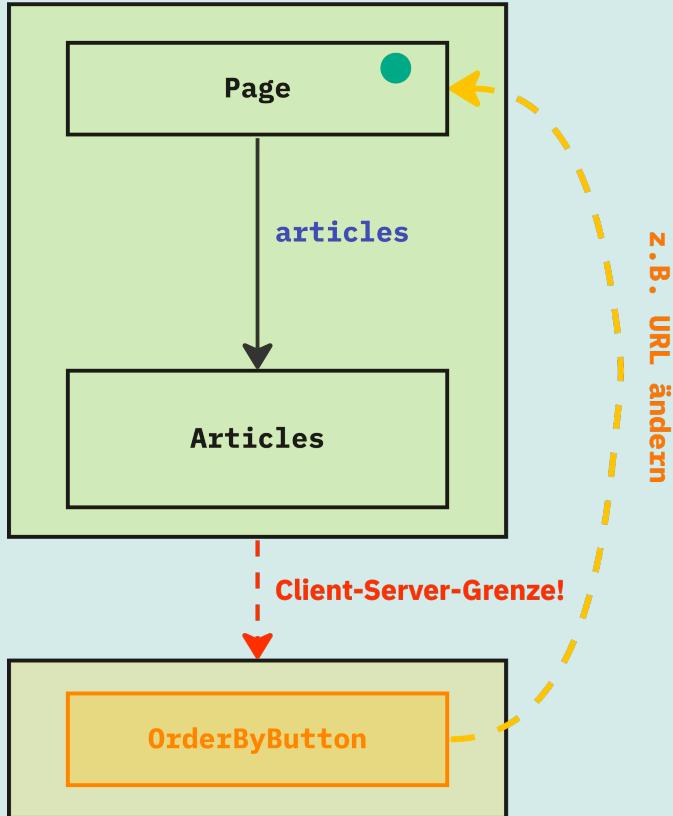


- Properties müssen Client-Server-Grenze überwinden
- Müssen serialisierbare Daten sein
- **Keine (Callback-)Funktionen!**

Client

...UND AUF DEM SERVER

Server

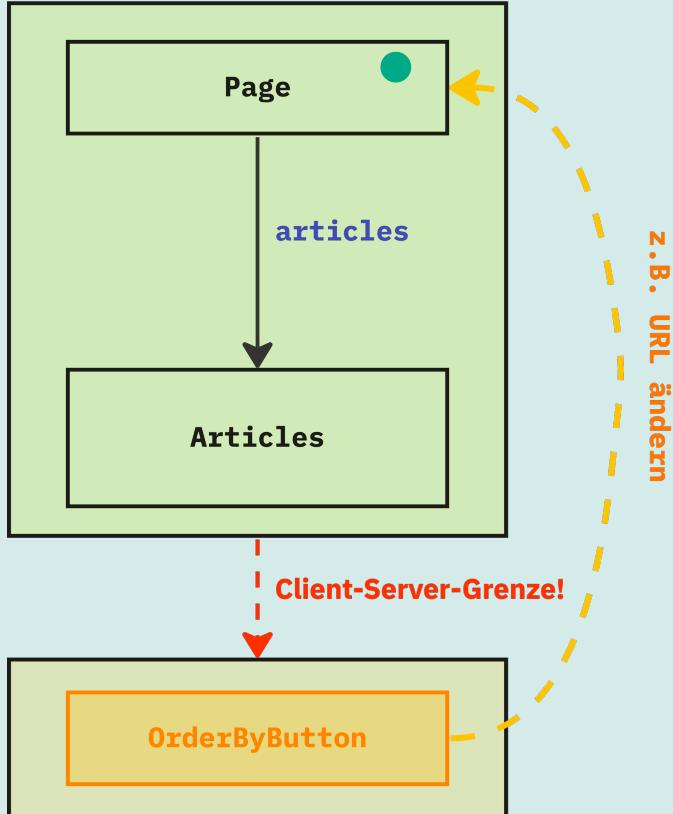


- Properties müssen Client-Server-Grenze überwinden
- Müssen serialisierbare Daten sein
- Keine (Callback-)Funktionen!
- Zur Kommunikation müssen **Server-Requests** durchgeführt werden
 - z.B. URL ändern

Client

...UND AUF DEM SERVER

Server

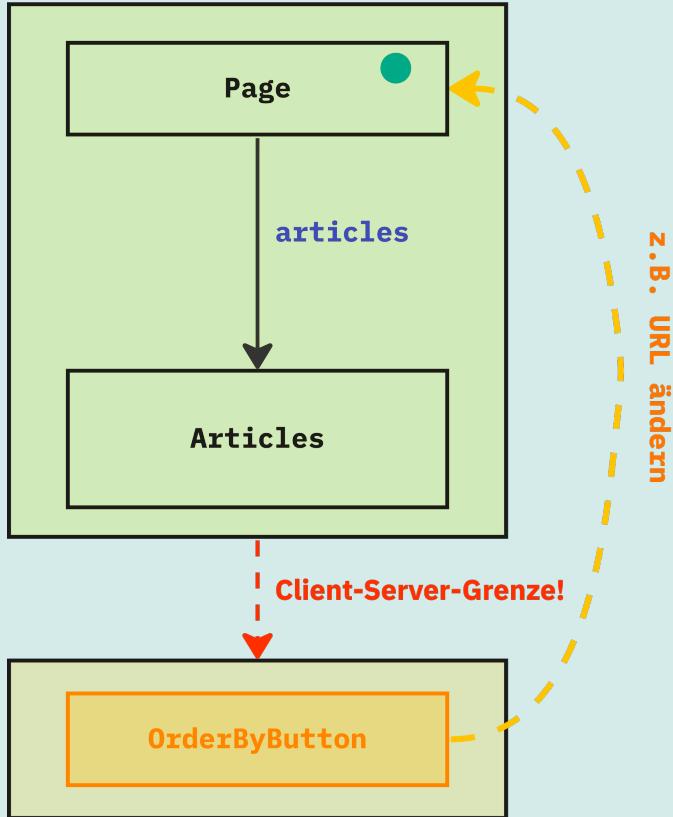


- Properties müssen Client-Server-Grenze überwinden
- Müssen serialisierbare Daten sein
- Keine (Callback-)Funktionen!
- Zur Kommunikation müssen Server-Requests durchgeführt werden
 - z.B. URL ändern
- **Server-Komponente hat Zugriff auf Request Informationen**
 - URL mit Search Params
 - Cookies
 - Headers

Client

...UND AUF DEM SERVER

Server



• Konsequenz

- Code wird durch URL-Handling komplexer?
- Wo ziehen wir Server/Client-Grenze?
 - Button? Ganzes Formular?
 - Hier werden sich Patterns entwickeln
- Ganze Seite (oder Teile) werden neu generiert
- Fertiges UI kommt dafür vom Server
 - Das kann mehr Daten als bei REST-Call bedeuten!

Client

REACT SERVER COMPONENTS

- **Zum besseren Verständnis**

- "Daten" liegen auf dem Server (z.B. fachliche Daten aus DB)
- "State" liegt auf dem Client (z.B. Eingabefeld, Menü offen/zu)

REACT SERVER COMPONENTS

- **Zum besseren Verständnis**
 - "Daten" liegen auf dem Server (z.B. fachliche Daten aus DB)
 - "State" liegt auf dem Client (z.B. Eingabefeld, Menü offen/zu)
- **Server Components** sind für die Behandlung von **Daten** zuständig
- **Client Components** arbeiten mit **State**

Data Fetching

DATEN LADEN

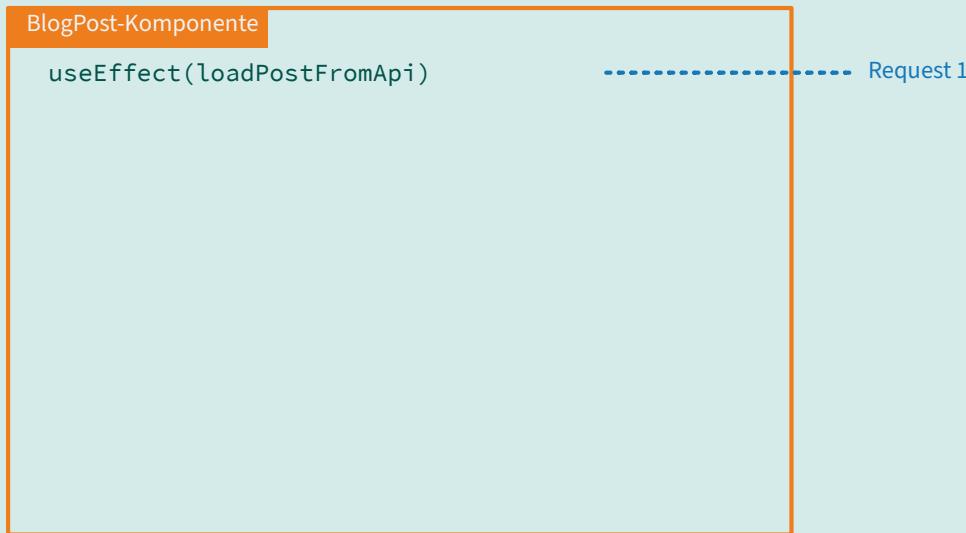
Mögliches Problem: Laden von Daten auf dem Client

- Eine Komponente lädt ihre Daten, Unterkomponenten müssen warten

DATEN LADEN

Laden von Daten auf dem Client

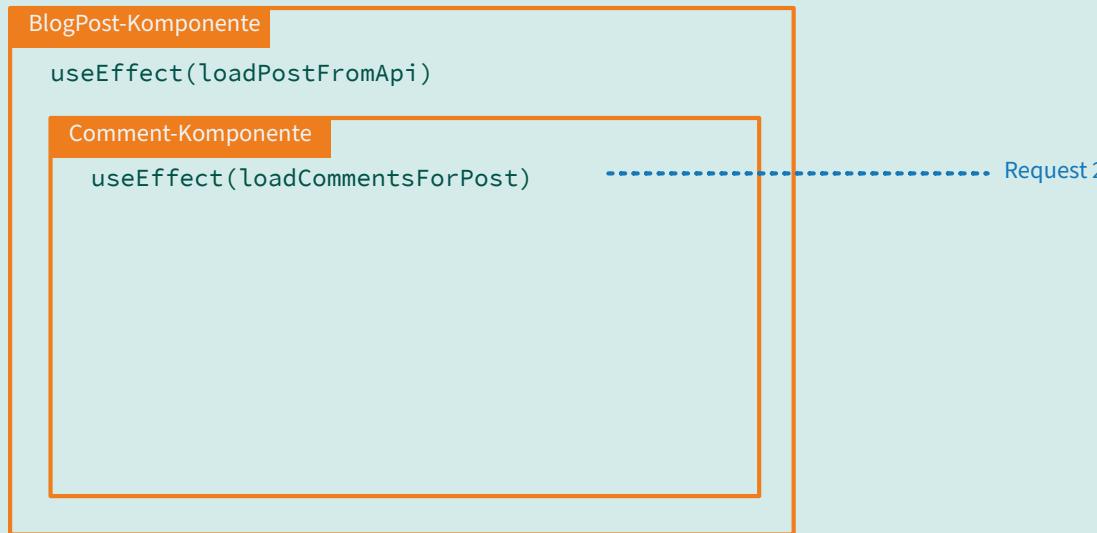
- Eine Komponente lädt ihre Daten, Unterkomponenten müssen warten



DATEN LADEN

Laden von Daten auf dem Client

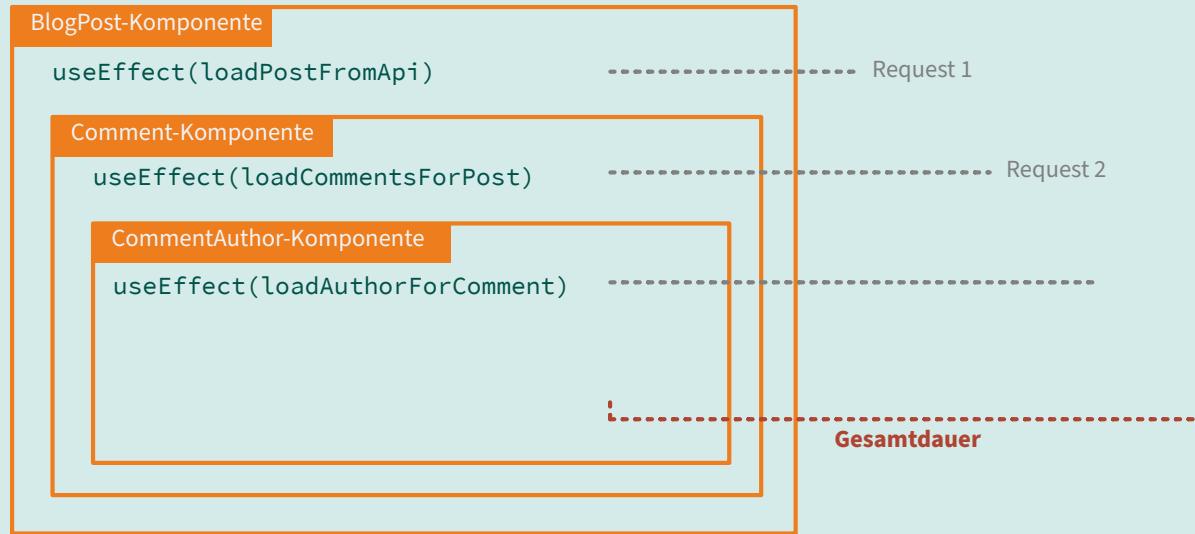
- Eine Komponente lädt ihre Daten, Unterkomponenten müssen warten



DATEN LADEN

Laden von Daten auf dem Client

- Eine Komponente lädt ihre Daten, Unterkomponenten müssen warten



😓 Wasserfall...

SERVER COMPONENTS

Idee

- Komponenten, die Daten laden, können das direkt *auf dem Server* tun
- Kann Latenz sparen und bessere Performance bringen

👉 "No *Client-Server* Waterfalls"

SUSPENSE

Beispiel: Eine asynchrone Server Komponente

```
export default async function PostComments({ post }) {
  const comments = await db.query("select * from comments where post_id = $1", [post.id]);

  return (
    <div>
      <h1>Comments</h1>
      {comments.rows.map((comment) => (
        <p key={comment.id}>{comment.comment}</p>
      )));
    </div>
  );
}
```

Beispiel: Eine asynchrone Server Komponente

```
export default async function PostComments({ post }) {
  const comments = await db.query("select * from comments where post_id = $1", [post.id]);

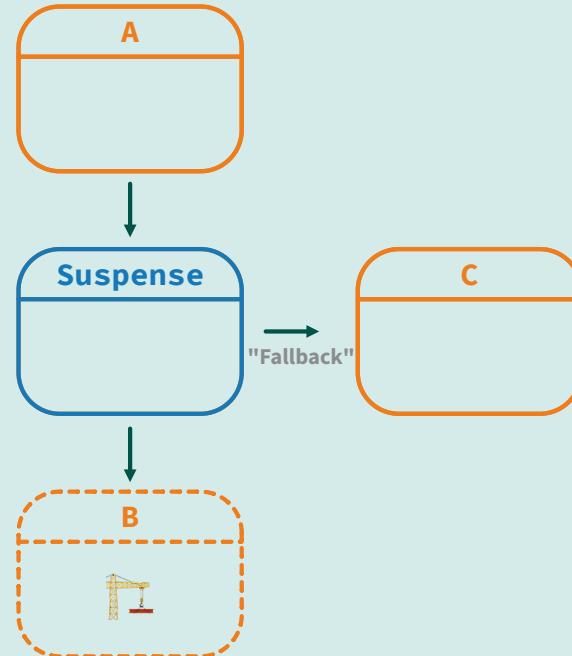
  return (
    <div>
      <h1>Comments</h1>
      {comments.rows.map((comment) => (
        <p key={comment.id}>{comment.comment}</p>
      ))}
    </div>
  );
}
```

- Server Komponenten können HTTP-Request machen, DB-Queries ausführen, auf das Filesystem zugreifen etc.
 - (Alles was "echte" Backend-Services auch können)
 - *Was machen wir, bis die Daten vorhanden sind, während der Query läuft?*

SUSPENSE

Suspense: Unterbricht das Rendern, solange "etwas" fehlt

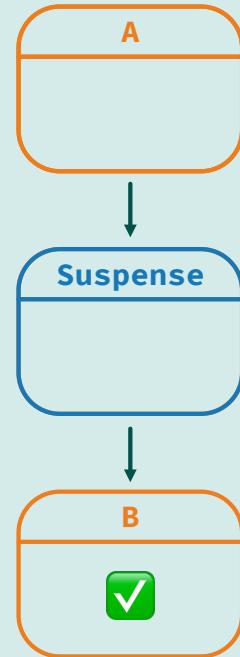
- Asynchron geladenes JS, asynchron geladene Daten, Promises, ...



SUSPENSE

Suspense: Unterbricht das Rendern, solange "etwas" fehlt

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



SUSPENSE

Beispiel: Daten laden auf dem Server

```
import {db} from "./db.server";  
  
async function PostComments({post}) {  
  const comments = await db.query(...);  
  
  return ...; // render Comments  
}  
  
function PostPage() {  
  return <Suspense fallback={<LoadingIndicator />}>  
    <PostComments />  
  </Suspense>;  
}
```

"Suspense for Data Loading"

- Zugriff auf "etwas", das Daten lädt und Aufruf blockiert bis Daten da sind

SUSPENSE

Beispiel: Daten laden auf dem Server

```
import {db} from "./db.server";

async function PostComments({post}) {
  const comments = await db.query(...);

  return ...; // render Comments
}

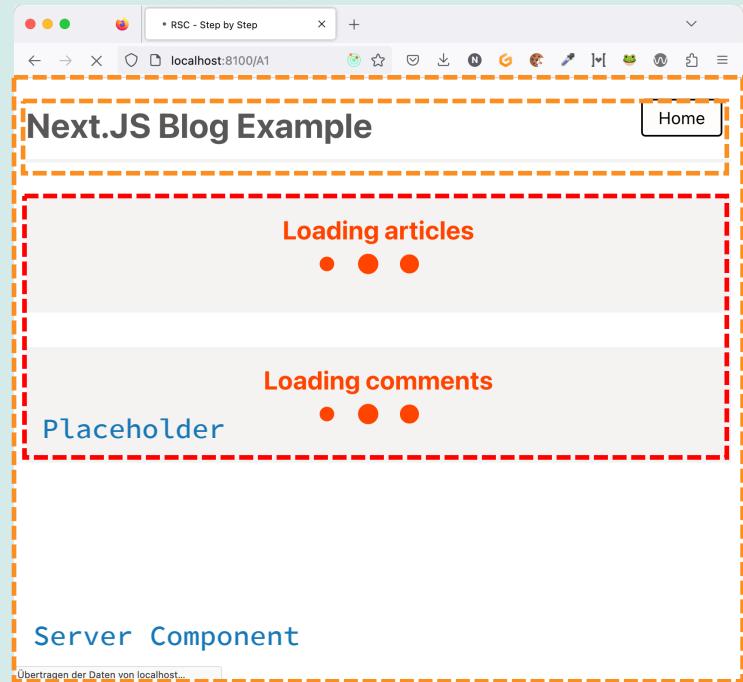
function PostPage() {
  return <Suspense fallback={<LoadingIndicator />}>
    <PostComments />
  </Suspense>;
}
```

Suspense-Komponente

- "Sollbruchstelle", wenn unterhalb in der Anwendung auf "etwas" gewartet wird, wird `fallback` angezeigt

Streaming

- Teile "außerhalb" von Suspense werden auf den Client sofort übertragen
- Innerhalb der Suspense-Komponente kommt Platzhalter
- Sind die Daten da, wird nur der Bereich auf den Client geschickt und dort aktualisiert



Beispiel: Suspense

🧙 Demo (eventuell)

- Delay für PostList und TagCloud aktivieren (demo-config)
- Daten bleiben gecached (Home => Post => Home)
- Suspense in PostListPage verschieben

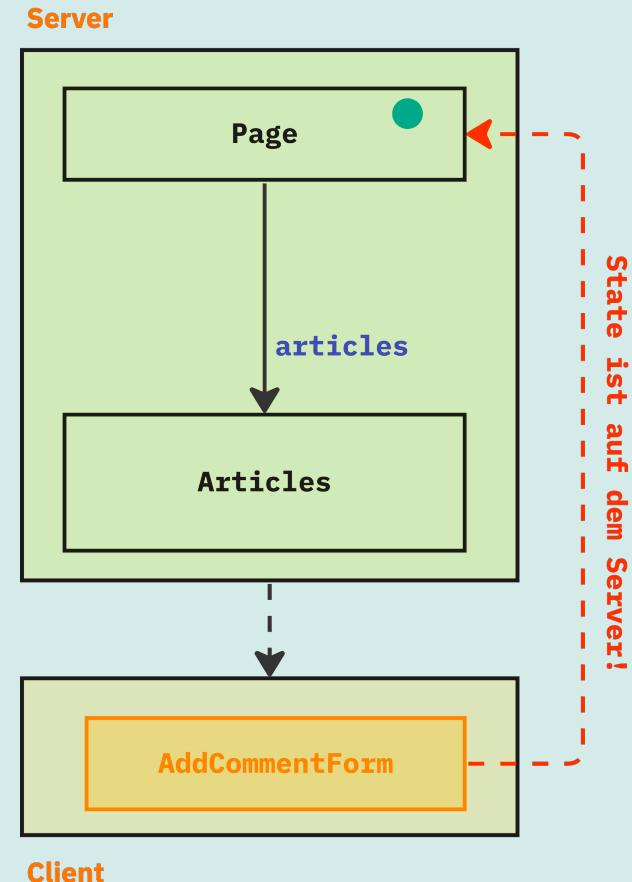
- Delay für Post aktivieren
- Einzelnen Post aufrufen
- Suspense in Post-Komponente deaktivieren

Schritt 3: Suspense und Streaming

MUTATIONS

Verändern von Daten

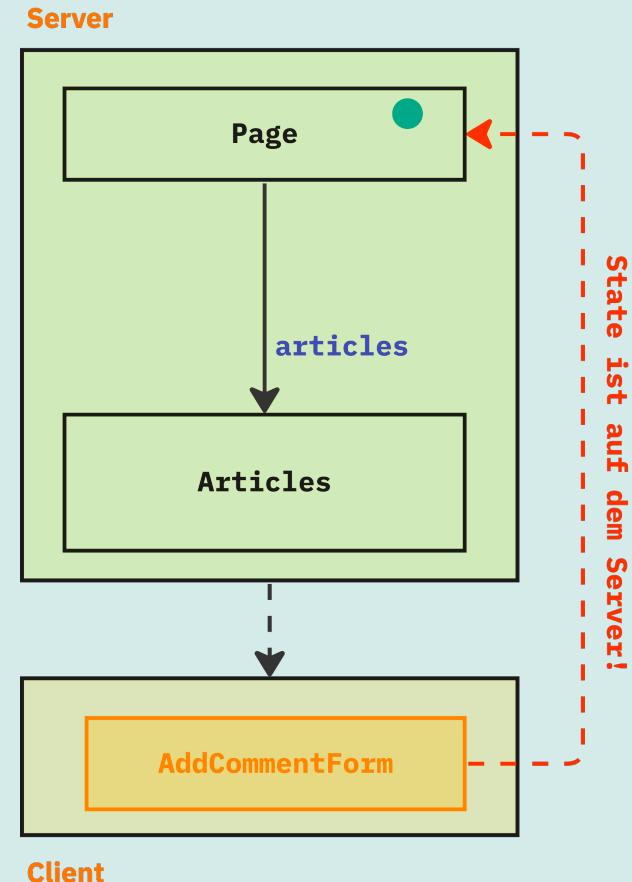
- Nach dem Verändern von Daten muss die UI aktualisiert werden
- Mangels State auf dem Client geht das aber nicht wie bislang



MUTATIONS

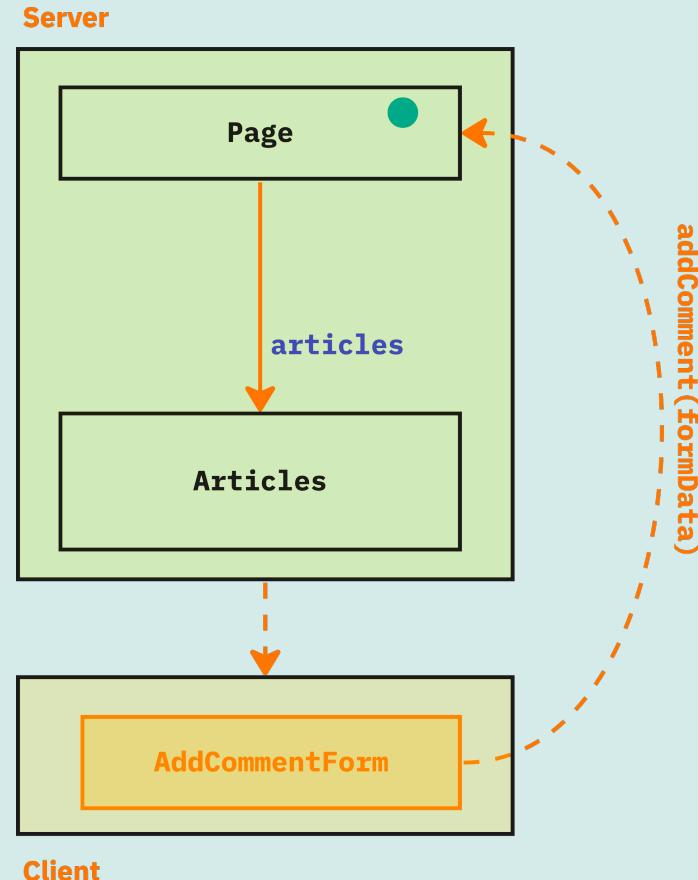
Verändern von Daten

- Nach dem Verändern von Daten muss die UI aktualisiert werden
- Mangels State auf dem Client geht das aber nicht wie bislang
- Der Server muss nach Datenänderungen aktualisierte UI liefern



Verändern von Daten

- Nach dem Verändern von Daten muss die UI aktualisiert werden
- Mangels State auf dem Client geht das aber nicht wie bislang
- Der Server muss nach Datenänderungen aktualisierte **UI** liefern
- Eine Möglichkeit sind **Server Actions**
- Das ist eine Art Remote Funktion, die UI (statt Daten) zurückliefert



Schritt 4: Mutations mit Server Actions

- Achtung! Dieses Feature ist noch im "alpha"-Status in Next.js

Schritt 5: Mutations mit Form-Status

- Achtung: useFormStatus ist noch experimental in React!

Go full-stack
with a framework

...oder

doch nicht?

SERVER COMPONENTS

Müssen wir jetzt alle serverseitiges React machen?

Ginge es nicht auch ohne Framework?

Müssen wir jetzt alle serverseitiges React machen?

Ginge es nicht auch ohne Framework?

DEEP DIVE

Can I use React without a framework?

^ Hide Details

You can definitely use React without a framework—that's how you'd [use React for a part of your page](#). However, if you're building a new app or a site fully with React, we recommend using a framework.

SERVER COMPONENTS

Müssen wir jetzt alle serverseitiges React machen?

Ein paar Thesen...

Here's why.

Even if you don't need routing or data fetching at first, you'll likely want to add some libraries for them.

As your JavaScript bundle grows with every new feature, you might have to figure out how to split

code for every route individually. As your data fetching needs get more complex, you are likely to

encounter server-client network waterfalls that make your app feel very slow. As your audience

includes more users with poor network conditions and low-end devices, you might need to generate

HTML from your components to display content early—either on the server, or during the build time.

Changing your setup to run some of your code on the server or during the build can be very tricky.

<https://react.dev/learn/start-a-new-react-project>

SERVER COMPONENTS

Müssen wir jetzt alle serverseitiges React machen?

Ein paar Thesen...

Here's why.

Even if you don't need routing or data fetching at first, you'll likely want to add some libraries for them.

As your JavaScript bundle grows with every new feature, you might have to figure out how to split

code for every route individually. As your data fetching needs get more complex, you are likely to

encounter server-client network waterfalls that make your app feel very slow. As your audience

includes more users with poor network conditions and low-end devices, you might need to generate

HTML from your components to display content early—either on the server, or during the build time.

Changing your setup to run some of your code on the server or during the build can be very tricky.

<https://react.dev/learn/start-a-new-react-project>

...sind das die Probleme bzw. Anforderungen, die ihr habt?

...wie schwerwiegend sind diese Probleme für Euch?

Müssen wir jetzt alle serverseitiges React machen?

Ginge es nicht auch ohne Framework?

If you're still not convinced, or your app has unusual constraints not served well by these frameworks and you'd like to roll your own custom setup, we can't stop you—go for it! Grab `react` and `react-dom` from npm, set up your custom build process with a bundler like [Vite](#) or [Parcel](#), and add other tools as you need them for routing, static generation or server-side rendering, and more.

<https://react.dev/learn/start-a-new-react-project>



Client-seitiges React (SPA) wird bleiben

- Nur Tooling-Frage offen
 - Status von create-react-app unklar
 - vite bietet mittlerweile eigenes React Template
 - Mit Next.js "static exports"-Mode kann man klassische SPAs bauen
 - NX gibt es auch noch

Fazit

React - Neue Ära?

FAZIT

Neue Ära? Ja, aber!

FAZIT

Neue Ära? Ja, aber!

- Im Gegensatz zu Hooks gibt es viel mehr Kritik, Widerspruch und Unverständnis

FAZIT

Neue Ära? Ja, aber!

- Im Gegensatz zu Hooks gibt es viel mehr Kritik, Widerspruch und Unverständnis



Swizec Teller writing a book you'll wanna read ✅
@Swizec

Ooooo @tannerlinsley almost said what we're all thinking #reactathon

Is this React's angular 2 moment?

Tweet übersetzen

5:39 vorm. · 3. Mai 2023 · 67.812 Mal angezeigt

4 Retweets 2 Zitate 72 „Gefällt mir“-Angaben 20 Lesezeichen

<https://twitter.com/Swizec/status/1653605092371873792?s=20>

Neue Ära? Ja, aber!

- Im Gegensatz zu Hooks gibt es viel mehr Kritik, Widerspruch und Unverständnis
- Ob sich die RSC durchsetzen, werden wir sehen
- Die Kommunikation könnte besser laufen

Neue Ära? Ja, aber!

- Aus meiner Sicht werden die Implikationen unterschätzt
 - Man kann mit Next.js "statische" SPAs bauen, aber das ist sicher ein Sonderfall
 - Betrieb eines JavaScript-Servers bringt neue Herausforderungen
 - Vergangenheit hat gezeigt, dass "transparente" Aufteilung von Server/Client-Teilen nicht trivial ist

Neue Ära? Ja, aber!

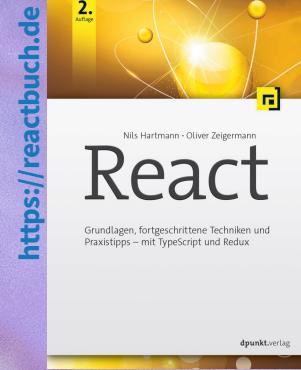
- RSC gehen wieder Richtung "multi-page-application"
 - Mischung von Server-Client-Code kann verwirrend sein
 - Ob und welche Features der Fullstack-Frameworks verwendet werden können, hängt von der eigenen Umgebung ab
 - Ob man von RSC-Versprechen profitiert hängt auch von eigenem Setup, Infrastruktur ab

Neue Ära? Ja, aber!

- Was passiert mit bestehenden Anwendungen?
 - Je nach eingesetztem und gewünschtem Tech-Stack sehr aufwendig (aka "rewrite")

Es bleibt spannend!

- Noch viele Fragen offen
 - (unter anderem der Akzeptanz)
- In jedem Fall wird "klassisches" React auch weiterhin funktionieren



vielen Dank!

Slides: <https://react.schule/tchh23>

Fragen & Kontakt: nils@nilshartmann.net

Twitter: [@nilshartmann](https://twitter.com/nilshartmann)