

**NILS HARTMANN**

<https://nilshartmann.net>

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

# React 2023

Beginn einer

neuen

Ära?

# NILS HARTMANN

nils@nilshartmann.net

Freiberuflicher Entwickler, Architekt, Trainer aus Hamburg

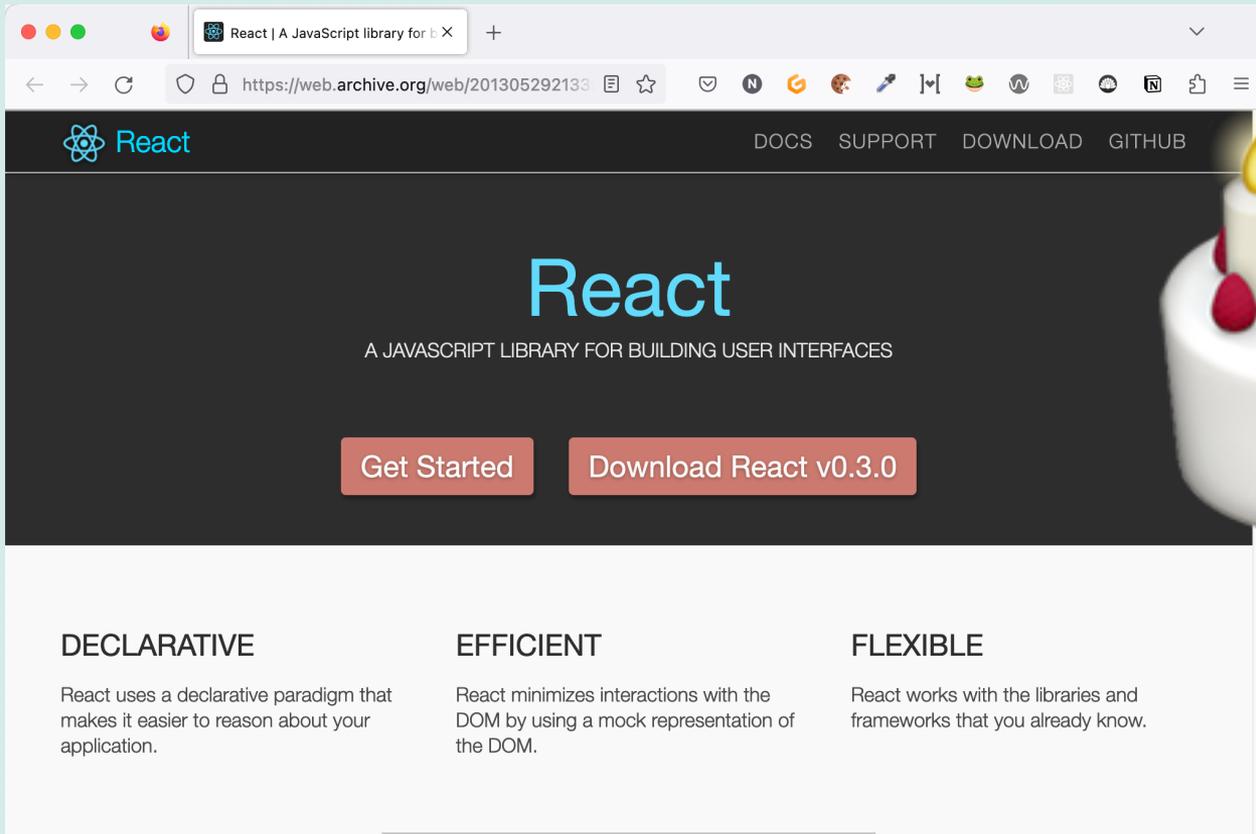
Java, Spring, GraphQL, React, TypeScript



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

<https://reactbuch.de>

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

A screenshot of a web browser displaying the React website. The browser's address bar shows the URL: https://web.archive.org/web/201305292133. The page features the React logo and navigation links (DOCS, SUPPORT, DOWNLOAD, GITHUB). The main heading is "React" in a large blue font, followed by the subtitle "A JAVASCRIPT LIBRARY FOR BUILDING USER INTERFACES". Below this are two buttons: "Get Started" and "Download React v0.3.0". The page is divided into three columns, each with a heading and a short paragraph: "DECLARATIVE", "EFFICIENT", and "FLEXIBLE".

React | A JavaScript library for building user interfaces

DOCS SUPPORT DOWNLOAD GITHUB

# React

A JAVASCRIPT LIBRARY FOR BUILDING USER INTERFACES

[Get Started](#) [Download React v0.3.0](#)

## 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!

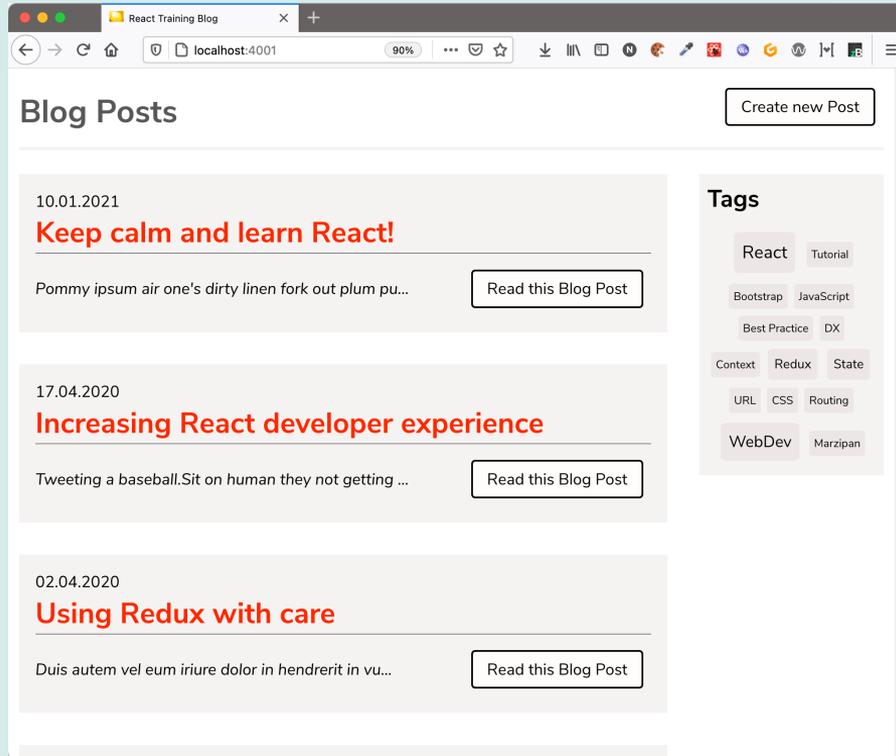
<https://legacy.reactjs.org/blog/2019/02/06/react-v16.8.0.html>

# 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/>

...UND HEUTE ?!



EIN BEISPIEL...

# EIN BEISPIEL

## Was macht die Beispiel-Anwendung aus?

- Viel statischer Content

The screenshot displays a web application interface for a blog. At the top right, there is a button labeled "Create new Post". Below this, there are two sorting options: "Order by date Desc" and "Order by date Asc". The main content area features two blog posts, each enclosed in a green border. The first post is dated "10.01.2021" and has the title "Keep calm and learn React!". Its content is a block of placeholder text, and it includes a "Read this Post" button. Below the text is a "Latest comment:" section with another block of placeholder text. The second post is dated "17.04.2020" and has the title "Increasing React developer experience". It also contains placeholder text and a "Read this Post" button, followed by a "Latest comment:" section with more placeholder text. On the right side, there is a "Tags" sidebar with a green border, containing a grid of tag buttons: WebDev, State, JavaScript, Tutorial, Context, DX, Marzipan, React, URL, Redux, Bootstrap, Best Practice, Routing, and CSS.

# EIN BEISPIEL

## Was macht die Beispiel-Anwendung aus?

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

The screenshot shows a web application interface for a blog. At the top, there is a header with the text "Blog Posts" and a "Create new Post" button. Below the header, there are two sorting options: "Order by date Desc" and "Order by date Asc". The main content area displays two blog posts. The first post is dated "10.01.2021" and has the title "Keep calm and learn React!". It includes a snippet of text, a "Read this Post" button, and a "Latest comment" section. The second post is dated "17.04.2020" and has the title "Increasing React developer experience". It also includes a snippet of text, a "Read this Post" button, and a "Latest comment" section. On the right side of the page, there is a "Tags" section with a grid of tag buttons including "WebDev", "State", "JavaScript", "Tutorial", "Context", "DX", "Marzipan", "React", "URL", "Redu", "Bootstrap", "Best Practice", "outing", and "CSS".

Annotations with red arrows point to various parts of the application:

- An arrow points from the text "React!" to the "React" tag in the tag cloud.
- An arrow points from the text "tag-cloud.js" to the "React" tag in the tag cloud.
- An arrow points from the text "MomentJS!" to the date "17.04.2020" of the second post.
- An arrow points from the text "Marked!" to the "Read this Post" button of the second post.
- An arrow points from the text "React!" to the "Order by date Desc" button.

# 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)

The screenshot displays a web application interface for a blog. At the top right, there is a button labeled "Create new Post" enclosed in a green rectangular box. Below this, the main content area is titled "Blog Posts". It features two sorting buttons: "Order by date Desc" and "Order by date Asc".

The first blog post is dated "10.01.2021" and has the title "Keep calm and learn React!". The post body contains placeholder text: "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...". A "Read this Post" button is located to the right of the text. Below the post body, a "Latest comment:" section shows a snippet of text: "Disrupt inspire and think tank, social entrepreneur but preliminary thinking think tank compelling."

The second blog post is dated "17.04.2020" and has the title "Increasing React developer experience". The post body contains placeholder text: "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 located to the right of the text. Below the post body, a "Latest comment:" section shows a snippet of text: "Yolo ipsum dolor sit amet, consectetur adipiscing elit. Curabitur mattis odio at erat viverra lobortis."

On the right side of the interface, there is a "Tags" section displaying a grid of tag buttons: "WebDev", "State", "JavaScript", "Tutorial", "Context", "DX", "Marzipan", "React", "URL", "Redux", "Bootstrap", "Best Practice", "Routing", and "CSS".

### 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!

### 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>

21. 12. 2020

## 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*

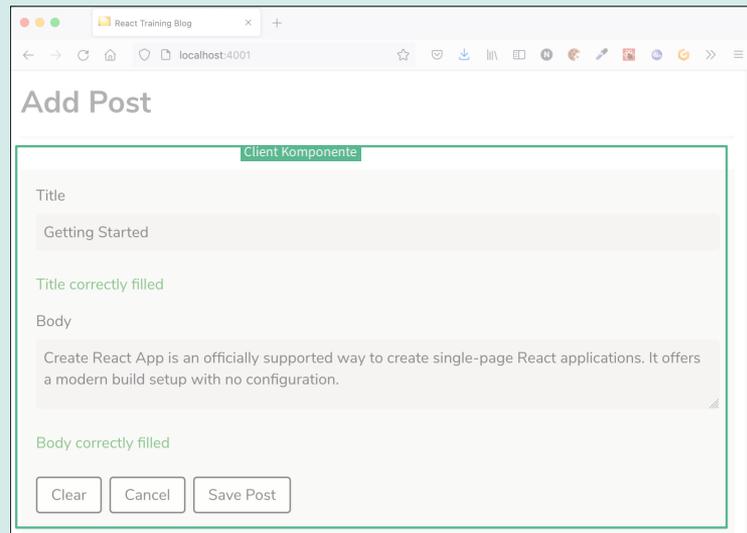
👉 "Zero-Bundle-Size"

## 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



### 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, ...)

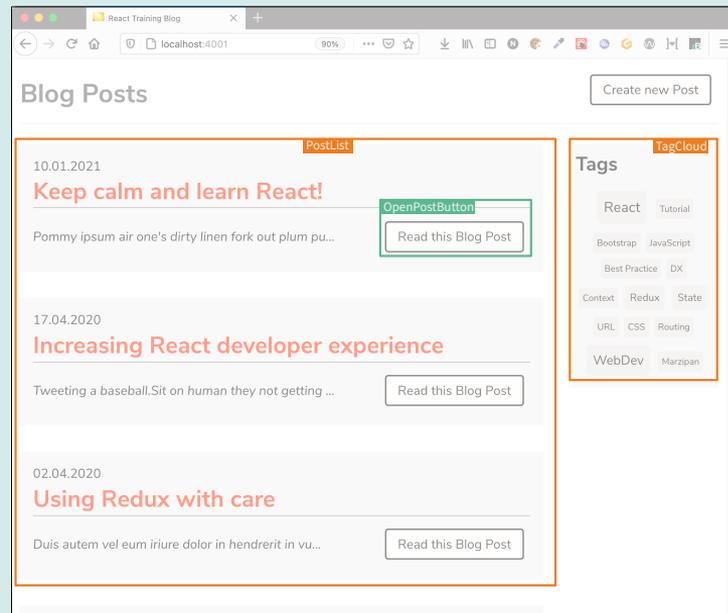
### Neu: Server-Komponenten

- 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

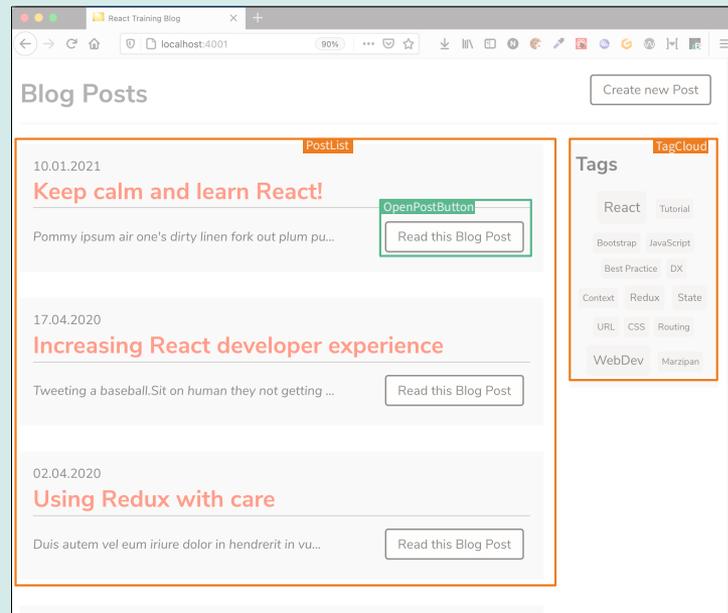
## Weiterhin ein Komponenten-Baum

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



### Demo

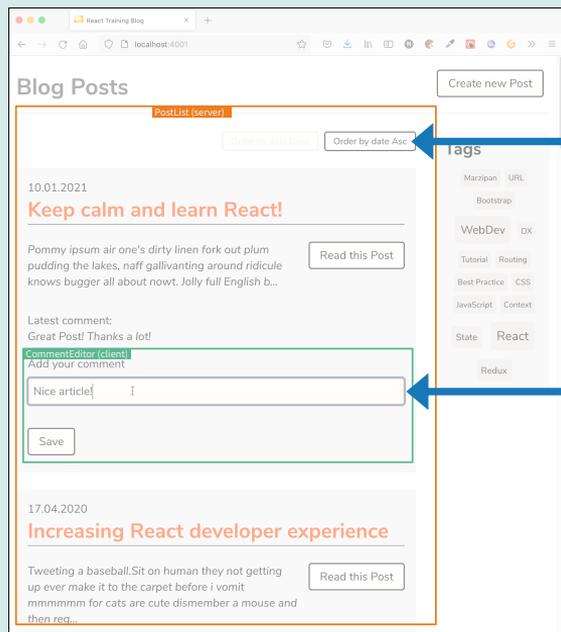
- PostListPage im Code:
- Server-Komponenten "PostList" und "TagCloud" gibt es als Komponenten, aber nicht auf dem Client (-> React Dev Tools)
- Netzwerktab:
  - Client Komponenten wie gewohnt



# 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



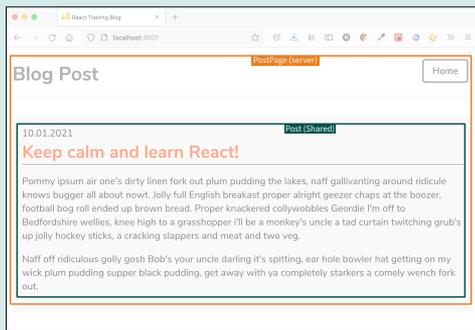
Demo

- **PostPreview:** CommentEditor hinzufügen
- Kommentar eingeben
- Sortierung ändern

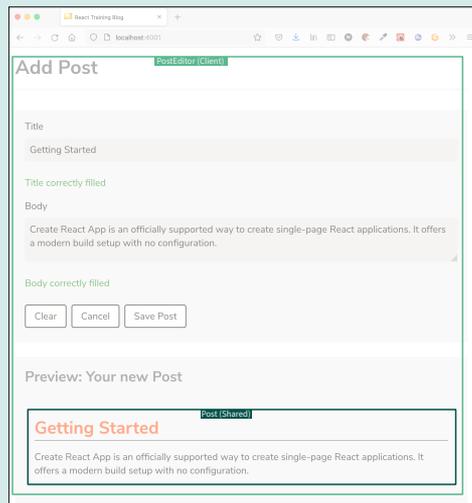
# DREI ARTEN VON KOMponentEN

## 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 mmmmm 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

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 mmmmm for cats are cute disembear 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

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 mmmmm for cats are cute disembear 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...

Read this Post

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 mmmmm 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

## Demo: UI aktualisieren

The screenshot shows a web browser window with the title "React Training Blog" and the address bar displaying "localhost:4001". The main content area is titled "Blog Posts" and contains two blog entries. The first entry, dated "10.01.2021", has the title "Keep calm and learn React!" and a "Read this Post" button. Below the entry is a comment form titled "CommentEditor (client)" with the text "Add your comment". The form contains a text input field with the text "Nice article!" and a "Save" button. The second entry, dated "17.04.2020", has the title "Increasing React developer experience" and a "Read this Post" button. A blue arrow points from the text "Gesendet (HTTP POST) werden Daten, gelesen wird UI" to the "Save" button in the comment form.

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/nishartmann/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.

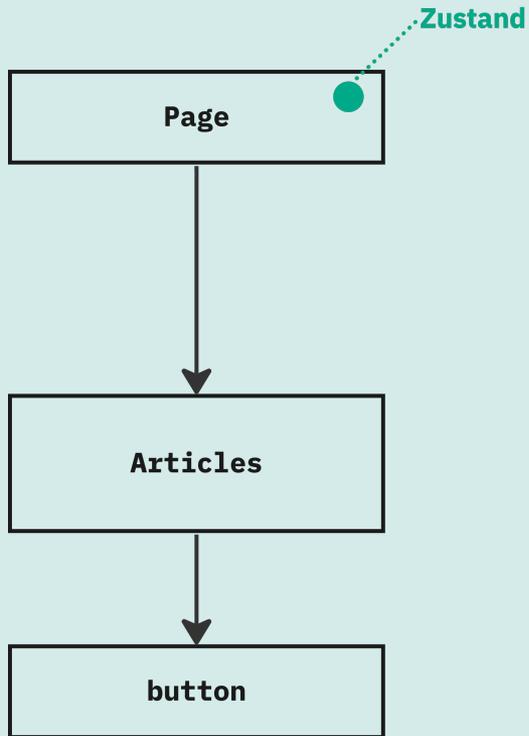
```
<button onClick={function} children=...>  
      ^^^^^^^^^^^
```

If you need interactivity, consider converting part of this to a Client Component.  
at stringify (<anonymous>)

**CAN YOU SPOT THE PROBLEM?**

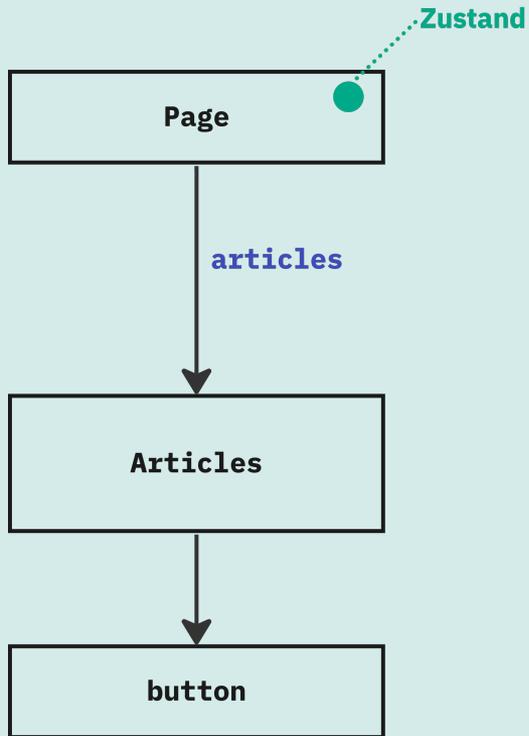
# EINE REACT ANWENDUNG IM BROWSER

- State befindet sich oben



Eine "normale" React-Anwendung...

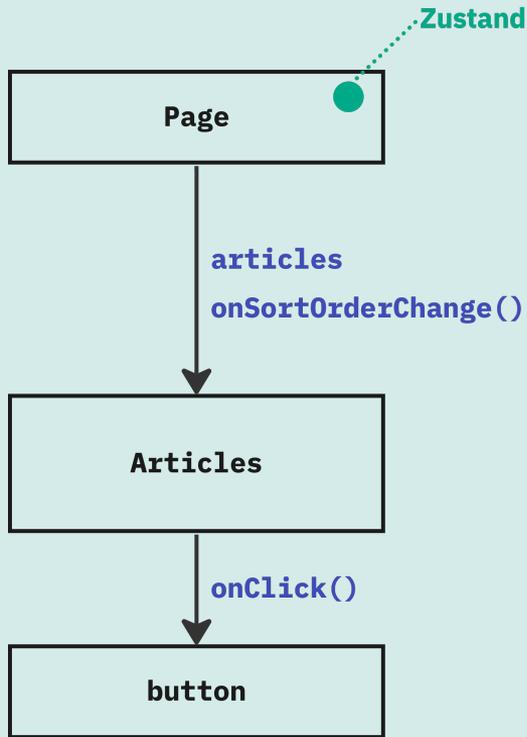
# EINE REACT ANWENDUNG IM BROWSER



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

Eine "normale" React-Anwendung...

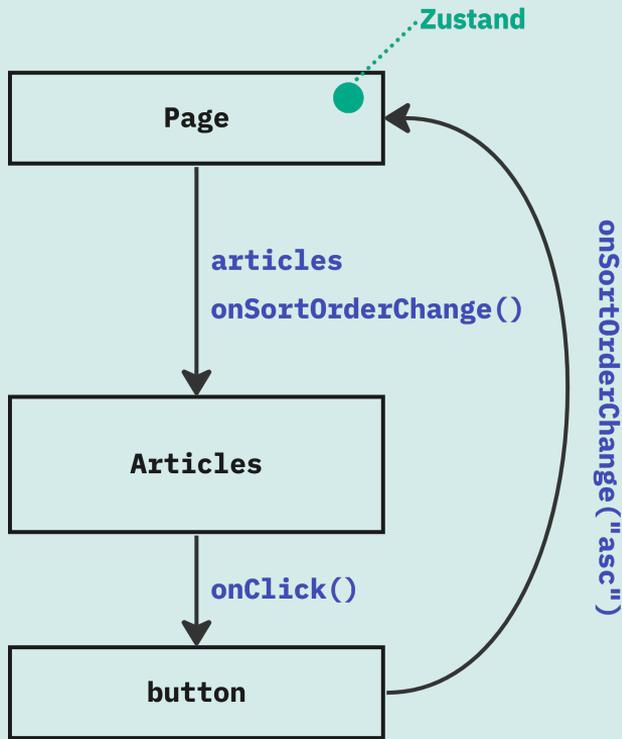
# EINE REACT ANWENDUNG IM BROWSER



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

Eine "normale" React-Anwendung...

# EINE REACT ANWENDUNG IM BROWSER

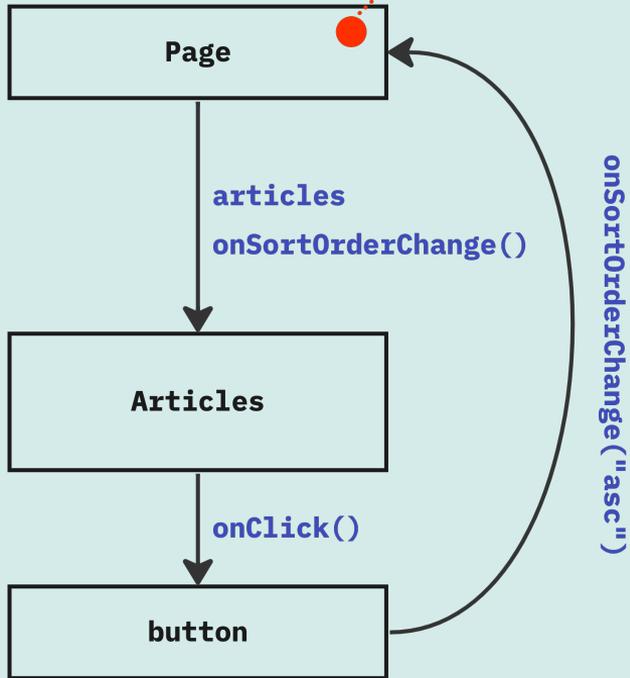


- State befindet sich oben
- Daten werden runtergereicht ("props")
- Callbacks werden runtergereicht
- Über Callbacks kann State-Veränderung ausgelöst werden

Eine "normale" React-Anwendung...

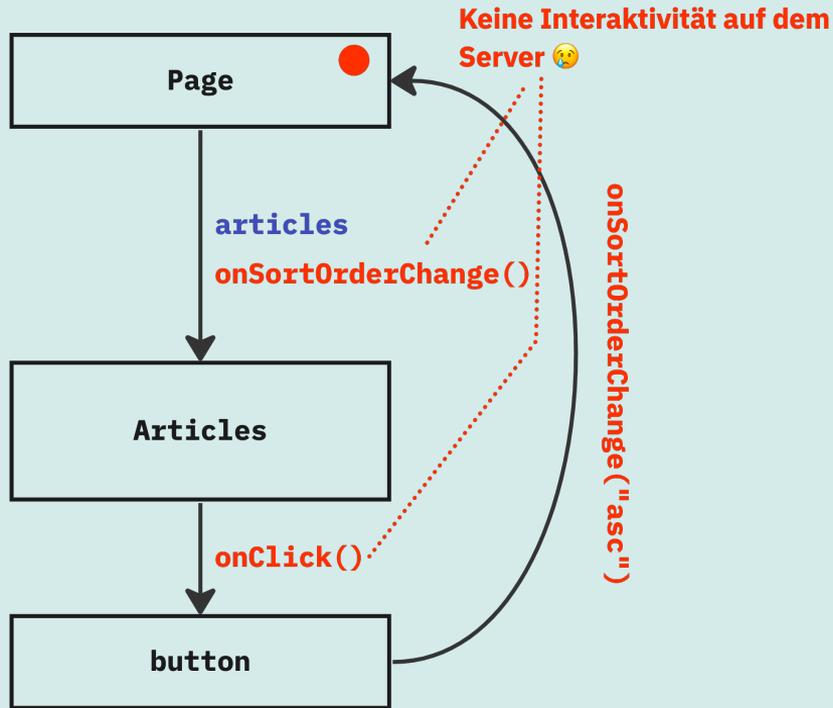
## ...UND AUF DEM SERVER

- Kein Zustand auf dem Server 😞 • Auf dem Server gibt es keinen State!



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

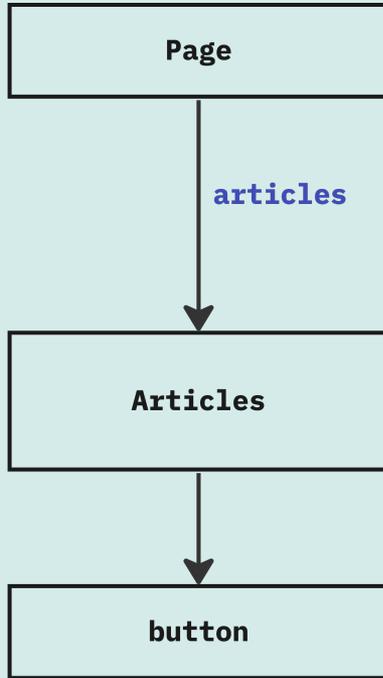
## ...UND AUF DEM SERVER



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

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

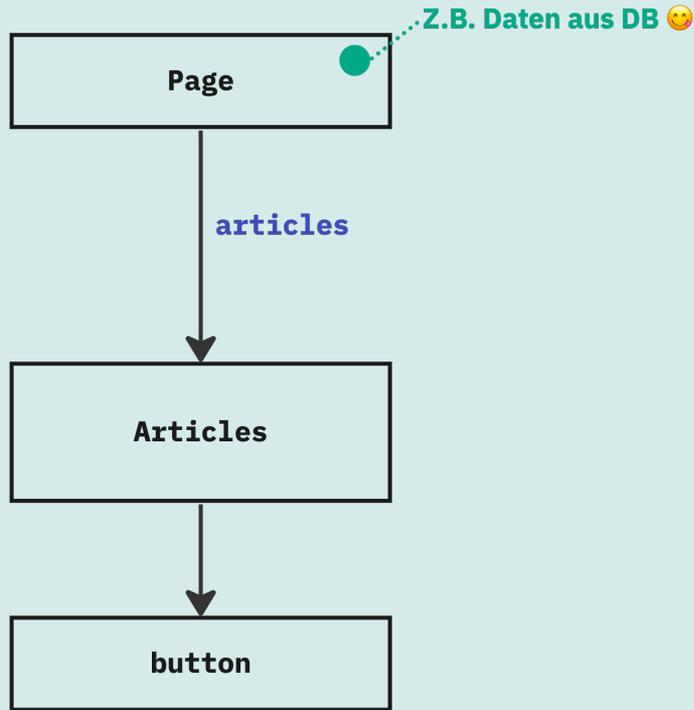
## ...UND AUF DEM SERVER



- 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

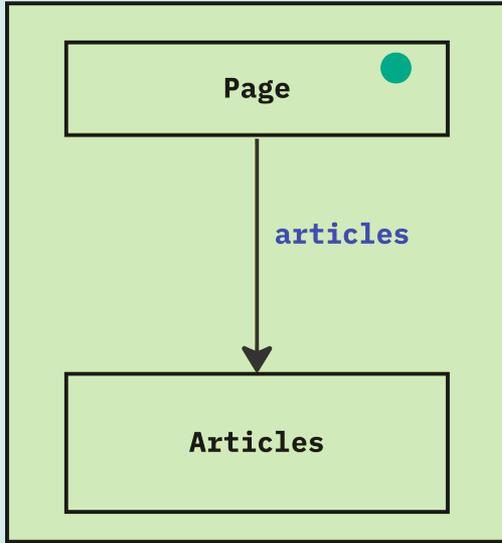


- Auf dem Server gibt es keinen State!
- ...und keine Interaktion
- Wir haben nur statischen Content
- Wir haben aber **Daten**  
z.B. aus DB, Microservice, Filesystem...

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

## ...UND AUF DEM SERVER

Server



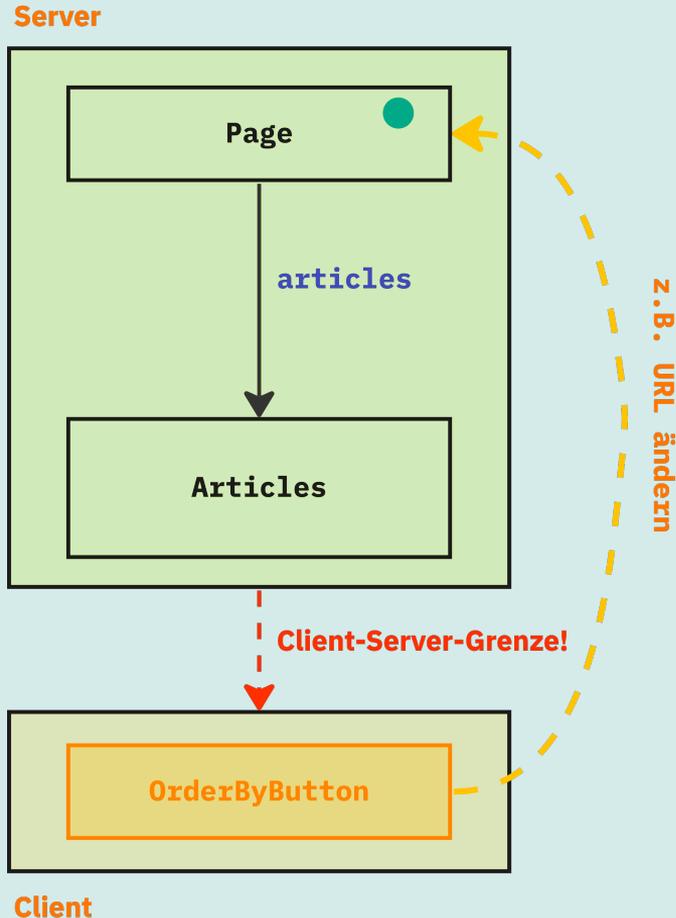
Client-Server-Grenze!



Client

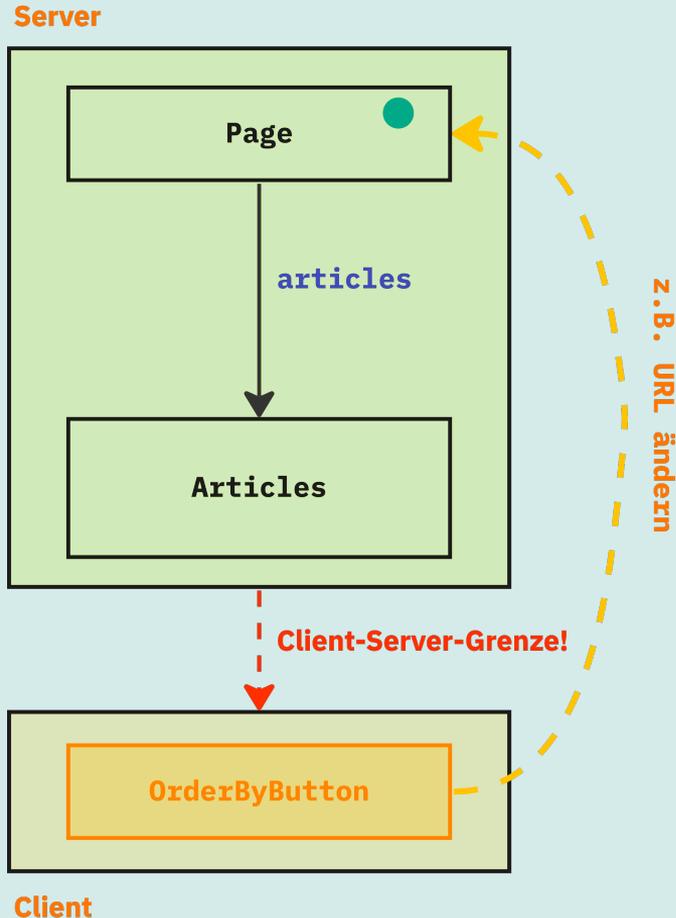
- Properties müssen Client-Server-Grenze überwinden
- Müssen serialisierbare Daten sein
- **Keine (Callback-)Funktionen!**

## ...UND AUF DEM SERVER



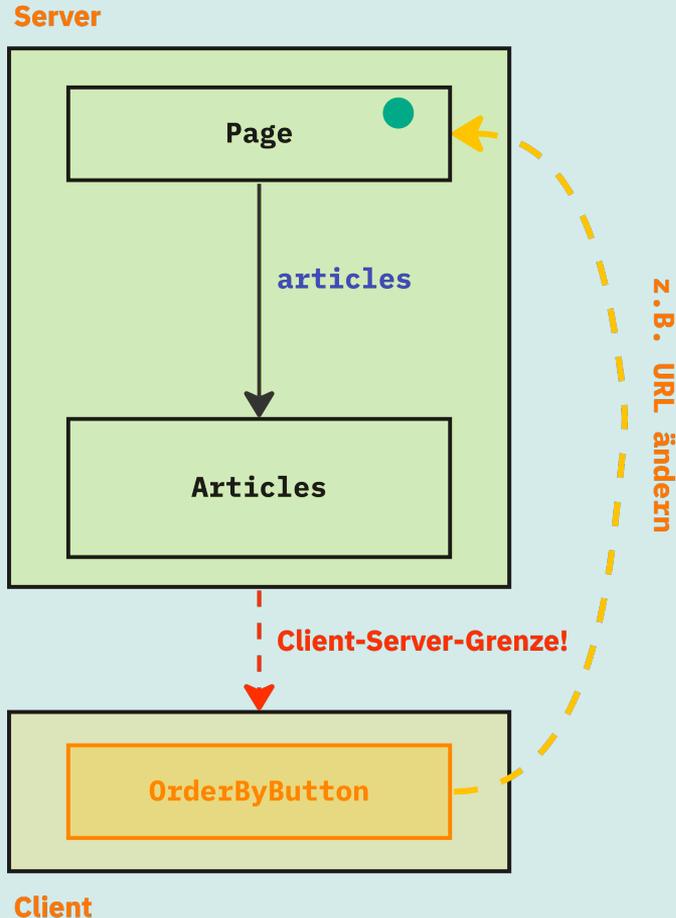
- Properties müssen Client-Server-Grenze überwinden
- Müssen serialisierbare Daten sein
- Keine (Callback-)Funktionen!
- Zur Kommunikation müssen **Server-Requests** durchgeführt werden
  - z.B. URL ändern

## ...UND AUF DEM 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

## ...UND AUF DEM 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 gerendert
- Fertiges UI kommt dafür vom Server
  - Das kann mehr Daten als bei REST-Call bedeuten!

- **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

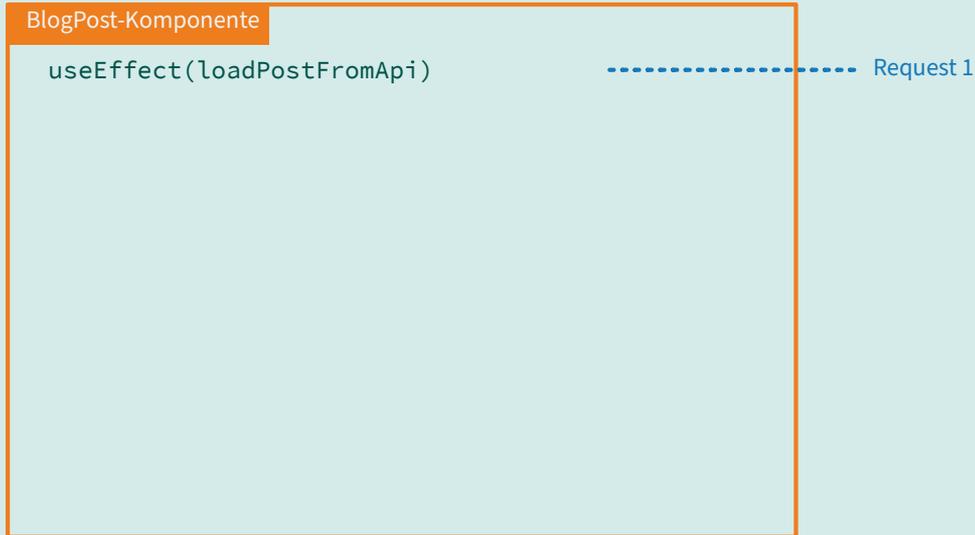
### **Mögliches Problem: Laden von Daten auf dem Client**

- Eine Komponente lädt ihre Daten, Unterkomponenten müssen warten

# DATEN LADEN

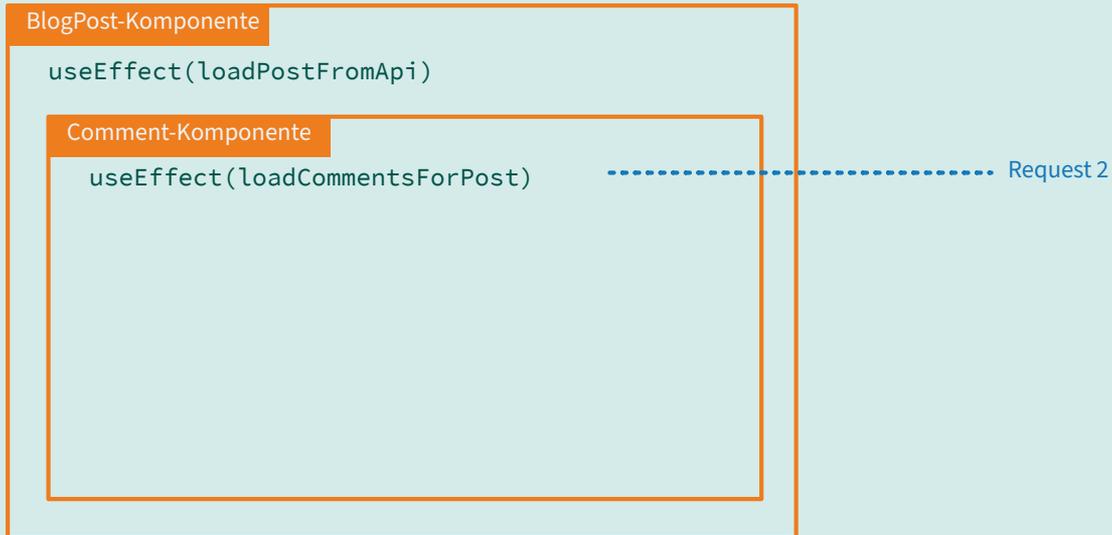
## Laden von Daten auf dem Client

- Eine Komponente lädt ihre Daten, Unterkomponenten müssen warten



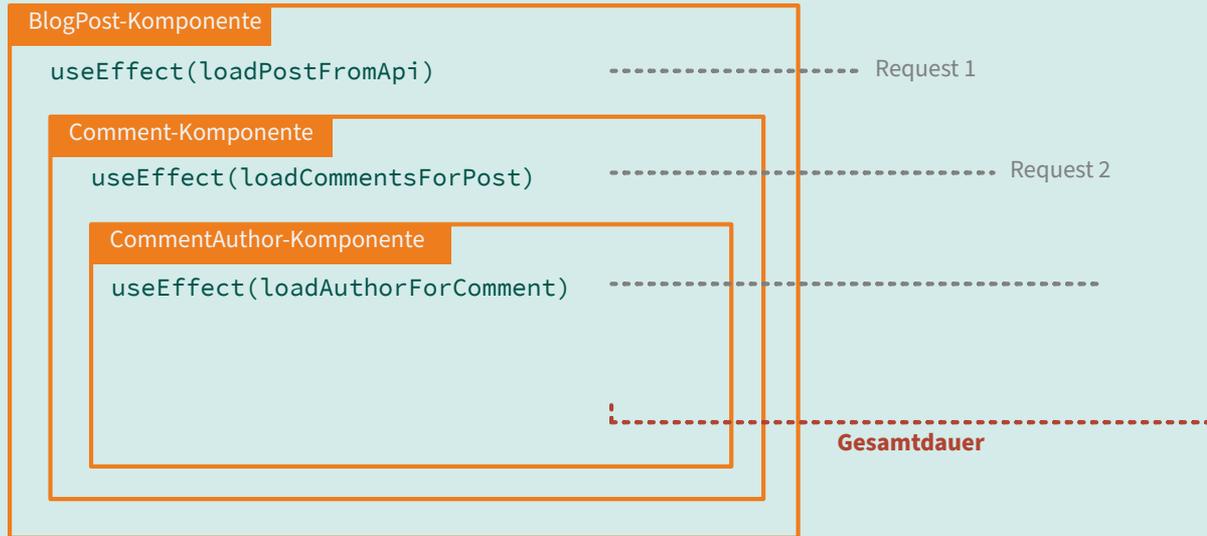
## Laden von Daten auf dem Client

- Eine Komponente lädt ihre Daten, Unterkomponenten müssen warten



## 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>  
  );  
}
```

# SUSPENSE

## Beispiel: Eine asynchrone Server Komponente

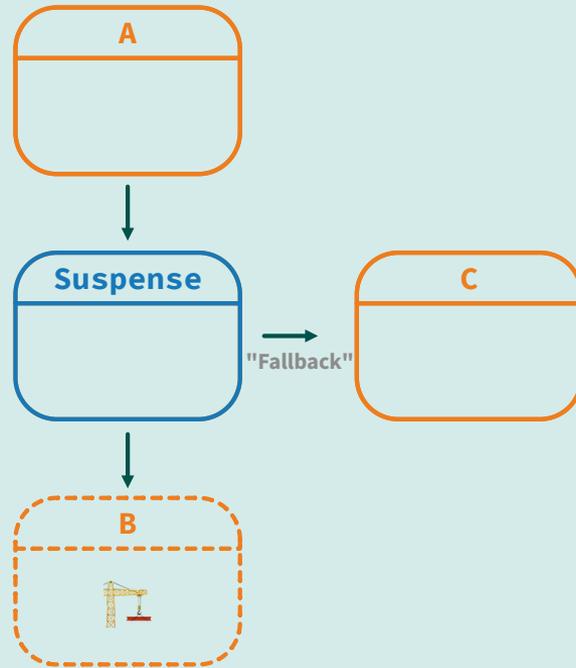
```
export default async function PostComments({ post }) {  
  const comments = await db.query("select * from comments where post_id = $1", [post.id]);  
  
  return (  
    <div>  
      <h1>Comments</h1>  
      {comments.rows.map((comment) => (  
        <p key={comment.id}>{comment.comment}</p>  
      ))}  
    </div>  
  );  
}
```

- 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

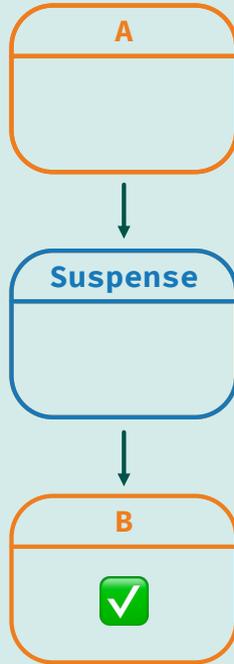
- Asynchron geladenes JS, asynchron geladene Daten, Promises, ...



# 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>;
}
```

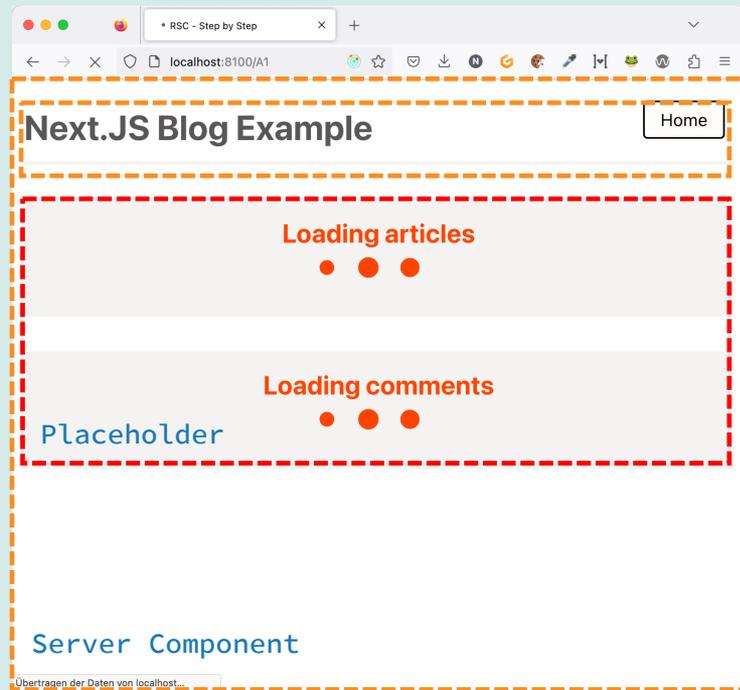
### Suspense-Komponente

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

# SUSPENSE

## 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)

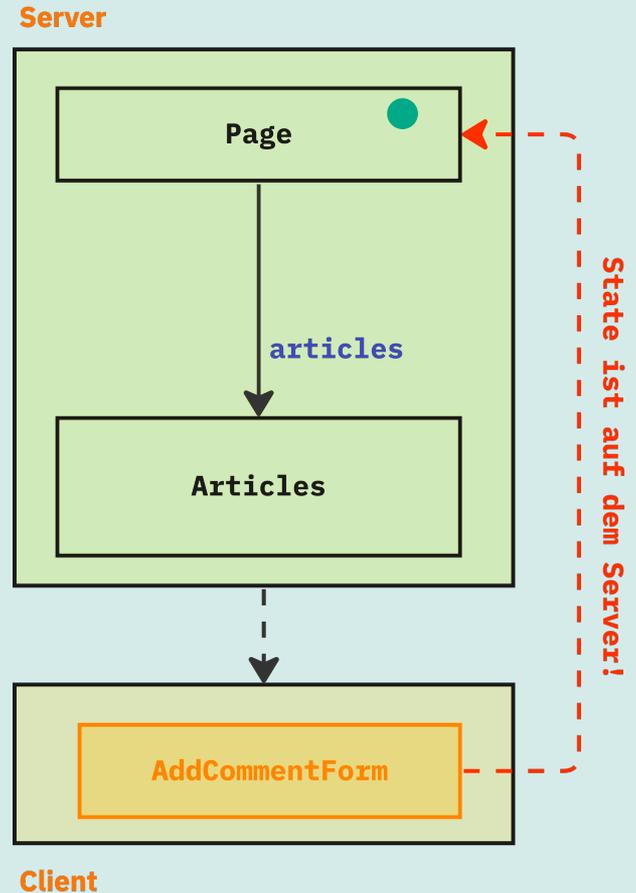
- Delay für PostList und TagCloud aktivieren (demo-config)
- Daten bleiben gecached (Home => Post => Home)
- Suspense in PostListPage verschieben
  
- Delay für Post aktivieren
- Einzelnen Post aufrufen
- Suspense in Post-Komponente deaktivieren

## Schritt 3: Suspense und Streaming

# MUTATIONS

## Verändern von Daten

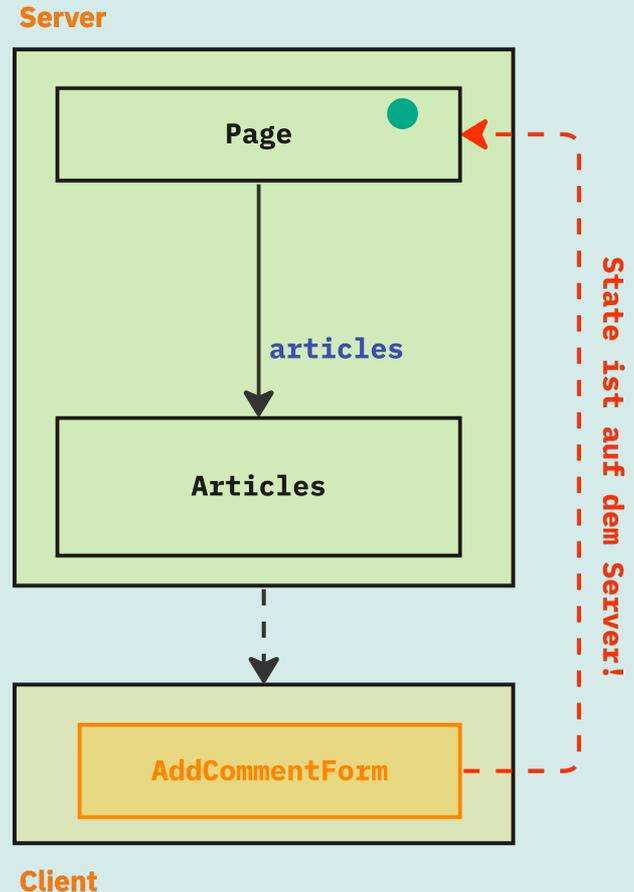
- Nach dem Verändern von Daten muss die UI aktualisiert werden
- Mangels State auf dem Client geht das aber nicht wie bislang



# MUTATIONS

## Verändern von Daten

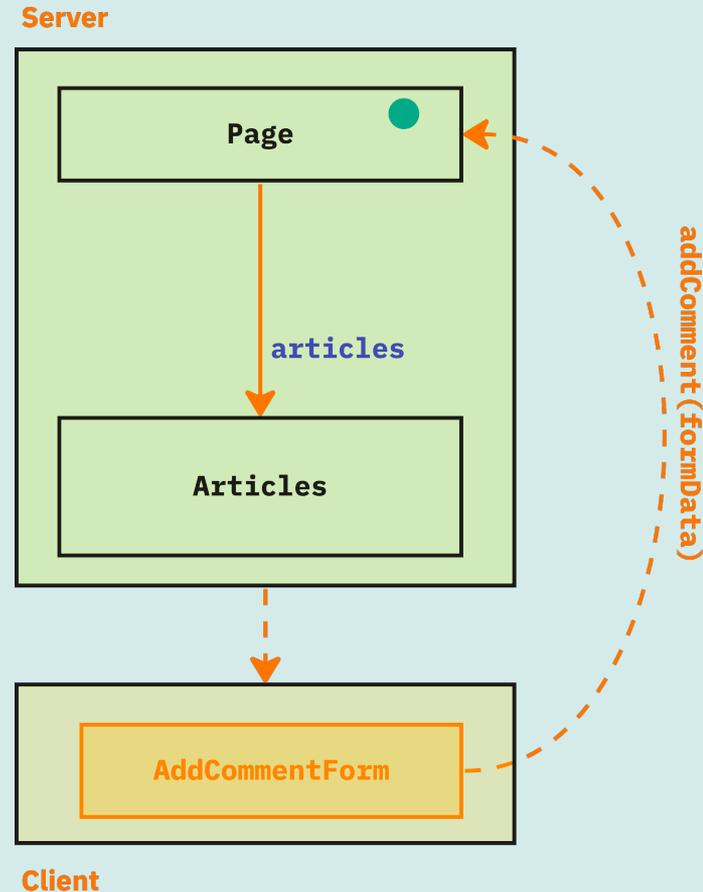
- Nach dem Verändern von Daten muss die UI aktualisiert werden
- Mangels State auf dem Client geht das aber nicht wie bislang
- Der Server muss nach Datenänderungen **aktualisierte UI** liefern



# 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
- 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?**

**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.

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

### 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?**

## SERVER COMPONENTS

### 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?

Neue Ära? Ja, aber!

### Neue Ära? Ja, aber!

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

## Neue Ära? Ja, aber!

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



<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~~ muss 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

**NILS HARTMANN**

<https://nilshartmann.net>



# Vielen Dank!

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

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

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