

**NILS HARTMANN**

<https://nilshartmann.net>

Slides: <https://react.schule/frontend-muenster-nextjs>

# Fullstack React

**Praktische Einführung in Next.js**

# NILS HARTMANN

nils@nilshartmann.net

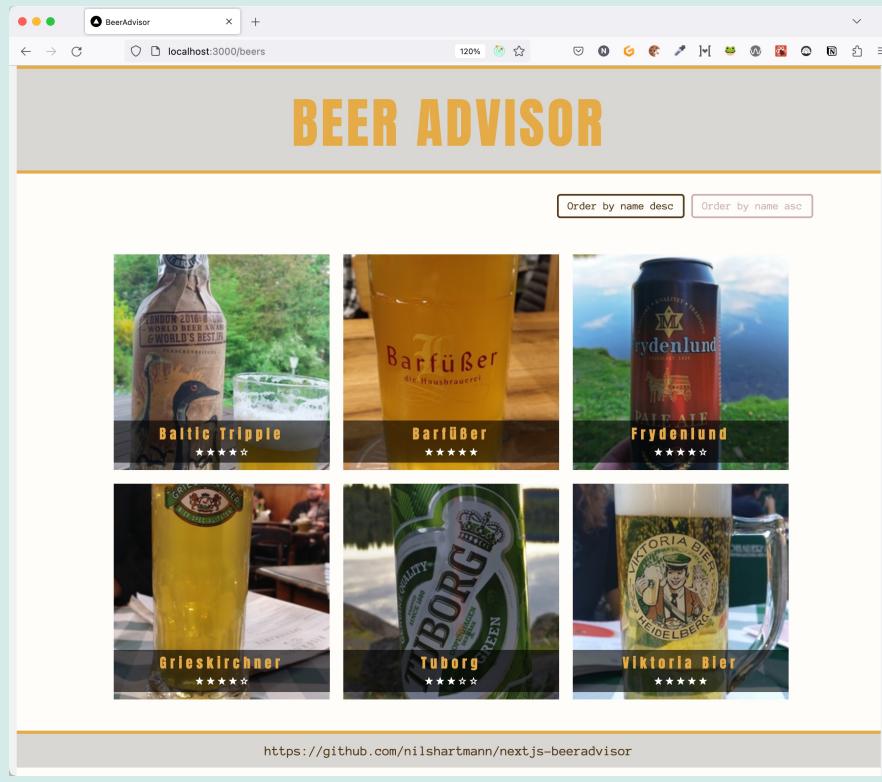
**Freiberuflicher Entwickler, Architekt, Trainer aus Hamburg**  
**Java, Spring, GraphQL, React, TypeScript**



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

<https://reactbuch.de>

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



Beispiel-Code: <https://github.com/nilshartmann/nextjs-beeradvisor>

## EIN BEISPIEL...

# RSC am Beispiel Next.js

## React empfiehlt "Fullstack-Framework"

- **Server Components** erfordern Rendern auf dem Server oder im Build
- Dazu braucht man ein "**Fullstack-Framework**"

## React empfiehlt "Fullstack-Framework"

- **Server Components** erfordern Rendern auf dem Server oder im Build
- Dazu braucht man ein "**Fullstack-Framework**"
- "**Framework**" ist verharmlosend, weil es sich in der Regel um einen kompletten Stack samt Build-Tools und Laufzeitumgebung handelt

## React empfiehlt "Fullstack-Framework"

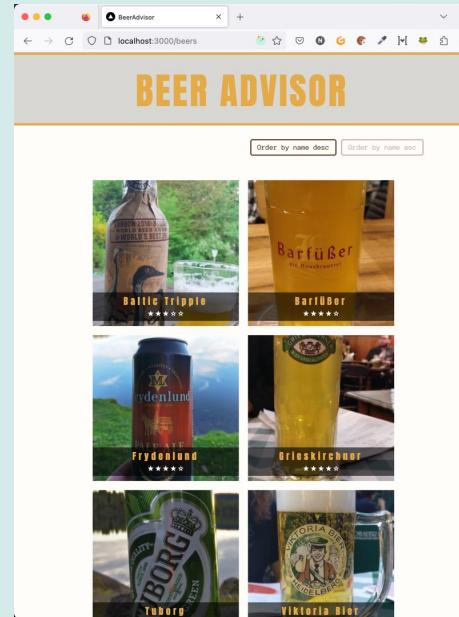
- **Server Components** erfordern Rendern auf dem Server oder im Build
- Dazu braucht man ein "**Fullstack-Framework**"
- "**Framework**" ist verharmlosend, weil es sich in der Regel um einen kompletten Stack samt Build-Tools und Laufzeitumgebung handelt
- Deswegen werden solche Frameworks auch als "**Meta-Frameworks**" bezeichnet (=> Sammlung von Frameworks)

## React empfiehlt "Fullstack-Framework"

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

## Schritt-für-Schritt

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



## Schritt 1: Eine Server Komponente



Demo

- Landing-Page mit Link auf /beers
- Children in Layout
- console.log in Page-Komponente

# Data Fetching

### Idee

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

## 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"

👉 Server-Komponenten können asynchron sein

## Schritt 2: Eine asynchrone Server-Komponente



Demo

- BeerListPage anlegen
- DB-Zugriff mit loadBeers
  - loadBeers zeigen
- BeerImageList verwenden, um Beers anzuzeigen
- 🔎 **statische Komponenten bislang! (Build!)**

## Schritt 3: Eine asynchrone Server-Komponente, träge ist



Demo

- beers/[beerId] Beer-Page mit DB (loadBeer)
- `type BeerPageProps = { params: { beerId: string } };`
- Fertige Komponente aus beer-details-page-fragment.tsx kopieren
- Aufruf künstlich verzögern (sleep in loadBeer)
- loading.tsx
- prefetch auf der /beers-Seite

## Schritt 3: Eine asynchrone Server-Komponente, die zwei Daten Quellen benötigt

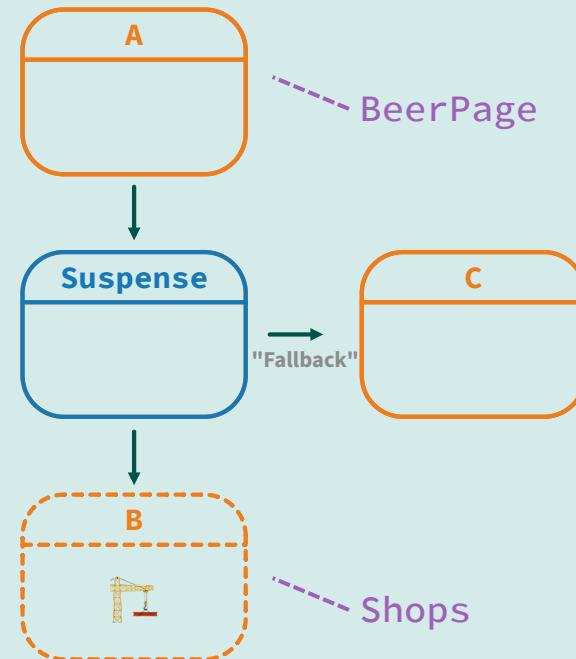


Demo

- beers/[beerId] Beer-Page wieder schnell machen (slow entfernen)
- beers/[beerId] Beer-Page shops erweitern (fertiges fetch in db-queries)
- Zeigen: Promise an Unterkomponente (Shops)
  - -> Parallel fetching!
- Aufruf künstlich verzögern (slow=2400)
- 😞 Jetzt wartet die ganze Seite auf die Shops...
- -> Suspense vorstellen (Slides)

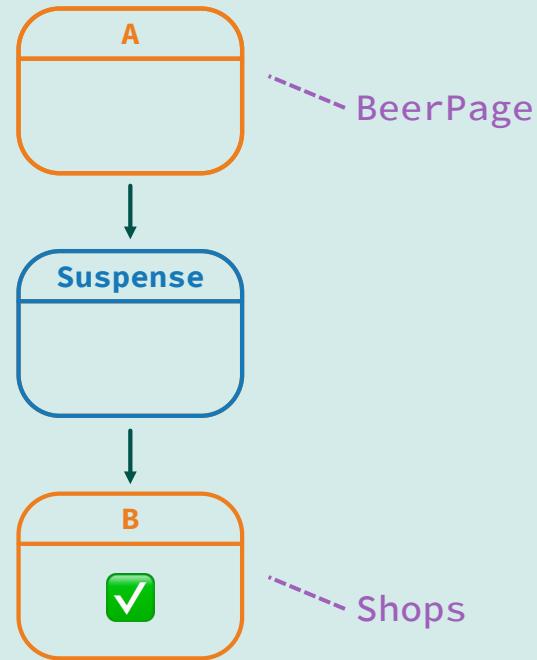
## SUSPENSE

**Suspense:** Unterbricht das Rendern, solange "etwas" fehlt



## SUSPENSE

**Suspense:** Unterbricht das Rendern, solange "etwas" fehlt



## Schritt 4: Suspense



Demo

- Aufruf von `loadShops`  
verlangsamen (`[beerId]/page.tsx`)
- Suspense in `BeerDetails`  
einführen

# Aufteilung in Server-Client: Konsequenzen

```
type BeerListProps = {
  beers: SingleBeer[];
  onToggleOrder(): void;
};

export default function BeerList({ beers, onToggleOrder }: BeerListProps) {
  return (
    <div>
      <h1>Beers</h1>

      <ul>
        {beers.map((b) => (
          <li key={b.id}>{b.name}</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.

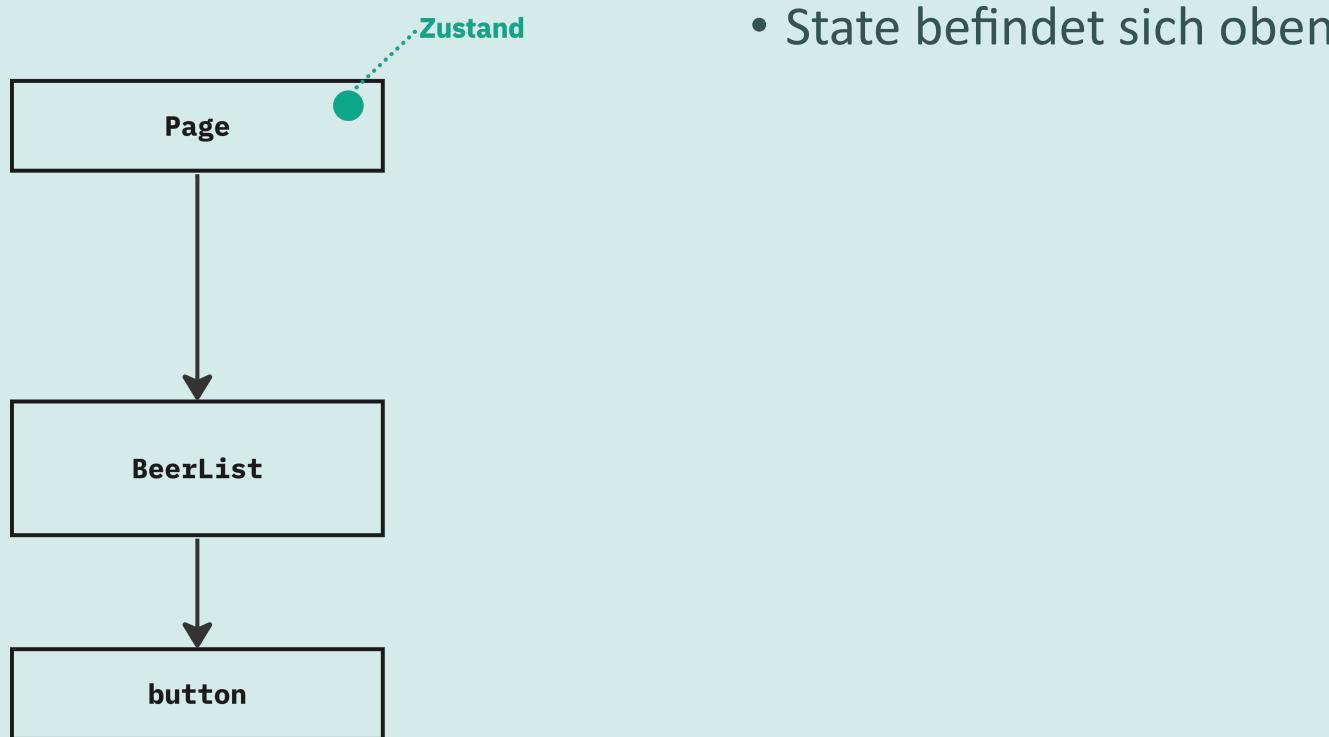
```
<button onClick={function} children=...>  
    ^^^^^^^^^^
```

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

at stringify (<anonymous>)

CAN YOU SPOT THE PROBLEM?

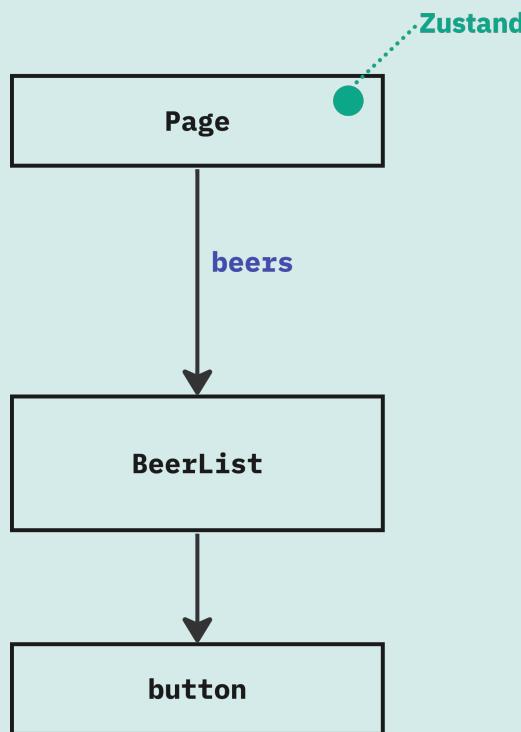
# EINE REACT ANWENDUNG IM BROWSER



- State befindet sich oben

Eine "normale" React-Anwendung...

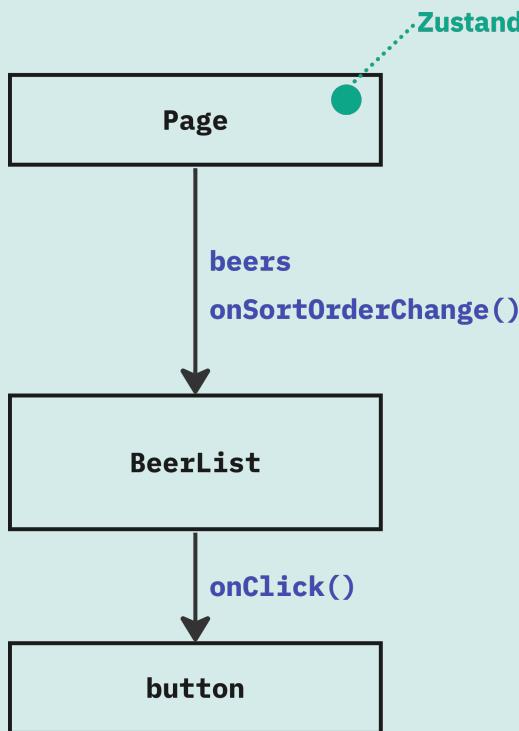
# EINE REACT ANWENDUNG IM BROWSER



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

Eine "normale" React-Anwendung...

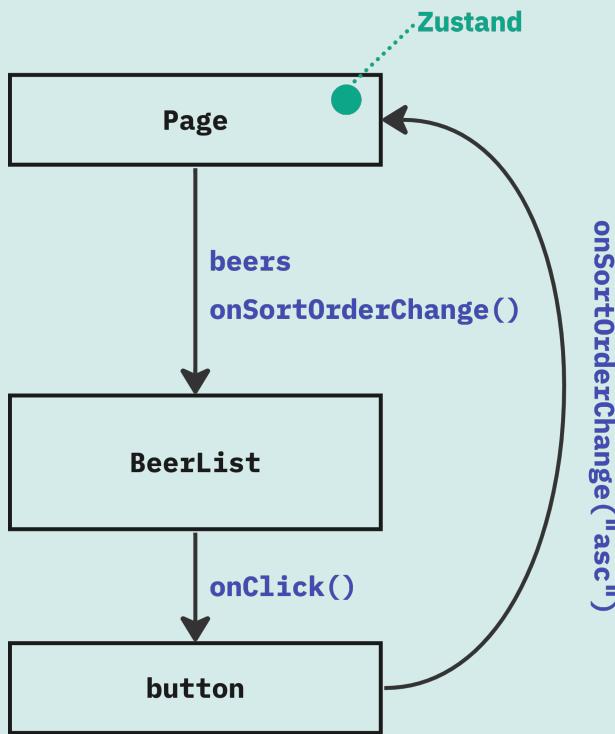
# EINE REACT ANWENDUNG IM BROWSER



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

Eine "normale" React-Anwendung...

# EINE REACT ANWENDUNG IM BROWSER

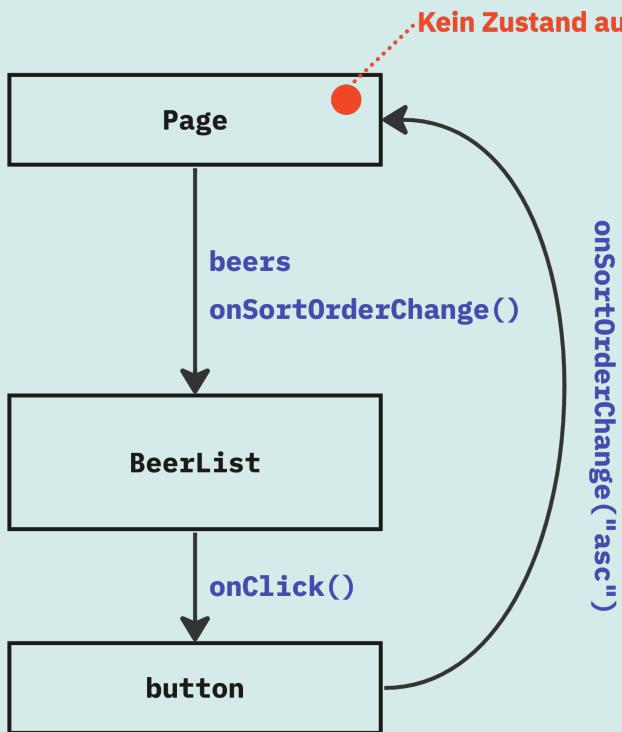


- State befindet sich oben
- Daten werden runtergereicht ("props")
- Callbacks werden runtergereicht
- Über Callbacks kann State-Veränderung ausgelöst werden

Eine "normale" React-Anwendung...

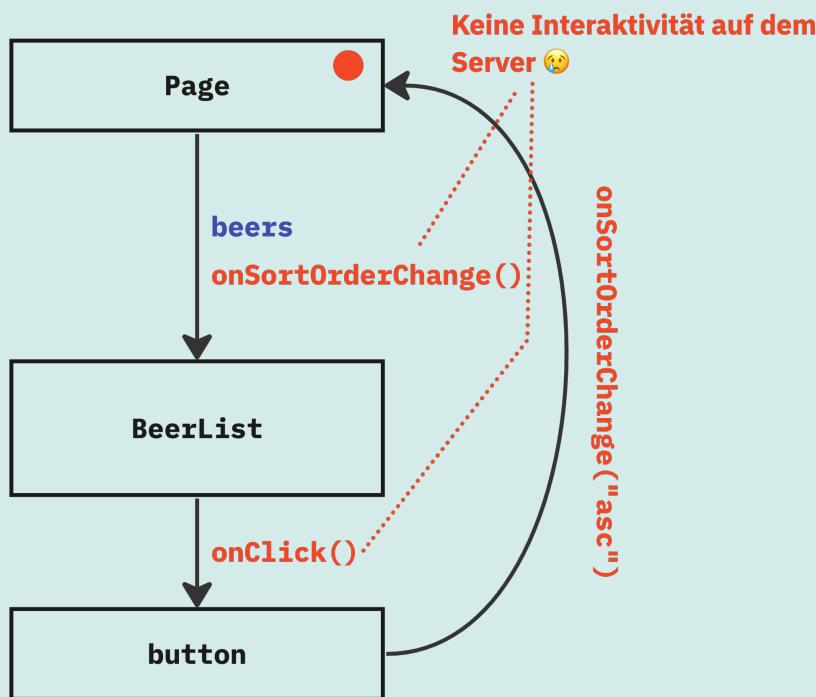
## ...UND AUF DEM SERVER

- Auf dem Server gibt es keinen State!



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

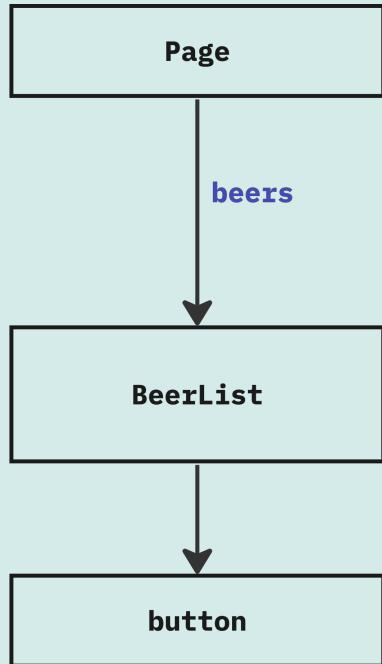
## ...UND AUF DEM SERVER



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

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

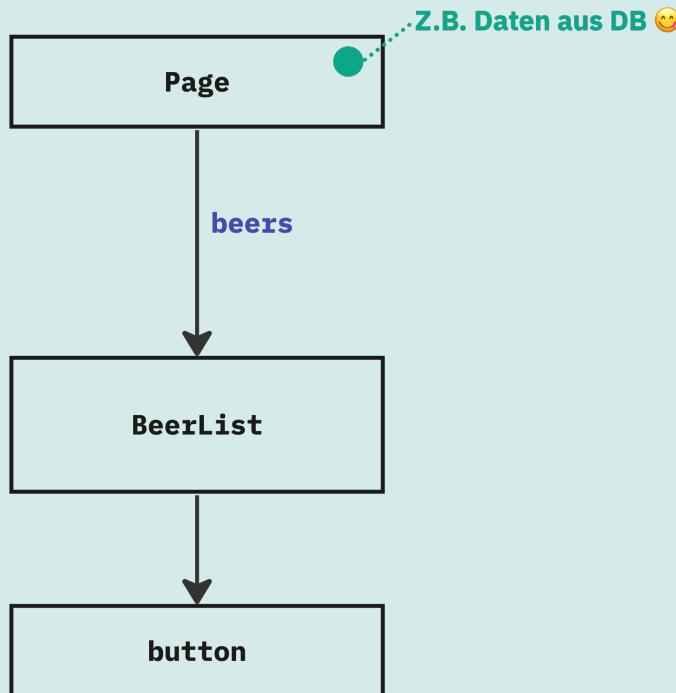
## ...UND AUF DEM SERVER



- 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

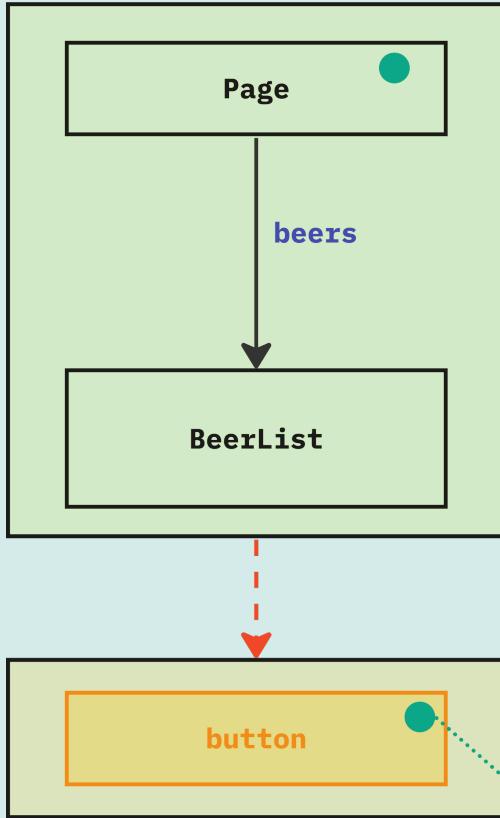


- Auf dem Server gibt es keinen State!
- ...und keine Interaktion
- Wir haben nur statischen Content
- Wir haben aber **Daten**  
z.B. aus DB, Microservice, Filesystem...

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

## ...UND AUF DEM SERVER

Server



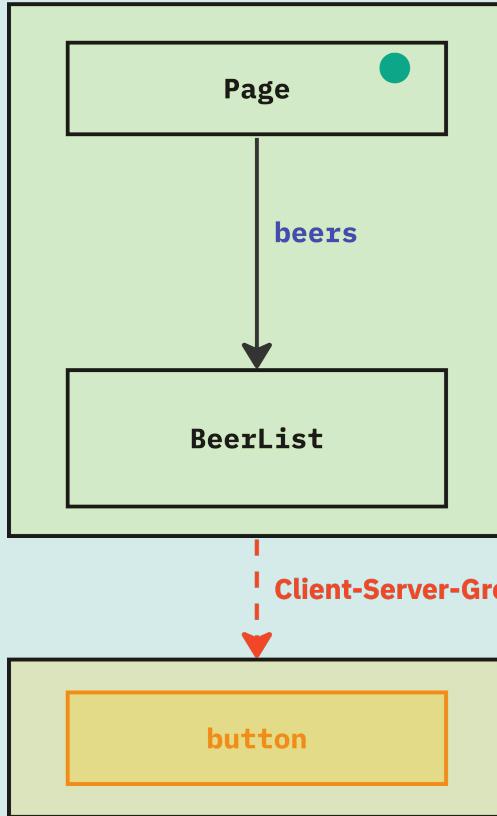
- Bestimmte Teile müssen auf den Client
  - Event-Handler
  - Lokaler State
  - Effekte

Client

Interaktives muss auf den Client 😎

## ...UND AUF DEM SERVER

Server

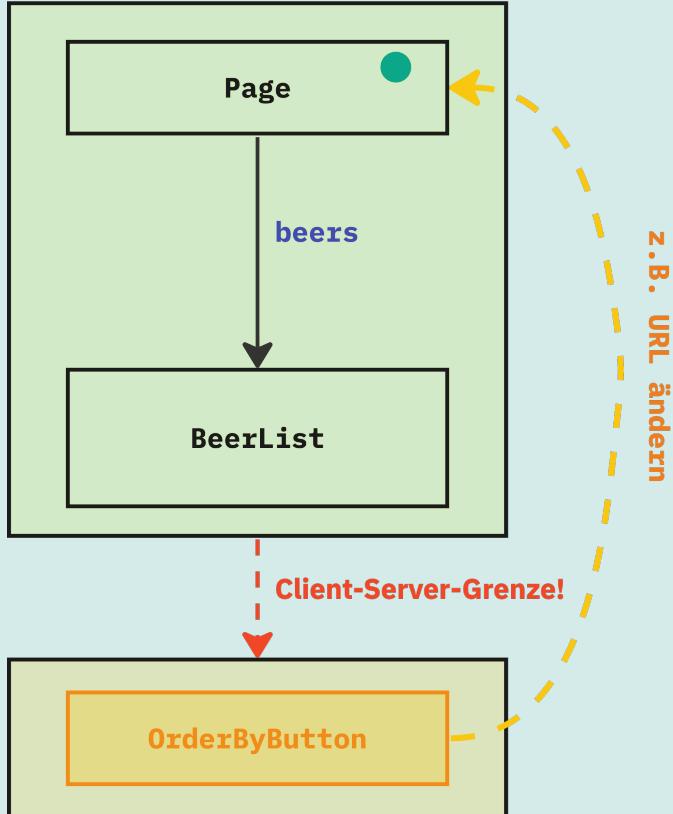


- Properties müssen hier Client-Server-Grenze überwinden
- Müssen serialisierbare Daten sein
- **Keine (Callback-)Funktionen!**

Client

## ...UND AUF DEM SERVER

Server

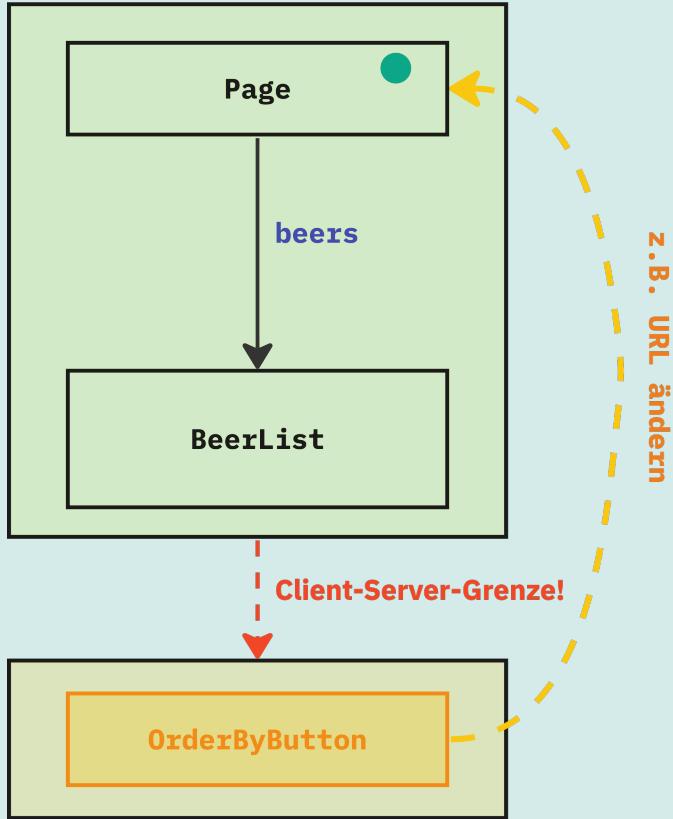


- Properties müssen Client-Server-Grenze überwinden
- Müssen serialisierbare Daten sein
- Keine (Callback-)Funktionen!
- Zur Kommunikation müssen **Server-Requests** durchgeführt werden
  - z.B. URL ändern

Client

## ...UND AUF DEM SERVER

Server

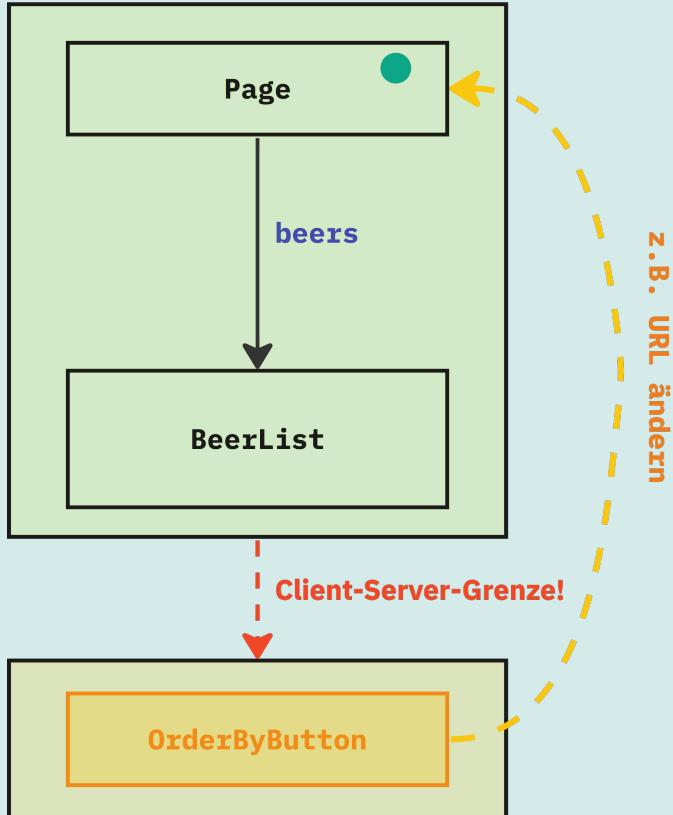


- Properties müssen Client-Server-Grenze überwinden
- Müssen serialisierbare Daten sein
- Keine (Callback-)Funktionen!
- Zur Kommunikation müssen Server-Requests durchgeführt werden
  - z.B. URL ändern
- **Server-Komponente hat Zugriff auf Request Informationen**
  - URL mit Search Params
  - Cookies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

Client

## Schritt 5: Eine Client-Komponente



### Demo

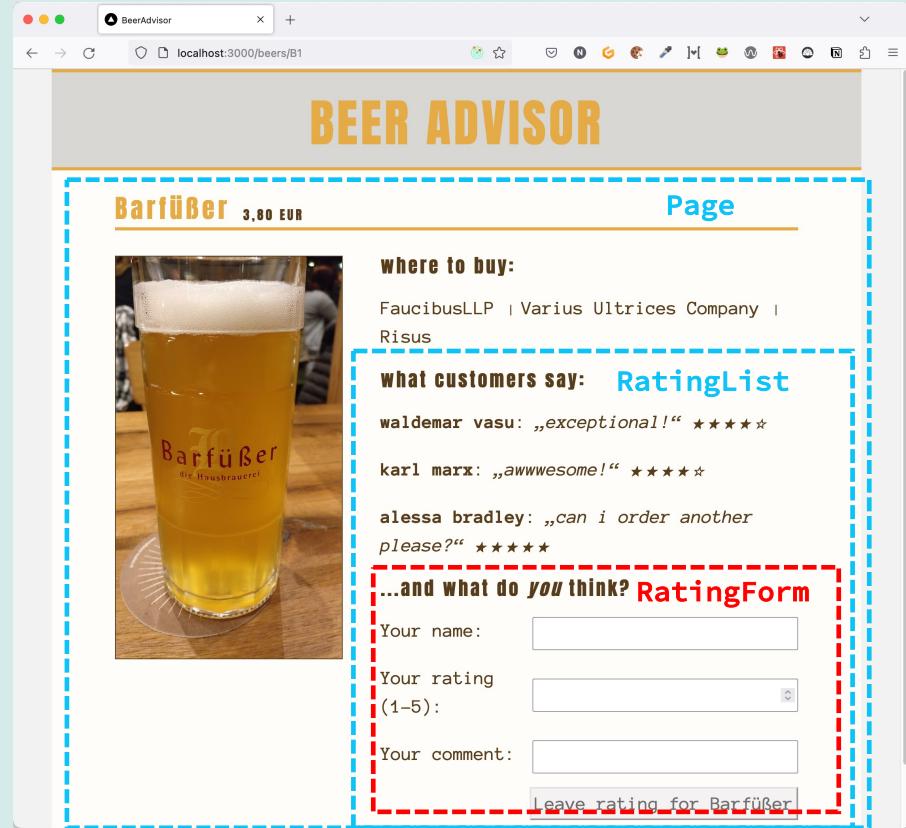
- OrderByButton Komponente bauen
  - OrderBy-Type als Property ("name\_asc" , "name\_desc")
  - Client-Komponente: Hooks möglich, useBeerAdvisorSearchParams
    - URL als "globaler Zustand"
- In BeerListPage einbauen
- In BeerListPage abhängig von SearchParams sortieren
  - An dieser Stelle Server Komponente, d.h. Hook ist hier nicht verwendbar
- ```
type BeerListPageProps = {
  searchParams?: { [key: string]: string };
};

const orderBy: OrderBy = (searchParams?.order_by || "name_asc") as OrderBy;
```

# MUTATIONS

## Verändern von Daten: Hinzufügen einer Bewertung

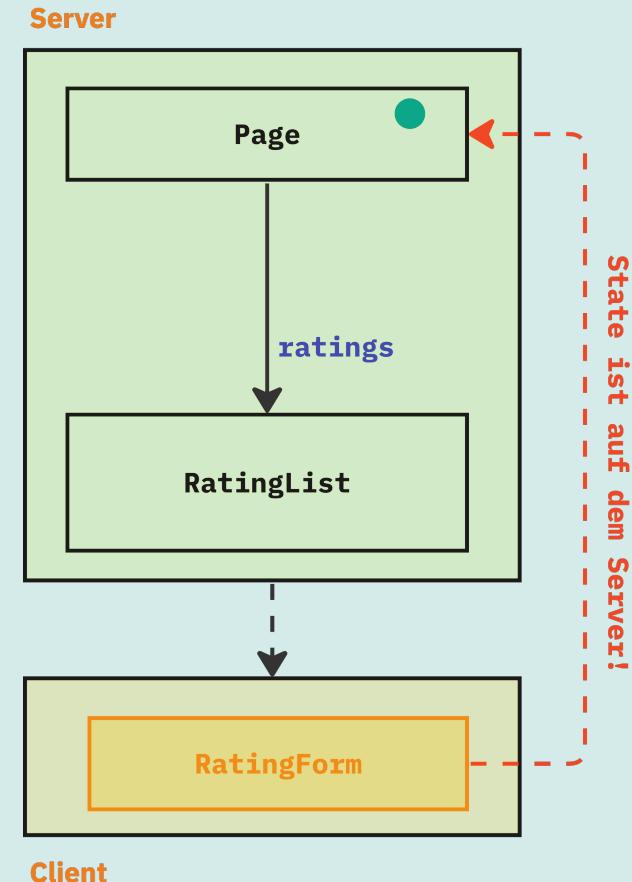
Server-Komponente  
Client-Komponente



## MUTATIONS

### Verändern von Daten

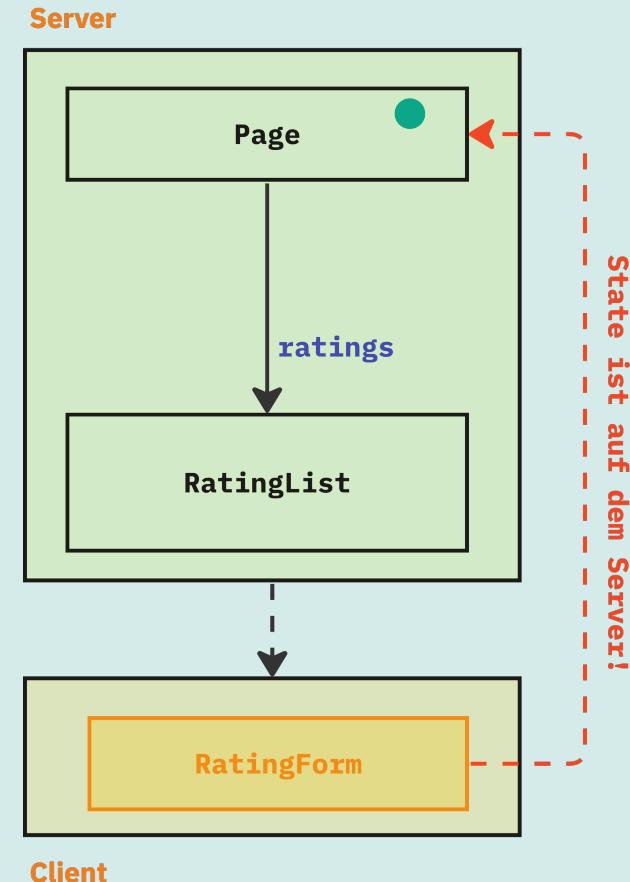
- Nach dem Verändern von Daten muss die UI aktualisiert werden
- Mangels State auf dem Client geht das aber nicht wie bislang



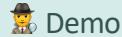
## MUTATIONS

### 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



## Schritt 6a: Exkurs API Endpunkte mit Next.js



Demo

- **API Route Handler**
  - Beispiel: GET Endpunkt implementieren !!
  - api/beers/route.ts
  - GET mit simplem NextResponse.json

## Schritt 6: Ein Formular

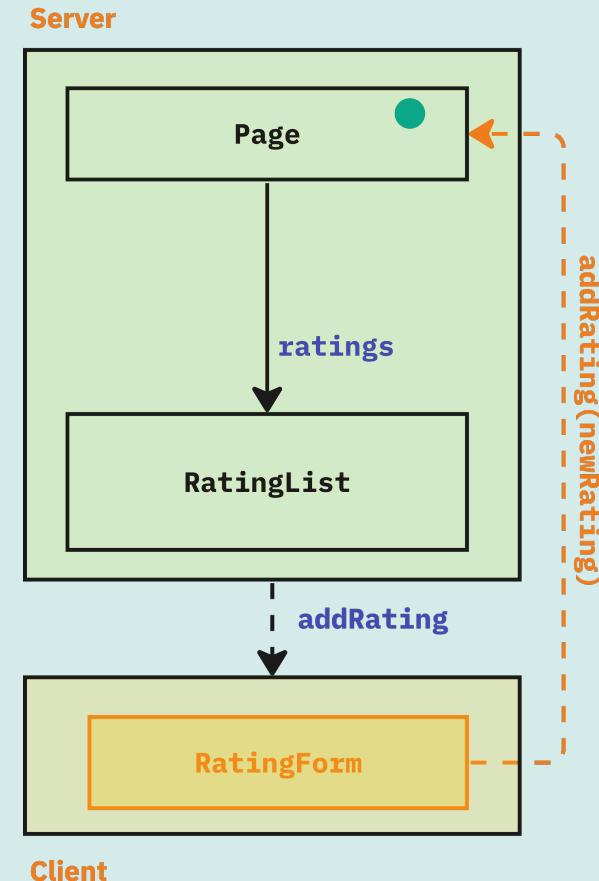


- API Route Handler
  - Fertigen POST Handler kopieren! (api/beers/[beerId])
- In AddRatingForm onSave implementieren
- mit fetch:
  - POST `/api/beers/\${beerId}/rating`
  - Content-Type-Header
  - Body: AddRatingRequestBody)
- Was passiert mit der Darstellung => nix
  - Warum?
- router.refresh()

## MUTATIONS

### Alternative: Server Actions

- **Experimentelles** React-Feature
- Das ist eine Art Remote Funktion, die aus einer Server- oder Client-Komponente aufgerufen werden kann



## Ausblick: Server Action



- Fertige Action in src-Verzeichnis kopieren (aus 99\_fertig)
  - Zeigen
- AddRatingForm anpassen
  - useTransition
  - wenn response.status === "created" dann ist alles gut, State zurücksetzen
  - Aktualisierung der UI: revalidatePath hinzufügen

**NILS HARTMANN**  
<https://nilshartmann.net>

<https://reactbuch.de>



# vielen Dank!

Slides: <https://react.schule/frontend-muenster-nextjs>

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

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