LOOX

React IL

The Story of Stores

LOOX

React IL

The Story of Stores

Tom Slutsky

💻 Works at Loox for 2 years

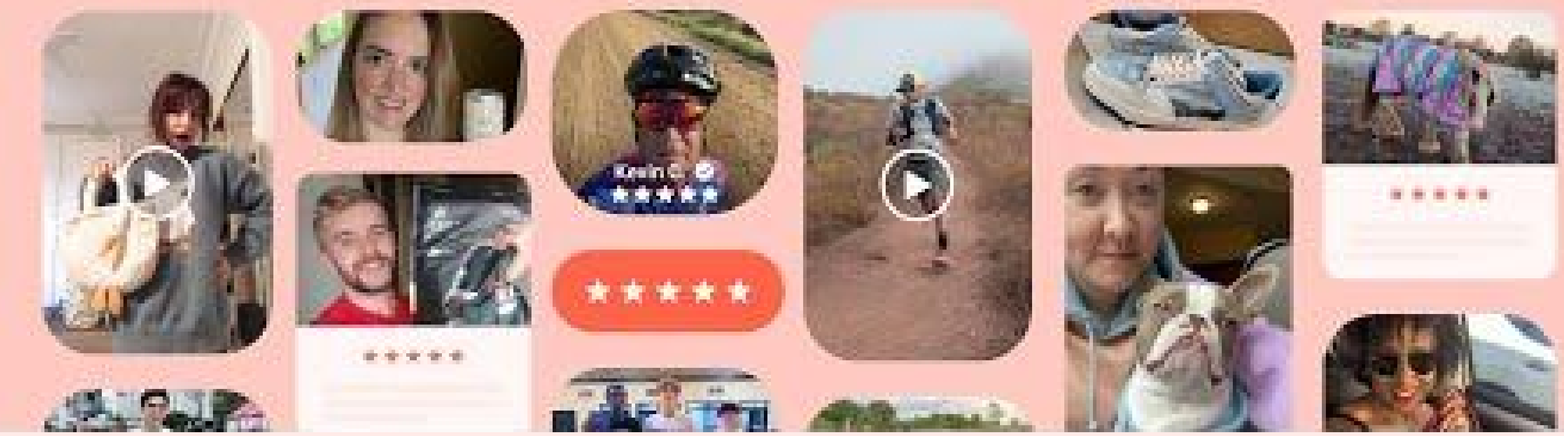
🏀 Former professional basketball player

🌳 Lives in Tivon

🐶 Married to Hadas and owner of Ness

LOOX

Social Proof That Looks Amazing



What is state?

State: A Component's Memory

Components often need to change what's on the screen as a result of an interaction. Typing into the form should update the input field, clicking "next" on an image carousel should change which image is displayed, clicking "buy" should put a product in the shopping cart. Components need to "remember" things: the current input value, the current image, the shopping cart. In React, this kind of component-specific memory is called *state*.

State: A Component's Memory

Components often need to change what's on the screen as a result of an interaction. Typing into the form should update the input field, clicking "next" on an image carousel should change which image is displayed, clicking "buy" should put a product in the shopping cart. Components need to "remember" things: the current input value, the current image, the shopping cart. In React, this kind of component-specific memory is called *state*.

Join <count> raters

DEMO TIME

```
import { useState } from "react";
import { RatingWidgetForm } from "./rating-widget-form";

export function RatingWidget() {
  const [ratings, setRatings] = useState<number[]>([]);
  return (
    <div className="rating-container">
      <h1>Join {ratings.length} raters!</h1>
      <RatingWidgetForm
        onSubmit={(rating) => setRatings((prev) => [...prev, rating])}
      />
    </div>
  );
}
```

```
import { useState } from "react";
import { RatingWidgetForm } from "./rating-widget-form";

export function RatingWidget() {
  const [ratings, setRatings] = useState<number[]>([]);
  return (
    <div className="rating-container">
      <h1>Join {ratings.length} raters!</h1>
      <RatingWidgetForm
        onSubmit={(rating) => setRatings((prev) => [...prev, rating])}
      />
    </div>
  );
}
```

```
import { useState } from "react";
import { RatingWidgetForm } from "./rating-widget-form";

export function RatingWidget() {
  const [ratings, setRatings] = useState<number[]>([]);
  return (
    <div className="rating-container">
      <h1>Join {ratings.length} raters!</h1>
      <RatingWidgetForm
        onSubmit={(rating) => setRatings((prev) => [...prev, rating])}
      />
    </div>
  );
}
```

```
import { useState } from "react";
import { RatingWidgetForm } from "./rating-widget-form";

export function RatingWidget() {
  const [ratings, setRatings] = useState<number[]>([]);
  return (
    <div className="rating-container">
      <h1>Join {ratings.length} raters!</h1>
      <RatingWidgetForm
        onSubmit={(rating) => setRatings((prev) => [...prev, rating])}
      />
    </div>
  );
}
```

Let’s see it live

State Lifting

```
export function Component() {
  const { title, image, description, price } = useLoaderData();
  const [ratings, setRatings] = useState<number[]>([]);

  return (
    <div className="product">
      <div className="product-image">
        <img src={image} alt={title} />
      </div>
      <div className="product-info">
        <span className="product-title">
          <h2>{title}</h2>
          <span>({avg(ratings).toFixed(1)})</span>
        </span>
        <p className="price">${price}</p>
        <p className="description">{description}</p>
        <button className="buy-button">Add to Cart</button>
        <div className="rating-section">
          <RatingWidget
            raters={ratings.length}
            onRatingAdded={(rating) => setRatings((prev) => [...prev, rating])}
          />
        </div>
      </div>
    </div>
  );
}
```

```
export function Component() {
  const { title, image, description, price } = useLoaderData();
  const [ratings, setRatings] = useState<number[]>([]);

  return (
    <div className="product">
      <div className="product-image">
        <img src={image} alt={title} />
      </div>
      <div className="product-info">
        <span className="product-title">
          <h2>{title}</h2>
          <span>({avg(ratings).toFixed(1)})</span>
        </span>
        <p className="price">${price}</p>
        <p className="description">{description}</p>
        <button className="buy-button">Add to Cart</button>
        <div className="rating-section">
          <RatingWidget
            raters={ratings.length}
            onRatingAdded={(rating) => setRatings((prev) => [...prev, rating])}
          />
        </div>
      </div>
    </div>
  );
}
```

```
export function Component() {
  const { title, image, description, price } = useLoaderData();
  const [ratings, setRatings] = useState<number[]>([]);

  return (
    <div className="product">
      <div className="product-image">
        <img src={image} alt={title} />
      </div>
      <div className="product-info">
        <span className="product-title">
          <h2>{title}</h2>
          <span>({avg(ratings).toFixed(1)})</span>
        </span>
        <p className="price">${price}</p>
        <p className="description">{description}</p>
        <button className="buy-button">Add to Cart</button>
        <div className="rating-section">
          <RatingWidget
            raters={ratings.length}
            onRatingAdded={(rating) => setRatings((prev) => [...prev, rating])}
          />
        </div>
      </div>
    </div>
  );
}
```

```
export function Component() {
  const { title, image, description, price } = useLoaderData();
  const [ratings, setRatings] = useState<number[]>([]);

  return (
    <div className="product">
      <div className="product-image">
        <img src={image} alt={title} />
      </div>
      <div className="product-info">
        <span className="product-title">
          <h2>{title}</h2>
          <span>({avg(ratings).toFixed(1)})</span>
        </span>
        <p className="price">${price}</p>
        <p className="description">{description}</p>
        <button className="buy-button">Add to Cart</button>
        <div className="rating-section">
          <RatingWidget
            raters={ratings.length}
            onRatingAdded={(rating) => setRatings((prev) => [...prev, rating])}
          />
        </div>
      </div>
    </div>
  );
}
```

Let’s see it live

Passing Data Deeply with Context

Usually, you will pass information from a parent component to a child component via props. But passing props can become verbose and inconvenient if you have to pass them through many components in the middle, or if many components in your app need the same information. *Context* lets the parent component make some information available to any component in the tree below it—no matter how deep—without passing it explicitly through props.

Passing Data Deeply with Context

Usually, you will pass information from a parent component to a child component via props. But passing props can become verbose and inconvenient if you have to pass them through many components in the middle, or if many components in your app need the same information. *Context* lets the parent component make some information available to any component in the tree below it—no matter how deep—without passing it explicitly through props.

DEMO TIME

```
const ctx = createContext<{
  ratings: Rating[];
  addRating: (rating: Rating) => void;
}>(null);

export const RatingContextProvider = ({children}) => {
  const [ratings, setRatings] = useState<Rating[]>([]);

  return (
    <ctx.Provider
      value={{
        ratings,
        addRating:(rating: Rating) => setRatings((prev) => [...prev, rating]),
      }}
    >
      {children}
    </ctx.Provider>
  );
};
```

```
export function Component() {
  const { title, image, description, price, slug } = useLoaderData();
  const { ratings, addRating } = useRatingContext();
  const filteredRatings = ratings.filter((r) => r.product === slug);

  return (
    <div className="product">
      <div className="product-image">
        <img src={image} alt={title} />
      </div>
      <div className="product-info">
        <span className="product-title">
          <h2>{title}</h2>
          <span>
            ({avg(filteredRatings.map((rating) => rating.rating)).toFixed(1)})
          </span>
        </span>
        <p className="price">${price}</p>
        <p className="description">{description}</p>
        <button className="buy-button">Add to Cart</button>
        <div className="rating-section">
          <RatingWidget
            raters={filteredRatings.length}
            onRatingAdded={(rating) => addRating({ product: slug, rating })}
          />
        </div>
      </div>
    </div>
  );
}
```

```
const ctx = createContext<{
  ratings: Rating[];
  addRating: (rating: Rating) => void;
}>(null);

export const RatingContextProvider = ({children}) => {
  const [ratings, setRatings] = useState<Rating[]>([]);

  return (
    <ctx.Provider
      value={{
        ratings,
        addRating:(rating: Rating) => setRatings((prev) => [...prev, rating]),
      }}
    >
      {children}
    </ctx.Provider>
  );
};
```

```
export function Component() {
  const { title, image, description, price, slug } = useLoaderData();
  const { ratings, addRating } = useRatingContext();
  const filteredRatings = ratings.filter((r) => r.product === slug);

  return (
    <div className="product">
      <div className="product-image">
        <img src={image} alt={title} />
      </div>
      <div className="product-info">
        <span className="product-title">
          <h2>{title}</h2>
          <span>
            ({avg(filteredRatings.map((rating) => rating.rating)).toFixed(1)})
          </span>
        </span>
        <p className="price">${price}</p>
        <p className="description">{description}</p>
        <button className="buy-button">Add to Cart</button>
        <div className="rating-section">
          <RatingWidget
            raters={filteredRatings.length}
            onRatingAdded={(rating) => addRating({ product: slug, rating })}
          />
        </div>
      </div>
    </div>
  );
}
```

```
const ctx = createContext<{
  ratings: Rating[];
  addRating: (rating: Rating) => void;
}>(null);

export const RatingContextProvider = ({children}) => {
  const [ratings, setRatings] = useState<Rating[]>([]);

  return (
    <ctx.Provider
      value={{
        ratings,
        addRating:(rating: Rating) => setRatings((prev) => [...prev, rating]),
      }}
    >
      {children}
    </ctx.Provider>
  );
};
```

```
export function Component() {
  const { title, image, description, price, slug } = useLoaderData();
  const { ratings, addRating } = useRatingContext();
  const filteredRatings = ratings.filter((r) => r.product === slug);

  return (
    <div className="product">
      <div className="product-image">
        <img src={image} alt={title} />
      </div>
      <div className="product-info">
        <span className="product-title">
          <h2>{title}</h2>
          <span>
            ({avg(filteredRatings.map((rating) => rating.rating)).toFixed(1)})
          </span>
        </span>
        <p className="price">${price}</p>
        <p className="description">{description}</p>
        <button className="buy-button">Add to Cart</button>
        <div className="rating-section">
          <RatingWidget
            raters={filteredRatings.length}
            onRatingAdded={(rating) => addRating({ product: slug, rating })}
          />
        </div>
      </div>
    </div>
  );
}
```

```
const ctx = createContext<{
  ratings: Rating[];
  addRating: (rating: Rating) => void;
}>(null);

export const RatingContextProvider = ({children}) => {
  const [ratings, setRatings] = useState<Rating[]>([]);

  return (
    <ctx.Provider
      value={{
        ratings,
        addRating:(rating: Rating) => setRatings((prev) => [...prev, rating]),
      }}
    >
      {children}
    </ctx.Provider>
  );
};
```

```
export function Component() {
  const { title, image, description, price, slug } = useLoaderData();
  const { ratings, addRating } = useRatingContext();
  const filteredRatings = ratings.filter((r) => r.product === slug);

  return (
    <div className="product">
      <div className="product-image">
        <img src={image} alt={title} />
      </div>
      <div className="product-info">
        <span className="product-title">
          <h2>{title}</h2>
          <span>
            ({avg(filteredRatings.map((rating) => rating.rating)).toFixed(1)})
          </span>
        </span>
        <p className="price">${price}</p>
        <p className="description">{description}</p>
        <button className="buy-button">Add to Cart</button>
        <div className="rating-section">
          <RatingWidget
            raters={filteredRatings.length}
            onRatingAdded={(rating) => addRating({ product: slug, rating })}
          />
        </div>
      </div>
    </div>
  );
}
```

```
const ctx = createContext<{
  ratings: Rating[];
  addRating: (rating: Rating) => void;
}>(null);

export const RatingContextProvider = ({children}) => {
  const [ratings, setRatings] = useState<Rating[]>([]);

  return (
    <ctx.Provider
      value={{
        ratings,
        addRating:(rating: Rating) => setRatings((prev) => [...prev, rating]),
      }}
    >
      {children}
    </ctx.Provider>
  );
};
```

```
export function Component() {
  const { title, image, description, price, slug } = useLoaderData();
  const { ratings, addRating } = useRatingContext();
  const filteredRatings = ratings.filter((r) => r.product === slug);

  return (
    <div className="product">
      <div className="product-image">
        <img src={image} alt={title} />
      </div>
      <div className="product-info">
        <span className="product-title">
          <h2>{title}</h2>
          <span>
            ({avg(filteredRatings.map((rating) => rating.rating)).toFixed(1)})
          </span>
        </span>
        <p className="price">${price}</p>
        <p className="description">{description}</p>
        <button className="buy-button">Add to Cart</button>
        <div className="rating-section">
          <RatingWidget
            raters={filteredRatings.length}
            onRatingAdded={(rating) => addRating({ product: slug, rating })}
          />
        </div>
      </div>
    </div>
  );
}
```

Let’s see it live

Stores

Stores

Stores

Stores are a centralized place to hold application state **outside** of React components.

Stores are plain Javascript objects.

Stores can be framework agnostic

`useSyncExternalStore` is a React Hook that lets you subscribe to an external store.

```
const snapshot = useSyncExternalStore(subscribe, getSnapshot, getServerSnapshot?)
```

`useSyncExternalStore` is a React Hook that lets you subscribe to an external store.

```
const snapshot = useSyncExternalStore(subscribe, getSnapshot, getServerSnapshot?)
```

It returns the `snapshot` of the data in the store. You need to pass two functions as arguments:

1. The `subscribe` function should subscribe to the store and return a function that unsubscribes.
2. The `getSnapshot` function should read a snapshot of the data from the store.

DEMO TIME

```
export const store = {
  data: [] as Rating[],
  addRating: ({ product, rating }: { product: string; rating: number }) => {
    store.data = [...store.data, { product, rating }];
    store.emit();
  },
  subscribers: new Set<Subscriber>(),
  subscribe: (subscriber: Subscriber) => {
    store.subscribers.add(subscriber);
    return () => store.subscribers.delete(subscriber);
  },
  emit: () => {
    for (const subscriber of store.subscribers) {
      subscriber();
    }
  },
};

export const useRatingsStore = () =>
  useSyncExternalStore(store.subscribe, () => store.data);
```

```
export function Component() {
  const { title, image, description, price, slug } = useLoaderData();
  const ratings = useRatingsStore();
  const filteredRatings = ratings.filter((r) => r.product === slug);
  return (
    <div className="product">
      <div className="product-image">
        <img src={image} alt={title} />
      </div>
      <div className="product-info">
        <span className="product-title">
          <h2>{title}</h2>
          <span>
            ({avg(filteredRatings.map((rating) => rating.rating)).toFixed(1)})
          </span>
        </span>
        <p className="price">${price}</p>
        <p className="description">{description}</p>
        <button className="buy-button">Add to Cart</button>
        <div className="rating-section">
          <RatingWidget
            raters={filteredRatings.length}
            onRatingAdded={(rating) =>
              store.addRating({ product: slug, rating })
            }
          />
        </div>
      </div>
    </div>
  );
}
```

```
export const store = {
  data: [] as Rating[],
  addRating: ({ product, rating }: { product: string; rating: number }) => {
    store.data = [...store.data, { product, rating }];
    store.emit();
  },
  subscribers: new Set<Subscriber>(),
  subscribe: (subscriber: Subscriber) => {
    store.subscribers.add(subscriber);
    return () => store.subscribers.delete(subscriber);
  },
  emit: () => {
    for (const subscriber of store.subscribers) {
      subscriber();
    }
  },
};

export const useRatingsStore = () =>
  useSyncExternalStore(store.subscribe, () => store.data);
```

```
export function Component() {
  const { title, image, description, price, slug } = useLoaderData();
  const ratings = useRatingsStore();
  const filteredRatings = ratings.filter((r) => r.product === slug);
  return (
    <div className="product">
      <div className="product-image">
        <img src={image} alt={title} />
      </div>
      <div className="product-info">
        <span className="product-title">
          <h2>{title}</h2>
          <span>
            ({avg(filteredRatings.map((rating) => rating.rating)).toFixed(1)})
          </span>
        </span>
        <p className="price">${price}</p>
        <p className="description">{description}</p>
        <button className="buy-button">Add to Cart</button>
        <div className="rating-section">
          <RatingWidget
            raters={filteredRatings.length}
            onRatingAdded={(rating) =>
              store.addRating({ product: slug, rating })
            }
          />
        </div>
      </div>
    </div>
  );
}
```

```
export const store = {
  data: [] as Rating[],
  addRating: ({ product, rating }: { product: string; rating: number }) => {
    store.data = [...store.data, { product, rating }];
    store.emit();
  },
  subscribers: new Set<Subscriber>(),
  subscribe: (subscriber: Subscriber) => {
    store.subscribers.add(subscriber);
    return () => store.subscribers.delete(subscriber);
  },
  emit: () => {
    for (const subscriber of store.subscribers) {
      subscriber();
    }
  },
};

export const useRatingsStore = () =>
  useSyncExternalStore(store.subscribe, () => store.data);
```

```
export function Component() {
  const { title, image, description, price, slug } = useLoaderData();
  const ratings = useRatingsStore();
  const filteredRatings = ratings.filter((r) => r.product === slug);
  return (
    <div className="product">
      <div className="product-image">
        <img src={image} alt={title} />
      </div>
      <div className="product-info">
        <span className="product-title">
          <h2>{title}</h2>
          <span>
            ({avg(filteredRatings.map((rating) => rating.rating)).toFixed(1)})
          </span>
        </span>
        <p className="price">${price}</p>
        <p className="description">{description}</p>
        <button className="buy-button">Add to Cart</button>
        <div className="rating-section">
          <RatingWidget
            raters={filteredRatings.length}
            onRatingAdded={(rating) =>
              store.addRating({ product: slug, rating })
            }
          />
        </div>
      </div>
    </div>
  );
}
```

```
export const store = {
  data: [] as Rating[],
  addRating: ({ product, rating }: { product: string; rating: number }) => {
    store.data = [...store.data, { product, rating }];
    store.emit();
  },
  subscribers: new Set<Subscriber>(),
  subscribe: (subscriber: Subscriber) => {
    store.subscribers.add(subscriber);
    return () => store.subscribers.delete(subscriber);
  },
  emit: () => {
    for (const subscriber of store.subscribers) {
      subscriber();
    }
  },
};

export const useRatingsStore = () =>
  useSyncExternalStore(store.subscribe, () => store.data);
```

```
export function Component() {
  const { title, image, description, price, slug } = useLoaderData();
  const ratings = useRatingsStore();
  const filteredRatings = ratings.filter((r) => r.product === slug);
  return (
    <div className="product">
      <div className="product-image">
        <img src={image} alt={title} />
      </div>
      <div className="product-info">
        <span className="product-title">
          <h2>{title}</h2>
          <span>
            ({avg(filteredRatings.map((rating) => rating.rating)).toFixed(1)})
          </span>
        </span>
        <p className="price">${price}</p>
        <p className="description">{description}</p>
        <button className="buy-button">Add to Cart</button>
        <div className="rating-section">
          <RatingWidget
            raters={filteredRatings.length}
            onRatingAdded={(rating) =>
              store.addRating({ product: slug, rating })
            }
          />
        </div>
      </div>
    </div>
  );
}
```

```
export const store = {
  data: [] as Rating[],
  addRating: ({ product, rating }: { product: string; rating: number }) => {
    store.data = [...store.data, { product, rating }];
    store.emit();
  },
  subscribers: new Set<Subscriber>(),
  subscribe: (subscriber: Subscriber) => {
    store.subscribers.add(subscriber);
    return () => store.subscribers.delete(subscriber);
  },
  emit: () => {
    for (const subscriber of store.subscribers) {
      subscriber();
    }
  },
};

export const useRatingsStore = () =>
  useSyncExternalStore(store.subscribe, () => store.data);
```

```
export function Component() {
  const { title, image, description, price, slug } = useLoaderData();
  const ratings = useRatingsStore();
  const filteredRatings = ratings.filter((r) => r.product === slug);
  return (
    <div className="product">
      <div className="product-image">
        <img src={image} alt={title} />
      </div>
      <div className="product-info">
        <span className="product-title">
          <h2>{title}</h2>
          <span>
            ({avg(filteredRatings.map((rating) => rating.rating)).toFixed(1)})
          </span>
        </span>
        <p className="price">${price}</p>
        <p className="description">{description}</p>
        <button className="buy-button">Add to Cart</button>
        <div className="rating-section">
          <RatingWidget
            raters={filteredRatings.length}
            onRatingAdded={(rating) =>
              store.addRating({ product: slug, rating })
            }
          />
        </div>
      </div>
    </div>
  );
}
```

```
export const store = {
  data: [] as Rating[],
  addRating: ({ product, rating }: { product: string; rating: number }) => {
    store.data = [...store.data, { product, rating }];
    store.emit();
  },
  subscribers: new Set<Subscriber>(),
  subscribe: (subscriber: Subscriber) => {
    store.subscribers.add(subscriber);
    return () => store.subscribers.delete(subscriber);
  },
  emit: () => {
    for (const subscriber of store.subscribers) {
      subscriber();
    }
  },
};

export const useRatingsStore = () =>
  useSyncExternalStore(store.subscribe, () => store.data);
```

```
export function Component() {
  const { title, image, description, price, slug } = useLoaderData();
  const ratings = useRatingsStore();
  const filteredRatings = ratings.filter((r) => r.product === slug);
  return (
    <div className="product">
      <div className="product-image">
        <img src={image} alt={title} />
      </div>
      <div className="product-info">
        <span className="product-title">
          <h2>{title}</h2>
          <span>
            ({avg(filteredRatings.map((rating) => rating.rating)).toFixed(1)})
          </span>
        </span>
        <p className="price">${price}</p>
        <p className="description">{description}</p>
        <button className="buy-button">Add to Cart</button>
        <div className="rating-section">
          <RatingWidget
            raters={filteredRatings.length}
            onRatingAdded={(rating) =>
              store.addRating({ product: slug, rating })
            }
          />
        </div>
      </div>
    </div>
  );
}
```

```
export const store = {
  data: [] as Rating[],
  addRating: ({ product, rating }: { product: string; rating: number }) => {
    store.data = [...store.data, { product, rating }];
    store.emit();
  },
  subscribers: new Set<Subscriber>(),
  subscribe: (subscriber: Subscriber) => {
    store.subscribers.add(subscriber);
    return () => store.subscribers.delete(subscriber);
  },
  emit: () => {
    for (const subscriber of store.subscribers) {
      subscriber();
    }
  },
};

export const useRatingsStore = () =>
  useSyncExternalStore(store.subscribe, () => store.data);
```

```
export function Component() {
  const { title, image, description, price, slug } = useLoaderData();
  const ratings = useRatingsStore();
  const filteredRatings = ratings.filter((r) => r.product === slug);
  return (
    <div className="product">
      <div className="product-image">
        <img src={image} alt={title} />
      </div>
      <div className="product-info">
        <span className="product-title">
          <h2>{title}</h2>
          <span>
            ({avg(filteredRatings.map((rating) => rating.rating)).toFixed(1)})
          </span>
        </span>
        <p className="price">${price}</p>
        <p className="description">{description}</p>
        <button className="buy-button">Add to Cart</button>
        <div className="rating-section">
          <RatingWidget
            raters={filteredRatings.length}
            onRatingAdded={(rating) =>
              store.addRating({ product: slug, rating })
            }
          />
        </div>
      </div>
    </div>
  );
}
```

```
export const store = {
  data: [] as Rating[],
  addRating: ({ product, rating }: { product: string; rating: number }) => {
    store.data = [...store.data, { product, rating }];
    store.emit();
  },
  subscribers: new Set<Subscriber>(),
  subscribe: (subscriber: Subscriber) => {
    store.subscribers.add(subscriber);
    return () => store.subscribers.delete(subscriber);
  },
  emit: () => {
    for (const subscriber of store.subscribers) {
      subscriber();
    }
  },
};

export const useRatingsStore = () =>
  useSyncExternalStore(store.subscribe, () => store.data);
```

```
export function Component() {
  const { title, image, description, price, slug } = useLoaderData();
  const ratings = useRatingsStore();
  const filteredRatings = ratings.filter((r) => r.product === slug);
  return (
    <div className="product">
      <div className="product-image">
        <img src={image} alt={title} />
      </div>
      <div className="product-info">
        <span className="product-title">
          <h2>{title}</h2>
          <span>
            ({avg(filteredRatings.map((rating) => rating.rating)).toFixed(1)})
          </span>
        </span>
        <p className="price">${price}</p>
        <p className="description">{description}</p>
        <button className="buy-button">Add to Cart</button>
        <div className="rating-section">
          <RatingWidget
            raters={filteredRatings.length}
            onRatingAdded={(rating) =>
              store.addRating({ product: slug, rating })
            }
          />
        </div>
      </div>
    </div>
  );
}
```

```
export const store = {
  data: [] as Rating[],
  addRating: ({ product, rating }: { product: string; rating: number }) => {
    store.data = [...store.data, { product, rating }];
    store.emit();
  },
  subscribers: new Set<Subscriber>(),
  subscribe: (subscriber: Subscriber) => {
    store.subscribers.add(subscriber);
    return () => store.subscribers.delete(subscriber);
  },
  emit: () => {
    for (const subscriber of store.subscribers) {
      subscriber();
    }
  },
};

export const useRatingsStore = () =>
  useSyncExternalStore(store.subscribe, () => store.data);
```

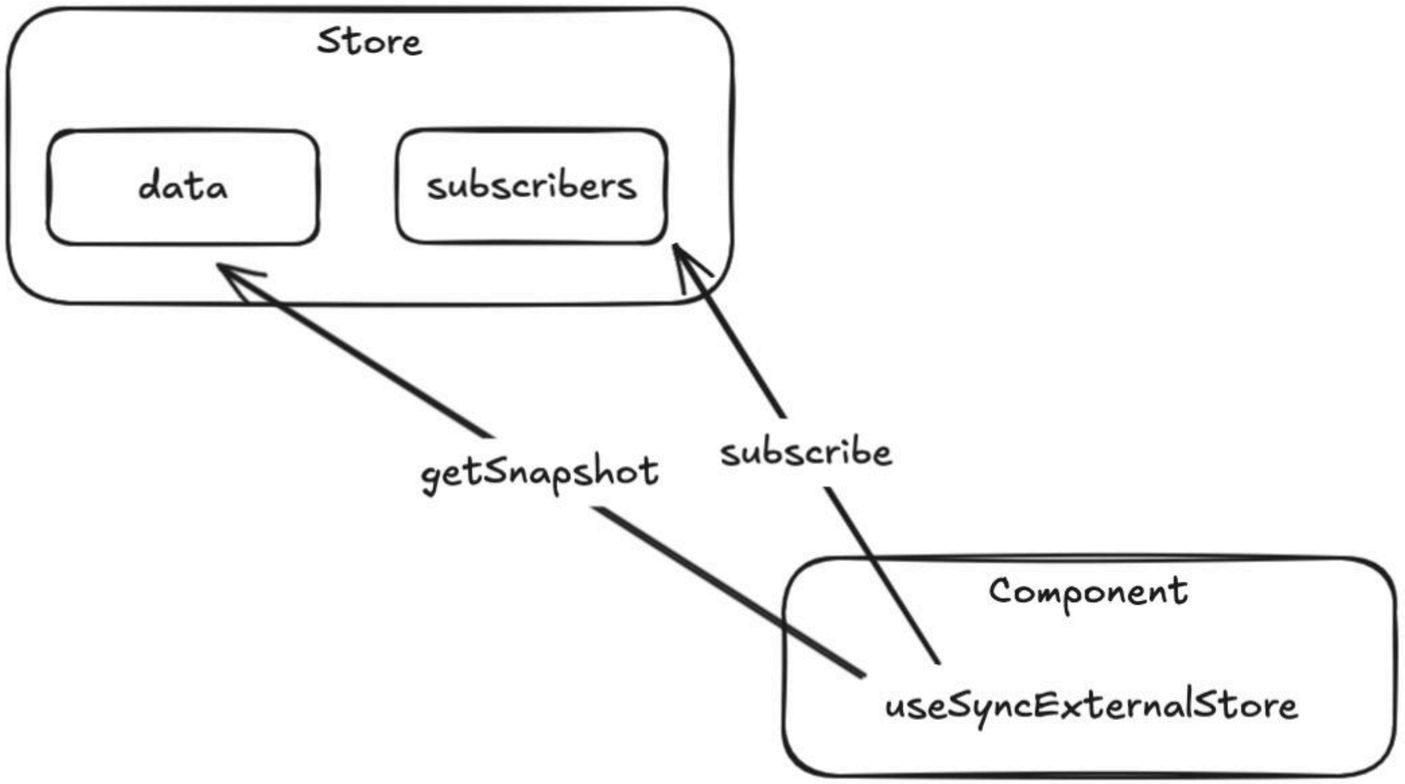
```
export function Component() {
  const { title, image, description, price, slug } = useLoaderData();
  const ratings = useRatingsStore();
  const filteredRatings = ratings.filter((r) => r.product === slug);
  return (
    <div className="product">
      <div className="product-image">
        <img src={image} alt={title} />
      </div>
      <div className="product-info">
        <span className="product-title">
          <h2>{title}</h2>
          <span>
            ({avg(filteredRatings.map((rating) => rating.rating)).toFixed(1)})
          </span>
        </span>
        <p className="price">${price}</p>
        <p className="description">{description}</p>
        <button className="buy-button">Add to Cart</button>
        <div className="rating-section">
          <RatingWidget
            raters={filteredRatings.length}
            onRatingAdded={(rating) =>
              store.addRating({ product: slug, rating })
            }
          />
        </div>
      </div>
    </div>
  );
}
```

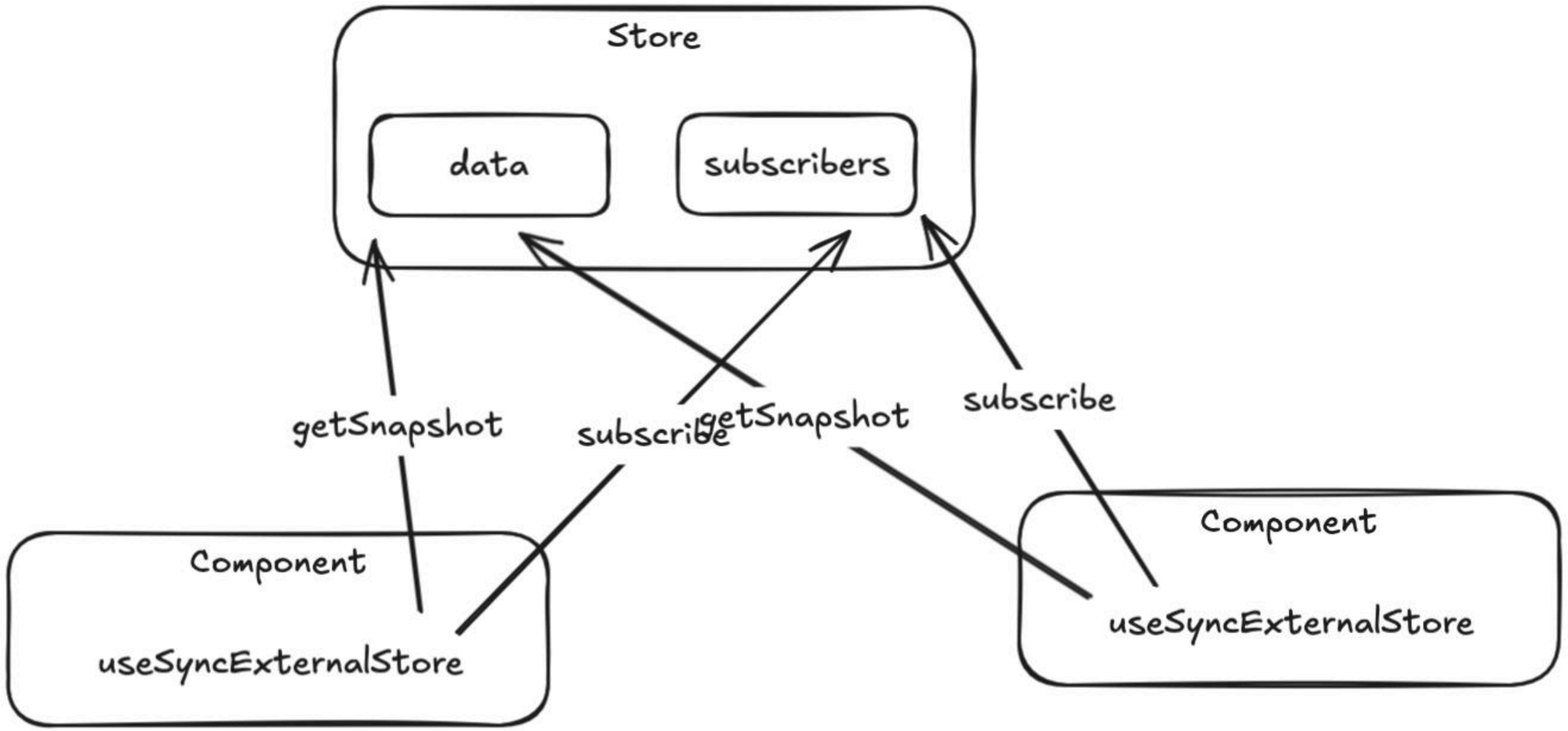
```
<script setup lang="ts">
import { store } from "../app/store";

function rate5Stars() {
  store.addRating({ product: getSlug(),rating: 5});
}

</script>

<template>
  <div class="container">
    <button @click="rate5Stars">
      <span>Rate 5 stars</span>
      <StarIcon />
    </button>
  </div>
</template>
```

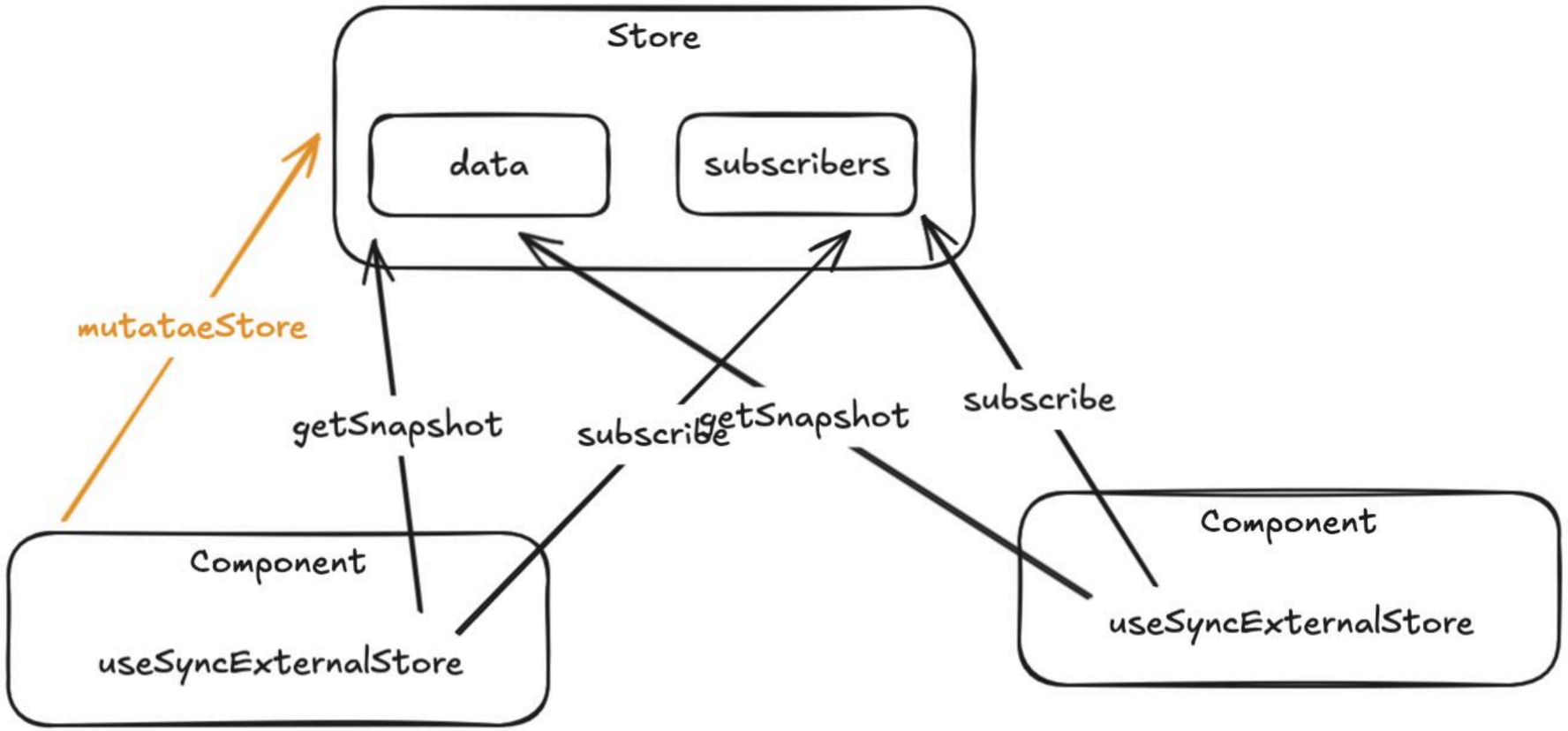
```
<script setup lang="ts">
import { store } from "../app/store";

function rate5Stars() {
  store.addRating({ product: getSlug(),rating: 5});
}

</script>

<template>
  <div class="container">
    <button @click="rate5Stars">
      <span>Rate 5 stars</span>
      <StarIcon />
    </button>
  </div>
</template>
```

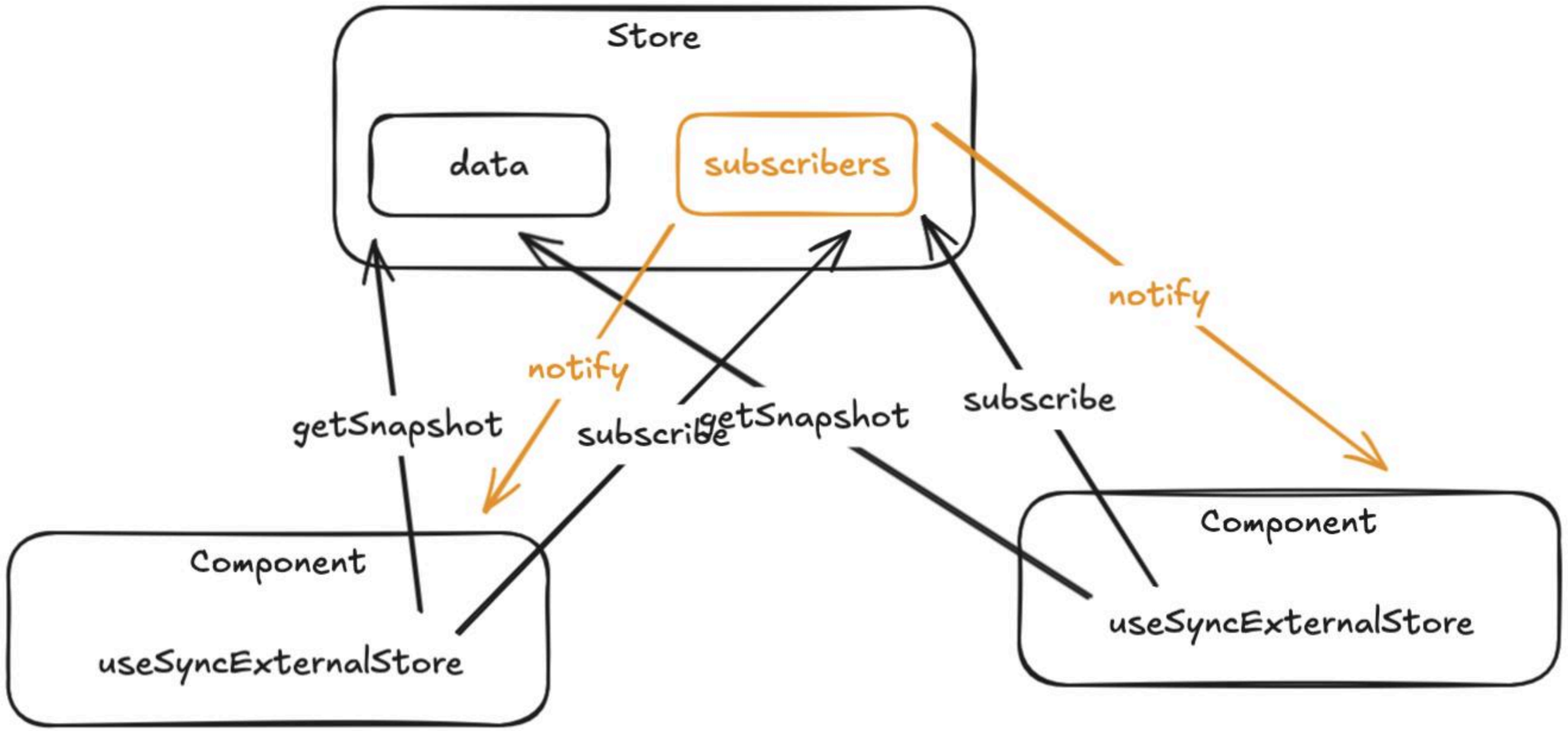
```
<script setup lang="ts">
import { store } from "../app/store";

function rate5Stars() {
  store.addRating({ product: getSlug(),rating: 5});
}

</script>

<template>
  <div class="container">
    <button @click="rate5Stars">
      <span>Rate 5 stars</span>
      <StarIcon />
    </button>
  </div>
</template>
```

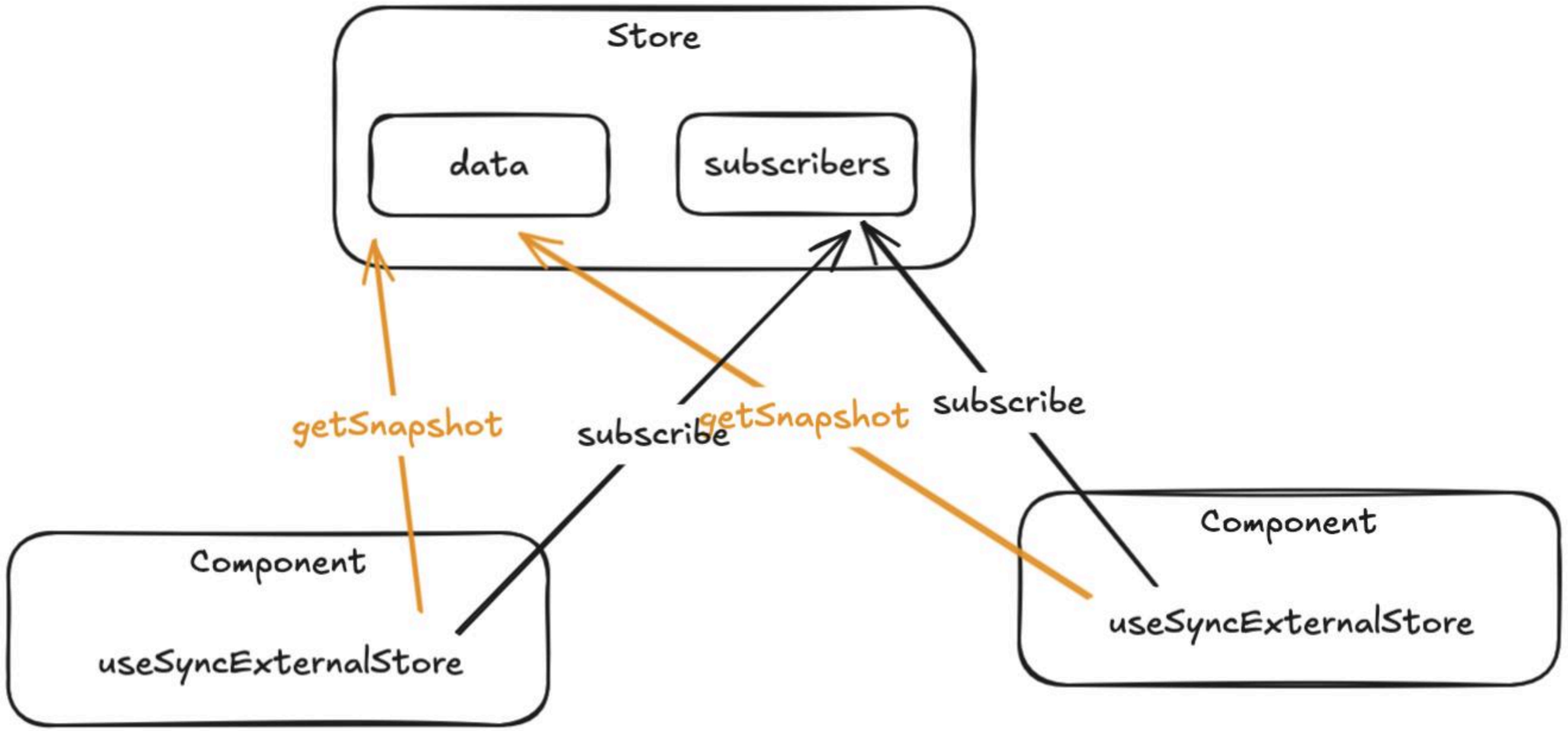
Let’s see it live


Store
data
subscribers
getSnapshot
subscribe
Component
useSyncExternalStore


Store
data
subscribers
getSnapshot
subscribe
getSnapshot
subscribe
Component
useSyncExternalStore
Component
useSyncExternalStore


Store
data
subscribers
mutataeStore
getSnapshot
subscribe
getSnapshot
subscribe
Component
useSyncExternalStore
Component
useSyncExternalStore


Store
data
subscribers
notify
notify
getSnapshot
subscribe
getSnapshot
subscribe
Component
useSyncExternalStore
Component
useSyncExternalStore


Store
data
subscribers
getSnapshot
subscribe
getSnapshot
subscribe
Component
useSyncExternalStore
Component
useSyncExternalStore

zustand / src / react.ts

Code | Blame | 64 lines (53 loc) · 1.69 KB

```
18      api: S,
19    ): ExtractState<S>
20
21    export function useStore<S extends ReadonlyStoreApi<unknown>, U>(
22      api: S,
23      selector: (state: ExtractState<S>) => U,
24    ): U
25
26    export function useStore<
27      api: ReadonlyStoreApi<T
28      selector: (state: TState)                    ice = identity as any,
29    ) {
30      const slice = React.useSyncExternalStore(
31        api.subscribe,
32        () => selector(api.getState()),
33        () => selector(api.getInitialState()),
34      )
35      React.useDebugValue(slice)
36      return slice
37    }
38
```

mobxjs / mobx

Type / to sea

Code | Issues 51 | Pull requests 15 | Discussions | Actions | Wiki | Security | Insights

09edf2b

mobx / packages / mobx-react-lite / src / useObserver.ts

dmitrytavern fix: unsynchronized useSyncExternalStore snapshots with server (#3830)

Code | Blame | 119 lines (105 loc) · 4.52 KB | Raw

```
import { Reaction } from "mobx"
import React from "react"
import { printDebugValue } from "./u
import { isUsingStaticRendering } fr
import { observerFinalizationRegistr                    verFinalizationRegistry"
import { useSyncExternalStore } from                    re/shim"

// Do not store `admRef` (even as pa                    is object,
// otherwise it will prevent GC and                    osal via FinalizationRegistry.
type ObserverAdministration = {
    reaction: Reaction | null // also serves as disposed flag
    onStoreChange: Function | null // also serves as mounted flag
    // stateVersion that 'ticks' for every time the reaction fires
    // tearing is still present,
    // because there is no cross component synchronization,
    // but we can use `useSyncExternalStore` API.
    // TODO: optimize to use number?
    stateVersion: any
    name: string
    // These don't depend on state/props, therefore we can keep them here instead of `useCallback`
    subscribe: Parameters<typeof React.useSyncExternalStore>[0]
    getSnapshot: Parameters<typeof React.useSyncExternalStore>[1]
}
```

s

Code | Blame | 64 lines (53 loc) · 1.69 KB

```
  api: S,
): ExtractState<S>

export function useStore<S extends ReadonlyStoreApi<unknown>, U>(
  api: S,
  selector: (state: ExtractState<S>)                    U,
): U

export function useStore<                    ice>(
  api: ReadonlyStoreApi<T
  selector: (state: TState)                    ice = identity as any,
) {
  const slice = React.useSyncExternalStore(
    api.subscribe,
    () => selector(api.getState()),
    () => selector(api.getInitialState()),
  )
  React.useDebugValue(slice)
  return slice
}
```

mobxjs / mobx

Type / to sea

Code | Issues 51 | Pull requests 15 | Discussions | Actions | Wiki | Security | Insights

09edf2b mobx / packages / mobx-react-lite / src / useObserver.ts

dmitrytavern fix: unsynchronized useSyncExternalStore snapshots with server (#3830)

Code | Blame | 119 lines (105 loc) · 4.52 KB | Raw

```
1   import { Reaction } from "mobx"
2   import React from "react"
3   import { printDebugValue } from "./u
4   import { isUsingStaticRendering } fr
5   import { observerFinalizationRegistr                    verFinalizationRegistry"
6   import { useSyncExternalStore } from                    re/shim"
7
8   // Do not store `admRef` (even as pa                    is object,
9   // otherwise it will prevent GC and                     osal via FinalizationRegistry.
10  type ObserverAdministration = {
11      reaction: Reaction | null // also serves as disposed flag
12      onStoreChange: Function | null // also serves as mounted flag
13      // stateVersion that 'ticks' for every time the reaction fires
14      // tearing is still present,
15      // because there is no cross component synchronization,
16      // but we can use `useSyncExternalStore` API.
17      // TODO: optimize to use number?
18      stateVersion: any
19      name: string
20      // These don't depend on state/props, therefore we can keep them here instead of `useCallback`
21      subscribe: Parameters<typeof React.useSyncExternalStore>[0]
22      getSnapshot: Parameters<typeof React.useSyncExternalStore>[1]
23  }
```

main query / packages / react-query / src / useBaseQuery.ts

Code | Blame | 165 lines (145 loc) · 4.89 KB

```
76
77      const [observer] = React.useState(
78        () =>
79          new Observer<TQueryFnData, TError, TData, TQueryData, TQueryKey>(
80            client,
81            defaultedOptions,
82          ),
83      )
84
85      // note: this must be                                   eSyncExternalStore
86      const result = observer.                             tResult(defaultedOptions)
87
88      const shouldSubscribe = !isRestoring && options.subscribed !== false
89      React.useSyncExternalStore(
90        React.useCallback(
91          (onStoreChange) => {
92            const unsubscribe = shouldSubscribe
93              ? observer.subscribe(notifyManager.batchCalls(onStoreChange))
94              : noop
95
96            // Update result to make sure we did not miss any query updates
```

Code | Blame | 64 lines (53 loc) · 1.69 KB

```
18      api: S,
19    ): ExtractState<S>
20
21    export function useStore<S extends ReadonlyStoreApi<unknown>, U>(
22      api: S,
23      selector: (state: ExtractState<S>) => U,
24    ): U
25
26    export function useStore<                                  ice>(
27      api: ReadonlyStoreApi<T
28      selector: (state: TState)                                ice = identity as any,
29    ) {
30      const slice = React.useSyncExternalStore(
31        api.subscribe,
32        () => selector(api.getState()),
33        () => selector(api.getInitialState()),
34      )
35      React.useDebugValue(slice)
36      return slice
37    }
38
```

mobxjs / mobx

Code | Issues 51 | Pull requests 15 | Discussions | Actions | Wiki | Security | Insights

09edf2b — mobx / packages / mobx-react-lite / src / useObserver.ts

dmitrytavern fix: unsynchronized useSyncExternalStore snapshots with server (#3830)

Code | Blame | 119 lines (105 loc) · 4.52 KB | Raw

```
import { Reaction } from "mobx"
import React from "react"
import { printDebugValue } from "./u
import { isUsingStaticRendering } fr
import { observerFinalizationRegistr                    verFinalizationRegistry"
import { useSyncExternalStore } from                    re/shim"

// Do not store `admRef` (even as pa                    is object,
// otherwise it will prevent GC and                     osal via FinalizationRegistry.
type ObserverAdministration = {
    reaction: Reaction | null // also serves as disposed flag
    onStoreChange: Function | null // also serves as mounted flag
    // stateVersion that 'ticks' for every time the reaction fires
    // tearing is still present,
```

main — query / packages / react-query / src / useBaseQuery.ts

Code | Blame | 165 lines (145 loc) · 4.89 KB

```
76
77      const [observer] = React.useState(
78        () =>
79          new Observer<TQueryFnData, TError, TData, TQueryData, TQueryKey>(
80            client,
81            defaultedOptions,
82          ),
83      )
84
85      // note: this must be c                    seSyncExternalStore
86      const result = observer                     Result(defaultedOptions)
87
88      const shouldSubscribe = !isRestoring && options.subscribed !== false
89      React.useSyncExternalStore(
90        React.useCallback(
91          (onStoreChange) => {
92            const unsubscribe = shouldSubscribe
93              ? observer.subscribe(notifyManager.batchCalls(onStoreChange))
94              : noop
95
```

main — query / packages / react-query / src / useBaseQuery.ts

Code | Blame | 165 lines (145 loc) · 4.89 KB

```
76
77      const [observer] = React.useState(
78        () =>
79          new Observer<TQueryFnData, TError, TData, TQueryData, TQueryKey>(
80            client,
81            defaultedOptions,
82          ),
83      )
84
85      // note: this must be                    eSyncExternalStore
86      const result = observer.                    tResult(defaultedOptions)
87
88      const shouldSubscribe = !isRestoring && options.subscribed !== false
89      React.useSyncExternalStore(
90        React.useCallback(
91          (onStoreChange) => {
92            const unsubscribe = shouldSubscribe
93              ? observer.subscribe(notifyManager.batchCalls(onStoreChange))
94              : noop
95
```


Code | Blame | 64 lines (53 loc) · 1.69 KB

```
18      api: S,
19    ): ExtractState<S>
20
21    export function useStore<S extends ReadonlyStoreApi<unknown>, U>(
22      api: S,
23      selector: (state: ExtractState<S>) => U,
24    ): U
25
26    export function useStore<                    ice>(
27      api: ReadonlyStoreApi<T
28      selector: (state: TState)                    ice = identity as any,
29    ) {
30      const slice = React.useSyncExternalStore(
31        api.subscribe,
32        () => selector(api.getState()),
33        () => selector(api.getInitialState()),
34      )
35      React.useDebugValue(slice)
36      return slice
37    }
38
```

Everyone does this!

mobxjs / mobx

Code | Issues 51 | Pull requests 15 | Discussions | Actions | Wiki | Security | Insights

09edf2b — mobx / packages / mobx-react-lite / src / useObserver.ts

dmitrytavern fix: unsynchronized useSyncExternalStore snapshots with server (#3830)

Code | Blame | 119 lines (105 loc) · 4.52 KB | Raw

```
import { Reaction } from "mobx"
import React from "react"
import { printDebugValue } from "./u
import { isUsingStaticRendering } fr
import { observerFinalizationRegistr                    verFinalizationRegistry"
import { useSyncExternalStore } from                    re/shim"

// Do not store `admRef` (even as pa                    is object,
// otherwise it will prevent GC and                     osal via FinalizationRegistry.
type ObserverAdministration = {
    reaction: Reaction | null // also serves as disposed flag
    onStoreChange: Function | null // also serves as mounted flag
    // stateVersion that 'ticks' for every time the reaction fires
    // tearing is still present,
```

main — query / packages / react-query / src / useBaseQuery.ts

Code | Blame | 165 lines (145 loc) · 4.89 KB

```
76
77      const [observer] = React.useState(
78        () =>
79          new Observer<TQueryFnData, TError, TData, TQueryData, TQueryKey>(
80            client,
81            defaultedOptions,
82          ),
83      )
84
85      // note: this must be                    eSyncExternalStore
86      const result = observer.                  tResult(defaultedOptions)
87
88      const shouldSubscribe = !isRestoring && options.subscribed !== false
89      React.useSyncExternalStore(
90        React.useCallback(
91          (onStoreChange) => {
92            const unsubscribe = shouldSubscribe
93              ? observer.subscribe(notifyManager.batchCalls(onStoreChange))
94              : noop
```

main — query / packages / react-query / src / useBaseQuery.ts

Code | Blame | 165 lines (145 loc) · KB

```
76
77      const [observer] = React.u State(
78        () =>
79          new Observer<TQueryFnData, TError, TData, TQueryData, TQueryKey>(
80            client,
81            defaultedOptions,
82          ),
83      )
84
85      // note: this must be c                  seSyncExternalStore
86      const result = observer                   Result(defaultedOptions)
87
88      const shouldSubscribe = !isRestoring && options.subscribed !== false
89      React.useSyncExternalStore(
90        React.useCallback(
91          (onStoreChange) => {
92            const unsubscribe = shouldSubscribe
93              ? observer.subscribe(notifyManager.batchCalls(onStoreChange))
94              : noop
95
```

Code | Blame | 64 lines (53 loc) · 1.69 KB

```
18      api: S,
19    ): ExtractState<S>
20
21    export function useStore<S extends ReadonlyStoreApi<unknown>, U>(
22      api: S,
23      selector: (state: ExtractState<S>) => U,
24    ): U
25
26    export function useStore<                    ice>(
27      api: ReadonlyStoreApi<T
28      selector: (state: TState)                  ice = identity as any,
29    ) {
30      const slice = React.useSyncExternalStore(
31        api.subscribe,
32        () => selector(api.getState()),
33        () => selector(api.getInitialState()),
34      )
35      React.useDebugValue(slice)
36      return slice
37    }
38
```

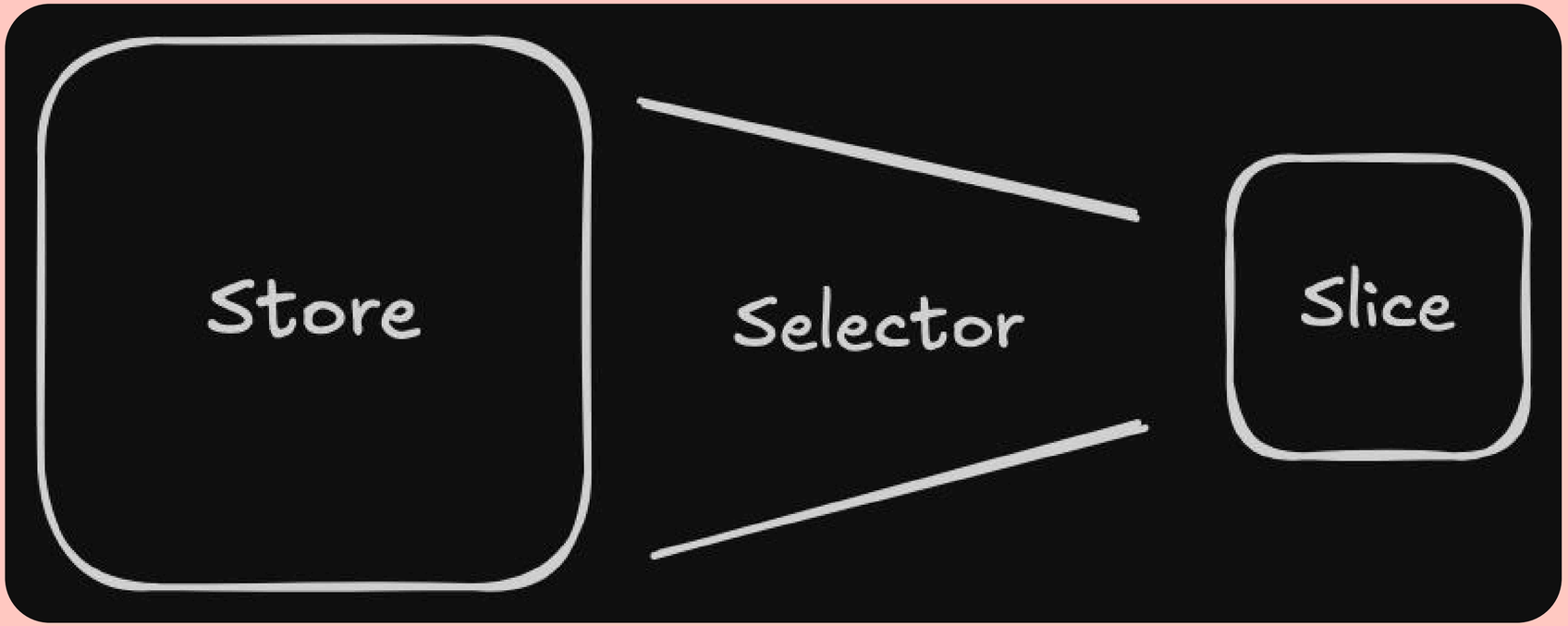
Performance

Performance

```
getSnapshot: A function that returns a snapshot of the data in the store that's needed by the component.
While the store has not changed, repeated calls to getSnapshot must return the same value.
If the store changes and the returned value is different (as compared by Object.is), React re-renders the component.
```

```
getSnapshot: A function that returns a snapshot of the data in the store that's needed by the component.
While the store has not changed, repeated calls to getSnapshot must return the same value.
If the store changes and the returned value is different (as compared by Object.is), React re-renders the component.
```

Fine grained snapshot


Store
Selector
Slice

DEMO TIME

```
export const useRatingsStore = (selector) => {
  return useSyncExternalStore(
    store.subscribe,
    () => selector(store.data));
};
```

```
export function RatingWidget({ onRatingAdded }) {
  const { slug } = useParams();
  const ratersCount = useRatingsStore(
    (ratings) => ratings.filter((rating) => rating.product === slug).length
  );
  return (
    <div className="rating-container">
      <h1>Join {ratersCount} raters!</h1>
      <RatingWidgetForm onSubmit={onRatingAdded} />
    </div>
  );
}
```

```
export const useRatingsStore = (selector) => {
  return useSyncExternalStore(
    store.subscribe,
    () => selector(store.data));
};
```

```
export function RatingWidget({ onRatingAdded }) {
  const { slug } = useParams();
  const ratersCount = useRatingsStore(
    (ratings) => ratings.filter((rating) => rating.product === slug).length
  );
  return (
    <div className="rating-container">
      <h1>Join {ratersCount} raters!</h1>
      <RatingWidgetForm onSubmit={onRatingAdded} />
    </div>
  );
}
```

```
export const useRatingsStore = (selector) => {
  return useSyncExternalStore(
    store.subscribe,
    () => selector(store.data));
};
```

```
export function RatingWidget({ onRatingAdded }) {
  const { slug } = useParams();
  const ratersCount = useRatingsStore(
    (ratings) => ratings.filter((rating) => rating.product === slug).length
  );
  return (
    <div className="rating-container">
      <h1>Join {ratersCount} raters!</h1>
      <RatingWidgetForm onSubmit={onRatingAdded} />
    </div>
  );
}
```

Conclusion

React's state management is tied to the component’s tree

Stores are plain Javascript objects and implement an interface to connect with and update React components.

Stores can be used to hold application state outside of the React components tree.

Stores can solve many problems that traditional state and contexts can’t

Questions?

Thank you!

Thank you!

@SlutskyTom

tom-slutsky