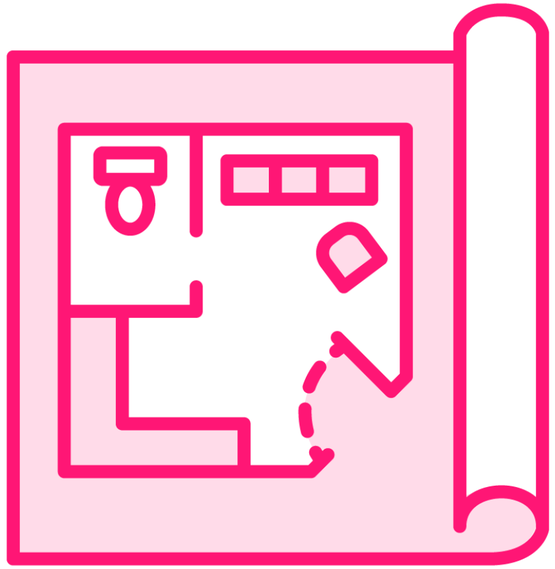
Optimizing and Refactoring Components

Kamran Ayub

Helping React developers create snappy apps

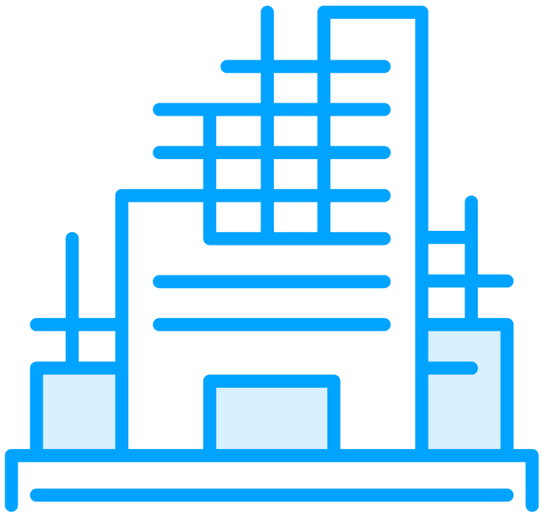
@kamranayub | www.kamranayub.com

Effectiveness vs. Efficiency



Effectiveness

Build the *right thing*



Efficiency

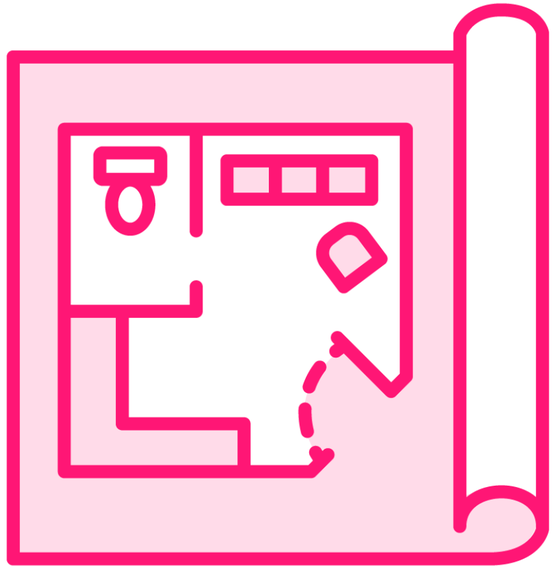
Build the *thing right*

Build the Right Thing:

React Architecture

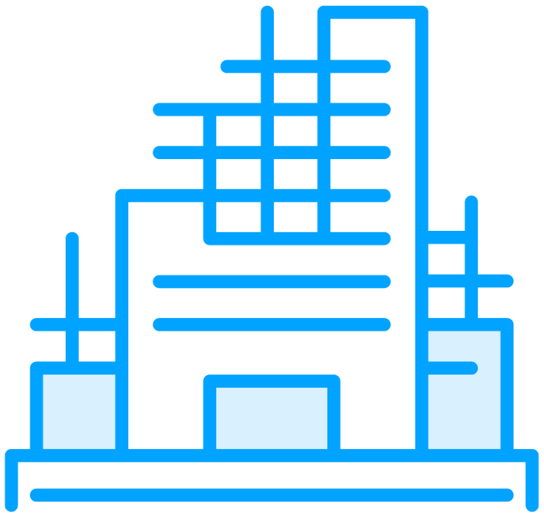
Cory House

Effectiveness vs. Efficiency



Effectiveness

Build the *right thing*



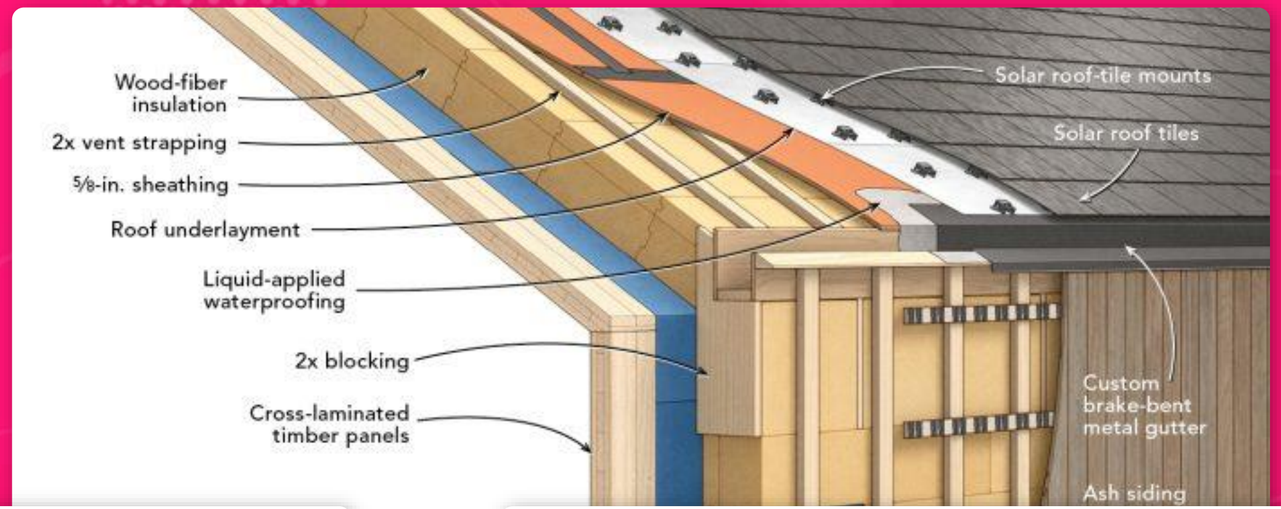
Efficiency

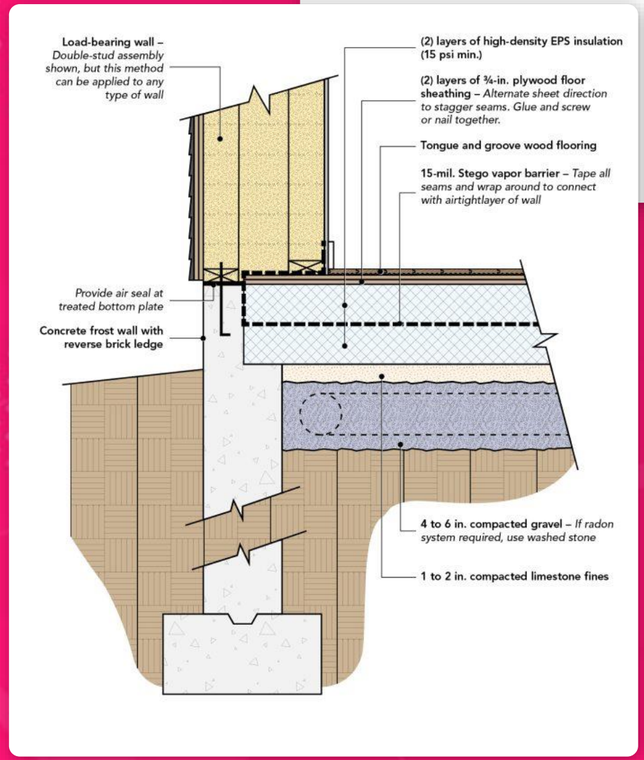
Build the *thing right*

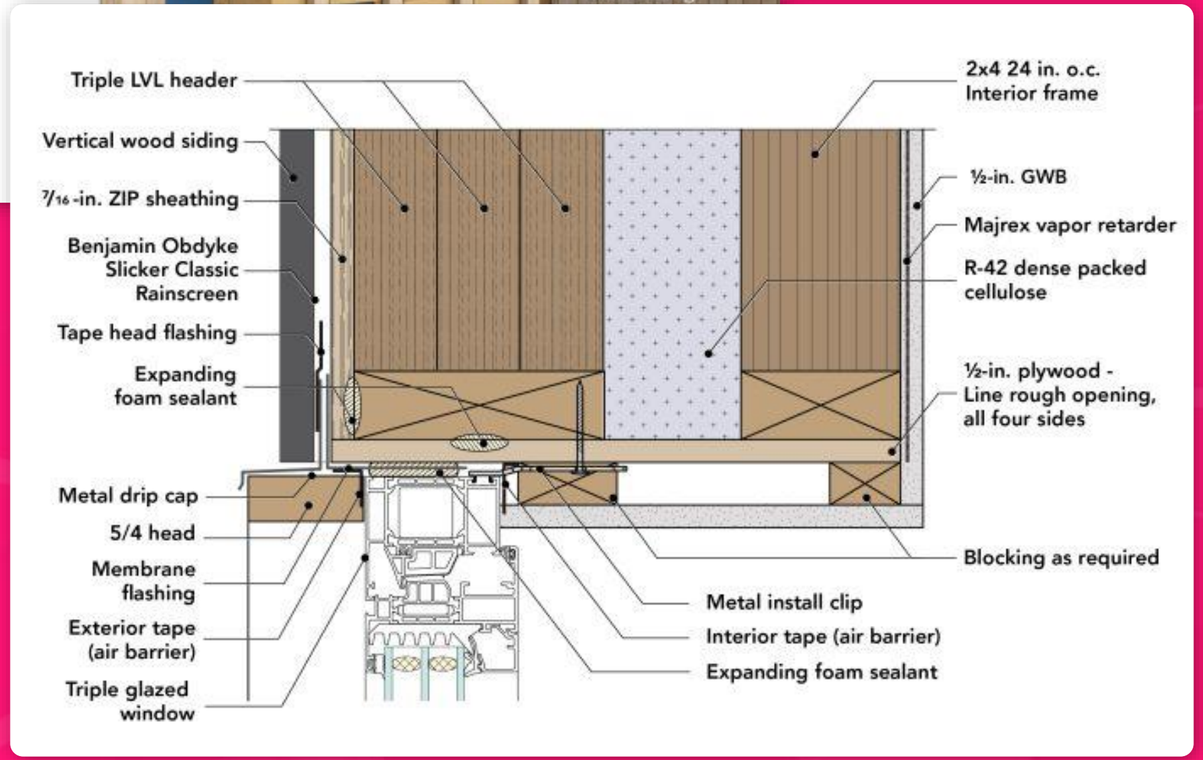
Dr. Russell Ackoff

"Building the right thing wrong is better than building the wrong thing right."

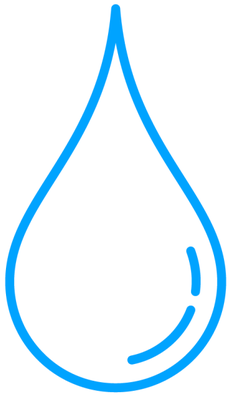
Stopping the World Rewrites


Wood-fiber insulation
2x vent strapping
5/8-in. sheathing
Roof underlayment
Liquid-applied waterproofing
2x blocking
Cross-laminated timber panels
Solar roof-tile mounts
Solar roof tiles
Custom brake-bent metal gutter
Ash siding


Load-bearing wall – Double-stud assembly shown, but this method can be applied to any type of wall
(2) layers of high-density EPS insulation (15 psi min.)
(2) layers of ¾-in. plywood floor sheathing – Alternate sheet direction to stagger seams. Glue and screw or nail together.
Tongue and groove wood flooring
15-mil. Stego vapor barrier – Tape all seams and wrap around to connect with airtightlayer of wall
Provide air seal at treated bottom plate
Concrete frost wall with reverse brick ledge
4 to 6 in. compacted gravel – If radon system required, use washed stone
1 to 2 in. compacted limestone fines

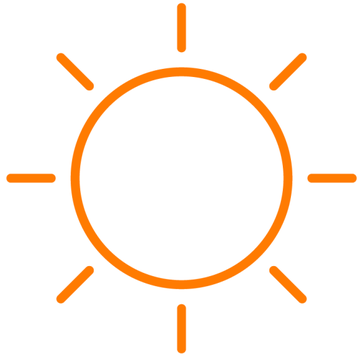

Triple LVL header
Vertical wood siding
7/16-in. ZIP sheathing
Benjamin Obdyke Slicker Classic Rainscreen
Tape head flashing
Expanding foam sealant
Metal drip cap
5/4 head
Membrane flashing
Exterior tape (air barrier)
Triple glazed window
2x4 24 in. o.c. Interior frame
½-in. GWB
Majrex vapor retarder
R-42 dense packed cellulose
½-in. plywood - Line rough opening, all four sides
Blocking as required
Metal install clip
Interior tape (air barrier)
Expanding foam sealant

High Performance Building Science



Water and Vapor Control

Air and Temperature Control



Energy Efficiency and Comfort

Managing Carbon Footprint

Efficiency in React components is less about speed and more about ways to render less often

Principal Objective of Lean Component Design

Reduce the amount and time spent doing unnecessary renders

Remove commits

Less time spent rendering

Shallower trees

Less time spent reconciling changes

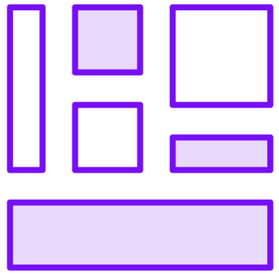
How Do You Design a Lean Component?

Lean Component	VS	Bloated Component
Re-renders less often		Causes wasted rendering
Manages state efficiently		Manages state inefficiently
Has a shallow DOM tree		Has a deep DOM tree
Less data coupling		Tight data coupling
Easier to reason about		Harder to reason about
Promotes reusability		Restricts usage

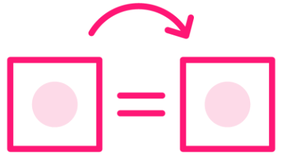
Lean Component Design is for Efficiency

Less re-rendering

Less reconciliation



More flexibility



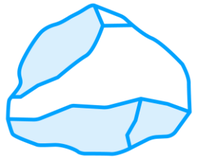
More reuse

Less resources

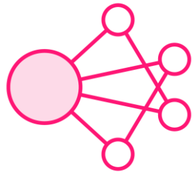
Less cognitive load

Inspiration for Lean Component Design

Clean Code



SOLID Principles



Domain-Driven Design

Systems Thinking

Component Design Scorecard

Pattern or Practice

Brief explanation

Reduce Wasted Renders

Quantitative

- Less re-rendering
- Less reconciliation
- Smaller footprint
- Less memory usage

Qualitative

- More flexibility
- Promotes reuse
- Less cognitive load

Promotes | Detracts | Depends | Neither

React Performance Scorecard
Find more at bit.ly/PSReactPerfScorecard

The Hidden Cost of DOM Reconciliation

Hidden Cost of Reconciliation

Trigger

- Run diffing algorithm
  - Initial render
  - State has been updated
- Add components to render queue

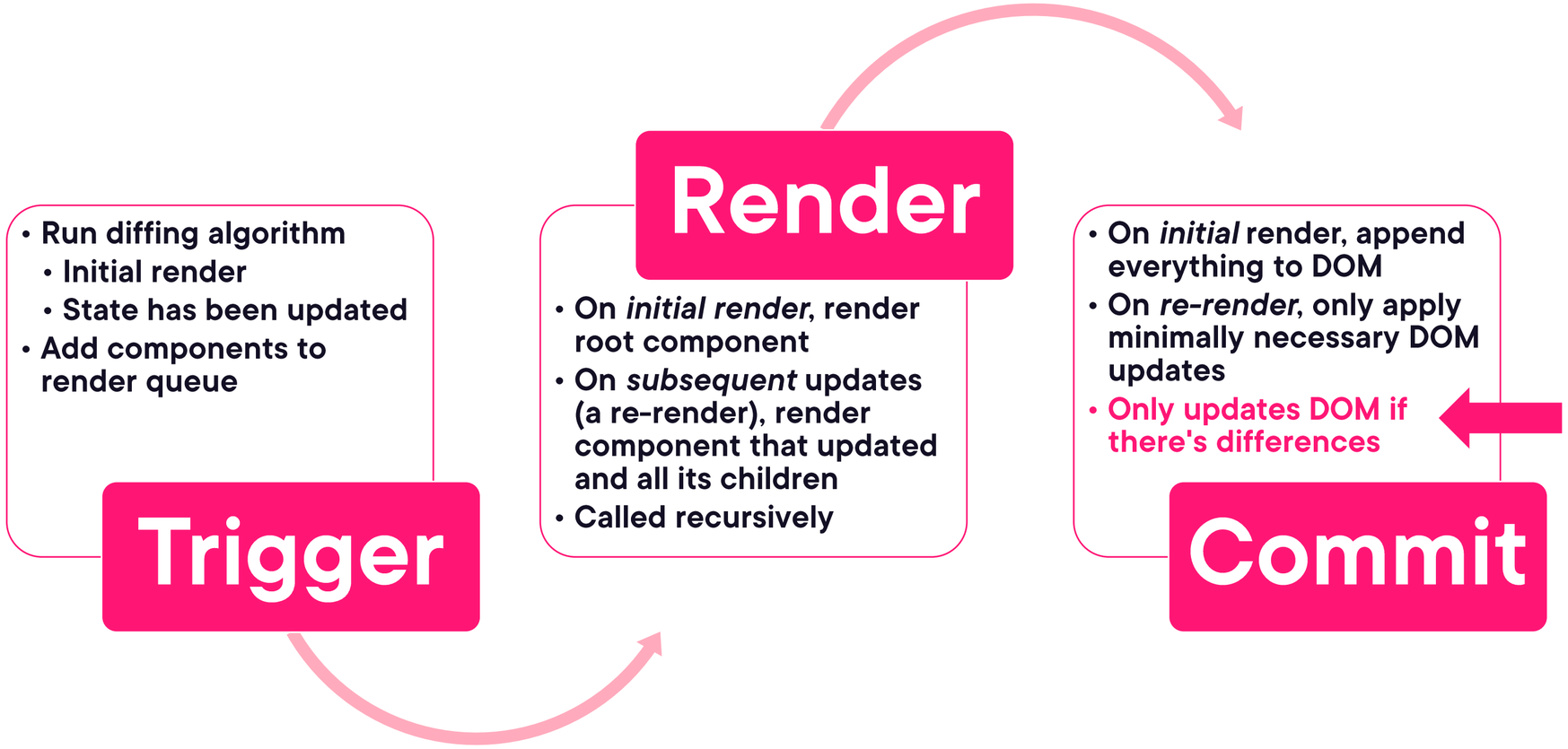
Render

- On *initial render*, render root component
- On *subsequent* updates (a re-render), render component that updated and all its children
- Called recursively

Commit

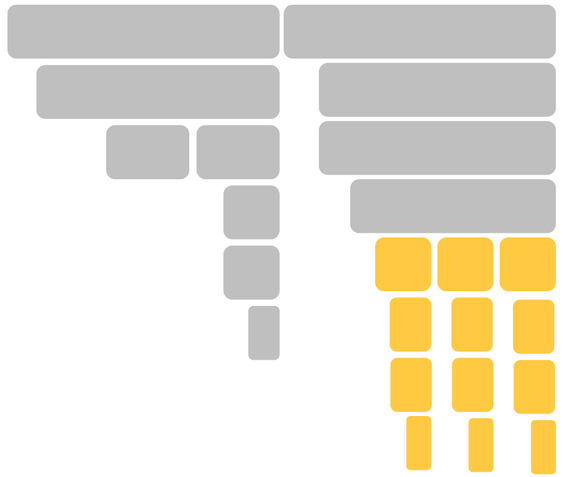
- On *initial* render, append everything to DOM
- On *re-render*, only apply minimally necessary DOM updates
- Only updates DOM if there's differences

Hidden Cost of Reconciliation

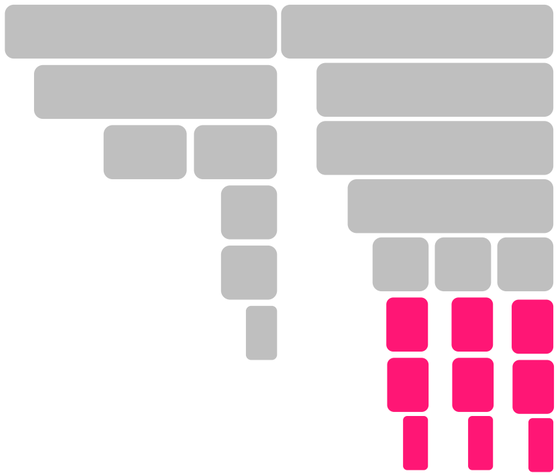


How Reconciliation Works

Virtual DOM



Actual DOM



Principal Objective of Lean Component Design

Reduce the amount and time spent doing unnecessary renders

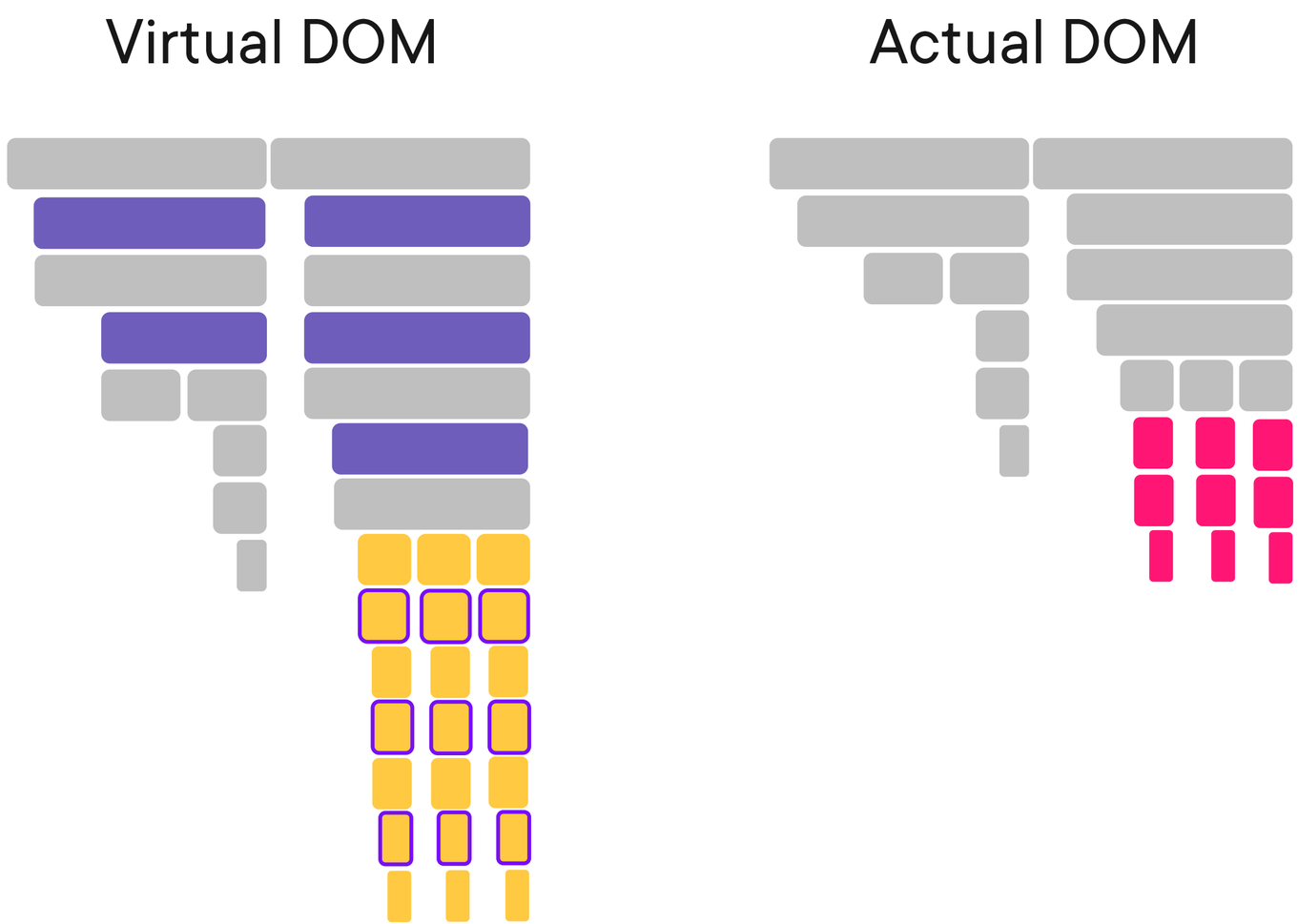
Remove commits

Less time spent rendering

Shallower trees

Less time spent reconciling changes

Expensive Reconciliation with CSS-in-JS



styled-components Example Screenshot

Component Design Scorecard

Pattern or Practice

Brief explanation

Reduce Wasted Renders

Quantitative

- Less re-rendering
- Less reconciliation
- Smaller footprint
- Less memory usage

Qualitative

- More flexibility
- Promotes reuse
- Less cognitive load

Promotes | Detracts | Depends | Neither

React Performance Scorecard
Find more at bit.ly/PSReactPerfScorecard

Optimizing the Family Tree Using Slots

React Family Tree

Owners

Parents

Children

React Family Tree

Owners

Parents

Children

App

- LeftNavigation
- Header
- Content

```
App
  LeftNavigation
  Header
    SearchBar
  Content
    ProductSpotlight
    LatestProducts
    Promotions
```

- App
  - LeftNavigation
  - Header
    - SearchBar
  - Content
    - ProductSpotlight
    - LatestProducts
    - Promotions

```
const App = () => (
  <>
    <LeftNavigation />
    <Header />
    <Content />
  </>
);
```

- App
  - LeftNavigation
  - Header
    - SearchBar
  - Content
    - ProductSpotlight
    - LatestProducts
    - Promotions

```
const App = () => (
  <>
    <LeftNavigation />
    <Header>
      <SearchBar />
    </Header>
    <Content>
      <ProductSpotlight />
      <LatestProducts />
      <Promotions />
    </Content>
  </>
);
```

```
const App = () => (
  <>
    <LeftNavigation />
    <Header />
    <Content />
  </>
);
```

```
const App = () => (
  <>
    <LeftNavigation />
    <Header>
      <SearchBar />
    </Header>
    <Content>
      <ProductSpotlight />
      <LatestProducts />
      <Promotions />
    </Content>
  </>
);
```

App

- LeftNavigation
- Header
- Content

```
const App = () => (
  <>
    <LeftNavigation />
    <Header />
    <Content />
  </>
);
```

Owner Hierarchy

Component Tree

App

- LeftNavigation
- Header
- Content

Owner Hierarchy

```
const App = () => (
  <>
    <LeftNavigation />
    <Header />
    <Content />
  </>
);
```

Component Tree

App

- LeftNavigation
- Header
  - SearchBar
- Content
  - ProductSpotlight
  - LatestProducts
  - Promotions

Parent Hierarchy

App

- LeftNavigation
- Header
- Content

Owner Hierarchy

Who "owns" rendering

```
const App = () => (
  <>
    <LeftNavigation />
    <Header />
    <Content />
  </>
);
```

Component Tree

App

- LeftNavigation
- Header
  - SearchBar
- Content
  - ProductSpotlight
  - LatestProducts
  - Promotions

Parent Hierarchy

App

- LeftNavigation
- Header
  - SearchBar
- Content
  - ProductSpotlight
  - LatestProducts
  - Promotions

Owner Hierarchy

```
const App = () => (
  <>
    <LeftNavigation />
    <Header>
      <SearchBar />
    </Header>
    <Content>
      <ProductSpotlight />
      <LatestProducts />
      <Promotions />
    </Content>
  </>
);
```

A child is re-rendered by its *owner*, not its parent

App

- LeftNavigation
- Header
- Content

```
const App = () => (
  <>
    <LeftNavigation />
    <Header />
    <Content />
  </>
);
```

Owner Hierarchy

Component Tree

```
App
  LeftNavigation
  Header
    SearchBar
  Content
```

Owner Hierarchy

```
const App = () => (
  <>
    <LeftNavigation />
    <Header>
      <SearchBar />
    </Header>
    <Content />
  </>
);
```

Component Tree

```
App
  LeftNavigation
  Header
    SearchBar
  Content
```

Owner Hierarchy

```
const App = () => (
  <>
    <LeftNavigation />
    <Header>
      <SearchBar showSuggestions={false} />
    </Header>
    <Content />
  </>
);
```

Component Tree

Component Design Scorecard

Component Slots

Using props to substitute nested component trees and change rendering ownership

Reduce Wasted Renders

Quantitative

- Less re-rendering ✓
- Less reconciliation ⊖
- Smaller footprint ✓
- Less memory usage ✓

Qualitative

- More flexibility ✓
- Promotes reuse ✓
- Less cognitive load ✓

React Performance Scorecard
Find more at bit.ly/PSReactPerfScorecard

✓ Promotes ✗ Detracts ? Depends ⊖ Neither

The Children Prop is the Default Slot

Nested JSX is passed through as the children prop

Header.jsx

```
const Header = ({
  children
}) => {
  return (
    <header>
      <SearchBar
        showSuggestions={false}
      />
    </header>
  )
}
```

Contains nested components (if any)

Passed as the children prop to the <header> element

Single Slot Using Children Prop

You can use the children prop to "slot" in nested trees within containers

Header.jsx

```
const Header = ({
  children
}) => {
  return (
    <header>
      {children}
    </header>
  )
}
```

Changes ownership to Header's parent

Multiple Component Slots

You can also use multiple props as slots

Header.jsx

```
const Header = ({
  slotMenu,
  slotSearch
}) => {
  return (
    <header>
      <nav>
        {slotMenu}
      </nav>
      <section className="search">
        {slotSearch}
      </section>
    </header>
  )
}
```

Slot Memoization

Slots are JSX complex objects subject to the same performance implications

App.jsx

```
const App = () => (
  <>
    <LeftNavigation />
    <Header
      slotMenu={<MainMenu />}
      slotSearch={<SearchBar showSuggestions={false} />}
    />
    <Content>
      <ProductSpotlight />
      <LatestProducts />
      <Promotions />
    </Content>
  </>
);
```

JSX objects

Slot Memoization

Slots are JSX complex objects subject to the same performance implications

App.jsx

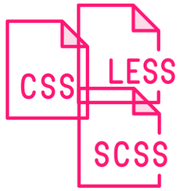
```
const App = () => {
  const [showSuggestions, setShowSuggestions] = useState(false);

  const slotMenu = useMemo(() => <MainMenu />, []);
  const slotSearch = useMemo(
    () => <SearchBar showSuggestions={showSuggestions} />,
    [showSuggestions]
  );

  return (
    <>
      <LeftNavigation />
      <Header slotMenu={slotMenu} slotSearch={slotSearch} />
      <Content>
        <ProductSpotlight />
        <LatestProducts />
        <Promotions />
      </Content>
    </>
  );
};
```

Choosing an Efficient Styling Approach

Styling Options in React

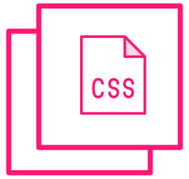
Inline style objects

CSS-in-JS



External styles

Atomic CSS



CSS Modules

CSS Frameworks

Inline Style Objects

Passing CSS using JS through the style prop on components

Card.jsx

```
const Card = ({ title, body, footer }) => {
  return (
    <section style={{
      marginLeft: 20,
      marginRight: 20,
      backgroundColor: "#eee"
    }}>
      <header>{title}</header>
      <section>{body}</section>
      <footer>{footer}</footer>
    </section>
  )
}
```

Component Design Scorecard

Inline Style Objects

Passing styles via the `style` prop and inline or const objects

Reduce Wasted Renders

Quantitative

- Less re-rendering
- Less reconciliation
- Smaller footprint
- Less memory usage

Qualitative

- More flexibility
- Promotes reuse
- Less cognitive load

Promotes | Detracts | Depends | Neither

React Performance Scorecard
Find more at bit.ly/PSReactPerfScorecard

External Styles

Importing external CSS stylesheets as standalone files for the page

Card.jsx

```
import './card.css';

const Card = ({ title, body, footer }) => {
  return (
    <card>
      <card-header>
        {title}
      </card-header>
      <card-body>
        {body}
      </section>
      <card-footer>
        {footer}
      </card-footer>
    </card>
  )
}
```

External Styles

Importing external CSS stylesheets as standalone files for the page

Card.jsx

```
import './card.css'; // or scss, less

const Card = ({ title, body, footer }) => {
  return (
    <card>
      <card-header>
        {title}
      </card-header>
      <card-body>
        {body}
      </section>
      <card-footer>
        {footer}
      </card-footer>
    </card>
  )
}
```

External Styles

Importing external CSS stylesheets as standalone files for the page

Card.jsx

```
import './card.css'; // or scss, less

const Card = ({ title, body, footer }) => {
  return (
    <card>
      <card-header>
        {title}
      </card-header>
      <card-body>
        {body}
      </section>
      <card-footer>
        {footer}
      </card-footer>
    </card>
  )
}
```

card.css

```
card { /* styles */ }
card-header { /* styles */ }
card-body { /* styles */ }
card-footer { /* styles */ }
```

External Styles

Importing external CSS stylesheets as standalone files for the page

Card.jsx

```
import './card.css'; // or scss, less

const Card = ({ title, body, footer }) => {
  return (
    <card has-body={!!body}>
      <card-header>
        {title}
      </card-header>
      <card-body>
        {body}
      </section>
      <card-footer>
        {footer}
      </card-footer>
    </card>
  )
}
```

card.css

```
card { /* styles */ }
card[has-body="true"] { /* styles */ }

card-header { /* styles */ }
card-body { /* styles */ }
card-footer { /* styles */ }
```

Component Design Scorecard

External Styles

Importing or linking to external stylesheet CSS files, including using pre- or post-processors

Reduce Wasted Renders

Quantitative

- Less re-rendering ✓
- Less reconciliation ✓
- Smaller footprint ✗
- Less memory usage ✓

Qualitative

- More flexibility ✗
- Promotes reuse ✓
- Less cognitive load ✗

Promotes | Detracts | Depends | Neither

React Performance Scorecard
Find more at bit.ly/PSReactPerfScorecard

CSS Modules

Scoped CSS that is imported, transformed, and injected as static classnames

Card.jsx

```jsx
import styles from './card.module.css'

const Card = ({ title, body, footer }) => {
  return (
    <div className={styles.card}>
      <div className={styles.cardHeader}>
        {title}
      </div>
      <div className={styles.cardFooter}>
        {body}
      </div>
      <div
        className={
          footer ? styles.cardFooter
            : styles.cardFooter__empty
      }>
        {footer}
      </div>
    </div>
  )
}
```

card.module.css

```css
.card { /* styles */ }
.cardHeader { /* styles */ }
.cardBody { /* styles */ }
.cardFooter { /* styles */ }

.cardFooter__empty {
  composes: cardFooter;
  display: none;
}
```

CSS Modules

Scoped CSS that is imported, transformed, and injected as static classnames

Card.jsx

```jsx
import styles from './card.module.css'

const Card = ({ title, body, footer }) => {
  return (
    <div className={styles.card}>
      <div className={styles.cardHeader}>
        {title}
      </div>
      <div className={styles.cardFooter}>
        {body}
      </div>
      <div
        className={
          footer ? styles.cardFooter
            : styles.cardFooter__empty
      }>
        {footer}
      </div>
    </div>
  )
}
```

card.module.css

```css
.card { /* styles */ }
.cardHeader { /* styles */ }
.cardBody { /* styles */ }
.cardFooter { /* styles */ }

.cardFooter__empty {
  composes: cardFooter;
  display: none;
}
```

Component Design Scorecard

CSS Modules

Importing scoped CSS files that transform and hash class selectors to static classNames

Reduce Wasted Renders

Quantitative

- Less re-rendering ✓
- Less reconciliation ✓
- Smaller footprint ✓
- Less memory usage ✓

Qualitative

- More flexibility ✓
- Promotes reuse ✓
- Less cognitive load ✗

Promotes | Detracts | Depends | Neither

React Performance Scorecard
Find more at bit.ly/PSReactPerfScorecard

CSS-in-JS Example: styled-components

Create React components with computed class names and scoped (or global) styles

Card.jsx

```
import {
  CardWrapper, CardHeader, CardBody, CardFooter
} from './Card.styles.js'

const Card = ({ title, body, footer }) => {
  return (
    <CardWrapper>
      <CardHeader>
        {title}
      </CardHeader>
      <CardBody>
        {body}
      </CardBody>
      <CardFooter empty={!footer}>
        {footer}
      </CardFooter>
    </CardWrapper>
  )
}
```

Card.styles.js

```
import styled from 'styled-components';

export CardWrapper = styled.div`
  /* styles */
`;

export CardHeader = styled.header`
  /* styles */
`;

export CardFooter = styled.footer`
  ${props => props.empty &&  css`
    display: 'none' : 'block'
  `}
`;
```

CSS-in-JS Example: styled-jsx

Scoped (or global) styles with inline style tags and template literals

Card.jsx

```
const Card = ({ title, body, footer }) => {
  return (
    <card>
      <card-header>
        {title}
      </card-header>
      <card-body>
        {body}
      </>
      <card-footer empty={!footer}>
        {footer}
      </card-footer>
      <style jsx>{`
        /* scoped */

        card { /* styles */ }
        card-header { /* styles */ }
        card-footer[empty='true'] {
          display: none;
        }
      `}</style>
    </card>
  )
}
```

Component Design Scorecard

CSS-in-JS

Importing scoped CSS files that transform and hash class selectors to static classNames

Reduce Wasted Renders

Quantitative

- Less re-rendering
- Less reconciliation
- Smaller footprint
- Less memory usage

Qualitative

- More flexibility
- Promotes reuse
- Less cognitive load

Promotes | Detracts | Depends | Neither

React Performance Scorecard
Find more at bit.ly/PSReactPerfScorecard

Atomic CSS Example: Tailwind.css

Atomic styles that can be composed together without writing custom CSS

Card.jsx

```
const Card = ({ title, body, footer }) => {
  return (
    <card className="p-5 bg-grey-500 rounded-md">
      <card-header className="border-b-1 bg-purple-400 text-white">
        {title}
      </card-header>
      <card-body>
        {body}
      </section>
      <card-footer className="border-t-1 bg-grey-700">
        {footer}
      </card-footer>
    </card>
  )
}
```

Component Design Scorecard

Atomic CSS

Atomic classes that can be composed into complex component UIs

Reduce Wasted Renders

Quantitative

- Less re-rendering ✓
- Less reconciliation ✓
- Smaller footprint ✓ ++
- Less memory usage ✓

Qualitative

- More flexibility ✓
- Promotes reuse ✓
- Less cognitive load ✗

Promotes | Detracts | Depends | Neither

React Performance Scorecard
Find more at bit.ly/PSReactPerfScorecard

CSS Frameworks Example: MUI (Material UI for React)

Shared design systems or CSS frameworks that offer flexible styling approaches

ProductCard.jsx

```
import Card from "@mui/material/Card";
import CardHeader from "@mui/material/CardHeader";
import CardContent from "@mui/material/CardContent";
import CardMedia from "@mui/material/CardMedia";
import CardActions from "@mui/material/CardActions";
import Typography from "@mui/material/Typography";

const ProductCard = ({ title, desc, image }) => {
  return (
    <Card>
      <CardHeader title={title} />
      <CardMedia component="img" image={image} alt={title} />
      <CardContent>
        <Typography variant="body2" color="text.secondary">
          {desc}
        </Typography>
      </CardContent>
      <CardActions sx={{ width: "50%" }}>
        {/* Actions */}
      </CardActions>
    </Card>
  );
};
```

Component Design Scorecard

CSS Frameworks

Shared design systems or CSS frameworks that offer flexible styling approaches

Reduce Wasted Renders

Quantitative

- Less re-rendering ?
- Less reconciliation ?
- Smaller footprint ✗
- Less memory usage ?

Qualitative

- More flexibility ✓
- Promotes reuse ✓
- Less cognitive load ✓

Promotes | Detracts | Depends | Neither

React Performance Scorecard
Find more at bit.ly/PSReactPerfScorecard

Recommended Reading

CUBE CSS
https://piccalil.li/blog/cube-css/

The Evolution of Scalable CSS

https://andreipfeiffer.dev/blog/2022/scalable-css-evolution

Building a Scalable CSS Architecture with BEM and Utility Classes

https://css-tricks.com/building-a-scalable-css-architecture-with-bem-and-utility-classes/

Reducing Hook Re-Rendering with Shared State

Complex Hook Logic

Don't even bother reading this...

AddToCartButton.jsx

```
const AddToCartButton = ({ productId, variant, quantity: initialQty = 1 }) => {
  const { products } = useAppState();
  const product = getProductByVariant(products, productId, variant);

  const [maxQuantity, setMaxQuantity] = useState(null);
  const [quantity, setQuantity] = useState(initialQty);
  const [canIncreaseQty, setCanIncreaseQty] = useState(false);
  const [quantityError, setQuantityError] = useState(null);

  useEffect(() => {
    services
      .checkItemQuantityPolicy(productId, variant, quantity)
      .then((result) => {
        setMaxQuantity(result.maxItemQuantity);
        setCanIncreaseQty(quantity < result.maxItemQuantity);
        if (result.error) {
          setQuantityError(true);
        }
      });
  }, [productId, variant, quantity]);

  const handleOnQuantityIncrease = useCallback(
    (amount = 1) => {
      const requestedQuantity = quantity + amount;

      if (requestedQuantity < maxQuantity) {
        setQuantity(requestedQuantity);
        setQuantityError(false);
      } else {
        setQuantityError(true);
      }
    },
    [quantity, maxQuantity]
  );

  // Rest of A2C logic and rendering
  // ...
};
```

Encapsulating Complex Hook Logic

Extracting logic from within components to be within custom hooks

AddToCartButton.jsx

```
const AddToCartButton = ({ productId, variant, quantity: initialQuantity }) => {
    const product = useProduct(productId, { variant });

    const { quantity, increaseQty, error } =
      useProductQuantityManager(product, { initialQuantity });

    const handleOnQuantityIncrease = useCallback((amount = 1) => {
      increaseQty(amount);
    }, [increaseQty]);


    // Rest of A2C logic and rendering
    // ...
  }
```

Component Design Scorecard

Custom Hooks

Reducing hook usage in components by encapsulating them into custom hooks

Reduce Wasted Renders

Quantitative

- Less re-rendering ✗
- Less reconciliation ✗
- Smaller footprint ✗
- Less memory usage ✗

Qualitative

- More flexibility ✓
- Promotes reuse ✓
- Less cognitive load ✓

Promotes | Detracts | Depends | Neither

React Performance Scorecard
Find more at bit.ly/PSReactPerfScorecard

Hook Factory

Used to encapsulate behavior and shared state or logic

createSimpleStateHook.js

```
const createSimpleStateHook = (initialValue) => {

  /* creates a closure */

  return function useSimpleState() {
    const [value, setValue] = useState(initialValue);

    /* custom logic that is always shared,
       and has access to the closure */

    return [value, setValue];
  };
};
```

Hook Factory

Used to encapsulate behavior and shared state or logic

createSimpleStateHook.js

```
const createSimpleStateHook = (initialValue) => {

  /* creates a closure */

  return function useSimpleState() {
    const [value, setValue] = useState(initialValue);

    /* custom logic that is always shared,
       and has access to the closure */

    return [value, setValue];
  };
};
```

Counter.js

```
const useStringState = createSimpleStateHook('');
const useNumberState = createSimpleStateHook(0);

const StringComponent = () => {
  const [text, setText] = useStringState();
  return (
    <div>
      <input type="text" value={text} onChange={e => setText(e.target.value)} />
      <p>Text: {text}</p>
    </div>
  );
};

const NumberComponent = () => {
  const [number, setNumber] = useNumberState();
  return (
    <div>
      <input type="number" value={number} onChange={e => setNumber(+e.target.value)} />
      <p>Number: {number}</p>
    </div>
  );
};
```

State Factory Example

Used to encapsulate behavior and shared state or logic

createSharedStateHook.js

```
const createSharedStateHook = (initialValue) => {
  let sharedState = initialValue;
  const listeners = new Set();

  const useSharedState = () => {
    const [, setState] = useState(sharedState);

    useEffect(() => {
      const listener = () => setState(sharedState);
      listeners.add(listener);

      return () => {
        listeners.delete(listener);
      };
    }, []);

    const setSharedState = (newState) => {
      sharedState = newState;
      listeners.forEach(listener => listener());
    };

    return [sharedState, setSharedState];
  };

  return useSharedState;
};

export default createSharedStateHook;
```

Counter.js

```
const useSharedCounter = createSharedStateHook(0);

export default function Counter() {
  return (
    <>
      <CounterDisplay />
      <CounterButton />
    </>
  );
}

const CounterDisplay = () => {
  const [count] = useSharedCounter();
  return <div>Count: {count}</div>;
};

const CounterButton = () => {
  const [count, setCount] = useSharedCounter();
  return (
    <button
      onClick={() => setCount(count + 1)}
    >
      Increment
    </button>
  );
};
```

zustand

Component Design Scorecard

State Factories

Generating specialized hooks
to share state or logic without Context

Reduce Wasted Renders

Quantitative

- Less re-rendering ✓
- Less reconciliation ✓
- Smaller footprint ✓
- Less memory usage ?

Qualitative

- More flexibility ✓
- Promotes reuse ✓
- Less cognitive load ✓

Promotes | Detracts | Depends | Neither

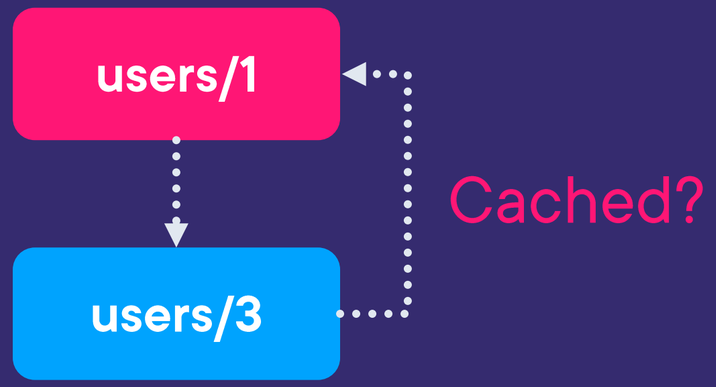
React Performance Scorecard
Find more at bit.ly/PSReactPerfScorecard

Limiting the Effects of useEffect

When useEffects get stuck in loops or fire too many times

UserProfile.jsx

```
const UserProfile = ({ userId }) => {
  const [userData, setUserData] = useState(null);

  useEffect(() => {
    const fetchData = async () => {
      const response = await fetch(`https://api.example.com/users/${userId}`);
      const data = await response.json();
      setUserData(data);
    };

    fetchData();
  }, [userId]);

  return (
    <div>
      {/* Render user data */}
      {userData && <div>{userData.name}</div>}
    </div>
  );
};
```



Limiting the Effects of useEffect

When useEffects get stuck in loops or fire too many times

UserProfile.jsx

```
const UserProfile = ({ user }) => {
  const [userData, setUserData] = useState(null);

  useEffect(() => {
    const fetchData = async () => {
      const response = await fetch(`https://api.example.com/users/${user.id}`);
      const data = await response.json();
      setUserData(data);
    };

    fetchData();
  }, [user]);

return (
    <div>
      {/* Render user data */}
      {userData && <div>{userData.name}</div>}
    </div>
  );
};
```

Needs to be memoized

Guarding Against useEffect

When useEffects get stuck in loops or fire too many times

UserProfile.jsx

```
const UserProfile = ({ userId }) => {
  const [userData, setUserData] = useState(null);
  const [error, setError] = useState(null);
  const [retries, setRetries] = useState(0);

  useEffect(() => {
    const fetchData = async () => {
      const response = await fetch(`https://api.example.com/users/${userId}`);
      const data = await response.json();

      if (data.error) {
        setUserData(null);
        setError(data.error)
      } else {
        setUserData(data);
      }
    };

    fetchData();
  }, [userId]);

  useEffect(() => {
    /* retry logic omitted for brevity */
  }, [error, retries]);

  return (
    <div>
      {userData && <div>{userData.name}</div>}
    </div>
  );
};
```

Query Hooks Example: TanStack Query

Hooks that cache data, share state, and orchestrate async code

UserProfile.jsx

```
const UserProfile = ({ userId }) => {
  const { data: name, error } = useQuery({
    queryKey: ["user", userId],
    queryFn: getUserById,
    retries: 3,
    select: (x) => x.name,
  });

  return (
    <div>
      {name && <div>{name}</div>}
    </div>
  );
};
```

Component Design Scorecard

Query Hooks

Specialized hooks that handle data fetching with caching and error handling

Reduce Wasted Renders

Quantitative

- Less re-rendering ✓
- Less reconciliation ✓
- Smaller footprint ✓
- Less memory usage ✗

Qualitative

- More flexibility ✗
- Promotes reuse ✓
- Less cognitive load ✗

Promotes | Detracts | Depends | Neither

React Performance Scorecard
Find more at bit.ly/PSReactPerfScorecard

Further Learning

Zustand
https://github.com/pmndrs/zustand

TanStack Query
https://tanstack.com/query/

Isolating Updates Using Render Props

Render Props

Used for sharing presentation coupled with stateful logic

DataFetcher.jsx

```
const DataFetcher = ({ url, render }) => {
    const [data, setData] = useState(null);

    useEffect(() => {
      fetch(url)
        .then(response => response.json())
        .then(data => setData(data))
        .catch(error => console.error(error));
    }, [url]);

    return render(data);
};

export default DataFetcher;
```

UserProfile.jsx

```
const UserProfile = () => {
  return (
    <DataFetcher
      url="https://api.example.com/user"
      render={(data) => data
        ? <div>{data.username}</div>
        : <div>Loading...</div>
      }
    />
  );
};
```

Render Props

Used for sharing presentation coupled with stateful logic

DataFetcher.jsx

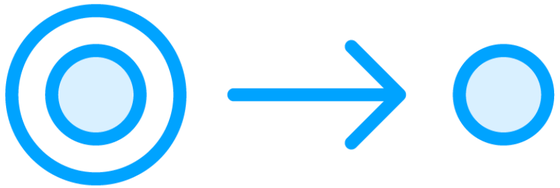
```
const DataFetcher = ({ url, render }) => {
    const [data, setData] = useState(null);

    useEffect(() => {
      fetch(url)
        .then(response => response.json())
        .then(data => setData(data))
        .catch(error => console.error(error));
    }, [url]);

    return render(data);
};

export default DataFetcher;
```

UserProfile.jsx

```
const UserProfile = () => {
  return (
    <DataFetcher
      url="https://api.example.com/user"
      render={(data) => data
        ? <div>{data.username}</div>
        : <div>Loading...</div>
      }
    />
  );
};
```

When to Use Render Props

Need fine-grained control over inline rendering



Avoiding "no-op" components to wrap hooks

Component Design Scorecard

Render Props

Sandbox parts of your component tree by limiting updates that could affect a parent component

Reduce Wasted Renders

Quantitative

- Less re-rendering ✓
- Less reconciliation ✓
- Smaller footprint ✓
- Less memory usage ✓

Qualitative

- More flexibility ✓
- Promotes reuse ✓
- Less cognitive load ✗

Promotes | Detracts | Depends | Neither

React Performance Scorecard
Find more at bit.ly/PSReactPerfScorecard

Render Props Example: React Final Form

Sandboxing components and isolating state updates

OrderForm.jsx

```jsx
import { Form, Field } from 'react-final-form'

const subscription = { submitting: true, pristine: true }

export const OrderForm = () => (
  <Form
    subscription={subscription}
    render={({ handleSubmit }) => (
      <form onSubmit={handleSubmit}>
        <Field name="firstName">
          {({ input, meta }) => (
            <div>
              <label>First Name</label>
              <input {...input} placeholder="First Name" />
              {meta.touched && meta.error && <span>{meta.error}</span>}
            </div>
          )}
        </Field>
        <Field name="lastName">
          {({ input, meta }) => (
            <div>
              <label>Last Name</label>
              <input {...input} placeholder="Last Name" />
              {meta.touched && meta.error && <span>{meta.error}</span>}
            </div>
          )}
        </Field>
      </form>
    )}
  />
)
```

Render Props Example: React Final Form

Sandboxing components and isolating state updates

OrderForm.jsx

```
import { Form, Field } from 'react-final-form'

const subscription = { submitting: true, pristine: true }

export const OrderForm = () => (
  <Form
    subscription={subscription}
    render={({ handleSubmit }) => (
      <form onSubmit={handleSubmit}>
        <Field name="firstName">
          {({ input, meta }) => (
            <div>
              <label>First Name</label>
              <input {...input} placeholder="First Name" />
              {meta.touched && meta.error && <span>{meta.error}</span>}
            </div>
          )}
        </Field>
        <Field name="lastName">
          {({ input, meta }) => (
            <div>
              <label>Last Name</label>
              <input {...input} placeholder="Last Name" />
              {meta.touched && meta.error && <span>{meta.error}</span>}
            </div>
          )}
        </Field>
      </form>
    )}
  />
)
```

Render Props Example: React Final Form

Sandboxing components and isolating state updates

OrderForm.jsx

```jsx
import { Form, Field } from 'react-final-form'

const subscription = { submitting: true, pristine: true }

export const OrderForm = () => (
  <Form
    subscription={subscription}
    render={({ handleSubmit }) => (
      <form onSubmit={handleSubmit}>
        <Field name="firstName">
          {({ input, meta }) => (
            <div>
              <label>First Name</label>
              <input {...input} placeholder="First Name" />
              {meta.touched && meta.error && <span>{meta.error}</span>}
            </div>
          )}
        </Field>
        <Field name="lastName">
          {({ input, meta }) => (
            <div>
              <label>Last Name</label>
              <input {...input} placeholder="Last Name" />
              {meta.touched && meta.error && <span>{meta.error}</span>}
            </div>
          )}
        </Field>
      </form>
    )}
  />
)
```

Render Props Example: React Final Form

Sandboxing components and isolating state updates

OrderForm.jsx

```
import { Form, Field } from 'react-final-form'

const subscription = { submitting: true, pristine: true }

export const OrderForm = () => (
  <Form
    subscription={subscription}
    render={({ handleSubmit }) => (
      <form onSubmit={handleSubmit}>
        <Field name="firstName">
          {({ input, meta }) => (
            <div>
              <label>First Name</label>
              <input {...input} placeholder="First Name" />
              {meta.touched && meta.error && <span>{meta.error}</span>}
            </div>
          )}
        </Field>
        <Field name="lastName">
          {({ input, meta }) => (
            <div>
              <label>Last Name</label>
              <input {...input} placeholder="Last Name" />
              {meta.touched && meta.error && <span>{meta.error}</span>}
            </div>
          )}
        </Field>
      </form>
    )}
  />
)
```

Isolated form sub-tree

Render Props Example: React Final Form

Sandboxing components and isolating state updates

OrderForm.jsx

```jsx
import { Form, Field } from 'react-final-form'

const subscription = { submitting: true, pristine: true }

export const OrderForm = () => (
  <Form
    subscription={subscription}
    render={({ handleSubmit }) => (
      <form onSubmit={handleSubmit}>
        <Field name="firstName">
          {({ input, meta }) => (
            <div>
              <label>First Name</label>
              <input {...input} placeholder="First Name" />
              {meta.touched && meta.error && <span>{meta.error}</span>}
            </div>
          )}
        </Field>
        <Field name="lastName">
          {({ input, meta }) => (
            <div>
              <label>Last Name</label>
              <input {...input} placeholder="Last Name" />
              {meta.touched && meta.error && <span>{meta.error}</span>}
            </div>
          )}
        </Field>
      </form>
    )}
  />
)
```

Isolated form sub-tree

Isolated field sub-trees

Render Props Example: React Final Form

Sandboxing components and isolating state updates

OrderForm.jsx

```jsx
import { Form, Field } from 'react-final-form'

const subscription = { submitting: true, pristine: true }

export const OrderForm = () => (
  <Form
    subscription={subscription}
    render={({ handleSubmit }) => (
      <form onSubmit={handleSubmit}>
        <Field name="firstName">
          {({ input, meta }) => (
            <div>
              <label>First Name</label>
              <input {...input} placeholder="First Name" />
              {meta.touched && meta.error && <span>{meta.error}</span>}
            </div>
          )}
        </Field>
        <Field name="lastName">
          {({ input, meta }) => (
            <div>
              <label>Last Name</label>
              <input {...input} placeholder="Last Name" />
              {meta.touched && meta.error && <span>{meta.error}</span>}
            </div>
          )}
        </Field>
      </form>
    )}
  />
)
```

Isolated form sub-tree

Isolated field sub-trees

Hooks Example: React Final Form

Hooks require extracting code into smaller components

OrderForm.jsx

```
import { Form, useField } from "react-final-form";

const subscription = { submitting: true, pristine: true };

export const OrderForm = () => {
  const handleSubmit = useCallback((values) => {
    // handle submit
  }, []);

  return (
    <Form onSubmit={handleSubmit}
subscription={subscription}>
      {({ handleSubmit }) => (
        <form onSubmit={handleSubmit}>
          <NameFields />
        </form>
      )}
    </Form>
  );
};
```

OrderForm.jsx (cont.)

```
const NameFields = () => {
  const firstName = useField("firstName");
  const lastName = useField("lastName");

  return (
    <>
      <div>
        <label>First Name</label>
        <input {...firstName.input} placeholder="First Name" />
        {firstName.meta.touched && firstName.meta.error && (
          <span>{firstName.meta.error}</span>
        )}
      </div>
      <div>
        <label>Last Name</label>
        <input {...lastName.input} placeholder="Last Name" />
        {lastName.meta.touched && lastName.meta.error && (
          <span>{lastName.meta.error}</span>
        )}
      </div>
    </>
  );
};
```

Hooks Example: React Final Form

Hooks require extracting code into smaller components to isolate renders

OrderForm.jsx

```jsx
import { Form, useField } from "react-final-form";

const subscription = { submitting: true, pristine: true };

export const OrderForm = () => {
  const handleSubmit = useCallback((values) => {
    // handle submit
  }, []);

  return (
    <Form onSubmit={handleSubmit} subscription={subscription}>
      {({ handleSubmit }) => (
        <form onSubmit={handleSubmit}>
          <div>
            <FirstNameField />
            <LastNameField />
          </div>
        </form>
      )}
    </Form>
  );
};
```

Still will re-render fields

OrderForm.jsx (cont.)

```jsx
const FirstNameField = () => {
  const firstName = useField("firstName");

  return (
      <div>
        <label>First Name</label>
        <input {...firstName.input} placeholder="First Name" />
        {firstName.meta.touched && firstName.meta.error && (
          <span>{firstName.meta.error}</span>
        )}
      </div>
  );
};

const LastNameField = () => {
  const lastName = useField("lastName");

  return (
    <div>
        <label>Last Name</label>
        <input {...lastName.input} placeholder="Last Name" />
        {lastName.meta.touched && lastName.meta.error && (
          <span>{lastName.meta.error}</span>
        )}
    </div>
  );
};
```

Hooks Example: React Final Form

Hooks require extracting code into smaller components to isolate renders

OrderForm.jsx

```
import { Form, useField } from "react-final-form";

const subscription = { submitting: true, pristine: true };

export const OrderForm = () => {
  const handleSubmit = useCallback((values) => {
    // handle submit
  }, []);

  return (
    <Form onSubmit={handleSubmit} subscription={subscription}>
      {({ handleSubmit }) => (
        <form onSubmit={handleSubmit}>
          <div>
            <FirstNameField />
            <LastNameField />
          </div>
        </form>
      )}
    </Form>
  );
};
```

OrderForm.jsx (cont.)

```
const FirstNameField = React.memo(() => {
  const firstName = useField("firstName");

  return (
      <div>
        <label>First Name</label>
        <input {...firstName.input} placeholder="First Name" />
        {firstName.meta.touched && firstName.meta.error && (
          <span>{firstName.meta.error}</span>
        )}
      </div>
  );
});

const LastNameField = React.memo(() => {
  const lastName = useField("lastName");

  return (
    <div>
        <label>Last Name</label>
        <input {...lastName.input} placeholder="Last Name" />
        {lastName.meta.touched && lastName.meta.error && (
          <span>{lastName.meta.error}</span>
        )}
    </div>
  );
});
```

Controlling Internal State with useReducer

Internal Component State

UI state that does not need to be managed at the application level

CheckoutForm.html

```
<form id="checkout">
  <label for="name">Name:</label>
  <input name="name" type="text">
</form>

<script type="text/javascript">
  var { name } = document.forms.checkout.elements;
  console.log(name);
</script>
```

Internal Component State

UI state that does not need to be managed at the application level

CheckoutForm.jsx

```
const CheckoutForm = () => {
  const formRef = useRef();

  useEffect(() => {
    var { name } = formRef.elements;
    console.log(name);
  }, []);

  return (
    <form ref={formRef}>
      <label for="name">Name:</label>
      <input name="name" />
    </form>
  );
};
```

Internal Component State

UI state that does not need to be managed at the application level

CheckoutForm.jsx

```
const CheckoutForm = ({ onCheckout }) => {

  const [name, setName] = useState('');
  const [addressLine1, setAddressLine1] = useState('');
  const [addressLine2, setAddressLine2] = useState('');
  const [addressLine3, setAddressLine3] = useState('');
  const [city, setCity] = useState('');
  const [stateProvince, setStateProvince] = useState('');
  const [zip, setZip] = useState('');
  const [country, setCountry] = useState('');

  // Form logic...
  // ...

  return (
    <form>
      <label for="name">Name:</label>
      <input value={name} name="name" />
    </form>
  );
};
```

Controlled input

Component Design Scorecard

Controlled Components

Creating a predictable data flow by having React manage the state of the DOM

Reduce Wasted Renders

Quantitative

- Less re-rendering
- Less reconciliation
- Smaller footprint
- Less memory usage

Qualitative

- More flexibility
- Promotes reuse
- Less cognitive load

Promotes | Detracts | Depends | Neither

React Performance Scorecard
Find more at bit.ly/PSReactPerfScorecard

Complex Internal State

But it will get out of hand quickly with complex state

CheckoutForm.jsx

```
const CheckoutForm = ({ onCheckout }) => {
  const [name, setName] = useState('');
  const [addressLine1, setAddressLine1] = useState('');
  const [addressLine2, setAddressLine2] = useState('');
  const [addressLine3, setAddressLine3] = useState('');
  const [city, setCity] = useState('');
  const [stateProvince, setStateProvince] = useState('');
  const [zip, setZip] = useState('');
  const [country, setCountry] = useState('');

  const [isNameValid, setIsNameValid] = useState(true);
  const [isAddressLine1Valid, setIsAddressLine1Valid] = useState(true);

  // Even more validation logic...
  // ...

  // Form submission logic...
  // ...
};
```

Dr. Russell Ackoff

"Building the right thing wrong is better than building the wrong thing right."

Complex Internal State

But it will get out of hand quickly with complex state

CheckoutForm.jsx

```
const CheckoutForm = ({ onCheckout }) => {
  const [name, setName] = useState('');
  const [addressLine1, setAddressLine1] = useState('');
  const [addressLine2, setAddressLine2] = useState('');
  const [addressLine3, setAddressLine3] = useState('');
  const [city, setCity] = useState('');
  const [stateProvince, setStateProvince] = useState('');
  const [zip, setZip] = useState('');
  const [country, setCountry] = useState('');

  const [isNameValid, setIsNameValid] = useState(true);
  const [isAddressLine1Valid, setIsAddressLine1Valid] = useState(true);

  // Even more validation logic...
  // ...

  // Form submission logic...
  // ...
};
```

useReducer

React API to consolidate local state management

```
const [          ,          ] = useReducer( reducer , initialState );
```

useReducer

React API to consolidate local state management

```
const [          ,          ] = useReducer( reducer , initialState );
```

```
function reducer(
  state: object,
  action: { type: string }
) {





}
```

useReducer

React API to consolidate local state management

```
const [          ,          ] = useReducer( reducer , initialState );
```

```
function reducer(
  state: object,
  action: { type: string }
) {



}
```

```
{
  addressLine1,
  addressLine2,
  addressLine3,
  stateProvince,
  postalCode,
  country
}
```

useReducer

React API to consolidate local state management

```
const [          ,          ] = useReducer( reducer , initialState );
```

```
function reducer(
  state: object,
  action: { type: string }
) {



}
```

```
{
  type: "<VERB_ACTION>",

}
```

useReducer

React API to consolidate local state management

```
const [          ,          ] = useReducer( reducer , initialState );
```

```
function reducer(
  state: object,
  action: { type: string }
) {



}
```

```
{
  type: "<VERB_ACTION>",
  ACTION_PARAM: <VALUE>,
  ACTION_PARAM: <VALUE>
}
```

useReducer

React API to consolidate local state management

```
const [          ,          ] = useReducer( reducer , initialState );
```

```
function reducer(
  state: object,
  action: { type: string }
) {



}
```

```
{
  type: "UPDATE_ADDRESS",
  field: "addressLine1",
  value: "555 Happy Ave N"
}
```

useReducer

React API to consolidate local state management

```
const [          ,          ] = useReducer( reducer , initialState );
```

```
function reducer(
  state,
  action
) {
 switch (action.type) {
    case "UPDATE_ADDRESS":
      return {
        ...state,
        [action.field]: action.value
      }
    default:
      return state;
  }
}
```

```
{
  type: "UPDATE_ADDRESS",
  field: "addressLine1",
  value: "555 Happy Ave N"
}
```

useReducer

React API to consolidate local state management

```
const [          ,          ] = useReducer( reducer , initialState );
```

```
function reducer(
  state,
  action
) {
 switch (action.type) {
    case "UPDATE_ADDRESS":
      return {
        ...state,
        [action.field]: action.value
      }
    case "VALIDATE_ADDRESS":
      const isValid = checkAddress(state);

      return {
        ...state,
        isValid
      }
    default:
      return state;
  }
}
```

```
{
  type: "VALIDATE_ADDRESS"
}
```

useReducer

React API to consolidate local state management

```
const [ state , dispatch ] = useReducer( reducer , initialState );
```

useReducer

React API to consolidate local state management

```
const [ state , dispatch ] = useReducer( reducer , initialState );

const handleOnAddressLine1Change = (e) => {
  const value = e.target.value;

  dispatch({
    type: "UPDATE_ADDRESS",
    field: "addressLine1",
    value
  });
};
```

useReducer

React API to consolidate local state management

```
const [ state , dispatch ] = useReducer( reducer , initialState );
```

```
function reducer(
  state,
  action
) {
 switch (action.type) {
    case "UPDATE_ADDRESS":
      return {
        ...state,
        [action.field]: action.value
      }
    case "VALIDATE_ADDRESS":
      const isValid = checkAddress(state);

      return {
        ...state,
        isValid
      }
    default:
      return state;
  }
}
```

useReducer

React API to consolidate local state management

```
const [ state , dispatch ] = useReducer( reducer , initialState );
```

```
function reducer(
  state,
  action
) {
 switch (action.type) {
    case "UPDATE_ADDRESS":
      return {
        ...state,
        [action.field]: action.value
      }
    case "VALIDATE_ADDRESS":
      const isValid = checkAddress(state);

      return {
        ...state,
        isValid
      }
    default:
      return state;
  }
}
```

Async Actions with useReducer

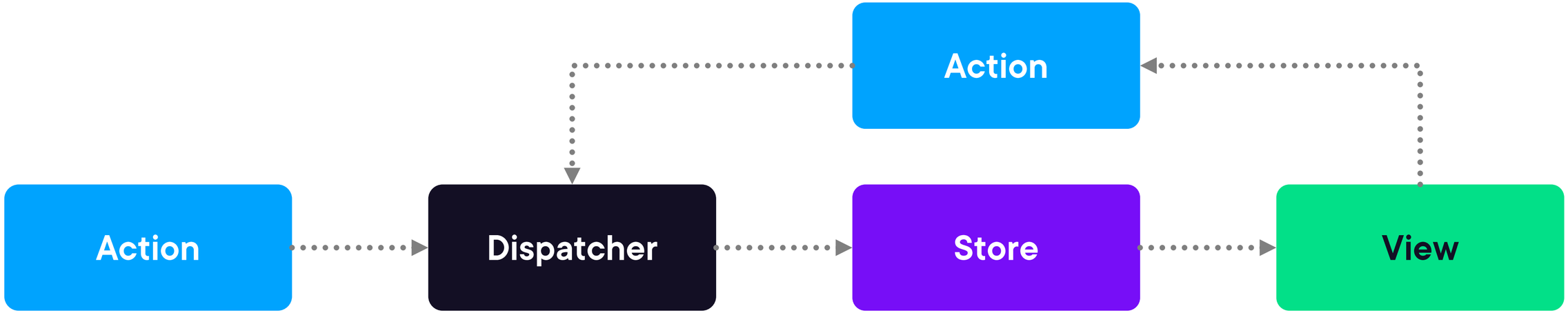
Perform the async work and then dispatch a state update with the result

```
const [state, dispatch] = useReducer(addressReducer, {
  ...initialAddressFields,
  isValid: null
});

const handleOnAddressLine1Change = (e) => {
  const value = e.target.value;

  dispatch({
    type: "UPDATE_ADDRESS",
    field: "addressLine1",
    value
  });

  validateAddressAsync(value).then((isValid) => {
    dispatch({
      type: "VALIDATE_ADDRESS",
      isValid
    });
  });
};
```

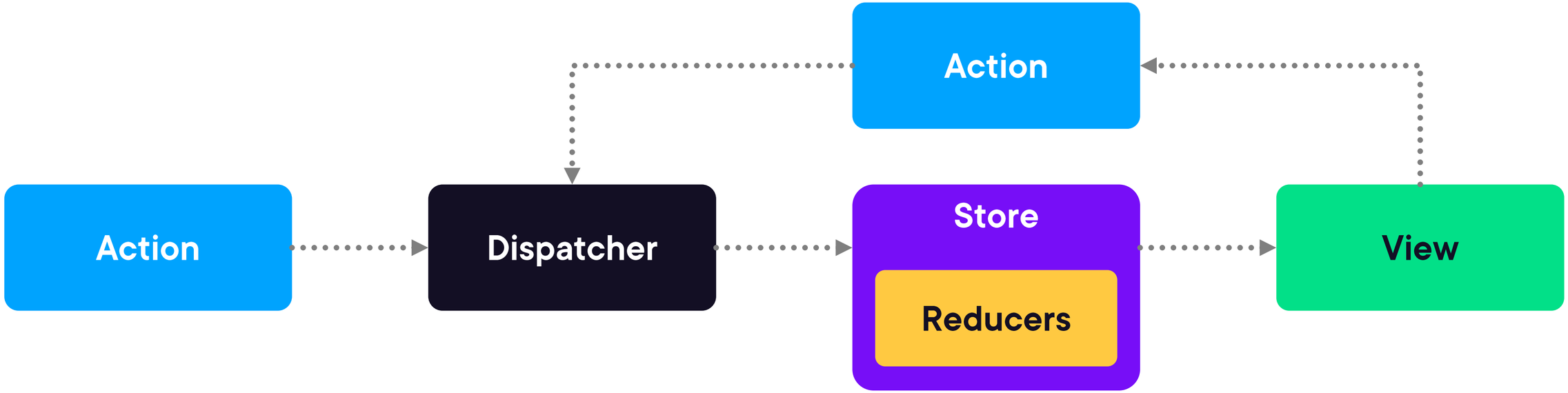
Step 1

Step 2

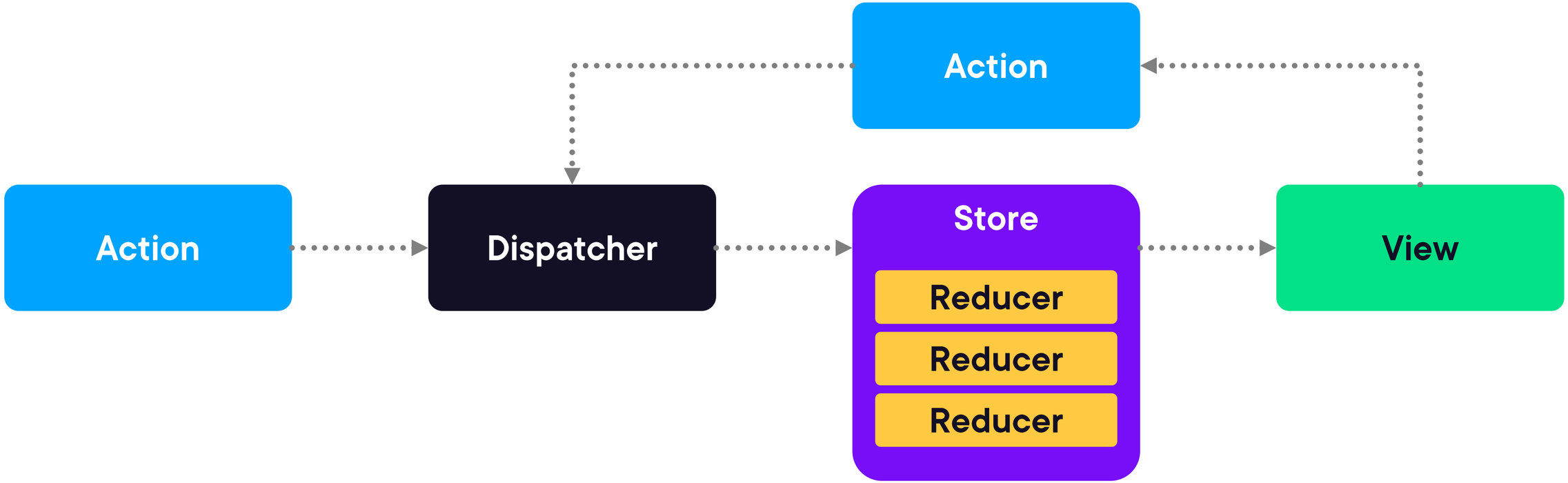
What is Flux State Architecture?



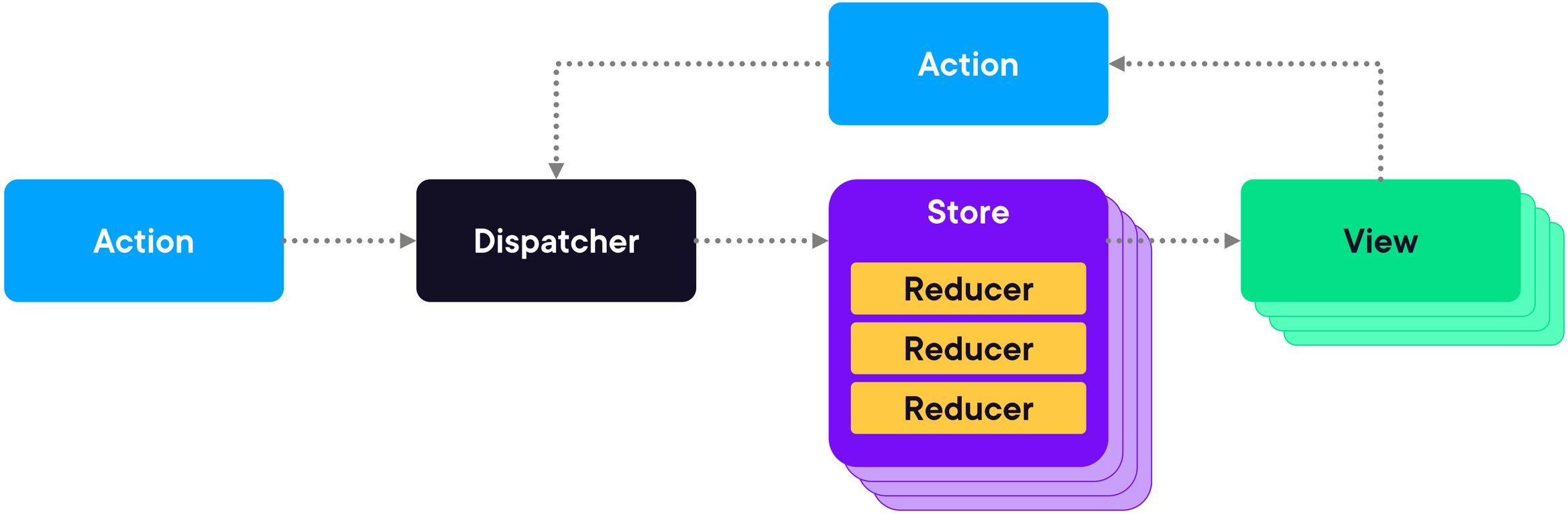
What is Flux State Architecture?



What is Flux State Architecture?



What is Flux State Architecture?



Learn More About Flux

Flux In-Depth Overview (Archived)

https://facebookarchive.github.io/flux/docs/in-depth-overview

Redux – Part 1 – Overview & Concepts

https://redux.js.org/tutorials/essentials/part-1-overview-concepts

Consolidating State Hooks with useReducer

Replaces multiple disparate useState hooks with a single hook

CheckoutForm.jsx

```
const CheckoutForm = ({ onCheckout }) => {
  const [state, dispatch] = useReducer(checkoutReducer, {
    name: '',
    shippingAddress: null,
    billingAddress: null,
  });

  /* callbacks to handle dispatching order state updates */

  return (
    <form onSubmit={onOrderSubmit}>
      <OrderShippingAddress
        address={state.shippingAddress}
        onChange={handleShippingAddressChange}
      />
      <OrderBillingAddress
        address={state.billingAddress}
        shippingAddress={state.shippingAddress}
        onChange={handleBillingAddressChange}
      />

      // Rest of checkout form
      // ...
    </form>
  )
};
```

Composing Reducers

You can create reducers to manage different slices of state

CheckoutForm.jsx

```
const CheckoutForm = ({ onCheckout }) => {
  const [state, dispatch] = useReducer(checkoutReducer, {
    name: '',
    shippingAddress: null,
    billingAddress: null,
  });

  /* callbacks to handle dispatching order state updates */

  return (
    <form onSubmit={onOrderSubmit}>
      <OrderShippingAddress
        address={state.shippingAddress}
        onChange={handleShippingAddressChange}
      />
      <OrderBillingAddress
        address={state.billingAddress}
        shippingAddress={state.shippingAddress}
        onChange={handleBillingAddressChange}
      />

      // Rest of checkout form
      // ...
    </form>
  )
};
```

OrderShippingAddress.js

```
function OrderShippingAddress({ address, onChange }) {
  const [state, dispatch] = useReducer(
    addressReducer, {
      ...address,
      isValid: null
  });

  // Rest of input handling and callbacks
  // ...
}
```

Component Design Scorecard

Flux Reducers

Consolidate complex local state using the Flux data flow pattern

Reduce Wasted Renders

Quantitative

- Less re-rendering ✓
- Less reconciliation ⊖
- Smaller footprint ✓
- Less memory usage ✓

Qualitative

- More flexibility ✓
- Promotes reuse ✓
- Less cognitive load ✓

Promotes | Detracts | Depends | Neither

React Performance Scorecard
Find more at bit.ly/PSReactPerfScorecard

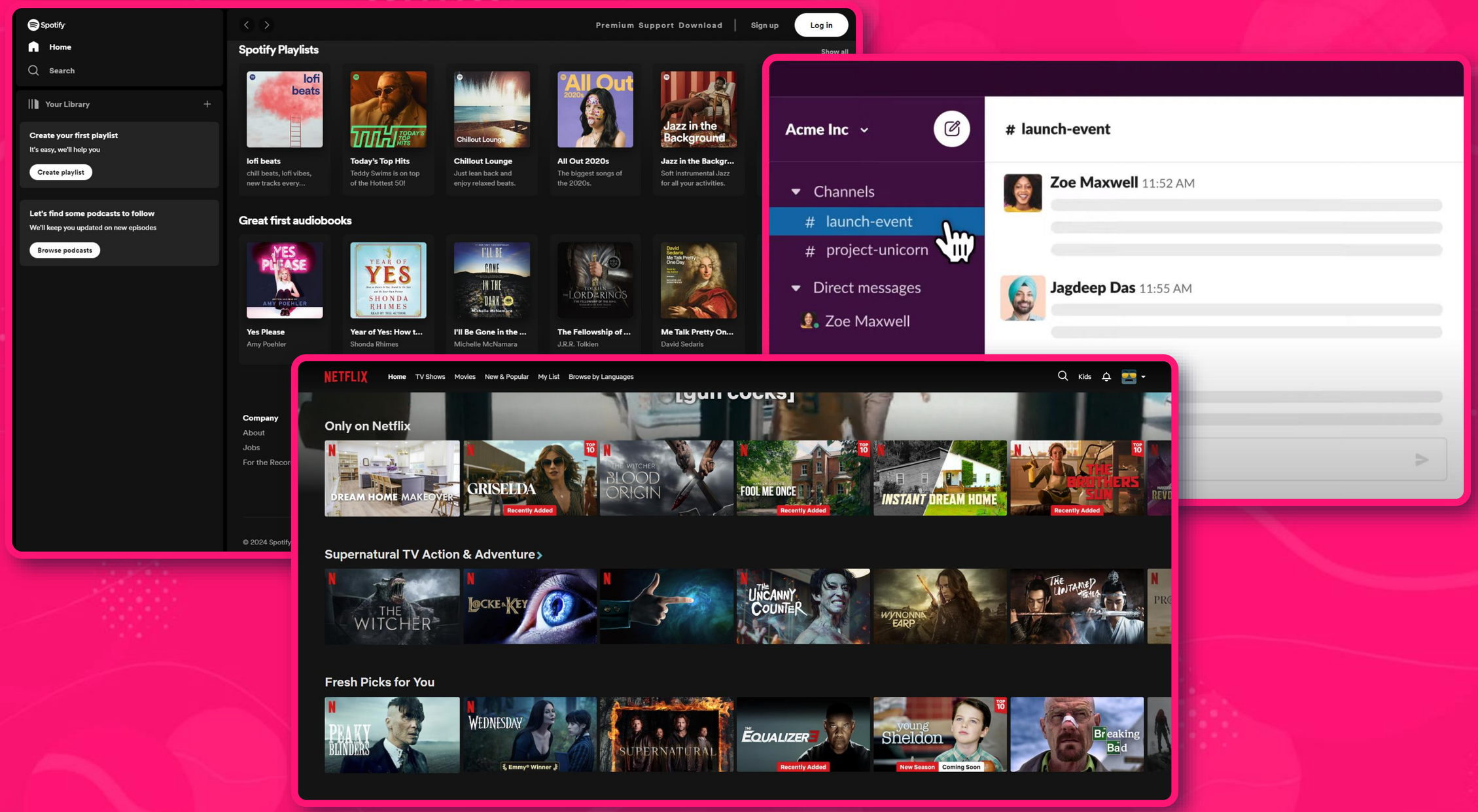
Controlled Form Libraries for React

React Final Form
https://final-form.org/react

Formik
https://formik.org/

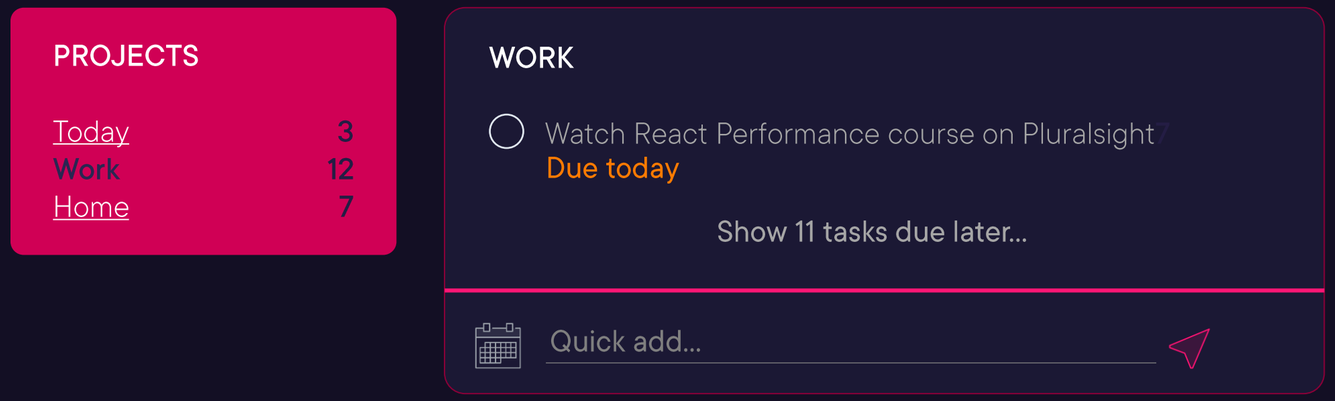
React Hook Form
https://react-hook-form.com/

Simplify Complex UI with State Management


Spotify
Home
Search
Your Library
Create your first playlist
It's easy, we'll help you
Create playlist
Let's find some podcasts to follow
We'll keep you updated on new episodes
Browse podcasts
Premium Support Download
Sign up
Log in
Spotify Playlists
Show all
lofi beats
chill beats, lofi vibes, new tracks every...
Today's Top Hits
Teddy Swims is on top of the Hottest 50!
Chillout Lounge
Just lean back and enjoy relaxed beats.
All Out 2020s
The biggest songs of the 2020s.
Jazz in the Backgr...
Soft instrumental Jazz for all your activities.
Great first audiobooks
Yes Please
Amy Poehler
Year of Yes: How t...
Shonda Rhimes
I'll Be Gone in the ...
Michelle McNamara
The Fellowship of ...
J.R.R. Tolkien
Me Talk Pretty On...
David Sedaris
Company
About
Jobs
For the Record
© 2024 Spotify
Acme Inc
Channels
launch-event
project-unicorn
Direct messages
Zoe Maxwell
launch-event
Zoe Maxwell 11:52 AM
Jagdeep Das 11:55 AM
NETFLIX
Home
TV Shows
Movies
New & Popular
My List
Browse by Languages
Kids
Only on Netflix
DREAM HOME MAKEOVER
GRISELDA
Recently Added
THE WITCHER BLOOD ORIGIN
FOOL ME ONCE
Recently Added
INSTANT DREAM HOME
THE BROTHERS SUN
Recently Added
Supernatural TV Action & Adventure
THE WITCHER
LOCKE & KEY
THE UNCANNY COUNTER
WYNONNA EARP
THE UNTAMED
Fresh Picks for You
PEAKY BLINDERS
WEDNESDAY
Emmy® Winner
SUPERNATURAL
THE EQUALIZER 3
Recently Added
young Sheldon
New Season
Coming Soon
Breaking Bad

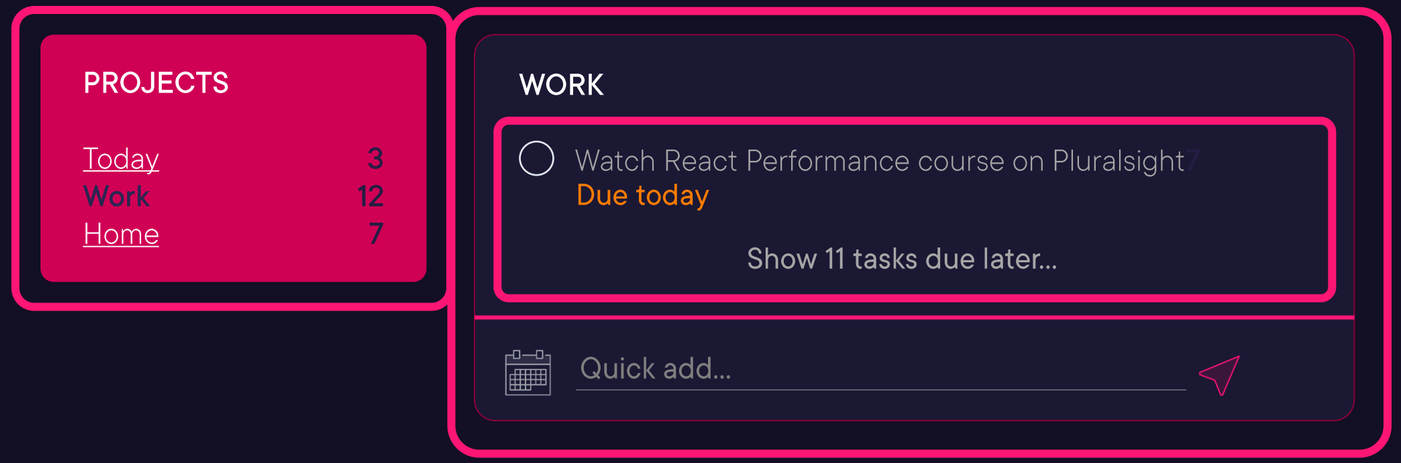
State Management Example: Todos App

Orchestrating disparate parts of the UI



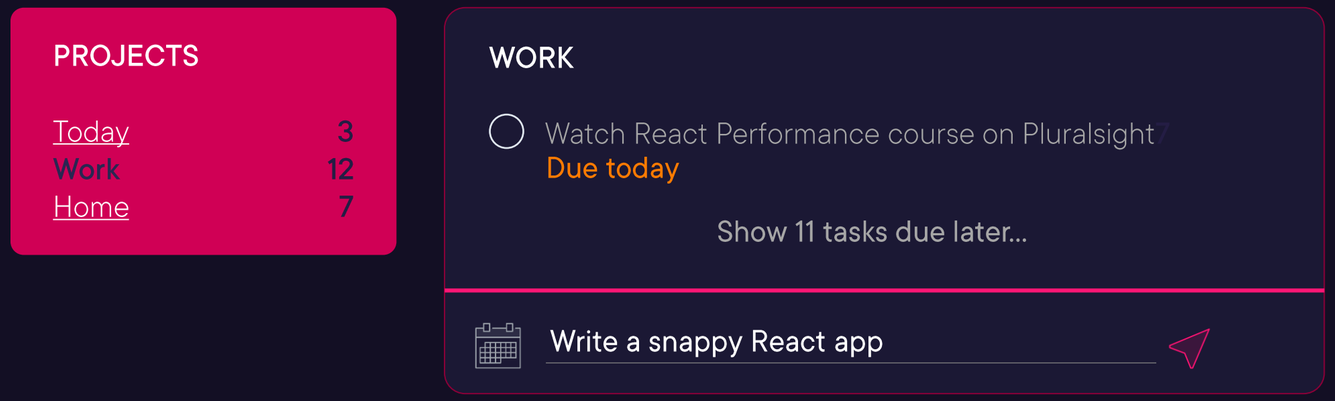
State Management Example: Todos App

Orchestrating disparate parts of the UI



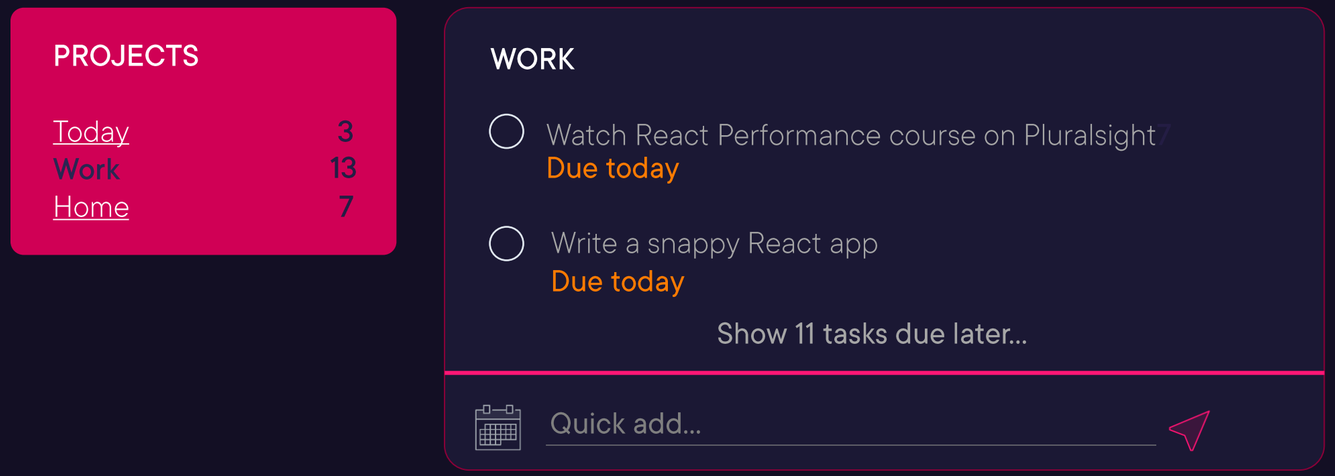
State Management Example: Todos App

Orchestrating disparate parts of the UI

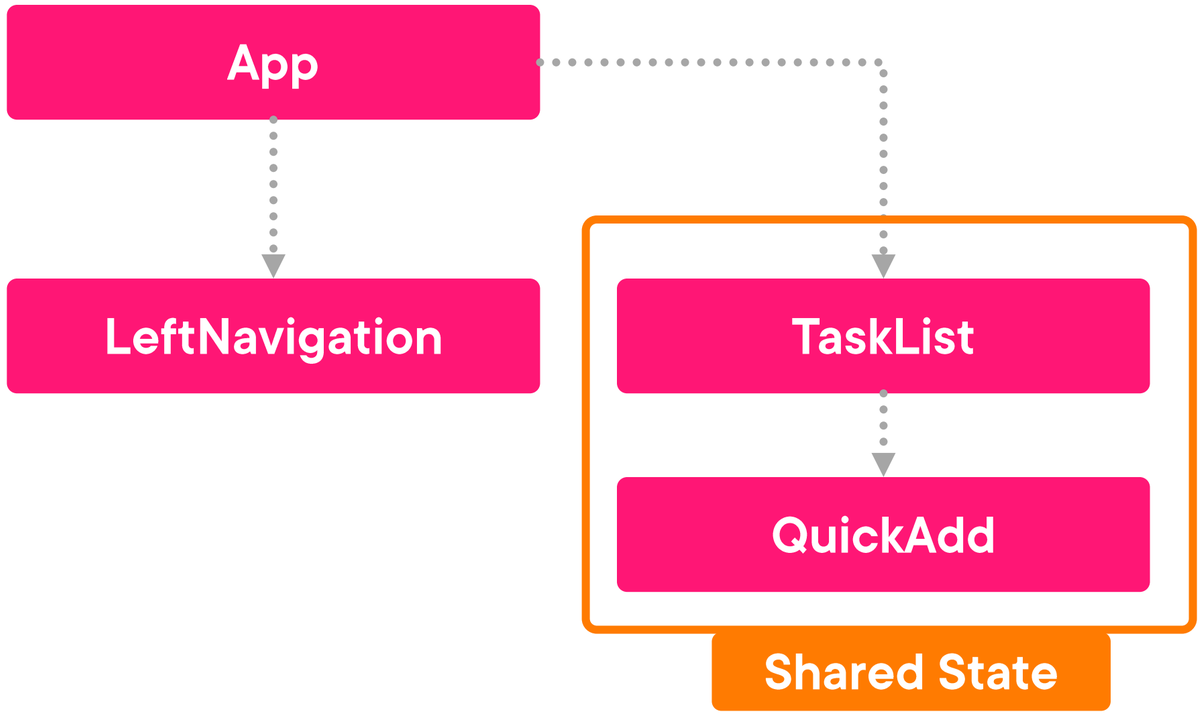


State Management Example: Todos App

Orchestrating disparate parts of the UI

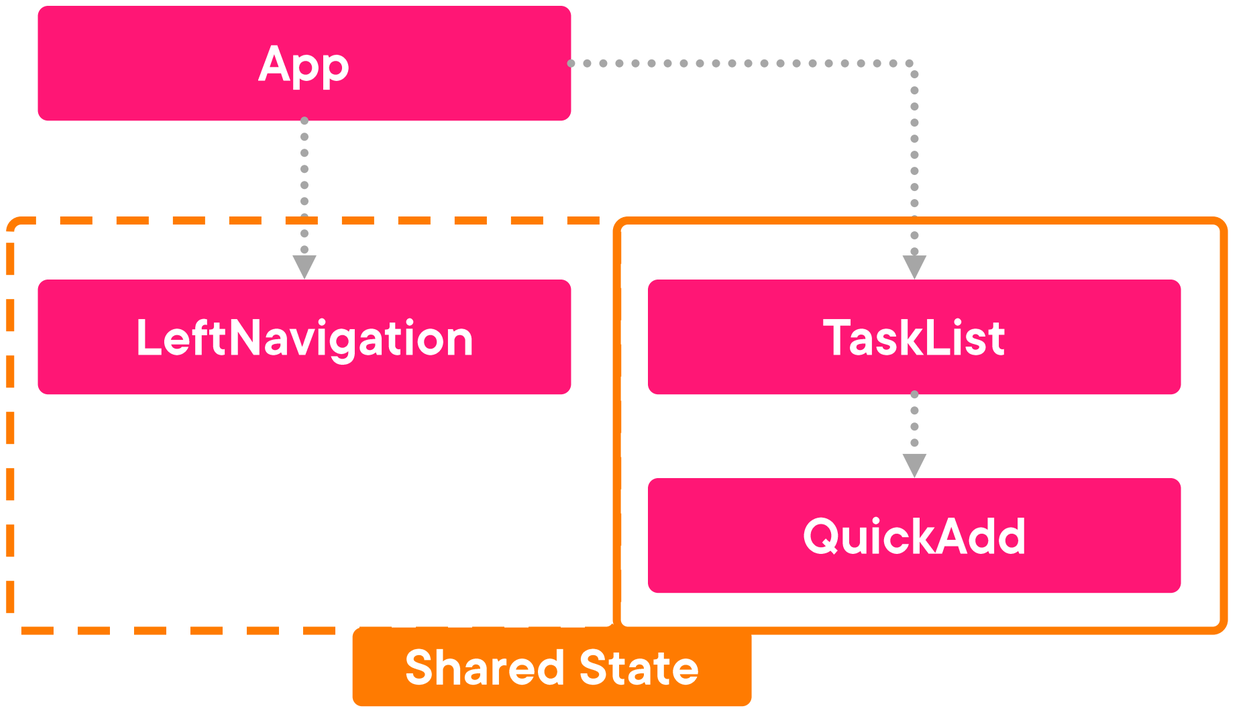


Lifting State Up



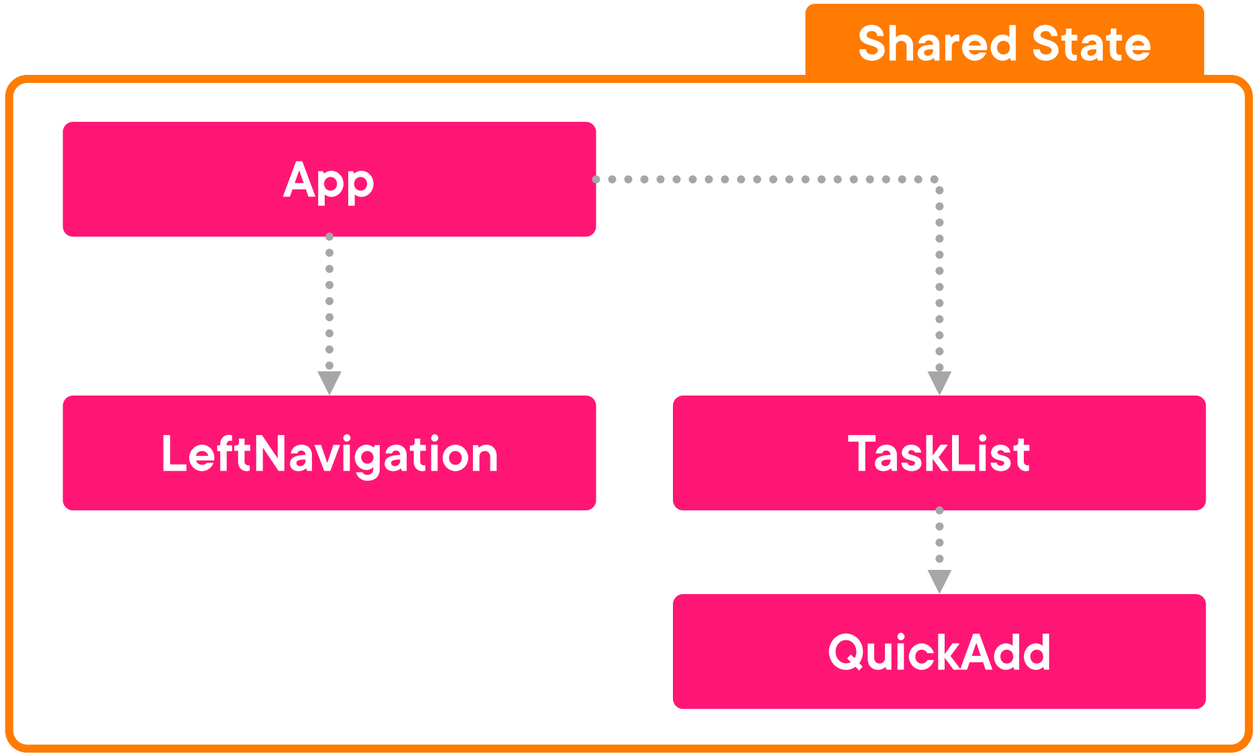
Owner relationship

Lifting State Up



Owner relationship

Lifting State Up



Owner relationship

Context API

Prone to misuse and re-rendering without careful memoization

App.jsx

```
const App = () => (
  <TodosContext.Provider initial={initialState}>
    <TaskList />
    <LeftNavigation />
  </TodosContext.Provider>
)
```

Component Design Scorecard

State Management Solutions

Share data across an app to help simplify complex UIs

Reduce Wasted Renders

Quantitative

- Less re-rendering ✓
- Less reconciliation ✗
- Smaller footprint ✓
- Less memory usage ✓

Qualitative

- More flexibility ✓
- Promotes reuse ✓
- Less cognitive load ✓

Promotes | Detracts | Depends | Neither

React Performance Scorecard
Find more at bit.ly/PSReactPerfScorecard

React State Management Flavors

React State Management Flavors

Flux

(e.g. Redux Toolkit)

React State Management Flavors

Flux

(e.g. Redux Toolkit)

Specialized

(e.g. Formik, AgGrid)

React State Management Flavors

Flux

(e.g. Redux Toolkit)

Specialized

(e.g. Formik, AgGrid)

Observable

(e.g. TanStack Query, MobX)

React State Management Flavors

Flux

(e.g. Redux Toolkit)

Specialized

(e.g. Formik, AgGrid)

Observable

(e.g. TanStack Query, MobX)

Flux "lite"

(e.g. zustand)

React State Management Flavors

Flux

(e.g. Redux Toolkit)

Specialized

(e.g. Formik, AgGrid)

Observable

(e.g. TanStack Query, MobX)

Flux "lite"

(e.g. zustand)

State Machines

(e.g. Xstate)

React State Management Flavors

Flux

(e.g. Redux Toolkit)

Specialized

(e.g. Formik, AgGrid)

Observable

(e.g. TanStack Query, MobX)

Flux "lite"

(e.g. zustand)

State Machines

(e.g. Xstate)

URL

(e.g. React Router)

React State Management Flavors

Flux

(e.g. Redux Toolkit)

Specialized

(e.g. Formik, AgGrid)

Observable

(e.g. TanStack Query, MobX)

Flux "lite"

(e.g. zustand)

State Machines

(e.g. Xstate)

URL

(e.g. React Router)

Recommended Learning

Reselect (Memoizing State Selectors)
https://github.com/reduxjs/reselect

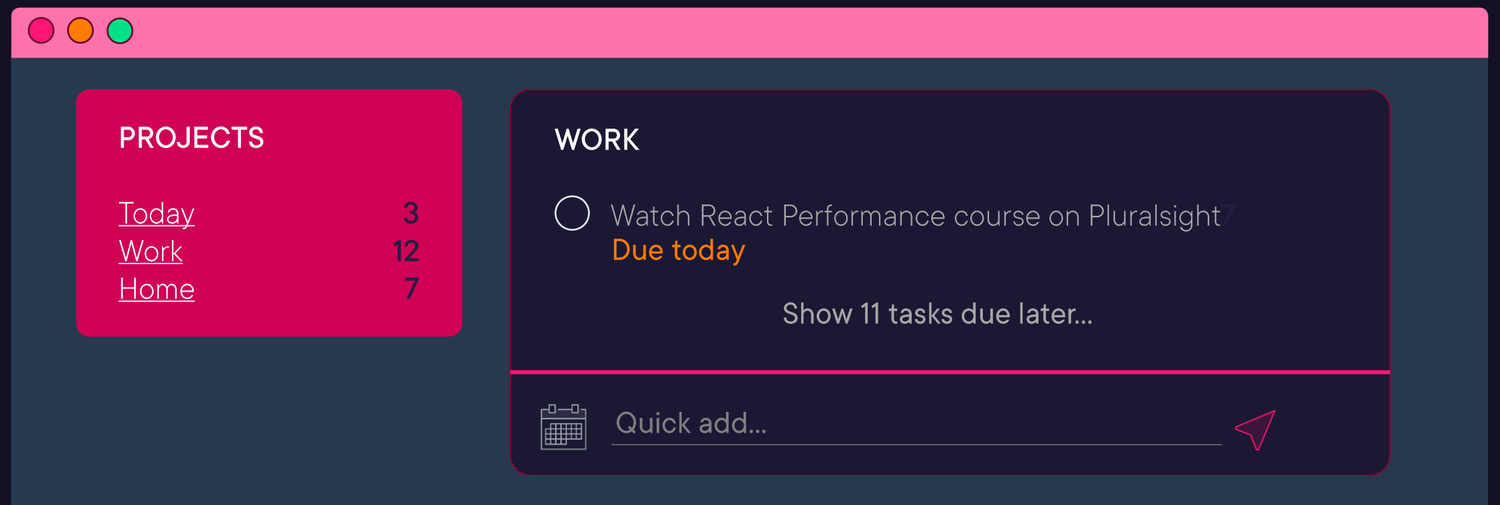
Zustand Comparison Documentation
https://docs.pmnd.rs/zustand/getting-started/comparison

React Libraries – State Management
https://www.robinwieruch.de/react-libraries/#react-state-management

Optimizing Derived State with Selectors and Slicing

State Management Example: Todos App

Orchestrating disparate parts of the UI



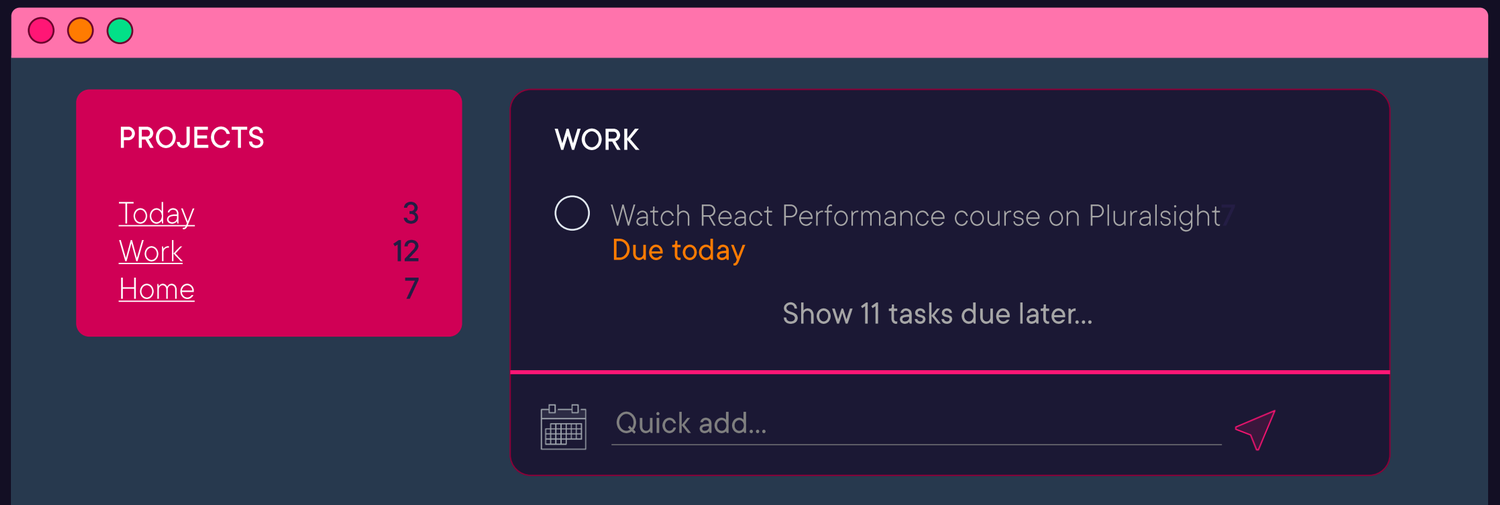
State Management Example: zustand

A lighter version of the Flux data architecture

```
const LeftNavigation = () => {

  const projects =
    useProjectStore(
      selectProjectsWithCount);

  // Rest of component
  // ...
};
```

```
export useProjectStore = create((set) => ({
  projects: ['Today'],
  activeProject: 'Today',
  todos: [],
  addTodo: (text) => set((state) => ({
    todos: [
      ...state.todos, {
        body: text,
        project: state.activeProject
      }
    ]
  }))
});
```

```
const TaskList = () => {
  const {
    projectName,
    dueSoon,
    dueLater
  } = useProjectStore(selectActiveProject);

  // Rest of component
  // ...
};
```




```
const QuickAdd = () => {
  const addTodo = useProjectStore(
    useShallow(x => x.addTodo));

  const handleOnSubmit = (e) => {
    addTodo(e.values.todo);
  };

  return (
    <form onSubmit={handleOnSubmit}>
    // ...
    </form>
  )
}
```

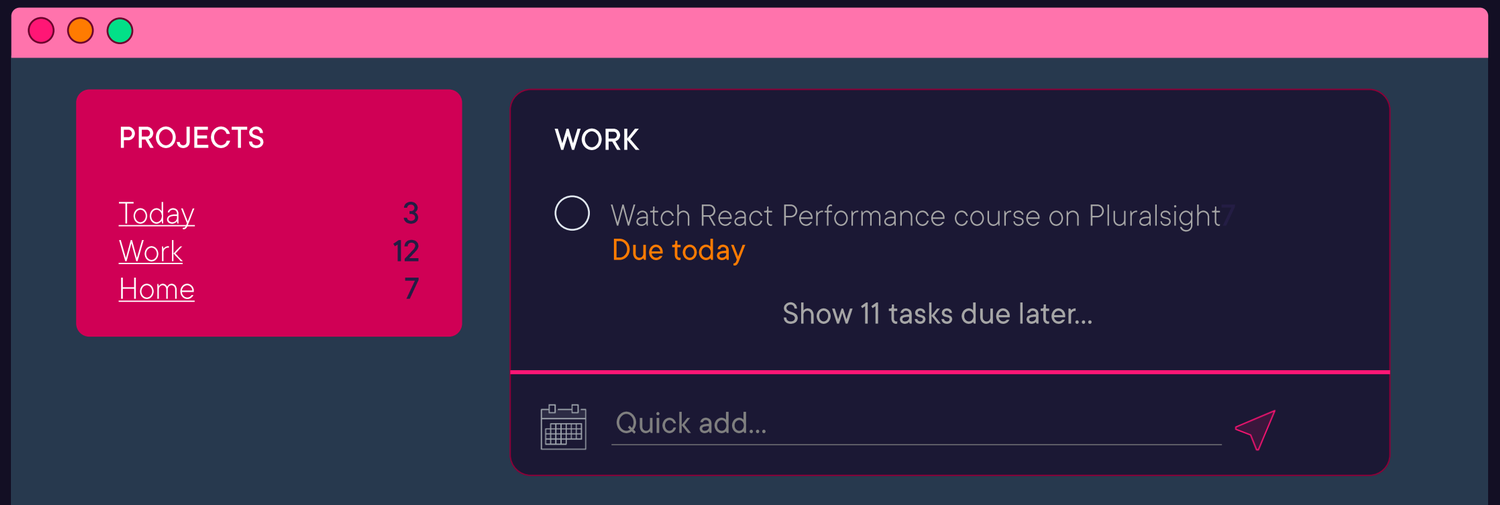
State Management Example: zustand

A lighter version of the Flux data architecture

```
const LeftNavigation = () => {

  const projects =
    useProjectStore(
      selectProjectsWithCount);

  // Rest of component
  // ...
};
```

```
export useProjectStore = create((set) => ({
  projects: ['Today'],
  activeProject: 'Today',
  todos: [],
  addTodo: (text) => set((state) => ({
    todos: [
      ...state.todos, {
        body: text,
        project: state.activeProject
      }
    ]
  }))
});
```

```
const TaskList = () => {
  const {
    projectName,
    dueSoon,
    dueLater
  } = useProjectStore(selectActiveProject);

  // Rest of component
  // ...
};
```




```
const QuickAdd = () => {
  const addTodo = useProjectStore(
    useShallow(x => x.addTodo));

  const handleOnSubmit = (e) => {
    addTodo(e.values.todo);
  };

  return (
    <form onSubmit={handleOnSubmit}>
    // ...
    </form>
  )
}
```

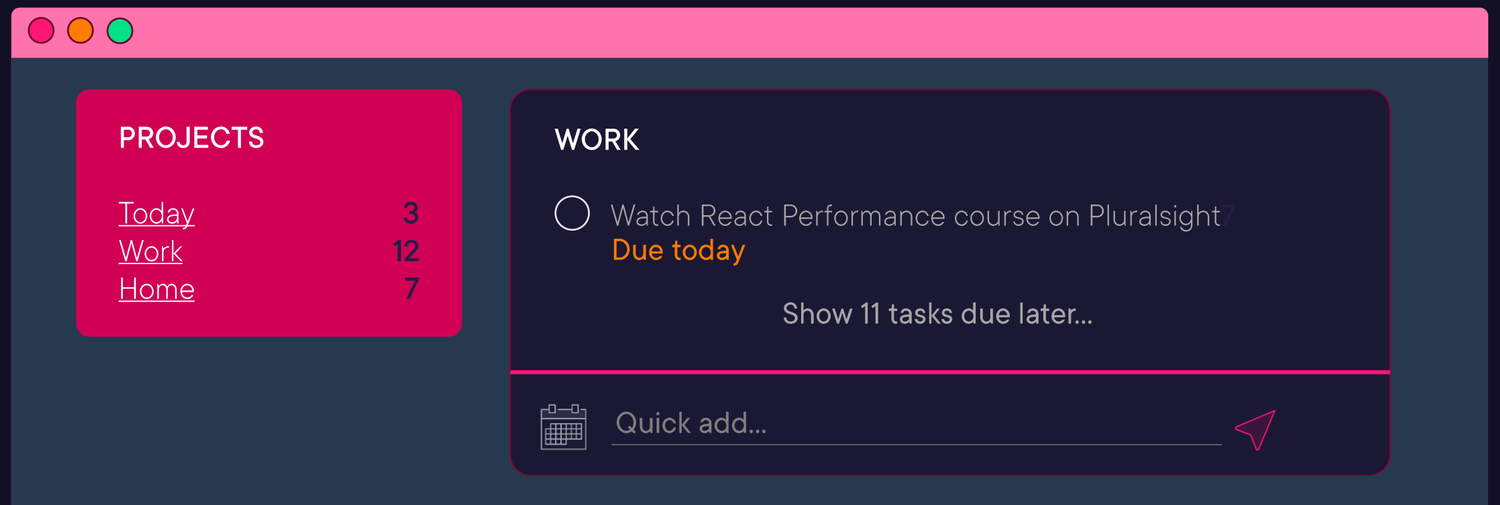
State Management Example: zustand

A lighter version of the Flux data architecture

```
const LeftNavigation = () => {

  const projects =
    useProjectStore(
      selectProjectsWithCount);

  // Rest of component
  // ...
};
```

```
export useProjectStore = create((set) => ({
  projects: ['Today'],
  activeProject: 'Today',
  todos: [],
  addTodo: (text) => set((state) => ({
    todos: [
      ...state.todos, {
        body: text,
        project: state.activeProject
      }
    ]
  }))
});
```

```
const TaskList = () => {
  const {
    projectName,
    dueSoon,
    dueLater
  } = useProjectStore(selectActiveProject);

  // Rest of component
  // ...
};
```




```
const QuickAdd = () => {
  const addTodo = useProjectStore(
    useShallow(x => x.addTodo));

  const handleOnSubmit = (e) => {
    addTodo(e.values.todo);
  };

  return (
    <form onSubmit={handleOnSubmit}>
    // ...
    </form>
  )
}
```

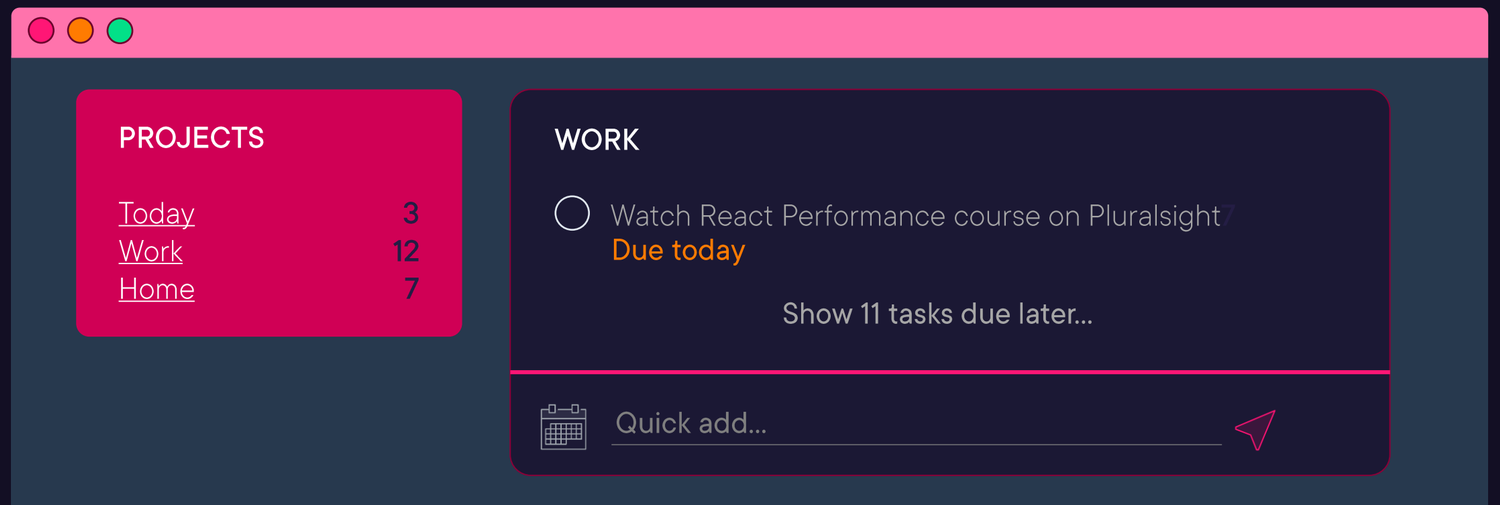
State Management Example: zustand

A lighter version of the Flux data architecture

```
const LeftNavigation = () => {

  const projects =
    useProjectStore(
      selectProjectsWithCount);

  // Rest of component
  // ...
};
```

```
export useProjectStore = create((set) => ({
  projects: ['Today'],
  activeProject: 'Today',
  todos: [],
  addTodo: (text) => set((state) => ({
    todos: [
      ...state.todos, {
        body: text,
        project: state.activeProject
      }
    ]
  }))
});
```

```
const TaskList = () => {
  const {
    projectName,
    dueSoon,
    dueLater
  } = useProjectStore(selectActiveProject);

  // Rest of component
  // ...
};
```




```
const QuickAdd = () => {
  const addTodo = useProjectStore(
    useShallow(x => x.addTodo));

  const handleOnSubmit = (e) => {
    addTodo(e.values.todo);
  };

  return (
    <form onSubmit={handleOnSubmit}>
    // ...
    </form>
  )
}
```

State Selectors

A pure function that can transform global state for local usage

LeftNavigation.jsx

```
const LeftNavigation = () => {

  const projects =
    useProjectStore(
      selectProjectsWithCount);

  // Rest of component
  // ...
};
```

State Selectors

A pure function that can transform global state for local usage

LeftNavigation.jsx

```
const LeftNavigation = () => {

  const projects =
    useProjectStore(
      selectProjectsWithCount);

  // Rest of component
  // ...
};
```

```
function selectProjectsWithCount(state) {

  /* derive from state */

  return derivedResult;
}
```

State Selectors

A pure function that can transform global state for local usage

LeftNavigation.jsx

```
const LeftNavigation = () => {

  const projects =
    useProjectStore(
      selectProjectsWithCount);

  // Rest of component
  // ...
};
```

```
{
  projects: ['Today', 'Work'],
  todos: [
    { id: 'todos/1',
      project: 'Today',
    },
    { id: 'todos/2',
      project: 'Work',
    }
  ]
}
```

State Selectors

A pure function that can transform global state for local usage

LeftNavigation.jsx

```
import _ from 'lodash';

const selectProjectsWithCounts = (state) => {
  const groupedTodos = _.groupBy(state.todos, 'project');

  return _.map(groupedTodos, (todos, project) => ({
    project: project,
    count: _.size(todos)
  }));
};
```

```
{
  projects: ['Today', 'Work'],
  todos: [
    { id: 'todos/1',
      project: 'Today',
    },
    { id: 'todos/2',
      project: 'Work',
    }
  ]
}
```

State Selectors

A pure function that can transform global state for local usage

LeftNavigation.jsx

```
import _ from 'lodash';

const selectProjectsWithCounts = (state) => {
  const groupedTodos = _.groupBy(state.todos, 'project');

  return _.map(groupedTodos, (todos, project) => ({
    project: project,
    count: _.size(todos)
  }));
};
```

```
{
  projects: ['Today', 'Work'],
  todos: [
    { id: 'todos/1',
      project: 'Today',
    },
    { id: 'todos/2',
      project: 'Work',
    }
  ]
}
```

Optimizing State Selectors

Reducing the computation required to transform state

LeftNavigation.jsx

```
import _ from 'lodash';

const selectProjectsWithCounts = (state) => {
  return _(state.todos)
    .groupBy('project')
    .map((todos, project) => ({
      project: project,
      count: todos.length
    }))
    .value();
};
```

Local optimization

The "System"

```
{
  projects: ['Today', 'Work'],
  todos: [
    { id: 'todos/1',
      project: 'Today',
    },
    { id: 'todos/2',
      project: 'Work',
    }
  ]
}
```

Optimizing State Selectors

Reducing the computation required to transform state

LeftNavigation.jsx

```
import _ from 'lodash';

const selectProjectsWithCounts = (state) => {
  return _(state.todos)
    .groupBy('project')
    .map((todos, project) => ({
      project: project,
      count: todos.length
    }))
    .value();
};
```

```
{
  projects: ['Today', 'Work'],
  todos: [
    { id: 'todos/1',
      project: 'Today',
    },
    { id: 'todos/2',
      project: 'Work',
    }
  ]
}
```

Big O Notation

In plain words, Big O notation describes the **complexity** of your code using algebraic terms.

E.g.

O(1)
O(N)
O(log N)
$O(N)^2$

https://www.freecodecamp.org/news/big-o-notation-why-it-matters-and-why-it-doesnt-1674cfa8a23c/

Optimizing State Selectors

Reducing the computation required to transform state

LeftNavigation.jsx

```
import _ from 'lodash';

const selectProjectsWithCounts = (state) => {
  return _(state.todos)
    .groupBy('project')
    .map((todos, project) => ({
      project: project,
      count: todos.length
    }))
    .value();
};
```

```
{
  projects: ['Today', 'Work'],
  todos: [
    { id: 'todos/1',
      project: 'Today',
    },
    { id: 'todos/2',
      project: 'Work',
    }
  ]
}
```

Optimizing State Selectors

Reducing the computation required to transform state

LeftNavigation.jsx

```
import _ from 'lodash';

const selectProjectsWithCounts = (state) => {
  return _(state.todos)
    .groupBy('project')
    .map((todos, project) => ({
      project: project,
      count: todos.length
    }))
    .value();
};
```

```
{
  projects: ['Today', 'Work'],
  todos: {
    Today: [
      { id: 'todos/1' }
    ],
    Work: [
      { id: 'todos/2' }
    ]
  }
}
```

Optimizing State Selectors

Reducing the computation required to transform state

LeftNavigation.jsx

```
const selectProjectsWithCounts = (state) => {
  return state.projects.map((project) => ({
    project,
    count: state.todos[project].length,
  }));
};
```

```
{
  projects: ['Today', 'Work'],
  todos: {
    Today: [
      { id: 'todos/1' }
    ],
    Work: [
      { id: 'todos/2' }
    ]
  }
}
```

Optimizing State Selectors

Reducing the computation required to transform state

useProjectStore.js

```
import { create } from "zustand";

export const useProjectStore = create((set) => ({
  projects: ["Today"],
  activeProject: "Today",
  todos: {},
  addTodo: (text) =>
    set((state) => {
      const id = `todos/${state.todos[state.activeProject].length + 1}`;

      return {
        todos: {
          ...state.todos,
          [state.activeProject]: [
            ...state.todos[state.activeProject],
            {
              id,
              body: text,
            },
          ],
        },
      };
    }),
}));
```

```
{
  projects: ['Today', 'Work'],
  todos: {
    Today: [
      { id: 'todos/1' }
    ],
    Work: [
      { id: 'todos/2' }
    ]
  }
}
```

Optimizing State Selectors

Reducing the computation required to transform state

TaskList.jsx

```
function selectActiveProject(state) {
  return state.todos.filter((todo) =>
    todo.project === state.activeProject);
}
```

Cost: O(N)

```
{
  projects: ['Today', 'Work'],
  todos: [
    { id: 'todos/1',
      project: 'Today',
    },
    { id: 'todos/2',
      project: 'Work',
    }
  ]
}
```

Optimizing State Selectors

Reducing the computation required to transform state

TaskList.jsx

```
function selectActiveProject(state) {
  return state.todos[state.activeProject];
}
```

Cost: O(1)

```
{
  projects: ['Today', 'Work'],
  todos: {
    Today: [
      { id: 'todos/1' }
    ],
    Work: [
      { id: 'todos/2' }
    ]
  }
}
```

Ignore State Updates

Choosing when components should ignore state updates

QuickAdd.jsx

```
import { useProjectStore } from './store';
import { useShallow } from 'zustand/shallow';

const QuickAdd = () => {
  const addTodo = useProjectStore(
    useShallow(x => x.addTodo));

  const handleOnSubmit = (e) => {
    addTodo(e.values.todo);
  };

  return (
    <form onSubmit={handleOnSubmit}>
      {/* ... */}
    </form>
  )
}
```

Only trigger a re-render if `addTodo` changes

Memoizing State Selectors

Having state selectors "remember" their computed results from the same state

LeftNavigation.jsx

```
const selectProjectsWithCounts = (state) => {
  return state.projects.map((project) => ({
    project,
    count: state.todos[project].length,
  }));
};
```

Memoizing State Selectors

Having state selectors "remember" their computed results from the same state

LeftNavigation.jsx

```
const selectProjectsWithCounts = (state) => {
  return state.projects.map((project) => ({
    project,
    count: state.todos[project].length,
  }));
};
```

```
const selectProjectsWithActiveCounts = (state) => {
  return state.projects.map((project) => ({
    project,
    count: state.todos[project].filter(t => !t.completed).length,
  }));
};
```

Memoizing State Selectors

Having state selectors "remember" their computed results from the same state

LeftNavigation.jsx

```
const selectProjectsWithCounts = (state) => {
  return state.projects.map((project) => ({
    project,
    count: state.todos[project].length,
  }));
};

const selectProjectsWithActiveCounts = (state) => {
  return state.projects.map((project) => ({
    project,
    count: state.todos[project].filter(t => !t.completed).length,
  }));
};
```

Memoizing State Selectors

Having state selectors "remember" their computed results from the same state

LeftNavigation.jsx

```
const selectProjectsWithCounts = (state) => {
  return state.projects.map((project) => ({
    project,
    allCount: state.todos[project].length,
    activeCount: state.todos[project].filter(t => !t.completed).length,
  }));
};
```

Memoizing State Selectors

Having state selectors "remember" their computed results from the same state

selectors.js

```
import _ from 'lodash';

const selectCompletedTodosWithProject = (state) => {
  return _(state.todos)
    .flatMap((todos, project) =>
      _(todos)
        .filter({ completed: true })
        .map(todo => ({ ...todo, project }))
        .value()
    )
    .value();
}
```

```
{
  projects: ['Today', 'Work'],
  todos: {
    Today: [
      { id: 'todos/1' }
    ],
    Work: [
      { id: 'todos/2' }
    ]
  }
}
```

Memoizing State Selectors

Having state selectors "remember" their computed results from the same state

selectors.js

Consume transformed result

```
import _ from 'lodash';
import { createSelector } from 'reselect';

const selectCompletedTodosWithinLastWeek = createSelector(selectCompletedTodosWithProject, (todos) => {
  const ONE_WEEK_AGO = 7 * 24 * 60 * 60 * 1000;
  return todos.filter(todo => {
    return todo.completedAt > Date.now() - ONE_WEEK_AGO;
  });
});

const selectCompletedTodosWithProject = (state) => {
  return _(state.todos)
    .flatMap((todos, project) =>
      _(todos)
        .filter({ completed: true })
        .map(todo => ({ ...todo, project }))
        .value()
    )
    .value();
}
```

Component Design Scorecard

State Selectors

A pure function that can transform global state for local usage

Reduce Wasted Renders

Quantitative

- Less re-rendering ✓
- Less reconciliation ⊖
- Smaller footprint ✓
- Less memory usage ✗

Qualitative

- More flexibility ✓
- Promotes reuse ✓
- Less cognitive load ✓

State Slicing

store.js

State Slicing

store.js

```
{
  projects: {
    allIds: ['Today', 'Work']
  },
  todos: {
    allIds: ['todos/1', 'todos/2'],
    byId: {
      'todos/1': { body: … },
      'todos/2': { body: … }
    },
    byProject: {
      Today: ['todos/1'],
      Work: ['todos/2']
    }
  }
}
```

Cost: O(1)

Cost: O(1) [+ O(N)]

State Slicing

store.js

```
{
  projects: {
    allIds: ['Today', 'Work']
  },
  todos: {
    allIds: ['todos/1', 'todos/2'],
    byId: {
      'todos/1': { body: … },
      'todos/2': { body: … }
    },
    byProject: {
      Today: ['todos/1'],
      Work: ['todos/2']
    }
  }
}
```

Only stores project relationship

State Slicing

store.js

```
{
  todos: {
    allIds: ['todos/1', 'todos/2'],
    byId: {
      'todos/1': { body: … },
      'todos/2': { body: … }
    }
  }
}
```

State Slicing

store.js

```
{
  orgs: {
    allIds: ['Globomantics'],
    // ...
  },
  projects: {
    allIds: ['Today', 'Work'],
    byOrg: {
      Globomantics: ['Work']
    }
  },
  todos: {
    allIds: ['todos/1', 'todos/2'],
    byId: {
      'todos/1': { body: … },
      'todos/2': { body: … }
    },
    byProject: {
      Today: ['todos/1'],
      Work: ['todos/2']
    }
  }
}
```

Component Design Scorecard

State Slicing

Structure data by tailoring it to reduce data lookup and transformation costs

Reduce Wasted Renders

Quantitative

- Less re-rendering ✓
- Less reconciliation ⊖
- Smaller footprint ✓
- Less memory usage ⊖

Qualitative

- More flexibility ✓
- Promotes reuse ✓
- Less cognitive load ✓

```
import { create } from "zustand";

export const useProjectStore = create((set) => ({
  projects: ["Today"],
  activeProject: "Today",
  todos: {},
  addTodo: (text) =>
    set((state) => {
      const id = `todos/${state.todos[state.activeProject].length + 1}`;

      return {
        todos: {
          ...state.todos,
          [state.activeProject]: [
            ...state.todos[state.activeProject],
            {
              id,
              body: text,
            },
          ],
        },
      };
    }),
}));
```

Immutable Updates Example: Immer middleware

Create more consistent ways to mutate and update state, especially for nested objects

```
import create from 'zustand'
import { immer } from 'zustand/middleware/immer'

const useProjectStore = create(immer((set) => ({
  projects: ['Today', 'Work'],
  activeProject: 'Today',
  todos: {
    Today: [],
    Work: []
  },
  addTodo: (text) => set((state) => {
    // Generate new ID
    const projectTodos = state.todos[state.activeProject] || [];
    const newId = `todos/${projectTodos.length + 1}`;

    // Push new todo to the relevant project array
    if (!state.todos[state.activeProject]) {
      state.todos[state.activeProject] = [];
    }
    state.todos[state.activeProject].push({ id: newId, body: text });
  })
})));

export default useProjectStore;
```

Immutable Updates Example: Immer middleware

Create more consistent ways to mutate and update state, especially for nested objects

Component Design Scorecard

Immutable State Helpers

Make working with immutable state easier and more consistent by "drafting" state

Reduce Wasted Renders

Quantitative

- Less re-rendering
- Less reconciliation
- Smaller footprint
- Less memory usage

Qualitative

- More flexibility
- Promotes reuse
- Less cognitive load

Immer Alternatives

Immer alternatives and similar libraries

Based on the "Data Structure" category.
Alternatively, view Immer alternatives based on common mentions on social networks and blogs.

immutable-js

9.4 7.4 L3 Immer VS immutable-js

Immutable persistent data collections for Javascript which increase efficiency and simplicity.

Mock.js

9.0 0.0 Immer VS Mock.js

A simulation data generator

Power Real-Time Data Analytics at Scale

Get real-time insights from all types of time series data with InfluxDB. Ingest, query, and analyze billions of data points in real-time with unbounded cardinality.

Promo www.influxdata.com

immutability-helper

6.0 0.0 Immer VS immutability-helper

mutate a copy of data without changing the original source

JSON-Schema Faker

5.3 7.5 L5 Immer VS JSON-Schema Faker

JSON-Schema + fake data generators

mori

5.3 0.0 Immer VS mori

ClojureScript's persistent data structures and supporting API from the comfort of vanilla JavaScript

buckets

3.7 0.0 L4 Immer VS buckets

A complete, fully tested and documented data structure library written in pure JavaScript.

Recommended Learning

Reselect (Memoizing State Selectors)
https://github.com/reduxjs/reselect

Zustand Comparison Documentation
https://docs.pmnd.rs/zustand/getting-started/comparison

React Libraries – State Management
https://www.robinwieruch.de/react-libraries/#react-state-management

Backend-for-Frontend (BFF) with React and GraphQL
https://youtu.be/dCawL3nAnUA

Reducing DOM Updates with useRef and Uncontrolled Components

Two Methods of Managing the DOM

Controlled	VS	Uncontrolled
DOM reflects component state		DOM state is separated
Easier to reason about		Harder to reason about
Hands-off DOM rendering		Hands-on DOM rendering
Optimized but less control		Unoptimized but total control

Controlled Component

Useful in many scenarios with a predictable data flow

PlainTextEditor.jsx

```
const PlainTextEditor = ({ text, onEdit, notify = "on-change" }) => {
  const [value, setValue] = useState(text);

  const handleOnChange = useCallback(
    (e) => {
      setValue(e.target.value);
      if (notify === "on-change") {
        onEdit(e.target.value);
      }
    },
    [notify]
  );

  const handleOnBlur = useCallback(
    (e) => {
      if (notify === "on-blur") {
        onEdit(value);
      }
    },
    [notify, value]
  );

  return (
    <textarea value={value} onChange={handleOnChange} onBlur={handleOnBlur} />
  );
};
```

Controlled Component

Useful in many scenarios with a predictable data flow

PlainTextEditor.jsx

```
const PlainTextEditor = ({ text, onEdit, notify = "on-change" }) => {
  const [value, setValue] = useState(text);

  const handleOnChange = useCallback(
    (e) => {
      setValue(e.target.value);
      if (notify === "on-change") {
        onEdit(e.target.value);
      }
    },
    [notify]
  );

  const handleOnBlur = useCallback(
    (e) => {
      if (notify === "on-blur") {
        onEdit(value);
      }
    },
    [notify, value]
  );

  return (
    <textarea value={value} onChange={handleOnChange} onBlur={handleOnBlur} />
  );
};
```

Mostly inexpensive

Controlled Component

Not as useful when dealing with complex DOM handling

RichTextEditor.jsx

```
const RichTextEditor = ({ html, onEdit }) => {
  const [innerHtml, setInnerHtml] = useState(html);

  const handleOnChange = useCallback((e) => {
    setInnerHtml(e.target.innerHTML);
    onEdit(e.target.innerHTML);
  }, []);

  return (
    <div
      contentEditable={true}
      value={value}
      onChange={handleOnChange}
      __dangerouslySetInnerHTML={innerHtml}
    />
  );
};
```

This will cause problems

Component Design Scorecard

Uncontrolled Components

Opt out of React handling state or managing DOM updates with useRef

Reduce Wasted Renders

Quantitative

- Less re-rendering ✓
- Less reconciliation ✓
- Smaller footprint ✓
- Less memory usage ✓

Qualitative

- More flexibility ✗
- Promotes reuse ✗
- Less cognitive load ✗

useRef Hook

Opt out of React managing state and create a mutable value

RichTextEditor.jsx

```
const RichTextEditor = ({ html, onEdit }) => {
  const editorRef = useRef(null);

  useEffect(() => {
    if (editorRef.current) {
      editorRef.current.innerHTML = html;
    }
  }, [html]);

  const handleOnChange = useCallback((e) => {
    const currentHtml = e.target.innerHTML;
    onEdit(currentHtml);
  }, [onEdit]);

  return (
    <div
      ref={editorRef}
      contentEditable={true}
      onChange={handleOnChange}
    />
  );
};
```

Will not cause re-renders

```
{
  current: null
}
```

useRef Hook

Opt out of React managing state and create a mutable value

RichTextEditor.jsx

```
const RichTextEditor = ({ html, onEdit }) => {
  const editorRef = useRef(null);

  useEffect(() => {
    if (editorRef.current) {
      editorRef.current.innerHTML = html;
    }
  }, [html]);

  const handleOnChange = useCallback((e) => {
    const currentHtml = e.target.innerHTML;
    onEdit(currentHtml);
  }, [onEdit]);

  return (
    <div
      ref={editorRef}
      contentEditable={true}
      onChange={handleOnChange}
    />
  );
};
```

Will not cause re-renders

```
{
  current: value
}
```

You can modify this as much as you want (i.e. mutable)

useRef Hook

Opt out of React managing state and create a mutable value

RichTextEditor.jsx

```
const RichTextEditor = ({ html, onEdit }) => {
  const editorRef = useRef(null);

  useEffect(() => {
    if (editorRef.current) {
      editorRef.current.innerHTML = html;
    }
  }, [html]);

  const handleOnChange = useCallback((e) => {
    const currentHtml = e.target.innerHTML;
    onEdit(currentHtml);
  }, [onEdit]);

  return (
    <div
      ref={editorRef}
      contentEditable={true}
      onChange={handleOnChange}
    />
  );
};
```

Will not cause re-renders

```
{
  current: null
}
```

useRef Hook

Opt out of React managing state and create a mutable value

RichTextEditor.jsx

```
const RichTextEditor = ({ html, onEdit }) => {
  const editorRef = useRef(null);

  useEffect(() => {
    if (editorRef.current) {
      editorRef.current.innerHTML = html;
    }
  }, [html]);

  const handleOnChange = useCallback((e) => {
    const currentHtml = e.target.innerHTML;
    onEdit(currentHtml);
  }, [onEdit]);

  return (
    <div
      ref={editorRef}
      contentEditable={true}
      onChange={handleOnChange}
    />
  );
};
```

Will not cause re-renders

```
{
  current: HTMLDivElement
}
```

useRef Hook

Opt out of React managing state and create a mutable value

RichTextEditor.jsx

```
const RichTextEditor = ({ html, onEdit }) => {
  const editorRef = useRef(null);

  useEffect(() => {
    if (editorRef.current) {
      editorRef.current.innerHTML = html;
    }
  }, [html]);

  const handleOnChange = useCallback((e) => {
    const currentHtml = e.target.innerHTML;
    onEdit(currentHtml);
  }, [onEdit]);

  return (
    <div
      ref={editorRef}
      contentEditable={true}
      onChange={handleOnChange}
    />
  );
};
```

Synchronize with DOM

useRef Hook

Opt out of React managing state and create a mutable value

RichTextEditor.jsx

```
const RichTextEditor = ({ html, onEdit }) => {
  const editorRef = useRef(null);

  useEffect(() => {
    if (editorRef.current) {
      editorRef.current.innerHTML = html;
    }
  }, [html]);

  const handleOnChange = useCallback((e) => {
    const currentHtml = e.target.innerHTML;
    onEdit(currentHtml);
  }, [onEdit]);

  return (
    <div
      ref={editorRef}
      contentEditable={true}
      onChange={handleOnChange}
    />
  );
};
```

"Pull" DOM values

useRef Hook

Opt out of React managing state and create a mutable value

RichTextEditor.jsx

```
const RichTextEditor = ({ html, onEdit }) => {
  const editorRef = useRef(null);

  useEffect(() => {
    if (editorRef.current) {
      editorRef.current.innerHTML = html;
    }
  }, [html]);

  const handleOnChange = useCallback((e) => {
    const currentHtml = e.target.innerHTML;
    onEdit(currentHtml);
  }, [onEdit]);

  return (
    <div
      ref={editorRef}
      contentEditable={true}
      onChange={handleOnChange}
    />
  );
};
```

Vulnerable to XSS attacks

Dangerously Setting HTML

Will bypass encoding and passthrough raw HTML

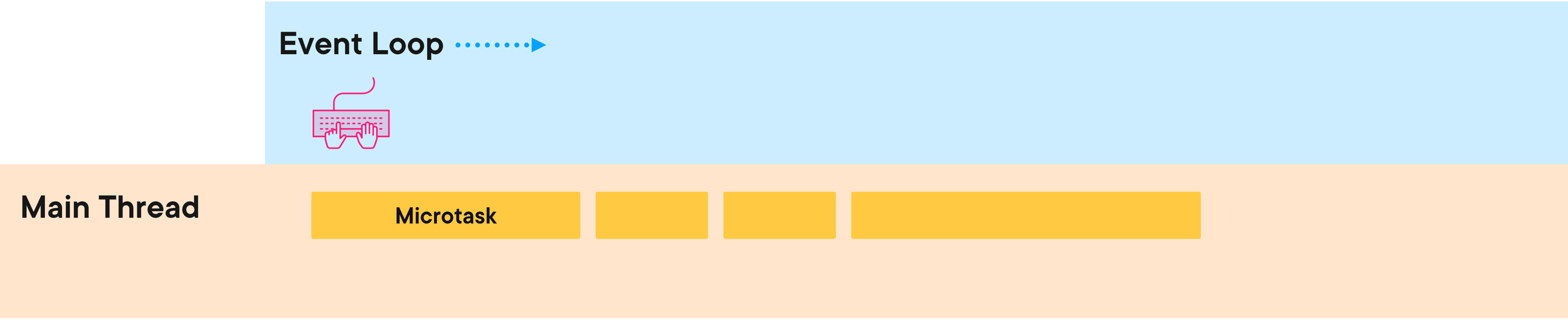
RichTextEditor.jsx

```
const RichTextEditor = ({ html, onEdit }) => {
  const [innerHtml, setInnerHtml] = useState(html);

  const handleOnChange = useCallback((e) => {
    setInnerHtml(e.target.innerHTML);
    onEdit(e.target.innerHTML);
  }, []);

  return (
    <div
      contentEditable={true}
      value={value}
      onChange={handleOnChange}
      __dangerouslySetInnerHTML={innerHtml}
    />
  );
};
```

XSS attack vector

Prioritizing Urgent State Updates with Suspense

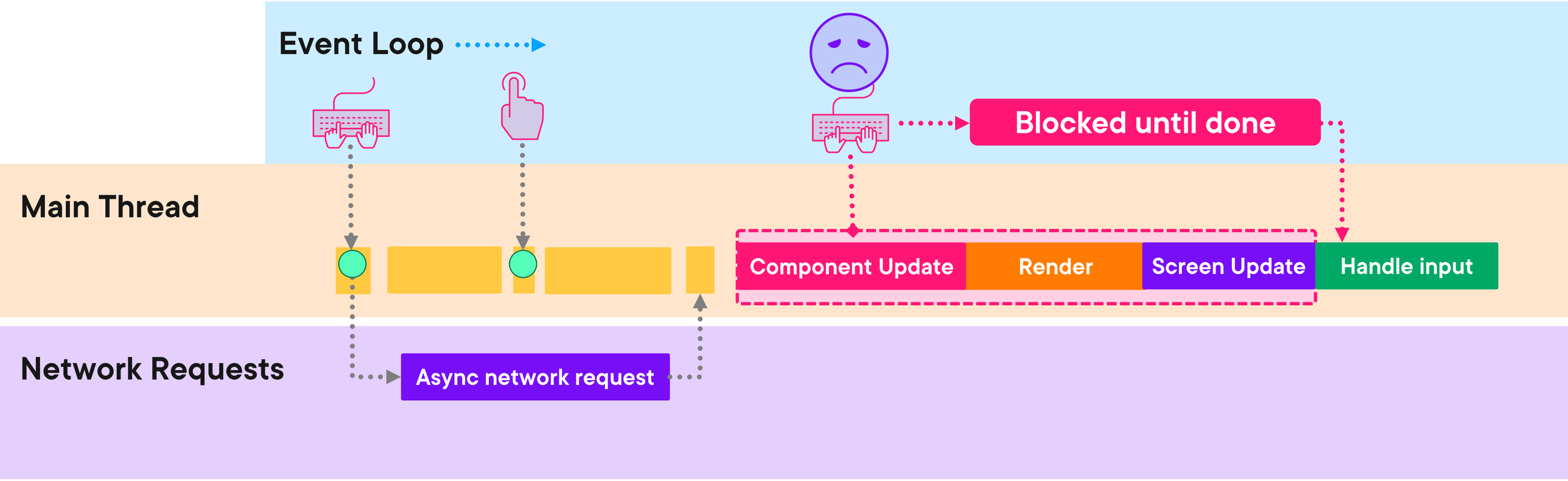
React Scheduling Visualized



For illustration only: diagram is not to scale

JavaScript can only do one thing at a time

React Scheduling Visualized



For illustration only: diagram is not to scale

Component Design Scorecard

Deferred Rendering

Unblock user input by interrupting non-urgent updates and deferring them

Reduce Wasted Renders

Quantitative

- Less re-rendering ✓
- Less reconciliation ✓
- Smaller footprint ⊖
- Less memory usage ✓

Qualitative

- More flexibility ✓
- Promotes reuse ⊖
- Less cognitive load ✗

Learn About Suspense

Working with Data in React

Pluralsight Course

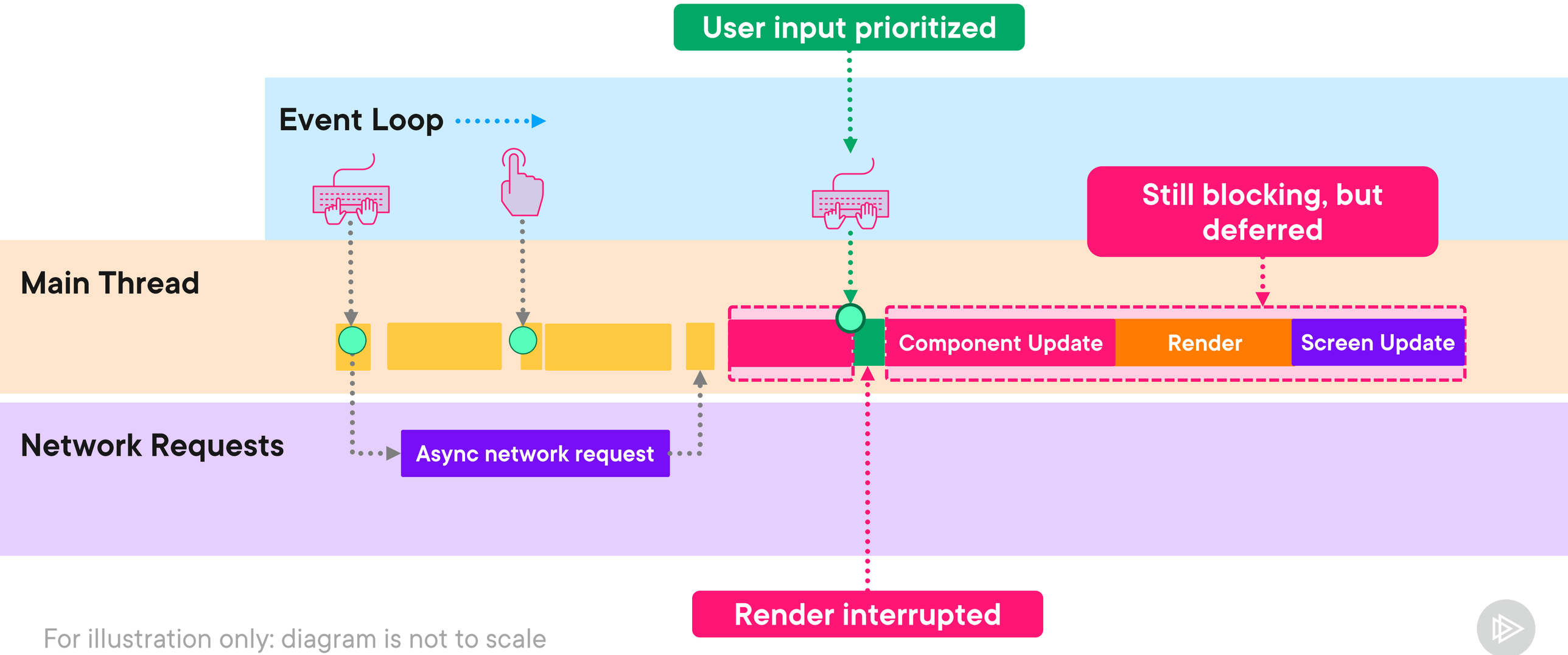
Handling Rapid Input

Avoiding a choppy experience when the user is typing rapidly

SearchPage.jsx

```
import { useState } from 'react';

function SearchPage() {
  const [query, setQuery] = useState('');

  const handleInputChange = (event) => {

    // Happens on every keypress
    setQuery(event.target.value);

    // Fire off our API call
    // ...
  }

  // Rest of component
  // ...
}
```

Prioritizing Urgent Updates



For illustration only: diagram is not to scale

useDeferredValue

Re-render updates in the background and only return the latest version

SearchPage.jsx

```
import { useState, useDeferredValue } from 'react';

function SearchPage() {
  const [query, setQuery] = useState('');
  const deferredQuery = useDeferredValue(query);
  // ...
}
```

useDeferredValue

Re-render updates in the background and only return the latest version

SearchPage.jsx

```
import { useState, useDeferredValue } from 'react';

function SearchPage() {
  const deferredValue = useDeferredValue(???);
  // ...
}
```

useTransition

Mark an update manually as non-urgent

SearchPage.jsx

```
import { useState, useTransition } from 'react';

function SearchPage() {
  const [query, setQuery] = useState('');
  const [isPending, startTransition] = useTransition();

  const handleInputChange = (event) => {
    startTransition(() => {
      // Happens on every keypress
      setQuery(event.target.value);
    });
  }

  // Rest of component
  // ...
}
```

startTransition

Standalone API outside React components to mark an update as non-urgent

SearchPage.jsx

```
import { startTransition } from 'react';
import { navigate } from 'third-party-router';

function goToScreen(screen) {
  startTransition(() => {
    navigate(screen);
  });
}
```

Recommended Learning

Visualizing the JavaScript Event Loop
http://latentflip.com/loupe

What the Heck is the Event Loop Anyway?
https://www.youtube.com/watch?v=8aGhZQkoFbQ

useDeferredValue Documentation
https://react.dev/reference/react/useDeferredValue

Summary

Our principal objective with efficient design is reducing wasted renders

Wasted renders can be reduced at the component level through optimizing hierarchies, sharing state, and isolating updates

React doesn't dictate optimizations but instead lets you "choose your own adventure" which can be frustrating to navigate

Use the Lean React Scorecard to help evaluate different approaches

Component Design Scorecard

Pattern or Practice

Brief explanation

Reduce Wasted Renders

Quantitative

- Less re-rendering
- Less reconciliation
- Smaller footprint
- Less memory usage

Qualitative

- More flexibility
- Promotes reuse
- Less cognitive load

Promotes | Detracts | Depends | Neither

React Performance Scorecard
Find more at bit.ly/PSReactPerfScorecard

View All the Scorecards

Find all the scorecards in this module as a bundled PDF to share with your team or extend on your own