Agenda

- **Navigation.**
- **The Virtual DOM**
- **Design patterns.**
- **Custom Hooks.**

Navigation

The React Router

Introduction

- **Allows** multiple views **and** flows **in an app.**

- **Keeps the URL in sync with the UI.**

- **Supports traditional web principles:**
  1. **Addressability.**
  2. **Information sharing.**
  3. **Deep linking.**
  - **1st generation AJAX apps violated these principles.**

- **Not part of the React framework - A separate library.**

Basic routing configuration

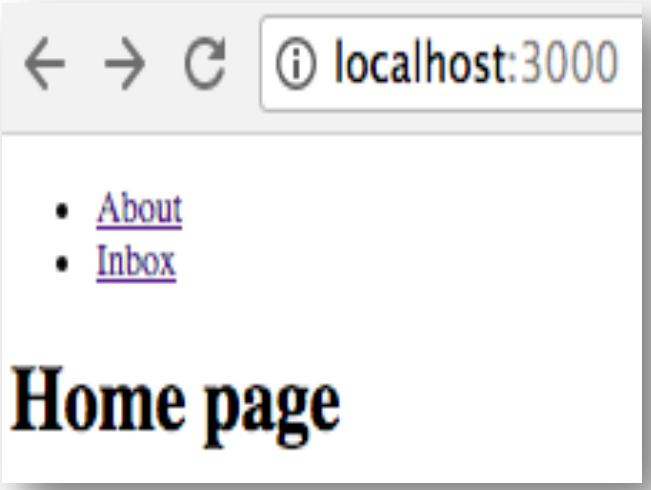
	URL	Components
1	/	Home
2	/about	About
3	/inbox	Inbox

```
17  const App = () => {
18    return (
19      <BrowserRouter>
20        <Switch>
21          <Route path="/about" component={About} />
22          <Route path="/inbox" component={Inbox} />
23          <Route exact path="/" component={Home} />
24          <Redirect from="*" to="/" />
25        </Switch>
26      </BrowserRouter>
27    );
28  };
29
30  ReactDOM.render(<App />, document.getElementById("root"));
```

- **Declarative routing.**
- <BrowserRouter**> - matchs browser's URL with a** <Route> path**.**
- **Matched** <Route> **declares** component **to be mounted.**
- <Route> path **supports regular expression pattern matching.**
  - **Use** exact **argument for precision.**
- **Use** <Redirect> to **avoid 404-type error.**
- <Switch> - **only one of the nested Routs can be active.**
- ReactDOM.render() **passed an app's Router component**.
- **Ref.** src/sample1/. **(see lecture archive)**

Hyperlinks

- **Use the** <Link> **component for internal links.**
  - **Use anchor tag for external links -** <a href . . . . . >
- **Ref**. src/sample2/




```
6   const Home = () => {
7     return (
8       <>
9         <ul>
10          <li>
11            <Link to="/about">About</Link>
12          </li>
13          <li>
14            <Link to="/inbox">Inbox</Link>
15          </li>
16        </ul>
17        <h1>Home page</h1>
18      </>
19    );
20  };
```

Absolute URL

- <Link> **gives us access to other useful router properties.**
- **Use** <LinlContainer> **when link wraps other 3rd party component, e.g. Bootstrap-React <Button />**

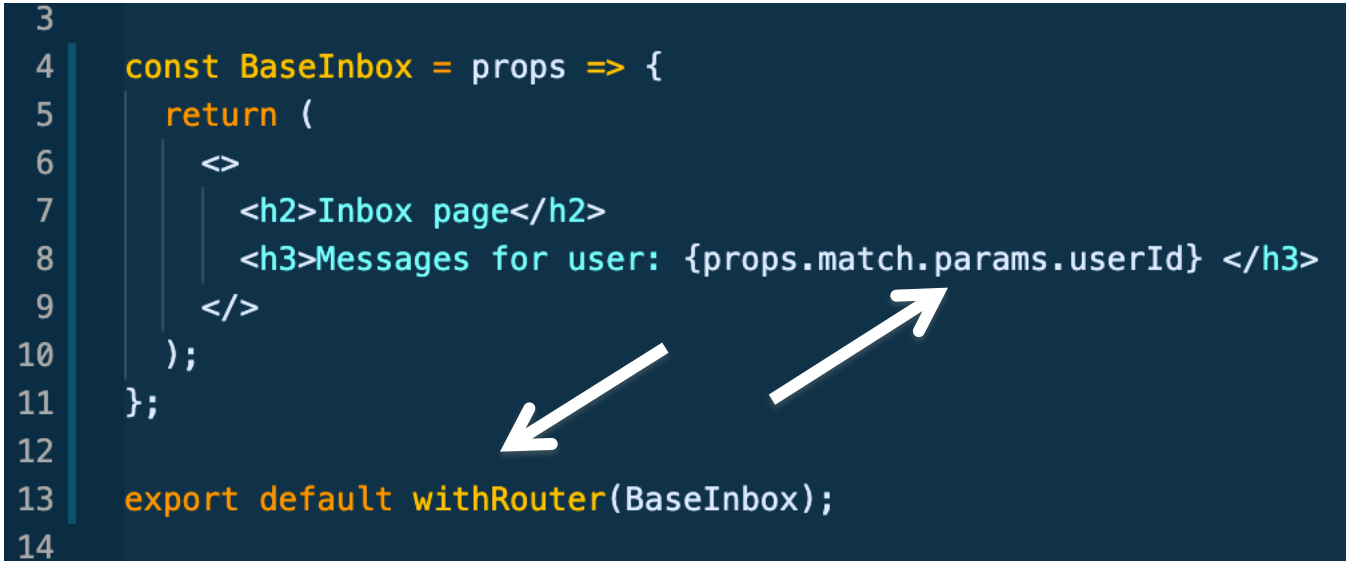
Dynamic segments.

- Parameterized URLs**, e.g. /users/***22***, /users/***12***/purchases**
  - **How do we declare a parameterized path in the routing configuration?**
  - **How does a component access the parameter value?**
- **Ex: Suppose the** Inbox **component shows messages for a specific user, where the user's id is part of the browser URL**

  e.g /inbox/123 where 123 is the user's id.

- **Solution**: <Route path='/inbox/:userId' component={ Inbox } />
  - **The colon (:) prefixes a parameter in the path; Parameter name is arbitrary.**
  - **Ref src/sample3**

Dynamic segments.



- withRouter() **function: Returns a new, enriched component.**
  - **Injects routing props into a component:**
    - props.match.params.(parameter-name)
    - props.history
- **More than one parameter allowed.**

  e.g. /users/:userId/categories/:categoryName

Nested Routes

- **Objective: A component's child is dynamically determined from the browser's URL (Addressability).**
- **EX.: (See** src/sample4**) Given the route:**

  <Route path='/inbox/:userId' component={ Inbox } />,

  when the browser URL is:

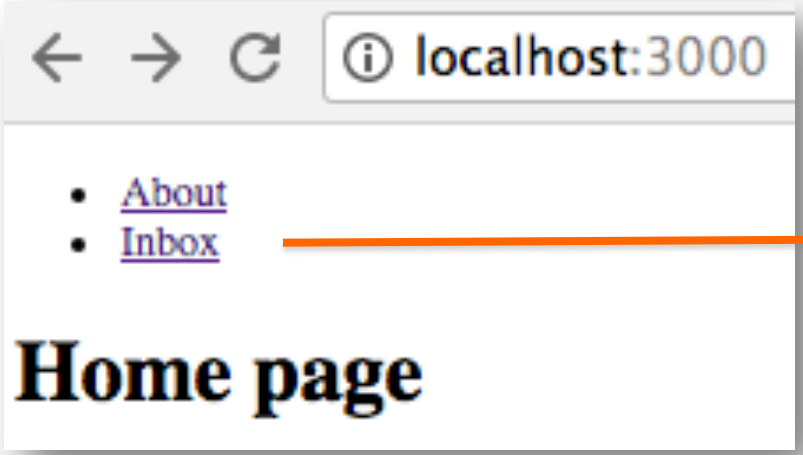
  1. /inbox/XXX/statistics **render Inbox +** Stats **components.**
  2. /inbox/XXX/draft **then render Inbox +** Drafts **components.**

```
const BaseInbox = (props) => {
  return (
    <>
      <h1>Inbox page</h1>
      <Messages id={props.match.params.userId} />
      <Route path={`/inbox/:userId/statistics`} component={Stats} />
      <Route path={`/inbox/:userId/drafts`} component={Draft} />
    </>
  );
};
```

Nested routes

Extended <Link>

- **Objective: Pass additional props via a <Link>.**
- **EX.: See** /src/sample5/**.**



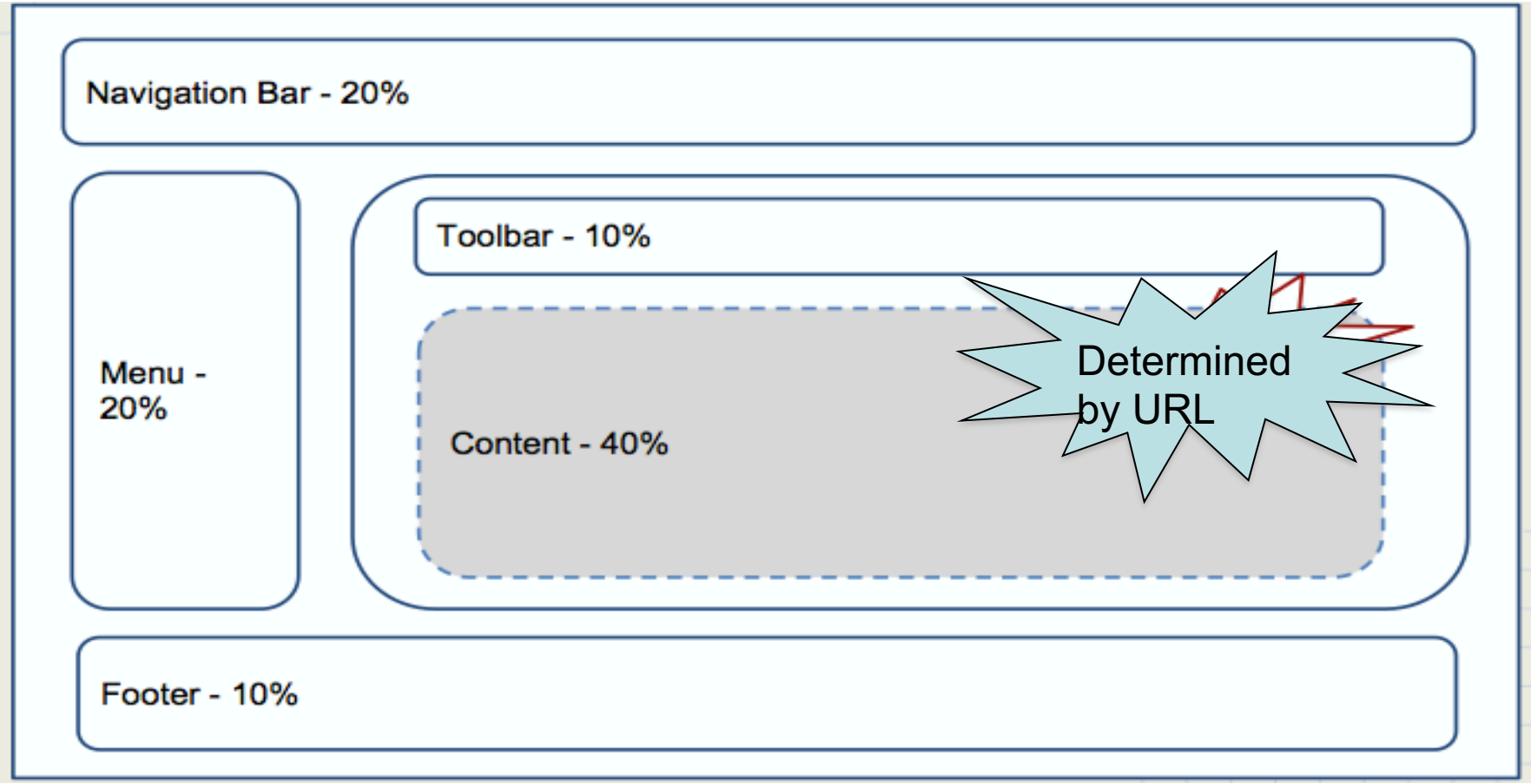

```
const userId = 'id1234'
const beta = 'something else'

<Link
    to={{
    pathname: "/inbox",
    state: {
        user: userId,
        beta: beta
    }
    }}
>Inbox </Link>
```

```
const Inbox = props => {
  const { user, beta } = props.location.state;
  return (
    <>
      <h2>Inbox page</h2>
      <p>{`Link Props: ${user}, ${beta}`}</p>
    </>
  );
};
```

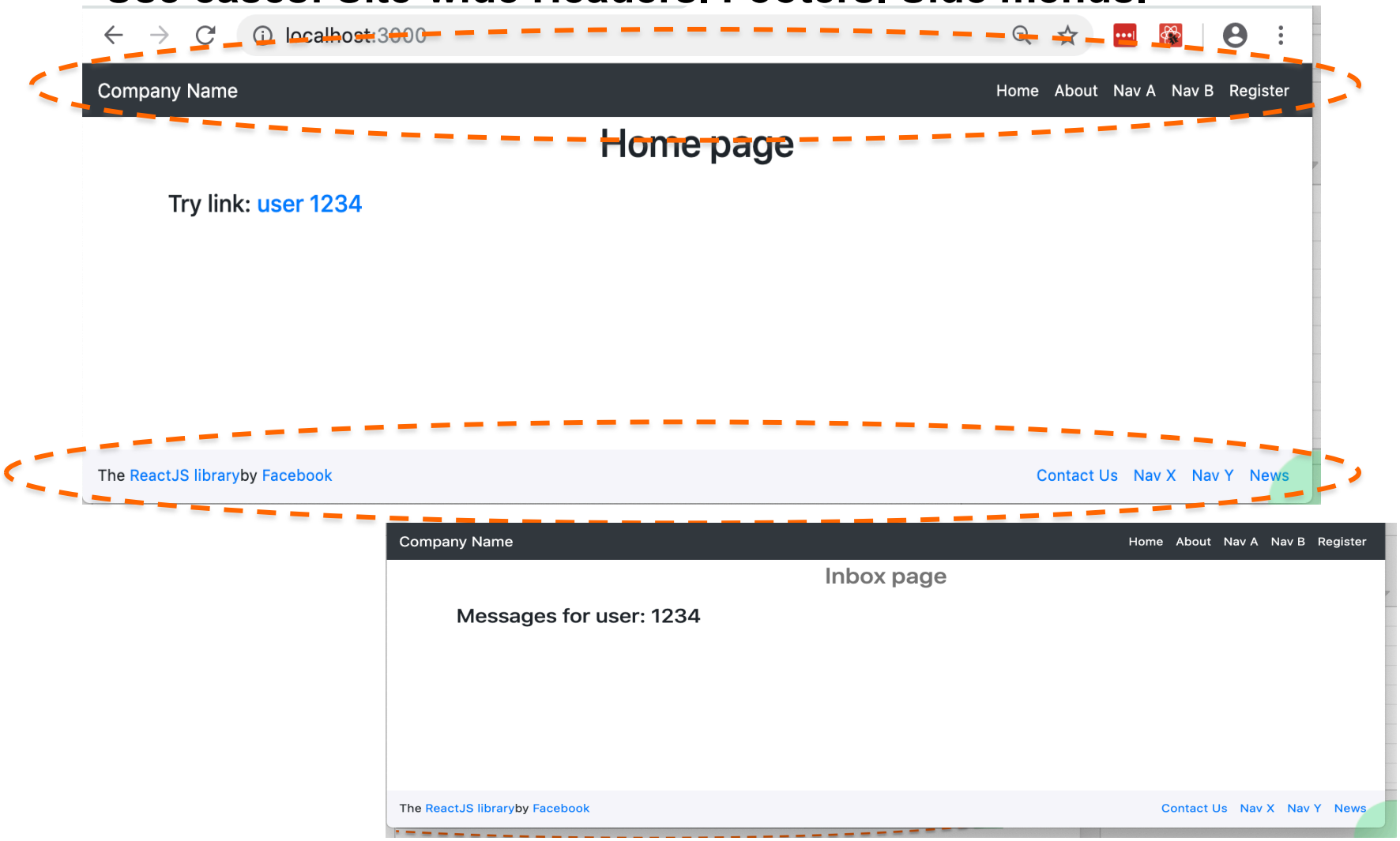
```
<Route path="/inbox" component={Inbox} />
```

Typical Web app layout



Persistent elements/components

- **Use cases: Site-wide Headers. Footers. Side menus.**



Persistent elements/components

- **Ref.** src/sample6

```
class Router extends Component {
    render() {
        return (
            <BrowserRouter>
                <div>
                <Header/>
                <div className="container">
                    <Switch>
                        <Route path='/about' component={ About } />
                        <Route path='/register' component={ Register } />
                        <Route path='/contact' component={ Contact } />
                        <Route path='/inbox/:userId' component={ Inbox } />
                        <Route exact path='/' component={ Home } />
                        <Redirect from='*' to='/' />
                    </Switch>
                </div>
                <Footer />
                </div>
            </BrowserRouter>
        )
    }
}
```

Routing.

- **The remaining routing samples the archive will be discussed in later lectures.**

The Virtual DOM

Modifying the DOM

- **DOM – an internal data structure representing the browser's current 'display area' ; DOM always in sync with the display.**

- **Traditional performance best practice:**
  1. **Minimize access to the DOM.**
  2. **Avoid expensive DOM operations.**
  3. **Update elements offline, then reinsert into the DOM.**
  4. **Avoid changing layouts in Javascript.**
  5. **. . . etc.**

- **Should the developer be responsible for low-level DOM optimization? Probably not.**
  - **React provides a *Virtual DOM* to shield developer from these concerns.**

The Virtual DOM

- **How React works:**
  1. **It create a lightweight, efficient form of the DOM – the Virtual DOM.**
  2. **Your app changes the V. DOM via components' JSX.**
  3. **React engine:**
     1. **Perform a *diff* operation between current and previous V. DOM state.**
     2. **Compute the set of changes to apply to real DOM.**
     3. **Batch update the real DOM.**

- **Benefits:**
  a) **Cleaner, more descriptive programming model.**
  b) **Optimized DOM updates and reflows.**

Virtual DOM – *Pre-commit & Commit phases*

- **EX.: The Counter component.**

User clicks button

→ onClick event handler executed

→ component state is changed

→ component re-executed (re-renders)

→ The Virtual DOM has changed.

→ (Pre=commit) Compute changes between the current and previous Virtual DOM (Diff).

→ (Commit) Batch updates the Real DOM.

→ Browser repaints screen

Virtual DOM – *Performance*

- **The Filter Friends App.**

User types a character in text box

→ onChange event handler executes

→ Handler changes state (FriendsApp component)

→ Re-renders FriendsApp.

→ Re-renders children (FilteredFriendList) with new prop values.

→.Re-renders children of FilteredFriendList.

(Re-rendering completed)

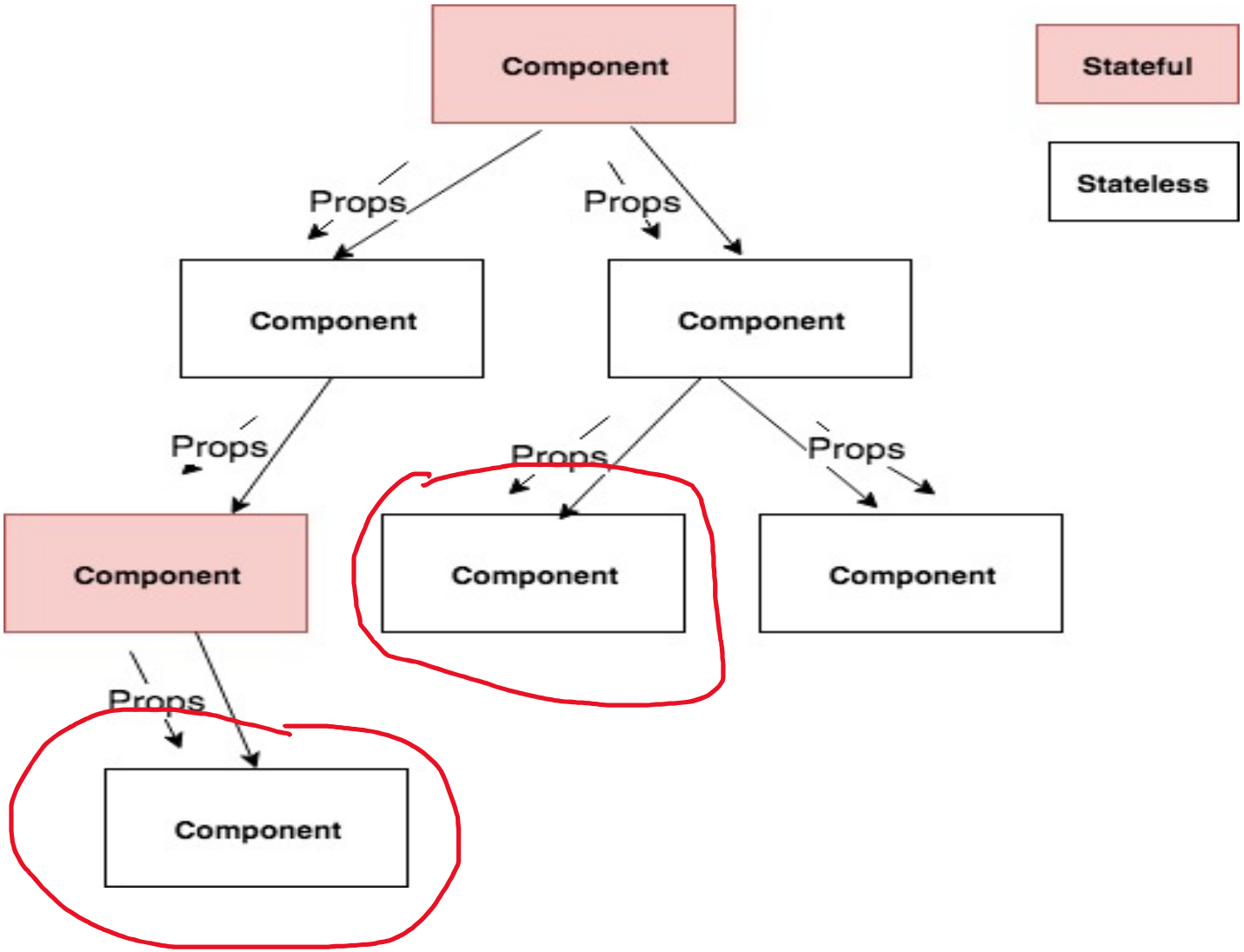
→ (Pre-commit) Computes the new Virtual DOM

+ Perform diff on new and previous Virtual DOMs.

→ Commit phase) Batch updates the Real DOM.

→ Browser repaints screen

Virtual DOM – *The Diff operation*



Design Patterns

In software engineering, a **design pattern** is a general repeatable solution to a commonly occurring problem in software **design**

Reusability & Separation of Concerns.

- **The DRY principle – Don't Repeat Yourself.**
- **Techniques to improve DRY (increase reusability):**
  1. **Inheritance (is-a relationships, e.g. Car is an automabile)**
  2. **Composition (has-a relationships, e.g. Car has an Engine)**

- **React favors** composition**. - We build components from other components, of varying complexity and specialization through** props**. Generalized components are used in building many other components.**

- **Core React composition Patterns:**
  1. **Containers.**
  2. **Render Props.**
  3. **Higher Order Components.**

What are children

```
<div>
    <h2>Some Heading</h2>
    <ul>
        <li> . . . . . </li>
        <li> . . . . . </li>
        <li> . . . . . </li>
    </ul>
</div>
```

- **div has two children; ul has three children**

The Container pattern.

All React components have a special children prop so that consumers can pass components directly by nesting them inside the jsx.

```
const Picture = (props) => {
  return (
    <div>
      <img src={props.src}/>
      {props.children}
    </div>
  )
}
```

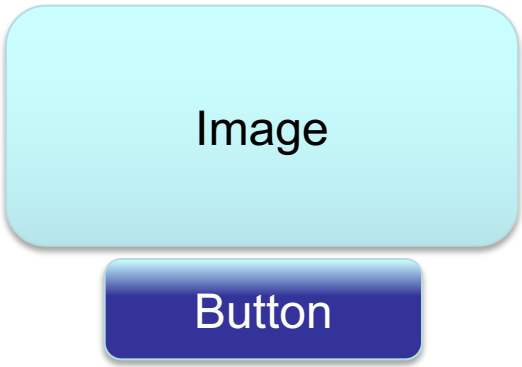
```
const OtherComponent = props => {
  . . . . . .
  return (
    <div className='container'>
      <Picture src={picture.src}>
        // what is placed here is
        // passed as props.children
      </Picture>
    </div>
  )
}
```

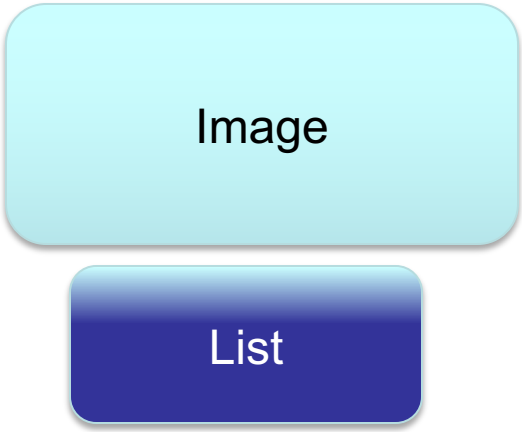
- **A Picture component displays its own JSX as well as that passed in by the consumer component from the children prop.**

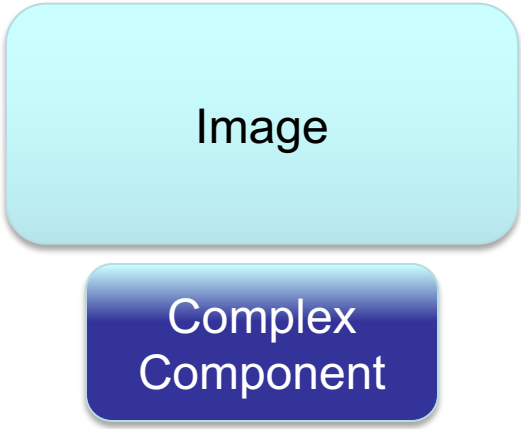
```
const OtherComponent1 = props => {
  . . . . . .
  return (
    <div className='container'>
      <Picture src={picture.src}>
          <button>.......</button>
      </Picture>
    </div>
  )
}
```

```
const OtherComponent2 = props => {
  . . . . . .
  return (
    <div className='container'>
      <Picture src={picture.src}>
          <ul>. . . . . </ul>
      </Picture>
    </div>
  )
}
```

```
const OtherComponent3 = props => {
  . . . . . .
  return (
    <div className='container'>
      <Picture src={picture.src}>
          <ComplexComponewnt>
            . . . . . .
          </ComplexComponewnt>
      </Picture>
    </div>
  )
```







Picture is **composed** with other elements / components

The Container pattern.

- **Benefits:**

1. **Greater reusability – Generalized components that can be specialized with props (children)**
2. **Cleaner, simpler codebase – Instead of many specialized components, create fewer generalized components.**
3. **Improved de-coupling between parent and child.**

- **More on patterns in a subsequent lecture.**

Custom Hooks

Custom Hooks.

- **Custom Hooks let you extract component logic into** reusable **functions.**
- **Improves code readability and modularity.**

Example:

```
const BookPage = props => {
  const isbm = props.isbn;
  const [book, setBook] = useState(null);
  useEffect(() => {
    fetch(
    `https://api.for.books?isbn=${isbn}`)
    .then(res => res.json())
    .then(book => {
      setBook(book);
    });
  }, [isbn]);
  . . . .rest of component code . . . .
}
```

Objective – Extract the book-related state code into a custom hook.

Custom Hook Example.

Solution:

```
const useBook = isbn => {
  const [book, setBook] = useState(null);
  useEffect(() => {
     fetch(
    `https://api.for.books?isbn=${isbn}`)
    .then(res => res.json())
    .then(book => {
      setBook(book);
    });
  }, [isbn]);
   return [book, setBook];
};
```

```
const BookPage = props => {
  const isbm = props.isbn;
  const [book, setBook] = useBook(isbn);

  . . . .rest of component code . . . .
}
```

- **Custom Hook is an ordinary function BUT can only be called from a React component function.**
- **Prefix hook function name with** use **to leverage linting support.**
- **Function can return any collection type (array, object), with any number of entries.**