

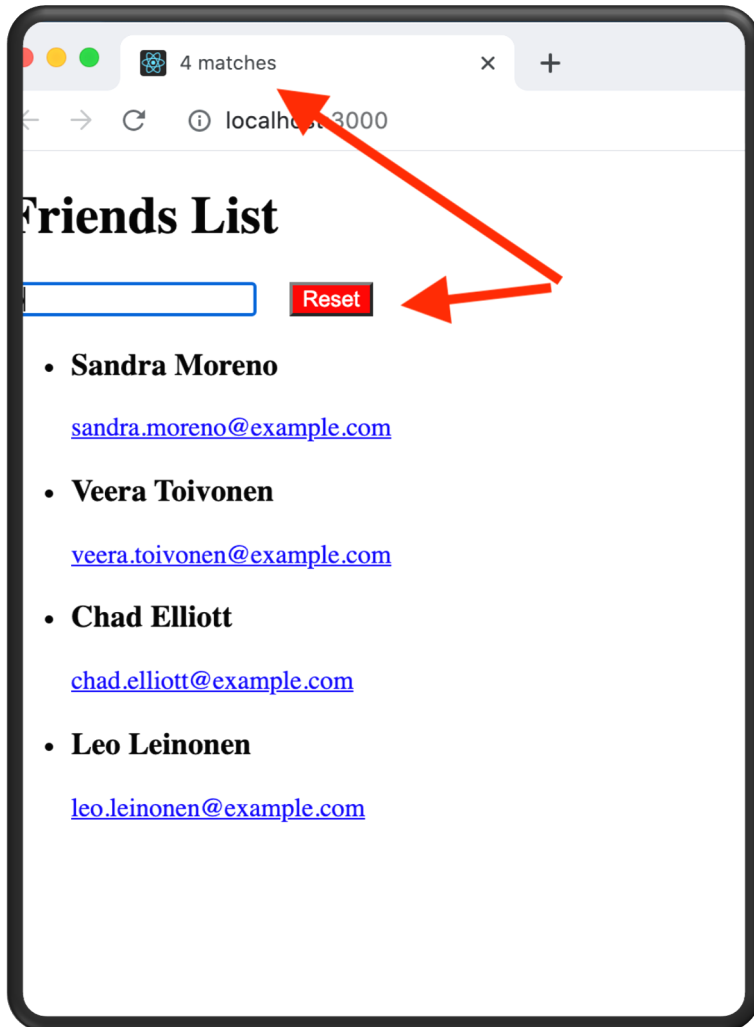
# ReactJS.

The Component model (Contd)

# Topics

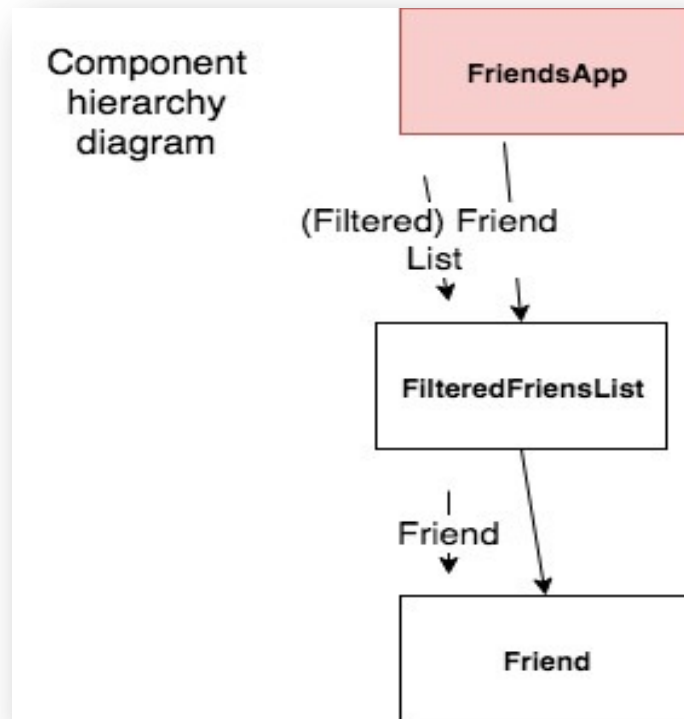
- **Hooks and Component Lifecycle.**
- **Data Flow patterns – Data Down, Action Up pattern.**
- **The Virtual DOM**

# Sample App 1 – Version 2



- **App UI changes:**
  1. A 'Reset' button – loads a new list of friends. Overwriting the current list.
  2. Browser tab title - shows # of matching friends (side effect).
- See lecture archive for source code

# Sample App 1 (v2) - Design



- **3 state variables:**
  1. **List of friends from API.**
  2. **Text box content.**
  3. *Reset button toggle.*
- **2 side effects:**
  1. **'Fetch API data' - dependent on change to reset button toggle.**
  2. **'Set browser tab title' - dependent on change to matching list length.**

# Sample App 1 (v2) - Events

The image shows a side-by-side comparison of a web application and its Redux state management logs. On the left is the application interface, and on the right is the Redux DevTools log.

**Application Interface (Left):**

- Friends List**
- Search input:
- Reset** button
- Friends list:
  - **Dennis Allen**  
[dennis.allen@example.com](mailto:dennis.allen@example.com)
  - **Valerie Welch**  
[valerie.welch@example.com](mailto:valerie.welch@example.com)
  - **Tobias Thomsen**  
[tobias.thomsen@example.com](mailto:tobias.thomsen@example.com)
  - **Gissele Oliveira**

**Redux DevTools Log (Right):**

- [HMR] Waiting for update signal from WDS...
- Initial mounting**
  - Render FriendsApp
  - fetch effect
  - set title effect
  - Render FriendsApp
  - set title effect
- After typing 1 character**
  - Render FriendsApp
  - set title effect
- After clicking reset button**
  - Render FriendsApp
  - fetch effect
  - Render FriendsApp
  - set title effect

# Sample App 1 - Events.

- **On mounting of FriensApp component:**  
**Both effects execute (Set browser tab to '0 matches').**
  - **'Fetch data' effect changes 'friends list' state.**
  - **Component re-renders + 'Set browser tab' effect executes.**
- **On typing a character in the text box:**  
**'Text box' state change.**
  - **FriendsApp rerenders + Matching friends list length changes**
  - **'Set browser title' effect executes.**
- **On clicking Reset button:**  
**'Reset toggle' state changes.**
  - **FriendsApp re-renders.**
  - **'Fetch data' effect executes.**
  - **'Friends list' state changes.**
  - **FriendsApp re-renders + Matching list length changes.**
  - **'Set browser title' effect executes.**

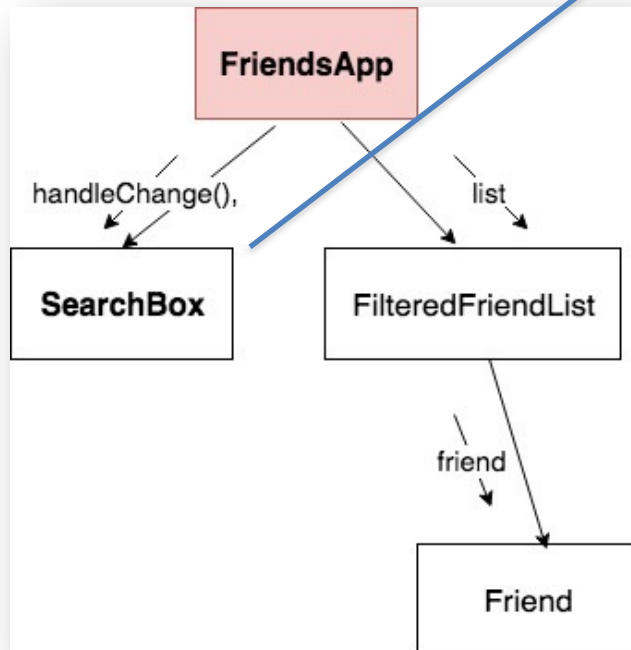
# Topics

- **Hooks and Component Lifecycle.**
- **Data Flow patterns – Data Down, Action Up pattern.**
- **The Virtual DOM**

# Sample App 2

(Data down, actions up pattern or Inverse data flow pattern )

- **What if a component's state is influenced by an event in a subordinate component?**
- **Solution:** The data down, action up pattern.



## Friends List

Search

- **Joe Bloggs**  
[jbloggs@here.con](mailto:jbloggs@here.con)
- **Paula Smith**  
[psmith@here.con](mailto:psmith@here.con)
- **Catherine Dwyer**  
[cdwyer@here.con](mailto:cdwyer@here.con)
- **Paul Briggs**  
[pbriggs@here.con](mailto:pbriggs@here.con)



# Data down, Action up.

## Pattern:

1. Stateful component (FriendsApp) provides a callback to the subordinate (SearchBox).
2. Subordinate invokes callback when the event (onChange) occurs.

```
const FriendsApp = () => {  
  const [searchText, setSearchText] = useState("");  
  const [friends, setFriends] = useState([]);  
  
  useEffect(() => { ...  
  }, []);  
  
  const filterChange = text =>  
    setSearchText(text.toLowerCase());  
  
  const updatedList = friends.filter(friend => { ...  
  });  
  return (  
    <>  
      <h1>Friends List</h1>  
      <SearchBox handleChange={filterChange} />  
      <FilteredFriendList list={updatedList} />  
    </>  
  );  
}
```

```
const SearchBox = props => {  
  const onChange = event => {  
    event.preventDefault();  
    const newText = event.target.value.toLowerCase();  
    props.handleChange(newText);  
  };  
  
  return <input type="text" placeholder="Search"  
    onChange={onChange} />;  
};
```

# Topics

- **Hooks and Component Lifecycle.**
- **Data Flow patterns – Data Down, Action Up pattern.**
- **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 direct accessing of the **DOM**.
  2. Avoid 'expensive' **DOM** operations.
  3. Update elements offline, then replace in 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 developers from these concerns.

# The Virtual DOM

- **How React works:**
  1. It create a lightweight, efficient form of the DOM, termed the *Virtual DOM*.
  2. Your app changes the V. DOM via components' JSX.
  3. React engine:
    1. Performs a *diff* operation between current and previous V. DOM instance.
    2. Computes 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.

# Automatic Re-rendering (detail)

- **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*

*→ React diffs the changes between the current and previous Virtual DOM*

*→ React batch updates the Real DOM*

# Re-rendering & the real DOM

- **What happens when the user types in the text box?**

***User types a character in text box***

***→ onChange event handler executes***

***→ Handler changes a state variable***

***→ React re-renders FriendsApp component***

***→ React re-renders children (FilteredFriendList) with new prop values.***

***→ React re-renders children of FilteredFriendList.  
(Re-rendering completed)***

***→ (Pre-commit phase) React computes the updates required to the browser's DOM***

***→ (Commit phase) React batch updates the DOM.***

***→ Browser repaints screen***

# Summary

- **A state variable change always causes a component to re-render.**
  - **State change logic is usually part of an event handler function.**
  - **Event handler may be in a subordinate component.**
- **Side effects:**
  - **Always execute at mount time.**
  - **The dependency array will either reference a state variable, a value computed from a state variable, or a prop.**
    - **Can be multiple entries**
  - **Callback performs the side-effect, and may also cause a state change.**
- **Data flows downward, actions flow upward.**

