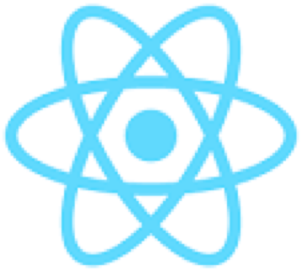


ReactJS.

The Component model

Topics

- **Component State.**
  - **Basis for dynamic, interactive UI.**
- **The Virtual DOM.**
- **Data Flow patterns.**
- **Hooks.**

Component DATA

- **Two sources of data for a component:**
  1. **Props - Passed in to a component; Immutable; the** props **object.**
  2. *State* **- Managed internally by the component; Mutable; can be any data type (primitive, array, object)**
     - ***** The basis for dynamic and interactive Uis *****

- **Props-related features:**
  - **Default values.**
  - **Type-checking.**

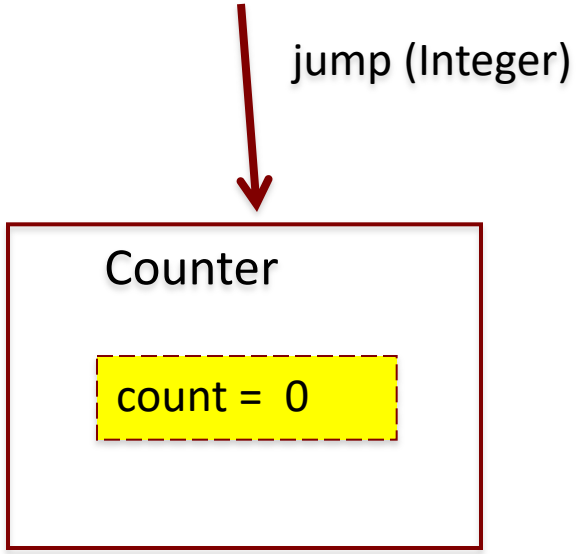
- **State-related features:**
  - **Initialization.**
  - **Mutation – setter method.**
    - **Performs an overwrite operation, not a merge.**
    - ***** Automatically causes component to re-render. *****

Component Data - Example

- **The Counter component.**
- **Ref.** samples/06_state.js

- **The useState() method:**
  - **A React hook.**
  - **Setter name arbitrary.**
  - **State variable (count) immutable.**

- **JS features:**
  - **Static function property, e.g.** defaultProps**, proptypes**



React's event system.

- **Cross-browser support.**
- **Event handlers receive** SyntheticEvent **– a cross-browser wrapper for the browser's native event.**
- **React event naming convention slightly different from native:**

React	Native
onClick	onclick
onChange	onchange
onSubmit	onsubmit

- See https://reactjs.org/docs/events.html for full details,

Automatic Re-rendering

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

User clicks button

→ *onClick event handler (incrementCounter) executed*

→ *state is changed (setCount())*

→ *component function re- executed (**re-rendering**)*

Modifying the DOM

- **DOM – an internal data structure; mirrors the state of the UI; always in sync.**
- **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. **Create a lightweight, efficient form of the DOM – the Virtual DOM.**
  2. **React app changes V DOM via components' JSX**
  3. **React engine:**
     1. **Perform *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) **Clean, descriptive programming model.**
  b) **Optimized DOM updates and reflows.**

Automatic Re-rendering (detail)

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

User clicks button

→ *onClick event handler executed*

→ *state is changed*

→ *component function re-executed*

→ *The Virtual DOM has changed*

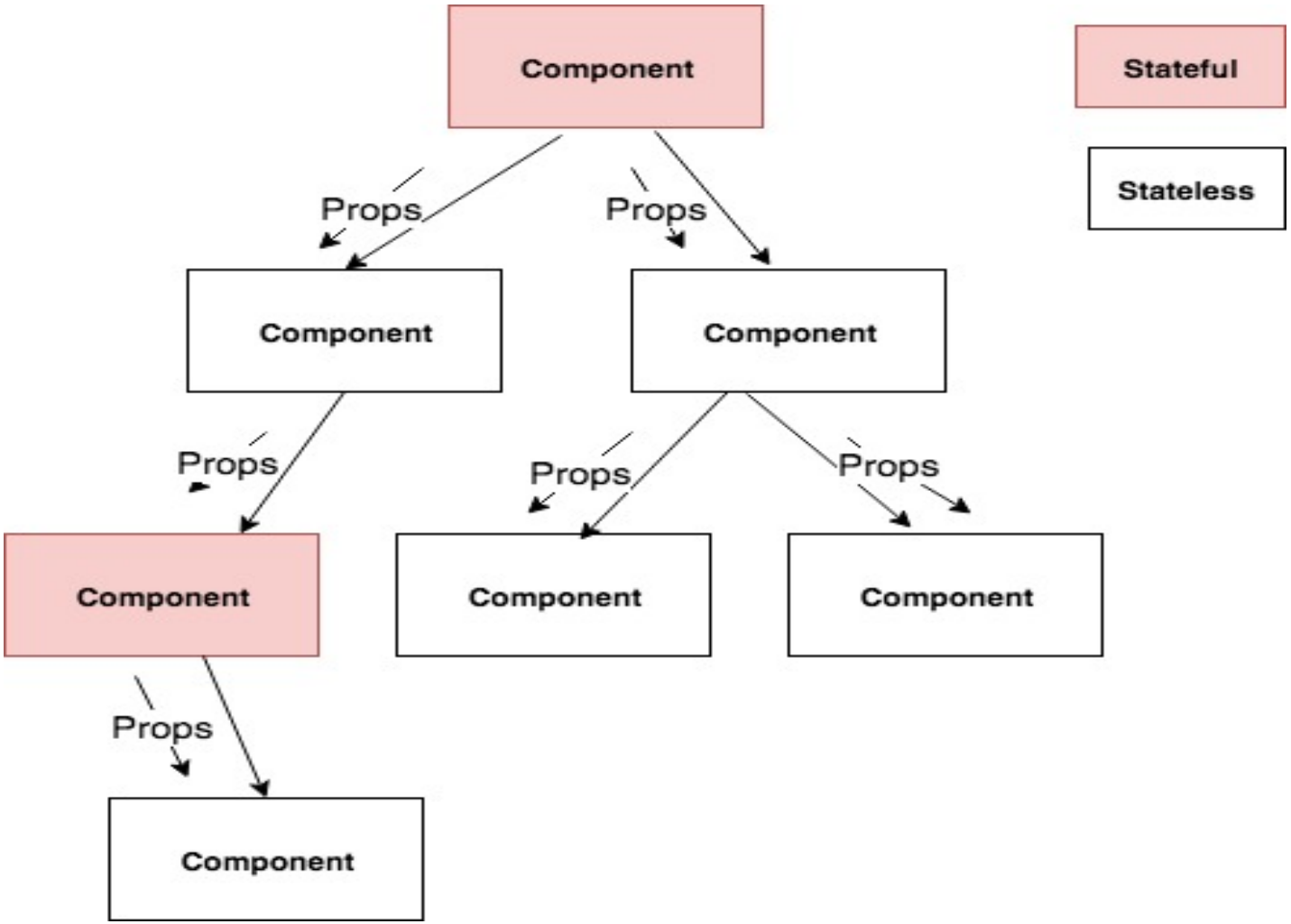
→ *React diffs the changes between the current and previous Virtual DOM*

→ *React batch updates the Real DOM*

Topics

- **Component State.** ✔
- **The Virtual DOM.** ✔
- **Data Flow patterns.**
- **Lifecycle methods.**

Unidirectional data flow



Unidirectional data flow

- **In a React app data flows uni-directionally ONLY.**
  - **Most other SPA frameworks use two-way data binding.**

- **In a multi-component app a common pattern is:**
  - **A small subset (maybe only 1) of components will be stateful; the rest are stateless.**

- **Stateful components:**
  - **Calls *state setter method* to update state variable(s).**
  - **Re-renders itself automatically.**
  - **Passes updated props to subordinate components.**
  - *React guarantees subordinate components are also re-rendered with new (perhaps unchanged!!) props.*

Topics

- **Component State.** ✔
- **The Virtual DOM.** ✔
- **Data Flow patterns.** ✔
- **Hooks**

React Hooks

- **Introduced in version 16.8.0 (February 2019)**

- **React Hooks are:**
  1. **functions (some HOF),**
  2. **that allow us easily manipulate the state and manage the lifecycle of a functional component,**
  3. **without needing to convert them into class components.**

- **Examples: useState, useEffect, useContext, useRef, etc**
  - **Names muat start with 'use' for linting purposes.**

- **Usage rules:**
  1. **Only Call Hooks at the Top Level.**
     - **Don't call Hooks inside loops, conditions, or nested function.**
  2. **Only Call Hooks from React Functions.**
     - **Don't call Hooks from regular JavaScript functions.**

useEffect Hook

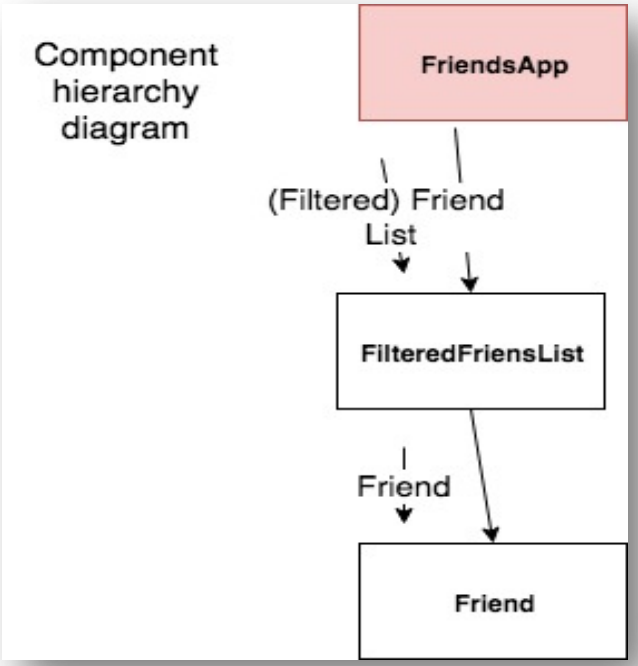
- useEffect **lets us perform** side effects **from a component.**

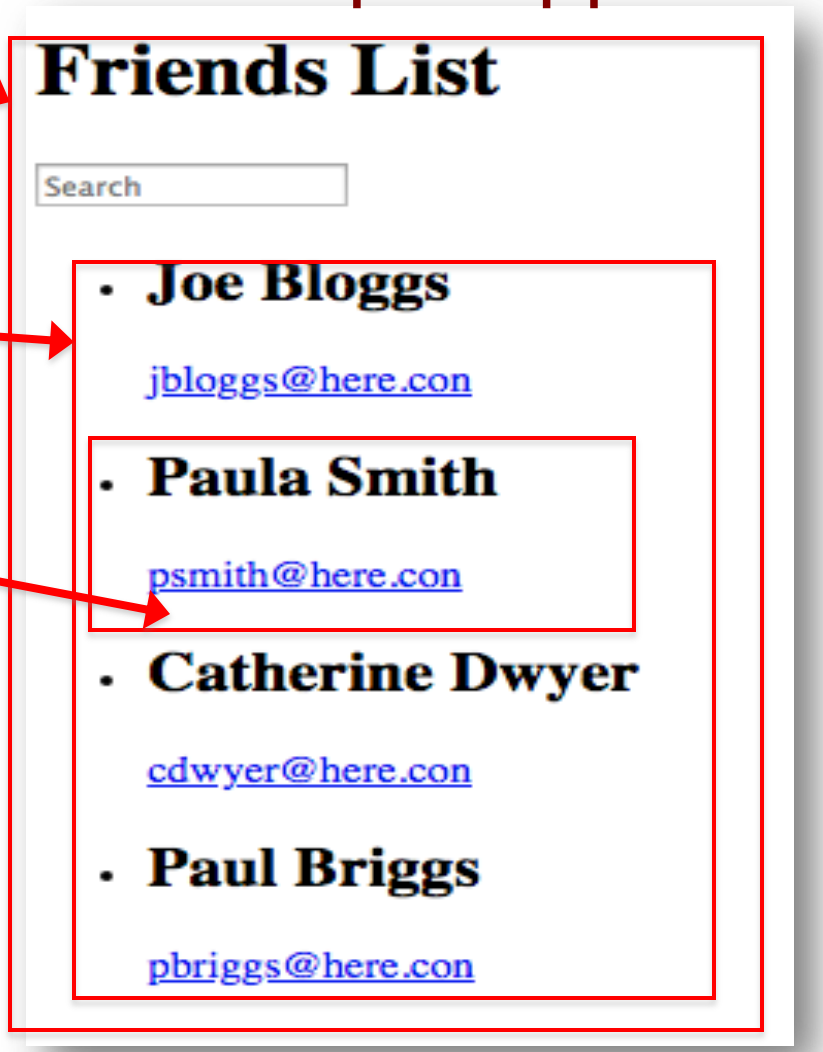
- **Side Effect example:**
  - **fetching some data from an API.**
  - **listening for upcoming browser events.**

- **Signature:** useEffect(callback, dependency array)

- **Execution times:**
  - **On mounting.**
  - **On every rendering,**
    - **Unless a dependency array is specified,**
      - **then only when a dependency variable changes value.**
  - **Use an empty dependency array to restrict execution at mount-time only.**

Sample App



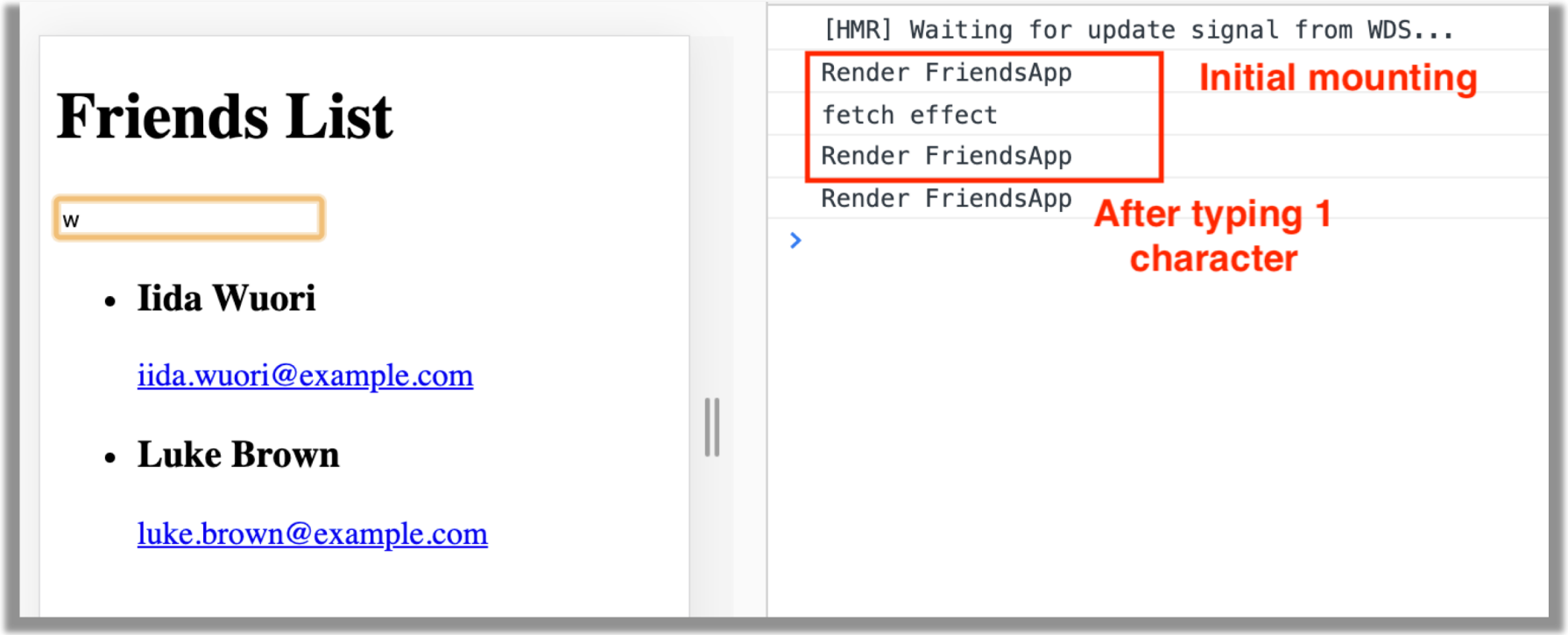


FriendsApp **component:**

1. **Manages app's** state **(i.e. text box, full list of friends).**
2. useEffect **hook gets API data**
3. **Computes matching friends.**
4. **Controls list re-rendering.**

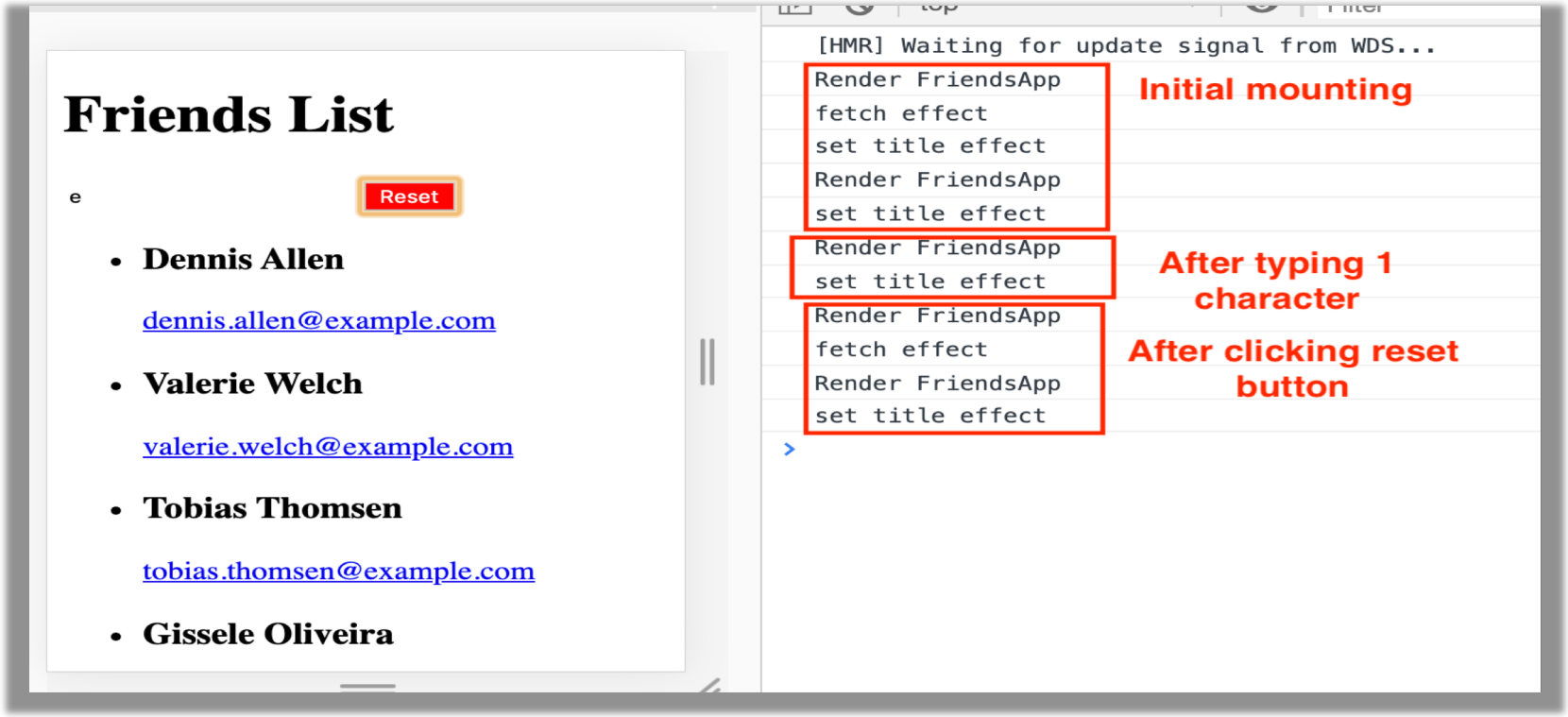
useEffect Hook

- **useEffect runs AFTER component has mounted.**
  - **First rendering occurs BEFORE data is available.**
    - **Must allow for this in implementation.**

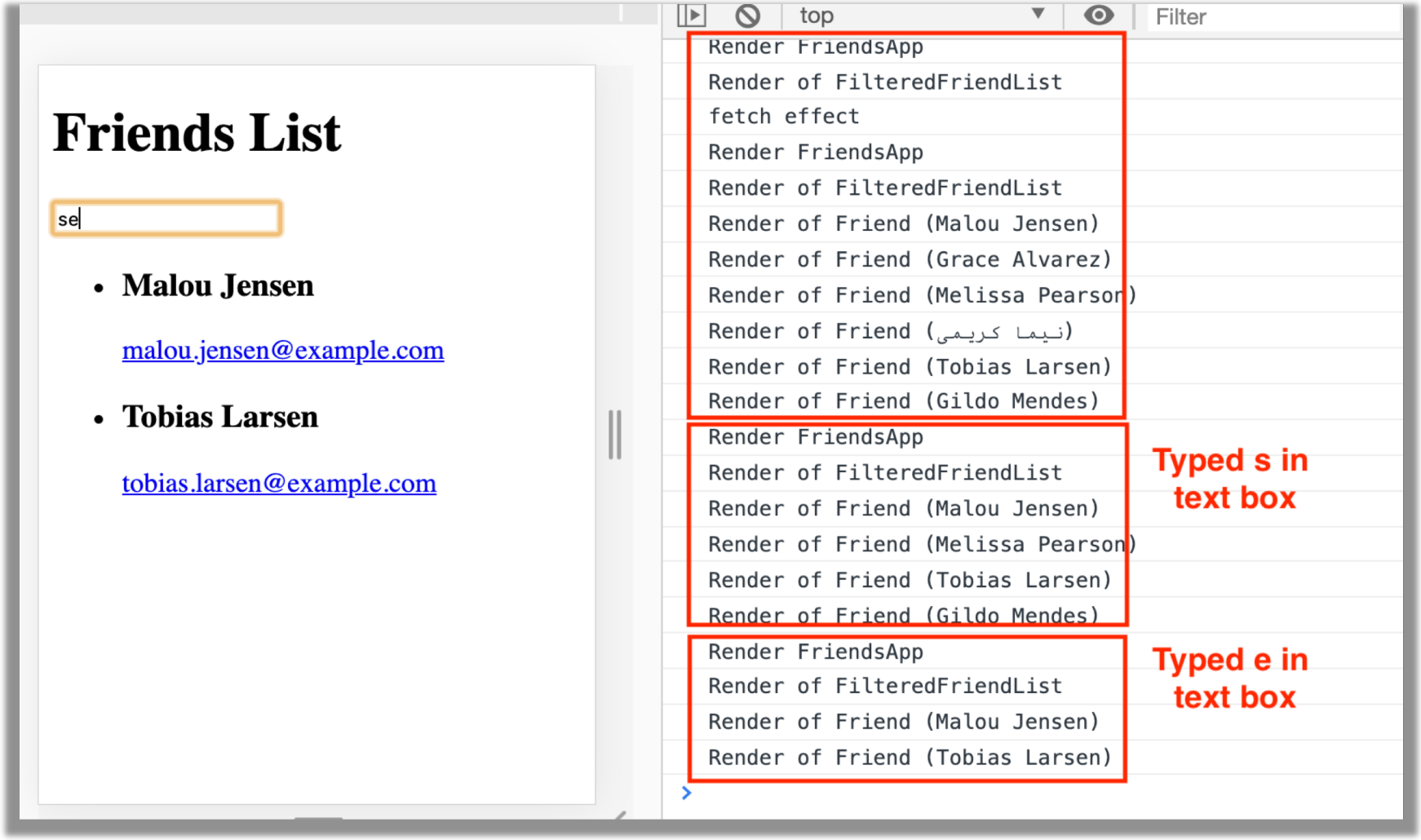


Sample App

- **App changes** (ref App2.js)**:**
  1. **Reset button to triggers load more data/friends from API.**
  2. **Browser tab title shows # of matching friends (side effect).**
- **‘Fetch’ effect dependent on** reset button **state change.**
- **‘Set title’ effect dependent on matching list size change.**



Unidirectional data flow & **Re-rendering**



Unidirectional data flow & **Re-rendering**

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

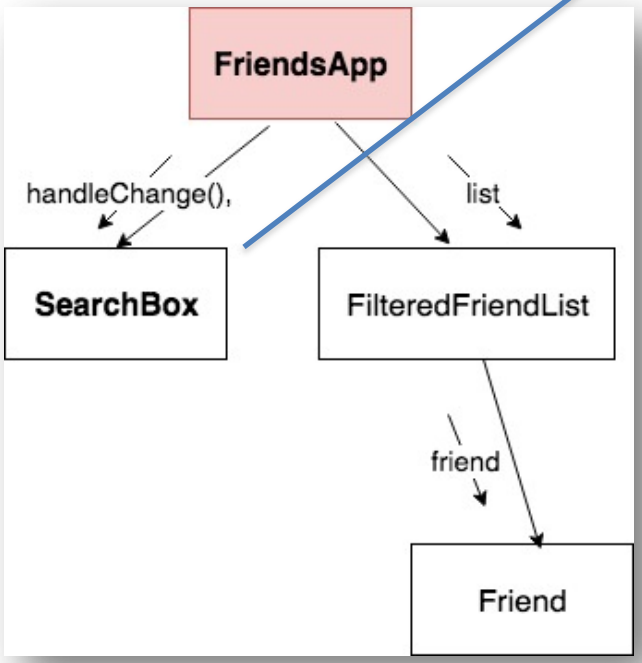
  User types a character in text box
  → *onChange* ***event handler executes***
  → ***Handler changes state (****FriendsApp* ***component)***
  → ***React re-renders*** *FriendsApp*
  → ***React re-renders children (****FilteredFriendList****) with new prop values.***
  → ***React re-renders children of*** *FilteredFriendList****.***
  (Re-rendering completed)
  → ***(Pre-commit phase) React computes the new Virtual DOM***
  → ***React diffs the new and previous Virtual DOMs***
  → ***(Commit phase) React batch updates the Real DOM.***
  → ***Browser repaints screen***

Topics

- **Component State.** ✔
- **The Virtual DOM.** ✔
- **Data Flow patterns.** ✔
- **Hooks. (More later)** ✔

Inverse data flow (Data down, actions up)

- **What if a component's** state **is influenced by an** event **in a subordinate component?**
- **Solution: The** inverse data flow pattern.



Inverse data flow

Pattern:

1. **Stateful component** (FriendsApp) **provides a callback to subordinate** (SearchBox)**.**
2. **Subordinate calls it when 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} />;
};
```

Summary

- **Component state.**
  - **User's input/interaction is 'recorder' in a component's state variable.**
  - **State changes cause component re-execution -> re-rendering**
  - **Re-rendering (may) results in UI changes -> dynamic apps.**

- **Hooks – allows us manipulate the state variables and hook into the lifecycle of a component**
  - **useState, useEffect, etc**

- **React achieves DOM update performance improvements by managing an intermediate data structure, the Virtual DOM.**

- **Data only flows downward through the component hierarchy – this aids debugging.**