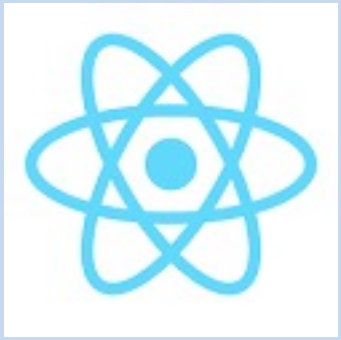


ReactJS.

The Component model

Topics

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

Component DATA

- **A component has two sources of data:**

  1. Props **- Passed in to a component; Immutable; the** props **object.**
  2. *State* **- Internal to the component; Causes the component to** re-render **when changed / mutated.**
  - **Both can be any data type - primitive, object, array.**

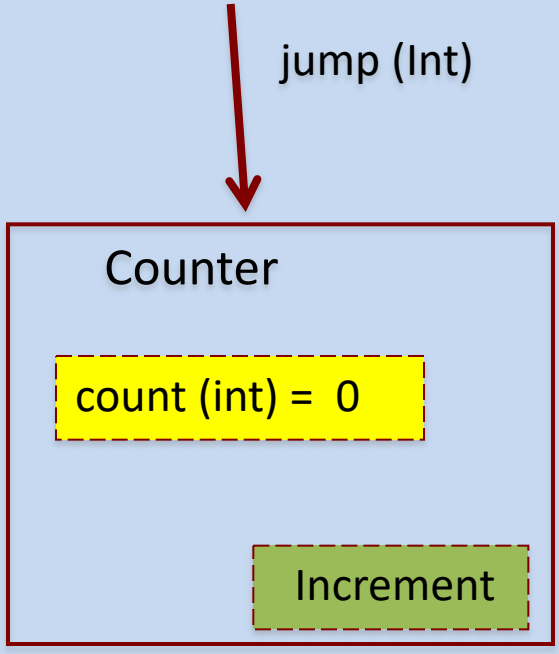
- **Props-related features:**
  - **Default values.**
  - **Type-checking.**

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

Stateful Component Example

- **The Counter component.**
  - **Reference basicReactLab, sample 6..**
- **The** useState() **function:**
  - **Termed a React** hook.
  - **Declares a state variable.**
  - **Returns a tuple – inclues Setter / Mutator method..**
- **Aside: Static function property,**

  e.g. defaultProps, proptypes



React's event system.

- **Enables cross-browser support.**
- **Your event handlers receive a** SyntheticEvent **– a wrapper for the browser's native DOM event.**
- **React event naming convention slightly different to 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 executes (incrementCounter)

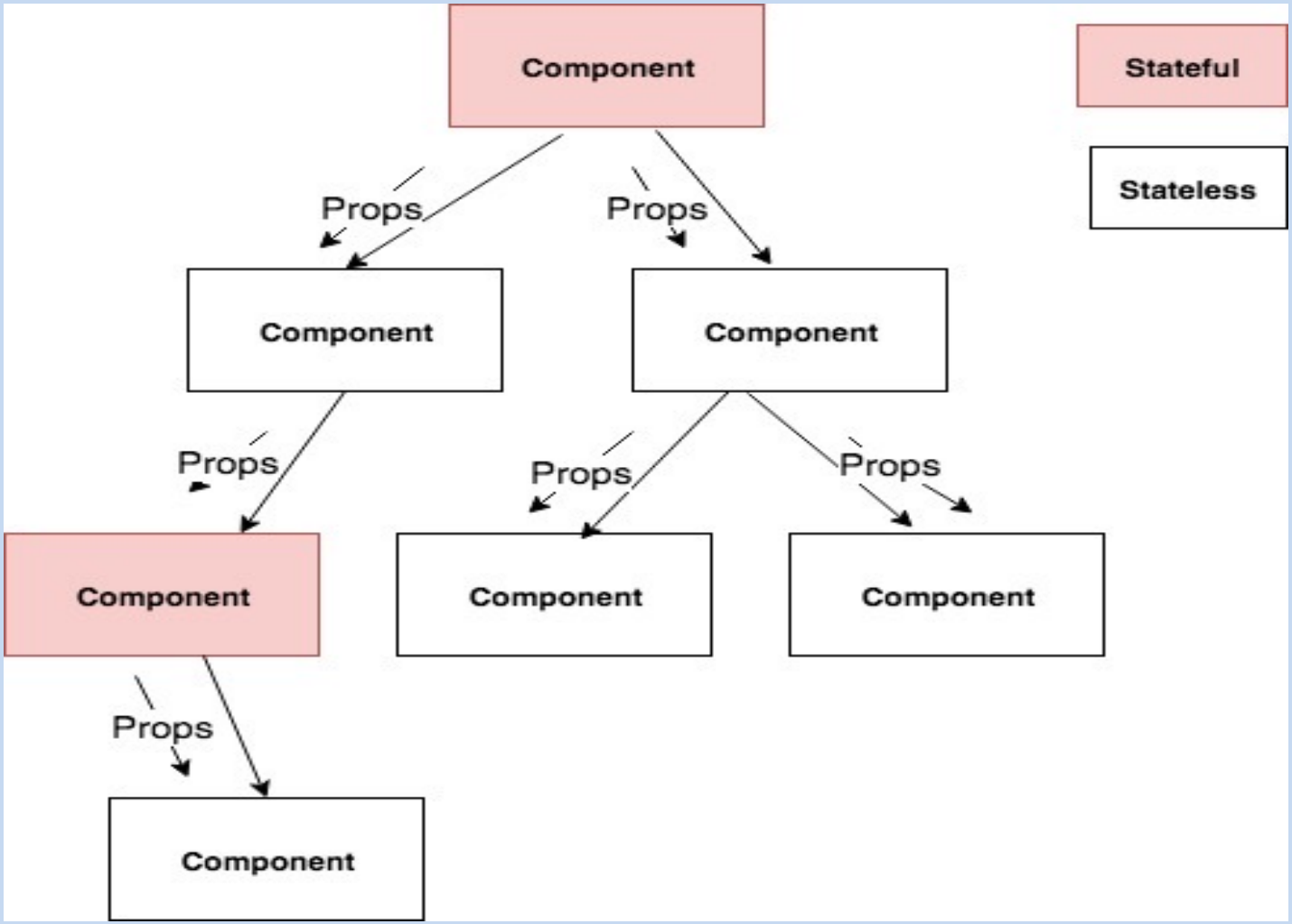
→ component state variable is changed (setCount())

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

Topics

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

Unidirectional data flow.



Unidirectional data flow

- **In a React app, data flows** unidirectionally **ONLY.**
  - **Other frameworks used** two-way data binding**.**

- **Typical app component breakdown:** Small subset **of** stateful **components; the rest are** stateless**.**

- **Typical Stateful component execution flow:**
  1. **User interaction causes component's state to change.**
  2. **Component re-renders (re-executes).**
  3. **It recomputes props for its subordinate components.**
  4. **Subordinate components re-render, and recomputes props for its subordinates.**
  5. **etc.**

Topics

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

React Hooks

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

- **A Hook is:**
  1. **A functions (some are HOFs).**
  2. **To** *manipulate* **a component's** *state* **and** *manage it's lifecycle*.
  3. **(Obviate the need to implement components as classes.)**

- **Examples:** *useState, useEffect, useContext, useRef, etc*
  - **'use' prefix is necessary for linting purposes.**

- **Hook usage rules:**
  1. **Can only call at the 'top level' in a component.**
     - **Don't call inside loops or condition statements.**
  2. **Only call from a component functions.**

useEffect Hook

- **Use when a component performs** side effects**.**

- **Side Effect example:**
  - **fetching data from a web API.**
  - **Subscribe to browser events, e.g. window resize.**

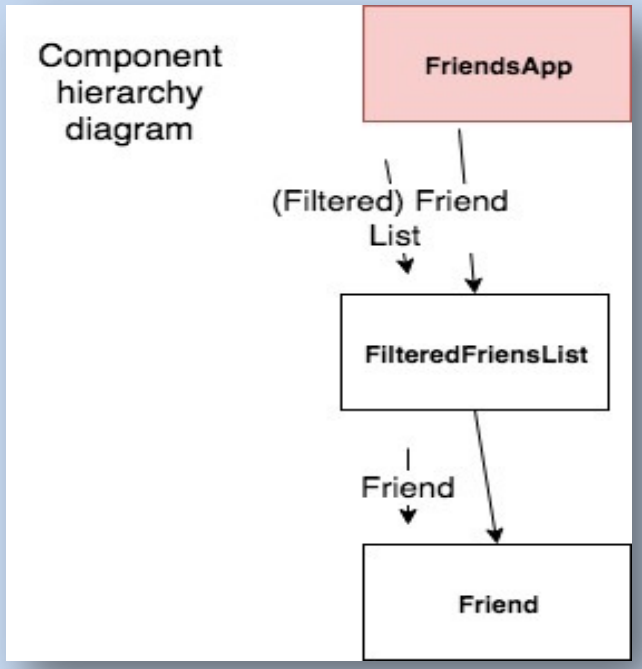
- **Signature:** useEffect(callback, dependency_array)
  - **The** *callback* **contains the side effect code.**

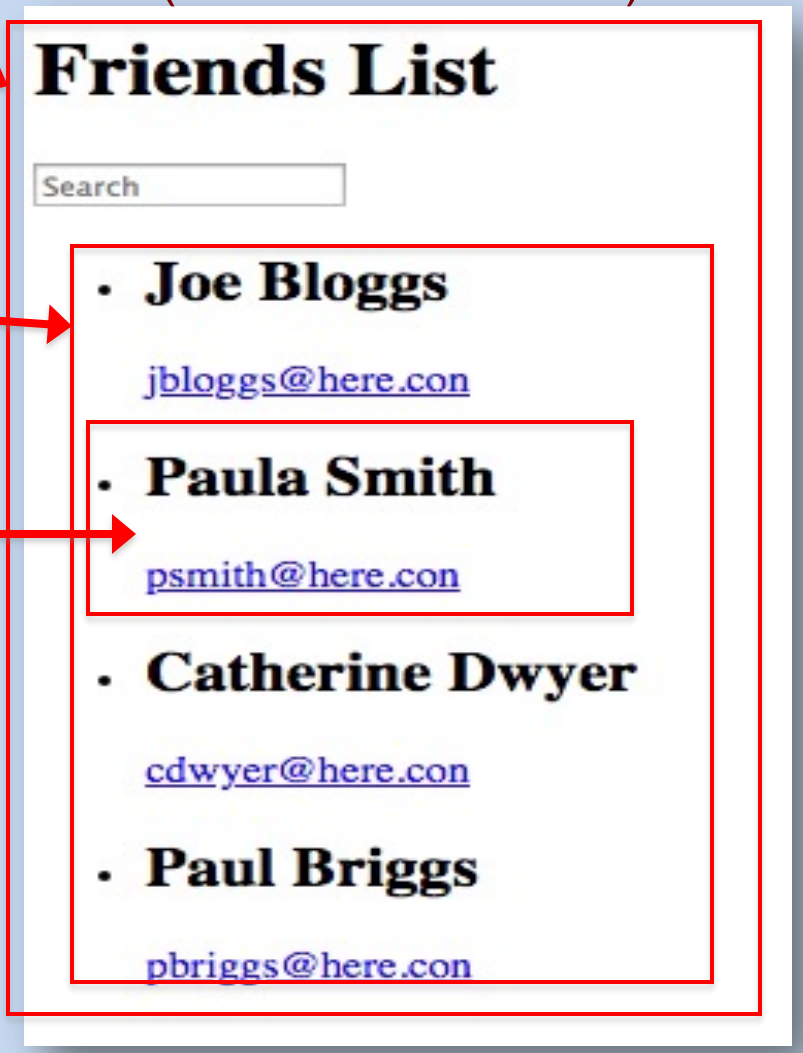
- **useEffect() is executed by a component:**
  1. **When it is mounted on the DOM.**
  2. **On every rendering, provided a dependency array entry has changed value since the previous rendering.**

  - **An** empty dependency array **restricts execution to mount-time only.**

Sample App 1

(see lecture archive)





FriendsApp **component:**

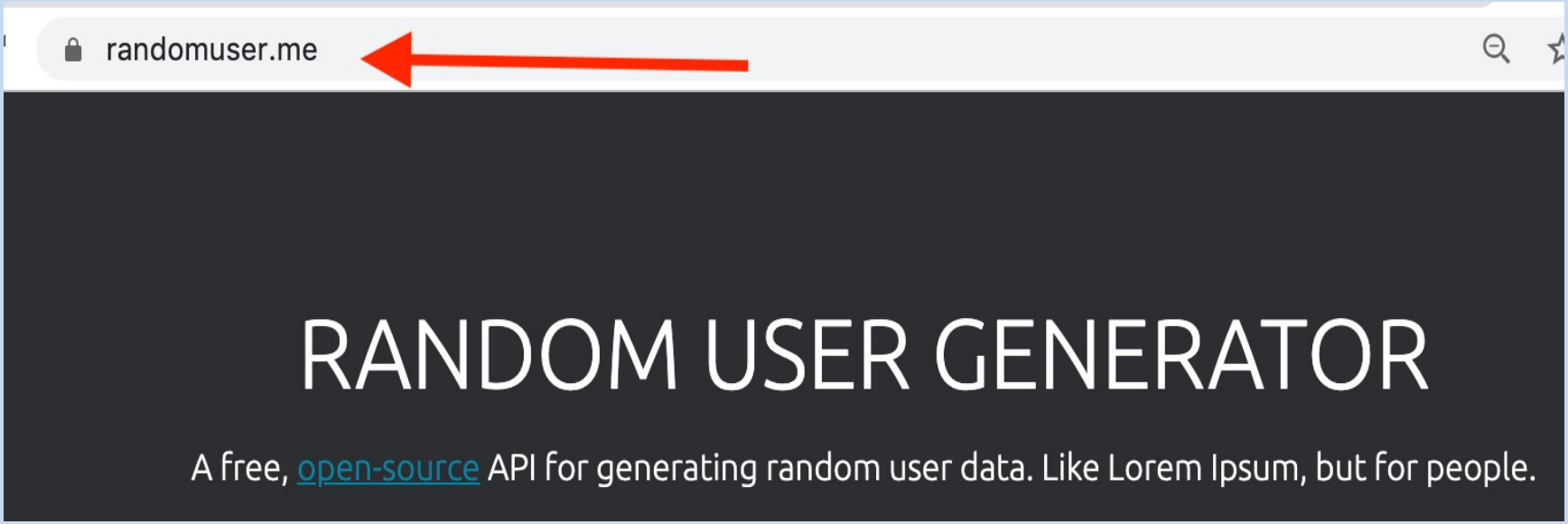
1. Manages state **(itext box + full list of friends).**

2. Fetches friends from the web API – side effect.

3. Computes matching friends.

4. Controls rendering of friend list.

RandomUser open API

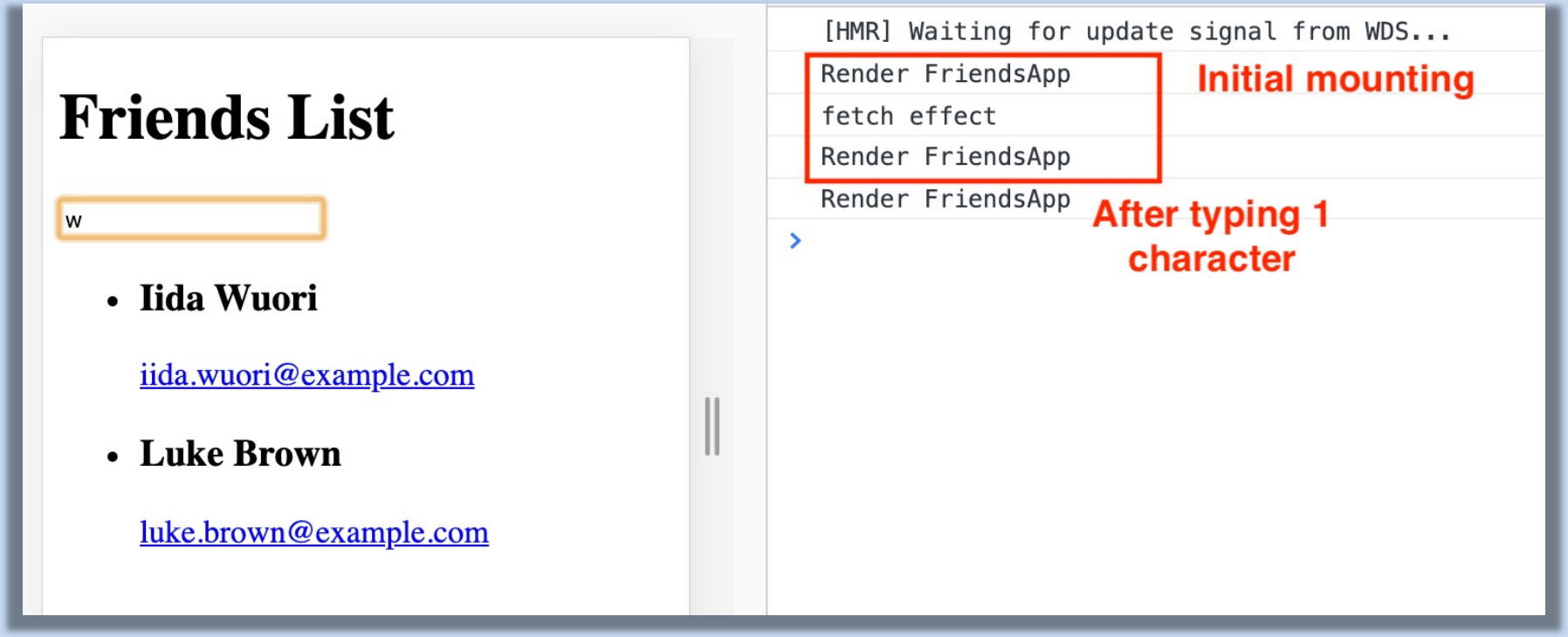


- Returns an auto-generates list of user profiles (friends).
- e.g. Get 10 user profiles:

  GET https://randomuser.me/api/?results=10

Sample App - *useEffect* Hook

- **(General) A useEffect runs <u>at the end</u> of a component's mount process.**
- **(Sample app) Callback makes an asynchronous call to web API → First rendering occurs <u>before</u> the API data is available.**



Sample App - *useEffect* Hook

- **We must accommodate asynchronous nature of API calls:**
  1. **Do not 'freezing' the browser while waiting.**
  2. **Allowing components to render without real data.**

- **Correct solution:**

```
const [friends, setFriends] = useState( [ ] ) ;
```

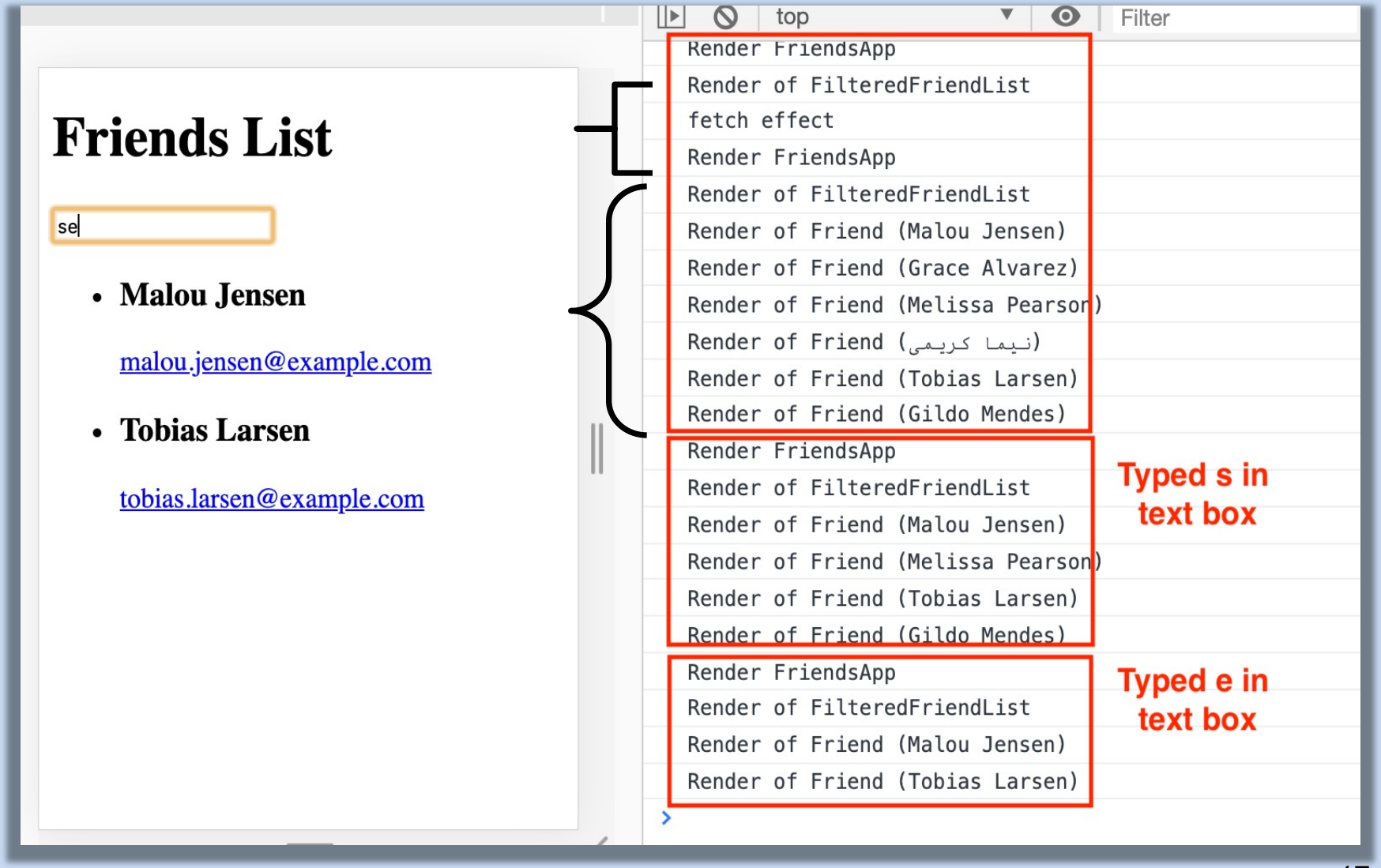
- **Incorrect solution:**

```
const [friends, setFriends] = useState(null);
```

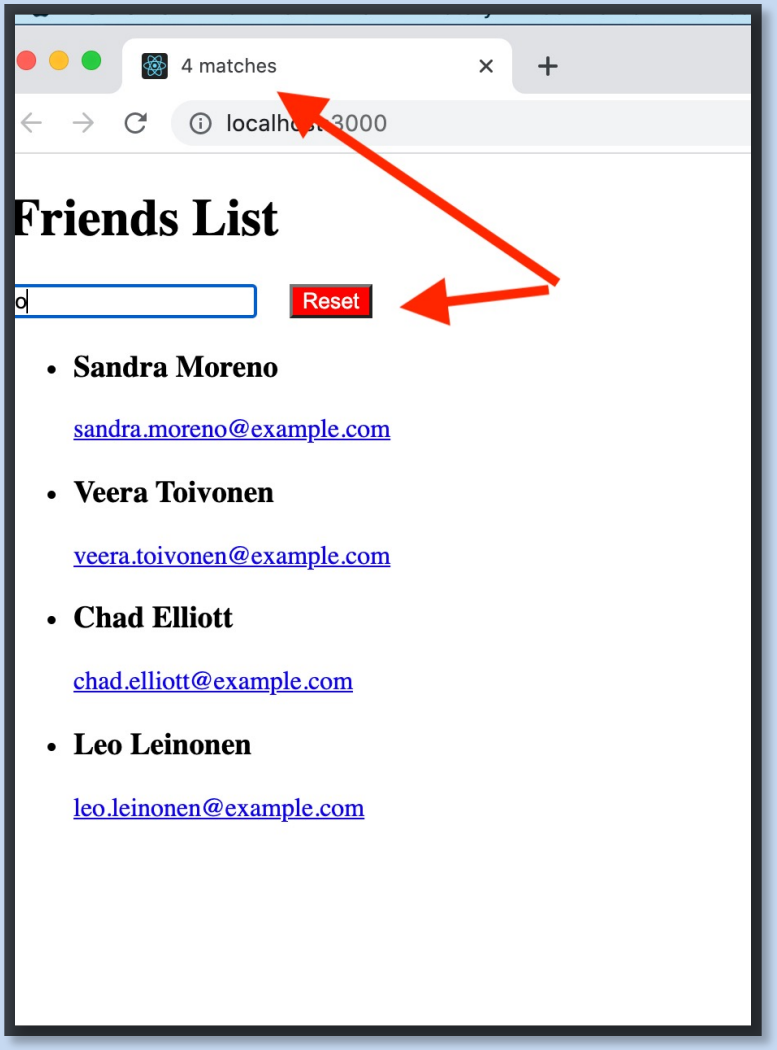
TypeError: Cannot read property 'filter' of null

Unidirectional data flow & **Re-rendering**

(Assume we request 6 friends from web API)

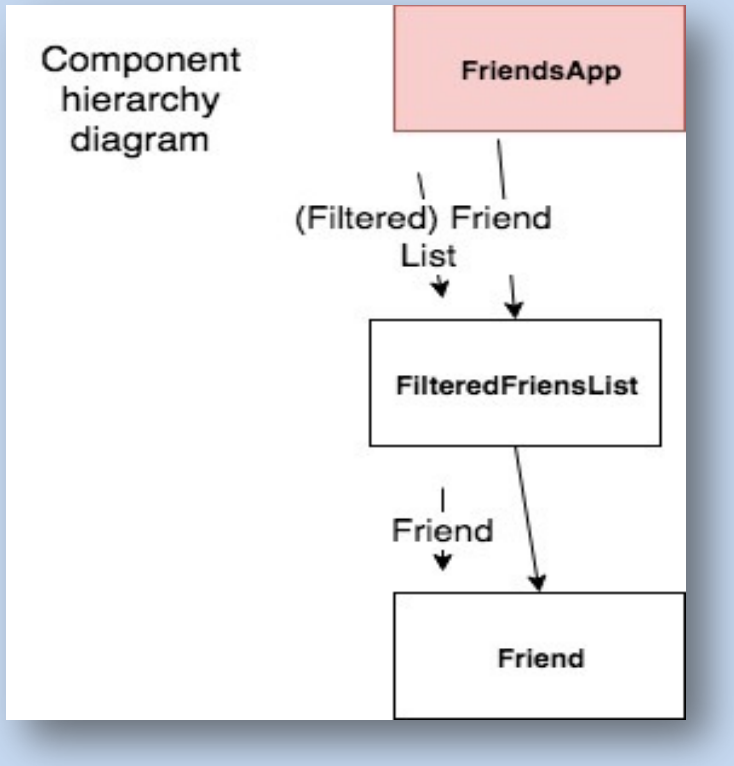


Sample App 1 – Version 2



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

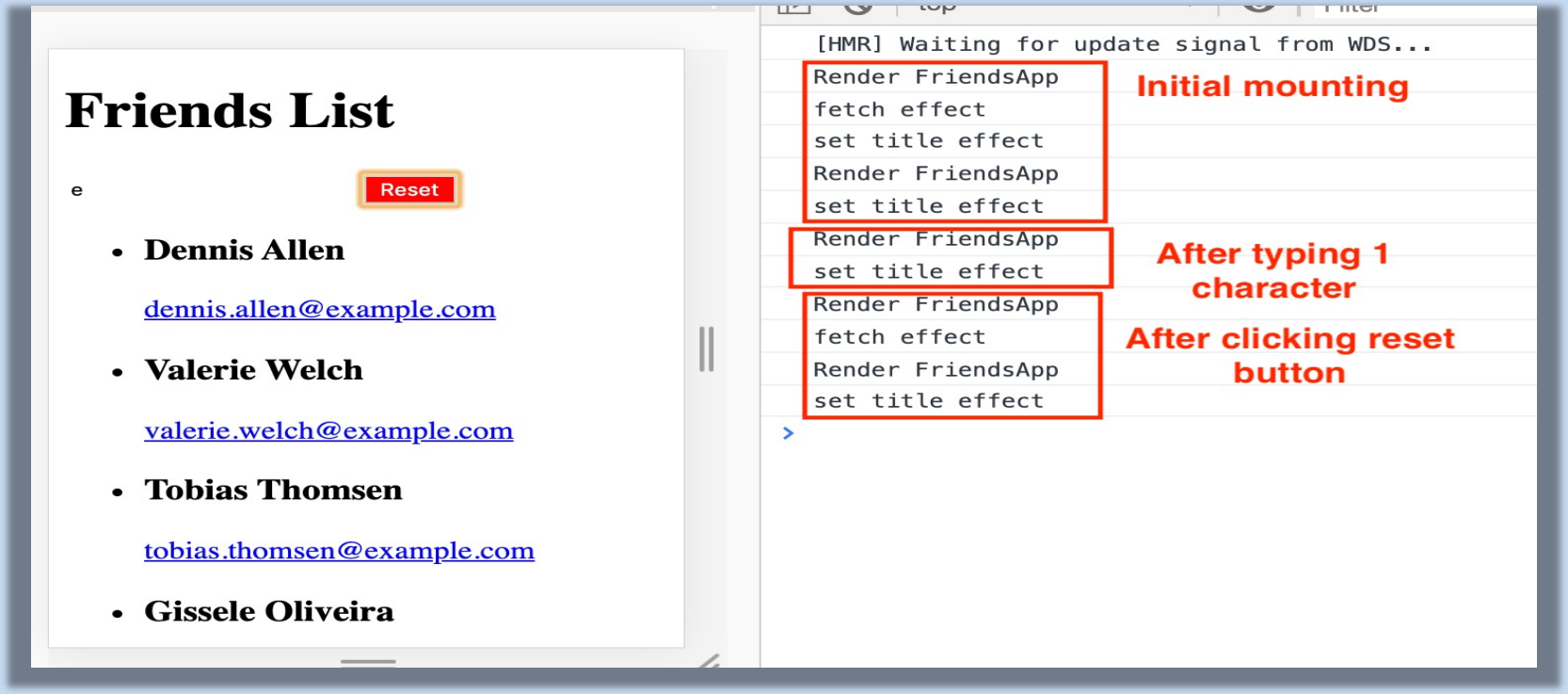
Sample App 1 (v2) - Design



- **Three state variables:**
  1. **List of friends from API.**
  2. **Text box content.**
  3. *Reset button toggle.*

- **Two side effects:**
  1. **'Fetch API data' - dependent on change to** reset button **toggle.**
  2. **'Set browser tab title' - dependent on change to # of matches from friend list.**

Sample App 1 (v2) - Events



Sample App 1 (v2) - Events.

- **When** FriensApp **component is mounted on the DOM:**
  - → **Both useEffects execute**
  - → **Update browser tab title.**
  - → **'Friends list' state changes.**
  - → **Component re-renders**
  - → **'Set browser tab' effect executes.**
- **When user types in the text box:**
  - **'Text box' state change.**
  - → FriendsApp **rerenders + Matching friends list recomputed**
  - → **'Set browser title' effect executes.**
- **When user clicks the Reset button:**
  - → **'Reset toggle' state changes.**
  - → FriendsApp **re-renders.**
  - → **'Fetch data' effect executes.**
  - → **'Friends list' state changes.**
  - → **FriendsApp re-renders + Matching list recomputed.**
  - → **'Set browser title' effect executes.**

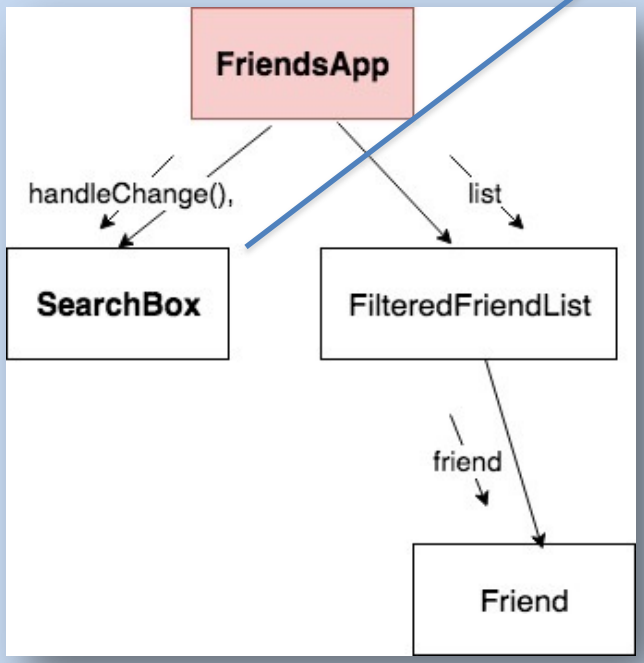
Topics

- **Component State.** ✔
- **Data Flow patterns.** ✔
- **Hooks and Component Lifecycle.** ✔
- **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.



Data down, Action up.

Pattern:

1. Stateful **component** (FriendsApp) **provides a callback to a 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

- **Component State.** ✔
- **Data Flow patterns.** ✔
- **Hooks and Component Lifecycle.** ✔
- **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 hadler 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.**

-