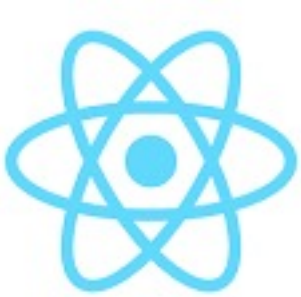


ReactJS.

The Component model

Topics

- **Component State.**
  - **Basis for dynamic, interactive UI.**
- **Data Flow patterns.**
- **Hooks.**
- **Material Design (Styling)**

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.**

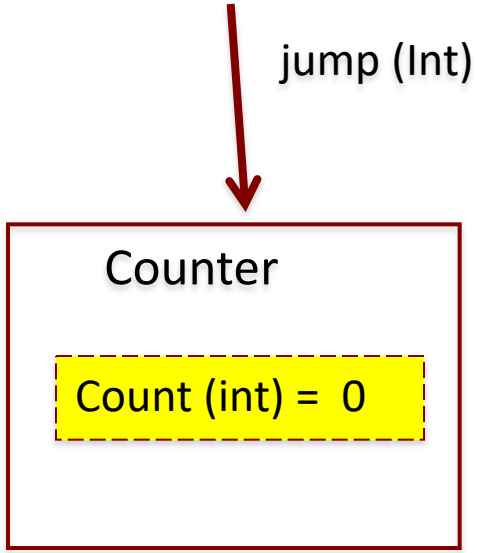
  – **Both can be any data type - primitive, object, array.**

- **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.**
- **Ref.** basicReactLab **samples – sample 06.**



- **The** useState() **function:**
  - **To declare a state variable.**
  - **Returns a Setter / Mutator method.**
  - **Termed a React** hook.

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

React's event system.

- **Cross-browser support.**
- **Event handlers receive a** 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 executes (incrementCounter)*

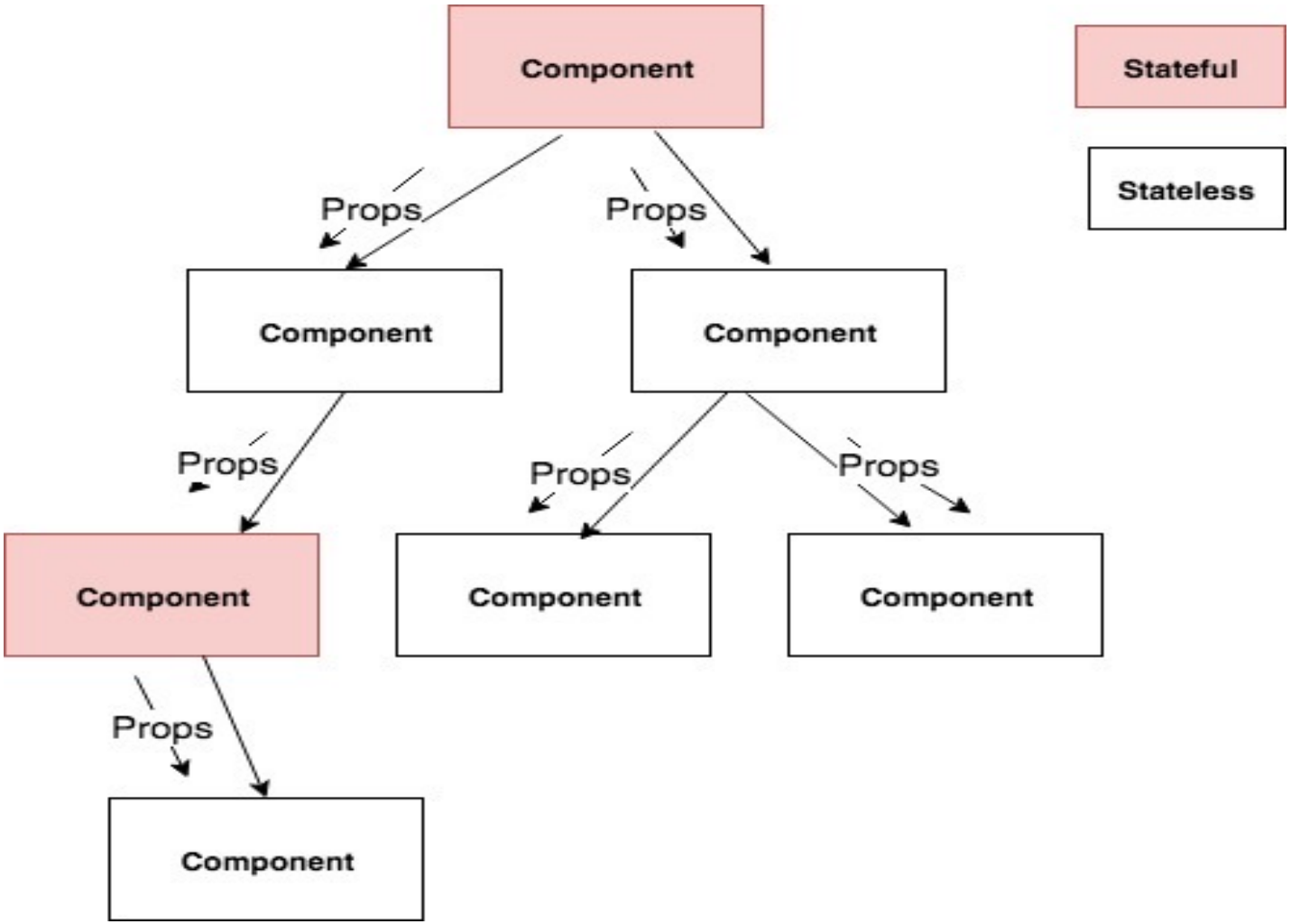
→ *component state variable is changed (setCount())*

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

Topics

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

Unidirectional data flow



Unidirectional data flow

- **In a React app, data flows** unidirectionally **ONLY.**
  - **Most other SPA frameworks use two-way data binding.**
- **Design pattern: A** small subset **of an app's components are** stateful **and the rest are** stateless**.**

- **Typical Stateful component execution flow:**
  1. **A user interaction causes component's state to change.**
  2. **Component re-renders automatically.**
  3. **Component recomputes props and passes them to its subordinate components.**
  4. **Subordinate components re-execute (re-render), and pass recomputed props to its subordinates.**
  5. **etc.**

Topics

- **Component State.** ✔
- **Data Flow patterns.** ✔ *(more later)*
- **Hooks**

React Hooks

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

- **React Hooks are:**
  1. **Functions (some HOF).**
  2. **That allow us to manipulate the state and manage the lifecycle of a component.**
  3. **(Obviate the need to implement components as classes.)**

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

- **Usage rules:**
  1. **Can only call hooks at the 'top level' in a component.**
     - **Don't call hooks inside loops, condition statements, or nested function.**
  2. **Only Call hooks from React component functions.**
     - **Don't call hooks from regular JavaScript functions.**

useEffect Hook

- **When a component needs to perform** side effects**.**

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

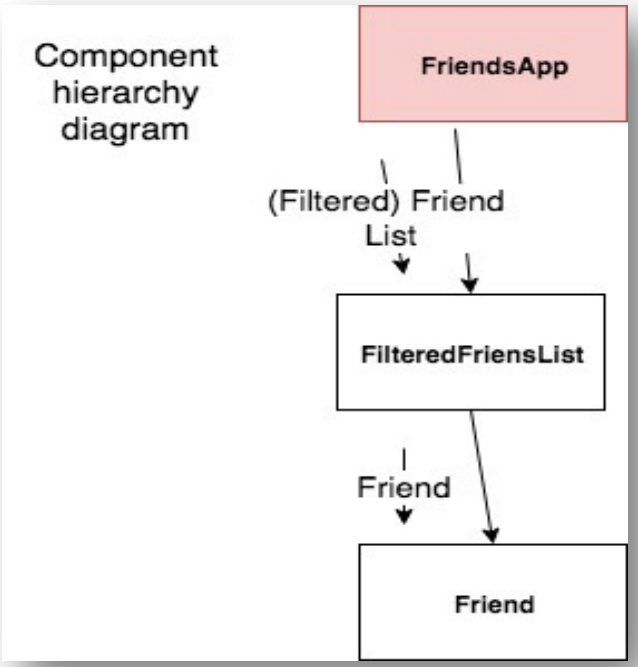
- **Signature:** useEffect(callback, dependency array)
  - **Side effect code is in the callback.**

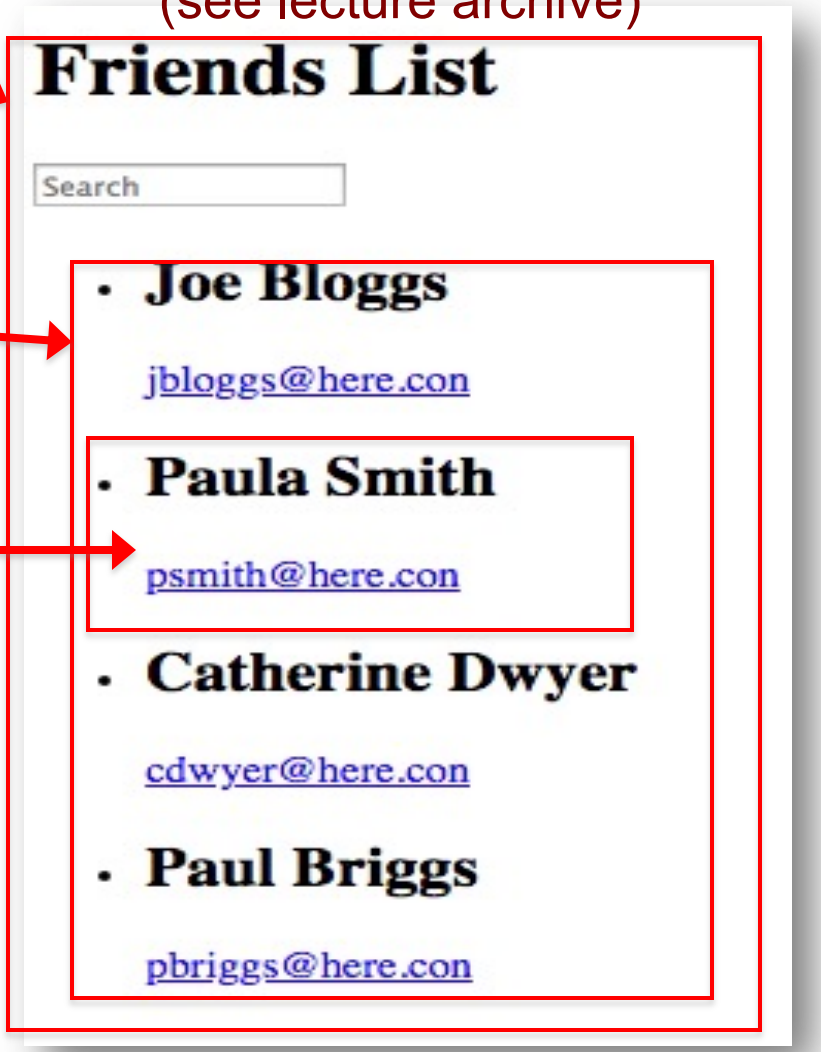
- **Execution times:**
  1. **On mounting.**
  2. **On every rendering <u>where</u> a member of its dependency array has changed value since the previous rendering.**

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

Sample App

(see lecture archive)





FriendsApp **component:**

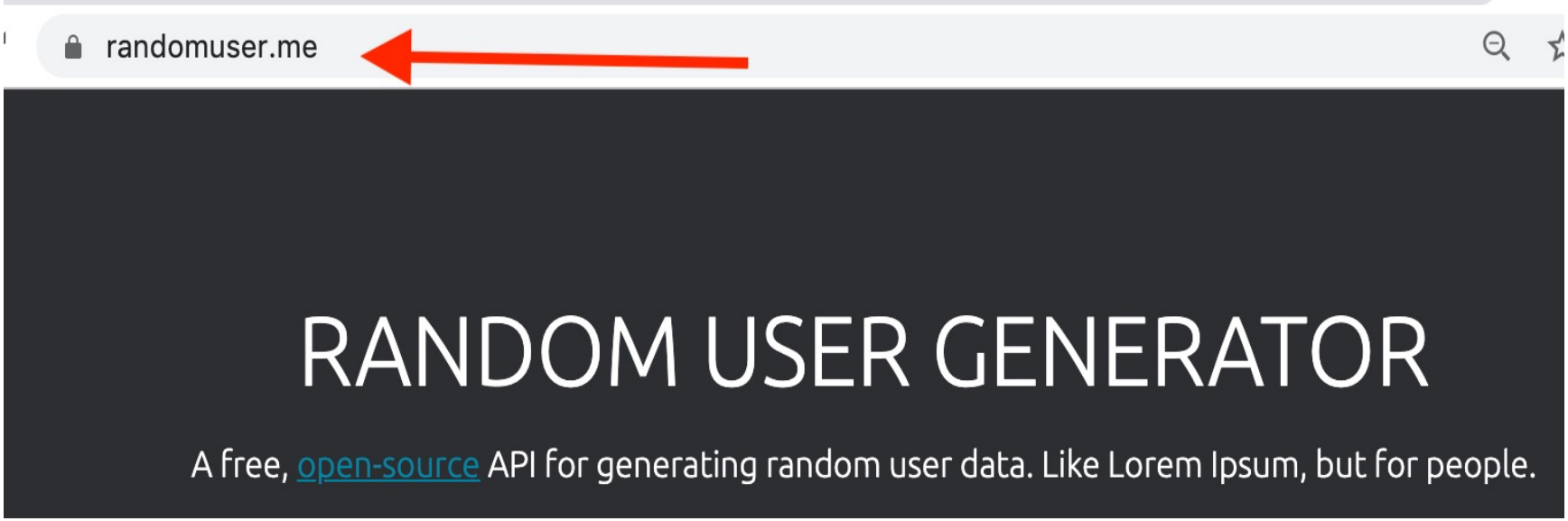
1. Manages app's state **(i.e. text box, full list of friends).**

2. Fetches all friends from aweb API - useEffect**.**

3. Computes matching friends.

4. Controls list rendering.

RandomUser open API

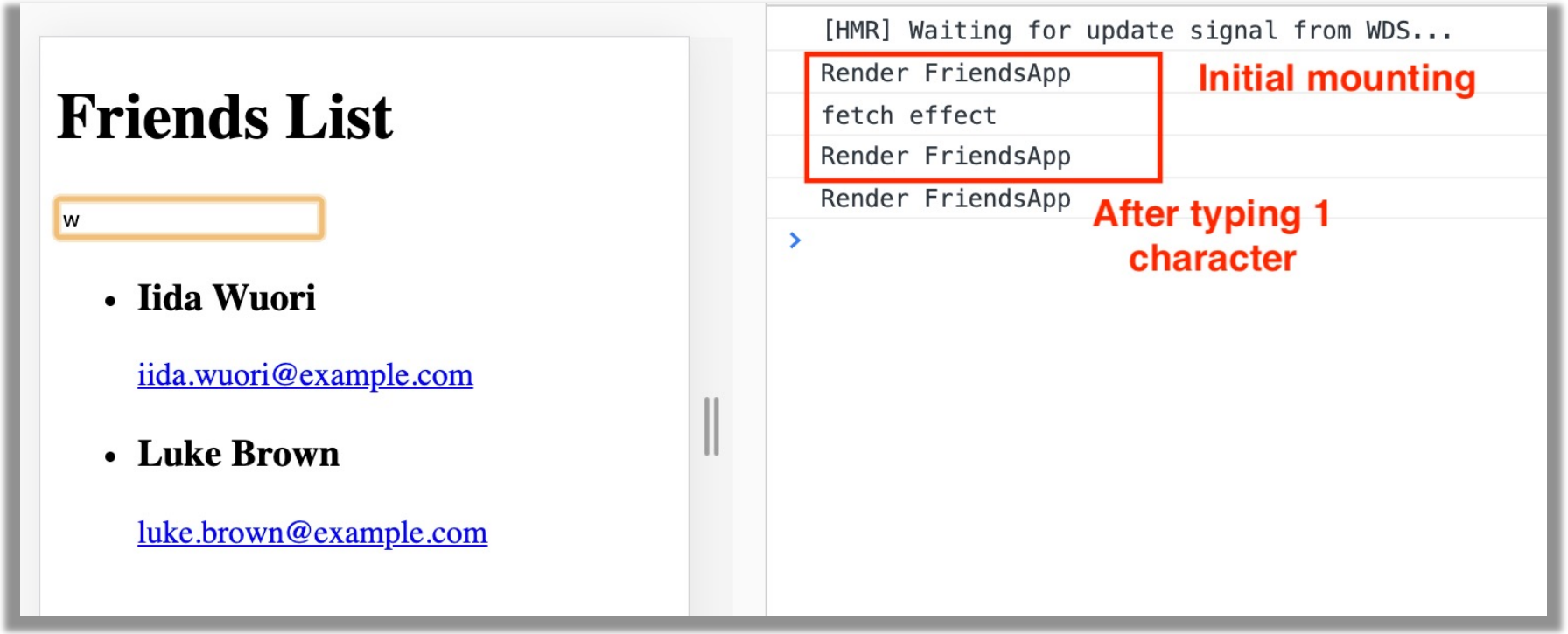


- Returns an auto-generates list of user profiles (friends).
- e.g. Get 10 user profiles:

  GET

Sample App - *useEffect* Hook

- **useEffect runs AT THE END of a component's mount process. i.e. First rendering occurs BEFORE the API data is available.**
- **We must accommodate this in the implementation.**



Sample App - *useEffect* Hook

- **You must allow for asynchronous nature of API calls**
  - **Do not ‘freeze’ the browser while waiting.**
  - **Enable components to render in the absence of data**

- **Correct solution:**

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

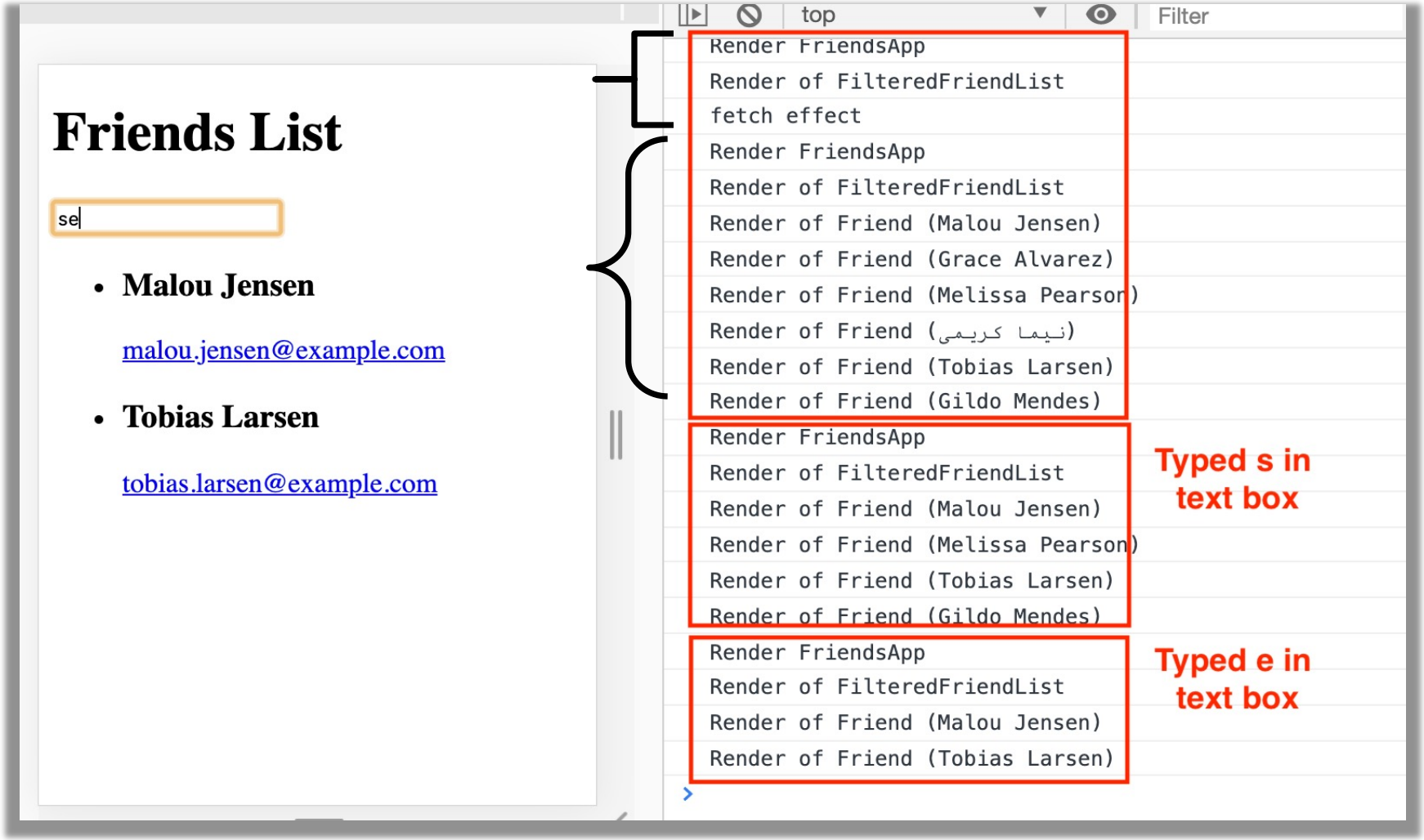
- **Incorrect solution:**

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

```
TypeError: Cannot read property 'filter' of nul
```

Unidirectional data flow & **Re-rendering**

(Assume we request 6 friends from web API)



Unidirectional data flow & **Re-rendering**

- **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***

Topics

- **Component State.** ✔
- **Data Flow patterns.** ✔ *(more later)*
- **Hooks.** ✔ *(more later)*
- **Material Design**

Material UI.

A 3rd party component library to build high quality digital UIs

Material Design.

- **Material (Design) is a design system created by Google to help teams build high-quality digital experiences for Android, iOS, and web.**
- **A visual language that synthesizes classic principles of good design with the innovation and possibility of technology and science.**
- **Inspired by:**
  - **the physical world and its textures, including how they reflect light and cast shadows.**
  - **the study of paper and ink.**

- **Material is a metaphor.**
  - **Material surfaces reimagine the mediums of paper and ink.**

Material Components.

- **Material Components are interactive** building blocks **for creating a digital user interface.**
- **They cover a range of interface needs, including:**
  1. **Display: Placing and organizing content using components like cards, lists, and grids.**
  2. **Navigation: Allowing users to move through an application using components like navigation drawers and tabs.**
  3. **Actions: Allowing users to perform tasks using components such as the floating action button.**
  4. **Input: Enter information or make selections using components like text fields and selection controls.**
  5. **Communication: Alerting users to key information and messages using snackbars, banners and dialogues.**

Theming.

- **Material Design does not mean copy Google design.**

- **Material Theming makes it easy to customize Material Design to match the look and feel of your brand, with built-in support and guidance for customizing colors, typography styles, and corner shape.**
- **Color - Material's color system is an organized approach to applying color to a UI. Global color styles have semantic names and defined usage in components – primary, secondary.**
- **Typography - The Material type system provides 13 typography styles for everything from headlines to body text and captions.**
  - **Each style has a clear meaning and intended application within an interface.**

Material UI.

- **A React component library based on the Material Design system.**
- **Components include <Card />, <Box />, <Grid />, <Menu />, <Button />, <Icon />, <Snackbar />, <Typography /> ……**
- **Build your own design system, or start with Material Design.**
- **The CSS-in-JS model.**

CSS-in-JS

- **Plain CSS - Separate files for CSS and JS.**

---------- CSS. ----------

```
.my-header {
  background-color: lightblue;
  padding: 10px;
}
```

---------- JS ----------

```
import ‘app.css’
   . . . . . .
<header
    className=”my-header">
       . . . . . .
</header>
```

Must be CamelCase

- **CSS-in-JS.**

```
.import { makeStyles } from
      "@material-ui/core/styles";

const useStyles =   makeStyles(({
      myHeader: {
        backgroundColor: "lightblue",
        padding: "10px”
      } );
  . . . . . .
const classes = useStyles();
  . . . . . .
<header
 className={classes.myHeader}>
  ……. </header>
```

1

2

3

Summary

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

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

- **Data only flows downward through the component hierarchy – this aids debugging.**
  - **Actions (Events) flow upwards**