ReactJS.

The Component model

Topics

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

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.**

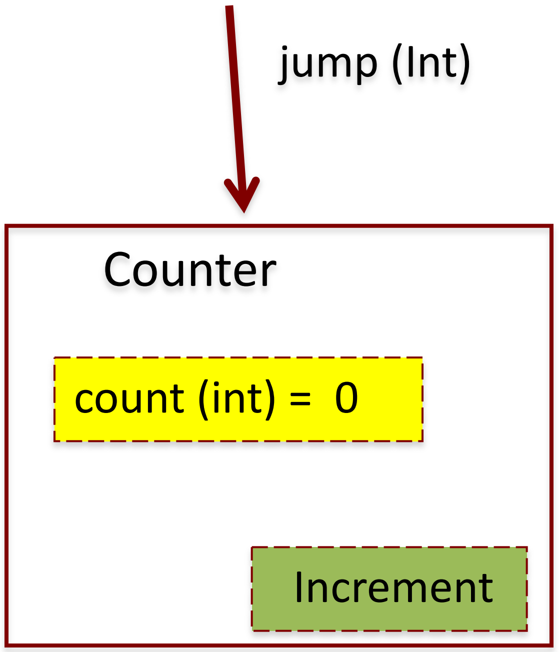
- **State-related features:**
  - **Initialization.**
  - **Mutation – using a** setter **method.**
    - Automatically causes component to re-render. ***
    - **Performs an overwrite operation, not a merge.**

Stateful Component Example

- **The Counter component.**
- **Ref.** basicReactLab **samples – sample 06.**
- **The** useState() **function:**
  - **Declares a state variable.**
  - **Returns a Setter / Mutator method.**
  - **Termed a React** hook.

- **Aside: 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 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)*

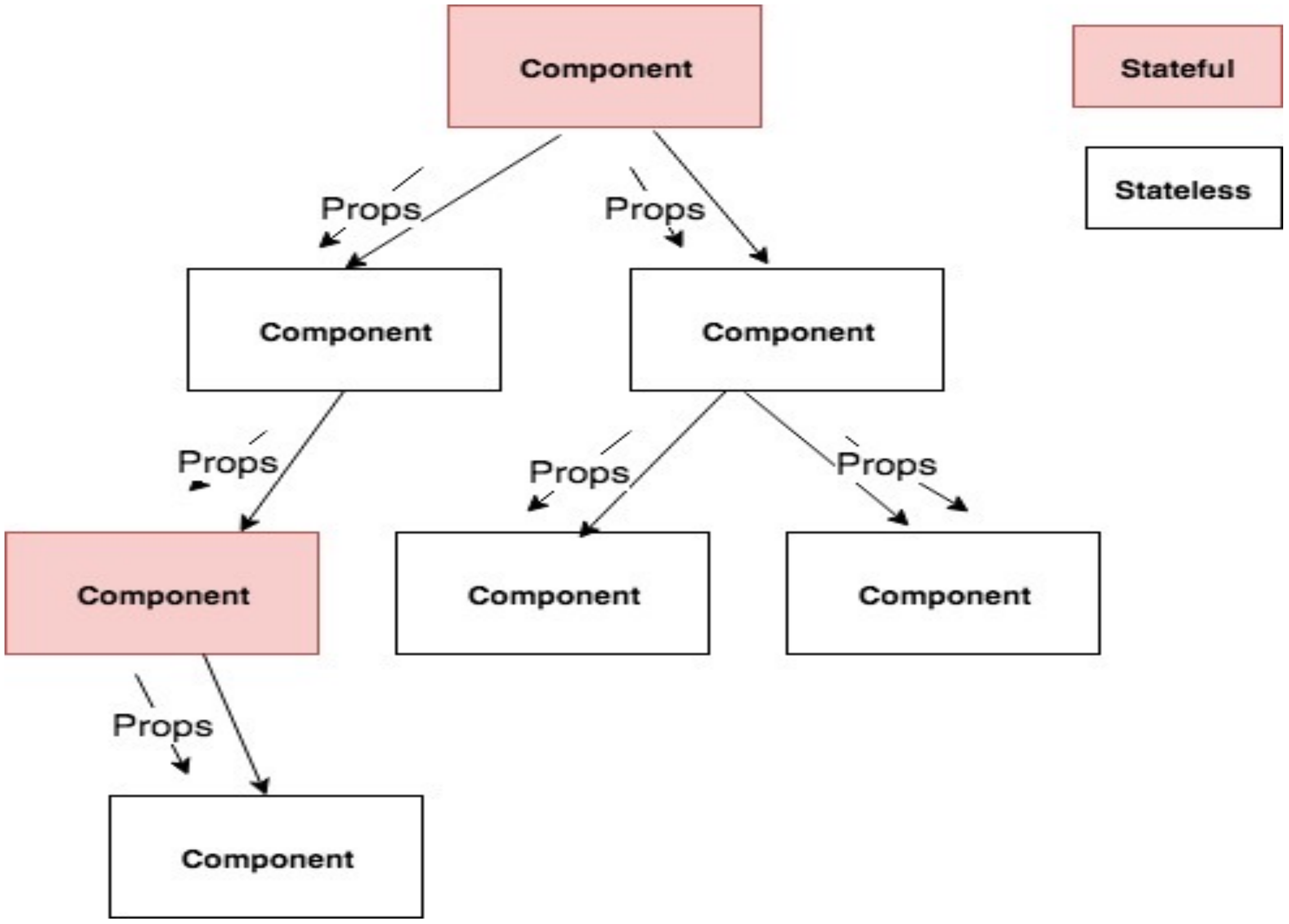
  → *component state variable is changed (setCount())*

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

Topics

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

Unidirectional data flow



Unidirectional data flow

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

- **Typical React app pattern: A** small subset **of the components are** stateful**; the majority are** stateless**.**
- **Typical Stateful component execution flow:**
  1. **User interaction causes a component state 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**

React Hooks

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

- **React Hooks are:**
  1. **Functions (some HOF).**
  2. **To** *manipulate* **the** *state* **and** *manage* **the component'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 rules:**
  1. **Can only call hooks at the 'top level' in a component.**
     - **Don't call hooks inside loops or condition statements.**
  2. **Only call hooks from React component functions.**

useEffect Hook

- **Use when a component needs to perform** side effects**.**

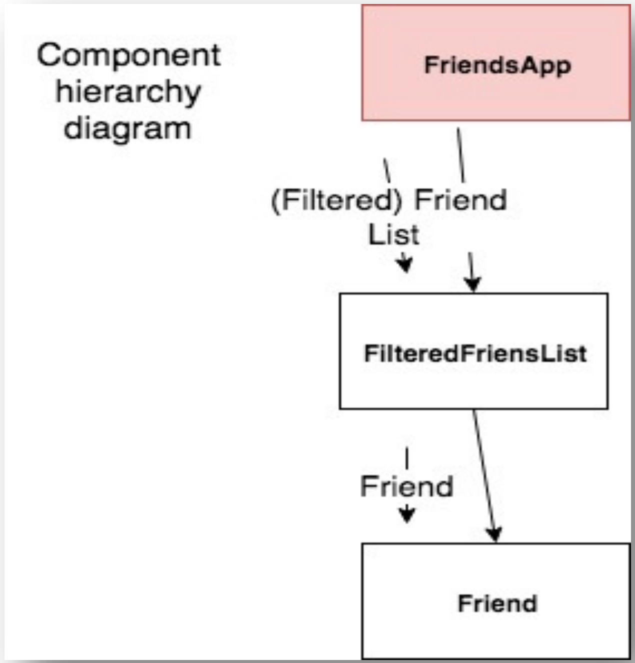
- **Side Effect example:**
  - **fetching data from a web API.**
  - **Subscribe to browser events, e.g. window resize.**

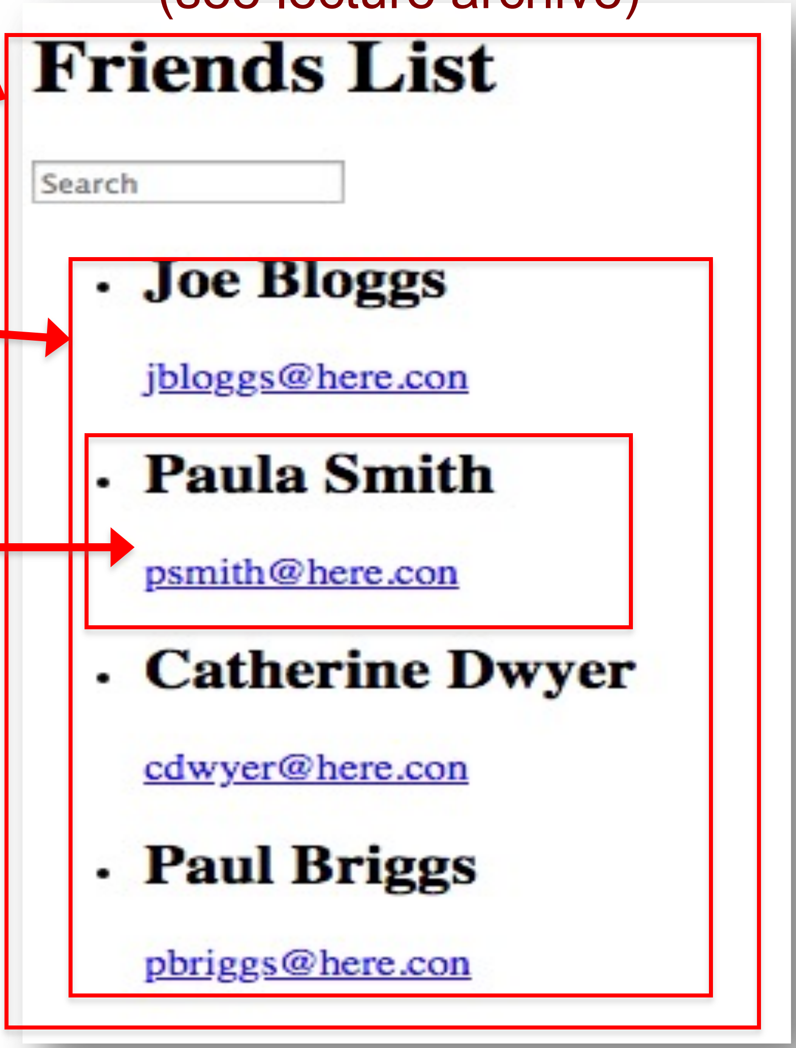
- **Signature:** useEffect(callback, dependency array)
  - **The** *callback* **contains the side effect code**

- **When is useEffect() executed?**
  1. **On mounting.**
  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:**

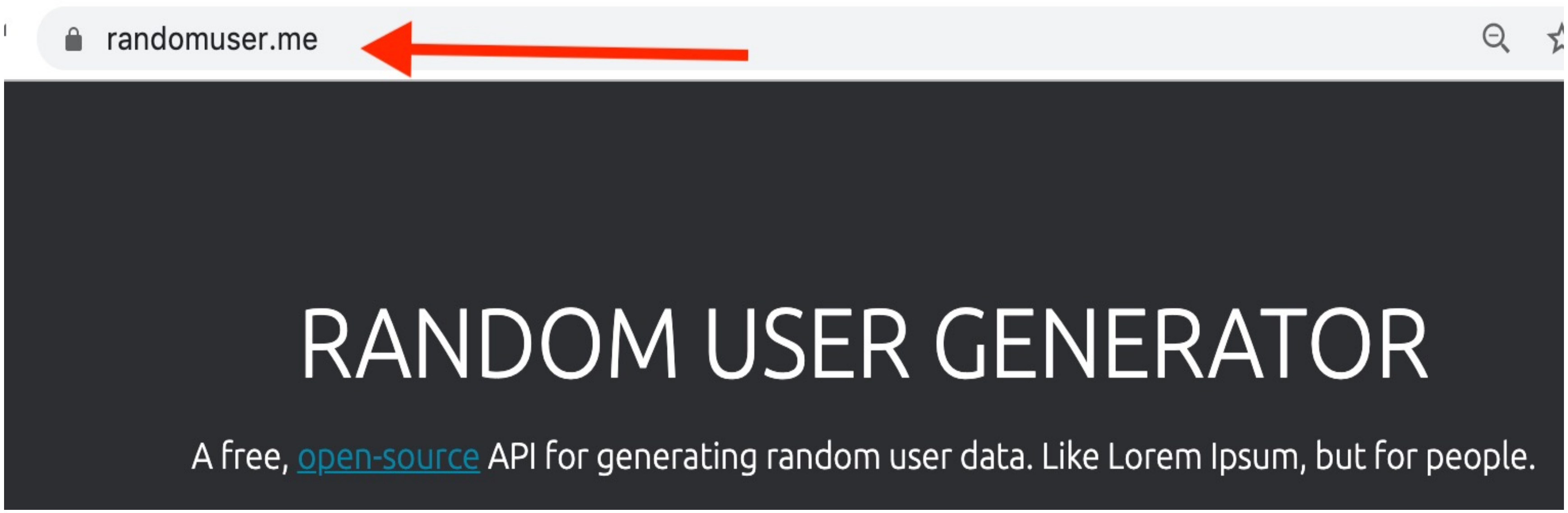
1. Manages state **(i.e. text box, full list of friends).**

2. Fetches data from a web API – side effect.

3. Computes matching friends.

4. Controls list rendering.

RandomUser open API


randomuser.me

RANDOM USER GENERATOR

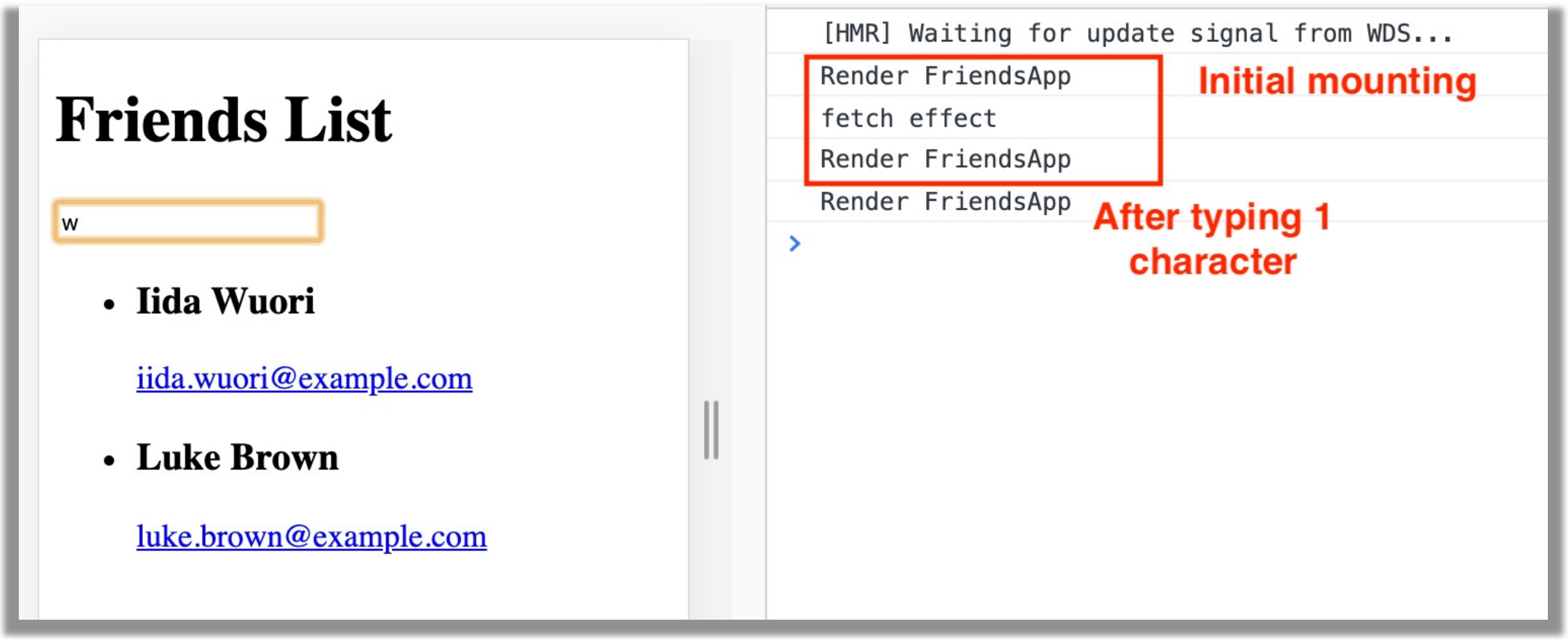
A free, open-source API for generating random user data. Like Lorem Ipsum, but for people.

- 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

- **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.**



Friends List

w

- **Iida Wuori**

  iida.wuori@example.com

- **Luke Brown**

  luke.brown@example.com

```
[HMR] Waiting for update signal from WDS...
Render FriendsApp
fetch effect
Render FriendsApp
Render FriendsApp
```

Sample App - *useEffect* Hook

- **You must allow for asynchronous nature of API calls, by**
  1. **Not ‘freezing’ the browser while waiting.**
  2. **Allowing components to render without real data.**

- **Correct solution:**

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

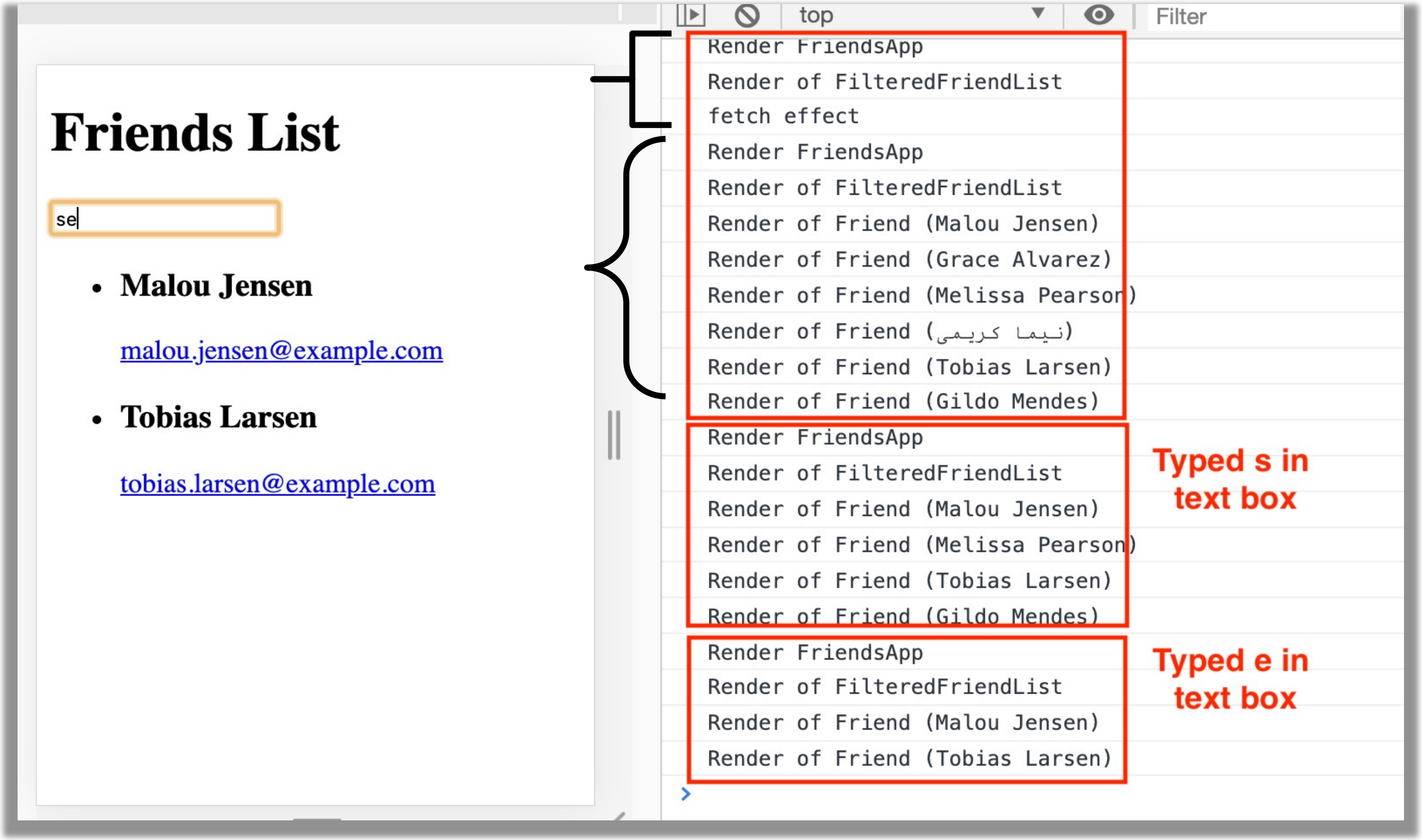
- **Incorrect solution:**

```
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)



Friends List

se

- **Malou Jensen**

  malou.jensen@example.com

- **Tobias Larsen**

  tobias.larsen@example.com

```
Render FriendsApp
Render of FilteredFriendList
fetch effect
Render FriendsApp
Render of FilteredFriendList
Render of Friend (Malou Jensen)
Render of Friend (Grace Alvarez)
Render of Friend (Melissa Pearson)
Render of Friend (نيما كريمي)
Render of Friend (Tobias Larsen)
Render of Friend (Gildo Mendes)
Render FriendsApp
Render of FilteredFriendList
Render of Friend (Malou Jensen)
Render of Friend (Melissa Pearson)
Render of Friend (Tobias Larsen)
Render of Friend (Gildo Mendes)
Render FriendsApp
Render of FilteredFriendList
Render of Friend (Malou Jensen)
Render of Friend (Tobias Larsen)
```

More to come ….

-