

# ReactJS.

The Component model

# Topics

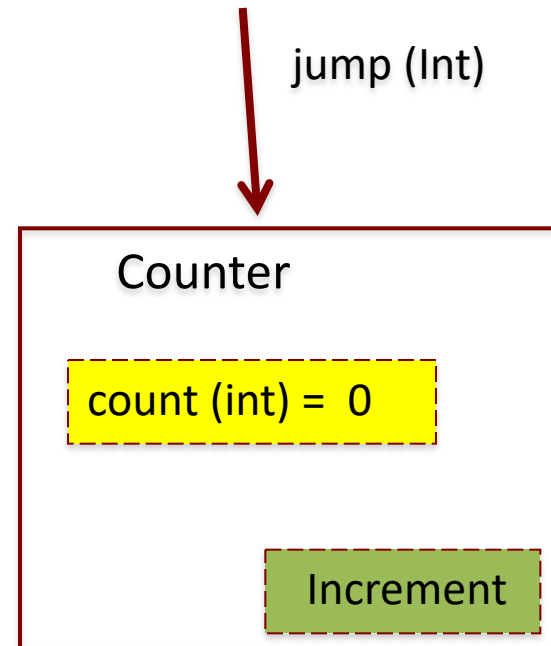
- **Component State.**
  - **Basis for dynamic, interactive UI.**
- **Data Flow patterns.**
- **Hooks.**

# 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

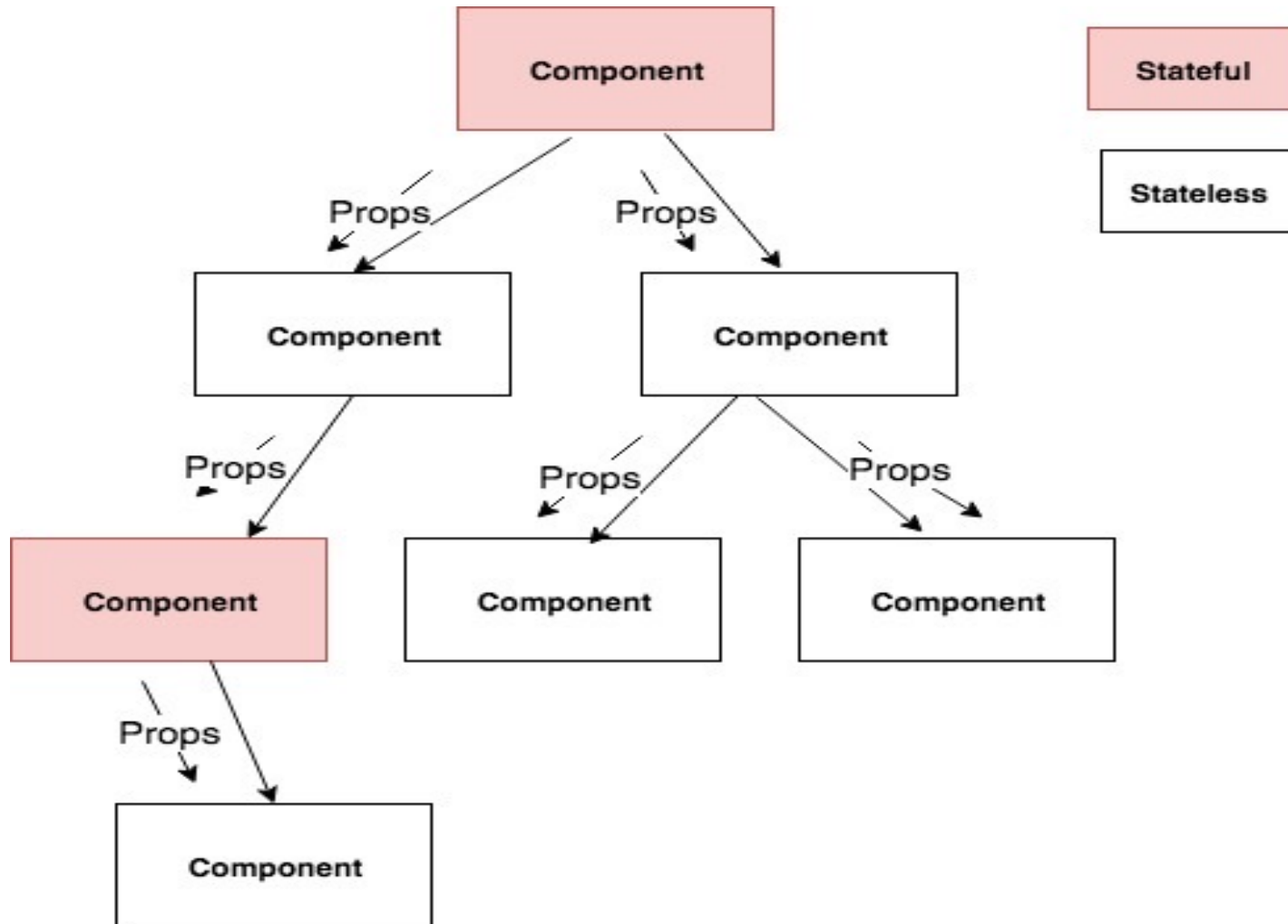
- **Component State.**



- **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 state change in a component.**
  2. **That component re-renders (re-executes).**
  3. **It recomputes the 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**

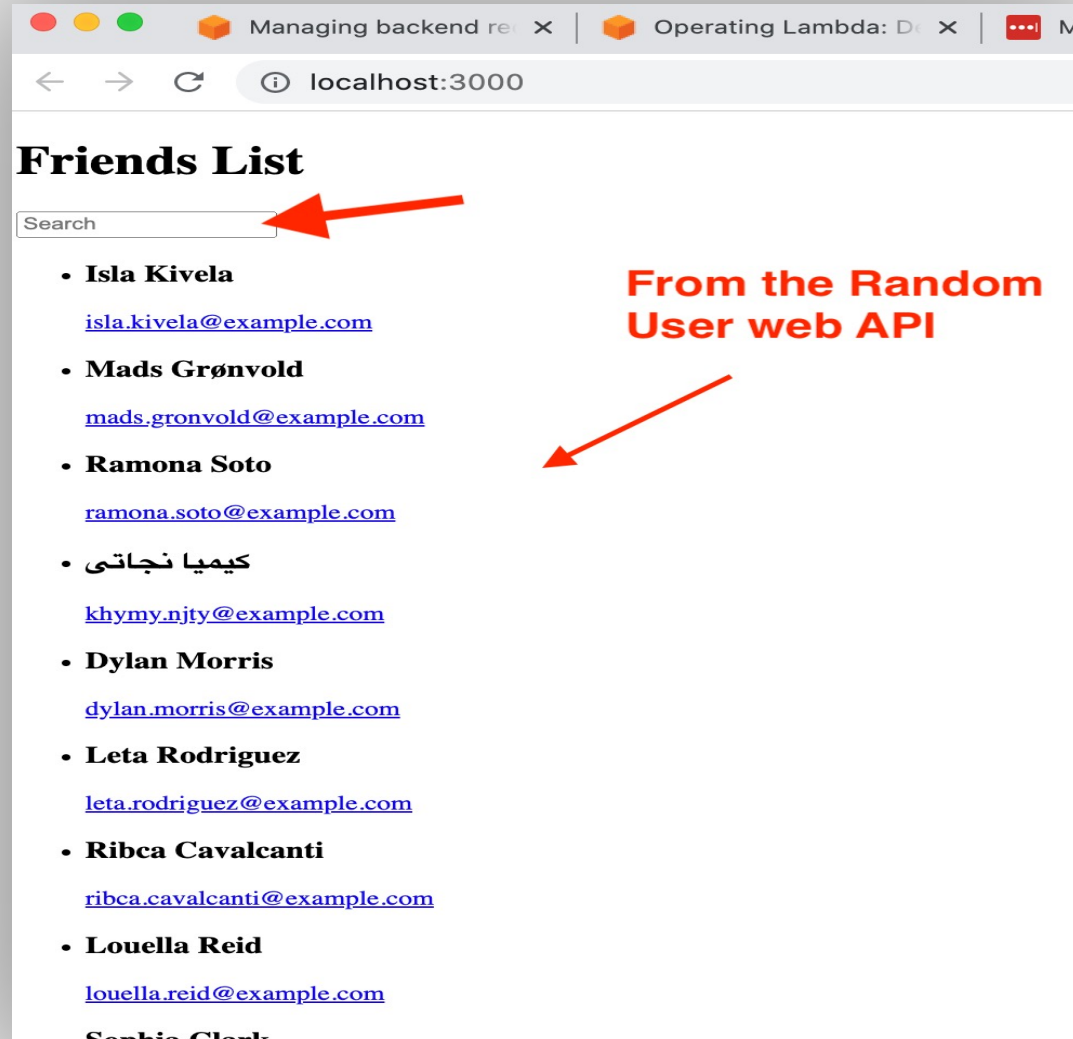
# React Hooks

- Introduced in version 16.8.0 (February 2019)
- React Hooks are:
  1. Functions (some are HOFs).
  2. That *manipulate the state and manage a component's lifecycle*.
- Examples: `useState`, `useEffect`, `useContext`, `useRef`, etc
  - ‘use’ prefix is necessary for linting tools.
- Hook usage 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

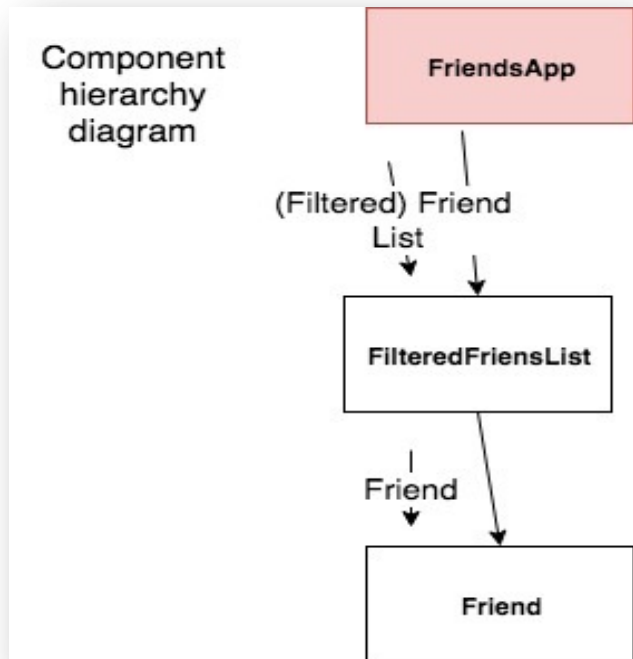
- **Used when a component needs to perform side effects.**
- **Side Effect example:**
  - **fetching data from a web API.**
  - **Subscribe to browser events, e.g. window resize.**
- **Signature:** `useEffect(callback, dependency array)`
  - **The *callback* contains the side effect code**
- **When is `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



# Sample App 1

(see lecture archive)



## Friends List

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

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.

# 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

- **Iida Wuori**

[iida.wuori@example.com](mailto:iida.wuori@example.com)

- **Luke Brown**

[luke.brown@example.com](mailto:luke.brown@example.com)

[HMR] Waiting for update signal from WDS...

Render FriendsApp

fetch effect

Render FriendsApp

Render FriendsApp

**Initial mounting**

**After typing 1  
character**

# Sample App - *useEffect* Hook

- **You must allow for asynchronous nature of API calls, by**
  1. **Not ‘freezing’ the browser while waiting for data.**
  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)

The screenshot shows a web application with a 'Friends List' section. It includes a search input field with the text 'se' and a list of two friends: Malou Jensen and Tobias Larsen, each with an email address. To the right, the DevTools component inspector is open, showing the component tree for the 'Friends List' component. The tree is divided into three sections, each highlighted with a red box. The first section, corresponding to the initial state, shows the rendering of the 'FriendsApp' component, the 'FilteredFriendList' component, and the 'fetch' effect, followed by the rendering of the 'FriendsApp' component and the 'FilteredFriendList' component, and then the rendering of six individual 'Friend' components (Malou Jensen, Grace Alvarez, Melissa Pearson, نېما كرىمى, Tobias Larsen, and Gildo Mendes). The second section, corresponding to the state after typing 's', shows the rendering of the 'FriendsApp' component, the 'FilteredFriendList' component, and the rendering of four individual 'Friend' components (Malou Jensen, Melissa Pearson, Tobias Larsen, and Gildo Mendes). The third section, corresponding to the state after typing 'e', shows the rendering of the 'FriendsApp' component, the 'FilteredFriendList' component, and the rendering of two individual 'Friend' components (Malou Jensen and Tobias Larsen). The text 'Typed s in text box' and 'Typed e in text box' is written in red next to the second and third sections, respectively.

**Friends List**

se

- **Malou Jensen**  
[malou.jensen@example.com](mailto:malou.jensen@example.com)
- **Tobias Larsen**  
[tobias.larsen@example.com](mailto: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 (نېما كرىمى)  
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)

Typed s in text box

Typed e in text box

# More to come ....

-