ReactJS.

Thinking in React

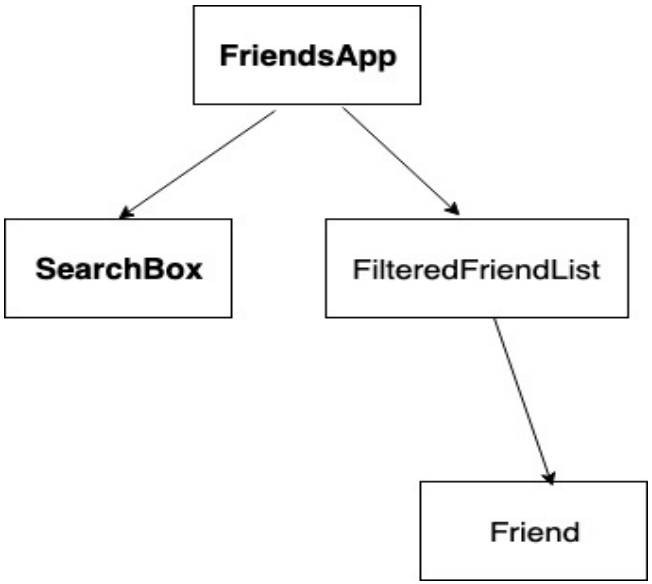
Developing a React web app

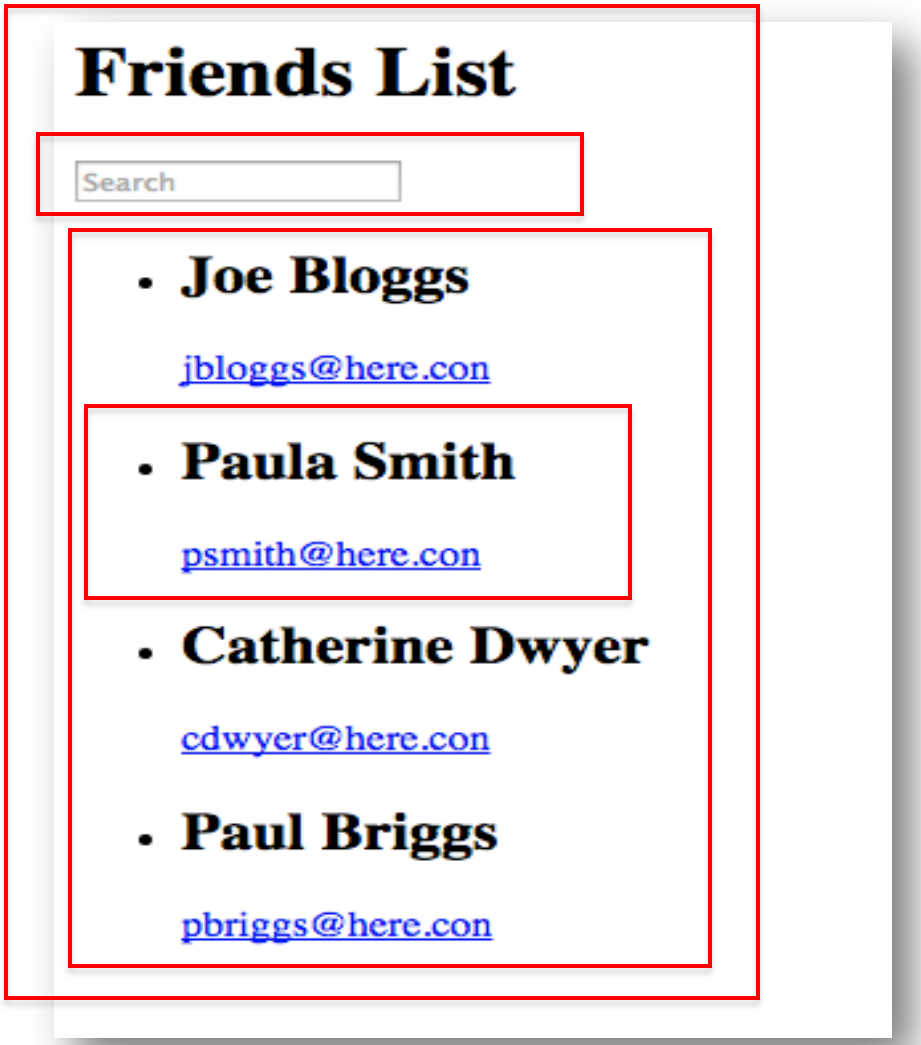
- **Step 1: Break the UI into a** component hierarchy**.**
- **Stap 2: Build a** static version **of the app.**
- **Step 3: Identify the** minimal **representation of** UI state**.**
- **Step 4: Identify where your state should live.**
- **Step 5: Add inverse data flow (Data down, Actions up), if required.**

Starting point.

- **At the start of the development process we have:**
  1. **A mock-up of the UI.**
  2. **(Optionally) A JSON representation of the web API data model.**

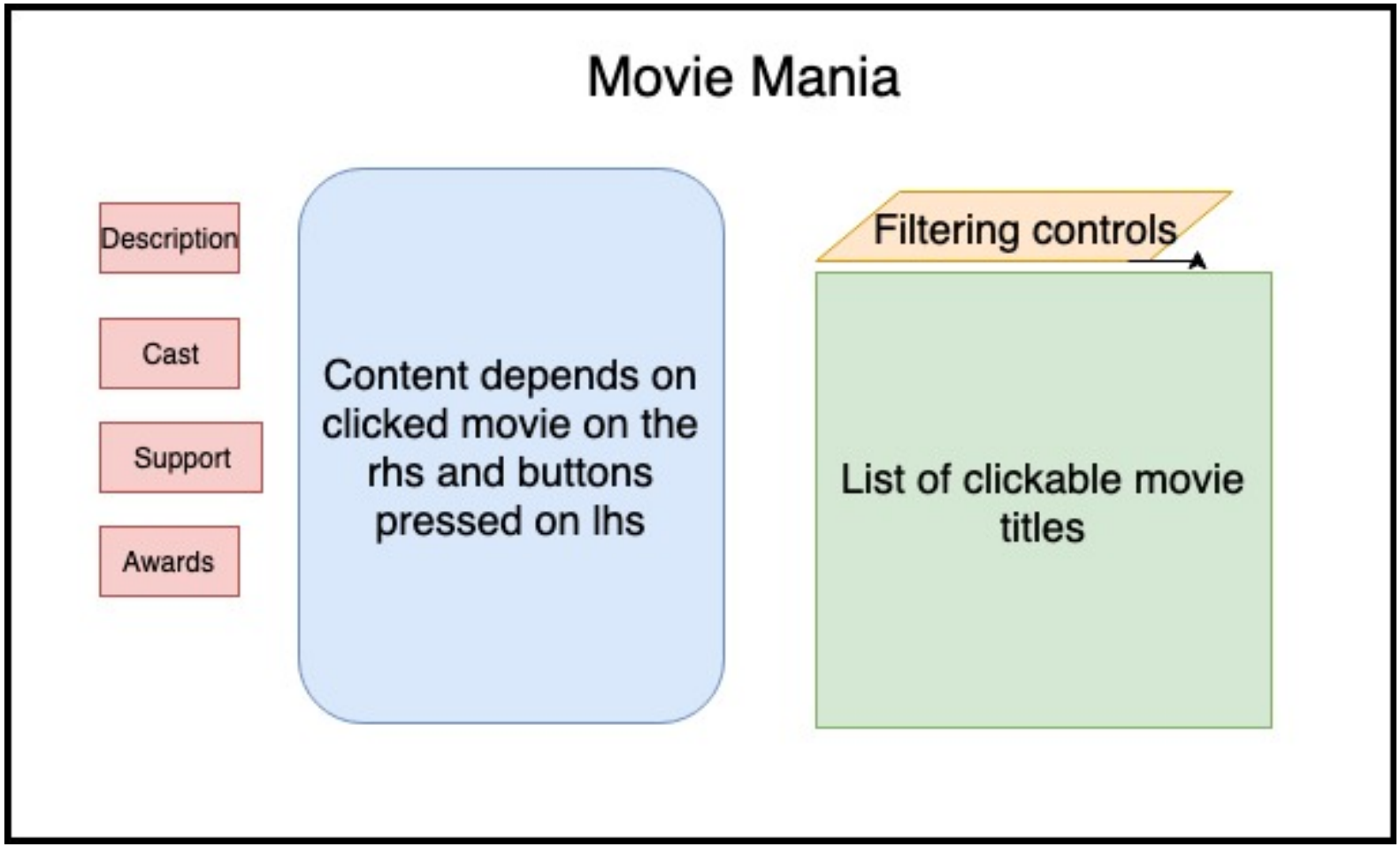
Step 1: Break the UI into a *component hierarchy*.



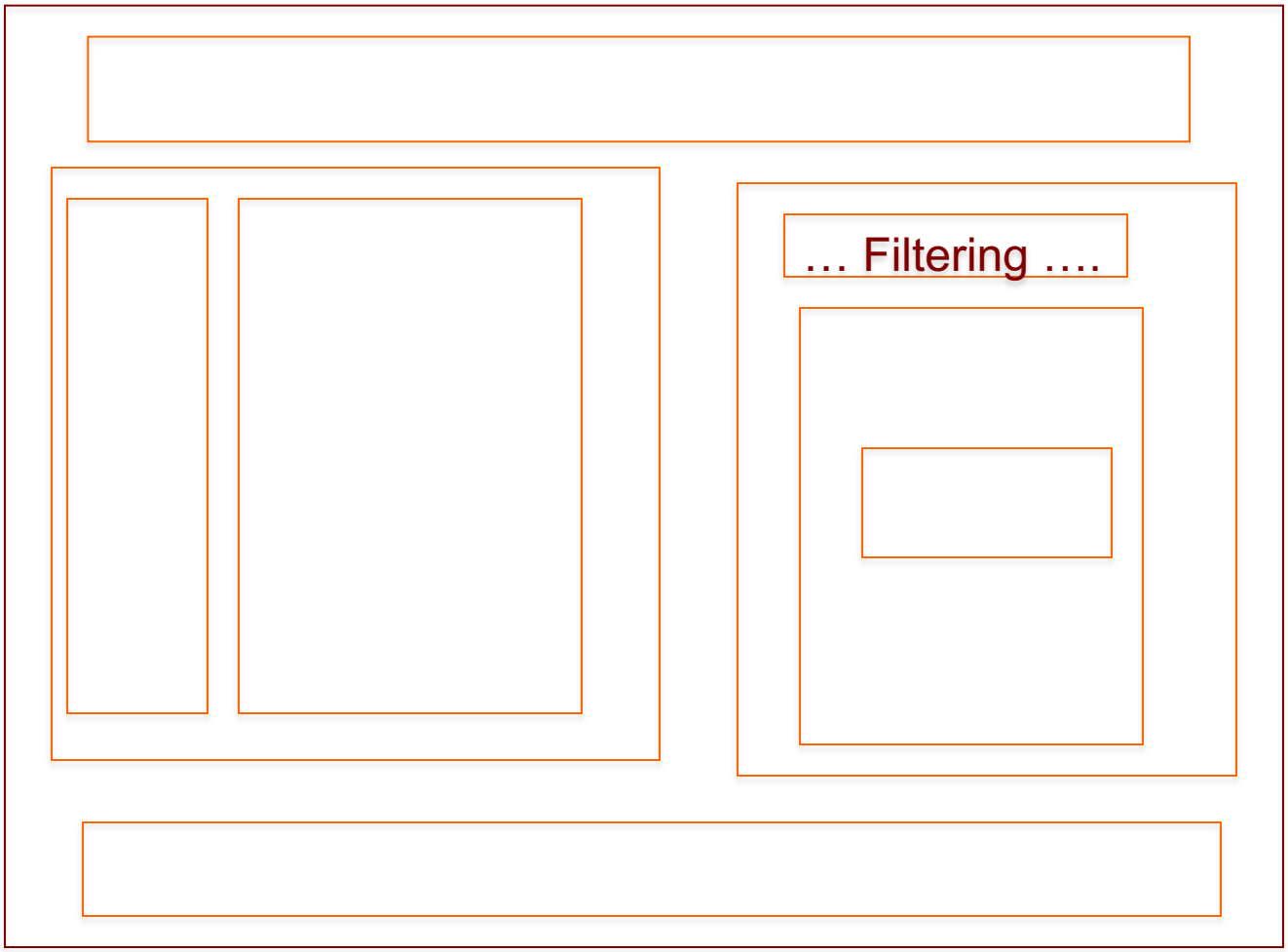


- **Possibly use the data model as a guide.**
  - **The UI and data models tend to adhere to the same information architecture.**

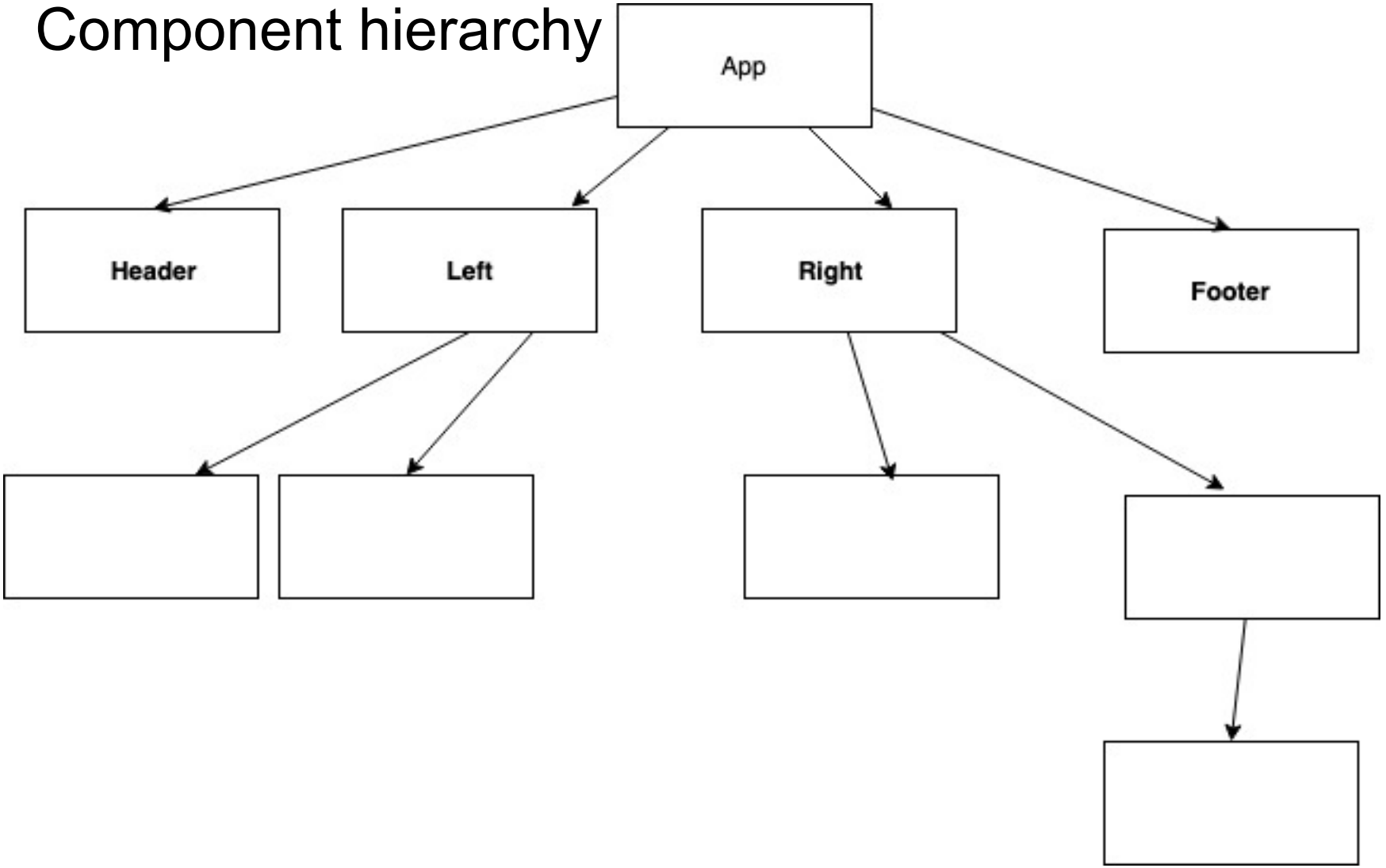
Sample App – Mock UI

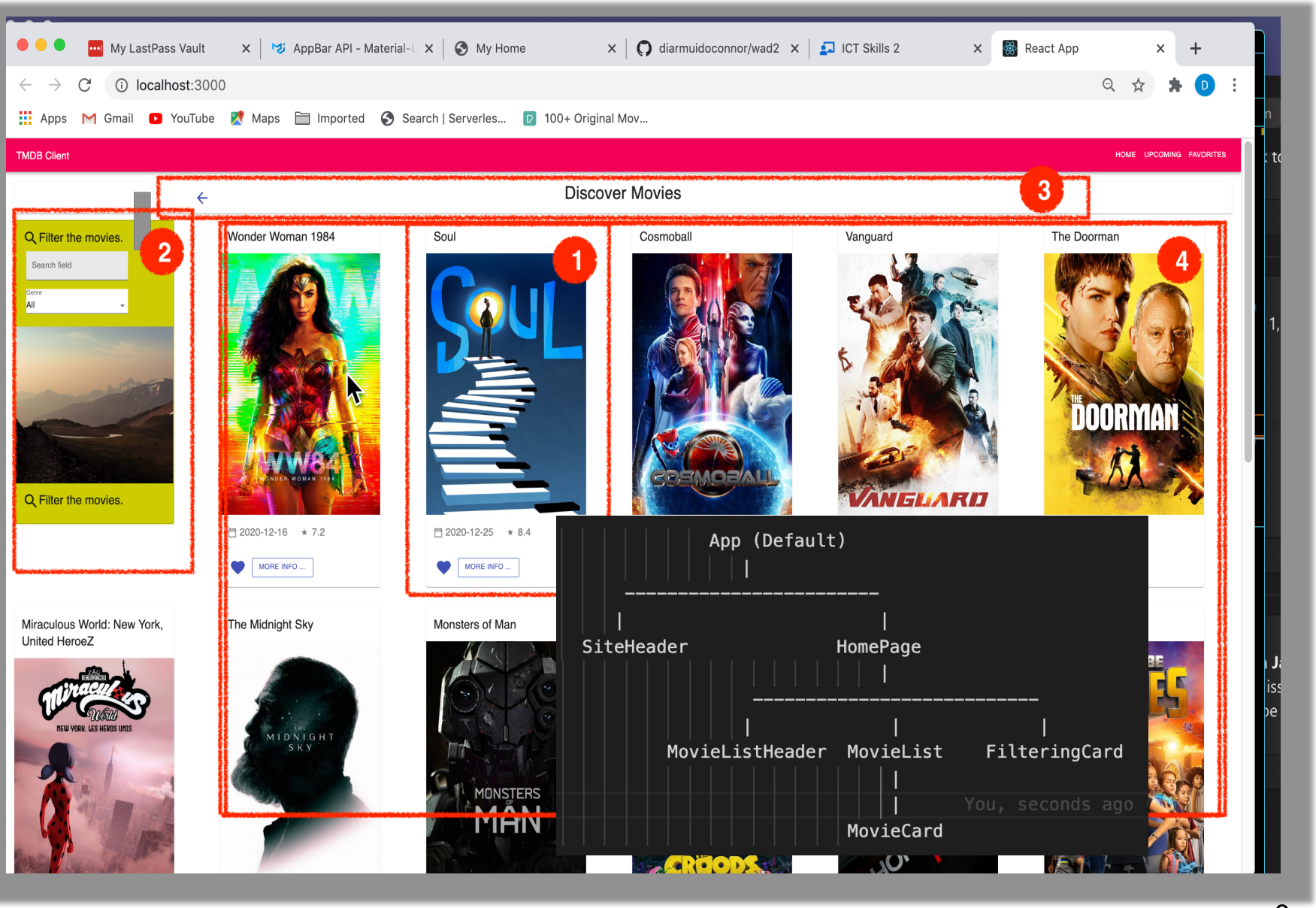


Sample App – Component breakdown



Sample App - Component hierarchy




My LastPass Vault
AppBar API - Material-U
My Home
diarmuidoconnor/wad2
ICT Skills 2
React App
localhost:3000
Apps
Gmail
YouTube
Maps
Imported
Search | Serverles...
100+ Original Mov...
TMDB Client
HOME
UPCOMING
FAVORITES
Discover Movies
3
Filter the movies.
Search field
Genre
All
2
Filter the movies.
Wonder Woman 1984
2020-12-16
★ 7.2
MORE INFO ...
Soul
1
2020-12-25
★ 8.4
MORE INFO ...
Cosmoball
Vanguard
The Doorman
4
Miraculous World: New York, United HeroeZ
The Midnight Sky
Monsters of Man
App (Default)
SiteHeader
HomePage
MovieListHeader
MovieList
FilteringCard
You, seconds ago
MovieCard

Step 1: Break the UI into a component hierarchy.

- **Additional crtieria for devising component breakdown:**
    1. **The** single responsibility principle.
    2. **If it's doing too much, break it up.**
    3. **If it has too much code, break it up.**

- **[Same principles when dealing with Object Oriented design.]**

Stap 2: Build a static version.

- **Use Storybook to build a component library.**
  - **Helps determine component** prop requirements**.** ****
  - **Start with 'leaf' components, and work up the hierarchy, e.g. MovieCard → MovieList -> FilterControls ……**
  - **Consider** multiple stories **for a component, e.g. prop boundry values, default value.**
- **Using a sample data set, render the UI** but ignore all interactivity.

- **Components should only have a** return **statement.**
  - **No hooks, event handlers, algorithms, yet.**

- Design Principle**: Decouple structure from interactivity, initially.**
- **"Lots of typing but little thinking."**

Step 3: Identify the *minimal* (but complete) representation of **UI** *state*.

- **Try to keep as many components as possible stateless.**
  - **Stateless components simply render props.**
- **Follow the DRY principle (Don't Repeat Yourself).**
- **Common app pattern – A stateful component computes the props for its subordinates based on current state, domain data and/or its own props.**

Step 3: Identify UI state.

- **What shouldn't go in State?**
  1. **Computed data, e.g. subset of matching friends.**
  2. **Copies of props: Props are the 'source of truth'.**
     - **Unless props' previous value(s) effects rendering.**
  3. **React components; State should always be JSON-serializable.**
  4. **Domain Data - data retrieved from a Web API/service.**
     - **Use a 3rd party state management module, e.g. react-redux, or a React** Context **feature.**

Step 3: Identify UI state.

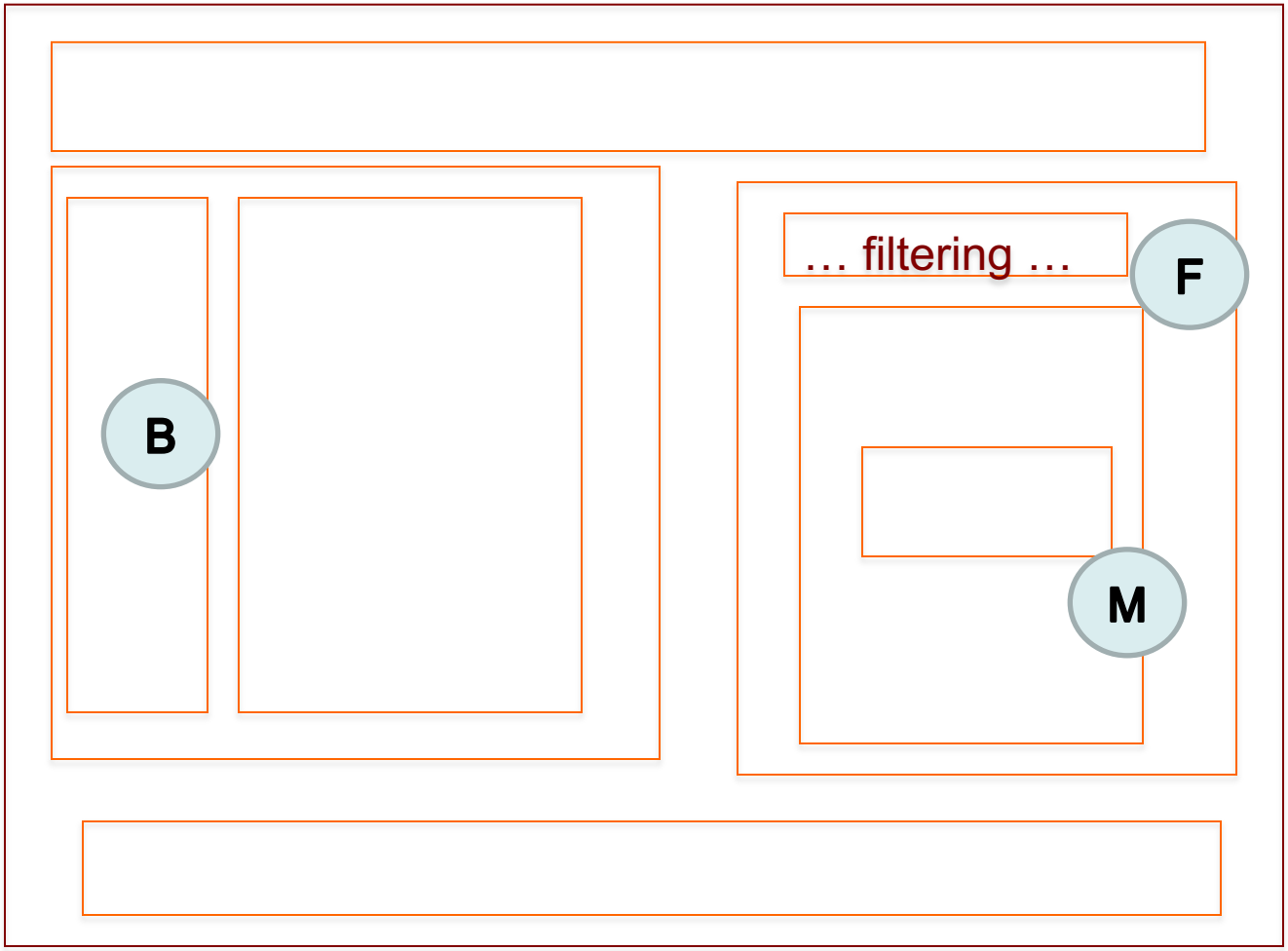
- **What should go in state?**
  - **User inputs – check box, menu, radio button. input text fields.**
  - **Data that an event handlers changes, e.g. counter, text box value.**

- **How to identify state:**
  1. **Identify all of the places where data appears in the UI.**
  2. **For each one, ask a set of questions:**
     I. **Is it passed in via props? If so, probably isn't state.**
     II. **Is it modifiable/interactive? If not, probably isn't state.**
     III. **Can you compute it based on any other state or props? If so, it's not state.**

Example: Filtered Friends app

- **Think of all of the places where data appears in the UI:**
  1. **Full List of friends.**
     - **Supplied from web API → Not state.**
       - **We should have used a** Context **instead.**
  2. **Search text.**
     - **Modifiable, User input → State.**
  3. **Filtered list of friends.**
     - **Computed → Not State.**
  4. **Friend details (name, etc)**
     - **Passed in as props, Not modifiable → Not state**

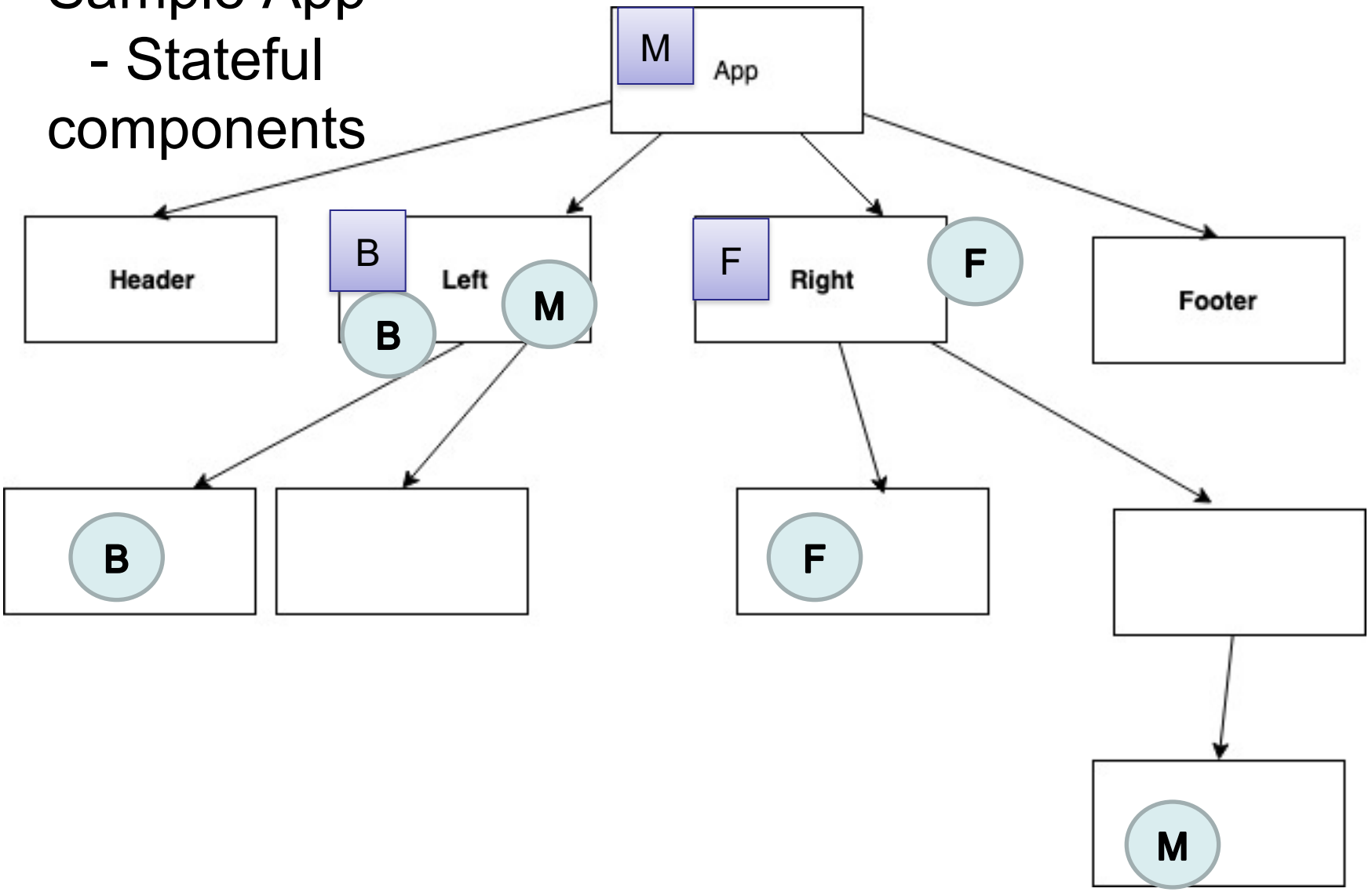
Sample App - Identify UI state



Step 4: Identify where state should live.

- **For each piece of UI state, go through this process:**
  1. **Identify every component that renders something based on its value.**
  2. **From 1 above, identify the 'common' ancestor component.**
  3. **Add** useState **hook to selected ancestor component.**

Sample App - Stateful components



Sample: Filtered Friends app

- **Only 1 state variable (searchText).**
- **Design A – '1-way data flow' design.**
  - FriendsApp **component needs to display the text & use it to compute the filtered list of friends.**
  - **No other component uses this state.**

- **Design B – 'Data Down, Action Up' design.**
  - FriendsApp **component needs it to compute the filtered list of friends.**
  - SearchBox **component needs to display it in the text field.**

  → FriendsApp **is the 'common ancestor' component**

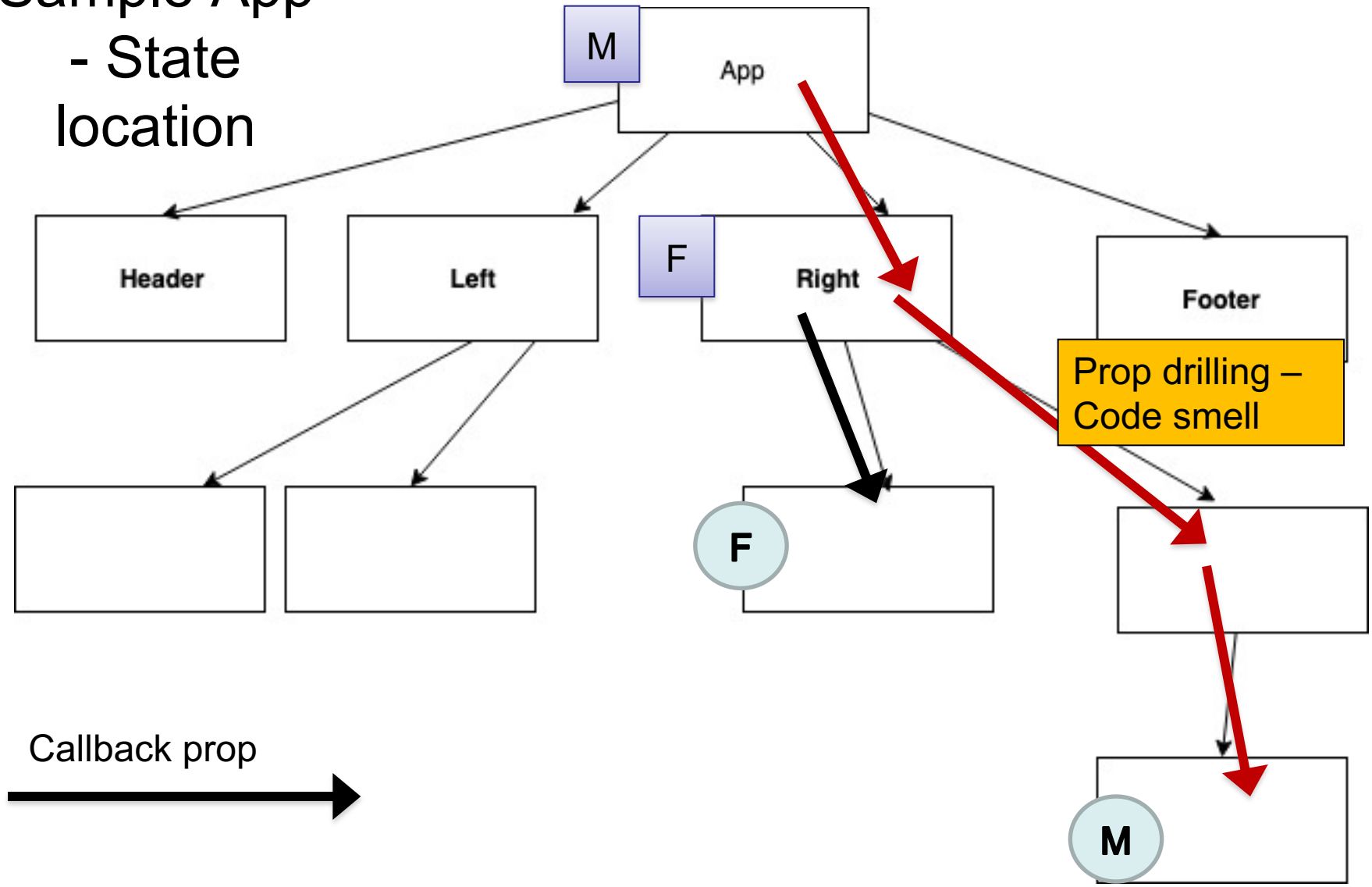
Step 5: Add inverse data flow

- Problem: **A component's state change is determined by an event that occurs in a subordinate component.**
  - **The event is usuallu triggered by user interaction.**
  - **The subordinate component must communicate the** event **to the (superordinate) stateful component.**

- Solution: **Data down, Action Up pattern:**
  - **Statefull component passes a callback function (prop) to the subordinate.**
  - **Subordinate invokes callback when event fires.**

- **Update Storybook stories to reflect callback prop,**

Sample App - State location



Developing a React web app

- **Step 1: Break the UI into a** component hierarchy**.**
- **Stap 2: Build a static version in React.**
- **Step 3: Identify the** minimal **representation of UI state.**
- **Step 4: Identify where your state should live.**
- **Step 5: Add inverse data flow, if required.**