O'REILLY®

From Local State to Global Control

September 2023

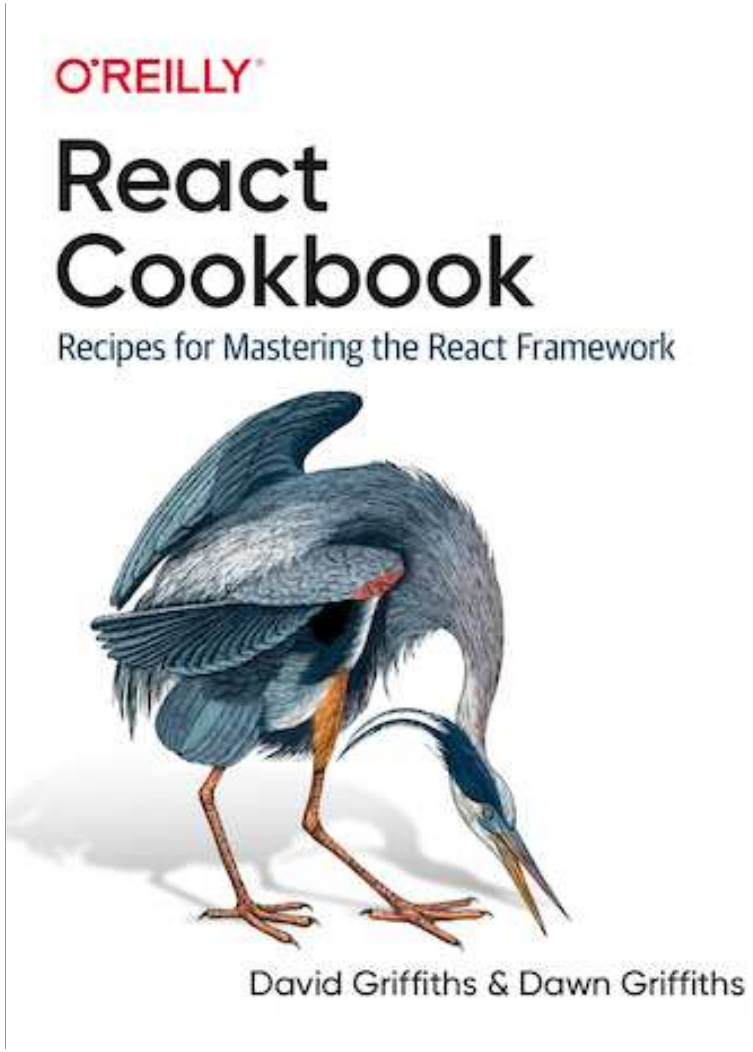
Introduction

About me

- Author

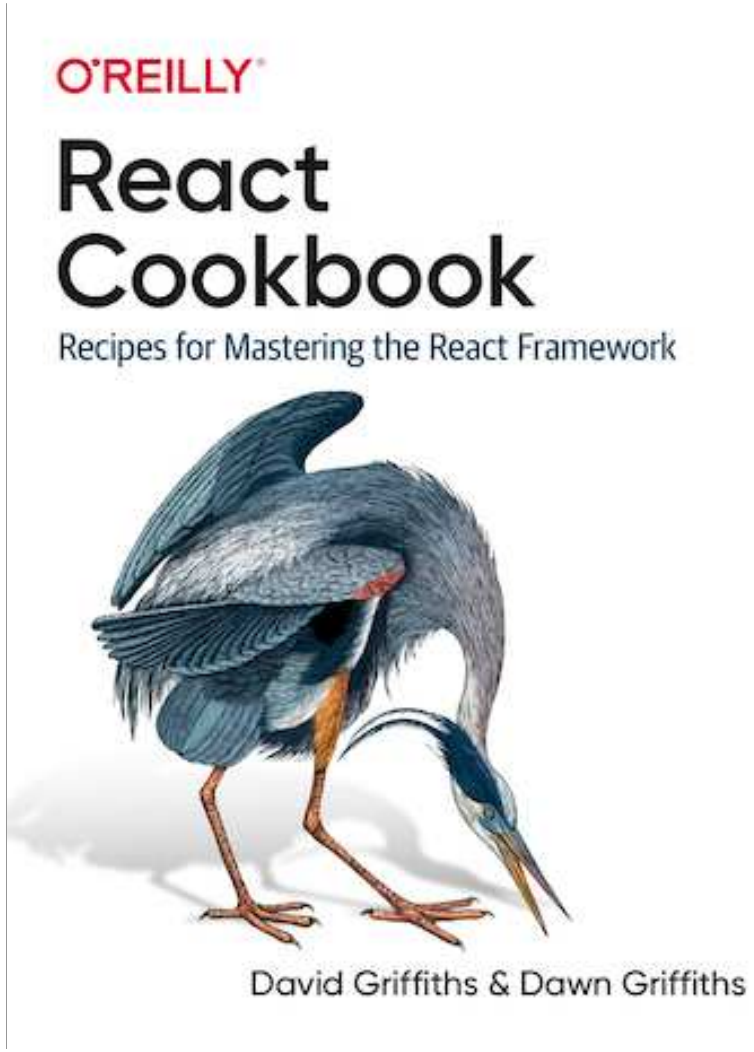
About me

- Author



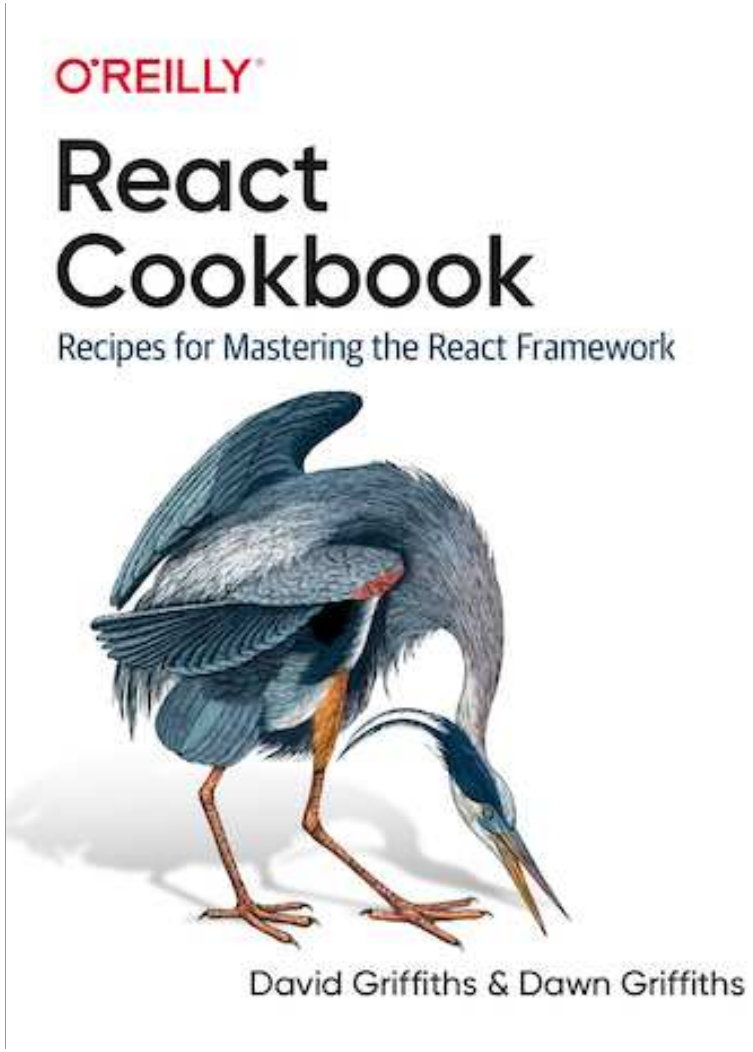
About me

- Author
- https://www.herescreen.com



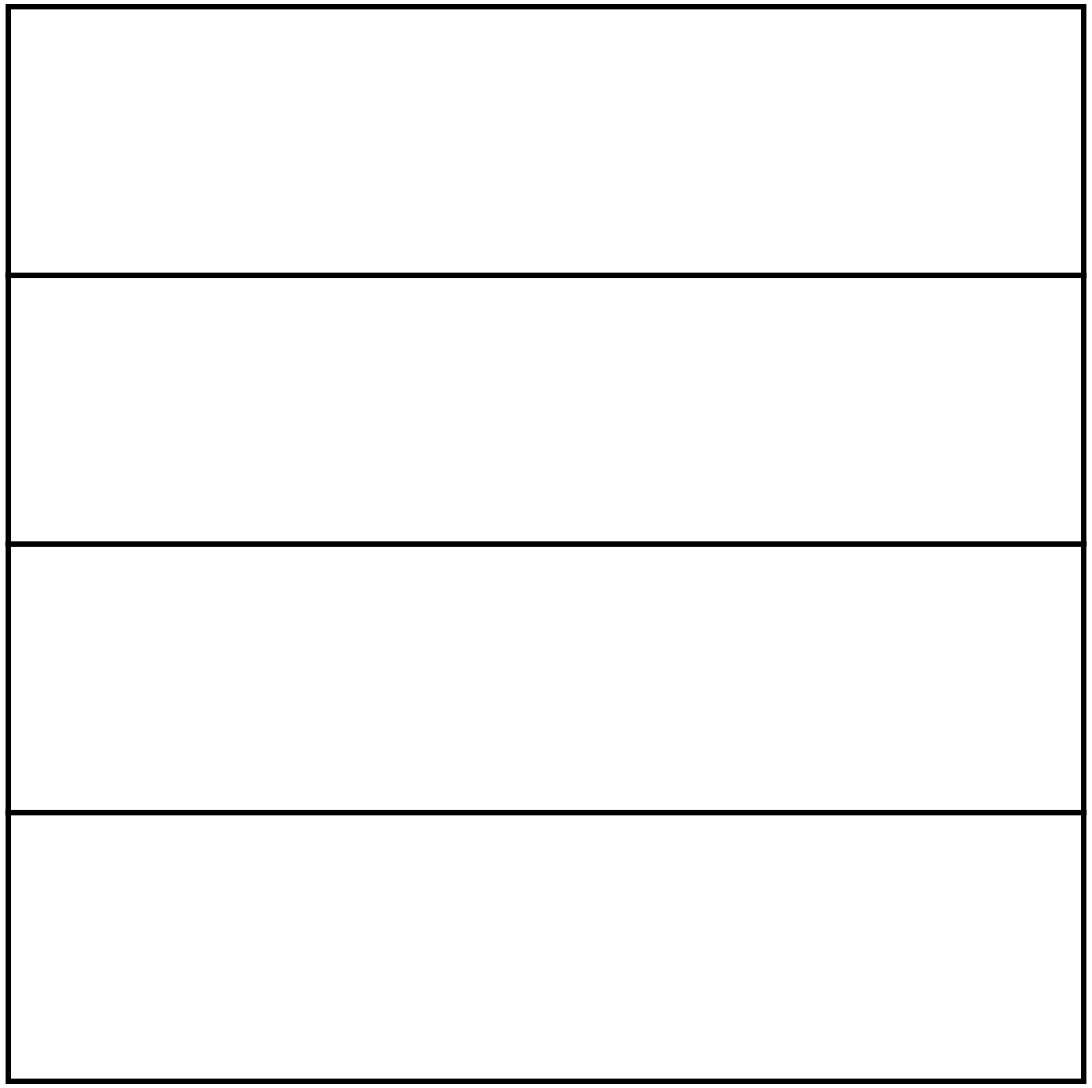
About me

- Author
- https://www.herescreen.com
- https://linktr.ee/dogriffiths

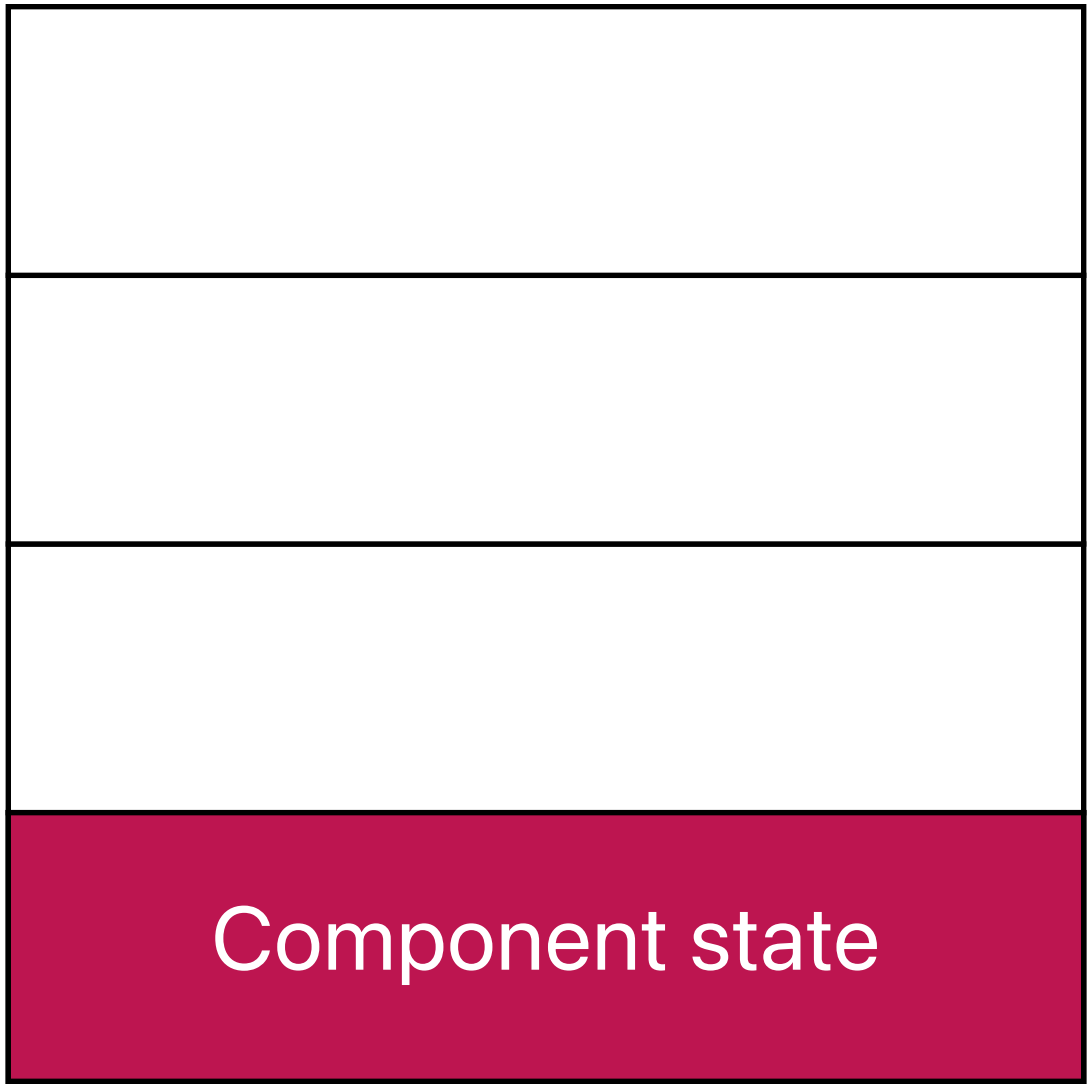


Component state

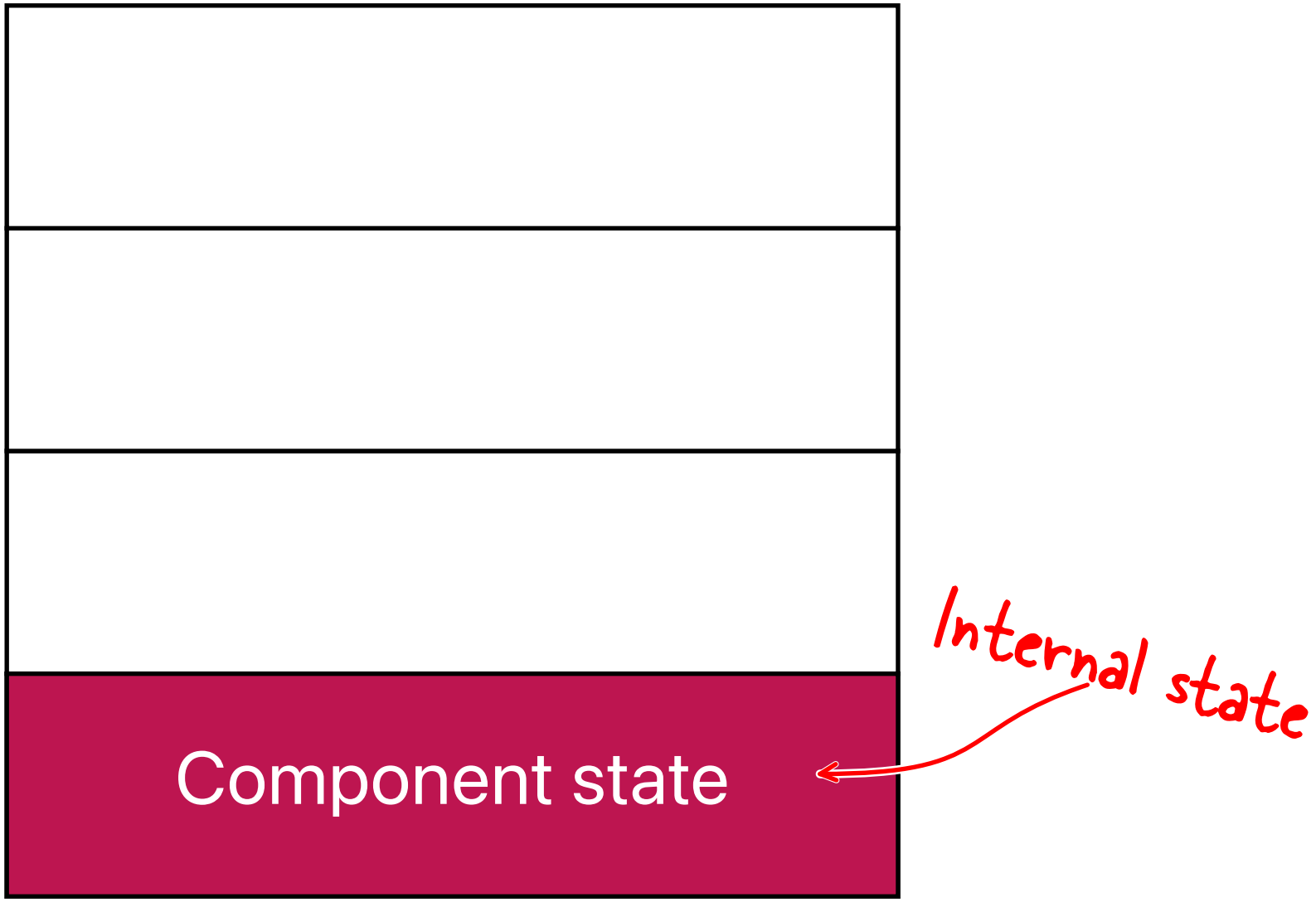
Different levels of state



Different levels of state



Different levels of state



Component state in React

Component state in React

- React 'components' are generated using JavaScript functions

Component state in React

- React 'components' are generated using JavaScript functions
- And components need to be able to remember things

Component state in React

- React 'components' are generated using JavaScript functions
- And components need to be able to remember things
- But how do you give a function a 'memory'?

Component state in React

- React 'components' are generated using JavaScript functions
- And components need to be able to remember things
- But how do you give a function a 'memory'?
- With `useState`

useState

```
const [name, setName] = useState("")
return <>
</>
```

useState

```
const [name, setName] = useState("")
return <>
  Name: {name}
</>
```

useState

```
const [name, setName] = useState("")
return <>
  Name: {name}
  <input value={name}
         onChange={(evt) => setName(evt.target.value)}
      />
</>
```

Simplify code with custom hooks (recipe 3.4)

Simplify code with custom hooks (recipe 3.4)

Simplify code with custom hooks (recipe 3.4)

- If you need a lot of code to manage a piece of state...

Simplify code with custom hooks (recipe 3.4)

- If you need a lot of code to manage a piece of state...
- ...use a custom hook

Simplify code with custom hooks (recipe 3.4)

- If you need a lot of code to manage a piece of state...
- ...use a custom hook
- A custom hook is just a JavaScript function named `use...`

The useClock custom hook

```
const time = useClock('HH:mm:ss')
```

The useClock custom hook

```
const time = useClock('HH:mm:ss')
const date = useClock('MMMM DD, YYYY')
```

The useClock custom hook

```
const time = useClock('HH:mm:ss')
const date = useClock('MMMM DD, YYYY')
const tickThreeSeconds = useClock(3000)
```

The useClock custom hook: useClock.js

```
const useClock = (formatOrInterval) => {
}

export default useClock
```

The useClock custom hook: useClock.js

```
const useClock = (formatOrInterval) => {
  const format = typeof formatOrInterval === 'string'
      ? formatOrInterval : 'YYYY-MM-DDTHH:mm:ss.SSS'
}

export default useClock
```

The useClock custom hook: useClock.js

```
const useClock = (formatOrInterval) => {
  const format = typeof formatOrInterval === 'string'
      ? formatOrInterval : 'YYYY-MM-DDTHH:mm:ss.SSS'
  const interval = typeof formatOrInterval === 'number' ? formatOrInterval : 500
}

export default useClock
```

The useClock custom hook: useClock.js

```
import { useEffect, useState } from 'react'
import moment from 'moment'

const useClock = (formatOrInterval) => {
  const format = typeof formatOrInterval === 'string'
      ? formatOrInterval : 'YYYY-MM-DDTHH:mm:ss.SSS'
  const interval = typeof formatOrInterval === 'number' ? formatOrInterval : 500
  const [response, setResponse] = useState(
    moment(new Date()).format(format)
  )

}

export default useClock
```

The useClock custom hook: useClock.js

```
import { useEffect, useState } from 'react'
import moment from 'moment'

const useClock = (formatOrInterval) => {
  const format = typeof formatOrInterval === 'string'
      ? formatOrInterval : 'YYYY-MM-DDTHH:mm:ss.SSS'
  const interval = typeof formatOrInterval === 'number' ? formatOrInterval : 500
  const [response, setResponse] = useState(
    moment(new Date()).format(format)
  )

  useEffect(() => {
    const newTimer = setInterval(() => {
      setResponse(moment(new Date()).format(format))
    }, interval)
  }, [format, interval])
}

export default useClock
```

The useClock custom hook: useClock.js

```
import { useEffect, useState } from 'react'
import moment from 'moment'

const useClock = (formatOrInterval) => {
  const format = typeof formatOrInterval === 'string'
      ? formatOrInterval : 'YYYY-MM-DDTHH:mm:ss.SSS'
  const interval = typeof formatOrInterval === 'number' ? formatOrInterval : 500
  const [response, setResponse] = useState(
    moment(new Date()).format(format)
  )

  useEffect(() => {
    const newTimer = setInterval(() => {
      setResponse(moment(new Date()).format(format))
    }, interval)

    return () => clearInterval(newTimer)
  }, [format, interval])
}

export default useClock
```

The useClock custom hook: useClock.js

```
import { useEffect, useState } from 'react'
import moment from 'moment'

const useClock = (formatOrInterval) => {
  const format = typeof formatOrInterval === 'string'
      ? formatOrInterval : 'YYYY-MM-DDTHH:mm:ss.SSS'
  const interval = typeof formatOrInterval === 'number' ? formatOrInterval : 500
  const [response, setResponse] = useState(
    moment(new Date()).format(format)
  )

  useEffect(() => {
    const newTimer = setInterval(() => {
      setResponse(moment(new Date()).format(format))
    }, interval)

    return () => clearInterval(newTimer)
  }, [format, interval])

  return response
}

export default useClock
```

Managing complex component state (recipe 3.1)

Managing complex component state (recipe 3.1)

Managing complex component state (recipe 3.1)

- Some components might have many `useState()` calls

Managing complex component state (recipe 3.1)

- Some components might have many `useState()` calls
- There might be complex code that then uses those values

Managing complex component state (recipe 3.1)

- Some components might have many `useState()` calls
- There might be complex code that then uses those values
- You can extract from the complex state code into a **reducer**

Create a reducer: reducer.js

```
function reducer(state, action) {
}

export default reducer
```

Create a reducer: reducer.js

```
function reducer(state, action) {
  switch (action.type) {
  }
}

export default reducer
```

Create a reducer: reducer.js

```
function reducer(state, action) {
  switch (action.type) {
    ....
    case 'shuffle': {
      let newState = { ...state }
      do {
        for (let i = 0; i < 300; i++) {
          newState = reducer(
            { ...newState },
            {
              type: 'move',
              payload: Math.floor(Math.random() * 9),
            }
          )
        }
      } while (newState.complete)
      return newState
    }
    default: {
      throw new Error('Unknown action: ' + action.type)
    }
  }
}

export default reducer
```

Using a reducer

```
import reducer from './reducer'
```

Using a reducer

```
import reducer from './reducer'
...
const [state, dispatch] = useReducer(reducer, {
    items: ['4', '1', '2', '7', '6', '3', null, '5', '8'],
  })
```

Using a reducer

```
import reducer from './reducer'
...
const [state, dispatch] = useReducer(reducer, {
    items: ['4', '1', '2', '7', '6', '3', null, '5', '8'],
  })
```

```
...
return <>
  <p>There are {state.items.length} items</p>
</>
```

Using a reducer

```
import reducer from './reducer'
...
const [state, dispatch] = useReducer(reducer, {
    items: ['4', '1', '2', '7', '6', '3', null, '5', '8'],
  })
...
return <>
  <p>There are {state.items.length} items</p>
  <button onClick={() => dispatch({ type: 'shuffle' })}>
     Shuffle
  </button>
</>
```

Make reducers "undo-able" (recipe 3.2)

Make reducers "undo-able" (recipe 3.2)

Make reducers "undo-able" (recipe 3.2)

- Would be nice to allow a user to undo things

Make reducers "undo-able" (recipe 3.2)

- Would be nice to allow a user to undo things
- Reducers make that simpler

Make reducers "undo-able" (recipe 3.2)

- Would be nice to allow a user to undo things
- Reducers make that simpler
- You create a `useUndoReducer` hook

Make reducer's "undo-able"

```
import reducer from './reducer'
import { useReducer } from 'react'

import './Puzzle.css'

const Puzzle = () => {
  const [state, dispatch] = useReducer(reducer, {
    items: ['4', '1', '2', '7', '6', '3', null, '5', '8'],
  })
```

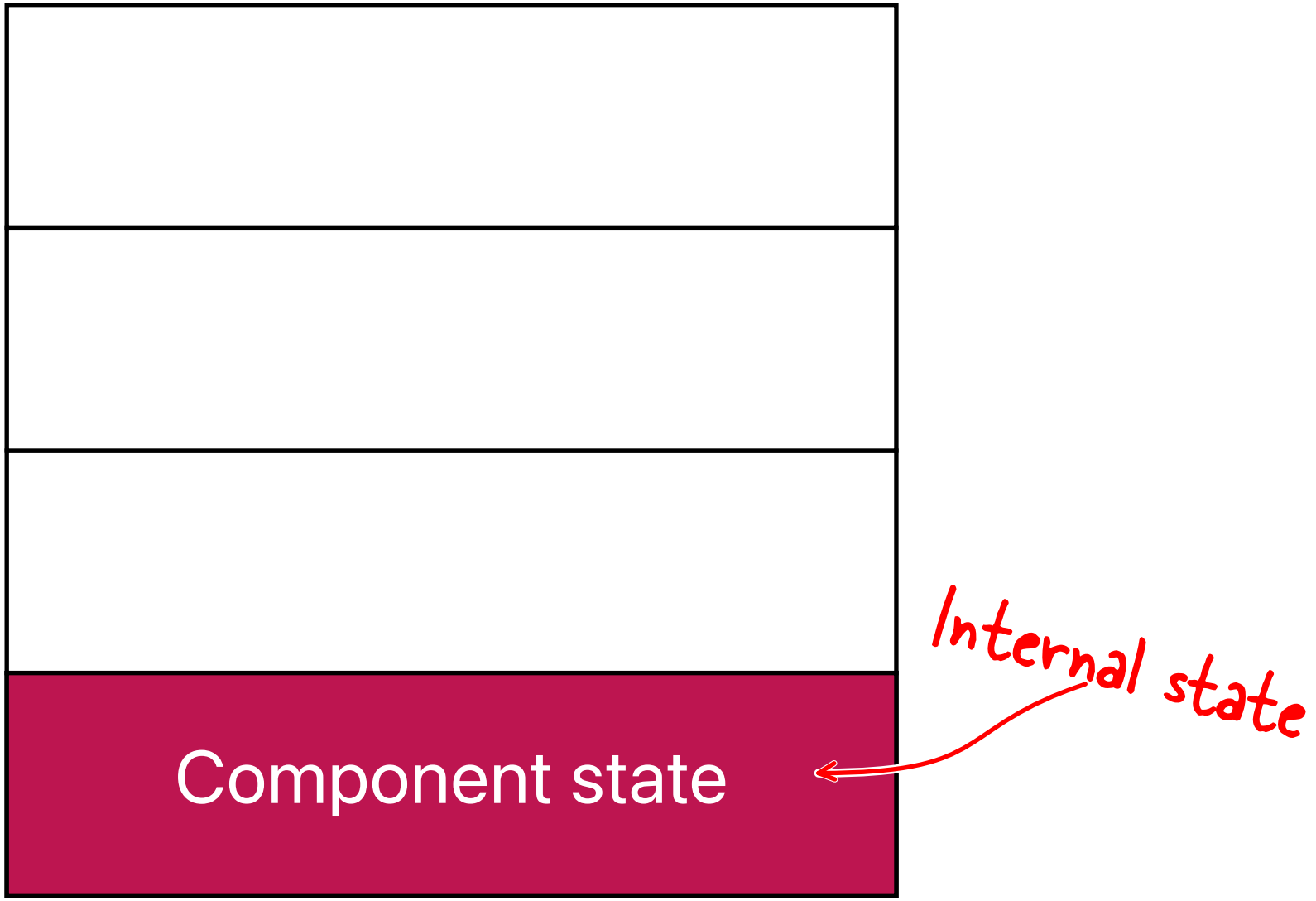
Make reducer's "undo-able"

```
import reducer from './reducer'
import useUndoReducer from './useUndoReducer'

const Puzzle = () => {
  const [state, dispatch] = useUndoReducer(reducer, {
    items: ['4', '1', '2', '7', '6', '3', null, '5', '8'],
  })
....
```

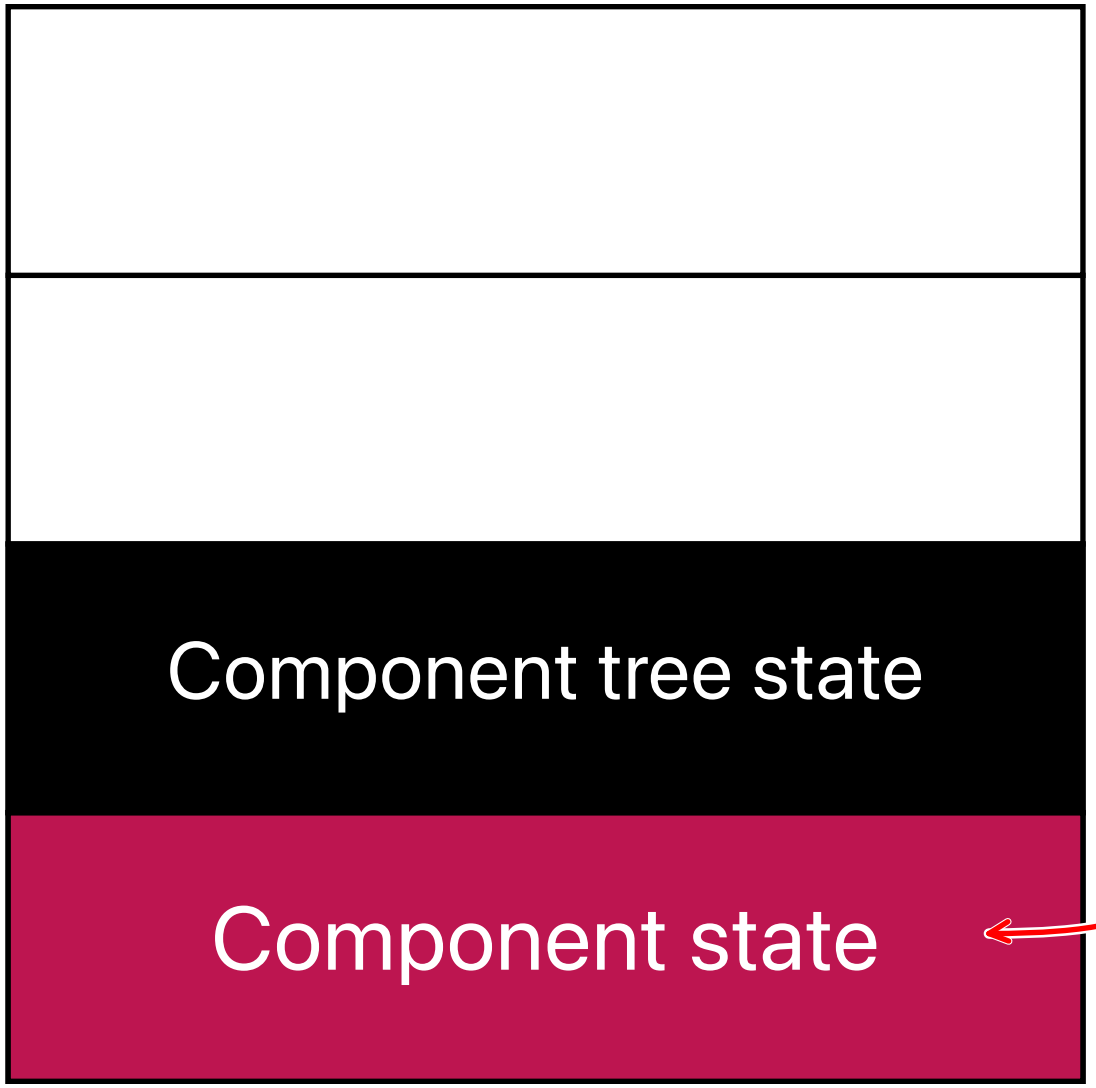
Make reducer's "undo-able"

```
import reducer from './reducer'
import useUndoReducer from './useUndoReducer'

const Puzzle = () => {
  const [state, dispatch] = useUndoReducer(reducer, {
    items: ['4', '1', '2', '7', '6', '3', null, '5', '8'],
  })
....
      <button
        onClick={() => dispatch({ type: 'undo' })}
      >
        Undo
      </button>
```

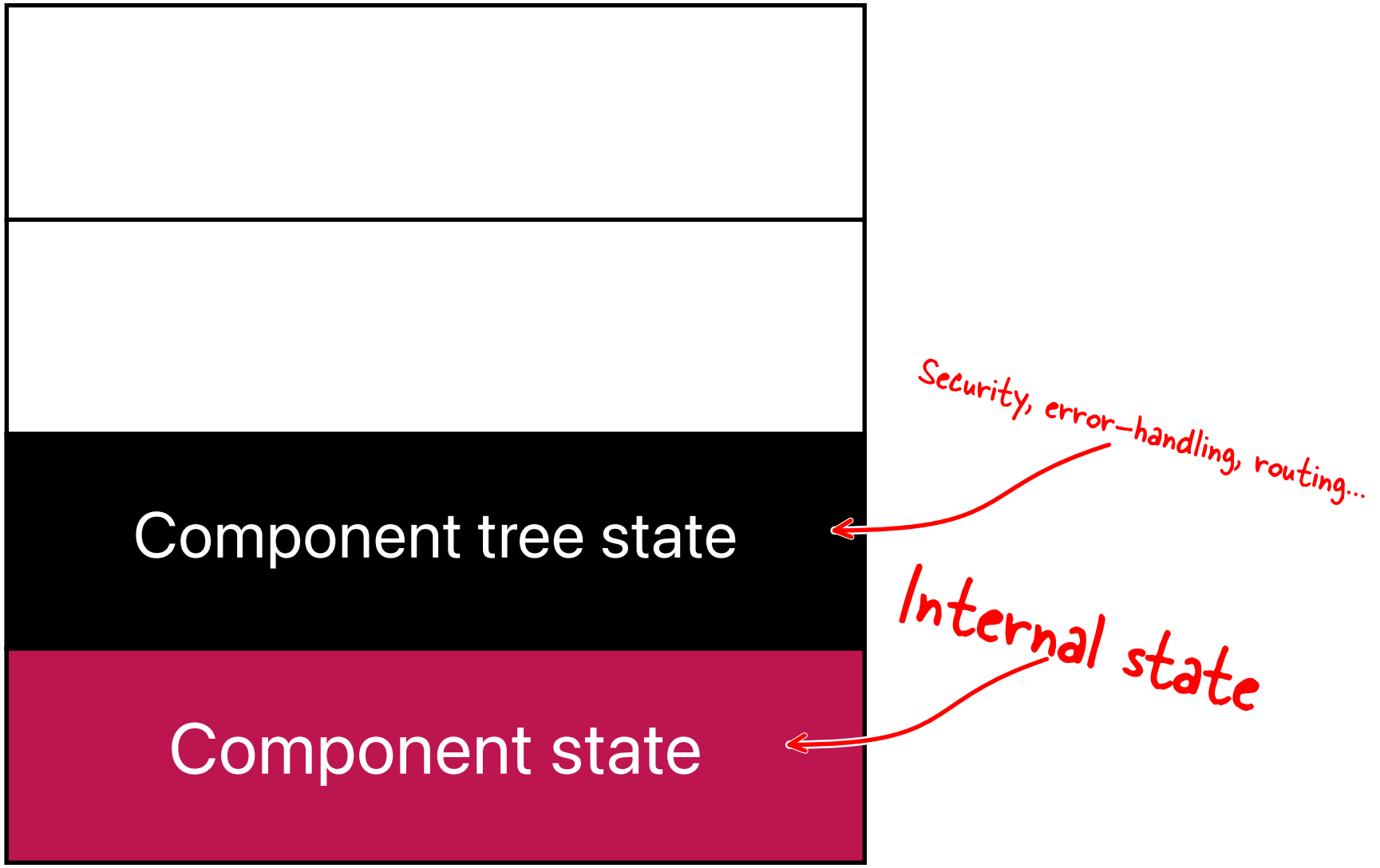
Different levels of state



Different levels of state



Different levels of state



Use context to secure a component tree (recipe 7.5)

Create a context: SecurityContext.js

```
import React from 'react'

export default React.createContext({})
```

Create a provider: SecurityProvider.js

```
import { useRef, useState } from 'react'
import SecurityContext from './SecurityContext'
import LoginForm from './LoginForm'

export default (props) => {
  return (
    <SecurityContext.Provider
      value={{
      }}
    >
      {props.children}
    </SecurityContext.Provider>
  )
}
```

Create a provider: SecurityProvider.js

```
import { useRef, useState } from 'react'
import SecurityContext from './SecurityContext'
import LoginForm from './LoginForm'

export default (props) => {
  const [showLogin, setShowLogin] = useState(false)
  ....
  return (
    <SecurityContext.Provider
      value={{
        login: () => setShowLogin(true)
        ...
      }}
    >
      {props.children}
    </SecurityContext.Provider>
  )
}
```

Create a provider: SecurityProvider.js

```
import { useRef, useState } from 'react'
import SecurityContext from './SecurityContext'
import LoginForm from './LoginForm'

export default (props) => {
  const [showLogin, setShowLogin] = useState(false)
  ....
  return (
    <SecurityContext.Provider
      value={{
        login: () => setShowLogin(true)
        ...
      }}
    >
      {showLogin ? (
        <LoginForm
          onLogin={async (username, password) => {
            // Clever code to do login goes here....
          }}
        />
      ) : null}
      {props.children}
    </SecurityContext.Provider>
  )
}
```

Use the provider

```
import './App.css'
import SecurityProvider from './SecurityProvider'

function App() {
  return (
    <div className="App">
        <MyComponent/>
        <MyOtherComponent/>
    </div>
  )
}

export default App
```

Use the provider

```
import './App.css'
import SecurityProvider from './SecurityProvider'

function App() {
  return (
    <div className="App">
      <SecurityProvider>
        <MyComponent/>
        <MyOtherComponent/>
      </SecurityProvider>
    </div>
  )
}

export default App
```

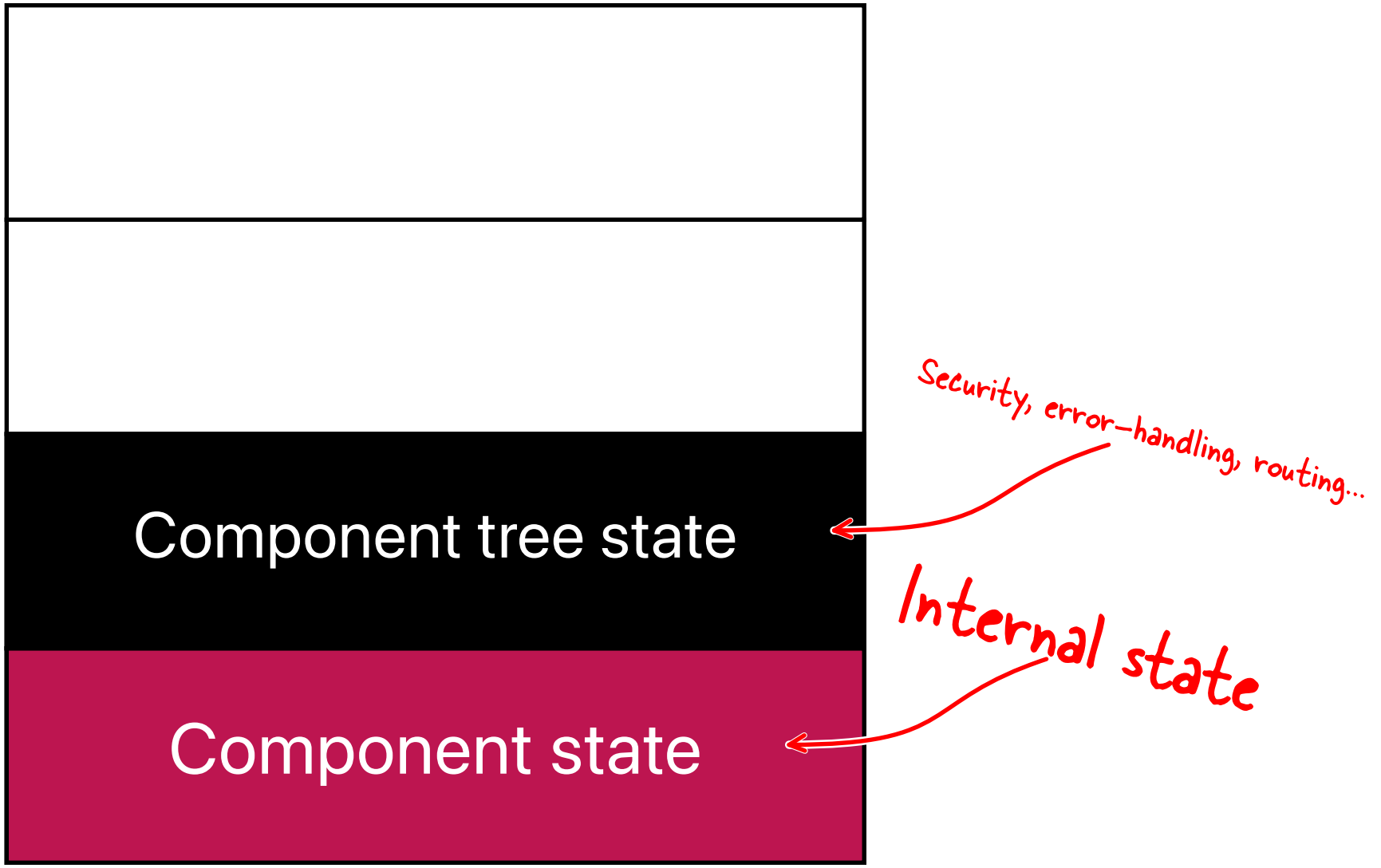
Access security from the context MyComponent.js

```
const security = useContext(SecurityContext)
```

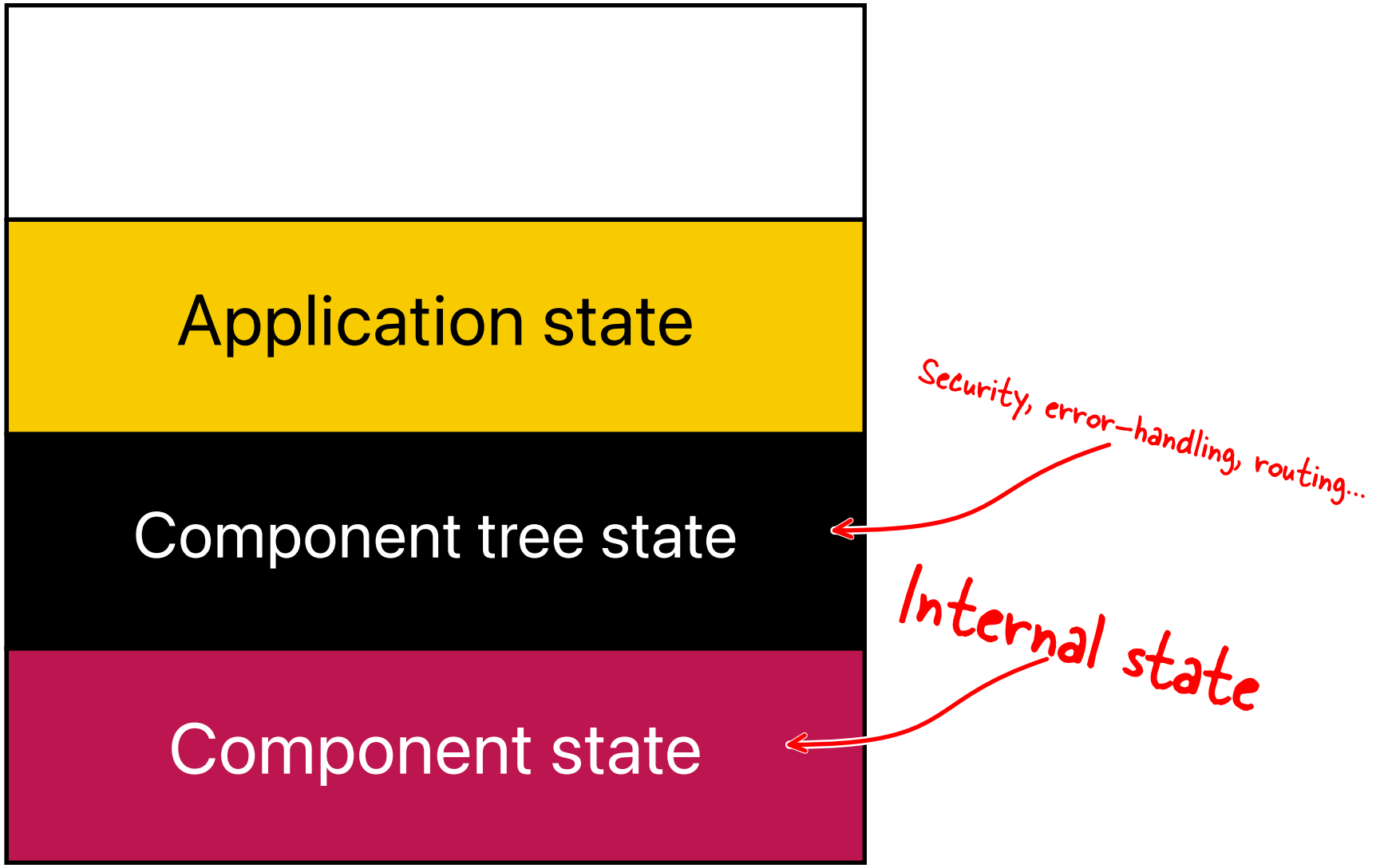
Access security from the context MyComponent.js

```
const security = useContext(SecurityContext)
<button
  onClick={() => security.login())}
>
  Open login form
</button>
```

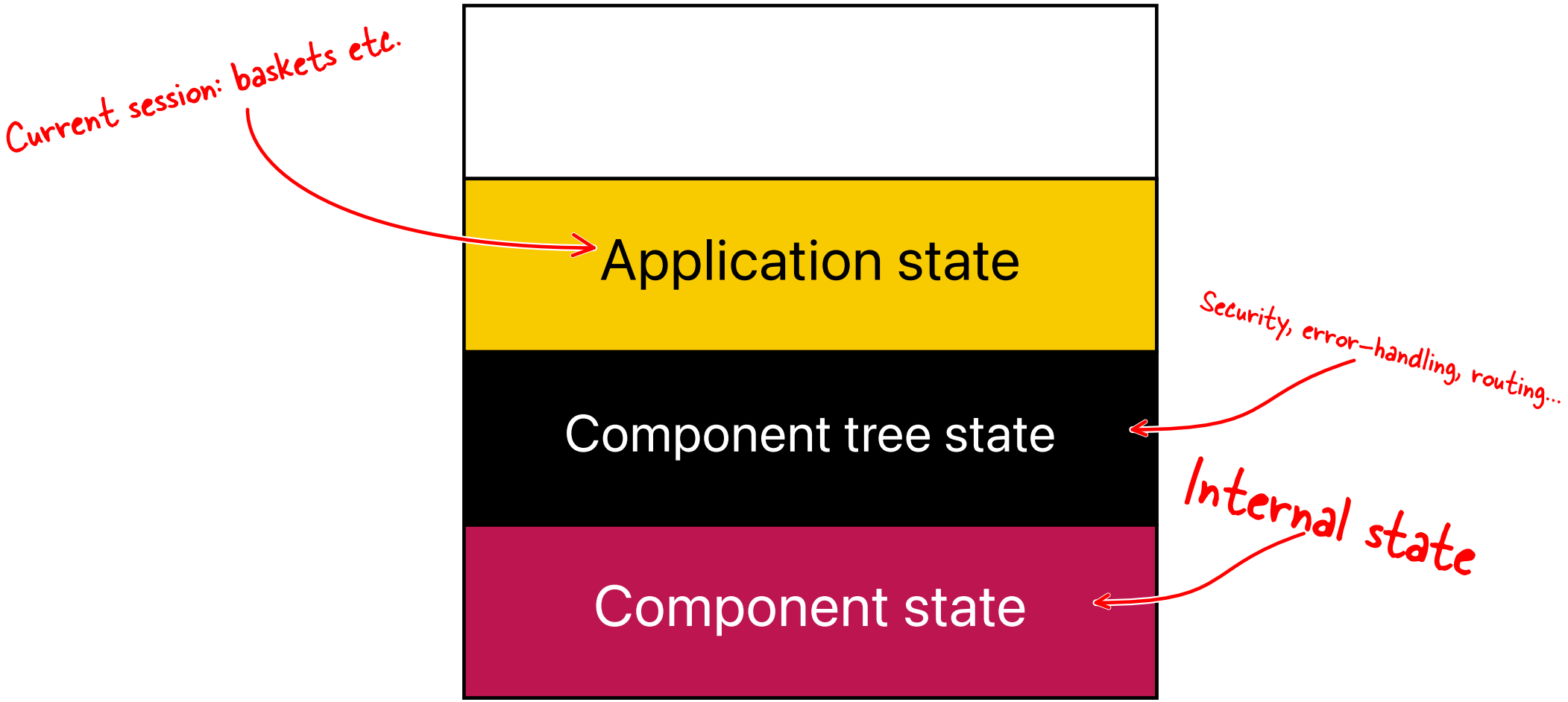
Different levels of state



Different levels of state



Different levels of state



Replace network calls with hooks (recipe 5.1)

Code that talks to a server

Code that talks to a server

- It's usually complex

Code that talks to a server

- It's usually complex
- It's normally asynchronous...

Code that talks to a server

- It's usually complex
- It's normally asynchronous...
- ...so it doesn't respond immediately

Code that talks to a server

- It's usually complex
- It's normally asynchronous...
- ...so it doesn't respond immediately
- It can throw lots of errors

Put horrible async code in a hook: useMessages.js

```
import { useEffect, useState } from 'react'

const useMessages = (forum) => {


}

export default useMessages
```

Put horrible async code in a hook: useMessages.js

```
import { useEffect, useState } from 'react'

const useMessages = (forum) => {
  const [data, setData] = useState([])



}

export default useMessages
```

Put horrible async code in a hook: useMessages.js

```
import { useEffect, useState } from 'react'

const useMessages = (forum) => {
  const [data, setData] = useState([])
  const [loading, setLoading] = useState(false)



}

export default useMessages
```

Put horrible async code in a hook: useMessages.js

```
import { useEffect, useState } from 'react'

const useMessages = (forum) => {
  const [data, setData] = useState([])
  const [loading, setLoading] = useState(false)
  const [error, setError] = useState()



}

export default useMessages
```

Put horrible async code in a hook: useMessages.js

```
import { useEffect, useState } from 'react'

const useMessages = (forum) => {
  const [data, setData] = useState([])
  const [loading, setLoading] = useState(false)
  const [error, setError] = useState()


  return { data, loading, error }
}

export default useMessages
```

Put horrible async code in a hook: useMessages.js

```
import { useEffect, useState } from 'react'

const useMessages = (forum) => {
  const [data, setData] = useState([])
  const [loading, setLoading] = useState(false)
  const [error, setError] = useState()

  useEffect(() => {
      // Clever code to contact the server, and set
      // 'loading', 'data' or 'error' goes here
  }, [forum])

  return { data, loading, error }
}

export default useMessages
```

Using the useMessages hook

```
const {
    data: messages,
    loading: messagesLoading,
    error: messagesError,
  } = useMessages(forum)
```

Using the useMessages hook

```
const {
    data: messages,
    loading: messagesLoading,
    error: messagesError,
  } = useMessages(forum)

if (error) {
  return <p>Error! {error}</p>
}
```

Using the useMessages hook

```
const {
    data: messages,
    loading: messagesLoading,
    error: messagesError,
  } = useMessages(forum)

if (error) {
  return <p>Error! {error}</p>
}

if (loading) {
  return <p>Loading...</p>
}
```

Using the useMessages hook

```
const {
    data: messages,
    loading: messagesLoading,
    error: messagesError,
  } = useMessages(forum)

if (error) {
  return <p>Error! {error}</p>
}

if (loading) {
  return <p>Loading...</p>
}

return // the HTML for displaying messages goes here
```

Synchronize data with the server (recipe 5.2)

Synchronize data with the server (recipe 5.2)

Synchronize data with the server (recipe 5.2)

- You might have code that **reads** data from the server

Synchronize data with the server (recipe 5.2)

- You might have code that **reads** data from the server
- And code that **updates** data from the server

Synchronize data with the server (recipe 5.2)

- You might have code that **reads** data from the server
- And code that **updates** data from the server
- But how do you get your code to re-read data after you change it?

Synchronize hooks with state counters

```
const {
    data: messages,
    loading: messagesLoading,
    error: messagesError,
  } = useMessages(forum)

const postMessage = (msg) => {
  // Very complicated code to post a message
}
```

Synchronize hooks with state counters

```
const [stateVersion, setStateVersion] = useState(0)
const {
    data: messages,
    loading: messagesLoading,
    error: messagesError,
  } = useMessages(forum, stateVersion)

const create = (msg) => {
  // Very complicated code to post a message
  ....
  setStateVersion((v) => v + 1)
}
```

Managing complex application state (recipe 3.6)

Managing complex application state (recipe 3.6)

Managing complex application state (recipe 3.6)

- Application state can get very complex

Managing complex application state (recipe 3.6)

- Application state can get very complex
- We fixed complex state in components with reducers

Managing complex application state (recipe 3.6)

- Application state can get very complex
- We fixed complex state in components with reducers
- We can do the same at the application level...

Managing complex application state (recipe 3.6)

- Application state can get very complex
- We fixed complex state in components with reducers
- We can do the same at the application level...
- ...with Redux

Reducer to manage a shopping basket

```
const reducer = (state = {}, action = {}) => {
  switch (action.type) {
    case 'buy': {
    }
    case 'clearBasket': {
    }
    default:
      return { ...state }
  }
}

export default reducer
```

Reducer to manage a shopping basket

```
const reducer = (state = {}, action = {}) => {
  switch (action.type) {
    case 'buy': {
      const basket = state.basket ? [...state.basket] : []
      // Code to add to basket goes here...
      return {
        ...state,
        basket,
      }
    }
    case 'clearBasket': {
    }
    default:
      return { ...state }
  }
}

export default reducer
```

Reducer to manage a shopping basket

```
const reducer = (state = {}, action = {}) => {
  switch (action.type) {
    case 'buy': {
      const basket = state.basket ? [...state.basket] : []
      // Code to add to basket goes here...
      return {
        ...state,
        basket,
      }
    }
    case 'clearBasket': {
      return {
        ...state,
        basket: [],
      }
    }
    default:
      return { ...state }
  }
}

export default reducer
```

Create a store using the basket reducer App.js

```
import { Provider } from 'react-redux'
import { createStore } from 'redux'

import reducer from './reducer'

const store = createStore(reducer)

function App() {
  return (
    <div className="App">
      <Provider store={store}>
          <MyFirstComponent/>
          <MySecondComponent/>
          <MyThirdComponent/>
      </Provider>
    </div>
  )
}

export default App
```

Use the store from MyFirstComponent.js

```
const MyFirstComponent = () => {
  const dispatch = useDispatch()

  return (
    <div className="Boots">
      <h1>Boots</h1>
       <button
         onClick={() =>
           dispatch({ type: 'buy', payload: {name: 'Boots'} })
         }
       >
         Add to basket
       </button>
    </div>
  )
}
```

Using Redux libraries

Using Redux libraries

- Can use middleware to sync Redux store with a server (recipe 3.6)

Using Redux libraries

- Can use middleware to sync Redux store with a server (recipe 3.6)
- Can use `redux-persist` to save the store in LocalStorage

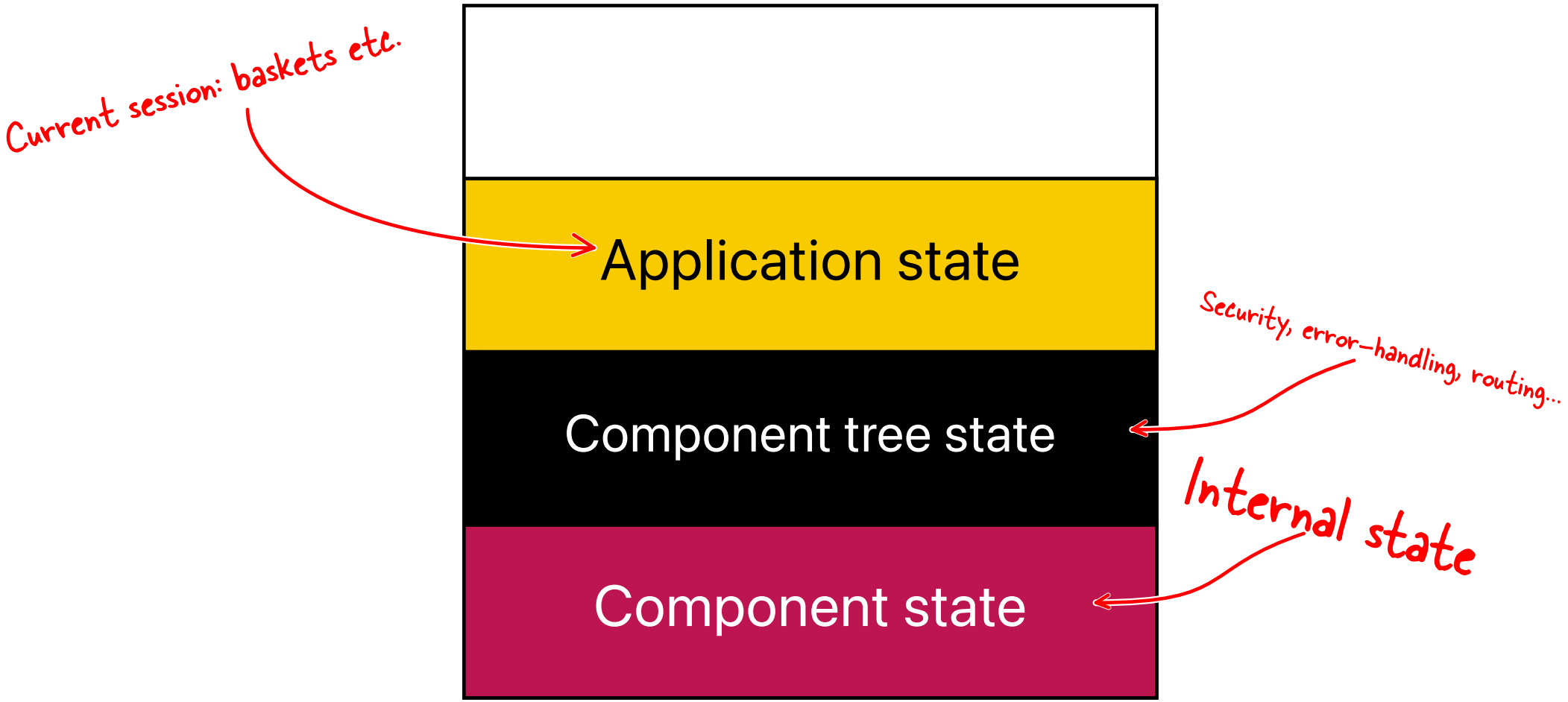
Using Redux libraries

- Can use middleware to sync Redux store with a server (recipe 3.6)
- Can use `redux-persist` to save the store in LocalStorage
- ...means you can refresh the page and the data is safe (recipe 3.7)

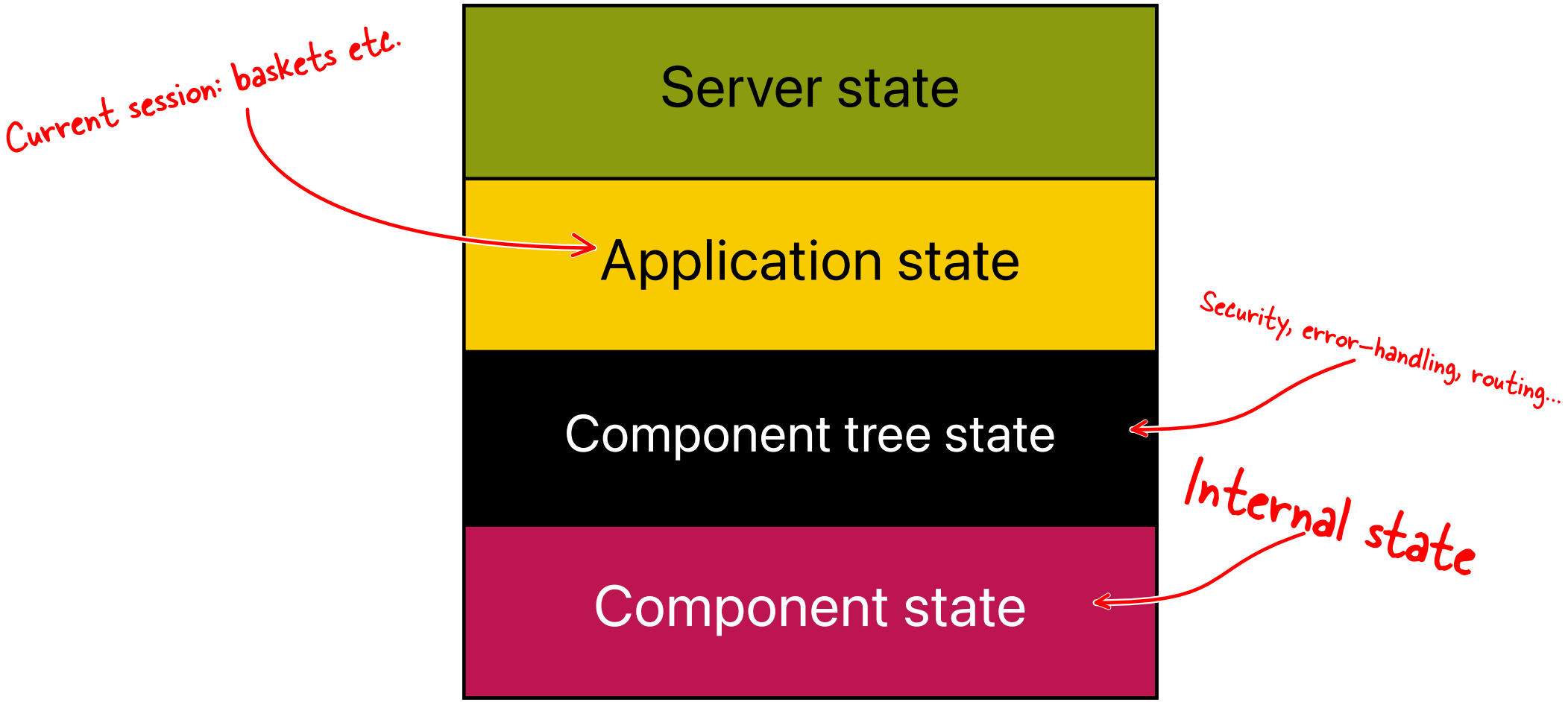
Using Redux libraries

- Can use middleware to sync Redux store with a server (recipe 3.6)
- Can use `redux-persist` to save the store in LocalStorage
- ...means you can refresh the page and the data is safe (recipe 3.7)
- Can use with libraries like `reselect` to derive states (recipe 3.8)

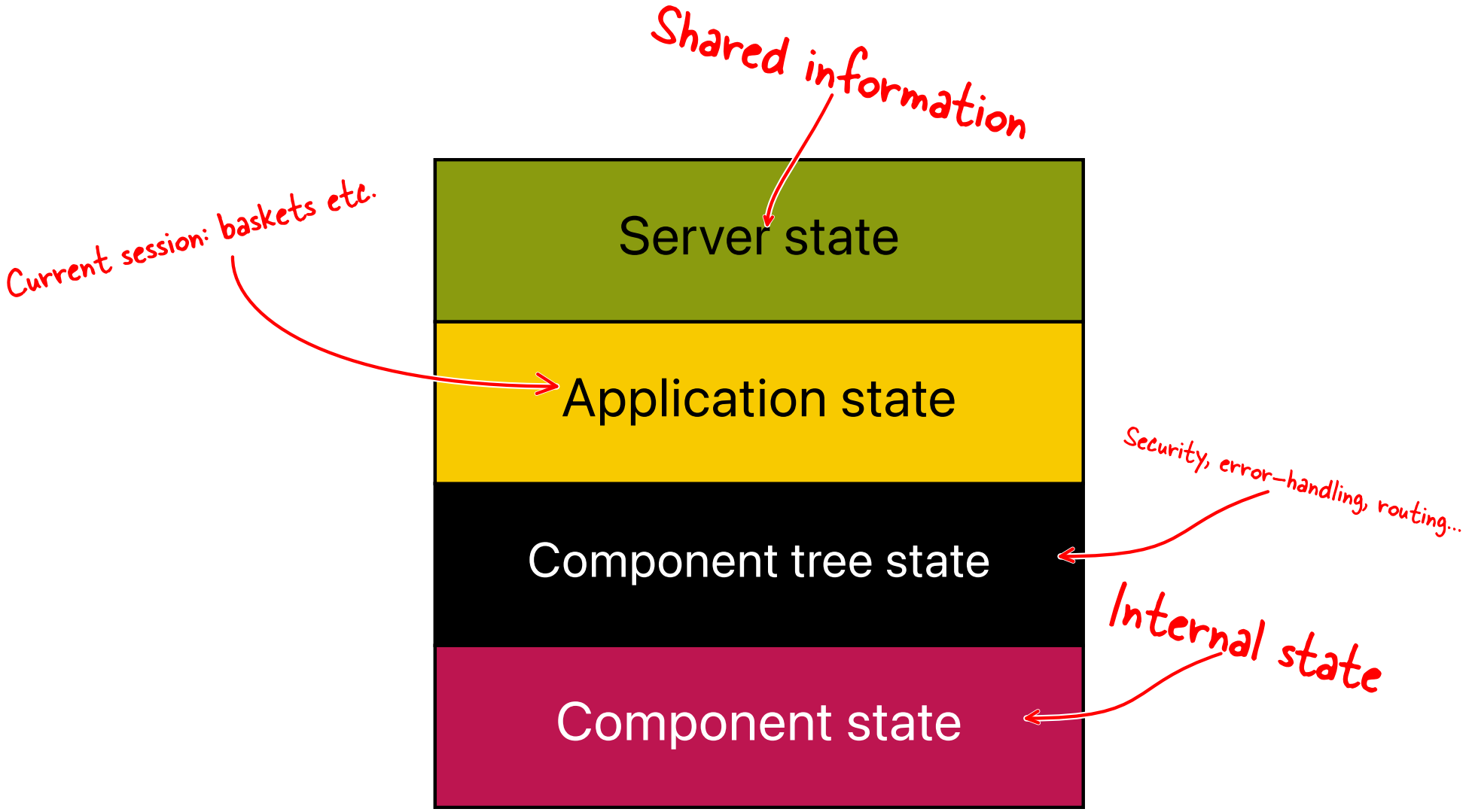
Different levels of state



Different levels of state



Different levels of state



Reading data when offline? (recipe 11.3)

Reading data when offline? (recipe 11.3)

Reading data when offline? (recipe 11.3)

- Many users are on mobile devices

Reading data when offline? (recipe 11.3)

- Many users are on mobile devices
- Mobile devices have poor network connections

Reading data when offline? (recipe 11.3)

- Many users are on mobile devices
- Mobile devices have poor network connections
- Can you cache data locally so it can work offline?

Service workers

Service workers

- JavaScript that runs in the background (not the page)

Service workers

- JavaScript that runs in the background (not the page)
- Can intercept all networks connections

Service workers

- JavaScript that runs in the background (not the page)
- Can intercept all networks connections
- Can use cache storage for images, fonts, network responses

Service workers (part 2)

Service workers (part 2)

- Not normally used when in development mode

Service workers (part 2)

- Not normally used when in development mode
- Only available with HTTPS or with localhost

Service workers (part 2)

- Not normally used when in development mode
- Only available with HTTPS or with localhost
- Might need to unregister when developing

Register a service worker index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorkerRegistration from './serviceWorkerRegistration';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

Register a service worker index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorkerRegistration from './serviceWorkerRegistration';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);

serviceWorkerRegistration.register();
```

The service worker: service-worker.js

```
import { registerRoute } from 'workbox-routing';
import { StaleWhileRevalidate } from 'workbox-strategies';

// All the other service-worker code...
```

The service worker: service-worker.js

```
import { registerRoute } from 'workbox-routing';
import { StaleWhileRevalidate } from 'workbox-strategies';

// All the other service-worker code...

registerRoute(
    ({url}) => url.origin === 'https://fonts.googleapis.com',
    new StaleWhileRevalidate({
        cacheName: 'stylesheets',
    })
);
```

Sending data when offline? (recipe 11.6)

Sending data when offline? (recipe 11.6)

Sending data when offline? (recipe 11.6)

- When people are offline they might want to do more than read

Sending data when offline? (recipe 11.6)

- When people are offline they might want to do more than read
- They might want to send changes to the server

Sending data when offline? (recipe 11.6)

- When people are offline they might want to do more than read
- They might want to send changes to the server
- But how can that work when they have no connection?

Typical code to send data

```
const sendData = () => {
        const options = {
            method: 'POST',
            body: JSON.stringify({timeIs: new Date()}),
            headers: {
                'Content-Type': 'application/json'
            }
        };
        fetch('/endpoint', options)
    };
```

Use background-sync in the server-worker

```
import { registerRoute } from 'workbox-routing';
import {NetworkOnly, StaleWhileRevalidate} from 'workbox-strategies';
import {BackgroundSyncPlugin} from "workbox-background-sync";

// All the other service-worker code...
```

Use background-sync in the server-worker

```
import { registerRoute } from 'workbox-routing';
import {NetworkOnly, StaleWhileRevalidate} from 'workbox-strategies';
import {BackgroundSyncPlugin} from "workbox-background-sync";

// All the other service-worker code...

registerRoute(
    /\/endpoint/,
    new NetworkOnly({
        plugins: [new BackgroundSyncPlugin(
            'endPointQueue1', {
            maxRetentionTime: 24 * 60
        })]
    }),
    'POST'
);
```

Get the code!

Get the code!

- All of the code from today:

Get the code!

- All of the code from today:
- https://tinyurl.com/rcookbook1

Get the code!

- All of the code from today:
- https://tinyurl.com/rcookbook1
- All of the code from the book:

Get the code!

- All of the code from today:
- https://tinyurl.com/rcookbook1
- All of the code from the book:
- https://tinyurl.com/rcookbook2


O'REILLY®