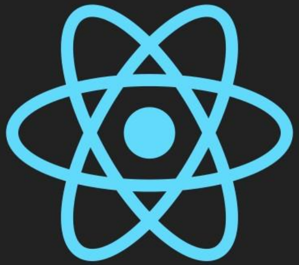
# Building Forms



# LEARNING OBJECTIVES

- Learn to build form elements that are controlled by React
- Learn to fetch form data directly from the DOM
- Implement debouncing & other techniques to efficiently handle multiple form elements

# Building Forms

## Controlled Form Components

# FORM ELEMENTS

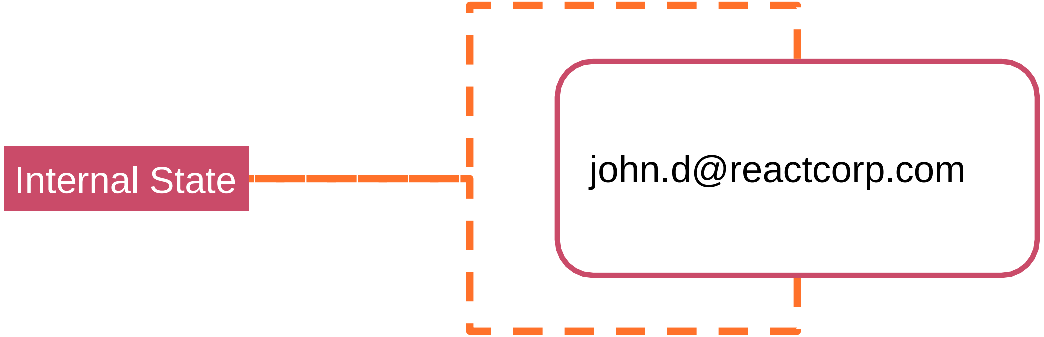
## WHAT ARE FORM ELEMENTS?

Form elements are critical components of any application that accepts user data.

Example: Input Elements

# FORM ELEMENTS

Form elements comprises of its own internal state



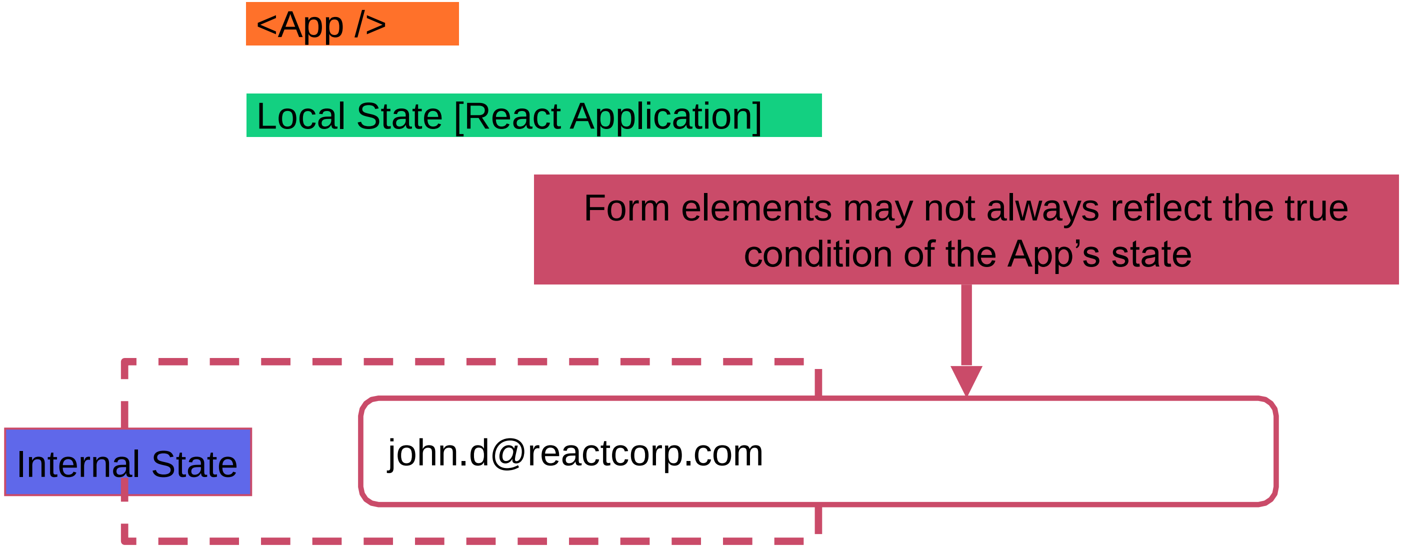
Internal State

john.d@reactcorp.com

# FORM ELEMENTS

When used in an application, Form Elements are maintained by Components in React.

Form Elements may not always reflect the true conditions of the App's state.



<App />

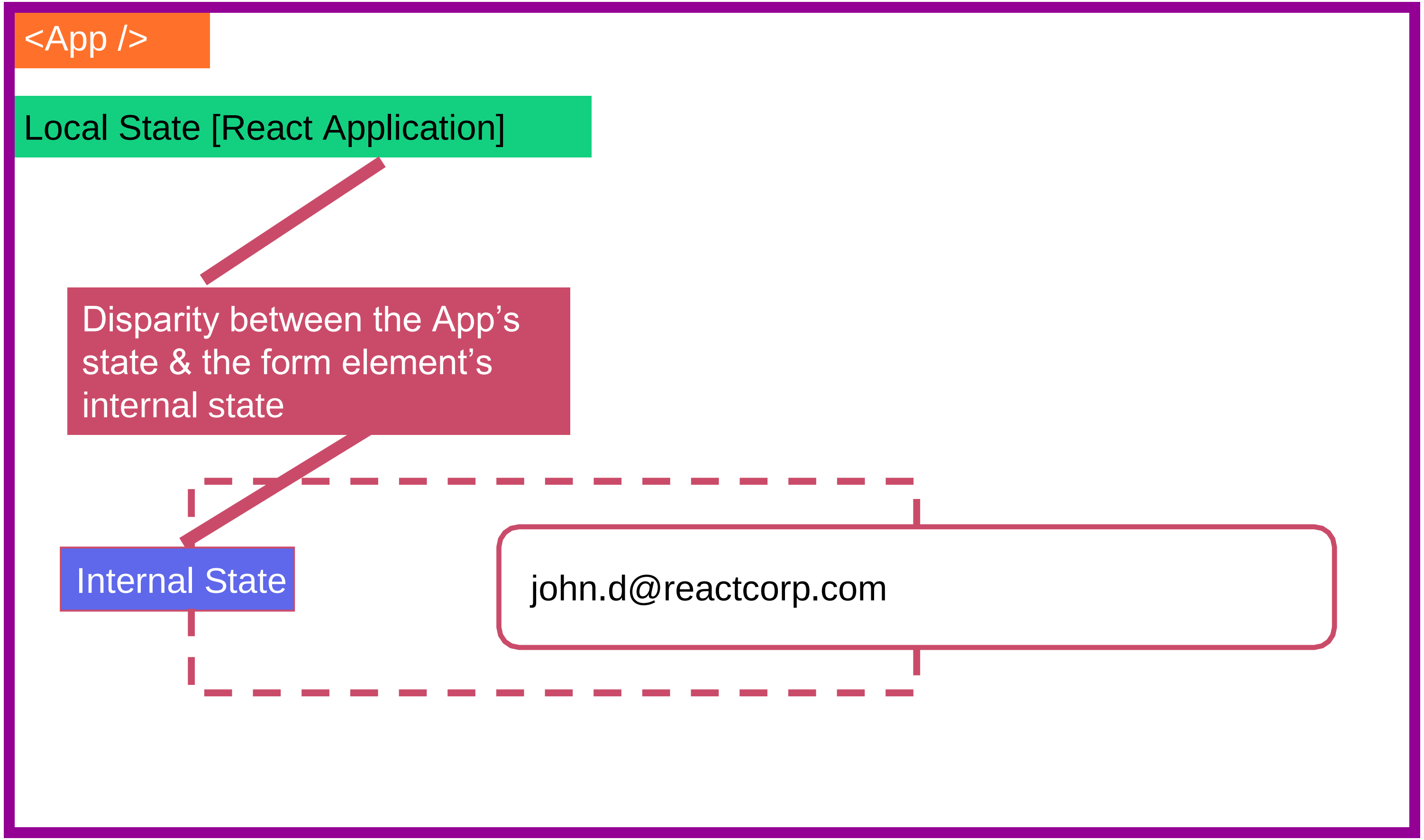
Local State [React Application]

Form elements may not always reflect the true condition of the App’s state

Internal State

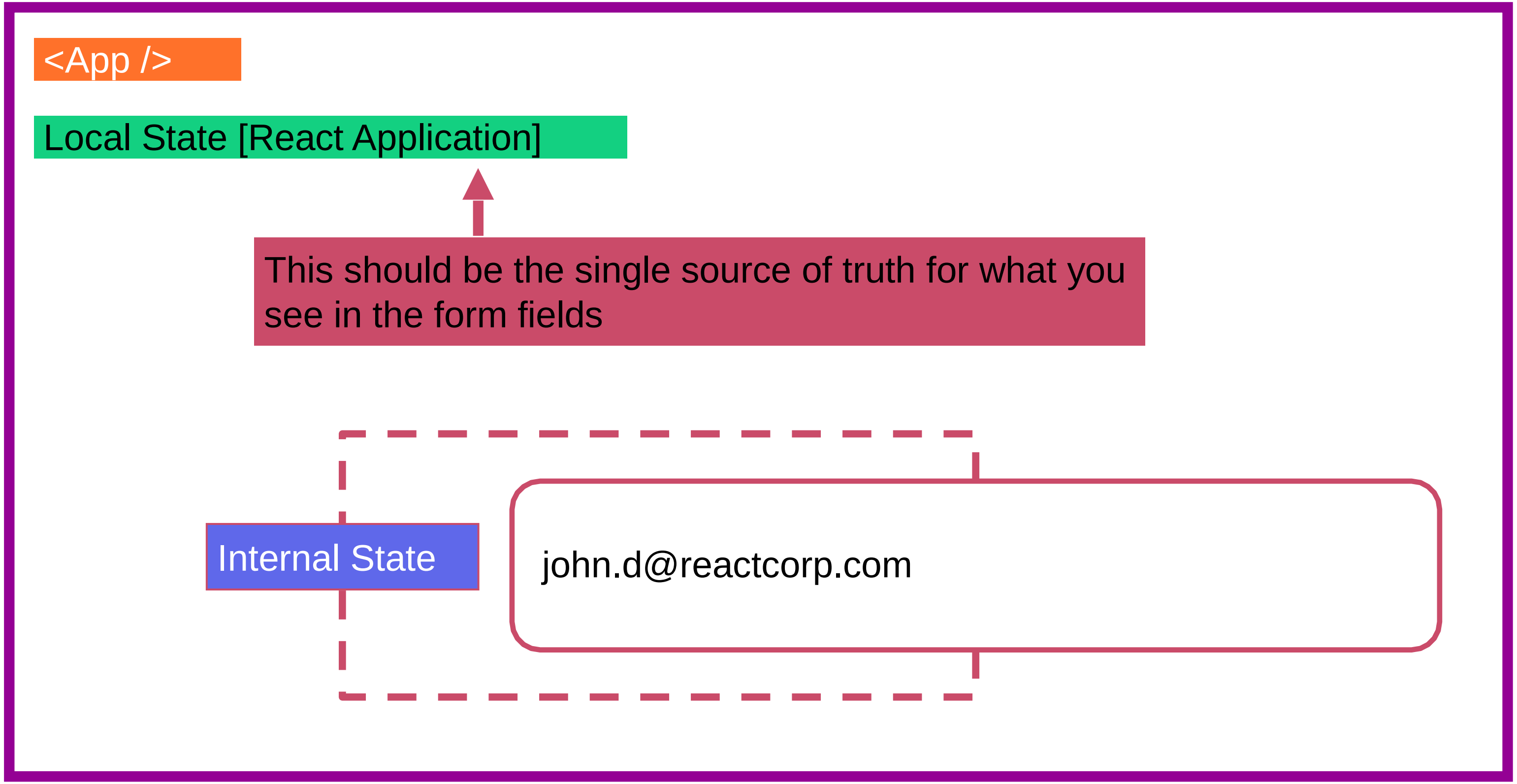
john.d@reactcorp.com

# FORM ELEMENTS



# FORM ELEMENTS

React’s State = Single Source of Truth



<App />

Local State [React Application]

This should be the single source of truth for what you see in the form fields

Internal State

john.d@reactcorp.com

# HOW DO WE CONVERT USER INPUT TO UPPER-CASE?

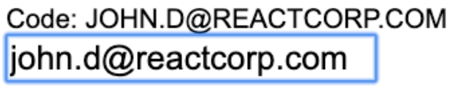
- By Using inline functions

# CONVERSION OF USER INPUT TO UPPER-CASE?

```
import React, {useState} from "react";
import {render} from "react-dom";


const App = () => {
  const [code, setCode] = useState("");
  return (
    <>
      <div className="output">Code: {code}</div>
      <input
        type="text"
        onChange={e => setCode(e.target.value.toUpperCase())}
      />
    </>
  );
};


render(<App />, document.getElementById("root"));
```



```
Code: JOHN.D@REACTCORP.COM
john.d@reactcorp.com
```

# How does the input element display the exact condition of the state?

- Using controlled components

Code: JOHN.D@REACTCORP.COM

JOHN.D@REACTCORP.COM

The form field displays exactly what is being stored in the state

**Controlled Components** ✏ React state becomes the single source of truth for form elements!

# Two Way Binding

- Using controlled components

Data Model (State)

View

- Common and popular technique in frameworks like Angular, Vue

- Easy to implement in React

# Hands-On

# FORM ELEMENTS

Setting the value property renders a form field as read-only in React

```
<input type="text" value="john.d@reactcorp.com" />
```

Static String

Read-only

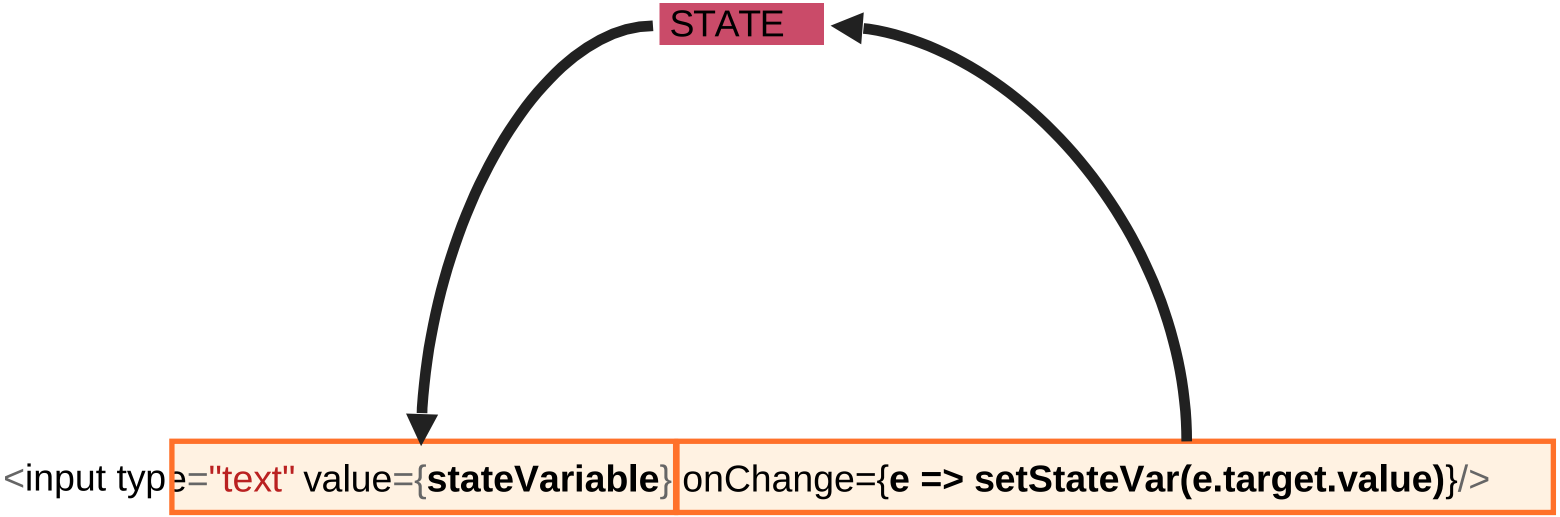
Exception: These will not be read-only

```
<input type="text" value={null} />
```

Or

```
<input type="text" value={undefined} />
```

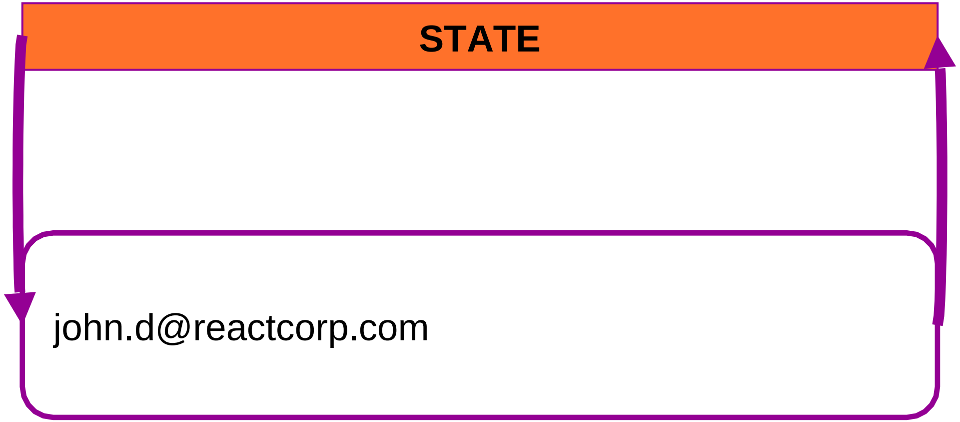
# FORM ELEMENTS



STATE

```
<input type="text" value={stateVariable} onChange={e => setStateVar(e.target.value)}/>
```

# CONTROLLED FORM ELEMENTS



**STATE**

john.d@reactcorp.com

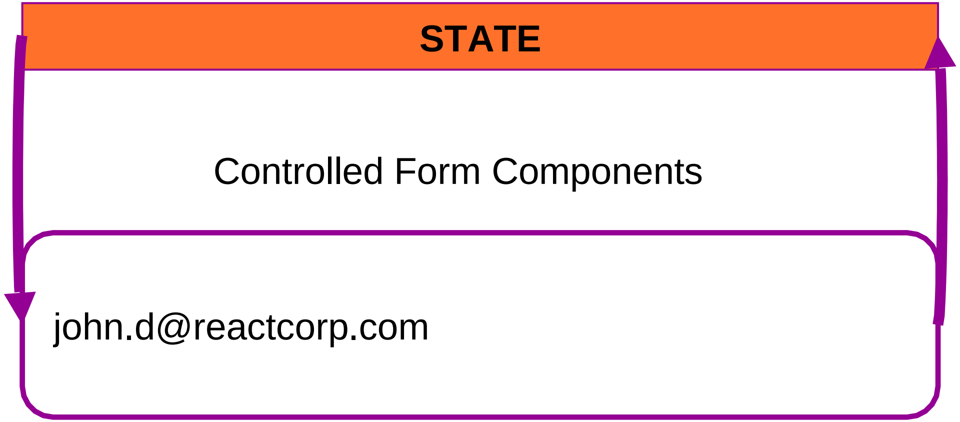
- Provides an accurate representation of data in the app’s state
- Form elements are controlled by React

# Building Forms

## Uncontrolled Form Components

# CONTROLLED FORM ELEMENTS

- Data resides in the state and form elements push data into the state as well as pull from it in real time.



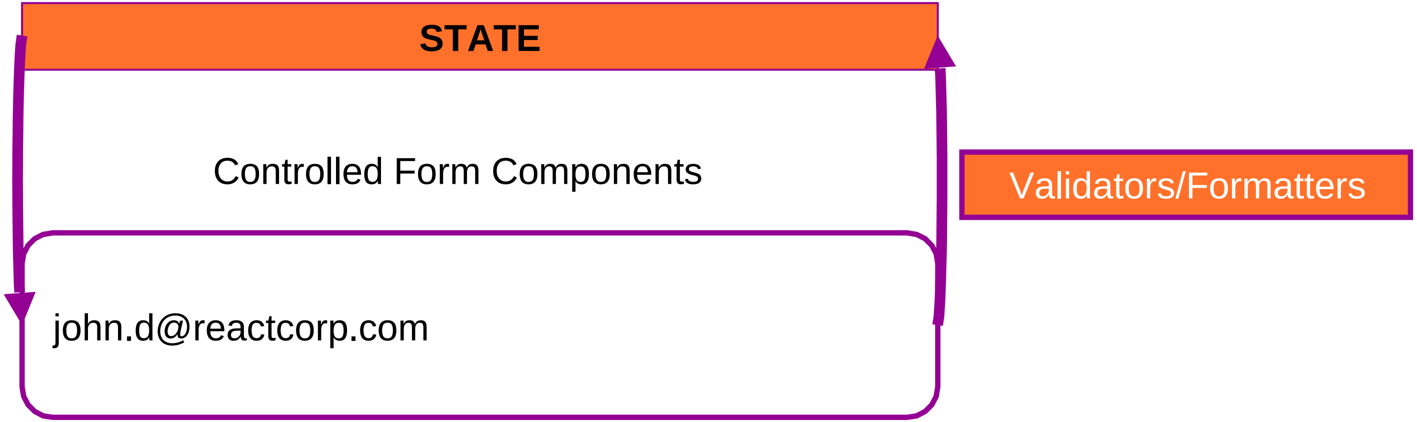
**STATE**

Controlled Form Components

john.d@reactcorp.com

# CONTROLLED FORM ELEMENTS

**Benefit:** Its ability to implement in-place effects like validation and formatting, which responds as the user types in



**STATE**

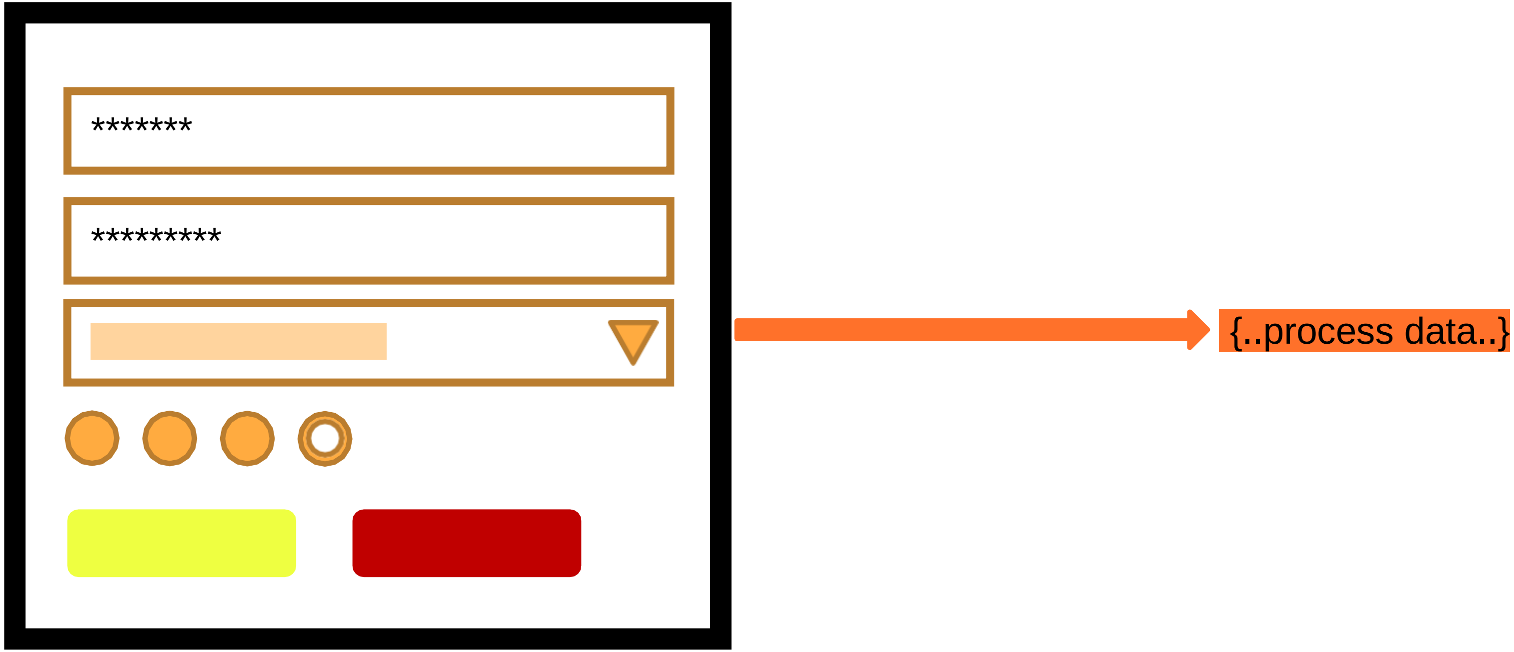
Controlled Form Components

Validators/Formatters

john.d@reactcorp.com

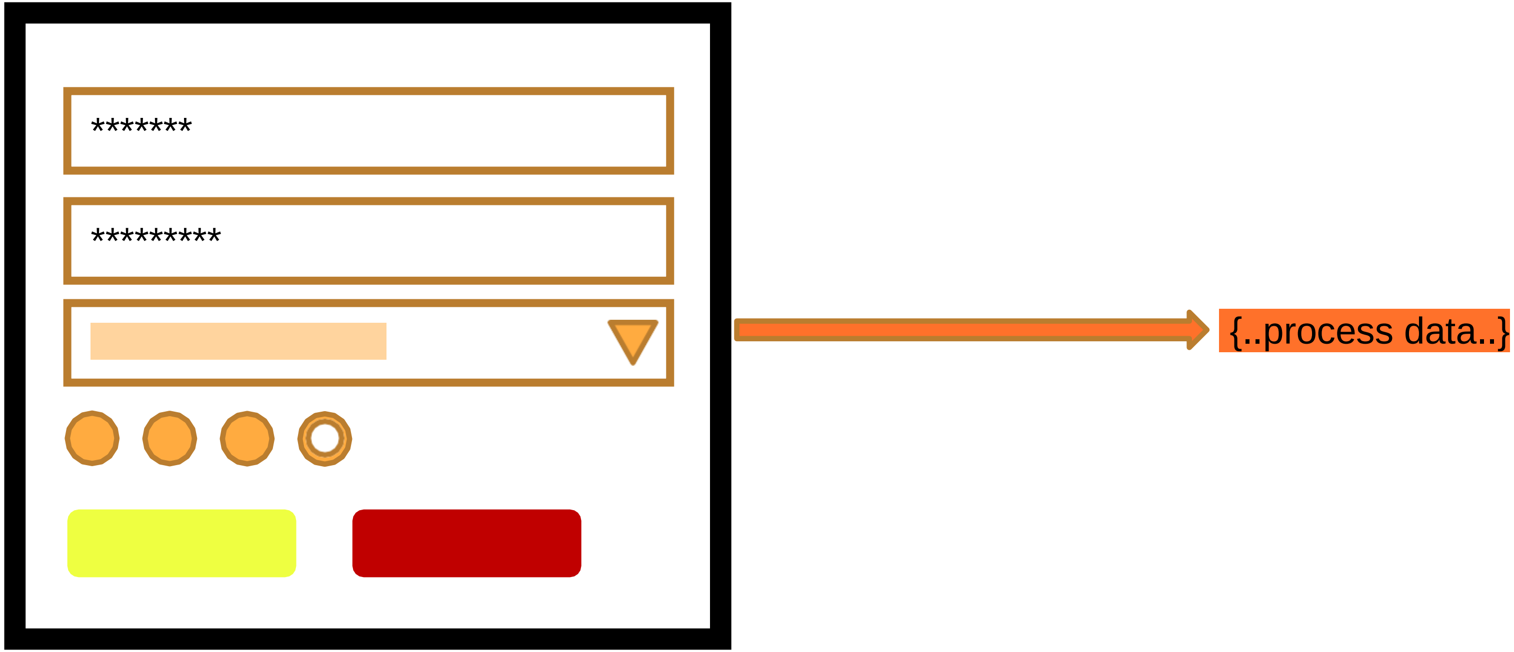
# Situation where an Uncontrolled Component can be used:

- When data is accessed only once, like submitting a form



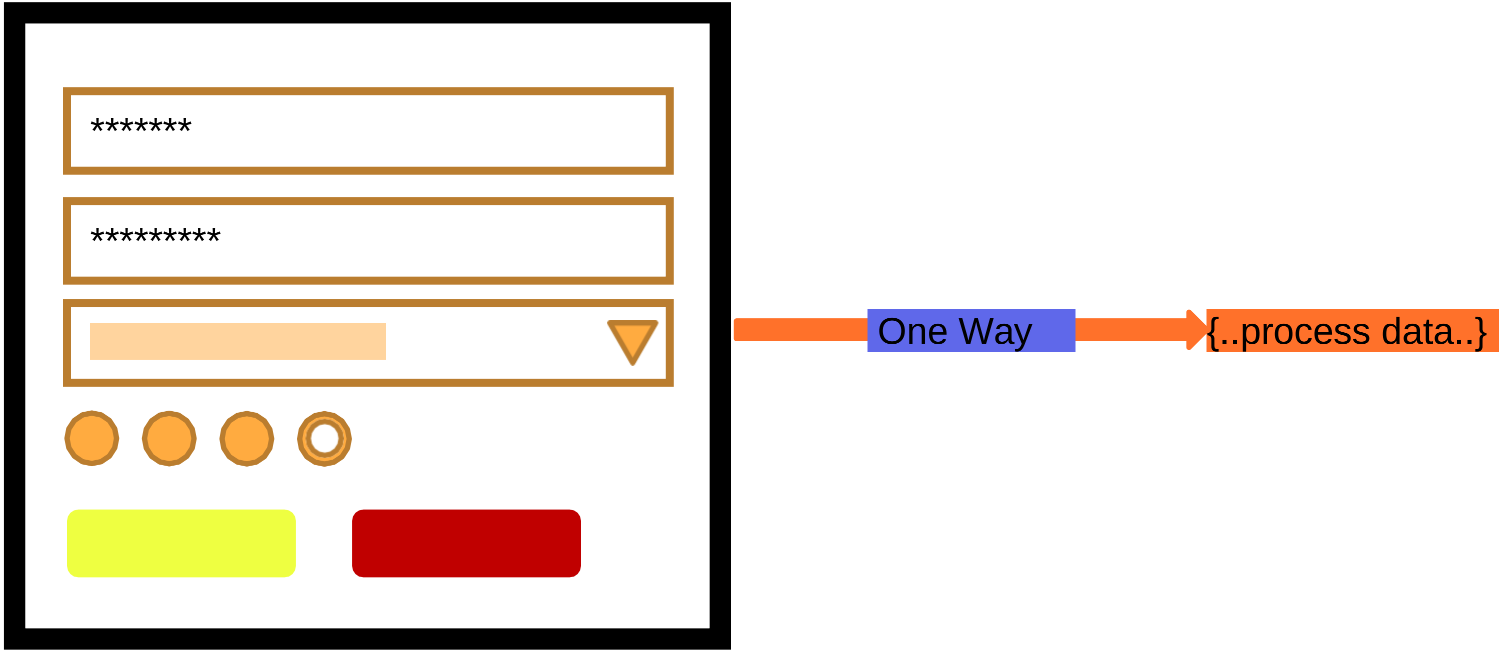
# Situation where an Uncontrolled Component can be used:

- Not controlled by React, but is a regular form element that is rendered and controlled by DOM



# Situation where an Uncontrolled Component can be used:

- One-way relationship where React access the DOM version of Form Element to access and retrieve data as per need
- Eminent for simple use cases where values are not set to the form elements



# Uncontrolled Form Elements: Hands-On

# REFS IN UNCONTROLLED FORM ELEMENTS

- Used to reach out into the DOM and access a node directly

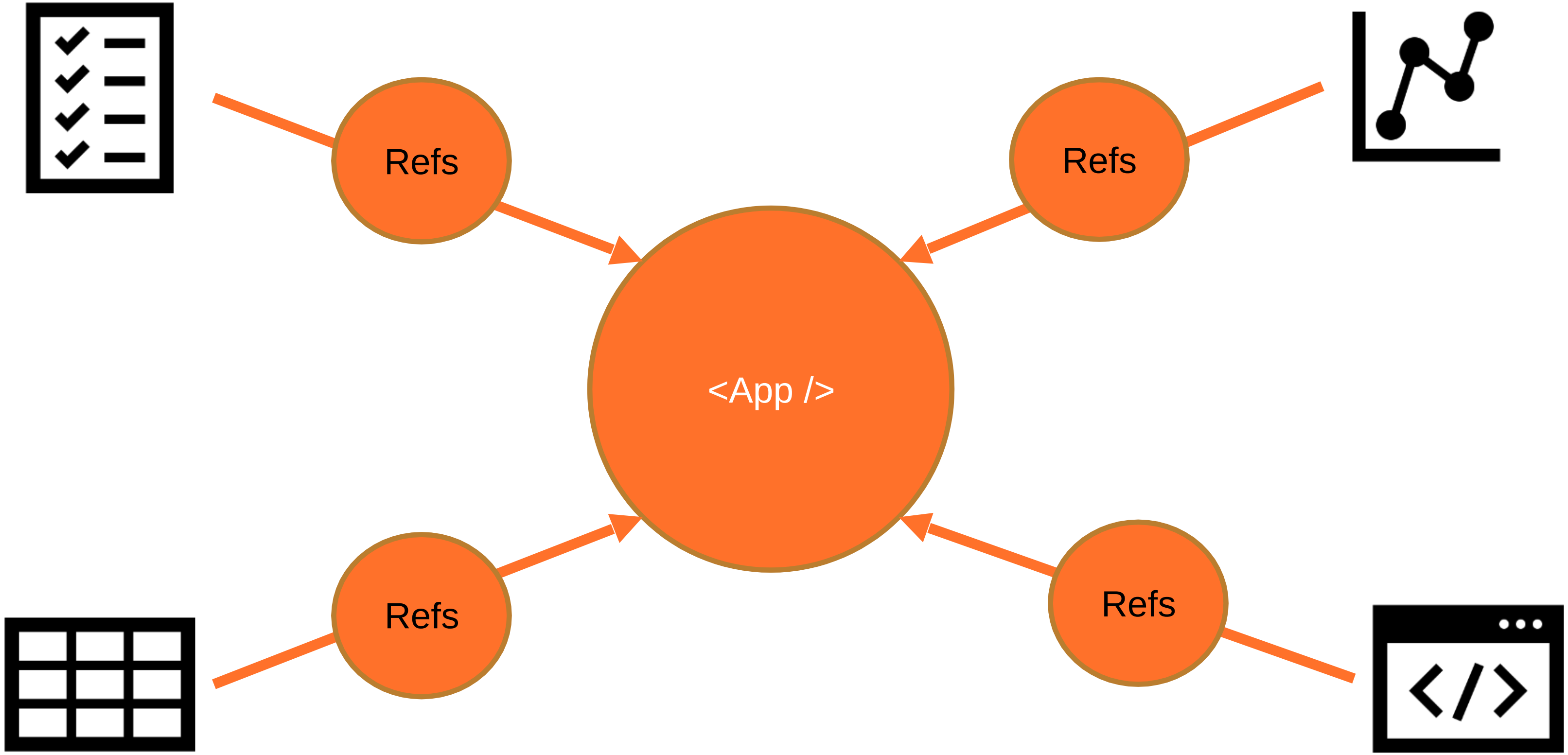
```
class MessageBox extends Component {
  state = {
    name: "",
    nature: "",
    query: ""
  };
  nameRef = createRef();
  natureRef = createRef();
  queryRef = createRef();
  submitForm = () => {
    this.setState({
      name: sentenceCase(this.nameRef.current.value),
      nature: this.natureRef.current.value,
      query: this.queryRef.current.value
    });
  };
  render() {…}
}
```

Instantiating refs

Accessing data from refs

# REFS IN UNCONTROLLED FORM ELEMENTS

- Used to interface React with non-react JavaScript libraries and utilities



Refs

Refs

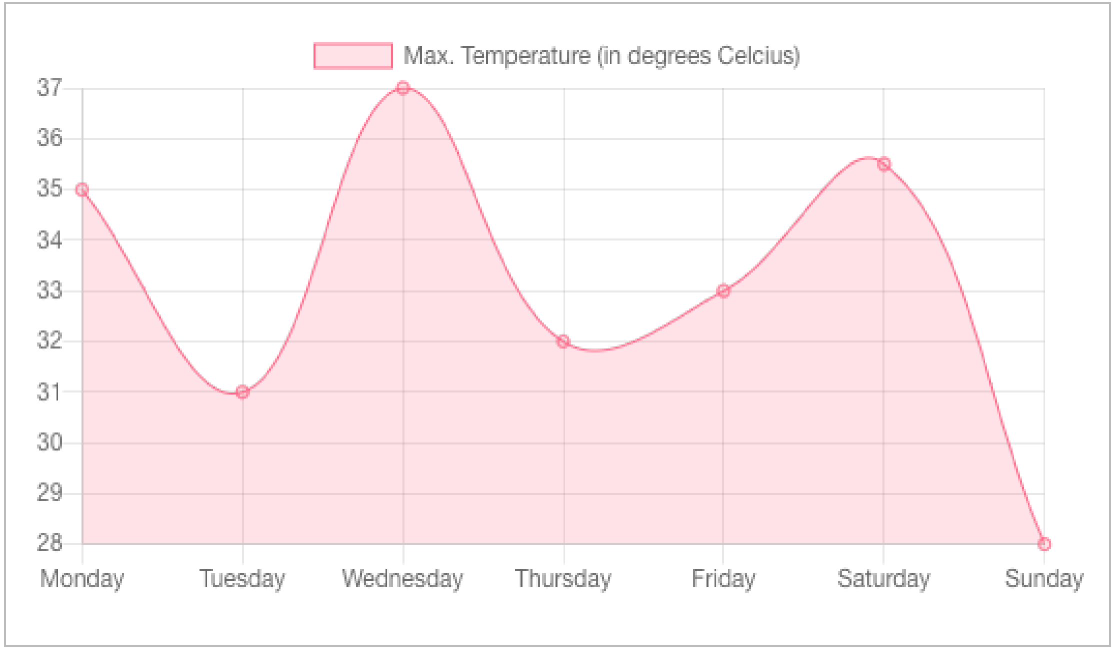
<App />

Refs

Refs

# Hands-On

# REFS IN UNCONTROLLED FORM ELEMENTS



Extensive usage of Refs should be avoided for performance reasons

# Building Forms

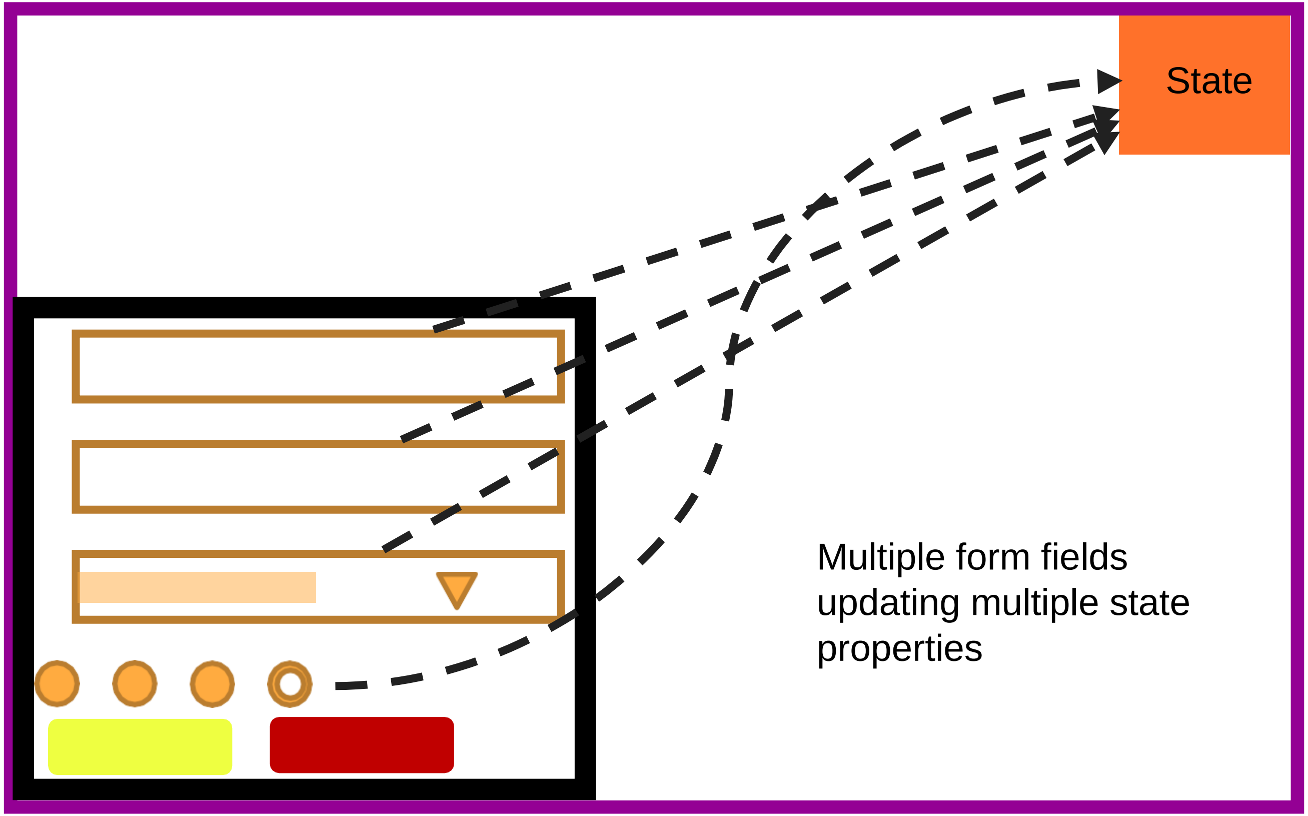
## Handling Inputs Efficiently

# HANDLING INPUTS EFFICIENTLY

- Techniques that promote efficiency when handling form components

**Performance is the key!**

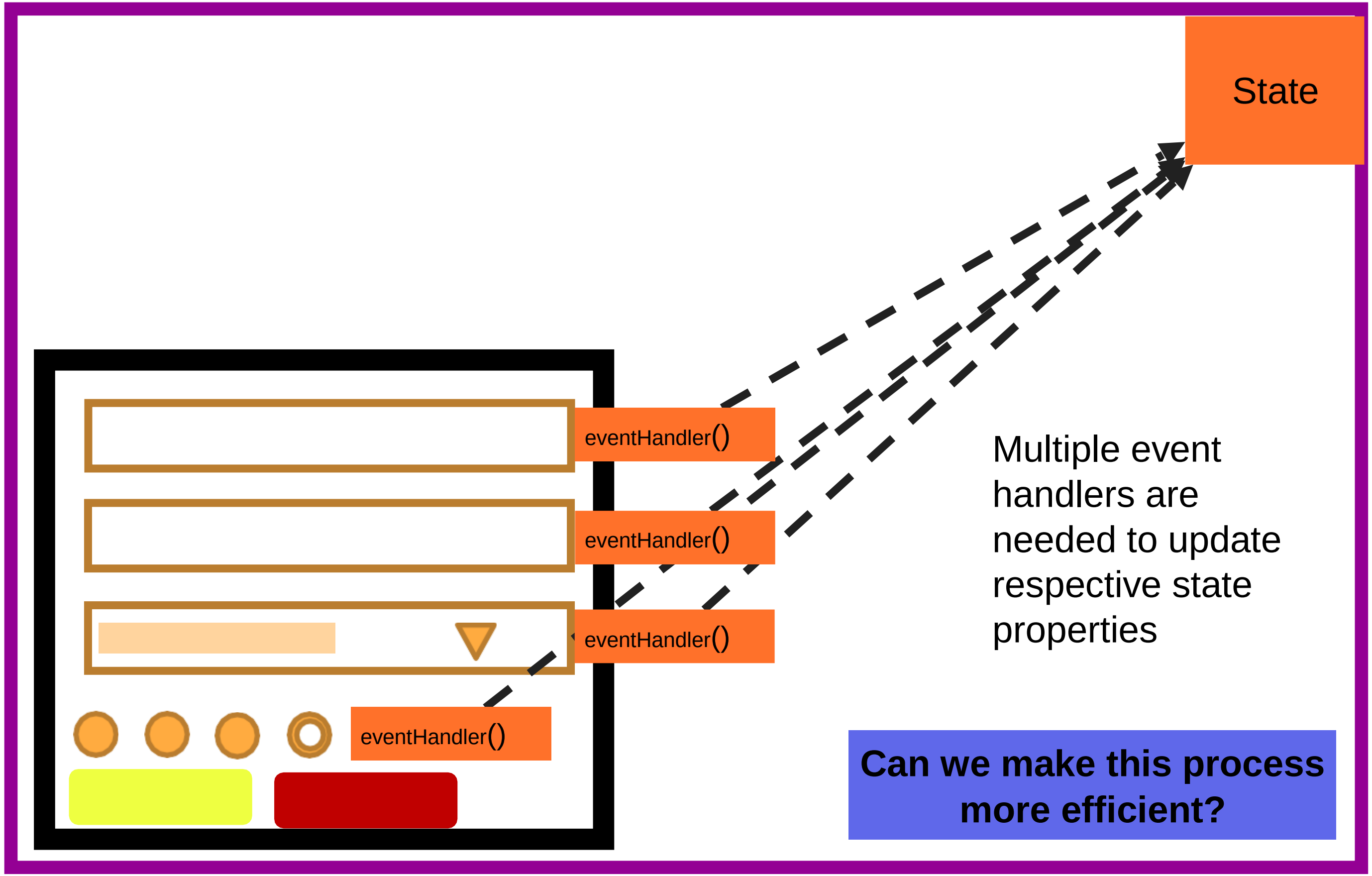
# How to Handle Multiple Input Elements



Multiple form fields
updating multiple state
properties

# How to Handle Multiple Input Elements

- By using multiple event handlers



Multiple event handlers are needed to update respective state properties

**Can we make this process more efficient?**

# Multiple Event Handlers: Hands-On

# How to Handle Multiple Input Elements

```
formHandler = ({target}) => {
    const {name, type, value} = target;
    this.setState({
      [name]:
        type === "checkbox"
          ? {...this.state[name], [value]: !this.state[name]
[value]}
          : value
    });
  };
```

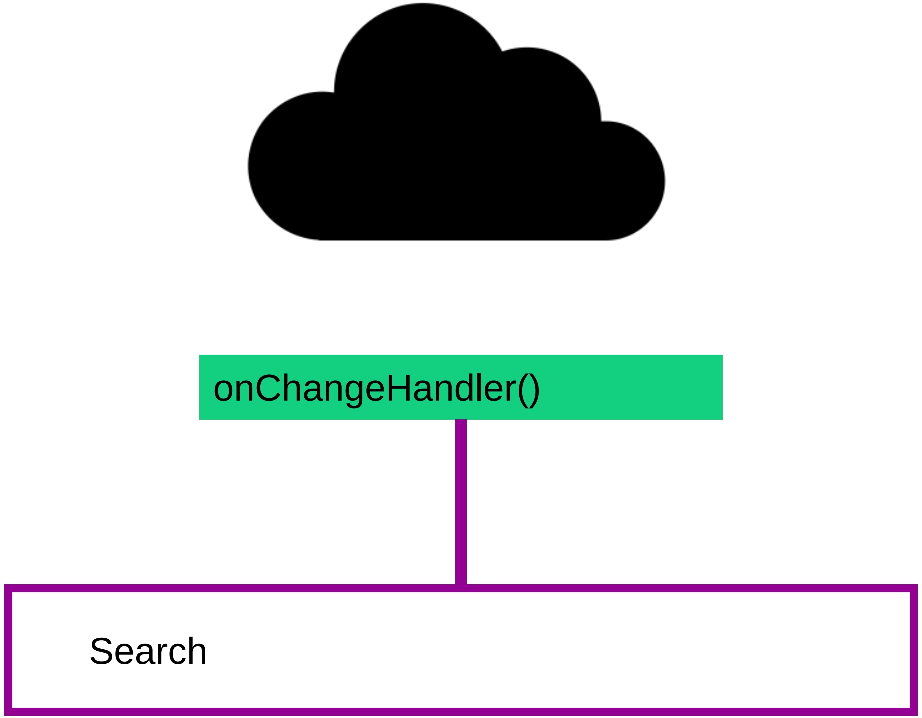
By writing a single & central event handler function:

- Optimize a React component

- Streamline the process of handling multiple input fields

# How to Handle Multiple Input Elements

A Search Component that queries a resource as typed

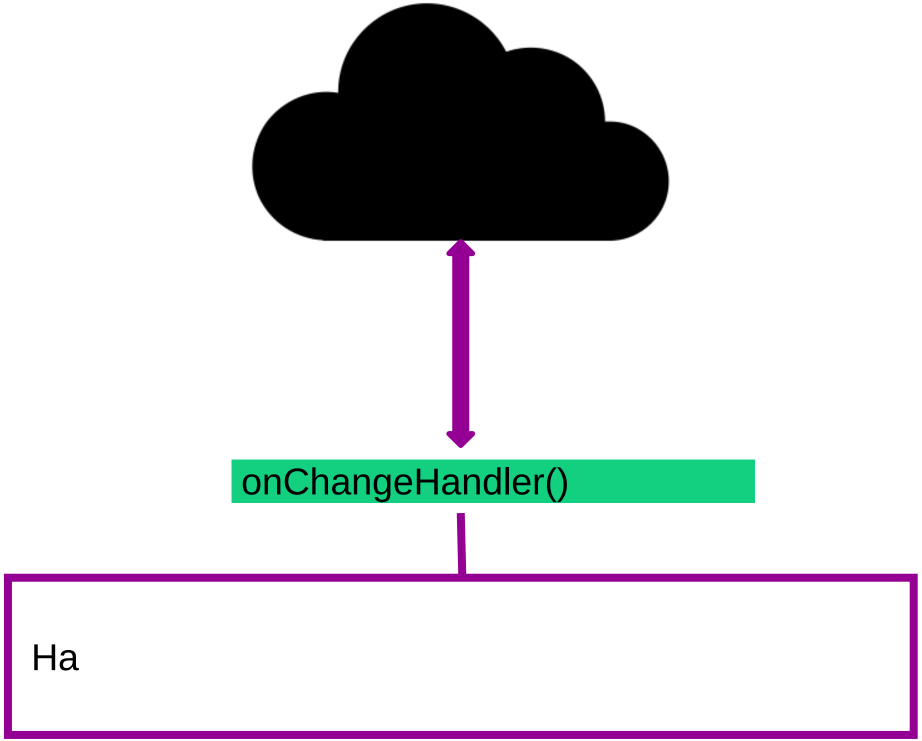


onChangeHandler()

Search

# How to Handle Multiple Input Elements

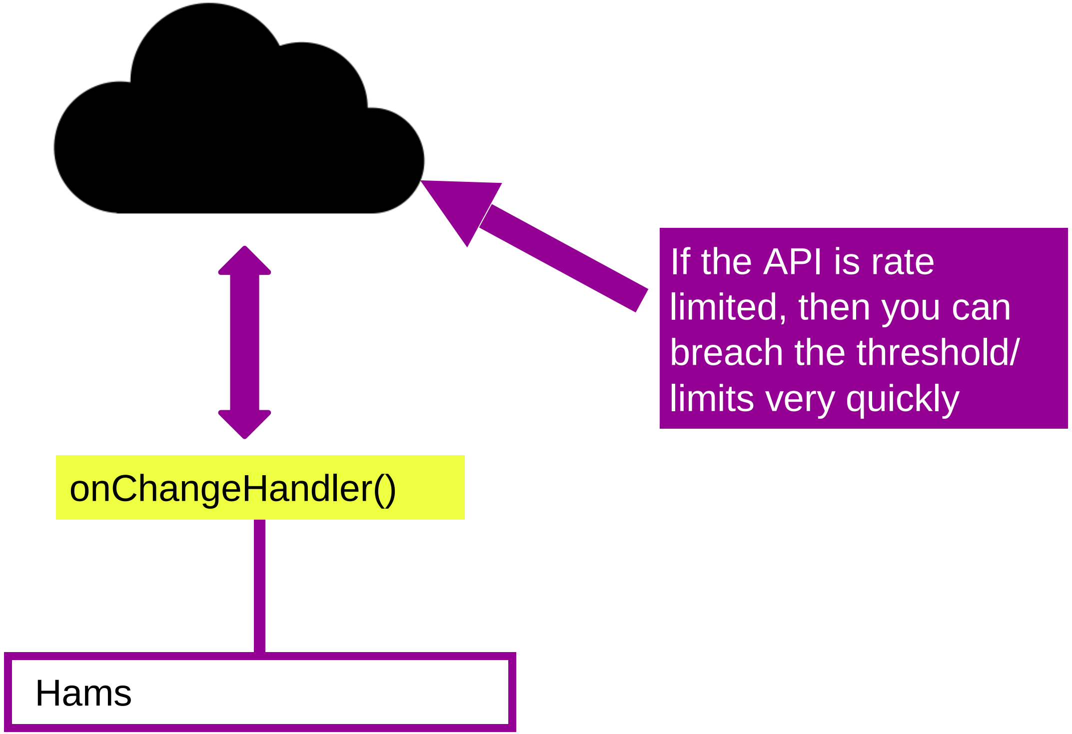
Every single character & change in content triggers an API call as the onChange



onChangeHandler()

Ha

# How to Handle Multiple Input Elements



If the API is rate limited, then you can breach the threshold/ limits very quickly

onChangeHandler()

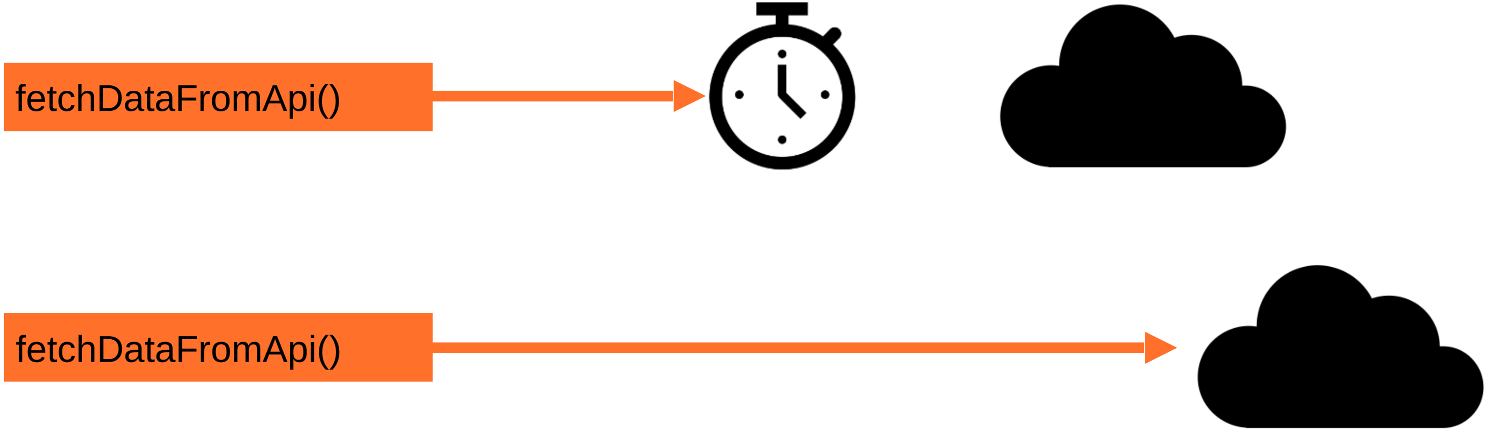
Hams

**Debouncing to the rescue!**

# What is Debouncing?

It is the process of delaying a function.

Wait for the user to stop typing, countdown to a set duration, then run the function!



fetchDataFromApi()

fetchDataFromApi()

Reduces the number of requests!

# How does Debouncing work?

- By initializing a timer every time an event is received

**queryApi()**

# How does Debouncing work?

- Delay a function by 400ms

400ms

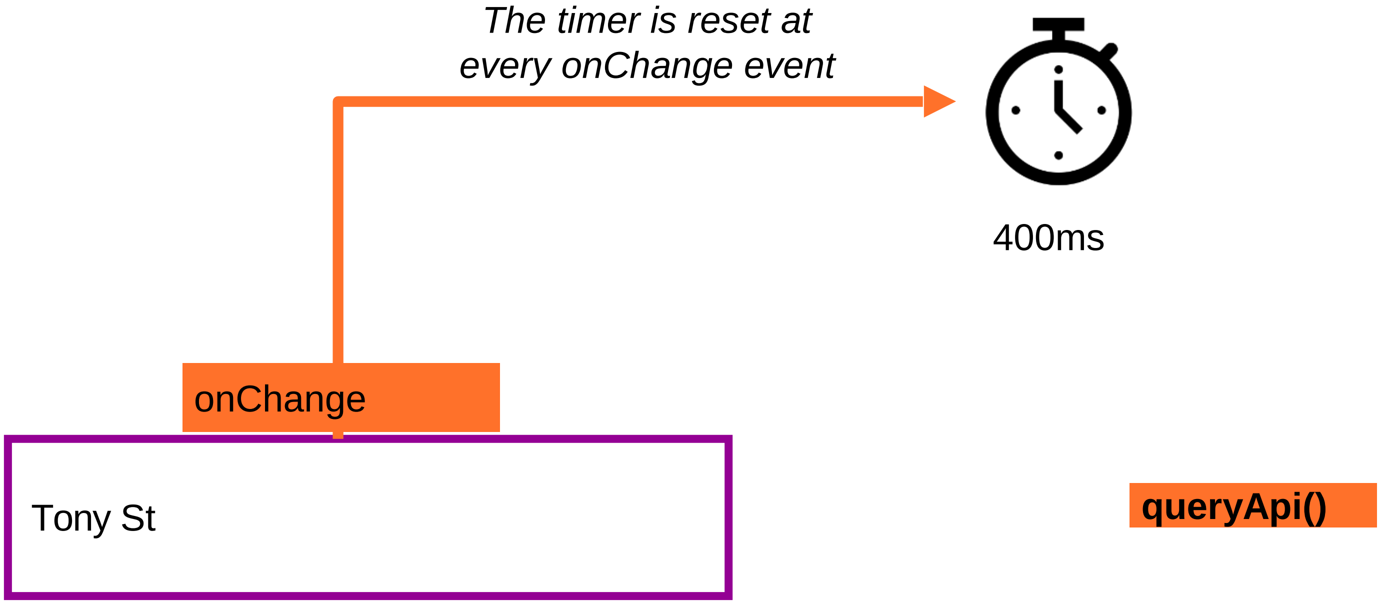
onChange

Tony

**queryApi()**

# How does Debouncing work?

- The timer is reset and set to count to 400ms at every onChange event



*The timer is reset at every onChange event*

400ms

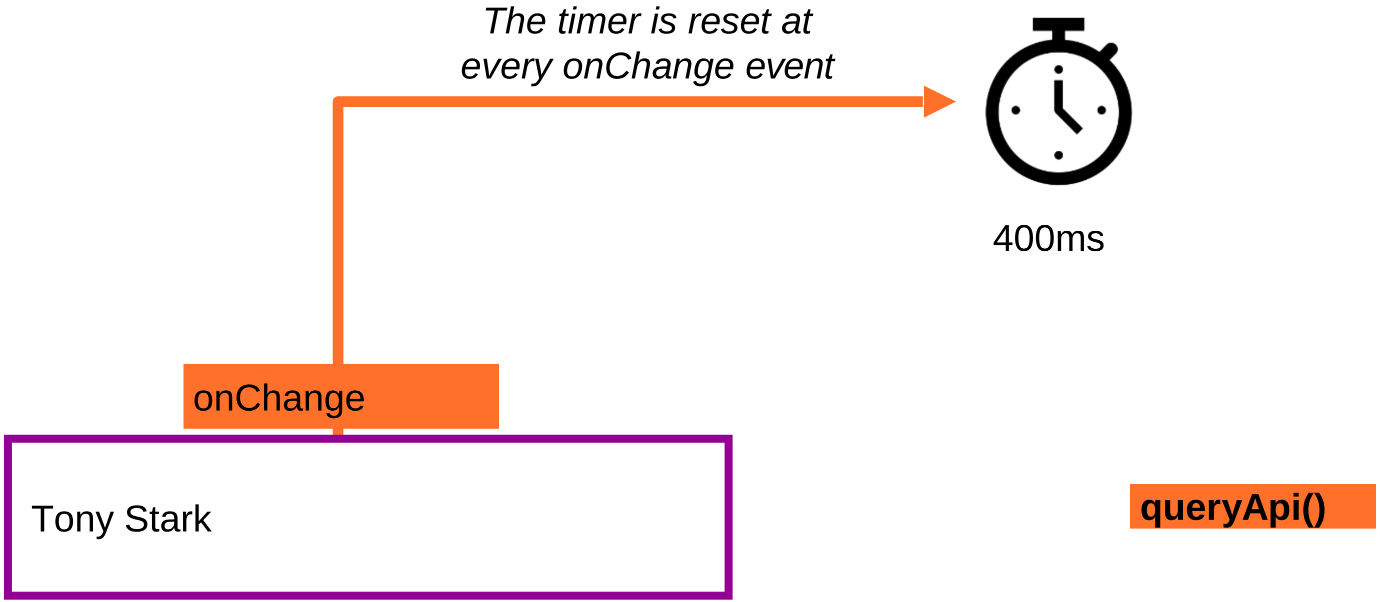
onChange

Tony St

**queryApi()**

# How does Debouncing work?

- Until the user stops typing



*The timer is reset at every onChange event*

400ms

onChange

Tony Stark

**queryApi()**

# How does Debouncing work?

- The timer reaches its mark and executes the API call

onChange

Tony Stark

**queryApi()**

Debouncing drastically reduces the number of API requests by delaying the execution

# Debouncing Example: Hands-On

# Debouncing:

- Debouncing a function cuts down the number of invocations, hence reducing network requests, database queries, etc.
- Debouncing can also be implemented using **setTimeout()**

```
searchBooks = debounce(keyword => {
  console.log(`Searching for ${keyword}`);
  if (keyword !== "") {
   const getTitles = filter(keyword);
   this.setState({
    results: getTitles
   });
  } else {
   this.setState({
   results: []
   });
  }
 }, 400);
```

# To Sum Up…

**Controlled Form elements:**

- Are critical components of any application that accepts user data
- Form elements push data into the state, which is then pushed back into the form element
- They are controlled by React

**Uncontrolled Form Components:**

- A component which is not contolled by React
- Refs allow to reach oiut into the DOM and access a node directly

**Handling inputs efficiently:**

- Multiple event handlers can be used to handle multiple input elements
- Debouncing is the process of delaying a function. It reduces the no. Of API requests by delaying the execution

# thank you!