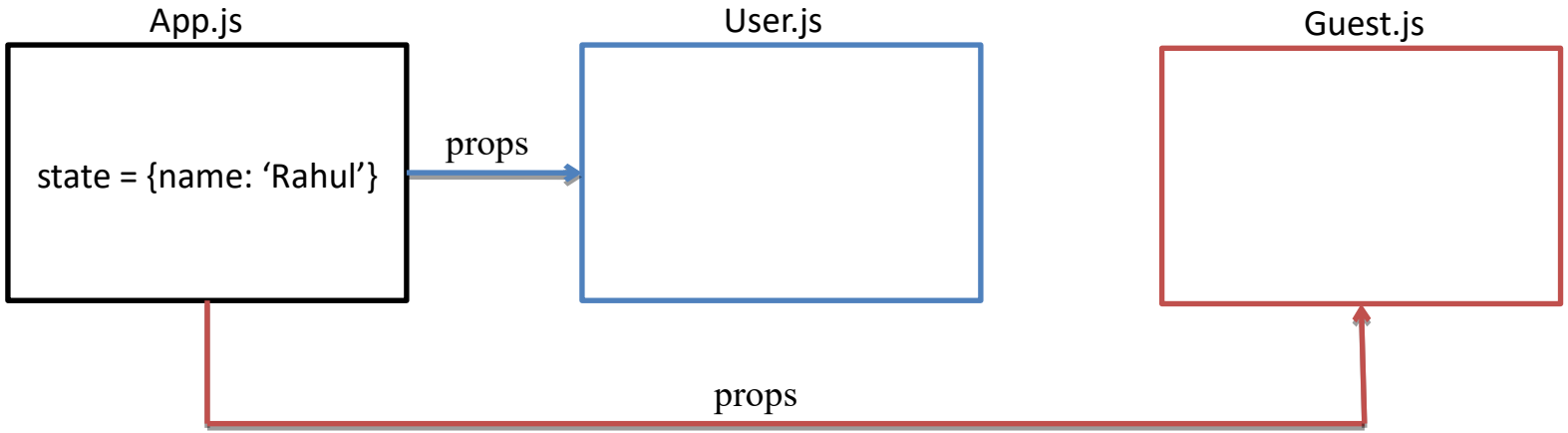
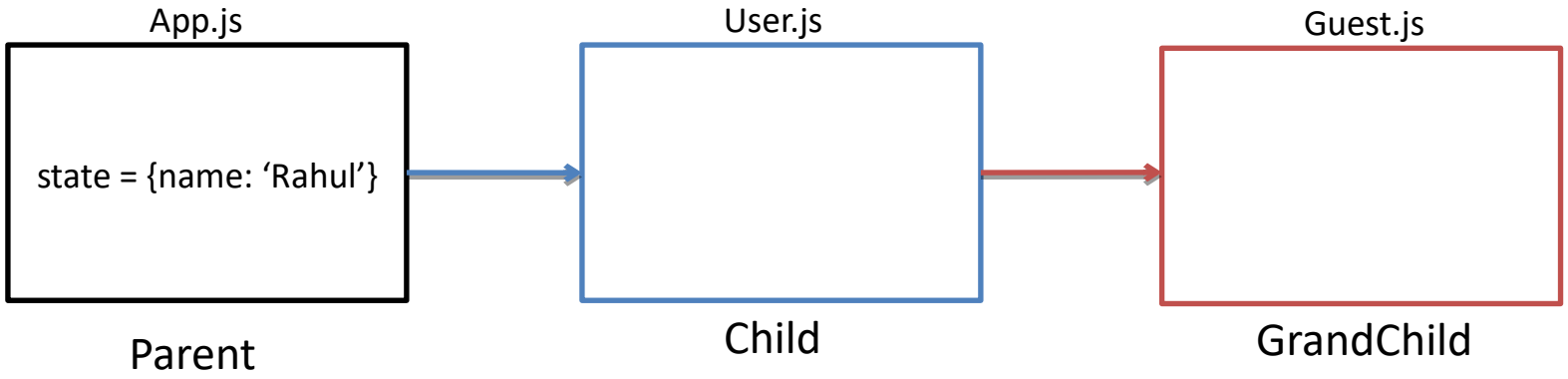
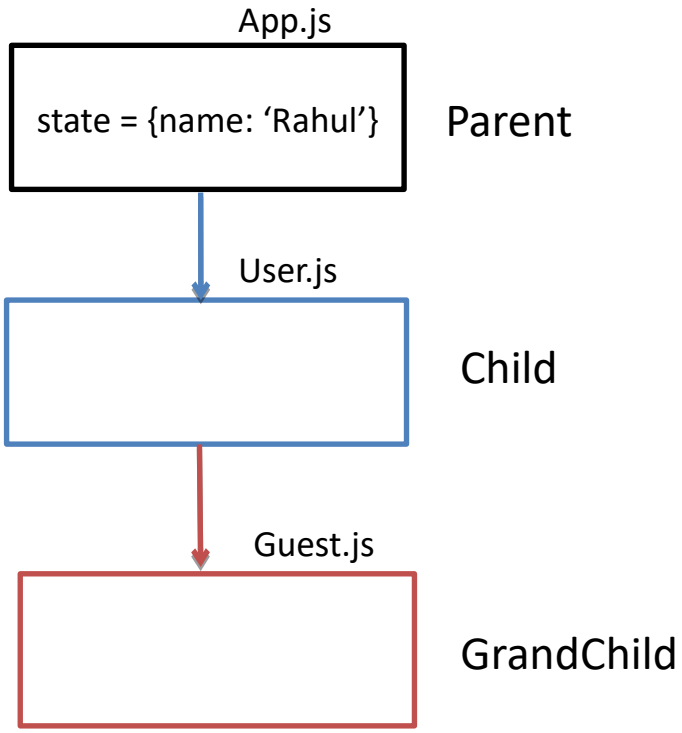
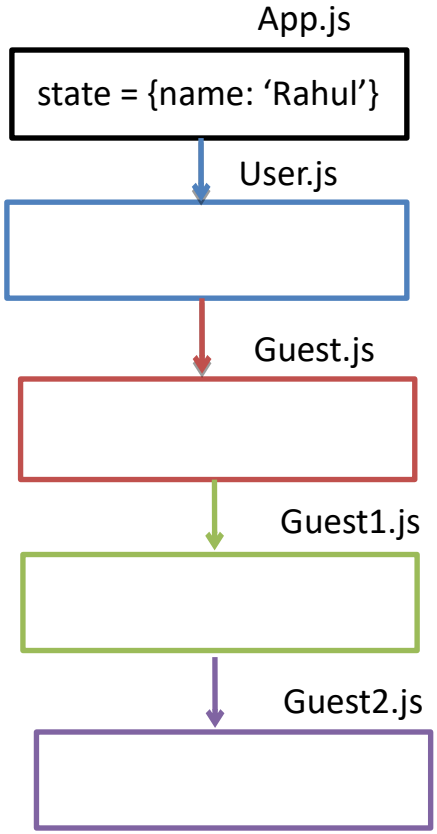
Lifting State up




App.js
state = {name: ‘Rahul’}
Parent
User.js
Child
Guest.js
GrandChild


App.js
state = {name: 'Rahul'}
Parent
User.js
Child
Guest.js
GrandChild


App.js
state = {name: 'Rahul'}
User.js
Guest.js
Guest1.js
Guest2.js

Context

Context provides a way to pass data through the component tree without having to pass props down manually at every level.

In a typical React application, data is passed top-down (parent to child) via props, but this can be cumbersome for certain types of props that are required by many components within an application.

Context provides a way to share values like these between components without having to explicitly pass a prop through every level of the tree.

Context is designed to share data that can be considered “global” for a tree of React components, such as the current authenticated user, theme, or preferred language.

- Passing the initial state to *React.createContext*. This function then returns an object with a *Provider* and a *Consumer*.
- Using the *Provider* component at the top of the tree and making it accept a prop called *value*. This *value* can be anything!
- Using the *Consumer* component anywhere below the Provider in the component tree to get a subset of the state.

Create Context

It creates a Context object.

When React renders a component that subscribes to this Context object it will read the current context value from the closest matching *Provider* above it in the tree.

Syntax: -

```
const MyContext = React.createContext(defaultValue);
```

defaultValue - It is only used when a component does not have a matching *Provider* above it in the tree.

Ex:-

```
const MyContext = React.createContext(false);
const MyContext = React.createContext(‘white’);
const MyContext = React.createContext({ user: 'Guest', });
```

Context Provider

Every Context object comes with a *Provider* React component that allows consuming components to subscribe to context changes.

One Provider can be connected to many consumers. Providers can be nested to override values deeper within the tree.

Syntax:-

```
<MyContext.Provider value={/* some value */}>
```

A *value* prop to be passed to consuming components that are descendants of this *Provider*.

Ex:-

```
<MyContext.Provider value={this.state.name}>
```

Context Consumer

A React component that subscribes to context changes. This lets you subscribe to a context within a function component.

It requires a function as a child. The function receives the current context value and returns a React node.

The *value* argument passed to the function will be equal to the *value* prop of the closest *Provider* for this context above in the tree.

If there is no *Provider* for this context above, the *value* argument will be equal to the *defaultValue* that was passed to createContext().

```
<MyContext.Consumer>
  {value => /* render something based on the context value */}
</MyContext.Consumer>
```

Context Type

The *contextType* property on a class can be assigned a Context object created by *React.createContext()*. This lets you consume the nearest current value of that Context type using *this.context*.

static *contextType* = MyContext;