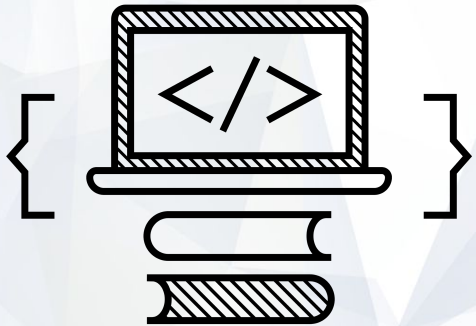
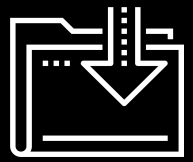


Sharing State

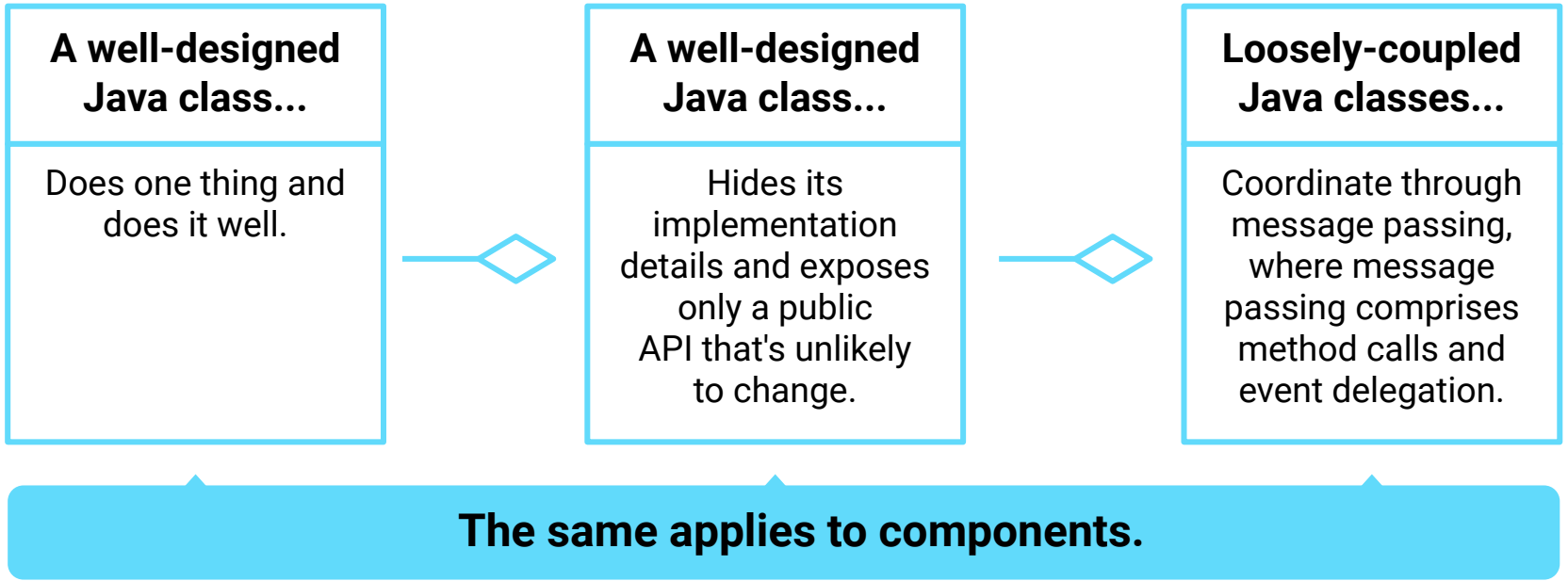
Course: Java

S1



Sharing State

Think back to Java class modeling. Class modeling and component design use the same principles. Java classes allow us to separate application concerns.

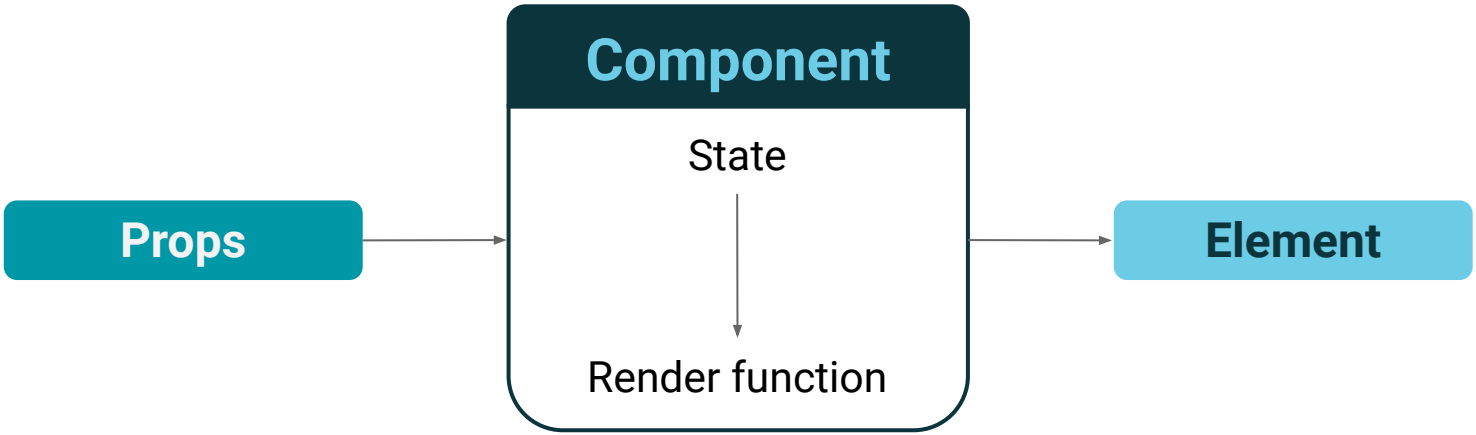


Component design and class design are the same thing but in a different layer of the application.

Sharing State (continued)

Like Java classes, components need a way to share messages, which is just a different way of saying that components need a way to share changes in state.

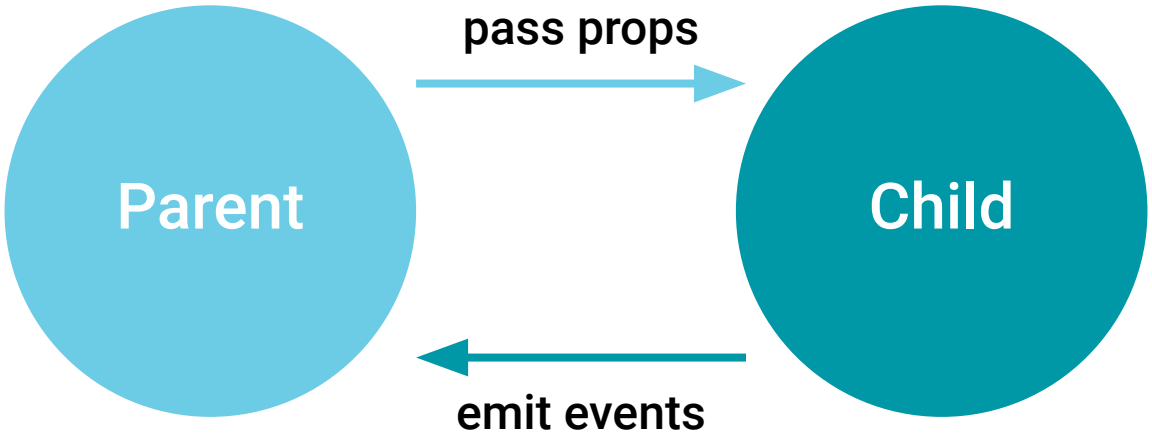
We've already seen an example of parent-to-child state sharing through props.



Children communicate with their parent via events.

Sharing State (continued)

When components communicate in both directions, they can solve interesting problems.



Learning Outcomes

By the end of this lesson, you will be able to:

01 Pass a callback function via props.

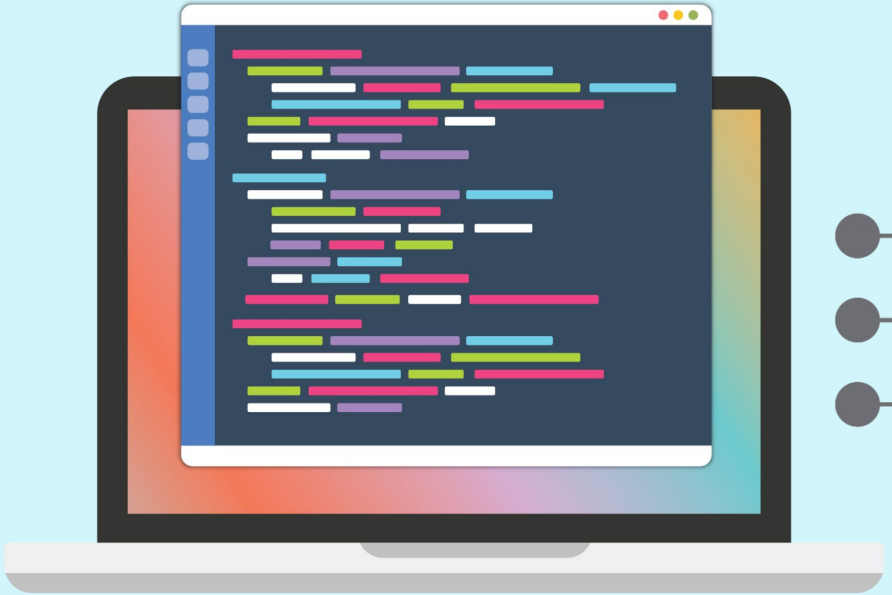
02 Execute a callback in a child component when an event is triggered in the child.

03 Update the parent state using a callback function.

04 Draw a diagram that shows how state is shared from parent to child and from child to parent.

Props and Events

Props and Events



Let’s start with a demonstration.

We’ll take a “show, explain, practice” approach versus an “explain, show, practice” approach.

In this case, it’s easier to identify the concepts once we've seen a bit of code.

Time to Code

Props Down, Events Up

Suggested Time:

15 Minutes

Props Down, Events Up

“Props down, events up” means the following:

01 The parent creates a callback function that is passed to child components to be used when an event is triggered in the child.

02 The parent also passes “props down” to the child, which may be part of its state.

03 An event in the child executes the parent’s callback. That’s the “events up.”

04 The callback causes a state change that updates the props. The “event up” creates a cascading “prop down.” It’s a cycle but never an infinite cycle.

Props Down, Events Up Summary

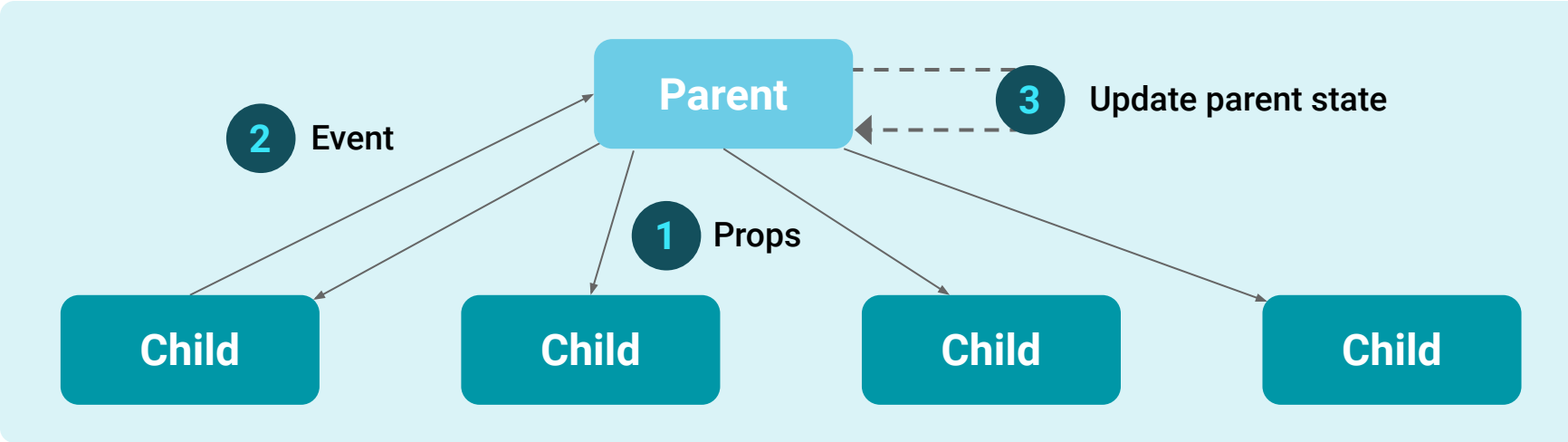
Props Down

- Parents share state with children via props.

Events Up

- Children share state with parents via events and callbacks.
- The parent shares a callback function via props.
- Then a child's event triggers that function, likely updating the parent state.
- This in turn can cause a change in props.

Props Down, Events Up



1. The parent shares state as props. It also shares a callback.
2. The child's event triggers the callback.
3. The callback causes a parent state change, which may also cause a prop change, which brings us back to Step 1.

In the Parent

- Create a callback.
- It can contain any number of parameters. (The child must know expectations.)
- Pass it with props.

```
// Give the callback a meaningful name.
const callback = (p1, p2) => {
    // Do something useful here.
    // Most callbacks alter parent state.
    // Callbacks can have zero to many parameters.
};

// Pass the callback and required state.
return <Child value={state} onClick={callback} />;
```

In the Child

- The callback's name in the child is the prop name.
- There's no one way to use a callback. Clicking is only one option.
- The only common theme is that an event in the child triggers the callback.

```
const handleClick = (evt) => {
    // Do something interesting that applies to
    // this component only.

    // Then send expected parameters to the parent.
    onClick(...state);
};

return <button onClick={handleClick}>Click Me!</button>;
```

Questions?

Time to Code

Lightswitch, Slightly Less MVP ToDos

Suggested Time:

15 Minutes

Lightswitch, Slightly Less MVP ToDos

Lightswitch

Focuses

- `Light` styles change colors as the `LightSwitches` turn on and off.
- `LightSwitch` is mostly a checkbox whose `checked` state synchronizes with the current `isOn` state of the `Light`.
- Both `Light` and `LightSwitch` track their own state. Point this out to learners. `Light` counts the total changes, whereas each instance of a `LightSwitch` counts only its changes.

Slightly Less MVP ToDos

Focuses

- Parents aren't limited to one callback. They can pass as many callbacks as they want to any child.
- Use components to separate concerns. (The `ToDo` component is so-so because it can't stand on its own. It relies on an `<ul>` or `<ol>`.)
- We're getting closer to a full CRUD application.

Questions?

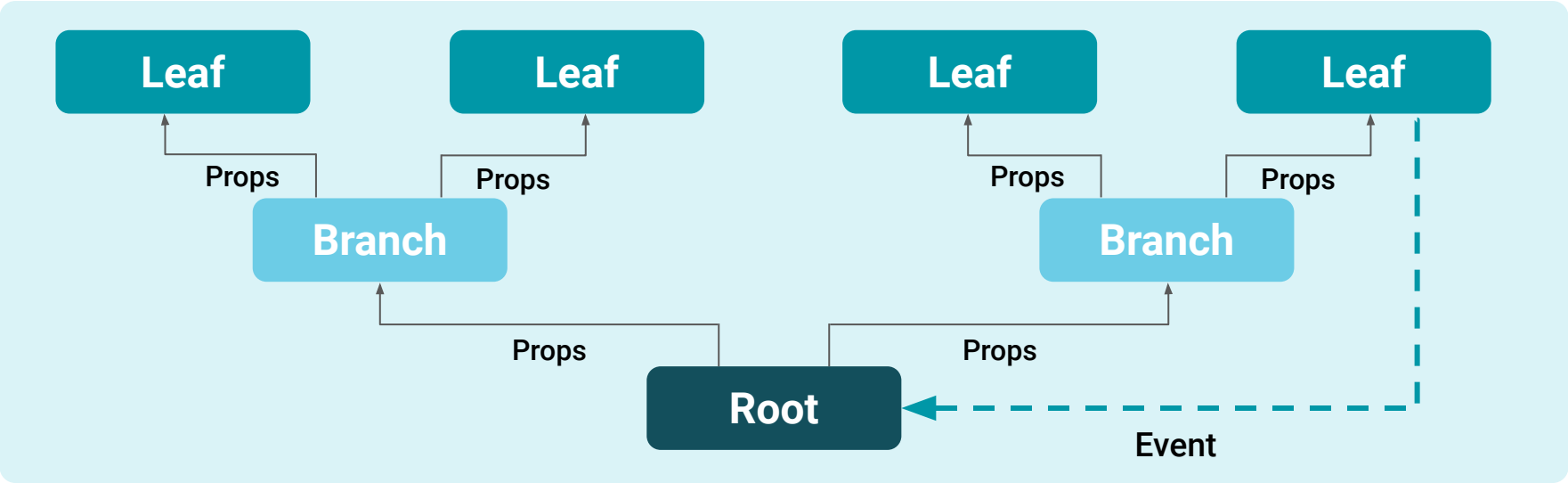

Break

Component Hierarchy

Messages from Grandchildren

A component can pass a callback beyond its immediate child.

Then the grandchild (or great-grandchild, etc.) can trigger an action in the root component based on its event.



Time to Code

Component Hierarchies

Suggested Time:

15 Minutes

Component Hierarchies

Focuses

Trace the callback’s path through props.

Neither `Branch` nor `Leaf` track state formally with `useState`. They rely only on props passed from `Root`.

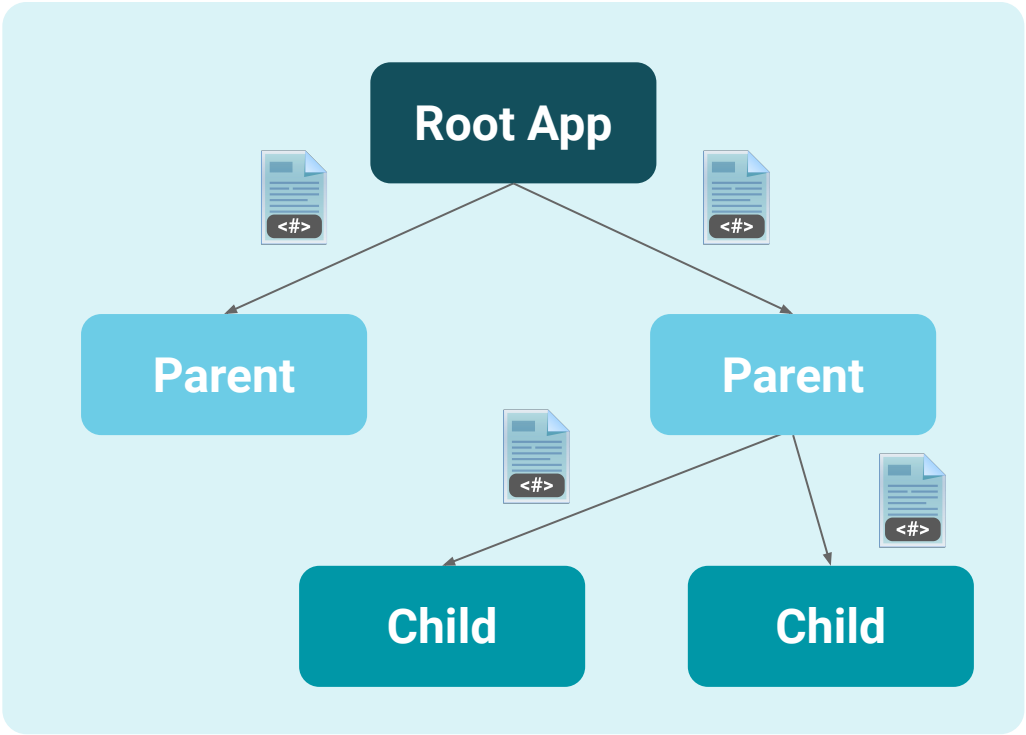
`Root` shares a CSS “class” via props.

Component Hierarchies

The React community talks about the concepts of **props drilling**.

Props drilling is when a component much higher in the hierarchy must “drill” a callback or bit of data deep into the hierarchy.

The prop just passes through many components. In this example, the callback drills past `Branch` and is of little use to it.



Can you imagine pros and cons for this approach?

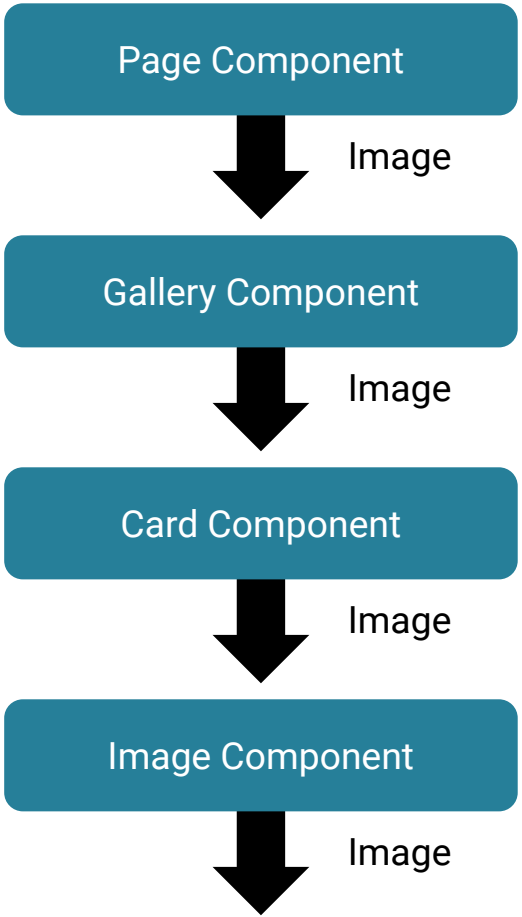
Prop Drilling

Prop drilling is the process you go through to get data to parts of the React component tree.

Although it seems tedious, prop drilling is often necessary to avoid complicating the global state of your application.

Remember, it's often best to keep our state as close as possible to where it's relevant.

Why can't we just add state to the lowest level component?



Questions?

Activity: Sharing State

Suggested Time:

30 Minutes

Time's Up! Let's Review.

Questions?


RECAP

Recap

- Which events can trigger a callback in the parent?
- Which values are valid as props?
- Explain “props down, events up” in your own terms.
- What should be tracked as private state, and what should be shared via props?
- Is it a good idea to add props to state?

Learning Outcomes

By the end of this lesson, you will be able to:

01 Pass a callback function via props.

02 Execute a callback in a child component when an event is triggered in the child.

03 Update the parent state using a callback function.

04 Draw a diagram that shows how state is shared from parent to child and from child to parent.


The End