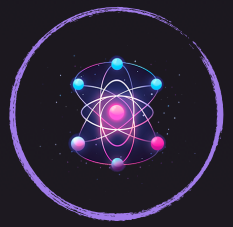
ACADE MIND



# React Essentials - Deep Dive

## Beyond The Basics

- Behind The Scenes of **JSX**
- Structuring **Components** and **State**
- **Advanced State** Usage
- **Patterns & Best Practices**

# You Don't Need JSX (But It's Convenient)

```jsx
<div id="content">
  <p>Hello World!</p>
</div>
```

Requires build process & code transformation

Easy to read & understand

```jsx
React.createElement(
  'div',
  { id: 'content' },
  React.createElement(
    'p',
    null,
    'Hello World'
  )
);
```

Works without special build process & transformation

Pretty verbose & not necessarily intuitive

**Component Type**

Identifies the to-be-rendered component

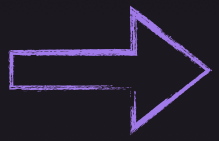
**Props Object**

Sets component props

**Child Content**

The content passed between the component tags

# Additional Key Component & Props Concepts



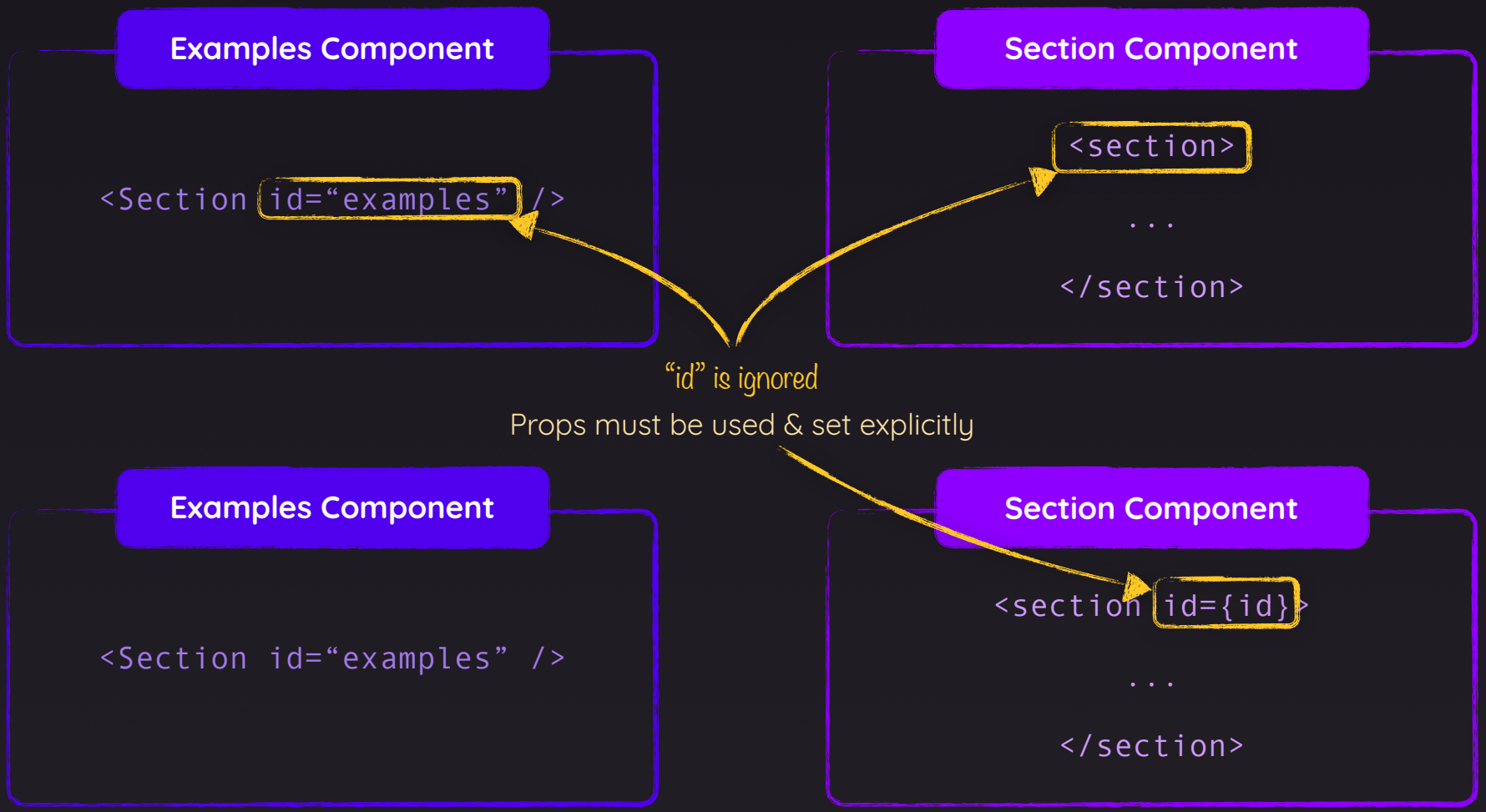
Forwarded Props

Multiple Component Slots

Element Identifiers as Props

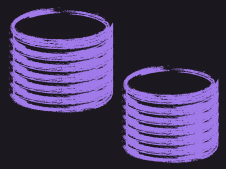
Default Prop Values

# Props Are Not Forwarded Automatically



ACADEMIND

# New Project, New Concepts



Multiple State Values

Nested Lists

Lifting State Up

Array & Object States

Derived State

Component Functions vs "Normal Functions"

# Updating State Based On Old State

```
setIsEditing(!isEditing);
```

If your **new state depends on your previous state** value, you should **not** update the state like this

```
setIsEditing(wasEditing => !wasEditing);
```

Instead, **pass a function** to your state updating function

This function will **automatically be called** by React and will receive the **guaranteed latest state value**

# React Is Scheduling State Updates

## Somewhere in your code

Code snippets in different places in your app's code

```
setIsEditing(true);
...
setIsEditing(false);
...
setIsEditing(true);
...
setIsEditing(false);
```

## React's State Updating Schedule

Managed behind the scenes by React

1 → Update state to `true`

2 → Update state to `false`

3 → Update state to `true`

4 → Update state to `false`

State updates are **not performed instantly** but at some point in the future
(when React has time for it)

In most cases, those state updates of course still are executed **almost instantly**

# Update Object-State Immutably

**Objects & arrays** (which technically are objects) are **reference values** in JavaScript

You should therefore **not mutate** them directly — instead create a **(deep) copy** first!

NOT creating a copy (because user is an object = a reference value)

```
const updatedUser = user;
user.name = 'Max'
```

Editing the user object in memory

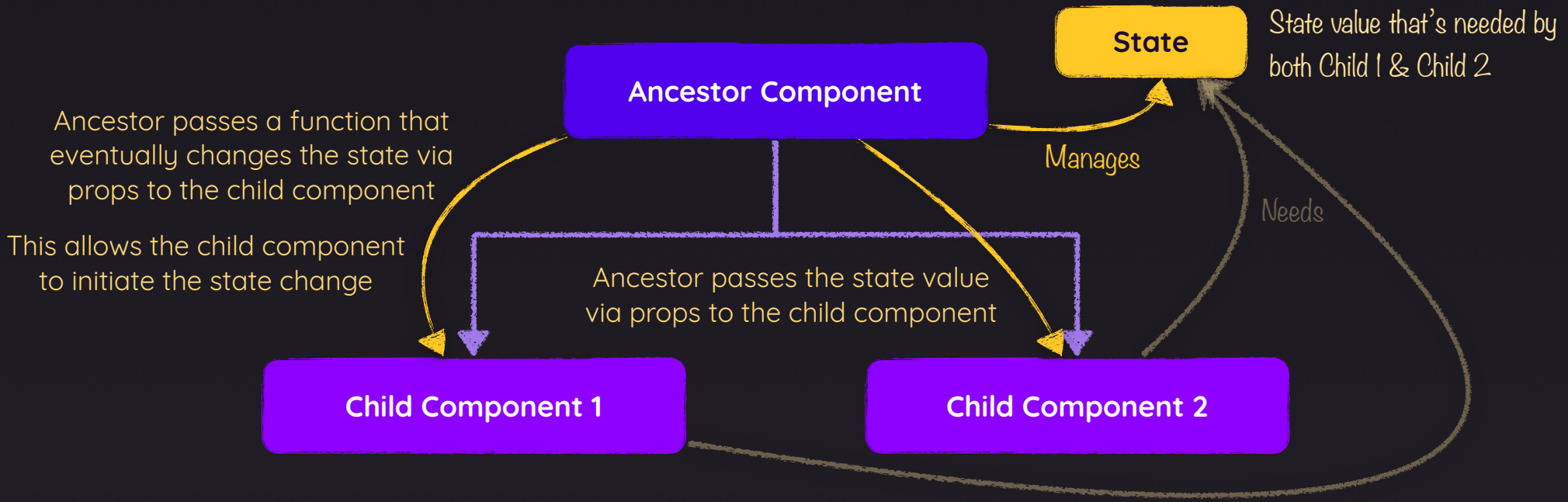
Creating a copy via JavaScript's "Spread" operator

```
const updatedUser = { ...user }
updatedUser.name = 'Max'
```

Editing the copy, not the original

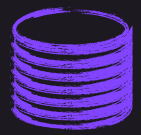
# Lifting State Up

Lift the state up to the **closest ancestor component** that has access to all components that need to work with that state



# You Don't Always Have Just One State Value

**Player data** and **game state** must be managed



## Combined State Object

Data from different app features is managed in a **single, shared state object**

You must ensure that you don't **accidentally lose data** related to feature B when feature A's data changes

## Separate State Slices

Data from different app features is managed in **different state values**

Suboptimal if states from different features depend on each other