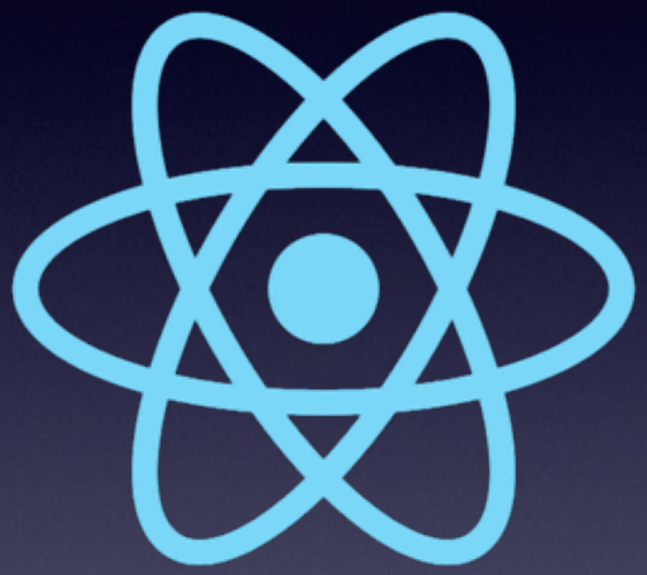


React实战

王沛 SAP高级工程师

关于我

- 2007年毕业于南京大学
- 《征服Ajax：Web2.0开发技术详解》，2006
- 《Web2.0界面开发模式》，2012
- InfoQ 深入浅出React专栏作者

React很简单

1个新概念

React很简单

一个新概念：组件

```
class HelloWorld extends React.Component {
  render() {
    return <div>Hello {this.props.name}!</div>;
  }
}
```

1个新概念

4个必须API

React很简单

4个必须API：render, setState, state, props

```
class HelloWorld extends React.Component {
  render() {
    return <div>Hello {this.props.name}!</div>;
  }
}

React.render(<HelloWorld name="Nate" />,
document.body);
```

1个新概念

4个必须API

React很简单

单向数据流

1个新概念

4个必须API

React很简单

单向数据流

完善的错误提示

反之：何为复杂?


24 mm

jQuery：为局部更新而生

Selectors

Basics
- *
- .class
- element
- #id
- selector1, selectorN, ...

Hierarchy
- parent > child
- ancestor descendant
- prev + next
- prev ~ siblings

Basic Filters
- :animated
- :eq()
- :even
- :first
- :gt()
- :header
- :lang()
- :last
- :lt()
- :not()
- :odd
- :root
- :target

Content Filters
- :contains()
- :empty
- :has()
- :parent

Visibility Filters
- :hidden
- :visible

Attribute
- [name|="value"]
- [name*="value"]
- [name~="value"]
- [name$="value"]
- [name="value"]
- [name!="value"]
- [name^="value"]
- [name]
- [name="value"]
- [name2="value2"]

Child Filters
- :first-child
- :first-of-type
- :last-child
- :last-of-type
- :nth-child()
- :nth-last-child()
- :nth-last-of-type()
- :nth-of-type()
- :only-child
- :only-of-type()

Forms
- :button
- :checkbox
- :checked
- :disabled
- :enabled
- :focus
- :file
- :image
- :input
- :password
- :radio
- :reset
- :selected
- :submit
- :text

Attributes/CSS

Attributes
- .attr()
- .prop()
- .removeAttr()
- .removeProp()
- .val()

CSS
- .addClass()
- .css()
- jQuery.cssHooks
- .hasClass()
- .removeClass()
- .toggleClass()

Dimensions
- .height()
- .innerHeight()
- .innerWidth()
- .outerHeight()
- .outerWidth()
- .width()

Offset
- .offset()
- .offsetParent()
- .position()
- .scrollLeft()
- .scrollTop()

Data
- jQuery.data()
- .data()
- jQuery.hasData()
- jQuery.removeData()
- .removeData()

Manipulation

Copying
- .clone()

DOM Insertion, Around
- .wrap()
- .wrapAll()
- .wrapInner()

DOM Insertion, Inside
- .append()
- .appendTo()
- .html()
- .prepend()
- .prependTo()
- .text()

DOM Insertion, Outside
- .after()
- .before()
- .insertAfter()
- .insertBefore()

DOM Removal
- .detach()
- .empty()
- .remove()
- .unwrap()

DOM Replacement
- .replaceAll()
- .replaceWith()

Traversing

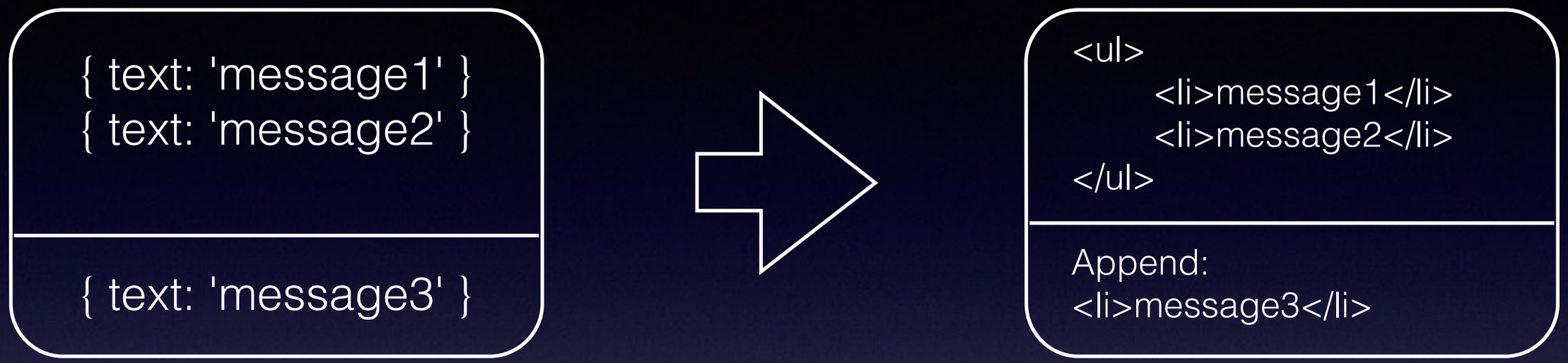
Filtering
- .eq()
- .filter()
- .first()
- .has()
- .is()
- .last()
- .map()
- .not()
- .slice()

Miscellaneous Traversing
- .add()
- .andSelf()
- .contents()
- .each()
- .end()

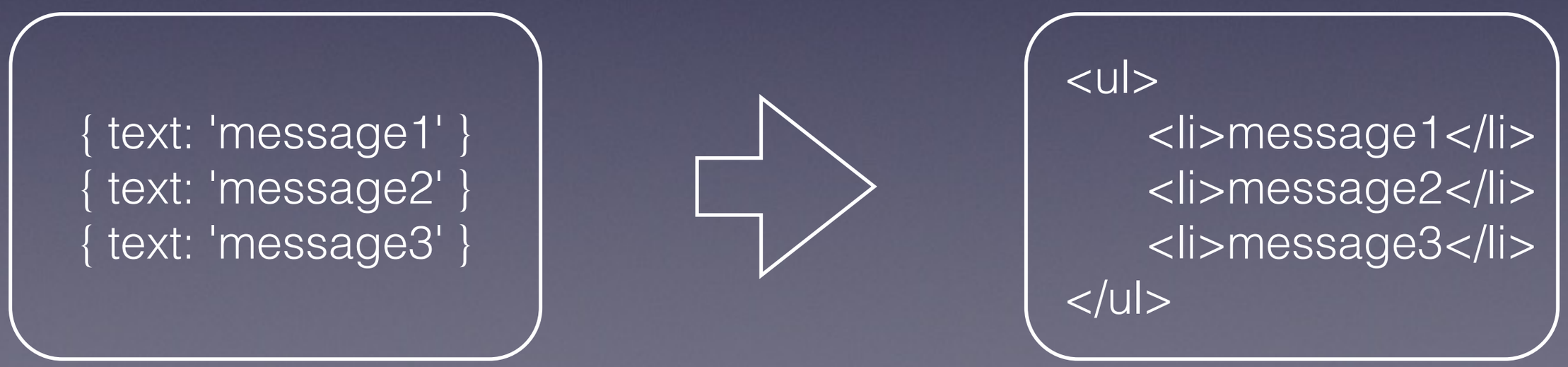
Tree Traversal
- .addBack()
- .children()
- .closest()
- .find()
- .next()
- .nextAll()
- .nextUntil()
- .parent()
- .parents()
- .parentsUntil()
- .prev()
- .prevAll()
- .prevUntil()
- .siblings()

React: 始终整体“刷新”

局部刷新

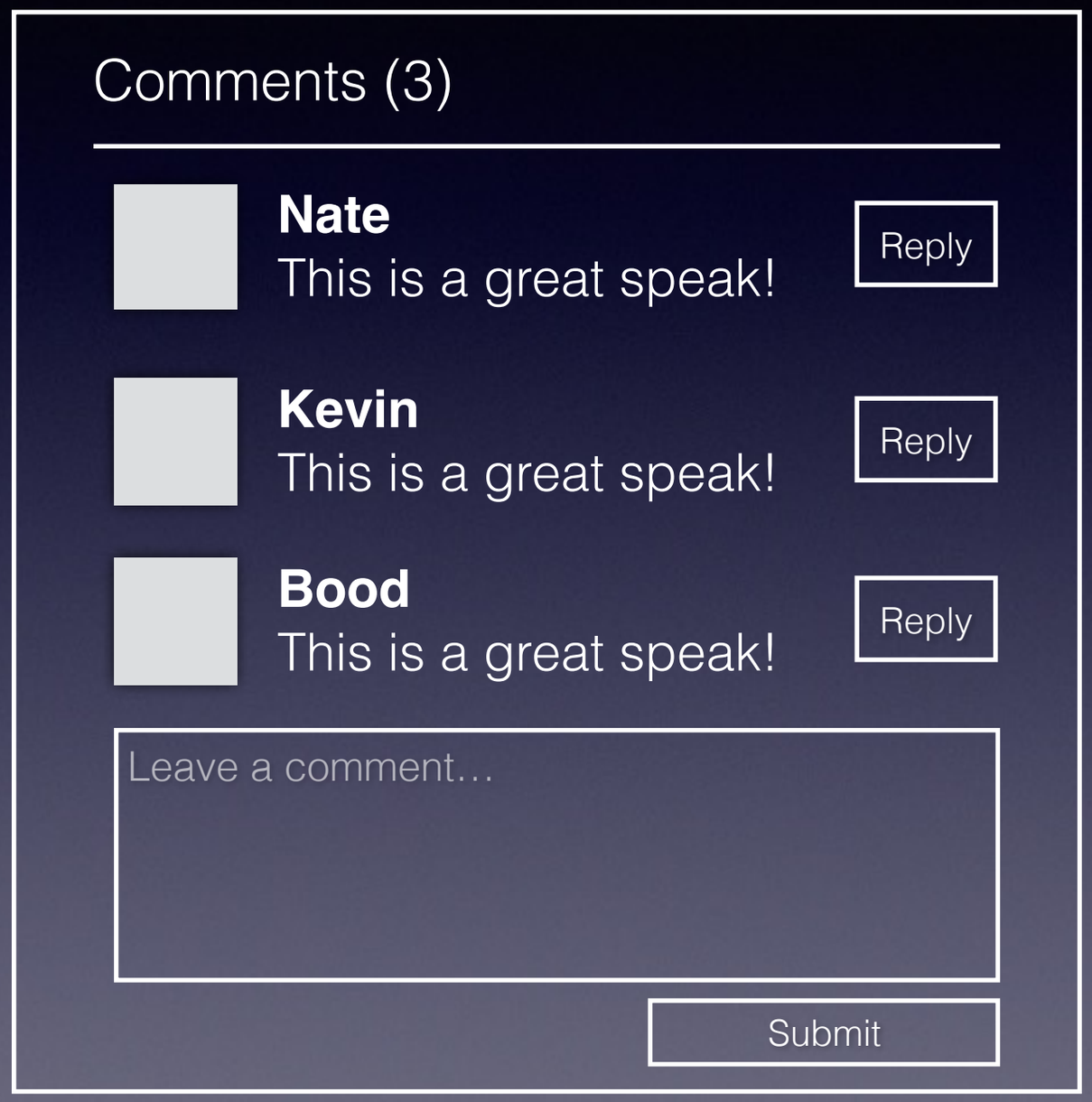


React：整体刷新

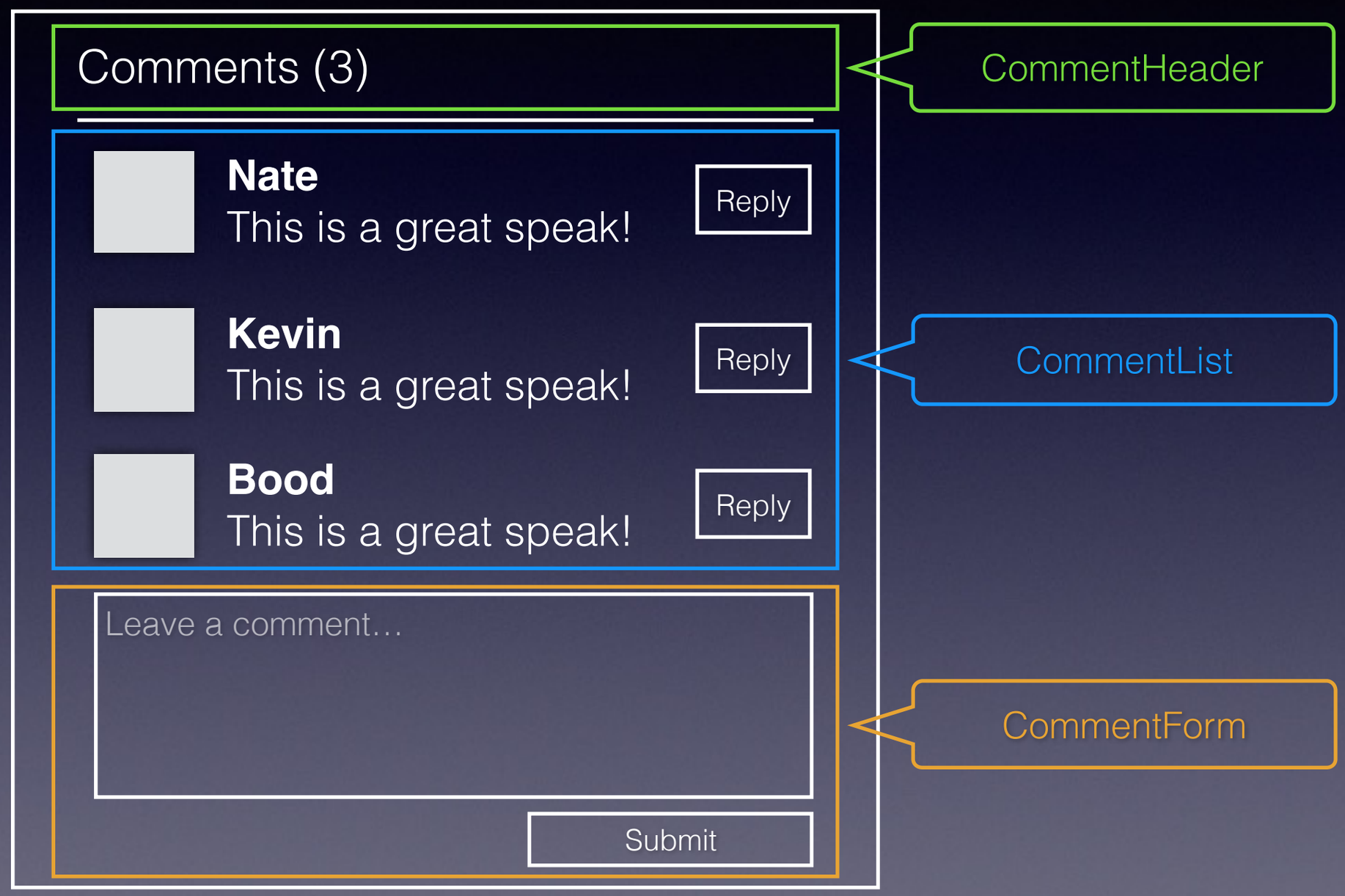


使用组件描述整体界面

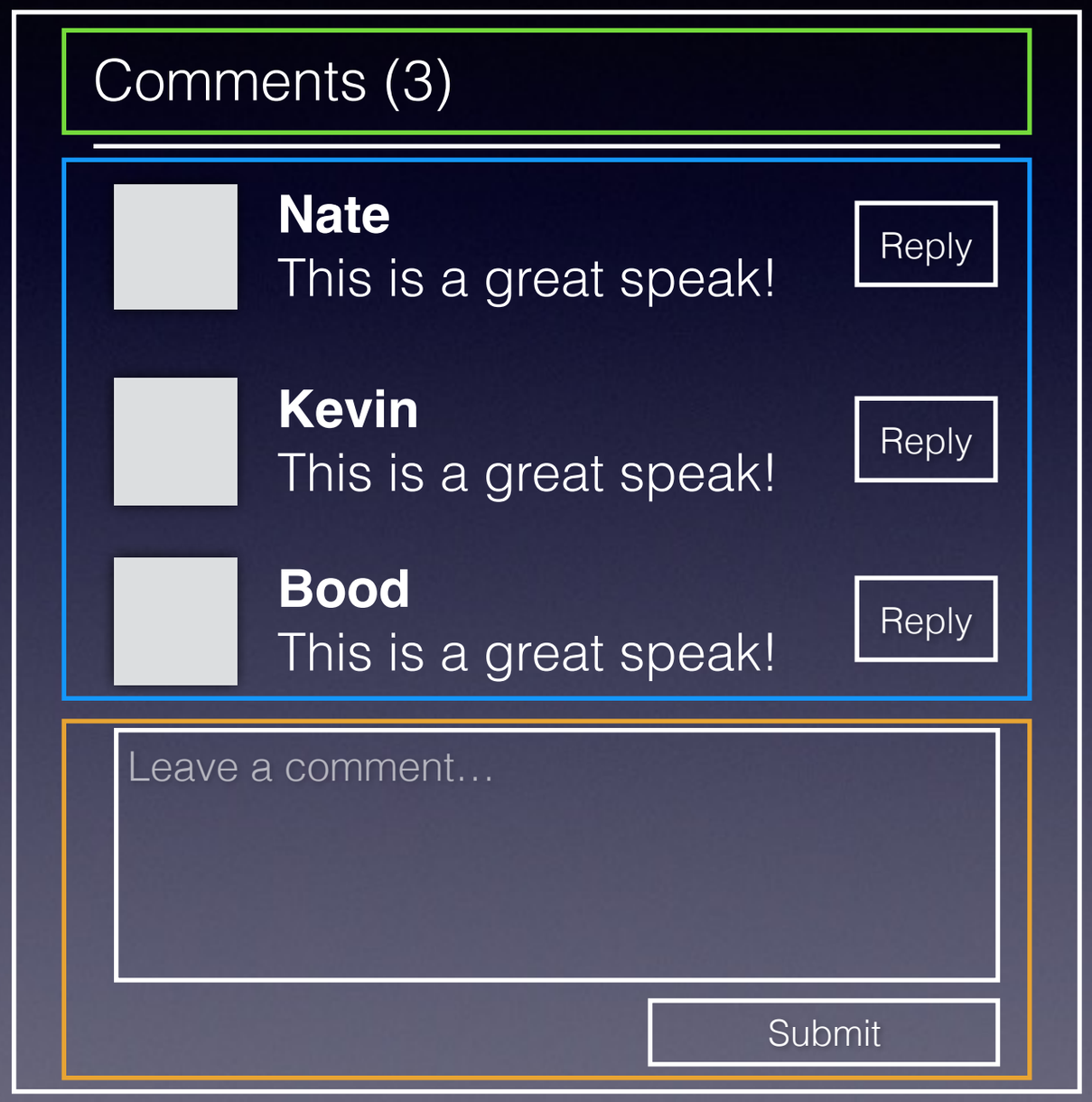
用组件化思路考虑UI构成



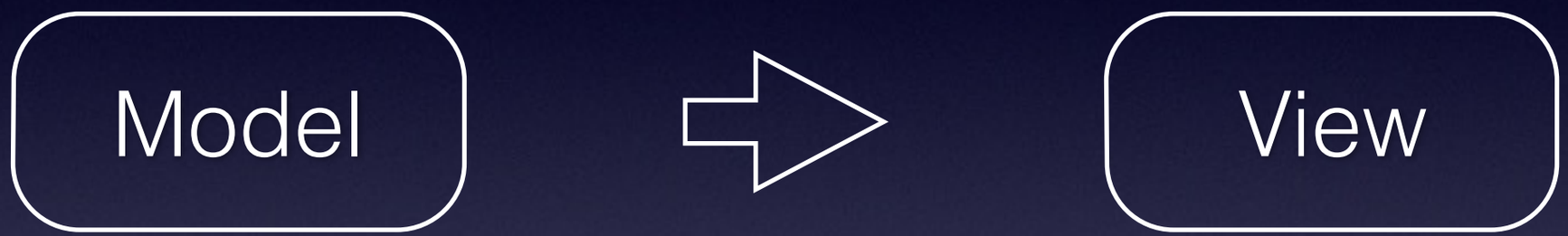
用组件化思路考虑UI构成



用组件化思路考虑UI构成




```
<CommentBox>
  <Header />
  <CommentList />
  <SubmitForm />
</CommentBox>
```


Model
View

Model是否足以定义View?

一个下拉框组件的例子

Close

Open

理解React组件

- React组件一般不提供方法，而是某种状态机
- React组件可以理解为一个纯函数
- 单向数据绑定

一个下拉框组件的例子

```
class DropDownList extends React.Component {
  static propTypes = {
    options: React.PropTypes.array.isRequired,
    onChange: React.PropTypes.func.isRequired,
  }

  state = {
    isOpen: false,
    value: '',
  }

  onOptionClick(value) {
    this.setState({value: value, isOpen: false});
    this.props.onChange(value);
  }

  onInputClick() {
    this.setState({isOpen: !this.state.isOpen});
  }

  render() {
    const options = this.props.options.map(function(option) {
      return <li onClick={this.handleOptionClick}>{option.name}</li>;
    });

    return (
      <div className={'dropdown ' + (this.state.isOpen ? 'is-open' : '')}>
        <input value={this.state.value} onClick={this.handleInputClick}/>
        <ul>
          {options}
        </ul>
      </div>
    );
  }
}
```

定义属性，接受外部传入参数

```
class DropDownList extends React.Component {
  static propTypes = {
    options: React.PropTypes.array.isRequired,
    onChange: React.PropTypes.func.isRequired,
  }

  state = {
```

```
static propTypes = {
  options: React.PropTypes.array.isRequired,
  onChange: React.PropTypes.func.isRequired,
}
```

```
  render() {
    const options = this.props.options.map(function(option) {
      return <li onClick={this.handleOptionClick}>{option.name}</li>;
    });

    return (
      <div className={'dropdown ' + (this.state.isOpen ? 'is-open' : '')}>
        <input value={this.state.value} onClick={this.handleInputClick}/>
        <ul>
          {options}
        </ul>
      </div>
    );
  }
}
```

定义初始内部状态

```
class DropDownList extends React.Component {
  static propTypes = {
    options: React.PropTypes.array.isRequired,
    onChange: React.PropTypes.func.isRequired,
  }

  state = {
```

```
state = {
  isOpen: false,
  value: '',
}
```

```
  render() {
    const options = this.props.options.map(function(option) {
      return <li onClick={this.handleOptionClick}>{option.name}</li>;
    });

    return (
      <div className={'dropdown ' + (this.state.isOpen ? 'is-open' : '')}>
        <input value={this.state.value} onClick={this.handleInputClick}/>
        <ul>
          {options}
        </ul>
      </div>
    );
  }
}
```

输出整体UI

```
class DropDownList extends React.Component {
  static propTypes = {
```

```
render() {
  const options = this.props.options.map(function(option) {
    return <li onClick={this.onOptionClick}>{option.name}</li>;
  });

  return (
    <div className={'dropdown ' + (this.state.isOpen ? 'is-open' : '')}>
      <input value={this.state.value} onClick={this.onInputClick}/>
      <ul>
        {options}
      </ul>
    </div>
  );
}
```

```
    <div className={'dropdown ' + (this.state.isOpen ? 'is-open' : '')}>
      <input value={this.state.value} onClick={this.handleInputClick}/>
      <ul>
        {options}
      </ul>
    </div>
  );
}
}
```

点击选项

```
class DropDownList extends React.Component {
  static propTypes = {
    options: React.PropTypes.array.isRequired,
    onChange: React.PropTypes.func.isRequired,
```

```
onOptionClick(value) {
  this.setState({
    value: value,
    isOpen: false,
  });
  this.props.onChange(value);
}
```

```
  });

  return (
    <div className={'dropdown ' + (this.state.isOpen ? 'is-open' : '')}>
      <input value={this.state.value} onClick={this.handleInputClick}/>
      <ul>
        {options}
      </ul>
    </div>
  );
}
}
```

点击下拉框展开/收起

```
class DropDownList extends React.Component {
  static propTypes = {
    options: React.PropTypes.array.isRequired,
    onChange: React.PropTypes.func.isRequired,
  }
```

```
onInputClick() {
  this.setState({
    isOpen: !this.state.isOpen
  });
}
```

```
    const options = this.props.options.map(function(option) {
      return <li onClick={this.handleOptionClick}>{option.name}</li>;
    });

    return (
      <div className={'dropdown ' + (this.state.isOpen ? 'is-open' : '')}>
        <input value={this.state.value} onClick={this.handleInputClick}/>
        <ul>
          {options}
        </ul>
      </div>
    );
  }
}
```

一个下拉框组件的例子

```
class DropDownList extends React.Component {
  static propTypes = {
    options: React.PropTypes.array.isRequired,
    onChange: React.PropTypes.func.isRequired,
  }

  state = {
    isOpen: false,
    value: '',
  }

  onOptionClick(value) {
    this.setState({value: value, isOpen: false});
    this.props.onChange(value);
  }

  onInputClick() {
    this.setState({isOpen: !this.state.isOpen});
  }

  render() {
    const options = this.props.options.map(function(option) {
      return <li onClick={this.handleOptionClick}>{option.name}</li>;
    });

    return (
      <div className={'dropdown ' + (this.state.isOpen ? 'is-open' : '')}>
        <input value={this.state.value} onClick={this.handleInputClick}/>
        <ul>
          {options}
        </ul>
      </div>
    );
  }
}
```

什么是JSX

```
render() {
  const options = this.props.options.map(function(option) {
    return <li onClick={this.onOptionClick}>{option.name}</li>;
  });

  return (
    <div className={'dropdown ' + (this.state.isOpen ? 'is-open' : '')}>
      <input value={this.state.value} onClick={this.onInputClick}/>
      <ul>
        {options}
      </ul>
    </div>
  );
}
```

理解JSX

JSX其实只是代码创建UI的一种语法糖

```
React.createElement(
  string/ReactClass type,
  [object props],
  [children ...]
)
```

  → React.createElement(‘div’)

<CommentBox /> → React.createElement(CommentBox)

理解JSX

```
var CommentBox = React.createClass({
  render: function () {
    return (
      <div className="comments">
        <h1>Comments ({this.state.items.length})</h1>
        <CommentList data={this.state.items}/>
        <CommentForm />
      </div>
    );
  }
});

React.render(<CommentBox topicId="5678" />, mountNode);
```

```
var CommentBox = React.createClass({displayName: "CommentBox",
  render: function () {
    return (
      React.createElement("div", {className: "comments"},
        React.createElement("h1", null, "Comments (", this.state.items.length,
")"),
        React.createElement(CommentList, {data: this.state.items}),
        React.createElement(CommentForm, null)
      )
    );
  }
});

React.render(React.createElement(CommentBox, {topicId: "5678"}), mountNode);
```

JSX的优点

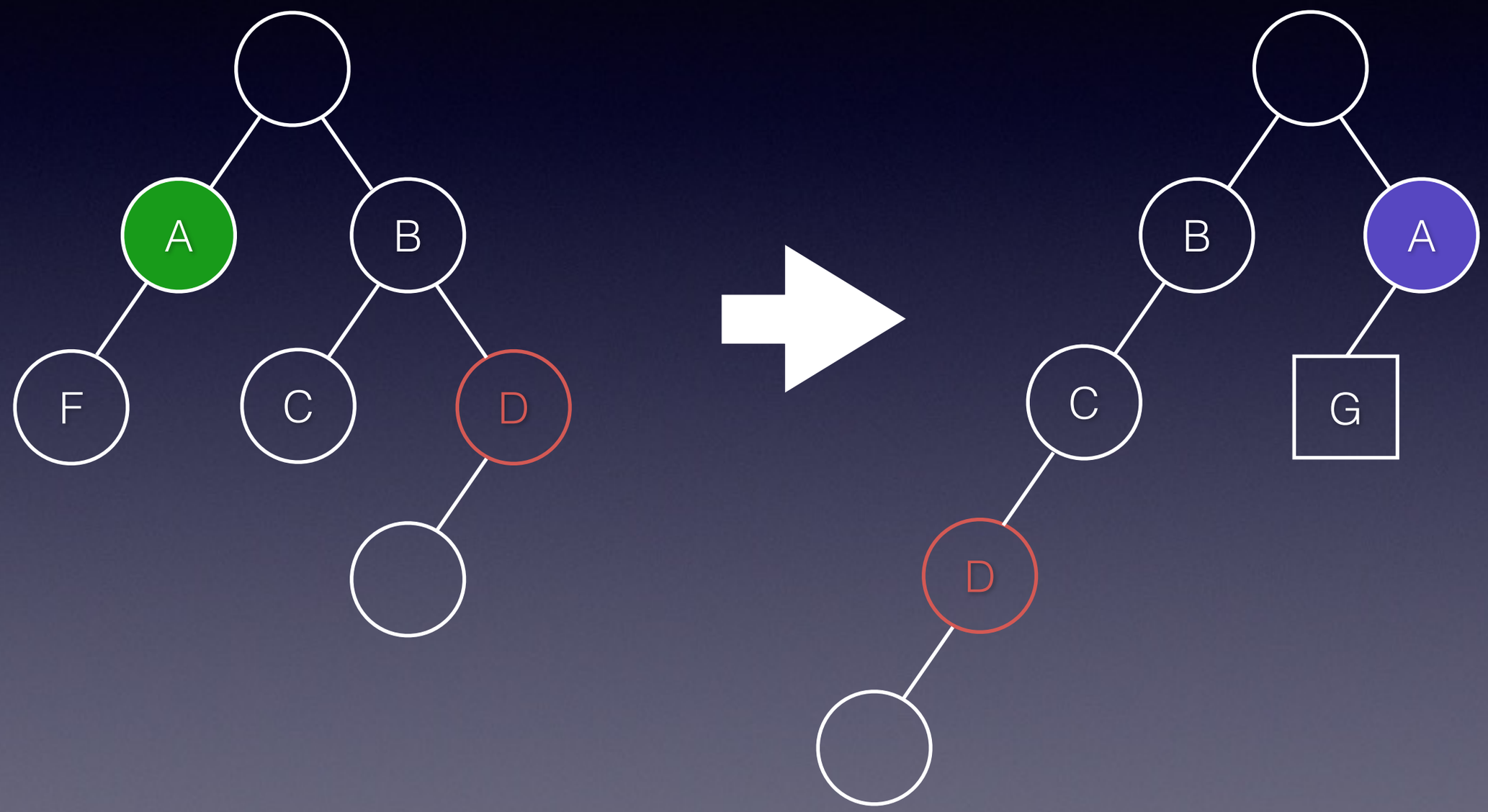
- 声明式创建界面的直观
- 代码动态创建界面的灵活

对比AngularJS模板引擎

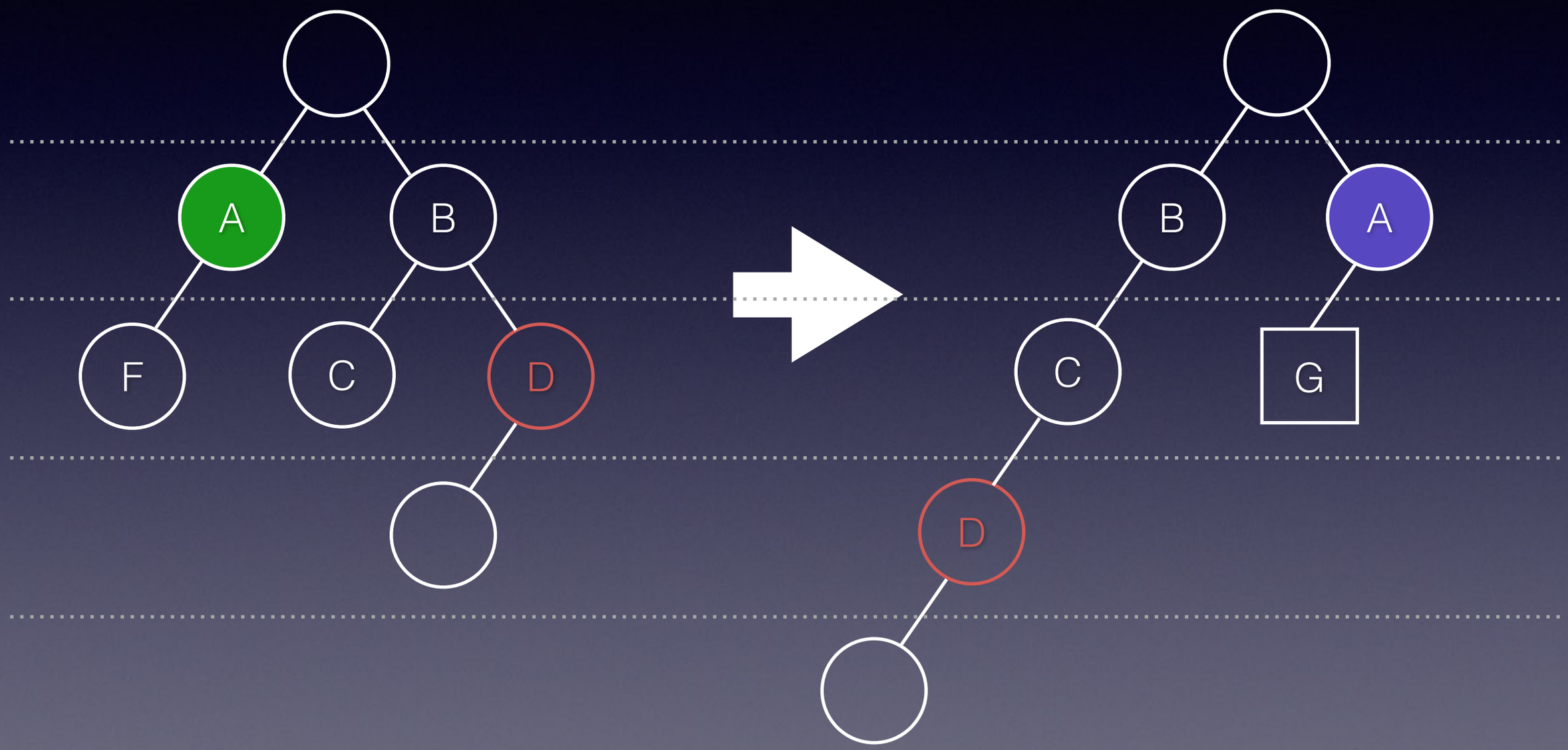
```
<html ng-app="todoApp">
  <head>
    <script src="angular.min.js"></script>
    <script src="todo.js"></script>
    <link rel="stylesheet" href="todo.css">
  </head>
  <body>
    <h2>Todo</h2>
    <div ng-controller="TodoListController as todoList">
      <span>
{{todoList.remaining()}} of {{todoList.todos.length}} remaining
      </span>
      [ <a href="" ng-click="todoList.archive()">archive</a> ]
      <ul class="unstyled">
        <li ng-repeat="todo in todoList.todos">
          <input type="checkbox" ng-model="todo.done">
          <span class="done-{{todo.done}}">{{todo.text}}</span>
        </li>
      </ul>
      <form ng-submit="todoList.addTodo()">
        <input type="text" ng-model="todoList.todoText"  size="30"
               placeholder="add new todo here">
        <input class="btn-primary" type="submit" value="add">
      </form>
    </div>
  </body>
</html>
```

JSX运行基础：虚拟DOM

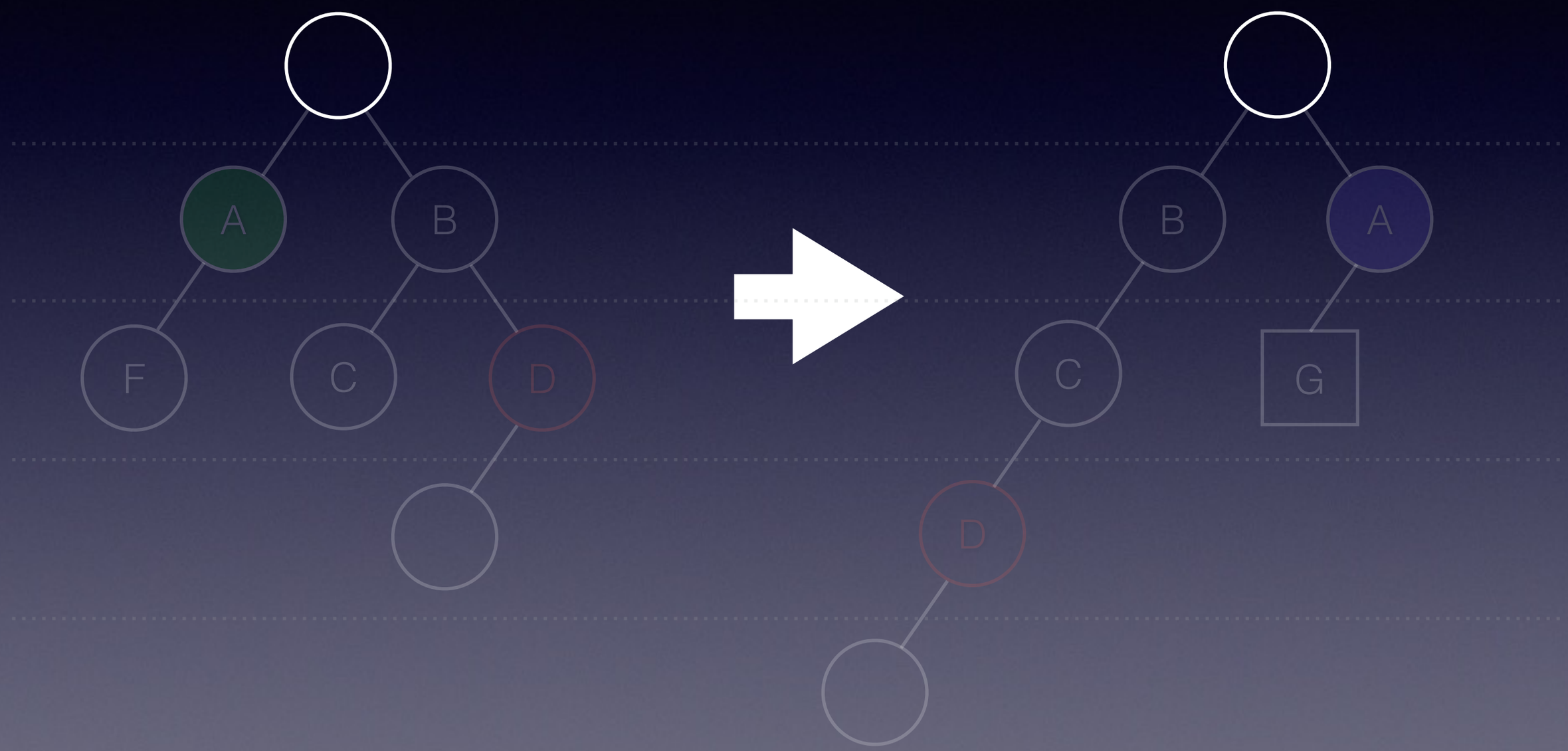
虚拟DOM是如何工作的



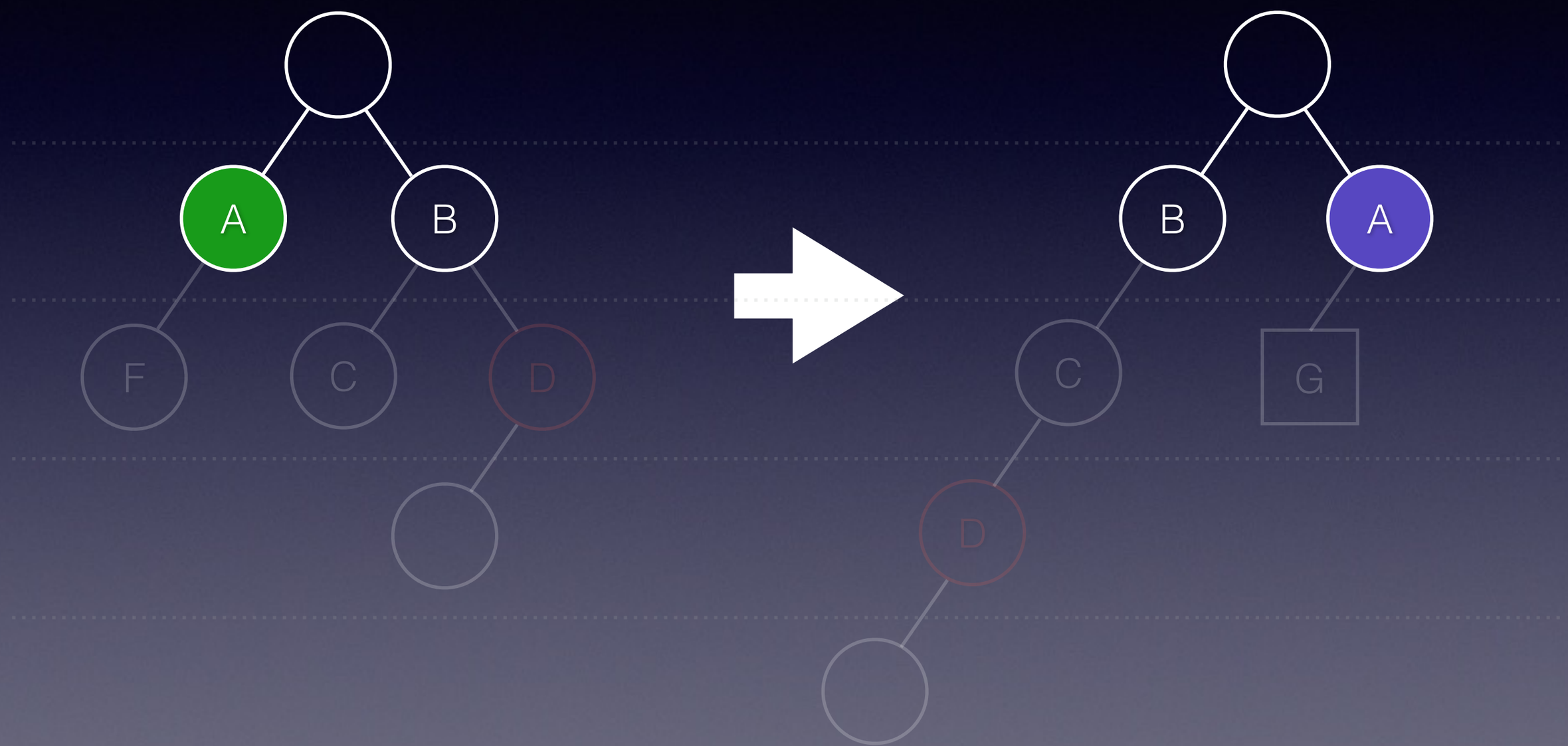
广度优先分层比较



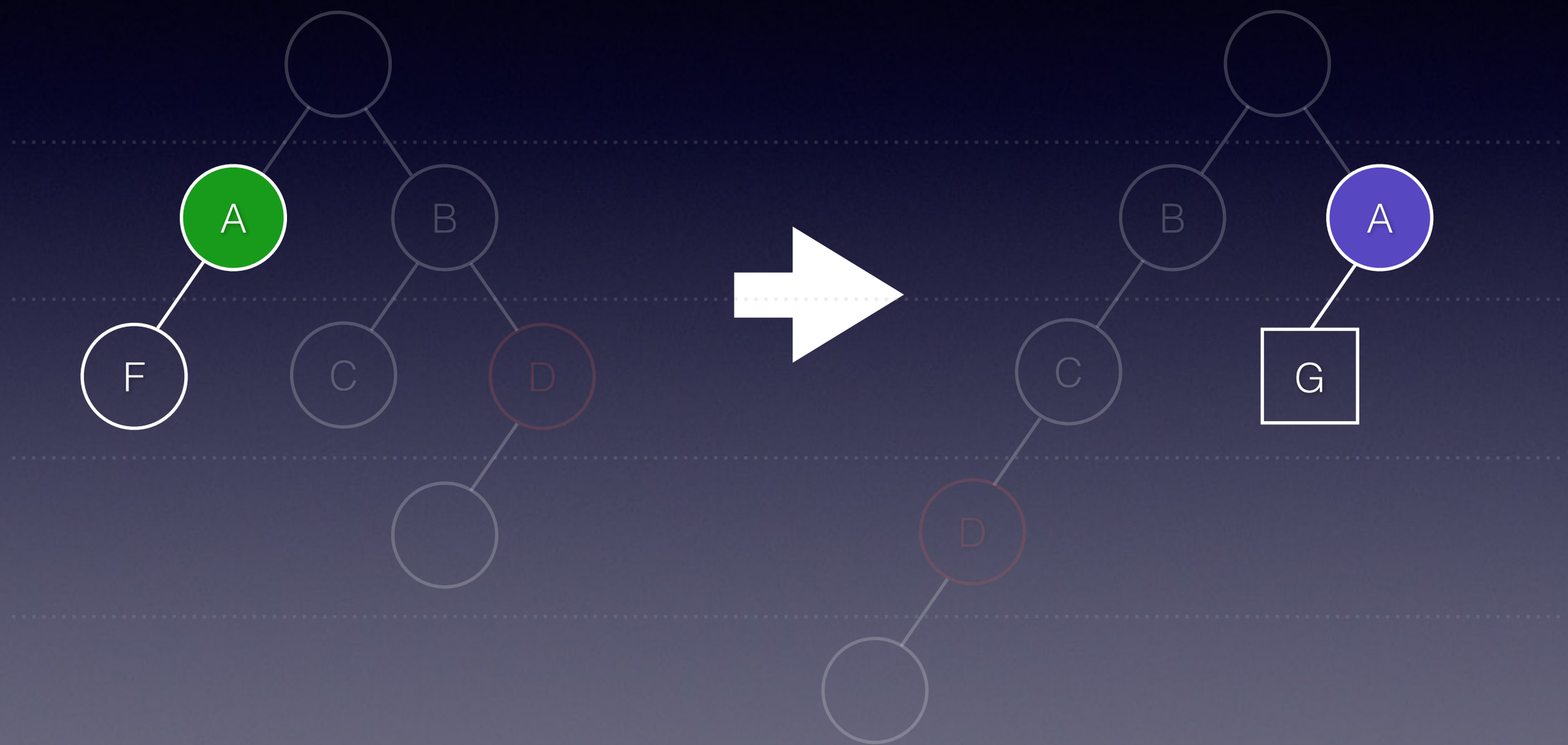
根节点开始比较



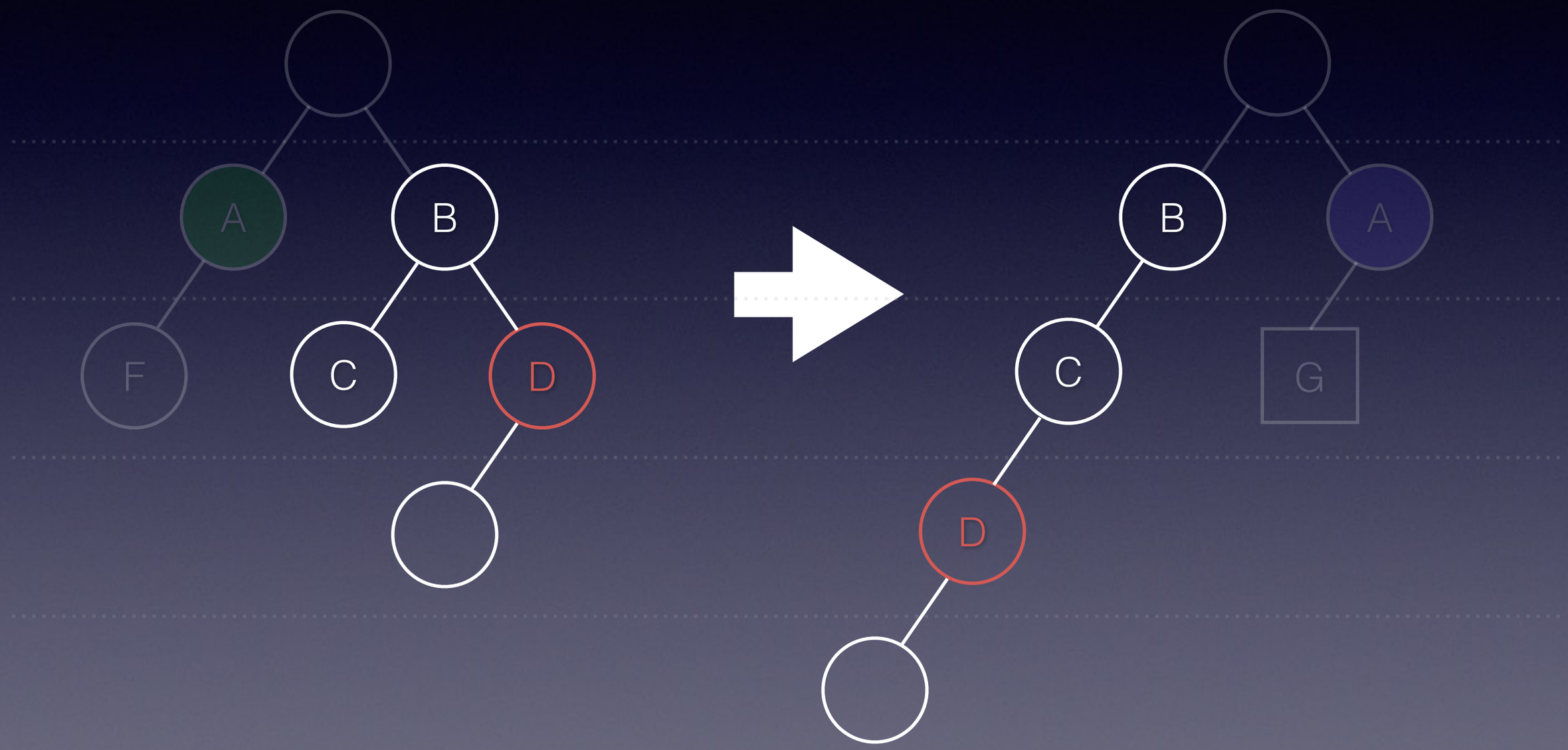
属性变化及顺序改变



结点类型发生变化



结点跨层移动



虚拟DOM的两个假设

1. 组件的DOM结构是相对稳定的
2. 类型相同的兄弟节点可以被唯一的标识

内容回顾

- 每一次界面更新都是整体“刷新”
- 理解组件和JSX
- 虚拟DOM如何工作

谢谢！