Component Rendering and Side Effects

Roland Guijt
Freelance consultant and trainer

@rolandguijt roland.guijt@gmail.com

When the state of a component changes it re-renders

Rendering !== Updating the browser

React element
tr

React element
td

React element
td

React element
tr

React element
td

React element
td

React element
tr

React element
td

React element
td

React element
tr

React element
td

React element
td

React element
tr

React element
td

React element
td

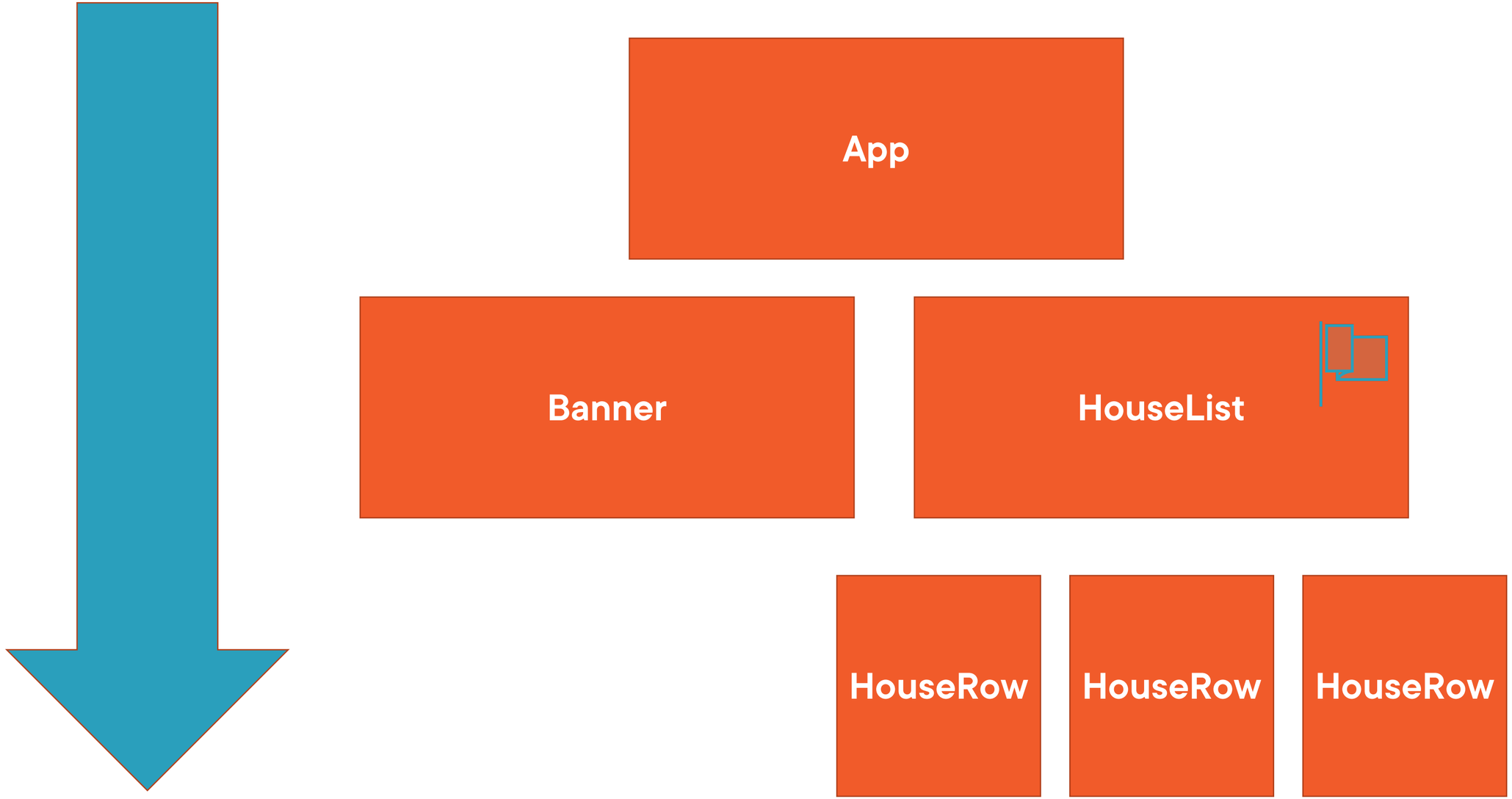
How React Keeps State

```
const [houses, setHouses] = useState(houses);
const [counter, setCounter] = useState(0);
```

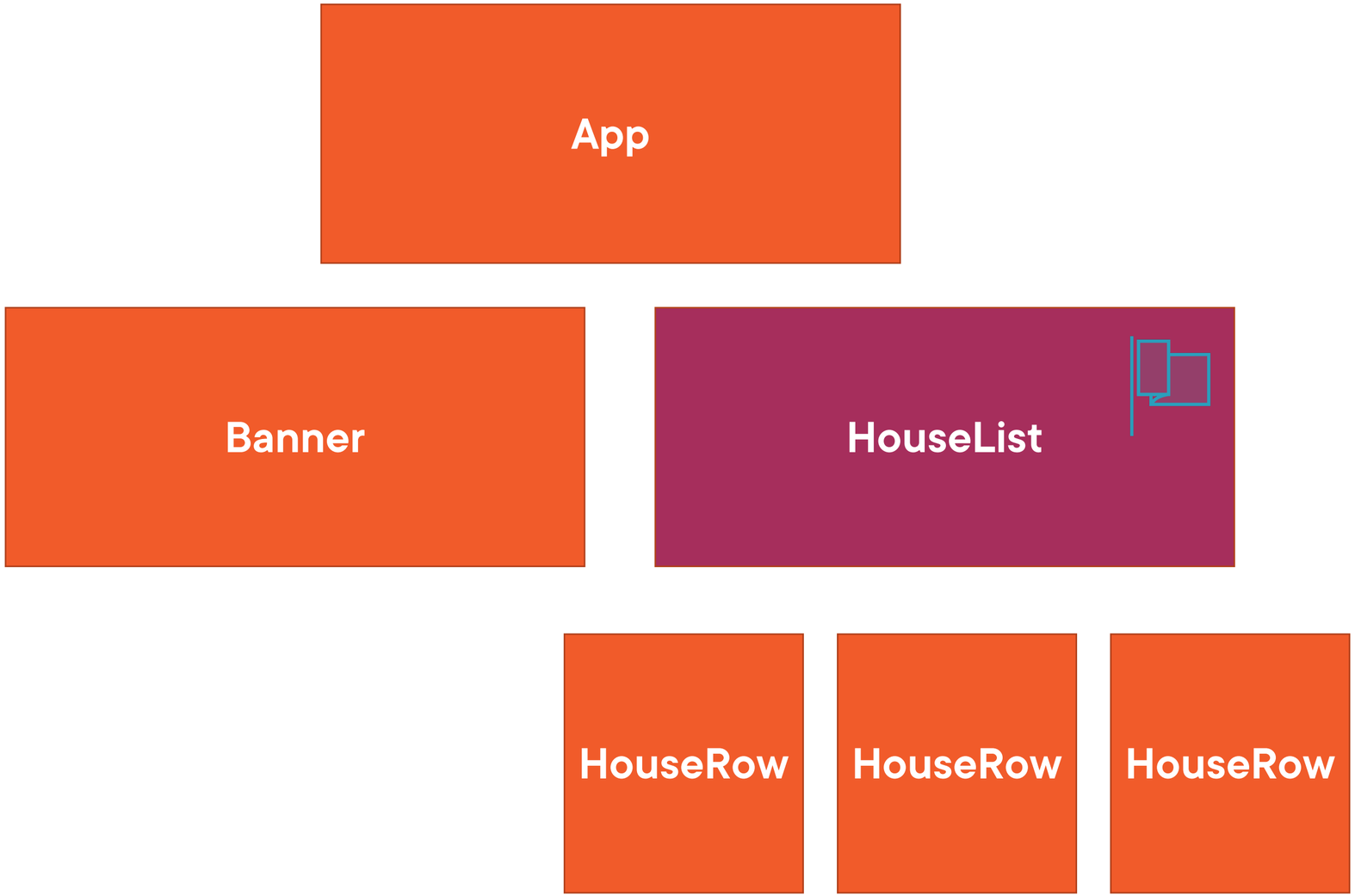
Array: [0]: [{..}, {

[1]: 42

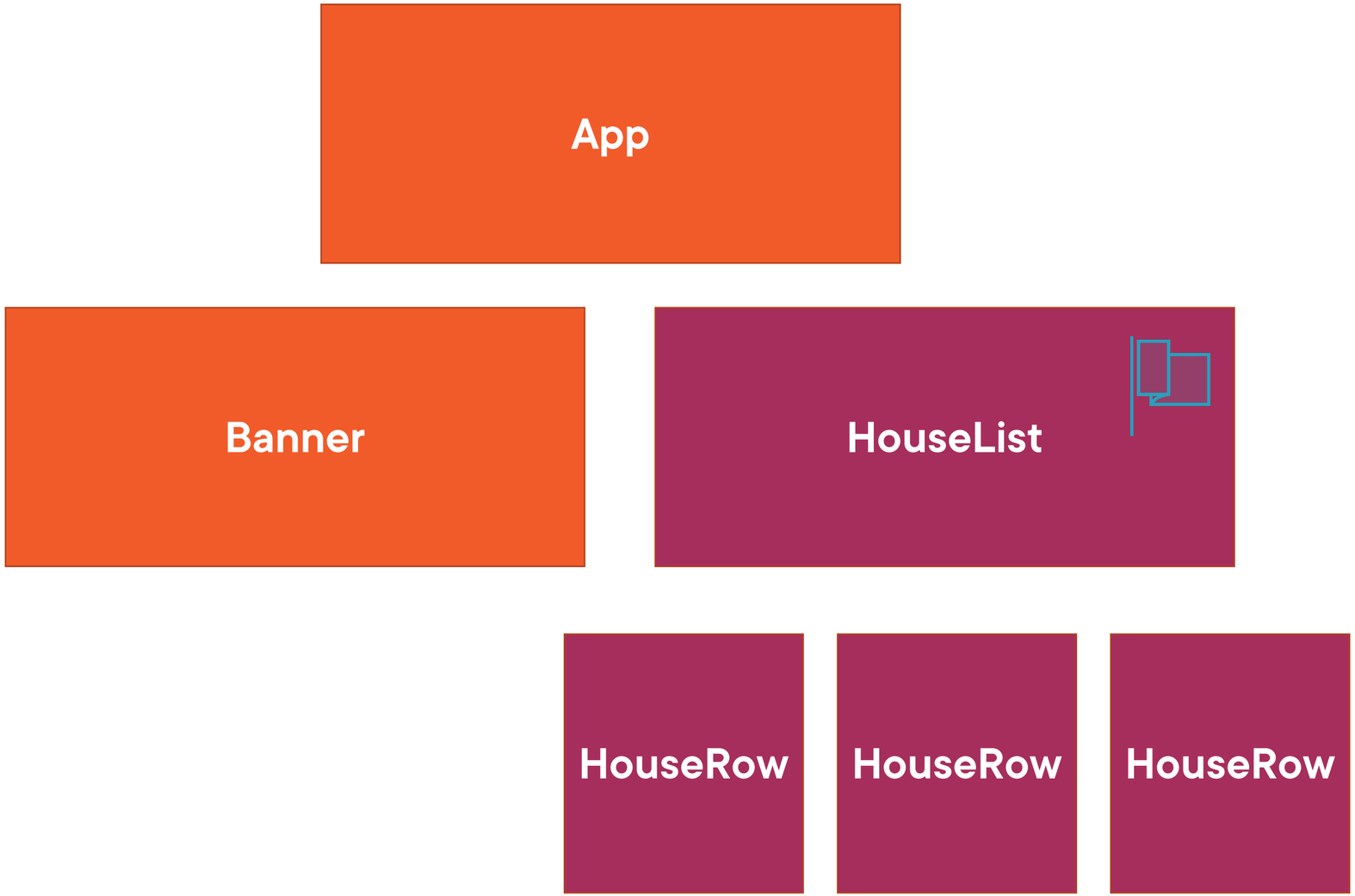
The Rendering Process



The Rendering Process



The Rendering Process



State Changes and Re-Renders

Cascading effect

Doesn't degrade performance in a well designed application

Remember reconciliation

Something to always keep in mind when designing the application

A Pure Function

```
const returnNumber = () => 42;
```

Another Pure Function

```
const add = (a, b) => a + b;
```

Pure Functions

Easy to test

Predictable

Reliable

Cacheable

Function components must be pure

When to Use React.memo

- When it's faster
- Measure
- Pure functional component
- Renders often
- With the same prop values
- JSX isn't trivial

https://reactjs.org/docs/react-api.html#reactmemo

Unpredictable operations in components should be set aside

(Side) effects

Effect Examples

API interaction

Use browser APIs (e.g. document, window)

Using timing functions (e.g. setTimeout)

The Effect Hook

```
useEffect(() => {
  //perform the effect
});
```

Effect Hook: Dependency Array

```
const [ counter, setCounter ] =
useState(0);

useEffect(() => {
  document.title = counter;
});
```

Effect Hook: Dependency Array

```
const [ counter, setCounter ] =
useState(0);

useEffect(() => {
  document.title = counter;
}, [ counter ]);
```

Optional exercise:
Persist the new house and extract the add button into a new component

Effect Hook: Separation of Concerns

```
useEffect(() => {
  document.title = counter;
}, [ counter ]);

useEffect(() => {
  //fetch from API
}, [ ]);
```

Effect Hook: Cleaning Up

```
useEffect(() => {
  //subscribe
  return () => {
    //unsubscribe
  };
}, [ ]);
```

Coming up:
The Memo and Ref hooks

Memo hook: Memoize values in components

The Memo Hook

```
const result =
  timeConsumingCalculation(houses)
;
```

The Memo Hook

```
const result = useMemo(() => {
  return
timeConsumingCalculation(houses);
}, [ houses ]);
```

Ref hook:
Persist values that survive re-renders
without causing a re-render

The Ref Hook

```
const TextInputWithFocusButton = () => {
  const inputEl = useRef(null);
  const onButtonClick = () => inputEl.current.focus();
  return (
  <>
    <input ref={ inputEl } type="text" />
    <button onClick={ onButtonClick }>Focus the input</button>
  </>
  );
};
```

Next up:

Conditional Rendering and Shared State