

Understanding How Diffing Algorithm Works

Virtual DOM / Behind the JSX / Batch /
Reconciliation / That's it.

Ham3d Esmaili

Just Another Software Engineer



@theham3d

How Diffing Algorithm Works

Today's Topics

Why DOM Manipulating Is Expensive?

- Let's Not Do It.

Virtual DOM Came On The Scene

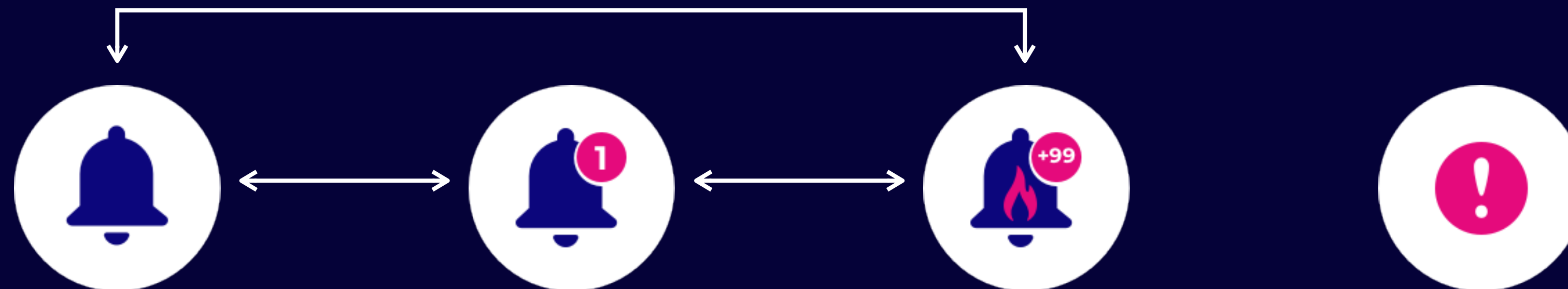
- Created to get rid of our headache.

React Reconciliation

- The real magic happens here.

DOM Manipulating In

Old Fashion



State Transition Complexity = $O(N^2 - N)$

DOM Manipulating Is Expensive

Let's **Not** Do It

Every time the DOM changes, the browser has to do two intensive operations:

- **Reflow**

Re-Calculate the layout of a portion of the page or the whole page layout.

- **Repaint**

Visual or content changes to an element that does not affect the layout and other elements.

Virtual DOM

Rebuild The Whole DOM, For Every Change !



Sounds Expensive

Just A Regular Javascript Object

We can manipulate it freely and frequently without touching the actual DOM until ...

```
class Counter extends Component {  
  ...  
  render() {  
    return [  
      <button key="1" onClick={...}>Count</button>,  
      <span key="2">{this.state.count}</span>  
    ]  
  }  
}
```

Fiber Nodes

Behind The JSX


```
class Counter extends Component {  
  ...  
  render() {  
    return [  
      <button key="1" onClick={...}>Count</button>,  
      <span key="2">{this.state.count}</span>  
    ]  
  }  
}
```



```
class Counter extends Component {  
  ...  
  render() {  
    return [  
      React.createElement(  
        'button',  
        {  
          key: '1',  
          onClick: this.onClick  
        },  
        'Count'  
      ),  
      React.createElement(  
        'span',  
        {  
          key: '2'  
        },  
        this.state.count  
      )  
    ]  
  }  
}
```

Behind JSX

Browsers have no clue about JSX and its syntax. Browsers only understand plain JavaScript, so JSX will be transformed into something else.

Fiber Nodes Are The Result Of

Create Element

<Button />

```
{
  $$typeof: Symbol(react.element),
  type: 'button',
  key: "1",
  props: {
    children: 'Count',
    onClick: () => { ... }
  },
  ...
}
```



```
{
  $$typeof: Symbol(react.element),
  type: 'span',
  key: "2",
  props: {
    children: '0',
  },
  ...
}
```

e t c
● ● ●

We Can Think Of The Group Of These Object As Our Virtual DOM Tree.

Rebuilding The DOM

The Real Magic Happens When We Wanna Update A Page
Important Part

Aka Reconciliation

Diffing Process

The process of **Updating** your **UI** to match your **Next** Application state

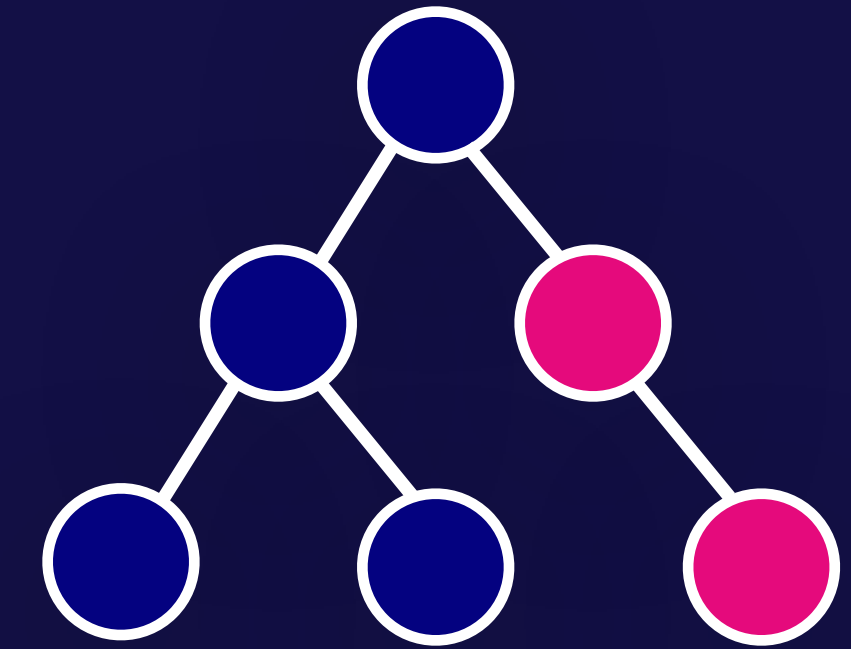
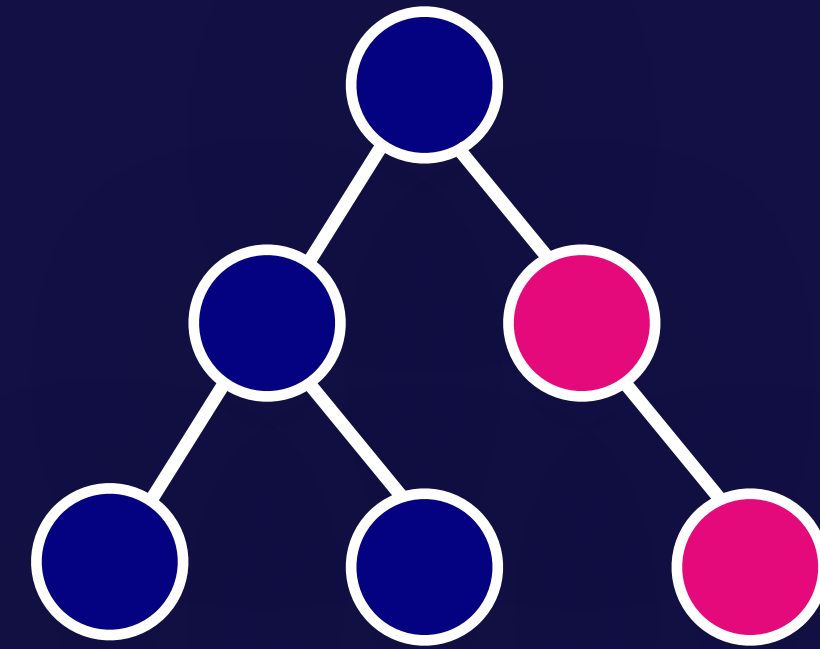
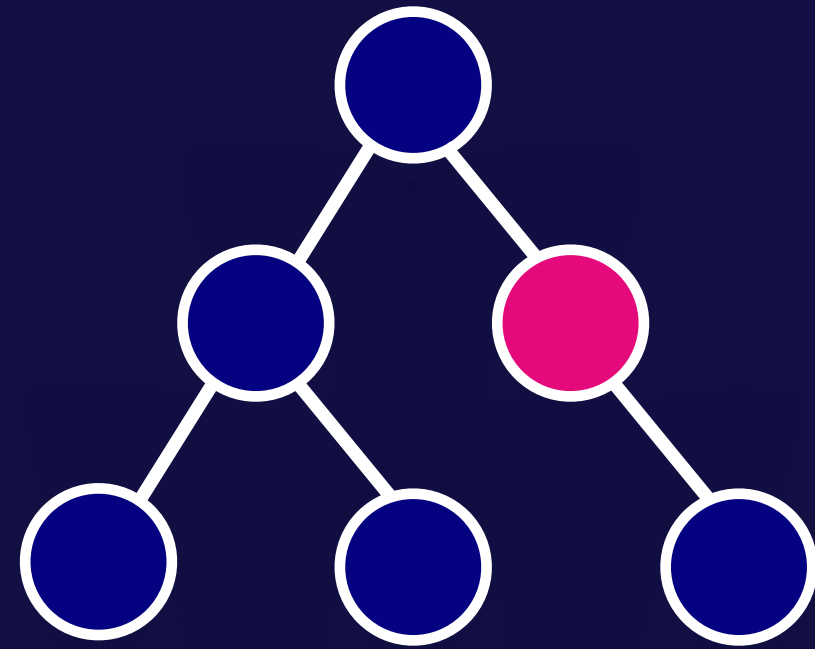
1. Work InProgress Tree

When React starts working on updates it builds a so-called **workInProgress** tree that reflects the future state to be flushed to the screen.

2. Reconciliation

React will start comparing the **current** tree and **workInProgress** tree to determine which parts of the node tree have to be updated and which can be left untouched.

Virtual DOM



State Change

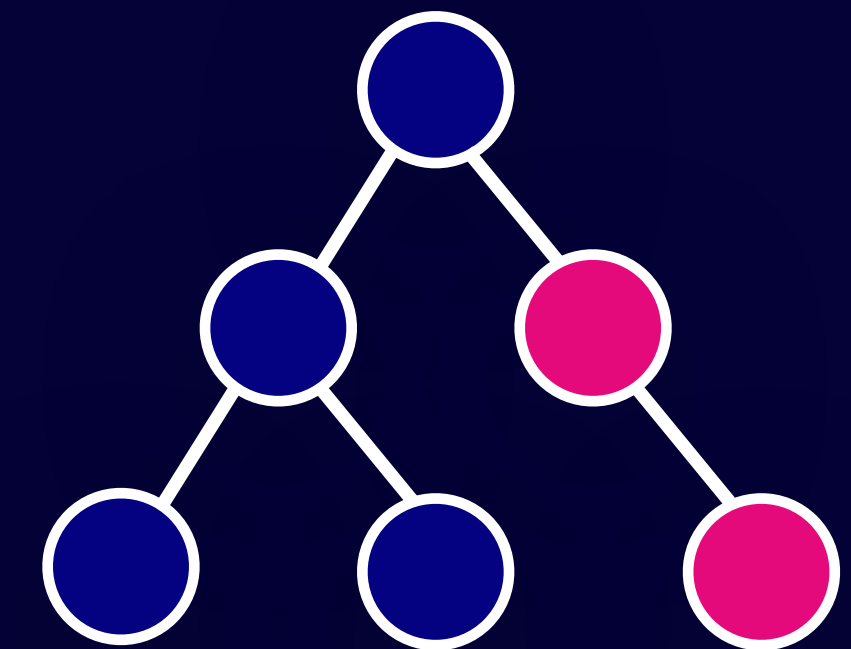
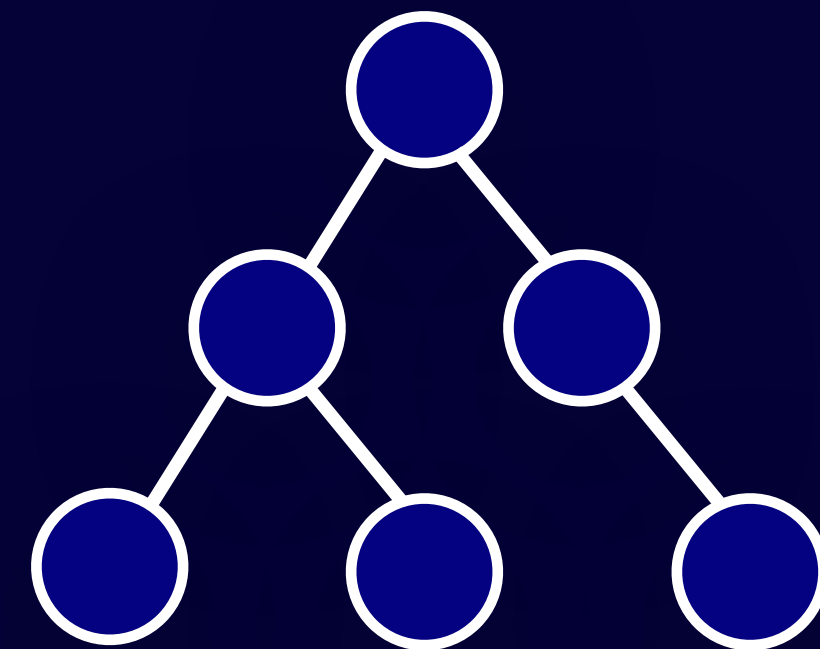
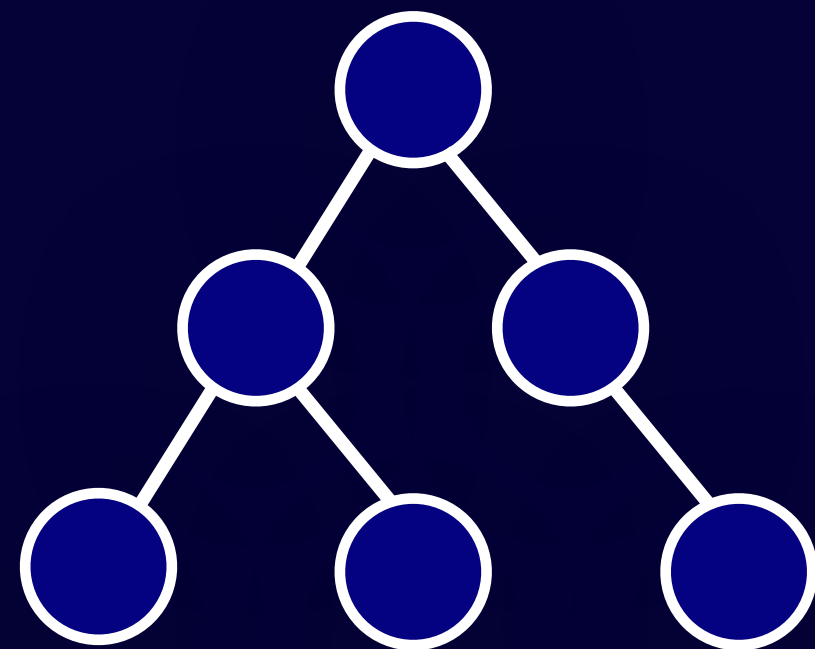


Compute Diff



Commit Changes

Browser DOM





React Complexity Diff

$O(N)$

Aka $O(ok)$



Truly Complexity Diff

$O(N^3)$

Aka $O(no)$

React Complexity Diff

How Possible

React relies on two assumptions to solve this problem in a linear time

1. Type

Two components of the same type will generate similar trees and vice versa.

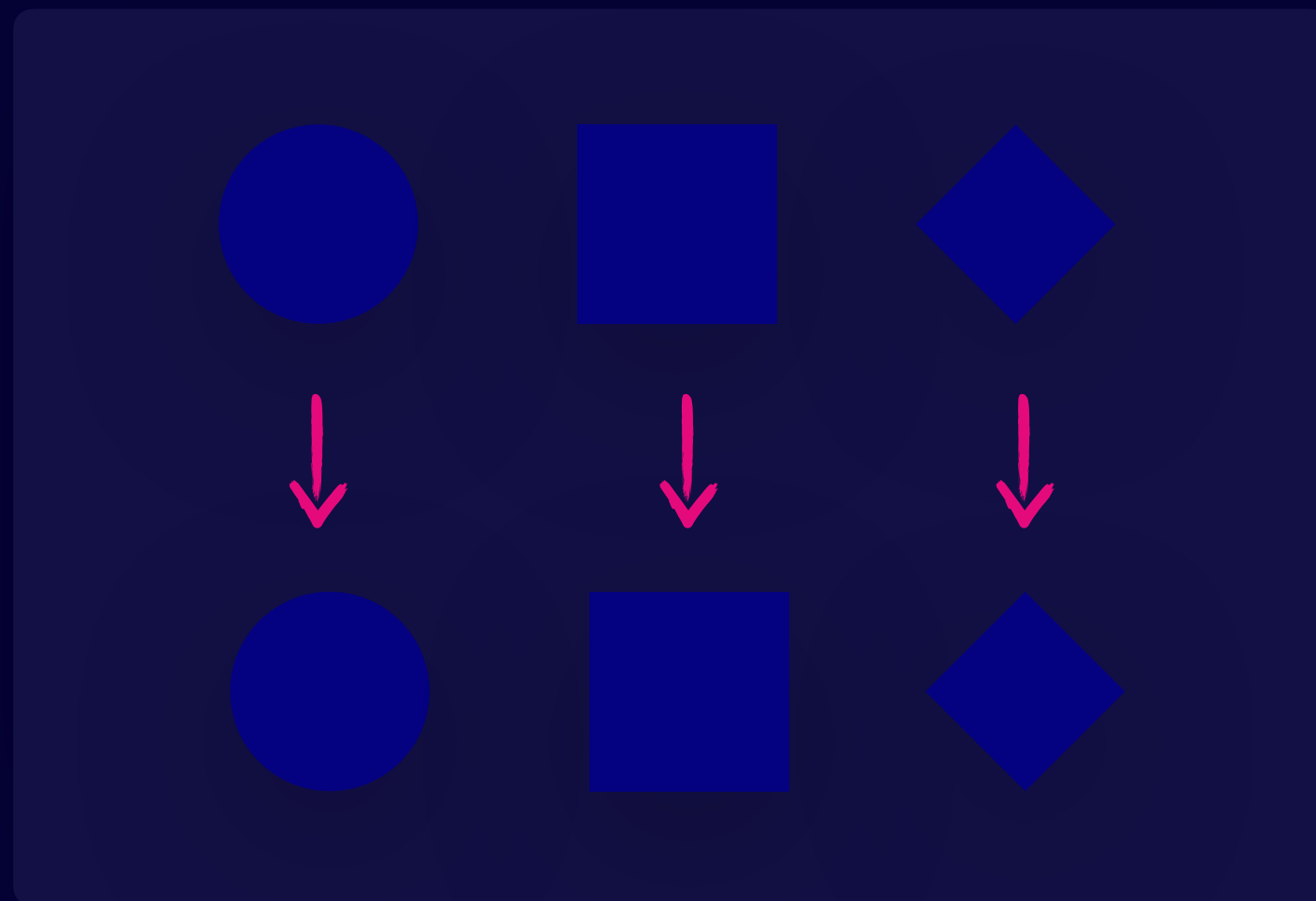
2. Key

We are able to provide a unique key for elements that is stable across different renders.

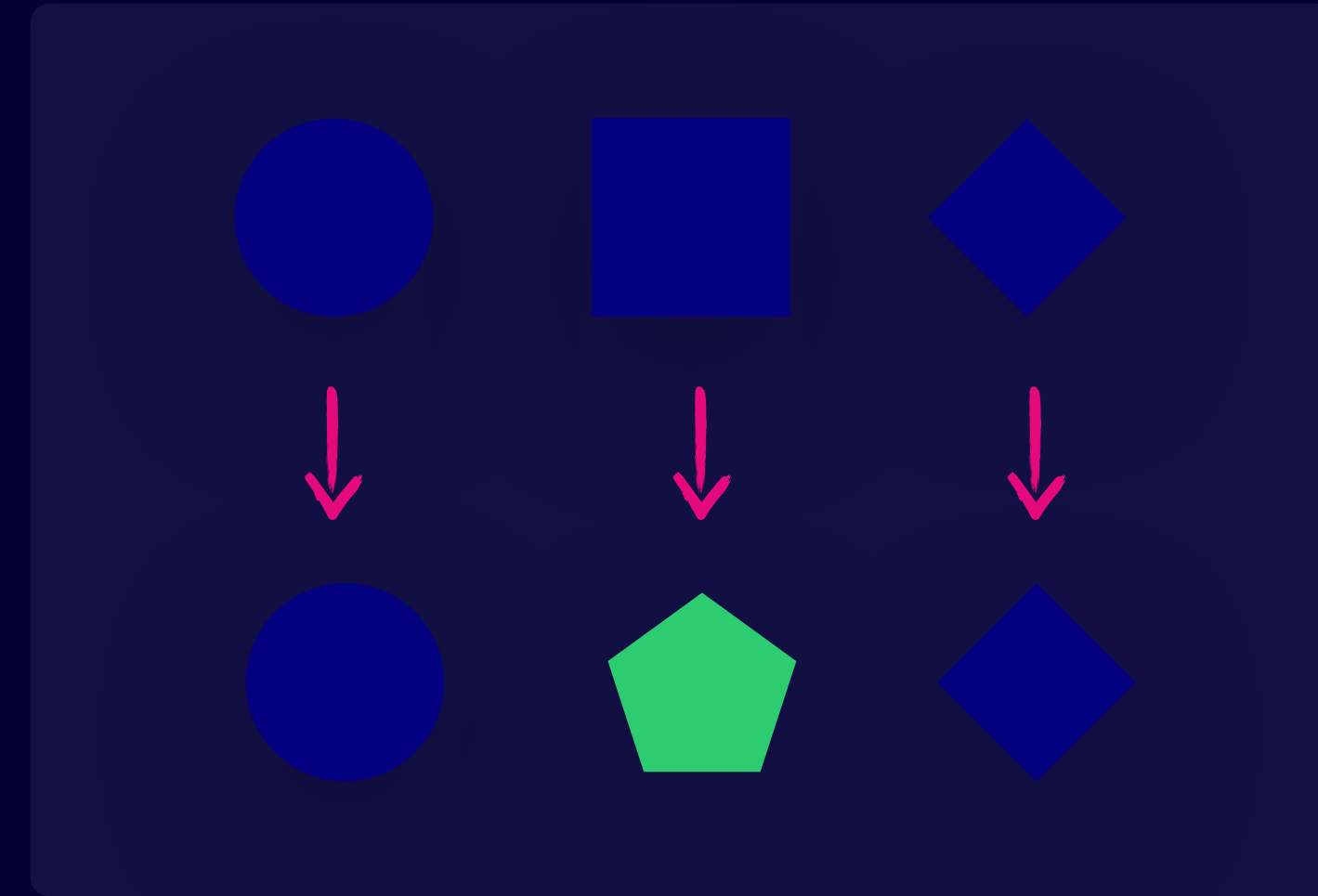
Checking Type

First Scenario

Same



Different



Checking Key

Second Scenario

Keys are about **Identity** in addition to **Performance**

Without Key

```
const CounterList = (props) => {
  return [
    <Counter />,
    <Counter />,
    <Counter />,
    <Counter />
  ];
}
```



New Item In 2nd Position

```
const CounterList = (props) => {
  return [
    <Counter />,
    <Counter />,
    <Counter />,
    <Counter />,
    <Counter />
  ];
}
```

Or

Remove Item In 2nd Position

```
const CounterList = (props) => {
  return [
    <Counter />,
    <Counter />,
    <Counter />
  ];
}
```

Checking Key

Second Scenario

Keys are about **Identity** in addition to **Performance**

With Key

```
const CounterList = (props) => {
  return [
    <Counter key="a" />,
    <Counter key="b" />,
    <Counter key="c" />,
    <Counter key="d" />
  ];
}
```



New Item In 2nd Position

```
const CounterList = (props) => {
  return [
    <Counter key="a" />,
    <Counter key="new" />,
    <Counter key="b" />,
    <Counter key="c" />,
    <Counter key="d" />
  ];
}
```

Or

Remove Item In 2nd Position

```
const CounterList = (props) => {
  return [
    <Counter key="a" />,
    <Counter key="c" />,
    <Counter key="d" />
  ];
}
```

Batched Updates

So The **Repaint** And **Reflow** The Browser Must Perform To
Render The Changes Are Executed Just **Once**

The **End**

Thank
You.
Ham3d
Esmaili

God Bless |     / **@theham3d**

