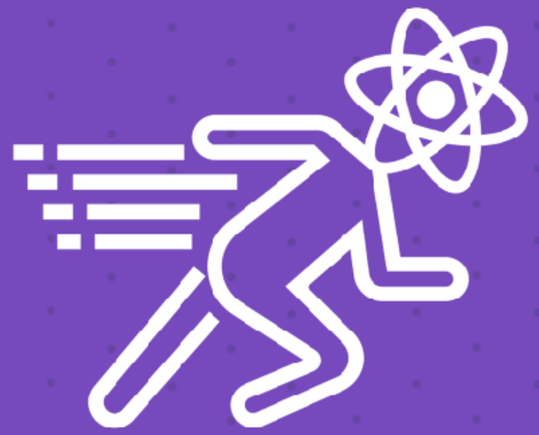


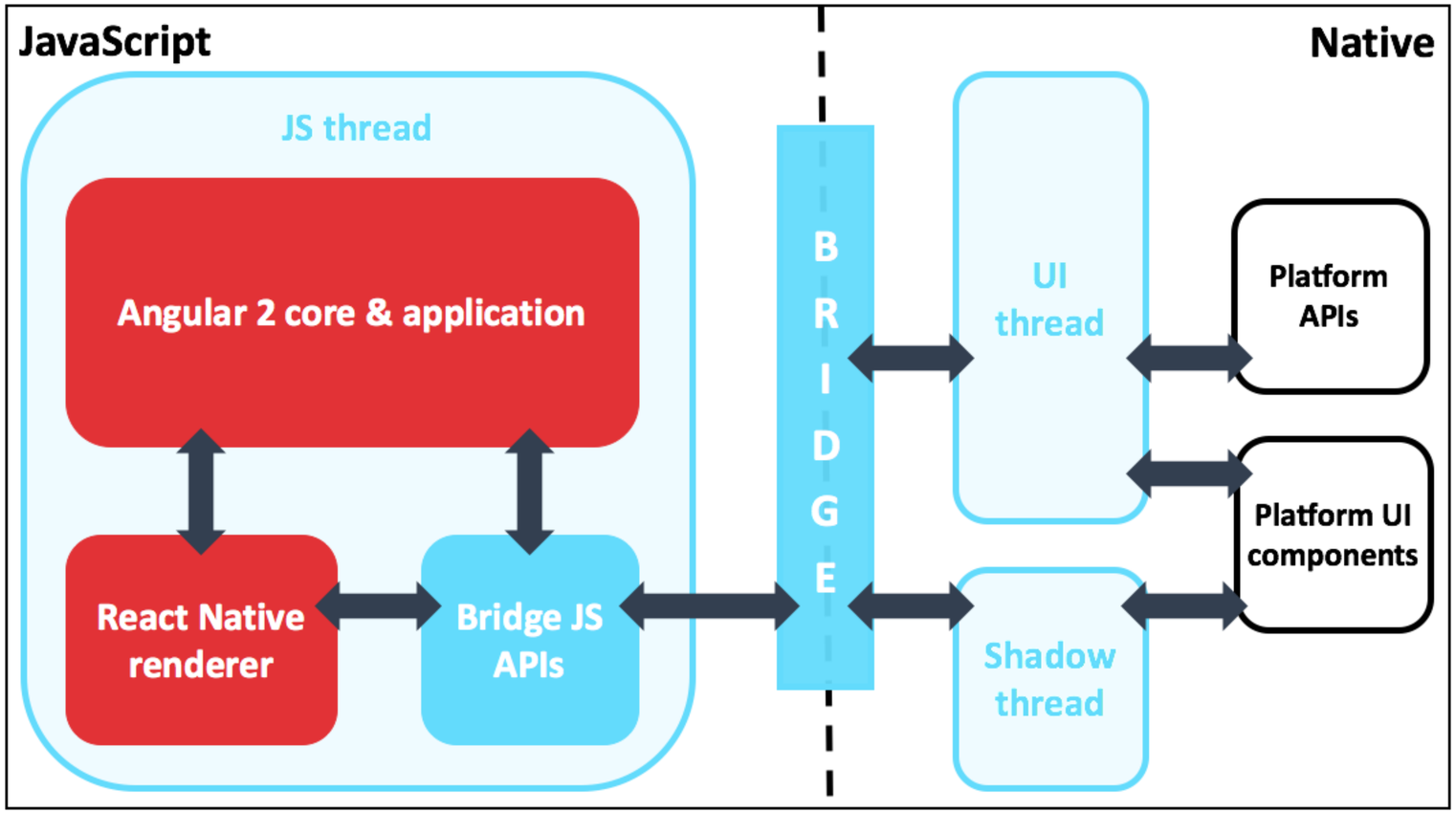
High performance React Native

Viet Tran - CTO Skylab

Agenda

- Is "React Native" native ?
- What is lagging, UI Main Thread ?
- Common mistakes.
- QA.

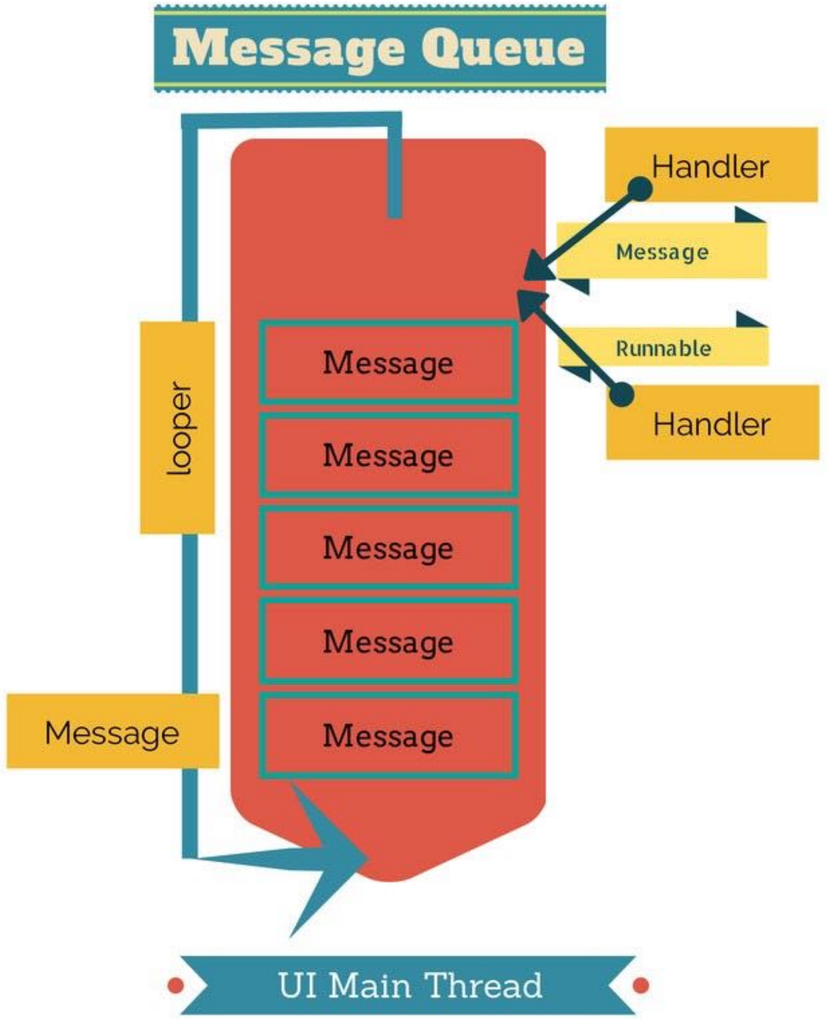
Is "React Native" native ?



NO

But what we get is **native**

What is lagging, UI Main Thread ?

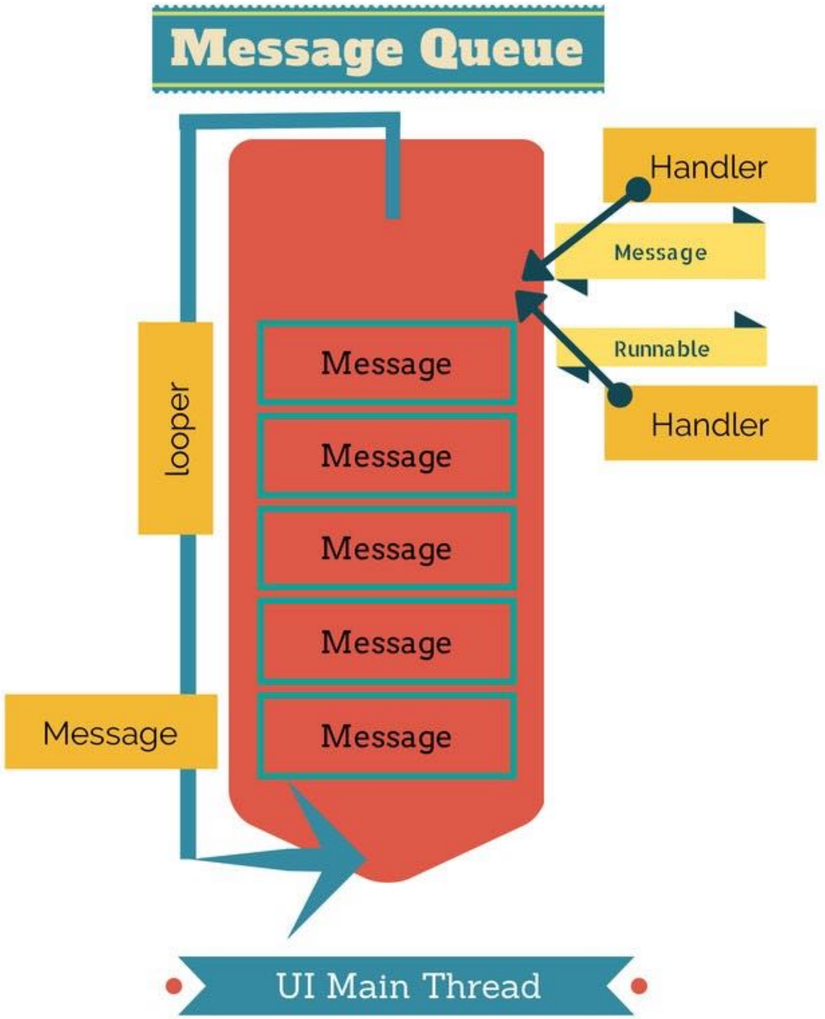


UI Main Thread ? (cont)

60 frames per second

= 60 loops per second

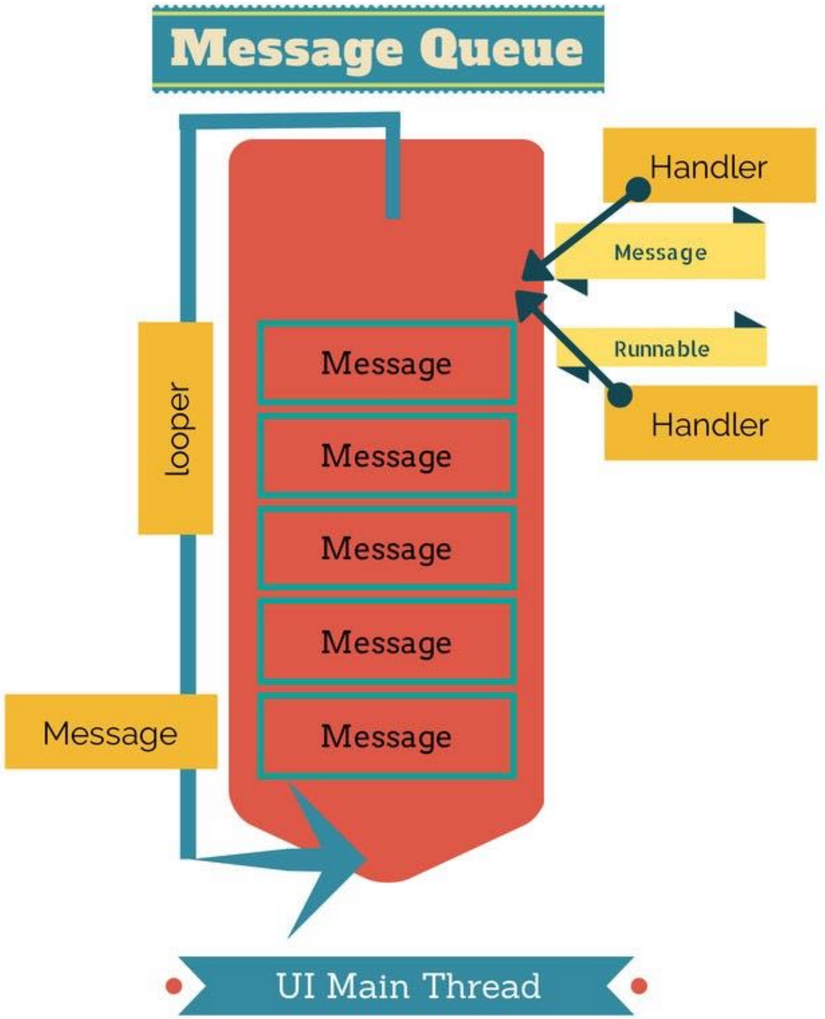
~ 16.67ms per loop



What is lagging ? (cont)

When some loops handle tasks lower 16.67ms

=> Low FPS (< 45 FPS)

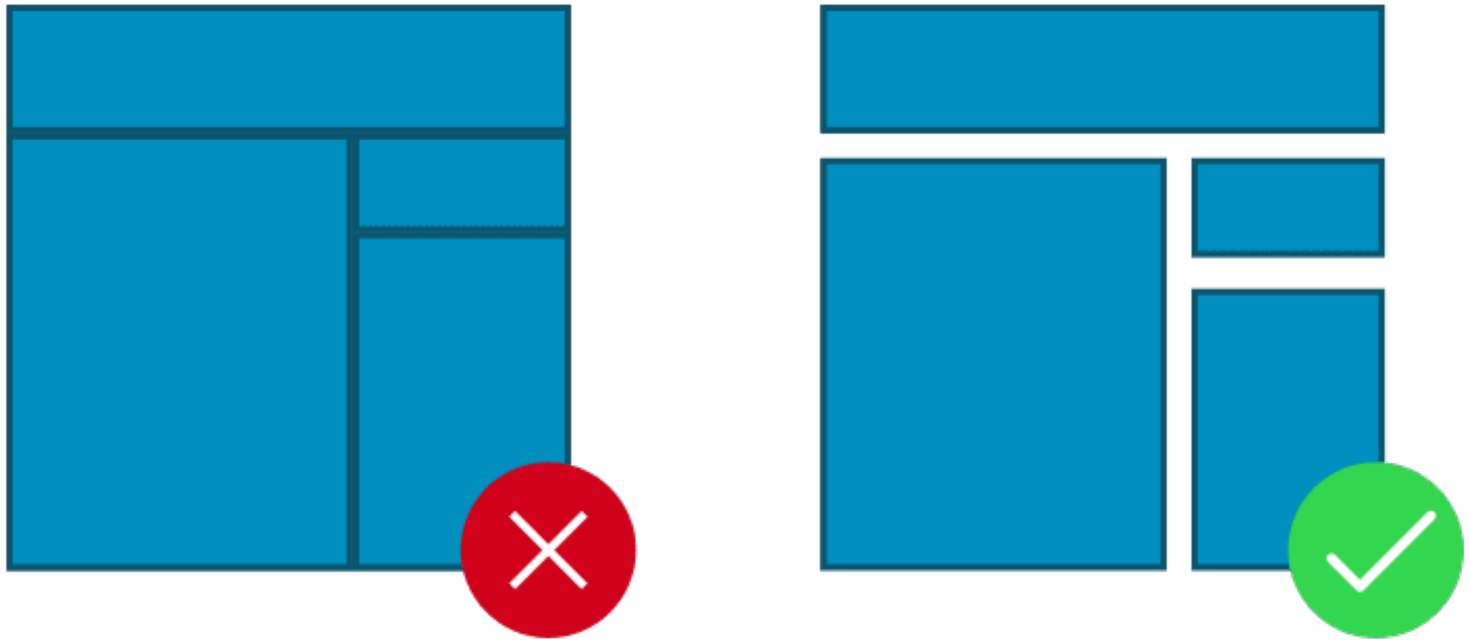


High performance is keeping the application running at least greater than 50FPS

Common mistakes

Massive component

Break your big components into smaller ones.



Massive component (cont)

```
1   class MyComponent extends Component {
2       render() {
3           const {todos, user} = this.props;
4           return (
5               <View>
6                 {user.name}
7                 <View>
8                     {todos.map(todo => <TodoView todo={todo} key={todo.id} />)}
9                 </View>
10              </View>
11          )
12      }
13  }
```

BAD

Massive component (cont)

```
1   class MyComponent extends PureComponent {
2       render() {
3           const {todos, user} = this.props;
4           return (<View>
5               {user.name}
6               <TodosView todos={todos} />
7           </View>)
8       }
9   }
10
11  class TodosView extends PureComponent {
12      render() {
13          const {todos} = this.props;
14          return (<View>
15              {todos.map(todo => <TodoView todo={todo} key={todo.id} />)}
16          </View>)
17      }
18  }
```

BETTER

Use anonymous function

```
1   render() {
2       return <MyWidget onClick={() => { alert('hi') }} />
3   }
```

Your component always re-render, even it's PureComponent

```
1   render() {
2       return <MyWidget onClick={this.handleClick} />
3   }
```

BETTER

Leak memory

```
componentDidMount() {
    this.timeout = setTimeout(() => this.setState({}), 1000)
}

componentWillUnmount() {
  clearTimeout(this.timeout)
}
```

Remove all listeners/schedulers when component unmount

Double render

```
1   async fetchAPI() {
2     const { isLoading } = this.state
3     if (isLoading) return
4
5     this.setState({ isLoading: true })
6     const data = await api('/products')
7     this.setState({ isLoading: false, data })
8   }
```

Your component might render multiple times at a moment

*** setState is a async method**

Double render (cont)

```
1  fetchAPI() {
2    const { isLoading } = this.state
3    if (isLoading) return
4
5    this.setState({ isLoading: true }, async () => {
6      const data = await api('/products')
7      this.setState({ isLoading: false, data })
8    })
9  }
```

BETTER

Re-render while pushing screen

```
componentDidMount() {
  this.timeout = setTimeout(() => this.fetchAPI(), 250) // or 300
}

fetchAPI() {
  const { isLoading } = this.state
  if (isLoading) return

  this.setState({ isLoading: true }, async () => {
    const data = await api('/products')
    this.setState({ isLoading: false, data })
  })
}
```

Animation duration: 250 - 300ms

User array index as keys

```
1   {todos.map((todo, index) =>
2     <Todo data={todo} key={index} />
3   )}
```

BAD

```
1   {todos.map((todo) =>
2     <Todo todo={todo} key={todo.id} />
3   )}
```

BETTER

Massive api calling

```
1  inputTextChanged = (text) => {
2    this.setState({ text }) // call massively
3  }
4
5  searchFromApi = (keyword) => {
6    // call api and setState response data
7  }
```

BAD

```
1  import debounce from 'lodash.debounce'
2
3  inputTextChanged = (text) => {
4    this.setState({ text }) // call massively
5  }
6
7  searchFromApi = debounce((keyword) => {
8    // call api and setState response data
9  }, 400)
```

BETTER

Heavy computation

```
1  travelNodes = (root) => new Promise((resolve, reject) => {
2    if (root.children) {
3      return Promise.all(root.children.map((n) => {
4        return this.travelNodes(n)
5      }))
6    }
7
8    return resolve(0)
9  })
```

This might block JS Thread for a while

Heavy computation (cont)

```
1   nextFrame = () => new Promise(resolve => requestAnimationFrame(resolve))
2
3   travelNodes = (root) => new Promise(async (resolve, reject) => {
4     await nextFrame()
5
6     if (root.children) {
7       return Promise.all(root.children.map((n) => {
8         return this.travelNodes(n)
9       }))
10    }
11
12    return resolve(0)
13  })
```

Move each step to next frame animation to reduce blocking

Some other ways

Some other ways

- Use native animation as much as possible.
- Use native module to reduce bridge call.
- Use local state (we don't need Redux/MobX for all cases).
- Caching some resources if needed: images, meta data.
- Divide components to smart and dump components.

High Performance Libraries

- https://github.com/wix/react-native-navigation
- https://github.com/wix/react-native-interactable
- https://github.com/react-community/lottie-react-native
- https://github.com/DylanVann/react-native-fast-image

QA

Thank you !

Viet Tran - CTO Skylab