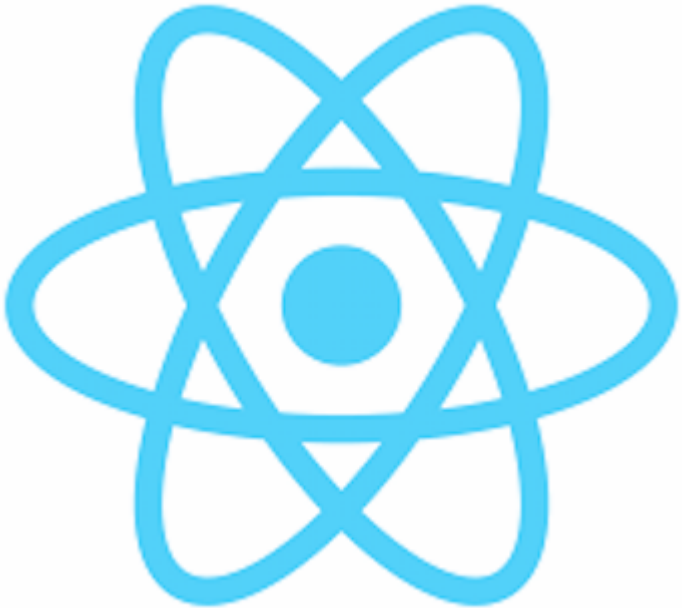
# Get hooked on React hooks

Mario Fernandez

# Where is React today?

# Times They Are a-Changin





# What are hooks?

> *Hooks* are a new addition in **React 16.8**. They let you use state and other React features without writing a class.

# Hooks are just functions

# useState

```
import React, { useState } from 'react'

const Counter = () => {
  const [count, setCount] = useState<number>(0)

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  )
}
```

```
class Counter extends React.Component {
  constructor(props) {
    super(props)
    this.state = { count: 0 }
  }

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button onClick={() => this.setState({count:this.state.count+1})
          Click me
        </button>
      </div>
    )
  }
}
```

https://dev.to/dan_abramov/making-sense-of-react-hooks-2eib

# Why hooks?

# Functional components over classes

# Encapsulate stateful logic

# Does anybody truly understand lifecycle methods?

# useEffect

perform side effects

```
import React, { useEffect } from 'react'

const Counter = ({ count }: { count: number }) => {
  useEffect(() => {
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
    </div>
  )
}
```

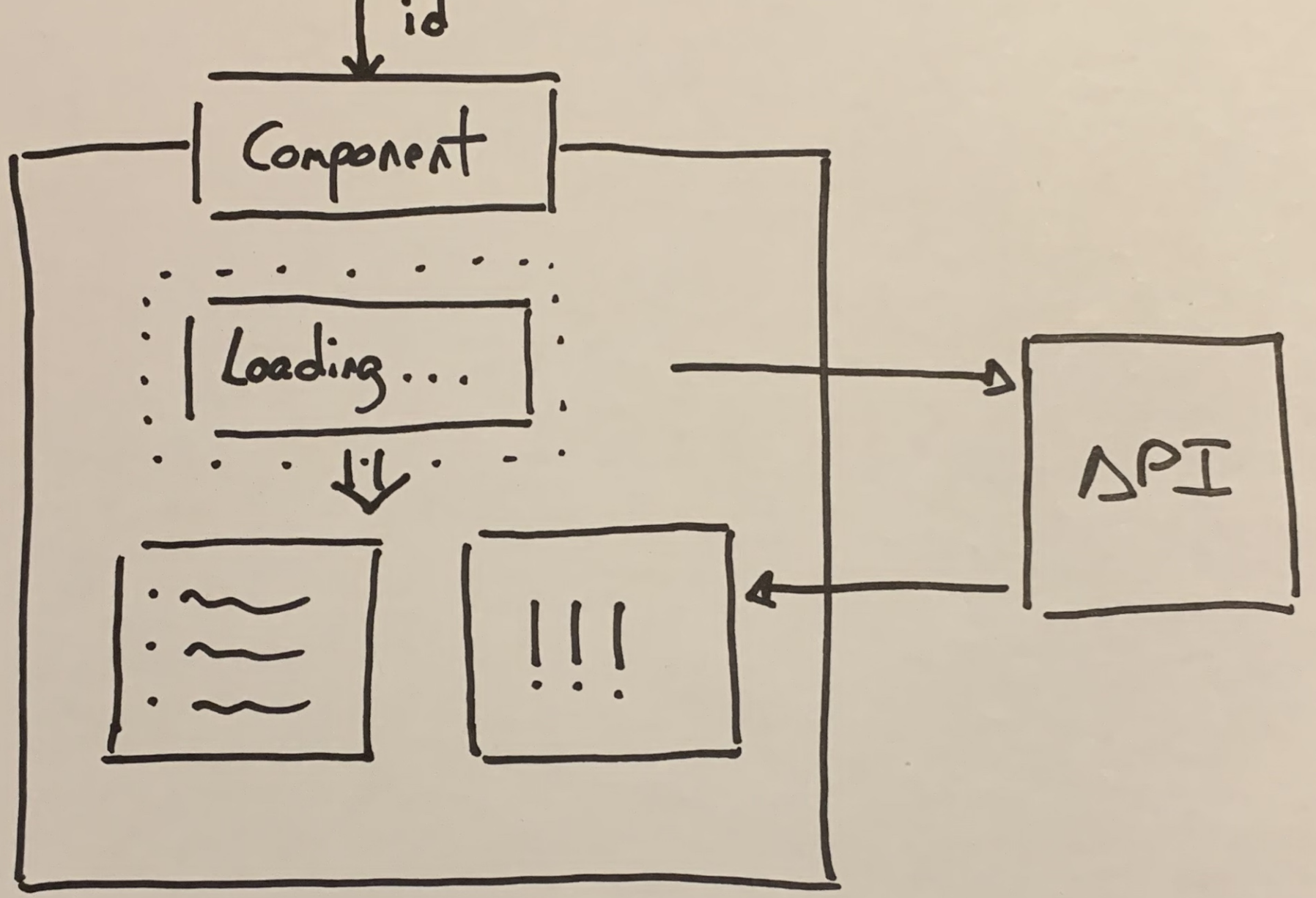
```
// similar to componentDidMount
useEffect(fn, []);
```

```
// similar to componentDidMount + componentDidUpdate
useEffect(fn, [counter]);
```

```
// similar to componentWillUnmount
useEffect(() => {
  return unmountFn
}, [counter]);
```

https://overreacted.io/a-complete-guide-to-useeffect/

# Case Study: REST populated component


id
Component
Loading...
API

```
const Stuff = ({ id }: { id: number }) => {
  const [error, setError] = useState<boolean>(false)
  const [data, setData] = useState<string | undefined>()

  useEffect(() => {
    const fetchData = async () => {
      try {
        setError(false)
        const result = await axios(`/route/${id}`)
        setData(result)
      } catch (e) { setError(true) }
    }

    fetchData()
  }, [id])

  ...
}
```

```
const Stuff = ({ id }: { id: number }) => {

  ...

  return (
    <>
      {error && 'Oh noooo'}
      {data && <p>{data}</p>}
    </>
  )
}
```

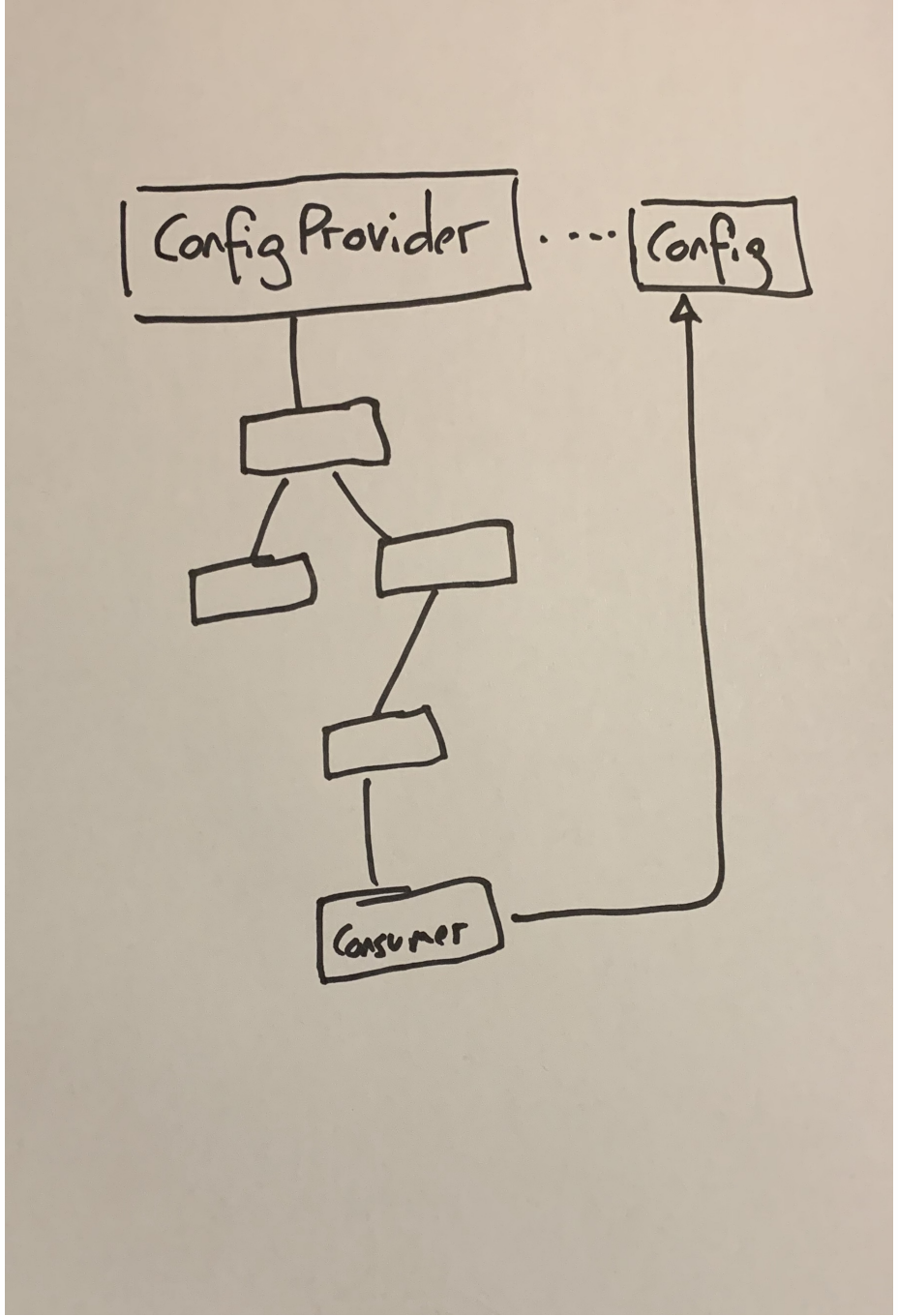
https://github.com/streamich/react-use

```
import { useAsync } from 'react-use'

const Stuff = ({ id }: { id: number }) => {
  const { loading, error, value } = useAsync(async () => {
    const result = await axios(`/route/${id}`)
    return result
  }, [id])

  return (
    <>
      {loading && 'Loading...'}
      {error && 'Oh noooo'}
      {value && <p>{value}</p>}
    </>
  )
}
```

# Case Study: Configuration


Config Provider
Config
Consumer

https://reactjs.org/docs/context.html

```
const ConfigContext = React.createContext<ConfigType>(
  initConfig
)
```

```
const Config = (props: Props) => {
  const state = useAsync(async (): Promise<ConfigType> => {
    return await configuration()
  }, [])

  const { loading, value } = state
  return (
    <ConfigContext.Provider value={value || initConfig}>
      {!loading && props.children}
    </ConfigContext.Provider>
  )
}
```

```
const useConfig = () =>
  useContext<ConfigType>(ConfigContext)
```

# Consuming context

```
const ConfigClient = () => {
  const { country } = useConfig()
  return (
    <>
      Country from the config: {country}
    </>
  )
}
```

# The old way using a HOC

```
const withConfig = <P extends object>(
  WrappedComponent: React.ComponentType<P & WithConfigProps>
) => {
  class WithConfig extends Component<
    Pick<P, Exclude<keyof P, keyof WithConfigProps>>
  > {
    static contextType = ConfigContext
    render() {
      const config: Config = this.context
      const props = { config, ...(this.props as P) }
      return <WrappedComponent {...props} />
    }
  }

  return WithConfig
}
```

# When all you have is a hook ...

```
const auth = useMemo<ContextType>(
  () => ({
    user,
    login,
    logout: () => logout(setUser),
    checkLogin: () => checkLogin(setUser)
  }),
  [user]
)
```

```
const AddressForm = ({ index }: Props) => {
  const { formatMessage } = useIntl()
  return (
    <div className={styles.body}>
      <Input
        name={`addresses.${index}.street`}
        label={formatMessage({ id: 'address.street' })}
      />
    </div>
  )
}
```

# That's kinda cool, huh?

# Consistency

# Avoid props pollution

# The elephant in the room

# useReducer

https://www.robinwieruch.de/react-usereducer-hook

# Custom Hooks

```
const useFieldValues = () => {
  const { locale } = useIntl()
  const { value, loading, error } = useAsync(
    async () => await getFieldValues(locale)
  )
  return { fieldValues: value, loading, error }
}
```

# Rules of Hooks

# Only Call Hooks at the Top Level

# Only Call Hooks from React Functions

```
{
  "plugins": [
    // ...
    "react-hooks"
  ],
  "rules": {
    // ...
    "react-hooks/rules-of-hooks": "error", // Checks rules of Hooks
    "react-hooks/exhaustive-deps": "warn" // Checks effect dependencies
  }
}
```

# Are we going to test this?

```
import { waitForElement, render } from '@testing-library/react'
import RecipeDetails from './RecipeDetails'

jest.mock('recipe-details/recipeDetails.service')

describe('RecipeDetails', () => {
  it('renders correctly', async () => {
    const { getByText } = render(<RecipeDetails id={1} />)
    await waitForElement(() => getByText('Pasta Carbonara'))
  })
})
```

```
import { renderHook, act } from '@testing-library/react-hooks'
import useCounter from './useCounter'

test('should increment counter', () => {
  const { result } = renderHook(() => useCounter())

  act(() => {
    result.current.increment()
  })

  expect(result.current.count).toBe(1)
})
```

# Are hooks ready for prime time?

**Yes!**

# Links

- https://reactjs.org/docs/hooks-intro.html
- https://wattenberger.com/blog/react-hooks
- https://www.robinwieruch.de/react-hooks-fetch-data