The Future of React

Asynchronous Components

About me

Dominic Langenegger

- Studied Computer Science at ETH Zurich, Switzerland
- Moved to Singapore in October 2017

Software Engineer @ Zuhlke Engineering Pte Ltd

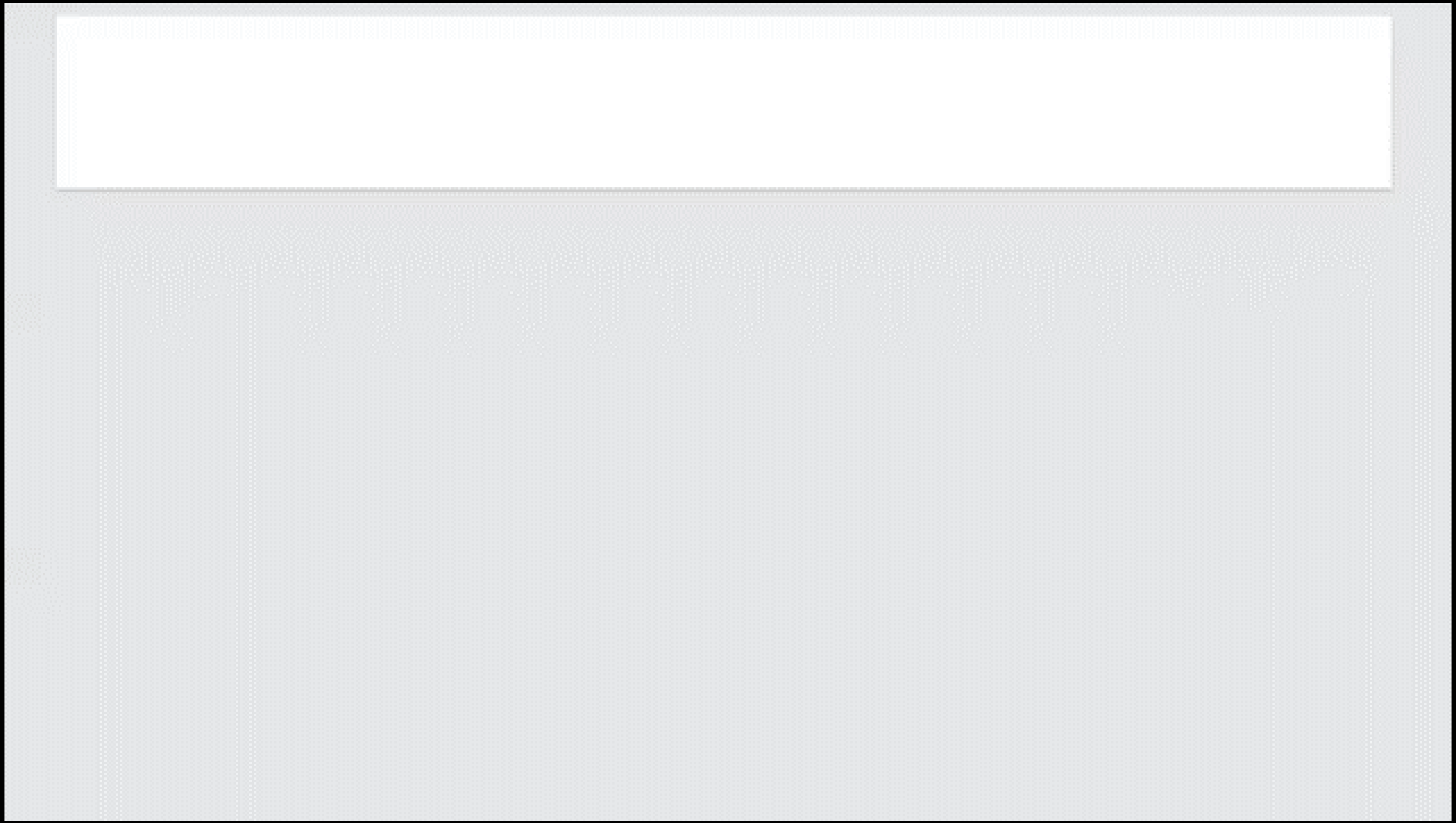
- Service provider and solution partner
- 1000 employees, mostly in Europe

zühlke
empowering ideas

What is new?

- Context API
- createRef() API
- Lifecycle Methods
- Time slicing
- Suspense API

Why?



Disclaimer

No, Really, It Is Unstable
The API ~~may~~ will change wildly between versions.

README.md https://github.com/facebook/react/simple-cache-provider

Dan Abramov
JSConf 2018

With vast differences in computing power and network speed, how do we deliver the best user experience for everyone?

Computing Power

Creating nodes

Re-rendering

Network Speed

Data fetching

Code splitting

CPU heavy tasks

Updating complicated view on changed input

- live list filter
- graph rendering
- …

Current Solution:

Debounce rendering

But fast devices will also be slowed down by this…

Dan Abramov

We’ve built a generic way to ensure that high-priority updates like user input don’t get blocked by rendering low-priority updates.

Time Slicing

- React doesn’t block the thread while rendering
- Feels synchronous if the device is fast
- Feels responsive if the device is slow
- Only the final rendered state is displayed
- Same declarative component model

Computing Power

Creating nodes

Re-rendering

Network Speed

Data fetching

Code splitting

Dan Abramov

We've built a generic way for components to suspend rendering while they load asynchronous data.

Demo Time

Suspense API

- Pause any state update until the data is ready
- Add async data to any component without “plumbing”
- On a fast network, render after the whole tree is ready
- On a slow network, precisely control the loading states
- There’s both a high-level and a low-level API

Async Rendering with React

- Adapt to user’s device and network
- Fast interactions feel instant
- Slower interactions feel responsive

Links

Official Blogposts

- Sneak Peek Beyond React 16

  https://reactjs.org/blog/2018/03/01/sneak-peek-beyond-react-16.html

- Update on Async Rendering

  https://reactjs.org/blog/2018/03/27/update-on-async-rendering.html

Talks on the topic

- Beyond React 16 @ JSConf Iceland, 2018

  https://www.youtube.com/watch?v=nLF0n9SACd4

- Suspense! @ ReactFest London, 9 March 2018

  https://www.youtube.com/watch?v=6g3g0Q_XVb4

Demo project by Andrew Clark

https://codesandbox.io/s/5zk7x551vk

Resources

All code and slides are on Github:

https://github.com/dola/react-future

Try it yourself?

```
npm install react@16.4.0-alpha.0911da3 react-dom@16.4.0-alpha.0911da3
```

Suspense API

- Intentional loading states
- No boilerplate code
- No race conditions

How?

Request Idle Callback API

In the same way that adopting `requestAnimationFrame` allowed us to schedule animations properly and maximize our chances of hitting 60fps, `requestIdleCallback` will schedule work when there is free time at the end of a frame, or when the user is inactive.

https://developers.google.com/web/updates/2015/08/using-requestidlecallback

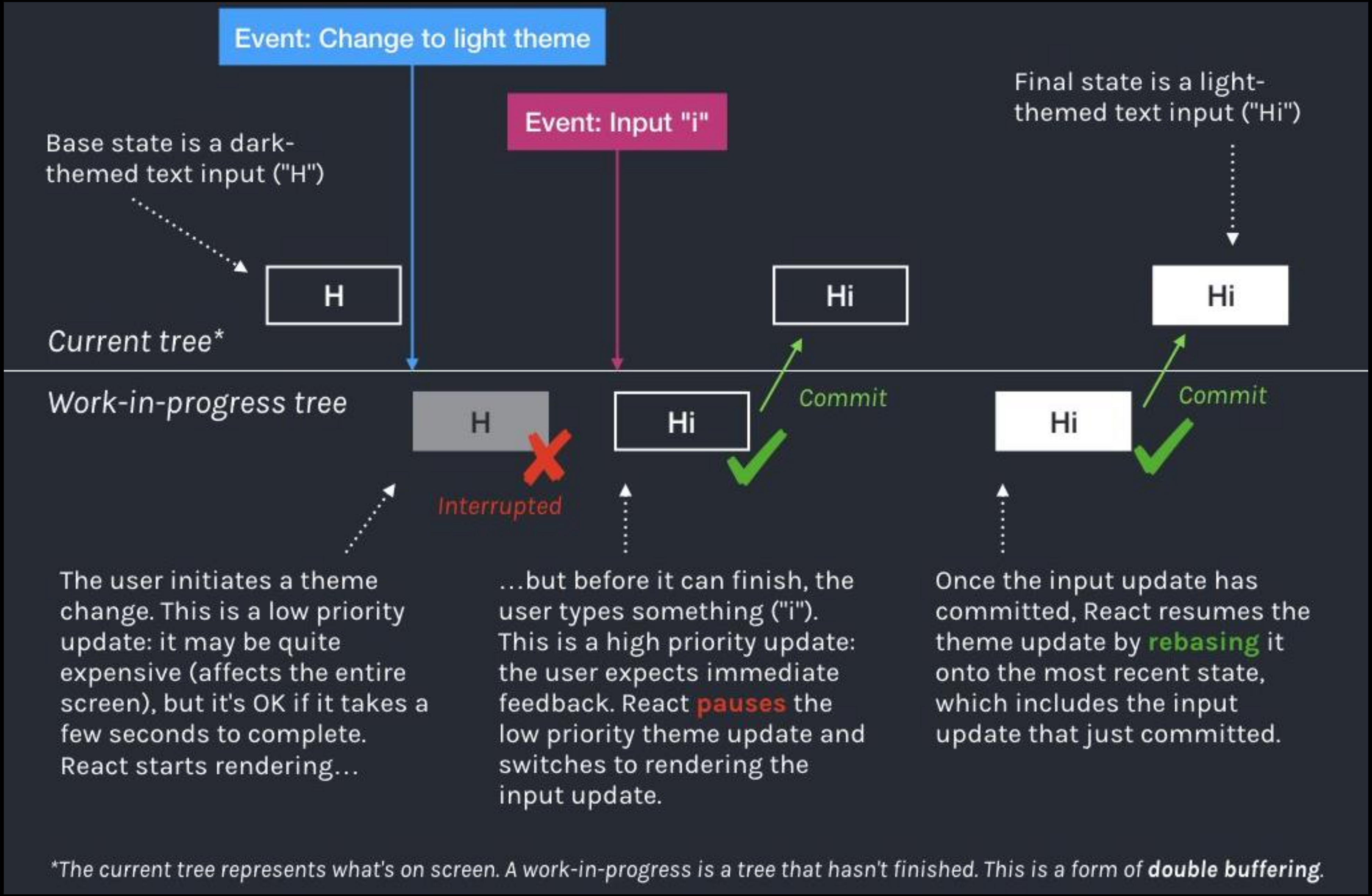
How?

Throw a Promise?

If a value is not in the cache, throw a Promise which an error boundary catches and awaits

After the Promise resolves, the rendering can restart where it left off

Andrew Clark https://twitter.com/acdlite/status/969171217356746752


Event: Change to light theme
Event: Input "i"
Base state is a dark-themed text input ("H")
Final state is a light-themed text input ("Hi")
H
Hi
Hi
Current tree*
Work-in-progress tree
H
Interrupted
Hi
Commit
Hi
Commit

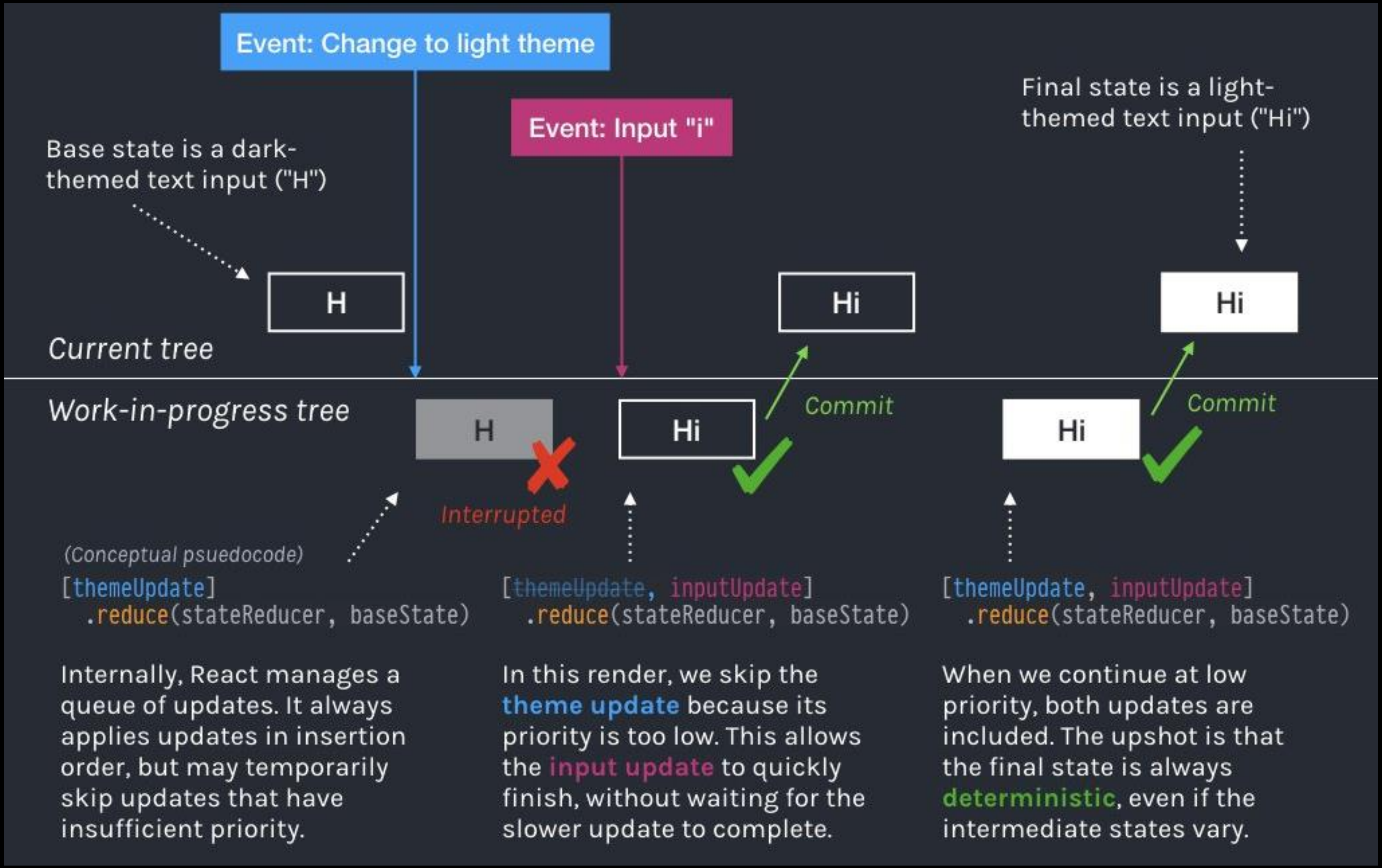
The user initiates a theme change. This is a low priority update: it may be quite expensive (affects the entire screen), but it's OK if it takes a few seconds to complete. React starts rendering...

...but before it can finish, the user types something ("i"). This is a high priority update: the user expects immediate feedback. React **pauses** the low priority theme update and switches to rendering the input update.

Once the input update has committed, React resumes the theme update by **rebasing** it onto the most recent state, which includes the input update that just committed.

The current tree represents what's on screen. A work-in-progress is a tree that hasn't finished. This is a form of* ***double buffering**.*

https://twitter.com/acdlite/status/978412930973687808


Event: Change to light theme
Event: Input "i"
Base state is a dark-themed text input ("H")
Final state is a light-themed text input ("Hi")
H
Hi
Hi
Current tree
Work-in-progress tree
H
Interrupted
Hi
Commit
Hi
Commit

(Conceptual psuedocode)

```
[themeUpdate]
  .reduce(stateReducer, baseState)
```

```
[themeUpdate, inputUpdate]
  .reduce(stateReducer, baseState)
```

```
[themeUpdate, inputUpdate]
  .reduce(stateReducer, baseState)
```

Internally, React manages a queue of updates. It always applies updates in insertion order, but may temporarily skip updates that have insufficient priority.

In this render, we skip the **theme update** because its priority is too low. This allows the **input update** to quickly finish, without waiting for the slower update to complete.

When we continue at low priority, both updates are included. The upshot is that the final state is always **deterministic**, even if the intermediate states vary.

https://twitter.com/acdlite/status/978412930973687808

New Lifecycle Methods

- static getDerivedStateFromProps(nextProps, prevState)
- getSnapshotBeforeUpdate()

Deprecated:

- componentWillMount
- componentWillReceiveProps
- componentWillUpdate

https://reactjs.org/blog/2018/03/29/react-v-16-3.html