React.js Concurrent Mode

Gogoro Software Billing Team

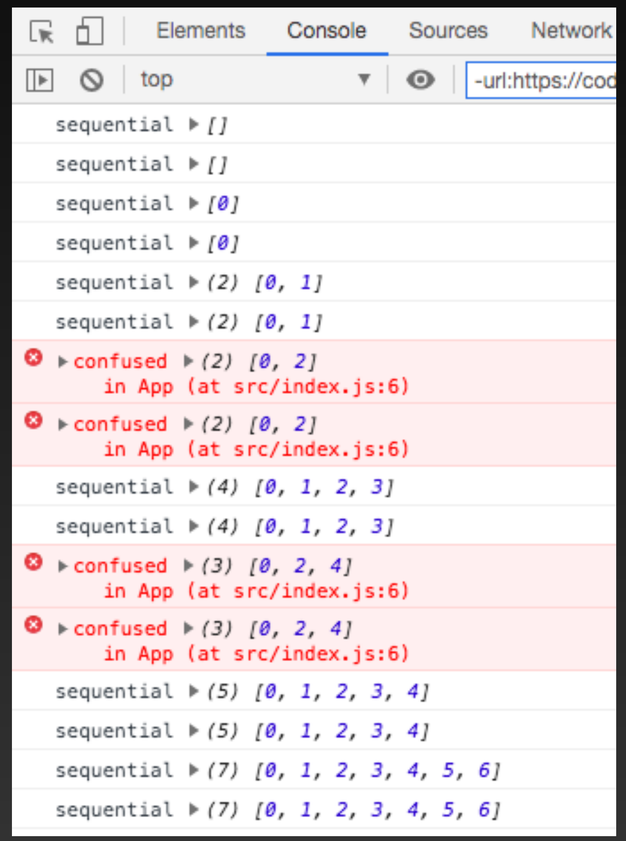
Jimmy Liao 2021

React.js Concurrent Mode

- Introducing Concurrent Mode
- Suspense for Data Fetching
- Concurrent UI Patterns
- Adopting Concurrent Mode
- My Example by Concurrent Mode

Concurrent Mode is experimental features

Concurrent Mode is experimental features

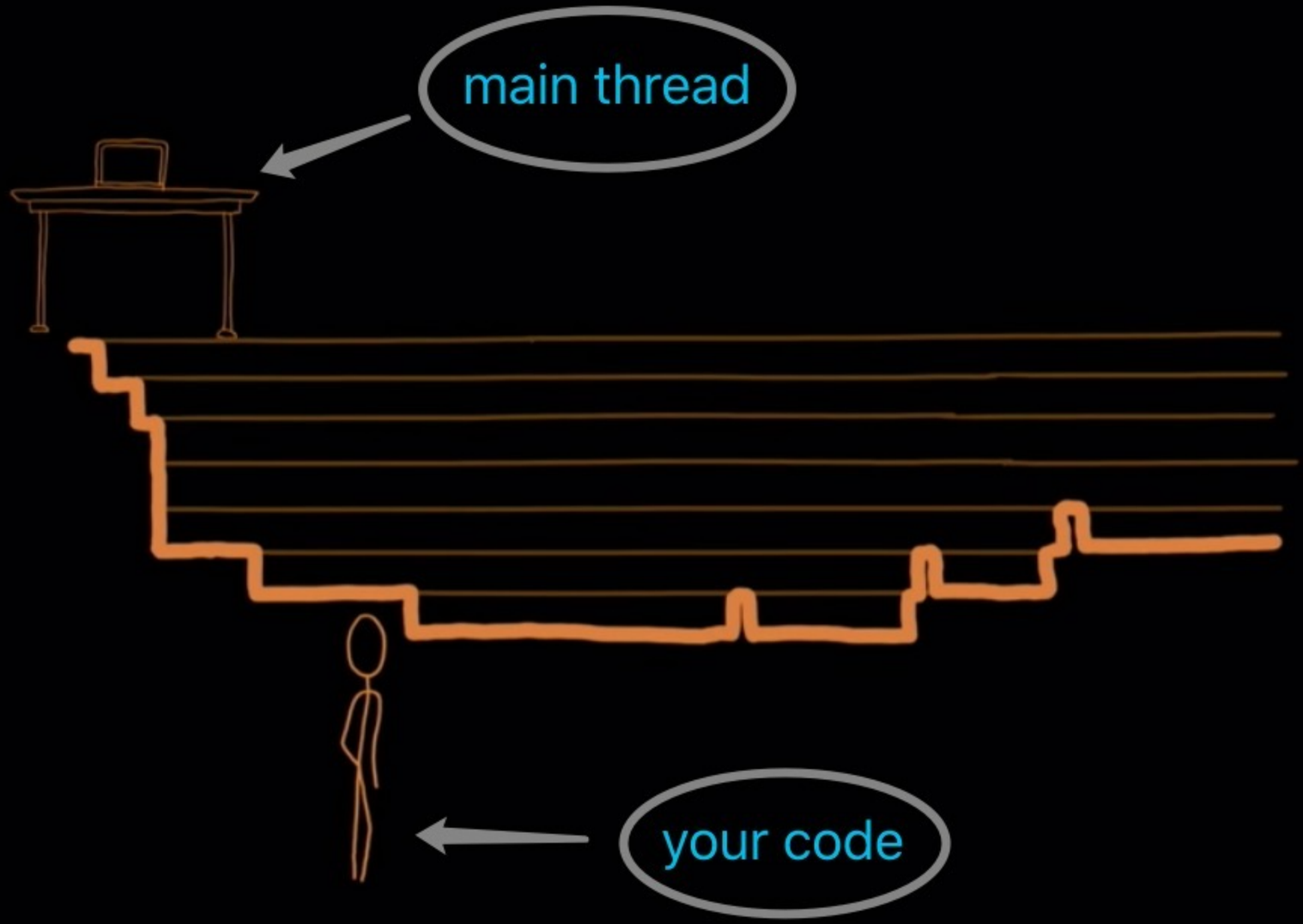


Introducing Concurrent Mode

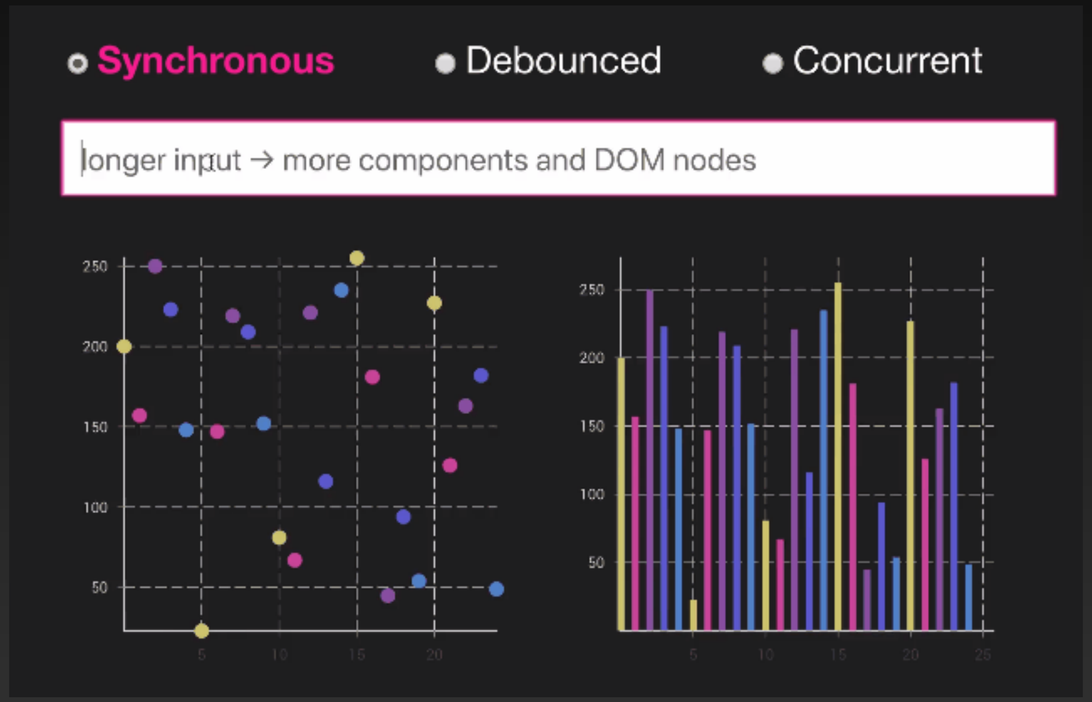
Introducing Concurrent Mode

- Concurrent Mode is a set of new features that help React apps stay responsive and gracefully adjust to the user’s device capabilities and network speed.

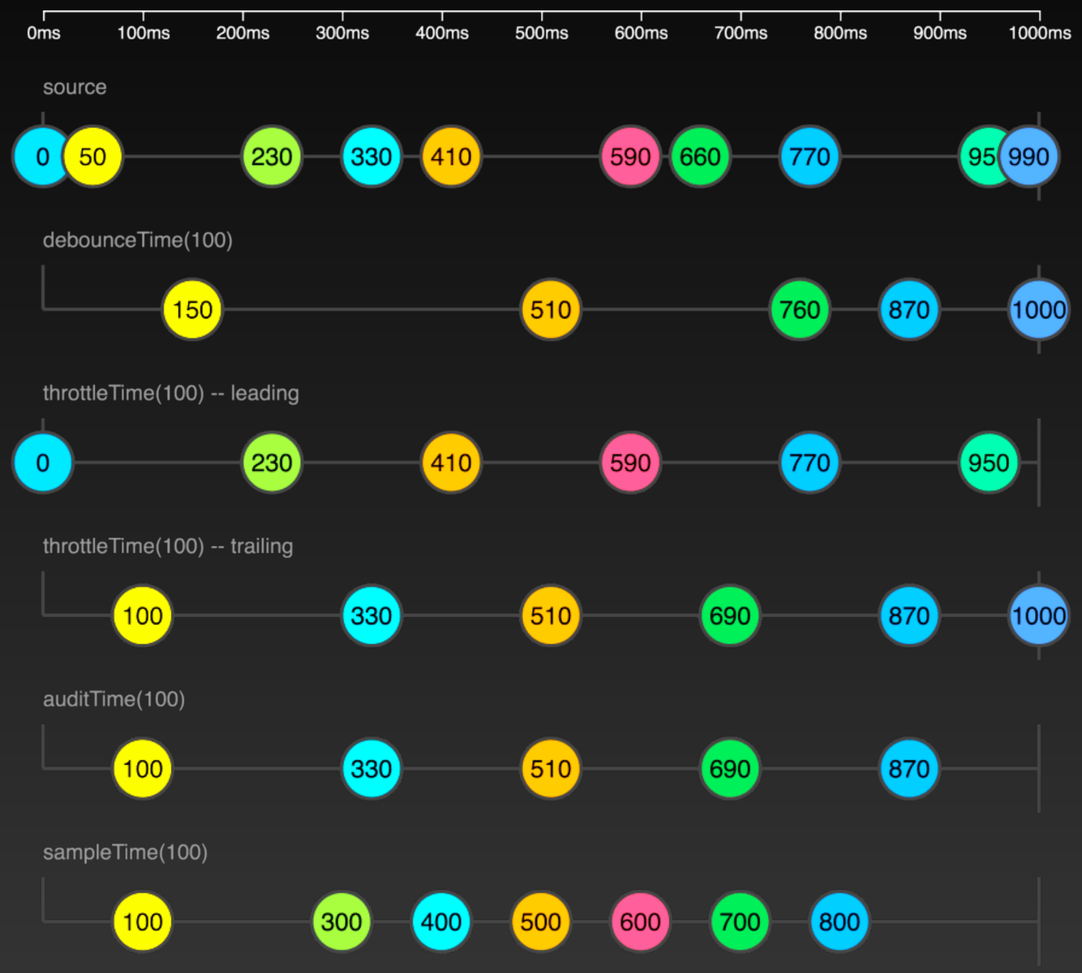
Blocking Render



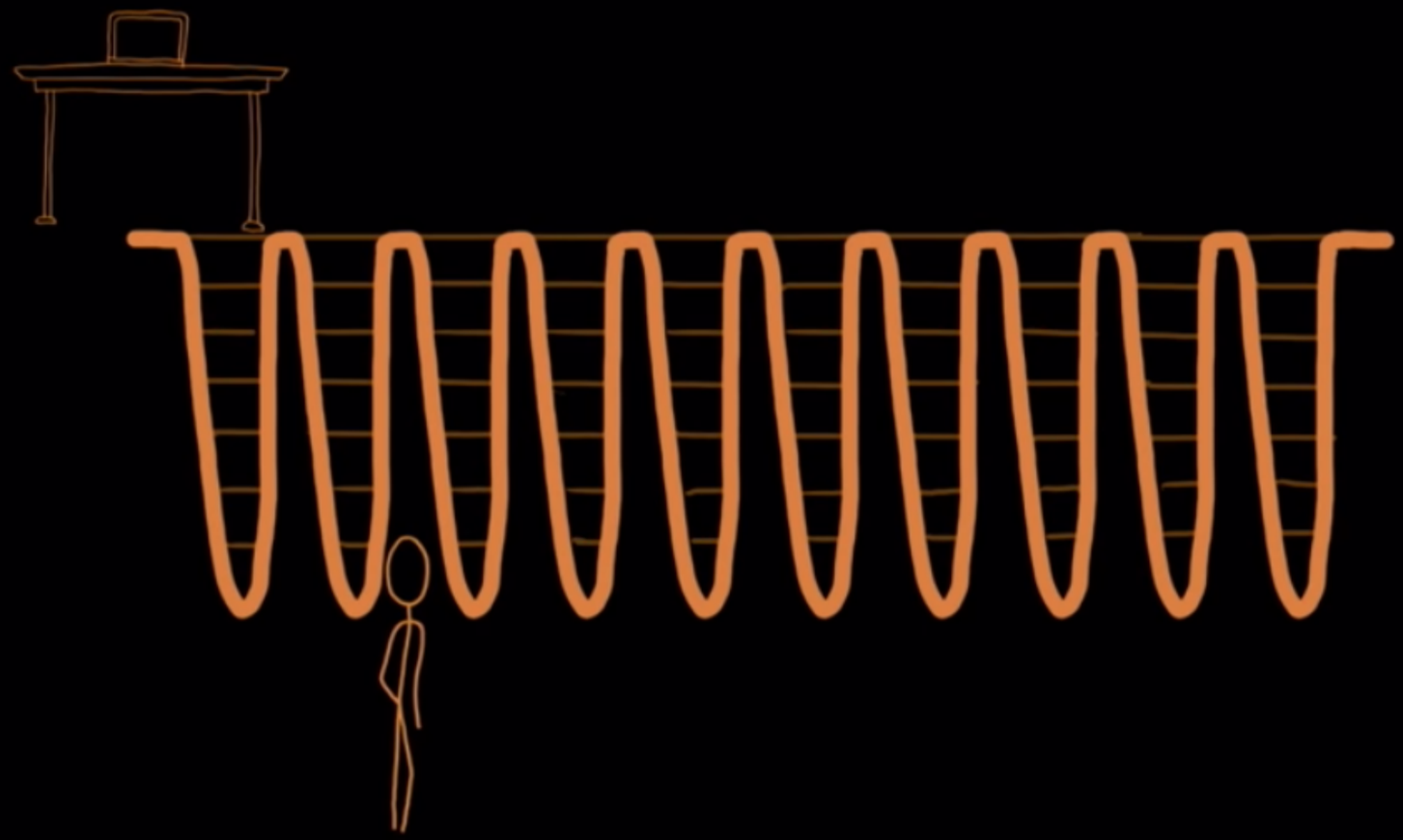
Blocking Render



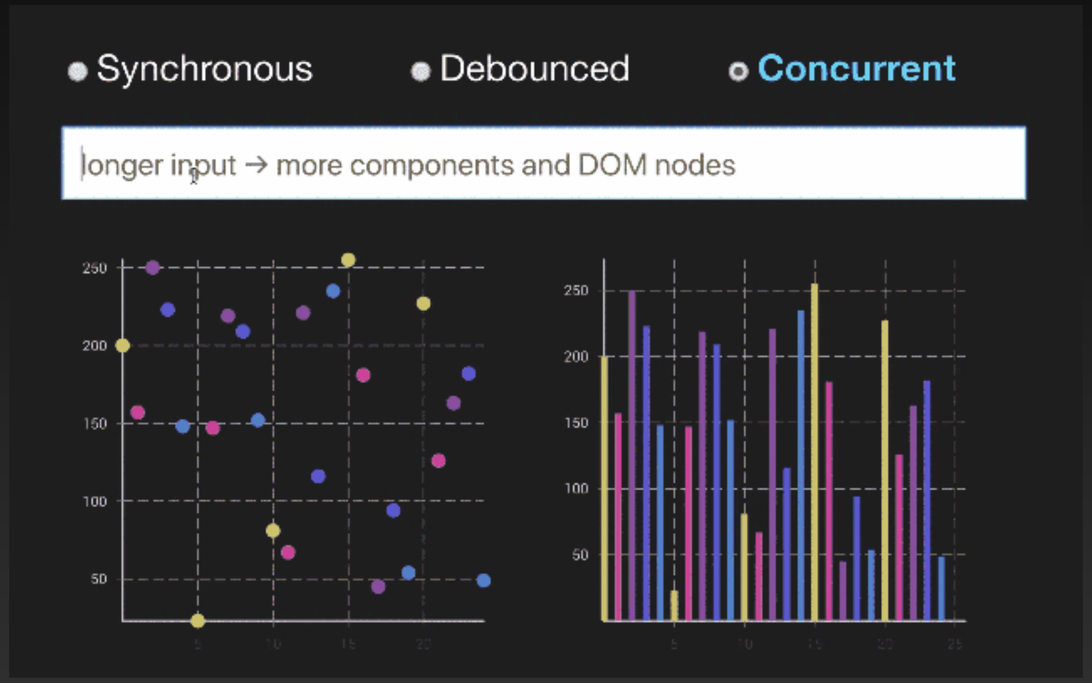
Debounce and Throttle ?



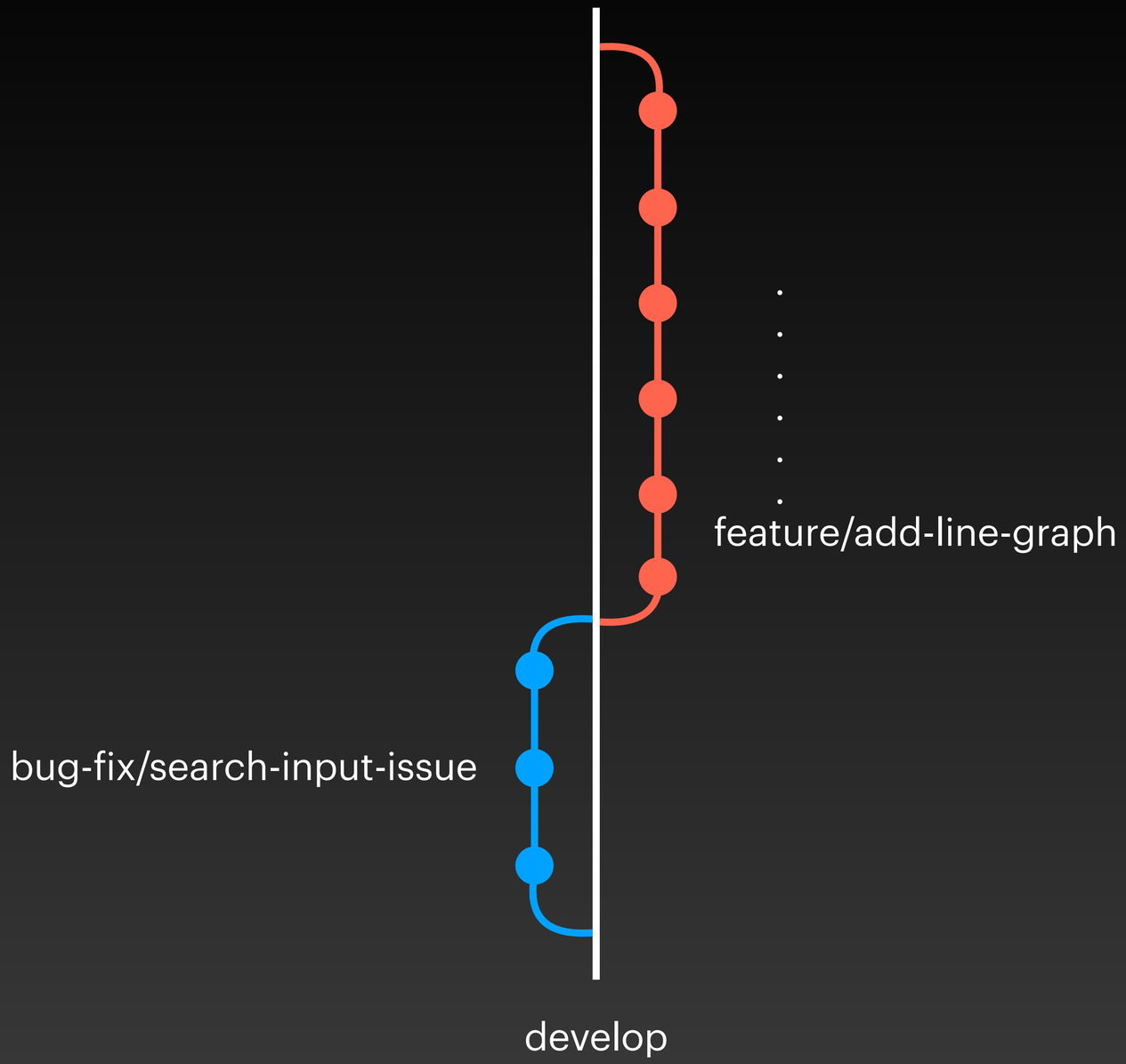
Interruptible Render



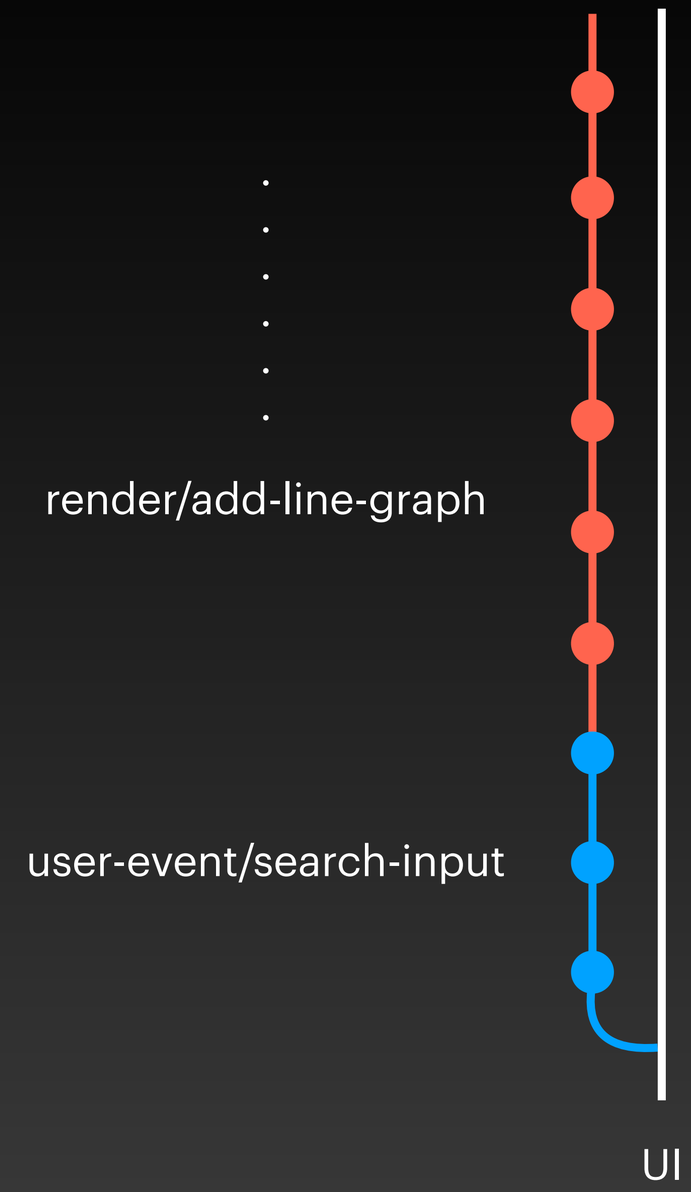
Interruptible Render



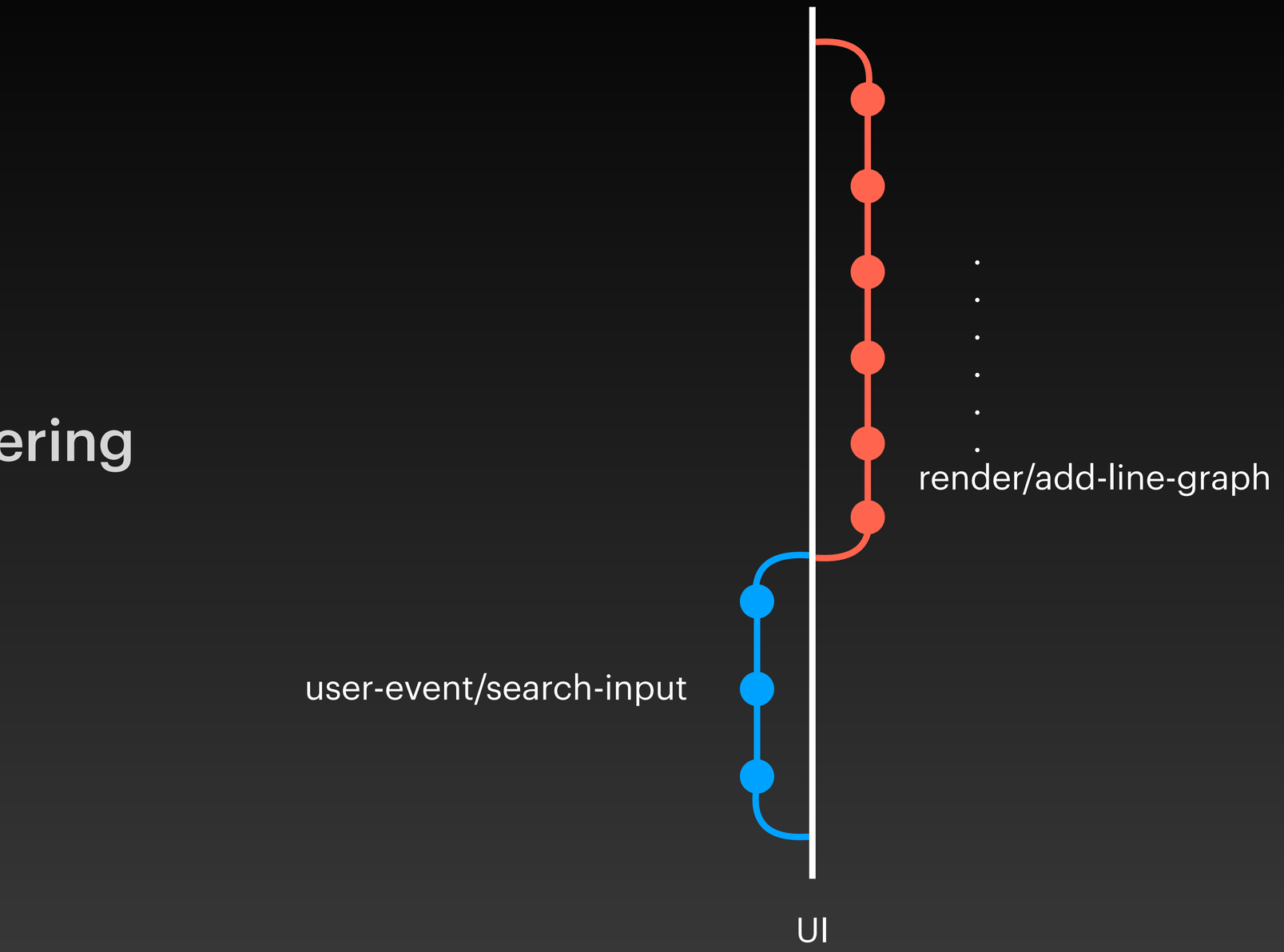
To explain Concurrent Mode, we'll use version control as a metaphor.



Blocking Rendering



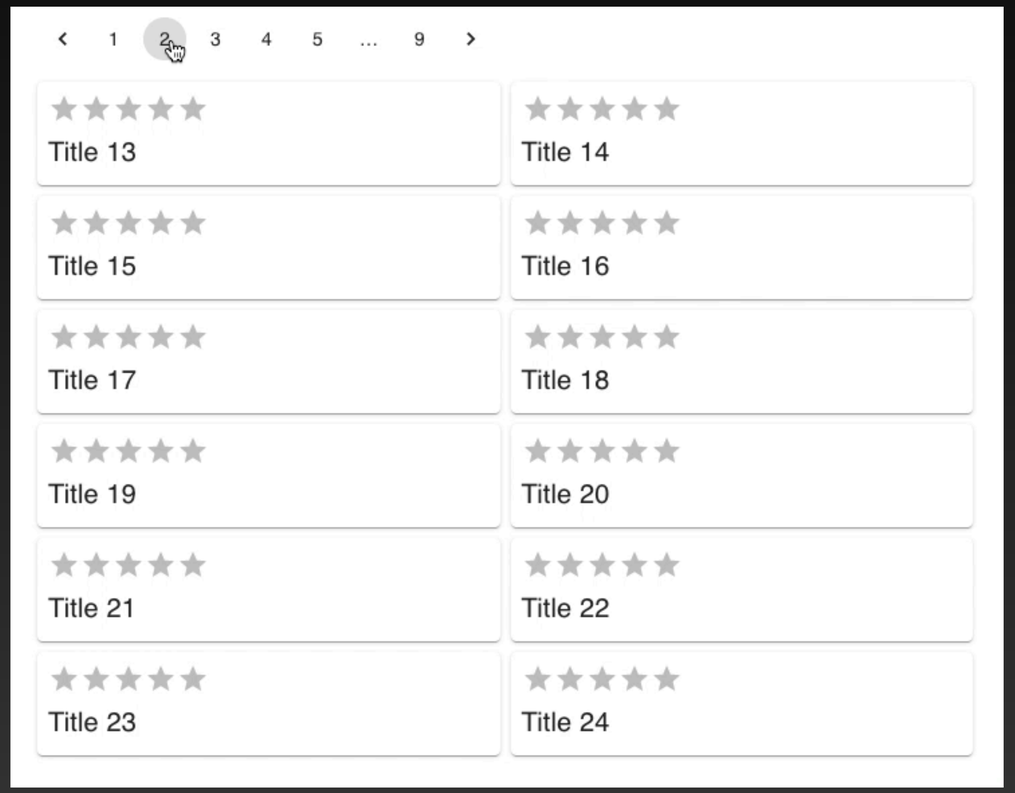
Interruptible Rendering

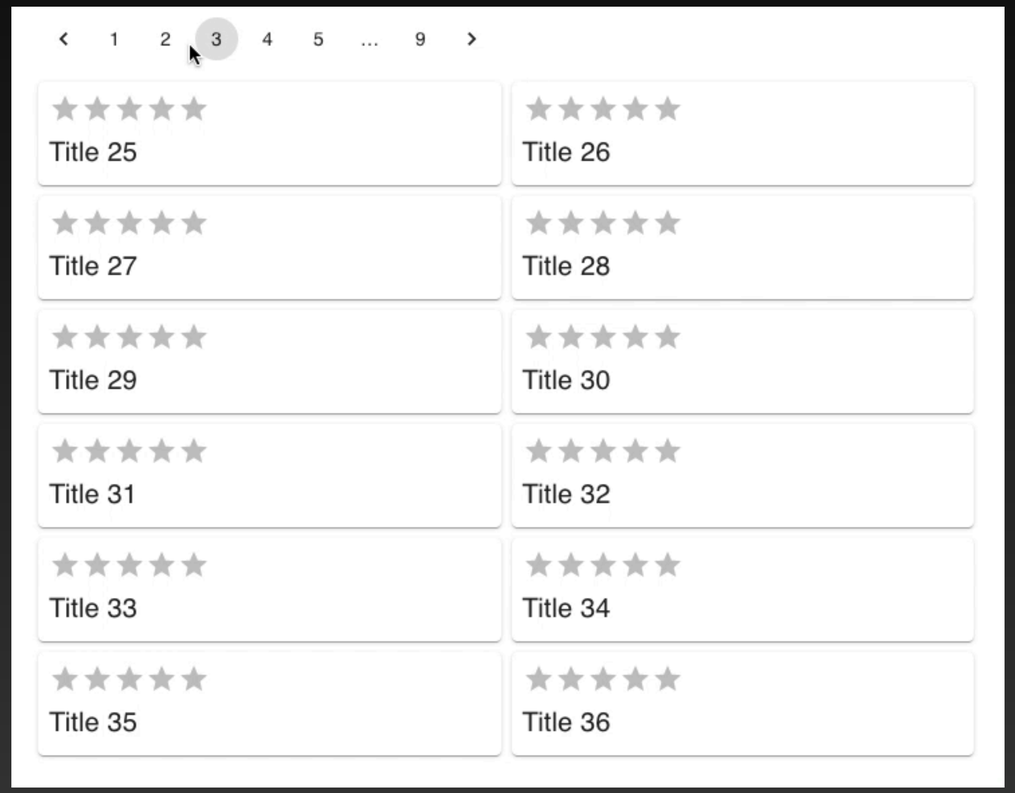


Putting Research into Production

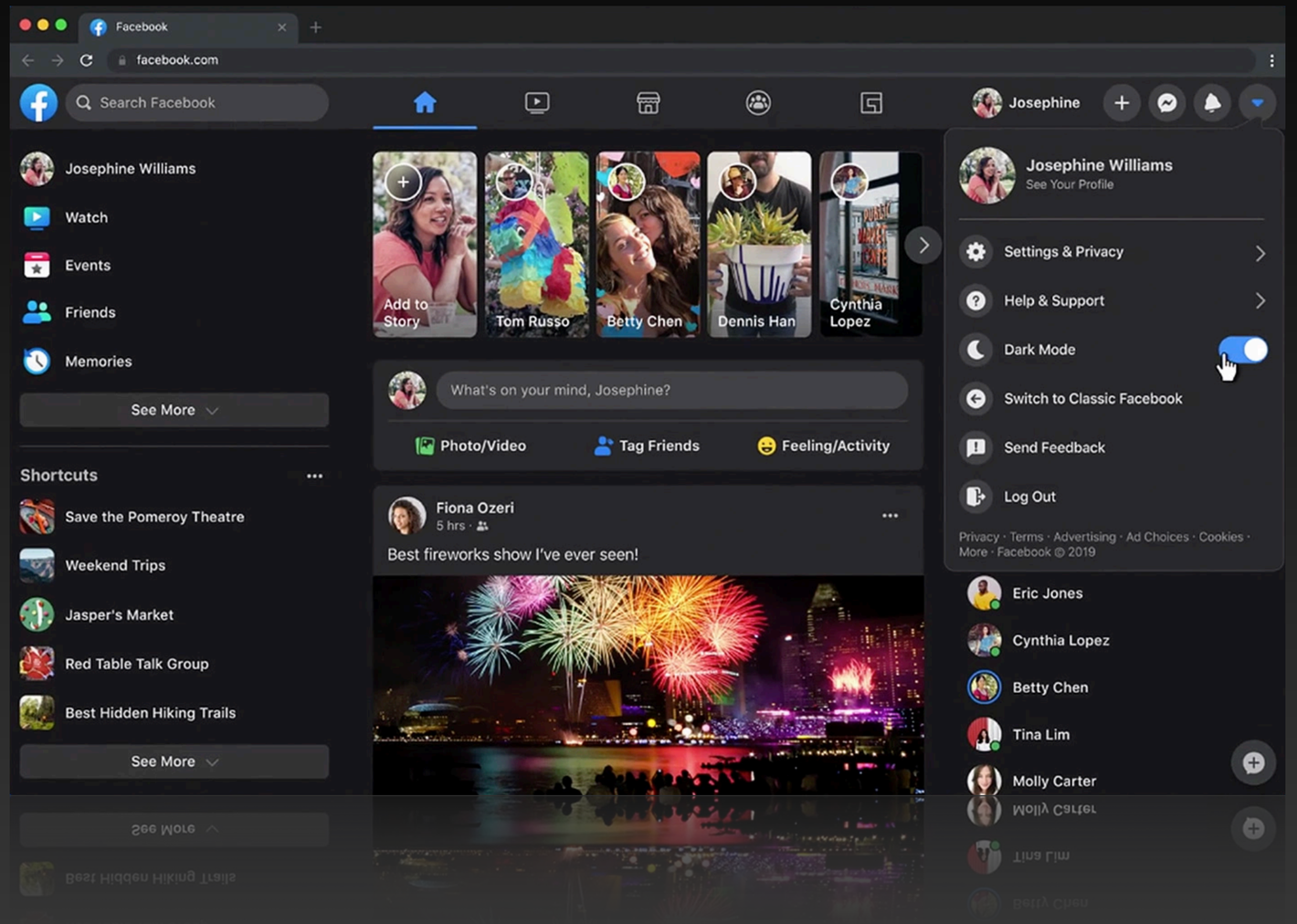
From the Human-Computer Interaction research into real UIs

Avoid too many loading spinner in Transition





New Facebook Website



Suspense for Data Fetching

<Suspense /> for Code Splitting

```jsx
import React, { Suspense, lazy } from 'react';

// This component is loaded dynamically
const OtherComponent = lazy(() => import('./OtherComponent'));

const MyComponent = () => {
  return (
    // Displays <Spinner> until OtherComponent loads
    <Suspense fallback={<Spinner />}>
      <OtherComponent />
    </Suspense>
  );
}
```

<Suspense /> for Code Splitting

- Today (React 16.6), <Suspense /> **only** supports this use case: loading components dynamically with React.lazy()
- And React.lazy() is not working in server side render now

<Suspense /> for Data Fetching

- New feature <Suspense> declaratively “wait” for anything else
  - Data
  - Scripts
  - Image
  - Or other asynchronous work

Traditional Approaches vs <Suspense />

- Fetch-on-render (fetch in useEffect)
- Fetch-then-render (fetch by Promise.all)
- Render-as-you-fetch (using Suspense)

Fetch-on-render (fetch in useEffect)

- https://codesandbox.io/s/fragrant-glade-8huj6

Fetch-on-render (fetch in useEffect)

```
function ProfilePage() {
  const [user, setUser] = useState(null);

  useEffect(() => {
    fetchUser().then(u => setUser(u));
  }, []);

  if (user === null) {
    return <p>Loading profile...</p>;
  }
  return (
    <>
      <h1>{user.name}</h1>
      <ProfileTimeline />
    </>
  );
}
```

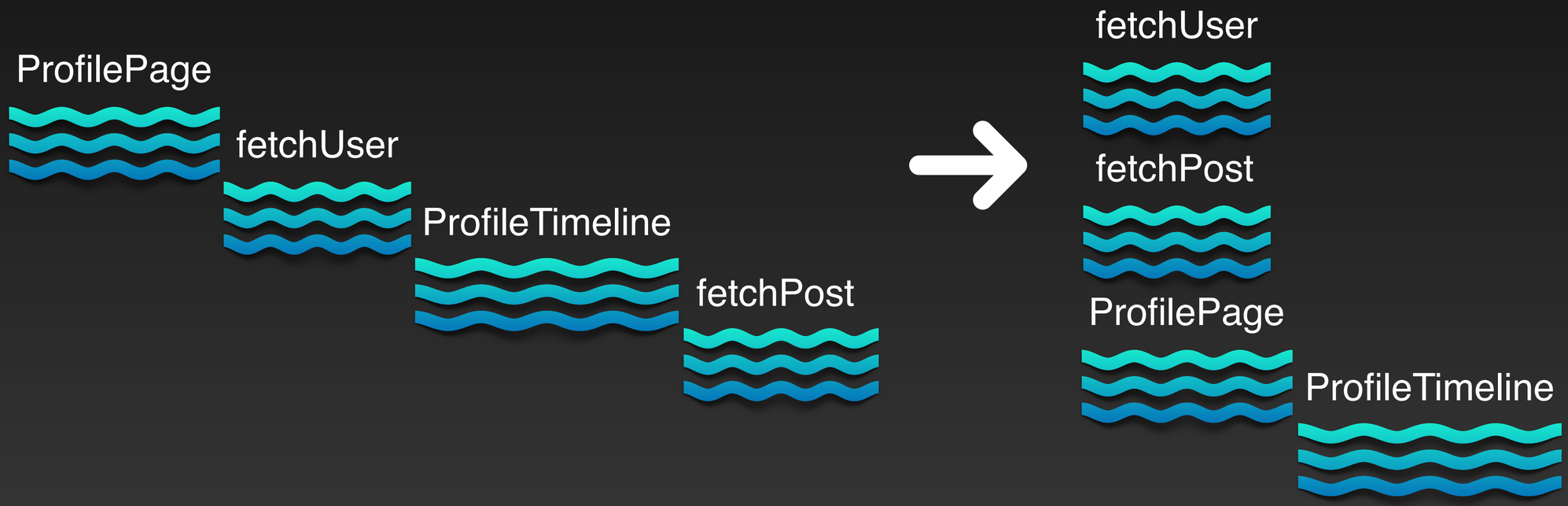
```
function ProfileTimeline() {
  const [posts, setPosts] = useState(null);

  useEffect(() => {
    fetchPosts().then(p => setPosts(p));
  }, []);

  if (posts === null) {
    return <h2>Loading posts...</h2>;
  }
  return (
    <ul>
      {posts.map(post => (
        <li key={post.id}>{post.text}</li>
      ))}
    </ul>
  );
}
```

Waterfall

- The unintentional sequence that should have been parallelized



Fetch-then-render (fetch by Promise.all)

```
// Kick off fetching as early as possible
const promise = Promise.all([
  fetchUser(), fetchPosts()
]).then(([user, posts]) => {
  return {user, posts};
})

function ProfilePage() {
  const [user, setUser] = useState(null);
  const [posts, setPosts] = useState(null);

  useEffect(() => {
    promise.then(data => {
      setUser(data.user);
      setPosts(data.posts);
    });
  }, []);

  if (user === null) return <p>Loading</p>;
  return (
    <>
      <h1>{user.name}</h1>
      <ProfileTimeline posts={posts} />
    </>
  );
}
```

```
// The child doesn't trigger fetching anymore
function ProfileTimeline({ posts }) {
  if (posts === null) {
    return <h2>Loading posts...</h2>;
  }
  return (
    <ul>
      {posts.map(post => (
        <li key={post.id}>{post.text}</li>
      ))}
    </ul>
  );
}
```

Fetch-then-render (fetch by Promise.all)

- https://codesandbox.io/s/wandering-morning-ev6r0

Fetch-then-render (fetch by Promise.all)

- We have to wait all API

fetchUser

fetchPost

We can’t render ProfilePage

Loading profile...

until fetchPost

Ringo Starr

- I get by with a little help from my friends
- I'd like to be under the sea in an octupus's garden
- You got that sand all over your feet

Render-as-you-fetch (using Suspense)

```
// This is not a Promise.
// It's a special object from our Suspense integration.
const resource = fetchProfileData();

function ProfilePage() {
  return (
    <Suspense fallback={<h1>Loading profile...</h1>}>
      <ProfileDetails />
      <Suspense fallback={<h1>Loading posts...</h1>}>
        <ProfileTimeline />
      </Suspense>
    </Suspense>
  );
}
```

```
function ProfileDetails() {
  // Try to read user info,
  // although it might not have loaded yet
  const user = resource.user.read();
  return <h1>{user.name}</h1>;
}

function ProfileTimeline() {
  // Try to read posts, although they might not have loaded yet
  const posts = resource.posts.read();
  return (
    <ul>
      {posts.map(post => (
        <li key={post.id}>{post.text}</li>
      ))}
    </ul>
  );
}
```

Render-as-you-fetch (using Suspense)

- https://codesandbox.io/s/frosty-hermann-bztrp

Render-as-you-fetch (using Suspense)

- With Suspense, we don’t wait for the response to come back before we start rendering.

fetchUser

fetchPost

Render ProfileDetails

Ringo Starr

Loading posts...

Render ProfileTimeline

Ringo Starr

- I get by with a little help from my friends
- I'd like to be under the sea in an octupus's garden
- You got that sand all over your feet

Render-as-you-fetch (using Suspense)

- And also waiting inner <Suspense />

fetchUser

fetchPost

Render ProfileDetails

Loading profile...

Render ProfileTimeline

Ringo Starr

- I get by with a little help from my friends
- I'd like to be under the sea in an octupus's garden
- You got that sand all over your feet

Render-as-you-fetch (using Suspense)

```
// This is not a Promise.
// It's a special object from our Suspense integration.
const resource = fetchProfileData();

function ProfilePage() {
  return (
    <Suspense fallback={<h1>Loading profile...</h1>}>
      <ProfileDetails />
      <Suspense fallback={<h1>Loading posts...</h1>}>
        <ProfileTimeline />
      </Suspense>
    </Suspense>
  );
}
```

Render-as-you-fetch (using Suspense)

- We kick off fetching before rendering
- No waterfall, no waiting for all
- Eliminated the if (...) “is loading” checks from our components

Simplifies making quick loading design changes

- https://codesandbox.io/s/frosty-hermann-bztrp?file=/src/index.js:220-440

Suspense and Race Conditions

- Incorrect assumptions about the order in which our code may run.
- https://codesandbox.io/s/nervous-glade-b5sel

Suspense and Race Conditions

- React components have their own “lifecycle”.
- However, each asynchronous request also has its own “lifecycle”.

fetchUser

fetchPost

New ProfileDetails
New ProfileTimeline

Old ProfileDetails
New ProfileTimeline

Suspense and Race Conditions

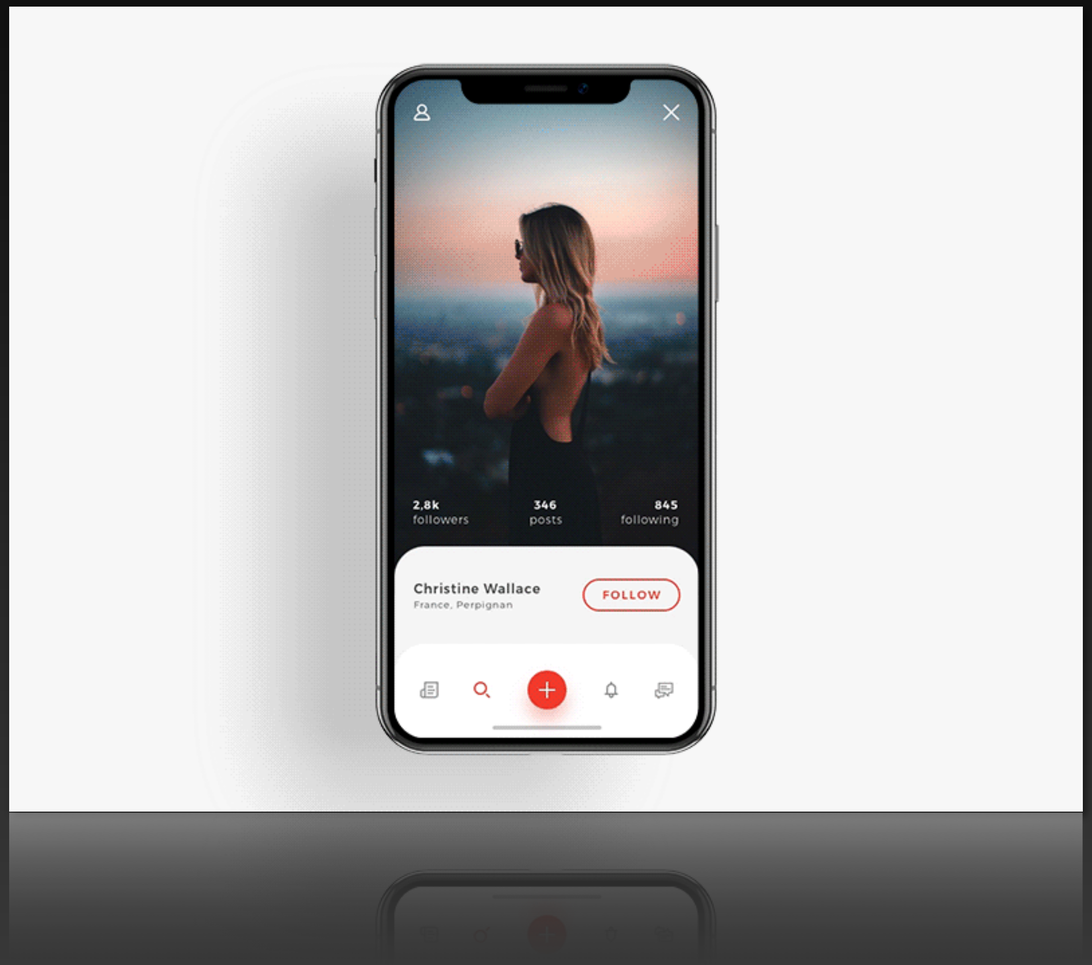
- We can fix this issue by <Suspense />
- https://codesandbox.io/s/infallible-feather-xjtbu

Great DX only matters if it delivers great UX

Tom Occhino - Director of the React.js

Concurrent UI Patterns

Transitions

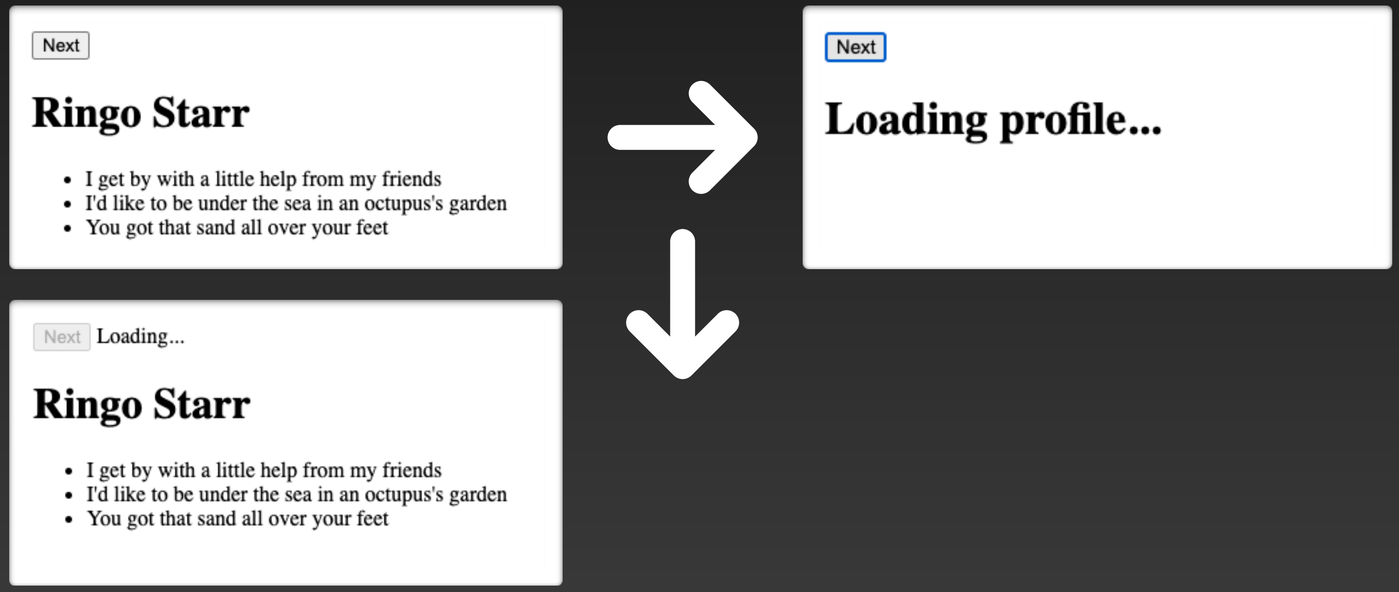


Transitions

- Transition are usually used to move content in or out of view
- Transition help to make a UI expressive and easy to use
- https://codesandbox.io/s/infallible-feather-xjtbu

Transitions

- It would be nice if we could skip “undesirable” loading state
- And wait for some content to load for a while
- https://codesandbox.io/s/jovial-lalande-26yep



useTransition

```
const SUSPENSE_CONFIG = { timeoutMs: number };

const [startTransition, isPending] = useTransition(SUSPENSE_CONFIG);
```

- useTransition hook waits for content to load before transitioning
- startTransition: it tells React which state we want to defer.
- isPending is a boolean. It informs us the transition is finished or not

startTransition

```
<button
  onClick={() => {
    const nextUserId = getNextId(resource.userId);
    setResource(fetchProfileData(nextUserId));
  }}
>
```

```
<button
  onClick={() => {
    startTransition(() => {
      const nextUserId = getNextId(resource.userId);
      setResource(fetchProfileData(nextUserId));
    });
  }}
>
```

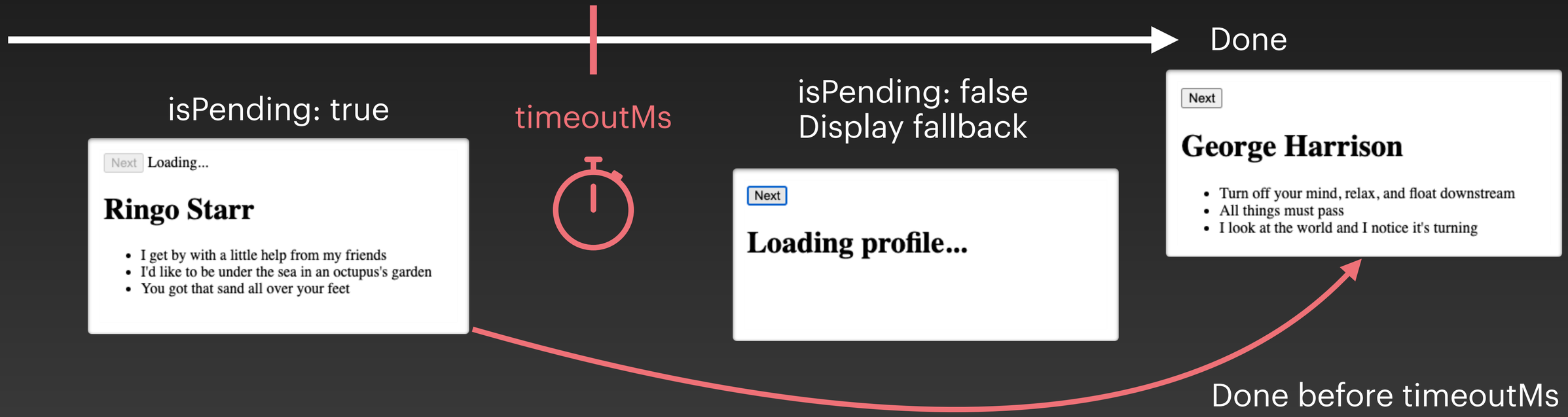
isPending

```
return (
  <>
    <button
      disabled={isPending}
      onClick={() => {
        startTransition(() => {
          const nextUserId = getNextId(resource.userId);
          setResource(fetchProfileData(nextUserId));
        });
      }}
    >
      Next
    </button>
    {isPending ? " Loading..." : null}
    <ProfilePage resource={resource} />
  </>
);
```

useTransition

```
const SUSPENSE_CONFIG = { timeoutMs: number };

const [startTransition, isPending] = useTransition(SUSPENSE_CONFIG);
```



Refresh Button Example

- https://codesandbox.io/s/sleepy-field-mohzb

Refresh Button Example

- useTransition is very common
- A lot of repetitive code across Button components
- Bake useTransition into the design system components of your app.

Baking Transitions Into the Design System

```
function Button({ children, onClick }) {
  const [startTransition, isPending] = useTransition({
    timeoutMs: 10000
  });

  function handleClick() {
    startTransition(() => {
      onClick();
    });
  }

  const spinner = (
    // ...
  );

  return (
    <>
      <button
        onClick={handleClick}
        disabled={isPending}
      >
        {children}
      </button>
      {isPending ? spinner : null}
    </>
  );
}
```

Baking Transitions Into the Design System

```
function ProfilePage() {
  const [resource, setResource] = useState(initialResource);

  function handleRefreshClick() {
    setResource(fetchProfileData());
  }

  return (
    <Suspense fallback={<h1>Loading profile...</h1>}>
      <ProfileDetails resource={resource} />
      <Button onClick={handleRefreshClick}>
        Refresh
      </Button>
      <Suspense fallback={<h1>Loading posts...</h1>}>
        <ProfileTimeline resource={resource} />
      </Suspense>
    </Suspense>
  );
}
```

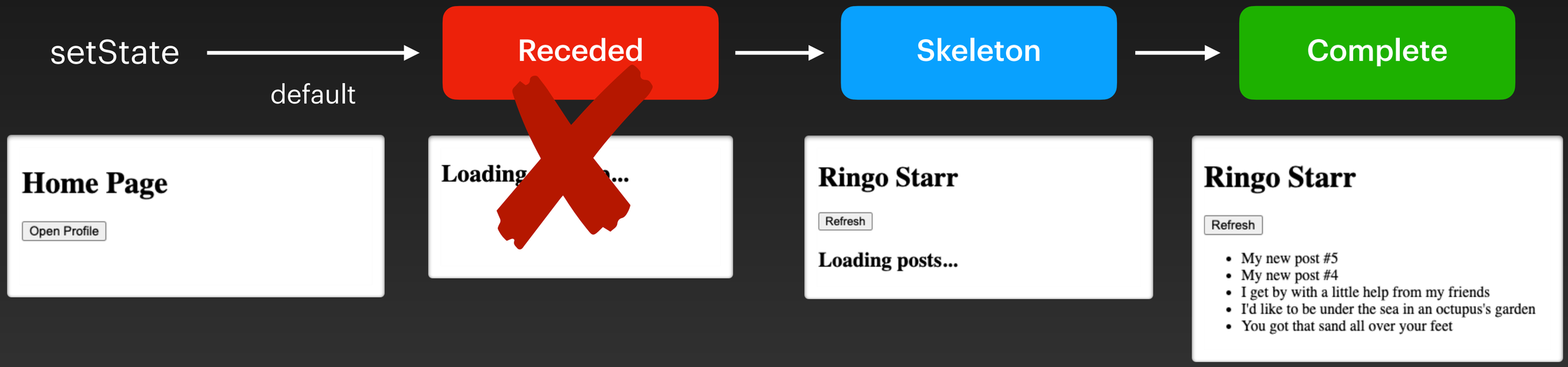
Baking Transitions Into the Design System

- https://codesandbox.io/s/modest-ritchie-iufrh

The Three Steps

- Default: Receded → Skeleton → Complete
- https://codesandbox.io/s/prod-grass-g1lh5

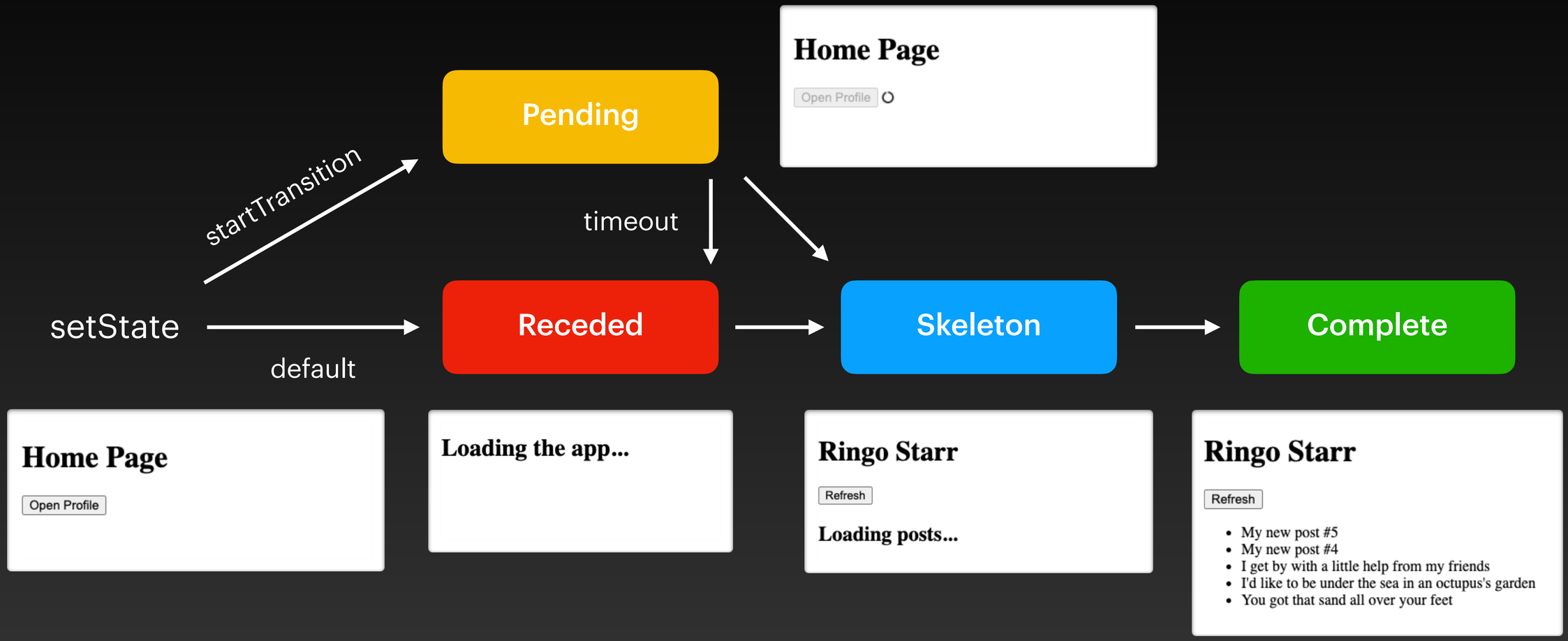
Default: Receded → Skeleton → Complete



The Three Steps

- Preferred: Pending → Skeleton → Complete
- https://codesandbox.io/s/focused-snow-xbkvl

Preferred: Pending → Skeleton → Complete



Splitting High and Low Priority State

- Documentation example is broken, please try my example
- https://codesandbox.io/s/translate-without-suspense-xyc57

Splitting High and Low Priority State

```
function handleChange(e) {
  const value = e.target.value;

  startTransition(() => {
    setQuery(value);
    setResource(fetchTranslation(value));
  });
}
```

```
function handleChange(e) {
  const value = e.target.value;

  // Outside the transition (urgent)
  setQuery(value);

  startTransition(() => {
    // Inside the transition (may be delayed)
    setResource(fetchTranslation(value));
  });
}
```

High

Low

Splitting High and Low Priority State

- Without Priority: https://codesandbox.io/s/without-set-priority-snq0q
- With Priority: https://codesandbox.io/s/suspense-set-priority-y24dh

Warning

- Warning: App triggered a user-blocking update that suspended.

  The fix is to split the update into multiple parts: a **user-blocking** update to provide immediate feedback, and another update that triggers the **bulk of the changes**.

<SuspenseList />

Orchestrating <Suspense> loading states

2 Difference Pending Time

```
function ProfilePage({ resource }) {
  return (
    <>
      <ProfileDetails resource={resource} />
      <Suspense fallback={<h2>Loading posts...</h2>}>
        <ProfileTimeline resource={resource} />
      </Suspense>
      <Suspense fallback={<h2>Loading fun facts...</h2>}>
        <ProfileTrivia resource={resource} />
      </Suspense>
    </>
  );
}
```

https://codesandbox.io/s/proud-tree-exg5t

If Post Response Arrives first

fetchPost

fetchTrivia

Ringo Starr

- I get by with a little help from my friends
- I'd like to be under the sea in an octupus's garden
- You got that sand all over your feet

Loading fun facts...

Ringo Starr

- I get by with a little help from my friends
- I'd like to be under the sea in an octupus's garden
- You got that sand all over your feet

Fun Facts

- The nickname "Ringo" came from his habit of wearing numerous rings.
- Plays the drums left-handed with a right-handed drum set.
- Nominated for one Daytime Emmy Award, but did not win

If Fun Facts Arrives first

fetchPost

fetchTrivia

Ringo Starr

Loading posts...

Fun Facts

- The nickname "Ringo" came from his habit of wearing numerous rings.
- Plays the drums left-handed with a right-handed drum set.
- Nominated for one Daytime Emmy Award, but did not win

Ringo Starr

- I get by with a little help from my friends
- I'd like to be under the sea in an octupus's garden
- You got that sand all over your feet

Fun Facts

- The nickname "Ringo" came from his habit of wearing numerous rings.
- Plays the drums left-handed with a right-handed drum set.
- Nominated for one Daytime Emmy Award, but did not win

If Fun Facts Arrives first

Ringo Starr

- I get by with a little help from my friends
- I'd like to be under the sea in an octupus's garden
- You got that sand all over your feet

Fun Facts

- The nickname "Ringo" came from his habit of wearing numerous rings.
- Plays the drums left-handed with a right-handed drum set.
- Nominated for one Daytime Emmy Award, but did not win

<SuspenseList />

```
import { SuspenseList } from 'react';

function ProfilePage({ resource }) {
  return (
    <SuspenseList revealOrder="forwards">
      <ProfileDetails resource={resource} />
      <Suspense fallback={<h2>Loading posts...</h2>}>
        <ProfileTimeline resource={resource} />
      </Suspense>
      <Suspense fallback={<h2>Loading fun facts...</h2>}>
        <ProfileTrivia resource={resource} />
      </Suspense>
    </SuspenseList>
  );
}
```

https://codesandbox.io/s/black-wind-byilt

If Fun Facts Arrives first (forward)

fetchPost

fetchTrivia

Ringo Starr

Loading posts... ①

Loading fun facts... ②

Ringo Starr

- I get by with a little help from my friends
- I'd like to be under the sea in an octupus's garden
- You got that sand all over your feet

Fun Facts

- The nickname "Ringo" came from his habit of wearing numerous rings.
- Plays the drums left-handed with a right-handed drum set.
- Nominated for one Daytime Emmy Award, but did not win

<SuspenseList /> revealOrder

- forwards
- backwards
- together
- https://n3818.csb.app/

<SuspenseList /> tail

- By default, <SuspenseList /> will show all fallbacks in the list.
- collapsed
- hidden
- https://codesandbox.io/s/adoring-almeida-1zzjh

Adopting Concurrent Mode

Installation

- npm install react@experimental react-dom@experimental

Enabling Concurrent Mode

```
import ReactDOM from ‘react-dom';

// If you previously had:
//
// ReactDOM.render(<App />, document.getElementById('root'));
//
// You can opt into Concurrent Mode by writing:

ReactDOM.createRoot(
  document.getElementById('root')
).render(<App />);
```

Experimental

- There are no semantic versioning guarantees
- Experimental releases will have frequent breaking changes
- It’s not recommended running them in production.

- At Facebook, they do run them in production, but that’s because they’re also there to fix bugs when something breaks.

What to Expect

- If you have a large existing app,
  or if your app depends on a lot of third-party packages,
  please don’t expect that you can use the Concurrent Mode immediately.

- For example, at Facebook
  they are using Concurrent Mode for the new website,
  but we’re not planning to enable it on the old website.

Migration Step: Blocking Mode

- For older codebases, Concurrent Mode might be a step too far
- Blocking Mode only offers a small subset of the Concurrent Mode features
- Blocking Mode is closer to how React works today and can serve as a migration step

Why So Many Modes?

- Gradual migration strategy
- If React.js make huge breaking changes, old codebase developers will be stuck on old versions
- As a result, in longer term we should be able to converge and stop thinking about different Modes altogether

- In the future, after it stabilizes, we intend to make **Concurrent Mode** the default React mode

Recap

```
import ReactDOM from 'react-dom';

// Legacy Mode:
ReactDOM.render(<App />, rootNode);

// Blocking Mode:
ReactDOM.createBlockingRoot(rootNode).render(<App />);

// Concurrent Mode:
ReactDOM.createRoot(rootNode).render(<App />);
```

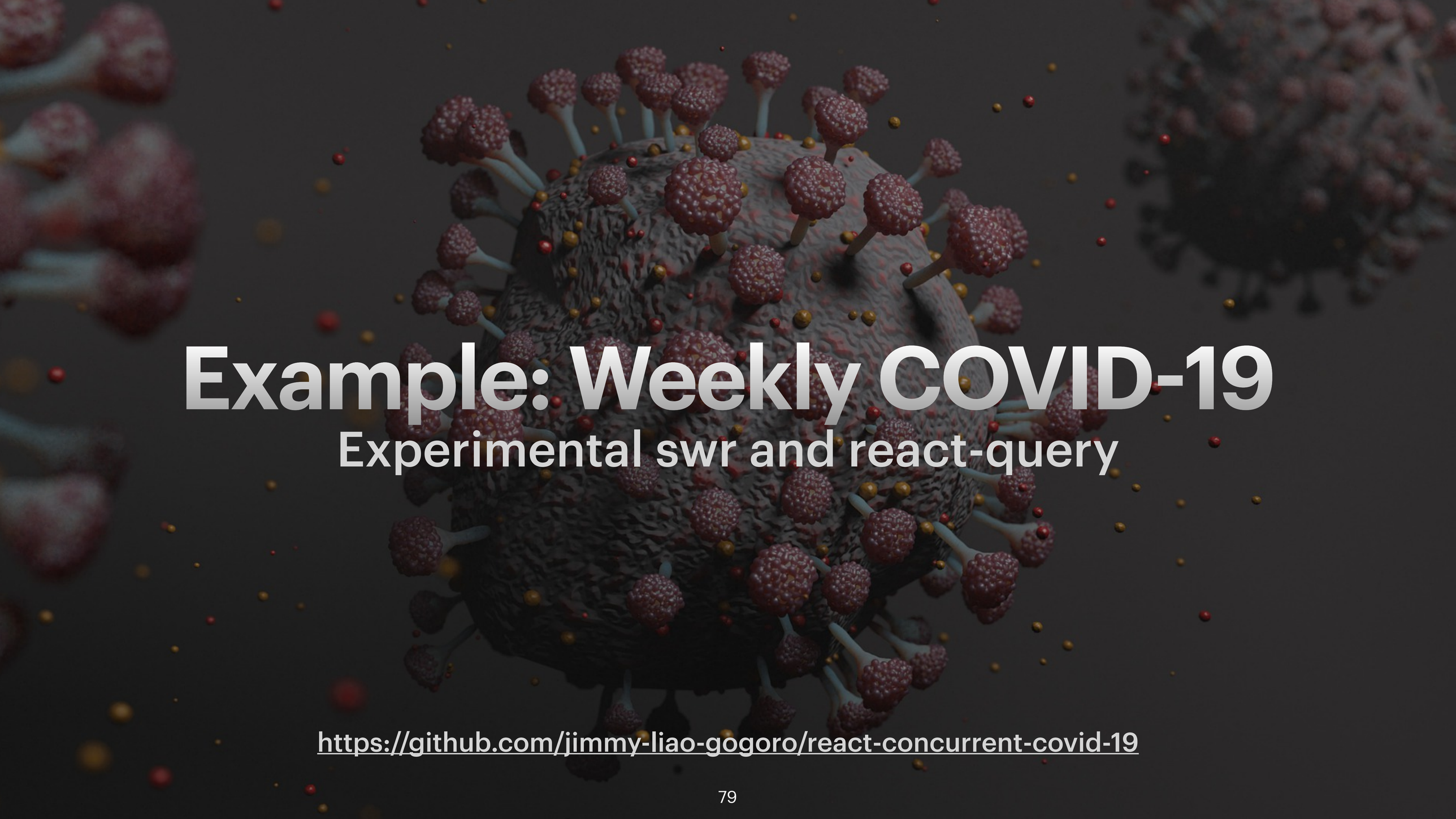
Feature Comparison

Feature Comparison

	Legacy Mode	Blocking Mode	Concurrent Mode
String Refs	✅	🚫**	🚫**
Legacy Context	✅	🚫**	🚫**
findDOMNode	✅	🚫**	🚫**
Suspense	✅	✅	✅
SuspenseList	🚫	✅	✅
Suspense SSR + Hydration	🚫	✅	✅
Progressive Hydration	🚫	✅	✅
Selective Hydration	🚫	🚫	✅
Cooperative Multitasking	🚫	🚫	✅
Automatic batching of multiple setStates	🚫*	✅	✅
Priority-based Rendering	🚫	🚫	✅
Interruptible Prerendering	🚫	🚫	✅
useTransition	🚫	🚫	✅
useDeferredValue	🚫	🚫	✅
Suspense Reveal "Train"	🚫	🚫	✅

*: Legacy mode has automatic batching in React-managed events but it's limited to one browser task. Non-React events must opt-in using `unstable_batchedUpdates`. In Blocking Mode and Concurrent Mode, all `setState`s are batched by default.

**: Warns in development.



Example: Weekly COVID-19

Experimental swr and react-query

https://github.com/jimmy-liao-gogoro/react-concurrent-covid-19

One More Thing

React 17 Disable timeoutMs argument

- Say good bye to useTransition() timeoutMs argument
- https://github.com/facebook/react/pull/19703

Concurrent Mode is experimental features

Q & A

Thank You For Listening