

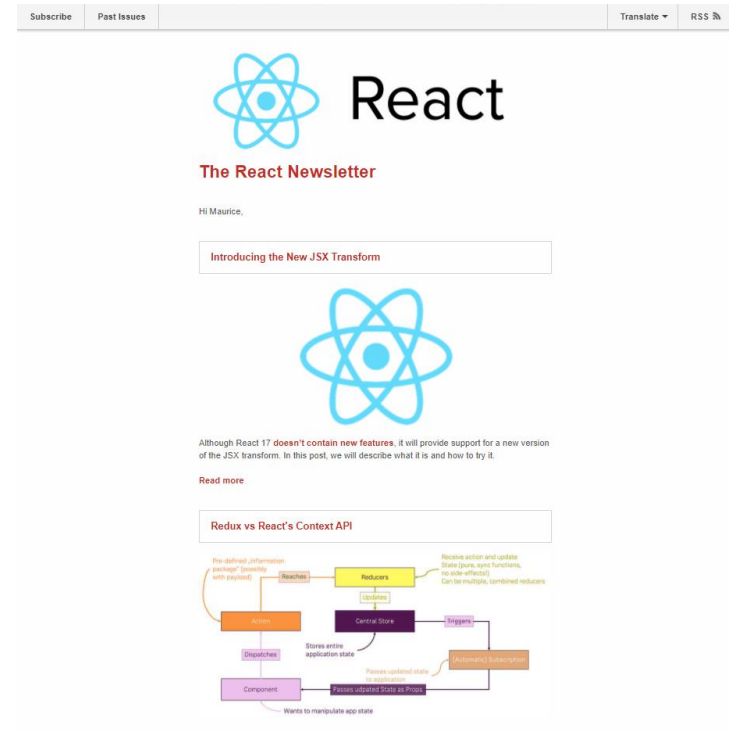
Getting started with Suspense & Concurrent Rendering

Maurice de Beijer - @mauricedb



- Maurice de Beijer
- The Problem Solver
- Microsoft MVP
- Freelance developer/instructor
- Twitter: @mauricedb
- Web: <http://www.TheProblemSolver.nl>
- E-mail: maurice.de.beijer@gmail.com

The React Newsletter



Workshop goal

- Use **<Suspense />**
 - Bundle splitting and lazy loading
 - Data resources
- UI **approaches**
 - Fetch from render
 - Fetching before render
- **Concurrent mode**
 - Enabling Concurrent mode
 - Orchestrating **<Suspense />** boundaries using **<SuspenseList />**
 - Using **useTransition()** to prioritize work

Type it out
by hand?

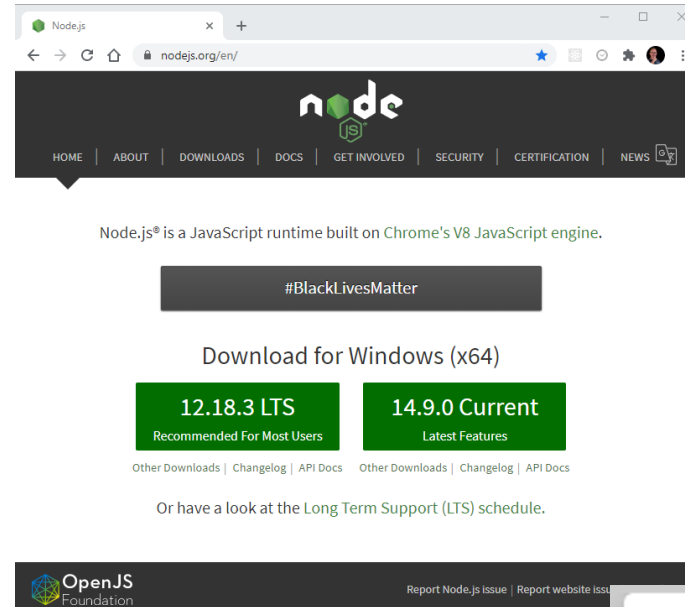
"Typing it drills it into your brain much better than simply copying and pasting it. You're forming new neuron pathways. Those pathways are going to help you in the future. Help them out now!"

Prerequisites

Install Node & NPM

Install the GitHub repository

Install Node.js & NPM



```
Windows PowerShell
PS C:\Temp> node --version
v12.18.3
PS C:\Temp> npm --version
6.14.8
PS C:\Temp> |
```

Clone the GitHub Repository

```
Windows PowerShell
PS C:\Temp> git clone git@github.com:mauricedb/react-suspense-2020.git
Cloning into 'react-suspense-2020'...
remote: Enumerating objects: 137, done.
remote: Counting objects: 100% (137/137), done.
remote: Compressing objects: 100% (75/75), done.
remote: Total 137 (delta 69), reused 122 (delta 57), pack-reused 0R
Receiving objects: 100% (137/137), 641.76 KiB | 1.72 MiB/s, done.
Resolving deltas: 100% (69/69), done.
PS C:\Temp> |
```



Install NPM Packages

```
Windows PowerShell
PS C:\Repos\react-suspense-2020> npm ci

> core-js-pure@3.6.5 postinstall C:\Repos\react-suspense-2020\node_modules\core-js-pure
> node -e "try{require('./postinstall')}catch(e){}"

Thank you for using core-js ( https://github.com/zloirock/core-js ) for polyfilling JavaScript standard library!

The project needs your help! Please consider supporting of core-js on Open Collective or Patreon:
> https://opencollective.com/core-js
> https://www.patreon.com/zloirock

Also, the author of core-js ( https://github.com/zloirock ) is looking for a good job -)

> core-js@2.6.11 postinstall C:\Repos\react-suspense-2020\node_modules\babel-runtime\node_modules\core-js
> node -e "try{require('./postinstall')}catch(e){}"

> fsevents@1.2.13 install C:\Repos\react-suspense-2020\node_modules\webpack-dev-server\node_modules\fsevents
> node install.js

Skipping 'fsevents' build as platform win32 is not supported

> fsevents@1.2.13 install C:\Repos\react-suspense-2020\node_modules\watchpack-chokidar2\node_modules\fsevents
> node install.js

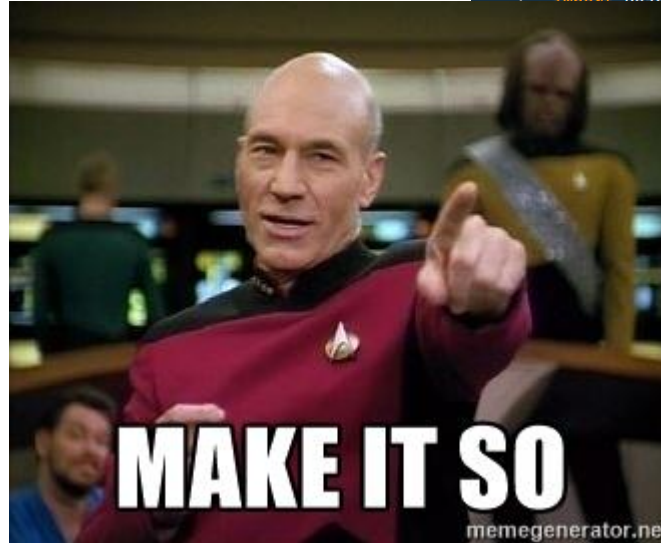
Skipping 'fsevents' build as platform win32 is not supported

> core-js@3.6.5 postinstall C:\Repos\react-suspense-2020\node_modules\core-js
> node -e "try{require('./postinstall')}catch(e){}"

> fsevents@1.2.13 install C:\Repos\react-suspense-2020\node_modules\jest-haste-map\node_modules\fsevents
> node install.js

Skipping 'fsevents' build as platform win32 is not supported
added 1916 packages in 8.732s
PS C:\Repos\react-suspense-2020> |
```

Following Along



```
TS App.tsx ×
src > TS App.tsx > ...
You, 4 days ago | 1 author (You)
1 import React from "react";
  BrowserRouter, Route, Switch } from "react-router-dom";

  r from "./components/NavBar";
  = React.lazy(() => import("./components/Movies"));
  = React.lazy(() => import("./components/Users"));
  tails = React.lazy(() => import("./components/UserDetails"));


  eact.FC = () => (
    uted>
    ssName="container">
    r />
    h>
    te path="/movies">
    ovies />
    ute>
```

- Repository: <http://bit.ly/suspense-2020>
- Slides: <http://bit.ly/suspense-2020-pdf>



Bundle Splitting & Lazy Loading

Bundle Splitting & Lazy Loading

- By default CRA puts all application code in a **single code bundle**
 - All code is loaded when the application first starts
- Many **components may not be rendered** until much later
 - Or maybe never at all
- Splitting the code into multiple bundles makes **the startup faster**
 - Only load additional parts when they are actually needed
- Using **a bundle per route** is a good start
 -  Beware that shared code could end up in multiple bundles
 - Import in the main bundle to really share the code
- **Using React.lazy()** automatically creates a new bundle

React.lazy()



```
TS App.tsx  X
src > TS App.tsx > ...
You, 4 days ago | 1 author (You)
1  import React from "react";
2  import { BrowserRouter, Route, Switch } from "react-router-dom";
3
4  import NavBar from "../components/NavBar";
5  const Movies = React.lazy(() => import("../components/Movies"));
6  const Users = React.lazy(() => import("../components/Users"));
7  const UserDetails = React.lazy(() => import("../components/UserDetails"));
8
9  const App: React.FC = () => (
10    <BrowserRouter>
11      <div className="container">
12        <NavBar />
13        <Switch>
14          <Route path="/movies">
15            <Movies />
16          </Route>
```



<Suspense />

<Suspense />

- Allows React to **“suspend”** rendering a component subtree
 - Used when a (grand) child component is not ready to be rendered
 - ECMAScript bundle containing the component isn't loaded yet
 - The data needed for a component to render isn't available yet
- The **“fallback” component** will be rendered instead
 - Replaces the complete children component tree
- Rendering is suspended when **a promise is thrown**
 - And resumed when the promise resolves

<Suspense />



```
TS index.tsx ×
src > TS index.tsx
11 ReactDOM.render(
12   <React.StrictMode>
13     <React.Suspense fallback={<Loading />}> ←
14       <ErrorBoundary
15         fallbackRender={({ error, componentStack }) => (
16           <ErrorDisplay error={error} componentStack={componentStack} />
17         )}
18       >
19       <App />
20     </ErrorBoundary>
21   </React.Suspense>
22 </React.StrictMode>,
23   document.getElementById("root")
24 );
```




Nesting `<Suspense />`

Nesting <Suspense />

- Multiple suspense components **can be nested**
- React will use **the closest parent** <Suspense /> component
 - Very useful to control what part of the UI is replaced by a fallback

Nesting <Suspense />



```
TS App.tsx  X
src > TS App.tsx > ...
14  const App: React.FC = () => (
15    <BrowserRouter>
16      <div className="container">
17        <NavBar />
18        <Suspense fallback={<Loading />}>
19          <Switch>
20            <Route path="/movies">
21              <Movies />
22            </Route>
23            <Route path="/users">
24              <Users />
25            </Route>
26            <Route path="/user/:userId/movie/:movieId">
27              <UserDetails />
28            </Route>
29          </Switch>
30        </Suspense>
31      </div>
32    </BrowserRouter>
```



Data Resources

Data Resources

- **Data resources** are functions that
 - Return data that is available
 - Throw a promise when data is not available yet
 - Resolve when that data becomes available
 - Reject if there is some failure
- How the data is requested is not important
 - Just an implementation detail
- Several data loading libraries already **support this approach**
 - useSWR(), react-async and Relay
- Workshop [createResource\(\)](#) based on gist by **Ryan Florence**

Creating a Resource

TS resource.ts ✕

src > components > TS resource.ts > ...

You, 4 days ago | 1 author (You)

```
1 import { createFetchResource } from "../utils/createResource";  
2  
3 const resource = createFetchResource();  
4  
5 export default resource;
```

Using a Resource

```
TS Movies.tsx ×
src > components > TS Movies.tsx > ...
1  import React from "react";
2
3  import MovieCard from "../MovieCard";
4  import resource from "../resource";
5
6  const Movies: React.FC = () => {
7    const data = resource.read<Movie[]>(
8      "https://the-problem-solver-sample-data.azurewebsites.net/top-rated-movies"
9    );
10
11    return (
12      <div className="row row-cols-1 row-cols-sm-2 row-cols-md-3">
13        {data.map((movie) => (
14          <MovieCard key={movie.id} movie={movie} />
15        ))}
16      </div>
17    );
18  };

```



Implement the Resource

- **Replace the `useAbortableFetch()` hook** and apply `resource.read()` in:
 - `.\src\components\Movies.tsx`
 - `.\src\components\Users.tsx`
 - `.\src\components\UserMovieDetails.tsx`
 - `.\src\components\UserUserDetails.tsx`





`<Suspense />` & Errors

<Suspense /> & Errors

- If a suspense resource **fails** to load an **error is thrown**
 - When requesting it during the render()
- Catch the error using an **ErrorBoundary**
 - Just like other runtime errors in React lifecycle functions
- Error boundaries **can be nested**
 - Just like suspense boundaries

<Suspense /> & Errors

```
TS index.tsx ×
src > TS index.tsx
10  ReactDOM.render(
11    <React.StrictMode>
12    <ErrorBoundary
13      fallbackRender={({ error, componentStack }) => (
14        <ErrorDisplay error={error} componentStack={componentStack} />
15      )}
16    >
17      <App />
18    </ErrorBoundary>
19  </React.StrictMode>,
20  document.getElementById("root")
21 );
```





Parallel <Suspense />

Parallel <Suspense />

- Multiple suspense boundaries can **suspend in parallel**
 - React will suspend them all and show multiple fallback components
- If you want to **render a component while others are still loading**
- **Multiple suspending components** in a **single <Suspense/>** is also fine
 - Will resume when all resource promises are resolved

Parallel
<Suspense />



```
TS UserDetails.tsx ×
src > components > TS UserDetails.tsx > ...
6  const UserDetails: React.FC = () => (
7      <
8      <Suspense fallback={<Loading />}>
9          <UserUserDetails />
10      </Suspense>
11      <Suspense fallback={<Loading />}>
12          <UserMovieDetails />
13      </Suspense>
14  </>
15  );
```




User Interface Approaches

Fetch from render

- Typically a component **requests its data when it first renders**
 - Then displays a fallback component
 - And re-renders when the data is available

Start fetching before render

- Sometimes we can **predict data requirements** before render
 - And start the request before the component is first rendered
 - Possibly the data is already loaded when the component renders
- For example when a **user hovers over a navigation link**
-  Beware: Can lead to extra AJAX requests
 - Not every hover is followed by a click on the link

Start fetching
before render

```
TS NavBar.tsx X
src > components > TS NavBar.tsx > ...
5  const NavBar: React.FC = () => (
6    <nav className="navbar navbar-light bg-light navbar-expand">
7      <Link to="/" className="navbar-brand mb-0 h1">
8        Home
9      </Link>
10     <div className="navbar-nav">
11       <resource.NavLink
12         to="/movies"
13         className="nav-link"
14         activeClassName="active"
15         cacheKeys="https://the-problem-solver-sample-data.azurewebsites.net/top-rated-movies"
16       >
17         Movies
18     </resource.NavLink>
19     <resource.NavLink
20       to="/users"
21       className="nav-link"
22       activeClassName="active"
23       cacheKeys="https://the-problem-solver-sample-data.azurewebsites.net/accounts?sleep=1000"
24     >
25       Users
26     </resource.NavLink>
27   </div>
28 </nav>
29 );
```

Start fetching
before render

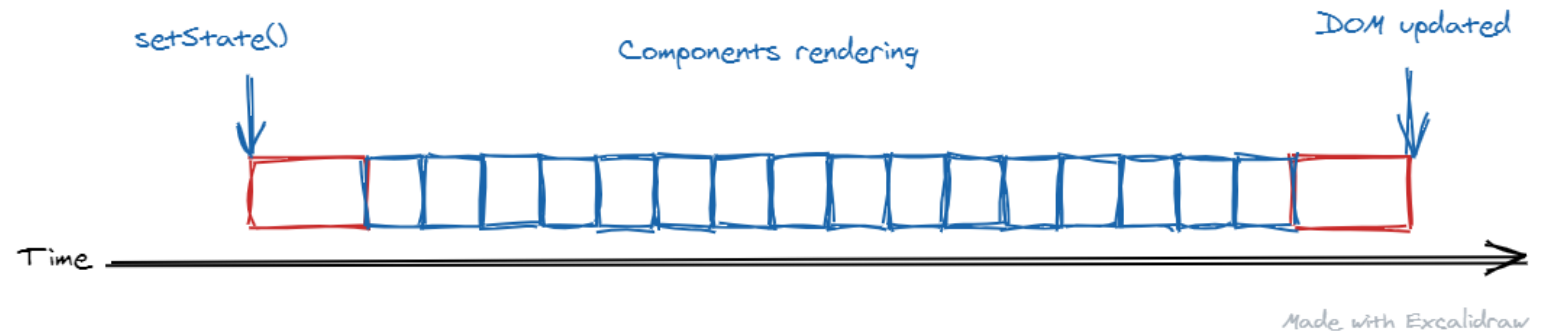


```
TS UserCard.tsx X
src > components > TS UserCard.tsx > ...
6  const UserCard: React.FC<Props> = ({ user }) => (
7    <div className="col mt-3">
8      <div className="card h-100">
9        <img src={user.picture} className="card-img-top" alt={user.firstname} />
10       <div className="card-body">
11         <h5 className="card-title text-truncate">`${user.firstname} ${user.surname}`</h5>
12         <p className="card-text">{user.email}</p>
13         <p className="card-text">{user.phone}</p>
14       </div>
15       <div className="card-footer">
16         <resource.Link
17           to={`/user/${user.id}/movie/${user.favorite_movie}`}
18           className="btn btn-primary"
19           cacheKeys={[
20             `https://the-problem-solver-sample-data.azurewebsites.net/accounts/${user.id}?sleep=2000`,
21             `https://the-problem-solver-sample-data.azurewebsites.net/top-rated-movies/${user.favorite_movie}?sleep=1000`,
22           ]}
23         >
24           Details
25         </resource.Link>
26       </div>
27     </div>
28   </div>
29 );
```

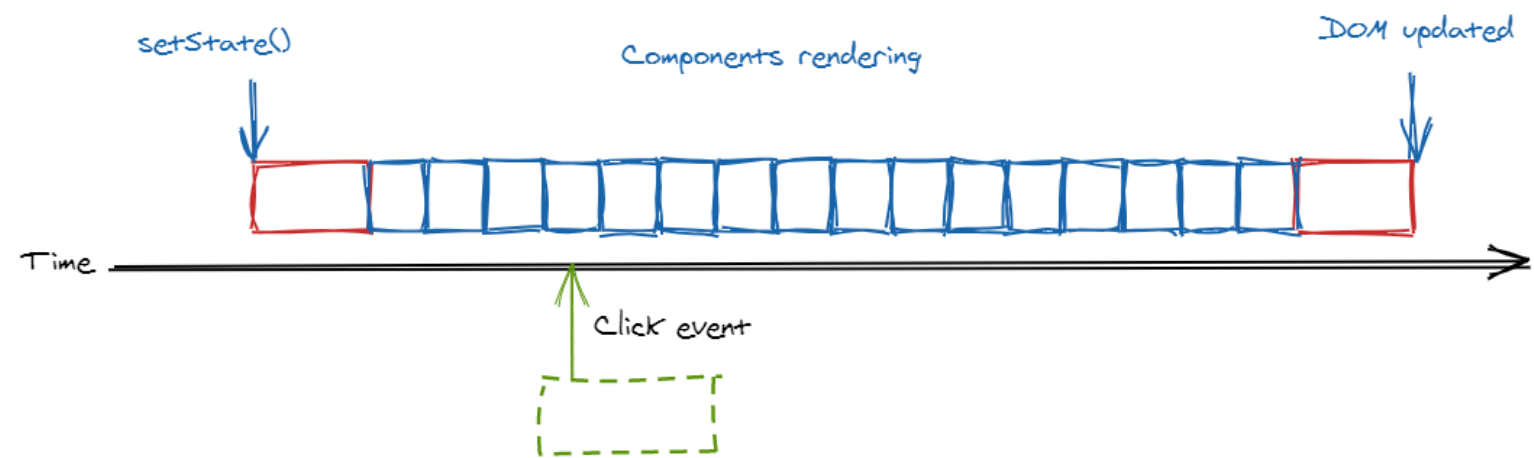


Concurrent Mode

React rendering component

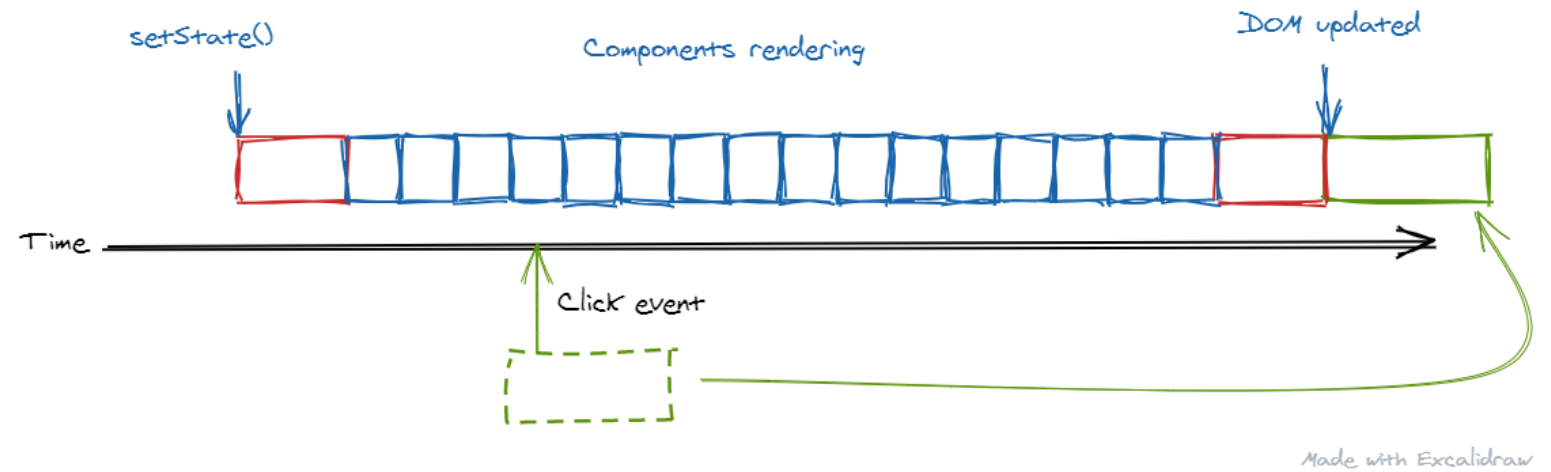


Use click event

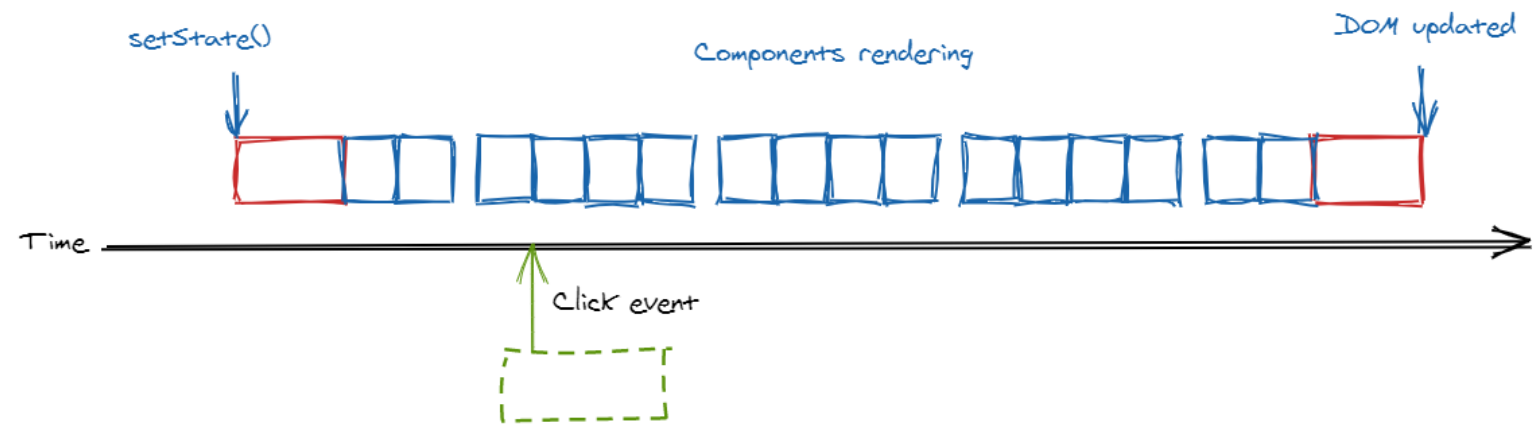


Made with Excalidraw

Event running

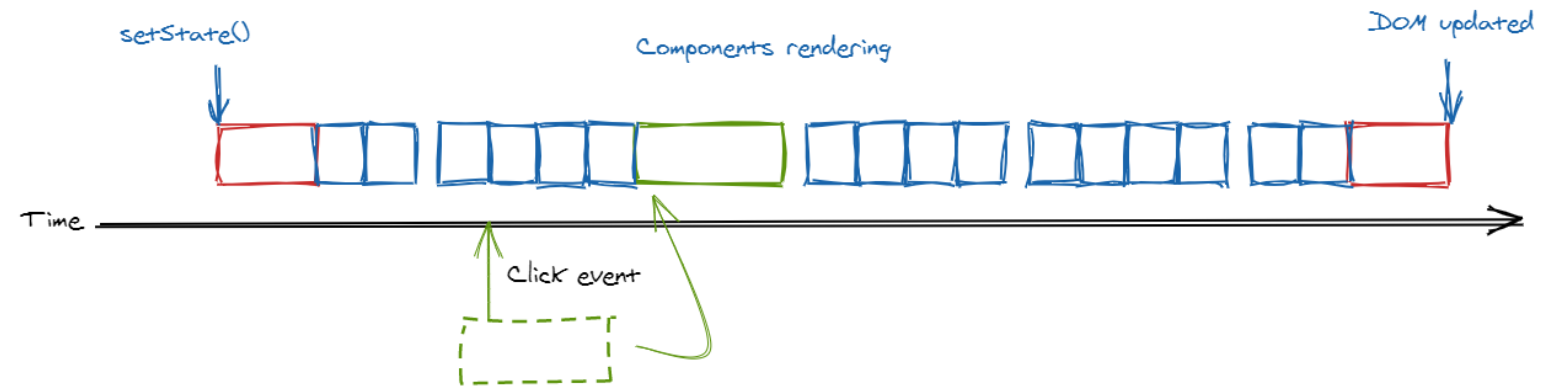


Concurrent mode



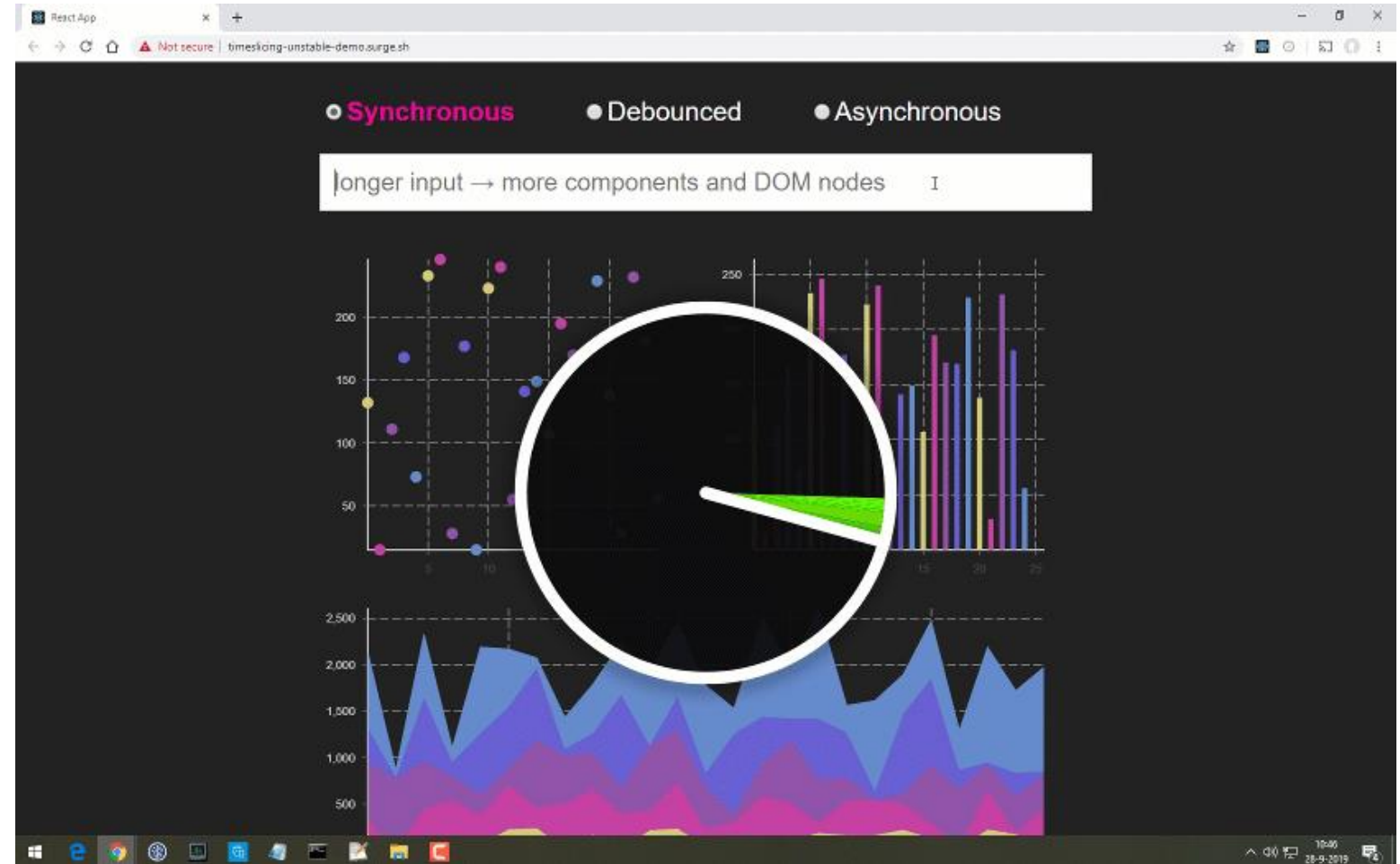
Made with Excalidraw

Event running with concurrent mode



Made with Excalidraw

Dan Abramov's 2018 Demo



Concurrent
Mode



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"	✗	✗	✓

Concurrent Mode

- **Legacy is the normal way** of rendering a React application
 - The only option that is fully supported
- **Concurrent mode** is the goal
 - Only available in experimental builds of React
- **Blocking mode** is an intermediate mode
 - Only available in experimental builds of React
- Use the **experimental type definitions** with TypeScript
- *npm install react@experimental react-dom@experimental*

Use the Experimental React

```
{} package.json ×
{} package.json > {} scripts
4   "dependencies": {
5     "bootstrap": "4.5.2",
6     "react": "0.0.0-experimental-94c0244ba",
7     "react-dom": "0.0.0-experimental-94c0244ba",
8     "react-error-boundary": "2.3.1",
9     "react-router-dom": "5.2.0",
10    "use-abortable-fetch": "2.6.14"
11  },
```



TypeScript Definitions

```
{ } tsconfig.json X
{ } tsconfig.json > ...
You, an hour ago | 1 author (You)
1 {
2   "compilerOptions": {
3     "target": "es5",
4     "lib": ["dom", "dom.iterable", "esnext"],
5     "allowJs": true,
6     "skipLibCheck": true,
7     "esModuleInterop": true,
8     "allowSyntheticDefaultImports": true,
9     "strict": true,
10    "forceConsistentCasingInFileNames": true,
11    "module": "esnext",
12    "moduleResolution": "node",
13    "resolveJsonModule": true,
14    "isolatedModules": true,
15    "noEmit": true,
16    "jsx": "react",
17    "types": ["react-dom/experimental", "react/experimental"]
18  },
19  "include": ["src"]
20 }
```

Concurrent Mode

- Use **ReactDOM.createRoot()** to create a root object
 - And use it to render the React application
- Note: Still has the “**unstable_**” prefix

Using createRoot()



```
TS index.tsx ×
src > TS index.tsx
10
11 ReactDOM.unstable_createRoot(document.getElementById("root")!).render(
12   <React.StrictMode>
13     <Suspense fallback={<Loading />}>
14       <ErrorBoundary
15         fallbackRender={({ error, componentStack }) => (
16           <ErrorDisplay error={error} componentStack={componentStack} />
17         )}
18       >
19         <App />
20       </ErrorBoundary>
21     </Suspense>
22   </React.StrictMode>
23 );
```

Using `<SuspenseList />`

Orchestrating `<Suspense />` boundaries

Using <SuspenseList />

- **<SuspenseList />** will let you control how multiple **<Suspense />** components **render their fallback**
 - The order in which child components show when ready
 - If multiple child fallbacks components are displayed

Using
<SuspenseList />



```
TS UserDetails.tsx X
src > components > TS UserDetails.tsx > ...
5
6  const UserDetails: React.FC = () => (
7  →  <React.unstable_SuspenseList revealOrder="forwards" tail="collapsed">
8      <Suspense fallback={<Loading />}>
9          <UserUserDetails />
10     </Suspense>
11     <Suspense fallback={<Loading />}>
12         <UserMovieDetails />
13     </Suspense>
14 </React.unstable_SuspenseList>
15 );
```



Using `useTransition()`

To prioritize work



Using useTransition()

- The **useTransition() hook** can be used to control how React renders when components suspend
 - Prevent the fallback component being rendered immediately
- The **new components will be rendered** when:
 - Their resources are ready
 - The timeout is expired
- The “old” UI can use the **isPending** state when rendering

Using useTransition()



```
TS UserUserDetails.tsx X
src > components > TS UserUserDetails.tsx > ...
11  const UserUserDetails: React.FC = () => {
12    const history = useHistory();
13    const { userId } = useParams<RouteParams>();
14    const data = resource.read<User>(
15      `https://the-problem-solver-sample-data.azurewebsites.net/accounts/${userId}?sleep=2000`
16    );
17
18    const [startTransition, isPending] = React.unstable_useTransition({
19      timeoutMs: 5000,
20    });
21
22    return (
23      <div>
24        <div>
25          <button
26            className="btn btn-primary float-right"
27            onClick={() => {
28              startTransition(() => {
29                history.push("/users");
30              });
31            }}
32          >
33            Back to users
34            {isPending && (
35              <span
36                className="spinner-border spinner-border-sm ml-3"
37                role="status"
38              />
39            )}
40          </button>
41        </div>
```

Conclusion

- You can use **<Suspense />** today
 - Suspend when lazily loading components
 - Suspend when fetching data
 - When not doing server side rendering
- Better UI **approaches** can make your application more responsive
 - Start fetching data before you render the component
 - It's a trade off
- **Concurrent mode**
 - Not recommended for production applications today
 - Can make large applications more responsive

Maurice de Beijer

@mauricedb

maurice.de.beijer
@gmail.com

