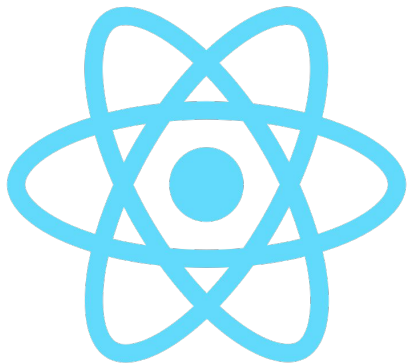
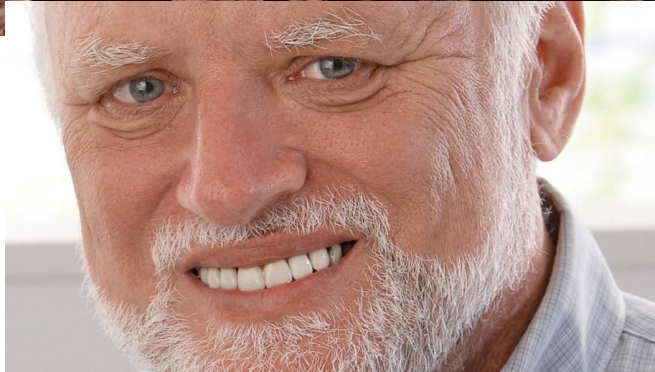
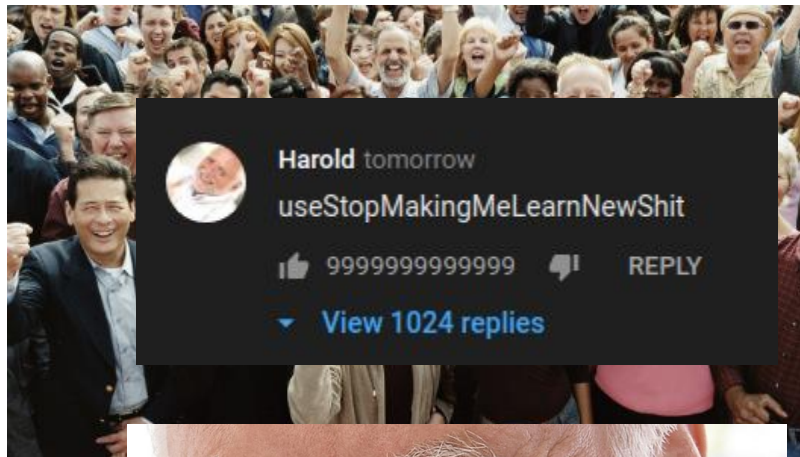


React Concurrent Mode






Появление новых инструментов в библиотеке



- Параллельный режим
- Конкурирующий режим
- Конкурентный режим
- Одновременный режим
- Совместный режим
- Согласованный режим
- Многозадачный режим

Конкурентный режим





 **gcor** commented 24 days ago Contributor + 😊 ...



Варианты:

- Параллельный режим
- Конкурирующий режим
- Конкурентный режим
- Одновременный режим

Первый вариант переводят все популярные переводчики. Плюс в википедии есть статья про параллельные вычисления (Concurrent computing).


👁 1

  **gcor** added **question** **translation-agreements** labels 24 days ago


  **gcor** mentioned this issue 24 days ago

Перевод раздела Concurrent Mode #422 🔓 Open

📋 0 of 5 tasks complete

 **gaearon** commented 24 days ago Member + 😊 ...

Параллельный близко по смыслу, но вносит путаницу с полным параллелизмом. Который может быть когда-нибудь и появиться в будущем.

 **gaearon** commented 24 days ago • edited Member + 😊 ...

Вброшу еще пару вариантов:

- Совместный режим
- Согласованный режим
- Многозадачный режим

Что такое конкурентный режим?



```
yarn add react@experimental react-dom@experimental
```

Таймлайн получения данных для компонента





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

class Example extends Component {
  state = {
    data: [],
  }

  componentDidMount() {
    fetchData()
      .then(data => this.setState({ data }))
      .catch(console.warn)
  }

  render() {
    return (
      // ...
    )
  }
}
```



- Монтируем компонент
- Запрашиваем данные
- Получаем данные
- Рендерим компонент с данными



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

const Example = () => {
  const [data, setData] = useState([])

  useEffect(() => {
    fetchData()
      .then(setData)
      .catch(console.warn)
  }, [])

  return (
    // ...
  )
}
```



- Монтируем компонент
- Запрашиваем данные
- Получаем данные
- Рендерим компонент с данными

```
import Example from "@/components/Example"

export default {
  components: {
    Example
```



DAYS

**SINCE LAST NEW
JAVASCRIPT FRAMEWORK**

```
.catch(console.warn)
}
}
}
```



- Монтируем компонент
- Запрашиваем данные
- Получаем данные
- Рендерим компонент с данными

X framework компонент
Vue компонент



Мы можем лучше!

```
function fetchProfileData() {  
  return Promise.all([  
    fetchUser(),  
    fetchPosts()  
  ]).then(([user, posts]) => {  
    return {user, posts};  
  })  
}
```

```
const promise = fetchProfileData()  
  
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 profile...</p>  
  
  return (  
    <>  
      <h1>{user.name}</h1>  
      <ProfileTimeline posts={posts} />  
    </>  
  )  
}  
  
function ProfileTimeline({ posts }) {  
  if (posts === null) return <h2>Loading posts...</h2>  
  
  return (  
    <ul>  
      {posts.map(post => (  
        <li key={post.id}>{post.text}</li>  
      ))}  
    </ul>  
  )  
}
```



Резолв промиса только после получения всех данных



Если попытаться убрать Promise.all(), мы получим сложности при масштабировании



Suspense (Задержка)

**experimental React Concurrent mode*

“

Изначально *Suspense* (Задержка) была добавлена в версии *React* 16.6. Использование этого компонента даёт возможность “подождать” пока код загрузится. Пока это не произошло - отображается *fallback*.

До появления *Suspense* уже существовали сторонние модули для решения похожей задачи, например ***react-loadable***.

```
import React, { lazy, Suspense } from 'react'
import Spinner from './Spinner'
const ProfilePage = lazy(() => import('./ProfilePage')) // Lazy-loaded

const CurrentSuspenseExample = () => (
  <Suspense fallback={<Spinner />}>
    <ProfilePage />
  </Suspense>
)

export default CurrentSuspenseExample
```



Новая версия Suspense может больше!

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

const resource = fetchProfileData()

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

function ProfileDetails() {
  const user = resource.user.read()
  return <h1>{user.name}</h1>
}

function ProfileTimeline() {
  const posts = resource.posts.read()
  return (
    <ul>
      {posts.map(post => (
        <li key={post.id}>{post.text}</li>
      ))}
    </ul>
  )
}
```

Запрашиваем данные и сразу же начинаем рендеринг

Дожидаемся получения данных





Что такое resource?

```
export function fetchProfileData() {  
  const userPromise = fetchUser()  
  const postsPromise = fetchPosts()  
  
  return {  
    user: wrapPromise(userPromise),  
    posts: wrapPromise(postsPromise)  
  }  
}
```

```
function wrapPromise(promise) {  
  let status = 'pending'  
  let result  
  const suspender = promise.then(  
    r => {  
      status = 'success'  
      result = r  
    },  
    e => {  
      status = 'error'  
      result = e  
    }  
  )  
  
  return {  
    read() {  
      if (status === 'pending') {  
        throw suspender  
      } else if (status === 'error') {  
        throw result  
      } else if (status === 'success') {  
        return result  
      }  
    }  
  }  
}
```



Подписываемся на резолв
промиса из параметра



Возвращаем resource (объект с
единственным методом .read())



Альтернатива для .catch() из промисов

```
class ErrorBoundary extends React.Component {
  state = { hasError: false, error: null }
  static getDerivedStateFromError(error) {
    return {
      hasError: true,
      error
    }
  }
  render() {
    if (this.state.hasError) {
      return this.props.fallback
    }
    return this.props.children
  }
}
```

```
function ProfilePage() {
  return (
    <Suspense fallback=<h1>Loading profile...</h1>>
      <ProfileDetails />
      <ErrorBoundary fallback=<h2>Could not fetch posts.</h2>>
        <Suspense fallback=<h1>Loading posts...</h1>>
          <ProfileTimeline />
        </Suspense>
      </ErrorBoundary>
    </Suspense>
  )
}
```

“

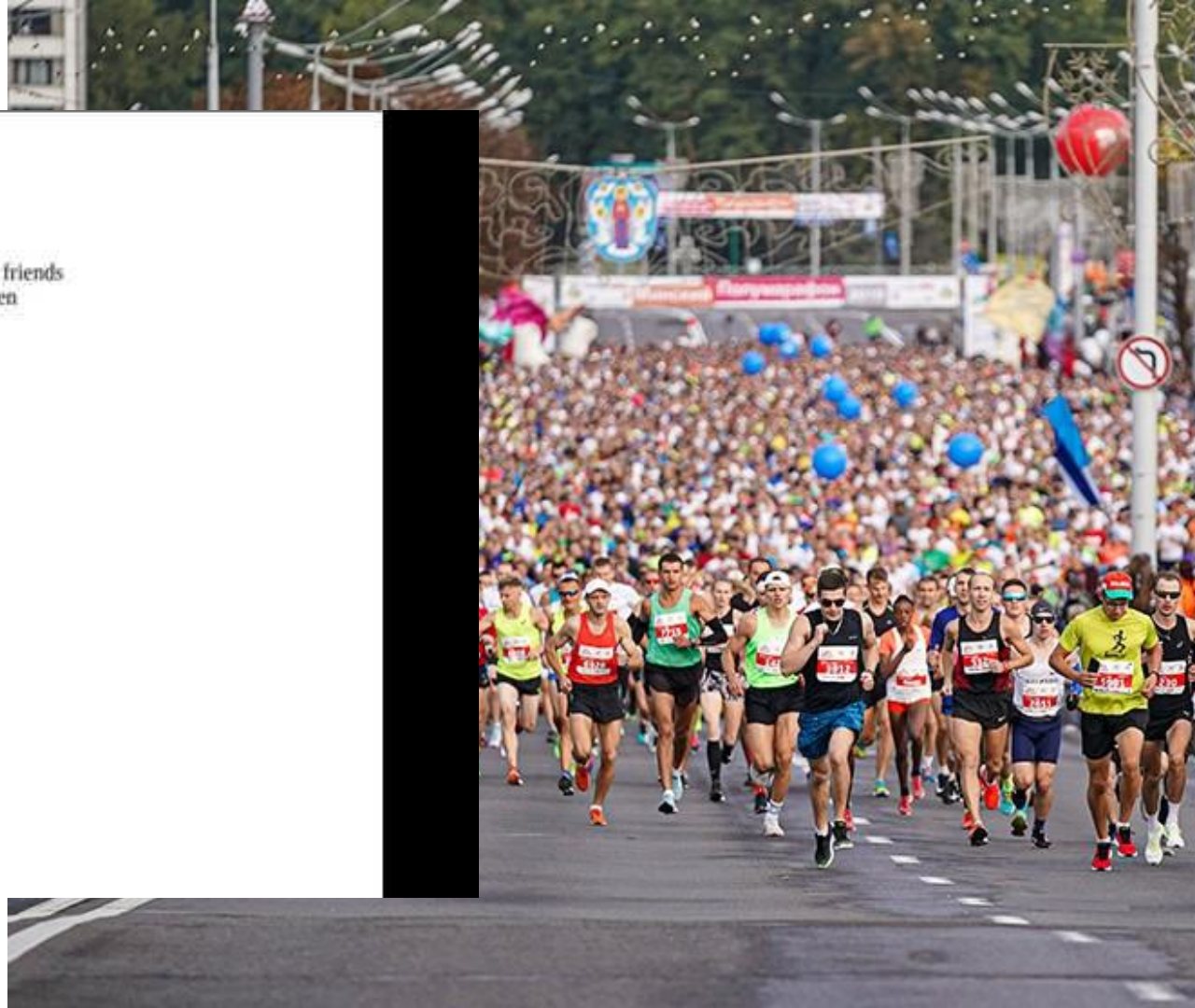
*Concurrent Mode:
Declarative vs Imperative*

Race conditions

Next

Paul McCartney

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




```
function ProfilePage({ id }) {  
  const [user, setUser] = useState(null)  
  
  useEffect(() => {  
    fetchUser(id).then(u => setUser(u))  
  })  
}
```

```
useEffect(() => {  
  fetchUser(id).then(setUser)  
}, [id])
```

```
{posts.map(post => (  
  <li key={post.id}>{post.text}</li>  
))}  
</ul>  
)  
}
```



```
const getNextId = () => {  
  //...  
}  
  
const [resource, setResource] = useState(fetchProfileData(0))  
  
return (  
  <>  
    <button  
      onClick={() => {  
        const nextUserId = getNextId(resource.userId)  
        setResource(fetchProfileData(nextUserId))  
      }}  
    >  
      Next  
    </button>  
    <ProfilePage resource={resource} />  
  </>  
)
```



Вместо того чтобы обновлять состояние после получения данных, мы обновляем данные сразу же после создания запроса.



useTransition hook

**experimental React Concurrent mode*

Slow 3G ☒ Use transition hook 2000 User name fetch time 2000 User facts fetch time

Loading...

Юлия Бухвалова

👤 Разработчица в LiveJournal
👤 Автор `Ignora Pro CSS`
👤 Любит экспериментировать, открывать новое в SVG и создавать инструменты

Пример использования *useTransition* для перехода на новый экран.

<http://wsd-react-concurrent-mode.herokuapp.com/>



Сравним 2 примера

Обычный сценарий

При клике на Next мы сразу же попадаем на новый экран и ожидаем появления данных.

useTransition

При клике на Next мы запускаем загрузку данных, но остаёмся на предыдущем экране до завершения запроса (или истечения таймаута).



useTransition

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

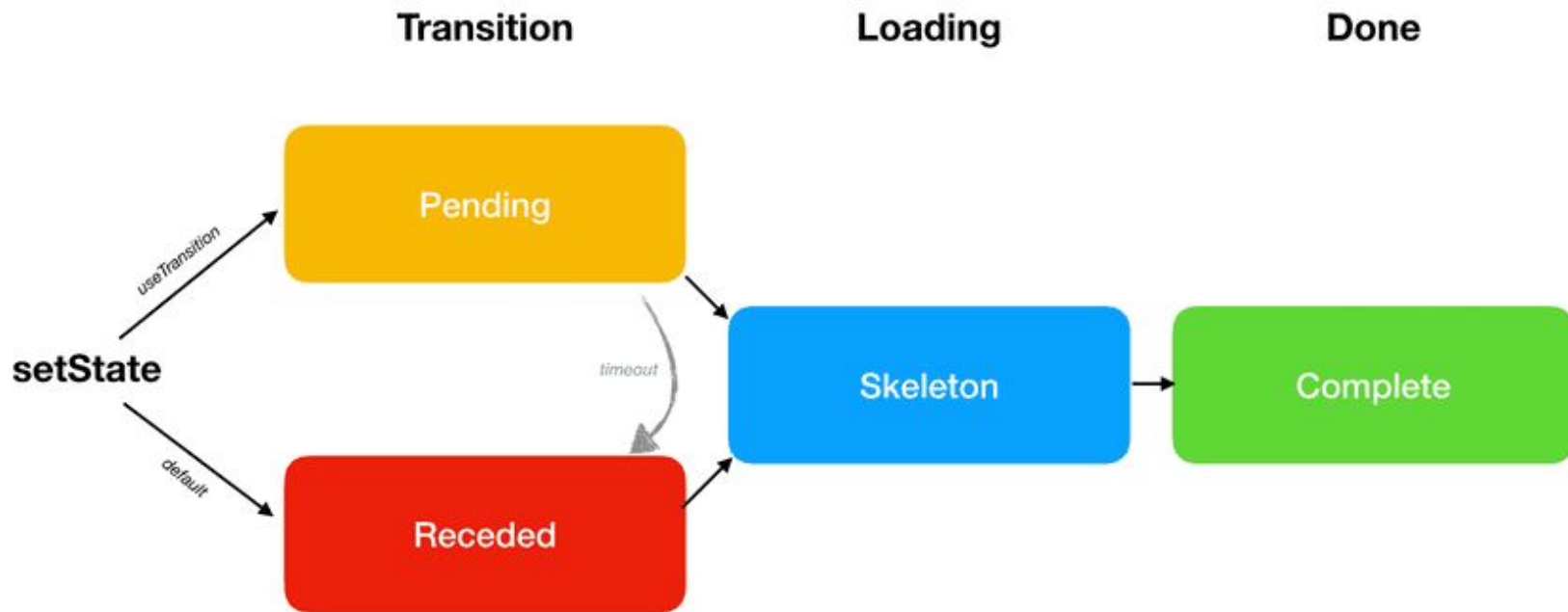
function App() {
  const [resource, setResource] = useState(initialResource)
  const [startTransition, isPending] = useTransition({
    timeoutMs: 3000
  })
  // ...
}
```



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

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

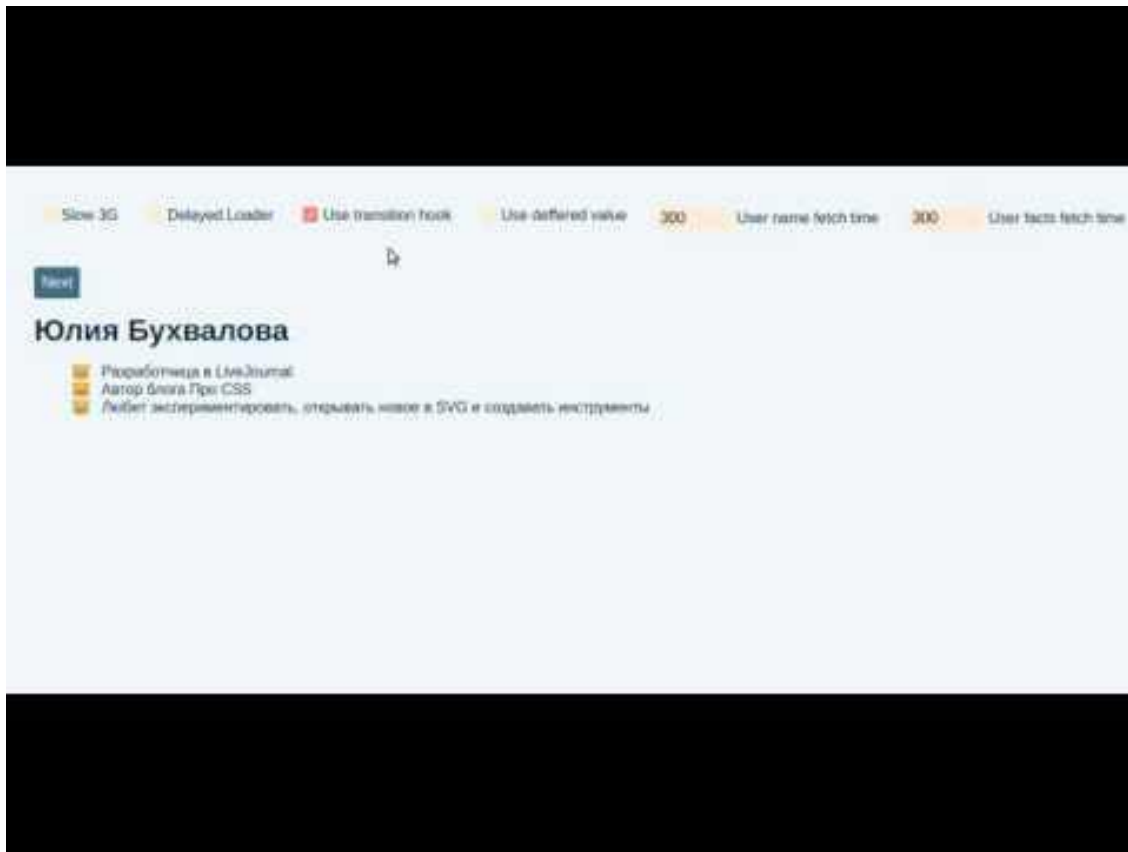
Шаги обновления



<https://reactjs.org/docs/concurrent-mode-patterns.html>



Задержка индикатора ожидания





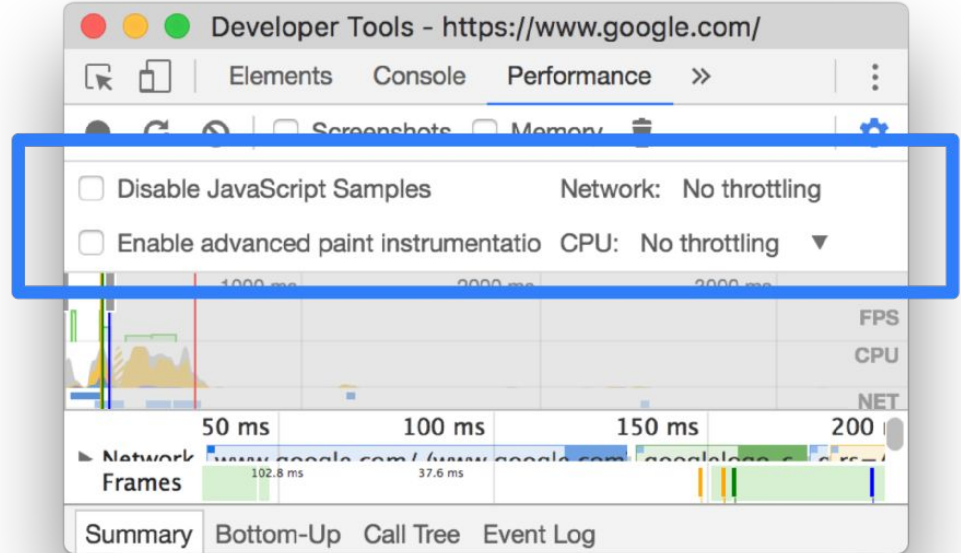
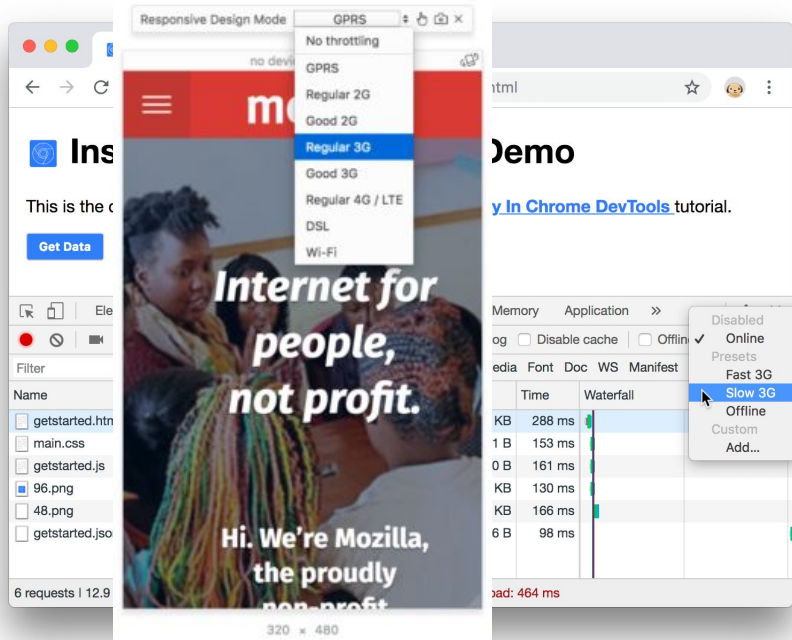
Задержка индикатора ожидания

```
.DelayedSpinner {  
  animation: 0s linear 0.5s forwards makeVisible;  
  visibility: hidden;  
}  
  
@keyframes makeVisible {  
  to {  
    visibility: visible;  
  }  
}
```

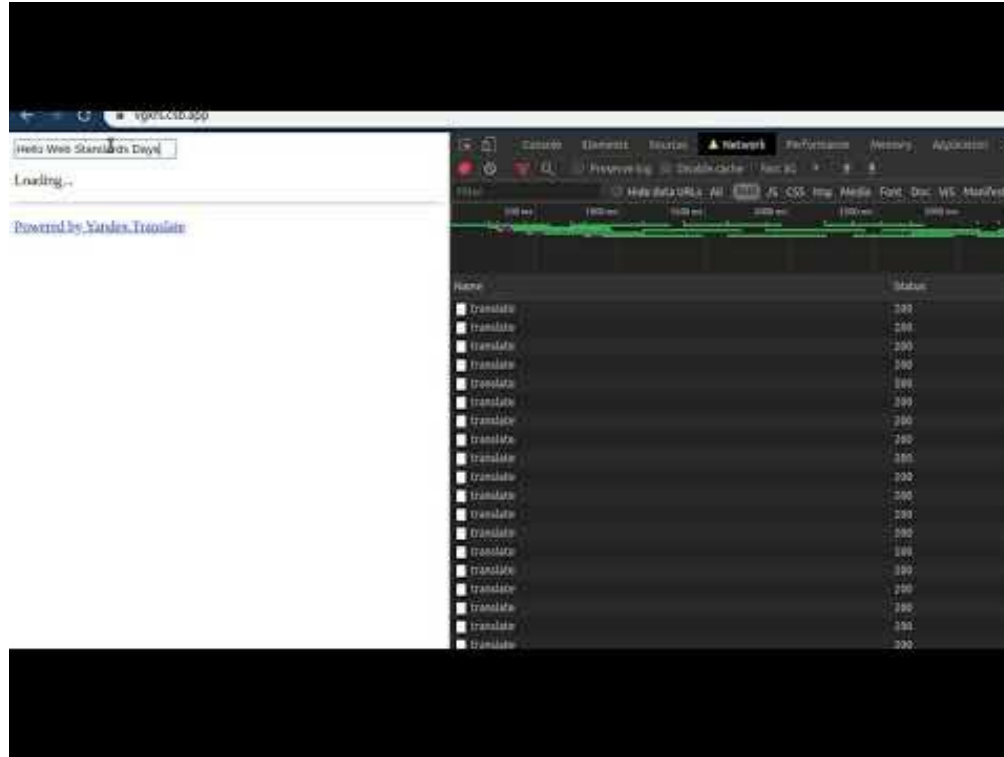


Совет

Используйте инструменты браузеров для эмуляции дополнительной нагрузки на сеть и CPU.



Разделение на “важное” и “второстепенное” обновление данных



```
const initialQuery = 'Hello WSD!'
```

```
function App() {  
  const [query, setQuery] = useSt  
  const [resource, setResource] =
```

```
  const handleChange = e => {  
    const value = e.target.value  
    setQuery(value)  
    setResource(fetchTranslation(  
  })
```

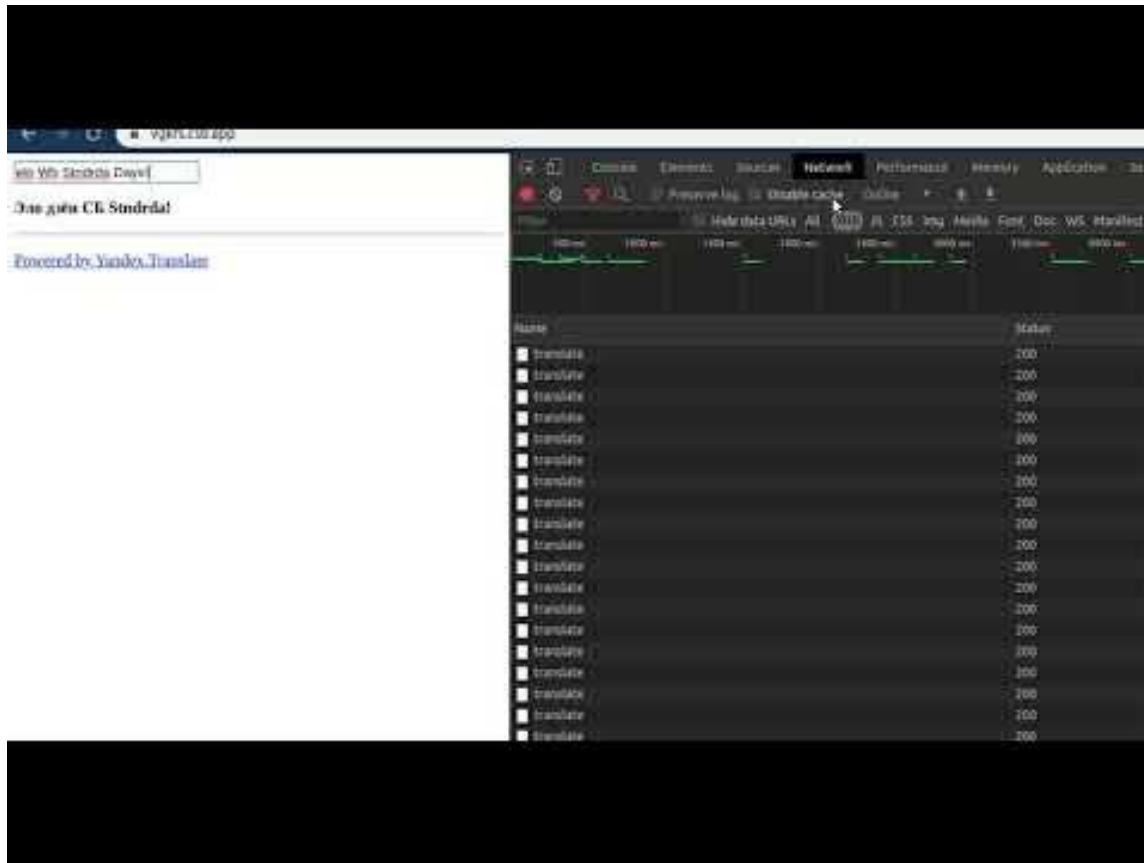
```
  return (  
    <>  
      <input value={query} onChange={handleChange} />  
      <Suspense fallback={<p>Loading</p>}>  
        <Translation resource={resource} />  
      </Suspense>  
    </>  
  )  
}
```

```
const handleChange = e => {  
  const value = e.target.value  
  setQuery(value)  
  setResource(fetchTranslation(value))  
}
```







```
  fetchTranslation({ resource }) {
```

```
    e.read()}</b>
```

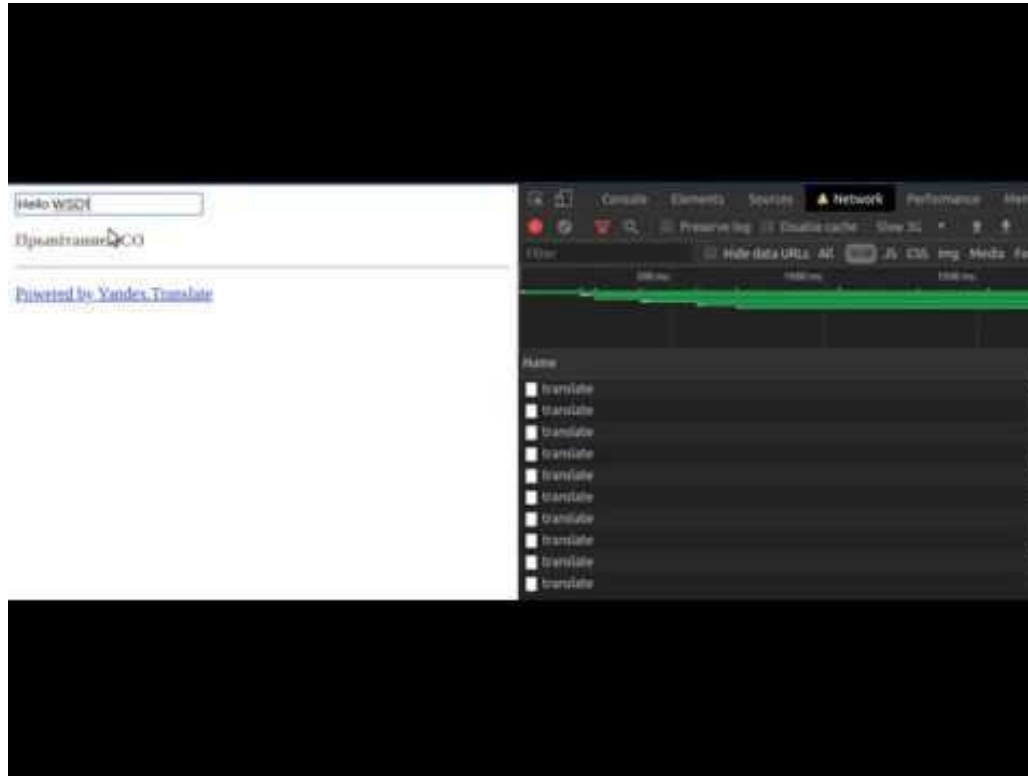
useTransition всё сломал 🙄



Разделение на “важное” и “второстепенное” обновление данных

```
fun     
  c  
    function handleChange(e) {  
      const value = e.target.value  
      /    (e)  
      s  
        startTransition(() => {  
          setQuery(value)  
          s  
            setResource(fetchTranslation(value))  
        })  
      }  
    }  
  }
```

Happy days 🐱



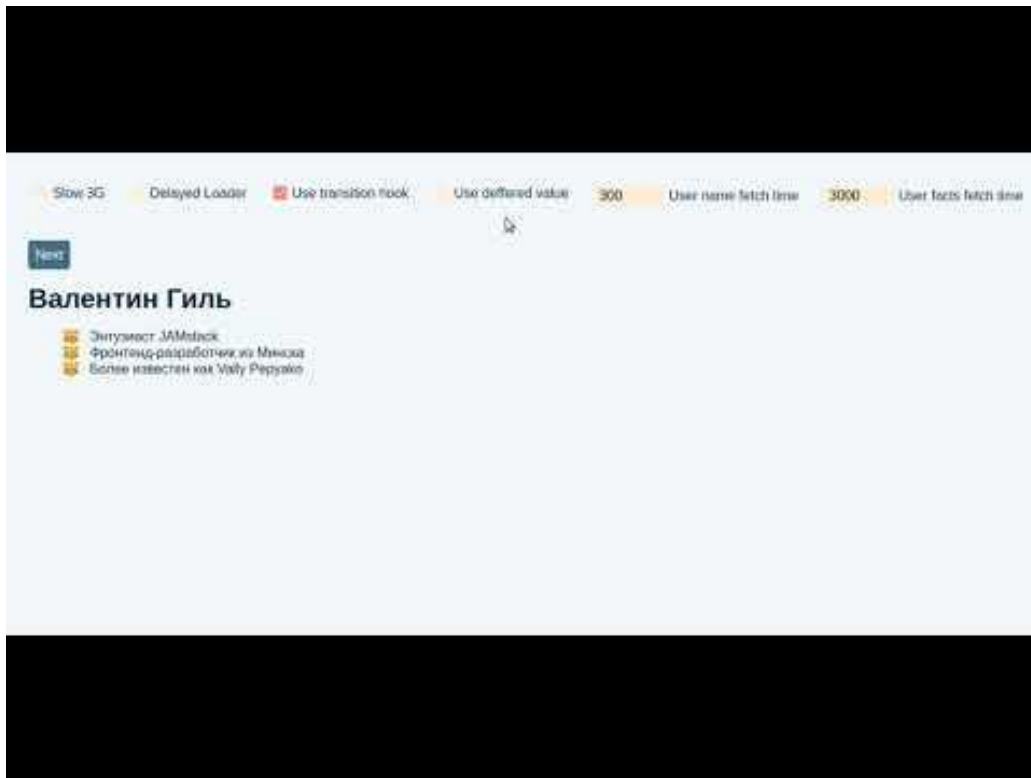


useDeferredValue

**experimental React Concurrent mode*

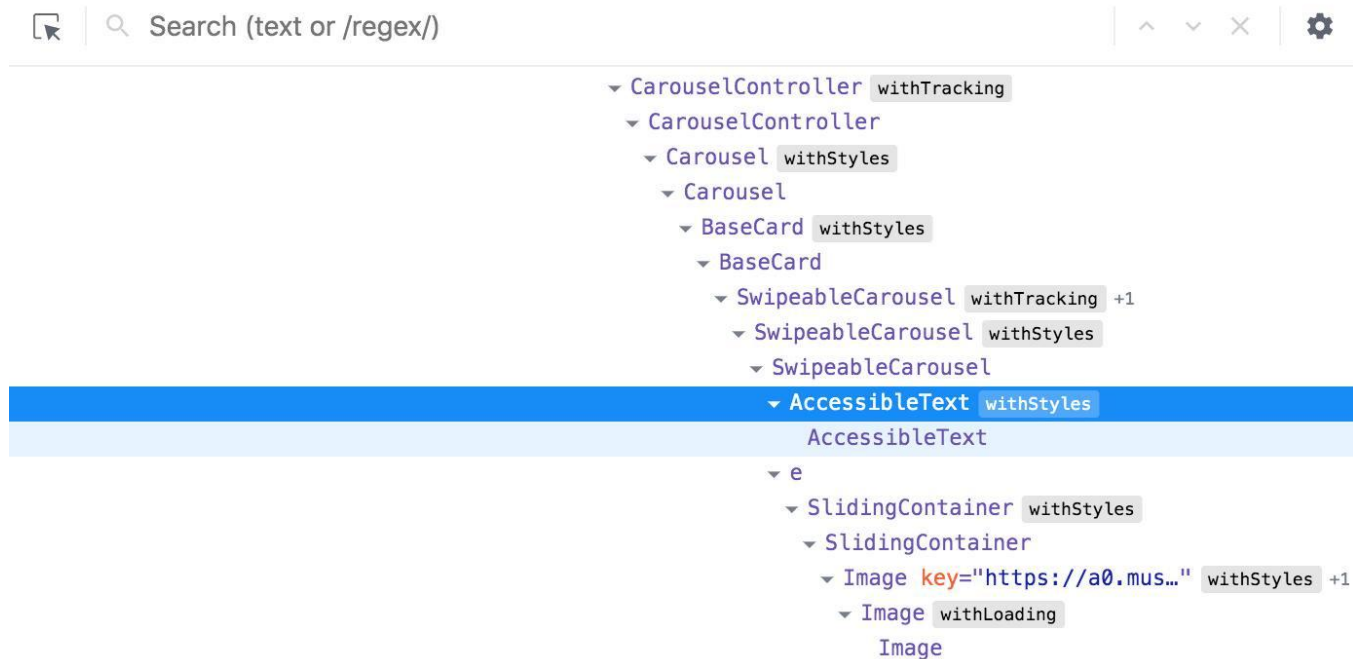


useDeferredValue для фетчинга данных





Использование *useDeferredValue* позволяет приложению оставаться отзывчивым при операциях с большой нагрузкой на DOM.



Пример использования *useDeferredValue* для операций нагружающих DOM

React Without Concurrent Mode

Type into the input:

Each list item in this demo completely blocks the main thread for 3 milliseconds. This makes typing into the input stutter because updates are not interruptible.

Results for "Hello WSD":

Result #0 for "Hello WSD"
Result #1 for "Hello WSD"
Result #2 for "Hello WSD"
Result #3 for "Hello WSD"
Result #4 for "Hello WSD"
Result #5 for "Hello WSD"
Result #6 for "Hello WSD"
Result #7 for "Hello WSD"
Result #8 for "Hello WSD"
Result #9 for "Hello WSD"
Result #10 for "Hello WSD"
Result #11 for "Hello WSD"
Result #12 for "Hello WSD"
Result #13 for "Hello WSD"
Result #14 for "Hello WSD"
Result #15 for "Hello WSD"
Result #16 for "Hello WSD"
Result #17 for "Hello WSD"
Result #18 for "Hello WSD"
Result #19 for "Hello WSD"
Result #20 for "Hello WSD"
Result #21 for "Hello WSD"
Result #22 for "Hello WSD"
Result #23 for "Hello WSD"
Result #24 for "Hello WSD"
Result #25 for "Hello WSD"
Result #26 for "Hello WSD"
Result #27 for "Hello WSD"
Result #28 for "Hello WSD"
Result #29 for "Hello WSD"
Result #30 for "Hello WSD"
Result #31 for "Hello WSD"
Result #32 for "Hello WSD"
Result #33 for "Hello WSD"

SuspenseList

```
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>  
  )  
}
```

Приостановка раскрытия “вагонов”

Иногда на новом экране данные приходящие в *Suspense* поступают через достаточно короткий интервал. Например, два разных ответа могут прийти через *1000 ms* и *1050 ms* соответственно. Если вы уже подождали секунду, дополнительные *50 ms* не будут играть важной роли.

Задержки раскрываются “периодически”, что позволяет уменьшить количество изменений представленных пользователю.



Slow 3G ☒ Use transition hook 1000 : User name fetch time 1050 User facts fetch time

Next

Юлия Бухвалова

- 👩 Разработчица в LiveJournal
- 👩 Автор блога Про CSS
- 👩 Любит экспериментировать, открывать новое в SVG и создавать инструменты



Разделяй и властвуй
свой UI с помощью
Suspense



Совет

Компоненты, которые могут долго получать данные и не представляют критической важности, лучше оборачивать в **Suspense**.

<https://codesandbox.io/s/focused-mountain-uhkzg>



Включение конкурентного режима

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


Что мы рассмотрели сегодня:

- `<Suspense />`
- `<SuspenseList />`
- `useTransition`
- `useDeferredValue`



Полезные ссылки

1. <https://github.com/SmolinPavel/wsd-react-concurrent-mode/blob/master/WSD.pdf>
2. <http://wsd-react-concurrent-mode.herokuapp.com>
3. <https://reactjs.org/docs/concurrent-mode-intro>
4. <https://reactjs.org/blog/2018/03/01/sneak-peek-beyond-react-16>
5. <https://reactjs.org/blog/2019/11/06/building-great-user-experiences-with-concurrent-mode-and-suspense>
6. <http://timeslicing-unstable-demo.surge.sh>



Спасибо за внимание!

Остались вопросы?

Можно связаться со мной:

 email: *isingwithaz@gmail.com*

 twitter: *@isingwithaz*

 github: *SmolinPavel*

 facebook: *pavelsmolinpavel*

 instagram: *smolinp*



Благодарность

Большое спасибо:



Команде KlikaTech



Команде WSD