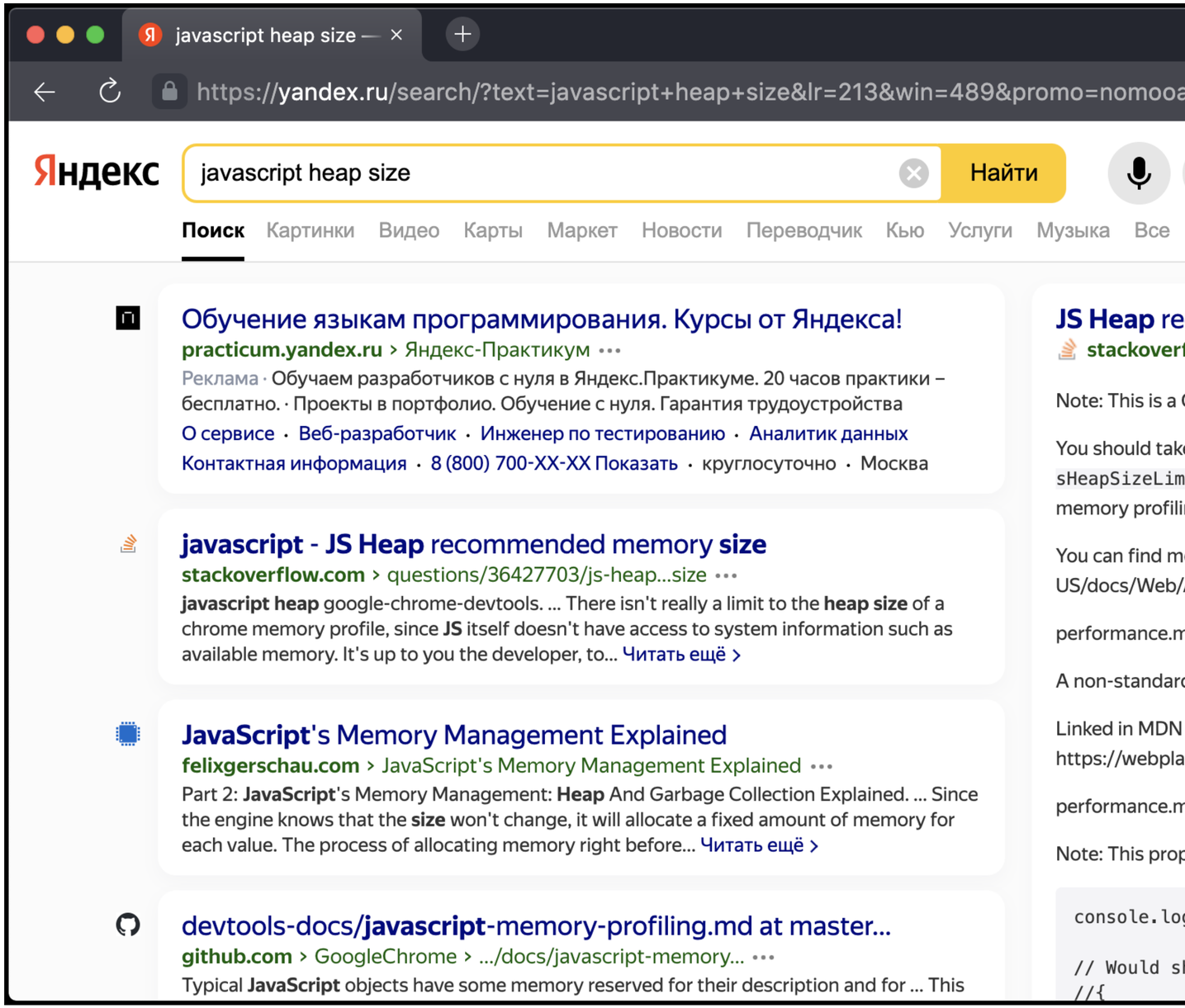
Яндекс

Планировщик задач: не замораживаем страницу

Виктор Хомяков

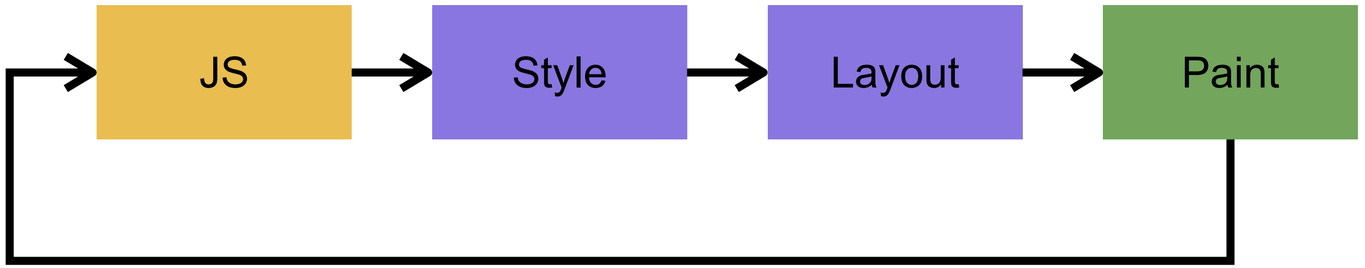
О себе

- 4 года в разработке страницы результатов поиска Яндекса
- команда Скорости



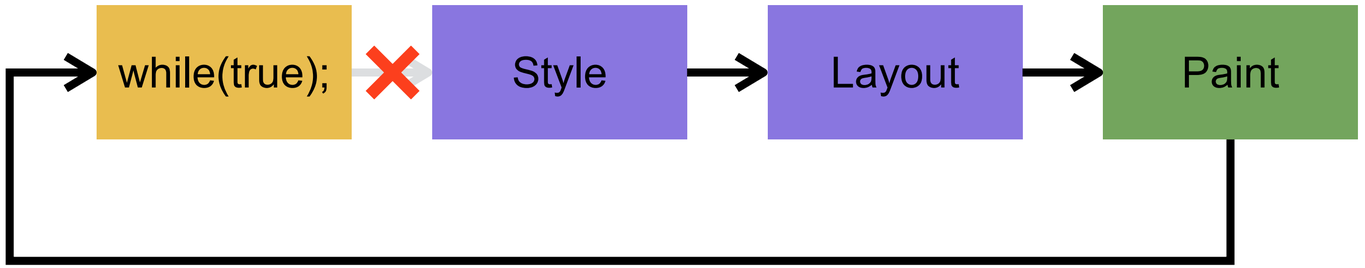
Иван Тулуп, long tasks и TBT

Иван Тулуп (event loop)



https://youtu.be/8aGhZQkoFbQ https://youtu.be/cCOL7MC4PI0

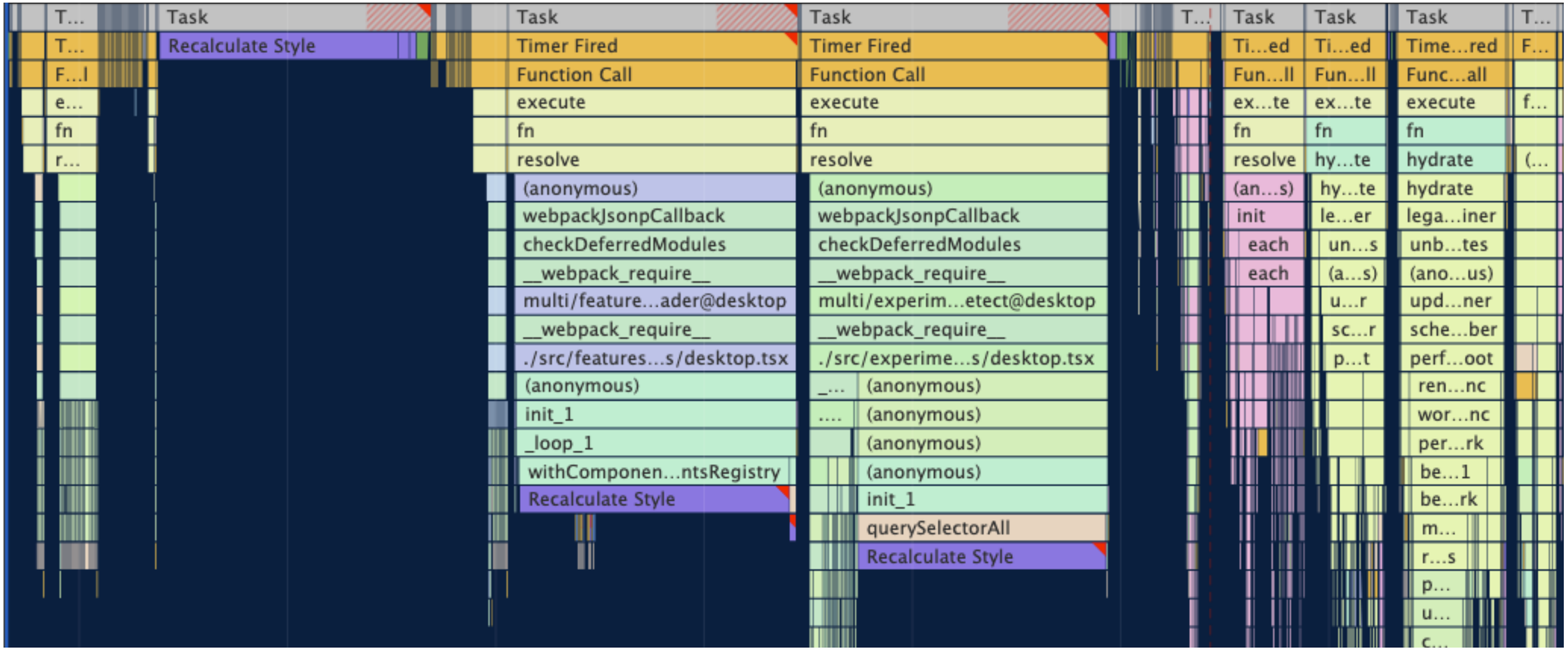
Иван Тулуп (event loop)



https://event-loop-tests.glitch.me/while-true-test.html

long task: >50ms

short task



https://web.dev/long-tasks-devtools/

First Contentful Paint (FCP)

Момент отрисовки первого содержимого DOM:

- текст (span, …)
- изображения (img, canvas, svg)
- не iframe, не фоновые изображения

https://web.dev/first-contentful-paint/

Time to Interactive (TTI)

Момент, когда страница становится *полностью* интерактивной:

- На странице уже видно что-то полезное (был FCP)
- На большинство видимых элементов навешены обработчики
- Страница реагирует на ввод быстрее, чем за 50мс

https://web.dev/interactive/

Total Blocking Time (TBT)

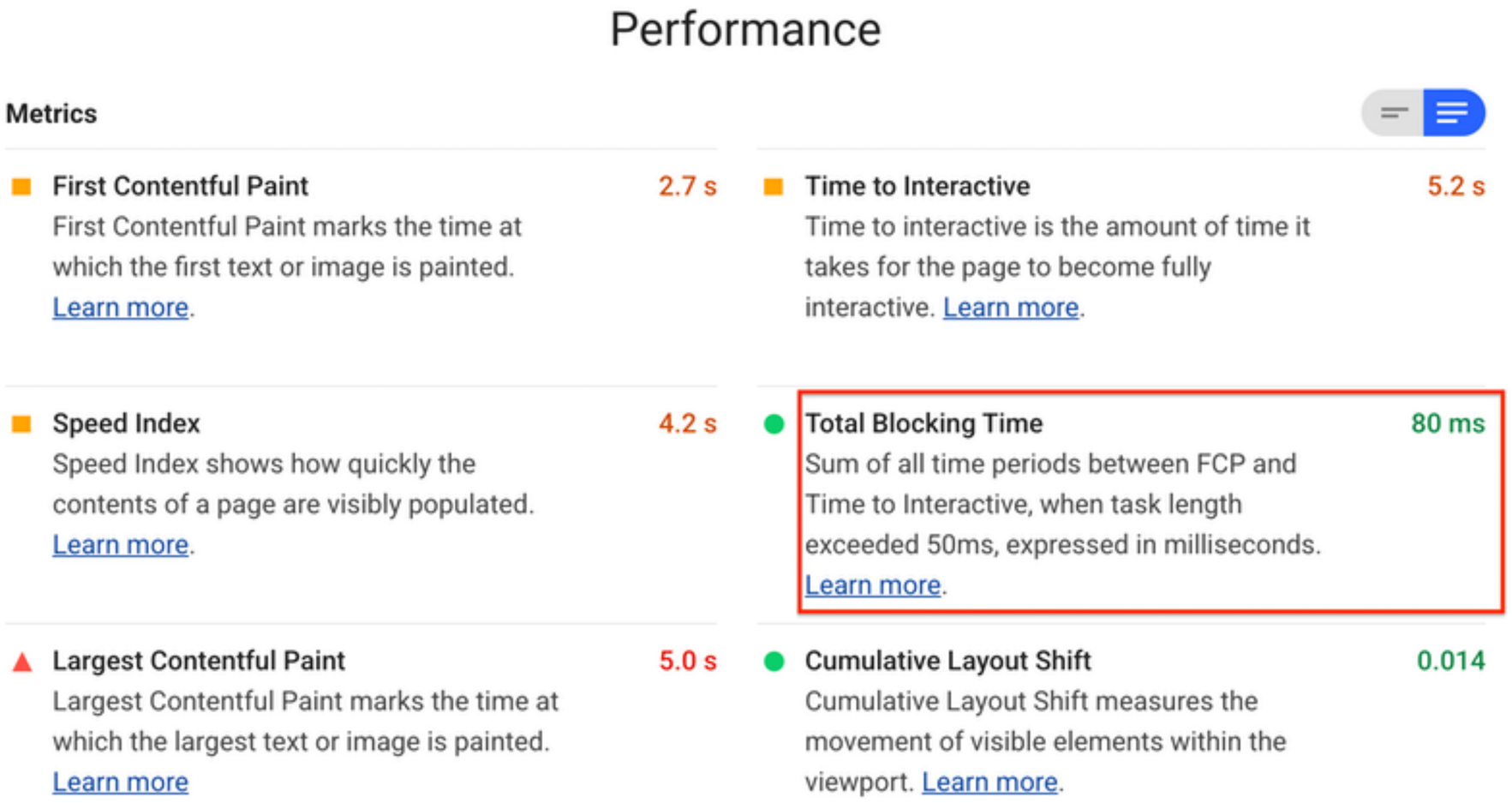
Суммарное время, на которое был заблокирован пользовательский ввод (клики мышкой, нажатия на экран, нажатия клавиш)

$$TBT = \sum_{t=FCP}^{t=TTI} (longtask_t - 50)$$

1. Берём все long tasks от момента First Contentful Paint (FCP) до момента Time to Interactive (TTI)
2. Для каждой считаем, насколько она превышает 50мс
3. Суммируем все превышения

https://web.dev/lighthouse-total-blocking-time/

TBT в Lighthouse



https://web.dev/lighthouse-total-blocking-time/

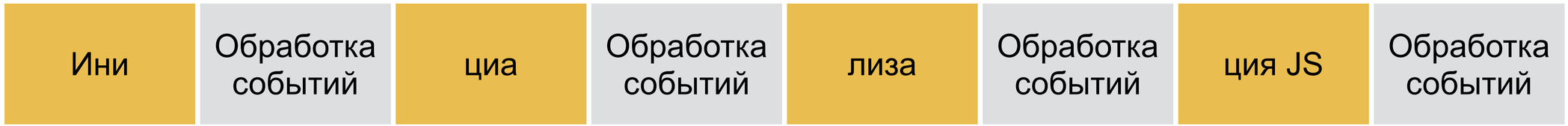
Зачем это нам знать?

При открытии страницы, написанной на **любом** фреймворке или библиотеке, надо выполнить много кода:

- React — выполнить первый рендер и монтирование компонентов
- React + SSR — выполнить гидрацию компонентов
- jQuery и т.п. legacy — проинициализировать виджеты/компоненты

Быстрая инициализация

Быстрая инициализация или быстрая реакция на ввод?



Legacy-проекты

Legacy-проект на jQuery

1. Находим DOM-элементы UI-блоков
2. Инициализируем соответствующие JS-компоненты

Инициализация

```
findDomElem(document, '.i-bem').each(function() {
    // …
    new BlockClass(…);
});
```

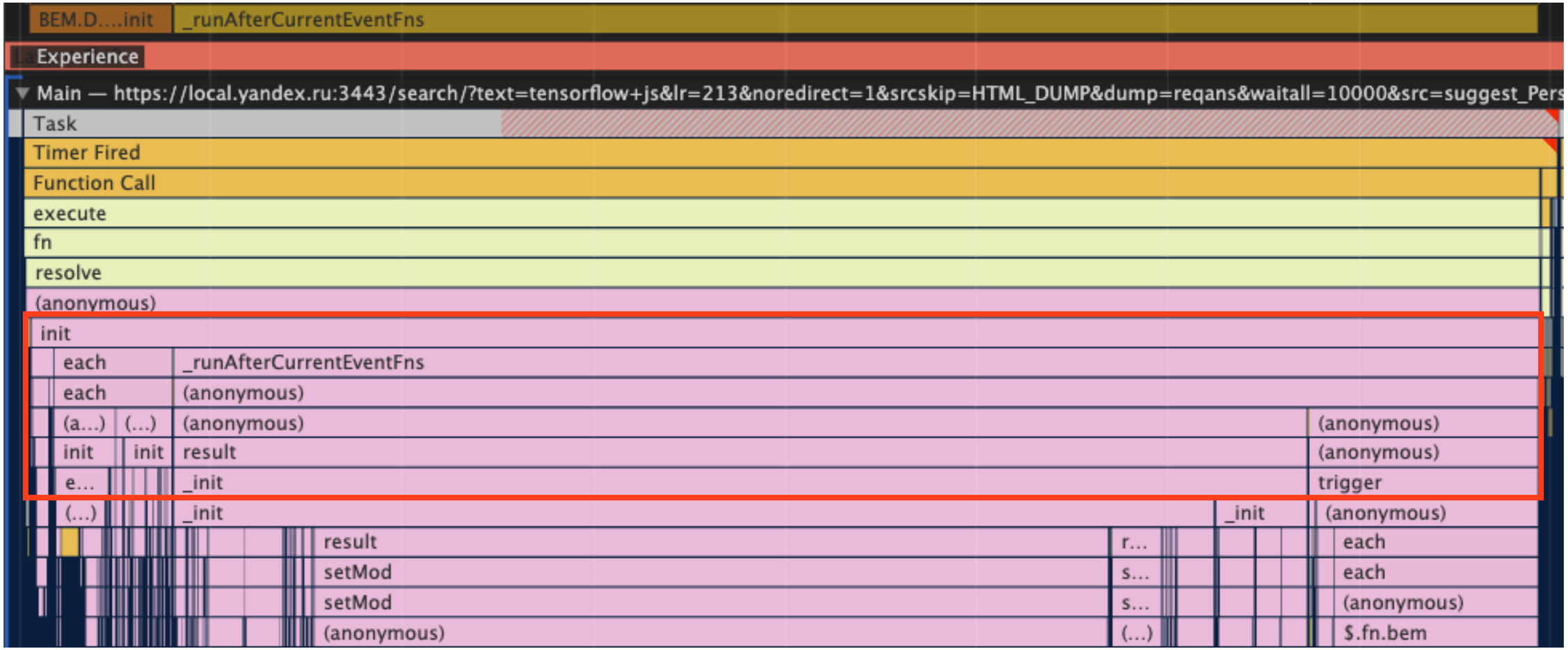
Инициализация

```
findDomElem(document, '.i-bem').each(function() {
    // …
    new BlockClass(…);
});
```

Инициализация

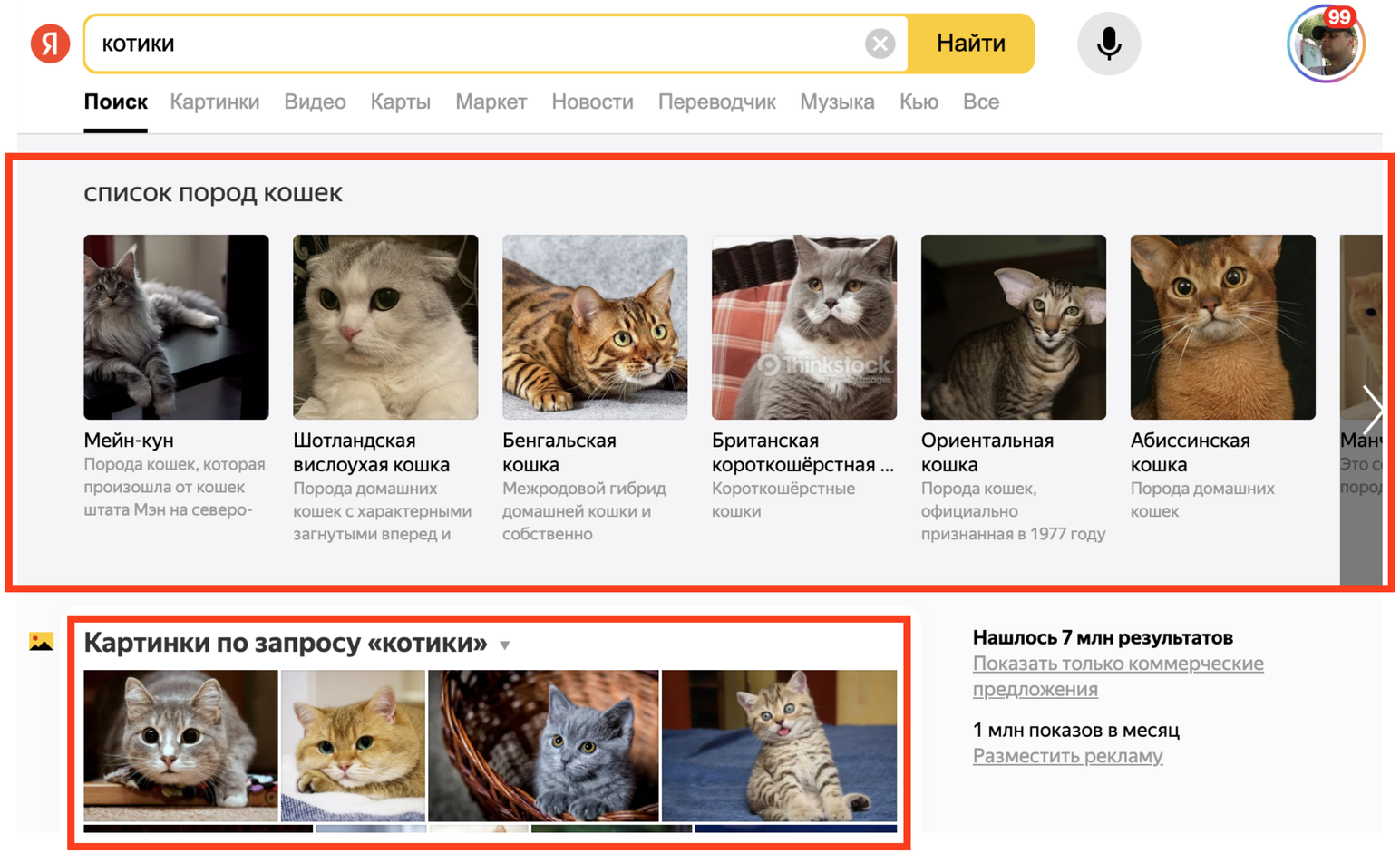
```
findDomElem(document, '.i-bem').each(function() {
    // …
    new BlockClass(…);
});
```

Инициализация — один long task



React

React в Поиске Яндекса: много Application



React в Поиске Яндекса: SSR + гидрация

- Рендерим разметку на сервере
- На клиенте находим все реактовые фичи
- Оптимизация: гидрируем только фичи на первом экране
- Фичи за первым экраном (below the fold) гидрируем позже

Гидрация компонентов React

```
const roots = document.querySelectorAll('.Root');
for (let i = 0; i < roots.length; i++) {
  // гидрируем сразу только то, что попадает во вьюпорт
  if (isWithinWindow(roots[i])) {
    hydrate(roots[i]);
  } else {
    requestAnimationFrame(hydrate.bind(null, roots[i]));
  }
}
```

Гидрация компонентов React

```
const roots = document.querySelectorAll('.Root');
for (let i = 0; i < roots.length; i++) {
  // гидрируем сразу только то, что попадает во вьюпорт
  if (isWithinWindow(roots[i])) {
    hydrate(roots[i]);
  } else {
    requestAnimationFrame(hydrate.bind(null, roots[i]));
  }
}
```

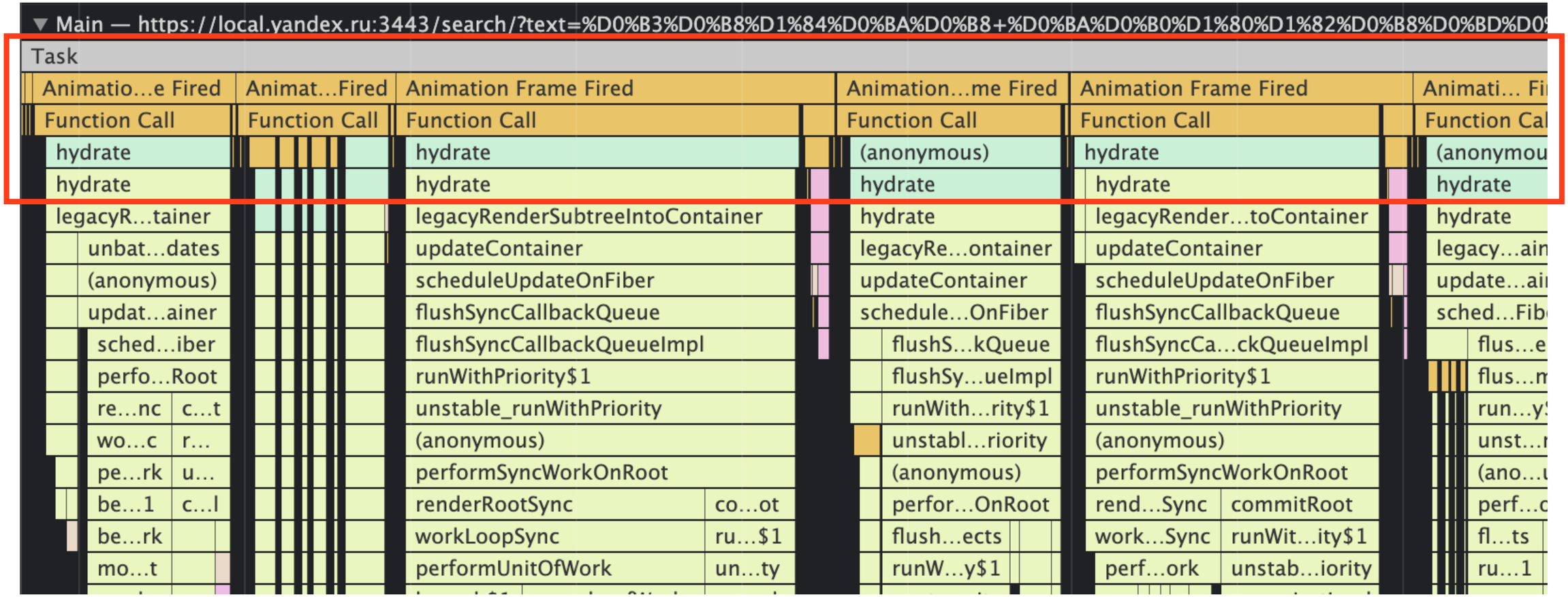
Гидрация компонентов React

```
const roots = document.querySelectorAll('.Root');
for (let i = 0; i < roots.length; i++) {
  // гидрируем сразу только то, что попадает во вьюпорт
  if (isWithinWindow(roots[i])) {
    hydrate(roots[i]);
  } else {
    requestAnimationFrame(hydrate.bind(null, roots[i]));
  }
}
```

Гидрация компонентов React

```
const roots = document.querySelectorAll('.Root');
for (let i = 0; i < roots.length; i++) {
  // гидрируем сразу только то, что попадает во вьюпорт
  if (isWithinWindow(roots[i])) {
    hydrate(roots[i]);
  } else {
    requestAnimationFrame(hydrate.bind(null, roots[i]));
  }
}
```

Все rAF склеиваются в один long task



Как убрать long task

Как убрать long task

- Выполнять за раз по одной задаче инициализации legacy-компонентов или гидрации React
- Планировать выполнение следующей задачи (setTimeout) только после выполнения текущей
- Не использовать rAF (инициализация в фоновой вкладке)
- Получилась асинхронная очередь задач

Асинхронная очередь задач

```
const executionQueue = [];
const asyncQueue = {
  push(task) {
    executionQueue.push(task);
    if (executionQueue.length === 1)
      setTimeout(this.execute.bind(this), 0);
  },
  execute() {
    try {
      const task = executionQueue.shift();
      task.fn.call(task.ctx || null);
    } catch (e) {
      // Ошибка не должна ломать исполнение всей очереди
    }
    if (executionQueue.length > 0)
      setTimeout(this.execute.bind(this), 0);
  }
};
```

```
const executionQueue = [];
const asyncQueue = {
  push(task) {
    executionQueue.push(task);
    if (executionQueue.length === 1)
      setTimeout(this.execute.bind(this), 0);
  },
  execute() {
    try {
      const task = executionQueue.shift();
      task.fn.call(task.ctx || null);
    } catch (e) {
      // Ошибка не должна ломать исполнение всей очереди
    }
    if (executionQueue.length > 0)
      setTimeout(this.execute.bind(this), 0);
  }
};
```

```
const executionQueue = [];
const asyncQueue = {
  push(task) {
    executionQueue.push(task);
    if (executionQueue.length === 1)
      setTimeout(this.execute.bind(this), 0);
  },
  execute() {
    try {
      const task = executionQueue.shift();
      task.fn.call(task.ctx || null);
    } catch (e) {
      // Ошибка не должна ломать исполнение всей очереди
    }
    if (executionQueue.length > 0)
      setTimeout(this.execute.bind(this), 0);
  }
};
```

```
const executionQueue = [];
const asyncQueue = {
  push(task) {
    executionQueue.push(task);
    if (executionQueue.length === 1)
      setTimeout(this.execute.bind(this), 0);
  },
  execute() {
    try {
      const task = executionQueue.shift();
      task.fn.call(task.ctx || null);
    } catch (e) {
      // Ошибка не должна ломать исполнение всей очереди
    }
    if (executionQueue.length > 0)
      setTimeout(this.execute.bind(this), 0);
  }
};
```

```
const executionQueue = [];
const asyncQueue = {
  push(task) {
    executionQueue.push(task);
    if (executionQueue.length === 1)
      setTimeout(this.execute.bind(this), 0);
  },
  execute() {
    try {
      const task = executionQueue.shift();
      task.fn.call(task.ctx || null);
    } catch (e) {
      // Ошибка не должна ломать исполнение всей очереди
    }
    if (executionQueue.length > 0)
      setTimeout(this.execute.bind(this), 0);
  }
};
```

A/B тест: первые результаты

Метрика	Было	Изменение
TBT	~1c	Примерно -200мс
TTI	~4c	Примерно +500мс
JS framework inited	~2c	Примерно +800мс

Ищем причину замедления

```
let t = performance.now();
for (let i = 0; i < 50; i++) {
  asyncQueue.push({
    fn: function() {
      const t2 = performance.now();
      console.log(t2 - t);
      t = t2;
    }
  });
}
```

Реальная длительность setTimeout(0), мс*

Браузер										
Chrome 93	1.3	1.2	1.4	1.5	4.5	4.3	5.4	4.7	4.3	4.3
Firefox 91	0**	1	0	0	5	4	5	6	5	6
Safari 13	2**	1	1	1	2	4	4	5	5	4

* Точные значения меняются при каждом запуске

** Firefox и Safari округляют значения до 1мс

Причина замедления

- setTimeout(0) даёт задержку в миллисекунды
- стандарт: после 5 вложенных setTimeout минимальная задержка 4мс
- задач в очереди очень много (N ~ 10…100),
  получаем порядка 4N миллисекунд ожидания

https://html.spec.whatwg.org/multipage/timers-and-user-prompts.html#timers

Второй подход к решению

Примитивный планировщик задач:

- объединяем задачи из очереди в пачку
- длительность пачки не должна превышать порога, чтобы не получился long task

Как выбрать порог

```
const MAX_BATCH_DURATION_MS = 30;
```

- 50мс — long task
- минус запас по времени на последнюю задачу в пачке

```
const asyncQueue = {
  // …
  execute() {
    var startTime = performance.now();
    while (executionQueue.length > 0) {
      var task = executionQueue.shift();
      try {
        task.fn.call(task.ctx || null);
      } catch (e) {
        // Ошибка не должна ломать исполнение всей очереди
      }
    }
    if (executionQueue.length > 0)
      setTimeout(this.execute.bind(this), 0);
  }
};
```

```
const MAX_BATCH_DURATION_MS = 30;
const asyncQueue = {
  // …
  execute() {
    var startTime = performance.now();
    while (executionQueue.length > 0 &&
        performance.now() - startTime < MAX_BATCH_DURATION_MS) {
      var task = executionQueue.shift();
      try {
        task.fn.call(task.ctx || null);
      } catch (e) {
        // Ошибка не должна ломать исполнение всей очереди
      }
    }
    if (executionQueue.length > 0)
      setTimeout(this.execute.bind(this), 0);
  }
};
```

Проблема

- Пусть в очереди две задачи
- Выполняем первую задачу, она заняла 25мс
- Пачка заняла 25мс < MAX_BATCH_DURATION_MS
- Выполняем вторую задачу, она заняла 40мс
- Пачка заняла 25 + 40мс — long task 65мс, TBT +15мс

Третий подход к решению

Более умный планировщик, предотвращающий создание long tasks

Big data + ML :)

```
const LONG_TASK_NAMES = {
  'ajax-updater': 1,  // ~60ms
  'footer': 1,        // 35…200ms
  'video-player': 1   // ~100ms
};
```

Инициализация legacy-компонентов

```
Ya.asyncQueue.push({
  fn: component.init,
  ctx: component,
});

// …

Ya.asyncQueue.execute();
```

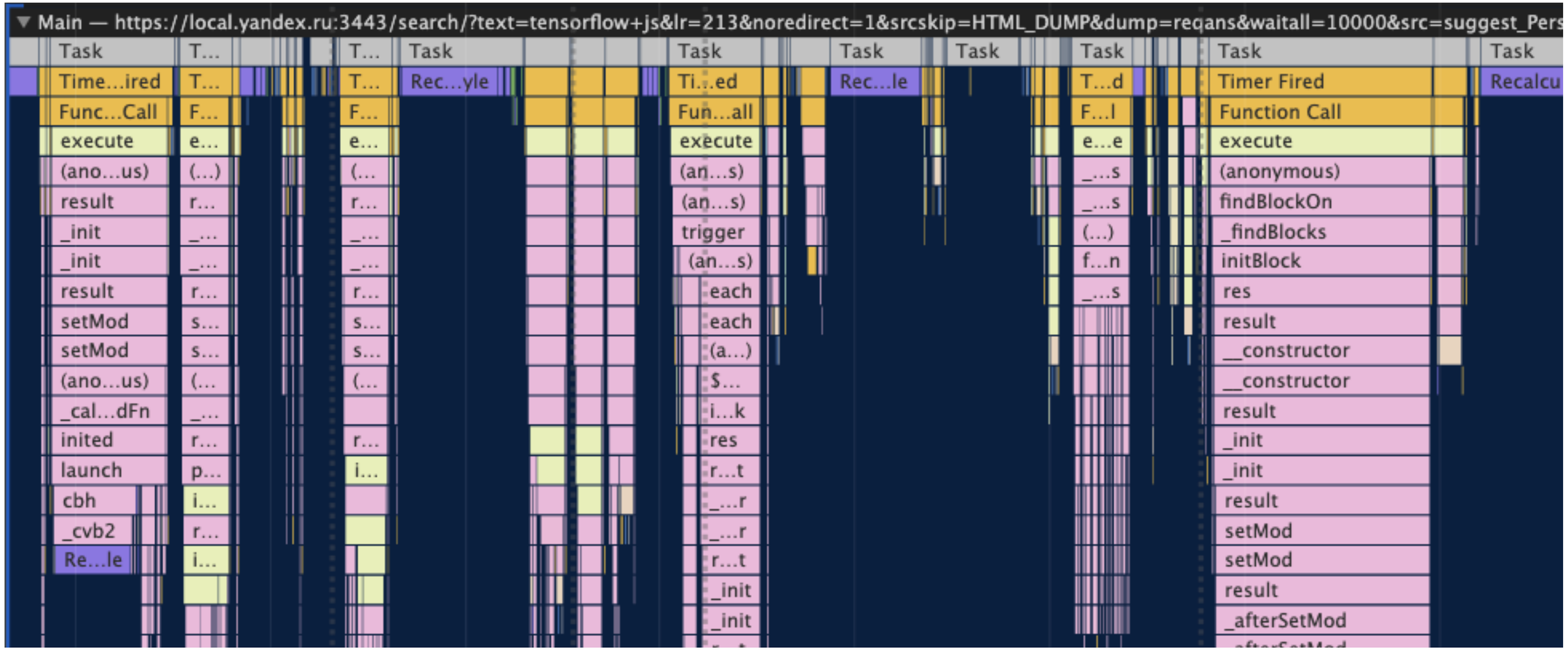
Инициализация legacy-компонентов

```
Ya.asyncQueue.push({
  fn: component.init,
  ctx: component,
  isLongTask: component.name in LONG_TASK_NAMES,
});

// …

Ya.asyncQueue.execute();
```

```
const asyncQueue = {
  // …
  execute() {
    var startTime = performance.now();
    while (executionQueue.length > 0 &&
        performance.now() - startTime < MAX_BATCH_DURATION_MS) {
      var task = executionQueue.shift();
      try {
        task.fn.call(task.ctx || null);
      } catch (e) {
        // Ошибка не должна ломать исполнение всей очереди
      }
    }
    if (executionQueue.length > 0)
      setTimeout(this.execute.bind(this), 0);
  }
};
```

```
const asyncQueue = {
  // …
  execute() {
    var startTime = performance.now();
    while (executionQueue.length > 0 &&
        performance.now() - startTime < MAX_BATCH_DURATION_MS) {
      var task = executionQueue.shift();
      try {
        task.fn.call(task.ctx || null);
      } catch (e) {
        // Ошибка не должна ломать исполнение всей очереди
      }
      // Заканчиваем пачку, если следующая задача длинная
      if (executionQueue.length > 0 && executionQueue[0].isLongTask)
        break;
    }
    if (executionQueue.length > 0)
      setTimeout(this.execute.bind(this), 0);
  }
};
```

Инициализация legacy-компонентов без long tasks



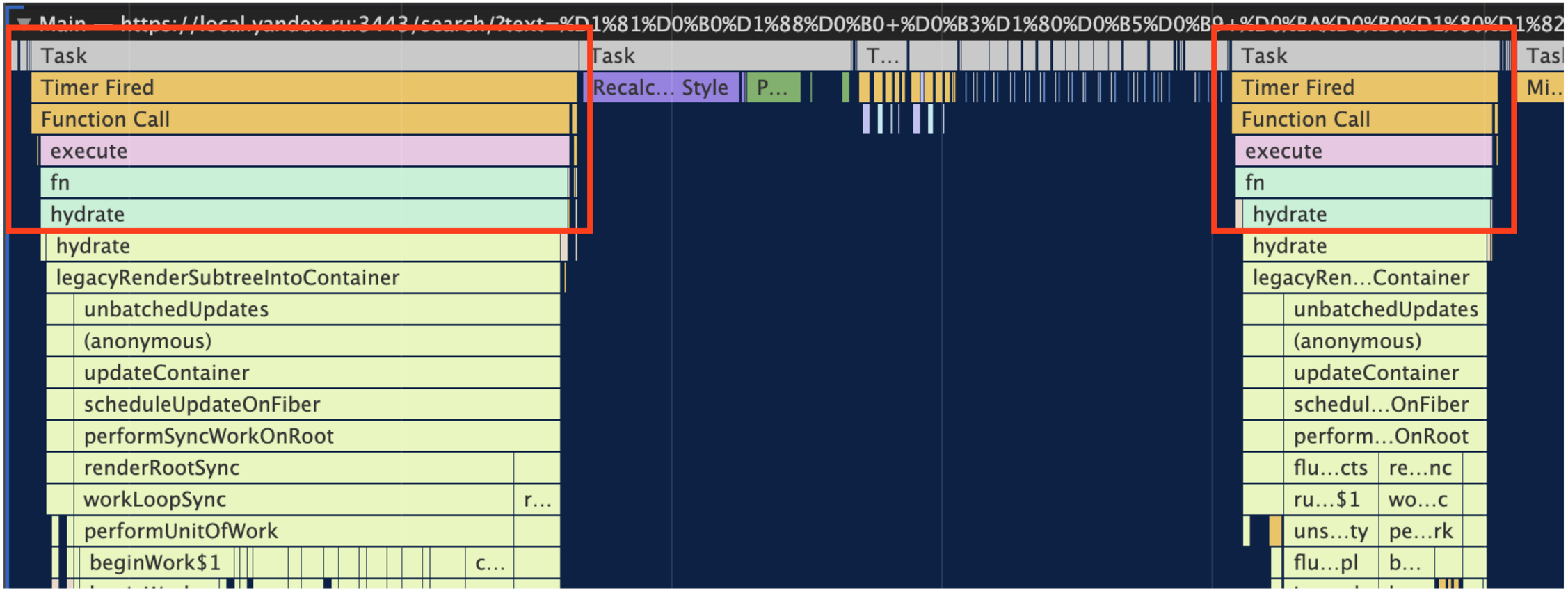
Гидрация компонентов React

```
if (isWithinWindow(roots[i])) {
  hydrate(roots[i]);
} else {
  requestAnimationFrame(
    hydrate.bind(null, roots[i])
  );
}
```

Гидрация компонентов React

```
if (isWithinWindow(roots[i])) {
  hydrate(roots[i]);
} else {
  window.Ya.asyncQueue.push({
    fn: hydrate.bind(null, roots[i])
  });
}
```

Гидрация компонентов React без long tasks



A/B тест: async legacy init + async hydrate

Метрика	Было	async init	async init + async hydrate
TBT	~1c	-120мс	-230мс
Full load (страница + ресурсы)	~3.5c	-110мс	-340мс
TTI	~4c	+75мс	+215мс
JS framework inited	~2c	+500мс	+430мс
React hydrate	~2c	±0мс	+310мс

Получился планировщик задач

- умеет объединять мелкие задачи в пачки
- умеет приоритизировать задачи:
  задачу с высоким приоритетом ставит в начало очереди,
  выполняет в ближайшей пачке

Главный недостаток

setTimeout(0) делает большой зазор в 4мс между тасками

Следствие — рост TTI и длительности инициализации фреймворка

Как убрать задержку в 4мс

postMessage

- в терминах event loop порождает отдельный таск
- не блокирует UI
- задержка меньше 4мс
- позволяет сделать аналог setTimeout(0)

https://dbaron.org/log/20100309-faster-timeouts

Задержка postMessage, мс

	1 postMessage	очередь из 100000 postMessage	средняя задержка
Chrome 93	0.2-0.3мс	2500-2600мс	~0.02мс
Firefox 91	<1мс*	580-620мс	~0.01мс
Safari 13	<1мс*	13000-14000мс	~0.13мс

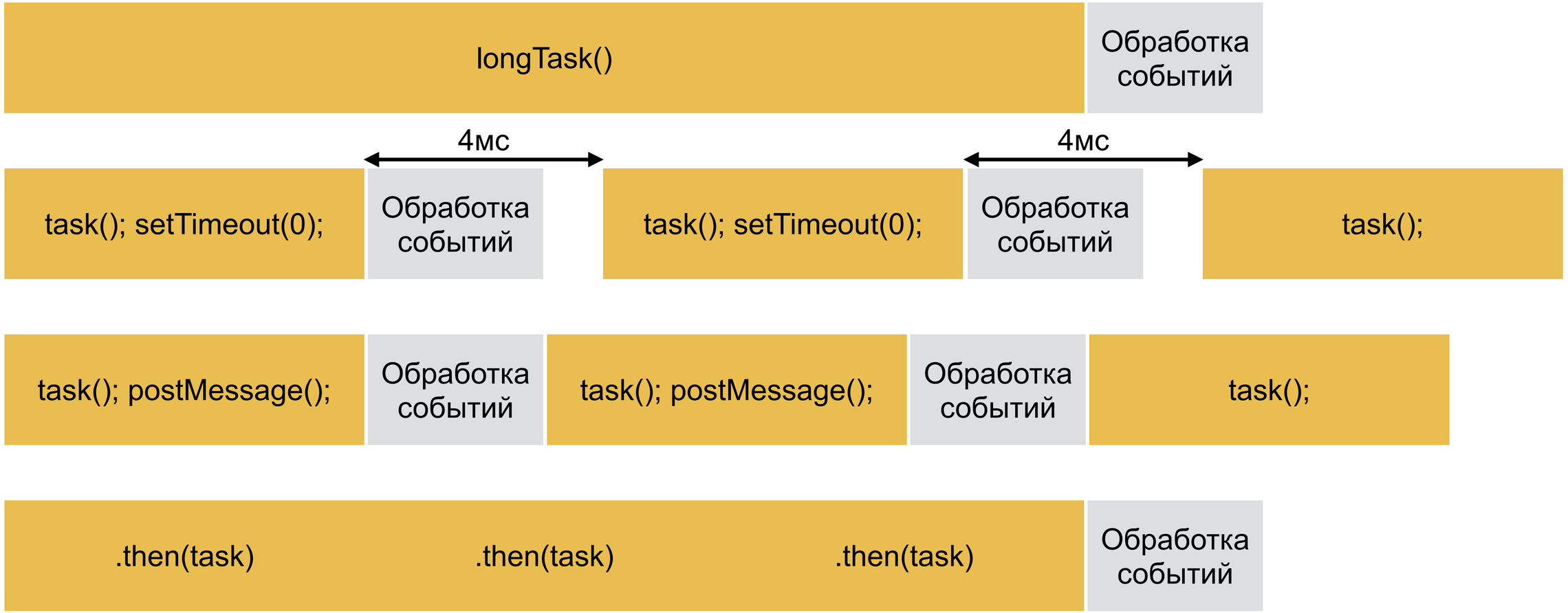
* браузер округляет измеренное время до 1мс

Почему не Promise.resolve.then?

Выполняет код в then быстрее, чем через 4мс

Но его нельзя использовать

Почему не Promise.resolve.then?



https://developer.mozilla.org/en-US/docs/Web/API/HTML_DOM_API/Microtask_guide

window.postMessage

```
const scheduledTasks = [];
const messageName = 'zero-timeout-message';

window.addEventListener('message', function(e) {
  if (e.source === window && e.data === messageName) {
    e.stopPropagation();
    if (scheduledTasks.length > 0)
      scheduledTasks.shift()();
  }
}, true);

function schedule(fn) {
  scheduledTasks.push(fn);
  window.postMessage(messageName, '*');
}
```

window.postMessage

```
const executionQueue = [];

const asyncQueue = {
  push(task) {
    // …
    if (executionQueue.length === 1)
      setTimeout(this.execute.bind(this), 0);
  },

  execute() {
    // …
    if (executionQueue.length > 0)
      setTimeout(this.execute.bind(this), 0);
  }
};
```

window.postMessage

```
const executionQueue = [];

const asyncQueue = {
  push(task) {
    // …
    if (executionQueue.length === 1)
      schedule(this.execute.bind(this));
  },

  execute() {
    // …
    if (executionQueue.length > 0)
      schedule(this.execute.bind(this));
  }
};
```

A/B тест: setTimeout vs postMessage

Метрика	setTimeout(0)	window.postMessage
TBT	-230мс	-215мс
Full load (страница + ресурсы)	-340мс	-300мс
TTI	+215мс	+55мс
JS framework inited	+430мс	+10мс
React hydrate	+310мс	+260мс

Что можно улучшить

- На window есть чужие слушатели события message
- На window есть чужие postMessage
- Всё ещё есть эвристическое объединение задач в пачки

v2: postMessage в MessageChannel

```
const executionQueue = [];
const channel = new MessageChannel();

channel.port1.onmessage = function() {
  if (executionQueue.length > 0)
    executionQueue.shift()();
};

function schedule() {
  channel.port2.postMessage(undefined);
}
```

v2: всё в отдельных макротасках

```
const asyncQueue = {
  push(task) {
    if (task.isHighPriority) {
      executionQueue.unshift(task);
    } else {
      executionQueue.push(task);
    }
    schedule();
  },

  execute() {
    // Пустая функция
  }
};
```

A/B тест: setTimeout vs postMessage

Метрика	Было	setTimeout(0)	window. postMessage	MessageChannel postMessage
TBT	~1c	-230мс	-215мс	-215мс
Full load (страница + ресурсы)	~3.5c	-340мс	-300мс	-370мс
TTI	~4c	+215мс	+55мс	+25мс
JS framework inited	~2c	+430мс	+10мс	-150мс
React hydrate	~2c	+310мс	+260мс	+310мс

А что у других?

SSR + ленивая гидрация

Vue + lazy hydration

```
<LazyHydrate never>
  <ArticleContent :content="article.content" />
</LazyHydrate>

<LazyHydrate when-visible>
  <AdSlider />
</LazyHydrate>

<LazyHydrate when-idle>
  <ImageSlider />
</LazyHydrate>
```

https://github.com/maoberlehner/vue-lazy-hydration

React + lazy hydration

```
<LazyHydrate ssrOnly>
  {...}
</LazyHydrate>

<LazyHydrate whenVisible>
  {...}
</LazyHydrate>

<LazyHydrate whenIdle>
  {...}
</LazyHydrate>
```

https://github.com/hadeeb/react-lazy-hydration

Qwik + React + partial hydration

```
npm create qwik@latest
cd qwik-app
npm run qwik add react
```

https://youtu.be/fLH28rDu97I https://qwik.builder.io/qwikcity/integrations/react/

Qwik + React + partial hydration

```
/** @jsxImportSource react */

import {qwikify$} from '@builder.io/qwik-react';
import {Button, Slider} from '@mui/material';

export const MUIButton = qwikify$(Button);

// eagerness: 'load' | 'visible' | 'idle' | 'hover';
export const MUISlider = qwikify$(Slider, {eagerness: 'hover'});
```

https://youtu.be/fLH28rDu97I https://qwik.builder.io/qwikcity/integrations/react/

React 18+

- Automatic batching
  (группировка изменений состояния в один перерендер)
- Streaming SSR
- Selective/Progressive Hydration

https://youtu.be/pj5N-Khihgc

Планировщик в React

Планировщик в React

- Версия 0.23.0, API может измениться в любой момент
- setImmediate, postMessage, setTimeout, форк с postTask

https://www.npmjs.com/package/scheduler

Планировщик в React

```
if (typeof MessageChannel !== 'undefined') {
  const channel = new MessageChannel();
  const port = channel.port2;
  channel.port1.onmessage = performWork;
  schedule = () => {
    port.postMessage(null);
  };
} else {
  schedule = () => {
    setTimeout(performWork, 0);
  };
}
```

react/packages/scheduler/src/forks/Scheduler.js

Планировщик в React

```
const ImmediatePriority = 1;
const UserBlockingPriority = 2;
const NormalPriority = 3;
const LowPriority = 4;
const IdlePriority = 5;
```

Планировщик в React

```
var IMMEDIATE_PRIORITY_TIMEOUT = -1;
var USER_BLOCKING_PRIORITY_TIMEOUT = 250;
var NORMAL_PRIORITY_TIMEOUT = 5000;
var LOW_PRIORITY_TIMEOUT = 10000;
var IDLE_PRIORITY_TIMEOUT = 1073741823;
// …
var expirationTime = currentTime + timeout;
// …
newTask.sortIndex = expirationTime;
```

Планировщик в Google Maps

Планировщик в Google Maps

Closed source

Краткое описание: https://github.com/WICG/scheduling-apis/blob/main/misc/userspace-schedulers.md#case-study-1-maps-job-scheduler

Планировщик в Google Maps

Главная цель — ровный FPS и плавный UI без дёргания

Планировщик в Google Maps

Пытается выравнять FPS по сетке:

- screen refresh rate (60, 90, 144Hz)
- 1/2 (30, 45, 72Hz)
- 1/3 (20, 30, 48Hz) и т.п.

Планировщик в Google Maps

Тяжёлые задачи, на которых нельзя поддержать постоянный FPS

- старт Google Maps
- переход в 3D-режим

Планировщик на них отключается

Планировщик в Google Maps

Приоритеты задач:

1. Input: обновление состояния на user input
2. Animation: обновление состояния по времени
3. Rendering: отрисовка состояния на экране
4. Other: всё остальное

Input, Animation, Rendering должны выполняться на каждый кадр, планируются через rAF

Other планируется через rIC или rAF + postMessage или setTimeout(0)

Планировщик LRT

Планировщик LRT

Автор — Дмитрий Филатов, Яндекс

Работает в Node.js и в браузерах. Использует на выбор

- requestIdleCallback
- requestAnimationFrame
- window.postMessage
- setImmediate
- setTimeout(0)
- ваш кастомный код

https://github.com/dfilatov/lrt https://www.npmjs.com/package/lrt

Планировщик LRT

```
function* task1Generator() {
  let i = 0;
  while(i < 10) { // 10 units will be executed
    doPartOfTask1();
    i++;
    yield;
  }
  return i;
}

function* task2Generator() { /* ... */ }

const scheduler = createScheduler();
// Run both tasks concurrently
const task1 = scheduler.runTask(task1Generator());
const task2 = scheduler.runTask(task2Generator());
```

https://github.com/dfilatov/lrt https://www.npmjs.com/package/lrt

Браузерные API

Браузерные API

- navigator.scheduling.isInputPending() — Chrome 87+

```
while (true && !navigator.scheduling.isInputPending())
    doTasks();
```

- scheduler.postTask() — Chrome 94+, Firefox 108+

```
scheduler.postTask(fn1, {priority: 'background'});
scheduler.postTask(fn2, {priority: 'user-visible'});
scheduler.postTask(fn3, {priority: 'user-blocking'});
```

Спасибо! Вопросы?

https://clck.ru/32a8dT