+40%

Как я ускорил Glimmer

Импортозамещаем Svelte

Aleksandr Kanunnikov

Люблю Open Source

16 лет пишу **код**

Работал с: ExtJS, Backbone.js, jQuery, Angular.js, React, Vue, Svelte, Angular, Ember

Aleksandr Kanunnikov

Работал в доменах

Telecom

Smart TV

Advertising

Agrotech

Life Science

Fintech

E-commerce

Aleksandr Kanunnikov

Решал задачи

миграции с легаси

тестирования

реплатформинга

и прочее

запуск новых проектов

ленивой загрузки

headless архитектуры

Performance matters

Интересуюсь производительностью веб-приложений больше **10 лет**.

Первый performance-related PR в **GlimmerVM** в **2019** году

7x

https://github.com/krausest/js-framework-benchmark/pull/592

Нет предела совершенству

А можно ли улучшить архитектуру реактивности GlimmerVM?

Переосмысляя реактивность

Базовая модель реактивности Glimmer

Переосмысляя реактивность

Базовая модель реактивности Glimmer основана на концепции

Часы Лампорта

Переосмысляя реактивность

Часы Лампорта / Lamport Timestamp

- алгоритм определения порядка событий в распределённой системе, разработанный Лэсли Лампортом в 1978 году

Переосмысляя реактивность

- pzuraq.com/blog/how-autotracking-works
- v5.chriskrycho.com/journal/autotracking-elegant-dx-via-cutting-edge-cs

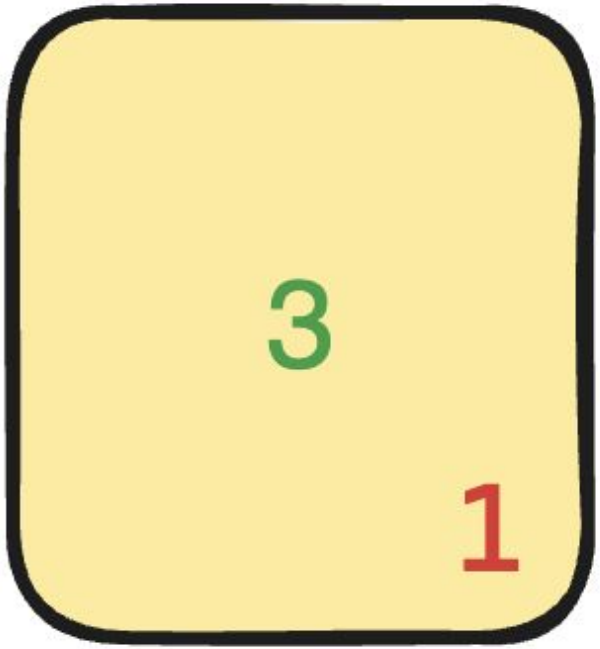
Словарик

Cell - ячейка с данными

Formula - вычисляемое значение

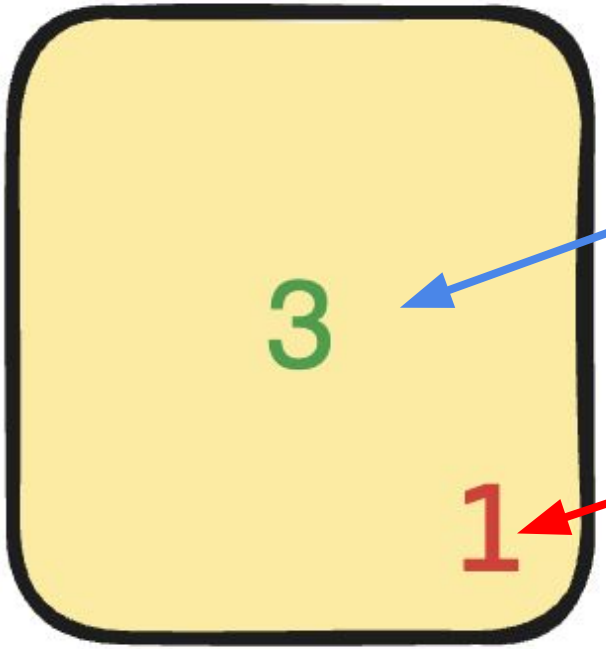
Opcode - функция мутации DOM

Это займёт не больше 5 минут



Cell

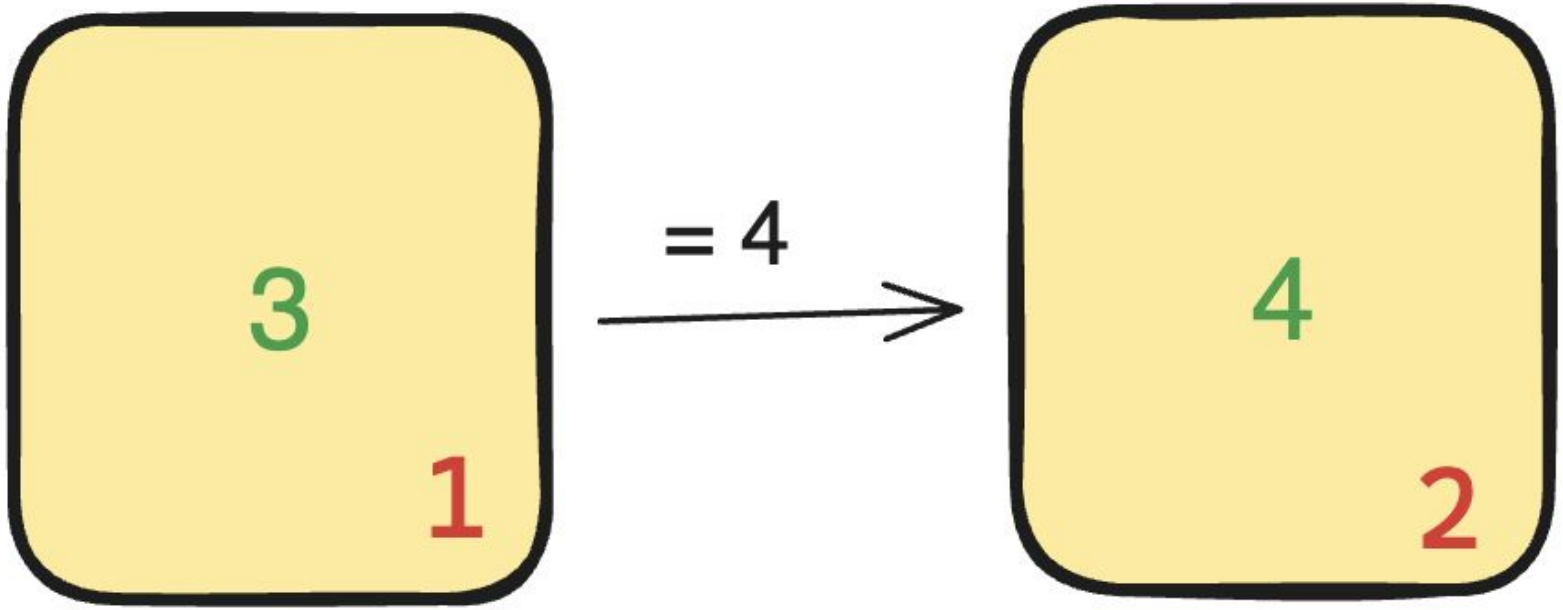
Это займёт не больше 5 минут



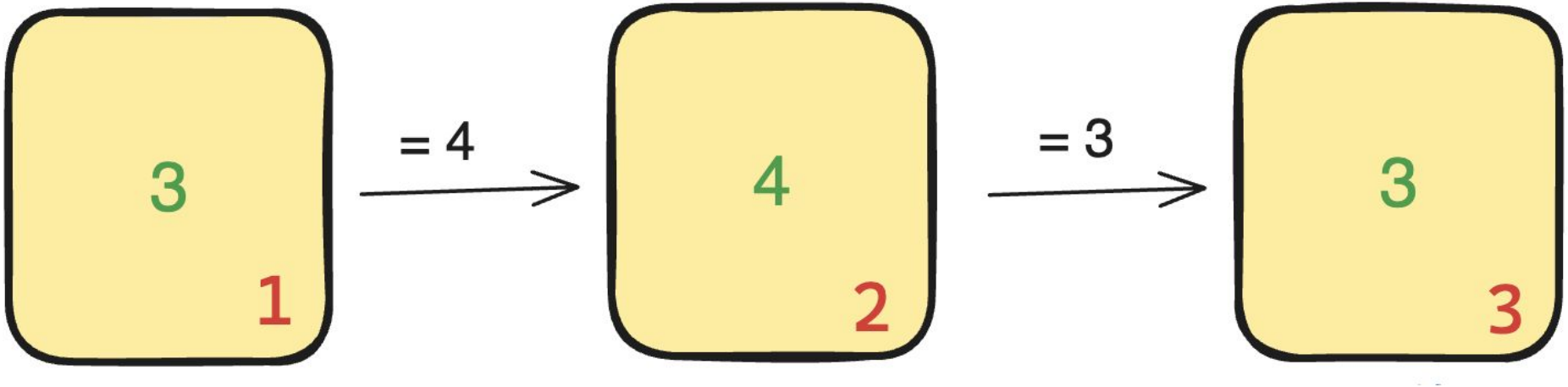
Значение

Инкрементальная версия мутации в системе

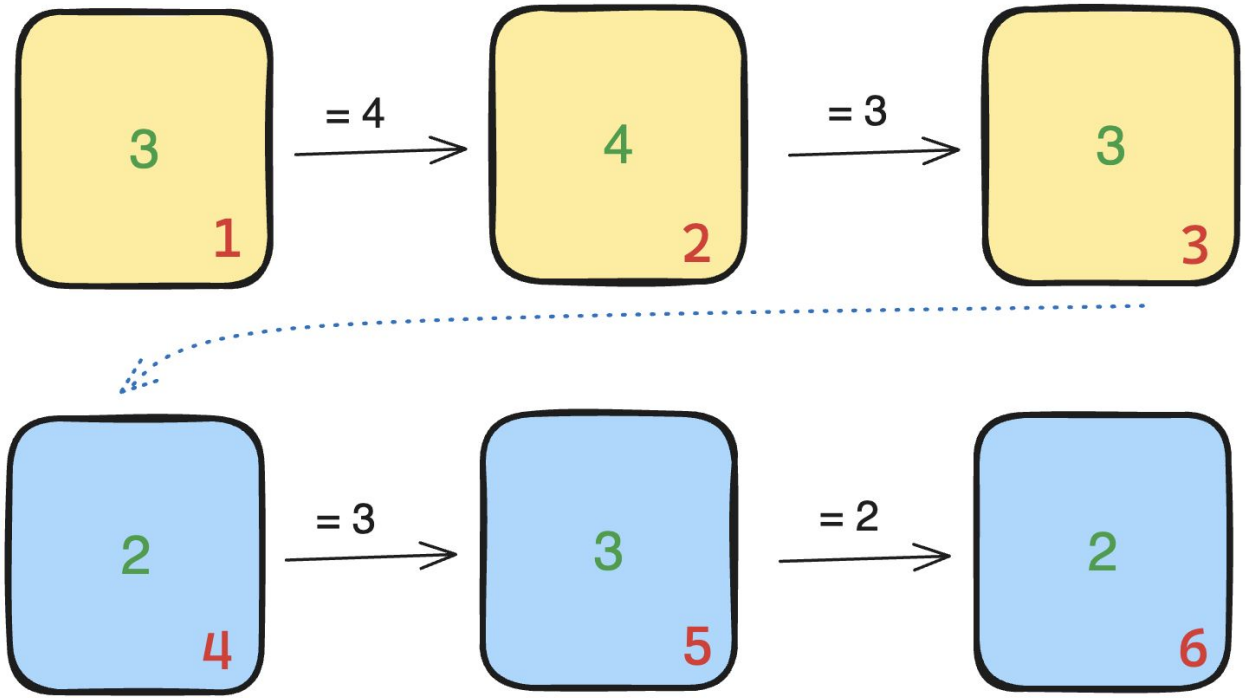
Это займёт не больше 5 минут



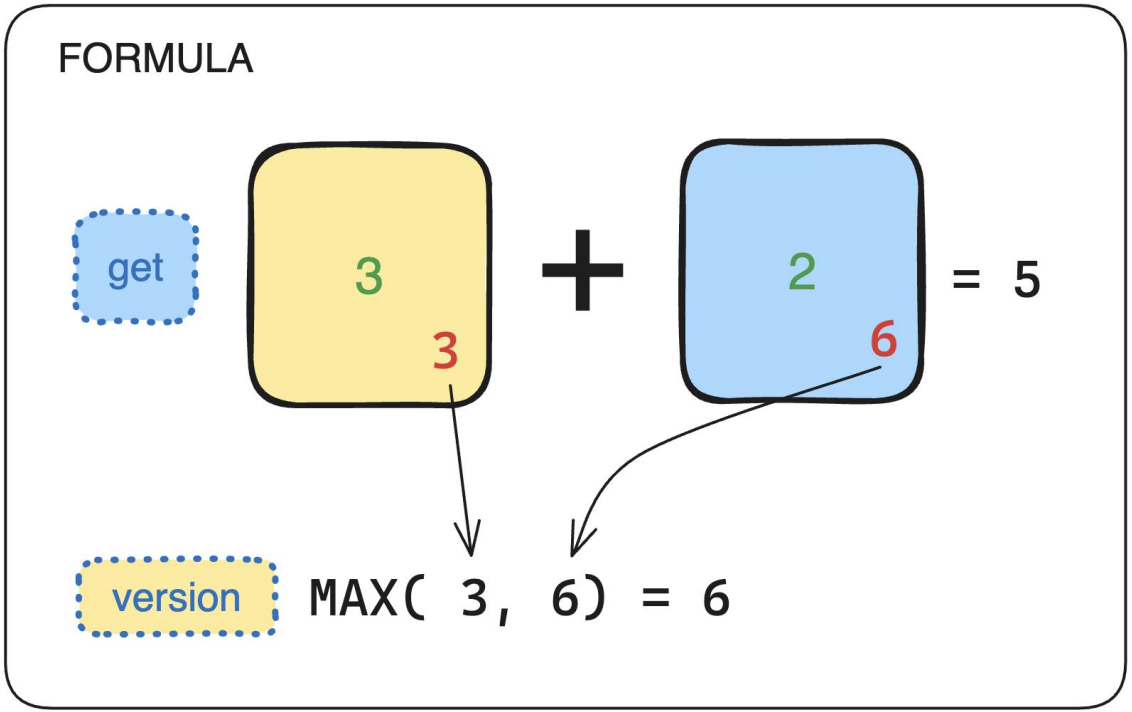
Это займёт не больше 5 минут

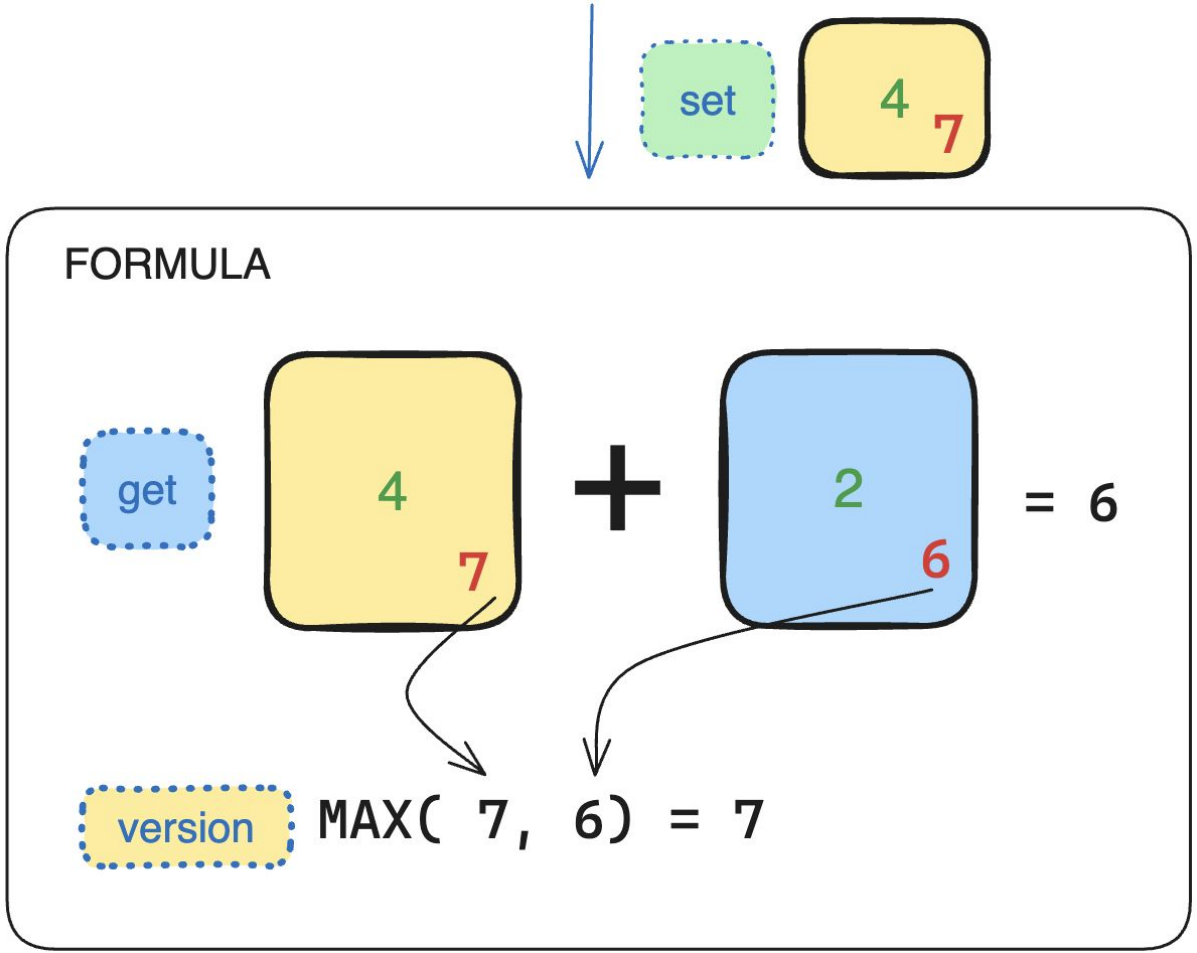


Это займёт не больше 5 минут



Это займёт не больше 5 минут




set
4
7
FORMULA
get
4
7
+
2
6
= 6
version
MAX(7, 6) = 7

```
let OLD_VERSION = 6
let CURRENT_VERSION = FORMULA.VERSION()

const IS_OUTDATED = CURRENT_VERSION > OLD_VERSION

if (IS_OUTDATED) {
  OLD_VERSION = CURRENT_VERSION
  DIV.textContent = FORMULA.get()
}
```

Преимущества

- ленивые вычисления значений

Преимущества

- ленивые вычисления значений
- кэширование вычислений

Преимущества

- ленивые вычисления значений
- кэширование вычислений
- знаем, что именно изменилось

Преимущества

- ленивые вычисления значений
- кэширование вычислений
- знаем, что именно изменилось
- быстрая валидация версии

Недостатки

- жрём память на кэшах

Недостатки

- жрём память на кэшах
- необходимость валидации всех формул

часть 14 / 834

Не так уж и страшно

часть 15 / 834

Проблема

10 000 не связанных между собой состояний

таблица
100x100

часть 16 / 834

Проблема

10 000 не связанных между собой состояний

При изменении 1 значения происходит 10 000 валидаций

часть 17 / 834

Проблема

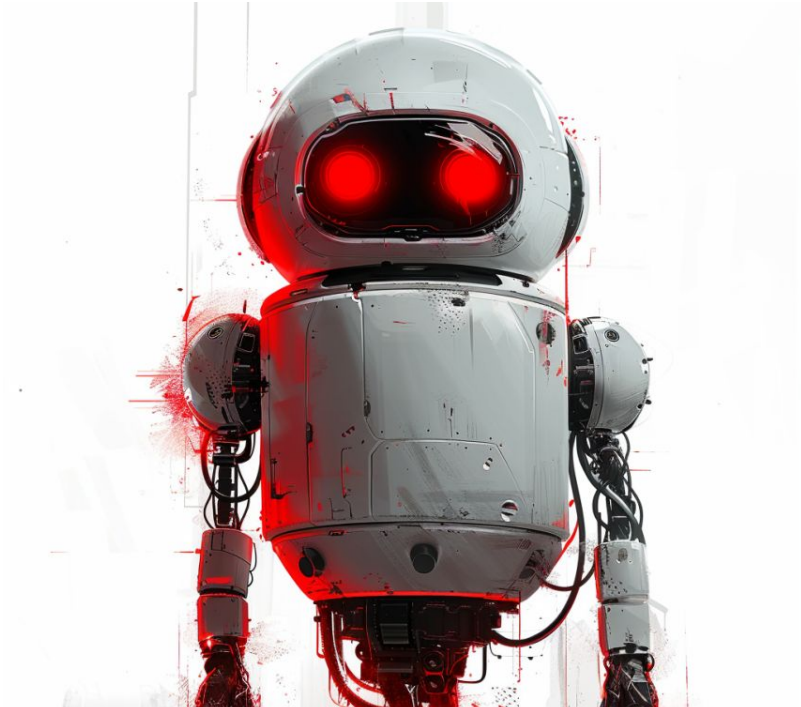
10 000 не связанных между собой состояний

При изменении 1 значения происходит 10 000 валидаций

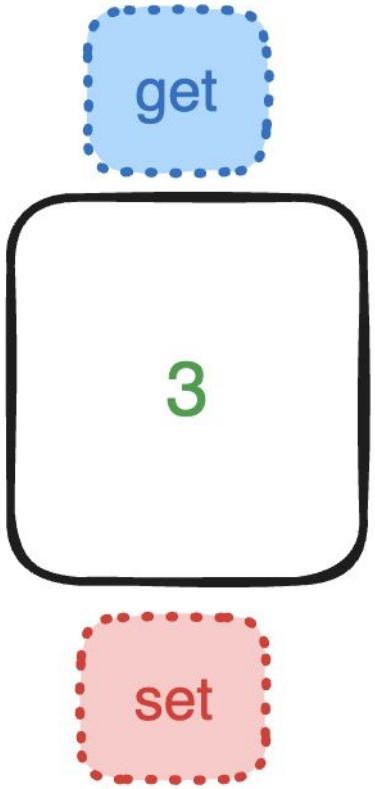
- **влияет** на **время отклика** системы

часть 18 / 834

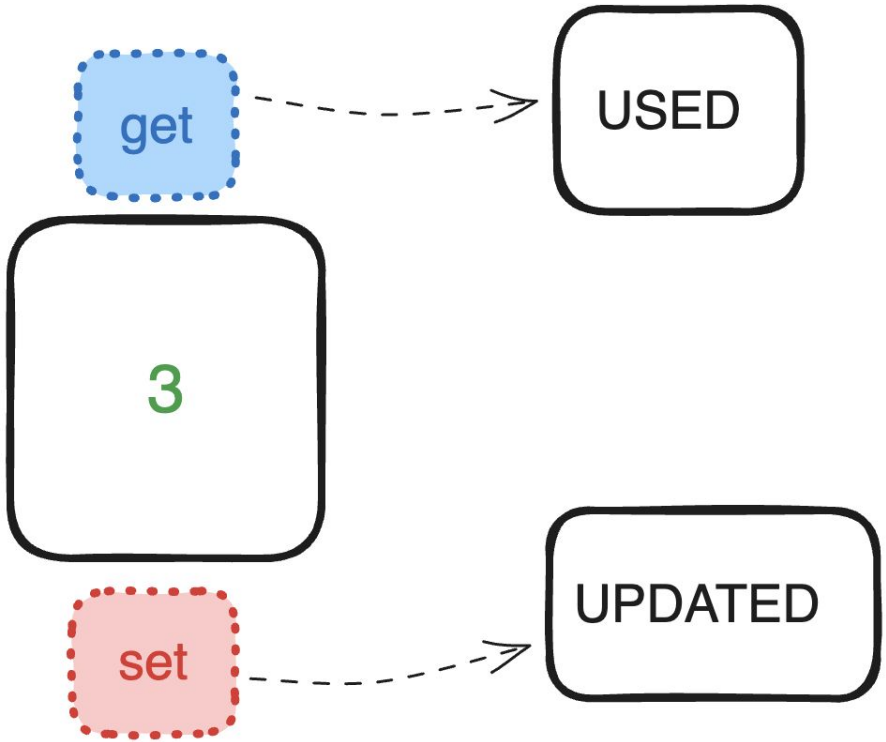
Давайте попробуем найти решение



Давайте попробуем найти решение



Давайте попробуем найти решение



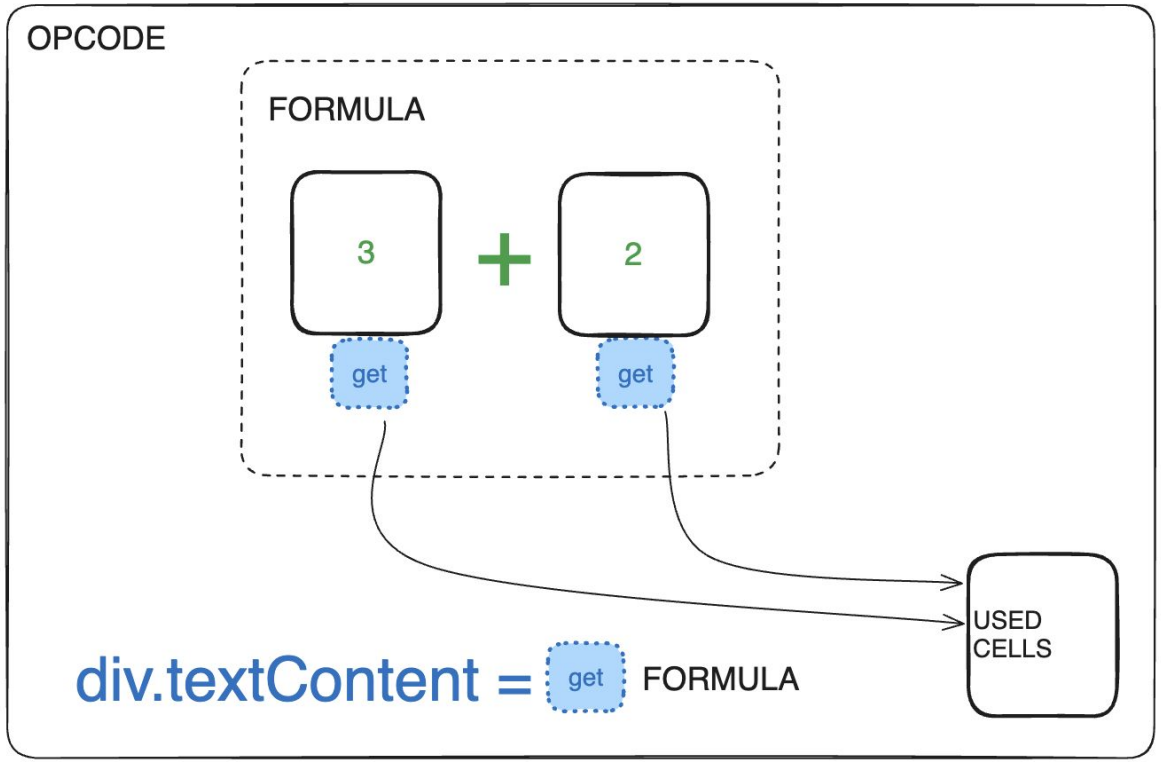
Давайте попробуем найти решение

Cell - ячейка с данными

Formula - вычисляемое значение

Opcode - функция мутации DOM

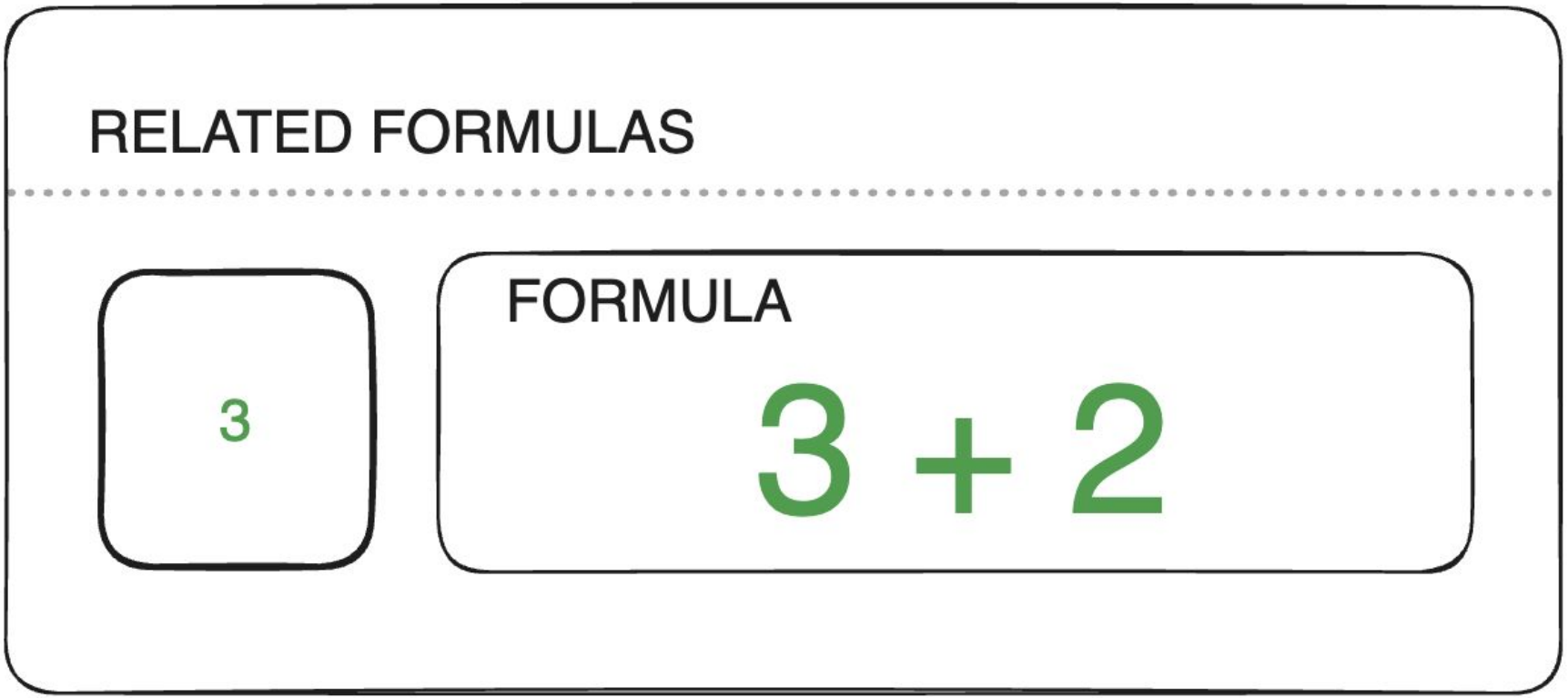
Давайте попробуем найти решение



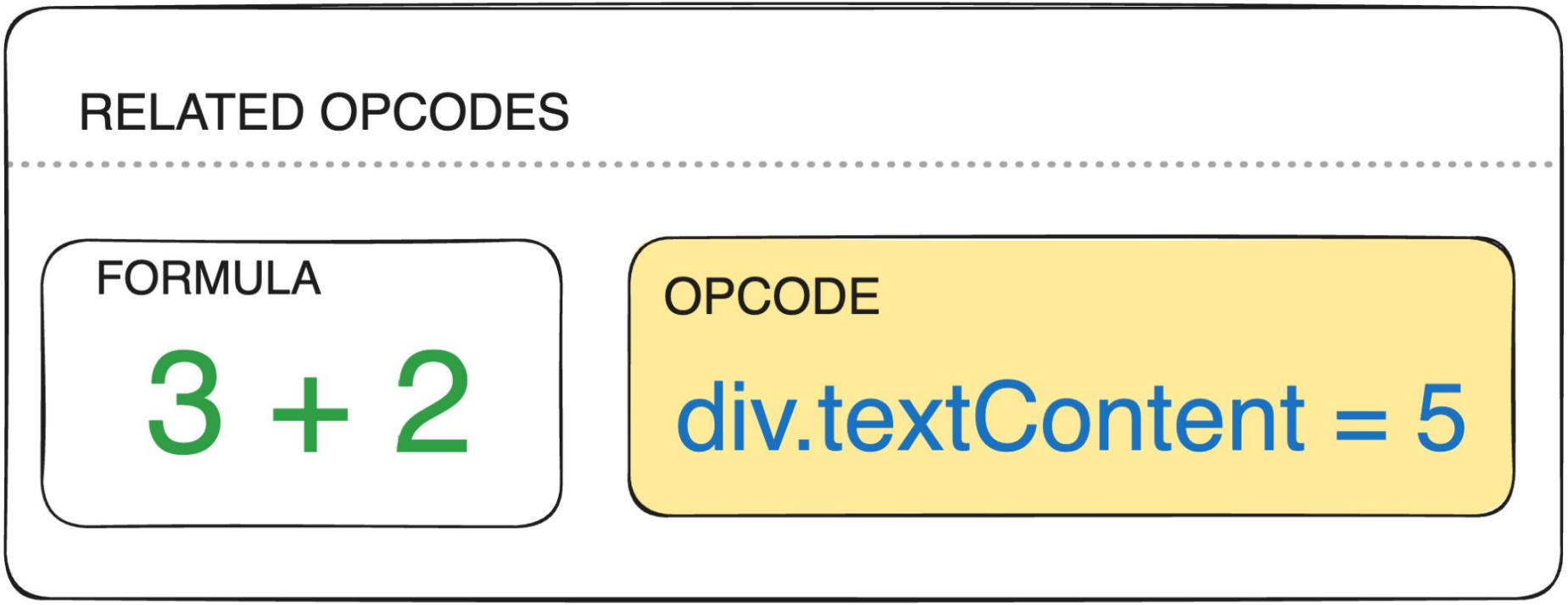
Давайте попробуем найти решение

I like Weak**Map**

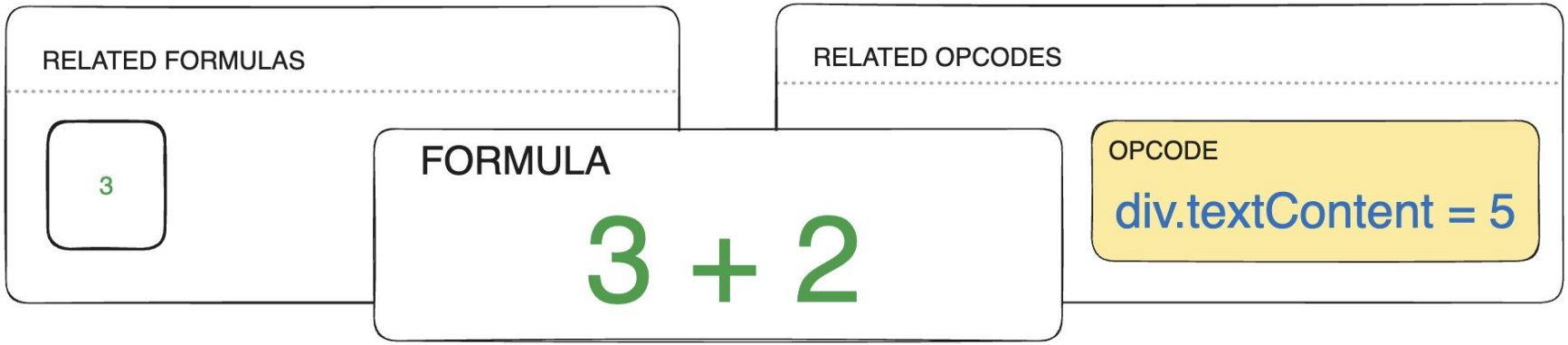
WeakMap < CELL, FORMULA >



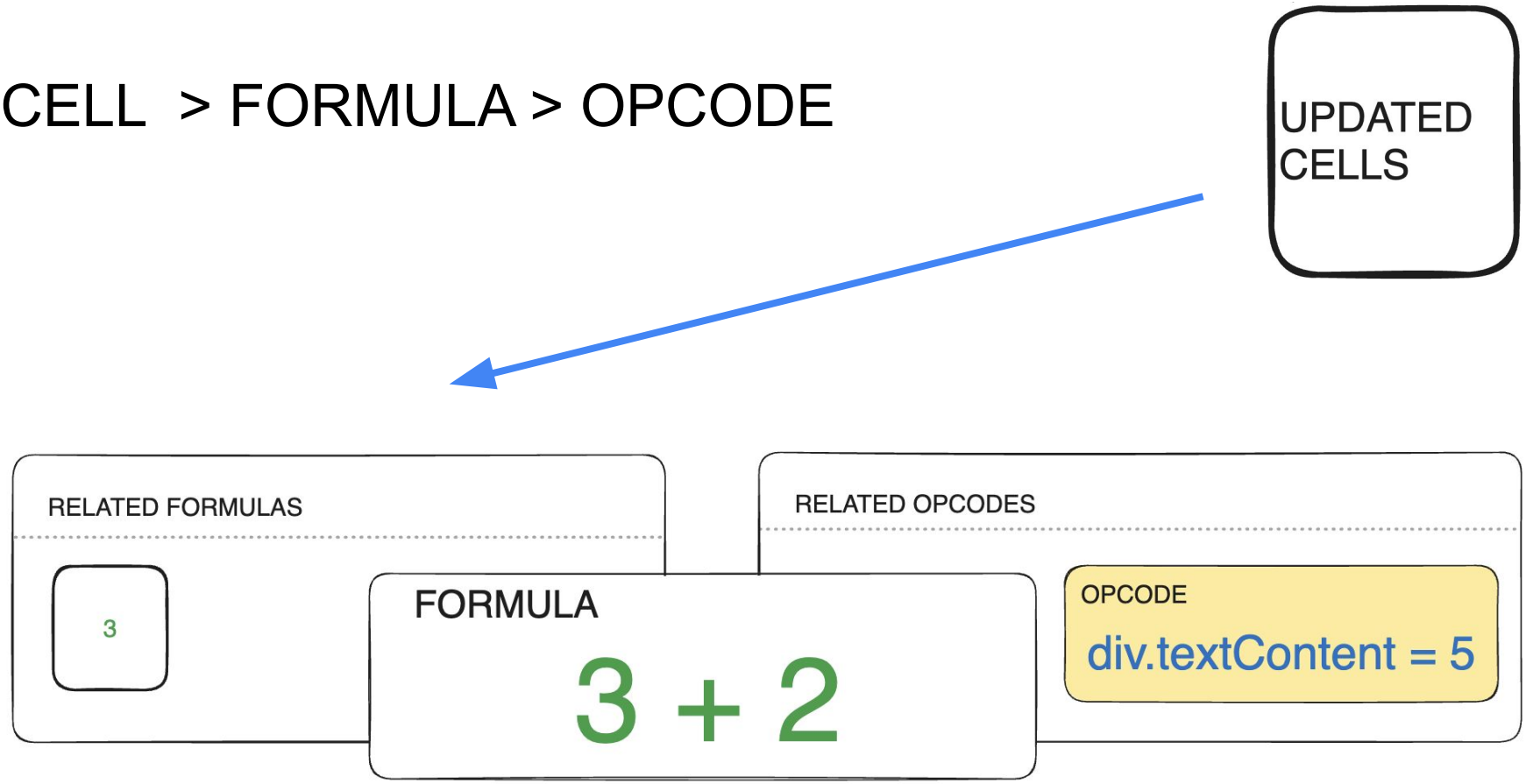
WeakMap < FORMULA, OPCODE >



CELL > FORMULA > OPCODE



CELL > FORMULA > OPCODE



Вот мы и придумали новую систему реактивности

До

1. update value
2. validate all existing tags in application
3. iterate over all opcodes and if changed
4. execute

После

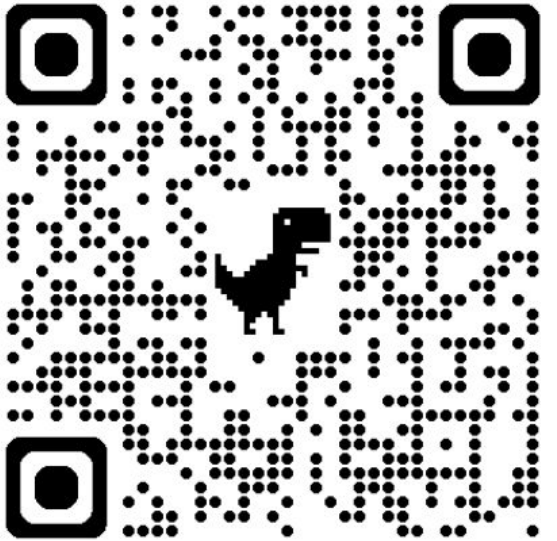
1. update value
2. get related opcodes
3. execute

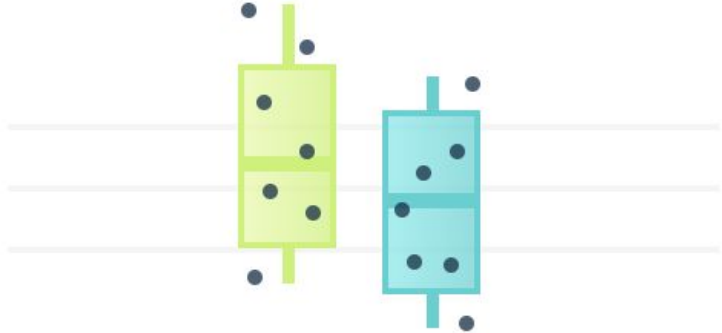
Тестируем

VanillaJS-"keyed"

Create 1,000 rows | Create 10,000 rows
Append 1,000 rows | Update every 10th row
Clear | Swap Rows

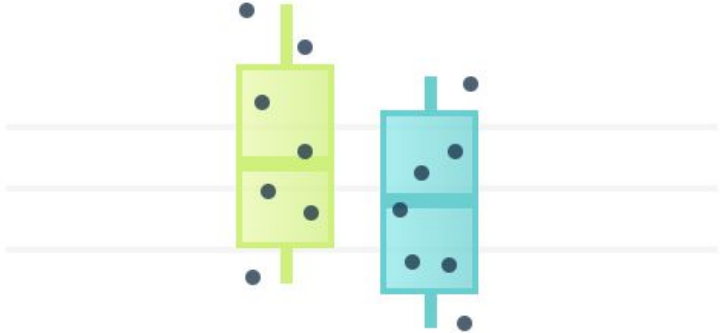
1	inexpensive white chair	✖
2	helpful blue cookie	✖
3	fancy black pony	✖
4	plain blue desk	✖
5	cheap blue desk	✖
6	plain red keyboard	✖
7	mushy white house	✖





TRACERBENCH

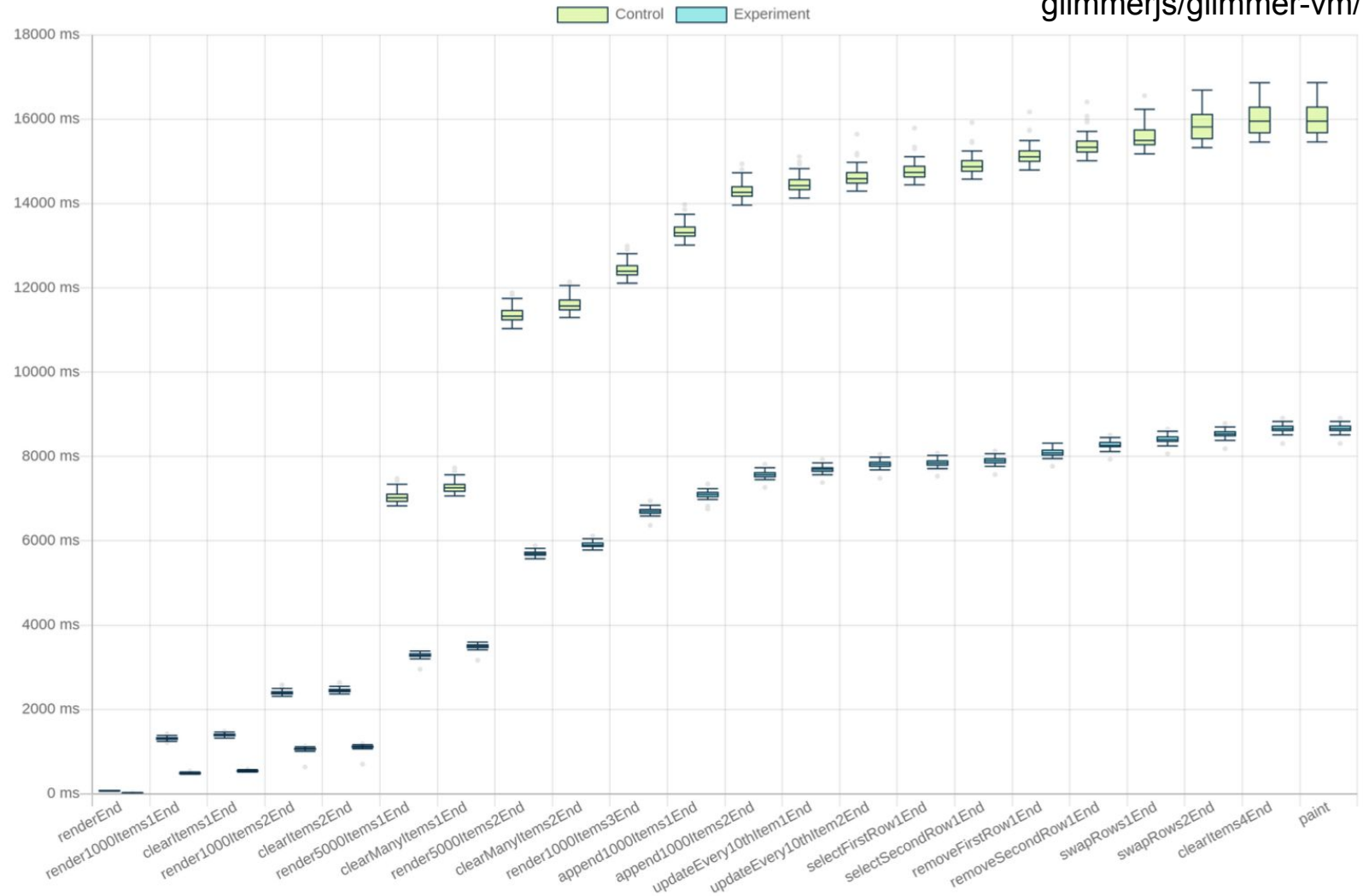
Controlled Performance Benchmarking Tool



TRACERBENCH

100 samples

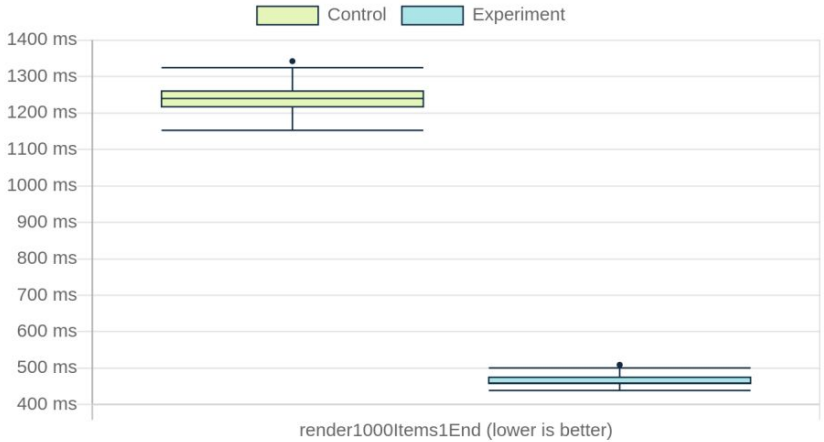
glimmerjs/glimmer-vm/pull/1541

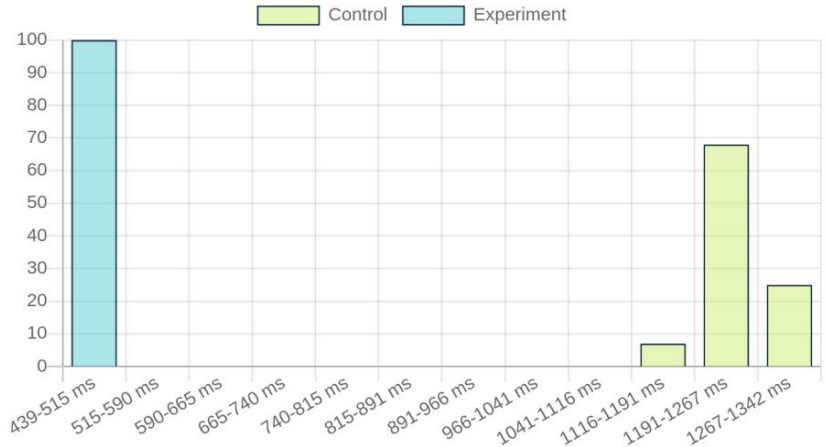


Тестируем

render1000Items1End **(776 ms faster)**

Based on the P-value of this benchmark the evidence for a metric shift is **very strong.** TracerBench has determined the results are **significant** meaning they are worth looking at. A statistics estimator *(Hodges–Lehmann estimator)* was used to determine "Experiment" is **faster** by **776 ms**. TracerBench is 95% confident "Experiment" is **faster** between **765 ms to 780 ms** based on 100 samples using a *(confidence interval)*.

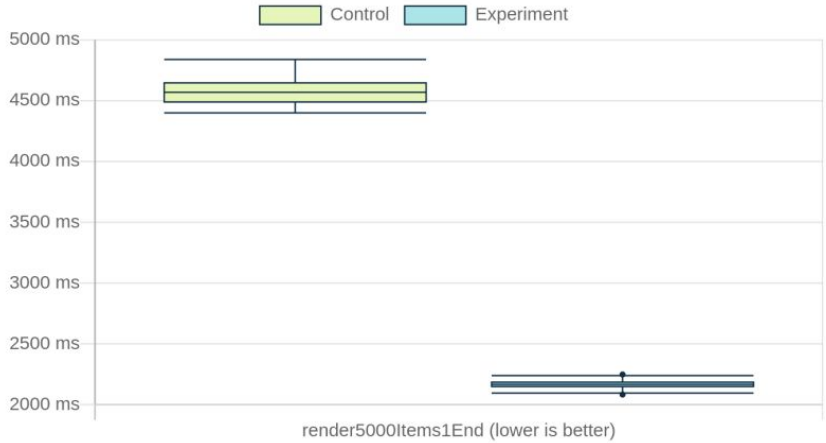


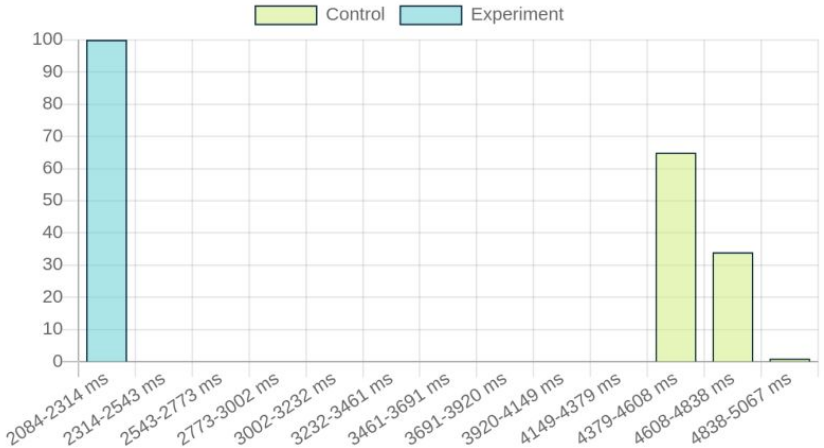


Тестируем

render5000Items1End (2400 ms faster)

Based on the P-value of this benchmark the evidence for a metric shift is **very strong.** TracerBench has determined the results are **significant** meaning they are worth looking at. A statistics estimator *(Hodges–Lehmann estimator)* was used to determine "Experiment" is **faster** by **2400 ms**. TracerBench is 95% confident "Experiment" is **faster** between **2377 ms to 2422 ms** based on 100 samples using a *(confidence interval)*.

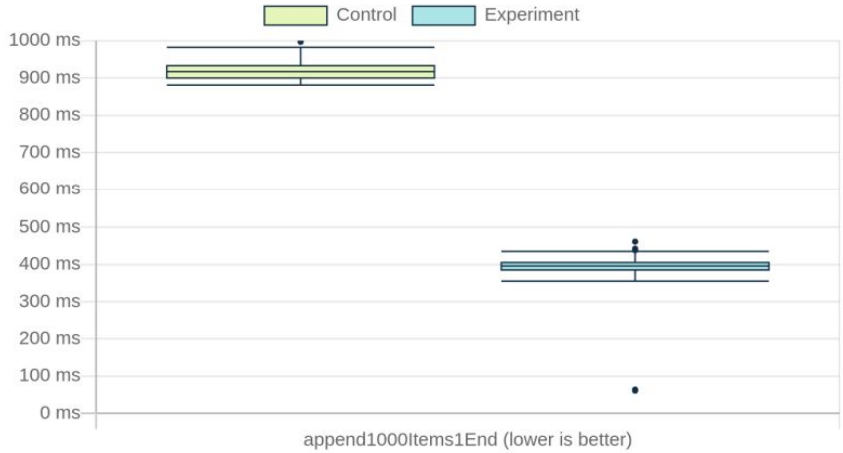


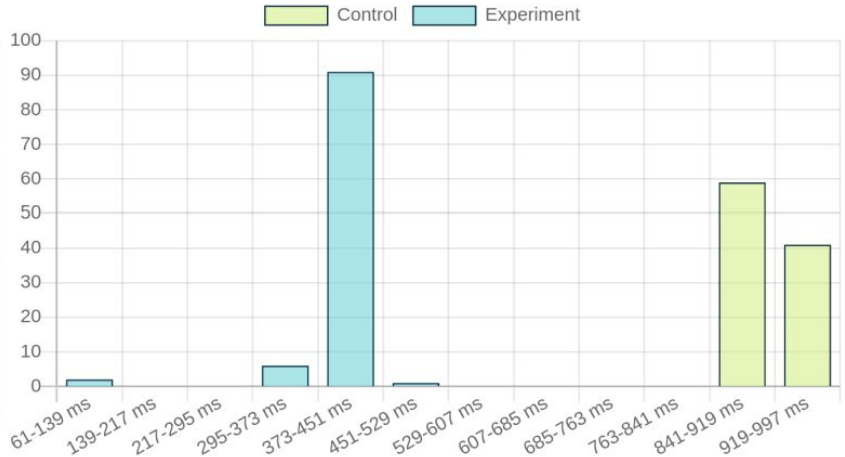


Тестируем

append1000Items1End **(524 ms faster)**

Based on the P-value of this benchmark the evidence for a metric shift is **very strong.** TracerBench has determined the results are **significant** meaning they are worth looking at. A statistics estimator *(Hodges–Lehmann estimator)* was used to determine "Experiment" is **faster** by **524 ms**. TracerBench is 95% confident "Experiment" is **faster** between **518 ms to 530 ms** based on 100 samples using a *(confidence interval)*.

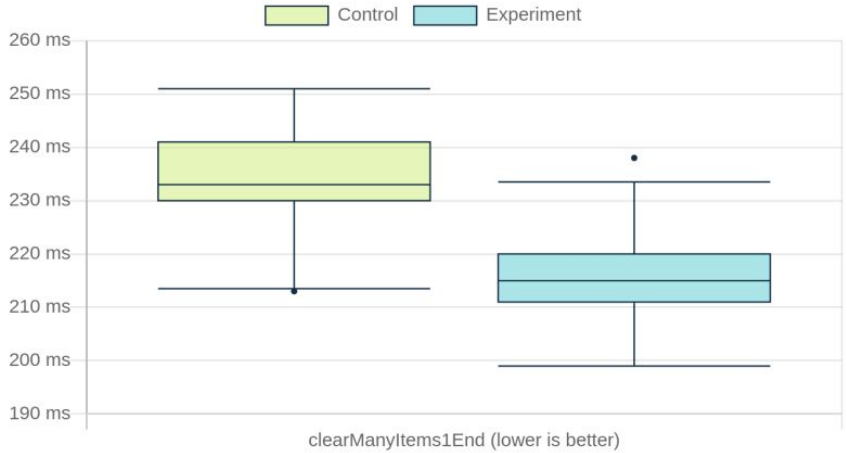


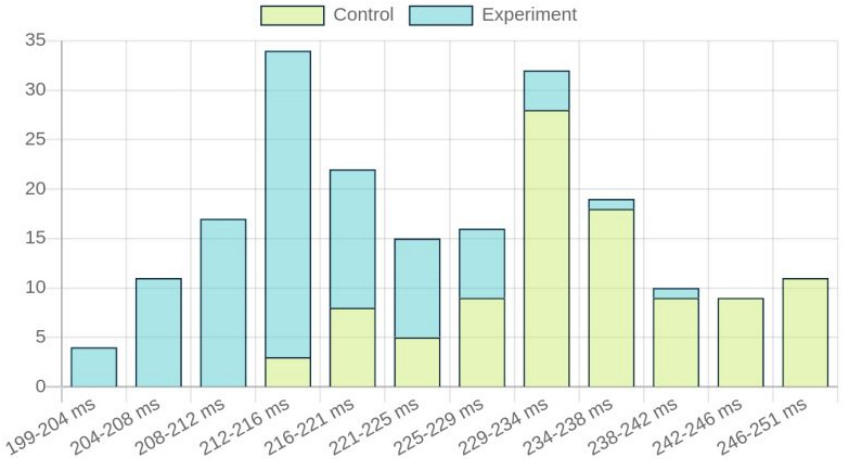


Тестируем

clearManyItems1End (19 ms faster)

Based on the P-value of this benchmark the evidence for a metric shift is **very strong.** TracerBench has determined the results are **significant** meaning they are worth looking at. A statistics estimator *(Hodges–Lehmann estimator)* was used to determine "Experiment" is **faster** by **19 ms**. TracerBench is 95% confident "Experiment" is **faster** between **17 ms to 21 ms** based on 100 samples using a *(confidence interval)*.

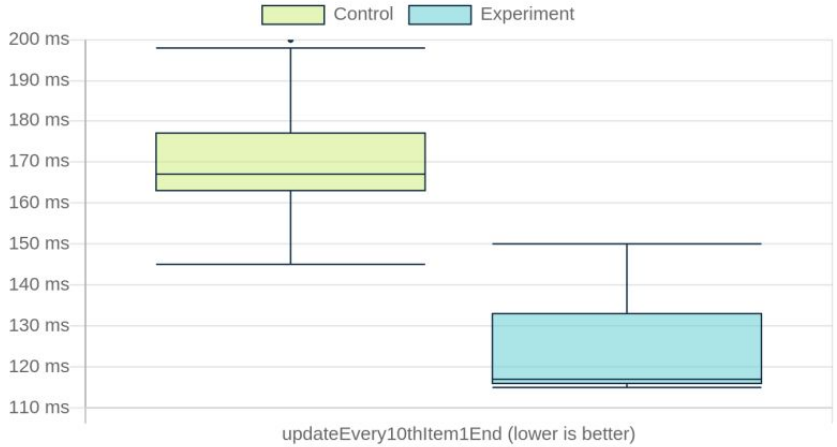


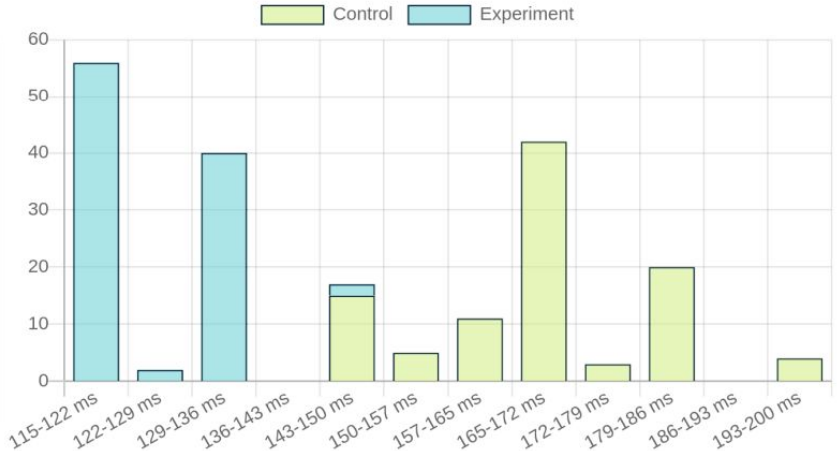


Тестируем

updateEvery10thItem1End **(47 ms faster)**

Based on the P-value of this benchmark the evidence for a metric shift is **very strong.** TracerBench has determined the results are **significant** meaning they are worth looking at. A statistics estimator *(Hodges–Lehmann estimator)* was used to determine "Experiment" is **faster** by **47 ms**. TracerBench is 95% confident "Experiment" is **faster** between **38 ms to 49 ms** based on 100 samples using a *(confidence interval)*.

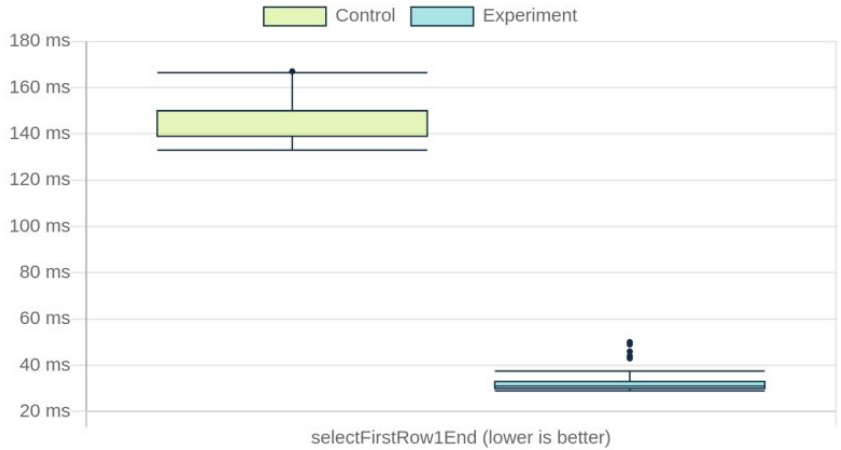


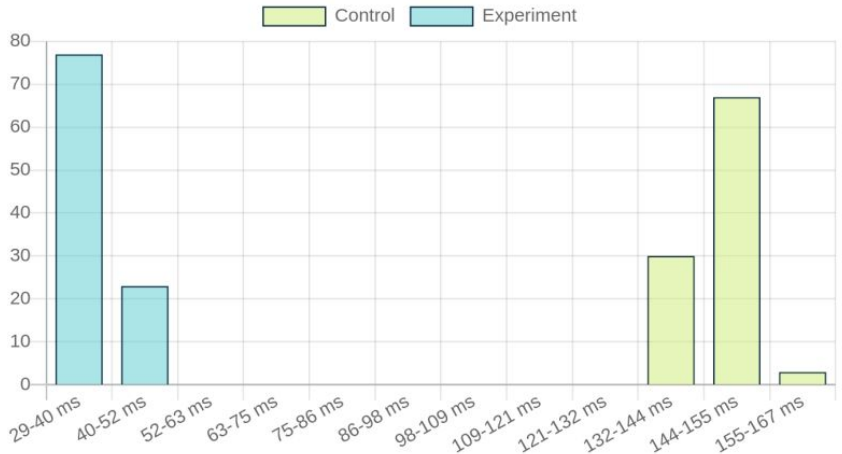


Тестируем

selectFirstRow1End (118 ms faster)

Based on the P-value of this benchmark the evidence for a metric shift is **very strong.** TracerBench has determined the results are **significant** meaning they are worth looking at. A statistics estimator *(Hodges–Lehmann estimator)* was used to determine "Experiment" is **faster** by **118 ms**. TracerBench is 95% confident "Experiment" is **faster** between **108 ms to 118 ms** based on 100 samples using a *(confidence interval)*.





```ts
export function ButtonComponent(
  { onClick, text, slot, id }: { onClick: () => void; text: string; slot?: Node; id?: string },
  outlet: HTMLElement
) {
  const button = document.createElement('button');
  button.setAttribute('class', 'btn btn-primary btn-block');
  button.type = 'button';

  const textNode = document.createTextNode(text);
  if (id) {
    button.setAttribute('id', id);
  }
  button.appendChild(textNode);
  if (slot) {
    button.appendChild(slot);
  }

  outlet.appendChild(button);

  return {
    nodes: [button],
    destructors: [addEventListener(button, 'click', onClick)],
    index: 0,
  };
}
```

```
this.children.push(
  ButtonComponent(
    {
      onClick: () => this.create_1_000_Items(),
      text: 'Create 1000 items',
      id: 'run',
    },
    btnW1
  ),
  ButtonComponent(
    {
      onClick: () => this.create_5_000_Items(),
      text: 'Create 5 000 items',
      id: 'runlots',
    },
    btnW2
  ),
  ButtonComponent(
    {
      onClick: () => this.append_1_000_Items(),
      text: 'Append 1000 rows',
      id: 'add',
    },
    btnW3
  ),
```

А что дальше?

А что дальше?

Попробуем сделать из этого

что-то большее

А что дальше?

Попробуем сделать из этого

что-то большее

за месяц

Web Framework in 2024

Requirements

Functional

- Реактивность
- Скорость работы
- Поддержка порталов
- Слоты
- Простая работа с Shadow DOM
- Tree Shaking
- SSR
- Rehydration
- Поддержка анимаций

Requirements

Developer Experience

- Поддержка JSX или какого-то шаблонизатора
- HMR
- Типизация
- DevTools
- Тесты должны работать в браузере
- Подсветка синтаксиса на GitHub

Requirements

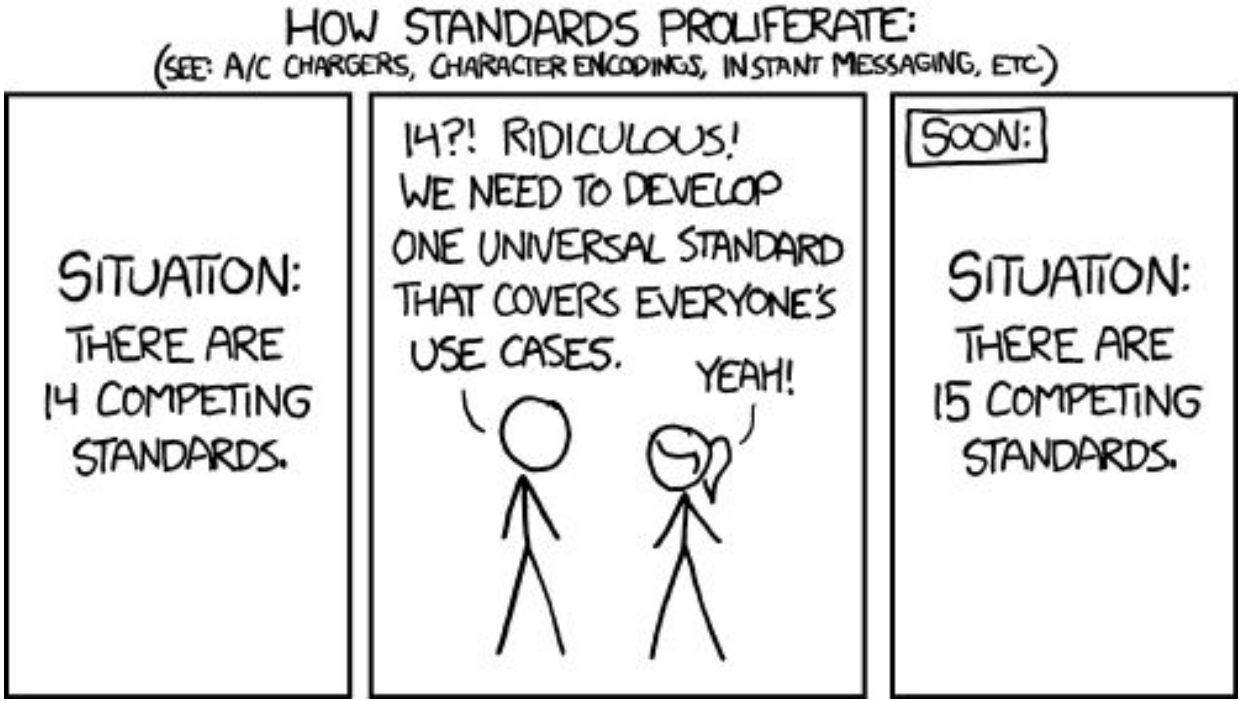
Functional

- Реактивность
- Скорость работы
- Поддержка порталов
- Слоты
- Простая работа с Shadow DOM
- Tree Shaking
- SSR
- Rehydration
- Поддержка анимаций

Developer Experience

- Поддержка JSX или какого-то шаблонизатора
- HMR
- Типизация
- DevTools
- Тесты должны работать в браузере
- Подсветка синтаксиса на GitHub

Ready?

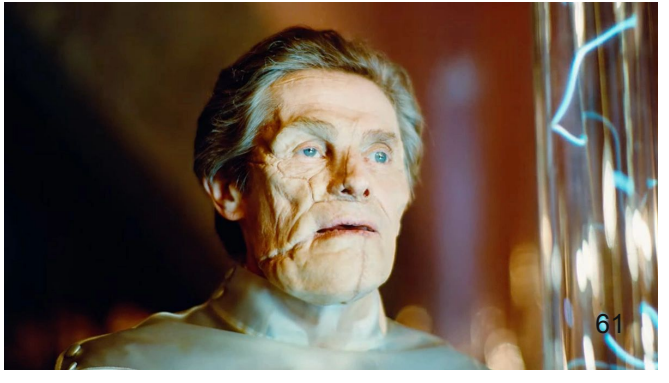

HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Просто добавь 10 лет опыта

- работа с Babel
- работа с AST деревьями
- вычитка исходного кода Vue, React, Svelte, Ember
- работа над инструментами для разработчиков:

  - transpiler, linter, language server, devtools

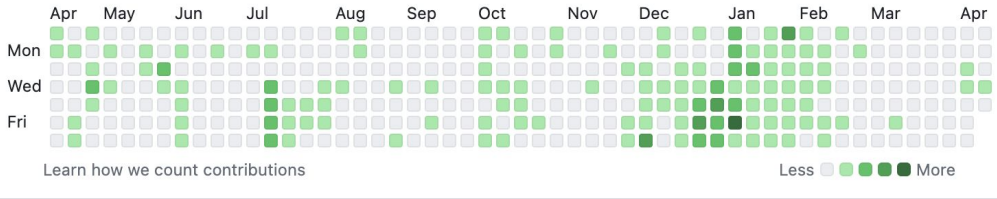
- > 80 собственных github проекта
- > 150 контрибьюшнов в сторонние проекты
- профилирование реальных приложений



Просто добавь 10 лет опыта

959 contributions in the last year

Contribution settings



Learn how we count contributions

Less More

@glimmerjs @Brain-up @ember-tooling More

Activity overview

Contributed to lifeart/glimmer-next, lifeart/demo-ember-vite, lifeart/sm-annotate and 41 other repositories

5%

Alex Kanunnikov's GitHub Stats

Total Stars Earned:	740
Total Commits (2024):	654
Total PRs:	989
Total Issues:	680
Contributed to (last year):	18

Просто добавь 10 лет опыта

НАСМОТРЕННОСТЬ

Давайте попробуем сдел

Разбираемся по пунктам

Разбираемся по пунктам

Поддержка JSX или какого-то шаблонизатора

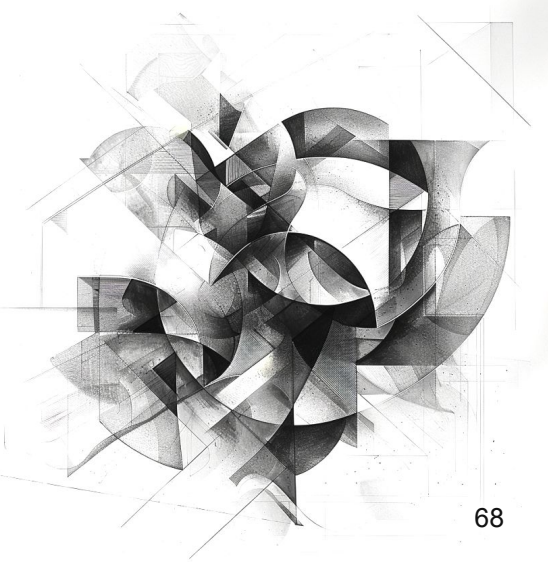
1. Выбираем синтаксис
2. Подбираем парсер для синтаксиса
3. Пишем компилятор

Синтаксис

- набор языковых конструкций, с помощью которых мы можем описать желаемое состояние DOM

Синтаксис

JavaScript, JSX, Svelte, Vue, Angular, Glimmer …



Синтаксис

JSX по количеству динамических частей превосходит все вышеперечисленные синтаксисы

Синтаксис

JSX по количеству динамических частей превосходит все вышеперечисленные синтаксисы, а это означает что нужно обработать больше корнер-кейсов, чего не хотелось делать на раннем этапе.

тут я транслирую JSX

Синтаксис

{{ **Glimmer Handlebars** }}

Синтаксис

{{ **Glimmer Handlebars** }}

handlebars

Парсер синтаксиса

- используется для анализа шаблона и его преобразования в объектное представление, в котором можно различить статические и динамические части.

Парсер синтаксиса

В общем случае у нас не так много динамичных частей на странице:

- аргумент компонента
- атрибут / свойство элемента
- текст

```
<Foo @name={{name}} />

<img src={{photo}} />

<span>{{age}}</span>
```

Парсер синтаксиса

babel

vue-eslint-parser

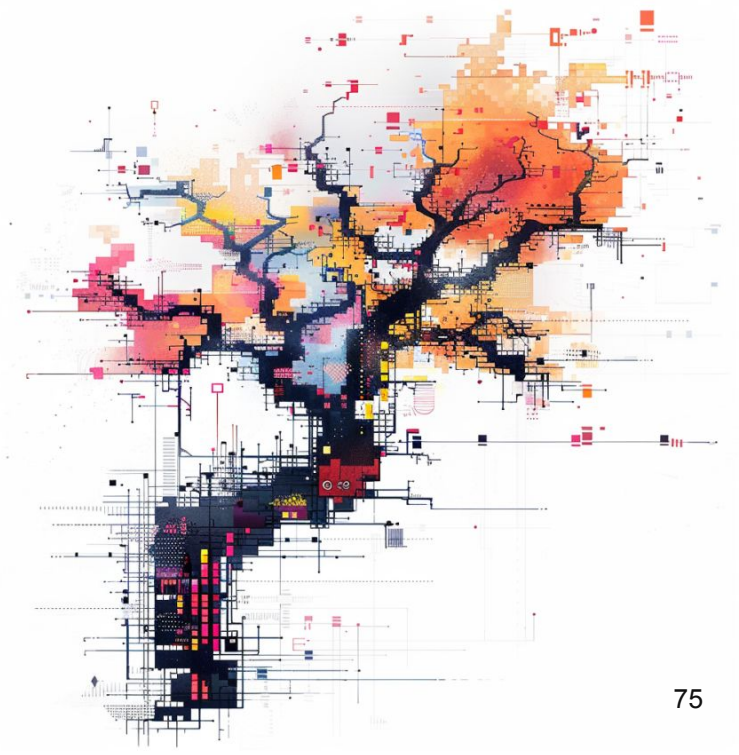
svelte

htmlparser2

parse5

angular

@glimmer/syntax



Парсер синтаксиса

Так как мы выбрали handlebars подобный синтаксис,

берём готовый парсер - **@glimmer/syntax**

Компилятор

Компилятор

- нужен для того чтобы из **объектного** представления, полученного на этапе парсинга можно было **создать код**, который непосредственно отвечает за отображение компонента в том виде, в котором его задумал разработчик.

Компилятор

В общем виде компилятор преобразует код вида

```
const Template = {
  type: 'Element',
  tag: 'div',
  children: [
    {
      type: 'Text',
      value: 'Hello World',
    }
  ]
}
```

Компилятор

в соответствующий JavaScript код

```
const Template = {
  type: 'Element',
  tag: 'div',
  children: [
    {
      type: 'Text',
      value: 'Hello World',
    }
  ]
}
```

```
const node = document.createElement('div');
node.textContent = 'Hello World';
```

Компилятор

в соответствующий JavaScript код

```
const Template = {
  type: 'Element',
  tag: 'div',
  children: [
    {
      type: 'Text',
      value: 'Hello World',
    }
  ]
}
```

```
const node = document.createElement('div');
node.textContent = 'Hello World';
```

исполняется в браузере, создавая необходимую DOM структуру

Движок

Движок

набор стратегий, с помощью которых мы создаём компоненты, DOM элементы и держим их в актуальном состоянии

Движок - состояния компонента

- Рендеринг
- Удаление

Движок - состояния компонента (расширенные)

- Создание
- Рендеринг
- Обновление
- Удаление

Движок - базовые примитивы

Компонент

```
<MyComponent />
```

DOM элемент

```
<input />
```

Текст

```
HELLO
```

Комментарий

```
<!-- hide me -->
```

Хэлпер (функция, используемая в шаблонизаторе)

```
{{toUpperCase "world"}}
```

Модифаер (функция-эффект, применимая к DOM элементу)

```
<div {{on 'click' this.handleClick}}></div>
```

Движок - базовые примитивы

```
<MyComponent />

<input />

HELLO

<!-- hide me -->

{{toUpperCase "world"}}

<div {{on 'click' this.handleClick}}></div>
```

Движок - управляющие конструкции

Движок - управляющие конструкции

- набор функций, реализующих управление отображением элементов страницы

Движок - управляющие конструкции

IF

EACH

SLOT

IN_ELEMENT

Движок - управляющие конструкции: IF

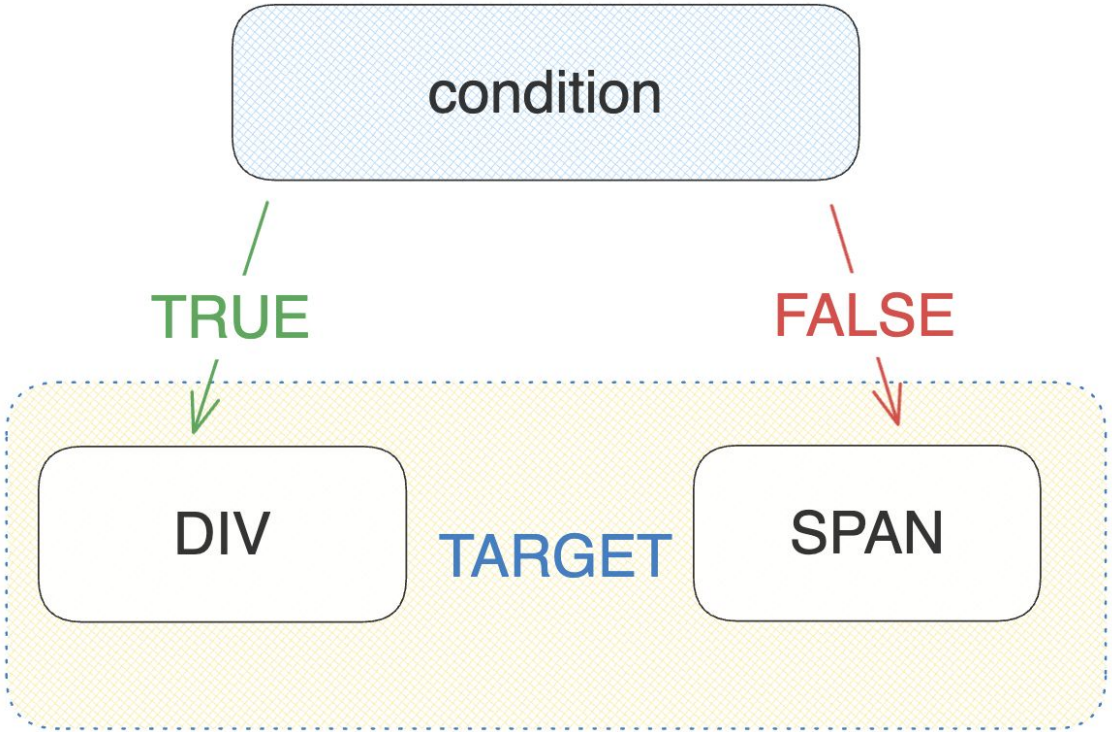
компонент, содержащий низкоуровневую логику рендеринга/удаления других компонентов

Движок - управляющие конструкции: IF

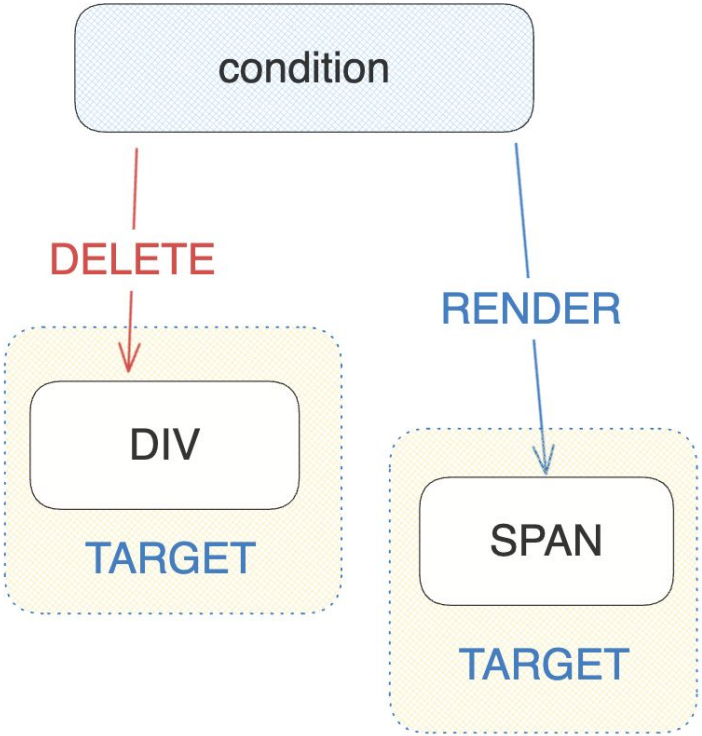
аргументы

1. Condition - выражение, которое мы проверяем
2. True Branch - ветка, которую выполняем в TRUE случае
3. False Branch - ветка, которую выполняем в FALSE случае
4. Render Target - место, куда нам нужно отрендериться

Движок - управляющие конструкции: IF



Движок - управляющие конструкции: IF



```
function ifCondition(
    cell: Formula<boolean>, target: DocumentFragment | HTMLElement,
    trueBranch: () => GenericReturnType,
    falseBranch: () => GenericReturnType,
  ) {
    let prevComponent = null
    opcodeFor(cell, (value) => {
        const nextBranch = value ? trueBranch : falseBranch
        destroyComponent(prevComponent)
        prevComponent = nextBranch()
        renderElement(target, prevComponent)
    })
}
```

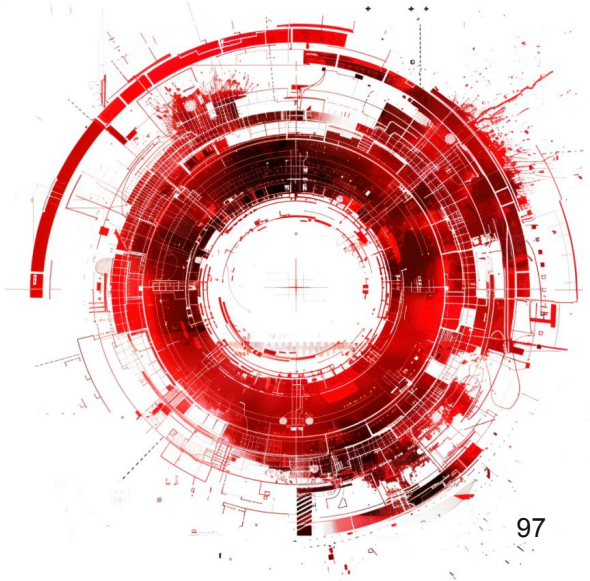
Движок - управляющие конструкции: IF

создаём формулу, которая проверяет кондишн, если он не изменился (не переключился), тогда ничего не делаем, в случае переключения -

1. Синхронно или асинхронно удаляем компонент предыдущей бранчи
2. Создаём компонент новой бранчи и рендерим его

Движок - управляющие конструкции: EACH

компонент, позволяющий отрендерить переданный компонент **N** раз

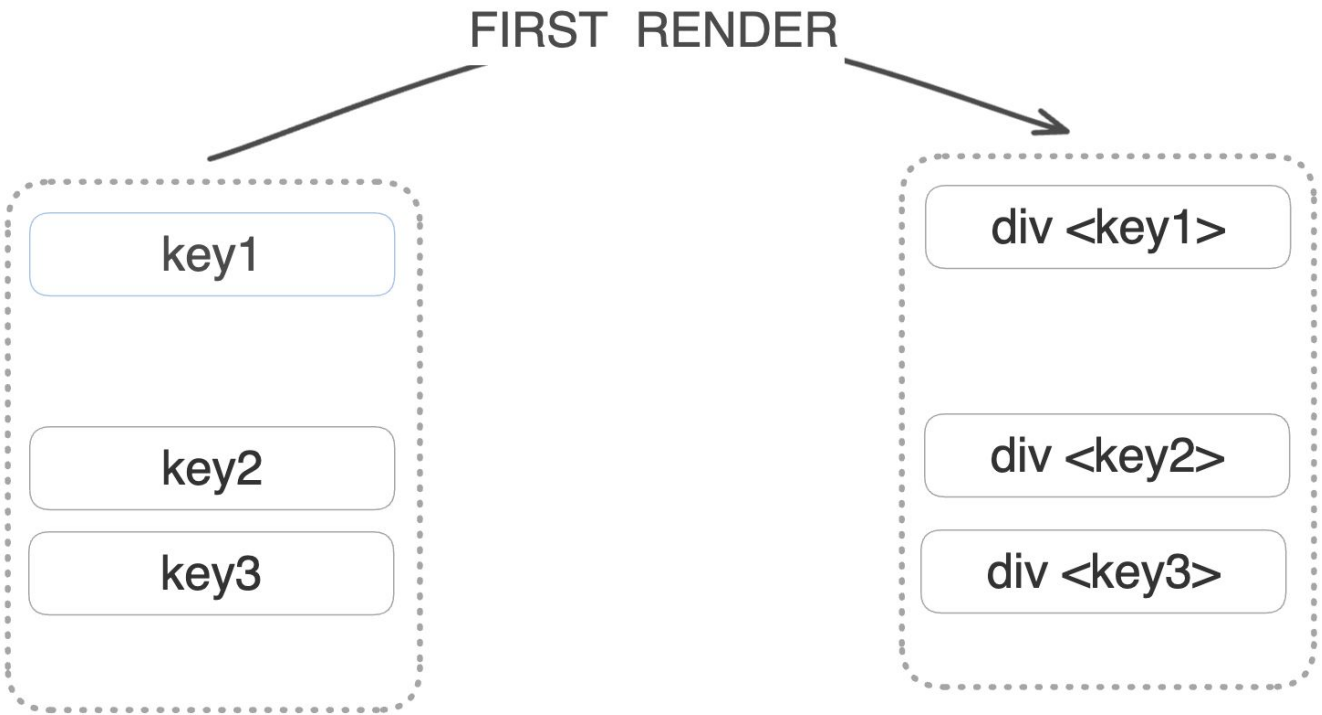


Движок - управляющие конструкции: EACH

аргументы

1. Множество для итерации
2. Компонент который нужно отрендерить
3. Ключ для поиска соответствия компонентов
4. Render Target - место, куда нам нужно отрендериться

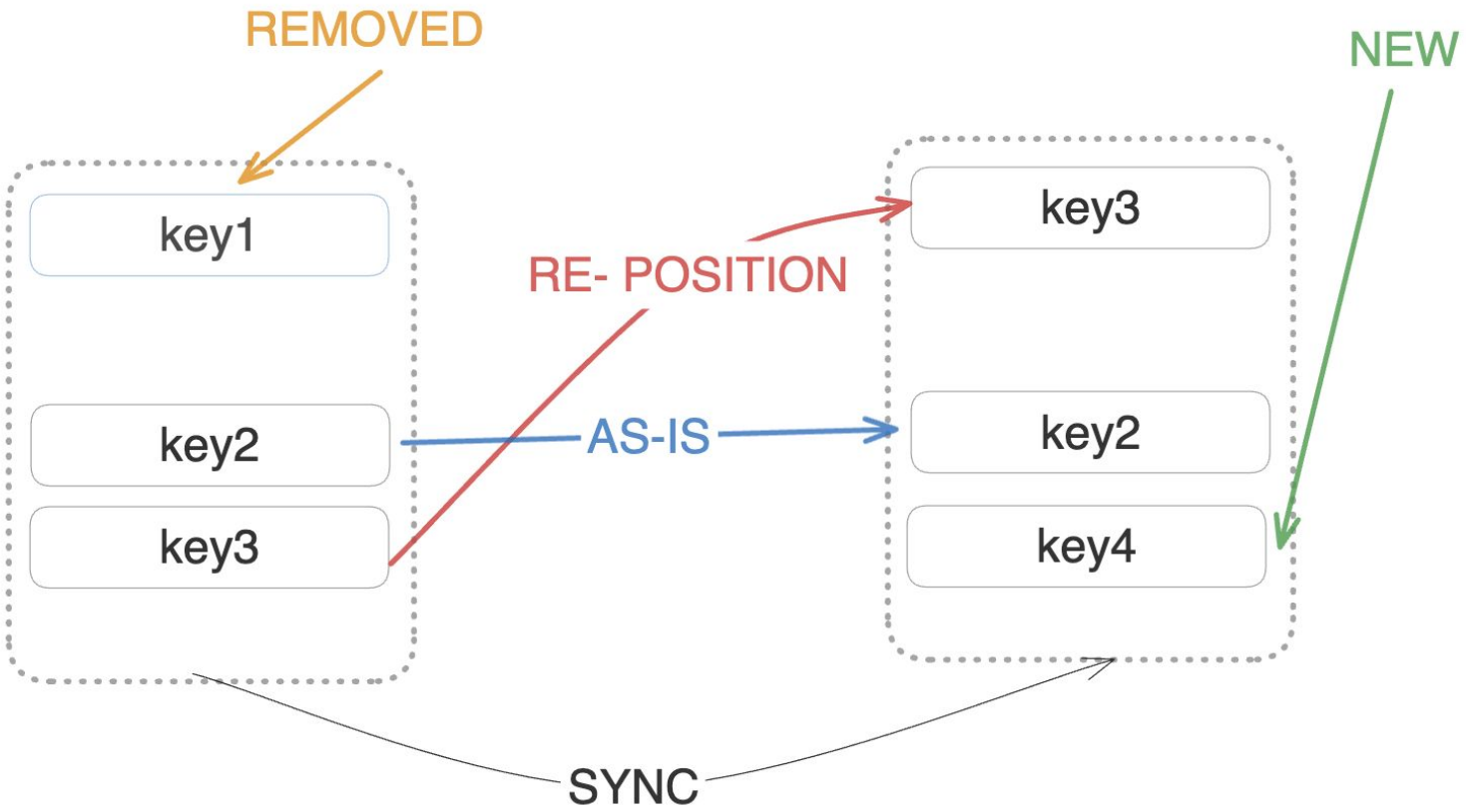
Движок - управляющие конструкции: EACH



Движок - управляющие конструкции: EACH

жизненный цикл EACH заключается в **реакции** на **изменение множества**, которое мы ему передали

Движок - управляющие конструкции: EACH



Движок - управляющие конструкции: EACH

Получаем ключи множества, которое нужно отрендерить и сравниваем с теми, которые были отрендерены:

если в новом массиве какие-то элементы отсутствуют

- нам нужно их удалить

(запускаем синхронное или асинхронное удаление)

Движок - управляющие конструкции: EACH

Итерируемся по оставшимся элементам, нам нужно выполнить 2 условия -

убедиться что компонент создан

(если нет - создать)

проверить что он находится на своём месте

(если нет - перенести на нужную позицию - синхронизировать позицию)

```
function eachCondition(
    cell: Formula<boolean>, target: DocumentFragment | HTMLElement,
    key: string, component: () => GenericReturnType,
  ) {
    const existingComponents = new Map()
    opcodeFor(cell, (items) => {
      for (item in items) {
        const id = item[key]
        if (existingComponents.has(id)) {
          continue
        }
        const instance = component(item)
        appendElement(target, instance)
        existingComponents.set(id, instance)
      }
    })
}
```

Простая работа с Shadow DOM

Хотелось иметь возможность создавать Shadow ноды наподобие декларативного Shadow DOM, только в рантайме

Простая работа с Shadow DOM

```
<host-element>
    <template shadowrootmode="open">
        <slot></slot>
    </template>
    <h2>Light content</h2>
</host-element>
```

Простая работа с Shadow DOM

```
<div shadowrootmode="open">
    I'm in the shadow
</div>
```

Простая работа с Shadow DOM

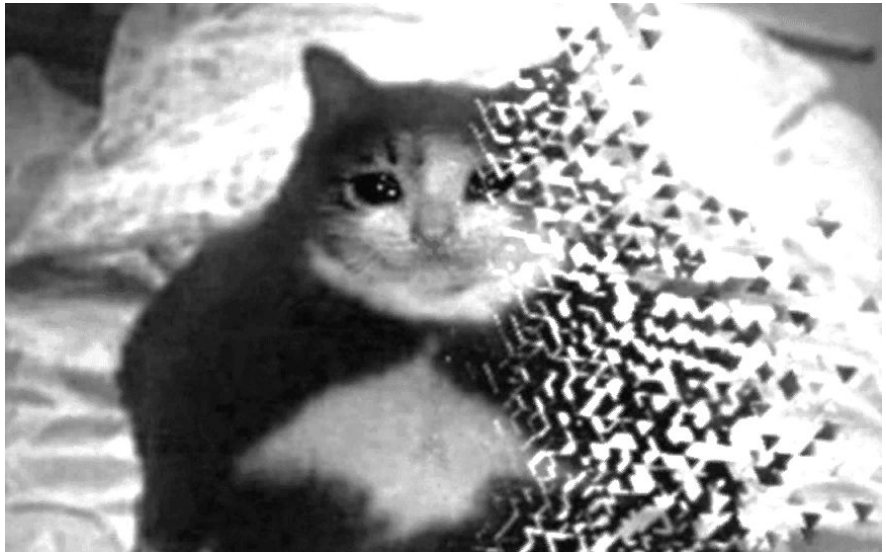
```
const appendRef = hasShadowMode !== null
  ? element.attachShadow({ mode: hasShadowMode }) || element.shadowRoot
  : element;

children.forEach((child, index) => {
  addChild(appendRef, child, destructors, index);
});
```

Tree Shaking

Tree Shaking

Позволяет нам **избавиться** от кода, на который ничего из нашего entry файла не ссылается.



Tree Shaking

Сгенерированный код не содержит конструкций вида:

```
import React from 'react';

React.createElement('div');
React.createText(42);
```

Tree Shaking

Но, содержит такой код:

```
import { element, text } from 'gxt';

element('div');
text(42);
```

Tree Shaking

- явно импортируем только те части которые нам нужны

- не засоряем рантайм неиспользуемой функциональностью

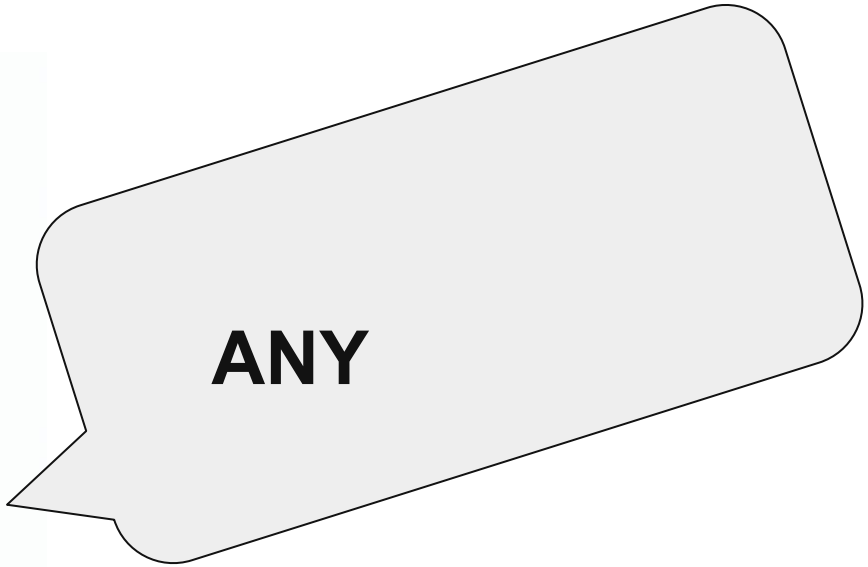
Tree Shaking

DEFINE флаги Vite

```
compiler(mode, {
  flags: {
    TRY_CATCH_ERROR_HANDLING: false,
    SUPPORT_SHADOW_DOM: false,
  }
})
```

Типизация в шаблонизаторах

Типизация в шаблонизаторах



Типизация в шаблонизаторах

Если мы выбрали какой-то существующий синтаксис, то можем воспользоваться созданными инструментами, а если они перестали работать - то починить их

Типизация в шаблонизаторах

Если синтаксис **новый**, тогда придётся пойти во все **тяжкие**, которые в общем случае состоят из двух подходов

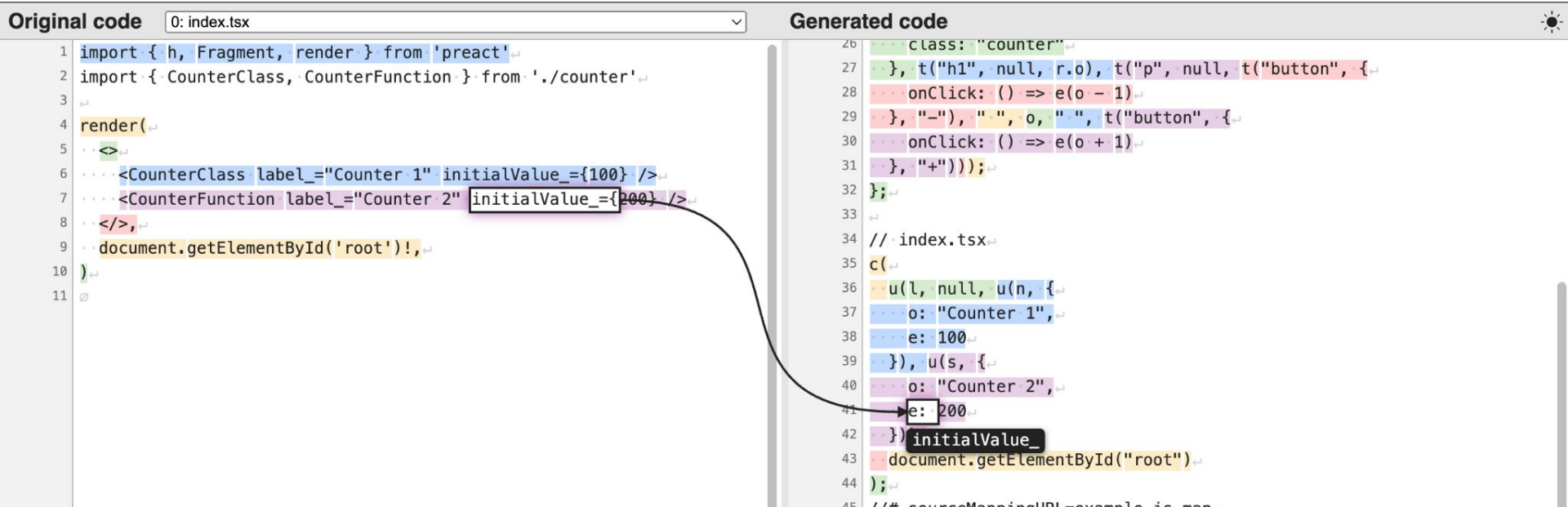
Типизация в шаблонизаторах #1

1. имплементировать сорсмапы на уровне компилятора
2. тайпчекать скомпилированный код
3. ре-мапить ошибки на синтаксис шаблона

* требует хорошей типизации скомпилированных результатов.

Типизация в шаблонизаторах #1

1. имплементировать сорсмапы на уровне компилятора
2. тайпчекать скомпилированный код
3. ре-мапить ошибки на синтаксис шаблона



Типизация в шаблонизаторах #2

1. написать ещё один компилятор
2. конвертировать шаблон в набор TypeScript выражений

- позволяет упростить представление результирующего кода и не сталкиваться с проблемами некорректной типизации runtime.

Типизация в шаблонизаторах #2

1. написать ещё один компилятор
2. конвертировать шаблон в набор TypeScript выражений

- позволяет упростить представление результирующего кода и не сталкиваться с проблемами некорректной типизации runtime.

```
templateForBackingValue(YieldingComponent, function (Γ) {
  expectTypeOf(Γ.this).toBeNull();
  expectTypeOf(Γ.args).toEqualTypeOf<YieldingComponentSignature['Args']>();
  expectTypeOf(Γ.element).toEqualTypeOf<YieldingComponentSignature['Element']>();
  expectTypeOf(Γ.blocks).toEqualTypeOf<YieldingComponentSignature['Blocks']>();
});
```

Реактивность компонентов

```
const text = cell('hello')

opcode(text, (value) => {
  div.textContent = value // hello
})
```

Реактивность компонентов

```
const text = cell('hello')

const transform = formula(() => toUpperCase(text.value))

opcode(
  transform,
  (value) => {
    div.textContent = value; // HELLO
  }
)
```

Реактивность компонентов

На слайдах ранее мы с вами разобрали концепцию реактивности. Остаётся описать синтаксический сахар.

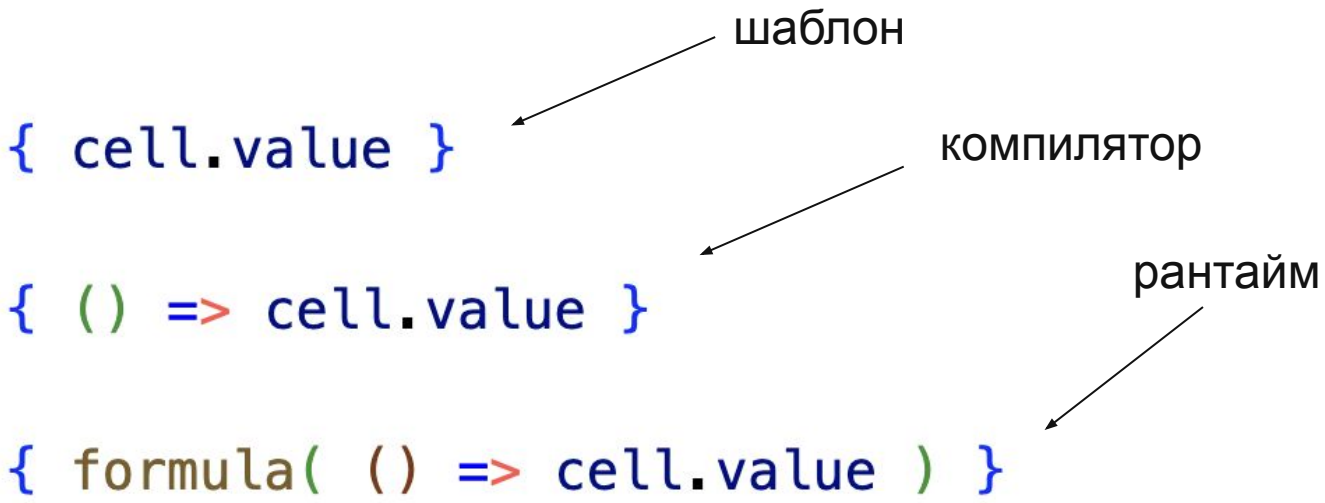
Реактивность компонентов

На слайдах ранее мы с вами разобрали концепцию реактивности. Остаётся описать синтаксический сахар.

cell -> formula(cell) -> opcode(formula(cell))

Реактивность компонентов

На слайдах ранее мы с вами разобрали концепцию реактивности. Остаётся описать синтаксический сахар.



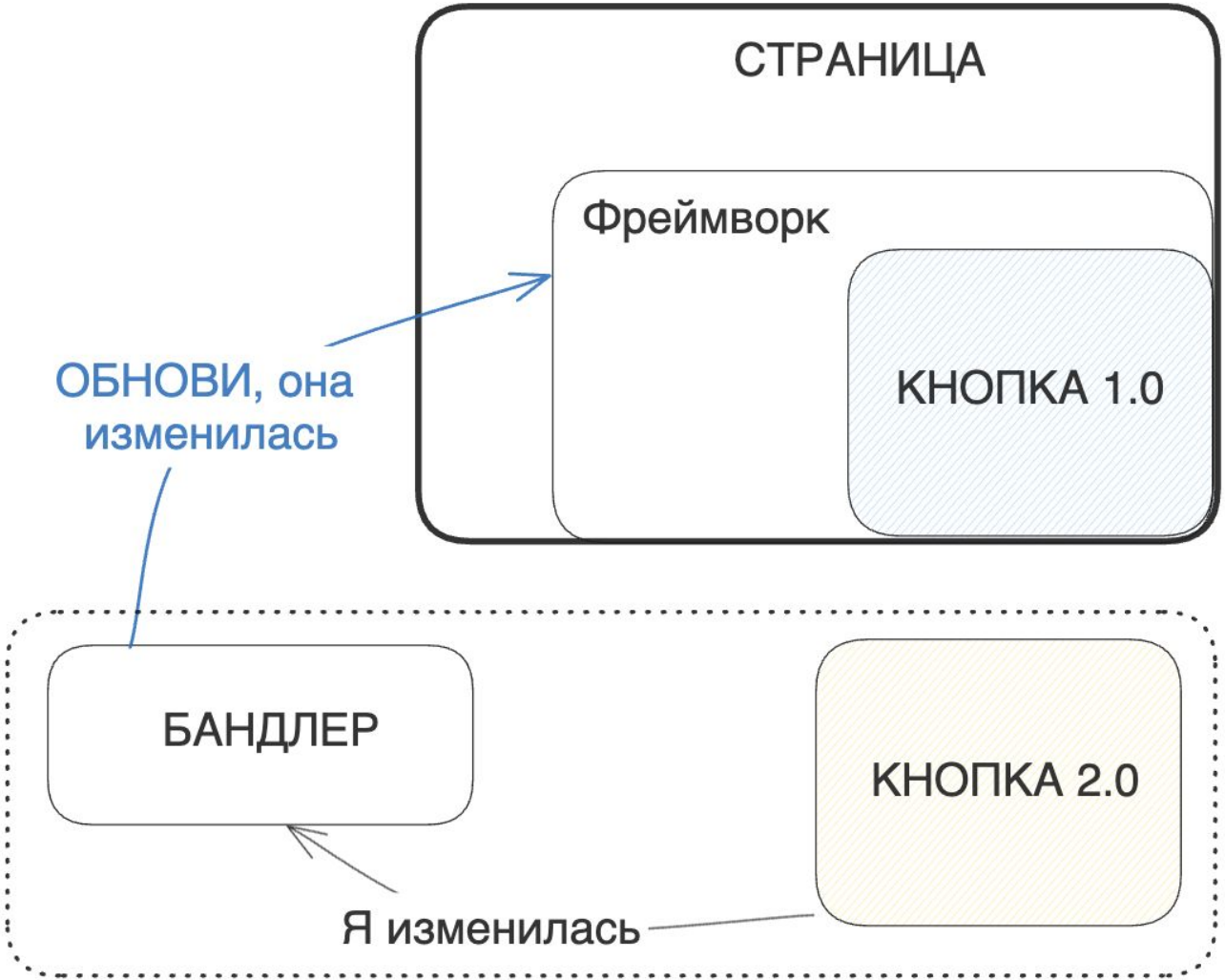
HMR

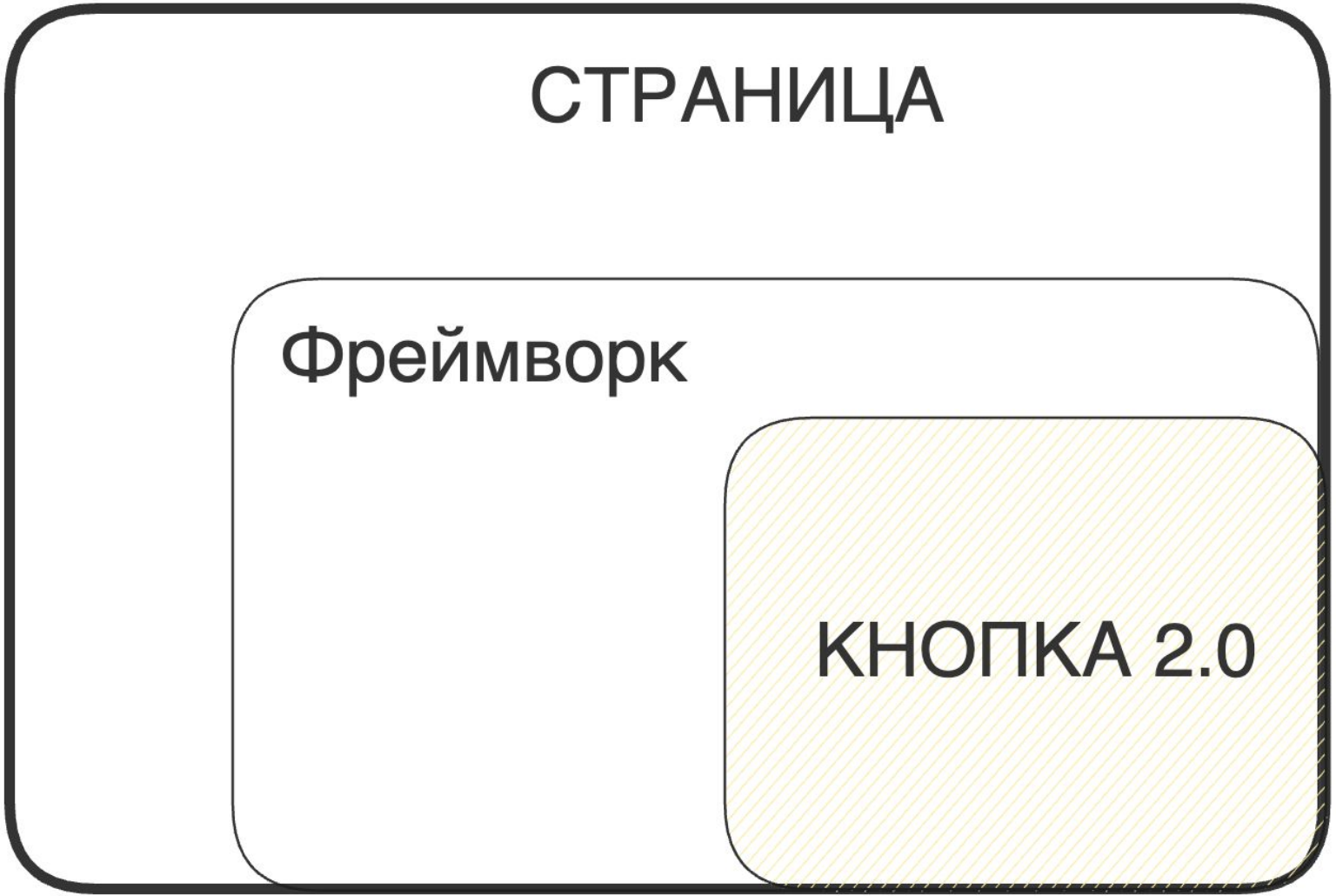
HMR

при **изменении** файла **компонента** мы не перезагружаем приложение, а **рендерим** заново только **изменённые** компоненты.

HMR

- бандлер **предоставляет API**, которое вызывается во время изменения файла
- фреймворк **имплементирует** логику обновления


СТРАНИЦА
Фреймворк
КНОПКА 1.0
ОБНОВИ, она изменилась
БАНДЛЕР
КНОПКА 2.0
Я изменилась


СТРАНИЦА
Фреймворк
КНОПКА 2.0

HMR: Менеджер компонентов

```
function MyComponent() { return <div></div> }

MyComponent()

ComponentManager(MyComponent)
```

HMR: Менеджер компонентов

```
function ComponentManager (Component, args, targetNode) {
  const instance = new Component(args)
  render (instance, targetNode)
  RenderedComponents.add(Component, {instance, args, targetNode})
}
```

HMR: Менеджер компонентов

```
RenderedComponents.add(Component, {instance, args, targetNode})
```

HMR: Менеджер компонентов

```
window.hotReload = (OldReference, NewReference) => {
  const { instance, args, targetNode } = RenderedComponents.get(OldReference)
  delete(instance)
  ComponentManager(NewReference, args, targetNode)
}
```

```
window.hotReload = function hotReload(
  oldKlass: Component | ComponentReturnType,
  newKlass: Component | ComponentReturnType,
) {
  const renderedInstances = COMPONENTS_HMR.get(oldKlass);
  if (!renderedInstances) {
    return;
  }
  const renderedBuckets = Array.from(renderedInstances);

  renderedBuckets.forEach(({ parent, instance, args }) => {
    const newCmp = component(newKlass, args, parent);
    const firstElement = getFirstNode(instance);
    const parentElement = firstElement.parentNode!;
    renderElement(parentElement, newCmp, firstElement);
    destroyElement(instance);
  });

  COMPONENTS_HMR.delete(oldKlass);
};
```

SSR

SSR

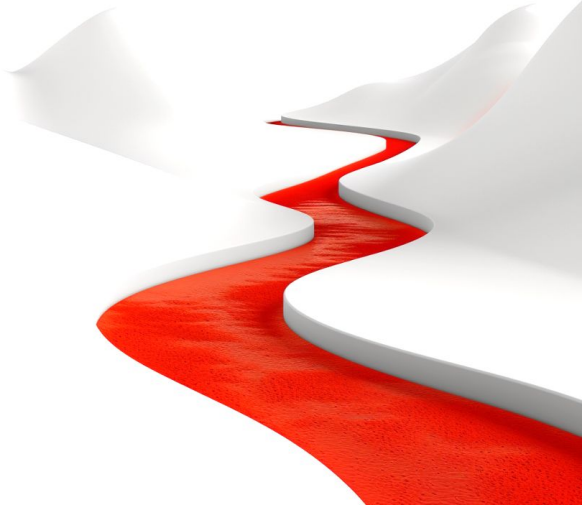
способность получать **HTML** репрезента

состояния нашего **приложения**

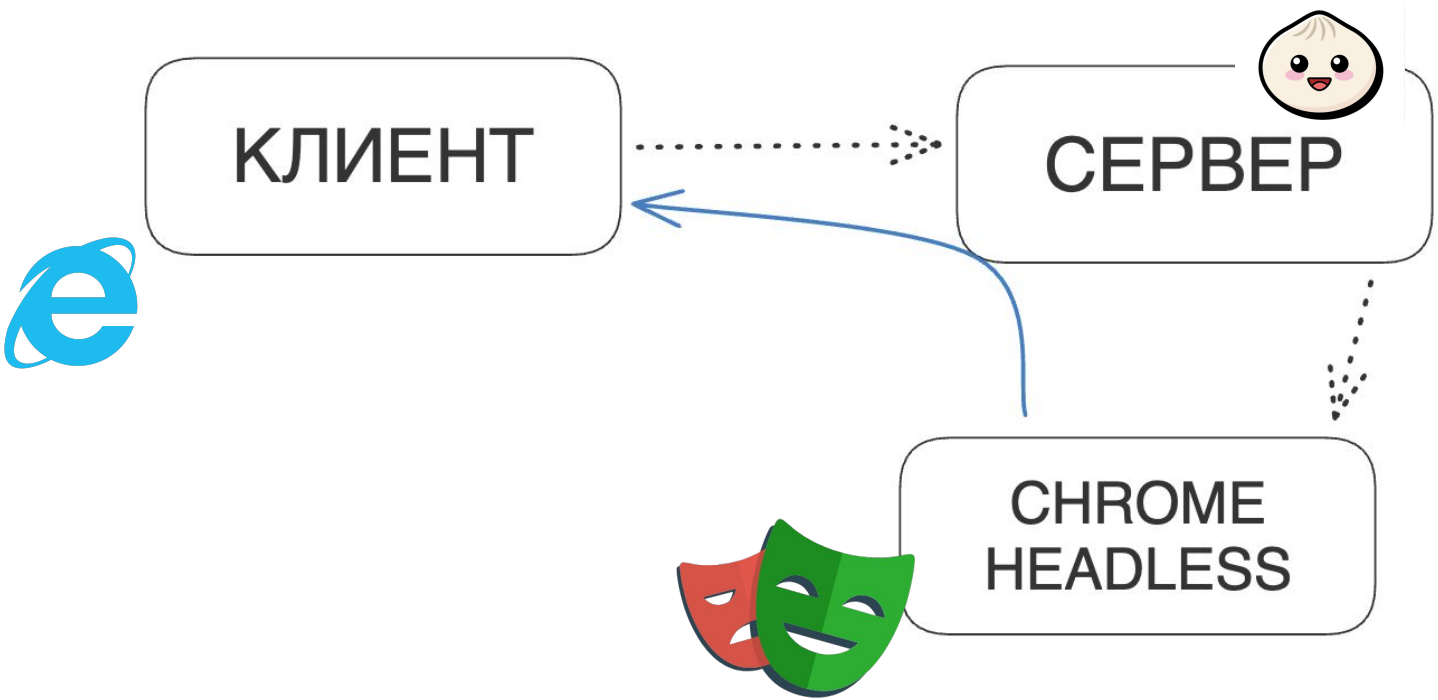
с **сервера**

SSR

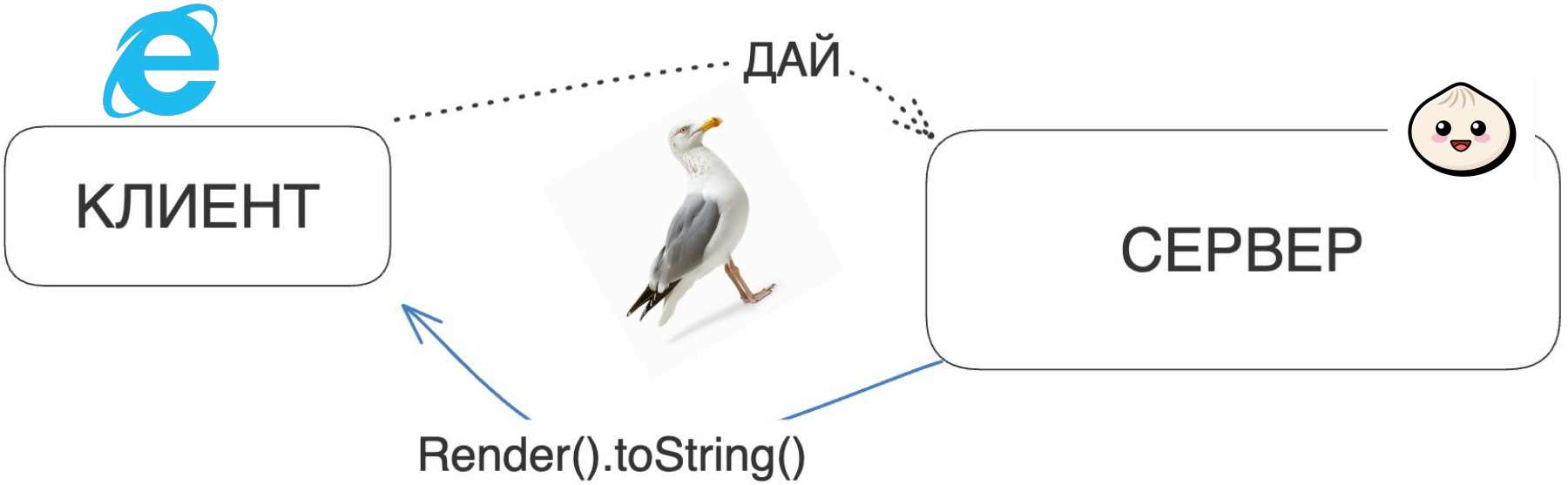
Существует несколько основных подходов, отличающихся как областью применения так и скоростью.



SSR #1



SSR #2



SSR

Я выбрал happy-dom потому что у неё **прикольный** логотип -

SSR: Happy DOM

эти библиотеки могут меняться как носки

не работает - возьми **другую**

SSR: Интеграция

новый слой абстракции **document**

```
const { Window } = await import('happy-dom');
const win = new Window({ url: params.url });

setDocument(win.document);
```

```
async function render(
  componentRenderFn: (rootNode: HTMLElement) => ComponentReturnType,
  params: EnvironmentParams,
) {
  const { Window } = await import('happy-dom');
  const win = new Window({ url: params.url });
  const doc = win.document;
  const rootNode = doc.createElement('div');

  setDocument(doc);

  doc.body.appendChild(rootNode);
  await componentRenderFn(rootNode);

  return rootNode.innerHTML;
}
```

Rehydration

Rehydration

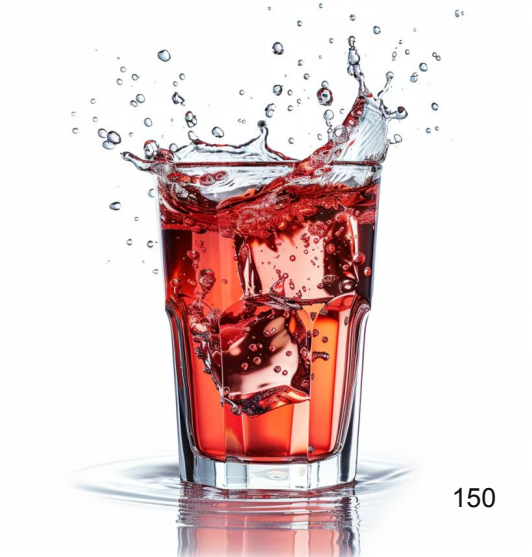
Регидрация - лучшая подруга SSR, хоть её некоторые и недолюбливают так как очень капризная.

Rehydration

Это концепция, в рамках которой во время начального рендеринга приложения мы **переиспользуем** уже **существующие** DOM ноды (отрендеренные на сервере и созданные браузером).

Rehydration

Чтобы это хоть как-то начало работать нам нужен уровень **абстракции**, который **вклинится** в логику **создания** DOM элементов и **возьмёт** уже готовый если **элемент** ранее был создан.

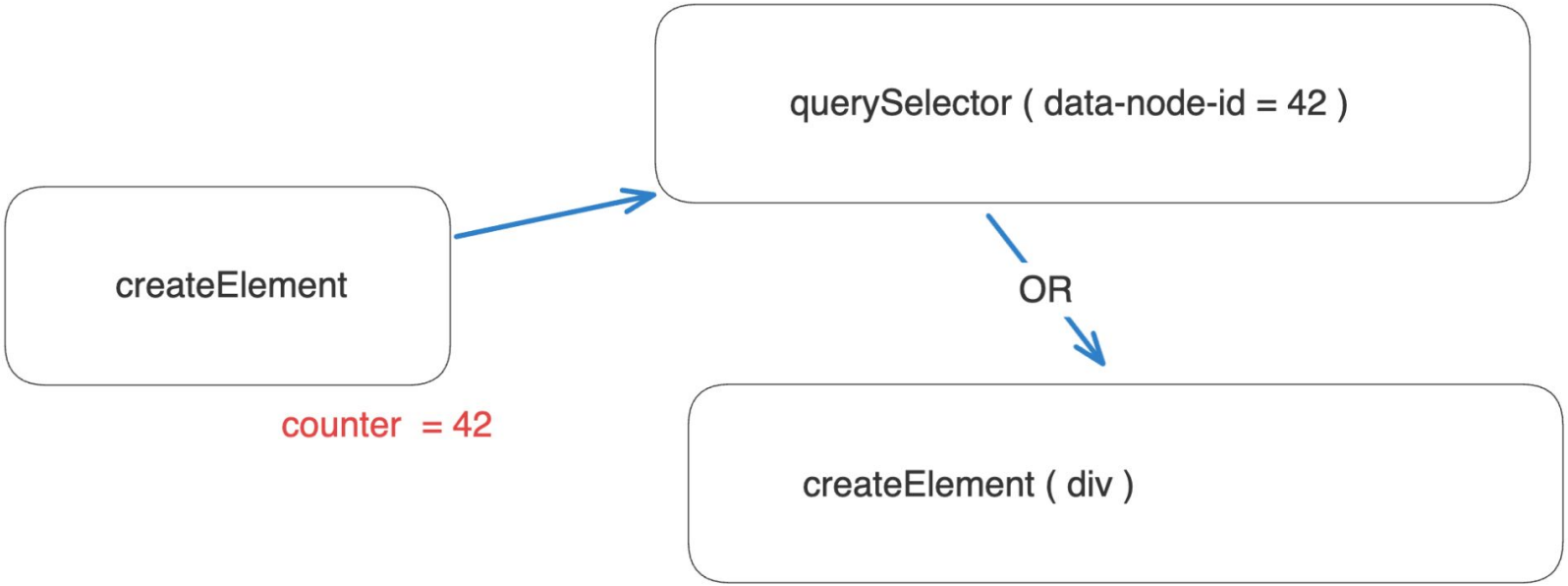


РЕГИДРАТИРОВЫВАЙ!

Rehydration

нужно **соотносить** существующие **элементы** с теми, которые мы пытаемся **создать** во время начального **рендеринга**

Rehydration



```
<div id="app"><header data-node-id="47" class="bg-white"><nav
    data-node-id="23"
    aria-label="Global"
    class="mx-auto flex max-w-7xl items-center justify-between p-6 lg:px-8"
  ><div data-node-id="6" class="flex lg:flex-1"><a
      data-node-id="5"
      href="#"
      class="-m-1.5 p-1.5"
    ><span data-node-id="3" class="sr-only">GXT</span><img
        data-node-id="4"
        width="32"
        height="32"
        src="/logo.png"
        alt=""
        class="h-8 w-auto"
      /></a></div><div data-node-id="11" class="flex lg:hidden"><button
      data-node-id="10"
      type="button"
      class="-m-2.5 inline-flex items-center justify-center rounded-md p-2.5
    ><span data-node-id="7" class="sr-only">Open main menu</span><svg
        data-node-id="9"
```

Rehydration

```
export function withRehydration(
  componentCreationCallback: () => ComponentReturnType,
  targetNode: HTMLElement
) {
  pushToStack(targetNode, true);
  resetNodeCounter();
  patchDOMAPI();
  componentCreationCallback();
  rollbackDOMAPI();
}
```

DevTools

инспекция компонентов, реактивных примитивов

DevTools

При работе с любым фреймворком крайне важны инструменты для разработчиков, которые позволяют увидеть состояние приложения и список отрендеренных компонентов.

DevTools

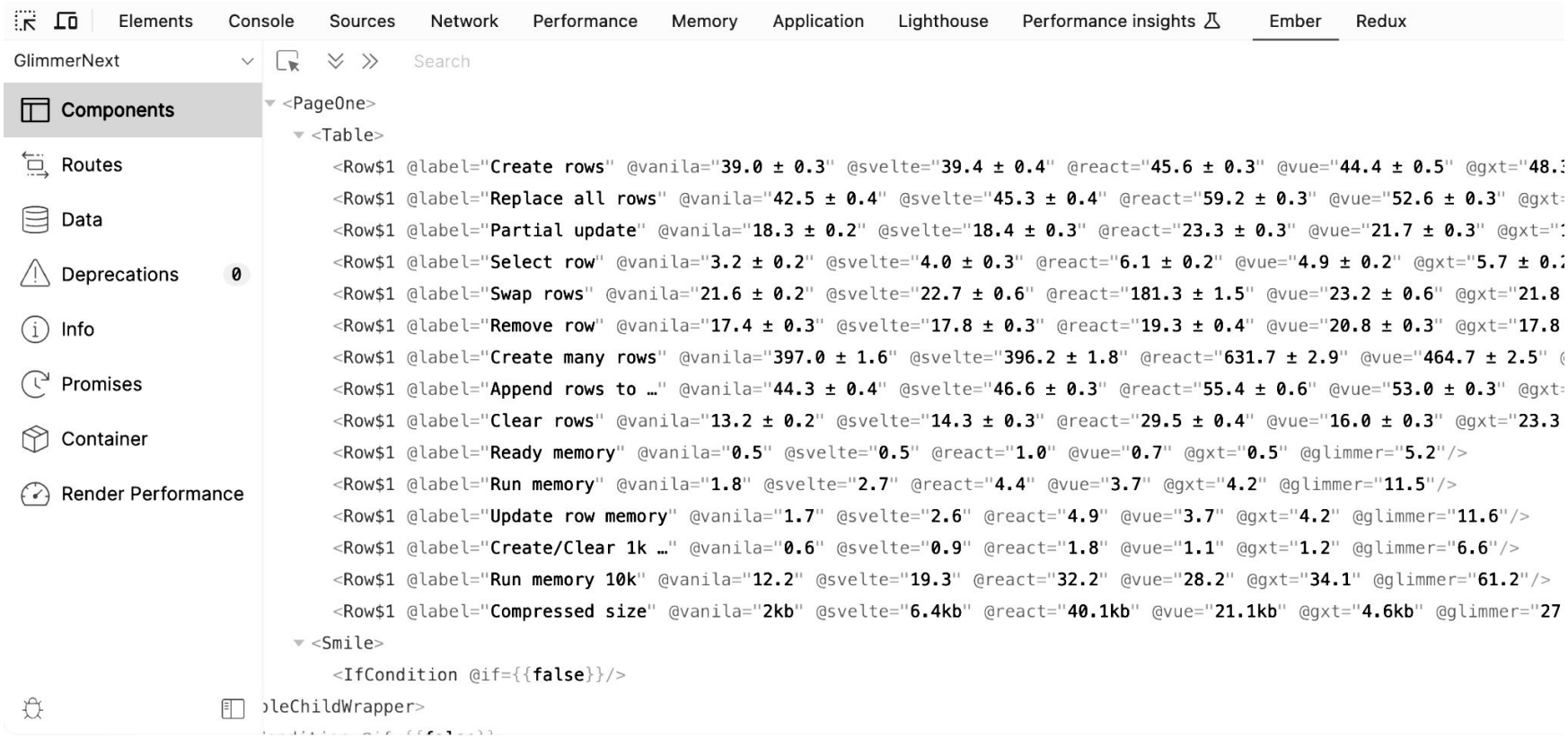
Так как логику рендеринга мы писали с нуля, то вариантов у нас было не так уж и много:

- Создавать своё расширение для дебага
- Мимикрировать на уровне API под любое существующее расширение

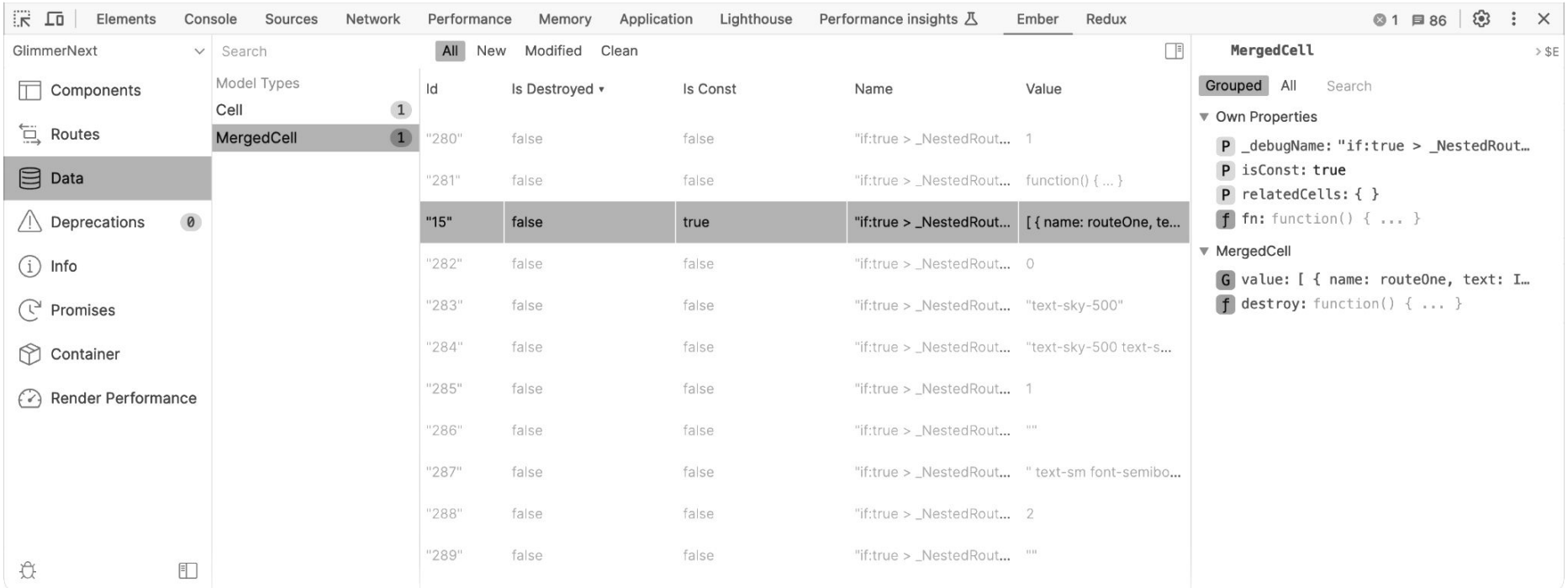
DevTools: мимикрирование

Я выбрал этот вариант, потому что он самый простой.

DevTools: Ember Inspector - component tree



DevTools: Ember Inspector - reactive primitives



Асинхронные деструкторы элементов

Асинхронные деструкторы ктов

Асинхронные деструкторы элементов

подход к анимации **удаления** в современных веб фреймворках:

- скопировал
- анимировал копию

react-transition-group / src / TransitionGroup.js

Code | Blame | 241 lines (197 loc) · 7.1 KB

```
22    class TransitionGroup extends React.Component {
178     render() {
202
203           // You may need to apply reactive updates to a child as it is leaving.
204           // The normal React way to do it won't work since the child will have
205           // already been removed. In case you need this behavior you can provide
206           // a childFactory function to wrap every child, even the ones that are
207           // leaving.
208           childrenToRender.push(React.cloneElement(
209             factoryChild,
210             {
211               key,
212               ref,
213             },
214           ));
215         }
216       }
217
```

Асинхронные деструкторы элементов

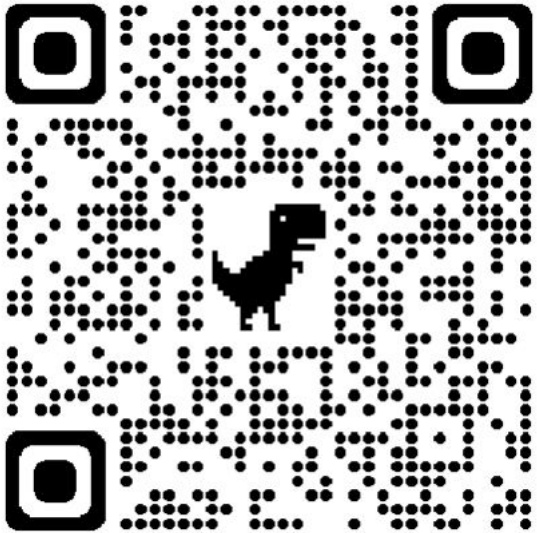
Напомню основные недостатки - из-за того что жизненный цикл компонентов (и соответственно, DOM элементов которые их составляют) никак не привязан к анимациям большинство библиотек делают специальный хак, а именно - цепляются к коллбеку “удаления” компонента, делают копию его элементов (с какой-нибудь позицией absolute или fixed) и анимируют эту копию, в то время когда движок удаляет сам компонент и все его DOM ноды.

Асинхронные деструкторы элементов

Я подумал - почему бы не попробовать сделать поддержку **асинхронных** деструкторов, когда эффект, применяемый на элементе может **возвращать промис**, и движок будет его **ждать** перед удалением элемента.

Асинхронные деструкторы элементов

Это получилось сделать для основных flow-control элементов, а именно: if и each.



Асинхронные деструкторы элементов

Это получилось сделать для основных flow-control элементов, а именно: if и each.

```
modifier = (element: HTMLDivElement) => {
  const destructor = async () => {
    const rect = element.getBoundingClientRect();
    element.style.position = 'absolute';
    element.style.top = `${rect.top + document.documentElement.scrollTop}px`;
    element.style.left = `${rect.left}px`;
    element.style.width = `${rect.width}px`;
    element.style.height = `${rect.height}px`;
    element.style.backgroundColor = 'blue';
    element.style.transition = 'all 1.4s ease';
    element.style.transform = 'scale(0)';
    await new Promise((resolve) => setTimeout(resolve, 1400));
  };
  return destructor;
};
```

Тесты в браузере

Много кто использует JEST для интеграционных и юнит тестов компонентов, что на мой взгляд не самый оптимальный подход

Тесты должны работать в браузере

Много кто использует JEST для интеграционных и юнит тестов компонентов, что на мой взгляд не самый оптимальный подход

- трудно дебажить
- тестируется не рендеринг компонента, а то как компонент должен рендериться по мнению библиотеки, мокающей DOM

Тесты должны работать в браузере

Есть отличная альтернатива - qUnit, много лет он в основном использовался в Ember сообществе, но теперь его переписали на es-модули и можно использовать с любым фреймворком.

Запускается в браузере - прекрасно дружит с playwright

Тесты должны работать в браузере

GlimmerNext

☐ Hide passed tests ☐ Check for Globals ☐ No try-catch Filter: Go Module: All modules

QUnit 2.20.0; Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36

155 tests completed in 2411 milliseconds, with 0 failed, 2 skipped, and 0 todo.
362 assertions of 362 passed, 0 failed.

1. **Integration | Component | Button: renders default slot (1)** Rerun — 12 ms
   1. Element [data-test-button] has text "world" — @ 1 ms

Source: at Object.<anonymous> (https://g-next.netlify.app/assets/nested-AAWwsDnF.js:5901:16)

2. **Integration | Component | Button: assept onClick function (1)** Rerun — 6 ms
3. **Integration | Component | Button: has default type (1)** Rerun — 10 ms
4. **Integration | Component | Button: allow type overriding (1)** Rerun — 9 ms
5. **Integration | Interal | @arguments: works as element attributes (1)** Rerun — 9 ms
6. **Integration | Interal | @arguments: support attr-like args (2)** Rerun — 10 ms
7. **Integration | Interal | @arguments: support args in subExpression control (1)** Rerun — 10 ms
8. **Integration | Interal | @arguments: support args in control expressions (1)** Rerun — 10 ms
9. **Integration | Interal | @arguments: support strings as arguments for textContent (1)** Rerun — 9 ms

```
import 'qunit/qunit/qunit.css';
import 'qunit-theme-ember/qunit.css';

import * as QUnit from 'qunit';
import { setup } from 'qunit-dom';

setup(QUnit.assert, {
  getRootElement() {
    return document.getElementById('app-testing')!;
  },
});

import.meta.glob('./unit/**/*-test.{gts,ts,js,gjs}', { eager: true });
```

```
import { module, test } from 'qunit';
import { render } from '@lifeart/gxt/test-utils';

module('Integration | InternalHelper | array', function () {
  test('it could be used as source for list', async function (assert) {
    await render(
      <template>
        <ul>
          {{#each (array 1 2 3) as |item|}}
            <li>{{item}}</li>
          {{/each}}
        </ul>
      </template>,
    );
    assert.dom('li').exists({ count: 3 });
  });
});
```

Подсветка синтаксиса на GitHub

Мало кто знает, но добавить подсветку синтаксиса не так сложно.

Github использует textmate грамматику (кучку регулярных выражений), всё это конфигурируется в этом репозитории - github.com/github-linguist/linguist

Подсветка синтаксиса на GitHub

1. **Language Extension Introduction**
   - Adding a new language with its extension being widely used across hundreds of repositories on GitHub.com.

2. **Search Results for Extension**
   - For `.gts` extension: Over 2,000 files found, excluding forks.

github.com/github-linguist/linguist/pull/6680

Подсветка синтаксиса на GitHub

Real-World Usage Samples Provided

- Sample source files for the `.gts` extension:
  - `class.gts`
  - `template-only.gts`

Sample Licenses for Sources

- For `class.gts`: MIT License
- For `template-only.gjs`: MIT License

Color Addition for the Language

- Hex value: `#3178c6`
- Rationale: Chosen for its association with the TypeScript color, which is also used by TypeScript and TSX.

Updated Heuristics for Language Distinction

- Heuristics have been updated to distinguish the new language from others using the same extension.

github.com/github-linguist/linguist/pull/6680

Подсветка синтаксиса на GitHub

glimmer-next / src / components / pages / todomvc / **page.gts**

lifeart [bugifx] yield stability in if for single text node (#62)

Code | Blame | 6 lines (5 loc) · 194 Bytes

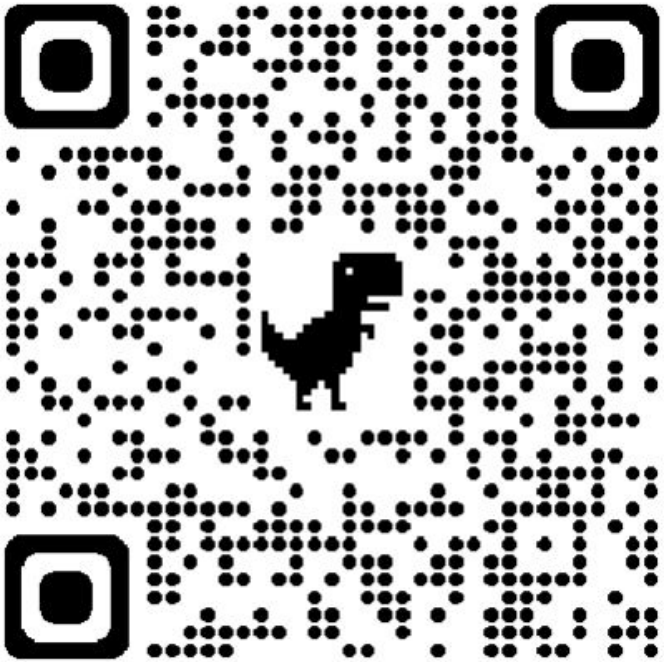
```
1  import { Component } from '@lifeart/gxt';
2  import { TodoList } from './TodoList.gts';
3
4  export default class Page extends Component {
5    <template><TodoList @todos={{@model.model}} /></template>
6  }
```

Спасибо за внимание

github.com/lifeart/glimmer-next

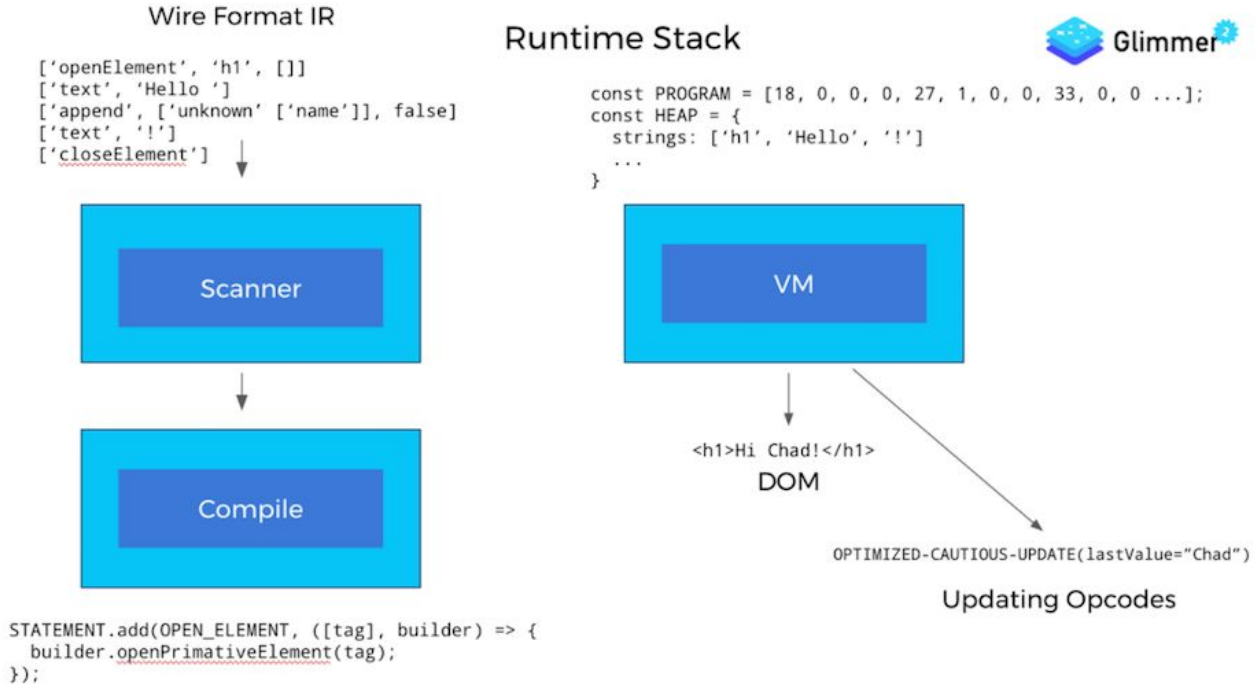
tg: **ilifeart**

x: **vaier**



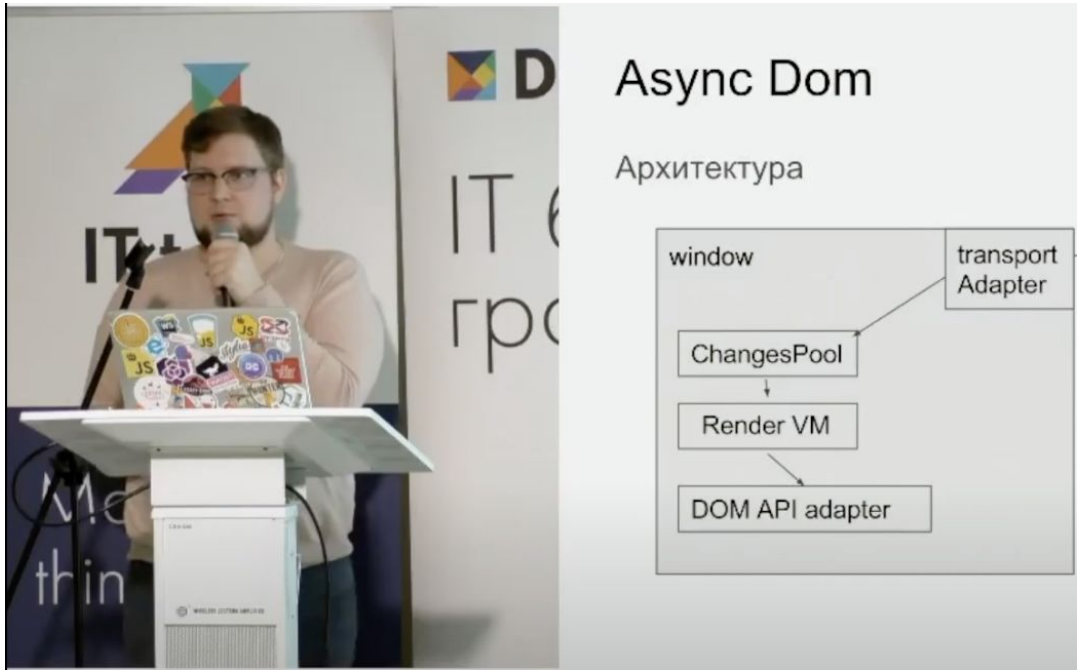
NAME	VANILLAJS	SVELTE-V5	REACT-V18	VUE-V3	GLIMMER-NEXT	GLIMMER-2
Create rows	39.0 (-23.8%)	39.4 (-22.6%)	45.6 (-5.9%)	44.4 (-8.8%)	**48.3**	69.1 (+30.1%)
Replace all rows	42.5 (-39.3%)	45.3 (-30.7%)	59.2	52.6 (-12.5%)	**59.2**	86.4 (+31.5%)
Partial update	18.3 (-1.1%)	18.4 (-0.5%)	23.3 (+20.6%)	21.7 (+14.7%)	**18.5**	26.0 (+28.8%)
Select row	3.2 (-78.1%)	4.0 (-42.5%)	6.1 (+6.6%)	4.9 (-16.3%)	**5.7**	22.5 (+74.7%)
Swap rows	21.6 (-0.9%)	22.7 (+4.0%)	181.3 (+88.0%)	23.2 (+6.0%)	**21.8**	30.7 (+29.0%)
Remove row	17.4 (-2.3%)	17.8	19.3 (+7.8%)	20.8 (+14.4%)	**17.8**	28.4 (+37.3%)
Create many rows	397.0 (-29.7%)	396.2 (-30.0%)	631.7 (+18.5%)	464.7 (-10.8%)	**514.9**	636.9 (+19.2%)
Append rows to large table	44.3 (-29.3%)	46.6 (-23.0%)	55.4 (-3.4%)	53.0 (-8.1%)	**57.3**	84.8 (+32.4%)
Clear rows	13.2 (-76.5%)	14.3 (-62.9%)	29.5 (+21.0%)	16.0 (-45.6%)	**23.3**	30.7 (+24.1%)
Ready memory	0.5	0.5	1.0 (+50.0%)	0.7 (+28.6%)	**0.5**	5.2 (+90.4%)
Run memory	1.8 (-133.3%)	2.7 (-55.6%)	4.4 (+4.5%)	3.7 (-13.5%)	**4.2**	11.5 (+63.5%)
Update row memory	1.7 (-147.1%)	2.6 (-61.5%)	4.9 (+14.3%)	3.7 (-13.5%)	**4.2**	11.6 (+63.8%)
Create/Clear 1k rows memory	0.6 (-100.0%)	0.9 (-33.3%)	1.8 (+33.3%)	1.1 (-9.1%)	**1.2**	6.6 (+81.8%)
Run memory 10k	12.2 (-179.5%)	19.3 (-76.7%)	32.2 (-5.9%)	28.2 (-20.9%)	**34.1**	61.2 (+44.3%)
Compressed size	2kb (-130.0%)	6.4kb (+28.1%)	40.1kb (+88.5%)	21.1kb (+78.2%)	**4.6kb**	27.9kb (+83.5%)

Rendering for Ember.js, Part 2



https://www.linkedin.com/blog/engineering/optimization/glimmer-blazing-fast-rendering-for-ember-js-part-2

AsyncDom


Async Dom
Архитектура
window
transport
Adapter
ChangesPool
Render VM
DOM API adapter