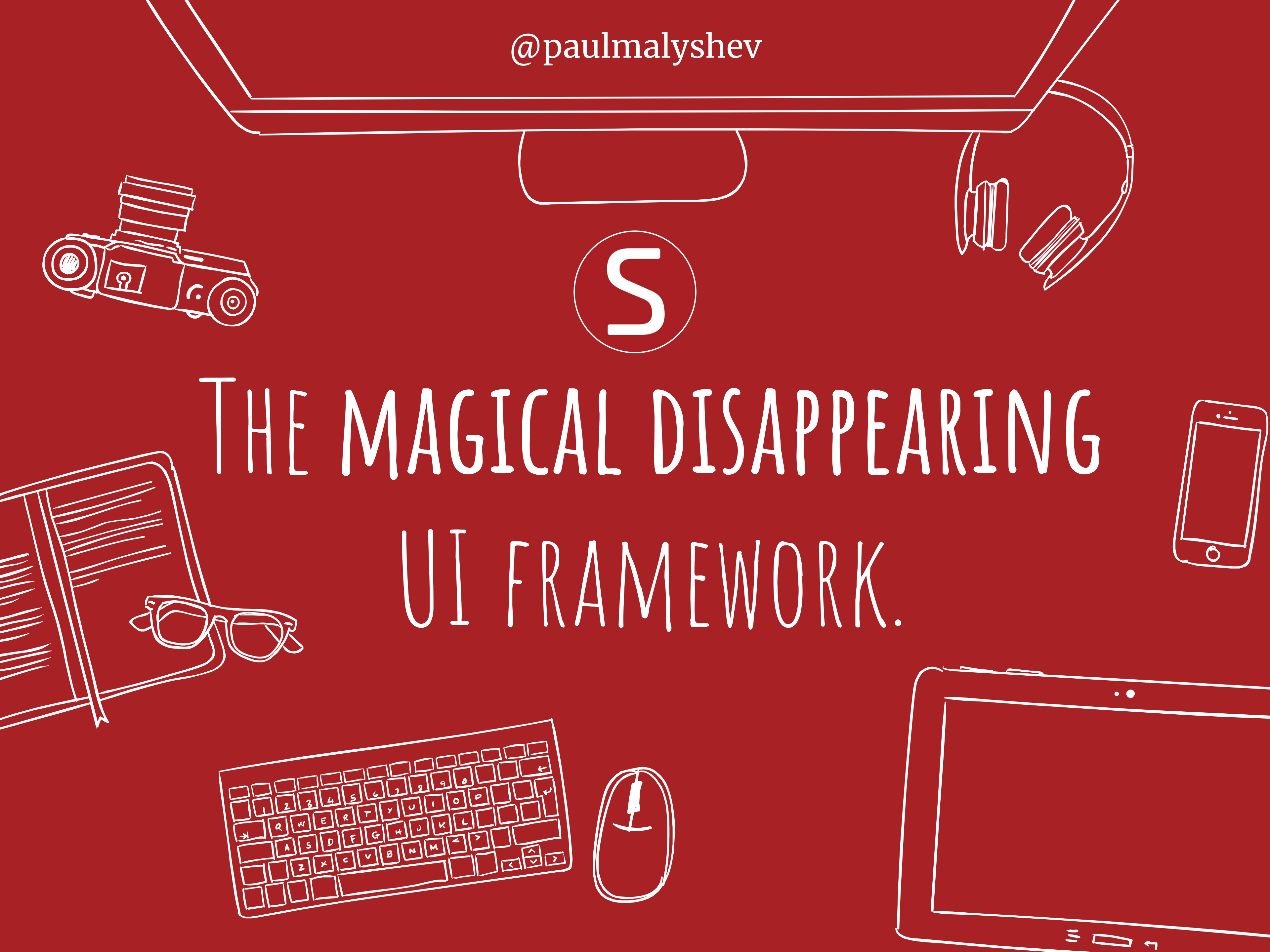
@paulmalyshev

The magical disappearing UI framework.



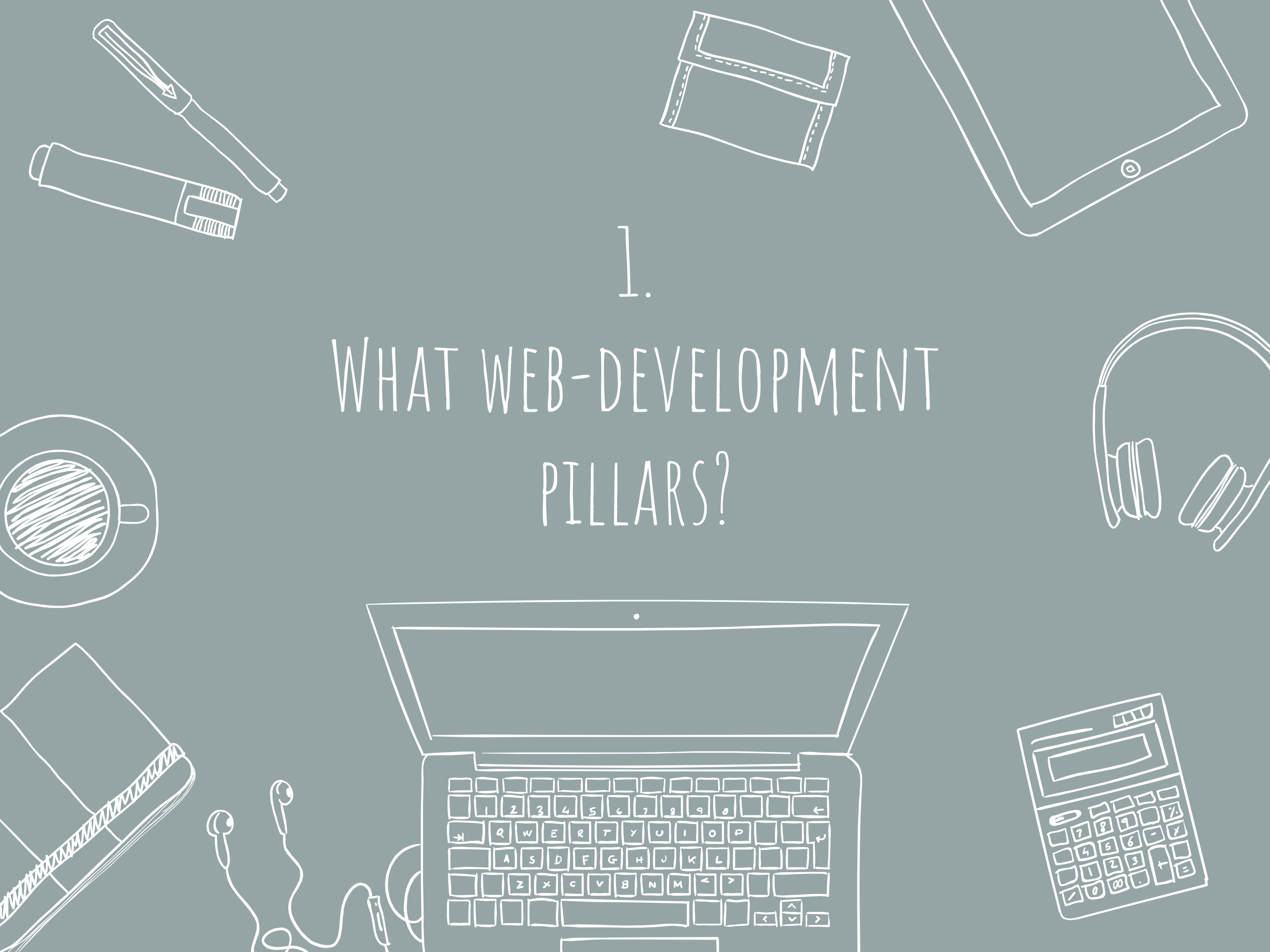
Real street magic!

Agenda

- ✓ **What** is framework without the framework?
- ✓ **Why** should we use it?
- ✓ **How** can it improve our apps?

"You can't write serious applications in vanilla JavaScript without hitting a complexity wall. But a compiler can do it for you."

– *Rich Harris, 2016*



1.
What web-development pillars?

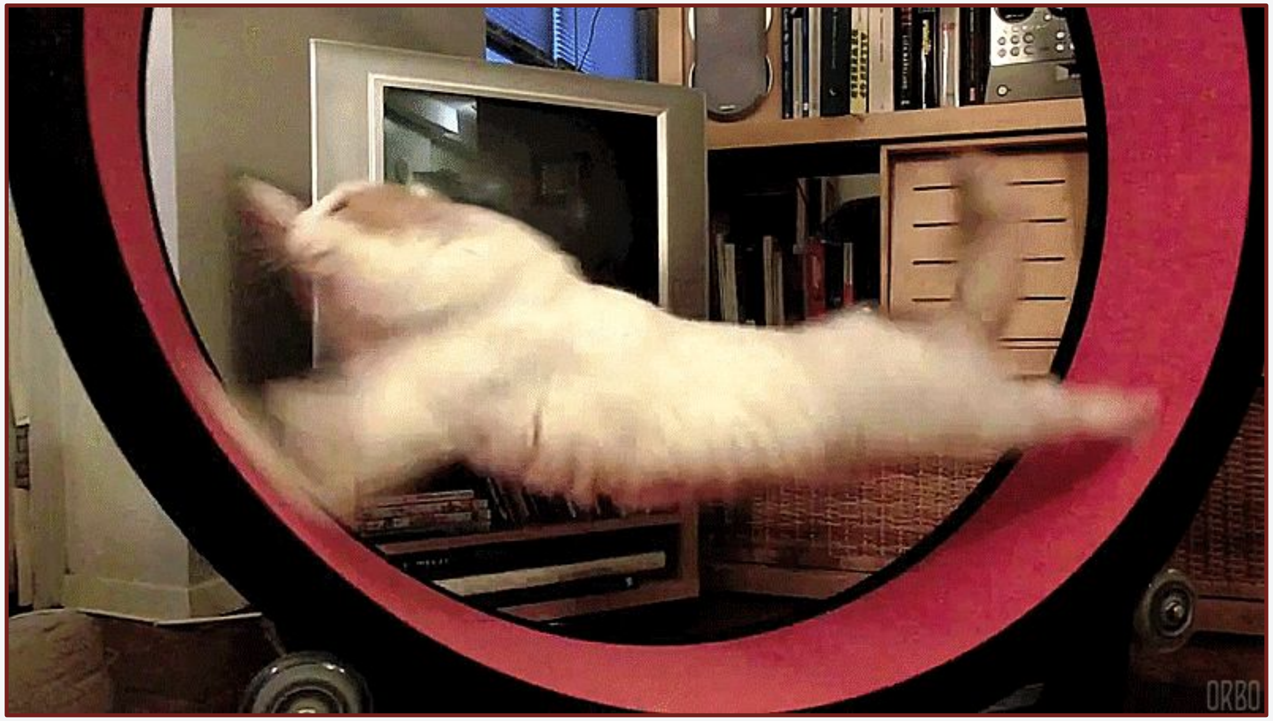
Size

We're shipping too much code to our users.

42Kb gz

143Kb gz

23Kb gz



Performance

We pay an upfront cost of a hefty runtime because of abstraction between your app and the browser.

Effectiveness

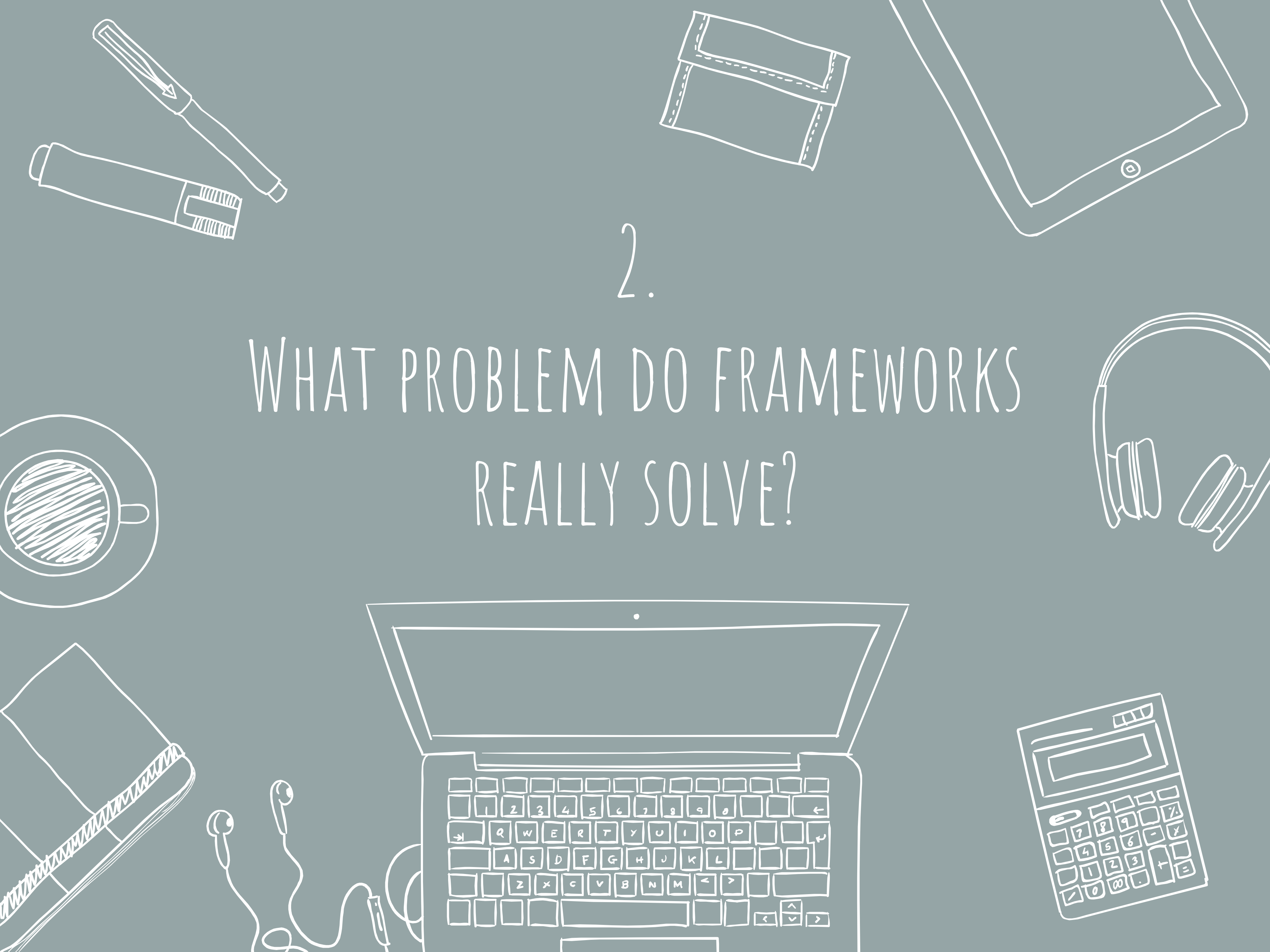
Too many approaches, “best practices” and “*your-framework*”-ways.

Other things what's matter

- ✖ Interoperability.
- ✖ Code splitting & tree shaking.
- ✖ Feature cost.

Perhaps we need to rethink the whole thing?

Let's make javascript great again!



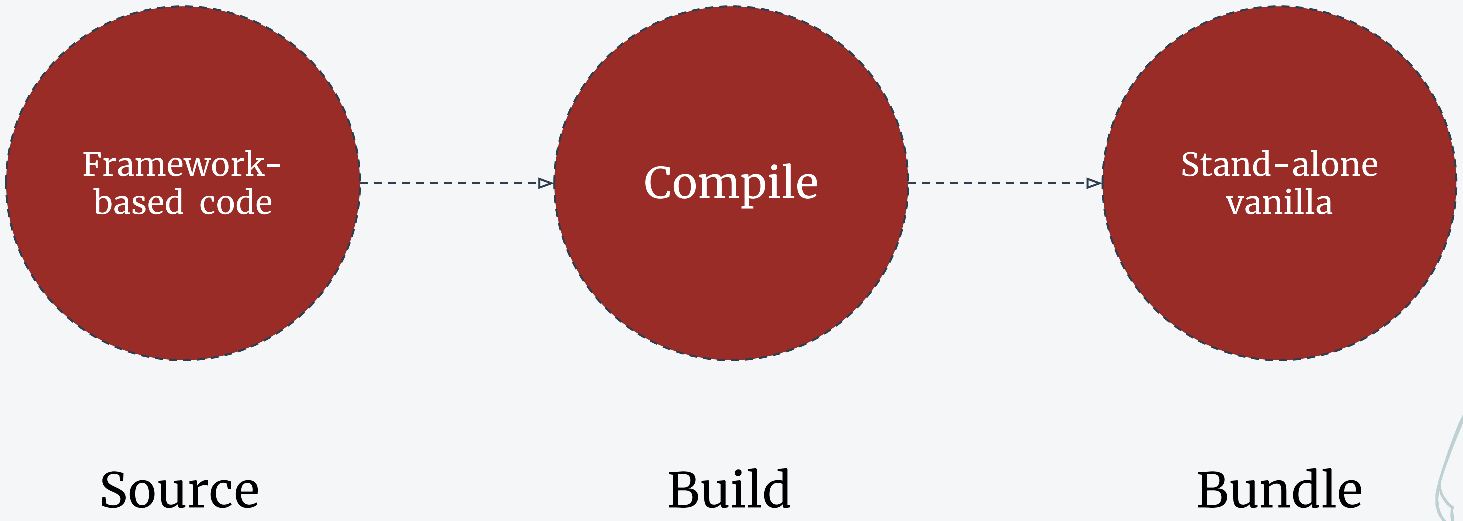
2.

What problem do frameworks really solve?

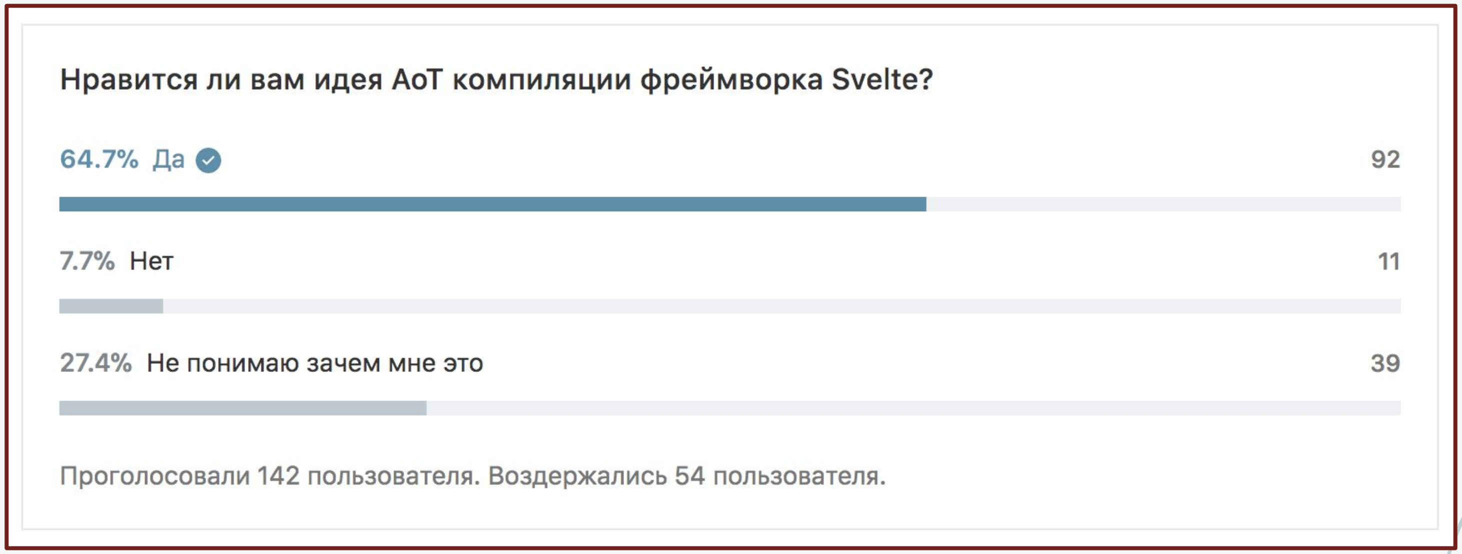
"Wait, this new framework has a runtime? Ugh. Thanks, I'll pass."

- *Front-end developer, 2018*

Development process



Is AoT-compilation awesome?



Нравится ли вам идея AoT компиляции фреймворка Svelte?

64.7% Да ✓ 92

7.7% Нет 11

27.4% Не понимаю зачем мне это 39

Проголосовали 142 пользователя. Воздержались 54 пользователя.

Source: ***https://habrahabr.ru/post/345028/***

Who

svelte

The magical disappearing UI framework.

STENCIL

The magical, reusable web component compiler.

ANGULAR

The Angular AoT compiler converts Angular code into efficient JavaScript code.

3.

Introducing Svelte

svelte

- ✓ is build-time UI framework
- ✓ is compile-time static analyzer
- ✓ is ahead-of-time (AoT) compiler

Author: Rich Harris (Ractive, Rollup, Roadtrip)

COMMON FEATURES

Components & composition

Templating & directives

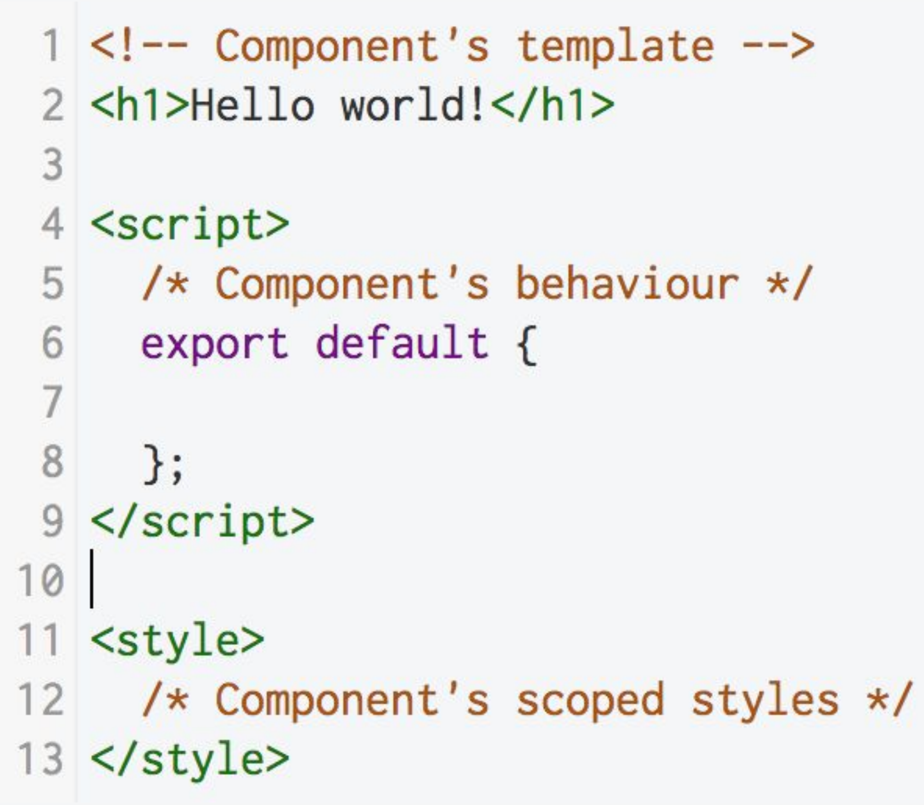
Data & computed

Observers & Hooks

Proxy & custom events

Methods & helpers

Components & composition

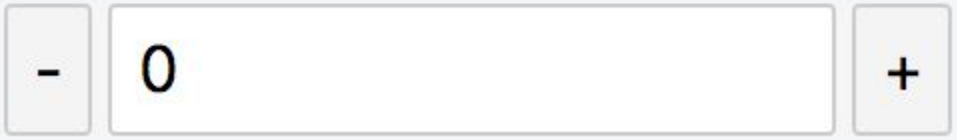


```
<!-- Component's template -->
<h1>Hello world!</h1>

<script>
  /* Component's behaviour */
  export default {

  };
</script>

<style>
  /* Component's scoped styles */
</style>
```

Component's definition (SFC)

Components & composition

```
<button on:click="set({ count: count - 1 })">-</button>
<input bind:value="count" readonly />
<button on:click="set({ count: count + 1 })">+</button>
```



Counter component

Components & composition

```html
<!-- Child.html -->
<div class="child">
  Hello
  <slot><!-- content is injected here --></slot>
</div>

<!-- Parent.html -->
<Child>
  <b>world!</b>
</Child>

<script>
  import Child from './Child.html';

  export default {
    components: { Child }
  };
</script>
```

Nested components & composing with slots

Components & composition

```
<:Component { foo ? Child1 : Child2 }/>

<:Self/>
```

```
<:Window />
```

```
<:Head>
  <title>My blog</title>
</:Head>
```

Special components

Templating & directives

```
<h1 style="color: {{color}};">{{color}}</h1>
<p hidden="{{hideParagraph}}">You can hide this paragraph.</p>

{{#if loggedIn}}
  <a href='/logout'>log out</a>
{{else}}
  <a href='/login'>log in</a>
{{/if}}

<ul>
  {{#each list as item}}
    <li>{{item.title}}</li>
  {{/each}}
</ul>
```

Mustache-like syntax

Templating & directives

```
<!-- Event handlers -->
<button on:click="set({ count: count + 1 })">+1</button>

<!-- Two-way binding -->
<input bind:value="count" />

<!-- Refs (like Vue) -->
<canvas ref:canvas></canvas>

<!-- Transitions -->
<div transition:fly="{y:20}">hello!</div>
```

Only built-in directives

Data & computed

```
<strong>{{hours}}:{{minutes}}:{{seconds}}</strong>

<script>
  export default {
    data: () => ({
      time: new Date()
    }),
    computed: {
      hours: time => time.getHours(),
      minutes: time => time.getMinutes(),
      seconds: time => time.getSeconds()
    },
    oncreate() {
      const interval = setInterval(() => this.set({time: new Date()}));
      this.on('destroy', () => clearInterval(interval));
    }
  };
</script>
```

Clock component

Observers & Hooks

```
<Widget ref:widget/>
<script>
  import Widget from './Widget.html';

  export default {
    components: { Widget },
    data: () => ({
      foo: 0
    }),
    oncreate() {
        const fooObserver = this.observe('foo', (curr, prev) => {});
        const barObserver = this.refs.widget.observe('bar', (curr, prev) => {});

        this.on('destroy', () => fooObserver.cancel(), barObserver.cancel());
    },
    ondestroy() {
      /* Do something else */
    }
  };
</script>
```

Observe data, computed props & even nested components

Proxy & custom events

```
<p>Select a category:</p>
{{#each categories as category}}
  <button on:click="fire('select', { category })">
    {{category}}
  </button>
{{/each}}

<script>
  import {categories} from './categories.js';

  export default {
    data: () => ({
      categories
    })
  };
</script>
```

CategoryChooser component

Methods & helpers

```
<p>{{ formatDate(Date.now()) }}</p>

<CategoryChooser on:select="doSomething(event.category)"/>

<script>
  import CategoryChooser from './CategoryChooser.html';

  export default {
    components: { CategoryChooser },
    methods: {
      doSomething(category) {
        console.log('Choosed category: ', category);
      }
    },
    helpers: {
      formatDate: date => new Date(date).toDateString()
    }
  };
</script>
```

Parent component

ADDITIONAL FEATURES

SSR & Hydration

Built-in state management

Custom elements

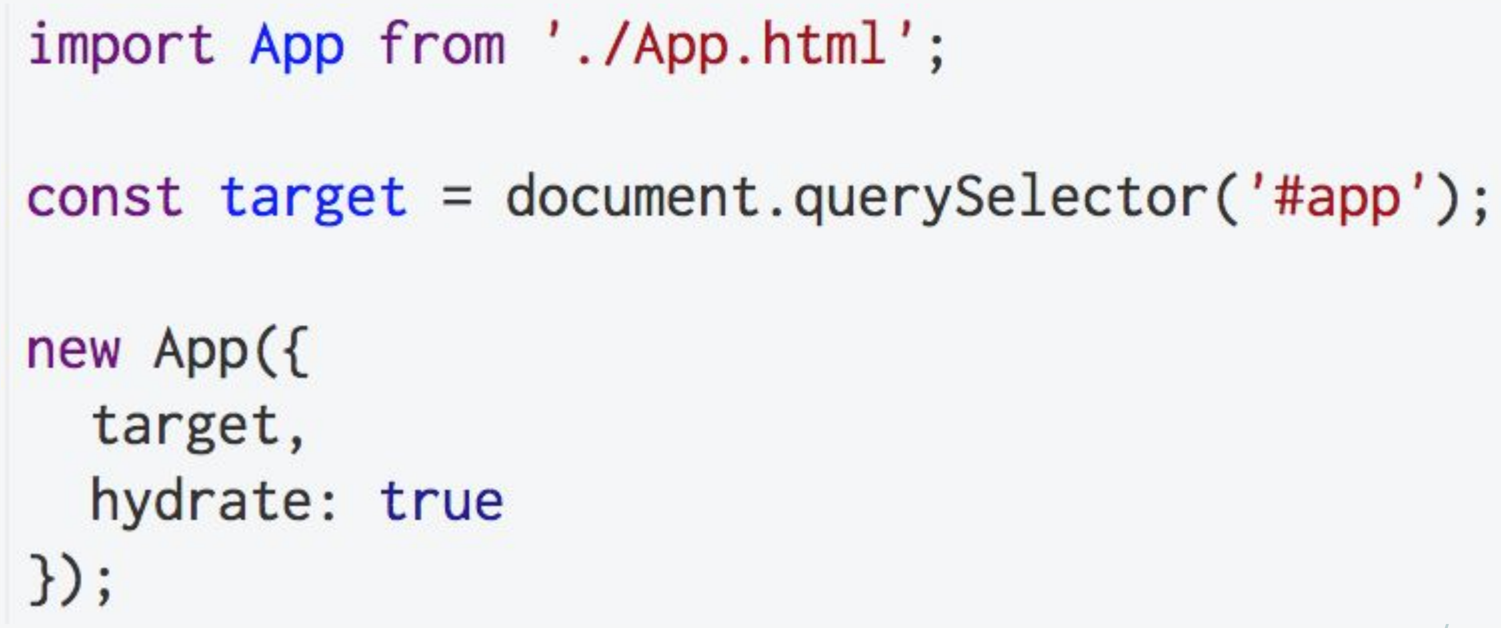
Sapper
(Svelte **app** mak**er)**

SSR & Hydration

```
require('svelte/ssr/register');
const App = require('./App.html');

const data = { foo: 'bar' };
const { html, css, head } = App.render(data);
```

Server-side rendering

SSR & Hydration



```
import App from './App.html';

const target = document.querySelector('#app');

new App({
  target,
  hydrate: true
});
```

Custom elements

```
<!-- HelloWorld.html -->
<h1>Hello {{name}}!</h1>

<script>
  export default {
    tag: 'hello-world'
  };
</script>
```

```
/* main.js */
import './HelloWorld.html';
document.body.innerHTML = `<hello-world name="world"/>`;

const el = document.querySelector('hello-world');
el.name = 'everybody';
```

Built-in state management

```
// main.js
import App from './App.html';
import { Store } from 'svelte/store.js';

const store = new Store({
  name: 'world'
});

const app = new App({
  target: document.querySelector('main'),
  store
});
```

Create store

Built-in state management

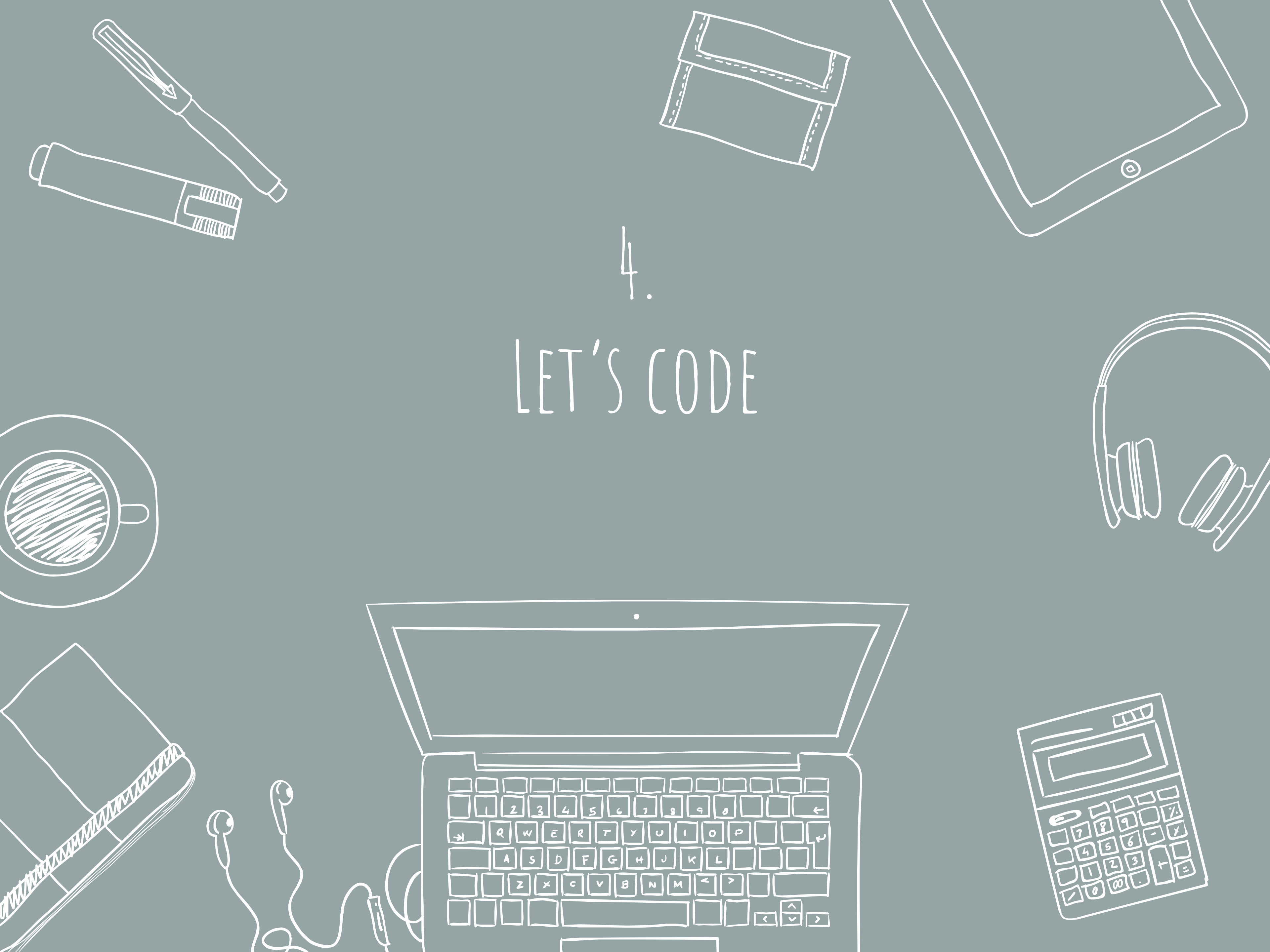
```
class TodoStore extends Store {
/* ... */
}
```

Store extending & store per hierarchy

Built-in state management

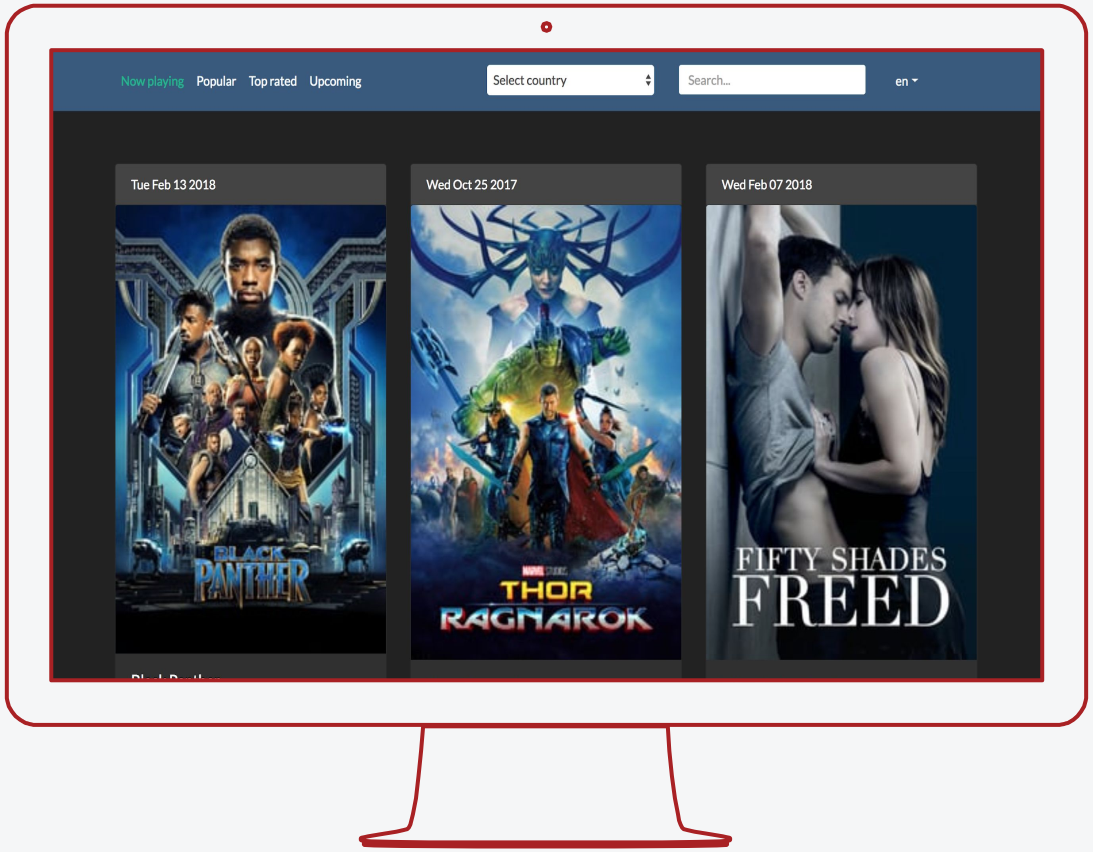
```
<button on:click="store.set({ muted: !$muted })">
  {{ $muted ? 'Unmute' : 'Mute' }}
</button>

<script>
  export default {
    oncreate() {
      this.store.observe('muted', muted => {
        // ...
      });
    }
  };
</script>
```

Store using in components



4.

Let's code



Hello Movies

A list of the movies with filters based on TMDb API.

6,4 Kb GZ

Whole bundle.

172 LOC

36 JS LOC

3,6Kb gz

(React: 300Kb / Vue: 80Kb / Vanilla: 11Kb)

ToDo MVC

40Kb gz

(React/Redux: 211Kb / Angular: 575Kb)

RealWorld

30Kb gz

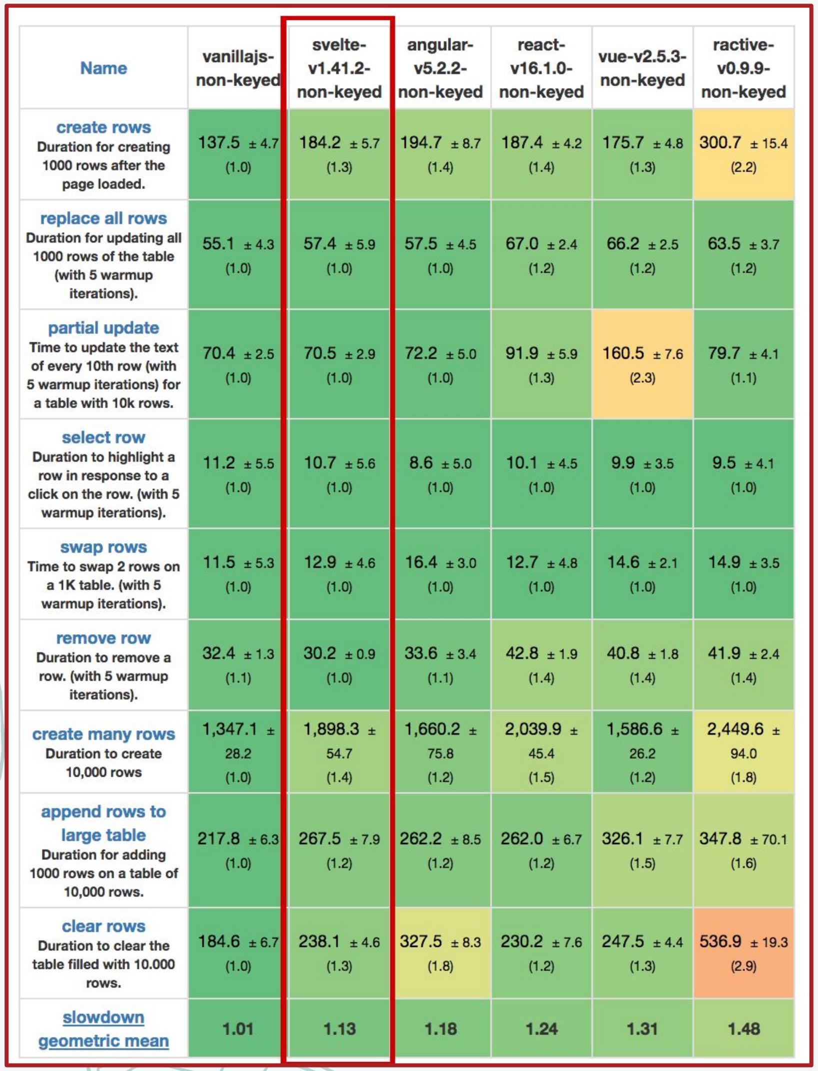
(React: 140Kb / Vue: 101Kb)

HN clone

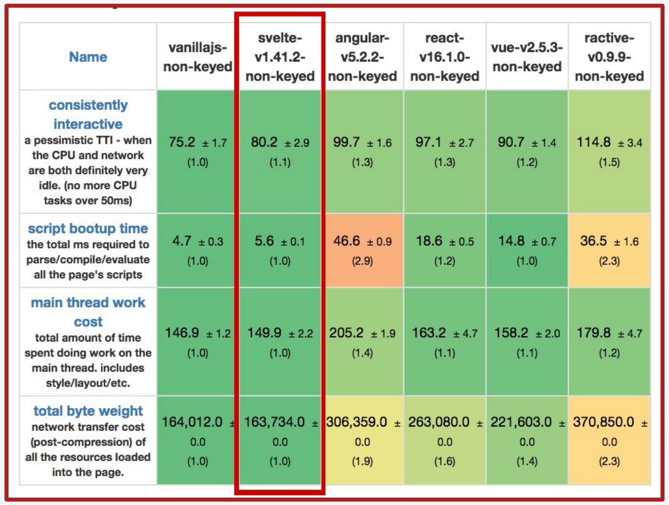
Benchmarks

Source: https://github.com/krausest/js-framework-benchmark

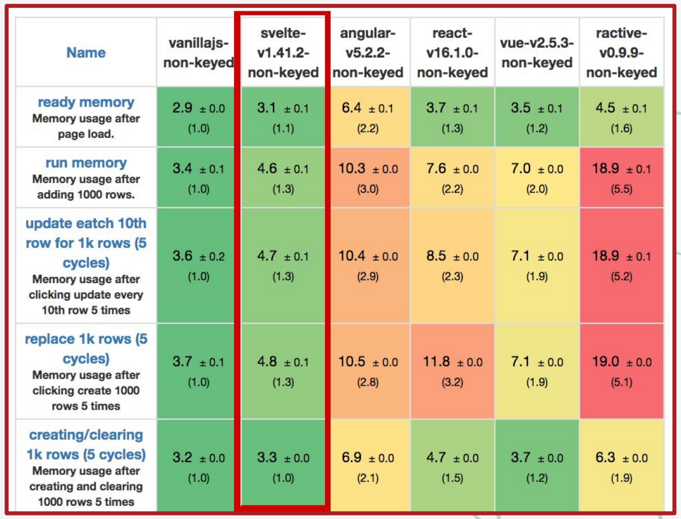
Js frameworks benchmark



Name	vanillajs-non-keyed	svelte-v1.41.2-non-keyed	angular-v5.2.2-non-keyed	react-v16.1.0-non-keyed	vue-v2.5.3-non-keyed	ractive-v0.9.9-non-keyed
create rows Duration for creating 1000 rows after the page loaded.	137.5 ± 4.7 (1.0)	184.2 ± 5.7 (1.3)	194.7 ± 8.7 (1.4)	187.4 ± 4.2 (1.4)	175.7 ± 4.8 (1.3)	300.7 ± 15.4 (2.2)
replace all rows Duration for updating all 1000 rows of the table (with 5 warmup iterations).	55.1 ± 4.3 (1.0)	57.4 ± 5.9 (1.0)	57.5 ± 4.5 (1.0)	67.0 ± 2.4 (1.2)	66.2 ± 2.5 (1.2)	63.5 ± 3.7 (1.2)
partial update Time to update the text of every 10th row (with 5 warmup iterations) for a table with 10k rows.	70.4 ± 2.5 (1.0)	70.5 ± 2.9 (1.0)	72.2 ± 5.0 (1.0)	91.9 ± 5.9 (1.3)	160.5 ± 7.6 (2.3)	79.7 ± 4.1 (1.1)
select row Duration to highlight a row in response to a click on the row. (with 5 warmup iterations).	11.2 ± 5.5 (1.0)	10.7 ± 5.6 (1.0)	8.6 ± 5.0 (1.0)	10.1 ± 4.5 (1.0)	9.9 ± 3.5 (1.0)	9.5 ± 4.1 (1.0)
swap rows Time to swap 2 rows on a 1K table. (with 5 warmup iterations).	11.5 ± 5.3 (1.0)	12.9 ± 4.6 (1.0)	16.4 ± 3.0 (1.0)	12.7 ± 4.8 (1.0)	14.6 ± 2.1 (1.0)	14.9 ± 3.5 (1.0)
remove row Duration to remove a row. (with 5 warmup iterations).	32.4 ± 1.3 (1.1)	30.2 ± 0.9 (1.0)	33.6 ± 3.4 (1.1)	42.8 ± 1.9 (1.4)	40.8 ± 1.8 (1.4)	41.9 ± 2.4 (1.4)
create many rows Duration to create 10,000 rows	1,347.1 ± 28.2 (1.0)	1,898.3 ± 54.7 (1.4)	1,660.2 ± 75.8 (1.2)	2,039.9 ± 45.4 (1.5)	1,586.6 ± 26.2 (1.2)	2,449.6 ± 94.0 (1.8)
append rows to large table Duration for adding 1000 rows on a table of 10,000 rows.	217.8 ± 6.3 (1.0)	267.5 ± 7.9 (1.2)	262.2 ± 8.5 (1.2)	262.0 ± 6.7 (1.2)	326.1 ± 7.7 (1.5)	347.8 ± 70.1 (1.6)
clear rows Duration to clear the table filled with 10.000 rows.	184.6 ± 6.7 (1.0)	238.1 ± 4.6 (1.3)	327.5 ± 8.3 (1.8)	230.2 ± 7.6 (1.2)	247.5 ± 4.4 (1.3)	536.9 ± 19.3 (2.9)
slowdown geometric mean	1.01	1.13	1.18	1.24	1.31	1.48



Name	vanillajs-non-keyed	svelte-v1.41.2-non-keyed	angular-v5.2.2-non-keyed	react-v16.1.0-non-keyed	vue-v2.5.3-non-keyed	ractive-v0.9.9-non-keyed
consistently interactive a pessimistic TTI - when the CPU and network are both definitely very idle. (no more CPU tasks over 50ms)	75.2 ± 1.7 (1.0)	80.2 ± 2.9 (1.1)	99.7 ± 1.6 (1.3)	97.1 ± 2.7 (1.3)	90.7 ± 1.4 (1.2)	114.8 ± 3.4 (1.5)
script bootup time the total ms required to parse/compile/evaluate all the page's scripts	4.7 ± 0.3 (1.0)	5.6 ± 0.1 (1.0)	46.6 ± 0.9 (2.9)	18.6 ± 0.5 (1.2)	14.8 ± 0.7 (1.0)	36.5 ± 1.6 (2.3)
main thread work cost total amount of time spent doing work on the main thread. includes style/layout/etc.	146.9 ± 1.2 (1.0)	149.9 ± 2.2 (1.0)	205.2 ± 1.9 (1.4)	163.2 ± 4.7 (1.1)	158.2 ± 2.0 (1.1)	179.8 ± 4.7 (1.2)
total byte weight network transfer cost (post-compression) of all the resources loaded into the page.	164,012.0 ± 0.0 (1.0)	163,734.0 ± 0.0 (1.0)	306,359.0 ± 0.0 (1.9)	263,080.0 ± 0.0 (1.6)	221,603.0 ± 0.0 (1.4)	370,850.0 ± 0.0 (2.3)



Name	vanillajs-non-keyed	svelte-v1.41.2-non-keyed	angular-v5.2.2-non-keyed	react-v16.1.0-non-keyed	vue-v2.5.3-non-keyed	ractive-v0.9.9-non-keyed
ready memory Memory usage after page load.	2.9 ± 0.0 (1.0)	3.1 ± 0.1 (1.1)	6.4 ± 0.1 (2.2)	3.7 ± 0.1 (1.3)	3.5 ± 0.1 (1.2)	4.5 ± 0.1 (1.6)
run memory Memory usage after adding 1000 rows.	3.4 ± 0.1 (1.0)	4.6 ± 0.1 (1.3)	10.3 ± 0.0 (3.0)	7.6 ± 0.0 (2.2)	7.0 ± 0.0 (2.0)	18.9 ± 0.1 (5.5)
update eatch 10th row for 1k rows (5 cycles) Memory usage after clicking update every 10th row 5 times	3.6 ± 0.2 (1.0)	4.7 ± 0.1 (1.3)	10.4 ± 0.0 (2.9)	8.5 ± 0.0 (2.3)	7.1 ± 0.0 (1.9)	18.9 ± 0.1 (5.2)
replace 1k rows (5 cycles) Memory usage after clicking create 1000 rows 5 times	3.7 ± 0.1 (1.0)	4.8 ± 0.1 (1.3)	10.5 ± 0.0 (2.8)	11.8 ± 0.0 (3.2)	7.1 ± 0.0 (1.9)	19.0 ± 0.0 (5.1)
creating/clearing 1k rows (5 cycles) Memory usage after creating and clearing 1000 rows 5 times	3.2 ± 0.0 (1.0)	3.3 ± 0.0 (1.0)	6.9 ± 0.0 (2.1)	4.7 ± 0.0 (1.5)	3.7 ± 0.0 (1.2)	6.3 ± 0.0 (1.9)

Summary

Pros

- ✓ Framework-less vanilla JS
- ✓ Tiny size
- ✓ Super fast
- ✓ Low learning curve
- ✓ Static analysis
- ✓ Micro-frontends

Cons

- ✖ Unproved approach
- ✖ One-man product
- ✖ No <script> tag
- ✖ Too strict
- ✖ Only composition

Thanks!

Any questions?

You can find me at:
@paulmalyshev
pavel@mustlab.ru

Component's overcost

```
function SvelteComponent(options) {
    init(this, options);
    this._state = assign({}, options.data);
    this._fragment = create_main_fragment(this._state, this);
    if (options.target) {
        this._fragment.c();
        this._fragment.m(options.target, options.anchor || null);
    }
}
assign(SvelteComponent.prototype, proto);
export default SvelteComponent;
```

Bonus