Get hyper-excited for web standards

1. Web Components
2. Custom Elements
3. HyperHTML
4. CSS Custom Properties
5. Shadow DOM

Web Components

```
<video controls src="video.mp4"></video>

<select>
  <option>Option 1</option>
  <option>Option 2</option>
  <option>Option 3</option>
</select>

<input type="date">
<input type="range">
```

```
<div class="dropdown">
  <button class="btn btn-secondary dropdown-toggle">
    Dropdown button
  </button>
  <div class="dropdown-menu">
    <a class="dropdown-item" href="#">Action</a>
    <a class="dropdown-item" href="#">Another action</a>
    <a class="dropdown-item" href="#">Something else here</a>
  </div>
</div>
```

... suspense

ANGULARJS

by Google

```
<tabs>
  <pane title="Hello">Hello</pane>
  <pane title="World">World</pane>
</tabs>
```

Dan Abramov
@dan_abramov

Following

React is “value UI”. Its core principle is that UI is a value, just like a string or an array. You can keep it in a variable, pass it around, use JavaScript control flow with it, and so on. That expressiveness is the point — not some diffing to avoid applying changes to the DOM.

Traduci il Tweet

14:56 - 24 nov 2018

111 Retweet **493** Mi piace

10 111 493

What if want

— Truly reusable components

— Interoperability

Web Components

1. Custom Elements
2. HTML Templates
3. Shadow DOM

Custom Elements

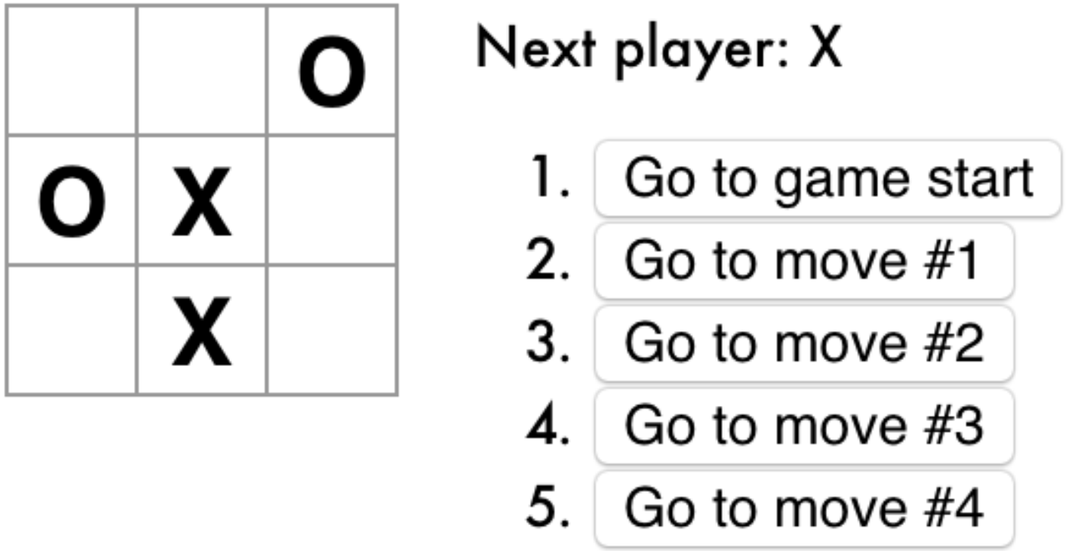
Create new HTML tags

```
<c-clock></c-clock>
```

Codepen

```
const template = document.createElement('template')
template.innerHTML = `
  <div>
      <h1>Hello, world!</h1>
      <h2>It is <span class="time"></span>.</h2>
  </div>
`

class Clock extends HTMLElement {
  constructor() {
    super();

    this.appendChild(template.content.cloneNode(true))
    this.timeEl = this.querySelector('.time')
  }

  connectedCallback() {
    this.token = window.setInterval(() => {
      this.timeEl.textContent = new Date().toLocaleTimeString();
    }, 1000)
  }

  disconnectedCallback() {
    if (this.token) window.clearInterval(this.token);
  }
}

customElements.define('c-clock', Clock)
```

Tic-tac-toe XO



in React

```
<c-square value="X"></c-square>
```

```
const template = document.createElement('template')
template.innerHTML = `<button class="square"></button>`

class Square extends HTMLElement {
  static get observedAttributes() {
    return ['value']
  }

  get value() { return this.getAttribute('value') }
  set value(val) { this.setAttribute('value', val) }

  constructor() {
    super();

    this.appendChild(template.content.cloneNode(true))
    this.btnEl = this.querySelector('.square')
  }

  attributeChangedCallback(attr, old, curr) {
    if (attr === 'value') this.btnEl.textContent = curr;
  }
}

customElements.define('c-square', Square)
```

Enter hyperHTML & HyperHTMLElement

```
class Square extends HyperHTMLElement {
  static get observedAttributes() {
    return ['value']
  }

  render() {
    return this.html`
      <button class="square">
        ${this.value}
      </button>
    `
  }
}

Square.define('c-square')
```

hyperHTML

```
function tick(render) {
  render`
    <div>
      <h1>Hello, world!</h1>
      <h2>It is ${new Date().toLocaleTimeString()}.</h2>
    </div>
  `;
}

setInterval(tick, 1000,
  hyperHTML.bind(document.body)
);
```

Codepen

hyperHTML

— Declarative and reactive templates
— Cross-platform IE9+
— Lightweight 4kB
— Based on ES6 template literals

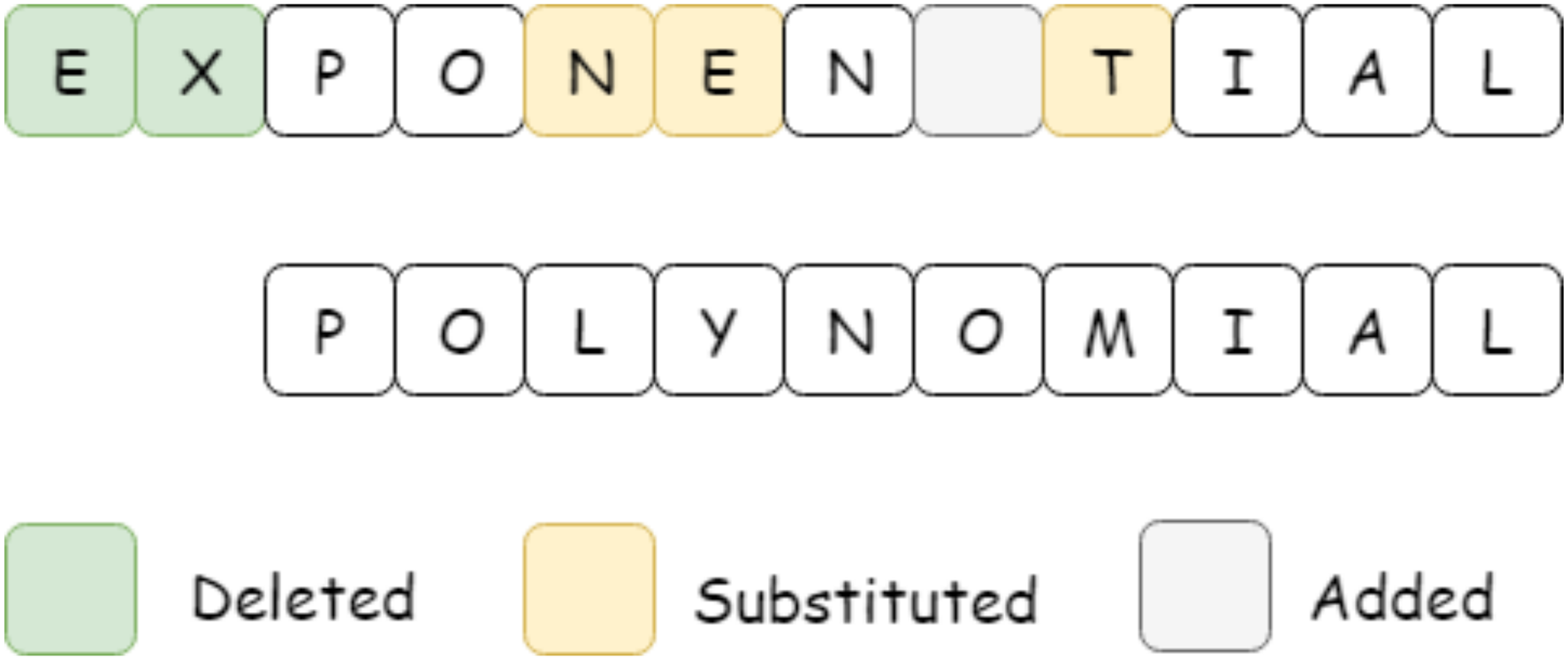
Template literals

```
function template(chunks, ...interpolations) {
  console.log(chunks);              // ['1 ', ' 3']
  console.log(interpolations);  // [2] or [4]
}

template`1 ${2} 3`;
template`1 ${4} 3`;
```

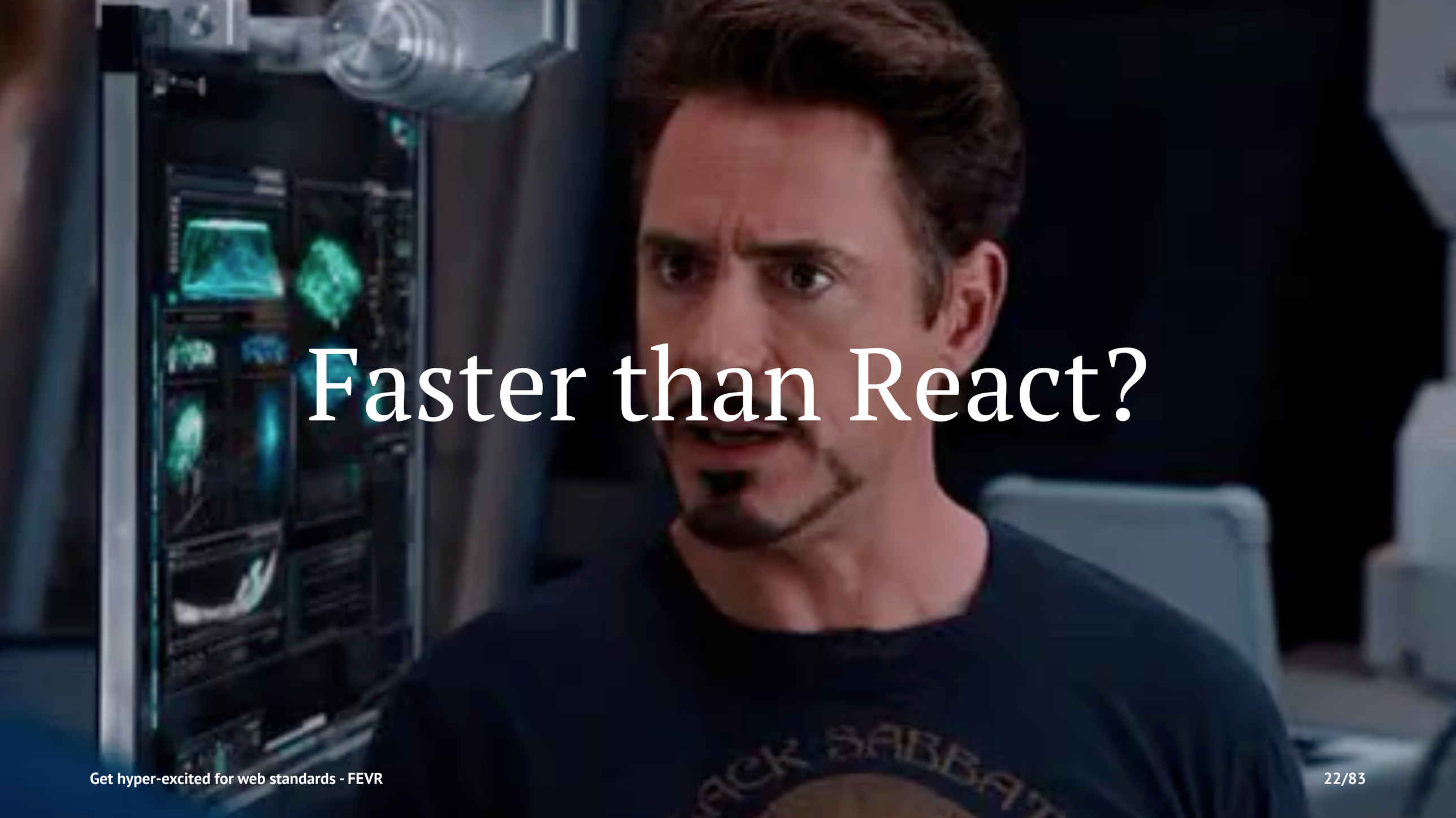
Efficient DOM

```
const bodyRender = hyperHTML.bind(document.body);
const names = [
  { name: 'First item' },
  { name: 'Second item' },
  { name: 'Third item' }
]

hyperHTML.bind(document.body)`
  <h1>${document.title}</h1>
  <ul>
    ${names.map(item => wire(item)`<li>${item.name}</li>`)}
  </ul>
`;
```

Levenshtein algorithm



domdiff based on petit-dom


Faster than React?

Declarative template

```
hyperHTML.bind(form)`
  <input
    class=${['one', 'more', 'class'].join(' ')}
    disabled=${!isAuthorized}
    oninput=${e => document.title = e.target.value}
    value=${defaultInputValue}
    placeholder=${'type something'}
    style=${{ color: 'red' }}
  >
`;
```

Not primitive types codepen-obj

```
customElements.define('h-welcome', class HyperWelcome extends HTMLElement {
    constructor(...args) {
      super(...args);
      this.html = hyperHTML.bind(this);
    }

    get user() { return this._user; }
    set user(value) {
      this._user = value;
      this.render();
    }

    render() { return this.html`<h1>Hello, ${this._user.name}</h1>`; }
  }
);

hyperHTML.bind(document.getElementById('root'))`
  <h-welcome user=${{ name: 'Sara' }} />
  <h-welcome user=${{ name: 'Cahal' }} />
`;
```

codepen-obj Codepen

HyperHTMLElement github

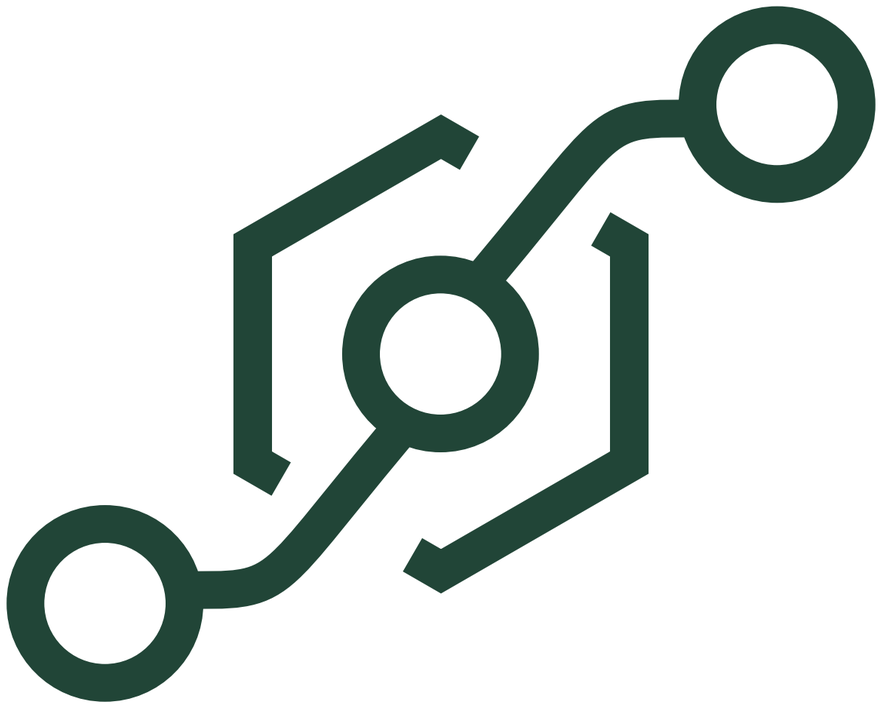
```
class Game extends HyperHTMLElement {
  get defaultState() {
    return { history: [], stepNumber: 0 };
  }

  handleClick(event) {
    this.setState({ ... })
  }

  render() {
    const history = this.state.history;
    const current = history[this.state.stepNumber];

    return this.html`
      <div class="game-board">
        <c-board
          squares=${current.squares}
          onboardclick=${i => this.handleClick(i)}
        />
      </div>
    `;
  }
}

Game.define('c-game');
```

github hyperHTML-Element

```
class Square extends HyperHTMLElement {
  static get observedAttributes() {
    return ['value']
  }

  static get booleanAttributes() {
    return ['disabled']
  }

  attributeChangedCallback() {
    this.render();
  }

  render() {
    return this.html`<button class="square">${this.value}</button>`
  }
}
Square.define('c-square')
```

SSR

viperHTML

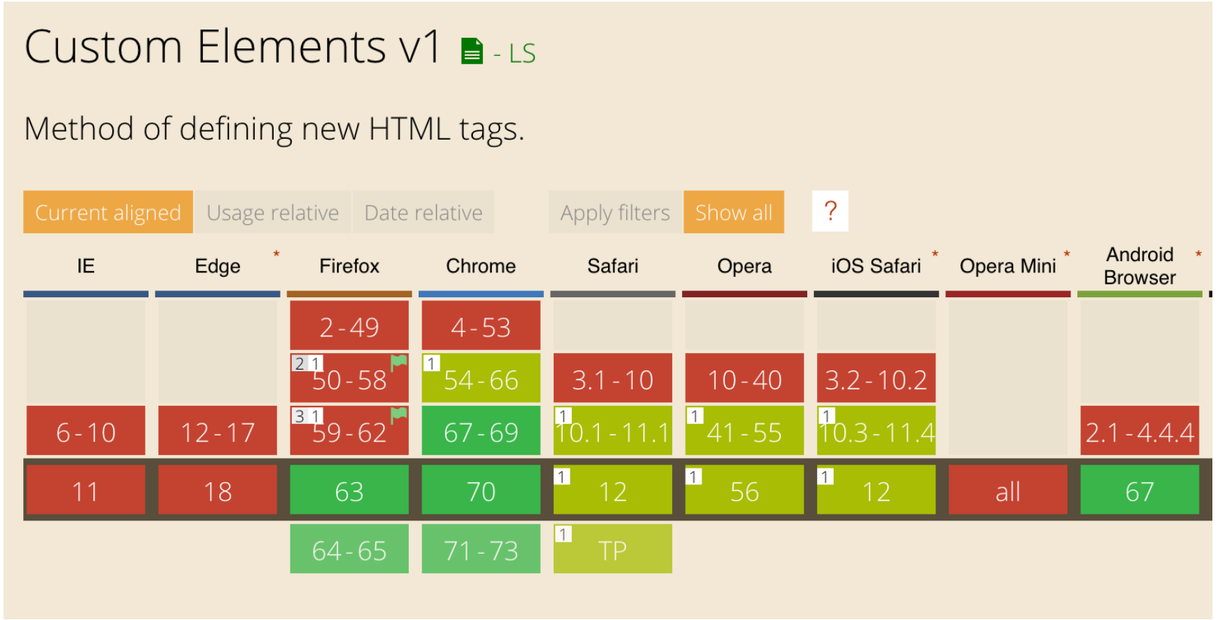


hyperHTML vs lit-html vs omi

vs lit-html

Tencent/omi

Custom Elements support



webcomponentsjs
WebReflection/document-register-element

Custom Elements in React

custom-elements-everywhere
Web Components in React

Use cases for Custom Elements and hyperHTML

- UI components and libraries
  - Primer - Github
  - Vaadin
- Third-party widgets
- Lightweight **framework-less compiler-less** development

Custom Properties

Custom Properties

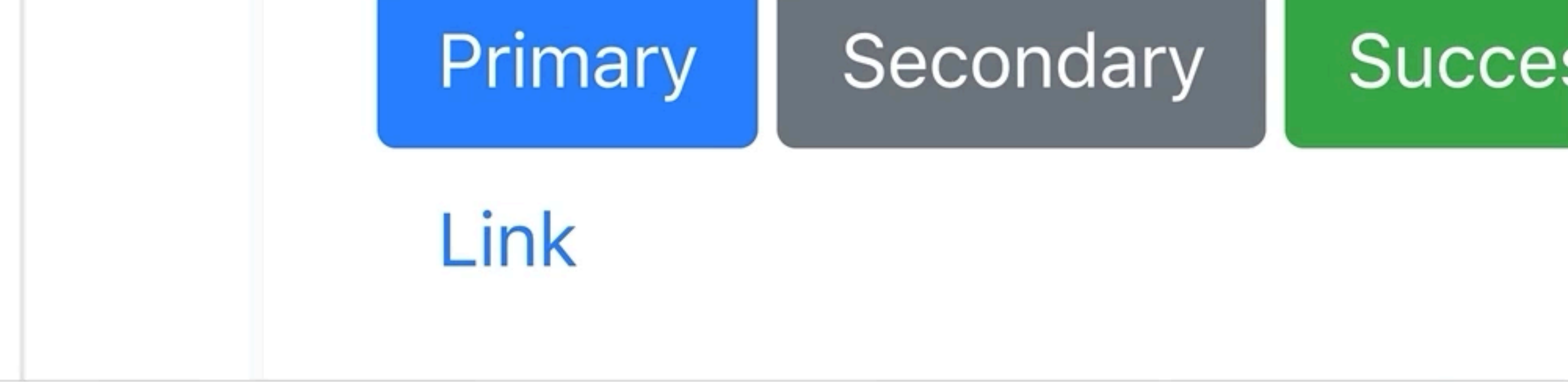
Runtime CSS *variables*

Or

Inherited user-defined *properties*

Runtime CSS Variables

```
:root { --primary: #007bff; }

.btn-primary { color: var(--primary, deepskyblue); }

.btn-primary:hover {
  --primary: crimson;
}
```



Elements Console Sources Network Performance Memory Application Security Audits Redux 1

```
<!doctype html>
<html lang="en" class="gr__getbootstrap_com">
▶<head>…</head>
▼<body data-gr-c-s-loaded="true">
  ▶<a id="skippy" class="sr-only sr-only-focusable" href="#content">…</a>
  ▶<header class="navbar navbar-expand navbar-dark flex-column flex-md-row bd-navbar">…
  </header>
  ▼<div class="container-fluid">
    ▼<div class="row flex-xl-nowrap">
      ▶<div class="col-12 col-md-3 col-xl-2 bd-sidebar">…</div>
      ▶<div class="d-none d-xl-block col-xl-2 bd-toc">…</div>
      ▼<main class="col-12 col-md-9 col-xl-8 py-md-3 pl-md-5 bd-content" role="main">
        <h1 class="bd-title" id="content">Buttons</h1>
        ▶<p class="bd-lead">…</p>
        <script async src="https://cdn.carbonads.com/carbon.js?
        serve=CKYIKKJL&placement=getbootstrapcom" id="_carbonads_js"></script>
        ▶<h2 id="examples">…</h2>
```

Styles Computed Event Listeners »

Filter :hov .cls

```
--pink: #e83e8c;
--red: #dc3545;
--orange: #fd7e14;
--yellow: #ffc107;
--green: #28a745;
--teal: #20c997;
--cyan: #17a2b8;
--white: #fff;
--gray: #6c757d;
--gray-dark: #343a40;
--primary: #007bff;
--secondary: #6c757d;
--success: #28a745;
--info: #17a2b8;
--warning: #ffc107;
```

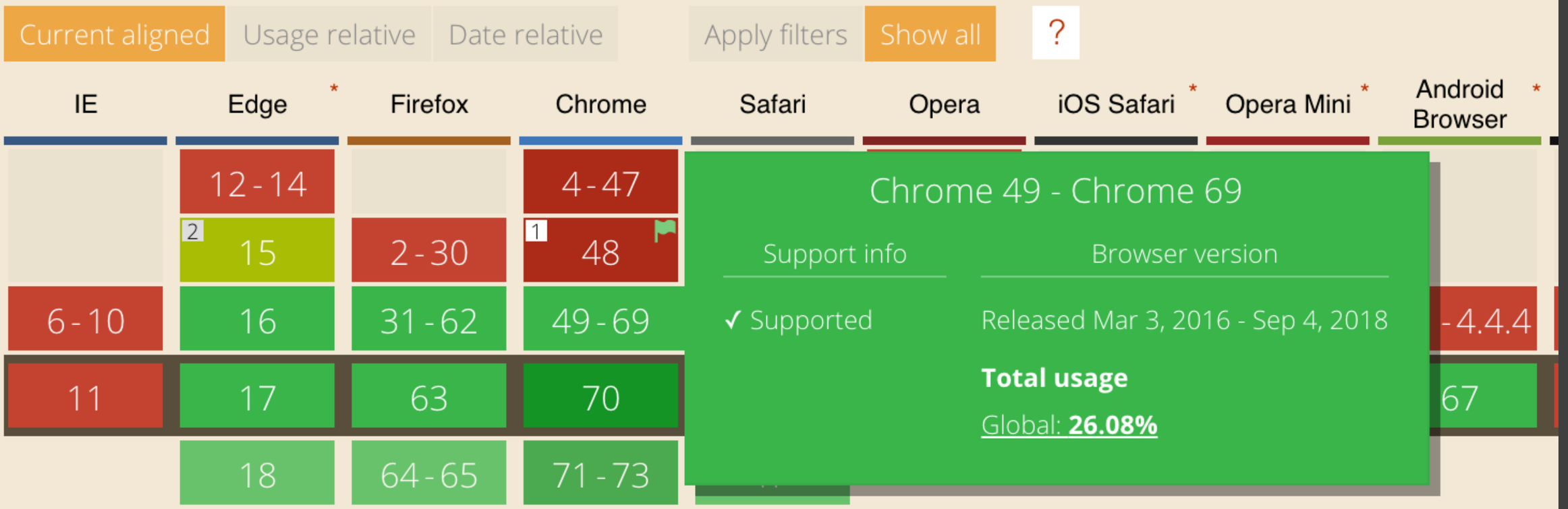
Inherited user-defined properties [ref]

```
/* CSS */
div > p { --primary: crimson; }
p { --primary: aqua; }

.c-block { background-color: var(--primary); }

<!-- HTML -->
<div>
  <p class="c-block"></p>
</div>
```

[ref] glazman: CSS Variables, why we drop the $foo notation

CSS Variables (Custom Properties) - CR

Permits the declaration and usage of cascading variables in stylesheets.



postcss-css-variables
postcss-custom-properties

```
:root {
  --color: red;
}

h1 {
  --background: midnightblue;
  color: var(--color);
  background-color: var(--background);
}

/* becomes */

:root {
  --color: red;
}

h1 {
  color: red;
  color: var(--color);

  --background: midnightblue;
  background-color: var(--background);
}
```

Interoperability

— Sass/Less

— React/Angular/Vue

— hyperHTML

— styled-components

— CSS-in-JS: linaria

Operations

```
:root {
  --columns: 12;
  --gutter: 16px;
}

.o-col {
  margin: 0 calc(var(--gutter) * 2);
  width: calc(100% / var(--columns));
}
```

Operations

```
:root {
  --alpha-hover: 0.04;
  --primary: 98, 0, 238;
}

.c-button:hover {
  background-color: rgba(var(--primary), var(--alpha-hover))
}
```

Operations

```
:root {
  --animation-duration-simple: 0.1s;
  --easing-standard: cubic-bezier(0.4, 0.0, 0.2, 1);
}

.c-box {
  transition:
              all
              var(--animation-duration-medium)
              var(--easing-standard);
}
```

Separate logic from design

Codepen

— All the logic at the top of the document

— See changing property

Custom properties in JS

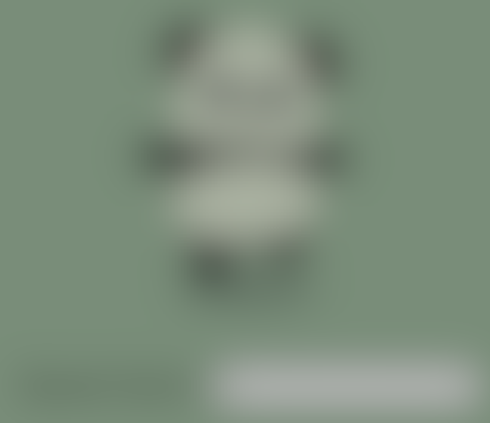
```
<button style={{ '--primary': colors.primary }}></button>
```

Vanilla JS

```
const getVariable = (el, propertyName) => {
  const styles = window.getComputedStyle(el);

  return String(styles.getPropertyValue(propertyName)).trim();
};

const setDocumentVariable = (propertyName, value) => {
  document.documentElement.style.setProperty(propertyName, value);
};
```

Speed factor: 1

Codepen



Codepen

Complaints

1. Syntax is ugly and verbose
2. Sass/Less already have variables

Preprocessor vs CSS Variables [2]

— Sass variables are static and lexically scoped

— CSS variables are live and scoped to the DOM

[2] philipwalton: Why I'm Excited About Native CSS Variables

What preprocessor cannot do

1. Interact with Javascript or 3rd party stylesheets
2. Aware of the DOM or CSSOM
3. Be changed dynamically
4. Cascade
5. Inherit

Responsive properties with media queries

```
$gutterSm: 1em;
$gutterMd: 2em;
$gutterLg: 3em;

.o-container {
  padding: $gutterSm;
}

@media (min-width: 30em) {
  .o-container {
    padding: $gutterMd;
  }
}

@media (min-width: 48em) {
  .o-container {
    padding: $gutterLg;
  }
}
```

Responsive properties with media queries

```
:root { --gutter: 1.5em; }

.o-container {
  padding: var(--gutter);
}

@media (min-width: 30em) {
  :root { --gutter: 2em; }
}
@media (min-width: 48em) {
  :root { --gutter: 3em; }
}
```

Reusable and extensible components

```html
<header class="header">
  <button class="c-button c-header-button"></button>
</header>
```

```css
.c-button {
  background-color: #eee;
  border: 2px solid crimson;
  color: crimson;
  font-size: 18px;
}

.c-header-button {
  background-color: #333;
  border: 2px solid aqua;
  color: aqua;
  font-size: 24px;
}

/* Or worse ... */
.header .c-button {}
```

Reusable and extensible components

```
.c-button {
  --btn-bg-color: #eee;
  --btn-primary-color: crimson;
  --btn-font-size: 18px;

  background-color: var(--btn-bg-color);
  border: 2px solid var(--btn-primary-color);
  color: var(--btn-primary-color);
  font-size: var(--btn-font-size);
}

.c-header-button {
  --btn-bg-color: #333;
  --btn-primary-color: aqua;
  --btn-font-size: 24px;
}
```

Component styling API [3]

API: Application programming interface [4]

“By abstracting the underlying implementation and only exposing objects or actions the developer needs, an API simplifies programming.”

[3] mrmrs: Component styling API

[4] Wikipedia - API

Theming

The act of laying a veneer over the top of an already styled website: an optional extra which alters or customises the UI

Static Theming

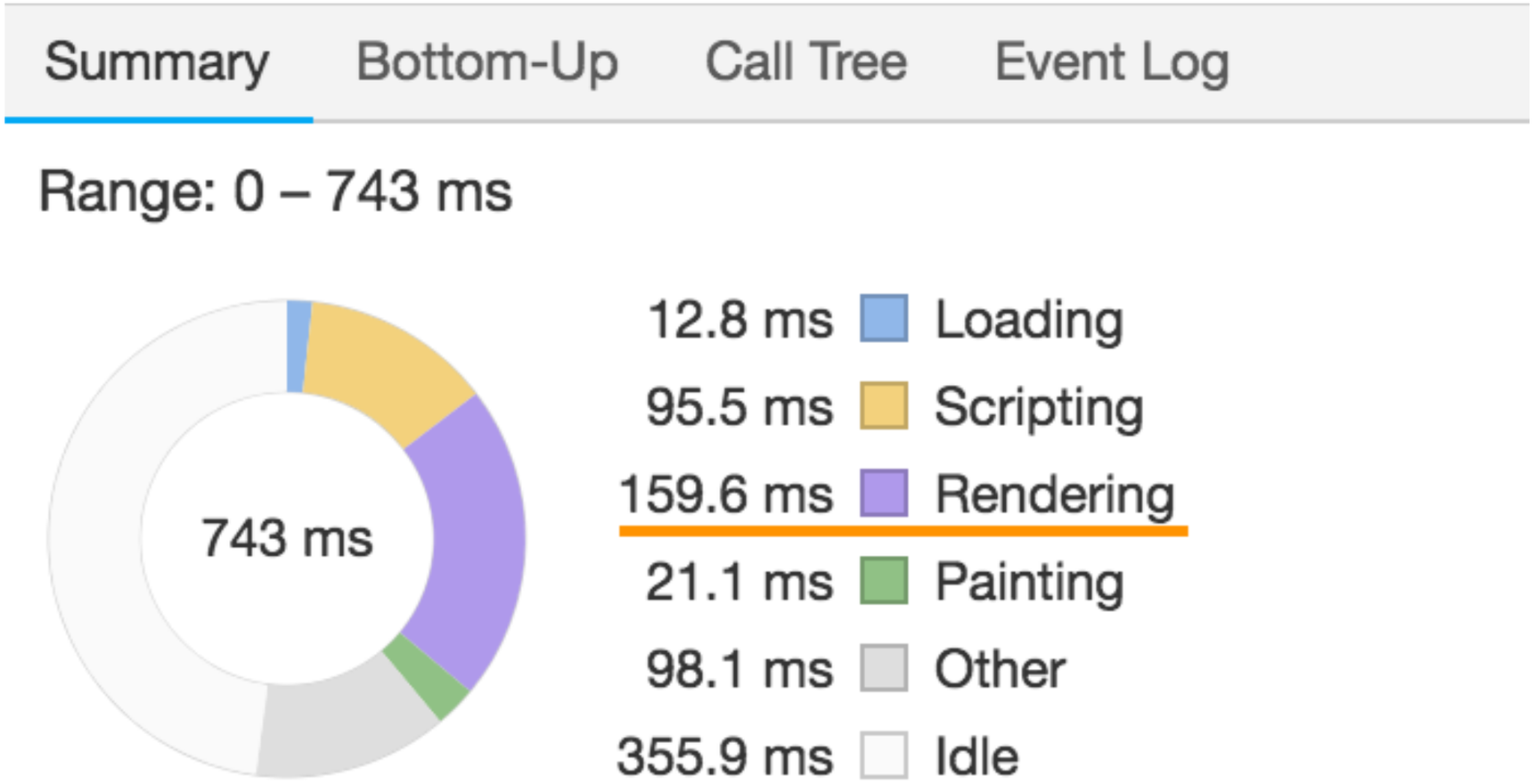
```css
/* settings.css */
:root {
  --main-color: #1b70de;
  --bg-color: #FFF;
  --text-color: #000;
  --button-color: rgba(0, 0, 0, 0.8);
  --header-color: #424242;
}

:root.dark {
  --main-color: darkblue;
  --bg-color: #333;
  --text-color: white;
  --button-color: black;
  --header-color: #333;
}
```

Theming with Sass

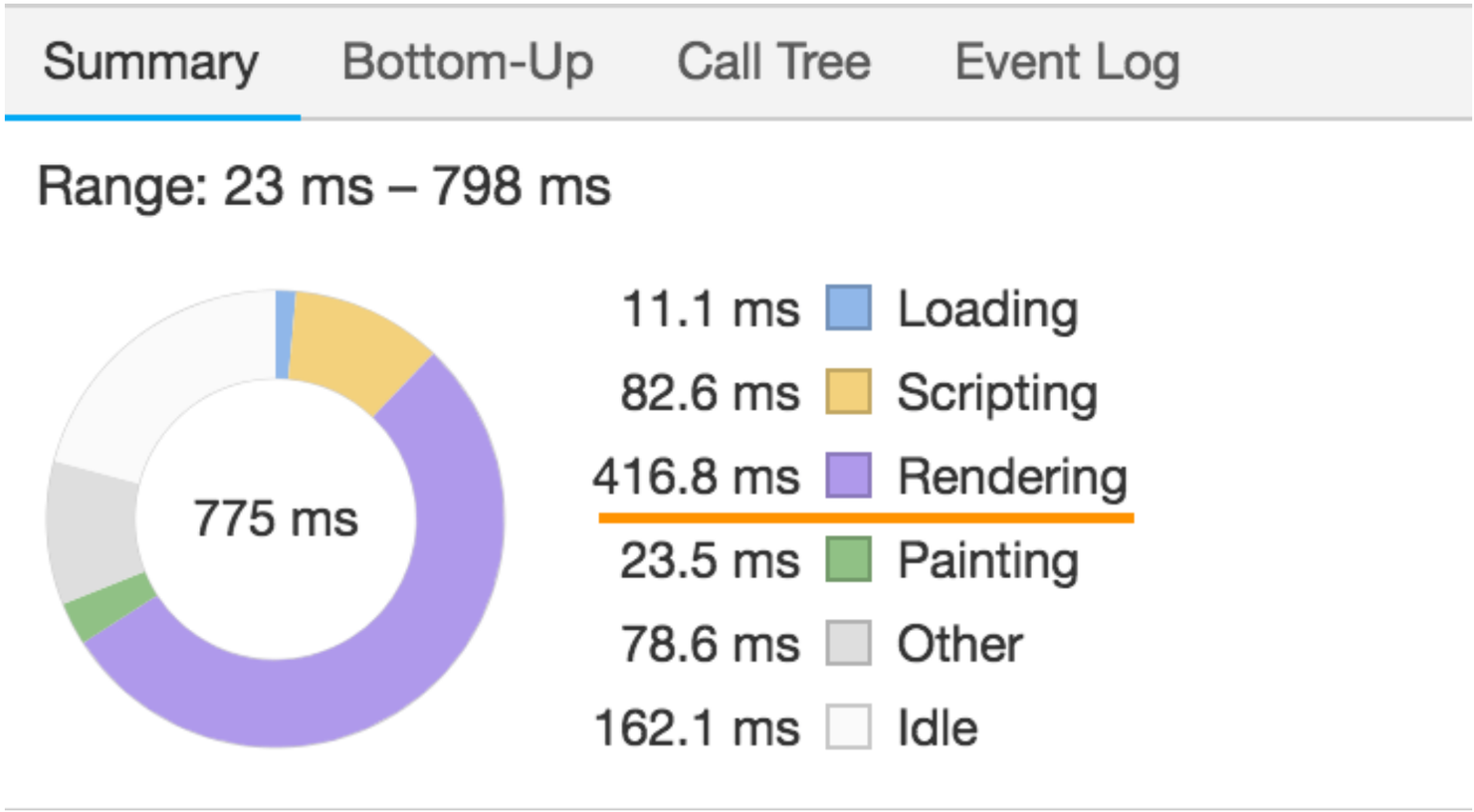
```
/* Navigation.css */
:host {
  /* --navigation-bg: var(--primary); */
  --navigation-bg: $primary;
}

.c-navigation {
  background-color: var(--navigation-bg, $primary);
}
```

Encapsulation and theming - Maxart

Start-up performance



Start-up performance [5]



[5] jiayihu: CSS Custom Properties performance in 2018

Recommendation

Use preprocessor for global static variables,
CSS custom properties for component styling API and theming.

About Custom Properties

It's like when OOP was first invented.

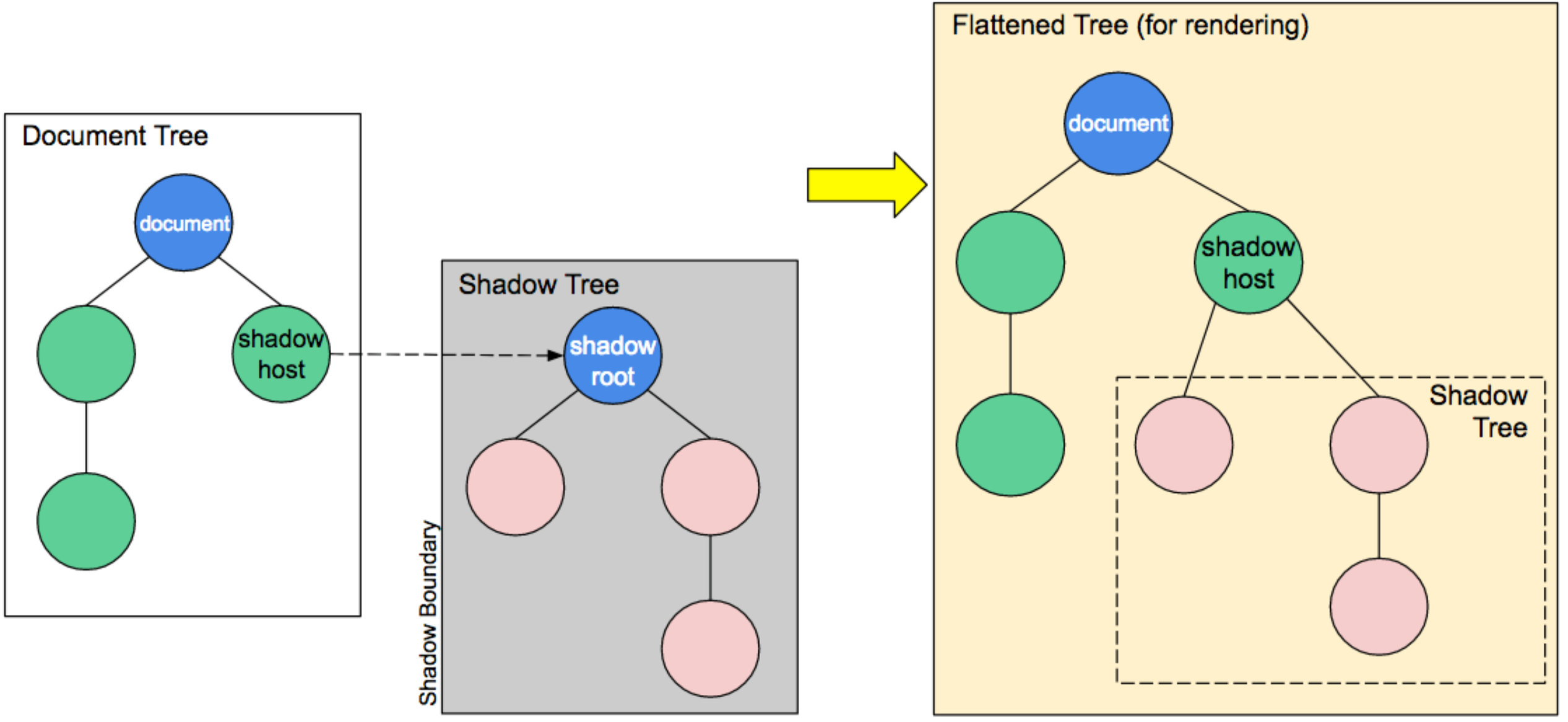
Shadow DOM

Shadow DOM

— Part of Web Components

— Used by native DOM elements

— Similar to `iframe`


Document Tree
document
shadow host
Shadow Tree
shadow root
Shadow Boundary
Flattened Tree (for rendering)
document
shadow host
Shadow Tree

— A boundary between the developer and the browser implementation

```
<input id="foo" type="range">
```

— <video>, <select> etc.

```
const hostEl = document.querySelector('.host');

const shadowRoot = hostEl.attachShadow({ mode: 'open' });
shadowRoot.innerHTML = `
  <style>
    p {
      color: red;
    }
  </style>

  <p>Element with Shadow DOM</p>
`;
```

Codepen

```
<c-button class="c-header-button"></c-button>
```

```
.c-header-button {
  --btn-bg-color: #333;
  --btn-primary-color: aqua;
  --btn-font-size: 24px;
}
```

```
const styles = `:host { background-color: ${props.theme} }`;

return (
    <ShadowDOM>
      <div>
        <style>{styles}</style>
        <h1>Calendar for {props.date}</h1>
      </div>
    </ShadowDOM>
);
```

ReactShadow

Encapsulation

Encapsulation is used to hide the values or state of a structured data object inside a class, preventing unauthorized parties direct access to them.

Publicly accessible methods are generally provided in the class.

Encapsulation

— Isolated DOM

— Re-targeted events

— Scoped CSS

— Simplify CSS selectors

New selectors

```
:host {
  --navigation-bg: var(--primary);

  all: initial;
}

:host([disabled]) {
  pointer-events: none;
  opacity: 0.4;
}
```

vjeux CSS-in-JS

1. Global namespace
2. Dependencies
3. Dead code
4. Minification
5. Sharing constants
6. Non-deterministic resolution
7. Breaking isolation

vjeux CSS-in-JS

1. Global namespace => **Shadow DOM**
2. Dependencies => **postcss**
3. Dead code => **Shadow DOM**
4. Minification => **Shadow DOM**
5. Sharing constants => **Custom Properties**
6. Non-deterministic resolution => **Shadow DOM**
7. Breaking isolation => **Shadow DOM**

Dead-code elimination

```
.c-promo-button {
  background: rebeccapurple;
}
```

Minification

CSS-Blocks => OptiCSS

```
.c-promo-button {
  background: rebeccapurple;
}

/* into */

.f {
  background: rebeccapurple;
}
```

Static analysis

No runtime cost for
- dead-code elimination
- minification

hyperHTML tic-tac-toe with Shadow DOM

One last slide…

Jiayi Hu

[dʒʌɪ]

Front-end developer

— jiayi.ghu@gmail.com

— Twitter: @jiayi_ghu

— GitHub: jiayihu/talks

— fevr.slack.com

Get hyper-excited