

# **Web Frameworks**

**Interface Pessoa-Máquina - 25/26 - LEI / UM**

**Hugo Pacheco**

**hpacheco@di.uminho.pt**

# The Web: O que usamos?

- 3 tecnologias / standards essenciais



- Largamente suportados pela vasta maioria dos browsers



# The Web: Foi sempre assim?

- W3C® fundado em 1994 por Tim Berners-Lee, criador da Web
  - Propósito de standardizar **HTML**, **DOM**, **CSS**, ...
- ecma INTERNATIONAL fundada em 1994 (renaming de agências anteriores)
  - Primeiro standard de **ECMAScript** (core de JS) lançado em 1997
- WHATWG fundado em 2004 como reação ao W3C
  - Atual maintainer de **HTML**, **DOM**, ...
- Standards relativamente estáveis desde 2015
  - Mas a Web nasceu em 1990...



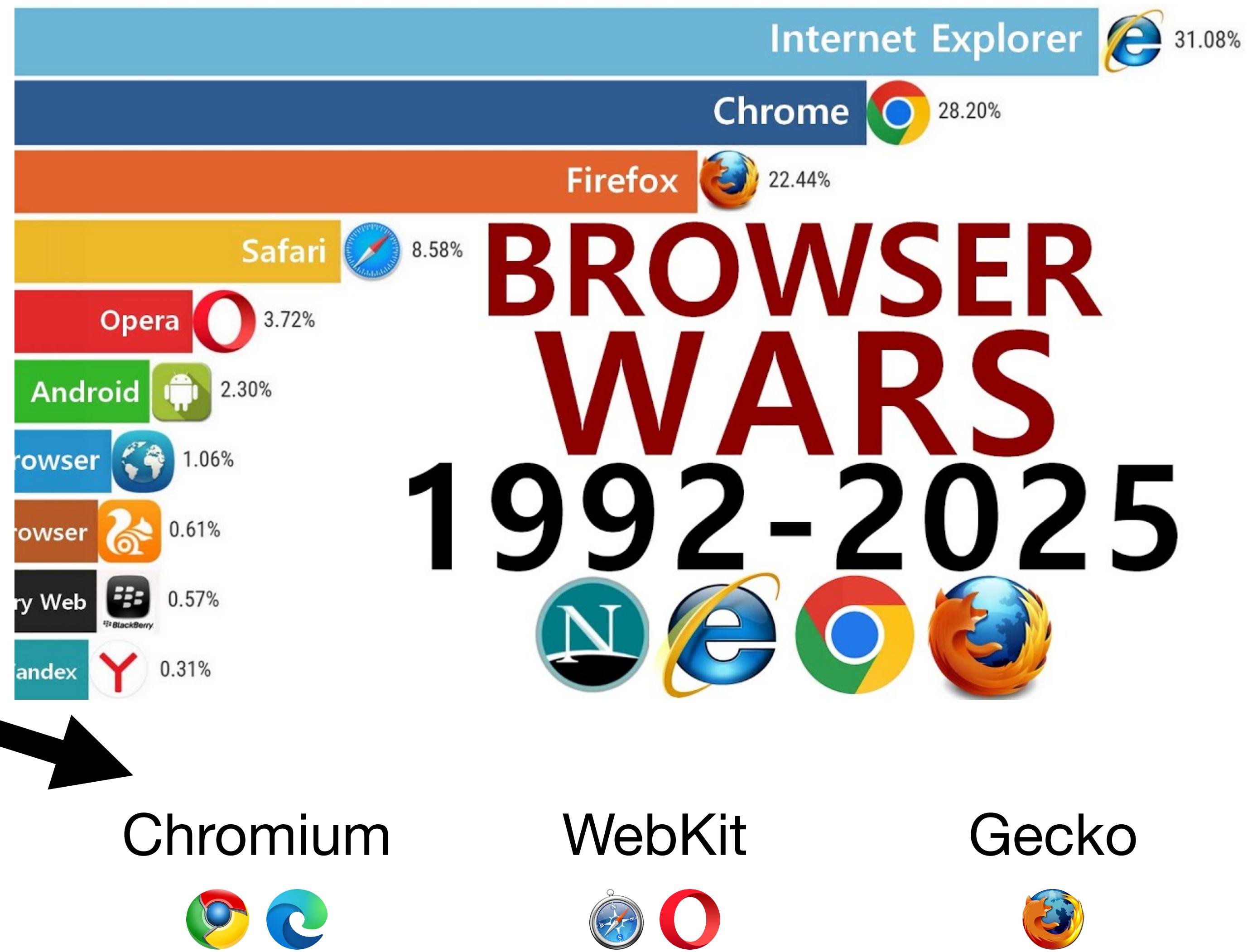
# 1995: Início das Browser Wars



Browsers freneticamente a “inovar” e a lançar funcionalidades incompatíveis, sem consideração por W3C e developers

# 2025: Browsers

- Pelo caminho...
  -  defunto ⇒ 
  -  descontinuado ⇒ 
- Hoje...
  -  domina
  - Web engines open-source
  - Fim da história?



# 2015: Início das Framework Wars

JS

TS

T

node

js

A

Backend

F

ex

Frontend

- Proliferação de “right framework for the job”



# 2025: Frameworks

- A guerra continua...
  - Longe de consenso, continuam a nascer muitas frameworks novas
- Como escolher uma framework?
  - Popularidade, suporte comunitário, preferências pessoais, evangelização
  - Funcionalidade (abstrações e ênfases diferentes)
  - Investimento (devido a baixa interoperabilidade no geral)
- Vale a pena usar uma framework?
  - Estamos em 2025...
  - Algumas lições comuns

JS

# Web Frameworks: O que são?

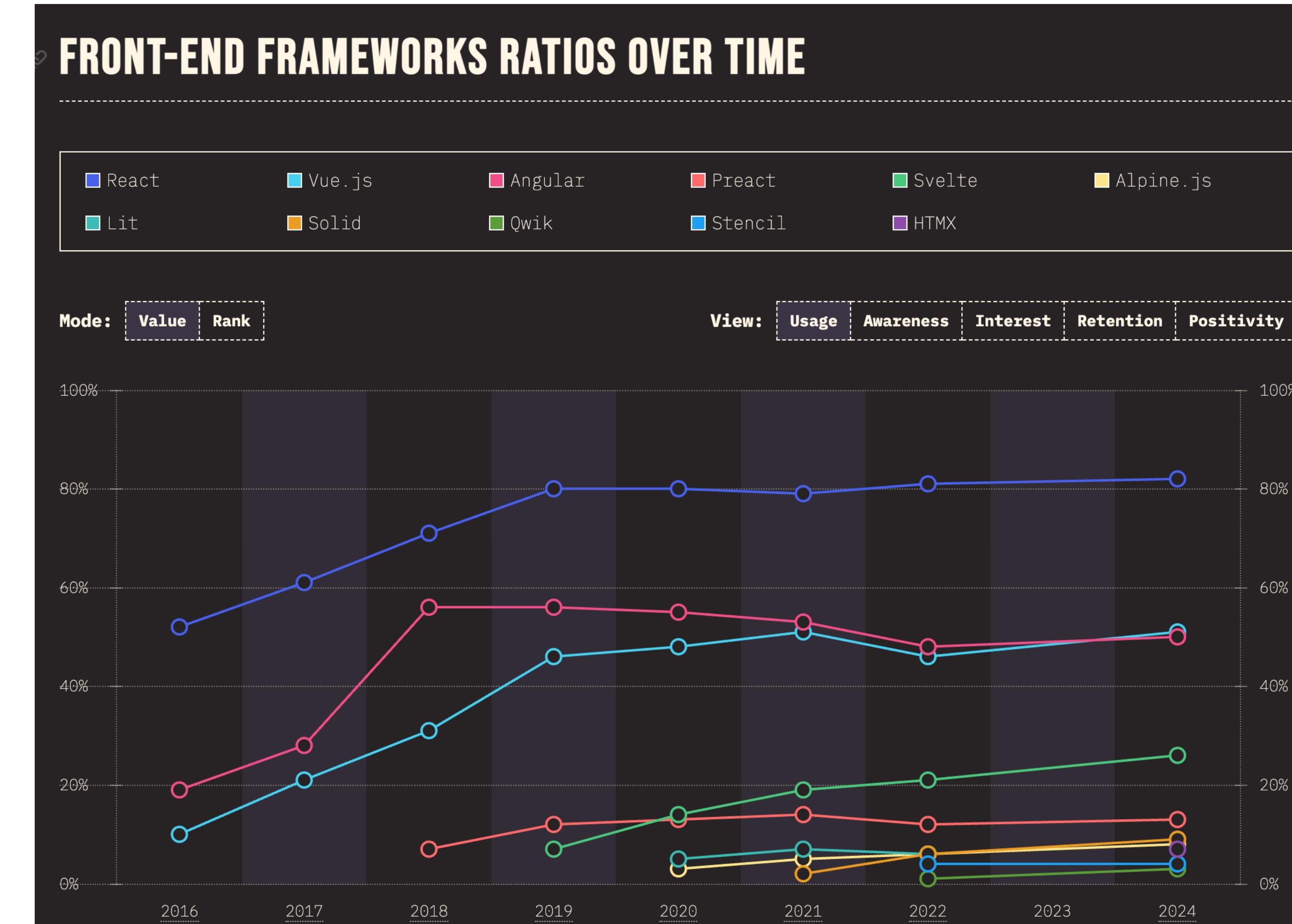
- Pre-written code libraries or collections of code modules that provide developers with a structured approach to building web applications. Streamline the development process by abstracting common tasks and promote code reuse.
- **Todos os exemplos anteriores são frameworks JS! Backend?**
  - Disseminado com 
  - Não há grandes razões, para além de aproveitar familiaridade de developers e evangelização
  - Muitas outras backend frameworks, por exemplo:



# Web Frameworks (Frontend)

- Vamos considerar apenas **frontend development**
- Tipicamente Single-Page Applications (SPAs)
  - Um só ficheiro HTML
  - Maioria da lógica executada (em JS) do lado do cliente
- Reduzir (exposição a) detalhes de implementação
  - Developers escrevem o “mínimo” de JS possível

<https://2024.stateofjs.com/en-US/libraries/front-end-frameworks/>



# Web Frameworks: Funcionalidades

- Desenvolver web apps completas, não apenas parte comportamental
- **Trend principal**
  - Promover separação entre
    - **modelo da aplicação** (anteriormente inexistente)
    - **visualização** (HTML + CSS)
  - “So you don't want to structure your **data** like your **UI**, that would be terrible. It would mean every time you want to move a widget from one panel to another, you have to change your entire **data model** end-to-end.” (Steven Wittens)

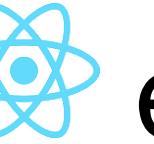
# Web Frameworks: Funcionalidades

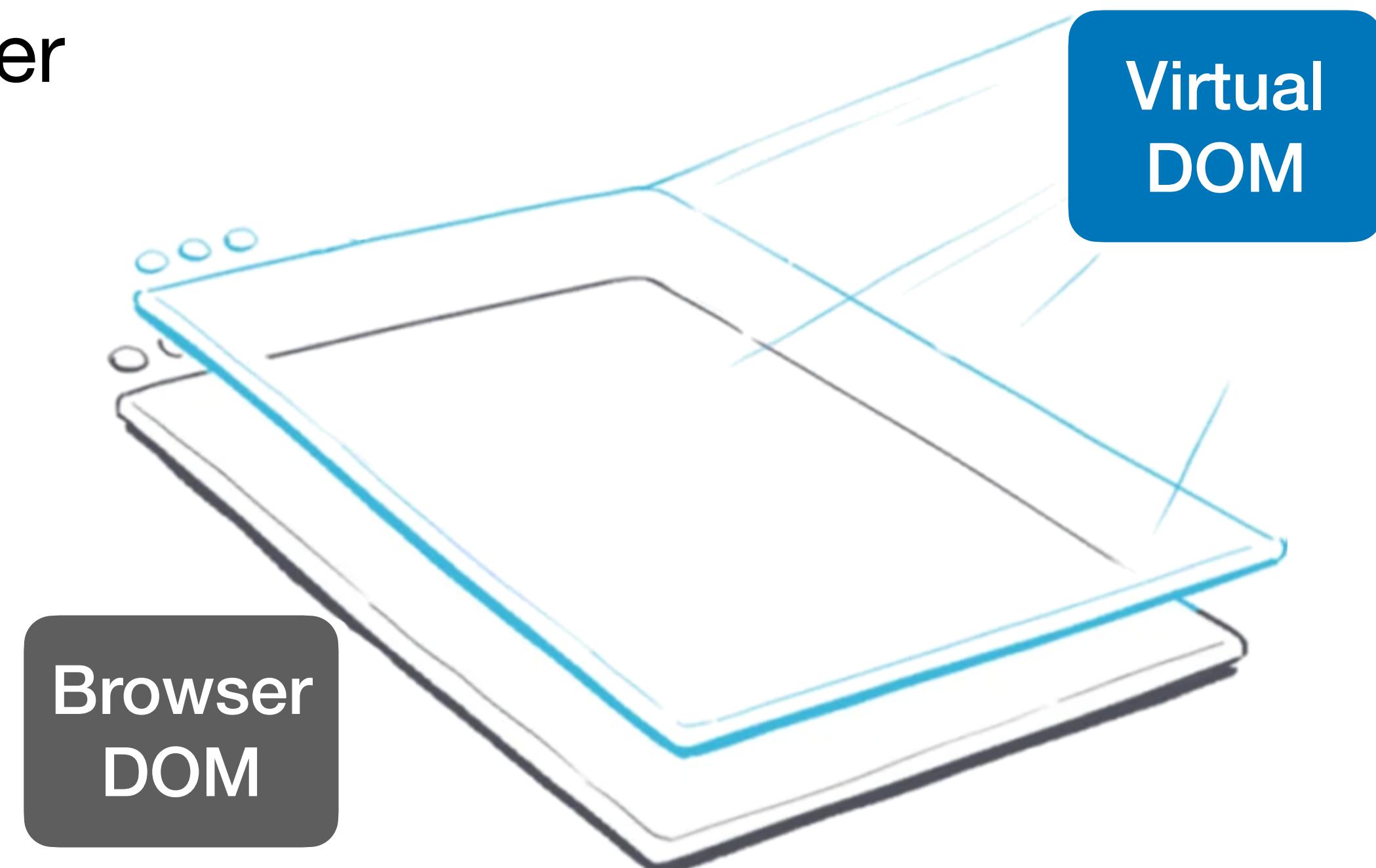
- Tentam abstrair e simplificar tarefas clássicas

O DOM (Document Object Model) é uma árvore que guarda todo o conteúdo e os elementos de uma página web como objetos, permitindo que o browser os organize e o JavaScript os manipule

- DOM manipulation
  - Anteriormente feito com JS
- Data binding
  - Ligação entre o modelo da aplicação e DOM
- Routing
  - Mapeamento de URLs em páginas
  - Anteriormente um ficheiro HTML por página
- State management
  - Como forma de comunicação entre diferentes páginas

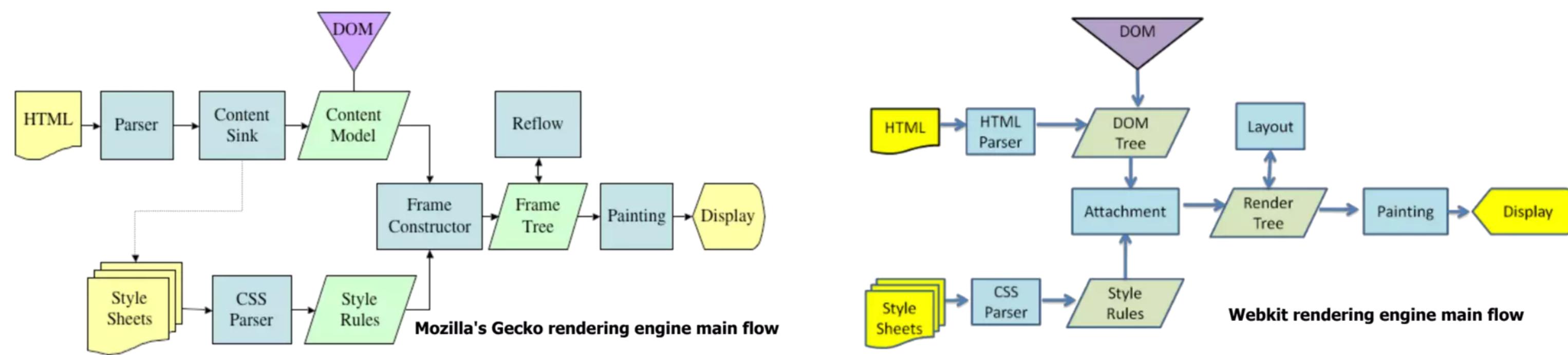
# Trend: Virtual DOM

- Popularizado pela framework React  em 2013. Hoje utilizado pela maioria das frameworks.
- Essencialmente uma optimização
  - DOM tree construída e mantida pelo browser
  - 💡 Aplicação mantém, em memória, uma representação “virtual” do DOM (VDOM)
    - Interação do utilizador toda feita com o VDOM
    - Sincronização automática do estado do VDOM com o DOM



# Trend: Virtual DOM

- No browser, a renderização completa do DOM é bastante rápida



- Modificações no DOM despoletam Reflow/Layout ⇒ bastante mais lento
- VDOM mitiga o problema, reduzindo o número de modificações do DOM
  1. *Diffing algorithm* calcula eficientemente modificações no VDOM
  2. *Batching optimizations* agregam várias modificações do DOM

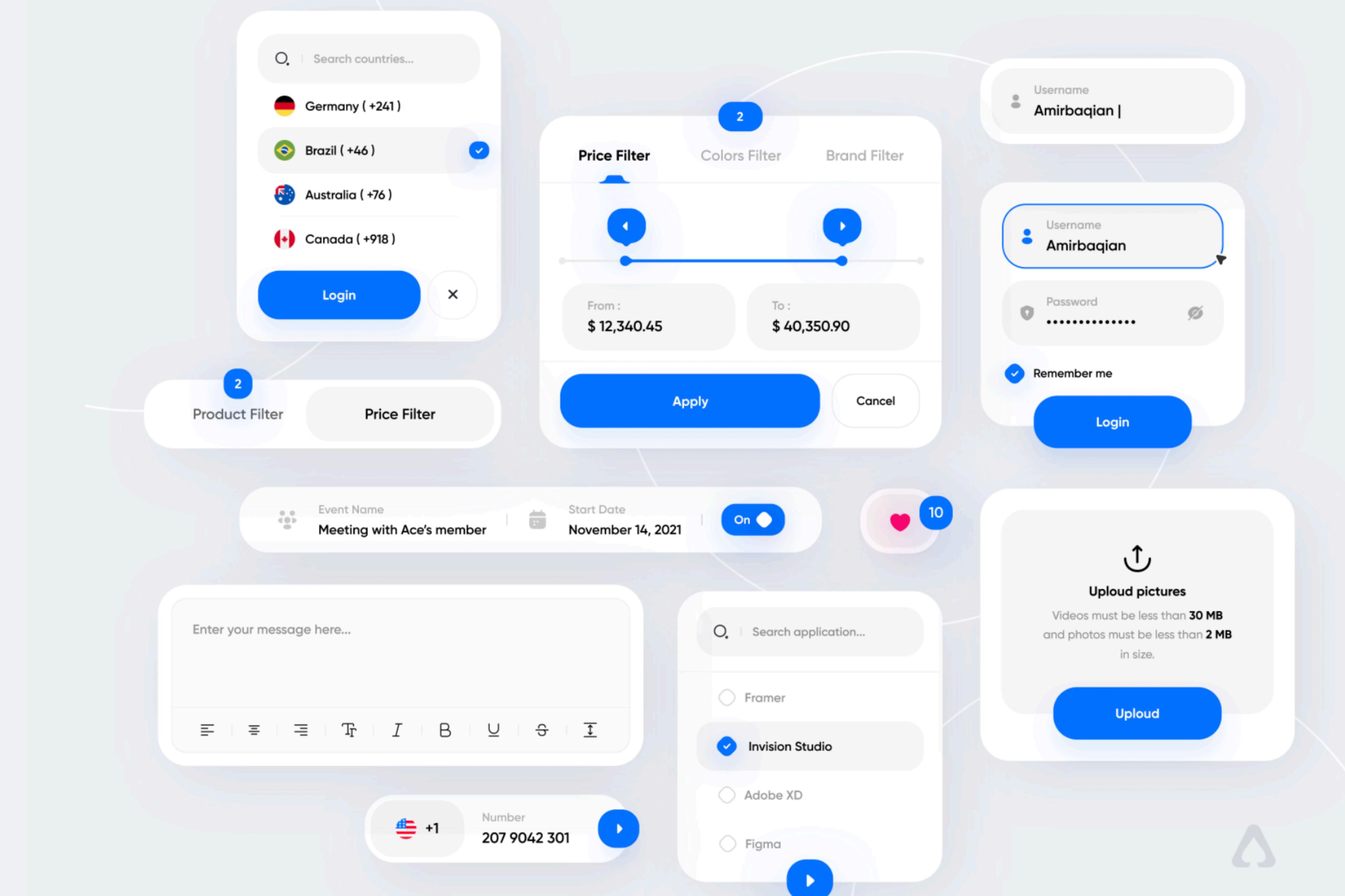
# Trend: Virtual DOM

- Uma **revolução declarativa**. O developer:
  - **Não tem que especificar** quais nodos no (V)DOM foram modificados
  - **Apenas declara** como o modelo da aplicação deve ser visualizado
- Paradigma conhecido como **reactive programming**
  - Programador escreve funções que mapeiam input em output
  - Linguagem regista dependências entre input e output
  - Output é automaticamente atualizado quando input muda
- 💡 Técnica tão mais poderosa quanto mais informação estática (tipos, compilador) a framework oferecer

# Web Frameworks: Modularização

- Procuram oferecer abstrações programáticas para organização interna de web apps e promover reutilização
- Frequentemente “opinionated”: definem um conjunto de convenções e boas práticas
- Explosão de frameworks = explosão de jargão. Dois termos recorrentes:
  - **Templates:** tipicamente excertos de HTML estático com anotações para dados dinâmicos
  - **Components:** unidades “independentes” dentro de uma web app
- Usam frequentemente diferentes termos e técnicas para modelar o mesmo conceito ⇒ **componentes parametrizáveis**

# Trend: Components

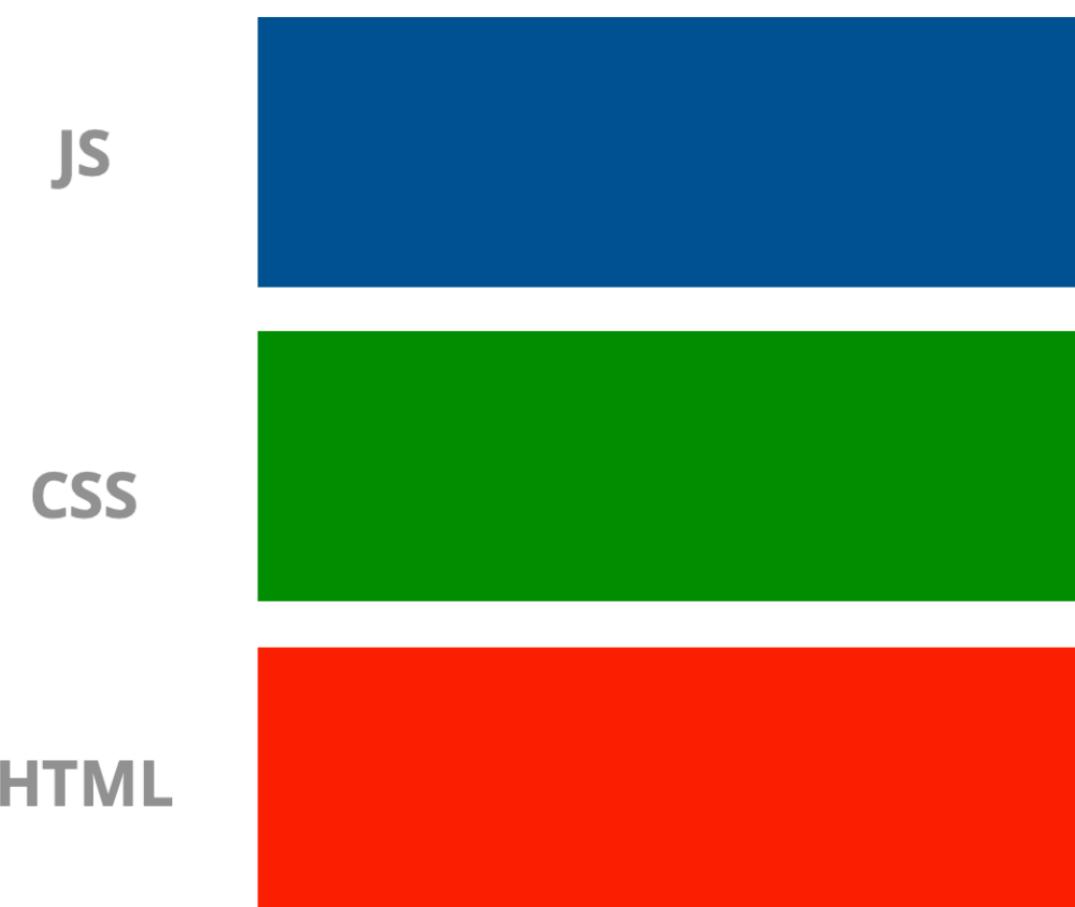


# Trend: Components

- “Rather than separating concerns by languages, we are working towards a model of **separating concerns at the component level.**”  
(GitHub)
- “...but that **doesn't mean we have to stop separating structure, presentation, and behaviour!**”

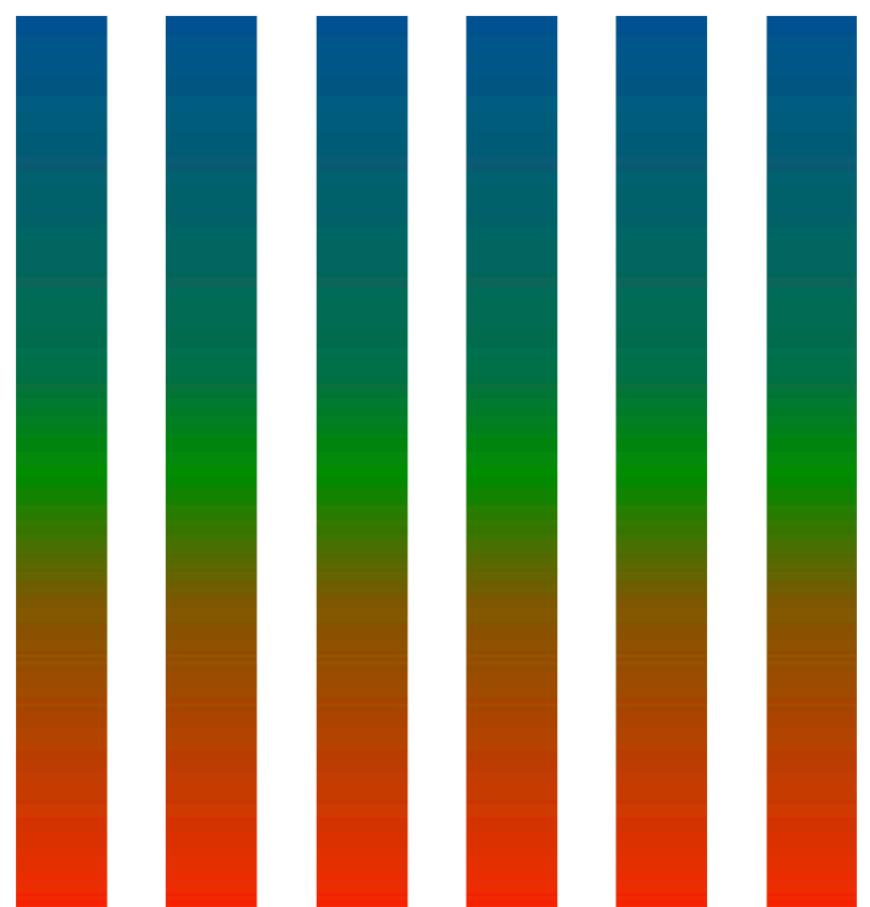
Arquitectura Web  
tradicional  
(Vertical)

Separation of Concerns



Padrão arquitectural  
emergente  
(Horizontal+Vertical)

Separation of Concerns  
*(only, from a different point of view)*



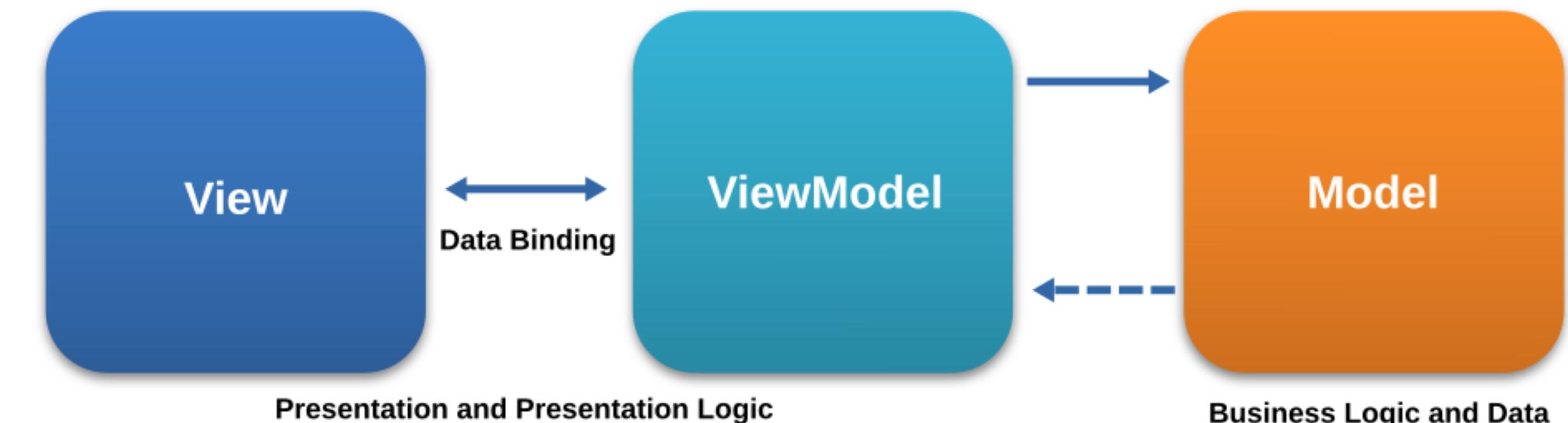
BUTTON  
DATE PICKER  
MODAL  
LIST  
LIST-ITEM  
MEDIA

# Trend: Components

- **Components instanciam/implementam Concepts**
  - “I like to think of **components** as encapsulating UI structure, behavior (perhaps even state), and styling (ideally, too), based on a **single unifying concern**. It’s like the atomic building blocks that make up a UI.” (Magne, a React developer)
  - Refinam **concepts** ao definirem UI concretas
- Design deve seguir o mesmo princípio de mapeamento “**purpose** → **component**”
- Overloading do termo **components**, por vezes apenas formas de modularização de código
  - “But they don’t really represent parts of the UI. They don’t denote WHAT is there, but HOW (something else) is there. That’s why I call “control-flow components” for logic-posing-as-markup, or imperative-posing-as-declarative constructs.” (Magne, a React developer)

# Web Frameworks: Architectures

- Arquitectura mais comum é talvez **Model-View-ViewModel** (MVVM)
  - Uma evolução para frontends de MVC (tipicamente utilizado em backends)
  - Aliada a uma separação horizontal entre **components**
- **Model:** armazena dados e business logic (operações sobre dados)
- **View:** define a UI, que representa os dados do model
- **ViewModel:**
  - Define propriedades que são lidas e atualizadas pela view (data binding)
  - Efetua updates no model



# Qual o padrão arquitectural da framework?



# JavaScript **js** (Calculadora)

- Relembrar como construímos uma calculadora bidirecional só com JS
  - Alterar número → Atualizar soma
  - Atualizar números (divide por 2) ← Alterar soma

A screenshot of a web-based calculator application. It features four input fields: the first contains '2', the second contains '3.5' with a blue border, the third contains '5.5', and the fourth contains the text 'Equalize'. Between the first and second fields is a plus sign '+'. Between the second and third fields is an equals sign '='. Between the third and fourth fields is a small up-and-down arrow icon.

- 💡 Variáveis DOM mapeiam cada elemento **HTML**
- 💡 Event listeners para cada **input** event de cada **input** field

# Angular

- A framework mais antiga e uma das mais populares, desenvolvida pela Google
- Difícil curva de aprendizagem
- Não tem propriamente uma arquitectura, inclinação para MVMM
- Funcionalidades principais:
  - Components
  - Templates + Directives + Pipes
  - Two-way data binding (DOM atualiza automaticamente quando modelo muda + modelo atualiza automaticamente quando DOM muda)
  - Services + Dependency Injection

# Angular (Calculadora)



CONTENT UNAVAILABLE

# React

- A framework mais popular, desenvolvida pelo Facebook
- Não tem propriamente uma arquitectura = fusão de conceitos dentro de cada component
- Funcionalidades principais:
  - Virtual DOM
  - One-way data binding (modelo não atualiza automaticamente)
  - Components
  - Hooks (state management para components)
  - JSX (fusão de HTML tags com JS)

# React (Calculadora)

Component

Model

JSX

View

Hooks

```
function Equalizer() {  
  const [a, setA] = React.useState(0);  
  const [b, setB] = React.useState(0);  
  const [c, setC] = React.useState(0);  
  ...  
  return (  
    <div>  
      <input  
        type="number"  
        step="0.1"  
        value={a}  
        onChange={handleAChange}  
      />  
      {" + "}  
      <input  
        type="number"  
        step="0.1"  
        value={b}  
        onChange={handleBChange}  
      />  
      <input  
        type="number"  
        step="0.1"  
        value={c}  
        onChange={handleCChange}  
      />  
      <button onClick={doEqualize}>  
        Equalize  
      </button>  
    </div>  
  );  
}
```

One-way data binding

ViewModel

# Vue.js

- Uma das frameworks mais populares, celebrada pela sua simplicidade e flexibilidade
- Desenhada para ser adoptada incrementalmente
- Arquitectura assumidamente MVVM
- Funcionalidades principais:
  - Virtual DOM
  - Reactivity system
    - One-way data binding, suporte limitado (syntactic sugar) para two-way
  - Single-File Components
  - Computed Properties + Watchers (reactive programming)

# Vue.js (Calculadora)

View

```
<template>
  <div>
    <input v-model.number="a"
      type="number" step="0.1" />
    +
    <input v-model.number="b"
      type="number" step="0.1" />
    =
    <input
      v-model.number="c"
      type="number"
      step="0.1"
      @input="doEqualize" />
    <button @click="doEqualize">
      Equalize</button>
  </div>
</template>
```

Two-way  
data binding

Model

```
<script setup>
...
const a = ref(0);
const b = ref(0);
```

Reactivity

```
const c = computed(() => a.value +
  b.value);
</script>
```

Computed  
Property

ViewModel

# Até agora... um dogma

- Funcionalidades úteis e poderosas, mas podemos ser mais declarativos?
  - E.g., é necessário separar variáveis mutáveis e não mutáveis?
- As frameworks são desenhadas para JS web developers
  - “Some frameworks choose to overcome such limitations by implementing reactivity during compilation. The Vue team did explore this direction via an experimental feature called Reactivity Transform”. “Some users have expressed strong objections [...] This is a valid concern because Reactivity Transform entails a different mental model that **distorts JavaScript semantics** (variable assignments being able to trigger reactive effects).” (Vue Documentation)



# Svelte

- Uma framework que prima pela elegância e simplicidade
- A arquitectura mais parecida com web development tradicional (**HTML\*** + **CSS** + **JS\***)
- Curva de aprendizagem mais pequena, em particular para developers com experiência limitada com outras frameworks
- Funcionalidades principais:
  - Compilador que gera JS optimizado
    - Ao contrário das frameworks anteriores, executa grande parte das decisões (e.g. updates incrementais do DOM) durante a compilação
  - Sem Virtual DOM
  - Reactivity with Simplicity

\*com extensões

# Svelte (Calculadora)

Única framework em que o código cabe no slide

Variável auxiliar computada

```
<script>
  let a = 0;
  let b = 0;
  $: c = a + b;

  function doEqualize() {
    const r = c / 2;
    a = r;
    b = r;
  }

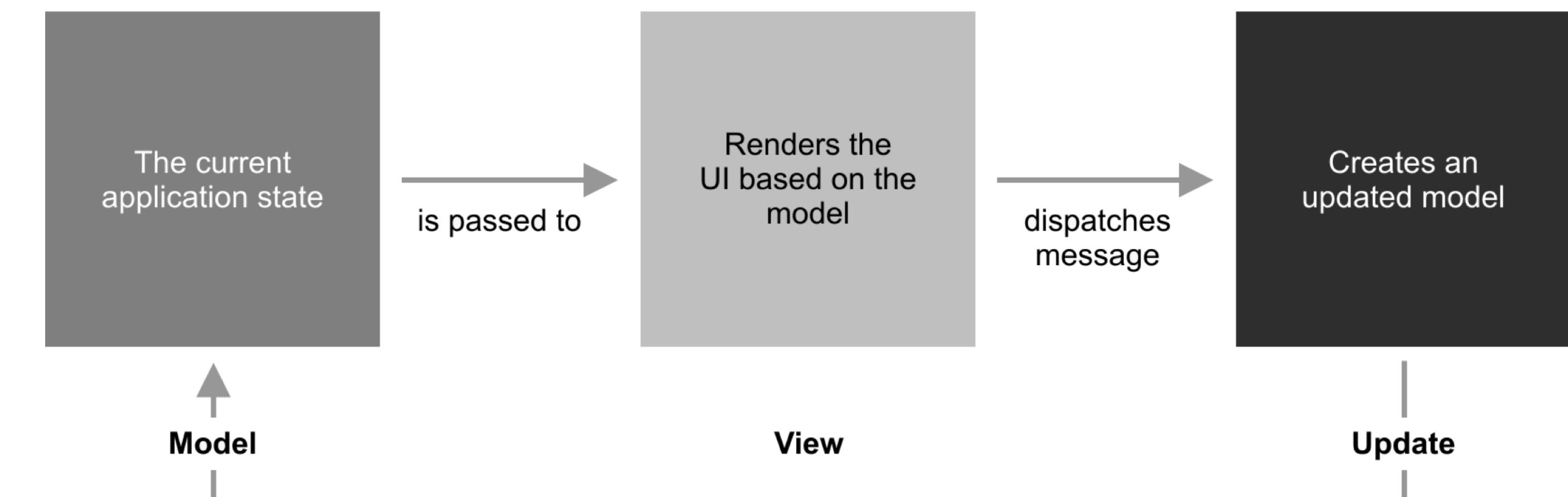
</script>
```

Variáveis reativas

```
<div>
  <input type="number"
  step="0.1" bind:value={a}> +
  <input type="number"
  step="0.1" bind:value={b}> =
  <input
    type="number"
    step="0.1"
    bind:value={c}
    on:input={doEqualize}
  >
  <button
    on:click={doEqualize}
  >Equalize</button>
</div>
```

# Elm

- Uma linguagem puramente funcional (Haskell simplificado)
- Arquitectura **Model-View-Update**, simplificação de MVVM com fluxo unidirecional
- Funcionalidades Principais:
  - Compilador gera JS otimizado e reporta erros anteriormente de runtime
  - Static types (em oposição a tipos dinâmicos de frameworks anteriores)
  - Immutability
  - Referential Transparency
  - Virtual DOM



# Elm (Calculadora)

```
module Main exposing (..)

import Browser
import Html exposing (Attribute, Html, button, div, input, text)
import Html.Attributes exposing (step, type_, value)
import Html.Events exposing (onClick, onInput)
import String

type alias Model = { a : Float, b : Float, c : Float }

init : Model
init = { a = 0, b = 0, c = 0 }

type Msg
    = UpdateA Float
    | UpdateB Float
    | UpdateC Float
    | Equalize

update : Msg -> Model -> Model
update msg model = case msg of
    UpdateA aVal -> { model | a = aVal, c = aVal + model.b }
    UpdateB bVal -> { model | b = bVal, c = model.a + bVal }
    UpdateC cVal -> let r = cVal / 2 in { model | a = r, b = r, c = cVal }
    Equalize -> let r = model.c / 2 in { model | a = r, b = r, c = model.c }

number : Float -> Attribute msg
number f = value <| String.fromFloat f

onInputNumber : (Float -> msg) -> Attribute msg
onInputNumber h = onInput (\str -> String.toFloat str |> Maybe.withDefault 0 |> h)
```

Model

Comparativo de Frameworks (Resumo Teórico)

Angular: Google; complexa; utiliza Two-way data binding (DOM e modelo sincronizam automaticamente).

React: Facebook; a mais popular; utiliza Virtual DOM e One-way data binding; introduziu o JSX (HTML dentro do JS).

Vue.js: Simplicidade e flexibilidade; arquitetura assumidamente MVVM; utiliza Virtual DOM e sistema de reatividade.

Svelte: Não usa Virtual DOM. É um compilador que gera código JS otimizado durante a build para atualizar o DOM diretamente.

Elm: Linguagem funcional pura; arquitetura MVU; tipos estáticos e imutabilidade; garante a inexistência de erros de runtime

Update

```
view : Model -> Html Msg
view model =
    div []
        [ input
            [ type_ "number"
            , step "0.1"
            , number model.a
            , onInputNumber UpdateA
            ]
        []
        , text " + "
        , input
            [ type_ "number"
            , step "0.1"
            , number model.b
            , onInputNumber UpdateB
            ]
        []
        , text " = "
        , input
            [ type_ "number"
            , step "0.1"
            , number model.c
            , onInputNumber UpdateC
            ]
        []
        , button [ onClick Equalize ] [ text "Equalize" ]
    ]

main = Browser.sandbox { init = init, update = update, view = view }
```

View

MVU