ORDEN EN EL CAOS

POLYMER

v-1.0

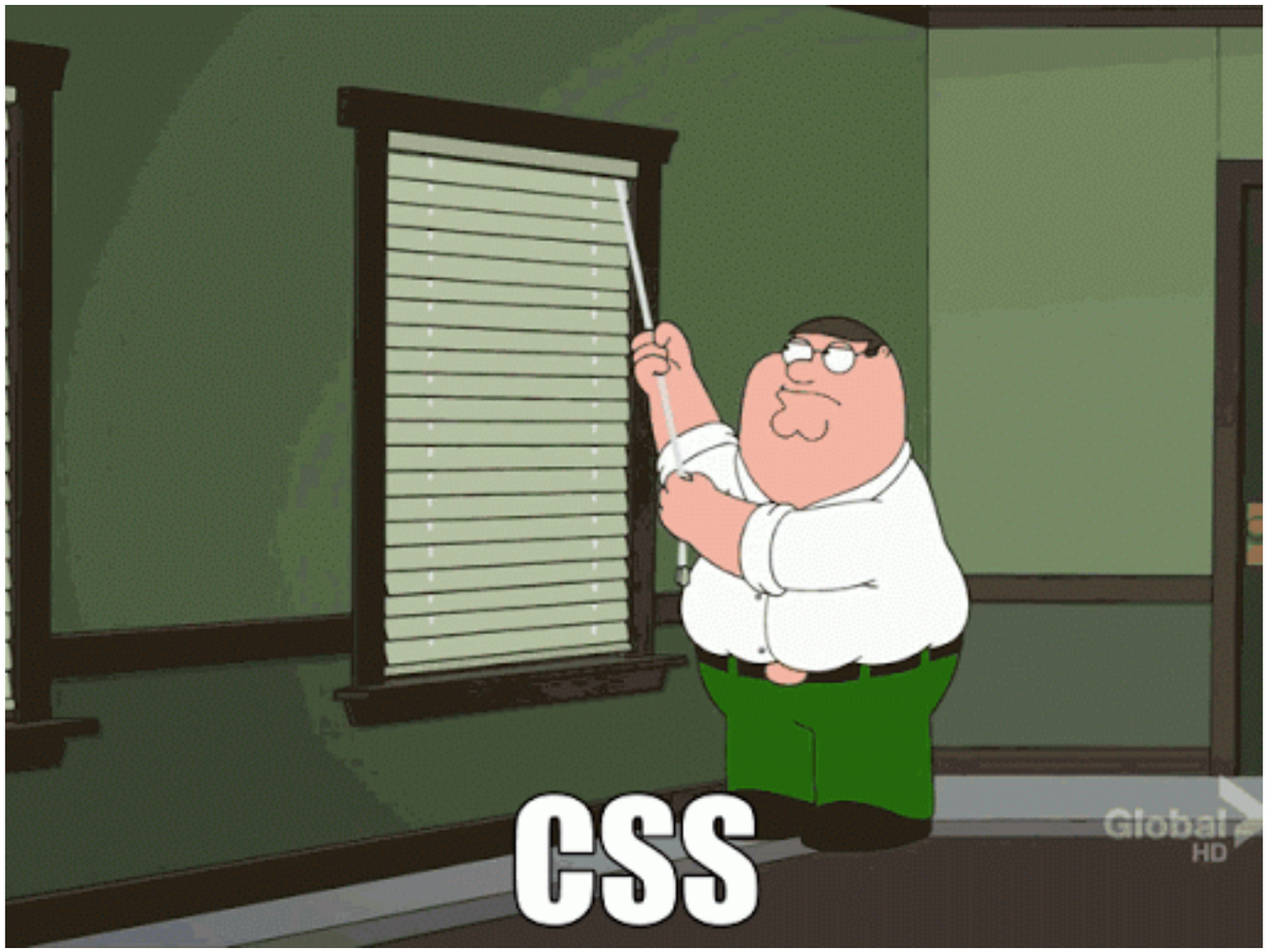
QUE SON LOS WEB COMPONENTS

- EL DESARROLLO FRONT DE APLICACIONES A GRAN ESCALA ANTES DE LOS WEB COMPONENTS
- DE LA WEB COMO REPOSITORIO DE DOCUMENTOS ESTÁTICOS A LA WEB COMO SERVIDOR DE APLICACIONES
- APLICACIONES NATIVAS FRENTE A APLICACIONES WEB, UNA PELEA DESIGUAL
- CONCLUSIÓN: SE NECESITA UNA NUEVA METODOLOGÍA DE DESARROLLO
- EL DISEÑO MODULAR COMO RESPUESTA
- LIMITES DE LA WEB
- DEFINICIÓN DE WEB COMPONENTS
- EJEMPLO NATIVO DE WEB COMPONENT: <INPUT TYPE="DATE />
- TECNOLOGÍAS UTILIZADAS EN WEB COMPONENTS:

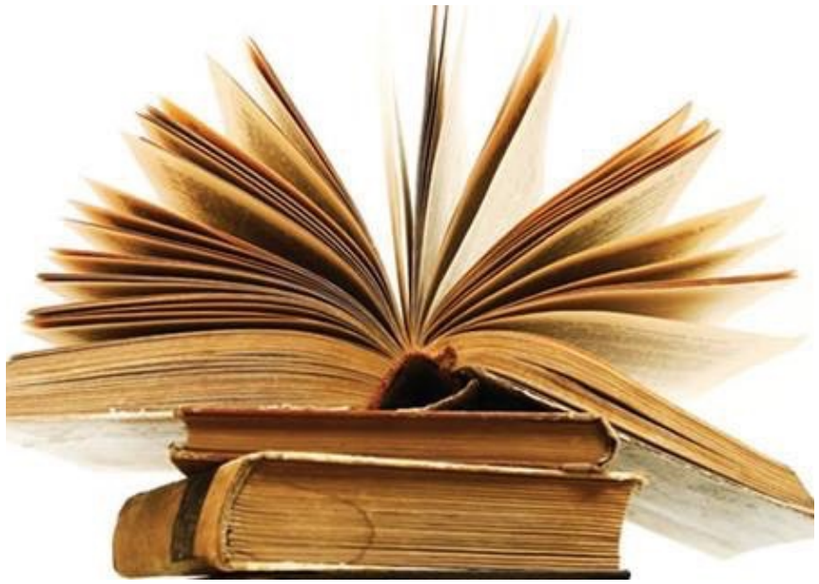
  I. HTML TEMPLATES
  II. HTML IMPORTS
  III. CUSTOM ELEMENTS
  IV. SHADOW DOM

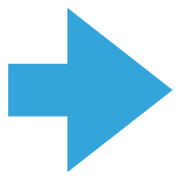
- COMPATIBILIDAD DE WEB COMPONENTS EN LOS DIFERENTES NAVEGADORES

EL DESARROLLO WEB A GRAN ESCALA ANTES DE LOS WEB COMPONENTS

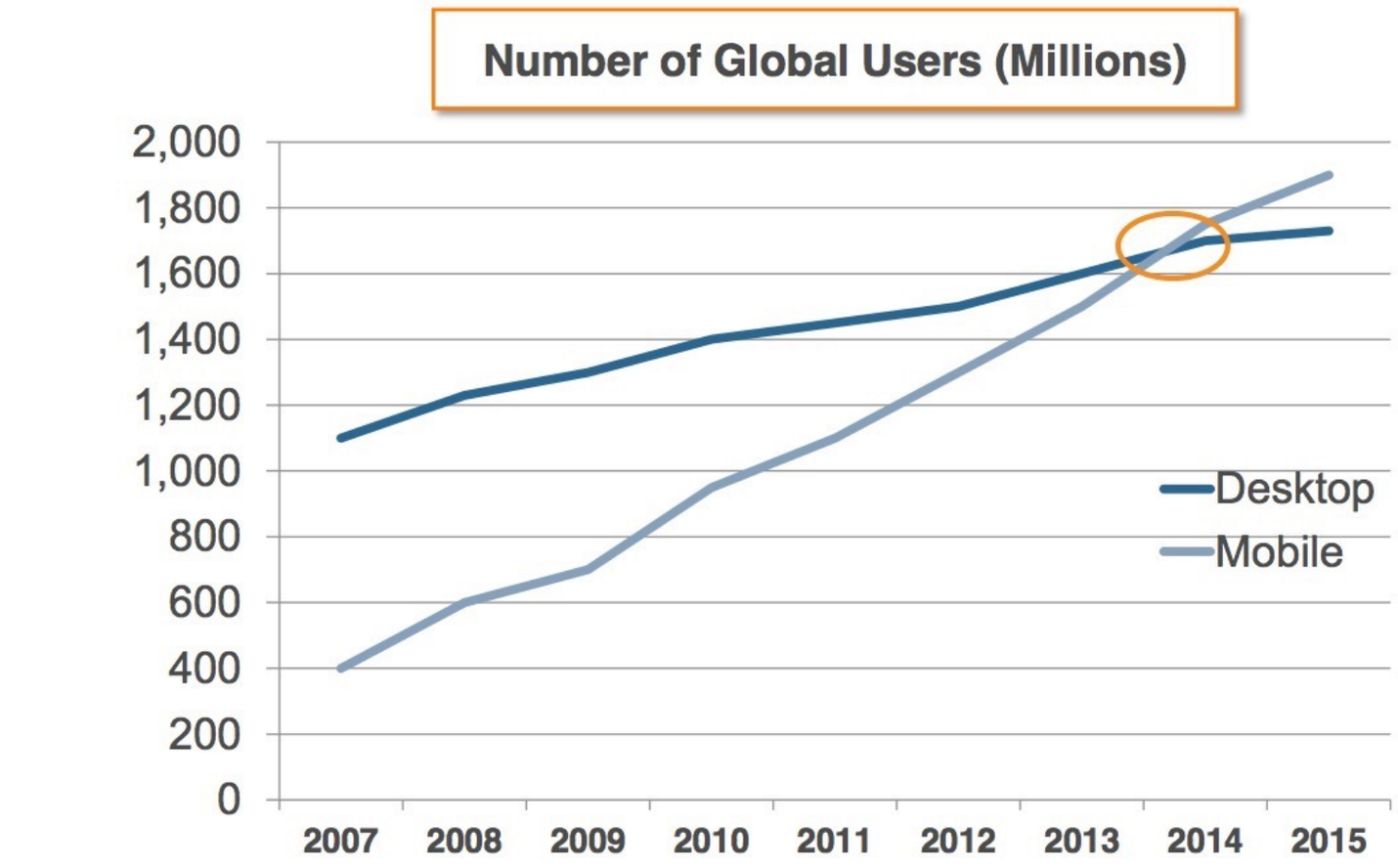


DE LA WEB COMO REPOSITORIO DE DOCUMENTOS A LA WEB COMO SERVIDOR DE APLICACIONES





APLICACIONES NATIVAS FRENTE A APLICACIONES WEB, UNA BATALLA DESIGUAL.



comScore

 24 Source: Morgan Stanley Research

CONCLUSIÓN, SE NECESITABA UNA NUEVA METODOLOGÍA DE DESARROLLO FRONT QUE PERMITA:

1. Una mayor interoperabilidad
2. Mayor accesibilidad
3. Mas estabilidad
4. Procesos de control de calidad mas refinados

Por ello, el W3C crea el **Extensible Web Group** en 2013, cuyo objetivo es el de resolver todos estos problemas permitiendo la implementación de nuevas features en los navegadores sin la necesidad de que la especificación sea cerrada, mediante el uso de polyfills.

EL DISEÑO MODULAR COMO RESPUESTA

- En el intento de añadir estructura y eficiencia en el desarrollo de aplicaciones web, se pone el foco en el modelo de diseño modular.

- La ISO/IEC/IEEE define la modularidad como “el grado en el que un sistema o programa se compone de elementos discretos de tal modo que un cambio en un elemento tiene un impacto mínimo en los demás”.

- El objetivo de la modularidad es que los módulos sean extensibles, reutilizables y mantenibles.

LIMITES DE LA WEB

- **Acoplamiento y separación de ámbitos**
  Cada sección de una aplicación debe ser responsable de una sola cosa y no debe contener código que maneje otras cosas. con ello encapsulamos la funcionalidad y reducimos la complejidad

- **Conflicto entre CSS**
  En CSS los selectores son globales.
  Posibles conflictos por nomenclatura: aunque sistemas como OCSS o BEM intentan normalizar la sintaxis, la posibilidad de no ser consistentes sigue existiendo.

- **HTML extensible**
  Limitación de elementos HTML: no se puede definir una API única para los elementos existentes ni extenderlos.

QUE SON LOS WEB COMPONENTS:

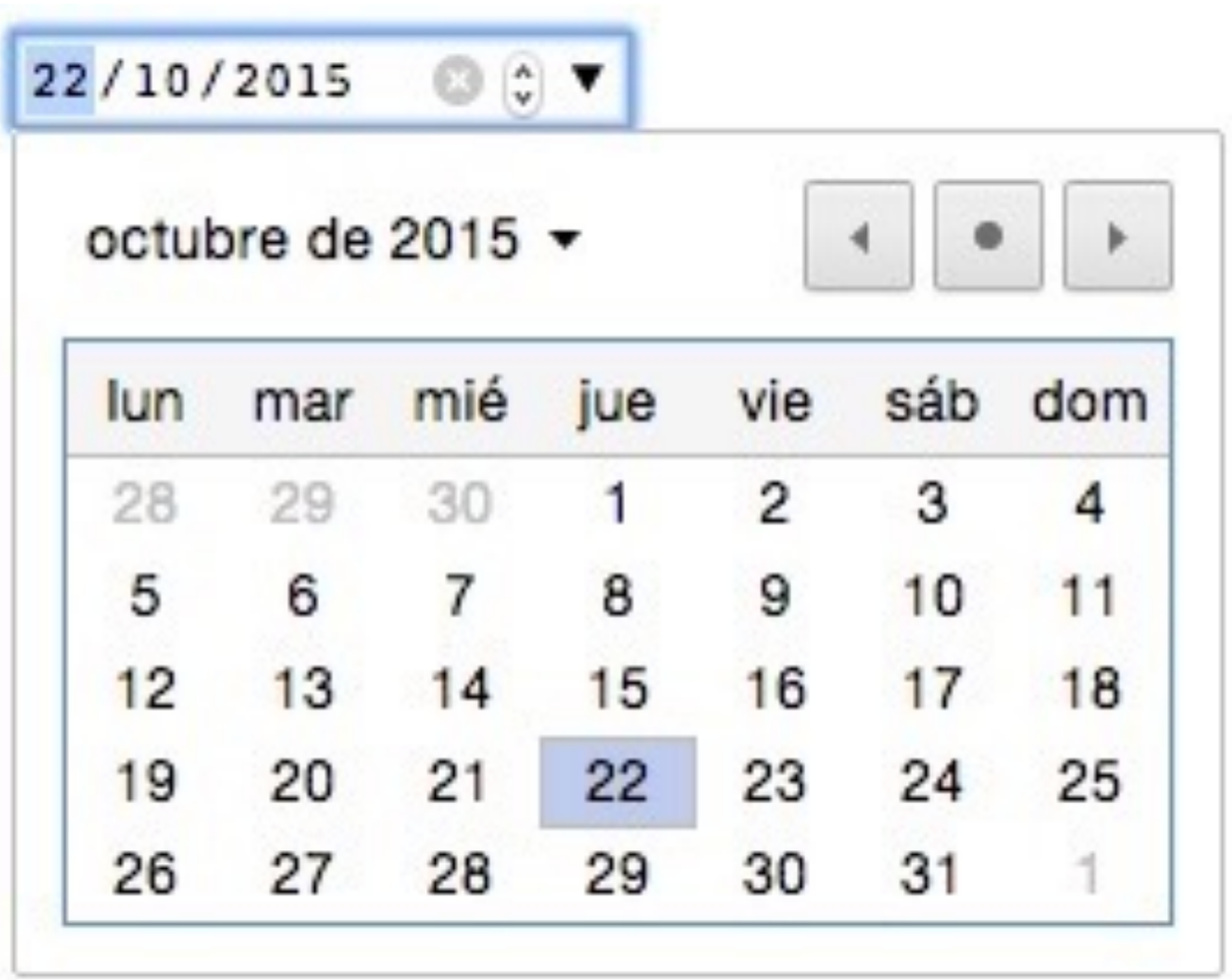
- Son un conjunto de tecnologías complementarias para encapsular HTML, CSS y JS en forma de paquetes reutilizables y con soporte nativo en los navegadores.

- El término lo acuña Microsoft para referirse a sus add-ons para Office. Posteriormente Sun lo utilizó para nombrar a sus servlets de java. Es desde 2011 cuando el término se usa para referirse a los nuevos estándares de la W3C

- Google asume el liderazgo en el desarrollo de los Web components desde la primera revisión del W3C en 2012.

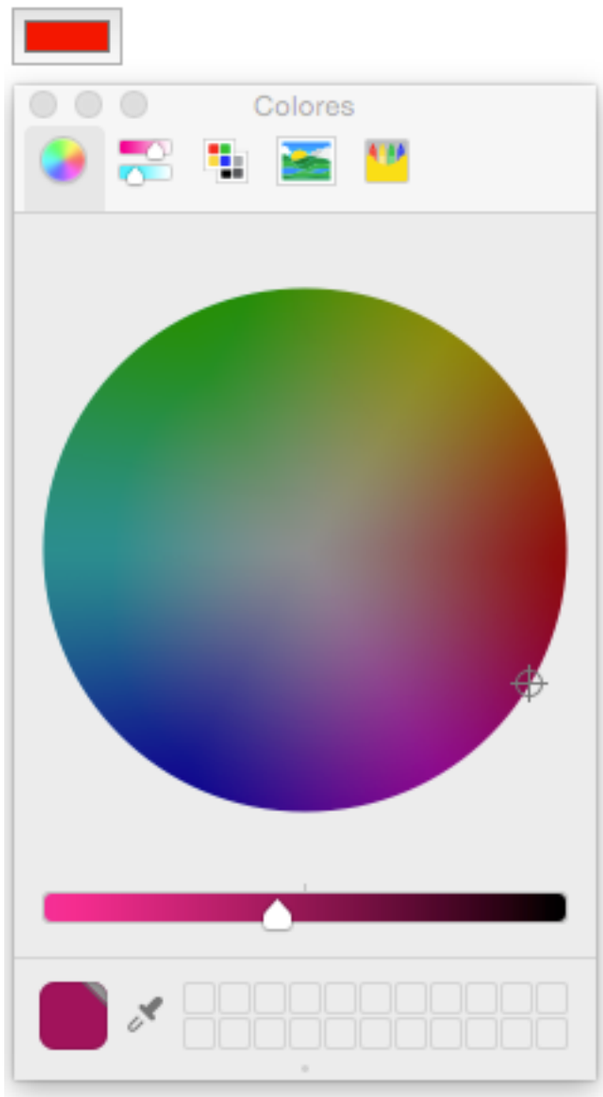
EJEMPLO NATIVO DE CUSTOM ELEMENT: <INPUT TYPE="DATE" />

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



EJEMPLO NATIVO DE CUSTOM ELEMENT: <INPUT TYPE="COLOR" />

<input type="color" value="#FF0000" />



TECNOLOGÍAS DE WEB COMPONENTS: HTML TEMPLATES (I)

- Una plantilla es un documento o archivo con un formato predefinido que se usa como punto de partida para una aplicación de tal modo que ese formato no tenga que ser modificado cada vez que se usa.

- La gran ventaja es que los elementos utilizados en la plantilla no se procesan hasta que esta no se invoca, lo que reduce las peticiones de inicio.

TECNOLOGÍAS DE WEB COMPONENTS: HTML TEMPLATES (II)

Primero, declaramos el template en nuestro HTML:

```
<template id="mytemplate"><img src="" /></template>
```

Luego, seleccionamos la plantilla e inyectamos el contenido:

```
var myNewTemplate =
document.querySelector('#mytemplate');
myNewTemplate.content.querySelector('img').src = 'logo.png'
```

TECNOLOGÍAS DE WEB COMPONENTS: HTML TEMPLATES (III)

Finalmente, clonamos o importamos el contenido en el documento:

```
var templateContent = myNewTemplate.content;
var activeContent = templateContent.cloneNode(true); //
usando cloneNode
var activeContent = document.importNode(templateContent,
true); //usando importNode
container.appendChild(activeContent);
```

TECNOLOGÍAS DE WEB COMPONENTS: CUSTOM ELEMENTS (I)

- Permiten crear nuevas etiquetas HTML y extender las existentes, facilitando un modo común de encapsular funcionalidad de un modo declarativo y almacenable en una etiqueta reusable con su propio ciclo de vida.

El registro de un custom element se efectúa de la siguiente manera, siendo mandatorio que el nombre del custom element no lleve guiones bajos y sí al menos un guión medio.

var customElement = document.registerElement('my-custom-element');

TECNOLOGÍAS DE WEB COMPONENTS: CUSTOM ELEMENTS (II)

Tras el registro, necesitamos añadir comportamiento al custom element por lo que creamos un objeto prototipo:

```
var newCustomPrototype = Object.create(HTMLElement.prototype);
```

Si lo deseamos, podemos añadir propiedades al prototipo en un objeto como segundo parámetro del método create:

```
var newCustomPrototype = Object.create(HTMLElement.prototype,
   bar: {
      get: function ()  { return 1; }
   },
   foo: {
      value: function ()  { alert ( 'foo() llamada' ); }
   }
)
```

Si no queremos crear las propiedades por parámetro, lo podemos hacer dinámicamente:

```
var newCustomPrototype = Object.create(HTMLElement.prototype);
newCustomPrototype.foo = function () { alert ( 'foo() llamada' ); }
Object.defineProperty(newCustomPrototype, 'bar', {value: 1});
```

TECNOLOGÍAS DE WEB COMPONENTS: CUSTOM ELEMENTS (III)

Una vez que la API para nuestro custom element está definida en el objeto prototipo, puede ser registrada mediante el método document.registerElement():

```
document.registerElement('foo', { prototype: newCustomPrototype });
```

También podemos extender elementos HTML nativos añadiendo la clave extends al segundo parámetro del método registerElement():

```
var superInput = document.registerElement('super-input', {
    prototype: Object.create(HTMLInputElement.prototype),
    extends: 'input'
})
```

El código anterior ilustra como el custom element <super-input> extiende al nativo <input> heredando todas sus propiedades, roles, estados y comportamientos. Puede ser instanciado declarativamente usando el atributo is:

```
<input is="super-input">
```

TECNOLOGÍAS DE WEB COMPONENTS: CUSTOM ELEMENTS (IV)

Se definen funciones callback para los diferentes estados del ciclo de vida del custom element. Son las **lifecycle callbacks**:

createdCallback() - Invocada una vez la instancia del elemento es creada
attachedCallback() - Invocada cuando el elemento es insertado en el DOM
detachedCallback() - Invocada cuando el elemento se elimina del DOM
attributeChangedCallback(<nombre>,<antiguoValor>,<nuevoValor) - Invocada cuando añadimos, cambiamos o eliminamos un atributo a un elemento

Para usar estos callbacks, definiremos las funciones como propiedades del objeto prototipo:

```
var newCustomPrototype = Object.create(HTMLElement.prototype);
newCustomPrototype.createdCallback = function() {…};
newCustomPrototype.attachedCallback = function() {…};
```

TECNOLOGÍAS DE WEB COMPONENTS: SHADOW DOM

El modelo DOM HTML se construye como un árbol (tree) de objetos llamados nodos. Estos nodos permiten tres tipos de subtrees:

Document Tree - el DOM normal cuyo nodo principal es un documento y es visible por el usuario
Shadow Tree - el DOM interno, no visible por el usuario
Composed Tree - lo que el navegador renderiza

Cualquier nodo del document puede alojar (host) uno o mas shadow trees. Estos nodos host son los **shadow hosts**.

TECNOLOGÍAS DE WEB COMPONENTS: SHADOW DOM (II)

Un subtree shadow DOM debe ser siempre adjuntado a un elemento host existente, pudiendo ser este un elemento HTML declarativo, un elemento imperativo creado por javascript o incluso un custom element. El método Element.createShadowRoot(<hostElement>) se usa para crear el shadow root encapsulado y el árbol shadow DOM adjuntado al elemento host:

```
<button>Hola!</button>
<script>
   var host = document.querySelector('button');
   var root = host.createShadowRoot();
<script>
```

Este ejemplo añade un shadow DOM al elemento <button>, que contiene el texto 'Hola'. Ahora, mediante javascript podemos añadir o eliminar contenido del shadow root de la siguiente manera:

```
root.textContent = 'Adios!';
```

TECNOLOGÍAS DE WEB COMPONENTS: SHADOW DOM (III)

Distribución es el mecanismo que determina qué nodos dentro del shadow host son proyectados dentro de los puntos de inserción en el shadow DOM del host.

Los **eventos** en el shadow DOM se comportan de distinto modo que en el document tree, reorientándose para que parezca que vienen del shadow Host, lo que previene cualquier error por encapsulación.

Los elementos en el shadow DOM pueden ser estilados interna o externamente. Los estilos definidos internamente mediante el pseudoelemento :host se encapsulan y no afectan a ningún elemento del exterior. Usando el pseudoelemento ::shadow podemos modificar desde el exterior el estilo de cualquier elemento definido dentro del shadow DOM.

TECNOLOGÍAS DE WEB COMPONENTS: HTML IMPORTS

Aunque se suele importar contenido declarativamente (<script>, <img>,<link> para css), para HTML hay diferentes maneras (AJAX, iFrames) muchas veces complejas e ineficientes.

HTML Imports mejoran el modelo permitiendo reutilizar documentos HTML en otros documentos HTML, incluyendo todos los recursos utilizados en cada documento en esa importación, lo que favorece la modularidad.

Importamos HTML usando el elemento <link> del mismo modo que hacemos con una hoja de estilos, pero usando el valor import en el atributo rel en vez de stylesheet:

<link rel="import" href="ruta/a/archivo.html">

TECNOLOGÍAS DE WEB COMPONENTS: HTML IMPORTS (II)

Parseo

El elemento link no bloquea el parseo del archivo importador, pero si bloquea el renderizado, dado que todo estilo dentro del archivo importado se aplica automáticamente sobre el documento principal para evitar FOUC (Flash of unstyled content). Para hacer completamente asíncrona la importación, usar el atributo async en la etiqueta link donde declaramos la importación.

Acceso

Todo lo que no sean scripts o estilos en un documento importado se referencia desde el importador, no insertándose. Se puede acceder al contenido de un documento importado mediante javascript:

```
var link = document.querySelector('link[rel="import"]');
var content = link.import;
```

TECNOLOGÍAS DE WEB COMPONENTS: HTML IMPORTS (III)

Eventos

El elemento link soporta dos manejadores de eventos: load y error, dando la posibilidad de definir callbacks en el atributo onload y onerror.

Rendimiento

Si diferentes componentes requieren una misma librería, mediante el de-dupling sólo se obtiene el mismo archivo requerido una vez, aunque el recurso debe solicitarse en la misma URL para que se aplique.

Debe evitarse el exceso de peticiones de recursos para los componentes porque afecta al rendimiento; Polymer ha creado la herramienta Vulcanize para combinar los recursos importados en un único recurso. La adopción del protocolo HTTP/2 hará que esto deje de ser un problema.

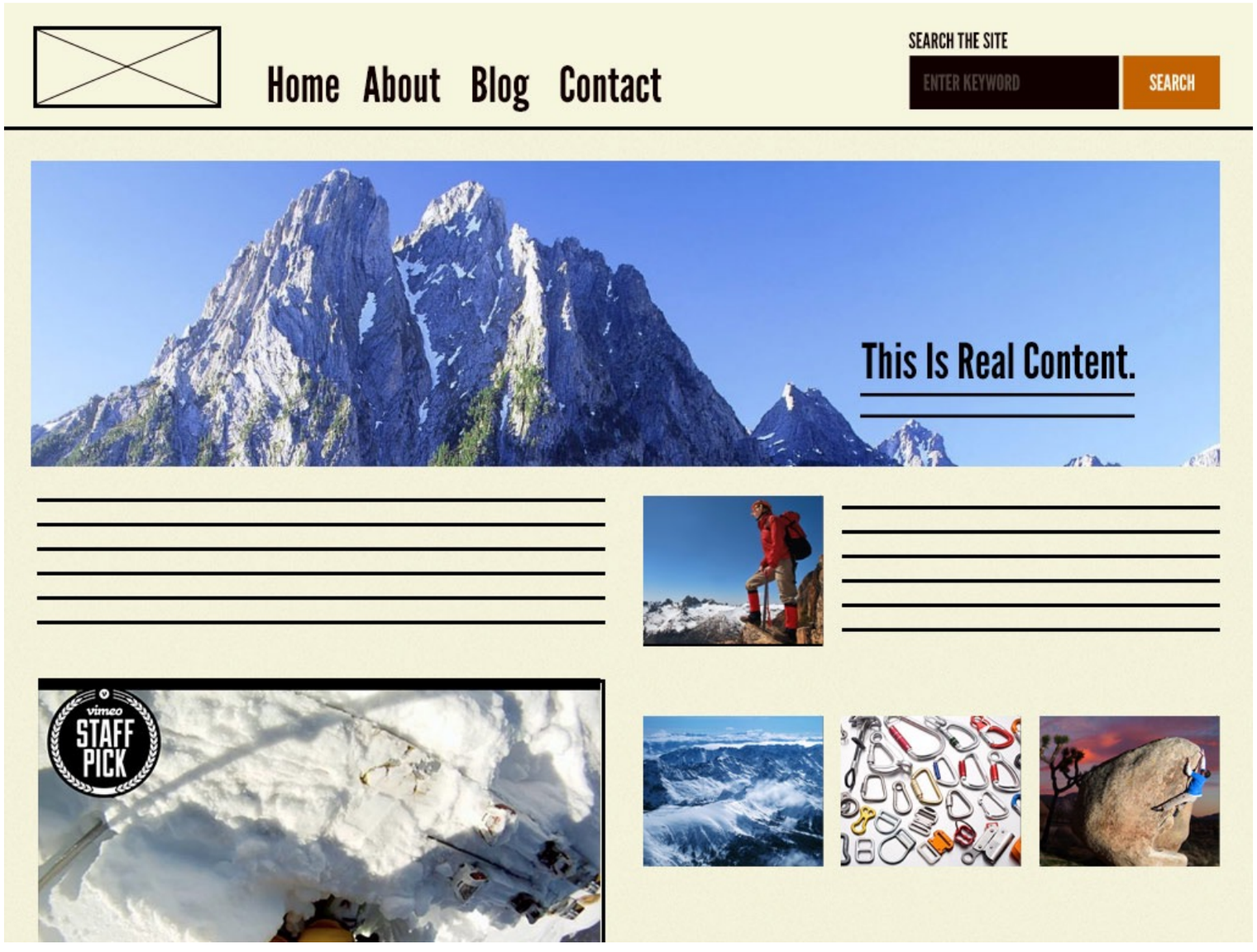
PROPUESTA DE ABSTRACCIÓN DE ELEMENTOS UI: ATOMIC DESIGN

- INTERFACES, DEL TODO A LAS PARTES
- QUÉ ES ATOMIC DESIGN
- ELEMENTOS QUE COMPONEN ATOMIC DESIGN:

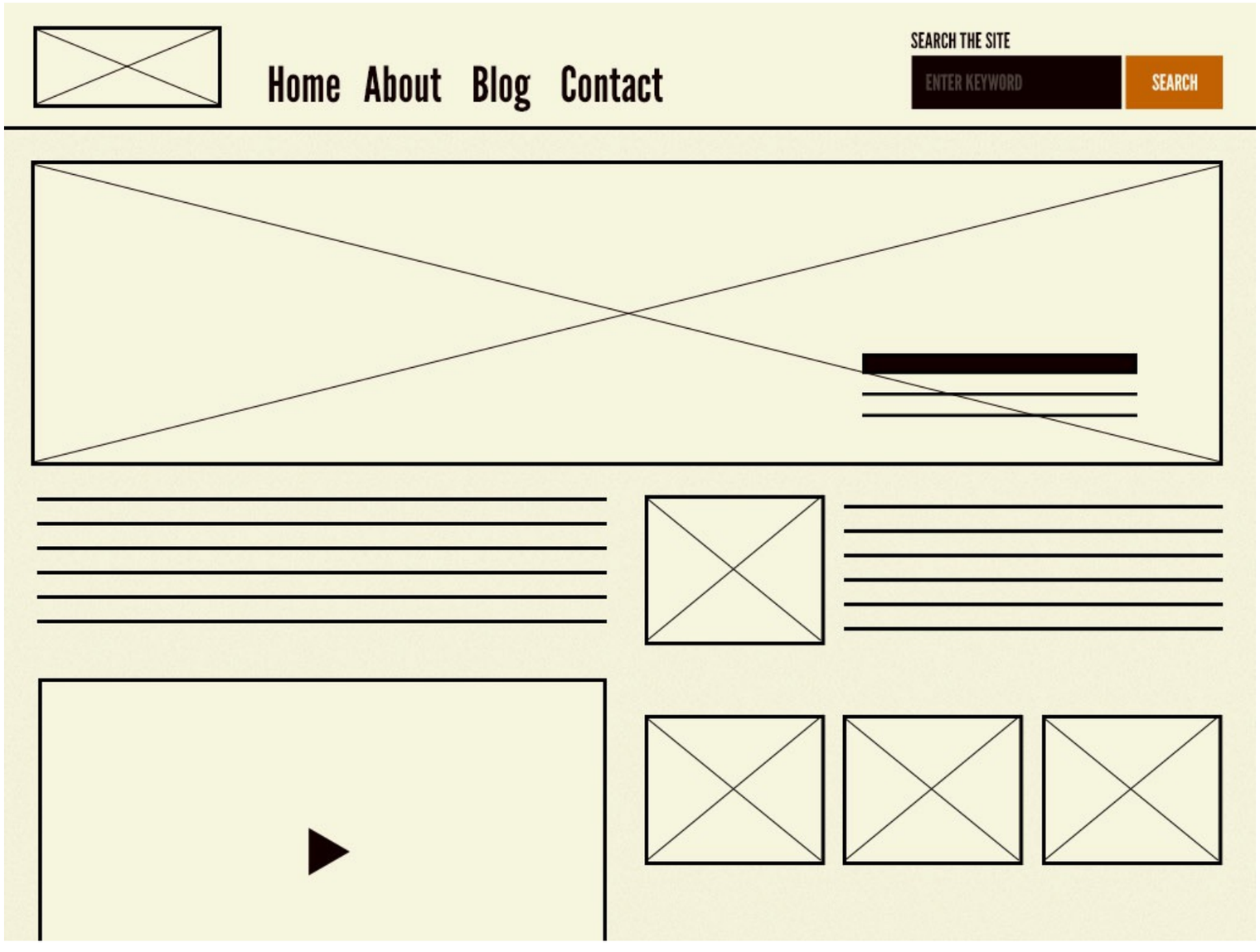
  I. ÁTOMOS
  II. MOLÉCULAS
  III. ORGANISMOS
  IV. PLANTILLAS (TEMPLATES)
  V. PÁGINAS

- PATTERN LAB, PROPUESTA DE UN SISTEMA ATÓMICO DE DISEÑO

INTERFACES, DEL TODO A LAS PARTES

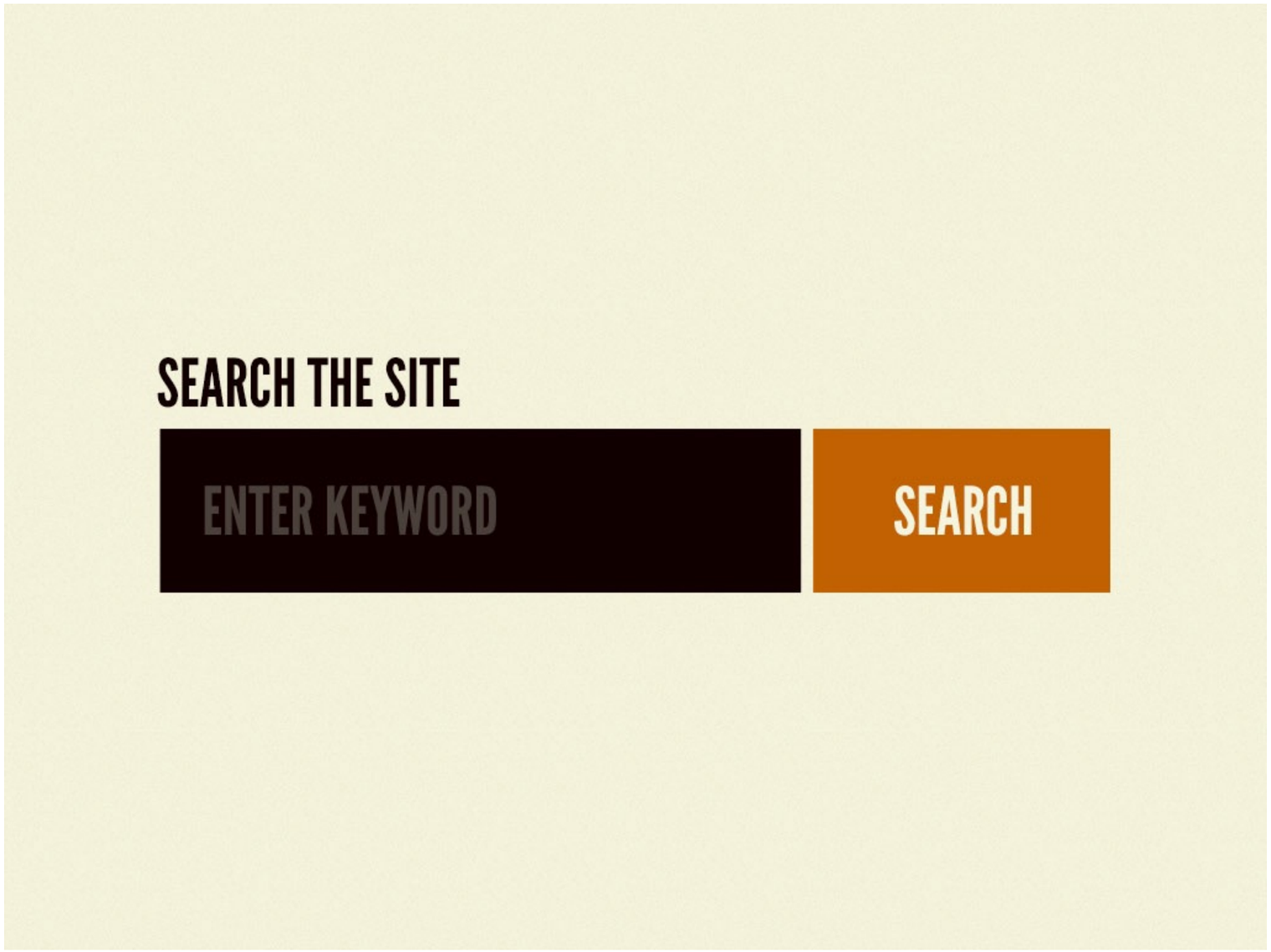


INTERFACES, DEL TODO A LAS PARTES (II)

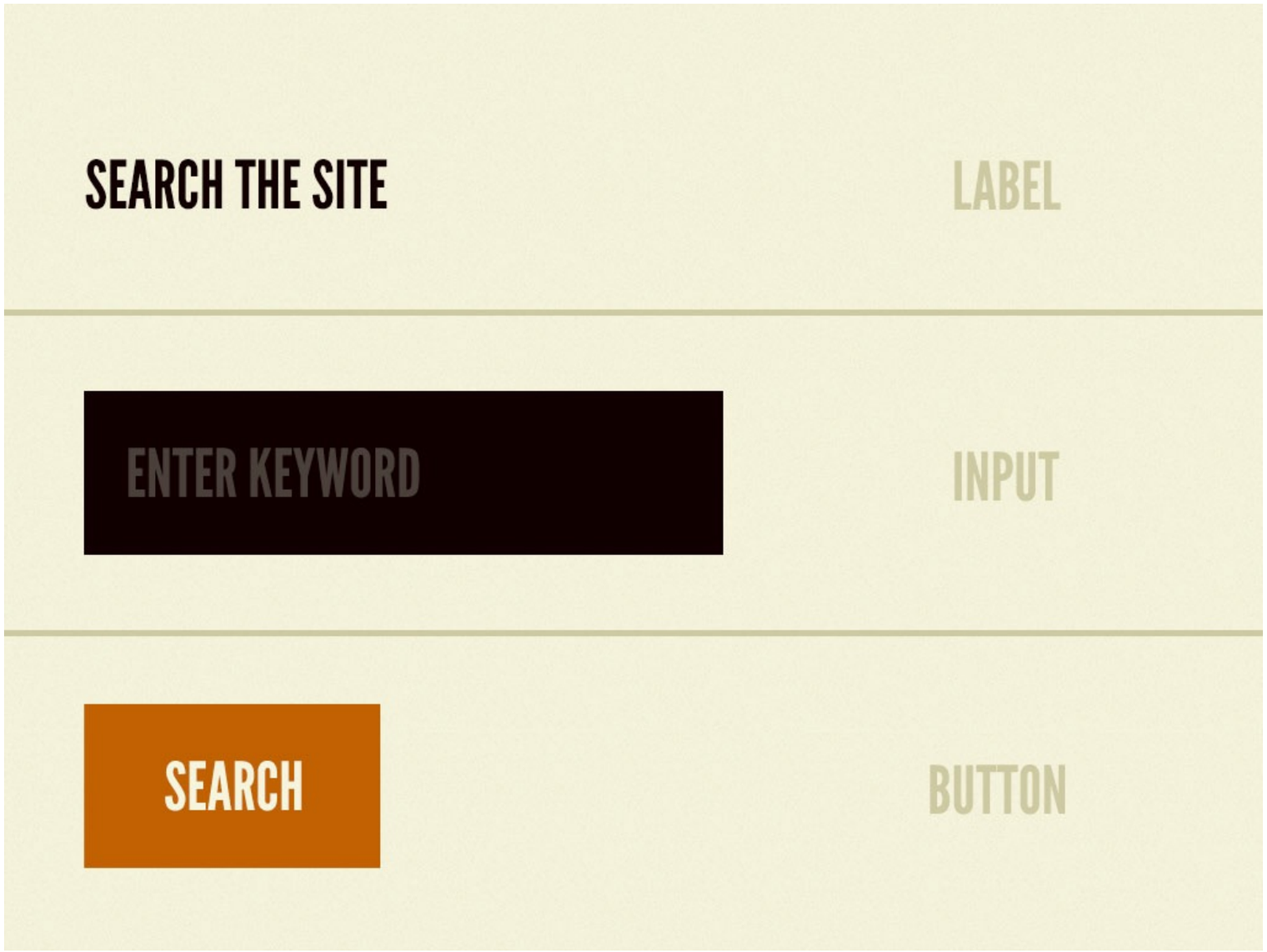


INTERFACES, DEL TODO A LAS PARTES (III)

INTERFACES, DEL TODO A LAS PARTES (IV)



INTERFACES, DEL TODO A LAS PARTES (V)



QUE ES ATOMIC DESIGN

We are not longer designing pages, we are designing systems of components. — @stephenhay

QUE ES ATOMIC DESIGN

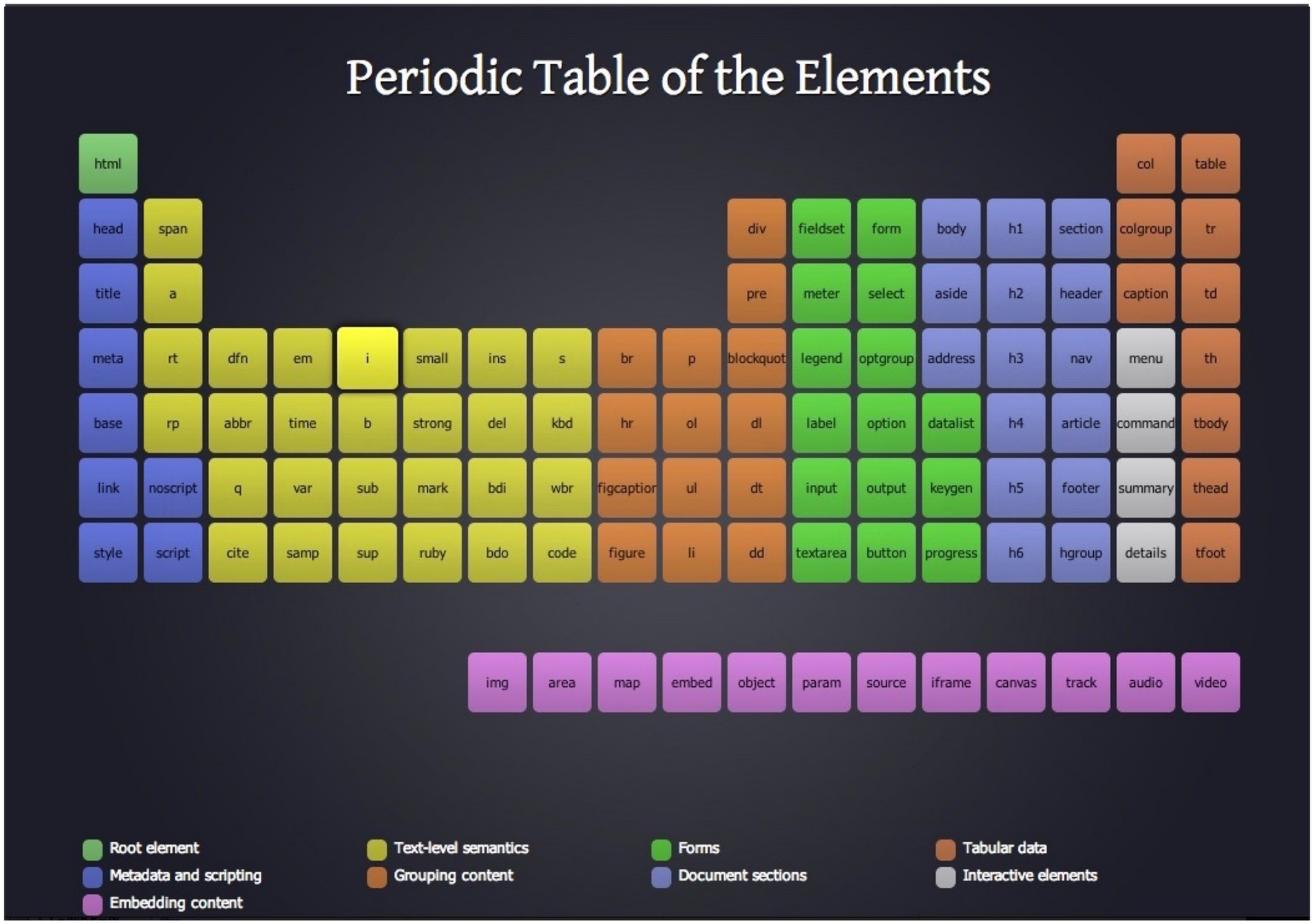
We are no longer ~~designing~~ developing pages, we are ~~designing~~ developing systems of components

QUE ES ATOMIC DESIGN

Antes que definir sistemas de diseño basados en colores, tipografías, grids, etc, se buscaba formular un sistema de diseño basado en los elementos que componen una interface.

Inspirados en la química, observaron que toda realidad se compone de átomos, cuya unión forma moléculas, que unidas componen organismos. De este modo, las interfaces se componen de componentes fundamentales en los que se puede trabajar de manera individual.

QUE ES ATOMIC DESIGN

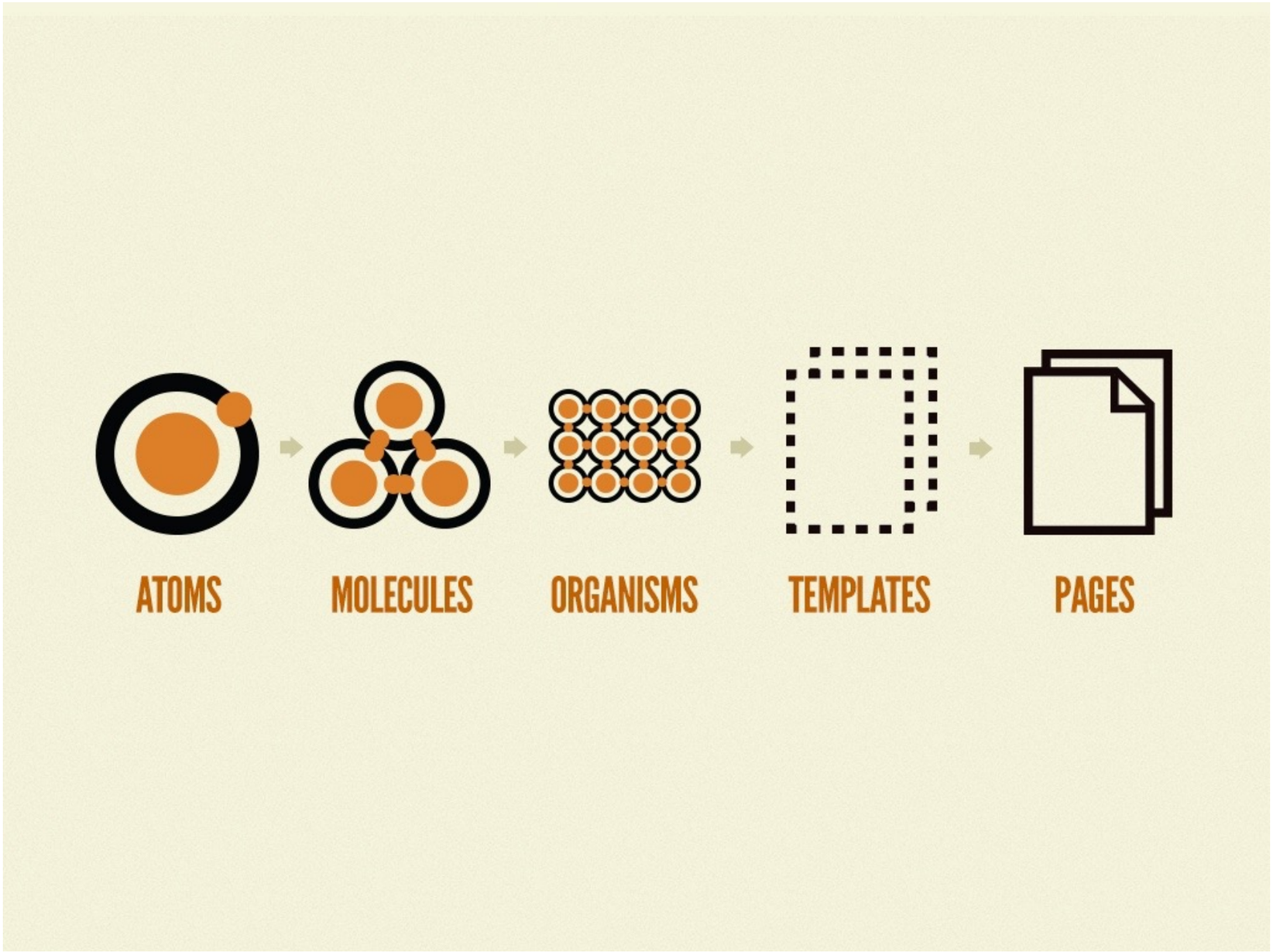


QUE ES ATOMIC DESIGN

Atomic design es una metodología para crear sistemas de diseño, estructurada en 5 niveles:

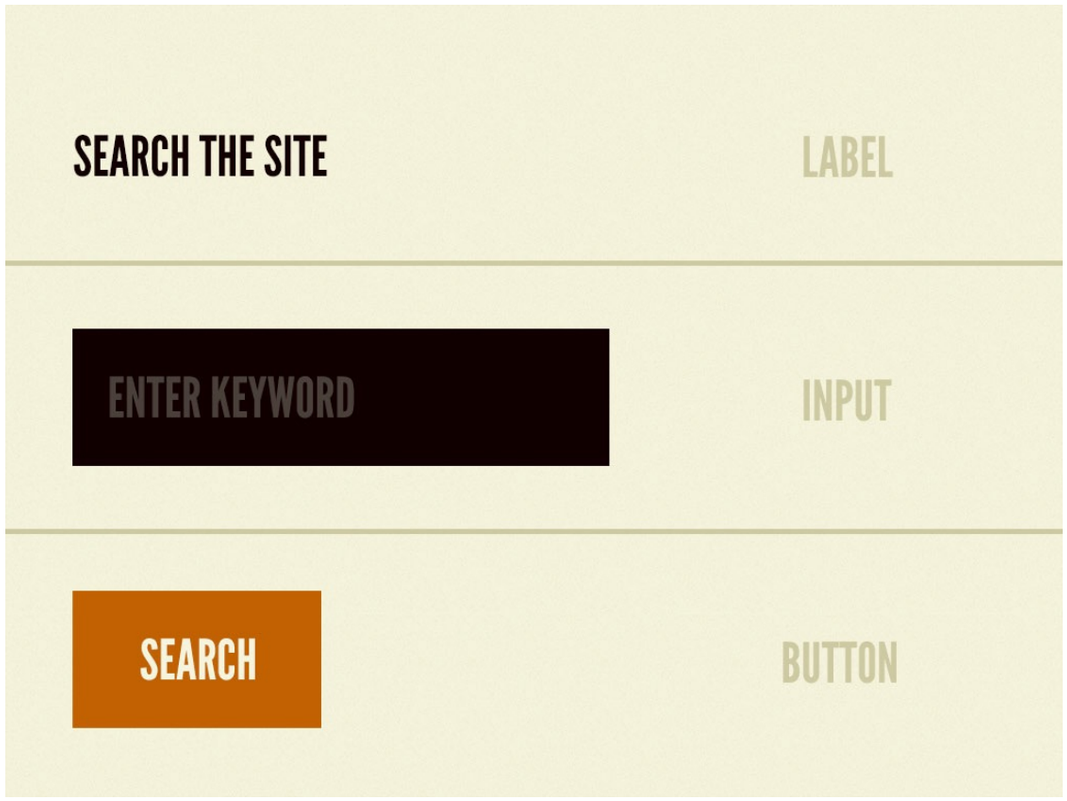
1. Átomos
2. Moléculas
3. Organismos
4. Plantillas
5. Páginas

QUE ES ATOMIC DESIGN



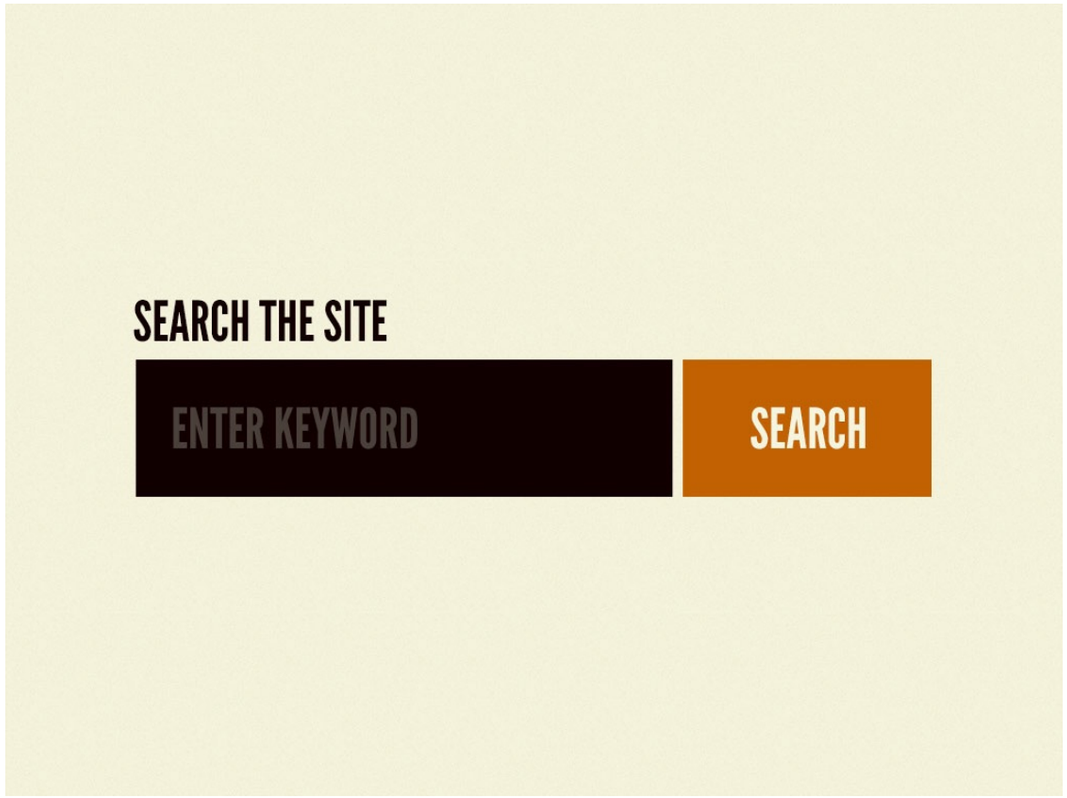
QUE ES ATOMIC DESIGN: ÁTOMO

El átomo es el elemento mas básico en la formación de la materia. Aplicado a una interface, sería una etiqueta HTML, como un input, un button o un label.



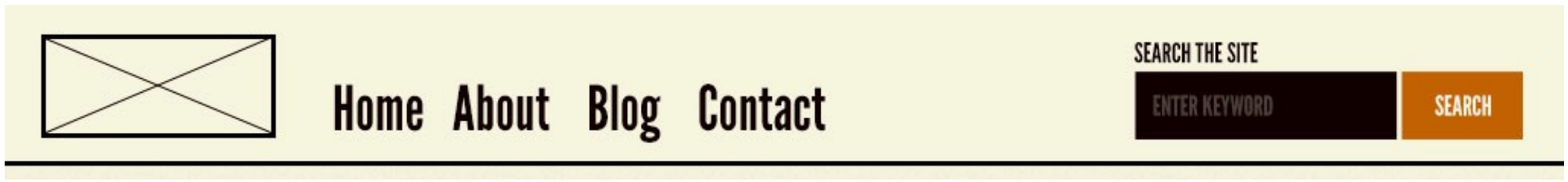
QUE ES ATOMIC DESIGN: MOLÉCULA

Las moléculas son grupos de átomos con entidad propia y forman el esqueleto de nuestro interface.



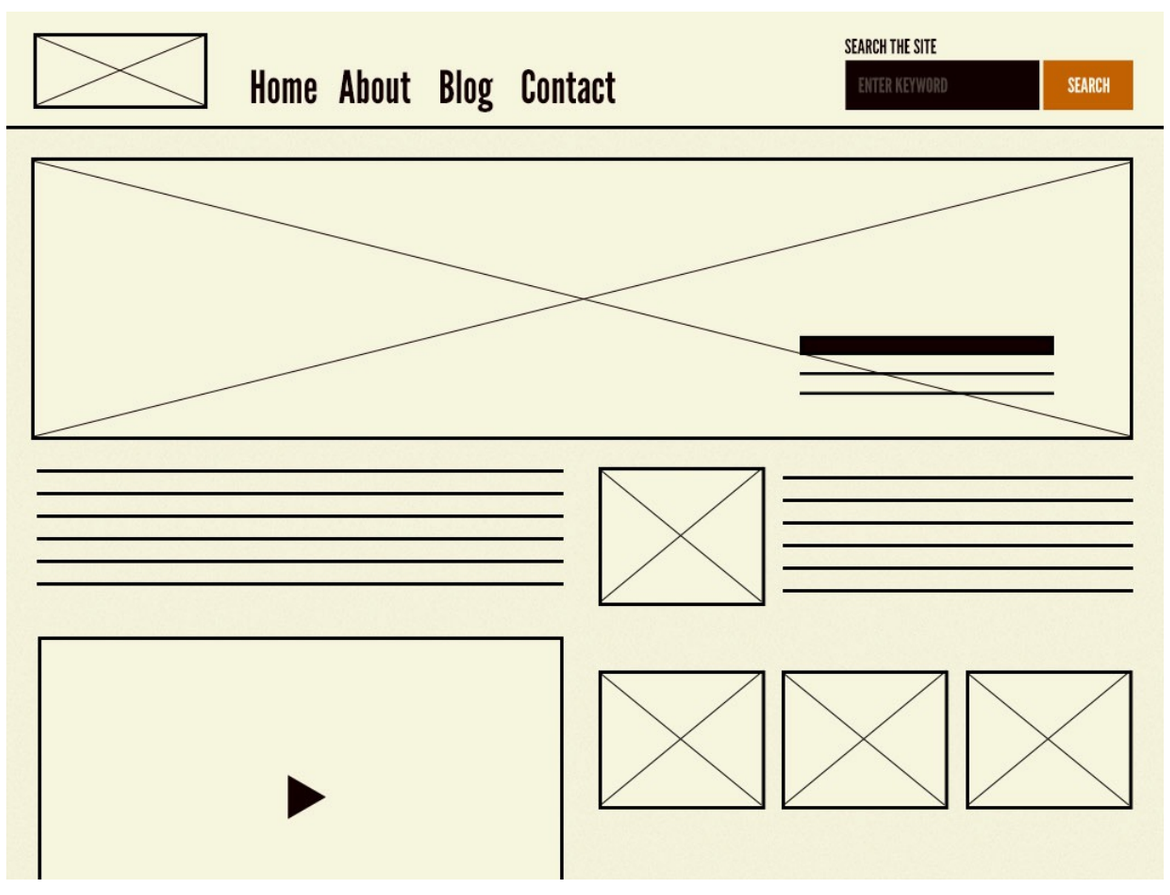
QUE ES ATOMIC DESIGN: ORGANISMO

Los organismos son grupos de moléculas combinados, formando una sección de un interface. Se pueden considerar como componentes encapsulados, portables y reutilizables.



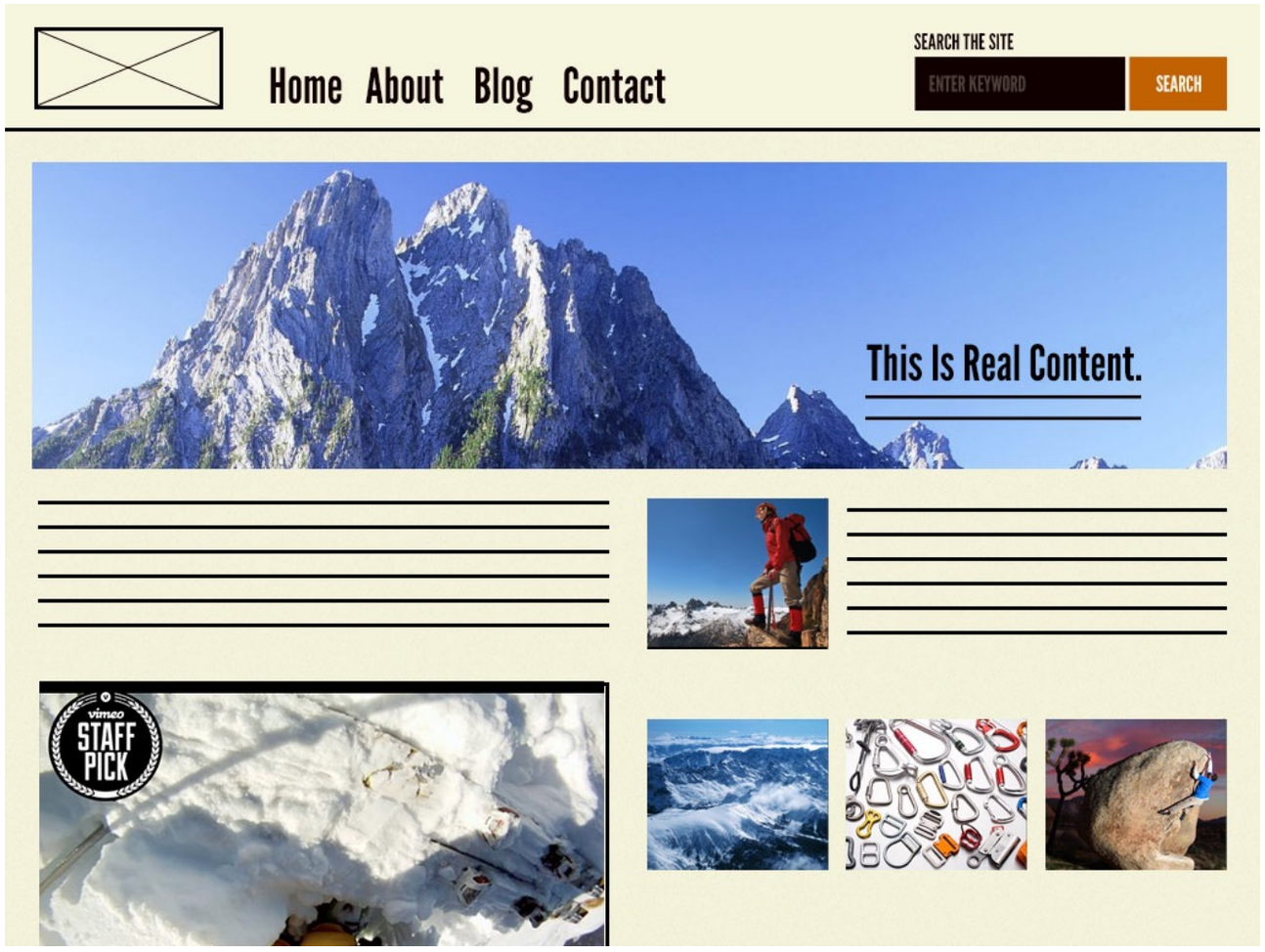
QUE ES ATOMIC DESIGN: TEMPLATE

Un template es un conjunto de organismos que forman una página, pero sin incluir aún el contenido final. En este punto ya percibimos el layout final.



QUE ES ATOMIC DESIGN: PÁGINA

La página es una instancia del template con contenido real.



PATTERN LAB: PROPUESTA DE UN SISTEMA ATÓMICO DE DISEÑO

Los creadores de Atomic design desarrollaron una serie de herramientas para asistir a los programadores en la definición de sistemas atómicos de diseño.

Para fijar esta conceptualización, revisar su demo en:

http://demo.patternlab.io

POLYMER ELEMENT CATALOG: EL PATTERN PORTFOLIO DE POLYMER

Polymer ha puesto a disposición de la comunidad una herramienta que permite compartir de manera agrupada todos los elementos que sus creadores desarrollan.

Aquí la agrupación es por funcionalidad, independientemente del “tamaño” de los elementos.

http://elements.polymer-project.org

COMPATIBILIDAD

	Edge	Internet Explorer	Chrome	Firefox	Safari	Opera
Templates	⚠	✖	✔ 31+	✔ 38+	✔ 7.1+	✔ 30+
Custom Elements	⚠	✖	✔ 42+	🚩	✖	✔ 30+
Shadow DOM	⚠	✖	✔ 31+	🚩	✖	✔ 30+
HTML Imports	⚠	✖	✔ 42+	🚩	✖	✔ 30+

⚠ In development or under consideration

🚩 Enabled with flag

RESUMIENDO...

1. El uso de Web components nos facilitará el desarrollo de aplicaciones de gran escala gracias a su carácter modular y extensible.

2. La aplicación de una estrategia de desarrollo como la descrita en el modelo Atomic design nos facilitará la creación de componentes.

3. La declaración de las primitivas de Web components es farragosa, y aquí es donde Polymer interviene facilitando la creación de componentes web.

- QUE ES POLYMER
  - COMO SE LLEGA A POLYMER: EL DOM ES EL FRAMEWORK
  - QUE ES EXACTAMENTE POLYMER
  - PENSAR EL COMPONENTE DE MANERA LOCAL
  - LEVERAGE COMPOSITION
  - INTEGRAR EL COMPONENTE EN SU ENTORNO MEDIANTE EL PATRÓN MEDIADOR

DOM IS THE FRAMEWORK

- Observemos el dom con sus conceptos de framework y usemos sus partes mas interesantes, sin inventar cosas:
  - **component model** ej: <*video*> --> elemento nativo. No es un componente sencillo
  - **flujo de datos**: atributos y propiedades
    Ej: src='video.mp4', controls, eventos de pausa, stop...etc
  - **html** --> sin código, por detrás se ejecutan muchas operaciones

<VIDEO SRC="VIDEO.MP4" CONTROLS/>

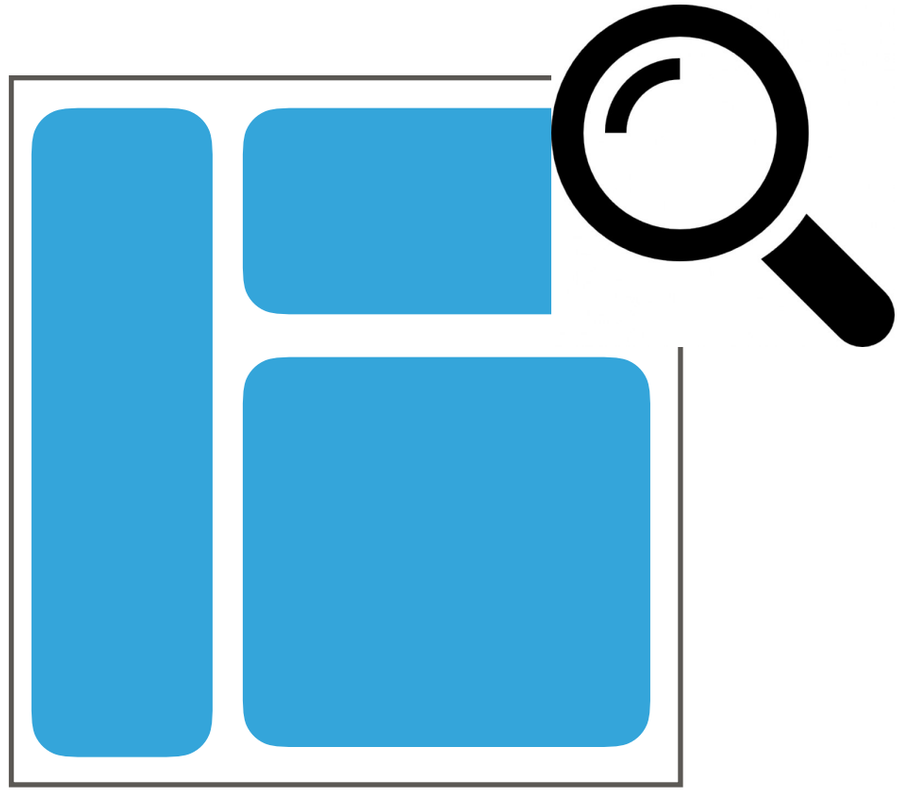
DOM era impenetrable -> ¡custom elements!
aquí es donde polymer interviene en ayudar a crear esos custom elements

3 CONCEPTOS BÁSICOS

- Conceptos que intervienen en la composición de una app con los distintos componentes.

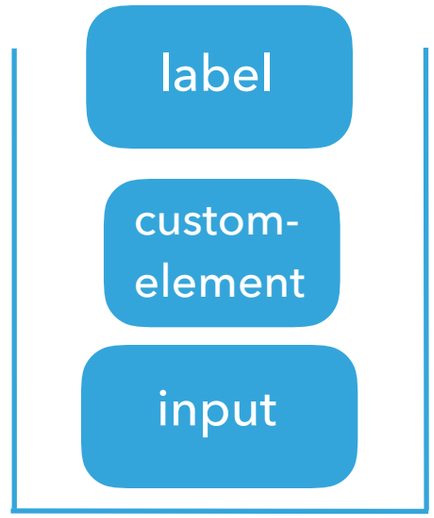
1. Thinking locally: dividiremos la app en pequeñas piezas y nos centraremos en cada una de esas piezas
2. Leveraging composition: pensamos en una pieza y observamos los elementos que tiene nuestra pieza y su composición
3. Mediator pattern: Arquitectura donde interviene un elemento padre que se encarga de la comunicación de los distintos elemento que se vayan componiendo

THINKING LOCALLY



- No visualizaremos toda la app a la vez sino en distintas piezas que lo componen.
- Nos centraremos en ese elemento, con una funcionalidad propia, sin necesidad de pensar el el resto de la app, reduciendo la complejidad de la misma

COMPOSITION

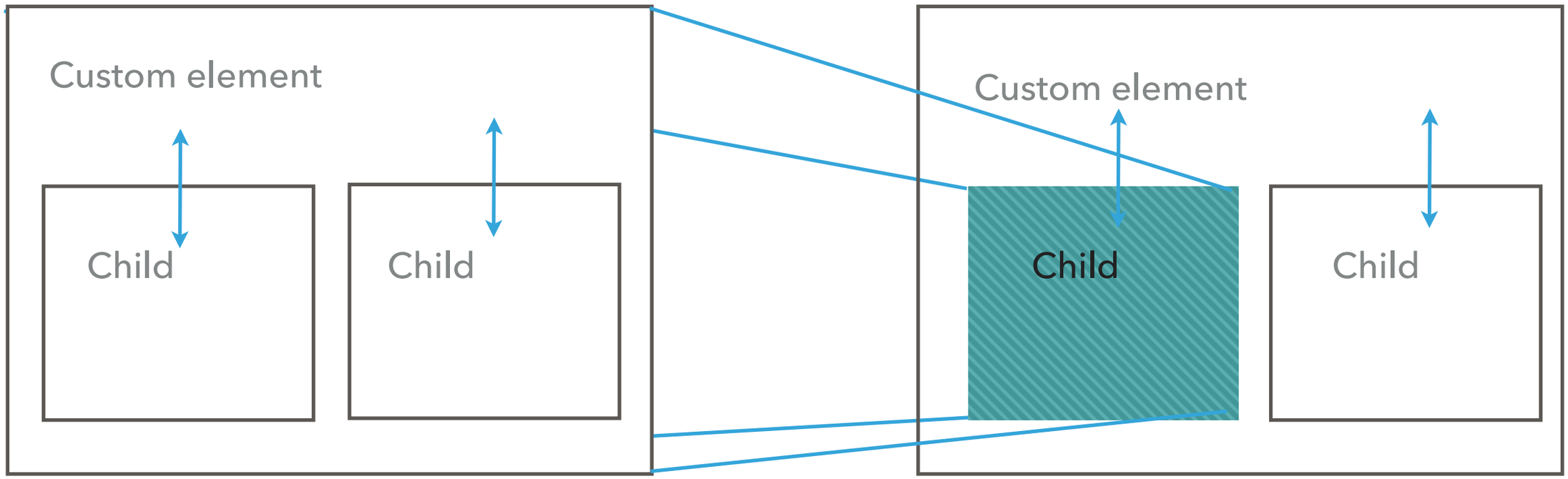


- Una vez nos hemos centrado en un elemento, podemos ver que puede tener mas elementos dentro, otros componentes u otros elementos nativos de html
- Con esos distintos elementos compondremos el nuevo elemento

PATRON MEDIADOR

- La esencia de este patrón sería un objeto que encapsula la forma de interactuar de distintos objetos. Así los objetos no se comunican entre si sino que usan este mediador
- Nuestro elemento padre, será el encargado de mediar entre los componentes hijos mediante eventos o data-binding

EN RESUMEN



- CARACTERÍSTICAS DE UN POLYMER
  - CUSTOM ELEMENTS
  - TEMPLATES
  - HTML IMPORTS
  - PROPERTIES
  - DATA BINDINGS (ONE WAY/DOUBLE WAY)
  - DATA BINDING: HELPERS
  - BEHAVIORS
  - LISTENERS
  - CICLO DE VIDA
  - THEMING

CUSTOM ELEMENTS

- "Custom elements" te permite definir tus propios elementos con sus propias etiquetas, asociándoles un código JS y luego usarlos como etiquetas normales de html (div, p, video....)

  Ej: <super-button></super-button>

- Al ser elementos reales, se pueden usar los métodos del DOM para acceder a sus propiedades, escuchar eventos o estilarlos con cualquier otra etiqueta

- **UI elements:**
  Renderizan elementos visuales en la página.
  Ej: <select> o <input>
  Ej. polymer: <paper-checkbox>

- **Non UI:**
  No renderizan nada. Aportan funcionalidad
  Ej: <script>, <style>, and <meta>
  Ej. polymer:<iron-ajax>

TEMPLATES

- El contenido del componente se inserta dentro de la etiqueta <template> para ser renderizado.
- Polymer sólo toma en cuenta la etiqueta template principal, ignorando los subtemplates que podamos definir dentro de ella, y que utilizaremos para mostrar fragmentos vía condiciones o expresiones

```
<template is="dom-if" if="{{value}}"></template>
<template is="dom-repeat" items="{{array}}"></template>
```

POLYMER, ORDEN EN EL CAOS / CARACTERÍSTICAS DE UN POLYMER

mobile-view.html × theme.html ×

```
23  -->
24  <dom-module id="mobile-view">
25
26    <style>
27      :host {
28        font-family: var(--fontDefault);
29      }
30
31      .container {
32        @apply(--layout-vertical);
33      }
34
35      @media (min-width: 600px) {
36        .container {
37          @apply(--layout-horizontal);
38        }
39        mobiles-list, mobile-data {
40          @apply(--layout-flex);
41        }
42      }
43
44    </style>
45
46    <template>
47      <h3>APP</h3>
48      <div class="container">
49        <mobiles-list></mobiles-list>
50        <template is="dom-if" if="{{mobileSelected}}">
51          <mobile-data mobile-detail="{{mobileSelected}}"></mobile-data>
52        </template>
53      </div>
54    </template>
55
56  </dom-module>
57
58  <script>
59
60    Polymer({
61
62      is: 'mobile-view',
63
64      listeners: {
65        "mobileSelected" : "setMobile"
66      },
67
68      properties: {
69        data: {
70          type: Object
71        },
```

Line 45, Column 1 | Spaces: 2 | HTML

HTML IMPORTS

- Mediante los HTML imports declaramos los polymers que vamos a utilizar

```
<link rel="import" href="../polymer/polymer.html">
<link rel="import" href="../iron-icon/iron-icon.html">
```

POLYMER, ORDEN EN EL CAOS / CARACTERÍSTICAS DE UN POLYMER

mobile-view.html

```
<!--
@license
Copyright (c) 2015 The Polymer Project Authors. All rights reserved.
This code may only be used under the BSD style license found at http://polymer.github.io/LICENSE.txt
The complete set of authors may be found at http://polymer.github.io/AUTHORS.txt
The complete set of contributors may be found at http://polymer.github.io/CONTRIBUTORS.txt
Code distributed by Google as part of the polymer project is also
subject to an additional IP rights grant found at http://polymer.github.io/PATENTS.txt
-->
<link rel="import" href="bower_components/polymer/polymer.html">
<link rel="import" href="bower_components/mobiles-list/mobiles-list.html">
<link rel="import" href="bower_components/mobile-data/mobile-data.html">
<link rel="import" href="theme/theme.html">

<!--
An element providing a solution to no problem in particular.

Example:

    <mobile-view></mobile-view>

@demo
-->
<dom-module id="mobile-view">

  <style>
    :host {
      font-family: var(--fontDefault);
    }

    .container {
      @apply(--layout-vertical);
    }

    @media (min-width: 600px) {
      .container {
        @apply(--layout-horizontal);
      }
      mobiles-list, mobile-data {
        @apply(--layout-flex);
      }
    }

  </style>

  <template>
    <h3>APP</h3>
    <div class="container">
      <mobiles-list></mobiles-list>
      <template is="dom-if" if="{{mobileSelected}}">
```

Line 45, Column 1 | Spaces: 2 | HTML

PROPERTIES

- Objeto que define las propiedades de un componente

```
properties: {
  user: String,

  number: {
    type: Number,
    value: 4,
    notify: true
  }
},
```

- **Tipos de keys:**
  - **type**: Constructor: Boolean, Date, Number, String, Array or Object
  - **value**: (boolean, number, string or function) valor por defecto
  - **readOnly**: (boolean) true --> impide el bindeo
  - **notify**: (boolean) true --> bindeo bidireccional
  - **computed**: (string) función que te devuelve un resultado en base a unos argumentos. Cada atributo tiene que encapsularse dentro de una etiqueta HTML
  - **observer**: (string) Escuchador que llama al método definido al modificarse el valor de la propiedad donde se asigna.

DATA BINDING

- Consiste en "unir" una propiedad de un componente a otro del DOM, bien sea el padre o el hijo

- Tenemos dos tipos de bindeos (Binding annotations):
  [[ ]] : bindeo en 1 dirección del elemento host al hijo
  {{ }} : bindeo automático, tanto de host a child como de child a host

- Ej. <child-element name="{{myName}}"></child-element>
  Bindea la propiedad "name" del componente hijo con la propiedad "myName" del host

- Bindeo bidireccional ha de cumplir 3 puntos:
  - Notación {{ }}
  - notify: true para comunicarle el cambio del hijo al padre, si no sólo seria del padre al hijo
  - No poner readOnly: true sino la comunicación sería de hijo a padre

- Expresiones en el bindeo de datos:
  - ! (negación)
  - Computed

```
  <div><span>[[message(isFav)]]</span></div>
  <icon-toggle toggle-icon="favorite" pressed="{{isFav}}"></icon-toggle>
</template>

<script>
  Polymer({
    is: "icon-toggle-demo",
    message: function(fav) {
      if (fav) {
        return "You really like me!";
      } else {
        return "Do you like me?";
      }
    }
  });
</script>
```

- Computed es una función bindeada que devuelve datos:

```
<span>{{computeFullName(first, last)}}</span>
computeFullName: function(first, last) {
 return first + ' ' + last;
}
```

- Bindeo dinámico a atributos de elementos nativos:
  $={{ }}

  Ej. *class$="{{foo}}", href$="{{url}}"...*

- Bindeo bidireccional a elementos nativos

  Polymer usa una convención de sintaxis de eventos para conseguir el binder bidireccional.
  Para bindear bidireccionalmente a elementos nativos que no sigan esta convención, se puede usar la siguiente sintaxis:

  target-prop="{{hostProp::target-change-event}}"

  Ejemplo:

```
<!-- Listens for `input` event and sets hostValue to <input>.value -->
<input value="{{hostValue::input}}">

<!-- Listens for `change` event and sets hostChecked to <input>.checked -->
<input type="checkbox" checked="{{hostChecked::change}}">
```

```
<dom-module id="data-bindings">

  <style>
    :host {
      display: block;
      box-sizing: border-box;
    }
  </style>

  <template>
    <h1>Host</h1>
    <h2 style="color: pink">{{name}}</h2>
    <child-1 name="{{name}}"></child-1>
    <child-2 name="{{name}}"></child-2>
  </template>

</dom-module>

<script>

  Polymer({

    is: 'data-bindings',

    listeners: {
      'loggedin': '_greet'
    },

    properties: {

      name: {
        type: String
      }

    },
```

```
  <template>
    <div id="block">
      <h1>&lt;child-1&gt;</h1>
      <h2>Hola, <span>{{name}}</span></h2>
      <!--<input type="text" value="{{name::input}}">-->
      <paper-input label="Introduce tu nombre" value="{{name}}"></paper-input>
    </div>
  </template>

</dom-module>

<script>

  Polymer({

    is: 'child-1',

    properties: {
      name: {
        type: String,
        value: 'Desconocido',
        notify: true
      }
    }
```

DATA BINDING: HELPERS

- Templates repetidos

  <template is="dom-repeat" id="employeeList" items="{{employees}}">

  item: El elemento del array utilizado.
  index: El índice de item dentro del array (El valor del índice cambia si el array se ordena o filtra).
  Los cambios en el array han de hacerse con los métodos propios de Polymer (push, pop, splice, shift, unshift) para que los cambios en los arrays sean observables y se actualicen.

- Templates condicionales

  <template is="dom-if" if="{{user.isAdmin}}">

```
<template>
  <h3>APP</h3>
  <div class="container">
    <mobiles-list></mobiles-list>
    <template is="dom-if" if="{{mobileSelected}}">
      <mobile-data mobile-detail="{{mobileSelected}}"></mobile-data>
    </template>
  </div>
</template>
```

```
<template>
  <iron-ajax
      auto
      url="mobiles.json"
      handle-as="json"
      on-response="handleResponse"
      last-response="{{data}}"
      debounce-duration="300">
  </iron-ajax>
  <h2>Listado de móviles</h2>
  <template is="dom-repeat" id="list" items="{{data.mobiles}}">
      <p class="list-mobile layout horizontal">
        <span class="flex-2">{{item.name}}</span>
        <span class="flex-1 button-detail"><button on-click="showDetail">Detalle</button></span>
      </p>
  </template>
</template>
```

DATA BINDING

Video de Rob Dodson sobre Data Binding

https://youtu.be/1sx6YNn58OQ?list=PLNYkxOF6rcIDdS7HWIC_BYRunV6MHs5xo

BEHAVIORS

- Módulos para extender nuestros elementos. Pueden definir:

1. Callbacks de ciclo de vida
2. Propiedades
3. Atributos
4. Observers
5. Listeners

LISTENERS

- Existen tres tipos:

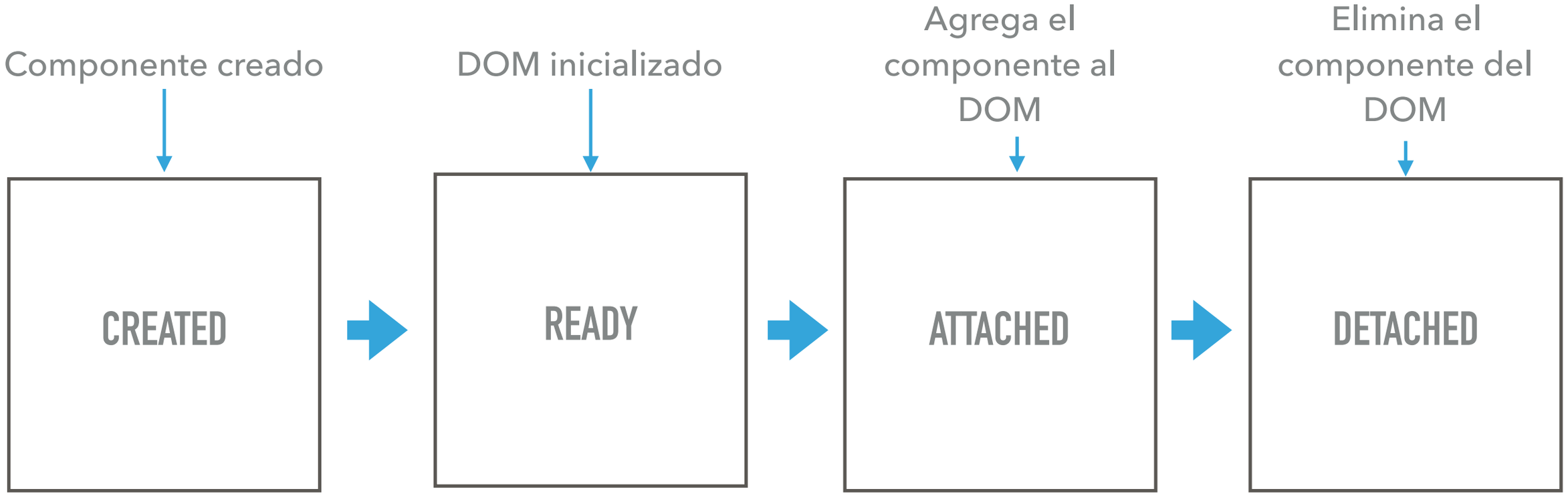
1. Declarativo
   <button on-tap="handleClick">Kick Me</button>
2. Simple
   listeners: {
   'tap': 'handleTap'
   }
3. Por nodo
   listeners: {
   'special.tap': 'specialTap'
   }

Fire

El método Fire nos permite lanzar eventos personalizados, pudiendo pasar datos a los manejadores de eventos como un argumento del propio método:
this.fire('kick', {kicked: true});

CICLO DE VIDA

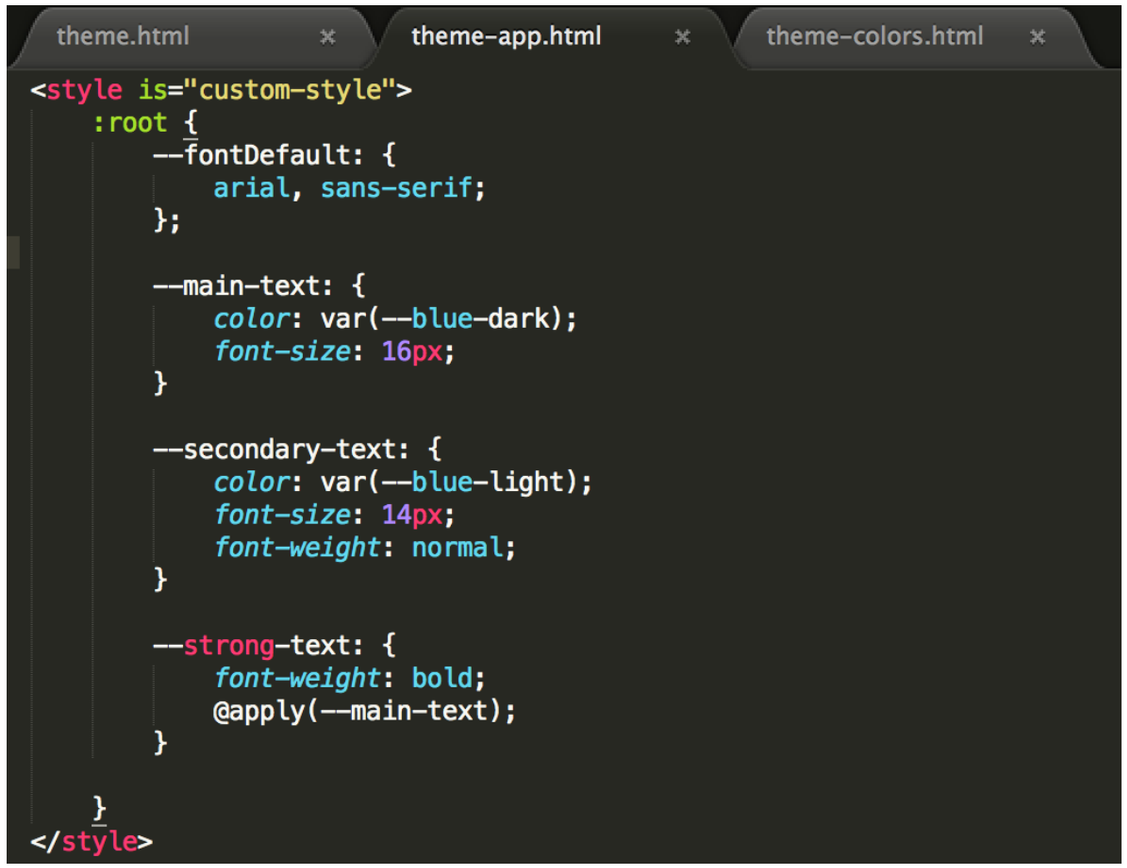
- Polymer provee una serie de callbacks para poder subscribirse a los eventos de su ciclo de vida. Solo hay que implementarlos en el prototipo:

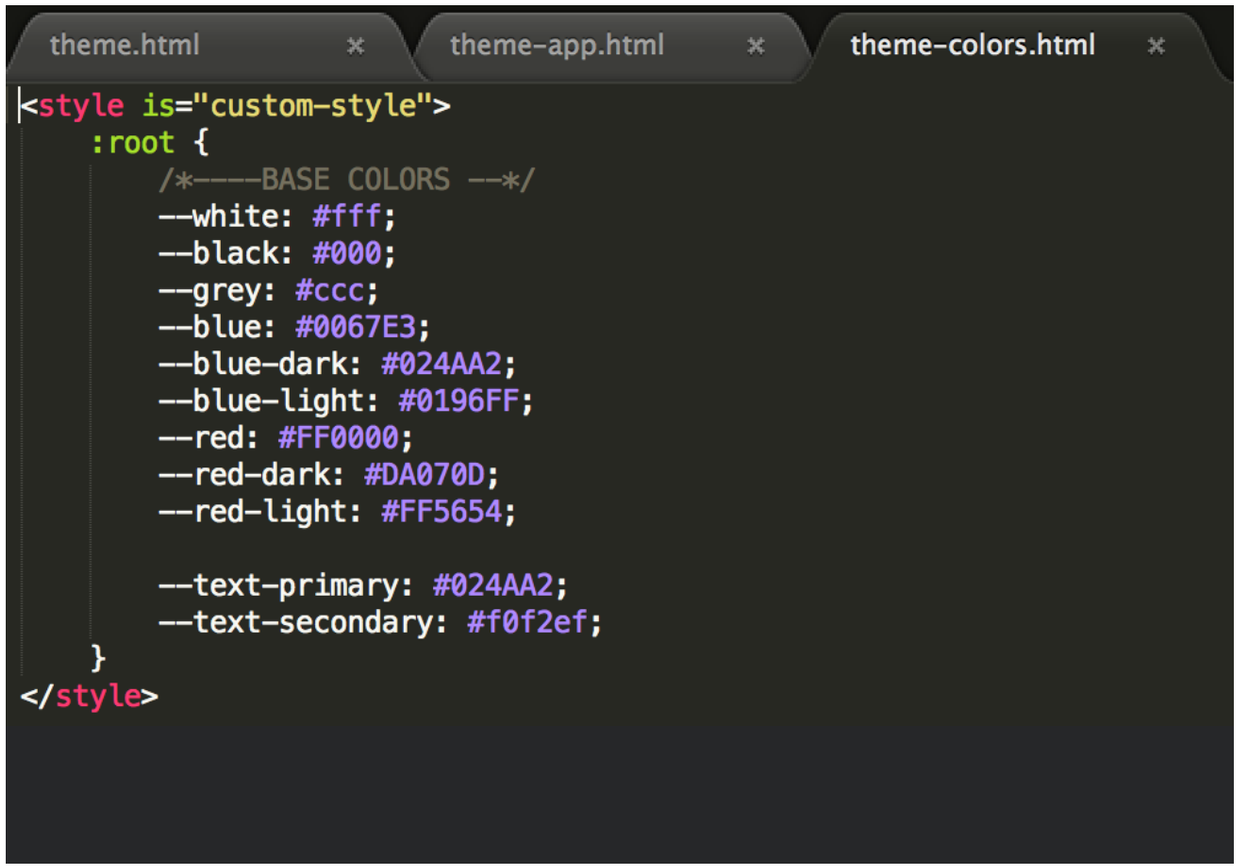


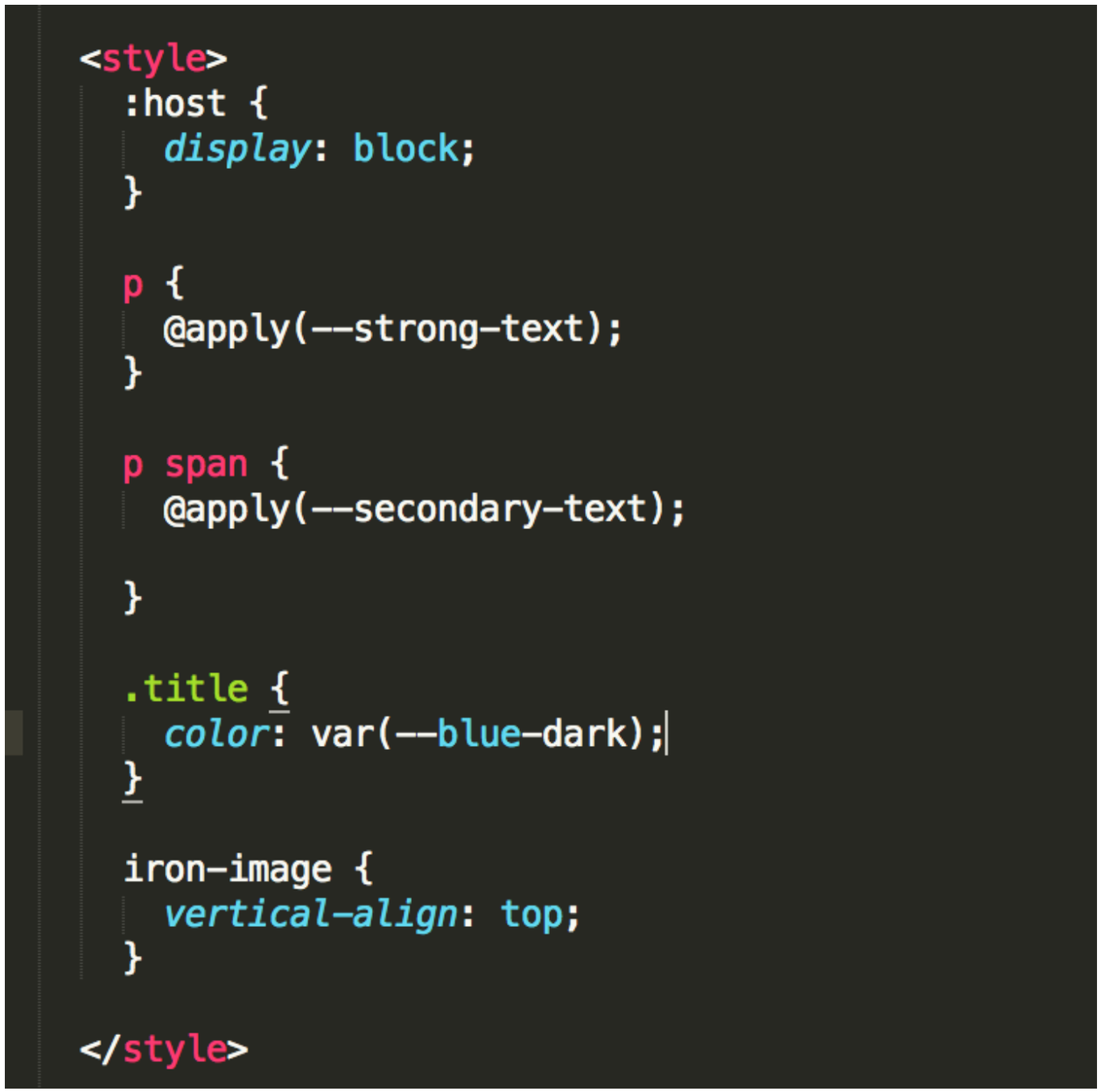
STYLING & THEMING

- ::shadow y /deep/ -> Mal, rompen el concepto de encapsulación y el nivel de profundidad hace que sea impreciso
- Polymer nos ofrece el atributo custom-style, por el que podemos definir una serie de estilos que se puede compartir a lo largo de la aplicación en forma de componente. La filosofía es similar a la de preprocesadores css como sass o stylus
- Aprovechamos el nuevo concepto de css3: Variables y @apply
- En nuestro componente llamaremos a las variables o "custom styles" declaradas en nuestro theme. *Ej: color: var (--color-red, red)*

- Además, Polymer ha preparado el componente **iron-flex-layout**, que nos va a permitir utilizar flex de manera declarativa en nuestros templates
- Ejemplo de trabajo: theme que incluye los distintos html de fuentes, variables generales, mixins, etc. Luego se importa ese theme y cada componente lo puede consumir posteriormente a su gusto







```
<style>
  :host {
    display: block;
  }

  p {
    @apply(--strong-text);
  }

  p span {
    @apply(--secondary-text);

  }

  .title {
    color: var(--blue-dark);
  }

  iron-image {
    vertical-align: top;
  }

</style>
```

- ANALIZANDO UN COMPONENTE POLYMER
- HERRAMIENTAS
- TESTING
- BUENAS PRÁCTICAS
- EL CATÁLOGO DE ELEMENTOS DE POLYMER
- ROADMAP: MIRANDO AL FUTURO
  - HTTPS://GITHUB.COM/POLYMER/PROJECT/BLOB/MASTER/ROADMAP.MD

ANALIZANDO UN COMPONENTE POLYMER

```
<link href="bower_components/polymer/polymer.html" rel="import">

<dom-module id="medianet-element">
  <style>
    p {
      color: red;
    }
  </style>
  <template>
    <content></content>
    <p>Adios, medianos</p>
  </template>
</dom-module>

<script>
  Polymer({
    is: "medianet-element",
    properties: {

    },
    ready: function(e){

    }
  });
</script>
```

ANALIZANDO UN COMPONENTE POLYMER

```
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, minimum-scale=1.0, initial-scale=1.0, user-scalable=yes">
    <title>medianet-element Demo</title>
    <script src="bower_components/webcomponentsjs/webcomponents-lite.js"></script>
    <link rel="import" href="medianet-element.html">
  </head>
  <body>

    <p>An example of <code>&lt;medianet-element&gt;</code>:</p>

    <medianet-element>
      <h2>Hola, medianos</h2>
    </medianet-element>

    <script>

    </script>

  </body>
</html>
```

An example of <medianet-element>:

Hola, medianos

Adios, medianos

HERRAMIENTAS

- EXTERNAS

  - Bower, Gulp, Yeoman

- POLYMER

  - Polyserve, Polylint, Polybuild (Vulcanize, Crisper y Polyclean), Polygit

TESTING

- Polymer incorpora el WCT (Web Component Tester) como solución para la confección de test unitarios para nuestros componentes. Utiliza Mocha Y Chai.

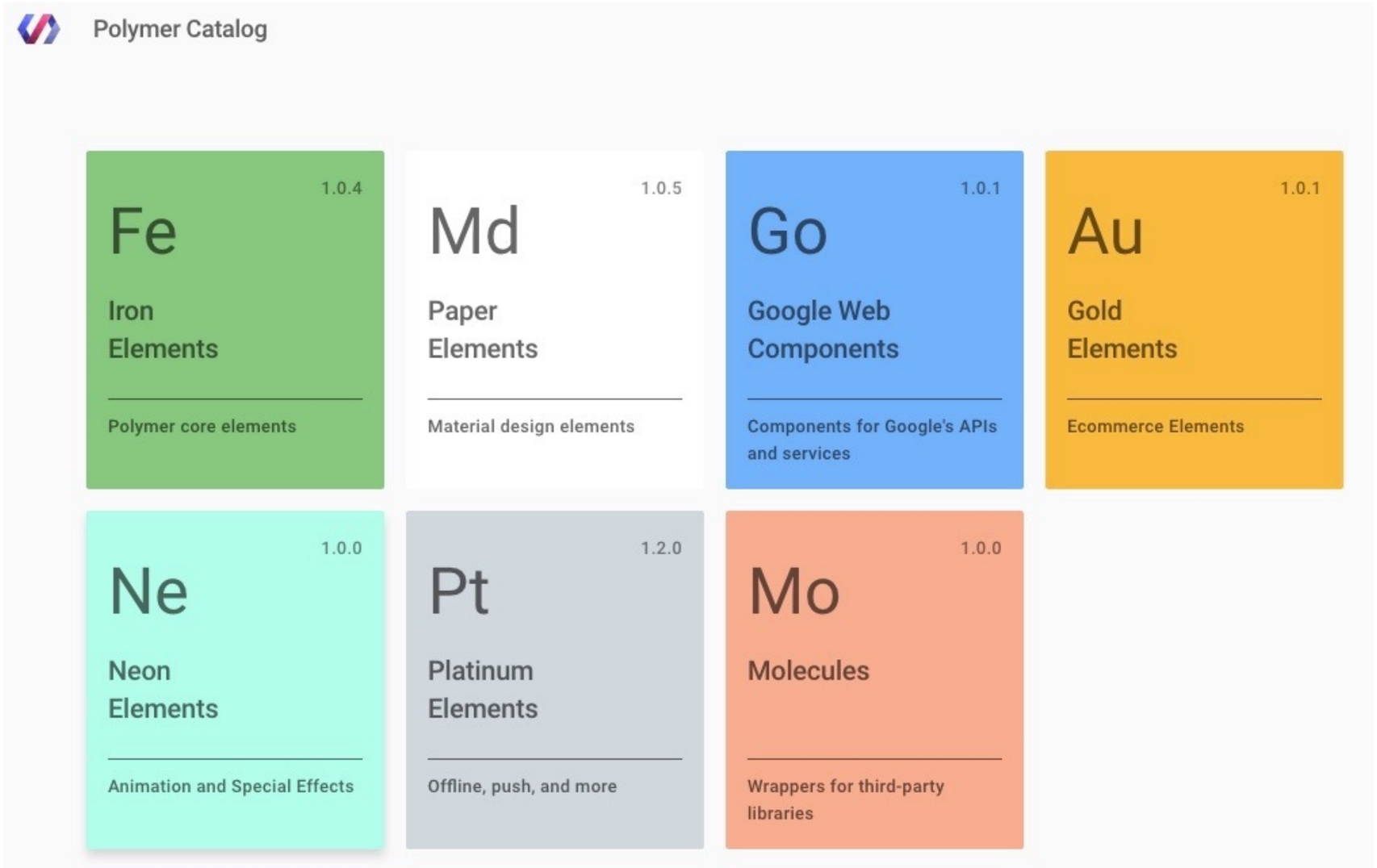
```
<script>
  var myEl = document.querySelector('data-bindings');

  suite('<data-bindings>', function() {

    test('defines the "author" property', function() {
      assert.equal(myEl.author.name, 'Dimitri Glazkov');
      assert.equal(myEl.author.image, 'http://addyosmani.com/blog/wp-content/uploads/2013/04/unicorn.jpg');
    });

    test('says hello', function() {
      assert.equal(myEl.sayHello(), 'data-bindings says, Hello World!');

      var greetings = myEl.sayHello('greetings Earthlings');
      assert.equal(greetings, 'data-bindings says, greetings Earthlings');
    });

    test('fires lasers', function(done) {
      myEl.addEventListener('data-bindings-lasers', function(event) {
        assert.equal(event.detail.sound, 'Pew pew!');
        done();
      });
      myEl.fireLasers();
    });

  });
</script>
```

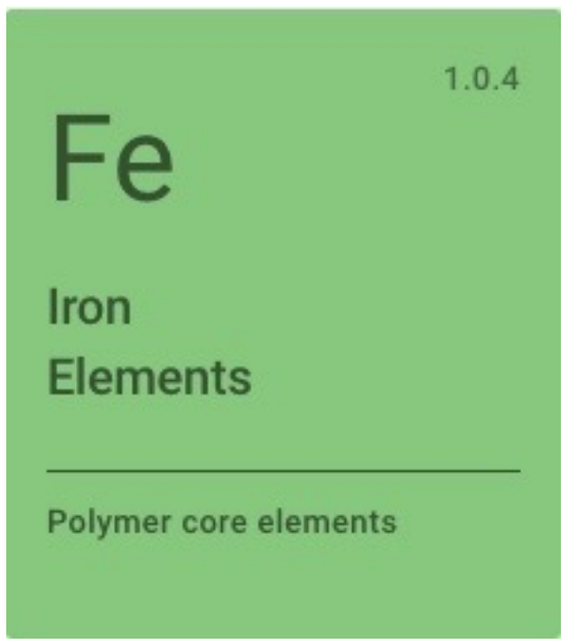
BUENAS PRÁCTICAS

- Documentar detalladamente las propiedades y métodos del componente (http://polymerelements.github.io/style-guide/)
- Confeccionar el test unitario de nuestro componente
- Si nuestro componente va a formar parte de un ecosistema, adherirse al theming que ese sistema tenga definido.
- **Mantener desacoplados los componentes comunicándolos mediante data-binding y eventos.**

EL CATÁLOGO DE ELEMENTOS DE POLYMER

- https://elements.polymer-project.org/





- Conjunto de elementos de utilidad visuales y no visuales
- Incluye desde elementos para trabajar con el diseño, la entrada de usuario, la selección, hasta llamadas AJAX, etc.

Ejemplo:

```
<iron-ajax
auto url="http://gdata.youtube.com/feeds/api/videos/"
params='{"alt":"json", "q":"chrome"}'
handle-as="json"
on-response="handleResponse">
</iron-ajax>
```

- Conjunto de elementos visuales que implementan el Google Material Design

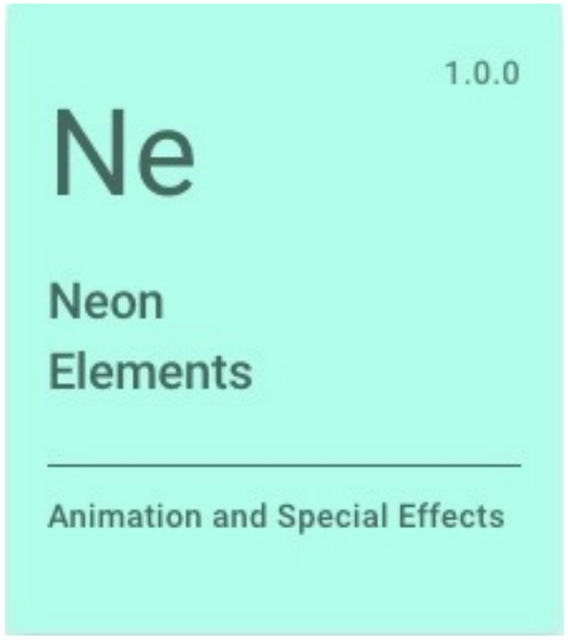
- Son elementos como botones, checkbox, modales, etc, cada uno con su respectivo comportamiento también implementado.

Ejemplo:

```
<paper-button raised>Raised button</paper-button>
```

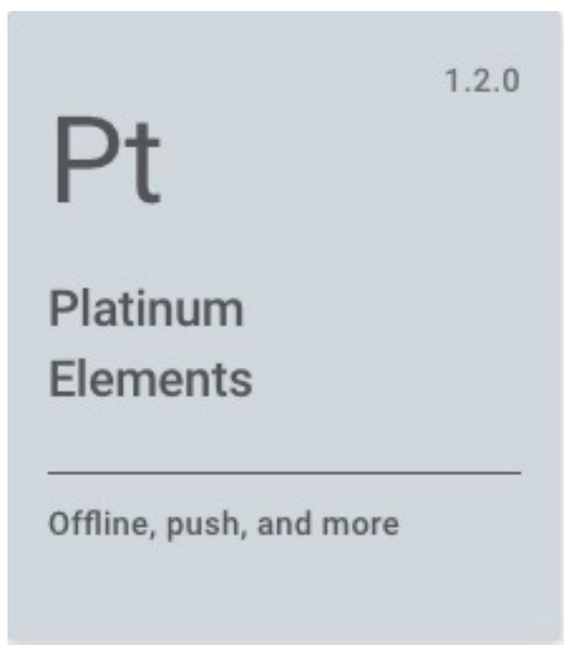
- Elementos para APIs y servicios de Google
- Relacionados con aplicaciones propias de Google
- Firebase, Analytics, Youtube...
- Ejemplo:
  <google-map latitude="37.77493" longitude="-122.41942"></google-map>

- Orientados a eCommerce
- Ejemplo:
  <gold-cc-expiration-input value="11/15"></gold-cc-expiration-input>



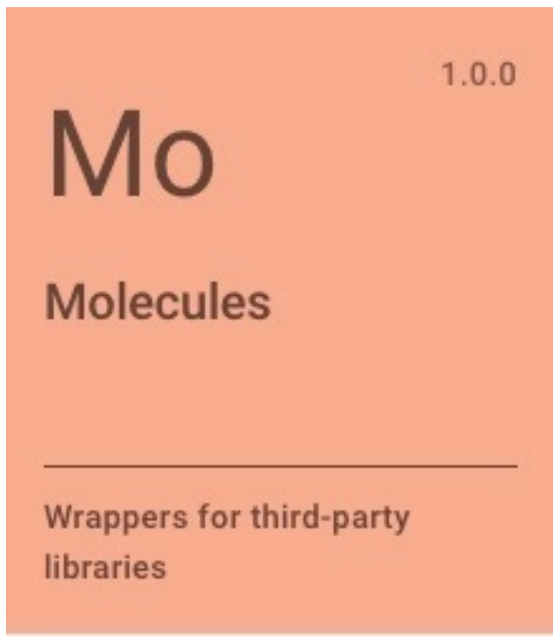
- Elementos de animaciones y efectos.
- Permiten implementar fácilmente animaciones y otros efectos en nuestros componentes.
- Ejemplo:

```
<neon-animated-pages entry-animation="[[entryAnimation]]" exit-animation="[[exitAnimation]]">
  <neon-animatable>1</neon-animatable>
  <neon-animatable>2</neon-animatable>
</neon-animated-pages>
```



- Offline, notificaciones…
- Ejemplo:

```
<platinum-sw-register auto-register>
  <platinum-sw-cache></platinum-sw-cache>
</platinum-sw-register>
```



- Wrappers de librerías de terceros.
- Ejemplo:

```
<marked-element markdown="`Markdown` is _awesome_!">
  <div class="markdown-html"></div>
</marked-element>
```

MUCHAS GRACIAS