

# **Patrones y principios de diseño**

# Principios de diseño

- Tell, Don't Ask
  - “Dime lo que necesitas”
  - Claridad y expresividad
  - Encapsular las comprobaciones

# Tell, Don't Ask

Los síntomas:

```
if (typeof(variables) === 'object' && !Array.isArray(variables)) {  
    variables = Object.keys(variables).map(function (k) {  
        var value = variables[k];  
  
        if (! (value instanceof tree.Value)) {  
            if (! (value instanceof tree.Expression)) {  
                value = new(tree.Expression)([value]);  
            }  
            value = new(tree.Value)([value]);  
        }  
        return new(tree.Rule)( '@' + k, value, false, 0);  
    });  
    frames = [new(tree.Ruleset)(null, variables)];  
}
```

# Tell, Don't Ask

Los síntomas:

```
if (typeof(variables) === 'object' && !Array.isArray(variables)) {  
    variables = Object.keys(variables).map(function (k) {  
        var value = variables[k];  
  
        if (! (value instanceof tree.Value)) {  
            if (! (value instanceof tree.Expression)) {  
                value = new(tree.Expression)([value]);  
            }  
            value = new(tree.Value)([value]);  
        }  
        return new(tree.Rule)('@' + k, value, false, 0);  
    });  
    frames = [new(tree.Ruleset)(null, variables)];  
}
```

# **Tell, Don't Ask**

Programación:

- Estructurada: adquiere info y toma decisiones
- OO: manda a los objetos hacer cosas

# **Tell, Don't Ask**

El error:

1. Preguntar a un objeto sobre su estado
2. Tomar una decisión
3. Decirle lo que tiene que hacer

# **Tell, Don't Ask**

El error:

1. Preguntar a un objeto sobre su estado
2. Tomar una decisión
3. Decirle lo que tiene que hacer

¡Probablemente ese código pertenece al objeto!

# Tell, Don't Ask

```
if (usuario.primerLogin) {  
    usuario.mostrarMensajeBienvenida();  
} else {  
    usuario.mostrarSaludo();  
}
```

# Tell, Don't Ask

```
var Usuario = Class.extend({
    saludar: function() {
        if (this.primerLogin) {
            this.mostrarMensajeBienvenida();
        } else {
            this.mostrarSaludo();
        }
    });
});
```

// y después...

 usuario.saludar();

# Tell, Don't Ask

```
function comprobarTimeout(respuesta) {  
    if ((Date.now() - respuesta.start) > 10000) {  
        respuesta.notificarTimeout();  
    }  
}
```

# Tell, Don't Ask

```
var Respuesta = Class.extend({
    comprobarTimeout: function() {
        if ((Date.now() - this.start) > 10000) {
            this.notificarTimeout();
        }
    }
});  
  
// y después...  
  
respuesta.comprobarTimeout();
```

# Tell, Don't Ask

```
var elementos = miColeccion.getItems();
for (var i=0; i<elementos.length; i++) {
    var elemento = elementos[i];
    console.log(elemento.nombre);
}
```

# Tell, Don't Ask

```
miColeccion.forEach(function(e) {  
    console.log(e.nombre);  
});
```

# Tell, Don't Ask

```
var elemento = new Elemento("hola", 12);
var lista = miColeccion.getItems();
lista.addElementAt(elemento.getOrder(), elemento);
```

# Tell, Don't Ask

```
var elemento = new Elemento("holá", 12);  
miColeccion.add(elemento);
```

# **Tell, Don't Ask**

Es decir:

- Los datos y las operaciones sobre esos datos deben estar en el mismo sitio (objeto)
- Encapsular, desacoplar
- “Command/Query Separation”
  - Consulta información
  - Da una orden y deja al objeto decidir
  - Pero no las mezcles!

# Tell, Don't Ask

Ventajas:

- ✓ Más robusto (menor acoplamiento)
- ✓ Menor tendencia a repetir lógica
- ✓ Mejor estructurado

Inconvenientes:

- Miles de métodos de 2 o 3 líneas
- “Ruido” en las clases

# Principios de diseño

- S.O.L.I.D.
  - Single Responsibility
  - Open-Closed
  - Liskov Substitution
  - Interface Segregation
  - Development Dependency Inversion

# Principios de diseño

- SRP: Single Responsibility Principle
  - El código de una elemento ha de tener solo una razón para cambiar.
  - EL principio de diseño
  - También el complementario: cada responsabilidad ha de tener un único lugar en el código (D.R.Y.)

# SRP

Es común ver cosas como esta:

```
$.ajax({ ... })
  .success(function() {
    cambioEnInterfaz();
    mostrarModal();
    if ($("#elemento").value() == "Ok") {
      /* ... */
    }
    globalSeHaGuardado = true;
})
.error(function() {
  // ...
});
```

# SRP

O como esta:

```
var Widget = Class.extend({
    onClick: function() { ... },
    guardar: function() { ... },
    render: function() { ... },
    mostrarError: function() { ... }
});
```

# SRP

```
var Widget = Model.extend({  
    guardar: function() { ... }  
});
```

```
var WidgetView = View.extend({  
    render: function() { ... }  
});
```

```
var WidgetController = Controller.extend({  
    onClick: function() { ... }  
});
```

```
var ErrorAlert = ModalWindow.extend({  
    mostrarError: function() { ... }  
});
```

# **SRP**

Caso práctico: masonry.js

- <https://github.com/desandro/masonry/blob/master/jquery.masonry.js>
- en el método **\_create** (línea 102)...

# SRP

```
// sets up widget
_create : function( options ) {
    // [...]

    // get original styles in case we re-apply them in .destroy()
    var elemStyle = this.element[0].style;
    this.originalStyle = {
        // get height
        height: elemStyle.height || ''
    };
    // get other styles that will be overwritten
    // [...]

    s.isFluid = this.options.columnWidth && typeof this.options.columnWidth === 'function';

    // add masonry class first time around
    var instance = this;
    setTimeout( function() {
        instance.element.addClass('masonry');
    }, 0 );

    // bind resize method
    if ( this.options.isResizable ) {
        $(window).bind( 'smartresize.masonry', function() {
            instance.resize();
        });
    }
}
```

# SRP

```
// sets up widget
_create : function( options ) {
  // [...]
```

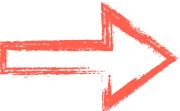


```
  // get original styles in case we re-apply them in .destroy()
  var elemStyle = this.element[0].style;
  this.originalStyle = {
    // get height
    height: elemStyle.height || ''
  };
  // get other styles that will be overwritten
  // [...]
```

```
s.isFluid = this.options.columnWidth && typeof this.options.columnWidth === 'function';
```



```
  // add masonry class first time around
  var instance = this;
  setTimeout( function() {
    instance.element.addClass('masonry');
  }, 0 );
```



```
  // bind resize method
  if ( this.options.isResizable ) {
    $(window).bind( 'smartresize.masonry', function() {
      instance.resize();
    });
  }
```

# **SRP**

El resultado: caos!

- No hay un lugar claro para cada operación
- Es difícil entender qué hace cada línea
  - El “qué” está enterrado en el “cómo”
- Muy complicado de testear
- Difícil de reutilizar

# **SRP**

## Caso práctico: BrowserQuest

- [https://github.com/mozilla/BrowserQuest/blob/  
master/client/js/chest.js](https://github.com/mozilla/BrowserQuest/blob/master/client/js/chest.js)

# SRP

```
var Chest = Entity.extend({
    init: function(id, kind) {
        this._super(id, Types.Entities.CHEST);
    },

    getSpriteName: function() {
        return "chest";
    },

    isMoving: function() {
        return false;
    },

    open: function() {
        if(this.open_callback) {
            this.open_callback();
        }
    },

    onOpen: function(callback) {
        this.open_callback = callback;
    }
});
```

# **SRP**

## Caso práctico: BrowserQuest

- Todo el proyecto está muy bien estructurado
  - entity.js
  - character.js
  - animation.js
  - ...
- A pesar de ser muy grande, cada responsabilidad tiene su sitio

# **SRP**

## Caso práctico: Backbone.js

- <https://github.com/documentcloud/backbone/blob/master/backbone.js>
- en **Backbone.Model**, línea 179...

# SRP

- Por un lado..
  - gestión de estado (**set, get**)
  - validación
  - formateo (**toJSON, escape**)
  - servidor (**fetch, save**)
- Por otro...
  - ✓ Delega los detalles a otros módulos (**Sync, Event**)
  - ✓ Bajo acoplamiento (“interfaces”)

# **SRP**

“Una responsabilidad”...

- Subjetivo
- “Una sola razón para cambiar”...
  - “Para qué todo funcione bien”
  - Muy dependiente del nivel de abstracción
  - Y de cada módulo
- El exceso es tan malo como el defecto

# **S.O.L.I.D.**

## Open-Closed

- “Un elemento ha de estar abierto a la extensión pero cerrado a la modificación”
  - Abierto a la extensión: poder ser adaptado a las (futuras) necesidades de la aplicación
  - Cerrado a la modificación: que la adaptación no implique modificar su código

# Open-Closed

```
var Lenguas = { Castellano: 0,  
                Ingles: 1 };  
  
var Persona = Class.extend({  
    init: function(lengua) { this.lengua = lengua; },  
    saludar: function(lengua) {  
        if (this.lengua == Lenguas.Castellano) {  
            alert("Hola!");  
        } else if (this.lengua == Lenguas.Ingles) {  
            alert("Hello!");  
        }  
    }  
});  
  
new Persona(Lenguas.Castellano).saludar();
```

# Open-Closed

```
var Persona = Class.extend({  
    saludar: function() {  
        alert(this.saludo);  
    }  
});
```

```
var Angloparlante = Persona.extend({  
    init: function() { this.saludo = "Hello!"; }  
});
```

```
var Hispanohablante = Persona.extend({  
    init: function() { this.saludo = "Hola!"; }  
});
```

```
new Hispanohablante().saludar();
```

# **Open-Closed**

Pretende:

- Promover el uso de abstracciones
- Código modular y flexible ante el cambio
- Evitar un torrente de cambios en cascada!

# Open-Closed

Pretende:

- Promover el uso de abstracciones
- Código modular y flexible ante el cambio
- Evitar un torrente de cambios en cascada!

Es decir:

- Especificar y respetar interfaces

# Open-Closed

```
var Canvas = Class.extend({
    render: function(figura) {
        if (figura instanceof Triangulo) {
            // ...
        } else if (figura instanceof Cuadrado) {
            // ...
        }
    }
});
```

# Open-Closed

```
var Canvas = Class.extend({
    render: function(figura) {
        figura.draw(this);
    }
});
```

```
var Triangulo = Figura.extend({
    draw: function(canvas) { ... }
});
```

```
var Cuadrado = Figura.extend({
    draw: function(canvas) { ... }
});
```

# Open-Closed

Caso práctico: three.js

- [https://github.com/mrdoob/three.js/blob/master/src/  
renderers/CanvasRenderer.js](https://github.com/mrdoob/three.js/blob/master/src/renderers/CanvasRenderer.js)
- método **render**, línea 225

# Open-Closed

```
if ( element instanceof THREE.RenderableParticle ) {  
    // ...  
}  
} else if ( element instanceof THREE.RenderableLine ) {  
    // ...  
}  
} else if ( element instanceof THREE.RenderableFace3 ) {  
    // ...  
}  
} else if ( element instanceof THREE.RenderableFace4 ) {  
    // ...  
}
```

# Open-Closed

Caso práctico: jasmine.js

- <https://github.com/pivotal/jasmine/blob/master/src/core/Reporter.js>

# Open-Closed

```
/** No-op base class for Jasmine reporters.  
 *  
 * @constructor  
 */  
jasmine.Reporter = function() {  
};  
  
//noinspection JSUnusedLocalSymbols  
jasmine.Reporter.prototype.reportRunnerStarting = function(runner) {  
};  
  
//noinspection JSUnusedLocalSymbols  
jasmine.Reporter.prototype.reportRunnerResults = function(runner) {  
};  
  
//...
```

# S.O.L.I.D.

## Dependency inversion

- “Depende de abstracciones. No dependas de cocreciones”
  - Entidades de alto nivel no deben depender de entidades de bajo nivel. Ambos deben depender de abstracciones.
  - Las abstracciones no deben depender de detalles. Los detalles deben depender de abstracciones.

# Dependency Inversion

```
var Model = Class.extend({
  save: function() {
    var tbind = curry(bind, this),
        stop = bind(this.icon, this.icon.stop);
    $.post(this.url, this.getData())
      .success(tbind(this.saved))
      .error(tbind(this.saveFailed))
      .complete(stop);
  }
});
```

# Dependency Inversion

```
var Model = Class.extend({  
    save: function() {  
        var tbind = curry(bind, this),  
            stop = bind(this.icon, this.icon.stop);  
        $().post(this.url, this.getData())  
            .success(tbind(this.saved))  
            .error(tbind(this.saveFailed))  
            .complete(stop);  
    }  
});
```



# Dependency Inversion

```
var Model = Class.extend({
  init: function(store) { this.store = store; } ←
  save: function() {
    var tbind = curry(bind, this),
        stop = bind(this.icon, this.icon.stop);
    this.store.save(
      this.data,
      tbind(this.saved),
      tbind(this.saveFailed),
      stop
    );
  }
});
```

```
var Store = Class.extend({
  save: function(data, success, error, complete) { }
});
```

# Dependency Inversion

```
var ServerStore = Store.extend({
  save: function(data, success, error, complete) {
    $.post(this.url, data)
      .success(success)
      .error(error)
      .complete(complete);
  }
});
```

```
var db = {};
var MemStore = Store.extend({
  save: function(data, success, error, complete) {
    db[this.url] = data;
    complete();
    success();
  }
});
```

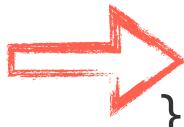
# Dependency Inversion

Caso práctico: backbone.js

- <https://github.com/documentcloud/backbone/blob/master/backbone.js>
- `Backbone.Model#fetch`, línea 335

# Dependency Inversion

```
fetch: function(options) {  
    options = options ? _.clone(options) : {};  
    var model = this;  
    var success = options.success;  
    options.success = function(resp, status, xhr) {  
        if (!model.set(model.parse(resp, xhr), options))  
            return false;  
        if (success) success(model, resp, options);  
    };  
    return this.sync('read', this, options);  
},
```



# **Patrones de organización**

# **Patrones de organización**

- Parámetros con nombre/por defecto
- Módulos y namespaces
- Control de acceso
- Mixins

# Parámetros con nombre

```
function ajax(url, data, method,
              success, error, complete) {
  url || url = "/";
  data || data = {};
  method || method = "POST";
  // ...
}

ajax("/",  
      {},  
      "GET",  
      function(){ ... },  
      function() { ... },  
      function() { ... });
```

# Parámetros con nombre

```
function ajax(options) {  
  var url = options.url || "/",  
      data = options.data || {},  
      method = options.method || "POST";  
  //...  
}  
  
ajax({data: [1, 2], complete: function() { ... }});
```

# Parámetros por defecto

```
function ajax(options) {  
  var fn = function() {},  
      defaults = {url: "/", data: [], method: "POST",  
                  success: fn, error: fn, complete: fn};  
  options = merge(defaults, options);  
  // ...  
}  
  
ajax({data: [1, 2], complete: function() { ... }});
```

# Parámetros por defecto

backbone.js:749

```
reset: function(models, options) {
  for (var i = 0, l = this.models.length; i < l; i++) {
    this._removeReference(this.models[i]);
  }
  this._reset();
  if (models) this.add(models, _.extend({silent: true}, options));
  if (!options || !options.silent) this.trigger('reset', this, options);
  return this;
},
```



# **Intermedio: merge**

¿Cómo sería esa función `merge`?

# Intermedio: merge

¿Cómo sería esa función `merge`?

```
function merge() {  
  var slice = Array.prototype.slice,  
      sources = slice.call(arguments),  
      target = {};  
  sources.forEach(function(source) {  
    for (var p in source) if (source.hasOwnProperty(p)) {  
      target[p] = source[p];  
    }  
  });  
  return target;  
}
```

# Módulos y namespaces

- JavaScript no tiene concepto de namespace
- Todo tirado en objeto global
  - Mucha polución
  - Colisión de nombres
  - Difícil de navegar

# Módulos y namespaces

- JavaScript no tiene concepto de namespace
- Todo tirado en objeto global
  - Mucha polución
  - Colisión de nombres
  - Difícil de navegar
- ¡Pero tenemos funciones!

# Módulos y namespaces

```
function miHelper() {  
    // ...  
}  
  
var miVariableTemporal = 0;  
var estadoLocal = {};
```

# Módulos y namespaces

```
function sandbox() {  
    function miHelper() {  
        // ...  
    }  
  
    var miVariableTemporal = 0;  
    var estadoLocal = {};  
}
```

# Módulos y namespaces

```
(function sandbox() {
    function miHelper() {
        // ...
    }

    var miVariableTemporal = 0;
    var estadoLocal = {};
})()
```

# Módulos y namespaces

```
(function sandbox() {  
    function miHelper() {  
        // ...  
    }  
  
    var miVariableTemporal = 0;  
    var estadoLocal = {};  
    }())
```



# Módulos y namespaces

```
function miFuncionUtil() {  
    // ...  
}
```

```
function miGranMetodo() {  
    // ...  
}
```

```
function miEstupendoHelper() {  
    // ...  
}
```

# Módulos y namespaces

```
function aux() { }
var state = "off";
```

```
function miFuncionUtil() {
    // ...
}
```

```
function miGranMetodo() {
    // ...
}
```

```
function miEstupendoHelper() {
    // ...
}
```

# Módulos y namespaces

```
(function() {  
    function aux() { }  
    var state = "off";  
  
    function miFuncionUtil() { }  
  
    function miGranMetodo() { }  
  
    function miEstupendoHelper() { }  
}())
```

# Módulos y namespaces

```
var Modulo = (function() {  
    function aux() { }  
    var state = "off";  
  
    function miFuncionUtil() { }  
  
    function miGranMetodo() { }  
  
    return {  
        miFuncionUtil: miFuncionUtil,  
        miGranMetodo: miGranMetodo  
    };  
}());
```

# Módulos y namespaces

```
→ var Modulo = (function() {  
    function aux() { }  
    var state = "off";  
  
    function miFuncionUtil() { }  
  
    function miGranMetodo() { }  
  
    → return {  
        miFuncionUtil: miFuncionUtil,  
        miGranMetodo: miGranMetodo  
    };  
}());
```

# Módulos y namespaces

```
Modulo.miFuncionUtil();
```

# Módulos y namespaces

```
var Modulo = {};  
  
(function(Modulo) {  
    function aux() {}  
    var state = "off";  
  
    Modulo.miFuncionUtil = function() {}  
  
    Modulo.miGranMetodo = function() {}  
  
})(Modulo);
```

# Módulos y namespaces

```
var Modulo = {};  
  
(function(Modulo) {  
    function aux() {}  
    var state = "off";  
    Modulo.miFuncionUtil = function() {}  
}(Modulo));  
  
(function(Modulo) {  
    Modulo.miGranMetodo = function() {}  
}(Modulo));
```

# Módulos y namespaces

```
var Modulo = (function(Modulo) {  
    function aux() { }  
    var state = "off";  
    Modulo.miFuncionUtil = function() { }  
    return Modulo;  
}(Modulo || {}));
```

```
var Modulo = (function(Modulo) {  
    Modulo.miGranMetodo = function() { }  
    return Modulo;  
}(Modulo || {}));
```

# Módulos y namespaces

```
→ var Modulo = (function(Modulo) {  
    function aux() { }  
    var state = "off";  
    Modulo.miFuncionUtil = function() { }  
    return Modulo;  
}(Modulo || {})); ←
```

```
var Modulo = (function(Modulo) {  
    Modulo.miGranMetodo = function() { }  
    return Modulo;  
}(Modulo || {}));
```

# **Intermedio: mejorar klass.js**

Convierte klass.js en un módulo Class

# Módulos y namespaces

Submódulos: muy fácil!

```
var MiLibreria = {};  
  
MiLibreria.eventos = (function(eventos) {  
    eventos.on = function() { };  
    eventos.off = function() { };  
    return eventos;  
}(MiLibreria.eventos));
```

# **Módulos y namespaces**

Según crece la aplicación...

# Módulos y namespaces

Según crece la aplicación...

```
var MiLibreria = MiLibreria || {};
```

# Módulos y namespaces

Según crece la aplicación...

```
var MiLibreria = MiLibreria || {};
MiLibreria.widgets = MiLibreria.widgets || {};
```

# Módulos y namespaces

Según crece la aplicación...

```
var MiLibreria = MiLibreria || {};
MiLibreria.widgets = MiLibreria.widgets || {};
MiLibreria.widgets.buttons = MiLibreria.widgets.buttons || {};

MiLibreria.widgets.buttons.actionButtons = (function(buttons) {
    buttons.ok = new Widget({ ... });
    buttons.cancel = new Widget({ ... });
})(MiLibreria.widgets.buttons.actionButtons || {}));
```

# Módulos y namespaces

Namespaces, pero más cómodos:

```
MiLib.namespace('widgets.buttons.actionButtons', function(my) {  
    my.ok = new Widget({ ... });  
    my.cancel = new Widget({ ... });  
});
```

# **Intermedio: namespace**

¿Como sería la función MiLib.namespace?

# Intermedio: namespace

¿Como sería la función MiLib.namespace?

```
var MiLib = (function(my) {  
    my.namespace = function(string, sandbox) {  
        // ???  
    };  
    return my;  
}(MiLib || {}));
```

# Intermedio: namespace

¿Como sería la función MiLib.namespace?

```
var MiLib = (function(my) {
    my.namespace = function(string, sandbox) {
        var spaces = string.split('.'),
            root = my,
            space;
        while (space = spaces.shift()) {
            root = root[space] || (root[space] = {});
        }
        return sandbox(root);
    };
    return my;
}(MiLib || {}));
```

# Mixins

- Otra forma de reutilizar código
- Sin las limitaciones de la herencia
- Para código de propósito general
- Algo similar a herencia múltiple

# Mixins

```
var Mixin = function() {};
Mixin.prototype = {
  inspect: function() {
    var output = [];
    for(key in this) {
      output.push(key + ': ' + this[key]);
    }
    return output.join(', ');
  }
};
```

# Mixins

```
var Persona = Class.extend({
    init: function(nombre) {
        this.nombre = nombre;
    }
});

augment(Persona, Mixin);

var pepito = new Persona("Pepito");
pepito.inspect();
```

# Mixins

```
function augment(target, source) {  
    for (var prop in source.prototype) {  
        if(!target.prototype[prop]) {  
            target.prototype[prop] = source.prototype[prop];  
        }  
    }  
}
```

# Intermedio: Mejores Mixins

Lo podemos hacer mejor!

- Mejor integración con klass.js
- Callbacks de inclusión (estilo ruby)

# Intermedio: Mejores Mixins

```
var StaticModule = {  
  propiedadDeClase: "Soy una propiedad de clase",  
  included: function(klass) { console.log("Incluido!"); }  
};  
  
var InstanceModule = {  
  diHola: function() { alert("HOLA!"); },  
  mixed: function(klass) { console.log("Mezclado!"); }  
};  
  
var MiClase = Class.extend({  
  init: function() { }  
});  
  
MiClase.include(StaticModule);  
MiClase.mixin(InstanceModule);
```

# **Patrones de abstracción**

# Patrones de abstracción

- Iteradores
- Decorador
- Fachada
- Estrategia
- Inyección de dependencias
- Proxy

# Iteradores

Recorrer una colección

- Sin revelar detalles de implementación
- Mayor control sobre la iteración

# Iteradores

```
var ListadoAlumnos = Class.extend({  
    init: function() { this.alumnos = []; },  
    add: function(nombre, ciudad) {  
        this.alumnos.push({  
            nombre: nombre,  
            ciudad: ciudad  
        });  
    }  
});
```

# Iteradores

```
var ListadoAlumnos = Class.extend({  
    init: function() { this.alumnos = []; },  
    add: function(nombre, ciudad) {  
        this.alumnos.push({  
            nombre: nombre,  
            ciudad: ciudad  
        });  
    }  
});  
  
var lista = new ListadoAlumnos();  
lista.add("Gonzalo", "Madrid");  
lista.add("Gerardo", "Madrid");  
lista.add("Guzman", "Valencia");
```

# Iteradores

```
var lista = new ListadoAlumnos();
lista.add("Gonzalo", "Madrid");
lista.add("Gerardo", "Madrid");
lista.add("Guzman", "Valencia");
```

```
var alumnos = lista.alumnos;
for (var i=0; i<alumnos.length; i++) {
    if (alumnos[i].ciudad == "Madrid") {
        console.log(alumnos[i].nombre);
    }
}
```

# Iteradores

```
var lista = new ListadoAlumnos();  
lista.add("Gonzalo", "Madrid");  
lista.add("Gerardo", "Madrid");  
lista.add("Guzman", "Valencia");
```

```
var alumnos = lista.alumnos;  
for (var i=0; i<alumnos.length; i++) {  
    if (alumnos[i].ciudad == "Madrid") {  
        console.log(alumnos[i].nombre);  
    }  
}
```

# Iteradores

```
var ListadoAlumnos = Class.extend({
    init: function() { this.alumnos = []; },
    add: function(nombre, ciudad) {
        this.alumnos.push({ nombre: nombre, ciudad: ciudad });
    },
    forEachIn: function(ciudad, fn) {
        for (var i=0; i<this.alumnos.length; i++)
            if (this.alumnos[i].ciudad == ciudad) {
                fn(this.alumnos[i].nombre);
            }
    }
});
```



# Iteradores

```
var lista = new ListadoAlumnos();
lista.add("Gonzalo", "Madrid");
lista.add("Gerardo", "Madrid");
lista.add("Guzman", "Valencia");

lista.forEachIn("Madrid", function(n) {
    console.log(n);
});
```

# Iteradores

```
var ListadoAlumnos = Class.extend({  
    init: function() { this.alumnos = []; },  
    add: function(nombre, ciudad) {  
        this.alumnos.push({ nombre: nombre, ciudad: ciudad });  
    },  
    next: function() {  
        this.index = this.index || 0;  
        return this.alumnos[this.index++];  
    }  
});
```

# Iteradores

```
var lista = new ListadoAlumnos();
lista.add("Gonzalo", "Madrid");
lista.add("Gerardo", "Madrid");
lista.add("Guzman", "Valencia");
```

```
var alumno;
while(alumno = lista.next()) {
    alert(alumno.nombre);
}
```

# Iteradores

```
var Iterator = Class.extend({
    init: function(collection) {
        this.col = collection;
        this.pos = 0;
    },
    next: function() {
        return this.col[this.pos++];
    }
});

var ListadoAlumnos = Class.extend({
    init: function() { this.alumnos = []; },
    add: function(nombre, ciudad) {
        this.alumnos.push({ nombre: nombre, ciudad: ciudad });
    },
    getIterator: function() {
        return new Iterator(this.alumnos);
    }
});
```

# Iteradores

```
var lista = new ListadoAlumnos();
lista.add("Gonzalo", "Madrid");
lista.add("Gerardo", "Madrid");
lista.add("Guzman", "Valencia");

var alumno, iter = lista.getIterator();
while(alumno = iter.next()) {
    if (alumno.ciudad == "Madrid") {
        console.log(alumno.nombre);
    }
}
```

# Iteradores

```
var Fibonacci = Class.extend({
  init: function() {
    this.a = 0;
    this.b = 1;
  },
  next: function() {
    var next = this.a;
    this.a = this.b;
    this.b = this.b + next;
    return next;
  }
});
```

```
var iter = new Fibonacci();
for (var i=10; i--;) {
  console.log(iter.next());
}
```

# Estrategia

Seleccionar algoritmos dinámicamente

- Dada una familia de algoritmos
- Encapsulados bajo un mismo interfaz
- Elegir el más apropiado

# Estrategia

```
var validate = (function() {
  var validators = {
    email: function(value) {
      return /^[([^\<>()\\.,;:\\s@"]+([^.][^\<>()\\.,;:\\s@"]+)*)(\".+")@([([0-9]{1,3}\\.){3}[0-9]{1,3}\.){3}[a-zA-Z\-\_0-9]+\.)+[a-zA-Z]{2,})$/ .test(value);
    },
    number: function(value) {
      return /^d+$/ .test(value);
    }
  };
  return function(data) {
    var validation = validators[data.type];
    return validation && validation(data.value);
  };
}());

validate({type: "email", value: "eliasagc@gmail.com"});
```



# Estrategia

```
var MoodConsole = Class.extend({
    init: function() {
        this.state = "normal";
        this.filters = {
            normal: new DummyFilter(),
            relajado: new RelaxedFilter(),
            cabreado: new AngryFilter()
        };
    },
    log: function(msg) {
        msg = this.filters[this.state].filter(msg);
        console.log(msg);
    },
    molestar: function() { this.state = "cabreado"; },
    masajear: function() { this.state = "relajado"; },
    dejarEnPaz: function() { this.state = "normal"; }
});
```



# Estrategia

```
var DummyFilter = Class.extend({
    filter: function(s) { return s; }
});

var AngryFilter = DummyFilter.extend({
    filter: function(s) { return s.toUpperCase() + "!!"; }
});

var RelaxedFilter = DummyFilter.extend({
    filter: function(s) { return "..." + s + "..."; }
});

var moodConsole = new MoodConsole();
moodConsole.log("hola!");
moodConsole.molestar();
moodConsole.log("grrr...");
moodConsole.masajear();
moodConsole.log("gracias");
```



# Estrategia

¿Cuándo usar estrategias?

- Una misma acción puede tener varios significados
  - Según un estado variable (selector de contexto)
  - click, touch, una botón “Guardar”, etc..
- Validaciones
- En general: cuando hay que elegir un algoritmo diferente para cada caso

# Inyección de dependencias

Selección dinámica de un componente

- Según el principio de Inversión de Dependencias
- El cliente consume siguiendo un interfaz
- La clase concreta se selecciona en tiempo de ejecución

# Inyección de dependencias

```
var View = Class.extend({
    init: function(renderer, model) {
        this.renderer = renderer;
        this.model = model;
    },
    render: function() {
        this.renderer.render(this.template, this.model);
    }
});
```



```
var HtmlRenderer = Class.extend({
    render: function() { ... }
});
```

```
var XMLRenderer = Class.extend({
    render: function() { ... }
});
```

# Inyección de dependencias

```
var miVista = new View(new XMLRenderer('fast'), {});
```

# **Patrones de interacción**

# **Patrones de interacción**

- Pub/Sub u Observador
- Mediator
- Comandos y Cadena de Responsabilidad
- “Hydra”

# Observador

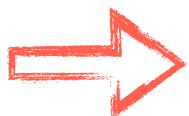
En vez de preguntar, escucha lo que sucede

- Muy común en JavaScript (eventos)
- Reaccionar ante cambios de estado o sucesos
- Dos entidades
  - Publicador
  - Suscriptor(es)

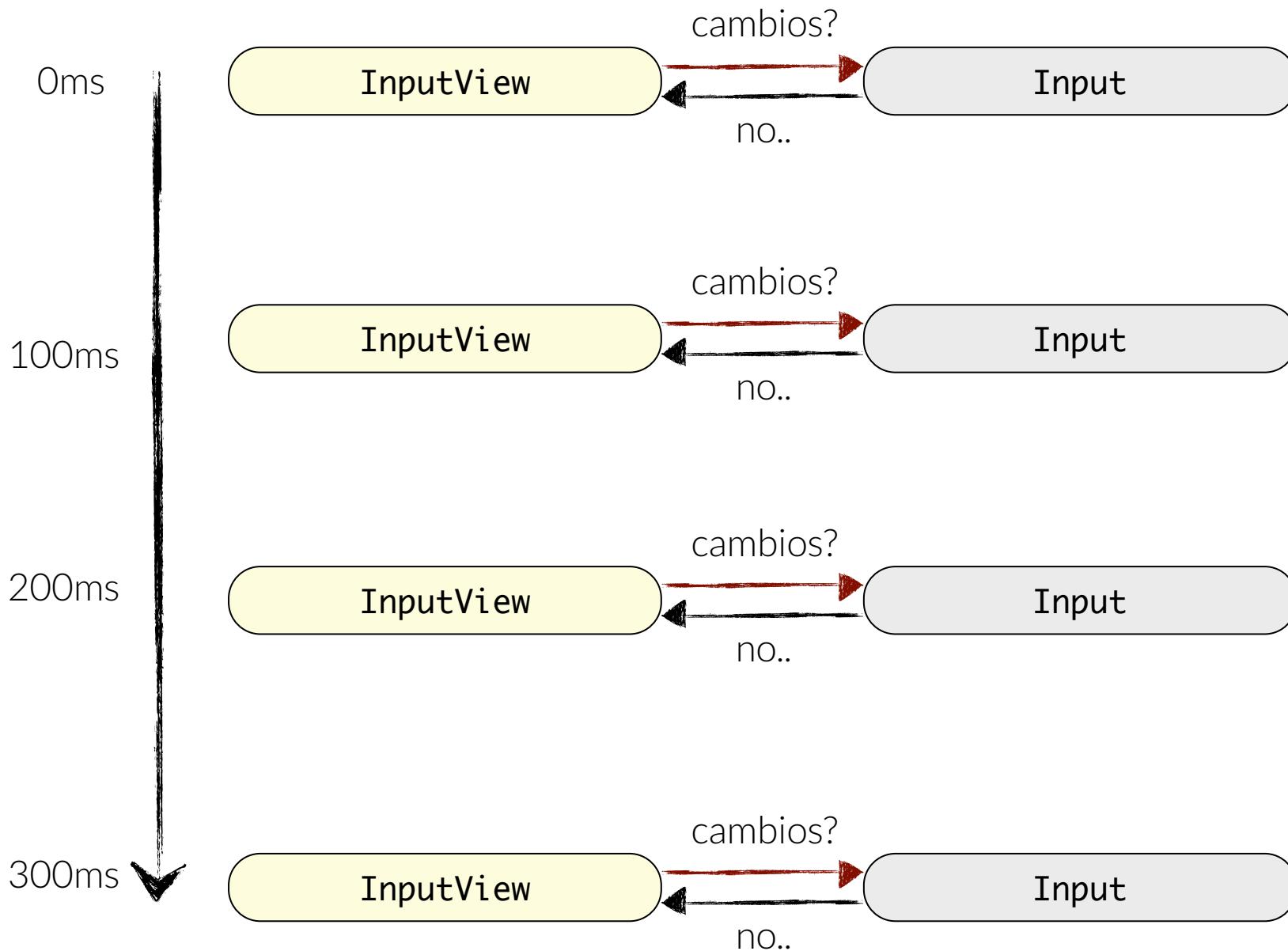
# Observador

```
var Input = Class.extend({
    getValue: function() { return this.value; }
});

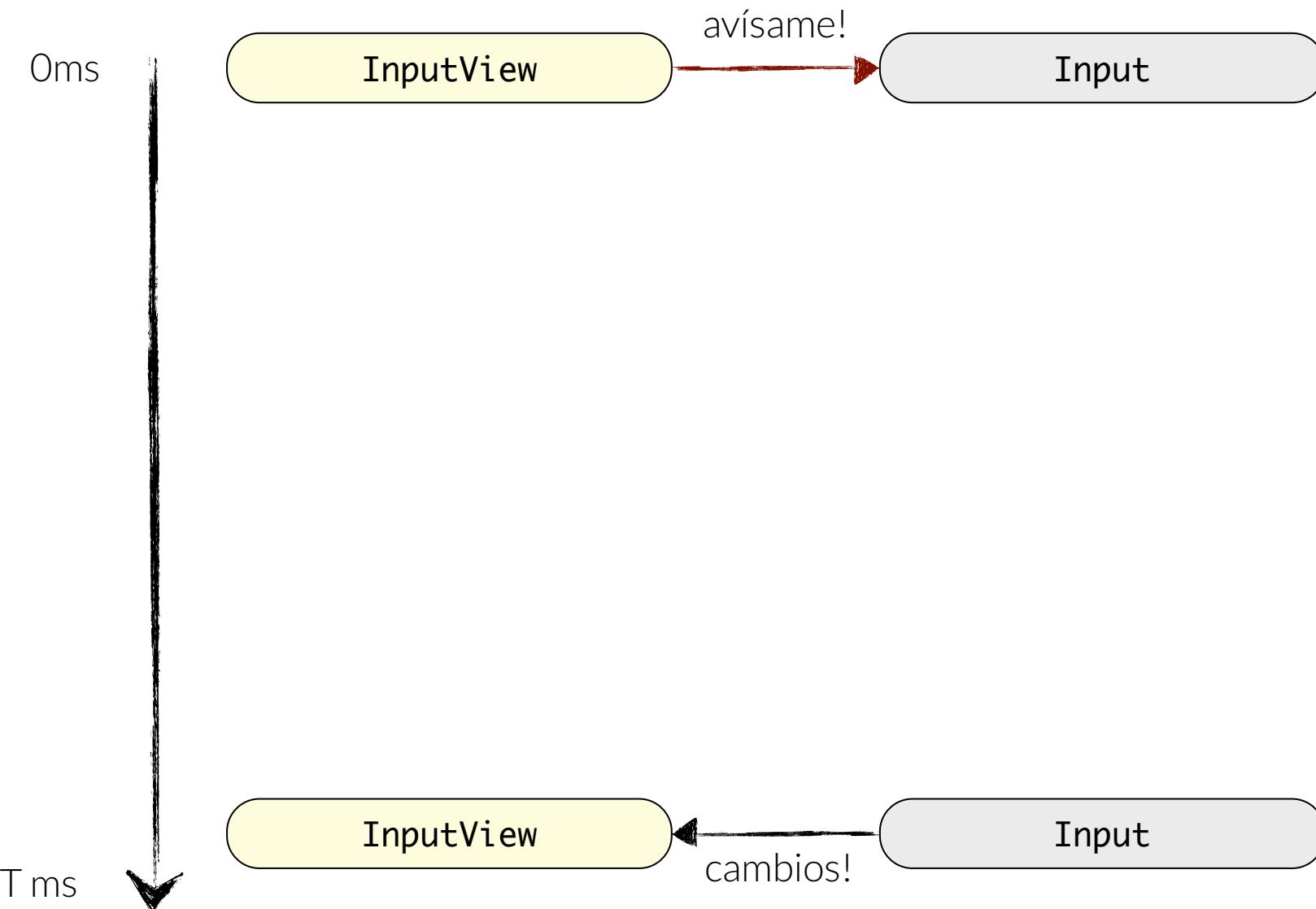
var InputView = Class.extend({
    init: function(input) {
        this.input = input;
        this.value = this.input.getValue();
        setInterval(bind(this, this.onTimer), 100);
    },
    onTimer: function() {
        var newValue = this.input.getValue();
        if (newValue !== this.value) {
            this.value = newValue;
            console.log("CHANGED: " + this.value);
        }
    }
});
```



# Observador



# Observador

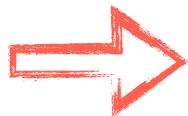


# Observador

```
var Input = Class.extend({
    init: function() {
        this.observers = [];
    },
    getValue: function() {
        return this.value;
    },
    setValue: function(v) {
        this.value = v;
        this.publish(v);
    },
    onChange: function(cb) {
        this.observers.push(cb);
    },
    publish: function(cb) {
        this.observers.forEach(function(o) { o(cb); });
    }
});
```

# Observador

```
var Input = Class.extend({
    init: function() {
        this.observers = [];
    },
    getValue: function() {
        return this.value;
    },
    setValue: function(v) {
        this.value = v;
        this.publish(v);
    },
    onChange: function(cb) {
        this.observers.push(cb);
    },
    publish: function(cb) {
        this.observers.forEach(function(o) { o(cb); });
    }
});
```



# Observador

```
var InputView = Class.extend({
    init: function(input) {
        this.input = input;
        this.input.onChange(bind(this, this.update));
    },
    update: function(newValue) {
        console.log("CHANGED: " + newValue);
    }
});
```

# Observador

```
var input = new Input();  
  
var observer1 = new InputView(input);  
var observer2 = new InputView(input);  
var observer3 = new InputView(input);  
  
input.setValue("No te duermas!");
```

# Intermedio: Observable

¡Podemos hacerlo mejor!

```
Input.mixin(Observable);
```

```
var InputView = Class.extend({
  init: function(input) {
    this.input.subscribe('change', bind(this, this.update));
    this.input.subscribe('invalid', bind(this, this.invalid));
  },
  update: function(newValue) {
    console.log("CHANGED: " + newValue);
  },
  invalid: function(value) {
    console.log("El valor " + value + " es inválido!");
  }
});
```

# Intermedio: Observable

```
var Observable = {  
  mixed: function(klass) {  
    // ???  
  },  
  subscribe: function(event, callback) {  
    // ???  
  },  
  unsubscribe: function(event, callback) {  
    // ???  
  },  
  publish: function(event) {  
    // ???  
  }  
};
```

# Observador

¿Cuándo utilizar Observador?

- Se utiliza muy a menudo!
- Propagar cambios
- Reaccionar a sucesos
- Comunicación uno-a-muchos

# **Mediator**

Un punto central de control del sistema

- Desacoplamiento a gran escala
- Sencillo y muy importante
- Dividir la lógica en dos niveles
  - Componente
  - Aplicación

# Mediator

Home Connect Discover Me Search

**Elias Alonso**  
[View my profile page](#)

135 TWEETS    160 FOLLOWING    55 FOLLOWERS

Compose new Tweet...

Who to follow · [Refresh](#) · [View all](#)

- Chad Fowler** @chadfowler  
Followed by Enrique Garcia  
[Follow](#)
- jQuery Style** @jQueryStyle  
Followed by Pedro Martin  
[Follow](#)
- Digg** @digg  
Followed by Digg\_Updates  
[Follow](#)

[Browse categories](#) · [Find friends](#)

Trends · [Change](#)

## Tweets

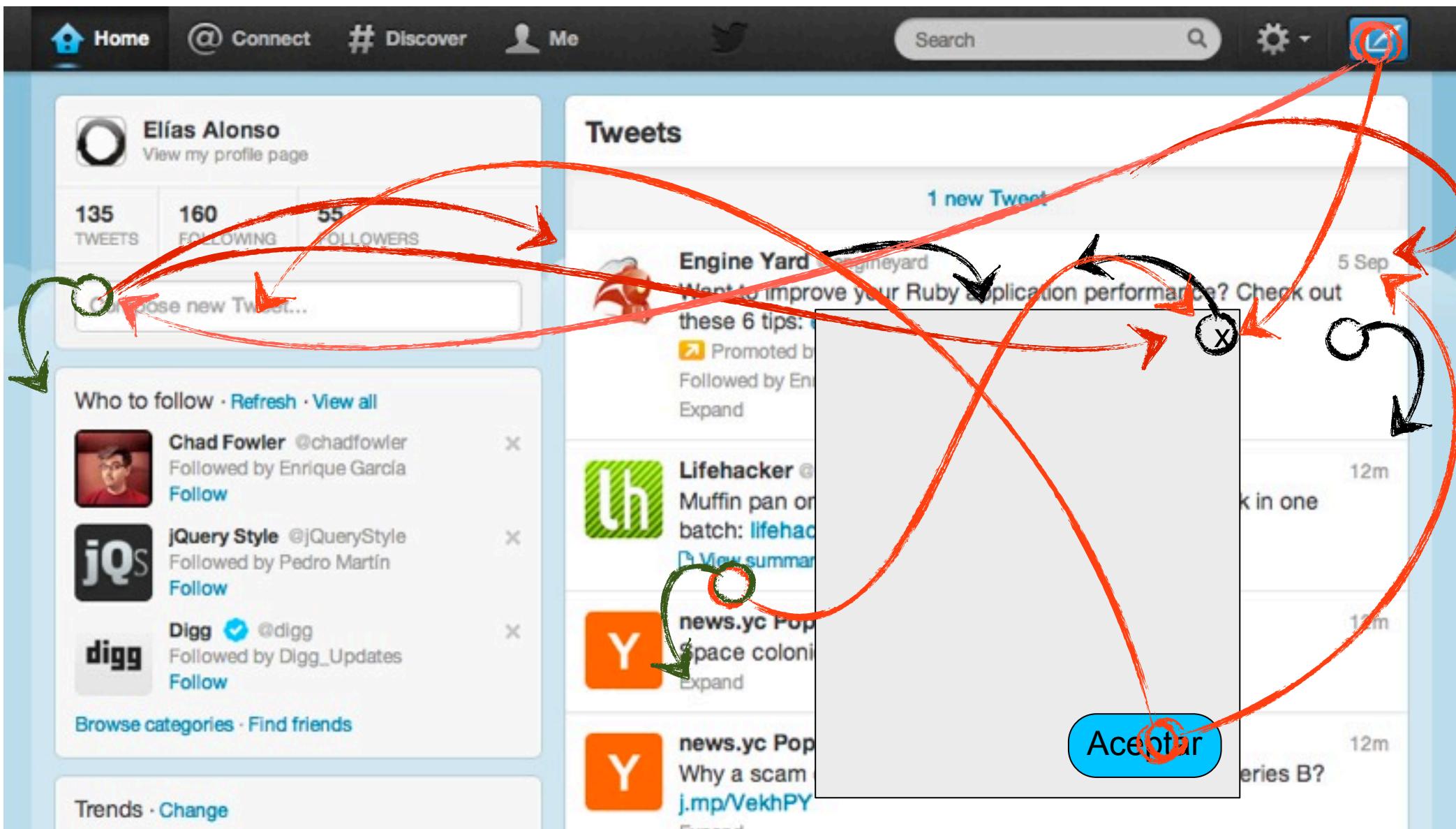
1 new Tweet

- Engine Yard** @engineyard 5 Sep  
Want to improve your Ruby application performance? Check out these 6 tips: [ey.io/RaHPoP](http://ey.io/RaHPoP)  
 Promoted by Engine Yard  
Followed by Enrique Macip, dotemacs and Rendered Text.  
[Expand](#)
- Lifehacker** @lifehacker 12m  
Muffin pan omelets let you make breakfast for the week in one batch: [lifehac.kr/IVYdp](http://lifehac.kr/IVYdp)  
[View summary](#)
- news.yc Popular** @newsycombinator 12m  
Space colonies of the 1970s [j.mp/pzsqnh](http://j.mp/pzsqnh)  
[Expand](#)
- news.yc Popular** @newsycombinator 12m  
Why a scam company was able to raises \$76 Million Series B? [j.mp/VekhPY](http://j.mp/VekhPY)

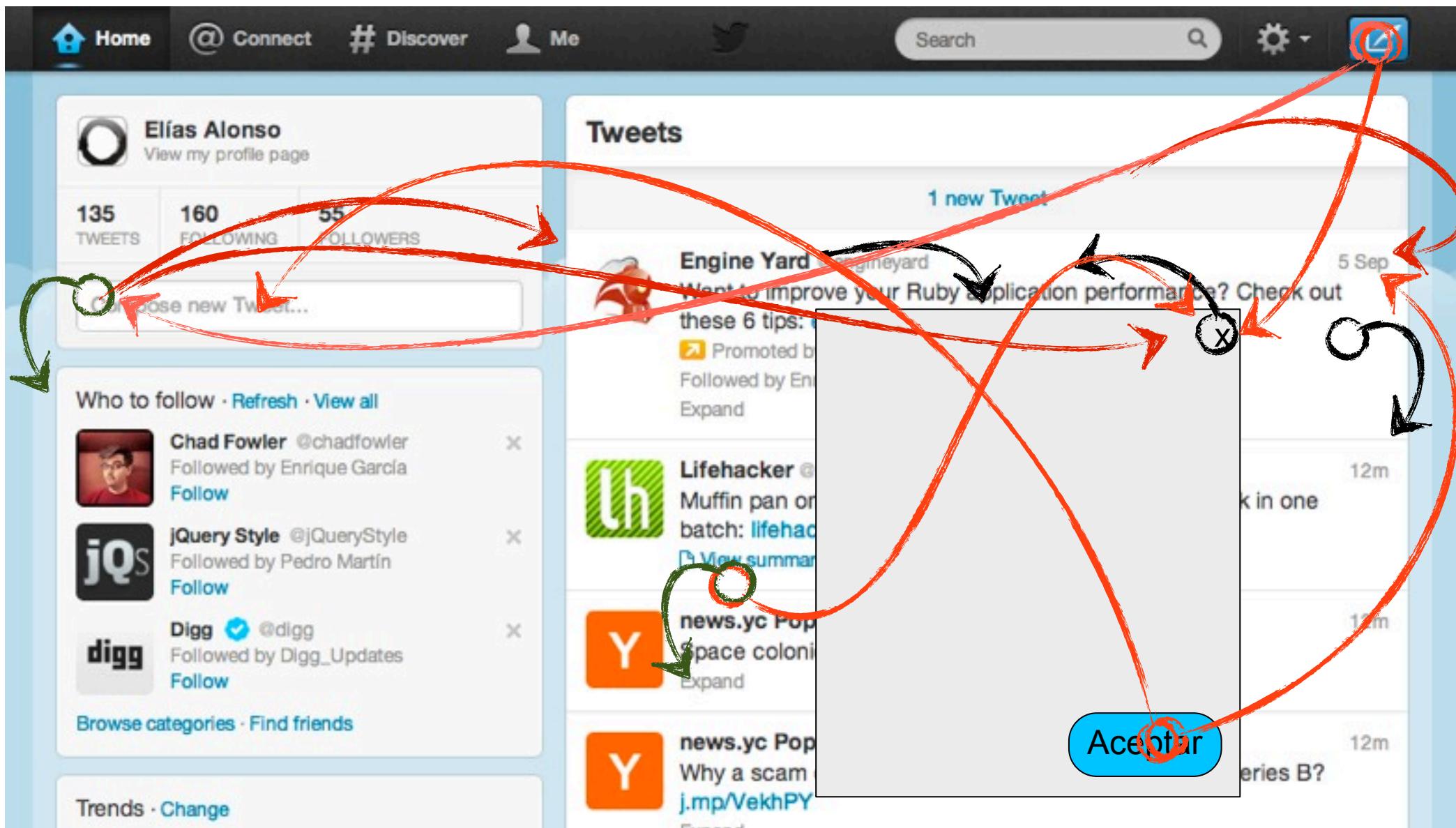
# Mediator

The image shows a screenshot of a Twitter-like application interface. On the left, there's a sidebar with a user profile for "Elías Alonso" (135 tweets, 160 following, 55 followers), a "Compose new Tweet..." input field, a "Who to follow" section listing "Chad Fowler", "jQuery Style", and "digg", and links for "Browse categories", "Find friends", and "Trends". On the right, the main area is titled "Tweets" and shows a single tweet from "Engine Yard" (@engineyard) posted on "5 Sep". The tweet content is: "Want to improve your Ruby application performance? Check out these 6 tips: [http://t.co/...](#)". Below the tweet, there are three more entries from "news.yc" with "Pop" status. Hand-drawn annotations include a red circle around the "Compose new Tweet..." field, several black arrows showing data flow between the sidebar and the main tweets area, and a large red circle highlighting the "Engine Yard" tweet.

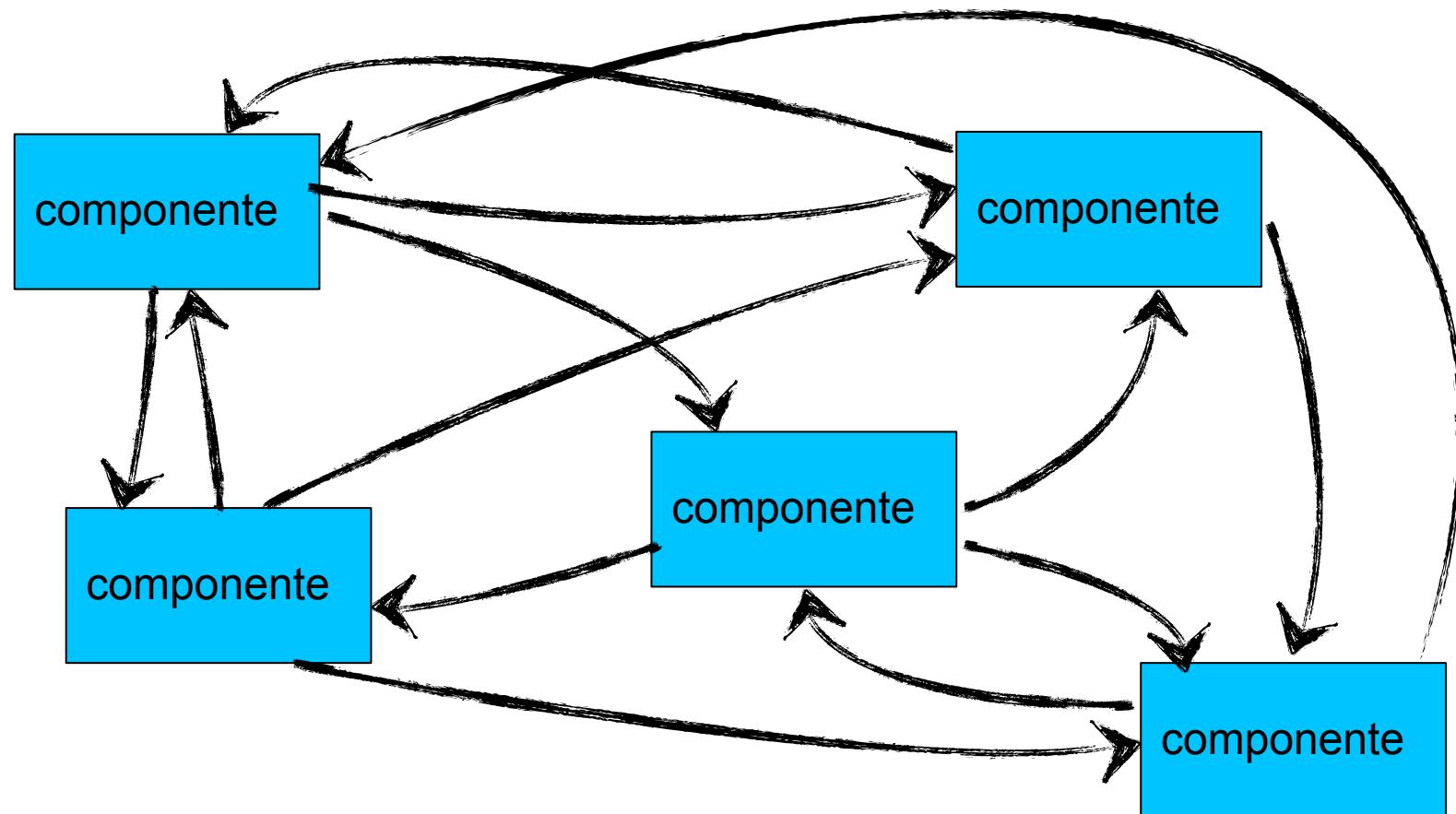
# Mediator



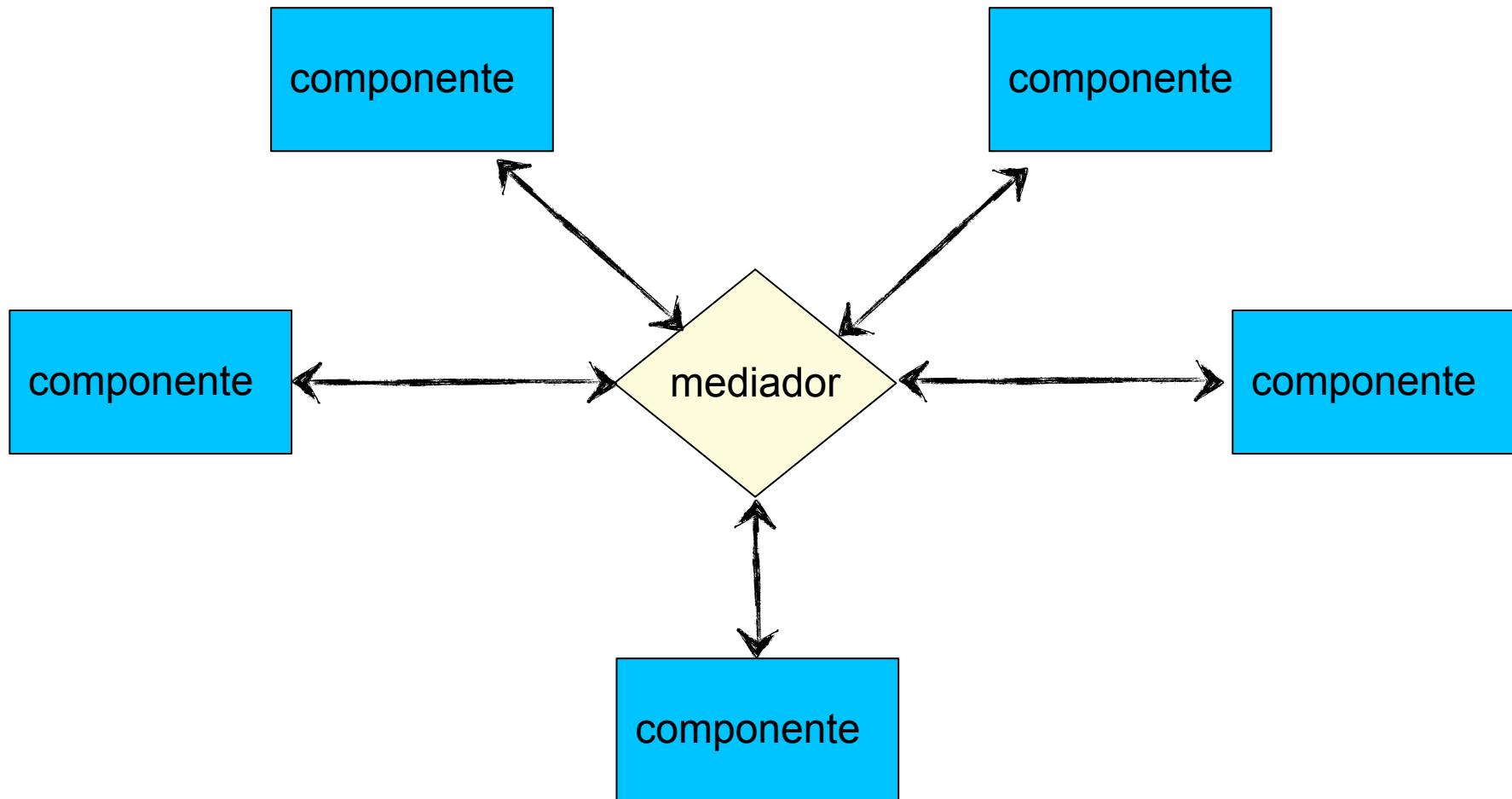
# Mediator



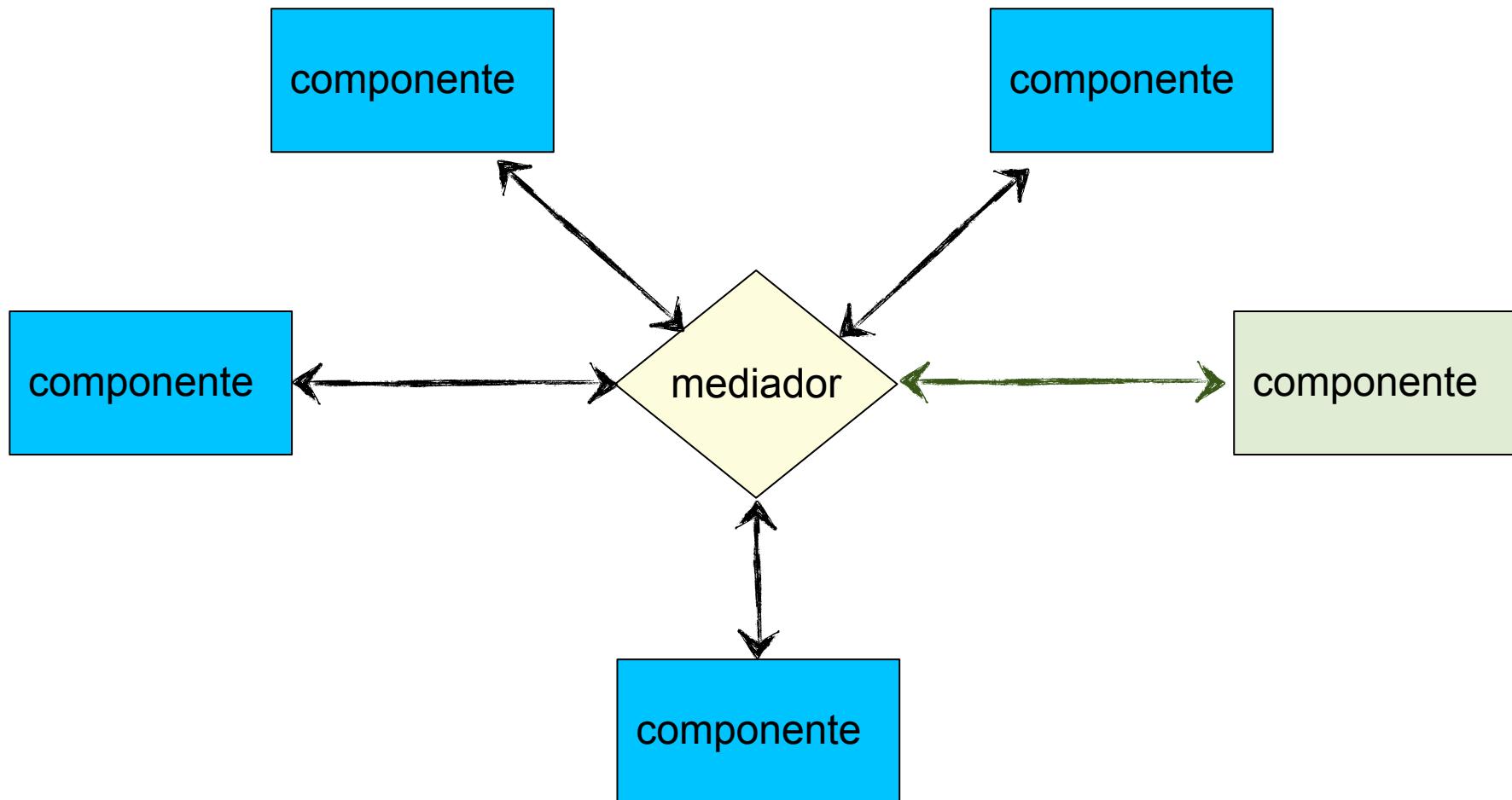
# Mediator



# Mediator



# Mediator



# Mediator

```
var Mediator = Class.extend({
    init: function(fn) {
        this.connections = {};
        if (fn) fn(this);
    },
    on: function(msg, cb) {
        this.connections[msg] = cb;
    },
    send: function(msg) {
        var args = [].slice.call(arguments, 1),
            action = this.connections[msg];
        if (action) action.apply({}, args);
    },
    addComponent: function(component) {
        component.setMediator(this);
    }
});
```

# Intermedio: Mediator

- Mediador en acción:
  - tema2/mediador-1/index.html

# Intermedio: Mediator

```
// Inicialización

$(function() {

    var mediator = new Mediator(),
        button1 = new Button('pulsador', '#button-1');

    mediator.addComponent(button1);
    mediator.on('pulsador:clicked', function() {
        alert("hola?");
    });

});
```

# Intermedio: Mediator

```
// Inicialización

$(function() {

    var mediator = new Mediator(function(mediator) {
        var button1 = new Button('pulsador', '#button-1');

        mediator.addComponent(button1);
        mediator.on('pulsador:clicked', function() {
            alert("hola?");
        });
    });
});
```



# Intermedio: Mediator

- Mediador en acción:
  - tema2/mediador-2/index.html

# Intermedio: Mediator

```
mediator.on('ignore:on', function() {  
    enabled = false;  
});  
  
mediator.on('ignore:off', function() {  
    enabled = true;  
});  
  
mediator.on('inc:clicked', function() {  
    if (enabled) { count.increment(); }  
    display.update(count.getCurrentValue());  
});  
  
mediator.on('dec:clicked', function() {  
    if (enabled) { count.decrement(); }  
    display.update(count.getCurrentValue());  
});
```

# Intermedio: Mediator

```
mediator.on('ignore:on', function() {  
    enabled = false;  
});
```

```
mediator.on('ignore:off', function() {  
    enabled = true;  
});
```

```
mediator.on('inc:clicked', function() {  
    if (enabled) { count.increment(); }  
    display.update(count.getCurrentValue());  
});
```

```
mediator.on('dec:clicked', function() {  
    if (enabled) { count.decrement(); }  
    display.update(count.getCurrentValue());  
});
```



# **Mediator**

¿Cuándo usar un Mediador?

- Muchos componentes interactuando
- La interacción se puede separar del componente
- Es decir: coordinar diferentes widgets de la aplicación

# **Mediator**

¿Cuándo usar un Mediador?

- Muchos componentes interactuando
- La interacción se puede separar del componente
- Es decir: coordinar diferentes widgets de la aplicación

¡Cuidado!

- El mediador tiende a convertirse en un cajón de sastre
- Una Gran Función que Todo lo Hace!

# Comandos

Separar la preparación de una acción y su ejecución

- Flujos de ejecución no convencionales
- Tres actores: cliente, invocador, receptor
  - cliente: prepara o configura una acción
  - invocador: controla la ejecución
  - receptor: lleva a cabo la acción
- Generalizar comportamientos

# **Comandos**

tema2/comandos-1/index.html

# Comandos

```
var Command = Class.extend({
    init: function(msg) { this.msg = msg; },
    execute: function() { alert(this.msg); }
});

var ActionList = Class.extend({
    init: function(selector) { this.ul = $(selector); },
    append: function(name, command) {
        var li = $("<li>")
            .append($('<a/>', {html:name, href:'#'}))
            .click(bind(command, command.execute))
            .appendTo(this.ul);
    }
});

var list = new ActionList('#menu');
list.append('saludo', new Command("Hola!"));
list.append('achis!', new Command("Salud!"));
```

# Comandos

```
var Command = Class.extend({  
    init: function(msg) { this.msg = msg; },  
    execute: function() { alert(this.msg); }  
});
```



```
var ActionList = Class.extend({  
    init: function(selector) { this.ul = $(selector); },  
    append: function(name, command) {  
        var li = $("<li>")  
            .append($('<a/>', {html:name, href:'#'}))  
            .click(bind(command, command.execute))  
            .appendTo(this.ul);  
    }  
});
```



```
var list = new ActionList('#menu');  
list.append('saludo', new Command("Hola!"));  
list.append('achis!', new Command("Salud!"));
```



# Comandos

```
var AlertCommand = Class.extend({  
    init: function(msg) { this.msg = msg; },  
    execute: function() {  
        alert(this.msg);  
    }  
});
```



```
var LogCommand = Class.extend({  
    init: function(msg) { this.msg = msg; },  
    execute: function() {  
        console.log(this.msg);  
    }  
});
```



# Comandos

Encapsular las acciones y controlar su ejecución

- Una idea muy potente
- Deshacer! (estados reversibles)
- Historial
- Control de cambios
- Sincronización de estados

# Comandos

Un ejemplo sencillo de estado reversible:

- tema2/comandos-2/index.html

```
var Movement = Class.extend({
    init: function(amount, axis) {
        this.amount = amount;
        this.axis = axis;
    },
    execute: function(mosca) {
        mosca.move(this.amount, this.axis);
    },
    undo: function(mosca) {
        mosca.move(-this.amount, this.axis);
    }
});
```



# Comandos

¿Cuándo usar Comandos?

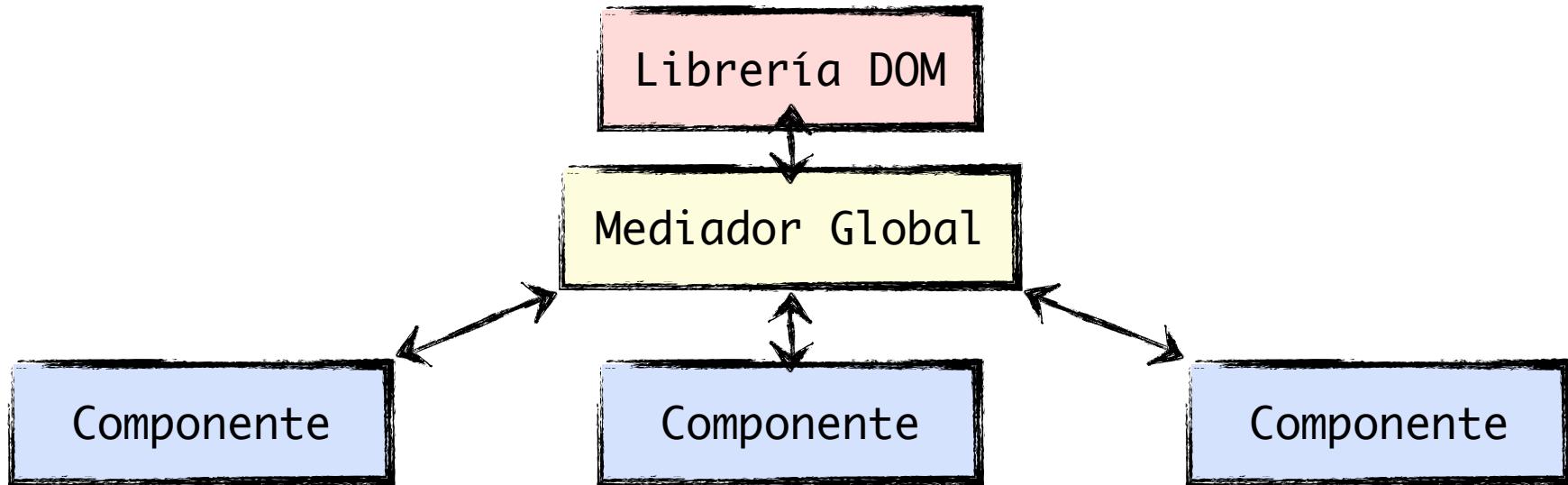
- Controlar la ejecución de acciones
- Separar la definición de la acción y su ejecución
- Es decir:
  - Intervienen actores que pueden fallar (servidor, usuario)
  - Menús, selectores y otros “contenedores de acciones”
  - Sincronizar cambios simultáneos (control de versiones)

# Hydra

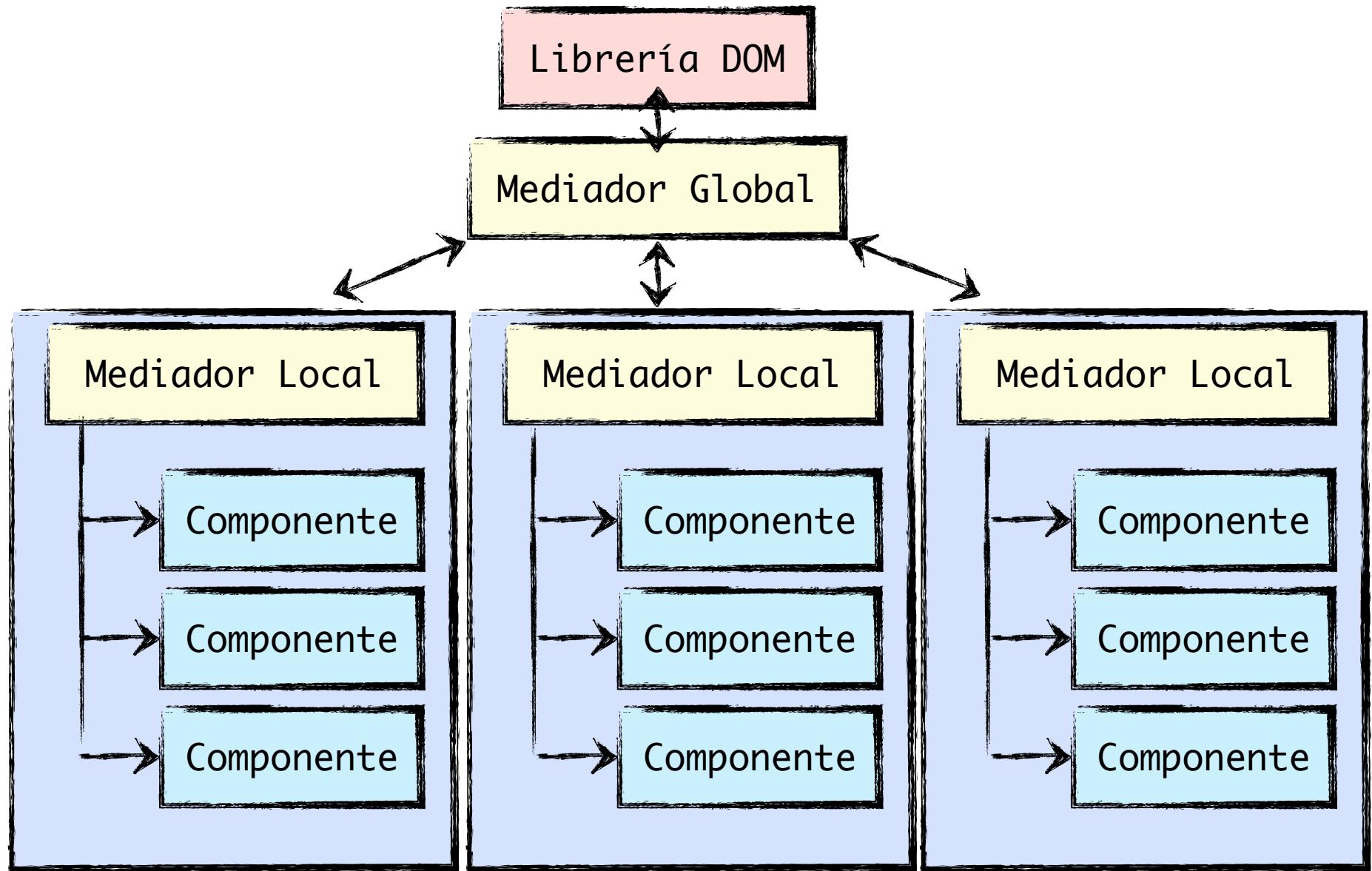
## Árbol de comportamientos

- No existe
- Organización al más alto nivel
- Mediador + Comandos + Cadena de Responsabilidad
- Extremadamente escalable!
- Para aplicaciones muy grandes

# Hydra



# Hydra



# Hydra

Librería DOM

Ajax + DOM

Mediador Global

Coordinación general

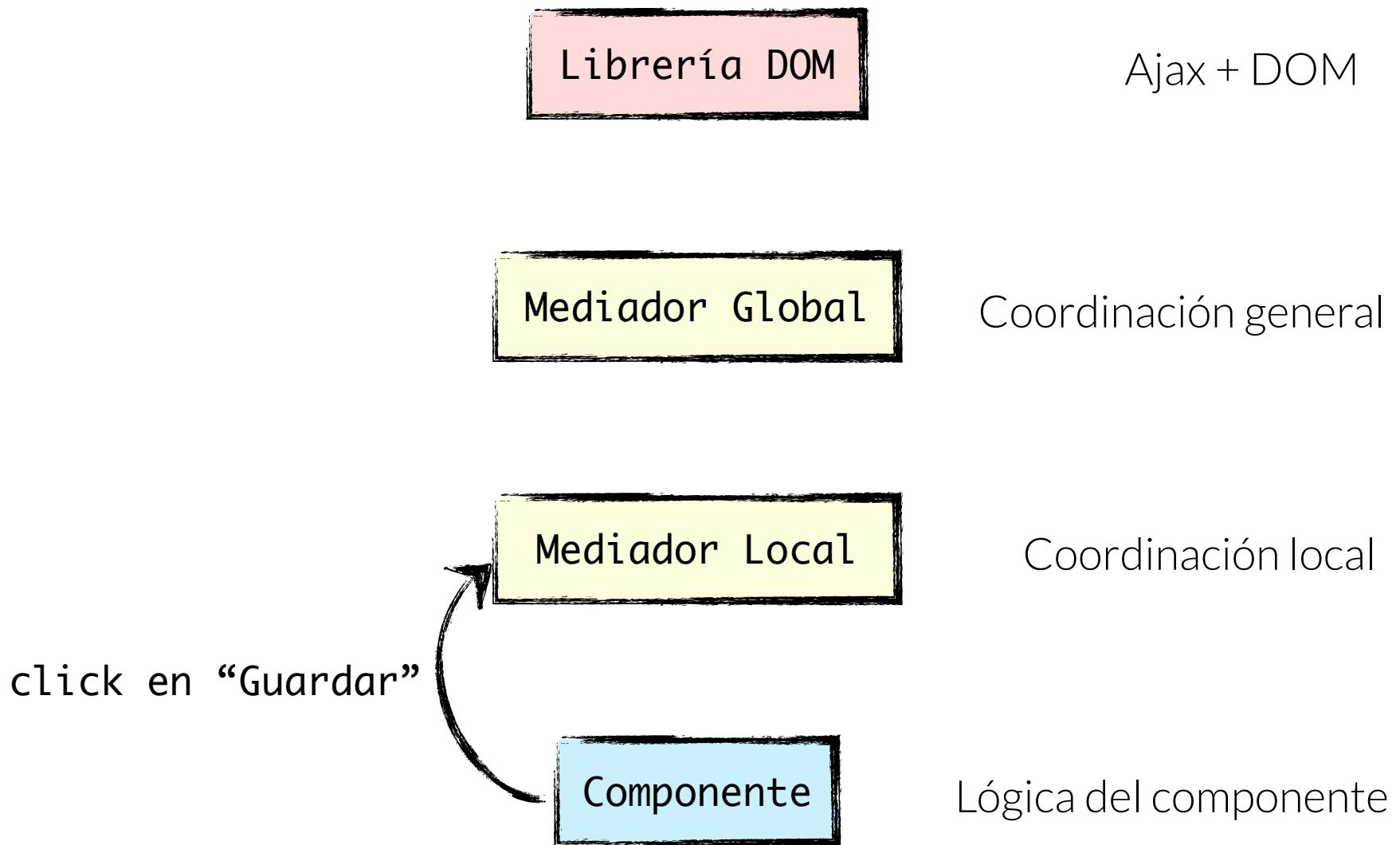
Mediador Local

Coordinación local

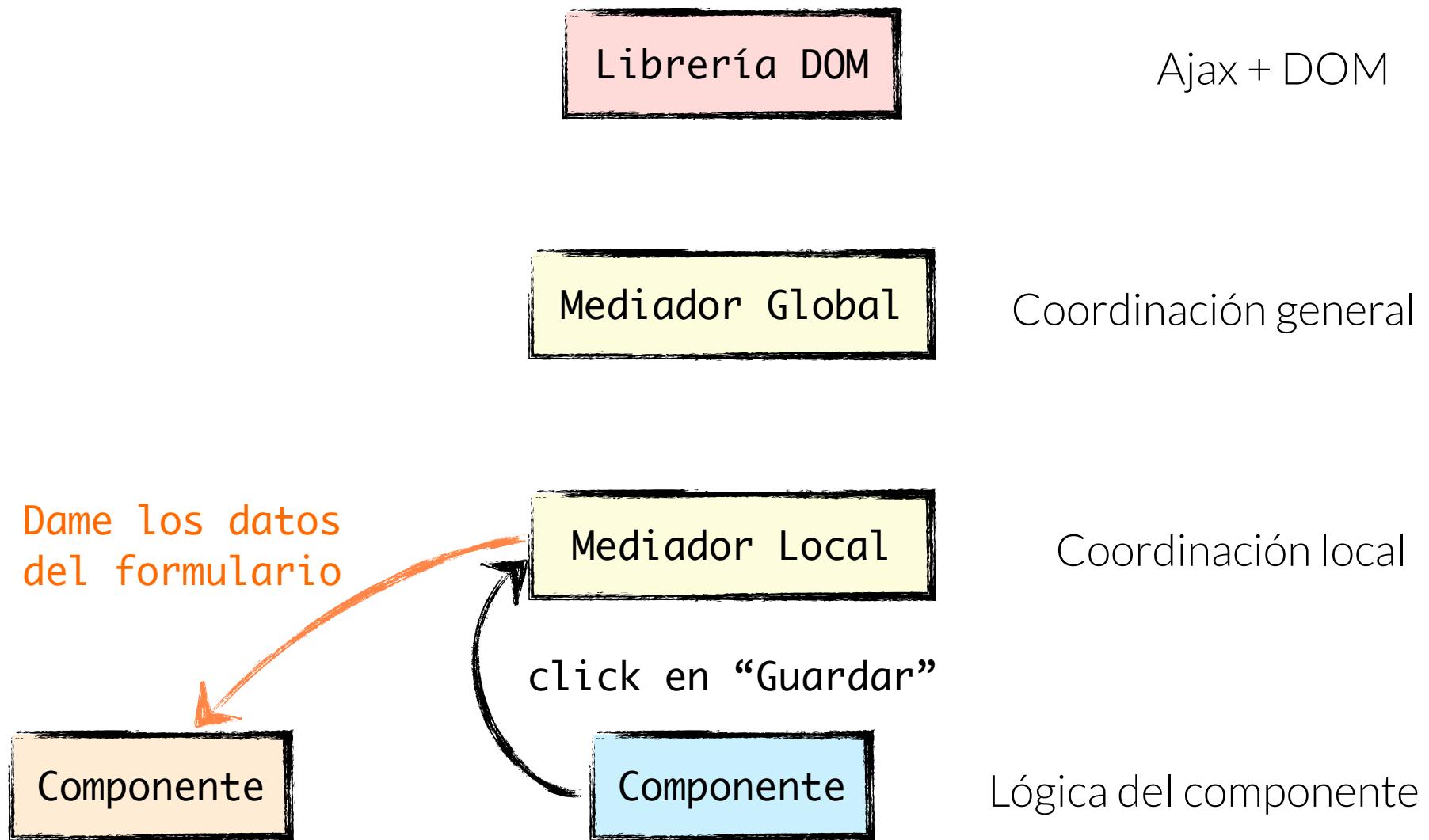
Componente

Lógica del componente

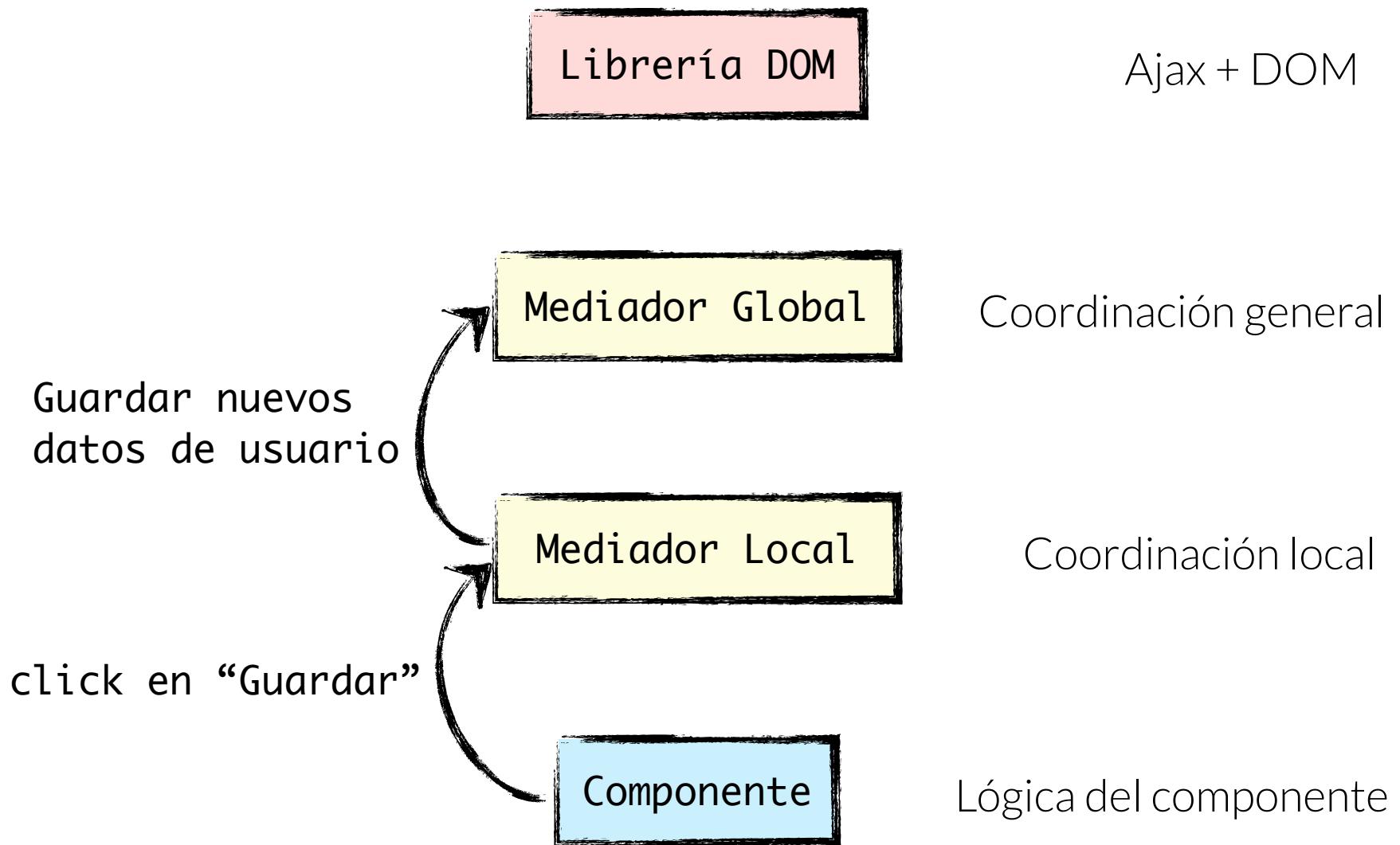
# Hydra



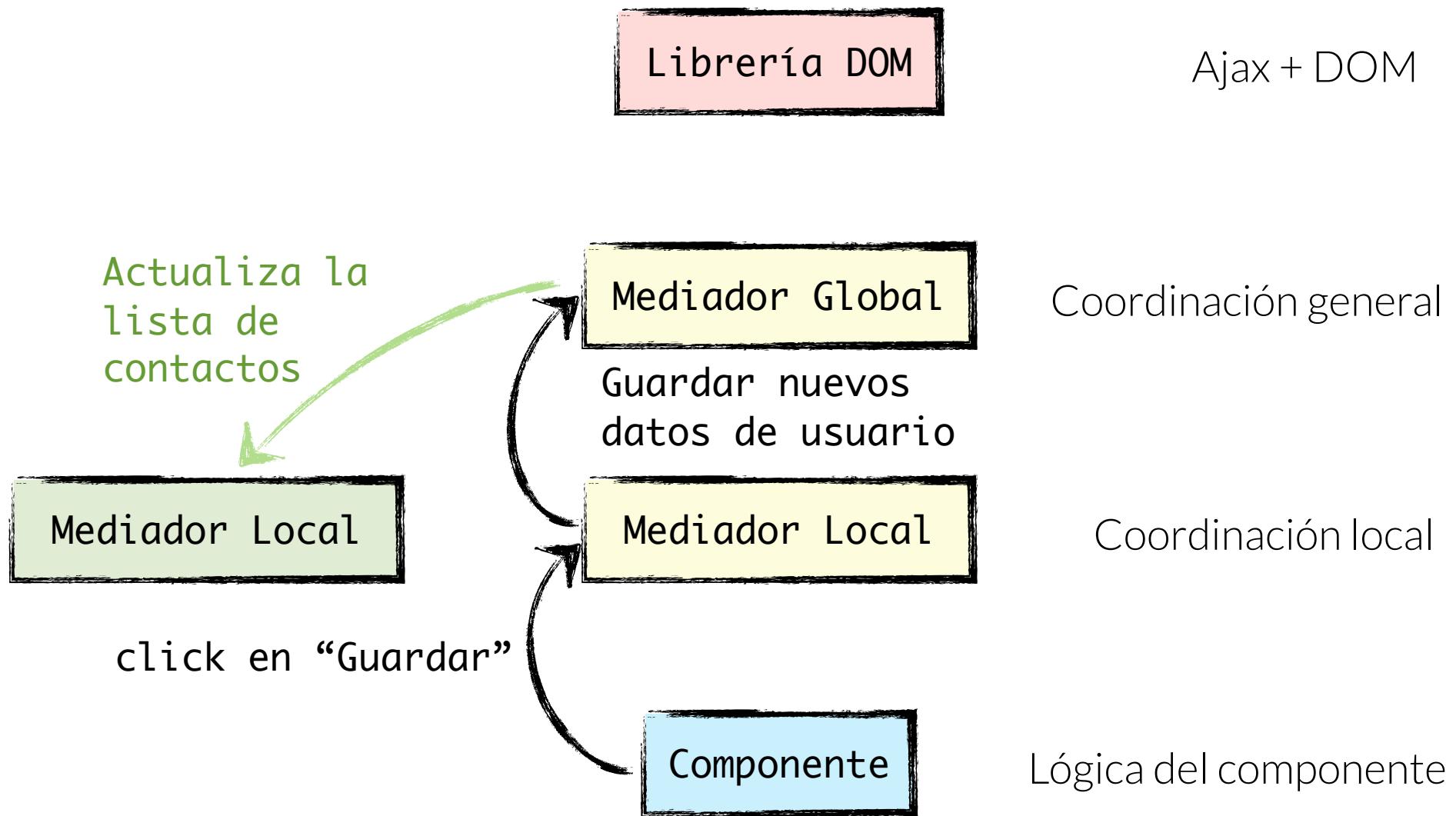
# Hydra



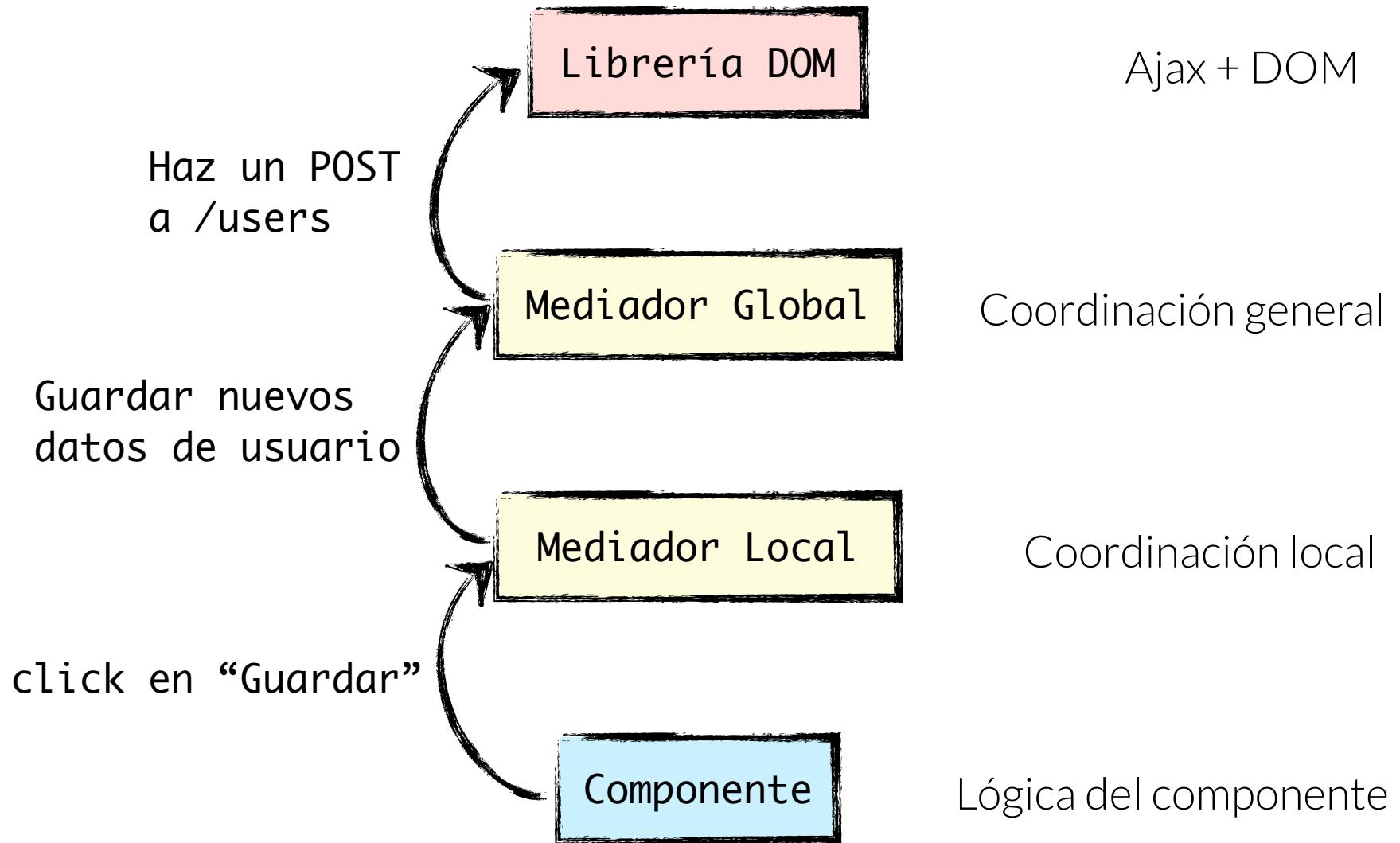
# Hydra



# Hydra



# Hydra



# **Hydra**

## Separación por capas de significado

- Cada mediador gestiona lo que hay a su nivel
- El significado de las acciones va cobrando sentido
- La lógica de los componentes se limita a hablar con su mediador
- La coordinación es “recursiva”

# Hydra

¿Cuándo utilizar Hydra?

- Aplicaciones grandes!
- Widgets complicados que se beneficien de mediación
- En general, en cuanto un mediador engorde demasiado

# Hydra

¿Cuándo utilizar Hydra?

- Aplicaciones grandes!
- Widgets complicados que se beneficien de mediación
- En general, en cuanto un mediador engorde demasiado

¡Cuidado!

- El alto desacoplamiento hace complicado leer el código
- El significado TOTAL de cada acción está diseminado