Programación Web

Introducción a la Web

Programación Web

Breve repaso
Tecnologías Web

Tecnologías Web - Tim Berners-Lee

- **Tres componente definen la esencia de la tecnología Web:**

1. Un **lenguaje de marcado** para formatear documentos de hipertexto: HTML (HyperText Markup Language)

2. Una manera de localizar los recursos sobre la red: **URL** (Uniform Resource Locator)

3. Un **protocolo** para transportar **mensajes** sobre la red: HTTP

Arquitecturas Web básicas - Base

Request HTTP

Response (HTML/XML)

Response (HTML/XML)

```
<HTML>
<HEAD>
<TITLE> Apache Tomcat 4.1.2 </TITLE>
</HEAD>

<BODY>

</BODY>
</HTML>
```

Servidor

Antes de la primer arquitectura

- **Sitios Web estáticos**
  - Recursos de información
    - Solo lectura

- Aplicación Web vs. Sitio Web
  - Contenido dinámico vs. estático

- Veamos algunos ejemplos…

Ejemplos actuales de aplicaciones Web

- Aerolíneas Argentinas

Ejemplos actuales de aplicaciones Web

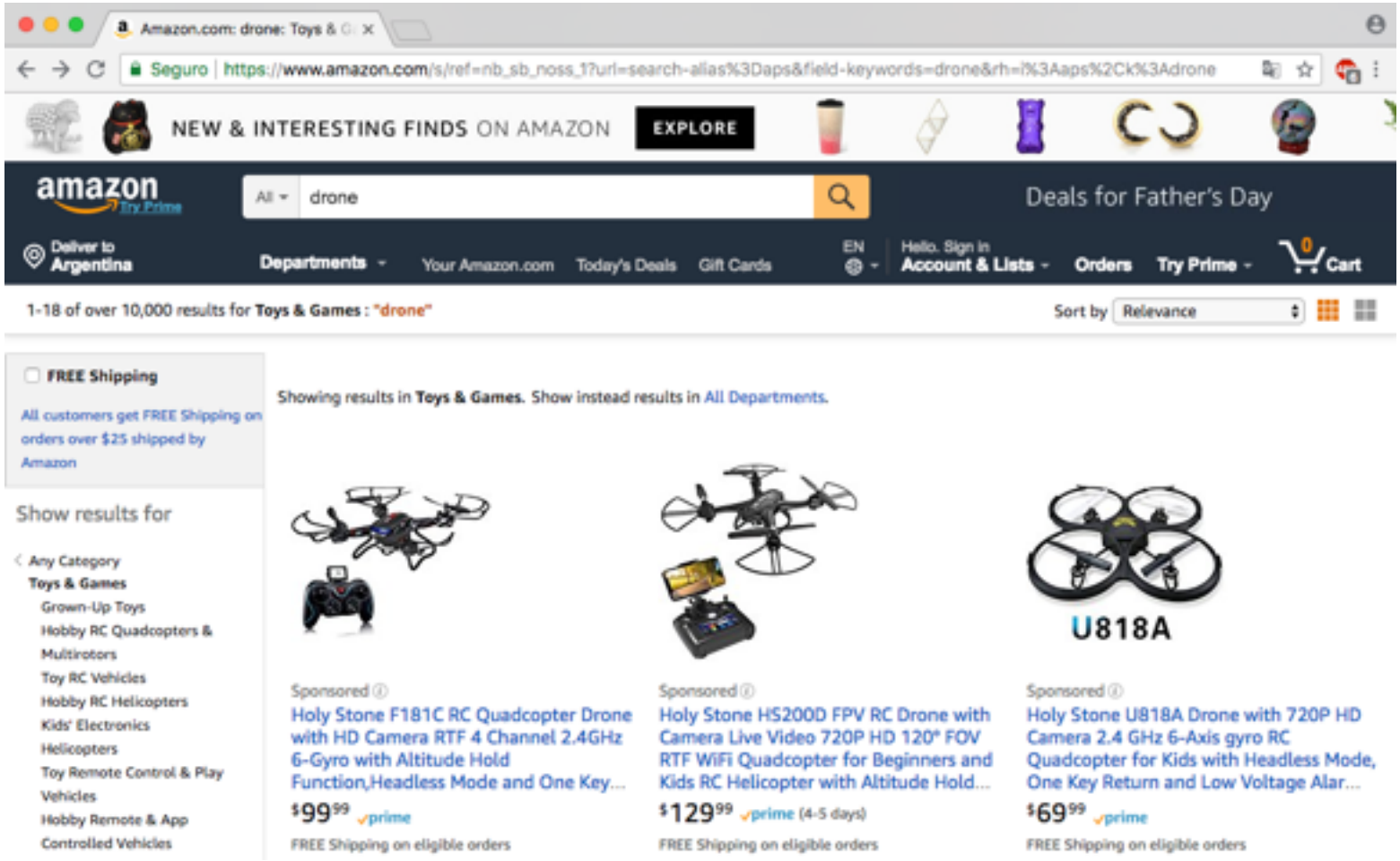
- Aerolíneas Argentinas

Ejemplos actuales de aplicaciones Web

- **Netflix**

Ejemplos actuales de aplicaciones Web

- Amazon



- Client-Side:
  - De base: HTML, CSS, JavaScript
  - Technology Stack: jQuery, Bootstrap, Angular, etc.

- Server-Side:
  - Capa de servicio:
    - Servidores Web
    - CGI
  - Capa de aplicación
    - Seaside, Django, Spring, Rails, Symfony, etc.
      - ¿Qué tienen en común?

- ¿Cómo se llegó hasta acá?
- ¿Qué cosas vinieron después?

Programación Web

Introducción a las arquitecturas Web

Arquitecturas Web básicas

- Sitios Web estáticos
  - Pros:
    - Sin carga computacional, indexación, caché
  - Contras:
    - Caro de actualizar, sin personalización, sin soporte a procesos de negocio

Arquitecturas Web básicas

- **Sitios dinámicos**
  - **Actividades en el servidor**
    - Manejar el request HTTP.
    - **Generar una respuesta para el request HTTP.**
    - Retornar la respuesta.

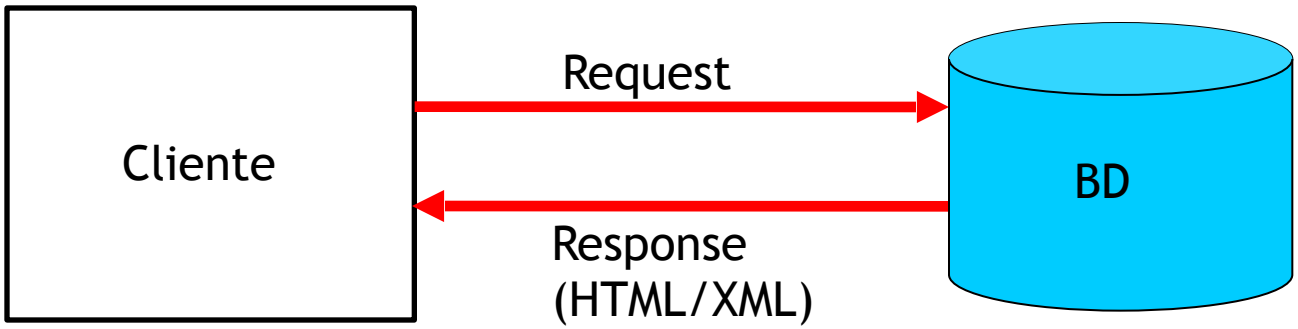
- Primeras arquitecturas
  - CGI
    - Contenido dinámico!

```
#!/usr/local/bin/perl
# hello.pl - My first CGI program

print "Content-Type: text/html\n\n";
# Note there is a newline between
# this header and Data

# Simple HTML code follows

print "<html> <head>\n";
print "<title>Hello, world!</title>";
print "</head>\n";
print "<body>\n";
print "<h1>Hello, world!</h1>\n";
print "</body> </html>\n";
```

¿Esto es el controlador?

- En general, mezclaban tanto la lógica de la aplicación como la generación de los documentos HTML.

- Oracle procedure



```
PROCEDURE pagina_tp_materia (el_dniE IN VARCHAR2,
 id_sesion IN VARCHAR2, la_materia IN VARCHAR2,
 el_plan IN VARCHAR2,la_sede IN VARCHAR2) IS…
```

```
--Cargo los datos de la materia seleccionada

                SELECT *
                INTO mat
                FROM MATERIA
                WHERE idmateria=la_materia;
```

```
htp.print('<TABLE width=90%>');
htp.print('<tr>');
htp.print('<td align="Center">');
htp.print('<font SIZE=4>Acta de Trabajos
 Prácticos y Promoción</font>');
htp.print('</tr>');htp.print('<tr>');
htp.print('<td align="Center">');
htp.print('<font SIZE=4><b><u>Asignatura: '||
 mat.nombre|| '</u></b></font>');
htp.print('</tr>');htp.print('<tr>');
htp.print('<td align="Center">');
htp.print('<font SIZE=2>Carrera: '||
 reg_plan.nombre_carrera||'-'||reg_plan.anio||'</
 font>');
htp.print('</
 tr>');htp.print('<tr>');htp.print('<td
 align="Center">');
```

¿Hay modelo?

```
<h2>PHP Form Validation Example</h2>
<p><span class="error">* required field</span></p>
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
  Name: <input type="text" name="name" value="<?php echo $name;?>">
  <span class="error">* <?php echo $nameErr;?></span>
  <br><br>
  E-mail: <input type="text" name="email" value="<?php echo $email;?>">
  <span class="error">* <?php echo $emailErr;?></span>
  <br><br>
  Website: <input type="text" name="website" value="<?php echo $website;?>">
  <span class="error"><?php echo $websiteErr;?></span>
  <br><br>
  Comment: <textarea name="comment" rows="5" cols="40"><?php echo $comment;?></textarea>
  <br><br>
  Gender:
  <input type="radio" name="gender" <?php if (isset($gender) && $gender=="female") echo "checked";?> value="female">Female
  <input type="radio" name="gender" <?php if (isset($gender) && $gender=="male") echo "checked";?> value="male">Male
  <input type="radio" name="gender" <?php if (isset($gender) && $gender=="other") echo "checked";?> value="other">Other
  <span class="error">* <?php echo $genderErr;?></span>
  <br><br>
  <input type="submit" name="submit" value="Submit">
</form>

<?php
echo "<h2>Your Input:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
```

¿Hay vista?

Arquitecturas Web básicas - Primeras arquitecturas

- **Desventajas:**
  - Alto acoplamiento, difícil reutilización
  - Conocimiento en todas las capas de la aplicación PL/SQL, BD, HTML, diseño gráfico
  - Mantenimiento complejo
  - Fuerte dependencia de la base de datos

Arquitecturas Web básicas - Sites dinámicos

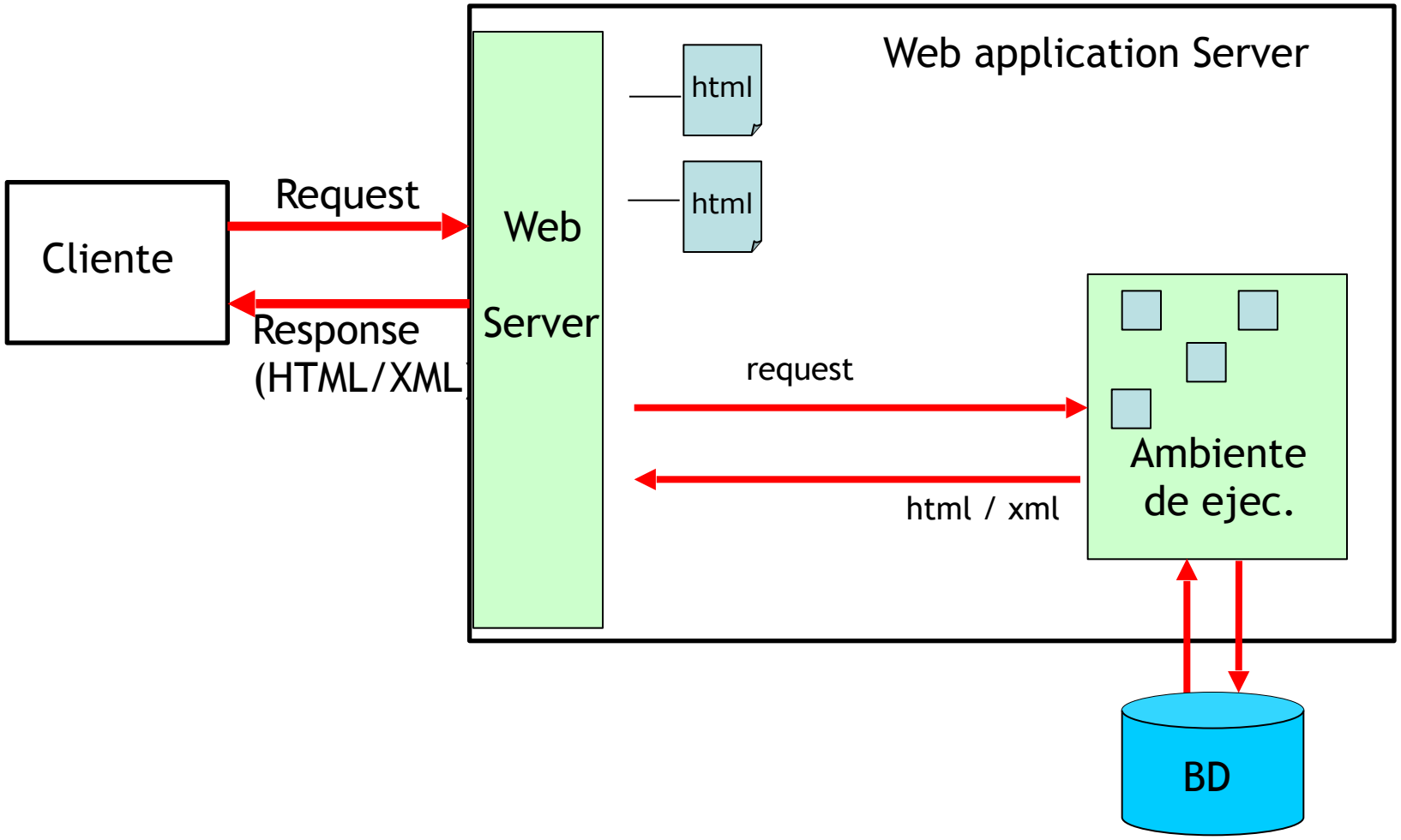
- **Pensando en los ejemplos vistos…**
  - **Actividades del servidor**
    - Manejar el request HTTP.
    - **Generar una respuesta para el request HTTP.**
    - Retornar la respuesta.

- **Generar una respuesta para el request HTTP involucra tareas complicadas:**
  - Recuperar/almacenar datos de una base de datos.
  - Aplicar complicadas reglas de negocio.
  - Personalizar contenido y aspectos de la UI a cada usuario.

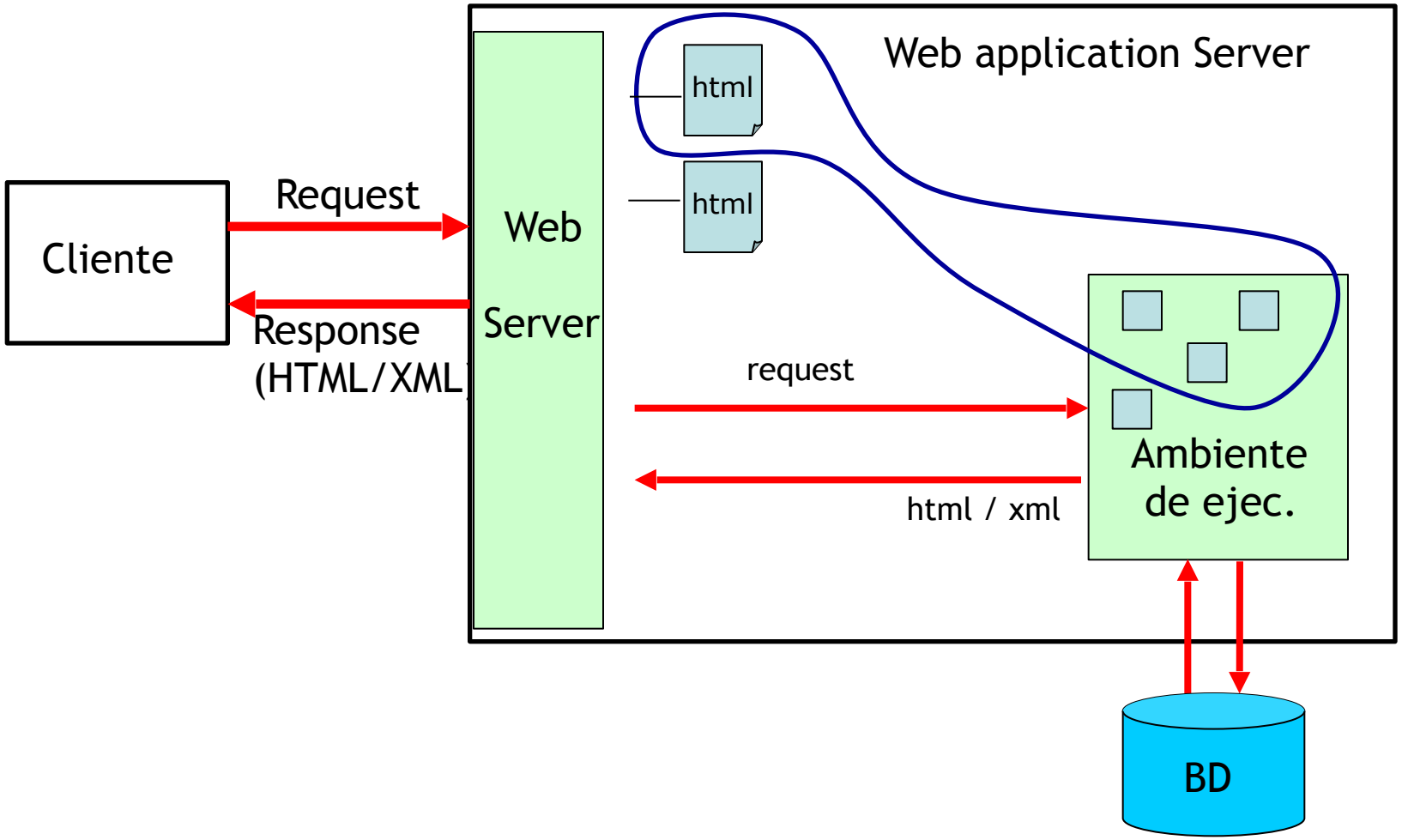
Arquitecturas Web básicas - Fundamentos

- **Uso de capas para separar responsabilidades**
- **Mayor nivel de abstracción, que nos permitan hacer aplicaciones escalables, mantenibles, reutilizables.**
- Crecimiento de la web —> **Modelos de dominio complejos**
  - **Idealmente querríamos poder interactuar con un modelo de objetos**
  - OOP: En los 60 ya habían tecnologías maduras
  - WWW: Recién nace en el 89
    - Principios de los 90, contenidos dinámicos
    - Incorporación de OOP en la Web
      - Cierta herencia de OOH

Arquitecturas Web básicas - Segunda arquitectura



Arquitecturas Web básicas - Segunda arquitectura



¿Cómo debería ser la parte que maneja la lógica de aplicación y genera la UI?

- Escalable
- De fácil mantenimiento
- Bajo acoplamiento
- Que facilite el reuso
- etc.

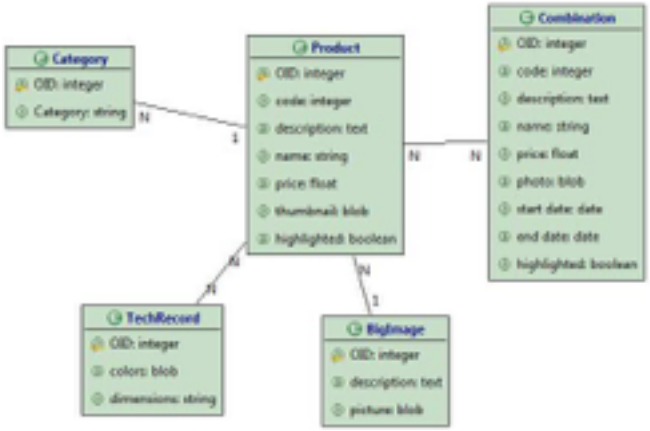
¿Cómo debería ser la parte que maneja la lógica de aplicación y genera la UI?

- Escalable
- De fácil mantenimiento
- Bajo acoplamiento
- Que facilite el reuso
- etc.

Model-View-Controller

Arquitectura - Patrón de diseño MVC

- El **Modelo**: el modelo de objetos que hacemos habitualmente.

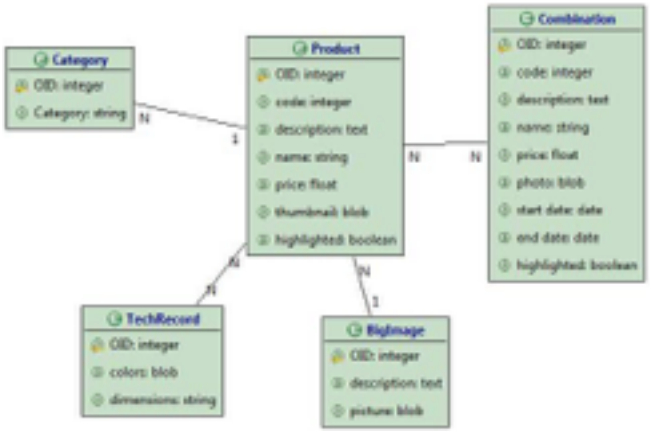


- El **Controlador**: responde a eventos, envía mensajes al modelo.

- La **Vista**: es la presentación visual al usuario de la aplicación.

Arquitectura - Patrón de diseño MVC, reinterpretación

- El Modelo: el modelo de objetos que hacemos habitualmente.



- El Controlador: es quien define cómo se responde a los requests HTTP, que se desencadenan con las acciones del usuario.

- La Vista: es el componente que genera la vista para al usuario de la aplicación.

Ejemplo de Framework MVC: django

Modelo

```
class Todo(models.Model):
        title = models.CharField(max_length=100)
        description = models.CharField(max_length=500)
        completed = models.BooleanField()


        def __str__(self):
            return self.title

        def toggle(self):
            self.completed = not self.completed
```

Ejemplo de Framework MVC: django

URL Dispatcher

```
#http://www.myapp.com/

urlpatterns = patterns('',
    url(r'^$',   home),
    url(r'^todoitems/$',        list_todoitems),
    url(r'^todoitem/(?P<item_id>\d+)/details/$',    details_todoitem)
)
```

Ejemplo de Framework MVC: django

Controlador

```
def list_todoitems(request):
        items_list = Todo.objects.all()
        return render_to_response(“home.html”, {’ items_list ': items_list ,})


def details_todoitem (request, item_id):
        poll = Todo.objects.get(id=item_id)
        return render_to_response(“item_details.html”, {’item': item ,})
```

Ejemplo de Framework MVC: django

Vista

```
<html>
<head>
        <title>ToDo list</title>
</head>
<body>
        <h1>ToDo list</h1>
        <ul>
                {% for item in items_list %}
                   <li>
                      {{ item.title }} - Description: {{ item.description }}
                   </li>
                {% endfor %}
        </ul>
</body>
</html>
```

Ejemplo de Framework MVC: django

Vista

```
<html>
<head>
        <title>ToDo list</title>
</head>
<body>
        <h1>ToDo list</h1>
        <ul>
                {% for item in items_list %}
                    <li>
                     {{ item.title }} - Description: {{ item.description }}
                   </li>
                {% endfor %}
        </ul>
</body>
</html>
```

ToDo list

- Estudiar Seaside – Description: Hacer tutorial indicado por la catedra
- Crear app Seaside – Description: Crear el componente root y configurar app

Repasando...

Repasando...

Repasando...

Client-Side

Server-Side

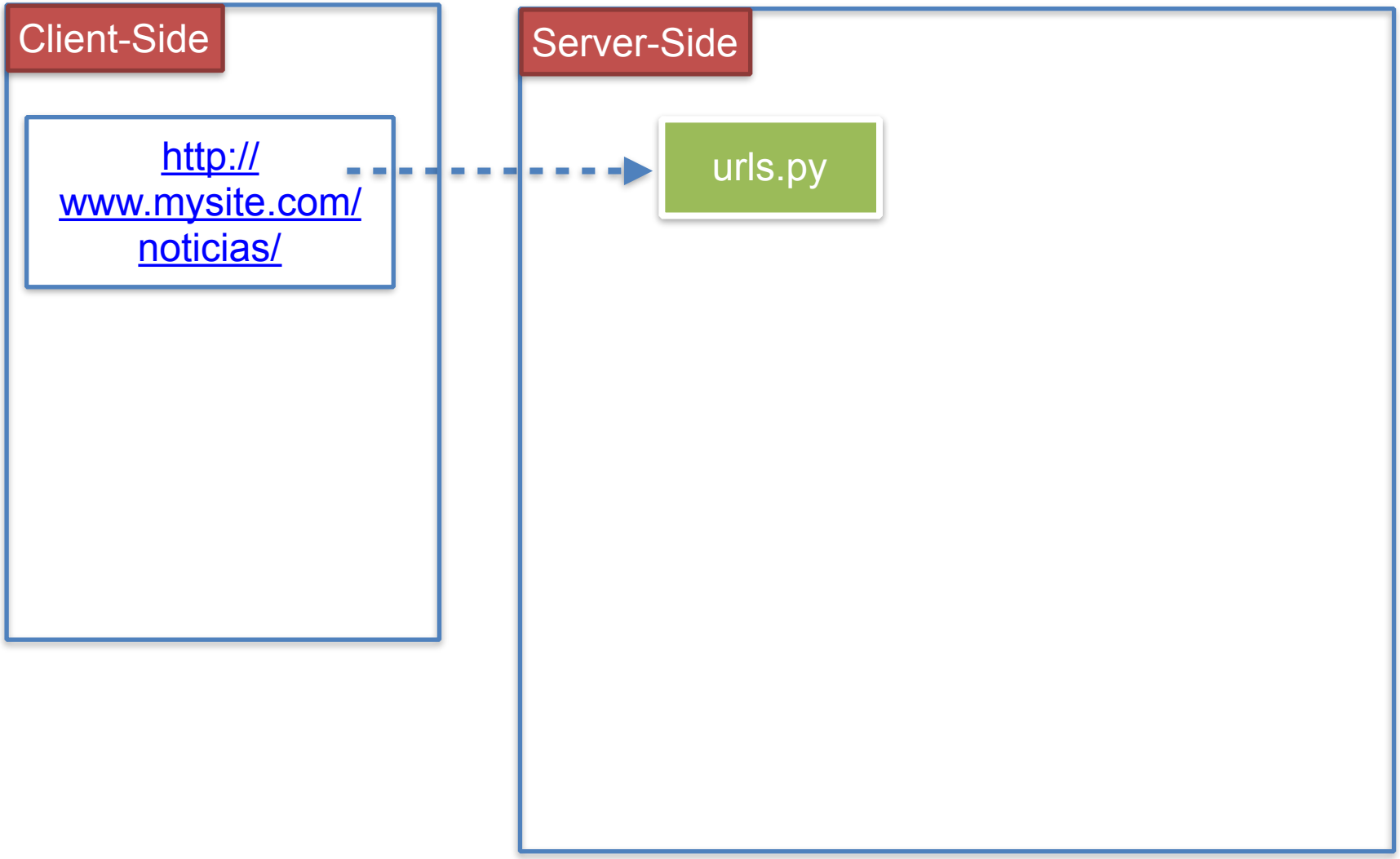
Repasando...

Client-Side

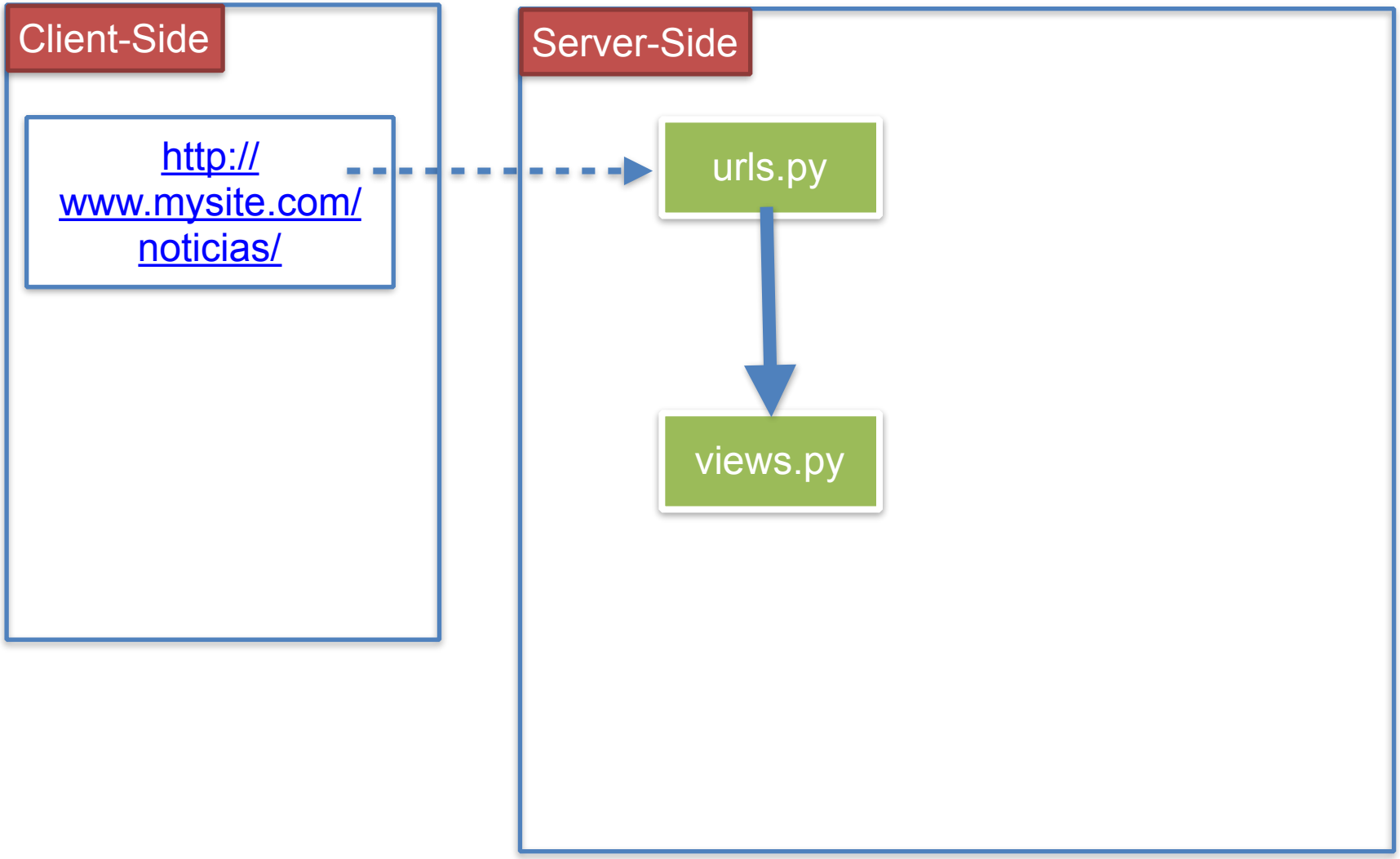
http://
www.mysite.com/
noticias/

Server-Side

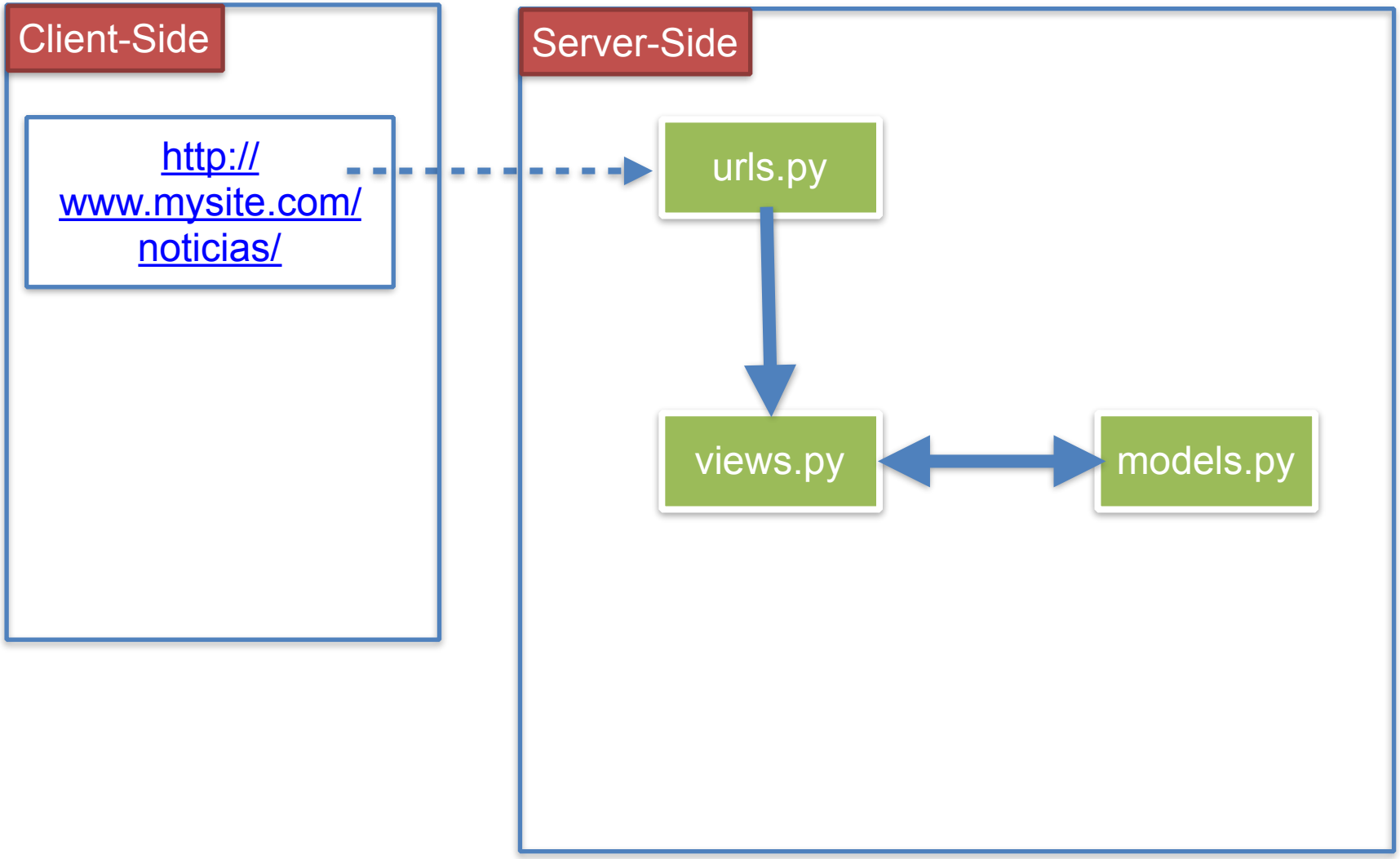
Repasando...


Client-Side
http://
www.mysite.com/
noticias/
Server-Side
urls.py

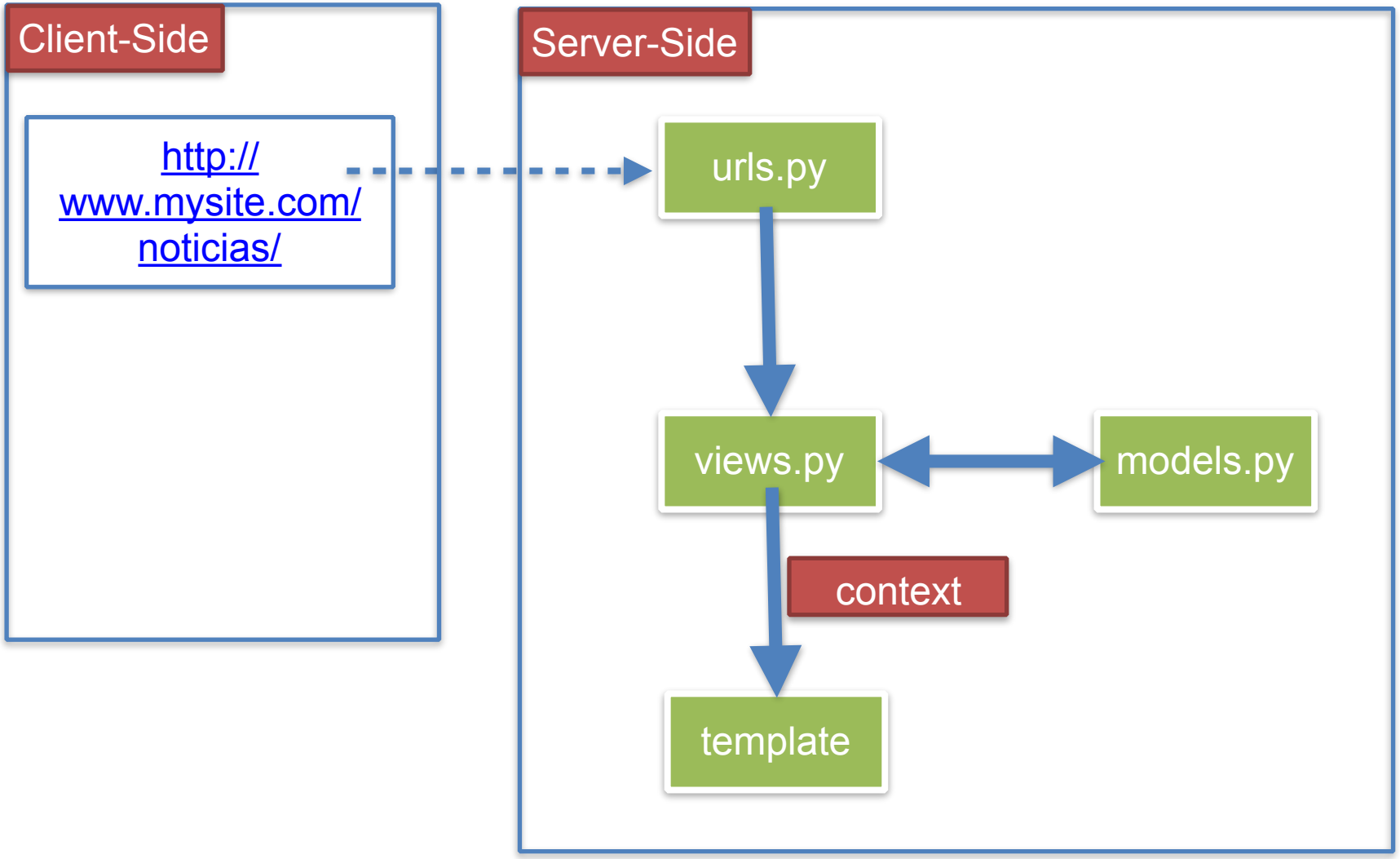
Repasando...


Client-Side
http://
www.mysite.com/
noticias/
Server-Side
urls.py
views.py

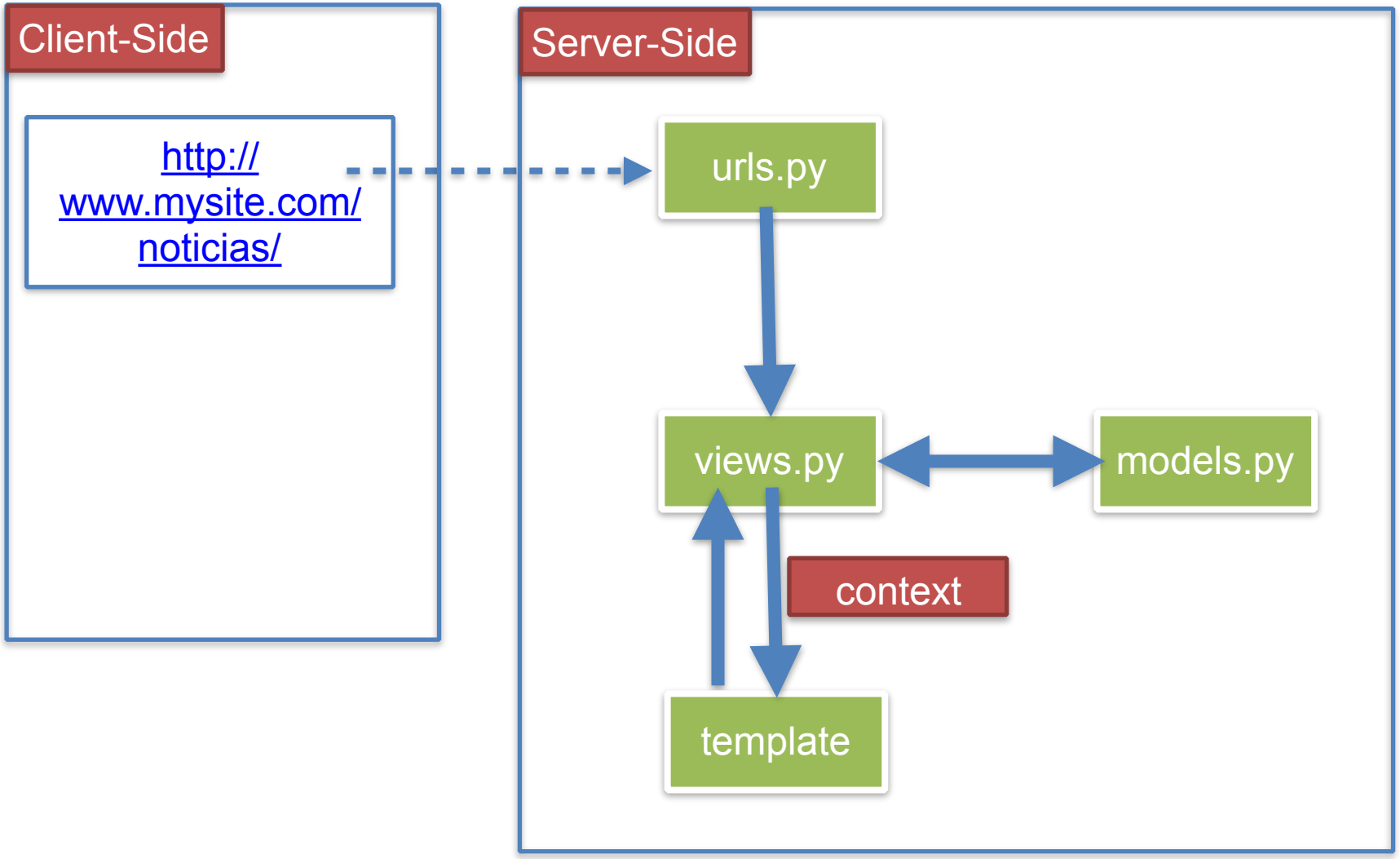
Repasando...


Client-Side
http://
www.mysite.com/
noticias/
Server-Side
urls.py
views.py
models.py

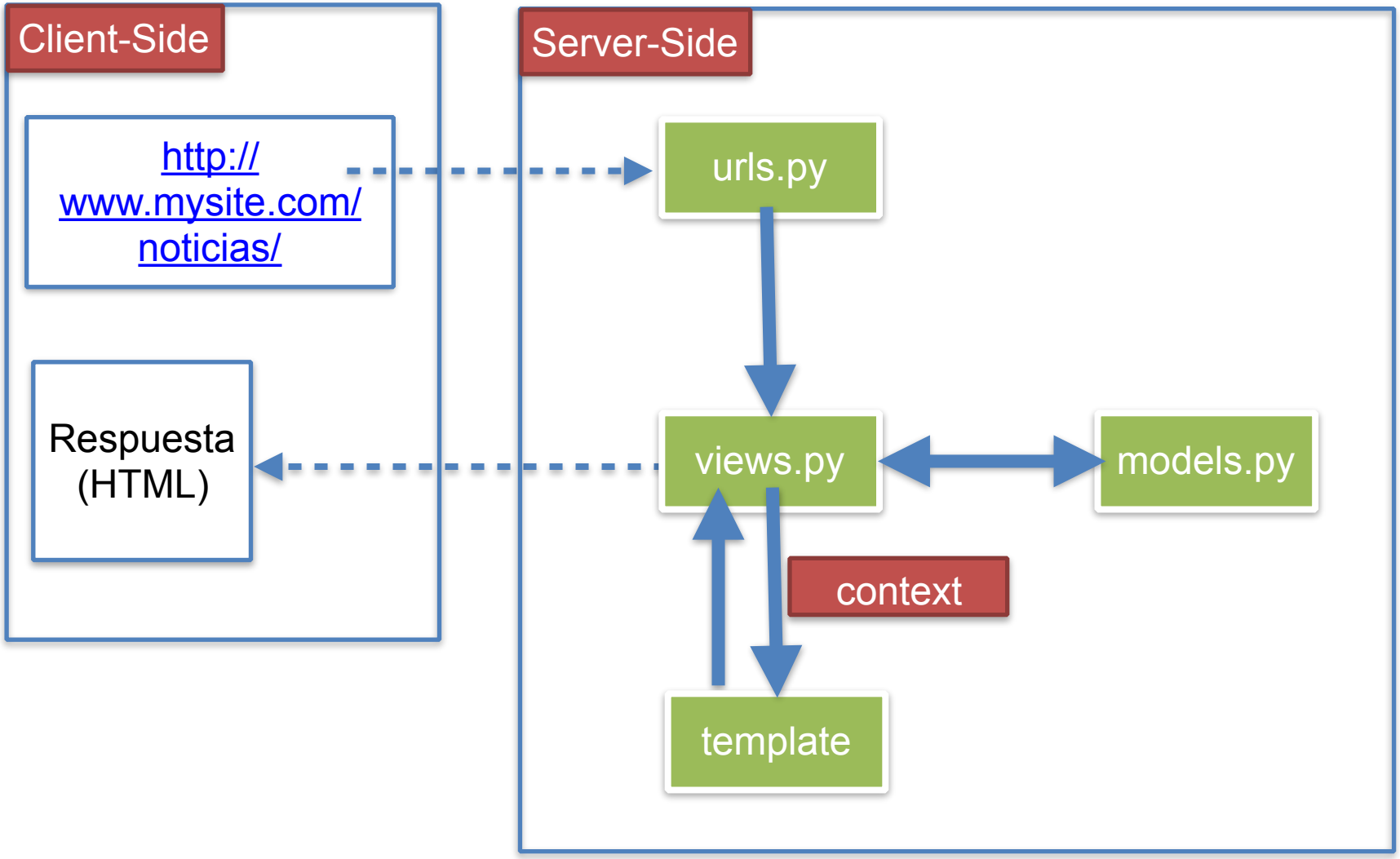
Repasando...


Client-Side
http://
www.mysite.com/
noticias/
Server-Side
urls.py
views.py
models.py
context
template

Repasando...


Client-Side
http://
www.mysite.com/
noticias/
Server-Side
urls.py
views.py
models.py
context
template

Repasando...


Client-Side
http://
www.mysite.com/
noticias/
Respuesta
(HTML)
Server-Side
urls.py
views.py
models.py
context
template

Ejemplo de Framework MVC: django, modelo persistente

- **Definición de clase Cliente**

```
class Cliente(models.Model):
    nombre = models.CharField("Nombre",max_length=256)
    apellido = models.CharField("Apellido",max_length=256)

    def set_nombre(self,unString):
        self.nombre = unString

    def __unicode__(self):
        return "%s %s" % (self.nombre, self.apellido)
```

Ejemplo de Framework MVC: django, modelo persistente

- Variables de clase referencian un “tipo de campo de base de datos”
- Variables de instancia referencian valores concretos para esos “tipos de datos”

```
class Cliente(models.Model):
    nombre = models.CharField("Nombre",max_length=256)
    apellido = models.CharField("Apellido",max_length=256)

    def set_nombre(self,unString):
        self.nombre = unString

    def __unicode__(self):
        return "%s %s" % (self.nombre, self.apellido)
```

- Punto de Extensión:
  - models.Model

```
__init__():
    …
    setattr(self, name, value)
    …
```

Ejemplo de Framework MVC: django, modelo persistente

```
class Operacion(models.Model):
    monto = models.IntegerField("Monto")
    cuenta = models.ForeignKey(CuentaBancaria)
    pendiente = models.BooleanField("Confirmada", default=True)

    def __init__(self, *args, **kwargs):
        super(Operacion, self).__init__(*args, **kwargs)
        self.operar()

    def __operar__(self):
        pass

    def operar(self):
        if self.pendiente:
            self.__operar__()
            self.confirmar()

    def confirmar(self):
        self.pendiente = False
        self.save()
```

Ejemplo de Framework MVC: django, modelo persistente

```
class Operacion(models.Model):
    monto = models.IntegerField("Monto")
    cuenta = models.ForeignKey(CuentaBancaria)
    pendiente = models.BooleanField("Confirmada", default=True)

    def __init__(self, *args, **kwargs):
        super(Operacion, self).__init__(*args, **kwargs)
        self.operar()

    def __operar__(self):
        pass

    def operar(self):
        if self.pendiente:
            self.__operar__()
            self.confirmar()

    def confirmar(self):
        self.pendiente = False
        self.save()
```

<Operacion object - 4>

cuenta=<CuentaBancaria object - 6>

monto=3

pendiente=False

Ejemplo de Framework MVC: django, modelo persistente

```
class Operacion(models.Model):
    monto = models.IntegerField("Monto")
    cuenta = models.ForeignKey(CuentaBancaria)
    pendiente = models.BooleanField("Confirmada", default=True)
```

<Operacion class object>

cuenta=<ForeignKey object>

monto=<IntegerField object>

pendiente=<BooleanField object>

```
gs):
__(*args, **kwargs)
```

```
self.pendiente = False
self.save()
```

<Operacion object - 4>

cuenta=<CuentaBancaria object - 6>

monto=3

pendiente=False

Ejemplo de Framework MVC: django, modelo persistente

```
class Operacion(models.Model):
    monto = models.IntegerField("Monto")
    cuenta = models.ForeignKey(CuentaBancaria)
    pendiente = models.BooleanField("Confirmada", default=True)
```

<Operacion class object>

cuenta=<ForeignKey object>

monto=<IntegerField object>

pendiente=<BooleanField object>

```
gs):
__(*args, **kwargs)
```

```
self.pendiente = False
self.save()
```

save()

<Operacion object - 4>

cuenta=<CuentaBancaria object - 6>

monto=3

pendiente=False

Ejemplo de Framework MVC: django, modelo persistente

```
class Operacion(models.Model):
    monto = models.IntegerField("Monto")
    cuenta = models.ForeignKey(CuentaBancaria)
    pendiente = models.BooleanField("Confirmada", default=True)
```

<Operacion class object>

cuenta=<ForeignKey object>

monto=<IntegerField object>

pendiente=<BooleanField object>

```
gs):
__(*args, **kwargs)
```

monto	cuenta	pendiente	id
			4

```
self.pendiente = False
self.save()
```

save()

<Operacion object - 4>

cuenta=<CuentaBancaria object - 6>

monto=3

pendiente=False

Ejemplo de Framework MVC: django, modelo persistente

```
class Operacion(models.Model):
    monto = models.IntegerField("Monto")
    cuenta = models.ForeignKey(CuentaBancaria)
    pendiente = models.BooleanField("Confirmada", default=True)
```

```
gs):
__(*args, **kwargs)
```

<Operacion class object>

cuenta=<ForeignKey object>

monto=<IntegerField object>

pendiente=<BooleanField object>

monto	cuenta	pendiente	id
			4

3

```
self.pendiente = False
self.save()
```

save()

<Operacion object - 4>

cuenta=<CuentaBancaria object - 6>

monto=3

pendiente=False

Ejemplo de Framework MVC: django, modelo persistente

```
class Operacion(models.Model):
    monto = models.IntegerField("Monto")
    cuenta = models.ForeignKey(CuentaBancaria)
    pendiente = models.BooleanField("Confirmada", default=True)
```

<Operacion class object>

cuenta=<ForeignKey object>

monto=<IntegerField object>

pendiente=<BooleanField object>

```
gs):
__(*args, **kwargs)
```

monto	cuenta	pendiente	id
3			4

3

```
self.pendiente = False
self.save()
```

save()

<Operacion object - 4>

cuenta=<CuentaBancaria object - 6>

monto=3

pendiente=False

Programación Web

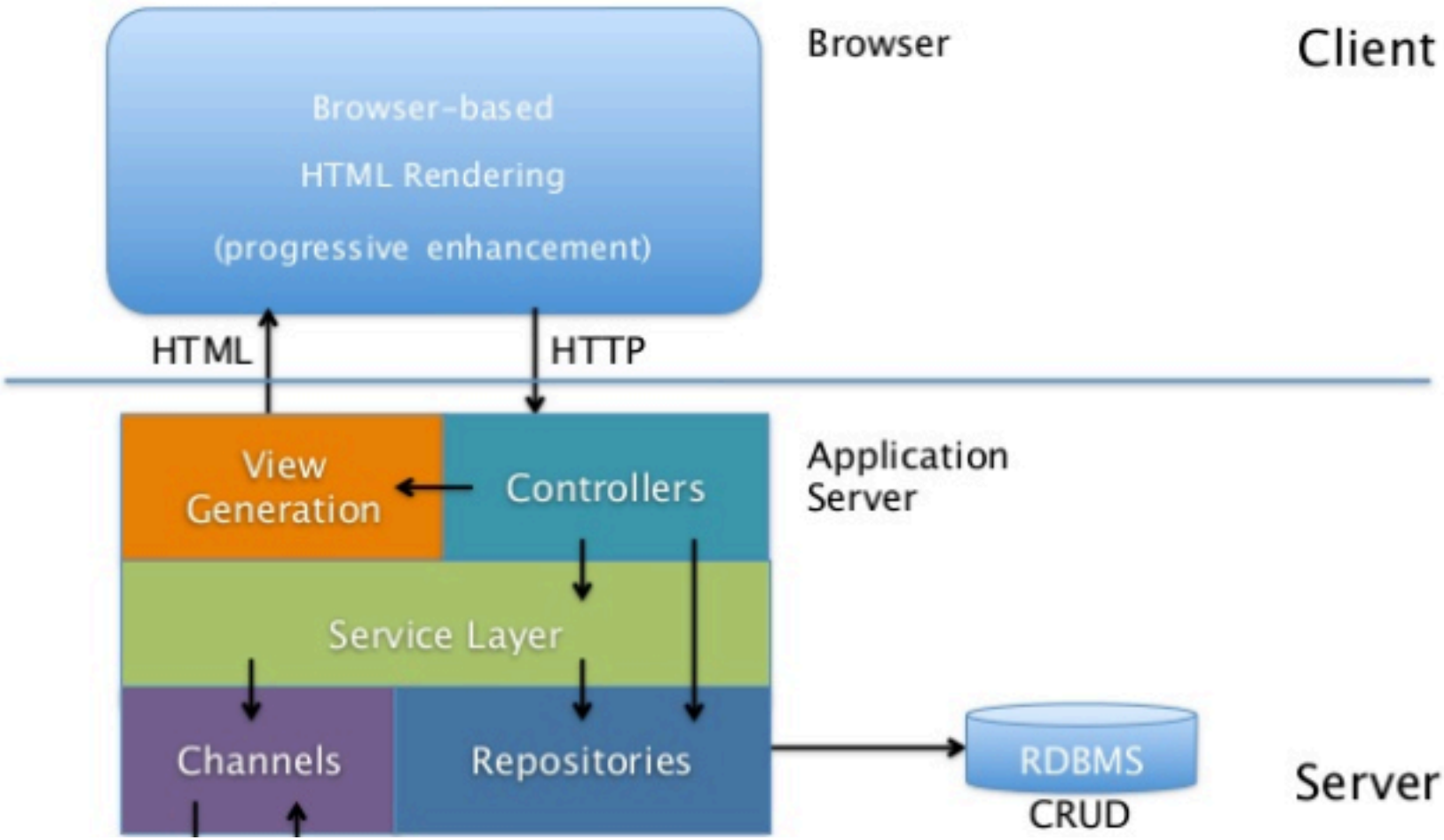
Tendencias en arquitecturas en los últimos años

Arquitecturas - hacia el cliente

- Proliferación de la Web
- Requerimientos de interacción en aplicaciones Web
  - RIA
- Evolución de la tecnología:
  - AJAX, HTML5, JavaScript
    - WebWorkers, WebStorage, etc
- y del uso de la tecnología
  - Responsive design, BigPipe
- y de la tecnología en general

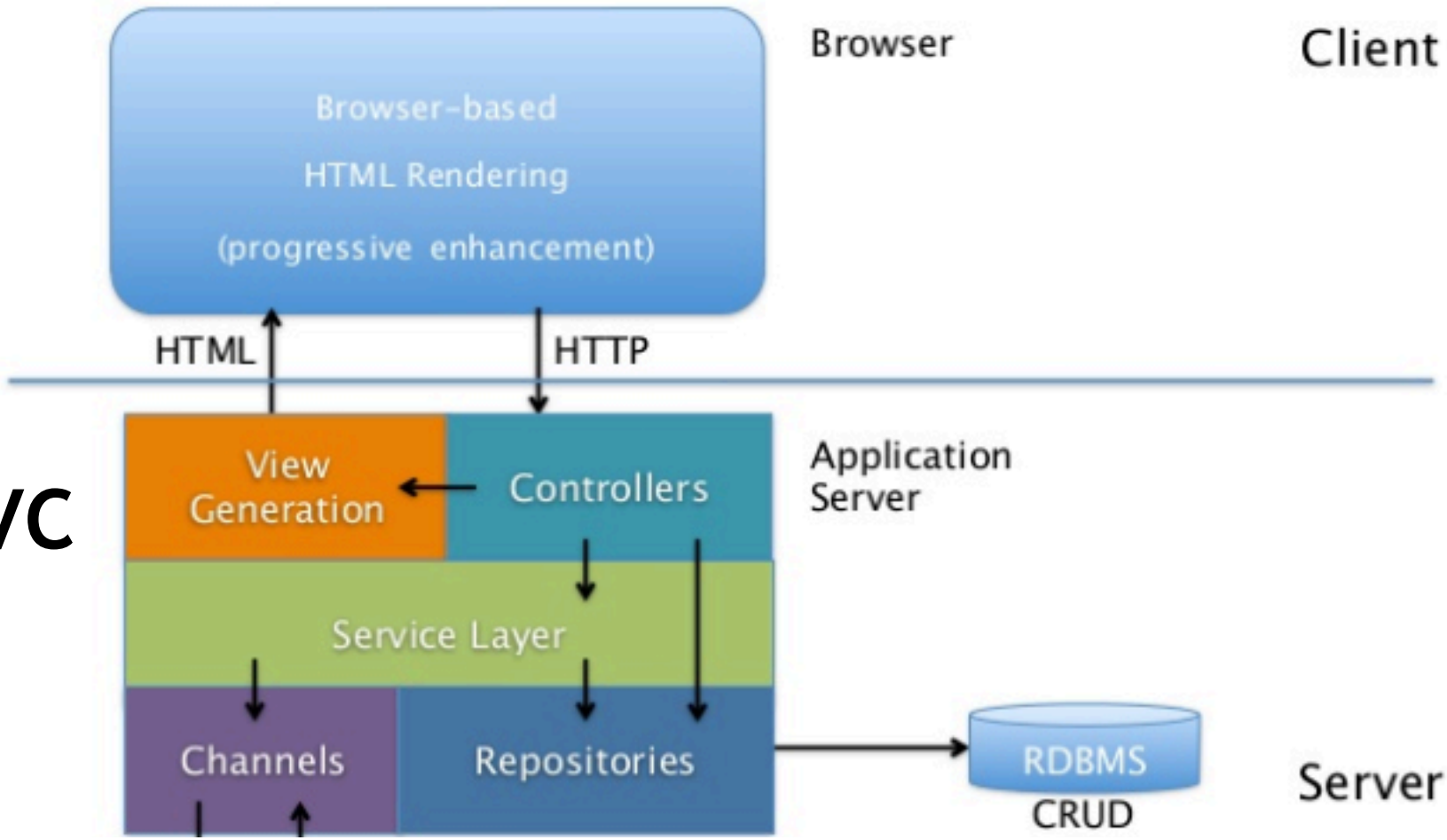
Arquitecturas - hacia el cliente

Arquitecturas - hacia el cliente



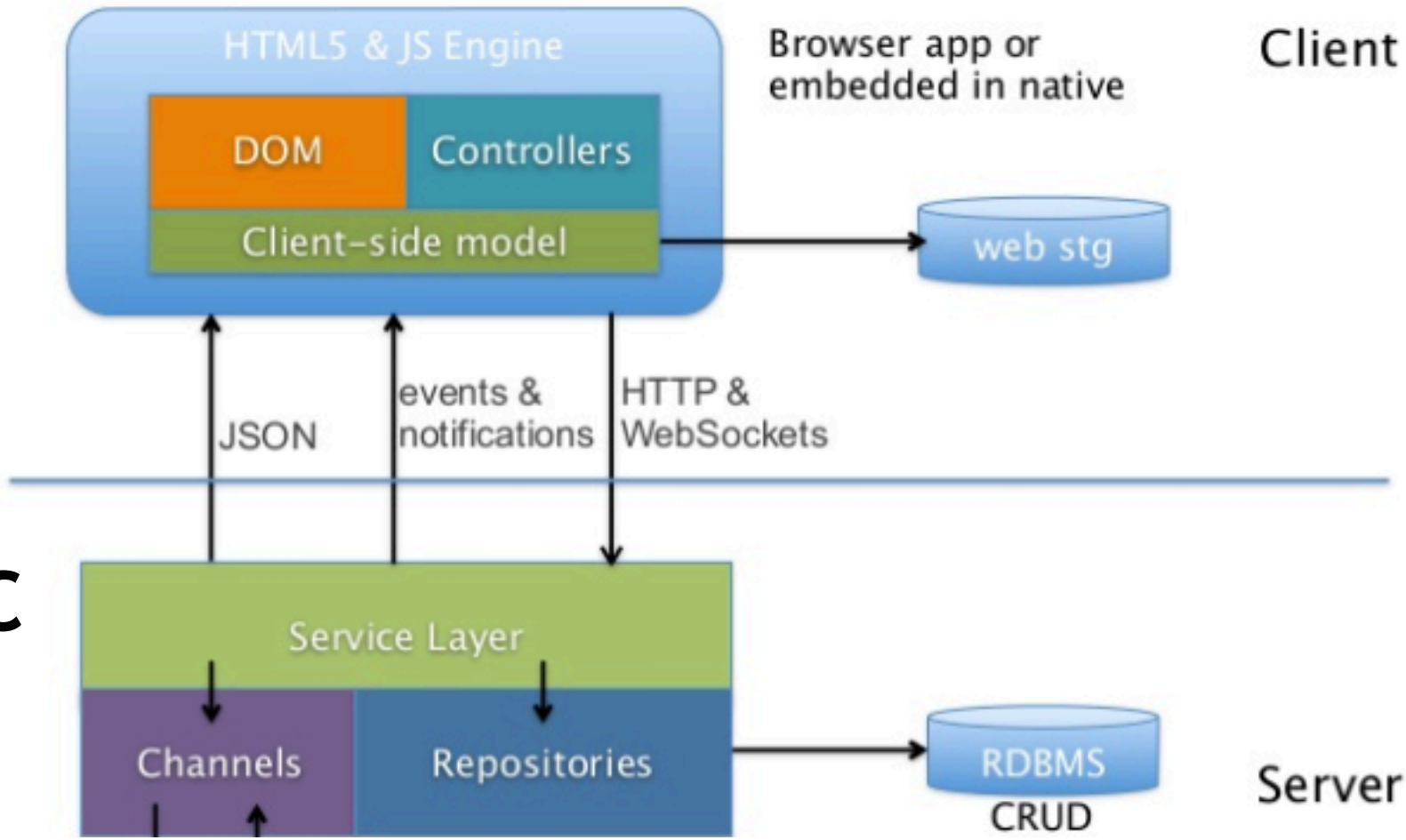
Arquitecturas - hacia el cliente

MVC



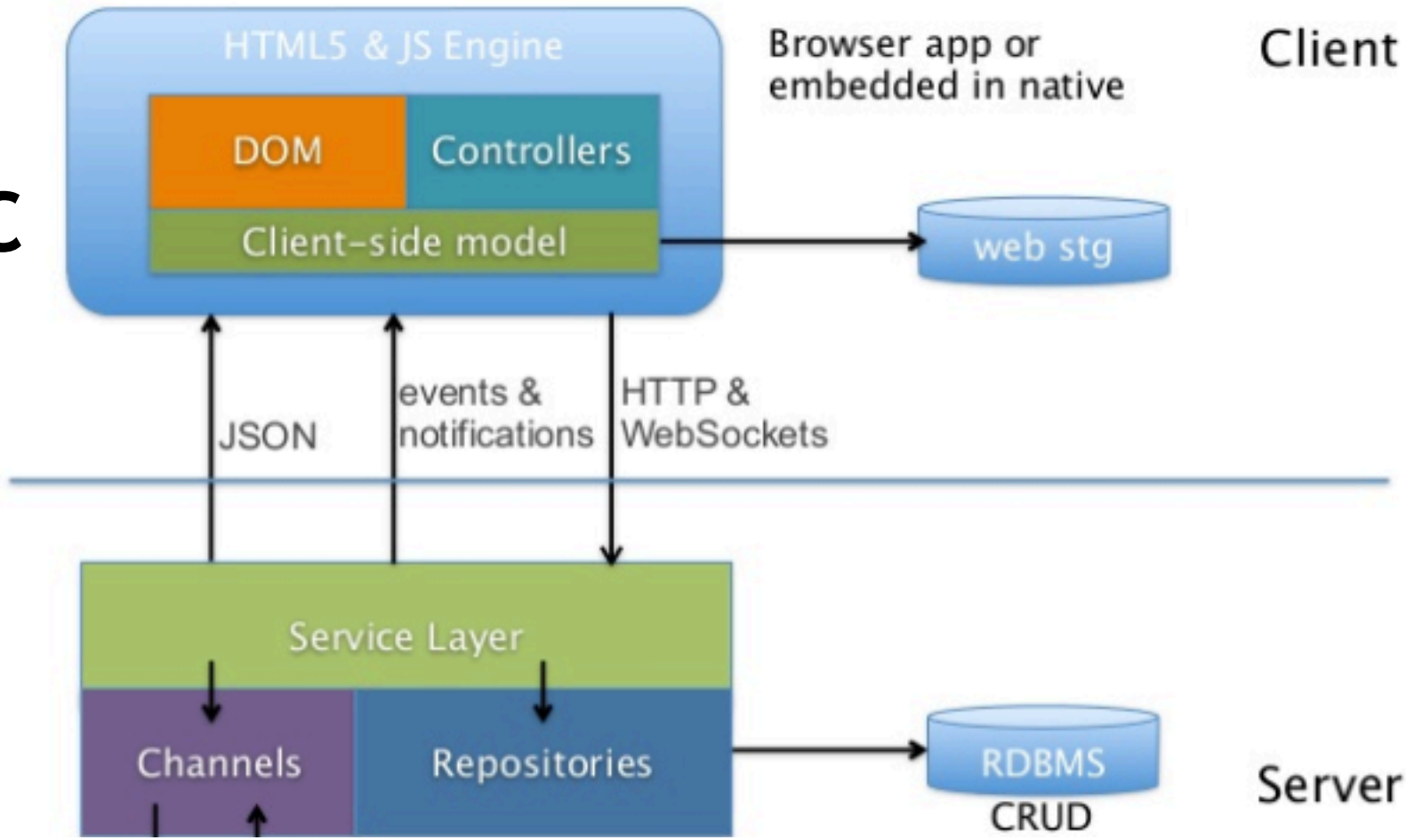
Arquitecturas - hacia el cliente

MVC

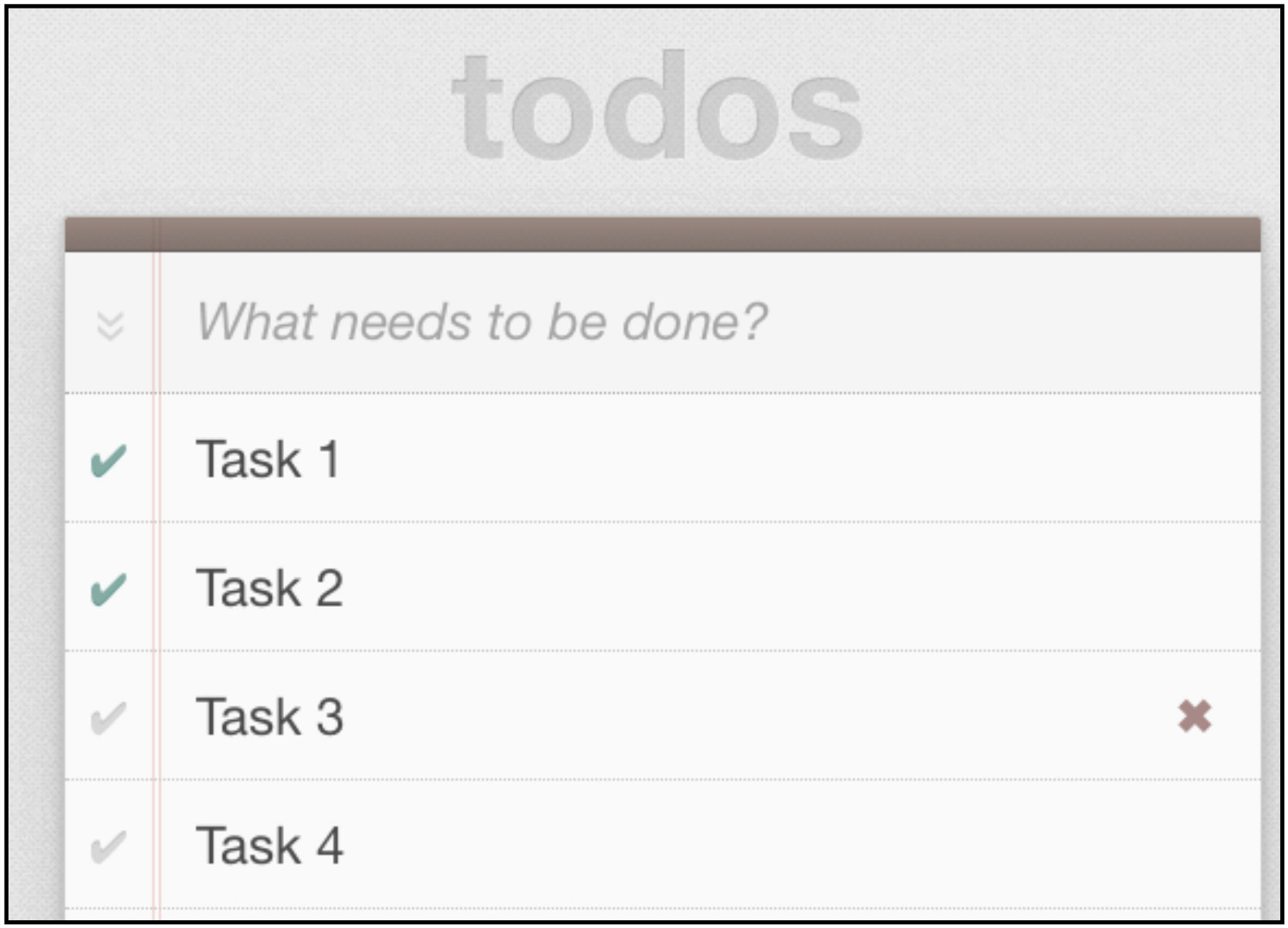


Arquitecturas - hacia el cliente

MVC



Ejemplo de ToDoApp SinglePage

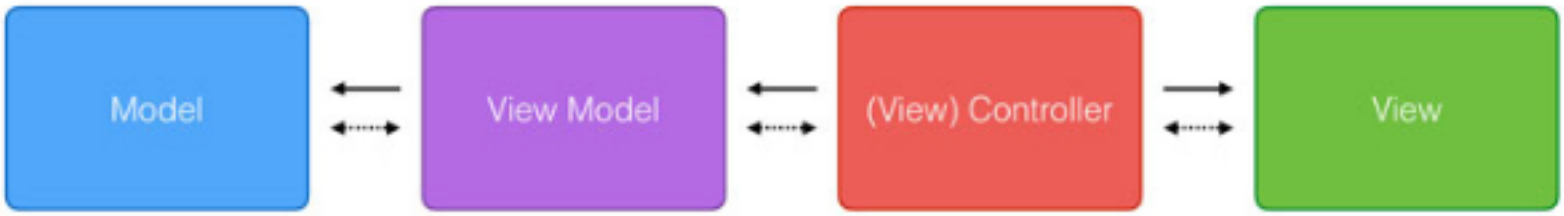


Ejemplo de Tecnología client-side: Angular.js

- **Motivación básica**
  - Mayor complejidad en el cliente
  - SinglePage applications
    - Mucha interacción
    - Implicancias técnicas
      - Manipulación del DOM
      - Manipulación del modelo subyacente
      - Manejo de información asincrónicamente

Ejemplo de Tecnología client-side: Angular.js

- MVC, ¿de nuevo?
  - Model-view viewmodel (MVVM)



Ejemplo de Tecnología client-side: Angular.js

```
<body>
  <h2>Todo</h2>
  <div ng-controller="TodoCtrl">
    <span>{{remaining()}} of {{todos.length}} remaining</span>
    [ <a href="" ng-click="archive()">archive</a> ]
    <ul class="unstyled">
      <li ng-repeat="todo in todos">
        <input type="checkbox" ng-model="todo.done">
        <span class="done-{{todo.done}}">{{todo.text}}</span>
      </li>
    </ul>
    <form ng-submit="addTodo()">
      <input type="text" ng-model="todoText"  size="30"
             placeholder="add new todo here">
      <input class="btn-primary" type="submit" value="add">
    </form>
  </div>
</body>
```

Ejemplo de Tecnología client-side: Angular.js

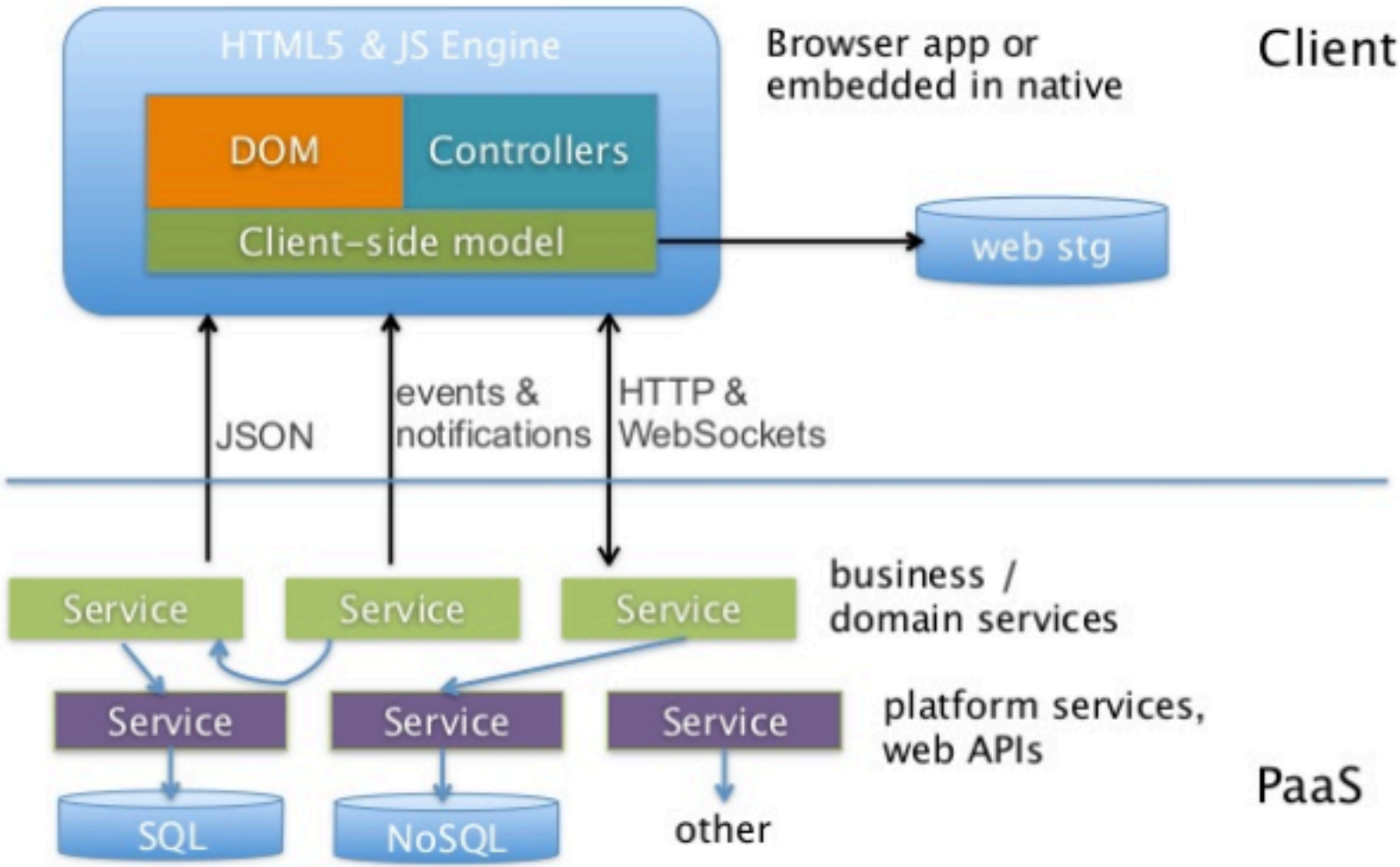
```
<body>
  <h2>Todo</h
  <div ng-con
    <span>{{r
    [ <a href
    <ul class
      <li ng-
        <inpu
        <span
      </li>
    </ul>
    <form ng-
      <input

      <input
    </form>
  </div>
</body>
```

```
<script>
  function TodoCtrl($scope) {
    $scope.todos = [
      {text:'learn angular', done:true},
      {text:'build an angular app', done:false}];

    $scope.addTodo = function() {
      $scope.todos.push({text:$scope.todoText, done:false});
      $scope.todoText = '';
    };

    $scope.remaining = function() {
      var count = 0;
      angular.forEach($scope.todos, function(todo) {
        count += todo.done ? 0 : 1;
      });
      return count;
    };

    $scope.archive = function() {
      var oldTodos = $scope.todos;
      $scope.todos = [];
      angular.forEach(oldTodos, function(todo) {
        if (!todo.done) $scope.todos.push(todo);
      });
    };
  }
</script>
```

Arquitecturas - aun mas

- **Thin clients**
  - El usuario utiliza un cliente estándar sin necesidad de ningún software adicional.
  - Se requiere una capacidad computacional mínima.
  - HTML4
- **Thick clients**
  - Reducen la carga de comunicación por red.
    - Reducen la carga de trabajo en el servidor
  - Se aprovecha de la capacidad de procesamiento osiosa que tienen los clientes.
  - Por ejemplo, Gmail modo offline
- **… Angular2, React**

Arquitecturas - hacia el cliente



Programación Web

Conclusiones

Conclusiones

- La evolución de la Web
  - Web 1.0, Web 2.0
  - RIA

- En la solución de estas complejidades
  - Han entrado en juego soluciones anteriores
    - Incorporación OOP en la Web
    - MVC en su 1era aplicación en la Web
    - MVC en su 2da aplicación en la Web
    - Uso del observer en aplicaciones client-side

- **Importancia de los patrones**