“The ideal architect should be a man of letters, a skillful draftsman, a mathematician, familiar with historical studies, a diligent student of philosophy, acquainted with music; not ignorant of medicine, learned in the responses of jurisconsults, familiar with astronomy, and astronomical calculations”

Vitruvius 25 B.C.

Web Sciences

Presente y futuro de la Web

Dr. Juan Ignacio Sanz Fuentes
enero 2016
jsanz.eps@ceu.es

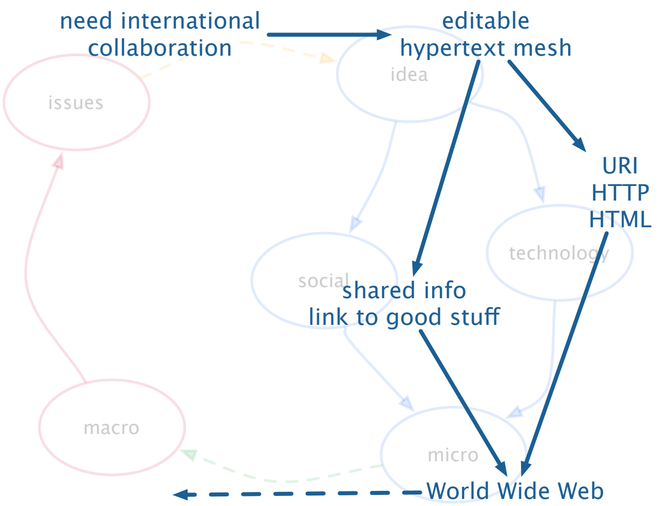
Tecnologías para la Programación
y el Diseño Web i
2º GCD

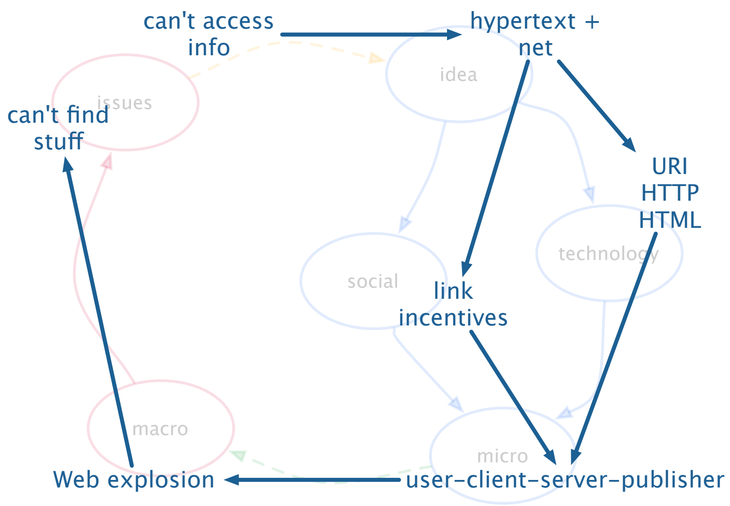
contenido

- Breve introducción y evolución de la Web
- Arquitectura de las aplicaciones web
- Desarrollo de aplicaciones web
- Web Sciences

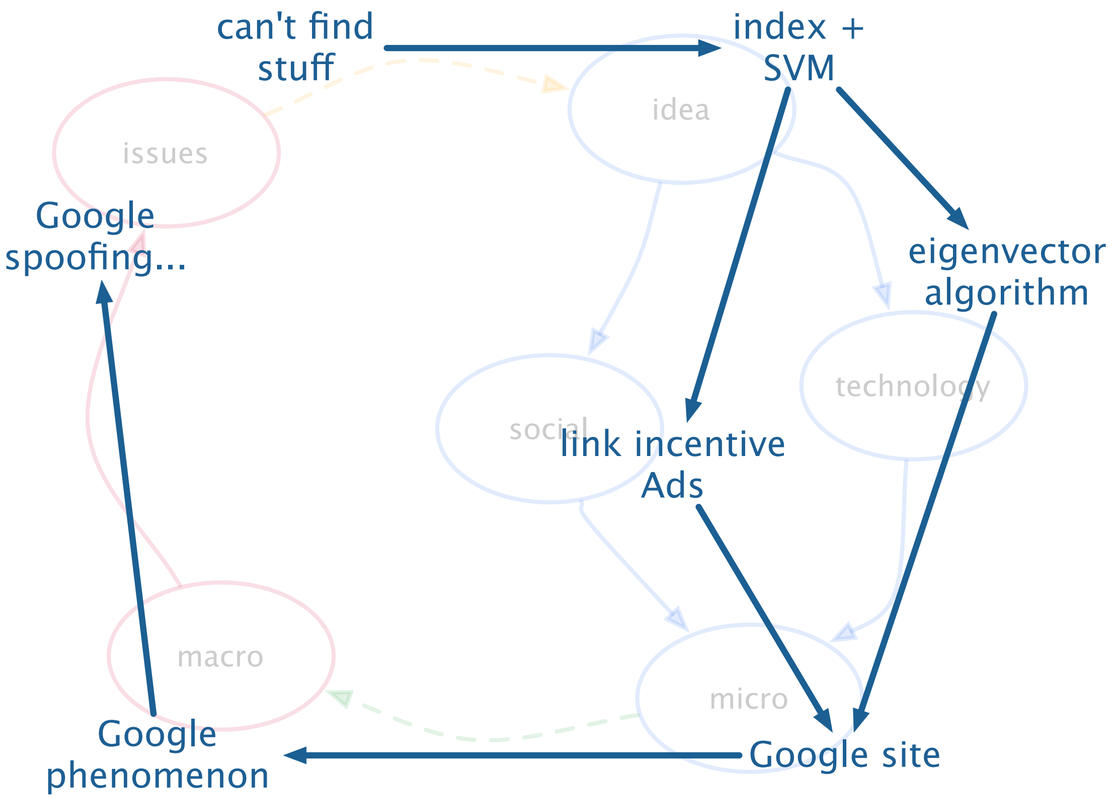
Breve introducción y evolución de la Web

Origen de la Web

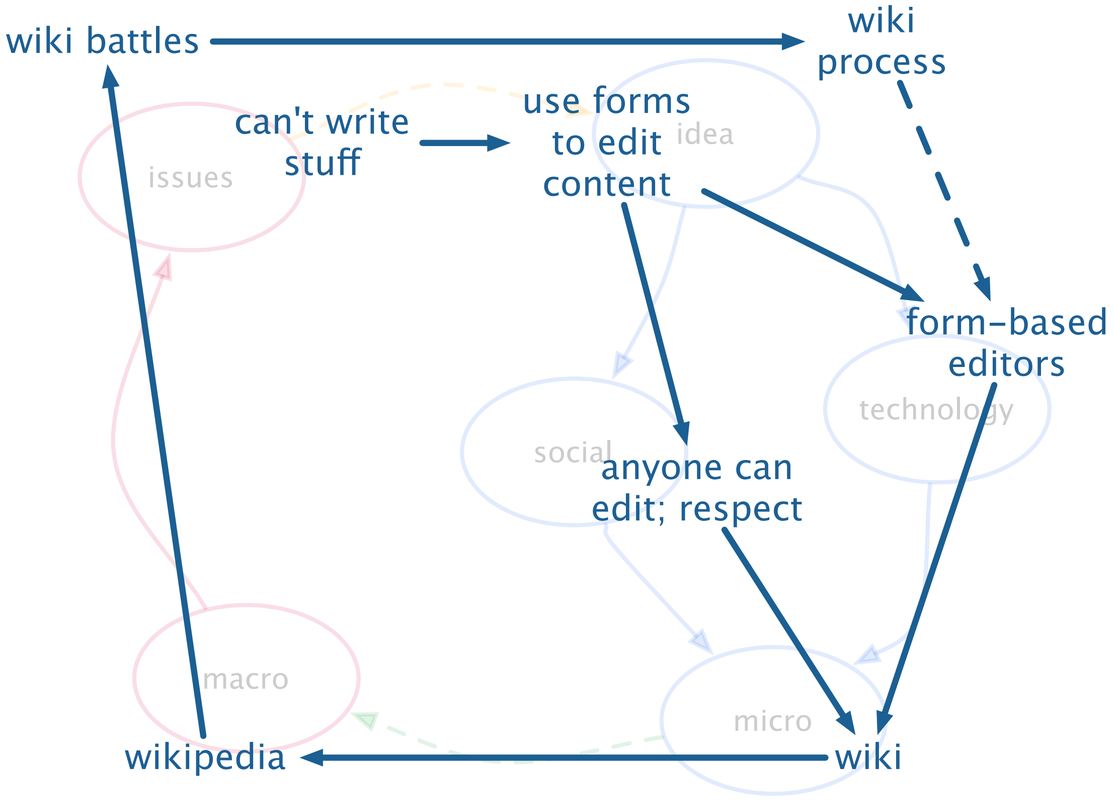




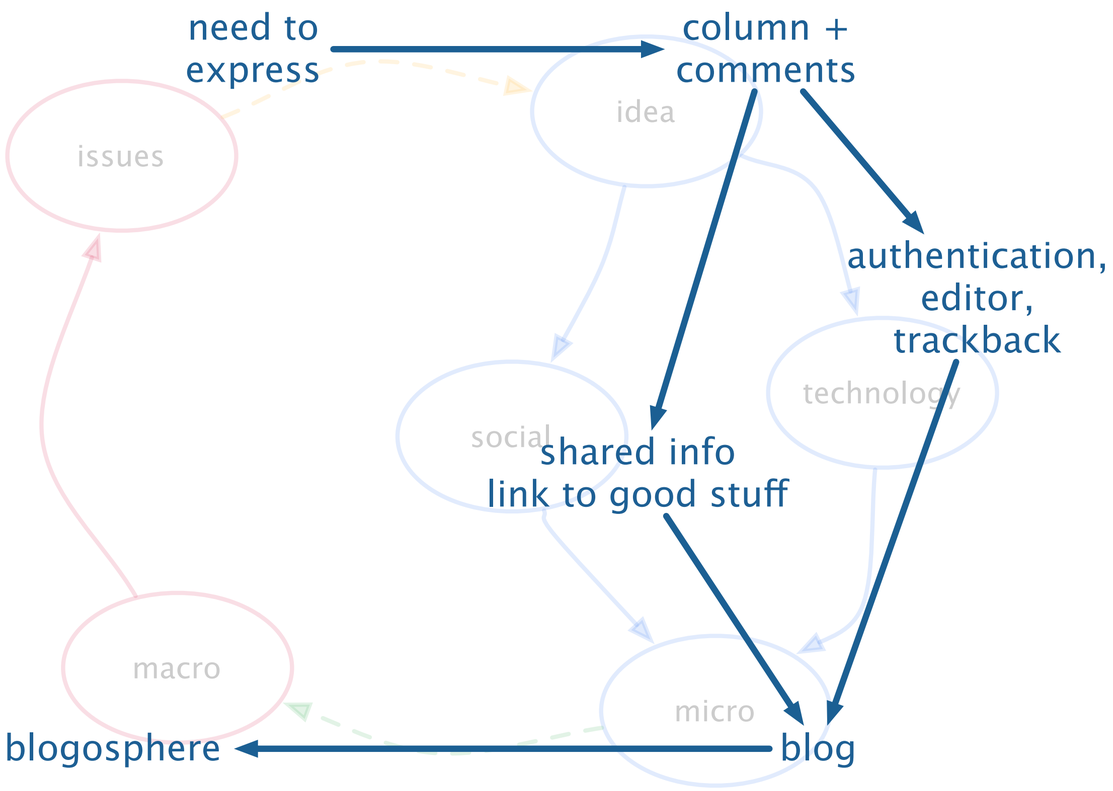
Google™


can't find stuff
index + SVM
idea
issues
Google spoofing...
eigenvector algorithm
technology
social
link incentive Ads
macro
micro
Google phenomenon
Google site

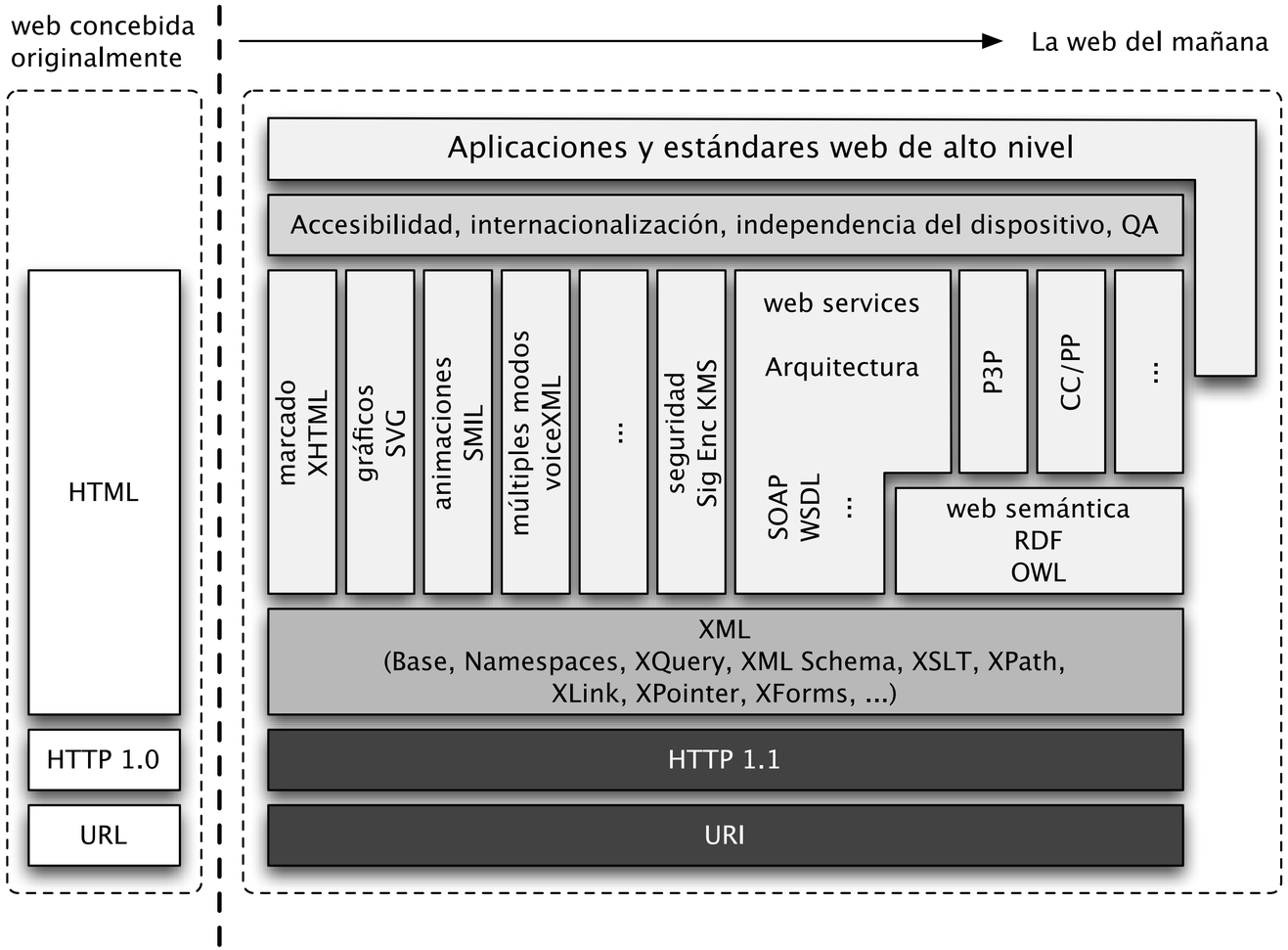
wiki


wiki battles
wiki process
can't write stuff
use forms to edit content
idea
issues
form-based editors
technology
social
anyone can edit; respect
macro
micro
wikipedia
wiki

blog


need to
express
column +
comments
idea
issues
authentication,
editor,
trackback
technology
social
shared info
link to good stuff
macro
micro
blogosphere
blog

Evolución de la Web



¿por qué HTML NO es suficiente?

<P> paragraph

 break

<B> bold

<H1> heading level 1

```
<P>09 19 2005
      10:30 AM
      MAD/LON
<BR>
      IB745
<P>18C /home/photos/madrid.jpg</P>
<P>12C /home/photos/london.jpg</P>
```

- Tags shows how data should be rendered, not its meaning.
- Automatic processing unavailable: an interpretation of the sequence of elements and content found is needed

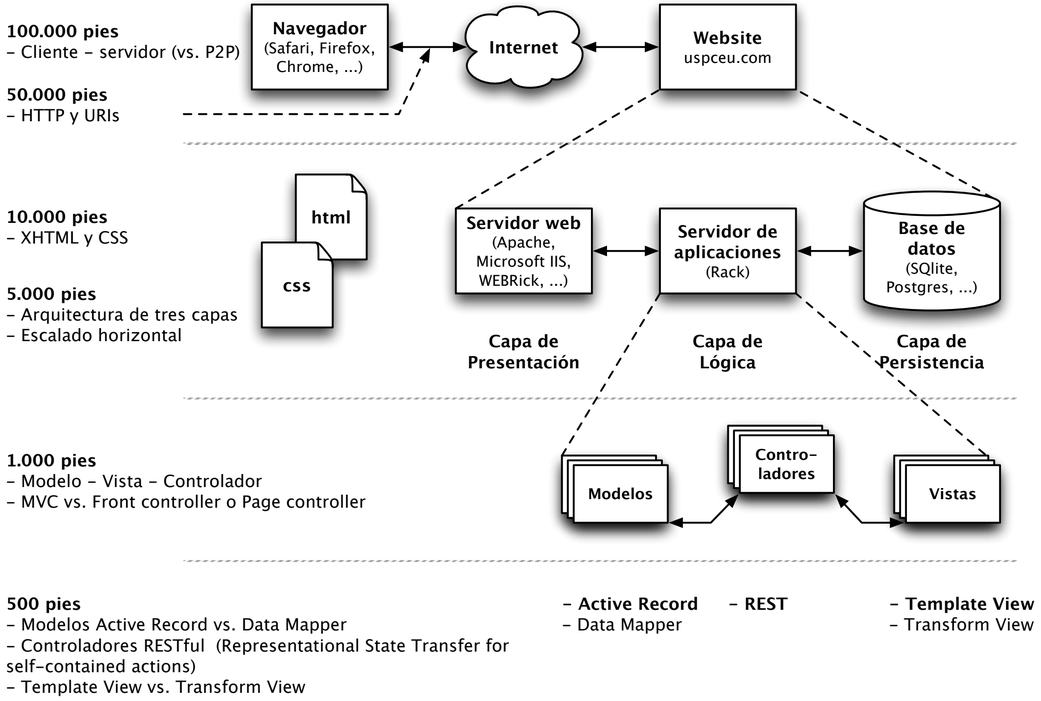
Es necesario definir estructura y etiquetas con significado

```
<?xml version="1.0" encoding="iso-8859-1"?>
<ticket>
    <date>
        <day>19</day>
        <month>09</month>
        <year>2005</year>
    </date>
    <flight>
        <airline>IB</airline>
        <number>745</number>
        <time>10:30 AM</time>
        <departure id=”MAD” temp=”18C” img=”/home/photos/madrid.jpg”/>
        <arrival id=”LON” temp=”12C” img=”/home/photos/london.jpg”/>
    </flight>
</ticket>
```

Arquitectura de las aplicaciones web

arquitectura Web

- Capas: separar lo que cambia de lo que permanece
- Varios niveles: arquitectura



arquitectura Web

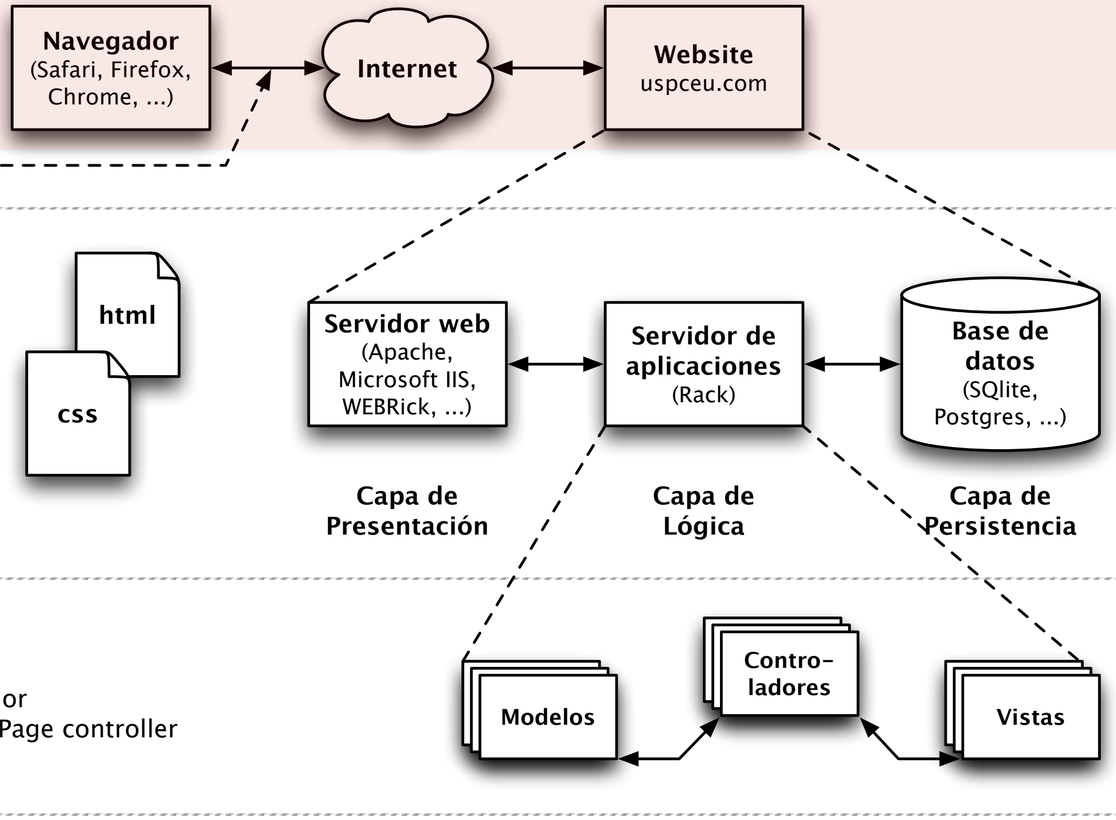
100.000 pies
- Cliente - servidor (vs. P2P)

50.000 pies
- HTTP y URIs

10.000 pies
- XHTML y CSS

5.000 pies
- Arquitectura de tres capas
- Escalado horizontal

1.000 pies
- Modelo - Vista - Controlador
- MVC vs. Front controller o Page controller



500 pies
- Modelos Active Record vs. Data Mapper
- Controladores RESTful (Representational State Transfer for self-contained actions)
- Template View vs. Transform View

- **Active Record**
- Data Mapper

- **REST**

- **Template View**
- Transform View

arquitectura Web a 100.000 pies

- Cliente:
  - Presentación, interfaz
  - complejo actualmente
- Servidor:
  - eficiencia
- Estándares web: cliente universal
- Otras arquitecturas: P2P como bitTorrent
- Arquitectura de producción/explotación vs desarrollo

arquitectura Web a 100.000 pies

1. El navegador (Safari) realiza una solicitud al servidor web (Bottle)
2. Bottle obtiene el contenido de la aplicación y lo envía de vuelta a Safari
3. Safari muestra el contenido en pantalla y cierra la conexión HTTP

arquitectura Web

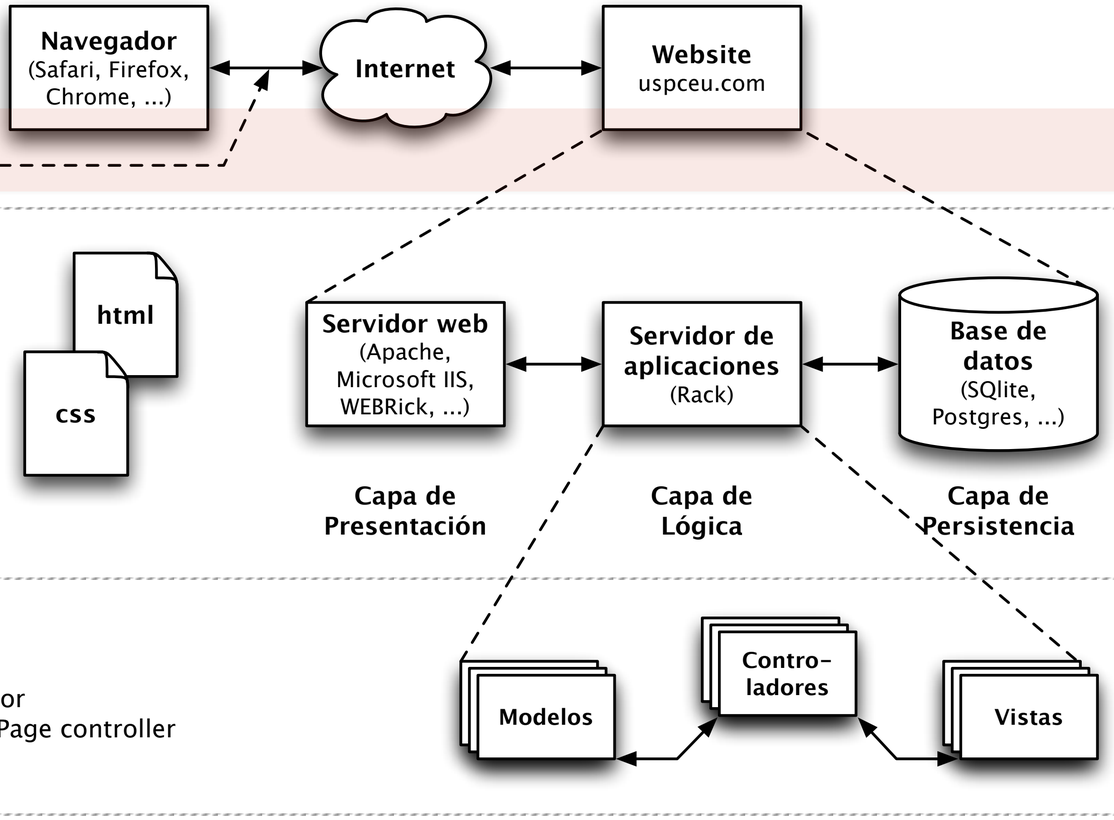
100.000 pies
- Cliente - servidor (vs. P2P)

50.000 pies
- HTTP y URIs

10.000 pies
- XHTML y CSS

5.000 pies
- Arquitectura de tres capas
- Escalado horizontal

1.000 pies
- Modelo - Vista - Controlador
- MVC vs. Front controller o Page controller



500 pies
- Modelos Active Record vs. Data Mapper
- Controladores RESTful (Representational State Transfer for self-contained actions)
- Template View vs. Transform View

- **Active Record**
- Data Mapper

- **REST**

- **Template View**
- Transform View

arquitectura Web a 50.000 pies

- Protocolo de red:
  - Reglas participantes en un acuerdo
  - Comunicación Navegador servidor web y vuelta
  - HTTP basado en TCP/IP
- IP:
  - número identificación de cada máquina
  - DNS traduce nombres a números (navegador)
  - localhost es la propia máquina
- Puerto:
  - mismos agentes distintas conexiones
- HTTP: ip/hostname + puerto

arquitectura Web a 50.000 pies

- URI:
  - `http://localhost:8080/todo`
  - Recurso: imagen, texto, video, formulario, hoja de estilo, js, …
  - SaaS: reglas recursos y REST
- HTTP: métodos Get, Post, Put, Delete, …
- Stateless:
  - peticiones independientes, responsabilidad de la app saber si el usuario está logueado o no
- Cookies:
  - responsabilidad del browser no SaaS ni app ni HTTP
  - asociar navegador con sesión en servidor
  - rails lo enmascara completamente
  - http header set_cookie primera vez

arquitectura Web a 50.000 pies

1. El navegador (Safari) realiza una solicitud al servidor web (Bottle)
    1.a. Safari construye una petición HTTP utilizando la URI `http://localhost:8080/todo` para contactar con el servidor web (Bottle) que escucha en el puerto 8080 de la misma máquina que el navegador (**localhost**)
    1.b. Bottle que escucha en el puerto 8080 recibe una petición HTTP para el recurso `/todo` (la lista de tareas en la aplicación todo)
2. Bottle obtiene el contenido de la aplicación y lo envía de vuelta a Safari
3. Safari muestra el contenido en pantalla y cierra la conexión HTTP

arquitectura Web

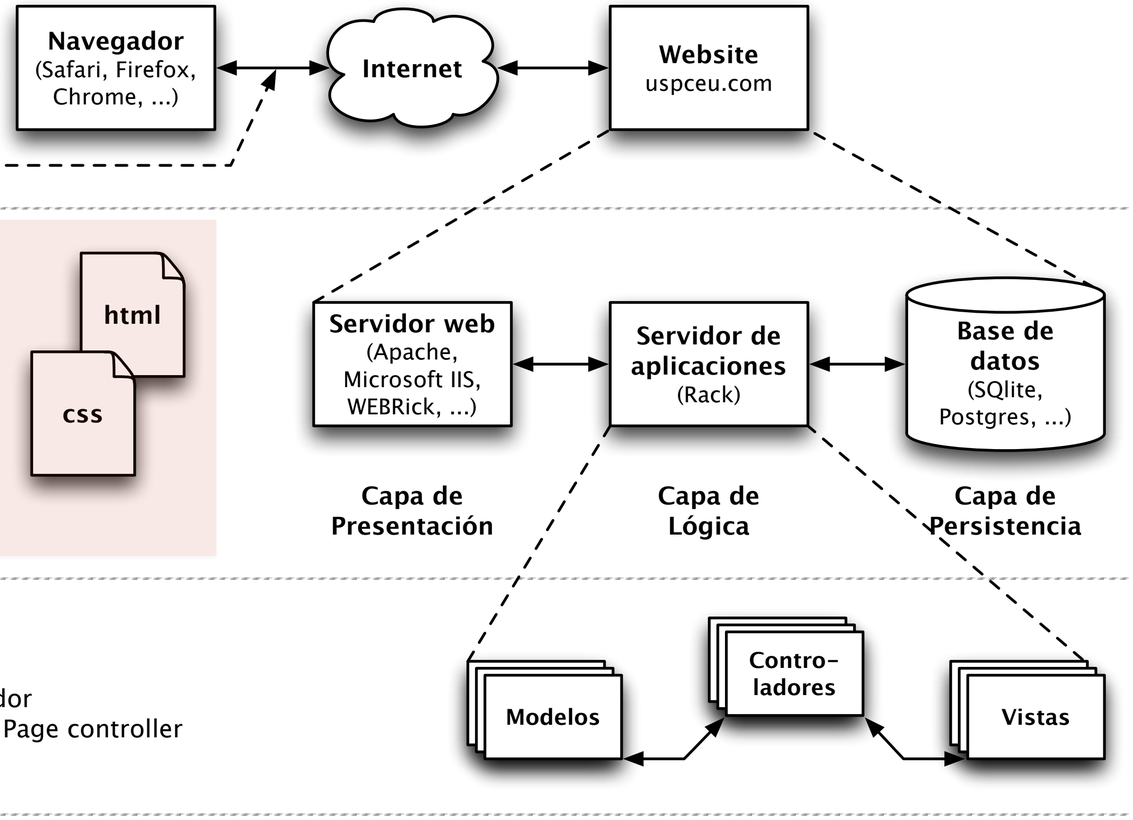
100.000 pies
- Cliente - servidor (vs. P2P)

50.000 pies
- HTTP y URIs

10.000 pies
- XHTML y CSS

5.000 pies
- Arquitectura de tres capas
- Escalado horizontal

1.000 pies
- Modelo - Vista - Controlador
- MVC vs. Front controller o Page controller



500 pies
- Modelos Active Record vs. Data Mapper
- Controladores RESTful (Representational State Transfer for self-contained actions)
- Template View vs. Transform View

- **Active Record**
- Data Mapper

- **REST**

- **Template View**
- Transform View

arquitectura Web a 10.000 pies

- HTML:
  - lenguaje de marcado
  - jerarquía, etiquetas anidadas
  - start tag contenido end tag
  - atributos
    - id, class: inspector web
- CSS:
  - selectores y propiedades
  - CSS nivel 1, 2 y 3

arquitectura Web a 10.000 pies

1. El navegador (Safari) realiza una solicitud al servidor web (Bottle)
    1.a. Safari construye una petición HTTP utilizando la URI `http://localhost:8080/todo` para contactar con el servidor web (Bottle) que escucha en el puerto 8080 de la misma máquina que el navegador (**localhost**)
    1.b. Bottle que escucha en el puerto 8080 recibe una petición HTTP para el recurso `/todo` (la lista de tareas en la aplicación todo)
2. Bottle obtiene el contenido de la aplicación y lo envía de vuelta a Safari
    2.a. Bottle devuelve el contenido codificado en HTML de nuevo utilizando HTTP. El código HTML puede contener referencias a otros tipos de contenido como imágenes embebidas en la página a visualizar. El código HTML puede además contener referencias a hojas de estilo CSS con información descrita sobre atributos visuales de la página (fuentes, colores, disposición, etc)
3. Safari muestra el contenido en pantalla y cierra la conexión HTTP
    3.a. Safari recupera cualquier contenido externo (CSS, imágenes, etc) repitiendo los cuatro pasos anteriores según necesite únicamente proporcionando las URIs de los componentes deseados tal como se encuentran referenciados en la página HTML.
    3.b. Safari muestra la página de acuerdo a las reglas de formato especificadas en la hoja de estilos CSS e incluyendo cualquier componente externo como imágenes embebidas y otros.

arquitectura Web

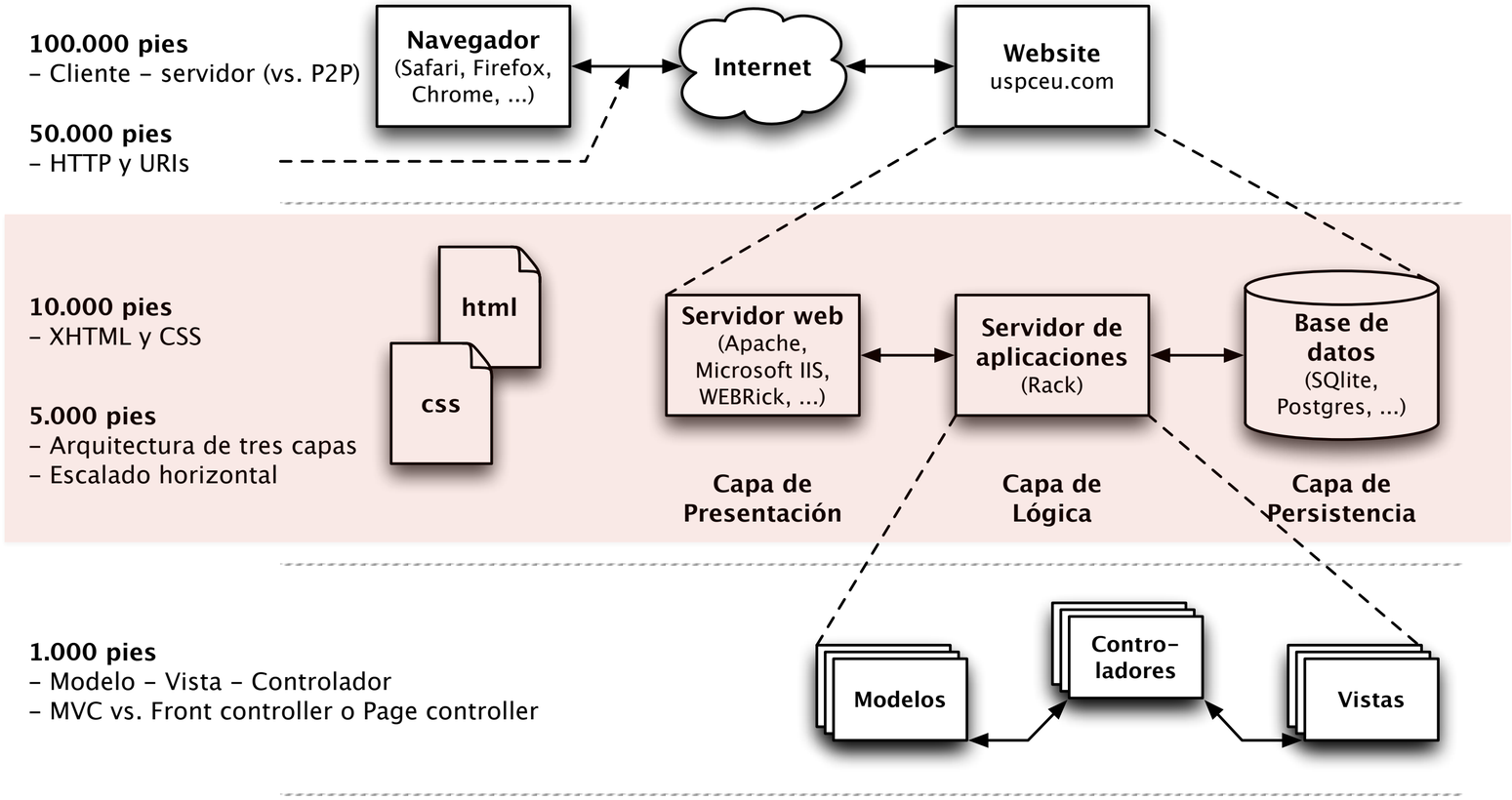
100.000 pies
- Cliente - servidor (vs. P2P)

50.000 pies
- HTTP y URIs

10.000 pies
- XHTML y CSS

5.000 pies
- Arquitectura de tres capas
- Escalado horizontal

1.000 pies
- Modelo - Vista - Controlador
- MVC vs. Front controller o Page controller



500 pies
- Modelos Active Record vs. Data Mapper
- Controladores RESTful (Representational State Transfer for self-contained actions)
- Template View vs. Transform View

- **Active Record**
- Data Mapper

- **REST**

- **Template View**
- Transform View

arquitectura Web a 5.000 pies

- Capa de presentación:
  - servidor http o servidor web
  - webrick en rails dev, thin o unicorn en producción
- Capa de lógica/negocio:
  - middleware conecta servidor web con persistencia
  - servidor de aplicaciones
  - ejecuta código: java python en google app server, Ruby en Rails, python en Django
  - rack en rails
- Capa de persistencia:
  - db2, postgres, oracle, mysql, sqlite3 según desarrollo o producción

arquitectura Web a 5.000 pies

1. El navegador (Safari) realiza una solicitud al servidor web (Bottle)
    1.a. Safari construye una petición HTTP utilizando la URI `http://localhost:8080/todo` para contactar con el servidor web (Bottle) que escucha en el puerto 8080 de la misma máquina que el navegador (**localhost**)
    1.b. Bottle que escucha en el puerto 8080 recibe una petición HTTP para el recurso `/todo` (la lista de tareas en la aplicación todo)
2. Bottle obtiene el contenido de la aplicación y lo envía de vuelta a Safari
    2.a. A través del middleware (escrito en Python), Bottle llama al código de `todo` en la capa de aplicación. Este código genera el contenido de la página utilizando la información de las tareas almacenadas en la capa de persistencia implementada por una base de datos SQLite instalada localmente.
    2.b. Bottle devuelve el contenido codificado en HTML de nuevo utilizando HTTP. El código HTML puede contener referencias a otros tipos de contenido como imágenes embebidas en la página a visualizar. El código HTML puede además contener referencias a hojas de estilo CSS con información descrita sobre atributos visuales de la página (fuentes, colores, disposición, etc)
3. Safari muestra el contenido en pantalla y cierra la conexión HTTP
    3.a. Safari recupera cualquier contenido externo (CSS, imágenes, etc) repitiendo los cuatro pasos anteriores según necesite únicamente proporcionando las URIs de los componentes deseados tal como se encuentran referenciados en la página HTML.
    3.b. Safari muestra la página de acuerdo a las reglas de formato especificadas en la hoja de estilos CSS e incluyendo cualquier componente externo como imágenes embebidas y otros.

desarrollo de aplicaciones web

construcción de aplicaciones web

1. **Definir rutas**
   - El servidor web escucha peticiones y determina qué rutas son conocidas y se procesarán y cuáles no.
2. **Configurar la acción para cada ruta**
   - Consisten básicamente en decir al ordenador qué debe hacer para cada ruta/petición (Java servlets, python, ruby, ...)

   2.1. Habitualmente implica acceso a una base de datos

   2.2. Generar una respuesta HTML utilizando plantillas (templates)

web site vs. aplicación web

- **Web site**: colección de páginas estáticas.
  - La información es almacenada principalmente como páginas estáticas para su descarga.
  - Los formularios simplifican el acceso a las bases de datos.
- **Aplicación Web**: aplicación desarrollada en un entorno web, accesible por medio de un navegador que permite:
  - Arrancar procesos
  - Filtrar información
  - Conexiones a bases de datos y sistemas de información
  - Ejecutar código (Java servlets, python, ruby, ...)

problemas inherentes

- **Protocolo de comunicación HTTP**
  - Interactividad limitada: request/response.

- **Información presentada en HTML**
  - Limitado
  - Diferentes interpretaciones por diferentes navegadores/dispositivos

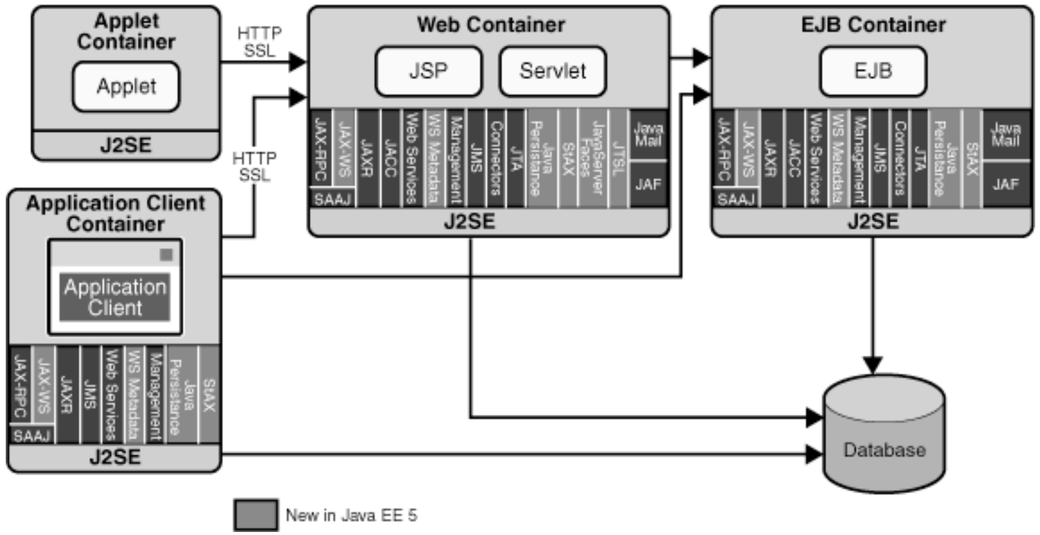
- **Acceso de los usuarios y concurrencia**
  - Independencia del cliente respecto al servidor
  - Fallos de comunicación

evolución (i)

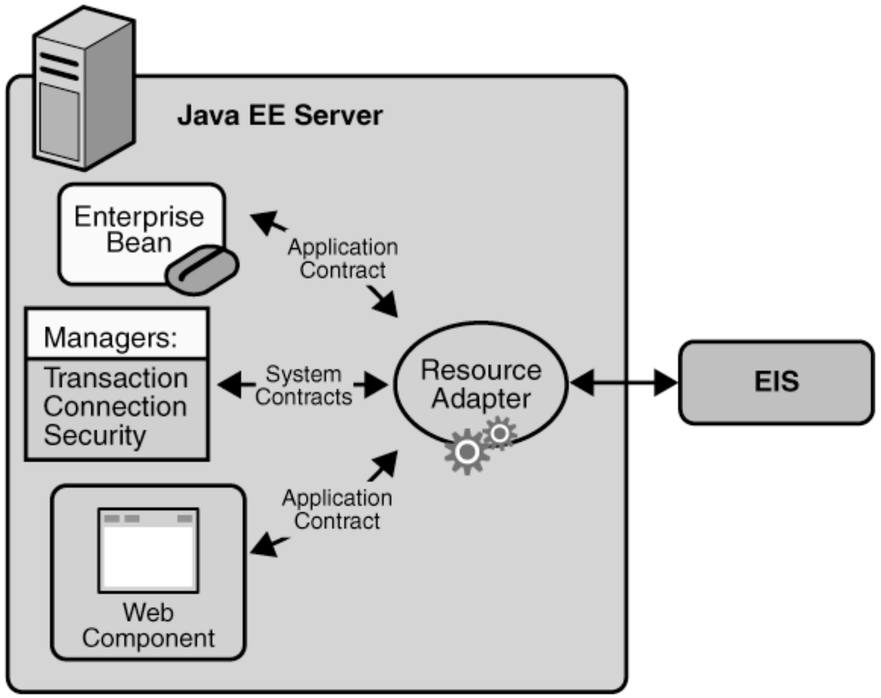
- **CGIs (Common Gateway Interface):**
  - Proporcionan acceso web a aplicaciones empresariales existentes
  - C y Perl
  - Penalizaciones de consumo de tiempo y memoria
  - Información de estado de la sesión difícil de gestionar
- **Java entra en escena**
  - Servlets corren en la JVM
  - Guarda información de estado a través de las sesiones
  - Concurrencia (threading)
  - Presentación (HTML) inmersa en el código
  - Programador vs. diseñador gráfico

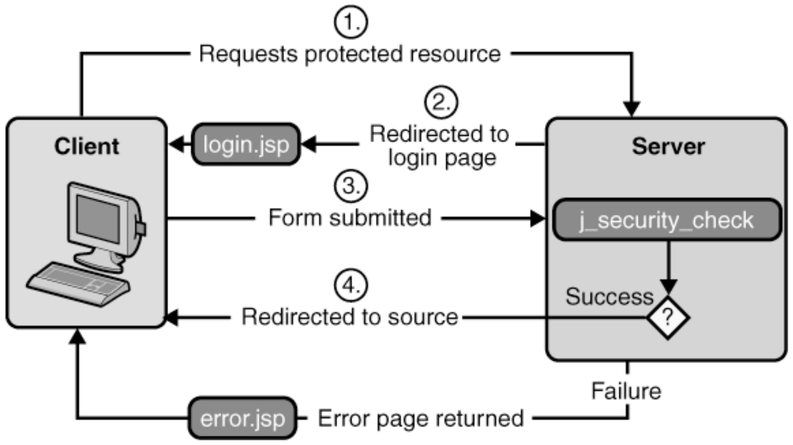
contenedor vs. componente

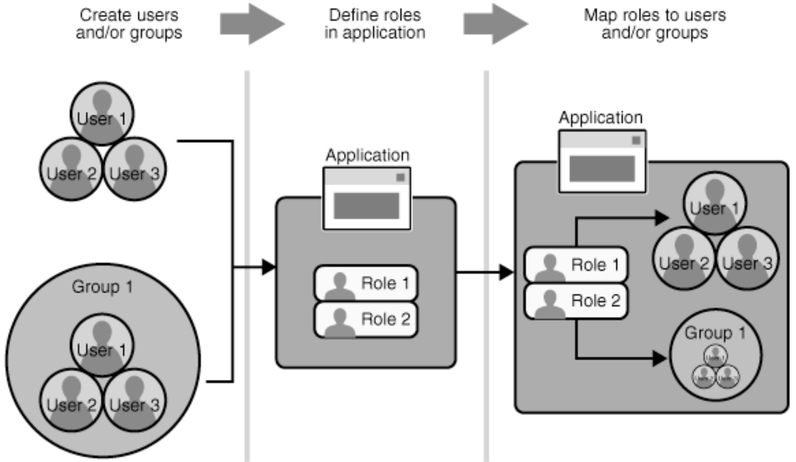
- **Contenedor**:
  - Concurrencia
  - Seguridad
  - Disponibilidad
  - Escalabilidad
  - Persistencia
  - Transacciones
  - Gestión del ciclo de vida
  - Administración (gestión de servicios y monitorización)
- **Componente**:
  - Renderizado
  - Lógica de negocio



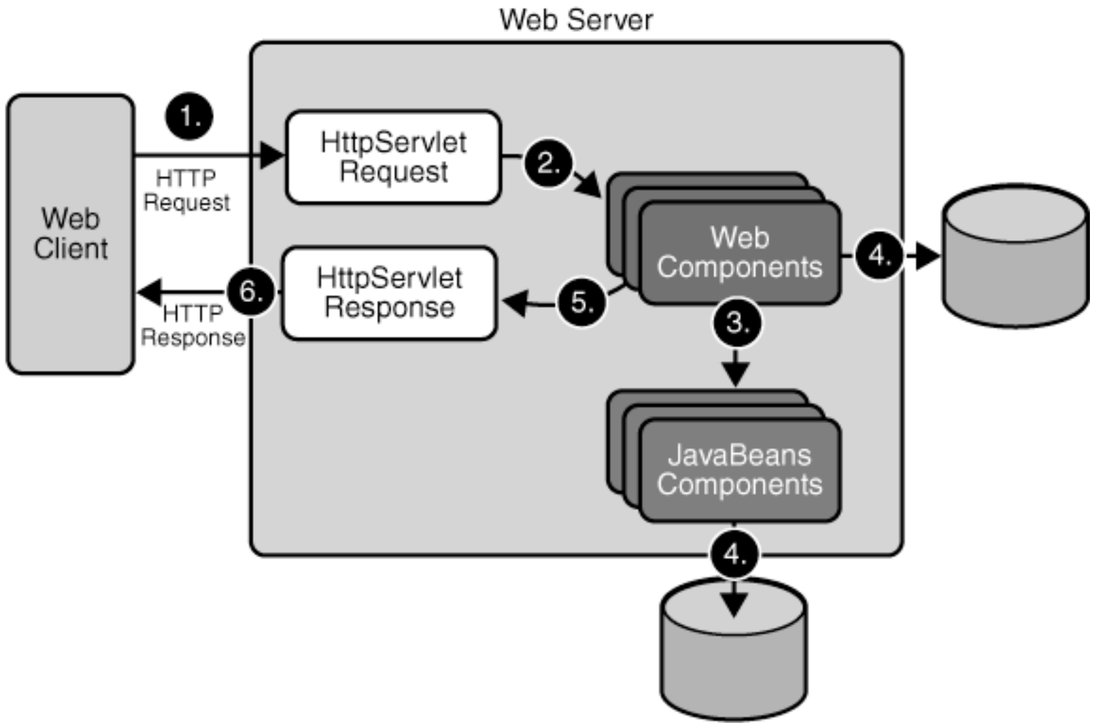
servidor de aplicaciones (i)





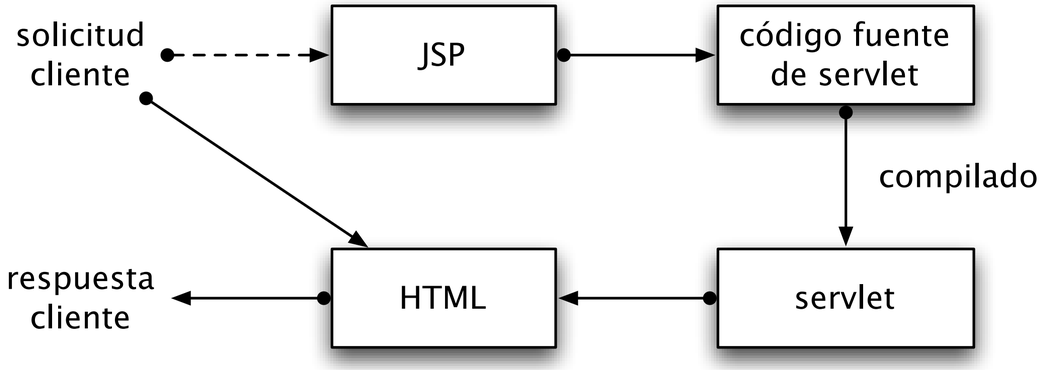


servidor de aplicaciones (ii)



evolución (ii)

- **Lenguajes de Scripting (JSPs, ASPs, ...)**
  - Funcionalidad limitada: dependencia del fabricante
  - Aplicaciones web como páginas estáticas
  - Código de la lógica de negocio inmerso en el código de presentación
  - Diseñador gráfico vs. Programador



- Compilación, despliegue, ...

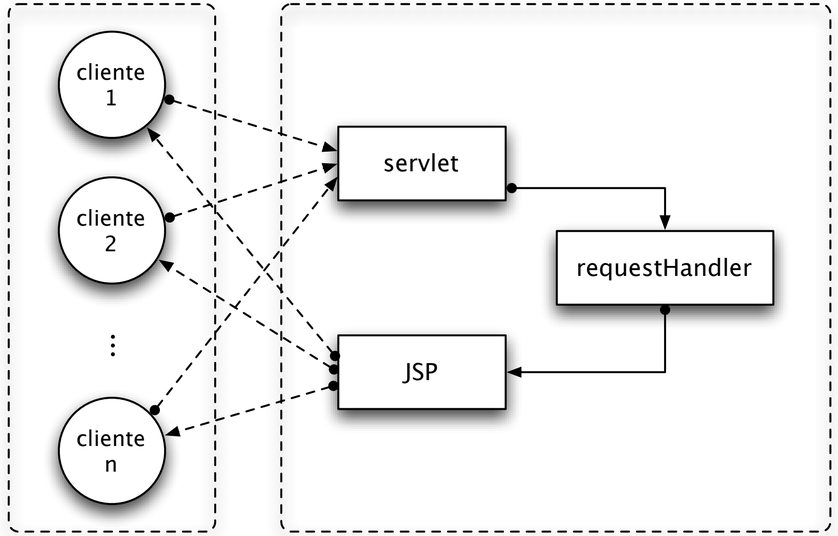
evolution (iii)

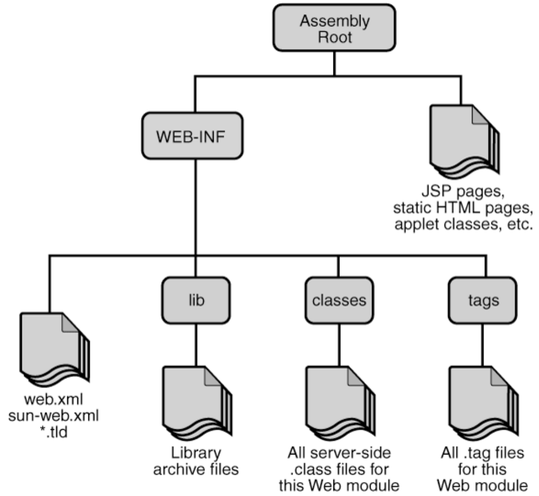
- **Desarrollo web pragmático**:
  - Técnicas híbridas:

    servlets + JSPs

  - JSPs : publicación
  - Servlets: procesamiento

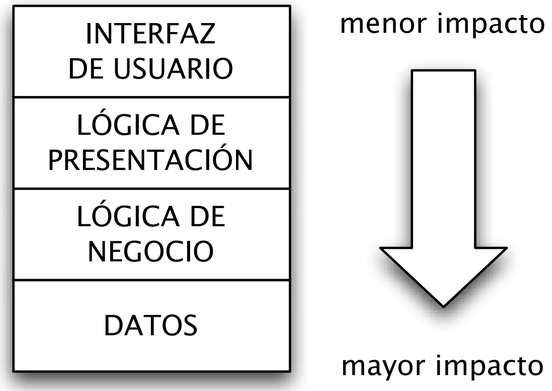
  - Solución completa técnicamente.

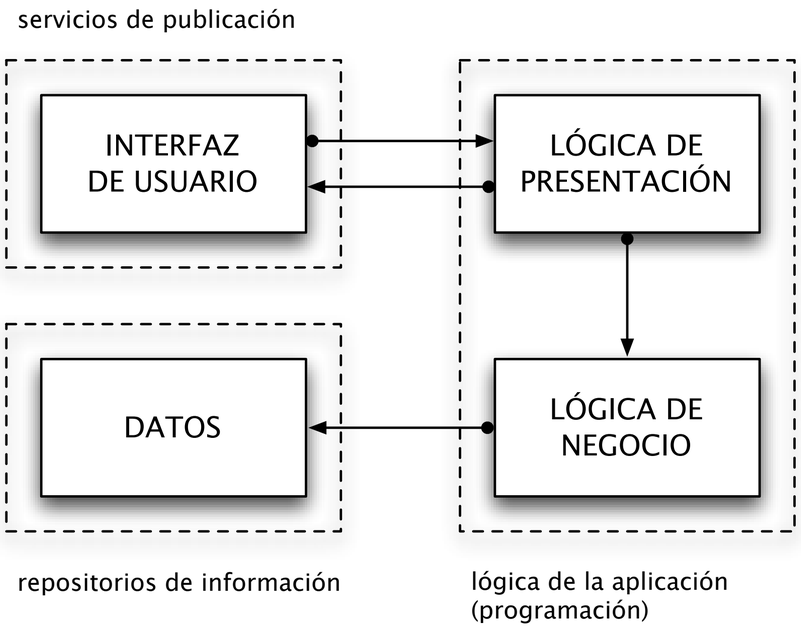




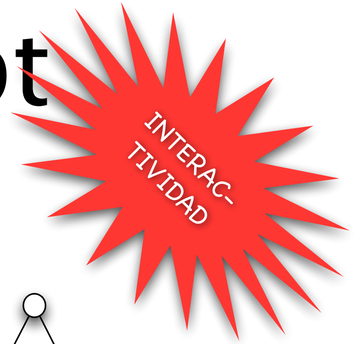
evolución (iv)

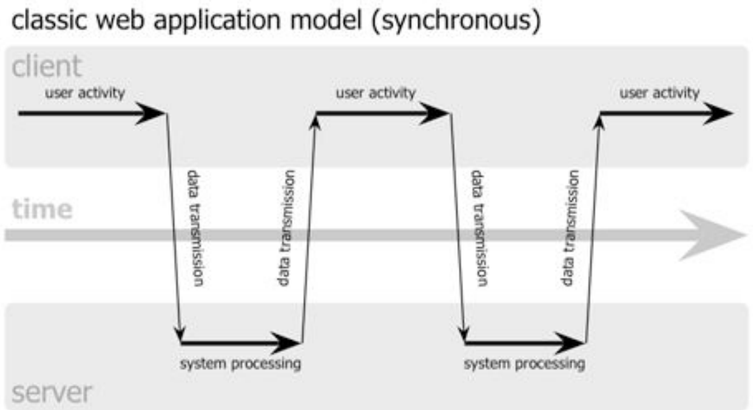
- Consideraciones finales:
  - Separación de la lógica de Negocio y de Presentación
  - **Arquitecturas MVC (Model View Controller)**

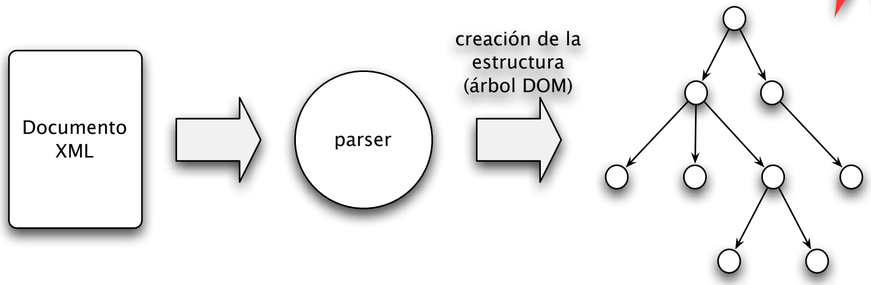


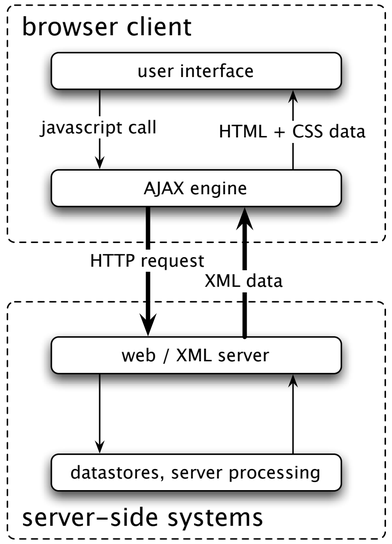


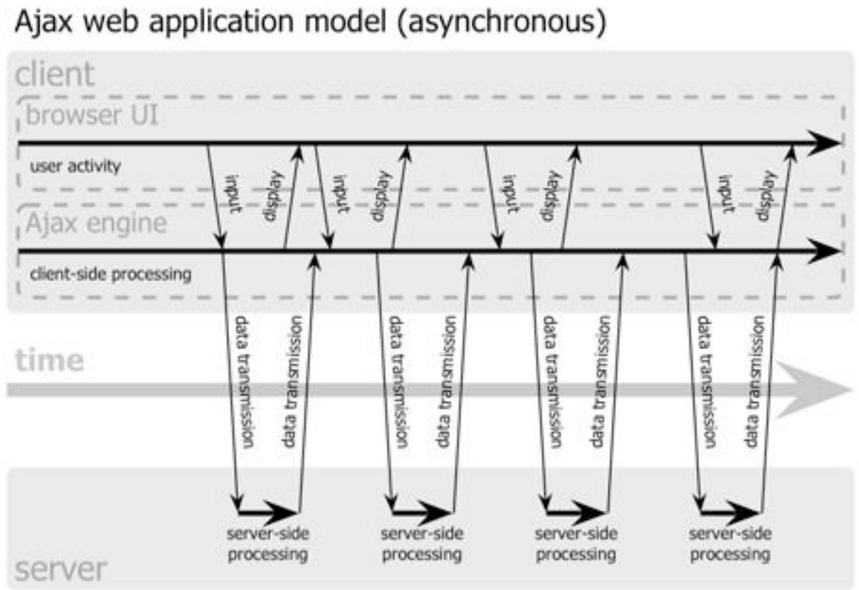
AJAX: asynchronous javascript and XML



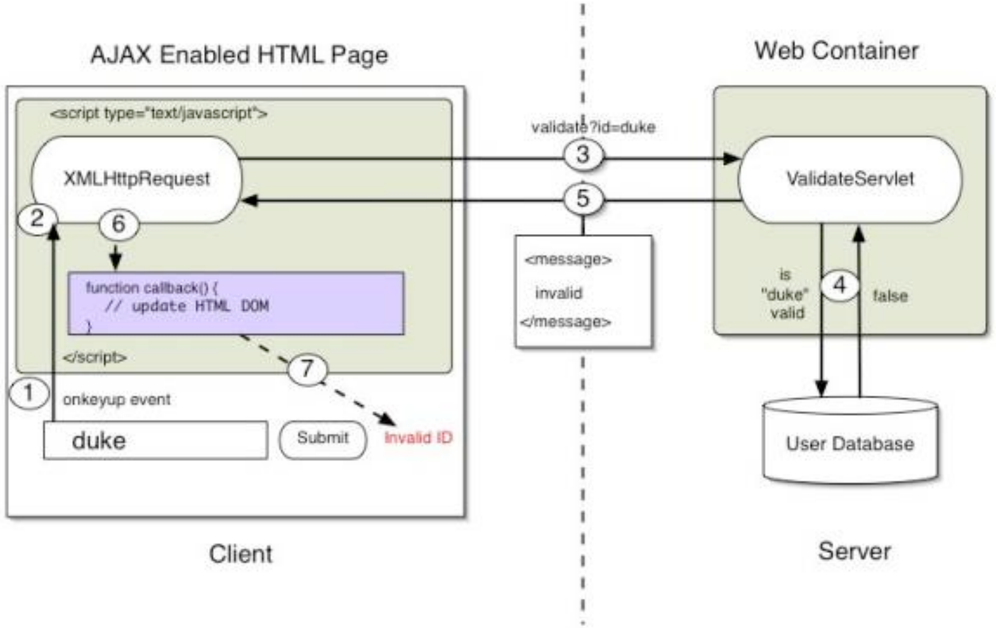


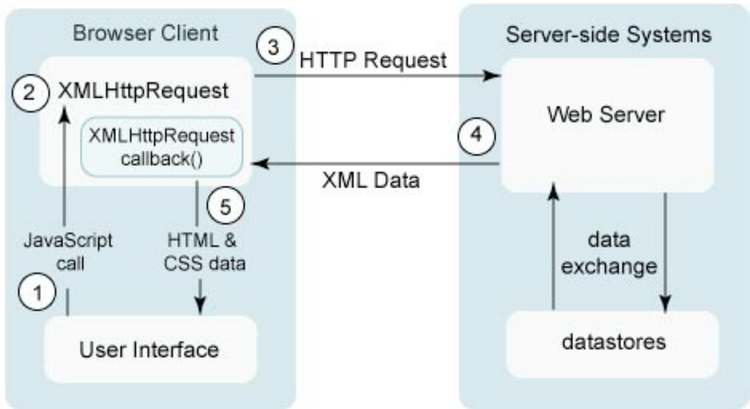




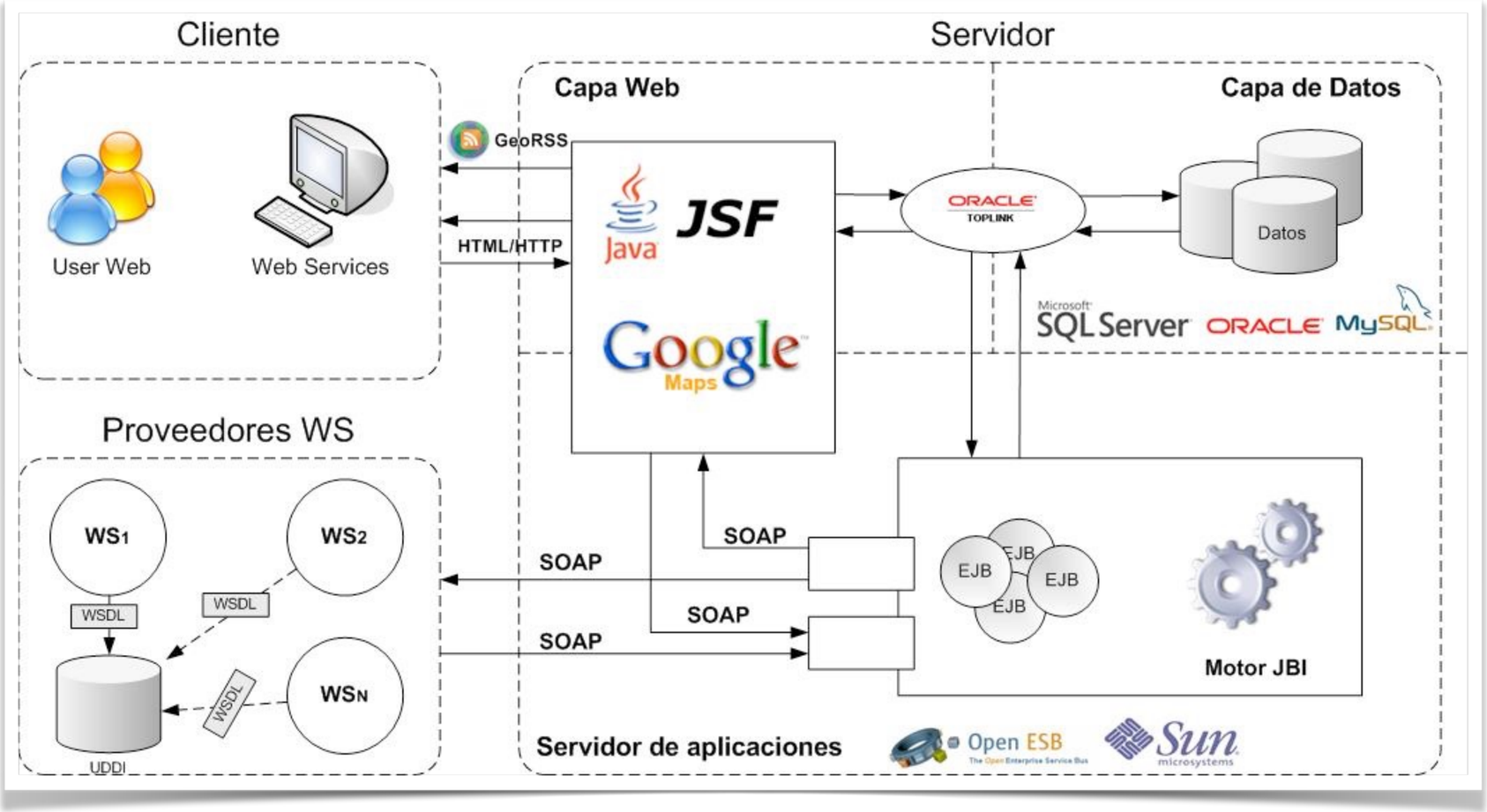


ejemplo Ajax: validación





arquitectura y protocolos de sistema



arquitectura Web

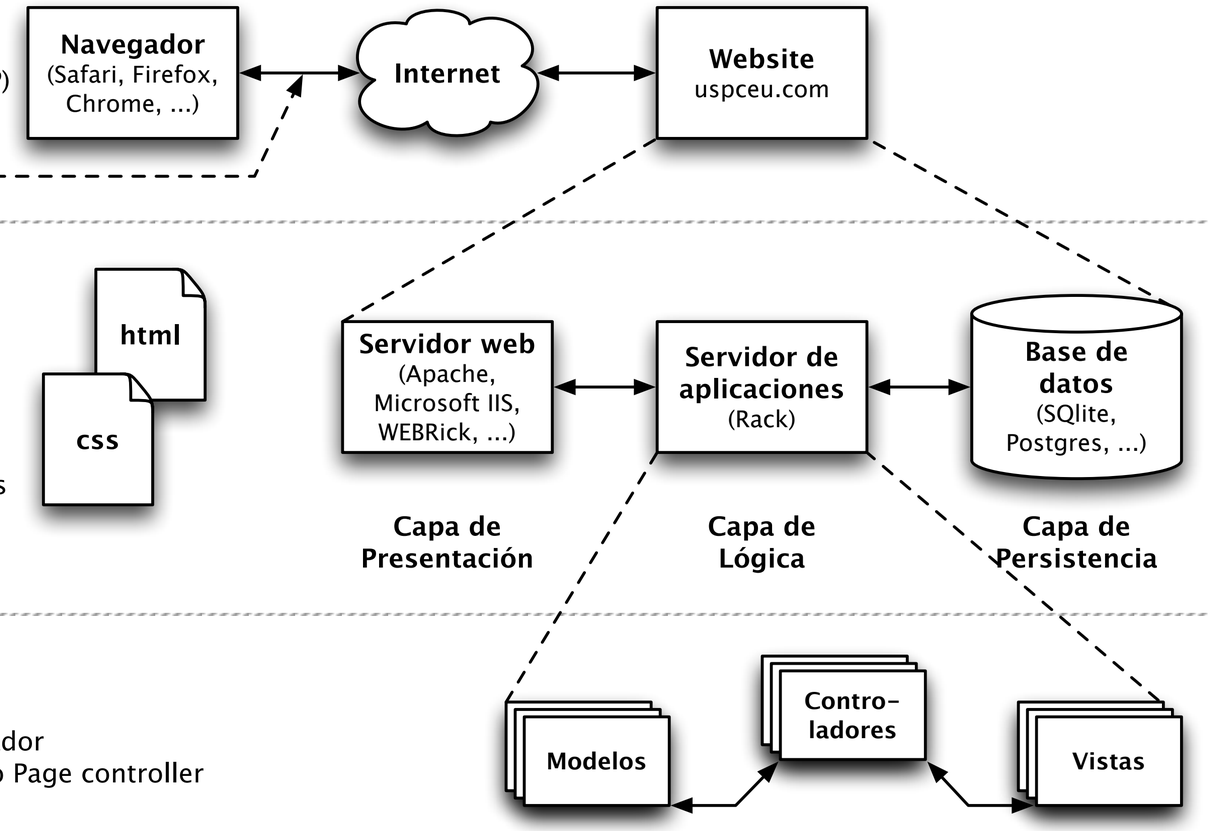
100.000 pies
- Cliente - servidor (vs. P2P)

50.000 pies
- HTTP y URIs

10.000 pies
- XHTML y CSS

5.000 pies
- Arquitectura de tres capas
- Escalado horizontal

1.000 pies
- Modelo - Vista - Controlador
- MVC vs. Front controller o Page controller



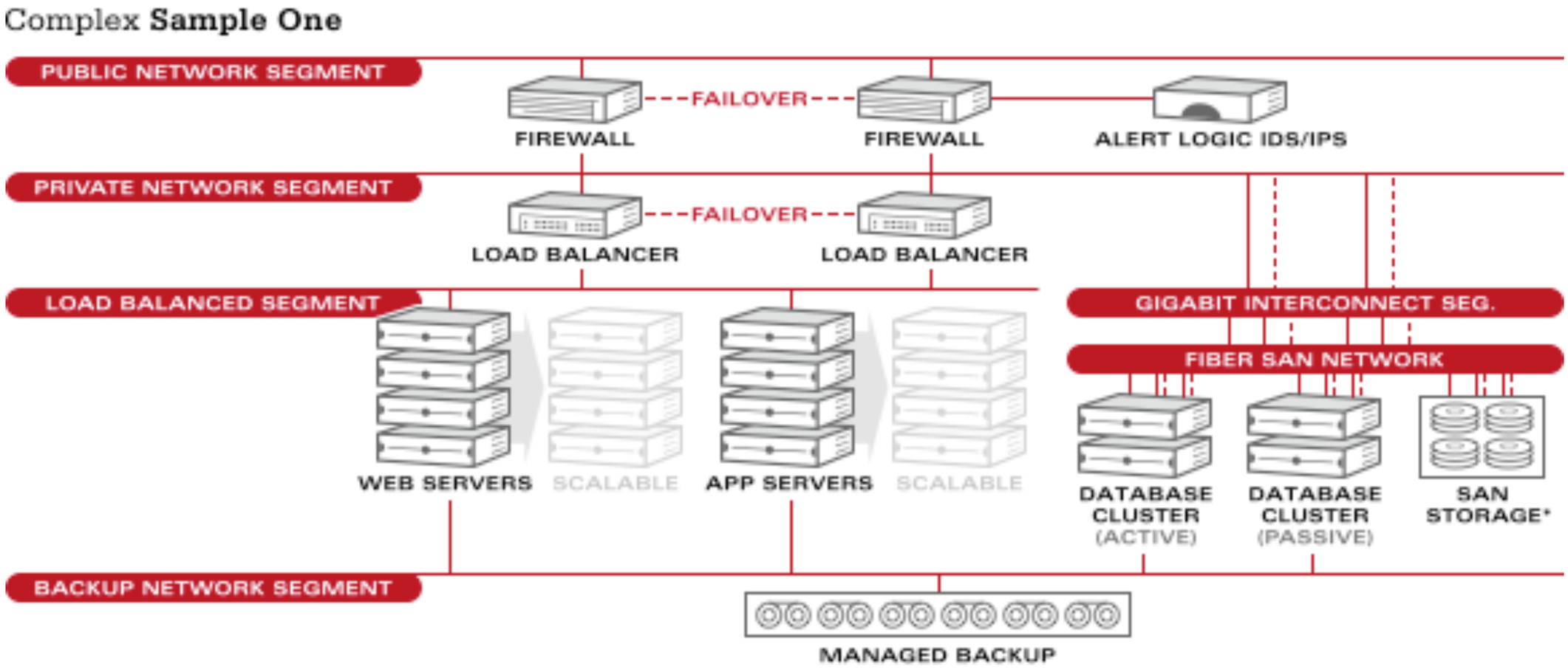
500 pies
- Modelos Active Record vs. Data Mapper
- Controladores RESTful (Representational State Transfer for self-contained actions)
- Template View vs. Transform View

- Active Record
- Data Mapper

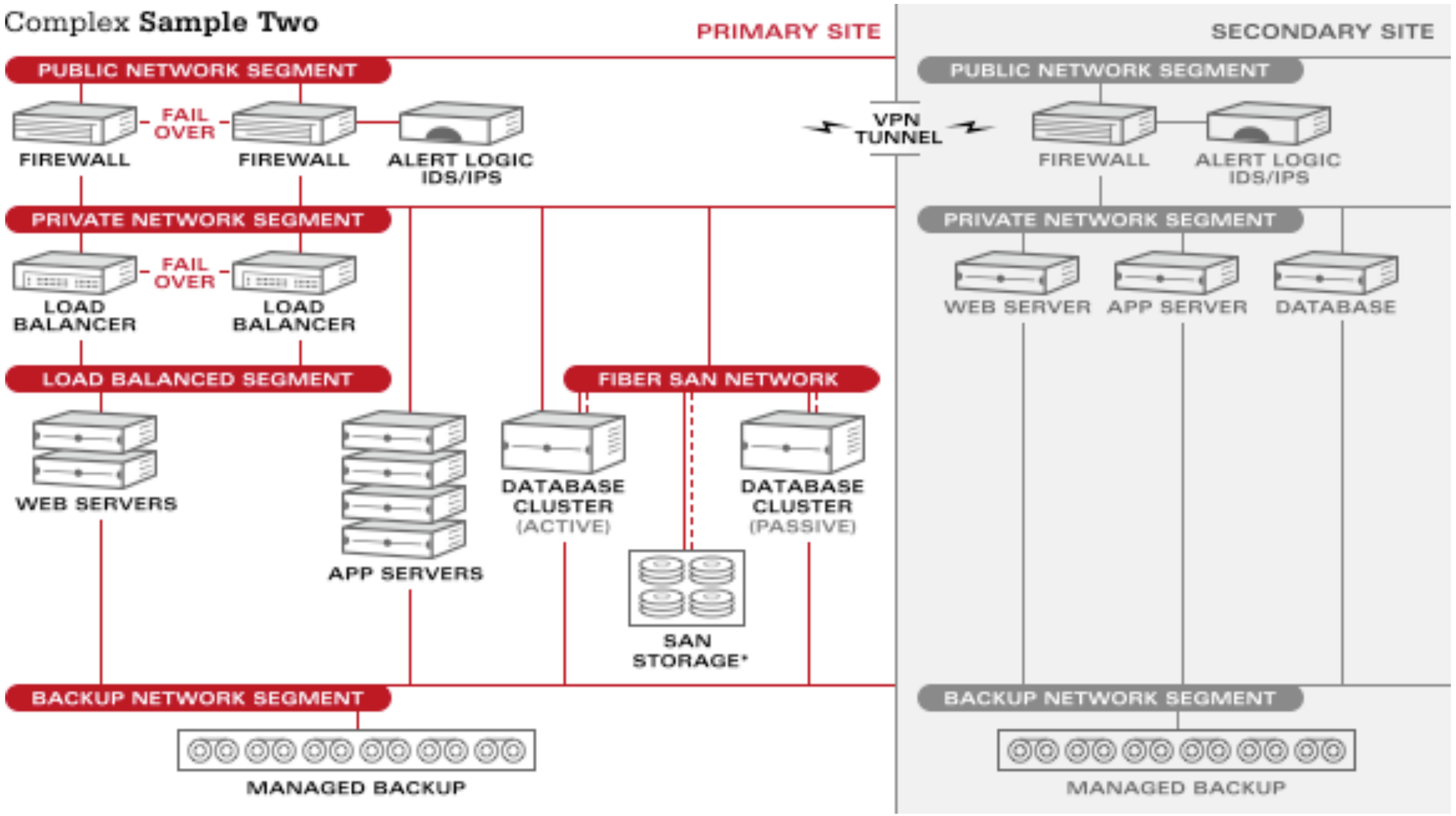
- REST

- Template View
- Transform View

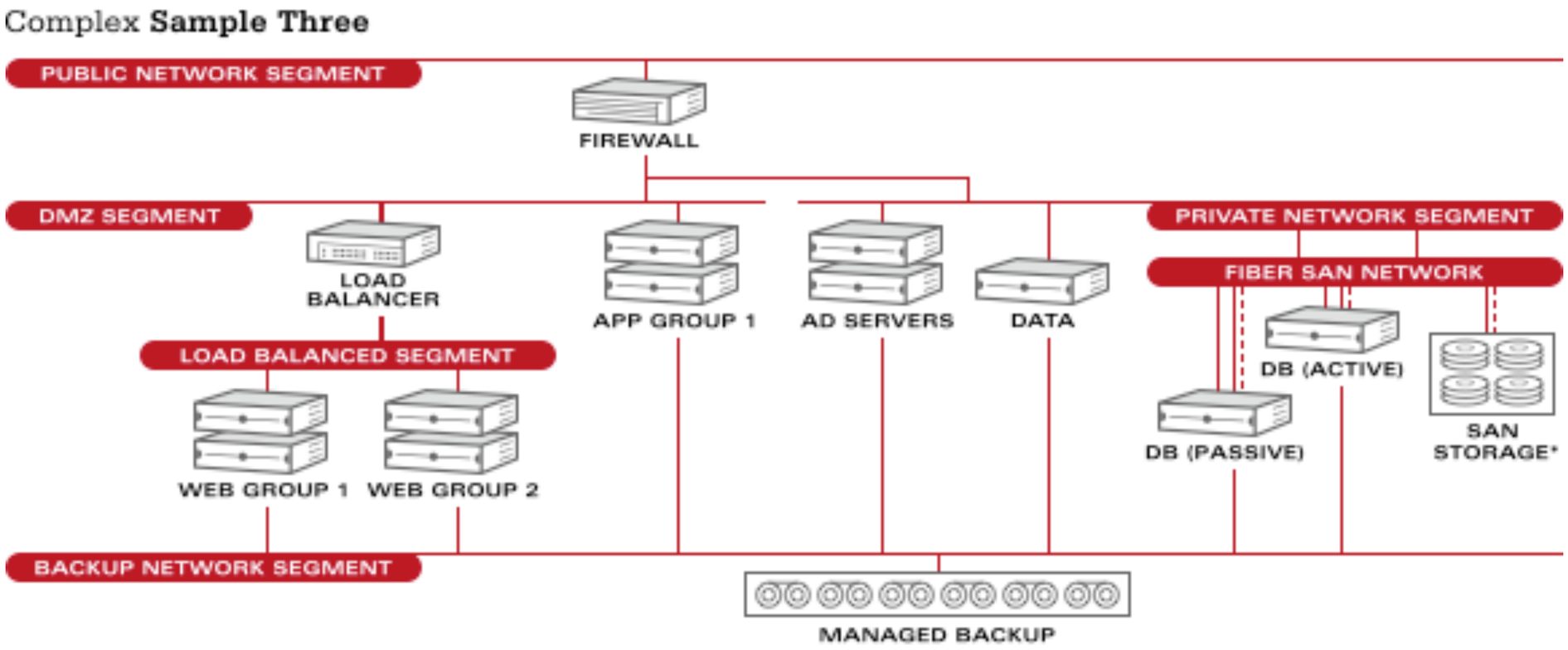
arquitectura de producción (i)



arquitectura de producción (ii)

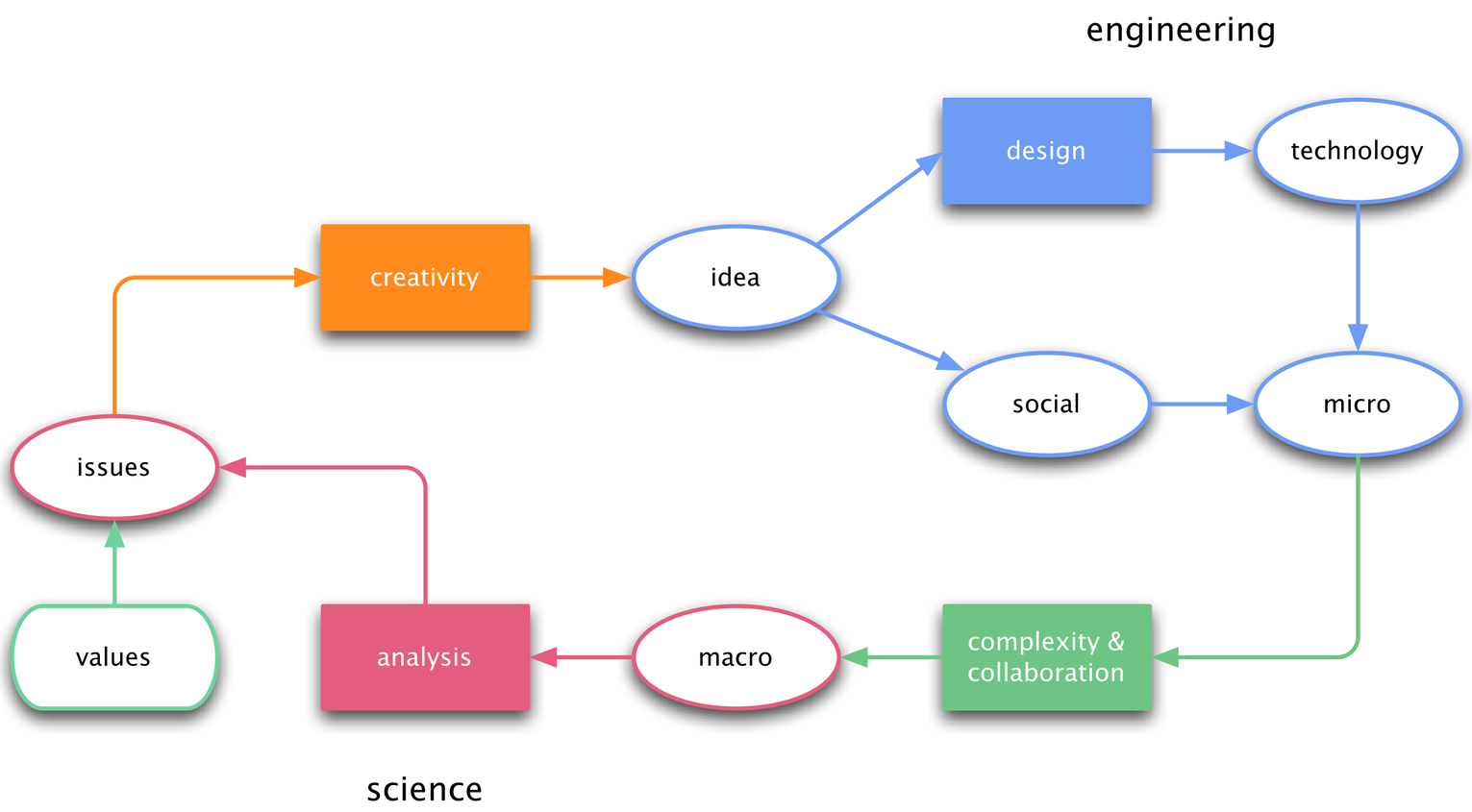


arquitectura de producción (iii)

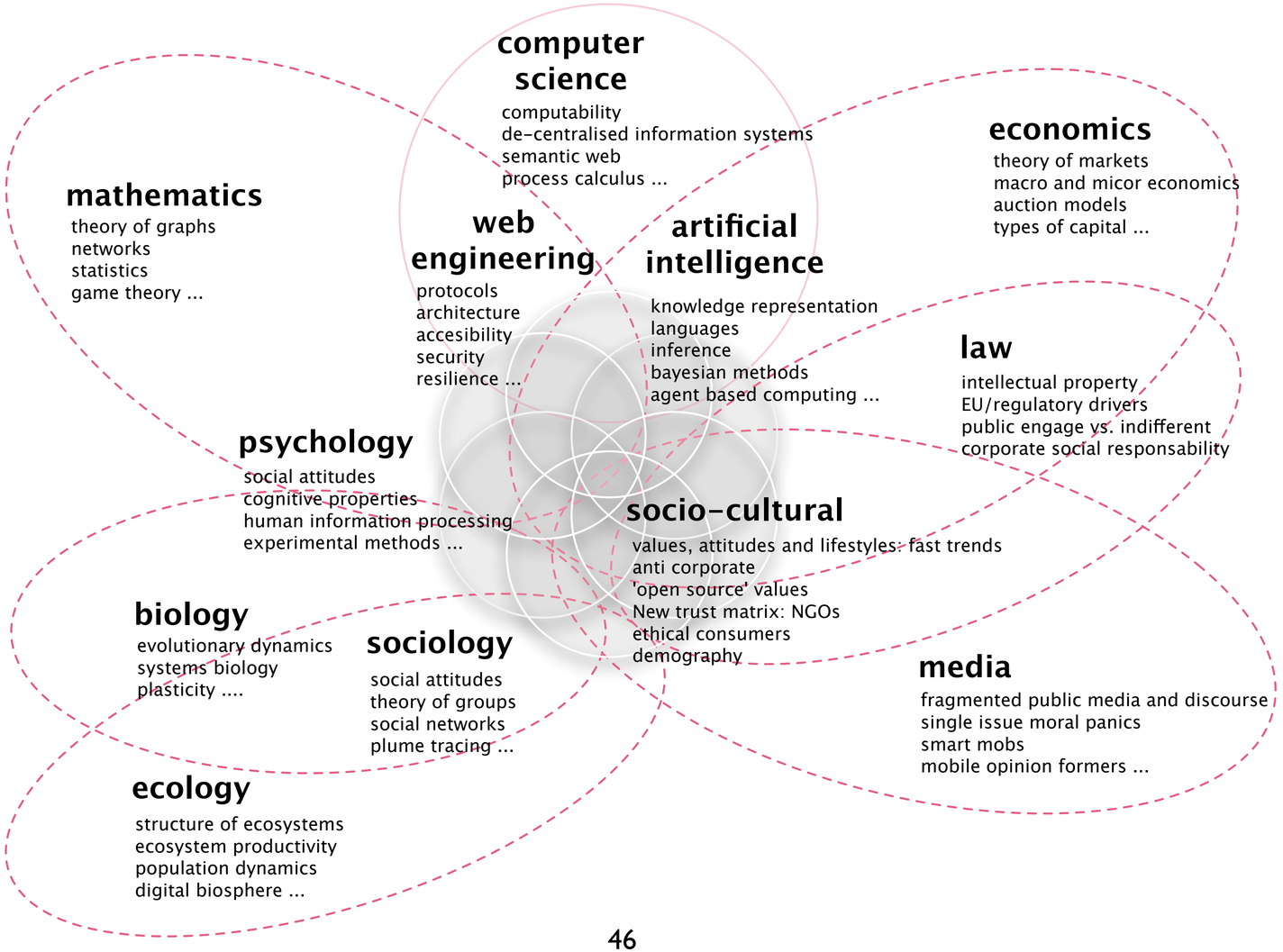


web sciences

ingeniería de la web



disciplinas presentes en las web sciences



computación tradicional vs. web sciences

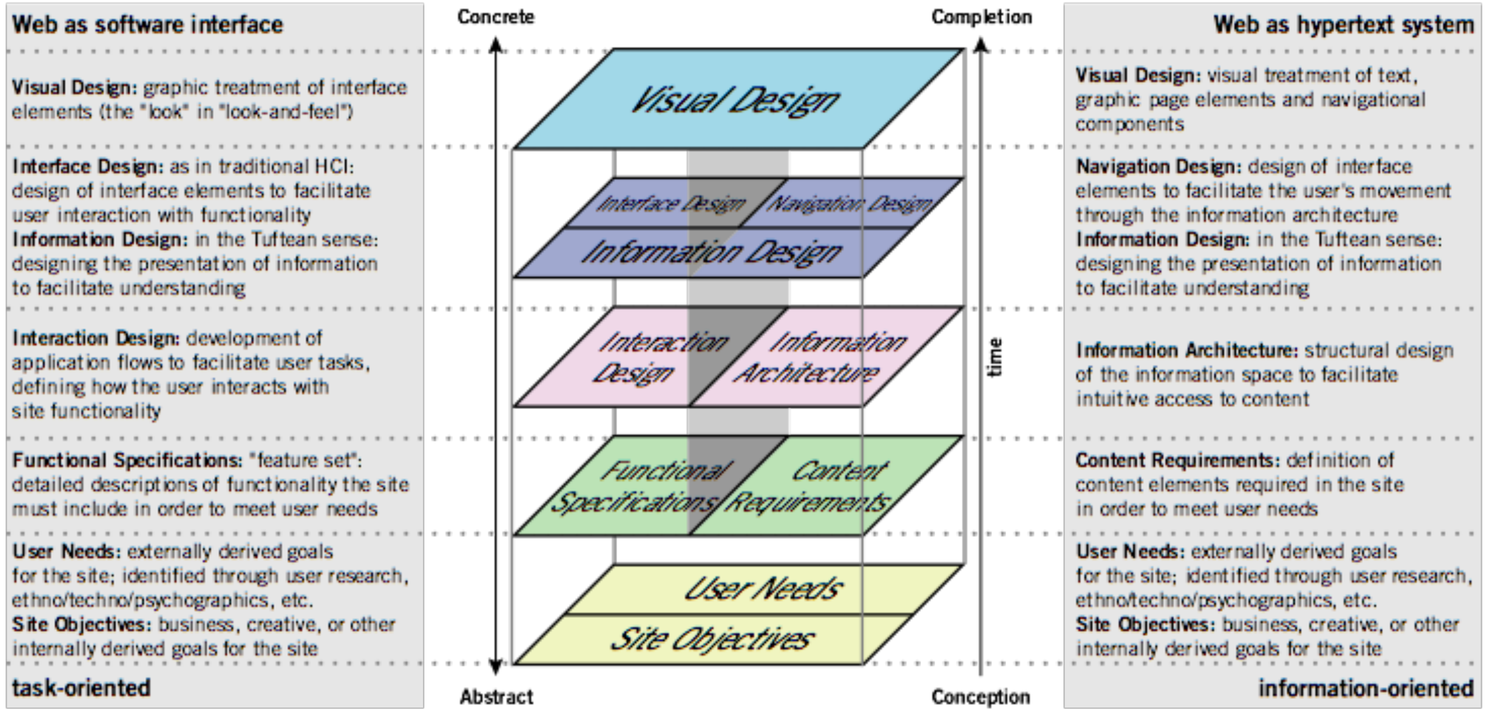
	Computer Science	Web Science
metrics	Moore's Law Order (n) algorithm analysis Gigabytes	Page views Unique visitors/month Number of songs or video
topics	Computer networks Packet switching Information Programming languages Databases, operating systems, compilers 3D graphics, rendering algorithms, computational geometry, object modeling	Social networks Voice over IP, music sharing Relationships Wikis, blogs, tagging E-commerce, e-learning, e-government, medical and health information, financial analysis Authoring and sharing video, animation, music, photos, maps
focus	Technology Computers Supercomputers Proficient programmers	Applications Users Mobile devices Universal usability

Web es más que hipertexto

The Elements of User Experience

Jesse James Garrett
jjg@jjg.net
30 March 2000

A basic duality: The Web was originally conceived as a hypertextual information space; but the development of increasingly sophisticated front- and back-end technologies has fostered its use as a remote software interface. This dual nature has led to much confusion, as user experience practitioners have attempted to adapt their terminology to cases beyond the scope of its original application. The goal of this document is to define some of these terms within their appropriate contexts, and to clarify the underlying relationships among these various elements.



This picture is incomplete: The model outlined here does not account for secondary considerations (such as those arising during technical or content development) that may influence decisions during user experience development. Also, this model does not describe a development process, nor does it define roles within a user experience development team. Rather, it seeks to define the key considerations that go into the development of user experience on the Web today.

© 2000 Jesse James Garrett

http://www.jjg.net/ia/

equipos de desarrollo web

The Nine Pillars
of Successful Web Teams

Jesse James Garrett <jjg@jjg.net>
9 July 2003

The most successful Web teams build their team structures and their processes on these nine essential competencies:

Project Management: The hub that binds all the tactical competencies together as well as the engine that drives the project forward to completion, project management requires a highly specialized set of skills all its own. Neglecting this area often results in missed deadlines and cost overruns.

Concrete Design: Before the abstract design can become a fully realized user experience, you must determine the specific details of interfaces, navigation, information design, and visual design. This realm of concrete design is essential to creating the final product.

Content Production: Knowing what content you need isn't enough. You also need to know how you'll produce it. Gathering raw information, writing and editing, and defining editorial workflows and approvals are all part of content production.

Technology Implementation: Building technical systems involves a lot of hard work and specialized knowledge: languages and protocols, coding and debugging, testing and refactoring. The more complex your site, the more important a competency in technology implementation becomes.

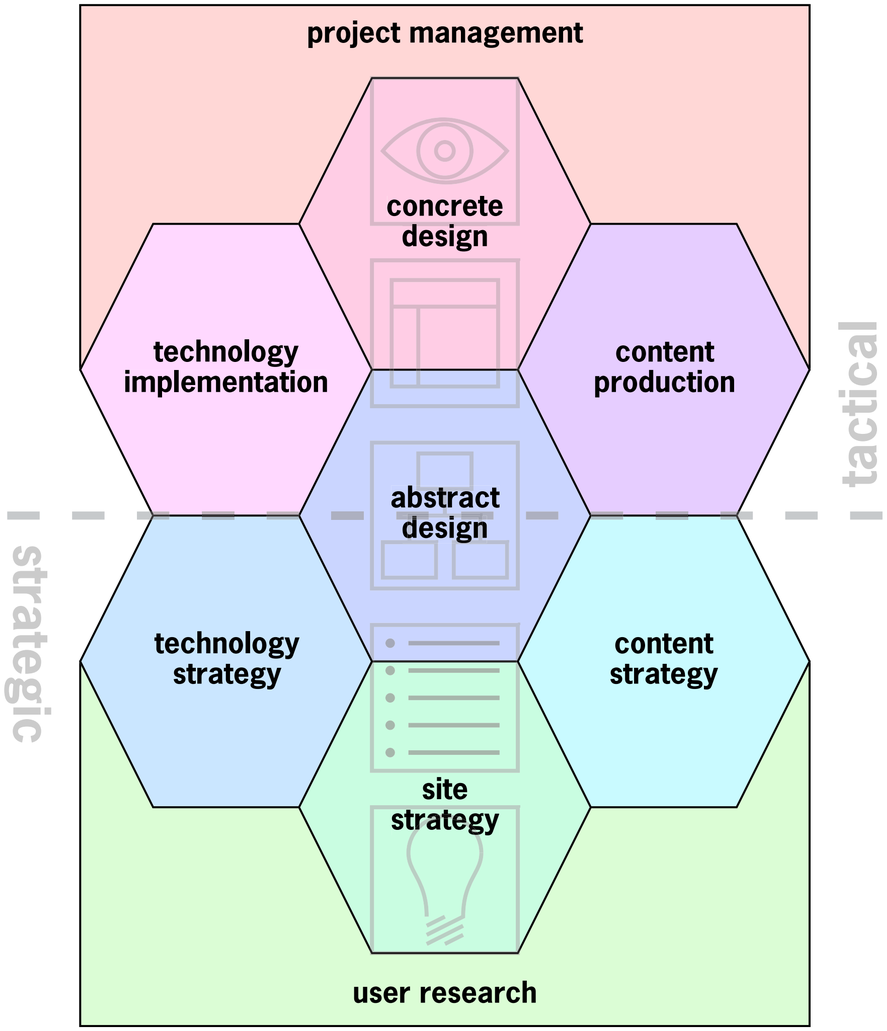
Abstract Design: Information architecture and interaction design translate strategic objectives into a conceptual framework for the final user experience. These emerging disciplines addressing abstract design are increasingly recognized for their value in the Web development process.

Content Strategy: Content is often the reason users come to your site. But what content can you offer to meet your users' expectations? How much content is appropriate, and what form should it take? What style or tone should it have? Before you can produce that content, you need to answer fundamental content strategy questions such as these.

Technology Strategy: Web sites are technologically complex, and getting more intricate all the time. Identifying the technology strategy for the site – platforms, standards, technologies, and how they can all interoperate – is essential to avoiding costly mistakes.

Site Strategy: Defining your own goals for the site can be surprisingly tricky. Arriving at a common understanding of the site's purpose for your organization, how you'll prioritize the site's various goals, and the means by which you'll measure the site's success are all matters of site strategy.

User Research: User-centered design means understanding what your users need, how they think, and how they behave – and incorporating that understanding into every aspect of your process. User research provides the raw observations that fuel this insight into the people your site must serve.

adaptive path

http://www.jjg.net/ia/pillars/
http://www.adaptivepath.com/