

Sistemas Operativos y Distribuidos

Iren Lorenzo Fonseca
iren.fonseca@ua.es



TEMA 3. Sistemas Distribuidos.

Tecnologías para los
Sistemas Distribuidos

Tecnologías para los Sistemas Distribuidos

Contenidos



Mecanismos de comunicación distribuida

IPC, Sockets, RPC, RMI, ORB



Revisión de tecnologías Web

Modelo HTTP Básico



Servicios Web

SOA, REST, gRPC, GraphQL, WebSocket

- ✓ Teoría General
- ✓ Transmisión de información
- ✓ Protocolos
- ✓ Sockets, RPC, RMI, ORB

Mecanismos de comunicación

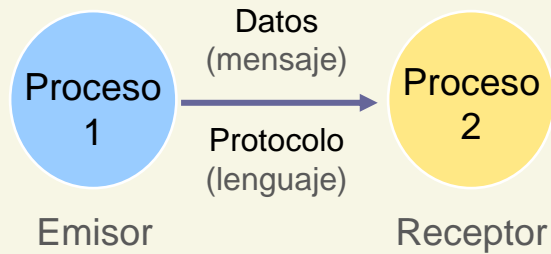
Contenidos

Teoría General

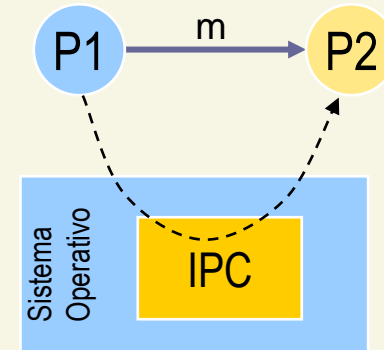
Teoría General

Introducción

IPC: conjunto de herramientas básicas del SO
→ comunicación entre procesos distribuidos



Elementos que intervienen en la comunicación entre procesos. Teoría de la comunicación



Arquitectura para la comunicación entre procesos mediante IPC

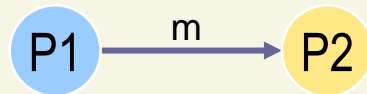
Teoría General

Introducción

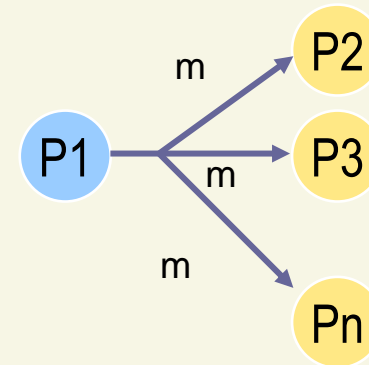
| **IPC:** conjunto de herramientas básicas del OS
→ comunicación entre procesos distribuidos

| Modelos básicos:

- | Unidifusión (unicast)
- | Multidifusión (multicast)



Enfoque Unicast



Enfoque Multicast

Teoría General

Introducción

- | **IPC**: conjunto de herramientas básicas del OS
 - comunicación entre procesos distribuidos
- | Modelos básicos:
 - | Unidifusión (**unicast**)
 - | Multidifusión (**multicast**)
- | Definición de interfaz:
 - | **Conectar** (solicitar-conexión/aceptar-conexión) y **Desconectar**
 - | **Enviar** y **Recibir**
- | Sincronización básica, temporizadores e interbloqueos
 - | Operaciones **bloqueantes** o síncronas
 - | Operaciones **no bloqueantes** o asíncronas

Teoría General

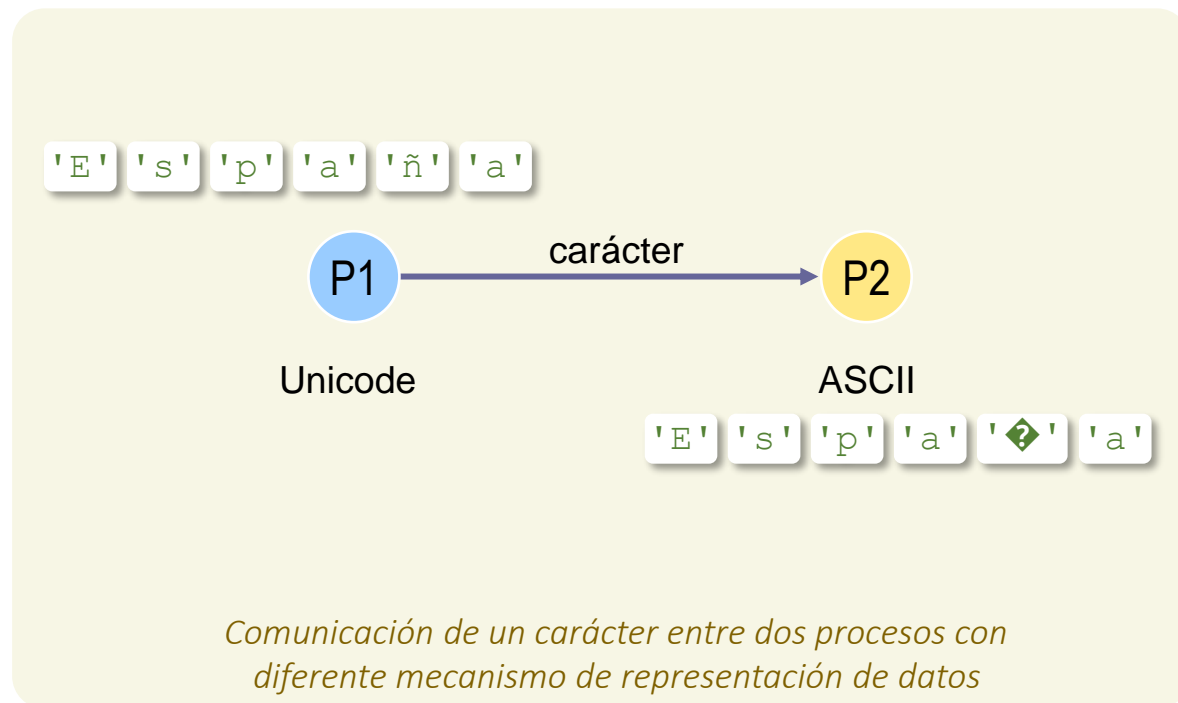
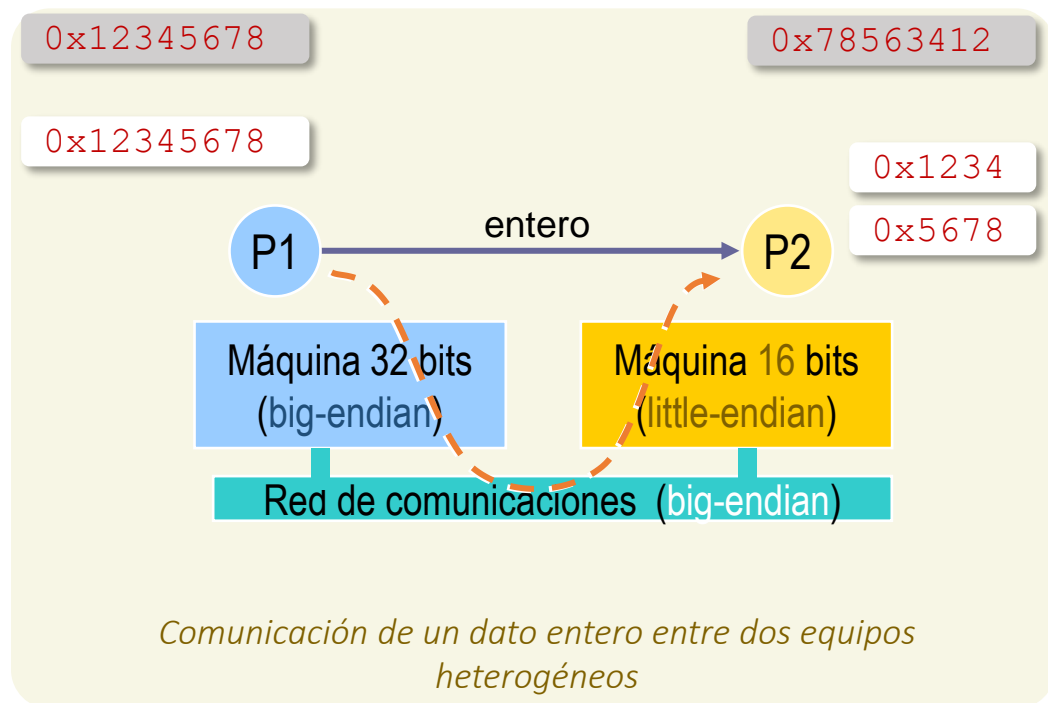
Características

Aunque a nivel de red

Es un flujo binario

Existe mucha heterogeneidad

Representación de la información



Teoría General

Características

Aunque a nivel de red

Es un flujo binario

Existe mucha heterogeneidad

Representación de la información

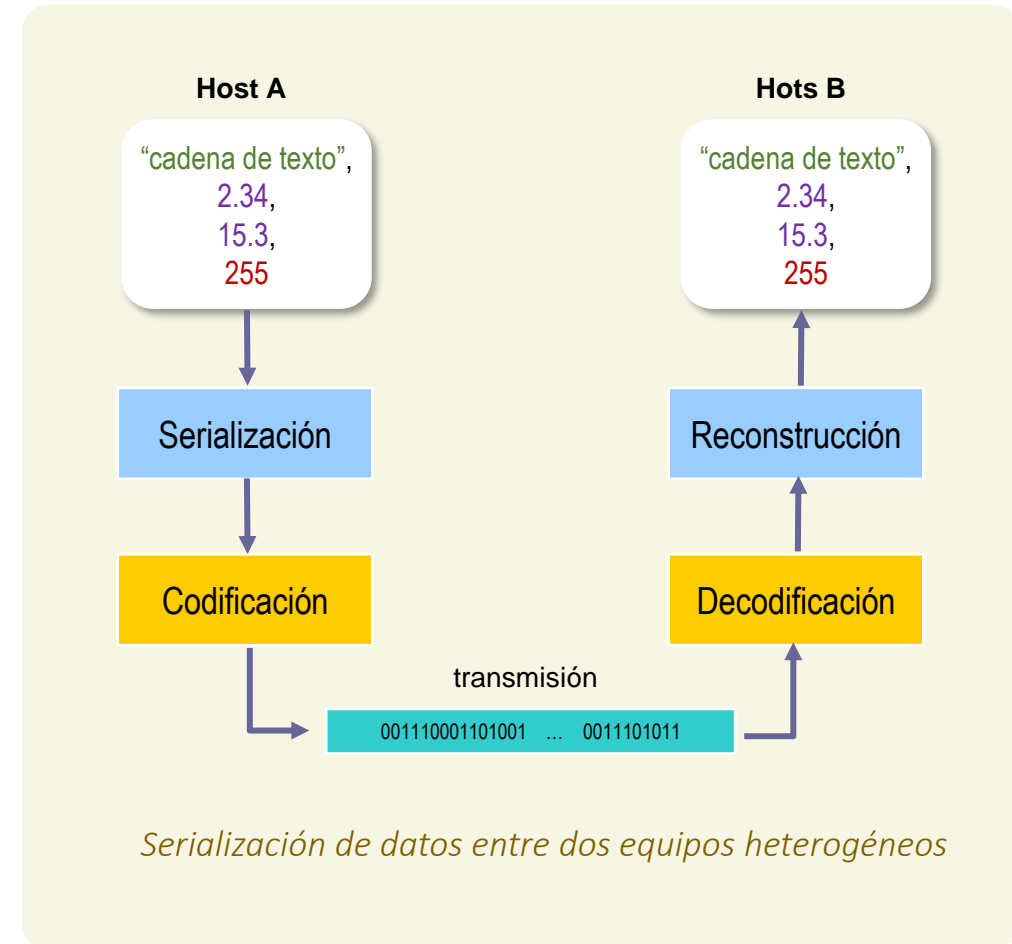
Solución

- ✓ Emisor adapta a la representación del receptor
- ✓ Receptor adapta la representación del emisor
- ✓ Representación externa común

Empaquetamiento de datos

Tipos complejos → Serialización

Objetos → Marshalling



Teoría General

Características

Aunque a nivel de red

Es un flujo binario

Existe mucha heterogeneidad

Representación de la información

Solución

- ✓ Emisor adapta a la representación del receptor
- ✓ Receptor adapta la representación del emisor
- ✓ Representación externa común

Empaquetamiento de datos

Tipos complejos → Serialización

Objetos → Marshalling

Codificación de los datos

Normalización → XDR / ASN.1 / XML / JSON / Protocol Buffer / ...

XDR (External Data Representation)

```
struct Person {  
    string name<>;  
    int age;  
    string email<>;  
};
```

ASN.1 (Abstract Syntax Notation One)

```
Person ::= SEQUENCE {  
    name UTF8String,  
    age INTEGER,  
    email IA5String  
}
```

XML (Extensible Markup Language)

```
<Person>  
  <name>Juan López</name>  
  <age>30</age>  
  <email>jlp@ua.es</email>  
</Person>
```

JSON (JavaScript Object Notation)

```
{  
  "name": "Juan López",  
  "age": 30,  
  "email": "jlp@ua.es"  
}
```

Protocol Buffer (protobuf)

```
syntax = "proto3";  
  
message Person {  
    string name = 1;  
    int32 age = 2;  
    string email = 3;  
}
```

Teoría General

Clasificación

Basados en texto o binario



Texto: HTTP, SMTP, POP3, IMAP



Binarios: FTP

Patrón de comunicación



Petición-Respuesta: FTP, HTTP, SMTP, IMAP



Publicación-Suscripción: MQTT, XMPP, AMQP

Orientados o no a la conexión



Con conexión (TCP): HTTP, FTP, IMAP, SMTP



Sin conexión (UDP): DNS, DHCP

Con estado (*state*) o sin estado (*stateles*)



Con estado: FTP, SSH



Sin estado: HTTP

Mecanismos de Comunicación

Mecanismos de Comunicación

Paso de mensaje (Sockets)

Llamadas a procedimientos remotos (RPC)

Invocación de métodos remotos (RMI)

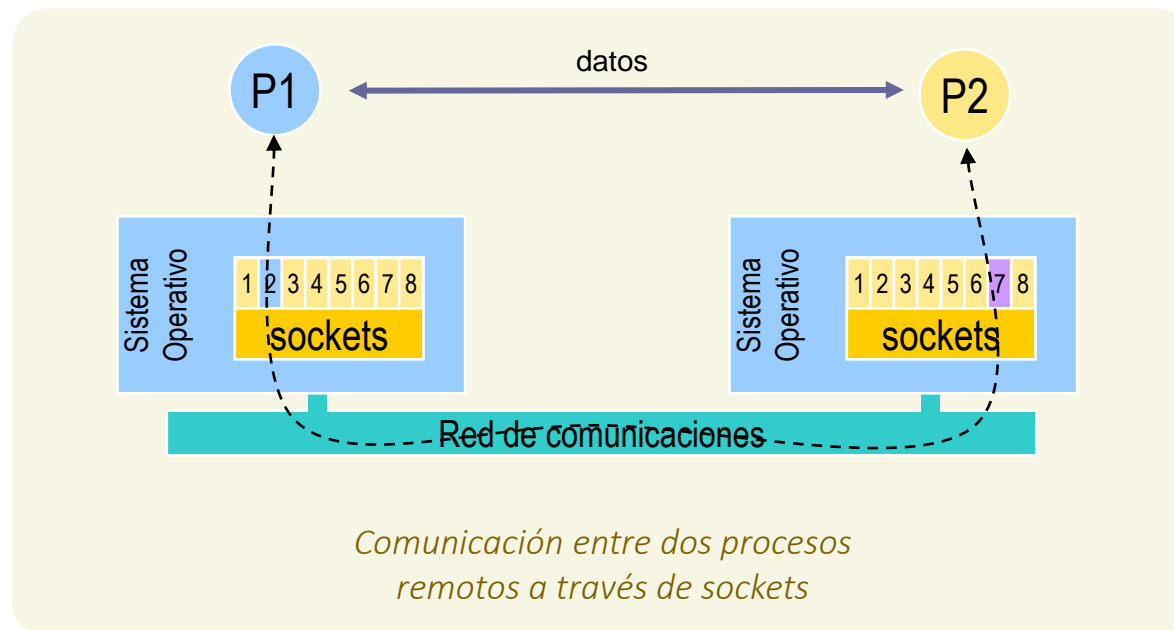
Intermediario de petición de objetos (ORB)

Paso de mensaje

Sockets

Sockets es un **mecanismo IPC** básico de intercambio de datos a través de un conector (socket):

Constituyes un punto final de **comunicación bidireccional** entre **procesos** en una red, multiplexados mediante **Puertos**.



Paso de mensaje

Sockets

Sockets es un **mecanismo IPC** básico de intercambio de datos a través de un conector (socket):

Constituyes un punto final de **comunicación bidireccional** entre **procesos** en una red, multiplexados mediante **Puertos**.

API de sockets: biblioteca de programación

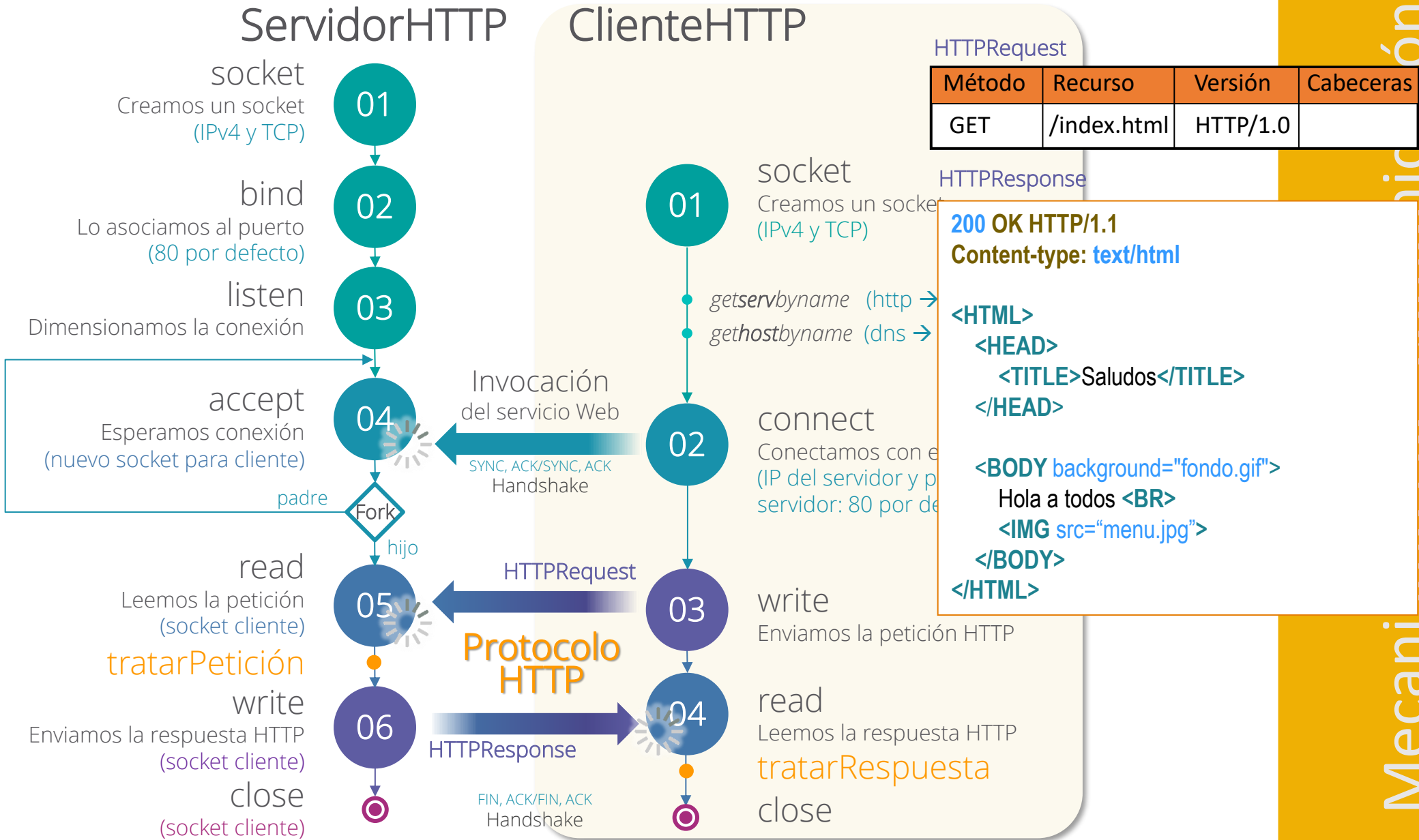
Protocolos Soportados:

TCP (orientado a conexión): flujo de datos (paquetes), confiable, mantiene conexión.

UDP (sin conexión): datagramas, rápido, sin necesidad de conexión establecida.

Principales llamadas al sistema:

`socket()`, `bind()`, `connect()`, `send()`, `recv()`, `close()`




```
void ClienteHTTP(char *dirIP, int puerto, char *recurso)
```

```
{
```

```
    int sd; // descriptor del socket del cliente
    static struct sockaddr_in sa; // dirección IPv4 del servidor
    int bRecibidos;
    char HTTP_request[8000]; // Reservamos 8KB para la petición
    char HTTP_response[40000]; // Reservamos 40KB para la respuesta
```

01

```
    // SOCKET: obtenemos el socket (sd) del tipo: IPv4 y TCP
    sd = socket(AF_INET, SOCK_STREAM, 0);
```

02

```
    // CONNECT: preparamos la conexión con el servidor (dirIP:puerto)...
    sa.sin_family = AF_INET;
    sa.sin_addr.s_addr = inet_addr(dirIP);
    sa.sin_port = htons((uint16_t)puerto);
```

03

```
    // ...Conectamos con el servidor y puerto indicados
    connect(sd, (struct sockaddr *)&sa, sizeof(sa));
```

04

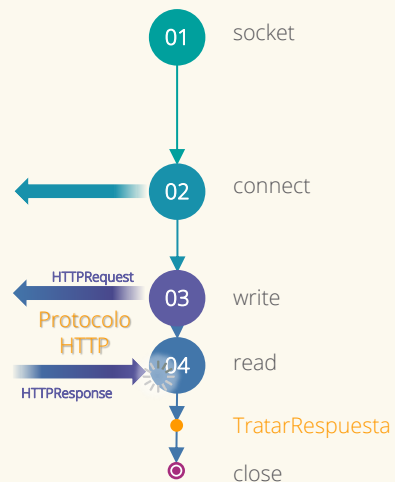
```
    // HTTP_REQUEST: enviamos la solicitud GET sobre el recurso solicitado (index.html por defecto)
    sprintf(HTTP_request, "GET %s HTTP/1.0\r\n\r\n", recurso);
    write(sd, HTTP_request, strlen(HTTP_request));
```

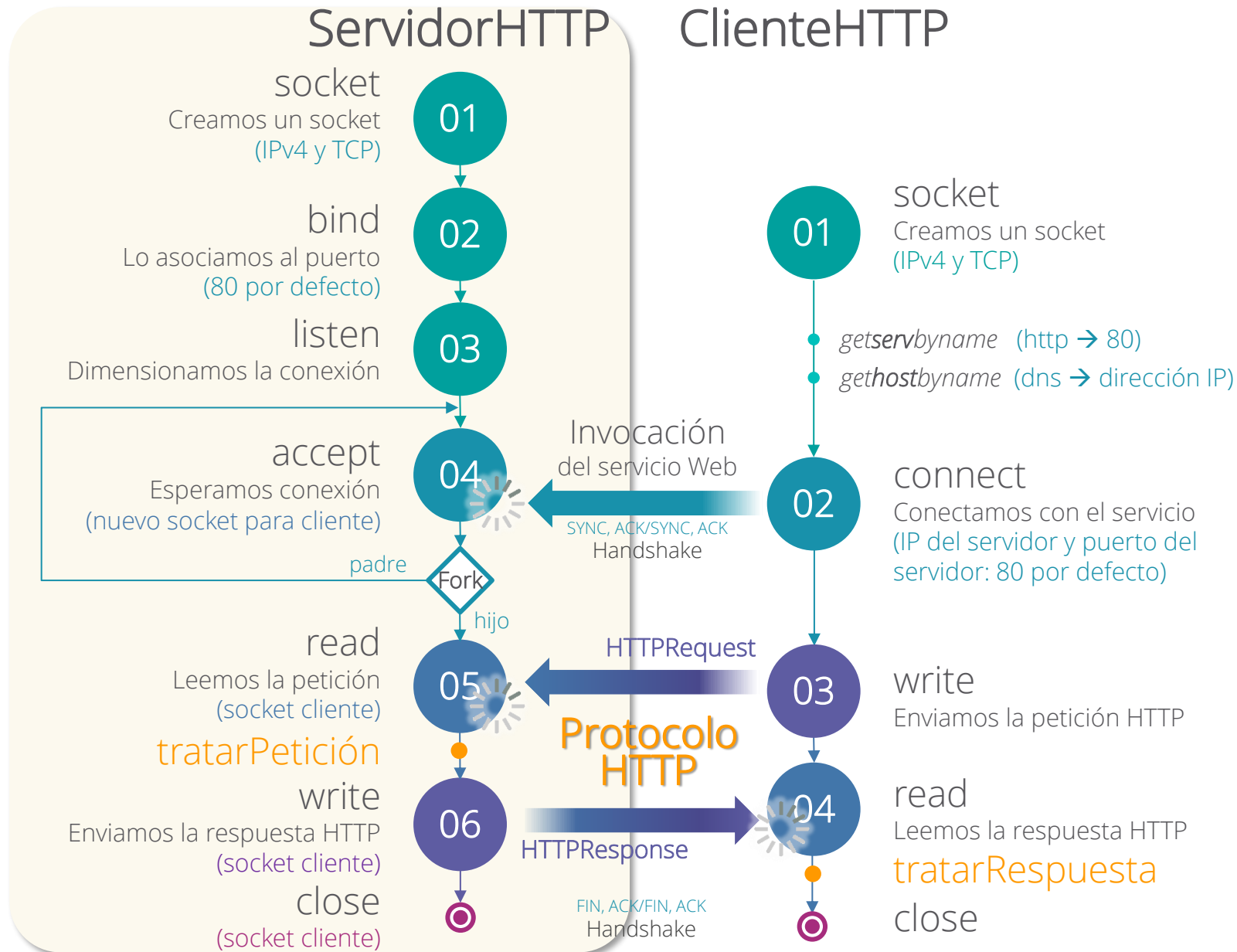
```
    // HTTP_RESPONSE: leemos la respuesta en HTTP_response (máximo 40KB)
    bRecibidos = (int)read(sd, HTTP_response, sizeof(HTTP_response));
```

```
    // Tratamos la respuesta: escribimos en terminal (tanto la cabecera como el cuerpo)
    printf("%s\n", HTTP_response);
```

```
    // CLOSE: Cerramos el socket
    close(sd);
}
```

ClienteHTTP





Paso de mensaje

servidor WEB

codificado en *pseudo-código*

servidorHTTP()

Begin

01

sd = socket(IPv4, TCP)

02

bind(sd, 80)

03

listen(sd, 5)

04

Do

sdHijo = accept(sd)

05

While (fork() != 0)

request = read(sdHijo)

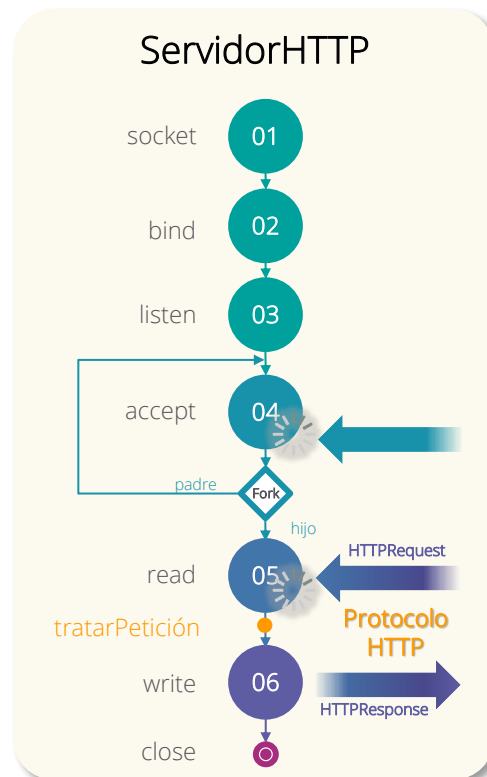
06

tratarPetición(request, response)

write(sdHijo, response)

close(sdHijo)

End



tratarPetición(cadena req, cadena res,)

Begin

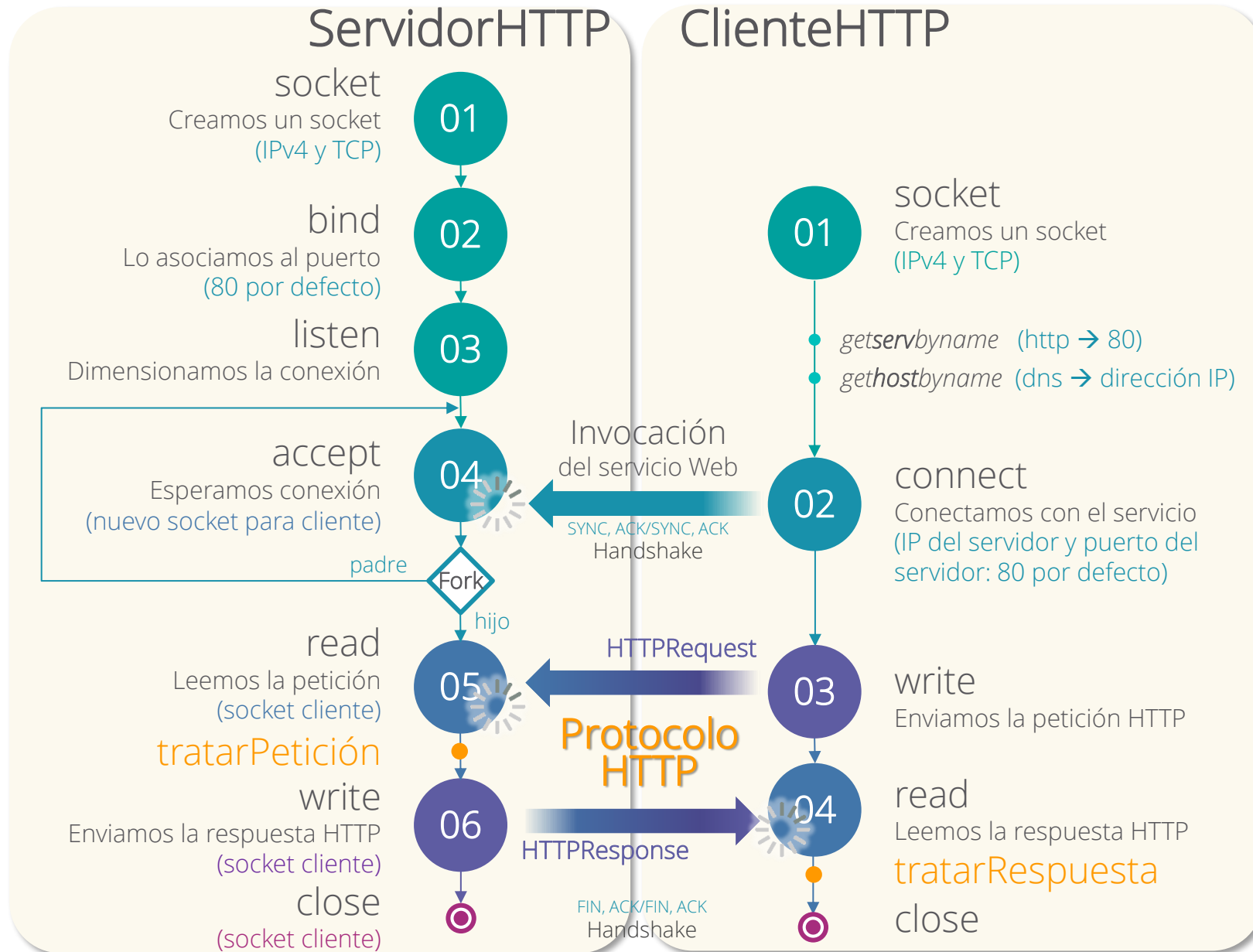
recurso = obtenerRecurso(req)

res = "200 OK HTTP/1.1<\n>" + headers + "<\n><\n>" + recurso

End

Versión en pseudo-código de un servidor Web

Mecanismos de comunicación



Paso de mensaje

Sockets

Características:

- Comunicación Directa → Control preciso sobre el flujo de datos.

Ventajas:

- Alto rendimiento
- Flexibilidad: Adecuado para sistemas personalizados

Desventajas:

- Complejidad en manejo de errores
- Complejo abordar proyectos con grandes volúmenes de conexiones

Aplicaciones Comunes:

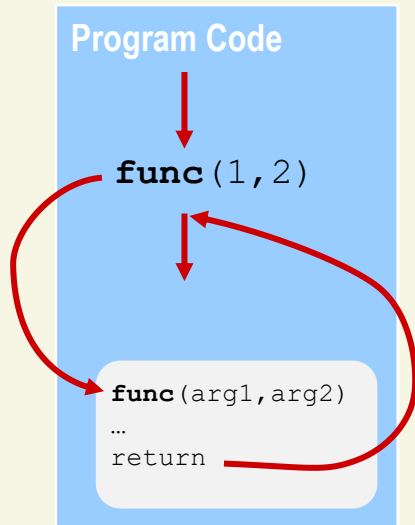
- Servidores Web (HTTP)
- Juegos en Red
- Streaming (video/audio con baja latencia)

SOCKETS

RPC (Remote Procedure Call)

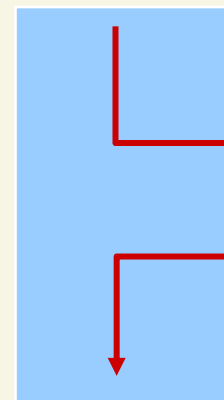
Características

RPC permite ejecutar **funciones** en un **servidor remoto** como si fueran locales.

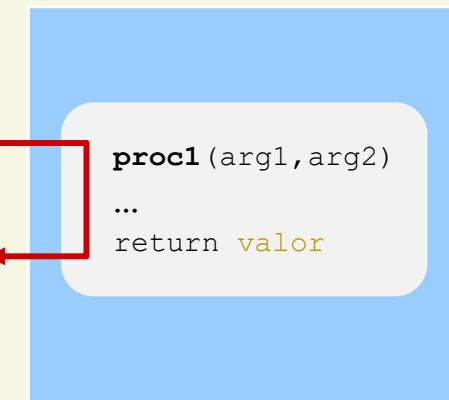


Llamada a una función o procedimiento local a un programa

Host A
Proceso 1



Host B
Proceso 2



*Llamada a un **procedimiento remoto** ubicado en un proceso diferente, en un host diferente*

RPC

SOCKETS

RPC (Remote Procedure Call)

Características

RPC permite ejecutar funciones en un servidor remoto como si fueran locales.

Evolución: Surgió como una mejora de la comunicación mediante sockets orientado a lenguajes procedimentales.

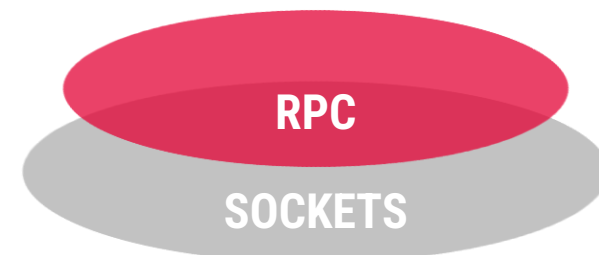
Objetivo: Simplificar la comunicación en red al abstraer detalles técnicos.

Características Principales:

Transparencia: oculta la localización del procedimiento.

Abstracción de Red: detalles de red invisibles para el desarrollador.

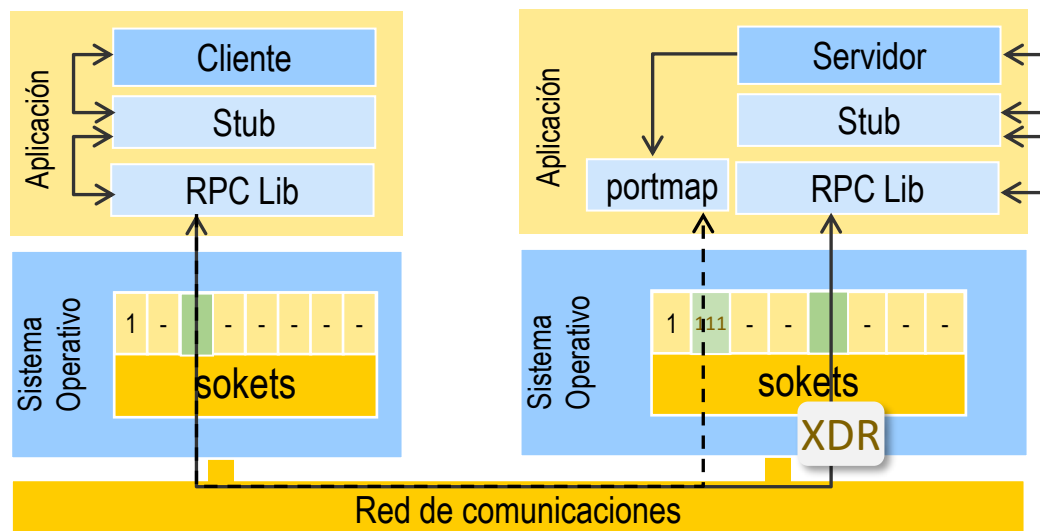
Simplicidad: facilita el desarrollo de sistemas distribuidos.



RPC (Remote Procedure Call)

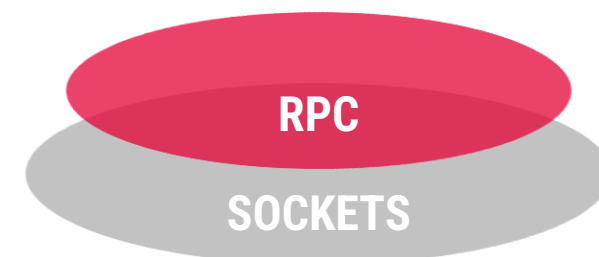
Características

Arquitectura de modelo de llamadas a procedimientos remotos



Componentes de RPC:

- **Cliente y Stub del Cliente:** Solicita y convierte las llamadas en solicitudes de red.
- **Biblioteca RPC:** Actúa como **Sistema de Transporte** transfiriendo las solicitudes y respuestas entre cliente y servidor.
- **Servidor:** Ejecuta la función y devuelve los resultados.
- **Stub del Servidor:** Deserializa la solicitud y la envía al servidor.
- **Portmap:** Servicio que registra y proporciona el **puerto** en el que estará escuchando el Servidor.
- **RPCGEN:** Compilador que genera el código a partir de una definición de interfaz (**IDL**)



RPC (Remote Procedure Call)

Ejemplo de RPC

Código de la aplicación

```
void Suma(int a, int b, int *c) ;
```

```
void main() {  
    int a=1, b=3, c;  
  
    Suma(a, b, &c);  
    printf("%d\n", c);  
}
```

```
void Suma(int a, int b, int *c) {  
    *c = a + b;  
}
```

RPC (Remote Procedure Call)

Ejemplo de RPC

Definición Interfaz

```
void Suma(int a, int b, int *c);
```

Código Cliente

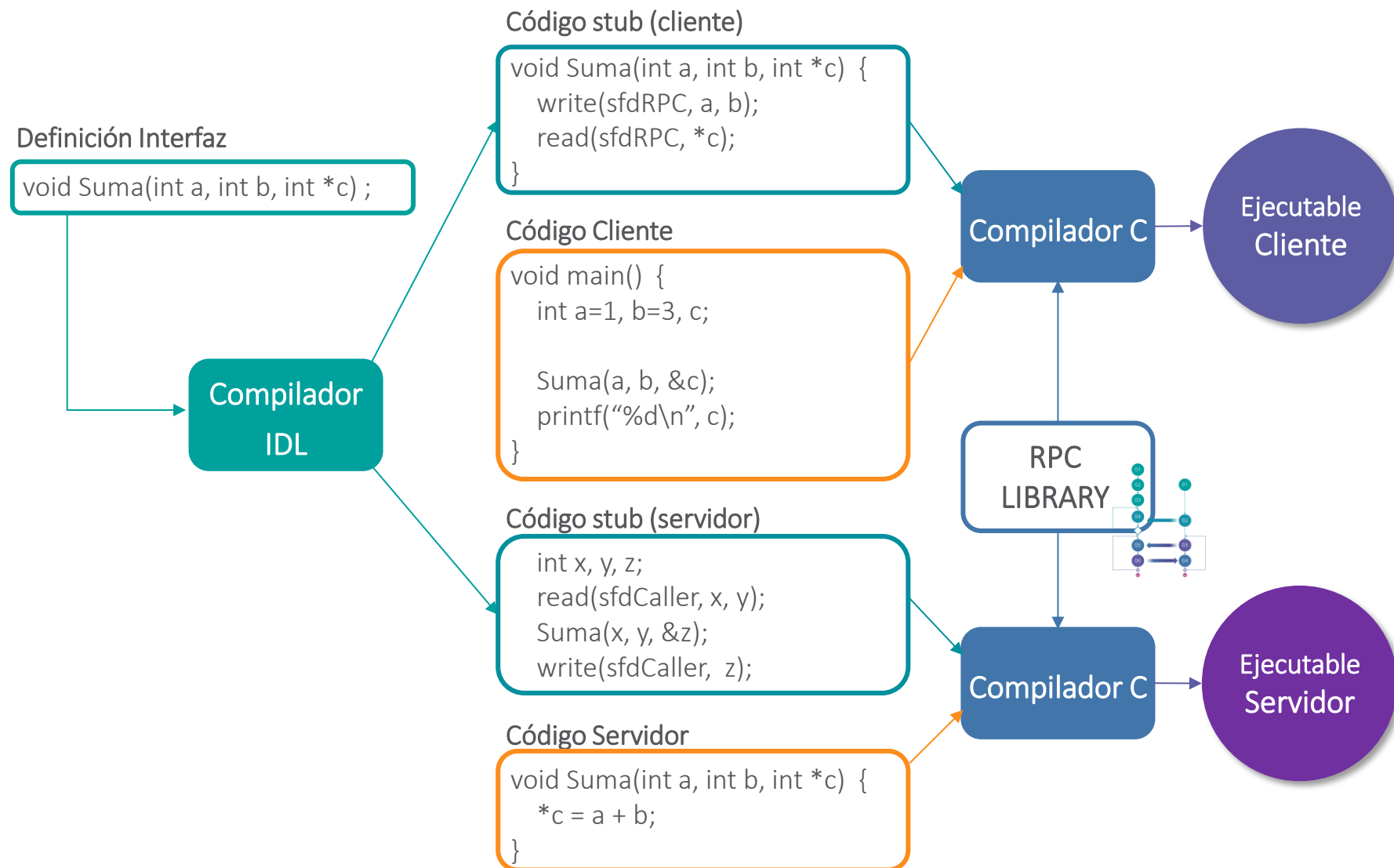
```
void main() {  
    int a=1, b=3, c;  
  
    Suma(a, b, &c);  
    printf("%d\n", c);  
}
```

Código Servidor

```
void Suma(int a, int b, int *c) {  
    *c = a + b;  
}
```

RPC (Remote Procedure Call)

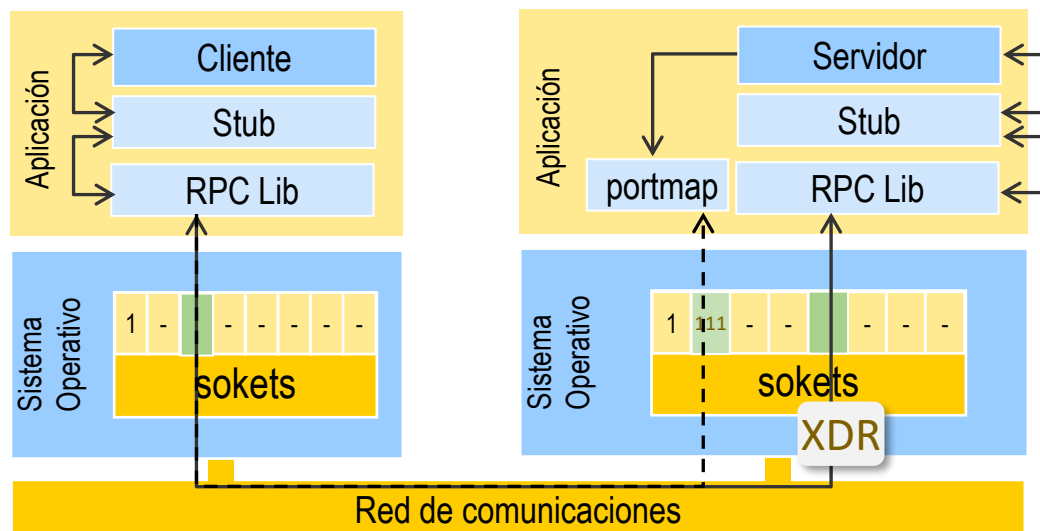
Ejemplo de RPC



RPC (Remote Procedure Call)

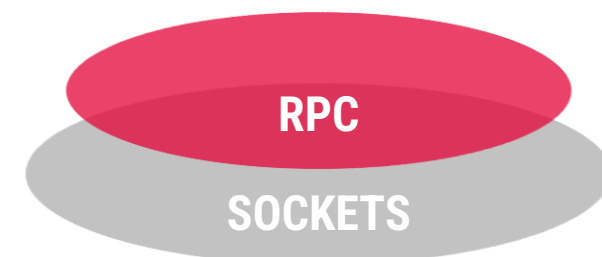
Características

Arquitectura de modelo de llamadas a procedimientos remotos



Componentes de RPC:

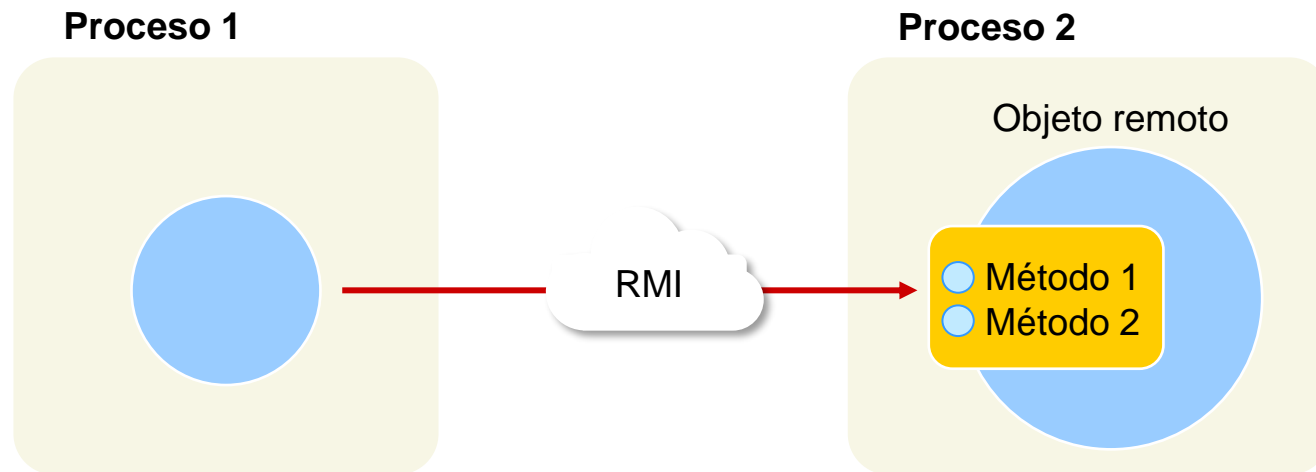
- **Cliente y Stub del Cliente:** Solicita y convierte las llamadas en solicitudes de red.
- **Biblioteca RPC:** Actúa como **Sistema de Transporte** transfiriendo las solicitudes y respuestas entre cliente y servidor.
- **Servidor:** Ejecuta la función y devuelve los resultados.
- **Stub del Servidor:** Deserializa la solicitud y la envía al servidor.
- **Portmap:** Servicio que registra y proporciona el **puerto** en el que estará escuchando el Servidor.
- **RPCGEN:** Compilador que genera el código a partir de una definición de interfaz (**IDL**)



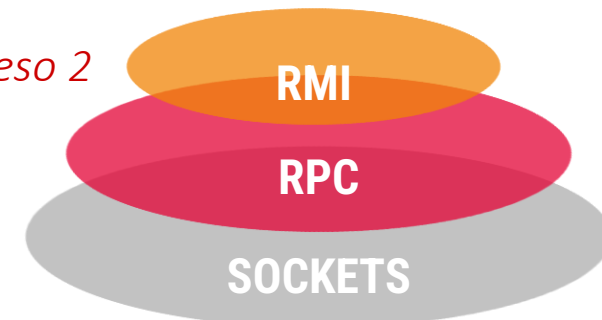
RMI (Remote Methods Invocation)

Características

RMI permite a una aplicación Java invocar métodos en objetos remotos como si fueran locales.



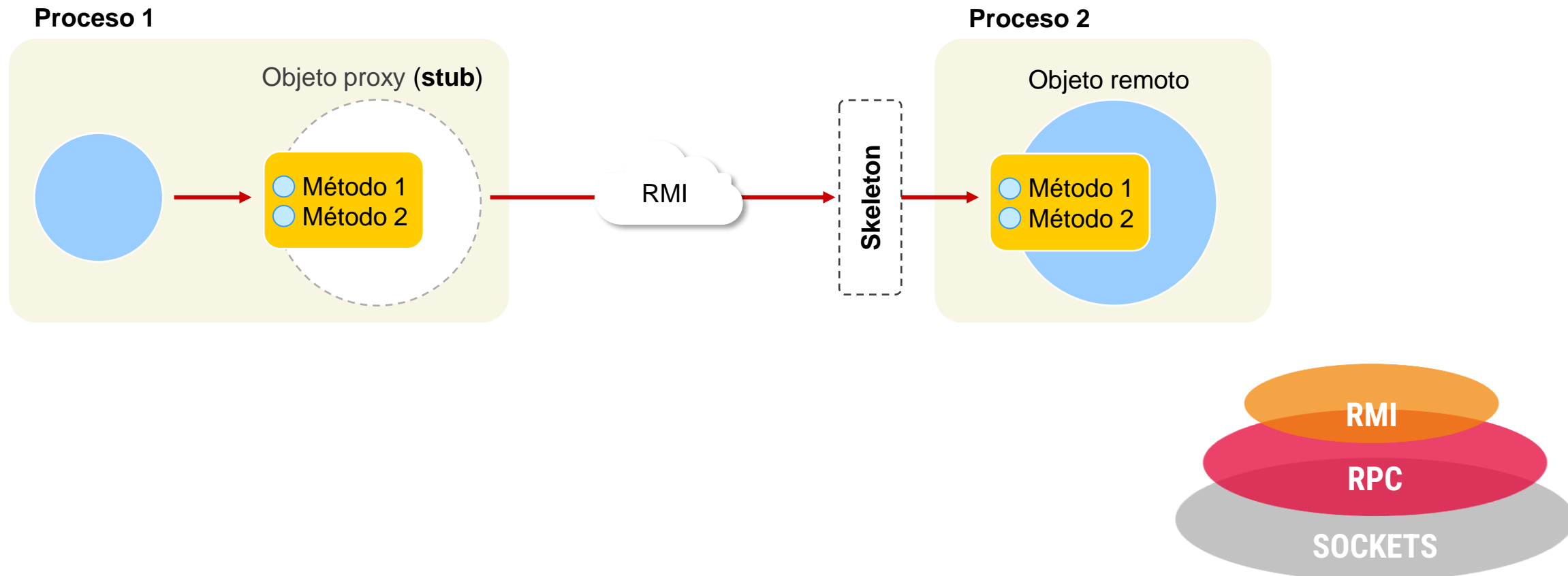
Invocación desde el proceso 1 a un método perteneciente a un objeto remoto ubicado en el proceso 2



RMI (Remote Methods Invocation)

Características

RMI permite a una aplicación Java invocar métodos en objetos remotos como si fueran locales.



RMI (Remote Methods Invocation)

Características

RMI permite a una aplicación Java invocar métodos en objetos remotos como si fueran locales.

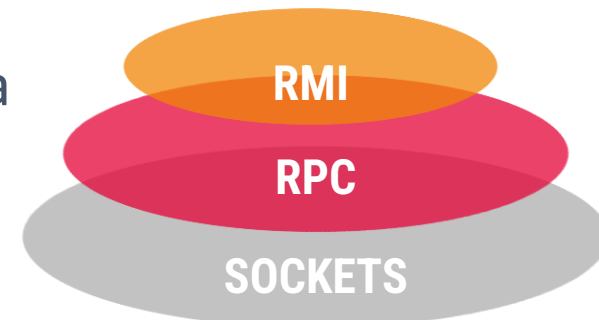
Basado en el paradigma orientado a objetos, ideal para Java.

Características

- Transparencia en la invocación
- Registro de objetos remotos
- Abstracción de la red

Novedades

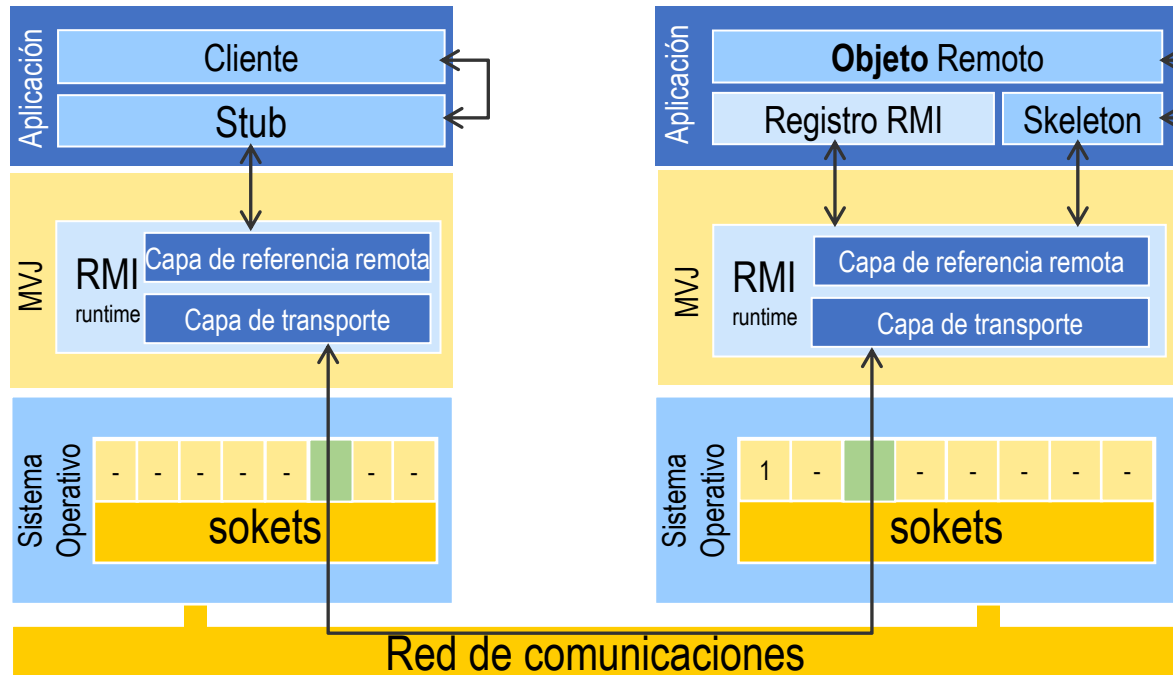
- Orientación a objetos
- Incorporación de servicios adicionales (Registro que sustituye a portmap)



RMI (Remote Methods Invocation)

Características

Arquitectura de modelo de invocación a métodos remotos



Componentes de RPC:

Cliente: solicita la ejecución de un método remoto.

Stub del cliente: actúa como un **proxy** local que convierte la llamada del método en una solicitud de red.

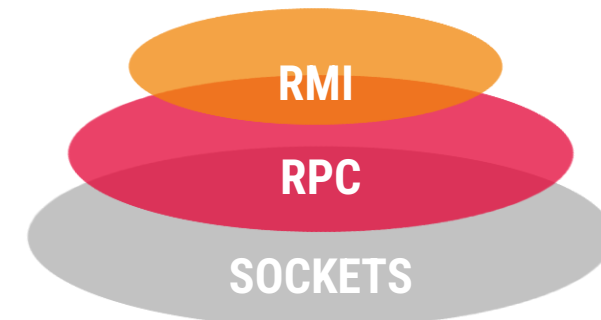
Biblioteca RMI: maneja la comunicación entre el cliente y el servidor, transfiriendo las solicitudes y respuestas de métodos remotos.

Servidor: implementa las interfaces remotas y ejecuta los métodos solicitados por el cliente, devolviendo los resultados.

Skeleton del servidor (obsoleto, sustituido por **stub**): deserializa la solicitud recibida y la envía al objeto servidor para su ejecución. Luego, serializa la respuesta para enviarla de vuelta al cliente.

Registro RMI (RMI Registry): Servicio que permite a los clientes localizar y obtener referencias a los objetos remotos registrados.

Compilador RMI (**rmic**): genera los stubs y skeletons necesarios a partir de las interfaces remotas definidas en el código.



RMI (Remote Methods Invocation)

Características

Ventajas

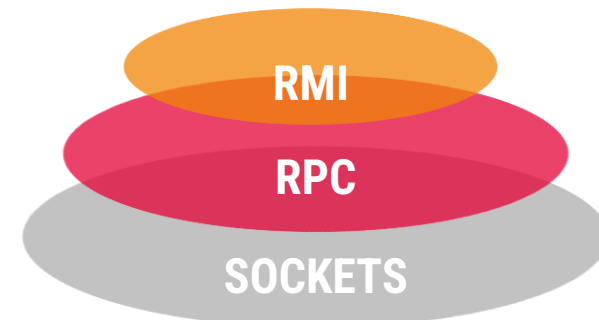
- Facilita la programación distribuida en Java
- Oculta la complejidad de red

Desventajas

- Dependencia de Java
- Sobrecarga de red por serialización

Casos de Uso

- Aplicaciones empresariales



ORB (Object Request Broker)

Características

Object Request Broker (ORB)

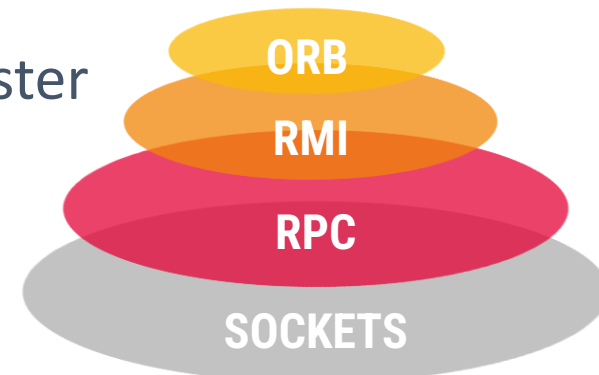
- Facilita la comunicación entre objetos distribuidos.
- Evolución de RMI para otros lenguajes e incorporación de más servicios.
- Ejemplo: **CORBA** (Common Object Request Broker Architecture).

Características

- Transparencia de localización y plataforma.
- Uso de IDL (Interface Definition Language) para definir interfaces.
- Serialización automática.

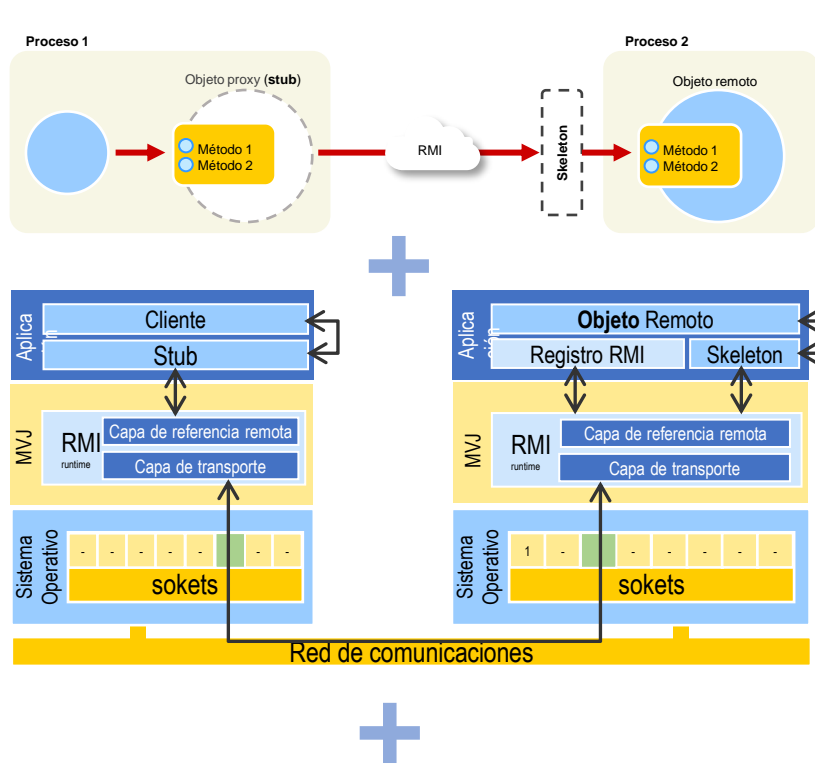
Componentes

- Cliente, Stub, ORB, Skeleton, Objeto remoto, Compilador IDL, Register
- Servicios adicionales: Name, events, transactions, security, ...



ORB (Object Request Broker)

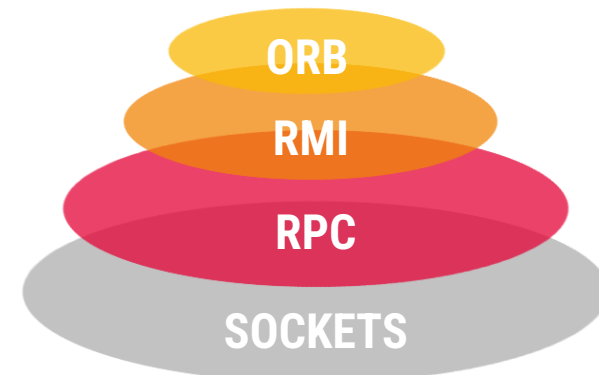
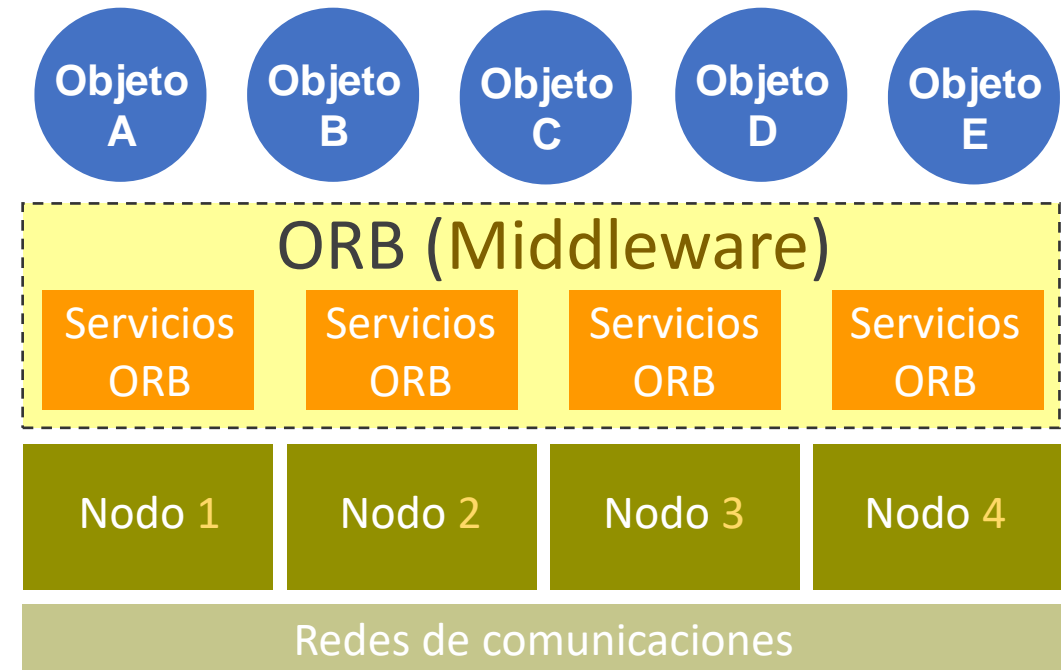
Características



*Multilenguaje + Otros servicios:
nombres, eventos, transacciones,
seguridad, ...*

Paradigma de objetos remotos basados en Invocación a Métodos Remotos

=



ORB (Object Request Broker)

Características

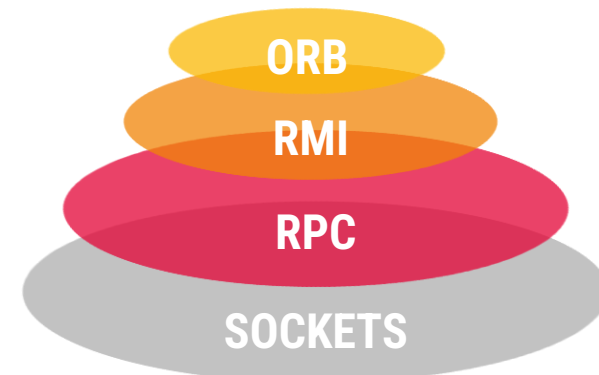
Ventajas

- Soporte multiplataforma: SO, lenguajes, hardware.
- Interoperabilidad en entornos heterogéneos.

Desventajas

- Complejidad en configuración.
- Latencia por serialización.

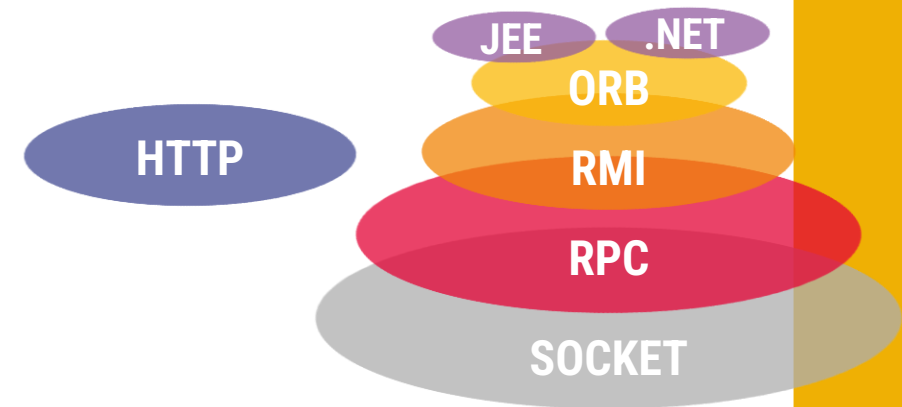
Base de concepto Middleware



Middleware

Características

- Capa de abstracción **compleja**
- **Reduce esfuerzos** del programador para la **comunicación** entre procesos
- Inconvenientes
 - Aumenta la **complejidad** y la **curva** de **aprendizaje** de los propios middlewares
 - Altos consumidores de **recursos**



Mecanismos de comunicación

HTTP

✓ Modelo HTTP Básicos

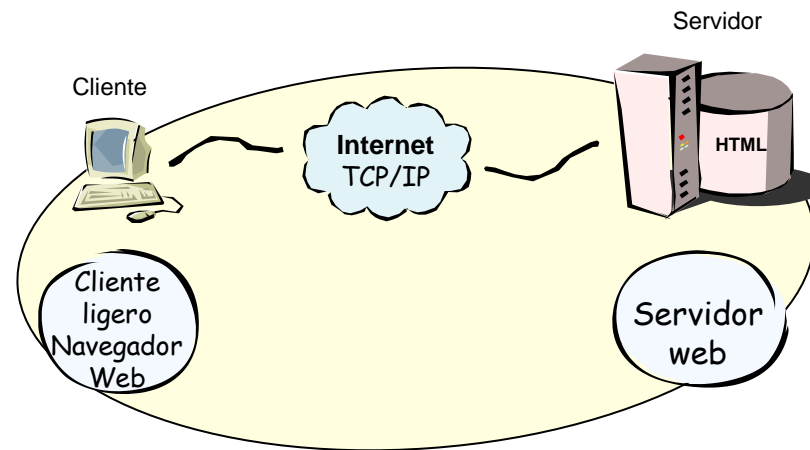
Revisión de tecnologías Web

Contenidos

Modelo HTTP Básico

Modelo HTTP Básico

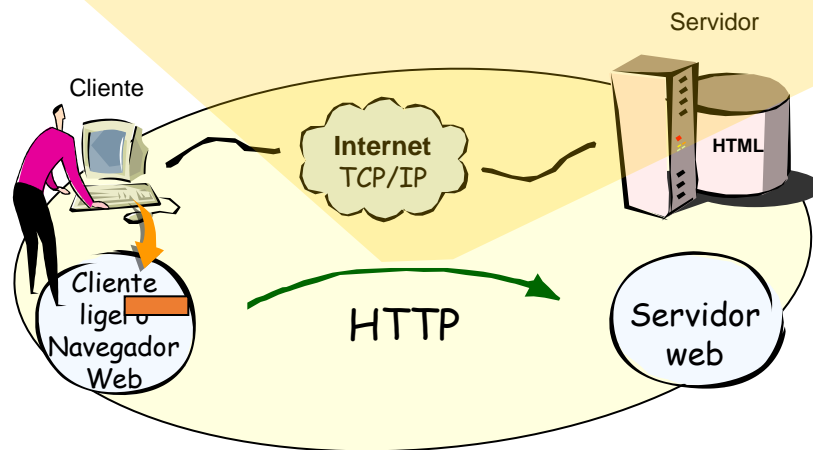
servicio HTTP básico



Modelo HTTP Básico

servicio HTTP básico

Método	Recurso	Versión	Cabeceras
GET	/index.html	HTTP/1.0	



Tecnologías Web

Servicio **HTTP** básico : **HTTPRequest**

*Formato de una **petición HTTP** (HTTP Request)*

Línea	Método direcciónRecurso versión Protocolo
Cabecera	Etiqueta: valor Etiqueta: valor . . . Etiqueta: valor <línea en blanco>
Cuerpo	Información adicional en formato texto (opcional)

*Principales **métodos HTTP***

- GET** Solicitar contenido
- HEAD** Solicitar únicamente la cabecera
- POST** Enviar datos al servidor (Crea nuevo contenido)
- PUT** Actualizar contenido existente
- DELETE** Eliminar contenido del servidor

Métodos HTTP: **GET**, **POST**, **PUT**, **DELETE**, HEAD, ...

Versión del Protocolo: HTTP/1.0, HTTP/1.1, ...

Etiquetas típicas de la **Cabecera**:

- Content-Length** indica el tamaño del cuerpo o **body**
- Content-Type** indica el tipo MIME del contenido del **body**
- Authorization** se emplea para enviar credenciales de acceso (como un token JWT)
- Connection** indica si la conexión TCP debe mantenerse abierta para otra petición HTTP

Tecnologías Web

Servicio HTTP básico : HTTPRequest

Cabecera	<code>POST /cgi/miAplicacion.cgi HTTP/1.0</code>
	<code>Accept: */*</code> <code>Connection: Keep-Alive</code> <code>User-Agent: Generic</code> <code>Content-type: application/json</code> <code>Content-length: 39</code> <code><línea en blanco></code>
Cuerpo	<code>{"nombre":"Iren","email":"ilorenzo@dtic.ua.es"}</code>

Ejemplo 1: petición HTTP tipo POST

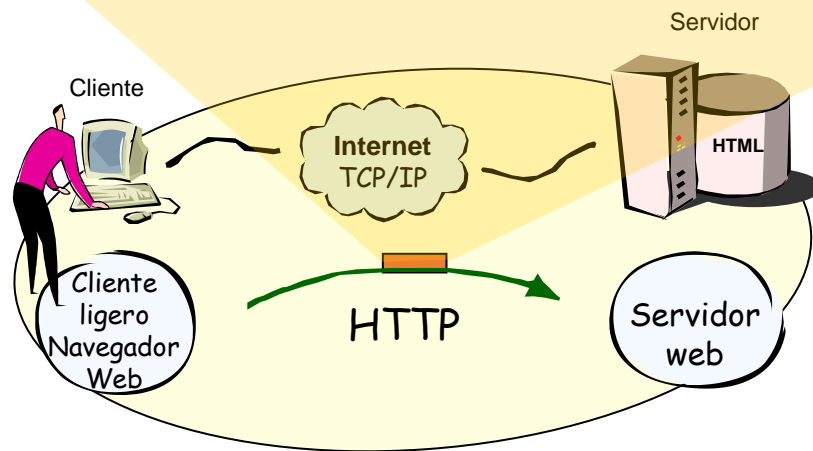
Cabecera	<code>GET /~ilorenzo/sod/index.html HTTP/1.0</code>
	<code>Accept: */*</code> <code>Connection: Keep-Alive</code> <code>User-Agent: Generic</code> <code><línea en blanco></code>
Cuerpo	<code><vacío></code>

Ejemplo 2: petición HTTP tipo GET

Modelo HTTP Básico

servicio HTTP básico

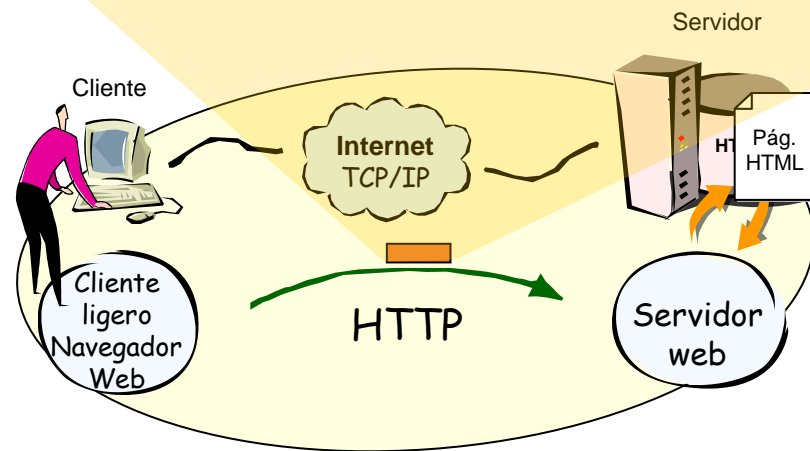
Método	Recurso	Versión	Cabeceras
GET	/index.html	HTTP/1.0	



Modelo HTTP Básico

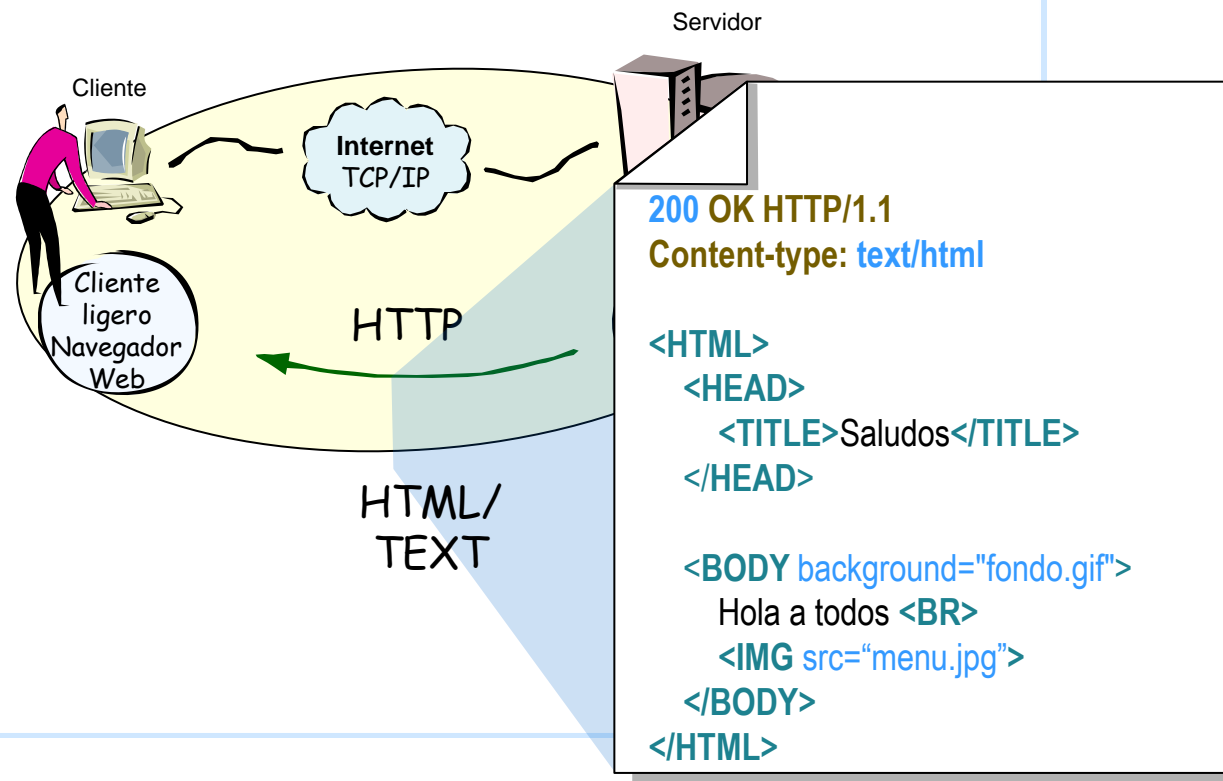
servicio HTTP básico

Método	Recurso	Versión	Cabeceras
GET	/index.html	HTTP/1.0	



Modelo HTTP Básico

servicio HTTP básico



Tecnologías Web

Servicio **HTTP** básico : **HTTPResponse**

Estado	HTTP/1.1 CódigoEstadoRespuesta MensajeRespuesta
Cabeceras	Etiqueta: valor Etiqueta: valor . . . Etiqueta: valor <línea en blanco que indica el final de las cabeceras>
Cuerpo	Información adicional en formato texto (el tipo se indica en la cabecera content-type) El body o cuerpo es (opcional)

*Formato de una **respuesta HTTP** (HTTP Response)*

Según el **código de estado**, las respuestas se agrupan en cinco clases:

- [**100** – 199] Respuestas informativas
- [**200** – 299] Respuestas satisfactorias
- [**300** – 399] Redirecciones
- [**400** – 499] Errores de los clientes
- [**500** – 599] Errores de los servidores

Códigos de estado de respuesta HTTP

Tecnologías Web

Servicio HTTP básico : HTTPRespon

Códigos de estado de respuesta HTTP

1xx Mensajes de información

- 100 Continua
- 101 Cambio de protocolo

2xx Operación exitosa

- 200 Ok
- 201 Creado
- 202 Aceptado
- 203 Información no oficial
- 204 Sin Contenido
- 205 Contenido para reset
- 206 Contenido parcial

3xx Redirección hacia otro URL

- 300 Múltiples posibilidades
- 301 Mudado permanentemente
- 302 Encontrado
- 303 Véa otros
- 304 No modificado
- 305 Utilice un proxy
- 307 REdirección temporal

4xx Error por parte del cliente

- 400 Solicitud incorrecta
- 401 No autorizado
- 402 Pago requerido
- 403 Prohibido
- 404 No encontrado
- 405 Método no permitido
- 406 No aceptable
- 407 Proxy requerido
- 408 Tiempo de espera agotado
- 409 Conflicto
- 410 No mpas disponible
- 411 Requiere logitud
- 412 Falló precondition
- 413 Entidad de solicitud demasiado larga
- 414 URI de solicitud demasiado largo
- 415 Tipo de medio no soportado
- 416 Rango solicitado no disponible
- 417 Falló expectativa

5xx Error por parte del servidor

- 500 Error interno
- 501 No implementado
- 502 Pasarela incorrecta
- 503 Servicio no disponible
- 504 Tiempo de espera de la pasarela agotado
- 505 Versión de HTTP no soportada

Tecnologías Web

Servicio HTTP básico : HTTPRespon

Estado	HTTP/1.1 200 OK
Cabeceras	<div><div>Date: Thu, 04 Nov 2004 17:59:15 GMT</div><div>Server: Apache/2.0.40 (Red Hat Linux)</div><div>Last-Modified: Wed, 03 Nov 2004 13:01:02 GMT</div><div>ETag: "378029-902-3bd83f80"</div><div>Accept-Ranges: bytes</div><div>Content-Length: 256</div><div>Content-Type: text/html</div></div>
Cuerpo	<div><html></div> <div><head></div> <div><title>Tecnología Informática y Computación</title></div> <div></head></div> <div>
</div> <div><body background="fondo.gif"></div> <div>Hola a todas y a todos
</div> <div></div> <div></body></div> <div></html></div>

Ejemplo: respuesta HTTP

Tecnologías Web

Servicio HTTP básico : HTTPRespon

Content-Type: text/html

HTML (HyperText Markup Language)

⌚ Documento de texto

⌚ Etiquetas de formato

▪ **<etiqueta [arg1 arg2 ...]> ... [</etiqueta>]**

⌚ Referencias cruzadas

▪ ** texto descriptivo **

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
<title>Mi Página Web</title>
</head>
<body>
  <h1>Bienvenidos a mi página web</h1>
  <p>Esta es una página web de ejemplo creada con HTML.</p>
</body>
</html>
```

Tecnologías Web

Servicio HTTP básico : HTTPRespond

Estado	HTTP/1.1 200 OK
Cabeceras	<div><div>Date: Thu, 04 Nov 2004 17:59:15 GMT</div><div>Server: Apache/2.0.40 (Red Hat Linux)</div><div>Last-Modified: Wed, 03 Nov 2004 13:01:02 GMT</div><div>ETag: "378029-902-3bd83f80"</div><div>Accept-Ranges: bytes</div><div>Content-Type: application/json</div><div>Content-Length: 138</div><div>Connection: close</div><div><línea en blanco></div></div>
Cuerpo	<div>{ Nombre: "José", Apellido1: "Pérez", Apellido2: "Jiménez", Edad: "22", Direccion: "C/ General Mola, 7", CP: "04302", EMail: "jperez@gmail.com", }</div>

Ejemplo: respuesta HTTP enviando un objeto JSON

Tecnologías Web

Servicio HTTP básico : MIME

Content-Type: **MIME** (Multipurpose Internet Mail Extension)

Texto sin formato: text/plain

- **HTML:** text/html
- **JSON:** application/json
- **XML:** application/xml
- **JavaScript:** application/javascript
- **CSS:** text/css

Imágenes:

- image/jpeg
- image/png
- image/gif
- image/webp
- image/svg+xml

Audio:

- audio/mpeg
- audio/ogg
- audio/wav

Archivos:

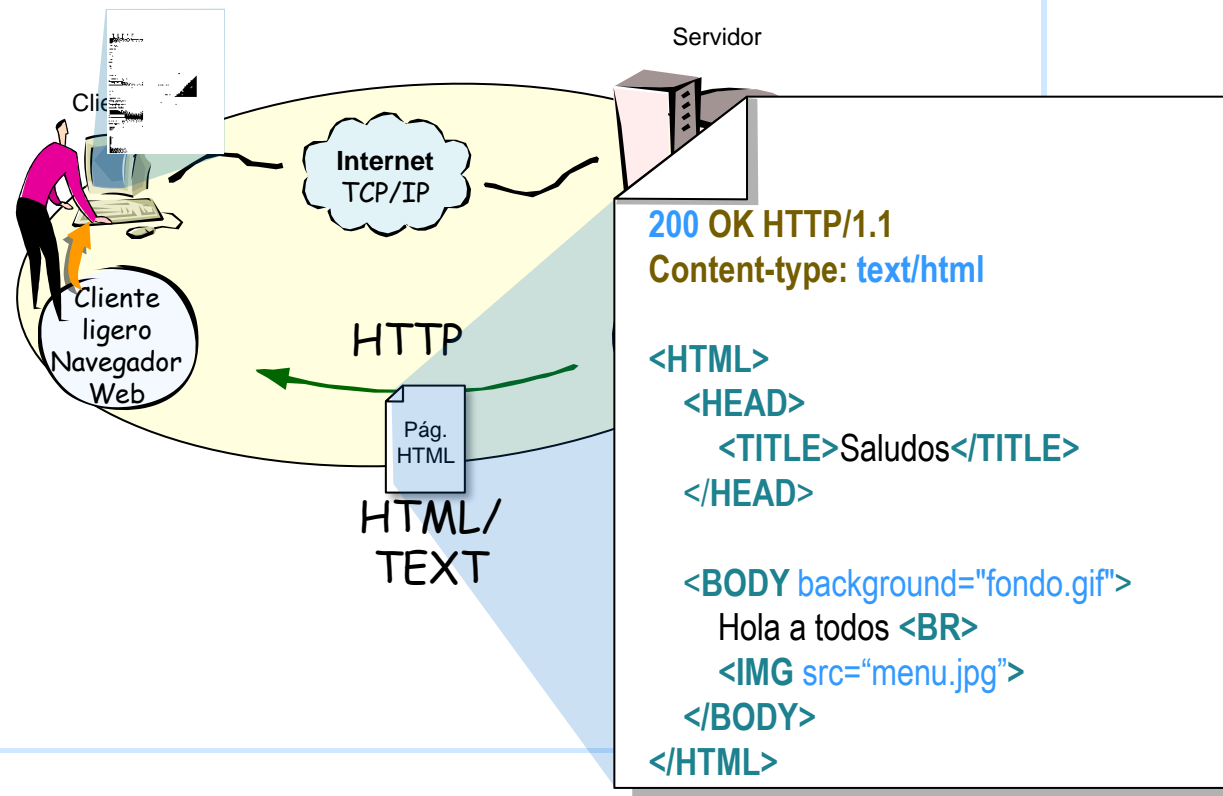
- application/pdf
- application/zip
- application/vnd.ms-excel (para archivos Excel)
- application/msword (para archivos Word)

Video:

- video/mp4
- video/webm
- video/ogg

Modelo HTTP Básico

servicio HTTP básico

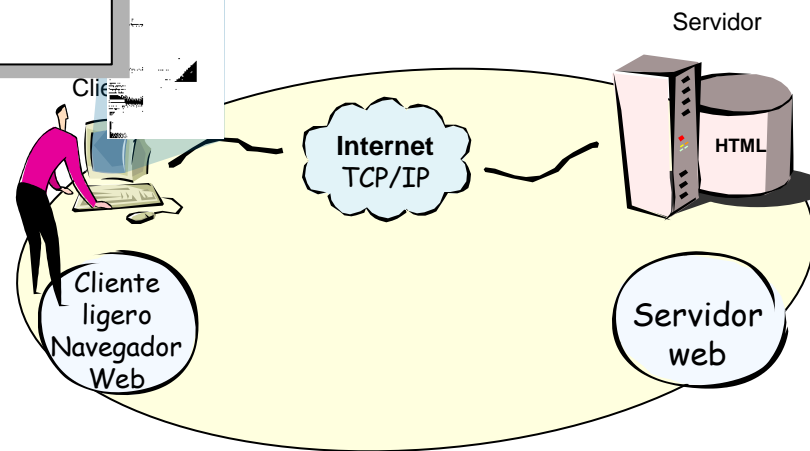


Modelo HTTP Básico

200 OK HTTP/1.1
Content-type: text/html

```
<HTML>
  <HEAD>
    <TITLE>Saludos</TITLE>
  </HEAD>

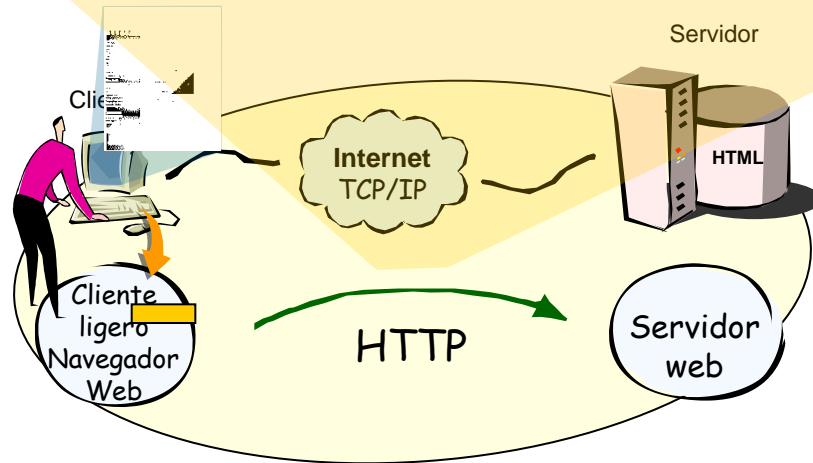
  <BODY background="fondo.gif">
    Hola a todos <BR>
    <IMG src="menu.jpg">
  </BODY>
</HTML>
```



Modelo HTTP Básico

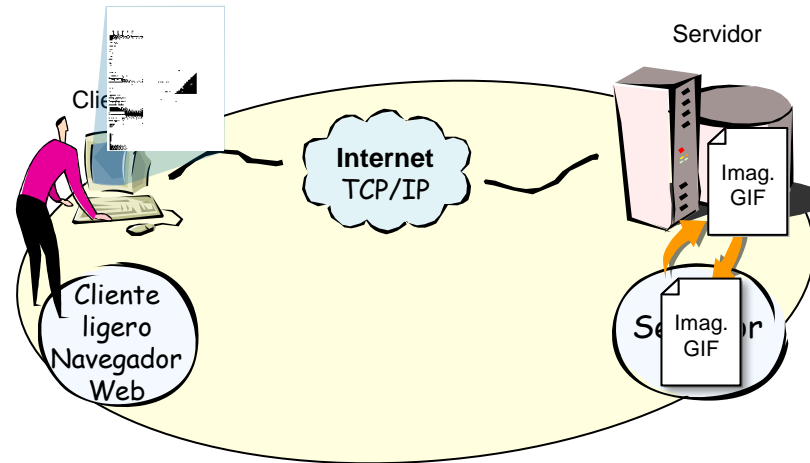
servicio HTTP básico

Método	Recurso	Versión	Cabeceras
GET	/fondo.gif	HTTP/1.0	



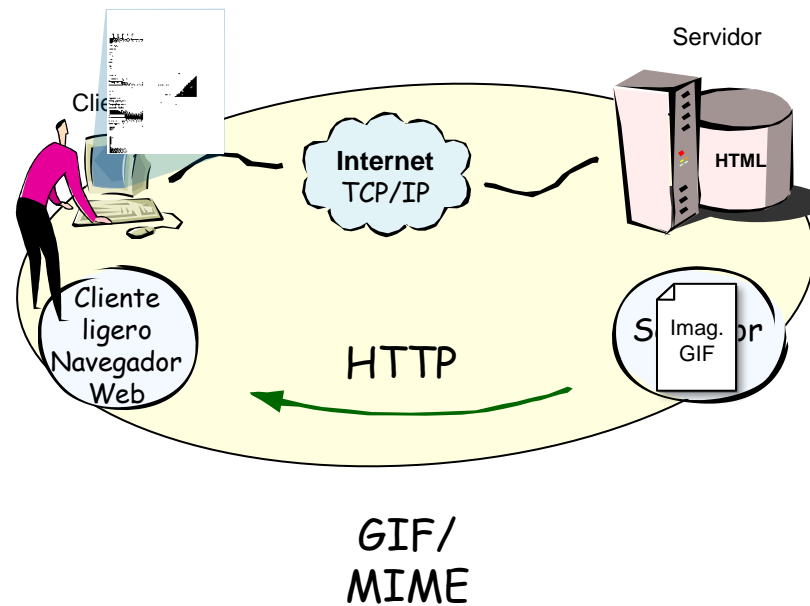
Modelo HTTP Básico

servicio HTTP básico



Modelo HTTP Básico

servicio HTTP básico



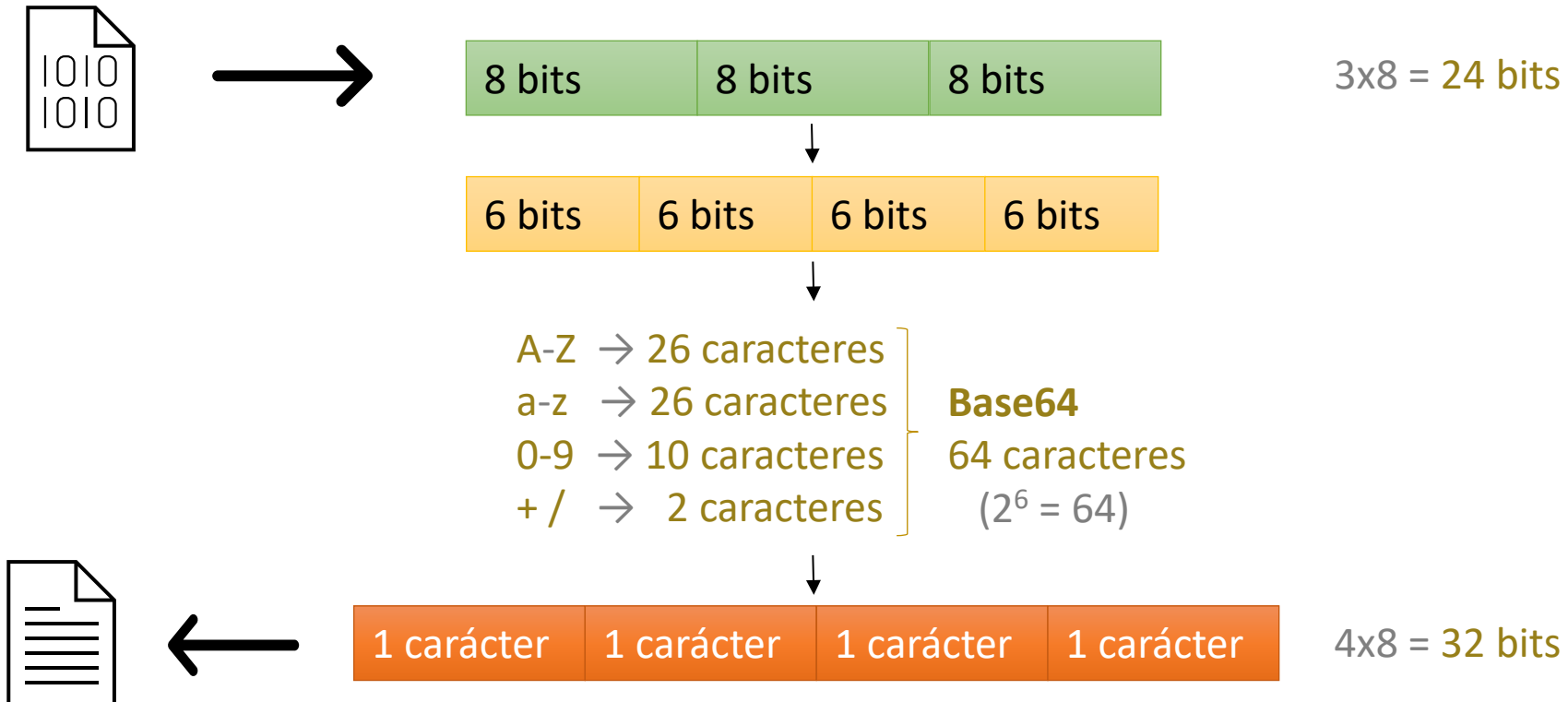
Revisión de tecnologías Web

[illegible]

05 Tecnologías Web

Modelos Básicos → Servicio HTTP básico

MIME (Multipurpose Internet Mail Extension)

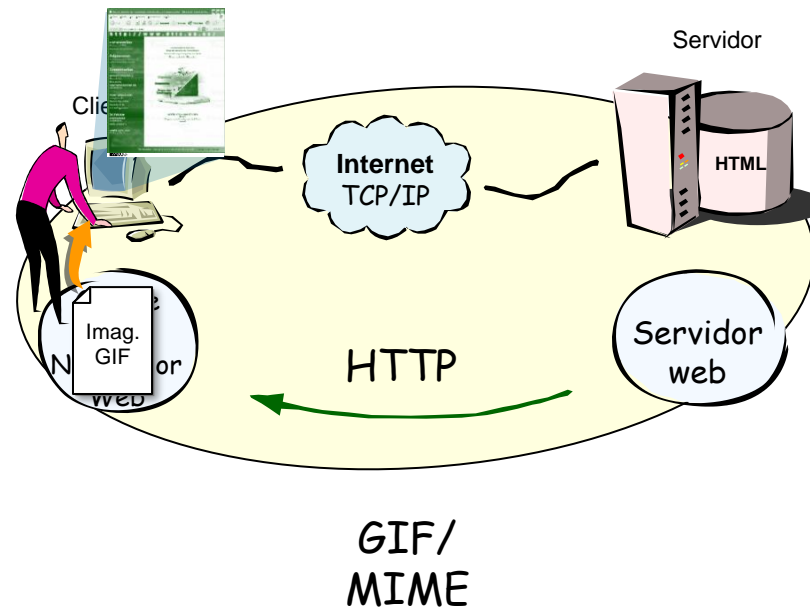


Revisión de tecnologías Web

[illegible]

Modelo HTTP Básico

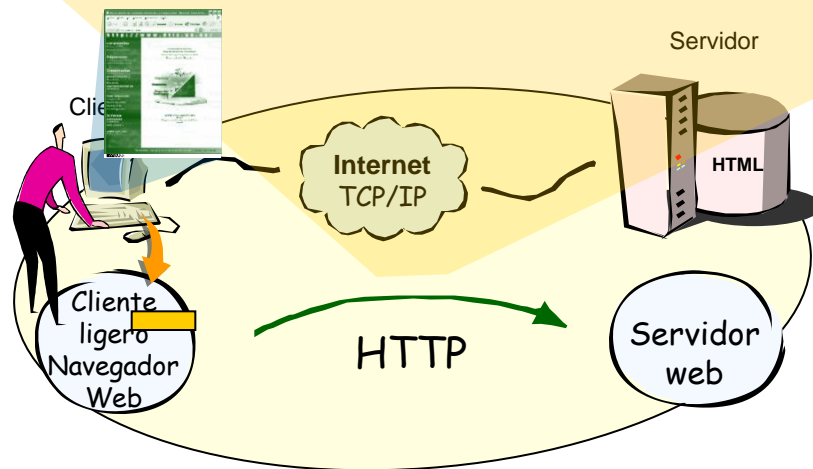
servicio HTTP básico



Modelo HTTP Básico

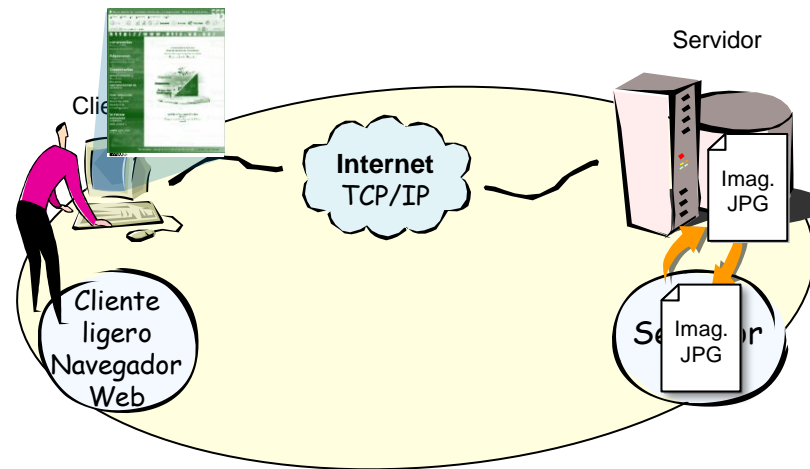
servicio HTTP básico

Método	Recurso	Versión	Cabeceras
GET	/menu.jpg	HTTP/1.0	



Modelo HTTP Básico

servicio HTTP básico



Revisión de tecnologías Web

The diagram illustrates the HTTP request and response cycle between a client and a server. On the left, a **Cliente ligero Navegador Web** (lightweight web browser client) is shown with a person at a computer. On the right, the **Servidor** (server) is shown with a rack of hardware. The **Internet TCP/IP** cloud connects them. A green arrow labeled **HTTP** points from the client to the server, and a blue arrow labeled **JPG/MIME** points from the server back to the client. A document icon labeled **Imag. JPG** is shown on the client side, representing the received image. The server side shows a **MIME** header and a large block of base64-encoded data representing the image file.

Client: Cliente ligero Navegador Web

Internet: Internet TCP/IP

Server: Servidor

HTTP Request: 200 OK HTTP/1.0
Content-Type: image/jpeg; name="menu.jpg"
Content-Transfer-Encoding: base64

Response Data (Base64):

```

/9j/4AAQSkZJRgABAQEAYABgAAD/2wBDAAgGBgcBQgQBQgHbWcJCQgKBQNDASIL
DbkSEwUHRofHh0aHBwgJC4nICsIxwKdcpLDAxNDQ0Hyc5PTgyPC4zNDL/
2wBDAQkJCQwLDBgNDRgyIRwhMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIy
MjIyMjIyMjIyMjIyMjL/wAARCAAMAAwDASIAAhEBAxEB/8QA
HwAAQUBAQEBAQEAAAAAAAAAAAEACAwQFBgcICQoL/8QAtRAAAgEDAwIEAwUF
BAQAAAF9AQIDAAQRBRIhMUEGE1FhByJxFDKBkaEII0KxwRVS0fAkM2JyggkK
FhcYGRolJicoKSo0NTY3ODk6Q0RFRkdISUpTVFVWV1h2WmNkZWZnaGlqc3R1
dnd4eXQdHIWGH4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW27i5uSLdXMxG
x8jJytLTlNXWl9jZ2Uhi4+Tl5ufo6erx8vP09fb3+Pn6/8QAHwEAAwEBAQEB
AQEBAQAAAAAAAAAECAwQFBgcICQoL/8QAtREAAgECBAQDBAcFBAQAAQJ3AAEC
AxEEBSEXbHJBUCdhcRMiMoEIFEKRobHBCSMzUvAVYnLRChYkNOEl8RcYGRom
JygpKjU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6goOE
hYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsxO0tba3uLm6wsPExcBHyMnK0tPU
1dbX2Nna4uPk5ebn6Onq8vP09fb3+Pn6/9oADAMBAAIRAxEAPwDk27c7rM/m
+QhjL28e4SFvnC7WIxggbjkgdOnzCoWcKeI1QquCRnLHJOTk9eccYHA75Jd
bTy2d1FcwPtmicSRtgHawOrwfcVFge/514D9m3yyk7dNO9+1/Tdvr5H7RCM1
r+v7H//2Q==

```

Header: MIME

Response Data: 200 OK HTTP/1.0
Content-Type: image/jpeg; name="menu.jpg"
Content-Transfer-Encoding: base64

Response Data (Base64):

```

/9j/4AAQSkZJRgABAQEAYABgAAD/2wBDAAgGBgcBQgQBQgHbWcJCQgKBQNDASIL
DbkSEwUHRofHh0aHBwgJC4nICsIxwKdcpLDAxNDQ0Hyc5PTgyPC4zNDL/
2wBDAQkJCQwLDBgNDRgyIRwhMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIy
MjIyMjIyMjIyMjIyMjL/wAARCAAMAAwDASIAAhEBAxEB/8QA
HwAAQUBAQEBAQEAAAAAAAAAAAEACAwQFBgcICQoL/8QAtRAAAgEDAwIEAwUF
BAQAAAF9AQIDAAQRBRIhMUEGE1FhByJxFDKBkaEII0KxwRVS0fAkM2JyggkK
FhcYGRolJicoKSo0NTY3ODk6Q0RFRkdISUpTVFVWV1h2WmNkZWZnaGlqc3R1
dnd4eXQdHIWGH4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW27i5uSLdXMxG
x8jJytLTlNXWl9jZ2Uhi4+Tl5ufo6erx8vP09fb3+Pn6/8QAHwEAAwEBAQEB
AQEBAQAAAAAAAAAECAwQFBgcICQoL/8QAtREAAgECBAQDBAcFBAQAAQJ3AAEC
AxEEBSEXbHJBUCdhcRMiMoEIFEKRobHBCSMzUvAVYnLRChYkNOEl8RcYGRom
JygpKjU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6goOE
hYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsxO0tba3uLm6wsPExcBHyMnK0tPU
1dbX2Nna4uPk5ebn6Onq8vP09fb3+Pn6/9oADAMBAAIRAxEAPwDk27c7rM/m
+QhjL28e4SFvnC7WIxggbjkgdOnzCoWcKeI1QquCRnLHJOTk9eccYHA75Jd
bTy2d1FcwPtmicSRtgHawOrwfcVFge/514D9m3yyk7dNO9+1/Tdvr5H7RCM1
r+v7H//2Q==

```

Header: MIME

Response Data: 200 OK HTTP/1.0
Content-Type: image/jpeg; name="menu.jpg"
Content-Transfer-Encoding: base64

Response Data (Base64):

```

/9j/4AAQSkZJRgABAQEAYABgAAD/2wBDAAgGBgcBQgQBQgHbWcJCQgKBQNDASIL
DbkSEwUHRofHh0aHBwgJC4nICsIxwKdcpLDAxNDQ0Hyc5PTgyPC4zNDL/
2wBDAQkJCQwLDBgNDRgyIRwhMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIy
MjIyMjIyMjIyMjIyMjL/wAARCAAMAAwDASIAAhEBAxEB/8QA
HwAAQUBAQEBAQEAAAAAAAAAAAEACAwQFBgcICQoL/8QAtRAAAgEDAwIEAwUF
BAQAAAF9AQIDAAQRBRIhMUEGE1FhByJxFDKBkaEII0KxwRVS0fAkM2JyggkK
FhcYGRolJicoKSo0NTY3ODk6Q0RFRkdISUpTVFVWV1h2WmNkZWZnaGlqc3R1
dnd4eXQdHIWGH4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW27i5uSLdXMxG
x8jJytLTlNXWl9jZ2Uhi4+Tl5ufo6erx8vP09fb3+Pn6/8QAHwEAAwEBAQEB
AQEBAQAAAAAAAAAECAwQFBgcICQoL/8QAtREAAgECBAQDBAcFBAQAAQJ3AAEC
AxEEBSEXbHJBUCdhcRMiMoEIFEKRobHBCSMzUvAVYnLRChYkNOEl8RcYGRom
JygpKjU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6goOE
hYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsxO0tba3uLm6wsPExcBHyMnK0tPU
1dbX2Nna4uPk5ebn6Onq8vP09fb3+Pn6/9oADAMBAAIRAxEAPwDk27c7rM/m
+QhjL28e4SFvnC7WIxggbjkgdOnzCoWcKeI1QquCRnLHJOTk9eccYHA75Jd
bTy2d1FcwPtmicSRtgHawOrwfcVFge/514D9m3yyk7dNO9+1/Tdvr5H7RCM1
r+v7H//2Q==

```

Header: MIME

Response Data: 200 OK HTTP/1.0
Content-Type: image/jpeg; name="menu.jpg"
Content-Transfer-Encoding: base64

Response Data (Base64):

```

/9j/4AAQSkZJRgABAQEAYABgAAD/2wBDAAgGBgcBQgQBQgHbWcJCQgKBQNDASIL
DbkSEwUHRofHh0aHBwgJC4nICsIxwKdcpLDAxNDQ0Hyc5PTgyPC4zNDL/
2wBDAQkJCQwLDBgNDRgyIRwhMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIy
MjIyMjIyMjIyMjIyMjL/wAARCAAMAAwDASIAAhEBAxEB/8QA
HwAAQUBAQEBAQEAAAAAAAAAAAEACAwQFBgcICQoL/8QAtRAAAgEDAwIEAwUF
BAQAAAF9AQIDAAQRBRIhMUEGE1FhByJxFDKBkaEII0KxwRVS0fAkM2JyggkK
FhcYGRolJicoKSo0NTY3ODk6Q0RFRkdISUpTVFVWV1h2WmNkZWZnaGlqc3R1
dnd4eXQdHIWGH4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW27i5uSLdXMxG
x8jJytLTlNXWl9jZ2Uhi4+Tl5ufo6erx8vP09fb3+Pn6/8QAHwEAAwEBAQEB
AQEBAQAAAAAAAAAECAwQFBgcICQoL/8QAtREAAgECBAQDBAcFBAQAAQJ3AAEC
AxEEBSEXbHJBUCdhcRMiMoEIFEKRobHBCSMzUvAVYnLRChYkNOEl8RcYGRom
JygpKjU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6goOE
hYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsxO0tba3uLm6wsPExcBHyMnK0tPU
1dbX2Nna4uPk5ebn6Onq8vP09fb3+Pn6/9oADAMBAAIRAxEAPwDk27c7rM/m
+QhjL28e4SFvnC7WIxggbjkgdOnzCoWcKeI1QquCRnLHJOTk9eccYHA75Jd
bTy2d1FcwPtmicSRtgHawOrwfcVFge/514D9m3yyk7dNO9+1/Tdvr5H7RCM1
r+v7H//2Q==

```

Header: MIME

Response Data: 200 OK HTTP/1.0
Content-Type: image/jpeg; name="menu.jpg"
Content-Transfer-Encoding: base64

Response Data (Base64):

```

/9j/4AAQSkZJRgABAQEAYABgAAD/2wBDAAgGBgcBQgQBQgHbWcJCQgKBQNDASIL
DbkSEwUHRofHh0aHBwgJC4nICsIxwKdcpLDAxNDQ0Hyc5PTgyPC4zNDL/
2wBDAQkJCQwLDBgNDRgyIRwhMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIy
MjIyMjIyMjIyMjIyMjL/wAARCAAMAAwDASIAAhEBAxEB/8QA
HwAAQUBAQEBAQEAAAAAAAAAAAEACAwQFBgcICQoL/8QAtRAAAgEDAwIEAwUF
BAQAAAF9AQIDAAQRBRIhMUEGE1FhByJxFDKBkaEII0KxwRVS0fAkM2JyggkK
FhcYGRolJicoKSo0NTY3ODk6Q0RFRkdISUpTVFVWV1h2WmNkZWZnaGlqc3R1
dnd4eXQdHIWGH4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW27i5uSLdXMxG
x8jJytLTlNXWl9jZ2Uhi4+Tl5ufo6erx8vP09fb3+Pn6/8QAHwEAAwEBAQEB
AQEBAQAAAAAAAAAECAwQFBgcICQoL/8QAtREAAgECBAQDBAcFBAQAAQJ3AAEC
AxEEBSEXbHJBUCdhcRMiMoEIFEKRobHBCSMzUvAVYnLRChYkNOEl8RcYGRom
JygpKjU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6goOE
hYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsxO0tba3uLm6wsPExcBHyMnK0tPU
1dbX2Nna4uPk5ebn6Onq8vP09fb3+Pn6/9oADAMBAAIRAxEAPwDk27c7rM/m
+QhjL28e4SFvnC7WIxggbjkgdOnzCoWcKeI1QquCRnLHJOTk9eccYHA75Jd
bTy2d1Fcw
```

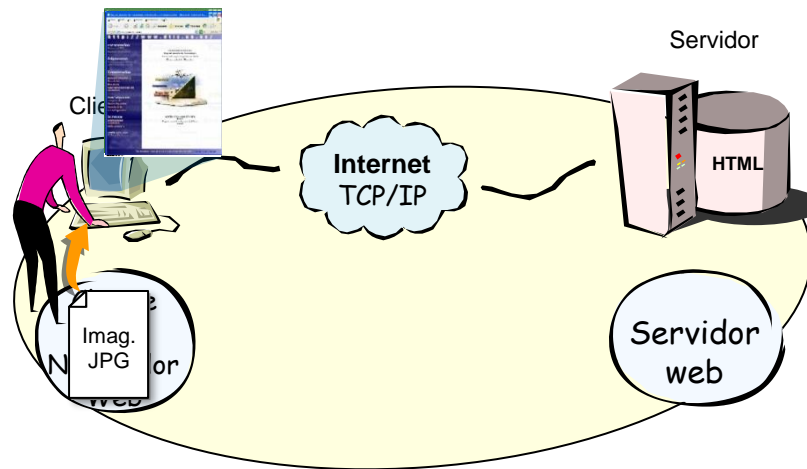
```
200 OK HTTP/1.0
Content-Type: image/jpg; name="menu.jpg"
Content-Transfer-Encoding: base64
```

[illegible]

-----8CFDA75A284D5A8033E016C87CBCE897-----

Modelo HTTP Básico

servicio HTTP básico



Modelo HTTP Básico

servici

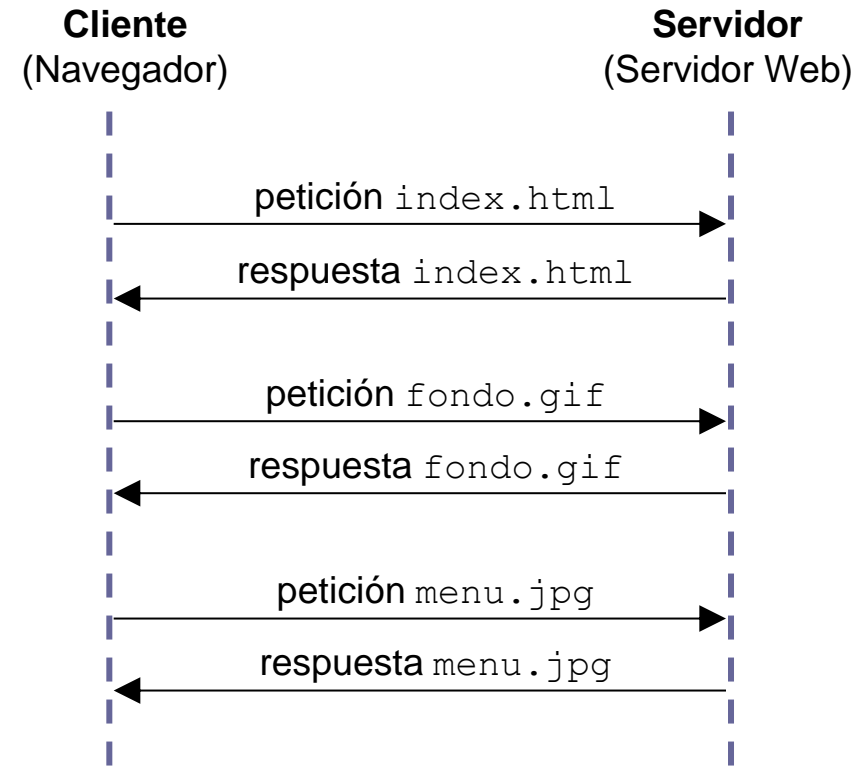


Diagrama de Secuencia petición-respuesta HTTP
con las interacciones a lo largo del tiempo

serv

```
C:\WINDOWS\system32\cmd.exe

C:\> telnet www.dtic.ua.es 80
Trying www.dtic.ua.es...
Connected to www.dtic.ua.es
Escape character is '^]'.
GET /index.html HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 04 Nov 2004 17:59:15 GMT
Server: Apache/2.0.40 (Red Hat Linux)
Last-Modified: Wed, 03 Nov 2004 13:01:02 GMT
ETag: "378029-902-3bd83f80"
Accept-Ranges: bytes
Content-Length: 256
Connection: close
Content-Type: text/html; charset=ISO-8859-1

<html>
<head>
<title>Tecnología Informática y Computación</title>
</head>

<body background="fondo.gif">
Hola a Todos<br>

</body>

</html>

Se ha perdido la conexión con el host.

C:\> _
```

serv

```
C:\WINDOWS\system32\cmd.exe

C:\> telnet www.dtic.ua.es 80
Trying www.dtic.ua.es...
Connected to www.dtic.ua.es
Escape character is '^]'.
GET /index.html HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 04 Nov 2004 17:59:15 GMT
Server: Apache/2.0.40 (Red Hat Linux)
Last-Modified: Wed, 03 Nov 2004 13:01:02 GMT
ETag: "378029-902-3bd83f80"
Accept-Ranges: bytes
Content-Length: 256
Connection: close
Content-Type: text/html; charset=ISO-8859-1

<html>
<head>
<title>Tecnología Informática y Computación</title>
</head>

<body background="fondo.gif">
Hola a Todos<br>

</body>

</html>

Se ha perdido la conexión con el host.

C:\> _
```

serv

```
C:\WINDOWS\system32\cmd.exe

C:\> telnet www.dtic.ua.es 80
Trying www.dtic.ua.es...
Connected to www.dtic.ua.es
Escape character is '^]'.
GET /index.html HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 04 Nov 2004 17:59:15 GMT
Server: Apache/2.0.40 (Red Hat Linux)
Last-Modified: Wed, 03 Nov 2004 13:01:02 GMT
ETag: "378029-902-3bd83f80"
Accept-Ranges: bytes
Content-Length: 256
Connection: close
Content-Type: text/html; charset=ISO-8859-1
[línea en blanco]
<html>
<head>
<title>Tecnología Informática y Computación</title>
</head>

<body background="fondo.gif">
Hola a Todos<br>

</body>

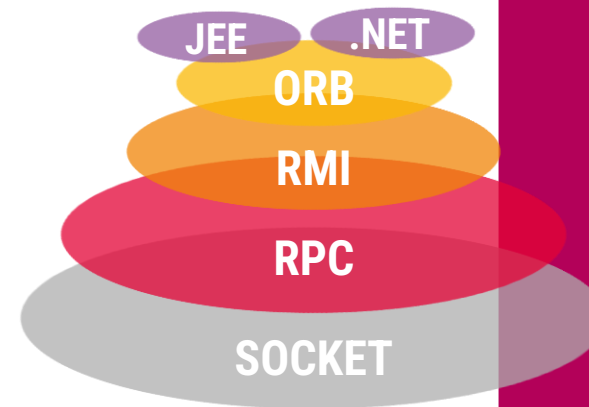
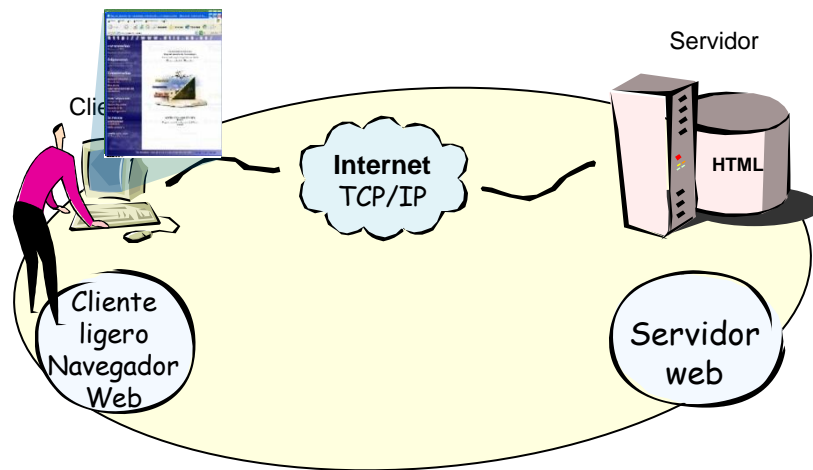
</html>

Se ha perdido la conexión con el host.

C:\> _
```

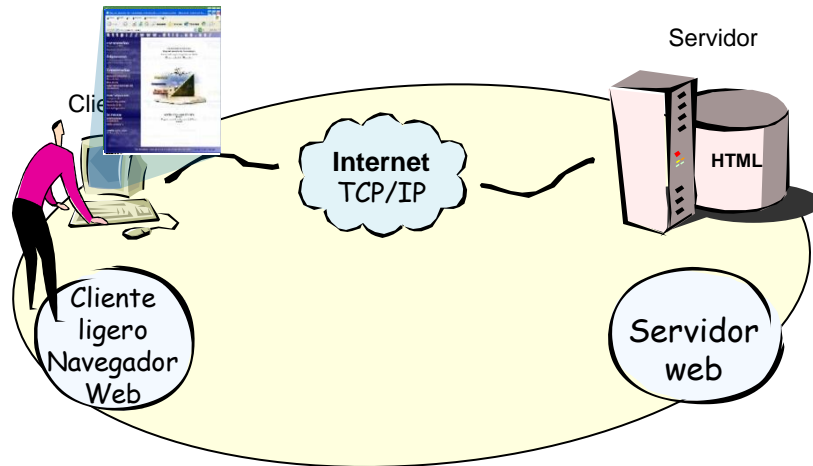
Modelo HTTP Básico

servicio HTTP básico

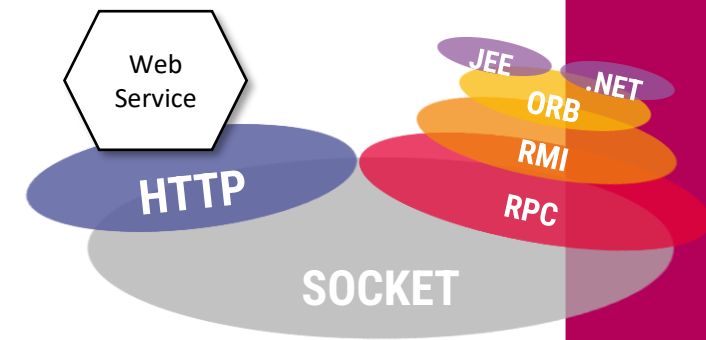


Modelo HTTP Básico

servicio HTTP básico



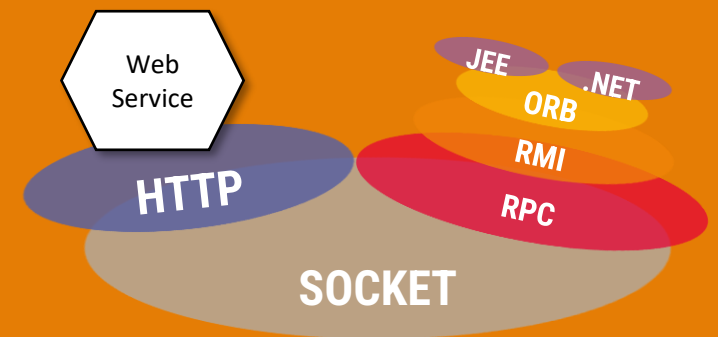
- Uso como **protocolo** de transporte
- **Aprovecha** las arquitecturas de red basadas en **firewalls** y **proxis**
- Orientado a **conexión**
- **Sin estado** (caché y balanceadores)



- ✓ Definición
- ✓ SOA
- ✓ REST
- ✓ Otras tecnologías

Servicios Web

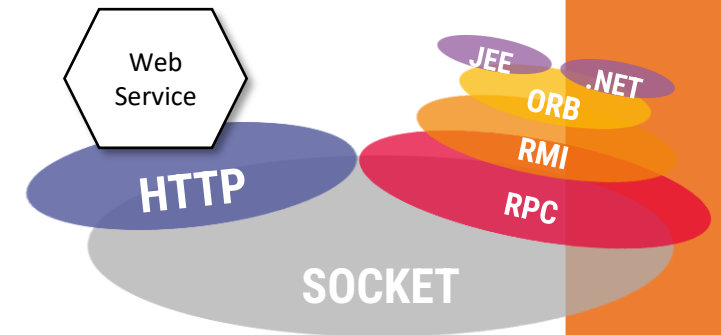
Contenidos



Servicios WEB

Características

- **Evolución** natural de los **sistemas distribuidos** y la necesidad de comunicación entre aplicaciones en diferentes plataformas

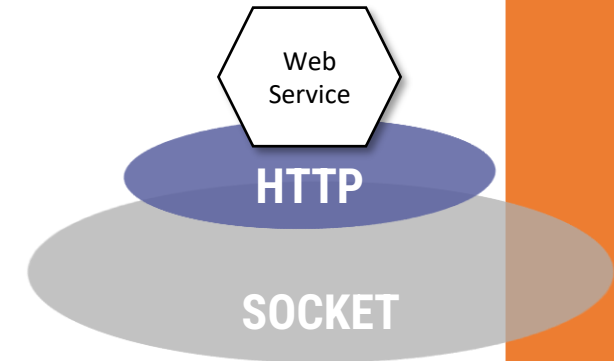


Servicios Web

Servicios WEB

Características

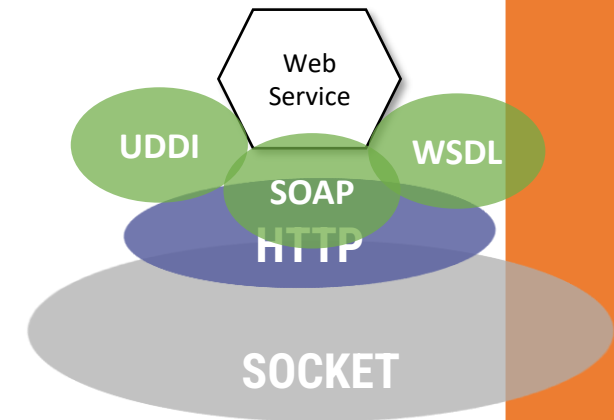
- **Evolución** natural de los **sistemas distribuidos** y la necesidad de comunicación entre aplicaciones en diferentes plataformas



Servicios WEB

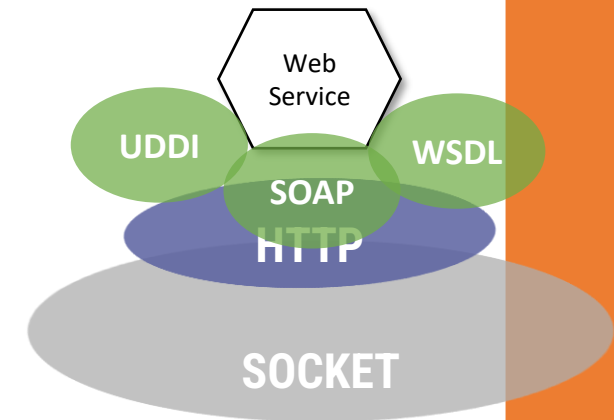
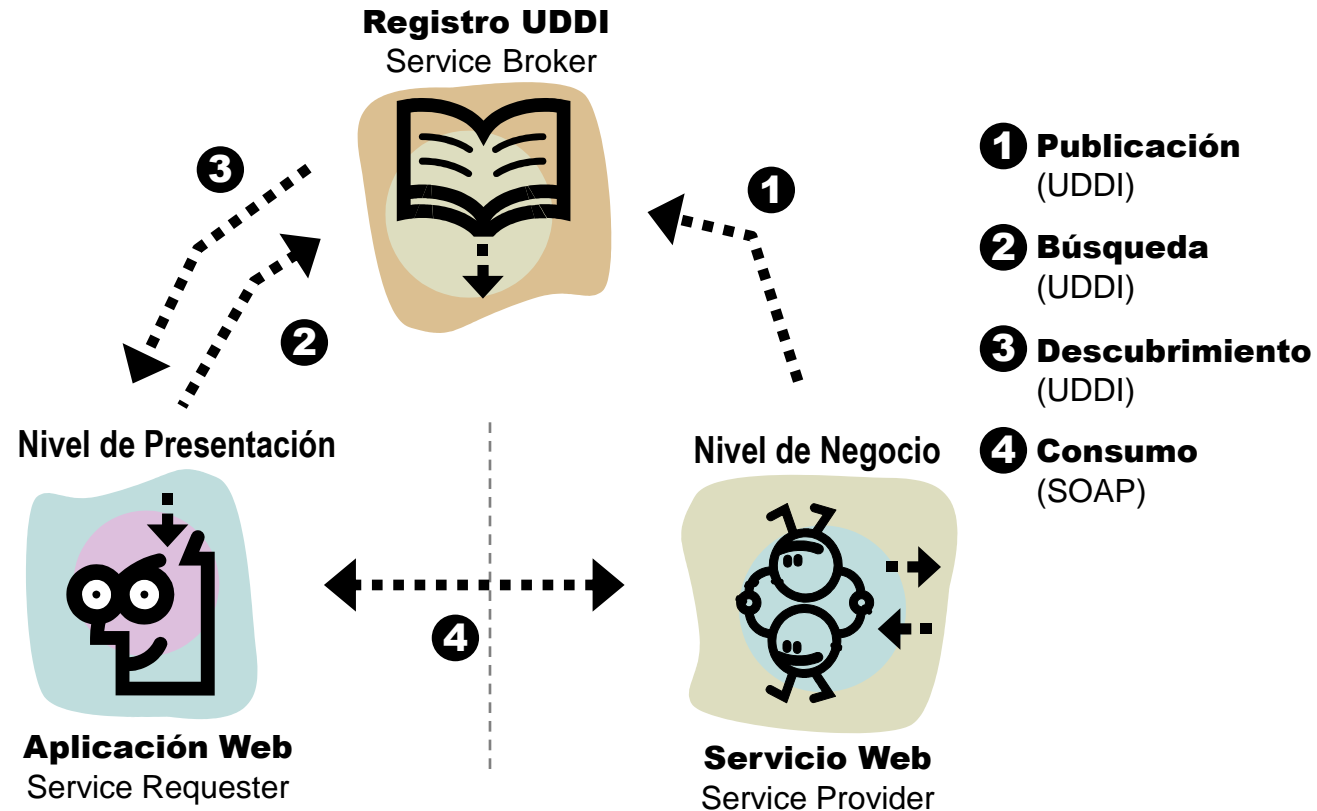
Características

- **Evolución** natural de los **sistemas distribuidos** y la necesidad de comunicación entre aplicaciones en diferentes plataformas
- **SOAP** (Protocolo de **acceso** a **objetos** simples) y el **lenguaje** de **descripción** de servicios web (**WSDL**) fue lo que realmente impulsó la adopción generalizada de los servicios web
- Se **consolidó** aún más con **UDDI** (Universal Description, Discovery, and Integration) para el **registro** y **descubrimiento** de servicios web



Servicios WEB

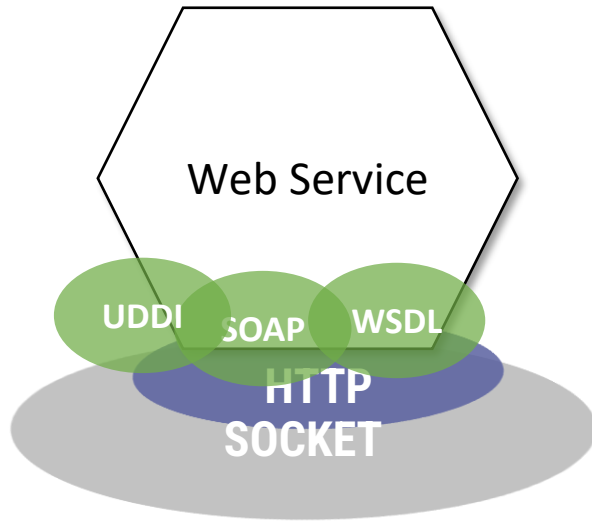
SOA



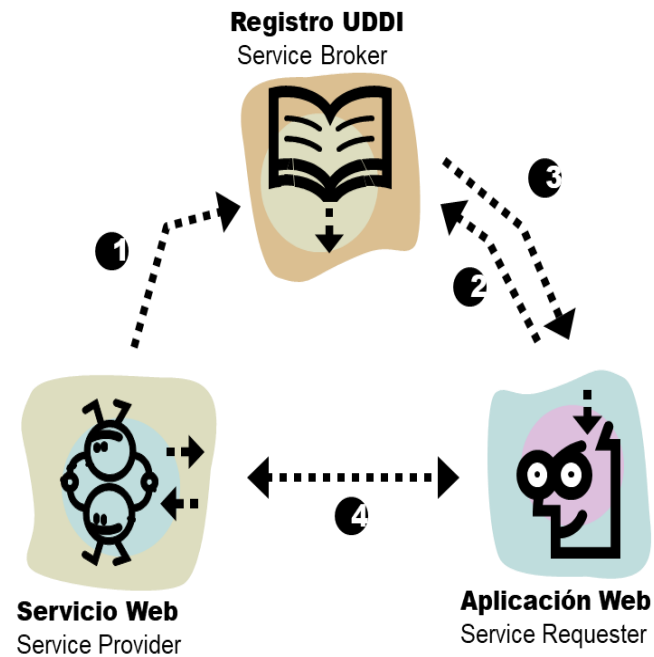
Servicios Web

Servicios WEB

Características



SOA

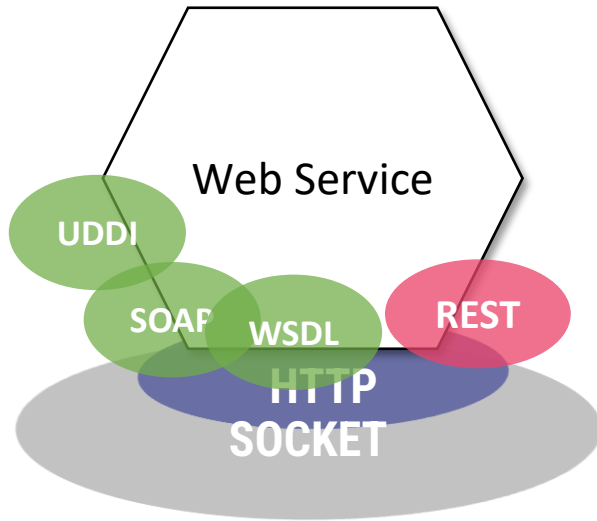


- 1 **Publicación**
(UDDI)
- 2 **Búsqueda**
(UDDI)
- 3 **Descubrimiento**
(UDDI)
- 4 **Consumo**
(SOAP)

Servicios Web

Servicios WEB

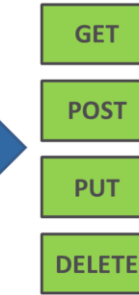
Características



REST



Client sends a **request**

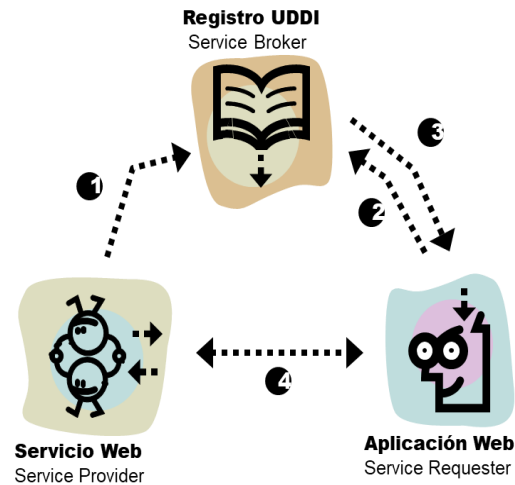


HTTP methods



Server sends a **response**

SOA



- 1 **Publicación**
(UDDI)
- 2 **Búsqueda**
(UDDI)
- 3 **Descubrimiento**
(UDDI)
- 4 **Consumo**
(SOAP)

Servicios Web

REST (Representational State Transfer)

- p.v. Programador: conjunto de **principios y buenas prácticas** para la **interacción** entre distintos **componentes**.

01

Alternativa a SW tradicionales

Alternativa más ligera a Servicios Web tradicionales (basados en: RPC, SOAP o WSDL) para mantener los principios de SOA (Arquitecturas Orientadas a Servicios).

02

Protocolo

El protocolo más usado que cumple esta definición es **HTTP**

- Toda aplicación web bajo HTTP es una aplicación REST.
- Aunque no necesariamente son servicios web RESTful.

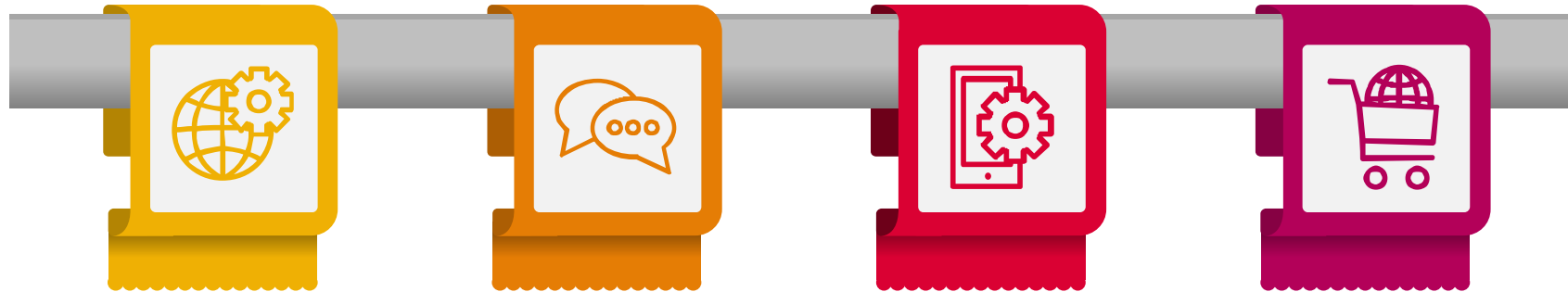
03

Selección

La **decisión** sobre la **tecnología** dependerá de las **características** del proyecto y requiere un análisis.

REST

Reglas básicas de una aplicación REST para alcanzar objetivos



Arquitectura cliente-servidor

Separación e independencia entre los 2 agentes básicos que intervienen: el cliente y el servidor

Stateless

El servidor no tiene por qué almacenar datos del cliente para mantener su estado.

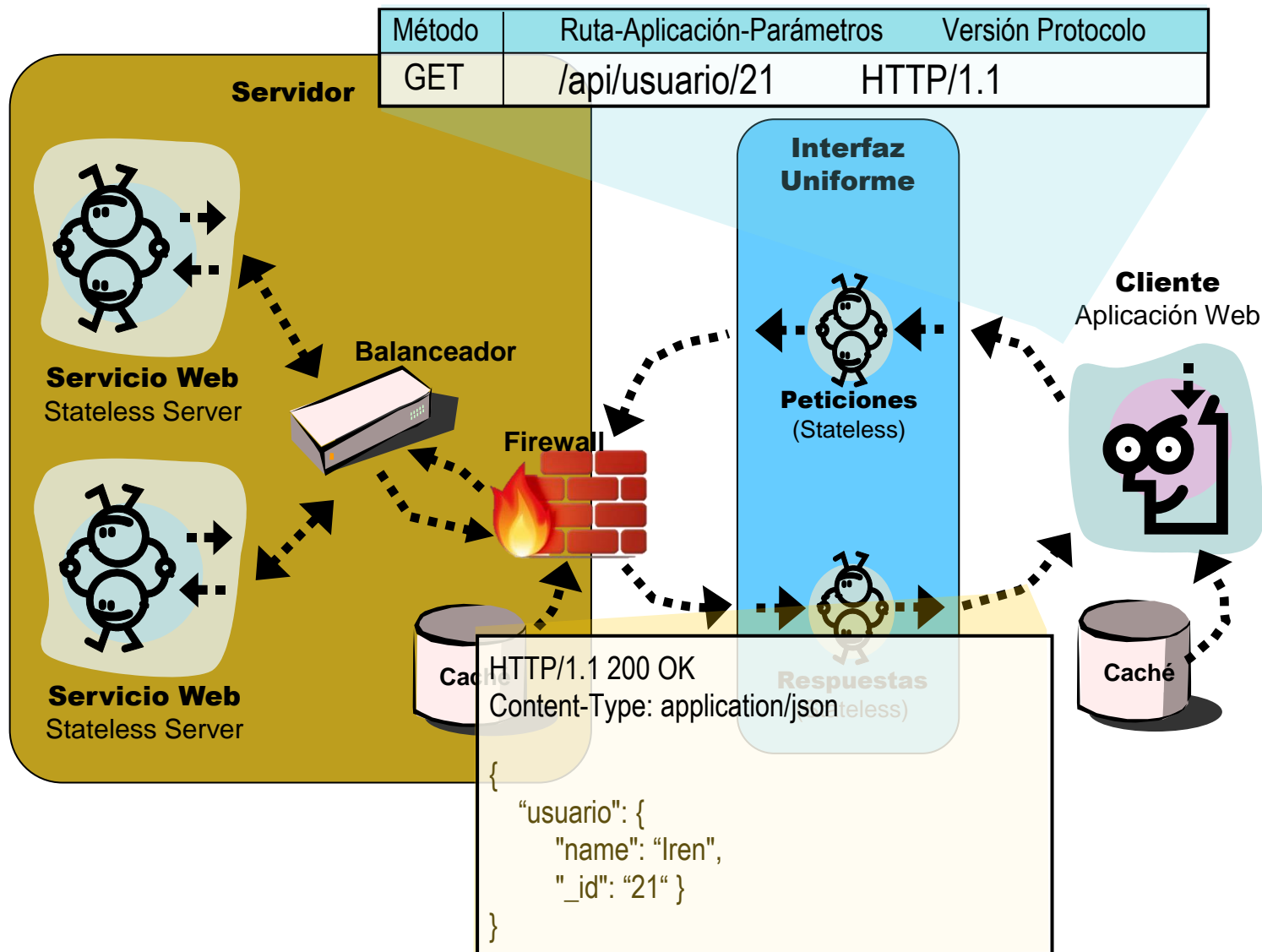
"Cacheable"

El servidor que sirve las peticiones del cliente debe definir algún modo de cachear dichas peticiones para aumentar el rendimiento, escalabilidad, etc. HTTP implementa esto con la cabecera Cache-control, mediante varios parámetros

Interfaz uniforme

Todos los servicios REST compartirán una forma de invocación y métodos uniforme utilizando los métodos GET, DELETE,...

Arquitectura REST



Servicio Web RESTful

¿Cómo diseñar un servicio Web REST?

Identificar

Identificar todas las **entidades** conceptuales que se desea exponer como servicio



URL

Crear una URL para cada recurso. Los recursos deberían ser nombres **no verbos** (acciones).



Servicio Web RESTful

¿Cómo diseñar un servicio Web REST?

Todos los recursos accesibles mediante **GET** **no** deberían tener **efectos** secundarios. Los recursos deberían devolver la representación del recurso. Por tanto, invocar al recurso **no** debería ser el **resultado** de **modificarlo**.



Categorizar

Categorizar los **recursos** de acuerdo con si los clientes pueden obtener una **representación** del recurso o si pueden **modificarlo**. Para el primero, debemos hacer los recursos accesibles utilizando un HTTP GET. Para el último, debemos hacer los recursos accesibles mediante los verbos HTTP: POST, PUT y DELETE.

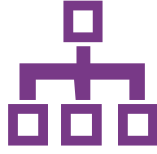
Servicios Web

Servicio Web RESTful

¿Cómo diseñar un servicio Web REST?

Describir cómo ha de ser invocado nuestro servicio mediante **OpenAPI**

Descripción



Hipervínculo

Ninguna representación debería estar **aislada**. Es recomendable poner hipervínculos dentro de la representación de un recurso para permitir a los clientes obtener **más información**.

Servicios Web

Servicio Web RESTful

REST, API REST y API RESTful

REST

Estilo de arquitectura de software que define un conjunto de **restricciones** y **principios** para el diseño de servicios web:

- Comunicación **sin estado** entre el cliente y el servidor
- Capacidad de **cachear** datos
- Uso de operaciones **CRUD** (Crear, Leer, Actualizar, Borrar) sobre recursos identificables mediante URLs

API REST

Implementación de una **API** que sigue los principios de **REST**.

El API debe cumplir con todas las restricciones de REST:

- **Métodos HTTP** de manera apropiada (GET, POST, PUT, DELETE, etc.)
- Manipulación de recursos a través de representaciones de estado,
- Hipermedios para la navegación entre recursos

Servicio Web RESTful

REST, API REST y API RESTful

REST

Estilo de arquitectura de software que define un conjunto de **restricciones** y **principios** para el diseño de servicios web:

- Comunicación **sin estado** entre el cliente y el servidor
- Capacidad de **cachear** datos
- Uso de operaciones **CRUD** (Crear, Leer, Actualizar, Borrar) sobre recursos identificables mediante URLs

API RESTful

Implementación de una **API** que sigue los principios de **REST**.

El API debe cumplir con todas las restricciones de REST:

- **Métodos HTTP** de manera apropiada (GET, POST, PUT, DELETE, etc.)
- Manipulación de recursos a través de representaciones de estado,
- Hipermedios para la navegación entre recursos

Servicio Web RESTful

REST, API REST y API RESTful

REST

Estilo de arquitectura de software que define un conjunto de **restricciones** y **principios** para el diseño de servicios web:

- Comunicación **sin estado** entre el cliente y el servidor
- Capacidad de **cachear** datos
- Uso de operaciones **CRUD** (Crear, Leer, Actualizar, Borrar) sobre recursos identificables mediante URLs

API RESTful

Implementación de una **API** que sigue los principios de **REST**.

El API debe cumplir con todas las restricciones de REST:

- **Métodos HTTP** de manera apropiada (GET, POST, PUT, DELETE, etc.)
- Manipulación de recursos a través de representaciones de estado,
- Hipermedios para la navegación entre recursos

API REST

Implementación de una **API** que **NO** sigue **estrictamente** todos los principios de **REST**.

Servicio Web RESTful

Ejemplo de implementación

Un ejemplo de una **API REST** sería una API que proporciona acceso a una base de datos de libros con estos **endpoints**:

- **GET /obtenerLibros**: Para obtener una lista de todos los libros.
- **GET /obtenerLibro/{id}**: Para obtener detalles sobre un libro específico.
- **POST /crearLibro**: Para agregar un nuevo libro a la base de datos.
- **POST /actualizarLibro/{id}**: Para actualizar la información de un libro existente.
- **GET /eliminarLibro/{id}**: Para eliminar un libro de la base de datos.

Endpoints en una **API RESTful** que por tanto sigue más de cerca los principios de REST y hace una definición correcta:

- **GET /libros**: Para obtener una lista de todos los libros.
- **GET /libros/{id}**: Para obtener detalles sobre un libro específico.
- **POST /libros**: Para agregar un nuevo libro a la base de datos.
- **PUT /libros/{id}**: Para actualizar la información de un libro existente.
- **DELETE /libros/{id}**: Para eliminar un libro de la base de datos.

API REST

VS

API RESTful

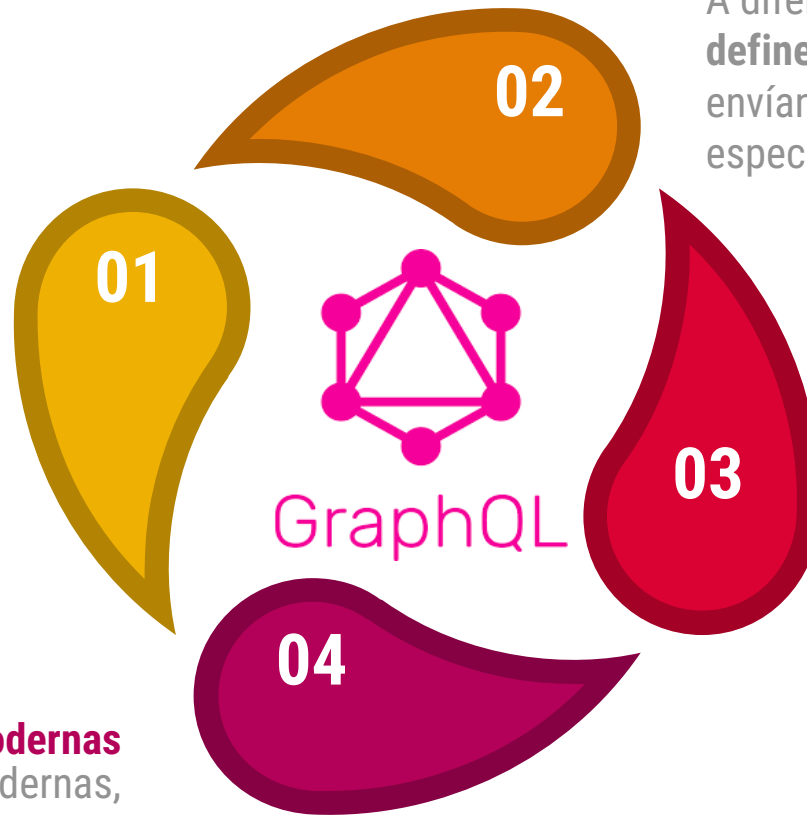
Otras Tecnologías

Otras Tecnologías

GraphQL

Alternativa
Alternativa relativamente nueva (2015) que permite a los clientes solicitar solo los datos que necesitan

Aplicaciones modernas
Popular en aplicaciones modernas, especialmente en entornos donde se necesita una **recuperación** de **datos** eficiente y **personalizada**



Especificar datos

A diferencia de REST, donde el **servidor** **define** la **estructura** de los **datos** que se envían, en GraphQL, los clientes pueden especificar qué datos desean recuperar

Reduce tráfico

Esto **reduce** el **exceso** de **datos** transmitidos y permite una mayor flexibilidad en la comunicación entre aplicaciones

Otras Tecnologías

Websocket

Comunicación

Proporciona un canal de comunicación **bidireccional** (cliente y servidor pueden iniciar comunicación), **full-duplex**

Alternativa a REST (en realidad a HTTP)

- Protocolo de más **bajo nivel**
- Protocolo **con estado**
- Protocolo **más ligero**



TCP

Utiliza una conexión **TCP** única (frente a REST)

Protocolo de comunicación

WebSocket es un **protocolo de comunicaciones** mientras que REST es un patrón arquitectónico basado en HTTP

Otras Tecnologías

gRPC

Uso

Ha ganado popularidad en **aplicaciones distribuidas** y **microservicios**

Streaming Bidireccional

Admite el streaming **bidireccional**, lo que permite la transmisión de datos de manera eficiente en ambas direcciones entre el cliente y el servidor..

Tipado Fuerte

Utiliza **Protocol Buffers** para definir sus servicios y mensajes, lo que permite una **serialización binaria** altamente **eficiente** y un **tipado fuerte** en la comunicación entre servicios



Comunicación

Especialmente diseñado para entornos donde se necesitan **comunicaciones eficientes** y de **baja latencia**

Protocolo





Utiliza **HTTP/2** como protocolo subyacente, lo que permite la transmisión de datos de manera más eficiente que HTTP/1.x utilizado por REST y GraphQL

Soporte Multilenguaje

Proporciona soporte **multilenguaje**, lo que lo hace ideal para entornos donde diferentes partes de una aplicación están escritas en diferentes lenguajes de programación

Selección de Tecnología

Resumen

	Para aplicaciones donde la interoperabilidad y la simplicidad son prioritarias, especialmente para operaciones CRUD	Aplicaciones web y móviles que requieren acceso a recursos a través de HTTP Ejemplos: APIs de redes sociales como Twitter , Facebook , APIs de servicios de pago como Stripe
	Para aplicaciones complejas que requieren flexibilidad en las consultas y manejo eficiente de datos complejos.	Aplicaciones con múltiples vistas de datos Ejemplos: APIs de GitHub , Yelp, APIs internas de Facebook . Suele ser utilizado de cara a la aplicación cliente.
	Para aplicaciones que requieren alta velocidad , eficiencia y comunicación entre microservicios .	Aplicaciones con requisitos de rendimiento y escalabilidad. Comunicación entre microservicios en un entorno de microservicios. Ejemplos: Servicios internos de Google Suele ser utilizado para comunicación servicios backend.
	Para aplicaciones que requieren comunicación bidireccional persistente y tiempo real .	Aplicaciones de chat en tiempo real, Juegos multijugador en tiempo real , Aplicaciones de seguimiento en tiempo real. Ejemplos: Aplicaciones de chat en tiempo real como Slack, aplicaciones de juegos en línea, sistemas de seguimiento de flotas en tiempo real

Tecnologías para los Sistemas Distribuidos

Contenidos



Mecanismos de comunicación distribuida

IPC, Sockets, RPC, RMI, ORB



Revisión de tecnologías Web

Modelo HTTP Básico



Servicios Web

Definición, SOA, REST

Resumen general

Tecnologías de Comunicación → Evolución



Sistemas Operativos y Distribuidos

Iren Lorenzo Fonseca
iren.fonseca@ua.es



TEMA 3. Sistemas Distribuidos.

Tecnologías para los
Sistemas Distribuidos