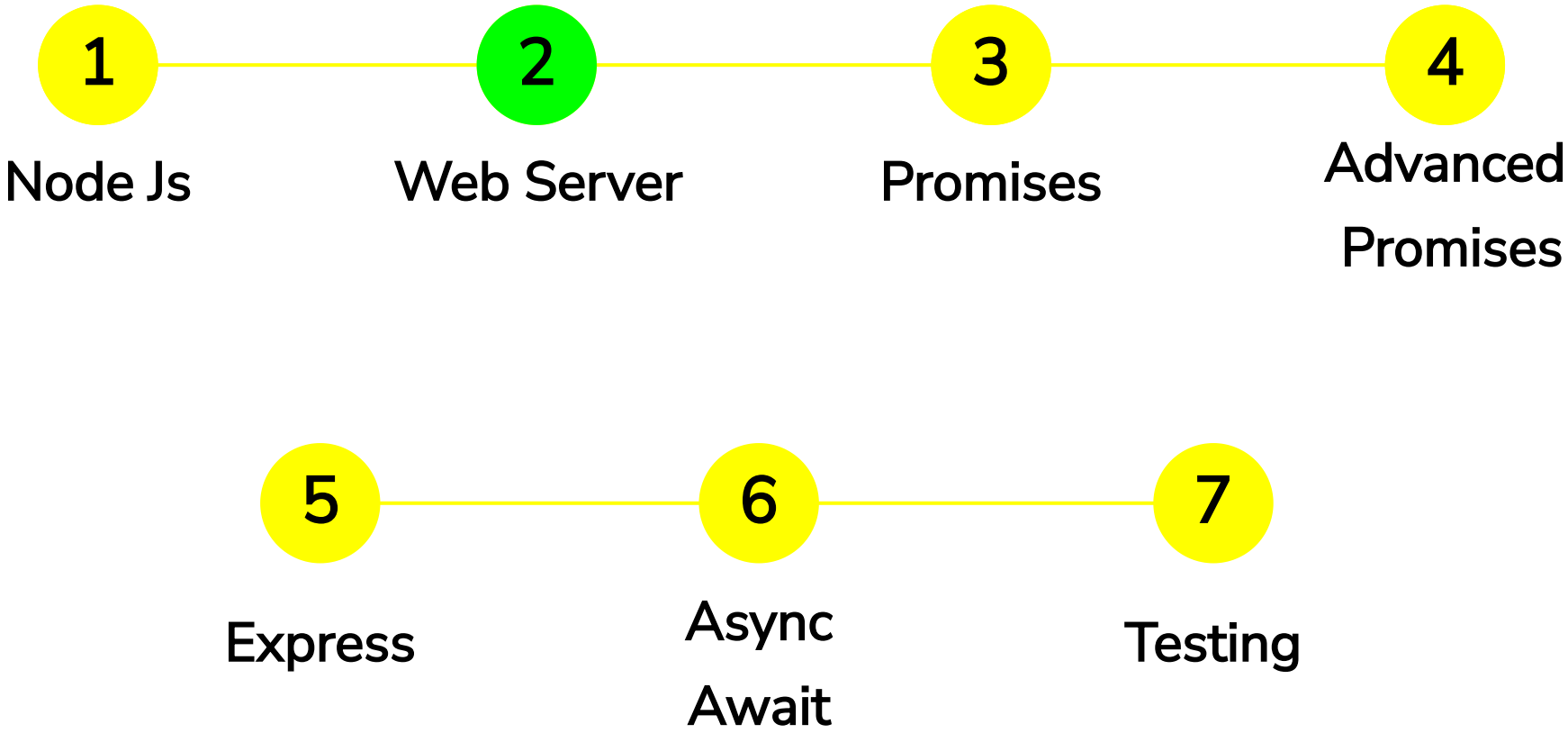

HENRY

Bienvenidos

Web Server

Contenido M3



capitulemos...

de -> *Es el entorno de ejecución de Javascript.*
s permite desarrollar aplicaciones escalables y
ablece una conexión bidireccional y dinámica
re cliente-servidor

dulos -> *Bloque de código reutilizable en una*
icación. Podemos utilizar CommonJs en donde
a archivo es un módulo y cada módulo es un
hivo.

stor de paquetes -> *Se encarga de descargar y*
ministrar las dependencias de paquetes de
de.js, este gestor es npm.

capitulemos...

kage.json -> *Es un archivo en formato json en de queda guardada la configuración de tro proyecto. Se debe crear en la raíz del ecto.*

ionado -> *La configuración de nuestro ecto la podemos versionar y se caracteriza un major que indica de cambios radicales, or nos agrega funcionalidades y patch que nos a corrección de bugs.*

OBJETIVO DE LA CLASE

Tener los conocimientos claros en la comunicación del modelo cliente-servidor, cómo se envía la información, para qué nos sirve un servidor web y vamos a crear nuestras primeras rutas backend.

¿Qué veremos hoy?

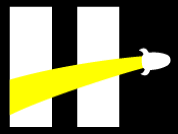
- Web Server
- Modelo cliente-servidor
- Networks
- Encapsulación
- Protocolos

¿Qué veremos hoy?

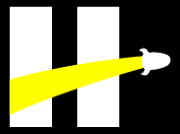
- Server Básico
- Formas de enviar la información
- Serialize / Deserialize
- Rutas
- RESTful API

Espacio de interacción

Recuerden hacer las preguntas en el **Q&A** con contexto para que no se pierdan en el chat.



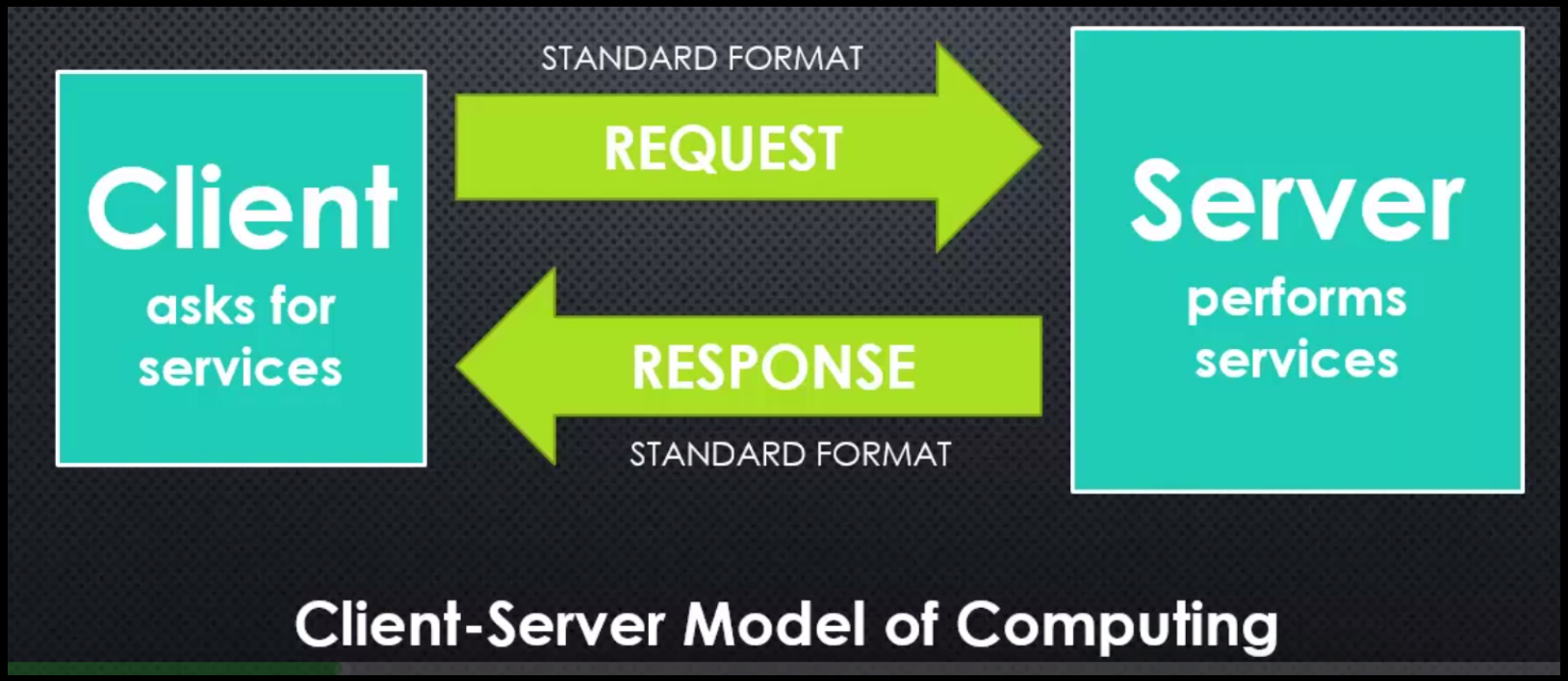
Web Server



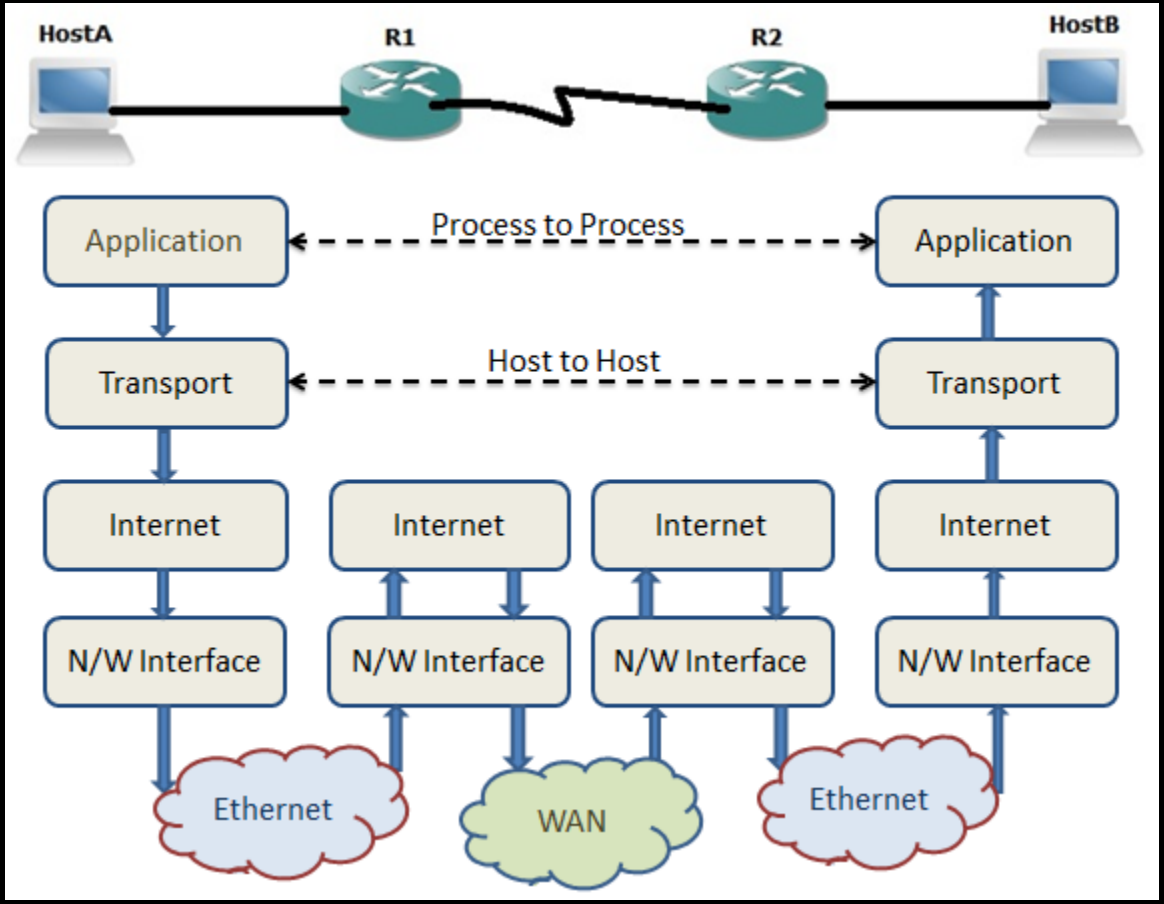
Web Server

“ Un servidor web es cualquier computadora o sistema que procese solicitudes (requests) y que devuelva una respuesta (response) a través de un protocolo de red.

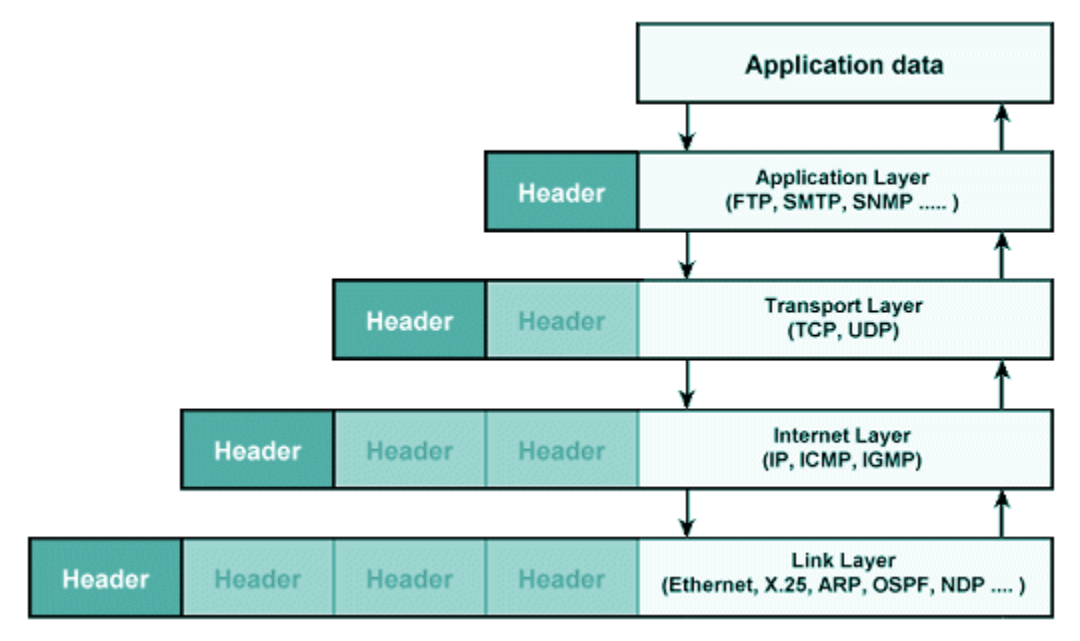
Modelo Cliente-Servidor



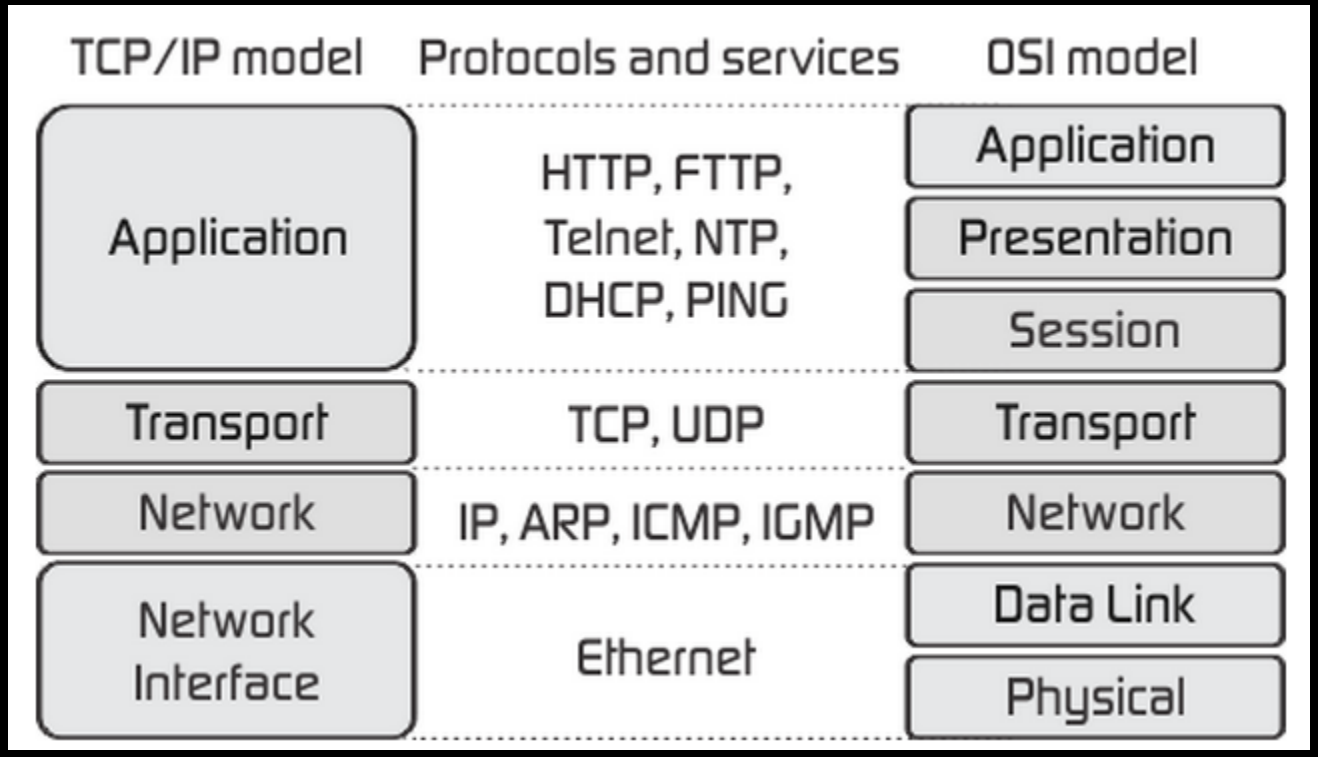
Networks

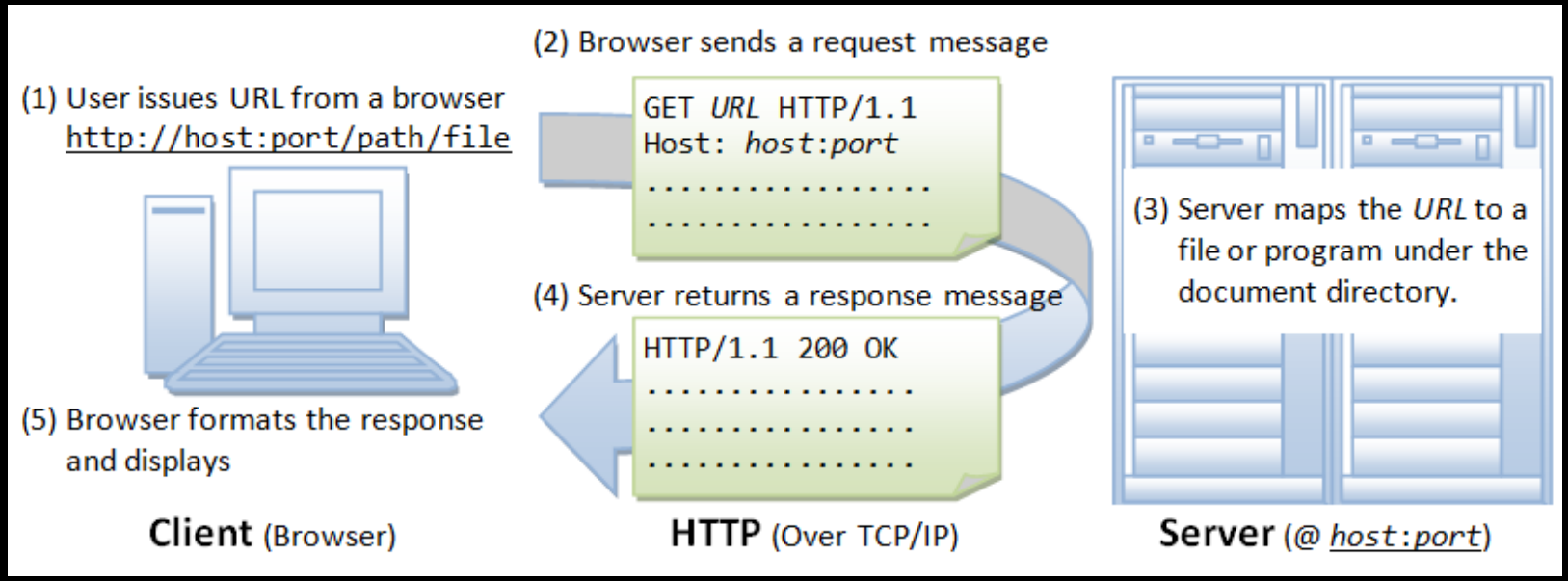

HostA
R1
R2
HostB
Application
Process to Process
Application
Transport
Host to Host
Transport
Internet
Internet
Internet
Internet
N/W Interface
N/W Interface
N/W Interface
N/W Interface
Ethernet
WAN
Ethernet

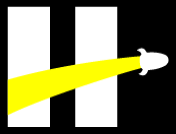
Encapsulación



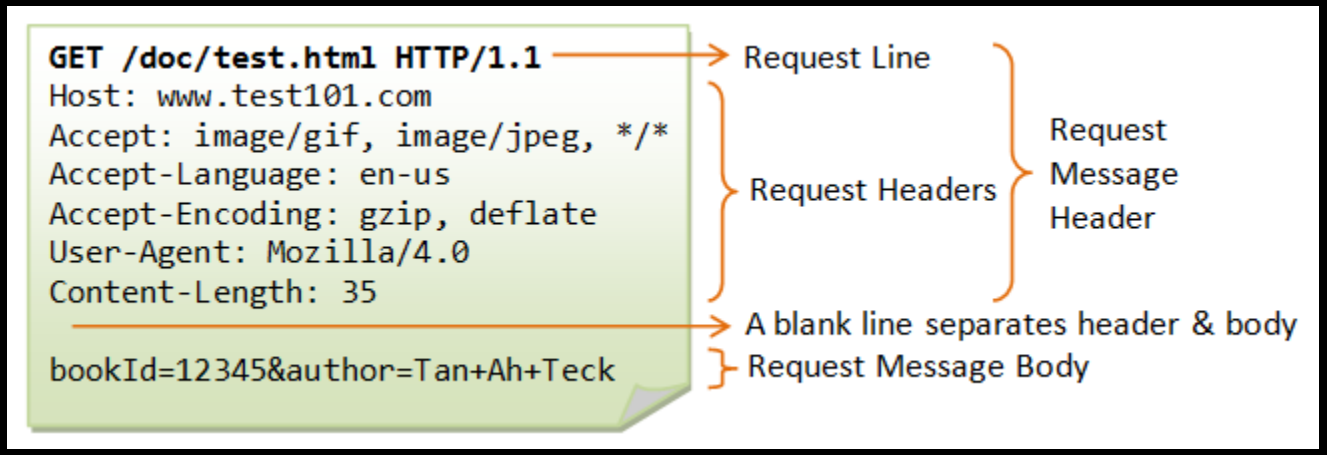
Protocolos


TCP/IP model
Protocols and services
OSI model
Application
HTTP, FTTP, Telnet, NTP, DHCP, PING
Application
Presentation
Session
Transport
TCP, UDP
Transport
Network
IP, ARP, ICMP, IGMP
Network
Network Interface
Ethernet
Data Link
Physical


(2) Browser sends a request message
(1) User issues URL from a browser
http://host:port/path/file
GET URL HTTP/1.1
Host: host:port
.....
.....
(3) Server maps the URL to a file or program under the document directory.
(4) Server returns a response message
HTTP/1.1 200 OK
.....
.....
.....
(5) Browser formats the response and displays
Client (Browser)
HTTP (Over TCP/IP)
Server (@ host:port)

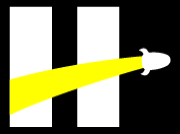


HTTP



A *socket* is one endpoint of a two-way communication link between two programs running on the network.

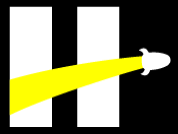
A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.



Server Básico

```
1
2  var http = require('http'); // importamos el módulo http para poder trabajar con el protocolo
3
4
5  // Creamos una serie de events listener,
6  // que van a escuchar por requests que ocurren en este socket
7
8  http.createServer( function(req, res){
9
10         //Para crear un response empezamos escribiendo el header
11         res.writeHead(200, { 'Content-Type':'text/plain' })
12     //Le ponemos el status code y algunos pair-values en el header
13         res.end('Hola, Mundo!\n');
14
15 }).listen(1337, '127.0.0.1');
16 // Por último tenemos que especificar en que puerto y
17 // en qué dirección va a estar escuchando nuestro servidor
```

< DEMO />

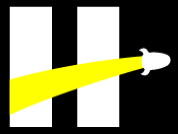


Enviando HTML

```
1  var http = require('http');
2  var fs   = require('fs');
3
4  //Importamos el módulo fs que nos permite leer y escribir archivos del file system
5
6  http.createServer( function(req, res){
7
8          res.writeHead(200, { 'Content-Type':'text/html' })
9          var html = fs.readFileSync(__dirname +'/html/index.html');
10         res.end(html);
11
12
13 }).listen(1337, '127.0.0.1');
```

< DEMO />

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4          <title>Prueba!</title>
5  </head>
6  <body>
7          <h1>Hola, Mundo!</h1>
8          <p>Bienvenidos!</p>
9  </body>
10 </html>
```

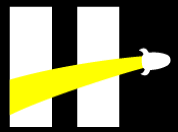


Enviando HTML (Templates)

```
var http = require('http');
var fs   = require('fs');
//Importamos el módulo fs que nos permite leer y escribir archivos del file system

http.createServer( function(req, res){

        res.writeHead(200, { 'Content-Type':'text/html' })
        var html = fs.readFileSync(__dirname +'/html/template.html', 'utf8');
    //Codificamos el buffer para que sea una String
        var nombre = 'Soy Henry';
    //Esta es la variable con la que vamos a reemplazar el template
        html = html.replace('{nombre}', nombre
    // Usamos el método replace es del objeto
        res.end(html);


}).listen(1337, '127.0.0.1');
```

```
<!DOCTYPE html>
<html>
<head>
        <title>Prueba!</title>
</head>
<body>
        <h1>Hola, {nombre}!</h1>
        <p>Bienvenidos!</p>
</body>
</html>
```

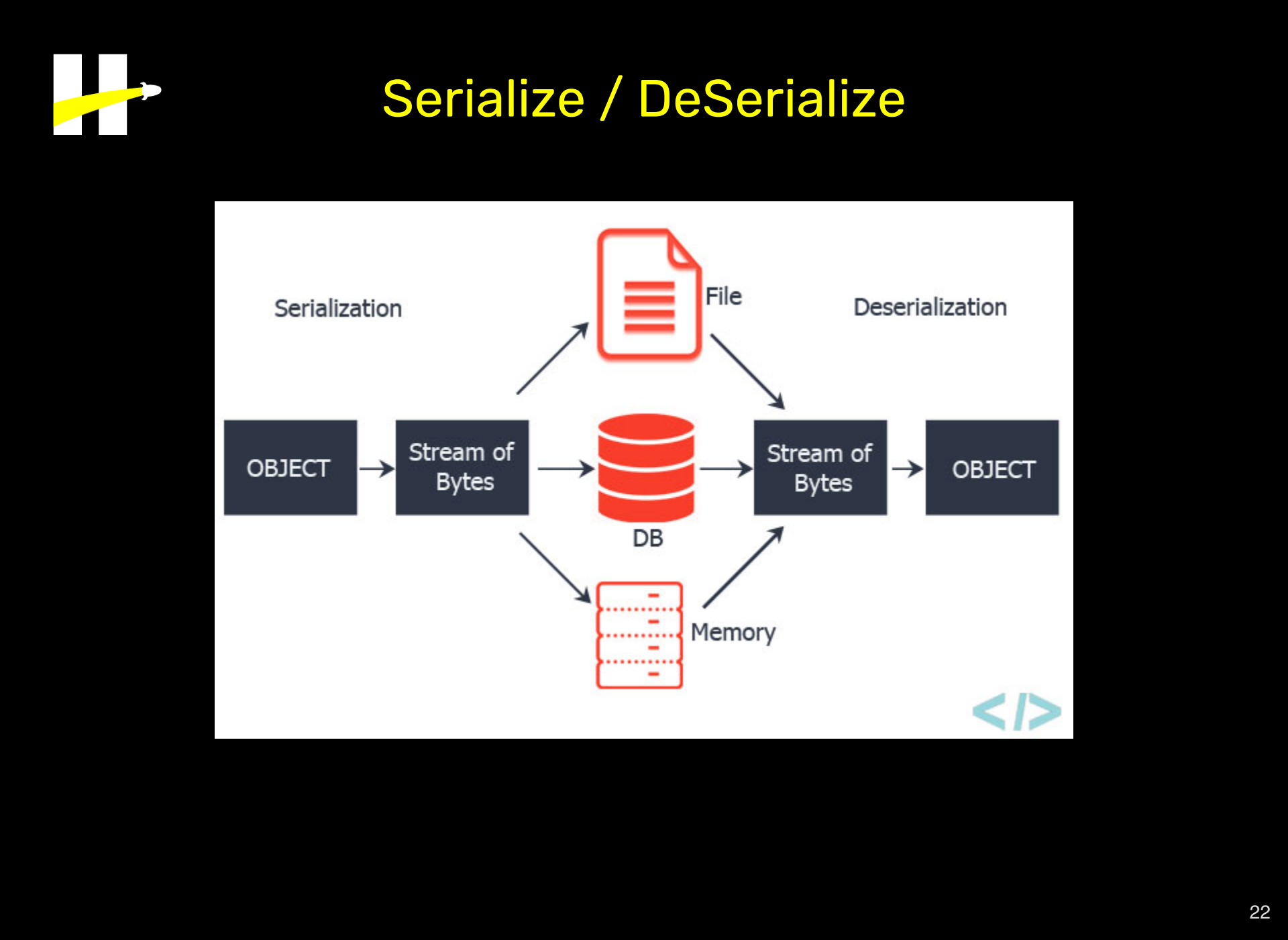
< DEMO />



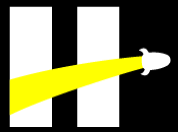
JSON

```
1  var http = require('http');
2  var fs   = require('fs');
3
4  http.createServer( function(req, res){
5
6          res.writeHead(200, { 'Content-Type':'application/json' })
7      //Vamos a devolver texto en formato JSON
8          var obj = {
9                  nombre: 'Juan',
10                 apellido: 'Perez'
11         }; //Creamos un objeto de ejemplo para enviar como response
12
13         res.end( JSON.stringify(obj) );
14     //Antes de enviar el objeto, debemos parsearlo y transformarlo a un string JSON
15
16 }).listen(1337, '127.0.0.1');
```

< DEMO />



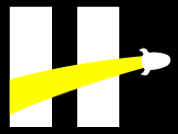
Serialize / DeSerialize



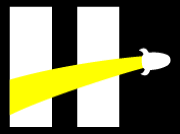
Rutas

```
1  var http = require('http');
2  var fs   = require('fs');
3
4  http.createServer( function(req, res){
5          if( req.url === '/'){
6                  res.writeHead(200, { 'Content-Type':'text/html' })
7                  var html = fs.readFileSync(__dirname +'/html/index.html');
8                  res.end(html);
9          }else if(req.url === '/api'){
10                 res.writeHead(200, { 'Content-Type':'application/json' })
11                 var obj = {
12                         nombre: 'Juan',
13                         apellido: 'Perez'
14                 };
15                 res.end( JSON.stringify(obj) );
16         } else{
17                 res.writeHead(404); //Ponemos el status del response a 404: Not Found
18                 res.end(); //No devolvemos nada más que el estado.
19         }
20
21 }).listen(1337, '127.0.0.1');
```

< DEMO />

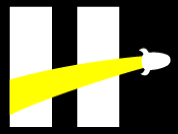


RESTful API



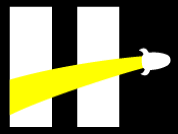
RESTful API

“ Rest es una arquitectura o forma de diseñar el backend de una aplicación que viva en internet. REST viene de "REpresentational State Transfer" y está basado fuertemente en cómo trabaja HTTP, que es el protocolo que usamos comunmente en la web.



RESTful API

Task	Method	Path
Create a new task	POST	/tasks
Delete an existing task	DELETE	/tasks/{id}
Get a specific task	GET	/tasks/{id}
Search for tasks	GET	/tasks
Update an existing task	PUT	/tasks/{id}



RESTful API

- Cliente - Servidor
- Stateless
- Cacheable
- Sistema de capas

Resumen

- **Web Server:** Es quien se encarga de despachar el contenido de un sitio web al cliente. Es la parte que el usuario no ve. Su función es recibir las solicitudes del cliente, consultarlas a la DB, procesarlas y devolver una respuesta.
- **Modelo cliente - servidor:** Es aquel en el cual el servidor envía recursos al cliente que los solicita.
- **Networks:** Vimos cómo viaja la información entre cliente y servidor a través de las redes de internet.
- **Protocolos:** Son reglas que permiten que el cliente entienda lo que comunica el servidor y viceversa.

Resumen

- **HTTP:** Protocolo de Transferencia de Hipertexto, es un protocolo que permite la comunicación entre cliente - servidor.
- **Nuestro primer server:** Recuerden que para ello precisamos el módulo http y conectarlo al puerto mediante el método listen.
- **Formas de enviar respuestas:** Vimos las distintas formas en las que podemos enviar respuestas al cliente: texto plano, html y json.
- **Serialize / Deserialize:** Aprendimos cómo un objeto se deserializa para viajar como archivos en flujo de bytes para luego serializarse una vez llega a su destino.
- **Nuestras primeras rutas:** Aprendimos a crear nuestra primera API.

¿Preguntas?

Homework

Pair Programming

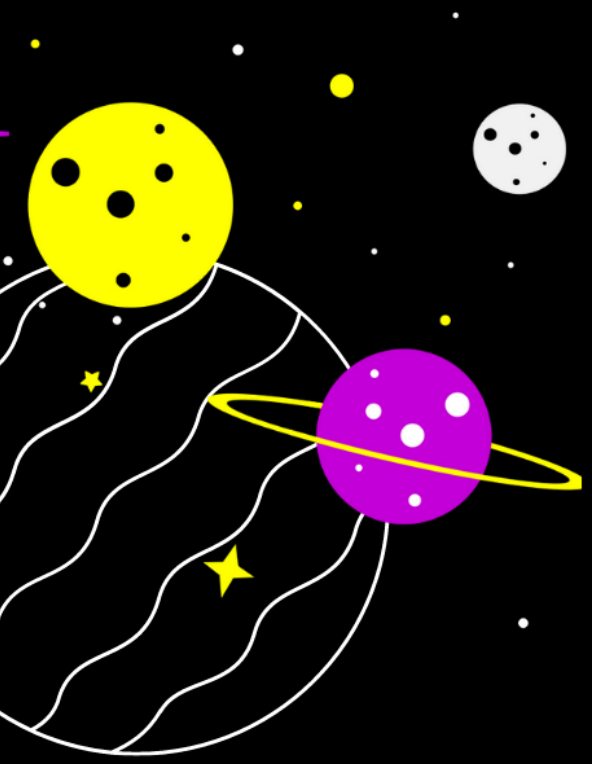
SUP

¿Preguntas?

PROXIMA CLASE

Promises

NRY



¡Muchas gracias!