

# PPSS PLANIFICACIÓN Y PRUEBAS DE SISTEMAS SOFTWARE

## Curso 2024-25



## Sesión S09: Pruebas de aceptación (2)

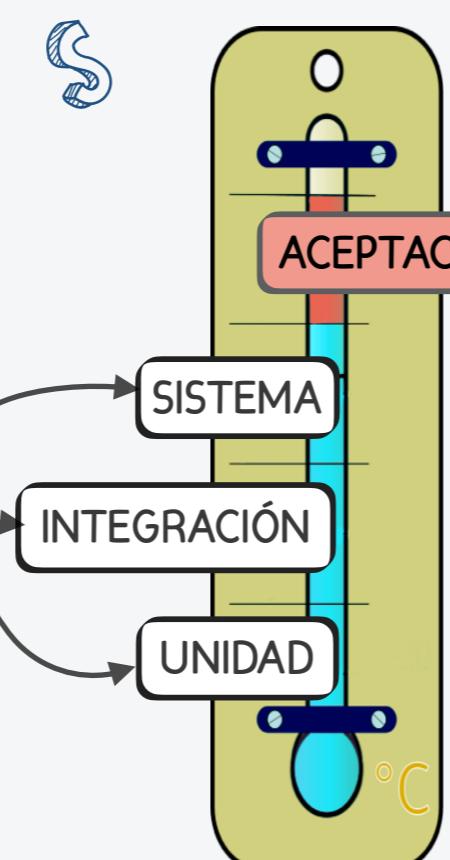
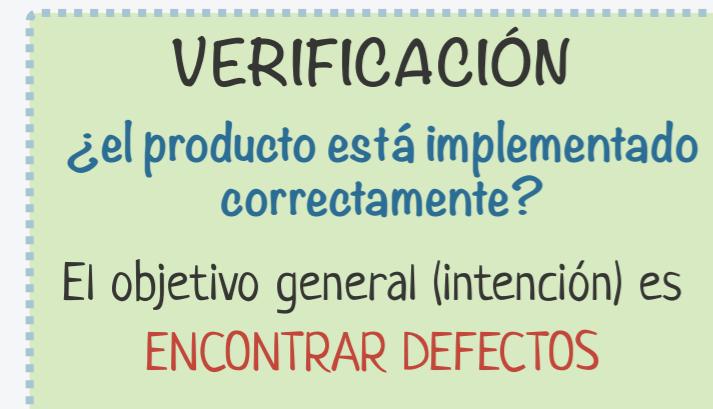
- Propiedades emergentes no funcionales
- Métricas
- Ejemplos de pruebas
  - Pruebas de carga
  - Pruebas de estrés
  - Pruebas estadísticas
- Pasos a seguir durante el proceso de pruebas
- Automatización de las pruebas: JMeter

Vamos al laboratorio...

# NIVELES DE PRUEBAS

P

P



**VALIDACIÓN**  
¿el producto es el correcto?

Las pruebas de aceptación determinan en qué **GRADO** el sistema satisface los **CRITERIOS DE ACEPTACIÓN**

Business acceptance testing

User acceptance testing ( $\alpha, \beta$ )

Propiedades emergentes

Cada nivel tiene una granularidad y objetivos concretos diferentes.  
Hay un **ORDEN** temporal entre ellos.

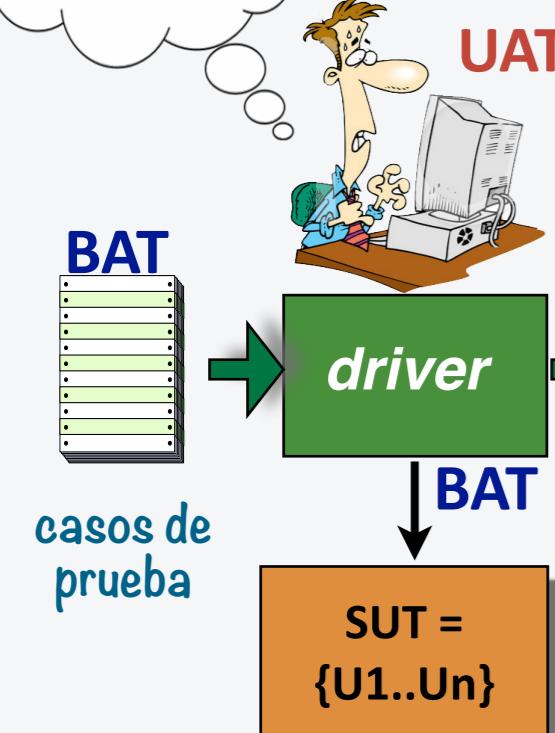
**FUNCIONALES**

Automatización de pruebas de aplicaciones Web

NO funcionales

Selenium WebDriver

JMeter



También para pruebas del **SISTEMA**

# PROPIEDADES EMERGENTES NO FUNCIONALES

P S → Solo tiene sentido aplicarlas al sistema como un TODO

- Hay dos tipos de propiedades **emergentes**:
  - **Funcionales**: describen lo **que** el sistema hace, o debería hacer (does view)
  - **No funcionales**: hacen referencia a "how well a system performs its functional requirements"
- Muchas de las propiedades emergentes NO FUNCIONALES se categorizan como "-ibilidades":
  - **fiabilidad**: probabilidad de funcionamiento sin fallos durante un tiempo determinado en un entorno específico (p.ej. fiabilidad del 90%)
  - **disponibilidad**: tiempo durante el cual el sistema proporciona servicio al usuario. Suele expresarse como: hh/dd (p.ej 24/7: 24 horas al día, 7 días por semana)
  - **mantenibilidad**: capacidad de un sistema para soportar cambios. Hay tres tipos de cambios: correctivos, adaptativos y perfectivos
    - \* **correctivos**: son provocados por errores detectados en la aplicación
    - \* **adaptativos**: son provocados por cambios en el hardware y/o software (sistema operativo) sobre los que se ejecuta nuestra aplicación
    - \* **perfectivos**: debidos a que se quiere añadir/modificar las funcionalidades existentes para ampliar/mejorar el "negocio" que sustenta nuestra aplicación
  - **escalabilidad**: hace referencia a la capacidad de mantener el tiempo de respuesta ante cambios en el número de usuarios que utilizan el sistema.
  - **robustez**: capacidad de un sistema para seguir funcionando a pesar de los fallos. (ante un fallo, el sistema es capaz de "recuperarse")

# MÉTRICAS

P

S

Las pruebas de aceptación determinan en qué **GRADO** el sistema satisface los **CRITERIOS DE ACEPTACIÓN**

S



Sí no podemos "medir"  
No podremos validar!!

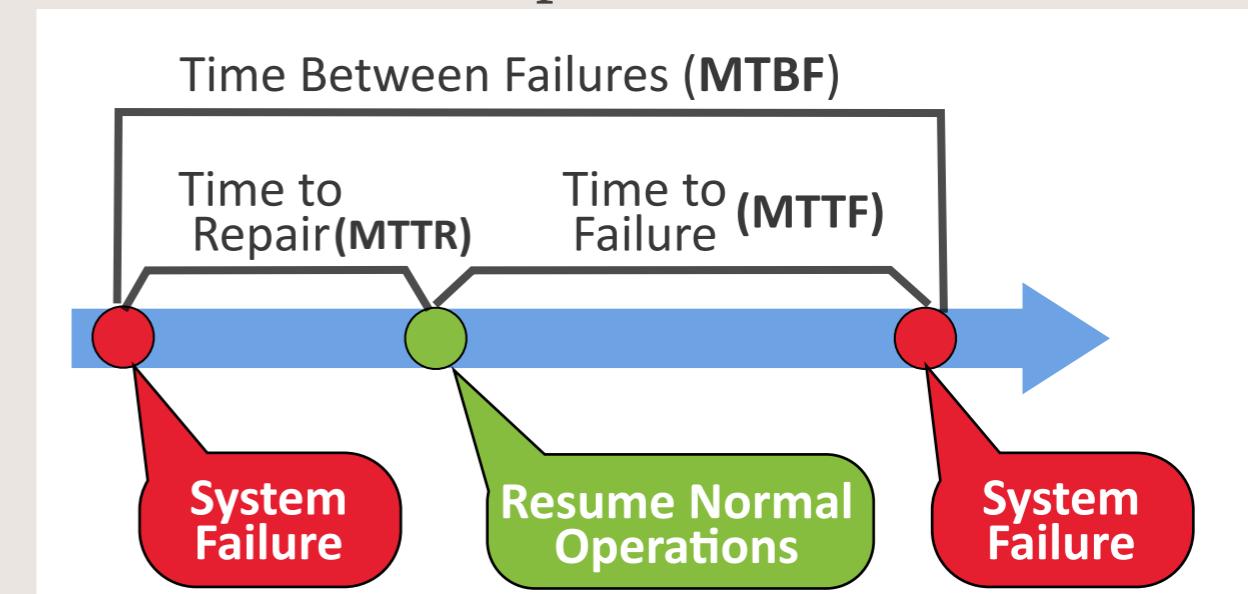
## O Los criterios de aceptación deben incluir propiedades emergentes "cuantificables"

P

- Hay que tener mucho cuidado con criterios de aceptación ambiguos, como "Las peticiones se tienen que servir en un tiempo razonable". Dichas sentencias son imposibles de cuantificar y por lo tanto, imposibles de probar (medir) con precisión

## O Para juzgar en qué grado se satisfacen los criterios de aceptación se utilizan diferentes métricas:

- Para estimar la **fiabilidad** se utilizan pruebas aleatorias basándonos en un perfil operacional. Se utilizan las métricas **MTTF** (Mean Time To Failure), **MTTR** ( Mean Time To Repair), y **MTBF=MTTF+MTTR** (**MTBF**: Mean time between failures)



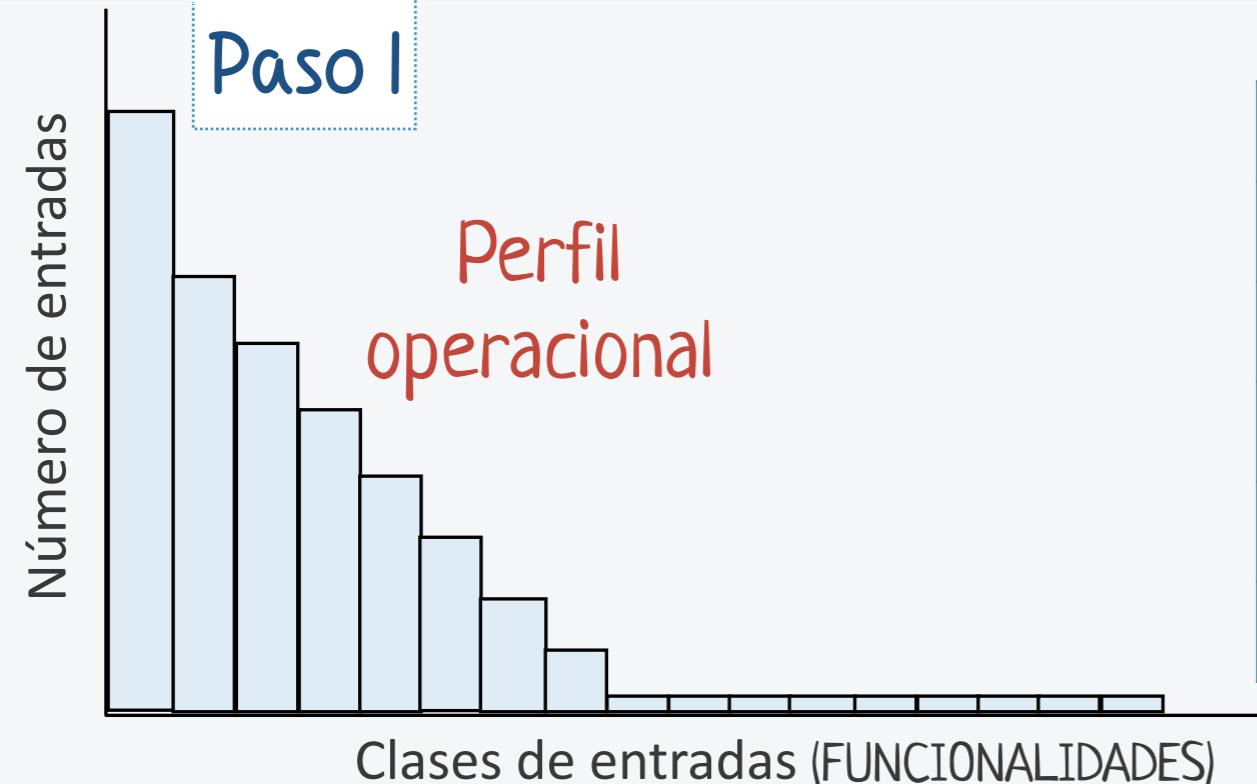
- Para estimar la **disponibilidad** se utiliza la métrica MTTR para medir el "downtime" del sistema. La idea es incluir medidas para minimizar el MTTR
- Para estimar la **mantenibilidad** se utiliza la métrica MTTR (que refleja el tiempo consumido en analizar un defecto correctivo, diseñar la modificación, implementar el cambio, probarlo y distribuirlo)
- La **escalabilidad** del sistema utiliza el número de transacciones (operaciones) por unidad de tiempo. Los sistemas suelen poder incrementar su escalabilidad siempre y cuando no sobrepasen limitaciones de almacenamiento de datos, ancho de banda o velocidad de procesador.

# EJEMPLOS DE PRUEBAS (PROCEDIMIENTOS)

Dependiendo de la propiedad a validar,  
se usan diferentes **métodos**

- Las **pruebas de carga** validan el **rendimiento** de un sistema en términos de "tratar un número específico de usuarios manteniendo un ratio de transacciones" (p.ej. "una petición del sistema se debe tratar en menos de 2 segundos cuando existen 10000 usuarios dentro del sistema")
- Las **pruebas de stress** consisten en "forzar" peticiones al sistema por encima del límite del diseño del software. Por ejemplo si el sistema se ha diseñado para permitir hasta 300 transacciones por segundo, comenzaremos por hacer pruebas con una **carga** de peticiones inferior a 300 e incrementaremos gradualmente la carga hasta sobrepasar los 300 y ver cuándo falla el sistema
  - Las pruebas de stress comprueban la **fiabilidad** y **robustez** del sistema cuando se supera la carga normal (robustez= capacidad de recuperación del sistema ante entradas erróneas u otros fallos)
- Para evaluar la **fiabilidad** de un sistema podemos utilizar lo que se denominan **pruebas estadísticas**, que consisten en:
  1. construir un "perfil operacional" (operational profile), que refleje el uso real del sistema (patrón de entradas). Como resultado se identifican las "clases" de entradas y la probabilidad de ocurrencia para cada clase, asumiendo un uso "normal" (diseño de los casos de prueba)
  2. generar un conjunto de datos de prueba que reflejen dicho perfil operacional
  3. probar dichos datos midiendo el número de fallos y el tiempo entre fallos, calculando la fiabilidad del sistema después de observar un número de fallos estadísticamente significativo

# EJEMPLO DE GENERACIÓN DE PRUEBAS (DISEÑO)



Clase entrada	Distrib. Probab.	Intervalo
C1	50 %	1-49
C2	15 %	50-64
C3	15 %	65-79
C4	15 %	80-94
C5	5 %	95-99

Paso 2

Se generan números aleatorios entre 1 y 99, por ejemplo:

13-94-22-24-45-56-81-19-31-69-45-9-38-21-52-84-86-97...

Se derivan casos de prueba según su distribución de probabilidad:

C1-C4-C1-C1-C1-C2-C4-C1-C1-C3-C1-C1-C1-C2-C4-C4-C5-...



El perfil operacional es la base para el diseño de pruebas emergentes no funcionales

Paso 3

... a continuación deberíamos ejecutar las pruebas midiendo el número de fallos y el tiempo entre fallos

# PRESUMEN DEL PROCESO DE PRUEBAS NO FUNCIONALES

- Escalabilidad, fiabilidad, carga... son ejemplos de propiedades emergentes no funcionales. Todas ellas influyen en el **rendimiento (performance)** del sistema.
  - En general, las propiedades emergentes no funcionales determinan el **RENDIMIENTO** de nuestra aplicación
- Para evaluar el **RENDIMIENTO**, necesitamos realizar las siguientes actividades:
  1. **Identificar** los criterios de **aceptación**: identificar y cuantificar las propiedades emergentes no funcionales que determinan cuál es el rendimiento aceptable para nuestra aplicación (p.e. tiempos de respuesta, fiabilidad, utilización de recursos...)
  2. **Diseñar** los tests: deberemos conocer el patrón de uso de la aplicación (perfil operacional) para que nuestros casos de prueba estén basados en ESCENARIOS reales de nuestra aplicación
  3. **Preparar** el entorno de pruebas: es importante que el entorno de pruebas sea lo más realista posible
  4. **Automatizar** las pruebas: utilizando alguna herramienta software, "grabaremos" los escenarios de prueba, y ejecutaremos los tests
  5. **Analizaremos** los resultados y realizaremos los cambios oportunos para conseguir nuestro objetivo

# CONSIDERACIONES SOBRE EL RENDIMIENTO

- **iii** No hay que dejar las pruebas de rendimiento para el final del proyecto !!!
- Posibles fallos relacionados con el rendimiento pueden conllevar el retocar mucho código
  - Las propiedades emergentes NO funcionales están condicionadas fundamentalmente por la ARQUITECTURA del sistema
  - Normalmente la arquitectura del sistema se determina en las primeras fases del desarrollo!! Eso significa que cambiar la arquitectura puede implicar cambiar "todo".
  - Recuerda que el coste de reparar un error siempre es proporcional al intervalo de tiempo transcurrido desde que se produjo el error hasta que éste es detectado.
- Minimizaremos los problemas derivados de la validación de las propiedades emergentes no funcionales mediante una buena **estrategia** de pruebas combinada con una arquitectura software que considere el rendimiento desde el inicio del desarrollo
  - Si usamos una metodología de desarrollo iterativa las iteraciones iniciales del proyecto son para construir prototipos exploratorios y comprobar todos los requisitos no funcionales.

# JMETER



Ver <http://jmeter.apache.org/usermanual/index.html>

- P O Apache JMeter (<http://jmeter.apache.org/>) es una herramienta de escritorio 100% Java diseñada para medir el rendimiento mediante pruebas de carga
- P O Permite múltiples hilos de ejecución concurrente, procesando diversos y diferentes patrones de petición
  - JMeter permite trabajar con muchos tipos distintos de aplicaciones. Para cada una de ellas proporciona un **Sampler** o Muestreador, para hacer las correspondientes peticiones

**EDITOR**  
Aquí "escribimos" nuestros tests

El "codigo" consiste en una serie de **ELEMENTOS** que pueden anidarse formando un "**ARBOL**"

El elemento TEST PLAN contendrá nuestro "driver"

Cada **TIPO** de elemento tiene asociado un **ICONO DIFERENTE!!!**

**Apache JMeter (5.6.3)**

Test Plan

Name: Test Plan

Comments:

Podemos cambiar el **NOMBRE** mostrado en el editor de cualquier elemento.  
El icono de cada elemento indica de qué **TIPO** es ese elemento

User Defined Variables

Name:	Value

Detail Add Add from Clipboard Delete Up Down

Run Thread Groups consecutively (i.e. one at a time)  
 Run tearDown Thread Groups after shutdown of main thread  
 Functional Test Mode (i.e. save Response Data and Sampler Results)

Selecting Functional Test Mode may adversely affect performance.

Add directory or jar to classpath

Library

**PANEL DE CONFIGURACIÓN** del elemento seleccionado en el Editor:  
Cada elemento tiene diferentes opciones de configuración



# P JMeter: PLAN DE PRUEBAS (I)

[https://jmeter.apache.org/usermanual/test\\_plan.html](https://jmeter.apache.org/usermanual/test_plan.html)

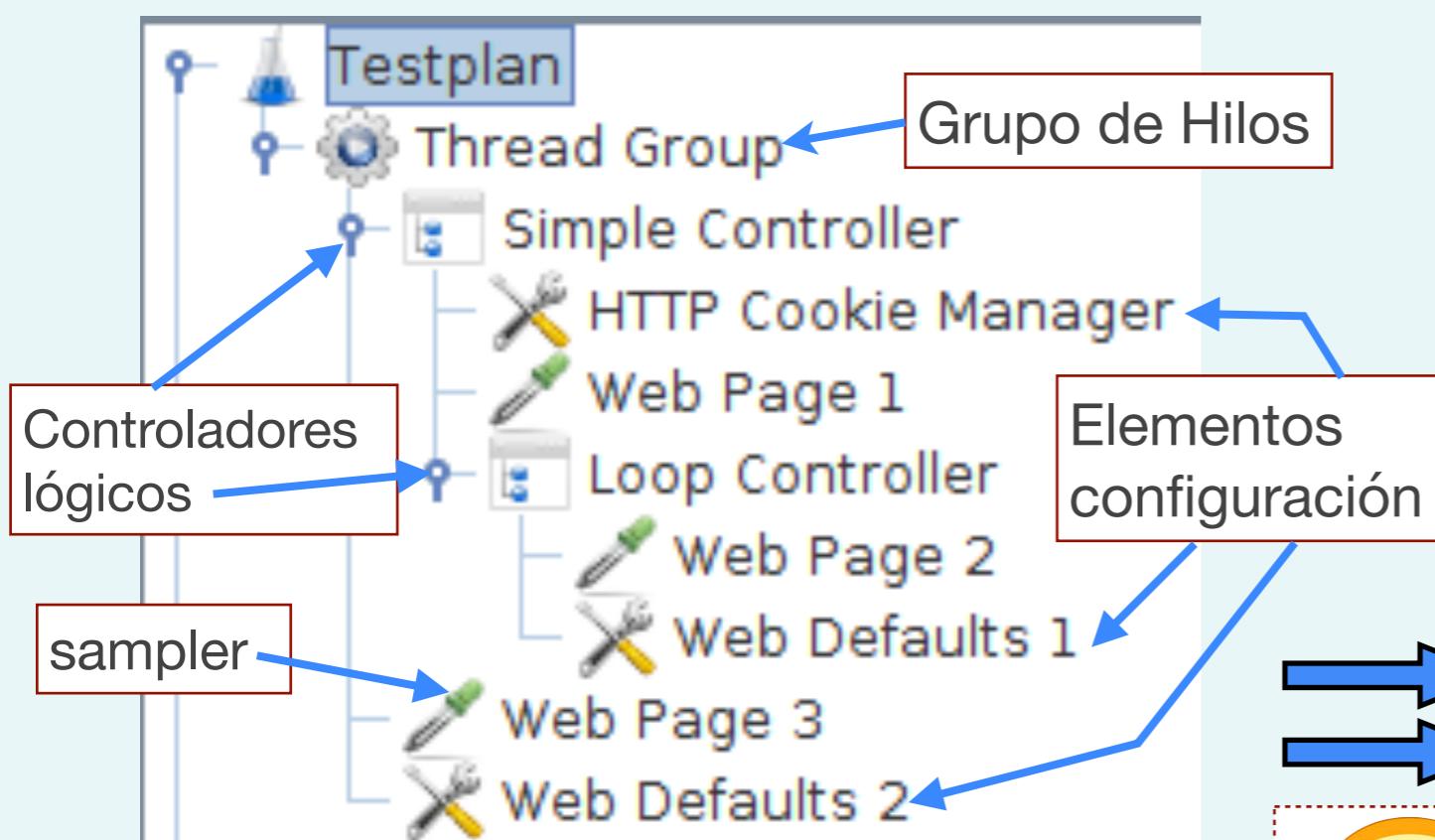
- Un plan de pruebas JMeter describe una serie de "pasos" (acciones) que JMeter realizará cuando se ejecute el plan

- Un **plan de pruebas** está formado por:

- Uno o más grupos de hilos (**Thread Groups**)
- Controladores lógicos (**Logic Controllers**)
- Samplers, Listeners, Timers, Assertions, Pre-Processors, Post-Processors y
- Elementos de Configuración (**Configuration Elements**)

Cualquier test (driver) requiere como mínimo:  
Test Plan + Thread Group + samplers

Nos centraremos en los elementos que hemos marcado con una flecha azul



El orden de ejecución de los elementos de un plan es el siguiente:

1. Configuration elements ←
2. Pre-Processors
3. Timers ←
4. Sampler ←
5. Post-Processors (unless SampleResult is null)
6. Assertions (unless SampleResult is null)
7. Listeners (unless SampleResult is null)



**CADA** sampler puede verse "afectado" por **VARIOS** Timers, Conf.Elements, Assertions...

<http://www.guru99.com/jmeter-element-reference.html>

# JMETER: HILOS DE EJECUCIÓN



[https://jmeter....#Thread\\_Group](https://jmeter....#Thread_Group)



- Un hilo de ejecución es el punto de partida de cualquier plan de pruebas

## Thread Group

Name: Thread Group

Comments: Representa un conjunto de usuarios

Action to be taken after a Sampler error

- Continue
- Start Next Thread Loop
- Stop Thread
- Stop Test
- Stop Test Now

### Thread Properties

Number of Threads (users): 50

Ramp-up period (seconds): 100

Loop Count:  Infinite 1

Same user on each iteration

Delay Thread creation until needed

Specify Thread lifetime

Duration (seconds):

Startup delay (seconds):

**TODOS** los Controladores y Samplers  
deben estar anidados en un Thread Group

CADA hilo ejecuta  
COMPLETAMENTE el plan  
de forma independiente de  
otros hilos

Cada hilo  
representa a  
un usuario



Podemos ejecutar el grupo  
de hilos un cierto número de  
veces o de forma indefinida  
(bucle "infinito")

**RAMP-UP** (periodo de subida): sirve para  
que los hilos se creen de forma gradual.  
Esto permite comprobar cómo rinde el  
servidor conforme crece la carga.

Por ejemplo, si el periodo de subida es de  
100 segundos y el número de hilos es 50,  
significa que el servidor tardará 100  
segundos en crear los 50 hilos, es decir, un  
nuevo hilo cada 2 segundos



P

# JMETER: SAMPLERS(I)

S  
el sampler HTTP REQUEST  
realiza peticiones http

- Los Samplers (muestreadores) envían peticiones a un servidor. Ejemplos de samplers: HTTP request, FTP request, JDBC Request,... Se ejecutan en el orden en el que aparecen en el plan

P

**HTTP Request**

Name: **HTTP Request**

Comments: Esto es una petición HTTP

Basic Advanced

Web Server

Protocol [http]:  Server Name or IP:  Port Number:

HTTP Request

GET Path:  Content encoding:

Redirect Automatically  Follow Redirects  Use KeepAlive  Use multipart/form-data

Parameters Body Data Files Upload

Send Parameters With the Request:

Name:	Value	URL Encode?	Content-Type	Include Equals?
-------	-------	-------------	--------------	-----------------

Podemos configurar diferentes propiedades en un HTTP Request.  
Adicionalmente, también podemos configurar algunas propiedades añadiendo uno (o varios) Configuration Elements en el plan

Detail Add Add from Clipboard Delete Up Down

Podemos usar cualquier **NOMBRE** pero ten en cuenta que dicho nombre será el que se mostrará en el plan. Deberíamos elegir nombres representativos de las acciones asociadas a cada elemento

Los comentarios se **MOSTRARÁN** cuando "situemos" el cursor por encima de ese sampler en el plan de pruebas

Flow Control Action

**HTTP Request**

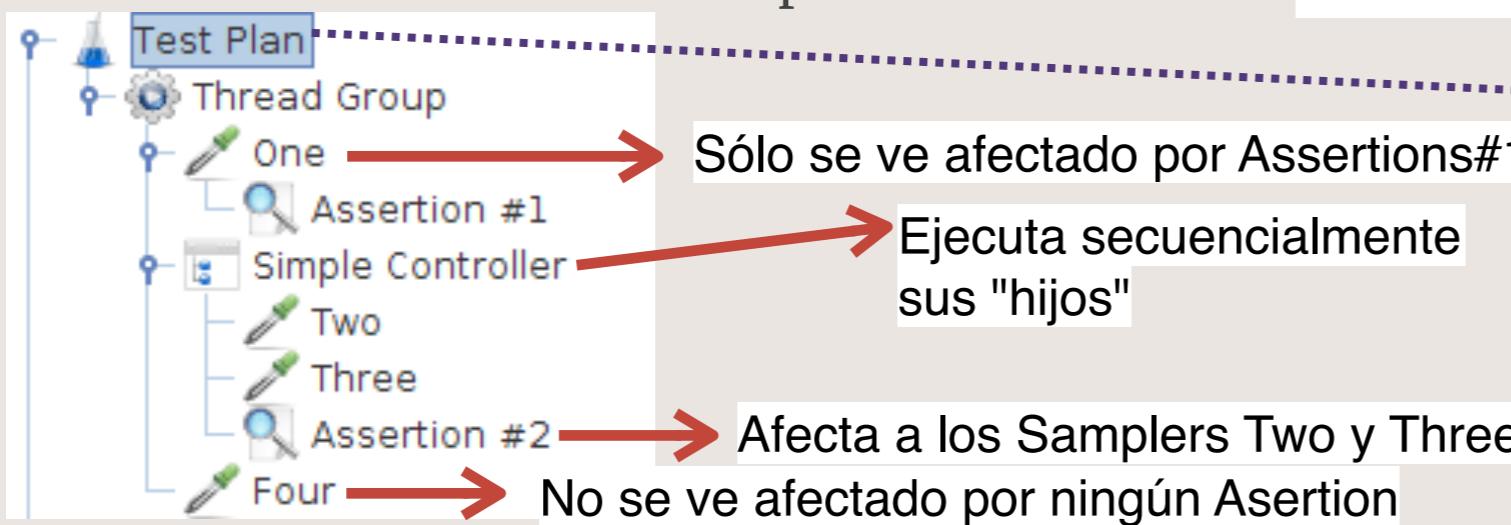
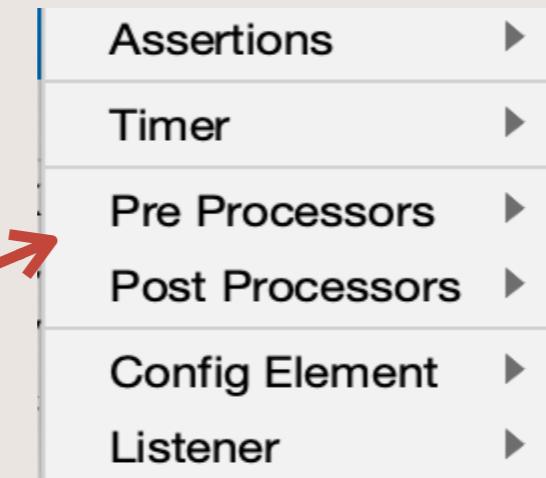
Debug Sampler  
JSR223 Sampler  
AJP/1.3 Sampler  
Access Log Sampler  
BeanShell Sampler  
Bolt Request  
FTP Request  
GraphQL HTTP Request  
JDBC Request  
JMS Point-to-Point  
JMS Publisher  
JMS Subscriber  
JUnit Request  
Java Request  
LDAP Extended Request  
LDAP Request  
Mail Reader Sampler  
OS Process Sampler  
SMTP Sampler  
TCP Sampler

Podemos usar muchos tipos de **SAMPLERS!!!**

# JMETER: SAMPLERS(II)

**S** Los samplers se ejecutan en el **ORDEN** en el que aparecen en el árbol).

- No todos los elementos pueden anidarse dentro de otros. Un sampler puede anidarse en un GRUPO DE HILOS y/o en un CONTROLADOR.
- Su ejecución puede verse "afectada" por otros elementos (p.ej. timers, configuration elements, controllers, Pre/Post Processors y assertions).
- En un sampler podemos anidar los siguientes elementos:
- Los elementos anidados en un sampler sólo "afectan" a dicho sampler



## IMPORTANTE:

El orden de ejecución del plan NO depende del orden en el que aparezcan en el árbol los diferentes elementos!!!!



Por ejemplo, en el siguiente Test Plan:

- Controller
  - Post-Processor 1
  - Sampler 1
  - Sampler 2
  - Timer 1
  - Assertion 1
  - Pre-Processor 1
  - Timer 2
  - Post-Processor 2

El **ORDEN** de ejecución es:

Pre-Processor 1	→	Pre-Processor 1
Timer 1		Timer 1
Timer 2		Timer 2
<b>Sampler 1</b>		<b>Sampler 1</b>
Post-Processor 1		Post-Processor 1
Post-Processor 2		Post-Processor 2
Assertion 1		Assertion 1
Pre-Processor 1		Pre-Processor 1
Timer 1		Timer 1
Timer 2		Timer 2
<b>Sampler 2</b>		<b>Sampler 2</b>
Post-Processor 1		Post-Processor 1
Post-Processor 2		Post-Processor 2
Assertion 1		Assertion 1

## ORDEN de ejecución:

One  
Assertions #1  
Two  
Assertions #2  
Three  
Assertions #2  
Four

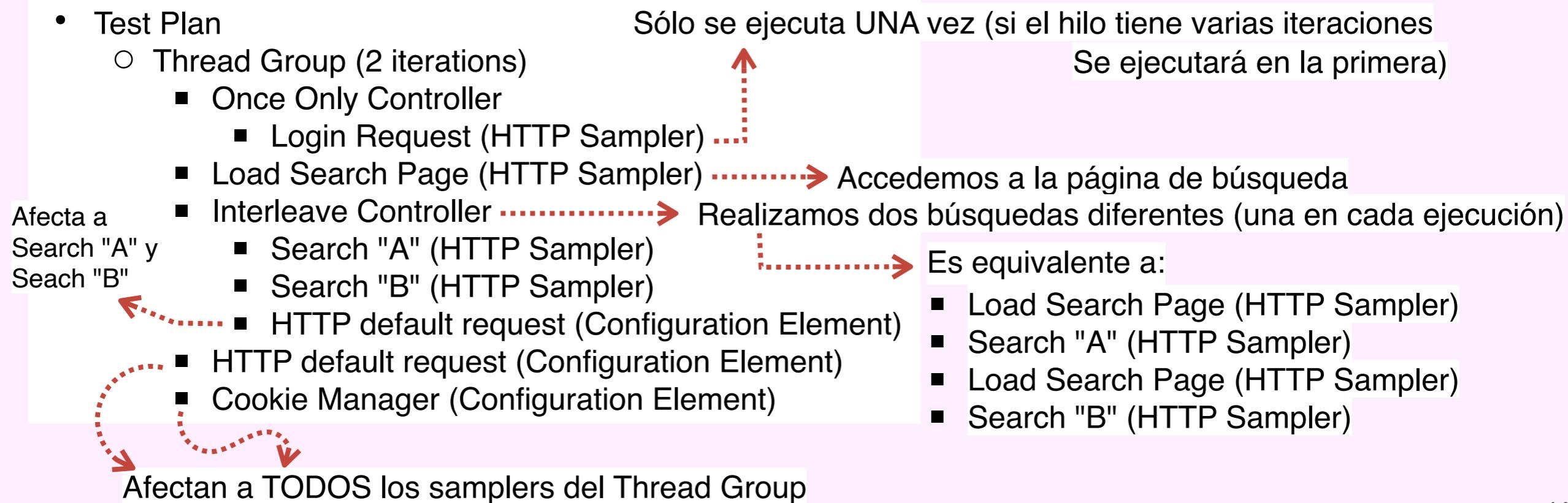
# P JMeter: CONTROLADORES LÓGICOS (I)

## Tipos de controladores

[https://jmeter.apache.org/usermanual/test\\_plan.html#logic\\_controller](https://jmeter.apache.org/usermanual/test_plan.html#logic_controller)

- Determinan el orden en el que se procesan los Samplers. Actúan sobre sus elementos hijo
- Ejemplos de controladores
  - **Simple controller**: No tiene efecto sobre sus elementos hijo. Simplemente sirve para "agrupar" dichos elementos.
  - **Loop controller**: Itera sobre sus elementos hijo un cierto número de veces
  - **Once only controller**: Indica a JMeter que sus elementos hijos deben ser procesados UNA ÚNICA vez por hilo de ejecución. Si este controlador es hijo de un loop controller, entonces se ejecuta sólo en la primera iteración de ese bucle
  - **Interleave controller**: ejecutará uno de sus subcontroladores o samplers en cada iteración del bucle de pruebas, alterándose secuencialmente a lo largo de la lista

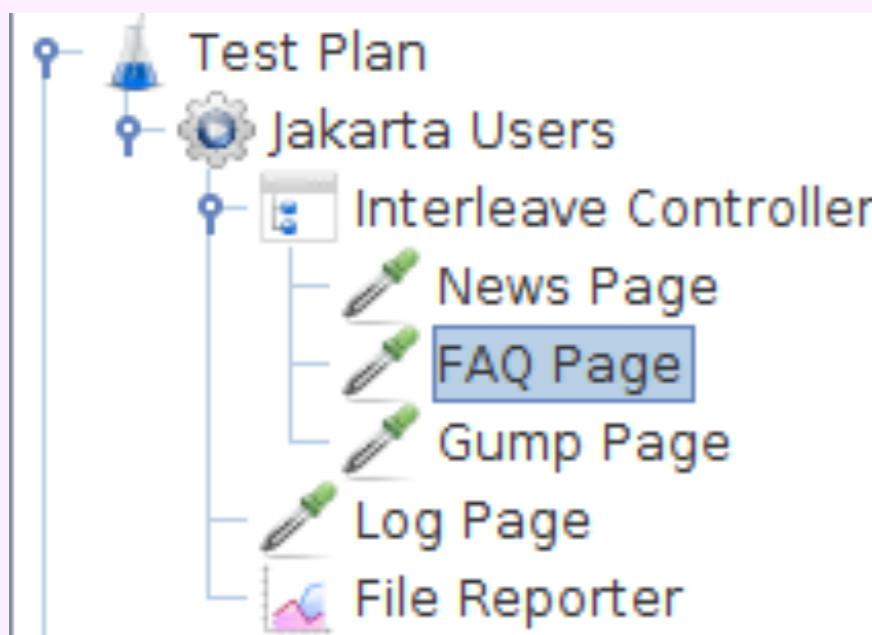
Ejemplo: supongamos el siguiente Test Plan



# JMETER: CONTROLADORES LÓGICOS (II)

[http://jmeter.apache.org/usermanual/component\\_reference.html#Interleave\\_Controller](http://jmeter.apache.org/usermanual/component_reference.html#Interleave_Controller)

**Ejemplo:** supongamos que el grupo de hilos está formado por 2 hilos, y 5 iteraciones



Loop Iteration	Each JMeter Thread Sends These HTTP Requests
1	News Page
1	Log Page
2	FAQ Page
2	Log Page
3	Gump Page
3	Log Page
4	Because there are no more requests in the controller, JMeter starts over and sends the first HTTP Request, which is the News Page
4	Log Page
5	FAQ Page
5	Log Page



Cada hilo realiza 10 peticiones

- El controlador Interleave Controller sólo "controla" sus samplers hijos (es decir: "News Page", "Faqs Page", y "Gump Page")
- El sampler "Log Page" se ejecutará siempre, independiente de la iteración de cada hilo
- El Listener "File Reporter" recopila los resultados de la ejecución de todos los elementos del grupo de hilos ("afecta" a todos los elementos de su mismo nivel y siempre se ejecuta en último lugar, aunque en el árbol aparezca "escrito" antes!!!)

# JMETER: ELEMENTOS DE CONFIGURACIÓN

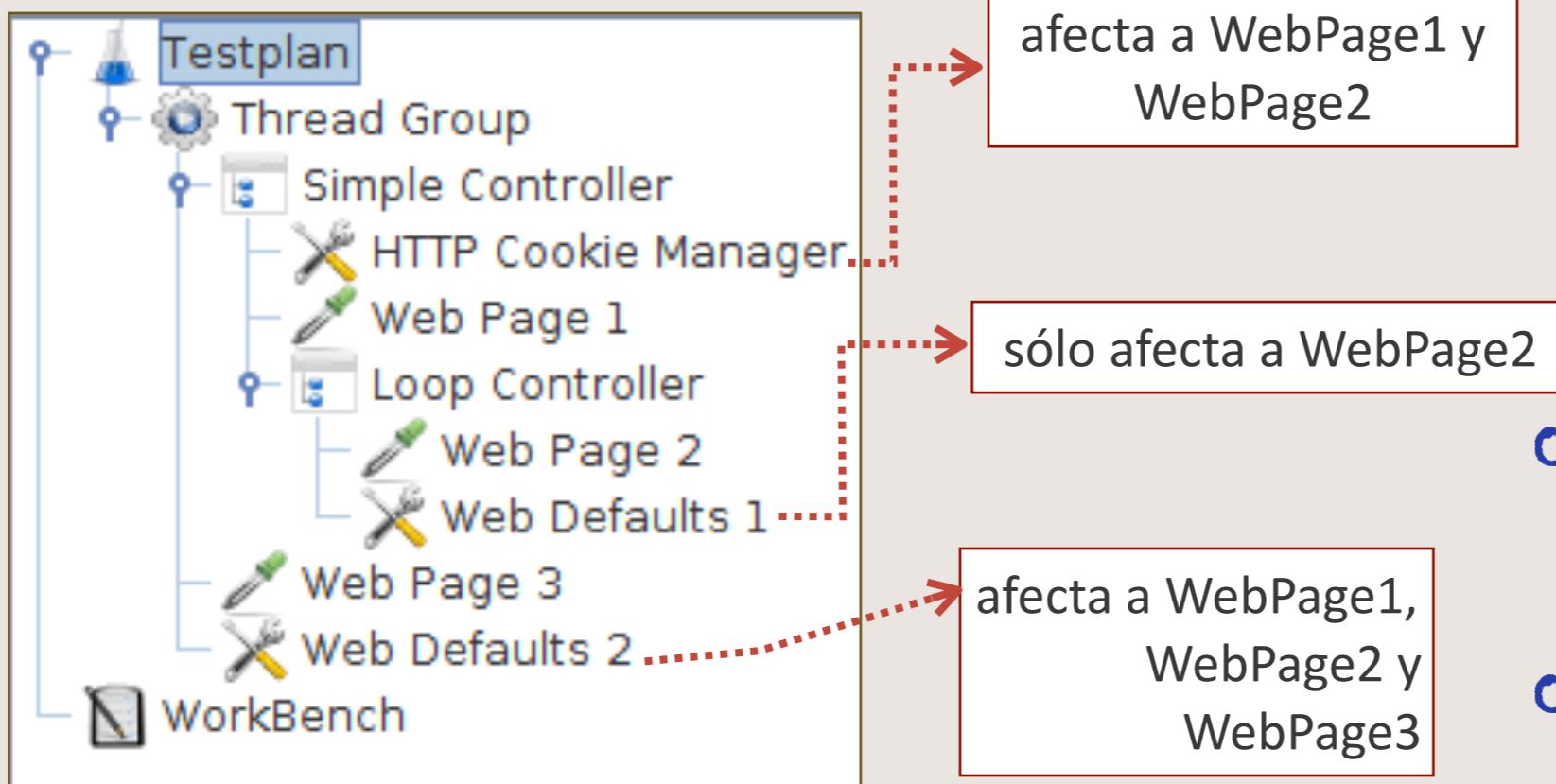
Tipos



[https://jmeter.apache.org/usermanual/test\\_plan.html#config\\_elements](https://jmeter.apache.org/usermanual/test_plan.html#config_elements)

[http://jmeter.apache.org/usermanual/test\\_plan.html](http://jmeter.apache.org/usermanual/test_plan.html)

- "Trabajan" conjuntamente con un sampler. Si bien no realizan peticiones, pueden modificarlas
- Un elemento de configuración es accesible sólo dentro de la rama del árbol (y sub-ramas) en la que se sitúa el elemento
- Un elemento de configuración dentro de una rama del árbol tiene mayor preferencia que el mismo elemento (mismo tipo de elemento) en una rama padre (con excepción de los Elementos Configuration DEFAULT, los cuales son combinados "merged")



CSV Data Set Config
HTTP Header Manager
HTTP Cookie Manager
HTTP Cache Manager
HTTP Request Defaults
Bolt Connection Configuration
Counter
DNS Cache Manager
FTP Request Defaults
HTTP Authorization Manager
JDBC Connection Configuration
Java Request Defaults
Keystore Configuration
LDAP Extended Request Defaults
LDAP Request Defaults
Login Config Element
Random Variable
Simple Config Element
TCP Sampler Config
User Defined Variables

- **HTTP Request Defaults** es útil por ejemplo cuando varios samplers comparten la misma url (o parte de ella)
- **HTTP Cookie Manager** permite almacenar y enviar cookies como si fuese un navegador. También podremos añadir nuevas cookies



# TIMERS

P

P

- Por defecto JMeter envía las peticiones sin realizar ninguna pausa entre las mismas. En este caso nuestro test podría "saturar" el servidor al enviar demasiadas peticiones en intervalos cortos de tiempo!!

- Los temporizadores permiten introducir pausas **antes** de realizar **cada** una de las peticiones de **cada** hilo

- Podemos utilizar varios tipos de temporizadores:

- Constant timer**: Retrasa cada petición de usuario la misma cantidad de tiempo
- Uniform random timer**: Introduce pausas aleatorias (con la misma probabilidad)
- Gaussian random timer**: Introduce pausas según una determinada distribución (gausiana)

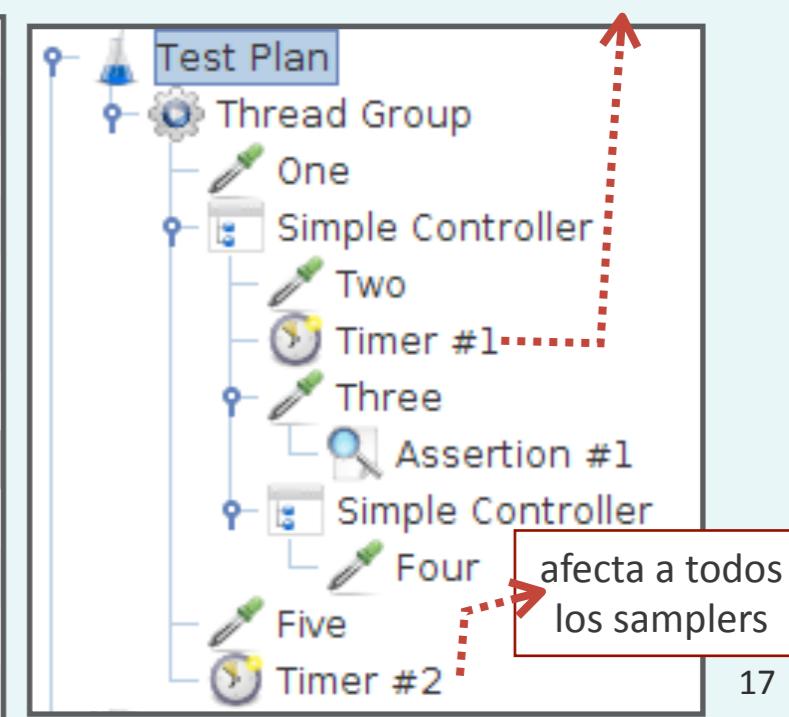
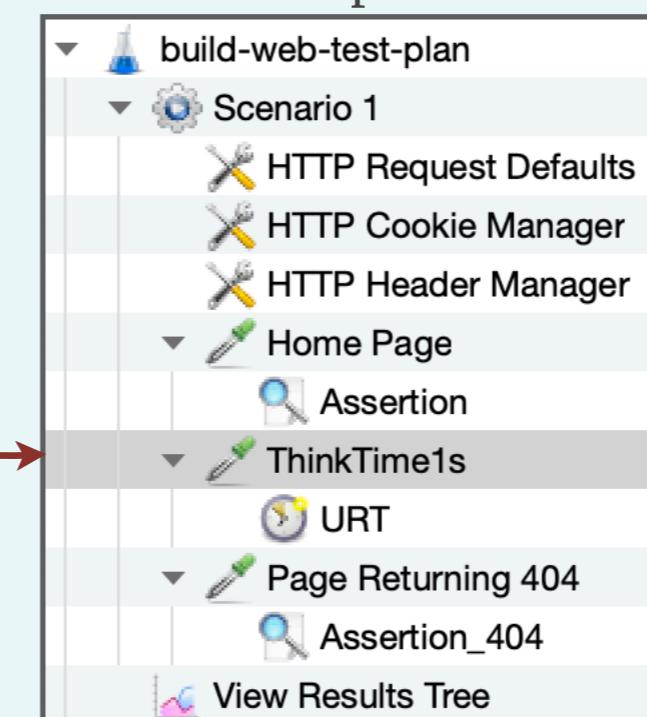
- Si hay varios timers en el mismo "ámbito" (nivel) del sampler se aplicarán todos ellos. En caso contrario, no se aplicará ninguno

- Si queremos que un timer afecte a un sólo sampler, debemos añadirlo como su hijo

- Si queremos que un timer se aplique después de un sampler:

- Lo añadiremos al siguiente sampler, o
- Lo añadiremos como hijo de un **Flow Control Action Sampler**

Siempre se procesan ANTES de cada sampler

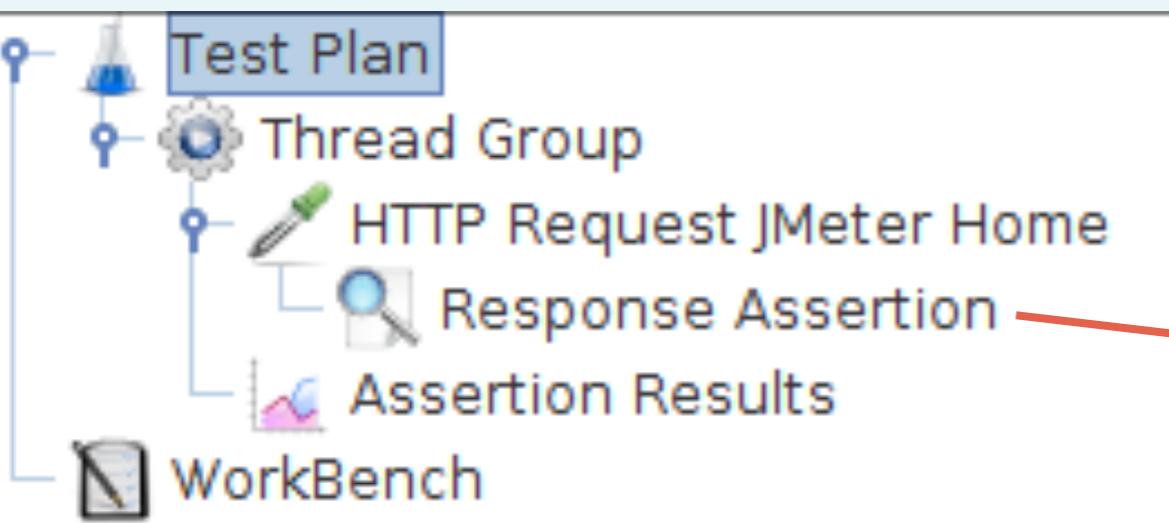




# JMETER: ASERCIIONES

P

S



- Las aserciones permiten hacer afirmaciones sobre las respuestas recibidas del servidor que se está probando. Podemos añadir aserciones a cualquier sampler. Se trata de probar que la aplicación devuelve el resultado esperado

The screenshot shows the 'Response Assertion' configuration dialog. It includes fields for 'Name' (set to 'Response Assertion'), 'Comments', 'Apply to' (set to 'Main sample only'), 'Field to Test' (set to 'Text Response'), 'Pattern Matching Rules' (set to 'Substring'), and 'Patterns to Test' (empty). A red box highlights the text 'Cualquier driver SIEMPRE debe incluir aserciones!!!' and an orange cartoon character with a pointing finger and the text 'Don't FORGET!'.

**Response Assertion**

Name: Response Assertion

Comments:

Apply to:

Main sample and sub-samples  Main sample only  Sub-samples only  JMeter Variable Name to use [ ]

Field to Test

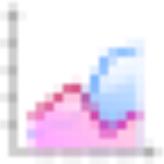
Text Response  Response Code  Response Message  Response Headers  
 Request Headers  URL Sampled  Document (text)  Ignore Status  
 Request Data

Pattern Matching Rules

Contains  Matches  Equals  Substring  Not  Or

Patterns to Test

Add Add from Clipboard Delete



# JMETER: LISTENERS (I)

[http://jmeter.apache.org/usermanual/component\\_reference.html#listeners](http://jmeter.apache.org/usermanual/component_reference.html#listeners)

- Los **listeners** se utilizan para ver y/o almacenar en el disco los resultados de las peticiones realizadas. Proporcionan acceso a la información que JMeter va acumulando sobre los casos de prueba a medida que se ejecutan los tests
  - TODOS los listeners guardan los MISMOS datos; la única diferencia es la forma en que presentan dichos datos en la pantalla
  - Ejemplos de listeners: Simple Data Writer, Aggregate Graph, Graph results, Aggregate Report, Summary Report, Response Time Graph, Assertion results,...

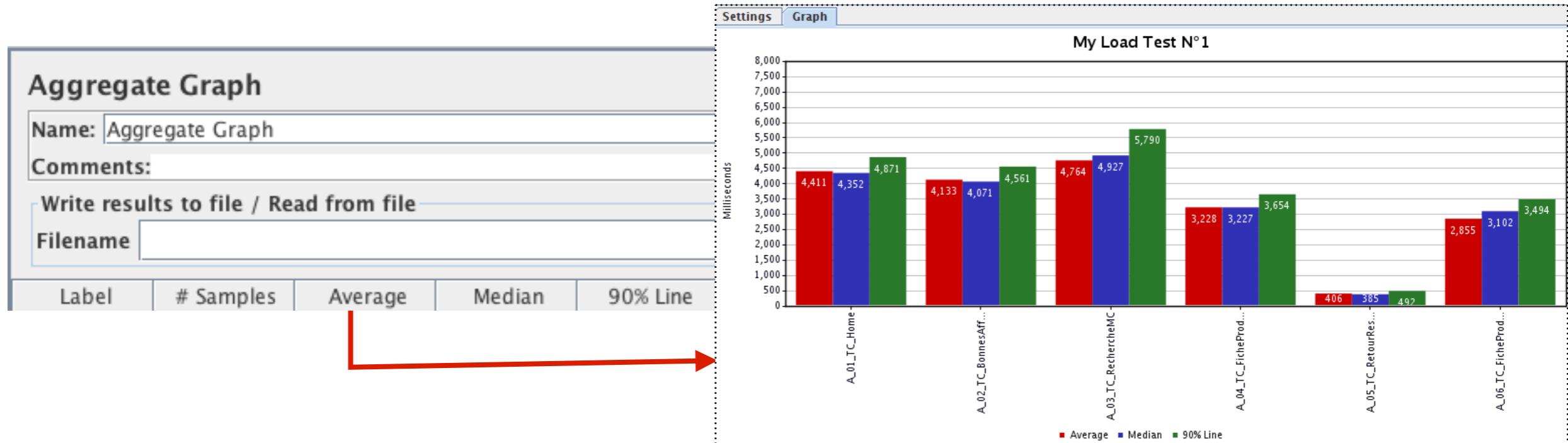
**Simple Data Writer**

Name: Simple Data Writer

Comments:

Write results to file / Read from file

Filename  Browse... Log/Display Only:  Errors  Successes



# JMETER: LISTENERS (II)



ver <http://www.guru99.com/jmeter-performance-testing.html>

### Aggregate Report

Name: Aggregate Report  
Comments:  
Write results to file / Read from file  
Filename  Browse... Log/Display Only:  Errors  Successes

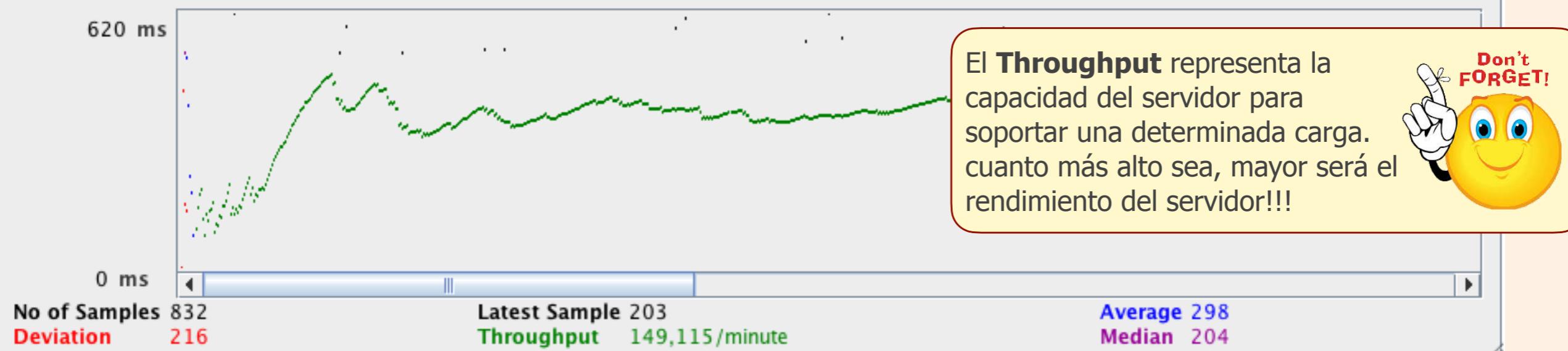
Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	Received KB/sec	Sent KB/sec
HTTP Request-3	80	222	224	317	104	353	0.00%	1.5/sec	2.62	0.45
HTTP Request-1	80	231	233	336	108	354	0.00%	1.5/sec	10.49	0.13
HTTP Request-2	80	233	233	329	102	354	1.25%	1.5/sec	5.23	0.28
TOTAL	240	229	229	330	102	354	0.42%	4.4/sec	18.18	0.86

Include group data  Save Table Header

### Graph Results

Name: Graph Results  
Comments:  
Write results to file / Read from file  
Filename  Browse... Log/Display Only:  Errors  Successes

Graphs to Display  Data  Average  Median  Deviation  Throughput



# SOBRE LOS DATOS REGISTRADOS POR JMETER

El tiempo se calcula en milisegundos

- Para cada muestra (sampler), JMeter calcula:
  - **# Samples** – Número de muestras con la misma etiqueta
  - **Average** – Tiempo medio de respuesta (en milisegundos)
  - **Median** – The median is the time in the middle of a set of results. 50% of the samples took no more than this time; the remainder took at least as long.
  - **Línea de 90%** (percentil): 90% of the samples took no more than this time. The remaining samples took at least as long as this
  - **Min** – Tiempo mínimo de respuesta para las muestras con la misma etiqueta
  - **Max** – Tiempo máximo de respuesta para las muestras con la misma etiqueta
  - **% Error** – Porcentaje de peticiones con errores
  - Rendimiento (**Throughput**) – número de peticiones por segundo/minuto/hora.  
La unidad de tiempo se elige en función de que el valor visualizado sea como mínimo 1.
    - \* Se calcula como:  $(\text{nº peticiones}) / (\text{tiempo total de respuesta})$   
El tiempo total de respuesta se mide desde el instante en que se envía la primera petición hasta que se recibe la última respuesta (aquí están incluido cualquier intervalo de espera como por ejemplo los temporizadores).
  - **Kb/sec** – rendimiento expresado en Kilobytes por segundo

# CONSEJOS JMETER



P

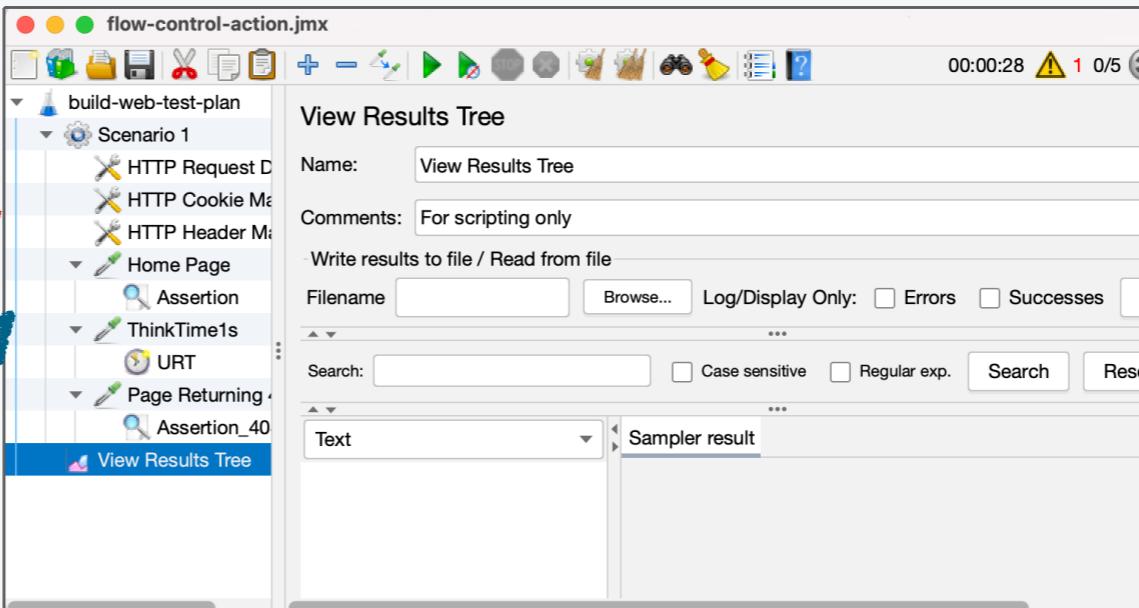
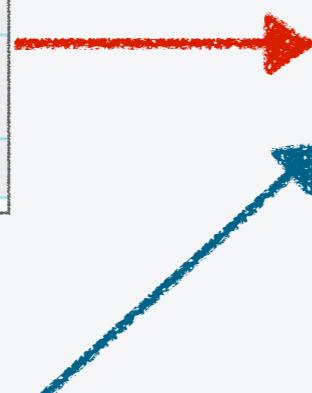
- Utiliza escenarios de prueba significativos, y construye planes de prueba que prueben situaciones representativas del mundo real
  - Los casos de uso ofrecen un punto de partida ideal. A partir de ellos debes generar un perfil operacional
- Asegúrate de ejecutar JMeter en una máquina distinta a la del sistema a probar.
  - Esto previene que JMeter influya sobre los resultados de las pruebas
- El proceso de pruebas es un proceso científico. Todas las pruebas se deben realizar bajo condiciones completamente controladas
  - Si estás trabajando con un servidor compartido, primero comprueba que nadie más está realizando pruebas de carga contra la misma aplicación web.
- Asegúrate de que dispones de ancho de banda en la estación que ejecuta JMeter
  - La idea es probar el rendimiento de la aplicación y el servidor, y no la conexión de la red.
- Utiliza diferentes instancias de JMeter ejecutándose en diferentes máquinas para añadir carga adicional al servidor
  - Esta configuración suele ser necesaria para realizar pruebas de stress. JMeter puede controlar las instancias JMeter de las otras máquinas y coordinar la prueba
- Deja una prueba JMeter ejecutarse durante largos períodos de tiempo, posiblemente varios días o semanas
  - Estarás probando la disponibilidad del sistema y resaltando las posibles degradaciones en el rendimiento del servidor debido a una mala gestión de los recursos

Y AHORA VAMOS AL LABORATORIO...

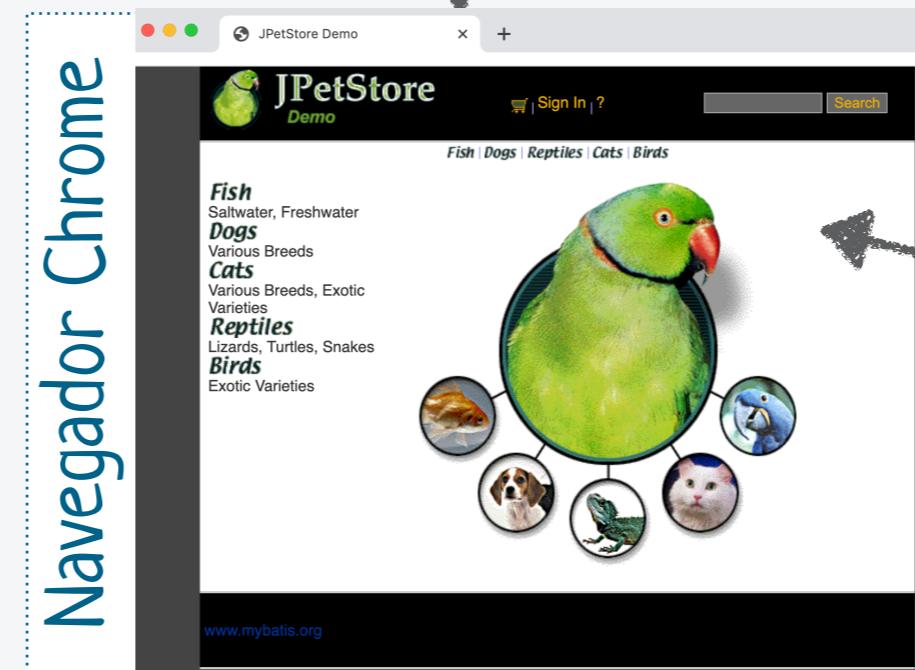
Vamos a implementar tests de aceptación (para validar propiedades emergentes NO funcionales) sobre una aplicación web con JMeter



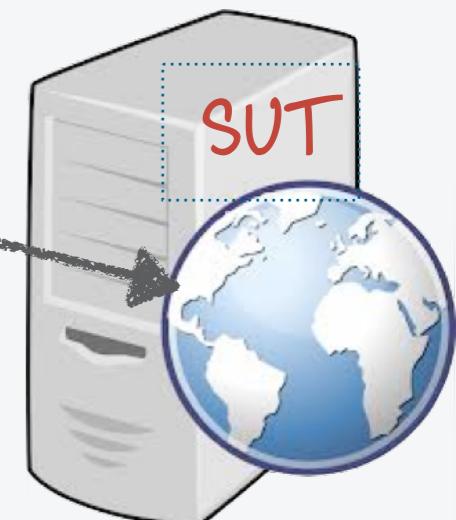
Camino	Datos Entrada	Resultado Esperado
C1	d1=... d2=... ... ..	r1
..		
CM	d1=... d2=... ...	rM



usaremos JMeter (el driver estará implementado en un fichero .jmx)



## Servidor web



# REFERENCIAS BIBLIOGRÁFICAS



- Software testing and quality assurance. Kshirasagar Naik & Priyadarshi Tripathy. Wiley. 2008
  - Capítulos 8 y 15
- Página oficial JMeter:
  - <http://jmeter.apache.org/usermanual/index.html>
- Otras referencias interesantes:
  - <http://www.guru99.com/jmeter-performance-testing.html>
  - [What is Throughput in Performance Testing?](#)
  - <http://jmeterperftest.blogspot.com.es>
  - [Performance Testing Concepts – What are Concurrent Users?](#)