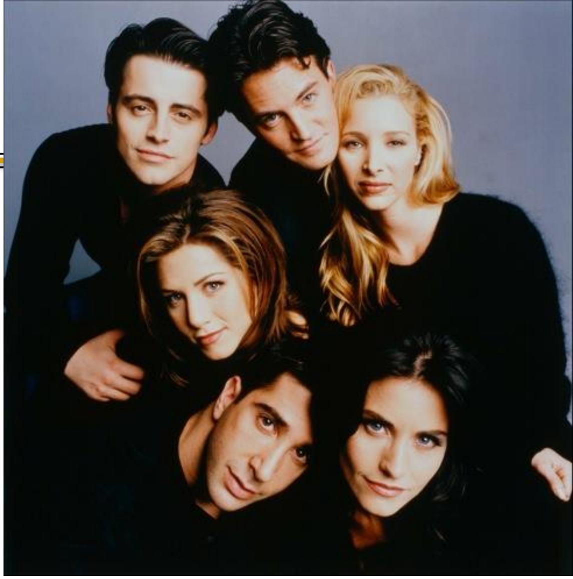


Legacy Web Application Architectures

Common Gateway Interface (CGI)

Server-side Scripting/Includes (SSI)

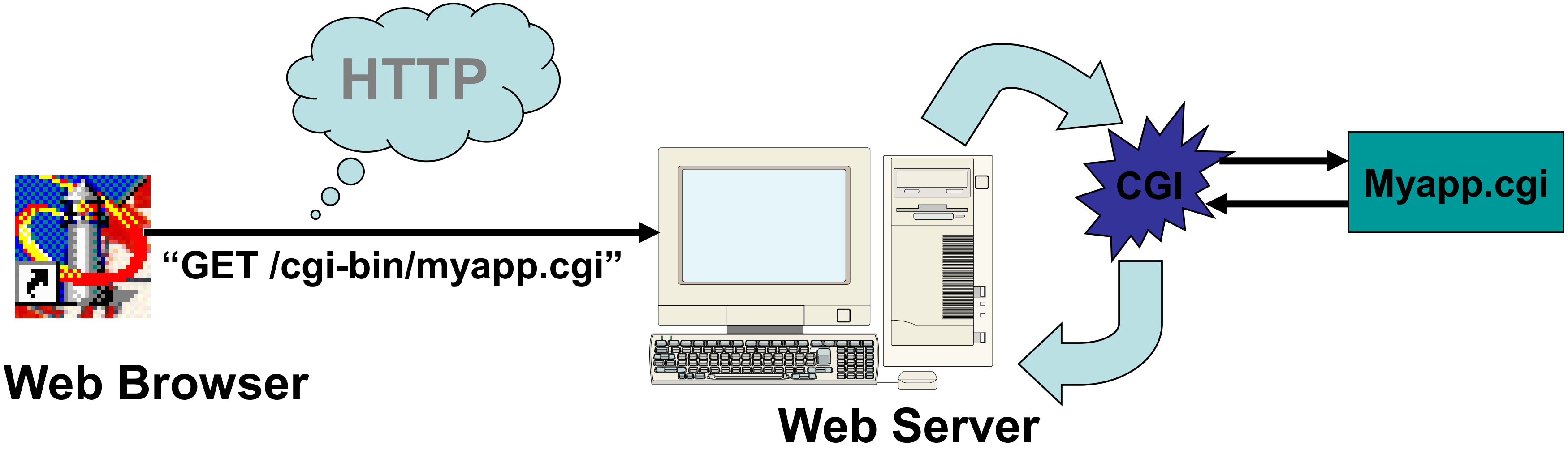
Implicit Invocation

Components & Containers

Common Gateway Interface

CGI is an interface

- A CGI program is not a server-side "script"
  - (The executed program may be a script, but this is different)
- A CGI program is just that - an executable program that resides on the server
- "CGI" refers to the standard way in which data is passed to the executing application, and received from the application when it terminates



Simple CGI Examples

Test1.cgi

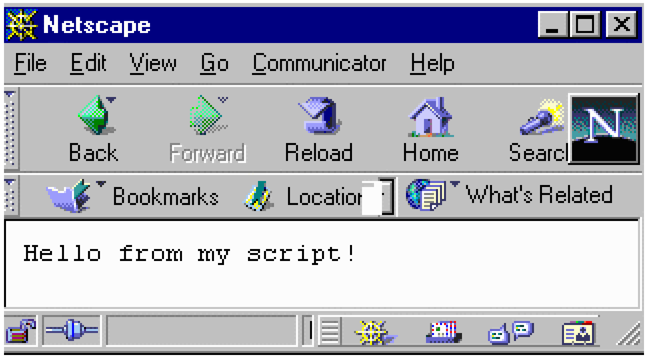
```
#!/bin/sh
echo "Content-type: text/plain"
echo ""
echo "Hello from my script!"
```

Test1.pl

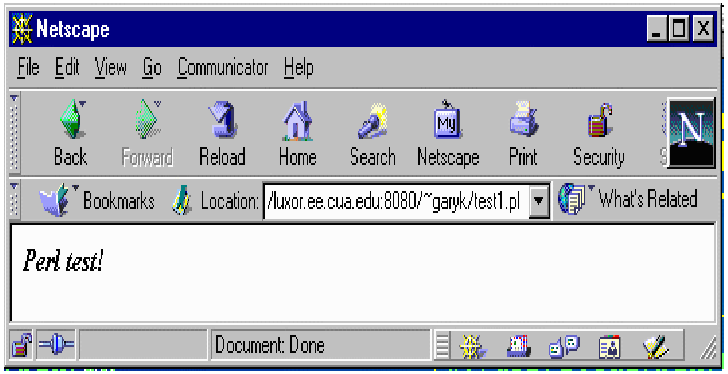
```
#!/usr/local/bin/perl
print "Content-type: text/html\n\n";
print "<b><i>Perl test!\n</i></b>\n";
```

test1.c

```
#include <stdio.h>
int main()
{
 printf("Content-type: text/html\n\n");
 printf("<b>Sample output from C
        program!</b>");
}
```


Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Searc
Bookmarks Locatior What's Related

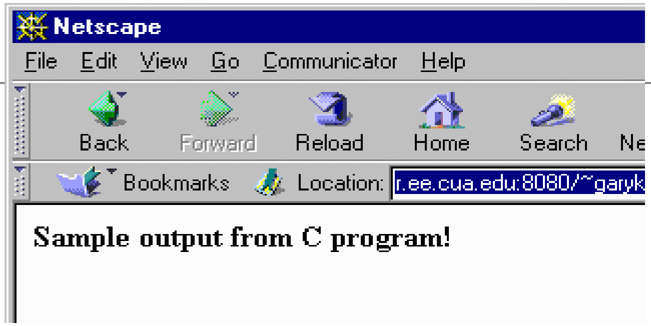
```
Hello from my script!
```


Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security
Bookmarks What's Related

Location: /luxor.ee.cua.edu:8080/~garyk/test1.pl

Perl test!

Document: Done


Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Ne
Bookmarks

Location: r.ee.cua.edu:8080/~garyk

Sample output from C program!

Simple CGI Examples: explained

CGI execution

- Web server must be configured to execute your scripts
- Server spawns application in a separate OS process
  - Operating System overhead
  - stdin/stdout hooked up to CGI

CGI output

- Standard output (stdout) is now captured by the CGI
- Any stdout output statements (echo, print, printf) have their output redirected by the CGI to the output stream of the Web server's return socket!

CGI Header Format

- Note the output of "Content-type: text/html" followed by 2 blank lines
- Whatever output is sent back by a Web server via an HTTP response must be *typed*!

CGI: Passing Input to a Program (GET)

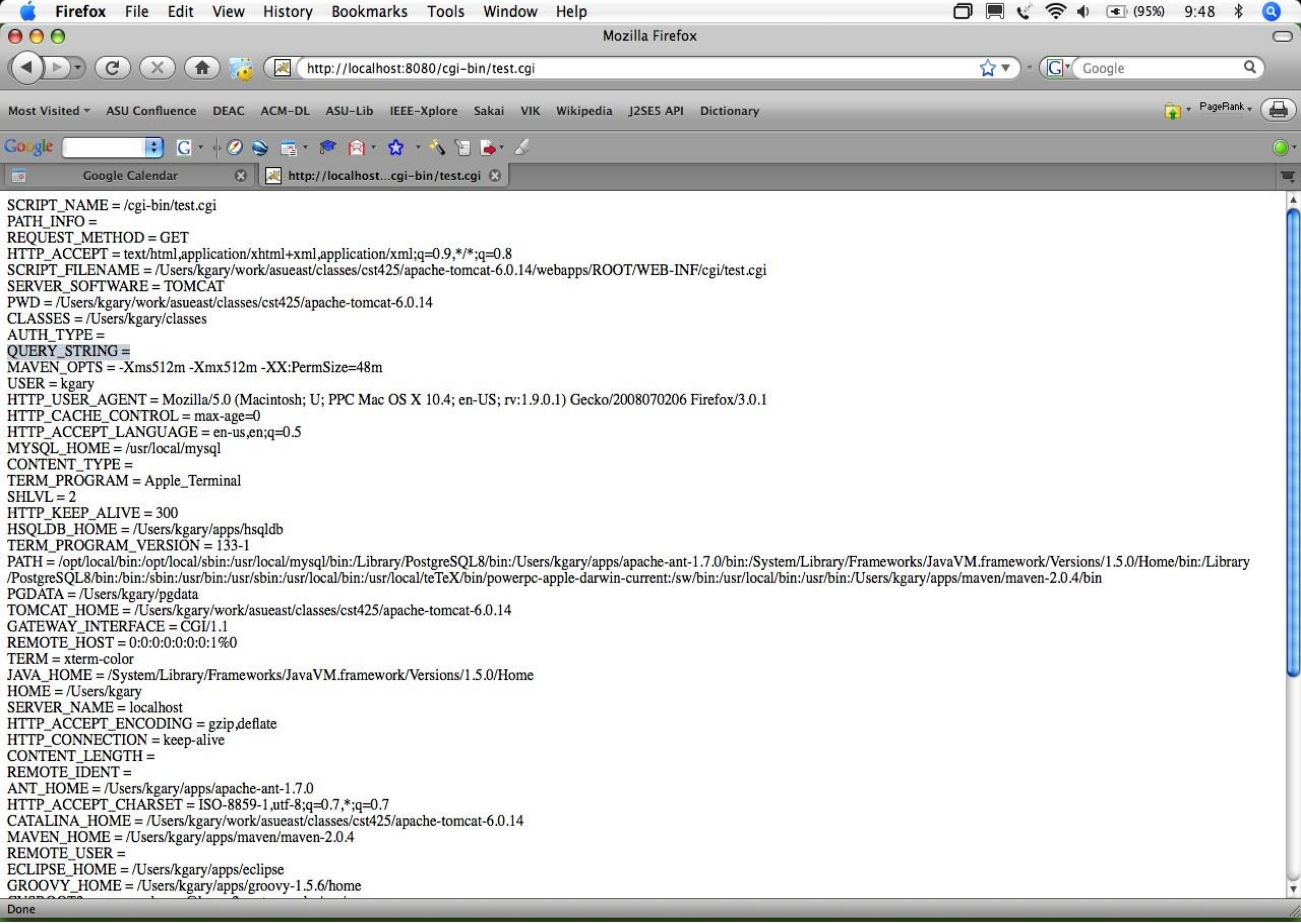
CGI uses environment variables to pass data to your application program

Environment Variables: a quick recap

- Every OS shell has environment variables
  - DOS: type “set”
  - Unix: type “env”
- environment variables customize a user’s environment
- environment variables may be read by applications - so they provide a means of passing data to applications!

Example:

```
#!/usr/local/bin/perl
print “Content-type: text/html\n\n”;
while (($key, $val) = each %ENV) {
  print “$key = $val<br>\n”;
```


Firefox File Edit View History Bookmarks Tools Window Help
(95%) 9:48
Mozilla Firefox
http://localhost:8080/cgi-bin/test.cgi
Google
Most Visited ASU Confluence DEAC ACM-DL ASU-Lib IEEE-Xplore Sakai VIK Wikipedia J2SE5 API Dictionary
PageRank
Google
Google Calendar
http://localhost...cgi-bin/test.cgi
Done

```
SCRIPT_NAME = /cgi-bin/test.cgi
PATH_INFO =
REQUEST_METHOD = GET
HTTP_ACCEPT = text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
SCRIPT_FILENAME = /Users/kgary/work/asueast/classes/cst425/apache-tomcat-6.0.14/webapps/ROOT/WEB-INF/cgi/test.cgi
SERVER_SOFTWARE = TOMCAT
PWD = /Users/kgary/work/asueast/classes/cst425/apache-tomcat-6.0.14
CLASSES = /Users/kgary/classes
AUTH_TYPE =
QUERY_STRING =
MAVEN_OPTS = -Xms512m -Xmx512m -XX:PermSize=48m
USER = kgary
HTTP_USER_AGENT = Mozilla/5.0 (Macintosh; U; PPC Mac OS X 10.4; en-US; rv:1.9.0.1) Gecko/2008070206 Firefox/3.0.1
HTTP_CACHE_CONTROL = max-age=0
HTTP_ACCEPT_LANGUAGE = en-us,en;q=0.5
MYSQL_HOME = /usr/local/mysql
CONTENT_TYPE =
TERM_PROGRAM = Apple_Terminal
SHLVL = 2
HTTP_KEEP_ALIVE = 300
HSQLDB_HOME = /Users/kgary/apps/hsqldb
TERM_PROGRAM_VERSION = 133-1
PATH = /opt/local/bin:/opt/local/sbin:/usr/local/mysql/bin:/Library/PostgreSQL8/bin:/Users/kgary/apps/apache-ant-1.7.0/bin:/System/Library/Frameworks/JavaVM.framework/Versions/1.5.0/Home/bin:/Library
/PostgreSQL8/bin:/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/teTeX/bin/powerpc-apple-darwin-current:/sw/bin:/usr/local/bin:/usr/bin:/Users/kgary/apps/maven/maven-2.0.4/bin
PGDATA = /Users/kgary/pgdata
TOMCAT_HOME = /Users/kgary/work/asueast/classes/cst425/apache-tomcat-6.0.14
GATEWAY_INTERFACE = CGI/1.1
REMOTE_HOST = 0:0:0:0:0:0:0:1%0
TERM = xterm-color
JAVA_HOME = /System/Library/Frameworks/JavaVM.framework/Versions/1.5.0/Home
HOME = /Users/kgary
SERVER_NAME = localhost
HTTP_ACCEPT_ENCODING = gzip,deflate
HTTP_CONNECTION = keep-alive
CONTENT_LENGTH =
REMOTE_IDENT =
ANT_HOME = /Users/kgary/apps/apache-ant-1.7.0
HTTP_ACCEPT_CHARSET = ISO-8859-1,utf-8;q=0.7,*;q=0.7
CATALINA_HOME = /Users/kgary/work/asueast/classes/cst425/apache-tomcat-6.0.14
MAVEN_HOME = /Users/kgary/apps/maven/maven-2.0.4
REMOTE_USER =
ECLIPSE_HOME = /Users/kgary/apps/eclipse
GROOVY_HOME = /Users/kgary/apps/groovy-1.5.6/home
```

CGI: Passing Input to a Program (GET)

Note the value of environment variable “QUERY_STRING”

- Contains whatever information is URL encoded after the ? on the URL
- Parameter values must be “marshalled”
  - The string has to be parsed to obtain the parameter values
  - QUERY_STRING may have:
    - “+” substitute for spaces
    - “=“ to denote attribute value pairs
    - “&” separates multiple values
  - The values are by default strings, so they must be type-converted to whatever the target type is in the application
  - Same goes for output - when writing back to stdout, you have to “marshal” your data back into strings
  - Because marshalling is such a pain, typically you will find libraries for each language that are tailored for marshalling data in and out of CGI programs.

CGI: Passing Input to a Program (POST)

POST method

- The QUERY_STRING environment variable is set only when the HTTP request uses the GET method
- A second method, the POST method, is also available
  - input data is not URL-encoded
  - input data is passed in via standard input (stdin)
  - special HTTP header CONTENT-LENGTH specifies the length in bytes of the input stream
  - POST actually sends the data from your Web browser to your Web server as a separate HTTP request, whereas GET passed the data with the originating request
  - preferred mechanism
    - special parsing routines are not required
    - the input interface is not exposed

So Have CGIs died a painful death?

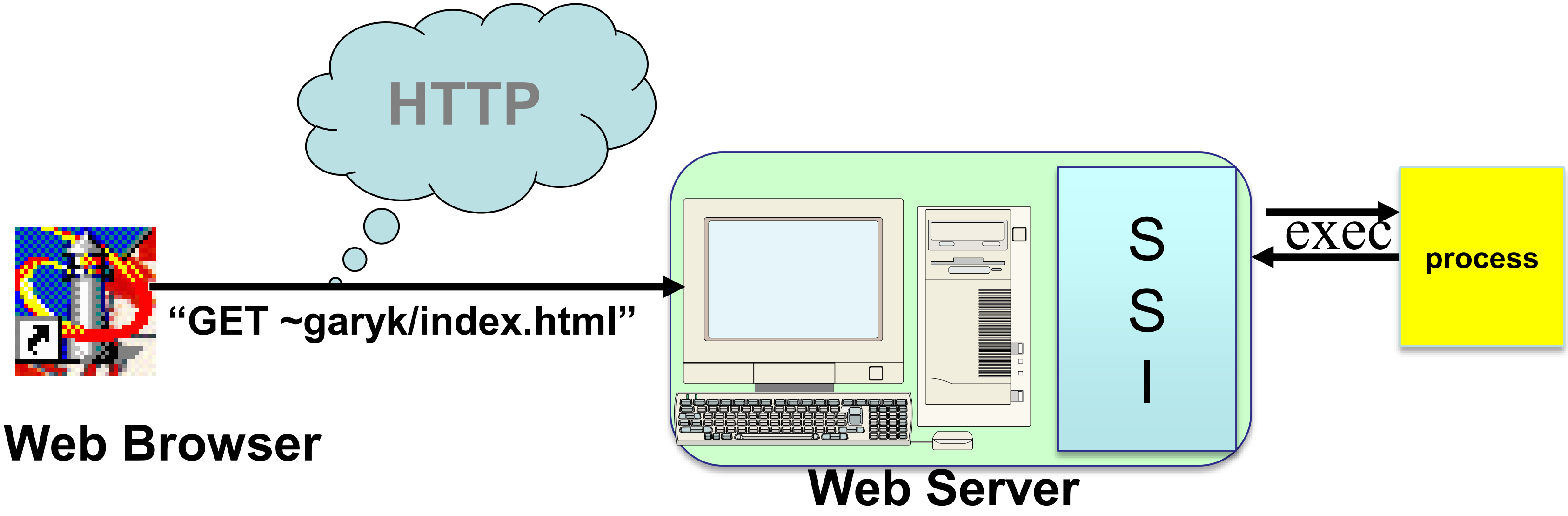
"*Yesterday's shiny new code is tomorrow's legacy code*"

- HTTP was originally a document-serving protocol – ONLY
- CGI was introduced as a hack
  - But wow! Now web pages were dynamic and interactive!
  - So, folks started writing a lot of CGIs
- Then, better architectures came about
  - Netscape *server* **NSAPI** – server-side "plugin"
  - **Servlets** – persistent, server-side component model
  - Apache's **mod_*** - in-process "implicit invocation" style
- Some folks got it, hacks started appearing
  - OpenMarket created a pseudo-standard early but many http servers did not adopt until scalability issues were understood
  - Architecture is fairly similar to servlet (not exactly)
  - Result: today many http servers support FastCGI

Web Server-side Technologies: SSI

Server-side Includes are scripts

- SSI executes scripts embedded in comments in an HTML page
  - Web server retrieves file
  - Web server parses file, looking for scripting tags
  - Server interprets scripts, inserting text in place of scripts
- Unlike CGI, SSIs execute within the server process
  - However, directives exist to spawn a process



SSI History and Directives

SSIs quickly became popular in the early/mid-90s

- NCSA’s httpd server added SSIs in the early 90s
- Several proprietary extensions popped up
- Apache addedsupport for SSI, and also extends the set of directives (“extended SSI”, or “XSSI”)

Sample Directives

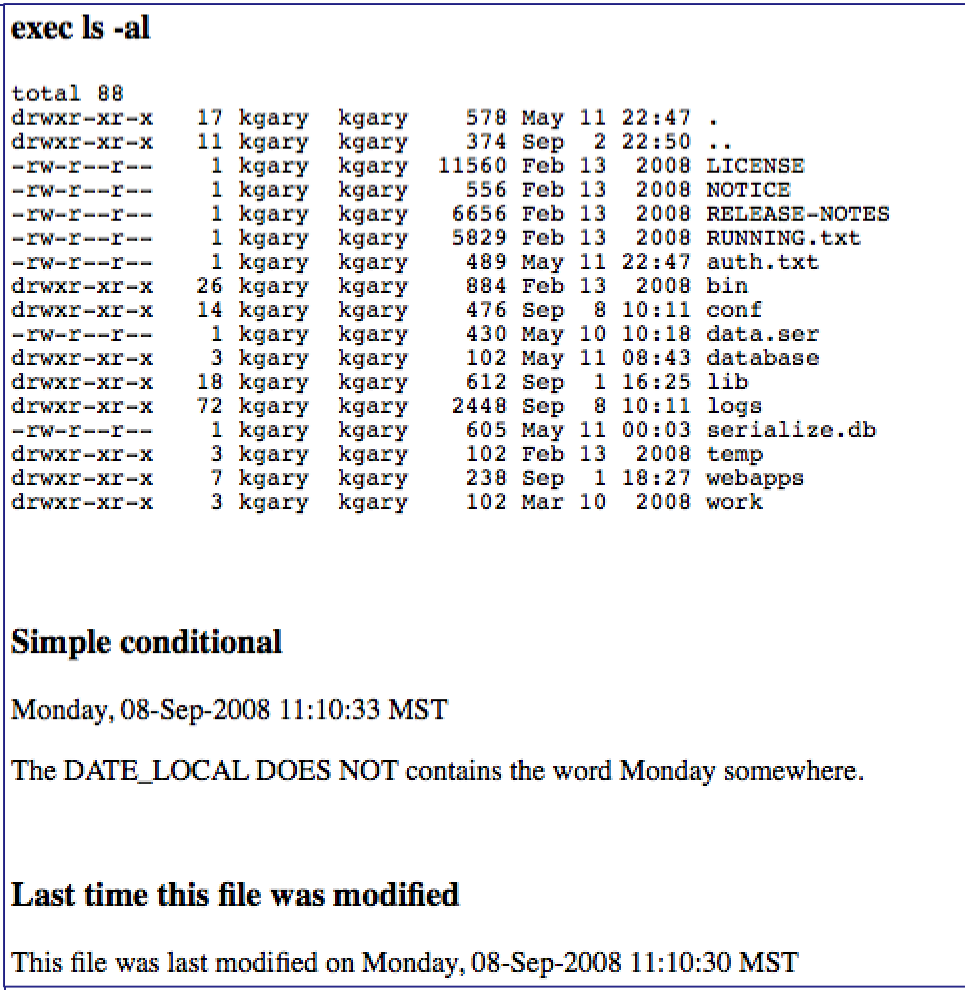
- #config – for formatting output of other SSI directives
- #echo – prints a message to standard out
- #flastmod – last modification time of file
- #exec [cmd] – forks a process and executes a cmd
- #include – includes the contents of another file or CGI
- #set – set the value of a variables (alias)
- #if/elif/else/endif – allows for conditional blocks

SSI has access to limited number of env vars like a CGI

- Query string (not decoded), client agent (HTTP_USER_AGENT)

Server-side Scripting: Example

```
<H3>exec ls -al</H3>
<pre><!--#exec cmd="ls -al”--></pre>
<h3>Simple conditional</h3>
    <!--#echo var="DATE_LOCAL"-->
    <!--#if expr="\"$DATE_LOCAL\" = /Monday/"
    -->
        <P>The DATE_LOCAL contains the word
    Monday somewhere.</P>
    <!--#else-->
        <P>The DATE_LOCAL DOES NOT contains
    the word Monday somewhere.</P>
    <!--#endif-->
<H3>Last time this file was modified</H3>
This file was last modified on <!--#flastmod
    file="ssitest.shtml”-->
```



exec ls -al

```
total 88
drwxr-xr-x   17 kgary  kgary    578 May 11 22:47 .
drwxr-xr-x   11 kgary  kgary    374 Sep  2 22:50 ..
-rw-r--r--    1 kgary  kgary  11560 Feb 13  2008 LICENSE
-rw-r--r--    1 kgary  kgary    556 Feb 13  2008 NOTICE
-rw-r--r--    1 kgary  kgary   6656 Feb 13  2008 RELEASE-NOTES
-rw-r--r--    1 kgary  kgary   5829 Feb 13  2008 RUNNING.txt
-rw-r--r--    1 kgary  kgary    489 May 11 22:47 auth.txt
drwxr-xr-x   26 kgary  kgary    884 Feb 13  2008 bin
drwxr-xr-x   14 kgary  kgary    476 Sep  8 10:11 conf
-rw-r--r--    1 kgary  kgary    430 May 10 10:18 data.ser
drwxr-xr-x    3 kgary  kgary    102 May 11 08:43 database
drwxr-xr-x   18 kgary  kgary    612 Sep  1 16:25 lib
drwxr-xr-x   72 kgary  kgary   2448 Sep  8 10:11 logs
-rw-r--r--    1 kgary  kgary    605 May 11 00:03 serialize.db
drwxr-xr-x    3 kgary  kgary    102 Feb 13  2008 temp
drwxr-xr-x    7 kgary  kgary    238 Sep  1 18:27 webapps
drwxr-xr-x    3 kgary  kgary    102 Mar 10  2008 work
```

Simple conditional

Monday, 08-Sep-2008 11:10:33 MST

The DATE_LOCAL DOES NOT contains the word Monday somewhere.

Last time this file was modified

This file was last modified on Monday, 08-Sep-2008 11:10:30 MST

Web Server-side Technologies: Server Scripting

Advantages

- Principal benefit is to take care of simple dynamic tasks that really don’t require a full-blown application component
  - e.g. Outputting current date, URI info, last modified, etc.
- Server can use information in the request to take specific actions
- Keeps clients “thin” and servers “fat”
- Platform dependence not an issue for client side
- Security: end-user never downloads server-side information

Disadvantages

- Lack of server-side scripting standards
- Degrades server performance
  - Only parse files with special extensions: “.asp”, “.shtml”, “.php”
  - Requires additional server-side modules and configuration
- Difficult to maintain good SoC
- Possible security risks if site is improperly configured

SSI/Server-side Scripting Summary

SSIs are like CGIs in that they…

- are a dated technology for web application development
- are potentially insecure
- may potentially reinforce poor design practices

SSIs are unlike CGIs in that they…

- are not run as distinct processes
- still have a role for very simple dynamic tasks

Enabling SSIs in Apache (httpd.conf):

1. Add Includes to your `Options` directive
2. Then *either:*
   1. `AddType text/html .shtml` AND `AddOutputFilter INCLUDES .shtml`
   2. OR add `XBitHack on` and change any .html file permissions you want parsed by Apache to executable (`chmod +x blah.html`)
      - Nope, you can't do this (easily) in Windows

So have SSIs died a slow painful death?

"*Sometimes tight coupling is just easier and faster*"

- In the mid-90s as folks started realizing CGI wasn't scalable, some entrepreneurs started creating custom HTTP servers that touted proprietary server-side scripts
  - These were just NCSA's server with some hacks to parse
  - But, an improvement!
    - No separate process
    - No awkward passing of env vars and query strings
  - Of course, there were also issues
    - No standard 3rd-party language support
    - Now your business logic was jumbled into the page
      - Cut-and-paste code everywhere
    - These one-offs tended to be easy to hack
      - Your kimono (business logic) was open for the world to see!

Wikipedia’s list of Server-side scripting languages

ASP (*.asp, *.aspx)

C via CGI (*.c, *.csp)

ColdFusion Markup Language (*.cfm)

Java via JavaServer Pages (*.jsp)

JavaScript using Server-side JavaScript (*.ssjs, *.js)

Lua (*.lp *.op)

Perl CGI (*.cgi, *.ipl, *.pl)

PHP (*.php)

Python via Django (*.py)

Ruby, e.g. Ruby on Rails (*.rb, *.rbw)

SMX (*.smx)

Lasso (*.lasso)

WebDNA (*.dna,*.tpl)

Progress WebSpeed (*.r,*.w)

- *PHP, ASP[x], Python, and Ruby dominate. JSPs don’t belong here!*

So it does live!
Not the worst thing if you are in a hurry! PHP in ~80% of websites

What about Apache's mod_* handlers

Apache uses an *implicit invocation* architecture style

→ *What is implicit invocation?*

- An architecture style described by Garlan and Shaw [1996]
- The Hollywood Principle: "*Don't call us we'll call you*"
  - Like a Hollywood agent, you register interest in events
  - Apache modules register interest via *Handlers* (AddHandler)
  - Apache maintains a linked list of registered handlers (modules) and invokes these based on its configuration
- The Apache Core has special properties/priorities in this chain
- A feature of interest in the core is how it manages its process multi-processing module space w.r.t. new requests:
  - **Prefork MPM** uses multiple child processes with one thread each and each process handles one connection at a time.
  - **Worker MPM** uses multiple child processes with many threads each. Each thread handles one connection at a time.
  - *Prefork is thread-safe but uses more memory than worker*

Apache mod_* handlers

Mod_cgi stills spawns that extra process

Mod_php, mod_python, etc. create a binary image of the dynamic language interpreter in the Apache process space

- Less overhead
- Not as robust

Still, generally

Prefer mod_X

Over running X

As a CGI

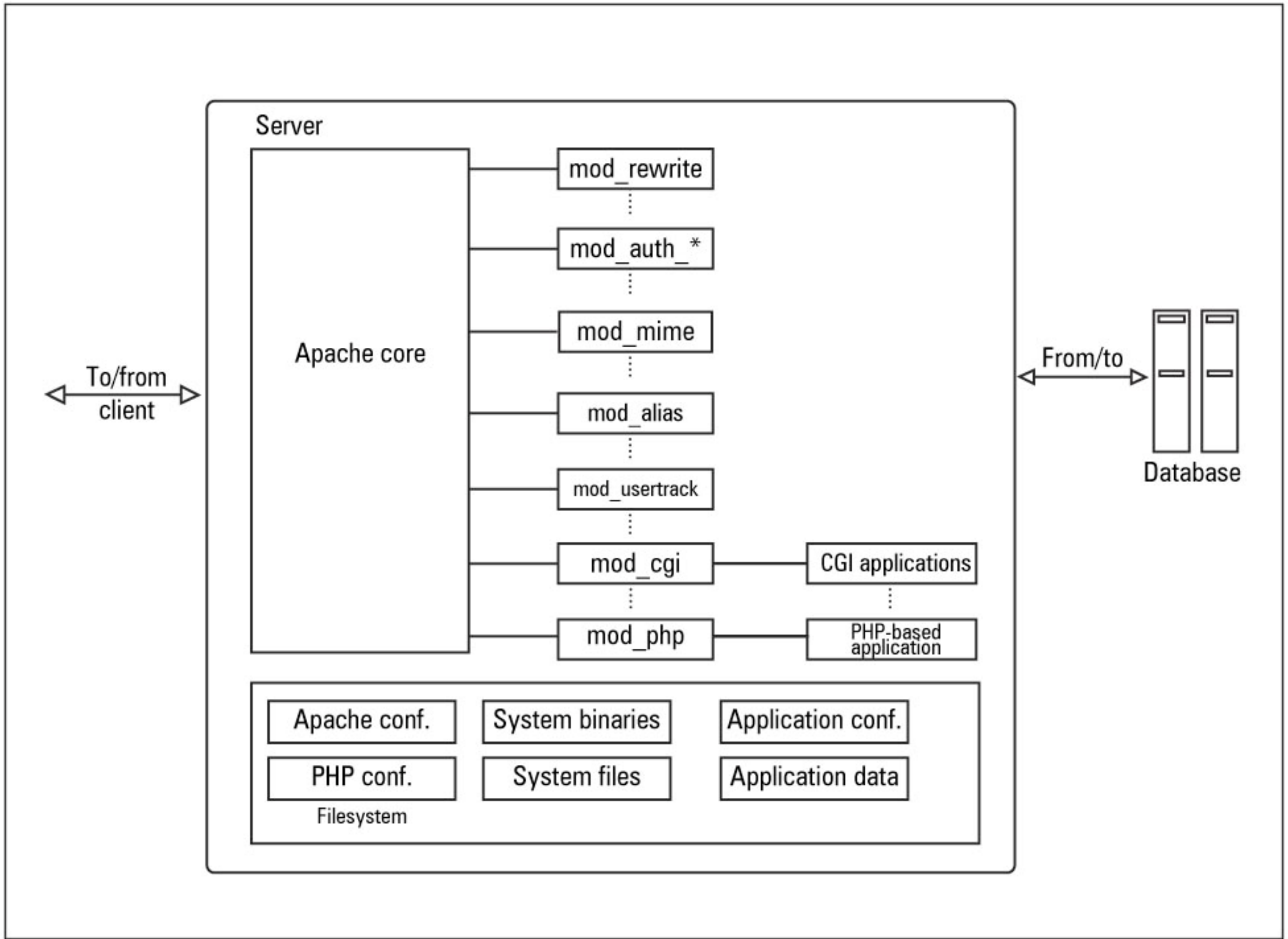


Image from: *http://www.linuxforu.com/2010/08/securing-apache-part-1/*

Arpit Bajpai

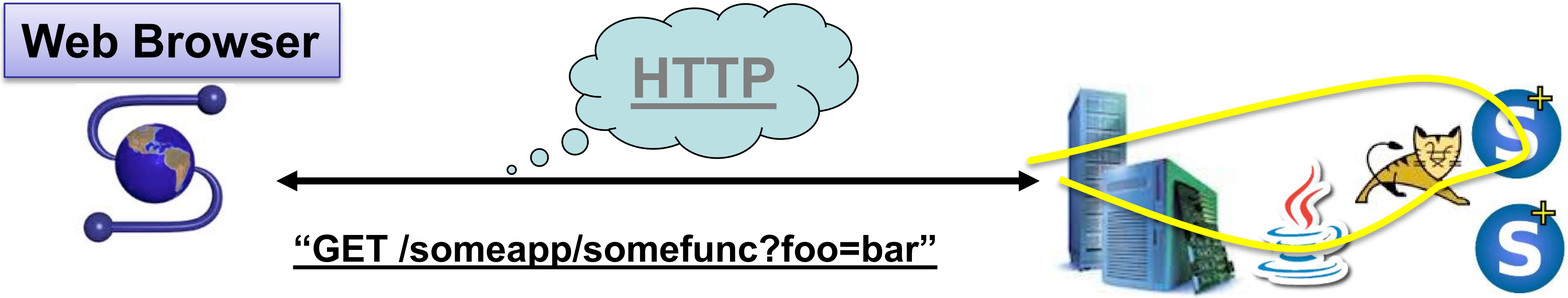
Java Servlets

A server-side *component* which responds to requests

- Resides within a server
- Receives information through parameters or a stream/reader
- Returns MIME-typed result(s) through a stream/writer

Servlets introduced a fundamentally new web architecture

- Server-side “components” that run inside a Java “container”
- JVM process sticks around, as can objects
- Specification/API formalized component/container model, provided class definitions, and supported deployment models



Servlet Example

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class DateServlet extends HttpServlet   {
```

Your servlet must extend HttpServlet

Processes HTTP GET requests

```
    public void doGet( HttpServletRequest req,
                       HttpServletResponse res)
                    throws ServletException, IOException     {
```

These encapsulate HTTP request/response

Need these exceptions

```
         res.setContentType("text/html");

         PrintWriter out = res.getWriter();

         out.println("<html>");
         out.println("<head><title>First Servlet</title></head>");
         out.println("<body><H2>Current time is</H2>");
         out.println( (new Date()).toString() );
         out.println("</body></html>");
    }
}
```

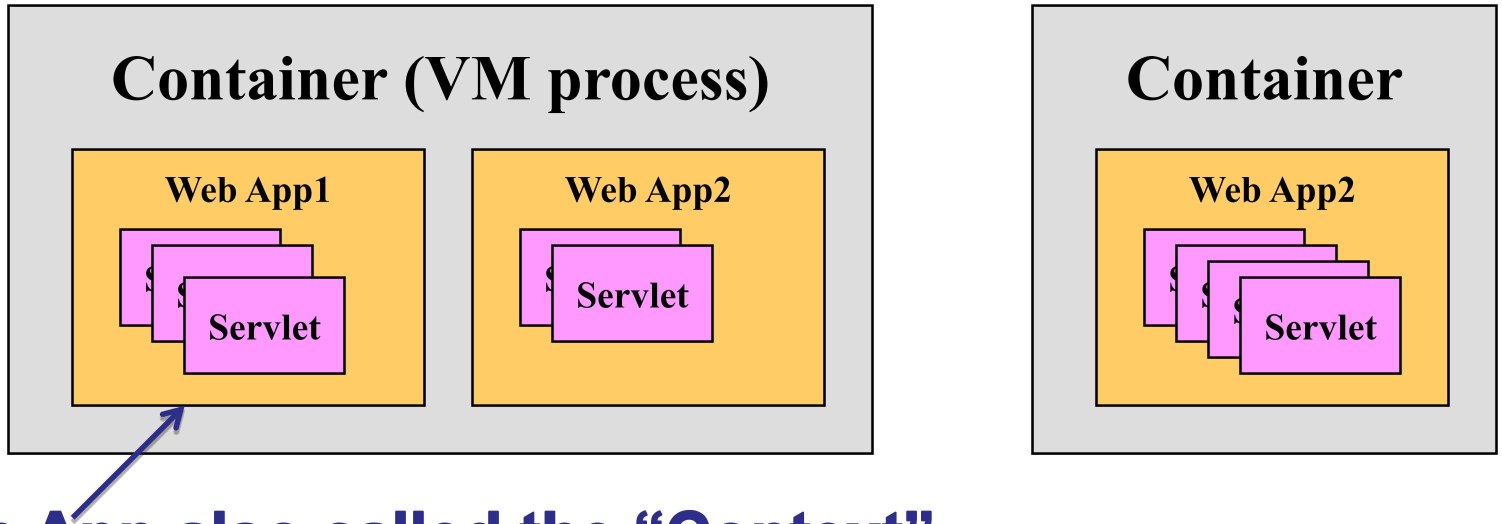
Get an output stream to send output to from the response object

Set the Content-type of our response header

Container-Managed Applications

Servlet components run inside a container

- Container is multi-threaded and processes requests concurrently
  - Servlets are Singletons
  - Servlets must be thread-safe
- Applications within a container share process resources
  - An important one is a shared thread pool of "workers"
  - Example of thread efficiency vs. process protection
  - **Remember:** 1 application may have many components (servlets)

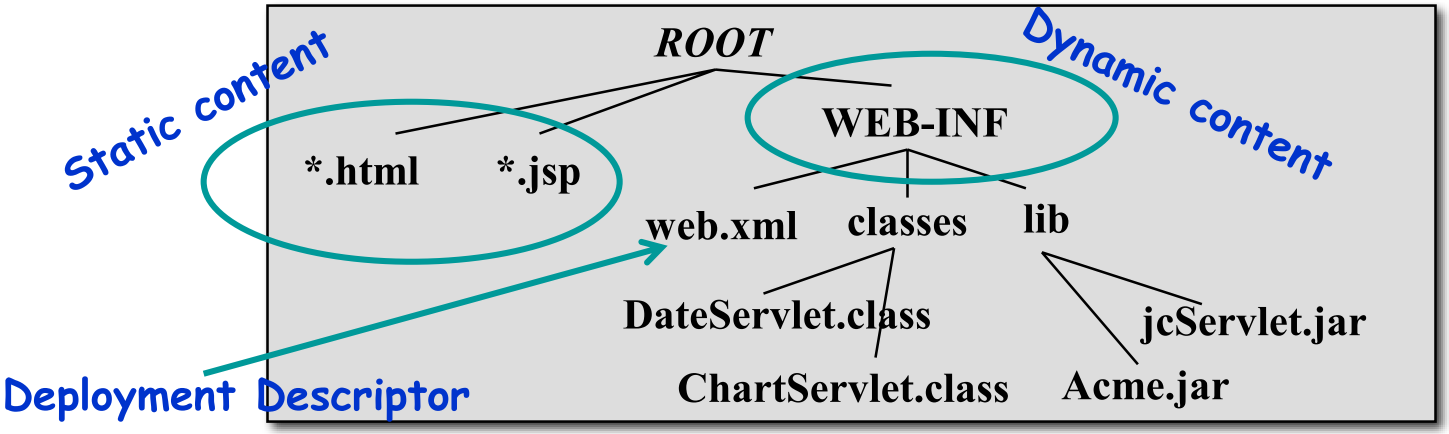


Web App also called the "Context"

Web ARchive (WAR) Files

Enables consistent deployment across engines

A jar file with the following structure:

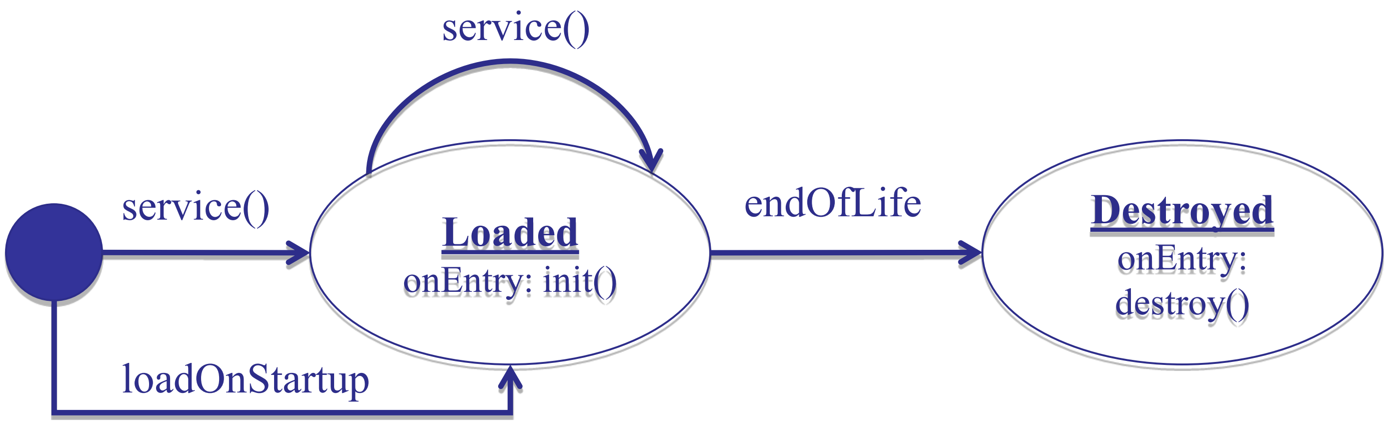


- WEB-INF content is secure and not directly accessible through the browser
- web.xml describes web application’s deployment
- This is an example of a standard component *packaging* so it can be presented to the *container* for execution.

Servlet Lifecycle

One instance of each component servlet exists per container

- Servlet instances must be thread-safe



endOfLife here means the servlet container may choose when to terminate the lifecycle of this servlet instance (unload the class)

This is a common practice for component-based computing

- `init(ServletConfig config)`
  - called once to notify servlet it is going to be utilized
- `destroy()`
  - called when servlet is being de-allocated

Java Servlets Summary

Java Servlets (at the time) represented the next evolutionary step in Web application architectures

1. CGIs as a separate process
2. Server-side scripts as an embedded process
3. JVM as a separate component-container process

Pros and Cons:

Pros:

- Significant install base
- Lots of trained folks with lots of tools and lots of frameworks
- Has been shown to be secure and scale for heavy computational apps

Cons:

- Operations staff has always had a hard time supporting
- Code can be quite verbose, bloated, and “boring”
- Does not easily play well with client-side focus

Summary

- CGIs were the original webhack
  - Spawns a separate process
  - Has evolved architecturally so not dead yet
- SSIs were little dynamic fragments
  - Javascript eliminate the need largely
  - Lots of one-offs in the early web (late 90s)
  - Scripts interpreted “in-process” so a bit better
- Server-side plugins and extensions
  - Optimized the web server process to manage dynamic execution spaces at run-time
  - Widely used but architecture often specific to the HTTP vendor
    - Example: Apache mod handlers, nginx FastCGI and async workers
- Component-container architectures represented by servlets
  - Still quite popular today (Spring)
  - Highly scalable but “heavy”, can be difficult to deploy and optimize