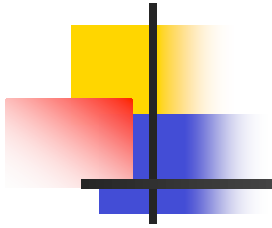
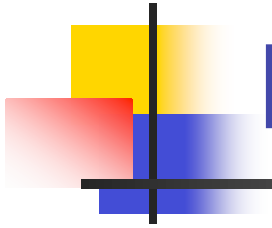


B336 Internet Systems Programming

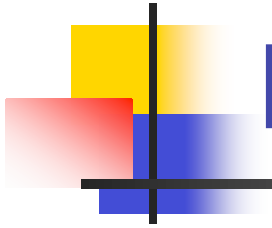
Web Server Configuration and Administration Using Apache

(Week 2 Lecture 1)



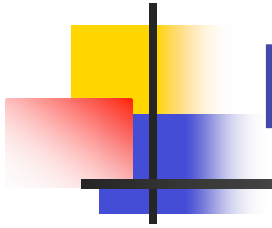
Learning Objectives

- Understand the basic web server installation and configuration issues using Apache.
- Understand some issues dealing with making a web server perform at an optimal level.



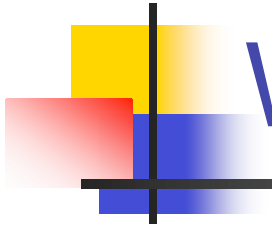
Lecture Objectives

- Relevance to unit objectives:
  - Learning objective 1: Understanding Technologies
  - Learning objective 2: Writing Software
  - Learning objective 3: Construct Solutions
- Relevance to assessments:
  - Introduce material required for
    - labs in week 2 and 3
    - Assignment 1.



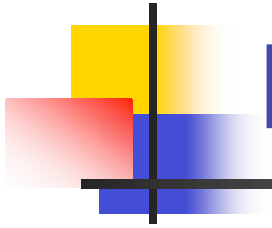
Lecture Outline

- Why use Apache in this unit?
- Basic server issues:
  - Apache Installation
  - Apache's process control
  - Server configuration directives
- Advanced issues:
  - Server process control and resource usage
  - Network configurations
  - Logging
  - Controlling dynamic content
  - Directory-level options and access restrictions to files and directories



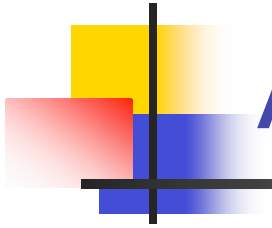
Why Use Apache in this Unit?

- We need an example web server for you to play with to learn about web serving issues.
- It needs a complete set of features so you can see the full power of a web server before starting to implement a simple one yourself (next week)
- Apache is the most popular web server on the web today
  - Estimated 50-60% of sites on the web uses Apache
  - You will improve your CV considerably having direct exposure to it.
- It's free, and it's Open Source.



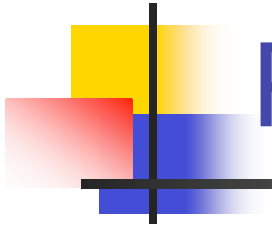
Examples of Other Servers

- Some other example major web servers:
  - Miscrosoft's IIS
  - Netscape's Enterprise
  - Sun-Netscape Alliance's iPlanet
  - NCSA's HTTPd
  - etc
- Review of current web servers:
  - http://serverwatch.internet.com/webservers.html



Apache Installation

- We will go through the installation steps in next week's labs
  - Details about what to do in the labs exercises.
- You will install and run the server in your own directory on the Unix server gryphon.murdoch.edu.au
  - you will all use different port numbers based on your student IDs, not the standard port 80 for HTTP access.



Port Numbers

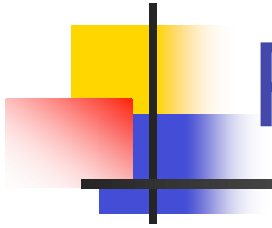
- So the situation we will have is that if someone uses the URL

  `http://gryphon.murdoch.edu.au/`

  Then they will get to the default, superuser-installed web server on gryphon.

- This is because the client defaults to if the port number is not typed in by the user. The actual URL for the above is:

  `http://gryphon.murdoch.edu.au:80/`



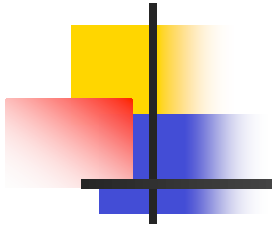
Port Numbers

- To get to your web server, which you will run only when you need to, the URL will be

  `http://gryphon.murdoch.edu.au:12345/`

  (replace12345 with whatever port number you are assigned)

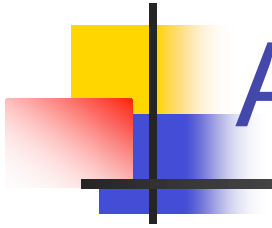
- This way, we can have multiple web servers running on one machine at the same time.



Forks and Threads

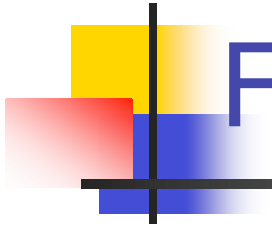
- Two important concepts in web serving are:
  - **forks**: when one process (called the parent) creates a complete copy of itself to execute certain things
  - **threads**: parts of a single process which can execute independently without worrying about each other.

- A web server must serve more than one request at a time, so there needs to be mechanisms like forking and threading in programming languages, and in the operating system to allow them to do this.



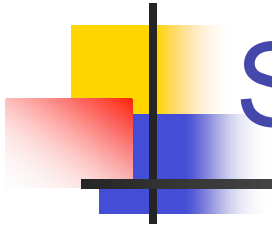
Apache‘s *Pre-Forking* Model

- Apache uses a *pre-forking* model
  - A running Apache always has a *parent* process running.
  - The parent process is responsible only for forking *child* processes, it does not serve any requests itself.
  - The child processes process connections, before dying (hopefully doing so gracefully, if nothing goes wrong).
  - The parent spawns new or kills off old children in response to changes in the load on the server.



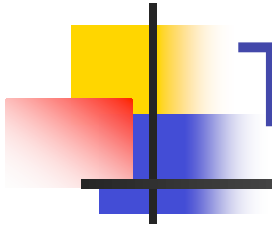
Forks and Threads

- We will not be going into the programming of forking and threading in this unit.
  - The unit **B310 UNIX and Network Programming** will cover them very extensively.
- For this unit, we will concentrate more on dealing with the request/response transaction, rather than processes required to do multiple transactions at once.
  - But do keep the basic ideas in mind, as you will need it to deal with administration issues for a web server like Apache.



Starting-Up Apache

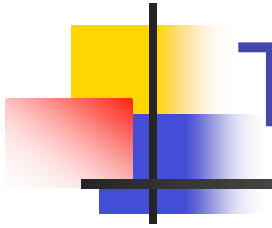
- The Apache program is called `httpd` (HTTP daemon)
  - A “daemon” is a common UNIX term for running processes.
- Starting and stopping the Apache process is done by typing the command “`httpd`” at the command-line.
- The Apache distribution also comes with a script called “`apachectl`” which you can use to control apache.
  - Eg. apachectl start
    apachectl stop
    apachectl restart



The Apache Start-up Process

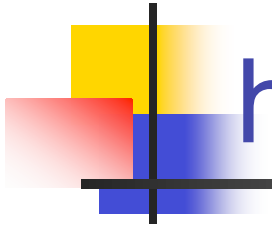
- When the Apache server starts up, it will determine what the appropriate configuration (eg. what port number to listen to) is by reading some configuration text files
  - by default in the `conf` directory your installed it in.
- The files are read in the following order:

  `httpd.conf, srm.conf, access.conf, magic, mime.types`



The Apache Configuration Files

- These days, most directives for resource (srm.conf) and access (access.conf) control are put in the main httpd.conf file as well
  - srm.conf and access.conf exist only for historical reasons
- The `magic` and `mime.types` files you usually leave as they are.

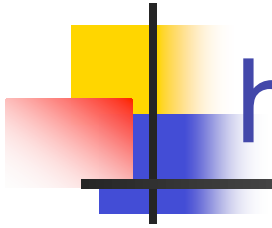


httpd.conf

- This is the file to tell the server how to run
  - ie. it determines the configuration.
- It is read by httpd process on start-up
  - any changes made to this file won't take effect until the server is restarted.
- Lines starting with "#" are comments and ignored when the server reads the file
  - Comments in the httpd.conf file contain very detailed description of the directives that comes with the standard installation. You should read them to get more details.

Example httpd.conf

```
ServerType standalone
ServerRoot "/usr/local/apache”
Port 80
ServerName www.it.murdoch.edu.au

<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>

<Directory "/export/home/staff/hiew/public_html">
    Options +ExecCGI +Includes +Indexes
    AllowOverride All
</Directory>

...
```



httpd.conf

- Besides the comments, all other lines are *directives* (ie. instructions for the server to follow)
  - Each line (except blank lines) contains one directive
- The next few overheads are example ones you need to know to start off.

`ServerType standalone`

- ServerType is either “inetd”, or “standalone”.

- “inetd” is the UNIX operating system’s Internet daemon process which collates and distributes all requests to ports it knows about
    - inetd mode is not supported in the MS-Windows version of Apache.
    - You need to be the superuser to register with inetd.
    - Also, you lose some control over configuring the resource management for your server (eg. how many child processes to have) if you get inetd to be your initial start-up point.

```
ServerRoot "/usr/local/apache"
```

• The top-level directory of the web server. All server's files are kept in this directory or sub-directories underneath it. Eg.

- Configuration files: `$ServerRoot/conf`
- Log files: `$ServerRoot/logs`

```
Port 80
```

- The port that the standalone server listens on

```
User nobody
Group nobody
```

- The user and group that the server process runs as.
- This determines what files it can permissions to access
    - when you want the web server to serve files in a certain directory, the user `nobody` must have access to that directory as well.

```
StartServers 5
MaxClients 150
MinSpareServers 5
MaxSpareServers 10
MaxRequestsPerChild 30
```

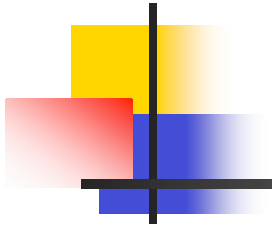
- Some settings for the child processes serving requests
- More on this when we talk about advanced server issues later.

```
DocumentRoot "/usr/local/apache/htdocs"
```

- The directory where your web-site files are

```
ServerName www.it.murdoch.edu.au
```

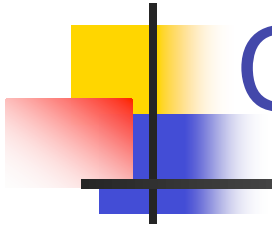
- name which is sent back to clients if it's different than the one the program would get.



Scope of Directives

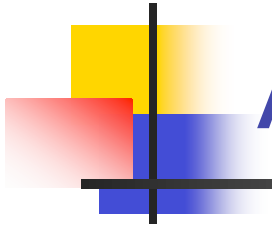
- The examples above are all *server-wide* directives.
  - They apply to the server itself.

- Some directives are only relevant to certain parts of the server. These directives are placed in *context* tags.
  - Eg. in the httpd.conf example on overhead 16 in this lecture, the second <Directory> tag encloses directives that apply ONLY to `/export/home/staff/hiew/public_html` directory (and all sub-directories in it).



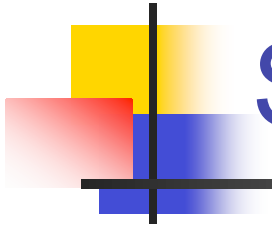
Configuration for your installation

- In next week's labs, you will be given a set of standard configurations for your own installation on `gryphon.murdoch.edu.au`.
  - Keep in mind that they will be different from the standard examples above, since it is not normal we have so many different installations running at the same time, and all of which do not have superuser/administrator priviledges.
- You can then try out different configurations of your own.
  - But please stick to the port number you are assigned to.



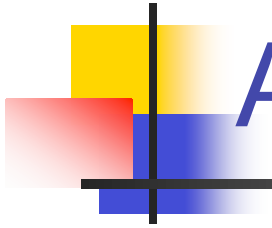
Advanced Server Issues

- For the rest of this lecture, we will deal with more advanced web server issues in Apache.



Server Process Creation

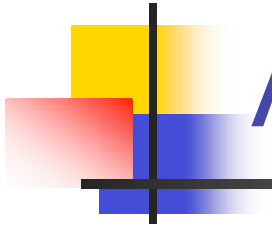
- When the `ServerType` directive is set to `standalone`, Apache (1.3 onwards) acts as a pre-forking server as described in the last lecture.
- The parent process
  - launches the child processes, and the child processes serve requests as they come in
  - launches *spare* child processes if the currently available set of processes are occupied
  - kills off idle child processes as appropriate



Apache “Family Planning”

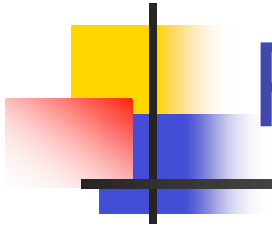
- Under Unix, the `StartServers`, `MinSpareServers`, `MaxSpareServers` and `MaxServers` directives determine the limits of how the parent server should behave in terms of child process creation.

- The `MaxClients` directive determines how many requests the server can serve at any single time
  - This is effectively the maximum number of child processes the parent can create.



Apache “Family Abuse”

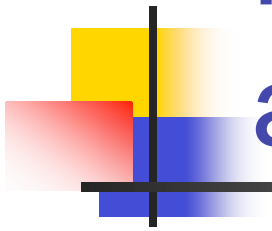
- The number set for the `MaxRequestsPerChild` directive determines how many requests a child server should serve before the parent kills it off
  - This is to prevent memory leaks or other errors that may be caused by a process that has hung around for too long.



Process creation under Windows

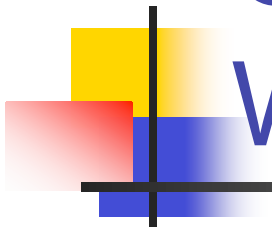
- In Microsoft Windows operating systems, Apache's main server launches one child process, and the child process creates multiple threads to serve requests.

- The `ThreadsPerChild` directive controls the number of threads the child can create.



Performance Issues: Hardware and OS

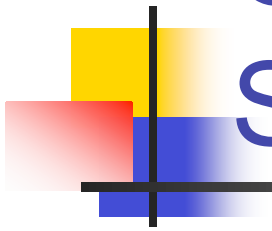
- Your web server's performance is highly dependant on the underlying hardware and OS supporting it.
  - Web server administration goes hand-in-hand with general system administration.
- The more available and efficient hardware (RAM, CPU, disk, network, etc), the better performing the web server will be.
- The better configured the OS is, the better performing the web server will be.



Operating System and the Web Server

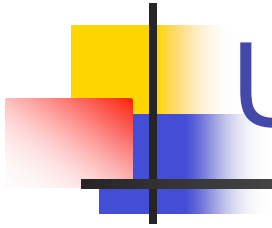
- Obviously the more resources your operating system allocates to the web server, the better the performance of the web server.
  - But generally there is no hard and fast rules that apply to all web servers - consult the web server documentation on any OS specific issues.
- The same web server ported to different operating systems have different performance issues.
  - Eg. we have mentioned in this lecture that Apache's process creation model is different for its Win32 version compared to its original UNIX version.

Operating System and the Web Server

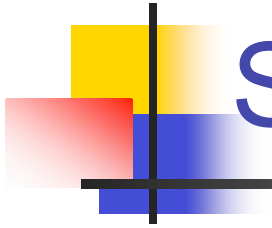


- Some examples from comparison for Apache between its UNIX and Win32 versions:
  - The performance of Apache's in UNIX depends on how well UNIX handles standalone processes or the inetd daemon, while in Win32 it depends on how well Windows handle their "services".
  - The `ThreadsPerChild` directive is not important in the UNIX version (since it depends more on multiple processes rather than threads in a single process), but is critical in the Win32 version (which depends solely on threads to serve multiple requests).

Monitoring System Resource Usage

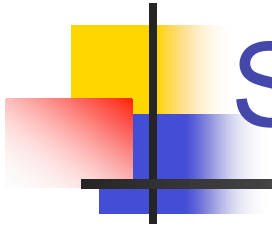


- We can monitoring the resource usage of the web server by using whatever tools is available through our operating system
  - In your lab exercises, we have a look at some simple operating system commands available in Linux to watch the performance of the Apache server.
- Apache also has a module which enables an administrator to watch the systems performance using a web interface.



Server Status using mod_status

- Apache has a module called `mod_status` which enables server status to be monitored through a web page.
- `mod_status` is compiled into the default installation.



Server Status using mod_status

- Eg. To enable anyone to view the server's status, put the following in the `httpd.conf` file:

```
<Location /server-status>
    SetHandler server-status
    Order Deny,Allow
    Deny from all
    Allow from all
</Location>
```

- Then access the page http://gryphon.murdoch.edu.au:12345/server-status

Another example using mod_status

- Eg. To enable only Murdoch machines to access the server status:

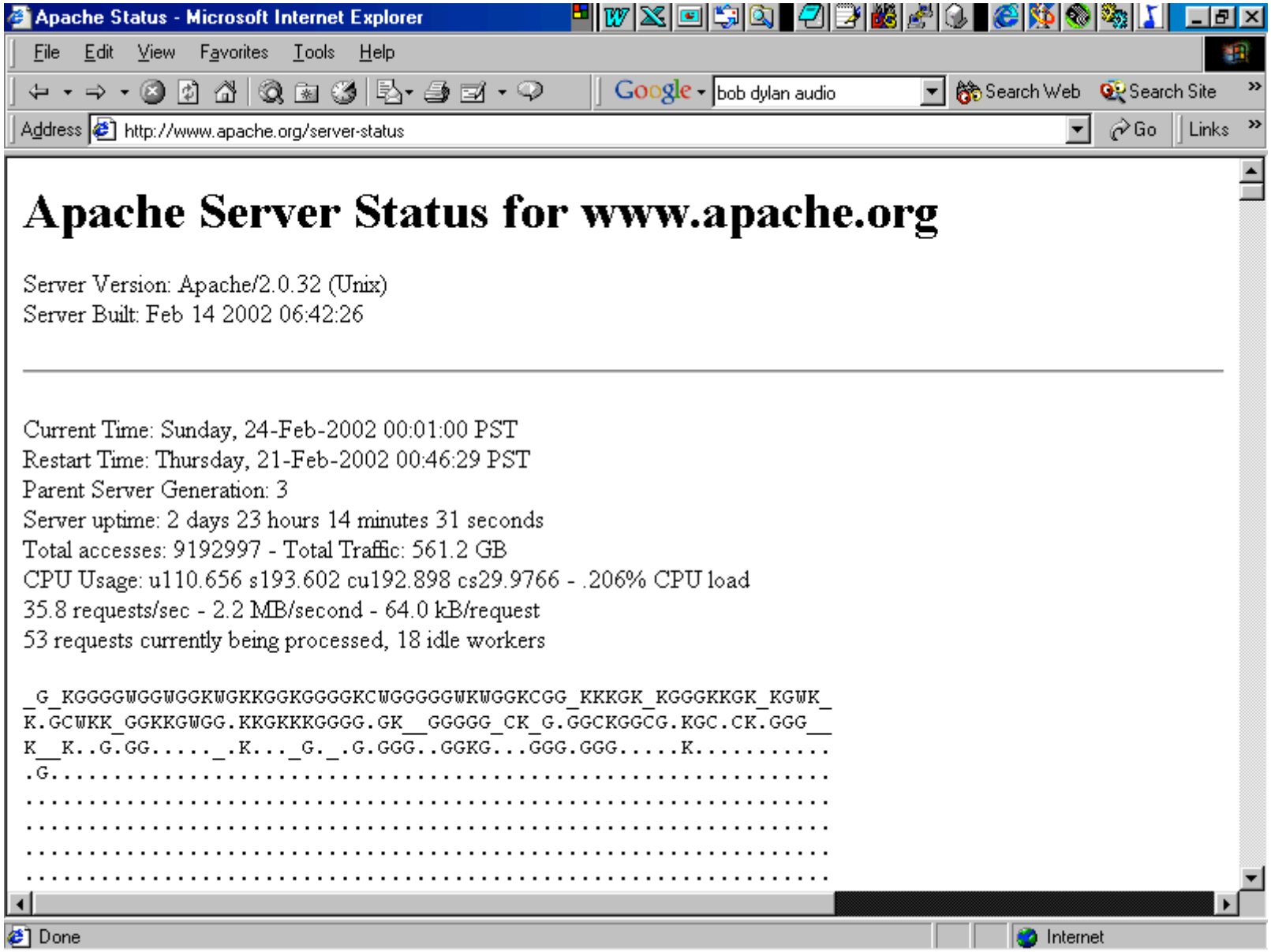
```
<Location /server-status>
        SetHandler server-status
        Order Deny,Allow
        Deny from all
        Allow from .murdoch.edu.au
</Location>
```

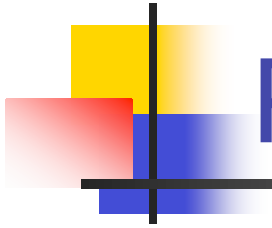
- You can also add an extra directive anywhere in `httpd.conf` for more status display:

```
ExtendedStatus On
```

What the server-status page shows

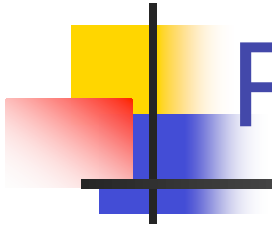
- The standard page shows:
  - The number of children serving requests
  - The number of idle children
  - The time the server was started/restarted and the time it has been running for
- With `ExtendedStatus` on, you also get:
  - The status of each child, the number of requests that child has performed and the total number of bytes served by the child
  - A total number of accesses and byte count served
  - Averages giving the number of requests per second, the number of bytes served per second and the average number of bytes per request
  - The current percentage CPU used by each child and in total by Apache
  - The current hosts and requests being processed


Apache Status - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Google bob dylan audio Search Web Search Site
Address http://www.apache.org/server-status
Go Links
Apache Server Status for www.apache.org
Server Version: Apache/2.0.32 (Unix)
Server Built: Feb 14 2002 06:42:26
Current Time: Sunday, 24-Feb-2002 00:01:00 PST
Restart Time: Thursday, 21-Feb-2002 00:46:29 PST
Parent Server Generation: 3
Server uptime: 2 days 23 hours 14 minutes 31 seconds
Total accesses: 9192997 - Total Traffic: 561.2 GB
CPU Usage: u110.656 s193.602 cu192.898 cs29.9766 - .206% CPU load
35.8 requests/sec - 2.2 MB/second - 64.0 kB/request
53 requests currently being processed, 18 idle workers
_G_KGGGGWGGWGGKWGKKGGKGGGGKCWGGGGGWKWGGKCGG_KKKGK_KGGGKKGK_KGWK_
K.GCWKK_GGKKGWGG.KKGKKKGGGG.GK__GGGGG_CK_G.GGCKGGCG.KGC.CK.GGG__
K__K..G.GG....._.K..._G._.G.GGG..GGKG...GGG.GGG.....K..........
.G.....
Done
Internet



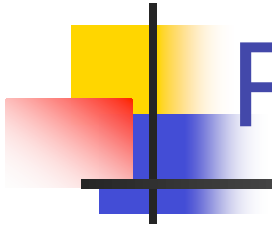
Process Control

- We have gone through quite a few issues with process control over the last two lectures, since at the performance of a web server, really means the performance of its processes.
- Determining the correct settings for process start-up like `MinSpareServers`, `MaxSpareServers`, `StartServers`, etc, can be made by monitoring the server's resource usage.



Process Control

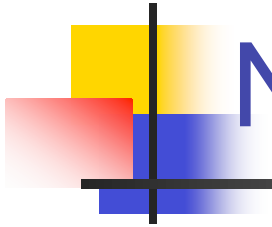
- The correct settings for process termination like the `MaxRequestsPerChild` directive is also important.
  - In Apache for UNIX, since `MaxRequestsPerChild` is usually set to 0, which means there is not fixed period on when a child process will die. The only time it does is if the parent have created a lot to serve a heavy period, and then find too many sitting idle after the period.
  - This is OK for most versions of UNIX, but for some like the old SunOS, the number recommended to be about 10000 since the process suffers from possible memory leaks.



Process Control

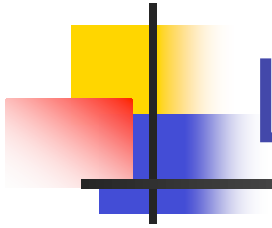
- The correct configurations for process control should also be guided by what you know, and what you believe your web-site content will be, and what access behaviours your users will have.

- Eg.
  - You should know how to estimate expected sizes of files on your web site.
  - You should be able to determine how often the files will be accessed, by either monitoring your server, or by using a web-counter.



Network Configurations

- Similar to process control configurations, a web server's network-based behaviour also determines how well it performs.
- Eg. Directives like `KeepAlive`, `KeepAliveTimeOut`, `LimitRequestBody`, `RLimitCPU`, `RLimitMEM`, etc.
- Refer to online manuals for more descriptions.



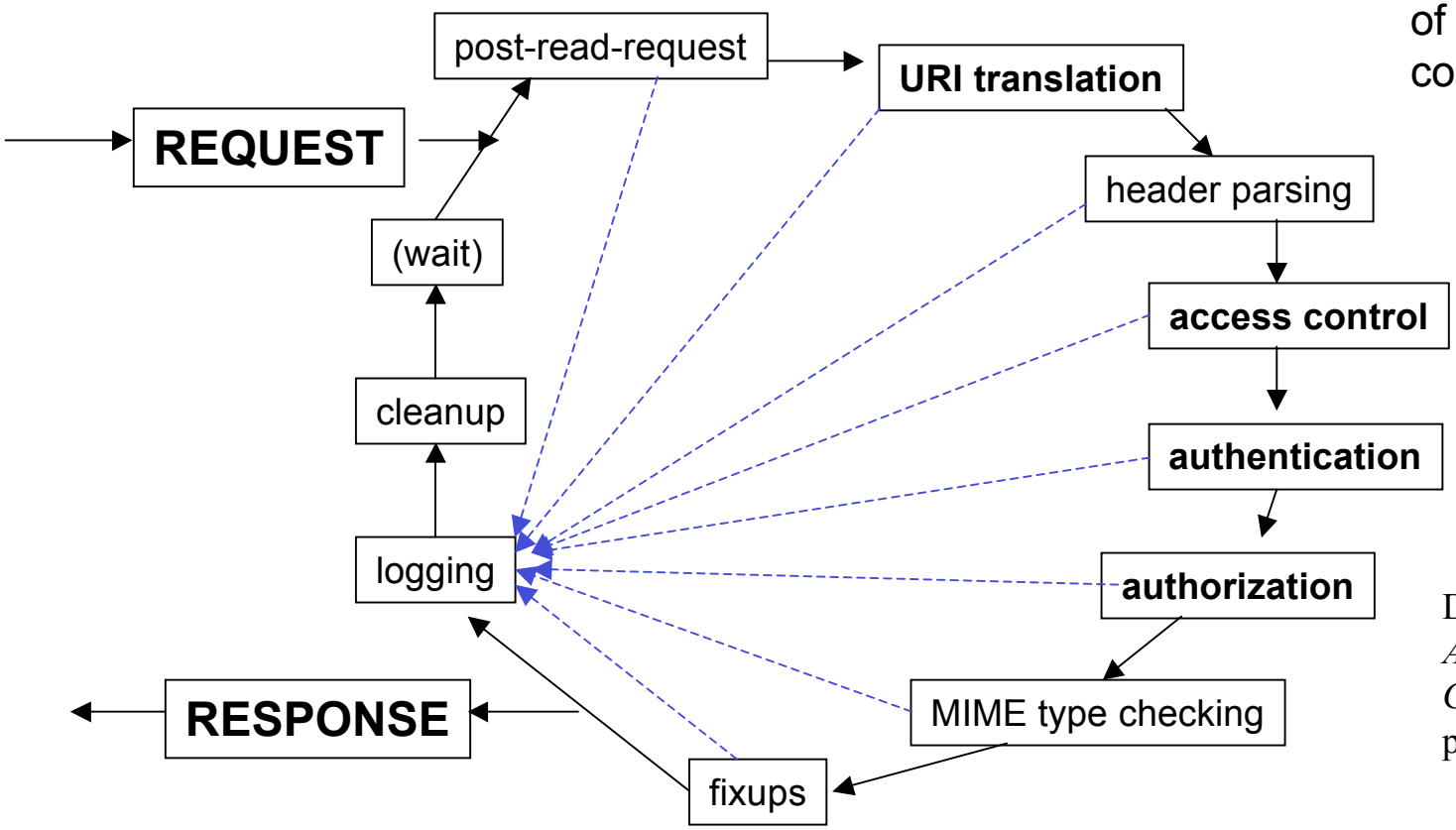
Logging

- By default, Apache has two log files:
  - **`error_log`** (or **`error.log`**) in the `logs` directory where you installed Apache - logs errors when serving requests.
  - **`access_log`** (or **`access.log`**) in the `logs` directory - logs of requests succesfully served.
- The location of the log files (or addition of more log files) can be set using the `ErrorLog` and `CustomLog` directives.

An example access_log

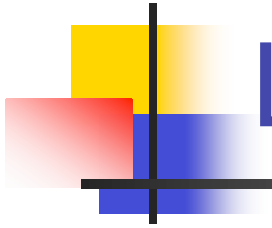
```
203.59.210.3 - - [02/Mar/2001:15:25:32 +0800] "GET /~goulding/unitmaterial/ HTTP/1.1" 401 478
203.59.210.3 - b208 [02/Mar/2001:15:25:33 +0800] "GET /~goulding/unitmaterial/ HTTP/1.1" 200 1999
203.59.210.3 - - [02/Mar/2001:15:25:35 +0800] "GET /~mann/discus/messages/1/1.html?983512181
HTTP/1.0" 200 5713
165.118.9.17 - - [02/Mar/2001:15:25:39 +0800] "GET /units/b105 HTTP/1.0" 301 322
203.59.210.8 - b208 [02/Mar/2001:15:25:41 +0800] "GET /~goulding/unitmaterial/lectures/ HTTP/1.1" 200
1119
165.118.9.17 - - [02/Mar/2001:15:25:42 +0800] "GET /units/b105/ HTTP/1.0" 200 35787
203.59.210.8 - - [02/Mar/2001:15:25:43 +0800] "POST /~mann/cgi-bin/discus/board- post-form.cgi
HTTP/1.0" 200 3260
134.115.157.208 - - [02/Mar/2001:15:26:01 +0800] "GET /~sk/b217/etched double line.gif HTTP/1.0" 404
287
203.59.210.3 - b208 [02/Mar/2001:15:27:50 +0800] "GET /~goulding/unitmaterial/lectures/topic2slides.doc
HTTP/1.1" 200 524288
203.59.210.8 - - [02/Mar/2001:15:28:06 +0800] "POST /~mann/cgi-bin/discus/board-post.cgi HTTP/1.0"
200 9657
```

Logging the Request Loop



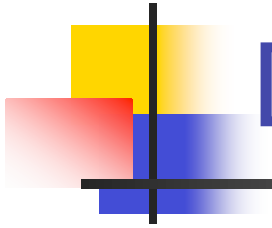
Apache can log any part of it's operations if you configure it to.

Diagram modified from *Writing Apache Modules with Perl and C*, Stein & MacEachern, 1999, page 60.



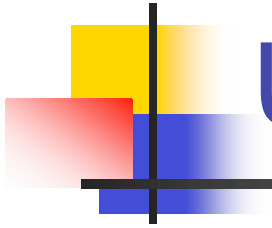
Logging

- Managing logs is an important task in web server administration. It allows you to :
  - detect security breaches, or possible security attacks
  - determine errors in your web server configurations, or web-site content (eg. a main link which has the wrong address will constantly generate an error when accessed)
    - Even CGI script errors!
  - Determine inefficiencies (eg. see example at `http://www.modperl.com/perl_conference/handout.html#Detecting_Robots` for how to use the logs to detect “impolite” web robots/spiders.



Dynamic Content using CGI

- Apache uses the standard CGI paradigm to interact with external content-generating programs
  - Perl scripts is only one example - it can interact with any external programs
- To enable CGI processing in Apache (ie. to get the server to execute and serve the results of CGI scripts/programs), there are a few different ways.

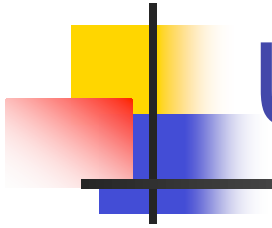


Using the ScriptAlias directive

- The `ScriptAlias` directive specifies directories which Apache will serve CGI programs from.

- Eg.

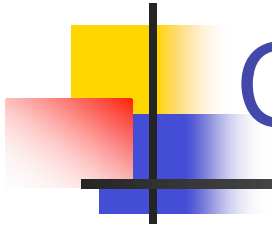
```
ScriptAlias /cgi-bin/  /usr/local/apache/cgi-bin/
```



Using the Options directive

- You can also use the `Options` directive to specify the same directories for CGI executables.
- To do this, you will first need use the `AddHandler` directive to specify the file extensions which corresponds to CGI files.
- Eg.

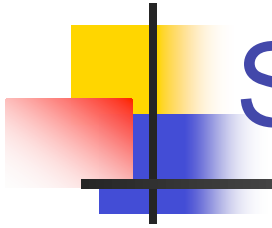
```
<Directory "/usr/local/apache/cgi-bin/">
        Options +ExecCGI
</Directory>

AddHandler cgi-script cgi pl
```



CGI Scripts in other directories

- *By default*, all other CGI scripts in other directories besides the ones specified using the above will be treated as non-executing files (eg. .html will be treated as normal HTML files).

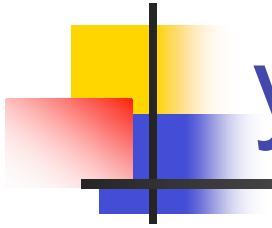
- There are many other ways to get around this default. Read the online documentation for more ideas.



Server-side Includes (SSI)

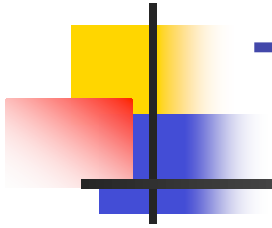
- Server-side Includes are directives in HTML files which the web server reads and executes before sending the final HTML file back to the requesting client.
- It is a much more efficient way of including dynamic content than starting up CGI processes, but less powerful.

Some Example SSI directives you can put in your HTML files



- In HTML files, SSI directives are just comments:

```
<!--#echo var=”DATE_LOCAL"-->

<!--#echo var="LAST_MODIFIED"-->

<!--#include virtual=”header.html"-->
```

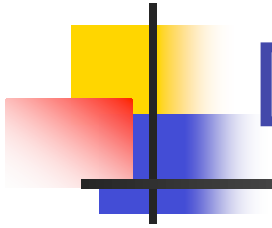


To Enable SSI

- You can use the `Options` directive again.
  - Eg. `Options +Includes`

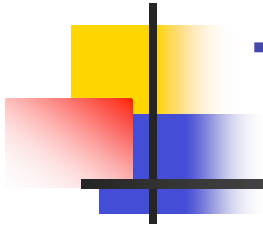
- Again, you also need to specify the files which should be parsed for SSI.
  - Eg.

```
AddType text/html .shtml
AddHandler server-parsed .shtml
```



Directory Level Options

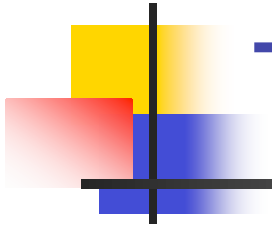
- We mentioned that the `httpd.conf` file (and possibly other .conf files if enabled) controls options and configurations for the web server.
- “Context container” tags in the configuration files like `<Directory>` or `<File>` controls the options specific to individual directories and files.
- Apache has a much more flexible way of controlling directory options: the `.htaccess` file.



The .htaccess files

- The name “.htaccess” is a remnant from the past where it only deals with restricting access. Nowadays, you can put any directory specific options in the .htaccess file.

- It is more flexible way of specifying directory options, since it allows the users to determine their own options without having the administrator modify the `httpd.conf` file.

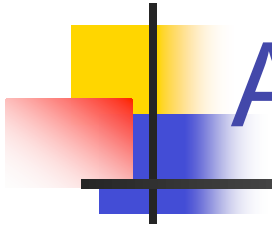


The .htaccess files

- The syntax of a `.htaccess` file is the same as any configuration files (eg. `httpd.conf`).
  - It consists of a set of directives, and context tags.

- The enable `.htaccess` files to work, you must have the following lines in your `httpd.conf`:

```
AccessFileName .htaccess

AllowOverride All
```



Another example .htaccess setting

- To be more specific, you can do:

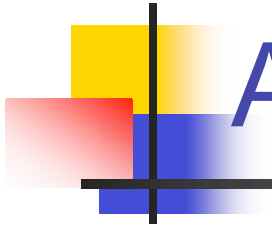
```
AccessFileName .htaccess
<Directory /home/~trusted/file>
        AllowOverride AuthConfig
</Directory>
```

- This means you are only granting the directory /home/~trusted/file the ability to change the default authorization configurations.

An example .htaccess file

- Here is an example `.htaccess` file I put in the B336 lecture-notes file directory:

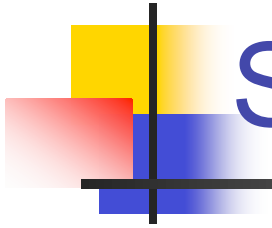
```
AuthName "B336 2003"
AuthUserFile /home/staff/hiew/.htpasswd.b336
AuthType Basic
<Limit GET POST>
        order deny,allow
        require valid-user
</Limit>
```



An example .htaccess file

- For me to use the .htaccess file in the previous page, I have to create a .htpasswd.b336 file. To do that I use the htpasswd program that came with Apache. On the command line I type:

```
/usr/local/apache/bin/htpasswd -c .htpasswd.b336 b336
New password: getanHD
Re-type new password: getanHD
```

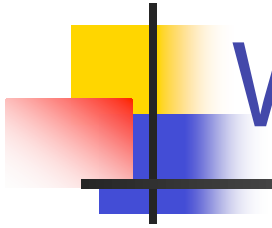


Security Implications

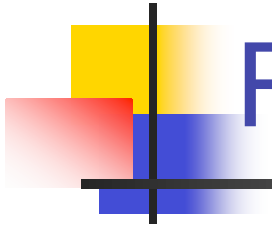
- By enabling `.htaccess` files, you are basically granting certain administration powers to users. You should be careful how much power you grant, and keep constant watch on potential problems.
- To disallow web clients from accessing the `.htaccess` files through the web, you should also include in `httpd.conf`:

```
<Files .htaccess>
        Order allow,deny
        Deny from all
</Files>
```

Web Server Administrator vs Web Content Managers



- Today, more and more, there is a distinction made between the person(s) who administers the web server, and the person(s) who administers the web-site contents.
  - The former deals with the installation, configuration and performance issues we have been dealing with in this course.
  - The other deals with the content of the web-site (HTML and JavaScript code, dynamic scripts, links to databases, etc)
- The term “webmaster” can mean either or both those responsibilities.



Reference Documentation

- The lectures and labs will only be able to give you a basic introduction to the web serving features of Apache.
- To operate it properly, you will frequently have to refer to the official online documentation - which includes quite a lot of tutorials.
- You will be shown the location of the documentation on your own installation in the labs. You can also get it from the Apache site `http://httpd.apache.org/docs/`