Apache Server Architecture

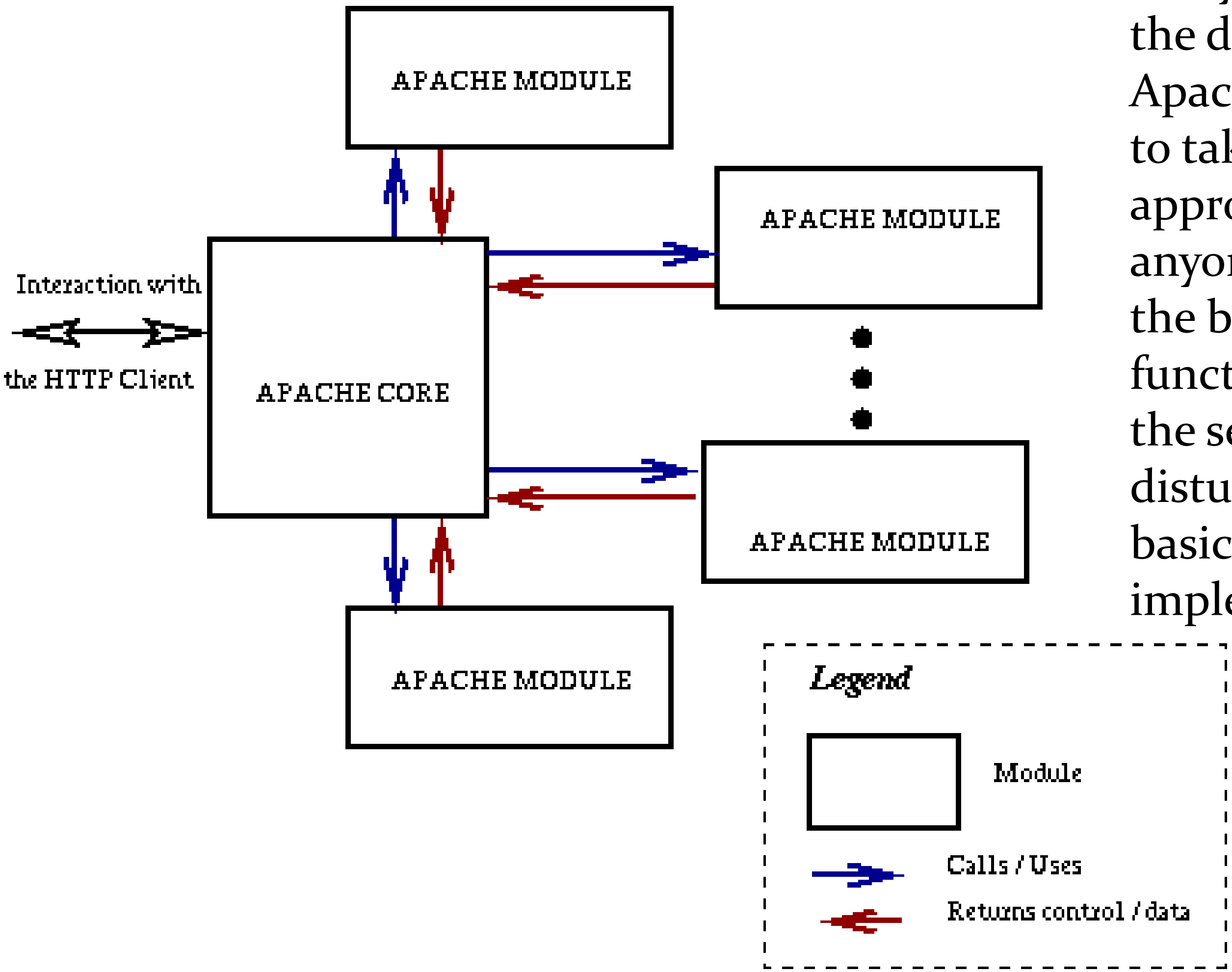
Brief Background

- Initially started in 1996 by Robert McCool.
- Since 1996, Apache web server has been the most popular HTTP server in the market on the World Wide Web.
- The Apache was the first web server architecture that was used by the Netscape Communication Corporation.
- Apache has evolved with the years of the internet. Server is used to support both static and dynamic pages online. Many programming languages are supported by the Apache Server are as follows: PHP, Perl, Python and alongside with MySql. As of April 2008, the Apache Server serves approximately 50% of the current web pages.

Overview of the Apache Architecture

- Apache is a open source HTTP web server. It handles HTTP Requests sent to it and then it is able to them
- Apache is Open source and is built and maintained over at Apache.org
- Apache is comprised of Two main building Blocks with the Latter being comprised of many other little building blocks. The Building Blocks are the Apache Core and then the Apache Modules that in a sense extend the Apache core. More detail on this on next couple of slides.
- Very easy to implement and very easy to add extend its abilities by the adding of different modules. This is why this server has become so popular.

Apache Overview Diagram



- As you can see the designers of Apache decided to take a modular approach so that anyone can add to the basic functionality of the server without disturbing the basic Core implementation.

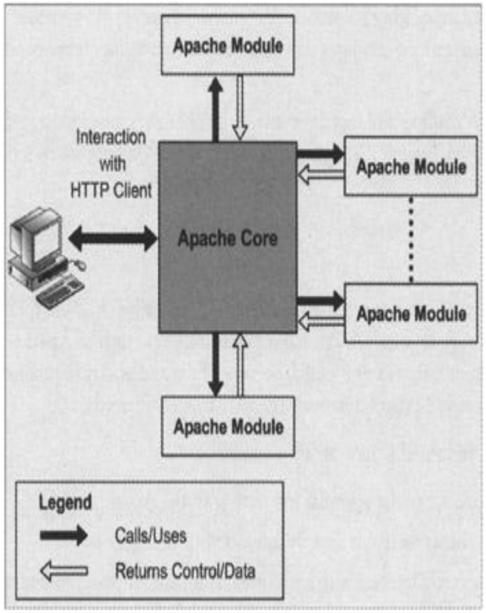
What is a Web Server? And What is HTTP?

- A Web Server is any Machine that receives requests from a client machine and is able to turn around process the requests and send back a response. This is usually in terms of a Web Server to send back a Web pages when people what to go navigate to a webpage that is hosted on that server now one may ask how do you send and receive the information to and from the server. This is where HTTP begins to play a role into how this all goes about.
- HTTP Protocol: HTTP stands for Hyper-Text-Transfer-Protocol
- This is the protocol that is used in order to send and receive information from the server. This is the protocol that the Apache Web Server Understands and it is what it uses to send information back to the client Machine. If you would want to get a bit more technical on the subject the Client Machine this case the Browser sends a HTTP.Request Object to the Server then the Server responds back by using an HTTP.Response Object. This is the general back and forth between the server and the browser. Apache is made to handle all of these requests.

Multi-processing

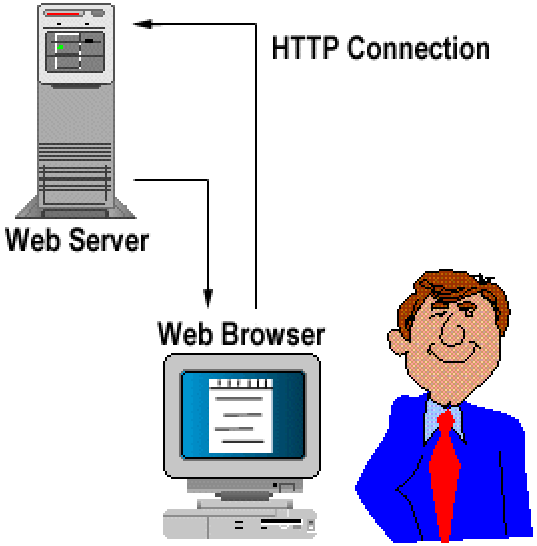
- **Multithreaded and Multi-processed Web Servers**
- When an HTTP request arrives, the Web server starts fetching the resource as requested by the client. While the Web server is busy fetching resources, the client computers might send more requests. This total of requests are needed for processing. These requests are either ignored or handled simultaneously by the Web server.
- Web servers that ignore other requests or they will even queue them while the servers are busy. This process is called single-threaded Web servers. This means that they are incapable of handling high Web server traffic. However, these types of Web servers are ideal for Web sites that encounter low or moderate traffic.

- Web servers that can handle simultaneous requests, manage the requests in two ways. Either they start a new process or they start a new thread within a process. Web servers that start a new process on each request are called multi-processed Web servers, while those that start a new thread within the main process are called multithreaded Web servers. The is a slight difference among those terminology.
- IIS (Internet Information Services) on the Windows platform is an example of a multithreaded Web server. Apache on a Unix platform is a multi-processed Web server.
- Windows platform lack forking for server client request interaction which put Apache on a Unix platform more efficient. Unix is the software for the hardware architecture of a web based server.



Client-Server Interactions

- As we stated before the Client makes an HTTP request to the server in this case the Apache Web server then the server handles the server pools the connections to it, by the basic instructions within the Apache Core then the server sends back a response. Many people do not realize that they be utilizing an Apache web server everyday since it is the most popular web server out right now. When you go online and request a webpage. Most likely an Apache Web server is processing your request and then it is sending you back the webpage you requested

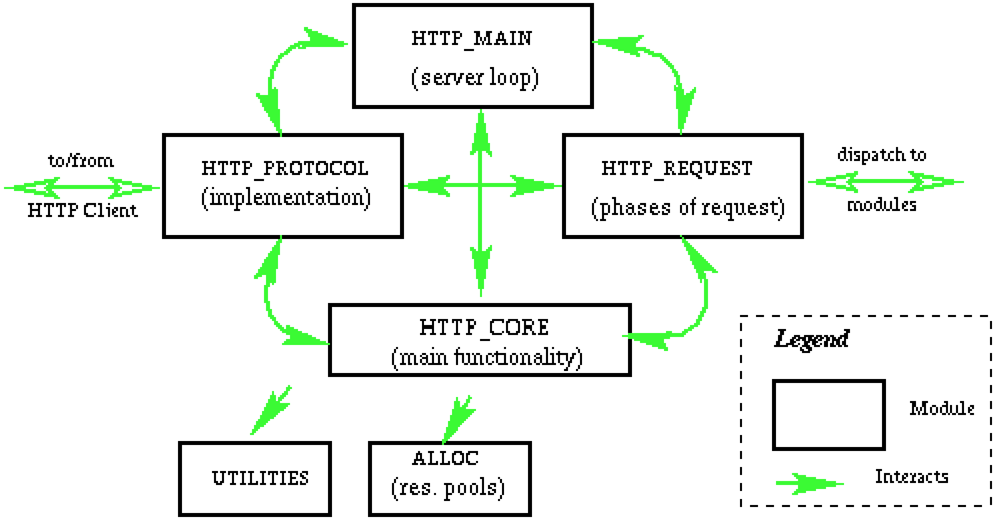


Apache Web Server Where all the Magic Happens…

- As I mentioned Before The overall overview of the Apache Web Server is comprised of a Modular approach to the way the system is built instead of just having the server just be one piece of code handling everything. This in turn allows for more robustness and allow for better customization without getting rid of the security that is implemented within the Apache Core.
- In order to achieve this Modular Approach the Apache Designers decided to break down the server into two main Components.
  - The Apache Core: Which Handles the Basic functionality of the Server. Such as allocating requests and maintaining and pooling all the connections.
  - The Apache Modules: Which are in a sense the added extensions to the server which handle a lot of the other types of processing the server must achieve such as doing user Authentication.

Apache Core

- This is what usually occurs in the Apache Core in a sense an overview of the flow that happens in the Apache Core. This is the Apache Core interacting with all the other components that surround it. These are the core components of the Apache architecture. The purpose for this was that the designers wanted to keep every component that didn’t need each other separate so they made them into modules. So this is what was left after everyt of the Apache Web S



Apache Core Continued…

- The Apache Core is comprised of many different little components that’s handle the Basic implementation of what a web server should be doing.
- The core components are a series of classes that handle specific tasks. These should not be confused with modules, which are just add on implementations of different things that Apache can be customized to do. Modules will be explained more in detail in the further slides.
- The Apache Core provides us with the Main functionality of a HTTP web server. Without it or allowing a change to it will remove its modularity, but also remove some of the security. This is why Modules are needed in order to extend the core functionality of Apache.

Apache Core Components not to be confused with Modules.

- The core components of make up the Apache core are as follows:
  - http_protocol.c: This is the component that handles all of the routines that communicate directly with the client by using the HTTP protocol. This is the component that knows how to also handle the socket connections through which the client connects to the server. All data transfer is done through this component.
  - http_main.c: this component is responsible for the startup of the server and contains the main server loop that waits for and accepts connections. It is also in charge of managing timeouts.
  - http_request.c: This component handles the flow of request processing, passing control to the modules as needed in the right order. It is also in charge of error handling.

Apache Core Components Continued..

  - http_core.c: the component implementing the most basic functionality, it just bairly serves documents.
  - alloc.c: the component that takes care of allocating resource pools, and keeping track of them.
  - http_config.c : this component provides functions for other utilities, including reading configuration files and managing the information gathered from those files (), as well as support for virtual hosts. An important function of http_config is that it forms the list of modules that will be called to service during different phases of the requests that are going on within the server.
- As you can see apache has many different components within the Core these all allow the server to be more secure and more robust, but also due to the implementation of the architecture raises security since anyone that wants to add functionality to the server must do so by the use of modules.

Request Phases

- Before we can continue to talk about the Apache Modules we must be able to talk about what are the request phases that are going on within the core. In other words how does Apache know what to do with a request that it received from the client but also what so it does after it has received the request and where does it go from there in order to handle the request that was made to it. This is where request Phases come into play.
- Modules due to the architecture of Apache do not know directly about each other and not one module alone can completely fill or process the request that is made to the Apache server. Most requests are processes by sending the information from one module back to the core then back to another module until the request is completely handled and then it is sent back to the client. Apache has something called Request Phases and is handled by the HTTP_REQUEST component of the core.

Request Phases Continued…

- The phases or the logic that the HTTP_REQUEST Module of the Apache core controls are as follows:
  - URI to filename translation;
  - Check access based on host address, and other available information;
  - Get an user id from the HTTP request and validate it;
  - Authorize the user;
  - Determine the MIME type of the requested object (the content type, the encoding and the language);
  - Fix-ups (for example replace aliases by the actual path);
  - Send the actual data back to the client;
  - Log the request;

Overview Of Modules

- Modules were made to extend/overwrite and implement the functionality of the Apache web server.
- However modules do not directly extend each other or "know" directly about each other.
- So in turn Modules are connected to the Apache core all the same way.
- Modules since they do not know directly about each other must pass all in formation back to the core and then the core sends that information to another appropriate module through the use of the HTTP_REQUEST component of the Apache Core. This in turn does not allow any changing of the stable Apache Core, but also implements a layer of security, because no process can move on without passing the in formation to the Core and the core checks and handles errors through the HTTP_REQUEST component.

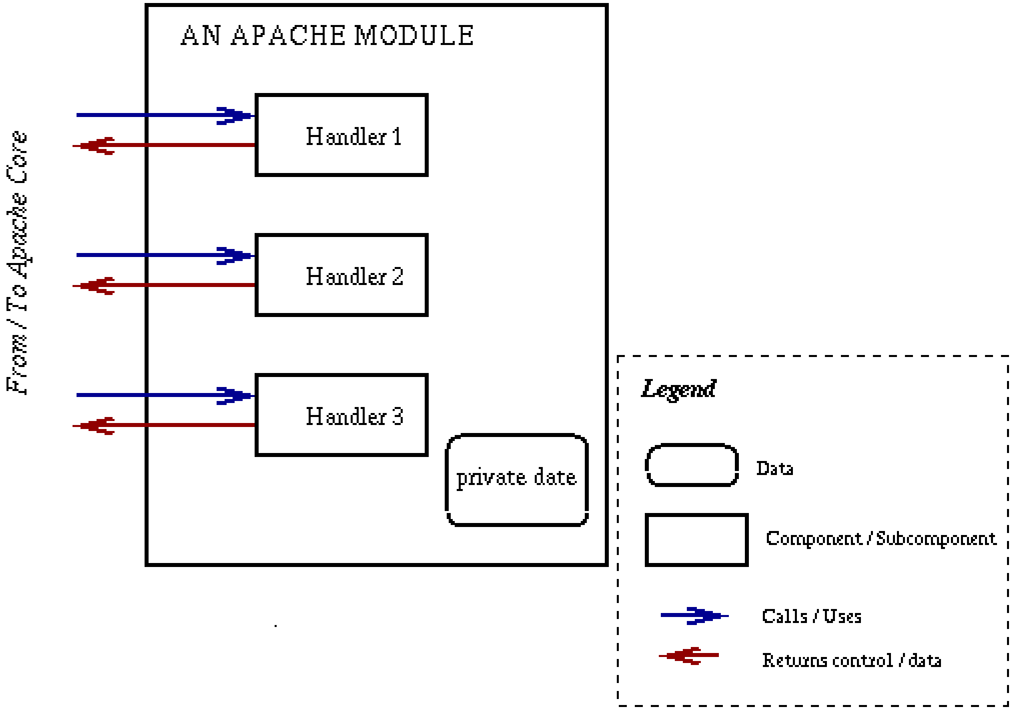
- Apache web server has a modular architecture with a core component that defines the most basic functionality of a web server and a number of modules which implements the steps of processing a HTTP request, offering handlers for one or more of the phases. The core is the one that accepts and manages HTTP connections and calls the handlers in modules in the appropriate order to service the current request by parent and child.
- Concurrency exists only between a number of persistent identical processes that service incoming HTTP requests on the same port. Modules are not implemented as separate process although it is possible to fork children or to cooperate with other independent process to handle a phase of processing a request.
- The functionality of Apache can be easily changed by writing new modules which complements or replace the existing one. The server is also highly configurable, at different levels and modules can define their own configuration commands.

Modules Continued

- One cool thing about Apache that makes it robust and allows for better speed is the fact that, Apache allows for initialization of modules Dynamically. So not every module is started when the server starts up which really allows for a giant speed boost.
- So what this allows is Apache to only initialize the modules that it needs at that moment. Which allows requests to be processed a lot faster than usual.
- Modules have something inside them that are called Handlers.
  - Handlers: A handler is for Apache the action that must be performed in some phase of servicing a request. For example a handler that requests a file must open the file then read the file then send it to the Apache core to then be sent to the client. Handlers are defined by the modules depending on when they are needed to fulfill a request then the Handlers are the ones that send back the processing from the Apache Module to the Apache Core HTTP_REQUEST component

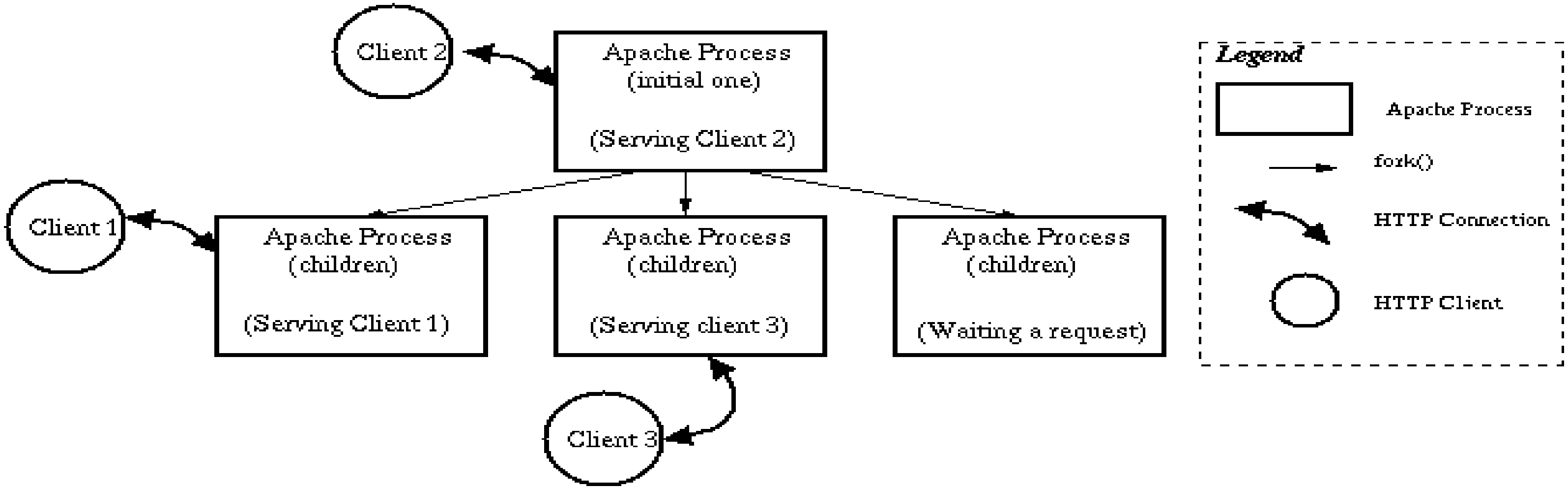
Modules Continued
More about Handlers

- Overview of the Handler system within an Apache Module. As you can see the Handler does what it needs to do to fulfill a request then the sends that process back to the HTTP_REQUEST component of the Apache core in order to be sent to another module for processing or back to the client.



- Module Configuration
- If you are using a static configuration of Apache, choose the modules you wish to incorporate with care. Using static mode comes at a price — the more modules, the more memory you use. Thus, a forked multi-processing module can have a significant effect on the machine's memory requirements.
- Note that some items are automatically included, so you'll need to explicitly enable and disable needed modules. Also remember to include any third-party modules (e.g., authentication, PHP, or mod_perl), the Web service requires. Use configure --help to get a list of the available options.

Concurrency in Apache



Apache provides access to two levels of concurrency. The concurrent processes executing truly simultaneously, in the case that they run on separate processors, as in the case of separate processes running on a multitasking system. As well, if multi-threading is supported by the operating system, a default of up to 50 threads is allowed for each process.

- Each request that the server receives is actually handled by a copy of the http program. So rather than creating a new instance copy when it is needed, and destroying it when a request is finished, Apache maintains at least 5 and at most 10 inactive children at any given time. The parent process runs a periodic check on a structure called the scoreboard, which keeps track of all existing server processes and their status.
- If the scoreboard lists is ever less than the minimum number of idle servers, then the parent will spawn more.
- If the scoreboard lists more than the maximum number of idle servers, then the parent will proceed to kill off the extra children.
- When it receives a request, the parent process passes it along to the next idle child on the scoreboard. Then the parent goes back to listening for the next request.

- When doing the parent and child request there is a limit which by default is set to 256 of the total number request at one time. The default settings was programmed by the creators for the server was to pick in order to keep the scoreboard file small enough so that it can be scanned by the processes without causing overhead concerns.
- Since the number of requests that can be processed at any one time is limited by the number of processes that can exist, there is a queue provided for waiting requests. The queue waiting list was mentioned to be when the parent passes a request to a child, which was idle, then the parent returns to receive next request. The maximum number of pending requests that can sit on the queue can reach somewhere in the 400-600.
- Apache server architecture was designed to maximize one connection. The uses the persistent connection to allow multiple requests from a client to be handled by one connection, rather than opening and closing a connection for each request. The default maximum number of requests allowed over one connection is 100. The connection is closed by a timeout.

Security Perspective

Security Perspective

Hopefully with the presentation of the Apache server. You can now know what multi-processed Web servers are and how Apache qualifies to be one.

Another important aspect that is worth discussing is how processes work. Two types of processes are invoked, *parent* and *child.* The parent process is the main process from which several child processes are invoked.

Now how does this imply to security right?

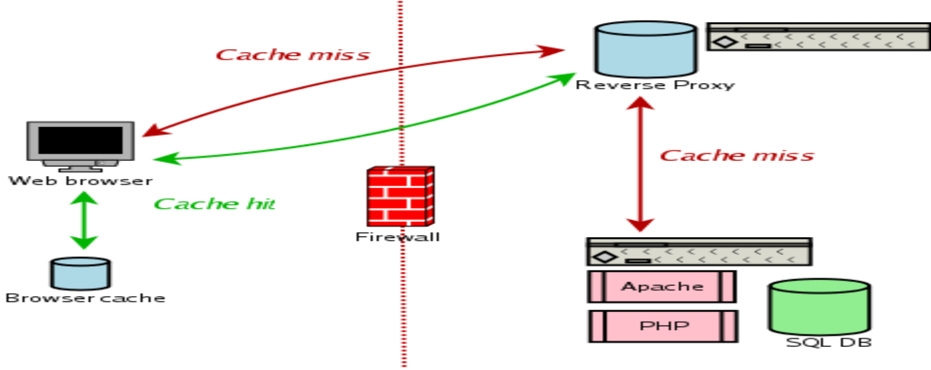
Whenever a request is sent to the Apache Web server, the parent process receives the request. Then the parent process forwards the request to one of the child processes. The child process then handles the request by responding to it. This behavior is supported for a valid reason: security.

Security…

- The root user (SuperUser has all privileges like an Admin) initiates the parent process of the Web server in UNIX systems. The root user, is the most powerful user on the UNIX system. For security reasons, the parent process doesn't directly process the requests sent by the clients.
- For example, if a client sends a request with malicious intentions and the parent process handles the request. The parent process will being run as a root user (SupperUser) will have all necessary rights to perform any operation on the computer, thereby making the system server vulnerable. However, if the request is forwarded to a process that has restricted permissions on the computer, no harm can be done which is handled in the child process. This is because child processes are run as users with restricted privileges.

Common Media Hack

- Providing any kind of system information to a hacker could potentially provide a hacker with the ammunition they need to break into your server. The less a hacker knows about the configuration of a system, the harder it is to break into. The issue that many business uses Apache server to host websites.
- One of the most common exploits used by hackers is to "take over" a service running on the server and use it for their own purposes. For example, gaining access to a mail application via an HTML form-based script. Hacker would use the mail server to send out spam or acquire confidential user information.
- While not a direct security threat, a poorly written application can use up a system's available resources to the point where it becomes almost completely unresponsive.



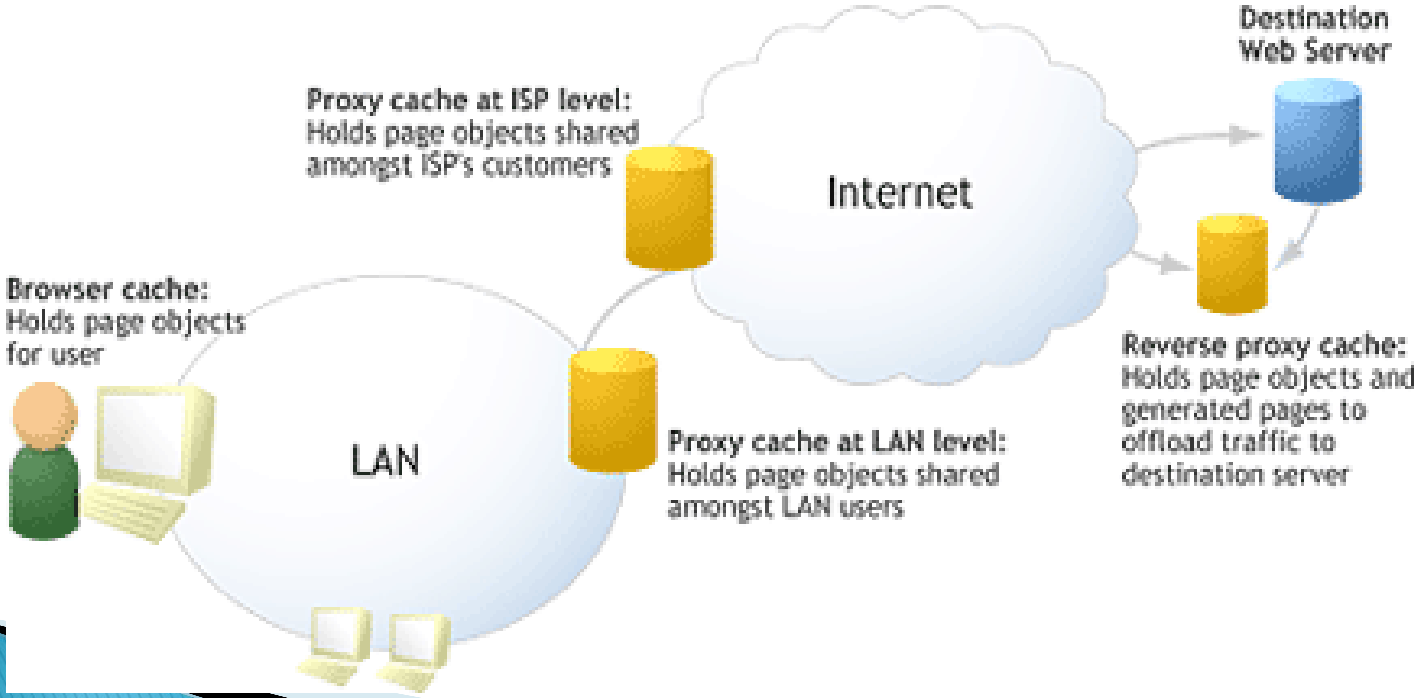
- Safe programming layer to prevent buffer overrun exploits and sandboxing to limit resource consumption
- Efficient output data stream buffering
- Apache compatible configuration
- Extensive logging and debug trace
- The apache server architecture was design to be highly customizable for business programmer to modify it for their needs.
- The configuration files flexible permits to customization of the modules of the apache server. Modules can recognize from configuration files and will be called when such commands are found through the proper procedure.

- The Apache server has a fully functional architect with an intension for security from the root user privileges (Super user).
- The server asks the web browser of the client for the user and password to access the server. For example, Internet Mozilla Firefox web browser has a feature to store username and password. The risk of the server automatically ask for the password, anyone can access the programmers account once on the client’s machine.
- password is send over the network not encrypted but "unencoded"
- password is not visible in the clear, but can easily be decoded by anyone who happens to catch the right network packet (“sniffers in action”)
- this method of authentication is as safe as telnet-style username and password security

Caching it.

- The reason for discussing the process of caching is because, by definition, cache is the temporary storage of frequently accessed data in higher speed media (such as SRAM or RAM) for more efficient retrieval of data through a process. Web caching stores frequently used objects closer to the client through browser, proxy, or server caches. By storing string objects closer to your users in order for the user to avoid making several same trip to access the server. By doing this, this reduce bandwidth consumption, server load, and most importantly, latency.
- The caching method and process is not just for static sites, but as mentioned before, even dynamic sites can benefit from caching. Streaming video request was an issue Graphics and multimedia typically don't change as frequently as XHTML files. Graphics that seldom change like logos, headers, and navigation can be given longer expiration times while resources that change more frequently like XHTML and XML files can be given shorter expiration times.

The diagram below illustrates a key point in the discussion of cache. Caches are found on the Web in many places and are constantly trying to hold your site content whenever possible.


Destination Web Server
Proxy cache at ISP level: Holds page objects shared amongst ISP's customers
Internet
Browser cache: Holds page objects for user
LAN
Proxy cache at LAN level: Holds page objects shared amongst LAN users
Reverse proxy cache: Holds page objects and generated pages to offload traffic to destination server

- There are two main stages in mod_cache that can occur in the lifetime of a request. A mod_cache is a terminology that represents the URL mapping module, which means that if a URL has been cached, and the cached version of that URL will not expired, the request will be served directly by mod_cache.
- When caching is locally generating content to ensure that UseCanonicalName is set to ON can dramatically improve the ratio of cache hits. The hostname of the virtual-host serving the content forms a part of the cache key. With the setting set to (ON) on the virtual-hosts with multiple server names or aliases will not produce differently cached entities, but instead, the content will be cached as per the canonical hostname reason for that is that caching is performed within the URL to filename translation phase. Cached documents will only be served in response to URL requests.

Conclusion

- **Apache**
  - architecture, modules and handlers
  - directory structure; configuration files & directives; running
  - access control; authentication
  - Passing Data; Security
  - cache control