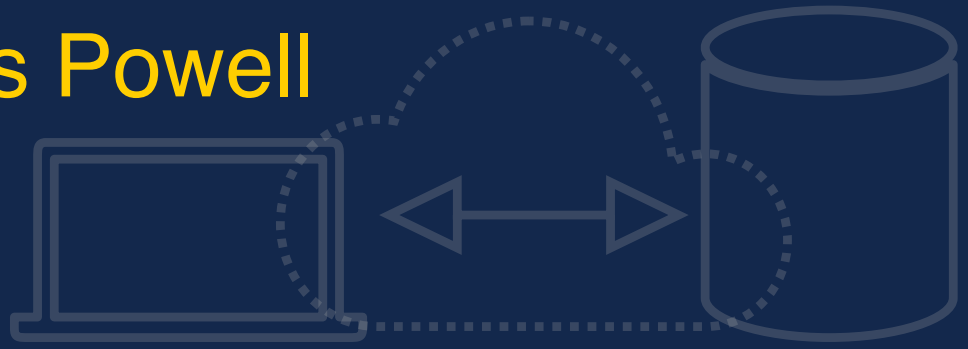
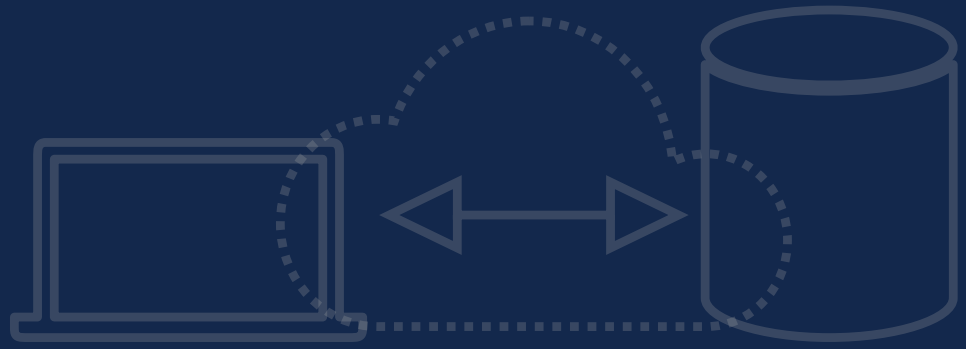
Web Servers Overview

with Thomas Powell



The Programmer – IT Divide

- The Obvious Why – Division of Labor
  - IT: Don't touch those knobs! Put them in a sand box so they don't hurt things, etc.
  - Dev: I wanna try new framework/software X. I am too busy programming to read logs / install patches
- Mind the gap!
    - Security
    - Performance
    - Huge waste of time and money because we don't know enough to meaningfully interact



Hmm…something smells fishy

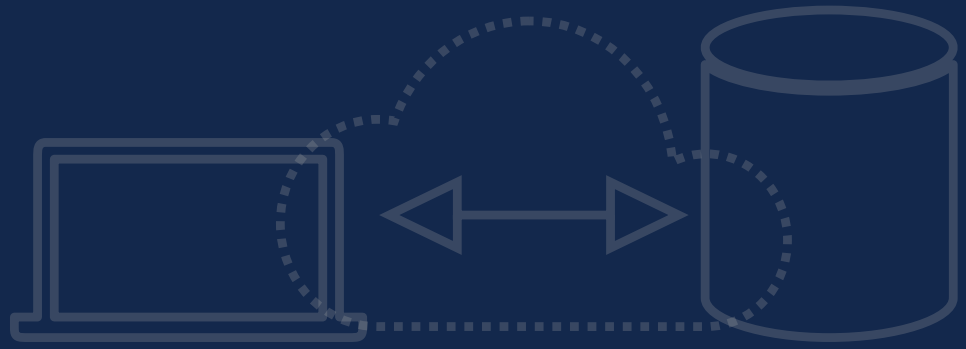
versus

http://www.channelregister.co.uk/2010/07/29/cray_1_replica/

- "An irony is that the resulting scale model Cray-1, …, is probably more powerful than Cray's original near 40-year-old design."
- Today you need scads of PCs to serve simple Web apps?

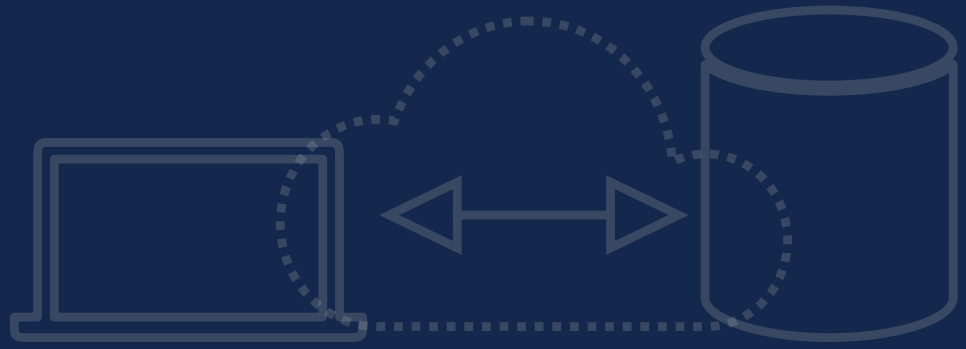
The Role of a Web Server

- A box and a service
- Web servers serve various *resources*
  - As file (document) servers
  - As application front ends
- A physical server can of course serve many protocols (SMTP, FTP, etc.) or may be protocol specific
  - Web Servers are of course HTTP servers
  - But that's easy, right?

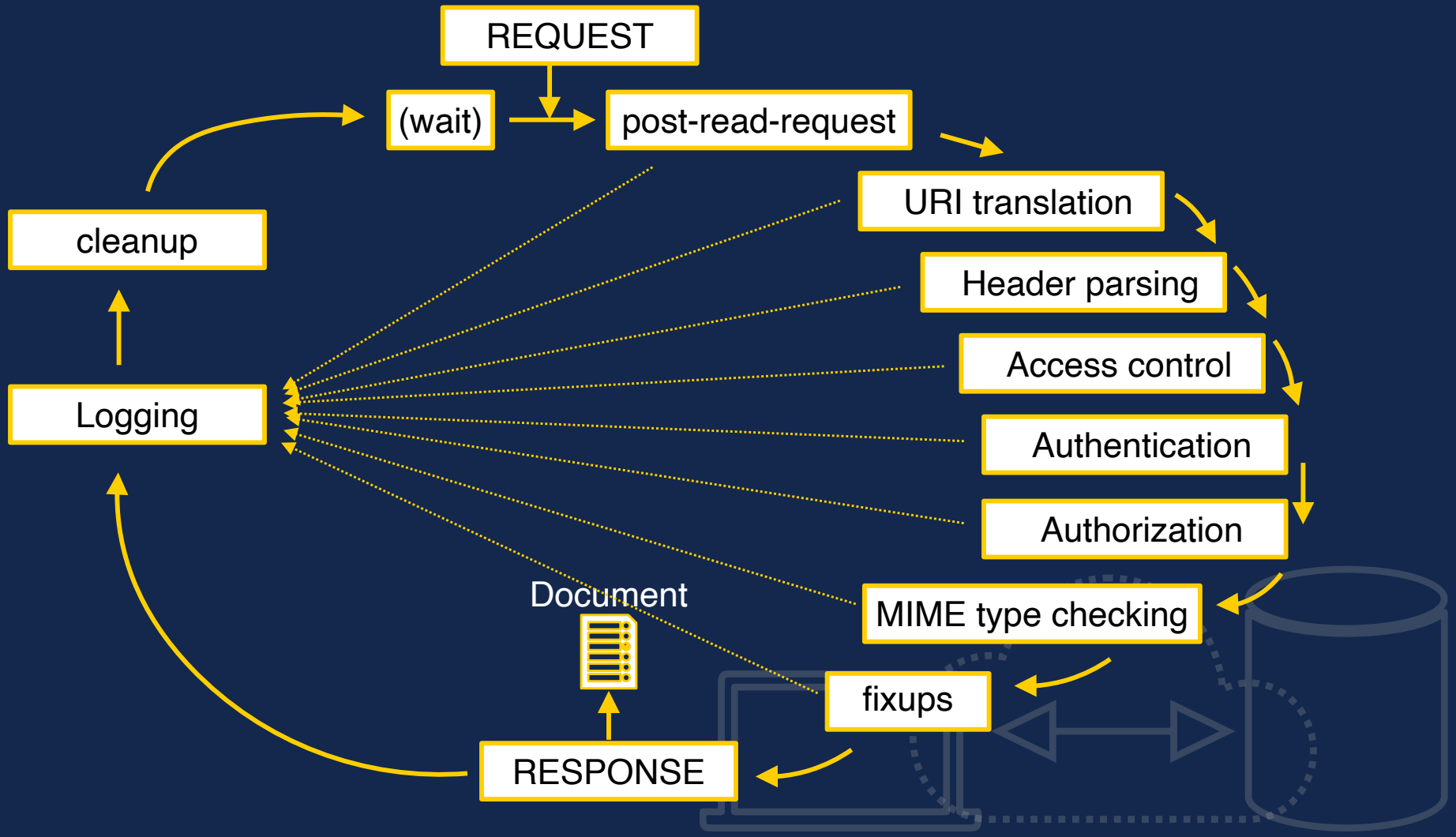


Roughly how web servers work

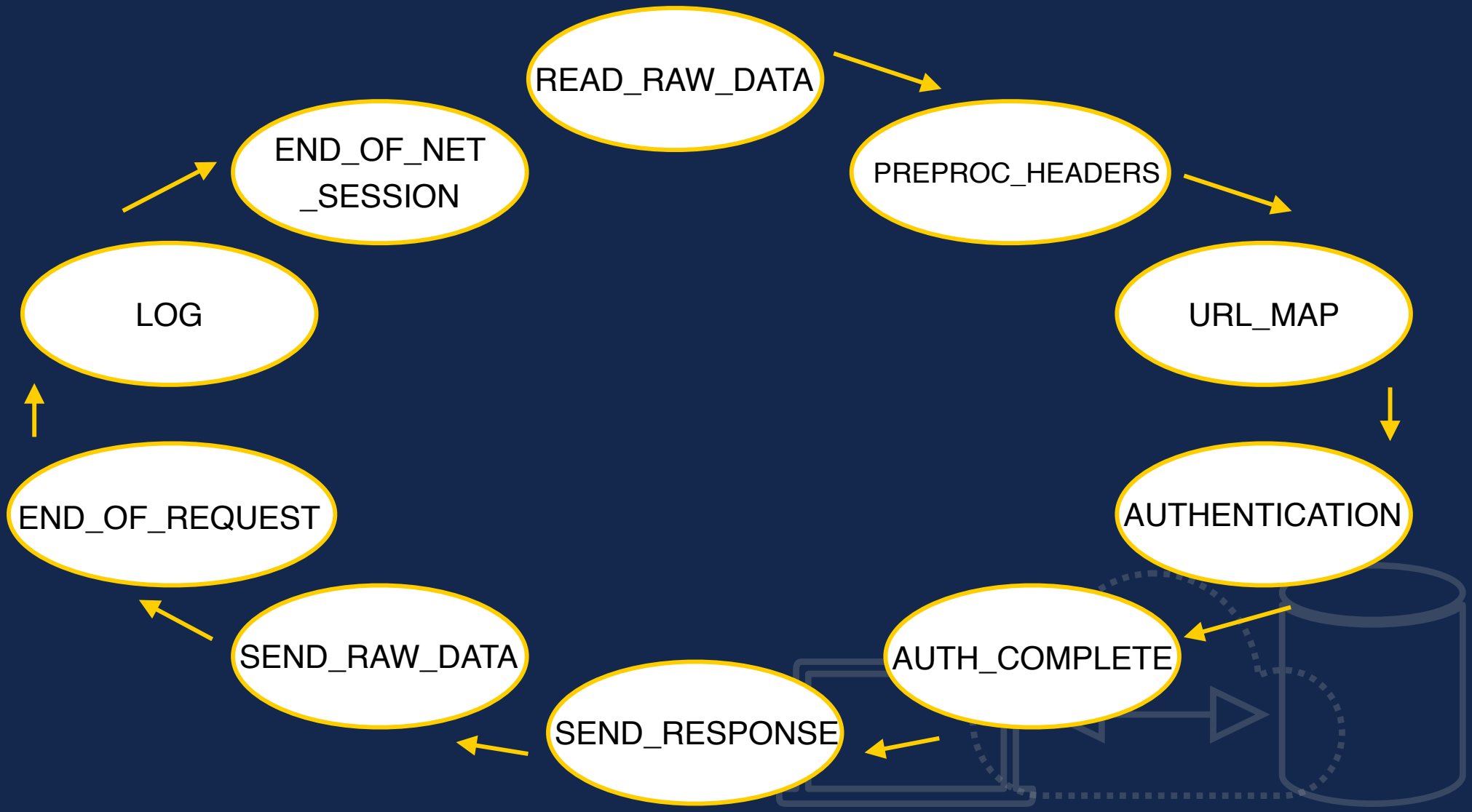
- Listen for requests
- When a request comes in
  - Decode the request
    - What resource? Get the headers? Etc.
  - Is that allowed?
    - Yes -> move on
    - No → deny (Skip to log)
    - Maybe -> Authenticate
  - After authentication if needed (or just public)
    - Get resource or Run Resource
    - Map MIME
    - Return data
  - Log
- Keep listening until shutdown or crash



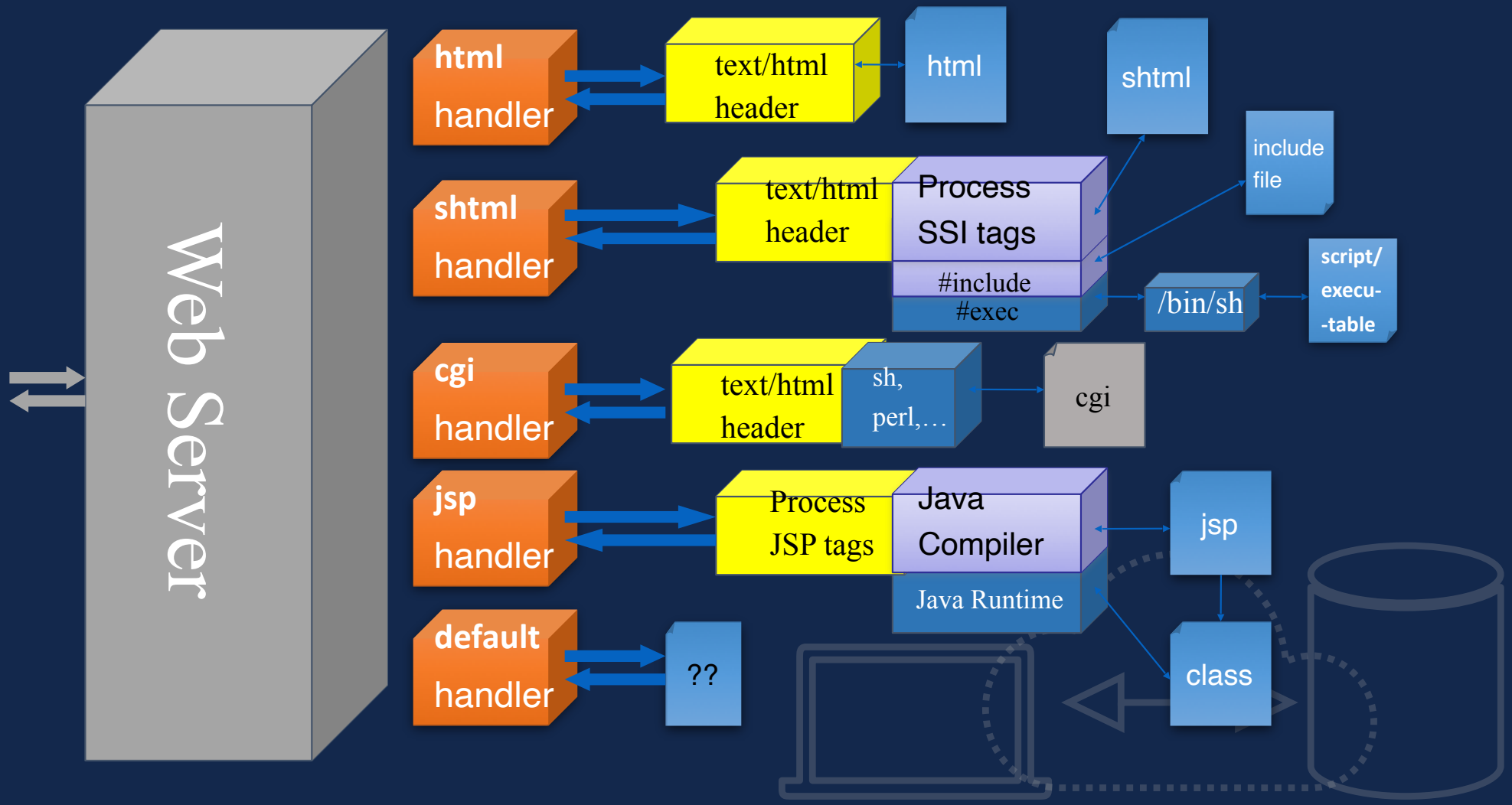
How they Work - Apache Request / Response Lifecycle



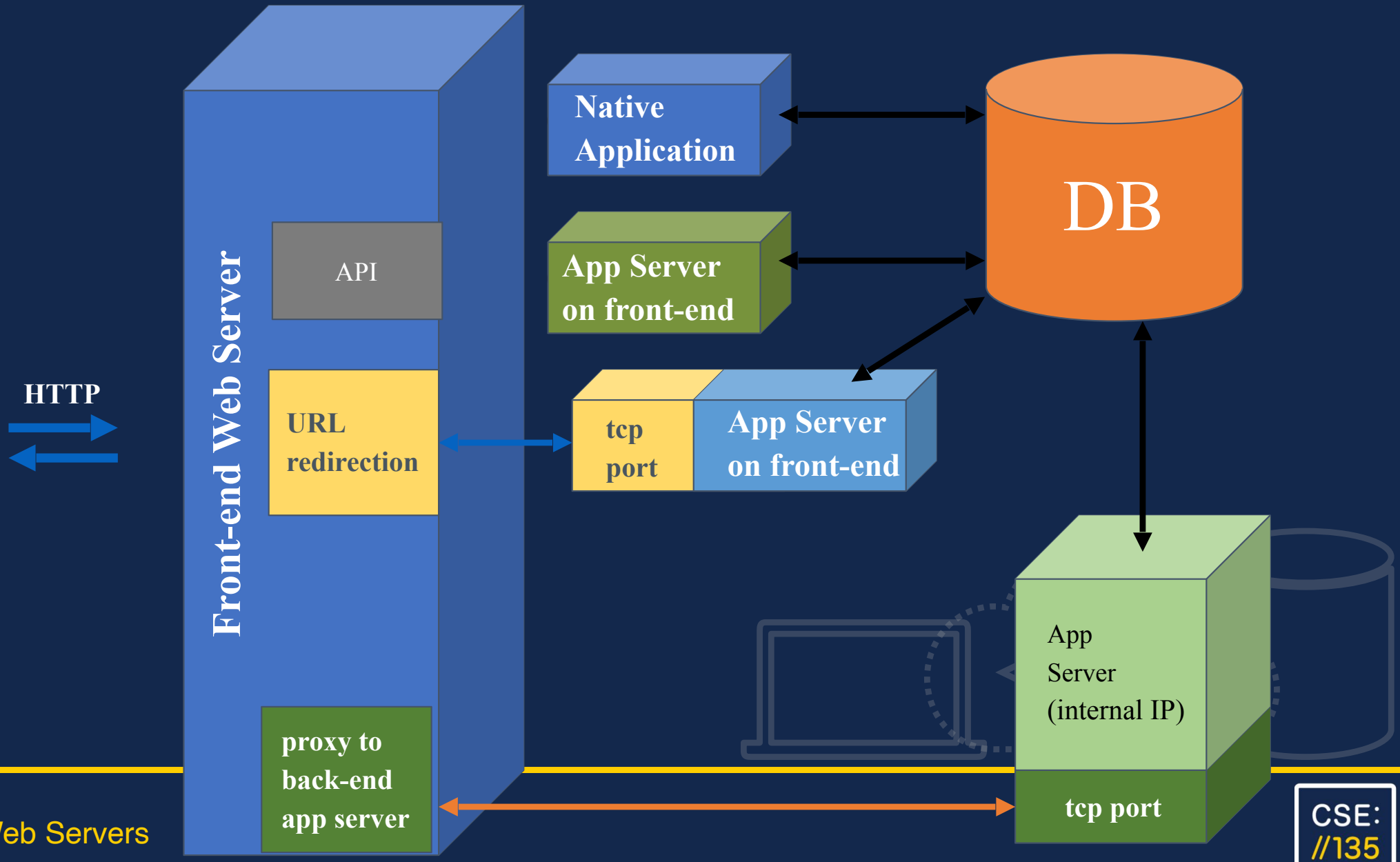
How They Work - IIS Request / Response Lifecycle



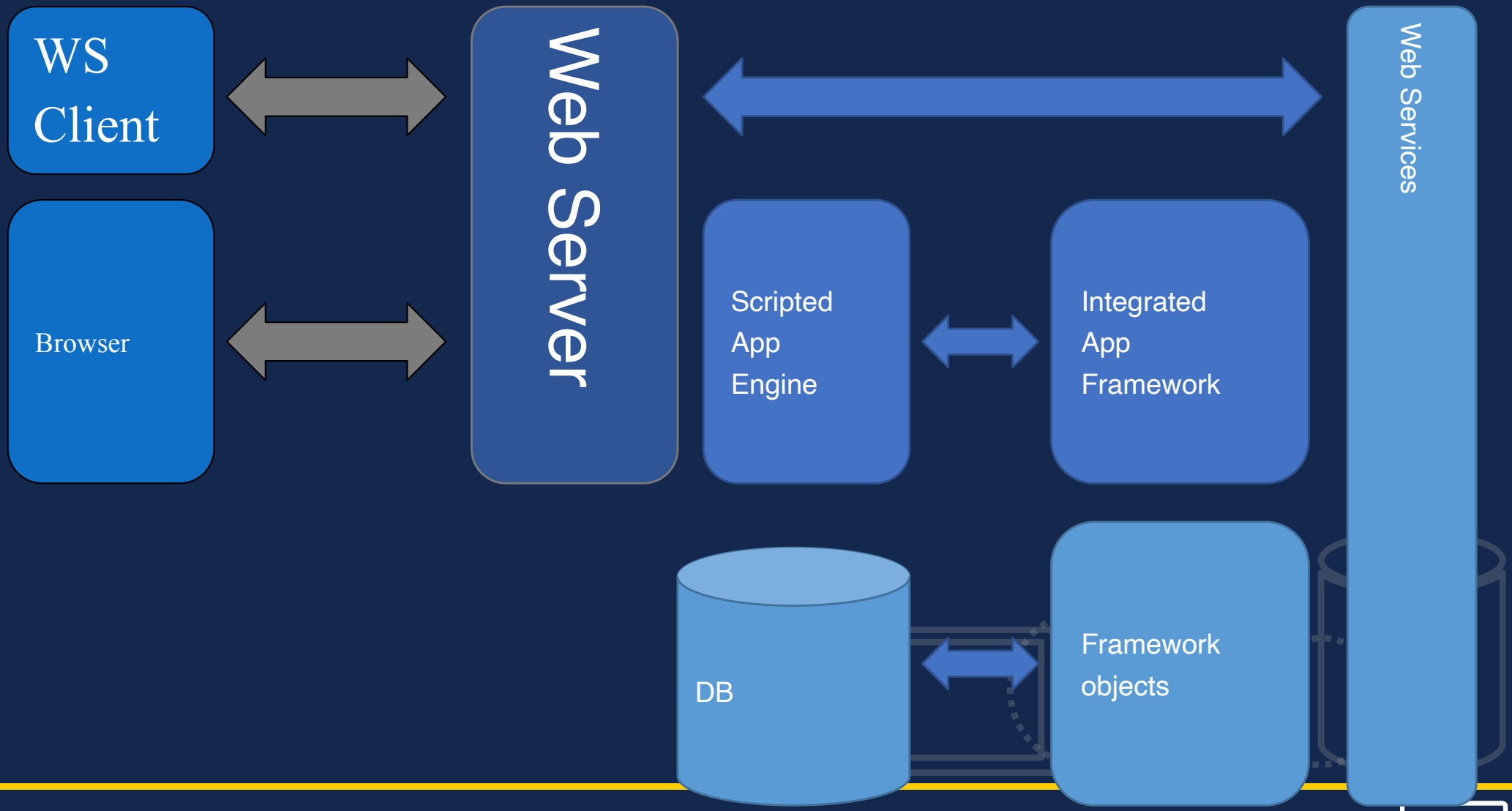
Handlers and Resources



Web Server and App Server Interfaces



Modern Web Application Architecture



Yeah they are kind of at the heart of it

- Your Web Server goes down it is going to take everything with it
- Your Web Server can be a bottle neck
  - So can your database
  - So can your App server
  - Let's not forget the network – that's the real trouble

- Clearly the Web Server effects Apps and Apps effect the Web Server shouldn't we know about that and deal with it? – Yes see HW1 ☺

Planning Web Server Deployments

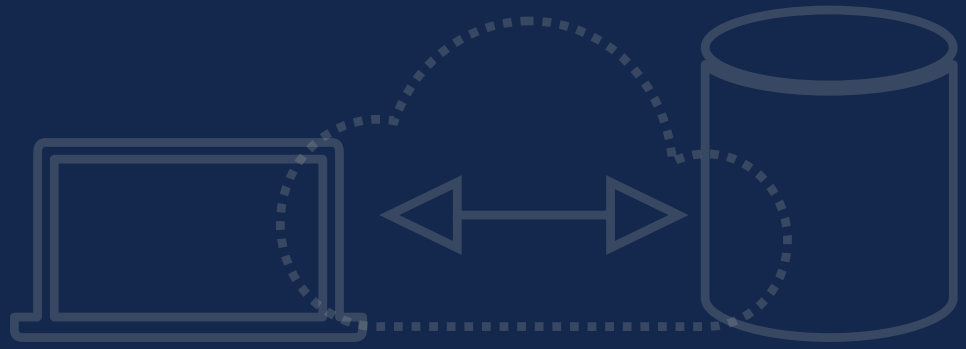
- Major issues to consider when planning a Web server or Web site deployment
  - What is the appropriate form of Web hosting?
  - What type of server software will be used?
  - What are the sizing requirements?
  - How will DNS be handled?
- There are no fixed answers to any of these questions
- Planning should be guided by the *goals* of the deployment and should harmonize with the related *business processes*

Choosing Among the Hosting Options

- Host your own
  - Pro: Complete control over the physical box
  - Con: Expensive and difficult to maintain well
- Hosting provider schemes
  - Dedicated Server
    - Pro: Control without the hardware purchase
    - Con: Must manage the box – remotely
  - Co-located Server
    - Pro: Admin control of entire box
    - Con: Must purchase box and manage remotely
  - Virtual Hosting
    - Pro: Cheapest and easiest to maintain solution
    - Con: Server is shared, admin access limited
  - Now we have virtualized machines and even “serverless” styles or even functions as service

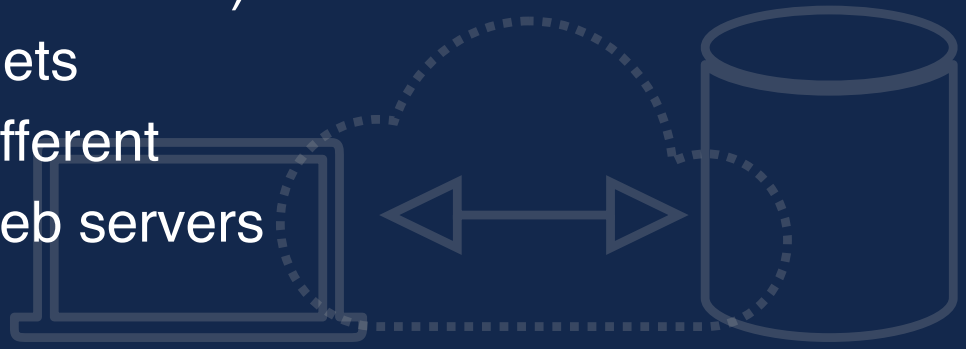
Cloud Computing and Serverless!?

- Cloud computing has commoditized servers to the point of instant provision and scaling of even pre-defined OS / App configurations
  - We can go farther! Infrastructure as service then to Platform as service then to …
- Serverless thought patterns say just upload code and it works and scales. No need to consider servers or administration just focus on coding.
  - This is very enticing but like anything has downsides as well in terms of flexibility restrictions



Choosing Server Software

- Apache, Ngnix, others
  - Best reputation historically until Ngnix came along!
  - Fun with usage stats for public sites (ex. Netcraft)
  - Features extended & refined via modular and open development model
  - Strong administrator ethos = well managed boxes
- IIS
  - Included in the Windows server environment
  - Security black-eye (or is it from the OS?)
  - Favored in business and intranets
  - Servers are more a like than different
- Beware of sectarian quarrels about Web servers



Choosing Server Software

- There actually is more than just than these
    - Most are high-performance Web servers used by some really large sites or dev servers
    - Tries to provide APIs from both main worlds
  - Many app servers (Tomcat, Zope, etc.) include Web servers (or Apache) as part of their distribution
- Careful nodejs (JS), Perl, Python, etc. are used to serve HTTP directly but those are very simple Web servers and may reinvent things
  - Be warned direct node HTTP serving is actually dangerous

Choosing Server Software, cont.

- In real world, usually a conditioned choice if not a forgone conclusion
  - Biggest single factors are type of deployment and prior commitment to an underlying OS
  - Apache on UNIX and Linux predominates in universities, research institutes and for virtual hosting setups – has majority of hosted domains
  - Microsoft's IIS software is common in Windows focused firms. You may find it commonly internally more so than externally

Sizing a Web Server

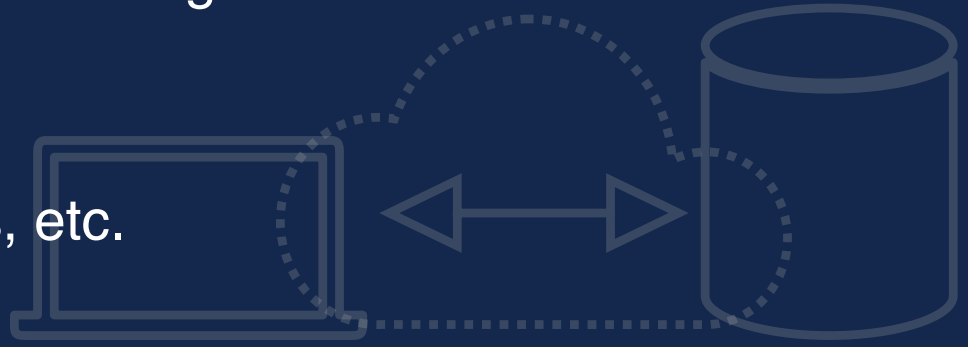
- Sizing is process of determining the physical resources required to meet anticipated demand
- Processing power and memory are not typically a problem for the Web server
  - Basic HTTP server job of fetching files is not processor intensive
  - Resource constraints on the box probably an effect of other server-side mechanisms
    - Automated session management by app servers
    - Manipulation of large database queries
    - Lots of non-optimized code in Web applications
    - Network concerns concerns! (next slide)

Sizing a Web Server – Network Bottlenecks

- Network bottlenecks
  - Available bandwidth should accommodate max HTTP operations (“hits”) under peak load
  - Could you figure out given an average file size a peak load for a 10mbps connection?
    - So then would file size be an important consideration for Web design in a high traffic site?
  - Bandwidth sizing should be adjusted based on your actual request frequency and size
    - Assume peaks at triple or more the average loads
  - Also watch out for collisions and overloading of routers, switches, hubs and network interfaces

Common Web Server Tasks

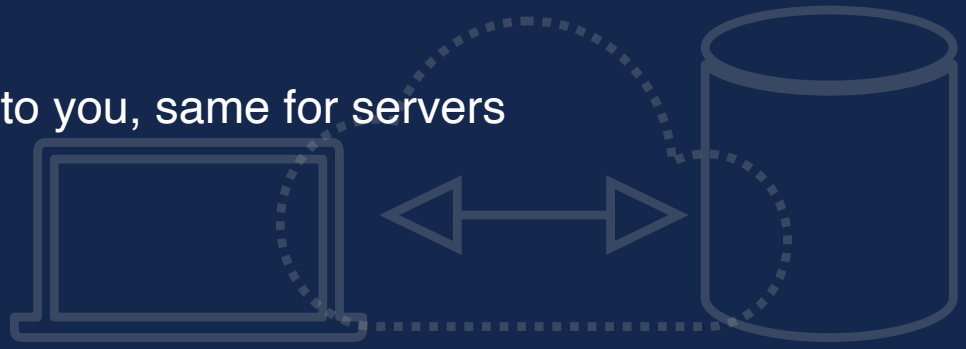
- Set-up / install Web server
  - Done here but usually set name, IP, root directories
  - Define protected directories with basic authentication, etc.
  - Configure error pages for 404 errors, 403 errors, etc.
  - Turn off directory browsing maybe?
  - Small security changes (remove or modify server header)
  - Set-up aliases and other redirects
  - Tune for performance
  - Monitor for security, performance and usage
    - Logs
  - Support Web applications
    - Installation of frameworks, files, etc.



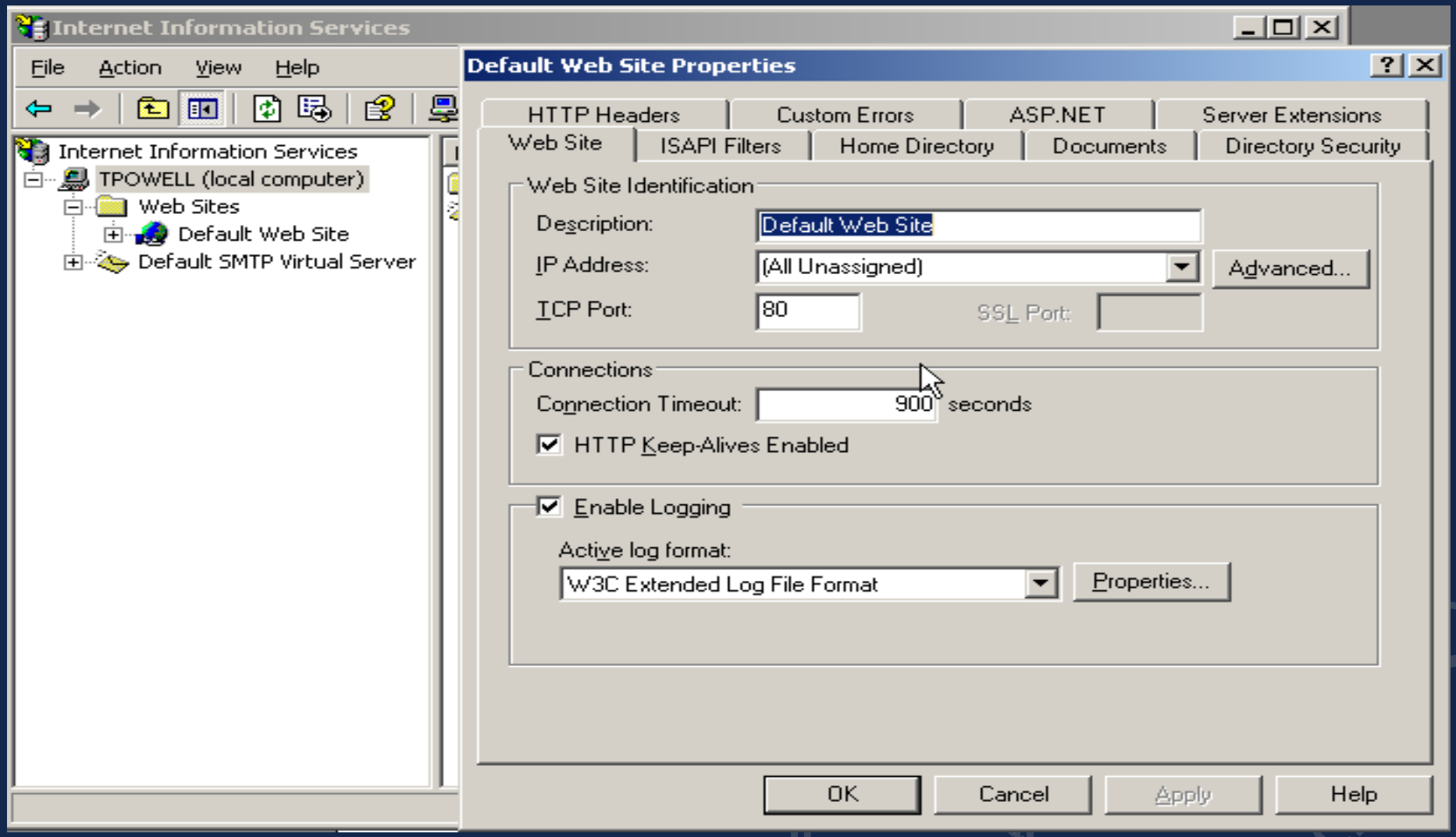
Some Examples using IIS

- The following pictures show common screens for an **older** version of IIS that made it easy to see everything. This is not tutorial this is to show you the sense that ALL web sever have the same abilities. The specifics are the point here the BROAD features and roles is what you want to see to spot **WEB SERVER FOUNDATIONS**
- If we study these screens and then look at an Apache.conf file the ideas are the same. Your HW1 has you touch that compare the screens
- Understand Web servers are just HTTP servers so once you know the HTTP lifecycle what web servers actually do is pretty much the same (speed and ease of admin may change though)

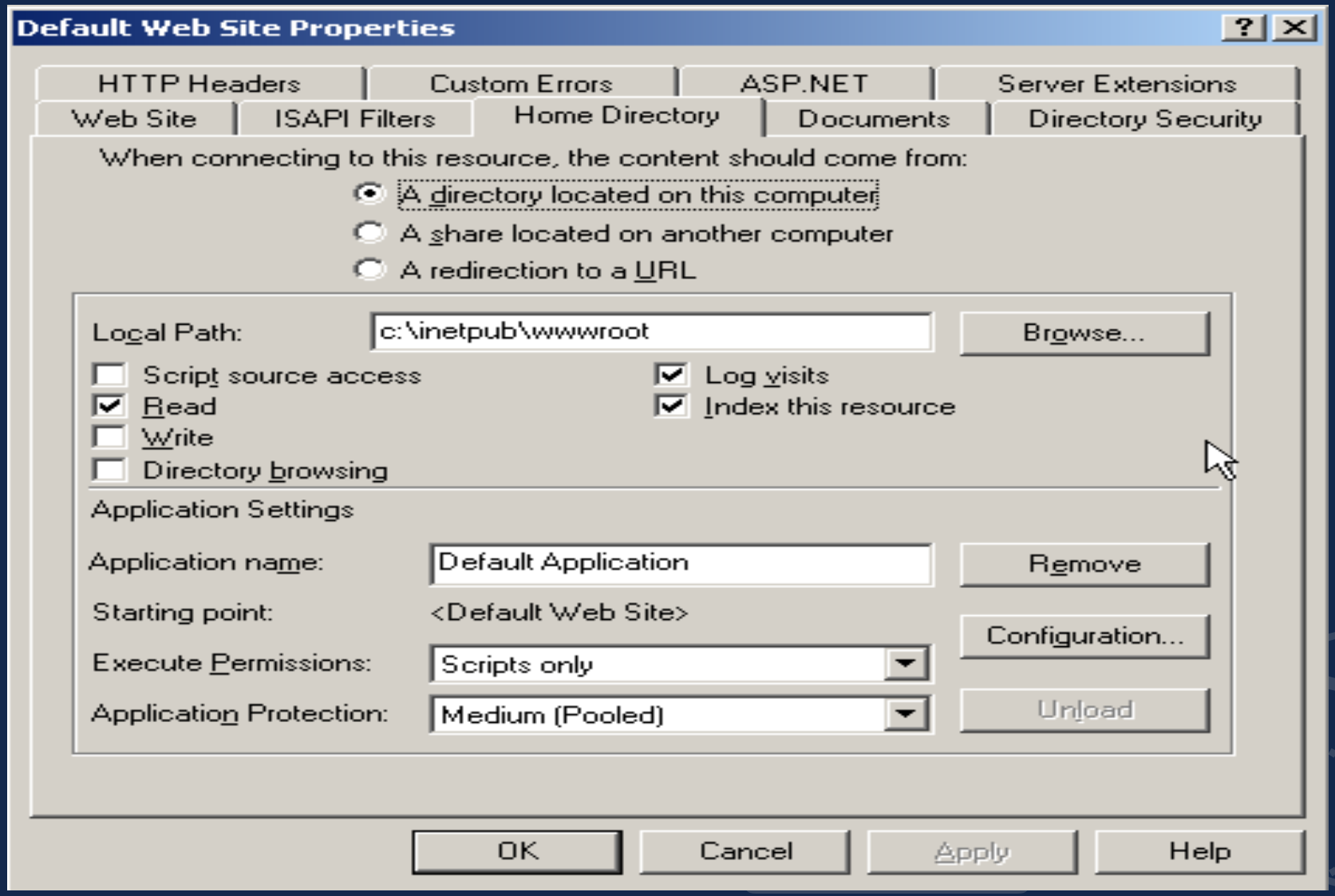
  TL;DR - Chrome, Firefox, Safari kind of same to you, same for servers



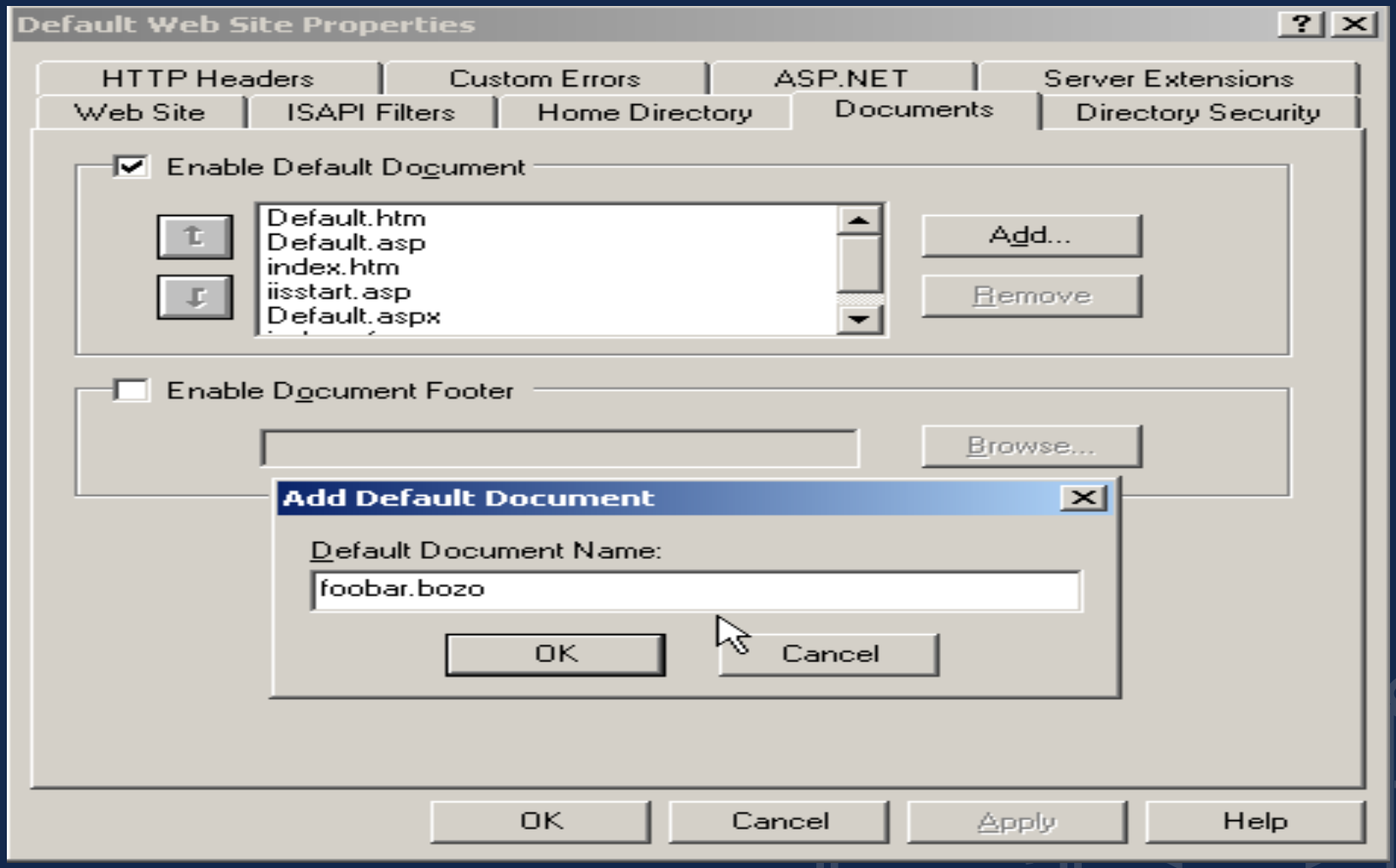
Server Example: IIS - Main Settings Dialog



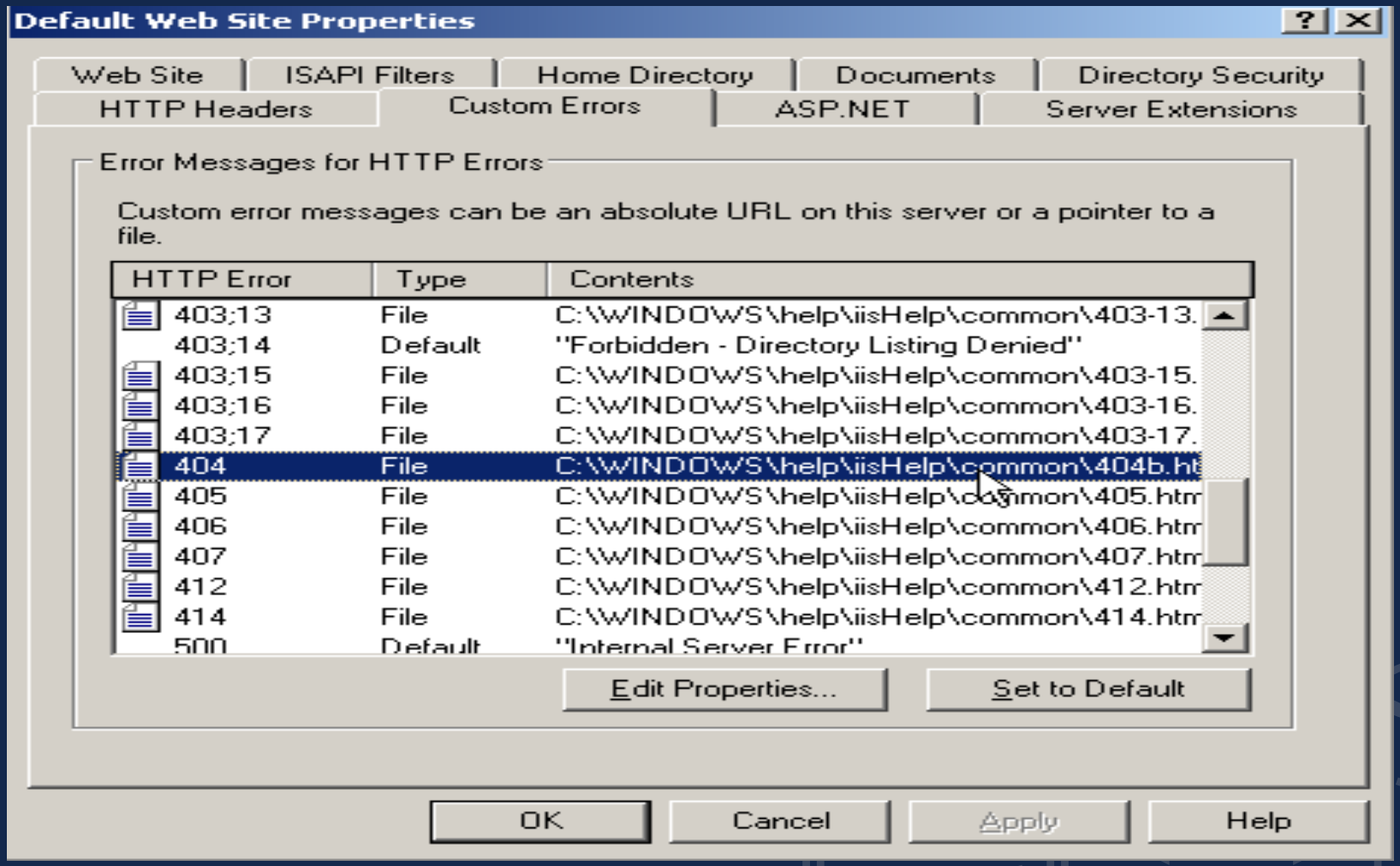
Server Example: IIS - Directory Configuration



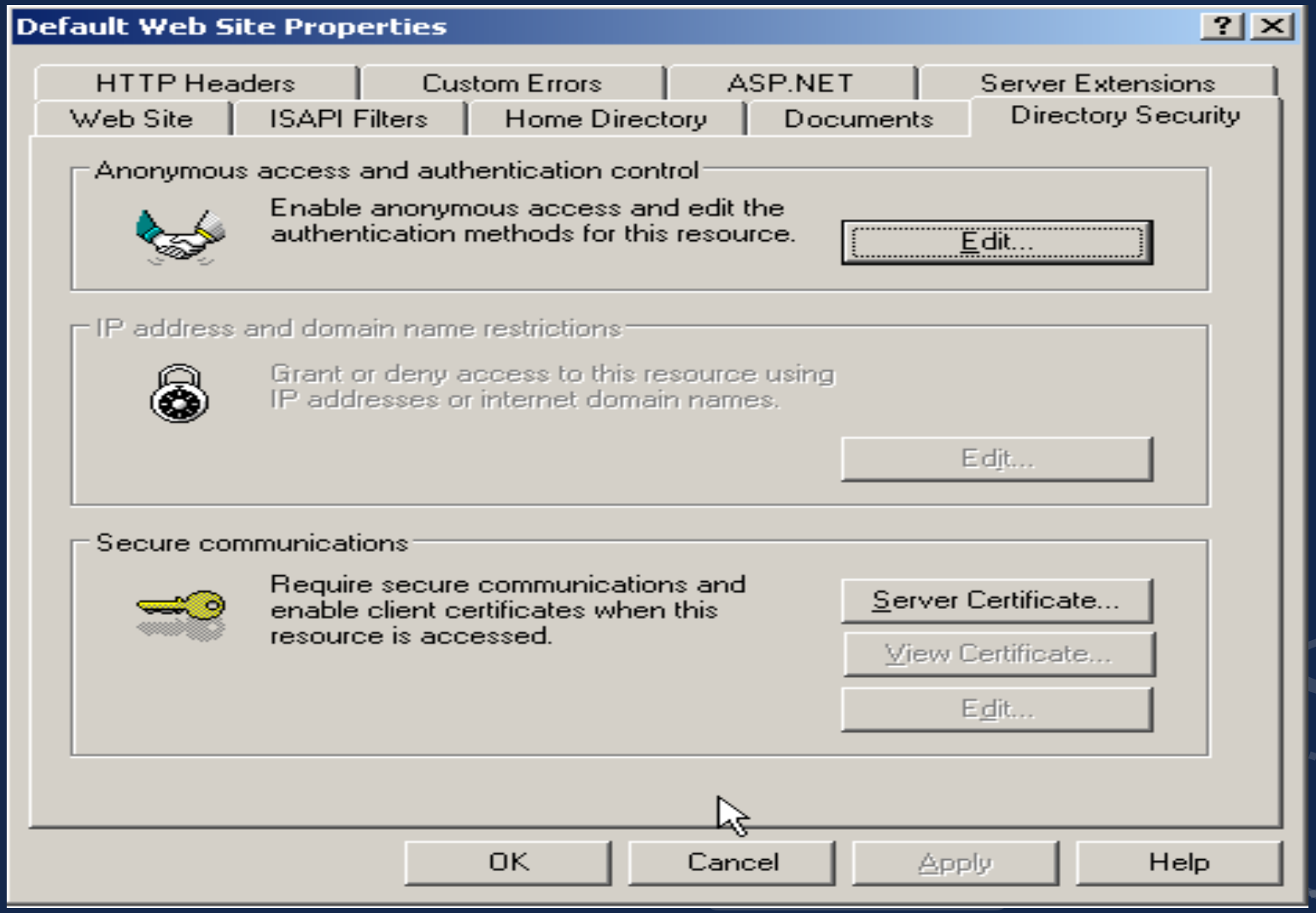
Server Example: IIS - Default Documents



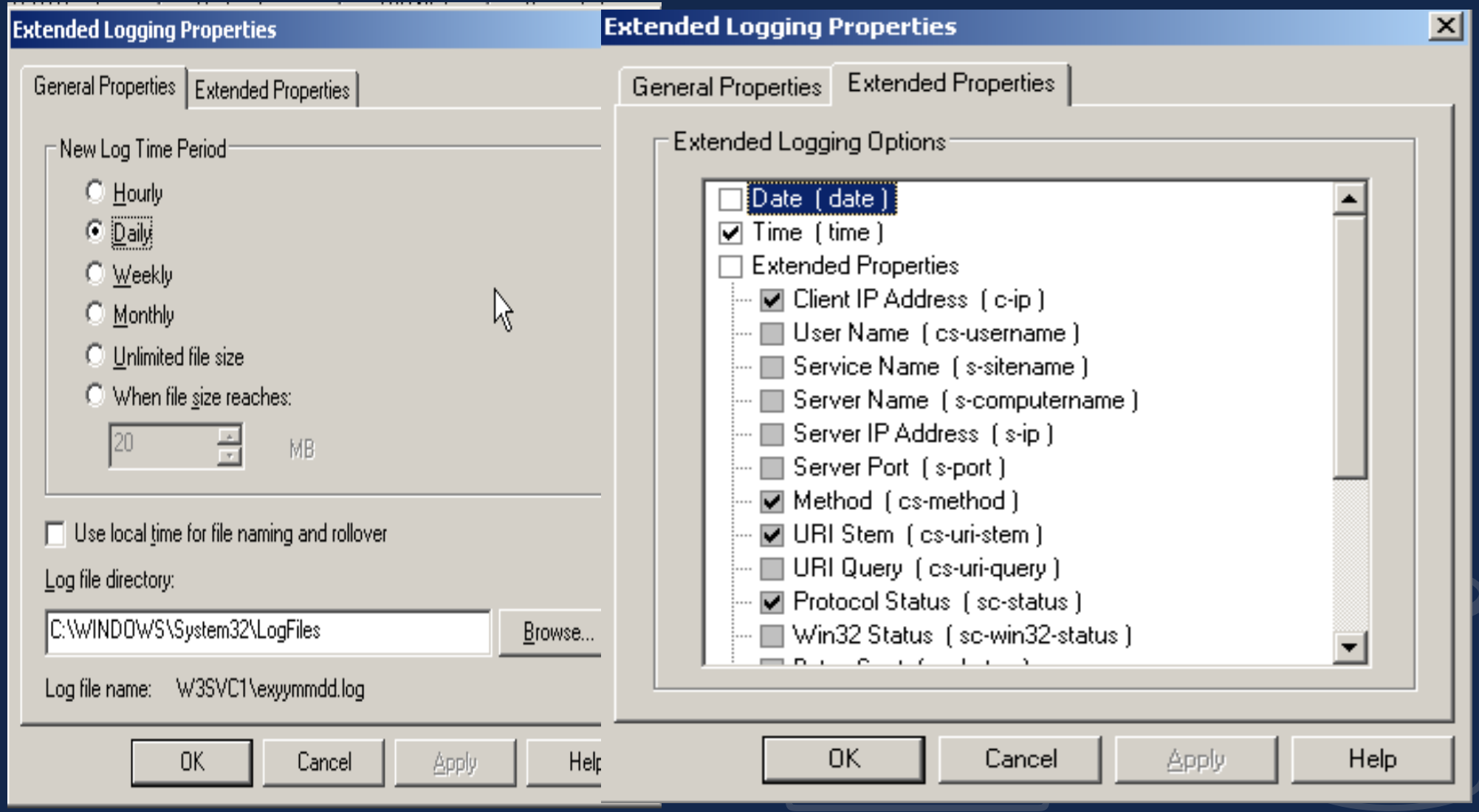
Server Example: IIS - Error Messages



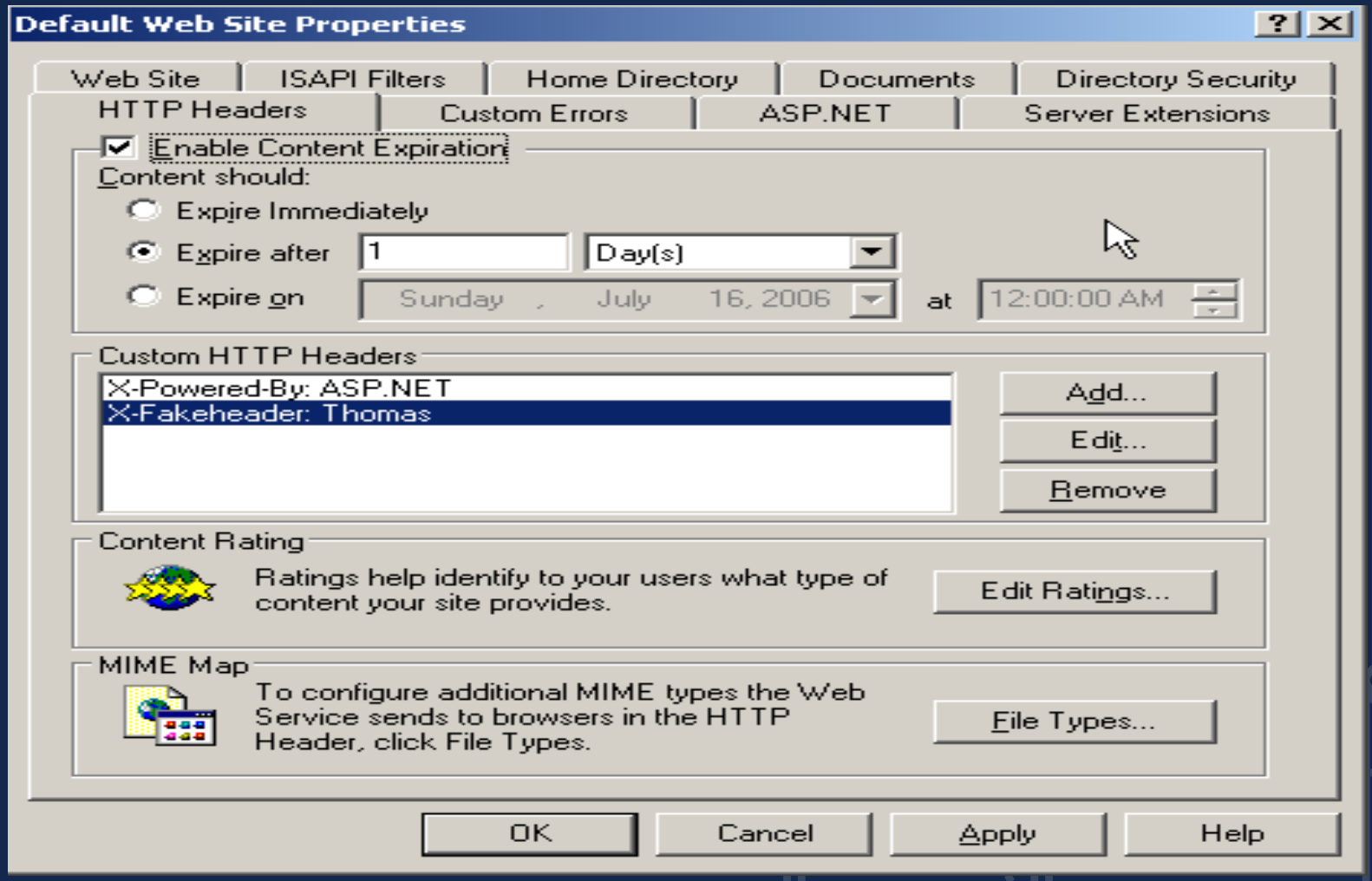
Server Example: IIS - Access Control



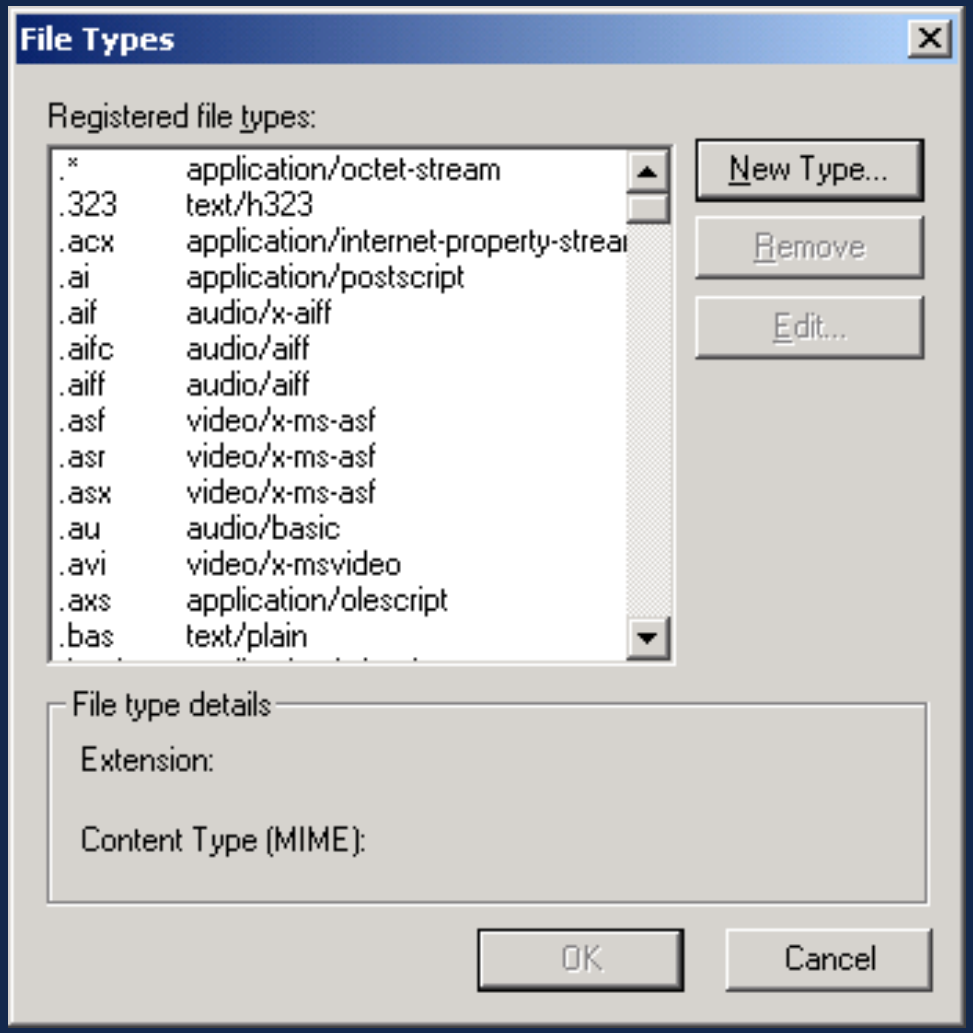
Server Example: IIS – Log Files



Server Example: IIS – Headers and Misc.

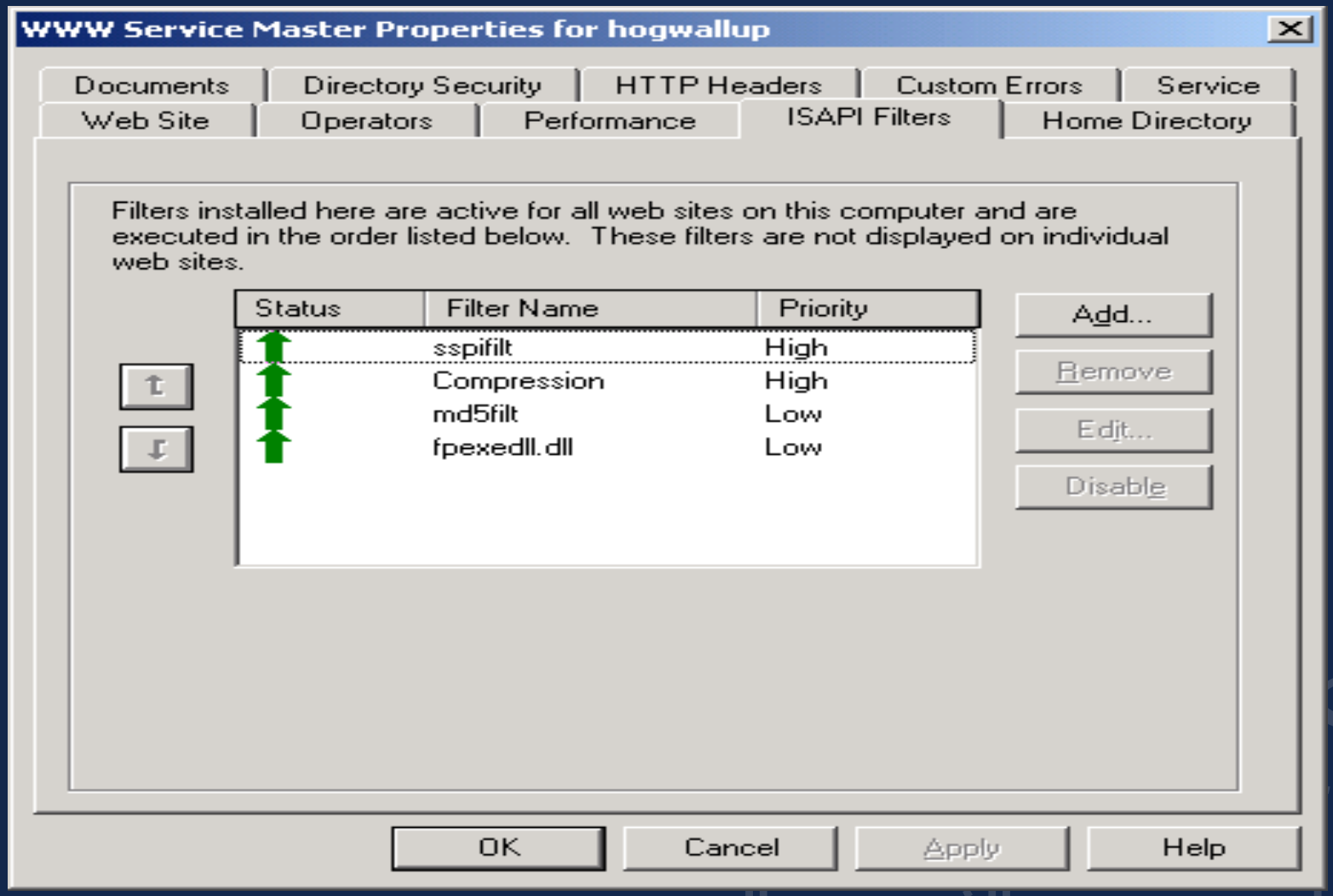


Server Example: IIS – MIME Types



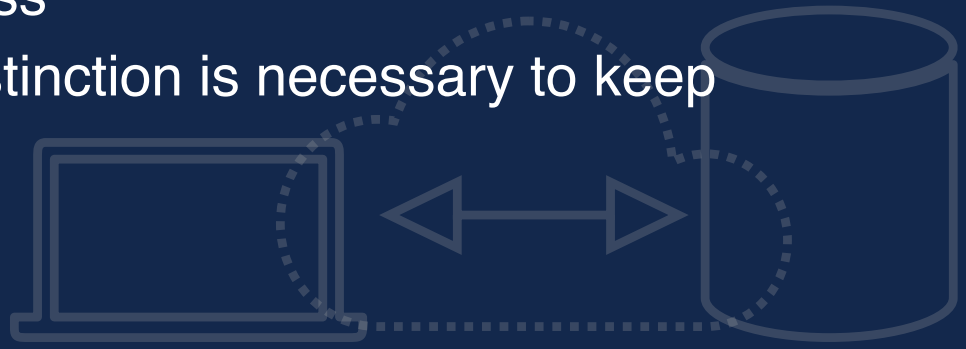
- So you see here that File extensions map to MIME types which is really what the user agent cares about when deciding what to do with an HTTP response
- File extensions are often the key to how a server-side scripting engine works

Server Example IIS – App Filter List



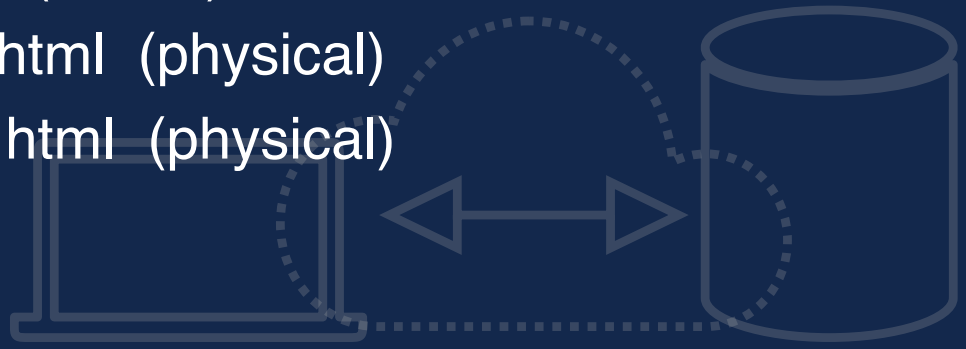
Organization- Virtual and Physical Site Structure

- Think of a site as having not one structure but two – *virtual* and *physical*
  - Virtual structure is described by the URLs used to request resources from the site
    - This is the public view of the site – the site as visitors will see it when they browse to it
  - Physical structure is the organization of the files and directories in the file system on the host machine's hard disk
    - This is the private view of the site seen only by you and those users you choose to give access
  - It will become obvious why this distinction is necessary to keep things straight



Configuring Virtual-Physical Mappings

- The Document Root
  - A directory in the file system of the host machine where the Web server looks for the files that constitute the Web site
    - Also called the *root directory*
  - Often given an *index* or *default* document that serves as the *homepage* of the site.
  - Corresponds to the "/" at the end of hostname portion of the URL:
    - http://www.foo.com/index.html (virtual)
    - Ex: /var/www/index.html (physical)
    - Ex: C:\inetpub\wwwroot\index.html (physical)



Configuring Virtual-Physical Mappings

- Notice how the hostname portion of the URL maps to the same place pointed to by the *physical* path that lies to the left of the the "/" representing the document root
  - The URL is virtual to the left of the document root, but it seems to be physical to the right of the document root
- In fact, a URL is purely virtual – there is no guarantee that the path to the right of the document root looks this way on disk
  - Could http://www.foo.com/index2.html map to C:/foo/a/b/c/myfile.html?
  - Sure – you can do this with aliases, redirects, local OS mappings, all sorts of stuff

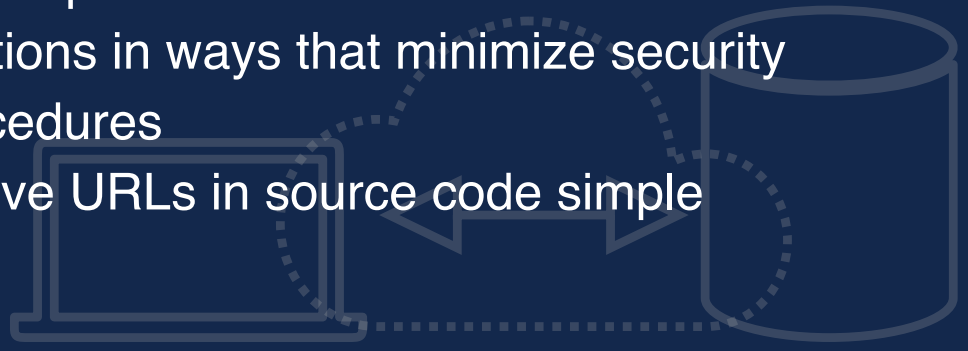
Configuring Virtual-Physical Mappings

- A *virtual directory* or *alias* in the URL path preempts the lookup in the document root
- This extends the virtual structure to the right of (or "below") the root "/" in the URL path

  http://www.foo.com/virtual/index2.html

  /htdocs/physical/index2.html

  - You can (and should) take advantage of this virtual/physical distinction to:
    - Preserve the site's URL scheme even if the physical structure has to change
    - Avoids broken links due to site expansion/revision
    - Manage directory and file locations in ways that minimize security risks and facilitate backup procedures
    - Allow developers to keep relative URLs in source code simple

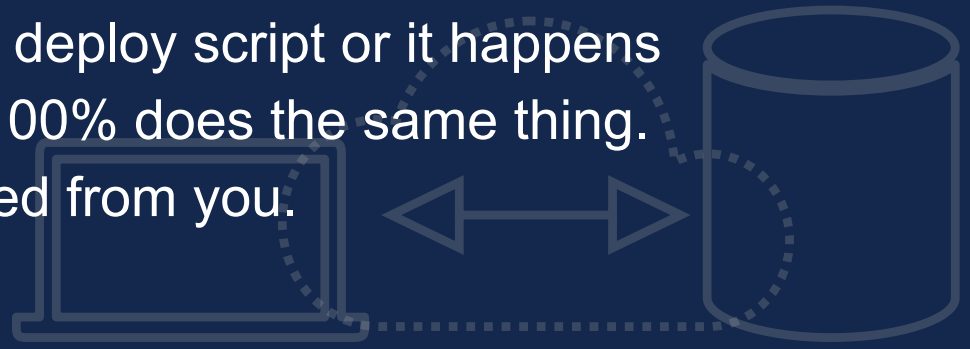


Virtual Hosting

- We know the hostname part of the URL is a virtual locator for files that live (physically) in a site's document root
- The idea of *virtual hosting* takes this a step further by allowing a single server to host many domains, each with its own document root
- Two methods of virtual hosting
  - Old way: multiple IP addresses per server
    - Not a great idea given the number of IP addresses available
  - Modern way: name-based using host headers
    - Allows for many HTTP servers mapped to a single IP

Developer Access

- Developers will need remote access allowing them to transfer files to and from the site's physical structure
- FTP (and other file transfer mechanisms with SSH or others) allow the administrator to restrict this access to areas of the site and block by IP
- These restrictions should be backed up by access control lists on the directories that enforce the "principle of least access"
- Note today even if you are using some deploy script or it happens automatically on push to a repo it still 100% does the same thing. Nothing has changed it is just abstracted from you.



Managing Users and Hosts

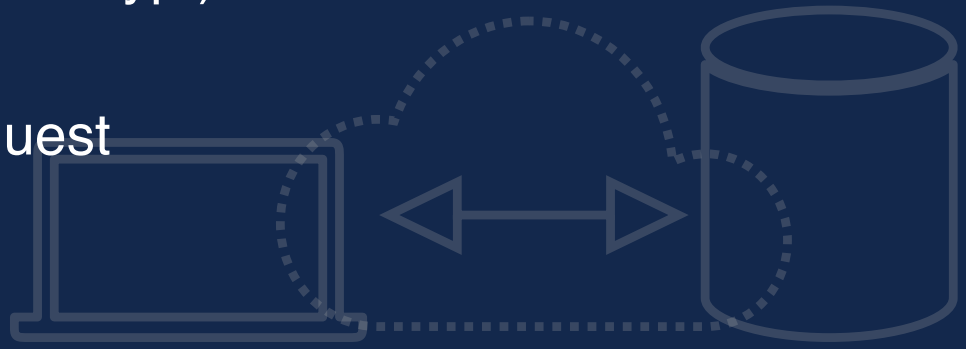
- Similar rules apply to managing access to the Web site itself by visitors
  - ACLs (access control lists) in the Web site's physical file structure should be set to the minimum required by the Web server to serve the resources on the site
    - This gets tricky with server side programming
  - If the Web site (or part of it) does not need to be available for anonymous access from everywhere then users, groups, hosts and IPs should be restricted
  - HTTP Authentication can also be employed to require to require login on a site or directory basis
    - NOTE: THIS IS A BASIC FORM OF AUTHENTICATION. IT IS USED FOR QUICK AND DIRTY TASKS AS OPPOSED TO FOR APPLICATIONS

Managing Users and Hosts

- Although authentication can offer security when poorly implemented trouble can ensue.
  - Challenges: Lack of end-to-end encryption of the entire message transmission makes hijacking, scanning and spoofing easy
- If all or part of the site requires authentication and serious security for user's login credentials, form based authentication over SSL is the only choice
  - 2019: You need to use SSL to not be penalized by Google.

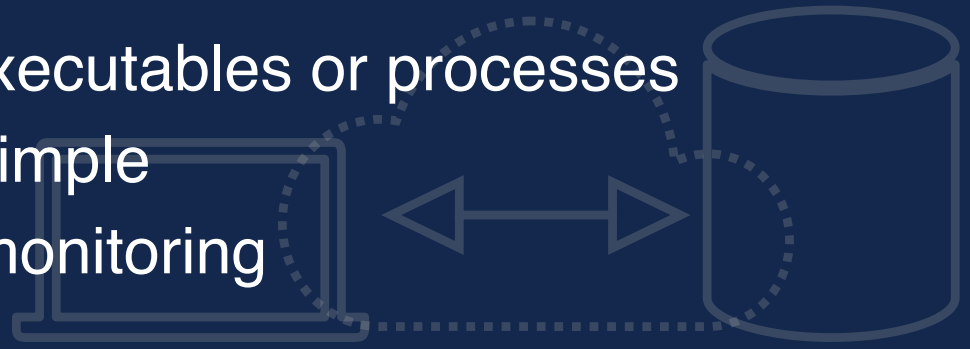
Basic SSL Configuration

- Initiate an application for a certificate from a recognized Certificate Authority (CA)
  - The site (domain) owner will have to prove they are who they say they are
- Create a Certificate Signing Request (CSR)
  - Contains the site's Public Key and matches up with a Private Key that is created simultaneously and stored on the server
- Submit the request to the CA and pay up
  - Today that isn't an issue (ex: Let's Encrypt)
- Retrieve the certificate and install it
- Test the certificate with an HTTPS request



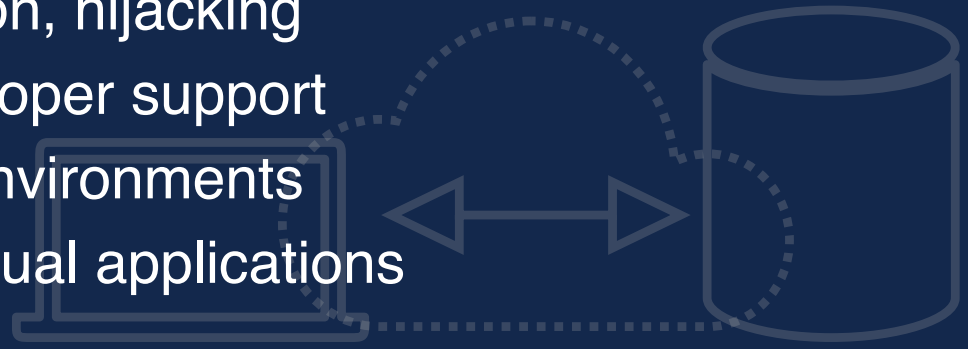
Supporting Web Applications

- Comparing static and dynamic sites
- Static site demands
  - Few performance demands on Web server
    - Serving files is light work
    - Caching is easy to do
    - State management probably not an issue
  - Few security risks
    - Tight permissions possible
    - No interaction with other executables or processes
  - Developer support relatively simple
    - Basic access control and monitoring



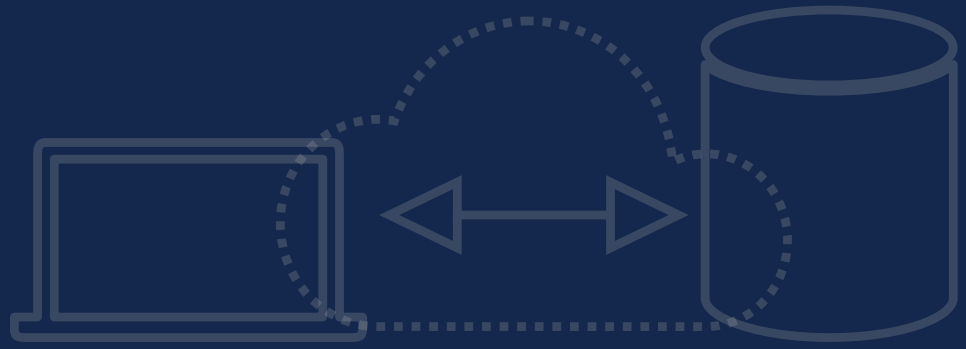
Supporting Web Applications, cont.

- Demands introduced by dynamic page generation on server side
  - Significantly heavier performance demands
    - Code execution
    - Database access
    - Caching more difficult to do
    - Complex state management schemes
  - Security risks go way up
    - Higher level permissions required
    - Buffer overflows, code injection, hijacking
  - Significantly more complex developer support
    - Install, maintain application environments
    - Potentially help debug the actual applications



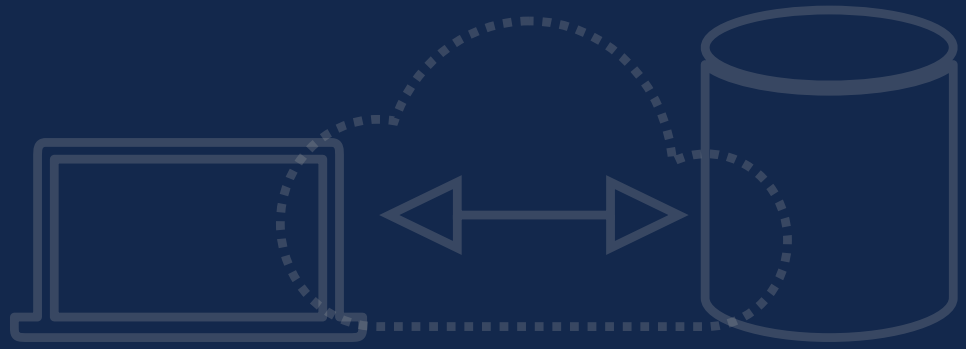
A Digression on Web Server Internals

- Server-side processing makes a simple model significantly more complex
- Basic internal request/response cycle
  - Read request
  - Do authentication if any
  - Process other headers
  - Map URL to physical path
  - Read file or retrieve cached response
  - Send response
  - Log
  - Cleanup



Web Server Internals, cont.

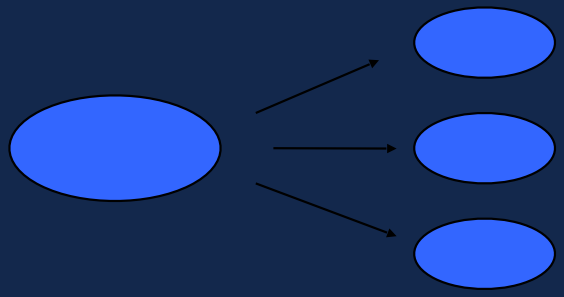
- Server programming adds a new dimension
  - Read request, set up internal data structures
  - Do authentication if any
  - Process other headers
  - Map URL to script or program
    - Script or program diverts request handling into new code paths
    - Server must wait for result of processing before it finds out what it is supposed to send back
  - Send response
  - Log
  - Cleanup



3 Server-Side Programming Models

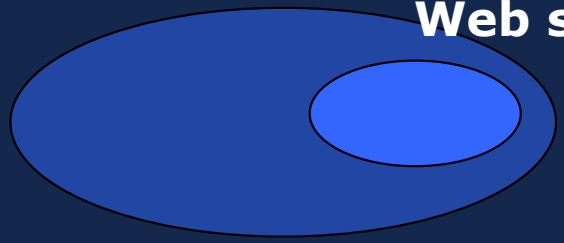
- What happens when the request gets diverted from the server's own internals?
  - Classic CGI model – "fork and exec"
    - Web server creates new child process, passing it request data as environment variables
    - CGI script issues response using standard I/O stream mechanisms
  - Server API model
    - Web server runs additional request handling code inside its own process space
  - Web application frameworks
    - Web server calls API application, which may manage request within its own pool of resources and using its native objects

3 Server-Side Programming Models

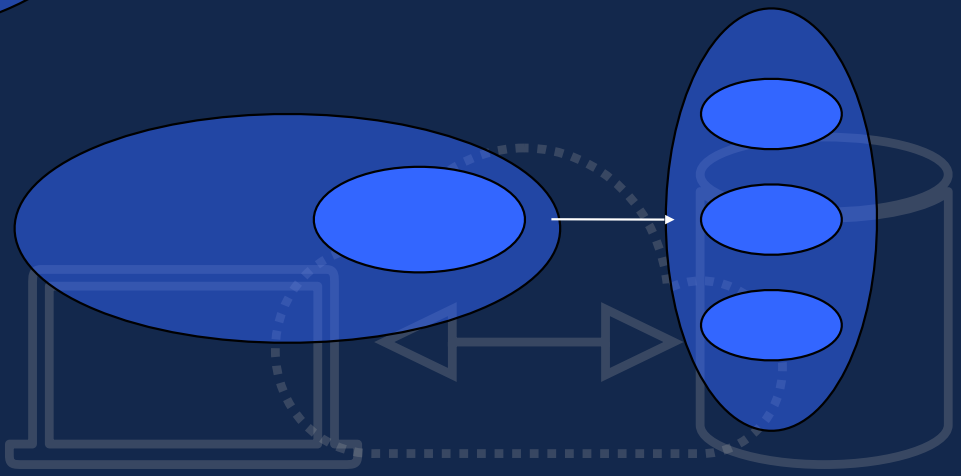


Classic CGI "fork and exec"

Server API running inside Web server's address space

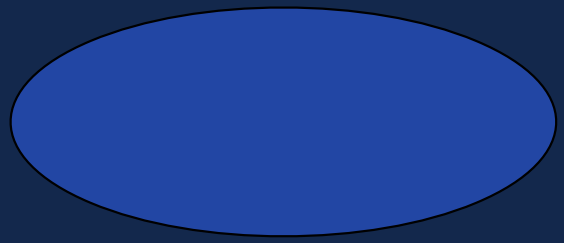


Web application framework running inside Web server process but managing its own pool of resources via IPC



Model 4?

The Web Server is the app / the app is the Web server?

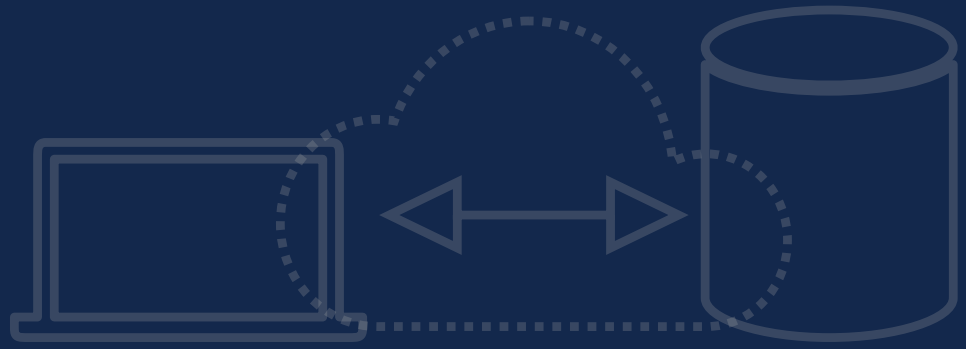


Common in the NodeJS world our code defines routes and the app is the server. This is both a good idea (limited surface area) and an insane idea (you have no idea what serving HTTP really means) Suggestion; Use Node behind a HTTP proxy or in a co-server model

Interestingly this is an old model that was abandoned and is now back

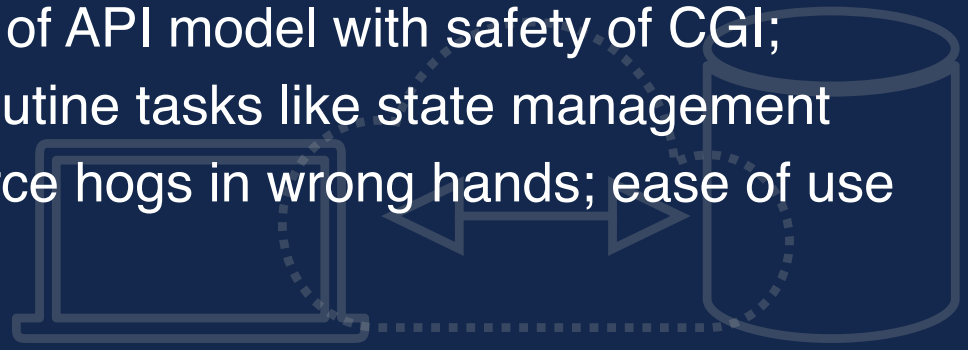
A new fun phrase!

Is your program being served or is it a server itself?



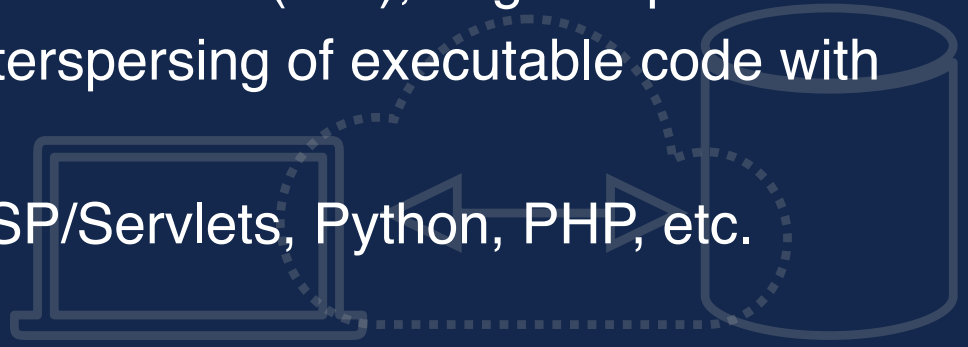
3 Server-Side Programming Models

- Each model has its pros and cons
  - Classic CGI model
    - Pro: isolation means easiest in principle to secure, least damaging if something goes wrong
    - Con: isolation makes it slow & resource intensive
  - Server API model
    - Pro: very fast & low overhead if written properly
    - Con: hard to write; blows up server if done wrong
  - Web application frameworks
    - Pro: ideally combines efficiency of API model with safety of CGI; adds helpful encapsulation of routine tasks like state management
    - Con: built-in tools can be resource hogs in wrong hands; ease of use may encourage carelessness



3 Server-Side Programming Models

- Many examples of each
  - Classic CGI
    - Scripts written in Perl (or whatever)
    - Programs written in C (or whatever)
  - Server API
    - Apache modules
    - ISAPI filters and extensions
  - Web application frameworks
    - All descended from Server Side Includes (SSI), original “parsed HTML” solution that allowed interspersing of executable code with markup
    - ASP, ASP.NET, Cold Fusion, JSP/Servlets, Python, PHP, etc.

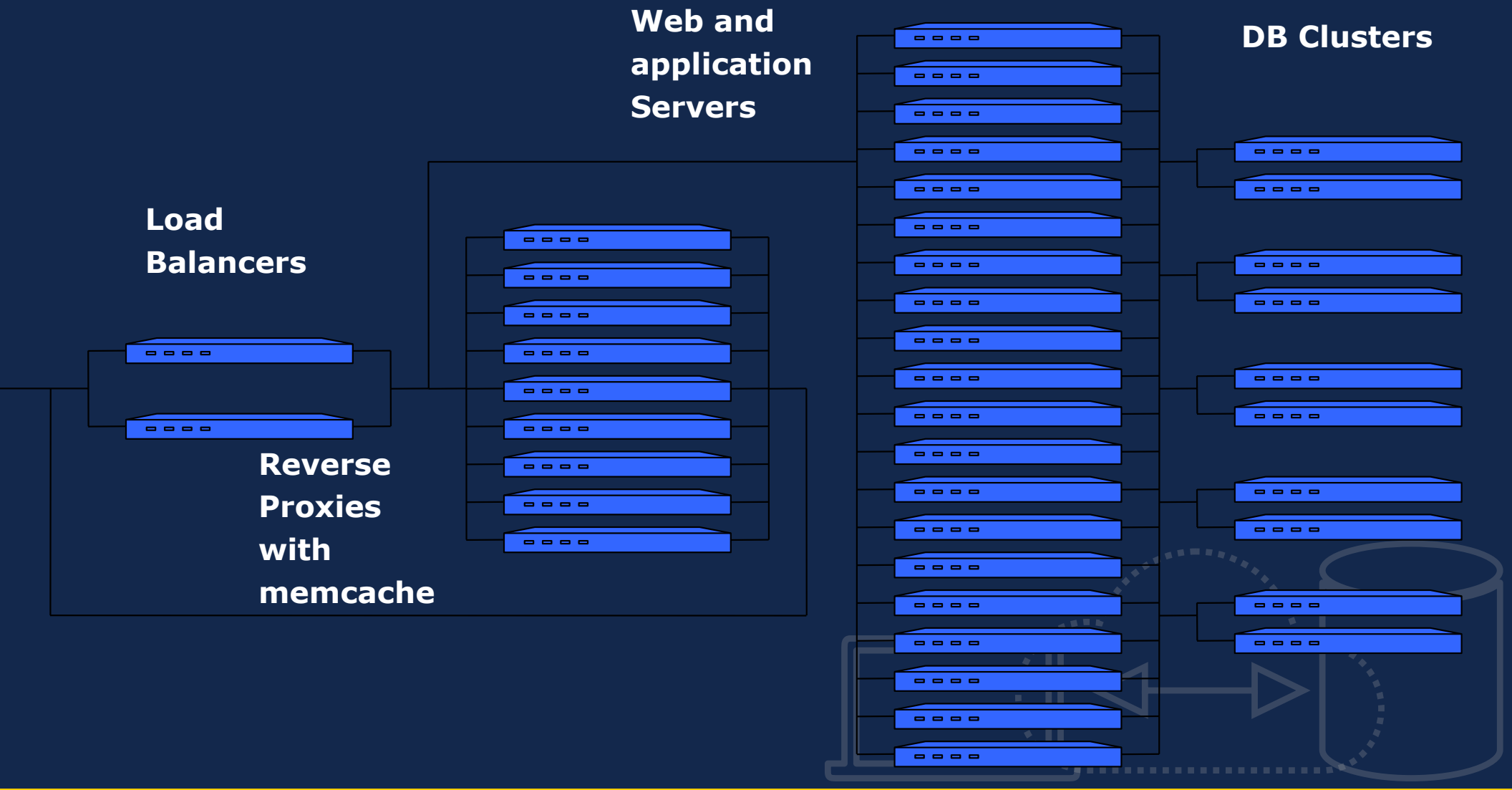


Server Sizing with Dynamic Content

- In high traffic scenarios with dynamic pages, when bandwidth is plentiful, disk access can be the major bottleneck
  - Especially problematic when backend databases are being accessed to build pages

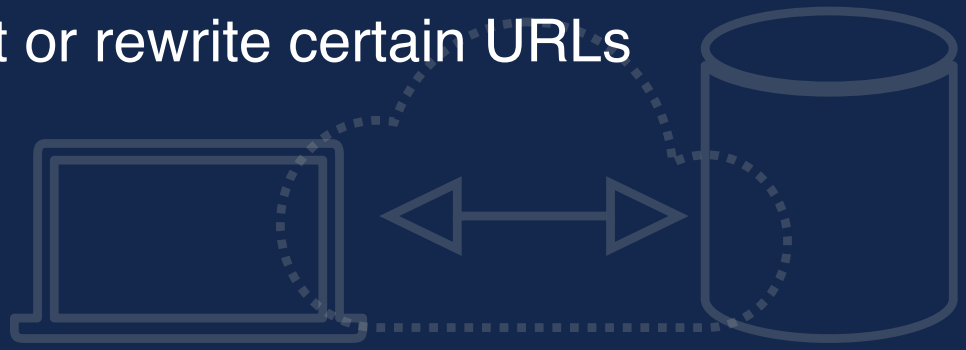
- There are many possible performance solutions
  - Use fast disk controllers
  - Exploit caching mechanisms
  - Keep as much data as possible in memory
  - Add hardware! (and give it specialized roles)

A complex server farm configuration



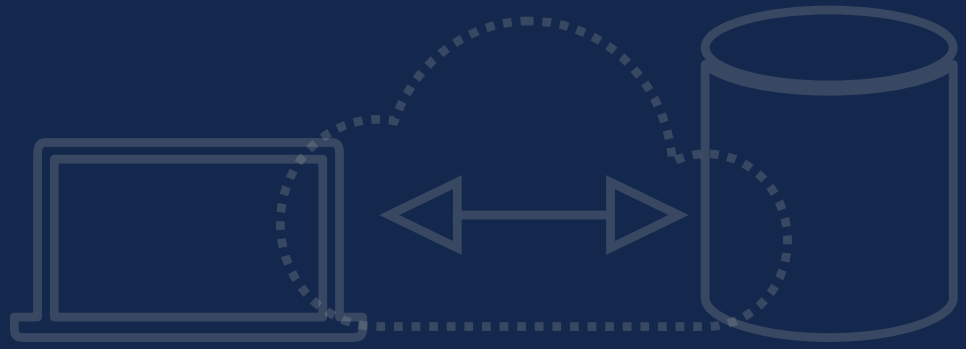
Web Applications and Site Structure

- With server-side programming it becomes even more important to treat the URL as virtual rather than physical
  - Each file called by an URL can generate many different responses
  - At the extreme, some methodologies call for a single file to generate all pages in the site
  - Many different physical resources, including database tables and additional files (includes) might be required to produce one response
  - Filters or modules might preempt or rewrite certain URLs altogether



Web Analytics – Overview of Log Files

- Log File Formats, Configuration, Management
- Why do Log Analysis?
  - Traffic Analysis (internal and external)
  - Quality of Service Analysis
  - Security audits
  - Performance analysis
- Statistics, Tracking, Reporting
  - Basic Concepts
  - Limitations and Caveats
  - Free and commercial tools
- Setting up a Robust Logging System

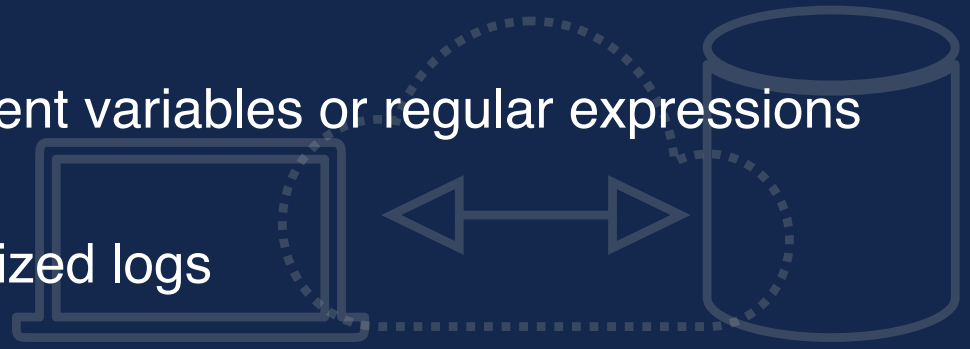


Log File Formats

- Apart from error logs, Web servers generate “access” or “transfer” logs that record per request activity
- Two formats
  - Common Logfile Format (CLF)
    - remotehost rfc1430 authuser [date] “request” status bytes
    - Combined Logfile Format adds referer and user-agent
  - Extended Logfile Format (ELF)
    - Two required directives (Version and Fields) at the top tell consumers of the log file how to parse it
      - #Version: 1.0
      - #Fields: date time c-ip sc-bytes time-taken cs-version

More on Extended Logfile Format

- date and time are standard fields
- Beyond those, the administrator is free to specify a wide range of extended fields
  - In IIS: `c-ip cs-username s-sitename s-computername s-ip s-port cs-method cs-uri-stem cs-uri-query sc-status sc-win32-status sc-bytes cs-bytes time-taken cs-version cs-host cs(User-Agent) cs(Cookie) cs(Referer)`
- Apache has particularly customizable formatting
  - Arbitrary ordering of fields
  - interspersing of text and formatting
  - Conditional logging using environment variables or regular expressions on the URL
  - Routing of certain entries to specialized logs



Managing Logs – Best Practices

- Log everything you need, but not what you do not need
- Rotate log files at intervals appropriate for your analysis and archiving requirements
- Write logs to a convenient, distinct, ample and secure location
- For heavy duty analysis on high traffic sites, consider using dedicated database server(s)
  - Records can be inserted directly or asynchronously
  - Analysis carried out without burdening site
  - Especially necessary for analysis of logging that covers extended time periods (i.e., longer than a single day)

Why do Log Analysis? (Traffic)

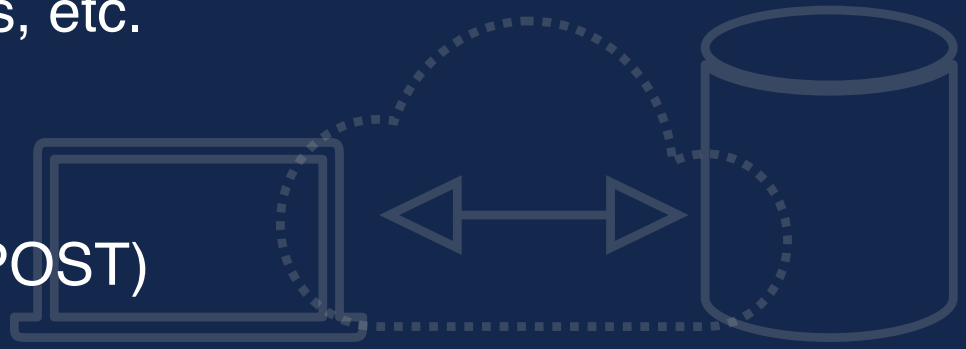
- Optimize content or ad pricing or positioning, assess popularity of site areas/ features
  - Most popular pages
  - Top entry point pages
- Billing in hosting environment or resource allocation in enterprise environment
  - Most active domains
- Search engine activity
  - Indexing and query frequency
- Campaign tracking
  - Top referring sites/domains/URLs
  - Time/event based spikes or dips
- Audience analysis
  - IP geography, language preference, client host type (.com, .edu, .org, etc.)

Why do Log Analysis?

- Optimize first views, adjust site structure
  - Top entry point pages
- Adjust for browser capabilities
  - User agents
- Identify points of failure
  - Error codes and counts (404, 500)
- Identify navigation patterns and frequent exit points
  - IP, referrer and cookie tracking
  - Not easy to do, but maybe worth the effort for finding out if users are aborting an application path early

Why do Log Analysis?

- Identify “leaching” or “scraping” activity
  - Most requested files
  - IP, referrer and cookie tracking
  - Entry point pages
  - Bandwidth utilization
- Track sources and methods of reconnaissance attempts, exploits and attacks
  - Error codes
  - Attempted access of shells, scripts, etc.
  - Attack and worm signatures
  - Long/malformed request URLs
  - Unusually large request entities (POST)

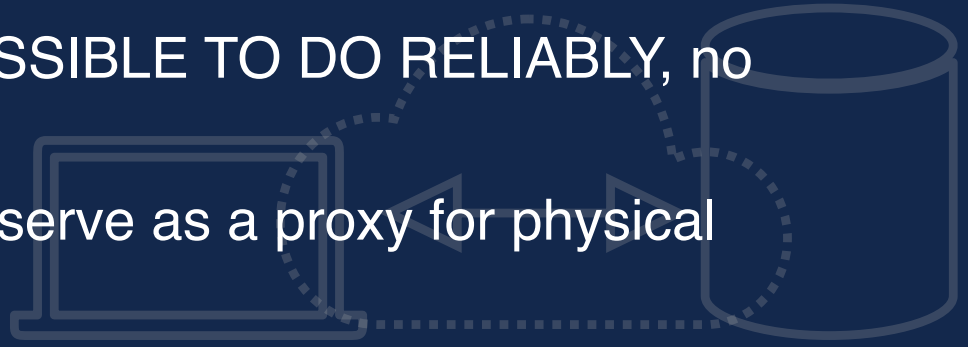


Why do Log Analysis?

- Verify or update Web server sizing estimates by using actual data
- Issue or verify bandwidth bills
  - Bytes sent (within given time frame)
  - Request frequencies, especially peaks and valleys over given periods of time
- Assess caching efficiency
  - Harder to do but possible by looking at (dependent) requests per page and 304 response codes

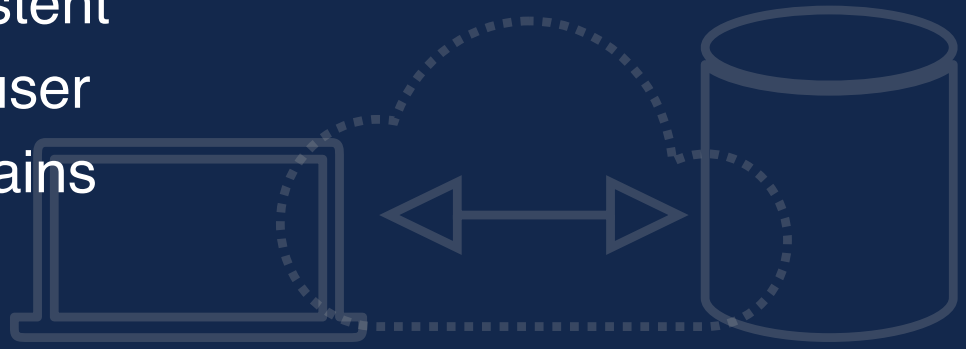
Statistics, Tracking, Reporting

- Basic concepts
  - Counting hits versus counting page views
  - Distinguishing page views from hits
    - File name
    - File type
    - Web server response code (to exclude errors)
    - Client host (if excluding internals)
  - Counting unique visitors
    - Sets of page views attributable to one user
    - MUCH harder to do and IMPOSSIBLE TO DO RELIABLY, no matter what anyone tells you
    - Requires a unique identifier to serve as a proxy for physical presence of the virtual visitor



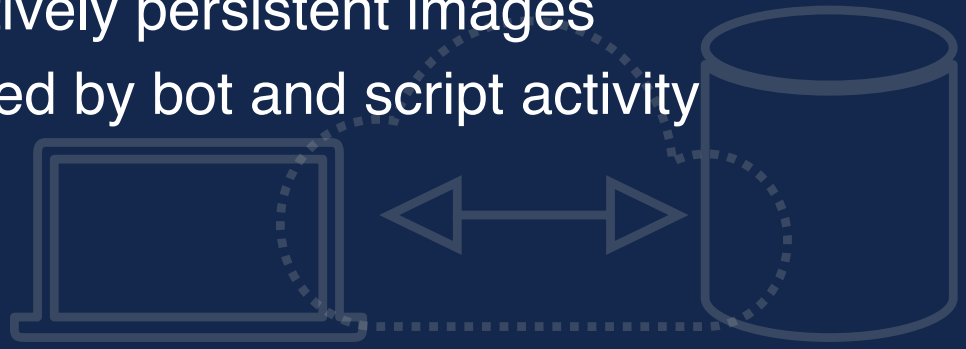
Statistics, Tracking, Reporting, cont.

- Counting Unique Visitors, continued
  - Client IP is easiest identifier to use, but also least reliable
    - Dynamic IPs, proxies with NAT
  - Login is highly reliable (except for sharing) but limited in applicability to sites/sections where it won't discourage users
  - Cookies (transparently placed) are the best all-purpose compromise, but still have limits
    - Must have backup if disabled on client
    - Still not guaranteed to be persistent
    - Bound to machine rather than user
    - Can not be shared across domains



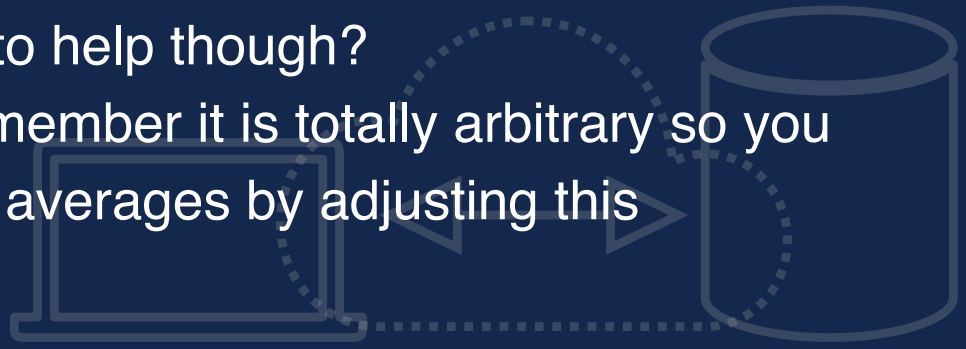
Statistics, Tracking, Reporting, cont.

- Be aware of limitations and caveats when counting requests and page views
  - Browser and proxy caching stop requests from ever reaching the server and its logs, deflating actual page views by actual users
    - Can be partially mitigated by use of HTTP cache control headers, but this is neither guaranteed to work nor cost-free in bandwidth terms
    - A good compromise is to flag pages for non caching but take advantage of caching for relatively persistent images
  - Request counts will also be inflated by bot and script activity (desirable or undesirable)



Statistics, Tracking, Reporting, cont.

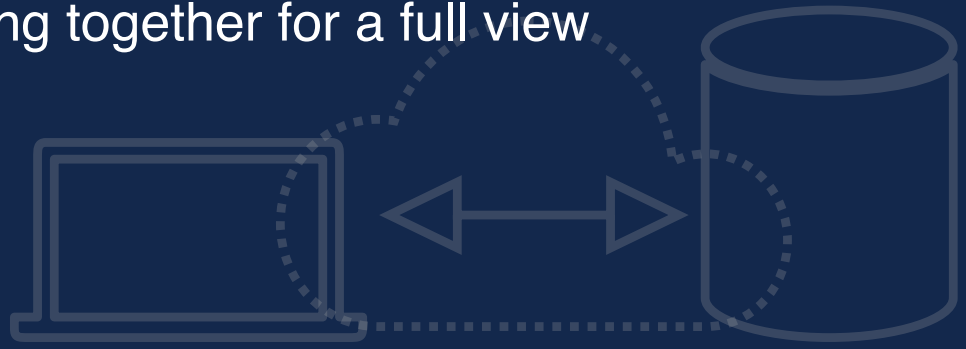
- Tracking the elusive “Visit”
  - How long a unique visitor spends on the site before exiting
  - The concept has *tremendous* potential utility for marketing and quality of service analysis
  - Stateless nature of HTTP makes it next to impossible to determine with any degree of accuracy 100% accurate visit time
  - What is commonly done is to use rule of thumb such as “a series of page requests by a visitor without 30 consecutive minutes of inactivity” – common assumption
    - Maybe JavaScript can be used to help though?
    - Otherwise for session length remember it is totally arbitrary so you can make longer or shorter visit averages by adjusting this assumption



Analytics Beyond Logs

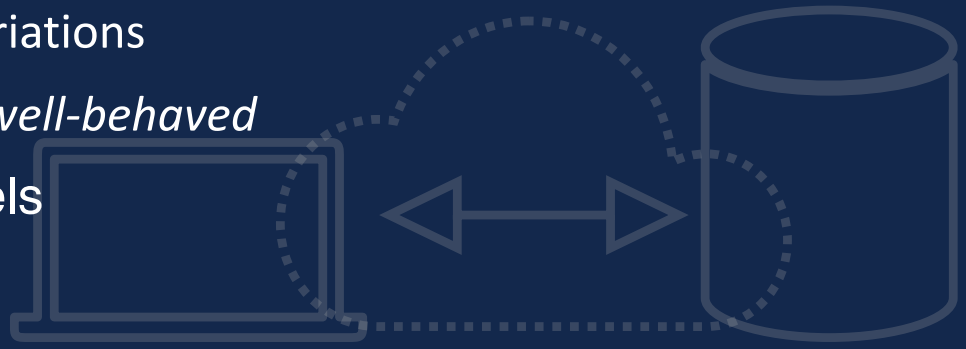
- JavaScript Analytics (ex. Google Analytics) is far more popular than log based and we'll cover it later.
  - Pros – Degree of detail, access to log problems, ease of Web types adding tracking, consolidation of tracking data
  - Cons – Requires JavaScript (execution limits particularly bots, failures, load time concerns)

- Network Tap based systems also exist which provide insight into delivery
- Given the three sides of the Web equation one wonders if this isn't once again a question of not versus but working together for a full view



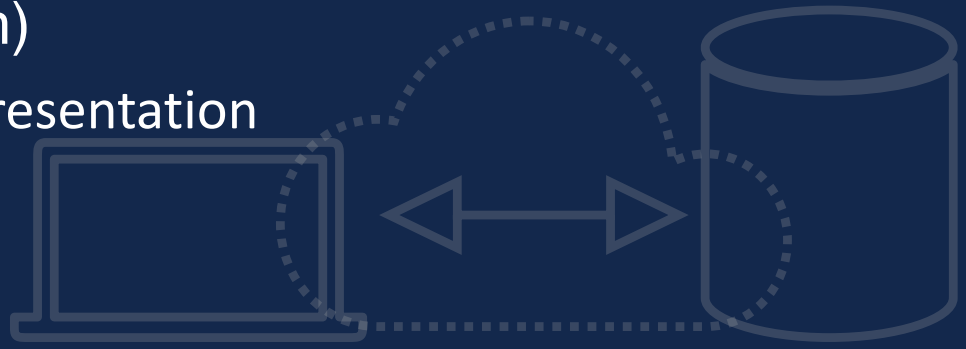
Dealing with Bots and Spiders

- Automated User Agents
  - Bots, Robots, Crawlers, Spiders, etc.
  - Most capable of automated site traversal
  - Bots come in both benign and malign forms
    - Search engine indexers, link checkers, monitors
    - Spam bots, leechers & scrapers, attack bots
  - Benign bots usually (not always!) announce themselves with unique User Agent headers
    - Frequently updated lists of common search agent bots is available online
    - "googlebot" and other well-known variations
  - Benign bots are usually (not always!) *well-behaved*
    - Crawl at rates well below DoS levels
    - Obey Robot exclusion directives



Special Handling for Search Agents

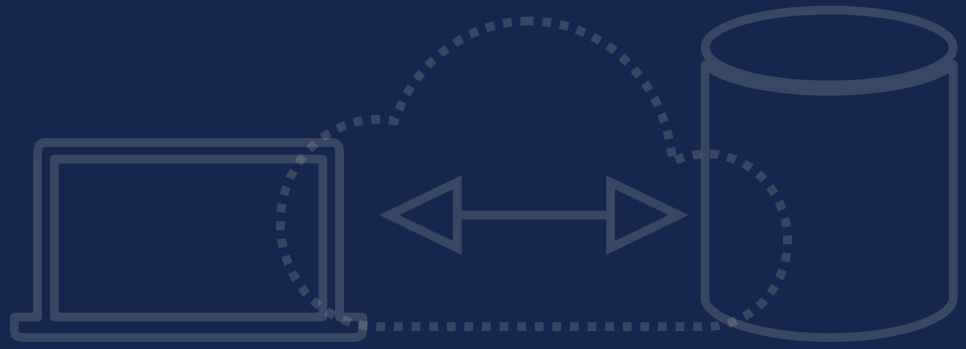
- What to do about indexing bots and dynamic pages?
  - May need to exclude them to prevent indexing of content that will vary per user or request
  - May need to provide spider-friendly versions of dynamic pages to expose content to desired indexing
  - Alternate, search-optimized pages *can* be helpful but proceed with caution! Black Hat approach
    - Bots can impersonate user agents to prevent/punish spamming (bait pages, stealth)
    - Content should not vary, only presentation



Using the Robot Exclusion Protocol

- Place a robots.txt file in the site's document root
  - Careful: don't show your soft spots? Or maybe you want to?
- Well-behaved bots will request this first, and obey its directives
  - #sample robots.txt file

```
User-Agent: *
Disallow: /newtoday
Disallow: /downloads
User-Agent: newsbot
Disallow: /pressreleases
```



Beyond the Robot Exclusion Protocol

- For controlling unfriendly bots, robot exclusion is insufficient
- Access control is hard to do, since neither IP ranges nor User Agents are reliable identifiers of unfriendly bots
- Access control based on traversal pattern and rate is possible
  - Using IP and request path against time elapsed it should be possible to identify a traversal and dynamically block it
  - Nontrivial to program and subject to countermeasures if it catches on
  - Passive bot detection is certainly an open area for research – notice all the CAPTCHAS as well

Server and Site Monitoring

- Monitoring Site Availability
  - Content monitors request portions of key pages and compare actual to expected results to verify that site is alive and working properly
  - Application monitors submit form data and analyze result to verify backend systems are up
- Monitoring Server Uptime
  - Service monitors warn when services go down or become unreachable
  - Automated restart can be attempted
- All monitors usually alert via email, pager, SMS
- Thresholds can be set to allow for transient errors & delays, or warn of degrading performance

Server and Site Monitoring, cont.

- More active monitoring is also possible
- Can be useful especially in testing and diagnostic situations
  - Process monitors allow for isolation of specific processes to pinpoint trouble spots, especially resource bottlenecks and leaks
  - Performance monitors, especially in conjunction with stress tools that simulate traffic, help in accurate dimensioning
  - Network monitors allow examination of packet level data and protocol details for uncovering connection related problems
- We are currently seeing finally a rise in DevOps to mind the gaps between apps, server, and network

Summary

- Web Servers are pretty similar in terms of duties - they service HTTP requests and perform authentication, authorization, retrieval of static files, execution of code (usually via an app framework or module) and logging
- While interfaces may differ the rules are the same
- Scaling and managing servers is a large topic, but you should use data to understand load to avoid premature (or to late) scaling
- Servers present a possibility for analytical tracking with log files