Server-Side Scripting Environments

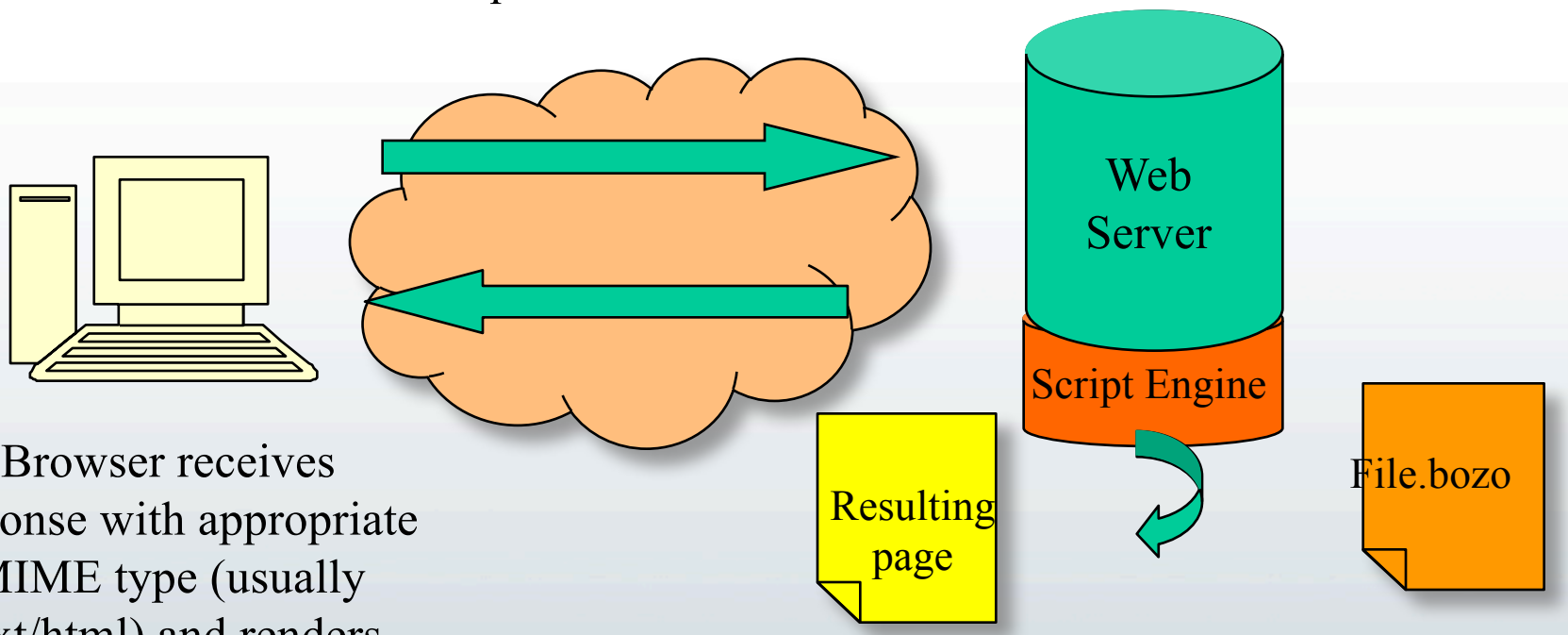
Server Side Scripting Intro

- Server-side scripting offers a balance between complexity and performance
  - Not as hard as server-modules (or maybe CGI), but not as performance oriented as server-modules
- The general idea is to add scripting directives with template files (often thought of as modified HTML files)
  - The scripted pages are intercepted by a Web server on the way out and the code converted into the appropriate output – generally HTML

Implementing a Server-Side Scripting Language

- Consider the idea of writing a special BozoScript server module that looks to see if a file is being requested with an extension .bozo
  - The server module either intercepts the page, parses and produces or a result or passes it on without parsing
  - The trigger for interception is generally a file extension, though some may foolishly intercept anything by setting interception on .html
    - Good because it hides implementation, bad because you may parse something you don't need to

How Server-Side Scripting Works Contd.



Server-Side Scripting Language Characteristics

- When looking at server-side scripting languages you see some common characteristics
  - Syntax variations
    - Tag like or script like
  - Focus on troubles of Web programming
    - Session management, form handling, database programming
  - Occasionally understanding of separation of programmer, designer, and markup specialists
    - Through syntax, separation of presentation and logic, etc.
  - Reliance on external objects for “heavy lifting”

Server-Side Scripting Language Characteristics

- Generations of Server-side scripting?
  - Zero Generation: SSI (server-side includes)
  - 1st Generation: Classic ASP, ColdFusion, PHP
  - 2nd Generation: JSP, ASP.NET
  - 3rd Generation: Declarative style- Flex –OR- 4GL like - Ruby on Rails
- Yet for all their supposed differences server-side scripted environments suffer from an exchange of ease for scalability and performance, some also suffer from portability being tied to a particular platform.
- The main thrust of 2nd generation efforts being performance and/or support for larger system development.
- Oddly the thrust of 3rd generation is a pendulum swing back to the ease of development offered by 1st generation server-side scripting which made it so popular.
- And in all this maybe we are right back JavaScript which was 1st generation - the immovable rock of the Internet. (see Node.js)

Parsed Script Intro

- We review a few of the basic issues with the first generation server-side scripting languages here before studying PHP more in-depth.

- Server Side Includes (SSI)
  - .shtml files
  - In the form of a comment
  - <!--#include file="footer.html"-->
  - SSI is common to nearly every Web server, though not always enabled. Some Web servers may have unique features
- Already even with SSI you see the main issues
  - How do you indicate the file has some script in it (file extension)
  - Within the file how do you delimit or separate template from script (in this case HTML comment)
  - Performance hit vs. the productivity gain
- Some editors like Dreamweaver provide this same type of template style without the use of server-side page composition

ColdFusion Markup Overview

- ColdFusion initially invented by Allaire, merged with Macromedia now Adobe
  - Simple server-parsed tag based scripting language that is very focused on connecting databases to Web pages
  - ColdFusion requires a special Application Server and it runs on Windows, Linux, and Solaris
  - Like other server parsed scripting solutions the app server is engaged when a file of a particular extension (.cfm) is requested
- Cold Fusion markup looks like basic HTML tags and thus is popular with people who are new to programming
  - Some black-eyes from syntax and security concerns
- Main purpose of CF is to connect to a database
  - Either using ODBC or native database drivers

ColdFusion Markup Overview

- You need to specify basic SQL (Structured Query Language) statements to use CF
- Given a DB table called *Positions* with the fields *Position-Num*, *JobTitle*, *Location*, *Description*, *Hiring Manager*, and *PostDate*. It is easy to query the table using statements like
  - *SELECT * FROM Positions*
  - *SELECT * FROM Positions WHERE Location="Austin"*
  - *SELECT *
    FROM Positions
    WHERE ((Location="Austin" OR
    (Location="Los Angeles") AND
    (Position="Game Tester"))*

<CFQUERY>

```
<CFQUERY NAME="ListJobs"
        DATASOURCE="CompanyDataBase">
            SELECT * FROM Positions
 </CFQUERY>
```

*Note: The **DATASOURCE** attribute is set equal to CompanyDataBase, which is the ODBC data source that contains a database called Company, which contains the Positions table from which data is pulled.*

<CFQUERY> supports attributes like NAME, DATASOURCE, MAXROWS, USERNAME, PASSWORD, TIMEOUT, and DEBUG

<CFOUTPUT>

- Once you have data for example from variables, calculations or a from a database query you can output it with <CFOUTPUT>

```
<CFOUTPUT QUERY="ListJobs">
        <hr noshade><br>
        Position Number: <b>#PositionNum#</b><br><br>
        Title: #JobTitle#<br><br>
        Location: #Location#<br><br>
        Description: #Description#
</CFOUTPUT>
```

- Notice the use of # symbols to relate the fields of the database with output “slots” Also notice how HTML and database fields are intermixed.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<CFQUERY NAME="ListJobs" DATASOURCE="CompanyDataBase">
SELECT * from Positions
</CFQUERY>
<html>
<head><title>Demo Company Job Listings</title></head>
<body bgcolor="#FFFFFF">
<h2 align="center">Job Listings</h2>
<hr>
<CFOUTPUT QUERY="ListJobs">
<hr noshade><br>
Position Number: #PositionNum#<br><br>
Title: #JobTitle#<br><br>
Location: #Location#<br><br>
Description: #Description#
</CFOUTPUT>
<hr>
<address>
Demo Company, Inc.
</address>
</body>
</html>
```

Common CFML Elements

- <CFABORT>
- <CFAPPLICATION>
- <CFCOL>
- <CFCONTENT>
- <CFCOOKIE>
- <CFERROR>
- <CFFILE>
- <CFHEADER>
- <CFIF>
- <CFINCLUDE>
- <CFINSERT>
- <CFLOCATION>
- <CFLOOP>
- <CFMAIL>
- <CFOUTPUT>
- <CFPARAM>
- <CFQUERY>
- <CFREPORT>
- <CFSET>
- <CFTABLE>
- <CFUPDATE>

CFM Summary

- Very focused on building DB driven sites
- Very high level (like a 4GL)
- Tag like so integrates well with HTML
- Some programmers find clumsy because not script like. However, later versions of ColdFusion addressed this, yet never lost that “baby coding” feeling for some people
- Still Windows oriented despite appearing cross server
  - Attempt to revive platform in Flash/RIA/declarative guise
- Interesting recent rise of tag based systems like XUL, XAML and others suggest CF might have been onto something all along

ASP Intro

- Microsoft's Active Server Pages (ASP) was (and still is to some degree) a very popular server parsed scripting framework
  - Bundled with IIS directly
  - Indicated by files ending in .asp
  - Does not specify language used is a framework
    - Not understood and often confused with VBScript or Jscript (MS JavaScript clone)
  - More code like but slightly less friendly than PHP and ColdFusion
  - With the rise of ASP.NET many class ASP folks unable to make the complexity leap moved to PHP or even Ruby

ASP Example

Note <script> tag showing location of execution and language and script is delimited by <% %>

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
    <script language="VBScript" runat="Server"></script>
    <html>
    <head>
    <title>ASP Example</title>
    </head>
    <body>
    <h1>Breaking News</h1>

    <% = date() %>

    <p>Today the stock of a major software company <br>
    reached an all time high, making the Demo Company CEO<br>
    the world's first and only trillionaire.</p>
    </body>
    </html>
```

ASP Summary

- ASP syntax can get more familiar to programmers, for example consider a similar database example
  - ```
    <%
        Set Conn = Server.CreateObject("ADODB.Connection")
       Conn.Open ODBCPositions
       SQL = "SELECT JobTitle, Location, Description, HiringManager,
    PostDate FROM Positions"
        Set RS = Conn.Execute(SQL)
        Do While Not RS.EOF
    %>
    ```

- Yet is it correct to want to get more complicated with script? Instead should we do simple tasks with scripts and do heavy lifting with objects?

- Script + objects makes sense and in this respect you would find that CFM and ASP are more similar than they are dissimilar. Syntax issues more than anything.
    ```

Classic ASP is Back to the Future?

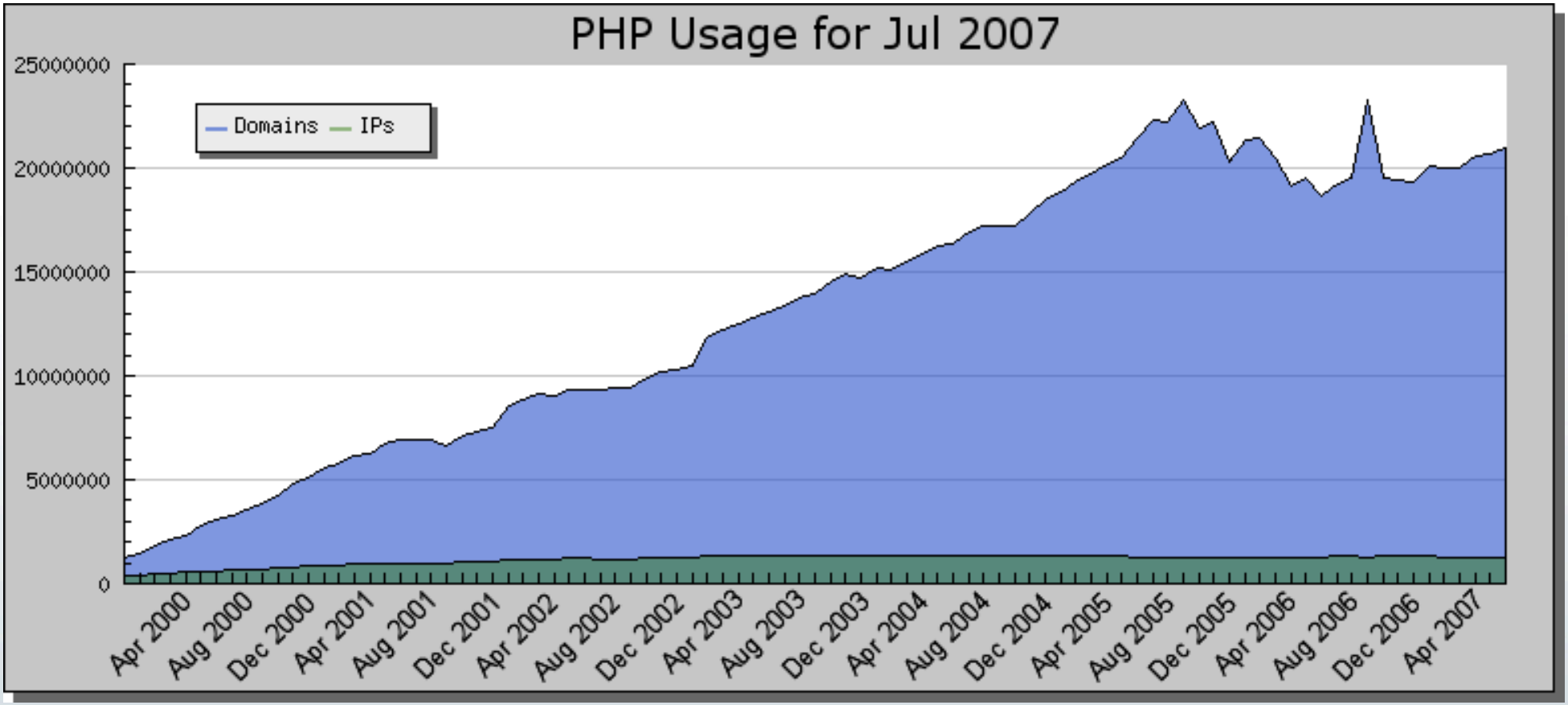
- Today server-side scripting is taking a hard right turn towards JavaScript
- Obvious reasons
  - All things JavaScript
  - Better to use one language than 2 or 3 or ...
  - Callback / Event Style Architecture makes sense for some types of network apps
- Interestingly classic ASP was often written in JScript (it was language neutral and usually VBScript or JScript), it looks a lot like some "new" stuff

Introduction to PHP

PHP Intro

- PHP (www.php.net) is a popular open source server-side scripting language that works with a wide range of Web servers and operating systems.
  - The language is relatively easy to learn but includes numerous functions and related packages to
  - The language is probably not suited for extremely large system development though like other server-side scripting environments (classic ASP, ColdFusion, etc.) it is often used in this manner
  - According to Netcraft PHP may be the most popular server-side technology on the Web
    - File extension survey – be careful does this oversample people using a message board implemented in PHP vs those who raw code?

PHP Usage Growth



Source: Netcraft (not finding newer surveys on this ☹)

Why PHP?

- It's easy
  - Syntax is familiar and forgiving
- It's flexible
  - Doesn't force a particular coding style
- It's powerful
  - Lots of functions, lots of free code, extensible and works with lots of other technologies (ex. DBs, payment gateways, XML,etc.)
- It's free
  - Runs on just about anything* *prefers Linux
- Lots of people use it
  - Good job opportunity? Lots of assistance

Why not PHP?

- Open source stigma
  - Who do I call if it is broken? Who is to blame (sue)?*
  - Who would pay for programmers in it
- It's too easy?
  - Type of people who use it often don't follow standard programming practices leading to "spaghetti code"
- Lack of quality and standardization
  - Too many frameworks, too many versions, platform quirks, etc.
- Not good enough for the enterprise?
  - Strong religious beliefs supporting "enterprise Web platforms" like J2EE and .NET
  - Lack of some bridges to really large systems like ERP platforms
  - You might want to investigate the "share nothing" architecture idea of PHP and findings from scalability conferences before you buy this
- No enforced programming model and environment
  - Many contributed one but central authority for components and framework was traditionally weaker than .NET and J2EE - changing

Example PHP Applications

- Basic form processing
  - Email form contents, save sign-ups to a DB, etc.
- User/browser aware pages
  - Change look or technology based upon user/browser
- Collaboration
  - Message boards, polls, blogs, etc.
- Content Management Systems (CMS)
  - Maintain and update site content via browser interface
- E-commerce/Shopping carts
- Portals
- Just about anything else you might dream up – games, etc. - people it's just a programming language!

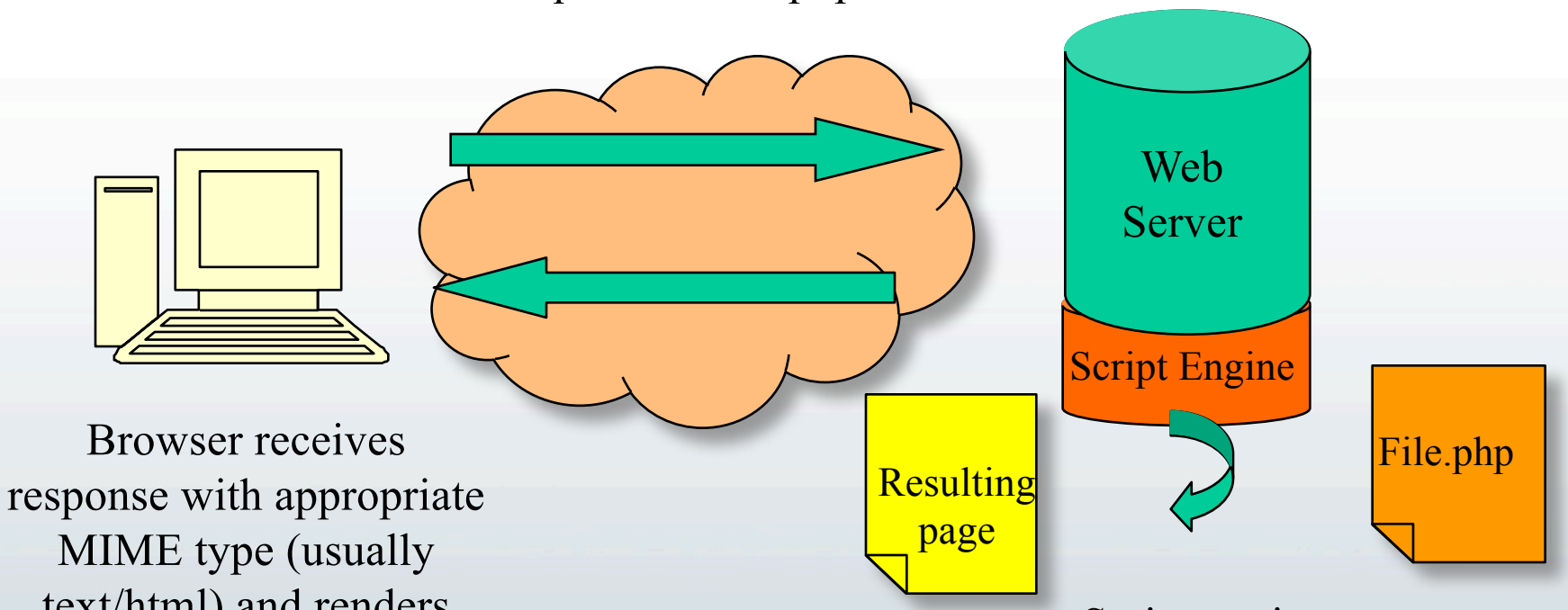
Brief History

- Starts life as "Personal Home Page" tool)
  - initially announced as 1.0 – June 1995
- Mid-1996 (PHP/FI) – starting to be more of a fleshed out script embedded in HTML technology
- June 1998 – PHP 3.0
  - This version starts the real PHP adoption effort
  - This and the economic consideration of hosting vendors coupled with the simplicity of the technology
- May 2000 – PHP 4.0
- July 2004 - PHP 5

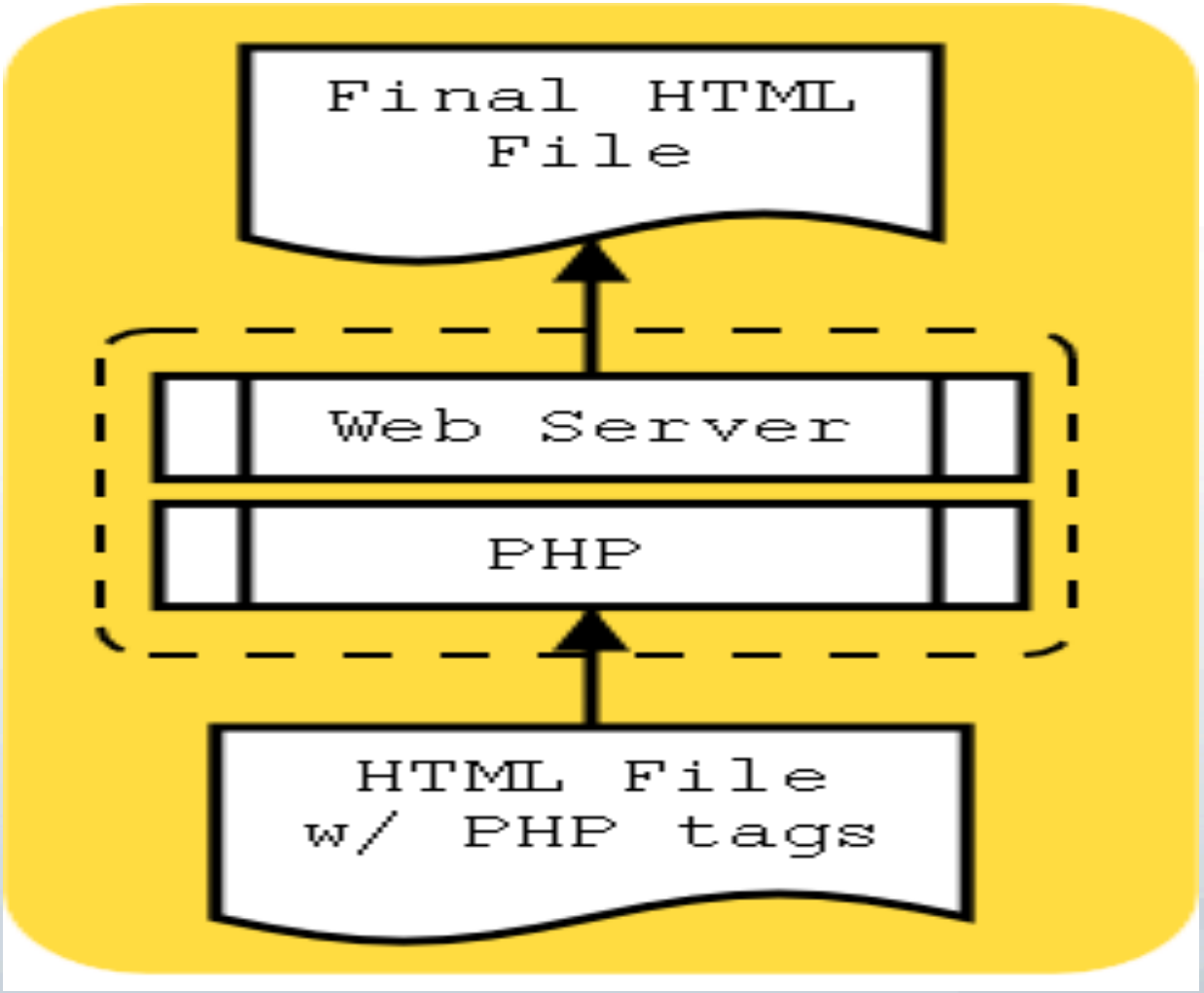
How PHP Works

- PHP files are generally an intermixture of PHP code and (X)HTML/CSS/Javascript
  - If the PHP is contained in the (X)HTML or the (X)HTML in the PHP
  - May not be clear but is not really that important
- PHP files are intercepted and evaluated by a scripting engine (generally implemented as a server module/ ISAPI) when requested producing the appropriate page
  - The file extension .php is used to trigger the script engine though the extension is configurable

How PHP Works Contd. (Same picture!)



Same Idea Simplified



PHP Helloworld

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-
    8859-1" />
<title>Hello World PHP Style</title>
</head>
<body>

<?php
  print "Hello World from PHP!";
?>

</body>
</html>
```

PHP Embedding Methods

- Typically PHP included in XML style
  - `<?php ?>`
  - Short style `<? ?>` is possible though not recommended and may require php.ini file change
  - If you want to include PHP in XHTML or XML files this is the best approach though <script> inclusion discussed next may also validate if appropriate CDATA directives are used
- Alternatively use the <script> tag
  - `<script language="php"> </script>`
  - ```
    <script language="php">
    //<![CDATA[
       PHP code here
    //]]>
    </script>
    ```
    ```

PHP Embedding Methods

- ASP style
  - <% %>
  - Goal to provide easy upgrade path for ASP developers
  - To use this inclusion you will likely need to modify your server's php.ini file
- Direct echo
  - Within HTML you often find PHP variables and such being output directly using <?= ?> For example
    <input type="text" name="magic" value="<?= $foo; ?>" />

PHP External File Inclusion

- A valuable use of PHP is to include common site elements from a standard location such as headers, footers, navigation, and so on.
- There are numerous ways to include things in PHP including `readfile()`, `include()`, and `require()`.
  - It is also possible to use lower level file functions to perform inclusions.
- The `readfile()` function takes a specified file and writes its contents to output.
  - `readfile('footer.html')`

PHP File Inclusion methods

- You can also include content from another file using include

```
<?php include 'navbar.html';  ?>
```

  - If included files are not found the page will continue to process but a warning will be issued
  - Note the file contents is not just included it is evaluated
  - Included file contents must be wrapped in <?php ?> since parsing will return to HTML mode upon inclusion

- The `require` statement is identical to `include`, but if the file is not found the page is aborted

```
<?php require 'global.inc' ?>
```

Determining PHP Settings

- PHP has a variety of settings for inclusion methods, loaded functions, etc.
  - Use function call `phpinfo();` to have the engine tell you its configuration
  - Usually you will want to locate the php.ini to make some changes (if you have rights to do so)
    - You will not on most lower end hosted sites or ones that you may not admin but rely on an IT group to help you
  - A simple call to phpversion() can give you simple version info
    - Example: `echo 'Current PHP version: ' . phpversion();`

Determining PHP Settings Contd.

- Part of the power of PHP comes from all the various modules that can be added in

- A simple way to see what is added is to make a call to `get_loaded_extensions()` from your script
  - Example: `print_r(get_loaded_extensions());`

- You can further look at the functions within a module named *modname* with `get_extension_funcs(`*modname*`);`
  - Example: `print_r(get_extension_funcs('standard'));`

PHP Language Overview – Case Sensitivity

- PHP is somewhat case-sensitive
  - Built-in constructs and keywords like while, class, if, and so on are not case-sensitive
  - User defined classes and functions are not case-sensitive
  - Variables are case sensitive

PHP Language Overview – Statements and Blocks

- PHP statements are terminated by a semi-colon.
- Blocks are indicated by curly braces { }

Note: The final statement before a closing script delimiter does not require a semi-colon.

PHP Language Overview – White Space

- PHP is generally white space insensitive, though when intersecting with other languages (e.g. HTML, JavaScript, etc.) via output you should be cautious.

PHP Language Overview – Intermixing HTML & PHP

- You'll notice that people often use PHP to printout small snippets of XHTML or other client side tech.
  - <?php print "<h1>Hey there!</h1>"; ?>
- Alternatively they might do this instead

```
<h1>
  <?php print "hey there!" ; ?>
</h1>
```

- You can see the issue of the PHP in the HTML or the HTML in the PHP
- If you go the later route you might find a PERL style of output very useful to do larger blocks

PHP Language Overview – Comments

- PHP supports 3 forms of comments
  - UNIX Shell style:
    - `# I am a single line comment`
  - Standard C style:
    - `/* I am a multi-line comment */`
  - C++ style:
    - `// I am a single line comment`
- Note: Comments are stripped from output (good thing) by the interpreter.
- Question: Are comments and formatting a good thing in your PHP source?

PHP Language Overview

- Variable names in PHP always begin with a `$` and are case sensitive so `$Name` is not equivalent to `$NAME` or even `$NaMe`.
- Legal identifiers in PHP start with a letter or underscore and may be followed by these characters as well as digits

  - Good Variable Names
    - $name
    - $_browser
    - $ThreeStooges
  - Bad Variable Names
    - $3Stooges
    - $$fun var
    - $|foo

PHP Language Overview

- PHP has eight data types:
  - 4 scalars
    - Integers
    - Floating-points
    - Strings
    - Booleans
  - 2 collections
    - Arrays
    - Object
  - 2 specials
    - Resource
    - Null

PHP Language Overview

- Integers
  - Decimal, octal, and hex all allowed
    - Good: 754, -3, 0755 (octal), 0xFF (hex)
  - Range typically -2,147,483,648 to 2,147,483,648
    - Usually similar to the long type in C
    - When you exceed allowed range the data should be converted to floating point
  - Use `is_int()` or `is_integer()` to test a value to be an integer

- Floating Points
  - Normal style 3.14, -0.54 and scientific notation 0.314E1 or 1.56E10
  - Be wary of normal comparison problems with floats
  - Use `is_float()` or `is_real( )` to test a value to be a float

PHP Language Overview

- Strings
  - Sequence of characters of any length (no char type)
  - Single and double quotes to delimit but be careful there are differences
  - **Careful within “” a variable like $name is expanded to its value but within ‘’ its is treated as a string**
    - **‘ ‘ printing should be faster executing since it doesn’t have to interpolate variables**
  - Escape sequences a la C used within “” like \n, \”, \t, \\, etc.
    - You might find \$ to be quite useful
  - Within ‘ only \\ and \’ are recognized escape sequences
  - Use `is_string()` to test if a value is a string
  - Numerous string manipulation functions as expected

PHP Language Overview

- Booleans
  - False values are defined as
    - The keyword `false`
    - Numbers like 0 and 0.0
    - Empty string “” and “0” string
    - An array with zero elements
    - Object with no values or function
    - Null
  - True is everything else so,
    - True
    - Any number but 0 and 0.0
    - Non-empty strings
  - Use `is_bool( )` function to test for the data type

PHP Language Overview

- Arrays
  - Zero indexed
  - Use `array( )` construct to make create an array
    - `$stooges = array('Larry', 'Curly', 'Moe');`
    - `print $stooges[0];`
    - Use `foreach` to iterate over an array

```
foreach ($stooges as $name)
    {
      print "Hello $name <br />";
    }
```

  - Numerous functions for array manipulation like `sort()`
  - Use `is_array( )` to test for the type

PHP Language Overview

- Objects
  - PHP does support OOP but still has a way to go
  - Define a class with the `class` keyword

```
class Dog {
    var $name = '';
    function name ($thename == NULL)
      {
        if (! is_null($thename)) {
         $this->name = $thename;
         }
       return $this->name;
      }
}
$dog1 = new Dog;
$dog1->name('Tucker');
$dog2 = new Dog;
$dog2->name('Angus');
```

- Use `is_object( )` to detect if something is an object type

PHP Language Overview

- Resources
  - A special data type used in some sense like a file handle (it is an integer under the covers)
  - Often used with a database connection
  - Function `is_resource( )` used to check the data type

- Null
  - Simple null value indicating a lack of data
  - Somewhat useful for garbage collection hinting
  - The function `is_null( )` used to check for a null value

PHP Language Overview

- Variables
  - Prefixed by a dollar sign $
    - $browser , $name , $FAVORITE_NUMBER
  - Variables do not have to be declared before use, they are defined upon first use
  - There is no need to limit a variable to a particular type, PHP allows a variable to hold any data type
    - You can cast variables though to a particular type to be safe
      - $myName = (string) “Thomas”;
        $favNum = (string) $favNum;
  - Unset variables act as null
  - Variables can be created out of strings found in other variables
    - $foo = ‘bar’;
      $$foo = ‘funky’; # now we have $bar = ‘funky’

PHP Language Overview

- Variables can have aliases
  - $foo =& $bar # now $foo is an alias to $bar
  - Reference values can be somewhat tricky in that you can modify values indirectly
  - More confusion comes if you unset a value as it may retained in aliases
    - $foo = “bar”;
      $foo2 =& $foo;
      unset($foo);
      print $foo2; # shows bar
  - Aliases values can be used in functions for parameter passing and to avoid large copy tasks

PHP Language Overview

- You can define constants in PHP using the define construct

```
define('NAME', 'Thomas A. Powell');
// now you can use NAME anywhere
echo 'HI' , NAME
```

  - Convention for constants is to use all caps
  - Constants can only contain scalar data
  - C programmers should watch out it isn't #define
  - Careful with collisions particular if using other people's code
    - get_defined_constants() function may be useful
  - Some built in "MAGIC constants" __LINE__ , __FILE__, __FUNCTION__ , __CLASS__ , __METHOD__ mostly used for debugging

PHP Language Overview

- Scoping
  - Variables declared (first use) within a function are local to that function
  - Variables declared outside a function are considered global
  - Variable within a function using the keyword static are local to the function and retain value between calls
- Function parameters are of course local to the function they are defined for

PHP Language Overview

- Garbage Collection
  - Given the described variable and scope system it is pretty obvious that PHP garbage collects
  - You can use NULL or more appropriately unset( ) to add in memory collection
  - Question - Why might this not be that important to you?

PHP Language Overview

- Operators Overview
  - Generally similar to most languages but in mixed expression you need to watch out for type conversions
  - You can cast yourself using (int), (float), (string), etc. in front of the variable you want to convert
    - $a = “5”; $b = (int) $a;
- Specific operators
  - Arithmetic: +,-,*,/,%
  - String concatenation: .
    - Awkward here if you think OOP style
  - Increment/Decrement: ++, --
    - You can autoincrement/decrement letters!

PHP Language Overview

- More operators
  - Comparison: ==, ===, !=, >, >=, <, <=
    - You can also use <> for !=
  - Bitwise: ~, &, |, ^, <<, >>
  - Logical: &&, ||, !
    - Also supports and, or, xor
  - Assignment: =, +=, -=, /=, *=, %=, .=
  - Conditional: ? :
- Misc. operators
  - @ for error suppression
  - ` ` for shell execution. Watch it!

PHP Language Overview

- Statements are pretty much the same as most imperative languages but you see some funny syntax here and there
  - `if (`*expression*`)`
    statement or block
  - `if (`*expression*`)`
    statement or block
    `else`
    statement or block
  - `elseif` is also supported
  - Special `:` form is used in place of `{ }` and then has an `endif`

PHP Language Overview

- Switch statement
  - Standard style with block

```
switch ($grade) {
  case 'A': echo "Great!";
            break;
  case 'B": echo "Good";
            break;
  …
  default: echo "Error";
}
```

  - Substitute `{` for `:` and `}` for `endswitch` if you like

```
switch ($grade) :
    case …
endswitch;
```

  - Without break statement you get standard switch fall-through effect to create an "or" selector

PHP Language Overview

- While loop
  - Standard style

```
while (expression)
    statement or block;
```

  - Use `continue` statement to skip back to the loop condition check
  - Use `break` statement to break a set of braces
    - Interesting PHP allows you to pass a number to break to indicate the number of blocks to exit. Most languages provide a label target
  - A `do/while` loop is also supported and causes the loop to execute at least once

PHP Language Overview

- For loop
  - Standard style

```
for (start; check; increment;)
     statement or block;
```

  - Alternate style

```
for (start; check; increment;)
   statement1;
    …
   statement n;
endfor;
```

  - Remember you can string statements together in loop start, check, etc. using the comma operator
  - `break` and `continue` statements used to control for loops

PHP Language Overview

- Foreach loop
  - Allows you to iterate over elements in an array

```
foreach ($array as $current)
    statement or block;
```

  - Alternate style

```
foreach ($array as $current)
    statement1;
     …
    statement n;
endforach;
```

  - You can also loop and access both key and value

```
foreach ($array as $key => $value)
    statement1;
     …
    statement n;
endforach;
```

PHP Language Overview

- Return
  - A return statement will exit a function or if at top level the script
- Exit
  - The exit( ) statement will exit the current process/script
  - You can pass a value indicating a status code or error message
  - It is a synonym for die( )
    - die("This script just croaked!");

Calling Functions

- Calling functions whether user defined or system defined is the same
  - `$result=function_name([ parameter,… ] );`
- Functions of course do not need to take parameters (though the very often do) or return a value that is used (though it typically is)
- Example:

```
$len = strlen("UCSD"); // $len = 4
```

Defining Functions

- Basic function definitions of the form

```
function [&]function_name(parameter-list)
   {
      statement(s) often with return(s)
   }
```

- Example

```
function add2($x)   {
   return $x+2;
}

$result = add2(5);
echo $result;
```

Returning Values

- You can return a single value with return, multiple values of course can come within an array.
- Normally a single value is passed back by copy, but for performance you may find a reference indication to be useful
  - The optional `&` in a function definition causes return values to be passed by reference rather than by value (a copy)

Local and Global Variables

- Variables defined within a function are local and not accessible outside the function.
- Global variables are not usable inside a function either unless prefixed by the keyword global
- There are "superglobals" like $_GET[ ] that you can access without the **global** keyword

Example

```
$a = 1;
function foo()
  {
   $a = 2;  // changed a local a
  }
function foo2()
  {
   global $a = 2;  //change the global a
  }
```

Snip….

- We could go on and on and talk about all the various built-in functions for PHP
  - Big PHP Win - the rich ecosystem
- We might get heavy on PHP based OOP
  - Find a surprise how it may be of little value given how PHP is used
  - Explain why PHP4 lives on crazily!
- Etc. etc. etc.
- But our point instead now should be on the Web part of PHP so let’s get to the first part of it

PHP Web Application Basics

Using Forms

- When defining an XHTML form you must specify the method attribute which corresponds to the HTTP method, either GET or POST, which the form contents will be sent to the URL specified by the action attribute.

- Examples

```
<form action="http://www.fake.com/examples/saveit.php"
      method="get" >

<form action="add.php" method="post" >
```

- The submission of the form to be triggered either by the use of a submit button (<input type="submit" />) or some JavaScript that runs the corresponding Form objects submit() method.

Using Forms – GET Method

- Default method if no method attribute specified
- Passes the form data via the URL itself
  - Query parameters following a ?
    - Name-value pairs form-name=value encoded to be URL safe and separated by &'s POST
  - Data size limited to what can be passed via a URL
  - GET is useful to create canned or bookmarkable queries

- Data in query string is encoded as `x-www-form-urlencoded` which replaces spaces with + symbols, special characters with %hex values and separates arguments with & symbol.

Using Forms – GET Method Contd.

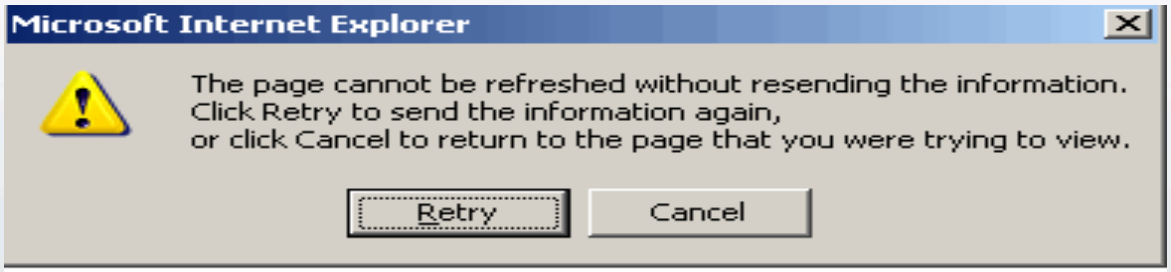
- Examples
  - http://www.fake.com/foo.php?name=Thomas+Powell&num=33
  - http://www.fake.com/foo2.php?
    ?name=Thomas+O%27Reilly&num=38.5&password=secret
- Note that Data may be encoded but it is not secured in any way by default
- "Dirty URLs" with complex query strings may trouble some search engines and are not terribly usable from an end user perspective, but otherwise are normal
  - Remember: Users may fiddle with URL parameters which leads to security violations and unexpected bugs if you don't assume that

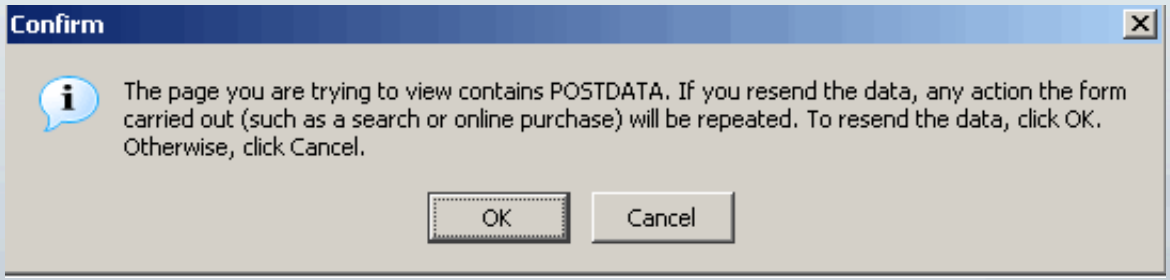
Using Forms – Post Method

- When using the Post method form data is sent via standard input within the HTTP message body
- Post considerations
  - Does not have any limitations to content size
    - There can be a timeout issue though
  - Not as easily manipulatable by some end-users, though no problem for a skilled individual
  - Encoded in the same manner as GET submissions
    - Name=Al+O%27Reilly&Age=30&Sex=male

Using Forms – Post Method

- Not static in the sense of easy reuse, bookmarking, etc.
- Consider how the browser provides you a facility to reuse posted data to avoid user aggravation





Traditional Form Handling in PHP

- In traditional PHP if you had a page with a form with fields like

```
<form action="page2.php" method="get">
  <input type="text" name="username" />
  <input type="text" name="age" />
  <input type="submit" value="send" />
</form>
```

Then after submission on page2.php you should automatically have variables `$username` and `$age` which corresponded to whatever the user had filled in on the previous page.

- So in page2.php you might simply have

```
<?php
 print "Hi $username you are $age years old!";
?>
```

Registered Globals Issues

- For security and program safety issues this style is typically not available by default in many PHP installations though you can enable it by setting register_globals to on in the php.ini file.
  - Issue: Injecting data directly into your program via GET or POST
  - Result: Trigger unintended actions (ex. Code flow, debug statements, etc.)
- Checking for setting
  - `ini_get('register_globals');`
  - Use with @ to suppress errors in an if to decide what to do
  - Note corresponding `ini_set()` does not allow register_globals to be set at runtime

Safer Form Handling

- Rather than relying on registered globals use the superglobal variable arrays $_GET, $_POST or $_REQUEST which includes GET, POST, and Cookie vals

- For example to fetch out the values in the next page (page2.php) we might use code like if a GET method was used

```
$username = $_GET['username'];
$age = $_GET['age'];
```

or alternatively regardless of method

```
$username = $_REQUEST['username'];
$age = $_REQUEST['age'];
```

- The added complexity forces a safer coding posture

Example: form1.html -> form1.result.php

Importing Request Variables

- For those folks that really like the style of coding that registered globals provides you can use the `import_request_variables( )` function.
  - Syntax: `import_request_variables("what" [,prefix]);`
  - Where what is a string that contains a string containing g,p, and/or c indicating to import GET, POST, or COOKIE values respectively
  - The prefix value is optional and will append the string specified in front of any imported values. For safety sake it is encouraged
  - Example: `import_request_variables("gp","r_");`
  - Example: form2.html -> form2.result.php

Self Posting

- One design pattern that PHP developers often use with forms is the idea of a single file form/response

- The basic idea is to have the page fork depending on if its collecting to user data or responding to the input. It can be extended of course to address validation as well

- Pseudo code looks something like

```
if  form data not collected
        print form fields and allow user to submit
else if invalid form data
            print form fields with errors and allow resubmit
      else
         perform form action (and often redirect)
```

Self Posting Contd.

- An easy way to set the action of a self posting form is to use the PHP_SELF value in the form action
  - `<form action="<?= $_SERVER['PHP_SELF'] ?>" method="GET or POST" >`
  - Often times the trigger is to look for the existence of a particular variable being set either in a submit button or <input type="hidden"> or to look at the method the page is called with GET or POST
  - You may also note the troubling issue with form posting that happens particularly with POST.

What Could Go Wrong?

- Well just about everything, you should assume nothing works
  - What happens if the fields aren't filled in that are needed?
  - What happens if you get the wrong type for fields?
  - What about if someone figures out how to send fields they shouldn't or avoid maxlengths you might have set?
  - Could data be badly formatted?
- Rule: Never trust user input

Basic Validations

- isset()
  - Make sure a variable is set before you play with it
- trim()
  - Trim a variable of extra spaces
- empty()
  - Determines if a value is empty
- Type casting
  - Cast a variable the way you want it
  - $age = (integer) _POST['age'];
  - Settype($age, "integer");
- Explicit range checking
  - If statements
  - Regular expressions

Form Action

- Up until now we have echoed form data, we try a simple example to show what's next by sending an email based upon form data
  - Basic syntax:

```
bool mail ( string to, string subject, string message [, string additional_headers [, string additional_parameters]] )
```

  - Example:

```
mail( "tpowell@pint.com", "Feedback Form Results",$message, "From: $email"
```

  - Manual entry -http://us3.php.net/manual/en/function.mail.php

Reading Headers and Environment

- If you recall from our lengthy HTTP discussion there is more than just reading the query string or message body (as in a POST), you can also gain valuable insight from the various headers transmitted in a request
  - Useful headers reminder – User Agent, Accept, Accept-Language, etc.
- You also find that the environment in which the transaction takes place might be quite useful
  - Time and IP address in particular might be of use

Reading Headers and Environment Contd.

- Most of the interesting items are in the $_SERVER[] super global array
  - All the HTTP headers will be prefixed with HTTP_
    - Examples – HTTP_ACCEPT, HTTP_REFERER, HTTP_USER_AGENT, etc.
  - Others are variables
    - PHP_SELF, REMOTE_ADDR, REMOTE_HOST, REQUEST_METHOD but should be familiar from the "environment variables from CGI"

Header Output

- You can output HTTP headers as you like using the header() function.
  - header("Content-Type: text/plain");
- If you understand the typical HTTP data stream you can see that this must be output first for it to work
  - Any screen output before this line will cause PHP to do its normal duty creating an HTML header stream and application of the function later should result in an error
- Consider the following used right and wrong

```
<?php
 header("Content-type: text/plain");
?>
<h1>I am supposed to be HTML!</h1>
```

Header Trouble

```
<h1>I am supposed to be HTML!</h1>
```

I am supposed to be HTML!

Warning: Cannot modify header information - headers already sent by (output started at /usr/local/www/data-dist/examples/headers.php:2) in **/usr/local/www/data-dist/examples/headers.php** on line **3**

I am supposed to be HTML!

Response Headers

Date	Tue, 29 Jan 2008 21:06:03 GMT
Server	Apache/1.3.37 (Unix) PHP/5.2.0 with Suhosin-Patch
X-Powered-By	PHP/5.2.0
Keep-Alive	timeout=15, max=100
Connection	Keep-Alive
Transfer-Encoding	chunked
Content-Type	text/html

Fixes and Consequences

- Of course in modern PHP such problems of headers disappear!
  - Not without penalty now we may buffer the response just in case which could limit performance
- The reality of such a fine point as this last example is to remind students that lurking underneath all this is just plain old HTTP
- Up next dealing with HTTP lack of statefuleness.