The History of PHP

- Server-side scripting language
  - PHP = Personal Home Page tools
  - Somewhat like C but much higher level, OOP model added later
  - Especially with Apache/Linux/MySQL
  - PHP is the most widely used scripting language for web programming, used today with many commercial sites
  - Runs on both Unix and Windows platforms, with most web servers
- Available for free
  - http://www.php.net

The History of PHP

- Rasmus Lerfdorf
- Not a trained computer scientist
- Consultant building dynamic web sites - got tired of doing the same thing over and over in C

PHP & HTML

- PHP extends HTML pages by adding server-executed code segments to HTML pages.
- The output of the execution of the PHP code is merged into the HTML page.

```
<?php
    echo "Hello World. Today is ".date( ).". ";
?>How are you?
```

Hello World. Today is Wednesday. How are you?

echo & print

- ?

MySQL

- MySQL is one of the most popular free and open source database engines in the market place.
- MySQL powers Facebook, Yahoo!, and millions of other dynamic web sites.

```
INSERT INTO users VALUES('Smith', 'John', 'jsmith@mysite.com');

SELECT surname,firstname FROM users WHERE
email='jsmith@mysite.com';
```

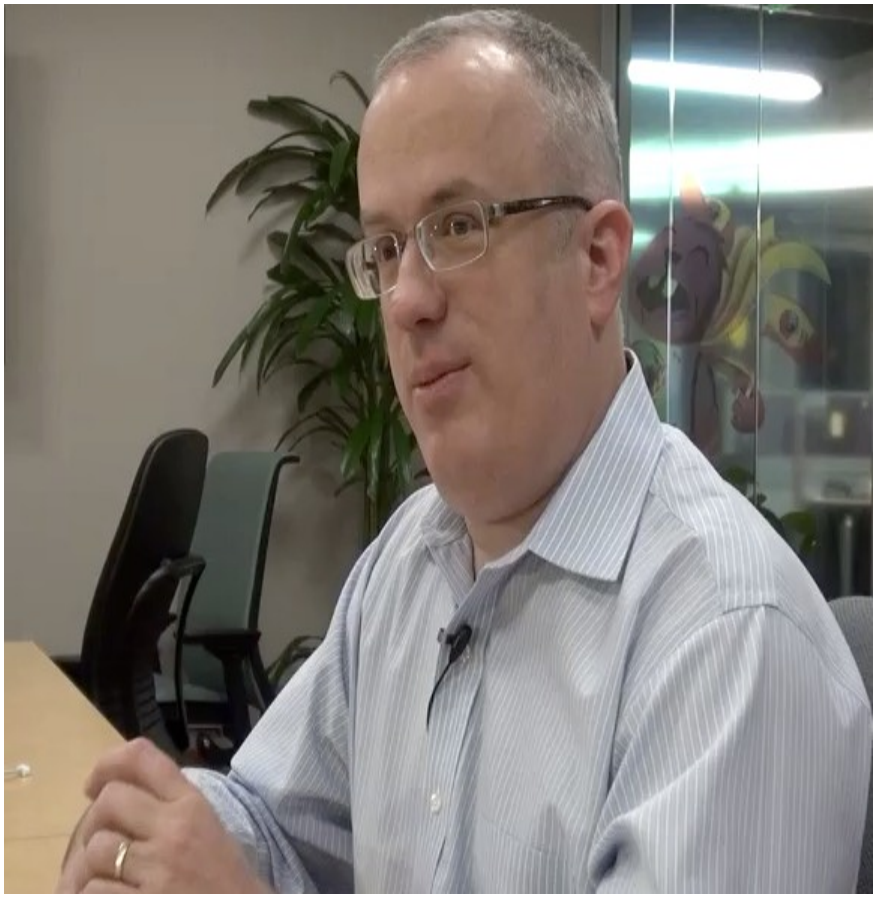
JavaScript

- JavaScript is a C-like programming language that can be included in an HTML web page. JavaScript allows the builder of a web page to embed dynamic elements within a web page. JavaScript programs run in the browser (i.e. the Client)

```
<script type="text/javascript">
  document.write("Hello World. Today is " + Date() );
</script>
```

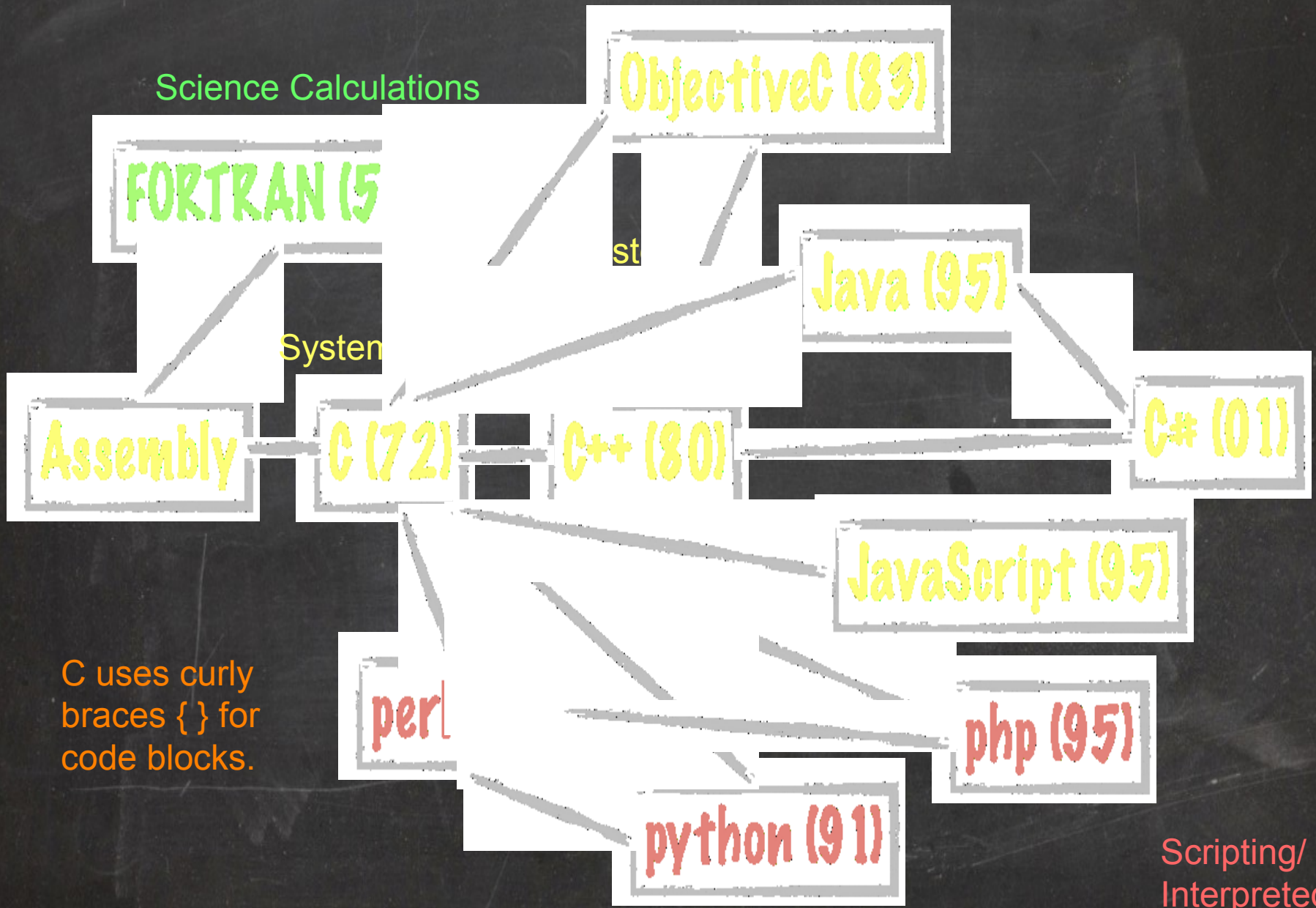
JavaScript: Brendan Eich

- Invented JavaScript in May 1995 in ten days



About the PHP Language

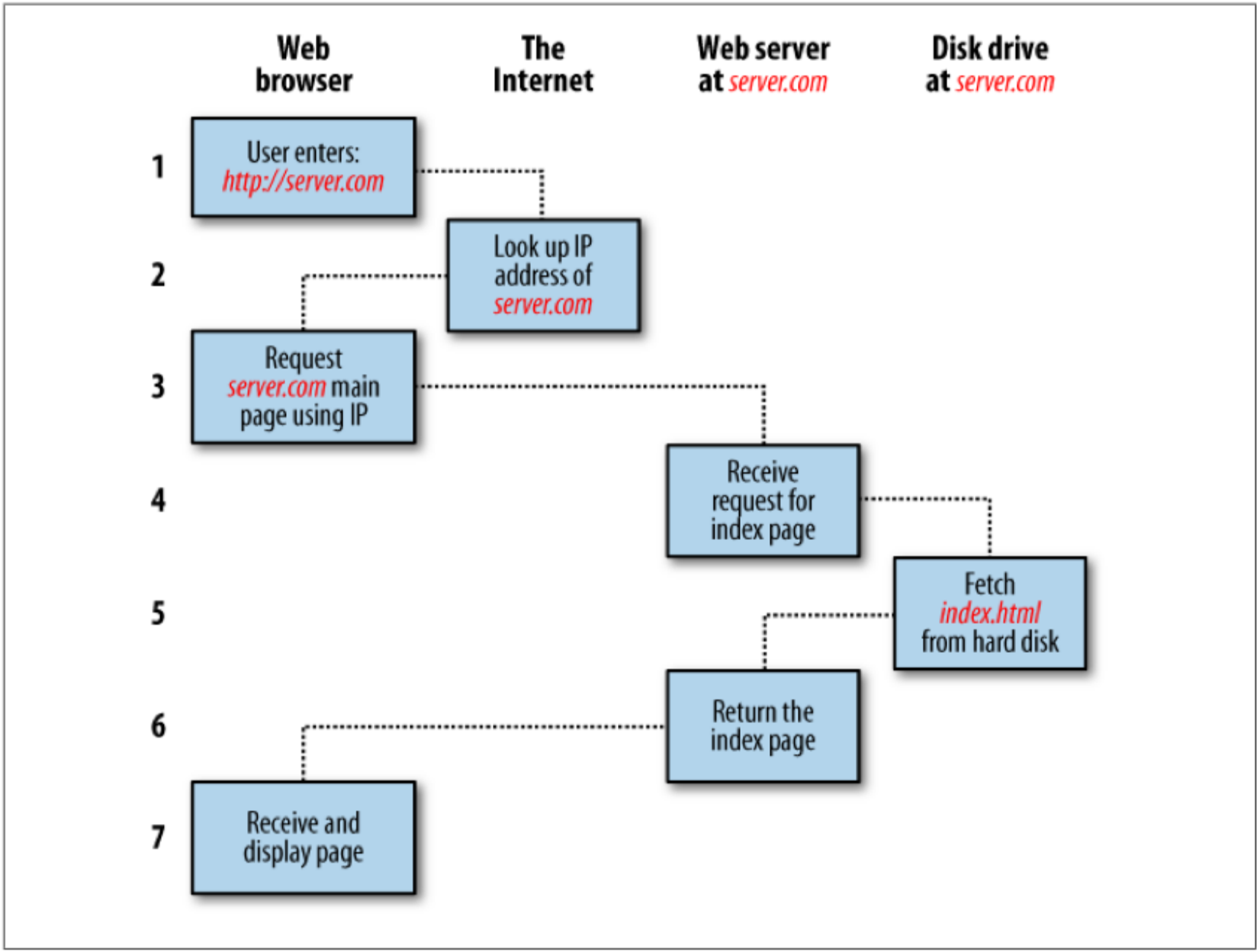
- Syntax is inspired by C
  - Curly braces, semicolons, no significant whitespace
- Syntax inspired by perl
  - Dollar signs to start variable names, associative arrays
- Extends HTML to add segments of PHP within an HTML file.



http://en.wikipedia.org/wiki/History_of_programming_languages

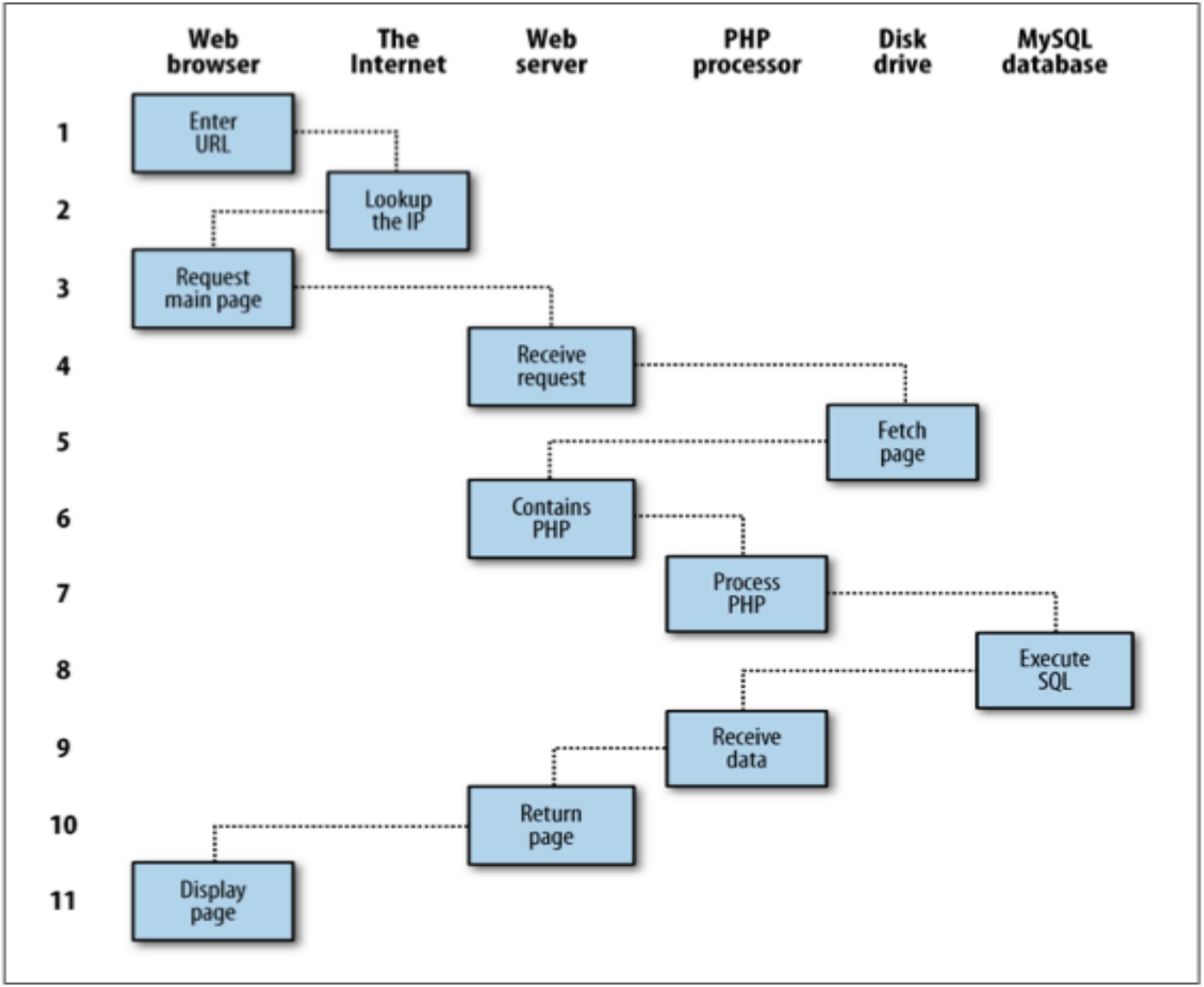
Request / Response Cycle

- You enter http://server.com into your browser's address bar.
- Your browser looks up the IP address for server.com.
- Your browser issues a request for the home page at server.com.
- The request crosses the Internet and arrives at the server.com web server.
- The web server, having received the request looks for the web page on its hard disk.
- The web page is retrieved by the server and returned to the browser.
- Your browser displays the web page.


Web browser
The Internet
Web server at server.com
Disk drive at server.com
1
User enters: http://server.com
2
Look up IP address of server.com
3
Request server.com main page using IP
4
Receive request for index page
5
Fetch index.html from hard disk
6
Return the index page
7
Receive and display page

In More Detail...

- You enter http://server.com into your browser's address bar.
- Your browser looks up the IP address for server.com.
- Your browser issues a request to that address for the web server's home page.
- The request crosses the Internet and arrives at the server.com web server.
- The web server, having received the request, fetches the home page from its hard disk.
- With the home page now in memory, the web server notices that it is a file incorporating PHP scripting and passes the page to the PHP interpreter.
- The PHP interpreter executes the PHP code.
- Some of the PHP contains MySQL statements, which the PHP interpreter now passes to the MySQL database engine.
- The MySQL database returns the results of the statements back to the PHP interpreter
- The PHP interpreter returns the results of the executed PHP code, along with the results from the MySQL database, to the web server.
- The web server returns the page to the requesting client, which displays it.


Web browser
The Internet
Web server
PHP processor
Disk drive
MySQL database
1
2
3
4
5
6
7
8
9
10
11
Enter URL
Lookup the IP
Request main page
Receive request
Fetch page
Contains PHP
Process PHP
Execute SQL
Receive data
Return page
Display page

Internet

HTML

JavaScript

AJAX

Cookies

HTTP

Request

Response

GET

PHP

MySql

How do Web Servers work?

- Client specifies document at a specific web address that is desired (specified by a URL)
  - Ex: http://www.just.edu.jo/
- If the document is HTML or text, the server simply forwards it back to the client
  - If it is text, it is shown unaltered in the browser
  - If it is HTML it is rendered in the client's browser
    - html tags are interpreted and result is shown to the user

How do Web Servers work?

- However, the requested document may be an executable script, or it may be HTML with an embedded script
  - The script could be written in any of many different web scripting languages
- In these cases, the server executes the script
  - If the entire document was a script, the server simply sends the output back to the client
  - If the document had an embedded script, the script sections are replaced with the output and the modified document is then sent to the client

How do Web Servers work?

- Note that **the client never sees the server-side script code**
  - This is important – typically client should not see code that the server executes to process requests
  - The server may be accessing files whose names should not be seen, or preprocessing data that it does not want the client to see

Introduction. to HTML

- HTML is a mark-up language
  - Idea is that extra characters / symbols in the text provide information to a parser, which uses that information to render the document in a certain way
  - Ex:

    Hello There
    - The tags do not appear in the rendered document
    - The parser utilizes them to alter the appearance of the text

Introduction. to HTML

- HTML has evolved greatly over the years
  - New tags have been added
  - Some tags have been removed
  - Syntax has been standardized
- The current version is HTML 5
  - Still not universally used

Introduction to PHP

- PHP scripts are often embedded within HTML documents
  - The server processes the HTML document, executing the PHP segments and substituting the output within the HTML document
  - The modified document is then sent to the client
  - As mentioned previously, **the client never sees the PHP code**
    - The only reason the client even knows PHP is involved is due to the file extension → .php

PHP Program Structure

- PHP, as with many scripting languages, does not have nearly the same structural requirements as a language like Java
  - A script can be just a few lines of code or a very large, structured program with classes and objects
    - The complexity depends on the task at hand
  - However, there are some guidelines for incorporating PHP scripts into HTML files

Processes

- When a PHP file is requested, the PHP interpreter parses the entire file
  - Any content within PHP delimiter tags is interpreted, and the output substituted
  - Any other content (i.e. not within PHP delimiter tags) is simply passed on unchanged
  - This allows us to easily mix PHP and other content (ex: HTML)
    - See:
      - http://us3.php.net/manual/en/language.basic-syntax.phptags.php
      - http://us3.php.net/manual/en/language.basic-syntax.phpmode.php

Consider the following PHP file

```
<!DOCTYPE html>
<html>

 <head>
  <title>Simple PHP Example</title>
 </head>
 <body>
    <?php echo "<p><h1>Output</h1>";
          echo "<h2>Output</h2>";
          echo "<h3>Output</h3></p>";
    ?>
    <script language="PHP">
          echo "\n<b>More PHP Output</b>\n";
          echo "New line in source but not rendered";
          echo "<br/>";
          echo "New line rendered but not in source";
    </script>
</body>
</html>
```

HTML 5 Document

Root HTML Tag

Document Head

PHP Code

DOC BODY

Now consider the resulting HTML

```
<!DOCTYPE html>
<html>
 <head>
  <title>Simple PHP Example</title>
 </head>
 <body>
    <p><h1>Output</h1><h2>Output</h2><h3>Output</h3></p>
<b>More PHP Output</b>
New line in source but not rendered<br/>New line rendered but not in so
</html>
```

- How will it look in the browser?
  - Look at it in the browser!

Variable Names

- Start with a dollar sign ($) followed by a letter or underscore, followed by any number of letters, numbers, or underscores
- Case matters

```
$abc = 12;
$total = 0;
$largest_so_far = 0;
```

```
abc = 12;
$2php = 0;
$bad-punc = 0;
```

Variable Declarations

- Variables do not need to be declared before you use them.
- Example: $var1 = 25;
- To help set off a variable identifier within a string, you can surround it with curly brackets.
- This will become helpful when we start discussing arrays and objects.
- Example: echo "The value is {$var1}." will display "The value is 25."

Data Types

- Scalar types
  - boolean
  - float
  - integer
  - string
- Compound types
  - array
  - object

Using Scalar Types

- A boolean variable can be assigned only values of ***true*** or ***false***.

```
$answer = false;
$finished = true;
```

- An integer is a whole number (no decimal point)

```
$age = 31;
```

Using Scalar Types (continued)

- A float has a decimal point and may or may not have an exponent

```
$price = 12.34;
$avog_num = 6.02e23; //6.02x10^23
```

- A string is identified as a sequence of characters

```
$name = "John Smith";
```

- Strings can be concatenated using a dot (.)

```
$name = "John" . " Smith";
```

Constants

- Constants associate a name with a scalar value.
- Constants are defined using the function ***define()***.

```
define("PI", 3.141593);
```

- There are a number of predefined constants. These include:
  - M_E = 2.718281828459
  - M_PI = 3.1415926535898
  - M_2_SQRTPI = 1.1283791670955 (Square root of pi)
  - M_1_PI = 0.31830988618379 (Square root of 1/pi)
  - M_SQRT2 = 1.4142135623731 (Square root of 2)
  - M_SQRT1_2 = 0.70710678118655 (Square root of ½)

Arithmetic Operators

Operator	Operation	Example	Result
+	Addition	`$y = 2 + 2;`	$y will contain 4
–	Subtraction	`$y = 3;` `$y = $y - 1;`	$y will contain 2
/	Division	`$y = 14 / 2;`	$y will contain 7
*	Multiplication	`$z = 4;` `$y = $z * 4;`	$y will contain 16
%	Modulo	`$y = 14 % 3;`	$y will contain 2
++	Increment	`$y = 7;` `$y++;`	$y will contain 8
--	Decrement	`$y = 7;` `$y--;`	$y will contain 6

Bitwise Logical Operations

- ~ Bitwise NOT operator: Inverts each bit of the single operand placed to the right of the symbol
- & Bitwise AND: Takes the logical-bitwise AND of two values
- | Bitwise OR operator: Takes the logical-bitwise OR of two values
- ^ Bitwise XOR: Takes the logical-bitwise exclusive-OR of two values

Bitwise Shift Operations

- << Left shift: Shifts the left operand left by the number of places specified by the right operand filling in with zeros on the right side.
- >> Sign-propagating right shift: Shifts the left operand right by the number of places specified by the right operand filling in with the sign bit on the left side.
- >>> Zero-fill right shift operator: Shifts the left operand right by the number of places specified by the right operand filling in with zeros on the left side.

Flow Control

- As in JavaScript, flow control consists of a number of reserved words combined with syntax to allow the computer to decide which parts of code to execute, which to jump over, and which to execute multiple times.
- For the most part, the flow control that you learned for JavaScript is the same for PHP.

If-Statement

- The code below represents the syntax of a typical *if-statement*:

```
if ($grade > 93)
        print "Student's grade is A";
```

- If grade was 93 or below, the computer would simply skip this instruction.

If-Statement (continued)

- Just like JavaScript, multiple instructions may be grouped using curly brackets. For example:

```
if ($grade > 93)
{
   print "Student's grade is A";
   $honor_roll_value = true;
}
```

If-Statement (continued)

- As in JavaScript, the programmer can string together if-statements to allow the computer to select from one of a number of cases using *elseif* and *else*. (Note that JavaScript allows ***else if*** while PHP uses ***elseif***.)
- For example:

```
if ($grade > 93)
   print "Student's grade is an A";
elseif ($grade > 89)
   print Student's grade is an A-";
else
   print "Student did not get an A";
```

Comparison Operators

- > Returns true if the first value is greater than the second
- >= Returns true if the first value is greater than or equal to the second
- < Returns true the first value is less than the second
- <= Returns true if the first value is less than or equal to the second
- == Returns true if first value is equal to second
- != Returns true if first value is not equal to second

Logical Operators

- ! Returns true if its operand is zero or false
- && Returns false if either operand is zero or false
- || Returns false if both operands are zero or false

Switch-Statement

- The switch statement can be used as an alternative to the if, elseif, else method.

```
switch($menu)
{
    case 1:
          print "You picked one";
          break;
    case 2:
          print "You picked two";
          break;
    default:
          print "You did not pick one or two";
          break;
}
```

Switch-Statement (continued)

- Note that if a break is not encountered at the end of a case, the processor continues through to the next case.
- Example: If $var1=1, it will print both lines.

```
switch($var1)
{
    case 1:
          print "The value was 1";
    default:
          print "Pick another option";
          break;
}
```

While-loop

- PHP uses the *while-loop* just like JavaScript.
- Like the if-statement, this format also uses a condition placed between two parenthesis
- As long as the condition evaluates to true, the program continues to execute the code between the curly brackets in a round-robin fashion.

While-loop (continued)

- Format:

```
while(condition)
{
  statements to execute
}
```

- Example:

```
$count = 1;
while($count < 72)
{
  print "$count ";
  $count++;
}
```

do … while loop

- The do … while loop works the same as a while loop except that the condition is evaluated at the end of the loop rather than the beginning
- Example:

```
$count = 1;
do
{
    print "$count ";
    $count++;
}while($count < 72);
```

for-loop

- In the two previous cases, a counter was used to count our way through a loop.
- This task is much better suited to a for-loop.

```
for ($count = 1; $count < 72; $count++)
{
   print "$count ";
}
```

- A "break" can be used to break out of a loop earlier.

Type Conversion

- Different programming languages deal with variable types in different ways. Some are strict enforcing rules such as not allowing an integer value to be assigned to a float.
- The process of converting from one data type to another is called "casting".
- To convert from one type to another, place the type name in parenthesis in front of the variable to convert from.
- In some cases, there are functions that perform the type conversion too.

Some Examples of Type Conversion

- $ivar = (int) $var;
- $ivar = (integer) $var;
- $ivar = intval($var);
- $bvar = (bool) $var;
- $bvar = (boolean) $var;
- $fvar = (float) $var;
- $fvar = floatval($var);
- $svar = (string) $var;
- $svar = stringval($var);

Examples of Type Conversion (continued)

Value	Cast to int	Cast to bool	Cast to string	Cast to float
null	0	false	""	0
true	1	true	"1"	1
false	0	false	""	0
0	0	false	"0"	0
3.8	3	true	"3.8"	3.8
"0"	0	false	"0"	0
"10"	10	true	"10"	10
"6 feet"	6	true	"6 feet"	6
"foo"	0	true	"foo"	0

Type Conversion (continued)

- PHP can automatically convert types too.
- If a variable is used as if it were a different type, the PHP script engine assumes a type conversion is needed and does it for you.
- Examples:

```
$var = "100" + 15;   // var$ set to integer = 115
$var = "100" + 15.0; // var$ set to float = 115
$var = 15 + " bugs"; // var$ set to integer = 15
$var = 15 . " bugs"; // var$ set to string = "15
bugs"
```