SQL Application Programming

CS 377: Database Systems

Recap: SQL Queries

```
SELECT      [DISTINCT] <attribute list>
FROM         <table list>
[WHERE      <condition on the tables>]
[GROUP BY <grouping attributes>]
[HAVING      <group condition>]
[ORDER BY  <attribute list>   ASC | DESC]
[LIMIT       <number of tuples>]
```

Recap: SQL Usage

- Stand-alone: user enters SQL commands via a command line or in a GUI
- Embedded in a host language: SQL commands are embedded (written inside) an “ordinary” program in a high level language (e.g., Java, C++, C, etc.)
- Library-based: SQL commands are made available through library functions (e.g., Java, Python)
- Web-based: various languages with extensions allow webpages to access database server

Today's Lecture

1. Introduction to Database Programming
2. JDBC
3. PHP

Standard Client / Server Model

- Server: machine with specific functionalities (e.g., file server, print server, email server)
- Client: user machine with user interface capabilities and local processing
- Application / web server: intermediate layer to check security and process data

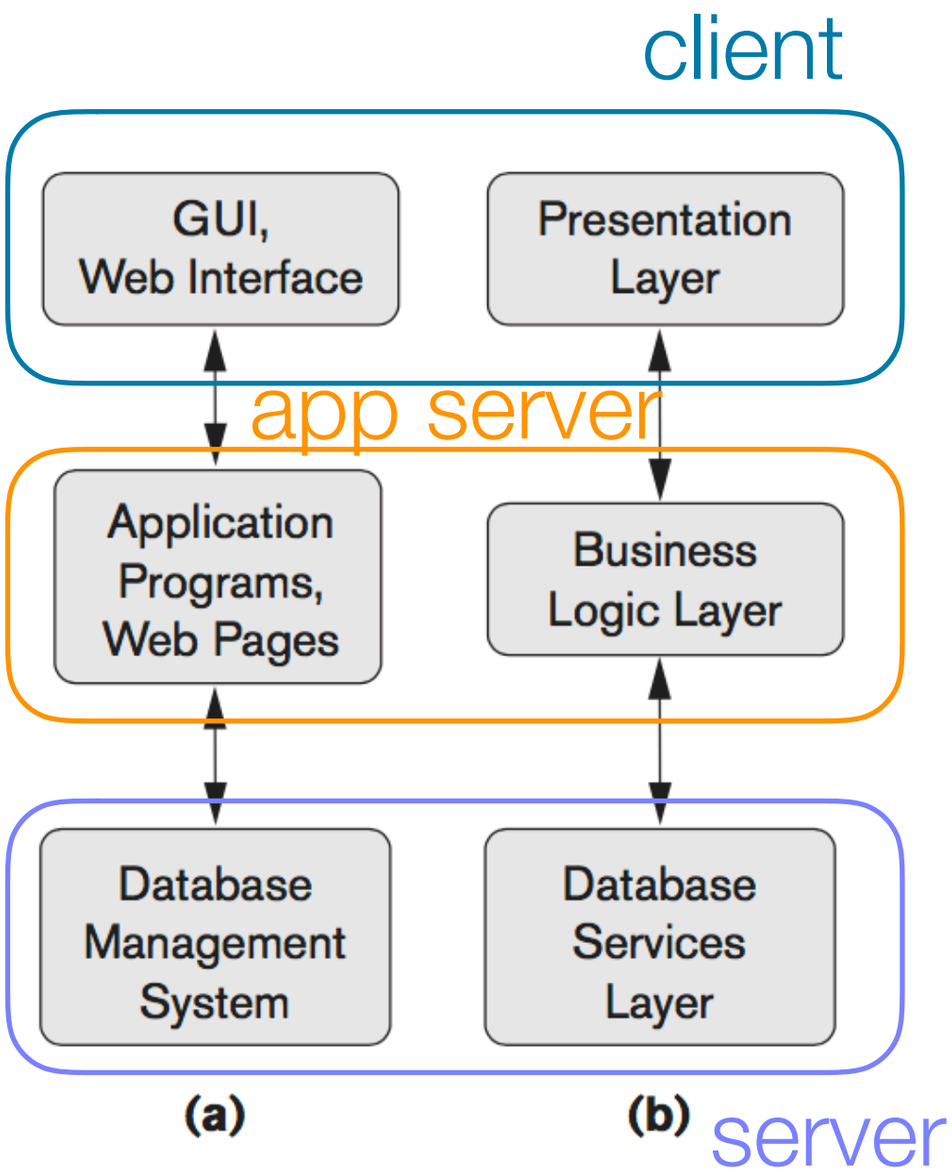


Figure 2.7 from book

Typical Database Program Sequence

1. Establish / open a connection to database server — specify url of database server and account / password details

2. Submit database commands (e.g., queries, updates, etc.) — most types of SQL statements can be included

3. Terminate / close the connection to the database

Impedence Mismatch

- Problems that occur because of differences between database model and programming language model
  - Bind attribute data types to programming language data types
  - Map between query result data structure (sets or multi sets of tuples) to appropriate data structure in programming language

Java Database Connection (JDBC)

JDBC

- Provides capability to access a database server through a set of library functions
  - Set of library functions forms a standardized Application Program Interface (API)
  - Allows programmer to send SQL statements for execution and query retrieval
- Supported by most major database vendors

JDBC Program: Step 1

1. Establish / open a connection to database server

   I. Import JDBC library (java.sql.*)

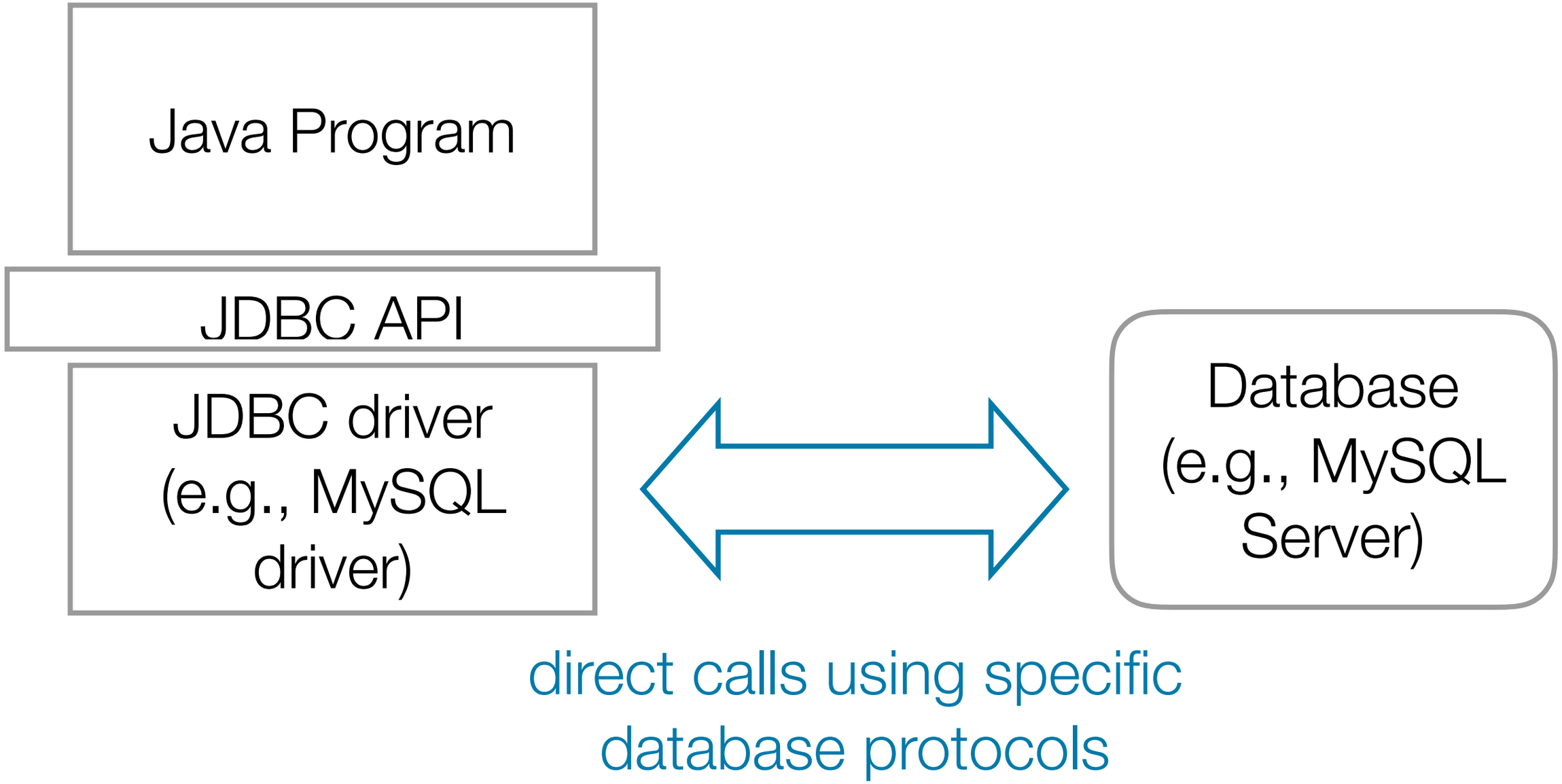
   II. Load appropriate JDBC driver

   III. Create a connection object

JDBC: java.sql Package

- Library functions are contained in the java.sql package
- Every JDBC must import the classes in this package
  import java.sql.*
- Designed to access any database platform
  - Un-avoidable that there will be a system-dependent component to access a specific database type
  - Drivers are used to support communication transparency between the different vendors

JDBC Driver


Java Program
JDBC API
JDBC driver (e.g., MySQL driver)
Database (e.g., MySQL Server)
direct calls using specific database protocols

JDBC Driver

- A communication driver is a system dependent software module that is written specifically according to a given communication protocol
- Different vendors can provide the same data service through different communication protocols
- A JDBC program must first load the desired communication driver
  - Each system has its own way to load the driver (Biggest headache in JDBC programming)

Dealing with SQLException

Most methods in the JDBC SQL library will throw SQLException

- Catch the exception

```
try
{
 < method in java.sql package >
}
catch ( Exception e)
{
< statements to execute when there is an error>
}
```

- Specify throws SQLException to each method in your program - exit program upon error

DriverManager: Managing the JDBC Driver

- Attempt at standardization of loading the JDBC communication driver
- Contains methods for managing a set of JDBC drivers
- Methods contained in the class:
  - **static void registerDriver(Driver driver)** - registers the given driver with the device manager by reading in the driver code from the installed library
  - **static Connection getConnection(String url, String user, String password)** - attempts to establish a connection and should only be used after the driver was registered

Registering a Driver (Textbook Way)

- Standard way to register a platform dependent JDBC driver is to use the registerDriver() method
- Example: registering the JDBC driver for Oracle:
  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
- Unfortunately not all vendors use this approach to load its JDBC driver (e.g., MySQL)

Registering a MySQL Driver

- Exploits Java's built-in capability to load a class
- Driver is loaded using the java.lang.reflect package
- Syntax:
  Class.forName("com.mysql.jdbc.Driver");

Dynamic Loading Feature

- Java has the ability to load user-written classes dynamically into a compiled program and execute it
- Load a different class that has a method with the same name, you can get the behavior of the method to change

Dynamic Loading: java.lang.reflect

- Commonly used by programs which require ability to examine or modify runtime behavior of applications
  - Applications: create instances of objects using their fully-qualified names
  - Debuggers & Test Tools: examine private members of classes
  - Class browser: enumerate members of a class

Dynamic Loading: java.lang.reflect

- Drawbacks:
  - Performance overhead
  - Security restrictions
  - Exposure of internals

Location of the JDBC Driver Software

- Java must be able to find (locate) the JDBC Driver
  - CLASSPATH variable must point to the SQL Java JDBC library (depends on your installation)
- Include the PATH to run the JDBC program
- Example:
  java -cp <location of jdbc driver library> <your program>

Create a Connection Object

- Network connection to a database server is established using the getConnection method in the DriverManager class

- Syntax:
  Connection SQLconnection; // variable for connection
  SQLconnection = DriverManager.getConnection(URL, user, password);

location of database server

JDBC: SQL Connection

- Connection contains a reference to the data structure that stores information on the network connection
- Connection must be passed to subsequent methods to communicate with the MySQL server
- Only need a single connection to the server
- URL must contain the protocol, the host name, the port number, and the database name (e.g., “jdbc:mysql://cs377db.mathcs.emory.edu:3306/companyDB”)

JDBC Program: Step 2

2. Submit database commands

   I. Create statement object

   II. Send statement

   III. Process query results

Creating a Statement Object

- java.sql.Statement class is used to execute a SQL statement (by sending it to the database)
- It also has buffers to receive the result tuples
- Before submitting a query, you must first create a Statement object for the processing of the query
- Syntax:
  Statement SQLstatement; // variable ref for obj
  SQLstatement = <sqlconnection>.createStatement();

Submit a SQL Query

- executeQuery method sends the SQL query using the DBMS connection to the DBMS server for processing

- Syntax:

```
ResultSet rset;    // reference variable for results
rset = SQLstatement.executeQuery("<SQL query>") ;
```

- Example:

```
ResultSet rset;    // reference variable for results
rset = SQLstatement.executeQuery("select * from
employee");
```

Submit a SQL Query

- Statement object can be recycled if SQL queries are executed in serial
  - Execute one query and read the result completely before executing next query
- For multiple queries at the same time, you need to create multiple Statement objects — one per parallel query

Submit a SQL Update

- executeUpdate method sends the SQL command using the DBMS connection to the DBMS server for processing
- SQL command maybe an INSERT, UPDATE, or DELETE statement or even creation of a table or constraint
- Example:
  rset = SQLstatement.executeUpdate("delete * from employee");
- Returns an update count

Process Query Results

- ResultSet returns an iterable that contains all the tuples in the output relation

- Retrieve one tuple:
  rset.next() - returns null if there are no more tuples otherwise returns the next tuple

- Retrieve all tuples in the result set

```
while (rset.next() != null) {
   <process the tuple>
}
```

Demo: Employee.java

Useful ResultSet Methods

Java Function	Description
beforeFirst()	moves the read cursor to the front of the ResultSet object, just before the first row (can be used to re-read the data again)
first()	moves the read cursor to the first row of the ResultSet object
absolute(rowNum)	moves the read cursor to the row rowNumber of the ResultSet object
afterLast()	moves the read cursor to the end of this ResultSet object, after the last row (can be used to read the data in reverse order)
last()	moves the read cursor to the last row of this Result object (can be used to find number of rows)
getRow()	returns the row index of the current row

Demo: EmployeeRSMethod.java

Retrieve Field in the Result Tuple

Java Function	SQL Type	Description
getInt(index)	INTEGER	returns attribute at position index as a <javatype> (e.g., getInt —> int type)
getLong(index)	BIG INT	
getFloat(index)	REAL / FLOAT	
getDouble(index)	DOUBLE	
getBignum(index)	DECIMAL	
getBoolean(index)	BIT / BOOLEAN	
getString(index)	VARCHAR / CHAR	
getDate(index)	DATE	
getTime(index)	TIME	
getTimeStamp(index)	TIMESTAMP	
getObject(index)	any type	

Metadata About ResultSet

- ResultSetMetData class contains meta information about the ResultSet
  - Methods to retrieve/obtain the meta data
  - Variables to store values of the meta data
- Syntax:

```
ResultSet rset;
rset = SQLstatement.executeQuery("<SQL query">);
ResultSetMetaData metaData;
metaData = rset.getMetaData();
```

Useful ResultSetMetaData Methods

- **int getColumnCount()**: returns the number of columns in the tuples of the ResultSet
- **String getColumnName(int columnIndex)**: returns the name of the column whose index is specified
- **String getColumnType(int columnIndex)**: returns the integer code for the data type of the attribute whose index is specified (see java.lang.Types for the codes)
- **String getColumnTypeName(int columnIndex)**: returns the type of column whose index is specified

Useful ResultSetMetaData Methods

- **int getColumnDisplaySize(int columnIndex)**: returns the display width (number of characters needed to display the value) of the attribute whose index is specified
- **int getPrecision(int columnIndex)**: returns the number of digits of the field/column whose index is specified - data type must be numeric
- **int getScale(int columnIndex)**: returns the number of decimal places of the field/column whose index is specified - data type must be numeric
- **String getColumnClassName(int columnIndex)**: returns the name of the Java class (e.g., “java.lang.String”) for the attribute whose index is specified

Demo: MetaData.java

JDBC Program: Step 3

3. Close connection

   I. Close & free result set

   II. Close & free statement object

   III. Close & free connection object

Closing Connections

- Upon completion, you should close the various connections and free resources
- Close and free the result set:
  rset.close();
- Close and free the Statement object
  SQLstatement.close();
- Close and free the Connection buffer
  SQLconnection.close();

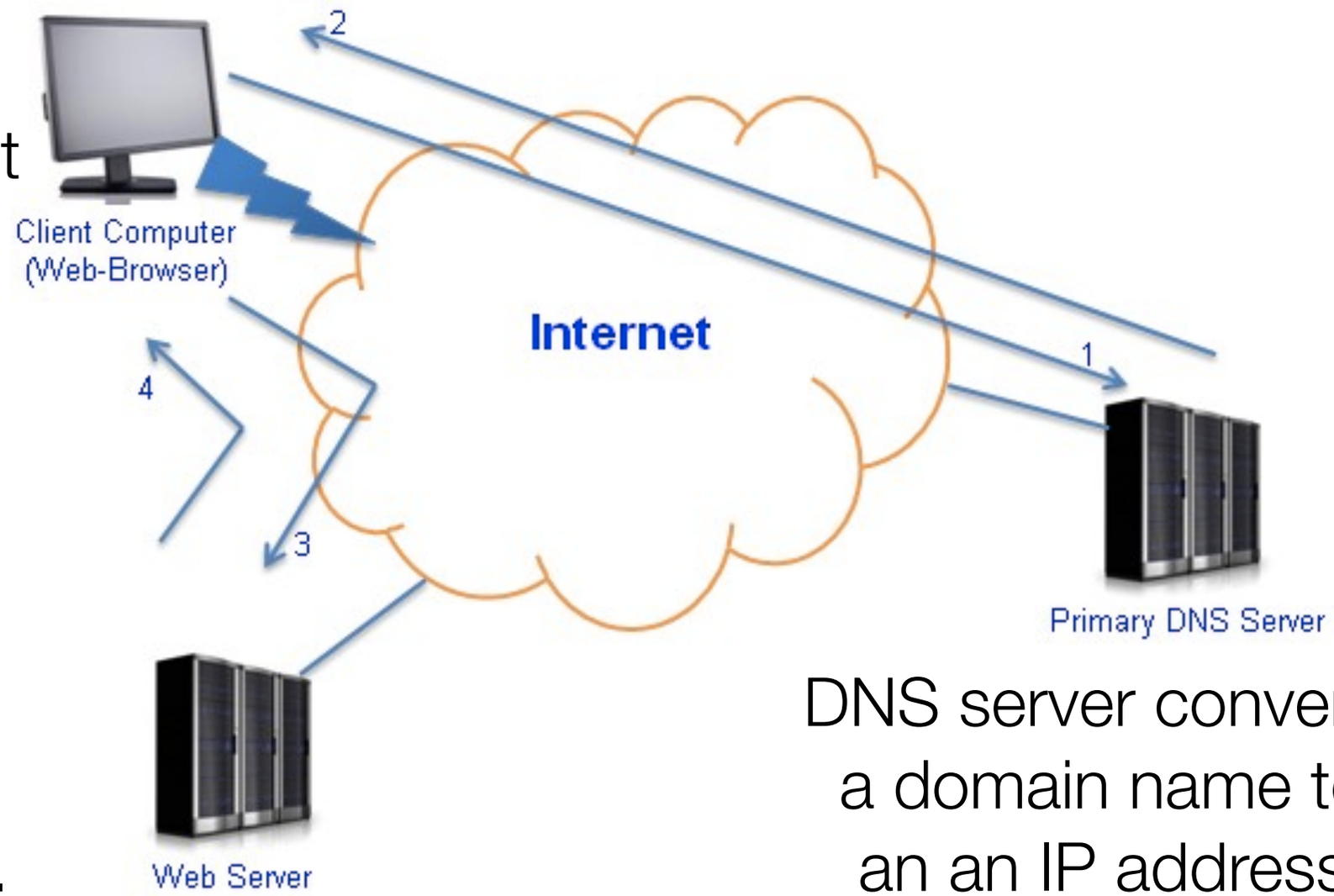
JDBC Program Steps

1. Establish / open a connection to database server
   I. Import JDBC library (java.sql.*)
   II. Load appropriate JDBC driver
   III. Create a connection object
2. Submit database commands
   I. Create a statement object
   II. Submit SQL statement
   III. Process query results
3. Close connection

PHP: Web Programming

World Wide Web (WWW)

Web browser is a program that receives web content and displays them



Web server serves content, whether it is webpages, images, movies, etc.

DNS server converts a domain name to an an IP address

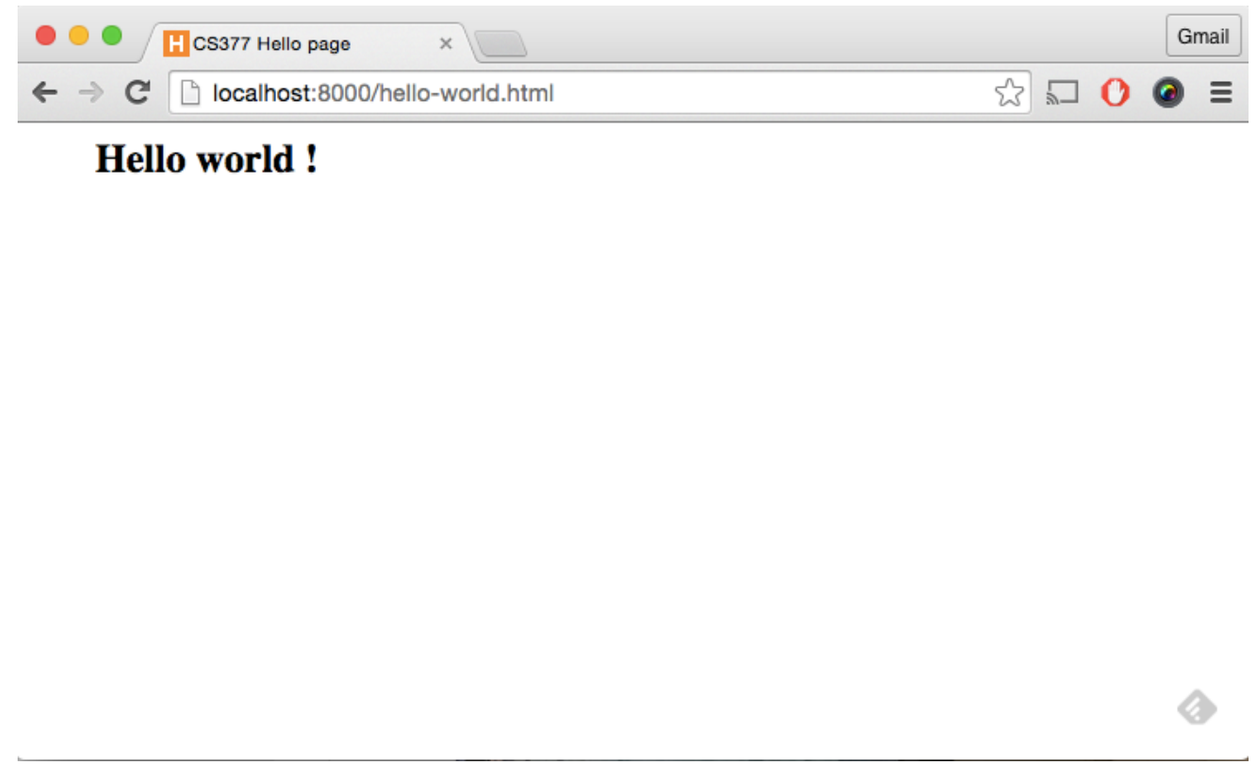
http://www.vebbsite.com/admin/photos/world-wide-web.jpg

Website

- Nothing more than a collection of computer files (webpages)
- Files are written in a special language called HTML (Hyper Text Markup Language)
  - Tags are used to specify how an items are displayed
  - Content can be static or dynamically generated

Example: Hello World HTML

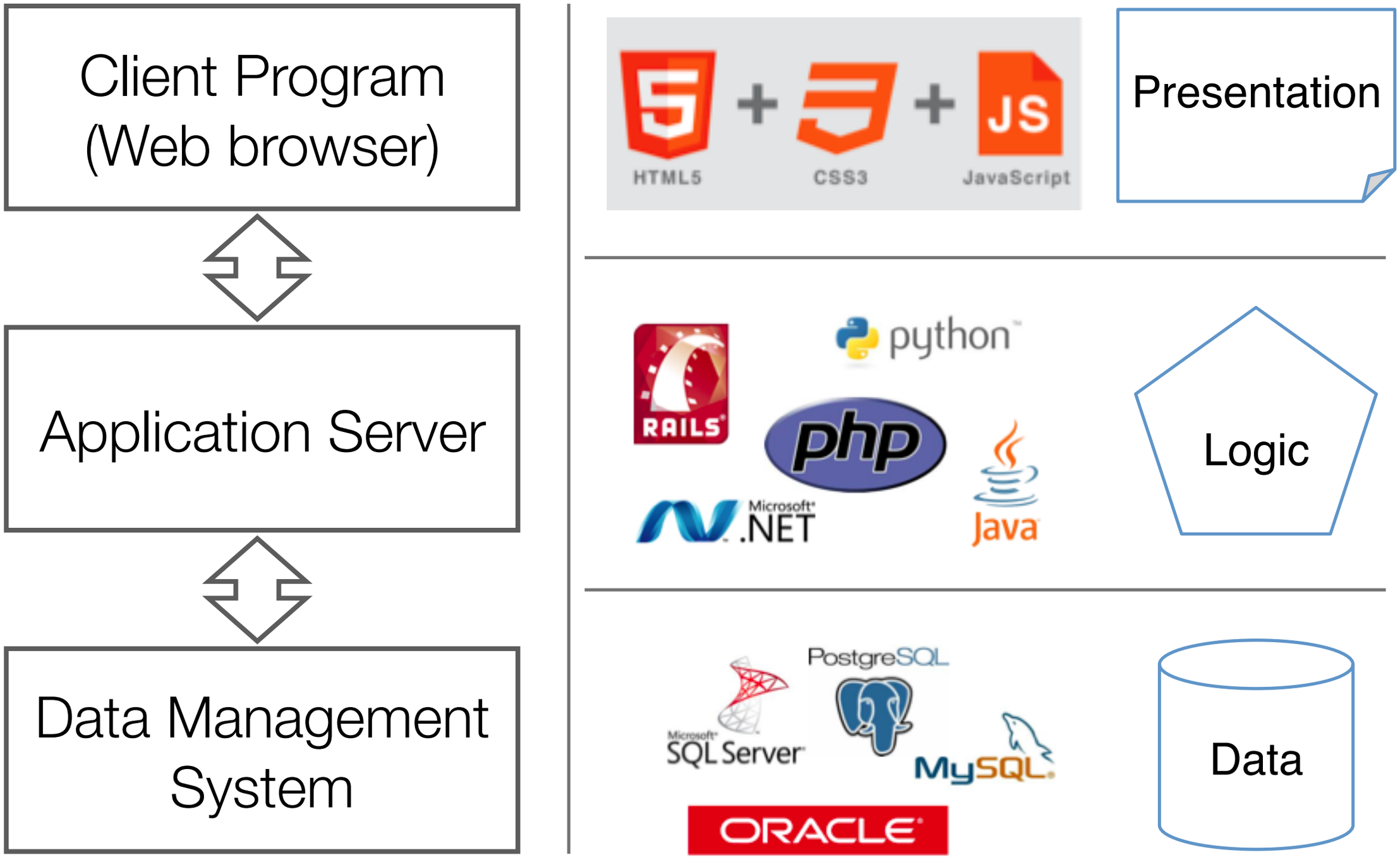
```
<html>
  <head>
     <title>
        CS377 Hello page
     </title>
  </head>
  <body>
     <UL>
        <H2>
           Hello world !
        </H2>
     </UL>
   </body>
</html>
```



Dynamic Content

- Static webpages are considered passive content as they don't perform any operations
  - Example: My personal webpage
- Webserver can execute programs that produce an HTML file (webpages)
  - Active content are web pages that are created dynamically
  - Common example is online ordering or shopping

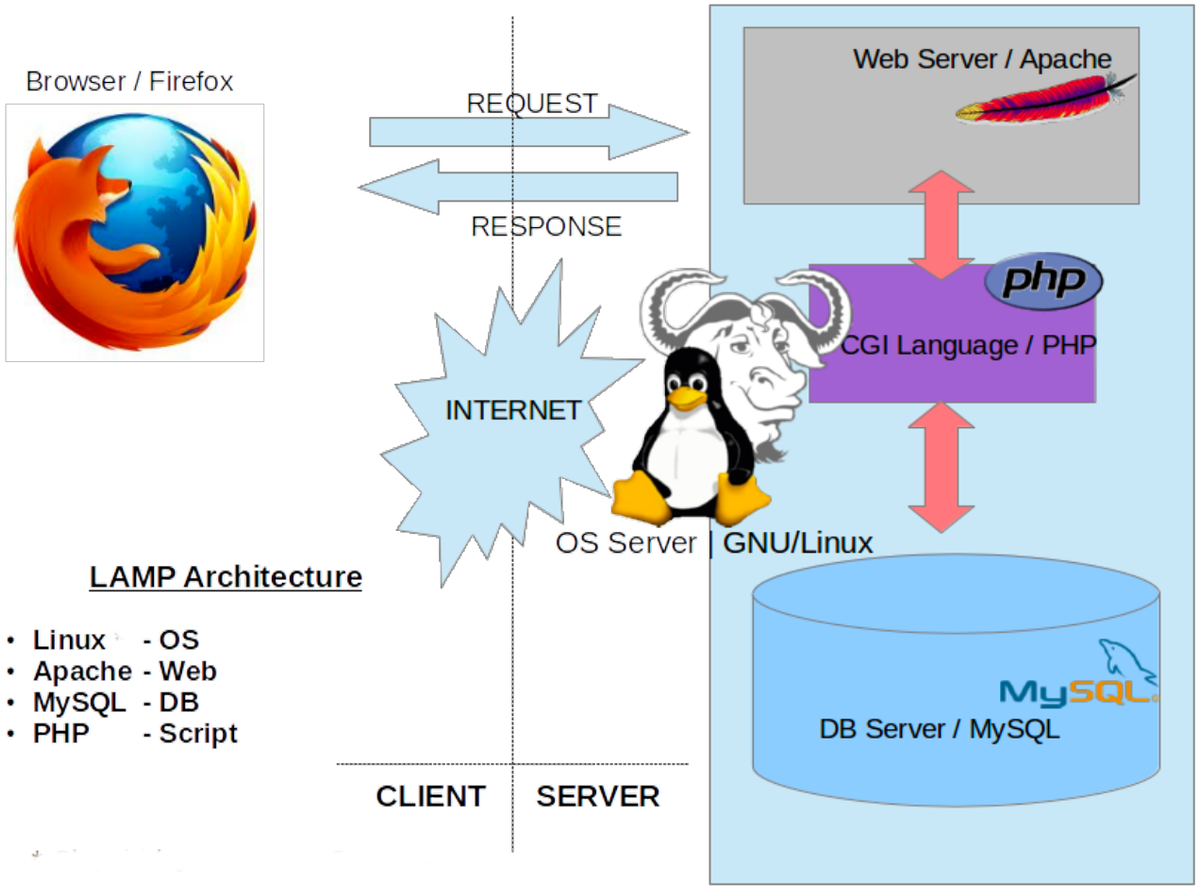
Three-Tier Web Architecture



http://apigee.com/about/blog/technology/new-three-tier-architecture-html5-proxy-apis

LAMP

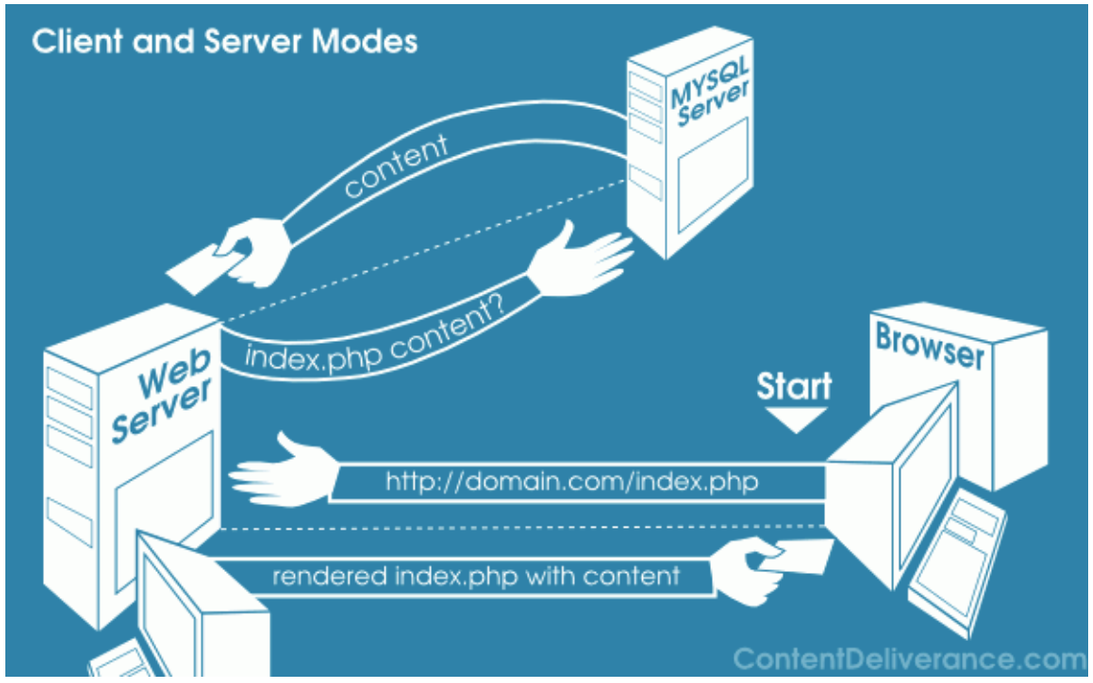
- Typical web service solution stack
- Original phrase was Linux, Apache, MySQL, and Perl (Perl —> PHP)
- Components are largely interchangeable



https://en.wikipedia.org/wiki/LAMP_(software_bundle)

PHP: PHP Hypertext Processor

- Open source, server-side scripting language for producing dynamic web pages
- Allows access to a database and executions of calculations and logic
- PHP web server interprets PHP code and dynamically constructs web page



http://contentdeliverance.com/cms-school/wp-content/uploads/2011/05/client-server-diagram-mysql.png

PHP: Strengths

- Ease of learning and use
- Open source and stable
- Speed - relatively fast
- Powerful library support & interface to many different database systems
- Availability of support

PHP: Disadvantages

- Security - many exploits of weaknesses of PHP
- Not suitable for large scales - not very modular
- Ugly and unpredictable type system (type casting and other conversion mechanism)
- Culture of messiness
- Poor debugging facilities

Exploring PHP: Setup

- PHP is currently installed on cs377db.mathcs.emory.edu
- Created a common user (cs377_s17) on the machine with the password to be posted on piazza
  - Remote login to the server:
    ssh -X cs377_s17@cs377db.mathcs.emory.edu
  - Scripts should be placed inside the public_html directory:
    cd ~/public_html
  - You can access the PHP scripts via a brower:
    http://cs377db.mathcs.emory.edu/~cs377_s17/filename

PHP Program Structure

- PHP code can be embedded in HTML code
- PHP program consists of
  - Main program
    - Statements enclosed by the PHP tags
  - Function definitions

PHP Interpreter

- Echo everything that is not enclosed inside a PHP tag
- Text that is enclosed inside a PHP tag are considered to be PHP
- Syntax:

```
<? php
   … one or more PHP statements …
?>
```

Example: Hello World in PHP (helloworld.php)

start tag to denote PHP statement

comment

```
<?php
// prints hello world
echo "Hello World!";
?>
```

end tag to close PHP statement

http://cs377db.mathcs.emory.edu/helloworld.php

Running PHP Programs

- Stand-alone (good for debugging)
  - UNIX-prompt>> php <script-name>
- Web browser
  - PHP script inside ~/public_html on cs377db server
  - Point your favorite web browser to: http://cs377db.mathcs.emory.edu/~<userid>/<script-name>

Example: PHP with HTML (luckyNum.php)

```
<html>
<head>
<title> PHP Test </title>
</head>
<body>

<UL>
Welcome stranger, here is your lucky number:
<?php print rand(1, 1000); ?>
</UL>

</body>
</html>
```

http://cs377db.mathcs.emory.edu/luckyNum.php

PHP Variables

- Syntax: $variableName
- Variables start with letter or underscore
- Variable name is case-sensitive
- Implicitly defined — automatically defined when you use the variable for the first time in a program

Example: PHP Variables (var.php)

```
<?php
  $a = 1;
  $A = 2;
  print("a = " . $a . "\n");    # . is string concatenation
  print("A = " . $A . "\n");   # Var name is case sensitive !
  print("b = " . $b . "\n");   # Warning, not fatal !
?>
```

http://cs377db6.mathcs.emory.edu/var.php

PHP Variable Types

- Support 8 primitive types
  - 4 scalar types: boolean, integer, float, string
  - 2 compound types: array, object (C's struct)
  - 2 special types: resource (special variable holding a reference to an external resource), NULL
- Dynamic typing — type of variable is determined by the type of value that was stored in the most recent assignment statement

Example: Dynamic Typing (dynatype.php)

```
<?php
  $a = 12;
  print ("a = " . $a . " Type of a = " . gettype($a) . "\n");
  $a = 12.0;
  print ("a = " . $a . " Type of a = " . gettype($a) . "\n");
  $a = "12";
  print ("a = " . $a . " Type of a = " . gettype($a) . "\n");
  $a = true;
  print ("a = " . $a . " Type of a = " . gettype($a) . "\n");
?>
```

http://cs377db.mathcs.emory.edu/dynatype.php

PHP Operators

- Operators are similar to Java
  - Arithmetic operators: +, -, *, / , %, **
  - Logical operators: and, or, xor, !
  - Comparison: ==, !=, <, <=, >, <=
- Example:

```
<?php
  $b = 3 * 3 % 5;
  $b = $a++ + 23;
  $a = ++$b - 23;
?>
```

PHP Statements

- If statement & elsif

```
if (expr1)
{
.. statements …
}
elseif (expr2)
{
.. statements 2…
}
[else
{ .. more statements …
}]
```

- While statement

```
while (expr)
{
.. statements …
}
```

- For statement

```
for (expr1; expr2; expr3)
{
… statements …
}
```

break and continue work similarly

PHP Functions

- Similar to functions in other programming languages
- Can appear anywhere in the main program
- Need not be defined before it is used
- Syntax:
  function <funcName> ($<param1>, $<param2>, …)
  {
  … one or more statements …
  }

Example: PHP Function

```
<?php
   $a = square(4);
   print("Square of 4 = " . $a . "\n");
   # Function definition
   function  square( $x )
   {
      $r = $x * $x ;
      return( $r );
   }
?>
```

prints out the square of 4 = 16

http://cs377db.mathcs.emory.edu/squareFunc.php

PHP Variable Scope

2 scopes in PHP

- Global (program) scope — variable created in the main program has global scope and can be accessible from everywhere in the main program
  - Access variable inside a function by declaring it a global variable with the keyword global
- Function scope — variable created in the function has a function scope and will be different than a variable with the same name in global scope

Example: PHP Variable Scope

```
<?php
 $a = 1;                    # Global a
 print("Main: a = " . $a . "\n");
 f($a);
 print("Main: a = " . $a . "\n");

 function  f( )
 {
    global $a;      # ******** a will now access a global variable
    print("f before: a = " . $a . "\n");  # Global scope a
    $a = 4444;
    print("f before: a = " . $a . "\n");  # Global scope a
 }
 print("Main: a = " . $a . "\n");
?>
```

http://cs377db.mathcs.emory.edu/varscope.php

Beware! PHP Weirdness

- Things in PHP that are unlike Java/C
  - String — different ways to quote a string
    - Variables can appear inside a string
    - Variables are evaluated differently depending on how the string is quoted
  - Array — use of associative arrays (key, value pairs)

Strings: Single-quote

- Always treated verbatim and no evaluation takes place
- Example:

```
$x = 1;
print ‘This is a single-quoted string. This is $x\n’;
```

Output:
This is a single-quote string. This is $x\n

Strings: Double-quote

- Perform evaluation of variables to construct final strings
- Use escape character "\" before $ to prevent evaluation
- Example:

```
$x = 1;
print "This is a double-quoted string. This is $x\n";
print "This is an escaped double-quoted string. This is \
$x\n";
```

Output:
This is a double-quote string. This is 1
This is an escaped double-quote string. This is $x

Strings: “Here” Documents

- Inline multi-line text that is evaluated

- Example:

```
$x = 12345;
print <<<MARKER
    Here document text.. type away... This is $x
    Another line. Just keep going - the string will not stop
    until there is a line with MARKER at the START of the line\n
MARKER;
```

Output:

```
Here document text.. type away... This is 12345
Another line. Just keep going - the string will not stop
until there is a line with MARKER at the START of the line
```

Example: Strings

```
<?php
  $x = "Hello World !";
  print 'Single-quoted string. This is $x';
  print"\n";
  print "Double-quoted string. This is $x";
  print"\n";
  print <<<MARKER
    Here document text.. type away... This is $x
    Another line. Just keep going - until a line with MARKER is found
MARKER;
  print"\n";
  print <<<MARKER2
    Here document text.. type away... This is \$x
    Another line. Just keep going - until a line with MARKER is found
MARKER2;
  print "\n";
?>
```

http://cs377db.mathcs.emory.edu/stringEx.php

Arrays: Ordered Map

- Associates keys with values
- General:

```
$varName = array (
                    key1 => value1 ,
                    key2 => value2 ,
                    ...
                  );
```

Arrays: Ordered Map

- Integer indices

```
$varName = array (
                    value1 ,
                    value2 ,
                    ...
                  );
```

- “Traditional” way

```
$arrName[ index ] = value;
```

Array Functions

- Count the number of elements in an array:
  count($<array variable>)

- Accessing elements in array
  foreach ($<array variable> as $KEY_VAR =>
  $VALUE_VAR)
  {
  $KEY_VAR = key of the current array element
  $VALUE_VAR = value of current array element
  }

Example: Associate Arrays

- array01.php — different syntax for defining an array
- array02.php — counting the number of elements in an array
- array03.php —accessing the array using the special foreach structure
- array04.php — an example of a true associate array where the keys are not integers

PHP Program: Step 1

1. Establish connection to database

   I. Pick extension / library module to connect to database system

   II. Connect to database

PHP: Access to MySQL

- ext/mysql (MySQL extension which is not recommended and deprecated now)
- **ext/mysqli (MySQL improved extension)**
- PDO (PHP Data objects - pure object oriented programming)

PHP: Connect to Server

- Command: **mysqli_connect(host, user, passwd [, dname [, port [, socket]]])**

- Example:
  $conn = mysqli_connect("cs377db.mathcs.emory.edu", "cs377", “cs377_s17");

```
// check connection
if (mysqli_connect_errno())
{
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
```

PHP: Connect to Database

- Specify the database in the connection:
  $conn = mysqli_connect(“cs377db.mathcs.emory.edu","cs377","cs377_s17", "companyDB");

- Use **mysqli_select_db()** function:

```
if ( ! mysqli_select_db ($conn, "companyDB") )
{
    printf("Error: %s\n", mysqli_error($conn) );
    exit(1);
}
```

PHP Program: Step 2

2. Submit database commands

   I. Send statement

   II. Process query results

PHP: Submit SQL Query

- Execute a query using mysqli_query()

```
if ( ( $result = mysqli_query( $conn, "SQL-command" ) ) == 0 )
{
    printf("Error: %s\n", mysqli_error($conn));
    exit(1);
}
```

- Returns 0 if there was an error, otherwise the result

Example: Submit SQL Query

- PHP code:

```
$conn = mysqli_connect(“cs377db.mathcs.emory.edu”,”cs377",
“cs377_s17", "companyDB");
 if (mysqli_connect_errno())
 {
     ...
}
$query = 'select fname, lname, salary from employee';
if ( ! ( $result = mysqli_query($conn, $query)) )
{
    printf("Error: %s\n", mysqli_error($conn));
    exit(1);
}
```

PHP: Obtain SQL Results

Many different functions to retrieve result tuples

- **mysqli_fetch_all($result)** : fetches all result rows and returns the result set as an associative array
- **mysqli_fetch_array($result)** : returns the current (fetched) row as an array
- **mysqli_fetch_assoc($result)** : returns the current (fetched) row as an associative array or NULL if there is no more rows

PHP: Obtain SQL Results

- Focus on **mysqli_fetch_assoc($result)**
- Returns associative array that contains (key, value) pairs with the attribute name and value
- Example:
-

$key	$value
SSN	111-11-111
Fname	John
Lname	Smith
…	…

Example: Print SQL Results

Print attribute names and attribute values from $result array

```
while ( $row = mysqli_fetch_assoc( $result ) )
{
    foreach ($row as $key => $value)
    {
    print ($key . " = " . $value . "\n");
    }
    print("================");
}
```

Example program: employee0.php

PHP Program: Step 3

3. Close connection

I. De-allocate and free resources

II. Close connection

Step 4: Free Resources & Disconnect

- De-allocate and free resources using mysqli_free_result()
  - Syntax: **mysqli_free_result(<result variable>);**
- Disconnect our connection with MySQL server using mysqli_close()
  - Syntax: **mysqli_close(<connection variable>);**

Example: Stand-Alone PHP program

- Print all the employees in the company database in a "tabular" format
- Print the attribute names only once
- Print the tuples
- To RUN: PHP emp-table.php

PHP via Web Browser

- Extremely easy to execute a program with a web browser
- Add some HTML header and trailer tags to the PHP script
- Put the MySQL PHP script in the special directory (will depend on what web server architecture you use)
- Load the PHP script in the web browser

Example: PHP Program via HTML

- HTML is ideally suited for formatting outputs
- Same example as before where you want to display all the employees in the company database in a “tabular” format — utilize HTML table format
  - <TABLE> tag to denote start of table
  - <TR> denotes a new row
  - <TD> denotes one data item in the row
- Example: emp-html-table.php

http://cs377db.mathcs.emory.edu/emp-html-table.php

PHP: HTML FORM

- HTML FORM tag allows a webpage to obtain input field(s) from the user
- <input type="<type>" name="<varname>"> element
  - Each input field must have a type
  - Each input field must have a name attribute
  - Optional: specify the size of the input field
    <input type="<type>" name="<varname>", size=40>

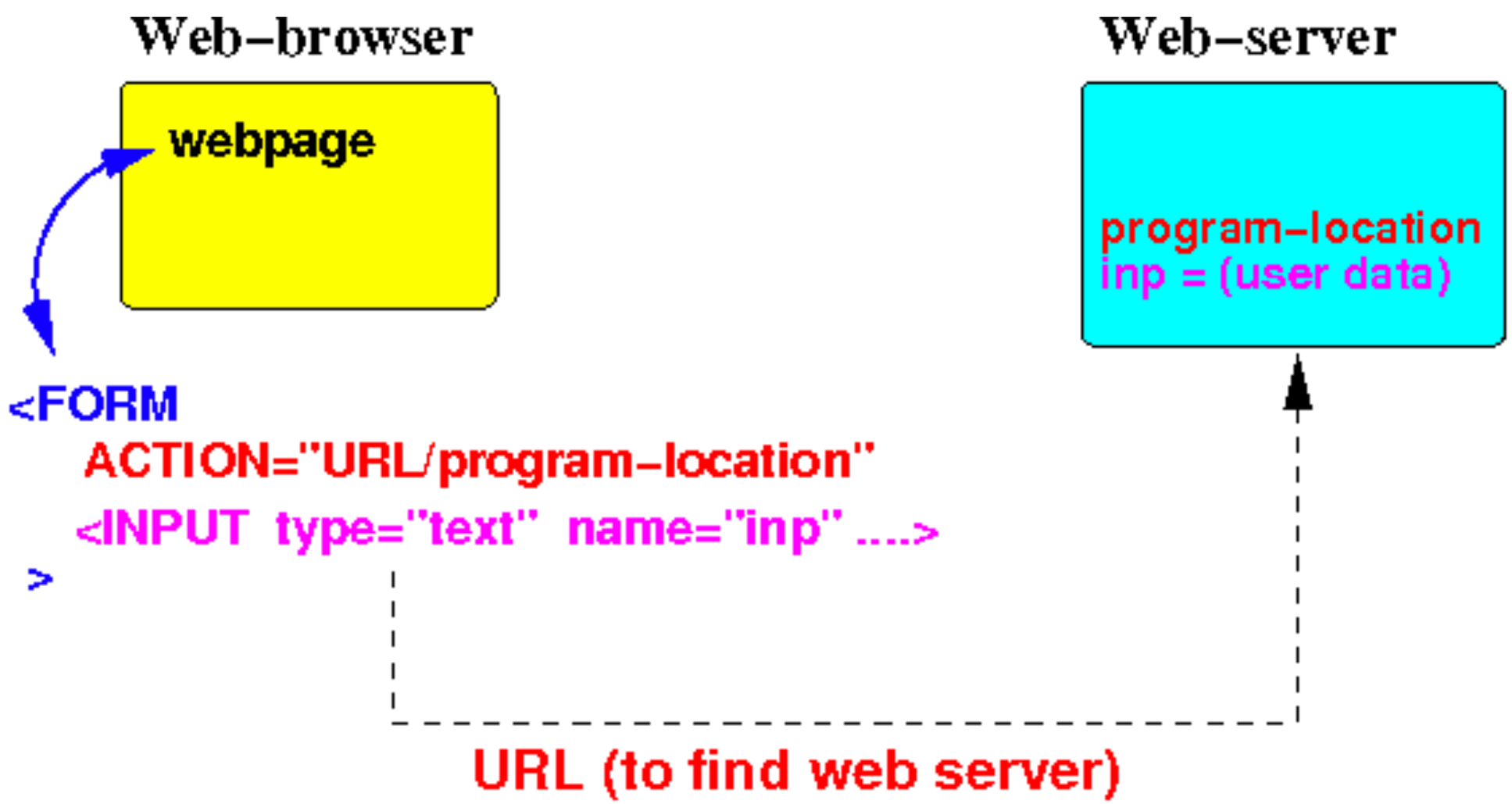
PHP: HTML FORM

- Common types
  - <input type = “text”> defines a one-line input field for text input
  - <input type = “radio”> defines a radio button (limit to 1 choice)
  - <input type = “submit”> defines button to submit a form to form-handler

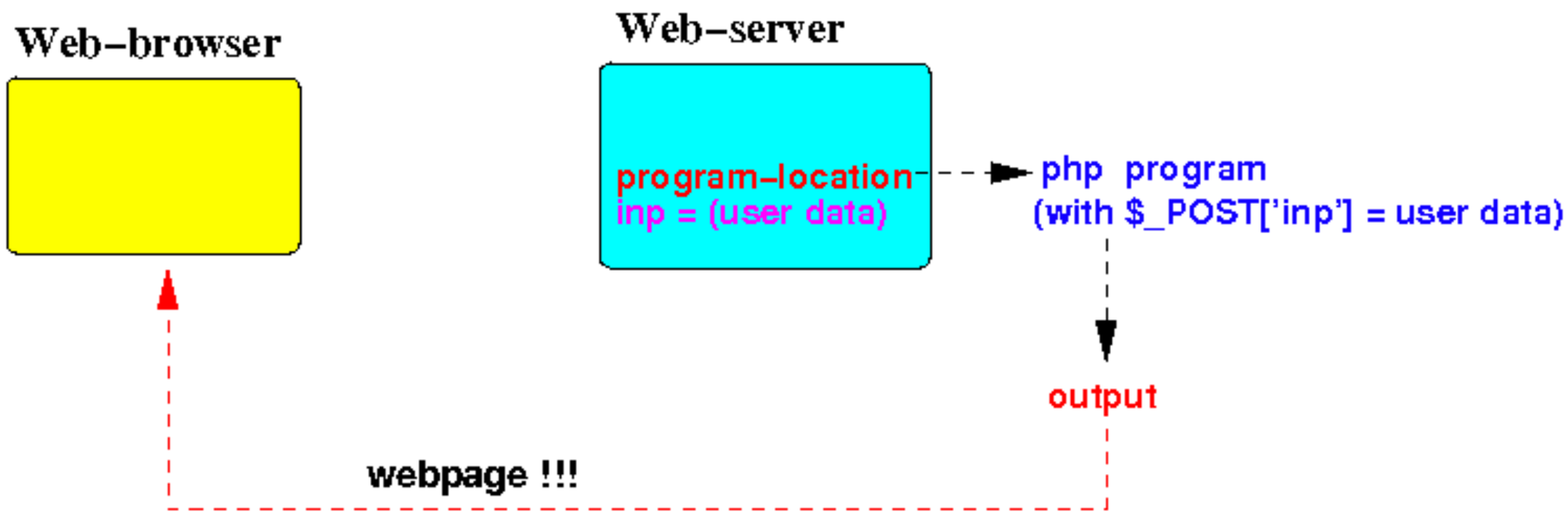
PHP: HTML Form

- <form action="filename.php" method="{get | post}">
  - action defines the address or URL where to submit the form
  - method specifies the HTTP method to be used when submitting the forms
    - GET (default) is generally used for short amounts of data and without sensitive information (data is encoded after a ? symbol)
    - POST offers better security because submitted data is not visible in the page address

PHP: Receiving Data using POST



PHP: Receiving Data using POST



PHP: Receiving Data using POST

- PHP interpreter receives form tagged data from the HTML form
  - Data comes in an associative array named $_POST[]
  - Initializes the element $_POST[‘input-var-name’] with the value entered in the corresponding input field in the form tag

Example: PHP Script for form1 (echo.php)

```
<html>
<head>
<title> Form1 test </title>
</head>
<body>
<HR>
<B>
<?php
# ------------------------------------------------------------------
# PHP program: echo the data send in the "inp" field by the form
# ------------------------------------------------------------------
  $data = $_POST['inp'] ;
  print("Post Data is $data \n");
?>
</B>
<HR>
</body>
</html>
```

Example: HTML FORM

```
<html>
<head>
   <title> HTML Form 1</title>
</head>
<body>
   <HR>
   <B> Form: </B>
   <HR><P>
   <FORM ACTION="http://cs377db.mathcs.emory.edu/echo.php"
METHOD="POST">
      <p>Enter input: <input type="text" name="inp" size=40></p>
      <p><input type="submit" value="Press to send"></p>
   </FORM>
</body>
</html>
```

http://cs377db.mathcs.emory.edu/formEx.html

Example: PHP Client for companyDB

- Web form to submit a query: http://cs377db.mathcs.emory.edu/companyDB-queryform.html
- PHP script to handle the query: http://cs377db.mathcs.emory.edu/companydb-query.php

SQL Application Programming: Recap

- General application interaction sequence
- JDBC
- PHP
  - Connecting to MySQL
  - HTML web forms
  - For more information about PHP: http://us2.php.net/manual/en/index.php