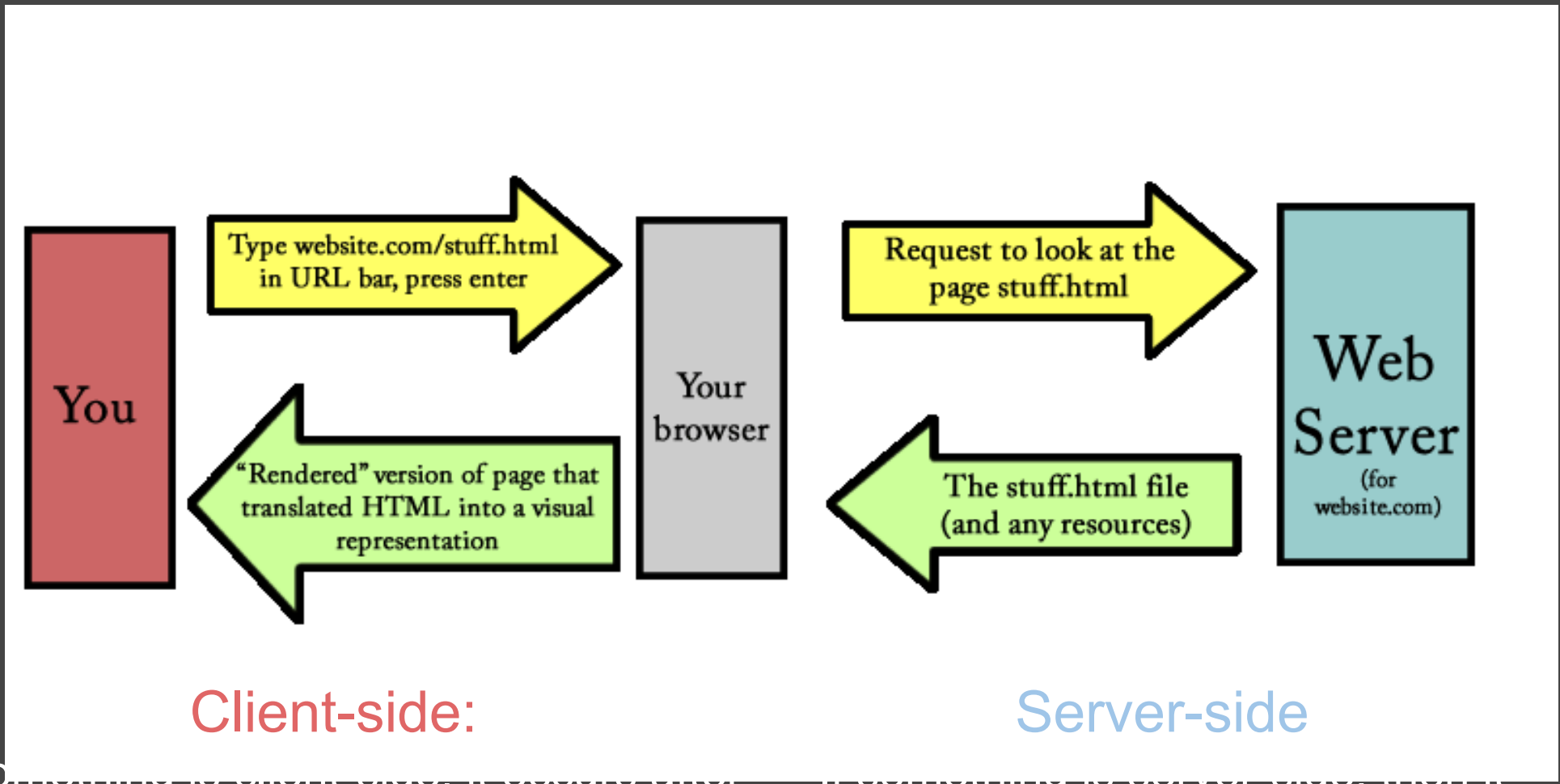
Intro to Dynamic Web Development

ACM Webmonkeys @ UIUC, 2010

Static vs Dynamic

- "Static" is an adjective meaning "does not change".
  - In the context of web programming, it can be used several different ways, but the most common is simply to refer to a webpage which is written purely in HTML/CSS -- a "static" page.
  - More generally, a static page does not change itself before being sent from the server to the browser.
- "Dynamic" means the opposite.
  - In web programming, a dynamic page is one which uses some means to change itself before being sent to the browser.
- We've already learned how to write static pages. Now, we'll go over what "dynamic" means in practice and how to create dynamic websites.

Review: How websites work



Client-side:

If something is client-side, it occurs after the page is sent to the web browser. For example, HTML rendering is client-side.

Server-side

If something is server-side, then it occurs *before* the page is sent to the web browser. For example, interpreting PHP is server-side.

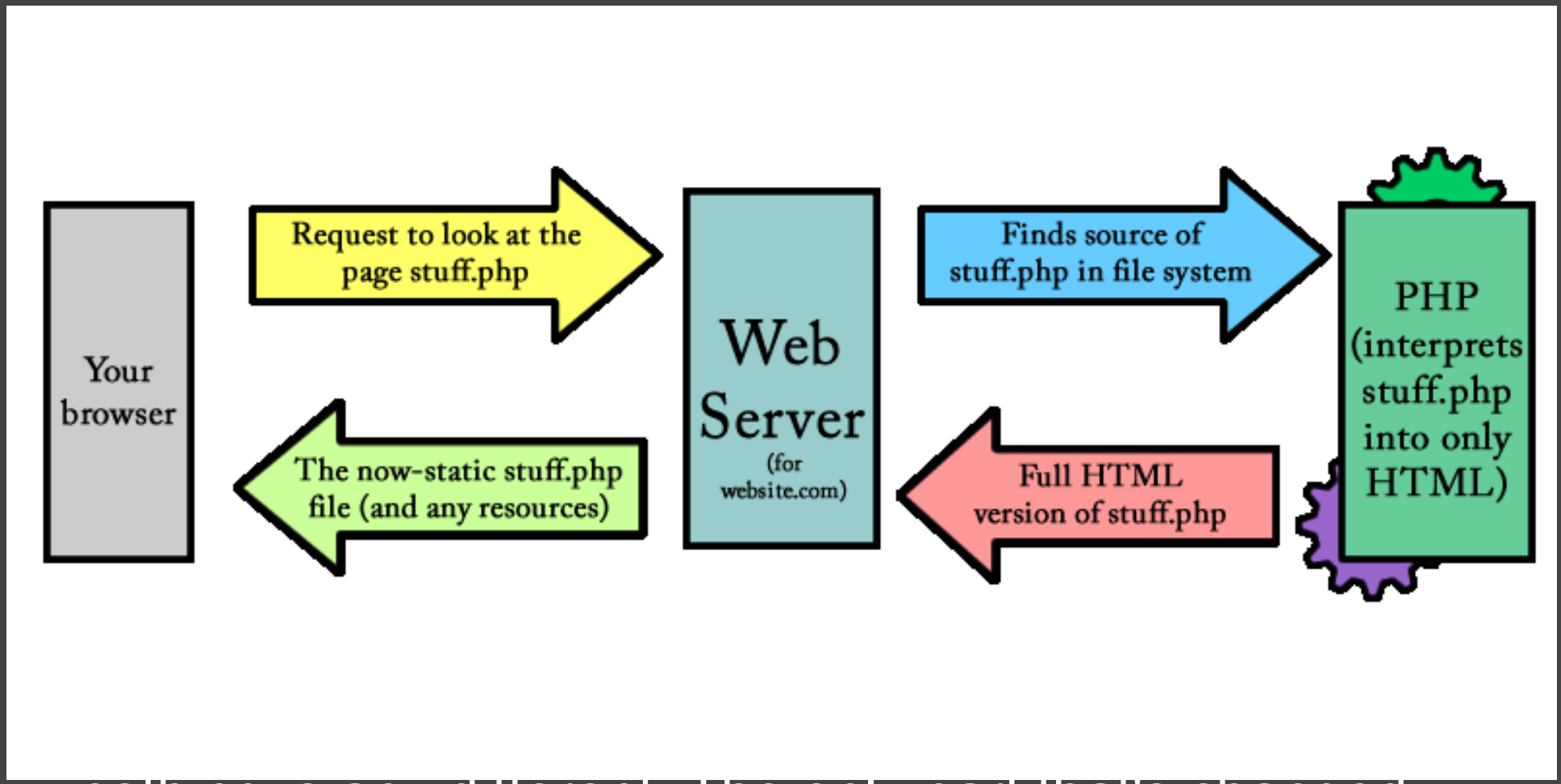
How do we add dynamic content?

- We can only send a file to the web browser which it can "understand"
  - Initially this meant only HTML, CSS, and images
  - Soon, browsers were built to understand client-side scripting languages, which allow websites to change themselves after being downloaded
    - Javascript is the most common client-side language
- Technically, we *could* write a dynamic site using these client-side languages.
  - However, **anything client-side can be viewed, changed, and tainted by the user!**
  - Doing so would expose our site logic and represent a major security hole.

So, we focus on the server side...

- Given that we can only safely send a static page to the client (Javascript for fancy effects notwithstanding), we make our site dynamic by **dynamically generating the static page**.
- So, instead of having our website directly fetch the .html file from the server's file system and send it straight to the client, we first have the server **pre-process** the file.
- The most intuitive way to do this on top of HTML is to have a special tag that gets pre-processed before the page is sent to the user's browser.
  - This is the key idea behind PHP and ASP.
  - A more complex model is needed for full websites, however -- more on that next week!

Revisiting our diagram



...

nothing is any different. The only part that's changed is on the web server!

So all we need is:

- A web server capable of:
  - Understanding that we want some pages to be preprocessed
  - Preprocessing such pages **before** sending them back to the user
- These requirements are pretty minimal.
  - We also want web servers to be able to manage user sessions, enforce security, etc., etc... but we'll worry about that later.
- If you want to deploy a website using a particular dynamic technology, make sure the production server you plan on using supports it.
  - PHP is almost universally supported. Django, not so much.

Setting up a web server on your computer

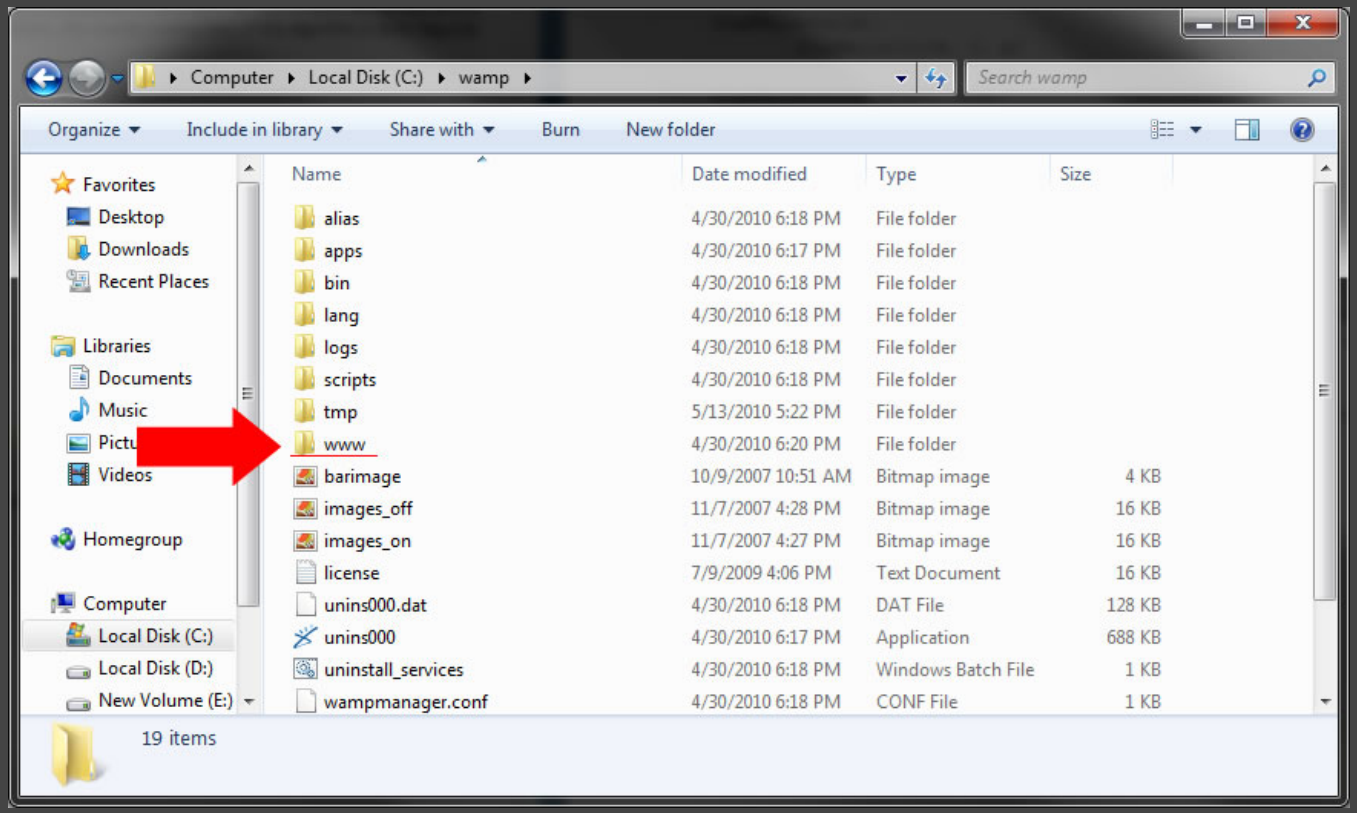
- If we want to develop a dynamic website, we really need a way to test the dynamic functionality on our local computer.
- However, a full web server consists of several layers of applications (sometimes called the software stack)... it's a pain to set each layer up manually.
- The quick and easy way to do this is to install an all-in-one package
  - Windows: WAMP (www.wampserver.com/en/)
  - Linux: LAMP (check your repository for a good one)
    - Ubuntu: `sudo tasksel install lamp-server`
  - Mac: MAMP (www.mamp.info)
- LAMP = Linux, Apache, MySQL, PHP

Installing WAMP

- Installing WAMP (or the respective XAMP stack for your platform) is usually pretty easy -- just click through the installer.
- Make note of where the system installs to
  - By default, C:\wamp\
  - On Linux, usually /var/www/
- As a note, make sure your webserver is NOT listening for connections from outside 127.0.0.1 (a special IP address meaning your computer and only your computer).
  - By default, any XAMP install should be configured this way.
- If you need help, stop by afterward or come to my office hours (Wednesdays @ 6pm in the ACM office) -- or just Google your question.

It's installed! Now what?

- Remember last week, when we discussed what a **web root** was for web servers?
- You need to find the web root for your brand-new, local web server.
- Probably at <installation path>/www (e.g., C:\wamp\www)



Making sure it worked

- To make sure PHP and your server are properly configured, create a file named **test.php** in the web root.
- Type in the following three lines:

```
<?php
phpinfo();

?>
```

- Open your web browser and go to:
  - http://localhost/test.php
- 'localhost' is a shortcut for 127.0.0.1, your local computer

If it worked, you should see this:

System	Windows NT TSATHOGGUA-PC 6.1 build 7600 ((null)) i586
Build Date	Jun 29 2009 21:23:30
Compiler	MSVC6 (Visual C++ 6.0)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--disable-isapi" "--enable-debug-pack" "--with-pdo-oci=D:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8=D:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8-11g=D:\php-sdk\oracle\instantclient11\sdk,shared" "--with-enchant=shared"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded	C:\wamp2\bin\apache\Apache2.2.11\bin\php.ini

Now that we know PHP works, we can get started.

What is PHP?

- Remember how we wanted a script that could pre-process HTML pages and interpret server-side code?
- PHP stands for PHP Hypertext Preprocessor
- It's a relatively-simple scripting language that you can use for dynamic web programming
  - Has some problems scaling up, due to security, language features, etc.
    - Can be mitigated through good design
  - Good choice for simple, cross-platform sites
- Uses a very-simplified, C-like syntax
  - Expanded in version 4 to add OOP support

PHP Basics: Variables

- In PHP, variables are prefixed with dollar signs ($) to indicate that they're variables.
  - e.g. $varname, $time, $first_name, $date
- Variables are dynamically typed:
  - PHP does not care what type is stored in any given variable
  - It is up to you to make sure your code makes sense, or you'll get weird errors!
- As in most C-like languages, you use the = operator to assign variables.

```
$first_name = "string";
$var = "4";
$var = 4;       // works fine, overwriting the old value.
```

More on variables: Strings

- A string is a variable type which holds a snippet of text.
- To define a string in PHP, use quotes (" or ') around the text you want to make a string.
  - Single-quotes will not change the string as written. If you're concerned about speed, these are faster.
  - Double-quotes will cause PHP to **interpolate** the string for variable references:
    - $var = 4; echo("The value of var is $var"); will be turned into "The value of var is 4" when run!

```
$str1 = 'This is a string.';
$metastr = "$str1 is a string";
$str1 = 'Changed?'
echo($metastr); // This is a string. is a string
// Caveat: interpolation is not repeated unless you ask PHP nicely.
```

PHP Basics: Statements

- Scripts in PHP are comprised by a series of **statements**. Each statement either defines something (like a class or a control structure), calls a function, or assigns a variable.
- Remember to end each statement with a semi-colon, unless the statement is a control structure or a class definition.

```
initialize_some_stuff(); // function call
$day_created = get_day_created();  //assgnmt.
if ($day_created > 3) {
  echo("More than three.");
} else {
  echo("Less than or equal to 3.");
}
```

PHP Basics: Control Structures

- A **control structure** somehow controls the execution of code. The two basic examples are **if-then** statements and **while** loops.
- If-then checks if the conditional evaluates to a non-false value, and executes the first block of code if it does.
  - In PHP, the values 0, "", and **false** evaluate to false. Everything else is evaluated as true in a conditional.
- A block of code is a list of statements surrounded by brackets { }
  - Think of a block as a single statement with multiple steps
- A while loop does the same evaluation, but continues to execute the block until the conditional evaluates as false.

Examples of If-Then and While

```
if ($something = "fruit") {
    echo("Fruit!");
    echo("This is in the same block.");
} else {
    echo("Not fruit.");
}

while ($something != "fruit") {
    echo("Still not fruit!");
    $something = get_new_something();
}
```

Back to variables: Arrays

- An array is a variable that stores a list of other variables.
  - $arr = array("red", "green", "blue");
- To add to an array, do: **$arr[] = "new color";**
- To reference a particular item, use **$arr[0], $arr[1], etc.**
- Terminology:
  - "key" refers to the unique identifier you use to access an array element.
  - "value" refers to the actual element.
- Arrays can also have non-integer keys -- this makes them more intuitive to access.
  - $friends = array("best" => "Joe", "worst" => "Bill");
    - Note the =>, which indicates a key-to-value pair.
  - $my_best_friend = $friends["best"];

For-loops and For-each loops

- A **for-loop**, at its simplest, executes a block of code a certain number of times. The current iteration number can be used inside the loop if needed.

```
for ($i = 0; $i < 10; $i++) { /* goes from 0 to 9 */ }
```

- A **for-each** loop iterates over the items in an array.
  - Necessary for associative (non-integer-keyed) arrays, as we have no other way of telling what keys we have in a given array!

```
for ($current_key => $current_value) {
  // do stuff with each key and value
}
```

Defining functions

- A function is a block of statements that has arguments passed into it from another part of the program, and may return a value to the calling statement.
- There are hundreds of built-in PHP functions, like echo(), file_get_contents(), mysql_connect(), etc.
  - Check php.net for arguments to these functions
- To define your own, do:

```
function myCoolFunction($arg1, $arg2) {
    echo "doing stuff";
    return $arg1 + ($arg2 * 4);
}
```

Using PHP

- You can look up much more complete/helpful information on PHP practically anywhere on the web (php.net being your best bet).
- Let's move on to how we can actually use this stuff to make our webpages less boring.

Scenario: Custom Links

- Your client wants to send out a mass email campaign to each one of his customers. When they click on the link to his site inside it, he wants them to be greeted with their name.
- Since we already have the name of each customer, we can do this by changing the link inside *each* email to be custom.
  - Do we really want to have index_robert.php, index_ted.php, etc., etc....?
  - No.
- Instead, we can use **URL parameters** to add information to each URL without having to create a brand-new page.

URL Parameters

- You've probably seen links on the web that look like:
  - http://www.website.com/foo.php?lang=en&id=4
- By default, Apache is configured to ignore everything after a ? in a URL -- it only uses foo.php to find the file, and passes the rest of the URL to PHP to use as it likes.
  - This extra data is often called the **querystring**.
- PHP can use what follows to get additional information.
  - It interprets the data as a list of parameters, in the form **key=value**, separated by ampersands (&).
  - So, above, it sees two parameters:
    - lang, with a value of 'en'
    - id, with a value of 4
- We can use these to make special URLs for each customer.

Customizing our links

- If we give each client a special URL with their name passed in as a URL parameter, then when they click on it, we can get that name in PHP and display it.
  - e.g. http://www.business.com/welcome.php?name=Bob
- So, our email list has the names already. We just have to modify our mailer script to generate these URLs.
- Note: how do we deal with spaces and weird characters?
  - To properly encode these characters in URLs, use URL encoding (PHP has a function to do this).
  - A space is %20, when URL encoded.
  - e.g. welcome.php?name=Mary%20Jane

Now, the names are in the URL

- So how do we get them out of it?
- PHP has a few very special **superglobal** arrays:
  - $_GET
    - URL parameters (which we want)
  - $_POST
    - Form data (covered later)
  - $_REQUEST
    - Union of GET and POST.
  - $_SESSION
    - Session data (covered later)
- We can use the $_GET array to get our customers' names.

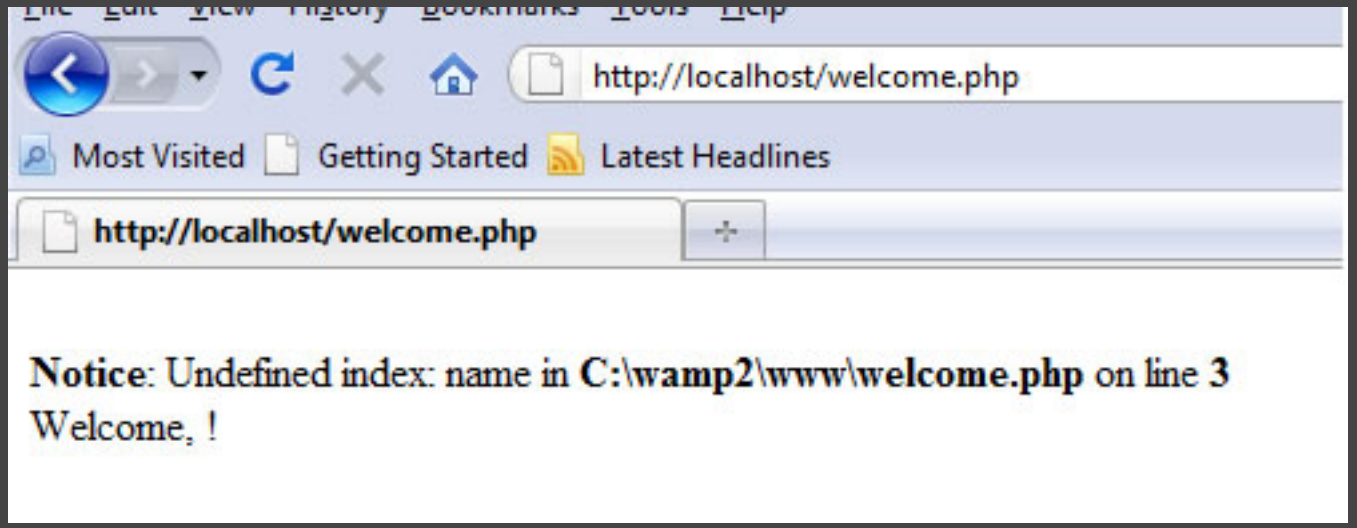
Making welcome.php

```
<?php

$customer_name = $_GET['name'];

echo "Welcome, $customer_name!";

?>
```

Now, go to http://localhost/welcome.php?name=Robert and see what comes up.

```
    Welcome, Robert!
```

Problem: what if there is no name?



- What happened?
- PHP by default does not like when you use uninitialized values. You can turn this off, but that's not a good idea.
  - Instead, make sure the index 'name' exists in $_GET.
  - Use array_key_exists($key, $array) to test

Update our code:

```
<?php
if (array_key_exists('name', $_GET)) {
  $customer_name = $_GET['name'];
  echo "Welcome, $customer_name!";
} else {
  echo "Welcome!";
}
?>
```

Now, go to http://localhost/welcome.php and:

```
Welcome!
```

Problem: what if the user is malicious?

- If we use this code as-is, we're allowing people to craft links which insert arbitrary HTML into our page.
  - The user could craft malicious JavaScript, capable of exploiting some browsers.
  - Even though only that user will see the page, what if they use bit.ly to encode the URL and then tweet it to all their friends?
    - This is one type of XSS (cross-site scripting) attack, because the attack script is usually loaded from another site.
- Rule 1 of web programming: **don't trust request data.**
- So... no problem, we can just clean the name first.
  - PHP has strip_tags($str), which (in this context only!) will remove any possibility of dangerous side effects.

New, more-secure code:

```
<?php
if (array_key_exists('name', $_GET)) {
  $customer_name = strip_tags($_GET['name']);
  echo "Welcome, $customer_name!";
} else {
  echo "Welcome!";
}
?>
```

Now, users can't try stuff like welcome.php?name=%3Cscript%20type%3D%22javascript%22%20src%3D%22www.virus.com%2Fsploit.js%22%3E

That is, welcome.php?name=<script type="javascript" src="www.virus.com/sploit.js">

However...

- This scheme still allows users to trivially change the content of a page, and is for that reason not ideal.
  - Could use substr() to trim down the name string to a reasonable length
  - Could store name values in a database with a corresponding ID, and give out URLs like: welcome.php?id=42
- But for now, let's move on to another scenario.

Scenario 2: Simple Blog

- Our client wants a simple blog site that can dynamically retrieve blog entries from a database and display them.
- For now, we don't need to worry about providing for posting or editing blog entries, just displaying them.
- Questions we need to answer:
  - What's a database?
  - How do we retrieve data from a database?

Brief intro to databases

- A **database** is a means of storing data, optimized for very large amounts of data and fast retrieval.
  - Data is organized into **tables**, which contain **columns**
    - Think of an Excel spreadsheet with multiple pages.
- Simplistic attempts at storing data in such a way use a **flat file** approach
  - Just a normal file with a list of data
  - The CSV format (comma-separated values) is a good example of typical flat file
  - Really only viable when the data does not need to be accessed, just stored.
- Databases store the same information that a flat file does, but more efficiently.

Example of a database table

Here, we have a table storing information about the users of some website.

id	**username**	**email**	full**_name**
1	rnubel	test@gmail.com	Robert Nubel
2	otherperson	test@aol.com	Other Person
3	thirduser	test3@gmail.com	Third User

Designing databases

- It's non-trivial to design databases, so we won't cover that here.
  - Take CS411 if you're interested
- However, pulling data from a database is pretty easy, and we almost know enough to do it.
  - Need to know how the blog posts table is designed (this is called a table's **schema**)
  - Need to somehow select the correct row from the blog posts table

The blog posts table

blog_posts:

id	title	author	content
1	Hello Blogosphere!	Robert Nubel	Blah blah blah.
2	Blogpost Two	Other Person	Trivial nonsense.
3	Probably Never Going to Update This Blog Again	Third User	We're done here.

Note: That's not really a schema

- While just showing what data a table holds is often enough to infer the schema from, the actual schema for the table is more like:

```
CREATE TABLE blog_posts (
    id          INTEGER PRIMARY KEY,
    title        VARCHAR(255),
    author     VARCHAR(32),
    content    TEXT
);
```

But where do we get a database from?

- MySQL is a popular database management system (DBMS) that came bundled with the WAMP stack we installed earlier.
- To use the database, you need to start it up (Start All Services will do this)
  - This starts the mysql daemon, which among other things will accept connections from, for example, your PHP scripts and allow you to query the database.
- Then, we need to run an SQL query to insert some data into the database.
  - SQL = Structured Query Language
  - Lets you create tables, insert data, retrieve data, etc.

Running an SQL query on MySQL

- You can use MySQL's command prompt to enter queries if you'd like, or you can use a web interface, phpMyAdmin, which connects to your MySQL server and lets you manage it graphically.
  - If you're using WAMP, left-click on the tray icon and select "phpMyAdmin".
  - Once it loads, click on the SQL tab.
- If you don't have phpMyAdmin, find your mysql command prompt and log in.
  - Username is probably 'root' with no password
  - Once logged in, you're ready.

Run this query:

```
CREATE DATABASE blog;
USE blog;
CREATE TABLE blog_posts (
    id INTEGER PRIMARY KEY,
    title VARCHAR(255),
    author VARCHAR(32),
    content TEXT
);
INSERT INTO blog_posts VALUES (1, "Title of Post", "Robert
Nubel", "Blah blah blah content");
```

That's it for the database.

- Now, we need to write a PHP script which can connect to our database.
- Luckily, PHP has a set of built-in functions that let us work with MySQL:
  - mysql_connect: connect to the server
  - mysql_select_db: select the database to work with
  - mysql_query: run a query on the server

Connecting

- Open up a new file and save it as blog.php.
- Type in this code:

```
<?php
if (!array_key_exists('post_id', $_GET)) {
    die("No post id provided!"); // halt execution now
}
$post_id = $_GET['post_id']; // what could go wrong?
mysql_connect('localhost', 'root', ''); // remember localhost?
mysql_select_db('blog');
```

Querying the database

```
$sql = "SELECT * FROM blog_posts WHERE id=$post_id";
$qry = mysql_query($sql) or die(mysql_error());

// Fetch an associative array of the first row's data. If no rows
// were returned, this will fail.
$result = mysql_fetch_assoc($qry);

echo "<h1>" . $result['title'] ."<h1>";
echo "<h2>By " . $result['author'] . "</h2>";
echo "<p>" . $result['content'] . "</p>";

?>
```

Try it out!

- Go to localhost/blog.php?post_id=1 in your browser.

Important note: don't trust your request

- Try going to localhost/blog.php?post_id=' (apostrophe included)
- An SQL error comes up -- this means the ' is being inserted right into the query
  - This means your database can have arbitrary SQL commands sent to it, as you've opened yourself up to **SQL injection**
- To prevent this, **always sanitize your data** before putting it into a query.
- $post_id = (int) $_GET['post_id'];
  - This typecasts the parameter to an int. The worst that happens is you get a blog post ID that doesn't exist.

In conclusion

- Dynamic web programming is essential to the modern web.
- Many different technologies -- PHP and MySQL just being a small subset -- exist which can work together to power a website.
- In the next tutorial, we'll explore a design pattern used to structure large-scale websites: the MVC pattern.