ICOM 6034
Website Engineering

Dr. Roy Ho
Department of Computer Science, HKU

Session 3: Techniques for adaptability

Session objectives

- Server-side scripting
  - PHP for handling form submission
  - Storing state information with PHP
- Other issues in website development
  - Basic web security
  - Supports for multiple/mobile devices and Responsive Web Design (RWD)
  - Internationalization (i18n) and Unicode
- Introduction to Web 2.0
- Summary of Part 1 of the course
- Lab 1C: Simple PHP programming

Server-side scripting

The server side for providing dynamic contents

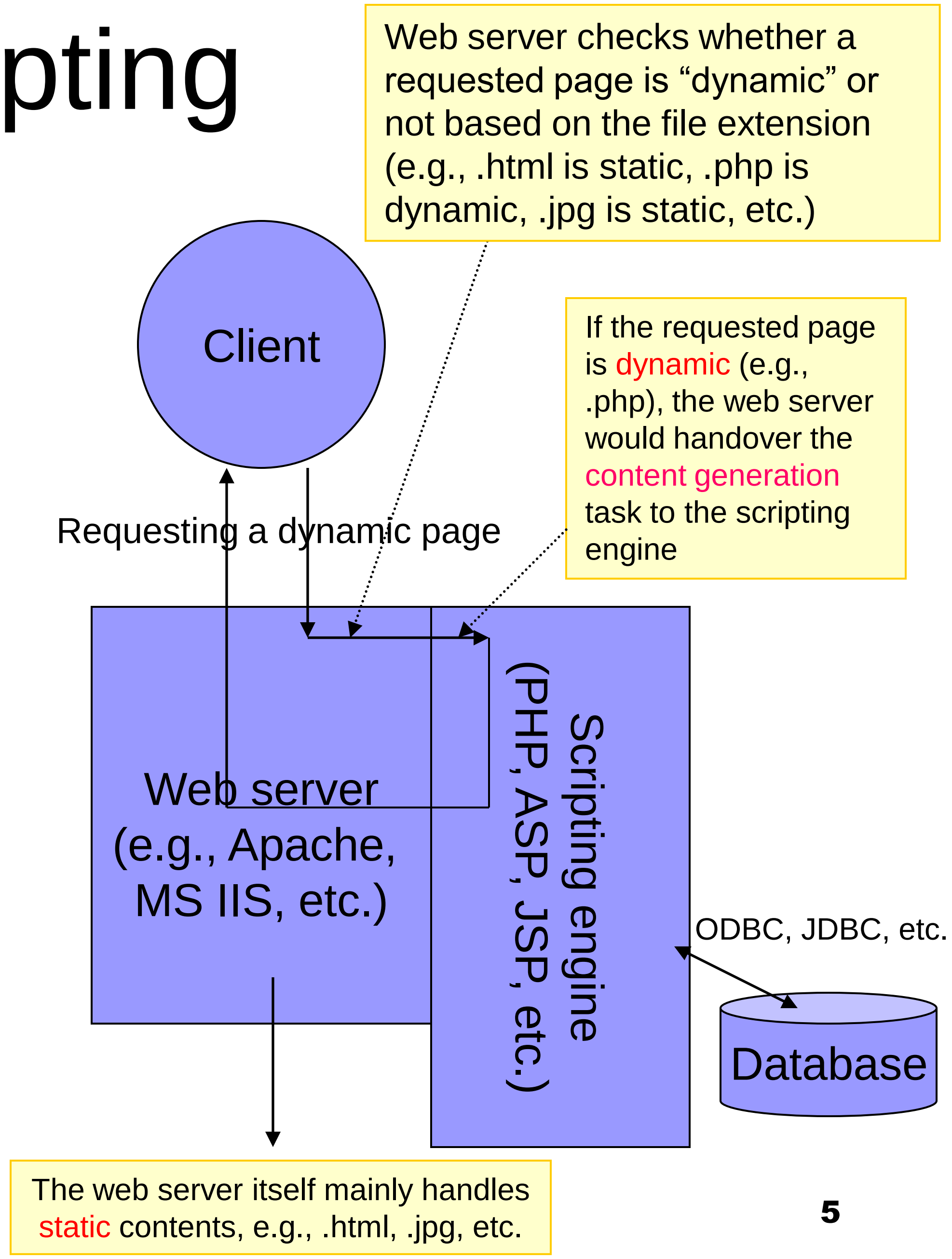
- Client-side technologies (HTML, CSS, JS, DOM, etc.) mainly deal with how content is presented based on the client's platform (e.g., browser/device) and user actions
  - The "content" itself (i.e., the webpage) is mostly static once delivered
  - Even if JavaScript is used, the content can't change much without fetching new data from the server

- In many cases, dynamically-generated contents (or simply "dynamic contents") are needed
  - News portals
  - Comments / number of "Likes" in Facebook, Instagram, etc.
  - Real-time stock quotes
  - ……

- The generation of dynamic contents relies on server-side technologies

Server-side scripting

- Popular technologies:
  - PHP (Hypertext Preprocessor)
  - Microsoft ASP.NET
  - JSP (Java Server Pages),
  - JavaScript on Node.js, …… etc.
- The scripting engine may connect to a backend database (DB) for data retrieval
- Popular DB for web: MySQL (or its open-source variant: MariaDB), MS SQL Server, Oracle, IBM DB2, etc.


Web server checks whether a requested page is "dynamic" or not based on the file extension (e.g., .html is static, .php is dynamic, .jpg is static, etc.)
Client
If the requested page is dynamic (e.g., .php), the web server would handover the content generation task to the scripting engine
Requesting a dynamic page
Web server (e.g., Apache, MS IIS, etc.)
Scripting engine (PHP, ASP, JSP, etc.)
ODBC, JDBC, etc.
Database
The web server itself mainly handles static contents, e.g., .html, .jpg, etc.

Example: PHP

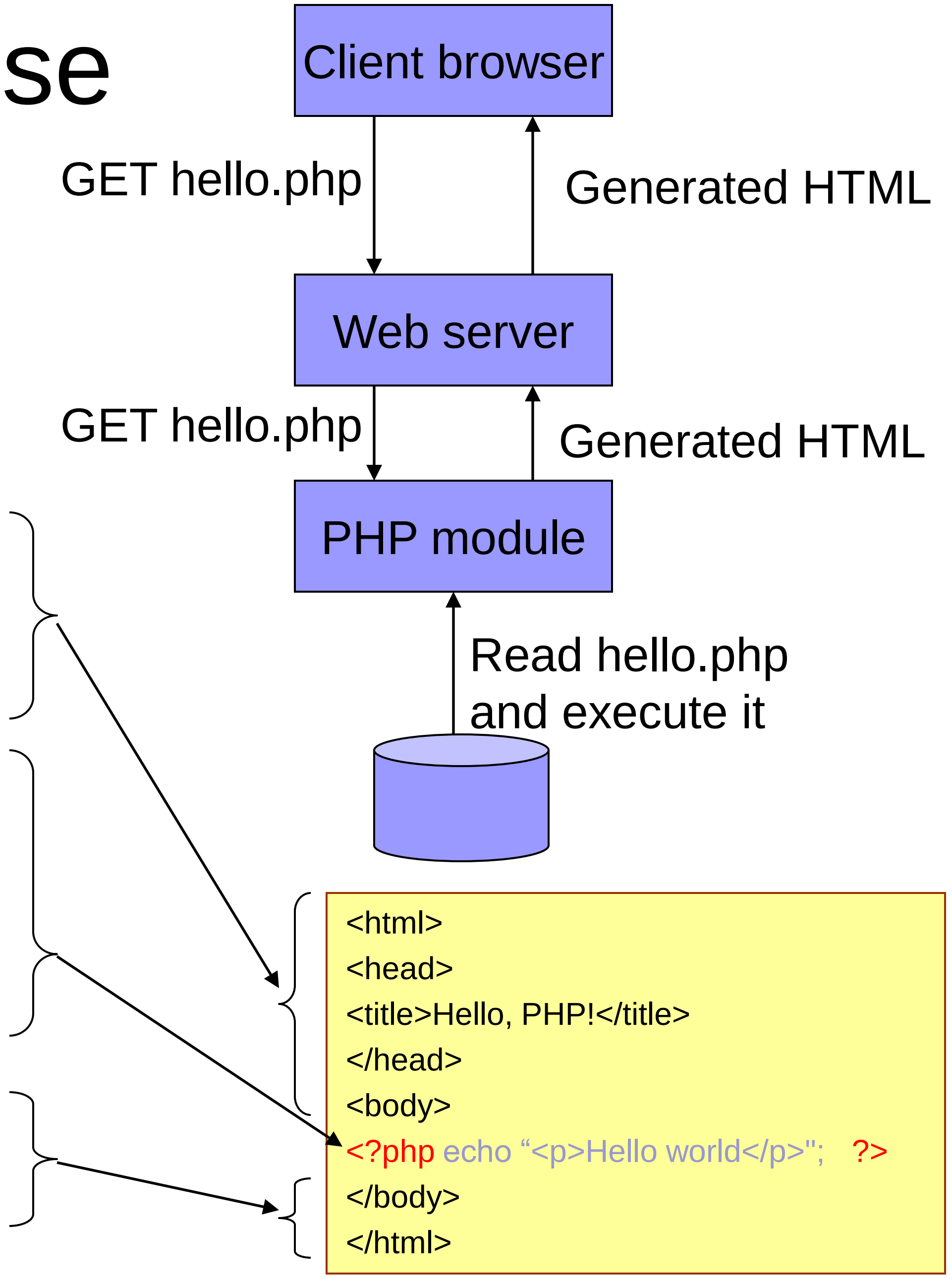
- PHP originally stood for “Personal Home Page”, now stands for “Hypertext Preprocessor”
  - Started since 1994
- Cross-platform - available in Windows, Mac OS and Linux
- Compared to the other server programming supports
  - Excellent for quick prototyping / small projects
  - Ready for even the biggest projects as well – Facebook.com
  - Great integration with MySQL/MariaDB – a popular open-source database server
- According to W3Techs.com, PHP is used by 79% of websites in the world (as of November, 2020)
- Syntax of PHP is very similar to the other popular server-side scripting languages, e.g., JSP, ASP, etc.

Request & response

- The web server, after receiving the request for hello.php, will handover the task to the embedded PHP module.
- The PHP module has two modes of operations: HTML mode and PHP mode
- In the HTML mode, the PHP module simply sends out all regular HTML content to the client
- But once a PHP code block is encountered:

  <?php … ?>

  it would be executed as a PHP program (i.e., the "PHP mode") and the output, if any, would be sent to the client.
- When the code block finishes, the PHP module would switch back to the "HTML mode" again.



```
<html>
<head>
<title>Hello, PHP!</title>
</head>
<body>
<?php echo "<p>Hello world</p>";   ?>
</body>
</html>
```

PHP vs. the final HTML output

```
<html>
<head>
<title>Hello, PHP!</title>
</head>
<body>
<?php echo "<p>Hello world</p>";   ?>
</body>
</html>
```

The PHP program processed by the web server

```
<html>
<head>
<title>Hello, PHP!</title>
</head>
<body>
<p>Hello world</p>
</body>
</html>
```

What your browser would receive

For an introduction to basic PHP syntax and programming, please refer to the PHP tutorial:

https://www.w3schools.com/php/default.asp

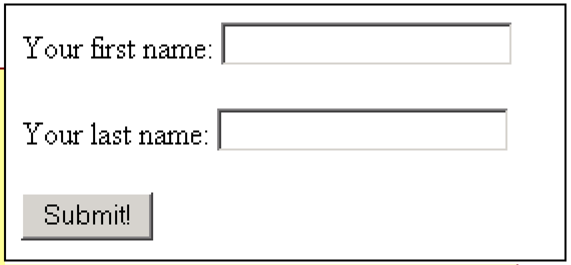
Form handling

- Form generation and handling are the most common usage of server-side scripting. Typical steps:
  - Server sends an empty form (in HTML) to a user
  - User fills in the form, and submits it to server
  - Server receives the data, generates the output to the user
- Two common approaches to perform the above steps:
- Approach 1
  - You send a plain HTML file (i.e., not generated by PHP) to a user, which contains a form
  - The form is submitted to a PHP program
  - The PHP program then processes the form and generates output to the user
- Approach 2
  - A single PHP program performs all:
    - Generation of the initial HTML page, which contains a form
    - The form would be submitted to the same PHP program
    - That same PHP program would generate the output to the user
- Other similar approaches: e.g., a PHP program for form generation and another for accepting form submission and generating the output

Approach 1:
A simple HTML form

Can be GET or POST

Shown in the next slide



```
name.html

<html><body>
<form method="POST" action="name.php">
    <p>Your first name: <input type="text" name="fname"></p>
    <p>Your last name: <input type="text" name="lname"></p>
    <button type="submit">Submit!</button>
</form>
</body></html>
```

- GET – user input data would be appended to the end of URL
  - Something like https://<site>/<path>/name.php?fname=Steven&lname=Chu

A “query string”

- POST – user input data (i.e.,form fields) would be embedded in (and sent through) the HTTP request header
  - The URL for submission (http://<site>/<path>/name.php) is unchanged
- => a form submitted through GET can be bookmark’ed, but not POST
  - Observe the implications
  - => GET should **NOT** be used for operations that can change DB data (“side-effects”)
    - Because if the bookmark is used again, the same form would be submitted and the DB data would be changed again
  - I.e., GET (and “query strings”) should only be used for “side-effect-free” operations

name.php

Suppose the user enters “Steven” as the first name and “Chu” as the last name.

```
<html><body>
<p>Thank you, <?php
echo $_POST["fname"] . " " . $_POST["lname"] . "!<p>";
?>
</body></html>
```

```
<html><body>
<p>Thank you, Steven Chu!<p></body></html>
```

The generated HTML

```
Thank you, Steven Chu!
```

What would be displayed in the browser

- $_POST[“fname”] retrieves the submitted form value that is named “fname”.
- $_GET[“input_name”] is similar, but for GET operations

Approach 2

Check if there is any submitted data

```
<html><body>
<?php
if ($_POST["fname"]) { ?>

<p>Thank you, <?php
echo $_POST["fname"] . " " . $_POST["lname"] . "!<p>";
?>

<?php
} else {  ?>

<form method="POST" action="name.php">
<p>Your first name: <input type="text" name="fname"></p>
<p>Your last name: <input type="text" name="lname"></p>
<button type="submit">Submit!</button>
</form>

<?php } ?>
</body></html>
```

If there is submitted data, process the form and generate the output

If there is no submitted data, generate the initial page which contains an empty form

This approach might be easier for error checking or data validation.

Adding error checking

```
<html><body>
<?php
if ($_POST["fname"]) {
    if (!$_POST["lname"])
        $error = "Please enter last name.";
}

if (!$_POST["fname"] || $error)  {

echo "$error\n“; ?>

<form method="POST" action="name.php">
<p>Your first name: <input type="text" name="fname"></p>
<p>Your last name: <input type="text" name="lname"></p>
<button type="submit">Submit!</button>
</form>

<?php } else { ?>
    <p>Thank you, <?php
    echo $_POST["fname"] . " " . $_POST["lname"] . "!<p>";
}  ?>

</body></html>
```

Check if the last name is entered

If $error is set <u>or</u> if fname is not set, send an empty form to the user

If $error is set, add a reminder

If both first and last names are provided, print the thank you message.

Storing state information in PHP

State information (1/2)

- Information about individual requests to a website is called **state information**
  - E.g., if you are logged in as a user called “`steven`”, you would want the server to know that you are “`steven`” in your future requests. That “`steven`” is a piece of state information

- Some common usage/examples of state information:
  - User IDs and passwords
  - Keep track the last login time of a user
  - Load a “theme” based on user settings
  - Handle a long HTML form that spans multiple pages
  - Shopping carts that store order information, etc.

- We often need to store state information, **but**: HTTP is designed to be **stateless**:
  - I.e., every web request is independent from each other;
  - Web servers have no prior knowledge about any requests to a website

- => We need special mechanisms to maintain state
  - So that state information can be preserved in-between multiple web requests

State information (2/2)

- Four common ways for maintaining state information with PHP:
  - Hidden form fields
  - Query strings
  - Cookies
  - Sessions

- The techniques for maintaining state information in other scripting languages are similar

Hidden form fields

- Hidden form fields temporarily store data inside an HTML form that the user does not need to see (e.g., the ID number of a user account)
  - But the server-side program can "see" it again once the form is submitted
  - Can be used to pass state information from one PHP program to another

- Syntax:
  - `<input type="hidden" name="..." value="...">`

- The target PHP program can access the submitted values with the `$_GET[]` or `$_POST[]` arrays

main.php:

```
<form action="enroll.php" method="POST">
<p><input type="submit" value="Enroll for classes" />
<input type="hidden" name="userid"
     value="<?php echo $userid; ?>" /></p>
(...... Other form elements ......)
</form>
```

main.php inserts the user ID into the generated HTML page

enroll.php will then receive $userid after form submission.

=> $userid is "passed" from main.php to enroll.php.

Query strings

- A **query string** is a set of name=value pairs appended to a target URL
  - For passing information from one webpage to another

The URL and the information are separated by a question mark (?)

name=value pairs are separated by ampersands (&)

- E.g., in the following link:
  - `<a href="http://www.test.com/test.php?fname=Fred&userid=fchan">This is a link.</a>`
  - The information “fname=Fred” and “userid=fchan” would be passed to test.php

- In PHP, values stored in a query string can be retrieved by using the $_GET array, e.g., $_GET[‘fname’].

Cookies

- Hidden form fields and query strings <u>cannot</u> permanently store state information at the client side
  - When the browser (tab) is closed, the hidden form fields or query string would be lost
- To store state information beyond the current web page session, cookies can be used
- **Cookies** are information sent from the web server and stored as text files on the client’s computer.
  - They will be sent <u>back</u> to the server next time when the user retrieves the page again.
  - So the web server would know the client’s previous “states”
- Each individual domain can store only up to 20 cookies on a client’s computer, each up to 4 kilobytes large

Creating and reading cookies

- Example - syntax in PHP:

```
setcookie(name [,value ,expires, path, domain, secure])
     e.g., setcookie('userid', 'fchan');
```

- `setcookie()` has to be called before sending out any HTML elements

- Reading cookies:
  - At the server side, information received from cookies are automatically stored in the `$_COOKIE` array (similar to $_GET or $_POST)
  - E.g.,

```
echo $_COOKIE['userid'];
```

- Detailed usage of setcookie: https://hk.php.net/setcookie

- Potential issue:
  - Users can disable cookie in browsers
  - But setcookie() always returns true, so you can never tell whether cookies are accepted at the client side or not

Sessions

- A **session** refers to a period of time when the state information of a particular client is stored at the server side (while cookies store data at the client side)
  - So it allows you to maintain state information even when clients disable cookies
- In PHP, the `session_start()` function:
  - Starts a new session **or** continues using the current one
  - For a new session, generates a unique session ID to identify the session
  - Creates a temporary text file on the web server that has the same name as the session ID, which stores session variables at the server side
  - E.g., <?php session_start(); ?>
  - No parameters are needed
- A **session ID** is a random alphanumeric string that looks something like:

  `7f39d7dd020773f115d753c71290e11f`

Working with session variables

```
<?php
session_start();
$_SESSION['firstName'] = "Fred";
$_SESSION['lastName'] = "Chan";
$_SESSION['userid'] = "fchan";
?>
<p><a href='<?php echo "test.php?PHPSESSID="
     . session_id() ?>'>This is a test.</a></p>
```

Session variables stored at the server side

By providing the session ID to a target program (test.php), it will know which session this client is using, and will be able to retrieve the stored information.

Some complicated forms (spanning multiple HTML pages) use the session_id as the primary key and store the data in a database before all sub-forms are submitted.

Working with sessions

- Use the `isset()` function to test whether a session variable is set before using it

```
<?php
session_start();
if (isset($_SESSION['firstName'])) {
  // do something with 'firstname'
}
?>
```

- To delete a session manually, perform the following steps:

1. Execute the `session_start()` function
2. Assign an empty array (array()) to the `$_SESSION` variable
3. Use the `session_destroy()` function to delete the session

```
<?php
session_start();
$_SESSION = array();
session_destroy();
?>
```

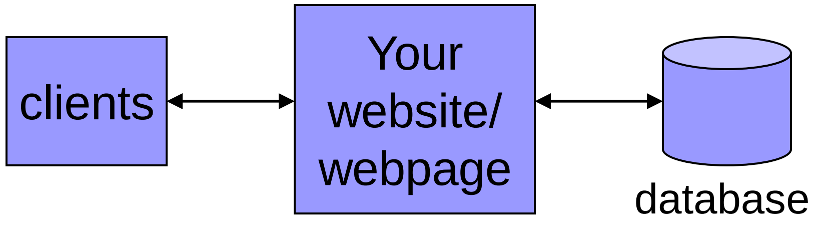
- You can also set an “expiry date/time” for a session and check its “age” – see the documentation for the details.

Basic web security

External data

Web security is covered in-depth in ICOM6045. This section will only introduce the most common web attacks and ways to avoid them.

- External data is not to be trusted.



- What is external data?
  - Anything from a form
  - Anything from $_GET, $_POST……
  - Cookies (can be modified by clients manually)
  - ----------
  - Database query results (can have I/O failures or contain errors)
  - Data files (can have I/O failures or contain errors)

- The main technique is to escape output (especially to DB) and check and filter input (from any external parties)

- Escape output: e.g., “O’Reilly” should be “escaped” as “O\’Reilly” so that it would not corrupt or affect any SQL statements that handle it

Common web attacks

- SQL injection
- XSS (Cross Site Scripting)
- Session fixation
- Session hijacking
- Cross site request forgeries (CSRF)

1. SQL injection

- An attacker may input SQL in form fields in a way that affects the execution of SQL statements.

$username is provided by a client

```
$username = $_POST['username'];

$query = "select * from auth where username = '".$username."'";
echo $query;
$db = new mysqli('localhost', 'demo', `demo', `demodemo');

$result = $db->query($query);

if ($result && $result->num_rows) {
  echo "<br />Logged in successfully";
} else {
  echo "<br />Login failed";
}
```

$username is not escaped!

This query could fail or even corrupt the DB if $username contains strings that transform your SQL statement ($query) to something harmful.

Preventing SQL injection

- Options:
  - Manually check and ensure that each data item has the correct format
  - Filter data using mysql[i]_real_escape_string()
  - Use prepared statements

- Use prepared statements
  - Goal: separate data from the SQL code
  - The prepared statements will do filtering (e.g., escape) automatically

```
$query = 'select name, district from city where countrycode=?';
if ($stmt = $db->prepare($query) )
{
  $countrycode = 'fr';                  // ‘fr’ = France
  $stmt->bind_param("s", $countrycode);
  $stmt->execute();
  $stmt->bind_result($name, $district);
  while ($stmt->fetch())
  {
    echo $name.', '.$district;
    echo '<br />';
  }
  $stmt->close();
}
$db->close();
```

2. XSS

- XSS = Cross Site Scripting
- An attacker may submit some data to your website, which contains JavaScript
- If you include this data in a web page sent to another client, the JavaScript would be executed at his browser…

Example

Accepting text comments from user:

```
<?php
if (file_exists('comments')) {
  $comments = get_saved_contents_from_file('comments');
} else {
    $comments = '';
}

if (isset($_POST['comment'])) {
    $comments .= '<br />' . $_POST['comment'];
    save_contents_to_file('comments', $comments);
}
?>
```

Outputting comments to (another) user:

```
<form action='xss.php' method='POST'>
  Enter your comments here: <br />
  <textarea name='comment'></textarea> <br />
  <input type='submit' value='Post comment' />
  </form><hr /><br />

<?php echo $comments; ?>
```

- What can happen:
  - Corrupted pages or forms
  - Annoying popups
  - Refresh or redirections
  - Steal cookies
    - (which can then be used to set up a session hijacking attack)

Preventing XSS

- Filter HTML output using htmlentities() in PHP (or something similar in other server-side languages).
  - Converts all characters to HTML (even if they are JS code)

- Basic usage of htmlentities() is simple, but there are many advanced controls

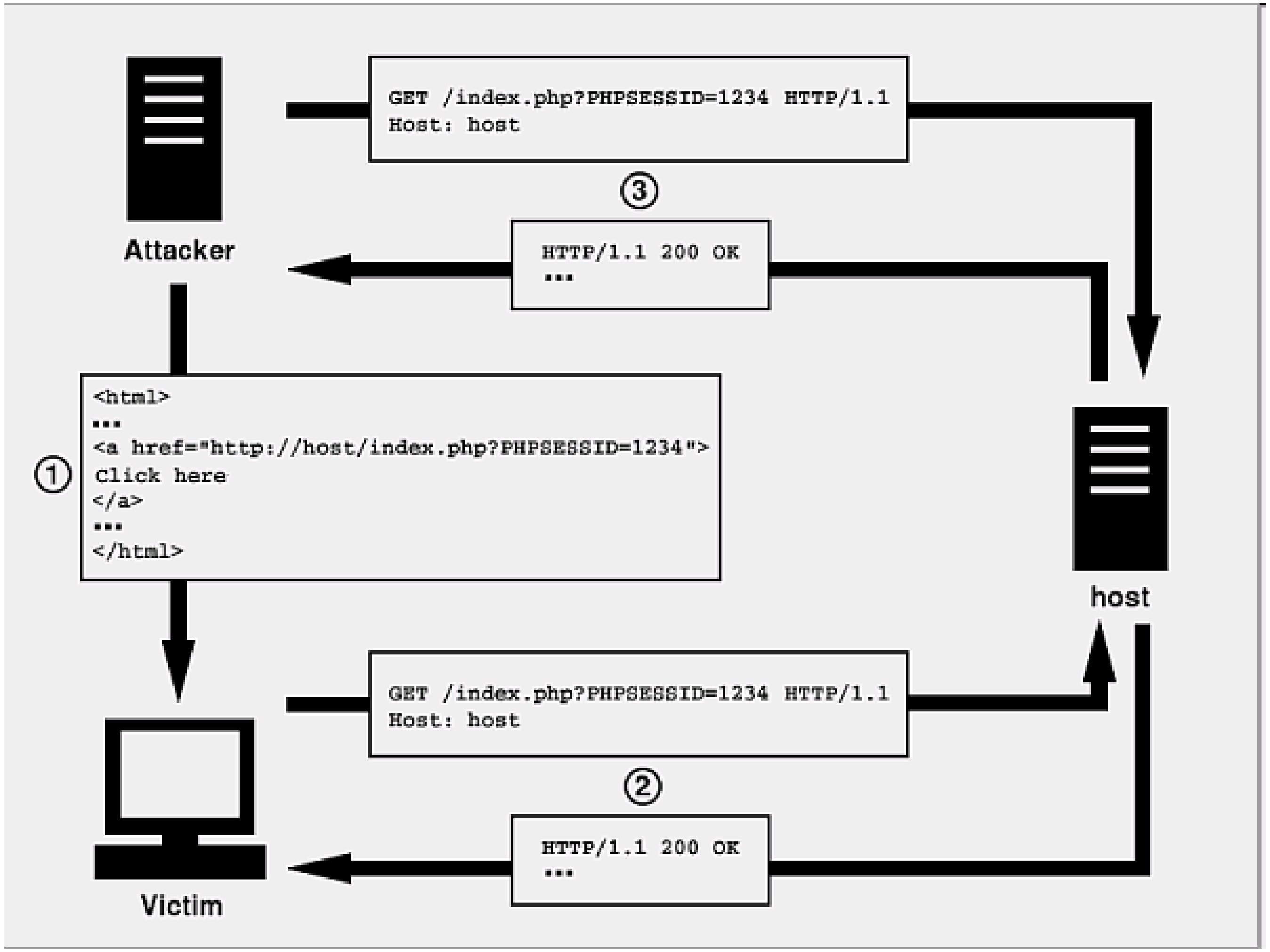
- For other techniques, see the XSS prevention cheat sheet:

  https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

3. Session fixation

- Session security works on an assumption that the session ID is difficult to guess

- However: an attacker can trick a victim to use a known (or pre-defined) session ID => a phishing attack would become possible



```
GET /index.php?PHPSESSID=1234 HTTP/1.1
Host: host
```

```
HTTP/1.1 200 OK
...
```

```
<html>
...
<a href="http://host/index.php?PHPSESSID=1234">
Click here
</a>
...
</html>
```

```
GET /index.php?PHPSESSID=1234 HTTP/1.1
Host: host
```

```
HTTP/1.1 200 OK
...
```

Solution: use session_regenerate_id() in PHP whenever a user logs in, changes IP, or changes his level of privilege.

4. Session hijacking

- Same idea but involves stealing the session ID.
- If session IDs are stored in cookies, attackers can steal them through XSS and JavaScript
- Session IDs can also be sniffed, or obtained from proxy servers if contained in the URL
- Solutions:
  - Regenerate IDs
  - Avoid to include session IDs in URLs (i.e., use POST instead of GET)
  - Always encrypt communication (with SSL/TLS) for sessions :
    - I.e., use “https://” instead of “http://”

5. CSRF

- CSRF = Cross Site Request Forgeries
- A request that looks like it was sent from a trusted user, but wasn't. Many variations.
- Example:

```
<img
src='http://example.com/single_click_to_buy.php?user_id=123&item=12345'>
```

- In general, make sure that clients really come from pages generated by you, and each form submission is matched to a particular form that you send out.
  - Use session with appropriate security measures
    - Regenerate IDs and use SSL for every session
  - Generate another one-time token and embed it in the form, save it in the session, and check it on submission.
  - Some modern server-side frameworks (e.g., Laravel) perform these steps automatically by default – more on this in Part 2.

Other general principles

- Don’t rely on server configuration to protect your website especially if your web server is managed by a third-party (e.g., web hosting company), or if your website may be migrated to somewhere else in future
  - Embed the security checking/logic in the website code (PHP, JavaScript, etc.)
- Design your server-side scripts with security from the ground up:
  - Delegate all authentication/security logic in one PHP function to be included in all pages
    - So that no pages are omitted from security checking, and that problems can be easily traced and isolated
  - Do the same for data filtering
- Keep your servers, code and libraries up to date. Stay on top of patches and advisories.

Additional references on web security

- Those who are interested to learn more can refer to the following book:
  - H. Wu and L. Zhao. Web security: a WhiteHat perspective. CRC Press. 2014.
    - Available as e-book through HKU Library.
    - http://find.lib.hku.hk/record=HKU_IZ51488843860003414

Supports for mobile/multiple devices and Responsive Web Design (RWD)

Limitations of mobile web

- Limited network bandwidth and long network latency
  - Mobile networks for some people/countries are still slow
  - Some faster data plans are expensive
  - Limited battery in mobile devices
  - 4G networks have higher bandwidth (e.g., 10+Mbps, etc.), but the latency is still a lot higher than broadband networks (partially due to the higher error rate in mobile communication)
  - 5G has shorter latencies than 4G, but its usage is far less common in most countries
- Limited screen size - lots of scrolling required
- Text input and navigation
  - Low typing speed
  - Inaccuracies of touch-based interactions
- Limited CPU power and memory – slow JS executions

W3C recommendations for mobile web

- For <u>informational web pages</u>, W3C has compiled the **Mobile Web Best Practices**, which recommends to:
  - Provide the best possible client experience in the context where their service has the most appeal
    - E.g., what elements are most "appealing" (i.e., important) in the <u>mobile web pages</u> of HKU? Provide the best possible client experience for these elements.
      - E.g., class schedules, campus maps, opening hours of libraries, travel instructions, etc.
      - Teachers' info might be less important.
  - Provide as **reasonable experience** as is possible given device limitations and not to exclude access from any particular class of device
  - http://www.w3.org/TR/mobile-bp/

- For interactive web applications, W3C has the **Mobile Web Application Best Practices**
  - Many recommendations; some of the more important ones:
    - Use local storage (e.g., through HTML5 – see Session 2) to shorten client-perceived latencies
    - Use data compression for data transfer whenever possible (more on this in Session 10)
    - Avoid redirects – why?
    - Keep pages simple (so that the DOM can be stored in the limited RAM)
    - Make telephone numbers "click-to-call"
    - (and many others)
  - http://www.w3.org/TR/mwabp/

Adaptation for mobile-/multi-devices

- Suppose we already have a desktop website in place, how to make it mobile-friendly?

- Possible approaches:
  - do nothing
  - reduce image resolution and simplify styling
  - special stylesheets (and JS routines, if needed) for mobile devices
  - mobile-specific site or application

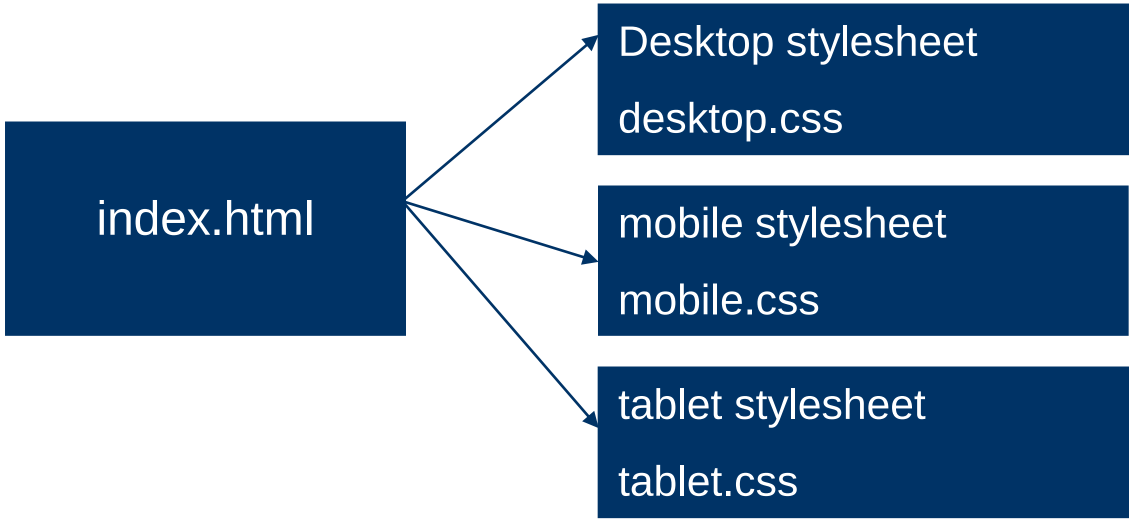
A solution flexible enough for simple, informational websites

Easiest to do; worst client experience

More efforts required; better client experience

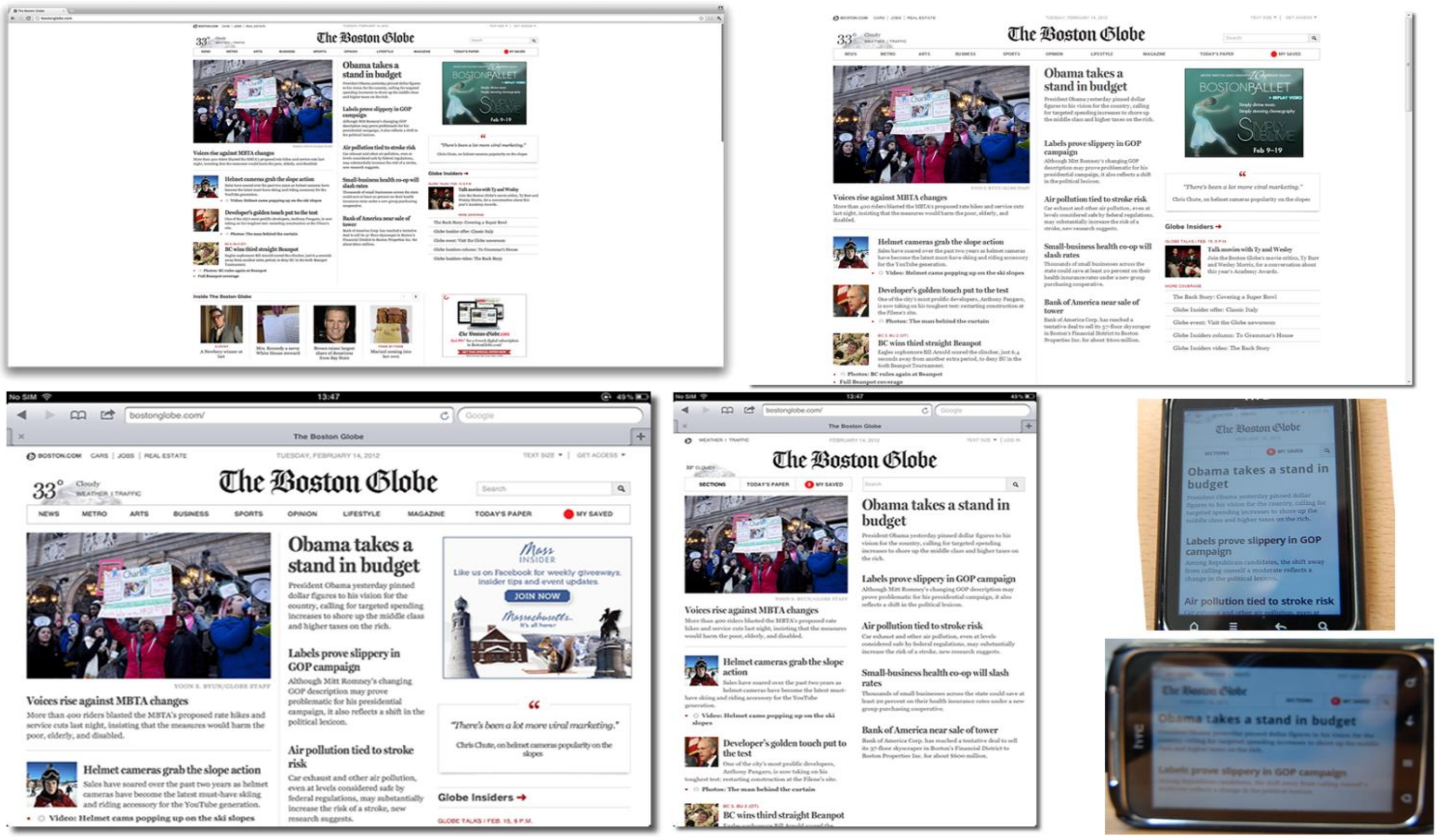
Stylesheets for mobile devices

- If the document structure and presentation are well separated, apply the appropriate css file based on the detected platform (desktop / phone / tablet, etc.) and screen sizes
  - Yet another important reason for clean separation



- Okay for simple websites, but what if we want to customize the web pages for each (mobile/desktop) device in order to provide the best user experiences?
  - Example: even a single product (e.g., iPad) can have a unique screen size and resolution; new generations of devices appear every year…
  - Problem: many css files would be needed

Q: How to make the website itself “adaptive” - automatically?


The Boston Globe
Obama takes a stand in budget
Labels prove slippery in GOP campaign
Air pollution tied to stroke risk
Voices rise against MBTA changes
Helmet cameras grab the slope action

E.g., The Boston Globe

Responsive web design

- **Responsive web design** (**RWD**) aims to:
  - Provide an optimal viewing experience
    - I.e., easy reading and navigation with a minimum of resizing and scrolling
  - …… across a wide range of devices (from desktop computers to tablets to mobile phones)

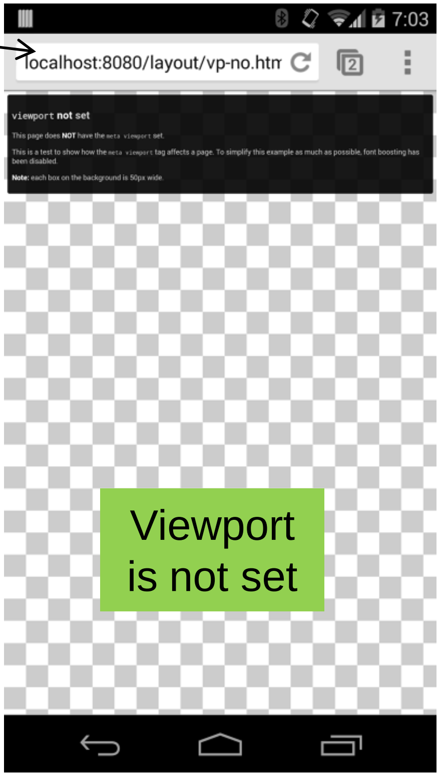
- Common techniques for RWD:
  - Viewport setting
  - Media queries
  - Flexible images
  - Fluid grids

Viewport setting

- Goal: to avoid horizontal scrolling in mobile devices
- Viewport is the user’s visible area of a web page
  - => to avoid horizontal scrolling, viewport’s width should be set to the device’s width (in pixels)
- Insert this to the page’s <head> section:
  - <meta name="viewport" content="width=device-width, initial-scale=1">

Many mobile browsers assume that a webpage is designed for desktop by default. So, if the viewport is not set, the browser would try to display the page in a 980px-wide “container” so that it would look closer to its original (desktop) design.

Users would have to zoom in (with horizontal scrolling) to read the content => poor user experience.



viewport **not set**

This page does **NOT** have the `meta viewport` set.

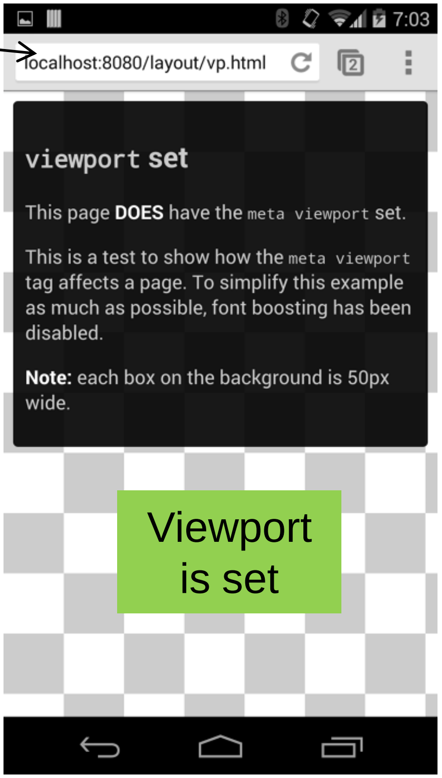
This is a test to show how the `meta viewport` tag affects a page. To simplify this example as much as possible, font boosting has been disabled.

Note: each box on the background is 50px wide.

Viewport is set

The page would span the screen’s width, with no scaling of content.

The browser can “reflow” the content to match different screen sizes.



viewport **set**

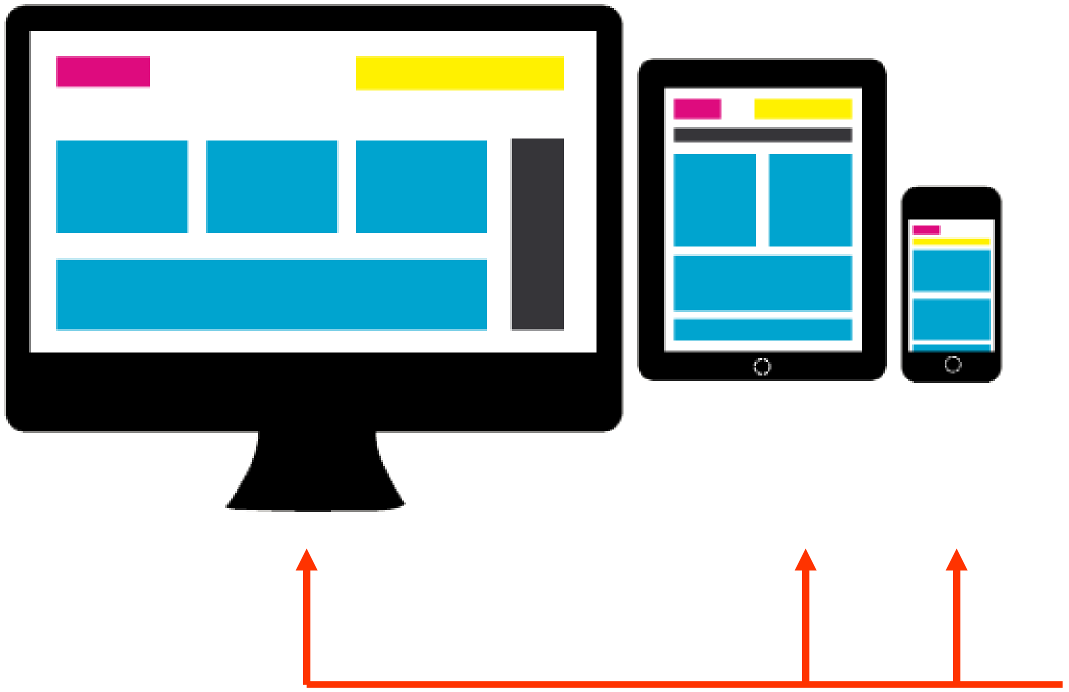
This page **DOES** have the `meta viewport` set.

This is a test to show how the `meta viewport` tag affects a page. To simplify this example as much as possible, font boosting has been disabled.

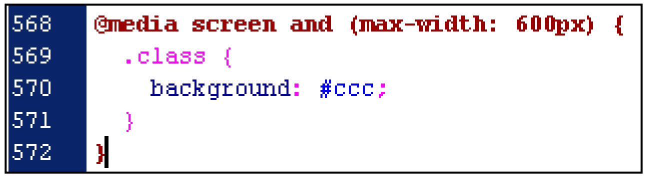
Note: each box on the background is 50px wide.

CSS3 media queries

How does it work?



CSS3 Media Queries



```
@media screen and (max-width: 600px) {
  .class {
    background: #ccc;
  }
}
```

Apply the style if the screen width is smaller than 600px

With more conditions, the site changes, adapting to the resolution of the device

The site changes (“reacts”) depending on the settings

CSS3 media queries

- A media query consists of a media type and an expression to check for certain conditions of a particular media feature.
- The most commonly-used media features are screen width (in pixels), aspect-ratio, screen orientation (portrait/landscape), resolution (in dpi), etc.

- See https://www.w3schools.com/cssref/css3_pr_mediaquery.asp for a list of @media features

- Another example of @media in CSS:

```
@media screen and (min-width: 480px) {
   .content { float: left; }
   .icons { display: none }
   // and so on...
}
```

- Media queries can also be used for selecting external stylesheets:

```
<link rel="stylesheet" href="tablet.css" media="(min-width: 960px)">
```

Typical viewport sizes

(for reference only)

- **1440 x 2960 px:** *Smartphone*
- **2960 x 1440 px:** *Smartphone in landscape orientation*
- **1668 x 2224 px:** *iPad / tablet*
- **2224 x 1668 px:** *iPad in landscape orientation*
- **1920 x 1080 px:** *Desktop / laptop computer*

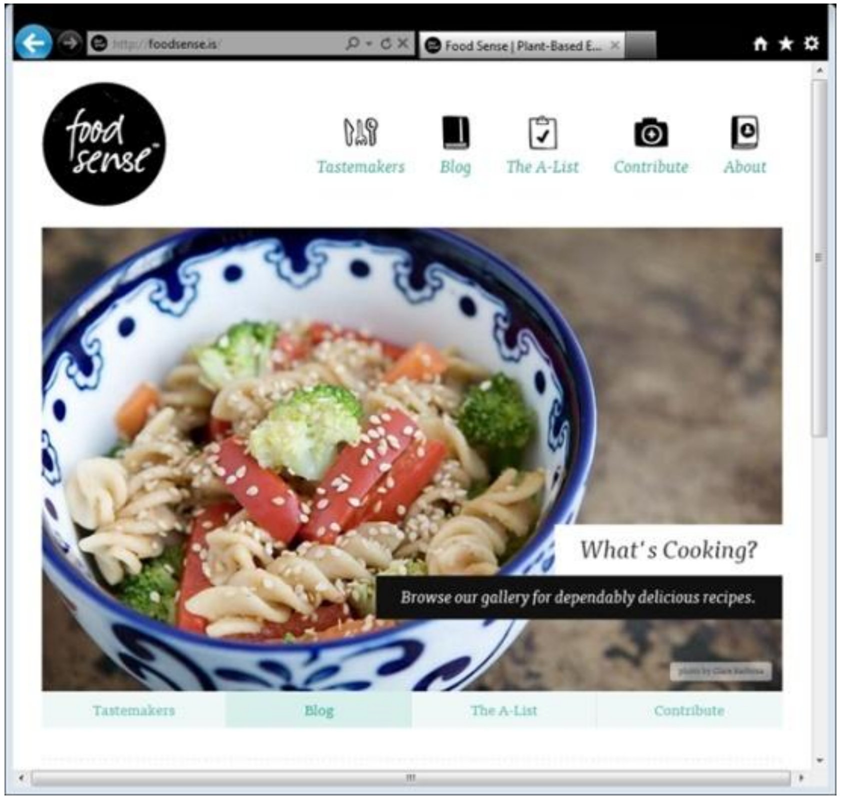
These numbers are changed from time to time with hardware upgrades.

Website maintainers constantly need to update these numbers for newer generations of hardware (or even for a new, popular device, e.g., a new version of iPad/MacBook).

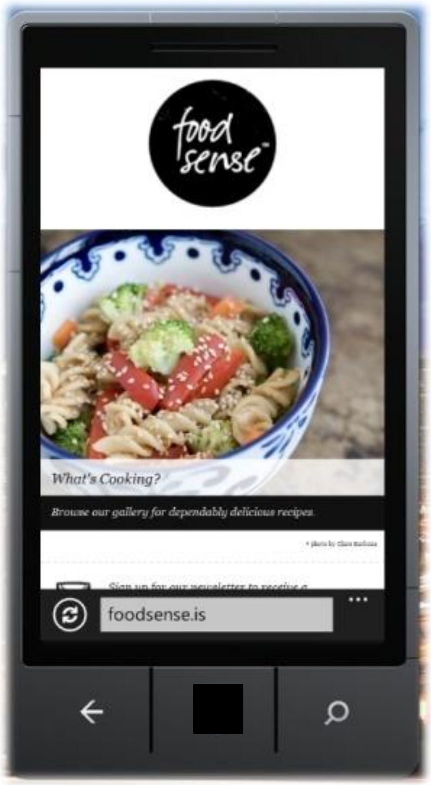
Flexible images

- Flexible images are sized in relative units (up to 100% of its “container”), instead of pixels, so as to prevent them from displaying outside their containing element (or browser)
- E.g.,
  - img { max-width: 100%; }

Desktop:



Mobile:



Fluid grids

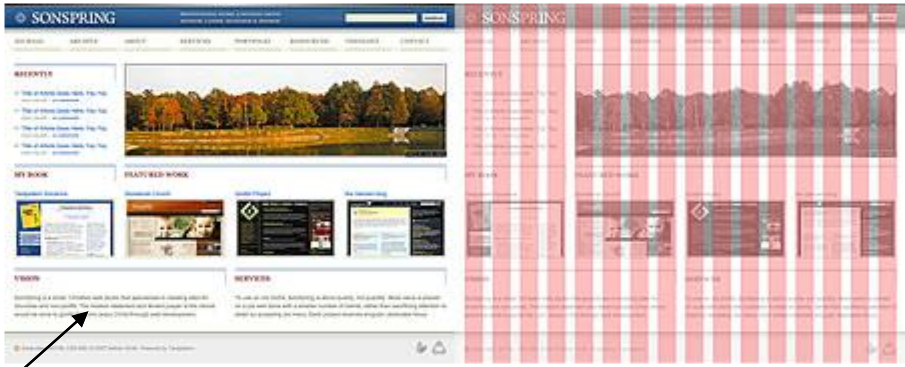
Mobile users prefer vertical scrolling to horizontal scrolling, so the number of “columns” of content should be constrained according to the screen width.

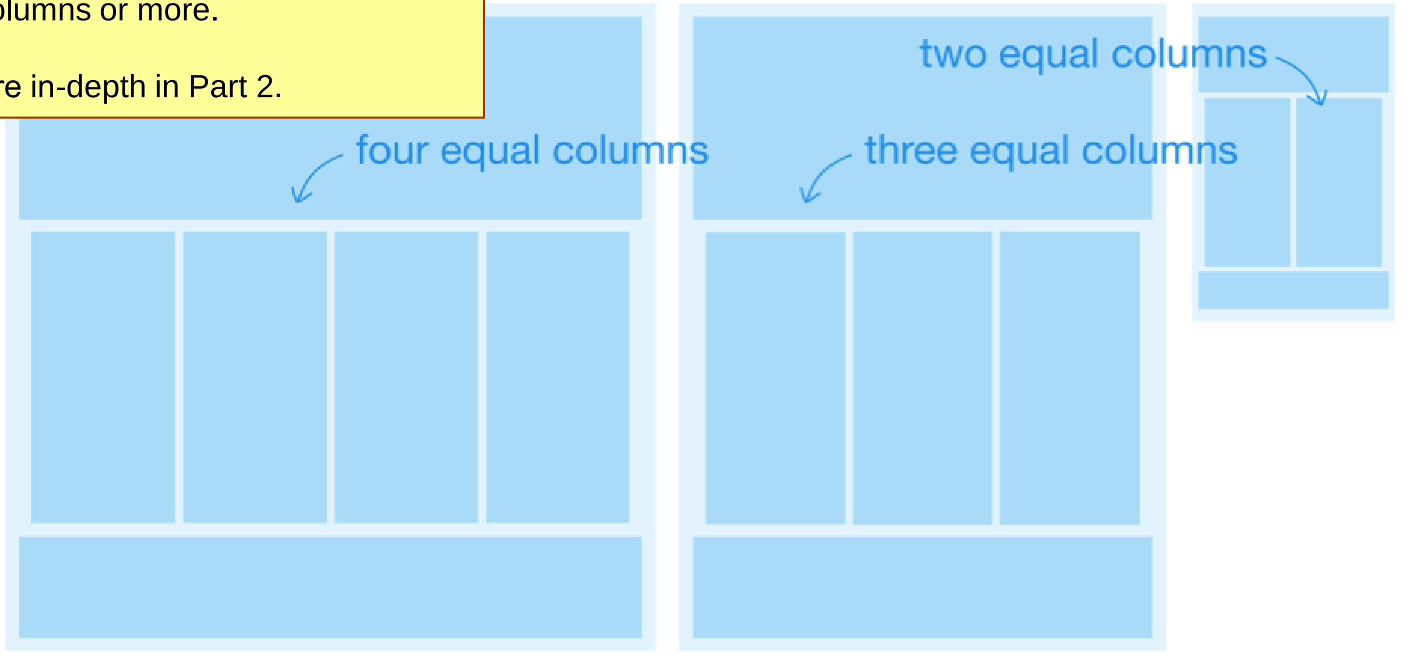
CSS3 already supports multi-column layouts (see Session 2’s slides) and CSS grids, which when combined with media queries can make a web page “adaptive” to different screen sizes.

But in addition to plain CSS, developers may use CSS frameworks (e.g., Bootstrap, etc.) which support easier definition of **column-based grids**, which can “**flow**” freely according to the screen resolution.

While a mobile page should be organized in 1 column, its desktop version can use as many as 16 columns or more.

“Frameworks” will be covered more in-depth in Part 2.



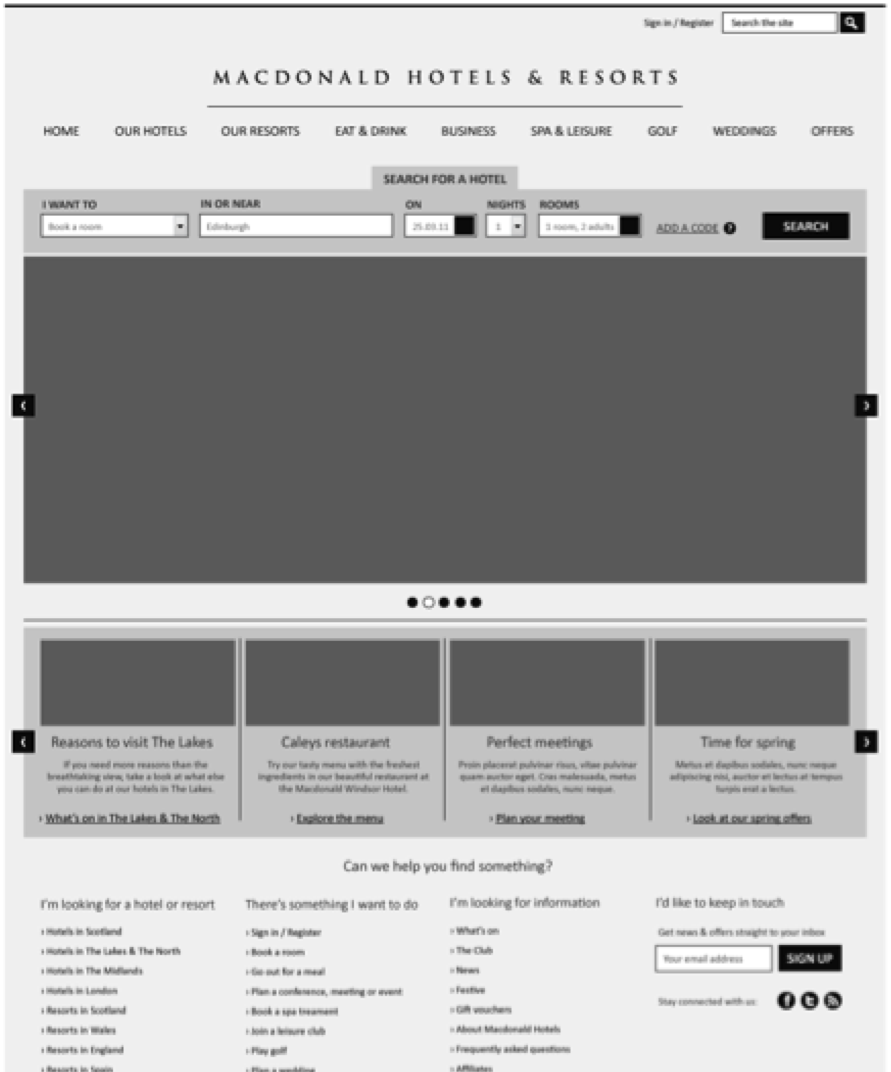


Fluid grids

- Placing contents in “grids” with an appropriate number of columns

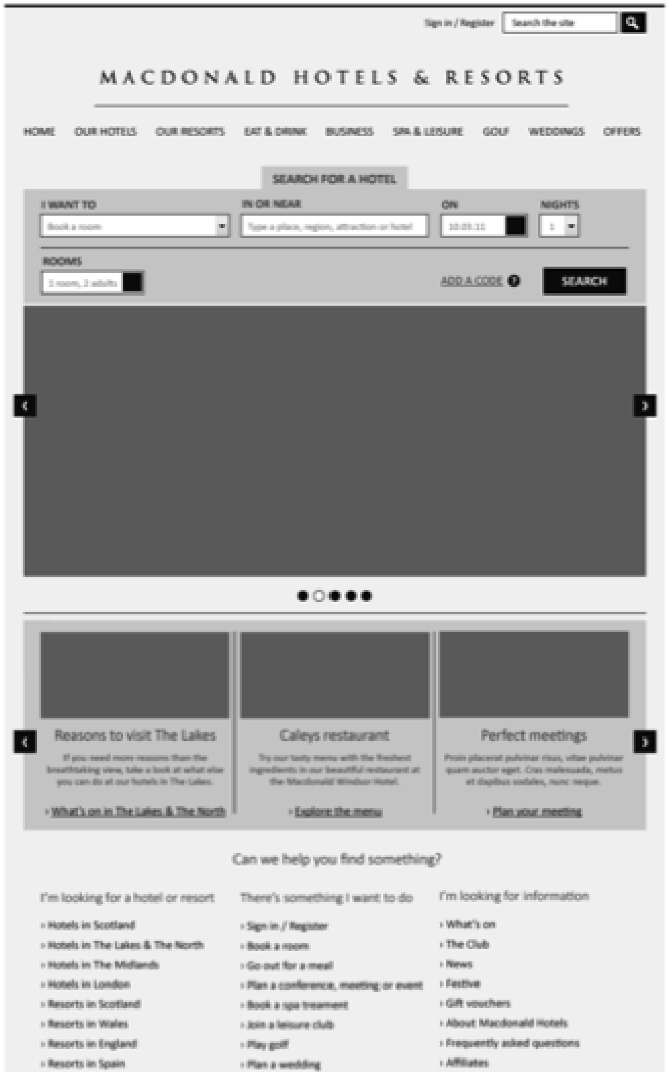
4 columns

Desktop



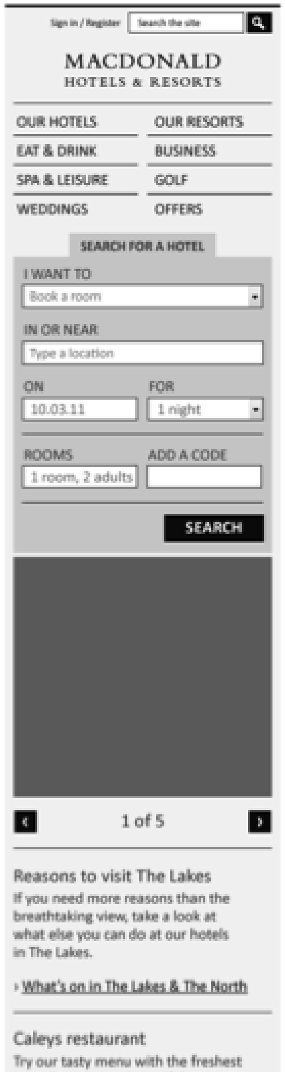
3 columns

Tablet



1 column

Mobile



Design approaches for mobile web

- In general, there are two approaches for designing a website for both desktops and mobile devices:
  - Graceful Degradation; and
  - Progressive Enhancement (PE)
- Traditional approach – graceful degradation (**<u>not</u>** recommended):
  - Focus on building the website for the most advanced/capable browsers (i.e., the desktop version).
  - Mobile devices are expected to have a poor, but “barely acceptable” experience (e.g., by “trimming down” the images, features, contents, etc.).

Progressive enhancements (PE)

Refer to Session 2's HTML 5 section for semantic markup, e.g., <section>, <nav>, <footer>, etc.

- Core principle: **"Mobile First"**
  - Basic content and functionality should be available to all browsers
- Idea: focus on the content (not devices/browsers):
  - First: structure the content in HTML, in a semantic manner
  - Then: apply CSS (media queries, flexible images, fluid grids, etc.) based on different screen sizes
  - Last: apply JS for user interactions in the right contexts for the right browsers
- Advantages:
  - Accessibility: because of the "mobile first" principle, PE pages are by nature more accessible because mobile devices are usually more limited in capabilities
  - SEO: because basic content is always available to search engine robots
- Only possible if "content (HTML)", "presentation (CSS)" and "behavior (JS)" are well separated

Internationalization

Internationalization

- Needs for customizing contents for different geographic regions:
  - Desirable contents and human languages for different people/countries
  - Desirable formats for the user interface (UI)
  - Compliance with local laws
  - Hardware/software concerns
    - E.g., network speed, browsers commonly-used, etc.

- All these are crucial elements for good user experiences

- Internationalization (i18n) is to design a website:
  - without any built-in cultural assumptions
  - that is easy to localize to different countries/regions

- Localization (l10n) is to tailor a website to meet the needs of a particular region, market, or culture

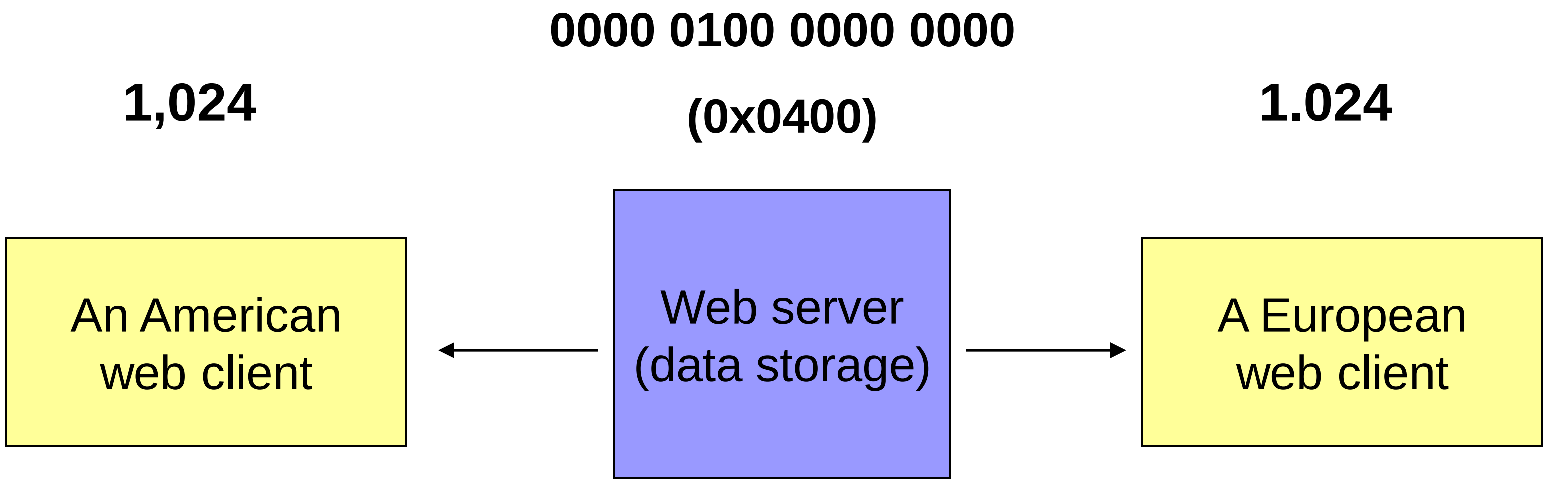
- So, the effort required for globalization (g11n) = i18n + l10n * n (where n is # target regions)

Two main techniques for i18n

- Dynamic selection of logic for:
  - Data presentation
  - Data validation
- Use the universal character set: Unicode

Data presentation

(Example 1: the thousand separator)



```
Value = 0x0400;
Style = American;
result = Format(Value, style)
```

```
Value = 0x0400;
Style = European;
result = Format(Value, style)
```

Separation of data (content) and code (for presentation) is crucial.

Example 2: date formats

Locale	Example	Format
U.S.A.	2/16/15	mdy, /
France	16.2.15	dmy, .
France	16-2-15	dmy, -
CJK	2015/2/16	ymd, /
CJK	2015年2月16日	ymd,年月日
Japan	平成27年2月16日	e¥md,
Japan	27/2/16	¥md, /

Example 3: phone numbers

USA:	+1 (781) 789-1898
France:	33.1.6172.8041
HK:	+852 9876-5432

Example 4: string sorting

English:	ABC...RSTUVWXYZ
German:	AÄB...NOÖ...SßTUÜV…YZ
Swedish/Finnish:	AB...STUVWXYZÅÄÖ
Norwegian:	AB...VWXYÜZÆØÅ...

Spanish: “ch” should sort between “c” & “d”

- Color, Charlar, Dar

Data validation

- Dynamically choose data validation logic based on user’s location or language settings

```
/* choose validation logic */
if (intl = “jp”) then validator = JapaneseDateFormat;
if (intl = “uk”) then validator = EnglishDateFormat;
if (intl = “fr”) then validator = EuropeanDateFormat;

    result = ValidateData(input, validator)
```

- Again, can be done only if data and code are well separated

Implementation strategies

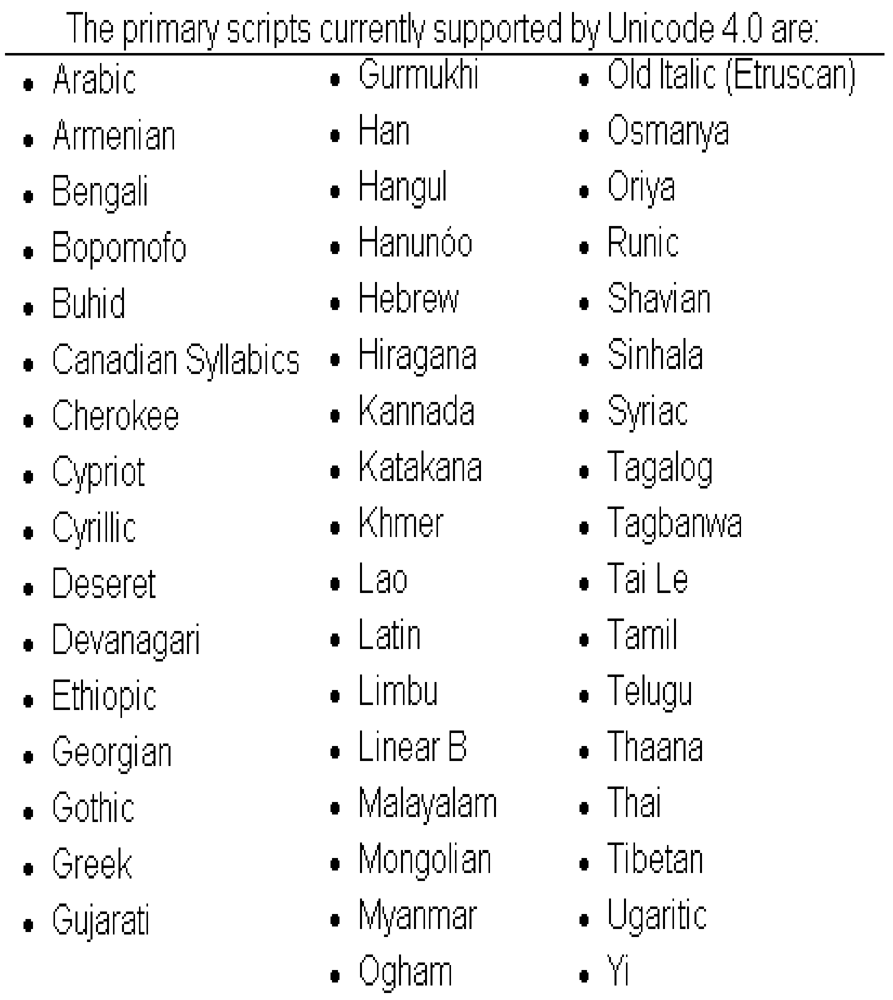
- **Separate** the following:
  - i18n-related data (content)
  - Code for data presentation (for generating output)
  - Code for data validation (for accepting input)

- Make sure that the required code for each target region can function properly
  - Most server-side languages have fairly complete support for i18n/l10n - read their documentation before implementation.
  - E.g., PHP has a very comprehensive “Intl” extension that handles i18n.

- Plan your website for i18n before implementation.
  - Maximize locale-independence in code
    - The above “separations”
  - Consider the need of localization for every UI element (e.g., image, text, user input, etc.)
  - Use Unicode whenever possible for easier expansion

Unicode character standard

- Developed by the Unicode Consortium
- Multilingual
  - Covers all major languages (or character sets)
- Two-way conversions to legacy encodings (e.g., ISO, ASCII…, GB, BIG5, etc.)
- 3 equivalent forms
  - UTF-8: 8-bit variable width (1-4 8-bit words per character)
  - UTF-16: 16-bit, variable width (1-2 16-bit words per character)
  - UTF-32: 32-bit, fixed width (1 32-bit word per character)
- UTF-8 (ASCII compatible) is the most commonly-used Unicode version for the web



The primary scripts currently supported by Unicode 4.0 are:

- Arabic
- Armenian
- Bengali
- Bopomofo
- Buhid
- Canadian Syllabics
- Cherokee
- Cypriot
- Cyrillic
- Deseret
- Devanagari
- Ethiopic
- Georgian
- Gothic
- Greek
- Gujarati
- Gurmukhi
- Han
- Hangul
- Hanunóo
- Hebrew
- Hiragana
- Kannada
- Katakana
- Khmer
- Lao
- Latin
- Limbu
- Linear B
- Malayalam
- Mongolian
- Myanmar
- Ogham
- Old Italic (Etruscan)
- Osmanya
- Oriya
- Runic
- Shavian
- Sinhala
- Syriac
- Tagalog
- Tagbanwa
- Tai Le
- Tamil
- Telugu
- Thaana
- Thai
- Tibetan
- Ugaritic
- Yi

Implementation with Unicode

- Two steps:
  - Add the encoding in HTML header:
    - E.g., <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
  - And store data in Unicode in databases

- Benefits:
  - Single source code can support multilingual data
  - Programmers can know very little about the human languages being processed
  - Tools for handling Unicode are available in all major programming languages

- But: conversion may be required for interfacing with (backend) legacy software/DB

Introduction to Web 2.0 and the related technologies

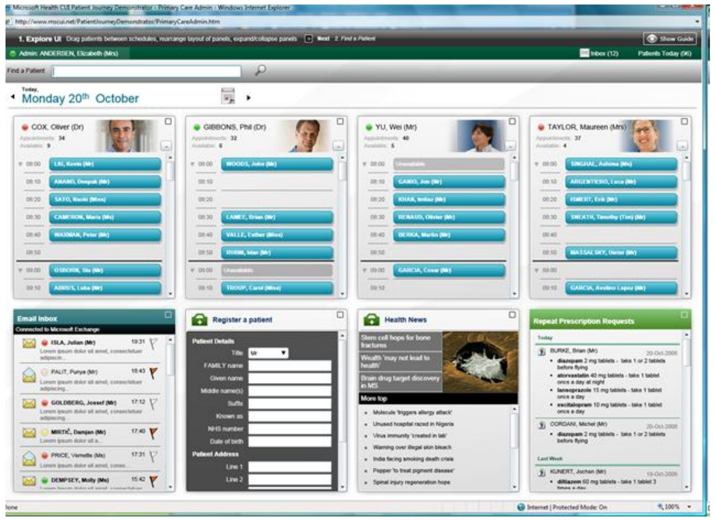
(Topics to be covered in Parts 2-3 of the course)

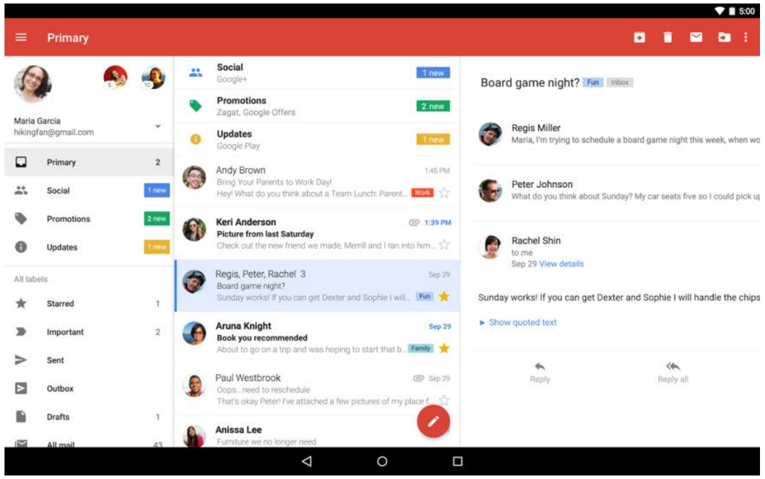
“Web 2.0”

- Evolved from the basic web (or “Web 1.0”) with three trends:
  - Enhanced functionality
  - Socialization
  - Web data consolidation
- Not referring to any specific technology, but a general term for describing the above trends

Trend #1: enhanced functionality

- Web pages are no longer static content. "Rich Internet Applications" (RIA) running in browsers are able to provide desktop applications' functions.
  - E.g., Gmail, Facebook, Twitter, Youtube, Google Docs, etc.
  - Much more complicated and interactive than traditional HTML pages
- Web 2.0: a "read/write" platform (vs. "read-only" in "1.0")





Social Google+ 1 new
Promotions Zagat, Google Offers 2 new
Updates Google Play 1 new
Andy Brown 1:45 PM Bring Your Parents to Work Day! Hey! What do you think about a Team Lunch: Parent... Work
Keri Anderson 1:39 PM Picture from last Saturday Check out the new friend we made, Merrill and I ran into him...
Regis, Peter, Rachel 3 Sep 29 Board game night? Sunday works! If you can get Dexter and Sophie I will... Fun
Aruna Knight Sep 29 Book you recommended About to go on a trip and was hoping to start that b... Family
Paul Westbrook Sep 29 Oops...need to reschedule That's okay Peter! I've attached a few pictures of my place f...
Anissa Lee Furniture we no longer need

Board game night? Fun Inbox
Regis Miller Maria, I'm trying to schedule a board game night this week, when wo
Peter Johnson What do you think about Sunday? My car seats five so I could pick u
Rachel Shin to me Sep 29 View details
Sunday works! If you can get Dexter and Sophie I will handle the chips
Show quoted text

Challenges of supporting more functionalities

- Challenge #1: the web (based on HTML) is “page”-based, i.e., pages have to be reloaded to obtain new content
  - => difficult to simulate the user experiences provided by desktop software
  - Imagine the whole page needs to reload every time when you want to “expand” an email thread in Gmail, or every time you “expand” a menu in Google Docs
- Solution: AJAX (Asynchronous JavaScript and XML)
  - A client-side (JS) technology that enables data transfer “in the background” when a user is reading a page.
  - Individual components in a webpage can be updated (based on the newly-arrived data) without reloading the entire page.
  - No (or much less) page reloads, more similar to desktop applications

Challenges of supporting more functionalities

- Challenge #2: complicated application logic and UI difficult to be implemented with basic web technologies (JS, PHP, etc.)
- Programmers also desire support for higher-levels of abstraction
  - E.g., we want to display a “calendar” instead of “7x5 boxes + 4 round corners”
  - E.g., we want a readily-available authenticate() function rather than handling strings that are user-id’s and passwords
- Solution: Client-side and server-side development libraries and frameworks which support:
  - Much higher-level functionalities (for rapid development); and
  - Better structures of the code (for maintainability)
- Three main types of web development libraries and frameworks:
  - Client-side: e.g., jQuery, Bootstrap, Dojo, etc.
  - Server-side: e.g., Laravel, Ruby on Rails, CakePHP, Struts, Node.js, etc.
  - Special content management systems (CMS) with readily available modules/plugins: e.g., Drupal, Wordpress, Wiki, etc.

Trend #2: socialization

- User-generated contents and user communities
  - Data in web 2.0 is created by individuals instead of the website owners
  - Blogs, photos, videos, tags, bookmarks, etc. – all shared to everyone
  - Examples: Flickr/Instagram/Facebook (photo), Youtube/Facebook (video), WordPress (blog), online forums……

- The same piece of data might be modified by multiple users – collaborations

- Challenge: developing websites for managing user-generated contents, their metadata, user communities and user collaborations can be very tedious

- Solution: use development frameworks that support socialization
  - Content management systems (CMS)
  - Wiki – allows collaborative editing

Trend #3: Web data consolidation

- Two (related) demands:
  - Data reuse and sharing:
    - Use remote data for simplifying website design
      - E.g., news archives/portals, maps, etc.
    - Websites also want to share their own data with the others for attracting traffic
  - Data consolidation: people want to combine existing data/info (available in multiple websites) to form “value-added” contents/services.
    - E.g., Flight schedules + weather info + an archive of local events + note-taking facilities = a travel planner
    - => we want to create “mashups”: a mashup is a combination of data retrieved from multiple websites to form a single integrated web application.

- Challenge: how to retrieve and reuse data across websites?
  - HTML and HTTP are too limited
  - We need higher-level, standard data formats and protocols (or web APIs) for data sharing and consolidation

- Standard data formats
  - E.g., XML, JSON, feeds (e.g., RSS, ATOM, etc.), etc.

- Web API protocols
  - Example protocols: SOAP, RESTful, XML-RPC, JSON-RPC, etc.

Web 2.0: summary

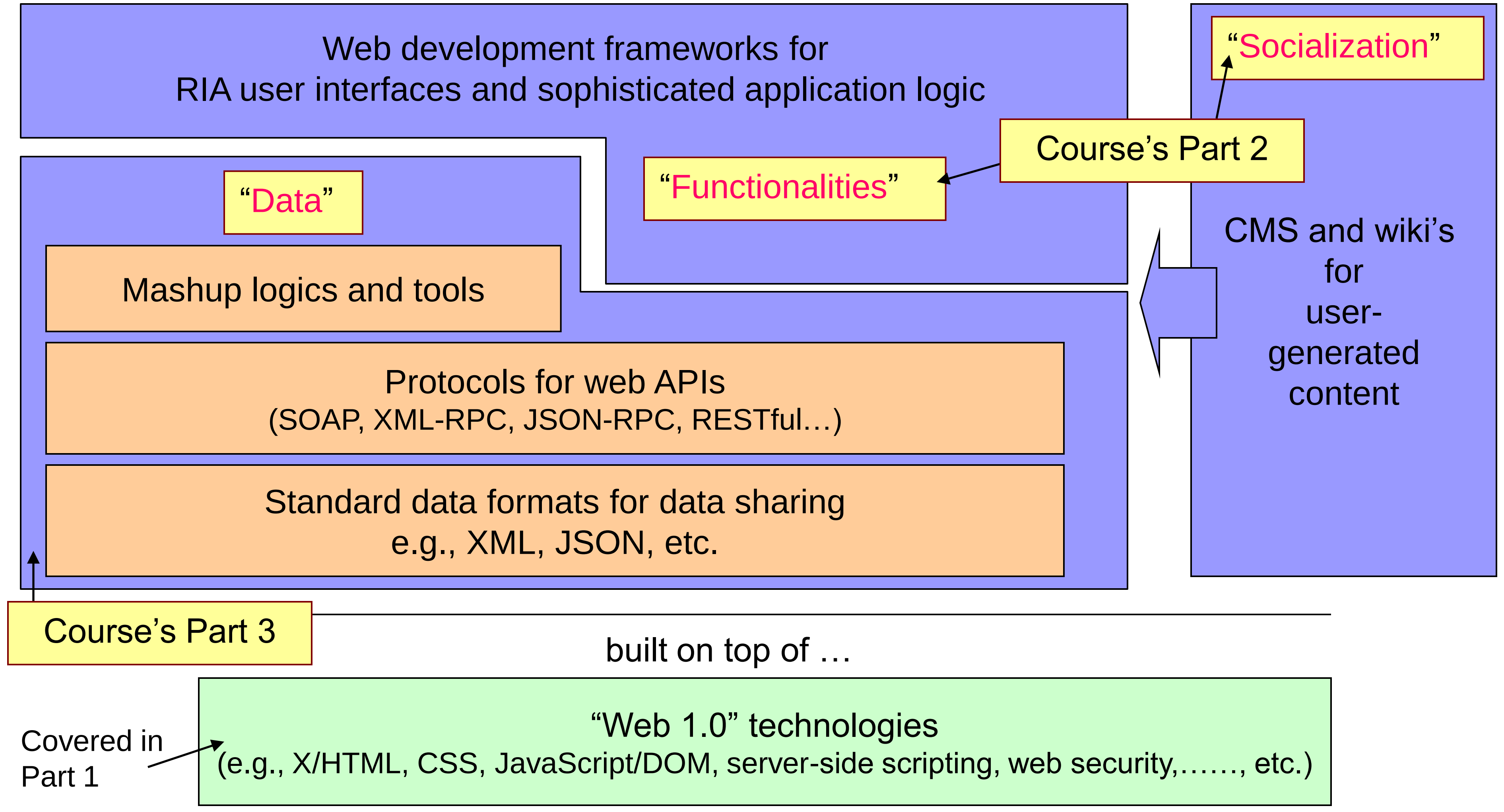
- Functionality: Rich Internet Applications or RIAs with complicated UI and logic
- Socialization: tools for publishing and sharing user-generated data and metadata
- Data: abilities to (re)use remote data to form new information or added value

Simplified by web development frameworks [covered in Part 2 of this course]

Simplified by standard data formats and Web API protocols – also supported by some dev. frameworks [covered in Part 3]

- These three elements can further be combined to form powerful and user-friendly web 2.0 applications.
  - An RIA can be built by fetching data from multiple websites, some of these websites might deliver user-generated data or metadata

Technologies for Web 2.0 (Parts 2-3 of the course)


Web development frameworks for
RIA user interfaces and sophisticated application logic
"Socialization"
Course's Part 2
"Functionalities"
"Data"
Mashup logics and tools
CMS and wiki's for user-generated content
Protocols for web APIs
(SOAP, XML-RPC, JSON-RPC, RESTful…)
Standard data formats for data sharing
e.g., XML, JSON, etc.
Course's Part 3
built on top of …
"Web 1.0" technologies
(e.g., X/HTML, CSS, JavaScript/DOM, server-side scripting, web security,……, etc.)
Covered in Part 1

Summary of Part 1: Web development basics

- The web is shaped by web standards; important standardization bodies include W3C, IETF, WHATWG, etc.

Sessions 1 & 2

Web standards

- Some selection criteria of web technologies --- standard-compliance, open source, and have cross-platform and widely-deployed implementations

Criteria

- X/HTML as the data format for defining the structure of the document
- CSS defines how the document is presented; it separates presentation details from the structure (HTML) of the document
- JavaScript adds client-side logic and alters how X/HTML elements and CSS behaves during run-time
- JavaScript relies on DOM for naming/updating X/HTML elements
- HTML5/CSS3 as the ongoing upgrades of HTML/CSS

Client-side technologies

- **Separation** of {structure (HTML), presentation (CSS), behaviors (JS/DOM)} is crucial

“Separations”!

- Server-side scripting for dynamic contents

Server-side

- Common attacks to websites and ways to avoid them
- Supports for mobile/multiple devices:
  - Techniques for responsive web design (RWD)
- Internationalization and Unicode
  - **Separation** of data and data presentation/validation logic

Other concerns / techniques

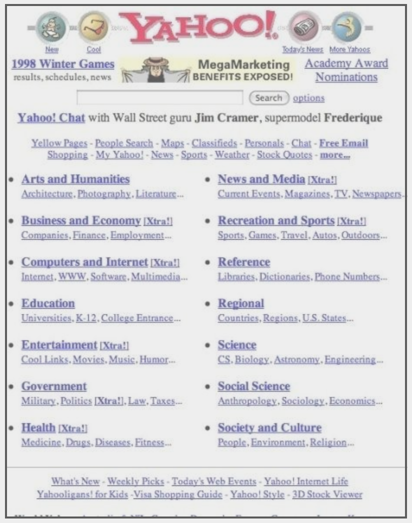
Session 3

- Introduction to Web 2.0

Web 2.0 vs. 1.0

Summary of scope

Part 1 (done):



New · Cool · Today's News · More Yahoos

1998 Winter Games results, schedules, news · MegaMarketing BENEFITS EXPOSED! · Academy Award Nominations

Search options

Yahoo! Chat with Wall Street guru **Jim Cramer**, supermodel **Frederique**

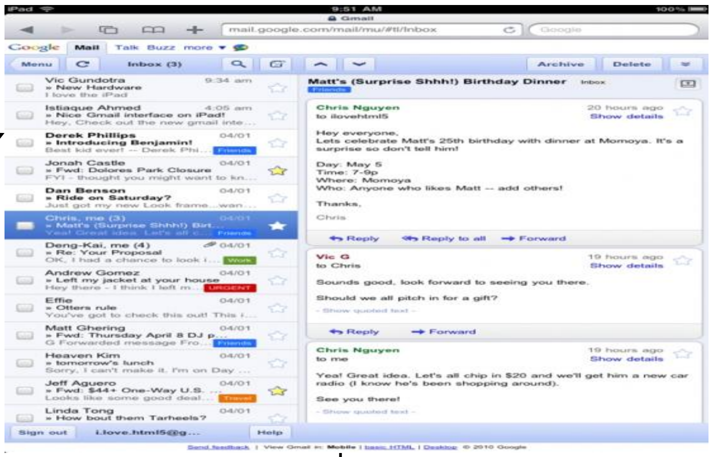
Yellow Pages - People Search - Maps - Classifieds - Personals - Chat - Free Email
Shopping - My Yahoo! - News - Sports - Weather - Stock Quotes - more...

- Arts and Humanities
  Architecture, Photography, Literature...
- News and Media [Xtra!]
  Current Events, Magazines, TV, Newspapers...
- Business and Economy [Xtra!]
  Companies, Finance, Employment...
- Recreation and Sports [Xtra!]
  Sports, Games, Travel, Autos, Outdoors...
- Computers and Internet [Xtra!]
  Internet, WWW, Software, Multimedia...
- Reference
  Libraries, Dictionaries, Phone Numbers...
- Education
  Universities, K-12, College Entrance...
- Regional
  Countries, Regions, U.S. States...
- Entertainment [Xtra!]
  Cool Links, Movies, Music, Humor...
- Science
  CS, Biology, Astronomy, Engineering...
- Government
  Military, Politics [Xtra!], Law, Taxes...
- Social Science
  Anthropology, Sociology, Economics...
- Health [Xtra!]
  Medicine, Drugs, Diseases, Fitness...
- Society and Culture
  People, Environment, Religion...

What's New - Weekly Picks - Today's Web Events - Yahoo! Internet Life
Yahooligans! for Kids - Visa Shopping Guide - Yahoo! Style - 3D Stock Viewer

Websites becoming "web apps" -> more sophisticated, lots of interactions with users, "Web 2.0", etc.

Part 2 (lectures 4-6):



Vic Gundotra 9:34 am » New Hardware I love the iPad
Istiaque Ahmed 4:05 am » Nice Gmail interface on iPad! Hey, Check out the new gmail inte...
Derek Phillips 04/01 » Introducing Benjamin! Best kid ever! -- Derek Phi... Friends
Jonah Castle 04/01 » Fwd: Dolores Park Closure FYI - thought you might want to kn...
Dan Benson 04/01 » Ride on Saturday? Just got my new Look frame...wan...
Chris, me (3) 04/01 » Matt's (Surprise Shhh!) Birt... Yea! Great idea. Let's all c... Friends
Deng-Kai, me (4) 04/01 » Re: Your Proposal OK, I had a chance to look i... Work
Andrew Gomez 04/01 » Left my jacket at your house Hey there - I think I left m... URGENT
Effie 04/01 » Otters rule You've got to check this out! This i...
Matt Ghering 04/01 » Fwd: Thursday April 8 DJ p... G Forwarded message Fro... Friends
Heaven Kim 04/01 » tomorrow's lunch Sorry, I can't make it. I'm on Day ...
Jeff Aguero 04/01 » Fwd: $44+ One-Way U.S. ... Looks like some good deal... Travel
Linda Tong 04/01 » How bout them Tarheels?

Matt's (Surprise Shhh!) Birthday Dinner Inbox Friends

Chris Nguyen to ilovehtml5 — 20 hours ago Show details
Hey everyone,
Lets celebrate Matt's 25th birthday with dinner at Momoya. It's a surprise so don't tell him!
Day: May 5
Time: 7-9p
Where: Momoya
Who: Anyone who likes Matt -- add others!
Thanks,
Chris
Reply · Reply to all · Forward

Vic G to Chris — 19 hours ago Show details
Sounds good, look forward to seeing you there.
Should we all pitch in for a gift?
- Show quoted text -
Reply · Forward

Chris Nguyen to me — 19 hours ago Show details
Yea! Great idea. Let's all chip in $20 and we'll get him a new car radio (I know he's been shopping around).
See you there!
- Show quoted text -

Send feedback | View Gmail in: Mobile | basic HTML | Desktop © 2010 Google

Integration and interoperability issues -> how to reuse existing & remote data?

Part 3 (lectures 7-9):

YouTube, Gmail, Amazon, online databases, Maps, updated event lists, YOUR websites, ………

The web/cloud(s)

Part 4 (lecture 10):

Optimizations

You have a great website, how to make it loaded fast at users' computers, and most important… popular?

Post-class self-learning resources

- W3C Mobile Web Best Practices and Mobile Web Application Best Practices
  - http://www.w3.org/TR/mobile-bp/
  - http://www.w3.org/TR/mwabp/

- W3Schools.com's tutorials on RWD with HTML and CSS
  - https://www.w3schools.com/html/html_responsive.asp
  - https://www.w3schools.com/css/css_rwd_intro.asp

- References
  - An extensive collection of cheatsheets for web developers
    - Covering HTML, CSS, PHP, etc.
    - https://medium.com/level-up-web/the-best-cheat-sheets-guides-docs-for-web-designers-and-web-developers-8e335a0aad77

  - An e-book (available through HKU library) on web security:
    - H. Wu and L. Zhao. Web security: a WhiteHat perspective. CRC Press. 2014.
      - http://find.lib.hku.hk/record=HKU_IZ51488843860003414