ELECS425F Computer and Network Security

Lec 08

Web Security - Part One

URL and HTTP

URL (Uniform Resource Locators)

All web resources are addressed with the use of uniform resource locators.

Syntax = `scheme://domain:port/path?query_string#fragment_id`

Example:
`https://www.chsc.hk/ssp2018/sch_detail.php?lang_id=2&sch_id=450`

Every HTTP URL consists of the following. Other schemes also share this general format with some variation.

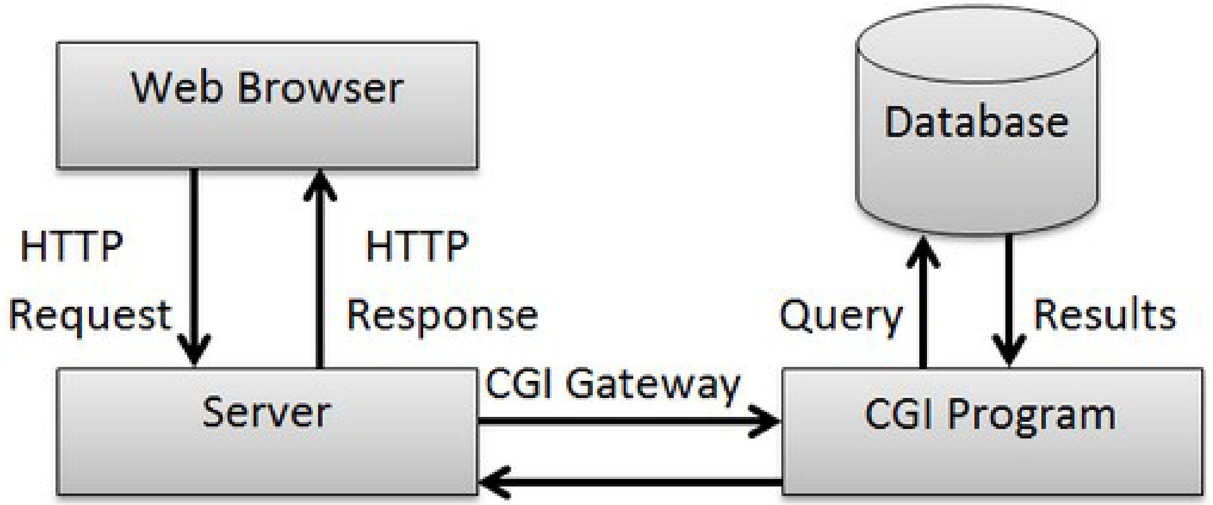
1) the scheme name (commonly called protocol)
2) a colon and two slashes
3) a host, normally given as a domain name but sometimes as a literal IP address
4) optionally a port number
5) the full path of the resource

The scheme says how to connect, the host specifies where to connect, and the remainder specifies what to ask for.

Class discussion: can you give some examples of specific schemes in URL?

Example: `ws://plbpc013.ouhk.edu.hk:8080/websocketserver/server.php`

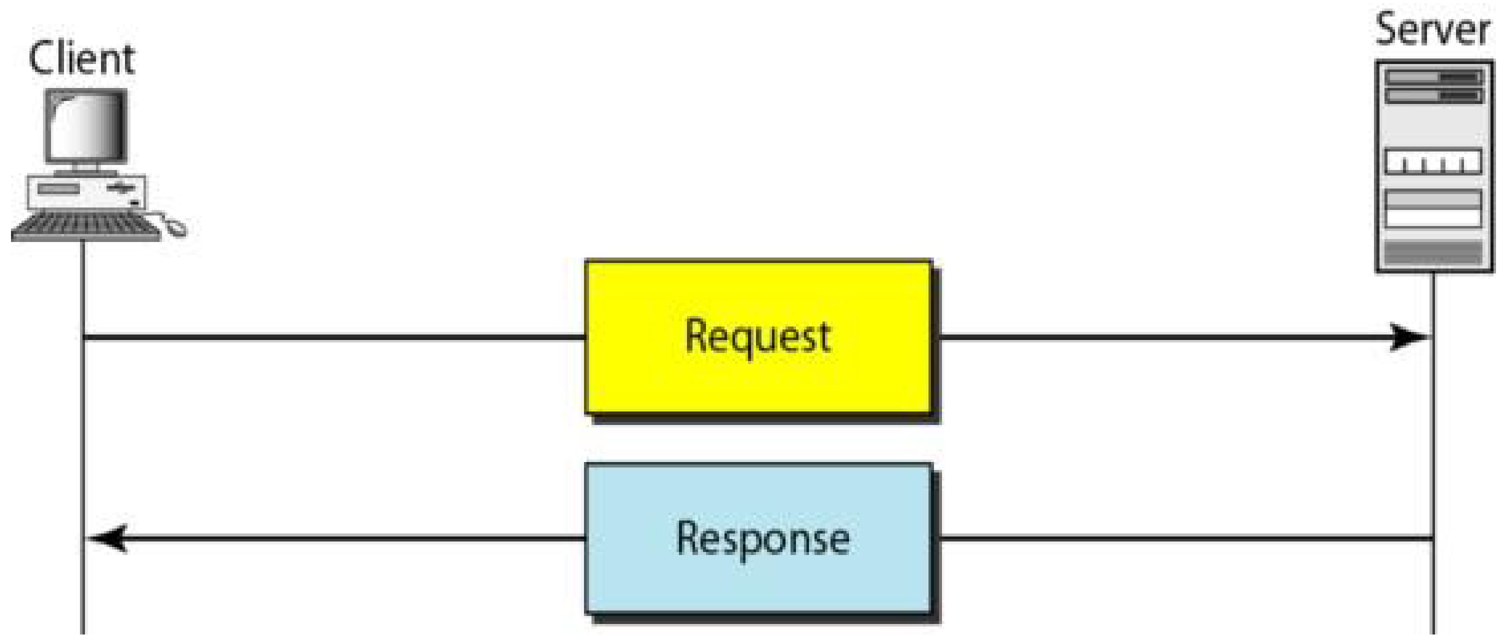
For programs such as Common Gateway Interface (CGI) scripts, this is followed by a query string, and an optional fragment identifier.



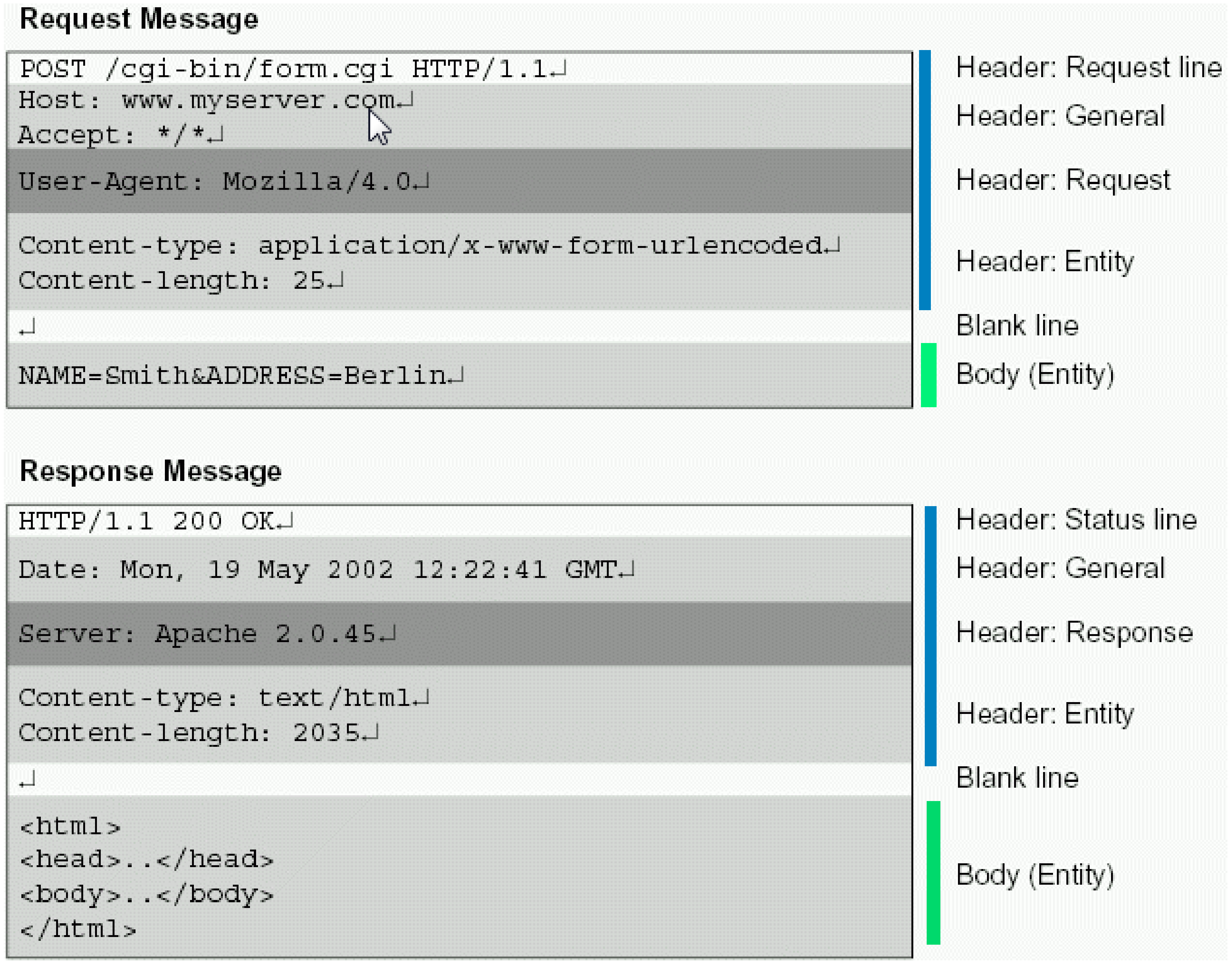
URLs do not have any particular character set defined by relevant RFCs.
Demo: copy a URL that contains Chinese characters and paste into an editor.
Example: https://www.w3schools.com/tags/ref_urlencode.asp

HTTP (Hypertext Transfer Protocol)

HTTP is the network protocol used to deliver virtually all resources on the web. Example of a HTTP transaction:



HTTP Request/Response and Header Structure



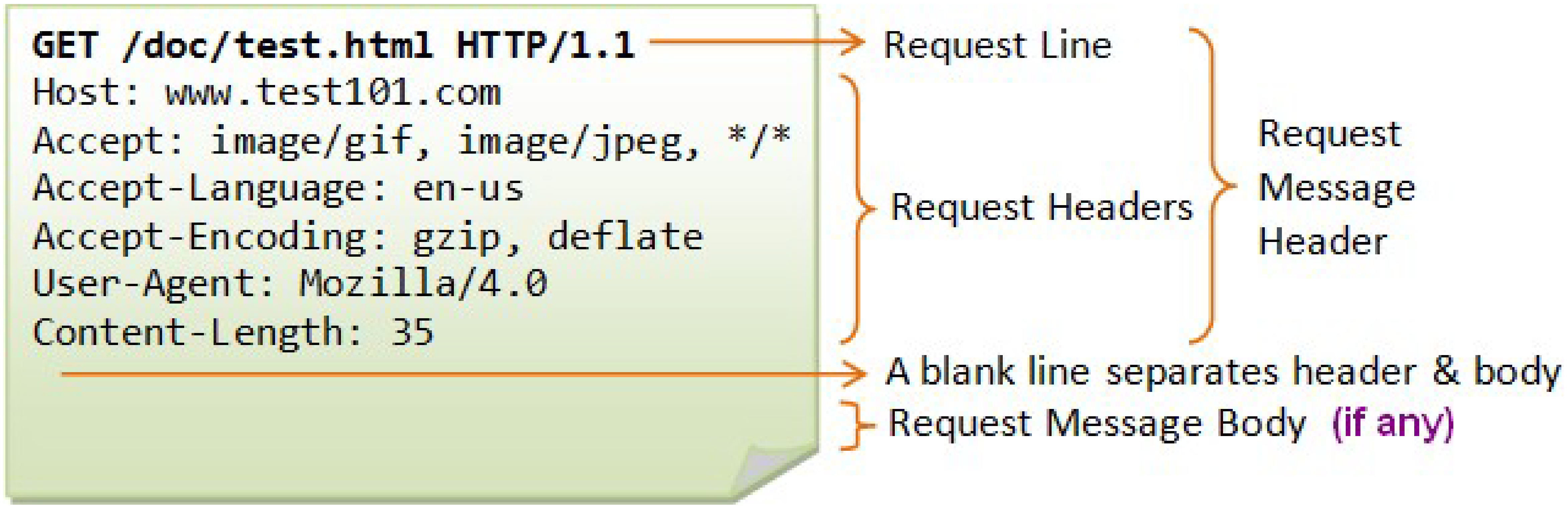
Request Message

```
POST /cgi-bin/form.cgi HTTP/1.1↵
Host: www.myserver.com↵
Accept: */*↵
User-Agent: Mozilla/4.0↵
Content-type: application/x-www-form-urlencoded↵
Content-length: 25↵
↵
NAME=Smith&ADDRESS=Berlin↵
```

Response Message

```
HTTP/1.1 200 OK↵
Date: Mon, 19 May 2002 12:22:41 GMT↵
Server: Apache 2.0.45↵
Content-type: text/html↵
Content-length: 2035↵
↵
<html>
<head>..</head>
<body>..</body>
</html>
```

HTTP Request Message Example

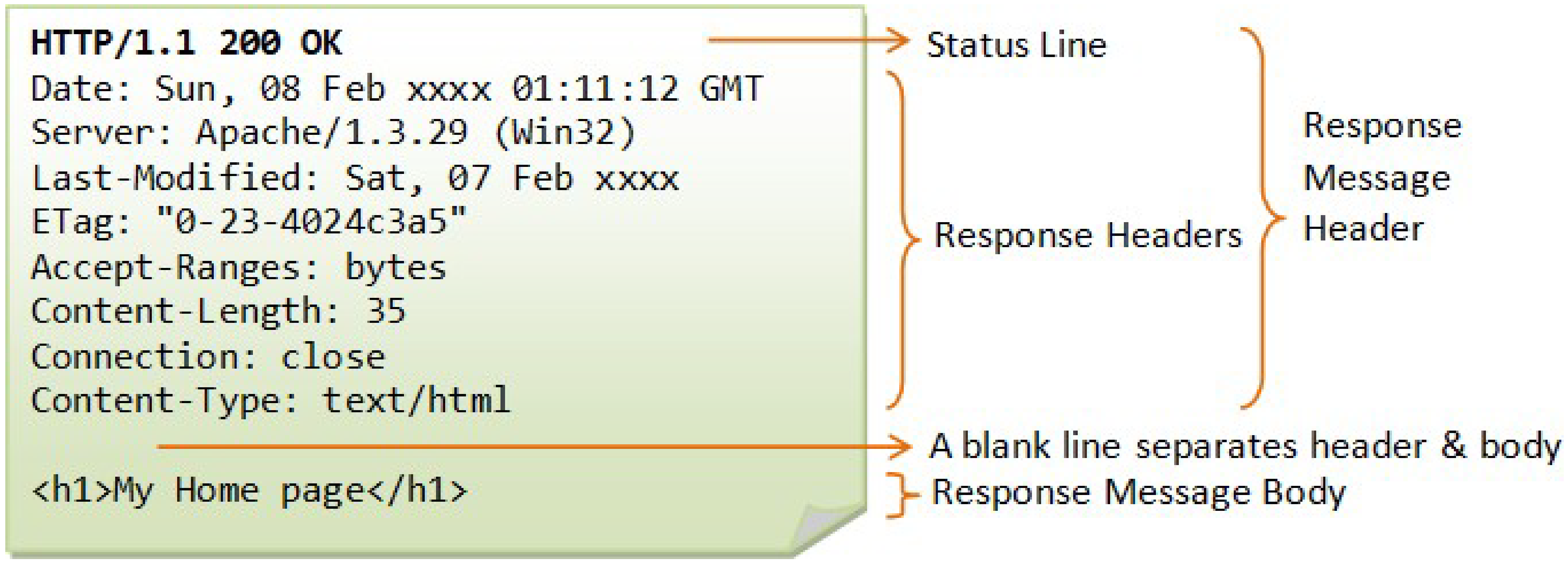


```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

```

Request Line
Request Headers
Request Message Header
A blank line separates header & body
Request Message Body (if any)

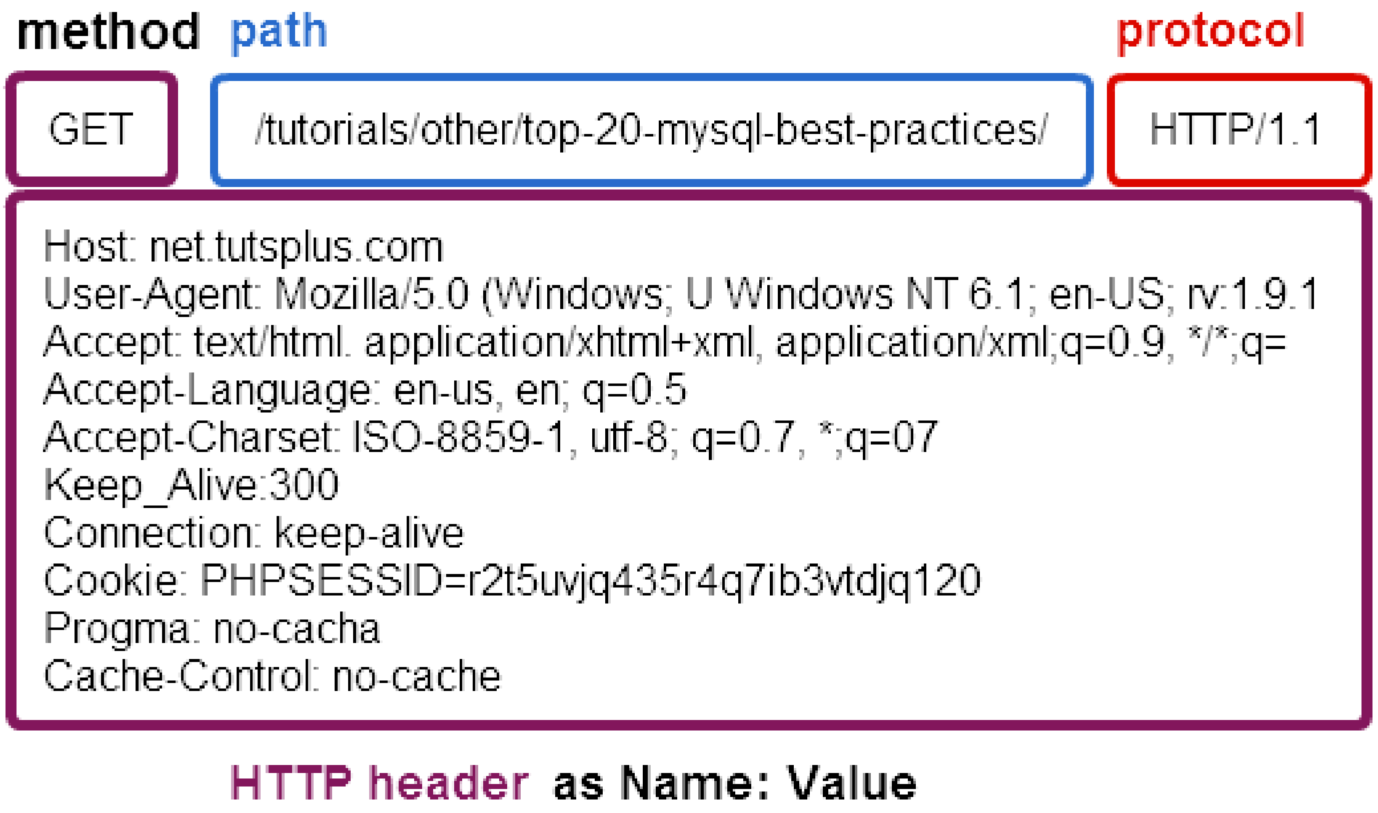
HTTP Response Message Example



```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1>My Home page</h1>
```

HTTP Request Structure



GET /tutorials/other/top-20-mysql-best-practices/ HTTP/1.1

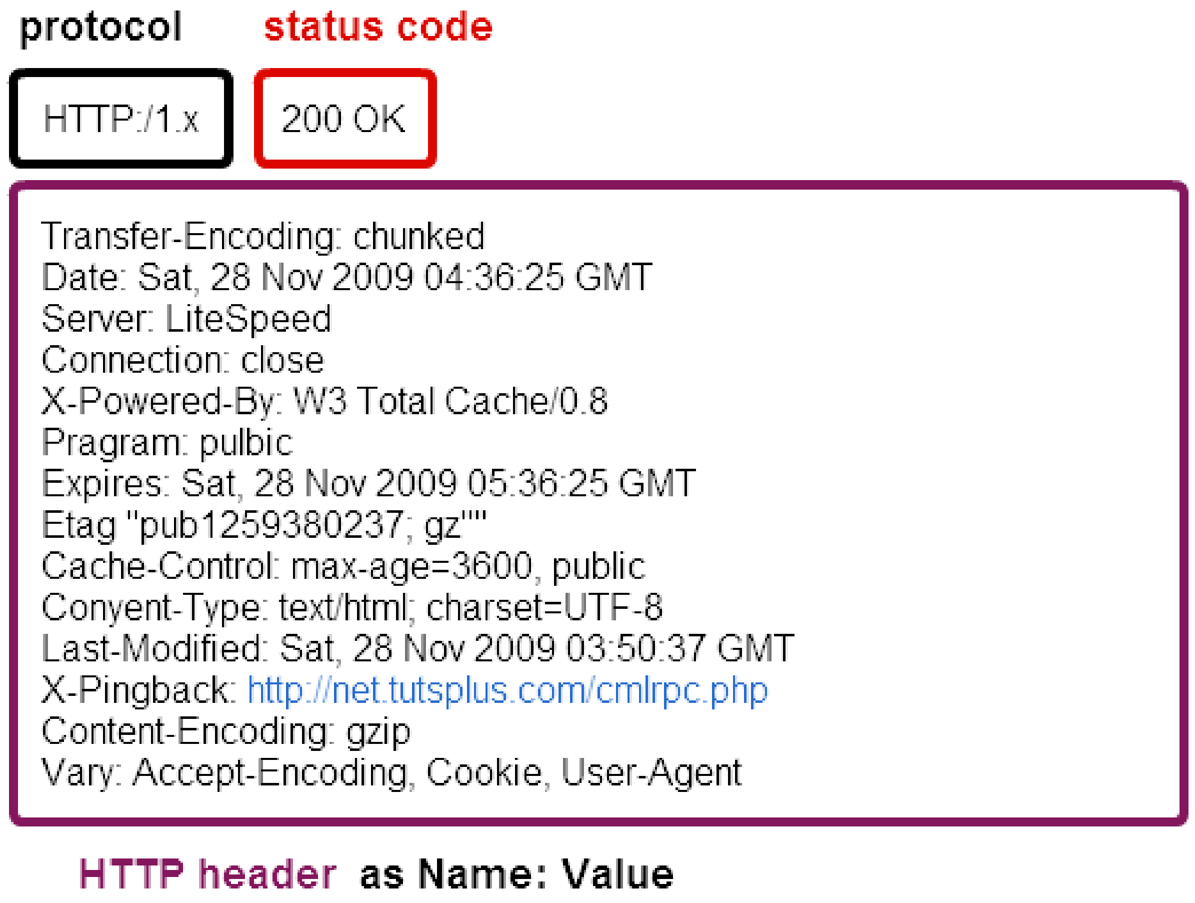
```
Host: net.tutsplus.com
User-Agent: Mozilla/5.0 (Windows; U Windows NT 6.1; en-US; rv:1.9.1
Accept: text/html. application/xhtml+xml, application/xml;q=0.9, */*;q=
Accept-Language: en-us, en; q=0.5
Accept-Charset: ISO-8859-1, utf-8; q=0.7, *;q=07
Keep_Alive:300
Connection: keep-alive
Cookie: PHPSESSID=r2t5uvjq435r4q7ib3vtdjq120
Progma: no-cacha
Cache-Control: no-cache
```

Request line = request method + path + protocol

Request methods: indicates what kind of request this is; most common methods are GET and POST. HTTP

headers: name-value pairs

HTTP Response Structure



```
HTTP:/1.x 200 OK
Transfer-Encoding: chunked
Date: Sat, 28 Nov 2009 04:36:25 GMT
Server: LiteSpeed
Connection: close
X-Powered-By: W3 Total Cache/0.8
Pragram: pulbic
Expires: Sat, 28 Nov 2009 05:36:25 GMT
Etag "pub1259380237; gz""
Cache-Control: max-age=3600, public
Conyent-Type: text/html; charset=UTF-8
Last-Modified: Sat, 28 Nov 2009 03:50:37 GMT
X-Pingback: http://net.tutsplus.com/cmlrpc.php
Content-Encoding: gzip
Vary: Accept-Encoding, Cookie, User-Agent
```

Response line = protocol + status code

HTTP headers: name-value pairs

The status code is a three-digit integer

HTTP Status Code

1XX = Informational
2XX = Successful
3XX = Redirection
4XX = Client Error
5XX = Server Error

Reference: https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

HTTP Headers

HTTP message headers are used to precisely describe the resource being fetched or the behavior of the server or the client. Custom proprietary headers can be added using the 'X-' prefix; others are listed in an IANA registry, whose original content was defined in RFC 4229. IANA also maintain a registry of proposed new HTTP message headers.

The web page summaries the headers and their usage: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

Burp Proxy (or other HTTP proxies) is a good tool to learn more about HTTP and the working of a particular website.

Sessions and Cookie

- HTTP is stateless protocol; cookies add state. Cookies are used to store state on user's machine.

- How cookies are passed in HTTP

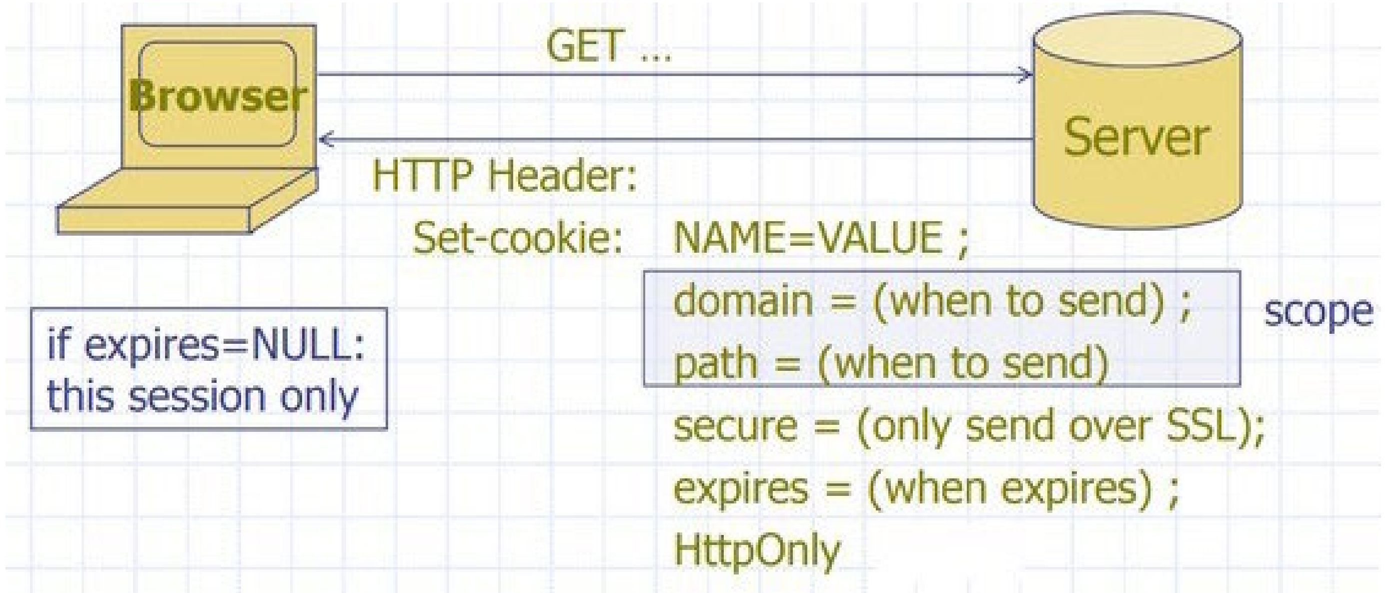
  Cookies are passed as HTTP headers, both in the request (client to server), and in the response (server to client).

- How cookies are set and sent to the client side?

```
Set-Cookie: value[; expires=date][; domain=domain][; path=path][; secure]
```

- How cookies are sent to the server side?

```
Cookie: name=value
```


Browser
GET ...
Server
HTTP Header:
Set-cookie: NAME=VALUE ;
domain = (when to send) ;
path = (when to send)
scope
secure = (only send over SSL);
expires = (when expires) ;
HttpOnly
if expires=NULL: this session only

Examples: https://hk.dictionary.search.yahoo.com/

- In an HTTP Response:

```
set-cookie: B=00q3nlle74jr6&b=3&s=a3; expires=Mon, 24-Feb-2020 08:01:42
GMT; path=/;  domain=.yahoo.com
```

- In an HTTP Request:

```
cookie: cookie: B=00q3nlle74jr6&b=3&s=a3; GUCS=AVkJktzf;
```

- Browser will store at most 50 cookies per domain or 4 KB per domain.

Uses of cookies

1) user authentication
2) personalisation
3) user tracking

Additional flags in setting cookies

Example: (in an HTTP response)

```
  Set-Cookie: id=a3fWa; Expires=Fri, 22 Nov 2019 07:28:00 GMT; Secure; HttpOnly
```

1) Secure Cookies

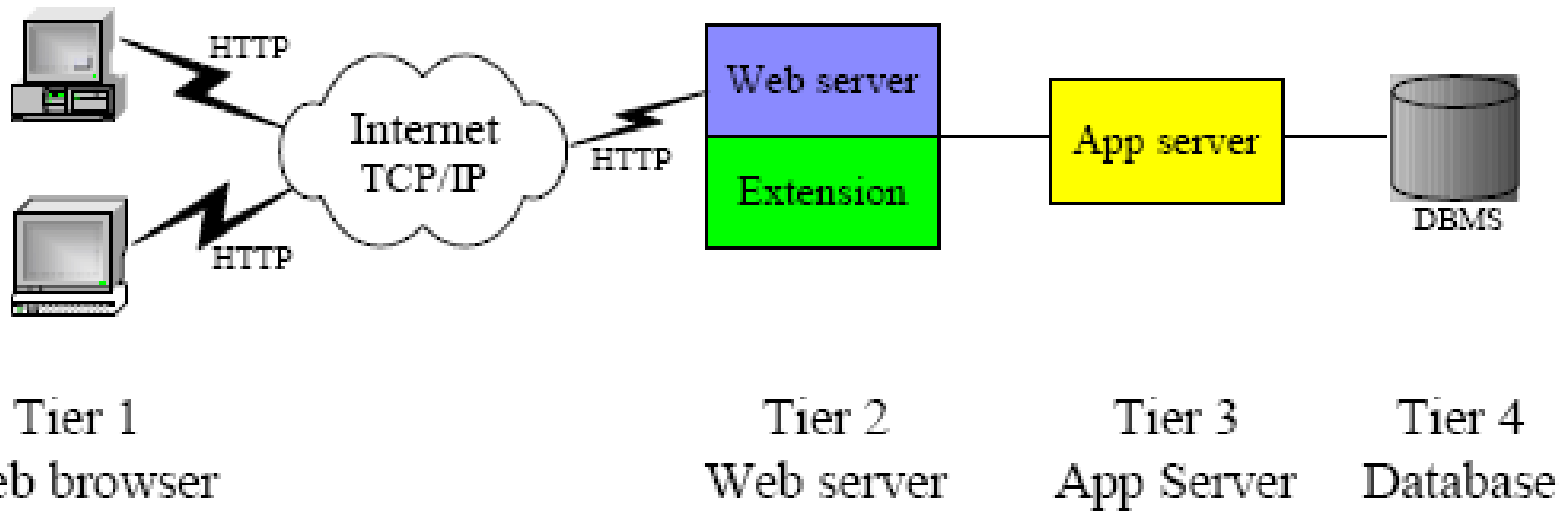
The **secure flag** is an option that can be set by the application server when sending a new cookie to the user within an HTTP Response. The purpose of the secure flag is to prevent cookies from being observed by unauthorized parties due to the transmission of the cookie in clear text. To accomplish this goal, browsers which support the secure flag will only send cookies with the secure flag when the request is going to a HTTPS page. Said in another way, the browser will not send a cookie with the secure flag set over an unencrypted HTTP request.

2) HTTP Only Cookies

HttpOnly is an additional flag included in a Set-Cookie HTTP response header. Using the HttpOnly flag when generating a cookie helps mitigate the risk of client side script accessing the protected cookie (if the browser supports it). If the HttpOnly flag (optional) is included in the HTTP response header, the cookie cannot be accessed through client side script.

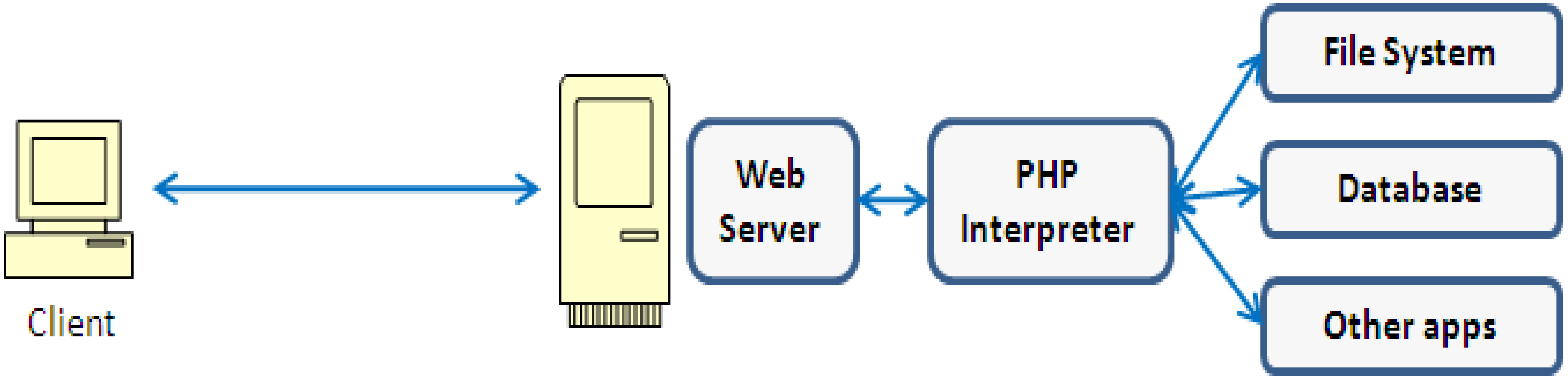
Server-side Logic

A typical web application architecture involves four tiers



- An application (written in a mixture of PHP, Java, Perl, Python, C, ASP) runs on a web server or application server
- It takes input from remote users.
- It interacts with back-end databases and third parties.
- It prepares and outputs results for users.
  - It will dynamically generate web pages.
  - It can obtain inputs/content from many different sources, including from users themselves.

An example architecture of web based applications



PHP

It is a server scripting language with C-like syntax.

It can get data inputted by users via web form in global arrays `$_GET`, `$_POST`.

It can output values in variables in HTML.

Example: `<h1>Hello <?php echo $username; ?></h1>`

SQL

It is a widely used database query language.

It can fetch a set of records.

```
SELECT * FROM Person WHERE Username=‘Vitaly’;
```

It can add data to a database table.

```
INSERT INTO Key (Username, Key) VALUES (‘Vitaly’, 3611BBFF);
```

It can modify data in a database table.

```
UPDATE Keys SET Key=FA33452D WHERE PersonID=5;
```

A Case Study:

Coding: https://tutorialzine.com/2010/08/ajax-suggest-vote-jquery-php-mysql

Some sample code:

```
$selecteduser = $_GET['user'];
$sql = "SELECT Username, Key FROM Key WHERE Username='$selecteduser'";
$rs = $db->executeQuery($sql);
```

Browser-side Logic

Rendering and Events

- **Document Object Model (DOM)**

  DOM is an object-based representation of HTML document layout.

    DOM is a convention for representing and interacting with objects in HTML.
    Objects in the DOM tree may be addressed and manipulated by using methods on the objects.

  HTML document needs to be programatically accessible and modifiable on the fly in response to user actions.

  DOM emerged as the prevailing method for accomplishing this task.

- **Browser-side Javascript**

  JavaScript is a relatively simple but rich, object-based imperative scripting language tightly integrated with HTML, and supported by all contemporary web browsers.

  Browser-side JavaScript is invoked from within HTML documents in the following ways:

  1) Standalone <SCRIPT> tags that enclose code blocks,

  2) Event handlers tied to HTML tags (e.g. onmouseover="..."),

  3) Special URL schemes specified as targets for certain resources or actions (javascript:...) - in HTML and in stylesheets.

- **Inclusion of resources from other URL in HTML**

  Web pages allow inclusion of resources from any place on the web. There are

  several ways to include resources from other URLs in HTML:

1) Using an <img> tag to include an image.
2) Using a <link> tag to include a style sheet.
3) Using a <script> tag to include a JavaScript file.
4) Using an <object> or <embed> tag to include media files.
5) Using an <iframe> tag to include another HTML file.

```
<iframe src="hello.html" width=450 height=100>
If you can see this, your browser doesn't understand IFRAME.
</iframe>
```

Browser Security Feature: Isolation

- **Security Policy Goals**
  1) Safe to visit an evil web site
  2) Safe to visit two pages at the same time
  3) Allow safe delegation

- **Same-origin policy**

Definition of an origin

Two pages have the same origin if the protocol, port and host are the same for both pages.

Example URL: http://store.company.com/dir/page.html

http://store.company.com/dir2/other.html Success

http://store.company.com/dir/inner/another.html Success

https://store.company.com/secure.html Failure (Different protocol)

http://store.company.com:81/dir/etc.html Failure (Different port)

http://news.company.com/dir/other.html Failure (Different host)

The same-origin policy

It is a policy that restricts how a document or script loaded from one origin can interact with a resource from another origin. It is a critical security mechanism for isolating potentially malicious documents.

- **Same-origin policy (SOP) for network access**

The same-origin policy controls interactions between two different origins, such as when you use XMLHttpRequest or an <img> element. These interactions are typically placed in three categories:

- Cross-origin writes are typically allowed. Examples are links, redirects and form submissions.
- Cross-origin embedding is typically allowed.
- Cross-origin reads are typically not allowed.

Examples of resources which may be embedded cross-origin:

JavaScript with <script src="..."></script>.

CSS with <link rel="stylesheet" href="...">. Images with <img>.

Media files with <video> and <audio>.

Plug-ins with <object>, <embed> and <applet>.

Anything with <frame> and <iframe>. A site can use the X-Frame-Options header to prevent this form of cross-origin interaction.

In HTTP response headers, there are three possible directives for X-Frame-Options:

```
X-Frame-Options: DENY
X-Frame-Options: SAMEORIGIN
X-Frame-Options: ALLOW-FROM https://example.com/
```

Example: Same-origin policy for XMLHttpRequest

The same origin policy basically means that if your AJAX call is from a page hosted on www.mysite.com, you can't make a call to www.othersite.com.

- **Same-origin policy (SOP) for cookies**

SOP for writing cookies

Which cookies can be set by login.site.com?

Allowed domains: login.site.com, .site.com
Disallowed domains: user.site.com, othersite.com

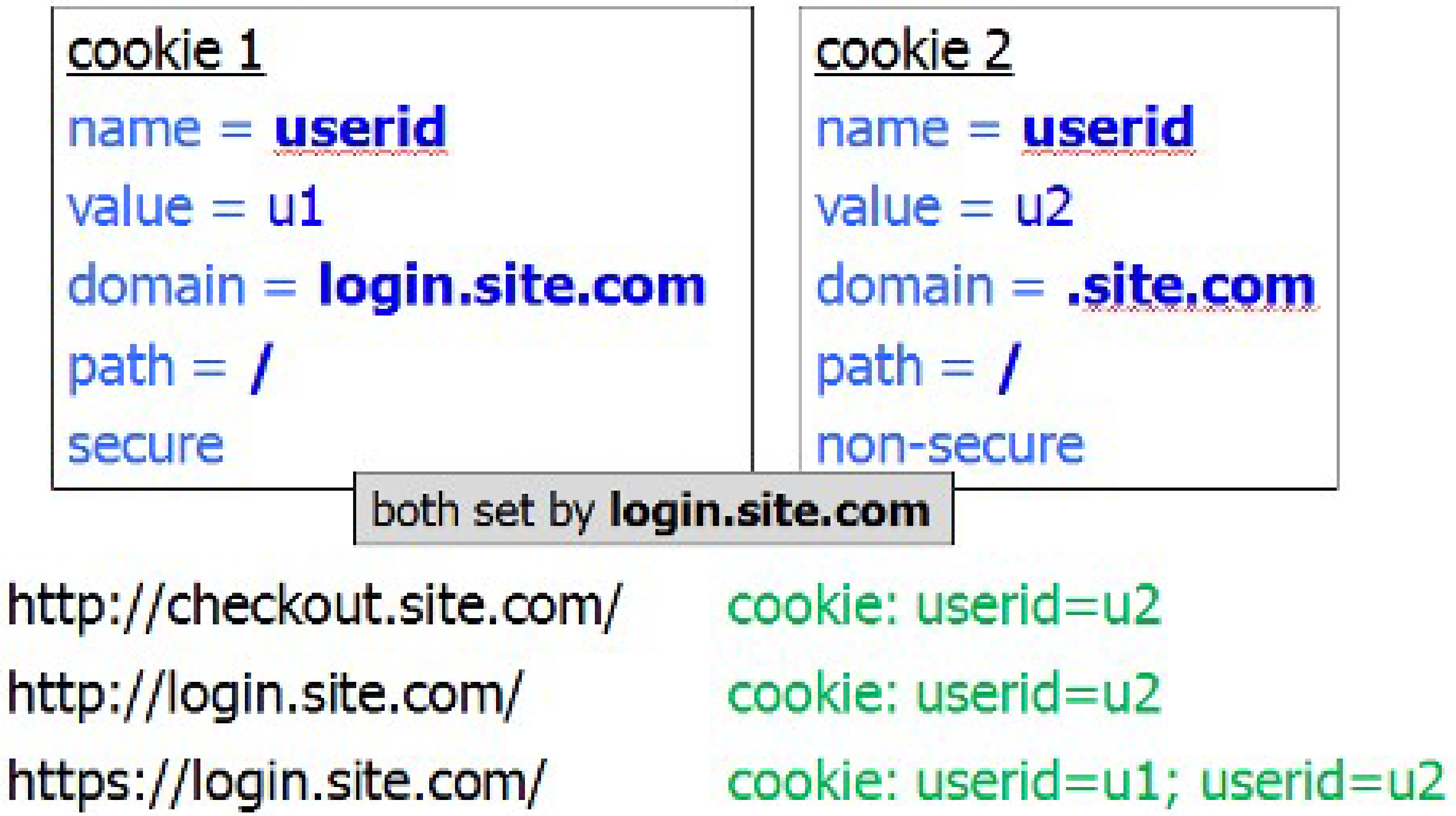
login.site.com can set cookies for all of .site.com but not for another site or TLD

SOP for reading cookies

Browser sends all cookies in URL scope:

1. cookie-domain is domain-suffix of URL-domain
2. cookie-path is prefix of URL-path
3. protocol=HTTPS if cookie is "secure"

Examples of Cookie Reading SOP



cookie 1	cookie 2
name = **userid**	name = **userid**
value = u1	value = u2
domain = **login.site.com**	domain = **.site.com**
path = **/**	path = **/**
secure	non-secure

both set by **login.site.com**

http://checkout.site.com/	cookie: userid=u2
http://login.site.com/	cookie: userid=u2
https://login.site.com/	cookie: userid=u1; userid=u2

References

HTTP/1.1 Status Code Definitions

HTTP codes as Valentine’s Day comics

HTTP headers

Same-Origin Policy

Cookie Same Origin Policy by Dan Boneh - PDF file from standford.edu

Cross-Origin Resource Sharing (CORS) - Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism that allows a server to indicate any other origins (domain, scheme, or port) than its own from which a browser should permit loading of resources.