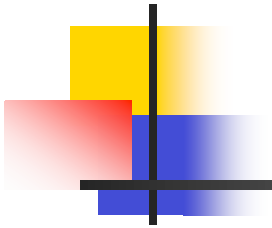
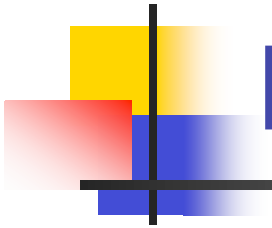


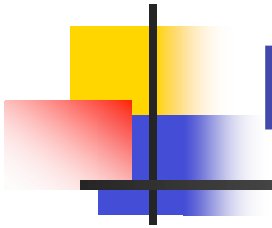
Web Clients and Servers

(Week 1 Lecture 2)



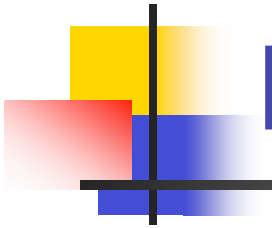
Lecture Objectives

- Understand the concepts involved in Web Serving
- Preparation for practical work on web server management
  - Acquire an understanding in the workings of a web server, in preparation for the nuts-and-bolts of managing a web server we will go through in lectures next week, and labs the week after.



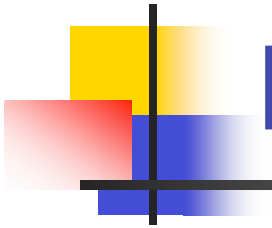
Lecture Objectives

- Relevance to unit objectives:
  - Learning objective 1: Understanding Technologies
- Relevance to assessments:
  - Introduce some of the foundation material required for
    - labs in week 2-5
    - Assignment 1.



Lecture Outline

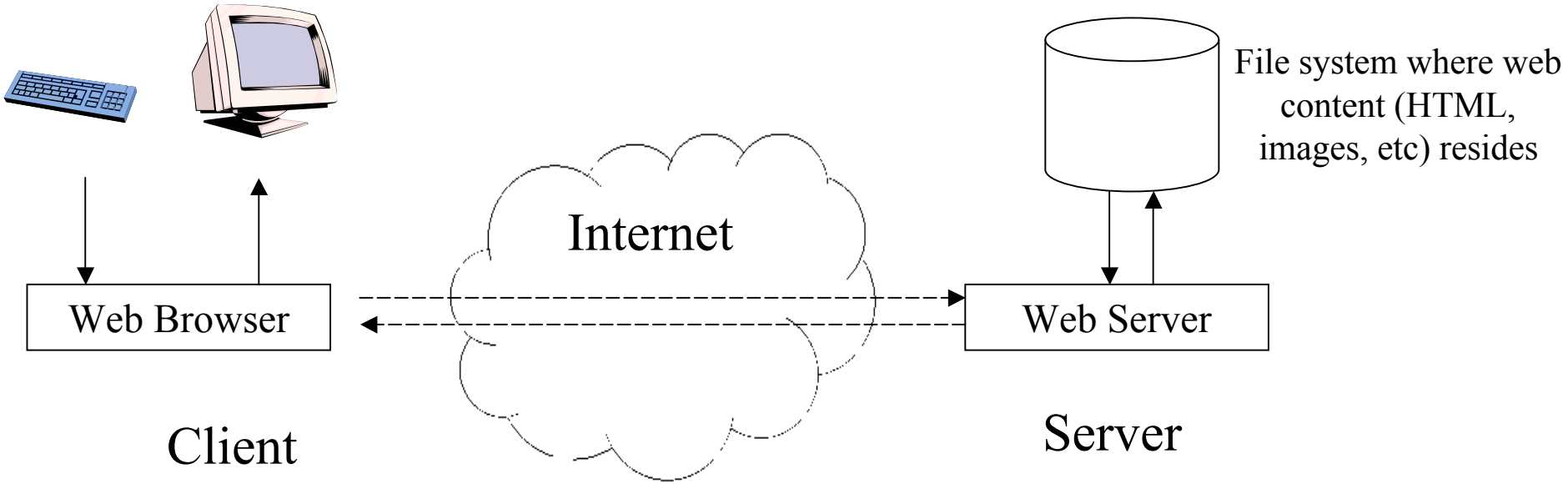
- Introduction to web serving
- Some important components of web serving:
  - HTTP
  - MIME
  - URLs

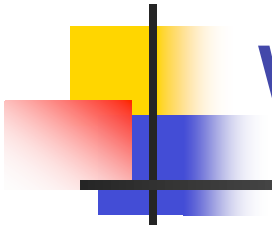


B108 and B211

- Those of you who did B108 and B211 may find some of today's material familiar
  - In B108 and B211, you only needed a rudimentary understanding. In this unit, you will need to learn the material in depth - so pay attention!

Basic Web Serving

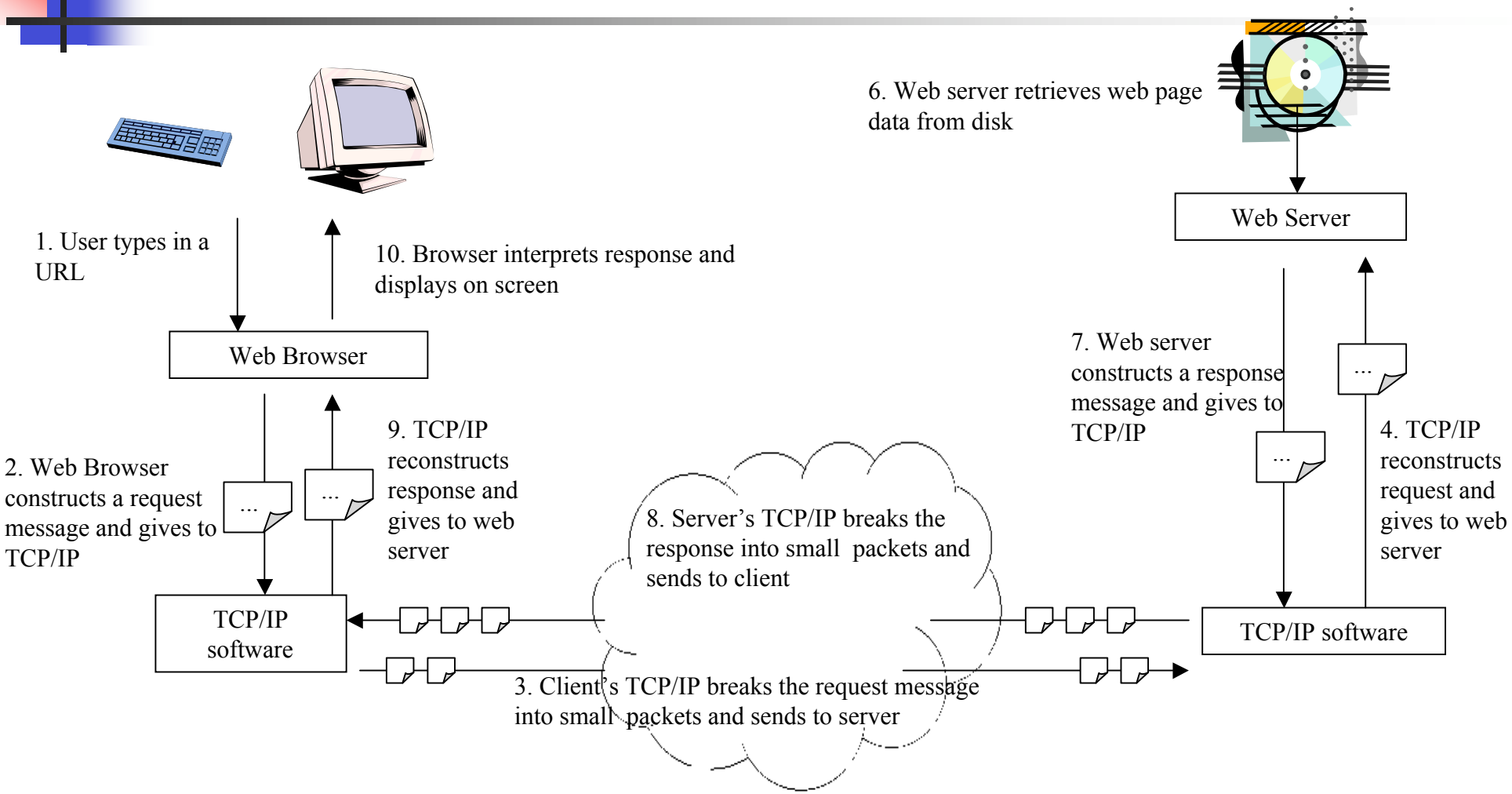


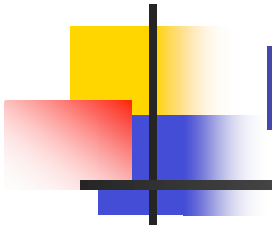


Web Servers and Clients

- Example web servers
  - Apache
  - Microsoft PWS (Personal Web Server) and IIS (Internet Information Server)
- Example web clients
  - Netscape Navigator
  - Microsoft Internet Explorer
- When users “surf the web”, they are basically asking their web client to request content from various web servers.

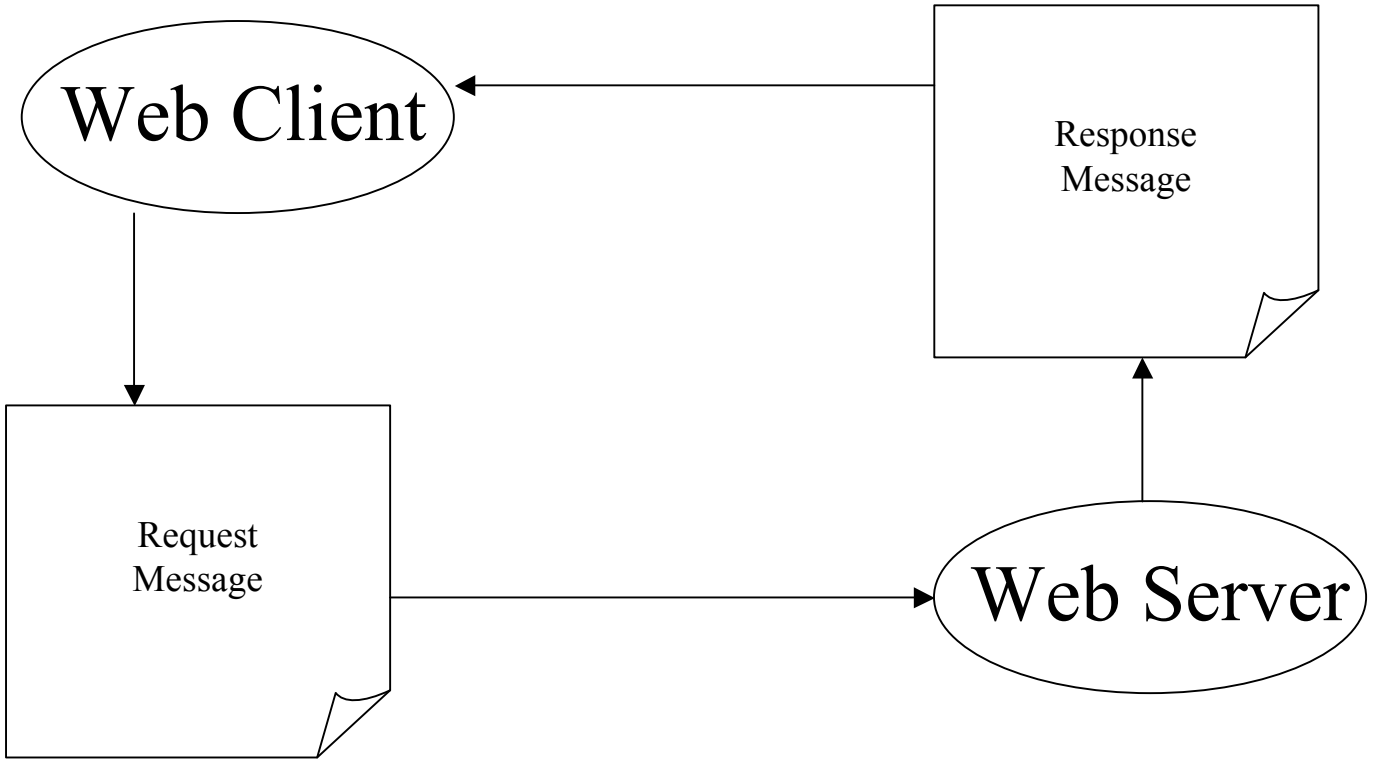
A More Complete Picture





In the next few weeks:

- We will **NOT** concern ourselves with what is happening
  - “above” web server - content-creation (how to make HTML pages, write PHP scripts, etc)
  - “below” the web server - at the TCP/IP level and below.
- What we will deal with is how the server handles the request and sends out of the content, and how the clients (browsers) make those requests and receives the content.

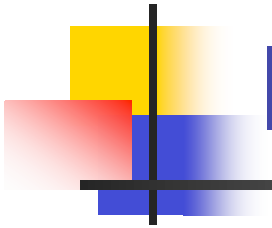


A major learning objective for this unit is for you to understand the above transaction well enough to program your own simple web client and web server.

Components of Web Communication

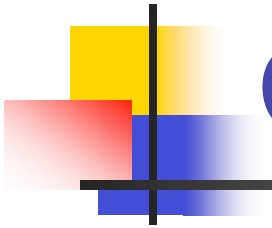
- Looking at the previous diagram, consider what needs to be in place for the exchange to happen:

1. A protocol on how to exchange the data (HTTP)
2. A protocol for the format of the data (MIME)
3. A way of identifying locations (URL)



Identifying Resources (the URL)

- We address resources on the web (HTML pages, images, scripts, etc) using the Uniform Resource Locator (URL).
- URLs are similar to filenames, except they include:
  - the name of the server they are on
  - network protocols used to access them
- All resources have their own *unique* URL on the web.



Components of a URL

- A URL has the following general form:

  <scheme> : <scheme-specific name>

- Most schemes have the following formats:

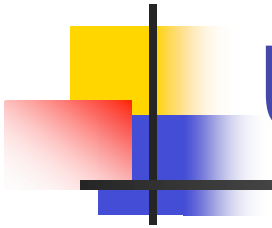
  <protocol>://<user>:<password>@<host>:<port>/<path>

- When using common browsers, we only specify the host and path. The protocols is assumed to be “http”, and the port number is “80”.

Example URLs

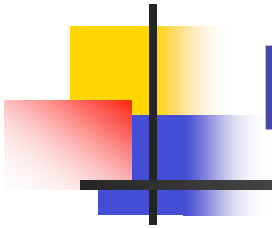
- http://www.w3.org/hypertext/WWW/TheProject.html
- ftp://hiew@ftp.it.murdoch.edu.au/pub/paper.txt
- mailto:h.hiew@murdoch.edu.au

- http://www.it.murdoch.edu.au:8888/~jsmith/index.html



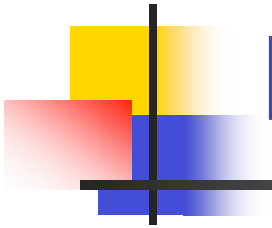
URI

- URL is a subset of URI (Uniform Resource Indentifier)
- URIs identifies resources by some meta-information about the resource.
- Besides URIs can be one of the following:
  - URL - indentifies by location
  - URN (Uniform Resource Name) - identifies by name



Ports

- In a multiple clients and multiple servers environment on one machine, it is not enough to have a machine address (domain name or IP number) for the hostname part of the URL.
- If a request only has a machine address, the operating system on the receiving side will not know which server program to give it to when it receives the request.
- The way to get around this is to always add a **port number** to addresses. This port number uniquely identifies the server the data is mean for.

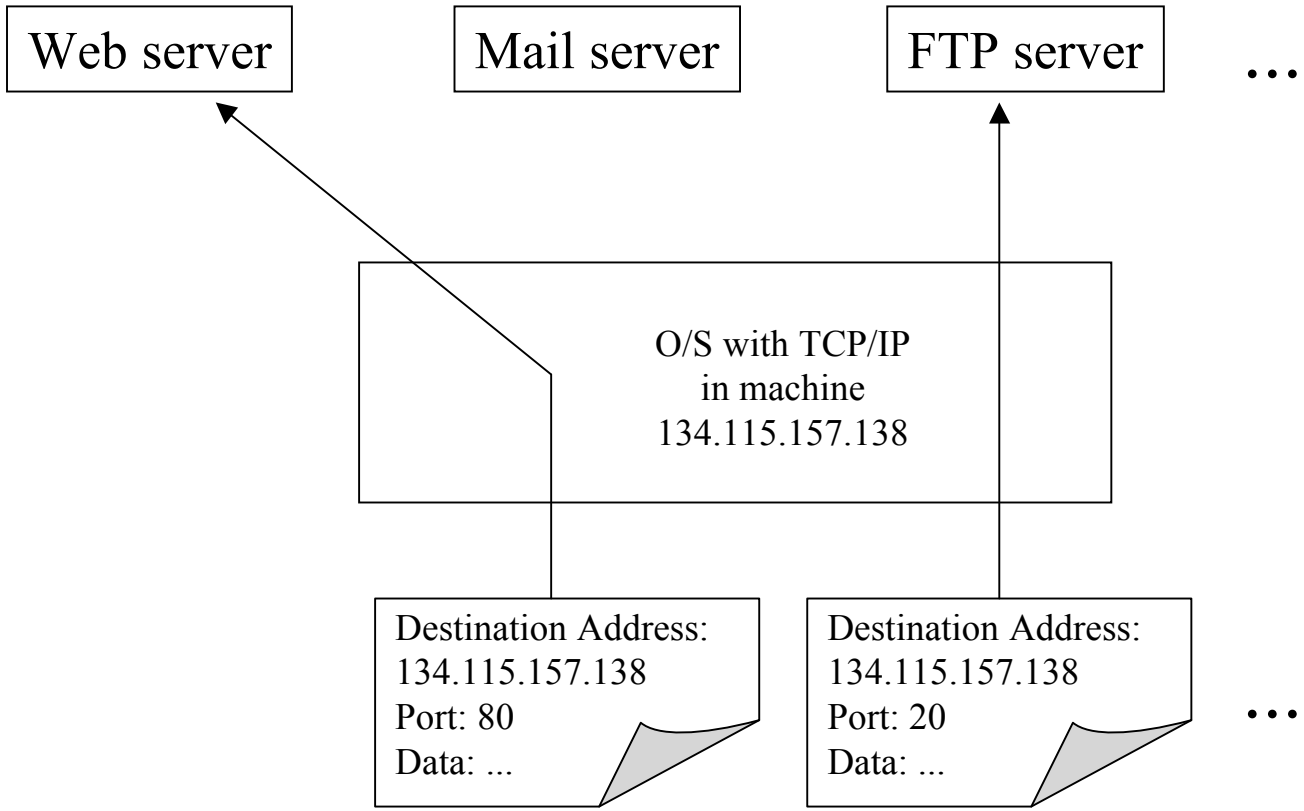


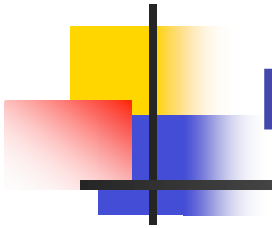
Ports (cont'd)

- Some common port numbers:
  - 80 for HTTP (web communication)
  - 20 and 21 for FTP (file transfers)
  - 25 for SMTP (e-mail)

- If a protocols uses a certain port number to identify itself, we say that the protocol "listens" on that port.

Ports (cont'd)

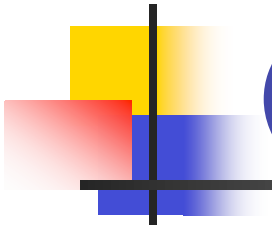

Web server
Mail server
FTP server
...
O/S with TCP/IP
in machine
134.115.157.138
Destination Address:
134.115.157.138
Port: 80
Data: ...
Destination Address:
134.115.157.138
Port: 20
Data: ...
...



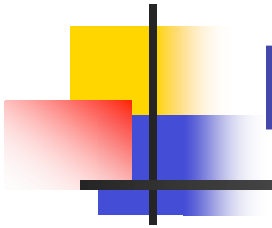
So, how does the actual resource “get” to the client?

- By using the URL:
  - The *protocol* part determines how to transmit the data.
  - The *hostname* part (with the *port* number) tells each intermediate machine on the Internet, including the receiving machine, which web server gets the request.
  - The web server converts the remaining part of the URL (the *path* part) to a file in the local file system, and then get that file.

The Hypertext Transfer Protocol (HTTP)

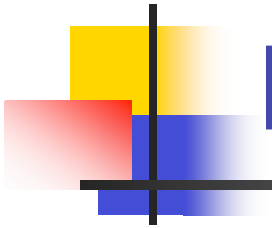


- The default protocol for transferring data on the web is HTTP
- In an HTTP interaction, we have one request, and one response.
- Each request-response transaction is independent of each other
  - The server/client must use other ways of keeping track of state information if they need it (eg. using cookies).
  - Note: the transactions at the request-response level is independent, but these days not at the TCP connection level.



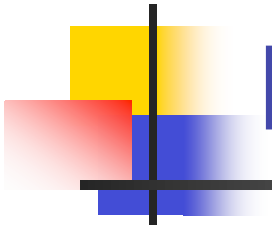
HTTP 1.0 and 1.1

- The current version of HTTP is 1.1.
- Some servers/clients still supports version 1.0
  - That means that if they receive request/response messages in version 1.0 format, they can handle it.



HTTP Client-Server Exchange

- At the most basic level, HTTP defines that:
  - A web client (user agent) initiates communication by sending a request to the web server.
  - The format of the request is precisely defined. All information the client wants the server to know is put in the request message.
  - A web server when it receives a request message, determines what to do based on what is in the request message.
  - The server responds to the web client by sending a response.
  - The format of the response is precisely defined. All information the client asked for, as well as other information the server wants the client to know (eg. error descriptions) is put in the response message.

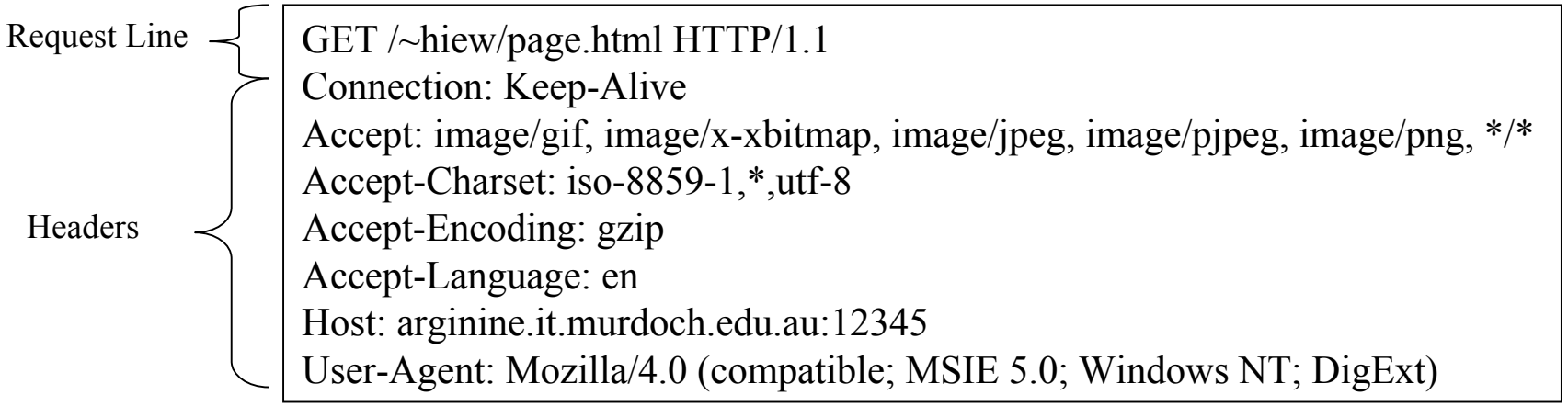


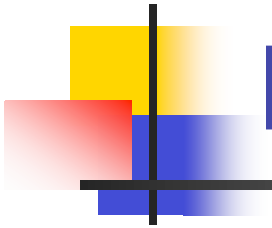
Multipurpose Internet Mail Extensions (MIME)

- HTTP requests and responses are in a format called MIME.
- Since MIME is a specifications for email, most HTTP messages look like email messages.
- HTTP messages consists of
  - A message body - ie. the content of the request/response.
  - Message headers - ie. control information required to transfer the data

HTTP Requests

- An example:

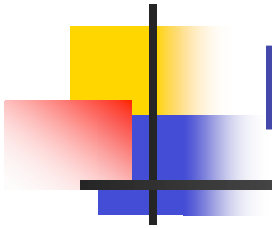




HTTP Request Line

- The *Request-Line* in a HTTP request (the first line in the previous example) consist of
  - a **method** to be executed
  - a **resource** to execute on
  - the **version of HTTP** used

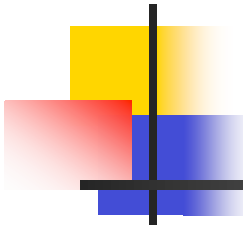
```
GET  /index.html  HTTP/1.1
```



HTTP Methods

- HTTP/1.1 servers and clients must implement the following 2 methods
  - **GET**: retrieve a resource from server and send to client.
  - **HEAD**: same request as GET, but only want the headers in the response.

- 5 other optional methods:
  - **POST**: send data from client to server (eg. when users fills a HTML form on the client) by appending to a resource.
  - **PUT**: request a server to replace a resource.
  - **DELETE**: request a server to remove a resource
  - **TRACE**: request a loop-back, by having the server send back the whole request message. Used for diagnostics.
  - **CONNECT**: used by proxies only.
  - **OPTIONS**: request information about the options available on the server.

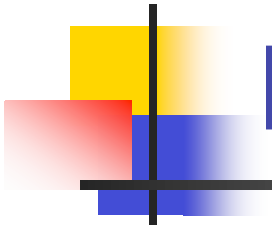


HTTP Responses

An example:

```
HTTP/1.1 200 OK
MIME-Version: 1.0
Server: Apache/1.3.12 (Unix)
Content-Type: text/html
Content-Length: 1234

<HTML>
   <HEAD>
      <TITLE>My Web Page</TITLE>
   </HEAD>
   <BODY>
      <P>This is my web page</P>
   </BODY>
</HTML>
```

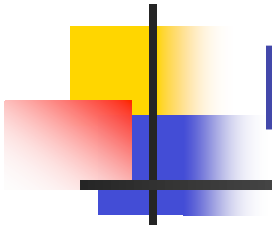


HTTP Responses

- HTTP responses are similar in format to a request, but instead of a Request-line, it has a Status-line to show the status of the response.

- The status line will contain a status code, consisting of a number and a text string describing the status.

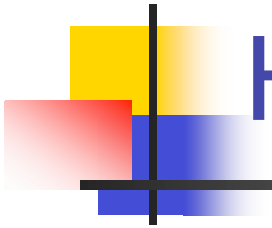
```
HTTP/1.1   200   OK
```



HTTP Status Codes

- Categories of status codes

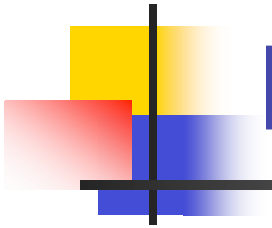
Category	Code numbers	Description
Informational	100-199	Application specific
Successful	200-299	Request successful
Redirection	300-399	Client needs to do further action
Client Error	400-499	Problems on client-side
Server Error	500-599	Problems on serve-side



HTTP Status Codes (cont'd)

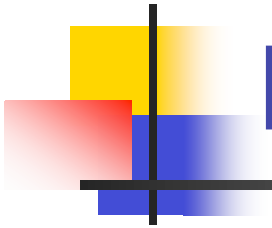
Come example common response status codes:

Code #	Code String	Description
200	OK	No error, request succeeded
301	Moved Permanently	Requested resource has moved to another URL permanently
400	Bad Request	The server does not understand the request
403	Forbidden	The client is not allowed to access the resource
404	Not Found	The server cannot find the resource



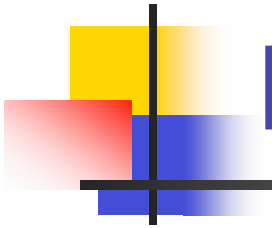
HTTP Header Fields

- Besides the request-line and status-line, HTTP messages also contains other header fields.
- Some are specific to requests, some to responses, and some are not supported by various clients and server.



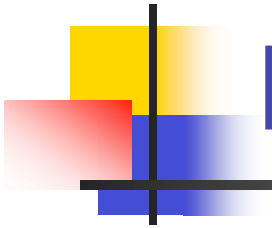
Example Header Fields

- Some example header fields used in most HTTP messages:
  - **User-Agent**: the name and version of the client
  - **Server**: the name and version of the server
  - **Accept**: media types the client is capable of accepting
  - **Allow**: the HTTP methods supported by the server
  - **Content-Type**: the media type of the message body
  - **Expires**: date and time the data in this message becomes invalid
  - **Pragma**: general-purpose field. A commonly used pragma value is “no-cache”, which indicates the data is very temporary and should not be cache.



MIME Types

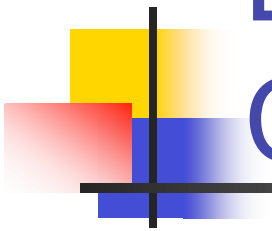
- Web clients (browsers) need to know what *type* the content contains.
  - Eg. Display a JPG image, play a Quicktime movie, etc.
- This is done by having all HTTP content associated with MIME-types
  - Email clients uses the same way to identify attachments.



MIME Types

- Some example standard MIME types:
  - text/plain
  - text/html
  - text/xml
  - image/gif
  - audio/wav
  - video/mpeg
- MIME-types for HTTP messages are indicated by the “Content-type” header.

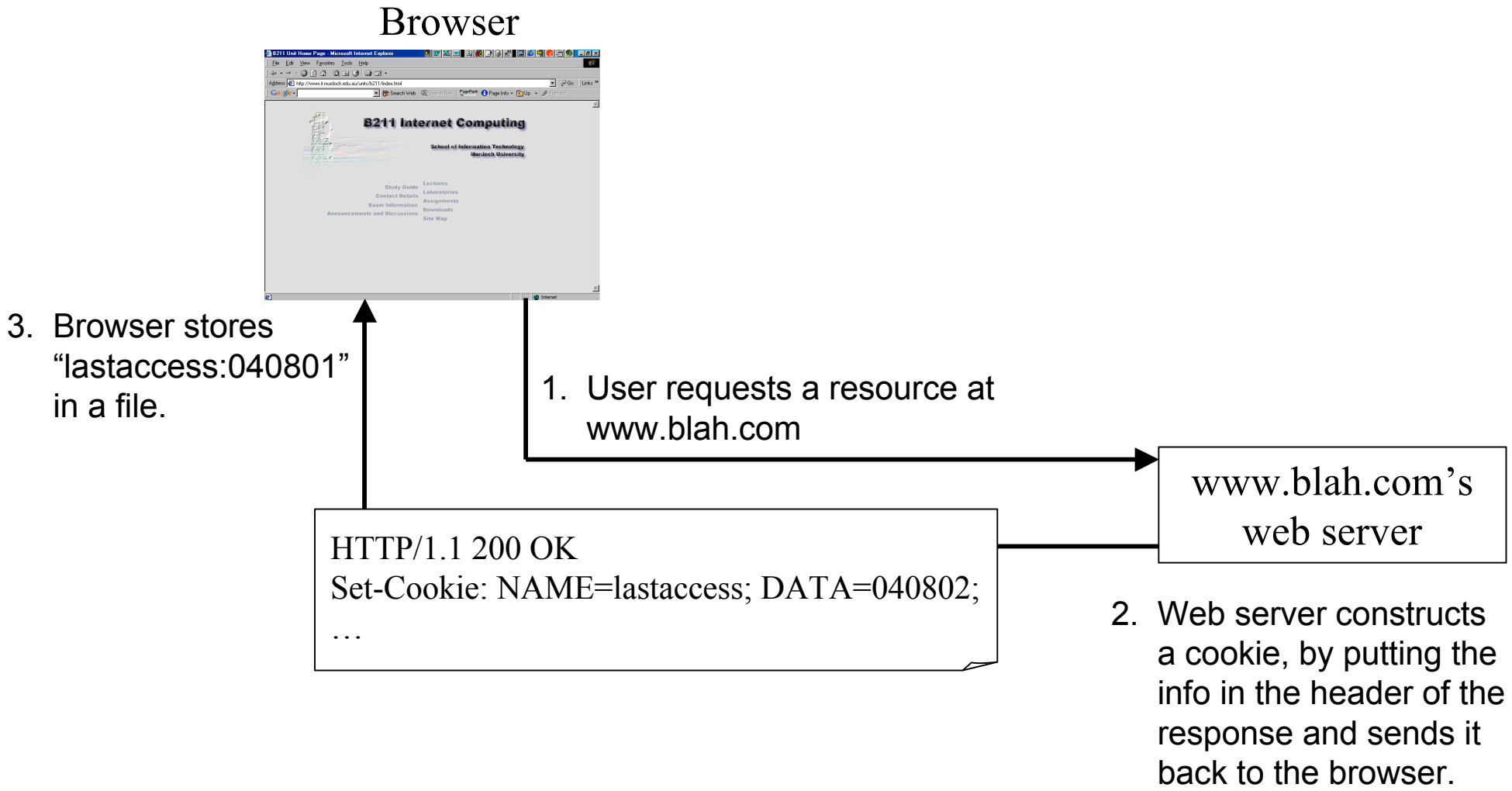
Example Header Fields: HTTP Cookies



- HTTP cookies are data sent by servers to clients, and are managed and stored locally by the clients. Clients send such cookie data back to the same server at later sessions.

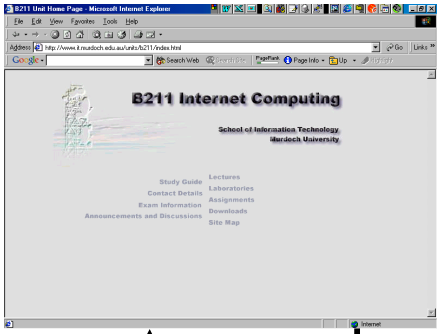
- Cookie data are sent using header fields in HTTP messages.

Example Cookie Transaction (first time)



Example Cookie Transaction (subsequent times)

Browser



1. User requests a resource at www.blah.com again

2. Browsers sees that there is a local cookie file for www.blah.com, so sends it with the request.

3. Server (or the requested resource) uses the cookie data as appropriate.

3. Browser cookie info in the same file as before.

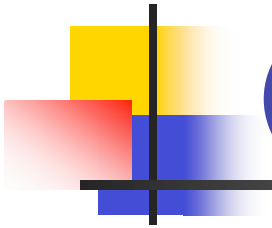
GET /index.html HTTP/1.1
Cookie: NAME=lastaccess; DATA=040802;
…

www.blah.com's web server

HTTP/1.1 200 OK
Set-Cookie: NAME=lastaccess; DATA=070802;
…

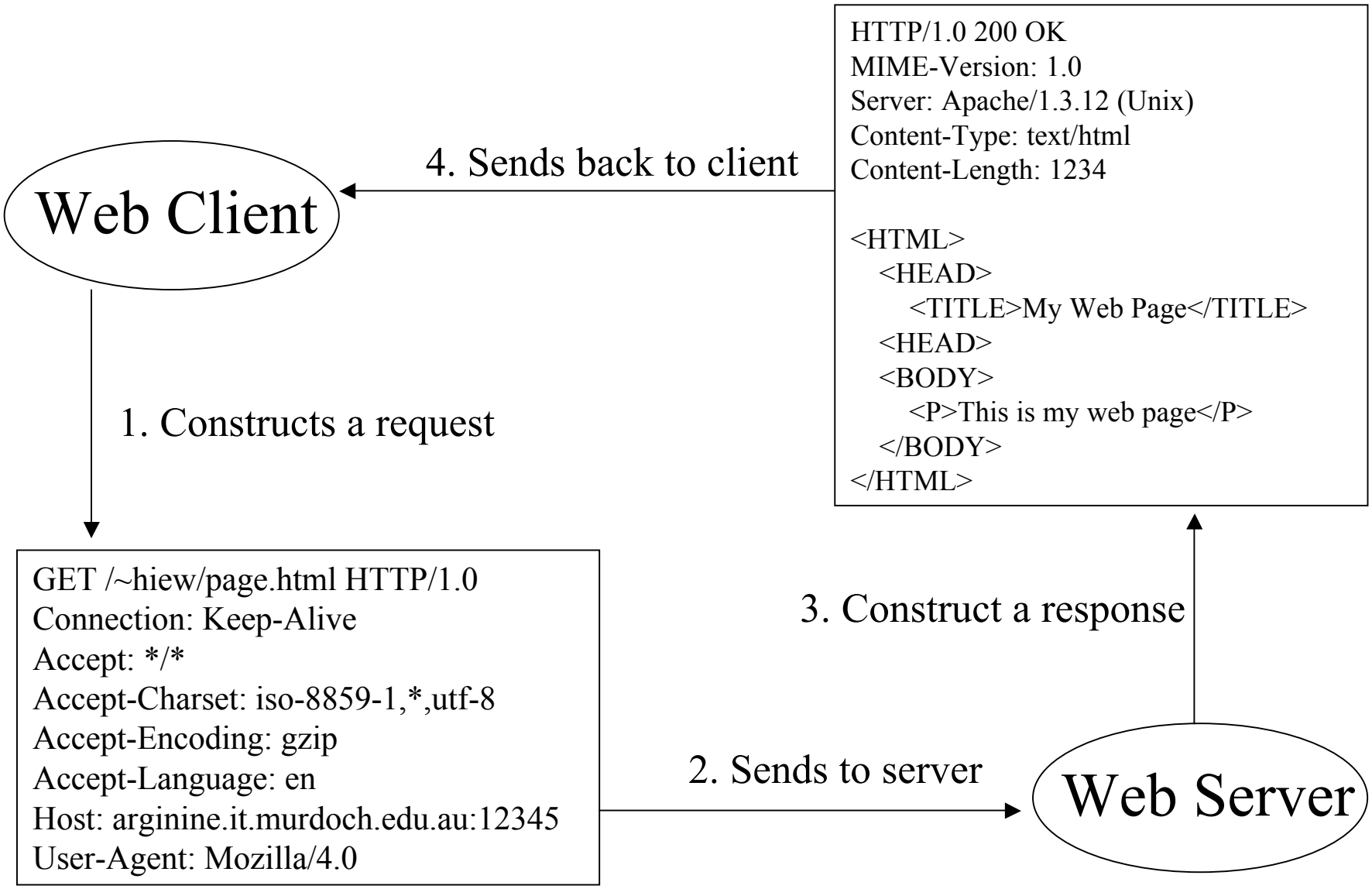
4. Web server sends the new cookie data back.

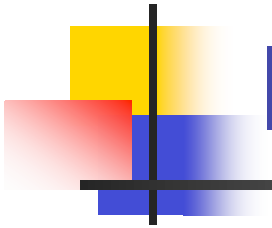
Summary: An HTTP Exchange (Very Important!)



- We will refer to the picture in the next page again and again in this unit. KEEP IT IN MIND!

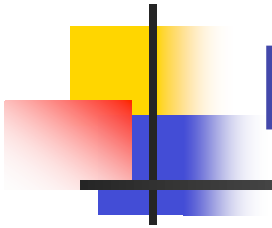
- Your assignment work will involve
  - Configuring Apache to handle the exchange.
  - Writing web clients to construct a request, send the request and receive the response.
  - Writing web servers to receive a request, construct a response and send the response.
  - To do these, you will need to know the details of HTTP (a lot more than what I have presented today).


HTTP/1.0 200 OK
MIME-Version: 1.0
Server: Apache/1.3.12 (Unix)
Content-Type: text/html
Content-Length: 1234
<HTML>
<HEAD>
<TITLE>My Web Page</TITLE>
<HEAD>
<BODY>
<P>This is my web page</P>
</BODY>
</HTML>
4. Sends back to client
Web Client
1. Constructs a request
GET /~hiew/page.html HTTP/1.0
Connection: Keep-Alive
Accept: */*
Accept-Charset: iso-8859-1,*,utf-8
Accept-Encoding: gzip
Accept-Language: en
Host: arginine.it.murdoch.edu.au:12345
User-Agent: Mozilla/4.0
2. Sends to server
3. Construct a response
Web Server



In the next few weeks...

- We will
  - First use Apache to first look at the specifics of how a web server handles the above tasks
    - Eg. What to do when it receives a request, where to find the resources, how to deal with multiple requests, etc.
  - Go even lower and then deal with how to program the basic operations of web servers and web clients.



Further Reading

- Required Reading
  - Unit Reader 1: Web Servers
- Optional Reference
  - Official HTTP 1.1 specifications (http://www.w3.org/Protocols/rfc2616/rfc2616.html)