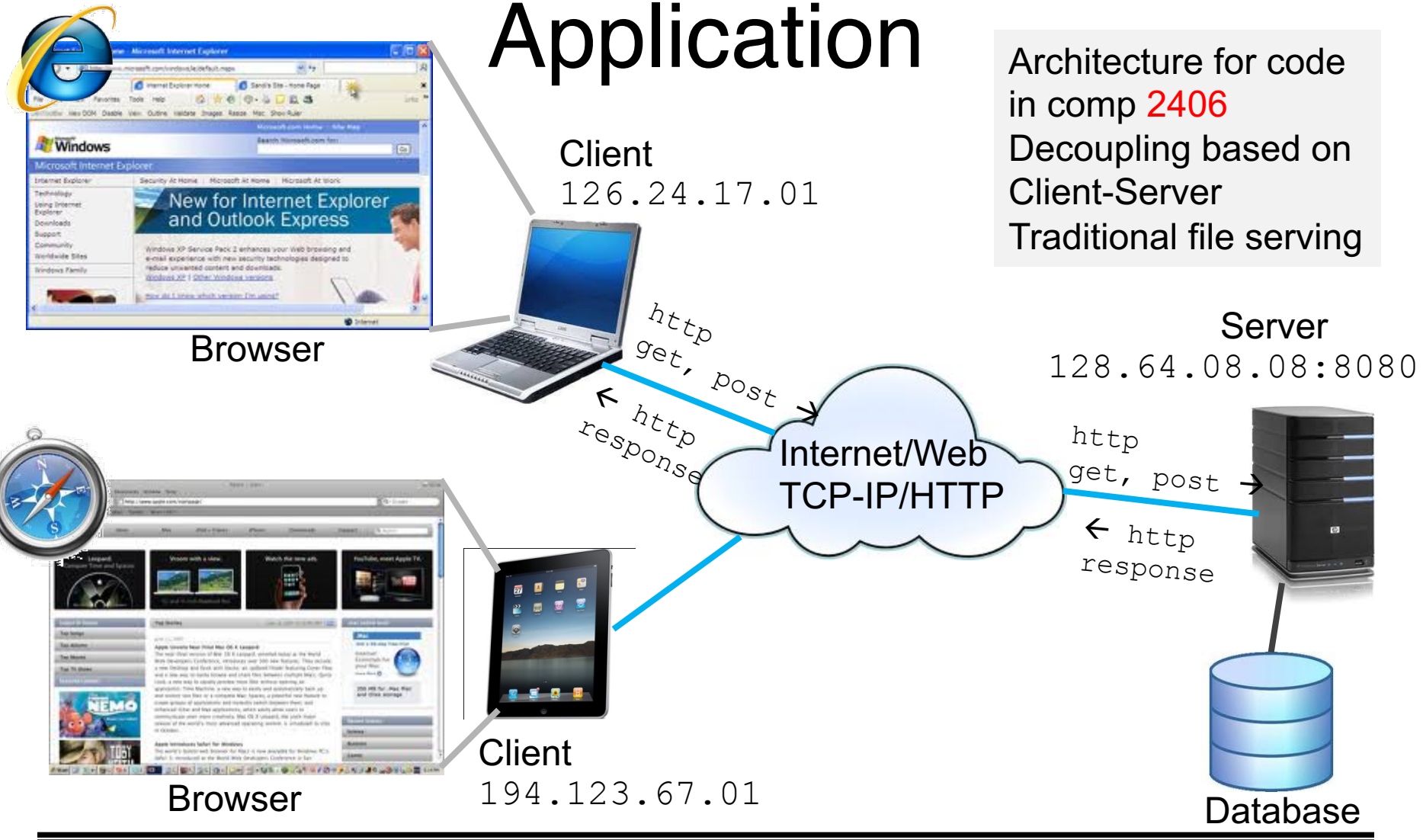
HTTP Basics

Reference Picture of Web Application



Architecture for code in comp 2406
Decoupling based on Client-Server
Traditional file serving

Internet of Things

Internet of things:

tcp/ip or http based

who is the server?

Thermostat
126.24.17.01:3000

other TCP/IP protocol

other TCP/IP protocol

Furnace
128.64.08.08:3000

Internet-Web
TCP/IP - HTTP

http
get, post

http
response

Browser

http
get, post

http
response

Weather Service
api.openweather.org

HTTP

- HyperText Transfer Protocol
- Implements a request-response model.
- Basis of all WWW communications
- Defines format for requests and responses

HTTP

Important Design philosophy:

- Stateless protocol/service
- Open plain-text protocol

Implications of these are far reaching and has formed the basis of how the internet works (and crashes, and is breached)

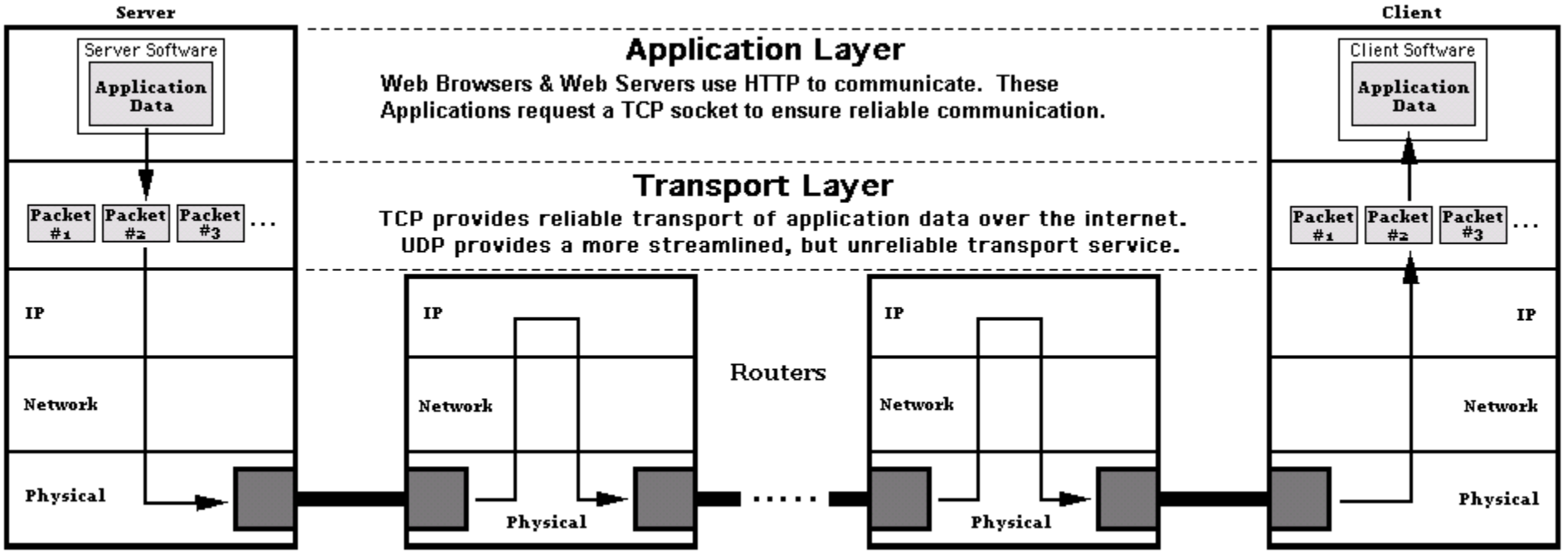
IP Addresses

- Internetworking Protocol (IP) dominates Internetwork Layer
- Every packet has *to* and *from* addresses
- *IP Addresses* are 4 byte numbers –based on current IPv4 standard (1984)
- Assigned by IANA (Internet Assigned Numbers Authority)
- Usually separated by decimals:

  134.117.29.41

  – We are fast running out of IP addresses
  – Newer IPv6 standard has much bigger address space

Application to Application



Transport Layer: reliable transmission of packets (TCP) and re-assembly in correct order
Reliability achieved through re-transmission, checksums
Outstanding message count provides flow control mechanism
UDP provides streamlined unreliable transport.

- *Domain names*
  - Form: host-name.domain-names

  e.g. `scs.carleton.ca`
  `people.scs.carleton.ca`

  - First domain is the smallest; last is the largest (like mail addresses on envelopes)

  - Last domain typically specifies the type of organization or country (e.g. `.com`, `.ca`, `.za`)

  - *Fully qualified domain name* - the host name and all of the domain names

- DNS servers - convert fully qualified domain names to IP addresses

(like a phonebook maps peoples names to phone #s)

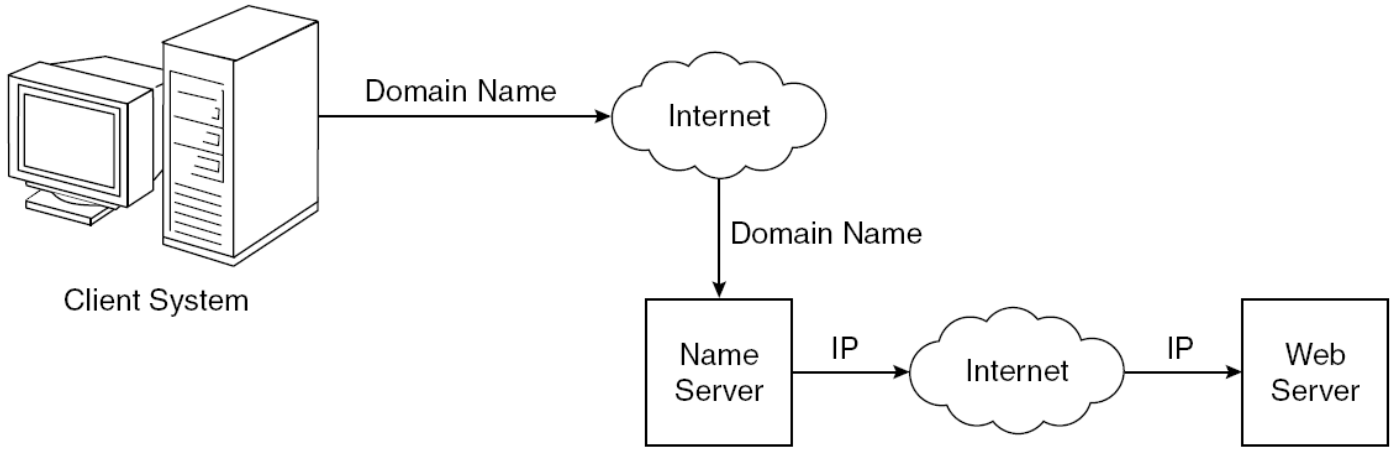


Figure 1.1 Domain name conversion

Source: Sebesta "Programming the World Wide Web" 6th ed

Example DNS translation

In theory you could substitute the looked up IP address for the domain.

http://people.scs.carleton.ca/~ldnel/2405winter2011

http://134.117.29.41/~ldnel/2405winter2011

WARNING: I did the translation using one of the many free DNS lookup websites –it loaded my machine with malware. Also: some of the "what is my IP address" web sites will load malware.

Example DNS translation

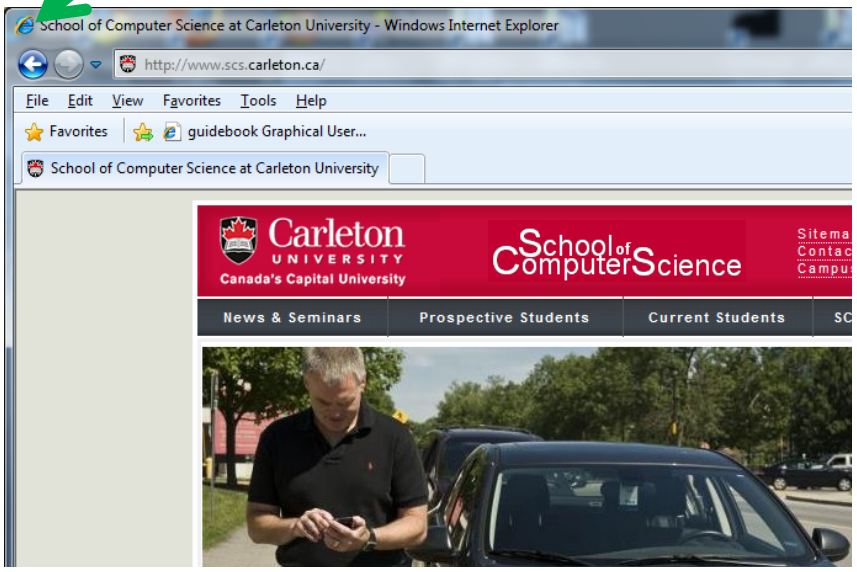
http://people.scs.carleton.ca/~ldnel/2405winter2011

http://134.117.29.41/~ldnel/2405winter2011

Example DNS translation

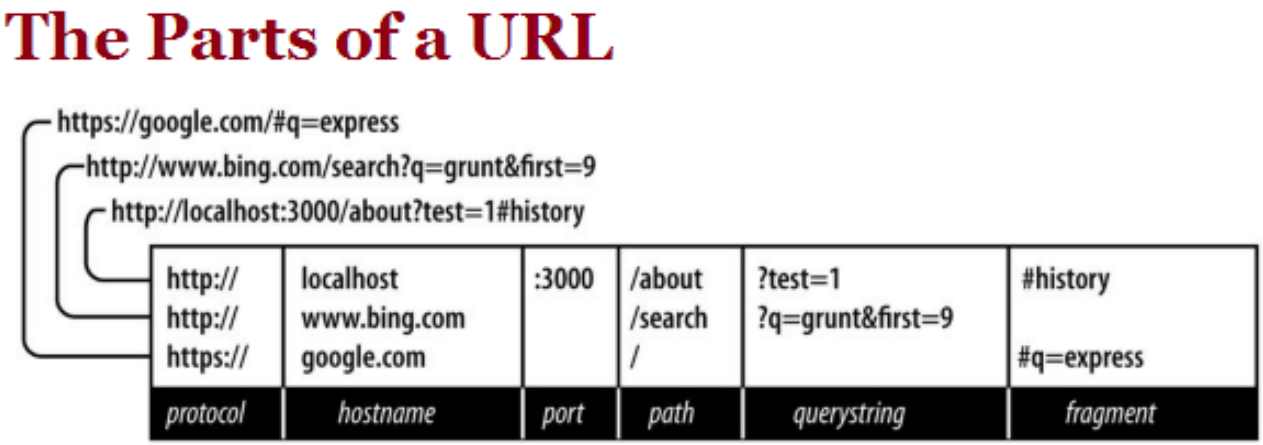
Because servers get to see the actual domain, a simple Substitution does not always work (i.e. server redirects based on seeing actual domain name.

http://www.scs.carleton.ca/



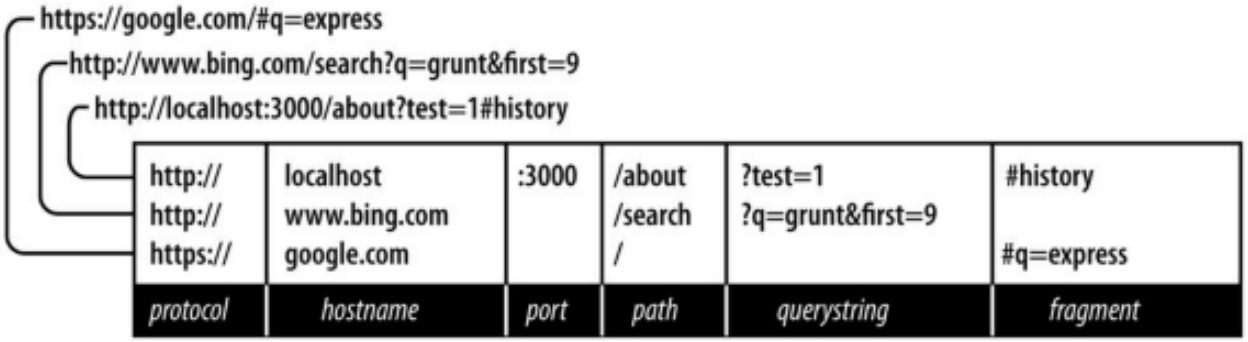
http://134.117.29.40/

Parts of URL



Source: Brown, Ethan (2014-07-01). Web Development with Node and Express

The Parts of a URL

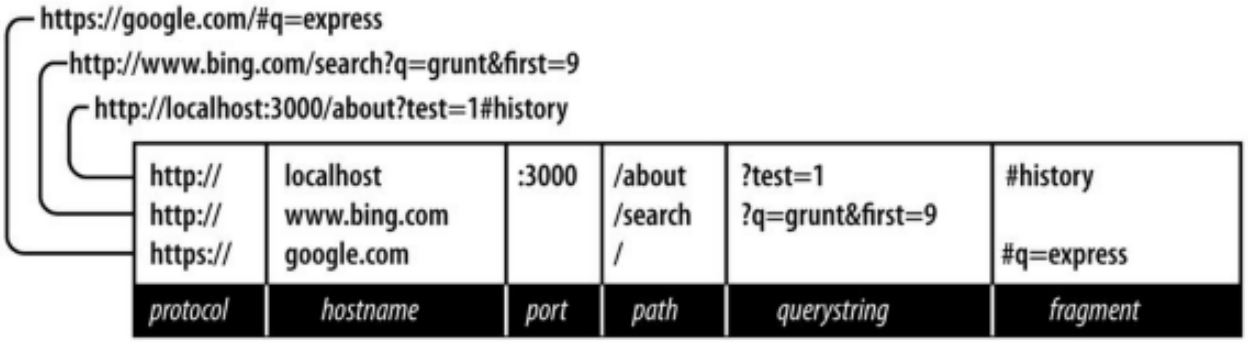


Protocol

The protocol determines how the request will be transmitted. We will be dealing exclusively with *http* and *https*. Other common protocols include *file* and *ftp*.

Source: Brown, Ethan (2014-07-01). Web Development with Node and Express

The Parts of a URL

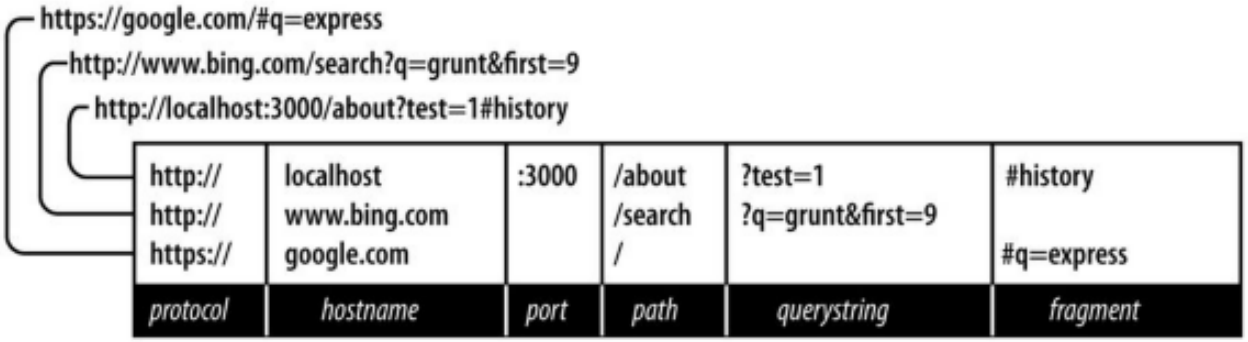


Host

The host identifies the server. Servers on your computer (localhost) or a local network may simply be one word, or it may be a numeric IP address. On the Internet, the host will end in a top-level domain (TLD) like *.com* or *.net*. Additionally, there may be *subdomains*, which prefix the hostname. *www* is a very common subdomain, though it can be anything. Subdomains are optional.

Source: Brown, Ethan (2014-07-01). Web Development with Node and Express

The Parts of a URL

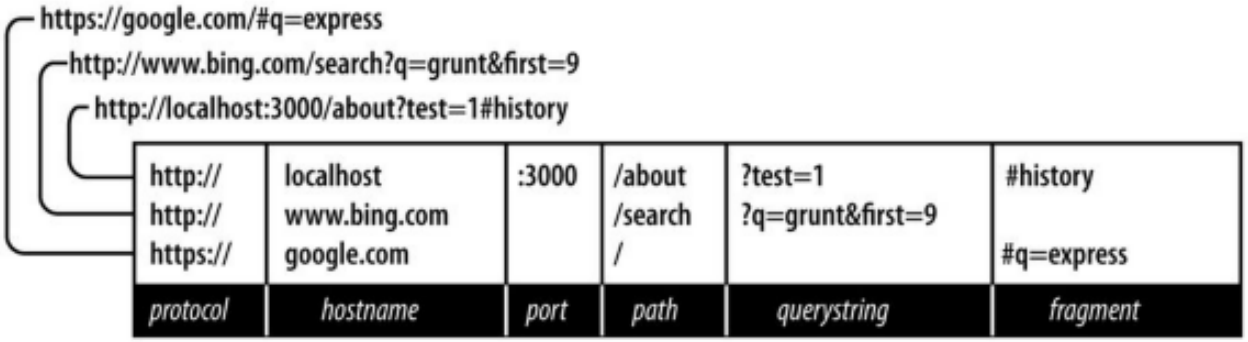


Port

Each server has a collection of numbered ports. Some port numbers are "special," like 80 and 443. If you omit the port, port 80 is assumed for HTTP and 443 for HTTPS. In general, if you aren't using port 80 or 443, you should use a port number greater than 1023.[6] It's very common to use easy-to-remember port numbers like 3000, 8080, and 8088.

Source: Brown, Ethan (2014-07-01). Web Development with Node and Express

The Parts of a URL

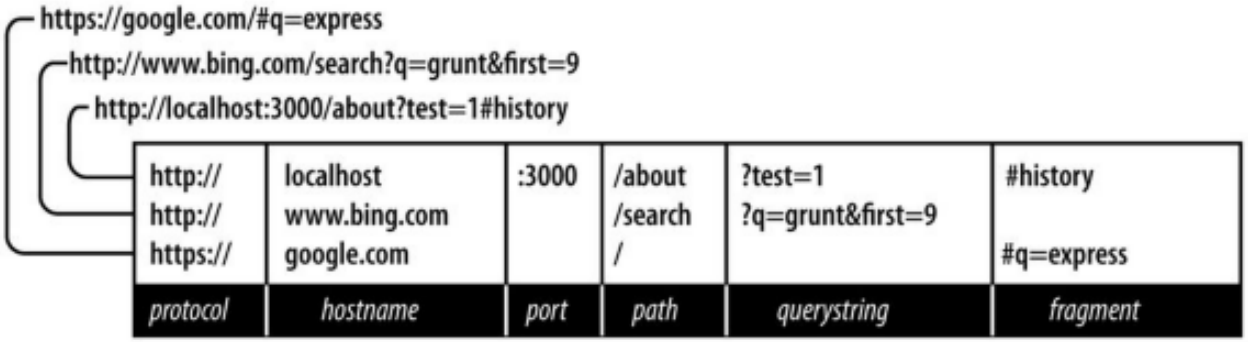


Path

The path is generally the first part of the URL that your app cares about (it is possible to make decisions based on protocol, host, and port, but it's not good practice). The path should be used to uniquely identify pages or other resources in your app.

Source: Brown, Ethan (2014-07-01). Web Development with Node and Express

The Parts of a URL

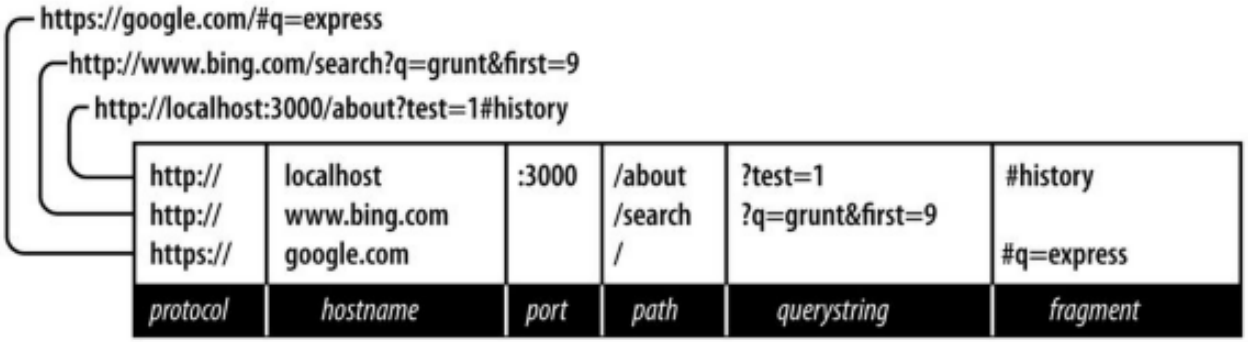


Querystring

The querystring is an optional collection of name/value pairs. The querystring starts with a question mark (?), and name/value pairs are separated by ampersands (&). Both names and values should be *URL encoded*. JavaScript provides a built-in function to do that: `encodeURIComponent`. For example, spaces will be replaced with plus signs (+). Other special characters will be replaced with numeric character references.

Source: Brown, Ethan (2014-07-01). Web Development with Node and Express

The Parts of a URL



Fragment

The *fragment* (or *hash*) is not passed to the server at all: it is strictly for use by the browser. It is becoming increasingly common for single-page applications or AJAX-heavy applications to use the fragment to control the application. Originally, the fragment's sole purpose was to cause the browser to display a specific part of the document, marked by an anchor tag (`<a id="chapter06">`).

Source: Brown, Ethan (2014-07-01). Web Development with Node and Express

Parsing URL Strings with Node.js

```
var url = require('url');

var parseIt = function(url_string){

   var parseQuery = true; //parseQueryStringIfTrue
   var slashHost = true; //slashDenoteHostIfTrue

   var urlObj = url.parse(url_string, parseQuery , slashHost );

   console.log('input: ' + url_string);
   console.log('URL properties');
   console.log('-------------');

   console.log('href: ' + urlObj.href);
   console.log('host: ' + urlObj.host);
   console.log('hostname: ' + urlObj.hostname);
   console.log('port: ' + urlObj.port);
   console.log('auth: ' + urlObj.auth);
   console.log('path: ' + urlObj.path);
   console.log('pathname: ' + urlObj.pathname);
   console.log('search: ' + urlObj.search);
   console.log('hash: ' + urlObj.hash);
   console.log('query: ' + urlObj.query);
   for(x in urlObj.query) console.log(x + ': ' +
urlObj.query[x]);

}
```

Source: Sebesta "Programming the World Wide Web" 6th ed

Parsing URL Strings with Node.js

```
input: http://localhost:3000/index.html?fname=Lou&lname=Nel#contents
URL properties
-------------
href: http://localhost:3000/index.html?fname=Lou&lname=Nel#contents
host: localhost:3000
hostname: localhost
port: 3000
auth: null
path: /index.html?fname=Lou&lname=Nel
pathname: /index.html
search: ?fname=Lou&lname=Nel
hash: #contents
query: [object Object]
fname: Lou
lname: Nel
```

HTTP Requests

- Form:

  HTTP method domain part of URL HTTP ver.

  Header fields

  blank line

  Message body

  - An example of the first line of a request:

```
GET  /cs.uccp.edu/degrees.html  HTTP/1.1
```

Examine Request Header in Node.js

```
http.createServer(function (request,response){
    //write HTTP header

    console.log('');
    console.log('==============');

    //display url string from the request object
    console.log('HEADERS');
    console.log(request.headers);
    for(x in request.headers) console.log(x + ': ' +
request.headers[x]);
    console.log('------------------');
    console.log('method: ' + request.method);
    console.log('domain: ' + request.domain);
    console.log('url: ' + request.url);

}
```

```
==============
HEADERS
{ accept: 'text/html, application/xhtml+xml, */*',
  'accept-language': 'en-CA,en;q=0.5',
  'user-agent': 'Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; Touch
 rv:11.0) like Gecko',
  'accept-encoding': 'gzip, deflate',
  host: 'localhost:3000',
  dnt: '1',
  connection: 'Keep-Alive' }
accept: text/html, application/xhtml+xml, */*
accept-language: en-CA,en;q=0.5
user-agent: Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; Touch; MDD
1.0) like Gecko
accept-encoding: gzip, deflate
host: localhost:3000
dnt: 1
connection: Keep-Alive
------------------
method: GET
domain: null
url: /index.html?lou=nel
```

1.7 The HyperText Transfer Protocol

- *Most commonly used methods (verbs)*:

`GET` - "Fetch a document"
`POST` – "Execute the document, using the data in body"
`HEAD` - Fetch just the header of the document
`PUT` - Store a new resource on the server
`DELETE` - Remove a resource from the server

The GET Method

Note that query strings (name/value pairs) is sent in the URL of a GET request:

```
/test/demo_form.asp?name1=value1&name2=value2
```

Some other notes on GET requests:

- GET requests can be cached
- GET requests remain in the browser history
- GET requests can be bookmarked
- GET requests should never be used when dealing with sensitive data
- GET requests have length restrictions
- GET requests should be used only to retrieve data

Source: http://www.w3schools.com/tags/ref_httpmethods.asp

The POST Method

Note that query strings (name/value pairs) is sent in the HTTP message body of a POST request:

```
POST /test/demo_form.asp HTTP/1.1
Host: w3schools.com
name1=value1&name2=value2
```

Some other notes on POST requests:

- POST requests are never cached
- POST requests do not remain in the browser history
- POST requests cannot be bookmarked
- POST requests have no restrictions on data length

Source: http://www.w3schools.com/tags/ref_httpmethods.asp

	GET	POST
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

Source: http://www.w3schools.com/tags/ref_httpmethods.asp

Other HTTP Request Methods

The following table lists some other HTTP request methods:

Method	Description
HEAD	Same as GET but returns only HTTP headers and no document body
PUT	Uploads a representation of the specified URI
DELETE	Deletes the specified resource
OPTIONS	Returns the HTTP methods that the server supports
CONNECT	Converts the request connection to a transparent TCP/IP tunnel

Source: http://www.w3schools.com/tags/ref_httpmethods.asp

1.7 The HyperText Transfer Protocol

(continued)

- Response Phase

- Form:
  - Status line
  - Response header fields
  - blank line
  - Response body

- Status line format:
  HTTP version status code explanation

  -status code meant to be machine readable
  -explanation meant to be human readable

- Example: `HTTP/1.1  200  OK`

  (Current version is 1.1)

HTTP Responses

- Header and Body separated by empty line
- Header must state content-type
- Example:

```
HTTP/1.0 200 OK
Date: Fri, 27 Aug 2004 10:05:30 GMT
Server: Apache/1.2.13 (Linux)
Last-Modified: Thu, 26 Aug 2004 20:14:26 GMT
Content-Length: 2523
Content-Type: text/html

<html> <head> <title>COMP 2405 Internet
   Applications</title> ...
```

Source: Sebesta "Programming the World Wide Web" 6th ed

Response Code

```
HTTP/1.0 200 OK
```

Response may come with a code number:

```
1xx = information
2xx = success
3xx = redirection
4xx = client error (e.g. common 404 "Not Found" error)
5xx = server error
```

Most significant digit could be used to identify the class of response

1.7 The HyperText Transfer Protocol

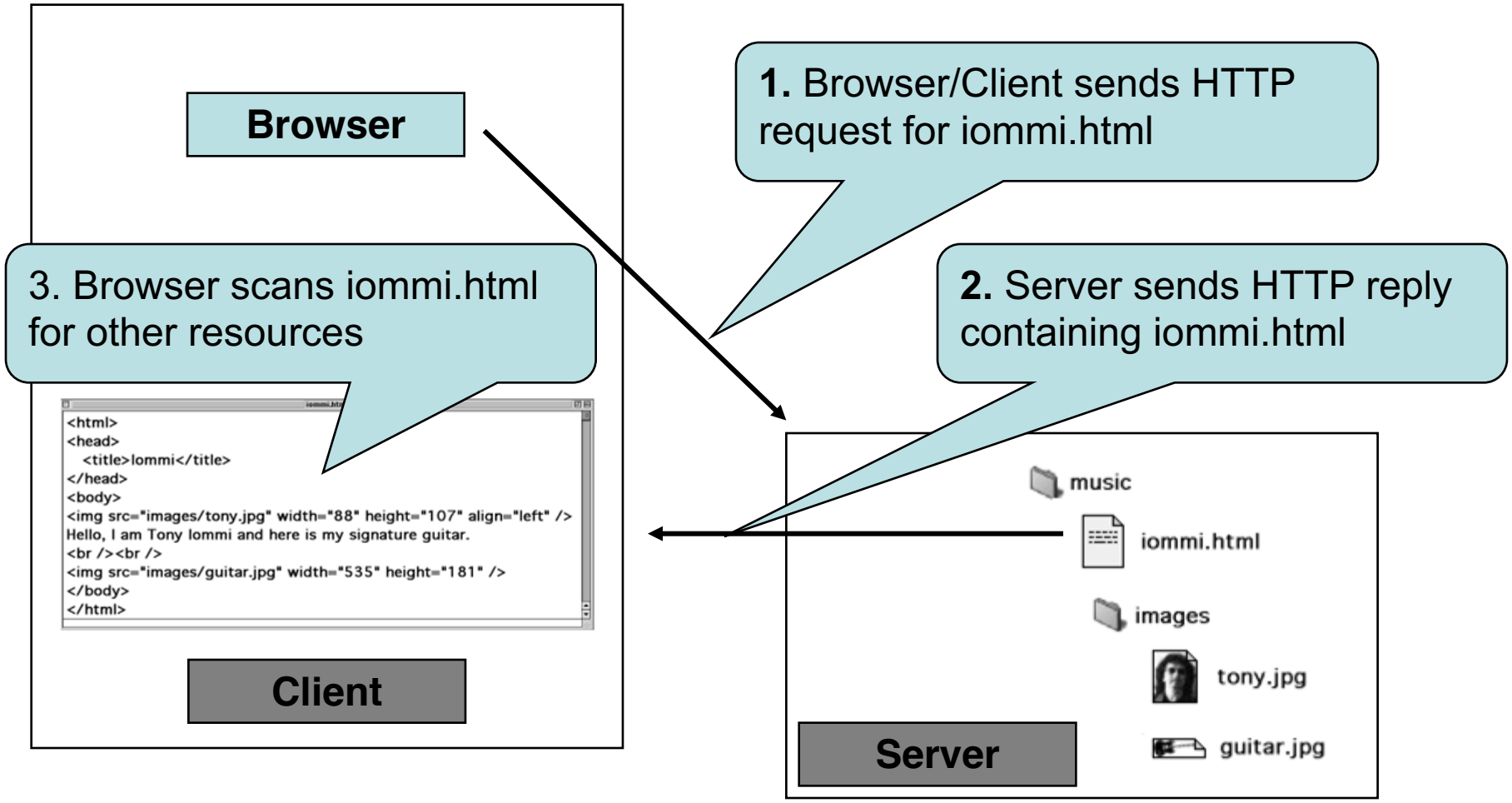
(continued)

- An example of a complete response header:

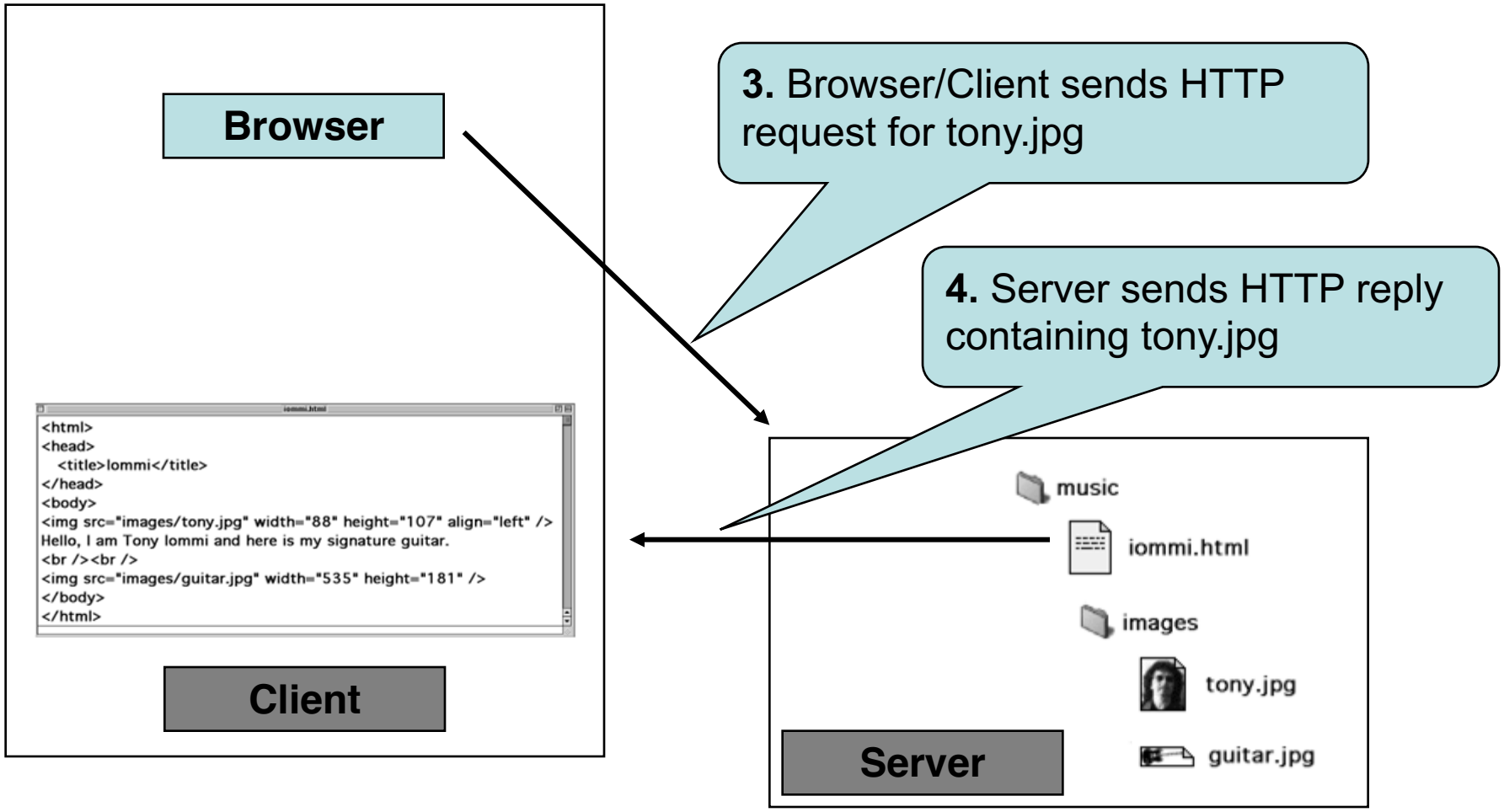
```
HTTP/1.1  200  OK
Date: Sat, 25 July 2009 20:15:11 GMT
Server: Apache /2.2.3 (CentOS)
Last-modified: Tues, 18 May 2004 16:38:38 GMT
Etag: "1b48098-16a-3dab592dc9f80"
Accept-ranges: bytes
Content-length: 364
Connection: close
Content-type: text/html, charset=UTF-8
```

- Both request headers and response headers must be followed by a blank line

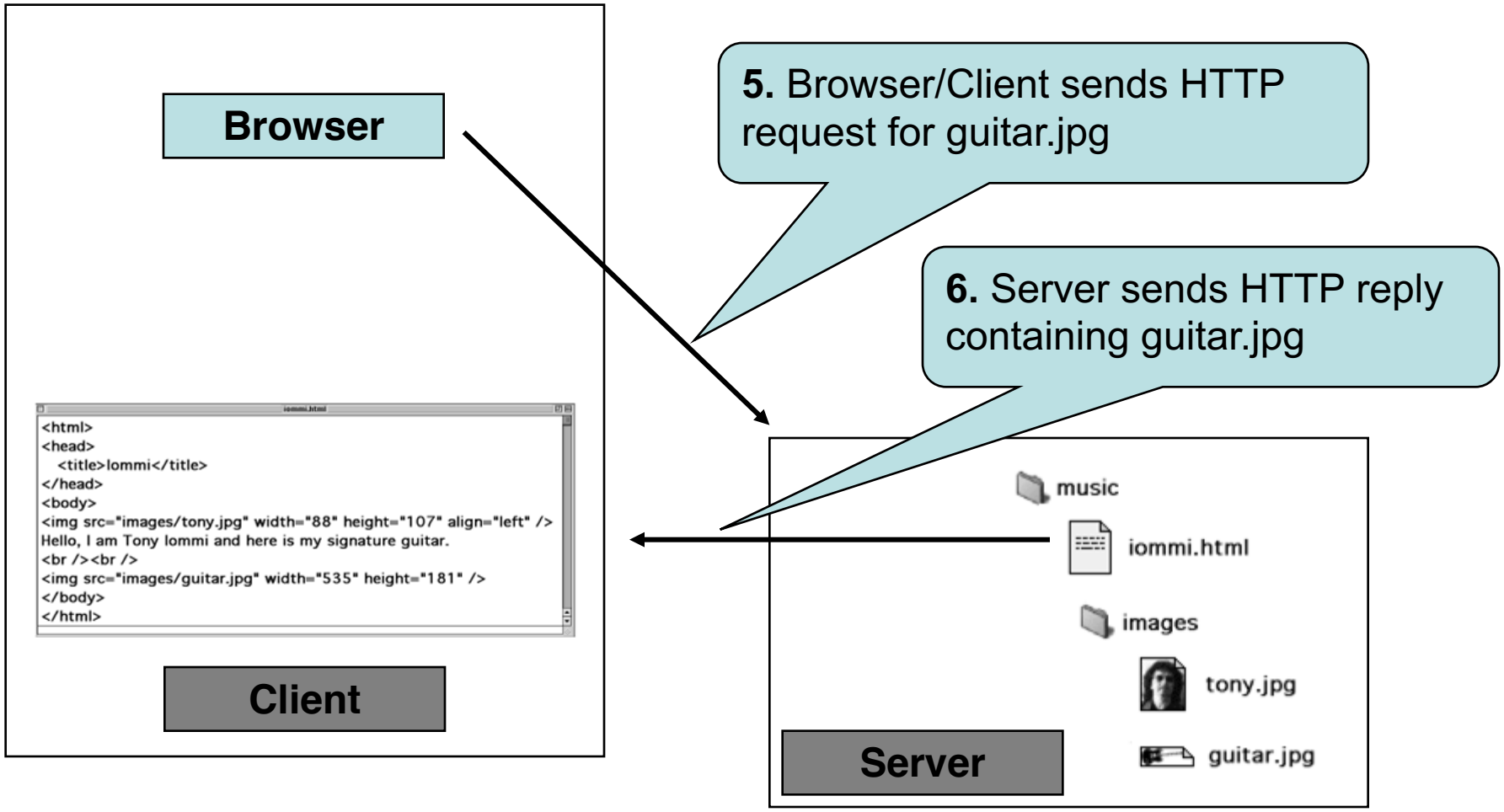
“Basic” HTTP Transaction



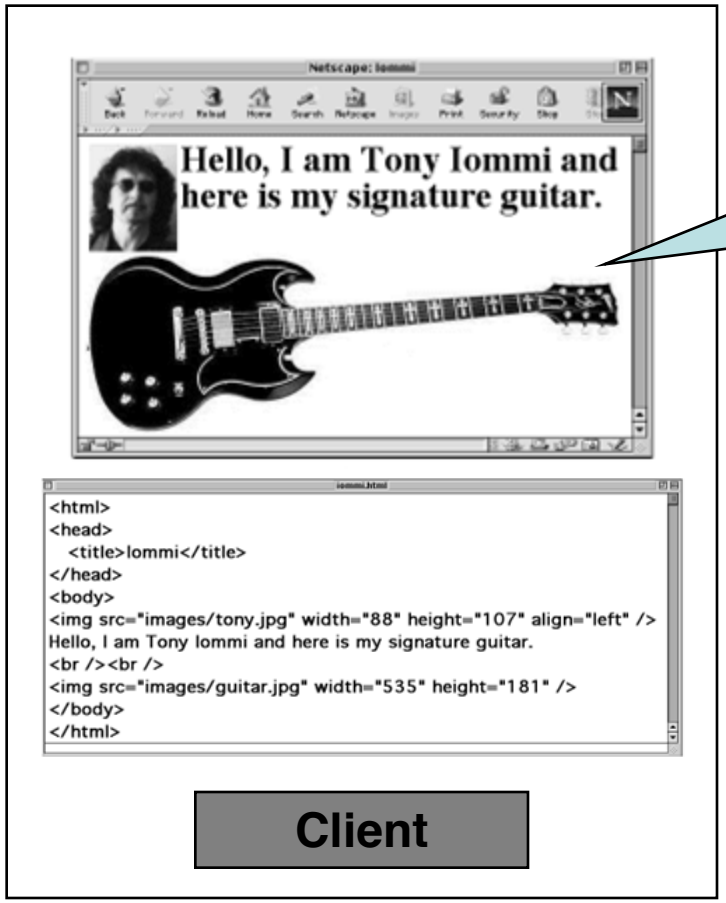
“Basic” HTTP Transaction



“Basic” HTTP Transaction



“Basic” HTTP Transaction



7. Browser uses all three resources to “render” the original requested page iommi.html

Q. Could the client and server be the same machine?
Ans. Yes