Introduction to Computer Science I
COMP 2406A – Winter 2020

Intro to Web Apps and HTTP

Dave McKenney
david.mckenney@carleton.ca

Learning Outcomes

by the End of this Lecture, Students that have Completed the Reading Assignment and Review Questions should be Able to:

Understand the basics of the HTTP protocol

Identify advantages of statelessness in HTTP

Explain a request/response model of processing

Identify/Explain parts of a Uniform Resource Locator

Explain benefits of caching in web applications

Summary of First-Year Programming

Typical first-year programming (1405/1406) involves:

Desktop applications, largely console-based

Single user doing a single thing at a time

No interacting resources

A little bit of event-driven programming (Java GUI)

Basic Desktop App Architecture

Output:
Console/Screen

Input/Output
Storage

PC

Less to be aware of and worry about than web apps

Input:
Keyboard/Mouse

Summary of Web App Programming

Web app programming involves:

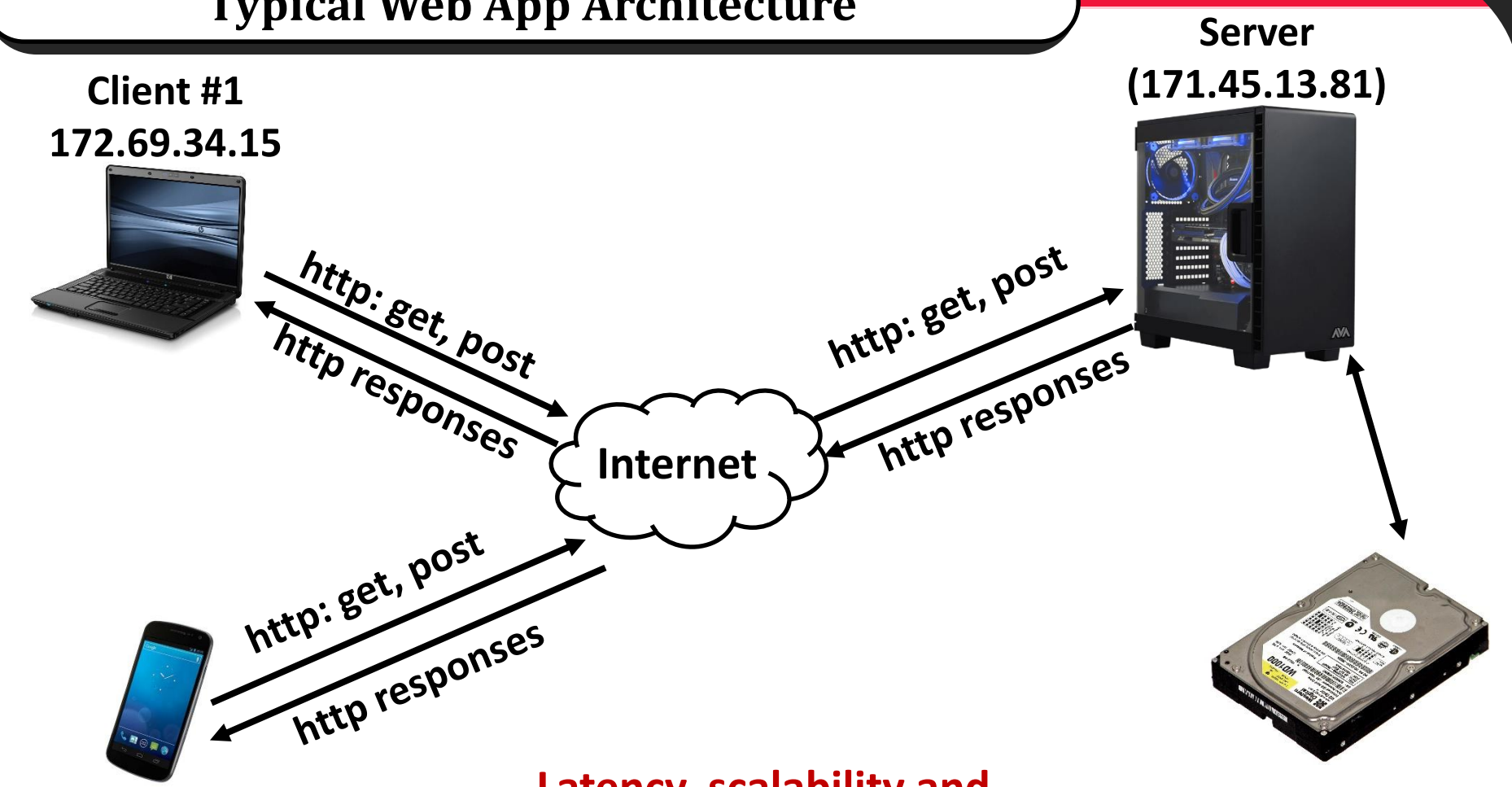
A client-server decoupling

Many simultaneous users

Multiple interacting resources/computers

A large amount of event-driven programming

Typical Web App Architecture



Latency, scalability and uncertainty become important issues

Typical Web App Architecture

All of the client/serve communication and data transfer is facilitated by HTTP

It provides a way for clients to specify what they want and a way for servers to respond appropriately

A relatively simple protocol, combined with data organization, leads to the complexity that is 'the web'

Things to Think About

Some important things to be aware of:

Requesting any new data is a round-trip

There are MANY requests happening

Asynchronous processing is good

Moving processing to the client is good

Any data you can avoid sending is good

Consider difference between mouse-click events on a desktop app and a web app

Important Technologies

Client-Side: HTML5, CSS, Javascript

Intermediary: HTTP

Server-Side: Node.js, NPM, Javascript, MongoDB

There are many other technologies available, but these are what we will look at in this course

HTTP – Hyper Text Transfer Protocol

The protocol that underlies the web

Who knows a bit about HTTP? What happens when you type an address into your browser or click a link?

HTTP Basics

Uses a request/response model
(e.g., you request a resource, server sends response)

Requests and responses contain 'headers', which specify details about the request/response

They may also contain a 'body', which makes up the main content of the request/response

HTTP is stateless

**The server does not remember information about the clients that are making requests*
(*as far as HTTP is concerned…)**

All information required for the request has to be specified in the request itself

Why is this a good thing?

Benefits of statelessness

With a stateless protocol, ANY server can handle ANY request from ANY client - all requests are independent

There is less overhead – the server does not need to store information about the connection

It handles failure well – the server is not concerned with the client once the request has been handled

All of these result in scalability!

HTTP Basics

HTTP protocol uses plain text for requests/responses

Note: the content of the body can be binary, but the HTTP parts are in plain text

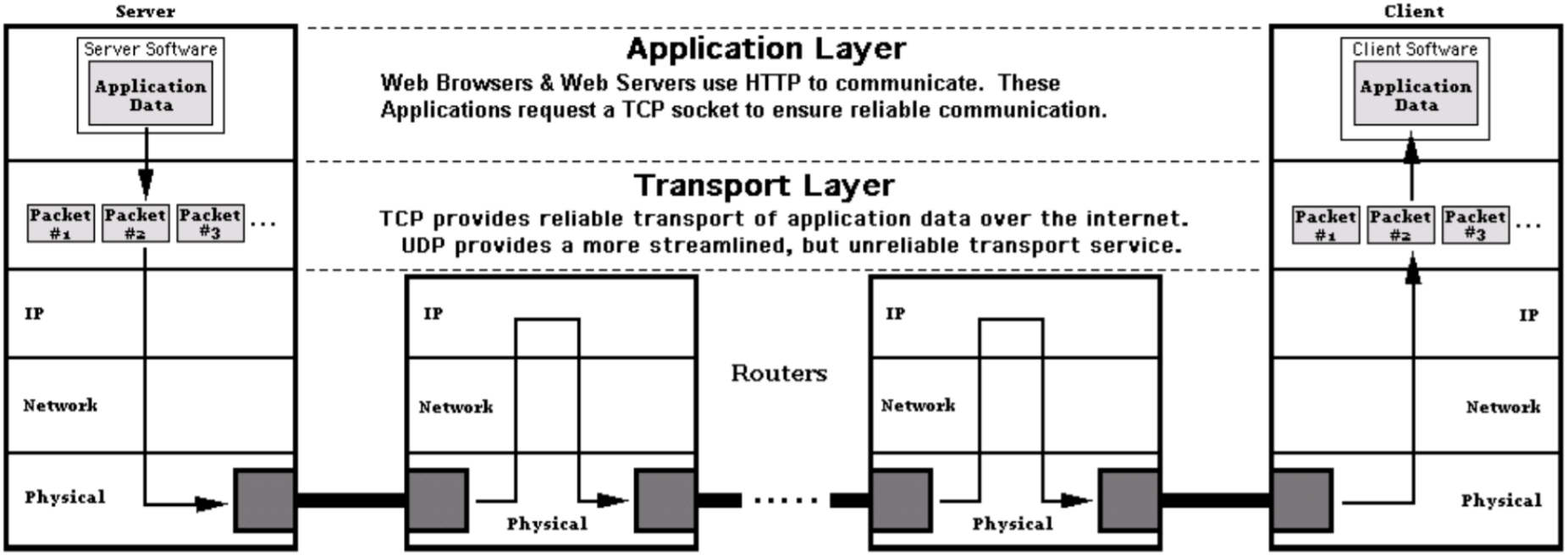
Plain Text – The Good and Bad

The good about plain text: it can be read easily by a human (e.g., for easy debugging)

The bad about plain text: it can be read easily by a human (e.g., somebody with bad intentions)

There are ways to address this (e.g., HTTPS)

HTTP – An Application Layer Protocol



Other Protocols at Work

There are other protocols involved, other than HTTP

One of which is DNS

Used for mapping domain names to IP addresses

It is a lot easier to remember carleton.ca than it is to remember 134.117.6.162

More About Domain Names

Domain name example:
people.scs.carleton.ca

Smallest → Largest

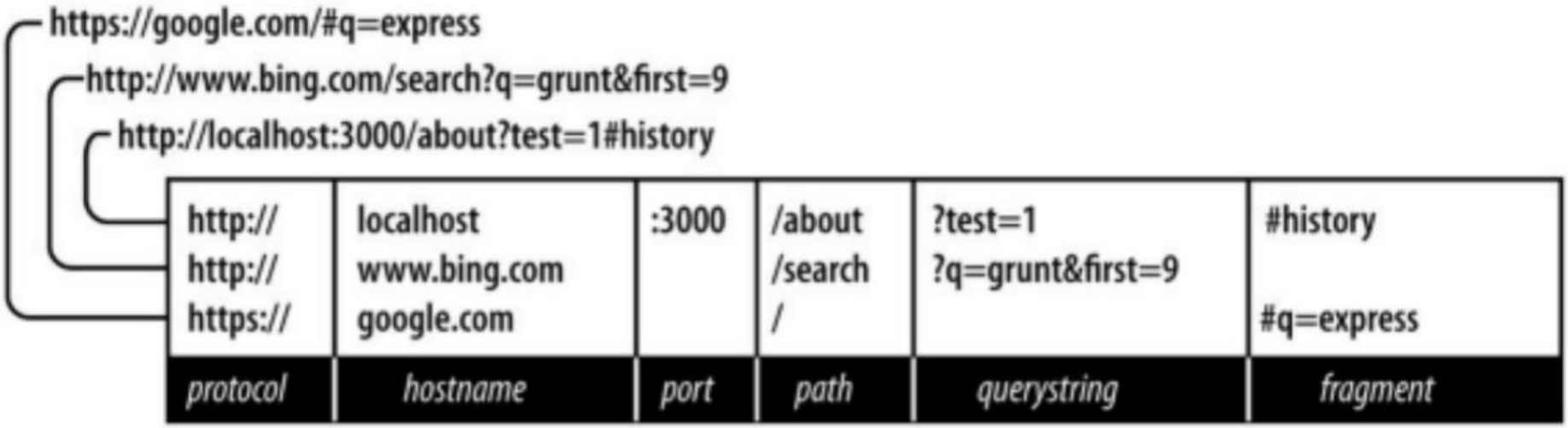
Most Specific → Most General

Uniform Resource Locators (URLs)

We use URLs to represent the location and name of a resource on the web

This allows us to request specific resources

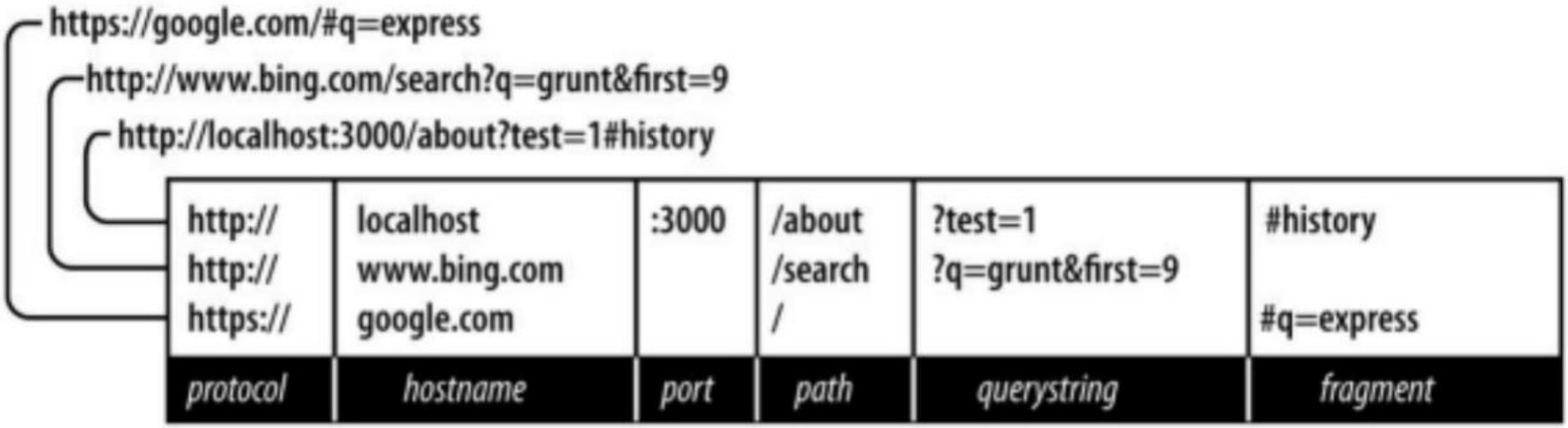
Parts of a URL



A single URL can be broken down into many parts, which gives us some information about the resource/request

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

Parts of a URL



The protocol determines how the request will be transmitted. We will use HTTP and (maybe) HTTPS. Other common protocols are file and FTP.

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

Parts of a URL

https://google.com/#q=express

http://www.bing.com/search?q=grunt&first=9

http://localhost:3000/about?test=1#history

protocol	hostname	port	path	querystring	fragment
http://	localhost	:3000	/about	?test=1	#history
http://	www.bing.com		/search	?q=grunt&first=9	
https://	google.com		/		#q=express

The hostname specifies the server. Locally, this may be a word (e.g., localhost) or IP address. In general, it will be a full domain name, ending in a top level domain (.com, .ca, .net, etc.)

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

Parts of a URL

https://google.com/#q=express

http://www.bing.com/search?q=grunt&first=9

http://localhost:3000/about?test=1#history

protocol	hostname	port	path	querystring	fragment
http://	localhost	:3000	/about	?test=1	#history
http://	www.bing.com		/search	?q=grunt&first=9	
https://	google.com		/		#q=express

The port allows information to be directed to a specific listener within the server. A default port of 80 is used if we do not specify one (this is the default HTTP port). You should use a port number > 1023.

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

Parts of a URL

https://google.com/#q=express

http://www.bing.com/search?q=grunt&first=9

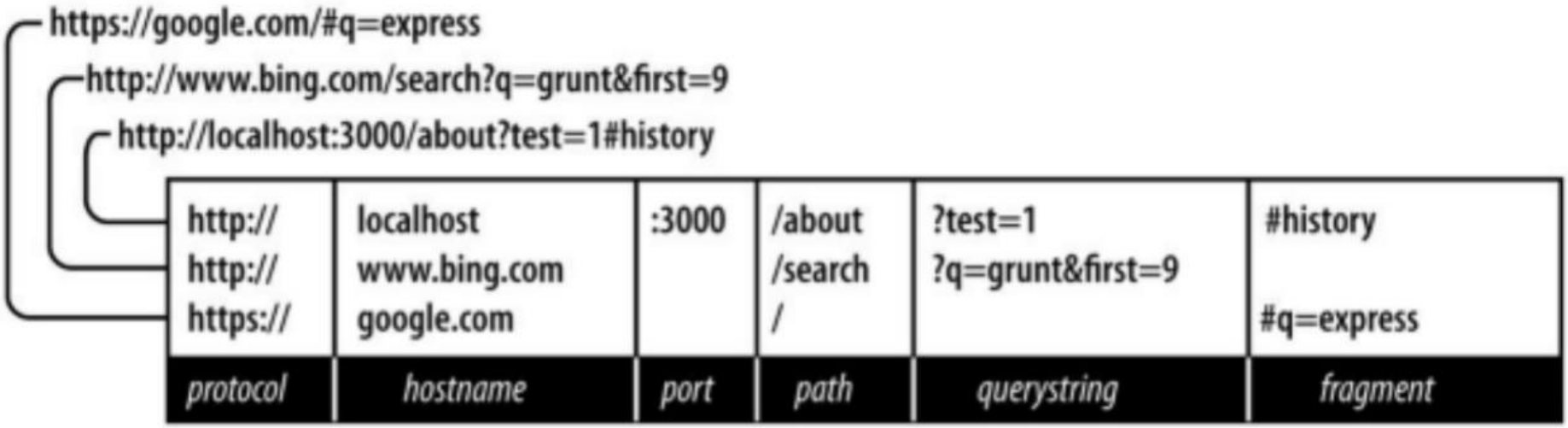
http://localhost:3000/about?test=1#history

protocol	hostname	port	path	querystring	fragment
http://	localhost	:3000	/about	?test=1	#history
http://	www.bing.com		/search	?q=grunt&first=9	
https://	google.com		/		#q=express

The path is generally the first part of the URL your server application cares about. Uniquely identifies pages or resources within your app.

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

Parts of a URL



The query string is an optional collection of key/value pairs. It starts with a ? and pairs are separated with &. This string should be 'URL encoded', which replaces spaces and special characters. Javascript has encodeURIComponent(str) to do this.

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

Parts of a URL

https://google.com/#q=express

http://www.bing.com/search?q=grunt&first=9

http://localhost:3000/about?test=1#history

protocol	hostname	port	path	querystring	fragment
http://	localhost	:3000	/about	?test=1	#history
http://	www.bing.com		/search	?q=grunt&first=9	
https://	google.com		/		#q=express

The fragment is not passed to the server. Its original intent was to allow markers within the page to be 'jumped' to by the browser.

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

HTTP Requests

An HTTP request requires the following format:

1. **A request line**
2. **Zero or more headers**
3. **A blank line (why?)**
4. **An optional message body**

HTTP Request Example

Method Resource HTTP Version

```
GET /public/index.html HTTP/1.1
Host: www.davemckenney.ca
Accept: text/html, */*
Accept-Language: en-us
Accept-Encoding: gzip
User-Agent: Mozilla/4.0
```

Request Headers

Blank Line

Request Body

Note: get requests do not generally have a body

HTTP Request Example

Method Resource HTTP Version

```
POST /public/profiles.html HTTP/1.1
Host: www.davemckenney.ca
Accept: text/html, */*
Accept-Language: en-us
Accept-Encoding: gzip
User-Agent: Mozilla/4.0

key1=value1&otherkey=anothervalue
```

Request Headers

Blank Line

Request Body

Common HTTP Request Methods

Common request methods include:

GET – retrieve a document

POST – sending data to the server to create/update

HEAD – like GET, but retrieve just the headers

PUT – store a new resource or replace one

DELETE – remove a resource

(GET and POST are most common)

More About GET Requests

Example GET URL:
/test/demo_form.php?name1=val1&name2=val2

Query strings are passed in the URL

https://www.w3schools.com/tags/ref_httpmethods.asp

More About GET Requests

GET requests:

Can be cached
Remain in the browser history
Can be bookmarked
Have length restrictions
Are only used to request data (not modify)
Should never be used when dealing with sensitive data (why?)

https://www.w3schools.com/tags/ref_httpmethods.asp

More About POST Requests

Example POST:

```
POST /test/demo_form.php HTTP/1.1
Host: w3schools.com

name1=value1&name2=value2
```

Data is stored in the body

https://www.w3schools.com/tags/ref_httpmethods.asp

More About POST Requests

POST Requests:

Are never cached

Do not remain in the browser history

Requests cannot be bookmarked

Requests have no restrictions on data length

https://www.w3schools.com/tags/ref_httpmethods.asp

Common Request Headers

Some common request headers:

Accept – specify what content type(s) to accept
Accept-Encoding – specify what compression
Authorization – used for HTTP authentication
Content-Length – the length of the body
Content-Type – the content type within the body
User-Agent – requesting user agent (e.g., browser)

There are many others as well

HTTP Response

Once a server receives a request, it typically does some processing and sends back a response

Like the request, this contains headers indicating properties of the response/data, as well as a body containing the response data

HTTP Response Example

HTTP Version Status Code Status Text

```
HTTP/1.1 404 Not Found.
Date: Fri, 6 Sep 2019 11:15:25 GMT
Server: Apache/1.2.13 (Linux)
Last-Modified: Thu, 5 Sep 2019
Content-Length: 1883
Content-Type: text/html

<html>…<html>
```

Response Headers

Blank Line

Response Body

Note: In general, Content-Type should always be specified

HTTP Response Status Codes

Status codes indicated something about the transaction:

1xx – Informational

2xx – Great success!

3xx – Redirection

4xx – Client error

5xx – Server error

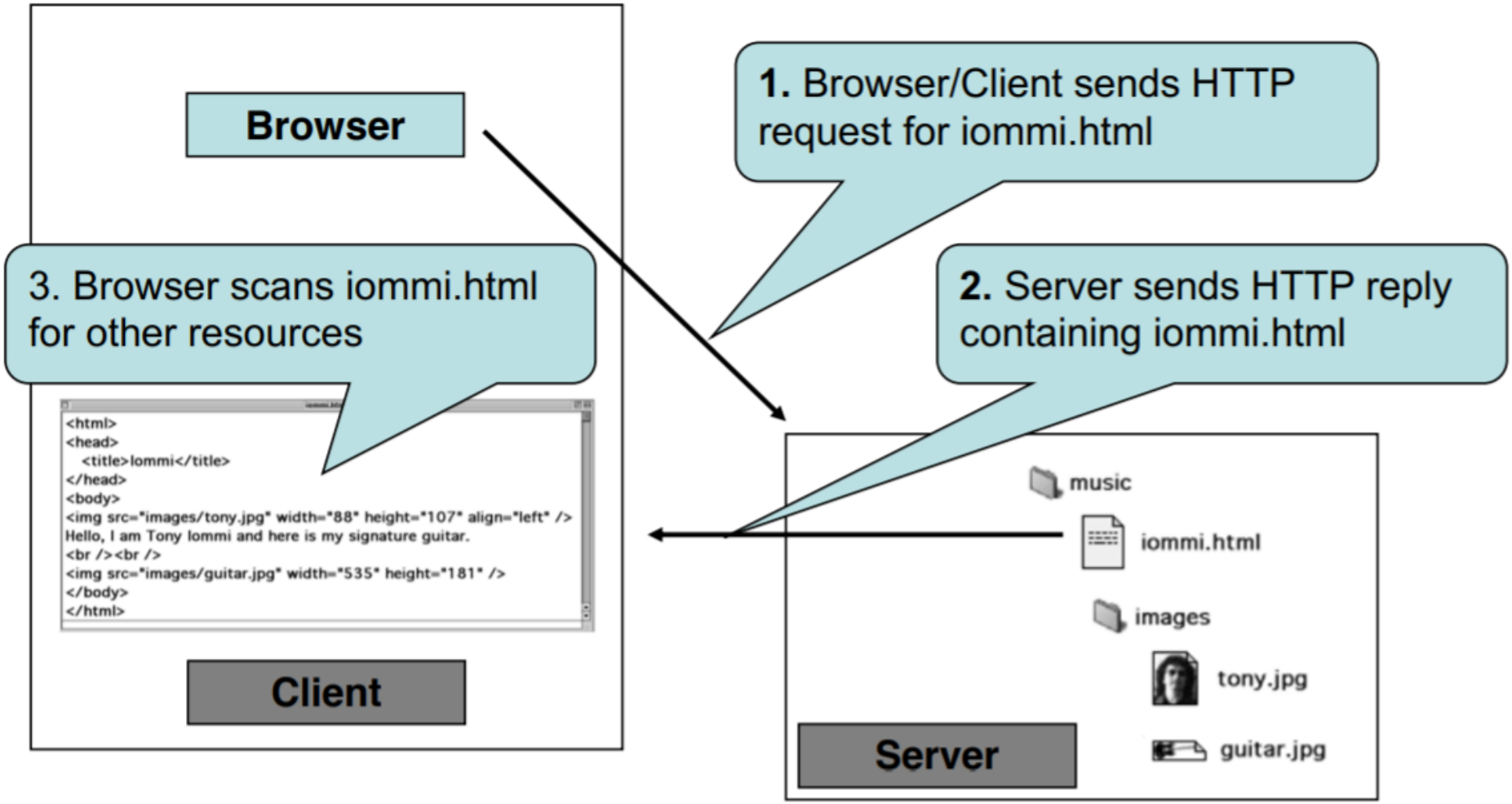
Code: machine readable, Text: human readable
(https://www.restapitutorial.com/httpstatuscodes.html)

Common Response Headers

Common response headers:

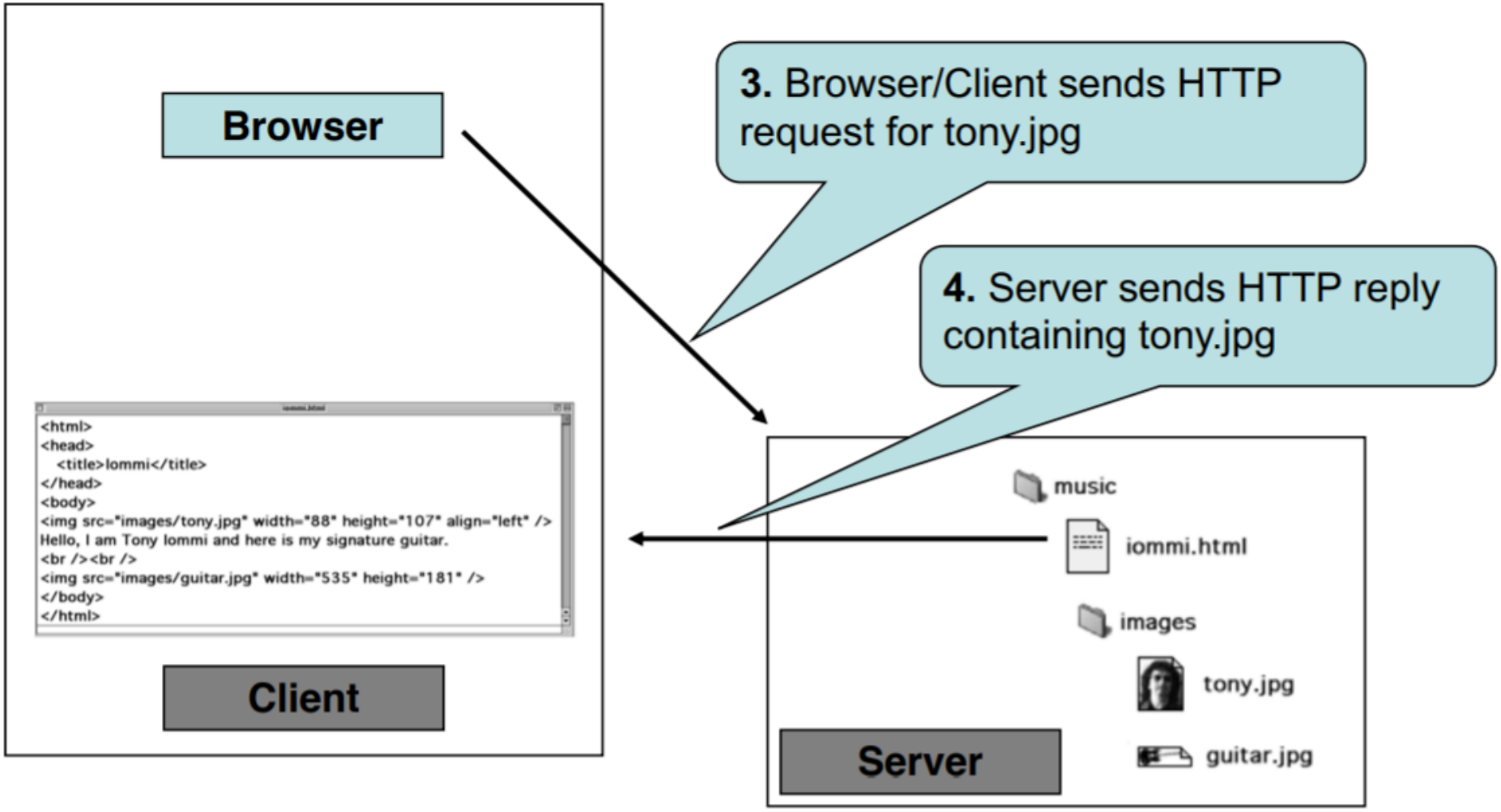
Content-Type – type of content in the response body
Content-Length – length of the response body
Cache-Control – the type of caching allowed
Content-Encoding – the compression used
Expires – the date at which cached data expires
Last-Modified – the last time the data was modified

Basic HTTP Interaction



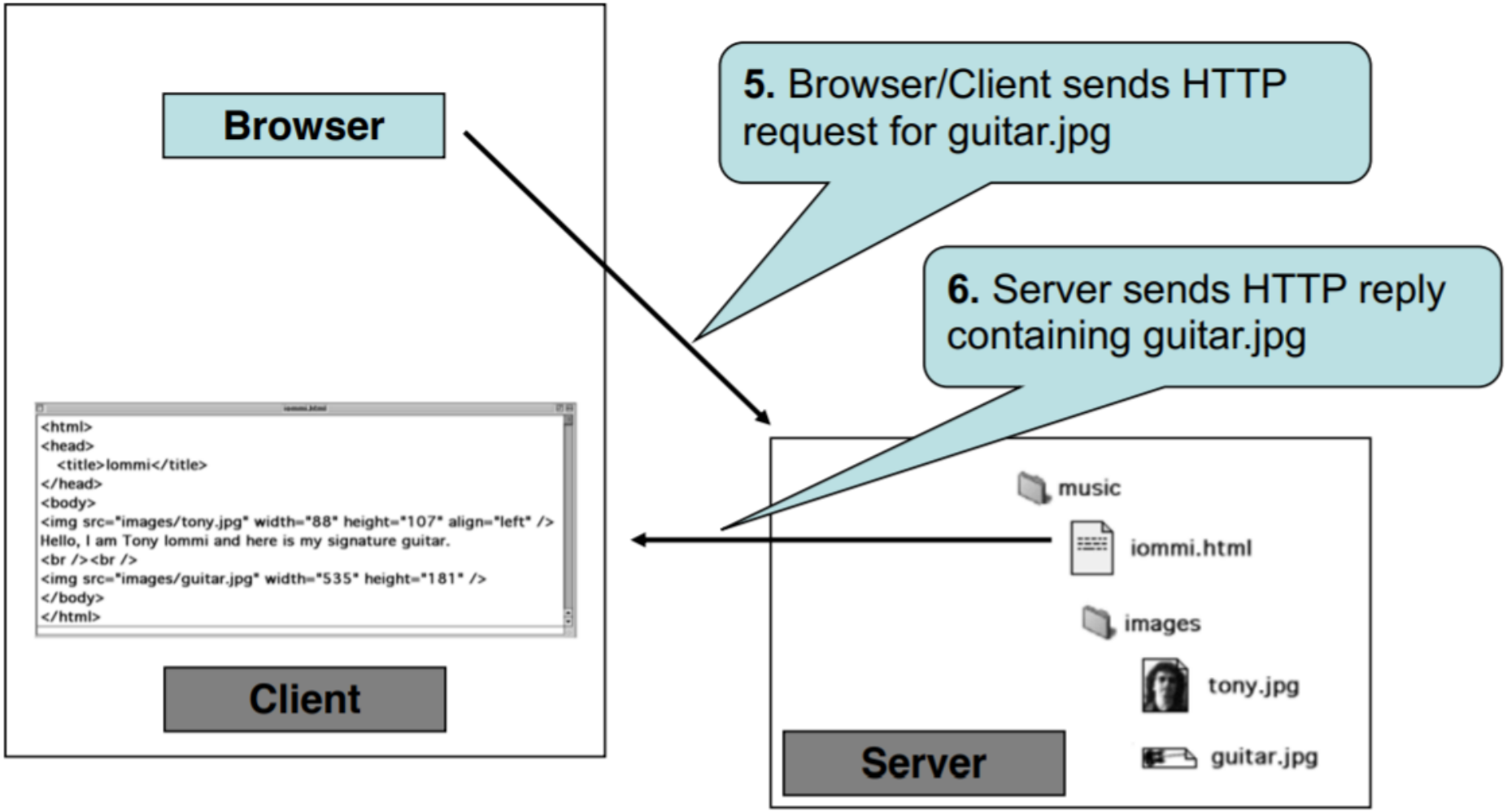
Example from Jason Morrison's COMP 2405 Notes

Basic HTTP Interaction



Example from Jason Morrison's COMP 2405 Notes

Basic HTTP Interaction



Example from Jason Morrison's COMP 2405 Notes

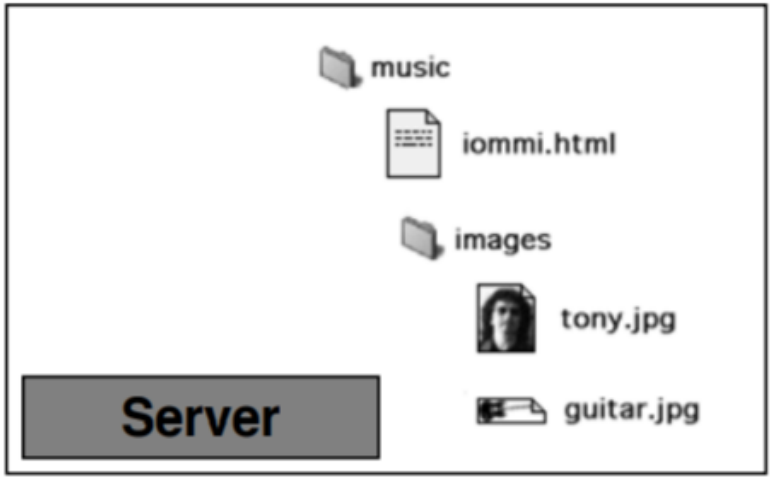
Basic HTTP Interaction


```
<html>
<head>
  <title>Iommi</title>
</head>
<body>
<img src="images/tony.jpg" width="88" height="107" align="left" />
Hello, I am Tony Iommi and here is my signature guitar.
<br /><br />
<img src="images/guitar.jpg" width="535" height="181" />
</body>
</html>
```

Client

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



Example from Jason Morrison's COMP 2405 Notes

Another HTTP Example

Another example: the index.html example hosted on the local machine

Another example: http://www.google.ca

HTTP Caching

Avoiding the transmission of information is a good thing when possible

HTTP has a caching mechanism to facilitate this

Caching involves storing information on the client so that it does not need to be transmitted many times

HTTP Caching

Many images, CSS stylesheets, Javascript files, etc., do not change very often

After sending once, it is inefficient to send on any subsequent request if they have not changed

Another example is having the same image on a page more than once (this is cached automatically)

HTTP Caching

Servers can control caching of the content they serve

This is done using the response headers

Cache-Control: no-cache/private/public/no-store

Defaults to public if nothing specified

HTTP Caching

no-cache: the content cannot be re-used without verifying with server

private: the content can be cached on the local machine

public: the content can be cached in public caches (e.g., proxy servers, CDNs)

no-store: the content cannot be cached (e.g., for private data you don't want to stick around)

HTTP Caching

You can also set the maximum age the content should be cached for:

Cache-Control: private, max-age=3600

This content can be cached for 1 hour on the local machine (1 year is the most you should specify)

HTTP Caching

There is also a header to specify the date the content was last modified:

Last-Modified: Wed, 25 Feb 2015 12:00:00 GMT

And when the content 'expires':

Expires: Thu, 25 Feb 2016 12:00:00 GMT

With cached data, it is possible to make 'conditional requests'

Conditional Request

A client can include the request header:

If-Modified-Since: Thu, 25 Feb 2016 12:00:00 GMT

If content has not been changed since given date, the server responds with status code 304 – unmodified (Only headers are sent, the data is not re-sent)

Authentication

Authentication is another useful aspect of the HTTP protocol

We will look at this later in the course

Questions?

Questions?