Edinburgh Napier
UNIVERSITY

ADVANCED WEB TECHNOLOGIES
SET09103
TOPIC #03

METHODS & STATUS CODES

Dr Simon Wells
s.wells@napier.ac.uk
http://www.simonwells.org

Edinburgh Napier
UNIVERSITY

OVERVIEW

- Part #1: HTTP Methods & HTTP Status Codes.
- Part #2: Using methods & status codes.

Edinburgh Napier UNIVERSITY

TL/DR

HTTP =

request (address+method)

+

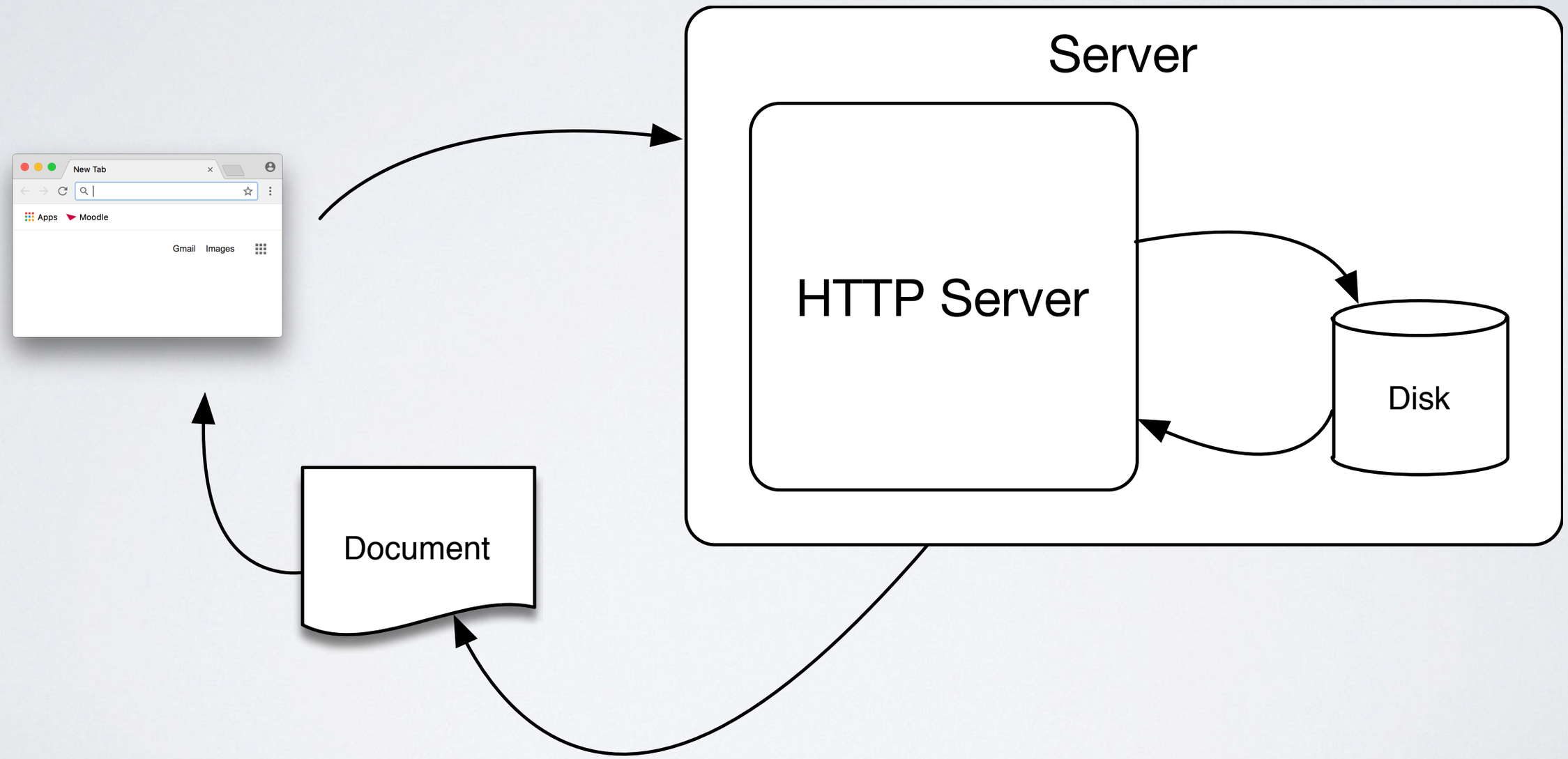
response (content+status code)

Edinburgh Napier
UNIVERSITY

PART #1: REQUESTS & RESPONSES

CLIENT SERVER ARCHITECTURES

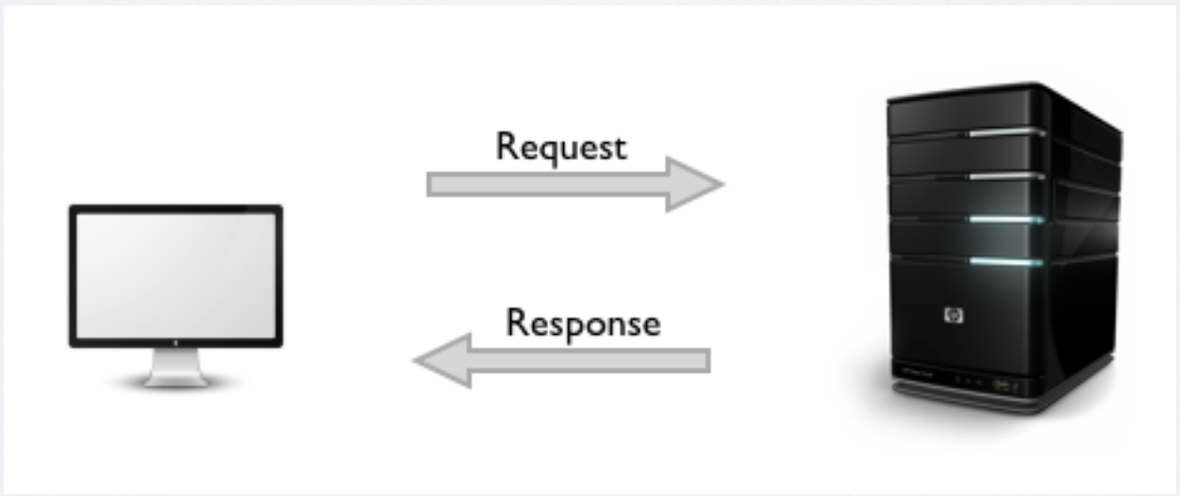
Edinburgh Napier UNIVERSITY



Edinburgh Napier UNIVERSITY

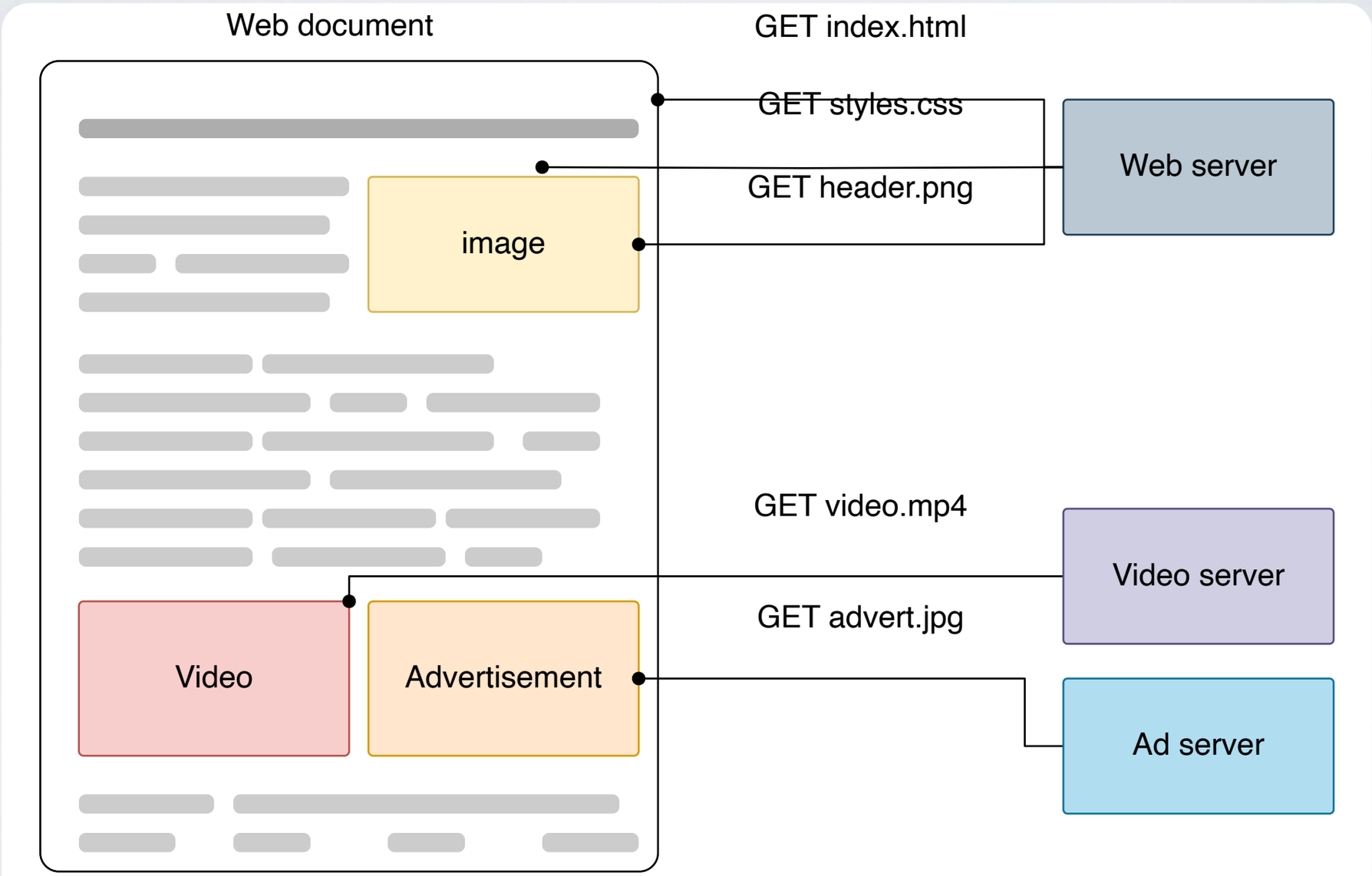
REQUESTS & RESPONSES

- Communication occurs between a **host (server)** and a **client** via a **request/response** pair
- Client initiates HTTP **request** message and the host *services* the request and returns a **response** message



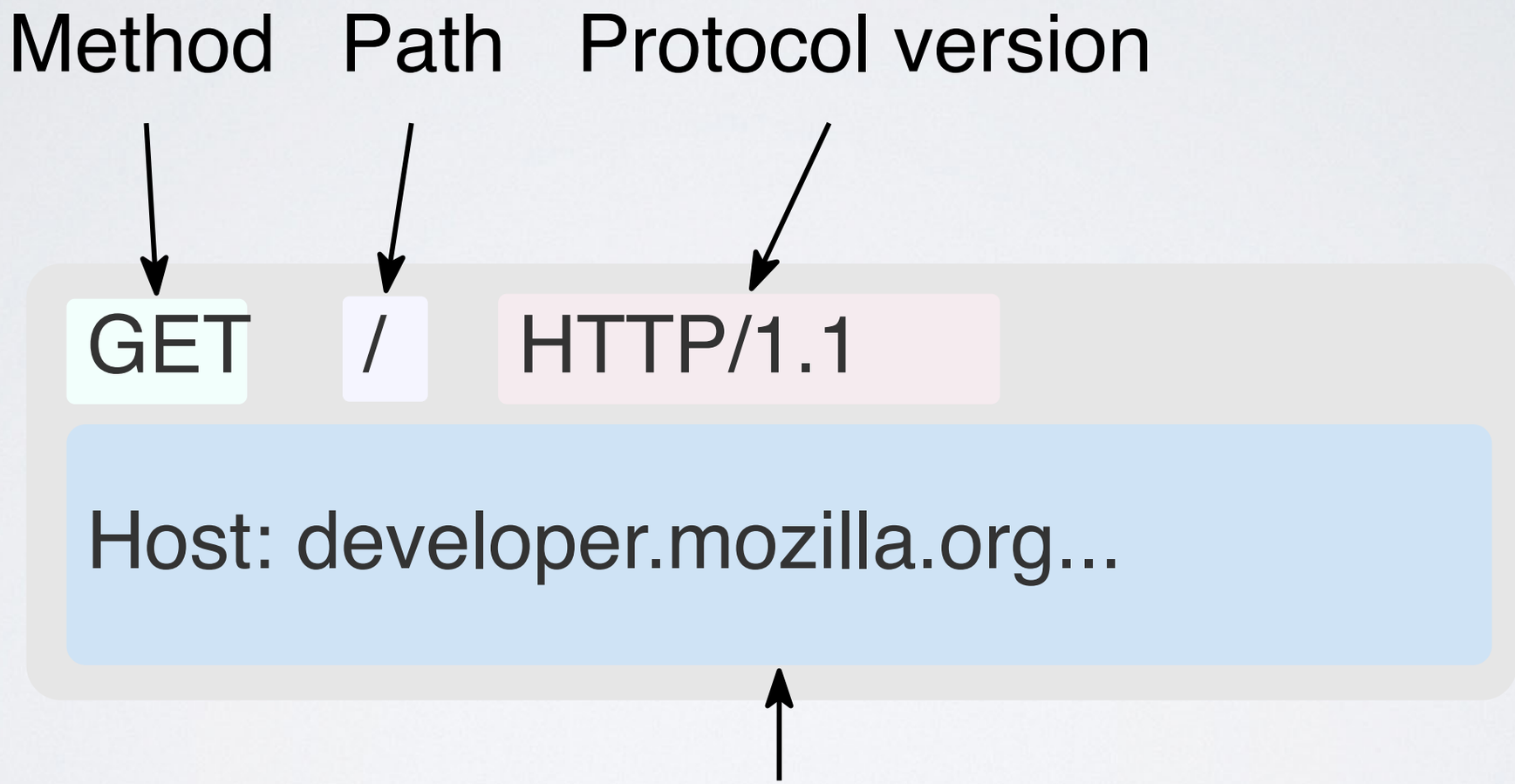
Edinburgh Napier UNIVERSITY

HTTP: A PROTOCOL FOR FETCHING RESOURCES



Edinburgh Napier
UNIVERSITY

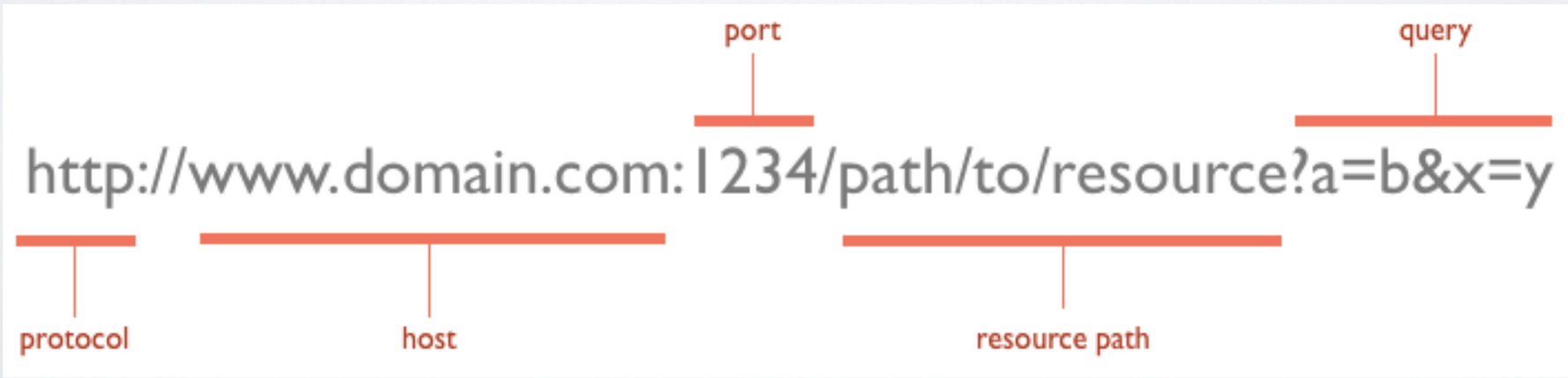
HTTP REQUESTS



Edinburgh Napier UNIVERSITY

URLS & PATHS

- The request message is the heart of communications via the web
- Requests are sent via **Uniform Resource Locators (URLs)**
- **Protocol** is usually http (but can also be https)
- **Port** defaults to 80 but can be set explicitly in the URL
- **Resource Path** is the local path to the resource on the server



Edinburgh Napier UNIVERSITY

HTTP METHODS

- URLs identify resources on the server with which we can interact.
- By default we **GET** resources (Web as data retrieval).
- But other **methods** are available to interact with resources in different ways.
- HTTP methods are **verbs** - They describe actions.
- HTTP formalises a set of verbs that capture the essentials of communication:. The four most popular are:
  - **GET** fetch an existing resource
  - **POST** - create a new resource. Post requests always carry a payload that specifies the data for the new resource
  - **PUT** - update an existing resource. Payload must contain the updated data for the resource
  - **DELETE** - remove the resource
- Browsers support GET & POST by default.
- PUT & DELETE are ofter repackaged as POST requests where the payload makes explicit the action to perform, e.g. creating, updating, or deleting
- **JavaScript Fetch API** is used in browsers to execute other HTTP methods.

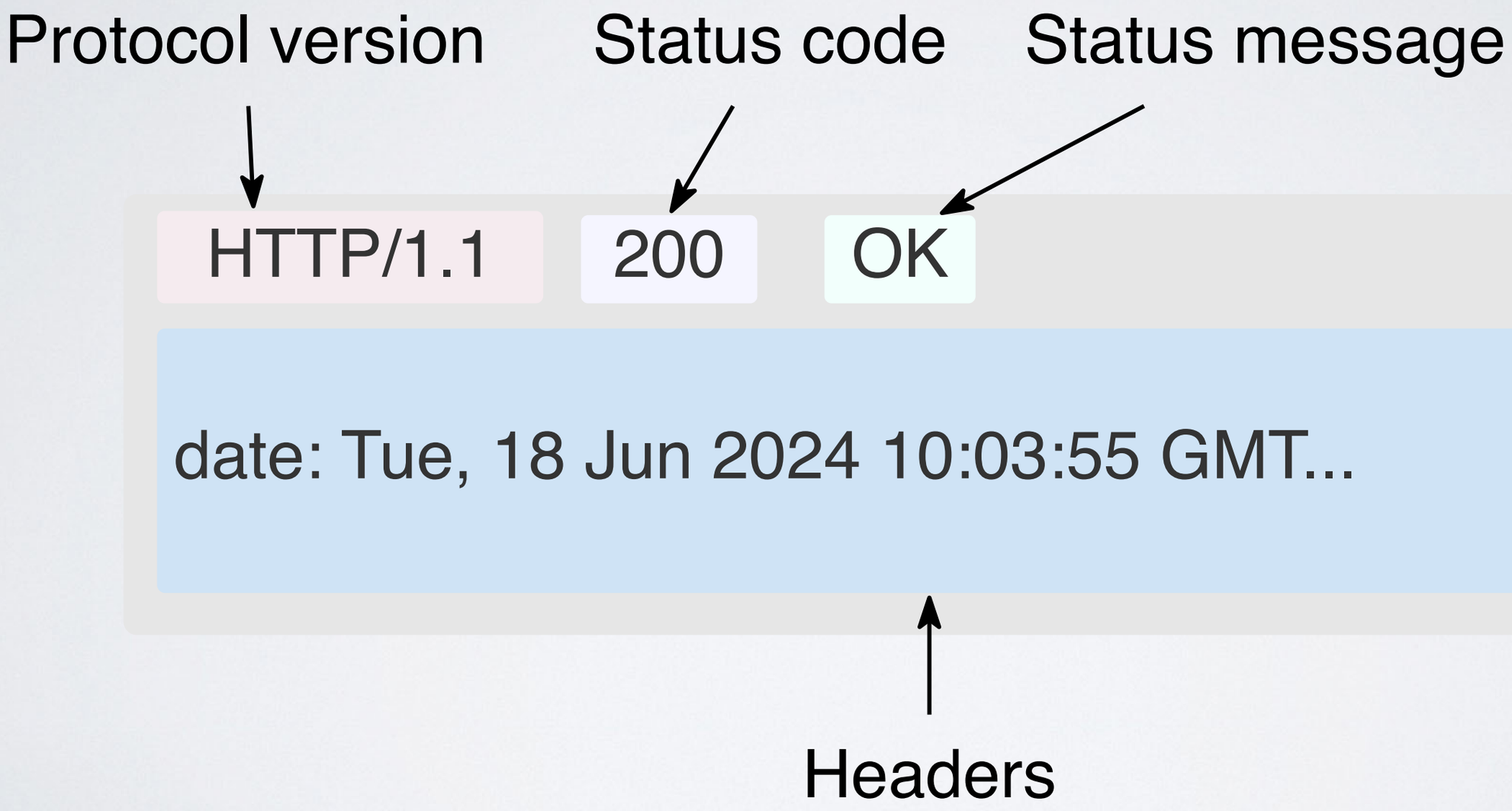
Edinburgh Napier UNIVERSITY

OTHER HTTP METHODS

- Most APIs will support GET, POST, PUT, DELETE
    - *Why support verbs that browsers can't make use of?*
- Other methods are also available, e.g.
    - ***HEAD*** - The same as GET but retrieves just the server headers for the resource without the message body - useful to check whether a large resources has changed (using timestamps) so that you don't retrieve it all
    - ***TRACE*** - retrieve hops that the request made during round trip to server, e.g. each intermediate proxy or gateways injects its IP address of DNS name into the *Via* field of the header (useful for diagnostics)
    - ***OPTIONS*** - to get information about the host/API capabilities
- URLs+Verbs provide a good proportion of an API for the host that we are trying to communicate with

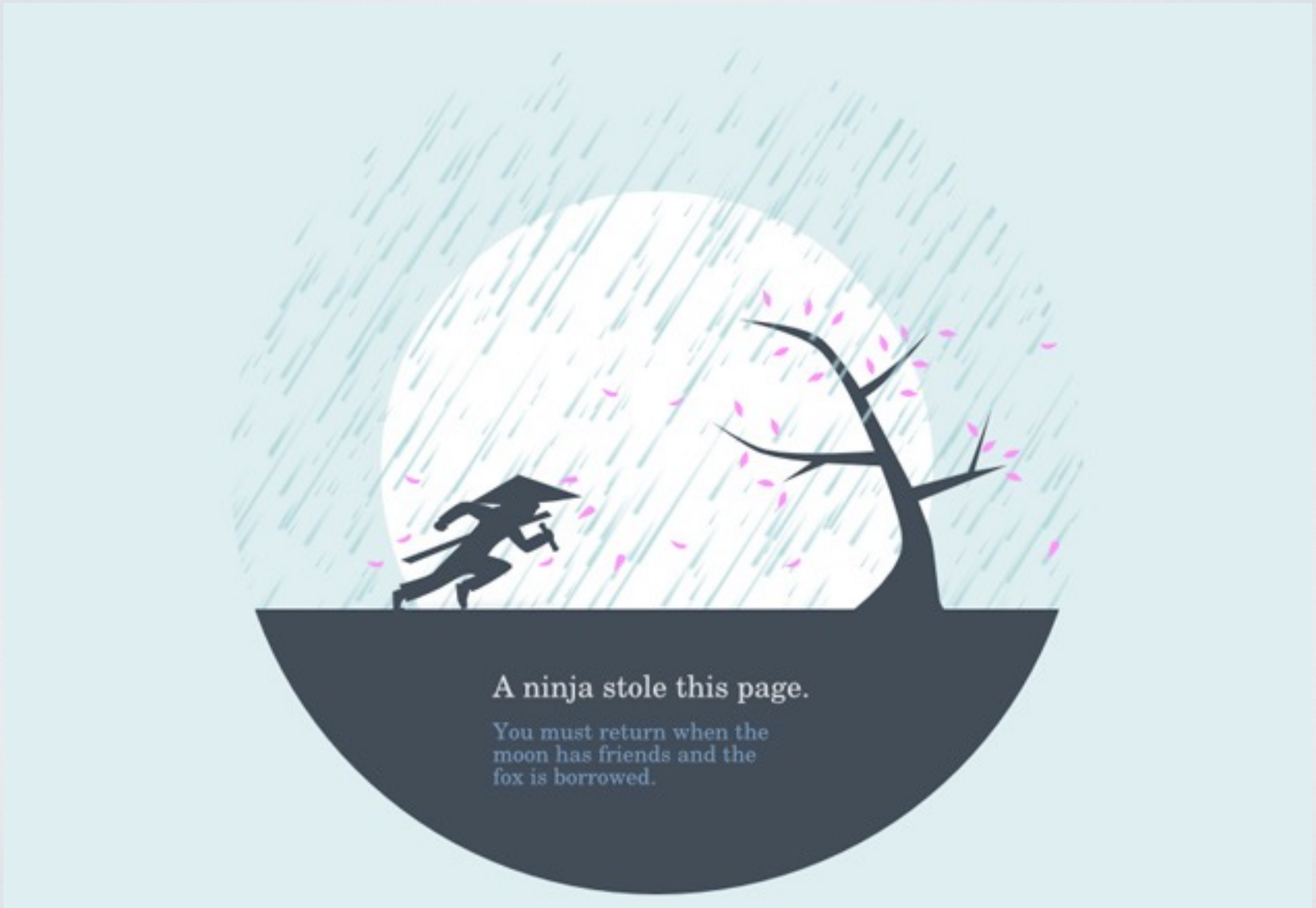
https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Methods

HTTP RESPONSES



Edinburgh Napier
UNIVERSITY

RESPONSE/STATUS CODES



Edinburgh Napier UNIVERSITY

HTTP STATUS CODES

1. Make request to a server,
2. Server processes request,
3. Server returns response & status code:

Request + Verb > Response + Status Code

- The status code is important - it tells the client how to interpret the response from the server.

https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Status

1XX: INFORMATIONAL MESSAGES

- Introduced in HTTP/1.1
- Ignored by HTTP/1.0 clients
- **100 Continue** - tell the client to continue sending the remainder of the request or ignore if it has already been sent.
- **101 Switching Protocols** - tell the client the protocol that the server has switched to.

Edinburgh Napier
UNIVERSITY

2XX: SUCCESSFUL

- Used by server to tell the client that the request was successfully processed.
- **200 OK** - Probably the most common code (except for 404) but as users we don't notice it when things are going well
- **202 Accepted** - the request was accepted but may not include the resource in the response. This is useful for async processing on the server side. The server may choose to send information for monitoring.
- **204 No Content** - there is no message body in the response.
- **205 Reset Content** - indicates to the client to reset its document view.
- **206 Partial Content** - indicates that the response only contains partial content. Additional headers indicate the exact range and content expiration information.

Edinburgh Napier UNIVERSITY

3XX: REDIRECTION

- Require the client to take additional action
  - e.g. request a different url if the location for a resource has changed
- Help provide stability if used correctly
- **301 Moved Permanently** - the resource is now located at a new URL.
- **303 See Other** - the resource is temporarily located at a new URL. The Location response header contains the temporary URL.
- **304 Not Modified** - the server has determined that the resource has not changed and the client should use its cached copy. This relies on the fact that the client is sending ETag (Entity Tag) information that is a hash of the content. The server compares this with its own computed ETag to check for modifications.

Edinburgh Napier
UNIVERSITY

4XX CLIENT ERROR

- Used when server believes the client is at fault, e.g. requesting an invalid resource or making a bad request
- **400 Bad Request** - the request was malformed.
- **401 Unauthorized** - the request requires authentication. The client can repeat the request with the Authorization header. If the client already included the Authorization header, then the credentials were wrong.
- **403 Forbidden** - the server has denied access to the resource.
- **404 Not Found** - the resource is invalid and does not exist on the server
- **405 Method Not Allowed** - invalid HTTP verb used in the request line, or the server does not support that verb.
- **409 Conflict** - the server could not complete the request because the client is trying to modify a resource that is newer than the client's timestamp. Conflicts arise mostly for PUT requests during collaborative edits on a resource.

Edinburgh Napier UNIVERSITY

5XX: SERVER ERROR

- Class of codes that indicate a server failure whilst processing the request

- **500 Internal Server Error** - general "something bad happened" code

- **501 Not Implemented** - server doesn't yet support the requested functionality

- **503 Service Unavailable** - something on the server has failed or the server is overloaded (*NB. often the server won't even respond & the request will timeout. Timeout can usually be interpreted as a 503*)

Edinburgh Napier UNIVERSITY

HTTPS

- Extension to HTTP for **secure** communication on the Internet
- The connection between the client & server is **encrypted** at the **transport layer** - can see where connections start & end but not there content - hence TLS (transport layer security)
- NB. Used to be Secure Sockets Layer (SSL)
- AIM:
  - Authentication of the site you're accessing
  - Protection of data transported between you & site (& vice versa)
    - i.e. prevent someone from eavesdropping on your messages or altering them en route

Edinburgh Napier UNIVERSITY

HTTP STRICT TRANSPORT SECURITY (HSTS)

- Extension to HTTPS to account for a possible security risk.
- Attempt to force greater security by preventing a class of attacks on HTTP/HTTPS sites
- Protects against protocol downgrade attacks/SSL-stripping/Man-in-the-middle attacks
- Because many sites use both HTTP & HTTPS it can be unclear whether the connection should be secure or not.
- Web server declares that it will **only** allow interaction using HTTPS connections and will **never** drop back to plain HTTP
- Communicated to client in HTTP response header field named "Strict-Transport-Security"

Edinburgh Napier UNIVERSITY

SUMMARY

- HTTP Methods.
- HTTP Status Codes.
- Requests & Responses
- HTTPS & HSTS

PART #2: PRACTICAL WORK

- **Example #1:** The greetings app with methods and status codes.

- **Example #2:** The serendipity app (from the Flask Project Book).

```
from flask import Flask, request
app = Flask(__name__)

@app.route("/account/", methods=['POST','GET'])
def account():
  if request.method == 'POST':
    print (request.form)
    name = request.form['name']
    return "Hello %s" % name
  else:
page ='''
<html><body>
  <form action="" method="post" name="form">
    <label for="name">Name:</label>
    <input type="text" name="name" id="name"/>
    <input type="submit" name="submit" id="submit"/>
  </form>
  </body><html>'''

return page
```

Edinburgh Napier UNIVERSITY

EXAMPLE #2

- Serendipity app:
  - Putting together everything we've seen so far into a single cohesive Web-app:
    - routes, methods, status codes, inline html, Python method calls
    - Also some things we haven't seen yet: Building and returning JSON instead of HTML.

Edinburgh Napier UNIVERSITY

WRAPPING UP

- More HTTP:
  - Methods, Status Codes in context of URLs, requests, & responses.
  - Some examples that bring these together.