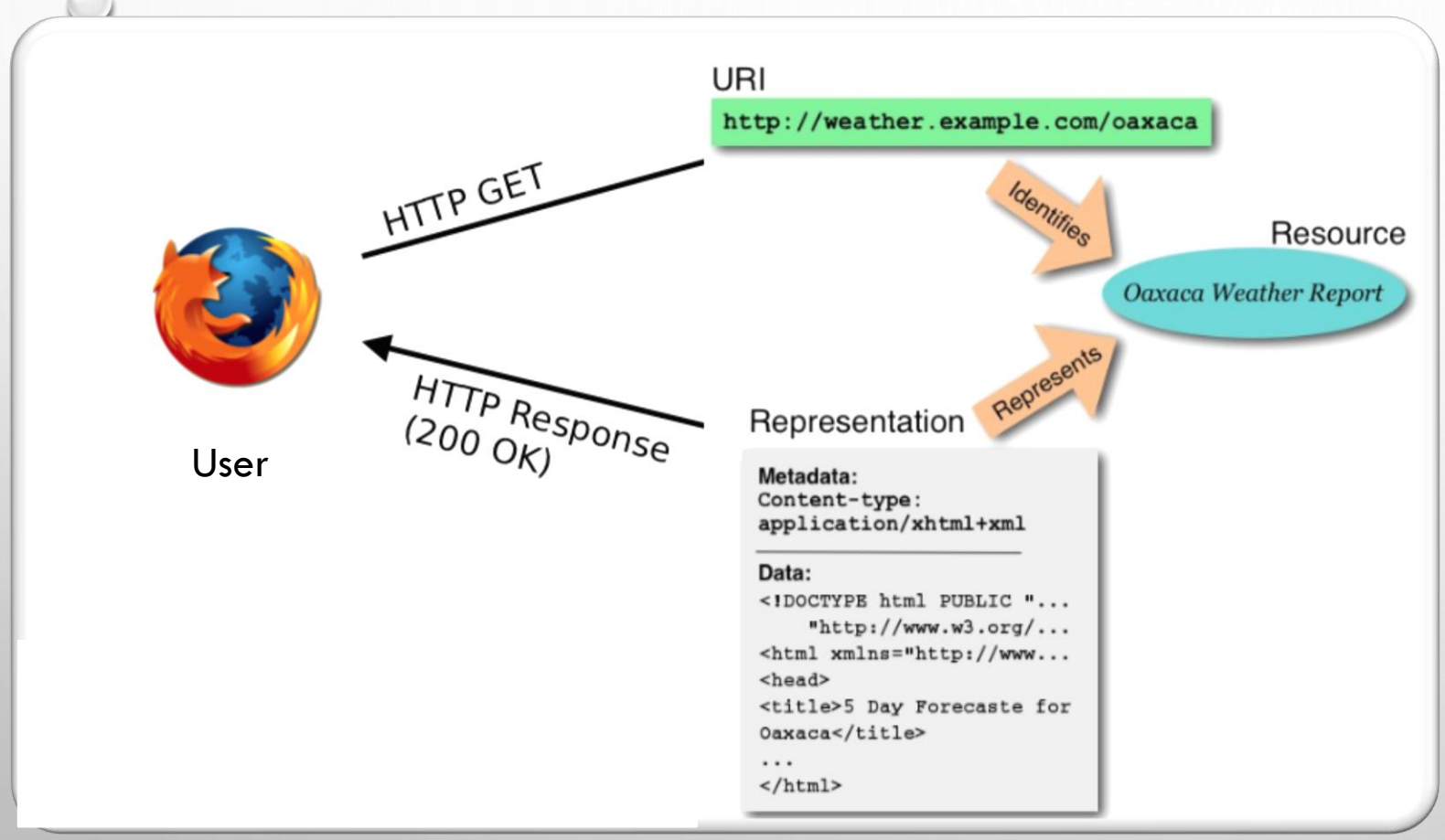
LECTURE02: WEB ARCHITECTURE AND WEB DEVELOPMENT FRAMEWORKS

CS418/518: Web programming

Nasreen Arif

By Courtesy: presentation slides from Dr. Justin Brunelle, Dr. Jian Wu



WEB ARCHITECTURE

HTTP REQUEST

```
$ curl -i -v http://www.odu.edu
*   Trying 128.82.112.29:80...
* Connected to www.odu.edu (128.82.112.29) port 80 (#0)
> GET / HTTP/1.1
> Host: www.odu.edu
> User-Agent: curl/7.79.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
* HTTP 1.0, assume close after body
< HTTP/1.0 301 Moved Permanently
HTTP/1.0 301 Moved Permanently
< Location: https://www.odu.edu/
Location: https://www.odu.edu/
< Server: BigIP
Server: BigIP
* HTTP/1.0 connection set to keep alive!
< Connection: Keep-Alive
Connection: Keep-Alive
< Content-Length: 0
Content-Length: 0

<
* Connection #0 to host www.odu.edu left intact
```

This tell me where I should go.

HTTPS REQUEST

```
$ curl -i -v https://www.odu.edu
* Rebuilt URL to: https://www.odu.edu/
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0*   Trying 128.82.112.29...
* TCP_NODELAY set
* Connected to www.odu.edu (128.82.112.29) port 443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
*  start date: Jun  5 00:00:00 2019 GMT
*  expire date: Jun  4 23:59:59 2021 GMT
*  subjectAltName: host "www.odu.edu" matched cert's "www.odu.edu"
*  issuer: C=GB; ST=Greater Manchester; L=Salford; O=COMODO CA Limited; CN=COMODO RSA Extended Validation Secure
Server CA
*  SSL certificate verify ok.
> GET / HTTP/1.1
> Host: www.odu.edu
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Tue, 19 Jan 2021 03:16:47 GMT
< Server: Apache/2.4.6 (Red Hat Enterprise Linux)
< Vary: Host, Accept-Encoding
< Accept-Ranges: bytes
< Connection: close
< Transfer-Encoding: chunked
< Content-Type: text/html; charset=UTF-8
< Set-Cookie: BIGipServerWEB_HTTPS_PROD.app~WEB_HTTPS_PROD_pool_int=rd741o00000000000000000000ffff8052619fo80;
path=/; Httponly; Secure
<
{ [575 bytes data]
100 94661    0 94661    0     0   320k      0 --:--:-- --:--:-- --:--:--  320k
* Closing connection 0
* TLSv1.2 (OUT), TLS alert, Client hello (1):
} [2 bytes data]
```

connection successful!

HTTP REQUEST AND RESPONSE

```
> GET / HTTP/1.1
> Host: www.odu.edu
> User-Agent: curl/7.79.1
> Accept: */*
```

request

```
< HTTP/1.1 200 OK
< Date: Mon, 29 Aug 2022 17:37:00 GMT
< Server: Apache/2.4.6 (Red Hat Enterprise Linux)
< Vary: Host, Accept-Encoding
< Accept-Ranges: bytes
< Connection: close
< Transfer-Encoding: chunked
< Content-Type: text/html; charset=UTF-8
```

Chunked transfer encoding (CTE) is a mechanism in which the encoder sends data to the player in a series of chunks. The player doesn't have to wait until the complete segment is available. CTE is available in HTTP 1.1.

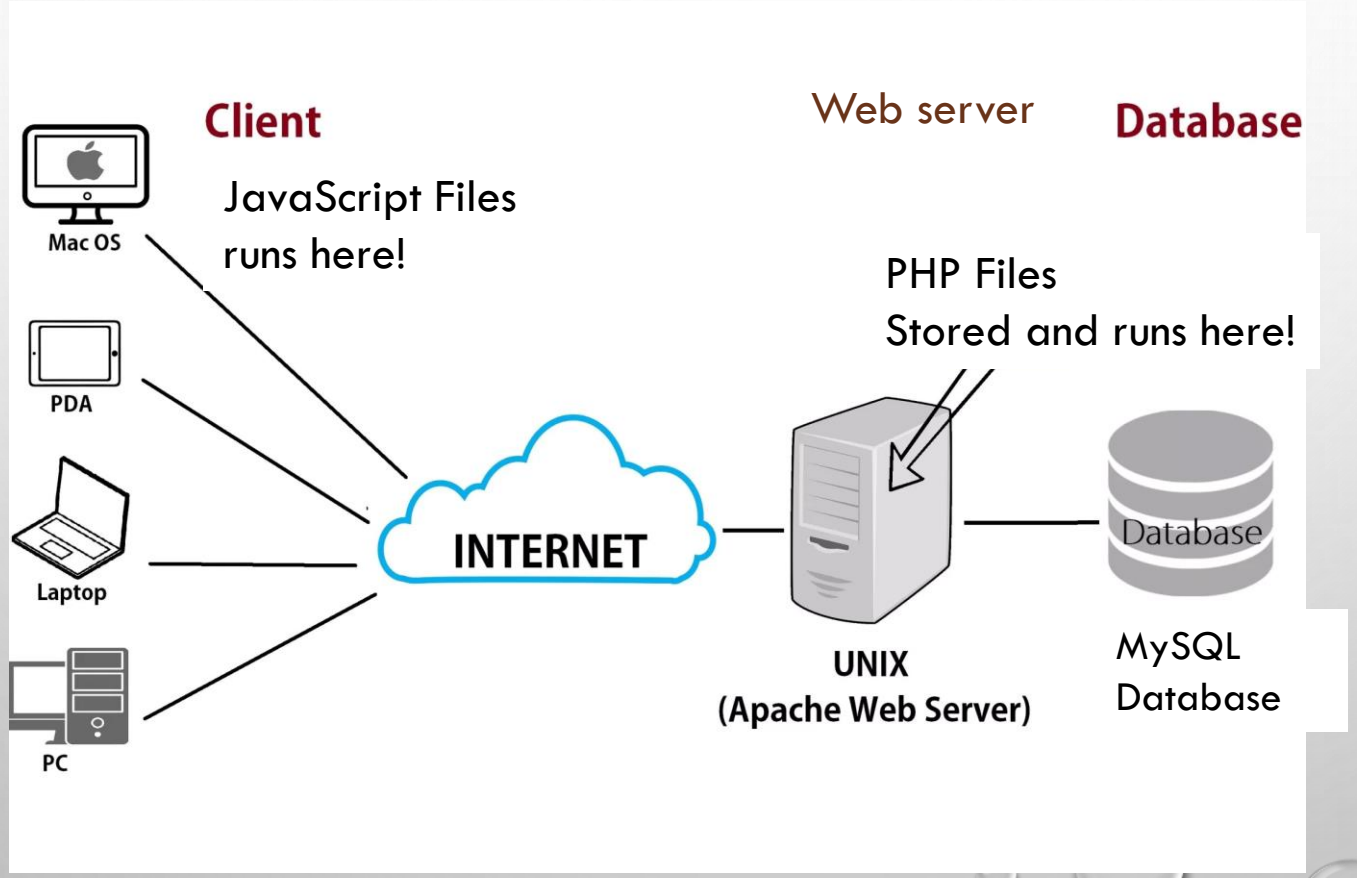
USER AGENT

```
> GET / HTTP/1.1
> Host: www.odu.edu
> User-Agent: curl/7.79.1>
> Accept: */*
```

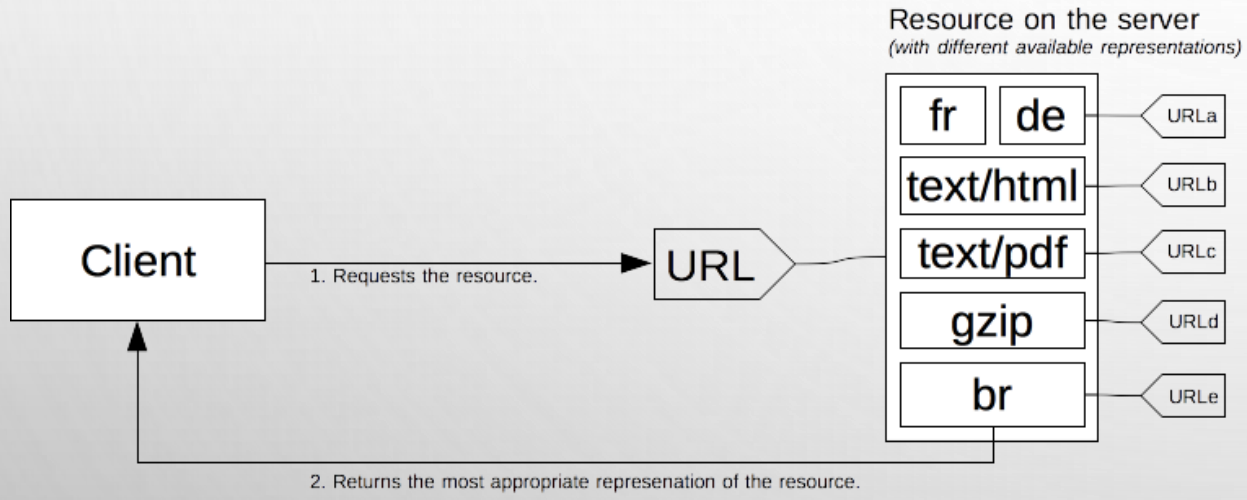
- Web browser, e.g., Mozilla, Chrome
- Command line, e.g., curl, wget
- Anything used to navigate the web, e.g., Googlebot
- Refer to this page to see how to view and change user-agent on the Google chrome browser (tip: you can disguise yourself by changing it)

WEB SERVER

- Handle requests from clients



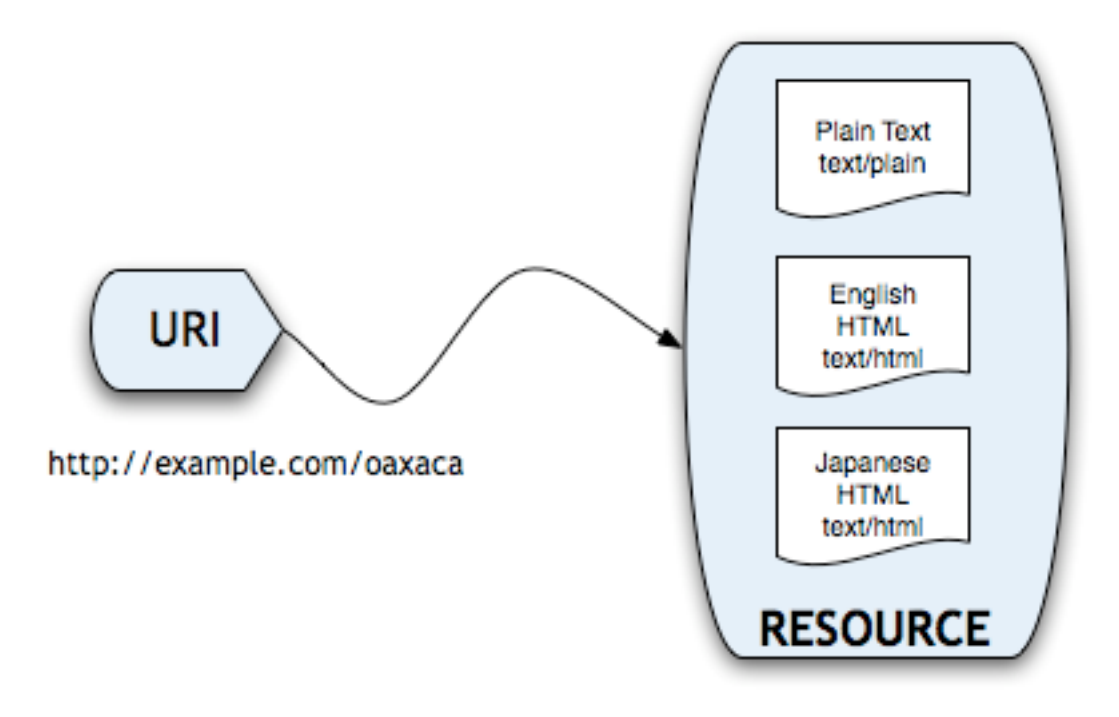
CONTENT NEGOTIATION



In HTTP, ***content negotiation*** is the mechanism that is used for serving different representations of a resource at the same URI, so that the user agent can specify which is best suited for the user (for example, which language of a document, which image format, or which content encoding).

The HTTP/1.1 standard defines list of the standard headers that start server-driven negotiation (Accept, Accept-Charset, Accept-Encoding, Accept-Language).

CONTENT NEGOTIATION EXAMPLES



Content Negotiation is a complex-sounding term for what is a rather simple mechanism.

Format Negotiation

Language Negotiation

Fore more information on how content negotiation works, here is an excellent blogpost:
https://www.w3.org/blog/2006/02/content-negotiation/

IDENTIFIERS

- Uniform Resource Identifier (URI): it is an identifier, not the file name
  - http://www.ietf.org/rfc/rfc2396.txt: identifies a file on the web
  - news:comp.infosystems.www.servers.unix: identifies a news site
  - https://foo.com/page.html#section2: identifies a content block on a webpage
  - In general: `<scheme>:<scheme-specific-part>`
- Uniform Resource Locator (URL) refers to the **location** of a particular file.
  - http://foo.com/page.html
  - ftp://foo.com/bar.tar.gz
  - mailto:John.Doe@example.com
- Uniform Resource Name (URN): A Uniform Resource Name (URN) is a URI that identifies a resource by name in a particular namespace.

See https://stackoverflow.com/questions/4913343/what-is-the-difference-between-uri-url-and-urn

URN

The **International Standard Book Number** (**ISBN**) is a numeric commercial book identifier which is intended to be unique. Publishers purchase ISBNs from an affiliate of the International ISBN Agency.

International Standard Book Number (ISBN)

SBN 0-486-27557-4

Romeo and Juliet is a tragedy written by William Shakespeare early in his career about two young star-crossed lovers whose deaths ultimately reconcile their feuding families. It was among Shakespeare's most popular plays during his lifetime and along with *Hamlet*, is one of his most

URN: urn:isbn:0-486-27557-4
However, it does not tell where to find it.

URI VS. URL

```
        userinfo       host      port
        ┌──────┐ ┌──────────┐ ┌──┐
https://john.doe@www.example.com:123/forum/questions/?tag=networking&order=newest#top
└──┬───┘└────────────────────────┬──┘└───────┬───────┘└───────────────────────────┘└┬┘
scheme          authority                  path                  query           fragment

ldap://[2001:db8::7]/c=GB?objectClass?one
└┬┘   └───────┬───────┘└┬┘ └──────┬──────┘
scheme   authority   path      query

mailto:John.Doe@example.com
└──┬──┘└────────┬────────┘
scheme      path

news:comp.infosystems.www.servers.unix
└──┬──┘└──────────────┬──────────────┘
scheme             path

tel:+1-816-555-1212
└┬┘└──────┬──────┘
scheme    path

telnet://192.0.2.16:80/
└──┬──┘   └────┬────┘|
scheme     authority  path

urn:oasis:names:specification:docbook:dtd:xml:4.1.2
└┬┘└──────────────────────────┬──────────────────────┘
scheme                      path
```

and git://

A Uniform Resource Locator (URL) is a subset of the Uniform Resource Identifier (URI) that specifies where an identified resource is available and the mechanism for retrieving it.

In non-technical context and in software for the WWW, the word “URL” (a.k.a. web address) remains widely used.

They are often used interchangeably.

URI SCHEME IN FINE GRANULARITY

```
 foo://username:password@example.com:8042/over/there/index.dtb?type=animal&name=narwhal#nose
 \_/   \_______________/ \_________/ \__/            \___/ \_/ \_______________________/ \__/
  |           |               |       |               |    |            |                  |
  |       userinfo          host    port              |    |          query          fragment
  |    \_______________________________/\_____________|____|/ \__/        \__/
scheme                |                          |    |    |    |          |
 name              authority                     |    |    |    |          |
  |                                           path    |    |    interpretable as keys
  |                                                   |    |
  |    \_____________________________________________|____|/       \____/     \_____/
  |                         |                         |    |         |           |
scheme              hierarchical part                 |    |    interpretable as values
 name                                                 |    |
  |            path               interpretable as filename |
  |   ___________|____________                             |
 / \ /                        \                            |
 urn:example:animal:ferret:nose               interpretable as extension

               path
          _________|________
scheme /                  \
 name  userinfo  hostname       query
 _|__   ___|__   ____|____   _____|_____
/    \ /      \ /         \ /           \
mailto:username@example.com?subject=Topic
```

COMMON RESPONSE CODE

- 200 OK -- The request has succeeded. The information returned with the response is dependent on the method used in the request
- 301 Moved Permanently -- The requested resource has been assigned a new permanent URI and any future references to this resource SHOULD use one of the returned URIs
- 404 Not Found -- The server has not found anything matching the Request-URI. No indication is given of whether the condition is temporary or permanent. The 410 (Gone) status code SHOULD be used if the server knows...that an old resource is permanently unavailable and has no forwarding address
- 405 Method Not Allowed -- The method specified in the Request-Line is not allowed for the resource identified by the Request-URI.
- For a complete list: https://www.rfc-editor.org/rfc/rfc9110.html#name-status-codes

TYPES OF HTTP RESPONSES

1. Informational responses (100–199)
2. Successful responses (200–299)
3. Redirects (300–399)
4. Client errors (400–499)
5. Server errors (500–599)

```
< HTTP/1.1 200 OK
```

For more examples please refer to :
https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

and

https://www.tutorialspoint.com/http/http_responses.htm

STRUCTURES OF HTTP RESPONSES

- A Status-line:
  - Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
- Zero or more header (General | Response | Entity) fields followed by CRLF
- An empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields
- Optionally a message-body

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 88
Content-Type: text/html
Connection: Closed
```

```
<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

HTTP HEADERS

- Transmitted after the request line (in case of a request HTTP message) or the response line (in case of a response HTTP message)
- It is the first part of the message
- Colon separated key-value pairs in clear text string format
- Terminated by a cartridge return (CR) and line feed (LF) character sequence
- Example (request): (q specifies the weights)
  `Accept-Language: de; q=1.0, en; q=0.5`

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 88
Content-Type: text/html
Connection: Closed
```

```
<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

HTTP HEADER FIELDS

- There are at least 39 common standard request fields and 15 common non-standard request fields
- There are at least 41 common standard response fields and 10 common non-standard response fields
- Back to the example we had

```
< Date: Mon, 29 Aug 2022 17:37:00 GMT
< Server: Apache/2.4.6 (Red Hat Enterprise
Linux)
< Vary: Host, Accept-Encoding
< Accept-Ranges: bytes
< Connection: close
< Transfer-Encoding: chunked
< Content-Type: text/html; charset=UTF-8
```

For a complete list, see
https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

HTTP METHODS

```
> GET / HTTP/1.1
```

- There are at least 9 HTTP methods
- GET, HEAD (covered in detail later)
- POST: Frequently used for passing credentials
- TRACE: What methods are defined on this URI?
- DELETE: Rarely supported for most URIs
- PUT: Rarely supported. Equivalent to the echo command in Unix
  $ echo "hello world" > temp.txt

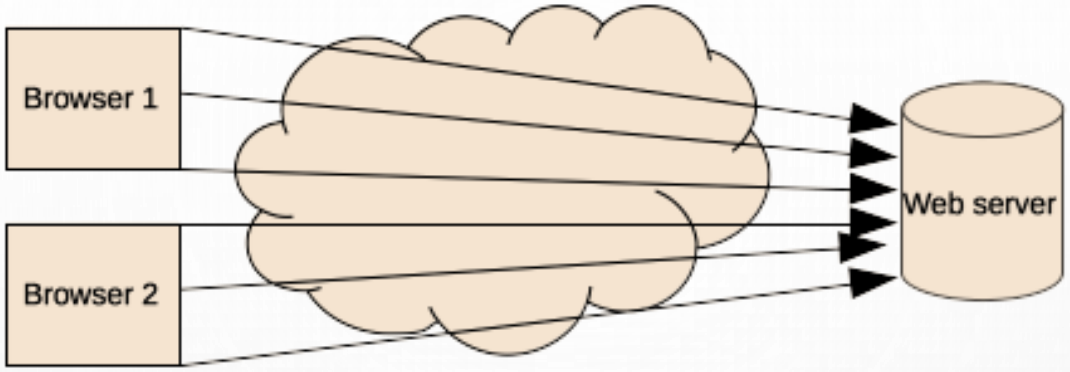
1. CONNECT
2. DELETE
3. GET
4. HEAD
5. OPTIONS
6. PATCH
7. POST
8. PUT
9. TRACE

GET

- GET is used to request data from a specified resource.
- GET is one of the most common HTTP methods. In the example below, the user is requesting data satisfying two conditions: name1=value1 and name2=value2.
  `/test/demo_form.php?name1=value1&name2=value2`
  - GET requests can be cached (different types of caching: https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching, https://developers.google.com/web/fundamentals/performance/get-started/httpcaching-6)
  - 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 (2048 characters, see this answer on Stackoverflow)
  - GET requests is only used to request data (not modify)

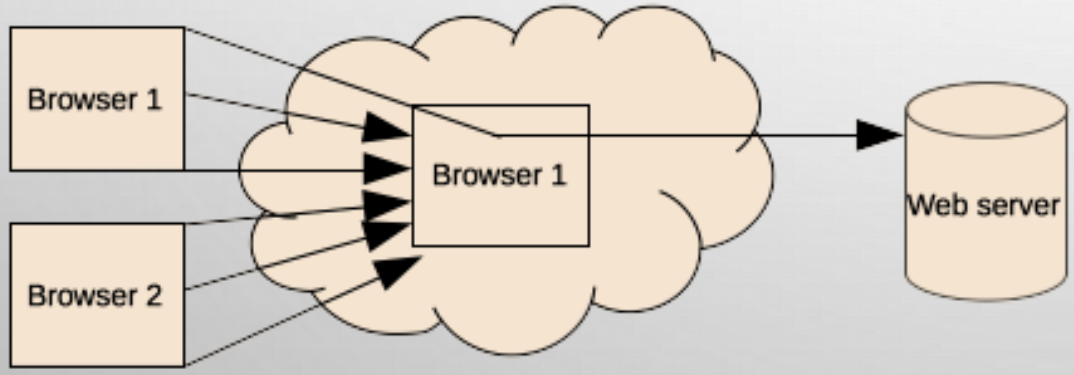
http caching

No cache



All identical requests are
going through to the server.

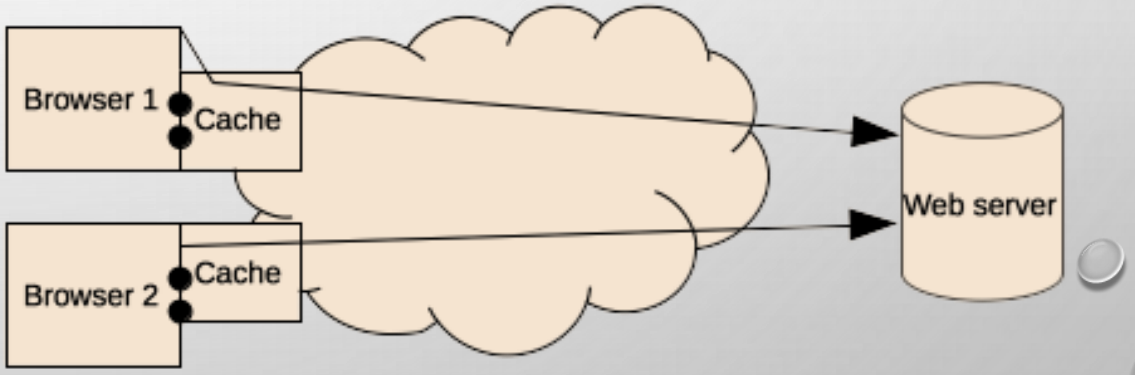
Shared cache



The first request is going through.

Subsequent identical requests are served
by the shared cache.
(more efficient)

Local (private) cache



The first request of each client is going through.

Subsequent identical requests are not even sent, but
served by the local cach.
(most efficient, except for first requests)

HEAD

The HEAD method asks for a response identical to that of a GET request, but without the response body.

You may try different methods at: https://reqbin.com/
ReqBin is a free, online HTTP/REST/SOAP API client.

POST

POST is used to send data to a server to create/update a resource.

-d, --data <data> HTTP POST data
-F, --form <name=content> Specify multipart MIME data
-H, --header <header/@file> Pass custom header(s) to server
-X, --request <command> Specify request command to use
For more options of curl, check the curl man page.

- 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

Examples:

To Post with data

```
$ curl -d "username=mkyong&password=abc" http://localhost:8080/api/login/
```

To post with a file

```
$ curl -F file=@"path/to/data.txt" http://localhost:8080/api/upload/
```

To post with JSON data

```
$ curl -H "Content-Type: application/json" -X POST -d
'{"username":"mkyong","password":"abc"}' http://localhost:8080/api/login/Copy
```

Default HTTP methods: If you just pass in a HTTP URL like curl http://example.com it will use GET. If you use -d or -F curl will use POST, -I will cause a HEAD and -T will make it a PUT.

GET VS POST

	GET	POST
How data is applied	In the URL	In the message body
Data type	Only ASCII characters	No restrictions, even binary
Security	Less secure	Safer as parameters are not stored in browser history of web server logs
Restrictions on data length	Limited to URL length: usually <2048 characters	No restrictions
Usability	Should not be used when sending sensitive information (e.g., password)	POST should be used
Visability	Displayed in the address bar	Not displayed on in the address bar

HOW TO SEND HTTP REQUESTS ON A COMMAND LINE

On Linux, you may use wget or curl commands to send HTTP requests.

Examples:

$ wget -S -O - http://www.google.com

$ curl -i http://www.google.com

How to use wget:

https://www.digitalocean.com/community/tutorials/how-to-use-wget-to-download-files-and-interact-with-rest-apis

How to use curl:

https://curl.se/docs/httpscripting.html

A MORE COMPLICATED EXAMPLE

curl -d "param1=value1&param2=value2" -H "Content-Type: application/x-www-form-urlencoded" -X POST http://localhost:3000/data

-d "param1=value1&param2=value2": Send specified data in POST request.

-H "Content-Type: application/x-www-form-urlencoded": Content type header

- -H "Content-Type: application/json"

-X POST: The request method to use.

- -X POST
- -X PUT

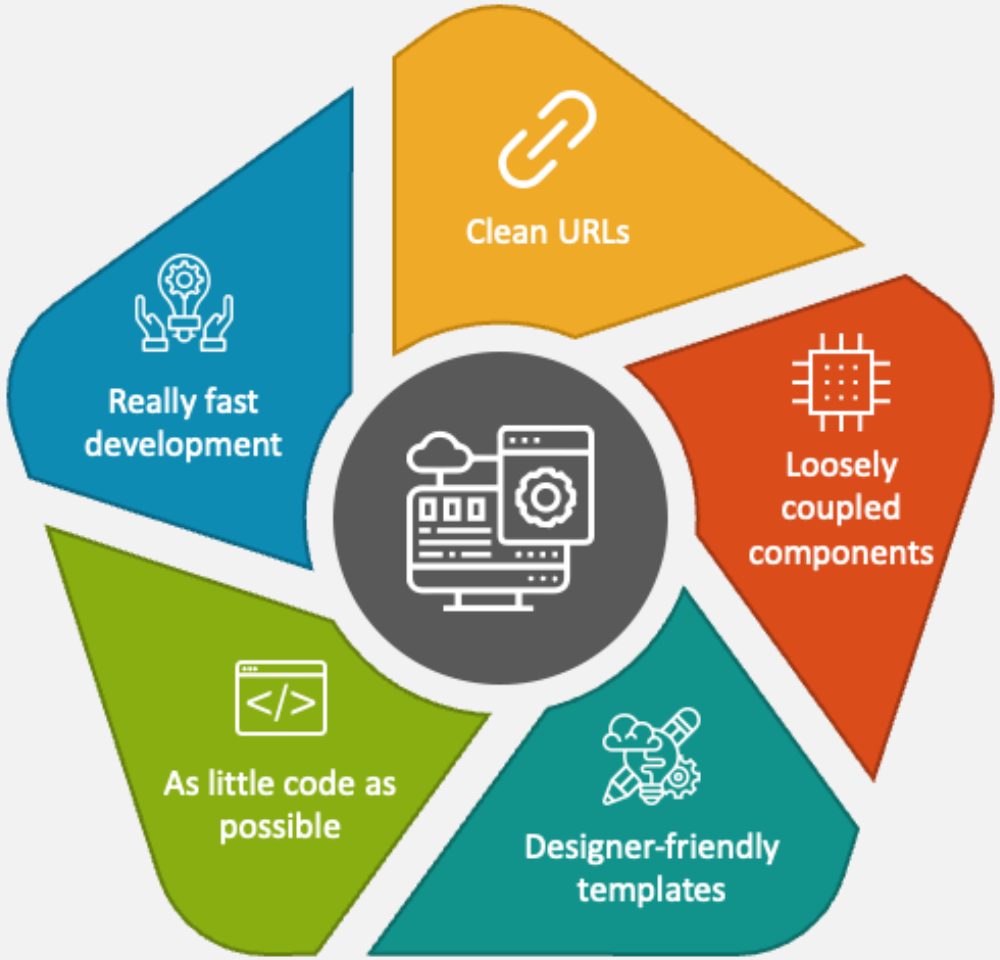
SUMMARY OF WEB ARCHITECTURE

- Principles: Internet Standard documented by a Request for Comments (RFC) IETF website: https://ietf.org/standards/rfcs/
- Identifiers: URI, URL, etc.
- Protocols: HTTP, etc.
- Meta formats: XML, etc.
- Internationalization: IRI (Internationalization of identifiers)

WEB DEVELOPMENT FRAMEWORKS

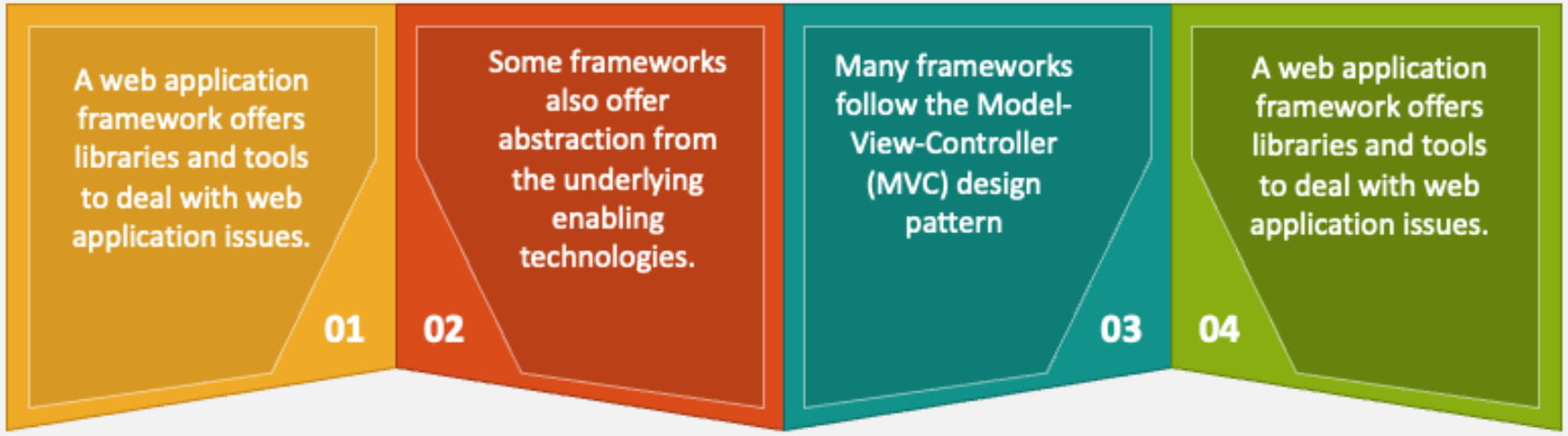
CS418/518

WHY DO WE NEED A WEB APP FRAMEWORK?



https://www.collidu.com/presentation-web-application-framework

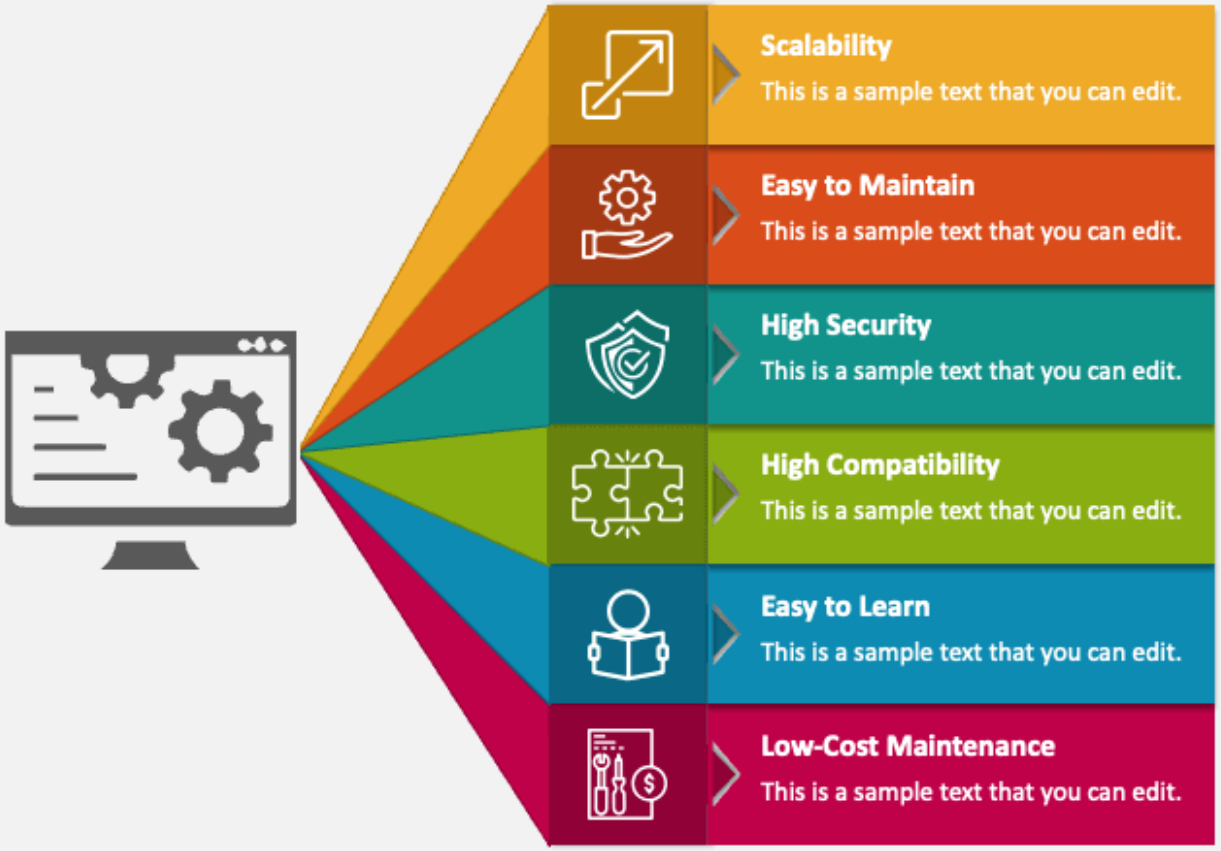
BENEFITS OF USING A WEB APP FRAMEWORK



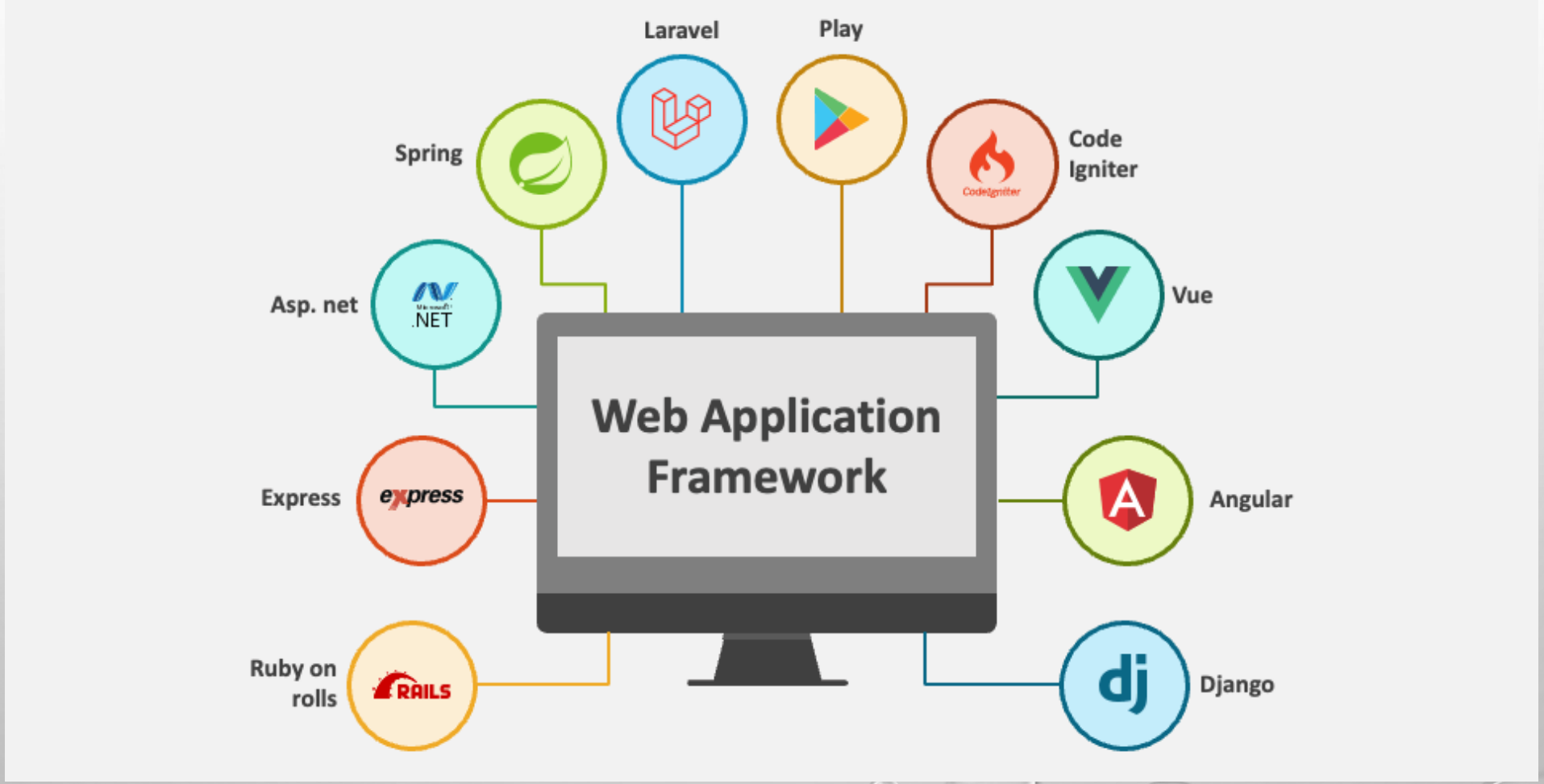
Source: Vrije University Brussel

https://www.collidu.com/presentation-web-application-framework

GENERAL PROPERTIES OF WEB APP FRAMEWORKS



EXAMPLES OF WEB APP FRAMEWORKS – NOT AN EXHAUSTIVE LIST

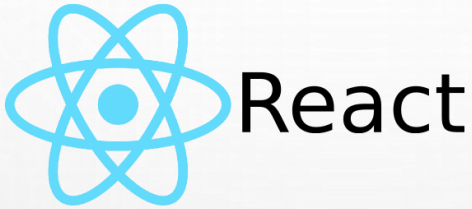


FRONTEND VS. BACKEND FRAMEWORKS

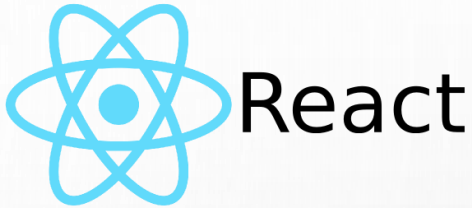
	FRONT-END FRAMEWORKS	BACK-END FRAMEWORKS
Components	Focus on the client-side components of a web application, including the user interface and user experience.	Focus on the server-side components of a web application, including handling requests, managing databases, and processing data.
Tools	Provide pre-written tools and libraries for building interactive and responsive user interfaces like buttons, forms, and data visualization components.	Provide pre-written tools and libraries for building scalable and secure server-side applications, such as routing, authentication, and database integration.
Languages	Typically use a variety of programming languages, such as HTML, CSS, and JavaScript.	Typically use a variety of programming languages, such as Ruby, Python, and Java.
Examples	Examples of front-end frameworks include React, AngularJS, and Vue.js.	Examples of back-end frameworks include Ruby on Rails, Django, and Express.

FRONTEND FRAMEWORKS

- ReAct
- AngularJS
- Vue.js
- jQuery
- etc.



REACT.JS



- React is not a framework, it's a frontend library
- Pros
  - Bringing HTML into your JavaScript helps developers write code quickly.
  - Break down complex UI into smaller components
  - Virtual DOM increases the page's performance
  - Allows developers to inspect and debug the DOM quickly
- Cons
  - Community space is smaller than others
  - Continuous upgradation in technology has become troublesome for developers in terms of making proper documentation

VUE.JS

- Vue.js is one of the progressive JavaScript frameworks allowing developers to build desktop and mobile apps and web interfaces.
- Pros
  - Small footprints
  - Simple and easy to grasp.
  - extensive documentation, which makes it easier to learn
  - easily handle 2-way data binding, thus making the code reactive
- Cons
  - Being flexible, but while working on large-scale projects with multiple developers, flexibility can lead to complexity
  - Smaller community but growing

JQUERY

- jQuery is a fast, small, and feature-rich JavaScript library-based web development framework that makes adding dynamic and interactive elements to web pages more accessible.
- Pros
  - Simple syntax and a wide range of built-in methods
  - Intuitive and straightforward APIs, making it easier for developers to get started and make progress quickly
  - A wide range of plugins and extensions that can add additional functionalities
- Cons
  - If JavaScript is disabled, the dynamic behavior provided by jQuery will not be available, and the user experience may be degraded
  - It can be complex in some cases, particularly for developers new to the library.

BACKEND FRAMEWORKS

- Express
- Next.js
- NuxtJS
- Nest.js
- Django
- Meteor
- Ruby on Rails
- Laravel
- Spring
- Flask

Note: some frameworks can be used both serverside and clientside, such as Laravel.

NEXT.JS

- A React-based web development framework that simplifies building server-side rendered React applications.
- Pros
  - It has built-in CSS support
  - Developers can automatically optimize images
  - Developers can update existing pages by re-rendering them in the background
  - Automatic TypeScript compilation makes AngularJS development easier.
- Cons
  - Since the framework is not front-end friendly, you must develop the entire UI layer from scratch.
  - It does not have a built-in state manager. You must use Redux or a similar library if you need one.
  - The usage is limited to its file-based router. To use dynamic routes, you must depend on a Node.js server

DJANGO

- Django is a Python-based Model-View-Template framework for web development. This platform is used by well-known companies such as Google, YouTube, and Instagram.
- Pros
  - Offers the best documentation since its inception, and its documentation has only improved drastically with current tech innovation. It is also available in multiple languages
  - There is no need for prior experience in the backend to build a fully functional website
  - It provides flexibility through pluggable apps, through which third-party applications can be quickly plugged
  - Django provides high security by default
- Cons
  - Not supported for highly scalable applications
  - Django has a steep learning curve

SPRING

- Spring is a popular open-source web development framework for building enterprise-level Java applications.
- Pros
  - It uses an IoC container responsible for managing the lifecycle of the application components. It makes it easy to build loosely coupled applications and reduces the amount of boilerplate code required.
  - It includes a Model-View-Controller (MVC) framework, which makes it easy to build web applications that are well-structured and maintainable.
  - It provides a suite of tools for testing your application
  - It includes a comprehensive security framework
- Cons
  - Large and complex, and it can be challenging for developers to understand all its features and components
  - Relies heavily on third-party libraries and components
  - It has a large memory footprint

Inversion of control is a software design principle that asserts a program can benefit in terms of pluggability, testability, usability and loose coupling if the management of an application's flow is transferred to a different part of the application.

FACTORS TO CONSIDER WHEN CHOOSING WEB APP FRAMEWORKS?

- Ease of Installation
- Documentation and Support
- Licensing
- Scalability and Flexibility
- Cost and Budget
- Learning Curve
- Customization and Extensibility

WHAT FRAMEWORK TO USE FOR THIS COURSE?

- Principle: at your discretion
- What we will cover
  - Express
  - NodeJS (JavaScript)
  - ReAct (JavaScript)
  - jQuery (JavaScript)

INTRODUCTION TO GIT

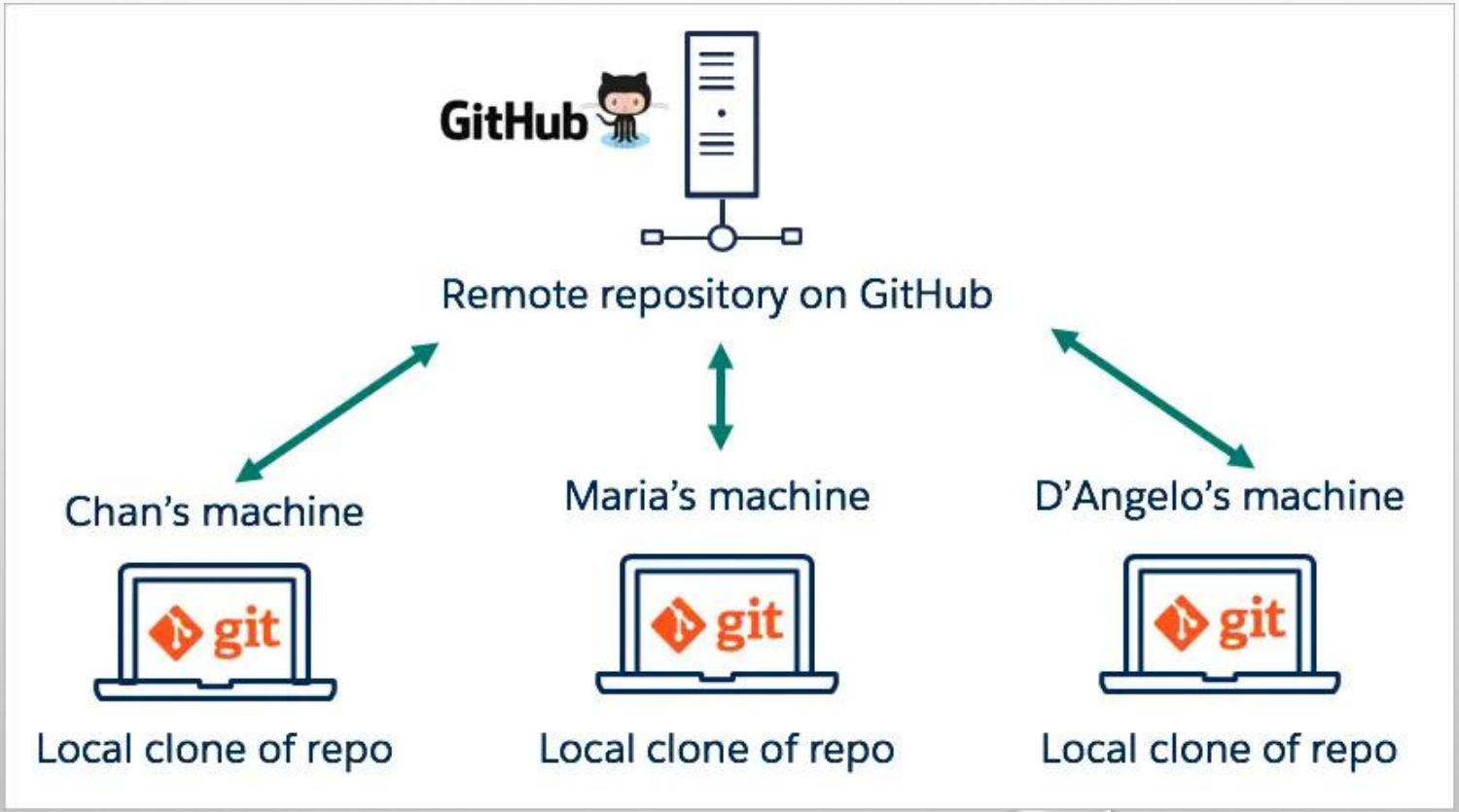
CS418/518

Nasreen Arif

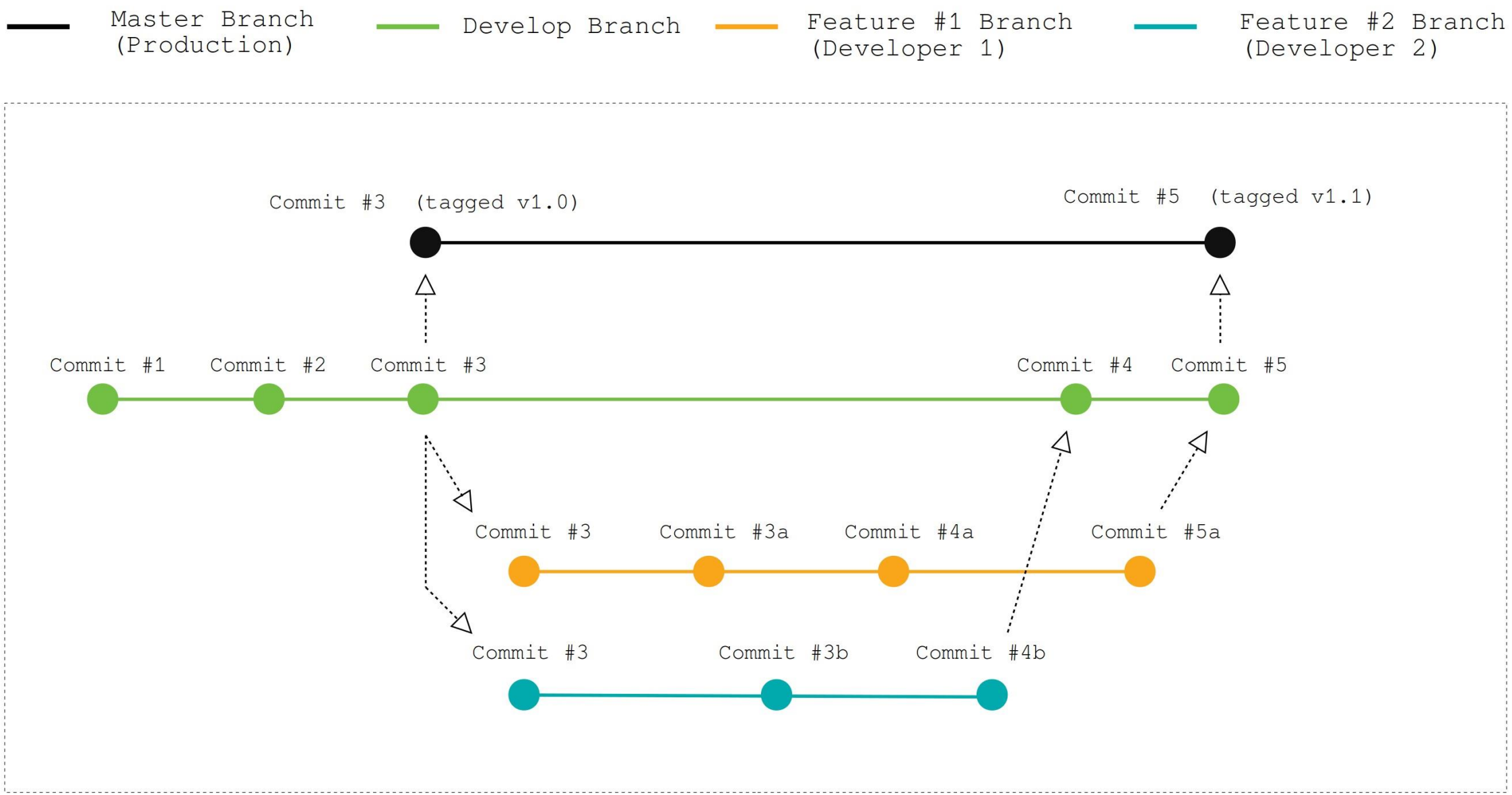
By Courtesy: presentation slides from Dr. Jian Wu

OUR GOAL

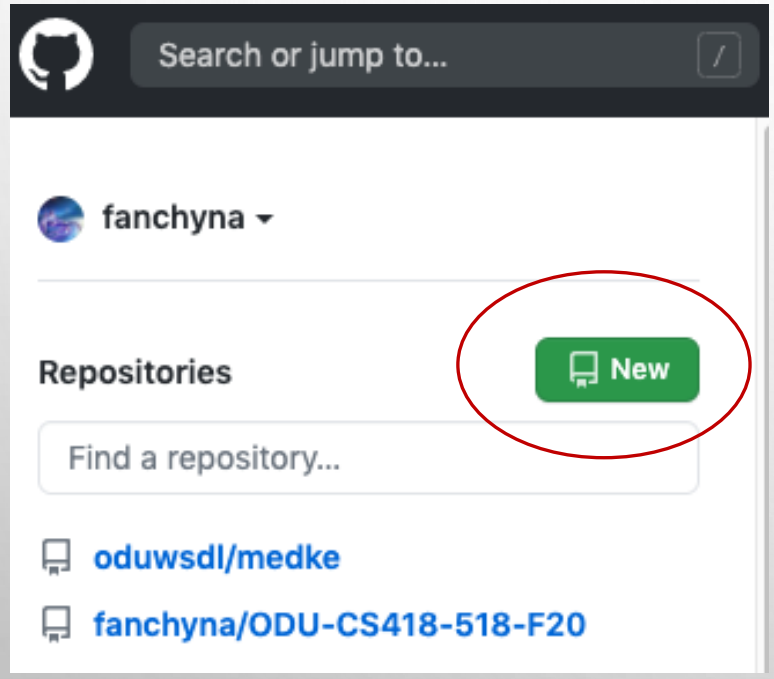
- Create a code repo on github and sync it with the repo on your local machine

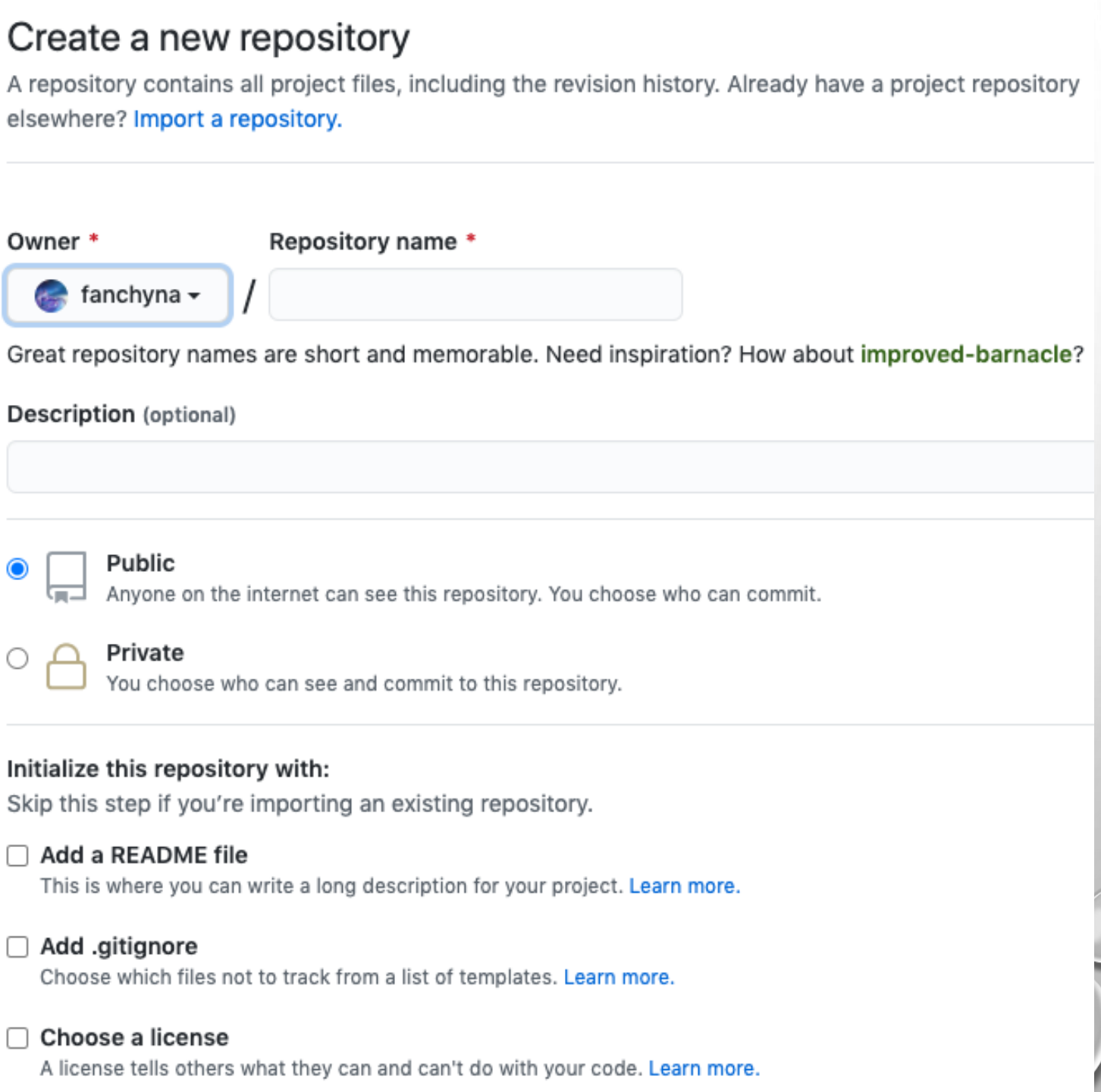


A Simple, Effective Git Workflow



First create a new repository on GitHub





3 ways to create repositories

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Settings

Quick setup — if you've done this kind of thing before

or HTTPS SSH https://github.com/jbrunelle/ODUCS418F17.git

We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# ODUCS418F17" >> README.md
  git init
  git add README.md
  git commit -m "first commit"
  git remote add origin https://github.com/jbrunelle/ODUCS418F17.git
  git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/jbrunelle/ODUCS418F17.git
  git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

BRANCHES

- Purpose: work on a new feature with impacting the stable code
- Command:
  - `$ git checkout –b [branchid]`
- Merge:
  - `$ git checkout master` Switched to branch ‘master’
  - `$ git merge [branchid]`
  - `$ git status`

GIT BRANCHING GUIDES AND TUTORIALS

- https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell
- http://rogerdudler.github.io/git-guide/