Geography: 495B
Web & Mobile GIS

Web & Mobile GIS
Module Two

Web Fundamentals

Bo Zhao

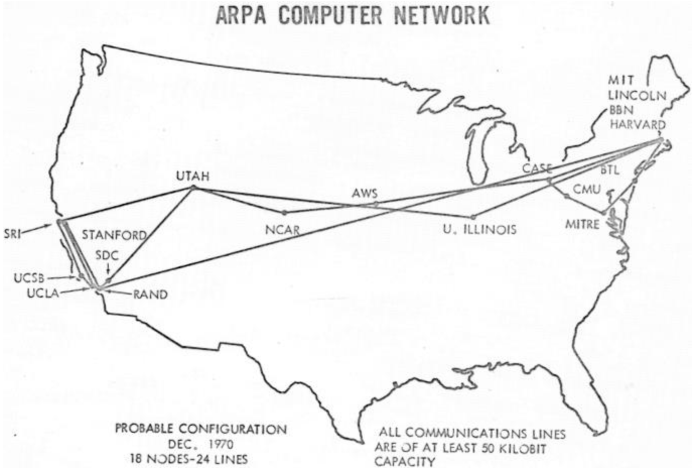
The Department of Geography

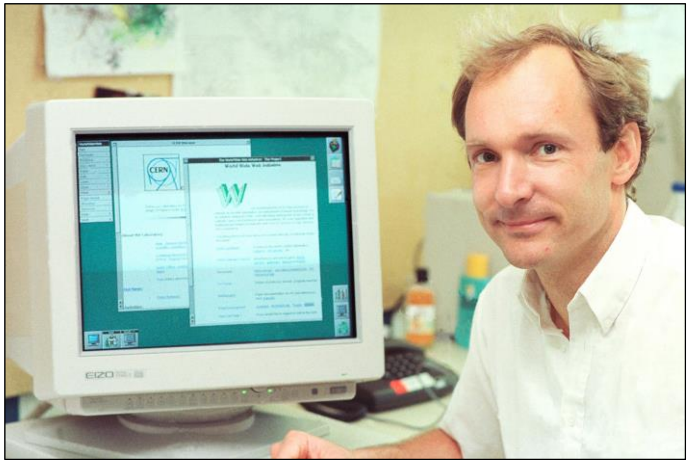
Outline

- Web
- DNS – Domain Name System
- URL - Uniform Resource Locators
- Web Server
- Documents
  - HTML, CSS and JavaScript
- Internet

Brief History of the Web

- In the late 1960s, the US military developed a communication network called ARPANET. This can be considered a forerunner of the Web, as it worked on packet switching, and featured the first implementation of the TCP/IP protocol suite. These two technologies form the basis of the infrastructure that the internet is built on.
- By late 1990, Tim Berners-Lee (often referred to as TimBL) had created all the things needed to run the first version of the web — HTTP, HTML, the first web browser, which was called World Wide Web, an HTTP server, and some web pages to look at.
- In the next few years that followed, the web exploded, with multiple browsers being released, thousands of web servers being set up, and millions of web pages being created. OK, that's a very simple summary of what happened, but we did promise you a brief summary.
- One last significant data point to share is that in 1994, TimBL founded the World Wide Web Consortium (W3C), an organization that brings together representatives from many different technology companies to work together on the creation of web technology specifications. After that other technologies followed such as CSS and JavaScript, and the web started to look more like the web we know today.





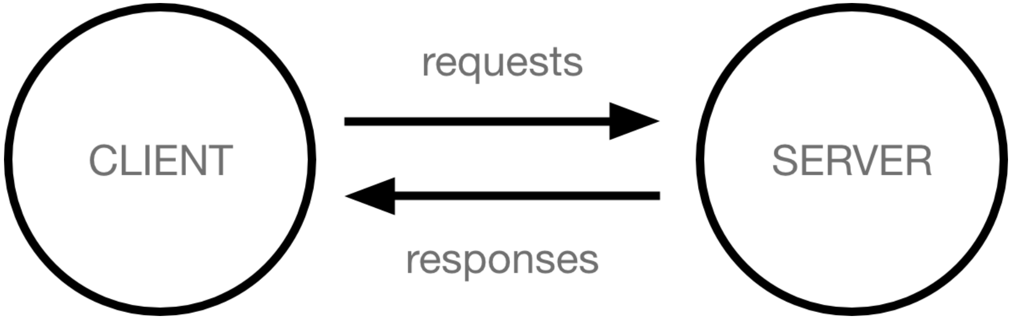
Tim Berners-Lee

Modern Web Technologies

- Browsers
  - Web browsers are the software programs people use to consume the web, and include Firefox, Chrome, Opera, Safari, and Edge.
- HTTP
  - Hypertext Transfer Protocol (HTTP) is a messaging protocol that allows web browsers to communicate with web servers (where web sites are stored).
- HTML, CSS, and JavaScript
  - are the main three technologies you'll use to build a website
- Tooling
  - Developer tools, testing tools, libraries and frameworks
- Server-side language and frameworks
  - ASP.NET, Python, PHP and NodeJS.

How the web works

Computers connected to the web are called **clients** and **servers**. A simplified diagram of how they interact might look like this:



- Clients are the typical web user's internet-connected devices (for example, your computer connected to your Wi-Fi, or your phone connected to your mobile network) and web-accessing software available on those devices (usually a web browser like Firefox or Chrome).
- Servers are computers that store webpages, sites, or apps. When a client device wants to access a webpage, a copy of the webpage is downloaded from the server onto the client machine to be displayed in the user's web browser.

How the web works (Cont’d)

- **Your internet connection**: Allows you to send and receive data on the web. It's basically like the street between your house and the shop.
- **TCP/IP**: Transmission Control Protocol and Internet Protocol are communication protocols that define how data should travel across the internet. This is like the transport mechanisms that let you place an order, go to the shop, and buy your goods. In our example, this is like a car or a bike (or however else you might get around).
- **DNS**: Domain Name Systems are like an address book for websites. When you type a web address in your browser, the browser looks at the DNS to find the website's real address before it can retrieve the website. The browser needs to find out which server the website lives on, so it can send HTTP messages to the right place (see below). This is like looking up the address of the shop so you can access it.
- **HTTP**: Hypertext Transfer Protocol is an application protocol that defines a language for clients and servers to speak to each other. This is like the language you use to order your goods.
- **Component files**: A website is made up of many different files, which are like the different parts of the goods you buy from the shop. These files come in two main types:
  - **Code files**: Websites are built primarily from HTML, CSS, and JavaScript, though you'll meet other technologies a bit later.
  - **Assets**: This is a collective name for all the other stuff that makes up a website, such as images, music, video, Word documents, and PDFs.

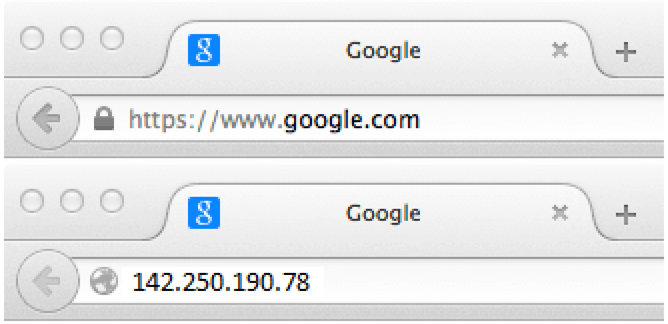
The whole process

When you type a web address into your browser (for our analogy that's like walking to the shop):

1. The browser goes to the DNS server and finds the real address of the server that the website lives on (you find the address of the shop).
2. The browser sends an HTTP request message to the server, asking it to send a copy of the website to the client (you go to the shop and order your goods). This message, and all other data sent between the client and the server, is sent across your internet connection using TCP/IP.
3. If the server approves the client's request, the server sends the client a "200 OK" message, which means "Of course you can look at that website! Here it is", and then starts sending the website's files to the browser as a series of small chunks called data packets (the shop gives you your goods, and you bring them back to your house).
4. The browser assembles the small chunks into a complete web page and displays it to you (the goods arrive at your door — new shiny stuff, awesome!).

Finding computers

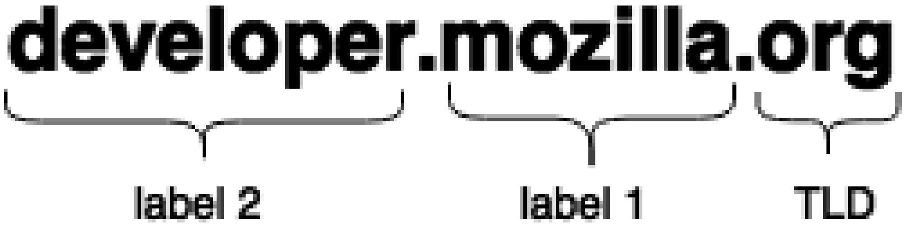
If you want to send a message to a computer, you have to specify which one. Thus, any computer linked to a network has a unique address that identifies it, called an "IP address" (where IP stands for Internet Protocol). It's an address made of a series of four numbers separated by dots, for example: 192.168.2.10.



That's perfectly fine for computers, but we human beings have a hard time remembering that sort of address. To make things easier, we can alias an IP address with a human readable name called a domain name. For example (at the time of writing; IP addresses can change) google.com is the domain name used on top of the IP address 142.250.190.78. So using the domain name is the easiest way for us to reach a computer over the Internet.

Domain Name

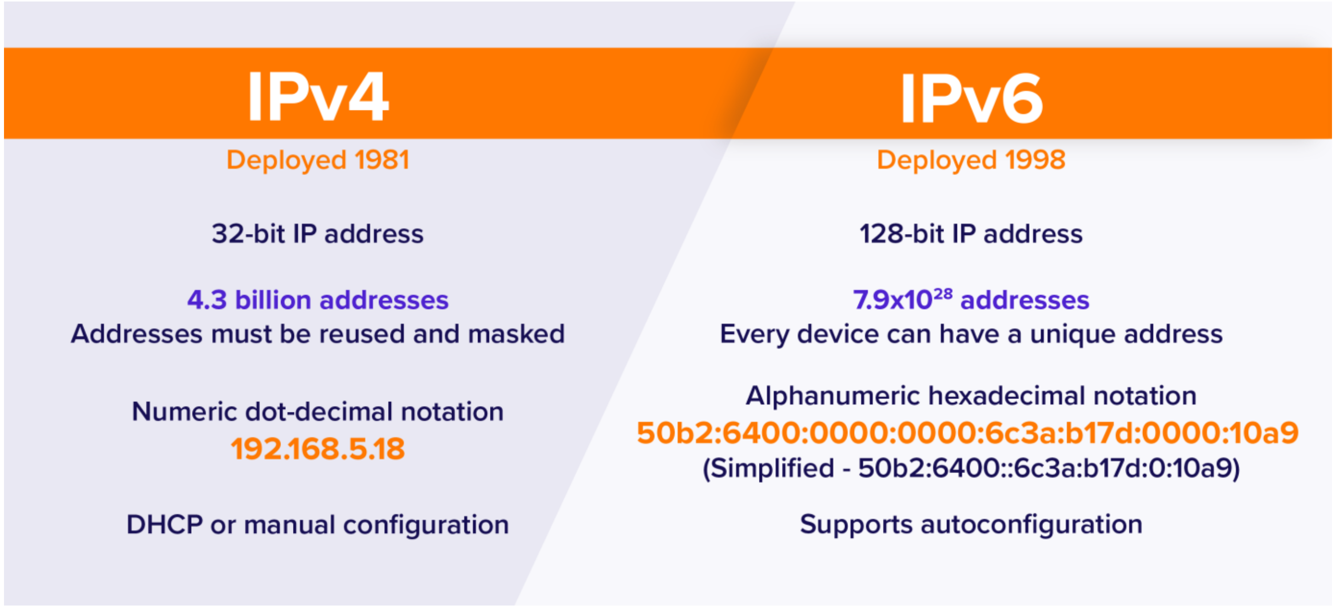
- Domain names are a key part of the Internet infrastructure. They provide a human-readable address for any web server available on the Internet.
- Any Internet-connected computer can be reached through a public IP address, either an IPv4 address (e.g., 173.194.121.32) or an IPv6 address (e.g., 2027:0da8:8b73:0000:0000:8a2e:0370:1337).
- Computers can handle such addresses easily, but people have a hard time finding out who's running the server or what service the website offers. IP addresses are hard to remember and might change over time.



Top-level Domain

DNS - Domain Name Systems

- Real web addresses aren't the nice, memorable strings you type into your address bar to find your favorite websites. They are special numbers that look like this: 63.245.215.20.
- This is called an IP address, and it represents a unique location on the web. However, it's not very easy to remember, is it? That's why **Domain Name Servers** were invented. These are special servers that match up a web address you type into your browser (like "mozilla.org") to the website's real (IP) address.
- Websites can be reached directly via their IP addresses. You can find the IP address of a website by typing its domain into a tool like IP Checker. Using “Ping” command in your terminal.
- Check your DNS IP address. https://www.bt.com/help/security/how-to-check-your-dns-settings

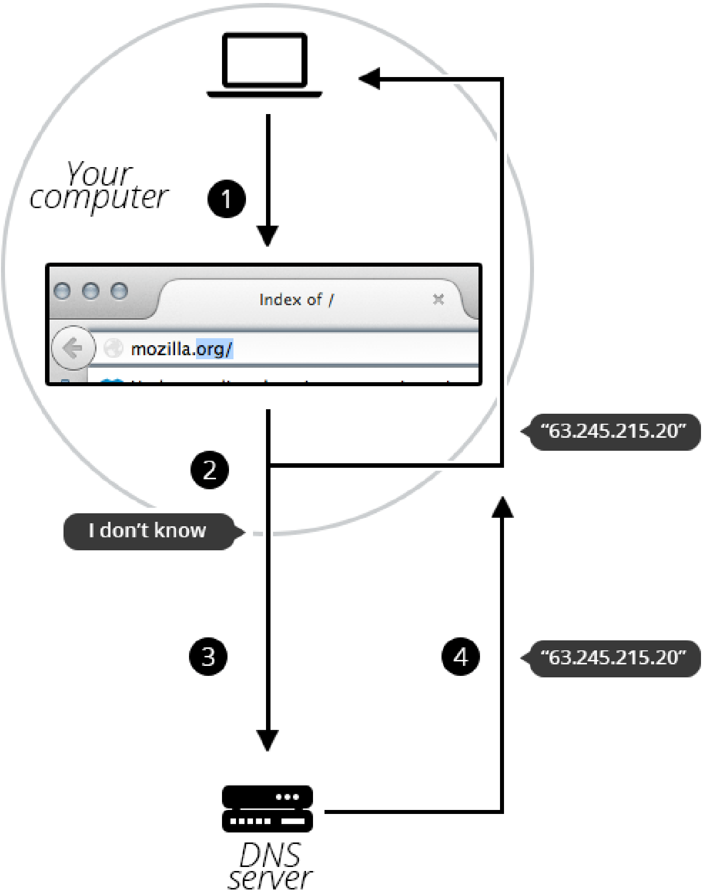


IPv4	IPv6
Deployed 1981	Deployed 1998
32-bit IP address	128-bit IP address
4.3 billion addresses Addresses must be reused and masked	7.9x10^{28} addresses Every device can have a unique address
Numeric dot-decimal notation 192.168.5.18	Alphanumeric hexadecimal notation 50b2:6400:0000:0000:6c3a:b17d:0000:10a9 (Simplified - 50b2:6400::6c3a:b17d:0:10a9)
DHCP or manual configuration	Supports autoconfiguration

How does a DNS request work?

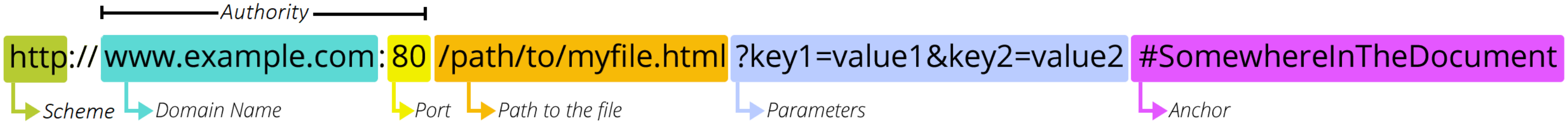
As we already saw, when you want to display a webpage in your browser it's easier to type a domai name than an IP address. Let's take a look at the process:

- Type mozilla.org in your browser's location bar.
- Your browser asks your computer if it already recognizes the IP address identified by this domain name (using a local DNS cache). If it does, the name is translated to the IP address and the browser negotiates contents with the web server. End of story.
- If your computer does not know which IP is behind the mozilla.org name, it goes on to ask a DNS server, whose job is precisely to tell your computer which IP address matches each registered domain name.
- Now that the computer knows the requested IP address your browser can negotiate contents with the web serve



URL - Uniform Resource Locators

URL stands for *Uniform Resource Locator*. A URL is nothing more than the address of a given unique resource on the Web. In theory, each valid URL points to a unique resource. Such resources can be an HTML page, a CSS document, an image, etc. In practice, there are some exceptions, the most common being a URL pointing to a resource that no longer exists or that has moved. As the resource represented by the URL and the URL itself are handled by the Web server, it is up to the owner of the web server to carefully manage that resource and its associated URL.


Authority
http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument
Scheme
Domain Name
Port
Path to the file
Parameters
Anchor

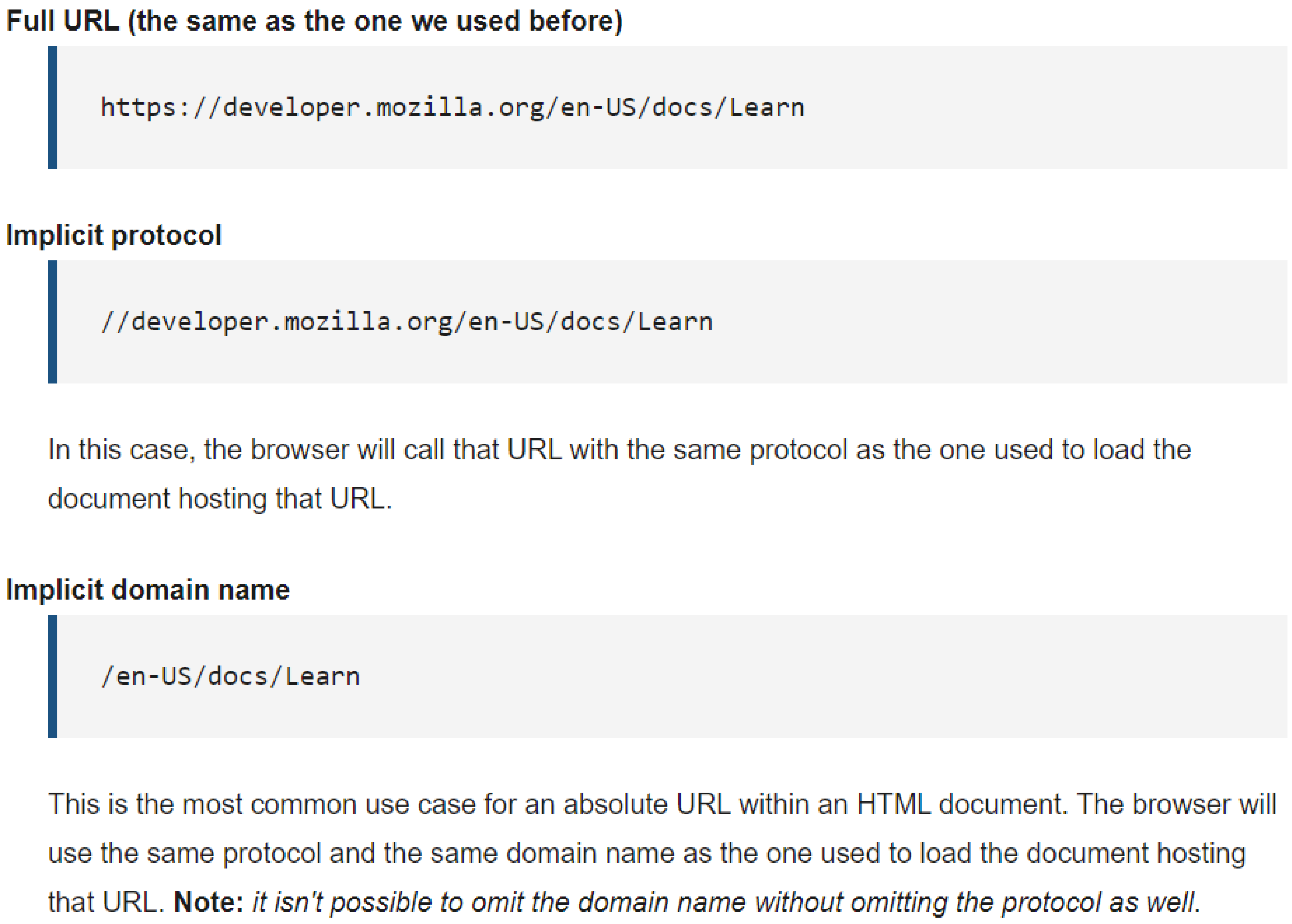
Absolute URLs vs relative URLs

- What we saw above is called an absolute URL, but there is also something called a relative URL. Let's examine what that distinction means in more detail.

- The required parts of a URL depend to a great extent on the context in which the URL is used. In your browser's address bar, a URL doesn't have any context, so you must provide a full (or absolute) URL, like the ones we saw above. You don't need to include the protocol (the browser uses HTTP by default) or the port (which is only required when the targeted Web server is using some unusual port), but all the other parts of the URL are necessary.

- When a URL is used within a document, such as in an HTML page, things are a bit different. Because the browser already has the document's own URL, it can use this information to fill in the missing parts of any URL available inside that document. We can differentiate between an absolute URL and a relative URL by looking only at the path part of the URL. If the path part of the URL starts with the "/" character, the browser will fetch that resource from the top root of the server, without reference to the context given by the current document.

Absolute URLs



Full URL (the same as the one we used before)

```
https://developer.mozilla.org/en-US/docs/Learn
```

Implicit protocol

```
//developer.mozilla.org/en-US/docs/Learn
```

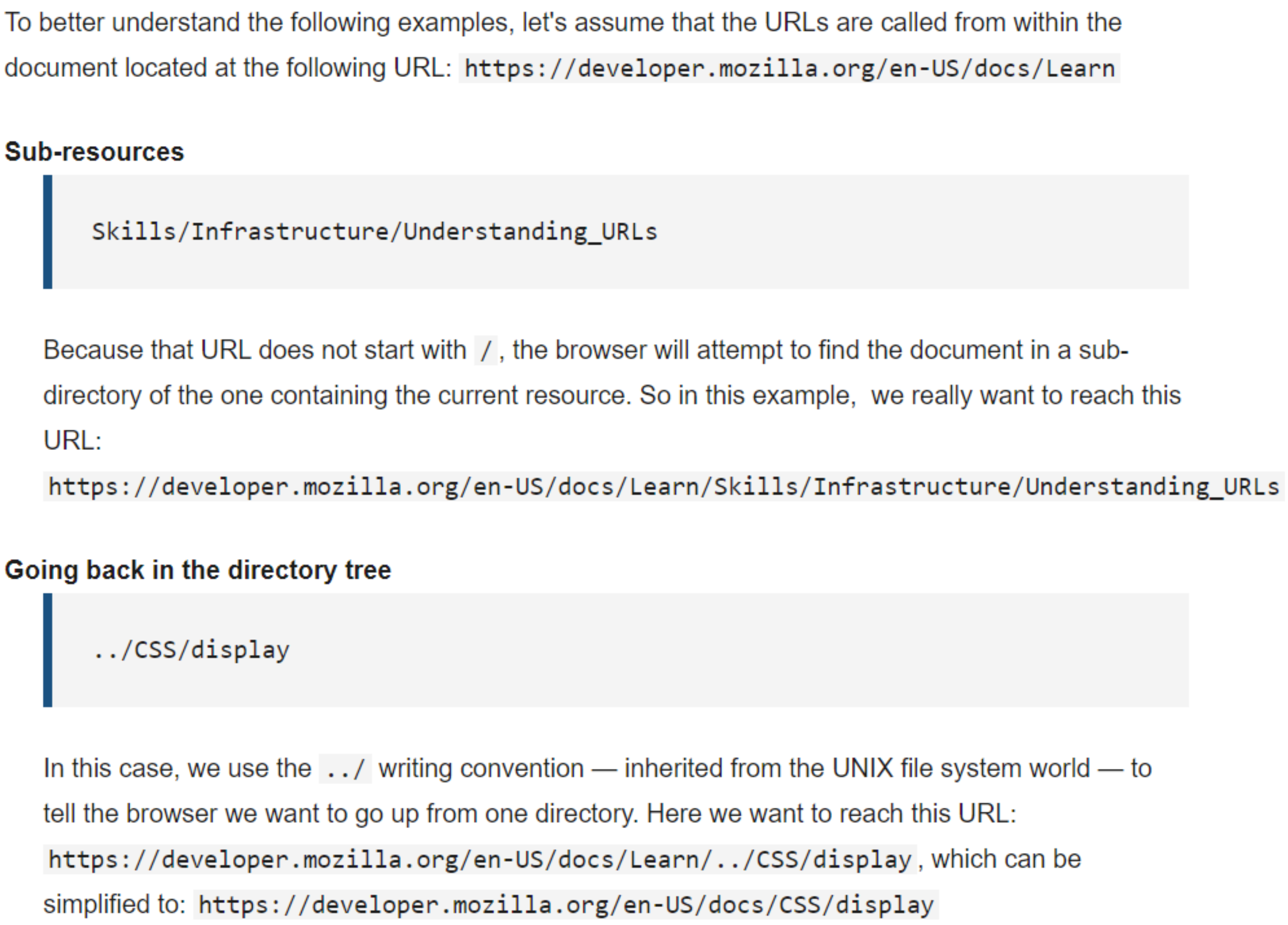
In this case, the browser will call that URL with the same protocol as the one used to load the document hosting that URL.

Implicit domain name

```
/en-US/docs/Learn
```

This is the most common use case for an absolute URL within an HTML document. The browser will use the same protocol and the same domain name as the one used to load the document hosting that URL. **Note:** *it isn't possible to omit the domain name without omitting the protocol as well.*

Relative URLs



To better understand the following examples, let's assume that the URLs are called from within the document located at the following URL: `https://developer.mozilla.org/en-US/docs/Learn`

Sub-resources

```
Skills/Infrastructure/Understanding_URLs
```

Because that URL does not start with `/`, the browser will attempt to find the document in a sub-directory of the one containing the current resource. So in this example, we really want to reach this URL: `https://developer.mozilla.org/en-US/docs/Learn/Skills/Infrastructure/Understanding_URLs`

Going back in the directory tree

```
../CSS/display
```

In this case, we use the `../` writing convention — inherited from the UNIX file system world — to tell the browser we want to go up from one directory. Here we want to reach this URL: `https://developer.mozilla.org/en-US/docs/Learn/../CSS/display`, which can be simplified to: `https://developer.mozilla.org/en-US/docs/CSS/display`

Web Server

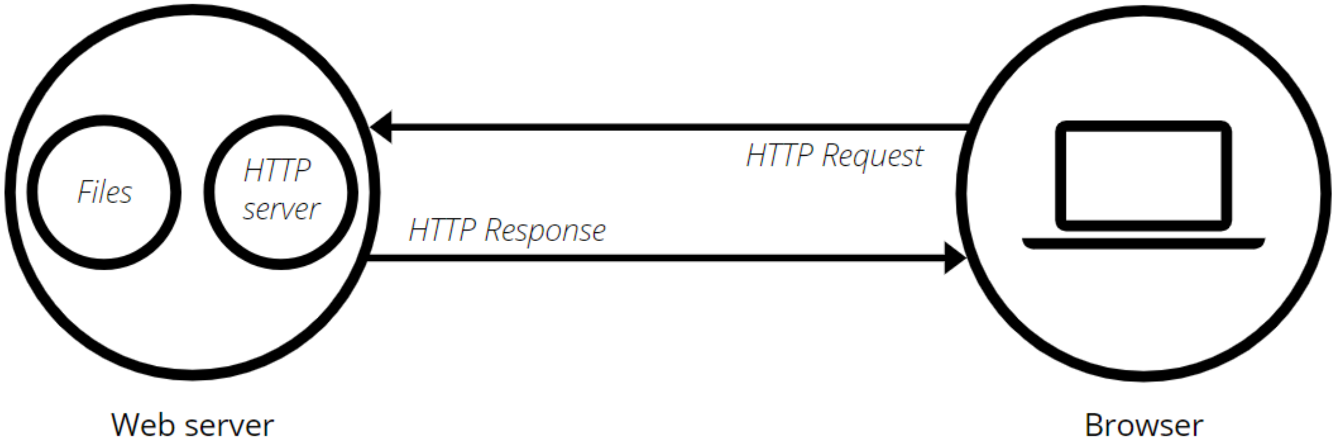
The term *web server* can refer to hardware or software, or both of them working together.

1. On the hardware side, a web server is a computer that stores web server software and a website's component files. (for example, HTML documents, images, CSS stylesheets, and JavaScript files) A web server connects to the Internet and supports physical data interchange with other devices connected to the web.

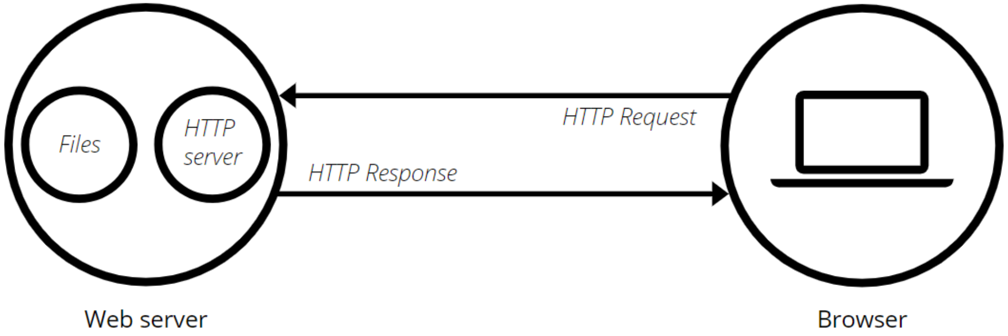
2. On the software side, a web server includes several parts that control how web users access hosted files. At a minimum, this is an HTTP server. An HTTP server is software that understands URLs (web addresses) and HTTP (the protocol your browser uses to view webpages). An HTTP server can be accessed through the domain names of the websites it stores, and it delivers the content of these hosted websites to the end user's device.

Web Server

At the most basic level, whenever a browser needs a file that is hosted on a web server, the browser requests the file via HTTP. When the request reaches the correct (hardware) web server, the (software) *HTTP server* accepts the request, finds the requested document, and sends it back to the browser, also through HTTP. (If the server doesn't find the requested document, it returns a 404 response instead.)



Web GIS Server



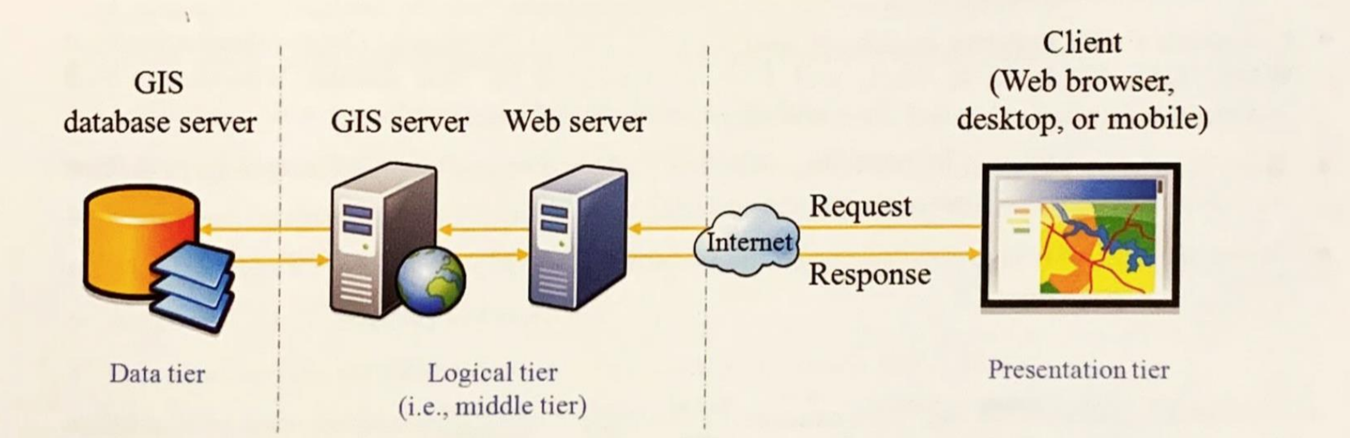


Figure 2.3 In the logical architecture and workflow of a basic Web GIS, the components can all be deployed on one computer, but logically, they are still three separate tiers.

Static vs. dynamic web server

- A static web server, or stack, consists of a computer (hardware) with an HTTP server (software). We call it "static" because the server sends its hosted files as-is to your browser.

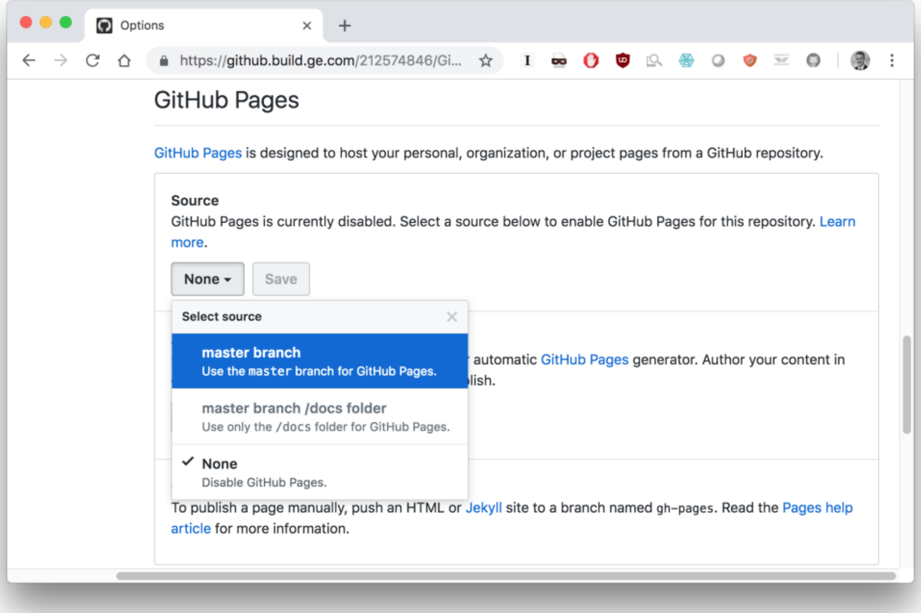
- A dynamic web server consists of a static web server plus extra software, most commonly an application server and a database. We call it "dynamic" because the application server updates the hosted files before sending content to your browser via the HTTP server.

Static vs. dynamic Content

- A server can serve either static or dynamic content. Remember that the term *static* means "served as-is". Static websites are the easiest to set up, so we suggest you make your first site a static site.
- The term *dynamic* means that the server processes the content or even generates it on the fly from a database. This approach provides more flexibility, but the technical stack is more complex, making it dramatically more challenging to build a website.

Web Server Hosting

- Getting hosting and a domain name.
- Using an online tool like GitHub or Google App Engine
- Using a web-based IDE such as CodePen



GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

Source

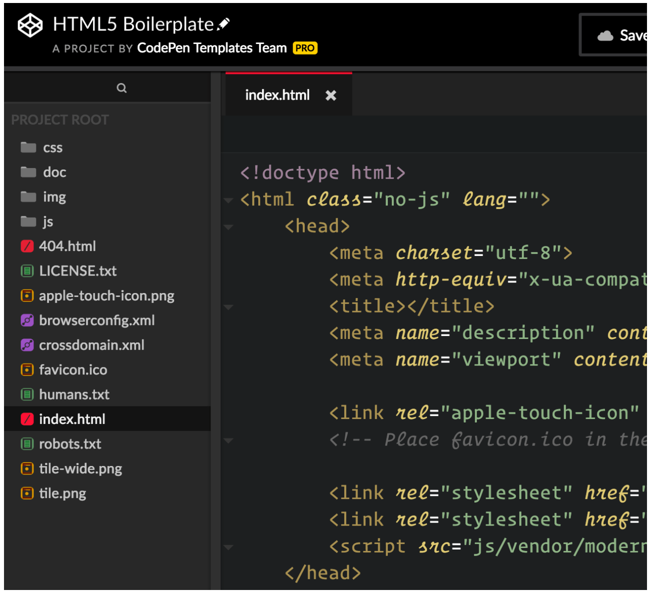
GitHub Pages is currently disabled. Select a source below to enable GitHub Pages for this repository. Learn more.

Select source

- **master branch** — Use the master branch for GitHub Pages.
- master branch /docs folder — Use only the /docs folder for GitHub Pages.
- ✓ None — Disable GitHub Pages.

automatic GitHub Pages generator. Author your content in
lish.

To publish a page manually, push an HTML or Jekyll site to a branch named gh-pages. Read the Pages help article for more information.



- css
- doc
- img
- js
- 404.html
- LICENSE.txt
- apple-touch-icon.png
- browserconfig.xml
- crossdomain.xml
- favicon.ico
- humans.txt
- index.html
- robots.txt
- tile-wide.png
- tile.png

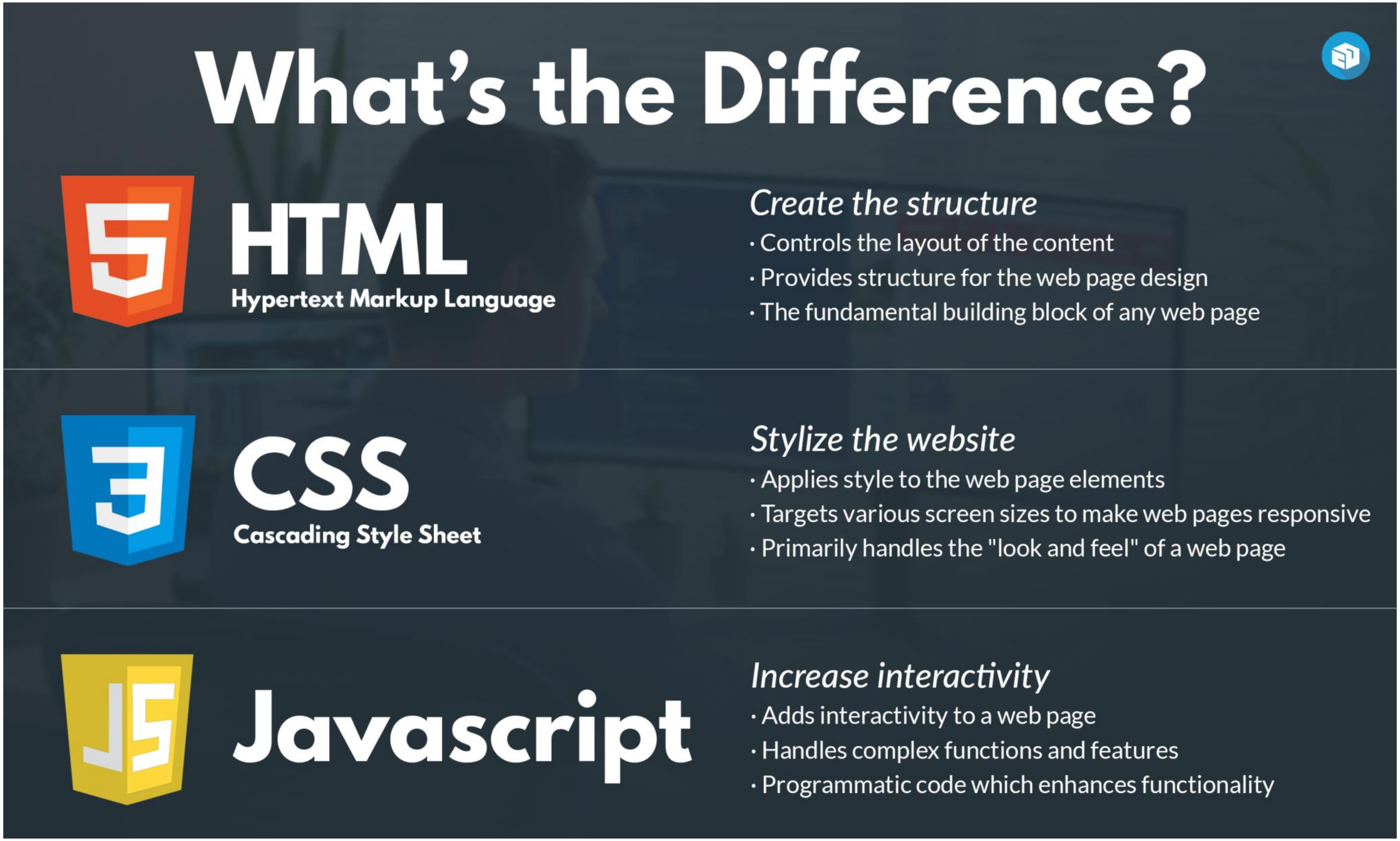
```
<!doctype html>
<html class="no-js" lang="">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="x-ua-compat
        <title></title>
        <meta name="description" cont
        <meta name="viewport" content

        <link rel="apple-touch-icon"
        <!-- Place favicon.ico in the

        <link rel="stylesheet" href='
        <link rel="stylesheet" href='
        <script src="js/vendor/modern
    </head>
```

File system in a Web Server

The most common things we'll have on any website project we create are an index HTML file and folders to contain images, style files, and script files. Let's create these now:

- **index.html:** This file will generally contain your homepage content, that is, the text and images that people see when they first go to your site. Using your text editor, create a new file called index.html and save it just inside your test-site folder.
- **images folder:** This folder will contain all the images that you use on your site. Create a folder called images, inside your test-site folder.
- **styles folder:** This folder will contain the CSS code used to style your content (for example, setting text and background colors). Create a folder called styles, inside your test-site folder.
- **scripts folder:** This folder will contain all the JavaScript code used to add interactive functionality to your site (e.g., buttons that load data when clicked). Create a folder called scripts, inside your test-site folder.

Web Documents



What's the Difference?

HTML
Hypertext Markup Language

Create the structure

- Controls the layout of the content
- Provides structure for the web page design
- The fundamental building block of any web page

CSS
Cascading Style Sheet

Stylize the website

- Applies style to the web page elements
- Targets various screen sizes to make web pages responsive
- Primarily handles the "look and feel" of a web page

Javascript

Increase interactivity

- Adds interactivity to a web page
- Handles complex functions and features
- Programmatic code which enhances functionality

The parsing order

When browsers send requests to servers for HTML files, those HTML files often contain <link> elements referencing external CSS stylesheets and <script> elements referencing external JavaScript scripts. It’s important to know the order in which those files are parsed by the browser as the browser loads the page:

1. The browser parses the HTML file first, and that leads to the browser recognizing any <link>-element references to external CSS stylesheets and any <script>-element references to scripts.
2. As the browser parses the HTML, it sends requests back to the server for any CSS files it has found from <link> elements, and any JavaScript files it has found from <script> elements, and from those, then parses the CSS and JavaScript.
3. The browser generates an in-memory DOM tree from the parsed HTML, generates an in-memory CSSOM structure from the parsed CSS, and compiles and executes the parsed JavaScript.
4. As the browser builds the DOM tree and applies the styles from the CSSOM tree and executes the JavaScript, a visual representation of the page is painted to the screen, and the user sees the page content and can begin to interact with it.

Web Documents: Casing and spacing

You'll notice that throughout this article, we ask you to name folders and files completely in lowercase with no spaces. This is because:

- Many computers, particularly web servers, are case-sensitive. So, for example, if you put an image on your website at test-site/MyImage.jpg and then in a different file you try to invoke the image as test-site/myimage.jpg, it may not work.
- Browsers, web servers, and programming languages do not handle spaces consistently. For example, if you use spaces in your filename, some systems may treat the filename as two filenames. Some servers will replace the areas in your filenames with "%20" (the character code for spaces in URIs), resulting in all your links being broken. It's better to separate words with hyphens, rather than underscores: my-file.html vs. my_file.html.
- The short answer is that you should use a hyphen for your file names. The Google search engine treats a hyphen as a word separator but does not regard an underscore that way. For these reasons, it is best to get into the habit of writing your folder and file names lowercase with no spaces and with words separated by hyphens, at least until you know what you're doing. That way you'll bump into fewer problems later down the road.

Web best practices

When doing web development, the main cause of uncertainty comes from the fact that you don't know what combination of technology each user will use to view your web site:

- User 1 might be looking at it on an iPhone, with a small, narrow screen.
- User 2 might be looking at it on a Windows laptop with a widescreen monitor attached to it.
- User 3 might be with visual impaired and using a screen reader to read the web page out to them.
- User 4 might be using a really old desktop machine that can't run modern browsers.

Screen Reader

Web best practices (Cont’d)

- **Cross-browser compatibility** is the practice of trying to make sure your webpage works across as many devices as possible
- **Responsive web design** is the practice of making your functionality and layouts flexible so they can automatically adapt to different browsers.
- **Performance** means getting web sites to load as quickly as possible, but also making them intuitive and easy to use so that users don't get frustrated and go somewhere else.
- **Accessibility** means making your websites usable by as many different kinds of people as possible (related concepts are diversity and inclusion, and inclusive design). This includes people with visual impairments, hearing impairments, cognitive disabilities, or physical disabilities. It also goes beyond people with disabilities — how about young or old people, people from different cultures, people using mobile devices, or people with unreliable or slow network connections?
- **Internationalization** means making websites usable by people from different cultures, who speak different languages to your own.
- **Privacy & Security**. These two concepts are related but different. Privacy refers to allowing people to go about their business privately and not spying on them or collecting more of their data than you absolutely need to. Security refers to constructing your website in a secure way so that malicious users cannot steal information contained on it from you or your users.

Internet v.s the web

As you might notice, when we browse the Web with a Web browser, we usually use the domain name to reach a website. Does that mean the Internet and the Web are the same thing? It's not that simple. As we saw, the Internet is a technical infrastructure which allows billions of computers to be connected all together. Among those computers, some computers (called Web servers) can send messages intelligible to web browsers.

The Internet is an infrastructure, whereas the Web is a service built on top of the infrastructure. It is worth noting there are several other services built on top of the Internet, such as email and IRC.

Internet

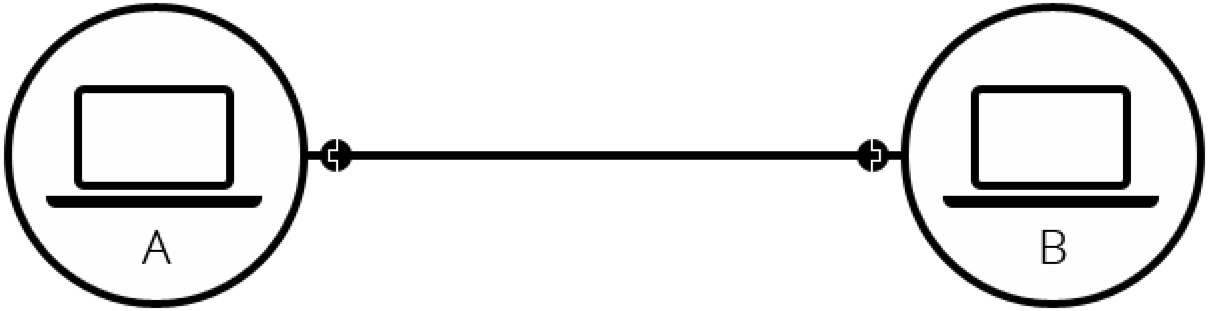
- The Internet is the backbone of the Web, the technical infrastructure that makes the Web possible. At its most basic, the Internet is a large network of computers which communicate all together.

- The history of the Internet is somewhat obscure. It began in the 1960s as a US-army-funded research project, then evolved into a public infrastructure in the 1980s with the support of many public universities and private companies. The various technologies that support the Internet have evolved over time, but the way it works hasn't changed that much: Internet is a way to connect computers all together and ensure that, whatever happens, they find a way to stay connected.

Internet Structures

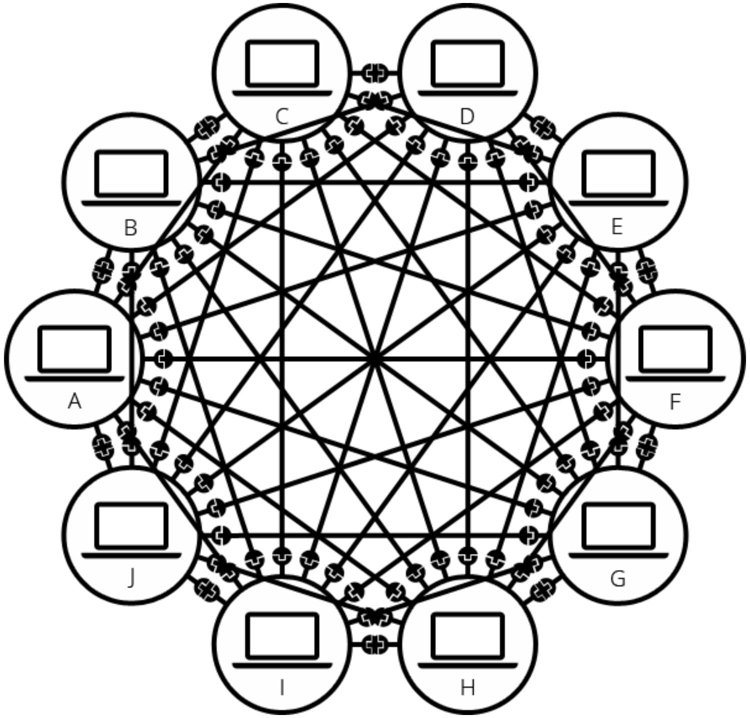
A simple network

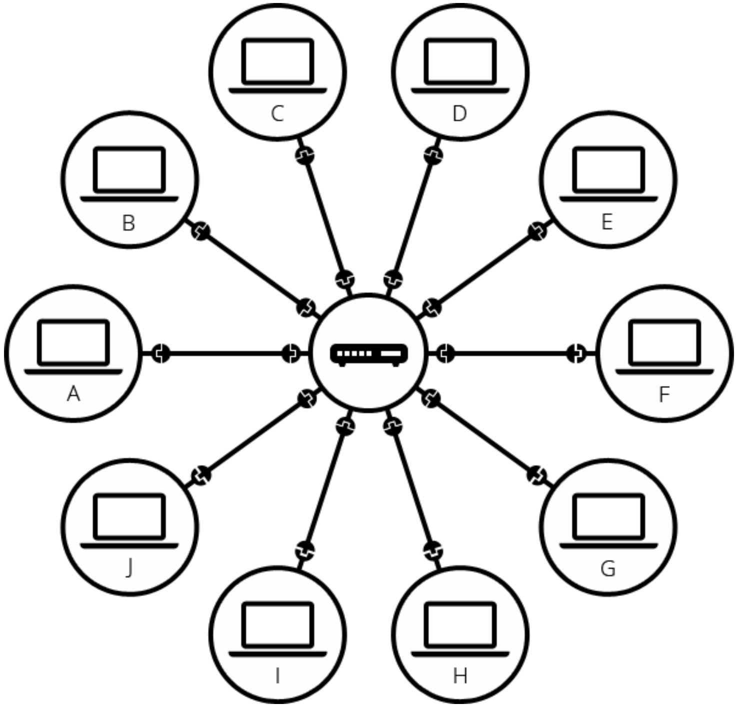
When two computers need to communicate, you have to link them, either physically (usually with an Ethernet cable) or wirelessly (for example with WiFi or Bluetooth systems). All modern computers can sustain any of those connections.



Internet Structures

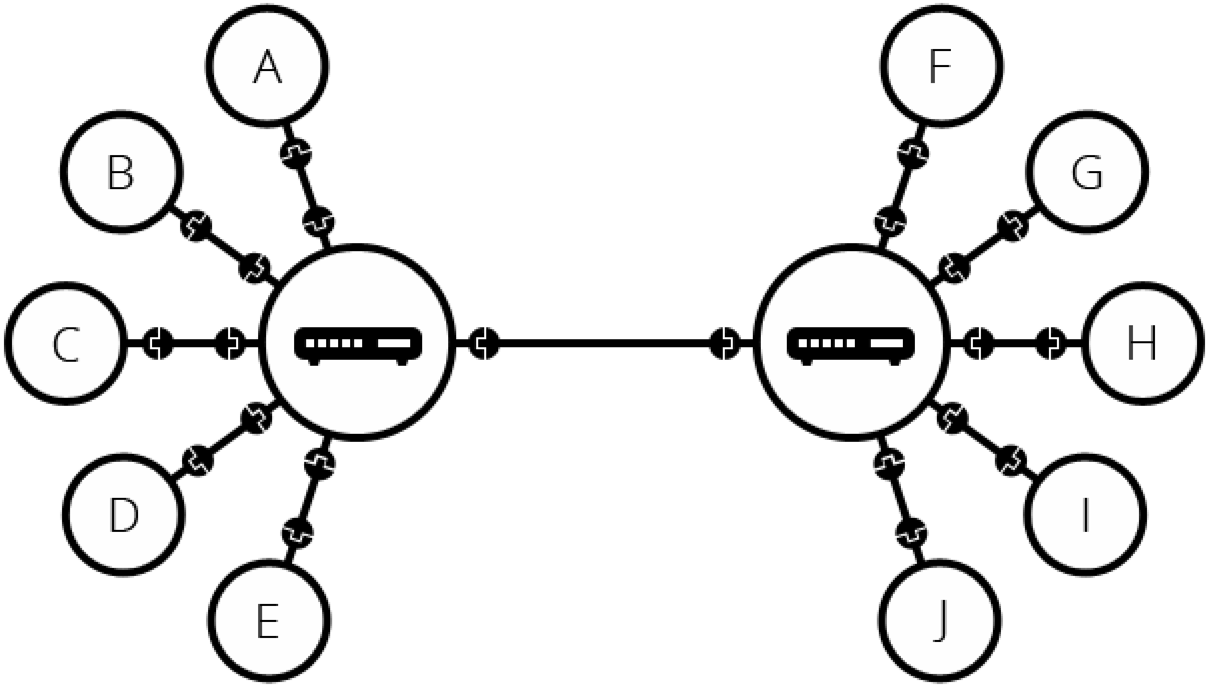
Such a network is not limited to two computers. You can connect as many computers as you wish. But it gets complicated quickly. If you're trying to connect, say, ten computers, you need 45 cables, with nine plugs per computer!





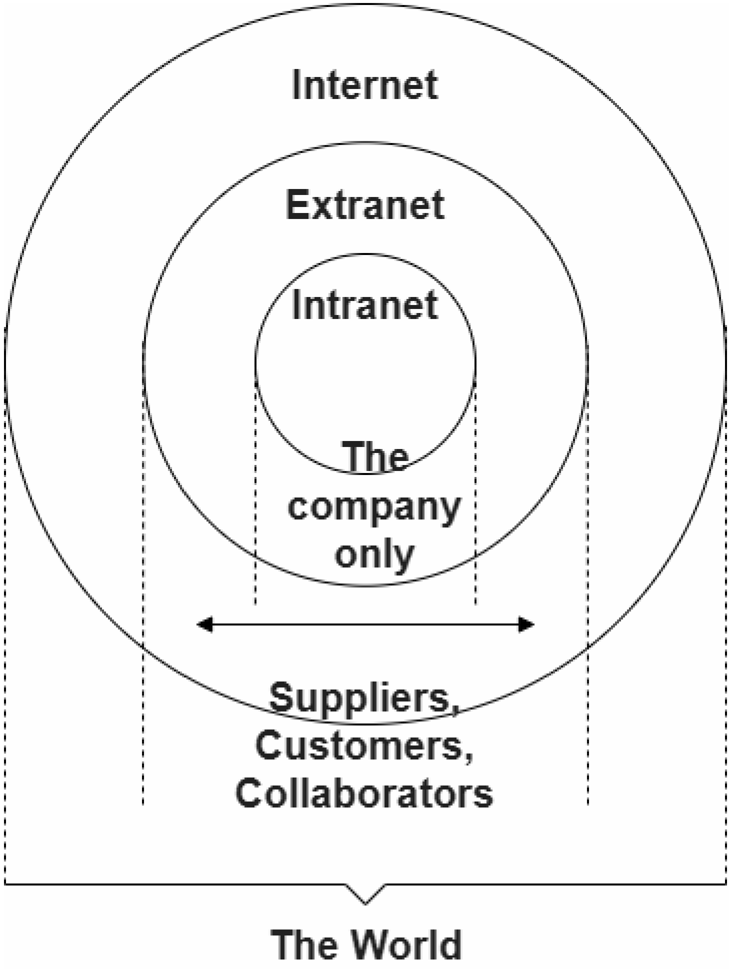
A network of networks

But what about connecting hundreds, thousands, billions of computers? Of course a single *router* can't scale that far, but, if you read carefully, we said that a *router* is a computer like any other, so what keeps us from connecting two *routers* together? By connecting computers to routers, then routers to routers, we are able to scale infinitely.



Intranets and Extranets

- Intranets are *private* networks that are restricted to members of a particular organization. They are commonly used to provide a portal for members to securely access shared resources, collaborate and communicate.
- Extranets are very similar to Intranets, except they open all or part of a private network to allow sharing and collaboration with other organizations.
- Both intranets and extranets run on the same kind of infrastructure as the Internet, and use the same protocols. They can therefore be accessed by authorized members from different physical locations.


Internet
Extranet
Intranet
The company only
Suppliers, Customers, Collaborators
The World