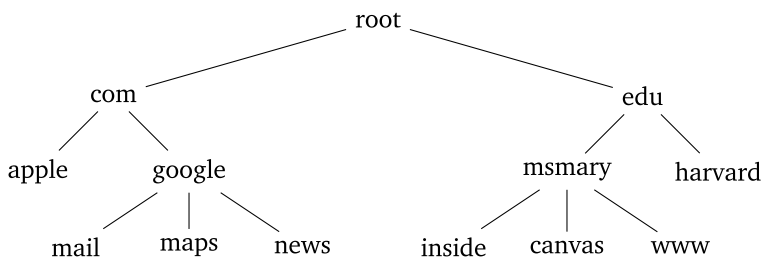
DNS

DNS is the *domain name system*. Machines on the internet have numerical addresses called *IP addresses*, like 35.206.124.218 or 45.79.189.210. In the newer IPv6 system, a typical address looks like 2607:f8b0:4004:80a::200e. These addresses are used by networking software in a similar way to how telephone numbers are used by the phone system. But IP addresses are not easy to remember, especially in IPv6, and organizations often change their IP addresses or have a whole pool of addresses.

To make things easier on humans, we use domain names. Domain names are things like `msmary.edu` or `maps.google.com`. DNS is the system that translates between these nice names and the IP addresses required by networking software to properly send around the internet.

Up until the mid 1980s, when the internet was still relatively small, there was a single file called the hosts file that was essentially a big list of which domain names corresponded to which IP addresses. As the internet grew, that file became unmanageably large and would change too frequently, so DNS was created. Instead of a using single file, DNS stores records all over the internet in a hierarchical system.

This hierarchical system is laid out like a tree. At the top of the system is the root level. The first level below it contains what are called *top-level domains* or TLDs. Below them are individual organizations; each organization can split up their domain into further levels as needed. Below is a very small portion of the overall tree.



Note that the tree can go deeper than this, though most of the domains you'll run into are on the second or third level below the root.

Subdomains Each level is a subdomain of the next higher level. For instance, in `inside.msmary.edu`, `msmary` is a subdomain of `edu` and `inside` is a subdomain of `msmary`. As you go right in a domain name, things get more general. For instance, `inside.msmary.edu` is used for the part of the Mount's website that is for people on campus. On the other hand, `msmary.inside.edu` would be something on Inside University's domain (if there even is such a school).

This is important for avoiding phishing attacks. For example, if you're doing your banking with Wells Fargo, you might log on at login.wellsfargo.com. However, if you see a link in an email telling you to click on `wellsfargo.login.com` or `wellsfargo.bheinold.namespace.com`, then you should not do so. Notice the domain right next to `.com` is not `wellsfargo` in those cases.

Top-level domains The domains at the level below the root are called *top-level domains* or TLDs. For a long time, there were only a few of them (com, org, edu, net, gov, mil, and int). Now there are many more, like tv, pizza. Also, each country was given its own two-letter TLD. For instance, uk is the United Kingdom, ca is Canada, etc. Some of these TLDs are now used for other purposes, though the TLD is technically are still administered by that country. One example is the ly domain, run by Libya, but used for sites like `bit.ly` (a URL shortener) and `music.ly` (now known as TikTok).

How to get a domain name Getting a domain name is pretty straightforward. There are various entities called *registrars* that will do this for you. It's usually just a few clicks and around $10 a year. However, if you want a domain name like `pets.com`, someone likely has already bought that domain name and will sell it to you for considerably more than $10.

Once you have a domain name, you'll likely want to set up a website at that domain. The quickest way to do this is to find a web hosting company. Again, this is a few clicks and can run anywhere from a few dollars a month to much more depending on what type of site you're running. That hosting site will take care of DNS for you. That involves setting up DNS servers that answer DNS queries about your domain. A cheaper, but more involved approach would be to host the website and the DNS on servers you control, either physical or virtual. This is a good way to learn about how the internet works.

Whois The `whois` service is used to find out who owns a particular domain name. There is a command line tool for this on Mac/Linux (it's a separate download on Windows), and there are also many websites that run it. The first few lines of a whois lookup for `msmary.edu` are shown below.

```
Domain Name: MSMARY.EDU

Registrant:
        Mount St. Mary's University
        Information Technology Department
        16300 Old Emmitsburg Road
        Emmitsburg, MD 21727
        US
```

Doing DNS lookups yourself

There are two useful command-line tools for doing DNS lookups: `nslookup` and `dig`. The `nslookup` tool is available on all major operating systems. On the other hand, `dig` comes preinstalled on Mac/Linux, but you have to install it separately to use it on Windows. Of the two, `dig` is more powerful. Here is the an example of using `nslookup`:

```
> nslookup msmary.edu
Server:  cdns01.comcast.net
Address:  75.75.75.75

Non-authoritative answer:
Name:    msmary.edu
Address:  35.206.124.218
```

The first two lines of the output give the name and address of the server (called a *DNS resolver*) that does the lookup for me. These belong to my ISP, which is Comcast Cable. Then comes the answer we are looking for, namely the IP address of `msmary.edu`. We'll talk see a little later why it's called "non-authoritative".

Here is the output of running `dig`. You'll notice that dig gives a lot more information. There are various options we can use with `dig` to get even more info.

```
> ./dig msmary.edu

; <<>> DiG 9.9.11 <<>> msmary.edu
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 59180
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
```

```
;msmary.edu.                    IN      A

;; ANSWER SECTION:
msmary.edu.             3410    IN      A       35.206.124.218

;; Query time: 20 msec
;; SERVER: 75.75.75.75#53(75.75.75.75)
;; WHEN: Sun Aug 16 22:49:07 Eastern Daylight Time 2020
;; MSG SIZE  rcvd: 55
```

How DNS works

DNS is essentially a distributed database. No one machine in the whole network of DNS servers has all the answers. Each machine is responsible for knowing a small portion of the DNS tree. And there is redundancy built into the system so that even if some servers go down, the system still functions.

There are two categories here: resolvers and name servers. Resolvers ask questions and name servers answer them. Each name server is responsible for knowing the answers to a specific *zone*. It is said to be *authoritative* for that zone. In DNS parlance, an authoritative answer is basically first-hand knowledge. Any other answer is non-authoritative, meaning it's second-hand knowledge, sort of like "I heard from this other server that the address you're looking for is 1.2.3.4".

In the DNS hierarchy, there are root name servers, TLD name servers, and name servers run by organizations with domains. Root name servers have one major responsibility: knowing the addresses of TLD name servers. Each TLD name server is responsible for knowing the addresses of the name servers for the individual organizations that use that TLD. For instance, edu name servers know the addresses of Mount St. Mary's name servers as well as all other schools with edu addresses. Finally, each organization's name servers are responsible for the domains belonging to that organization. For instance, the Mount's name servers are responsible for the addresses of `inside.msmary.edu`, `canvas.msmary.edu`, etc.

Looking back at the tree, servers at each level are responsible for knowing the addresses of the name servers at the level directly below them. In the tree, there is only one root, but that root is actually broken up into 13 separate networks of servers spread throughout the world. Each TLD is also an entire network of servers. This is important for making DNS resilient to servers going down and also so that it can handle trillions of requests a day.

A typical DNS lookup follows a multistep process. Let's say you want the IP address of `msmary.edu`. Here are the steps:

1. Your computer sends a request to a resolver to do the DNS query. If you're home, the resolver is usually running on a server at your ISP. If you're at an organization, that organization may have a resolver on their network.
2. The resolver contacts a root name server. The root name server returns the address of a TLD name server (in this case it's an edu name server).
3. The resolver contacts the TLD name server. That name server returns the address of the organization's name server (in this case Mount St. Mary's is the organization)
4. The resolver contacts the organization's name server and gets the address it is looking for. It then sends that back to your computer.

It's possible for this to go a few steps further if the organization itself breaks their DNS into multiple zones.

Caching This multistep process is pretty slow as it involves contacting several different servers and waiting for replies. If this process needed to be done every time you need to visit a web page or make a connection, things would take forever. So after a resolver gets a final answer, it *caches* the result, which is to say it remembers or stores it. Your web browser also caches results to save the time of contacting the resolver, and your OS might cache results, too.

How long will the answers be remembered for? This is determined by something called the TTL (time to live). This is a value set by the administrator of a domain's name servers. For instance, the people that run the Mount's name servers decide what the TTL should be for `msmary.edu`. The TTL value can run anywhere from a few seconds up to several days. Organizations don't want the TTL to be too short because otherwise you lose the benefits of caching, as entries will expire so often that tons of queries will have to be made. They also don't want it to be too large because if the organization needs to change addresses, people will be using the old, cached address instead of the new one. Long TTLs also mean that the cache may fill up with too many entries.

While connected to the Mount's network, I ran `dig gatorade.com`, and got the following:

```
;; ANSWER SECTION:
gatorade.com.           7199    IN      A       45.60.75.51
```

The value 7199 is the TTL in seconds, which means the entry is valid for about 2 hours. Anytime in the next two hours someone on the Mount's network asks for the IP address of `gatorade.com`, the Mount's resolver will return the cached value 45.60.75.51 and not do a full lookup. After those 2 hours are up, if someone asks for `gatorade.com`'s IP address, a full DNS multistep lookup will have to be performed.

Just for effect, after writing the above paragraph, I reran the dig command and got the following output. Note that the TTL is down to 6959, meaning about 4 minutes have elapsed.

```
;; ANSWER SECTION:
gatorade.com.           6959    IN      A       45.60.75.51
```

You can actually use the TTL to tell if other people on your network have been to a specific domain recently. If you run dig for that domain and get an usual TTL, like 6959, it's likely that you're seeing a cached result. The dig command will also tell you how long it took to do the query. Try a few different domains. Ones that are cached will take significantly less time because there doesn't need to be that multistep lookup.

Resolvers A resolver is the machine that does the lookups for you. When you connect to a network a process called DHCP (which we'll cover later) will give you the address of the resolver for that network. However, if for some reason you don't want to use that resolver, you can use configure your OS's network settings to use a different resolver. In particular, 8.8.8.8 is Google's public DNS resolver. A newer one is 1.1.1.1 run by Cloudflare.

This is important because not all resolvers are honest. Say you are on the Mount's network using their DNS resolver and you ask for the IP address of a competitor, like `loyola.edu`. The Mount's admins could configure their resolver to always return a bogus IP address instead of the real Loyola address. To get around this, you could use one of the open DNS resolvers mentioned above (assuming that you trust them). There have been many cases in the past of ISPs doing shady things with their resolvers. One of the most famous cases was when the country Turkey blocked address to Twitter by having all the resolvers in the country return bogus answers for DNS queries for Twitter. People spray-painted 8.8.8.8 on the sides of buildings to tell people how to get around it.

A related topic is the *hosts* file. This is a file on your computer that has hard-coded domain to IP address mappings. For instance, you can put an entry like `msmary.edu 144.126.4.114` into the file. The address `144.126.4.114` is Loyola College's IP address. Usually your OS will check the hosts file before contacting a resolver. So when anyone on your computer tries to go to msmary.edu, then will instead end up at Loyola. The file is located at `/etc/hosts` on Mac and Linux and at `windows/system32/drivers/etc/hosts` on Windows. Note that `etc` is a hidden directory in Windows.

DNS cache poisoning There's a whole class of attacks called DNS cache poisoning that involve attackers intercepting DNS queries and returning bogus answers to them. It takes some skill to do these on an ordinary network, but it's relatively easy to if the attacker sets up their own Wifi access point.

A typical way this happens is as follows: The attacker gets a USB Wifi device for a few dollars that lets them use their laptop as a Wifi access point. They also download a common DNS server software called Bind that lets them use their laptop as a DNS resolver. They set up shop somewhere in public and wait for people to connect to their wifi. When people connect, the attacker's access point tells the victim's computer that the resolver to use is the attacker's resolver. When the victim tries to go to something like Gmail, the attacker's DNS will instead give them the IP of a phishing site that looks just like Gmail. The attacker then will get the victim's login credentials.

DNS resource records A DNS resource record is the actual information returned by a DNS query. Here is a typical resource record returned by a call to `dig`.

```
gatorade.com.           6959    IN      A       45.60.75.51
```

We see it is broken into five parts. They are the domain name, the TTL, the class, the record type, and the value. Here is what they mean:

- Domain name — The name of the domain that the info was requested for.
- TTL — How long until the record expires from the cache.
- class — It's almost always IN for "internet".
- type — What type of record this is. There are several types, which are described below.
- value — The actual answer to the question being asked. In the example above, it's the IP address.

Here are some of the more common record types:

- A — Ordinary IPv4 address request.
- AAAA — IPv6 address request.
- MX — Requesting the address of the domain's mail server.
- CNAME — Used for aliases. Sometimes the domain name an organization wants the public to use is different from the domain name they want to use internally. As an example, say we run the command `dig CNAME msmary.instructure.com`. Note that this address is what we use for accessing Canvas. Here is what we get back:

  ```
  msmary.instructure.com. 269     IN      CNAME   cluster87.instructure.com.
  ```

  The company Instructure, who runs Canvas, internally stores Mount St. Mary's Canvas stuff at the domain `cluster87.instructure.com`. Presumably, Instructure stores our school's stuff at cluster87, some other school's stuff at cluster88, etc. Mount students don't want to have to remember that our stuff is at cluster87, so they give us the nicer domain `msmary.instructure.com` by using a CNAME alias. Moreover, Instructure might in the future move us to somewhere else, like cluster144. By using a CNAME record, all they will have to do is update that record and not have to worry about notifying us that we're now at cluster144.
- PTR — Used for reverse DNS lookups (looking up a domain name based on the IP address).
- SOA — Has administrative info about the domain.
- NS — The names of the domain's name servers.
- TXT — Catchall for various information that doesn't fit into one of the other types.