And if I keel over and fall down, somebody call the ambulance.

OK, what we're going to do today is, this is actually the last lecture on a topic that some might consider part of networking, but some might consider part of a more general topic of distributed systems.

So it actually forms a bridge between the stuff we learned about in networking, and what we're going to see from Wednesday over the next six or seven lectures of the class, and of the associated recitations having to do with fault tolerant, reliable computing.

And most of the interesting aspects of what we're going to talk about involve techniques in fault tolerance, and more techniques have to do with redundancy and replication.

And DNS, the domain name system, which is the system we're going to look at today in the context of distributed naming is a bridge because on the one hand it covers some of the aspects of networking that we talked about.

On the other hand, it shows an example of how you can achieve replication to achieve fault tolerance.

And were going to study these techniques systematically over the next few lectures.

So getting an example in mind is usually a good idea.

So you've already seen the network layer, and you've seen that.

At the network layer, attachment points on the network are identified by IP addresses in the internet are more generally identified by network layer addresses.

So as an example on the Internet are IP addresses.

And, this is the name that's used by the network layer to identify attachment points anywhere on the network.

And in fact, there's a name that's used by the end to end layer to name one endpoint of connection.

So in fact if you think about it and go back to our very early lecture on naming, an address is really just a name, but a name that's been overloaded with information that allows the user of that name to locate this object.

An address, really, is a name that has information in it, overloaded information in it that actually allows for it to be located.

And in fact, an IP address is nothing other than a name that tells you where in the Internet topology, the entity

being named by this IP address is located.

So my computer is something like 18.31.0.35. It doesn't mean anything other than the fact that somewhere on this Internet topology there is this big, complicated graph.

And that address allows you to do routing in the geography of that topology.

It has nothing to do with real-world geography.

It just allows you to do routing in that topology.

Now, in principle, you could build every Internet application, and have users interact with Internet applications purely with these network layer addresses.

But that would be quite inconvenient.

I mean, you would then have to be sending e-mail to your friends, or not, with Joe at MIT.edu. But, you'd have to do Joe at some IP address.

And, it's pretty hard and complicated to remember.

So the first problem with just using pure network layer addresses that we want to solve today, and we will solve to some degree, although not completely is to come up with a better way of naming things that are more convenient.

And you already know the answer.

The answer is you send e-mail to Joe at MIT.edu. You go to the 6.033 website at MIT.edu/6.033 or some other equivalent thing that leads to the same page.

You don't actually think about names in terms of IP addresses.

So in fact, to a large extent, the fact that these are human understandable and names that are mnemonics that you can easily remember is a good thing.

And so we do actually want to come up with a way of naming things that's independent of IP addresses.

The second goal here is to come up with a naming scheme with a solution that allows some degree of modularity.

As you know, names provide a level of indirection between the thing that you want to get to, and the handle that you want to associate with it.

And that level of indirection, if we come up with a good way of doing this, it will allow us to do a few things like, for example, I can tell you that the website for MIT.edu, for the institute's homepage is MIT.edu. And, behind that, I could change the actual computers on which the website is located.

And I could do that independently of telling other people of any change in it, whereas if I told them that the website was at a particular IP address, then every time I moved a page, moved the pages from one computer to another, I have to tell everybody in the world that the website has changed.

And we'd like to minimize doing that kind of thing.

And so, the domain name system provides a solution to this problem.

DNS provides a solution to this problem.

Most of you have already heard of this.

It maps between what are formerly called domain names, but what we are going to just call host names for convenience.

It maps between host names and records.

And it turns out there are many different kinds of records.

And we're going to look at a few of them in this lecture today.

But for your mental model right now, just assume that it maps between host names and IP addresses.

So it turns out that an IP address is just one example of a record called an address record.

But the general goal of DNS is to map between host names and records.

And, there is a variety of them, as I said.

For now, just assume they are IP addresses.

So for example, MIT.edu might be 18 dot whatever it is as an IP address.

So what are the goals in designing the system?

So, primarily we're going to be talking about how the domain name system is designed, and how it works.

And as we go along, we'll see some things that it does that are different from other systems or other ways you can

solve this problem.

There are basically two goals in the design of the system.

And the first goal is that it should scale.

In fact, the original motivation for the domain name system when it came about in the early 80s was that it was becoming extremely inconvenient to manage the mappings between names of hosts and their IP addresses in a way that as the network grew, and it was growing pretty rapidly even then, as the network grew it turned out to be a management nightmare.

And to understand this, basically there's three ways in which you can imagine; there are more than three ways.

But there are three more obvious ways in which you could imagine mapping between these names and these records, so the names and IP addresses.

And the first one is, in fact, the way the Internet names were being managed until DNS came along.

And that's to use a model that you might think of as the telephone book model.

It's actually astonishing the telephone companies use the telephone book model where every six months at your doorstep there are these three thick books that show up.

And you're just looking at it and going, what do I do with this?

And you lug it to your home, and by the time you might use a couple of times, and then the next big three books come along.

You actually wonder why they don't just put their phone books on the Web and make it easy to get at.

But the telephone book model really is, there's this central repository of information.

And everybody gets this information from the center repository.

But the repository actually pushes it to everybody.

So it's not really a queryable (sic) repository.

It's not something you go in contact on an as needed basis.

So, in fact, every few months, or in the old days of the Internet, everyday in the morning, every computer would go

and pull the current mapping between names and addresses from a central site.

And this model kind of works for a little bit, but stopped working.

It stops scaling on multiple levels.

First of all, the resources required for everybody to have to go and collect this information every day turns out to be significant.

But it also turns out to be a scaling bottleneck more fundamentally from the standpoint of any time you add a machine to your local organization, you have to go and tell the central person that you've added a machine.

And so, at a human level it doesn't scale very well because you can't allow for this automatically happen because then people would sort of willy-nilly do all sorts of, you know, just claim that they own various machines or various names at various places.

So the telephone book model is something that the Internet used to have.

It used to be a file card host.txt. And, every computer had a copy of this file that usually was current as of a 24 hour period.

So, the second approach that you could adapt knowing this approach of just sort of pushing a telephone book periodically doesn't work.

You might actually adapt a model, a centralized server model.

And the central server model is, the main difference from the telephone book model is that nothing is pushed to you.

Instead, imagine a search engine like, say, Google or something like that where if you want to know the mapping between a name and in it the address, you go and contact this name service, which turns out to be a central server.

And it does essentially the same thing that Google might do.

And this actually isn't that hard to implement from a technical standpoint because Google does much more than this.

And clearly it works.

But the real problem with the central server model is what I alluded to the last time.

It doesn't handle, when you assign names to machines, you want to make sure that people don't conflict on these names because these are well-defined names for machines on the Internet.

And you need a model by which you can decide who is allowed to name a machine as fool.MIT.edu and who's allowed to take X.CNN.com, and so on.

And so, for every computer in the Internet, having a central person deal with deciding whether that's OK or not isn't a scalable solution.

And that's why we don't really adopt that model for DNS.

The model that's adapted for DNS, and has a number of other attractive properties, which we'll talk about, is the distributed database model, or more generally, a distributed, federated model where every organization, and organizations could be recursively defined to have sub-organizations. Every organization sort of manages a portion of this overall global namespace.

And it manages everything about it.

It manages all of the mappings between names and records for that part of the namespace.

And the more names they have, the more work they have to do.

But nobody else really has to do that much more work.

And it's a pretty nice model because everybody does some amount of work in terms of technical resources, and more importantly in terms of human administrative resources.

And overall, that's one component of the scalability of the system.

And that's one of the main reasons why the system turns out to scale and work very well.

And the second goal is reliability.

Once you have a system like this, and people start getting used to it, and applications start using names, it had better be the case that the system is in fact generally available.

And it better not be the case that the DNS be the Achilles heel of the Internet in terms of, you know, the network infrastructure has a very, very high reliability because all this running stuff works out great.

And we find all these alternate paths.

And things don't work because I'm not able to get to my DNS.

That had better not be the case.

So we'll see what techniques DNS uses to get pretty good reliability. But the jury is still out as to exactly reliable it is.

But overall most users will admit that generally it seems to work about as well as the rest of the network.

So, what does DNS do?

Fundamentally, it provides an abstraction called a lookup abstraction.

You give it a name.

It returns a record to you.

And you can ask for particular types of records that you want.

We'll get into that in a second.

So you ask a name, and you get back a record.

And, there's many ways, and different operating systems have different ways in which this exact function is called.

For example, get host by name might be a commonly used way of going from a name to an IP address.

But, we are going to just say DNS resolved just more abstractly.

So an application can invoke DNS resolve, DNS name, and get an IP address or some other record associated with it.

Now if you think back at the way we did our idealized naming model or genetic naming model, we actually had a resolve which took a name and a context as an argument.

And, it returned back the value that was bound to that name.

So you might ask what the context is for a DNS resolve.

And it is actually multiple answers to this question.

The first answer is that applications specify a context usually.

So, for example, an application might specify that.

It wishes to know the IP address of this name.

For example, if it's a web browser that wants to know some server's IP address, so it can connect to it using the TCP connection.

Alternatively, an application like an e-mail program might specify that it wants not the IP address of this machine, but the name of a machine that can handle mail on behalf of this name.

So, for example, if I send somebody email to ABC.MIT.edu, my mail program, somewhere along the way some server would do a lookup for a special kind of record turns out to be called the MX record, which is a mail record, which would then return to caller not the IP address of MIT.edu, but the IP address of, in fact, we learn a name of a machine that's capable of handling mail for MIT.edu. And in general, that would be very different from the IP address associated with MIT.edu. So that's established as a context.

And more generally, there is a DNS configuration file.

There is some DNS configuration on registry, or whatever it is depending on the system that you have that tells this resolve step what context in which the name must be resolved.

So just for example, you might have a machine, cricket.MIT.edu, another machine cricket.berklee.edu. And DNS resolve might call DNS resolve of cricket.

And when the application calls that, the program doing the resolution needs to know what context to do it in.

And often, that's specified in the DNS configuration.

So on Windows and UNIX, there is this thing called a search path.

And you go through a set of search paths that provide this context.

So will see this in more detail later today.

Now users themselves, and there's an important point here about lookups, and how lookups are to be distinguished from something else that are called search.

Now, these names are generally very useful for programs because they allow modularity to occur, where it no longer are you worried about services being associated with IP addresses.

You can move your website between IP addresses of the back without telling everybody about it.

So from that standpoint this name to record mapping a DNS with lookups is very, very useful.

In terms of convenience, the DNS is not the whole story because although you most often send email to people with fool@MIT.edu so you remember that or you have some other file in which you find that information very easily, more generally speaking people often don't, we need something in addition to just lookups.

Human users need something in addition to lookups.

And the general term given to that is search.

So for example Google provides search on the Internet.

And tomorrow's recitation talks a little bit about one aspect of that.

And in fact, it's an interesting discussion because you will find from the reading tomorrow that users were using a search engine to essentially do a lookup task where they would go to Google and type CNN.com when in fact if they already knew to CNN.com they could just as well have typed it on their URL window.

And that's sort of the way real users turned out to use the Web.

But over peer-to-peer applications that most of you might be more familiar with than I am have a form of search.

And those applications are interesting because they do searches on all sorts of attributes of the content that you want, and by and large don't really use DNS in a particularly, if they use it at all it's sort of incidental.

They probably don't even need to use it.

So it's not like all Internet applications require DNS in order to work.

In fact, there are plenty of applications that don't need DNS at all.

So, and they benefit a lot from search.

OK, so let's get back to DNS and talk a little bit about a few different topics.

We're going to start first with the namespace, and how it works.

And then were going to talk about how name resolution works.

And then we're going to talk about things like performance and scalability and robustness.

So the DNS namespace has two properties to it.

The first is, in both of these are nice ideas, and hit upon a theme, or at least one of which hits upon a theme we've covered in 6.033. DNS is a hierarchical system.

And it's a structured namespace.

I'll describe what structured means in a moment.

So the way our DNS namespace works is it's a hierarchical system which is structured as a tree.

And at the top of the tree there is a little circle which really is a dot.

And I'm going to call that the root.

So, everything starts at the root.

And the root is the root of this namespace.

And it's a tree.

The namespace is divided into A, a bunch of domains.

And domains are divided into subdomains.

And subdomains are recursively divided into sub-subdomains, and so on.

And in fact, the depth is arbitrary.

It could be arbitrarily long.

In practice, nobody really has need for depth more than four or five.

And, in fact, 90% of the names, or more than 90% of the names are pretty flat.

You don't go more than two levels down.

So at the top level, this is pretty familiar to most of you.

You would have com, and edu, and net, and org, and gov, and a few others.

And these things, there is actually about 13 of them right now.

These things are called generic, top-level domains, so generic in the sense that they are not really associated with any country, for example.

And then in addition to this, you have a whole bunch of country codes like dot US, and I don't know how many countries there are, but there's a large number of them.

So those things are country code top level domains.

And they are not that interesting in that there's nothing different about them from anything else.

So we may as well just look at the generic top-level domains.

OK, and edu gets divided into MIT and other places that don't matter and so on and so forth.

So you might end up down here.

You might have www, or website, or for whatever the student's from.

I don't know who actually owns this.

C sale might be here.

EECS might be here.

And I might have a machine underneath here, let's say, X just around a machine.

And, the thing about the DNS namespace is that every label here, this is a label, right?

So the way you read this is if you start at any label here and go upward, you can read it out from bottom to top.

So you would say com is an example of a label.

MIT, edu, root is a label.

So that's read as MIT.edu dot.

And usually people omit the trailing dot because it's implicit.

Or you X.csale.MIT.edu dot is another fully formed, what's it called, fully qualified domain name.

OK, that's sort of the correct way to read it out is from bottom to top.

Now, every node here is associated with some information.

And that's what this record means.

OK, some nodes might have nothing in it, but in general, every node is associated with some information.

So, for example, X might have information here.

I'm just going to call it INFO for now.

But, X might have associated with it an A record, which is an IP address record.

I'll describe that in a moment.

But it might have other things associated with it.

In fact, DNS is pretty flexible.

You could define your own record and put that into the system, and have applications that read from it.

It's pretty opaque to it.

It doesn't really require; if you have something new that comes up and you want to use it, you could stick it into DNS and retrieve it out.

So people put all sorts of things now into DNS.

For example, there are weird proposals.

But in GPS coordinates of a name.

So if you know MIT.edu isn't going to move very much, then put in the GPS coordinates.

You might find applications that find it useful.

For example, if you are some mobile computing application, you might find it useful.

So there's all sorts of things you could stick into the DNS.

And people have come up with all sorts of very wacky things, including telephone numbers in DNS.

It's very flexible.

So not only are the leaves associated with things with information, but you can have information at any level, any node in the tree has information associated with it.

And the scale of this namespace today is extremely vague.

I mean, I don't know the exact number of things there are, but I've read that about 500 million, I don't know if it's 250 million or 500 million, somewhere in between, there are that many registered names in the system in aggregate.

That's a pretty big number.

Now, it's a very big number.

So you need a way in which you can make the system scale.

And that's done using a really nice idea called delegation.

And more than any technical decision that was made in DNS, I mean, we're going to talk about some of them like caching and all this other stuff.

But more than any technical decision, delegation is really the reason DNS scales.

And, it's ultimately really the reason why DNS is pretty successful.

So, what is delegation?

The best way to understand it is recursively.

So the root at the top is centrally owned, and it's owned by a trusted entity.

So what that means is that any name that ends in root, ultimately the authority for that name rests with whoever owns and runs root.

And if you paid attention to the press, and the newspapers, or magazines, you'll see that there's this fight for the root that's ongoing right now over the past few years.

And, the current owner of the root and things associated with it, and who essentially controls the namespace, is an entity called ICANN, I-C-A-N-N, and there's a lot of politics associated with it.

Now, continuing down on the delegation idea, the next layer down from the root, the technical term for it is top-

level domain because it's at the top level: TLD, OK?

And, these top-level domains that are delegated from the root, and it's really hard to come up with a new top-level domain.

In relation to these, you have a few more.

But you don't really come up with them willy-nilly. You kind of have to go through a long procedure before the root decides to delegate the top-level domain to somebody else.

And now it's recursive from here on.

Every label here can be sub-delegated arbitrarily by only contacting the owner of that label.

So once you get to com, you don't have to go further down.

You just have to go to whoever happens to own the com label and tell them that you want to register a new portion of the namespace with that.

So, for example, MIT must have gone at some point to the owner of edu and said, I want MIT.

And, there's some out-of-band human procedure that occurs before the other party is convinced that this MIT is a legitimate entity, and allocates this name to it, and likewise.

Whoever wanted CSALE went to the person who runs MIT's name.

This is called a zone, this DNS namespace and told it that it wanted CSALE, and established by human efforts rather than anything technical that it wanted that portion.

Now, the reason it scales is you can kind of add machines, you know, names at the bottom, not machines but names anywhere here without really having to go all the way up.

You just need to go up to whoever owns your parent label and convince them that you want a name.

So if I want to add a machine, Y, I don't really have to go and talk to even anybody at MIT.

I just have to talk to the person who runs the CSALE namespace and tell them that I want a name, Y.

And I can, then, sub-delegate that name.

I could, today, connect a computer X.CSALE.MIT.edu have in this IP address with it, and tomorrow decide I don't

really want X to be a computer.

I want it to be the name of my research group or whatever, and then have machines underneath it which are Y.X.CSALE.MIT.edu. What I do with the label is my business.

And there's no rule that these are IP addresses.

In fact, these are nothing.

These are just labels.

And I can associate arbitrary amounts of information I want with that label.

So, domains can be formed anywhere in tree.

And that's really nice because you don't have to go back to a central entity in order to add these names.

And that's the main reason why the central server model doesn't really fly.

OK, so examples of records, we've already seen a few of these.

So let's look in more detail at what this info could contain.

Like I said, it's very flexible.

You can have all sorts of things you add in here.

But there's a few very common ones.

The first one is called an A record, which stands for an address record.

And, it's what you might expect.

It's an IP version four address for a name.

So, X.CSALE.MIT.edu, whatever its IP address is.

So, that's stuck in this database.

It's really maintained in a file on the name server that handles that name.

You might have MX, which stands, the X is for mail exchanger.

So, it's mail exchanger.

So, that's for email.

So, when I send email to you@MIT.edu, somewhere along the way there's a lookup done for not the IP address of MIT.edu, but the MX record for MIT.edu. And in general, the MX record could be anywhere.

If MIT decided to outsource its email functionality to some other company, the MX record would just point to some name of a mail server in that other company.

So it doesn't even have to be local to us.

There's another one that's interesting in the context of stuff we've seen before called a C name, which stands for a canonical name.

But a C name is really a synonym.

This is where you can say there are many names that really mean the same thing.

So, for example, to take a very real example, there used to be AI Lab and LCS, and now there's the same lab, CSALE.

Now, there are a lot of subdomains from LCS.MIT.edu, and AI.MIT.edu. And sort of it's a nightmare to have to go and change all of the DNS entries for all of the machines.

So the standard way in which you manage this kind of thing is to set up these C names, which says [UNINTELLIGIBLE] NMS.LCS.MIT.edu. That's what it was.

You just set up a C name that says NMS.LCS.MIT.edu, there's a C name for NMS.CSALE.MIT.edu. So, you don't have to do anything more other than set up these synonyms.

And everything else just sort of continues to work out.

There are other useful things you could do with C names.

For example, if you decide you want, you're running your Web server on one machine, and then you want to change it over to another machine but not have to tell the whole world about it, what you do is you tell everybody that your Web server's running, let's say, MIT.EDU. And then you set up a C name for that MIT.edu to machineone.MIT.edu. It's another name.

And then someday you decide to change from machineone.MIT.edu to machinetwo.MIT.edu. All you have to do is

to change this one mapping in the backhand in the DNS that maps from MIT.edu, set up a C name mapping to a different machine, and that's all you have to do.

You don't have to tell anybody else about the change that you've made.

So, it's very useful.

And this is just an example of a more general concept that we've seen already called a synonym.

In the fourth thing, which is actually what we're going to spend most of our time on today, for the rest of today, is called an NS record, which is a name server record.

And, this is the thing that's really going to help us figure out how to implement DNS resolve in a scalable way.

I will describe what a name server record is in a moment.

So if you look at this tree, associated with many of the labels in the tree are not just A records, which are generally associated, but also things called NS records.

And, what an NS record says is that, let's say there's an NS record associated with MIT.edu. What it says is that that NS record gives the name of the machine that's responsible for managing all of the names that end with MIT.edu. So, for example, edu wouldn't know anything about, in general, edu may not know anything about how CSALE.MIT.edu is really mapped.

But all edu needs to know is what the NS record is for MIT.edu so that as you'll see in this procedure, you'll see that occasionally things at the top of the tree will get requests for a given name, and they need to know what to do with the full name.

And, the way they'll do it is they'll find out that they don't know anything about the full name.

But they know about other people who know more about that name.

And that's going to be implemented using this thing called an NS record.

So I'll show this with an example.

I think it'll become pretty clear.

So the way in which applications use DNS is you have an application that wants to called DNS resolve on a DNS name.

And, there's a piece of software that usually runs on every computer called a stub resolver.

OK, and a stub resolver is just something that allows applications to not have to worry about this whole RPC mechanism that DNS involves, that DNS entails.

So, that's just [UNINTELLIGIBLE] way into the stub resolver.

So the stub resolver really does all of the work.

So the way the stub resolver works is, and the way DNS resolution works is that the stub resolver really can send a DNS request that it has from the application to any name server that it wants to.

And later on we'll talk about how you pick this name server.

And there are lots of these name servers around.

On the internet, it's a massive distributed infrastructure.

And, the infrastructure consists of people, of nodes that have responsibility for different portions of this namespace.

So, you send a request to any name server, and they all participate in this system together, these name servers to help resolve names.

So let's take an example here.

Let's say it's X.CSALE.MIT.edu. So, at this point in general the name server knows nothing about X.CSALE.MIT.edu. And it's plan's going to be that it's going to send this request out to the root name server, OK?

And for now, assume that the root name server is well-known, that is, the IP address of the name server in charge of the root.

So everybody has a name server that got associated with themselves.

Assume that the IP address of the name server of the root is just well known to everybody, OK?

So it sends X.CSALE.MIT.edu all the way to the root name server.

And the root name server is actually going to look at this thing and say, well, you want to know the A record.

You might want to know the A record for X.CSALE.MIT.edu. It says, but I don't actually know what the IP address

associated with this name is.

But I do know somebody who can tell you more because I do know who runs the name service for edu.

So, it comes back with the response that's also called a referral saying, well, I don't know the answer, but here's where you need to go in order to find out more.

And that's done by sending back the name server, or in fact more generally a set of name servers, but sending back the NS records for nodes that handle the edu domain.

So that just comes back at you.

And now, you now know one or more name servers for the edu domain.

So, let me write that here.

And this name server then goes back to this edu domain and says, OK, tell me what the IP address for X.CSALE.MIT.edu is.

And, notice that at all stages, it's sending the full name because there's always some chance that these nodes have the answer, and we'll get to why that might be in a few minutes.

But you always send the full name.

And the edu name server record in this case is, in general, going to say, well, I don't know about X.CSALE.MIT.edu. But I do know that MIT came and delegated from me.

So I do know the name server record associated with MIT because it delegated from me.

And in general, everybody has to know the name server records for the people one level down from them.

So it sends back this information saying, well, I don't know what it is.

But here's a referral to the name server for MIT.edu. And this procedure basically continues.

So this is MIT.

Actually it's just the MIT.edu name server.

And you just go back and forth until eventually you get to the main server for CSALE.MIT.edu, which by definition maintains the mapping for everything of the form NAME.CSALE.MIT.edu. And so, you get the answer.

Or you get something that says X is not actually registered in which case you get a no-such-domain error message.

Actually it's a no-such-domain error code, which you then interpret as saying, OK, there is no such name that's been registered.

Now there's a couple of things of note here.

The name server records associated with the domain really have to have very little to do with that domain.

In fact, the name server record for, forget the root names for a moment.

The name server records for the edu domain don't actually have to end in edu, or don't have to end in anything that relates to edu.

In practice, in fact, today they're of the form something.NSTLD.com. I mean, they have nothing to do with the edu domain.

And, this is an important point.

You could associate here any label, any name of a name server that's willing to manage the delegation, manage the entries in your database.

They don't actually have to match the same domain name.

And, this is a very powerful feature of the namespace of the way DNS works because it's not like this thing has to be something.edu, and here the name servers for MIT.edu have to be actually something.MIT.edu. In practice, edu is not managed by anything.edu. It's something, NSTLD.com. In practice, it so happens that MIT.edu happens to run its own name servers; that's something like bitsy.MIT.edu. But that's by no means a requirement.

So there's a couple of problems that we need to solve about this basic mechanism it does what we saw was that once you know the root name server record, then you could go back and forth because everybody knows how the names one level down from them have been delegated and knows the name server entries for those delegates.

And then you get the final answer.

So there's a few things we need to solve.

The first one is bootstrap.

There's actually a couple of aspects of this bootstrap.

The first one is, how does this any name server here know the name servers of the root?

And, the answer here is that there's no magic.

I mean, you just have to know.

And, in some sense, in all naming systems, ultimately there is, at some level of this name discovery procedure, at some level there's out of band machinery that has to get in and be involved.

And the way this works out is that people publish the name server records for the root.

They post it on websites.

They publish it on mailing lists.

And you can figure DNS software is configured to manage those entries.

And there are proposals and protocols for automatically updating it and so on.

But you've got to be very careful with technical solutions like that because you want to make sure that malicious bodies don't pretend that they're telling you the correct name server entry because once they usurp DNS functioning, they could do a lot of damage.

In practice, in fact, DNS is not particularly secure.

It's a different discussion as to whether that's important or not.

But the way this root name server mapping works is that everybody just knows.

It's widely published.

You go to Google and you'll just find the answer.

It's also on all sorts of mailing lists.

So the first one is root identity.

The second issue is, how does the stub connect to any name server?

Would it just pick at random?

How does it find the name server that it can connect to?

And the answer to this is that this is actually running on your computer.

The stub result was running on your machine.

It's a library in your machine.

So, one approach, and the most common approach today is that you might obtain it when you obtain an IP address using a protocol like DCP where you turn on your computer and you get an IP address using some protocol.

That protocol, some gateway upstream might tell you which name server to use.

So if you have a computer at home, your Internet service provider would have something set up.

So they would tell you what name server to use.

So this can be done using a mechanism like DCP.

Or like in the old days, and these days if you want to do this kind of work, it's done manually.

For example, you could go into some registry somewhere or file like Kazaa.com. Then you just start stuff in.

The second issue that we need to spend some time about, which we are going to spend the next five or so minutes on is performance.

And the main point about performance is that if you look at that picture for how names are being resolved, if somebody at Berkeley wants to resolve X.CSALE.MIT.edu, there's a huge number of roundtrips that they have to do to all sorts of name servers all over the world, or at least all over the country before they can figure out what the IP address is from a machine.

And this is a little bit silly because often, they might want to connect to your webpage and get a 4 kB thing, which takes a couple of roundtrips to get.

And, in order to do that, they're spending a huge number of roundtrips latency, use latency in getting this right answer.

So the approach that we're going to take to solve the too-much-too-many roundtrips problem is an approach you've already seen.

We're going to use caching.

And, in order for this caching to work, there is actually something in this picture that you have to understand in a little bit more detail about two different kinds of name resolutions that are really going on.

The first kind of name resolution that's going on in DNS is the kind of resolution that the edu name server or the MIT.edu name server is doing in this picture, and that's called iterative resolution.

Iterative resolution says the following.

If I ask you to do some work for me to resolve a name, if you don't know the answer you just tell me, I don't know the answer, but you go here and find out the answer.

So you're getting these pointers referring you to the right place.

The second kind of resolution that's going on is the kind of resolution that this box is doing, this any name silver box.

And it's doing something called recursive resolution because what's going on here is the stub resolver's telling it, here's a name; resolve it for me.

And what he's doing is basically saying, OK, I'll resolve it for you and get you the final answer.

I'm not going to give you a referral back to other places, which means eventually once it figures out the answer, if there is one, it's going to know the answer even though it did not originate the query.

And the moment you have recursive resolution, it means that you can cache the answer because if somebody else comes and asks you the same query, you already have the answer cached.

And notice that this benefit doesn't accrue if you're doing purely iterative resolution.

If everybody was just sending referrals back to the stub resolver, the only caching you'd really primarily be gaining is for all of the requests that are common to this computer because nobody is actually cashing in getting any answers along the way.

Nobody's even getting any referrals along the way.

Only the node that's starting the name resolution is going to be getting any answers or any referrals at all.

So the secret to getting good DNS performance is for certain nodes, for certain name servers, to agree to do recursive resolution -- And an example of that is any name server in this picture.

So now if you have, a lot of the computers here with a lot of applications running on it, all of which use the same name server, and that name server is configured to recursively resolve names, then that name server benefits from being able to cache the answers to previous DNS lookups.

But notice that they can cache two kinds of answers, and one of which is much more important than the other.

The first kind of answer they can cache is the final answer that you get back that says X.CSALE.MIT.edu is at a particular IP address.

So the next time somebody goes to X.CSALE.MIT.edu, they have the answer right there.

But if you look at the statistics of DNS requests, there is some commonality in that everybody wants to go to www.CNN.com or Google.com or Yahoo.com, and so on.

But, there's a huge number of requests going to machines like yours and mine which aren't running anything interesting.

And you are really the only people interested in those machines.

So really what's going on, and why caching helps is that not only are the final answers being cached, these referrals are being cached.

So for example, [any name?] already knows the root, but now after the first request it knows the mapping for edu's name service.

And after the first one to MIT.edu, it can cache the mapping for MIT.edu's name server as well.

So, the next time somebody asks for anything.MIT.edu, this name server doesn't have to go all the way back to the root.

In fact, it doesn't even have to go all the way back to edu.

It just has to start with MIT.edu whose name service entry it already has in its cache.

And, that really is the reason why the DNS scales very, very well.

It's because it does caching.

But the real key is that it's caching referrals.

It's caching these name server entries associated with these labels.

It's getting some benefit from caching the final answers, but it's really getting a lot of benefit from caching referrals.

Now, of course, when you catch things, you have to worry about being stale because you certainly don't want to cache it forever.

If you cached it forever, then nobody could change anything.

So let's say you decide to change the mapping of www.MIT.edu's A record from one IP address to another.

How do you tell the whole world that you've changed it?

Well, there's two high-level strategies for this.

One is to somehow keep track of all the people who have cached it and invalidate entries.

And a few lectures from now, we'll look at ways in which that kind of approach might be made to work for different systems.

DNS deals with it in sort of a different way.

It doesn't worry about invalidation.

Instead, it sets expiration time on entries also called TTL is a time to live.

That's an abused and overloaded term in networks.

But it's really an expiration time.

It says that here's the answer to any of these questions, to a referral or to the final answer.

Here's the mapping.

And, it's valid for such and such a time, OK, like it could be anywhere from 15 seconds or 30 seconds to three hours or a day.

Usually it's a day or two.

It's usually never more than a couple of days because you reach the point of diminishing returns.

One request every two days is not a big deal.

So usually it's on the order of several seconds if you want the mapping to be fine-grained, or an hour, or a day.

And, after that, any access made after that to the same entry, whether it be a referral or whether it be a final answer, has to go back to the server that's authoritative that's responsible for that entry.

So, for example, you went for the first time doing the lookup of this name, and you got back that MIT.edu's name service was some name, and that it was valid for an hour.

Then the first request that happens for anything.MIT.edu from here after an hour has to go here.

I mean, assuming that edu is still a valid, hasn't yet expired.

This entry has to go here.

So, if you look at a time sequence of when you go, when you actually go back to the server responsible for a name, you can kind of divide up time into these chunks which are the expiration time intervals assuming they don't change.

They are set by the server.

And then, you might have accesses in between like this.

And the only accesses that go to the server responsible for the name are the first ones after every expiration.

So, this is the basic story for how you get reasonably good performance, and save a lot of roundtrips in DNS.

And the real reason for the scalability of the domain name system is a combination of administrative delegation.

So you don't have some human being involved in every name that's being added to the network.

It's distributed across different organizations.

And the fact that you have caching, in particular your caching these name server records, which means that you can save a lot of load on the root and on the edu or com name servers.

Originally, the designers of DNS, one thought that they had was that DNS would scale very well because the name space is extremely divided and hierarchical, which means that you were gaining a lot from the hierarchy.

And that's why it would scale.

But that's actually not true because more than 90% of the domain namespace is not hierarchical in any deep sense.

Everybody is something.com. Everybody of importance is something.com, or wants to be something.com. And so, most of the load gets here.

So, there's no real deep hierarchy that you're benefiting from here.

That's usually some flatname.com. But it's divided.

And it's delegated.

And that's the really nice thing about it.

So you are gaining much more from the fact that it's delegated.

You'd probably gain the same scalability is everything were just something.root, OK?

We didn't really need too much of this in terms of scalability, although it's very convenient to be able to do delegation.

But primarily it seems to be universities that take advantage of this kind of depth here.

Most companies tend not to pay much attention to depth.

But yet, the system scales because it is, in fact, administratively delegated.

So one word on replication.

The DNS name servers responsible for these names are replicated.

You can't set up a DNS name and have a name service associated with it.

A DNS name server record has to have at least two entries, and you have to be on two different networks.

And that's one way so that if one of them is down you can get to the other.

And unfortunately it turns out that that simple rule is often violated.

Probably the most celebrated incident here was Microsoft's update site, or one of the sites was down for more than 24 hours.

And in the end it turned out in all these cases to be a complicated set of reasons for why it failed.

Nothing fails for a simple reason.

But one of the root causes was that they had DNS name servers that were replicated but that happened to be behind the same Ethernet switch on the same subnet, which is not recommended.

But that's what they had.

So it's the kind of thing that you need to be careful about doing.

So I'm going to stop here.

And from Wednesday, we will talk about fault tolerance.

The recitation for tomorrow is a very short paper called Google and 9/11. But tomorrow you'll see a little bit about how Google works.