6.033 in the news

"Unfortunately, the system did not scale fast enough to accommodate the increased volume"

https://www.wbur.org/news/2021/02/19/prepmod-state-vaccine-appointment-site-crash-statement

"One such technology that the state could use is a cloud computing service that scales up server power to meet demand. Olivia Adams, a software developer who created MACovidVaccines.com while on maternity leave, uses one such service from Amazon. That's why her website didn't crash when the state's did, she thinks."

https://www.wbur.org/commonhealth/2021/02/18/hours-and-hours-of-frustration-mass-residents-emote-about-states-faulty-vaccine-websites

questions that arise:

Why didn't the state's website scale to meet demand? How do we test systems for that? How do we understand how far a system can scale, and in what dimensions?

Are companies like Amazon and Google the only places to get such highly-scaling services? If so, why? What implications does that have?

In particular, what are the consequences of public infrastructure relying on private companies?

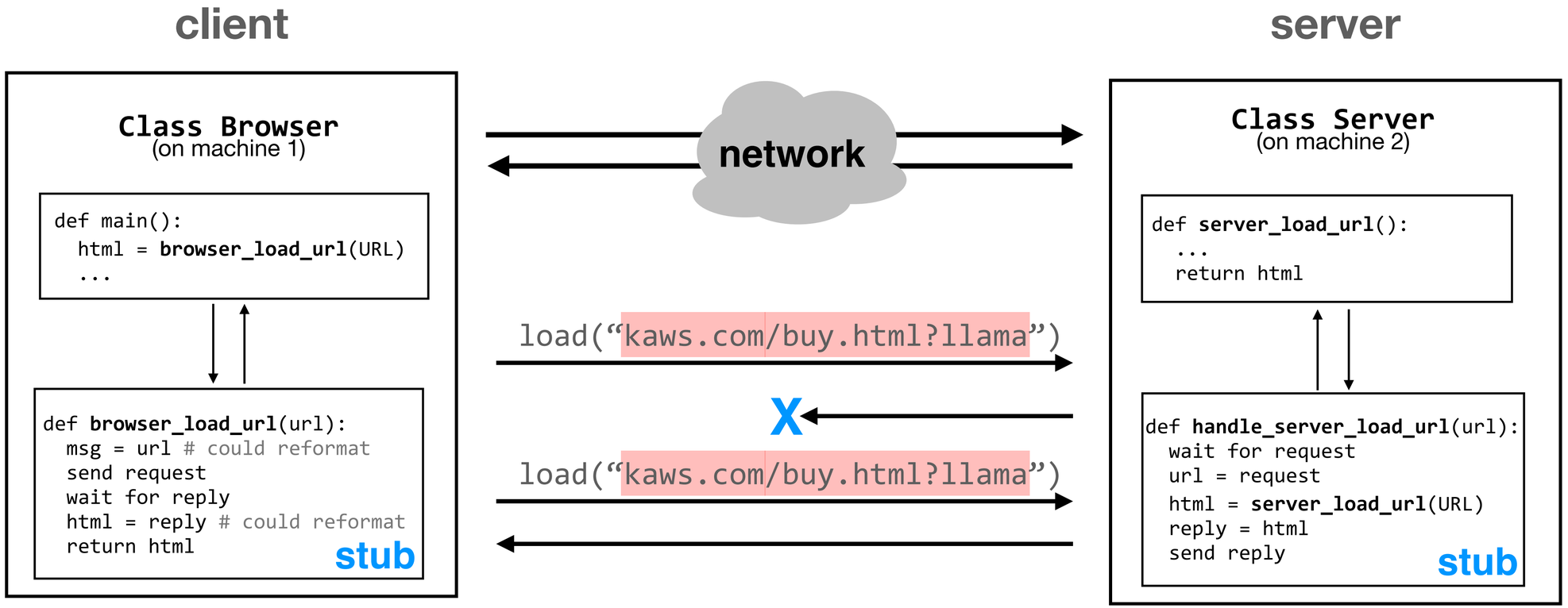
6.033 Spring 2021

Lecture #2: Naming

plus a case-study on DNS

Katrina LaCurts | lacurts@mit.edu | 6.033 2021

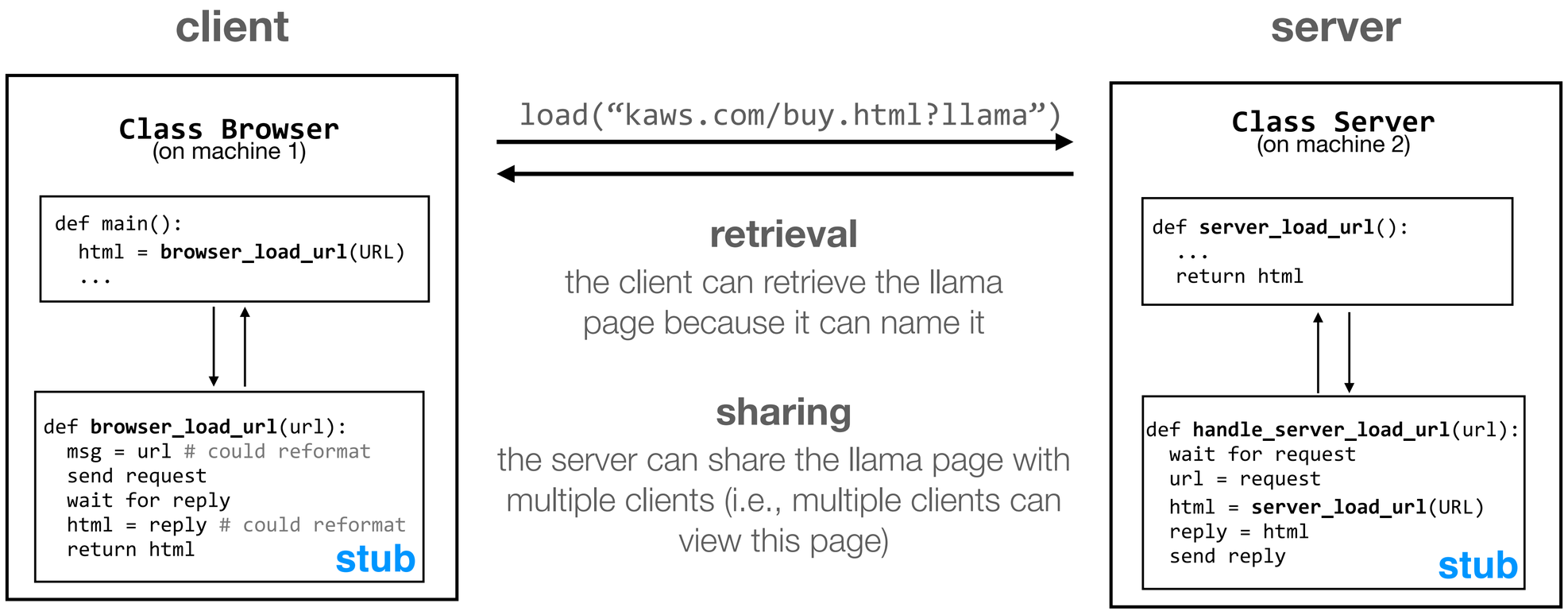
last time: enforced modularity via client/server



today: naming, which allows modules to interact

why use names?

they let us achieve modularity by providing communication and organization, as well as a number of other properties



retrieval

the client can retrieve the llama page because it can name it

sharing

the server can share the llama page with multiple clients (i.e., multiple clients can view this page)

user-friendly IDs

`kaws.com` is easier to remember than (say) `18.25.4.171`; the variable name "html" is easier to remember than a particular location in memory

addressing

some names also specify location information

why use names?

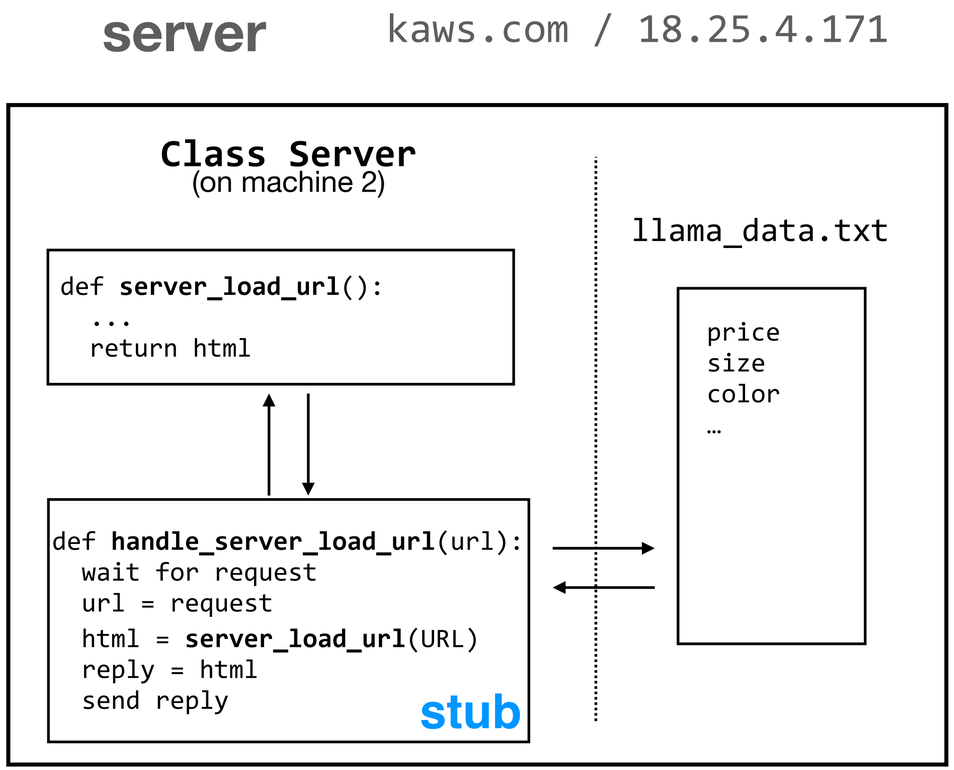
they let us achieve modularity by providing communication and organization, as well as a number of other properties

hiding

code on the server can access `llama_data.txt` without having to worry about how the file is laid out in memory

indirection

the server can change the memory layout of `llama_data.txt` without notifying the user



why use names? they let us achieve modularity by providing communication and organization, as well as a number of other properties

the design of a system's naming scheme(s) helps it achieve these properties

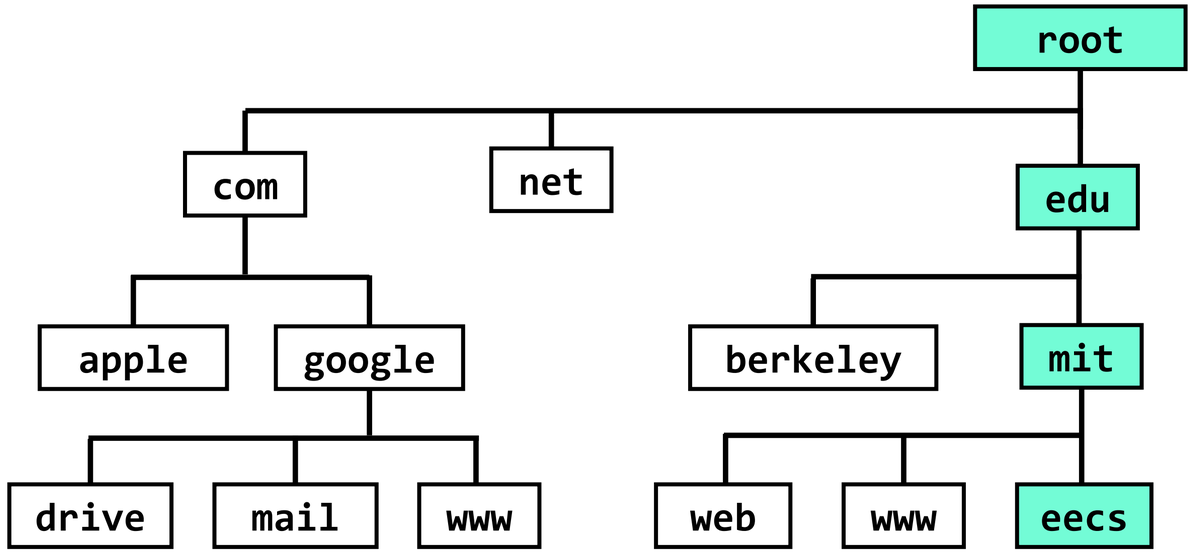
a naming scheme includes

1. the set of all possible **names**
2. the set of all possible **values**
3. a **look-up algorithm** to translate a name into a value (or a set of values, or "none")

naming case study:

the **domain name system (DNS)**, which maps **hostnames** (eecs.mit.edu) to **IP addresses** (18.62.1.6)

the **look-up algorithm** has to scale to the size of the Internet, while dealing with constant updates and issues of delegation



a partial view of the DNS hierarchy. each box represents a **zone**. name servers within a zone keep track of that zone's mappings

198.41.0.4

root nameserver

```
edu. 192.14.171.191  NS
com. 192.14.171.192  NS
net. 192.14.171.193  NS
```

192.14.171.191

edu. nameserver

```
mit.edu.      18.72.0.3      NS
berkeley.edu. 128.32.136.14  NS
```

18.72.0.3

mit. nameserver

```
eecs.mit.edu. 18.62.1.6   A
web.mit.edu.  18.9.2.69   A
www.mit.edu.  18.9.22.169 A
```

DNS client

e.g., your laptop

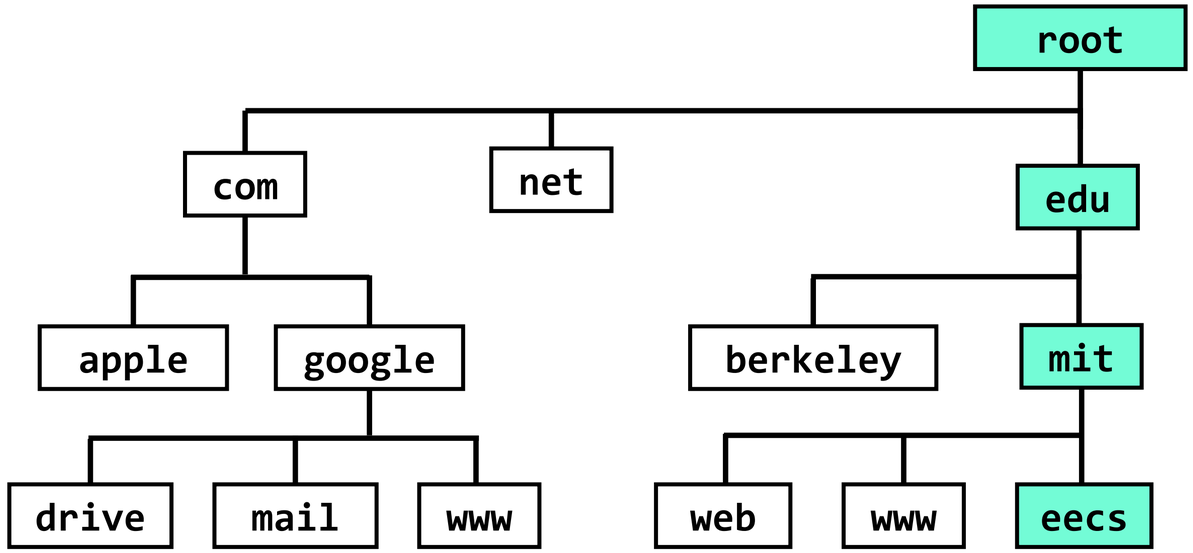
query sent to:

response:

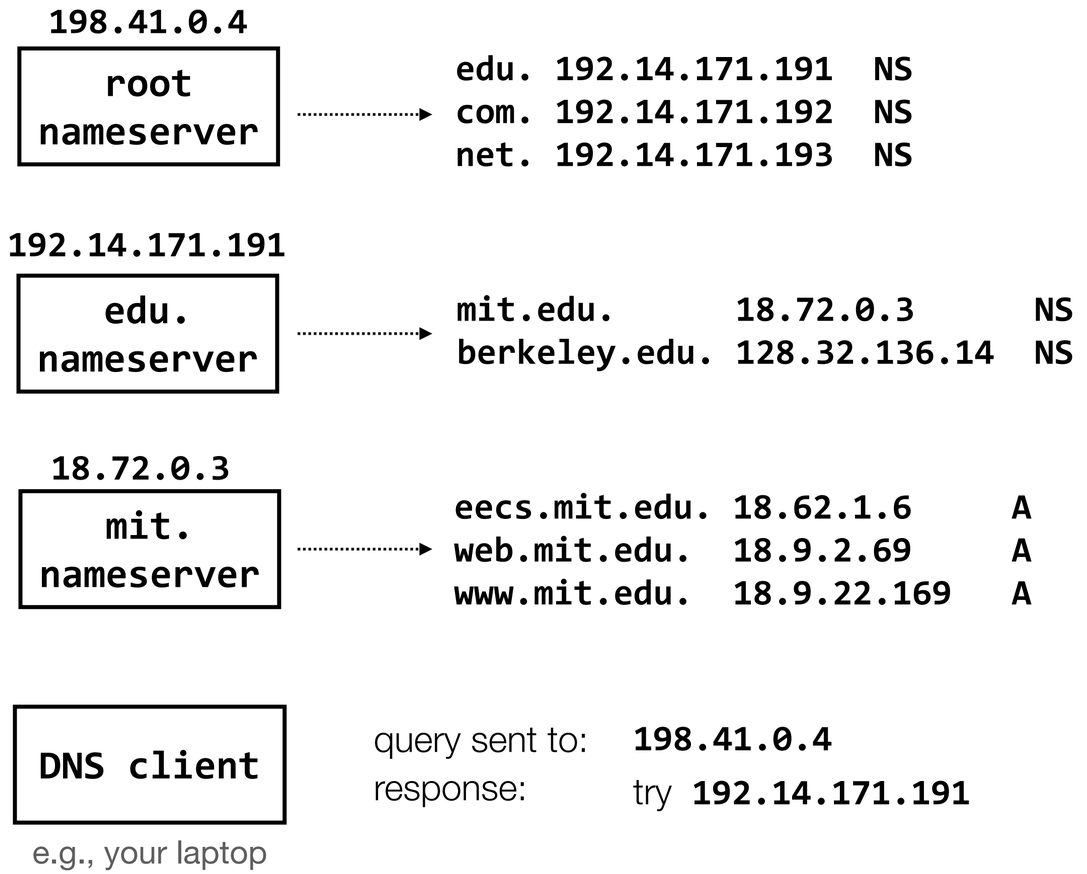
naming case study:

the **domain name system (DNS)**, which maps **hostnames** (eecs.mit.edu) to **IP addresses** (18.62.1.6)

the **look-up algorithm** has to scale to the size of the Internet, while dealing with constant updates and issues of delegation



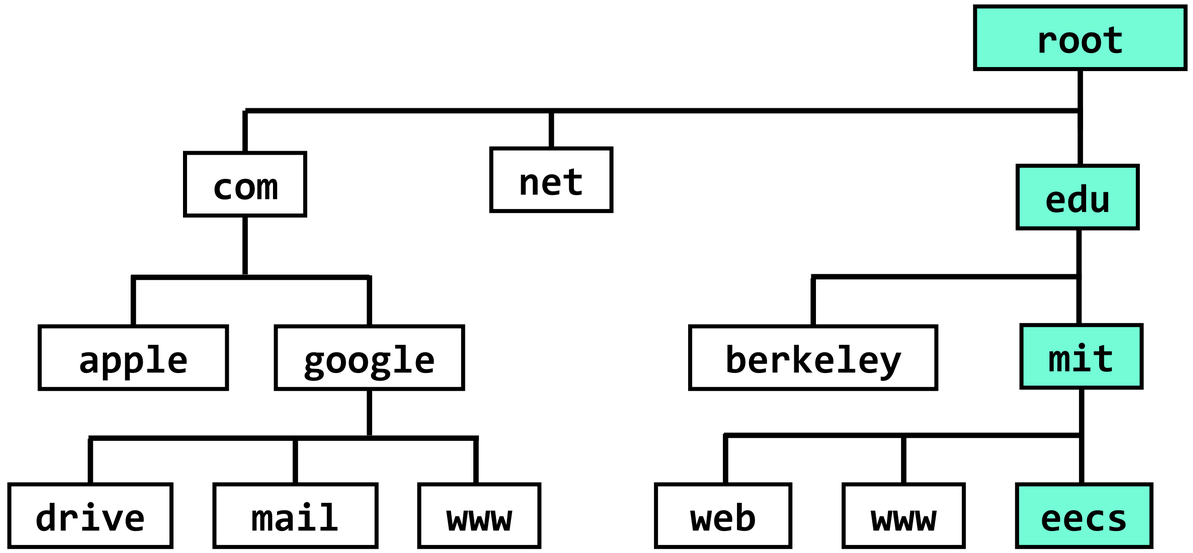
a partial view of the DNS hierarchy. each box represents a **zone**. name servers within a zone keep track of that zone's mappings



naming case study:

the **domain name system (DNS)**, which maps **hostnames** (eecs.mit.edu) to **IP addresses** (18.62.1.6)

the **look-up algorithm** has to scale to the size of the Internet, while dealing with constant updates and issues of delegation



a partial view of the DNS hierarchy. each box represents a **zone**. name servers within a zone keep track of that zone's mappings

198.41.0.4

root nameserver

```
edu. 192.14.171.191  NS
com. 192.14.171.192  NS
net. 192.14.171.193  NS
```

192.14.171.191

edu. nameserver

```
mit.edu.      18.72.0.3      NS
berkeley.edu. 128.32.136.14  NS
```

18.72.0.3

mit. nameserver

```
eecs.mit.edu. 18.62.1.6   A
web.mit.edu.  18.9.2.69   A
www.mit.edu.  18.9.22.169 A
```

DNS client

e.g., your laptop

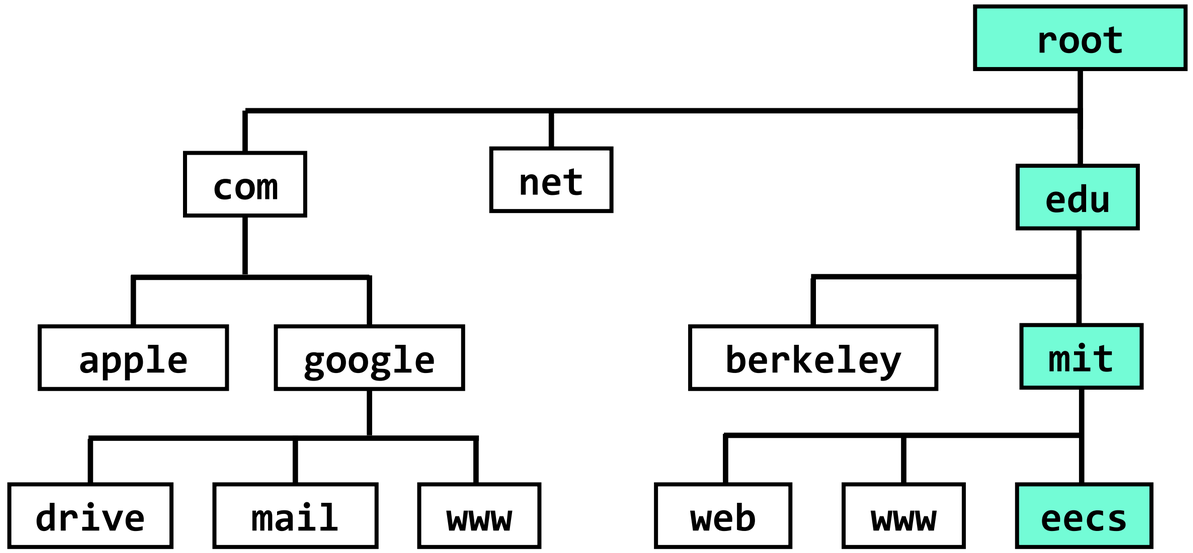
query sent to: **192.14.171.191**

response: try **18.72.0.3**

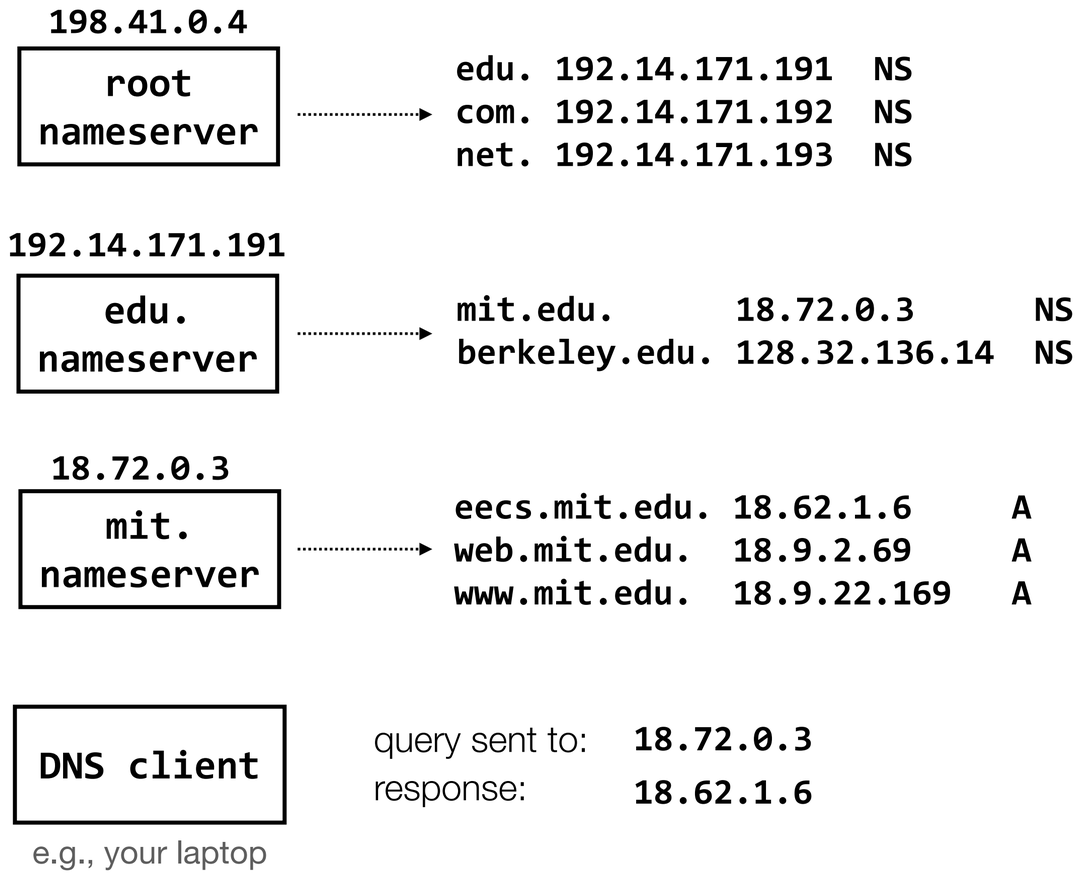
naming case study:

the **domain name system (DNS)**, which maps **hostnames** (eecs.mit.edu) to **IP addresses** (18.62.1.6)

the **look-up algorithm** has to scale to the size of the Internet, while dealing with constant updates and issues of delegation



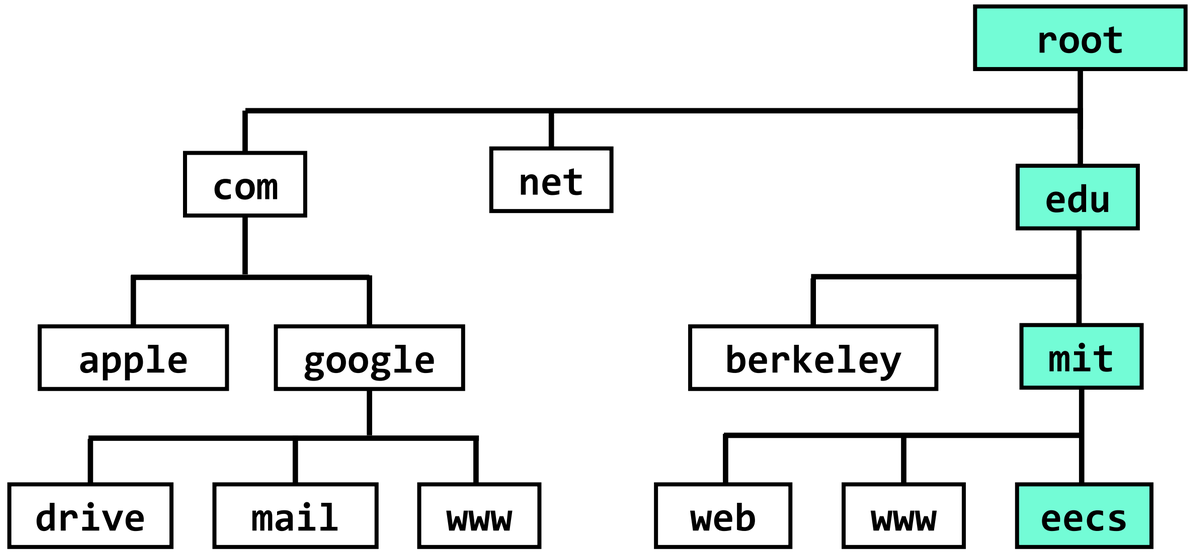
a partial view of the DNS hierarchy. each box represents a **zone**. name servers within a zone keep track of that zone's mappings



naming case study:

the **domain name system (DNS)**, which maps **hostnames** (eecs.mit.edu) to **IP addresses** (18.62.1.6)

the **look-up algorithm** has to scale to the size of the Internet, while dealing with constant updates and issues of delegation



a partial view of the DNS hierarchy. each box represents a **zone**. name servers within a zone keep track of that zone's mappings

performance issue: this is a *lot* of queries, especially to the root server

reliability issue: what happens when a nameserver fails or (**security issue**) is attacked?

control issue: who should own the root server?

modularity and **abstraction** mitigate complexity. a **client/server model** allow us to enforce modularity by putting modules on physically separate machines.

naming is what allows modules to interact, and can help us achieve other goals through properties such as indirection, user-friendliness, etc.

the **domain name system** is a great case-study in naming, and also illustrates principles such as **hierarchy**, **scalability**, **delegation**, and **decentralization**

and client/server models, and (tomorrow) caching, and (in May) security…

the example you saw in lecture was a fairly basic one; you will talk more about DNS's performance enhancements in recitation tomorrow, which change how some (many) DNS queries are resolved