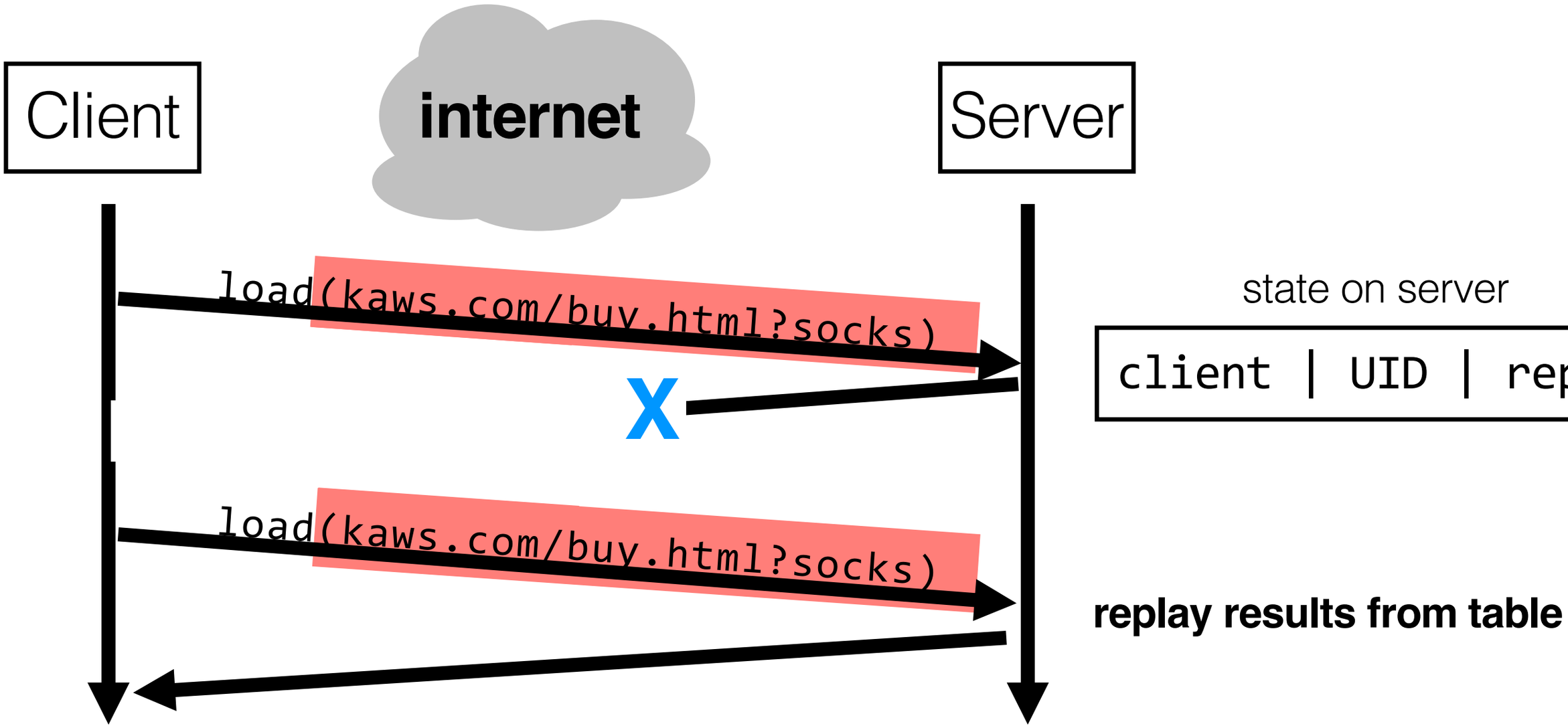
6.033 Spring 2018

Lecture #2

- Naming in systems
- Case study: DNS

Last Time: Enforced Modularity via Client/Server Model



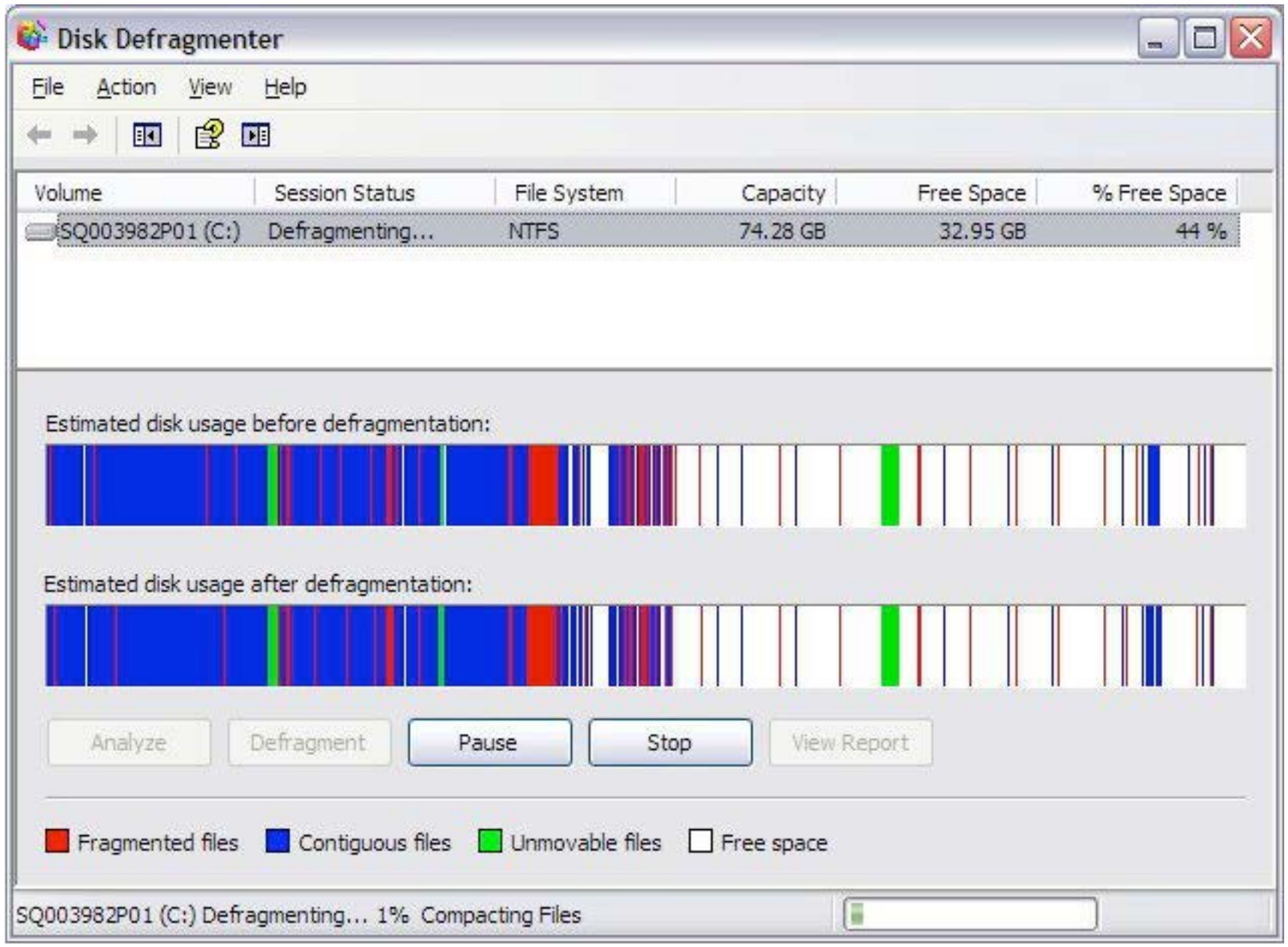
Today: Naming

allows modules to interact

Examples of Names

example.com	hostname
userA@example.com	email
userA	username
R0	x86 register name
main	function name
WebBrowser	class name
/mit/6.033/schedule.shtml	path name
http://example.com/about	URL
617-555-1234	phone number
128.30.2.121	IP Address

why use names?



why use names?

Naming Schemes

1. Set of all possible **names**

2. Set of all possible **values**

3. **Look-up algorithm** to translate a name into a value (or set of values, or “none”)

Domain Name System

1. **names:** hostnames (`web.mit.edu`)

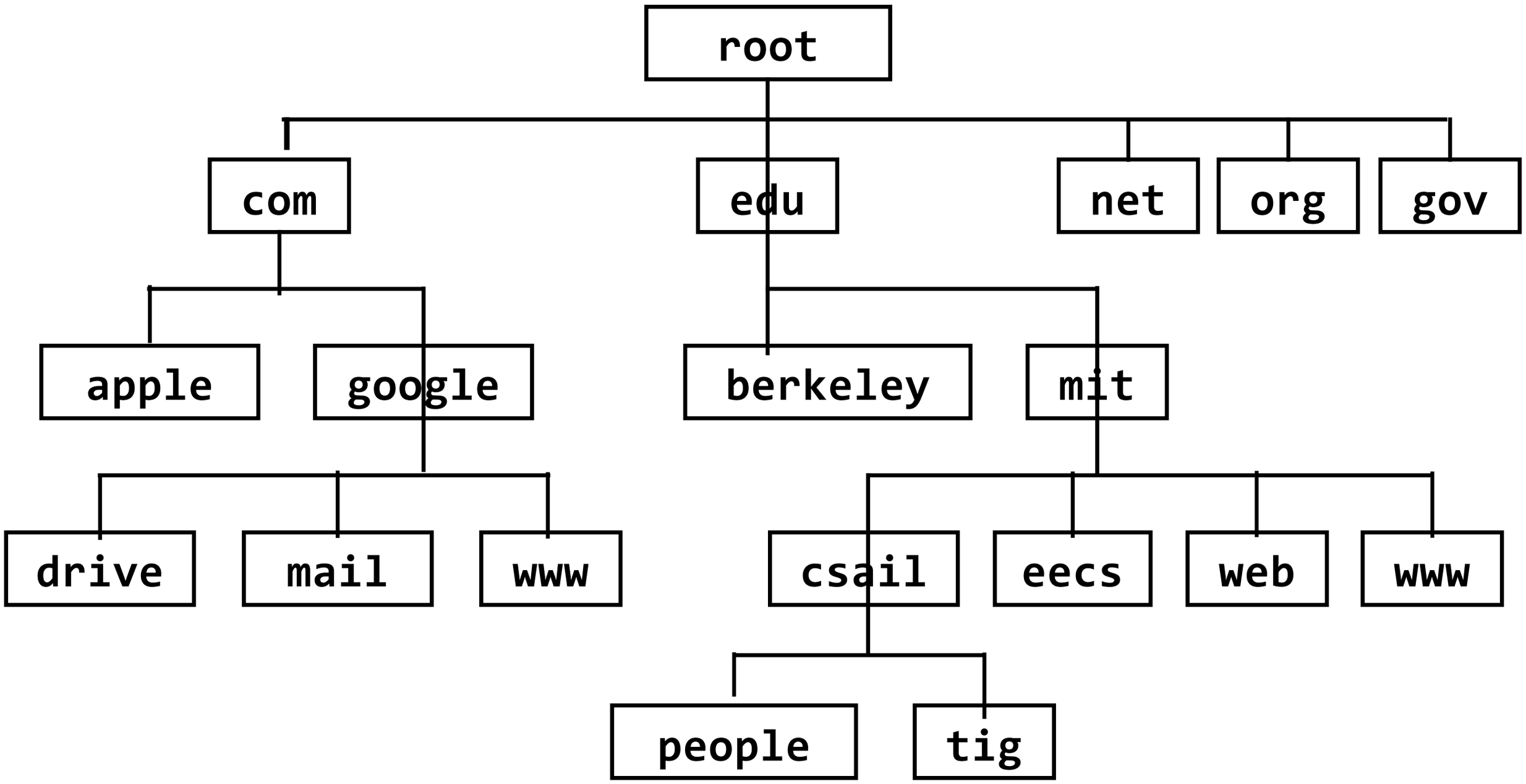
2. **values:** IP addresses (`18.9.22.69`)

    IP addresses are imbued with location information: routers can send packets to an IP address, but not to a hostname

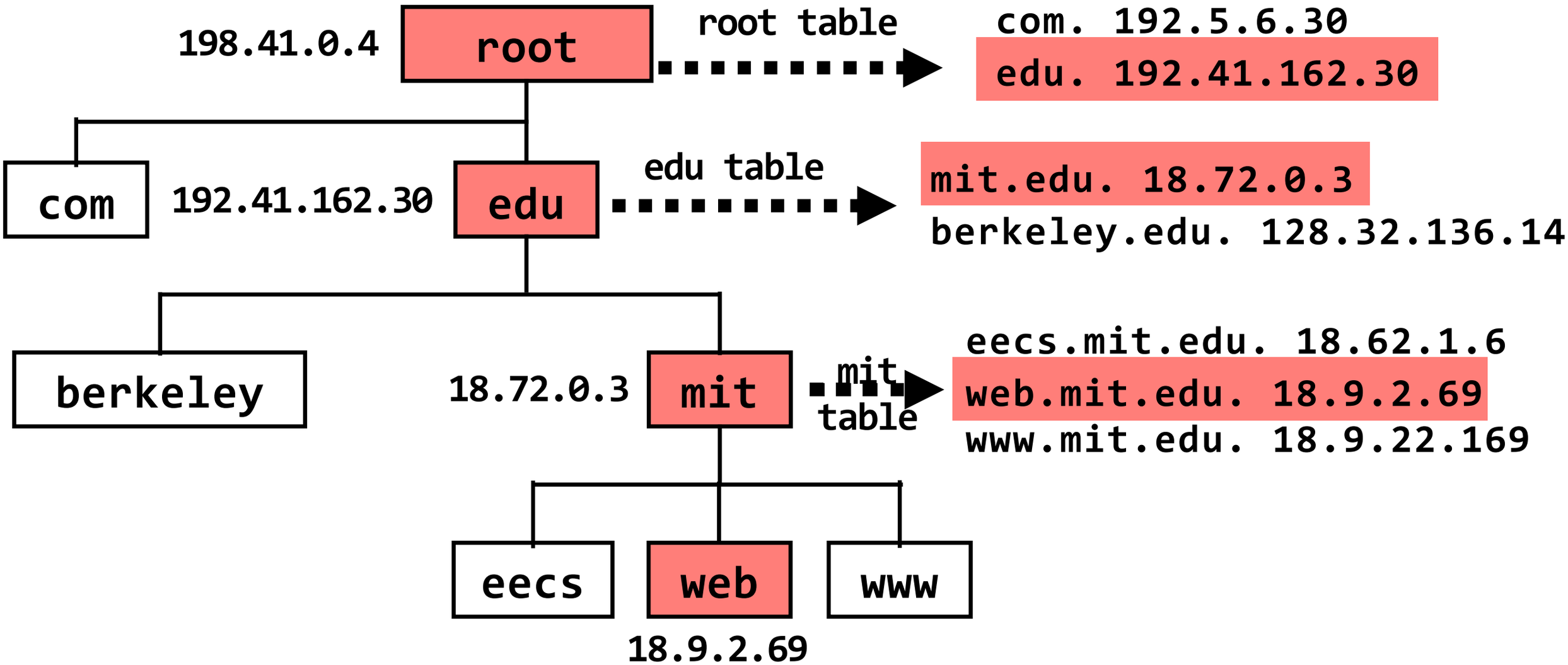
3. **look-up algorithm:** resolves a hostname to an IP address so that your machine knows where to send data

DNS Hierarchy

(a partial view)



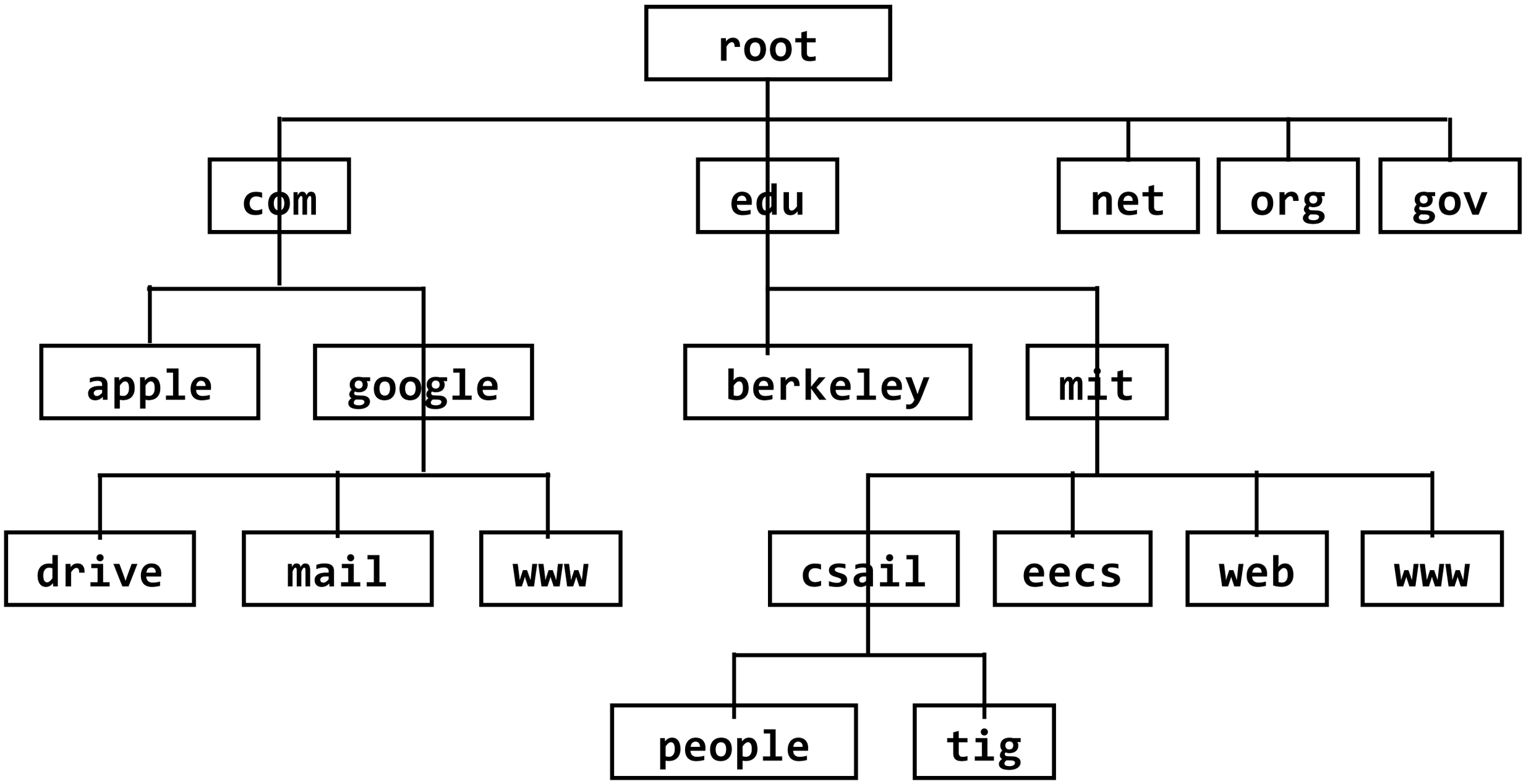
DNS Look-up for web.mit.edu



query to: `192.41.162.30` **result:** `web.mit.edu. 18.9.2.69`

DNS Hierarchy

(a partial view)

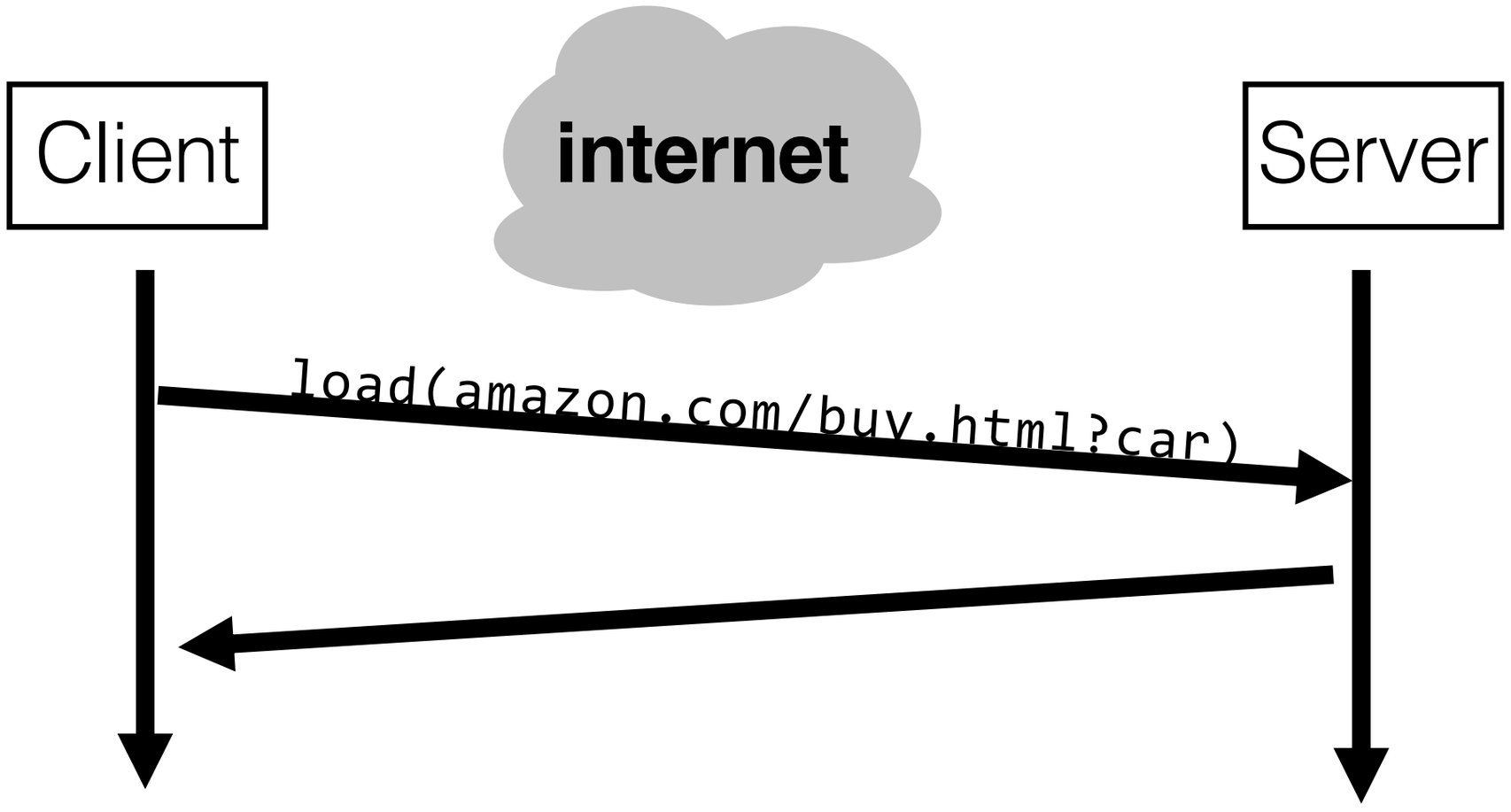


- **Modularity** (and abstraction) limit complexity. One way to enforce modularity is to use a client/server design

- **Naming** is what allows modules — for example, a client and a server — to communicate; it is pervasive across systems

- **DNS** maps hostnames to IP addresses. It is also a good example of **hierarchy**.

Lingering Problem



what if we don't want our modules to be on entirely separate machines? how can we **enforce modularity on a single machine**?

6.033 Computer System Engineering
Spring 2018