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
userA@example.com
userA
R0
main
WebBrowser
/mit/6.033/schedule.shtml
http://example.com/about
617-555-1234
128.30.2.121

hostname
e-mail
username
x86 register name
function name
class name
path name
URL
phone number
IP Address
why use names?
Disk Defragmentation Interface

Volume: SQ003982P01 (C:)
Session Status: Defragmenting...
File System: NTFS
Capacity: 74.28 GB
Free Space: 32.95 GB
% Free Space: 44%

Estimated disk usage before defragmentation:

Estimated disk usage after defragmentation:

Options: Analyze, Defragment, Pause, Stop, View Report

© source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see https://ocw.mit.edu/help/faq-fair-use.
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?