6.033 Spring 2016
Lecture #1

- Complexity
- Modularity and abstraction
- Enforced modularity via client/server models
http://mit.edu/6.033

Schedule

<table>
<thead>
<tr>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reg day</strong></td>
<td><strong>feb 2</strong></td>
<td><strong>feb 3</strong></td>
<td><strong>feb 4</strong></td>
<td><strong>feb 5</strong></td>
</tr>
<tr>
<td><strong>REC 1: Worse is Better</strong>&lt;br&gt;Assigned: Hands-on DNS&lt;br&gt;First day of classes</td>
<td><strong>REC 1: Coping with Complexity: Enforced Modularity and Client/server Organization</strong>&lt;br&gt;Reading: Book sections 1.1-1.5, and 4.1-4.3</td>
<td><strong>REC 2: Therac-25</strong>&lt;br&gt;Assigned: Paper critique #1</td>
<td><strong>TUT 1: Introduction to system critiques</strong>&lt;br&gt;Assigned: Paper critique #1</td>
<td><strong>TUT 2: How to read a paper</strong></td>
</tr>
<tr>
<td><strong>feb 8</strong></td>
<td><strong>feb 9</strong></td>
<td><strong>feb 10</strong></td>
<td><strong>feb 11</strong></td>
<td><strong>feb 12</strong></td>
</tr>
<tr>
<td><strong>LEC 2: Naming</strong>&lt;br&gt;Reading: Book sections 2.2, and 3.1</td>
<td><strong>REC 3: DNS</strong>&lt;br&gt;DUE: Hands-on DNS&lt;br&gt;Assigned: Hands-on UNIX</td>
<td><strong>REC 3: Virtual memory</strong>&lt;br&gt;Reading: Book sections 5.1, 5.3, and 5.4</td>
<td><strong>REC 4: UNIX</strong>&lt;br&gt;Reading: Book sections 5.1, 5.3, and 5.4</td>
<td><strong>TUT 2: How to read a paper</strong></td>
</tr>
</tbody>
</table>

Fill out form for recitation assignments <br>link on home page
what is a system?
a set of interconnected components that has an expected behavior observed at the interface with its environment
6.033 Approach to Systems

**lectures:** big ideas + examples  
Katrina LaCurts

**recitations:** read papers describing successful systems  
Michael Carbin, Asaf Cohen, Mark Day, Karen Sollins, Peter Szolovits, Michael Yee, Matei Zaharia

**hands-ons:** play with successful systems

**design project + critiques:** analyze, design, write  
*Writing staff:* Jared Berezin, Amy Carleton, Mary Caulfield, Amelia Herb, Nora Jackson, Janis Melvold, Juergen Schoenstein, Jessie Stickgold-Sarah, Linda Sutliff, Rebecca Thorndike-Breeze, Michael Trice  
*TAs:* Jodie Chen, Jacqui De Sa, Sumit Gogia, Anubhav Jain, Rebecca Kekelishvili, Steven Keyes, Pratheek Nagaraj, CK Ong, Cathie Yun

**exams:** reason about system design
what is a system?

a set of interconnected components that has an expected behavior observed at the interface with its environment

what makes building systems difficult?

complexity
Today’s Systems are Incredibly Complex

source: http://www.informationisbeautiful.net/visualizations/million-lines-of-code/
Emergent Properties
(ethernet example)

collision not detected!

for collision-detection to work, endpoints must send for at least twice the latency of the link
http://www.caida.org/research/topology/as_core_network/2014/
how can we mitigate complexity?
how do we enforce modularity?
Stub Clients and RPCs

Class `webBrowser`
(on machine 1)

```python
def main():
    html = browser_load_url(URL)
...
```

```python
def browser_load_url(url):
    msg = url # could reformat
    send request
    wait for reply
    html = reply # could reformat
    return html
```

Class `webServer`
(on machine 2)

```python
def server_load_url():
    ...
    return html
```

```python
def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply
```
Challenges with RPCs

problem: just bought the same thing twice
Challenges with RPCs

Client | internet | Server

load("buy.html")

load("buy.html")

X

client | UID | reply

state on server

replay results from table instead of reprocessing order

problem: server can still fail
• **Complexity**
  Comes from many sources, limits what we can build, causes unforeseen issues; can be mitigated with **modularity** and **abstraction**

• **Enforced modularity**
  One way to enforce modularity is with a **client/server model**, where the two modules reside on different machines and communicate with RPCs; network/server failures are still an issue

**next lecture:** naming, which allows modules to communicate

**subsequent lectures:** operating systems, which provide modularity on a single machine