• Complexity
• Modularity and abstraction
• Enforced modularity via client/server models
what is a system?

a set of interconnected components that has an expected behavior observed at the interface with its environment
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>feb 6</td>
<td></td>
<td></td>
<td>feb 9</td>
<td>feb 10</td>
</tr>
<tr>
<td>Reg day</td>
<td>feb 7</td>
<td>feb 8</td>
<td>REC 2: Therac-25</td>
<td>TUT 1: What/How/Why Framework</td>
</tr>
<tr>
<td></td>
<td>REC 1: Worse is Better</td>
<td>LEC 1: Coping with Complexity: Enforced</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Assigned: Hands-on DNS</td>
<td>Modularity and Client/server Organization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>First day of classes</td>
<td>Reading: Book sections 1.1-1.5, and 4.1-4.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Assigned: Paper critique #1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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/
complexity limits what we can build and causes a number of unforeseen issues
(Please note that while this map shows the host population of the network according to the best information obtainable, no claim can be made for its accuracy)

Names shown are IMP names, not (necessarily) host names.
complexity limits what we can build and causes a number of unforeseen issues

by limiting what we can build, complexity makes it difficult to achieve other properties, such as scalability, fault-tolerance, security, performance, etc.
how do we mitigate complexity?

with design principles such as modularity and abstraction
how do we enforce modularity?

one way is to use the client/server model
Stub Clients and RPCs

Class `webBrowser` (on machine 1)

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

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

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

Challenges with RPCs

Client → Server

Server → Client
Challenges with RPCs

Client \hspace{1cm} internet \hspace{1cm} Server
Challenges with RPCs

Client ➔ internet ➔ Server

load("view.html?item")

X

load("view.html?item")
Challenges with RPCs

Client  internet  Server

load("buy.html?item&ccNo=xxx")

problem: just bought the same thing twice
Challenges with RPCs

problem: server can still fail

state on server

replay results from table instead of reprocessing order

load("buy.html?UID")

load("buy.html?UID")

Client ←→ internet ←→ Server

client | UID | reply

6.033 | spring 2017 | lacurts@mit.edu
• **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