6.033 Spring 2019
Lecture #1

• 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

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

def server_load_url():
    ...
    return html
Stub Clients and RPCs

**Class Browser**  
(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 Server**  
(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
```

stub  
stub

6.033 | spring 2019 | lacurts@mit.edu
Challenges with RPCs

Client \[\text{internet} \rightarrow \] Server

load("view.html?item")

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

Client ➔ internet ➔ Server

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

\[ \times \]

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

**problem:** just bought the same thing twice
Challenges with RPCs

Client | internet | Server

Load("buy.html?UID")

Problem: server can still fail

Replay results from table instead of reprocessing order

State on server

client | UID | reply

6.033 | spring 2019 | lacurts@mit.edu
What else might we want?

scalability

internet
What else might we want?

scalability

fault-tolerance/reliability

internet
What else might we want?

scalability

internet

fault-tolerance/reliability

security
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 4</td>
<td>Feb 5 REC 1: <em>Worse is Better</em></td>
<td>Feb 6 LEC 1: Coping with Complexity: Enforced Modularity via Client/server Organization</td>
<td>Feb 7 REC 2: <em>We Did Nothing Wrong</em></td>
<td>Feb 8 TUT 1: <em>Intro to 6.033 Communication</em></td>
</tr>
<tr>
<td>Reg day</td>
<td><em>First day of classes</em></td>
<td><em>Reading: Book sections 1.1-1.5, and 4.1-4.3</em></td>
<td></td>
<td><em>Assigned: System critique #1</em></td>
</tr>
</tbody>
</table>

Class announcements happen via Piazza
• **Complexity** limits what we can build, but can be mitigated with **modularity** and **abstraction**

• 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

**coming up:** operating systems, which enforce modularity on a single machine