6.033 Spring 2018
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)
...
```

```python
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
```

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

X

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

problem: just bought the same thing twice
Challenges with RPCs

Client  internet  Server

load("buy.html?UID")

X

load("buy.html?UID")

state on server

client | UID | reply

replay results from table instead of reprocessing order

problem: server can still fail
What else might we want?

scalability

internet
What else might we want?

scalability

internet

fault-tolerance/reliability
What else might we want?

- scalability
- fault-tolerance/reliability
- security

internet
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 5</td>
<td>REC 1: Worse is Better</td>
<td>Feb 7: Coping with Complexity: Enforced Modularity via Client/server Organization</td>
<td>Feb 8: We Did Nothing Wrong</td>
<td>Feb 9: TUT 1: Intro to 6.033 Communication</td>
</tr>
<tr>
<td>Reg day</td>
<td>First day of classes</td>
<td>Reading: Book sections 1.1-1.5, and 4.1-4.3</td>
<td></td>
<td>Assigned: System critique #1</td>
</tr>
</tbody>
</table>

Class announcements happen via Piazza

Sign up online for a permanent recitation section
• **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