6.1800 Spring 2024

Lecture #1: Complexity, modularity, abstraction
plus an intro to client/server models
we care about you as people more than we care about any deadline

if you need help, ask for it. we need to balance the needs of a large group of students and the needs of the staff, but we will work with you to help as much as we can. in particular, as long as you reach out to your TA ahead of time, we will give you a 24-hour extension on any assignment, no questions asked.
what is a system?
what is a system?

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

why do we care?
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

why do we care?
complexity limits what we can build
why do we care?

complexity limits what we can build
why do we care?

complexity limits what we can build

how do we mitigate complexity?
why do we care?
complexity limits what we can build

how do we mitigate complexity?
with design principles such as modularity and abstraction
why do we care?
complexity limits what we can build

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

how do we enforce modularity?
why do we care?

complexity limits what we can build

how do we mitigate complexity?

with design principles such as modularity and abstraction

how do we enforce modularity?

one way is to use a client/server model
why do we care?
complexity limits what we can build

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

how do we enforce modularity?
one way is to use a client/server model
why do we care?

complexity limits what we can build

how do we mitigate complexity?

with design principles such as **modularity** and **abstraction**

how do we enforce modularity?

*one* way is to use a **client/server model**

---

<table>
<thead>
<tr>
<th>Class Browser (on machine 1)</th>
<th>Class Server (on machine 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>def main():</code></td>
<td><code>def server_load_url():</code></td>
</tr>
<tr>
<td><code>html = browser_load_url(URL)</code></td>
<td><code>...</code></td>
</tr>
<tr>
<td><code>...</code></td>
<td><code>return html</code></td>
</tr>
</tbody>
</table>
why do we care?
complexity limits what we can build

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

how do we enforce modularity?
one way is to use a client/server model
why do we care?
complexity limits what we can build

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

how do we enforce modularity?
one way is to use a client/server model

the browser is the client in this example

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
```

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

stub

Class Server
(on machine 2)

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

stub

the browser is the client in this example
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

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
```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

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
```

Class Browser
(on machine 1)

Class Server
(on machine 2)
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

stub

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
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

stub

---

def server_load_url():
    ...
    return html

stub

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

stub
```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
```

---

```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
```

---

**Class Browser**
(on machine 1)

**Class Server**
(on machine 2)
```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

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
```

**Class Browser**
(on machine 1)

**Class Server**
(on machine 2)
```
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
```

```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
```
```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

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
```

Class Browser (on machine 1)

Class Server (on machine 2)
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

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

load(“view.html?item”)
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

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

load("view.html?item")
X
load("view.html?item")
```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

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
```

```
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

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
```
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

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

stub

stub
```
class Class Browser:
    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 Class Server:
    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

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

stub

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

stub

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
```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
```

```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
```

Diagram:
- **Client**:
  - Class Browser
    - (on machine 1)
    - `def main()`: load “buy.html?item&ccNo=XX”
    - `def browser_load_url(url)`: stub

- **Server**:
  - Class Server
    - (on machine 2)
  - `def server_load_url()`: stub
  - `def handle_server_load_url(url)`: stub

- **Network**
  - Load from client
  - Load from server
```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

def server_load_url():
    ...
    return html

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

---

**Class Browser**
(on machine 1)

**Class Server**
(on machine 2)
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
```

**Problem:** We just bought two copies of *item*
**Problem:** We just bought two copies of *item*

There are ways to deal with this issue — for example, giving each request a unique ID, and keeping track of those IDs on the server.
client

**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
```

server

**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
```

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

**Problem:** we just bought two copies of item

there are ways to deal with this issue — for example, giving each request a unique ID, and keeping track of those IDs on the server — but then new problems arise: for example, what happens if the server crashes in the middle of handling a request?
client  network  server
network
**scalability:** how does our system behave as we increase the number of machines, users, requests, data, etc.?
**scalability**: how does our system behave as we increase the number of machines, users, requests, data, etc.?
**scalability:** how does our system behave as we increase the number of machines, users, requests, data, etc.?

**fault-tolerance/reliability:** how does our system deal with failures (☠️)? machines crashing, network links breaking, etc.
**scalability**: how does our system behave as we increase the number of machines, users, requests, data, etc.?

**fault-tolerance/reliability**: how does our system deal with failures (☠️) or (😈)? Machines crashing, network links breaking, etc.
**scalability**: how does our system behave as we increase the number of machines, users, requests, data, etc.?

**fault-tolerance/reliability**: how does our system deal with failures (☠)? machines crashing, network links breaking, etc.

**security**: how does our system cope in the face of targeted attacks (😈)?
**Scalability:** how does our system behave as we increase the number of machines, users, requests, data, etc.?

**Fault-tolerance/reliability:** how does our system deal with failures (☠️)? Machines crashing, network links breaking, etc.

**Security:** how does our system cope in the face of targeted attacks (😈)?
**scalability:** how does our system behave as we increase the number of machines, users, requests, data, etc.?

**fault-tolerance/reliability:** how does our system deal with failures (☠)? machines crashing, network links breaking, etc.

**security:** how does our system cope in the face of targeted attacks (😈)?
**scalability:** how does our system behave as we increase the number of machines, users, requests, data, etc.?

**fault-tolerance/reliability:** how does our system deal with failures (☠️)? machines crashing, network links breaking, etc.

**security:** how does our system cope in the face of targeted attacks (😈)?

**performance:** how do we define our performance requirements, and know if our system is meeting them? what do we do if performance is subpar (🐢)?

network

Katrina LaCurts | lacurts@mit.edu | 6.1800 2024
**scalability**: how does our system behave as we increase the number of machines, users, requests, data, etc.?

**fault-tolerance/reliability**: how does our system deal with failures (☠️)? machines crashing, network links breaking, etc.

**security**: how does our system cope in the face of targeted attacks (😈)?

**performance**: how do we define our performance requirements, and know if our system is meeting them? what do we do if performance is subpar (🐢)?

**who is impacted** by our design and implementation choices? **who makes those choices?**
http://mit.edu/6.1800

has all of the class material, due dates, deadlines, etc.

<table>
<thead>
<tr>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
</tr>
</thead>
</table>
| **First day of classes** | Feb 5  
LEC 1: Modularity, Abstraction, and the Impact of Systems | Feb 6  
REC 1: We Did Nothing Wrong | Feb 7  
LEC 2: Naming | Feb 8  
REC 2: DNS  
Assignment Available: Hands-on DNS | Feb 9  
TUT 1: Intro to 6.1800 Communication |
http://mit.edu/6.1800

has all of the class material, due dates, deadlines, etc.

Canvas

for submitting assignments and seeing your grades, and the occasional class-wide (or section-wide) announcement. everything on Canvas will be linked from the class website.

we’ve already sent out one announcement about a scheduling form — please fill it out today if you haven’t already!
http://mit.edu/6.1800

has all of the class material, due dates, deadlines, etc.

Canvas

for submitting assignments and seeing your grades, and the occasional class-wide (or section-wide) announcement. Everything on Canvas will be linked from the class website.

Piazza

for questions that are relevant to the entire class. Important information from Piazza will also end up on the website (e.g., some of your assignments will have FAQs)
http://mit.edu/6.1800

has all of the class material, due dates, deadlines, etc.

Canvas

for submitting assignments and seeing your grades, and the occasional class-wide (or section-wide) announcement. everything on Canvas will be linked from the class website

Piazza

for questions that are relevant to the entire class. important information from Piazza will also end up on the website (e.g., some of your assignments will have FAQs)

we care about you as people more than we care about any deadline

if you need help, ask for it. we need to balance the needs of a large group of students and the needs of the staff, but we will work with you to help as much as we can. in particular, as long as you reach out to your TA ahead of time, we will give you a 24-hour extension on any assignment, no questions asked.
complexity limits what we can build, but can be mitigated with design principles such as modularity and abstraction
complexity limits what we can build, but can be mitigated with design principles such as modularity and abstraction

you will see these principles applied over and over in this class

a student once told me that I say “modularity” in almost every lecture, which seems correct
complexity limits what we can build, but can be mitigated with design principles such as **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.

You will see these principles applied over and over in this class.

A student once told me that I say “modularity” in almost every lecture, which seems correct.
complexity limits what we can build, but can be mitigated with design principles such as 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.

You will see these principles applied over and over in this class.

A student once told me that I say “modularity” in almost every lecture, which seems correct.

Next lecture: naming, which allows modules to communicate.
complexity limits what we can build, but can be mitigated with design principles such as **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

you will see these principles applied over and over in this class

a student once told me that I say “modularity” in almost every lecture, which seems correct

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

**after that**: operating systems, which enforce modularity on a single machine