6.1800 Spring 2024

Lecture #1: Complexity, modularity, abstraction

plus an intro to client/server models

Katrina LaCurts | lacurts@mit.edu | 6.1800 2024

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?

“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?

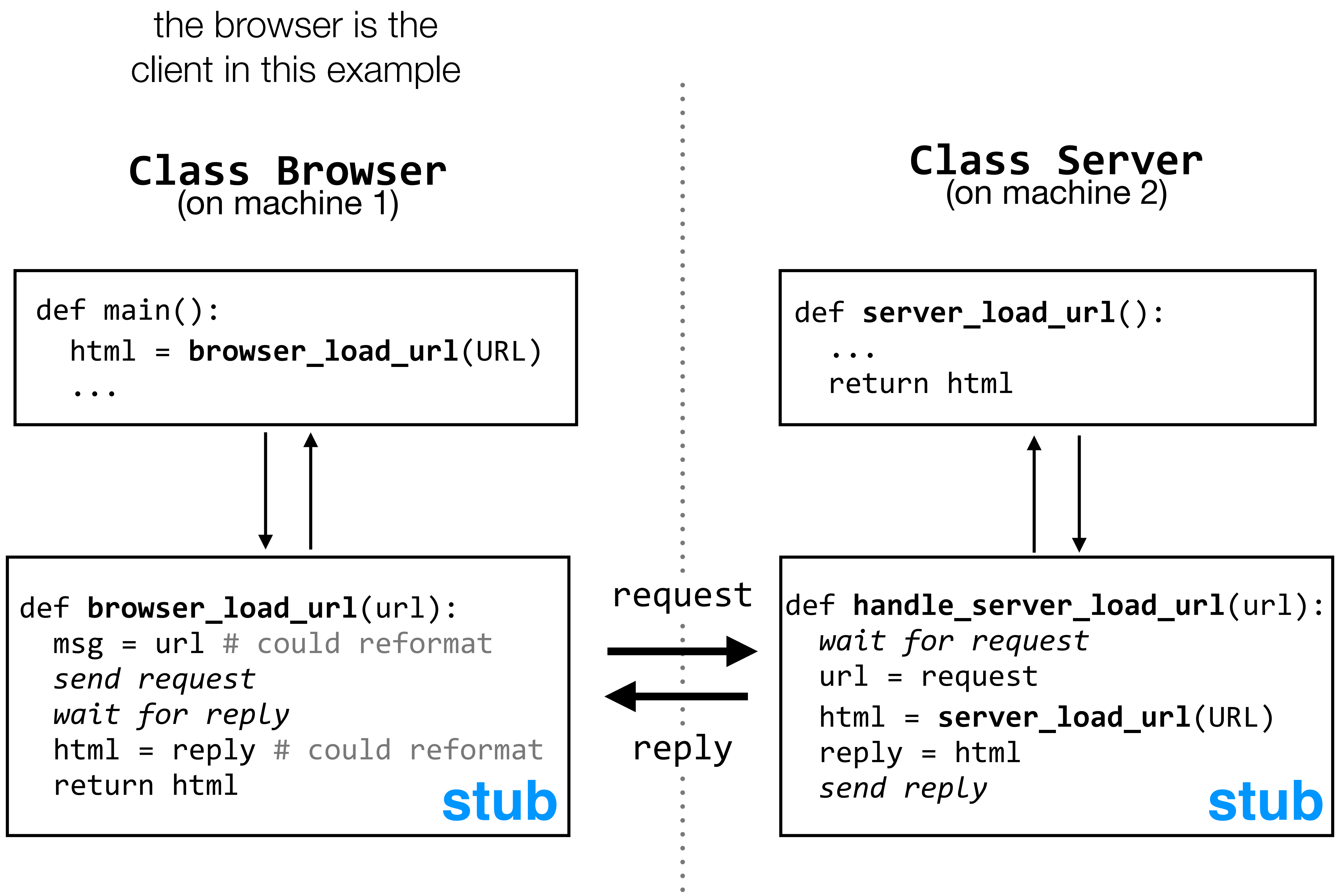
complexity **limits what we can build**

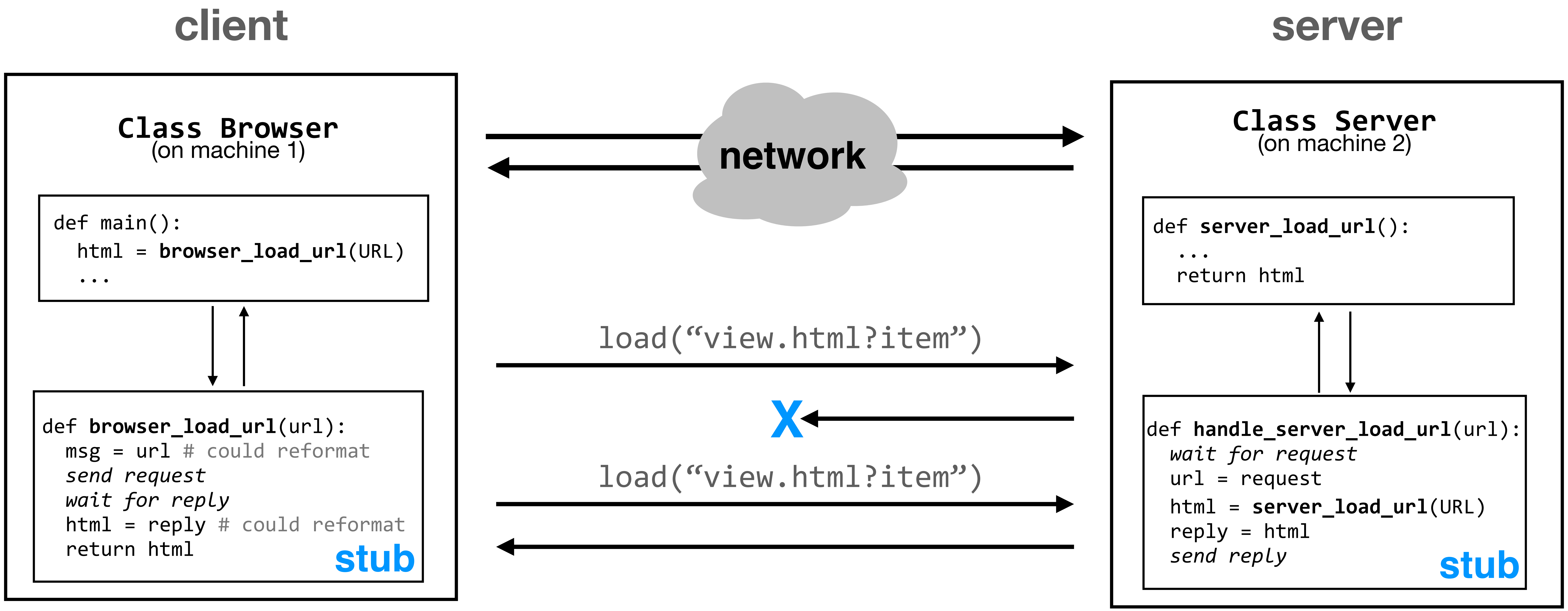
how do we mitigate complexity?

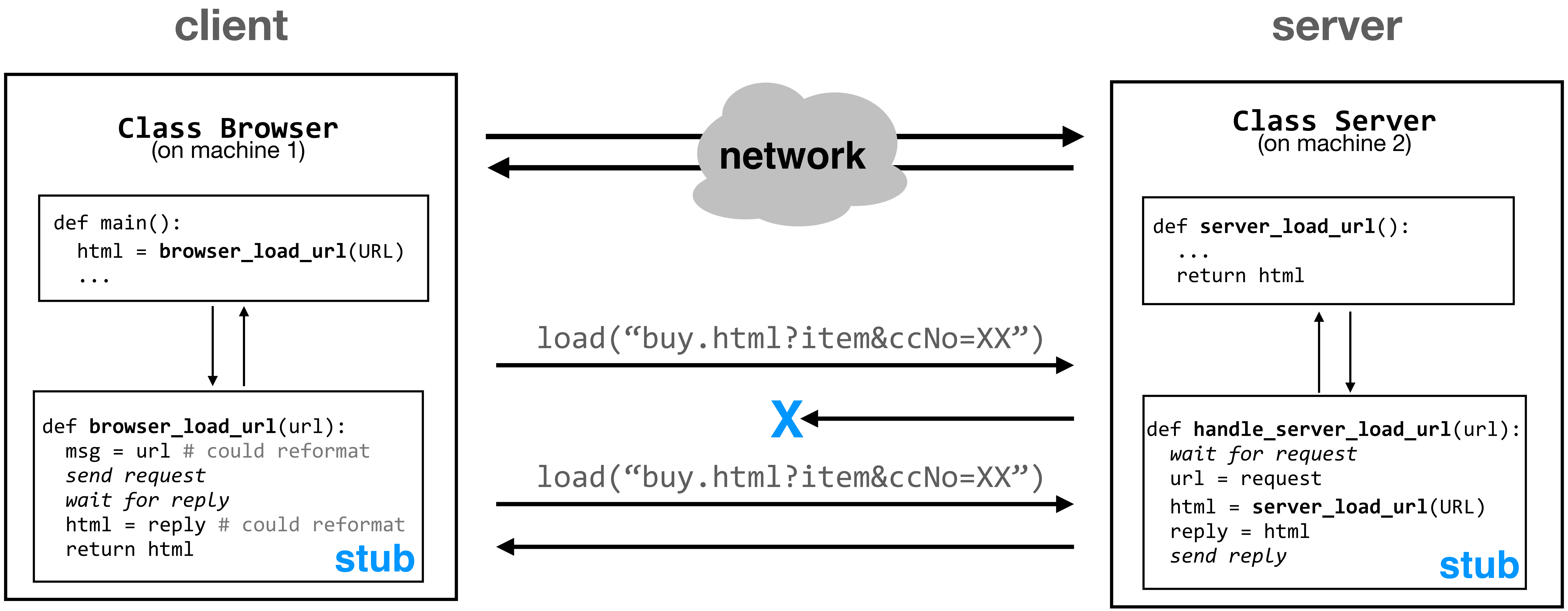
with design principles such as **modularity** and **abstraction**

how do we enforce modularity?

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

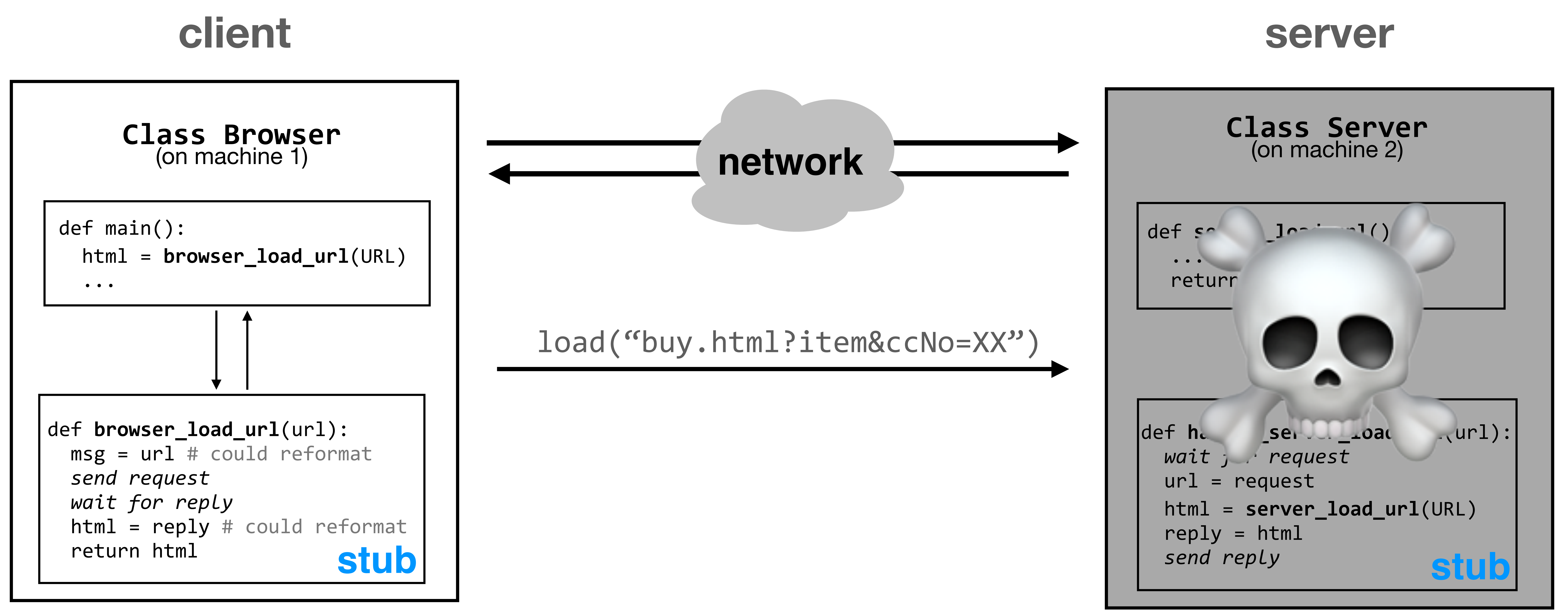



client
server
Class Browser
(on machine 1)
def main():
html = browser_load_url(URL)
...
def browser_load_url(url):
msg = url # could reformat
send request
wait for reply
html = reply # could reformat
return html
stub
network
load("view.html?item")
X
load("view.html?item")
Class Server
(on machine 2)
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



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

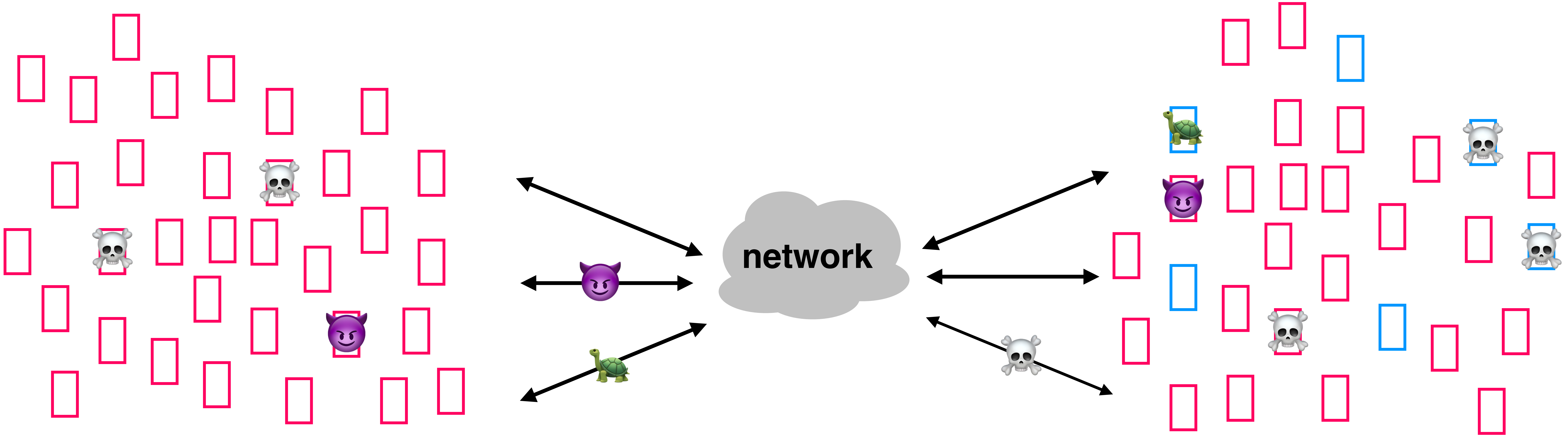


problem: we just bought two copies of `item`

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?

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.

Monday	Tuesday	Wednesday	Thursday	Friday
feb 5 **LEC 1:** Modularity, Abstraction, and the Impact of Systems *First day of classes*	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.

we've already sent out one announcement about a scheduling form — please fill it out today if you haven't already!

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

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**

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