6.033 Spring 2017

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

http://mit.edu/6.033

Schedule

Monday	Tuesday	Wednesday	Thursday	Friday
feb 6 *Reg day*	feb 7 **REC 1:** Worse is Better ***Assigned**: Hands-on DNS* *First day of classes*	feb 8 **LEC 1:** Coping with Complexity: Enforced Modularity and Client/server Organization ***Reading:** Book sections 1.1-1.5, and 4.1-4.3*	feb 9 **REC 2:** Therac-25	feb 10 **TUT 1:** What/How/Why Framework ***Assigned**: Paper critique #1*

Fill out form for recitation assignments

link on home page

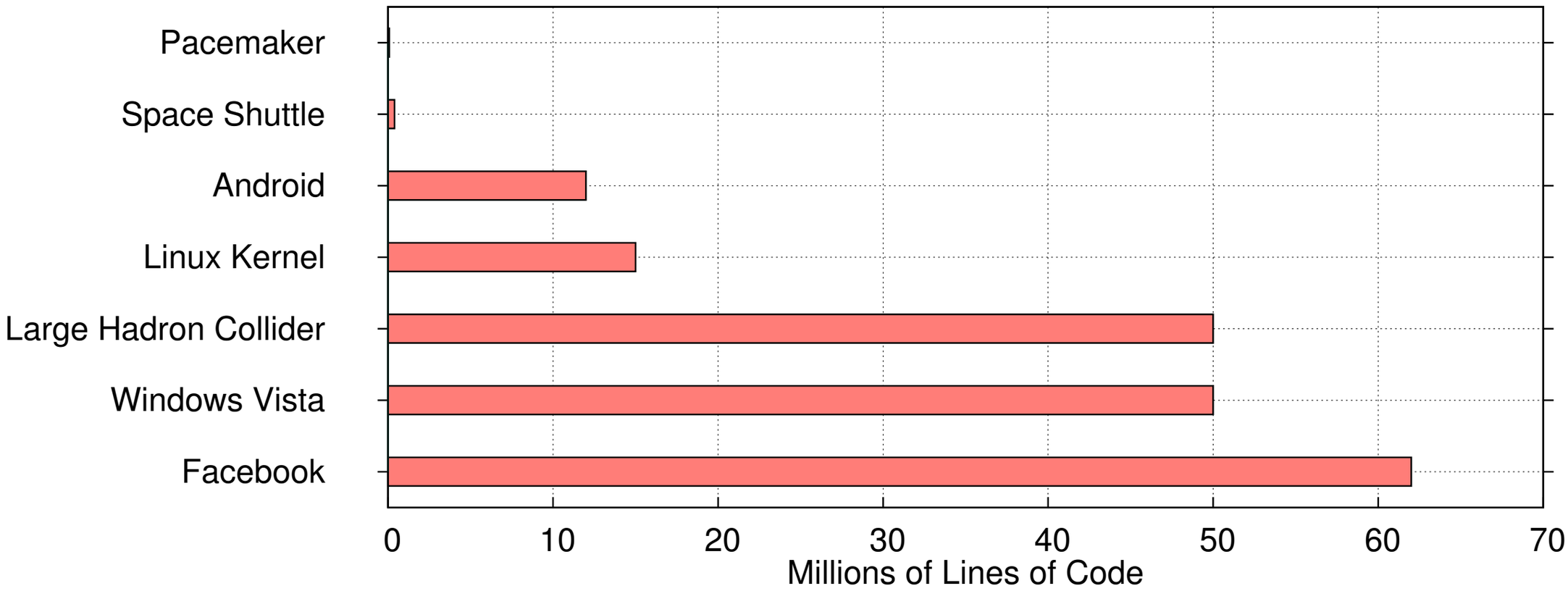
what is a system?

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

what makes building systems difficult?

complexity

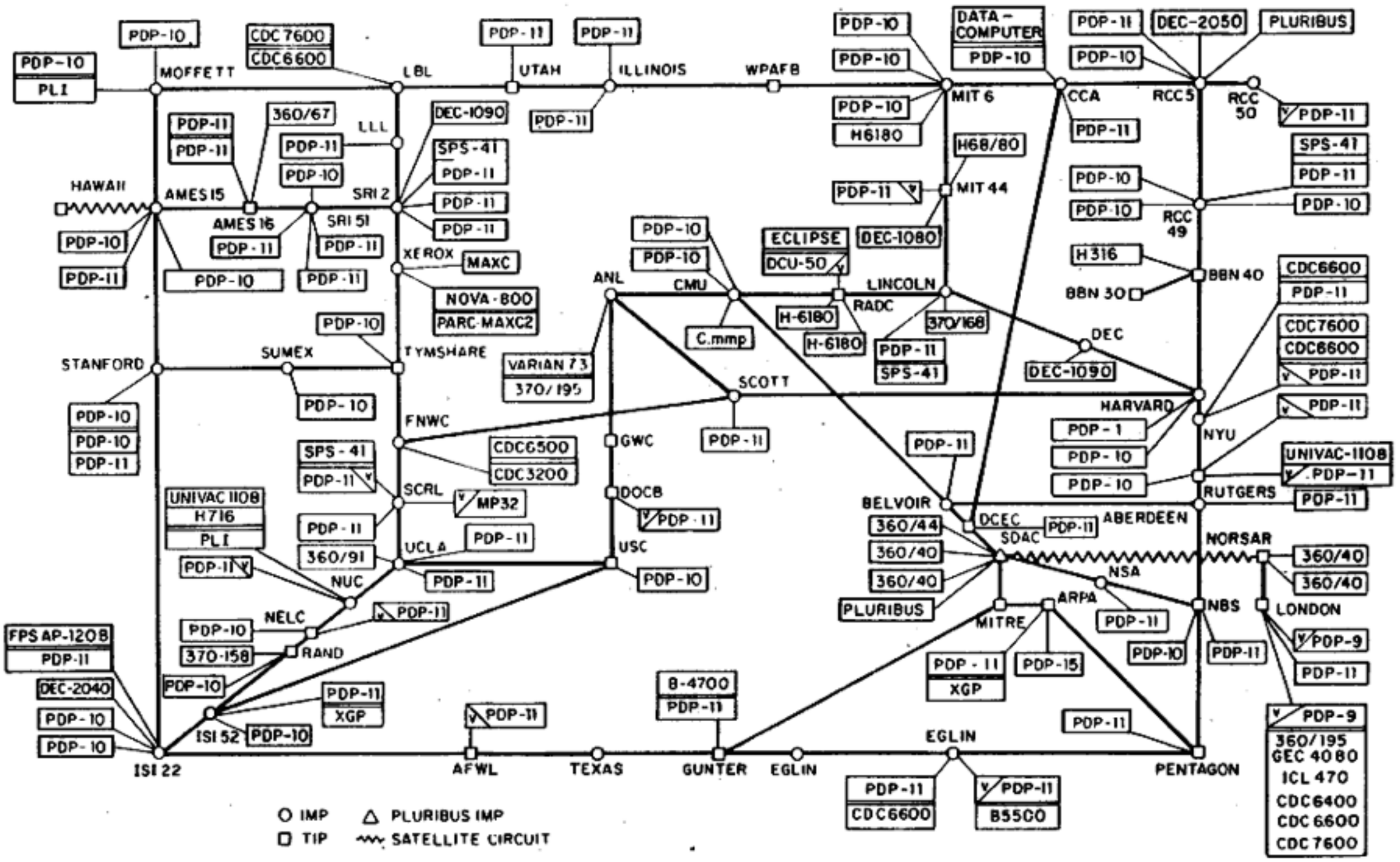
Today's Systems are Incredibly Complex



source: http://www.informationisbeautiful.net/visualizations/million-lines-of-code/

complexity limits what we can build and causes a number of unforeseen issues

ARPANET LOGICAL MAP, MARCH 1977

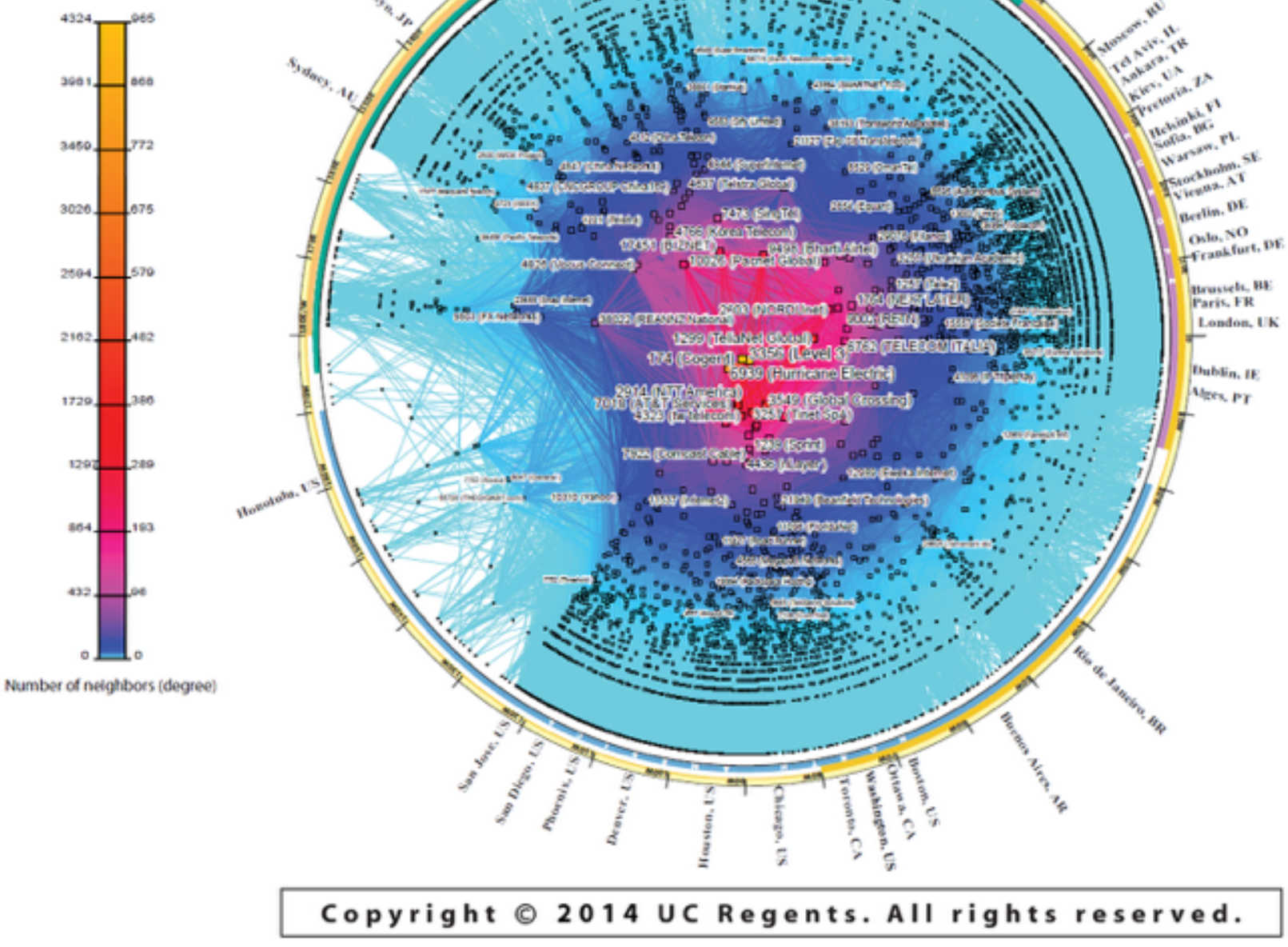


(PLEASE NOTE THAT WHILE THIS MAP SHOWS THE HOST POPULATION OF THE NETWORK ACCORDING TO THE BEST INFORMATION OBTAINABLE, NO CLAIM CAN BE MADE FOR ITS ACCURACY)

NAMES SHOWN ARE IMP NAMES, NOT (NECESSARILY) HOST NAMES

CAIDA's IPv4 AS Core
AS-level Internet Graph

Archipelago January 2014



Copyright © 2014 UC Regents. All rights reserved.

http://www.caida.org/research/topology/as_core_network/2014/

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)

```
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

request

reply

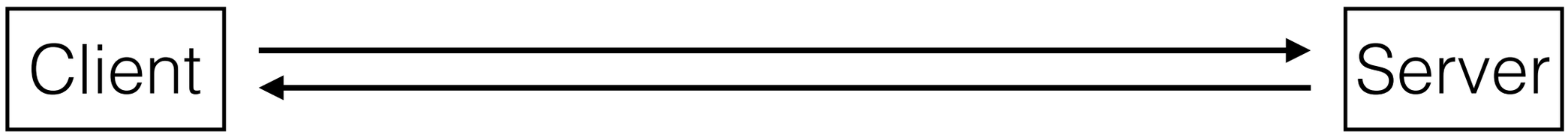
Class webServer
(on machine 2)

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

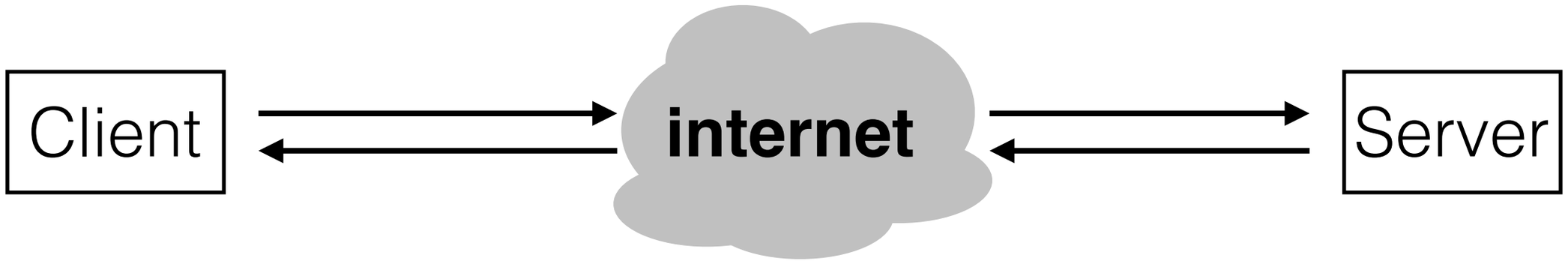
```
def handle_server_load_url(url):
  wait for request
  url = request
  html = server_load_url(URL)
  reply = html
  send reply
```

stub

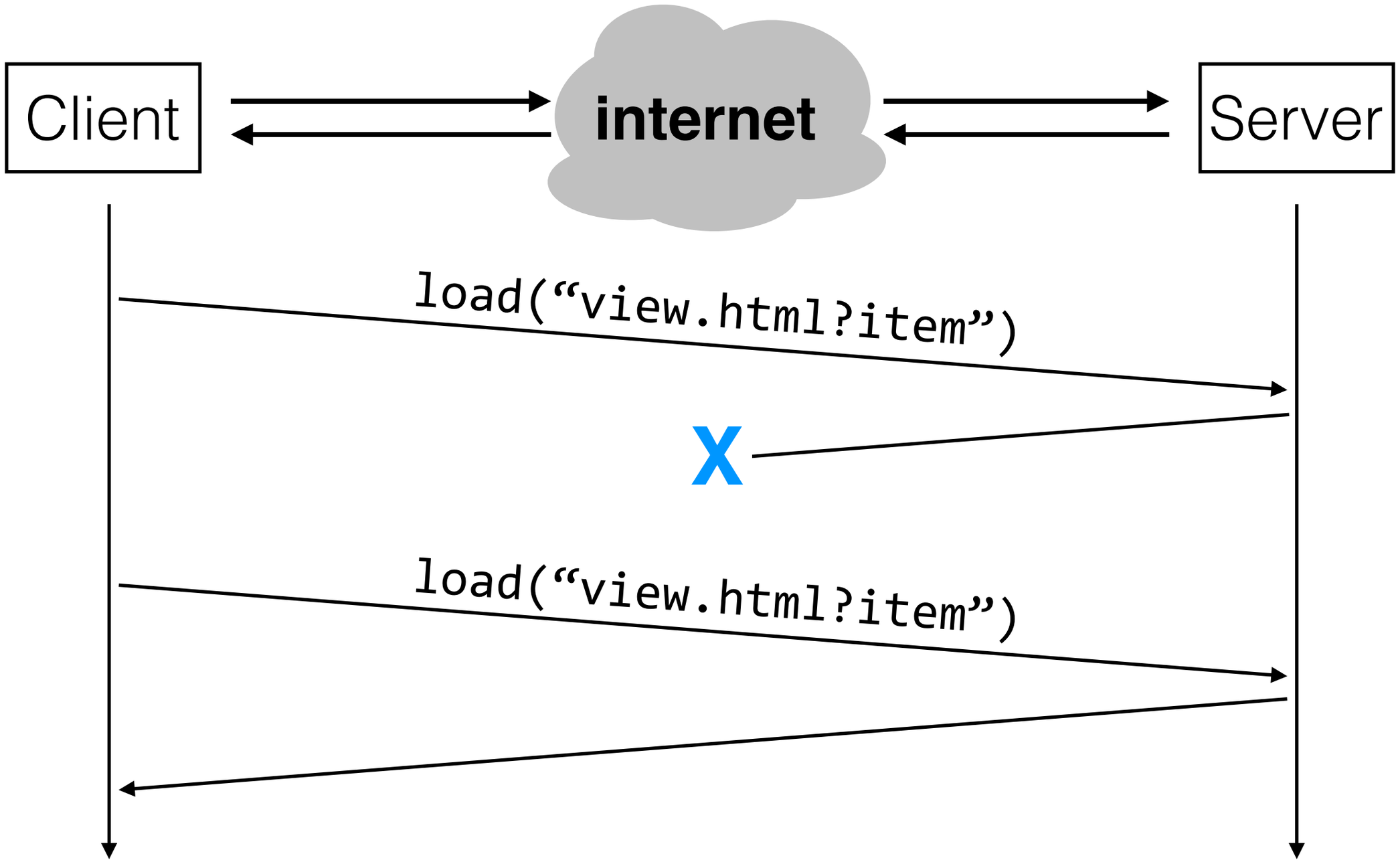
Challenges with RPCs



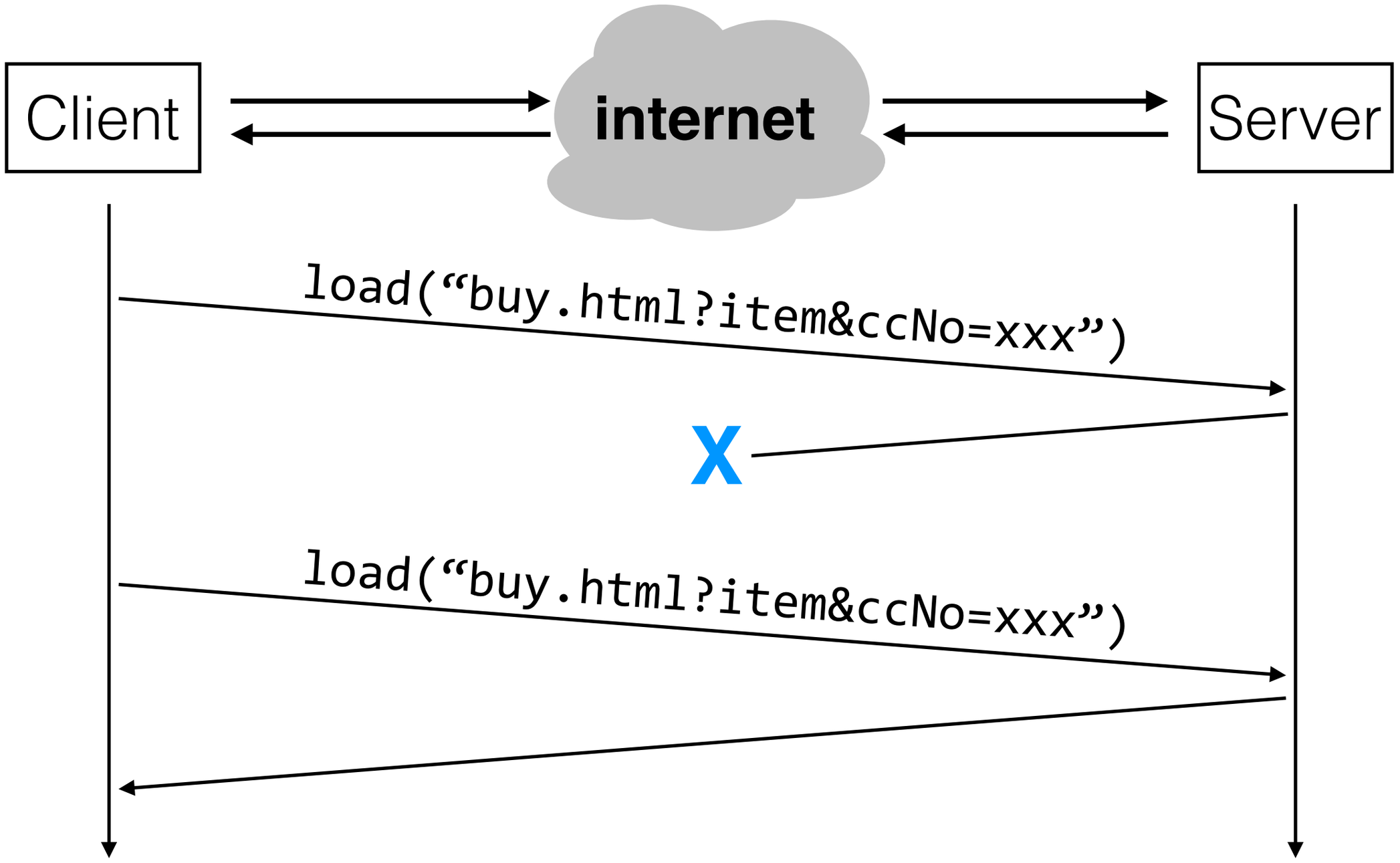
Challenges with RPCs



Challenges with RPCs

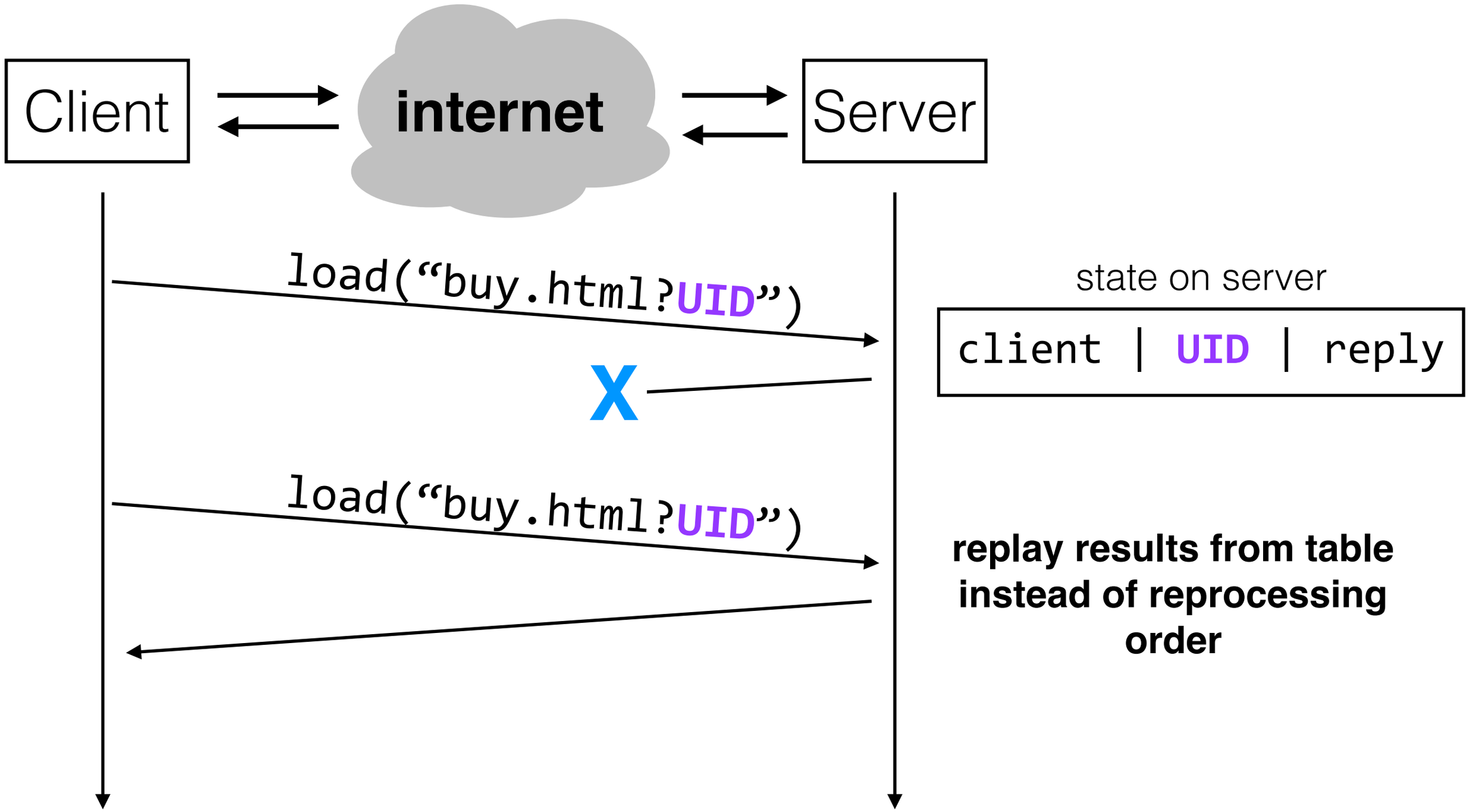


Challenges with RPCs



problem: just bought the same thing twice

Challenges with RPCs



problem: server can still fail

- **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