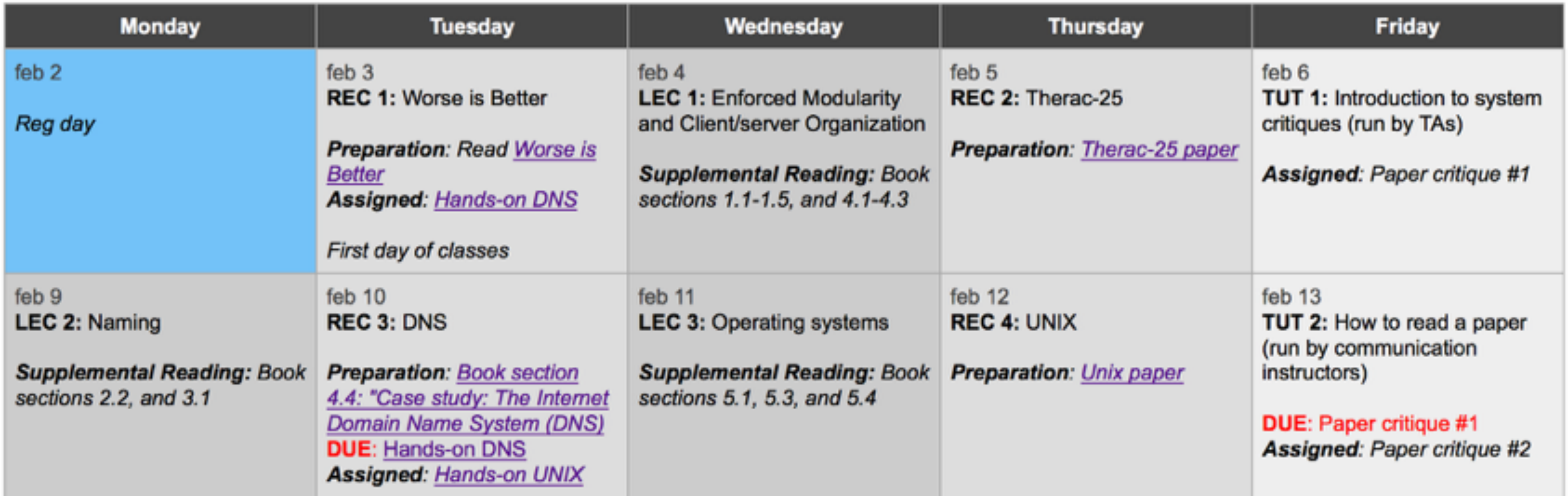
6.033 Spring 2015

Lecture #1

- Complexity
- Modularity and abstraction
- Enforced modularity via client/server models

http://mit.edu/6.033

Schedule



Monday	Tuesday	Wednesday	Thursday	Friday
feb 2 *Reg day*	feb 3 **REC 1:** Worse is Better ***Preparation**: Read Worse is Better* ***Assigned**: Hands-on DNS* *First day of classes*	feb 4 **LEC 1:** Enforced Modularity and Client/server Organization ***Supplemental Reading:** Book sections 1.1-1.5, and 4.1-4.3*	feb 5 **REC 2:** Therac-25 ***Preparation**: Therac-25 paper*	feb 6 **TUT 1:** Introduction to system critiques (run by TAs) ***Assigned**: Paper critique #1*
feb 9 **LEC 2:** Naming ***Supplemental Reading:** Book sections 2.2, and 3.1*	feb 10 **REC 3:** DNS ***Preparation**: Book section 4.4: "Case study: The Internet Domain Name System (DNS)* **DUE**: Hands-on DNS ***Assigned**: Hands-on UNIX*	feb 11 **LEC 3:** Operating systems ***Supplemental Reading:** Book sections 5.1, 5.3, and 5.4*	feb 12 **REC 4:** UNIX ***Preparation**: Unix paper*	feb 13 **TUT 2:** How to read a paper (run by communication instructors) **DUE**: Paper critique #1 ***Assigned**: Paper critique #2*

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

6.033 Approach to Systems

lectures: big ideas + examples

Katrina LaCurts, Hari Balakrishnan

recitations: read papers describing successful systems

Arvind, Mark Day, Dina Katabi, Sam Madden, Martin Rinard, Karen Sollins, Peter Szolovits

hands-ons: play with successful systems

design project: practice designing and writing

TAs: Ellen Finch, David Goehring, Ameesh Goyal, Webb Horn, Qian Long, Manali Naik, Andrew Nguyen, Amy Ousterhout, Cong Yan

Writing staff: Jared Berezin, Amy Carleton, Amelia Herb, Nora Jackson, Janis Melvold, Juergen Schoenstein, Jessie Stickgold-Sarah, Linda Sutliff, Michael Trice

exams: reasoning about system design

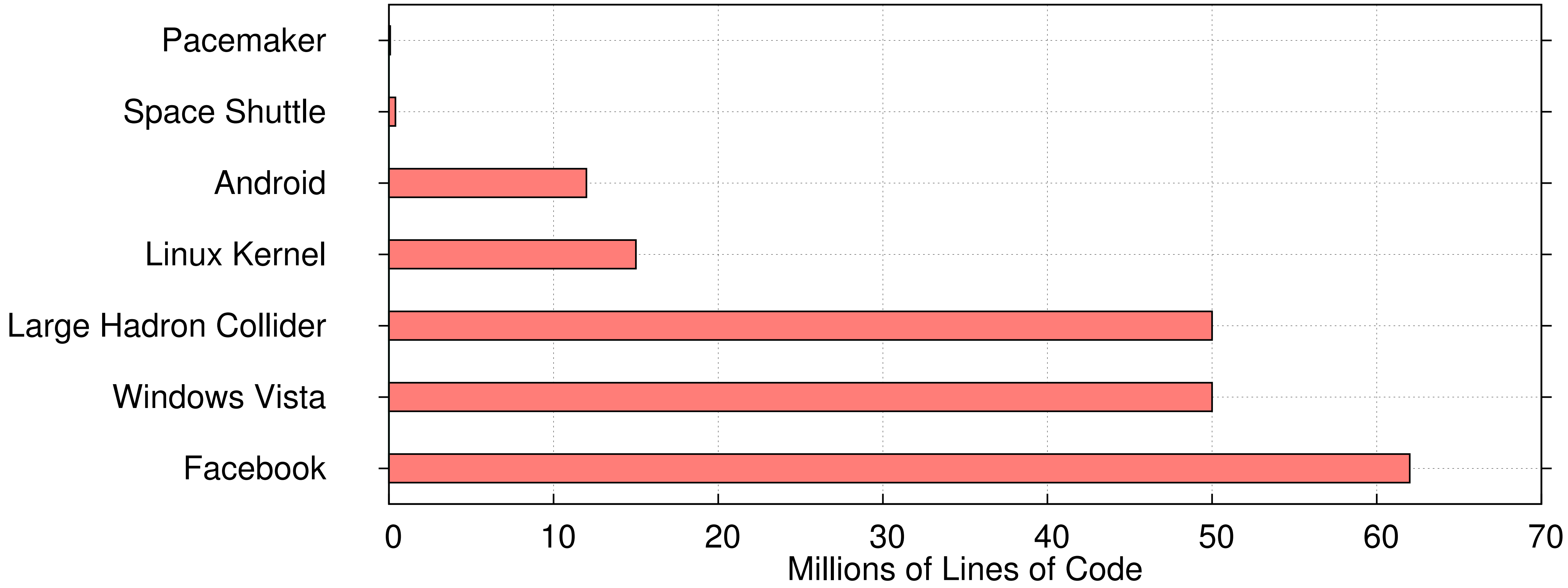
what is a system?

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

what makes building systems difficult?

complexity

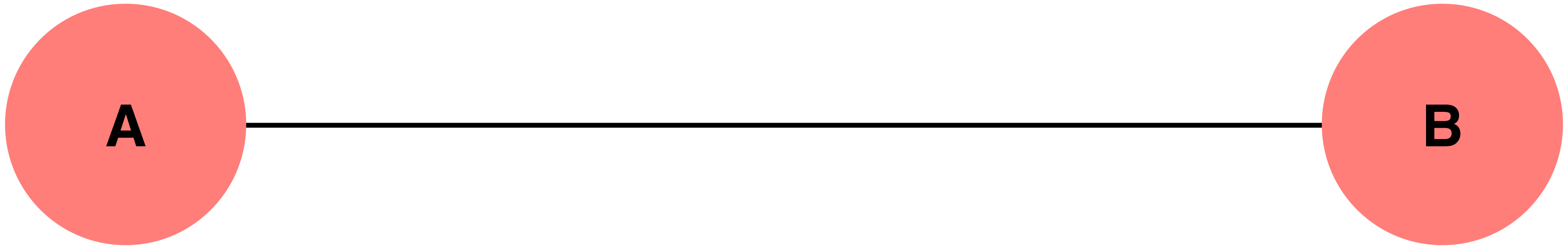
Today's Systems are Incredibly Complex



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

Emergent Properties

(ethernet example)



Emergent Properties

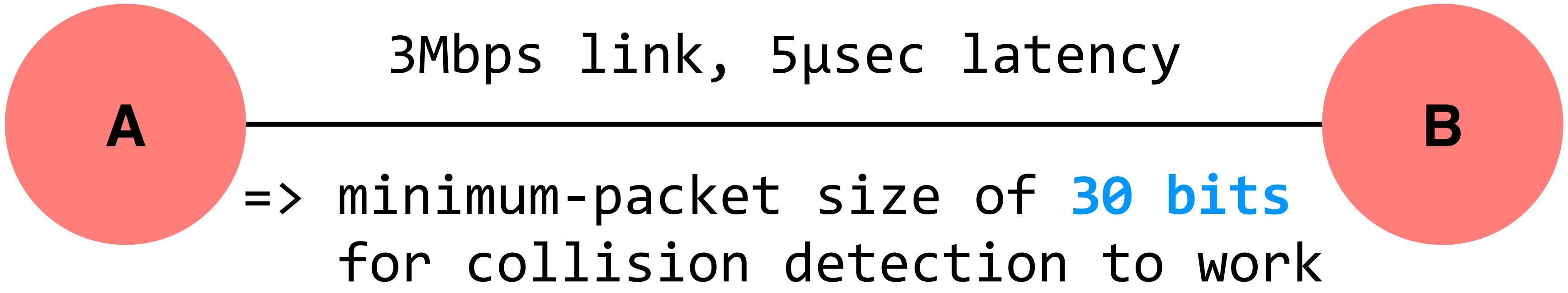
(ethernet example)

collision not detected!

for collision-detection to work, endpoints must send for at least twice the latency of the link

Emergent Properties

(ethernet example)

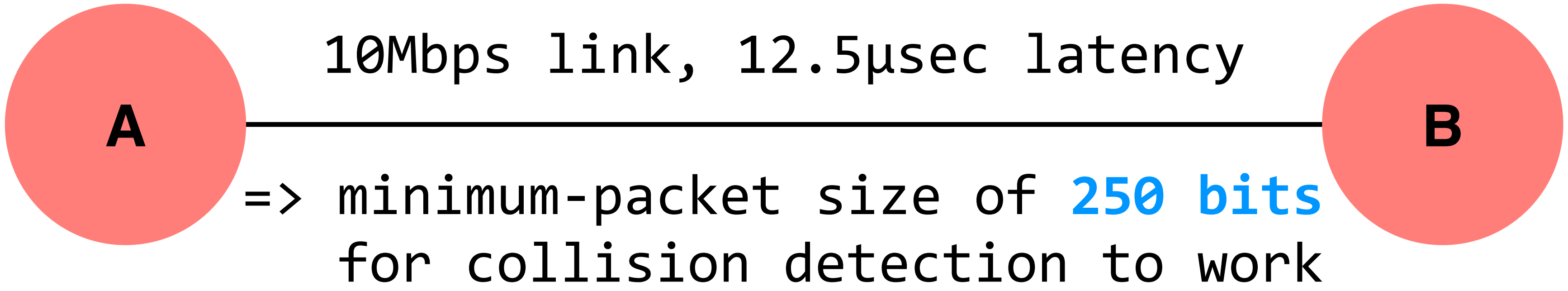


=> minimum-packet size of **30 bits** for collision detection to work

experimental ethernet: 3Mbps link, 5μsec latency, **40-bit** packet headers

Emergent Properties

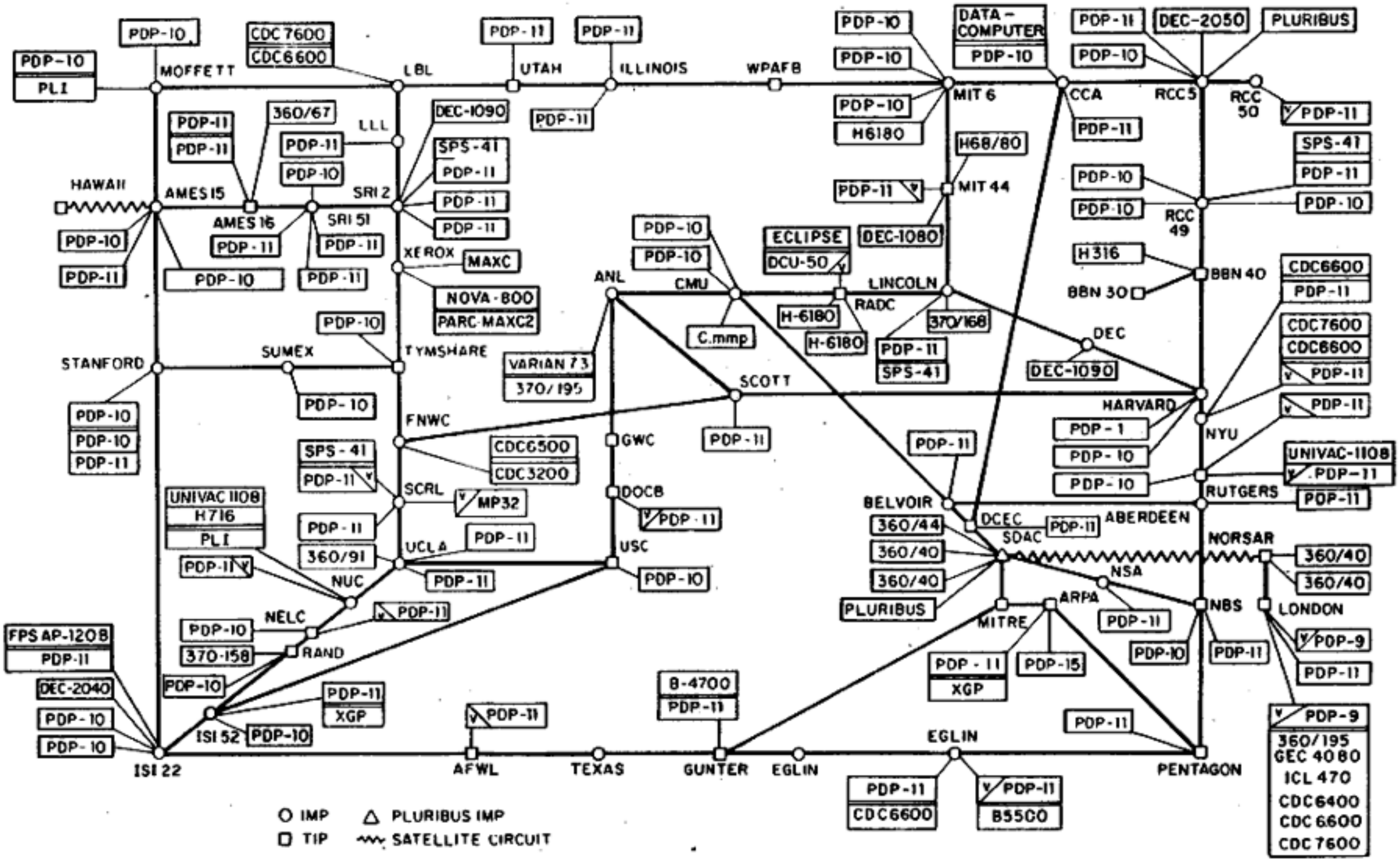
(ethernet example)


A
B
10Mbps link, 12.5μsec latency
=> minimum-packet size of 250 bits
for collision detection to work

first ethernet standard: 10Mbps link, 12.5μsec latency, **112-bit** packet headers

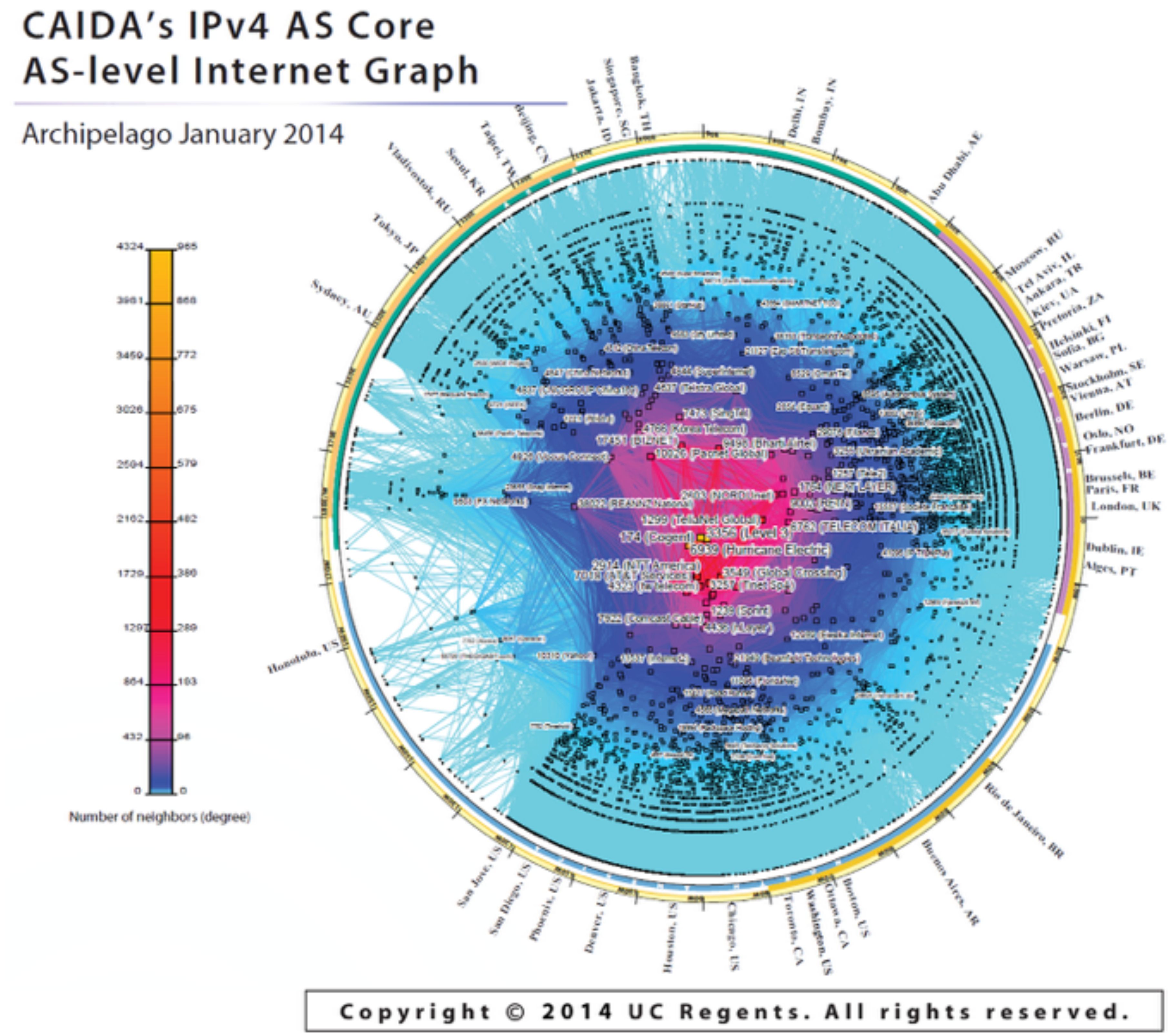
minimum packet size was an **emergent property** of ethernet

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

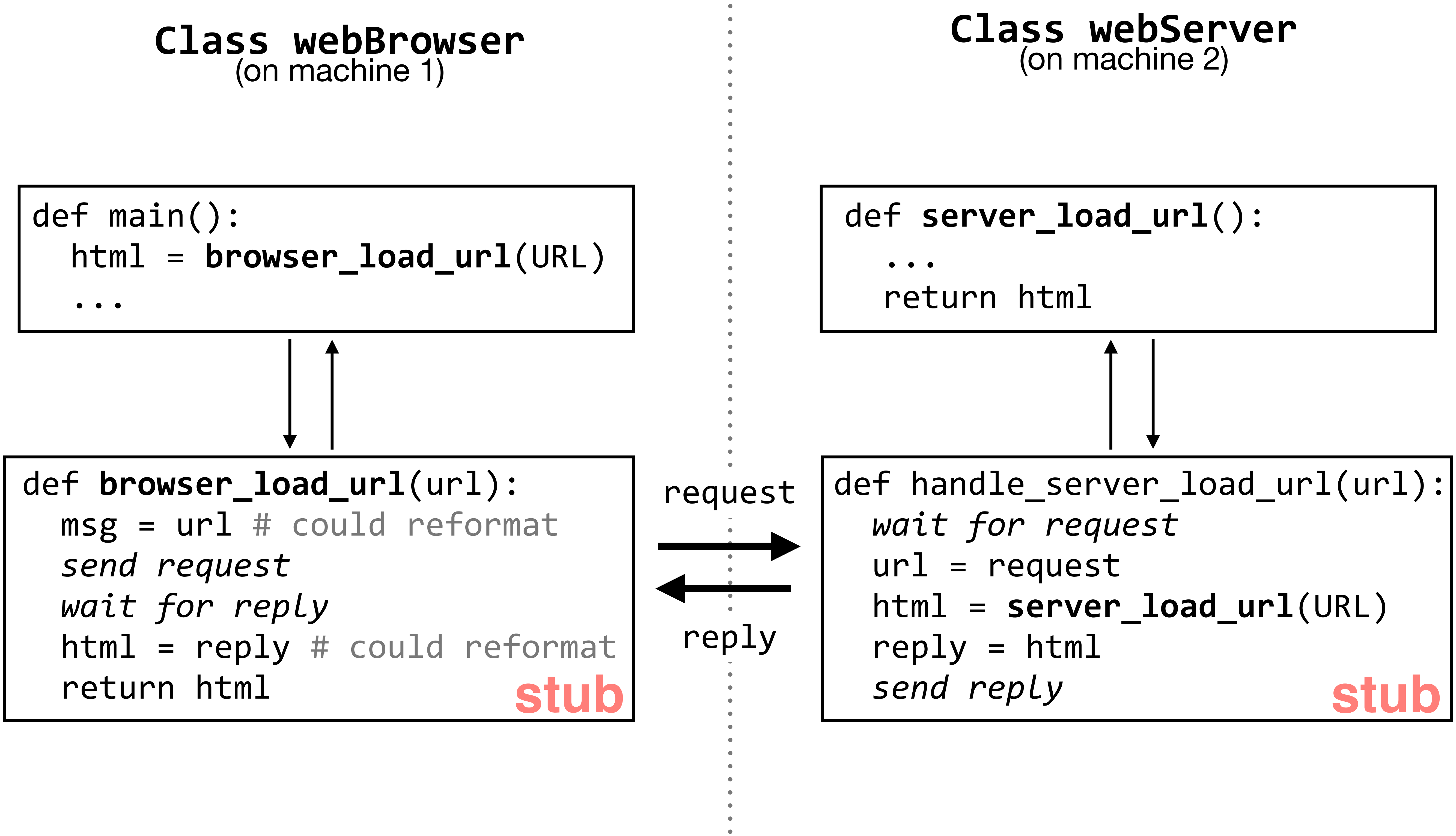


http://www.caida.org/research/topology/as_core_network/2014/

how can we mitigate complexity?

how do we enforce modularity?

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

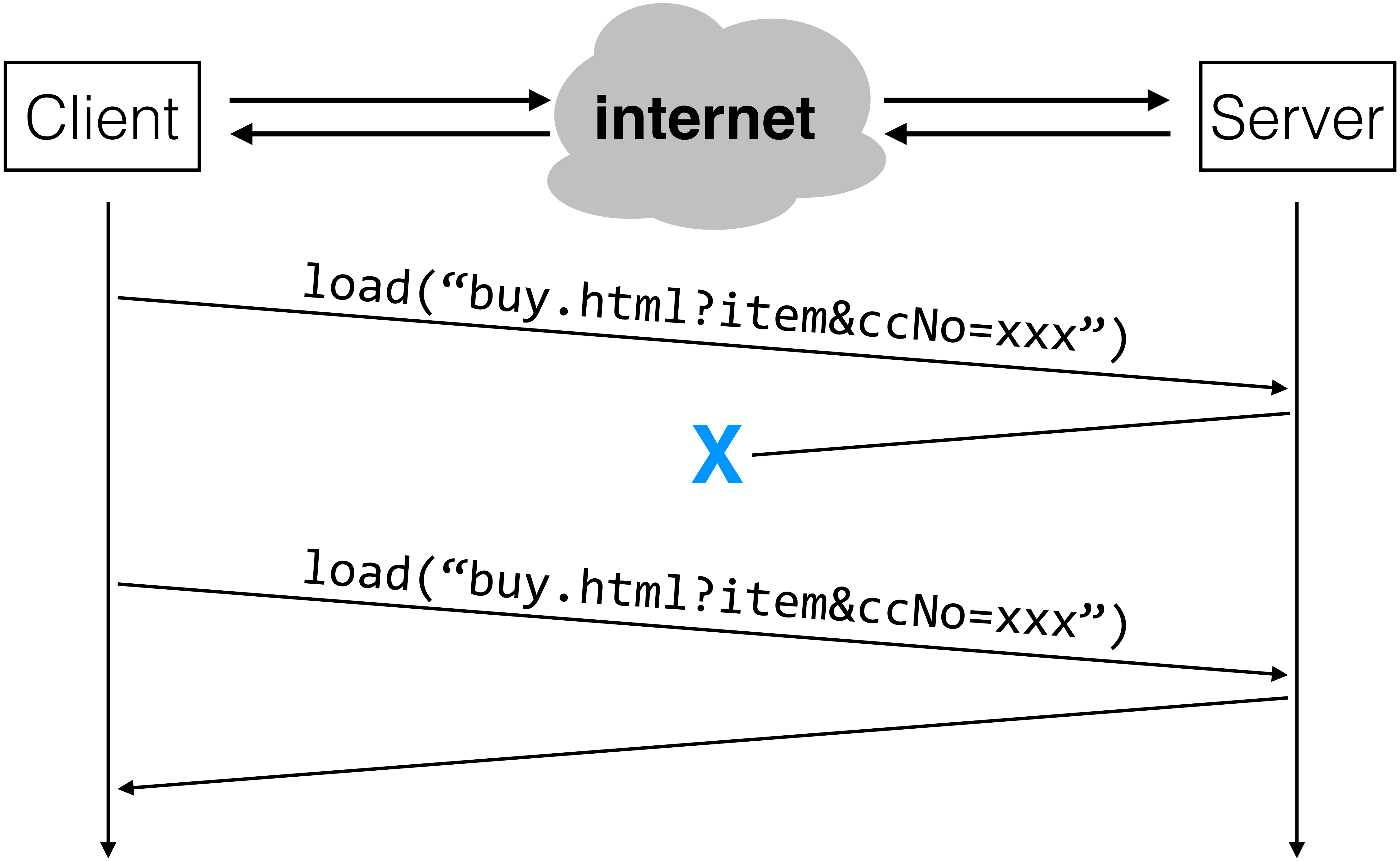
Class webServer
(on machine 2)

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

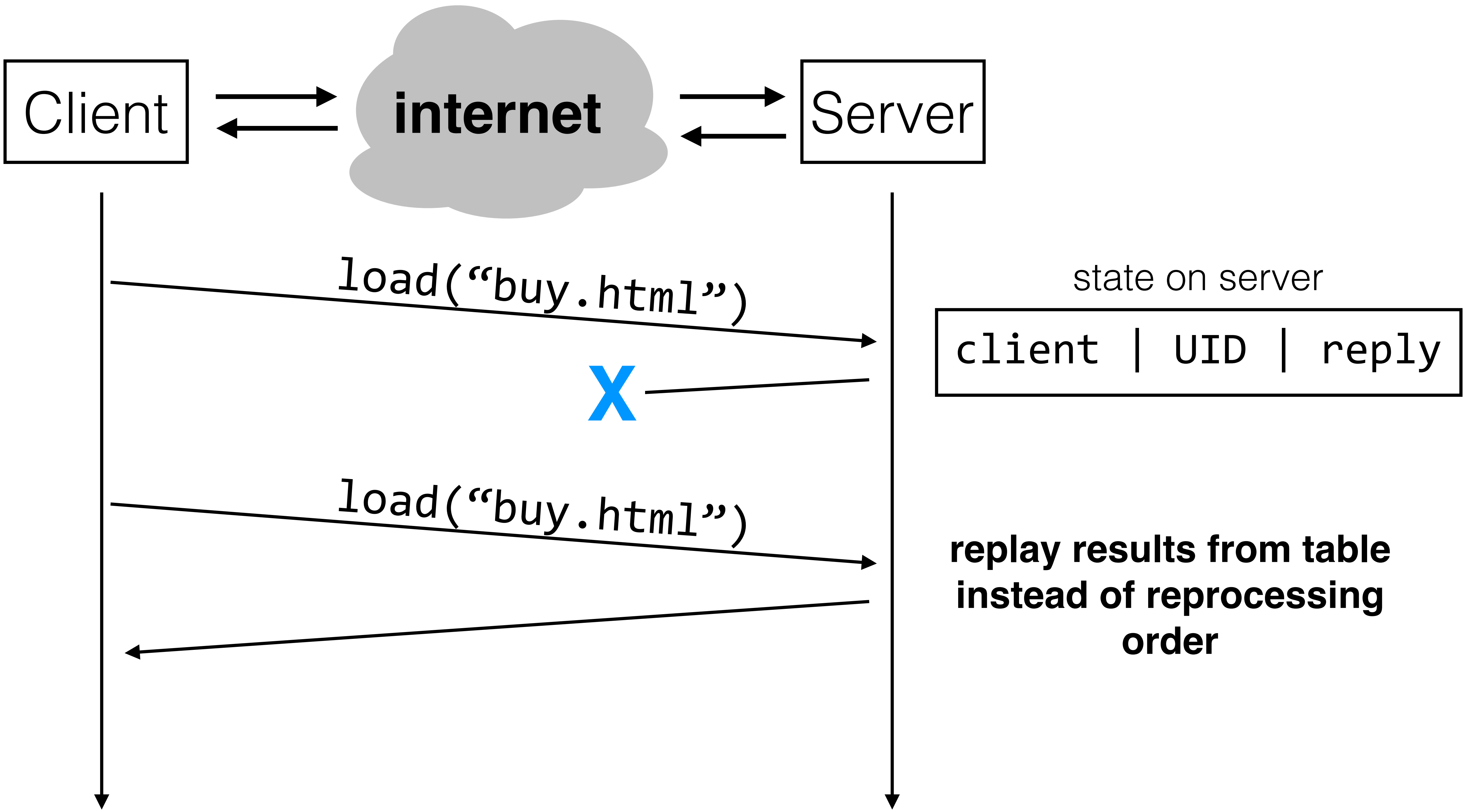
```
def handle_server_load_url(url):
  wait for request
  url = request
  html = server_load_url(URL)
  reply = html
  send reply
```

stub

Challenges with RPCs


Client
internet
Server
load(“buy.html?item&ccNo=xxx”)
X
load(“buy.html?item&ccNo=xxx”)

Challenges with RPCs



state on server

client	UID	reply

replay results from table instead of reprocessing order

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