L1: Complexity, Enforced Modularity, and client/server organization

Sam Madden and Dina Katabi

6.033 Spring 2014

http://web.mit.edu/6.033

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

http://web.mit.edu/6.033

- Schedule has all assignments
  - Every meeting has preparation/assignment

- On-line registration form to sign up for section and tutorial times
  - We will post sections assignment this evening

Monday	Tuesday	Wednesday	Thursday	Friday
feb 3 *Reg day*	feb 4 **REC 1:** Worse is Better **Preparation**: Read Worse is Better **Assigned**: Hands-on DNS *First day of classes*	feb 5 **LEC 1:** Enforced Modularity and Client/server Organization **Preparation**: Book sections 1.1-1.5, and 4.1-4.3	feb 6 **REC 2:** Therac-25 **Preparation**: Therac-25 paper	feb 7 **TUT 1:** Writing program section (run by CI and TAs) **Assigned**: Memo #1
feb 10 **LEC 2:** Naming **Preparation**: Book sections 2.2, and 3.1	feb 11 **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 12 **LEC 3:** Operating systems **Preparation**: Book sections 5.1, 5.3, and 5.4	feb 13 **REC 4:** UNIX **Preparation**: Unix paper	feb 14 **TUT 2:** Design project 1 (run by TAs) **Preparation**: Book section 2.5: "Case study: UNIX File System Layering and Naming" **DUE**: Memo #1

What is a system?

System = *Interacting set of components with a specified behavior at the interface with its environment*

Examples: Web, Linux

6.033 : study and design of systems, their components, and internals

6.033 Approach

- *Lectures/book*: big ideas and examples
- *Hands-ons*: play with successful systems
- *Recitations*: papers describing successful systems
- *Design projects*: you practice designing and writing
  - Design: choose problem, tradeoffs, structure
  - Writing: explain core ideas concisely
- *Exams*: focus on reasoning about system design

Why is building systems hard?

Example Complex System: Linux Kernel

- 1975 Unix kernel: 10,500 lines of code
- 2008 Linux 2.6.24 line counts:

  85,000 processes
  430,000 sound drivers
  490,000 network protocols
  710,000 file systems
  1,000,000 different CPU architectures
  4,000,000 drivers
  7,800,000 Total

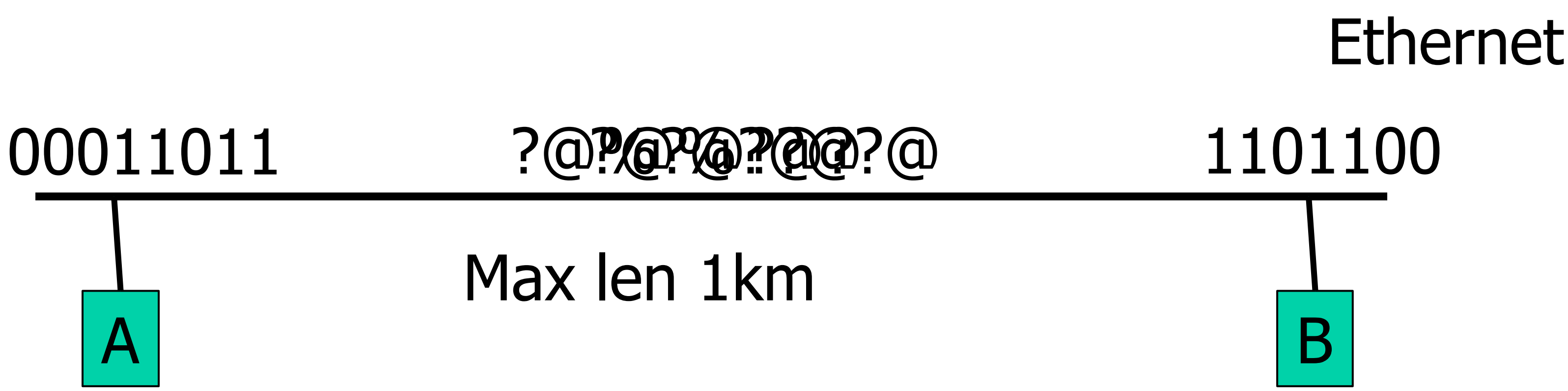
- More examples: http://www.informationisbeautiful.net/

Emergent Property Example: Ethernet

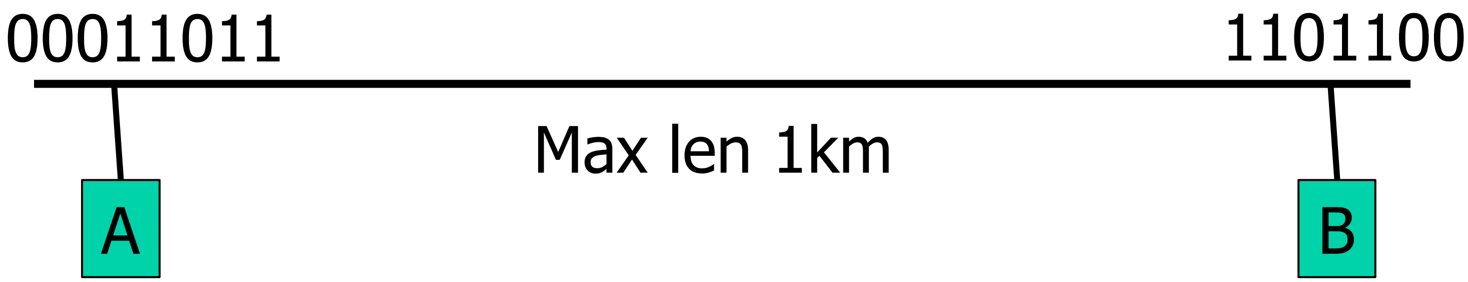
- All computers share single cable
- Goal is reliable delivery
- Listen while sending to detect collisions

Emergent Property Example: Ethernet

- All computers share single cable
- Goal is reliable delivery
- Listen while sending to detect collisions



Does Collision Detection Work?



What if A finishes sending before data from B arrives?
Can this happen?

1 km at 60% speed of light = 5 microseconds
Original Ethernet Spec: 3 Mbit / sec

➔A can send 15 bits before bit 1 arrives at B

➔A must keep sending for 2 * 5 microseconds

(To detect collision when first bit from B arrives)

➔Minimum packet size is 5* 2 * 3 = 30 bits
Default header is 5 bytes (40 bits), so no problem!

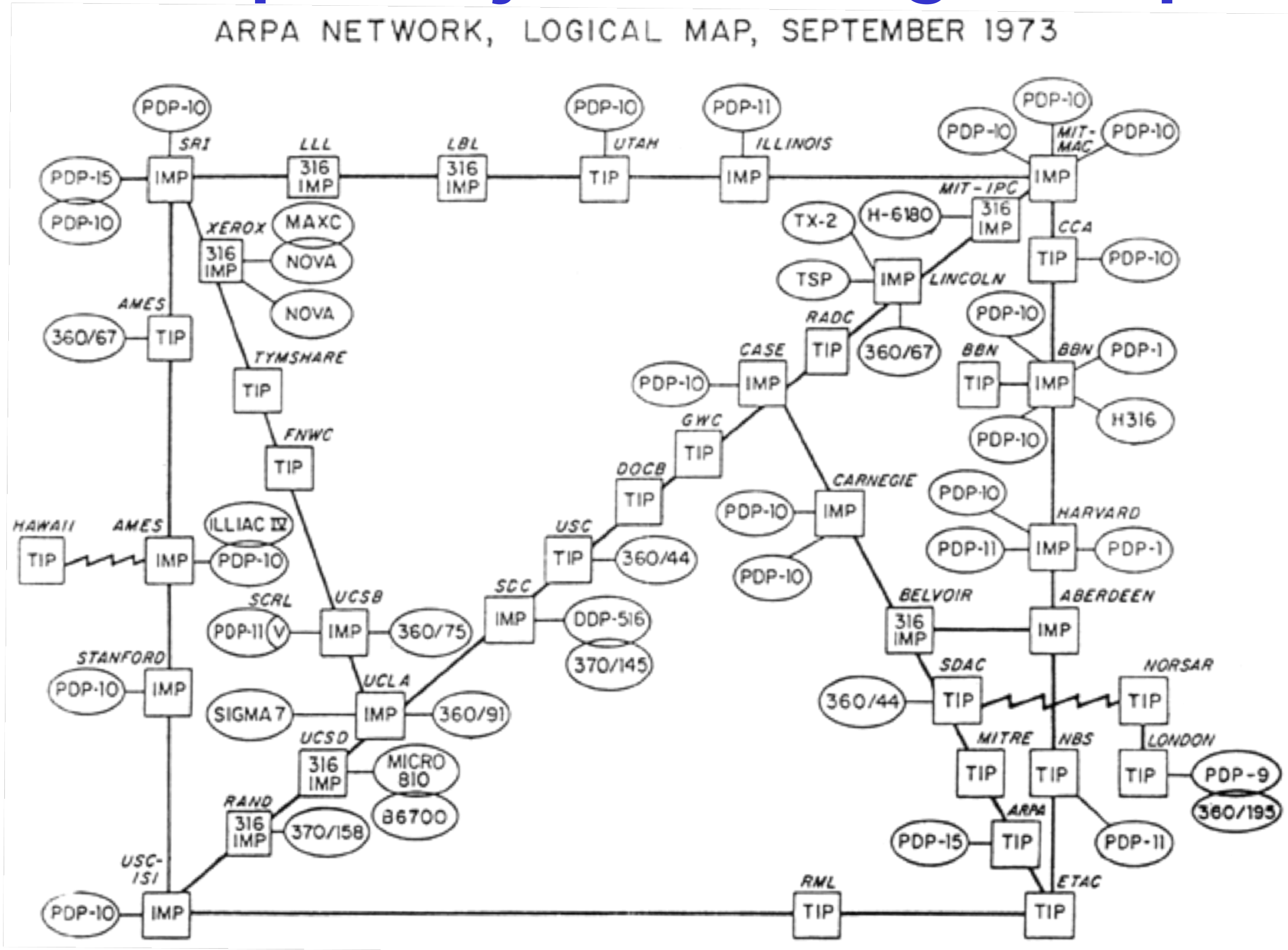
3 Mbit/s ➔ 10 Mbit/s

- First Ethernet standard: 10 Mbit/s, 2.5 km wire
  - Must send for 2*12.5 μseconds = 250 bits @ 10 Mb/s
  - Header was 14 bytes

  ➔Needed to pad packets to at least 250 bits (32 bytes)

Emergent property: Minimum packet size!

A computer system scaling example



Scaling the Internet

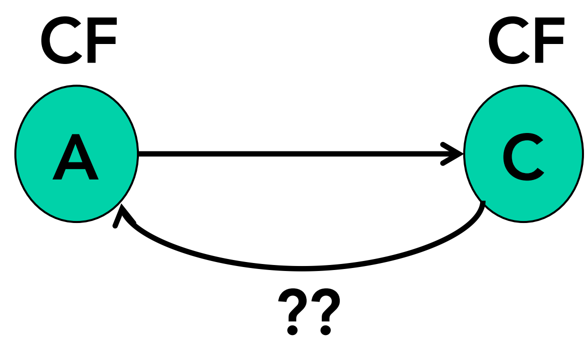
- Size routing tables (for shortest paths): $O(n^2)$
  - Hierarchical routing on network numbers
  - Address: 16 bit network # and 16 bit host #
- Limited networks (2^{16})

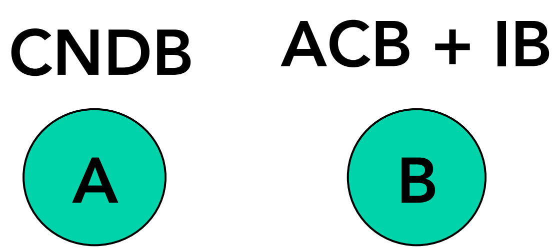
➔ Network Address Translators and IPv6

Example: No Small Changes

Phone network features

- Call Forwarding
- Call Number Delivery Blocking
- Automatic Call Back
- Itemized Billing



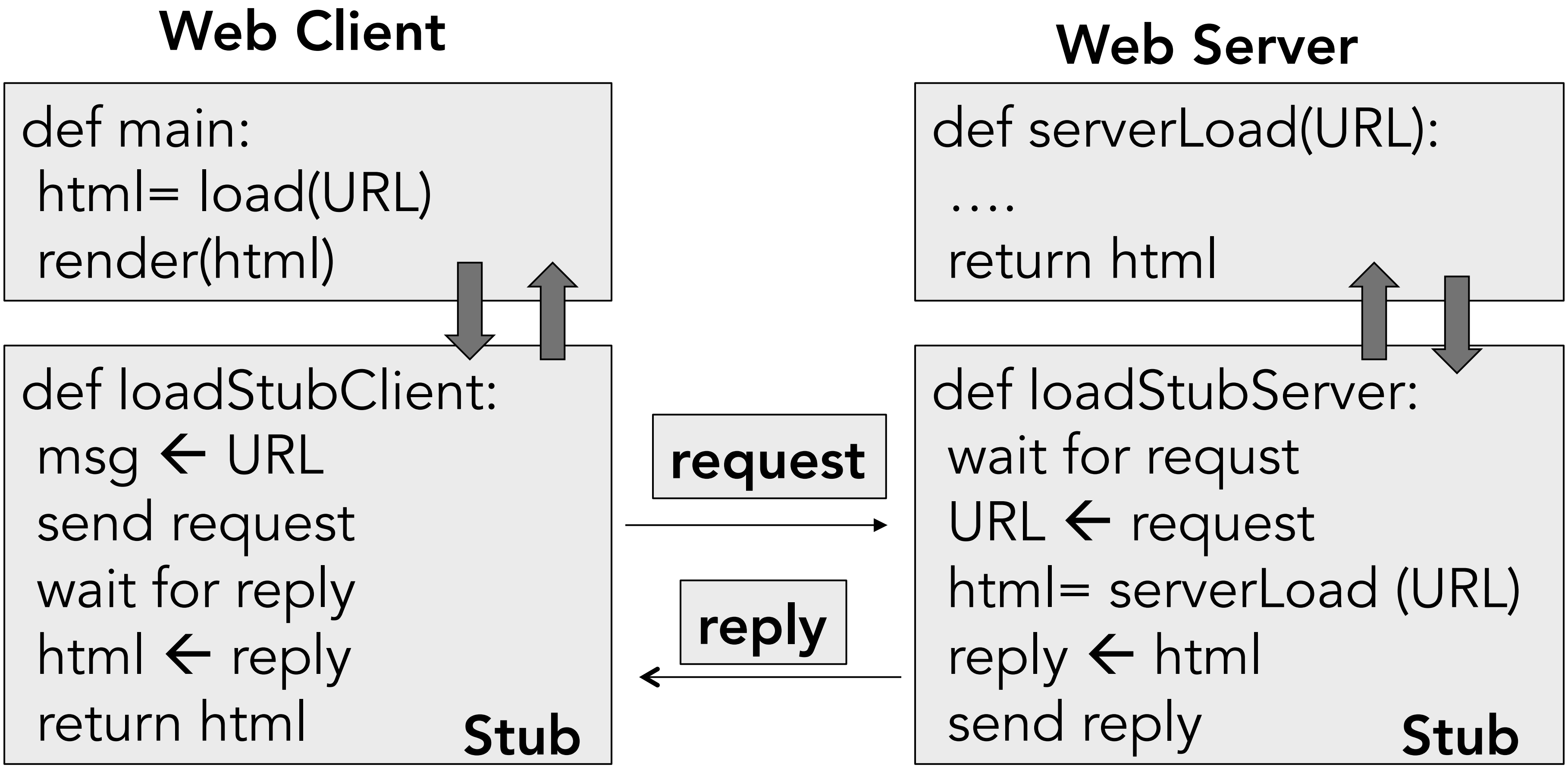


- **A calls B, B is busy**
- **Once B is done, B calls A**
- **A's number on appears on B's bill**

How can we mitigate the complexity of building systems?

Enforcing Modularity with Client/ Server

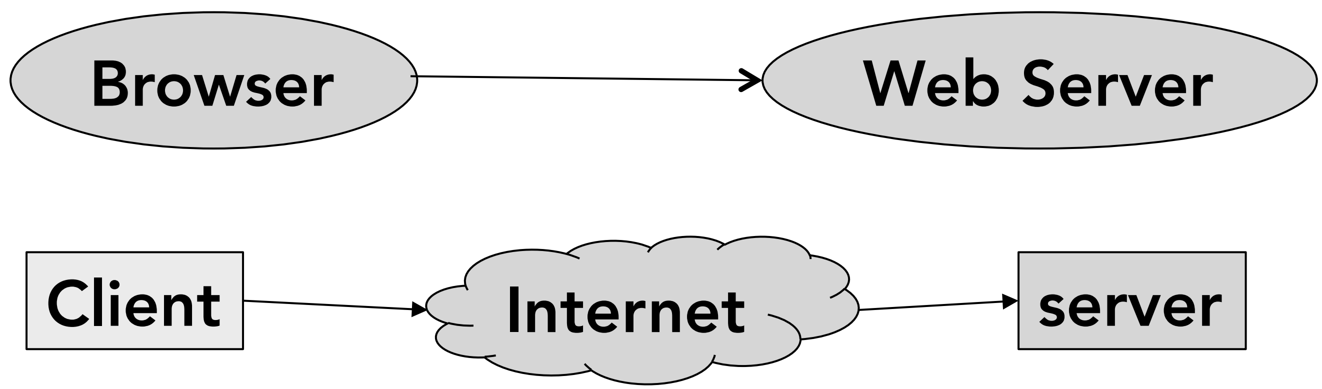
Remote Procedure Call



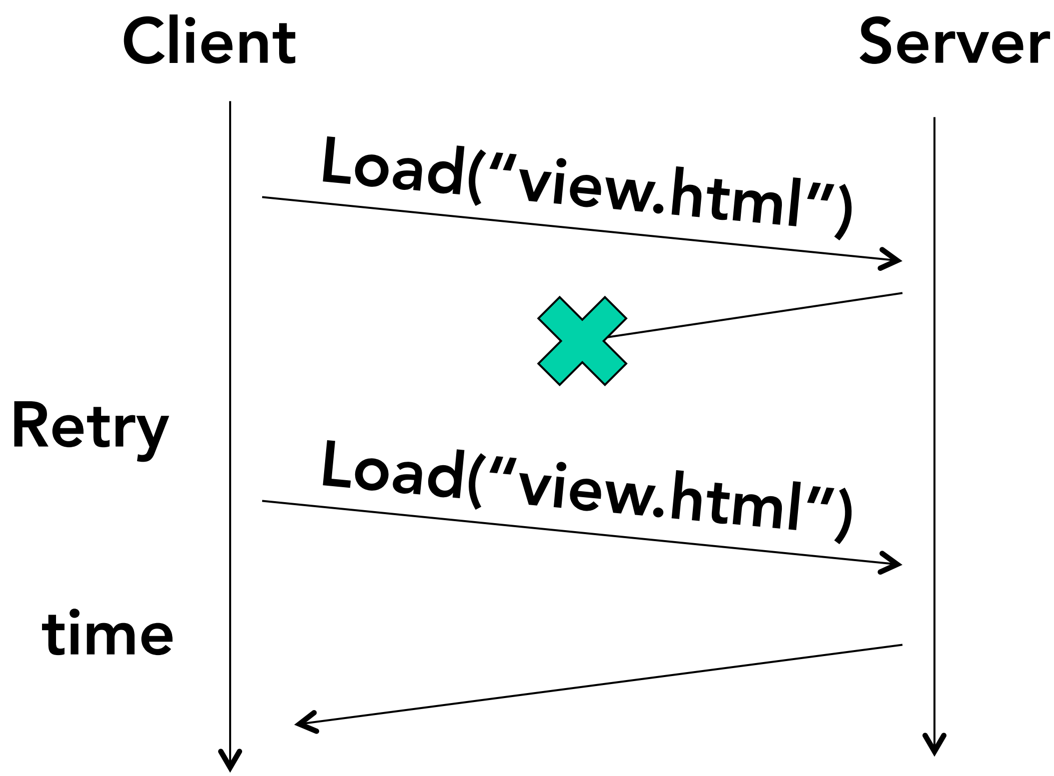
- Stubs make client/server look like procedure calls!
- Stubs can be automatically generated

RPC != PC

Load("view.html?bieberAlbum") ➔ HTML
Load("buy.html?bieberAlbum&ccNo=xxx")

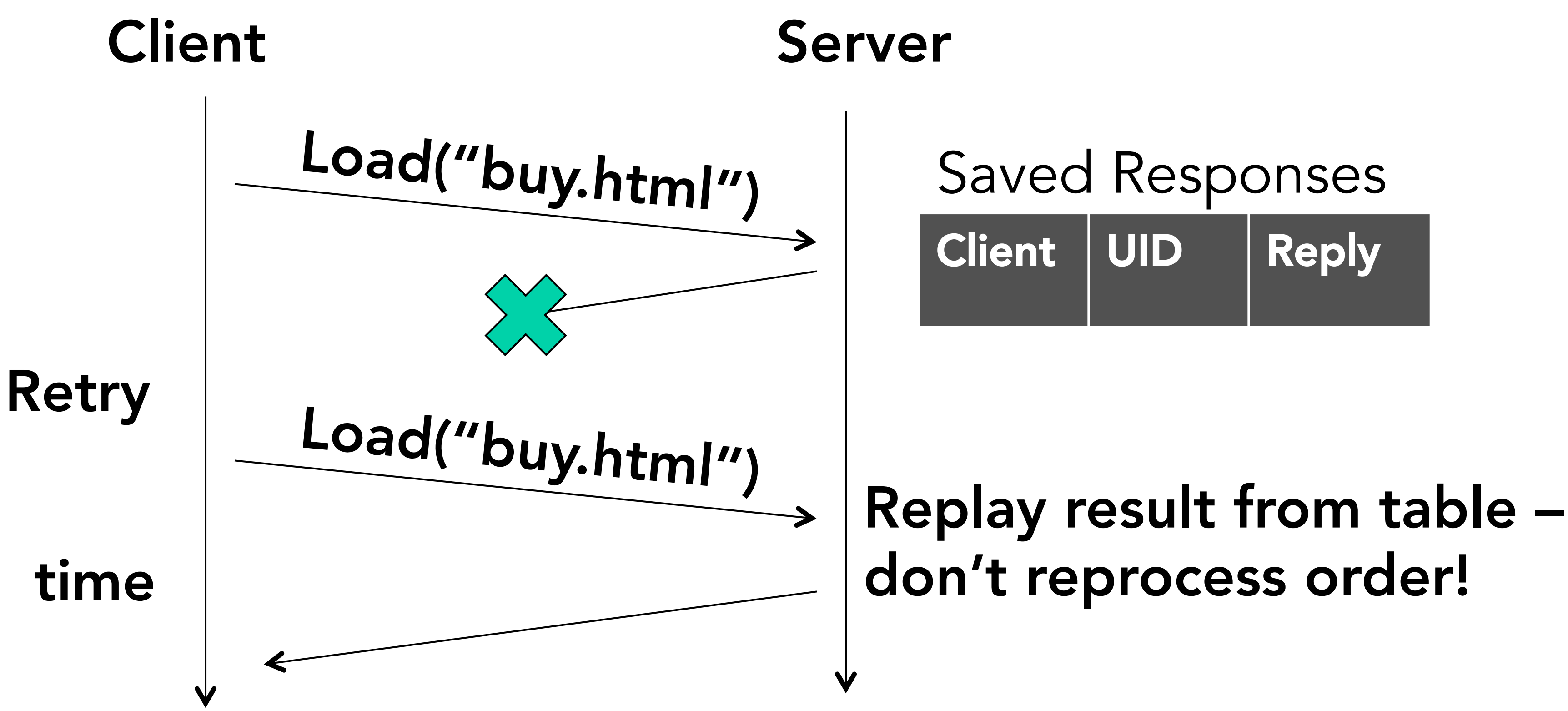


Challenge 1: network looses requests



- Approach: Retry after time out
- Doesn't work for buy.html

Soln: Filter Duplicate Requst



- What if server fails?

Challenge 2: server fails

- "Unknown" outcome for load("buy.html")

  Did the server process the request or not?

- Removing "unknown" outcome requires heavy-duty techniques

  Topic for April

- Practical solution: Expose that RPC != PC

  RPC caller must handle "serverFailed" exception

Summary so far

- Complexity makes building systems difficult
- Modularity and abstraction bound complexity
- Can enforce modularity through client/server
  - Remote procedure call simplifies C/S
  - Unfortunately, RPC != PC
- Failures will be a central challenge in 6.033
- No algorithm for successful system design

Example 6.033 systems

- Therac-25
  bad design, at many levels. detailed post-mortem
- UNIX
- The Internet
- MapReduce
- Relational Databases

Class plan

- *Client/server*: Naming
- *Operating systems*:
  - Enforced modularity within a machine
- *Networks*:
  - Enforced modularity between machines
- *Reliability and transactions*:
  - Handing hardware failures
- *Security*: handling malicious failures