CS61A Lecture 28
Distributed Computing

Jom Magrotker
UC Berkeley EECS

August 6, 2012

Computer Science in the News

AI predicts when you're about to get sick

11:54 26 July 2012

Michael Reilly, senior technology editor

If you've been walking around a public place lately, you've come in contact with a lot of people. Some of those people may have been sick. And if you've been hanging around enough of them as they cough and sneeze, then you might be about to get sick too.

http://www.newscientist.com/blogs/onepercent/2012/07/ai-predicts-when-youre-about-t.html

Today

- Distributed Computing
  - Network Architecture
  - Protocol
  - Design Ideas
- A Chat Program (with a demo, fingers crossed)

Review: Logic Programming

Write the rule `rotate_left`, which checks that the second input list is the result of taking the first list and shifting the first item to the back of the list.

```
P?> rotate_left(<a, b, c>, <b, c, a>)
Yes.
P?> rotate_left(<3, 2, 1>, ?wat)
Yes.
?wat = <2, 1, 3>
```

Review: Logic Programming

Write the rule `rotate_left`, which checks that the second input list is the result of taking the first list and shifting the first item to the back of the list.

```
rule append(<?f | ?r>, ?s, <?f | ?a>):
    append(?r, ?s, ?a)
fact append(<>, ?z, ?z)

fact rotate_left(<>, <>)
rule rotate_left(<?first | ?rest>, ?rotated):
    append(?rest, <?first>, ?rotated)
```

Review: Logic Programming

Write the rule `rotate_right`, which checks that the second input list is the result of taking the first list and shifting the last item to the front of the list.

```
P?> rotate_right(<a, b, c>, <c, a, b>)
Yes.
P?> rotate_right(<3, 2, 1>, ?wat)
Yes.
?wat = <1, 3, 2>
```

REVIEW: LOGIC PROGRAMMING

Write the rule `rotate_right`, which checks that the second input list is the result of taking the first list and shifting the last item to the front of the list.

```
rule rotate_right(?first, ?second):
    rotate_left(?second, ?first)
```

SO FAR

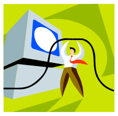
- functions
- data structures
- objects
- abstraction
- interpretation
- evaluation

One Program
One Computer

TODAY & TOMORROW

- Multiple Programs!
  - On Multiple Computers
    (Networked and Distributed Computing)

  - On One Computer
    (Concurrency and Parallelism)



TODAY: DISTRIBUTED COMPUTING

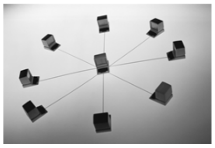
- Programs on different computers working together towards a goal.
  - Information Sharing & Communication
    Ex. Skype, The World Wide Web, Cell Networks
  - Large Scale Computing
    Ex. "Cloud Computing" and Map-Reduce

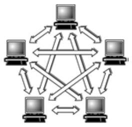
DISTRIBUTED COMPUTING

1. Independent Computers
2. (Often) In Different Locations
3. Connected by a Network
4. Communicating by Passing Messages
5. Shared Computational Goal

TOPICS IN DISTRIBUTED COMPUTING

- Architecture
  - Client-Server
  - Peer-to-Peer
- Message Passing
- Design Principles
  - Modularity
  - Interfaces





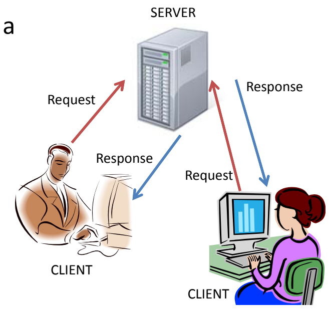
http://www.peertopeerprograms.com/

CLIENT-SERVER ARCHITECTURE

Great for dispensing a service.

2 Roles:

- Clients: make requests to the server.
- Server: listens for requests and responds to them.

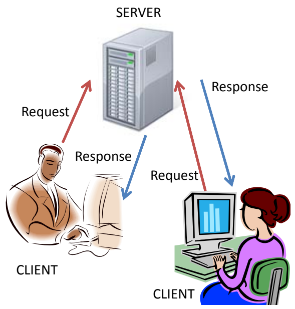


CLIENT-SERVER ARCHITECTURE

The Trick:

Where to put different features?

Example: “Should I store X data with the client or the server?”



Example: The World Wide Web

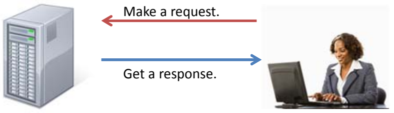
Server:

- Holds content.
- (Constantly) listens for clients to request content.
- Sometimes performs computation before responding.

Client:

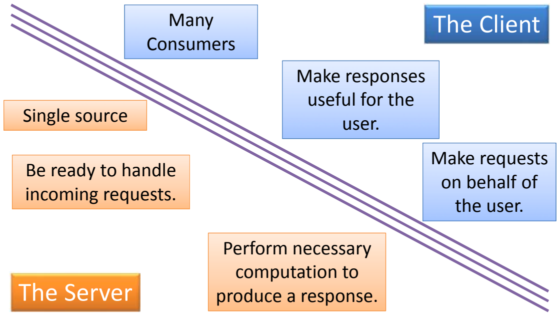
- Requests content.
- Makes (correctly formatted) requests to the server for content.
- Responsible for making sense of the response.

Figures out the proper web page to give back.



Figures out how to display the web page that it got.

DIVISION OF LABOR



Many Consumers

The Client

Make responses useful for the user.

Single source

Make requests on behalf of the user.

Be ready to handle incoming requests.

Perform necessary computation to produce a response.

The Server

SINGLE POINT OF FAILURE

What happens if the server goes down?

- Everything stops working for everyone!

What if the client goes down?

- Only that client stops working, everyone else continues as needed.

PEER TO PEER ARCHITECTURE

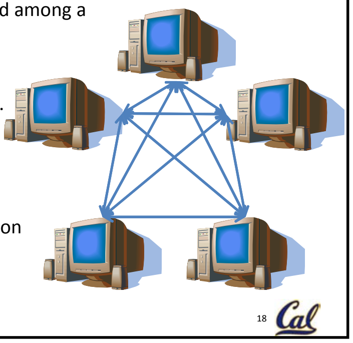
Great for distributing load among a large pool of computers.

All computers:

- Send and receive data.
- Contribute resources.

Useful for:

- Data Storage
- Large-scale Computation
- Communication



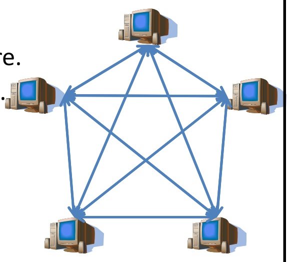
Peer to Peer Architecture

Advantages:

- No single point of failure.
- Can scale with demand.

Disadvantages:

- Gets pretty complex.
- Inefficient communication.



Announcements

- Project 4 due **Tuesday, August 7**.
  - Partnered project, in two parts.
  - Twelve questions, so *please start early*!
  - Two extra credit questions.
- Homework 14 due **Tuesday, August 7.**
  - Will include contest voting later today.
  - Assignment is short.

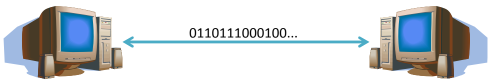
Announcements: Final

- Final is **Thursday, August 9**.
  - *Where*? 1 Pimentel.
  - *When*? 6PM to 9PM.
  - *How much*? *All* of the material in the course, from June 18 to August 8, will be tested.
- Closed book and closed electronic devices.
- One 8.5” x 11” ‘cheat sheet’ allowed.
- No group portion.
- We have emailed you if you have conflicts and have told us. If you haven’t told us yet, please *let us know* by yesterday.
- Final review sessions on **Tonight, August 6** and **Tomorrow, August 7**, from **8pm to 9:30pm** in the HP Auditorium (306 Soda).

Communicating

Computers need to be able to send messages back and forth to...

- Coordinate behavior.
- Transfer data.
- Make requests.
- Indicate status of a request.



Message Structure

There has to be a predefined message structure if computers are to understand one another.

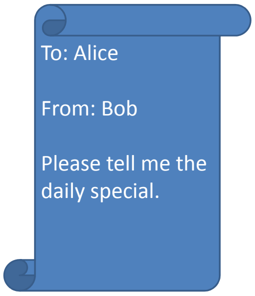
Similar to how people need to agree on a language if they are to communicate effectively.

Message Structure

Typically a message includes something like:

- Sender
- Receiver
- Action or Response

However, message format can vary a lot with the application.



To: Alice

From: Bob

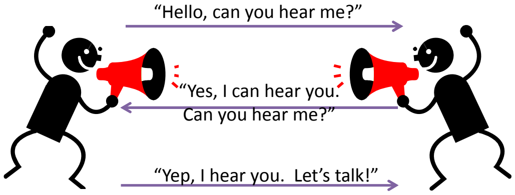
Please tell me the daily special.

Protocol

- For a distributed system to work, we need to have standardized methods for talking to each other.
- A protocol is the formalisms expected when programs talk to one another.
- One example is the message format.
- Another example: What's the convention used for two computers starting up a conversation with one another?

The Three Way Handshake

Extremely common protocol for establishing two-way communication.


"Hello, can you hear me?"
"Yes, I can hear you. Can you hear me?"
"Yep, I hear you. Let's talk!"

The Three Way Handshake

Why 3 ways?

Need to make sure both sides are hearing each other!

1. First message is to try to see if the first computer can be heard.
2. Second message confirms that the first computer can be heard and acts as a test to make sure the second computer can also be heard.
3. Third message confirms the second computer can be heard.

Break

YOUR EXCUSE FOR ANYTHING TODAY:
"SORRY— I WAS UP ALL NIGHT TRYING TO DOWNLOAD PHOTOS TAKEN BY A ROBOT LOWERED ONTO MARS BY A SKYCRANE."

http://xkcd.com/1091/

Demo: Chat Program

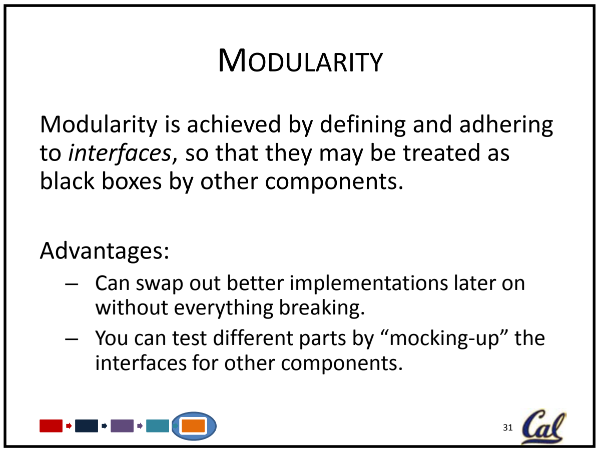
At this point we're going to attempt demo-ing the chat program you'll be working with in lab today.

It's not exactly the most robust chat server, so we'll see how it goes.

Design Principles in Distributed Systems

The idea of abstraction is still important for this style of programming!

The goal is to make each part of the distributed system as *modular* as possible. Should be able to switch out any component with something that behaves the same way without any noticeable changes.



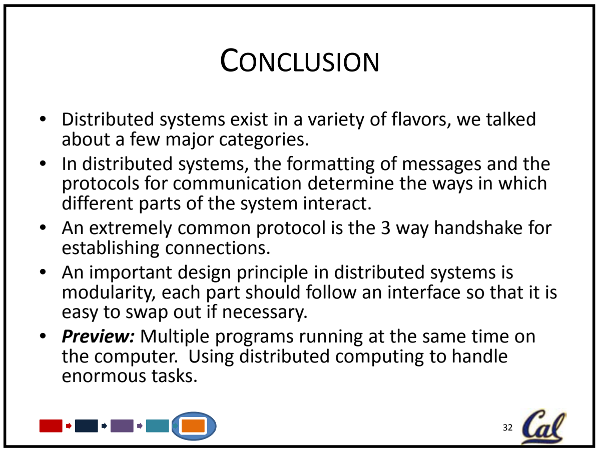
MODULARITY

Modularity is achieved by defining and adhering to *interfaces*, so that they may be treated as black boxes by other components.

Advantages:

- Can swap out better implementations later on without everything breaking.
- You can test different parts by "mocking-up" the interfaces for other components.

31



CONCLUSION

- Distributed systems exist in a variety of flavors, we talked about a few major categories.
- In distributed systems, the formatting of messages and the protocols for communication determine the ways in which different parts of the system interact.
- An extremely common protocol is the 3 way handshake for establishing connections.
- An important design principle in distributed systems is modularity, each part should follow an interface so that it is easy to swap out if necessary.
- ***Preview:*** Multiple programs running at the same time on the computer. Using distributed computing to handle enormous tasks.

32