

CSCW and Groupware (and some basic networking)

IAT 351

Week11 Lecture 1

19.03.2008

Lyn Bartram

lyn@sfu.ca



SCHOOL OF INTERACTIVE
ARTS + TECHNOLOGY

Today's agenda

- Administrivia – Assignment 5, Phidgets availability
- What is CSCW?
- What is groupware?
- How do we build it?
 - Networking and sockets
 - Servers and clients

Assignment 5

- Make your PhotoAlbum a shared application
- Several users (2) can view and annotate
- One user can see another user's comments
- Implement a simple server and client CSCW tool
- Introduction to groupware issues



What Is CSCW?

- Computer Supported C----- Work
 - Cooperative
 - Collaborative
 - Competitive
- Design and evaluation of new technologies to support social work processes
- Fusion of sociology and computing
- Creation of groupware systems

CSCW Space-Time Matrix

	Same Time	Different Times
Same Place	face to face (classes, meetings)	asynchronous interaction (post-it notes, Moosburg)
Different Places	sync distributed (shared editors)	asynchronous distributed (email, listservs)

Asynchronous Distributed (Different Time, Different Place)

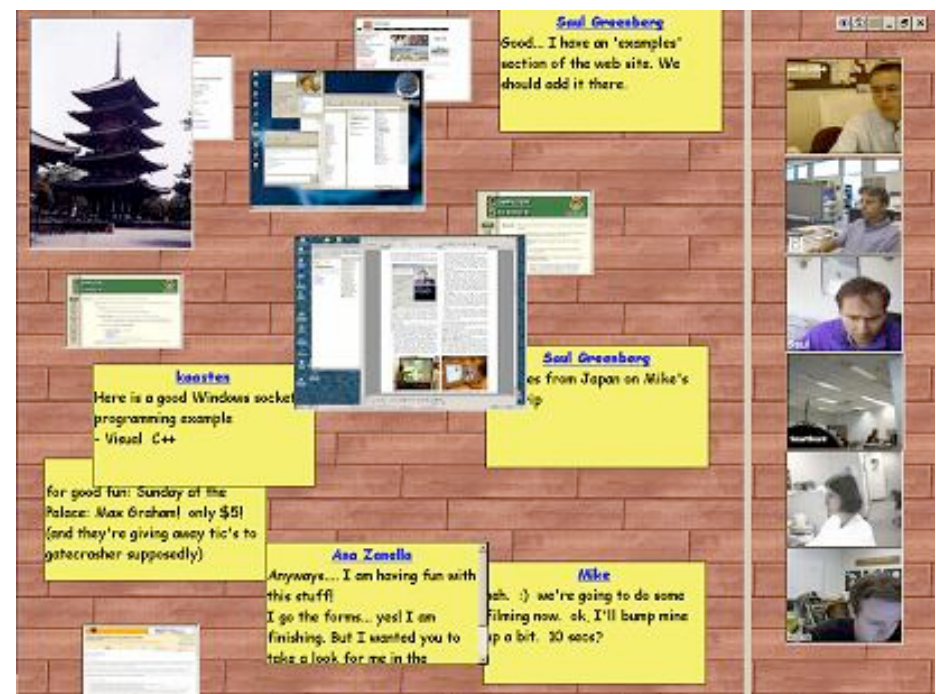
- Email
 - Delivery occurs in seconds or minutes
 - Multimedia becoming widespread
- Listservs
 - Leverage email to send to many people
- Newsgroups
 - Alternate forum facilitates longer (threaded) discussions, larger amounts of information

Asynchronous Interaction (Different Time, Same Place)

- Computer artifacts
 - Understand what previous users did or will do
 - Foster informal discussions via artifacts
 - Post-it notes, bathroom walls, room schedules
 - [Notification Collage](#)
- Place as a metaphor
 - Create and leverage virtual place much like physical place
 - Moosburg and other virtual worlds

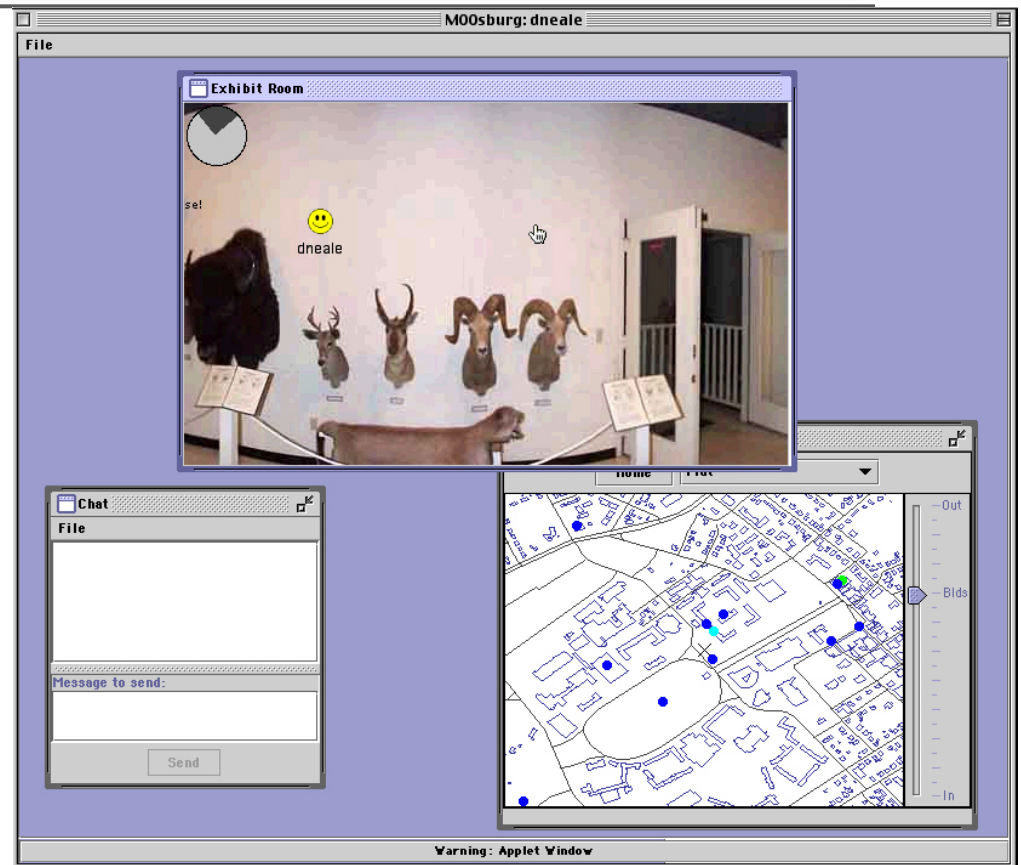
Notification Collage

- Users can post notes and pictures to NC
- Information appears on a large-screen display in a shared area
- Goal is to foster community and future discussions
- grouplab.cpsc.ucalgary.ca



Moosburg

- Synchronous chat augmented by ability to leave comments
- Future visitors can see who has been at a place and what their reactions were

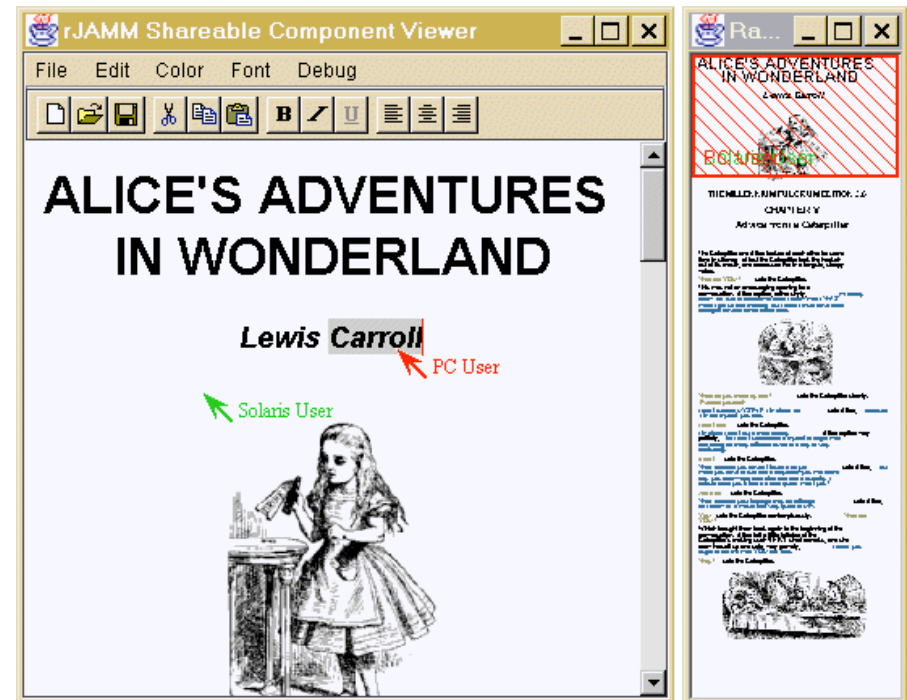


Synchronous Distributed (Same Time, Different Place)

- Initial interfaces included telephone and television
- Networked computers facilitate distributed applications that update in real-time
- Must address issues of ownership and control of information

Shared Editors

- Collaboration-aware shared editors
- Provide several insertion points
- Show who is editing document and where
- Include locking protocols or turn-taking
- Assignment 5

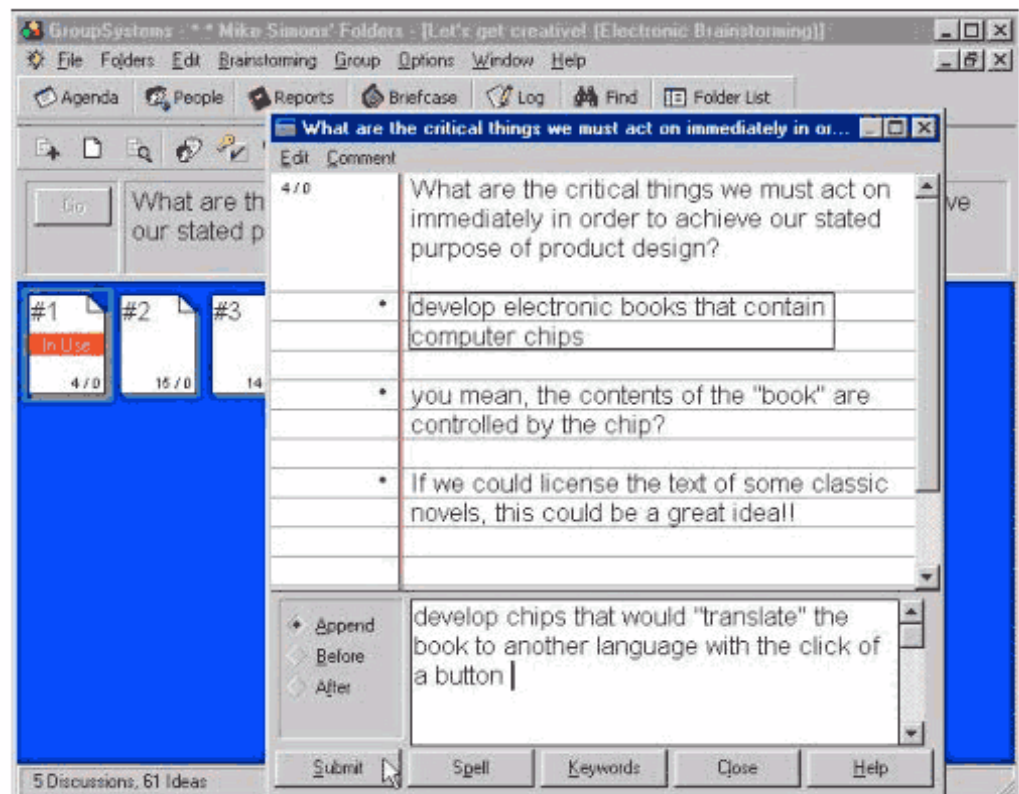


Face To Face (Same Time, Same Place)

- Teams often work together using complex shared technology
 - Pilots, co-pilots
 - Air traffic controllers
 - Stock market trading rooms
 - Offices and classrooms

GroupSystems

- Facilitates file sharing, shared workspace, and group activities
- Users can submit information, take surveys, or present ideas



Evaluating Groupware

- Complexities of human-human communication and group working make evaluation difficult
- Finding large numbers of groups is difficult, leading to smaller numbers and less chance for significant results

Experimental Studies

- Subject groups
 - Requires more subjects for longer times
 - Times and places often difficult to arrange
- Experimental task
 - Decision games test creativity, interaction, structure, leadership (desert survival, diplomacy)
- Data gathering
 - Logging not sufficient as human-human interaction is important
 - Multiple video shots often used to focus on individuals
- Analysis
 - Anecdotal analysis common (field/ethnographic studies)
 - Within-subject analysis, microanalysis common

Why Does Groupware Fail?

- Disparity between who does work and who gets benefit
- Threats to existing power structures
- Insufficient critical mass of users
- Violation of social taboos
- Arguments over measures of success

Why CSCW Applications Fail

- Consider group calendar scheduling
 - What are its advantages?
 - What are its disadvantages
 - Who does it support?

Why CSCW Applications Fail

- Consider voicemail
 - What are its advantages?
 - What are its disadvantages
 - Who does it support?

Why CSCW Applications Fail

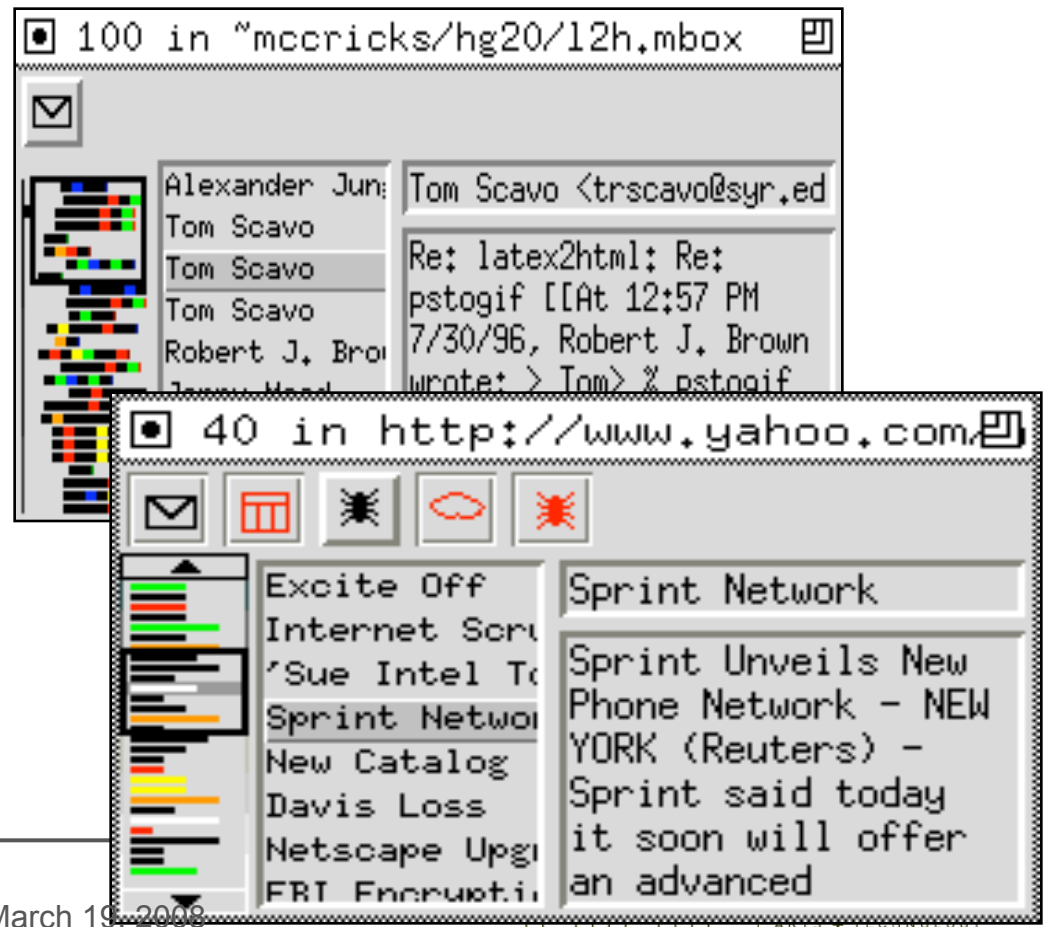
- Consider active badges
 - What are their advantages?
 - What are their disadvantages?
 - Who do they support?

Why Does Groupware Fail?

- Disparity between who does work and who gets benefit
- Threats to existing power structures
- Insufficient critical mass of users
- Violation of social taboos
- Arguments over measures of success

Case Study: Irwin

- Monitors Internet resources (email, Usenet news, Web pages, weather)
- Uses graphical, textual, and audio communication mechanisms
- Effective use of limited screen space

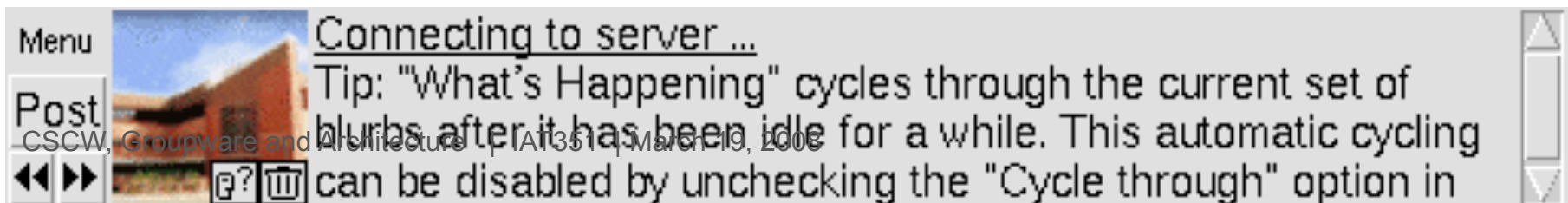
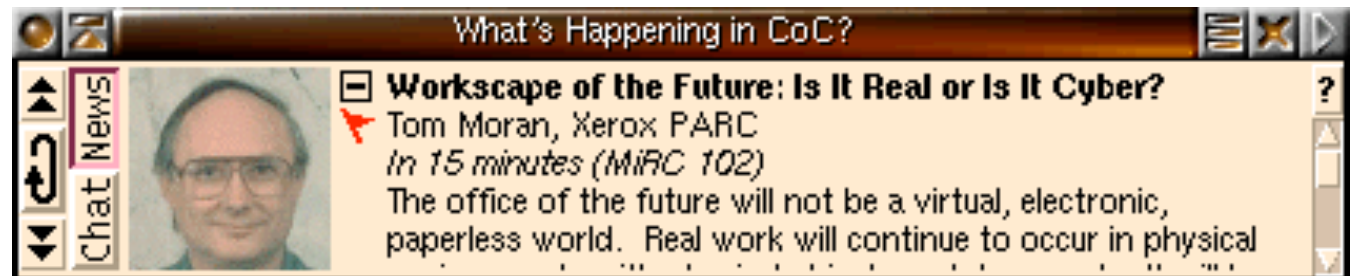


Positives from Irwin

- People seem receptive to concept of information awareness devices
- Significant variation among users for informational goals and communication mechanisms
- Examine the power of a single display mechanism in raising awareness

Case Study: What's Happening

- Goal: promote community awareness and expand social capital with glance-ables
- WH communication-bar unobtrusively and calmly sits in a corner cycling through news and chats
- Content from:
 - Users
 - Calendars
 - Web pages



WH Lessons

- At most 30 users in a community of 300
- Usage required frequent reminders and urging by the developer
- Occasional bursts of use followed by extended passive observation
- Chatting rare but occasionally animated, typically in response to an article

Case Study: Sideshow

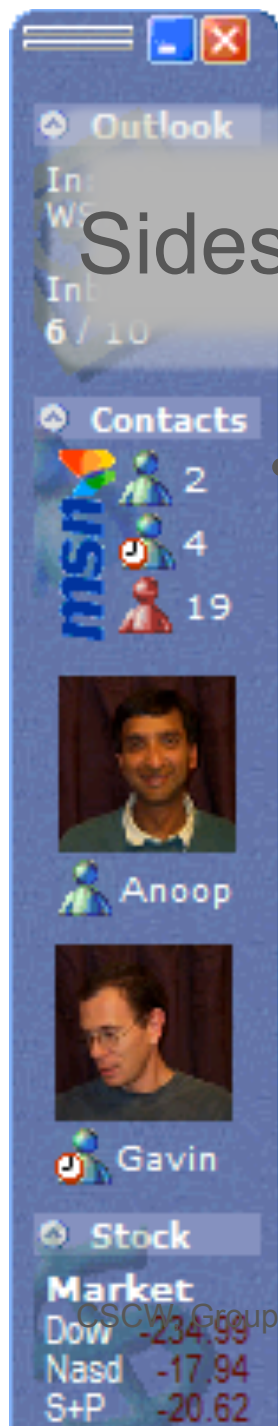
- Difficult to keep track of asynchronous, distributed info
- Sideshow is “a peripheral awareness interface designed to help users stay aware of people and info”
- Design principles
 - Make it always present
 - Minimize motion
 - Make it personal
 - Make it extensible
 - Support quick drill-down and escape
 - Make it scalable (up to dozens of items)



Sideshow Components

- Included are:
 - Meeting timers
 - Mailbox updates
 - Buddy lists
 - Video chat lists
 - Stock quotes
 - Bug reports
 - Weather forecast
 - Traffic pics/maps
- Mouseovers show “tooltip grandes” (large interactive tooltips)
- Tickets on Web pages allow content to be added
- Internal study: Sideshow distracting but worth it (?)

■ Ticket button



What Is the Biggest CSCW Success Story? What's Next?

- Email
- Instant messages
- World Wide Web
- Mobile telephones
- Video conferencing
- Irwin? WH? Sideshow?

Groupware

- CSCW applications and the tools used to build them are often referred to generically as *groupware*
- Groupware refers to software products that support groups of people who share a common task or goal and who collaborate to accomplish it.
- Teleconferencing is the use of electronic communication that allows two or more people at different locations to hold a simultaneous conference.

Groupware

- When/why is it used?
 - news, notification, broadcast
 - help systems
 - instant contact
 - work groups, physical networks
 - special interests, communities of practice

Groupware

- Long history of groupware beginning with the advent of user level networking
 - Bulletin boards, forums
 - Email!
 - Messaging (IM, IRC) and notifications
 - Team collaboration systems (Lotus Notes™, Sharepoint™, Groove™)
 - Social networks
 - MMOG (World of Warcraft™), virtual worlds (SecondLife™)
 - Blogs, albums, shared editors

What Groupware Functionality do Users Really Use? [Appelt 2001]

- BSCW (Basic Support for Cooperative Work)
 - Web based groupware system
- Central metaphor: *shared workspace*
 - Contents represented as information objects in folders
 - 150 features, incl utilities: search, format conversion, version mgmt, language support, event services
- analysis of usage, based on logfile (smart!)
 - Noone used *all* functions
 - Mostly browse & read (as expected)

What Groupware Functionality do Users Really Use? (cont.)

Popular Operations

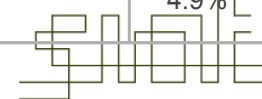
11.4%	Getting meta-information on objects
11.2%	Creation (upload) of documents
7.6%	Reading information about events
5.7%	Creation of folders
5.3%	Confirmation of events
5.1%	Modification of personal preferences
4.7%	Invitation of users to workspaces
4.4%	Modification of meta-information
3.8%	Creation of discussions or notes
3.8%	Display of sub-folders or threads

Popular Operation Categories

24.7%	Creation of information
12.4%	Modification of information
12.3%	Presentation of information
13.4%	Awareness features
12.3%	Reading information on objects
4.7%	Moving and deleting objects
8.2%	Access rights features
1.5%	Searching
1.6%	Personal information
0.6%	Meeting objects
2.8%	Java applets
5.1%	Other features

What Groupware Functionality do Users Really Use? (cont.)

	Group of operations	<i>Frequent users</i>	<i>Other users</i>
1	Creation of information	23.1%	25.6%
2	Modification of information	15.1%	11.0%
3	Presentation of information	8.6%	14.2%
4	Awareness features	17.3%	11.6%
5	Reading of information about objects	11.3%	12.9%
6	Moving and deleting objects	4.8%	4.7%
7	Access rights features	7.4%	8.7%
8	Searching	2.3%	1.1%
9	Personal information	0.9%	2.0%
10	Operations related to meeting objects	0.5%	0.6%
11	Java applets	2.7%	2.9%
12	Other features	5.6%	4.9%



What Groupware Functionality do Users Really Use? (cont.)

	Group of operations	Frequent users	Other users
1	Creation of information	23.1%	25.6%
2	Modification of information	15.1%	11.0%
3	Presentation of information	8.6%	14.2%
4	Awareness features	17.3%	11.6%
5	Reading of information about objects	11.3%	12.9%
6	Moving and deleting objects	4.8%	4.7%
7	Access rights features	7.4%	8.7%
8	Searching	2.3%	1.1%
9	Personal information	0.9%	2.0%
10	Operations related to meeting objects	0.5%	0.6%
11	Java applets	2.7%	2.9%
12	Other features	5.6%	4.9%

So how do we go about building this?

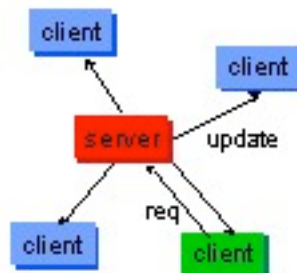
- Many approaches at the end-user application level
- Based on one of a few standard architectures
 - Server-client
 - Peer-to-peer
- Need a network substrate
 - TCP/IP
- And a middleware layer
 - Web services
 - Protocols (http, etc)

So how do we go about building this?

SICS

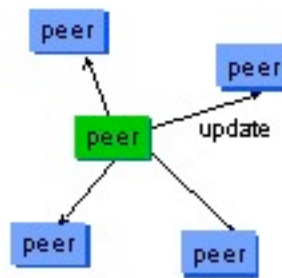
Client-Server vs. Peer-to-Peer

Client-Server



- Server manages data
- Easy to maintain consistency
- Clients can cache data for performance

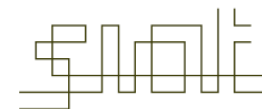
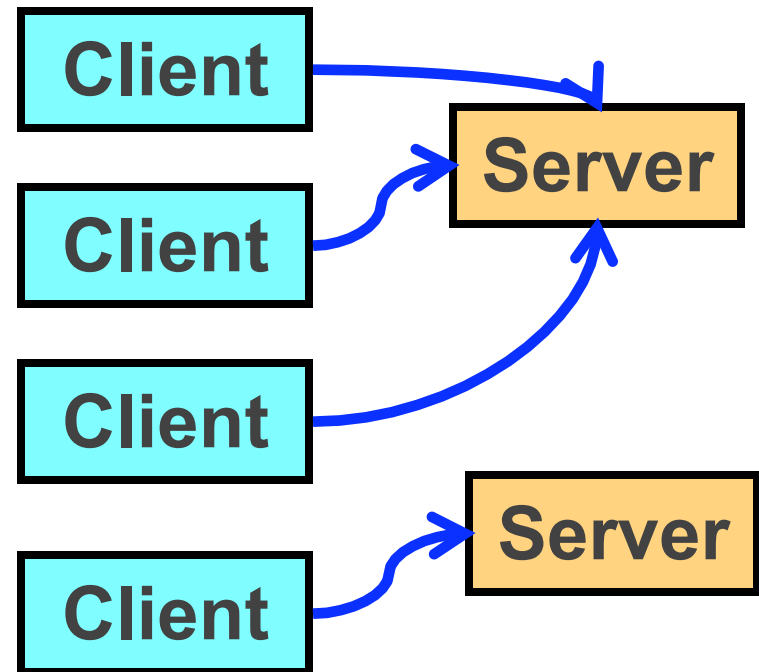
Peer-to-Peer



- Clients replicate data
- Low latency
- Set up anywhere

Client / Server Model

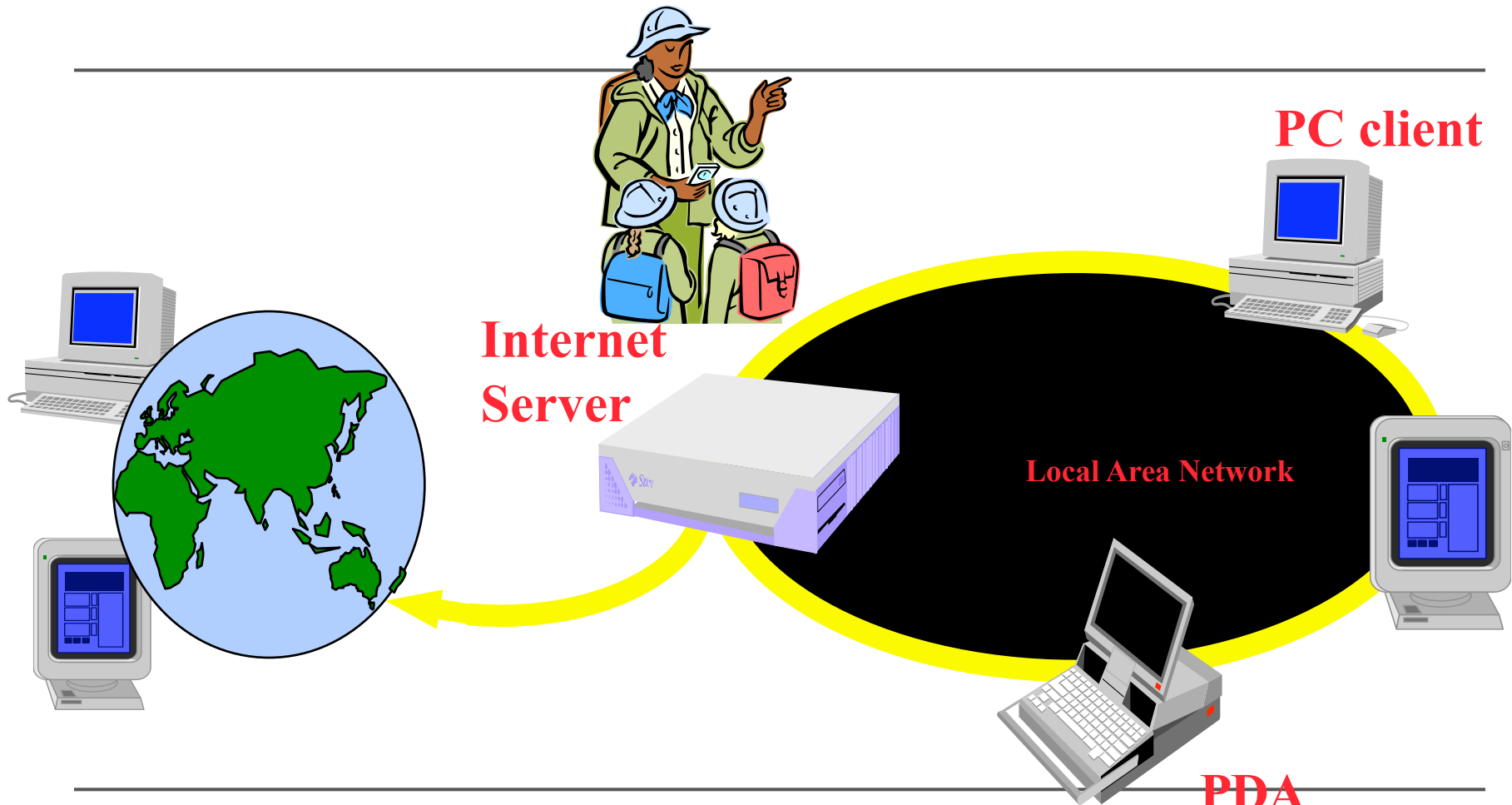
- Relationship between two computer programs
- Client
 - Initiates communication
 - Requests services
- Server
 - Receives communication
 - Provides services
- Other models
 - Master / worker
 - Peer-to-peer (P2P)



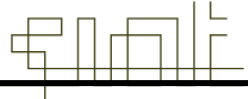
Server-client architecture: overview

- Networking Basics
- Understanding Ports and Sockets
- Java Sockets
 - Implementing a Server
 - Implementing a Client
- Sample Example

Internet Applications Serving Local and Remote Users



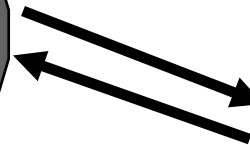
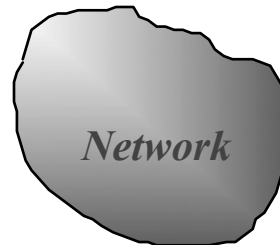
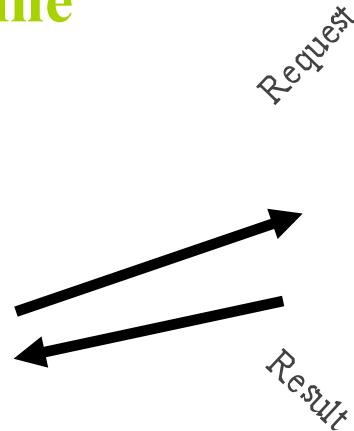
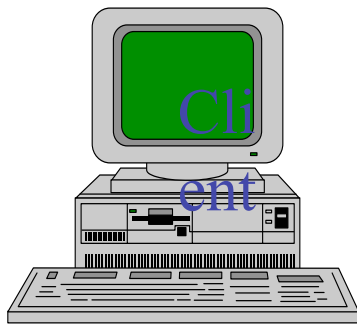
Client / Server Model Examples

Application	Client	Server
Web Browsing	Internet Explorer, Mozilla Firefox	Apache
Email	MS Outlook, Thunderbird	POP, IMAP, SMTP, Exchange
Streaming Music	Windows Media Player, iTunes	Internet Radio
Online Gaming	Half-Life, Everquest, PartyPoker	Game / Realm Servers
		 SCHOOL OF INTERACTIVE ARTS + TECHNOLOGY

Elements of C-S Computing

a client, a server, and network

Client machine



Server machine

- Processes follow **protocol** that defined a set of rules that must be observed by participants:
 - How the data is exchange is encoded?
 - How are events (sending, receiving) are synchronized (ordered) so that participants can send and receive in a coordinated manner?
- Face-to-face communication, humans beings follow unspoken protocol based on eye contact, body language, gesture.

Client Programming

- Basic steps
 1. Determine server location – IP address & port
 2. Open network connection to server
 3. Write data to server (request)
 4. Read data from server (response)
 5. Close network connection
 6. Stop client



Server Programming

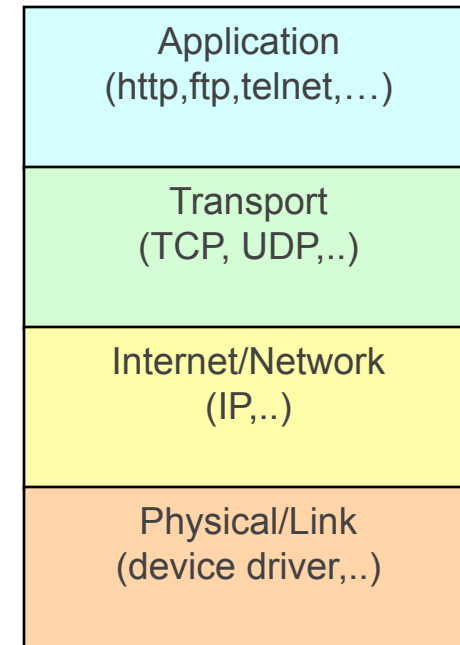
- Basic steps
 1. Determine server location - port (& IP address)
 2. Create server to listen for connections
 3. Open network connection to client
 4. Read data from client (request)
 5. Write data to client (response)
 6. Close network connection to client
 7. Stop server



Networking Basics

- Physical/Link Layer
 - Functionality for the transmission of signals, representing a stream of data from one computer to another.
- Internet/Network Layer
 - IP (Internet Protocols) – a packet of data to be addressed to a remote computer and delivered.
- Transport Layer
 - Functionalities for delivering data packets to a specific process on a remote computer.
 - TCP (Transmission Control Protocol)
 - UDP (User Datagram Protocol)
 - Programming Interface:
 - Sockets
- Applications Layer
 - Message exchange between standard or user applications:
 - HTTP, FTP, Telnet

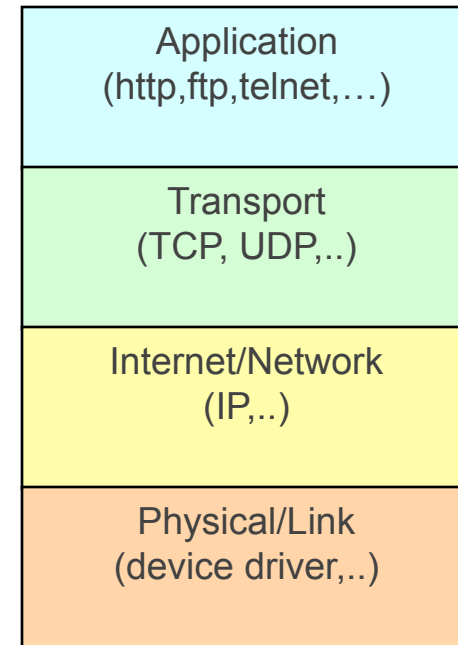
- TCP/IP Stack



Networking Basics

- TCP (Transmission Control Protocol) is a **connection-oriented** communication protocol that provides a reliable flow of data between two computers.
- Example applications:
 - HTTP
 - FTP
 - Telnet

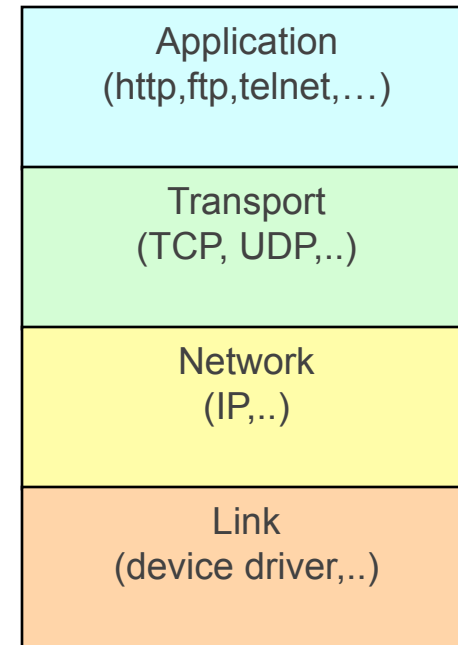
- TCP/IP Stack



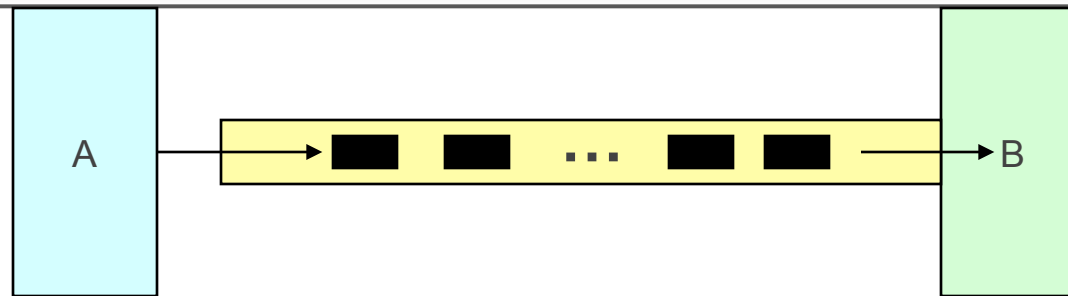
Networking Basics

- UDP (User Datagram Protocol) is a **connectionless communication** protocol that sends independent packets of data, called *datagrams*, from one computer to another with no guarantees about arrival or order of arrival.
- Similar to sending multiple emails/letters to a friends, each containing part of a message.
- Example applications:
 - Clock server
 - Ping

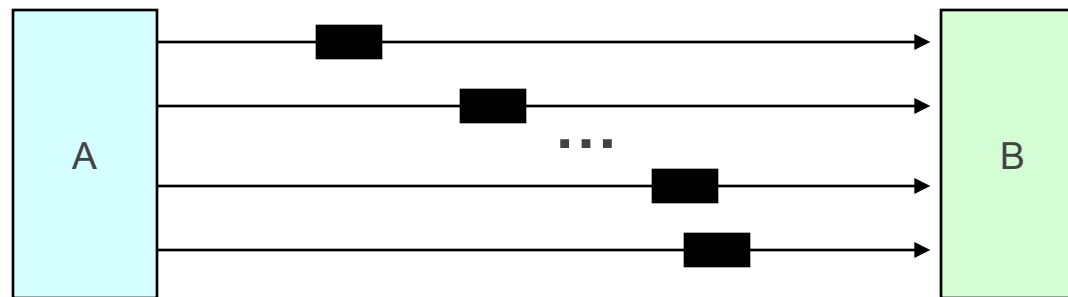
- TCP/IP Stack



TCP Vs UDP Communication



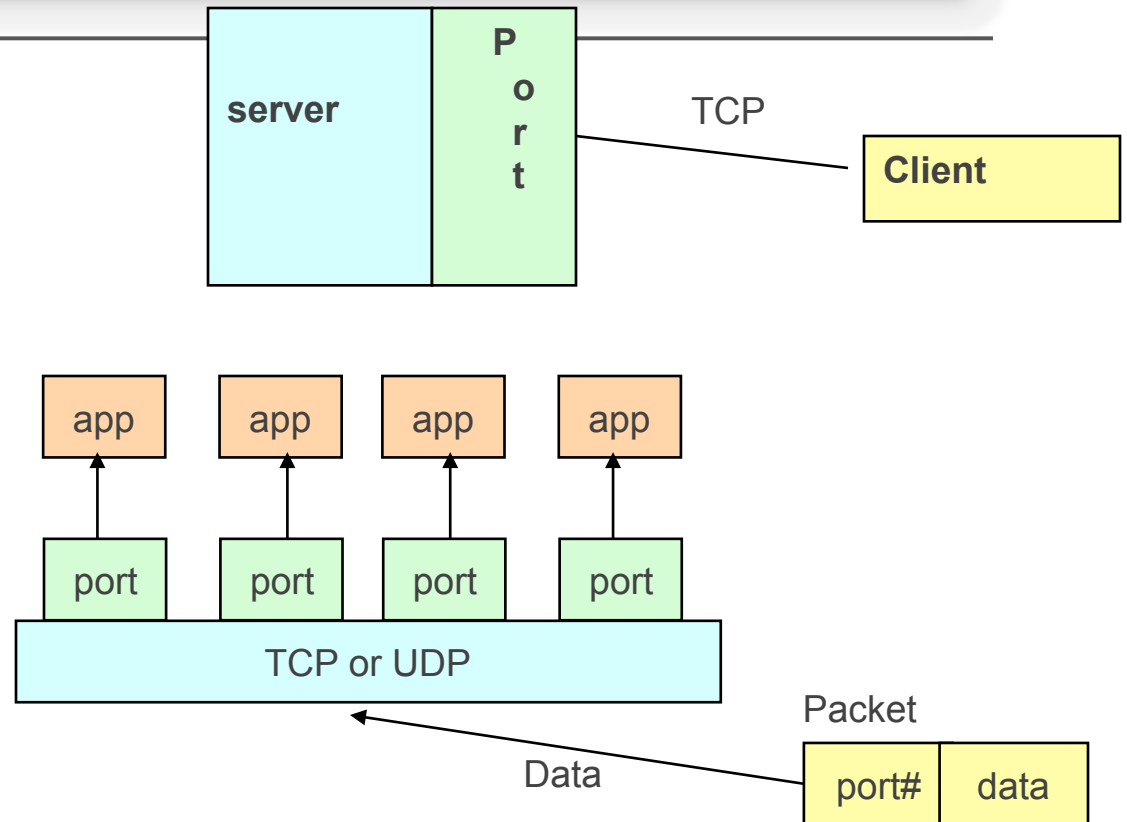
Connection-Oriented Communication



Connectionless Communication

Understanding Ports

- The TCP and UDP protocols use *ports* to map incoming data to a particular *process* running on a computer.



Understanding Ports

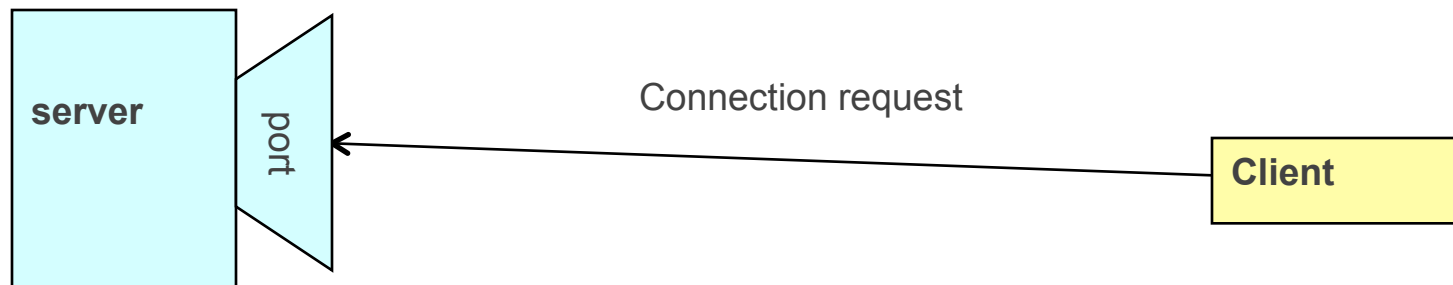
- Port is represented by a positive (16-bit) integer value
- Some ports have been reserved to support common/well known services:
 - ftp 21/tcp
 - telnet 23/tcp
 - smtp 25/tcp
 - login 513/tcp
- User level process/services generally use port number value ≥ 1024

Sockets

- Sockets provide an interface for programming networks at the transport layer.
- Network communication using Sockets is very much similar to performing file I/O
 - In fact, socket handle is treated like file handle.
 - The streams used in file I/O operation are also applicable to socket-based I/O
- Socket-based communication is programming language independent.
 - That means, a socket program written in Java can also communicate to a program written in Java or a non-Java socket program.

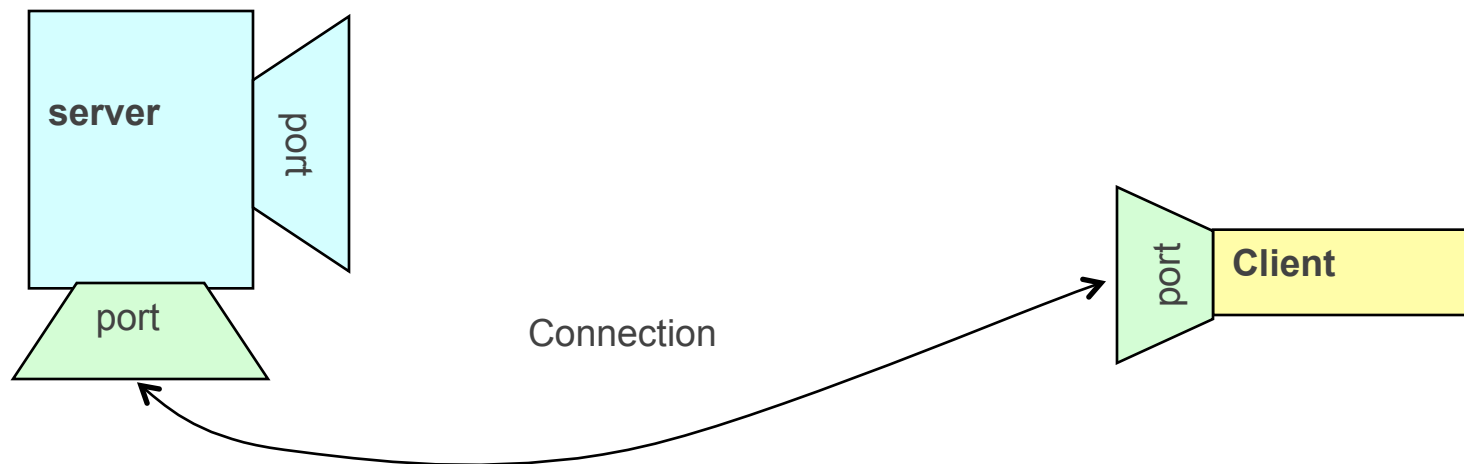
Socket Communication

- A server (program) runs on a specific computer and has a socket that is bound to a specific port. The server waits and listens to the socket for a client to make a connection request.



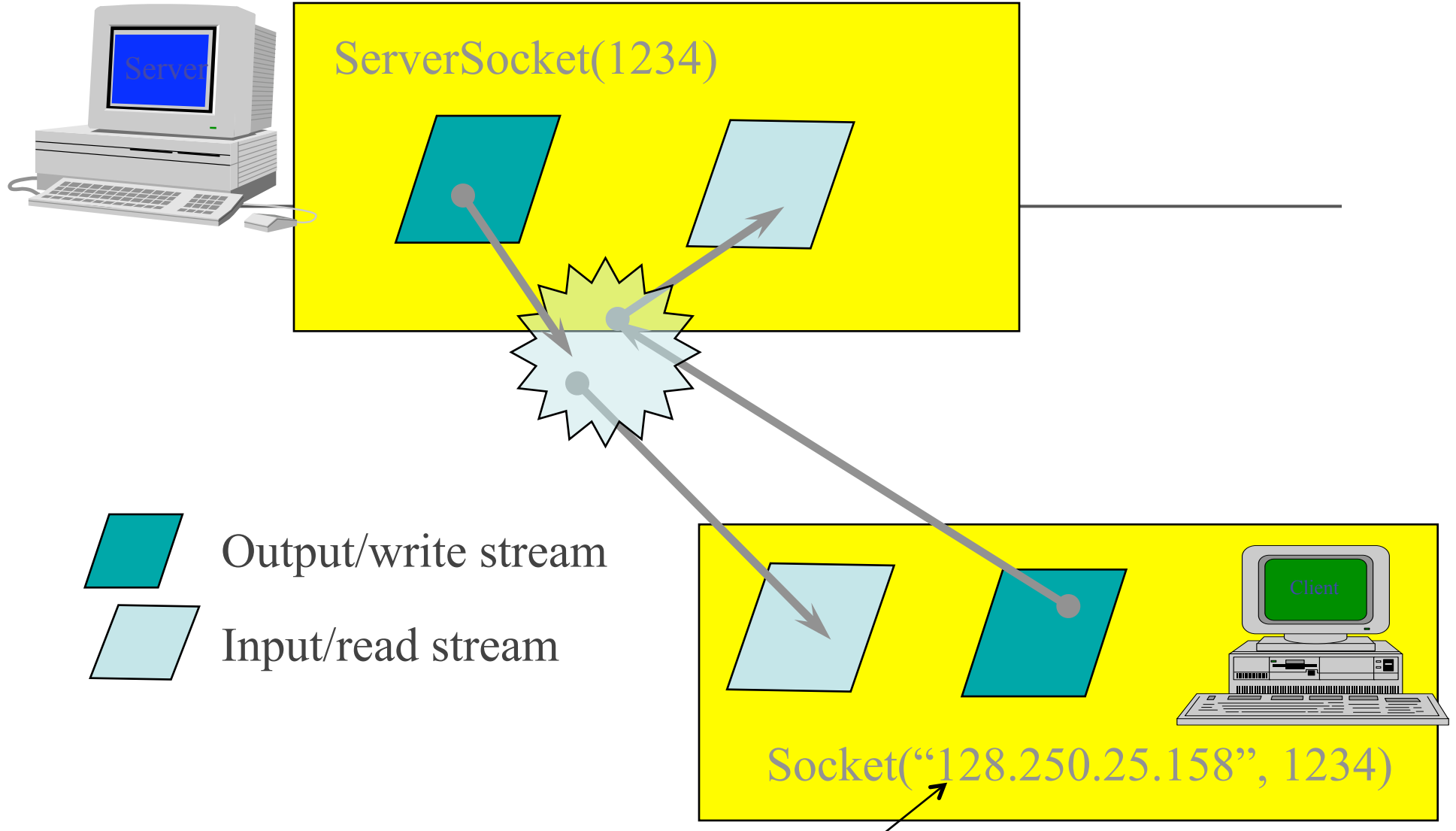
Socket Communication

- If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bounds to a different port. It needs a new socket (consequently a different port number) so that it can continue to listen to the original socket for connection requests while serving the connected client.



Sockets and Java Socket Classes

- A socket is an endpoint of a two-way communication link between two programs running on the network.
- A socket is bound to a port number so that the TCP layer can identify the application that data destined to be sent.
- Java's .net package provides two classes:
 - Socket – for implementing a client
 - ServerSocket – for implementing a server



— It can be a host_name like “lynmac.iat.sfu.ca”

Networking in Java

- Packages
 - java.net ⇒ Networking
 - java.io ⇒ I/O streams & utilities
 - java.rmi ⇒ Remote Method Invocation
 - java.security ⇒ Security policies
 - java.lang ⇒ Threading classes
- Support at multiple levels
 - Data transport ⇒ Socket classes
 - Network services ⇒ URL classes
 - Utilities & security



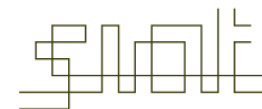
Java Networking API

- Application Program Interface
 - Set of routines, protocols, tools
 - For building software applications
- Java networking API
 - Helps build network applications
 - Interfaces to sockets, network resources
 - Code implementing useful functionality
 - Includes classes for
 - Sockets
 - URLs



Java Networking Classes

- IP addresses
 - InetAddress
- Packets
 - DatagramPacket
- Sockets
 - Socket
 - ServerSocket
 - DatagramSocket
- URLs
 - URL



InetAddress Class

- Represents an IP address
- Can convert domain name to IP address
 - Performs DNS lookup
- Getting an InetAddress object
 - `getLocalHost()`
 - `getByName(String host)`
 - `getByAddress(byte[] addr)`



Implementing a Server

1. Open the Server Socket:

```
ServerSocket server;  
DataOutputStream os;  
DataInputStream is;  
server = new ServerSocket( PORT );
```

2. Wait for the Client Request:

```
Socket client = server.accept();
```

3. Create I/O streams for communicating to the client

```
is = new DataInputStream( client.getInputStream() );  
os = new DataOutputStream( client.getOutputStream() );
```

4. Perform communication with client

```
Receive from client: String line = is.readLine();  
Send to client: os.writeBytes("Hello\n");
```

5. Close sockets: client.close();

```
}
```



Implementing a Server

1. Open the Server Socket:

```
ServerSocket server;  
DataOutputStream os;  
DataInputStream is;  
server = new ServerSocket( PORT );
```

2. Wait for the Client Request:

```
Socket client = server.accept();
```

3. Create I/O streams for communicating to the client

```
is = new DataInputStream( client.getInputStream() );  
os = new DataOutputStream( client.getOutputStream() );
```

4. Perform communication with client

```
Receive from client: String line = is.readLine();  
Send to client: os.writeBytes("Hello\n");
```

5. Close sockets: client.close();

For multithreaded server:

```
while(true) {  
    i. wait for client requests (step 2 above)  
    ii. create a thread with "client" socket as parameter (the thread creates streams (as in step  
        (3) and does communication as stated in (4). Remove thread once service is provided.  
}
```



Implementing a Client

1. Create a Socket Object:

```
client = new Socket( server, port_id );
```

2. Create I/O streams for communicating with the server.

```
is = new DataInputStream(client.getInputStream() );  
os = new DataOutputStream( client.getOutputStream() );
```

3. Perform I/O or communication with the server:

- Receive data from the server:

```
String line = is.readLine();
```

- Send data to the server:

```
os.writeBytes("Hello\n");
```

4. Close the socket when done:

```
client.close();
```

A simple server (simplified code)

```
// SimpleServer.java: a simple server program
import java.net.*;
import java.io.*;
public class SimpleServer {
    public static void main(String args[]) throws IOException {
        // Register service on port 1234
        ServerSocket s = new ServerSocket(1234);
        Socket s1=s.accept(); // Wait and accept a connection
        // Get a communication stream associated with the socket
        OutputStream slout = s1.getOutputStream();
        DataOutputStream dos = new DataOutputStream (slout);
        // Send a string!
        dos.writeUTF("Hi there");
        // Close the connection, but not the server socket
        dos.close();
        slout.close();
        s1.close();
    }
}
```

A simple client (simplified code)

```
// SimpleClient.java: a simple client program
import java.net.*;
import java.io.*;
public class SimpleClient {
    public static void main(String args[]) throws IOException {
        // Open your connection to a server, at port 1234
        Socket s1 = new Socket("mundroo.cs.mu.oz.au",1234);
        // Get an input file handle from the socket and read the input
        InputStream s1In = s1.getInputStream();
        DataInputStream dis = new DataInputStream(s1In);
        String st = new String (dis.readUTF());
        System.out.println(st);
        // When done, just close the connection and exit
        dis.close();
        s1In.close();
        s1.close();
    }
}
```

Run

- Run Server on any machine
 - [mach1] java SimpleServer &
- Run Client on any machine (including mach1):
 - [mach1] java SimpleClient
Hi there
- If you run client when server is not up:
 - sockets [1:147] java SimpleClient
Exception in thread "main" java.net.ConnectException: Connection refused
at java.net.PlainSocketImpl.socketConnect(Native Method)
at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:320)
at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:133)
at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:120)
at java.net.Socket.<init>(Socket.java:273)
at java.net.Socket.<init>(Socket.java:100)
at SimpleClient.main(SimpleClient.java:6)

Socket Exceptions

```
try {  
    Socket client = new Socket(host, port); handleConnection(client);  
}  
catch(UnknownHostException uhe) { System.out.println("Unknown  
    host: " + host); uhe.printStackTrace();  
}  
catch(IOException ioe) {  
    System.out.println("IOException: " + ioe); ioe.printStackTrace();  
}
```

ServerSocket & Exceptions

- public **ServerSocket**(int port) throws [IOException](#)
 - Creates a server socket on a specified port.
 - A port of 0 creates a socket on any free port. You can use [getLocalPort\(\)](#) to identify the (assigned) port on which this socket is listening.
 - The maximum queue length for incoming connection indications (a request to connect) is set to 50. If a connection indication arrives when the queue is full, the connection is refused.
- Throws:
 - [IOException](#) - if an I/O error occurs when opening the socket.
 - [SecurityException](#) - if a security manager exists and its checkListen method doesn't allow the operation.

Server in Loop: Always up

// SimpleServerLoop.java: a simple server program that runs forever in a single thread

```
import java.net.*;
```

```
import java.io.*;
```

```
public class SimpleServerLoop {
```

```
    public static void main(String args[]) throws IOException {
```

```
        // Register service on port 1234
```

```
        ServerSocket s = new ServerSocket(1234);
```

```
        while(true)
```

```
        {
```

```
            Socket s1=s.accept(); // Wait and accept a connection
```

```
            // Get a communication stream associated with the socket
```

```
            OutputStream s1out = s1.getOutputStream();
```

```
            DataOutputStream dos = new DataOutputStream (s1out);
```

```
            // Send a string!
```

```
            dos.writeUTF("Hi there");
```

```
            // Close the connection, but not the server socket
```

```
            dos.close();
```

```
            s1out.close();
```

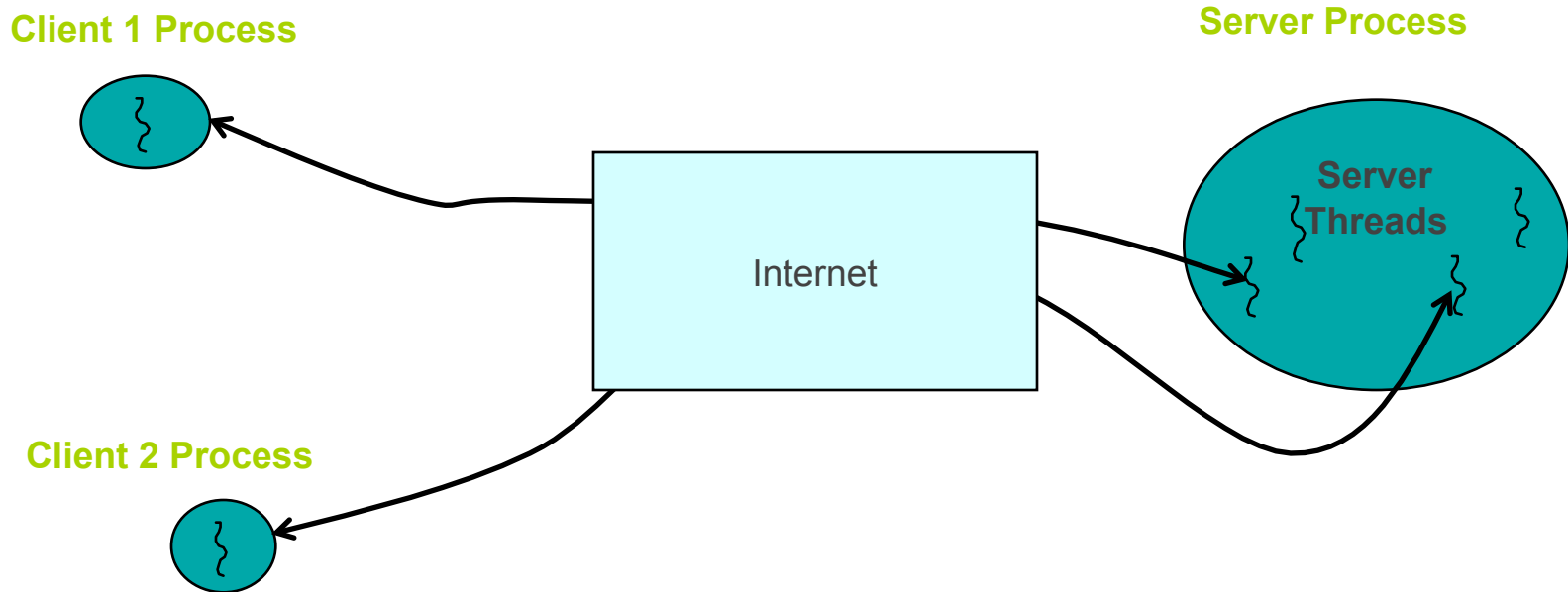
```
            s1.close();
```

```
        }
```

```
    }
```

```
}
```

Multithreaded Server: For Serving Multiple Clients Concurrently



More great examples

- Lots of good examples on the web
 - Hosted on web servers
- good example of 2-port server [/users.erols.com/ziring/TwoPortServer.java](http://users.erols.com/ziring/TwoPortServer.java)
- williams.comp.ncat.edu/Networks/JavaSocketExample.htm
- www.cs.utexas.edu/~chris/cs378/resources/sockets.html
- www.cs.unc.edu/Courses/jbs/lessons/java/java_client_server1/
- [Sun Java client-Server tutorial](http://Sun.com/java/client-server/tutorial)
- java.sun.com/docs/books/tutorial/networking/sockets/examples/KnockKnockServer.java



Summary

- Programming client/server applications in Java is fun and challenging.
- Programming socket programming in Java is much easier than doing it in other languages such as C.
- Keywords:
 - Clients, servers, TCP/IP, port number, sockets, Java sockets