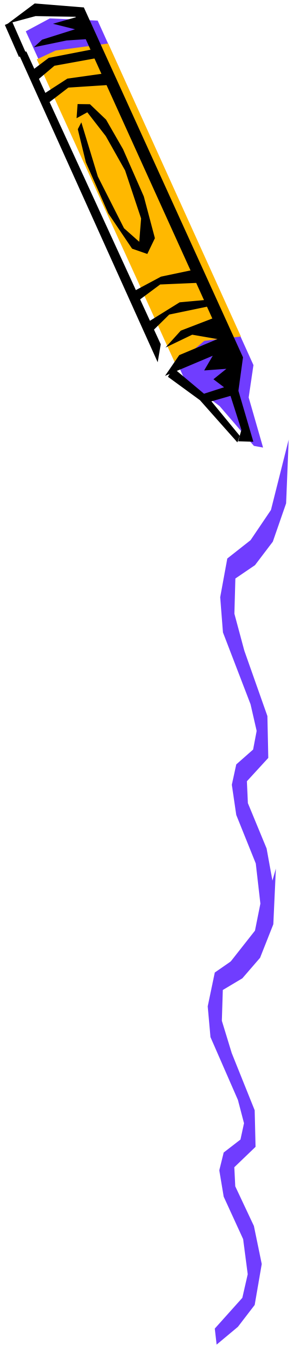
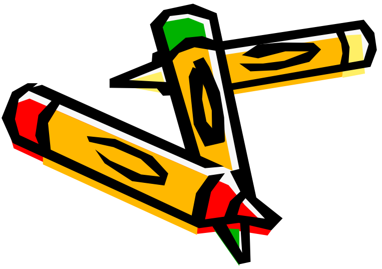
CSc 631 Network Issues for MultiPlayer Game Design

Networked Games: Doom

- id Software, 1993
- First-person shooter (FPS) for PCs
- Part of the game was released as shareware in 1993
- extremely popular
- created a gamut of variants
- Flooded LANs with packets at frame rate

Networking

- Data transfer
  - latency
  - bandwidth
  - reliability
  - protocol
- Internet protocols
  - TCP, UDP
  - unicast, broadcast, multicast
- Communication architectures
  - peer-to-peer
  - client-server

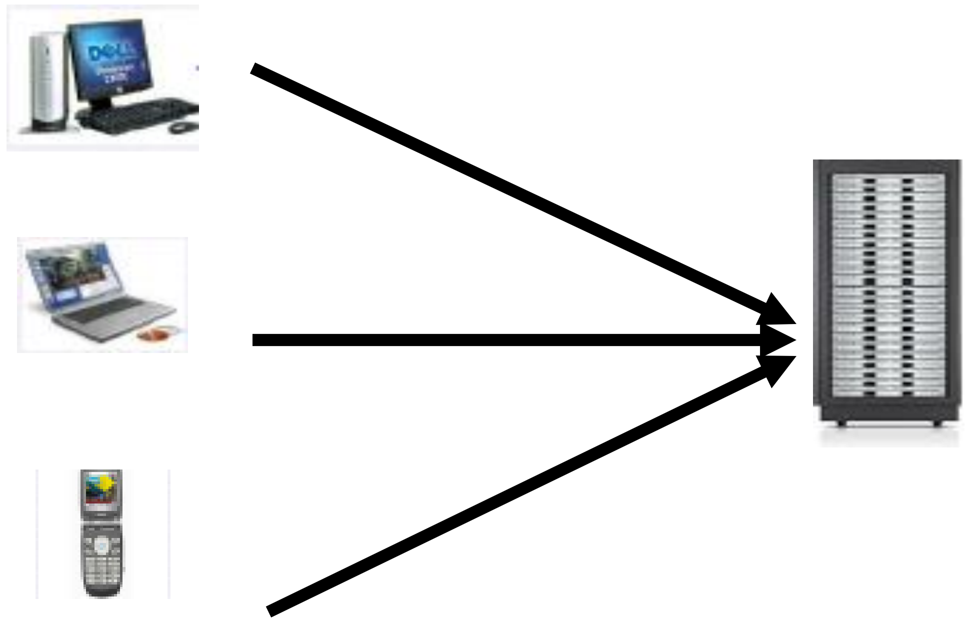



Server and Client

host:port

host:port

host:port



Clients make “calls” to that port #

Server listens on a port # host:port

Server and client can be the same machine!

$$\begin{tikzcd}\text{Client Process} \arrow[r] & \text{Server Process}\end{tikzcd}$$

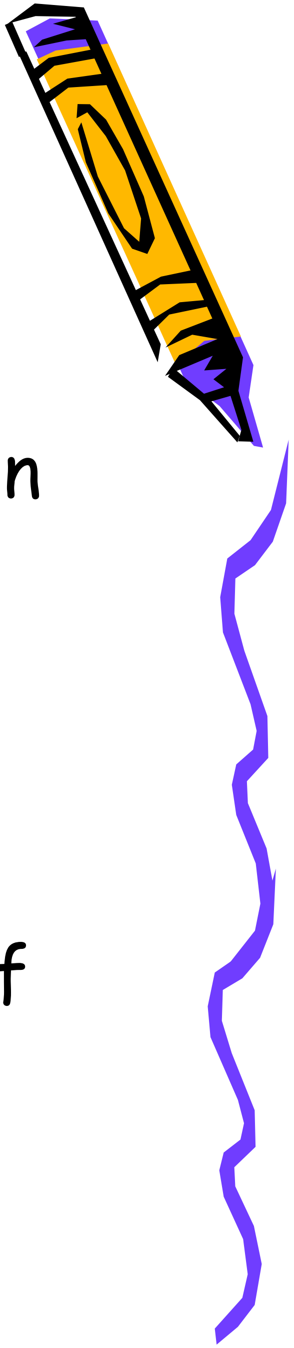
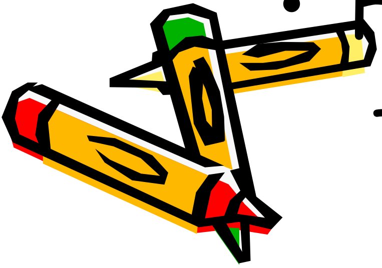
Socket Programming

Sockets are a protocol independent method of creating a connection between processes. Sockets can be either

- connection based or connectionless: Is a connection established before communication or does each packet describe the destination?
- packet based or streams based: Are there message boundaries or is it one stream?
- reliable or unreliable. Can messages be lost, duplicated, reordered, or corrupted?

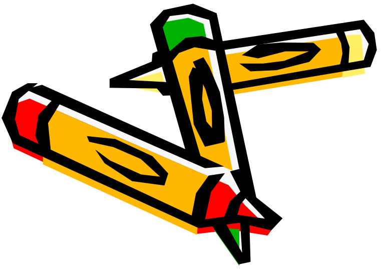
TCP

- Connection-Oriented
  - Port on “client” connects to port on “server”
- Reliable
  - 3-way handshake
- Byte-Stream
  - To application, looks like stream of bytes flowing between two hosts
- Flow Control
  - Prevents overrunning receiver / network capacity

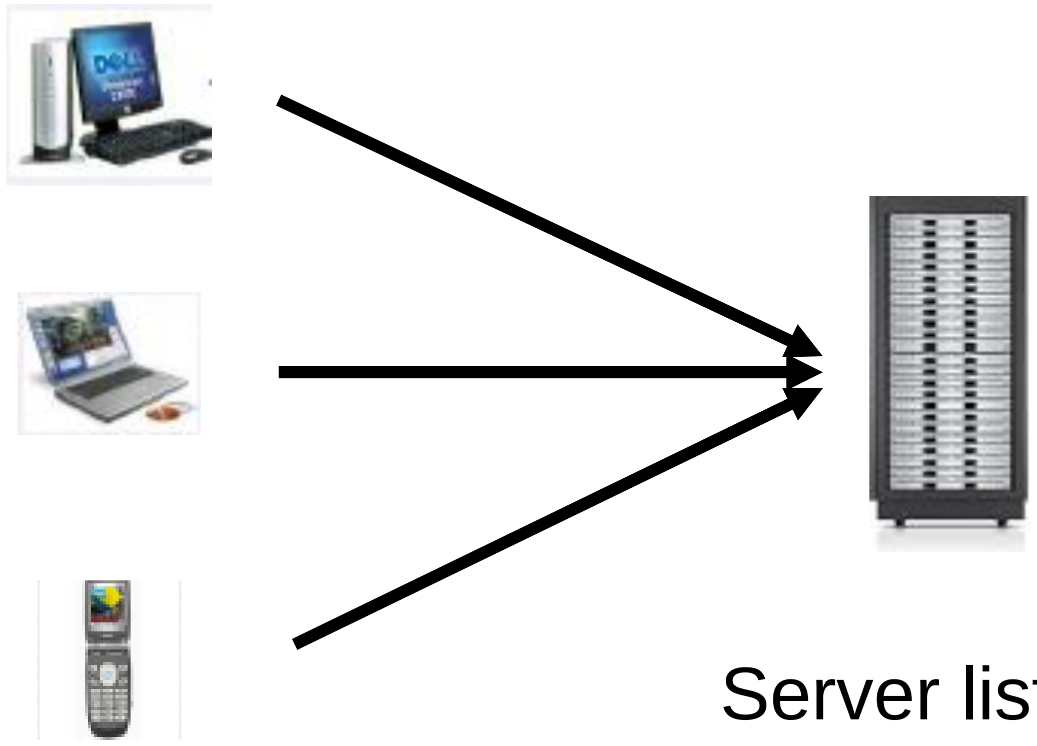



User Datagram Protocol (UDP)

- Characteristics
  - Connectionless, Datagram, Unreliable
- Good for Streaming Media, Real-time Multiplayer Networked Games, VoIP



Design of a Game Room Server/Client

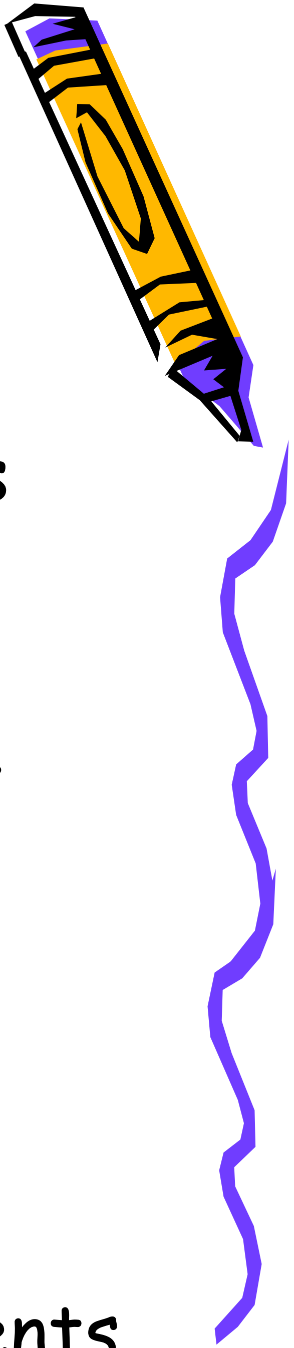
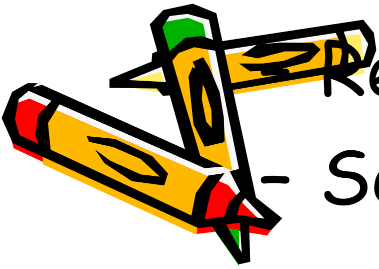


Server listens

Clients call

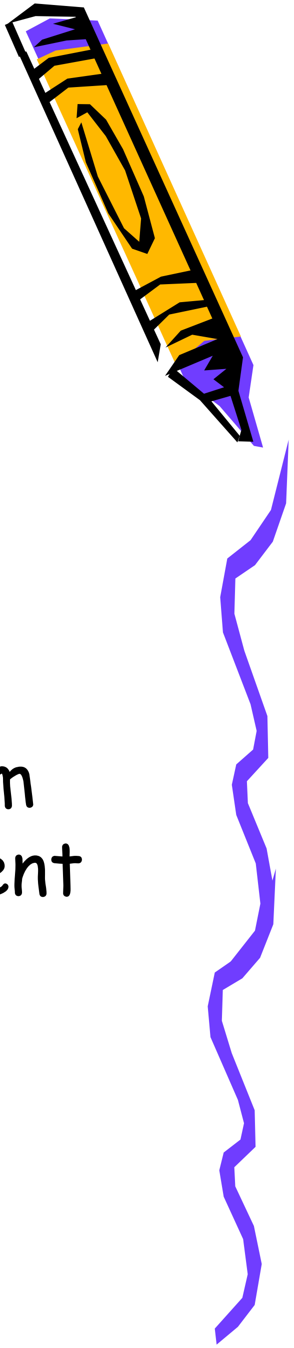
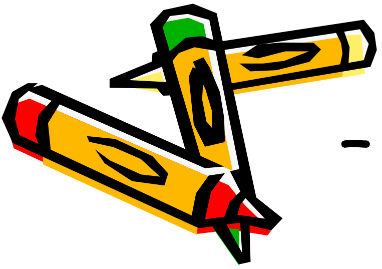
Game Server Design 1

- **Server State**
  - **Updated whenever a client connects**
  - Who is connected
  - Send out accumulated updates from each client to every connecting client
- **Server Abilities**
  - Accept a New Connection
  - Close a Connection
  - Receive an action from a client
  - Send the actions to all connected clients

Game Server Design 1

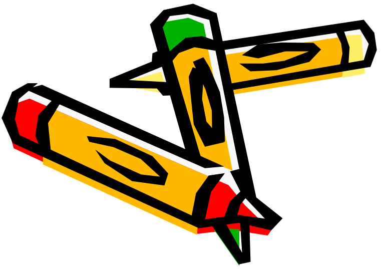
- **Client State**
  - **Updated asynchronously when a client connects to server**
  - Who is connected
  - Send out accumulated updates from each client to every connecting client
- **Client Abilities**
  - Receive a list of actions of other players from a server
  - Update the player's states in their own client copy

JAVA Example for Server

JAVA SERVER - singlethreaded

```
ServerSocket ding = null;
Socket dong = null;
try {
// ServerSocket, Socket are availalbe Java classes.
        ding = new ServerSocket(hcf.getListenPort());
        System.out.println("Opened socket " + hcf.getListenPort());

        while (true) { // keeps listening for new clients, one at a time
                try {
                dong = ding.accept(); // waits for client here
                }
                catch (IOException e) {
                System.out.println("Error opening socket");
                System.exit(1);
                }
                try {
        // Connection is built, so read stream from the socket
        // and parse request
                ReadHMsg(dong);
                } catch (Exception e){
                System.out.println("Error writing output"); }
```



Web Server - multithreaded

- A server class starts max number threads of request_handler that extends Thread class.

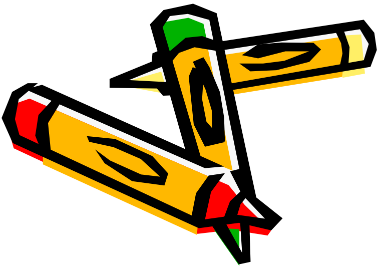
```
hconfig = new httpd_config("httpd.conf");
    if(!hconfig.isValid()){
    System.out.println("Configuration file not correct");
    return;}
  mime = new mime_config(hconfig.getValue("TypesConfig"));
  mime.setDefault(hconfig.getValue("DefaultType"));

  s = new server(hconfig,mime);
  if(s.isReady()){
    s.run();
  }
```

Server.run

```
for(i=0; i<maxthreads;i++){
     request_handler server_thread = new
           request_handler(ss,i,hc,mc,l);
     server_thread.start();
   }
```

Request_handler.get_socket

```
private synchronized Socket get_socket(){
    try{
      Socket incoming = ss.accept();
      return(incoming);
    }
    catch(IOException e){
      System.out.println(e);
    }
    return(null);
  }
```



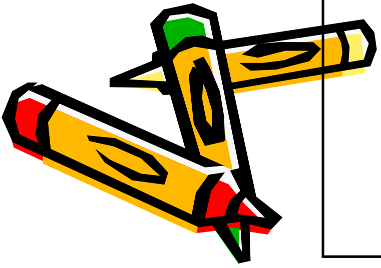
Web Server - multithread example II

- A main server class waits for connections and creates new thread when a new request arrives.
- The ServerHandler extends thread class.

```
while( true )
        {
            Socket socket = sSocket.accept();
            new ServerHandler( socket, numThreadsCreated,
                                    srmProp ).start();
        }
```

ServerHandler.run

```
BufferedReader in = new BufferedReader(
                    new InputStreamReader(
                        socket.getInputStream()));
request.readRequest(in);
response = new
HResponse(request,socket.getOutputStream());
socket.close();
```

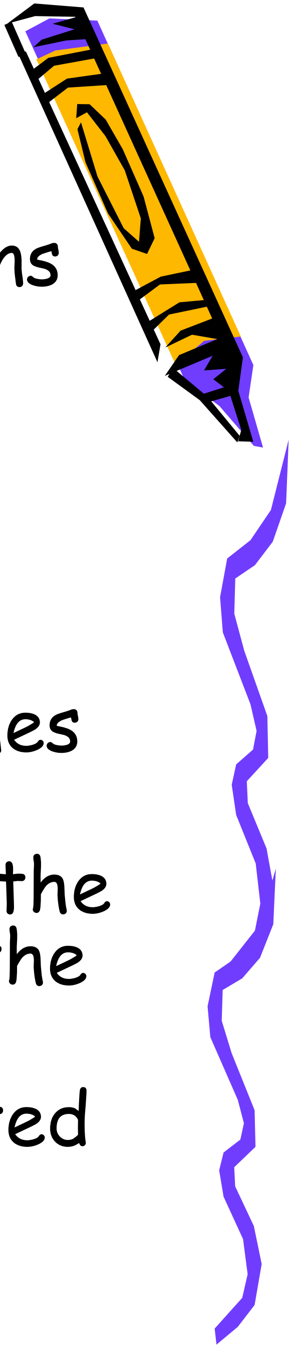
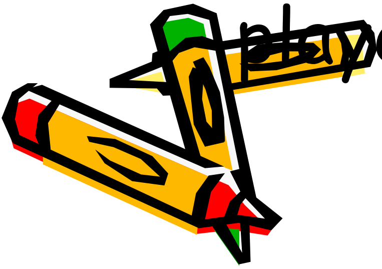


Request_handler.get_socket

```
private synchronized Socket get_socket(){
    try{
      Socket incoming = ss.accept();
      return(incoming);
    }
    catch(IOException e){
      System.out.println(e);
    }
    return(null);
  }
```

Extend web server to game server

- A main server class waits for connections and creates new thread when a new request arrives.
- The server maintains connections from clients and maintain status of all the connected clients.
- Each client updates server about 10 times per sec.
- Server updates the status and returns the status of connected clients who are in the same room.
- Clients updates status of other connected players on their side.

Our Game Protocol

“Endianness”

In computing, **endianness** is the byte (and sometimes bit) ordering used to represent some kind of data. Typical cases are th order in which integer values are stored as bytes in computer memory (relative to a given memory addressing scheme) and the transmission order over a network or other medium. When specifically talking about bytes, endianness is also referred to simply as **byte order**. [1] The usual contrast is between most significant byte (or bit) first, called **big-endian**, and least significant byte (or bit) first, called **little-endian**.

Game client: Panda3D – Little-endian
Game server: Java – Big-endian

Encoding

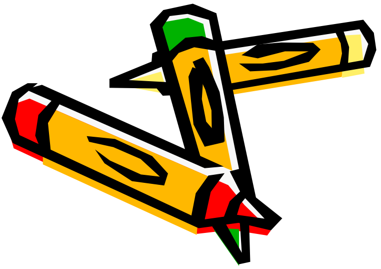
All data packages sending between client and server are in the format of “pydatagram” which is heavily used in the client-side programming. The protocol between client and server should strictly follow this format when sending data packages on the wire.

Generic Package Format:

[Total length of package] [Data1 encoded] [Data2 encoded]…
----->2 bytes Short<------
(fixed)

***Nurse Game C/S* Generic Package Format:**

[Total length of package] [Request or response ID] [Data1 encoded] [Data2 encoded]…
----->2 bytes Short<------ ----->2 bytes Short<------
(fixed) (fixed)



Requests (client to server in Pydatagram format)
# Some changes made to conform to the Python and Java communication
Implemented – Necessary functions
To be implemented – Need less client side support, doable

RequestLogin
Client requests to login with a username and password.
The server validates this and responds with ResponseAuth
	`Short Constants.CMSG_AUTH`
	`String Username`
	`String Password`

RequestHeartbeat
Client's state has not changed, but enough time has passed that the client would like an update from the server. Each client now is a thread and associated with a specific username, when request comes, server knows which client is sending the heartbeat, so no user id is required. The server will be able to check the client's queued response and send all of them out to the client socket.
	`Short Constants.REQ_HEARTBEAT`
	`String Greeting`

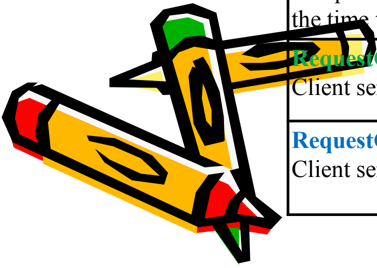
RequestLogout
Client wishes to log out from the game. No more requests are to be sent after this.
The server will respond with ResponseDisconnected, and server will update other users with ResponseRemoveUser.
	`Short Constants.CMSG_DISCONNECT_REQ`
	`String message`

RequestMove
Client issues a change to their location, and isMoving flag.
This is used when a client wishes to move or stop moving. It is followed by creating a number of ResponseMove and Server will update other users with these ResponseMove
	`Short Constants.CMSG_MOVE`
	`Float x //location vector`
	`Float y`
	`Float z`
	`Float h //facing direction`
	`boolean isMoving`

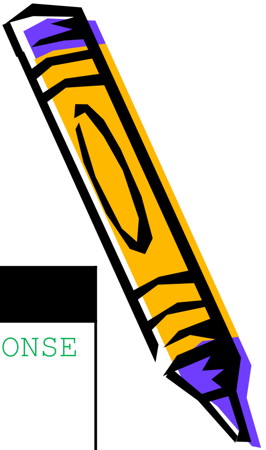
RequestChatPrivate
Client sends chat message to a single player. Message type can be used with a buddy list.
The private chat message will be delivered only if the "to" user is online at the time the update reaches them.
	`Short Constants.CMSG_PRIVATE_CHAT`
	`String receiverName`
	`String message`

RequestChatGlobal Client sends chat message to everyone currently online.	`Short Constants.REQ_CHAT_GLOBAL String message`

RequestChatTeam
Client sends chat message to every member of their team currently online.
	`int Constants.REQ_CHAT_TEAM`
	`long userId`
	`String message`



Responses (server to client in Pydatagram format)
# Some changes made to conform to the Python and Java communication
Implemented – Necessary functions
To be implemented – Need less client side support, doable



ResponseAuth
Affirmative response to RequestLogin.
It is followed by creating a number of ResponseCreate. All ResponseCreate will be queued up into all OTHER users' update queue.
	`Short Constants.SMSG_AUTH_RESPONSE`
	`Int flag (1 valid 2 invalid)`
	`String greeting`

ResponseCreate
Create one character in client game world represent one existing user who already logs in.
The new character in the world will be associated with username.
	`Short Constants.SMSG_CREATE`
	`String createCharacterName`
	`String greeting`

ResponseLogout Client sends out request to disconnect. Server replies and removes user from current list also inform other clients to remove the user. Followed by creating a number of ResponseRemoveUser. All ResponseRemoveUser will be queued up into all OTHER users' update queue.	`Short Constants.SMSG_DISCONNECT_ACK`

ResponseRemoveUser
One client disconnects. Server informs other clients to remove the user in their game worlds.
	`Short Constants.SMSG_REMOVE_USER`
	`String removeCharacterName`
	`String message`