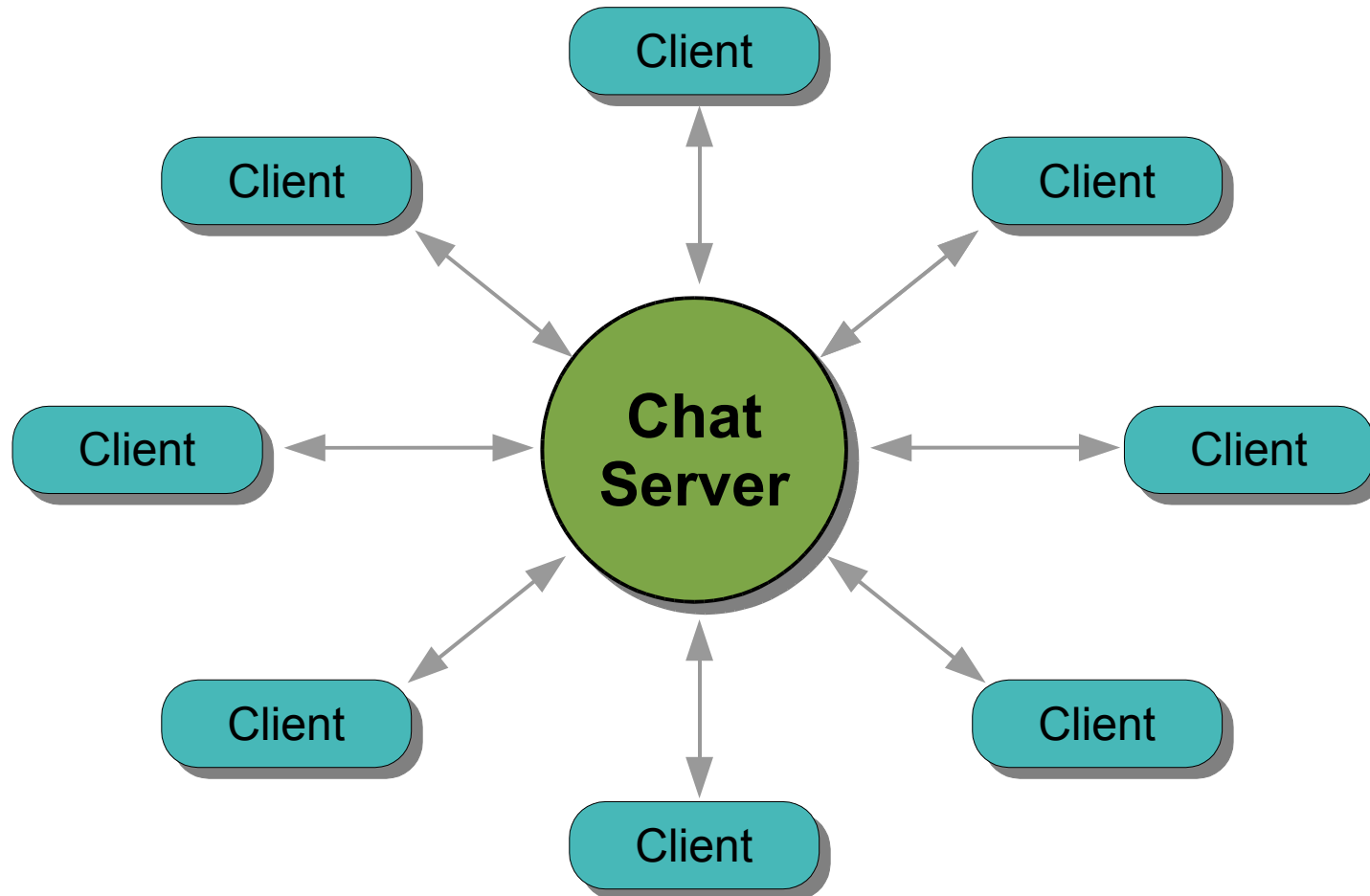


# HW5 – TCP Chat Server

- TCP server only (you don't need to write a client).
  - you can test with telnet
  - there is a minimal curses based client available.
- Very simple *application protocol*
  - lines of ASCII text
  - minimal commands (client->server).
- Support at least 10 clients.

# Chat Topology



# Application Protocol: client -> server

- Client can send:

`/login` username

`/help`

`/who`

`/priv` destuser message ...

broadcast message ...

each must be terminated by a newline '\n'

# Application Protocol: server -> client

- Server can send:

`ERR` some error text message...

`OK` some text ...

`MSG` a chat message ...

each must be terminated by a newline '\n'

# Formal Rules

- Most *real* application protocols include lots of rules about what can get sent when, and by whom, etc.
- For this homework, there is only one rule:
  - user cannot do anything else until a successful login.

# TCP Server Issue

- Server creates a passive TCP socket, prints out the port (so we know where to connect!).
- Server waits for connections, each new connection is a new client.
- Need to do the following (at the same time!):
  - look for new connections.
  - look for incoming messages from clients.

# Handling multiple clients

1 Server can use non-blocking I/O and poll all possible sources of input repeatedly.

busy waiting is bad...

2 Server can use I/O multiplexing support

system calls `select()` or `poll()`

3 Threads

one thread to watch each client socket

one thread to watch for new clients.

# Client Handling Threads

- Each thread handles all input from a single client.
- read a line of text from a socket.
- process the line:
  - might need to send chat message to other clients.
  - need access to all the socket descriptors.
  - need to make sure we don't have two threads messing with the list of socket descriptors at the same time.
  - Use a mutex!

# Testing

- Use telnet
  - telnet localhost 1234
  - what happens if you are in the middle of typing a command and the server sends a message?
- There is a curses based client available
  - minimal.
  - probably buggy (Dave's first curses program).
  - If you develop a better client (or a bot), feel free to share with others!

# Getting Started: Possible Approach

- Here is one approach:
  - global data structure that holds information about each active client:
    - socket descriptor
    - username
  - Exclusive access to the data structure (mutual exclusion).
  - main server waits for incoming connections.
    - updates data structure, and creates thread to handle new client.
  - each new thread waits for a line of text.
    - either sends reply, or sends chat message to other clients.
    - /quit means update the shared data structure.