Linux Instant Messenger

Chao Chen

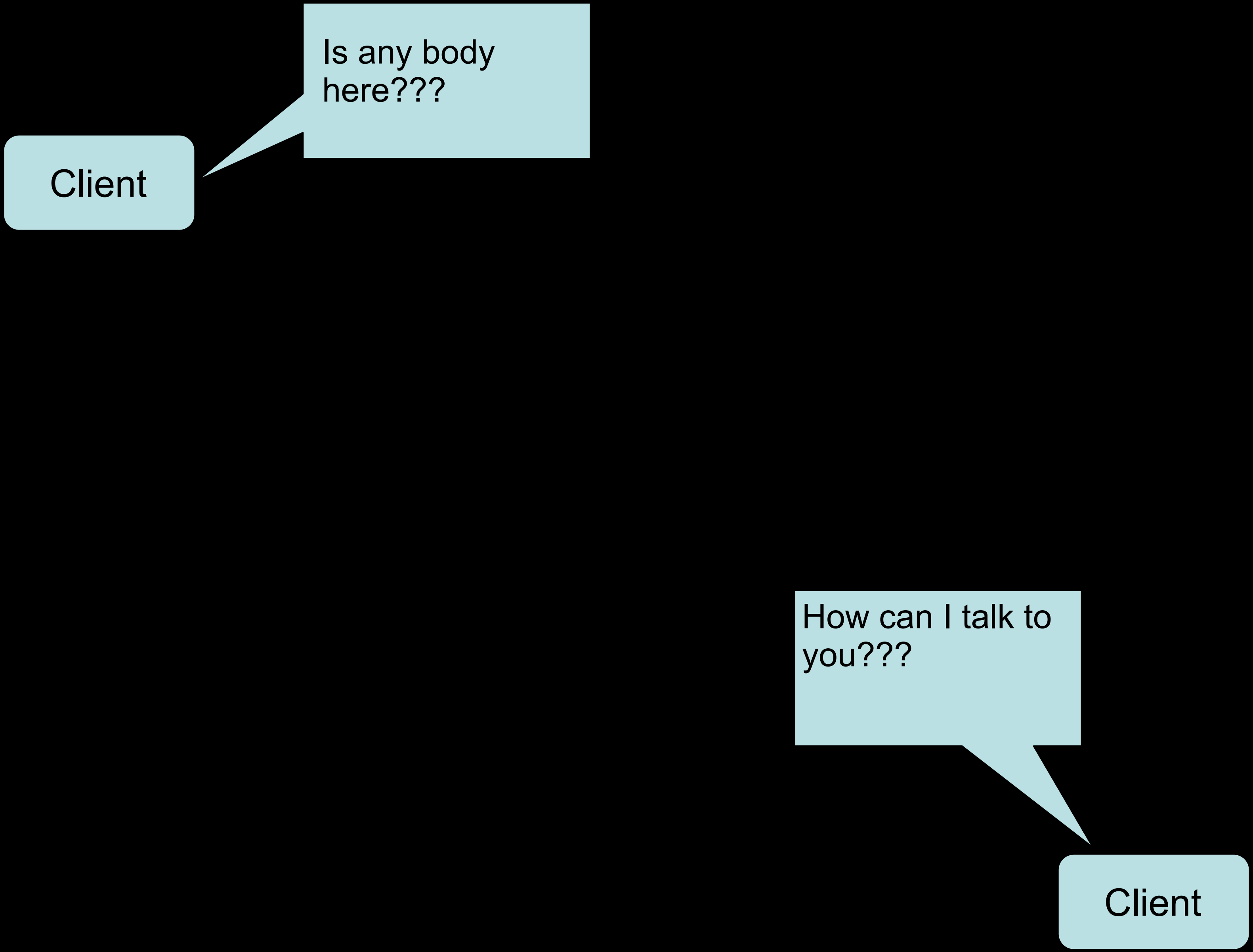
Overview of Project

1. Graphical user interface for Client and Server.
2. Each client can login in the IM system and send text message to others by Server.
3. Distributed servers system to improve IM server performance.
4. Asymmetric en/decryption password to guarantee the security of IM communication.

The Web-World's Most Popular Communication Tools

Gadu-Gadu
polski komunikator internetowy
AIM
Google
talk
BETA
ebuddy
skype
Y!
facebook
jabber inc.
Gizmo5
IBM
icq
everybody, everywhere

... ...



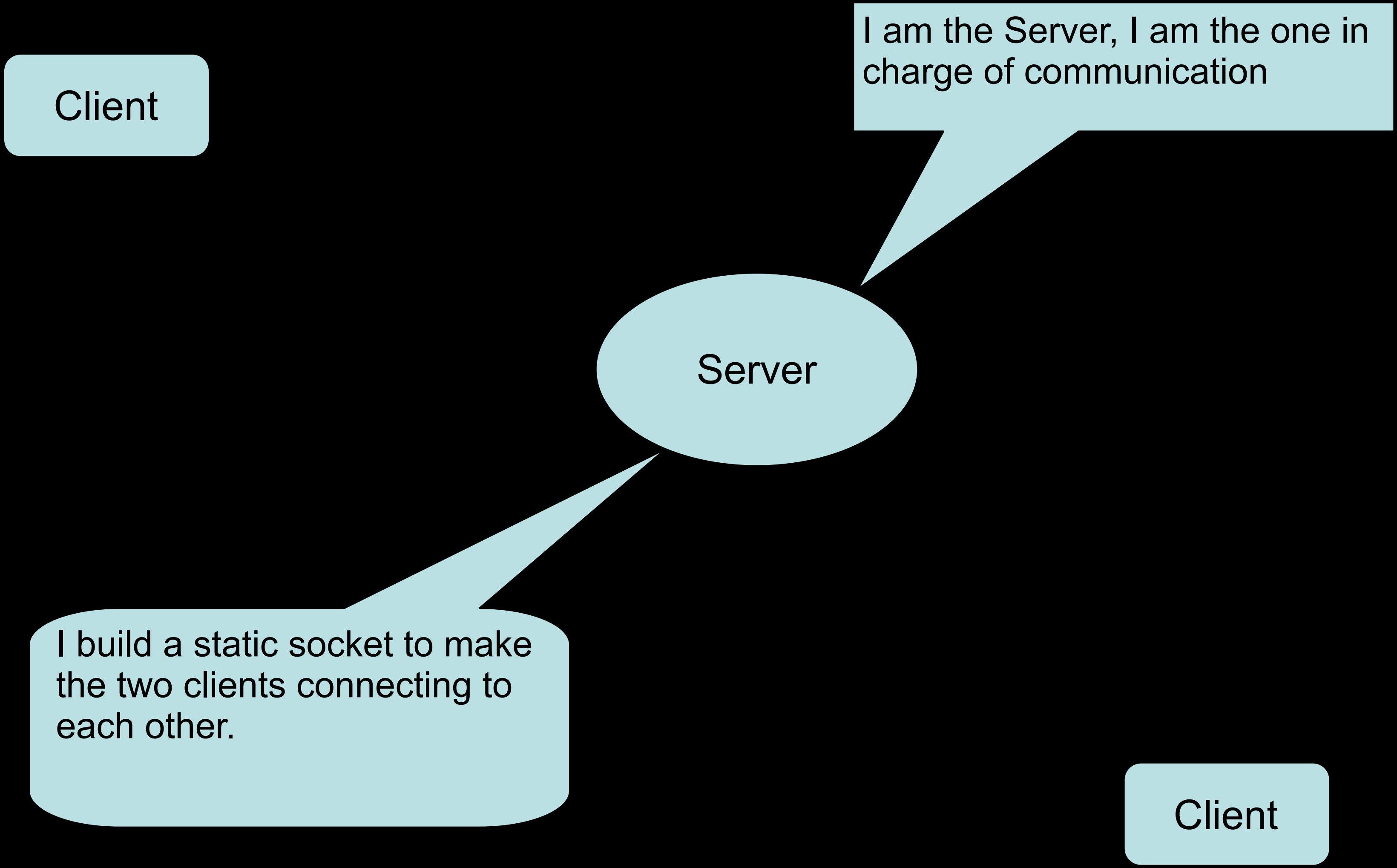
Is any body here???

Client

How can I talk to you???

Client

Client/Server Model



I am the Server, I am the one in charge of communication

Client

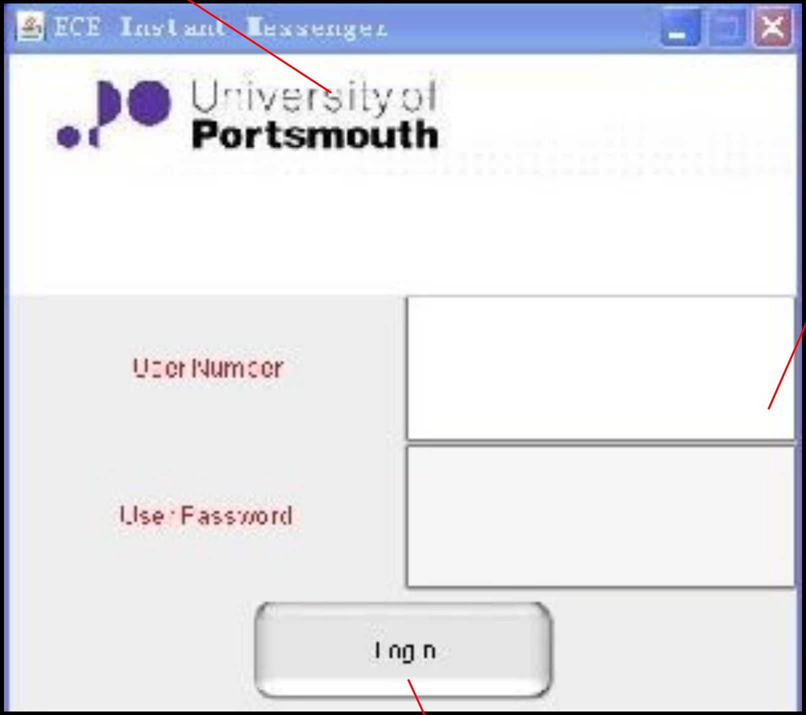
Server

I build a static socket to make the two clients connecting to each other.

Client

The Client Login Window

(North) JLabel (Flowlayout)

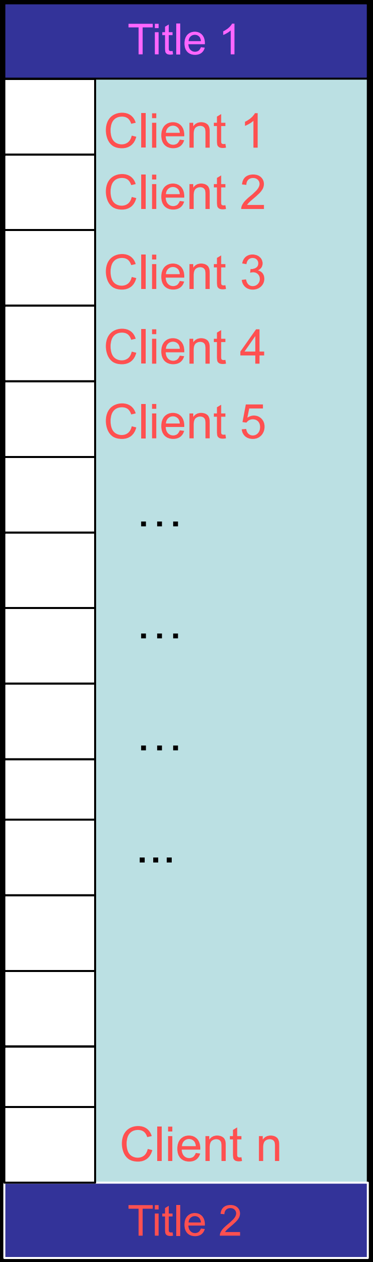


(Center)

JPanel, JLabel

JTextField,JPasswordField
(GridLayout 2,2)

(South) JButton (Flowlayout)



Title 1

Client 1

Client 2

Client 3

Client 4

Client 5

…

…

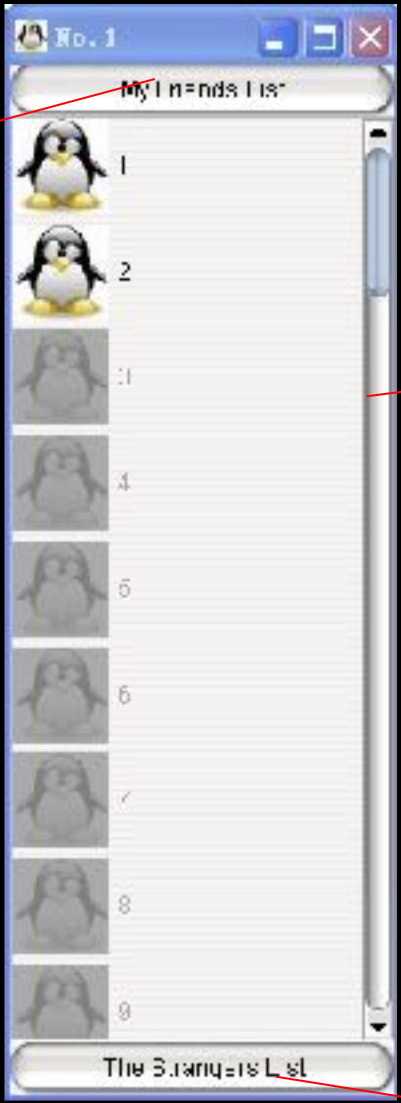
…

…

Client n

Title 2

The Client Friends List



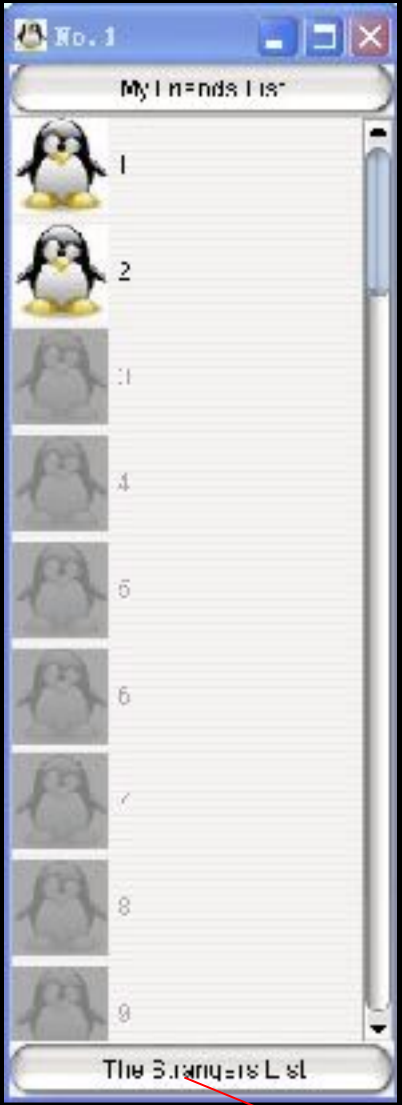
(North) JButton

(Center) JPanel, JScrollPanel

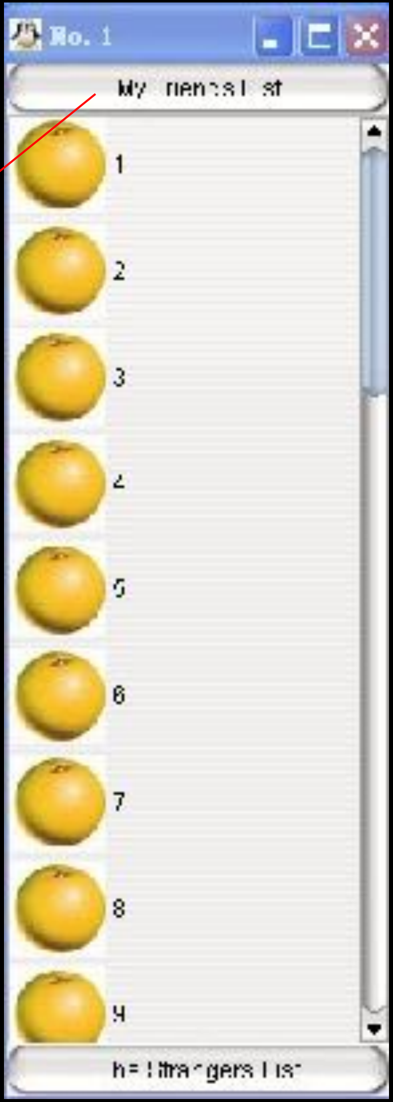
(GridLayout 50,1,5,5)

(South) JButton

CardLayout

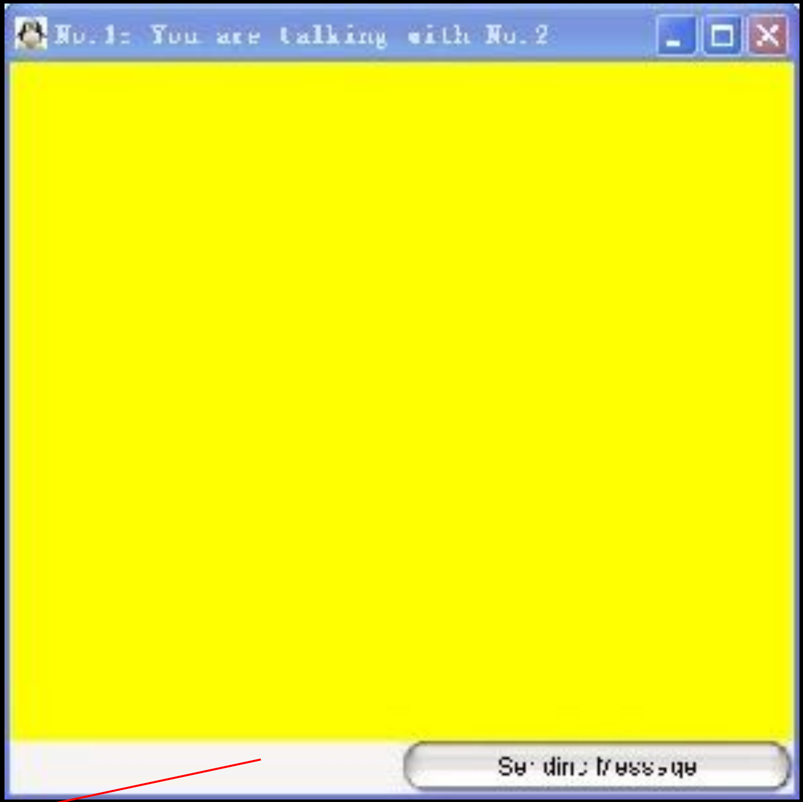


If click “My Friends List” button



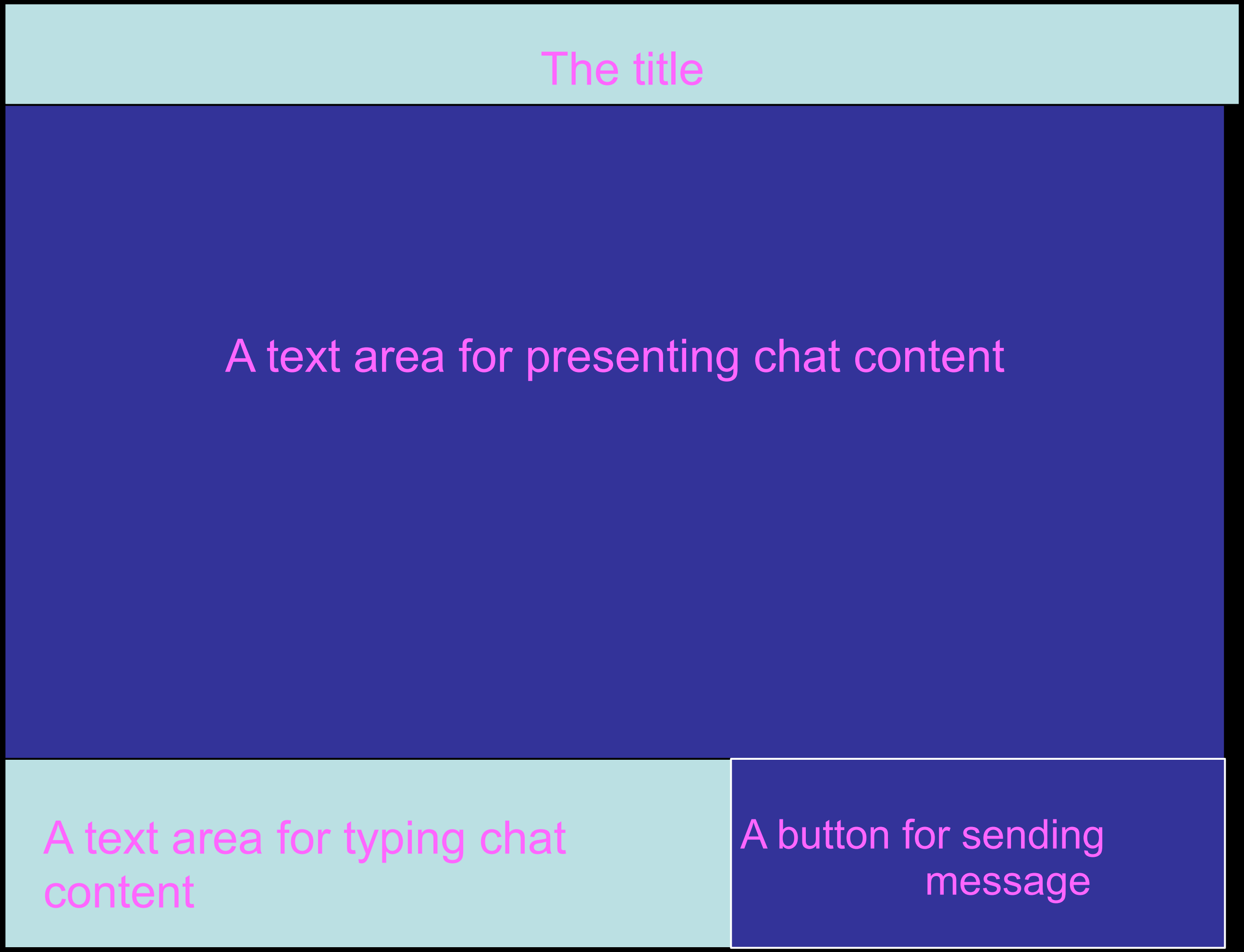
If click “The Strangers List” button

The Client Chat Window



(Center)JTextArea

(South)JPanel with JTextArea, JButton



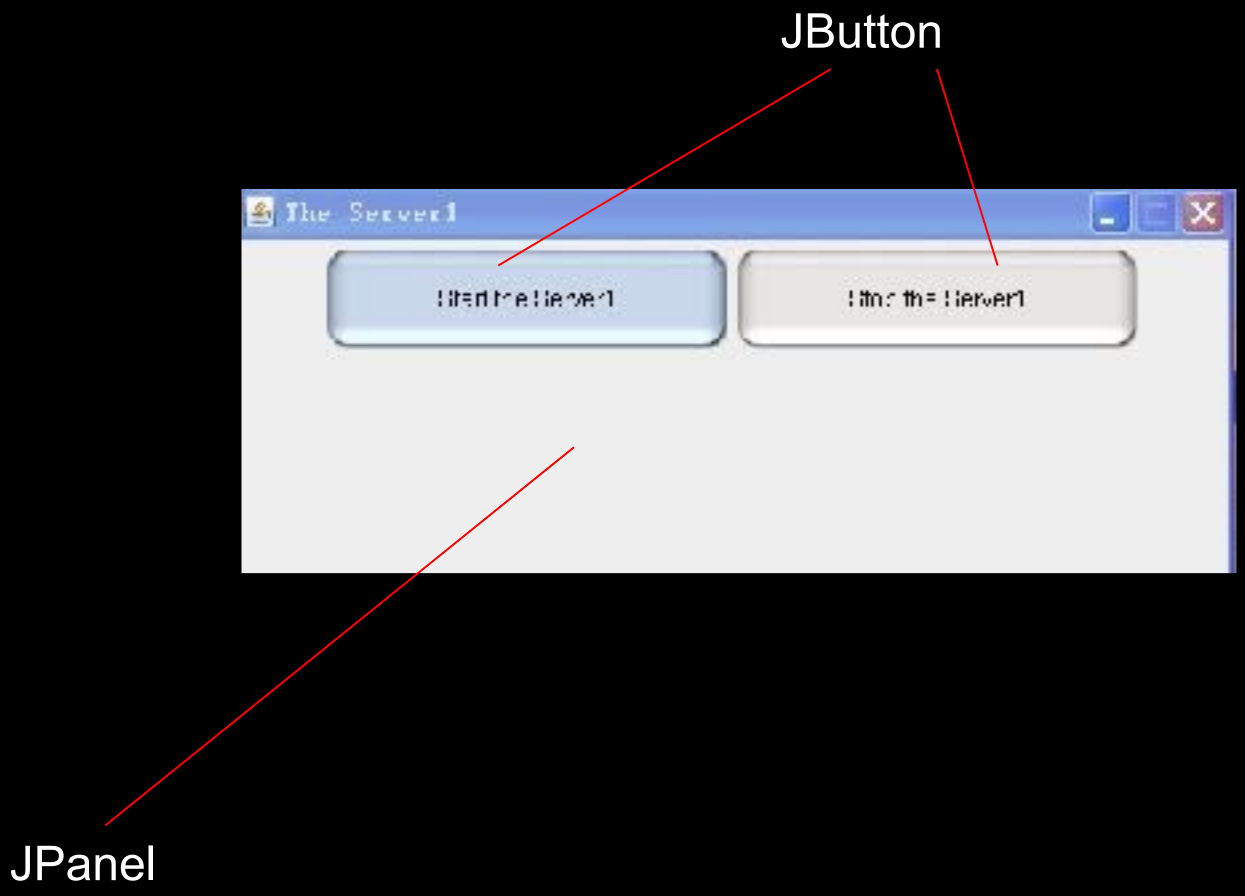
The title

A text area for presenting chat content

A text area for typing chat content

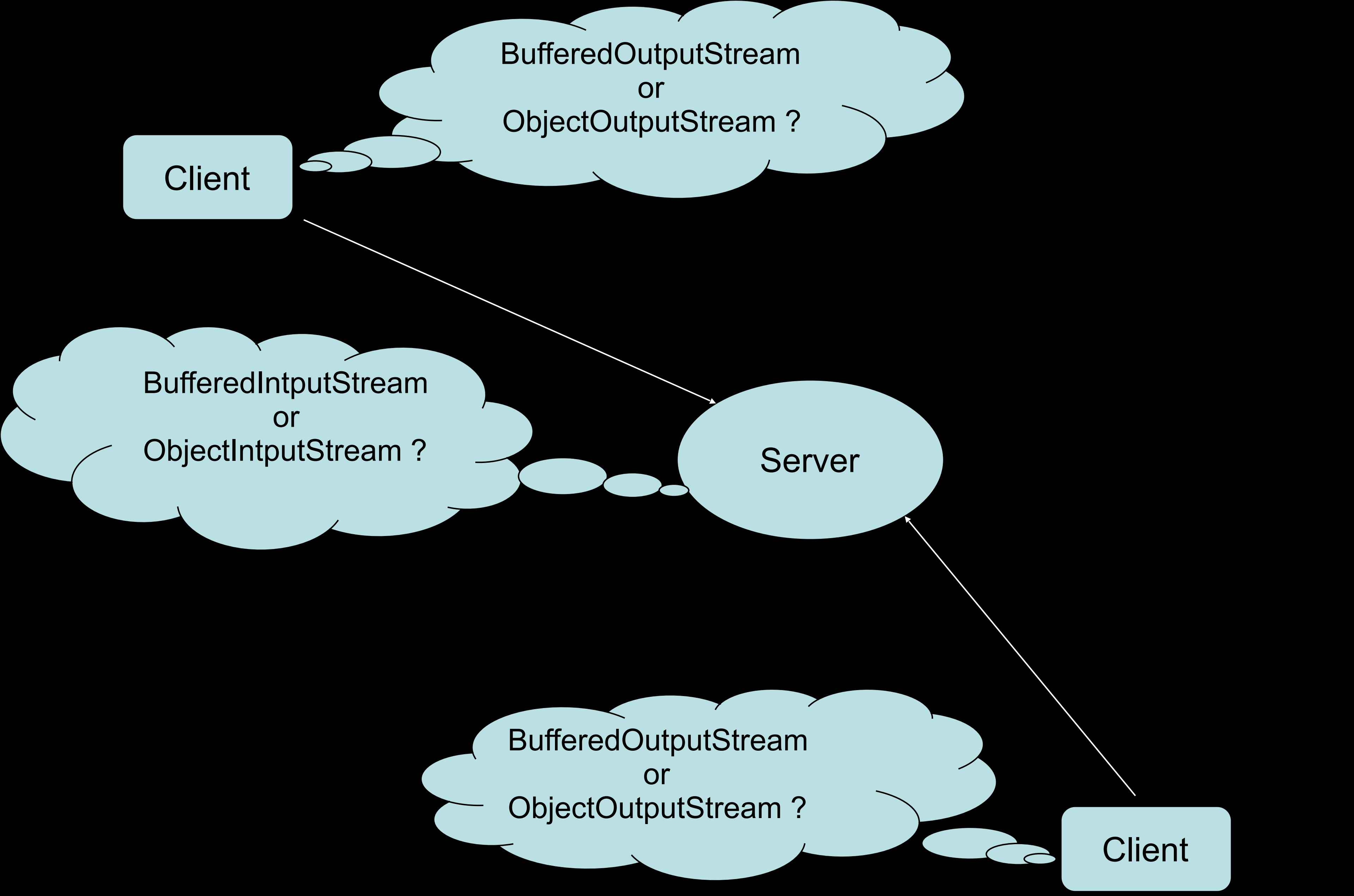
A button for sending message

The Server Frame



JButton

JPanel



BufferedOutputStream
or
ObjectOutputStream ?

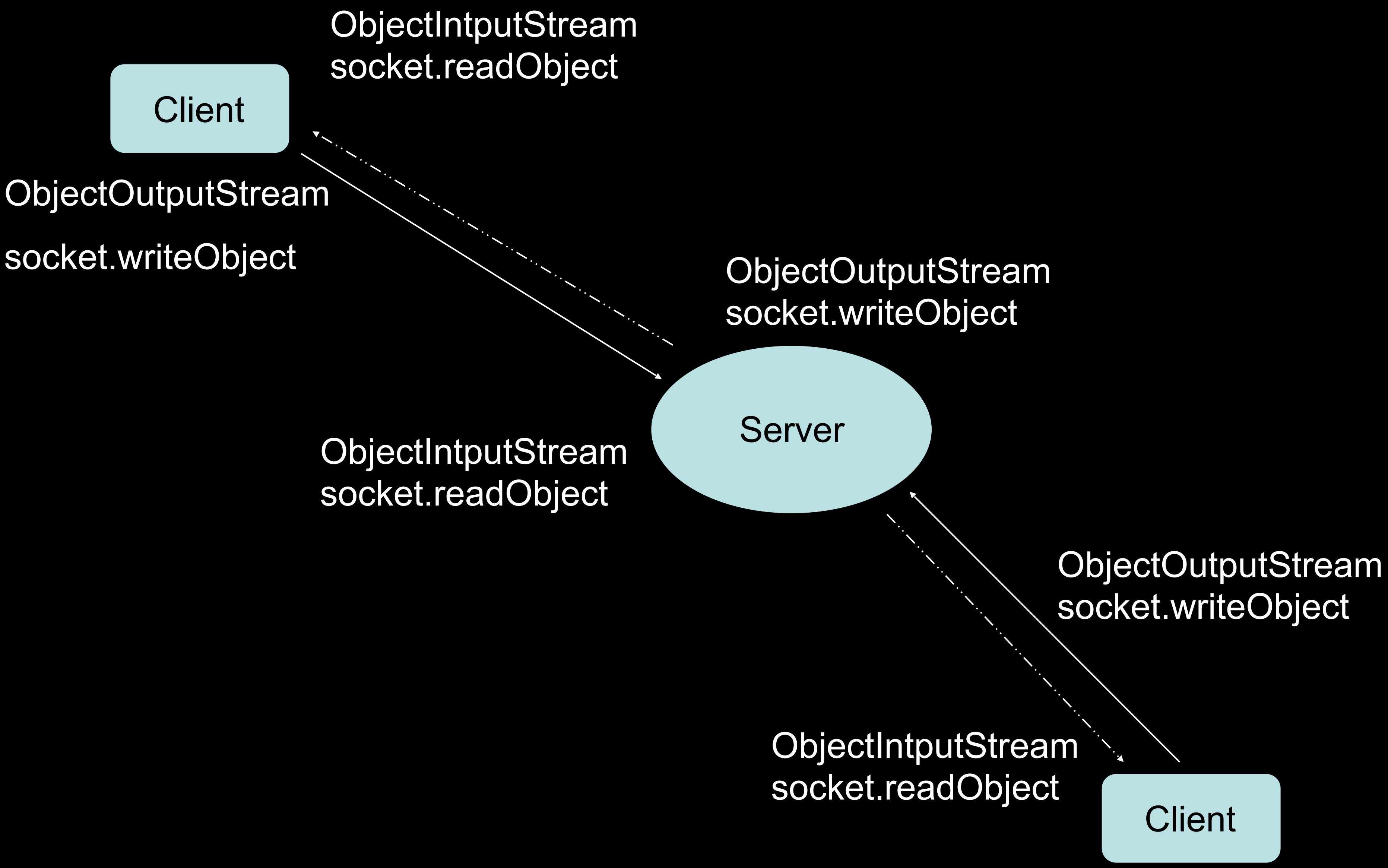
Client

BufferedIntputStream
or
ObjectIntputStream ?

Server

BufferedOutputStream
or
ObjectOutputStream ?

Client



ObjectIntputStream
socket.readObject

Client

ObjectOutputStream

socket.writeObject

ObjectOutputStream
socket.writeObject

Server

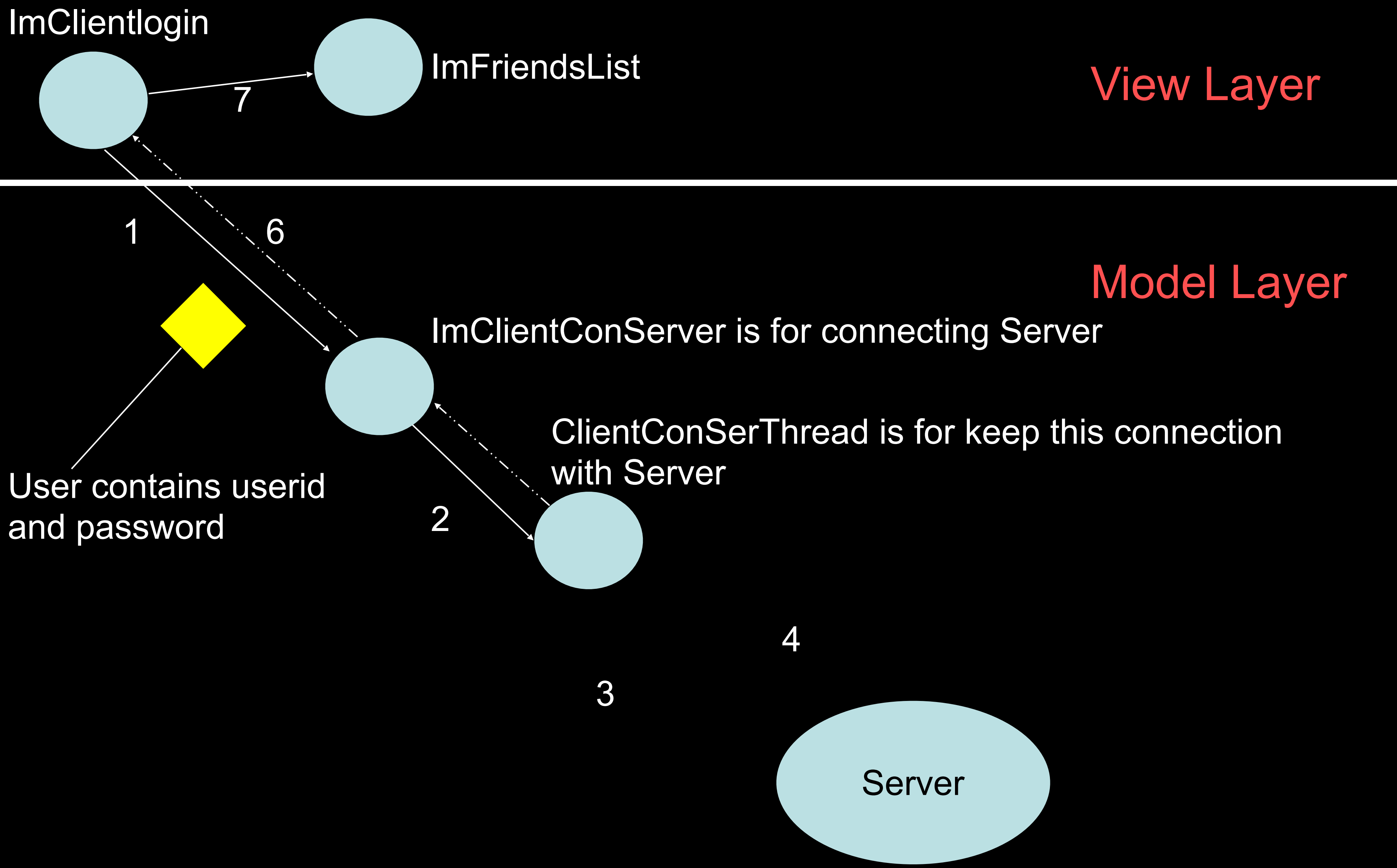
ObjectIntputStream
socket.readObject

ObjectOutputStream
socket.writeObject

ObjectIntputStream
socket.readObject

Client

Login System Architecture



ImClientlogin

ImFriendsList

View Layer

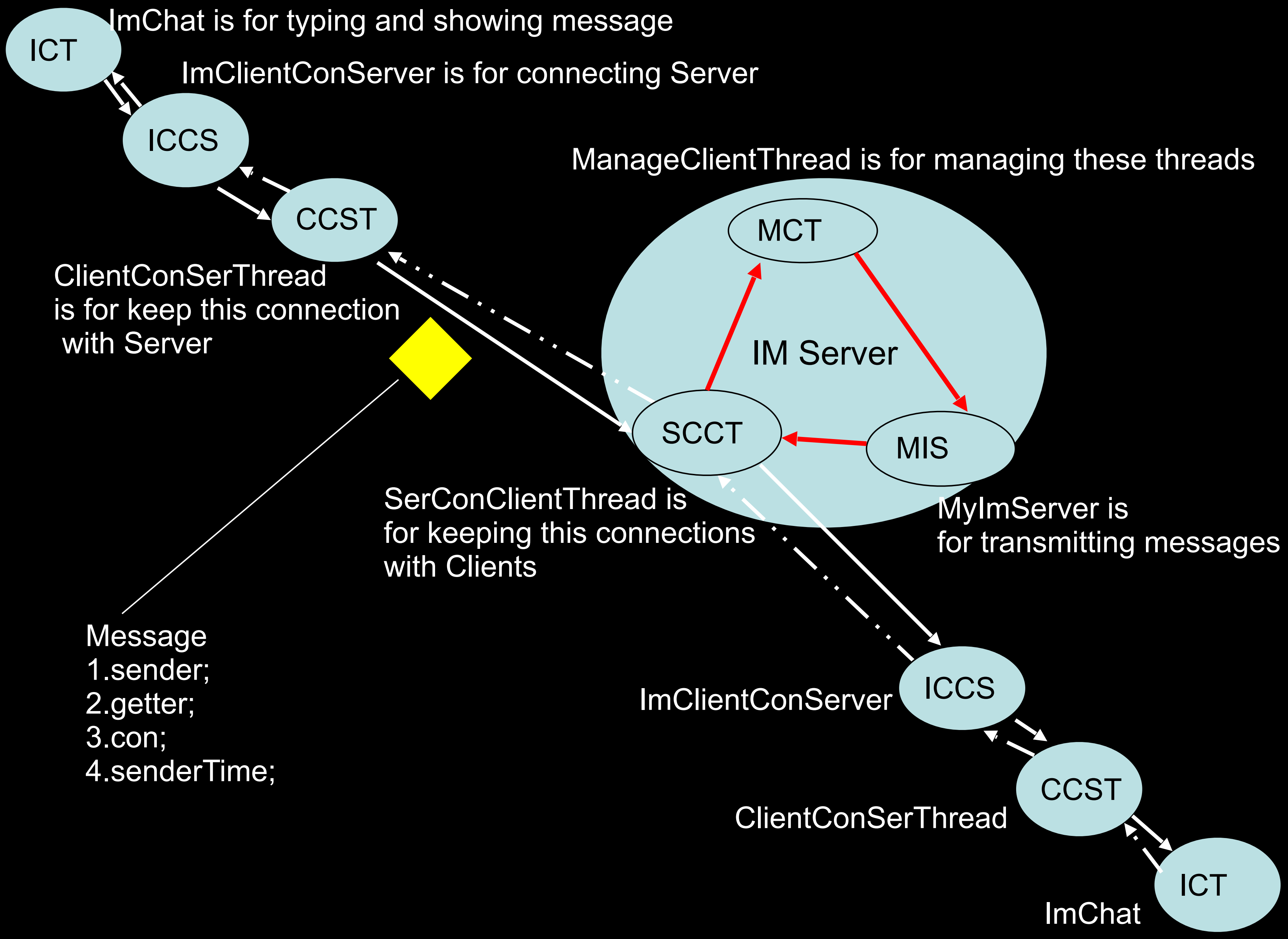
Model Layer

ImClientConServer is for connecting Server

ClientConSerThread is for keep this connection with Server

User contains userid and password

Server



ICT

ImChat is for typing and showing message

ImClientConServer is for connecting Server

ICCS

ManageClientThread is for managing these threads

CCST

MCT

ClientConSerThread
is for keep this connection
with Server

IM Server

SCCT

MIS

SerConClientThread is
for keeping this connections
with Clients

MyImServer is
for transmitting messages

Message
1.sender;
2.getter;
3.con;
4.senderTime;

ImClientConServer

ICCS

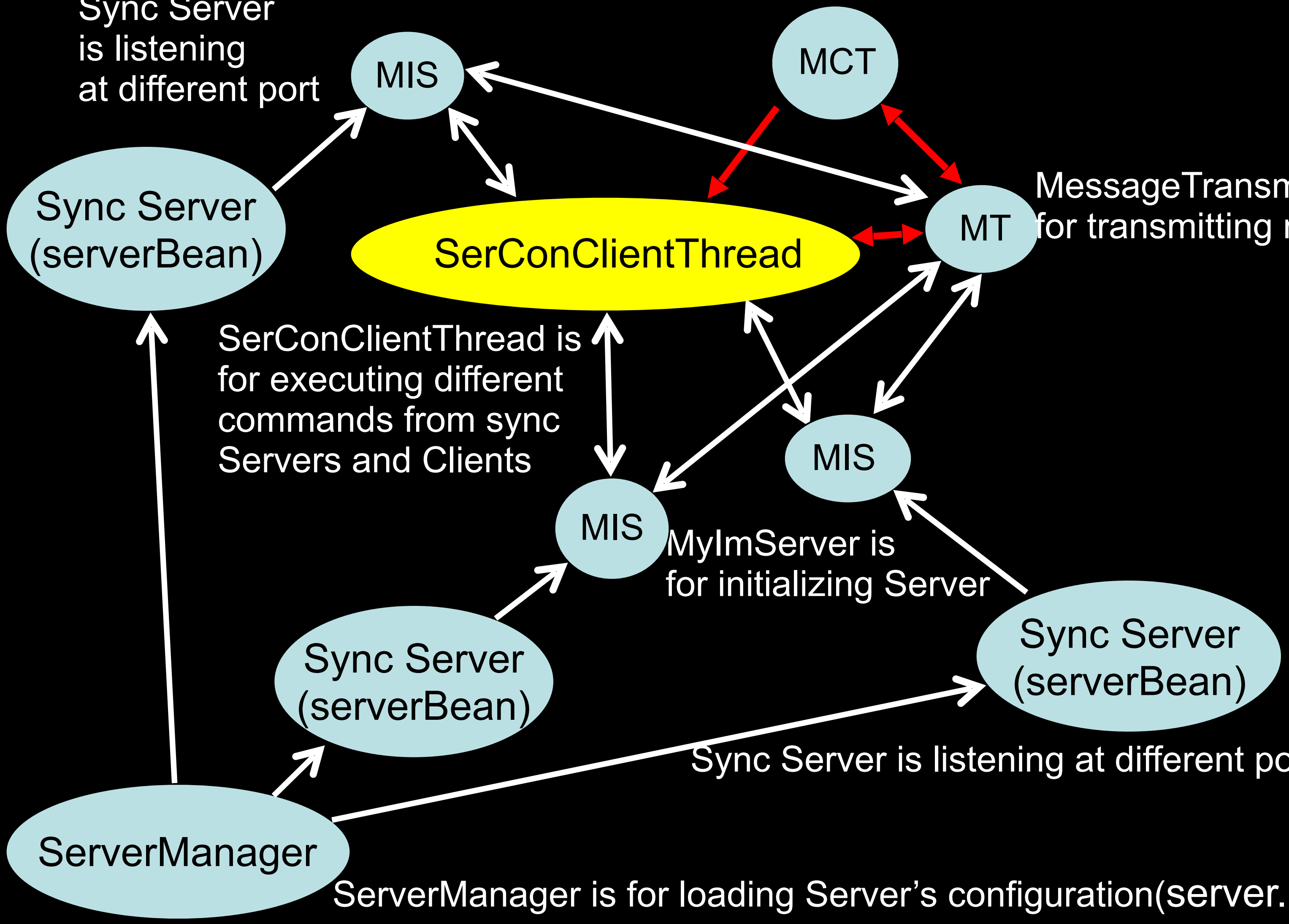
CCST

ClientConSerThread

ICT

ImChat

Server’s Side



ManageClientThread is for providing methods and collections stored connected threads

Sync Server is listening at different port

MessageTransmitter is for transmitting messages

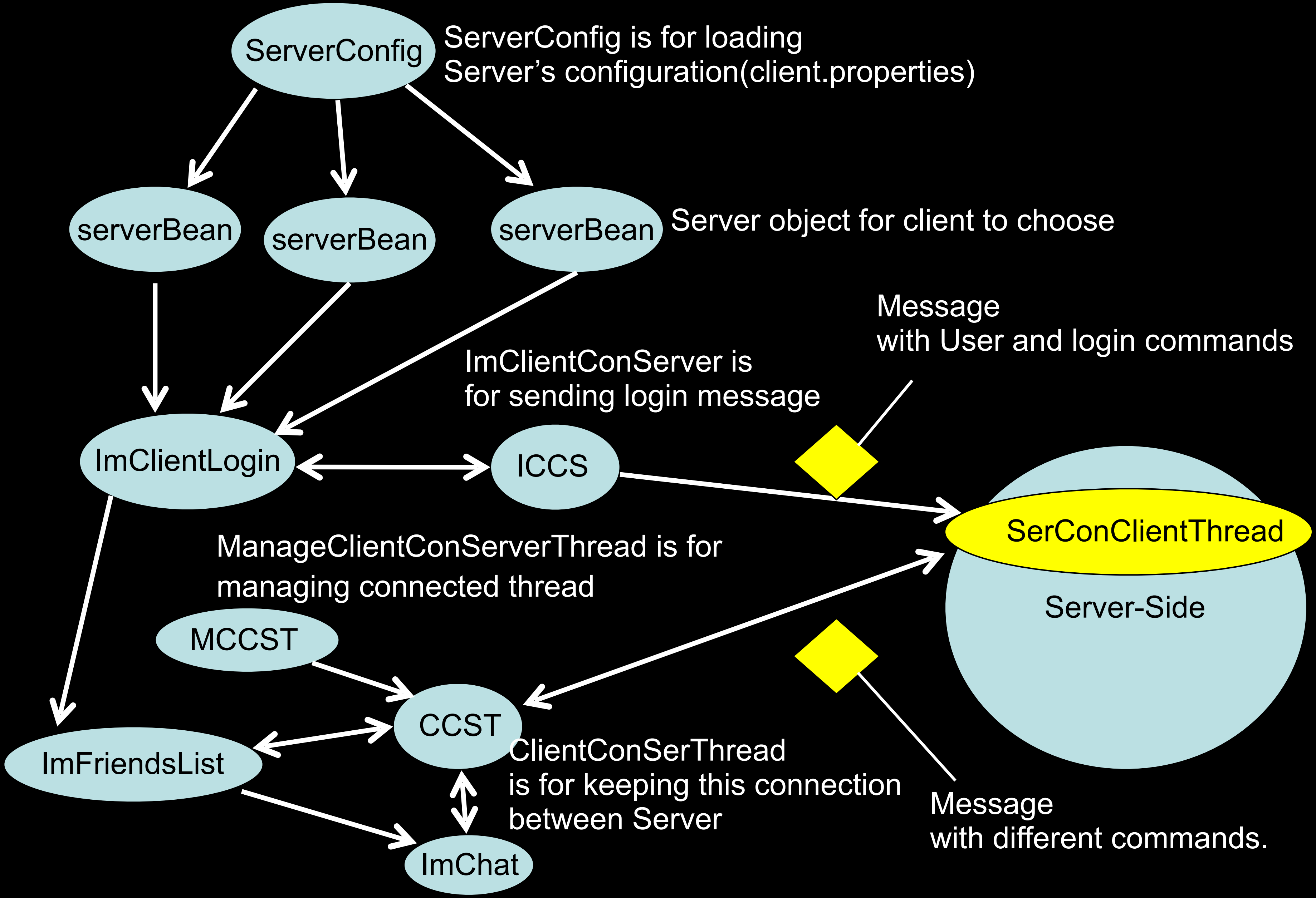
SerConClientThread is for executing different commands from sync Servers and Clients

MyImServer is for initializing Server

Sync Server is listening at different port

ServerManager is for loading Server’s configuration(server.properties)

Client's Side



ServerConfig is for loading
Server's configuration(client.properties)

Server object for client to choose

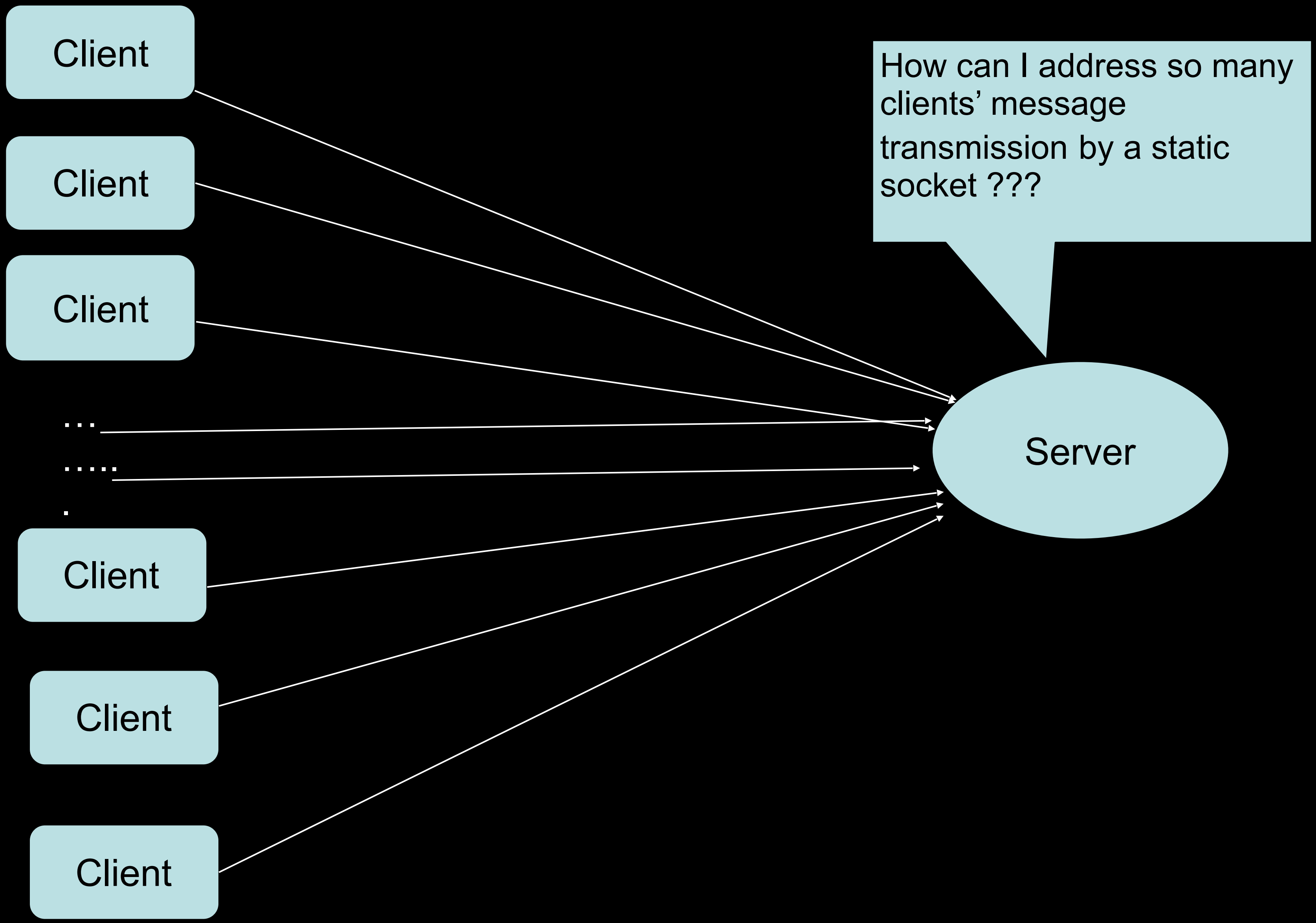
Message
with User and login commands

ImClientConServer is
for sending login message

ManageClientConServerThread is for
managing connected thread

ClientConSerThread
is for keeping this connection
between Server

Message
with different commands.



Client

Client

Client

…

…..

.

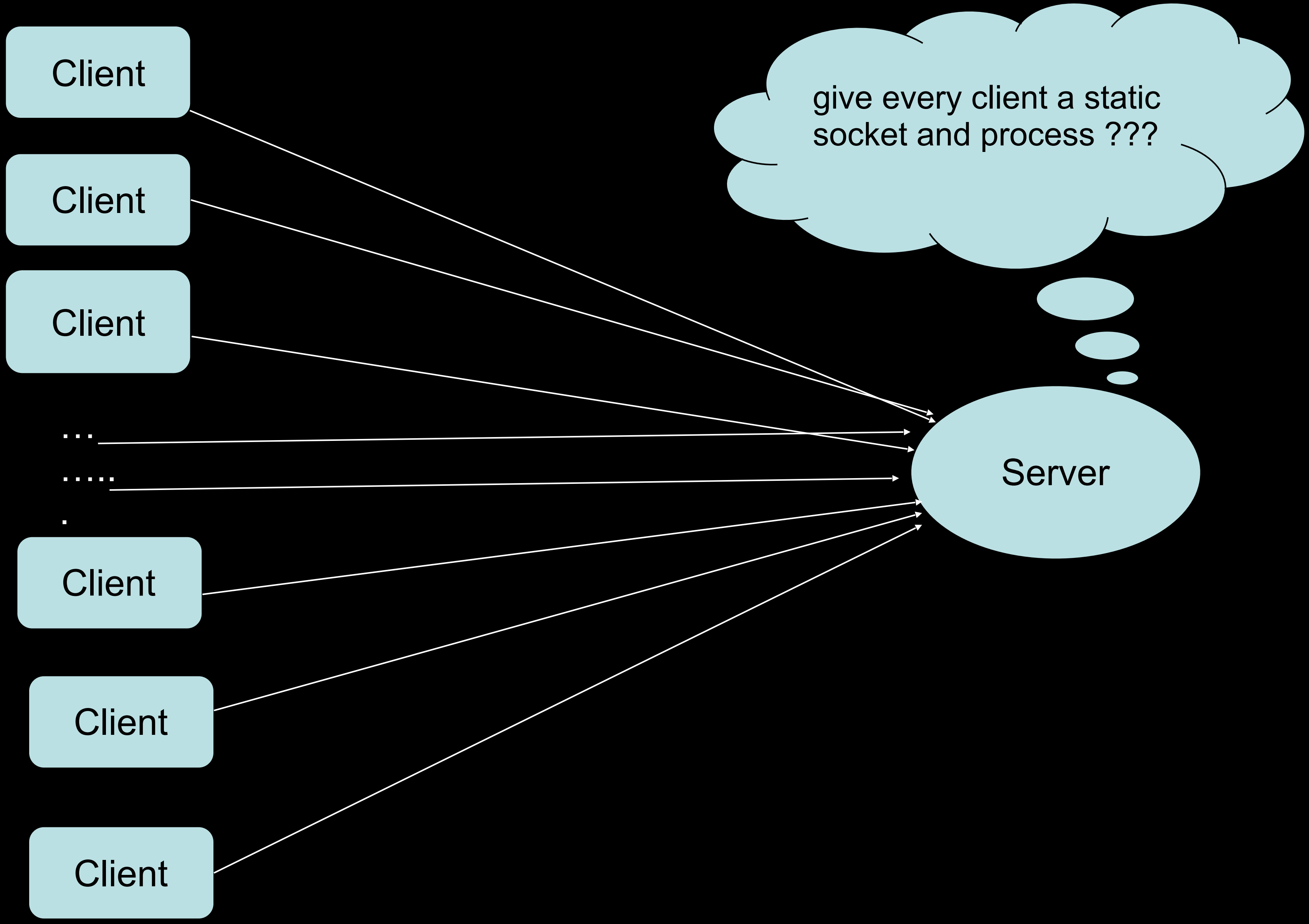
Client

Client

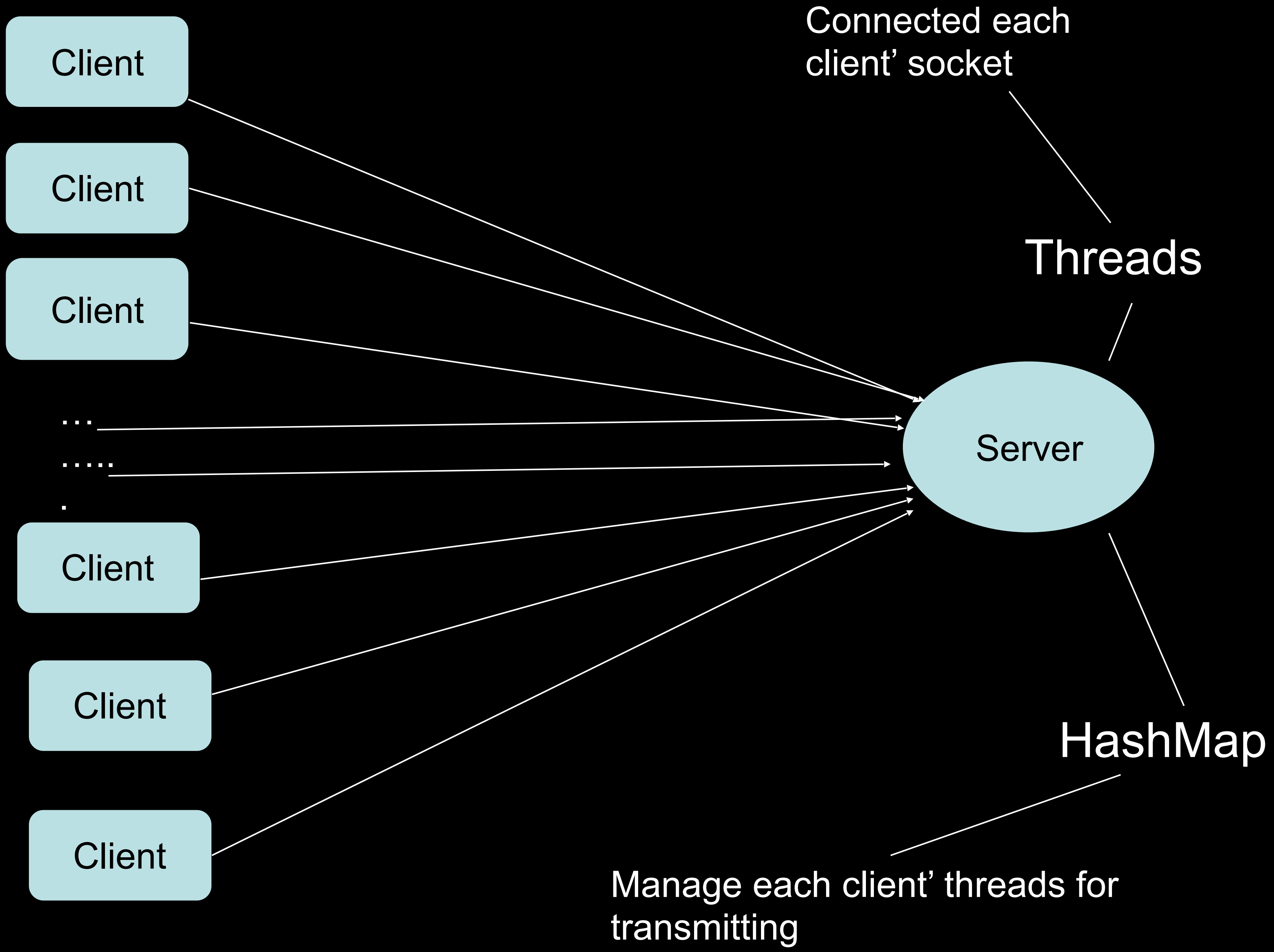
Client

Server

How can I address so many clients’ message transmission by a static socket ???



give every client a static socket and process ???



Client

Client

Client

...

.....

.

Client

Client

Client

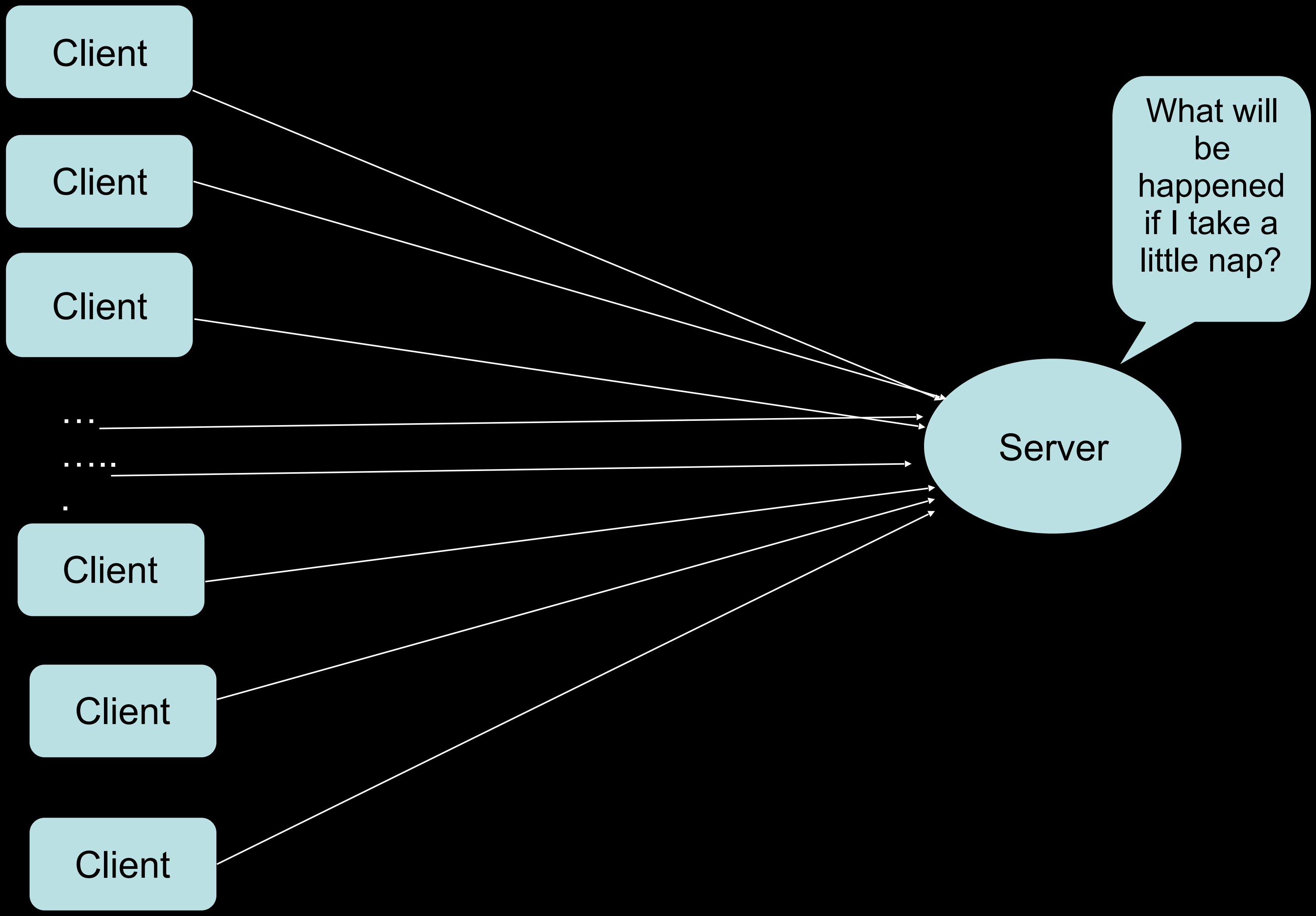
Server

Connected each
client’ socket

Threads

HashMap

Manage each client’ threads for
transmitting



Client

Client

Client

…

…..

.

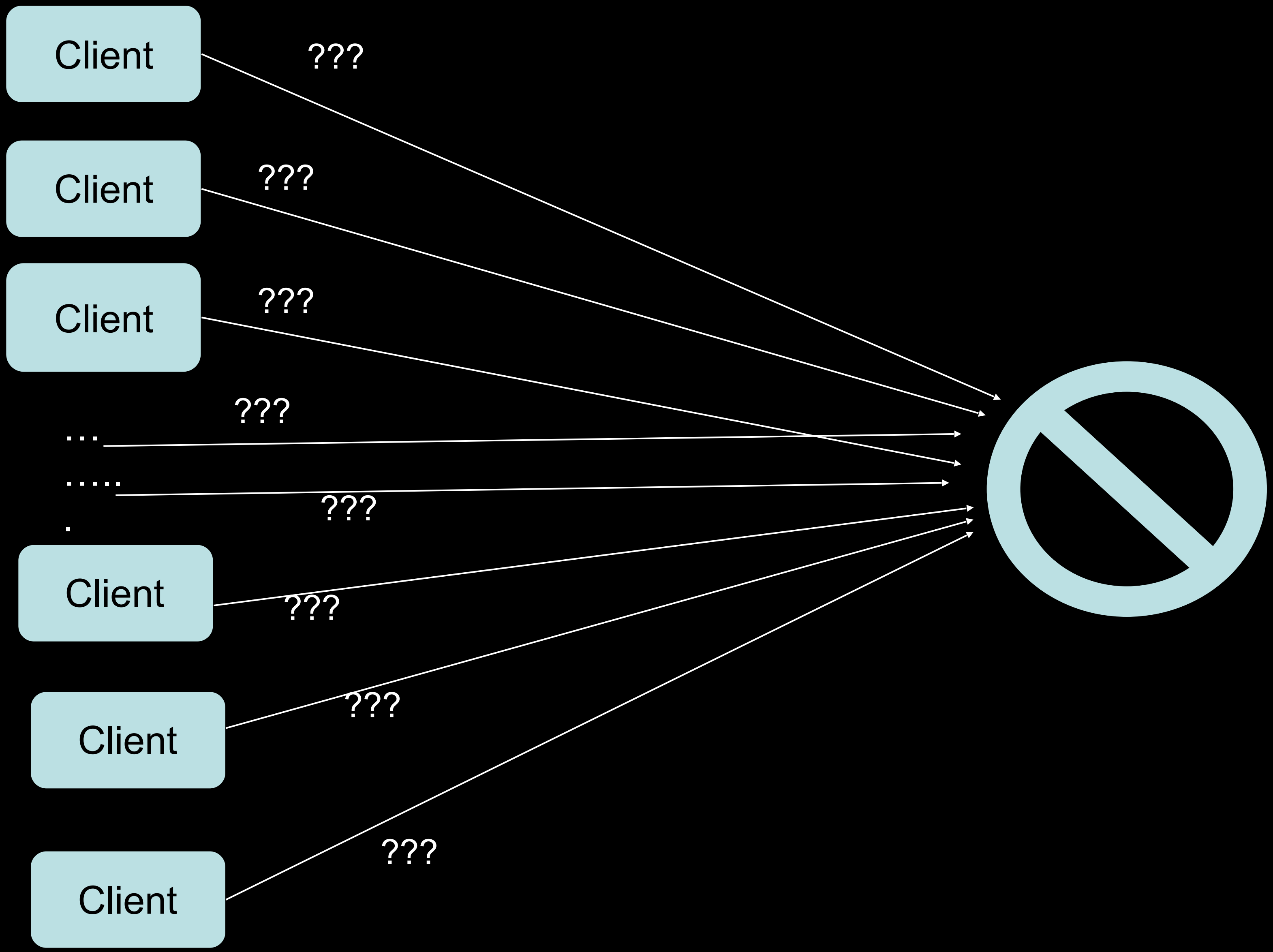
Client

Client

Client

Server

What will be happened if I take a little nap?



Client

???

Client

???

Client

???

…

…..

.

???

Client

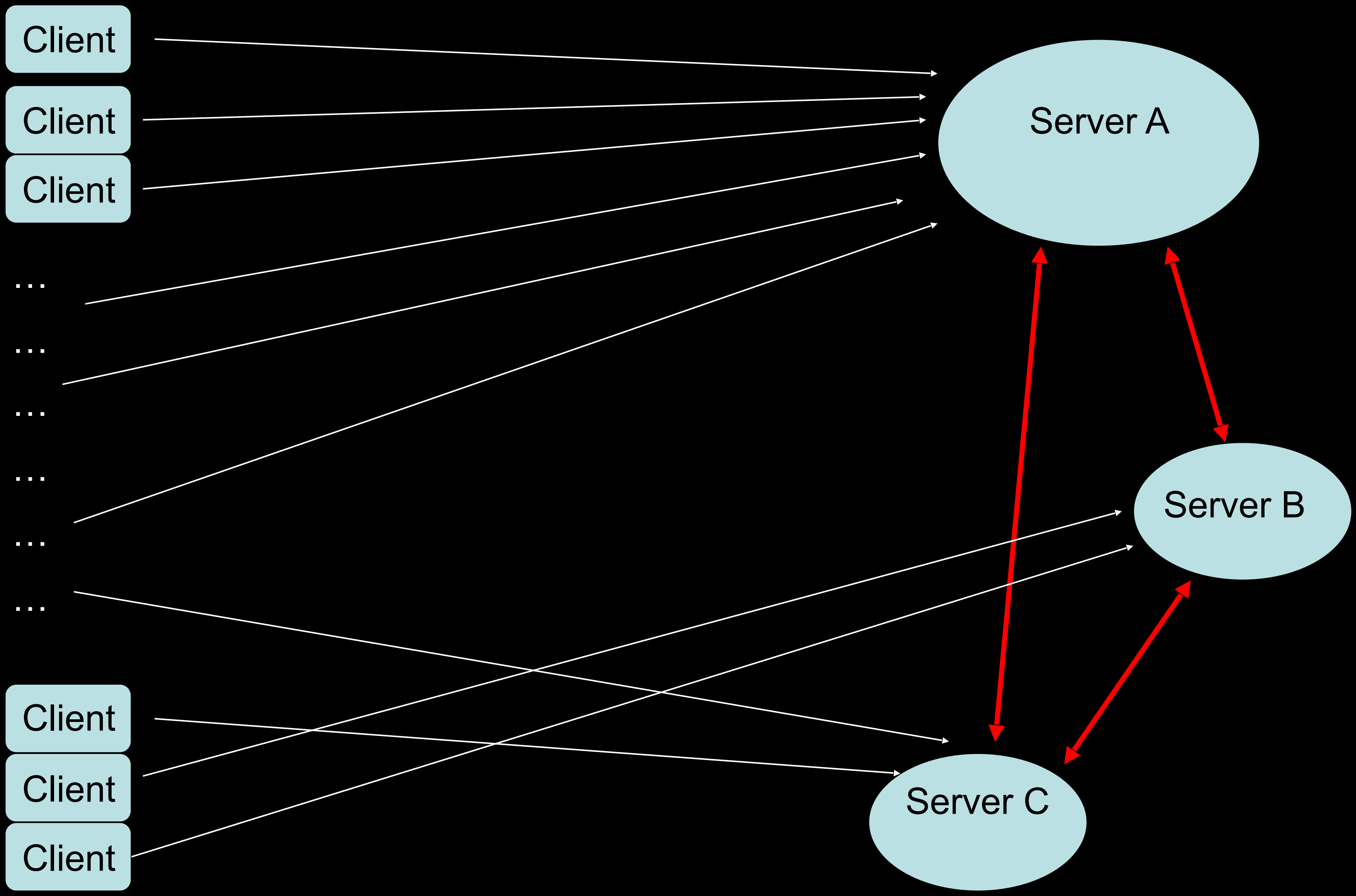
???

Client

???

Client

???


Client
Client
Client
…
…
…
…
…
…
Client
Client
Client
Server A
Server B
Server C

Programming Model for Distributed Servers

- Remote procedure call ?

- Provide clients with ability to call procedures in server programs running in separate process, likely on different computers.

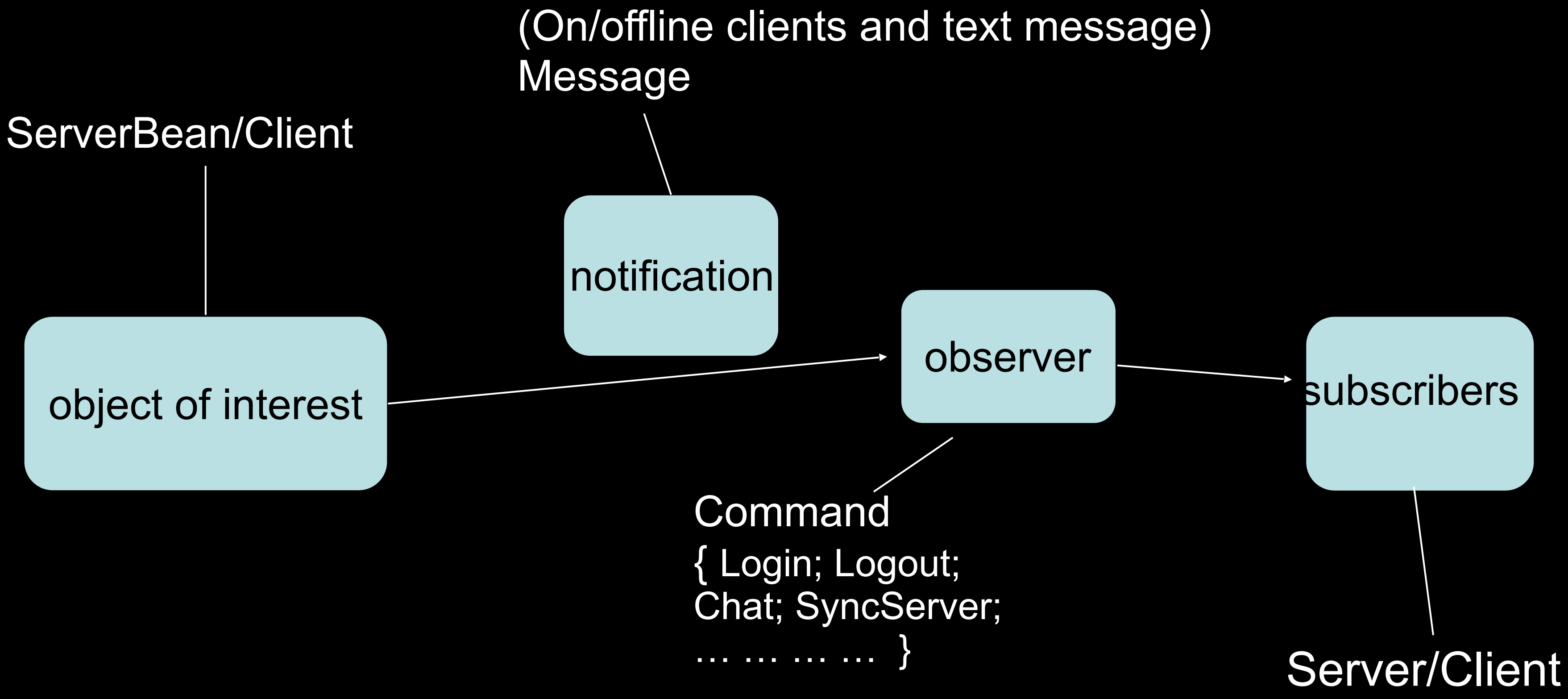
- Remote method invocation ?

- Allowing objects to invoke methods of objects that can be in different processes or computers

- Event-based programming ?

- Objects can receive notifications from other possibly
remote objects about events they are interested in essentially distributed driven programming

Event-based Programming



(On/offline clients and text message)
Message

ServerBean/Client

notification

object of interest

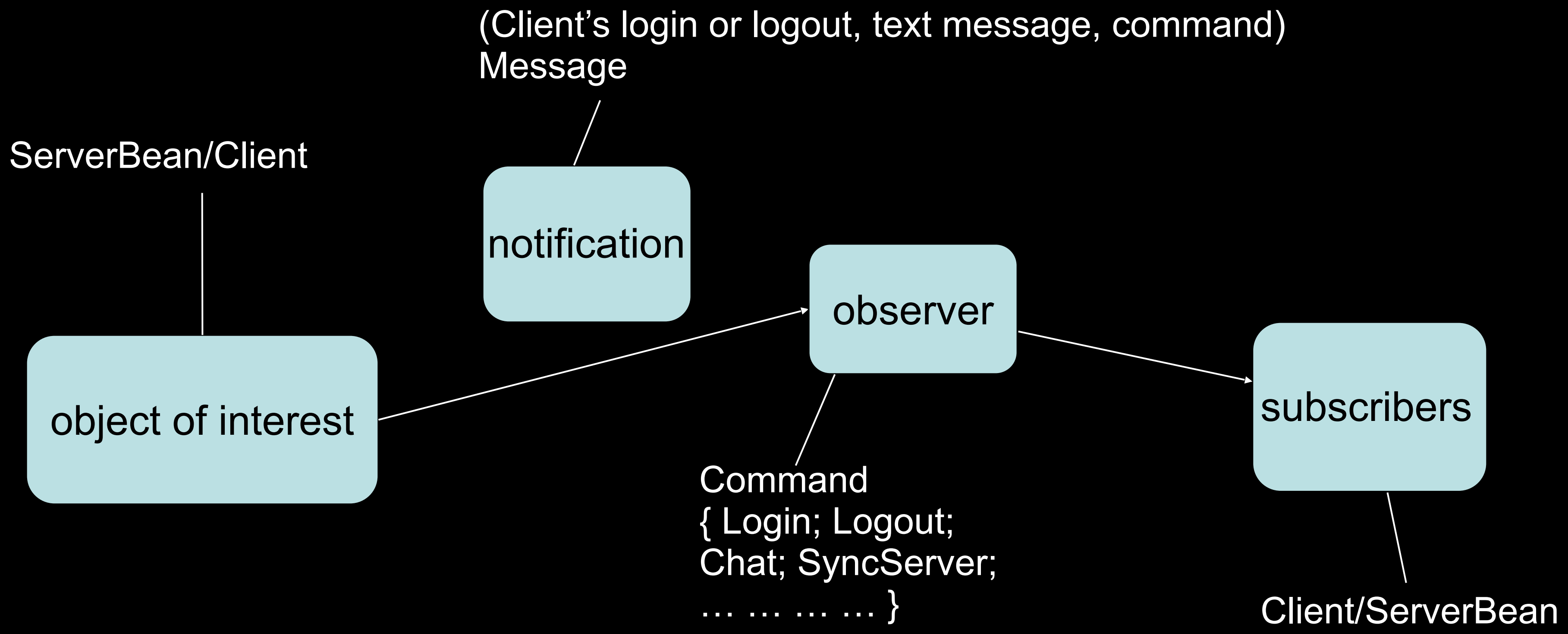
observer

subscribers

Command
{ Login; Logout;
Chat; SyncServer;
… … … … }

Server/Client

Event-based Programming



(Client's login or logout, text message, command)
Message

ServerBean/Client

notification

observer

object of interest

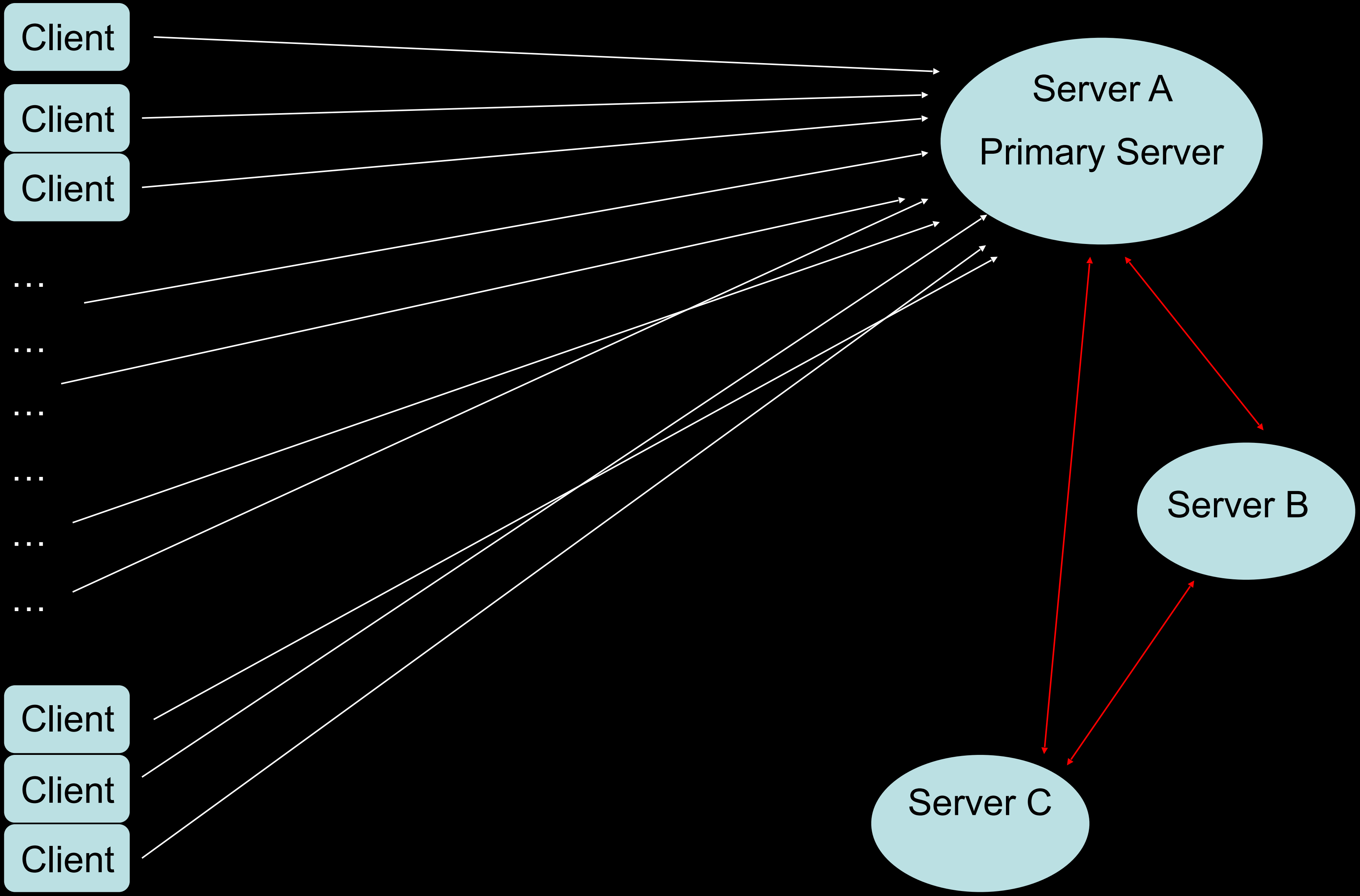
subscribers

Command
{ Login; Logout;
Chat; SyncServer;
… … … … }

Client/ServerBean

To store Server configuration information
XML or Properties ?

Properties :   Key ---> Value
Client.properties   : server’s ip address and ports
Server.properties : server’s ip address and ports



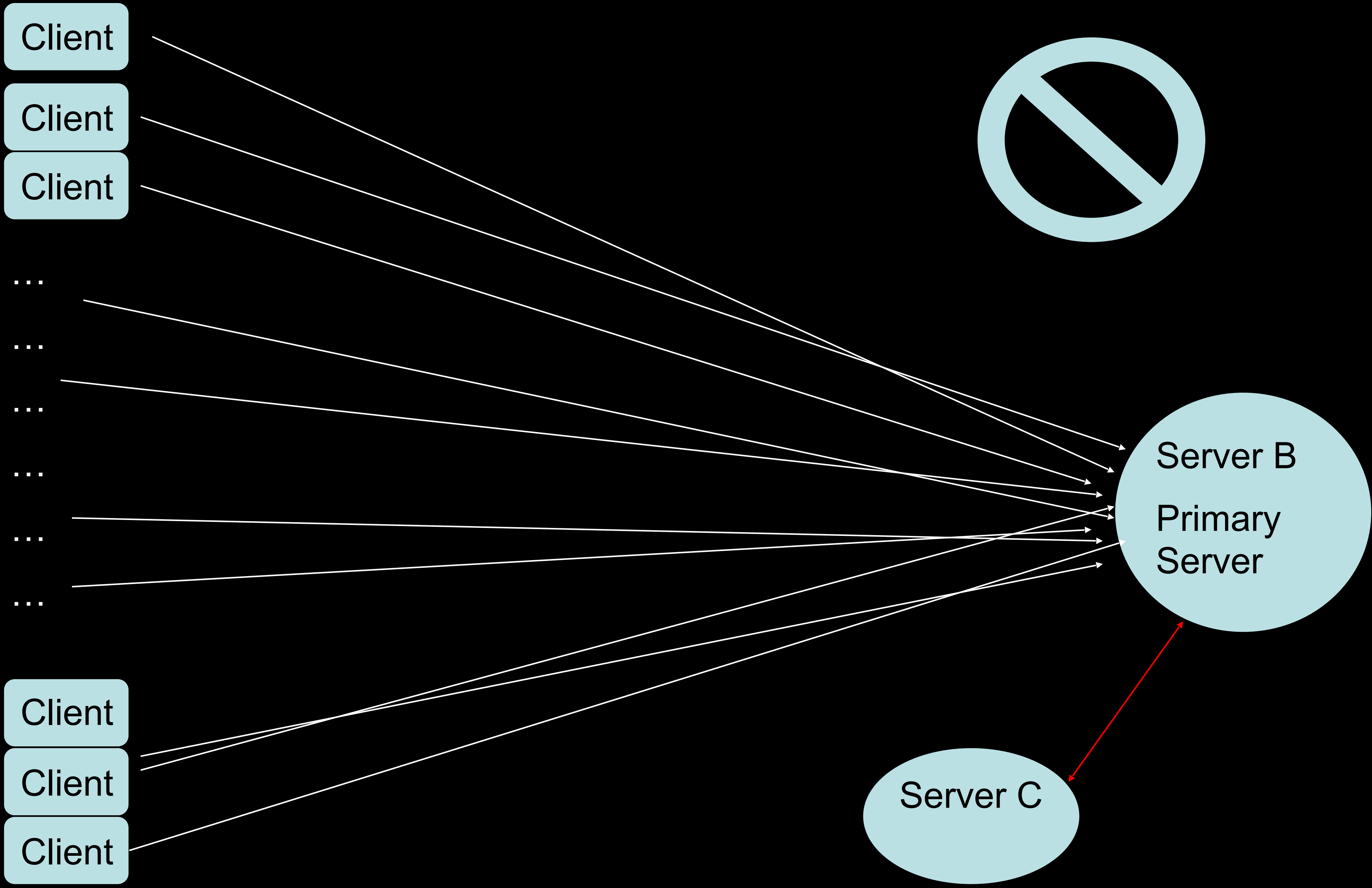
Client
Client
Client
…
…
…
…
…
…
Client
Client
Client
Server A
Primary Server
Server B
Server C

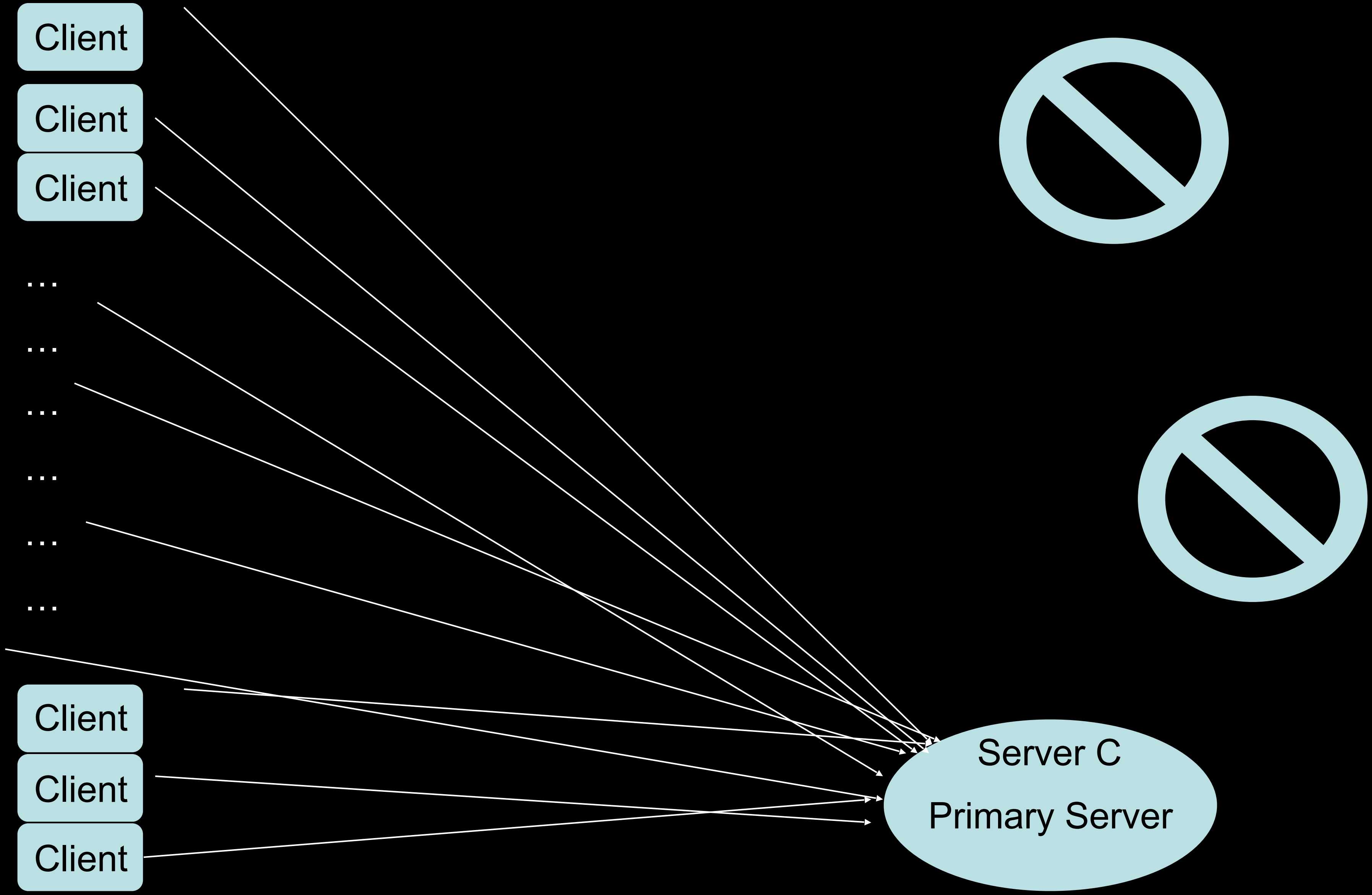
Pseudo-code

```
    // keep serverbean in sorted order
    Map serverMap <String, ServerBean> = Collections.synchronizedMap(TreeMap)
```

Ordering

```
Client:
// for(forever)
 for(Iterator<ServerBean> iterator = sc.iterator();iterator.hasNext();)
        {
         // iterate ServerBean to connect
        serverbean = iteraor.next();
        …
        }
Server:
// for(forever)
  for(Iterator<ServerBean> iter = serverManager.iteratorServerMap(); iter.hasNext();)
        {
        // iterate ServerBean to synchronize
        serverbean = iter.next();
        ….
        }
```


Client
Client
Client
…
…
…
…
…
…
Client
Client
Client
Server B
Primary Server
Server C



Client

Client

Client

…

…

…

…

…

…

Client

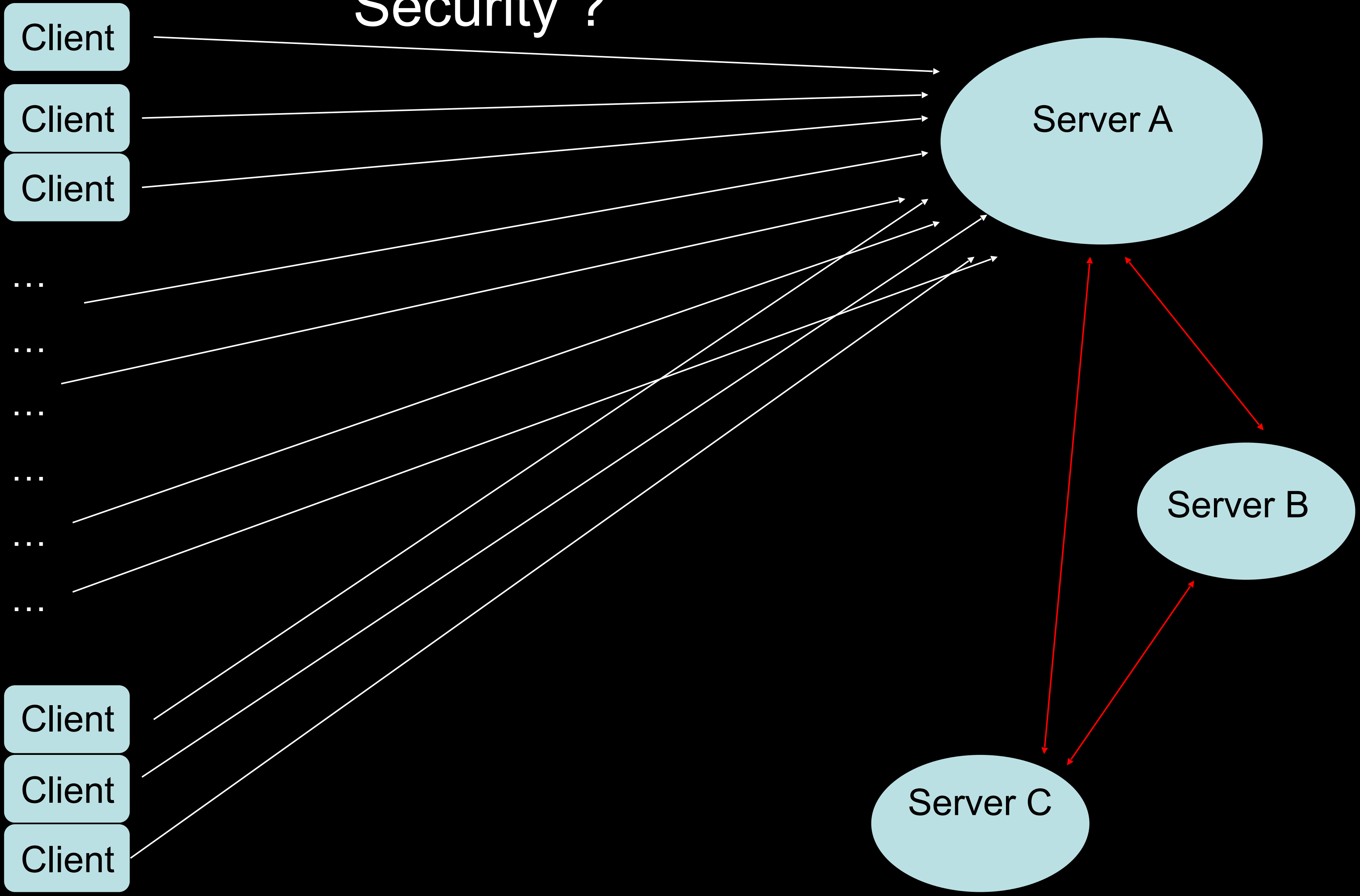
Client

Client

Server C

Primary Server

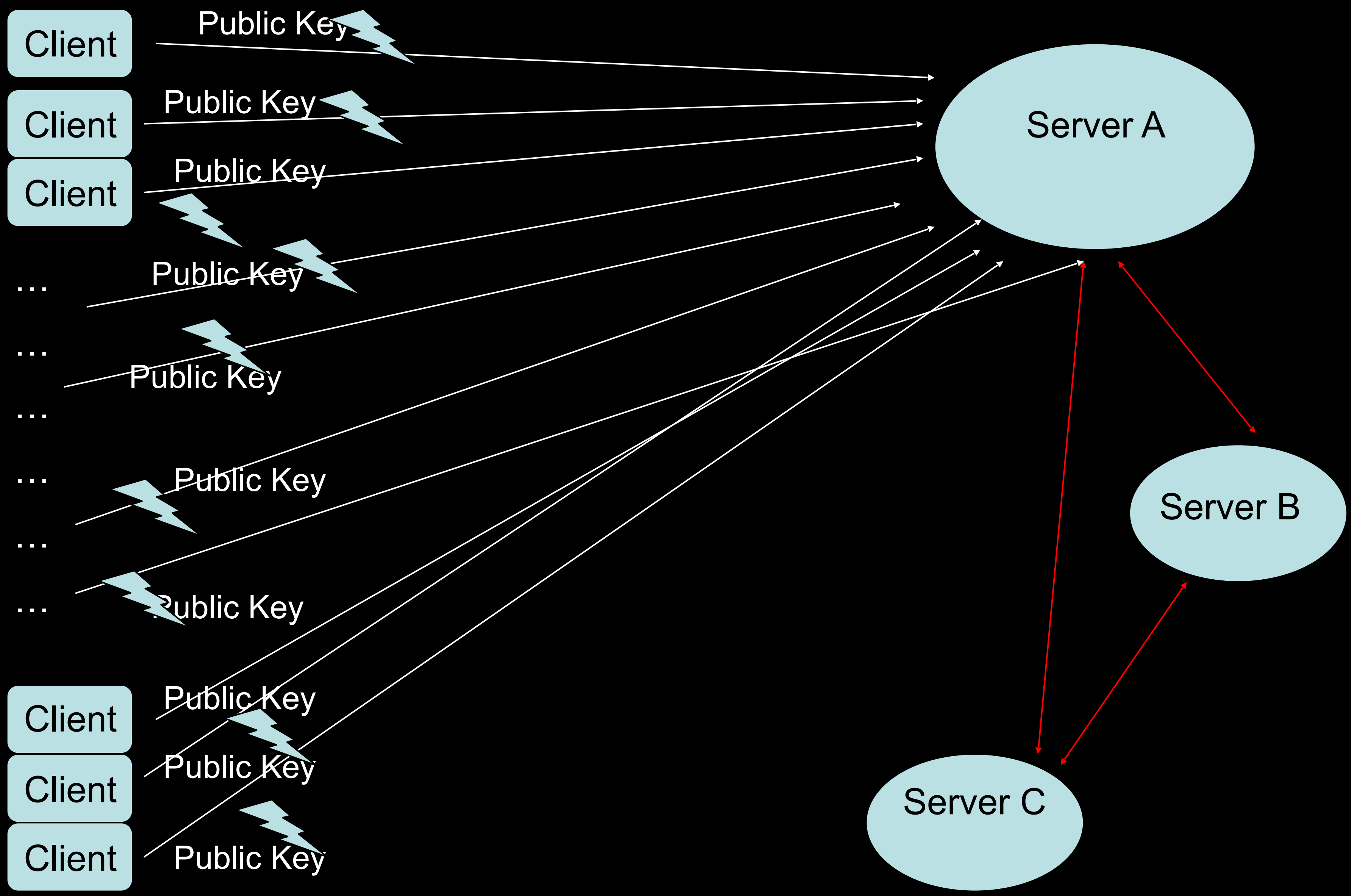
Security ?

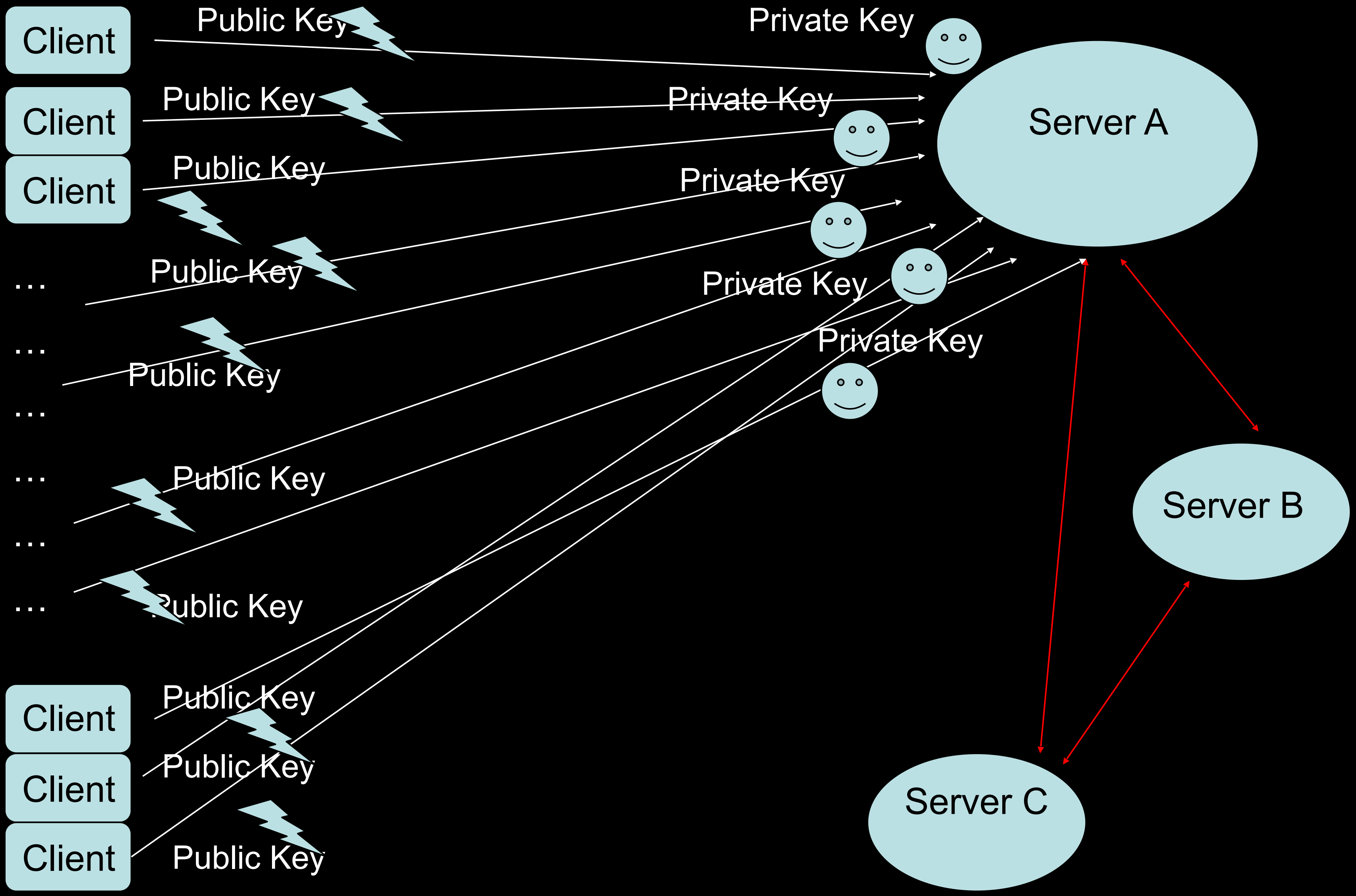

Client
Client
Client
…
…
…
…
…
…
Client
Client
Client
Server A
Server B
Server C

Symmetric En/decryption
(DES, AES…)

OR

Asymmetric En/decryption
(RSA…)

Client
Public Key
Client
Public Key
Client
Public Key
…
Public Key
…
Public Key
…
…
Public Key
…
…
Public Key
Client
Public Key
Client
Public Key
Client
Public Key
Server A
Server B
Server C



Client

Public Key

Private Key

Client

Public Key

Private Key

Server A

Client

Public Key

Private Key

…

Public Key

Private Key

…

Private Key

Public Key

…

…

Public Key

Server B

…

…

Public Key

Client

Public Key

Client

Public Key

Server C

Client

Public Key

Achievement so far

QUESTION ?