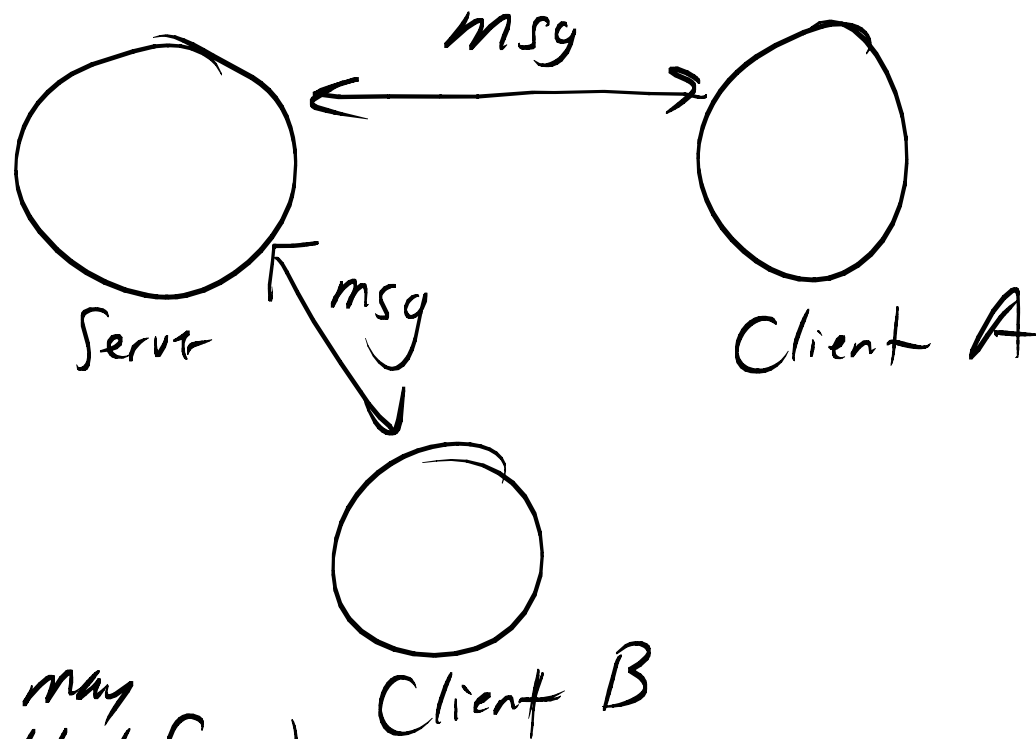


Problem: low level  
lots of set-up

Better:  
Appl.  
Developer  
model

- hide connection setup
- support concurrent clients  
(some clients may be slow or block forever)



Design a client/server appl. framework:

Asynchronous  
model

What should client developers do?

- define message types [classes  $\Rightarrow$  Object streams]
  - • GUI for user (sitting on top)
  - way to process messages received
- API design choice {
- ① active loop to read incoming messages in app.  
 $\Rightarrow$  `Message receive();`
  - ② react as we're told about messages (event-driven)  
 $\Rightarrow$  `void messageReceived(Message m);`
- active behavior of client (including sending messages)  
(optional, but allow & support Runnable client objects)
  - `send(Message m)` — sends `m` to the server (nonblocking)

• way to process messages received

API design choice

- ① active loop to read incoming messages in app.  
⇒ `Message receive();`
- ② react as we're told about messages (event-driven)  
⇒ `void messageReceived(Message m);`

①

Client sends m  
Client waits for reply

or

Client sends m,  
sends m2  
Client waits for  
reply 1  
reply 2

②

Client sends m  
⋮

①

AbstractClient

Blocking

Reactive

②

meanwhile,  
client handles incoming messages from server

	Vote
①	0
②	8
③	→ 8

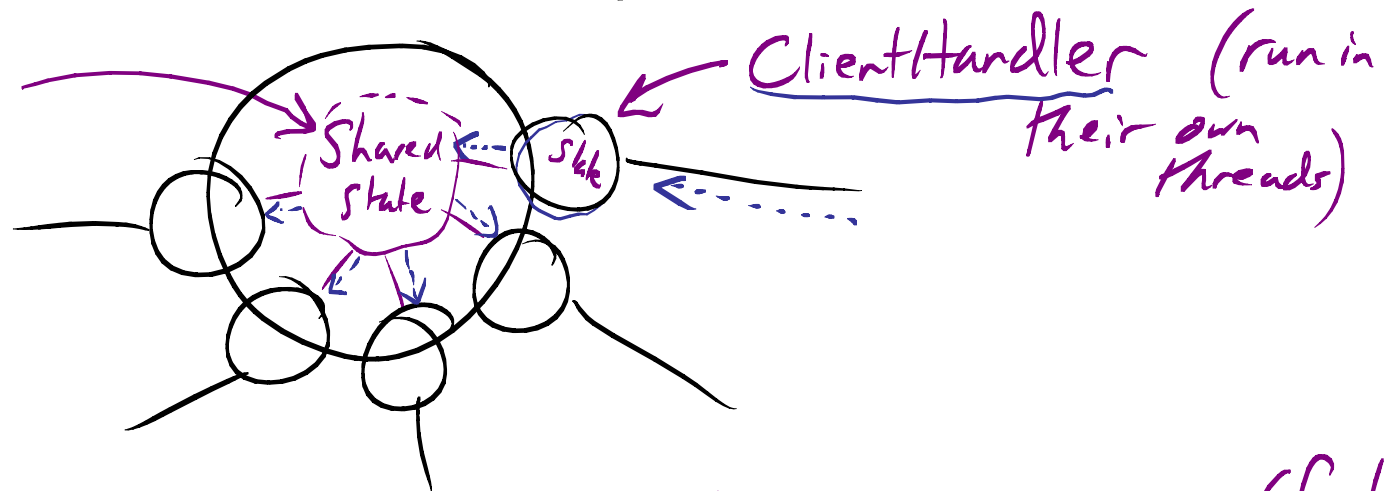
If both are available, using both simultaneously would be awkward: Don't know whether blocking call or msg handler should get the next msg.

# Server side:

Server developer will provide:

- mechanism for handling each client

should be thread safe



• need a way to create ClientHandler objects

(factory method)

same question

- way of handling messages received
- sending messages
- active? (Runnable?)

Runnable?

• Server Active computation (or just passive?)

# Client

① does it run? — does it actively do things

⇒ whether or not it's runnable

② how does it receive messages?

Active client —  
explicitly calls receive()

Reactive client  
implements messageReceived(M msg)