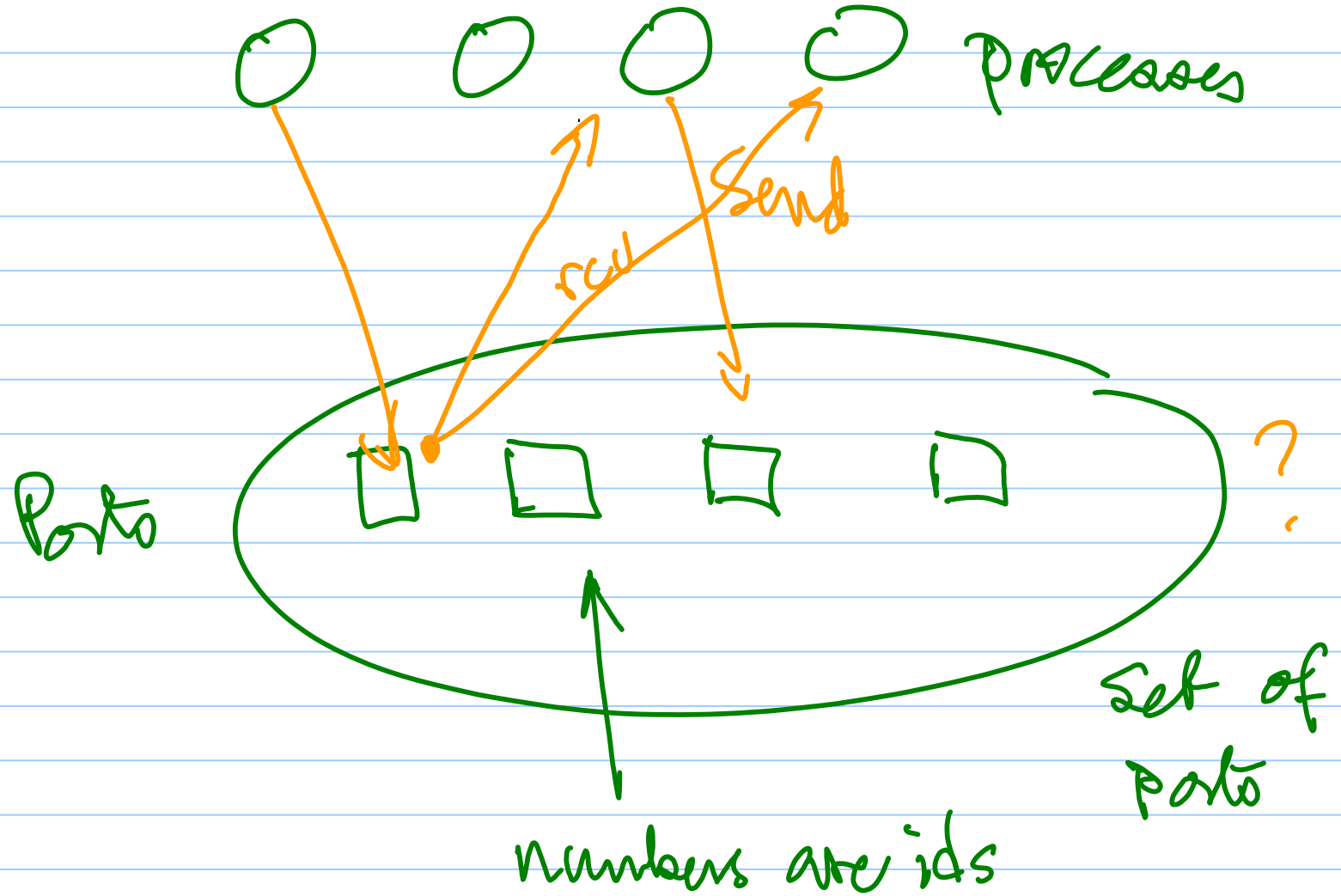
CSE 531: Distributed and Multiprocessor Operating Systems

Fall 2020

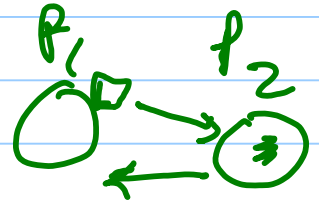
Class: 18 Date: 10/28

Note Title

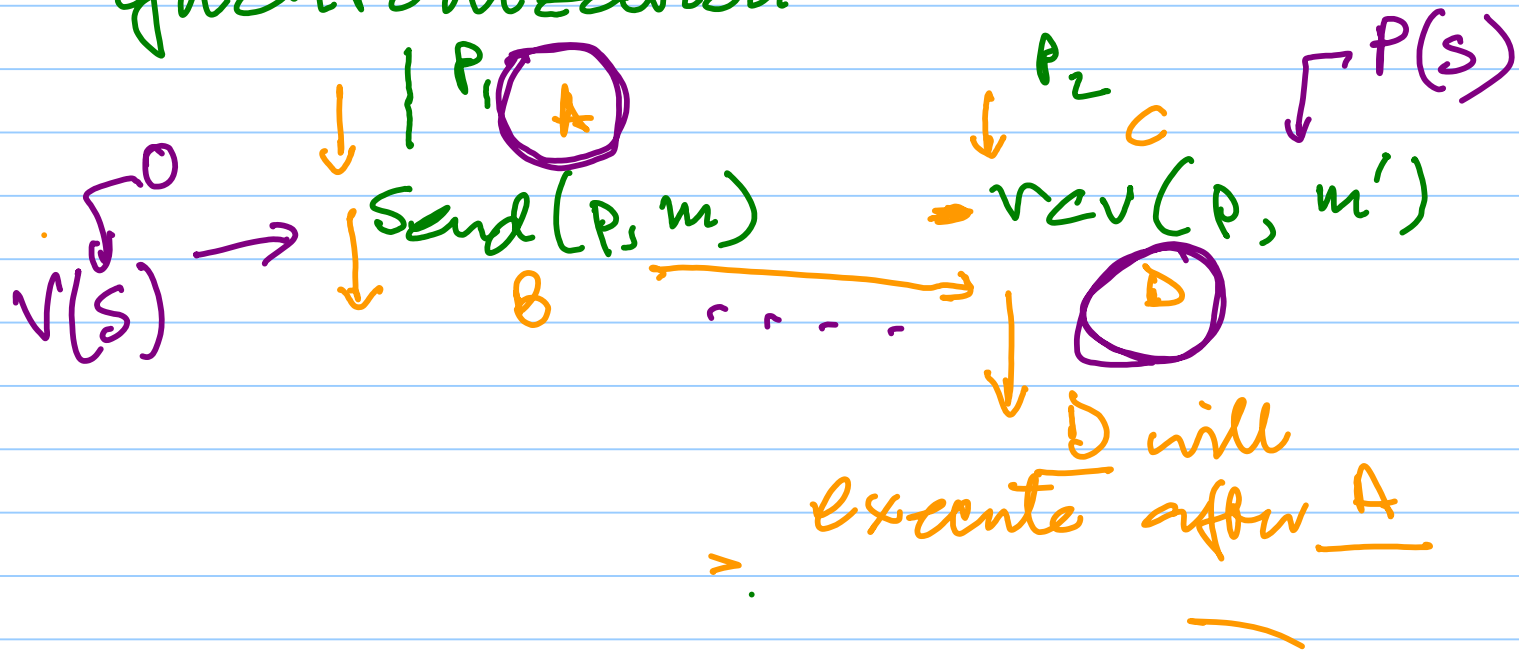


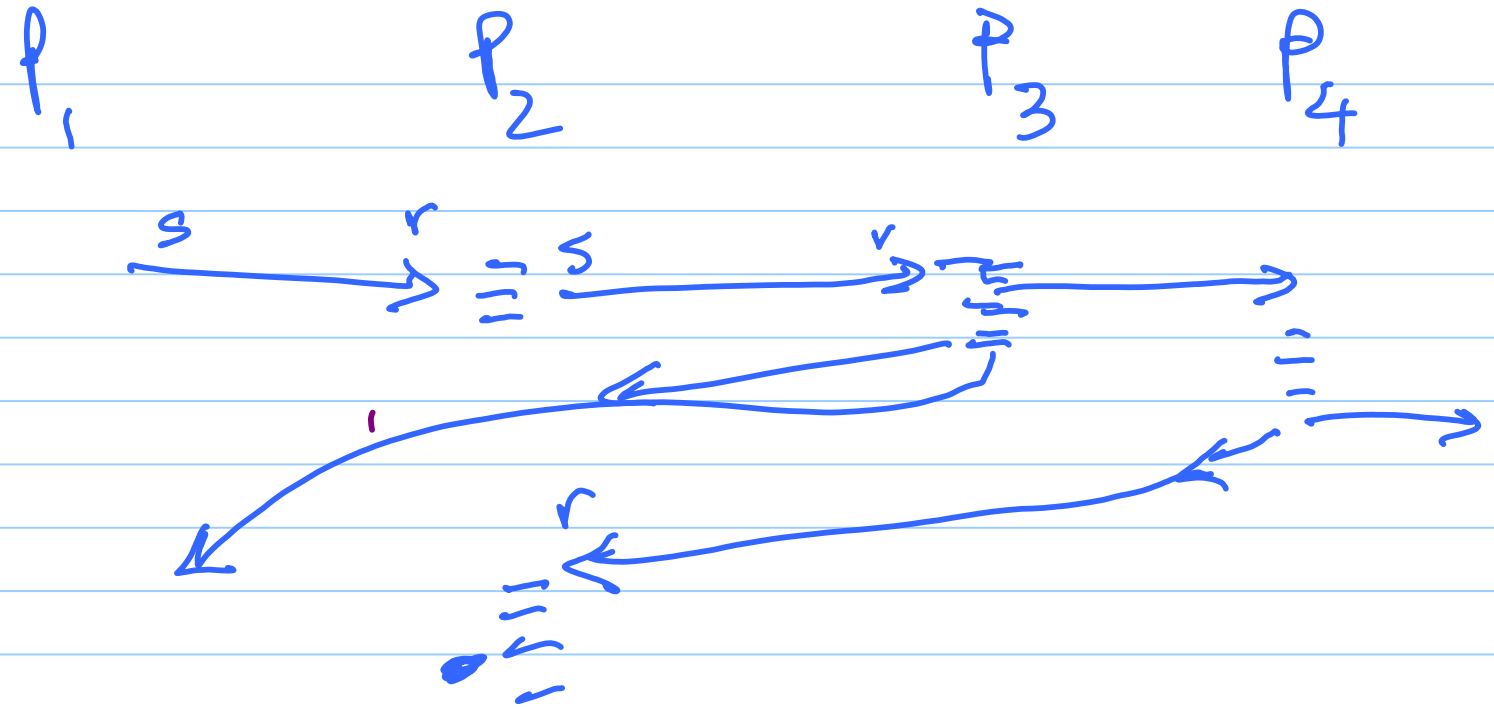
- Messages provide

① Data sharing →



② Synchronization

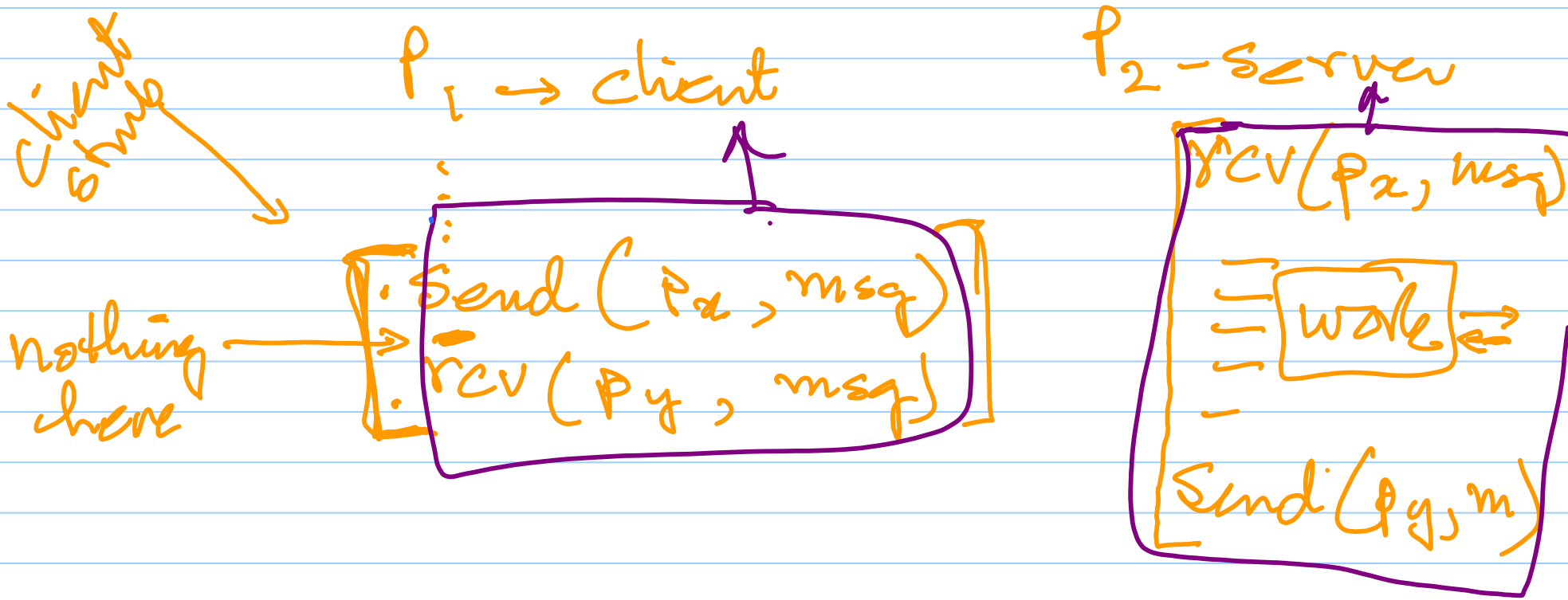


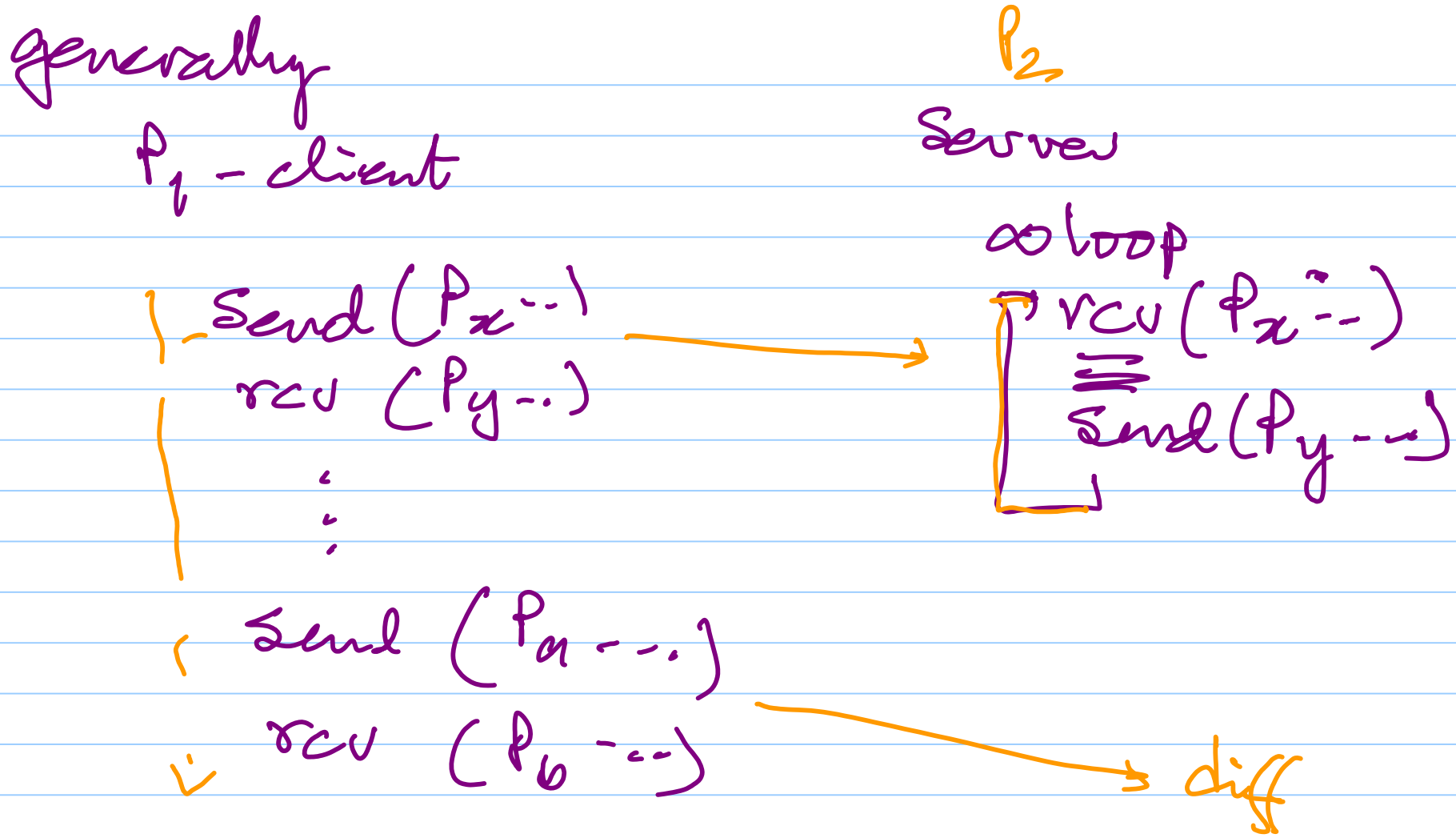


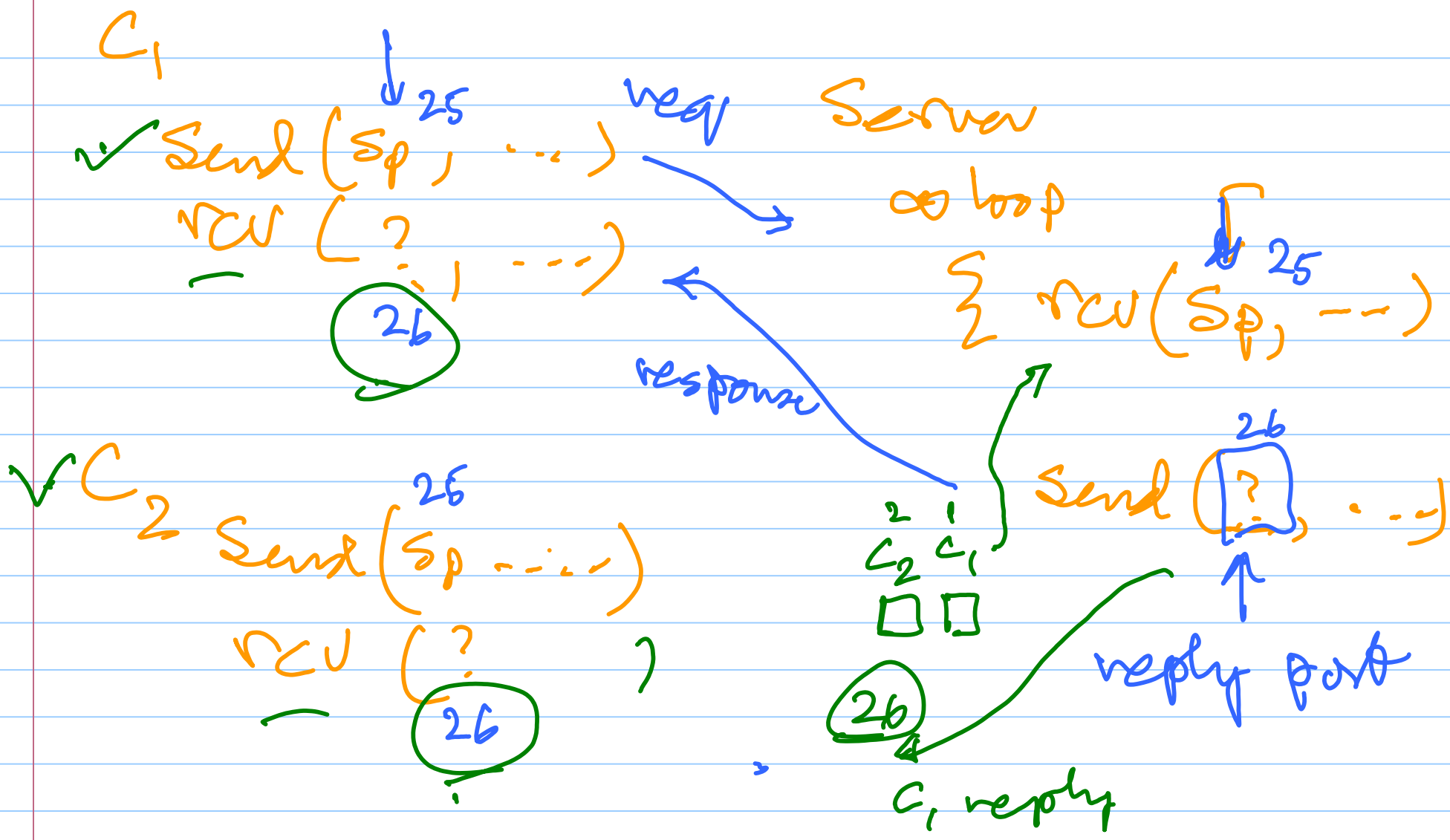
- unstructured
- like "goto"s

programming with msgs

→ must use client-server structure




generally
P_1 - client
P_2
Server
∞ loop
send (P_x ...)
rcv (P_y ...)
rcv (P_x ...)
send (P_y ...)
send (P_a ...)
rcv (P_b ...)
diff


C1
Send(sp, ...)
25
req
Server
∞ loop
{ rcv(sp, --)
25
rcv(?, ...)
26
response
C2
Send(sp ...)
26
rcv(?)
26
C2 C1
Send(?, ...)
26
reply port
26
C1 reply

Server port

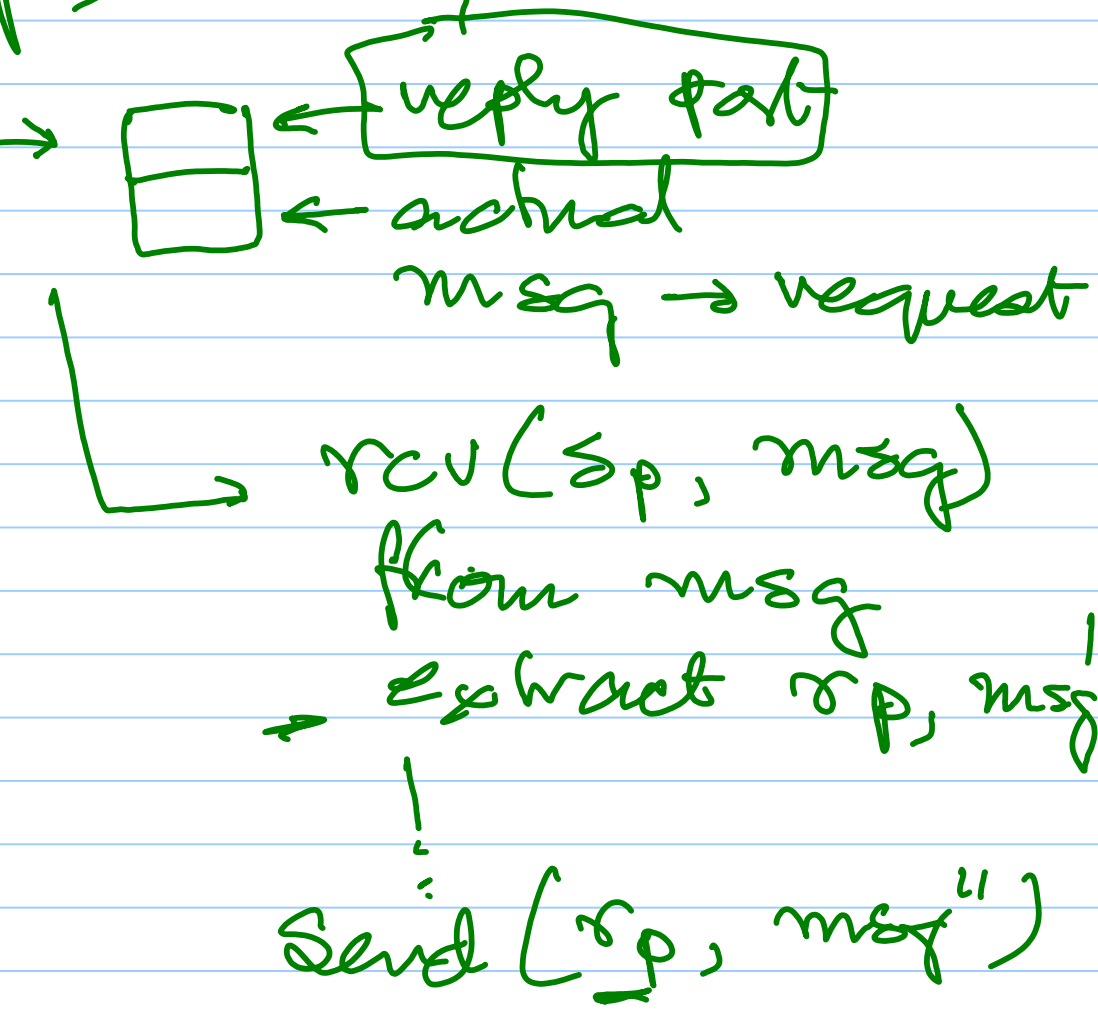
→ one well known port per server

reply_port

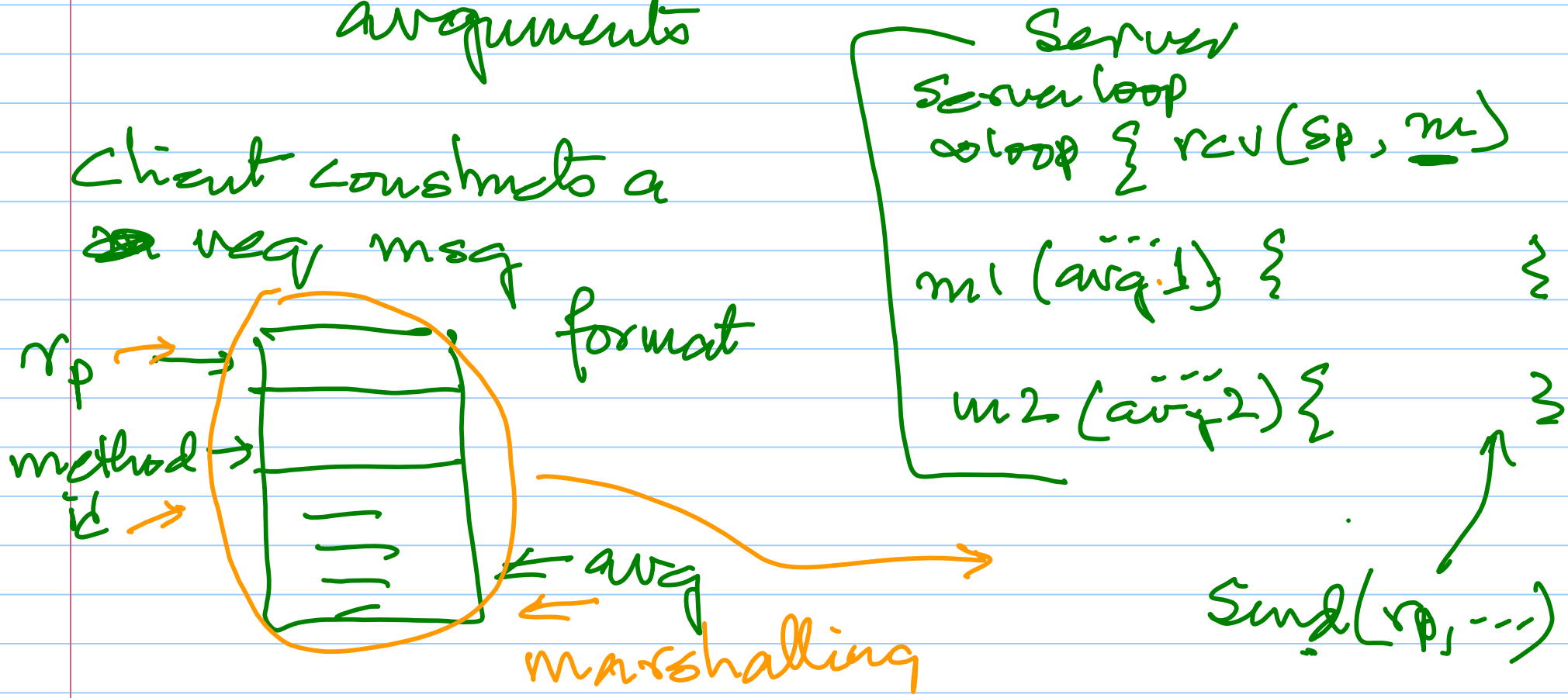
→ every client has a "private" reply port

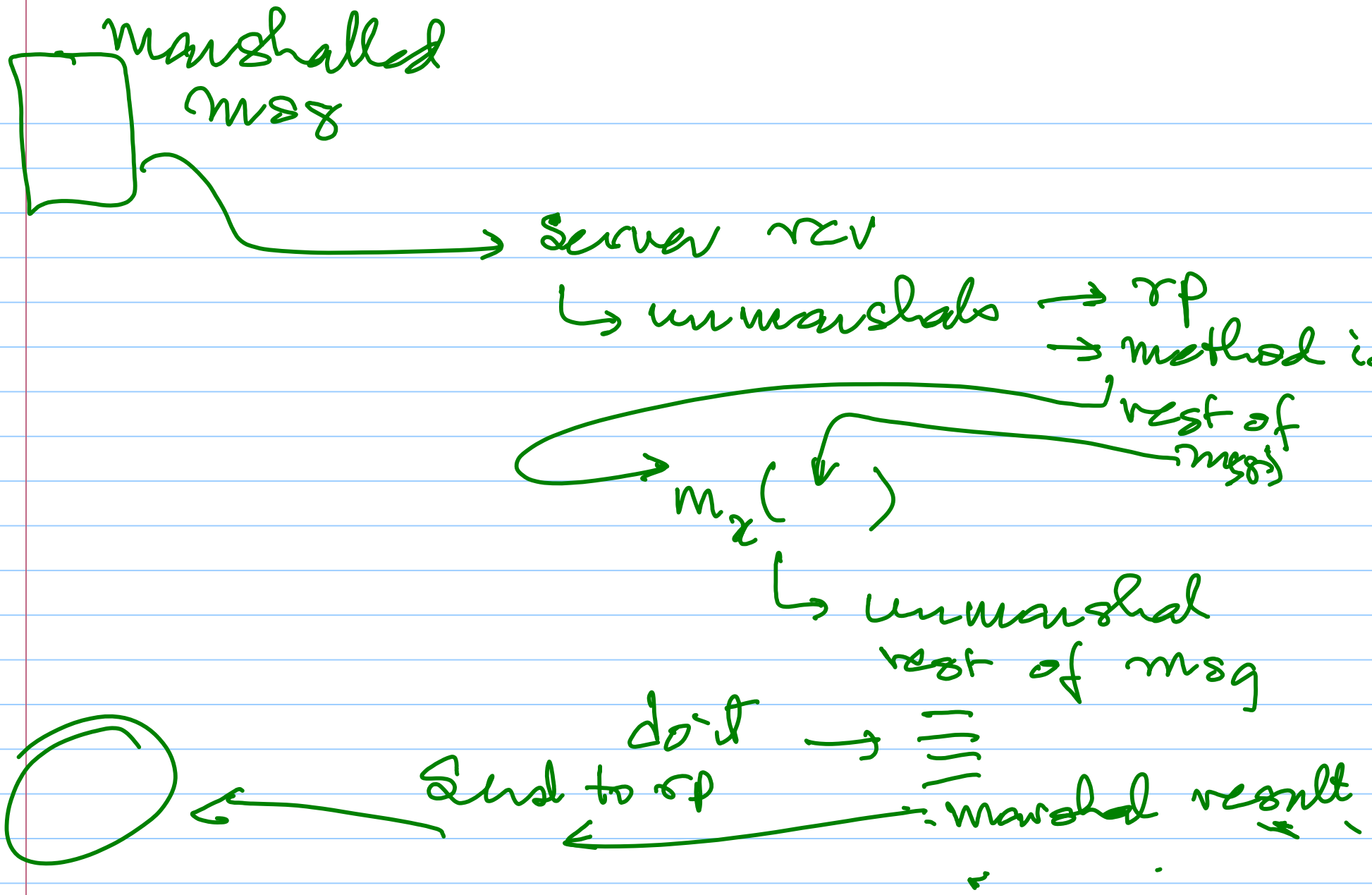
? = allocate (rp)

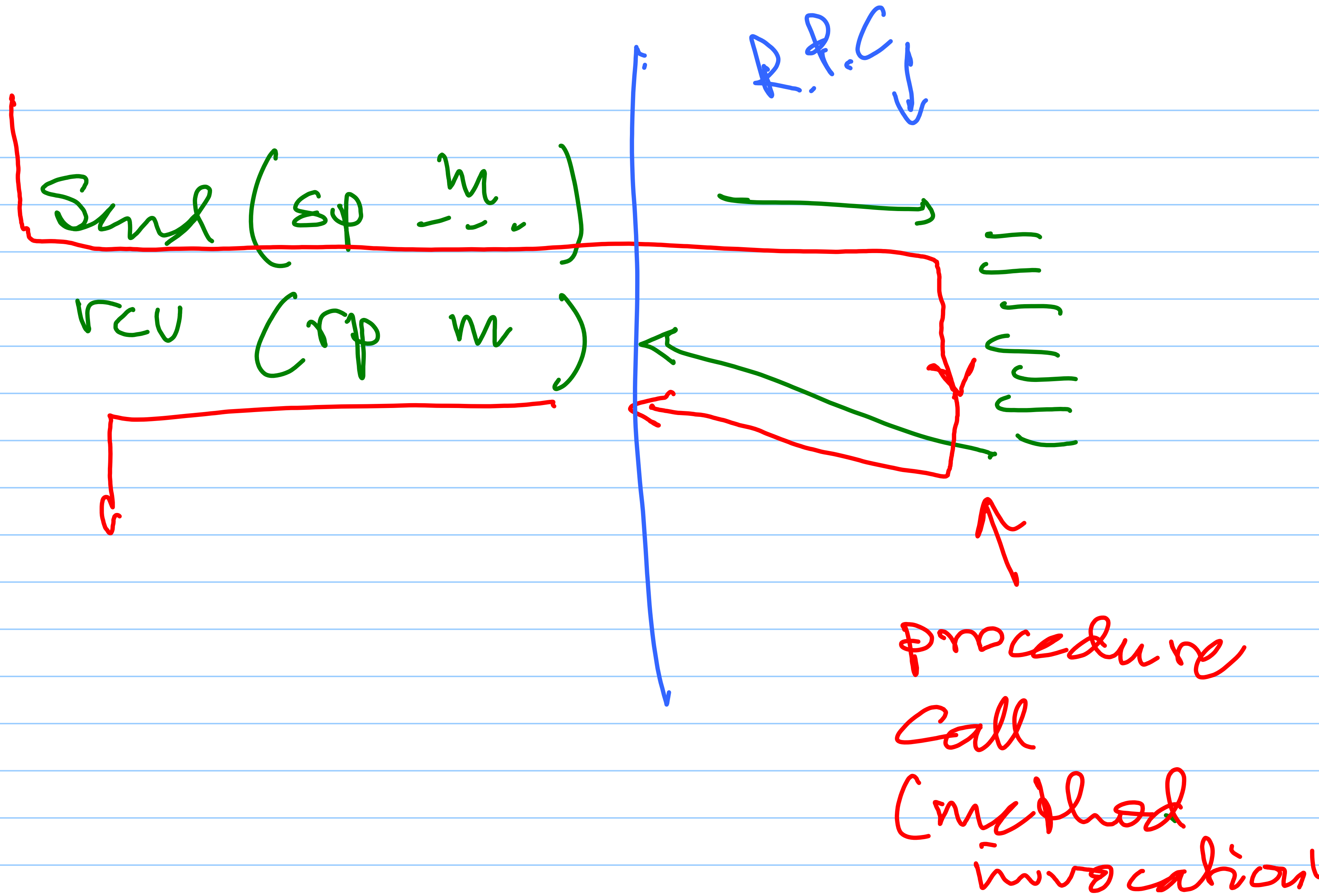
Send (sp, msg)



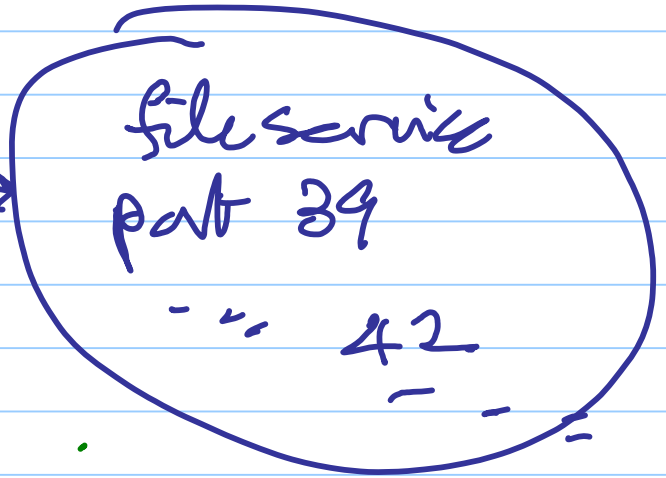
A server is an object
- it has methods & arguments



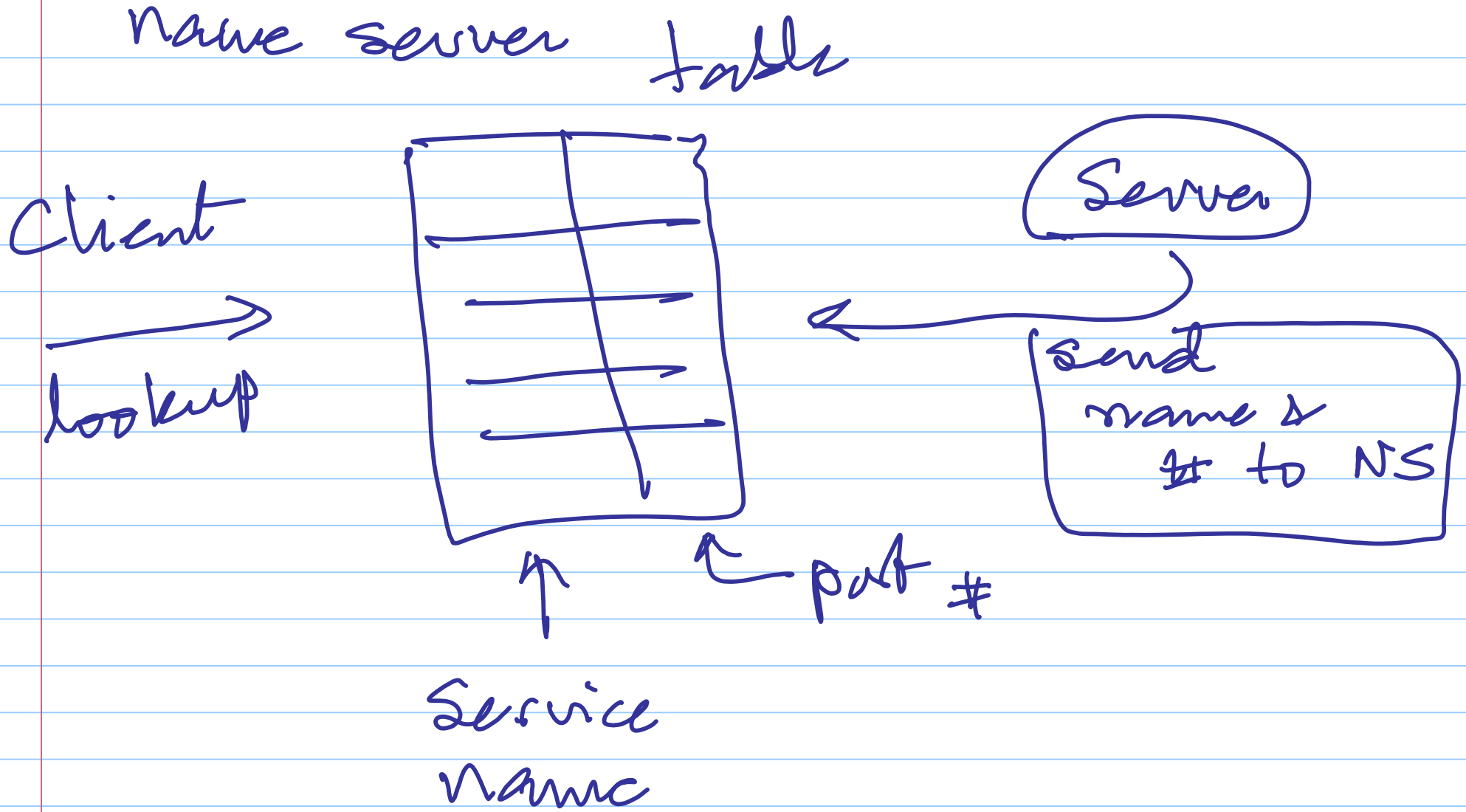

marshalled msg
server rcv
↳ unmarshals → rp
→ method id
rest of msg
m_x( )
↳ unmarshal rest of msg
do it →
send to cp
marshd result


R.P.C
Send (sp m ...)
rcv (rp m)
procedure call
(method invocation

naming → how do clients know server port?

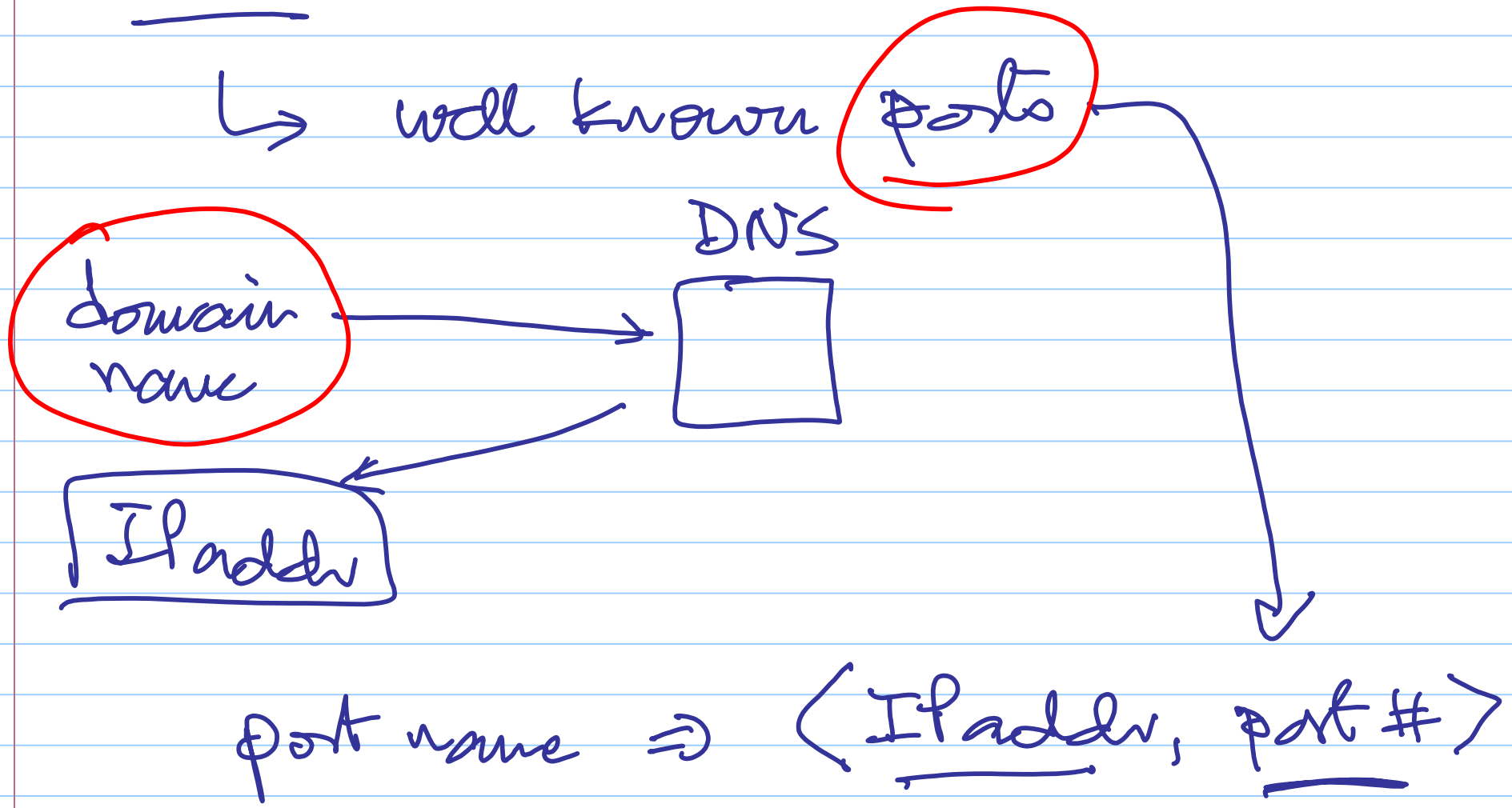
- well known ports → 

- name server
  - → one server @ a well known port → 0
    - →


Name server table
Client
lookup
Server
send name & # to NS
port #
Service name

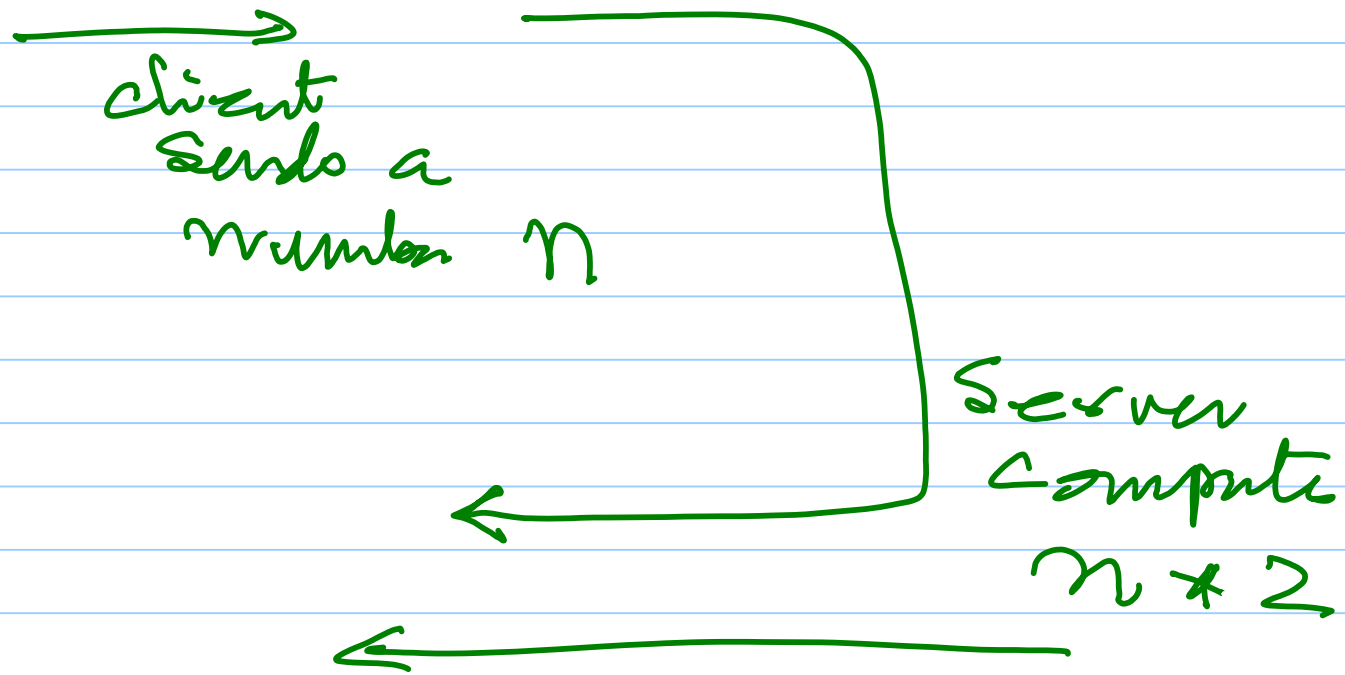
TCP IP

↳ well known ports



port name ⇒ ⟨IPaddr, port #⟩

Simple server



allocate mp

...

Send(&p, msg, ...)

Recv(&p, msg, ...)

2n

Recv(&p, msg, ...)

n <p2

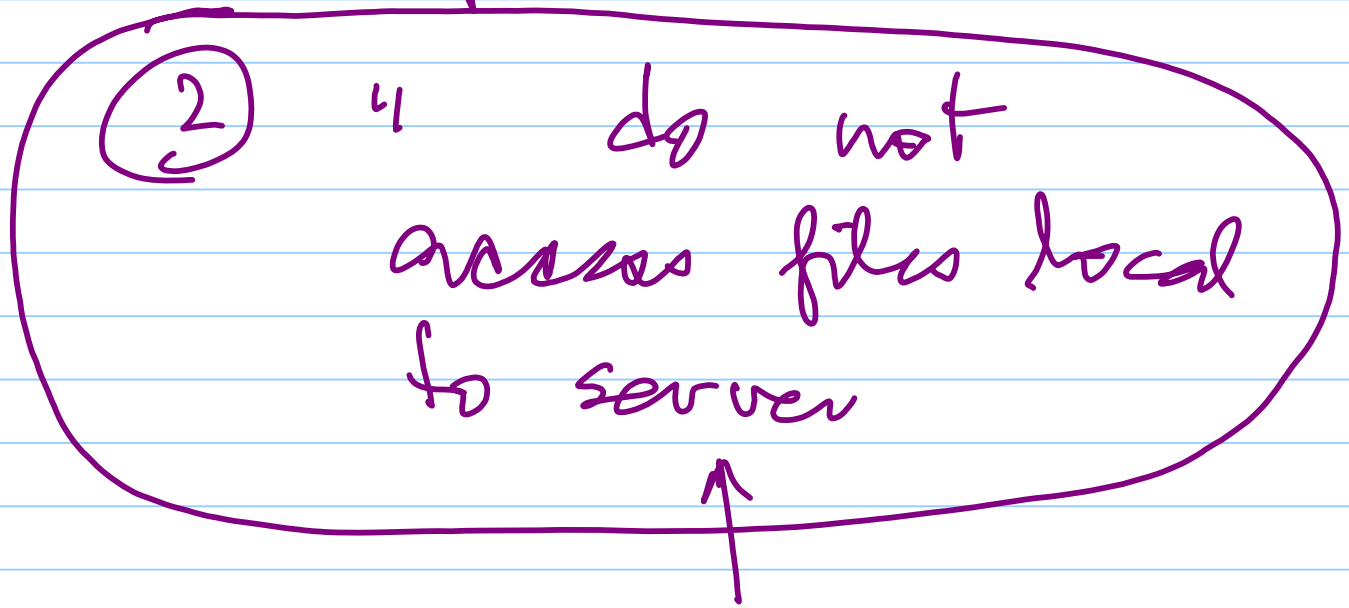
result = 2 * n

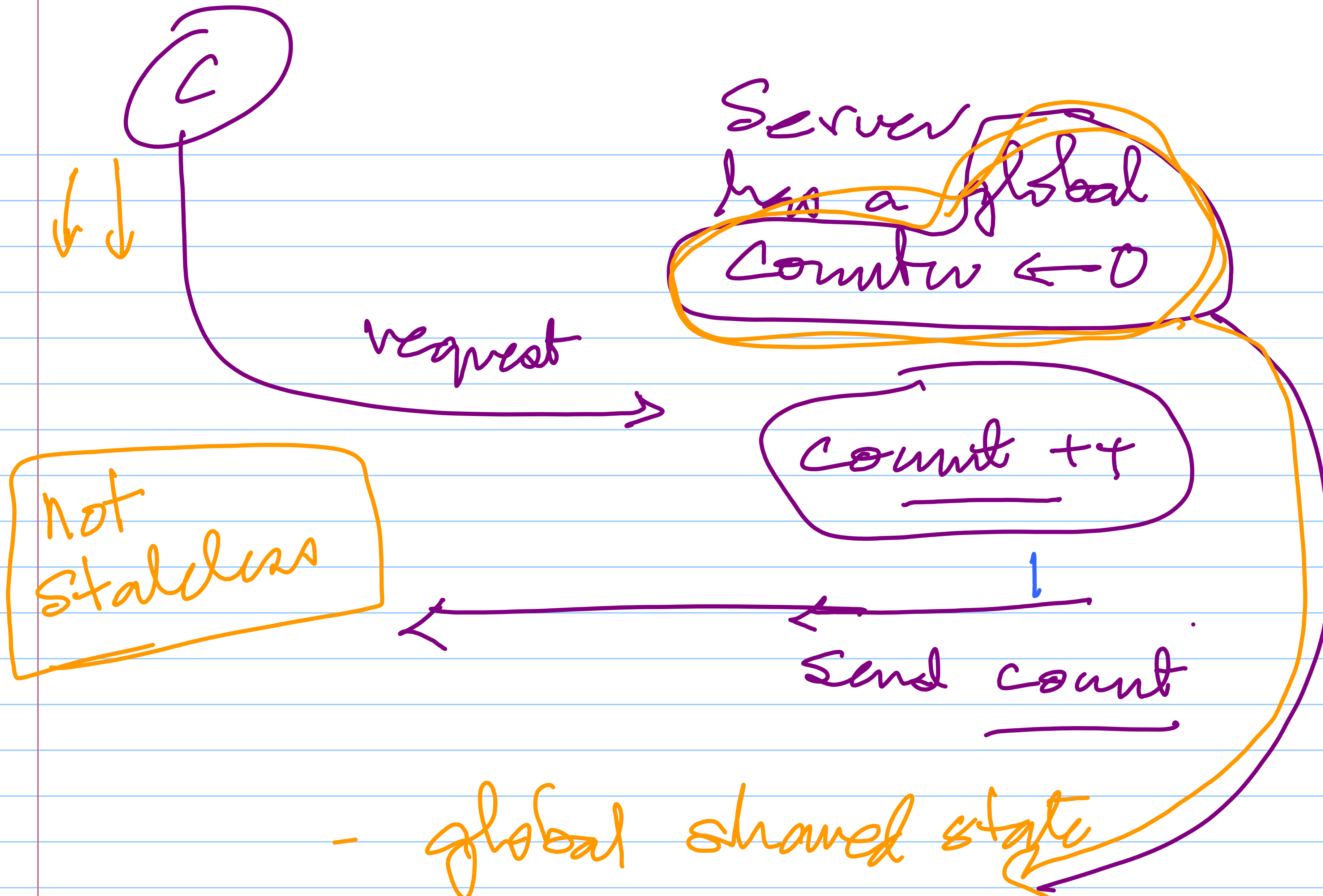
Send(&p, result, ...)

Server can be folded into clients if

① they are stateless

② " do not access files local to server




C
request
Server
has a global
Counter ← 0
count ++
1
Send count
not
stateless
- global shared state

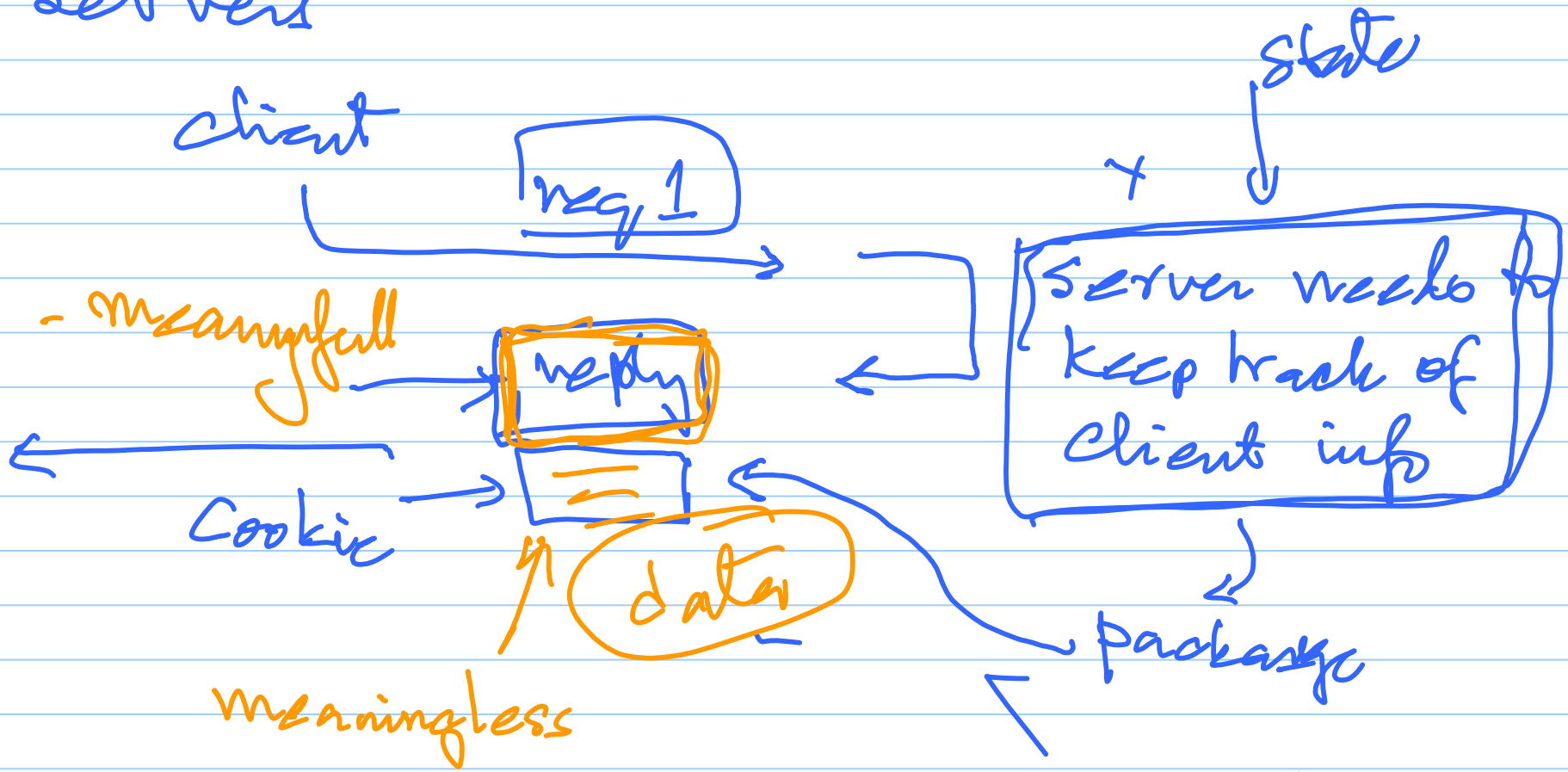
Stateless servers -

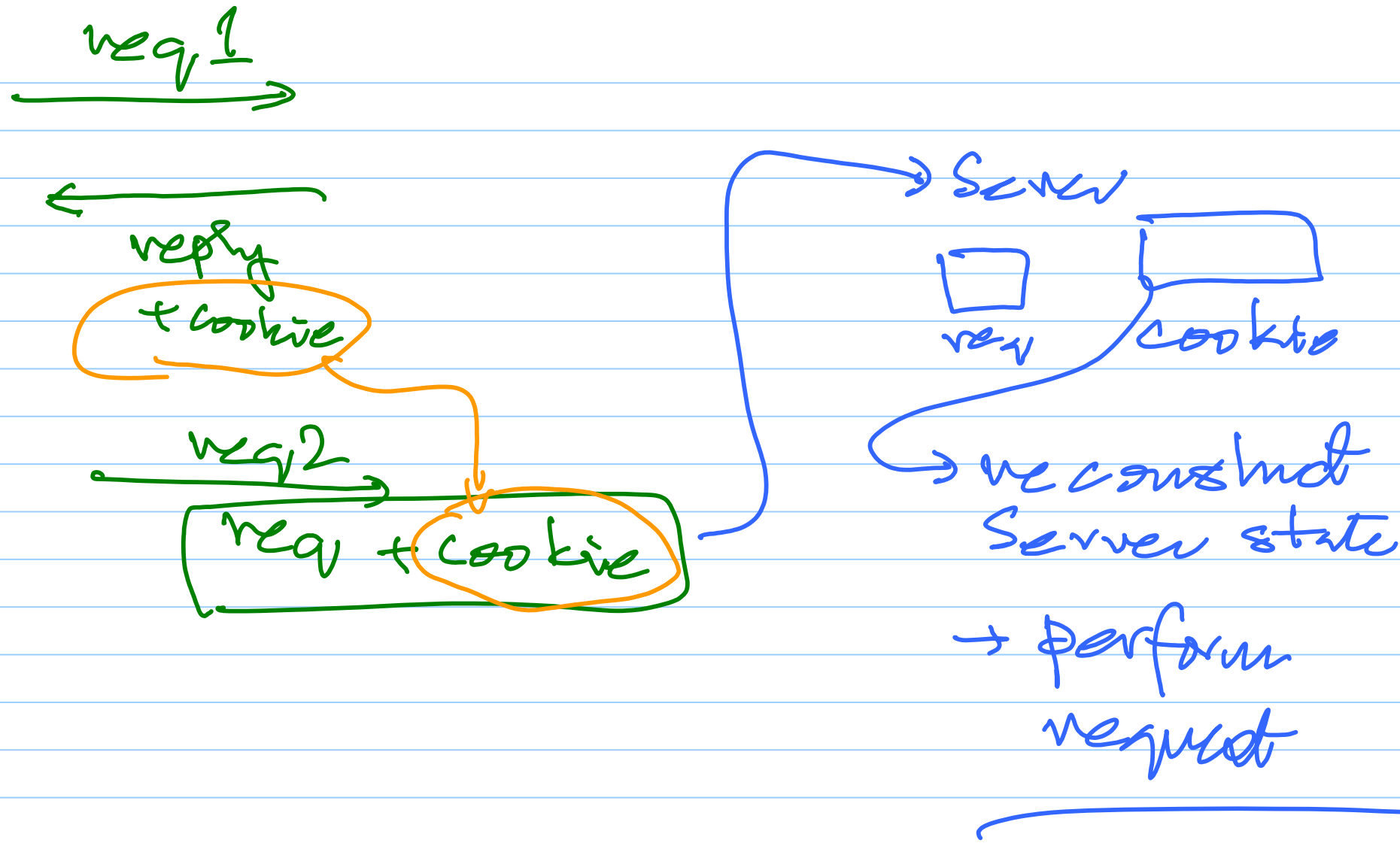
① → no state.

② → fail → recover → no problem

③ → replicate

- Using Cookies in stateless Servers




req 1
reply
+ cookie
req 2
req + cookie
Server
req
cookie
reconstruct
Server state
→ perform
request

Server requests → procedure calls

→ Can we nest servers?

↳ yes/no

$C_1 \rightarrow S_1 \Longrightarrow S_2 \rightarrow S_3$

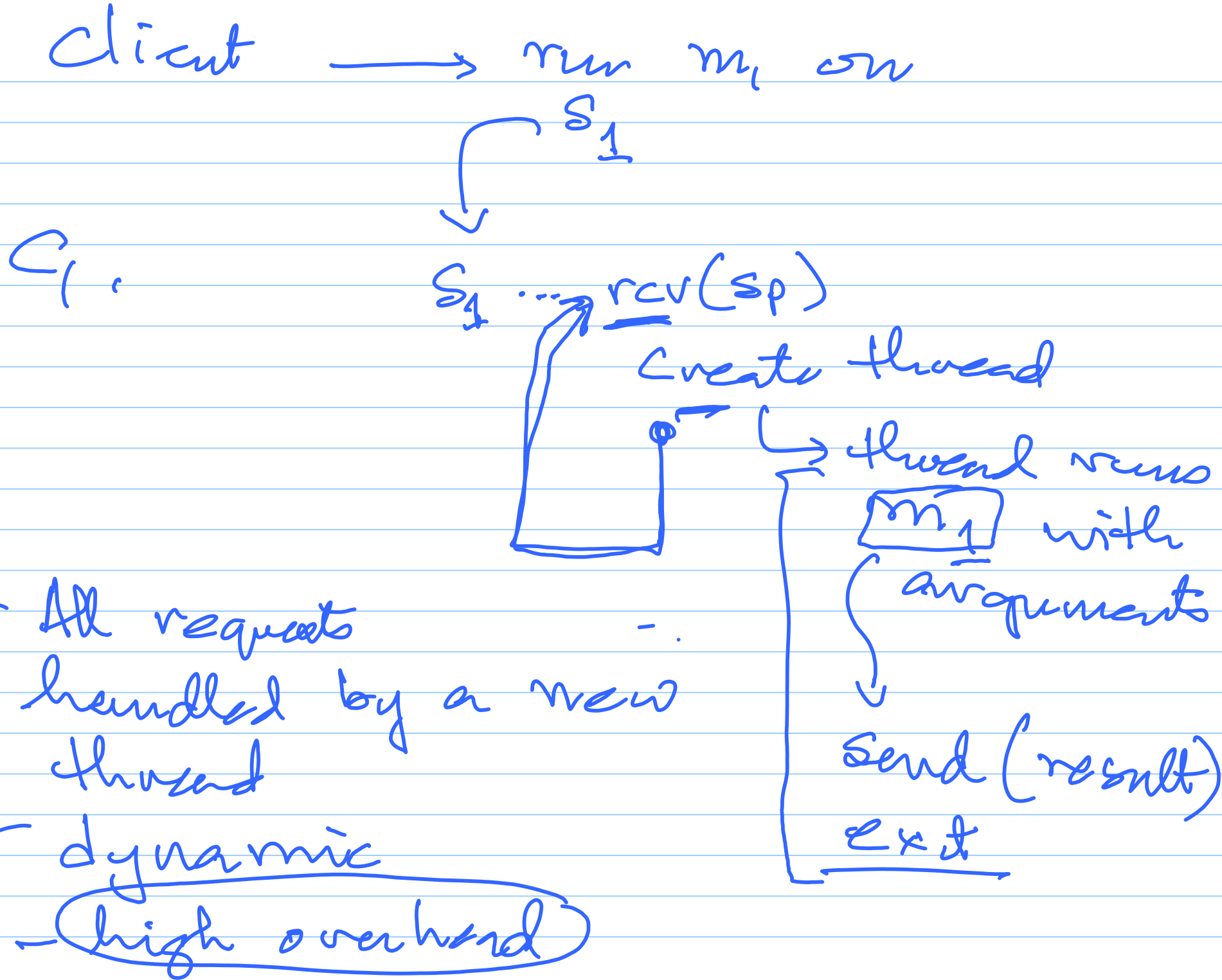
C1
C2
S1
send(sp...)
rcv(sp...)
rcv(sp...)
send(S2p,...)
rcv(rp',...)
block
send(rp',msg)
Client stub
Marshal
Call to S2

- nesting OK, but poor performance
  - ↳ server blocks
- recursion

$$S_1 \rightarrow S_1$$

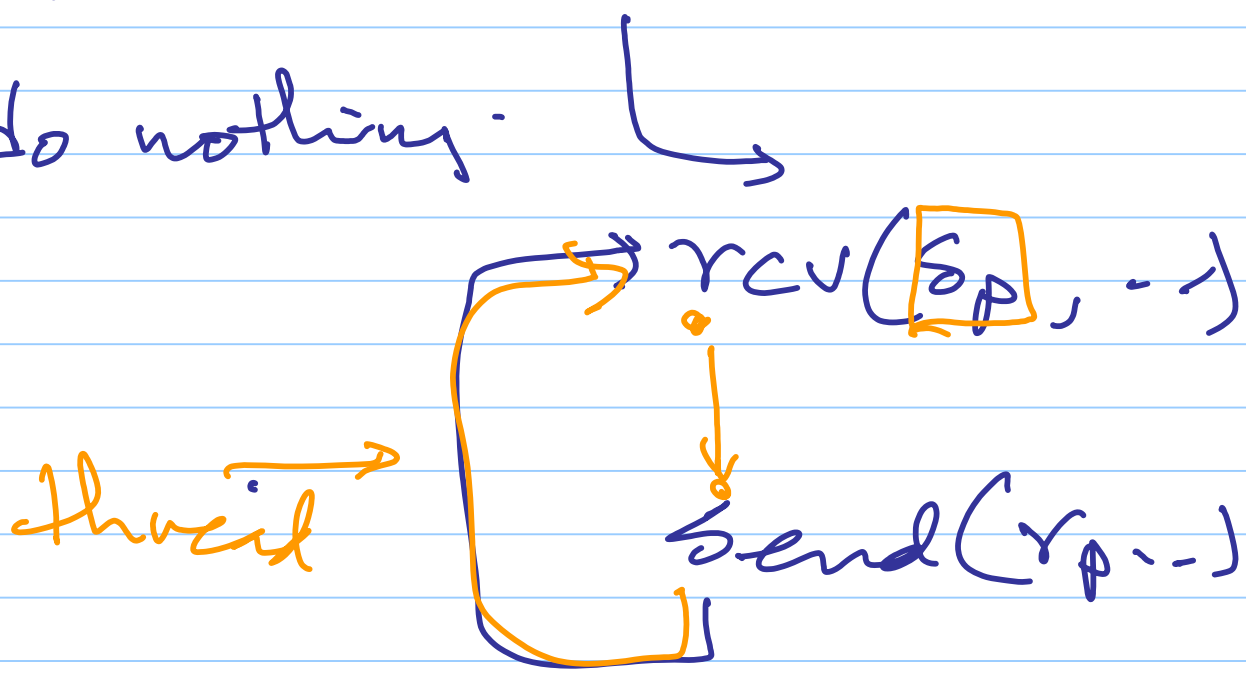
$$S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_1$$

multithreading


client
run m_1 on
s_1
c_1 .
s_1 ... rcv(sp)
create thread
thread runs
m_1 with
arguments
Send (result)
exit
- All requests handled by a new thread
- dynamic
- high overhead

static multi-threading

Server
{ start n threads
  //do nothing.
}



= multithreading

stateful server

↳ race conditions on global

↳ Semaphores or locks needed

how many threads?

Static