- Message passing systems
  - TCP-IP - standard messaging protocol, sockets
  - MPI has its own msg passing system
  - msg passing over TCP-IP → many

Send (port, data) → blocking
→ non blocking

rcv (port, data) → blocking or not

- Sends do not block → unless if port is full
- rcvs block if no msg available

→ non blocking msg passing

P_1 (client)

P_2 (server)

sends a msg with an int x →

← respond with 2x

what about port naming → 1 - 100 (static)

P1

```
x = value
client [ send(1, x)
         rcv(1x2, result)
```

P2

```
server [ rcv(1, data)
         data = 2*data
         send(1x2, data)
```

ports are for unidirectional comm.

multiple client

P_1

send(1, mydata)
rcv(2,)

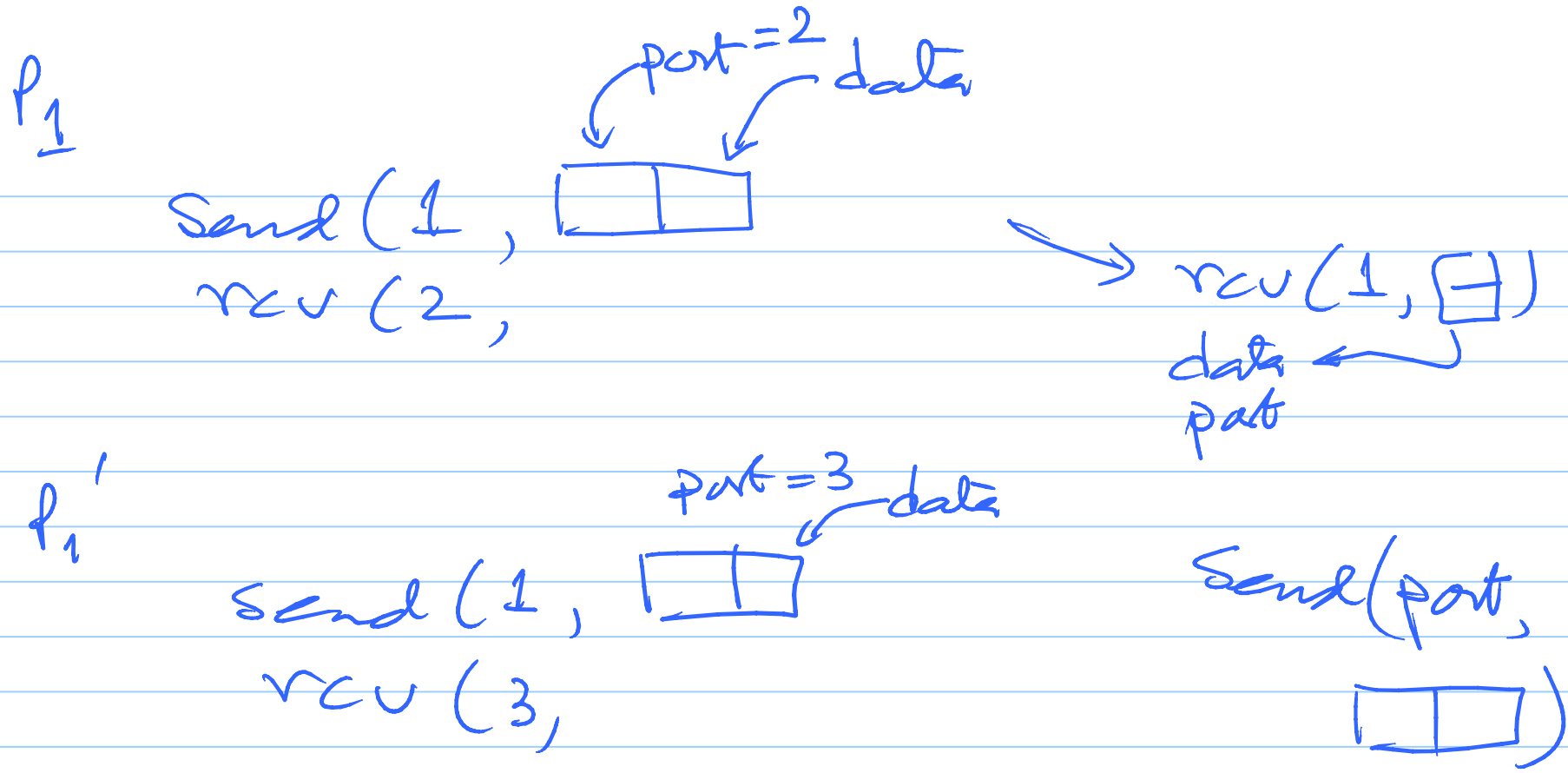
P_1' → send(1, mydata)
rcv(2, result)

P_1''))

single

P_2

∞ loop
{ rcv(1, data)
result = ...
send(2, --)


P_1
port=2
data
send(1,
rcv(2,
rcv(1,
data
port
P_1'
port=3
data
send(1,
rcv(3,
send(port,

Send(port, replyport, data)

rcv(port, replyport, data)

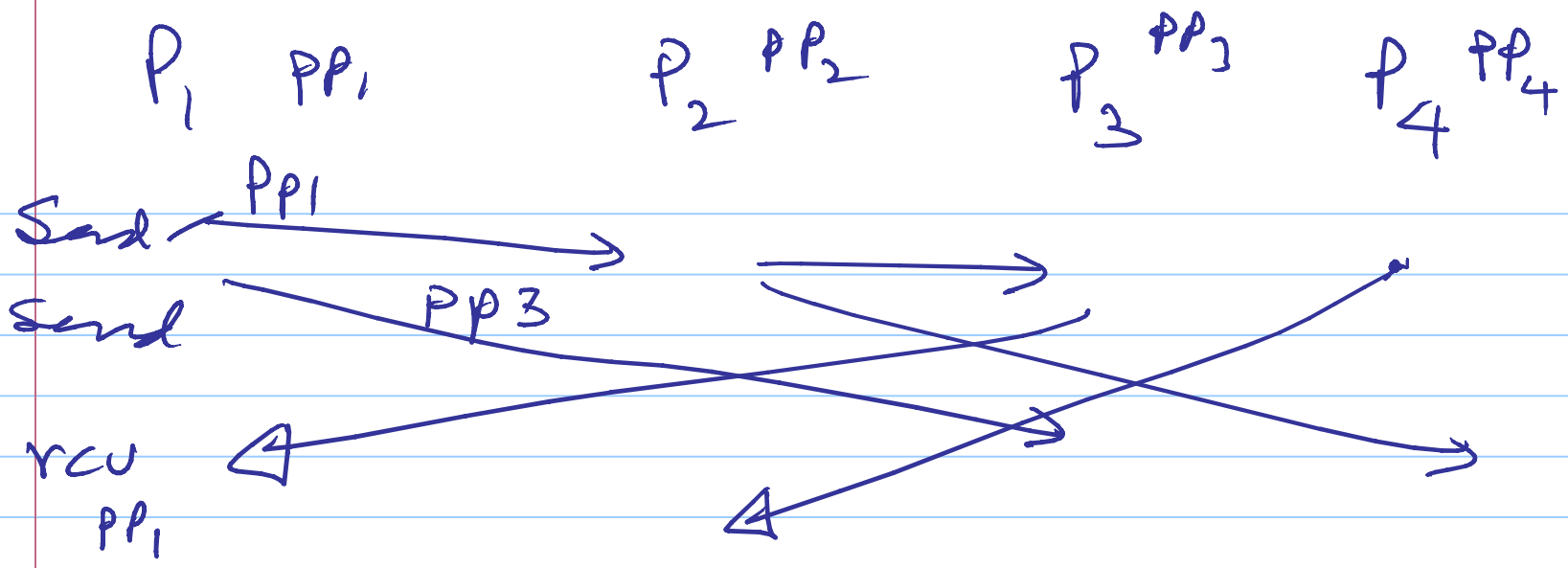
rcvd

- ports are unidirectional
- ports "belong" to receiver
  (can have multiple rcvrs owning same port)
  - Can be used as client id

programming with messages

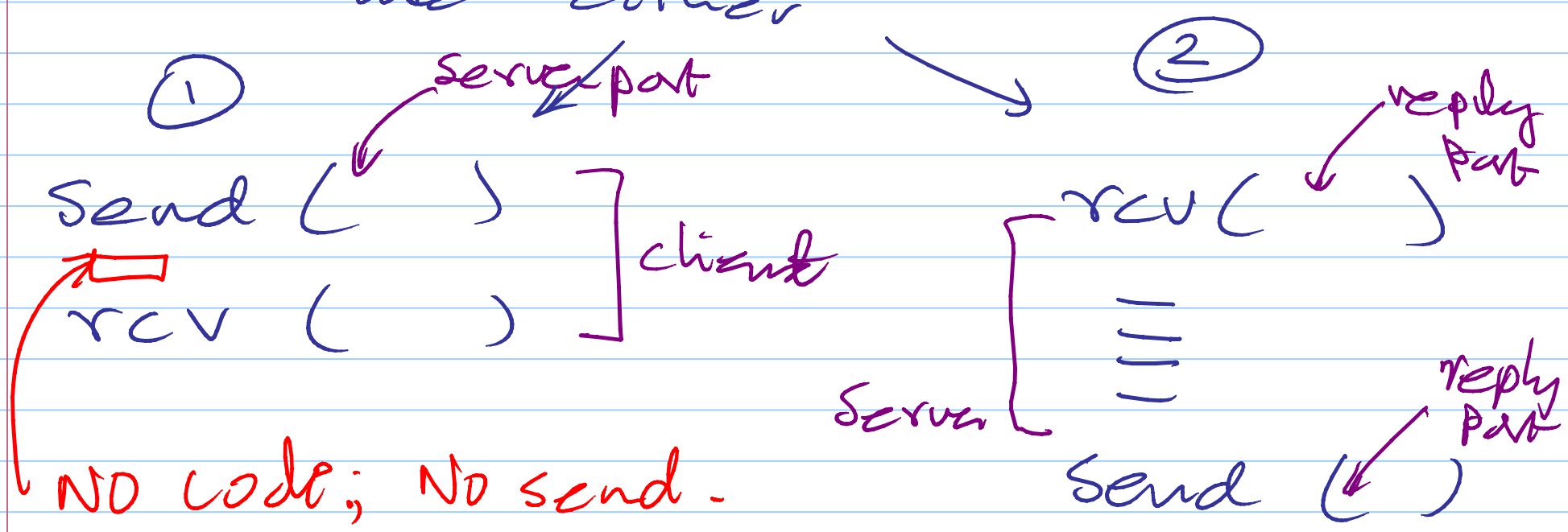
- centralized

- need structured


P_1 PP_1
P_2 PP_2
P_3 PP_3
P_4 PP_4
Send
PP_1
Send
PP_3
rcv
PP_1

structured msg passing programming (client-server)

use either



client – server program

Send – non blocking (OK)
blocking better

rcv – BLOCKING
non blocking → NO!
timeout → NO!

multiple clients (per server) → reply port

→ multiple servers

↳ • clients have to know server ports for all servers

• each server has to have a unique port

Nameserver (+createport)

↳ nameserver is a servers

- other servers register a name & a port → to nameserver
- other clients request a port for a name

Server "x"

sp = createport(),

Send (ns_port, sp,

register, x)

rcv (sp, _) ← OK

Nameserver

∞ loop

{ rcv (ns_port, reply,

well known port

request_type + data)

if register

add name + sp to table

Send (sp, OK)

Nameserver

Client

if lookup {

for name

search table ^ → port)

send (NS port,

replyport,

name)

send (replyport, port

rcv (replyport,

server port)

- Server register names + port

- clients lookup servers by name
  & get port

  - use port to contact
    servers.

TCP - IP

domain name ⟶ IP address
(DNS service)

- IP address has many servers
  - Servers → well known ports

Structure of message passing programs

client

get port num of server1
" " " " server2

mine = create port

[Send request to server1
rcv reply] ← remote proc call

also others

marshalling

- reply port
- function type
- arguments

Send(sp, request)

rcv(mine, reply)

unmarshalling

values of fields for function

Server is an object

global state
+ init code

register with nameserver

∞loop

rcv(sp, request)

unmarshalled
↳ reply port
↳ function → arg

switch

f1 : f1(args for f1)

send (reply-part

return values)

marshalling

f2

Similar

f3