Exam #1

Wednesday, October 1

at 7:00 pm

Closed books and notes.

The coverage is posted

A sample exam is posted

COVERAGE FOR EXAM #1

CS 586; Fall 2025

Exam #1 will be held on **Wednesday, October 1, 2025,** at **5:00 p.m.**
Location: 111 Stuart Building
The exam is a **closed-book and notes** exam.

Coverage for the exam:

- Modular design, information hiding, coupling, and cohesion.
- Object-oriented concepts: abstract data type, data encapsulation, inheritance, polymorphism, static and dynamic binding.
- Object-oriented design. Relationships between classes: aggregation, association, and inheritance. Class model. Sequence diagrams.
- OO design patterns: item description, whole-part, observer, state, proxy, and adapter patterns. [Textbook: Sections 3.1, 3.2; Section 3.4 (pp. 263-275); Section 3.6 (pp.339-343); Handout #1, class notes]

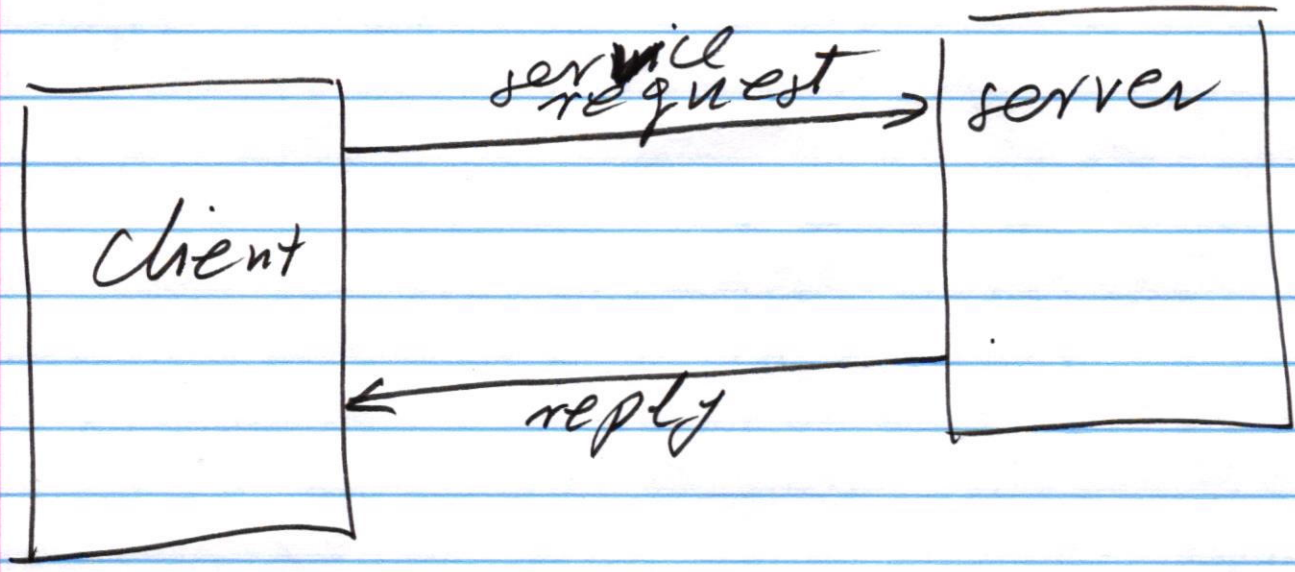
Sources:

- Textbook: F. Buschmann, et. al., Pattern-oriented software architecture, vol. I, John Wiley & Sons.
- Handout #1
- Class Notes

Client - Server Architecture

Servers: provide services

Clients: request -11-



Problem

Client wants to get a servie.

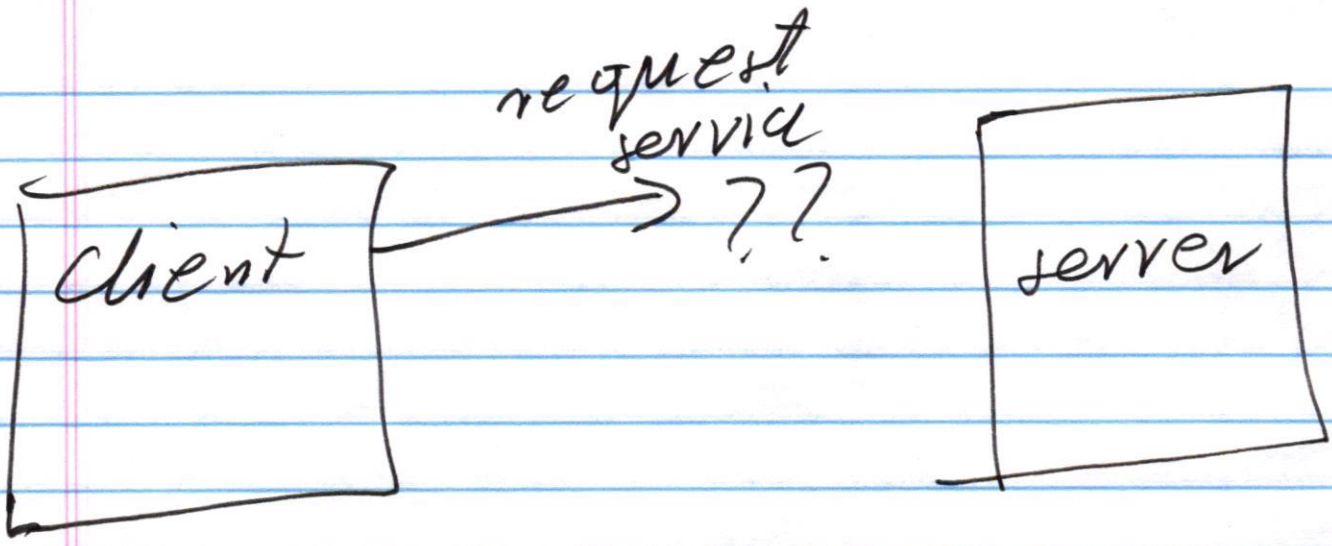
Where is the server that can provide the servie?

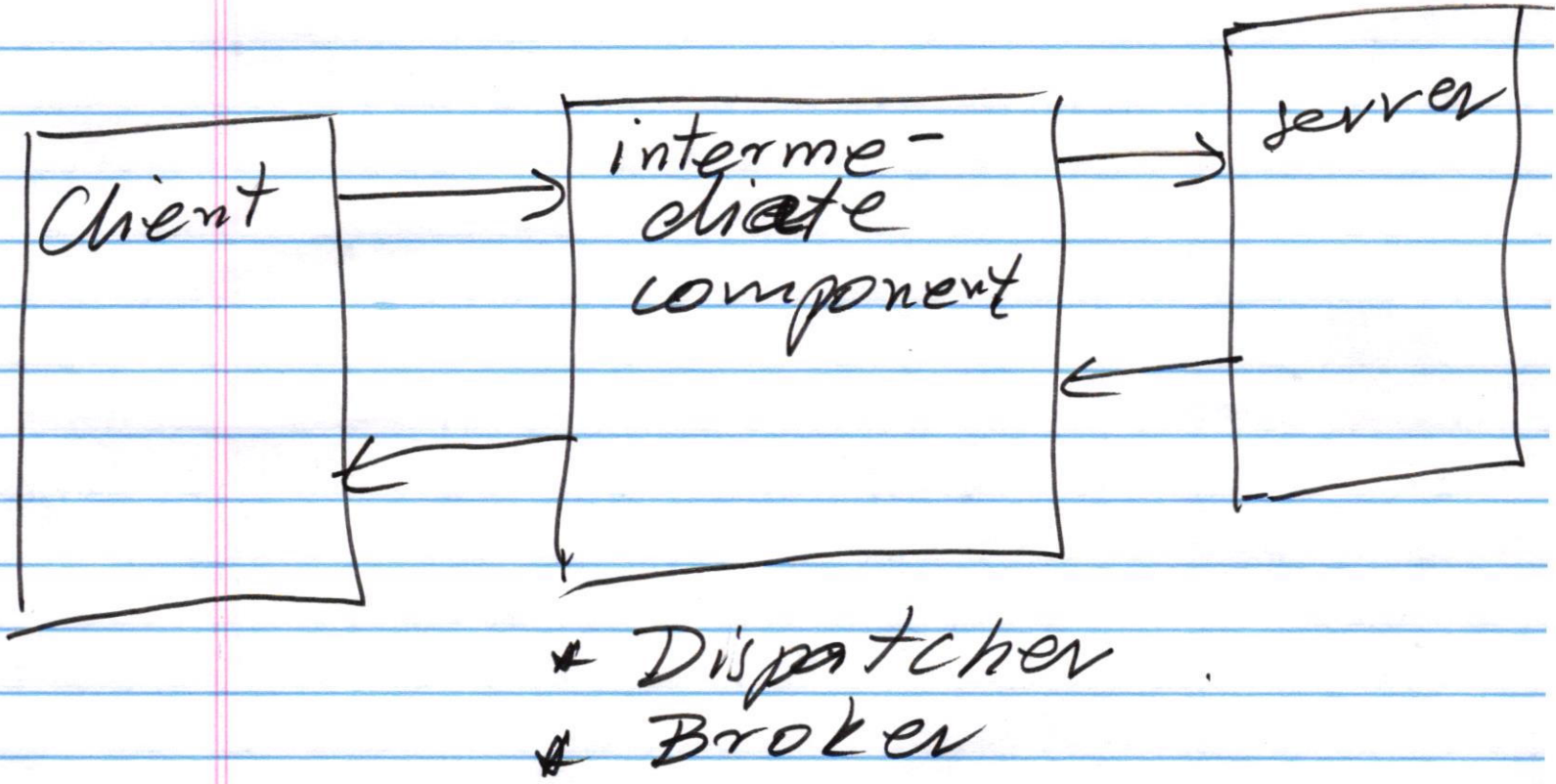
OO systems

servers are dynamic objects

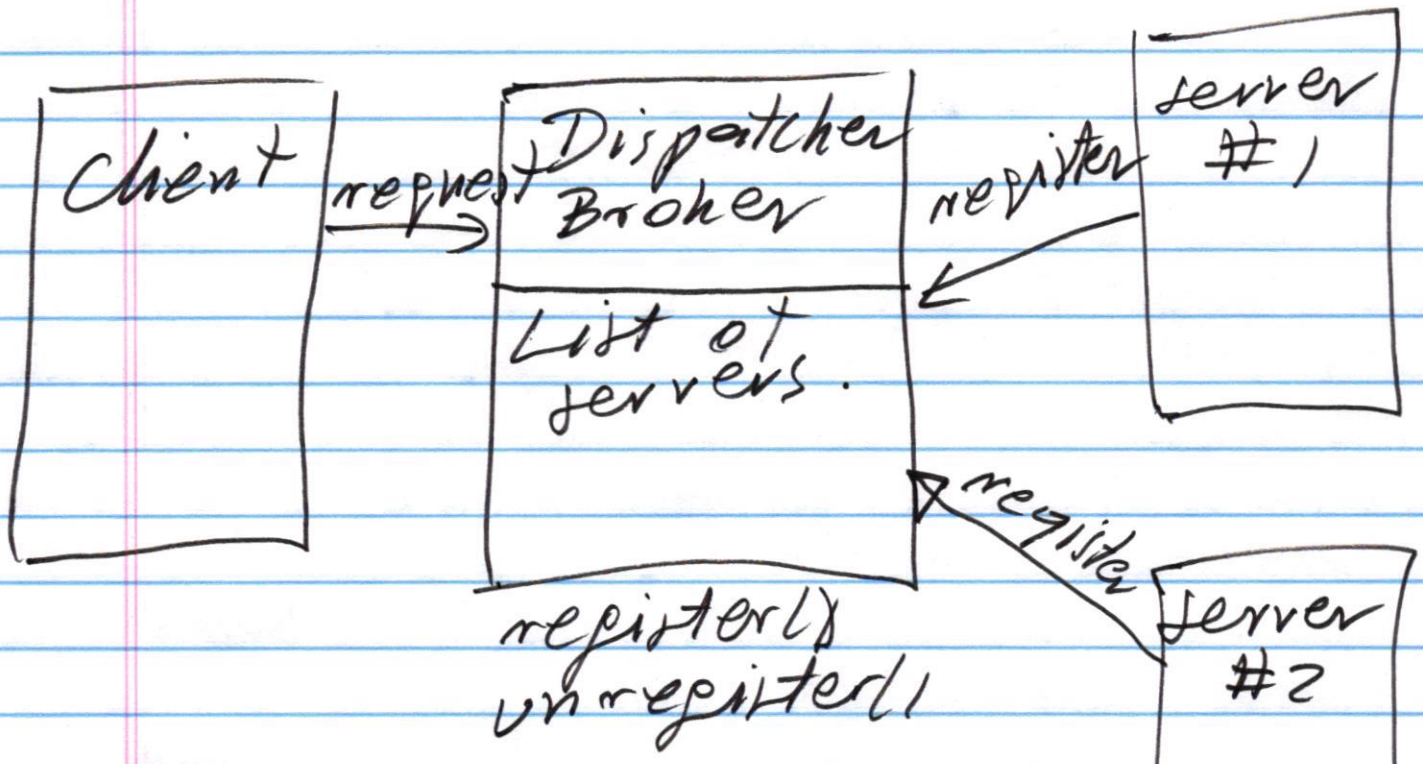
Client

```
Server *S;

S = ?

S-> servie()
```



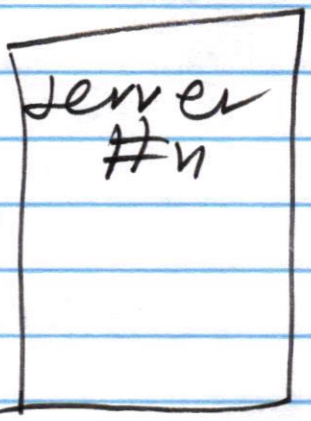


* Dispatcher
* Broker



register()

1. server location
2. service specification
   - * service name
   - * service signature
   - * service specification

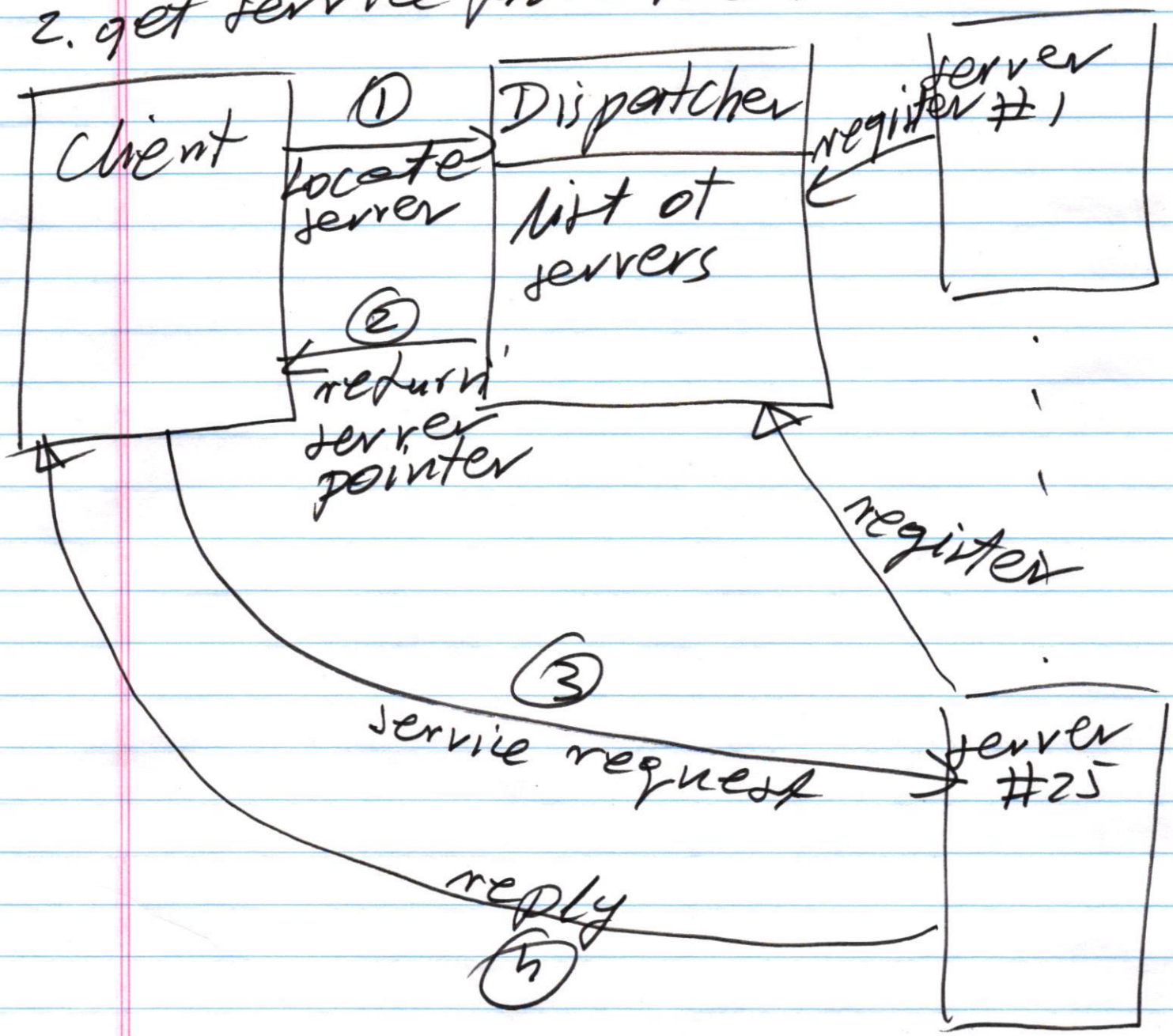


(*) Client-Dispatcher-Server

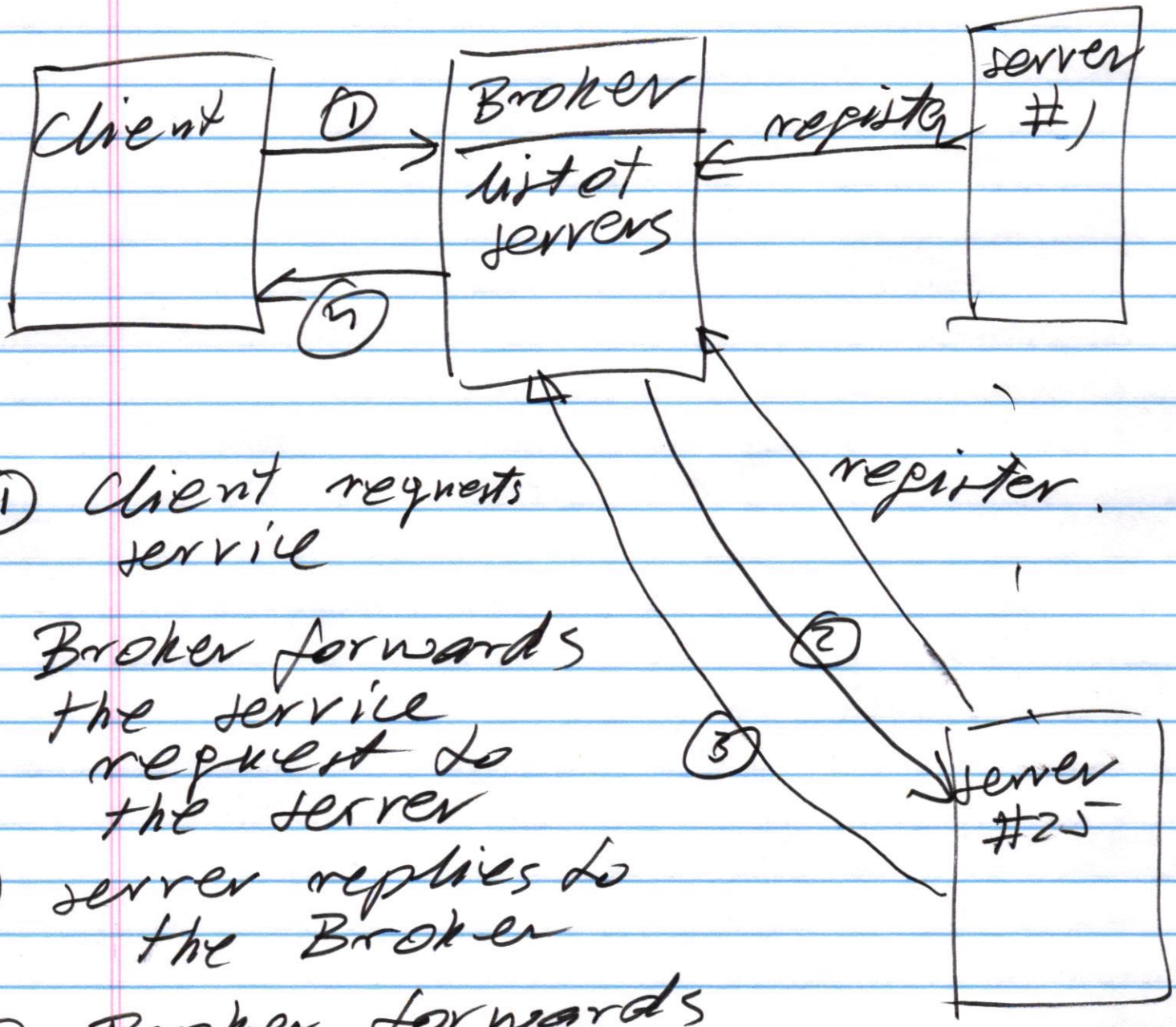
(*) Client-Broker → Server

Client - Dispatcher - Server

1. Locate server
2. get service from the server.



Client – Broker – server



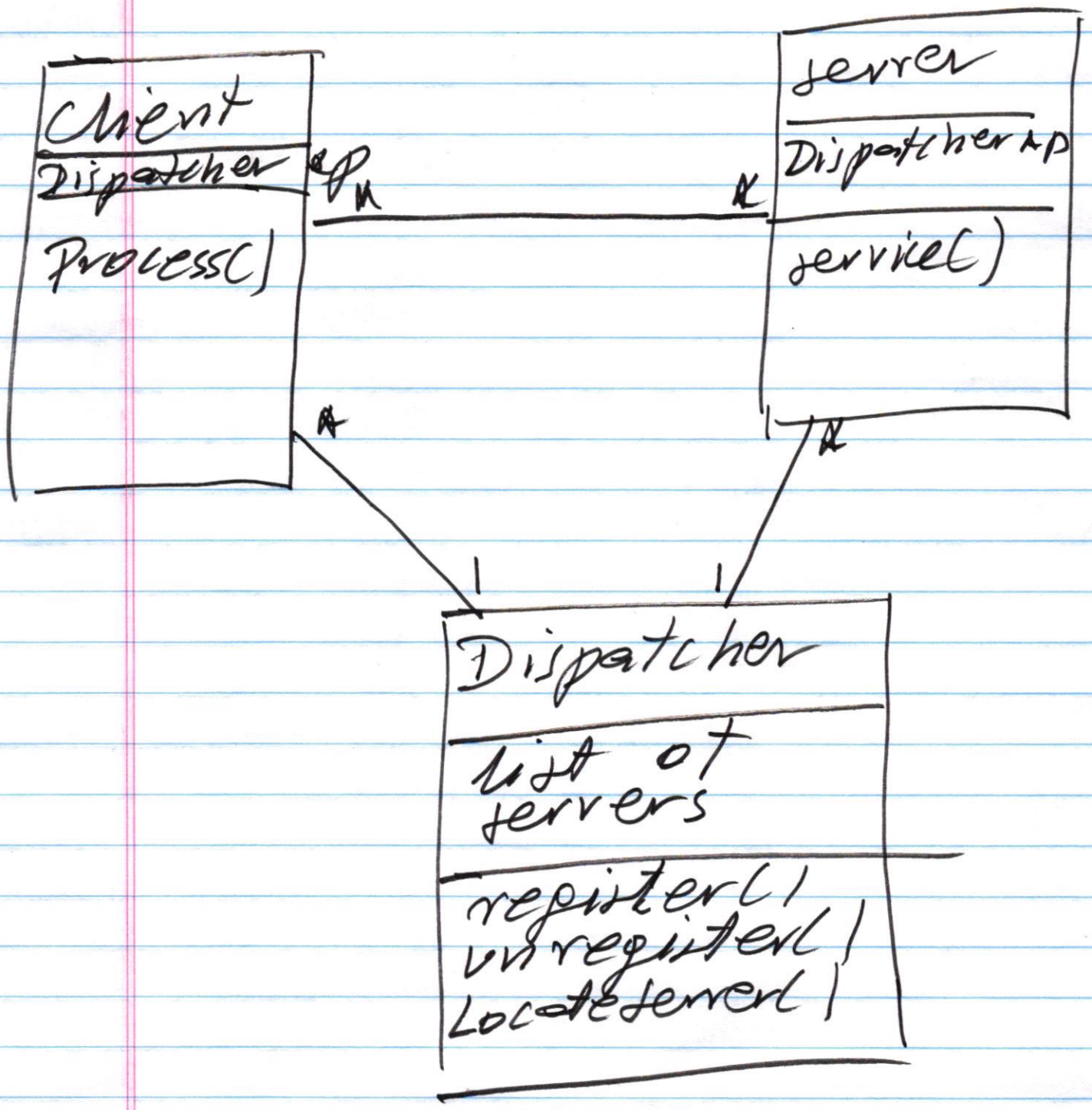
① client requests service

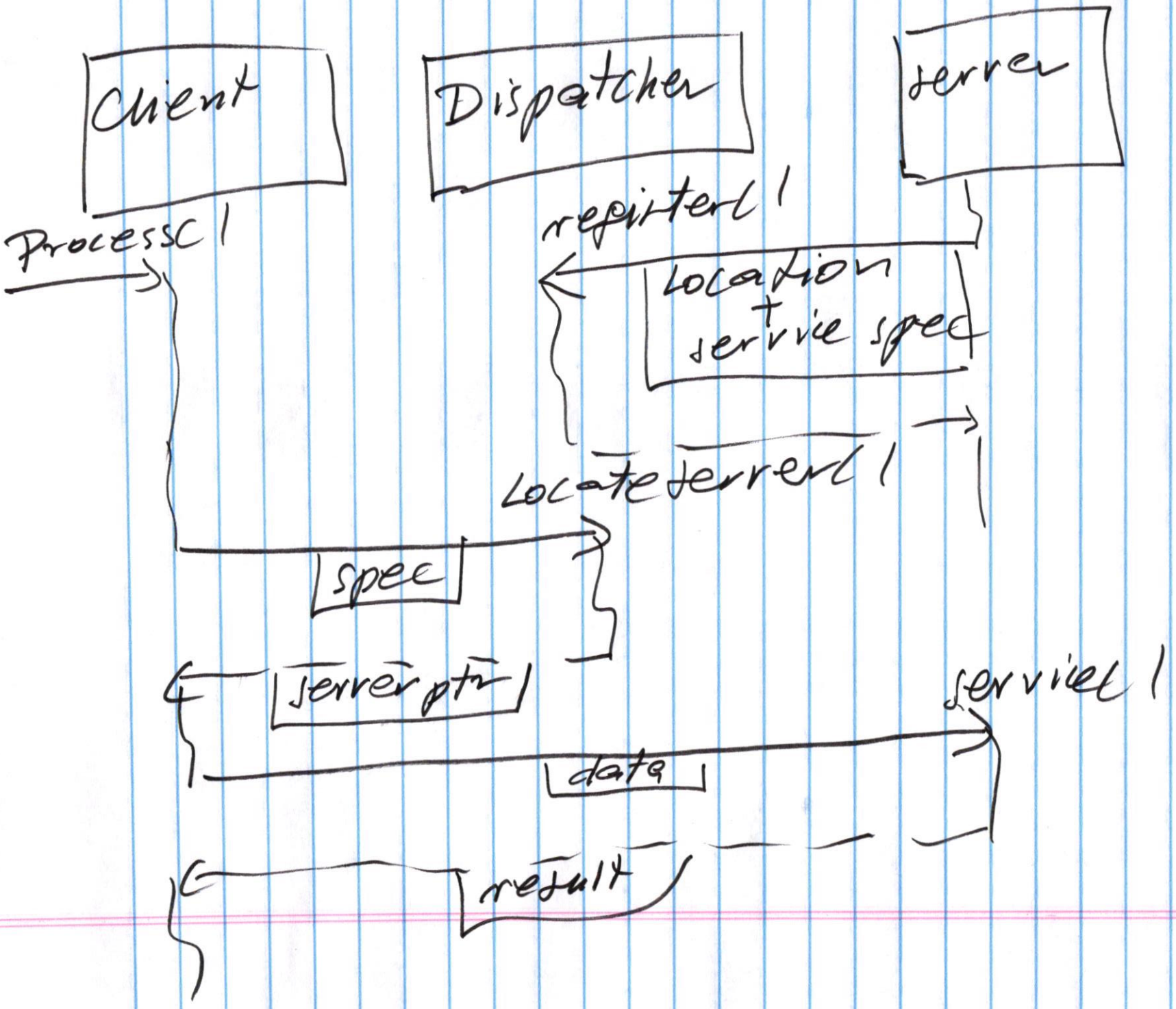
② Broker forwards the service request to the server

③ server replies to the Broker

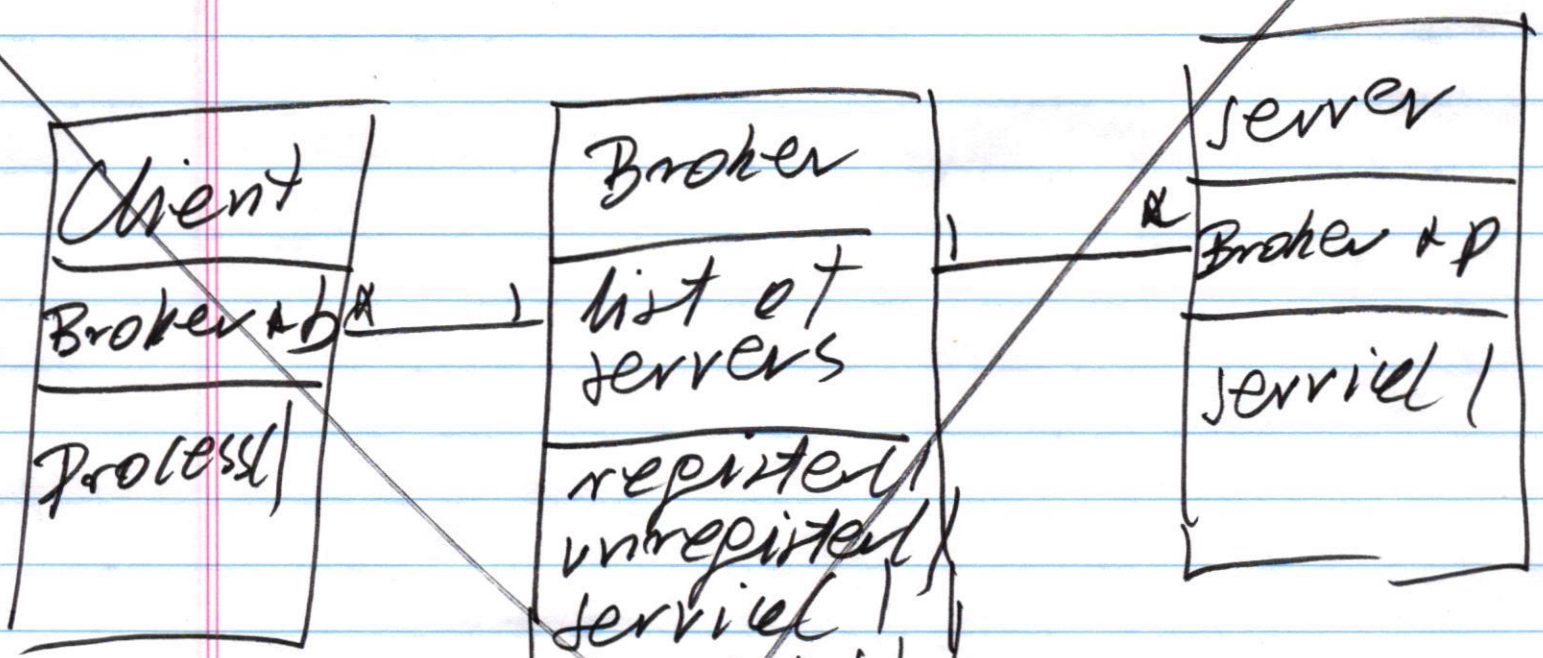
④ Broker forwards the reply to client

Client - Dispatcher - Server




Client
Dispatcher
Server
ProcessC()
registerC()
location
+
service spec
LocateServerC()
spec
ServerPtr
serviceC()
data
result

Client-Broker-Server



Process()

b -> service()

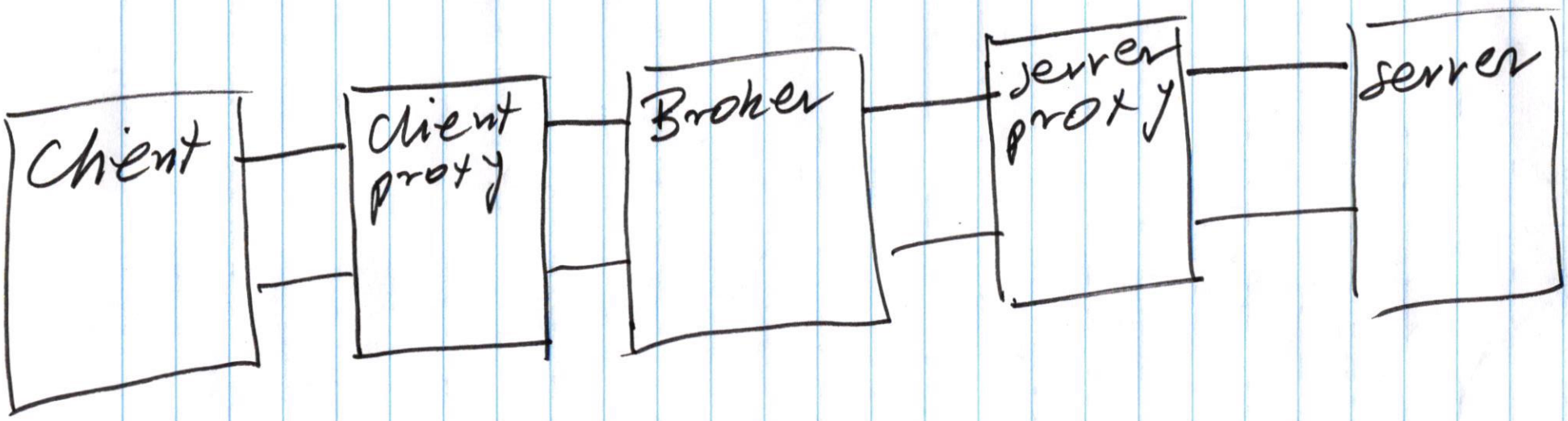
b -> serviceA()

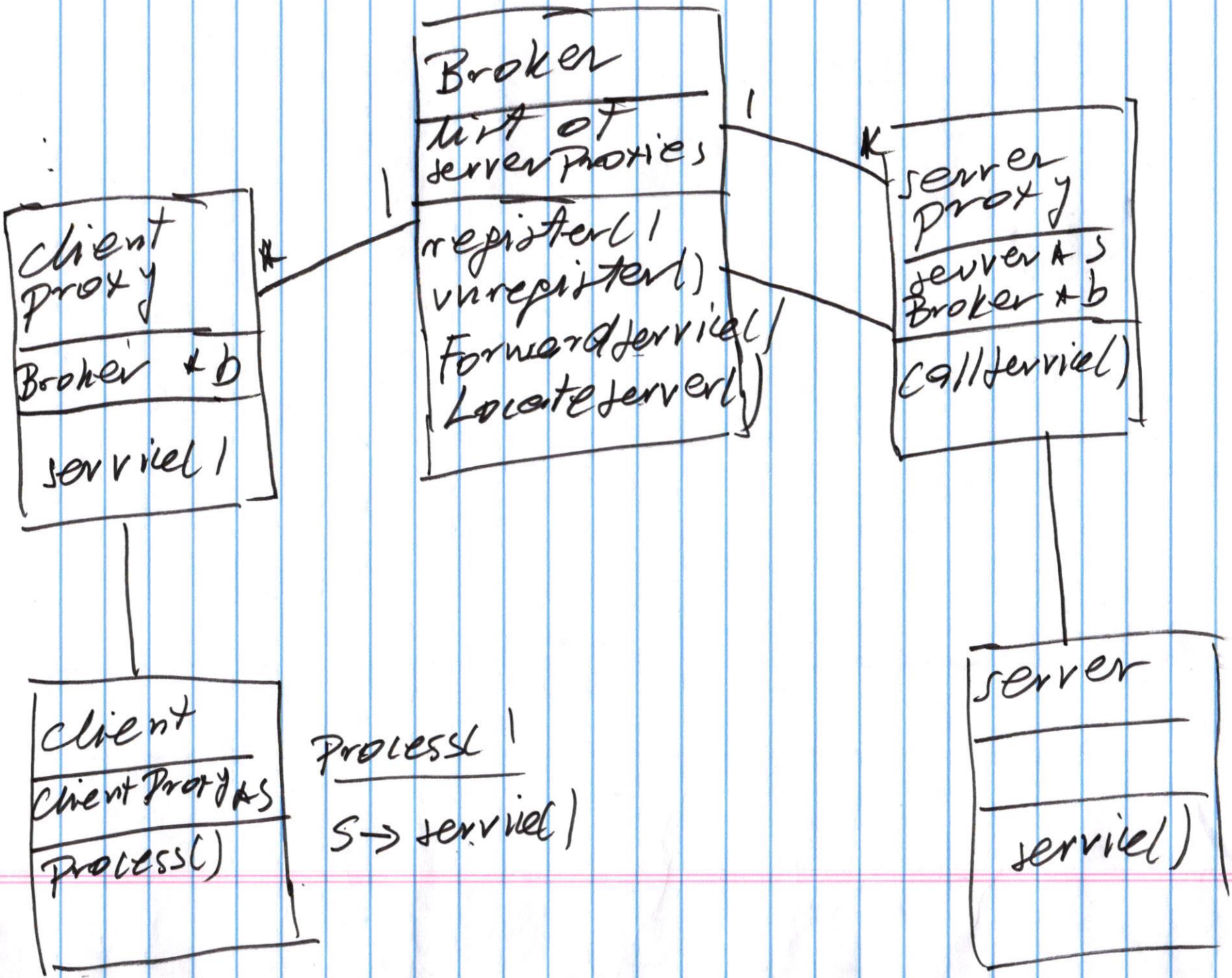
b -> serviceB()

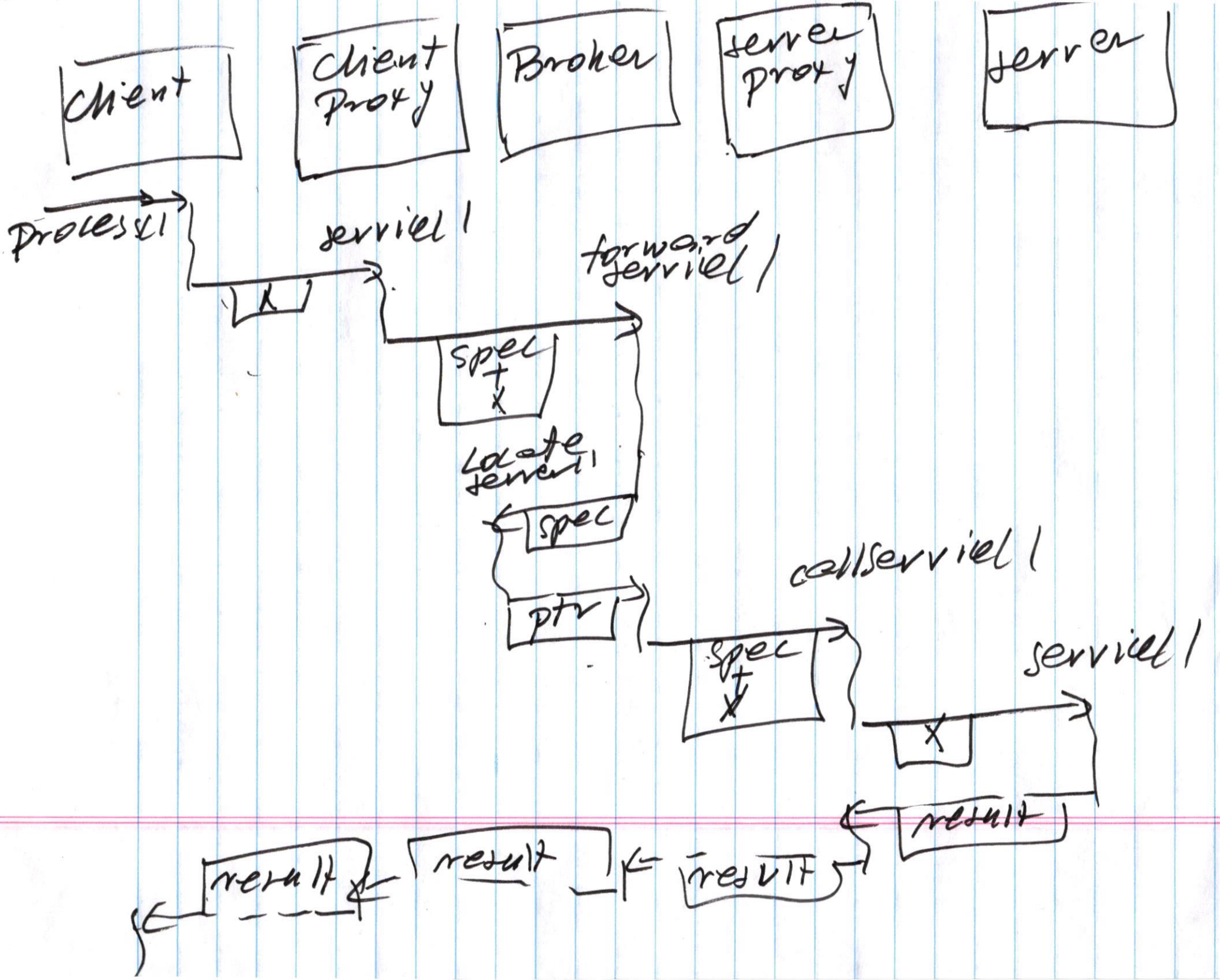
⊂ # of services

Assumptions

* Broker must be de-coupled from services
* Broker and clients should be de-coupled
* Broker and servers should be de-coupled


Client
Client proxy
Broker
server proxy
server


Broker
list of serverProxies
register()
unregister()
ForwardService()
LocateServer()
1
1
clientProxy
Broker *b
service()
*
serverProxy
serverAs
Broker *b
callService()
K
client
clientProxy *s
Process()
Process()
S-> service()
server
service()


Client
Client Proxy
Broker
server Proxy
server
Process()
serviel()
forward serviel()
spec + x
locate server()
spec
ptr
callserviel()
spec + x
serviel()
x
result
result
result
result

Example

Server provides
3 services

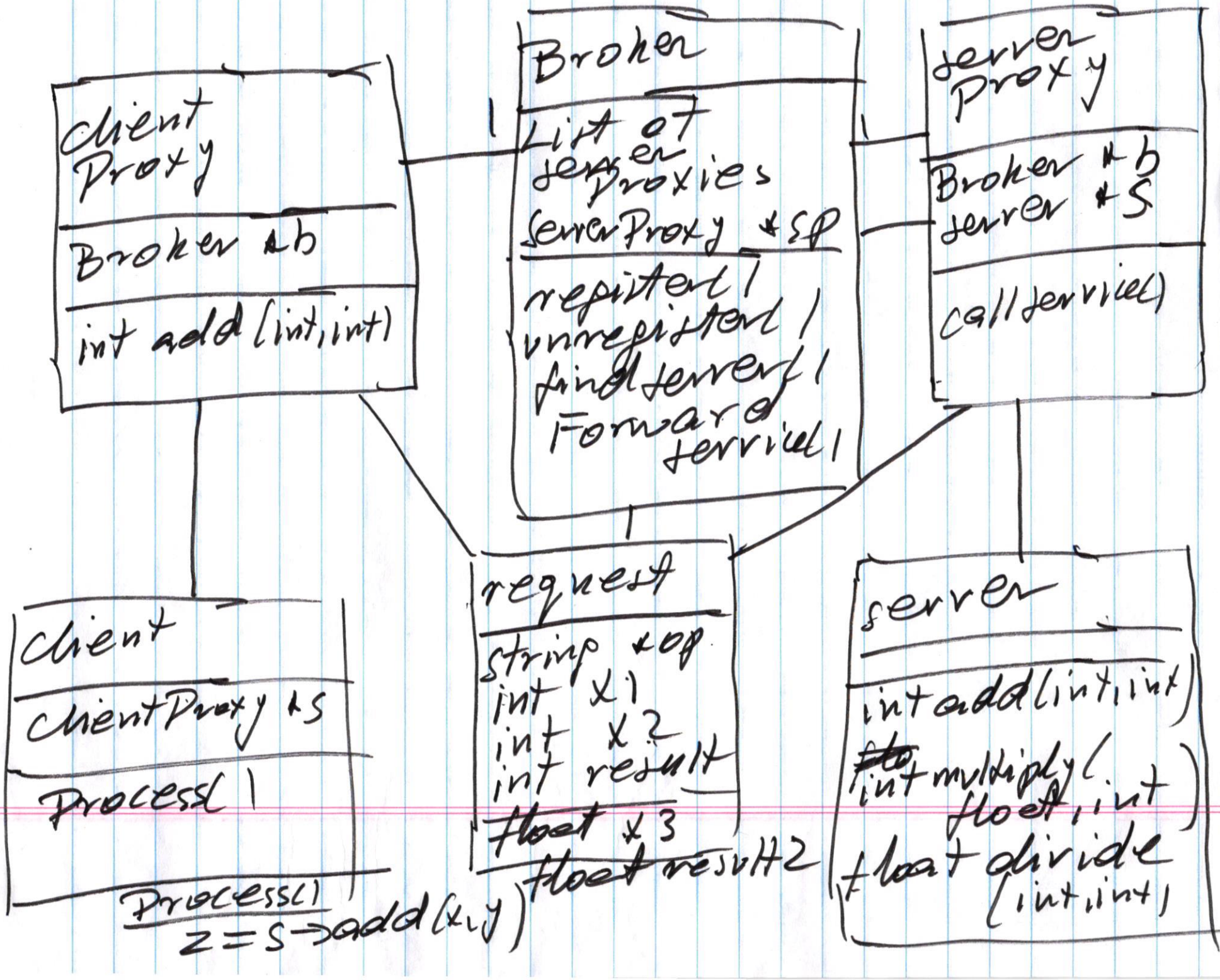
int add (int, int)
int multiply (float, int)
float divide (int, int)

Client wants to get
"add" service

specification of service
"int add(int, int)"

Process()

z = S→add(x, y)


client
Proxy
Broker *b
int add(int,int)
Broker
List of server Proxies
ServerProxy *SP
registerI()
unregisterI()
findServer()
ForwardServiceI()
server Proxy
Broker *b
server *S
callServiceI()
client
clientProxy *S
ProcessC()
ProcessC()
z=S->add(x,y)
request
string *op
int x1
int x2
int result
float x3
float result2
server
int add(int,int)
int multiply(float,int)
float divide(int,int)

Client Proxy

service spec:
"int add(int, int)"

```
int add(int a, int b)
{
    r = new request
    r.op = "int add(int, int)"
    r.x1 = a
    r.x2 = b
    b -> forward(r)
    return r.result
}
```

Broker

forward service (request r)

{

sp = findServer (r.op)

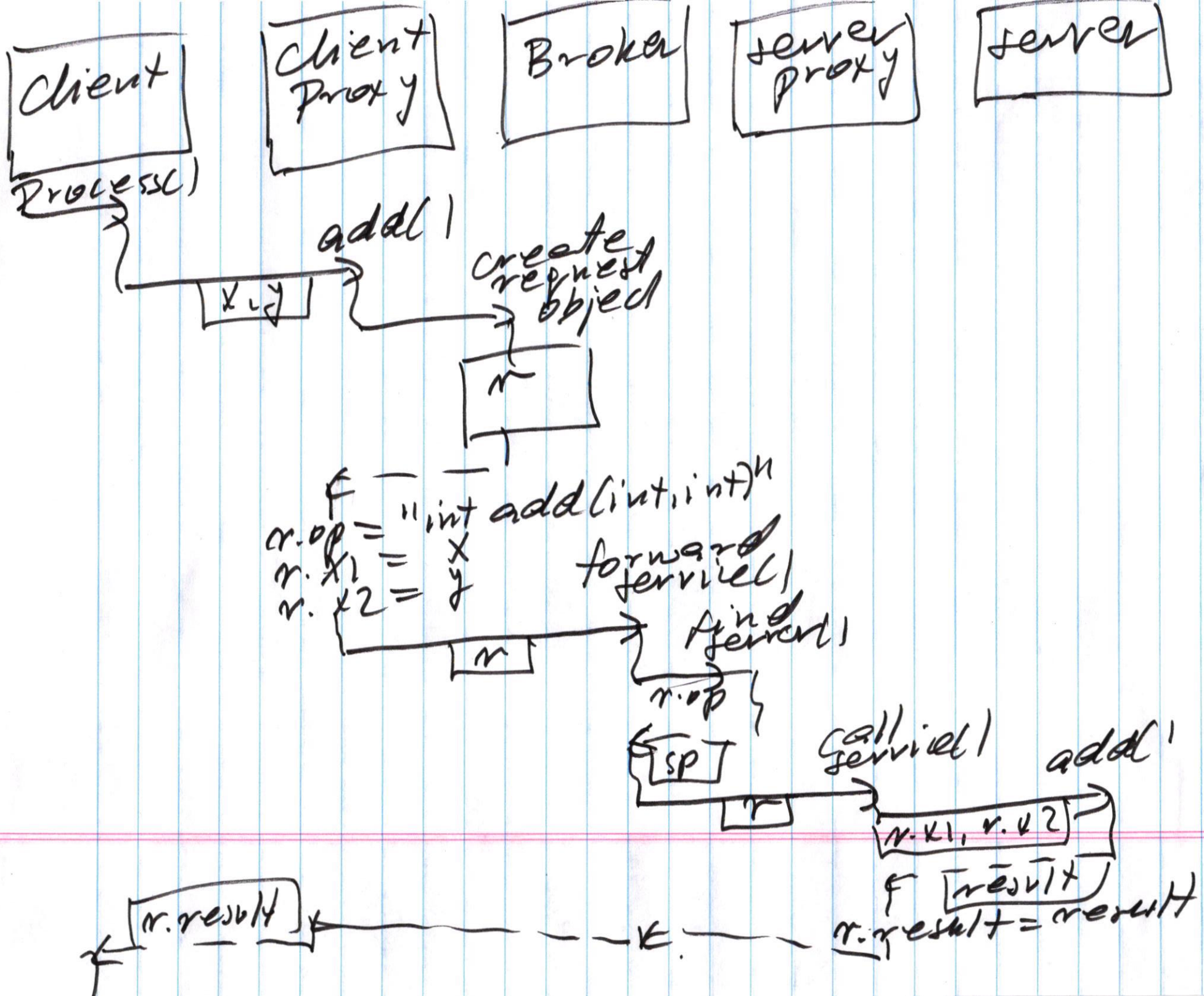
if (sp != nil)

sp → callService (r)

}

ServerProxy

```
callService(request r)
{
if(r.op == "int add(int, int)"
then
    r.result = S -> add(r.x1, r.x2)
elsif
   (r.op = "int multiply(float, int)"
   then
      r.result = S -> multiply(r.x3, r.x1)
   else if
   (r.op == ("float divide(int, int)
   then
   r.result2 = S -> divide(r.x1, r.x2)
}
```


Client
Client Proxy
Broker
Server Proxy
Server
Process()
add()
x, y
create request object
r
r.op = "add"
r.x1 = x
r.x2 = y
int add(int, int)
forward()
r
find()
r.op
SP
call()
add()
r.x1, r.x2
result
r.result = result
r.result