Exam #1

Wednesday, October 1

at 5:00 pm

closed books and notes exam

The coverage is posted

A sample exam with solutions is posted.

Homework #2 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]

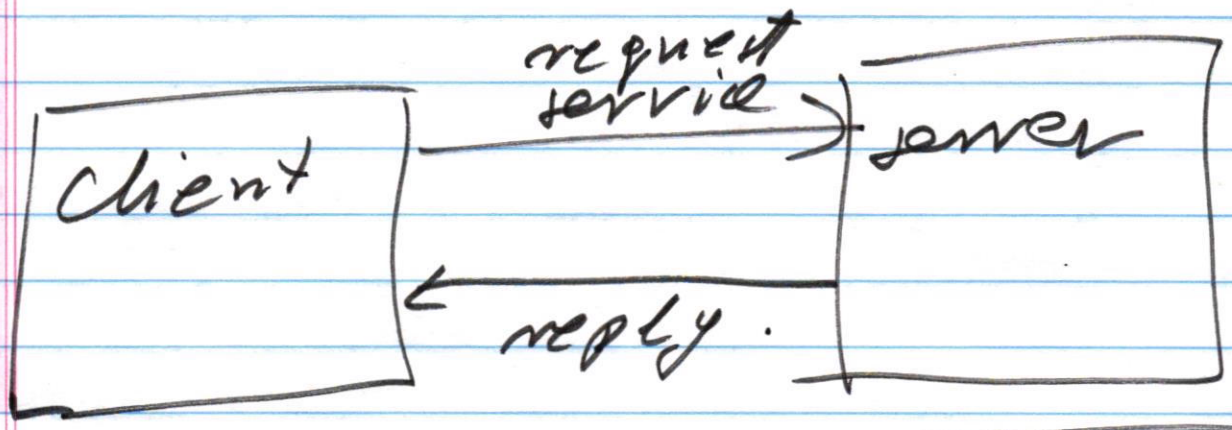
Sources:

- 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--



Client

server *S

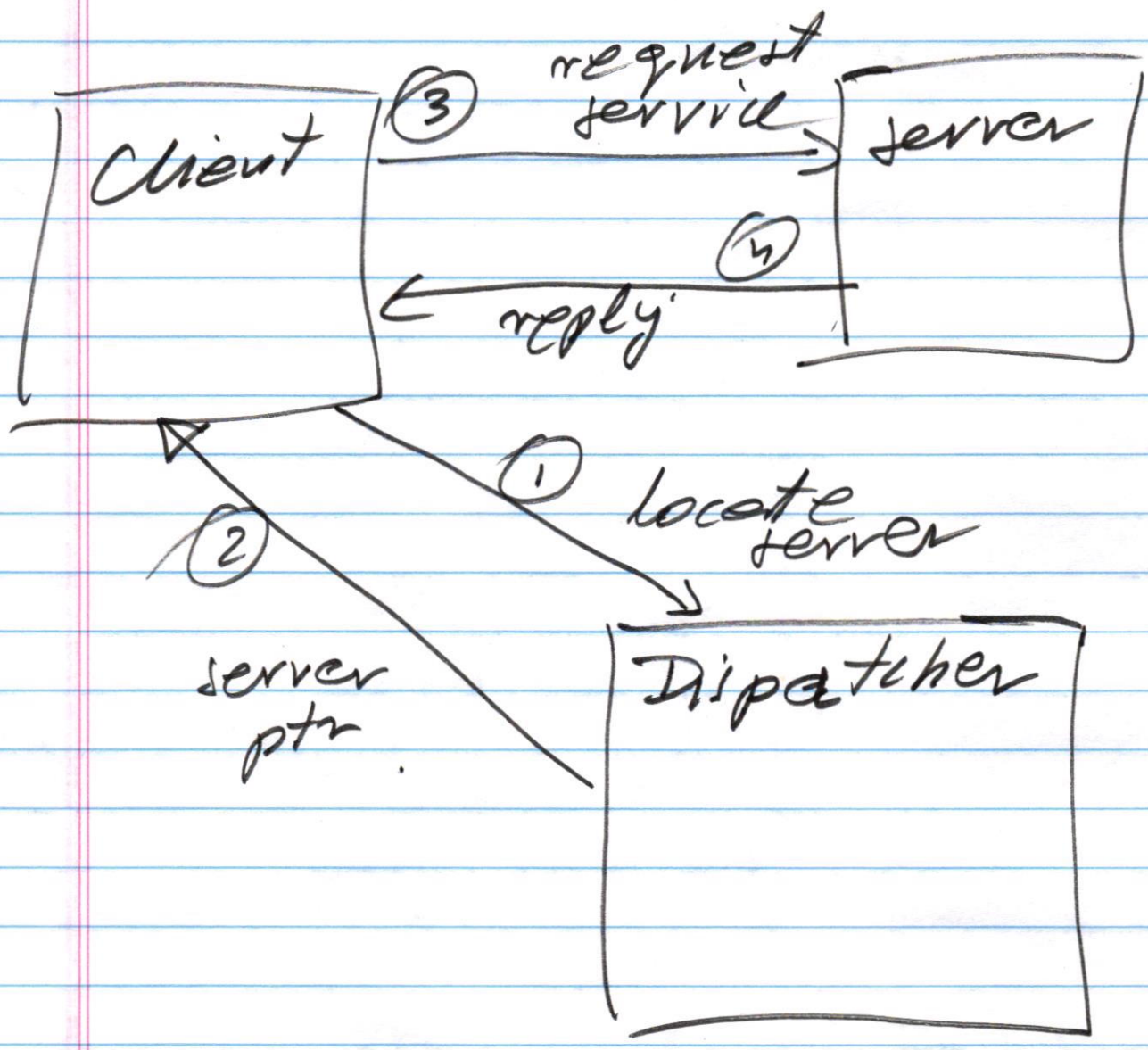
S= ??

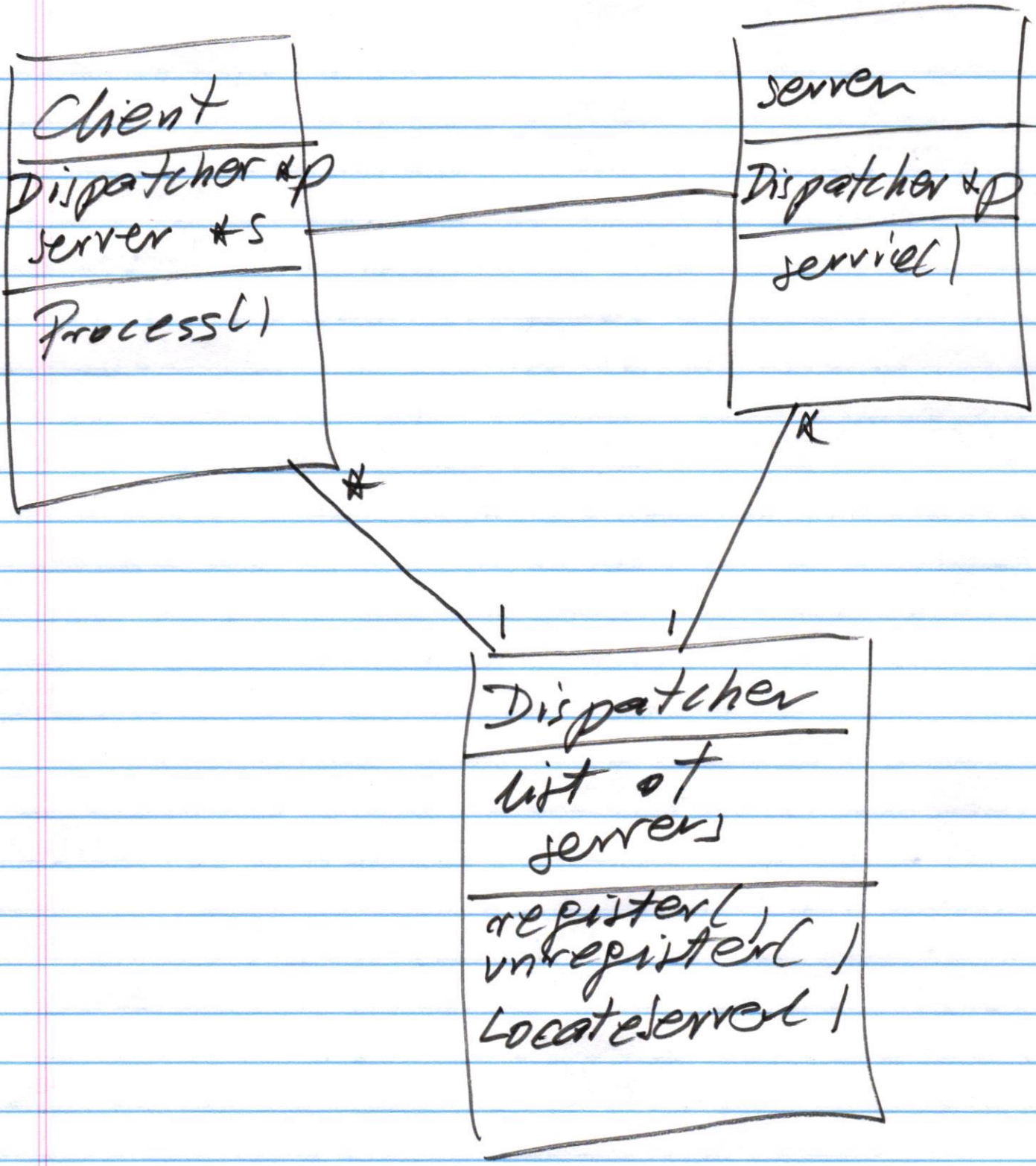
S→ service()

where is the server that can provide the service?

1. Client Dispatcher Server
2. Client Broker Server

Client-Dispatcher-Server




Client
Dispatcher *p
server *s
Process()
server
Dispatcher *p
service()
*
*
1
1
Dispatcher
list of servers
register()
unregister()
Locateserver()

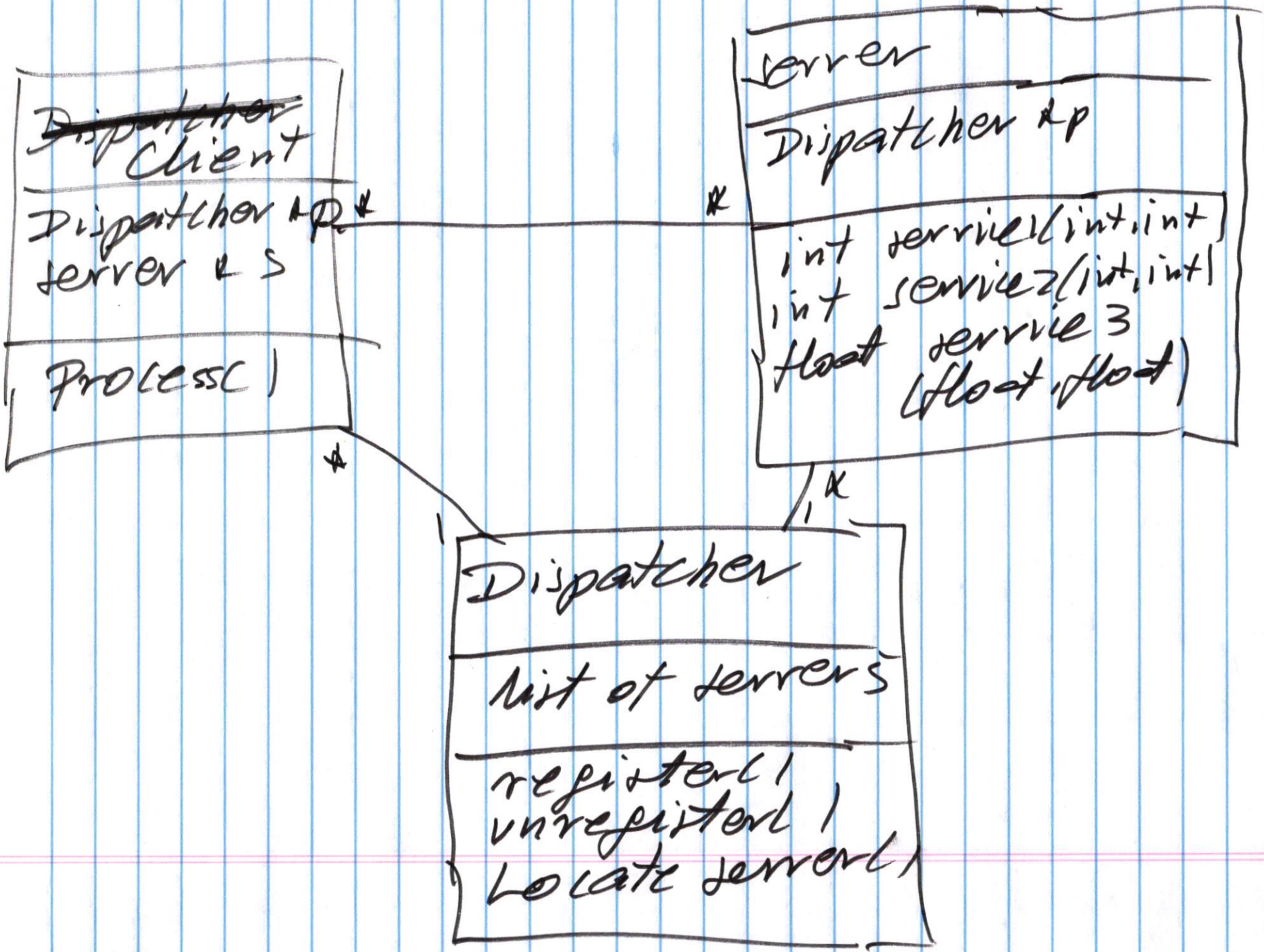
Server:

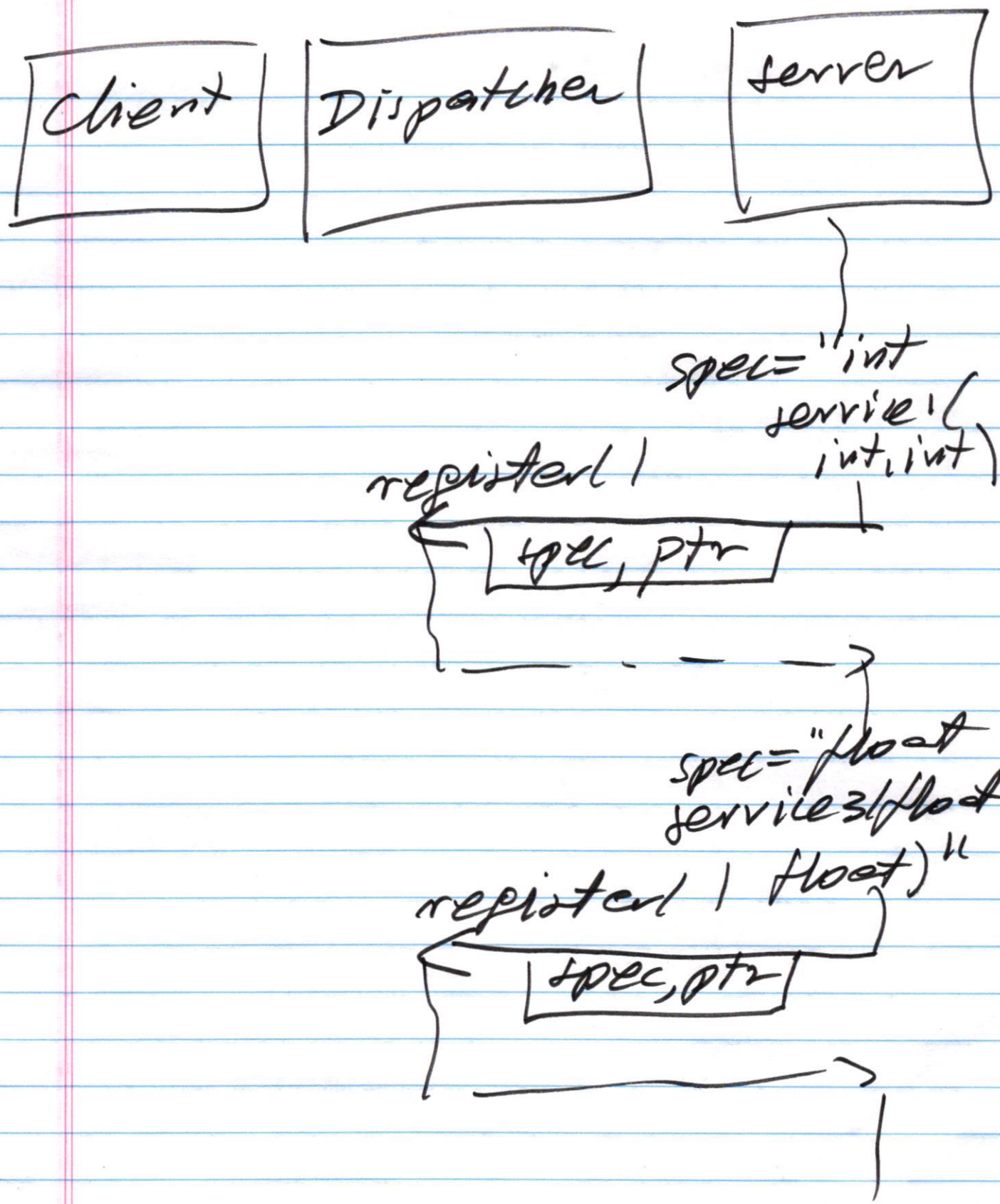
int service1(int, int)
int service2(int, int)
float service3(float, float)

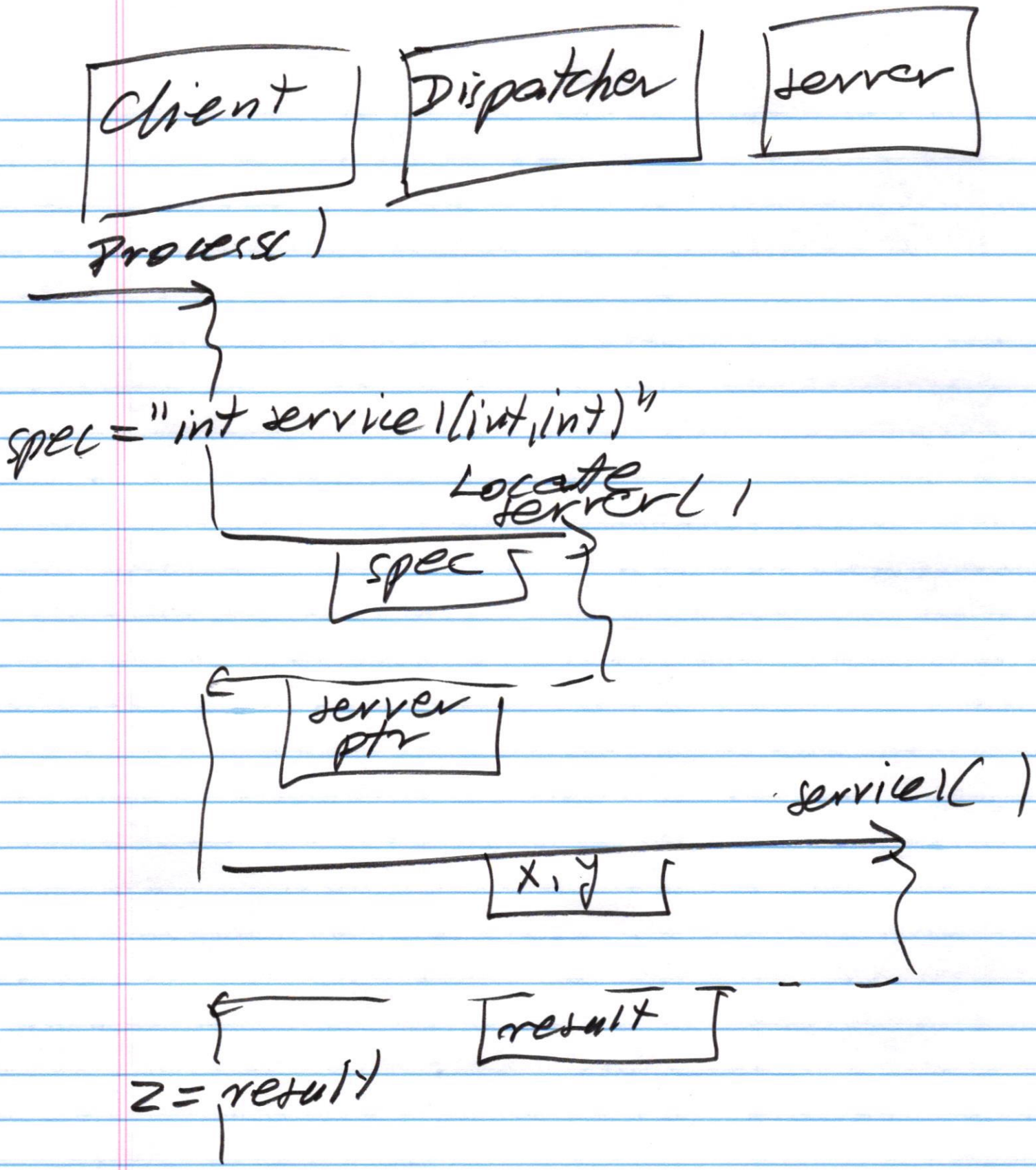
Client service1(), service3()

Process()

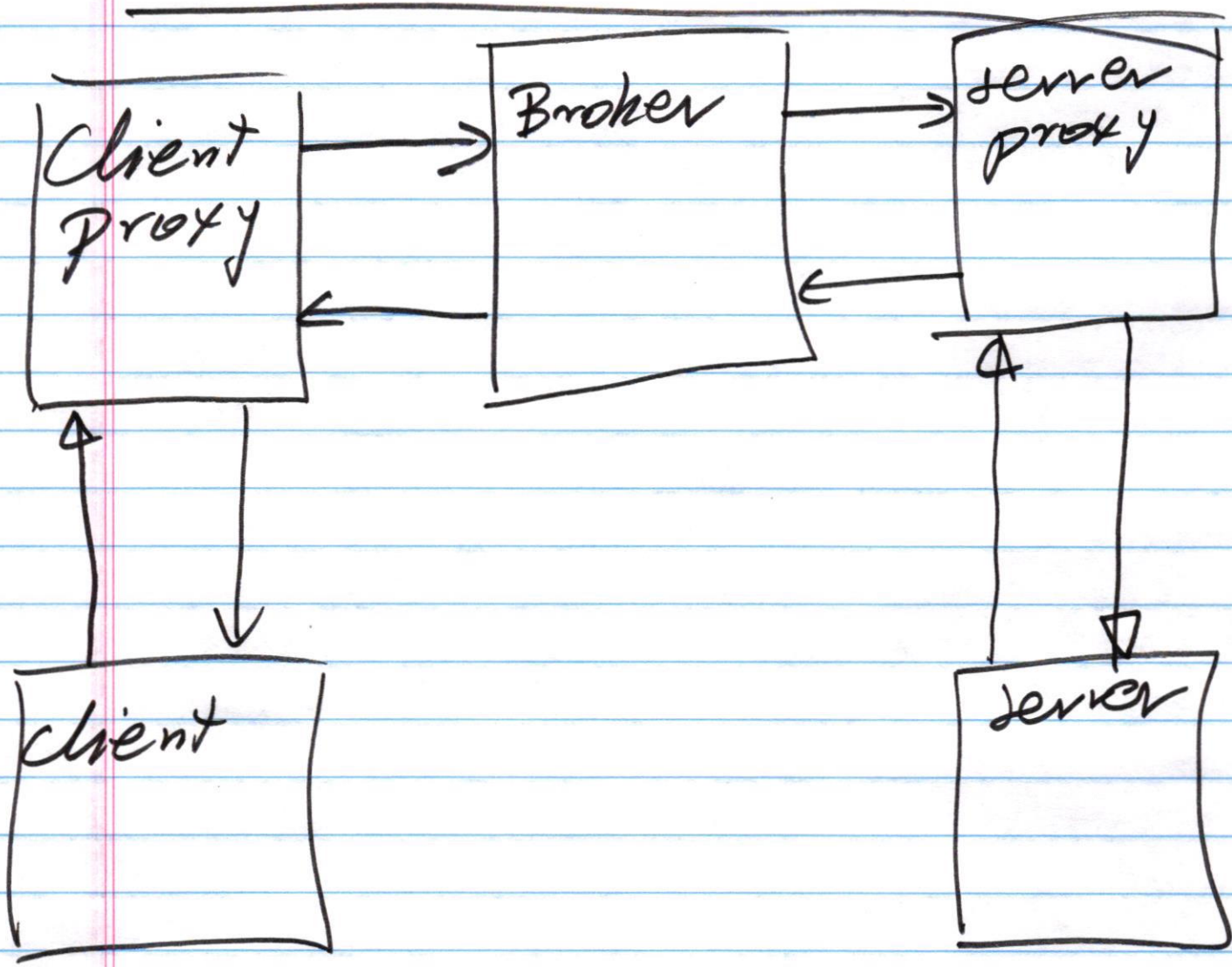
int z,x,y
float a,b,z1
1. Locate server for service1()
   spec = "int service1(int,int)"
   s = p -> Locate server(spec)
   if(s != nil) then
      z = s -> service1(x,y)
1. Locate server for service3()
   spec = "float service3(
                float, float)"
   s = p -> Locate server(spec)
   if(s != nil) then
   z1 = s -> service3(a,b)


~~Dispatcher~~ Client
Dispatcher *p
server *s
Process()
server
Dispatcher *p
int service1(int,int
int service2(int,int
float service3
(float,float)
Dispatcher
list of servers
register()
unregister()
Locate server()
*
*
*
1
*
1


client
Dispatcher
server
spec="int
service1(int,int)"
register1
spec, ptr
spec="float
service3(float,
float)"
register1
spec, ptr

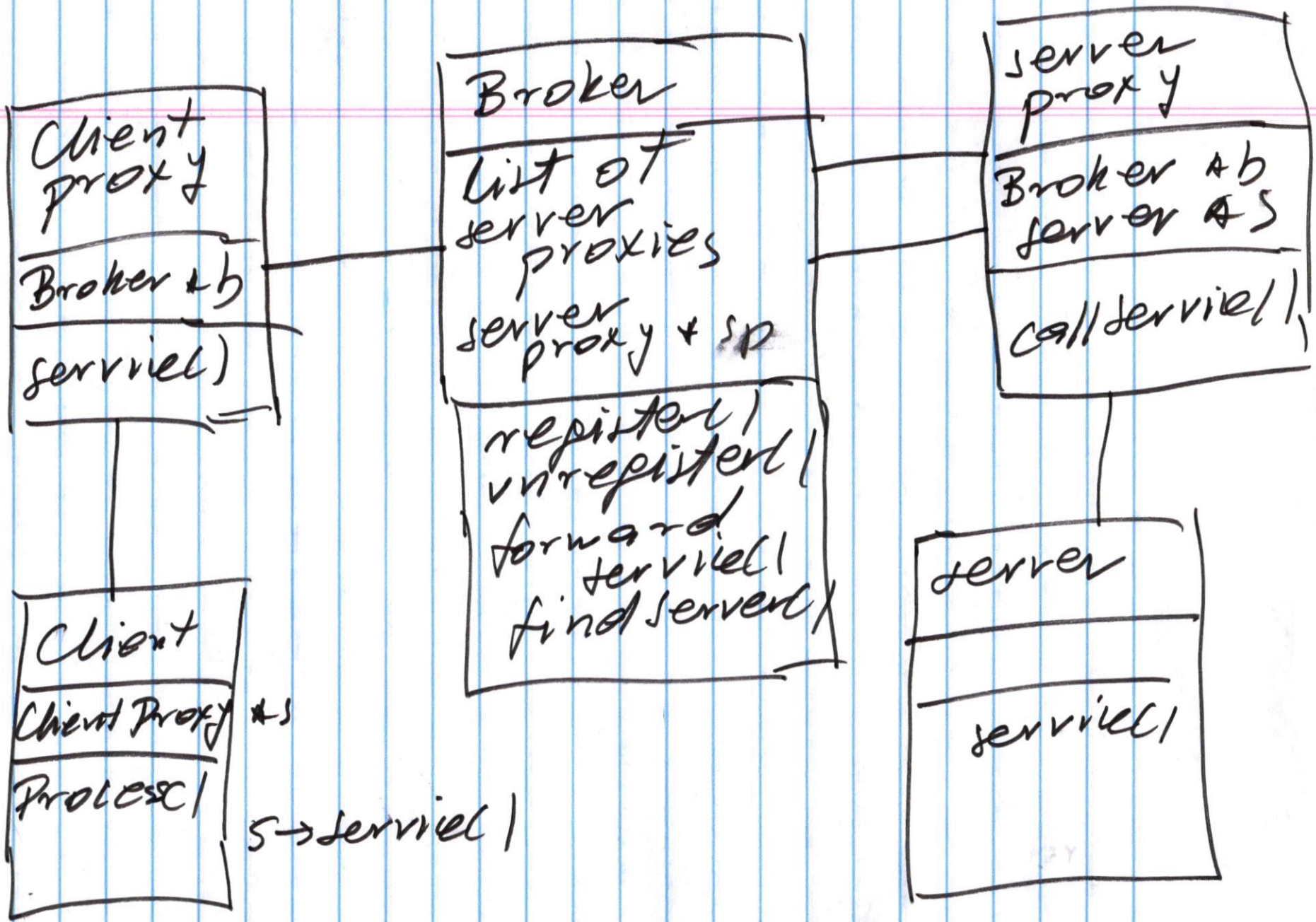

Client
Dispatcher
server
Process()
spec = "int service1(int,int)"
Locate server()
spec
server ptr
service1()
x, y
result
z = result

Client - Broker - Server



Broker MUST be decoupled from

- services
- clients
- servers


Client proxy
Broker *b
service()
Client
Client Proxy *s
Process()
s->service()
Broker
list of server proxies
server proxy * sp
register()
unregister()
forward service()
findServer()
server proxy
Broker *b
server *s
call service()
server
service()

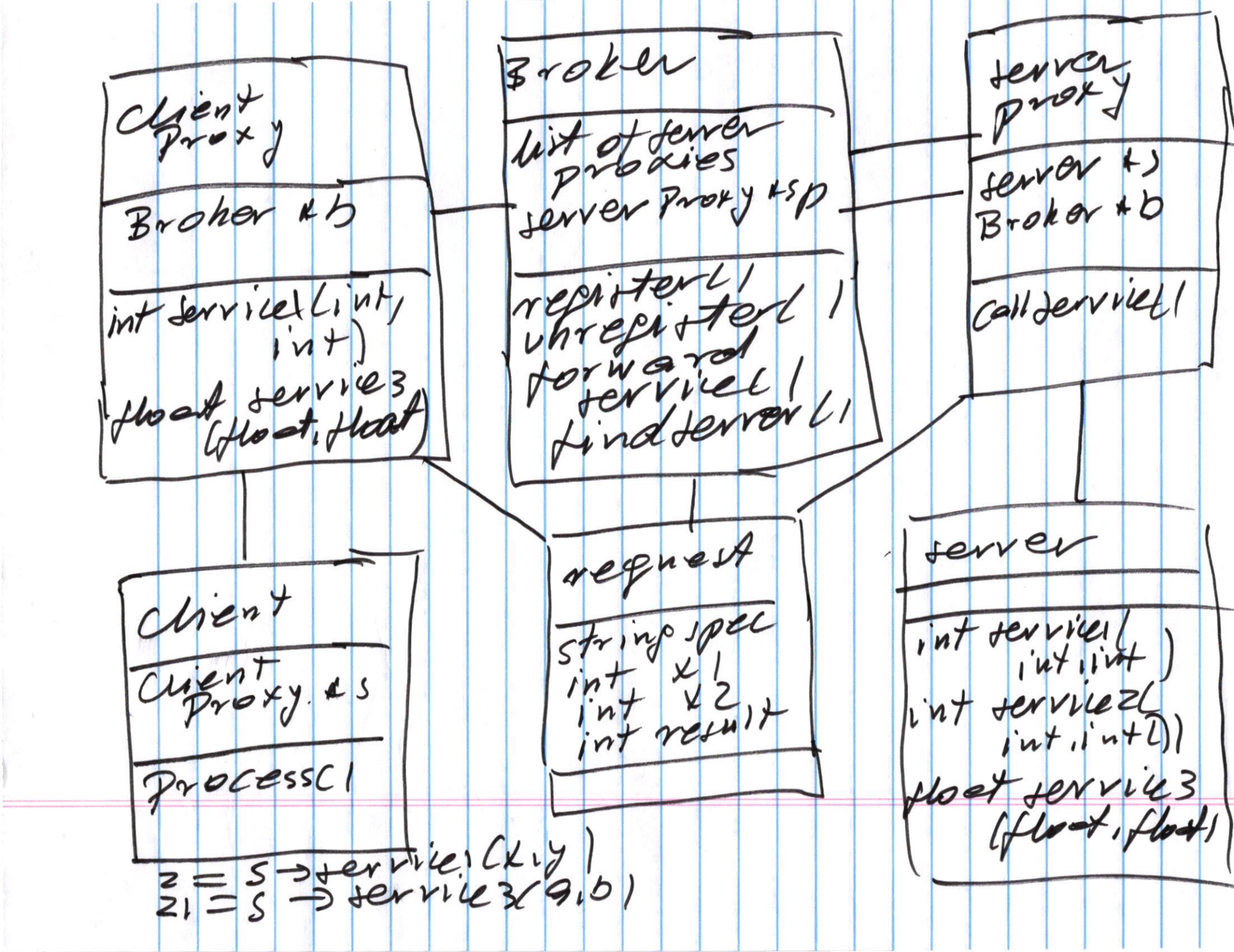
server

```
int service1(int, int)
int service2(int, int)
float service3(float, float)
```

Client

Process

```
Client Proxy ks
int z, x, y;
float a, b, z1
z = s->service1(x, y)
z1 = s->service3(a, b)
```


Client Proxy
Broker *b
int service1(int, int)
float service3 (float, float)
Broker
list of server Proxies
serverProxy *sp
register()
unregister()
forward service()
findServer()
server Proxy
server *s
Broker *b
callService()
Client
Client Proxy *s
ProcessC()
request
string spec
int x1
int x2
int result
server
int service1(int, int)
int service2(int, int)
float service3 (float, float)
z = s → service1(x, y)
z1 = s → service3(a, b)

Client Proxy.

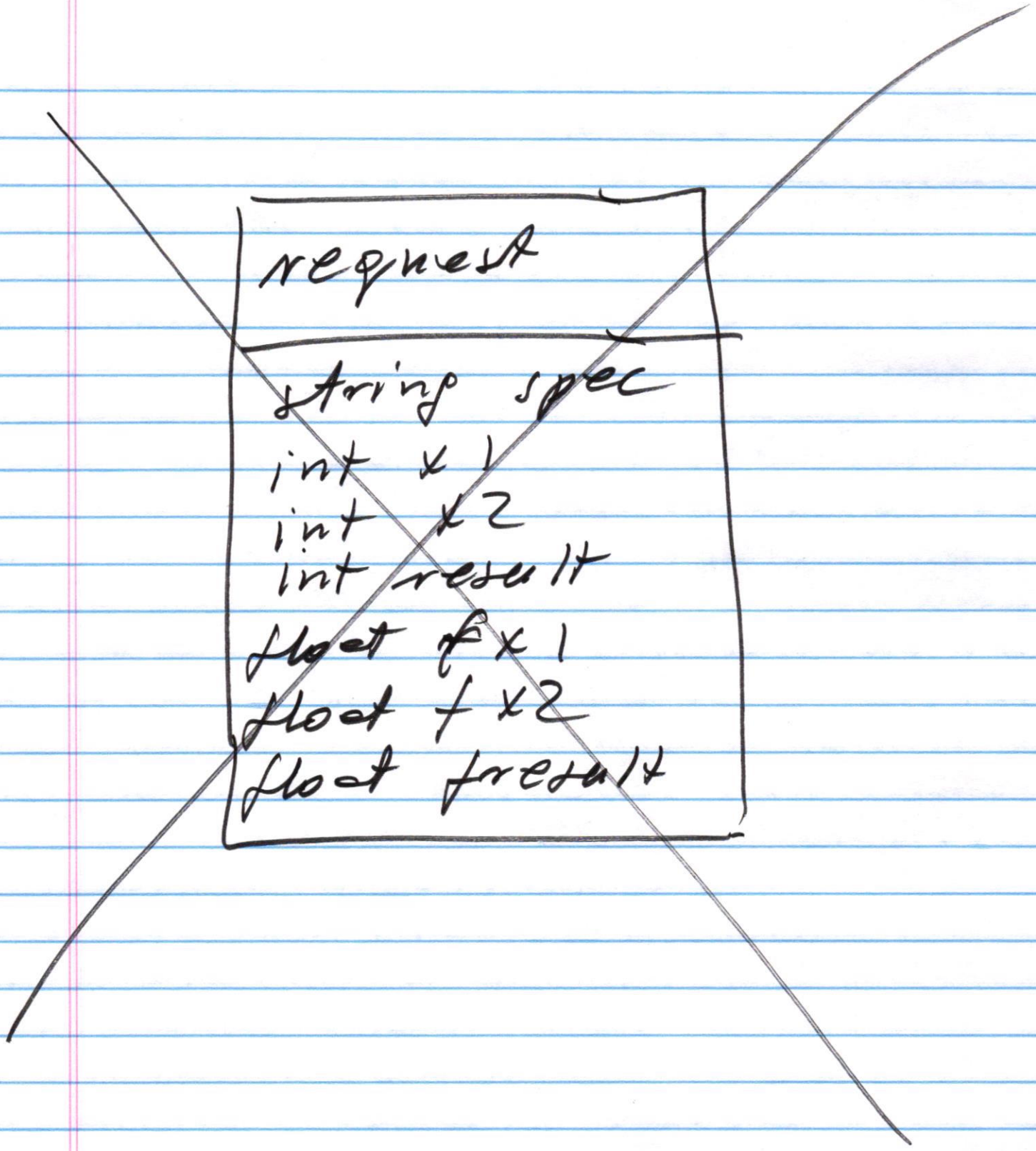
```
int service1(int a1, int a2)
{
    r = new request
    r->spec = "int service1(int, int)"
    r->x1 = a1
    r->x2 = a2
    b -> forward service(r)
    return r->result;
}
```

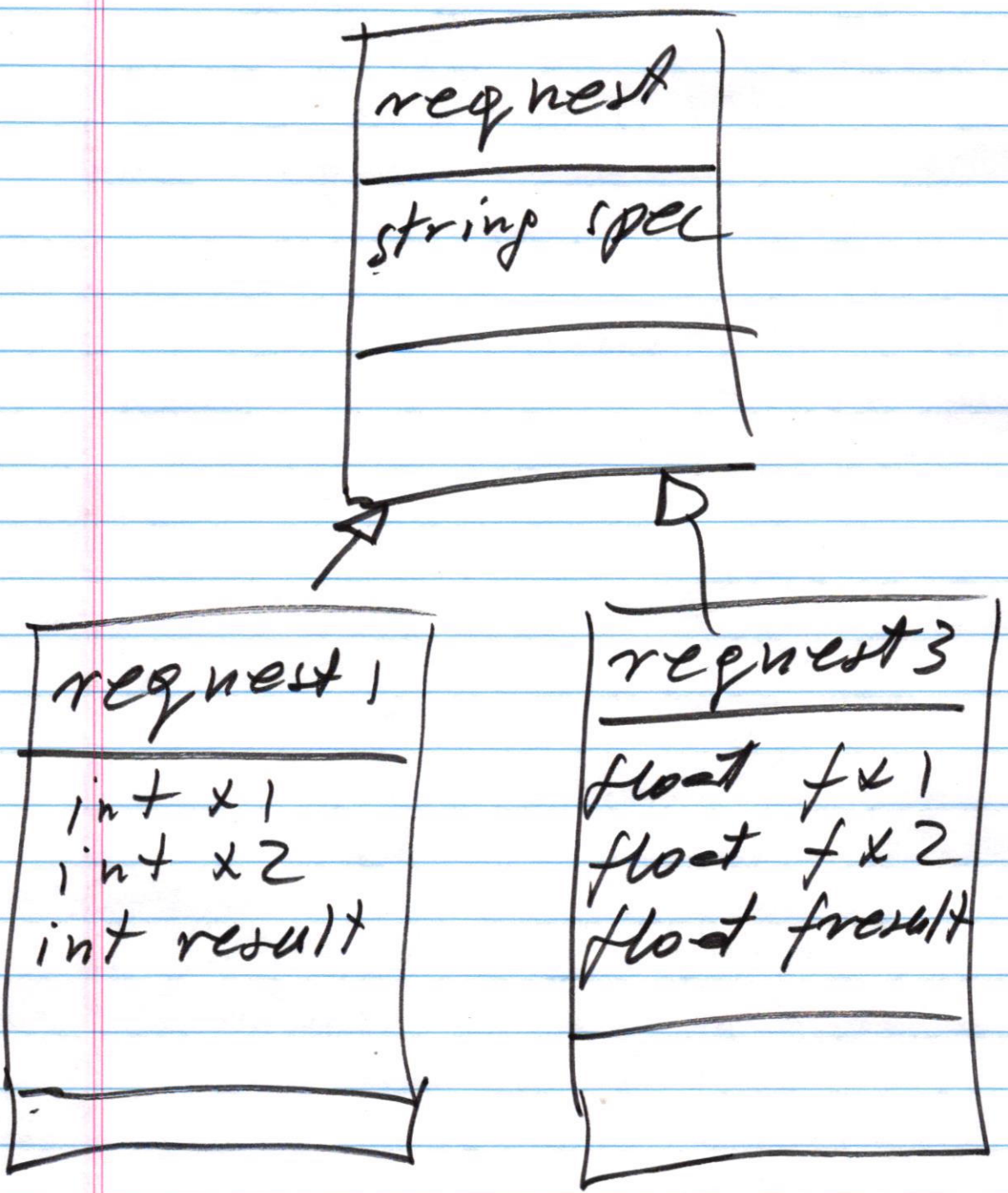
Broker

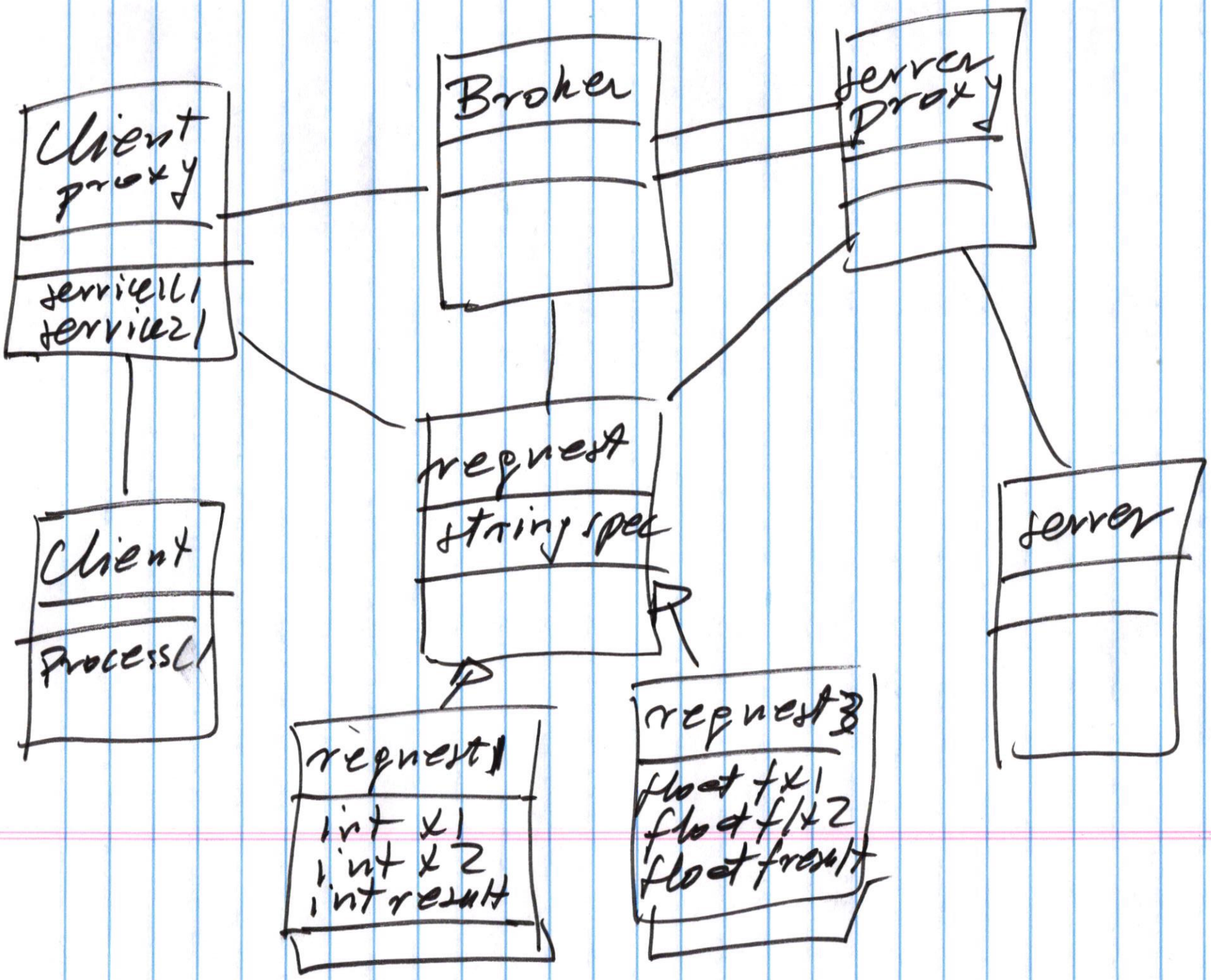
```
forwardservice (requestr)
{ sp = findserver(r->spec)
  if (sp != nil)
      sp -> callservice(r)
  else ??
      server is not
               found
}.
```

ServerProxy

```
callservice(repnest r)
{
if(r->spec == "int service1(int,int)"
    r->result = S-> service1(r->x1, r->x2)

else if
    (r->spec == "int service2(int,int)"
    r->result = S-> service2(r->x1, r->x2)

else if
    (r->spec = "float service3(float,
                               float)"
r->fresult = S-> service3(r->fx1, r->fx2)
}
```


request
string spec
int x1
int x2
int result
float fx1
float fx2
float fresult

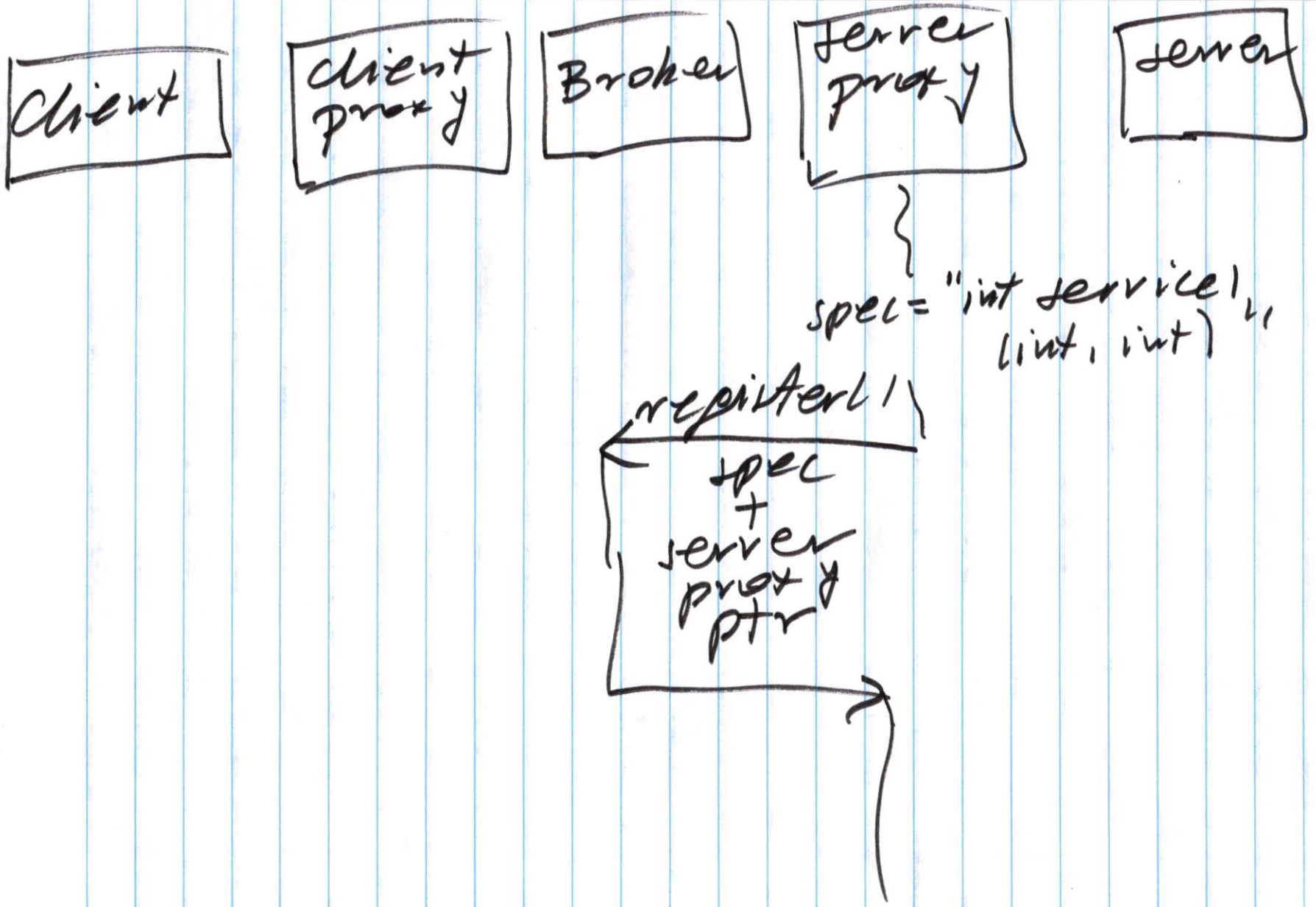

request
string spec
request1
int x1
int x2
int result
request3
float fx1
float fx2
float fresult

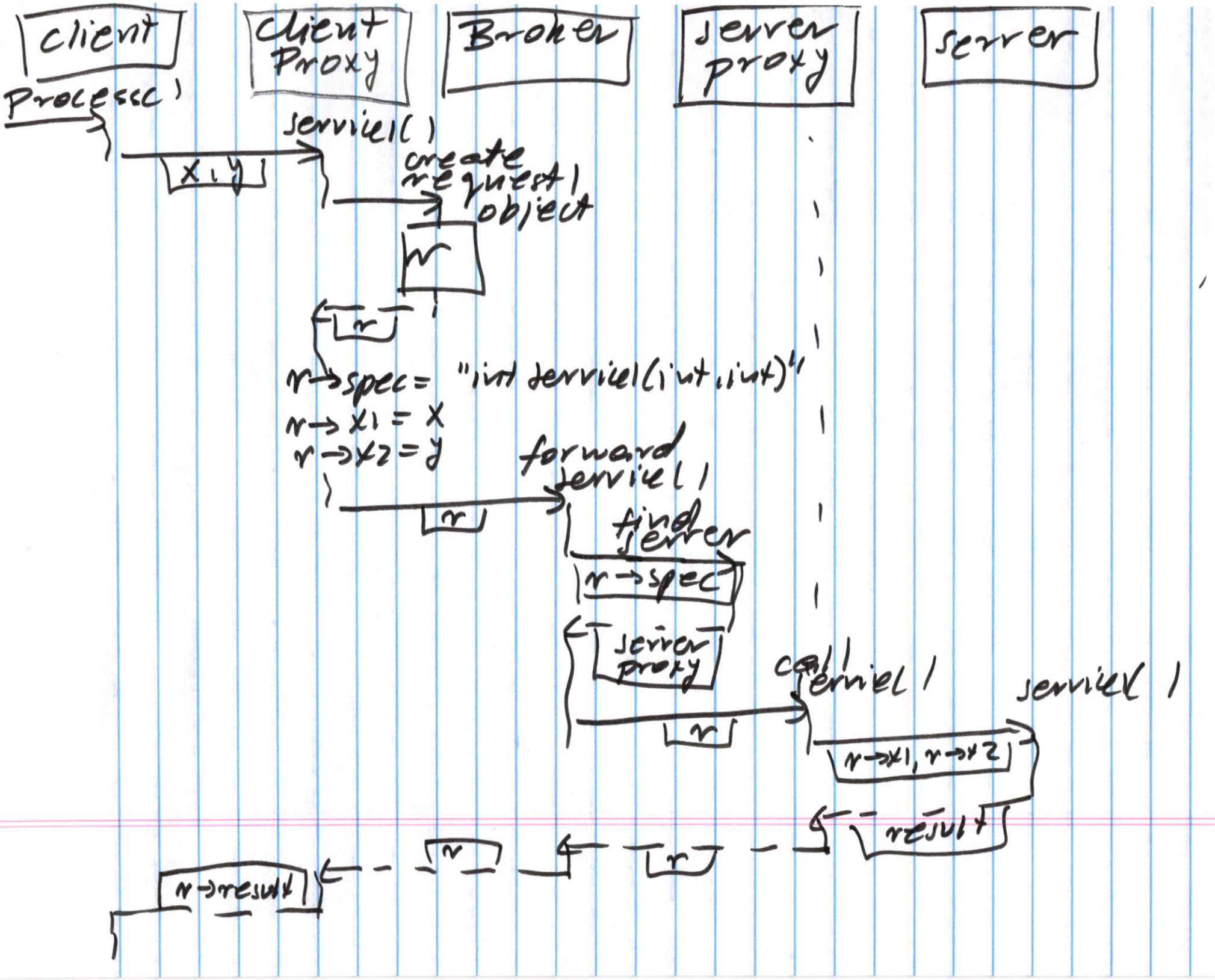

Client proxy
service1()
service2()
Broker
server proxy
request
stringspec
Client
Process()
server
request1
int x1
int x2
int result
request2
float fx1
float fx2
float fresult

Client Proxy

```
int service1(int a1, int a2)
{ request *r.

    r = new request()

    r->x1 = a1
    r->x2 = a2
    r->spec = "int service1(
                        int, int)"

    b -> forward service(r)
    return r->result

}
```

```
float service3(float a1, float a2)
{
    request r
    r = new request 3
    r->fx1 = a1
    r->fx2 = a2
    r->spec = "float service3
                (float, float)"
    b -> forward service(r)
    return r->fresult
}
```


client
client proxy
Broker
server proxy
server
spec = "int service1, (int, int)"
register()
spec + server proxy ptr


client
client Proxy
Broker
server proxy
server
Process()
service()
x, y
create request object
r
r
r->spec = "int service(int,int)"
r->x1 = x
r->x2 = y
forward service()
r
find server
r->spec
server proxy
call service()
service()
r
r->x1, r->x2
result
r
r
r
r->result

Advantages

- server transparency
- fault tolerance
- server migration
- Broker is de-coupled from servers, Clients, services
- error handling is done by server proxy.
- limited error handling by clients / client proxy.

Disadvantages

* overhead → lower performance.

* client/server proxies must be implemented

* client

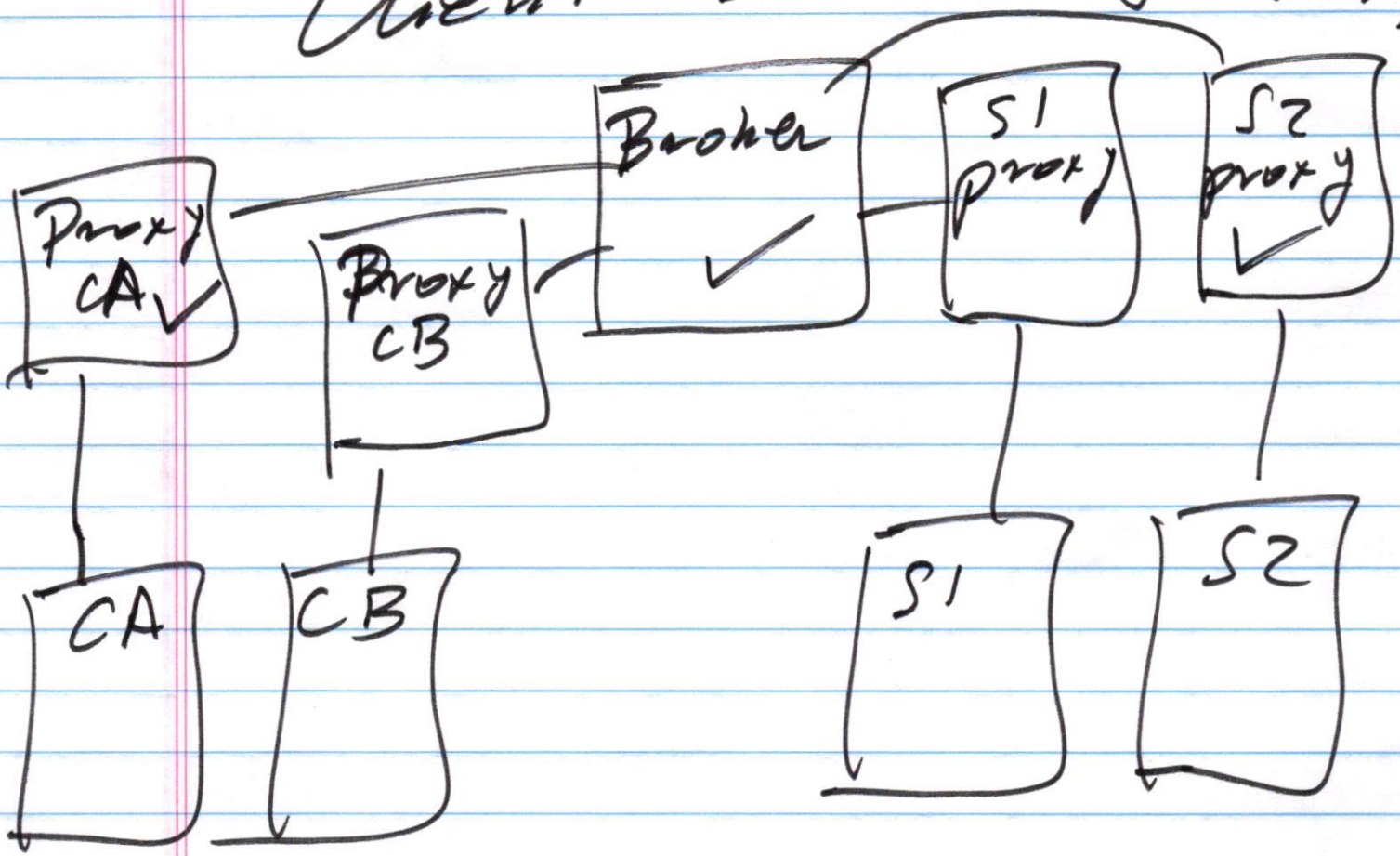
  s → service()

  server is not found?

Homework #2

Problem #1

Client - Broker - Server.



Problem #2

Adapter pattern

Problem #3

Abstract factory pattern.

HOMEWORK ASSIGNMENT #2

CS 586; Fall 2025
Due Date: **October 16, 2025**
Late homework: 50% off
After **October 21,** the homework assignment will not be accepted.

This is an **individual** assignment. **Identical or similar** solutions will be penalized.

Submission: All homework assignments must be submitted on the Blackboard. The submission **must** be as one PDF file (otherwise, a 10% penalty will be applied).

PROBLEM #1 (40 points)
There exist two servers S1 and S2. Both servers support the following services:

Services supported by **server-S1**:
void Service1(string, int, int)
void Service2(string, int, int)
int Service3(string)
float Service4(string)

Services supported by **server-S2**:
void Service1(string, int)
void Service2(string, int)
int Service3(string)
float Service4(string)

There exist two client processes, and they request the following services:

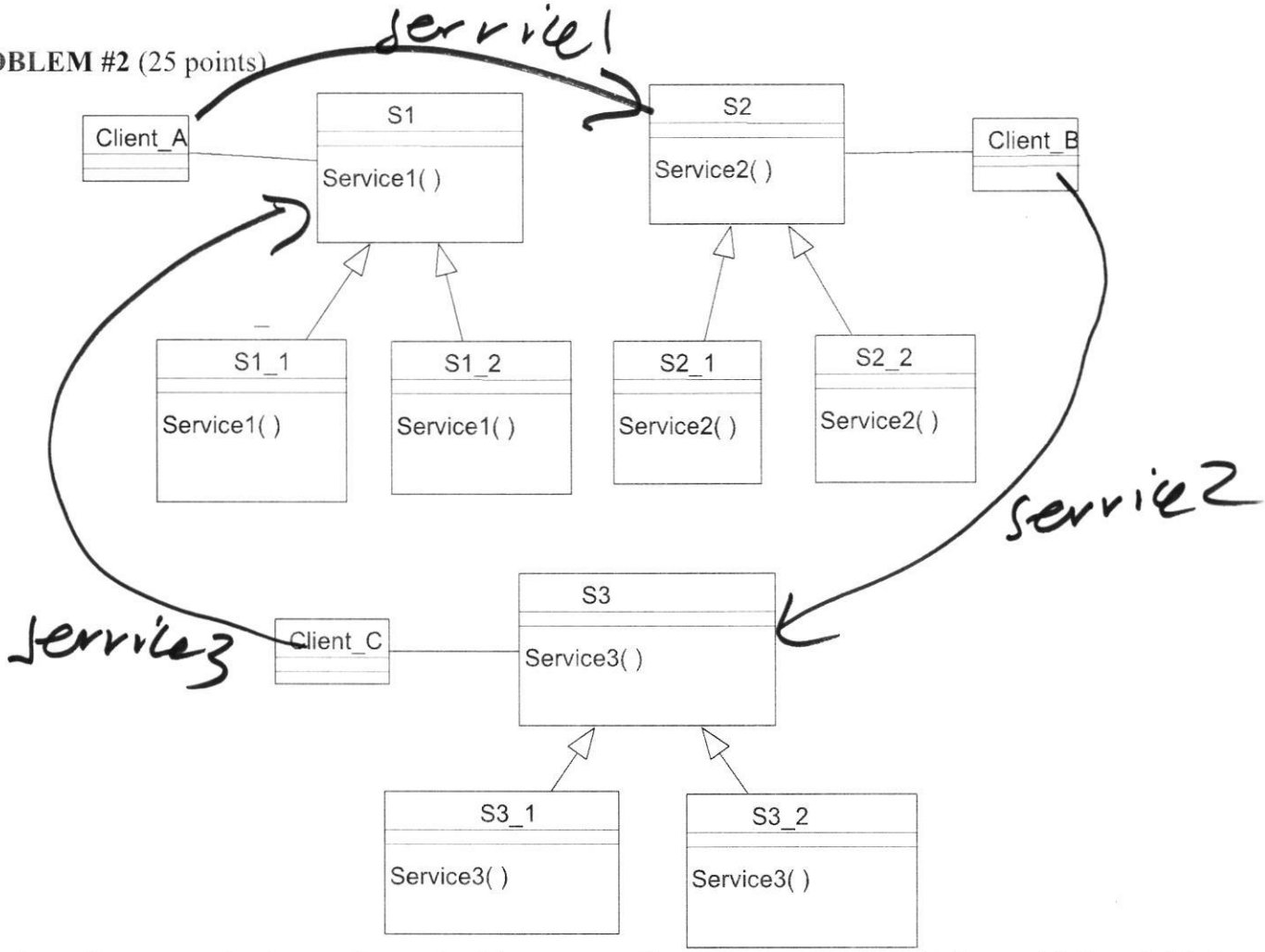
Client-A
void Service1(string, int, int)
void Service2(string, int)
int Service3(string)
float Service4(string)

Client-B
void Service1(string, int)
void Service2(string, int, int)
int Service3(string)
float Service4(string)

The client processes do not know the location (pointer) to servers that may provide these services. Devise a software architecture using a **Client-Broker-Server** architecture for this problem. In this design, the client processes are not aware of the location of the servers providing these services.

- Provide a class diagram for the proposed architecture. In your design, all components should be **decoupled** as much as possible.
- Provide the **pseudocode** for all operations of the following components/classes:
  - Broker
  - Client Proxy of *Client-A*
  - Server Proxy of *Server-2.*
- Provide a sequence diagram to show how *Client-A* gets "void *Service2(string, int)* service.

PROBLEM #2 (25 points)



A design of a system is shown above. In this system, *Client_A* uses objects of classes *S1_1* and *S1_2*, *Client_B* uses objects of classes *S2_1* and *S2_2*, and *Client_C* uses objects of classes *S3_1* and *S3_2*.

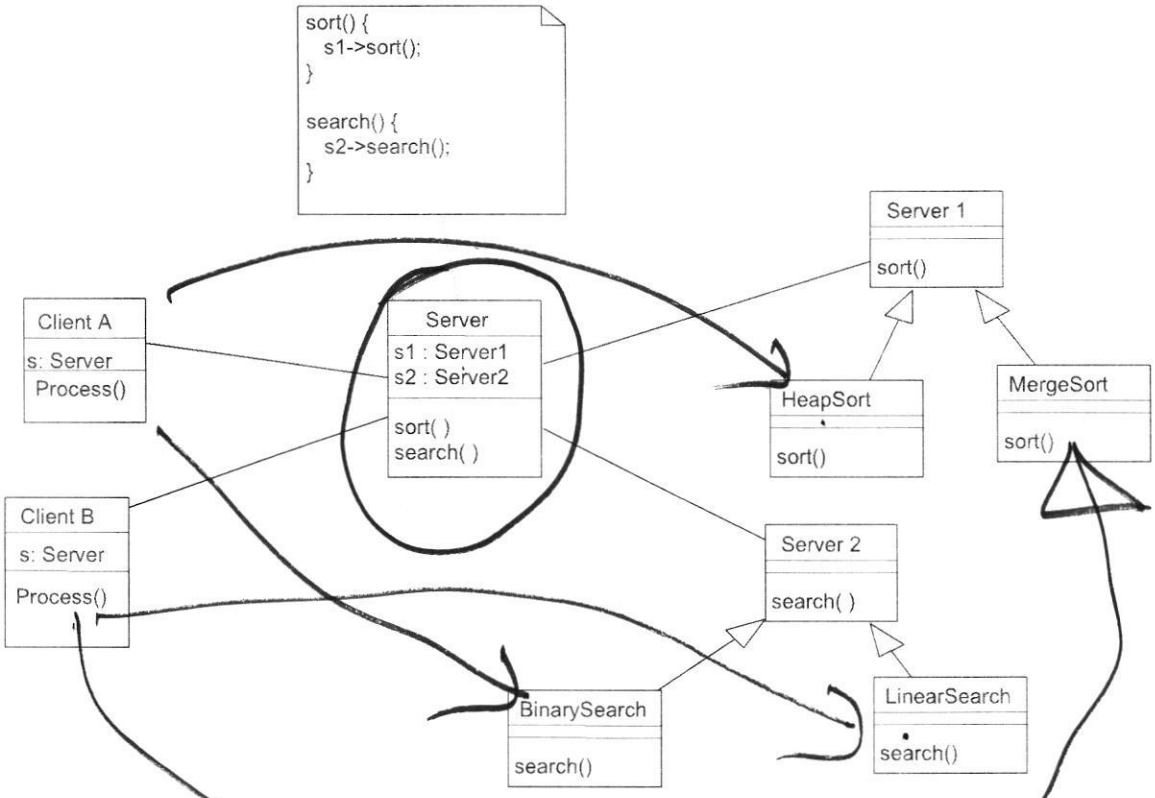
Client_A would like to use objects, operations *Service2()*, of classes *S2_1* and *S2_2* by invoking operation *Service1()*. *Client_C* would like to use objects, operation *Service1(),* of classes *S1_1* and *S1_2* by invoking operation *Service3()*. In addition, *Client_B* would like to use objects, operation *Service3(),* of classes *S3_1* and *S3_2* by invoking operation *Service2()*.

Provide a design with **minimal** modifications to the existing system using the **Adapter design pattern** in which (1) *Client_A* can use objects of classes *S2_1* and *S2_2* by invoking operations *Service1*(), (2) *Client_B* can use objects of classes *S3_1*, and *S3_2* by invoking operations *Service2(),* and (3) *Client_C* can use objects of classes *S1_1*, and *S1_2* by invoking operations *Service3()*. Notice that none of the classes shown in the above class diagram should be modified. Provide two solutions that are based on:

1. An **association-based** version of the Adapter pattern
2. An **inheritance-based** version of the Adapter pattern

Provide a class diagram for each solution. You do not have to provide any description for classes/operations of the above class diagram (only new classes/operations should be described using **pseudo-code**).

PROBLEM #3 (35 points)
Consider a design of the system shown below:



Client A and *Client B* get services, *sort()* and *search()*, directly from the *Server*. However, the *Server* gets the appropriate services from *HeapSort* or *MergeSort* servers for *sort()* service. In addition, the *Server* gets the appropriate services from *BinarySearch* or *LinearSearch* servers for *search()* service.

In this design, clients do not have control where the services are coming from. However, *ClientA,* by invoking *sort()* and *search()* on the *Server,* would like to get *sort()* service from *HeapSort* server and *search()* service from *BinarySearch* server. On the other hand, *ClientB,* by invoking *sort()* and *search()* on the *Server,* would like to get *sort()* from *MergeSort* server and *search()* from *LinearSearch* server. Notice that the current design does not support this option.

Use the **abstract factory** design pattern to solve this problem. In your solution, the *Client* classes should be completely **decoupled** from the issue of invoking services by the *Server* for appropriate versions of *sort()* and *search()*. Notice that none of the classes (and operations) shown in the above class diagram should be modified. However, new operations/classes can be introduced.

- Provide the class diagram and briefly describe the responsibility of each class and the functionality of each operation using **pseudo-code**. In your design, all components should be **decoupled** as much as possible. You do not have to provide any description for classes/operations of the above class diagram (only new classes/operations should be described).
- Provide a sequence diagram to show how *ClientB* gets *sort() service* from *MergeSort* server and *search()* service from *LinerSearch* server by invoking *sort()* and *search()* on the *Server*.