Exam #1

Wednesday, October 1
at 5:00 pm

Closed books and notes

The coverage is posted.

A sample exam with
solution

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

OO design patterns

Design process

↓

Design problem

↓

OO design patterns

↓

revise the design solution(s)

Architectural Patterns

specification

⇓

Design system
(subsystem)

⇓

Architectural
Patterns

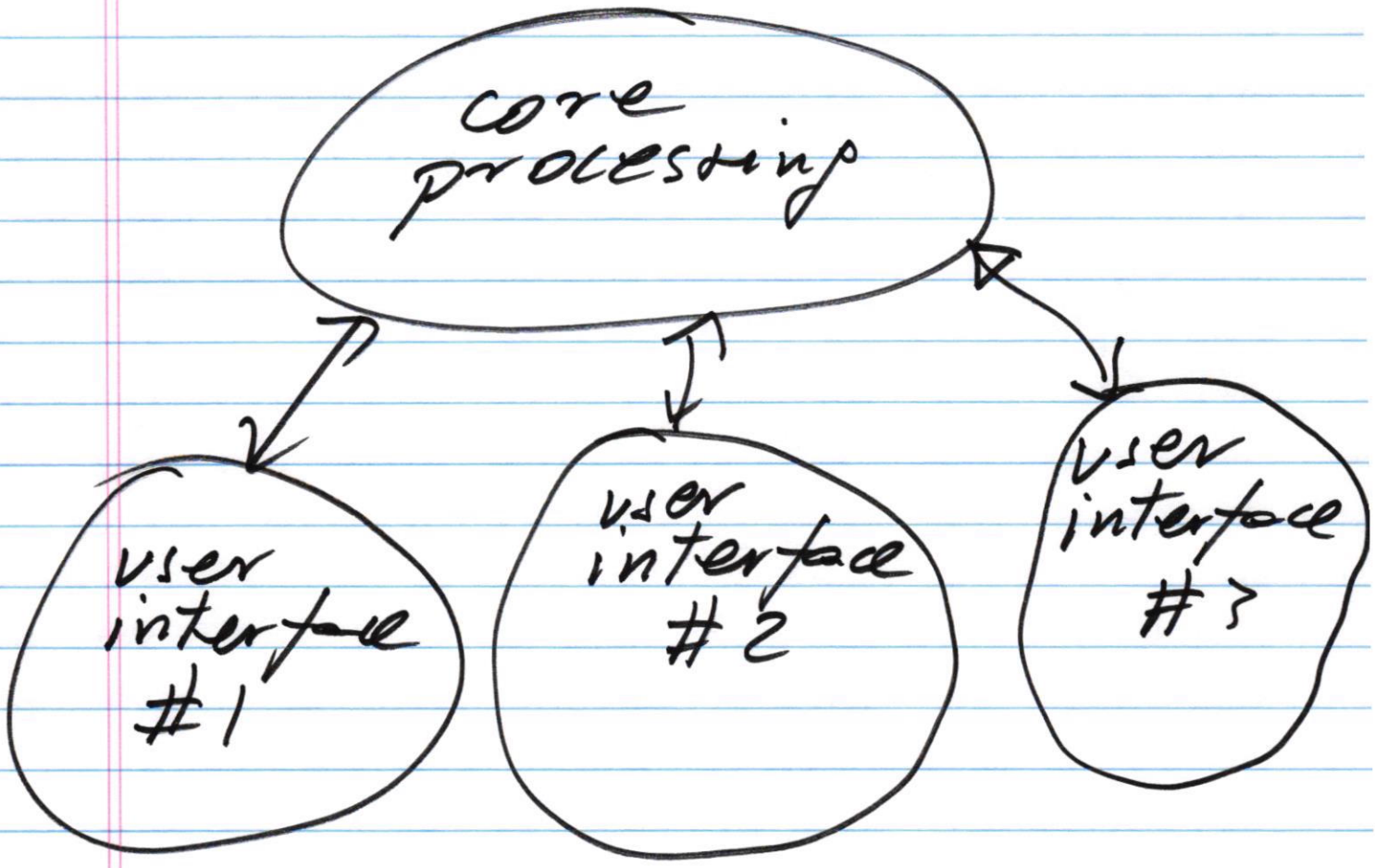
⇓

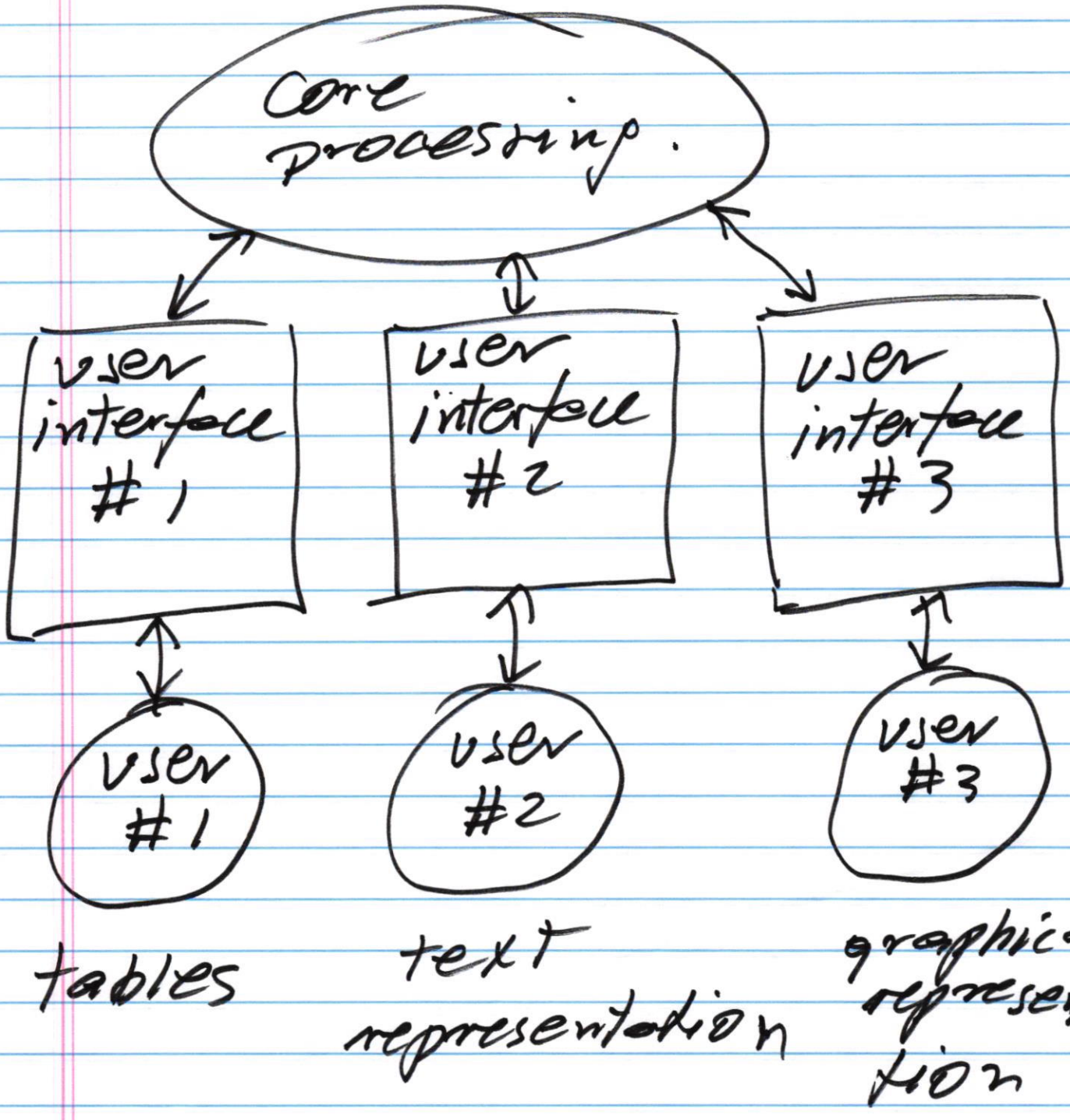
Architectural
structure of
the system

Architedural Patterns

help to identify a "fundemental" structure of the system

Interactive Systems




core processing.
user interface #1
user interface #2
user interface #3
user #1
user #2
user #3
tables
text representation
graphical representation

MVC architectural pattern

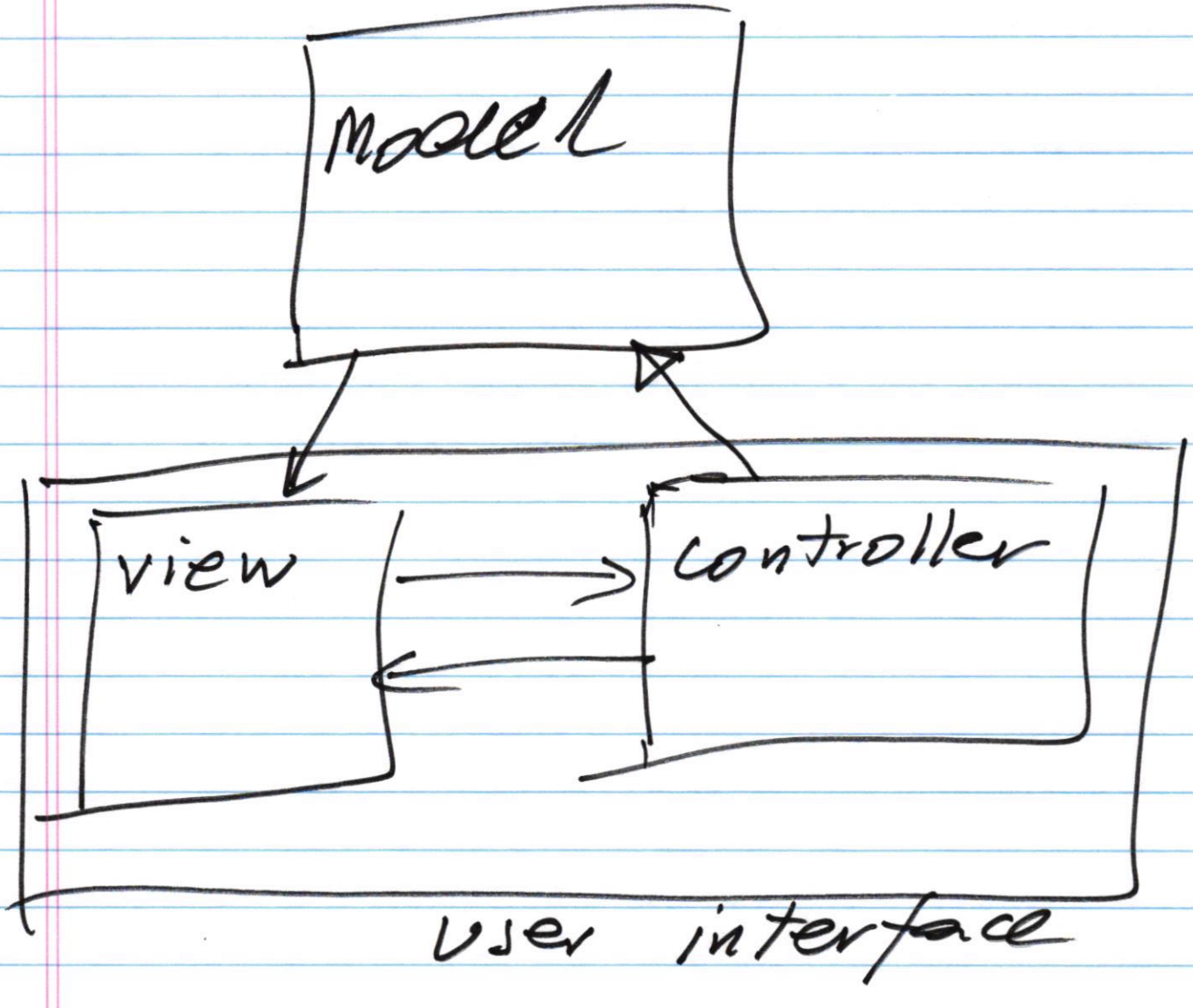
Model-View-Controller pattern

Interactive system consists of 3 types of components

1. Model (Processing)
   core processing.
2. View (Output)
   displays info to users
3. Controller (Input)
   handles user inputs.

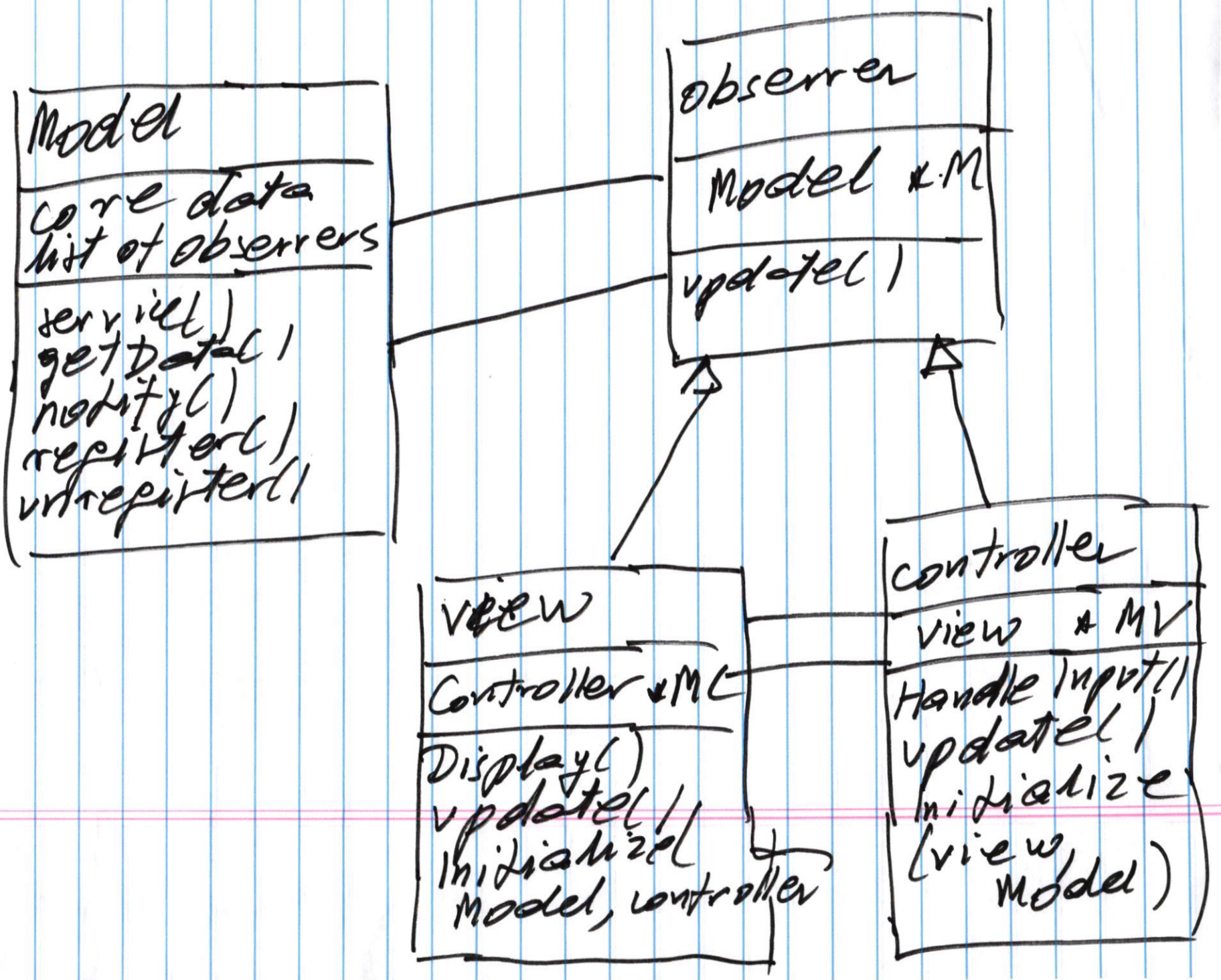
user interface

view component
+
controller -ll-



Assumptions

1. the same information is represented differently to different UIs (User Interfaces)
2. Any data changes in the Model should be immediately available to UIs
3. Any changes to UIs should not affect the Model
4. Adding new UIs should be very easy (even possible during run time)


Model
core data
list of observers
service()
getData()
notify()
registerO()
unregisterO()
observer
Model *M
update()
view
Controller *MC
Display()
update()
Initialize()
Model, controller
controller
view *MV
HandleInput()
update()
Initialize
(view
Model)

View	Model
Initialized()	→ repiter(), getData()
update()	→ getData()
destructor()	→ unrepiter()

Controller	Model
Initialize()	→ repiter(), getData()
update()	→ getData()
HandleInput()	→ servie()
destructor	→ unrepiter()

view

initialize() → initialize()

controller

Model

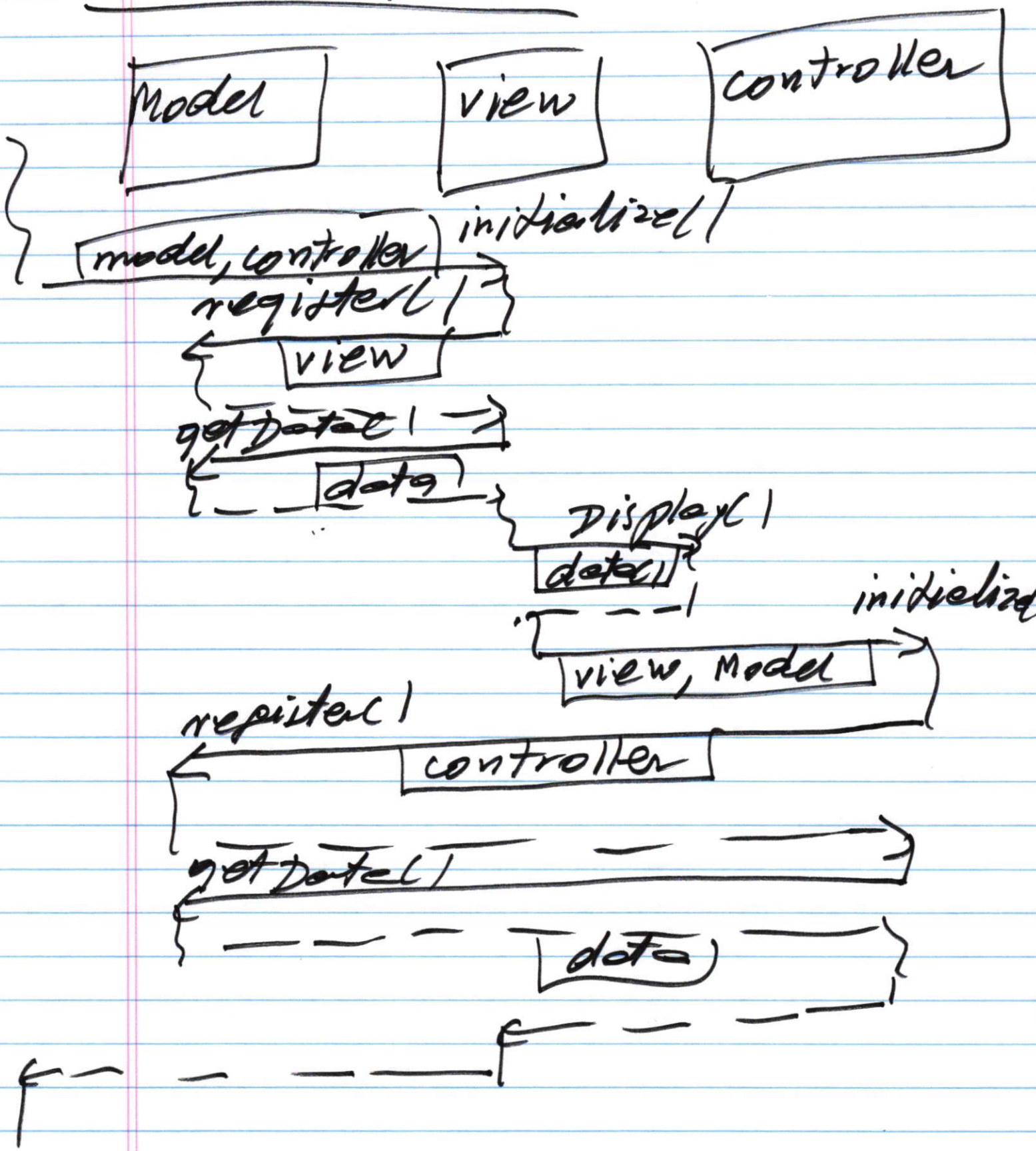
notify() → update()

observers

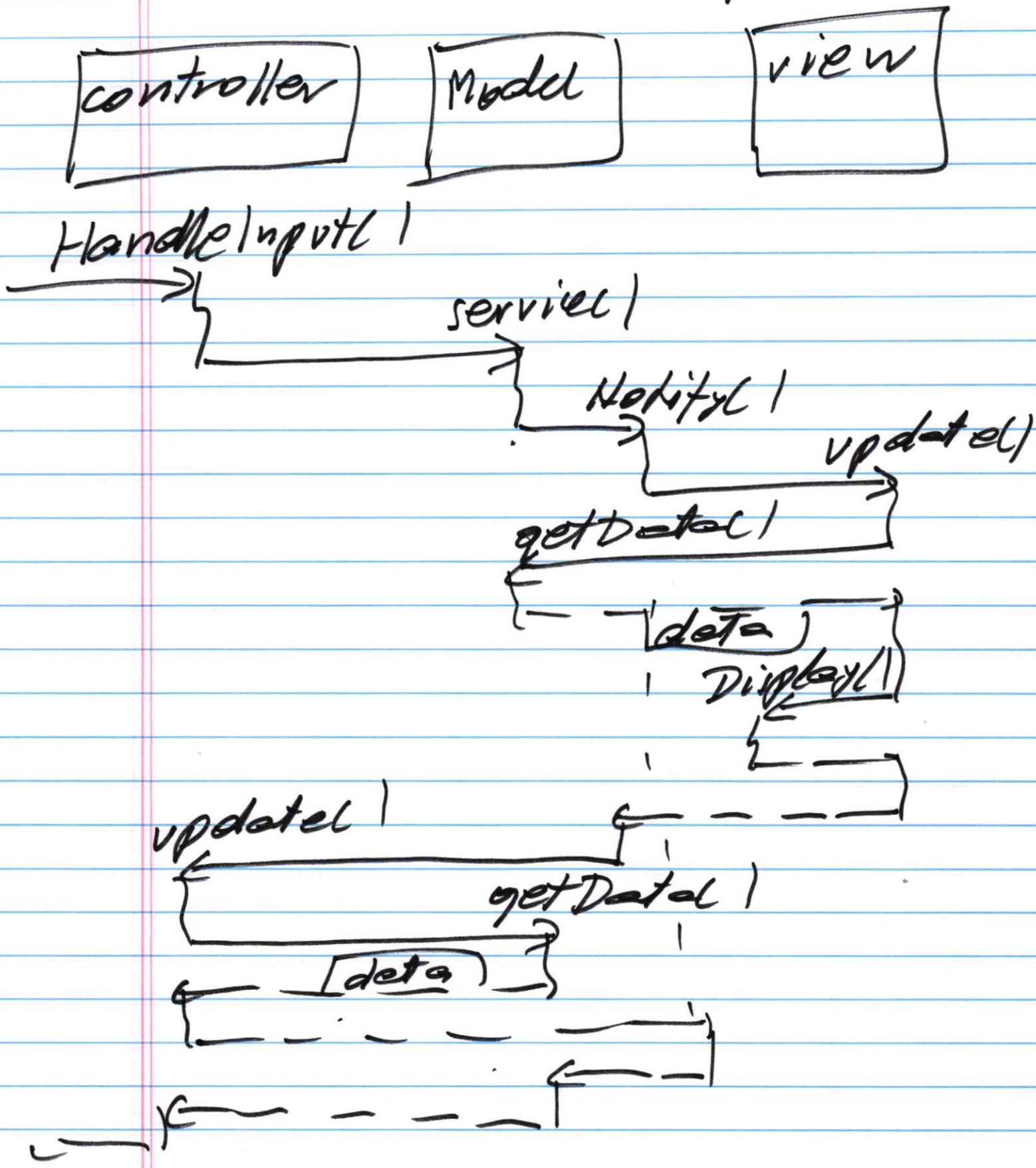
view/controller

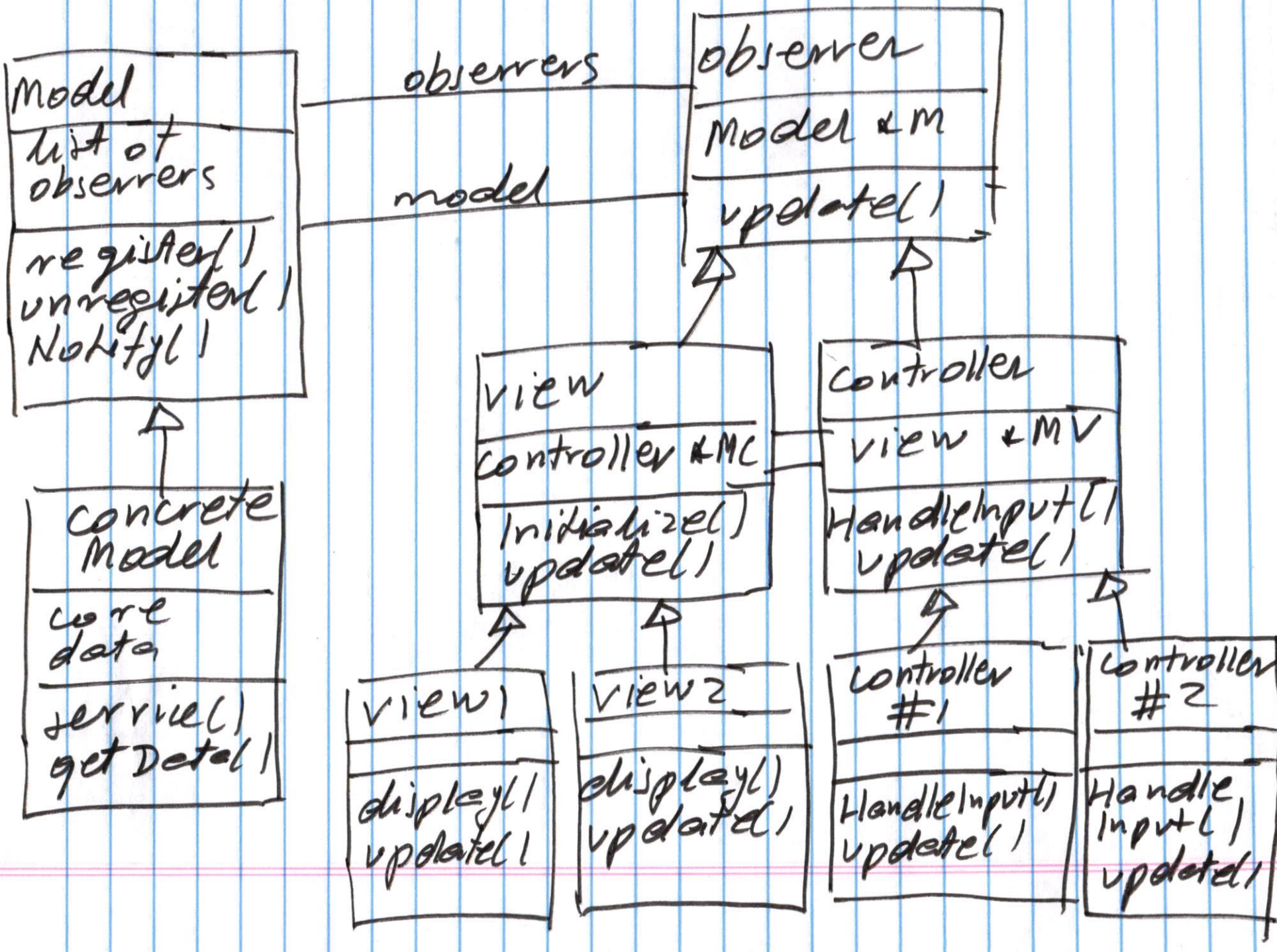
initialization of VIs

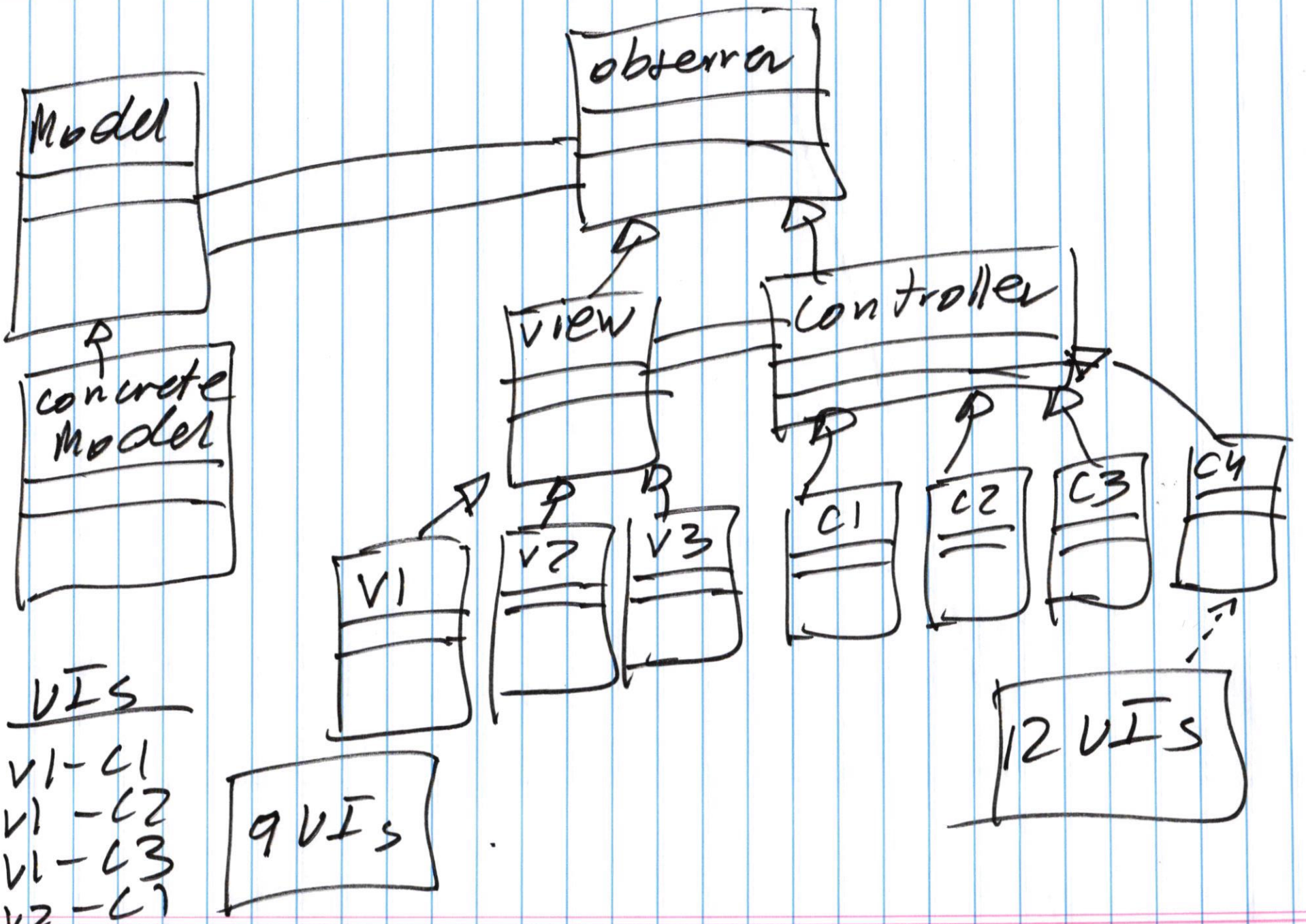
scenario #1



scenario #2 handling input




Model
list of observers
registerO()
unregisterO()
NotifyO()
observers
model
observer
Model *M
updateO()
view
controller *MC
InitializeO()
updateO()
controller
view *MV
HandleInputO()
updateO()
concrete Model
core data
serviceO()
getDataO()
view1
displayO()
updateO()
view2
displayO()
updateO()
controller #1
HandleInputO()
updateO()
controller #2
HandleInputO()
updateO()



VIs

V1 - C1
V1 - C2
V1 - C3
V2 - C1
V2 - C2
.
.
.

MVC Pattern

Advantages

* very easy to create multiple VIs of different types
* new VIs can be easily added
* supports modification
* Any changes to VIs do not affect the model.

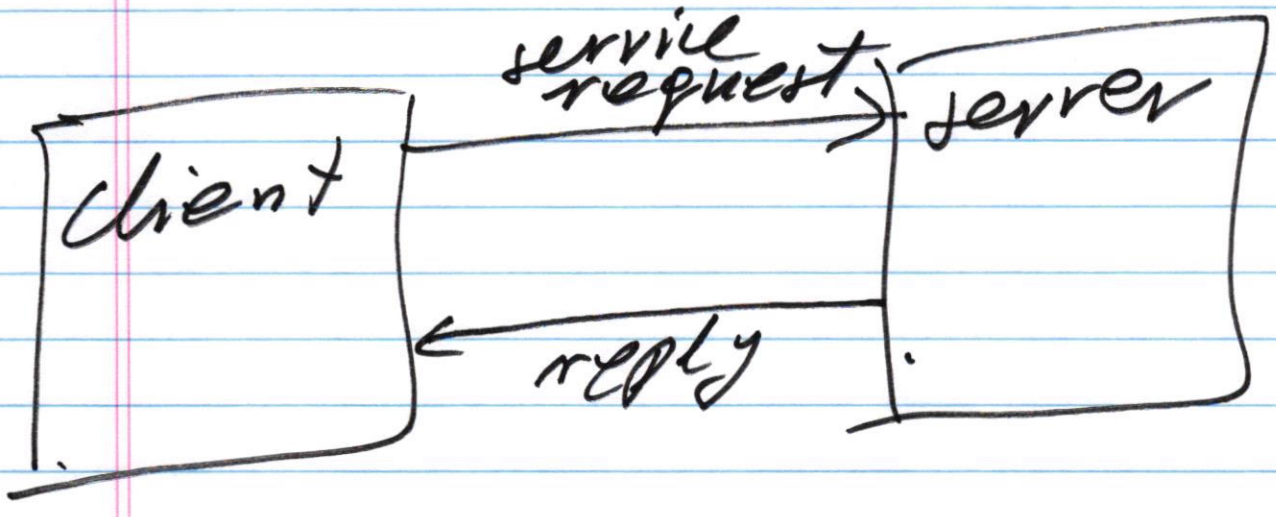
Disadvantages

- increased complexity.
- performance issue
- when there is a small # of VIs, there is no need to use this model

Client-server architecture

servers: provide services

clients: request —"—



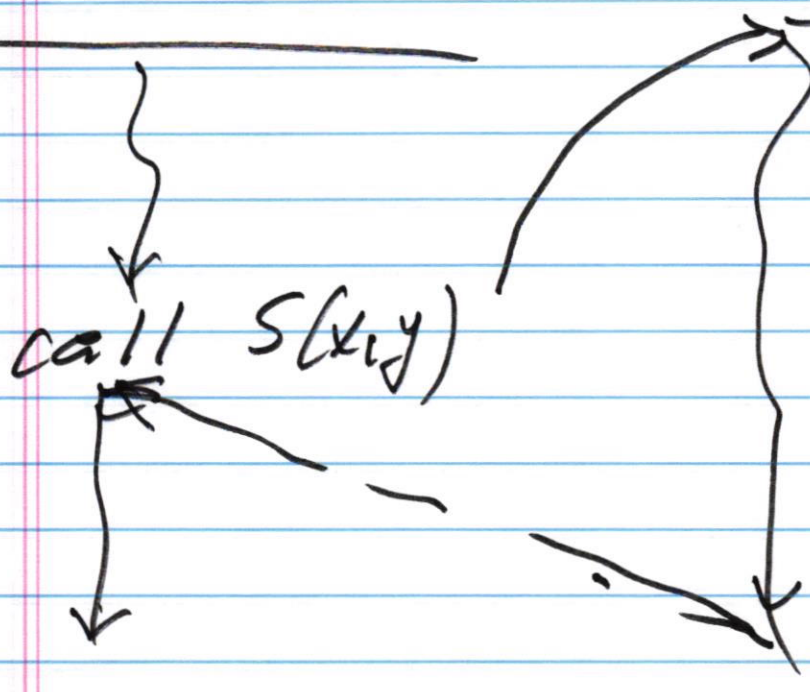
modular design

client

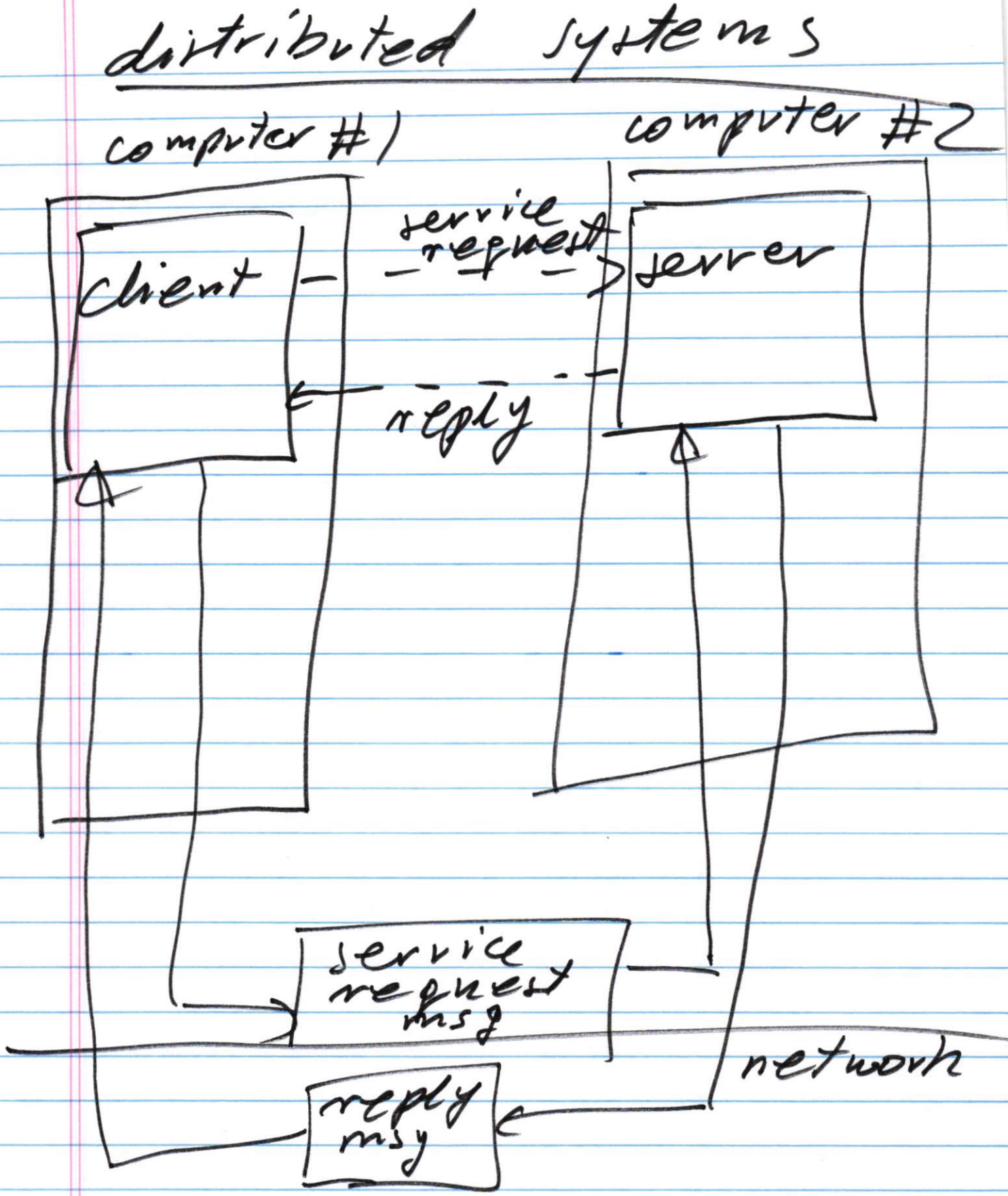
server

module "A"

module "S"



distributed systems



Problem

A client wants to get a service

where is a server that provides the service?

---

distributed system.

client: on which computer the server is located?

OO systems

servers → objects (dynamic objects)

client

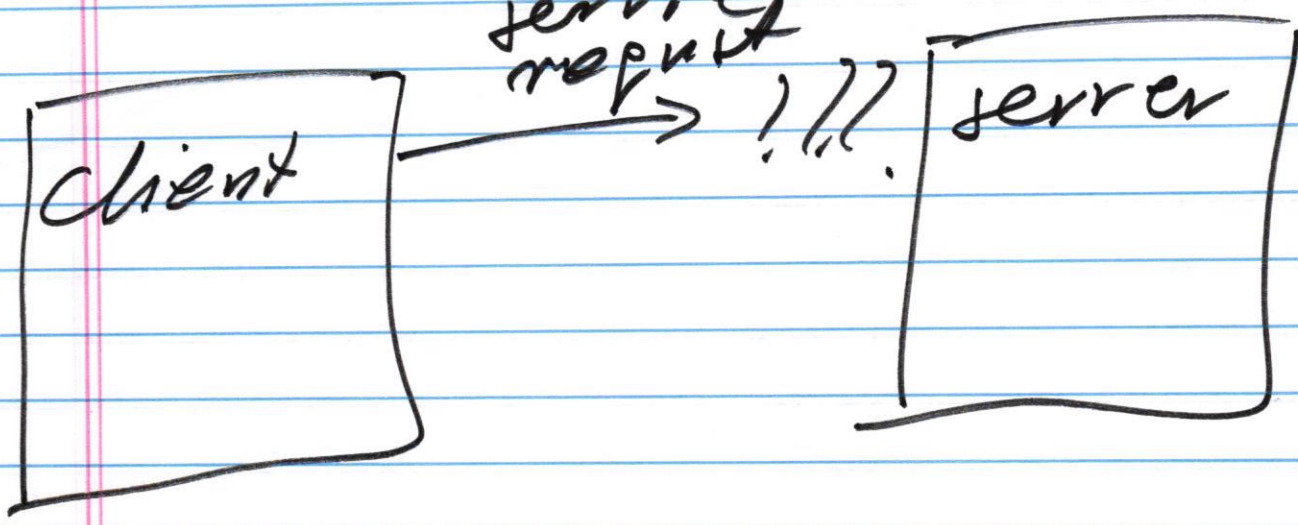
OO system

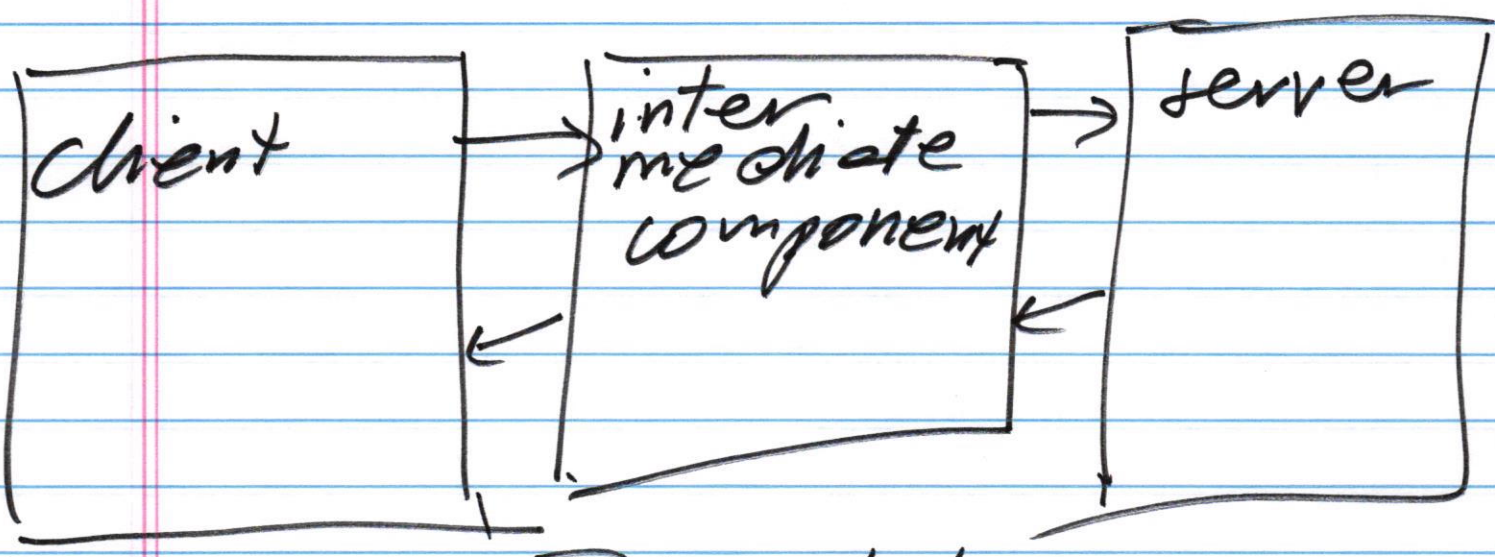
client

server *s

s = ???

s -> service()



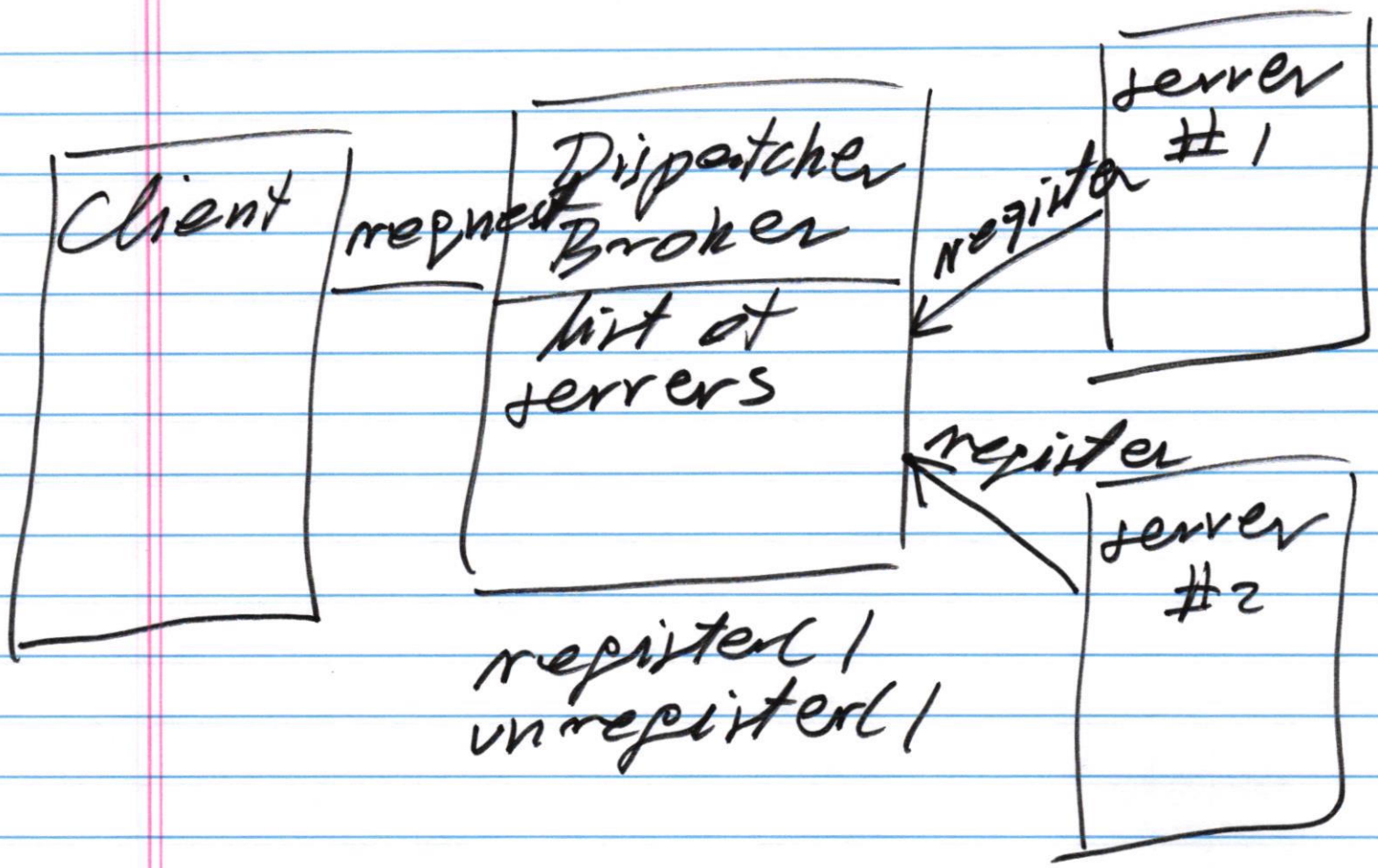


* Dispatcher

* Broker

1. Client-Dispatcher-Server

2. Client-Broker-Server



register()

1. server location
2. service specification
   - (a) service name
   - (b) service signature
   - (c) service semantic