Homework #2 is posted

Model-Driven Architecture

MDA

- multi-year architecture paradigm

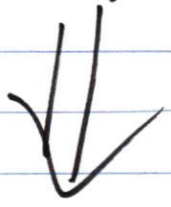
- We want to seperate
  - platform independent "aspects" of the system

FROM

  - platform & specific "aspects" of the system

Idea:

to create a platform independent model (PIM) that will be "stable"



will not change for many years/versions

PIM: Meta Model

Platform

* operating system
* programing language
* data base
* network.
* hardware.
* data structures
* algorithms
* ...

Data

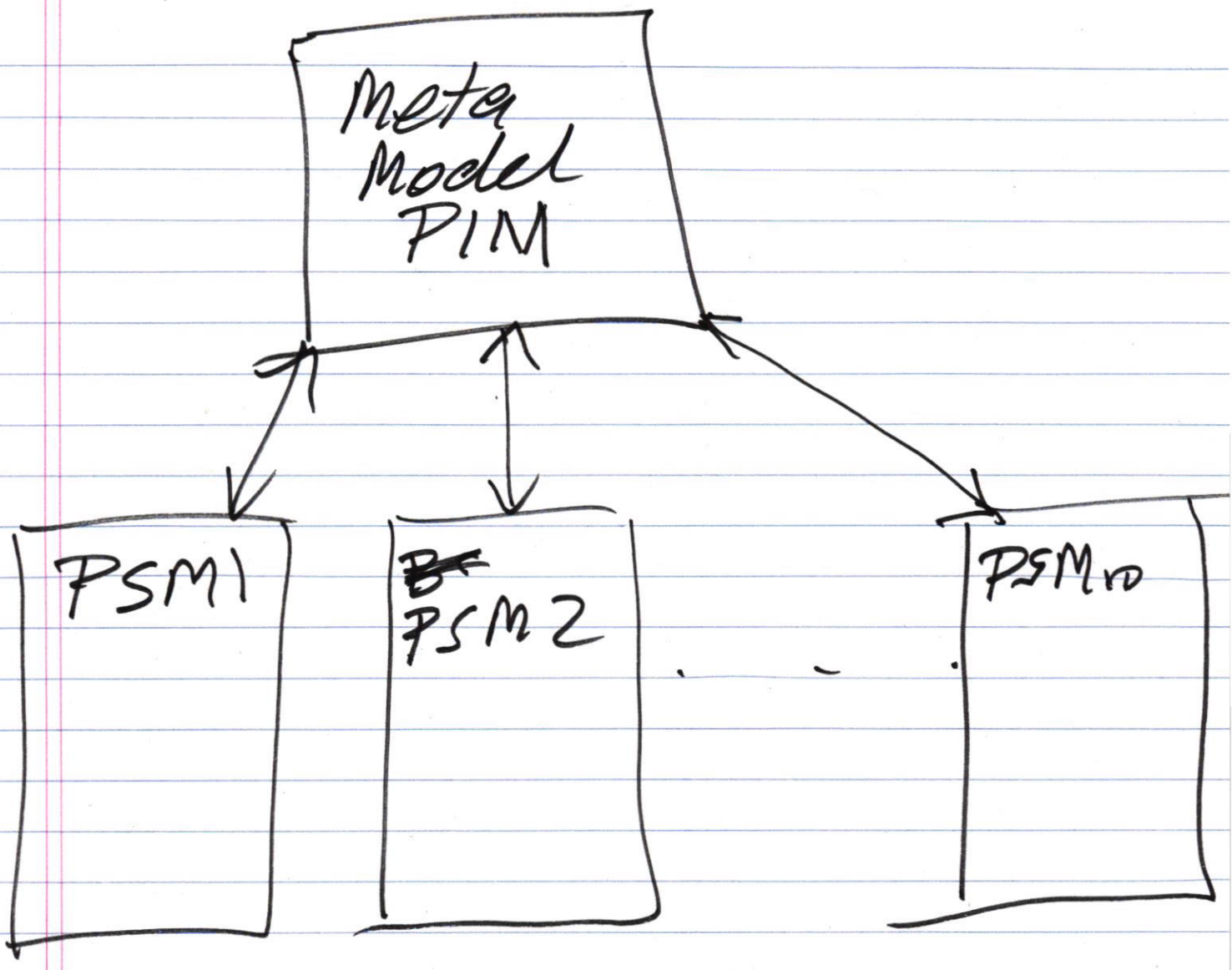
Data-oriented meta models

⇓

for data-base systems

Behavior

We want to concentrate on meta models that capture "meta" behavior of a family of systems.



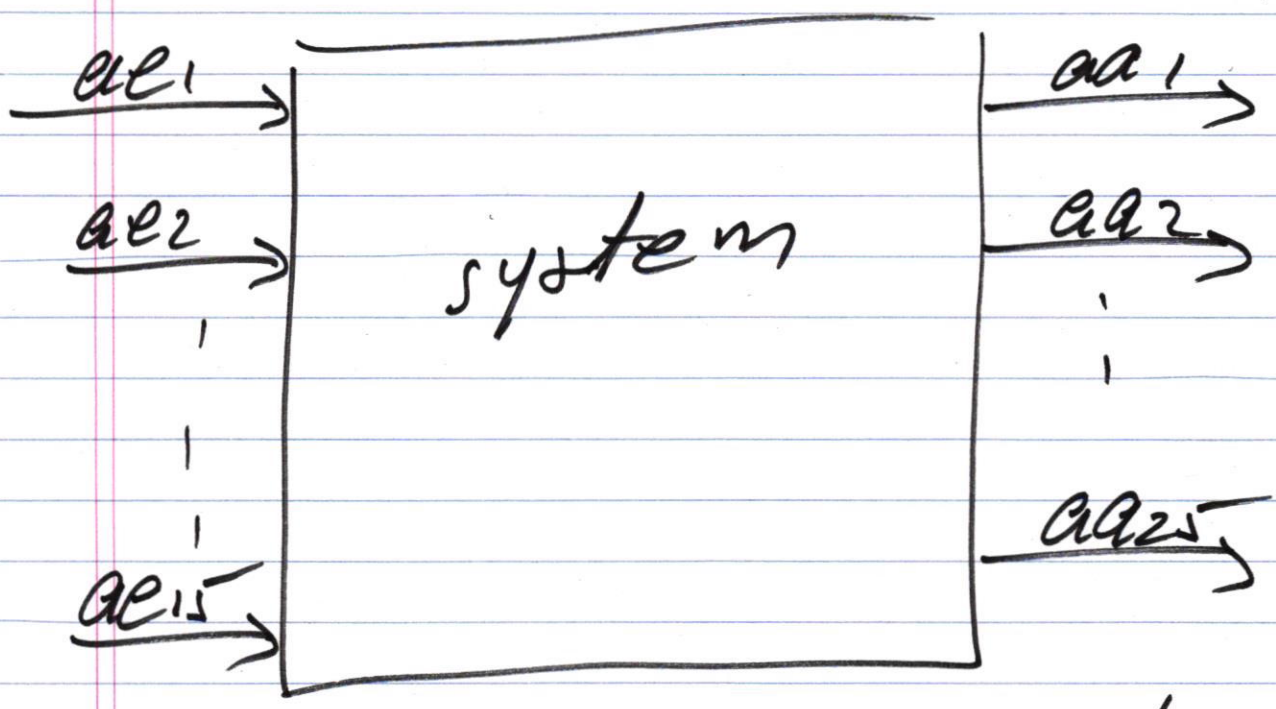
PSM: Platform Specific Model

Meta Model related to behavior

- should capture the behavior of a family of systems that is "platform" independent
- We want to seperate the behavior from different OS, DB, network, ...
- should be executable.
  - we should be able to test it
  - validate/verify the meta model.

Model Driven Architecture

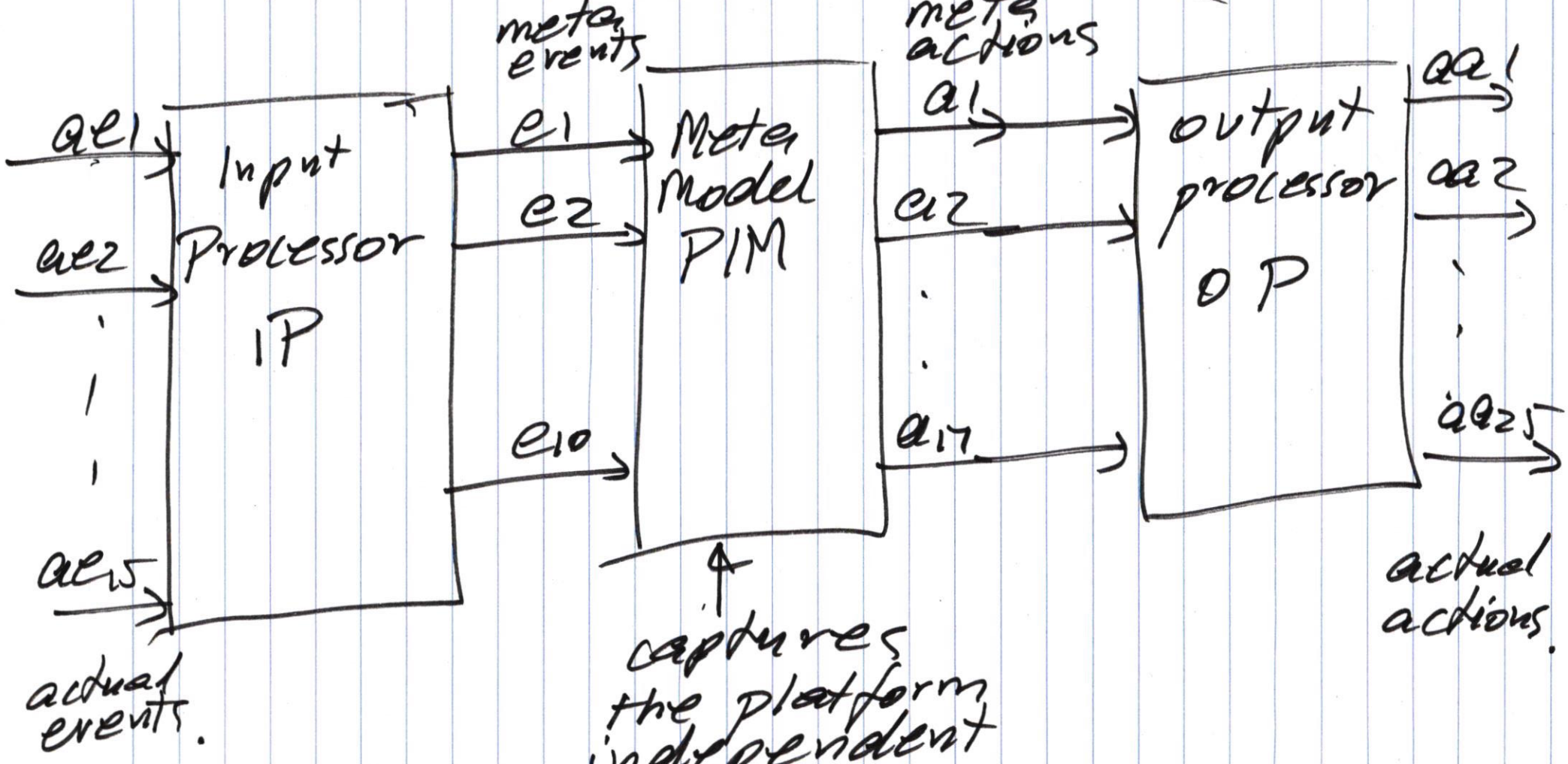
event based systems.



actual events.

actual actions (output, outcome)

Model-Driven Architecture

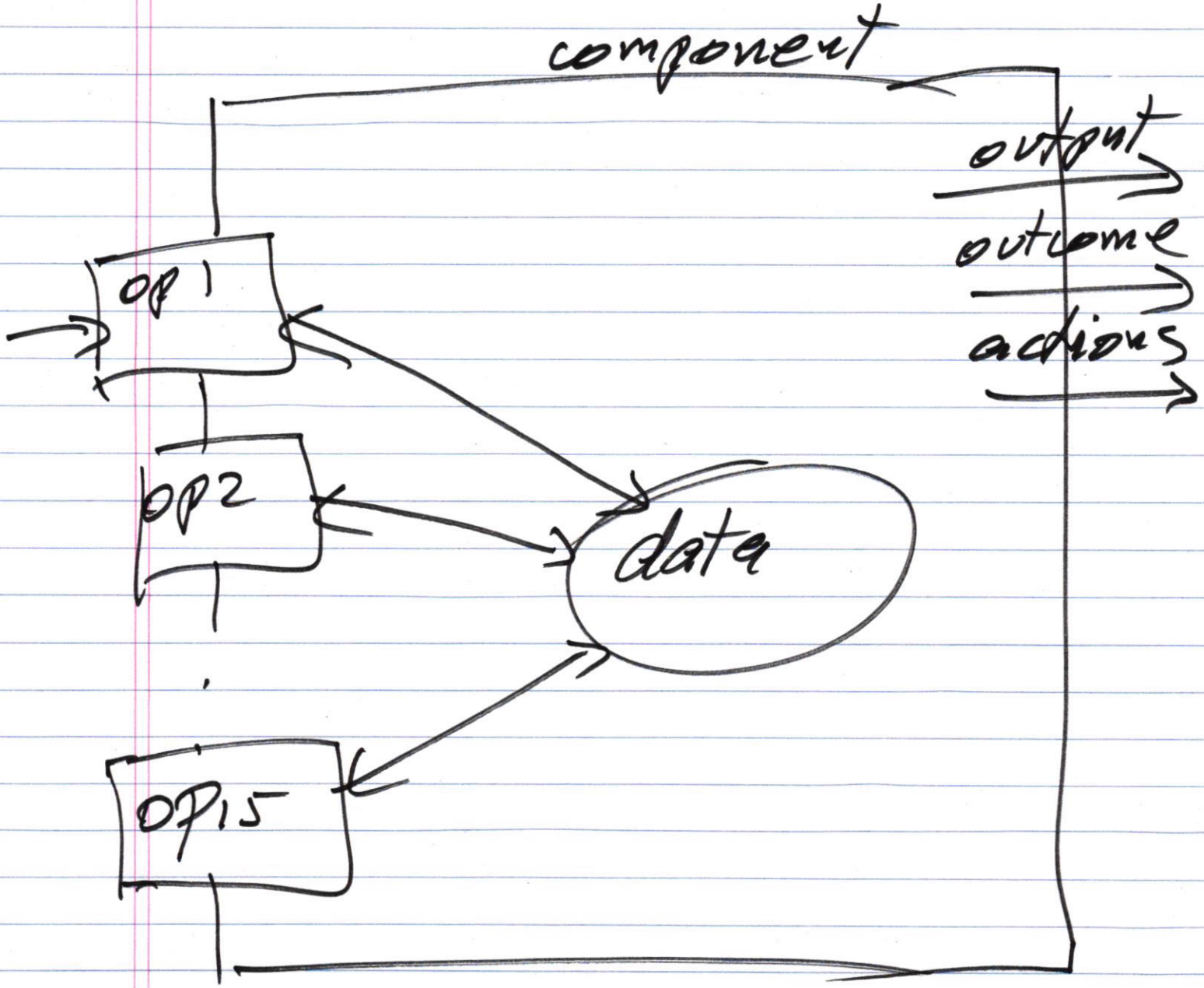


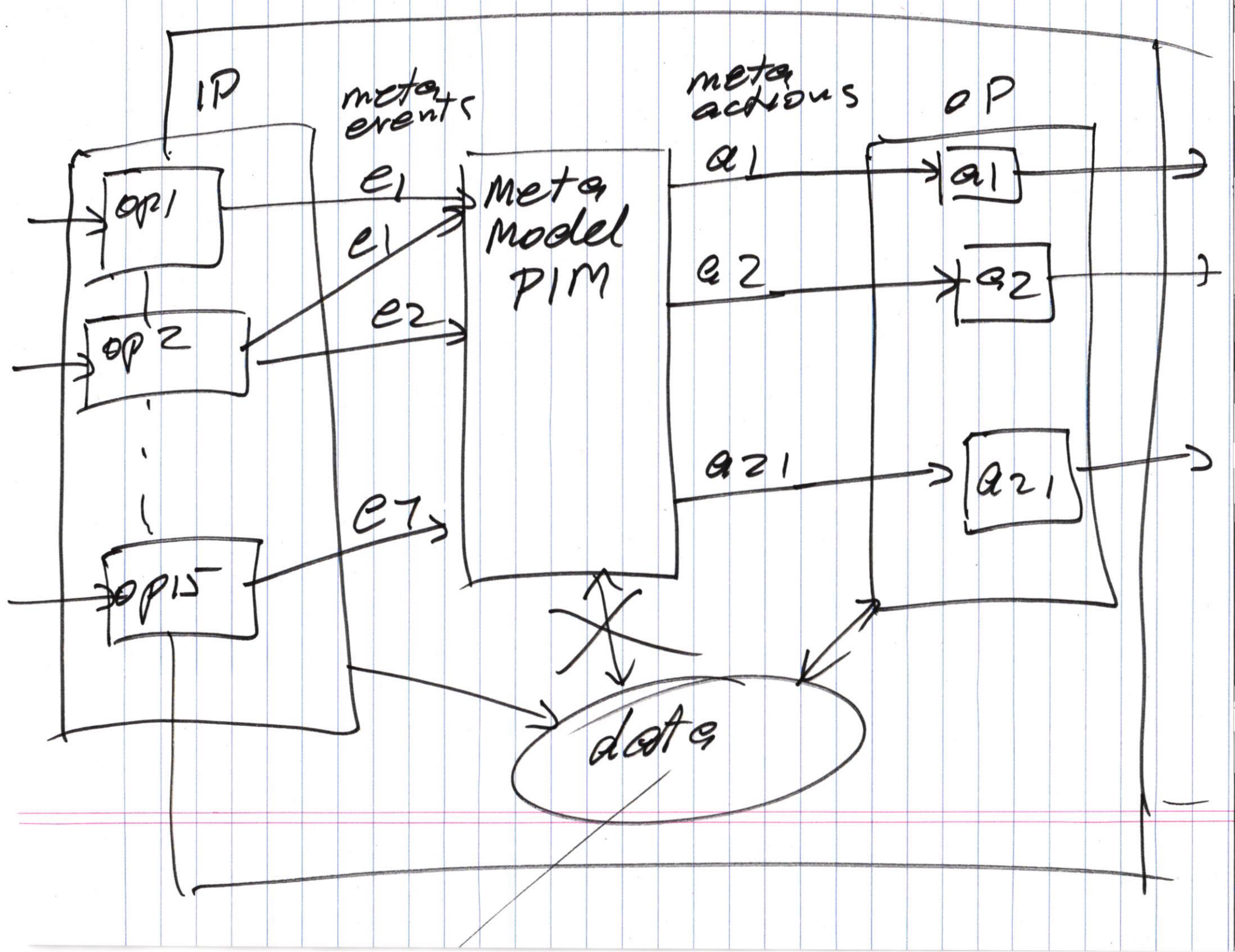
$$PSM = \begin{cases} IP \\ OP \end{cases}$$

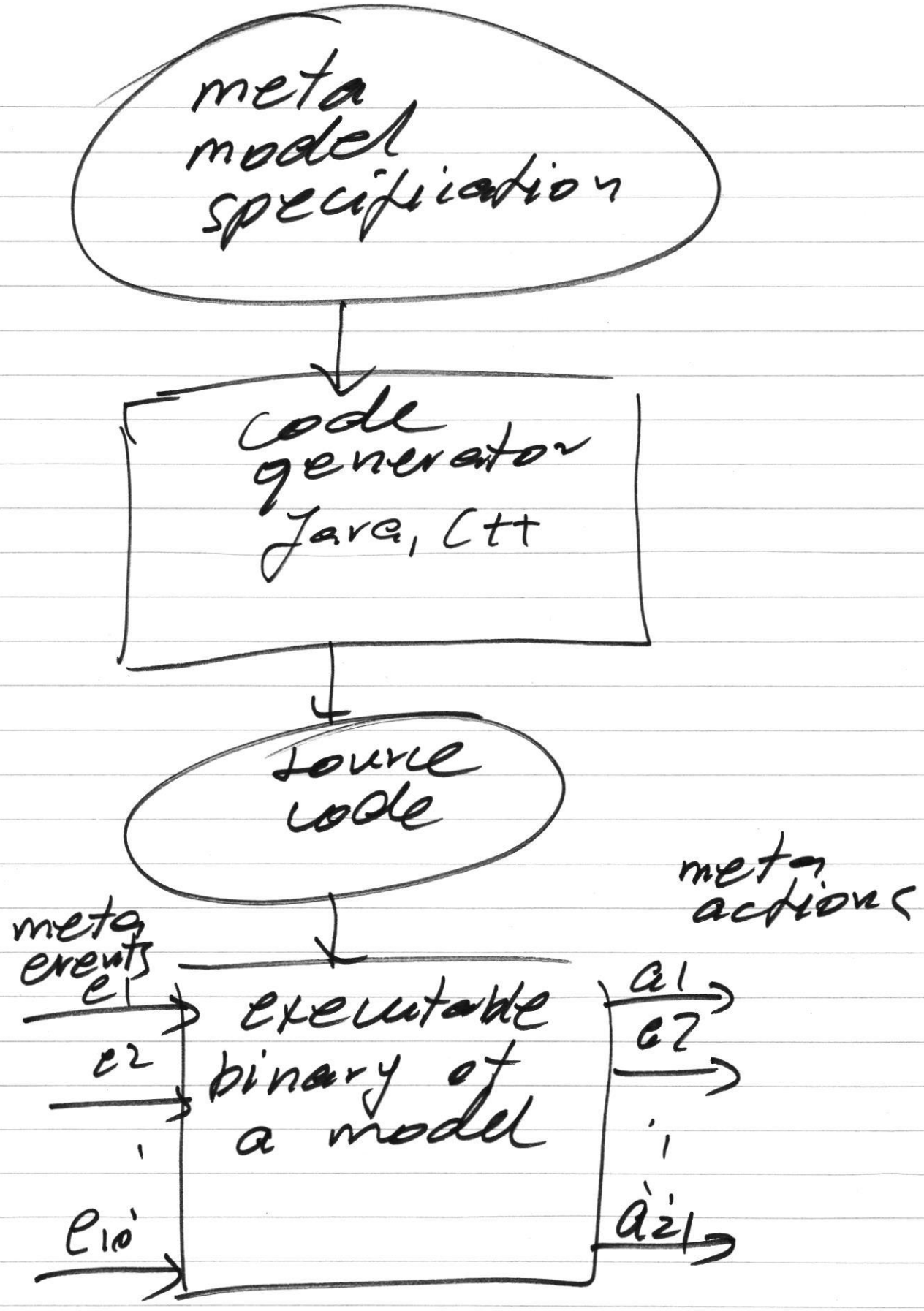
a component

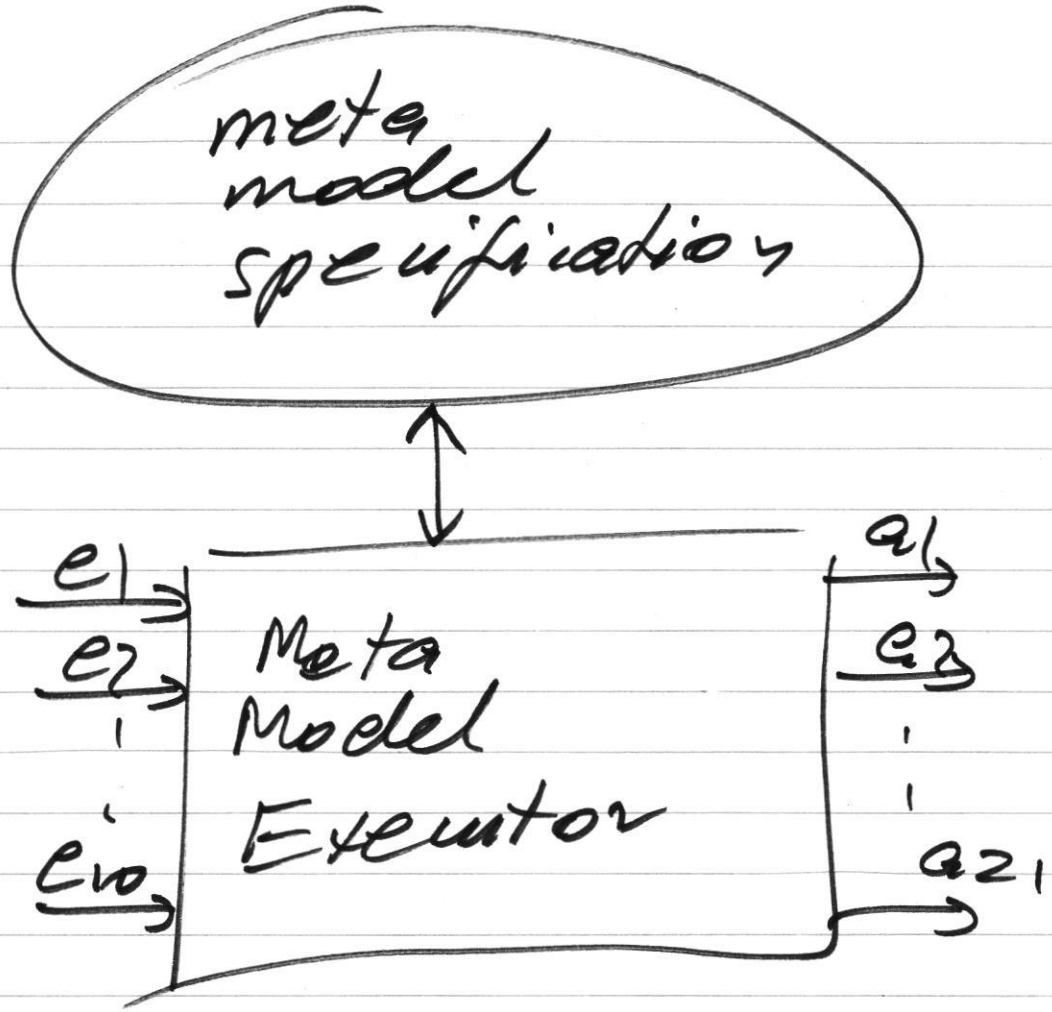
a set of operations

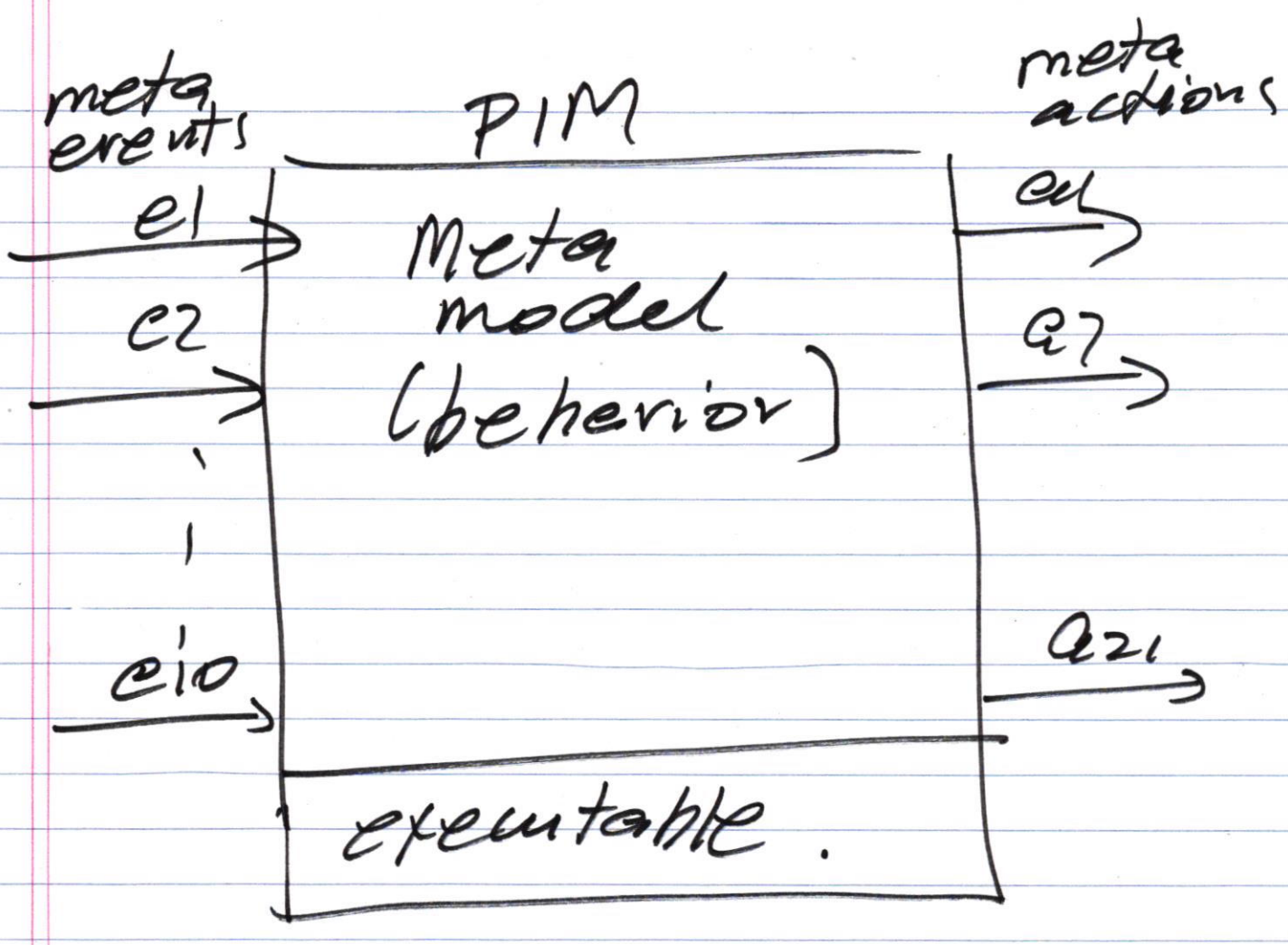
$op_1, op_2, \ldots, op_{15}$




IP
meta events
meta actions
OP
op1
op2
op15
e1
e1
e2
e7
Meta Model PIM
a1
a2
a21
a1
a2
a21
data


meta model specification
code generator Java, C++
source code
meta events
e1
e2
e10
executable binary of a model
meta actions
a1
a2
a21


meta
model
specification
e1
e2
e10
Meta
Model
Executor
q1
q3
q21



ei: meta event
ai: meta actions

} platform independent

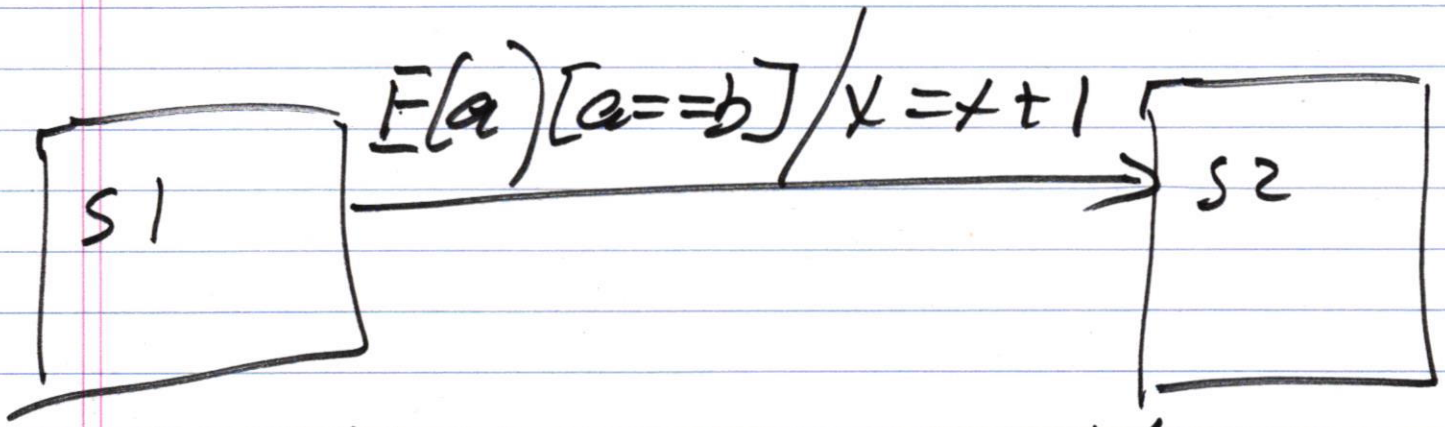
Modeling languages

that can capture meta behavior.

* FSM: Finite State Machine
* EFSM: Extended Finite State Machine
* VFSM: Virtual Finite State Machine
* ...

EFSM

- states
- transitions
- conditions
- variables/data
- actual events
- actual actions



EFSM is frequently platform dependent!!

ATM

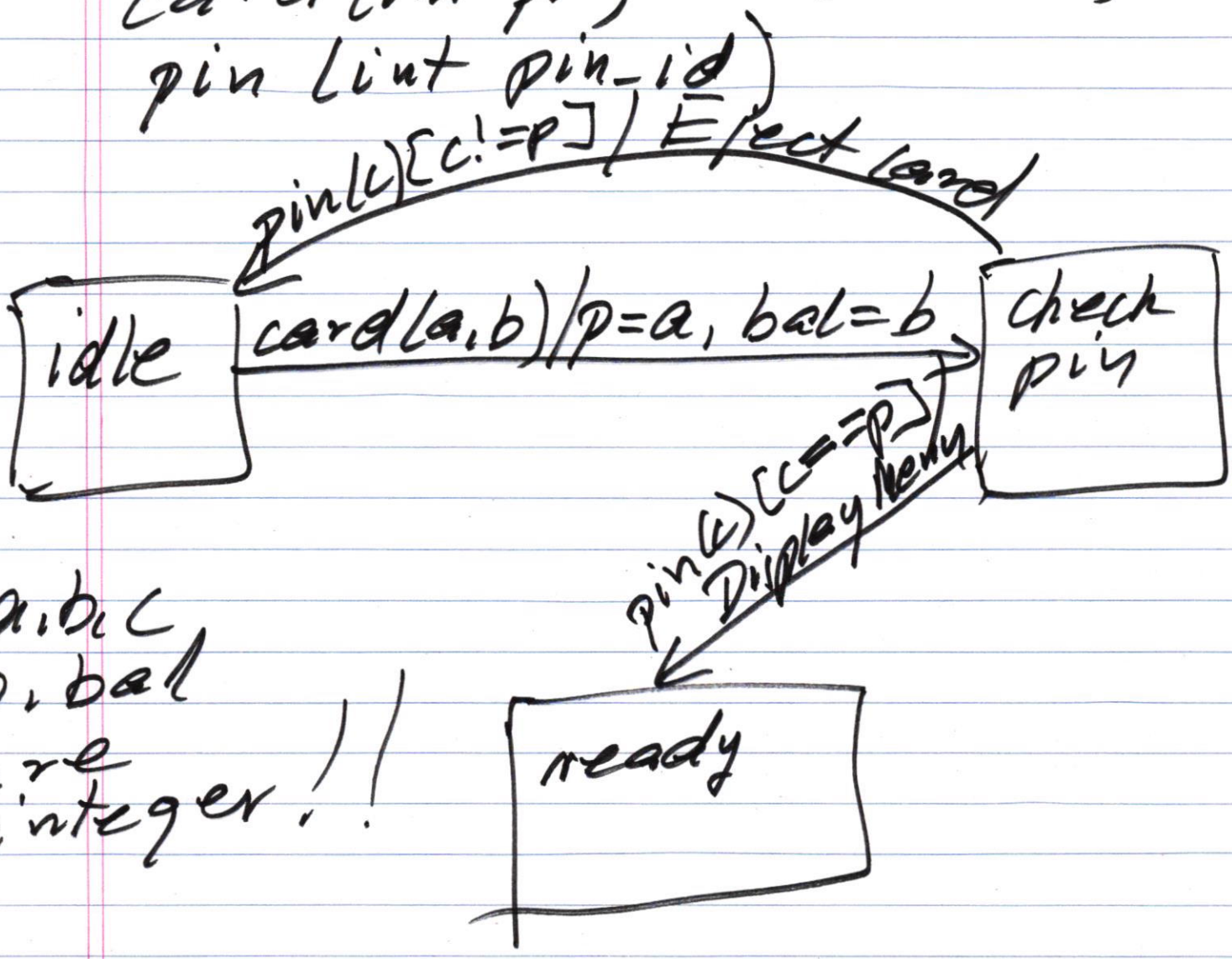
card(pin, balance)

pin(pin_id)

version I

card(int pin, int balance)

pin(int pin_id)



a, b, c, p, bal are integer!!

version II

card (float balance, string pin)
pin (string pin_id)

pin, balance are different data types than in version I

version III

card (string pin, int balance, string user_id)
pin (string pin_id)

VFSM

Virtual Finite State Machine.

- states
- transitions.
- virtual events
  - virtual actions
  - conditions

---

virtual event: name (Boolean variable)

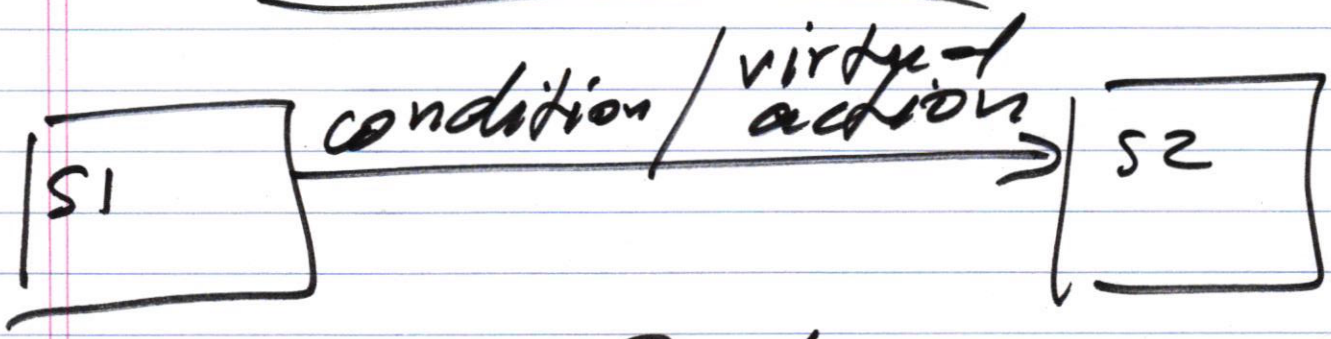
virtual action: name

ATM EFSM

- card(int pin, int bal)
- card(float bal, string pin)
- card(bal, P1, P2, pin)

VFSM: card

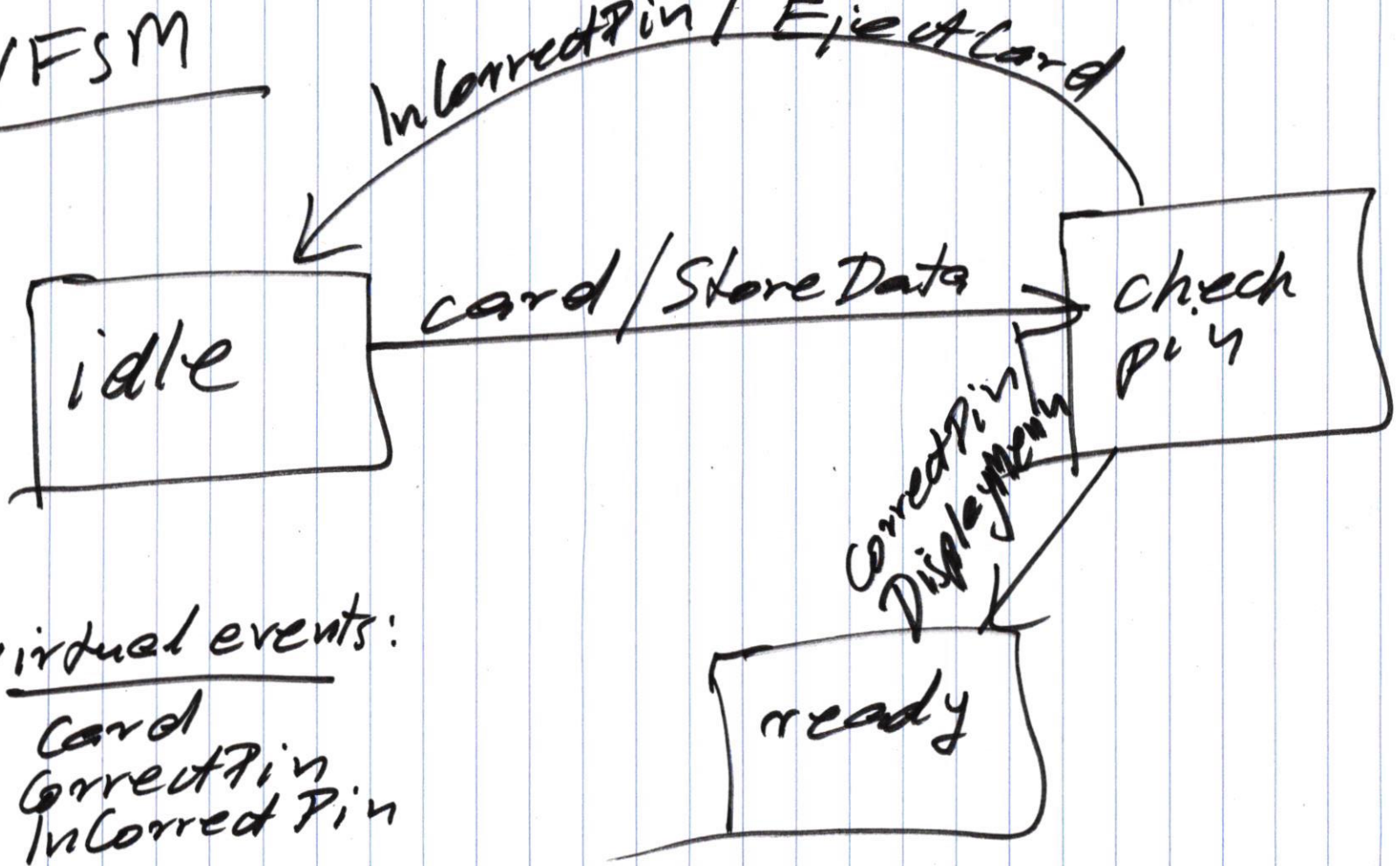
VFSM



Condition: Boolean expression

1. model is in S1
2. virtual event e is invoked
   e: Boolean variable.
   "True" value is assigned to e
3. condition is evaluated (e or e5)
   suppose condition evaluates to true
4. Transition from S1 to S2 and virtual action is invoked.

VFSM

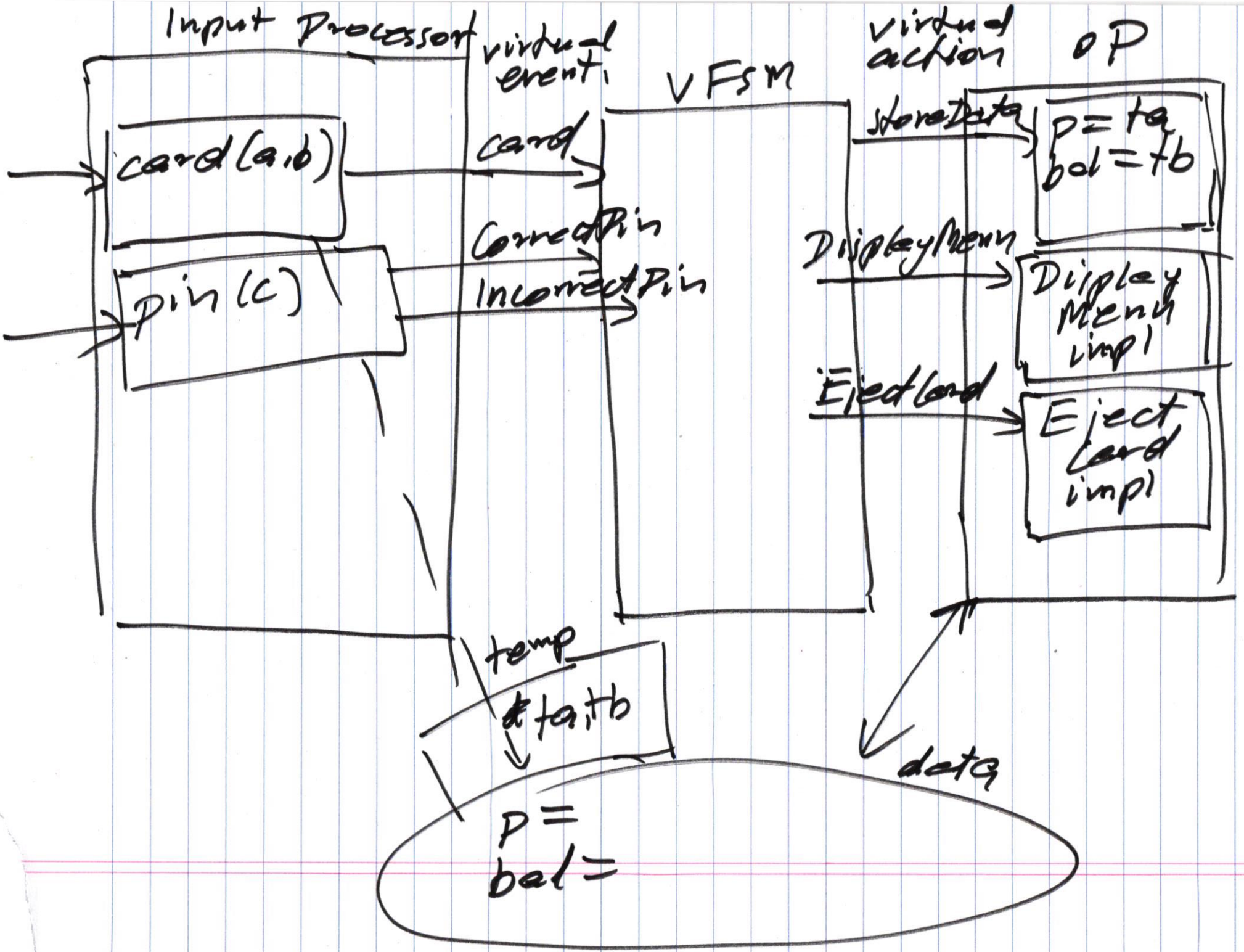


Virtual events:

- Card
- CorrectPin
- InCorrectPin

Virtual actions:

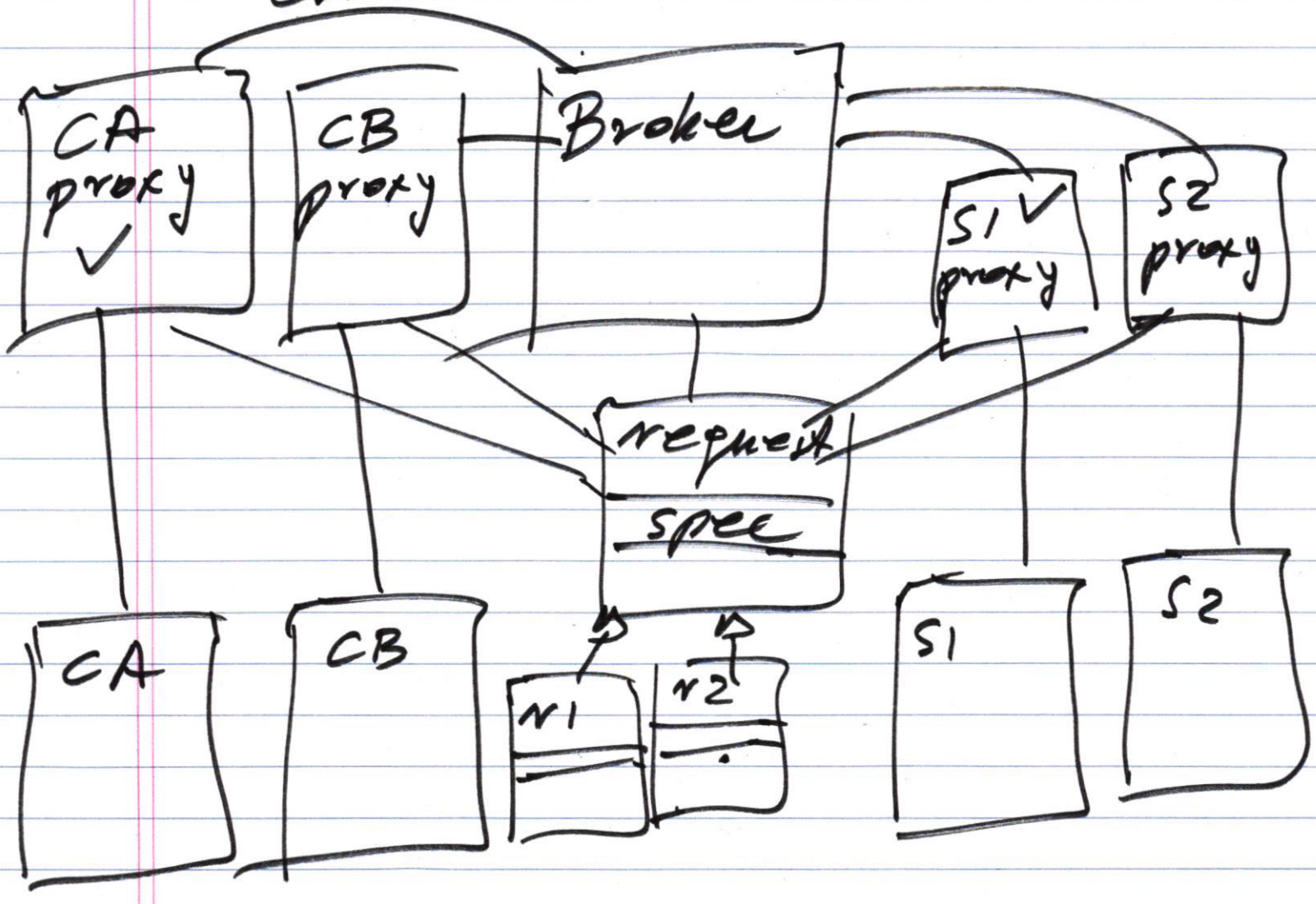
- StoreData
- Display Menu
- EjectCard


Input Processor
virtual event
VFSM
virtual action
OP
card(a,b)
pin(c)
card
CorrectPin
IncorrectPin
storeData
p = ta
bal = tb
DisplayMenu
Display Menu impl
EjectCard
Eject Card impl
temp
ta, tb
data
p =
bal =

Homework #2

Problem #1

Client-Broker-server



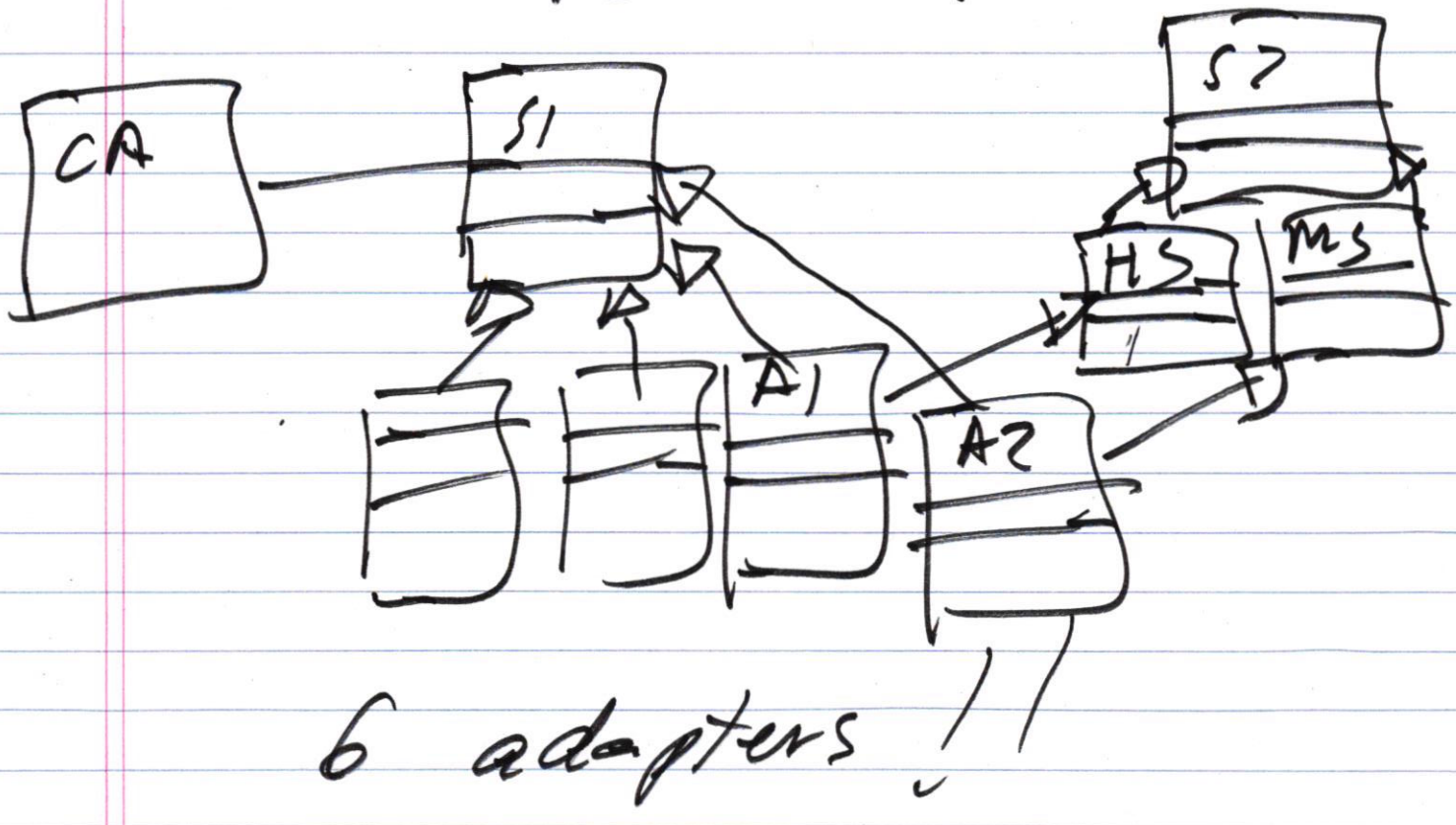
Problem #2

Adapter pattern

1. association-based solutions.

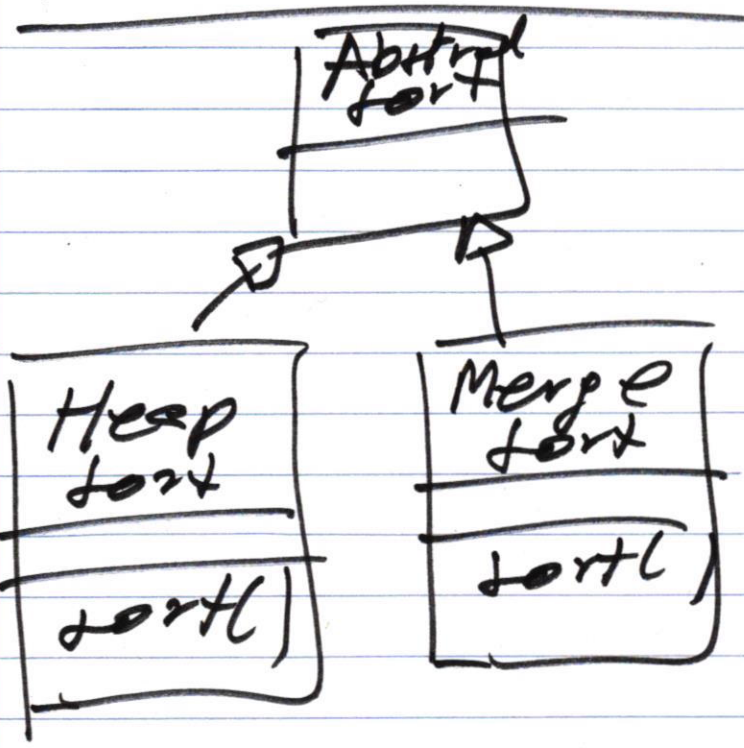
3 adapters

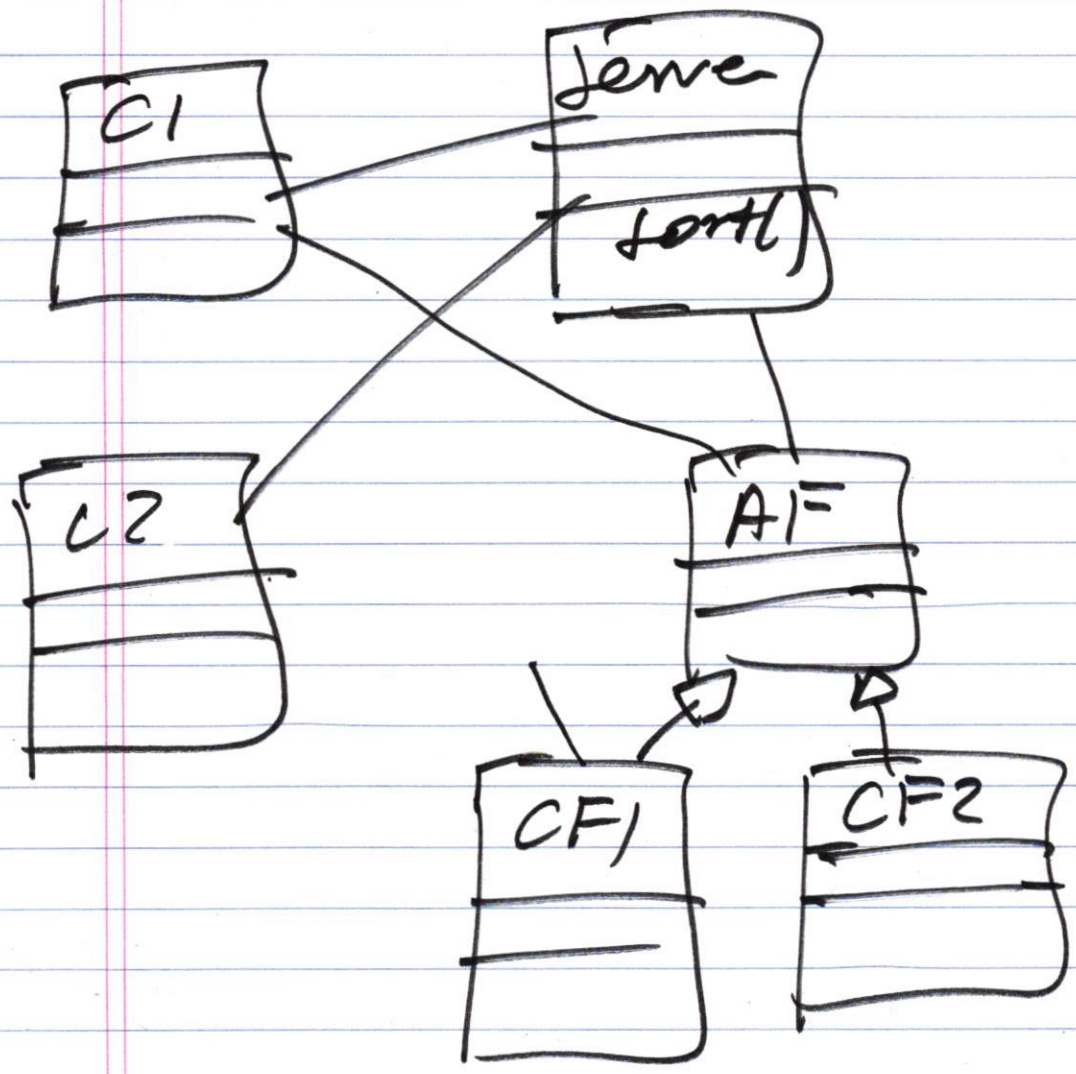
2. inheritance-based version.



6 adapters!

Problem #3




C1
Jerve
sort()
C2
AIF
CF1
CF2

HOMEWORK ASSIGNMENT #2

CS 586; Spring 2024
Due Date: **March 5, 2024**
Late homework 50% off
After **March 12,** 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 (35 points)
There exist two servers **S1** and **S2**. Both servers support the following services:

Services supported by server **S1**:
int SetPrice(string, float)
float GetPrice(string)
void BuyStock(string, int)
void SellStock(string, int)

Services supported by server **S2**:
int SetPrice(string, int)
float GetPrice(string)
void BuyStock(int, string)
void SellStock(string, int)
int SetPrice(string, float)

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

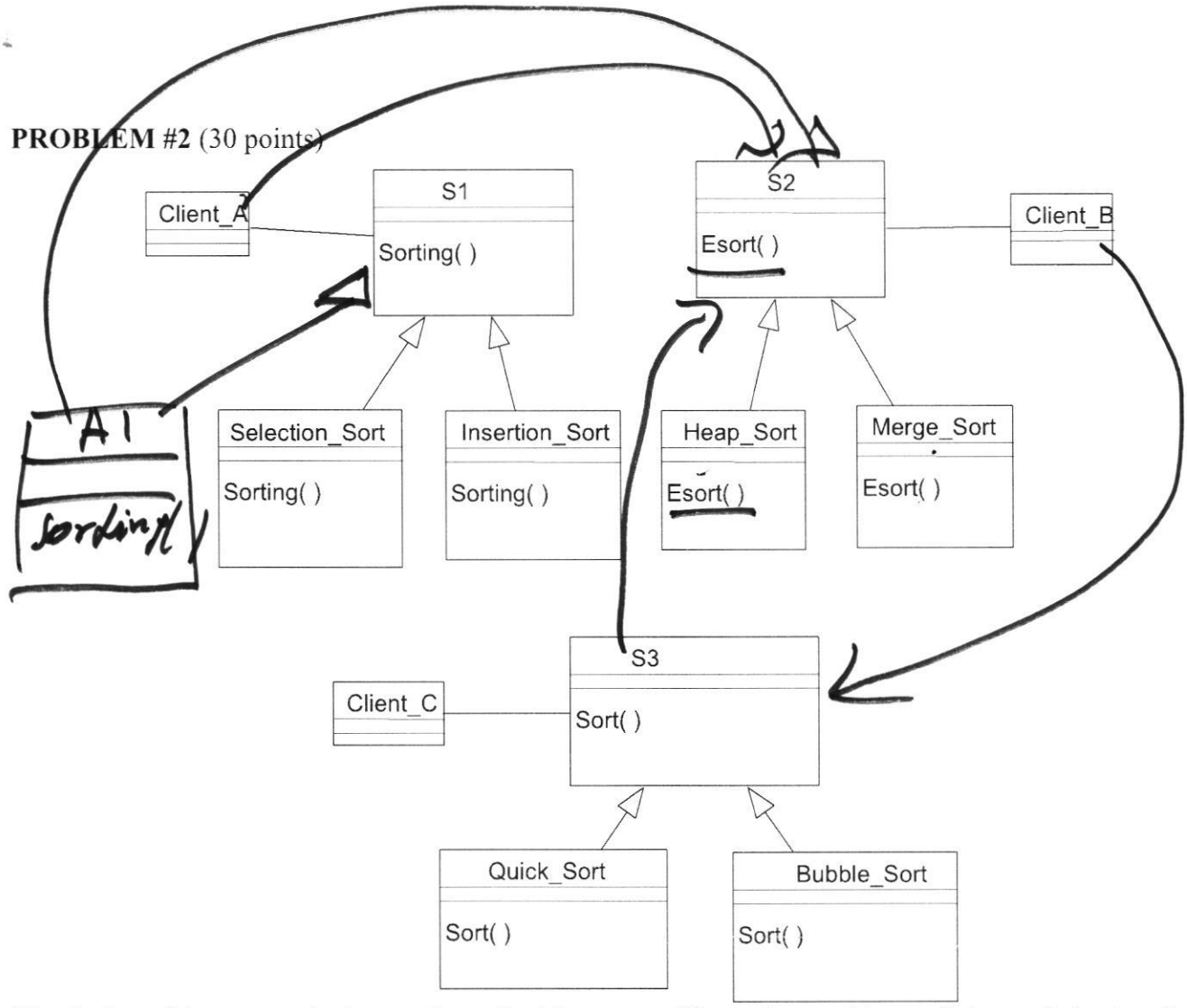
Client-A
int SetPrice(string, float)
float GetPrice(string)
void BuyStock(int, string)
void SellStock(string, int)

Client-B
int SetPrice(string, int)
float GetPrice(string)
void BuyStock(string, int)
void SellStock(string, int)

The client processes do not know the location (pointer) of 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 *S1*.
- Provide a sequence diagram to show how *Client-A* gets "*int SetPrice(string, float)*" service.

PROBLEM #2 (30 points)



The design of the system is shown above. In this system *Client_A* uses objects of classes *Selection_Sort* and *Insertion_Sort*, *Client_B* uses objects of classes *Heap_Sort* and *Merge_Sort*, and *Client_C* uses objects of classes *Quick_Sort* and *Bubble_Sort*.

Client_A would like to use objects, operation *Esort()*, of classes *Heap_Sort* and *Merge_Sort* by invoking operation *Sorting()*. *Client_B* would like to use objects, operations *Sort()*, of classes *Quick_Sort* and *Bubble_Sort* by invoking operation *Esort()*. In addition, *Client_C* would like to use objects, operation operation *Esort()*, of classes *Heap_Sort* and *Merge_Sort* by invoking operation *Sort()*.

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 *Heap_Sort* and *Merge_Sort* by invoking operation *Sorting*(),
(2) *Client_B* can use objects of classes *Quick_Sort* and *Bubble_Sort* by invoking operation *Esort()*, and
(3) *Client_C* can use objects of classes *Heap_Sort* and *Merge_Sort* by invoking operation *Sort()*.

Notice that none of the classes shown in the above class diagram should be modified.

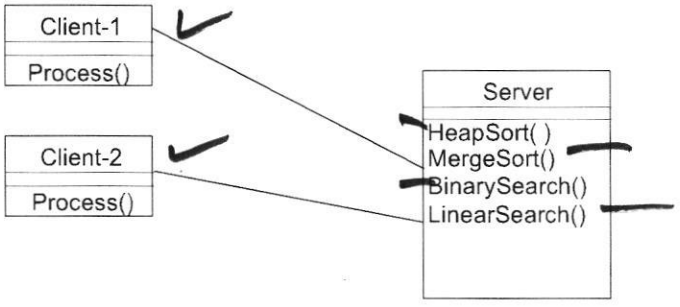
Provide two solutions that are based on:

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

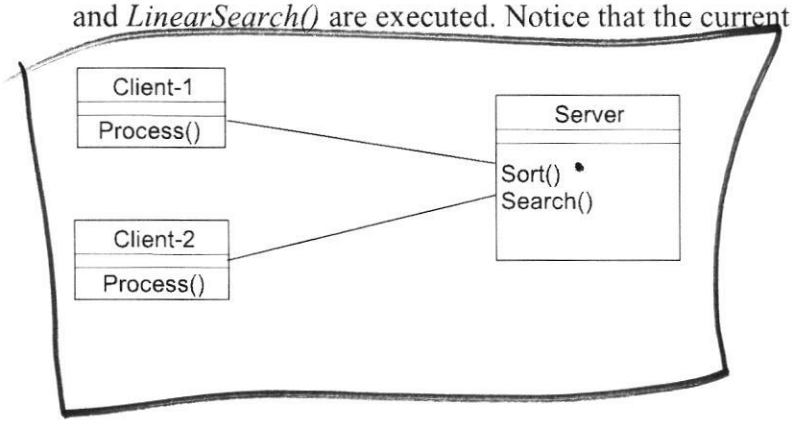
Provide a class diagram for each solution. You do not have to provide any description for classes/operations of the above class diagram (only new classes/operations should be described using **pseudo-code**).

PROBLEM #3 (35 points)

There exist two clients (*Client-1* and *Client-2*) and a *Server* class. The *Server* class supports the following operations: *HeapSort()*, *MergeSort()*, *BinarySearch()*, and *LinearSearch()*. *Client-1* invokes *HeapSort()* and *BinarySearch()* on the server. On the other hand, *Client-2* invokes *MergeSort()* and *LinearSearch()* on the server. The current design is shown below:



In a better design Clients should be shielded from different versions of sorting and searching. In the new design, as shown below, *Client-1* should invoke *Sort()* and *Search()*, where *HeapSort()* and *BinarySearch()* are executed. Similarly, *Client-2* should invoke *Sort()* and *Search()*, where *MergeSort()* and *LinearSearch()* are executed. Notice that the current design does not support this option.



Use the **strategy pattern** and the **abstract factory** design patterns to solve this problem. In your solution, the *Client* classes should be completely **de-coupled** from the issue of invoking appropriate versions of *Sort()* and *Search()*. Notice that in the design new classes/operations should be introduced according to these patterns.

- Provide the class diagram and 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. In addition, indicate which classes/operations are part of the strategy pattern and which classes/operations are part of the abstract factory pattern.

- Provide a sequence diagram to show how *Client-1* gets services *HeapSort()* and *BinarySearch()* by invoking *Sort()* and *Search()* on the *Server*.