* Homework #3 is posted
  today is deadline.

* project part #2 is posted.

* Exam #3
  Monday, December 8
  5:00 - 7:00 pm
  closed books and notes
  exam
  The coverage is posted
  comprehensive exam!

COVERAGE FOR EXAM #3

CS 586; Fall 2025

Exam #3 will be held on **Monday, December 8, 2025,** between **5:00-7:00 p.m.**
Location: 152 PS (Pritzker Science Center)
The exam is a **CLOSED books and notes** exam.

Coverage for the exam:

- OO design patterns: item description, whole-part, observer, state, proxy, adapter, strategy, and abstract factory patterns. [Textbook: Sections 3.1, 3.2; Section 3.4 (pp. 263-275); Section 3.6 (pp.339-343); Handout #1, class notes]
- Interactive systems. Model-View-Controller architectural pattern [Textbook: Section 2.4, pp. 123-143]
- Client-Server Architecture
  - Client-Dispatcher-Server [Section 3.6: pp. 323-337]
  - Client-Broker-Server Architecture [Textbook: Section 2.3; pp. 99-122]
- Layered architecture [Textbook: Section 2.2; pp. 31-51]
- Pipe and Filter Architecture [Textbook: Section 2.2; pp. 53-70]
- Adaptable Systems:
  - Micro-kernel architectural pattern [Textbook: Section 2.5, pp. 169-192]
- Fault-tolerant architecture [Handout #2]
  - N-version architecture
  - Recovery-Block architecture
  - N-Self Checking architecture

Sources:

- Textbook: F. Buschmann, et. al., Pattern-oriented software architecture, vol. I, John Wiley & Sons.
- Class notes
- Handouts

PART #2: PROJECT DESIGN, IMPLEMENTATION, and REPORT

CS 586; Fall 2025

Final Project Deadline: **Friday, December 5, 2025**
Late submissions: 50% off
After **December 9,** the final project will not be accepted.

Submission: The project must be submitted on Canvas. The hardcopy submissions will not be accepted.

This is an **individual** project, not a team project. Identical or similar submissions will be penalized.

DESIGN and IMPLEMENTATION

The goal of the second part of the project is to design two *Gas Pump* components using the Model-Driven Architecture (MDA) and then implement these *Gas Pump* components based on this design using the OO programming language. This OO-oriented design should be based on the MDA-EFSM for both *Gas Pump* components that was identified in the first part of the project. You may use your own MDA-EFSM (assuming that it was correct), or you can use the posted sample MDA-EFSM. In your design, you **MUST** use the following OO design patterns:

- state pattern
- strategy pattern
- abstract factory pattern

In the design, you need to provide the class diagram, in which the coupling between components should be minimized and the cohesion of components should be maximized (components with high cohesion and low coupling between components). In addition, two sequence diagrams should be provided as described on the next page (Section 4 of the report).

After the design is completed, you need to implement the *Gas Pump* components based on your design using the OO programming language. In addition, the driver for the project to execute and test the correctness of the design and its implementation for the *Gas Pump* components must be implemented.

Outline of the Report & Deliverables

I: REPORT

The report **must** be submitted as one PDF file (otherwise, a **10% penalty will be applied).**

1. MDA-EFSM model for the *Gas Pump* components
   a. A list of meta events for the MDA-EFSM
   b. A list of meta actions for the MDA-EFSM with their descriptions
   c. A state diagram of the MDA-EFSM
   d. Pseudo-code of all operations of the Input Processors of Gas Pumps: *GP-1* and *GP-2*

2. Class diagram(s) of the MDA of the *Gas Pump* components. In your design, you **MUST** use the following OO design patterns:
   a. State pattern
   b. Strategy pattern
   c. Abstract factory pattern

3. For each class in the class diagram(s), you should:
   a. Describe the purpose of the class, i.e., responsibilities.
   b. Describe the responsibility of each operation supported by each class.

4. Dynamics. Provide sequence diagrams for two Scenarios:
   a. Scenario-I should show how one liter of gas is dispensed in *GasPump-1*, i.e., the following sequence of operations is issued: *Activate(4.1), Start(), PayCash(5.2), StartPump(), PumpLiter(), PumpLiter()*

   b. Scenario-II should show how one gallon of Regular gas is dispensed in *GasPump-2*, i.e., the following sequence of operations is issued: *Activate(4, 7), Start(), PayDebit(123), Pin(124), Pin(123), Regular(), StartPump(), PumpGallon(), FullTank()*

II: Well-documented (commented) source code

In the source code, you should clearly indicate/highlight which parts of the source code are responsible for the implementation of the three required design patterns (**if this is not clearly indicated in the source code, 20 points will be deducted**).

- state pattern
- strategy pattern
- abstract factory pattern.

The source code must be submitted on Canvas. Note that the source code may be compiled during the grading and then executed. If the source code is not provided, **15 POINTS** will be deducted.

III: Project executables

The project executable(s) of the *Gas Pump* components, with detailed instructions explaining the execution of the program, must be prepared and made available for grading. The project executable should be submitted on Canvas. If the executable is not provided (or not easily available), **20 POINTS** will be automatically deducted from the project grade.

Project Part#2

Design and implement GP components based on MDA-EFSM from part#1.

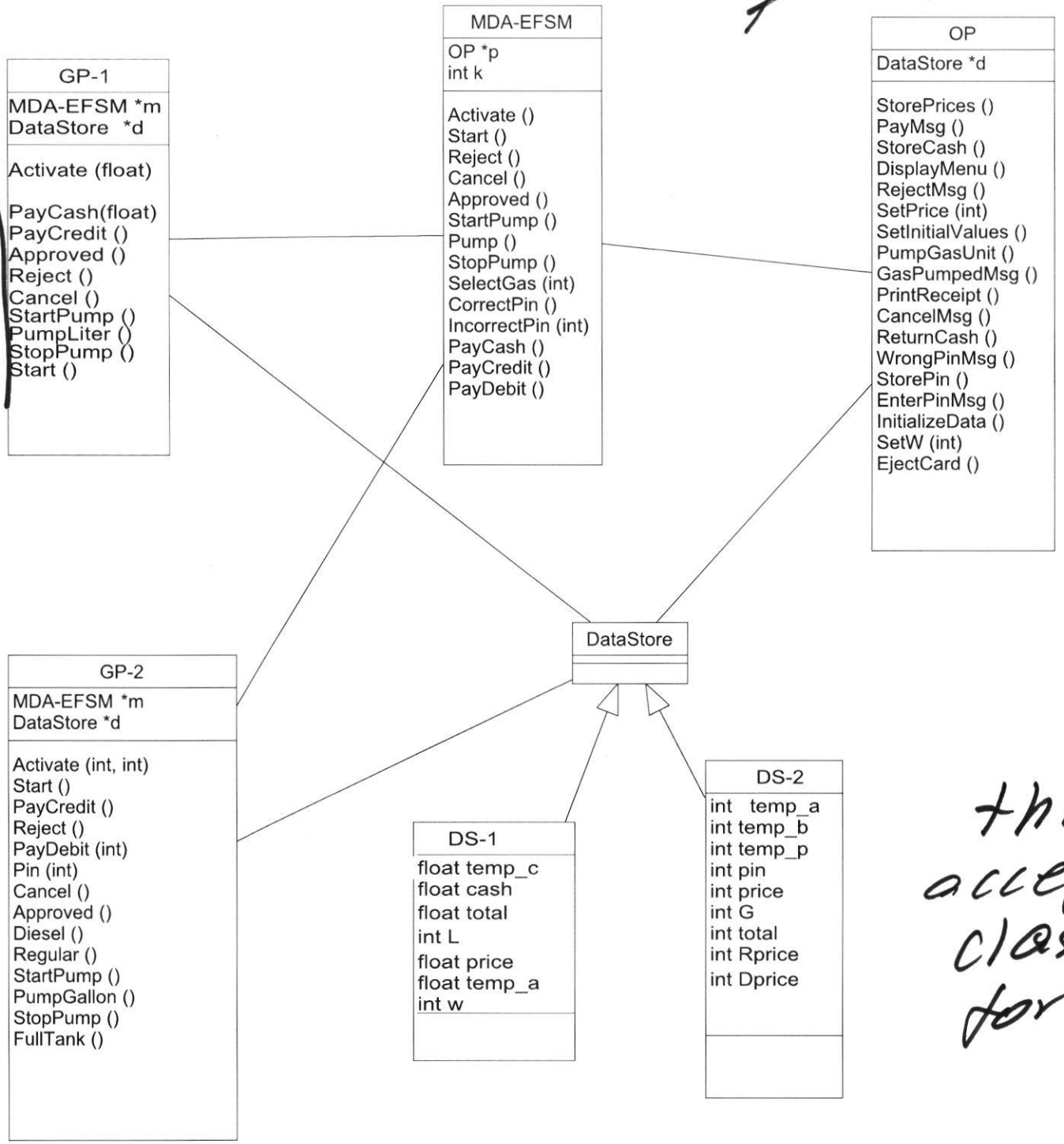
---

if your MDA-EFSM was correct, you can use it for part #2

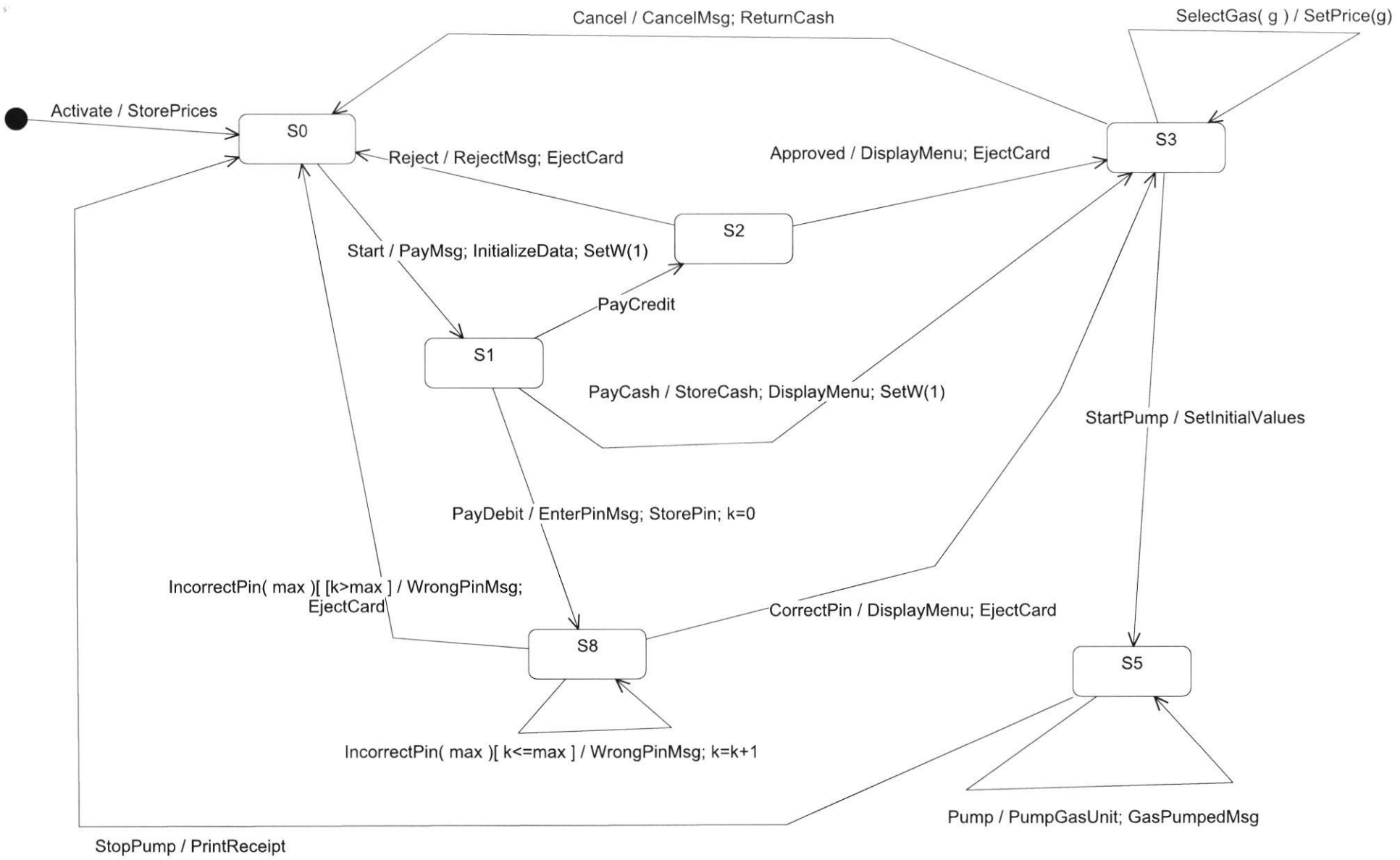
---

if your MDA-EFSM was ~~is~~ not correct, for part #2 you should use posted MDA-EFSM

Posted MDA-EFSM.



this is not acceptable class diagram for part #2.



MDA-EFSM for Gas Pumps

MDA-EFSM Events:

Activate()
Start()
PayCredit()
PayCash()
PayDebit()
Reject()
Cancel()
Approved()
StartPump()
Pump()
StopPump()
SelectGas(int g)
CorrectPin()
IncorrectPin(int max)

MDA-EFSM Actions:

StorePrices	// stores price(s) for the gas from the temporary data store
PayMsg	// displays a type of payment method
StoreCash	// stores cash from the temporary data store
DisplayMenu	// display a menu with a list of selections
RejectMsg	// displays credit card not approved message
SetPrice(int g)	// set the price for the gas identified by *g* identifier as in SelectGas(int g)
SetInitialValues	// set *G* (or *L*) and *total* to 0;
PumpGasUnit	// disposes unit of gas and counts # of units disposed
GasPumpedMsg	// displays the amount of disposed gas
PrintReceipt	// print a receipt
CancelMsg	// displays a cancellation message
ReturnCash	// returns the remaining cash
WrongPinMsg	// displays incorrect pin message
StorePin	// stores the pin from the temporary data store
EnterPinMsg	// displays a message to enter pin
InitializeData	// set the value of *price* to 0 for GP-2; do nothing for GP-1
EjectCard()	// card is ejected
SetW(int w)	// set value for cash flag

Operations of the Input Processor (GasPump-1)

```
Activate(float a) {
        if (a>0) {
            d->temp_a=a;
            m->Activate()
        }
}

Start() {
        m->Start();
}
PayCash(float c) {
        if (c>0) {
            d->temp_c=c;
            m->PayCash()
        }
}

PayCredit() {
        m->PayCredit();
}

Reject() {
        m->Reject();
}

Approved() {
        m-> Approved();
}
Cancel() {
        m->Cancel();
}

StartPump() {
        m->StartPump();
}
```

```
PumpLiter() {
if (d->w==1) m->Pump()
else if (d->cash>0)&&(d->cash < d->price*(d->L+1))
            m->StopPump();
else m->Pump()
}

StopPump() {
        m->StopPump();
}
```

Notice:
cash: contains the value of cash deposited
price: contains the price of the selected gas
L: contains the number of liters already pumped
w: cash flag (cash: w=0; otherwise: w=1)
cash, *L*, *price, w* are in the data store
m: is a pointer to the MDA-EFSM object
d: is a pointer to the Data Store object

Operations of the Input Processor (GasPump-2)

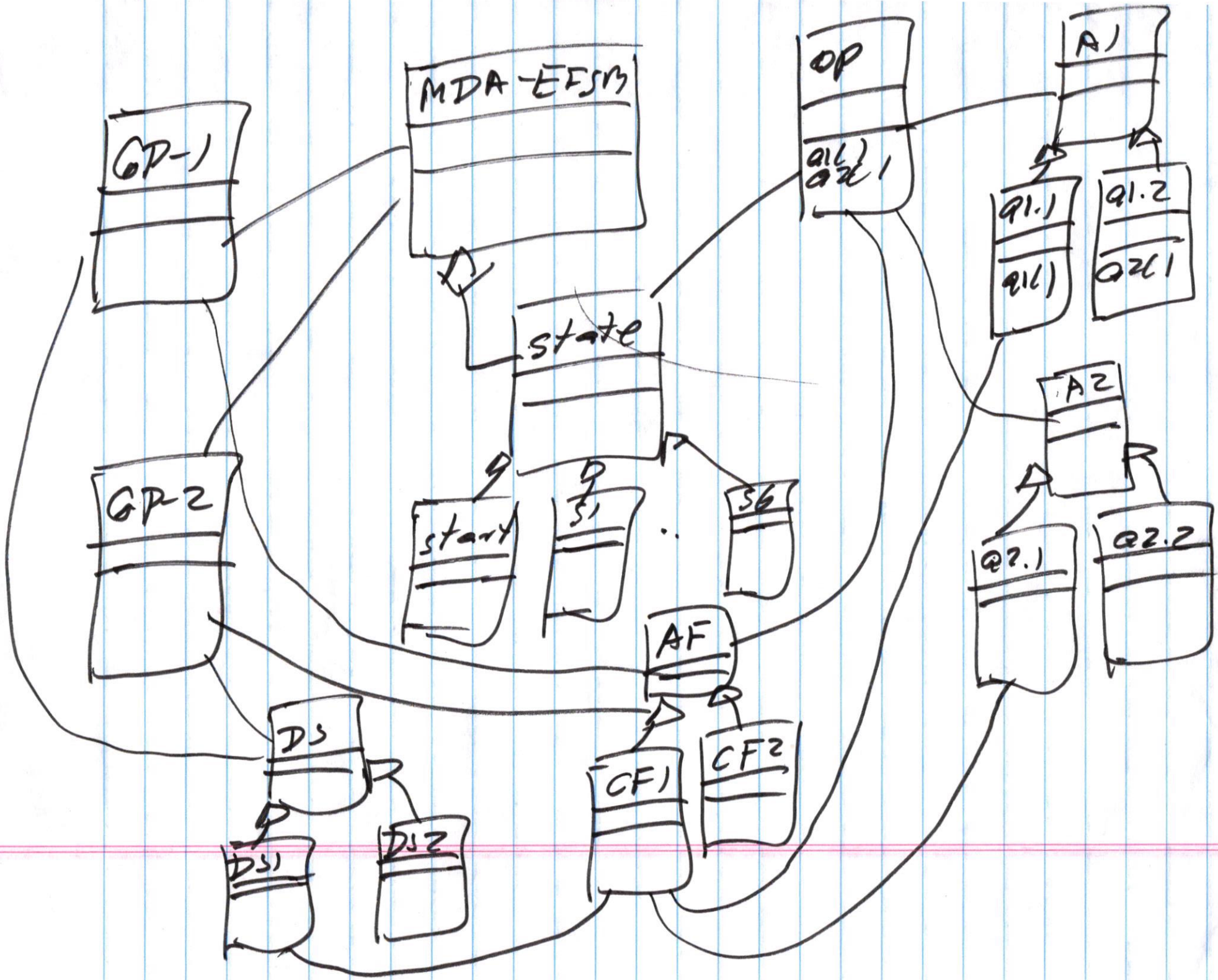
```
Activate(int a, int b) {
        if ((a>0)&&(b>0)) {
            d->temp_a=a;
            d->temp_b=b;
            m->Activate()
        }
}

Start() {
        m->Start();
}

PayCredit() {
        m->PayCredit();
}

Reject() {
        m->Reject();
}

PayDebit(int p) {
        d->temp_p=p;
        m->PayDebit();
}

Pin(int x) {
        if (d->pin==x) m->CorrectPin()
        else m->InCorrectPin(1);
}

Cancel() {
        m->Cancel();
}

Approved() {
        m->Approved();
}

Diesel() {
        m->SelectGas(2)
}

Regular() {
        m->SelectGas(1)
}

StartPump() {
        if (d->price>0) m->StartPump();
}

PumpGallon() {
        m->Pump();

StopPump() {
        m->StopPump();
}

FullTank() {
        m->StopPump();
}
```

Notice:
pin: contains the pin in the data store
m: is a pointer to the MDA-EFSM object
d: is a pointer to the Data Store object
SelectGas(g): Regular: g=1; Diesel: g=2

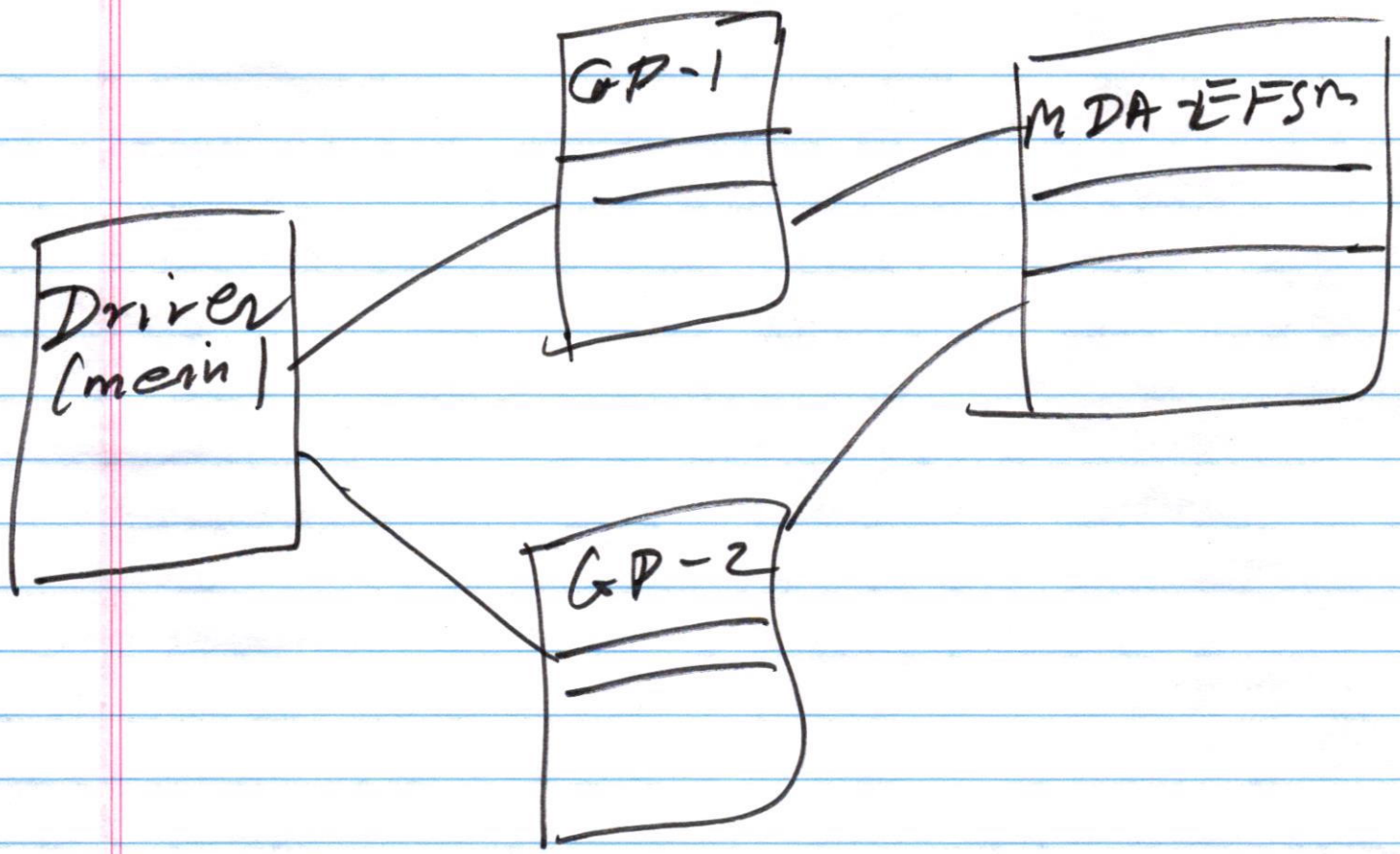
Part #2

In the design you MUST incorporate 3 patterns!

1. state pattern
2. strategy pattern
3. abstract factory pattern
   to initialize selected GP components for execution


MDA-EFSM
GP-1
GP-2
OP
A1
A2
state
start
S1
S6
AF
CF1
CF2
DS
DS1
DS2
Q1.1
Q1.2
Q2.1
Q2.2

Implementation of the design in OO language.



We need to implement "Driver"

Driver

1. Select GP for execution
2. execute selected GP
   We should be able to invoke any operation of selected GP (together with input data) at any time.
3. Test the correctness of the design

partial driver in C++

```
main () { // partial driver
….
GasPump_1 gp1;

      cout<< "                    GasPump-1" << endl;
      cout<< "               MENU of Operations" << endl;
      cout<< "          0. Activate(float)" << endl;
      cout<< "          1. Start()" << endl;
      cout<< "          2. PayCredit" << endl;
      cout<< "          3. Reject()" << endl;
      cout<< "          4. Cancel()" << endl;
      cout<< "          5. Approved()" << endl;
      cout<< "          6. PayCash(float)" << endl;
      cout<< "          7. StartPump()" << endl;
      cout<< "          8. PumpLiter()" << endl;
      cout<< "          9. StopPump()" << endl;
      cout<< "          q. Quit the program" << endl;

      cout<< "  Please make a note of these operations" << endl;
      cout<< "           GasPump-1 Execution" << endl;
          ch='1';
  while (ch !='q') {
      cout<< "  Select Operation: "<<endl;

cout<<"0-Activate,1-Start,2-PayCredit,3-Reject,4-Cancel,5-Approved,6-PayCash,7-St
artPump, 8-PumpLiter, 9-StopPump, q-quit"<<endl;
      ch=getch();
      switch (ch) {
        case '0': { //Activate()
                        cout<<"  Operation:  Activate(float a)"<<endl;
                        cout<<"  Enter value of the parameter a:"<<endl;
                        cin>>a;
                        gp1.Activate(a);
                        break;
                  };

        case '1': { //Start
                        cout<<"  Operation:  Start()"<<endl;
                        gp1.Start();
                        break;
                    };

        case '2': { //PayCredit
                        cout<<"  Operation:  PayCredit()"<<endl;
                        gp1.PayCredit();
                        break;
                    };

        case '3': { //Reject
                        cout<<"  Operation:  Reject()"<<endl;
```

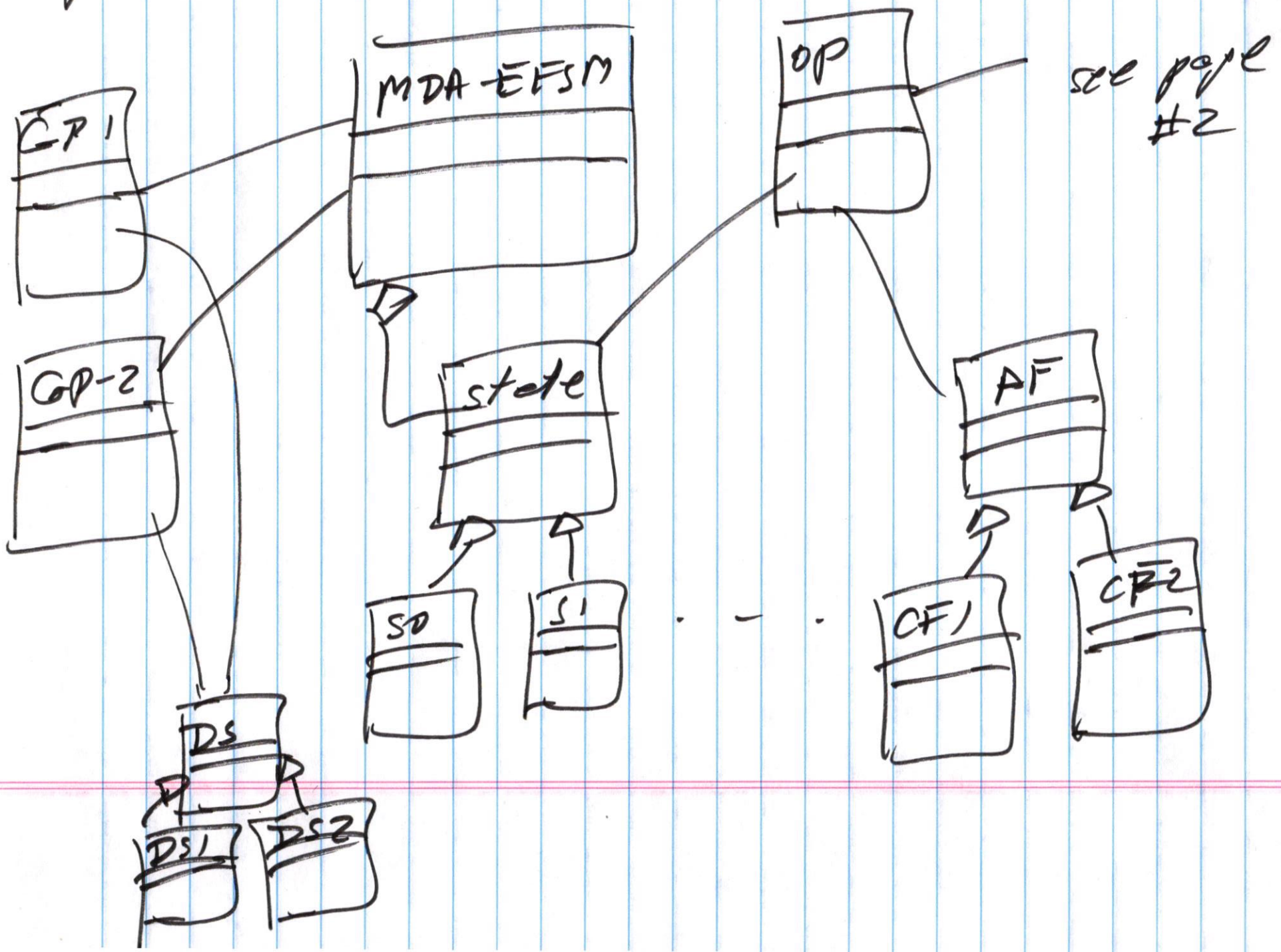
```
                    gp1.Reject();
                    break;
                };

    case '4': { //Cancel
                    cout<<"  Operation:  Cancel()"<<endl;
                    gp1.Cancel();
                    break;
                };

    case '5': { //Approved
                    cout<<"  Operation:  Approved()"<<endl;
                    gp1.Approved();
                    break;
                };

    case '6': { //PayCash
                    cout<<"  Operation:  PayCash(float c)"<<endl;
                    cout<<"  Enter value of the parameter c:"<<endl;
                    cin>>c;
                    gp1.PayCash(c);
                    break;
                };

    case '7': { //StartPump
                    cout<<"  Operation:  StartPump()"<<endl;
                    gp1.StartPump();
                    break;
                };


    case '8': { //PumpLiter
                    cout<<"  Operation:  PumpLiter()"<<endl;
                    gp1.PumpLiter();
                    break;
                };

    case '9': { //StopPump
                    cout<<"  Operation:  StopPump()"<<endl;
                    gp1.StopPump();
                    break;
                };
  };// endswitch

}; //endwhile
```

Partial Driver for Fare.

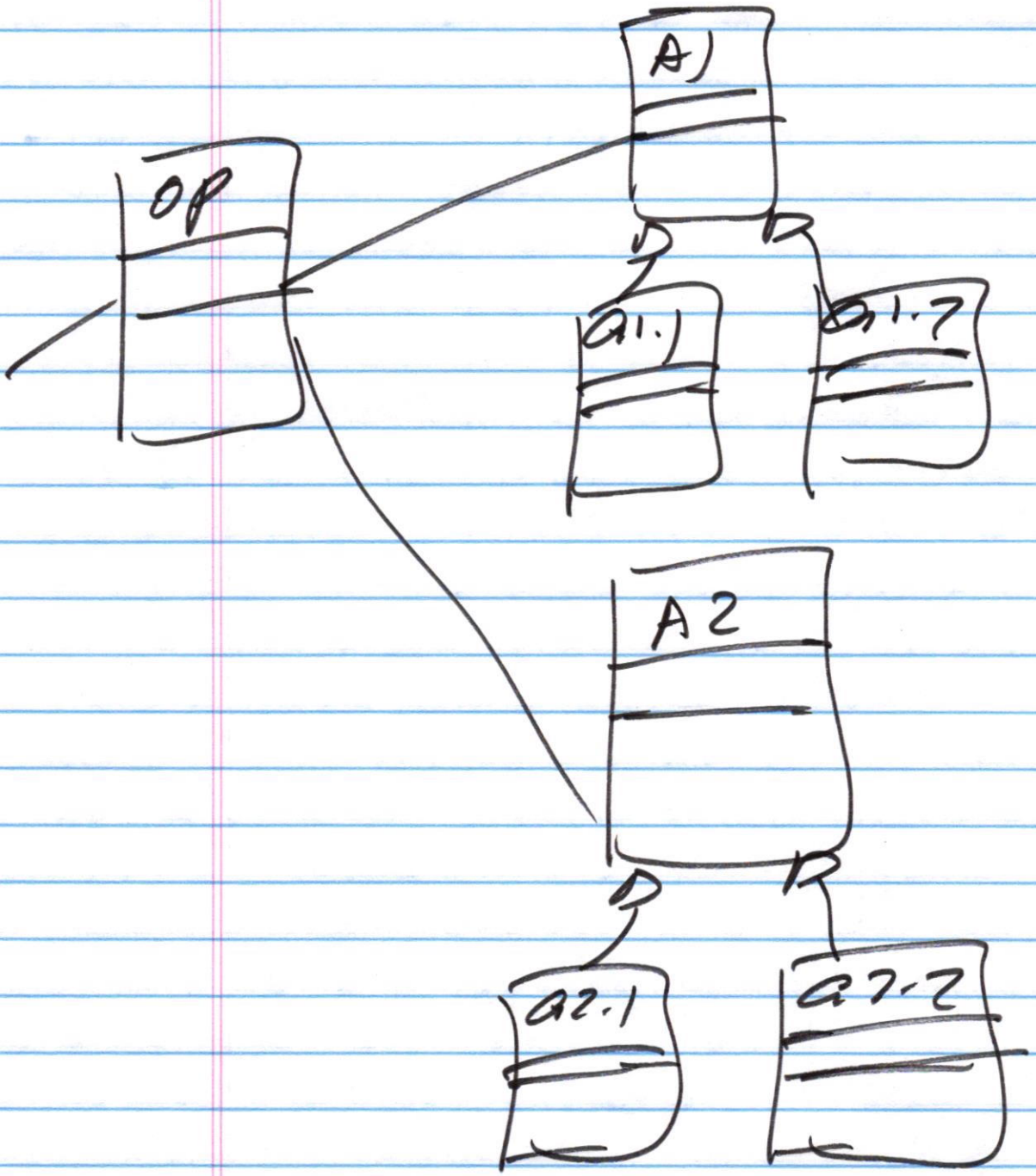
```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        GasPump_1 gp1 = new GasPump_1();

        System.out.println("                          GasPump-1");
        System.out.println("                    MENU of Operations");
        System.out.println("          0. Activate(float)");
        System.out.println("          1. Start()");
        System.out.println("          2. PayCredit");
        System.out.println("          3. Reject()");
        System.out.println("          4. Cancel()");
        System.out.println("          5. Approved()");
        System.out.println("          6. PayCash(float)");
        System.out.println("          7. StartPump()");
        System.out.println("          8. Pump()");
        System.out.println("          9. StopPump()");
        System.out.println("          q. Quit the program");
        System.out.println("  Please make a note of these operations");
        System.out.println("           GasPump-1 Execution");

        char ch = '1';
        while (ch != 'q') {
            System.out.println("Select Operation:");
            System.out.println("0-Activate,1-Start,2-PayCredit,3-Reject,4-Cancel,5-
Approved,6-PayCash,7-StartPump,8-PumpLiter,9-StopPump,q-quit");
            ch = sc.next().charAt(0);

            switch (ch) {
                case '0': {
                    System.out.println("Operation: Activate(float a)");
                    System.out.print("Enter value of the parameter a: ");
                    float a = sc.nextFloat();
                    gp1.Activate(a);
                    break;
                }
                case '1': {
                    System.out.println("Operation: Start()");
                    gp1.Start();
                    break;
                }
           case '2': {
                    System.out.println("Operation: PayCredit()");
                    gp1.PayCredit();
                    break;
                }
                case '3': {
                    System.out.println("Operation: Reject()");
                    gp1.Reject();
```

```
                    break;
                }
                case '4': {
                    System.out.println("Operation: Cancel()");
                    gp1.Cancel();
                    break;
                }
                case '5': {
                    System.out.println("Operation: Approved()");
                    gp1.Approved();
                    break;
                }
                case '6': {
                    System.out.println("Operation: PayCash(float c)");
                    System.out.print("Enter value of the parameter c: ");
                    float c = sc.nextFloat();
                    gp1.PayCash(c);
                    break;
                }
                case '7': {
                    System.out.println("Operation: StartPump()");
                    gp1.StartPump();
                    break;
                }
                case '8': {
                    System.out.println("Operation: PumpLiter()");
                    gp1.PumpLiter();
                    break;
                }
                case '9': {
                    System.out.println("Operation: StopPump()");
                    gp1.StopPump();
                    break;
                }
                case 'q': {
                    System.out.println("Quitting program...");
                    break;
                }
                default: {
                    System.out.println("Invalid option. Try again.");
                }
            }
        }

        sc.close();
    }
}
```

split class diagram into several pages.

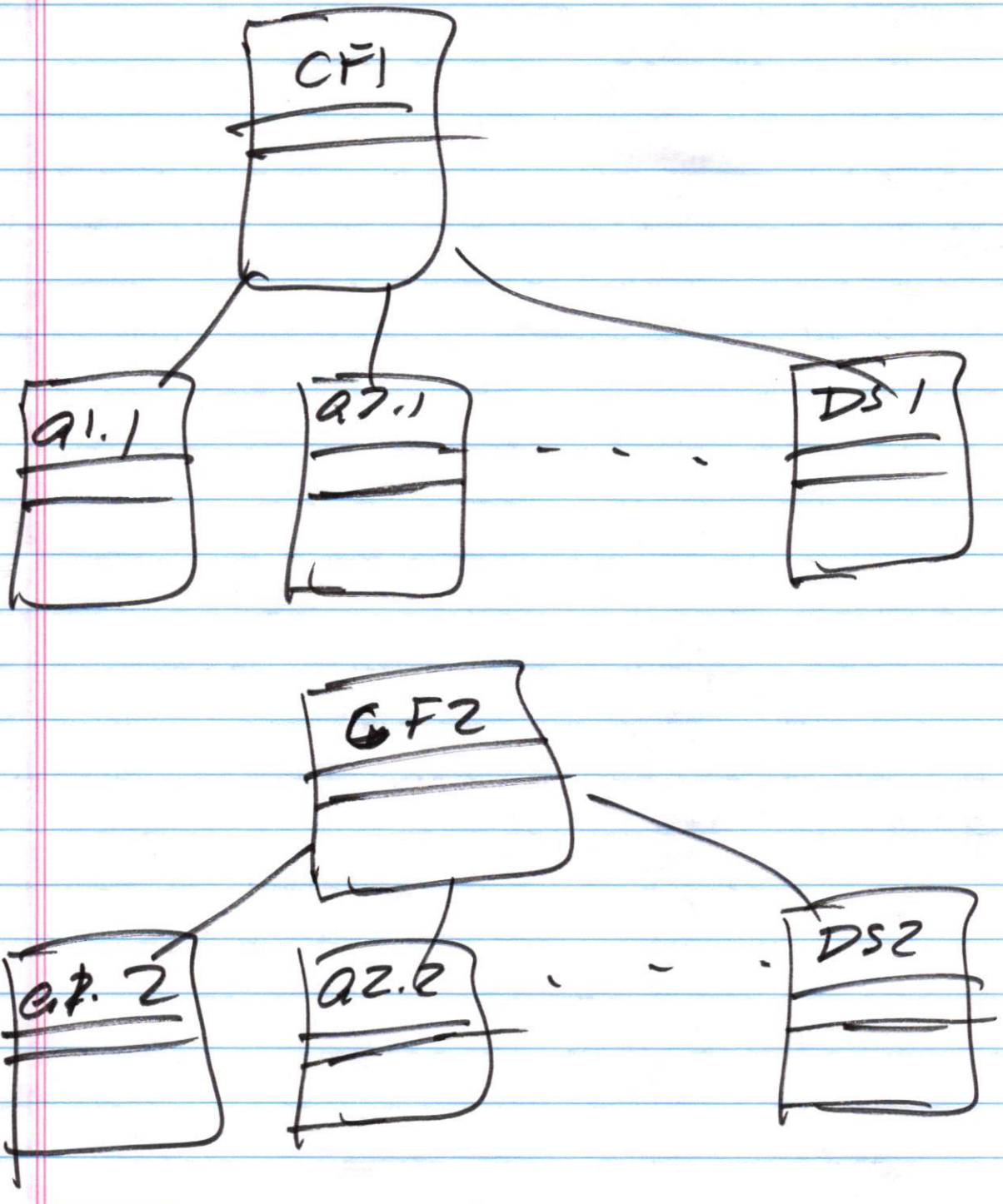


page #2

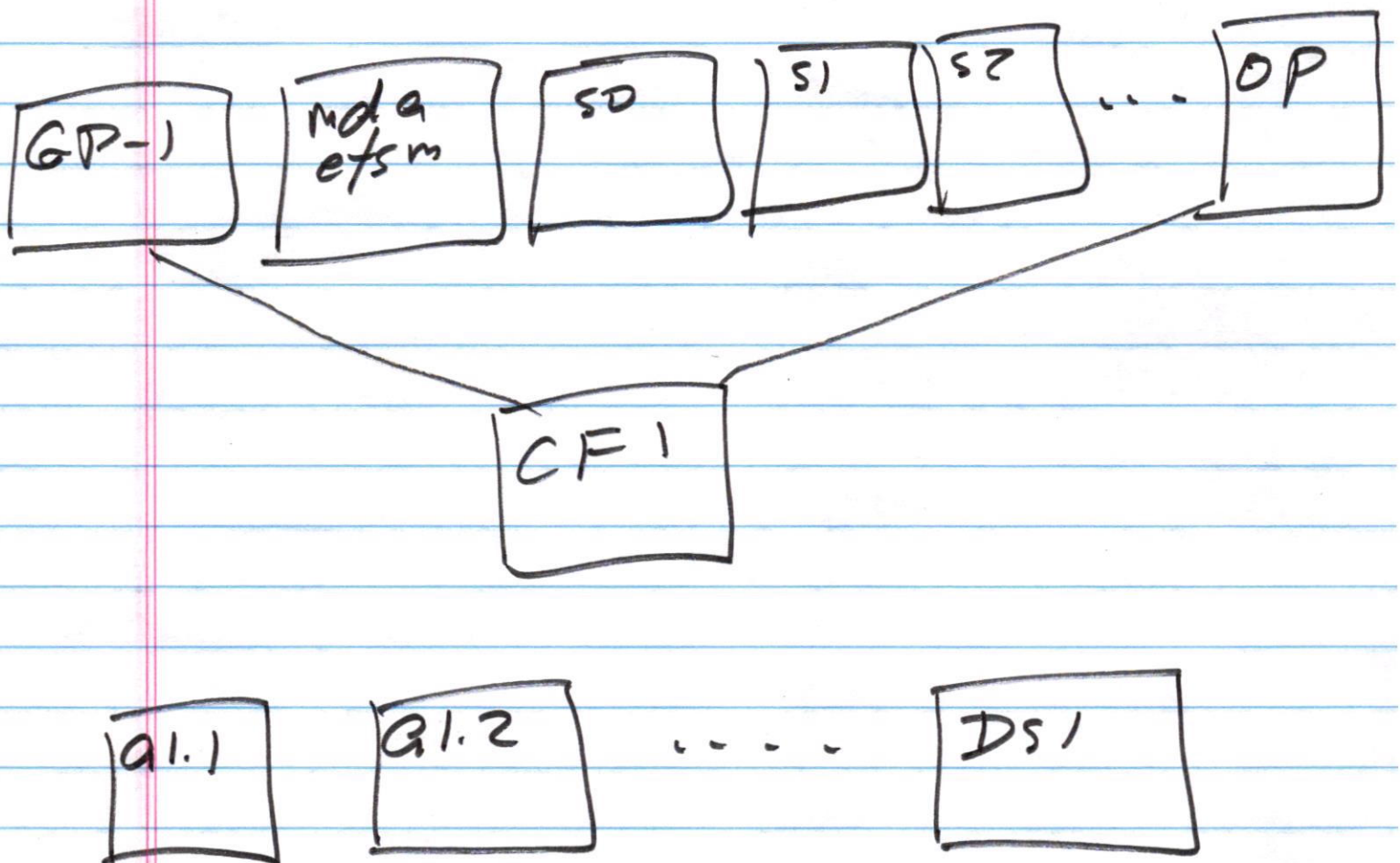


see page #3

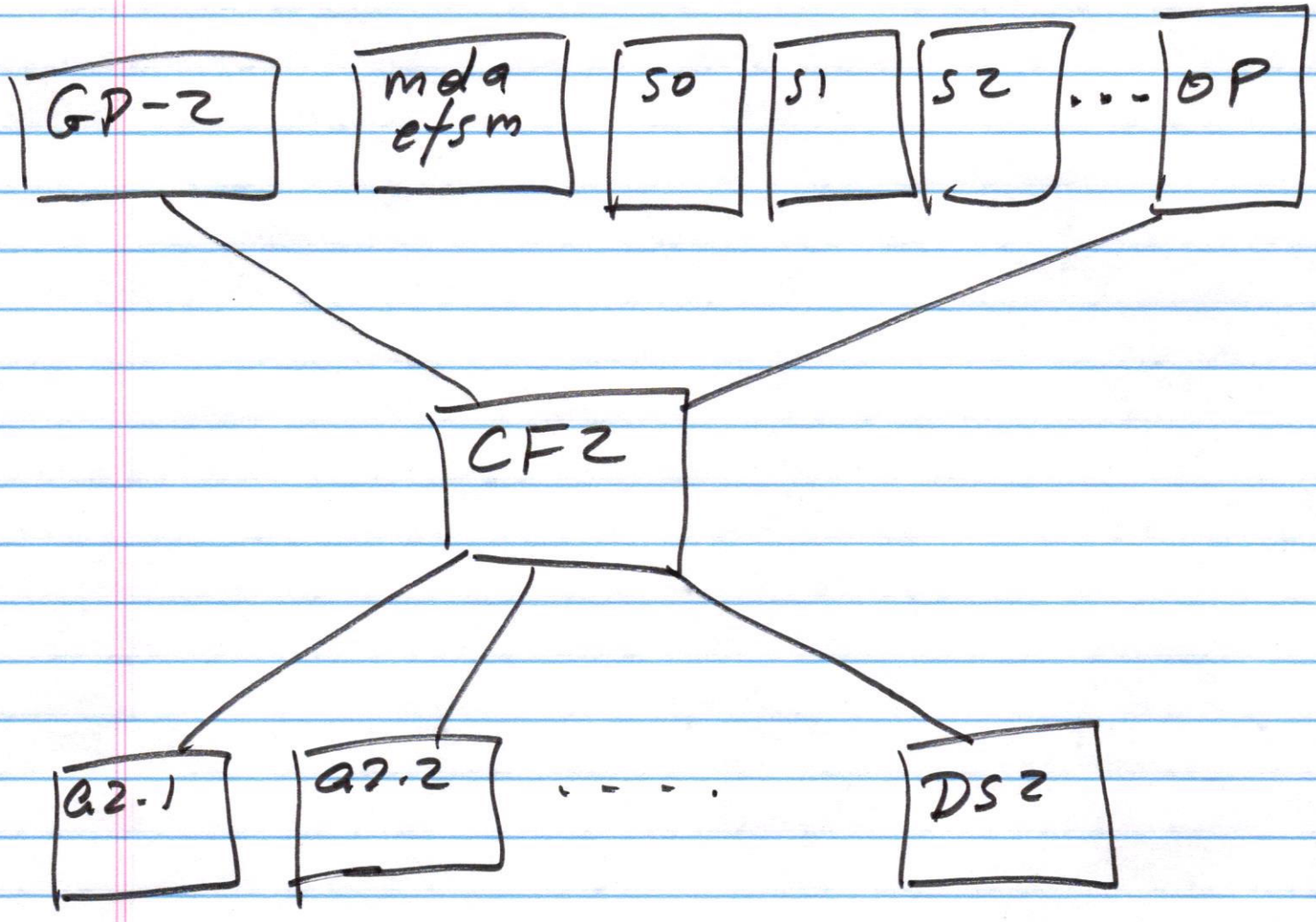
page #7



GP-1 objects



GP-2 objects



Fault-tolerant architecture

1. N-version architecture
2. Recovery block —"—
3. N-self checking —"—

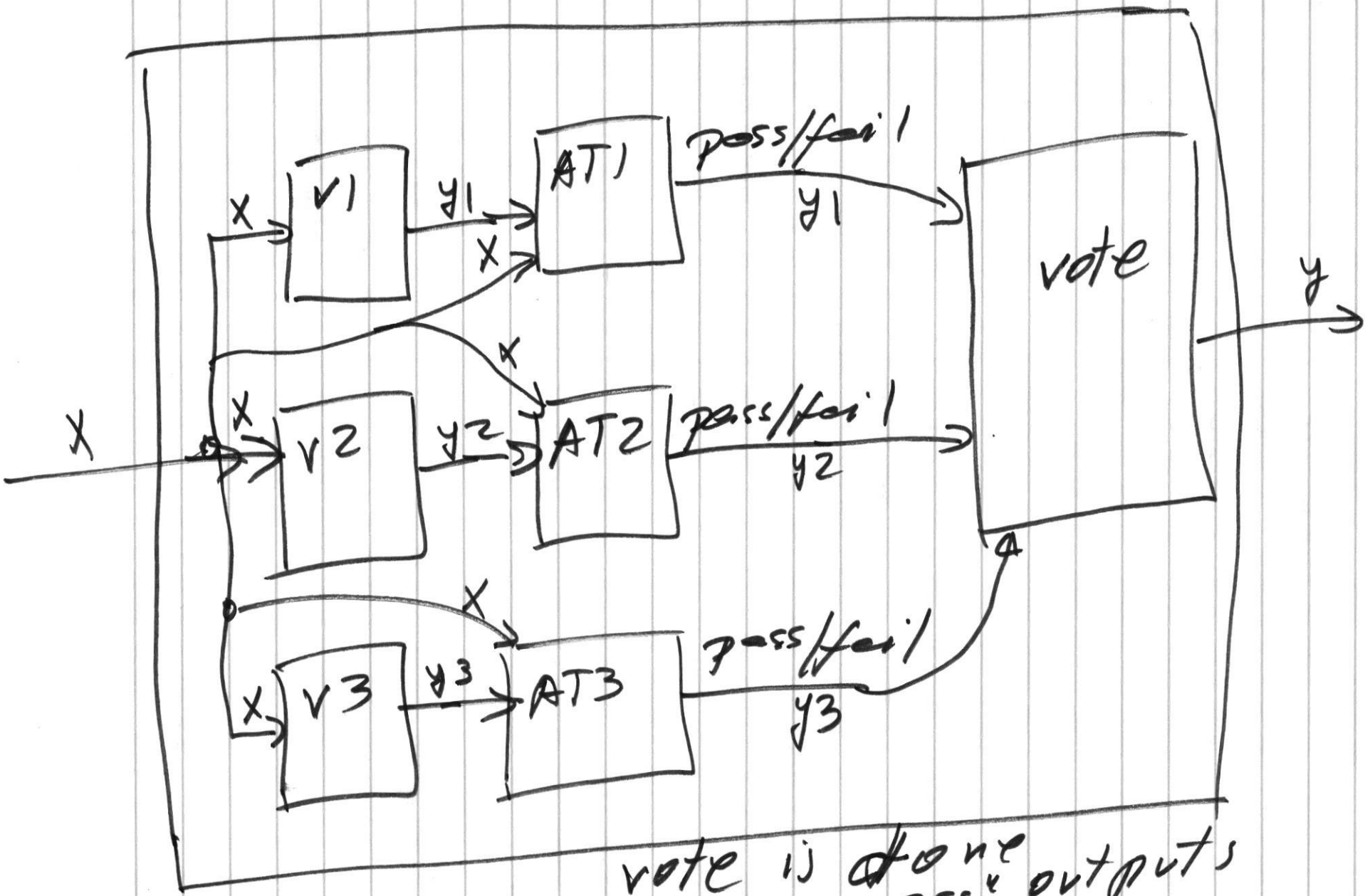
3. N-self checking architecture

combination of

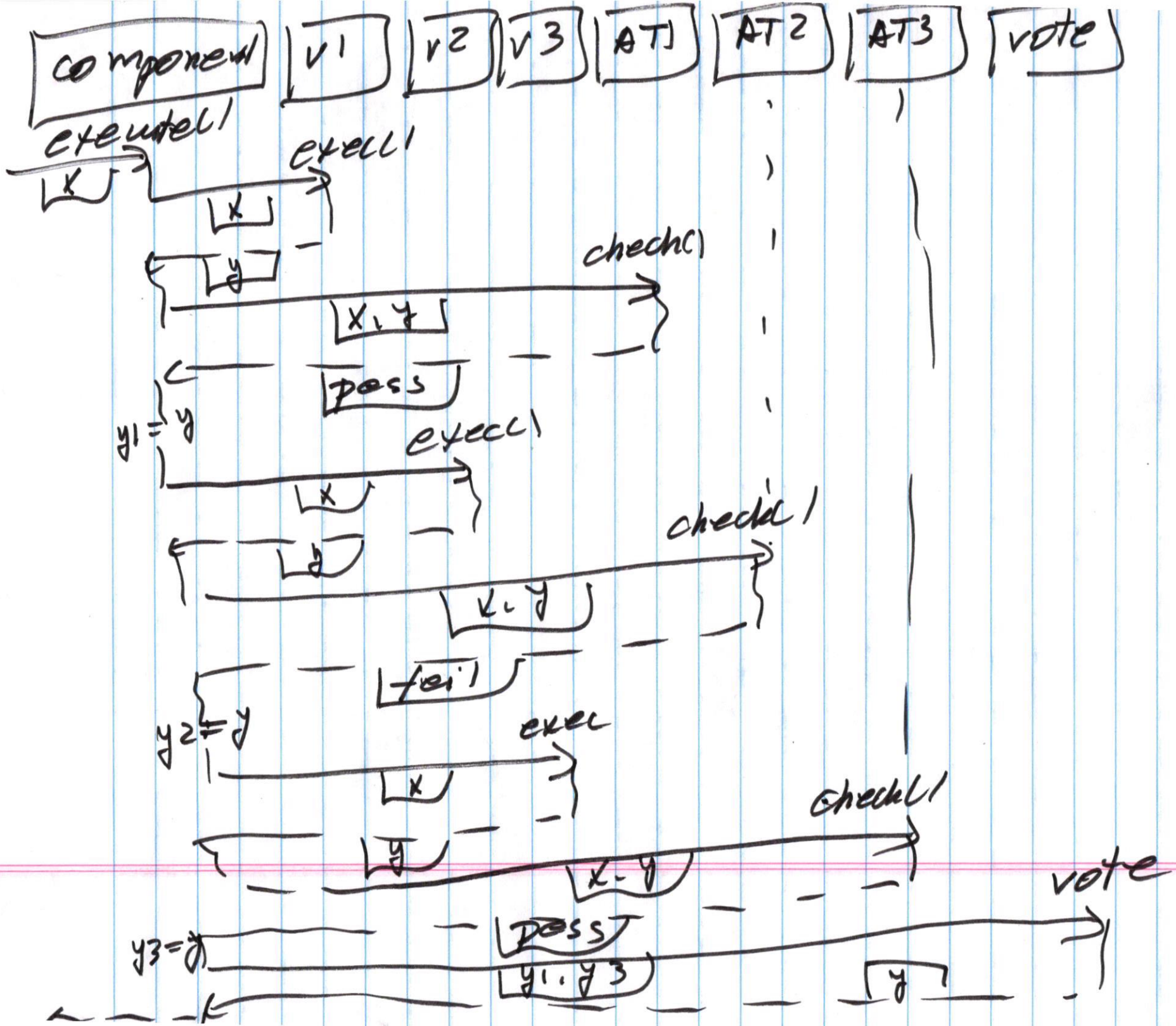
1. N-version architecture

+

2. Recovery block architecture



vote is done
for "pass" outputs
"failed" outputs
should be ignored


coponew
V1
V2
V3
AT1
AT2
AT3
vote
execute()
x
exec()
x
y
check()
x, y
pass
y1 = y
exec()
x
y
check()
x, y
fail
y2 = y
exec
x
y
check()
x, y
pass
vote
y3 = y
y1, y3
y

Adv

more reliable than
N-version architecture
Recovery Block —"—

Disdv

* very expensive.

strict layered architecture

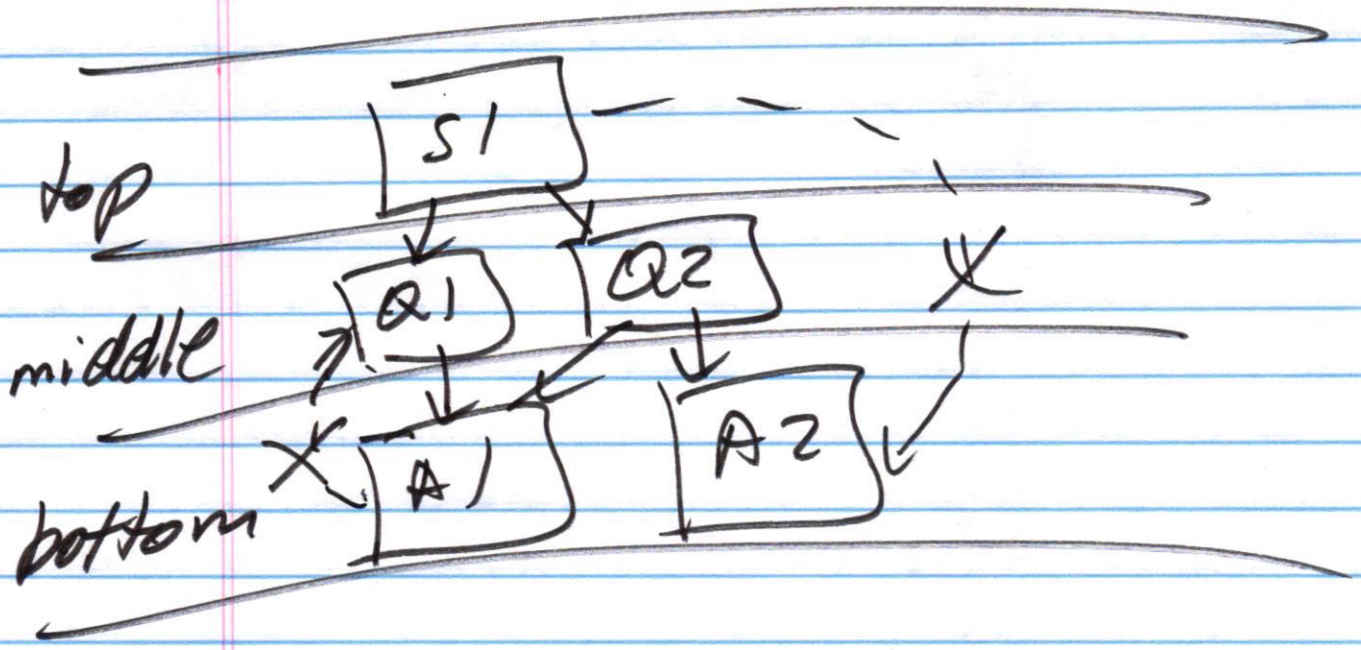
a set of layers

each layer interacts with two layers.

(1) layer above providing services.

(2) layer below opting services

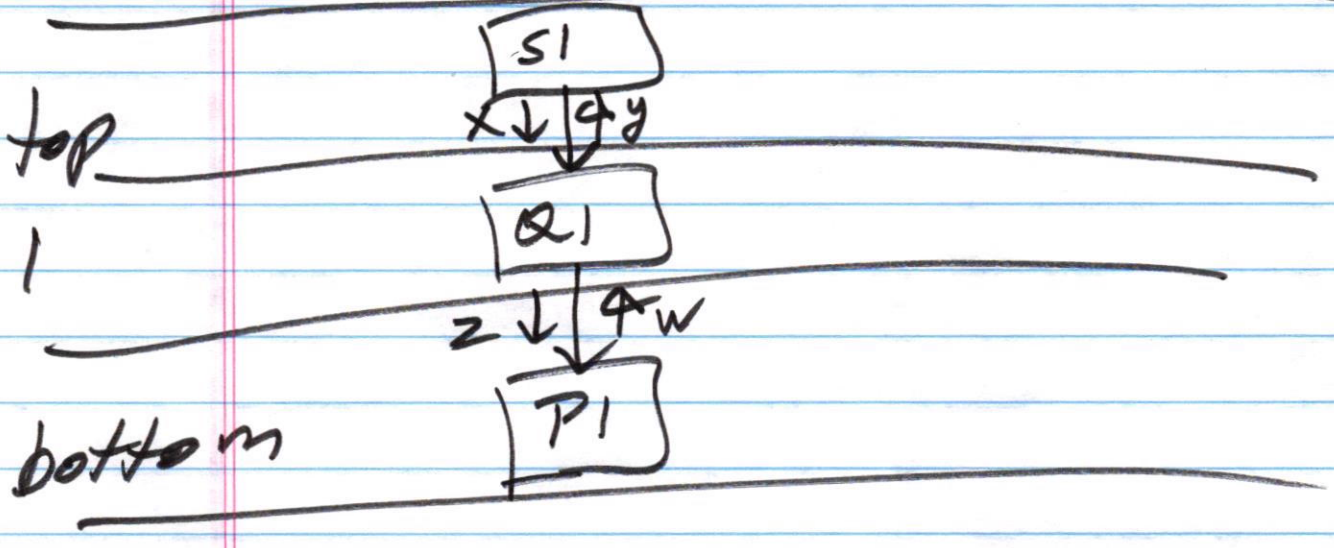
strict layered



communication between layers

1. top-down communication
2. bottom-up " "

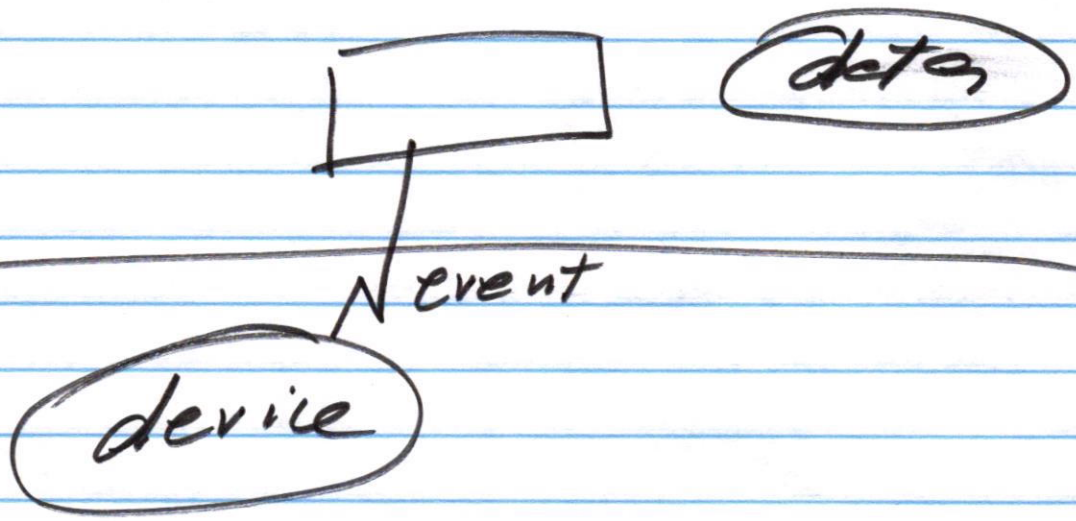
top-down communication

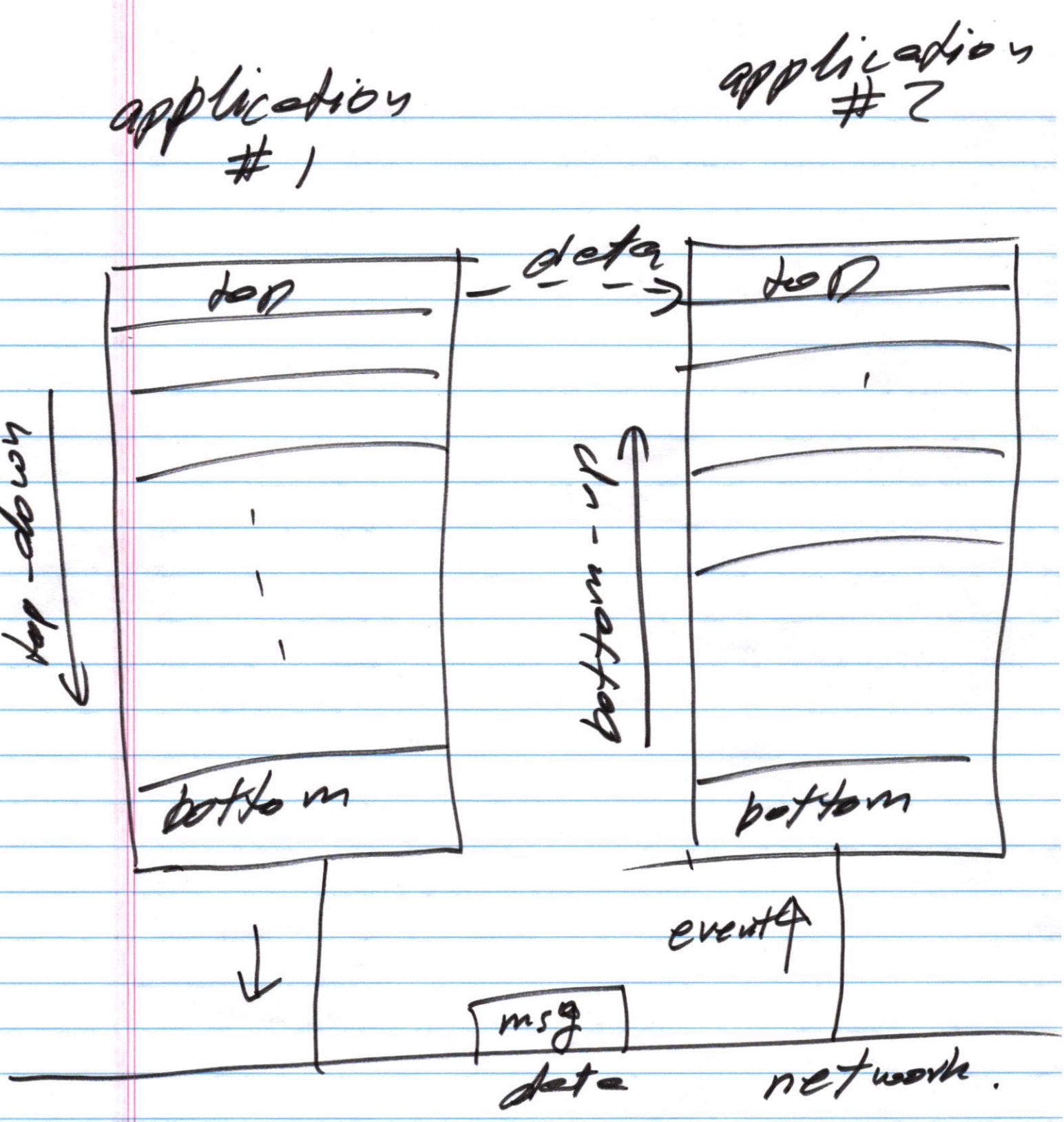


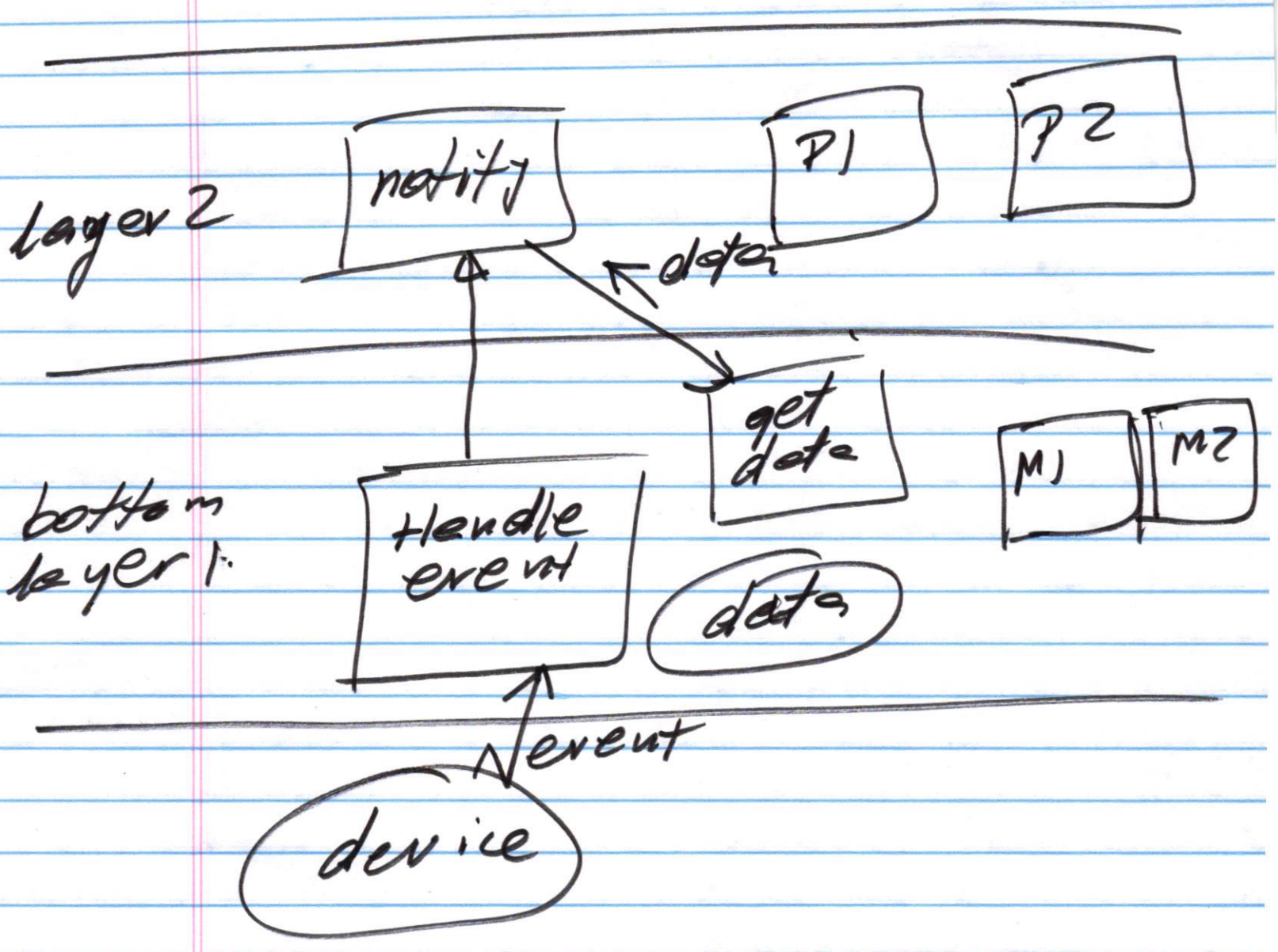
bottom-up communication

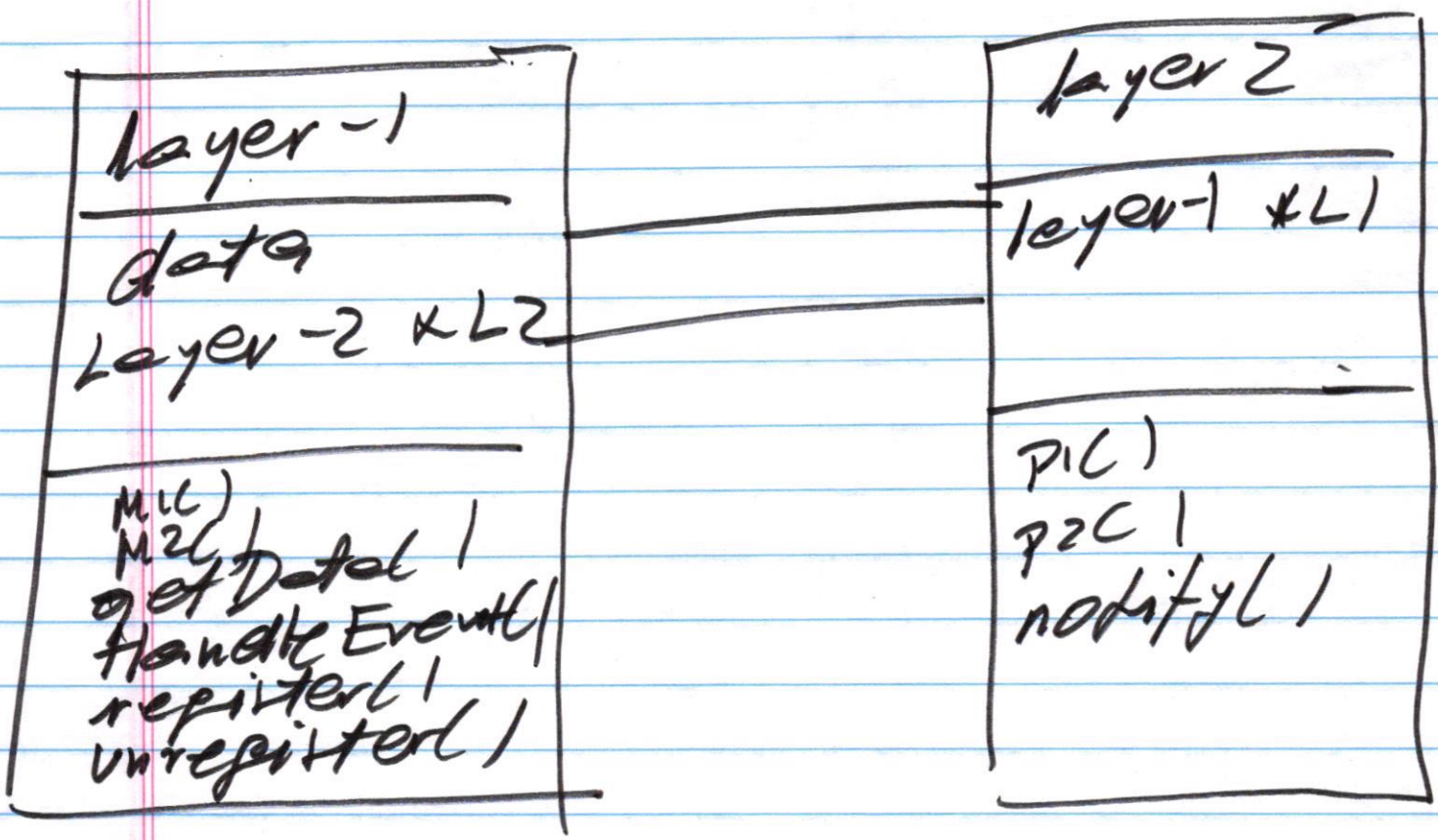
top

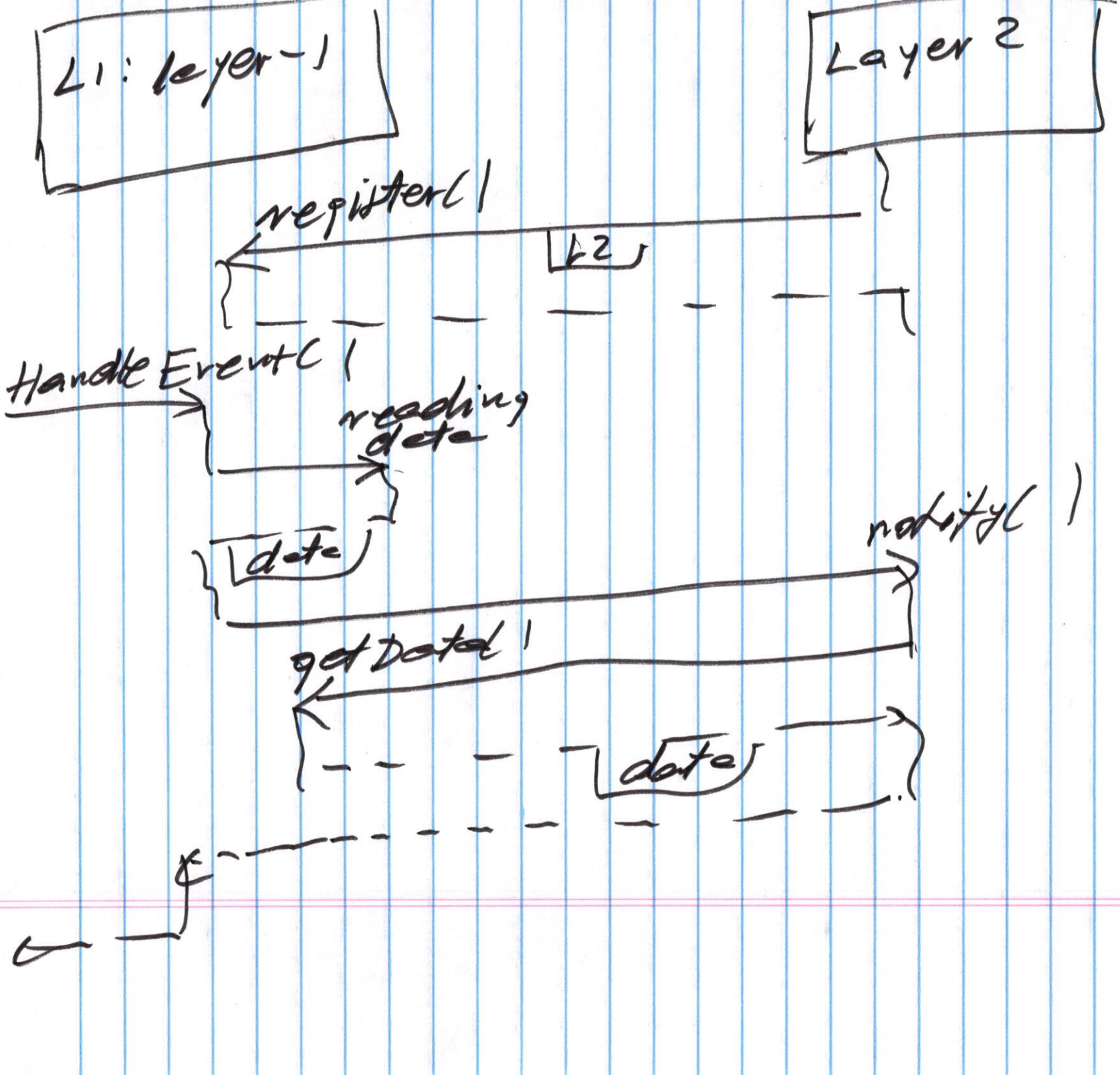
bottom




application #1
application #2
top
bottom
top
bottom
data
message
data-ready
event
msg
data
network.


layer 2
notify
P1
P2
data
get data
M1
M2
bottom layer
Handle event
data
event
device


Layer-1
data
Layer-2 *L2
M1()
M2()
getData()
HandleEvent()
registerL()
unregisterL()
Layer 2
layer-1 *L1
P1()
P2()
notify()


L1 : layer-1
Layer 2
register()
L2
HandleEvent()
reading data
data
notify()
getData()
data

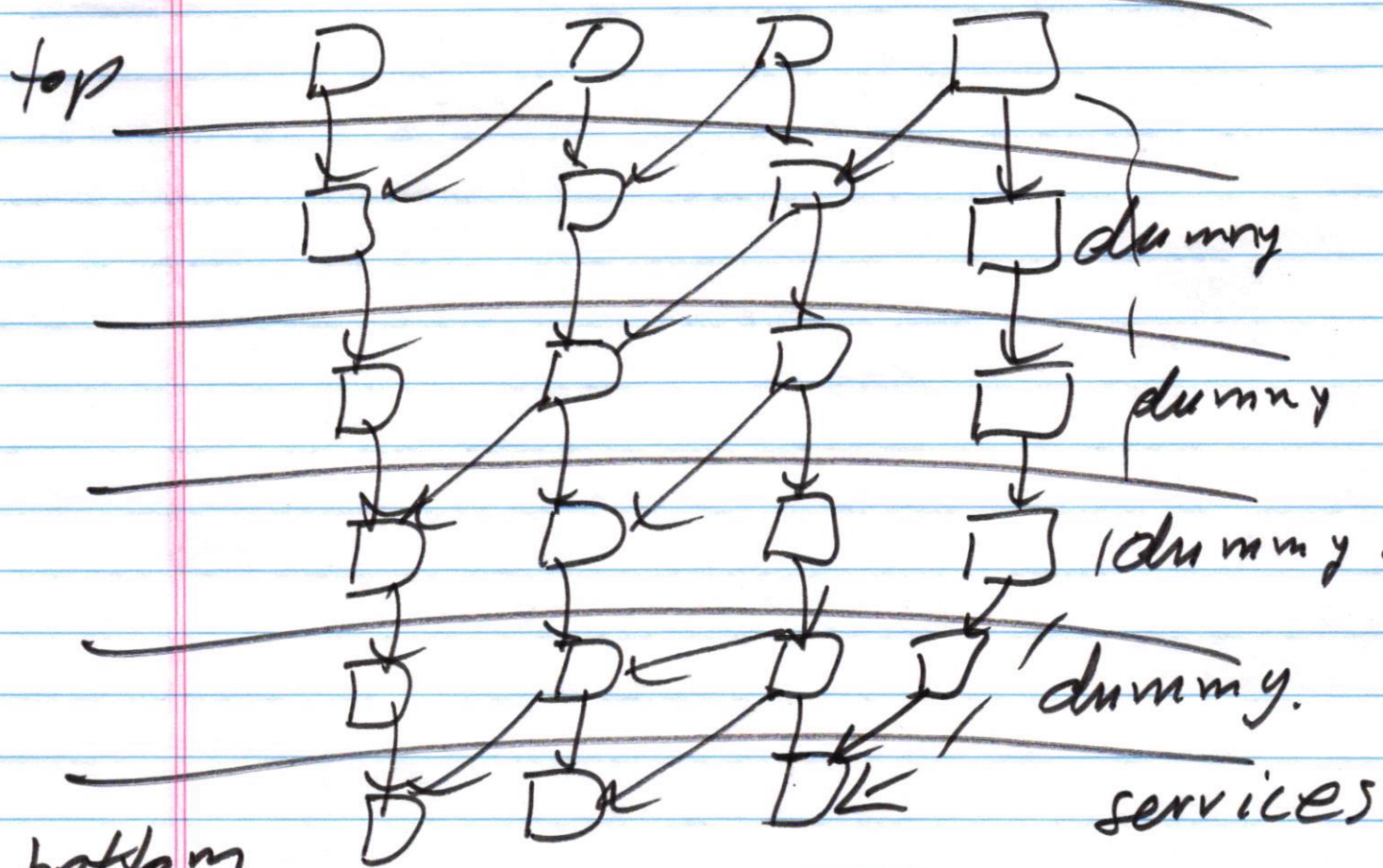
Strict layered architecture

each layer can interact with two layers.

1. Layer above by providing services.
2. Layer below by requesting services

strict layered architecture.

* performance problems

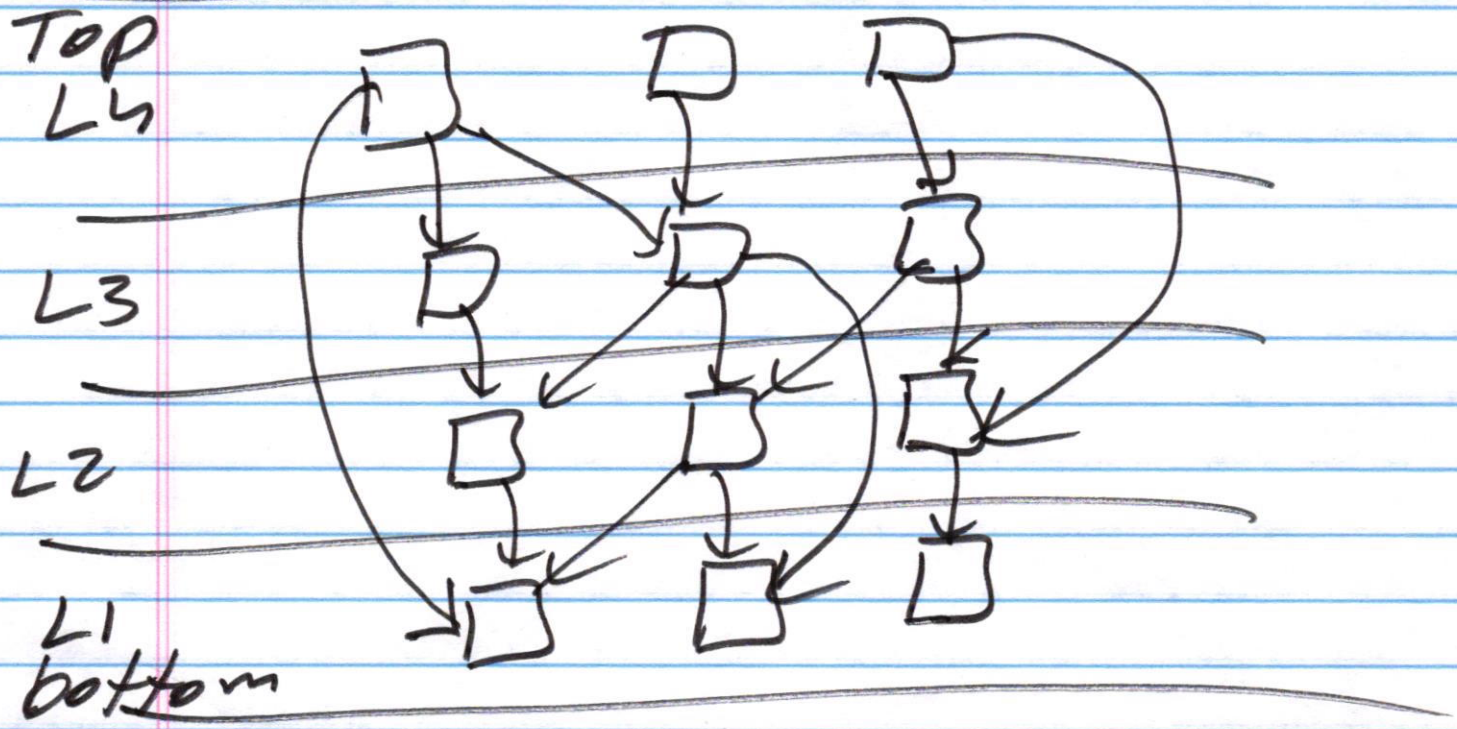


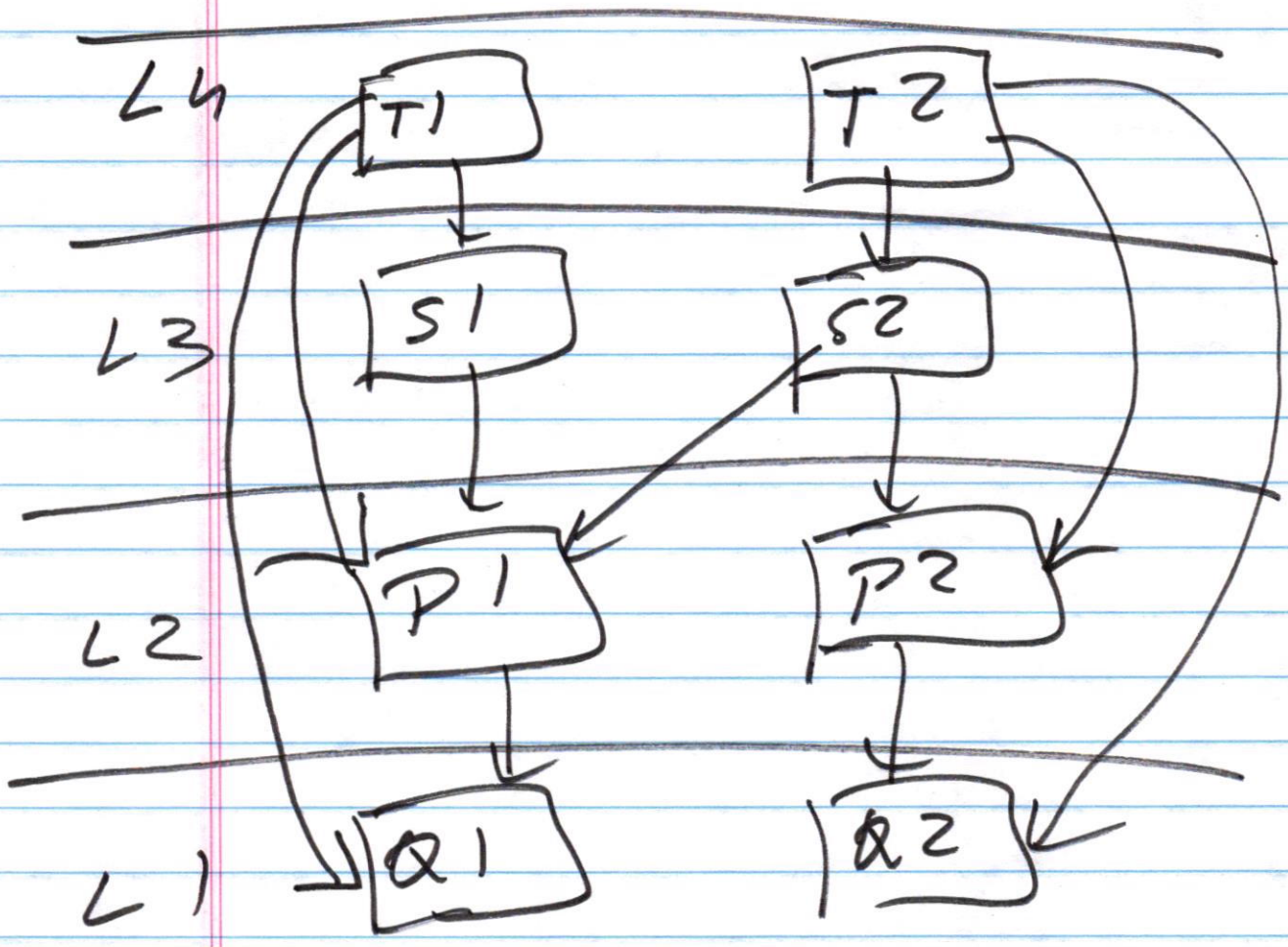
performance problems.

Relaxed layered architecture

Property:

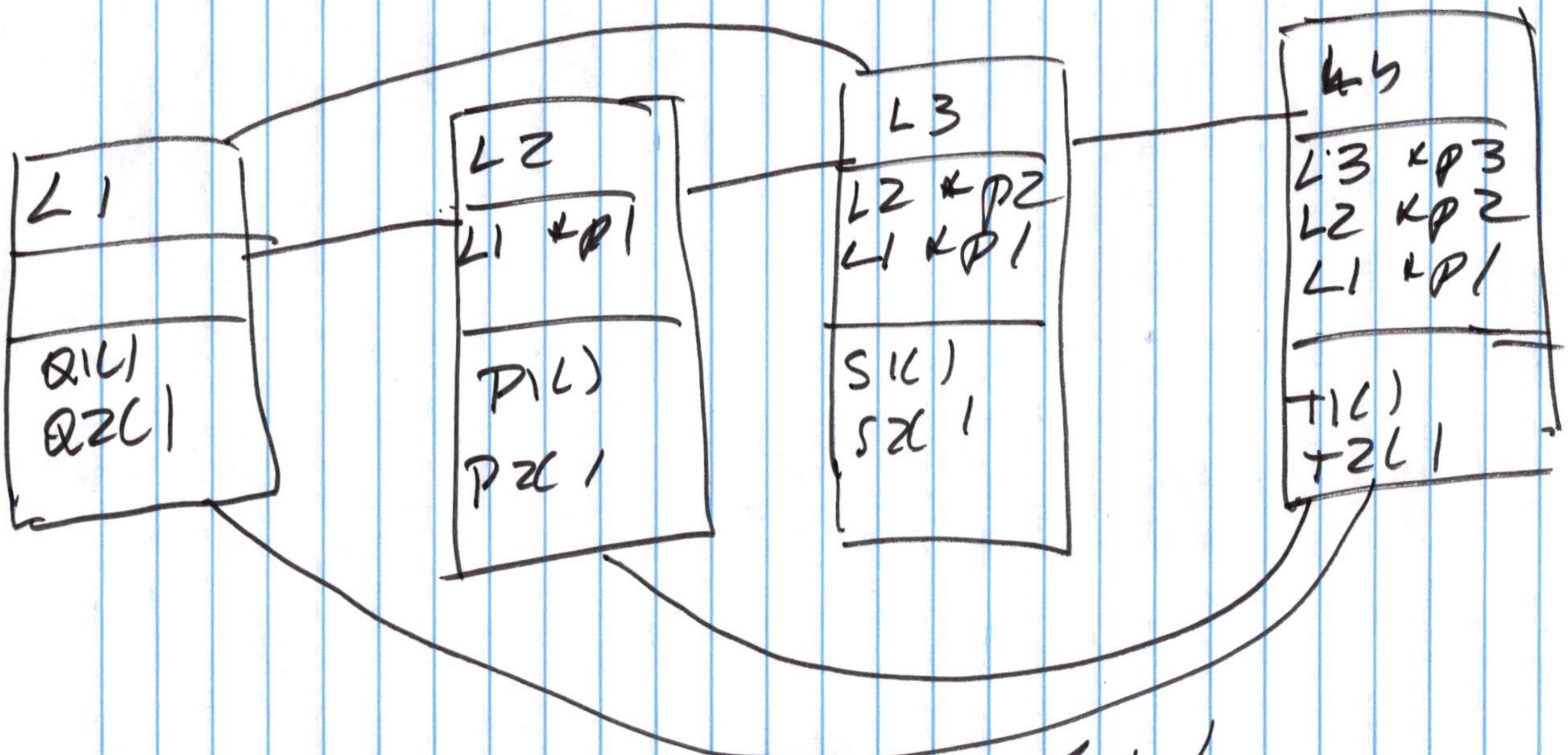
each layer can call "directly" services of all layers below





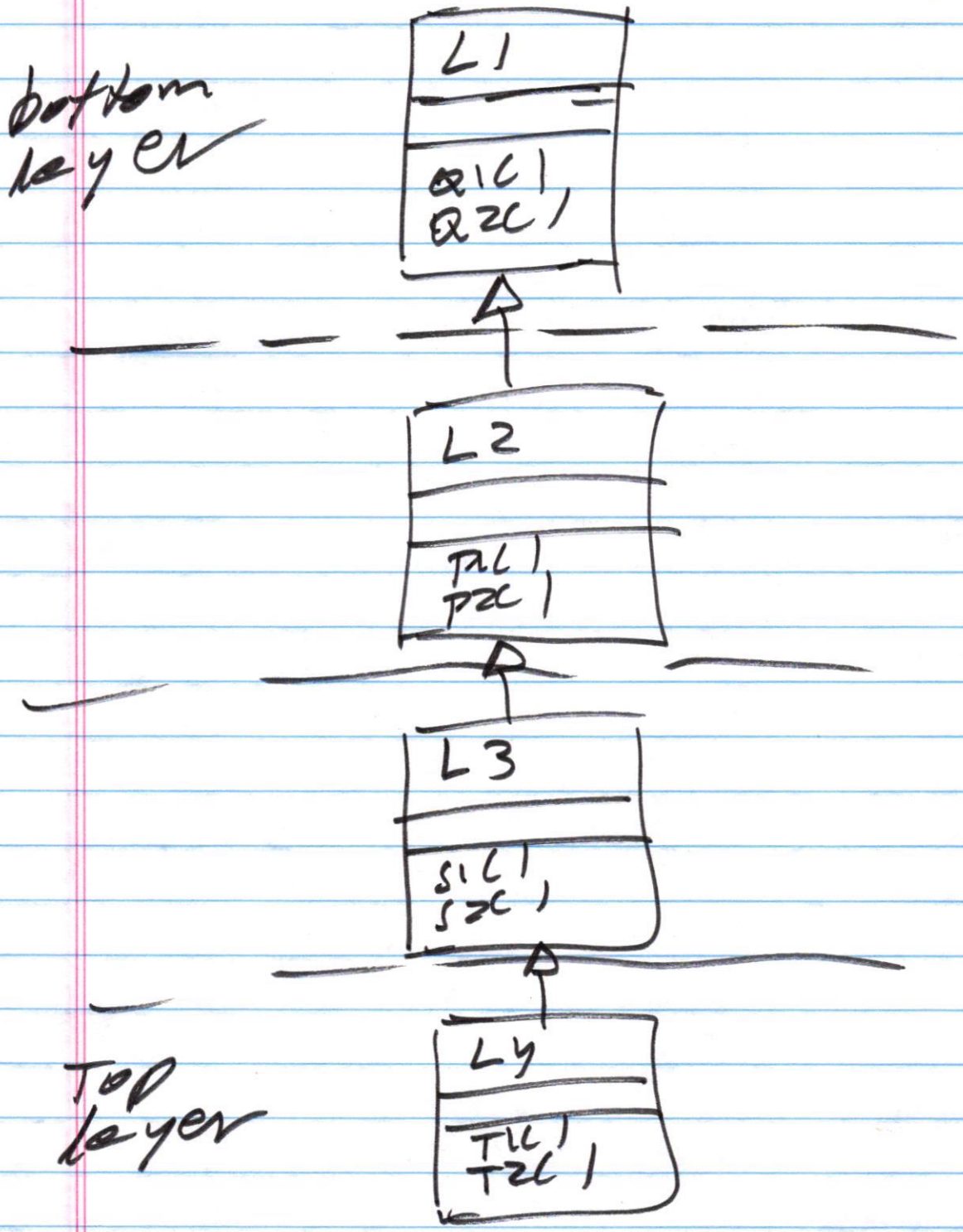
Adv: improved performance,

Disadv loss of maintainability.



Association-based solution!!!

inheritance based solution



Adv

relatively easy to implement

Disadv

very strong coupling between layers.