Project

Part #2 is posted

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

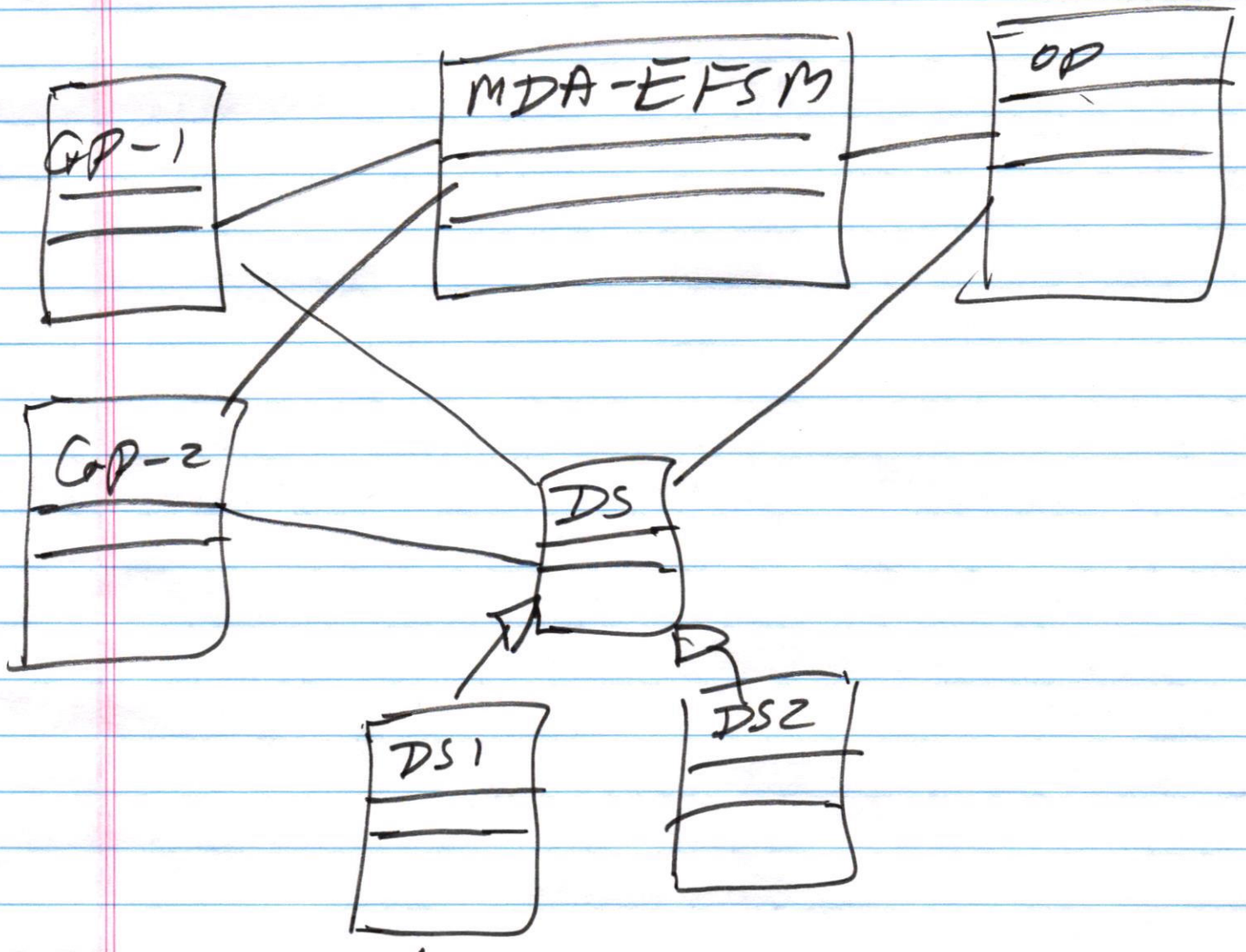
Design the GP components using MDA-EFSM from part # 1

(1) you can use your own MDA-EFSM providing. it is correct

OR

(2) you can use the posted sample MDA-EFSM

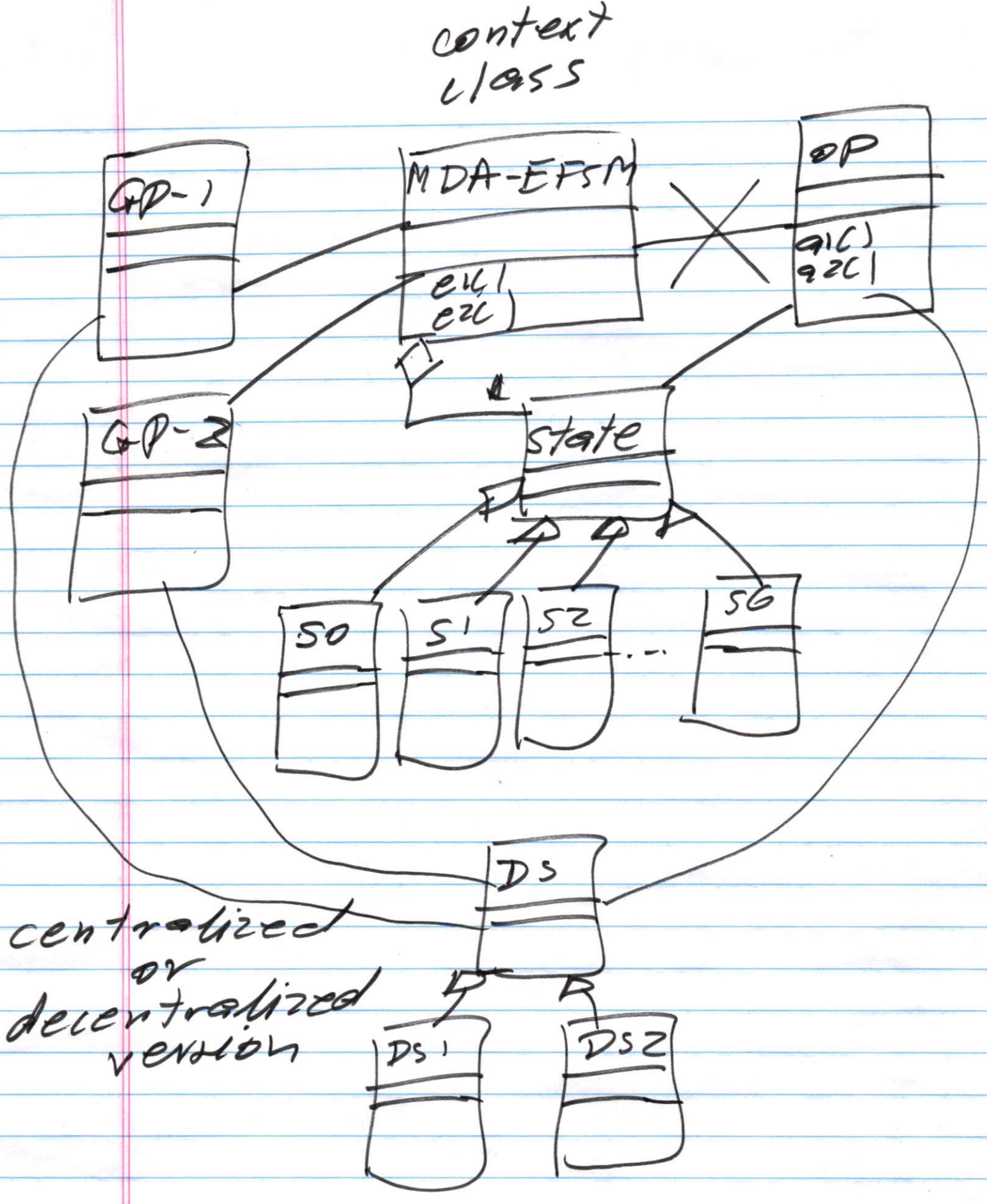
Initial Design for part #1



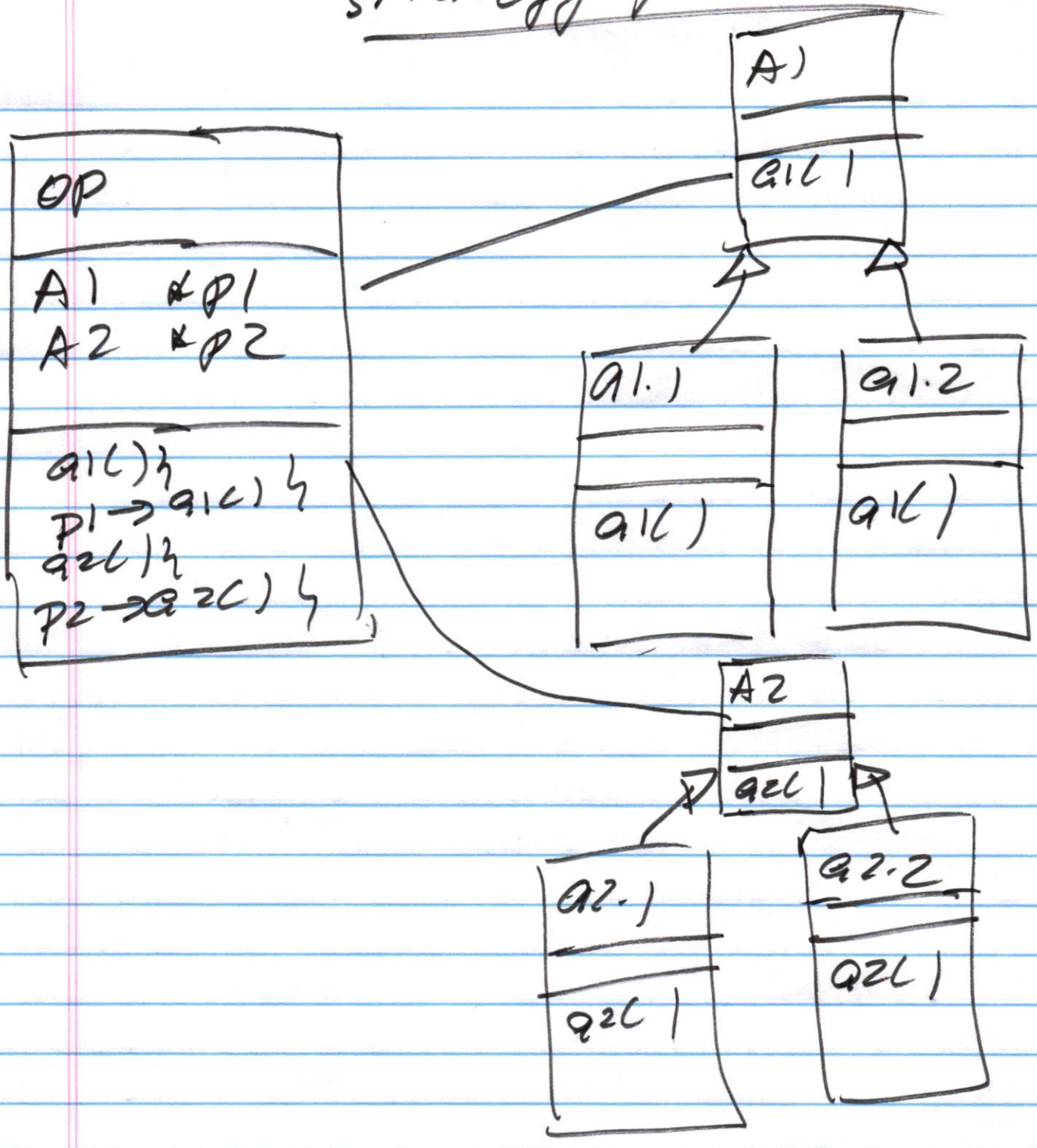
This design is not acceptable for part #2.

In the design for part #2 you MUST incorporate 3 design patterns.

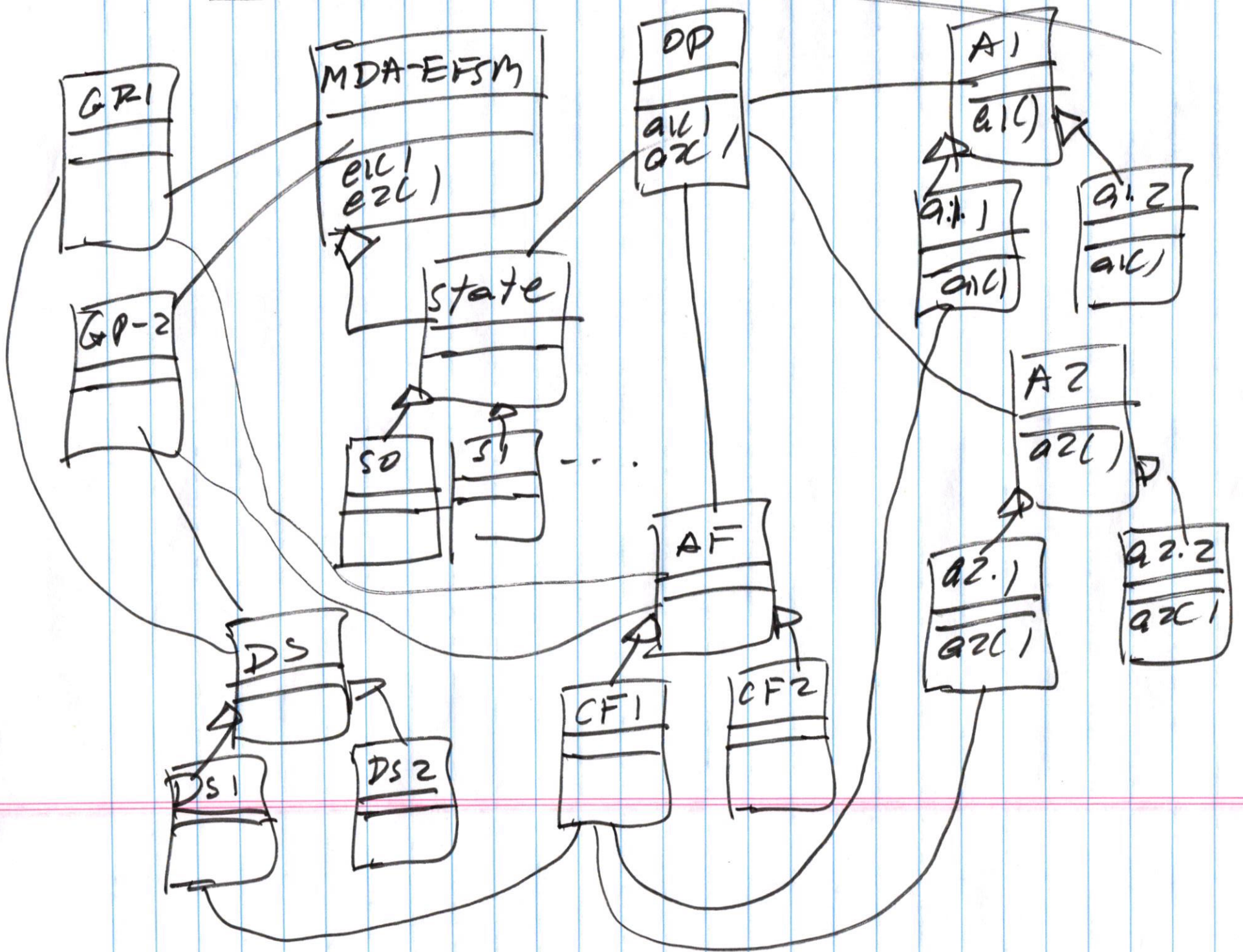
✓ (1) state pattern
✓ (2) strategy — " —
✓ (3) abstract factory pattern


context class
GP-1
MDA-EFSM
e1C)
e2C)
OP
q1C)
q2C)
GP-2
state
S0
S1
S2
S6
DS
centralized or decentralized version
DS1
DS2

strategy pattern



Abstract factory pattern



After the design

Implementation of the design using. OO prog. Language.

to test the design create a test driver to execute and test the correcness of the design and the implementation

Layered architecture

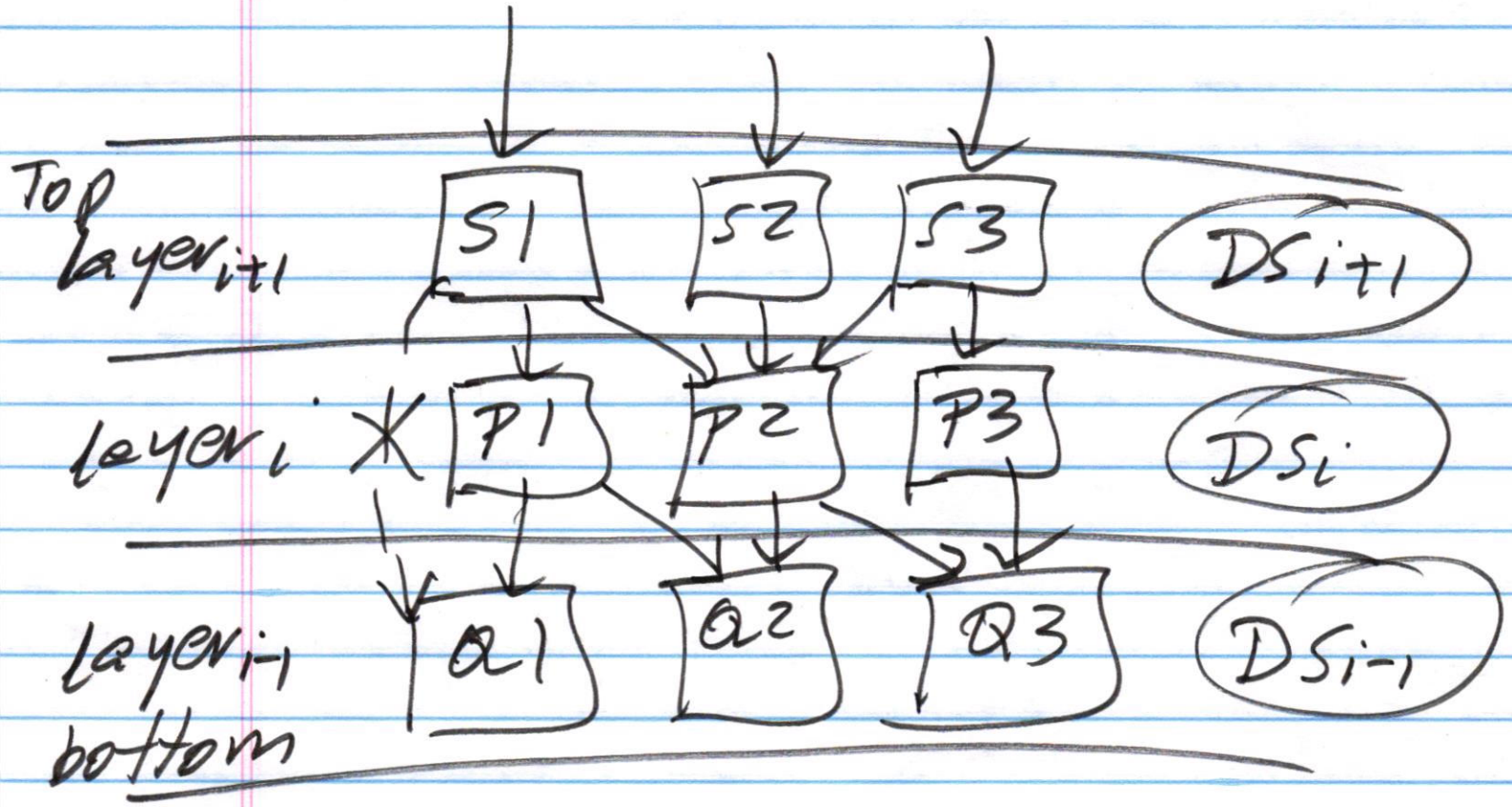
a set of components
+
relationships between components

---

component: layer

relationship: call relationship

"strict" layered architecture



"strict" layered architecture

Properties

1. hierarchical approach
2. a set of layers
3. each layer provides services to the layer above
4. supports the design based on the increasing level of abstraction.
5. Information hiding is supported
6. Each layer interacts with at most two layers.
   1. layer above
   2. layer below

Examples

* communication systems
* operating — " —
* data base — " —

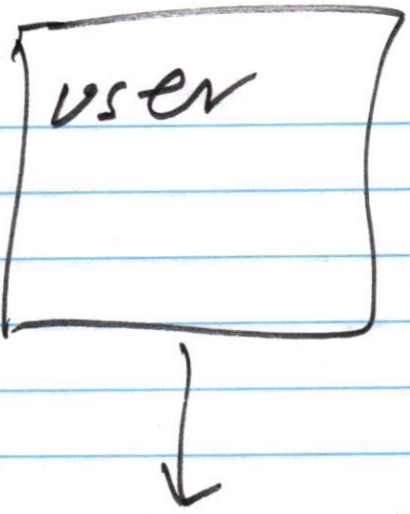
Networking.

application layer

Network layer

data link layer

physical layer



user interface layer

application layer.

core functionality layer

data base layer

1. Identify layers.

2. identify responsibilities and services provided by each layer

1. define abstraction & criteria for grouping together responsibilities.
2. identify layers.
3. identify responsibilities and services for each layer
4. identify interfaces / signatures for each services.
5. evaluate the layered architecture.

not ok.

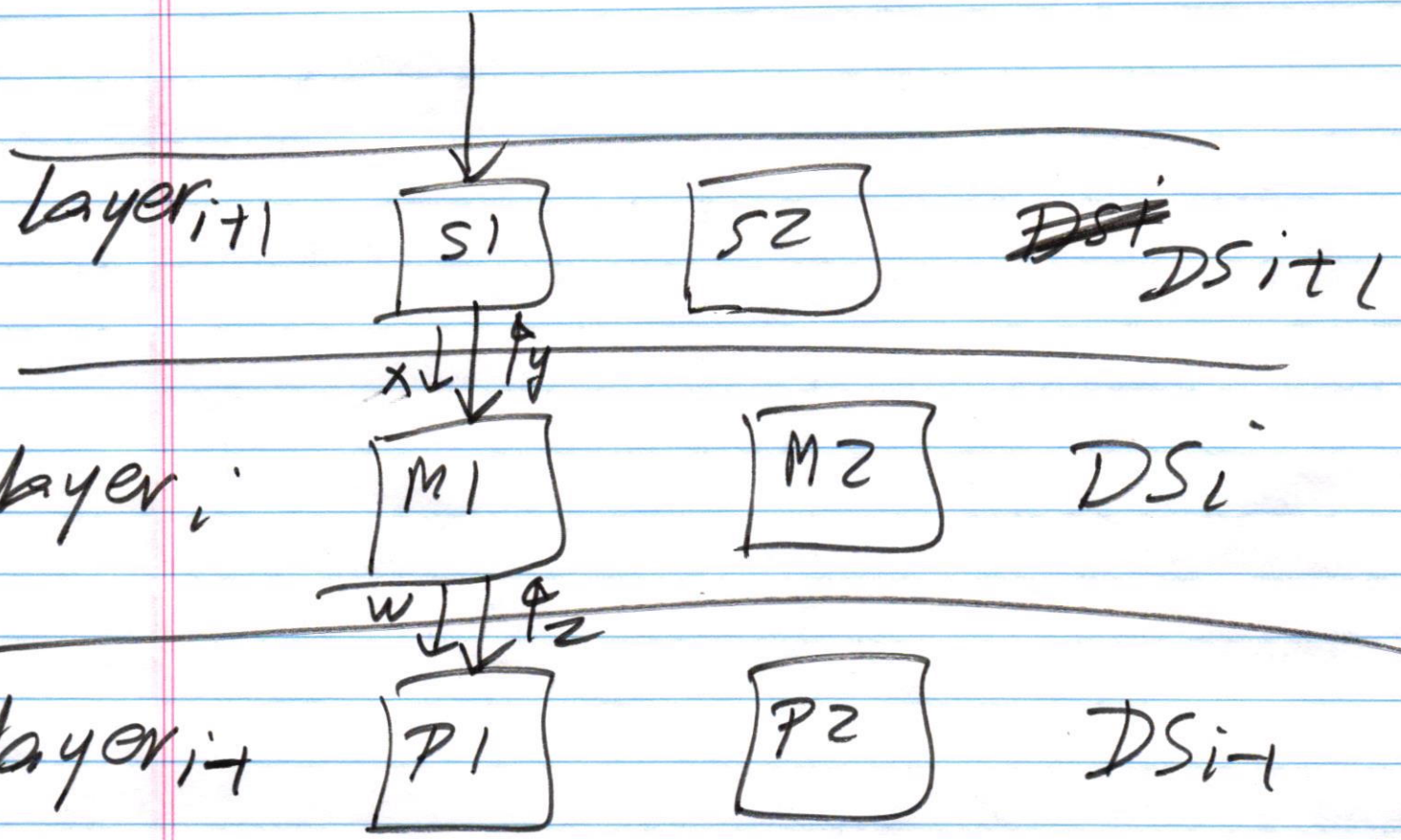
ok

communication between layers

1. top-down communication (request)

2. Bottom-up communication (notification)

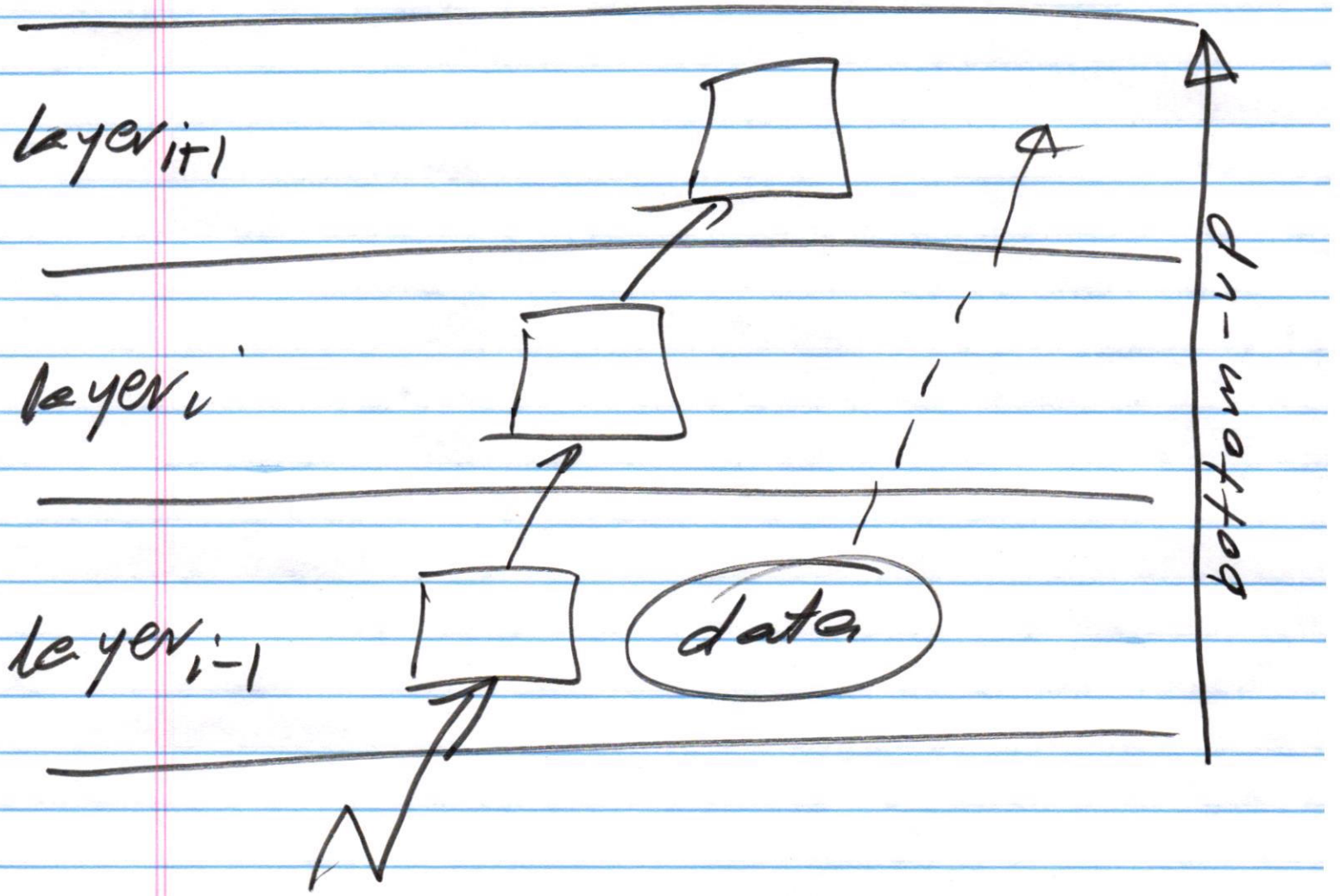
Top-down communication

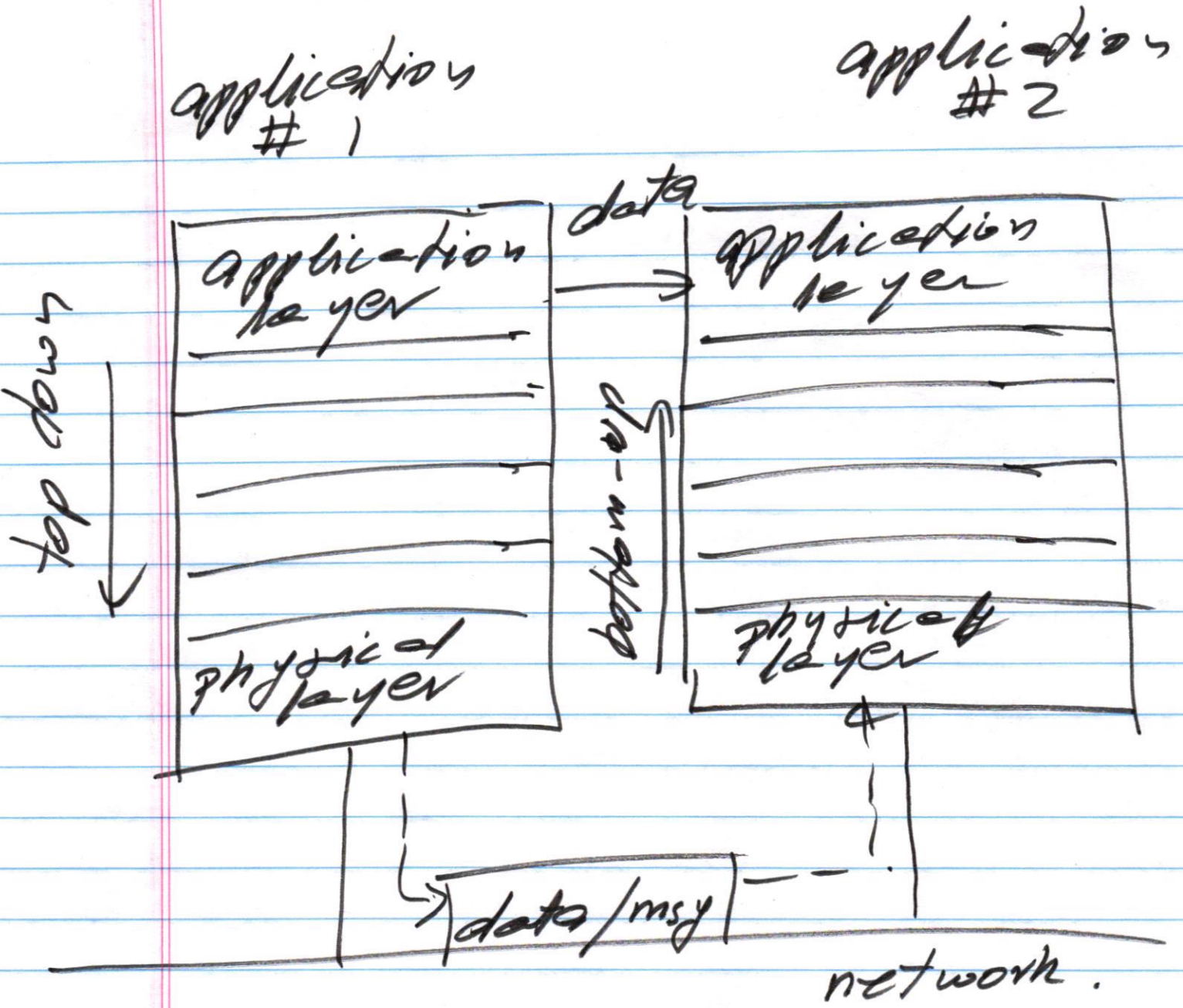


Data is exchanged between layers using parameter passing.

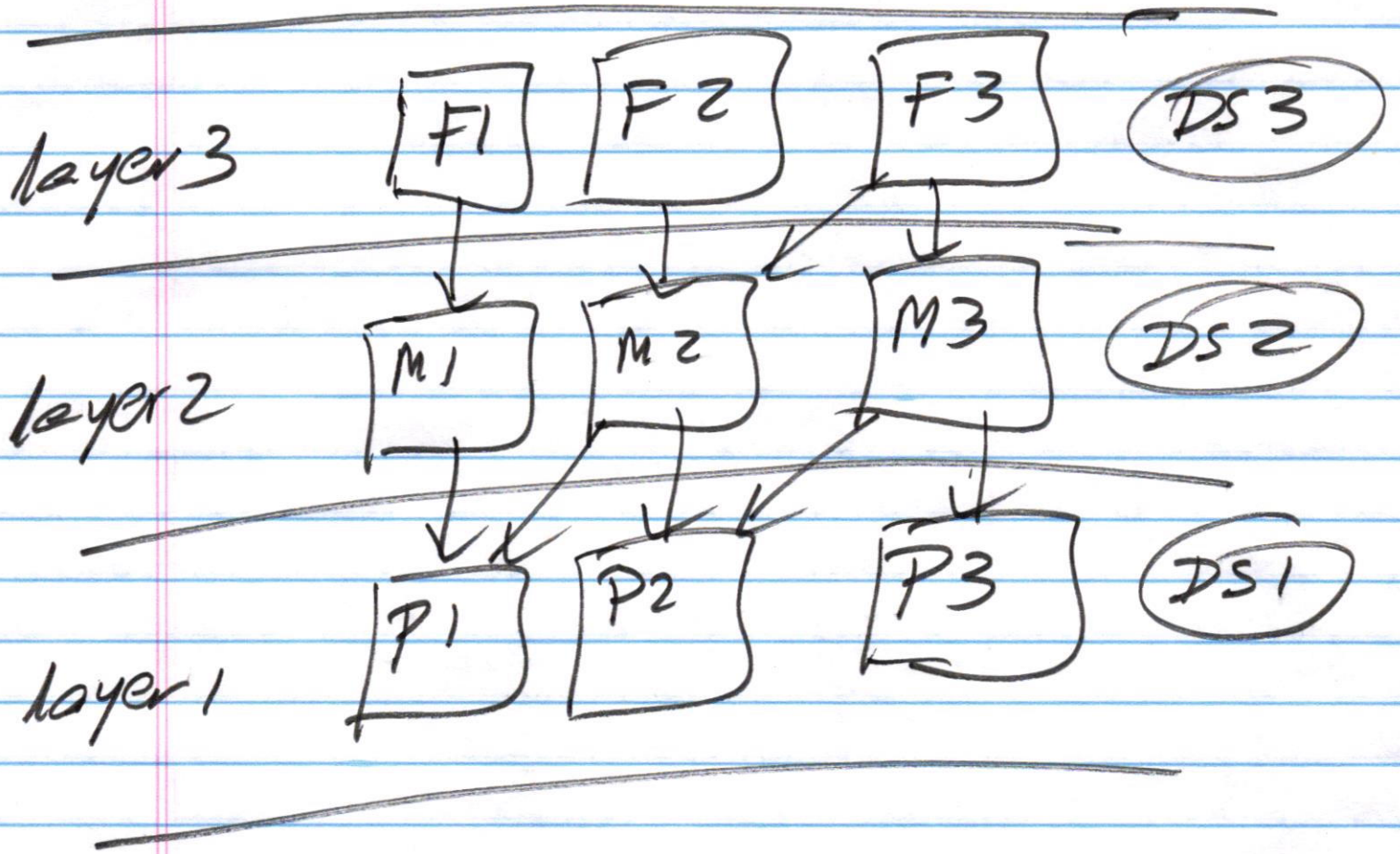
No global data structure!!

Bottom-up communication

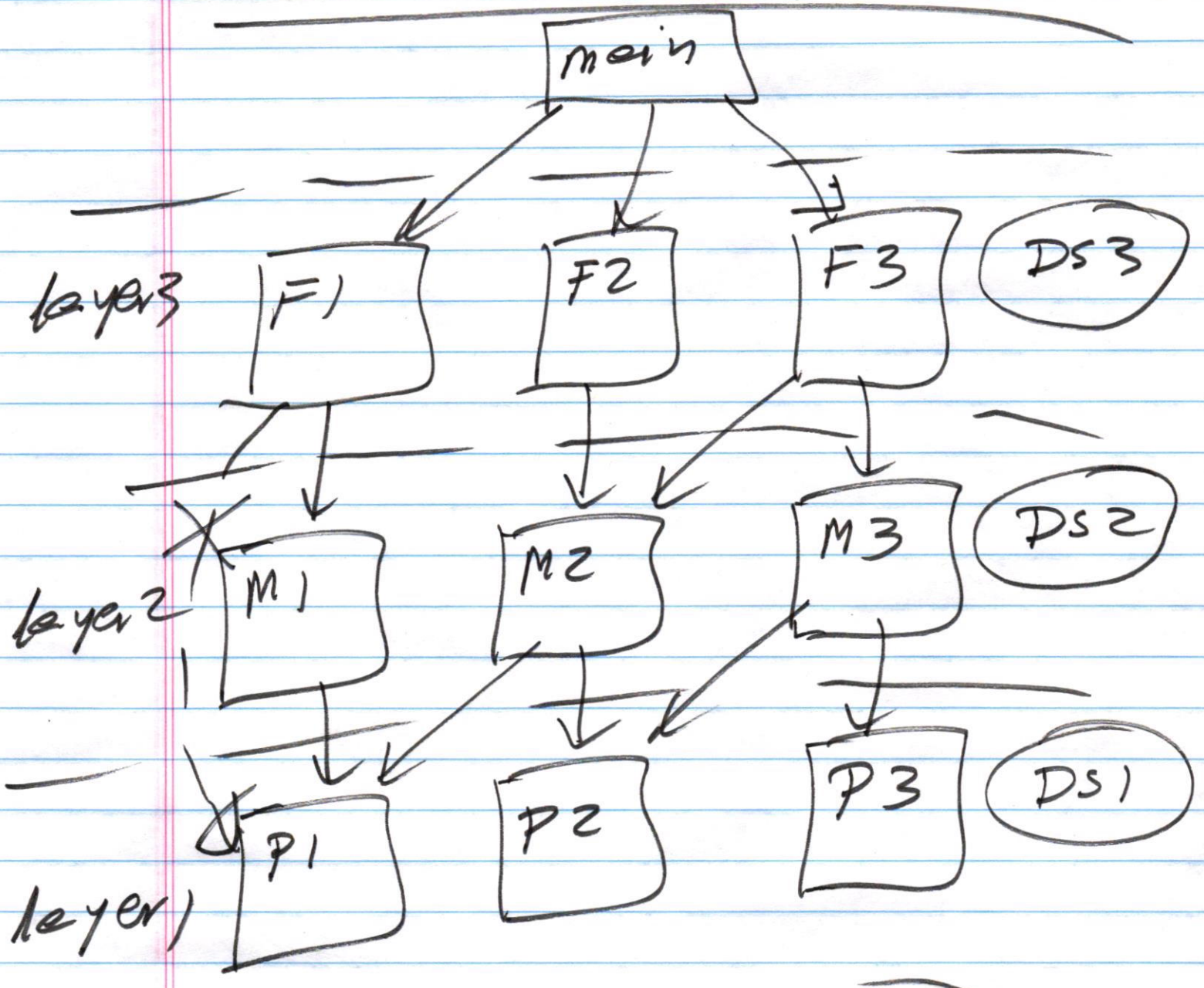



application #1
application #2
application layer
physical layer
application layer
physical layer
data
top down
bottom up
data/msg
network.

Designing layers

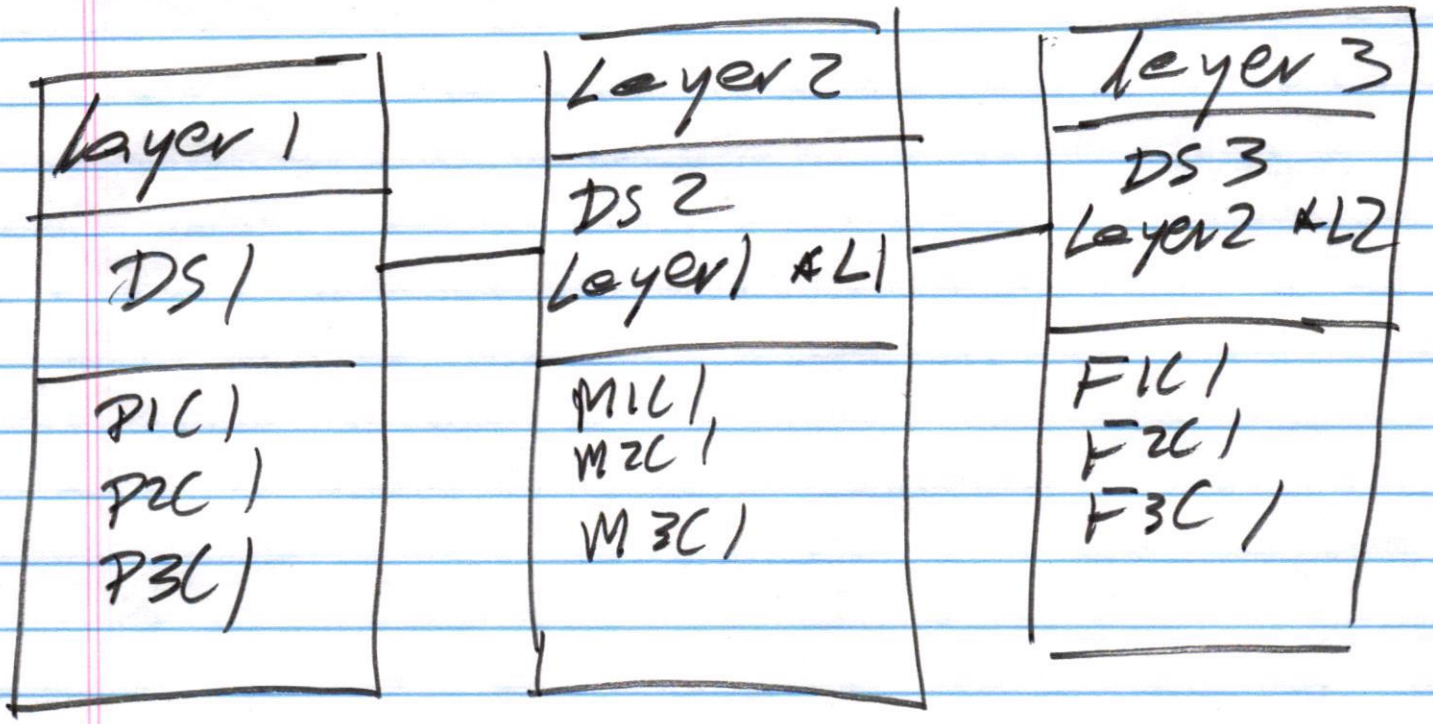


modular design



* hard to enforce information hiding.

* any module can call any other module.



layers are represented by classes, using, OO design