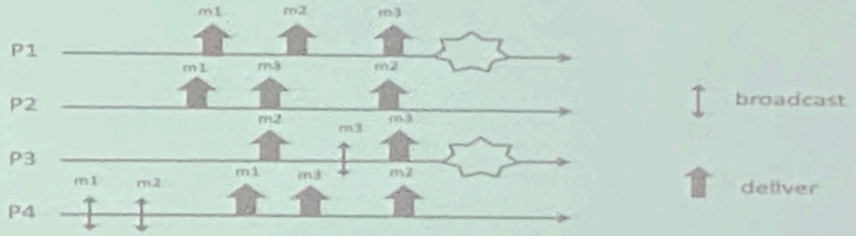
Esercizi 12 Dicembre

Ex 1: Consider the execution depicted in the Figure



Answer to the following questions:

1. Which is the strongest TO specification satisfied by the proposed run? Motivate your answer.
2. Does the proposed execution satisfy Causal order Broadcast, FIFO Order Broadcast or none of them?
3. Modify the execution in order to satisfy TO(UA, WUTO) but not TO(UA, SUTO).
4. Modify the execution in order to satisfy TO(NUA, WUTO) but not TO(NUA, SUTO).

NOTE: In order to solve point 3 and point 4 you can only add messages and/or failures.

Solution:

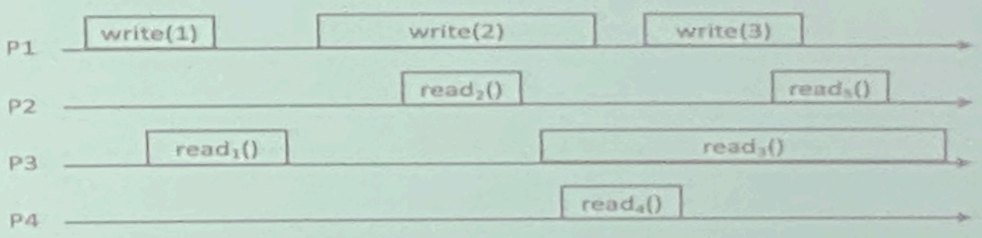
1_Point: We can have: Uniform and Non Uniform agreement. In this case we have a uniform agreement.
Let's check what's happen at the faulty: the order of the faulty is different from the order of the correct so not uniform total order.

2_Point: We have to identify the relationship: FIFO order we can say that is satisfied. Casual order: the execution of it is FIFO bu not casual.

3_Point: It's impossible if I cannot remove any messages, but if I can remove can I reach the solution .

4_Point: We have to modify the agreement. It's impossible.

Ex 2: Consider the execution depicted in the following figure and answer the questions



1. Define ALL the values that can be returned by read operations (Rx) assuming the run refers to a regular register.
2. Define ALL the values that can be returned by read operations (Rx) assuming the run refers to an atomic register.
3. Let us assume that values retuned by read operations are as follow: read$_1$() → 0, read$_2$() → 2, read$_3$() → 3, read$_4$() → 3, read$_5$() → 2. Is the run depicted in the Figure linearizable?

Solution

1. Read1 is concurrently with write1. So one has to be 0 and the other 1… read5 is concurrently with write 3.. and so on.
2. .. read3 is following read2 so it depending from read2. Exactly the same read4. I have to see the right solution on the slides.
NB: one popular mistake it that 3 depends on 4 but in this case are concurrent. We have to know distinguish from the figure the concurrent and the dependency.
3. —- Porco dio non ho fatto in tempo.

```
upon event ⟨ rb, Broadcast | m ⟩ do
    trigger ⟨ beb, Broadcast | [DATA, self, m] ⟩;

upon event ⟨ beb, Deliver | p, [DATA, s, m] ⟩ do
    if m ∉ from[s] then
        trigger ⟨ rb, Deliver | s, m ⟩;
        from[s] := from[s] ∪ {m};

upon event ⟨◊P,Suspect |p⟩do
    correct := correct \ {p};
    forall m ∈ from[p] do
        trigger ⟨ beb, Broadcast | [DATA, p, m] ⟩;

upon event ⟨◊P,Restore |p⟩do
    correct := correct ∪ {p};
```

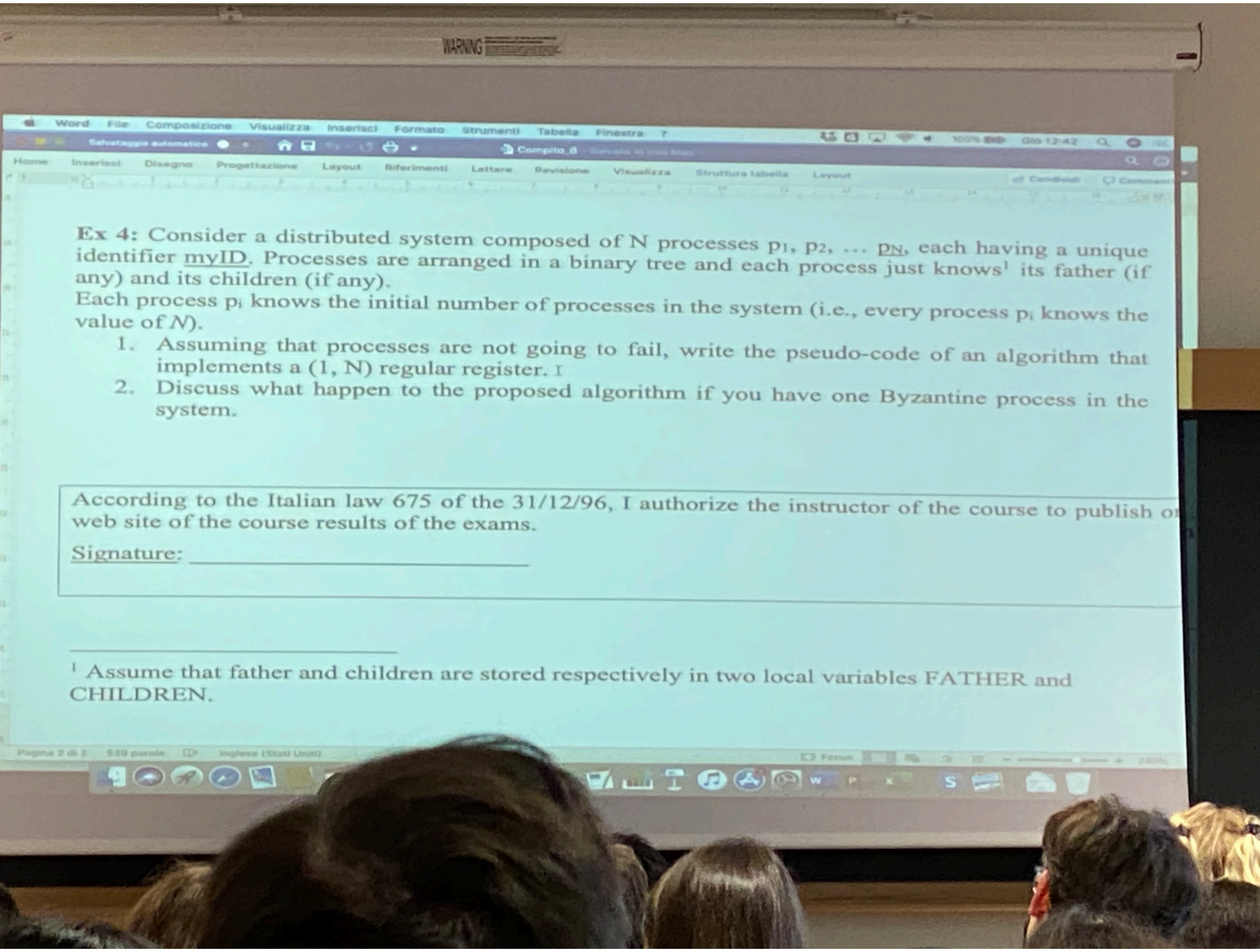
Assuming that the algorithm is using a Best Effort Broadcast primitive and an Eventually Perfec
Failure Detector ◊P discuss if the following properties are satisfied or not and motivate your answe

- *Validity*: If a correct process p broadcasts a message m, then p eventually delivers m.
- *No duplication*: No message is delivered more than once.
- *No creation*: If a process delivers a message m with sender s, then m was p
by process s.
- *Agreement*: If a message m is delivered by some correct process, then m is
delivered by every correct process.

Previously broadcast

Eventually

Validity**:** ?
NO duplication**:** is satisfied.
NO creation: satisfied.
Agreement: not satisfied.



Solution:

INIT

```
Father = get_father( )
Children = get_children( )
Val = 0
Ack = empty

upon event read( ):
        trigger read_return(val)

upon event write( v ):
        Val = v
        if father = null            % the writer is the root
                for each p ∈ Children:
                        trigger pp2pSend(WRITE, v)

        else:
                trigger pp2pSend(WRITE_ROOT, v) to father
```

#now I have to handle this two messages that I have generate

```
upon event pp2pDeliver(WRITE, v) from father
        Val = v
        if  children = null
                trigger pp2pSend(ACK, myID) to father
        else:
                for each p ∈ Children:
                        trigger pp2pSend(WRITE,v) to p


upon event pp2pDeliver(WRITE_ROOT, v) from p
        if father = null            % the writer is the root
                for each p ∈ Children:
                        trigger pp2pSend(WRITE, v) to p

        else:
                trigger pp2pSend(WRITE_ROOT, v) to father



upon event pp2pDeliver(ACK, myID) from p
        if father = null:
                ack = ack U {id}

        else:
                trigger pp2pSend(ACK, id) to father



when |ack| = N
        for each p ∈ Children:
                trigger pp2pSend(WRITE_COMPLETE) to p



upon event pp2pSend(WRITE_COMPLETE) from father:
        if myID = writer
                trigger write_Return

        else:
                for each p ∈ Children:
                        trigger pp2pSend(WRITE_COMPLETE) to p
```

Secondo punto: è ora de magna