- 28 Lezione

giovedì 9 maggio 2019 16:07

PROBLEM

LEZIONE IMPORTANTE PER IL REPORT DEL PROGETTO!!

Understand the business logic code

"timings" — "Control" — "data access" (state)

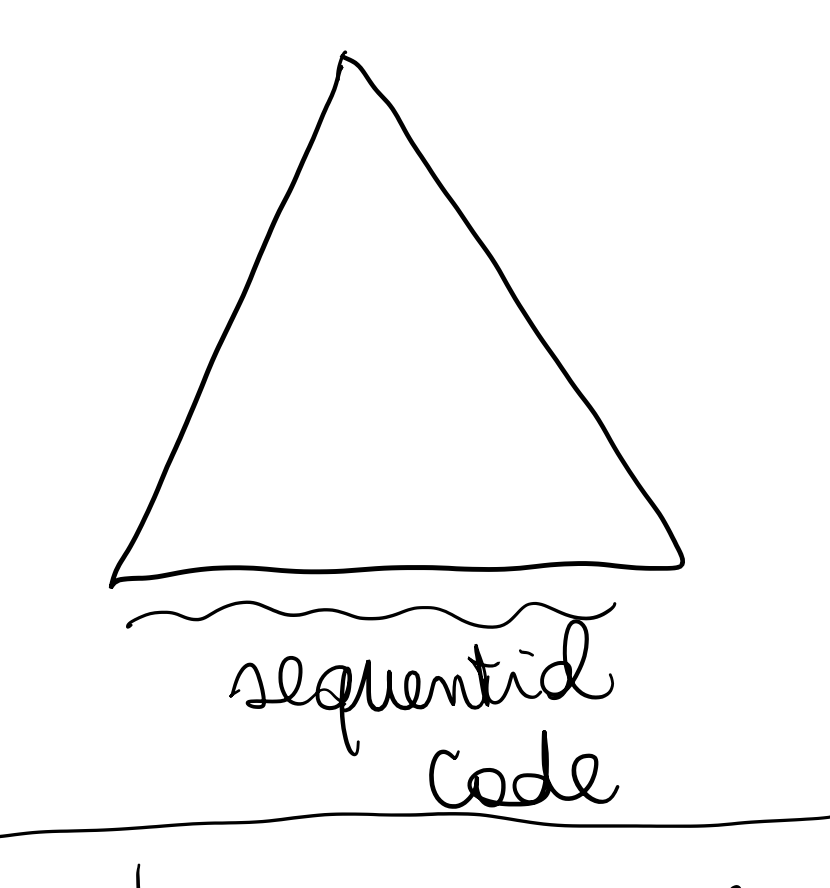
at the "logic" level

Process to follow

→ Pick up the correct patterns

- Choose Pi among alternatives
- Recognize and pattern match the application (the code)
  - ↓ It may require that you use the low level mechanism
  - → Use the patterns using BB level ↓ building block
- → Pattern composition: Because very often you only have pattern for some situations and you only have to compose these patterns.

Pattern Composition: a tree of patterns.



If I have to evaluate some data parallel pattern
↳ You can consider alternatives: if you have GPU move the computation to GPU.

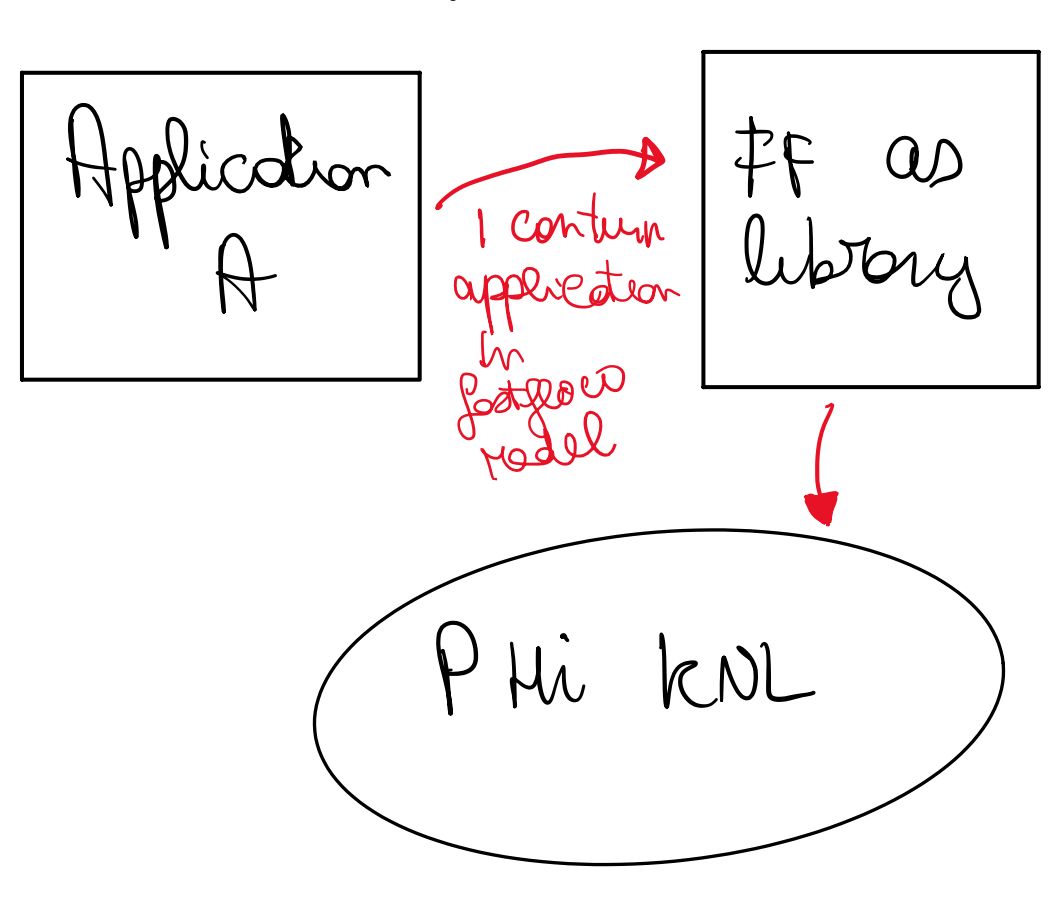
In certain case you can also change the pattern that you use. 16:28

First level: Consider alternatives:

- Different implementation of one pattern
- Different substitutes of one pattern
- Refactoring of my subtree

these alternatives must be evaluated using the performance model.

If I want to consider:



16:34 this performance model can be used in more light way.

If I figure out the barrier is not necessary for the application, if I succeed put 16:35

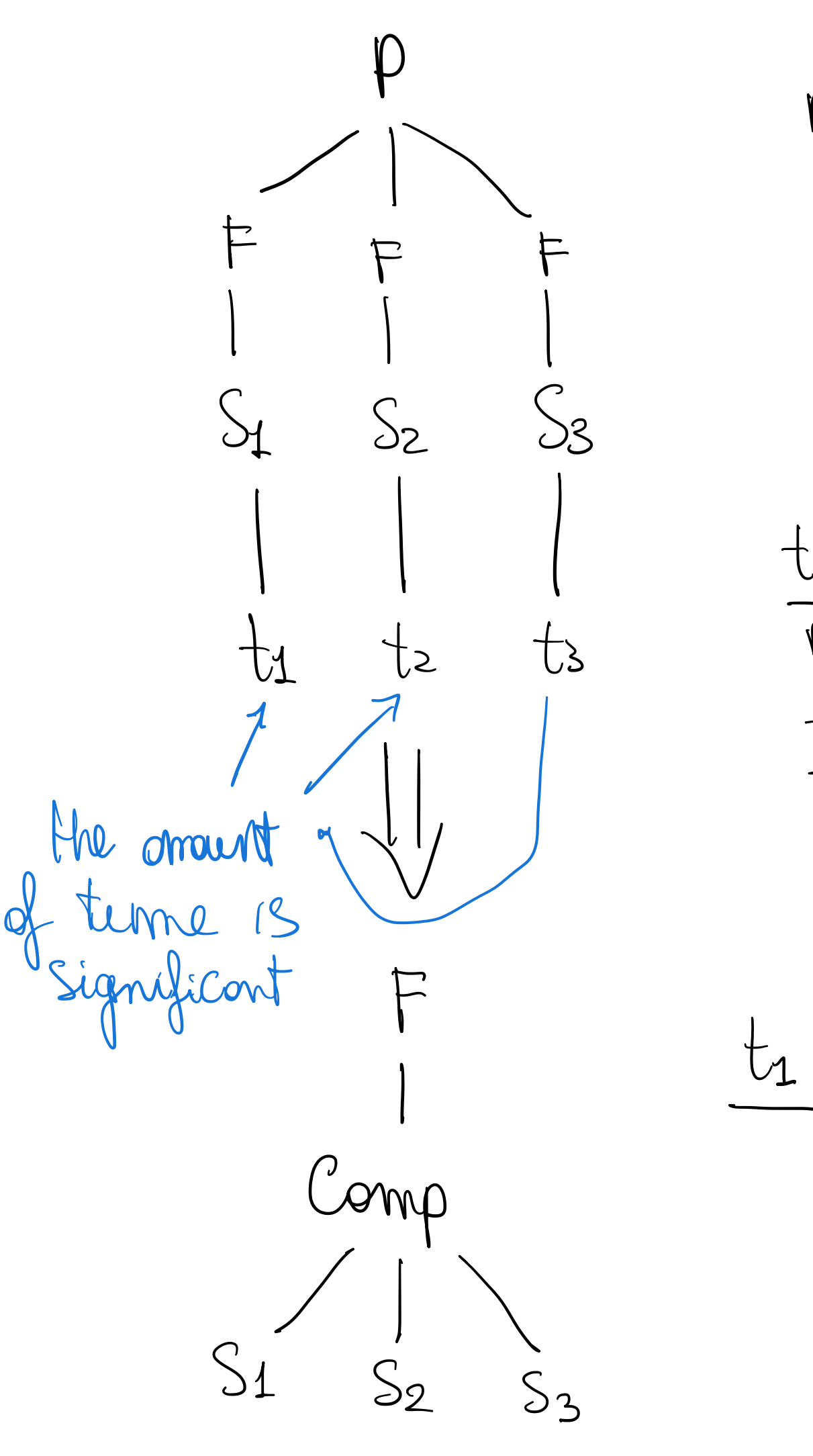
this is beneficial in any kind of architectures unless in some applications where there is a phase followed by another phase and the code diverges.

Ascoltare dal minuto 16:34 in poi.

$\alpha f \circ \alpha g$
↓
$\alpha(f \circ g)$ — DATA parallel stuff

normal form for stream parallel application

Example



n_1 worker in 1° farm:

$t_{s_1} = \frac{t_1}{n_1}$

$t_{s_2} = \frac{t_2}{n_2}$

$\frac{t_2}{n_2} \approx \frac{t_1}{n_1}$

$\frac{t_3}{n_3} \approx \frac{t_1}{n_1}$

$\frac{t_1 + t_2 + t_3}{n_w}$ in normal form

If you declare some threads: 16:43
→

Use Performance model in this way to understand the best pattern ?? 16:45

OVERHEADS — "SERIAL FRACTION"
- AMDAHL Perspective
- WORK/SPAN perspective

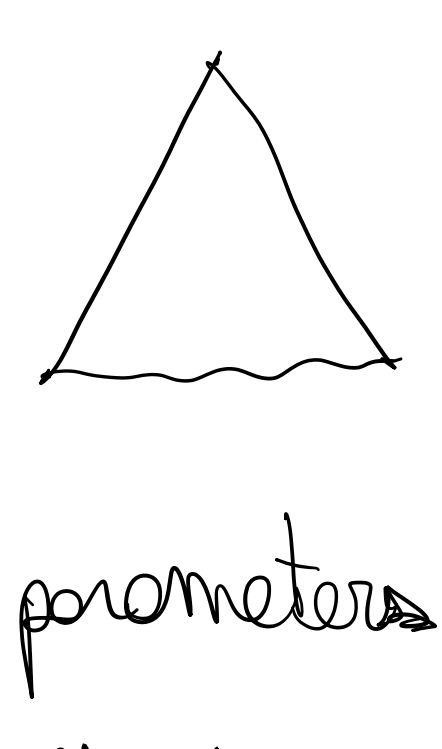
Targeting Hardware

Does not mean

But means considering different implementation.

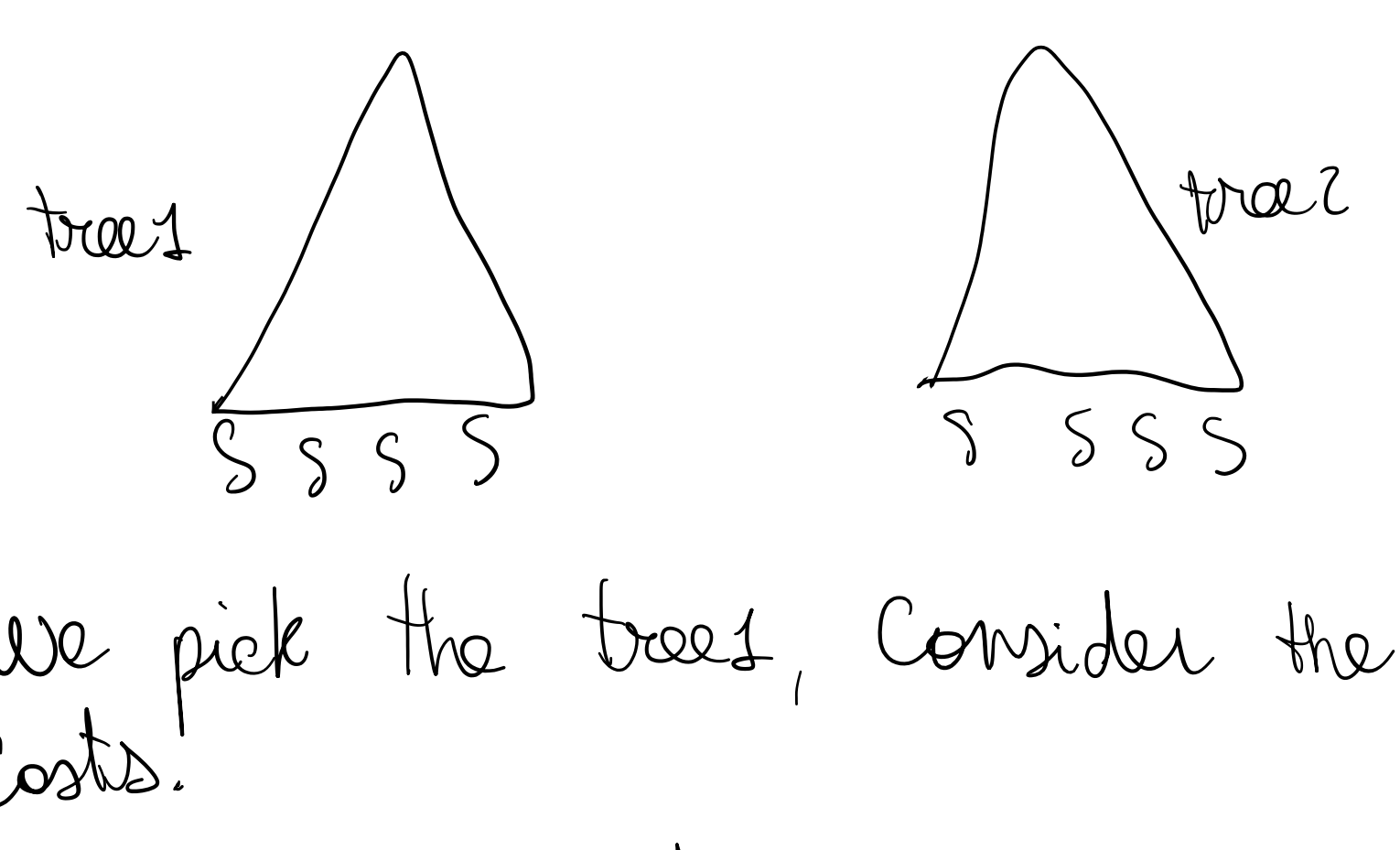
If I consider the pattern tree:

16:55



Consider some parameters.

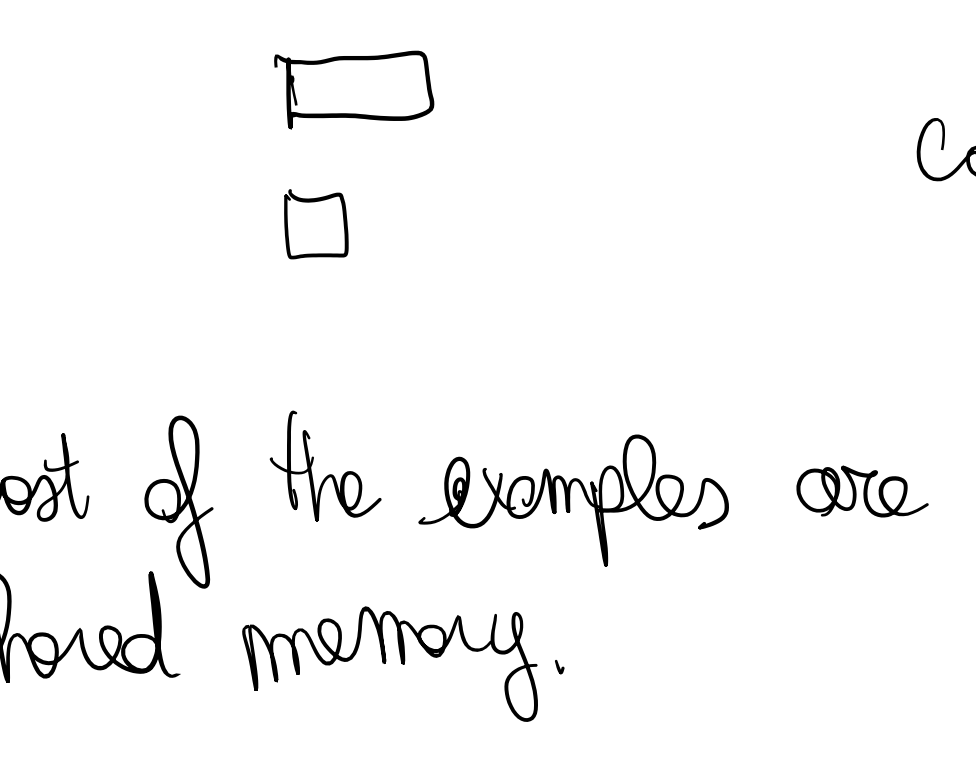
If I have two alternatives for the trees:



We pick the tree1, Consider the communication costs.

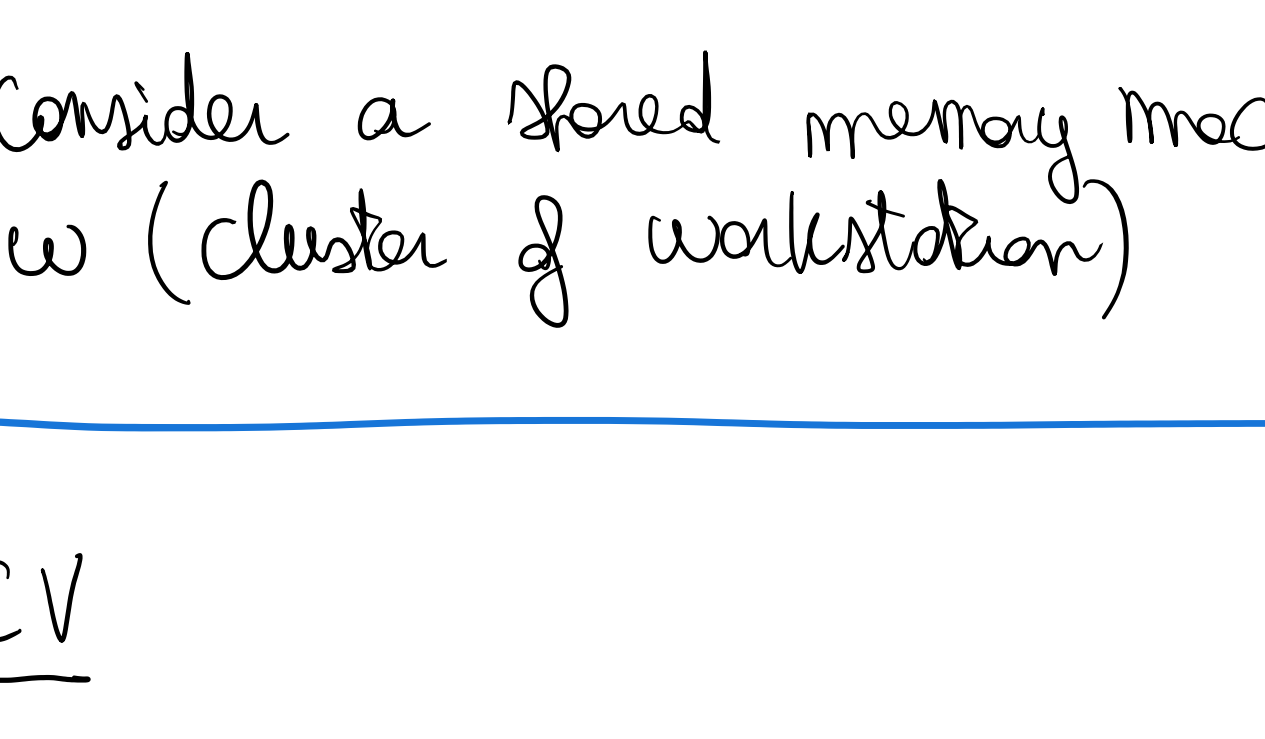
→ the costs can be masked with a proper communication strategy.

If I have a pipeline and I have to communicate



If I have the hardware 16:57

Computation takes an amount of time on communication less time

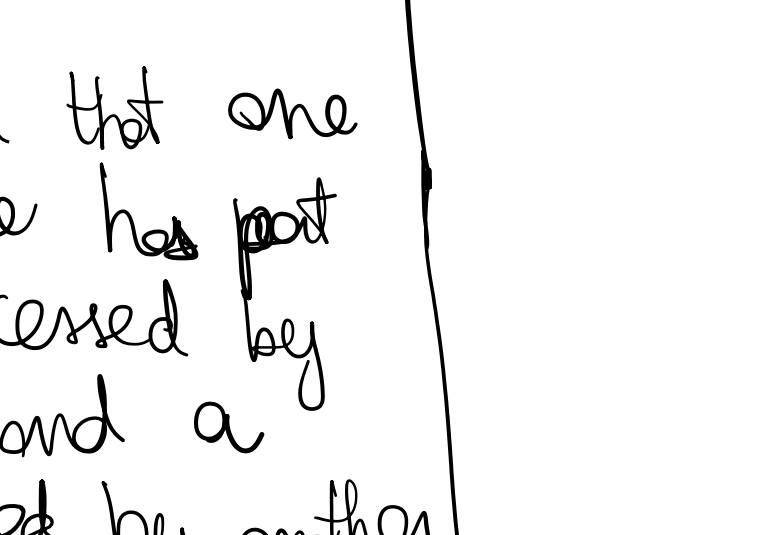


Most of the examples are made taking into account shared memory.

If we consider a shared memory machine or a COW (cluster of workstation)

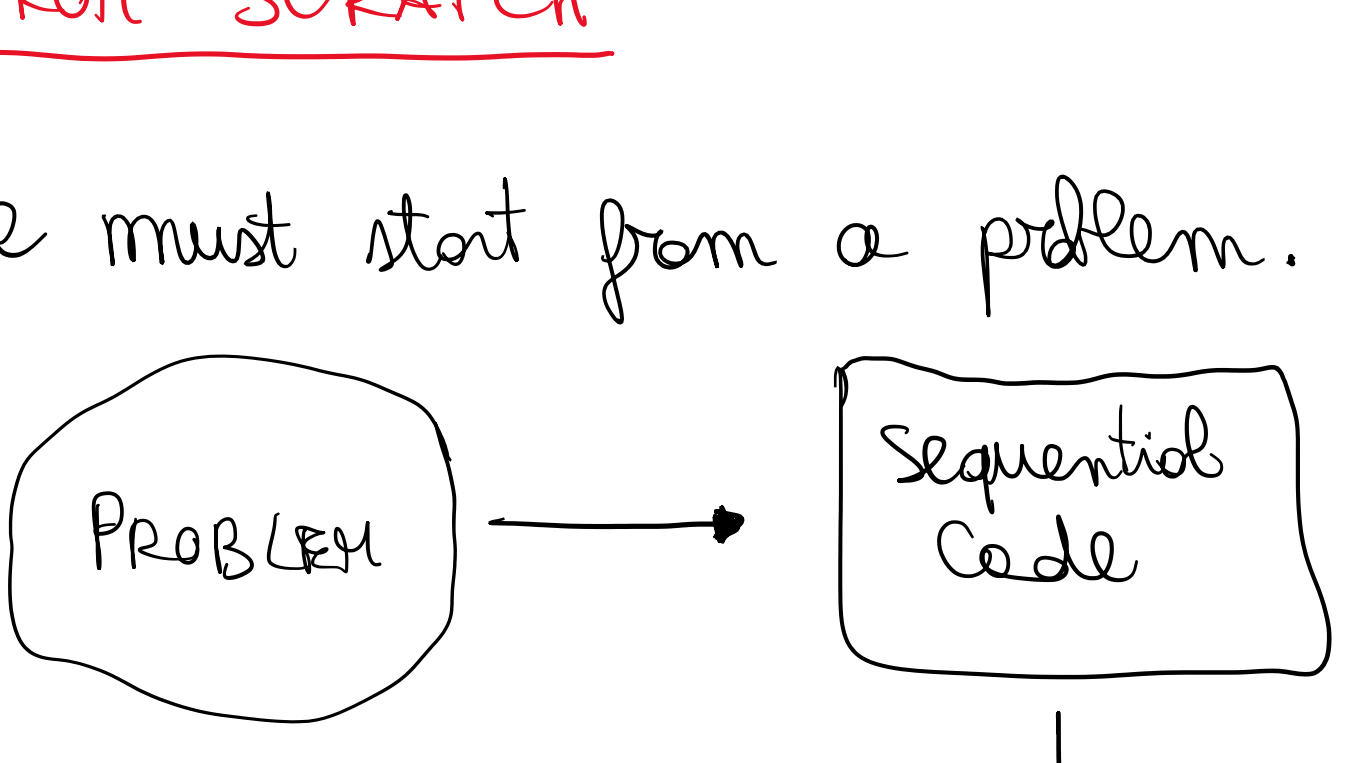
Open CV

Phase 1	Phase 2	Phase 3
• Prototype the application ↓ Something that works more or less but there is something that need to be refined	• Refine the application ↓ Do part of the job that you did before. When you refine you have to do again all the overhead operations. And try to optimize the overhead. Example: Consider codes we want to enhance locality. If it happen that one cache line has part of data accessed by a thread and a part accessed by another thread. (false sharing problem) We can't refine this until we have the real implementation	• Tune the application ↓ Is a minor activity when developing a parallel application

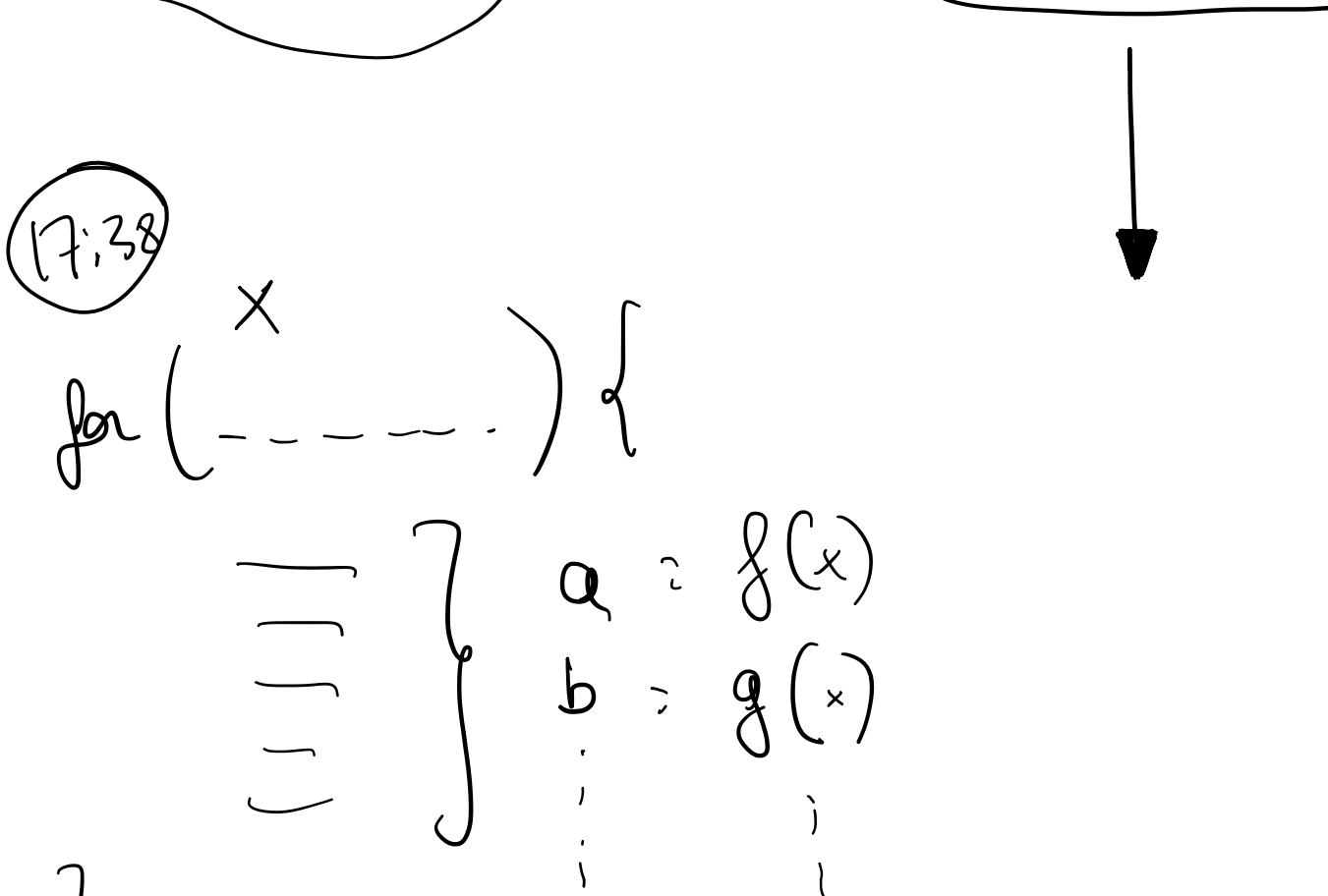


FROM SCRATCH

We must start from a problem.



17:38



Macchina Con GPU

→ Usare solo una gpu

→