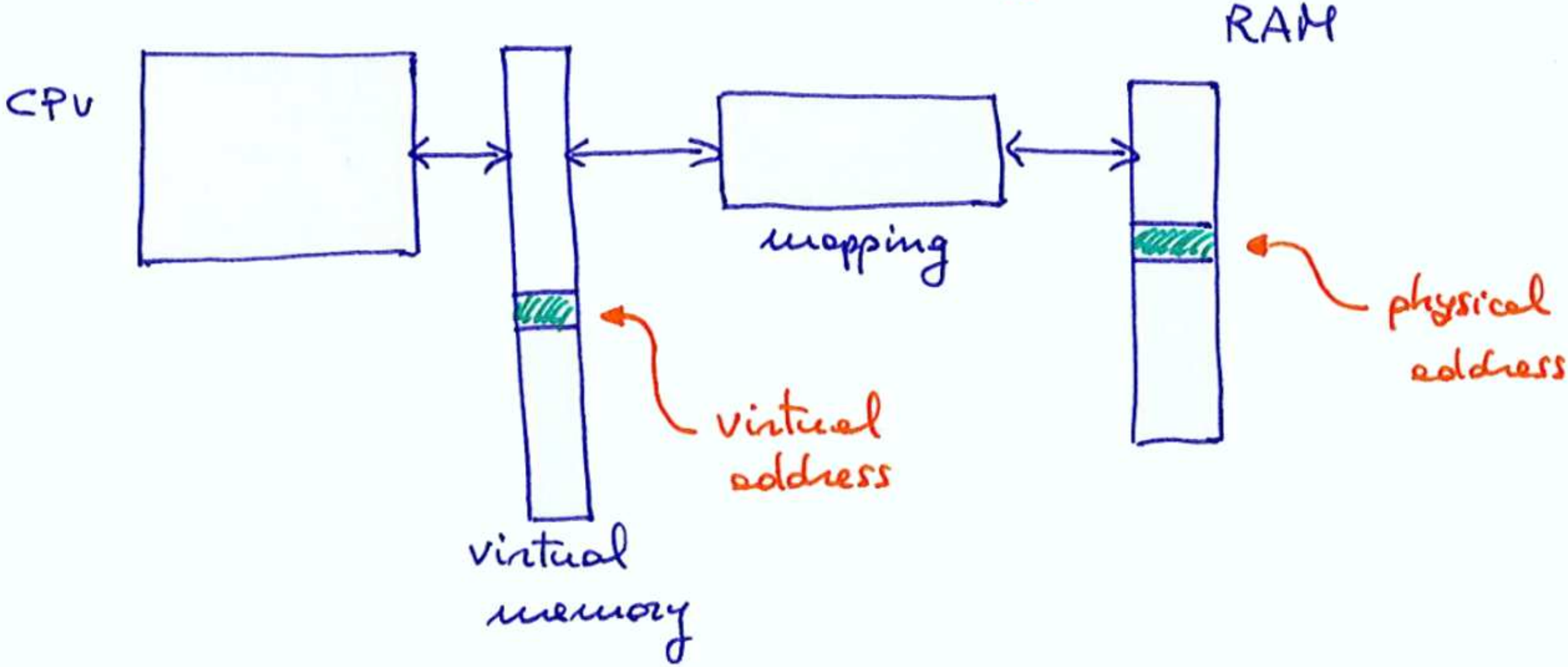
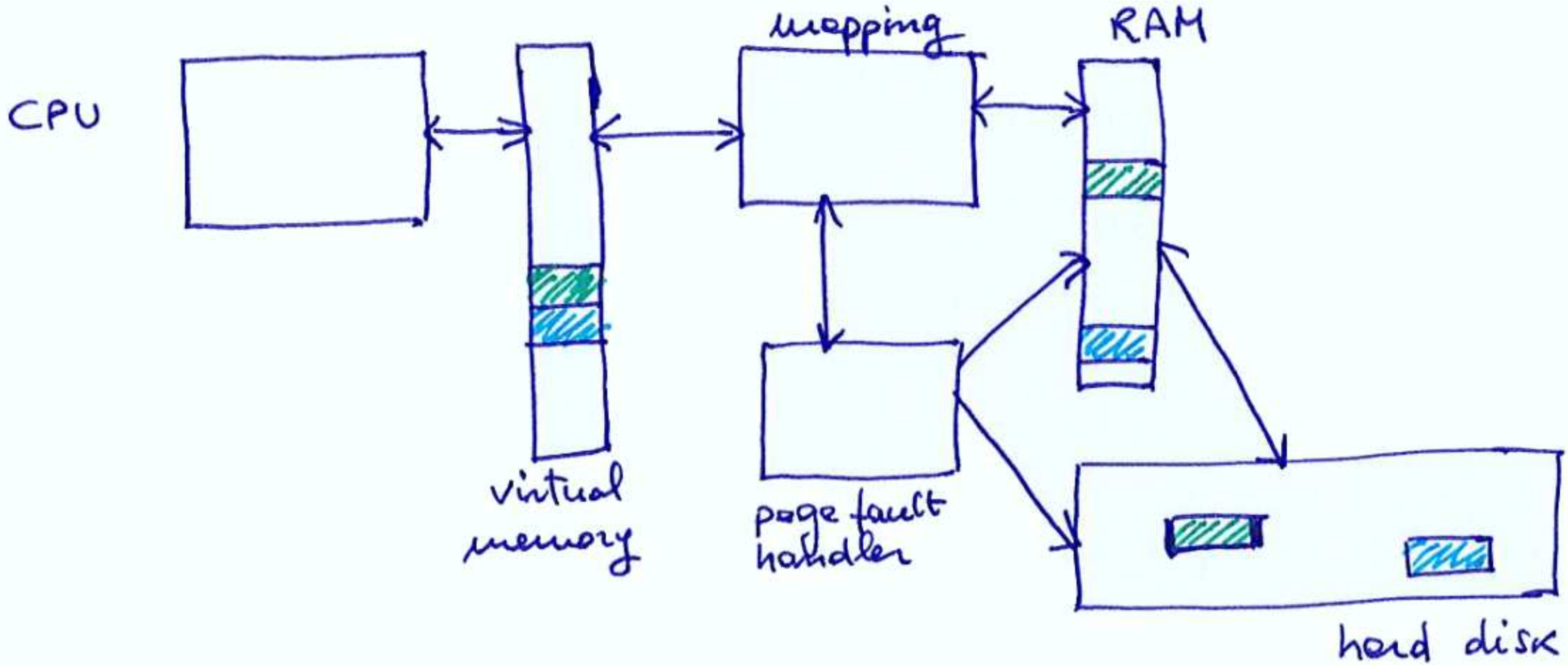
Virtual memory

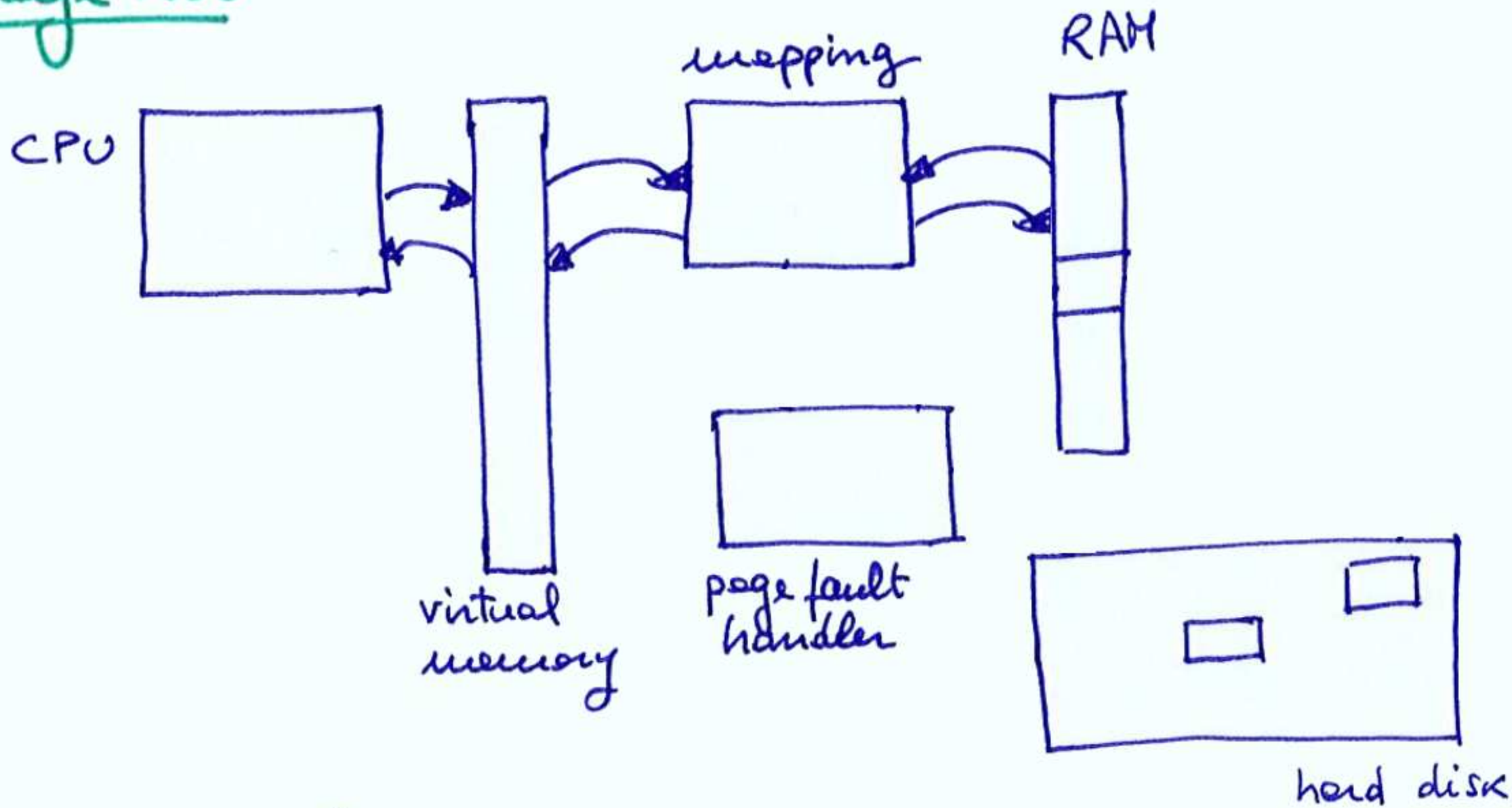


We use the hard disk to keep currently not in use pages:



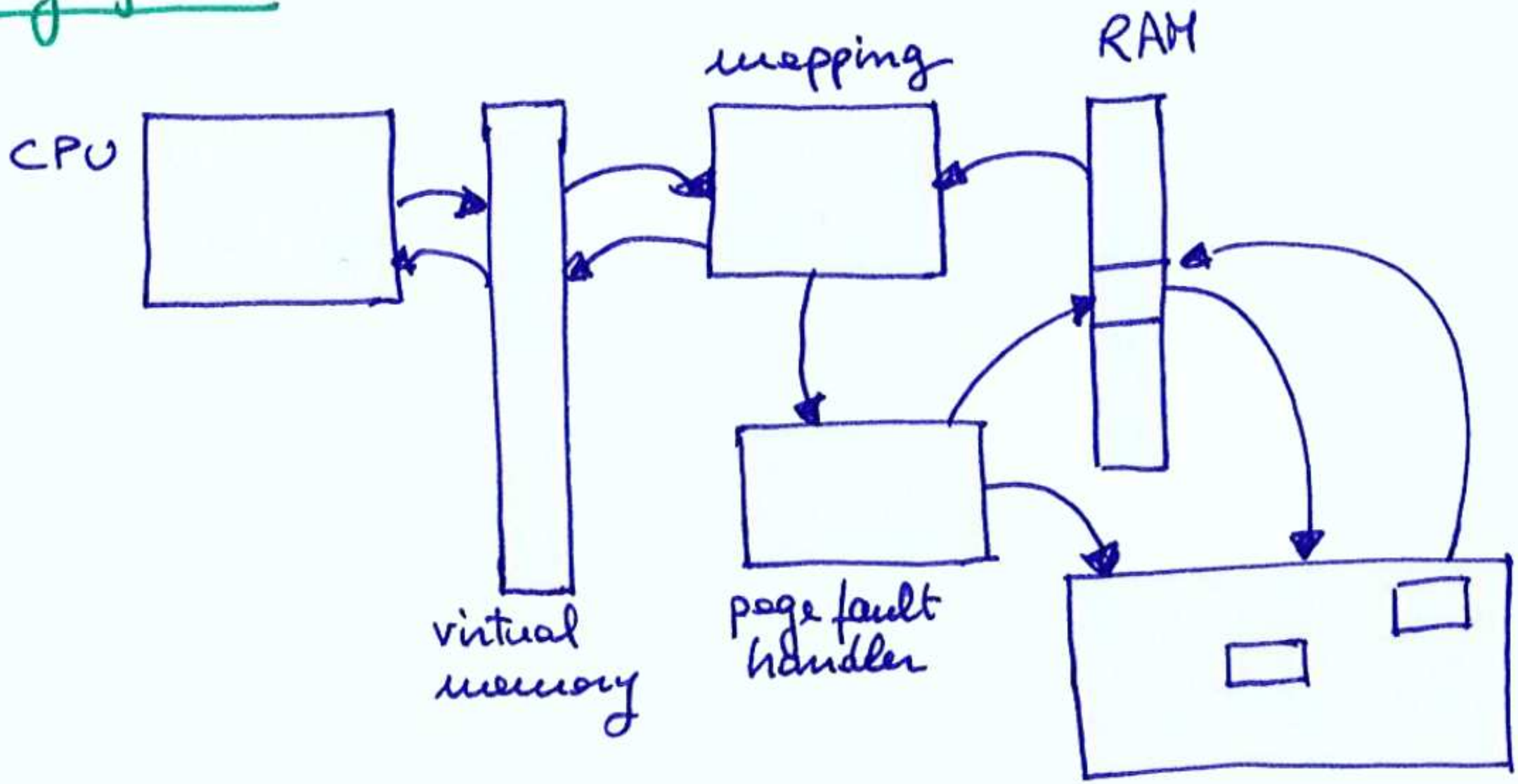
What we have added to the diagram?

Page hit

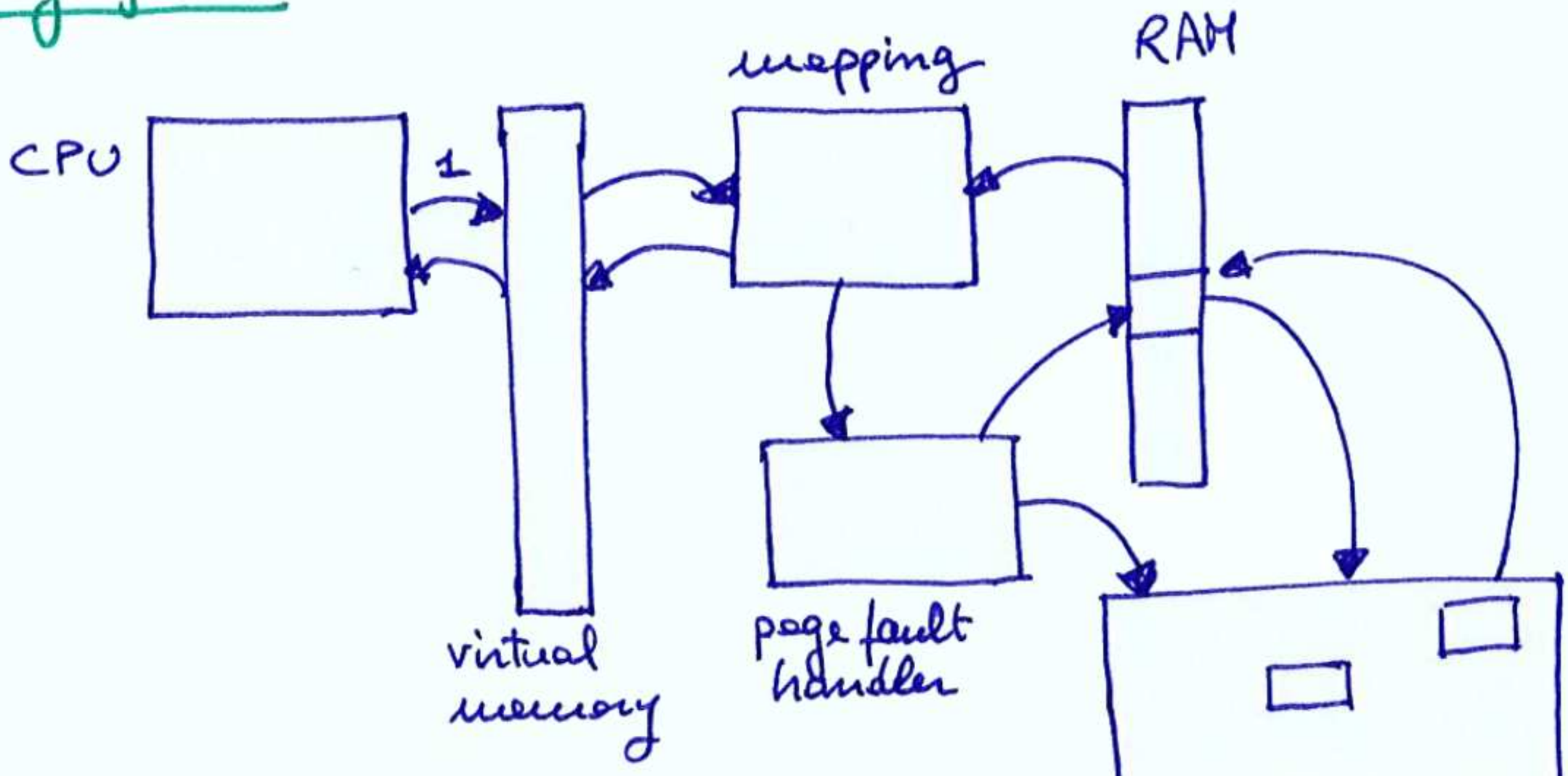


the "mapping" says that the page is in the RAM

Page fault

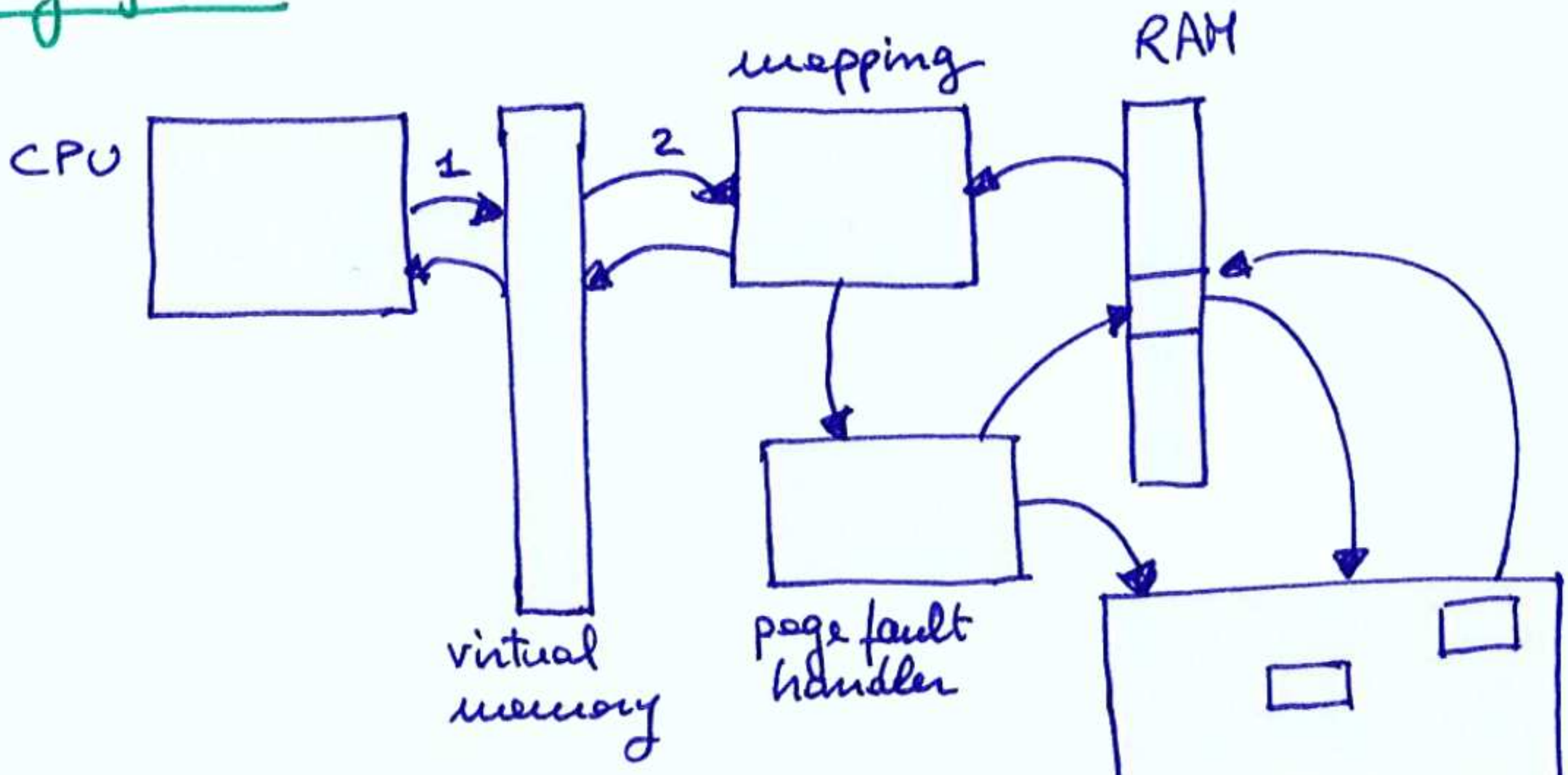


Page fault



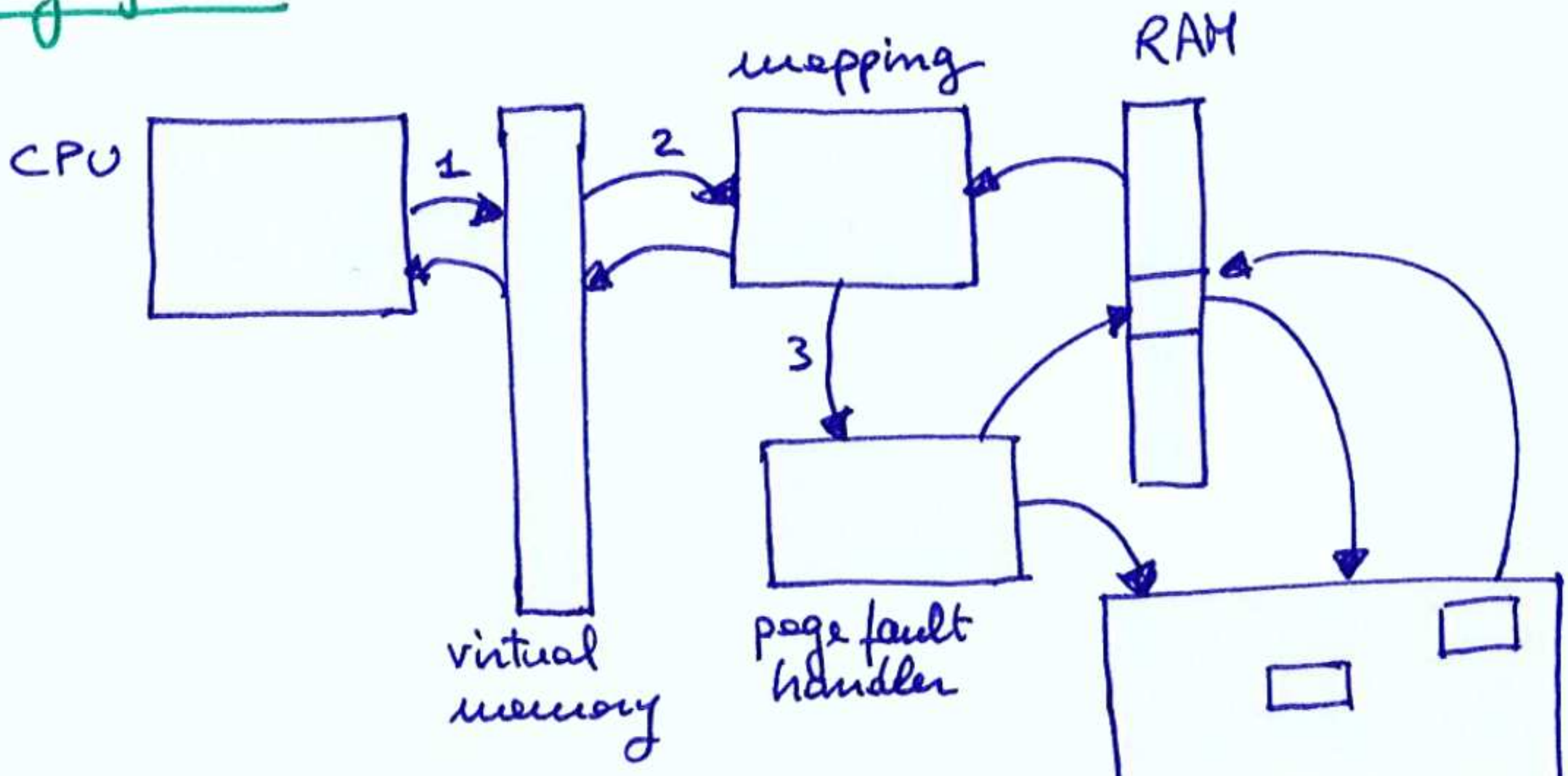
1) the CPU asks for the data pointed by a given virtual address;

Page fault



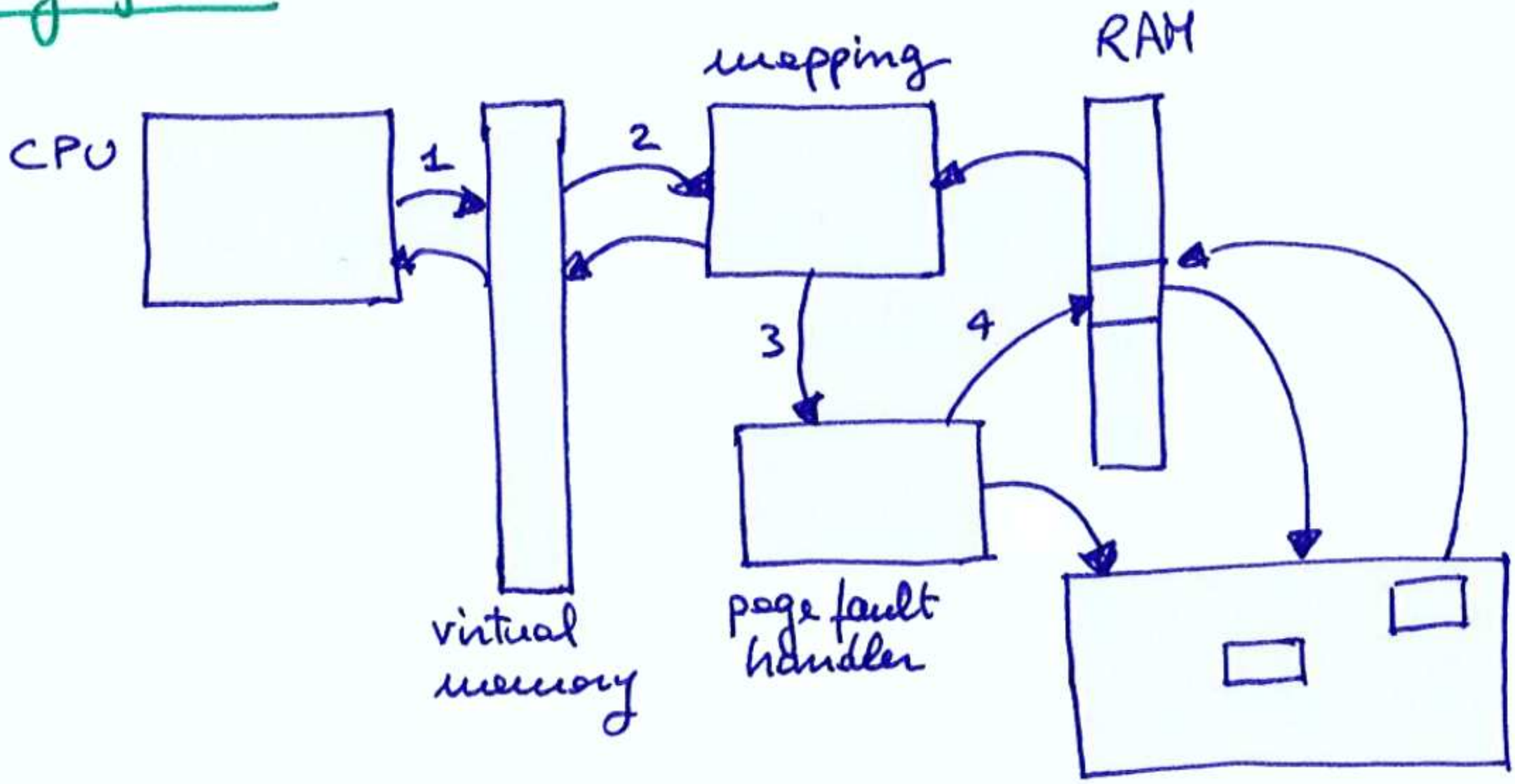
2) the "mapping" is asked to provide the corresponding physical address;

Page fault



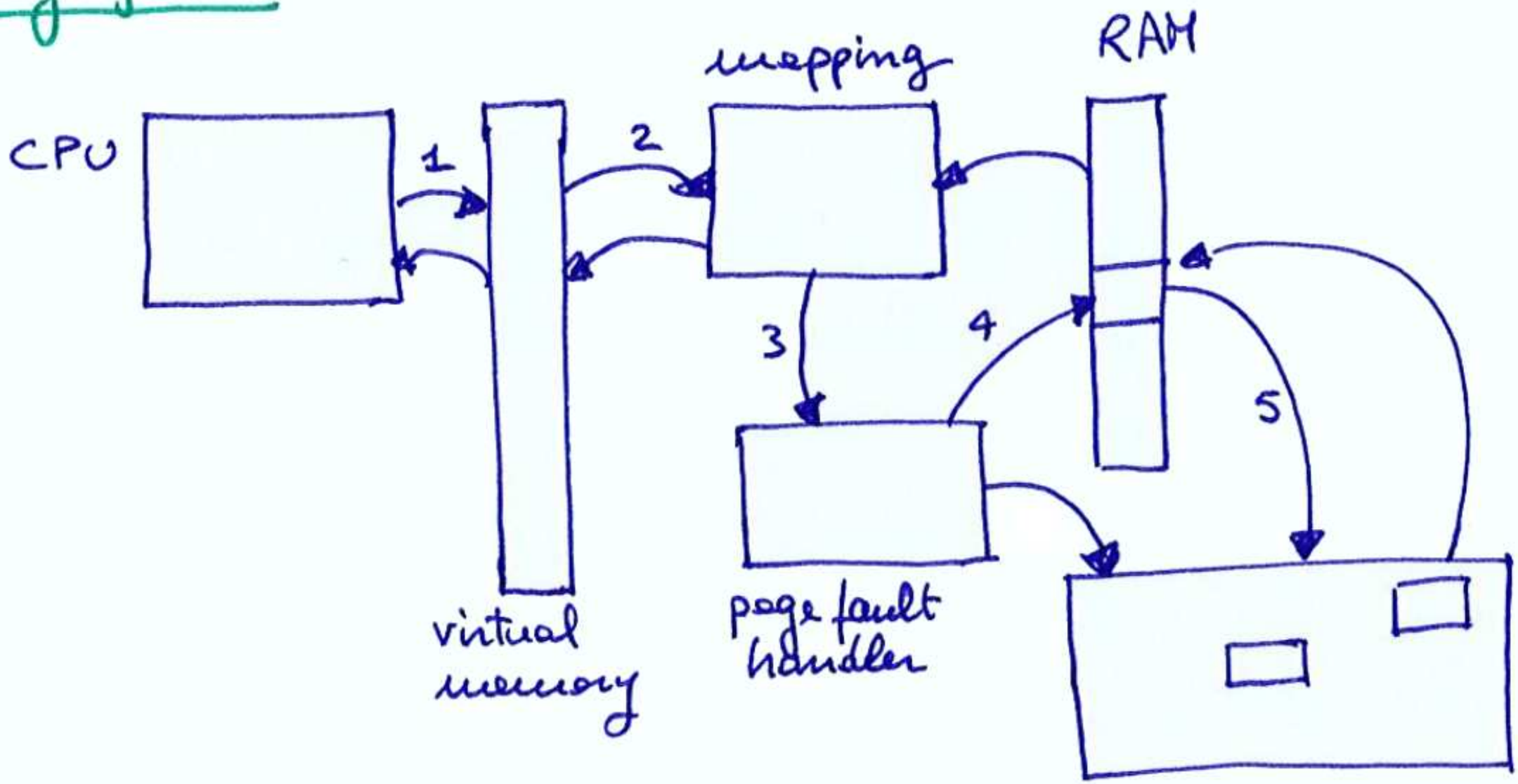
3) since the concerned page is not in the RAM (page fault), this information is send to the page fault handler;

Page fault



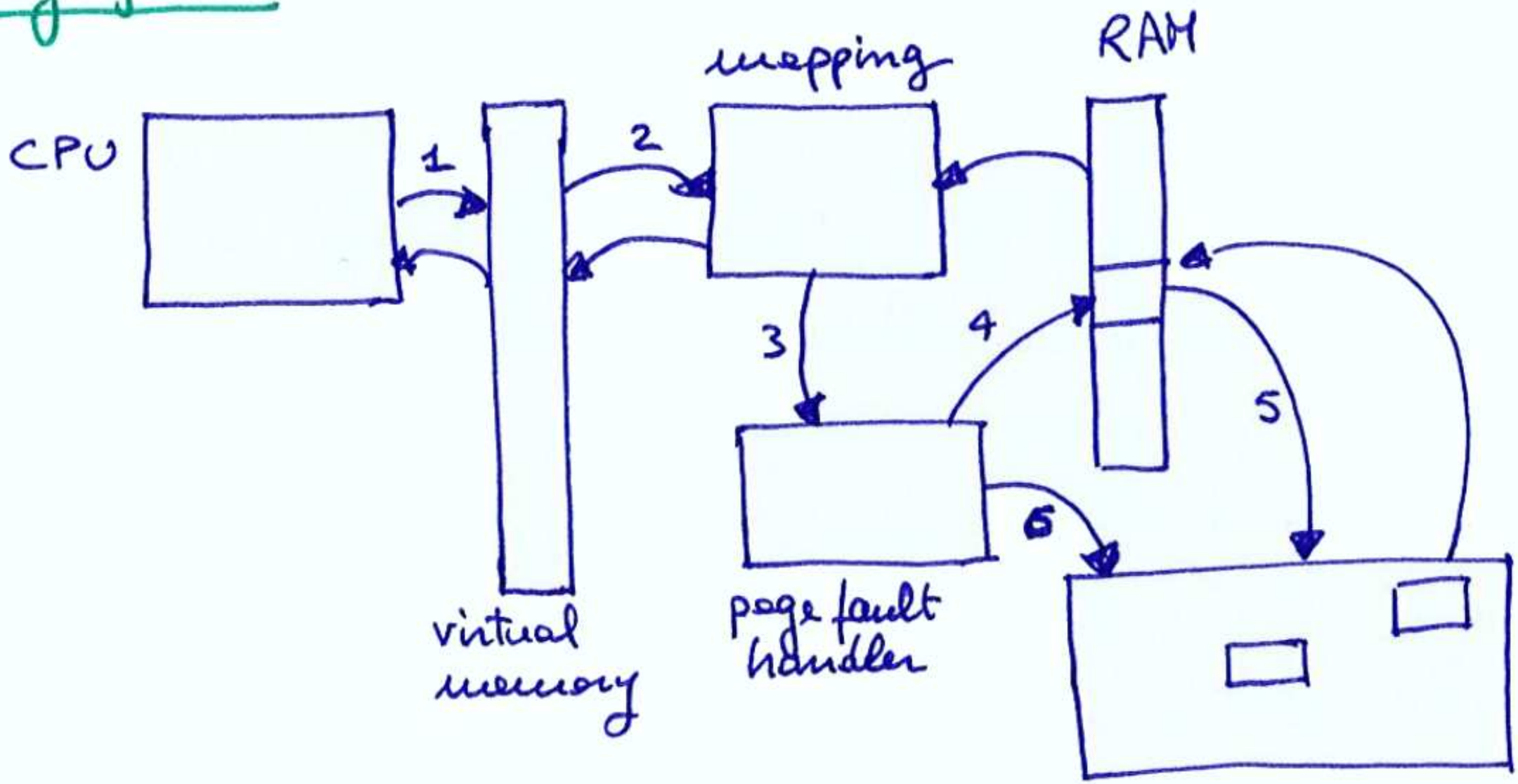
4) the page fault handler verifies whether there is free space in the RAM; if not, it chooses a "victime page";

Page fault



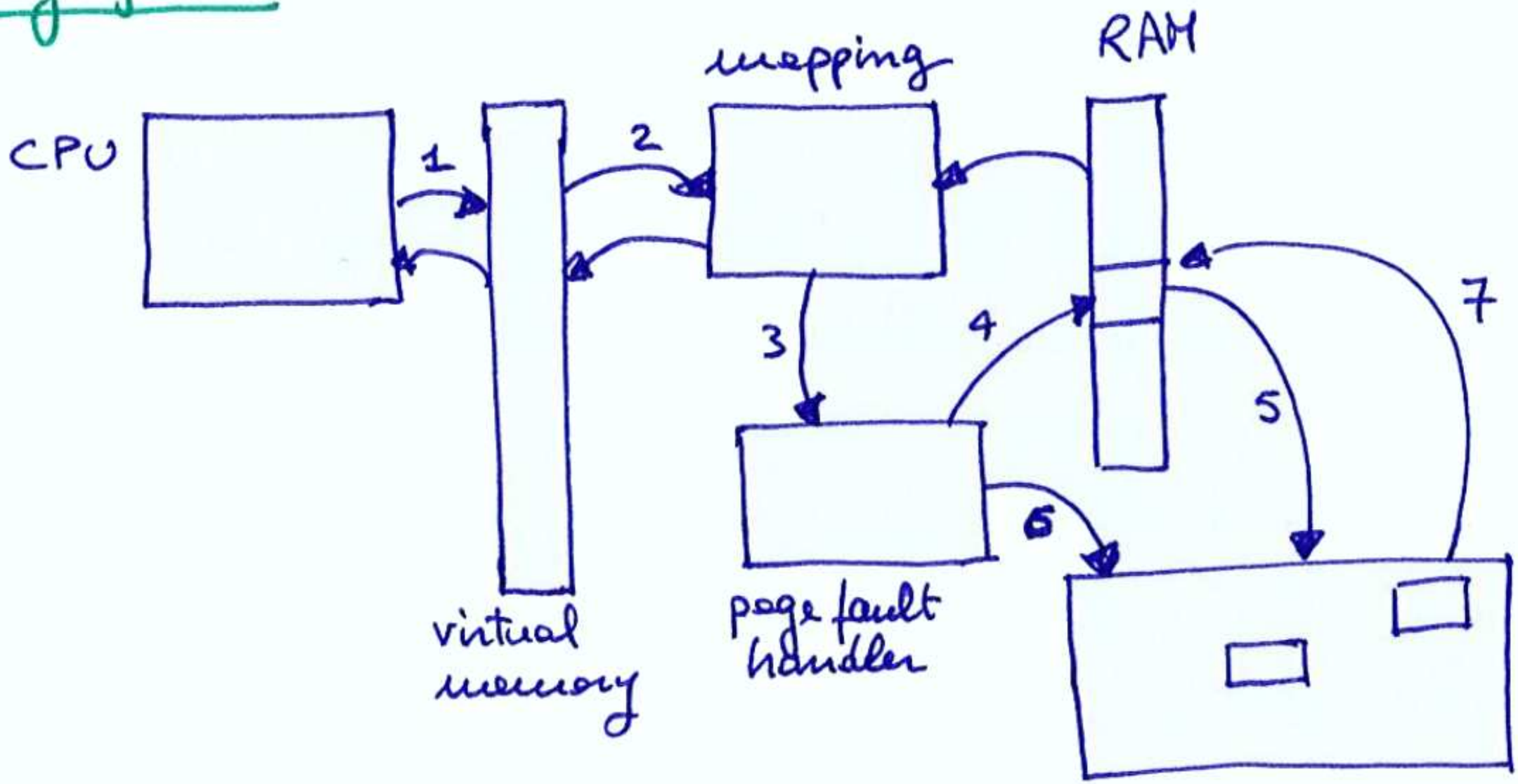
5) if the "victime page" was modified during "the stay" in the RAM, it is copied in the hard disk; otherwise, it is simply deleted;

Page fault



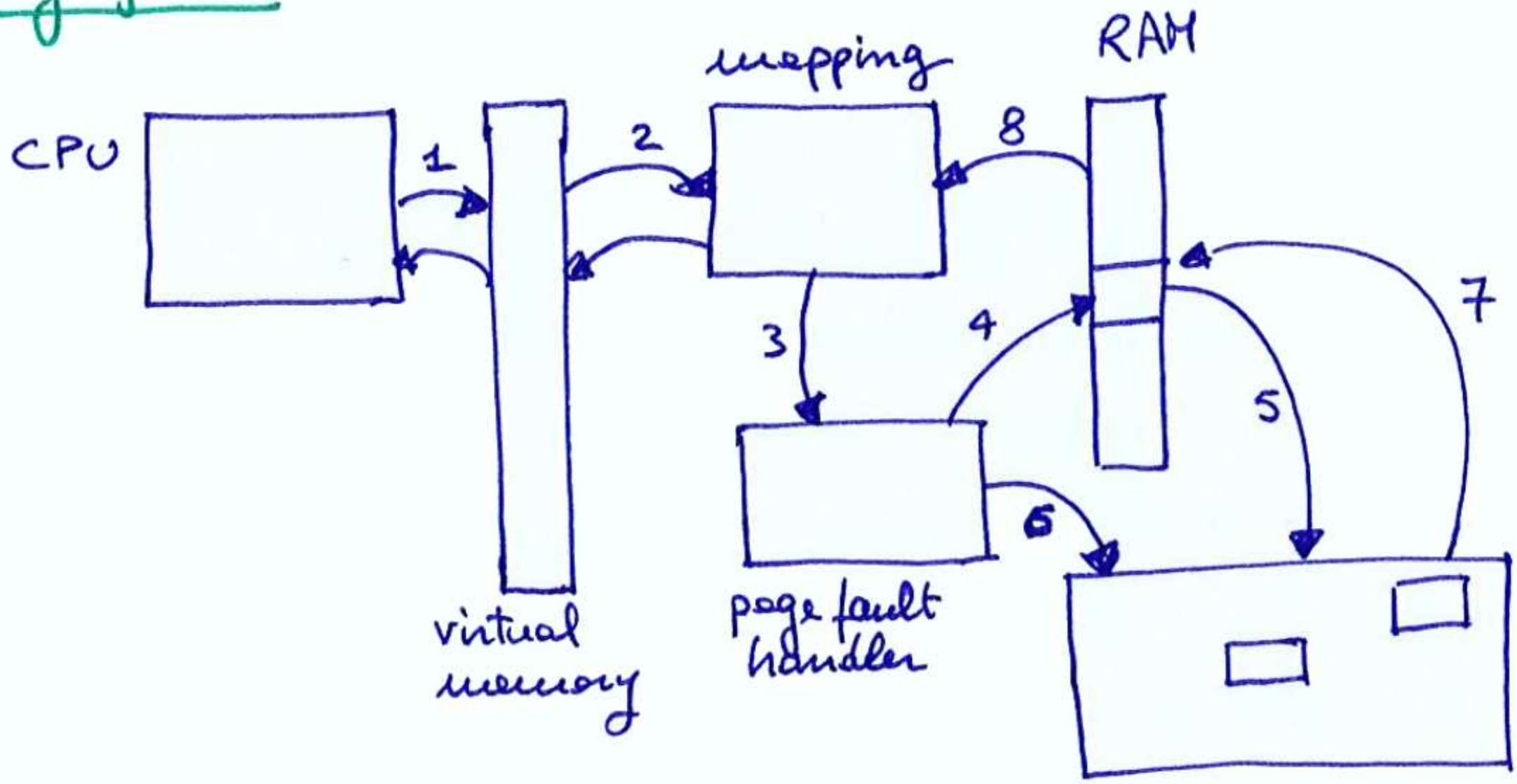
6) the page fault handler identifies the requested page on the hard disk;

Page fault



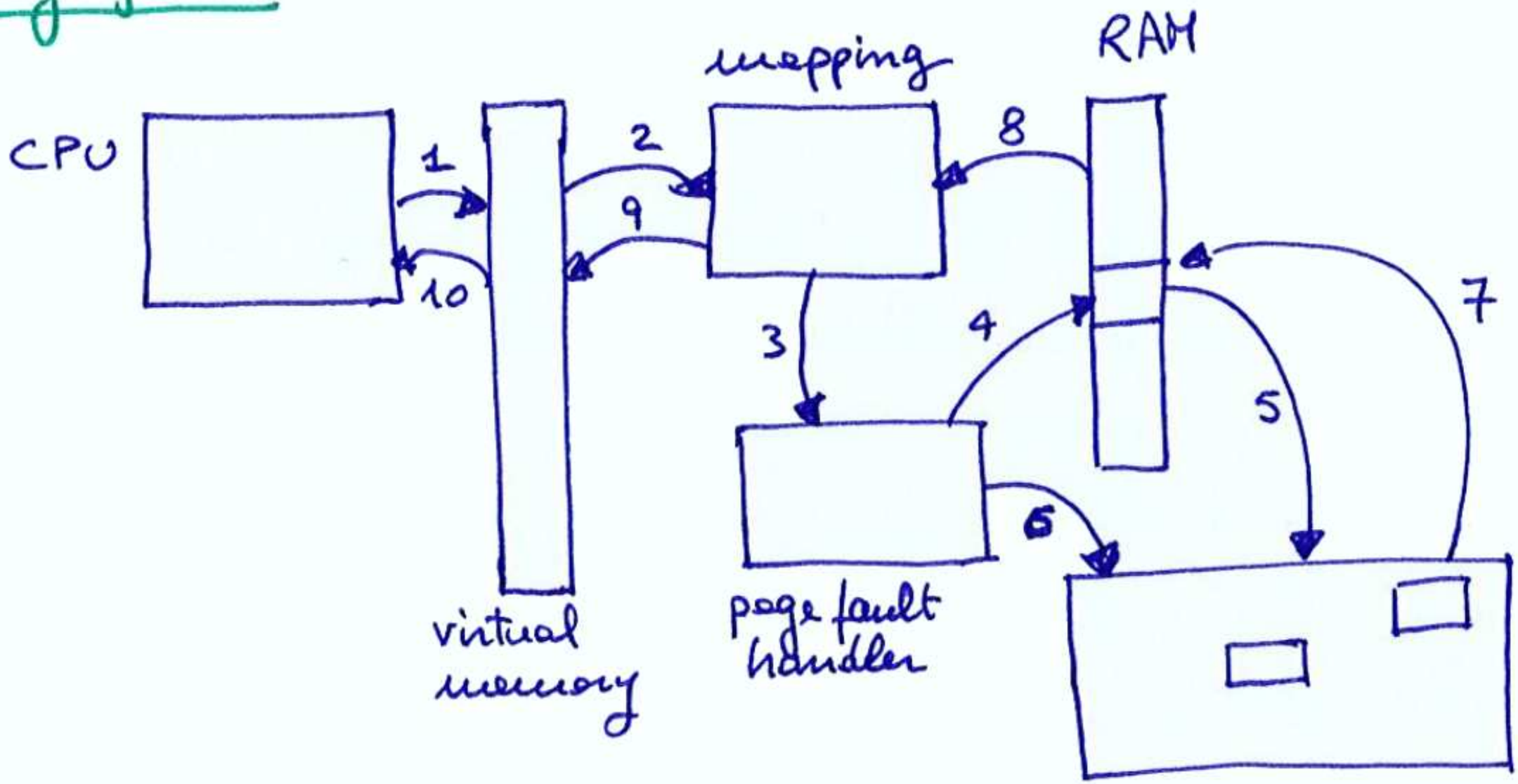
7) the requested page is copied in the RAM;

Page fault



8) the "mapping" is updated with the new physical address;

Page fault



9-10) The requested data "travel" back to the CPU.

Page tables

valid	virtual page address	dirty bit	write	exe	physical page address
1	---	0	1	1	---
0	---	0	0	0	/

How to associate the virtual page address with the physical page address ?

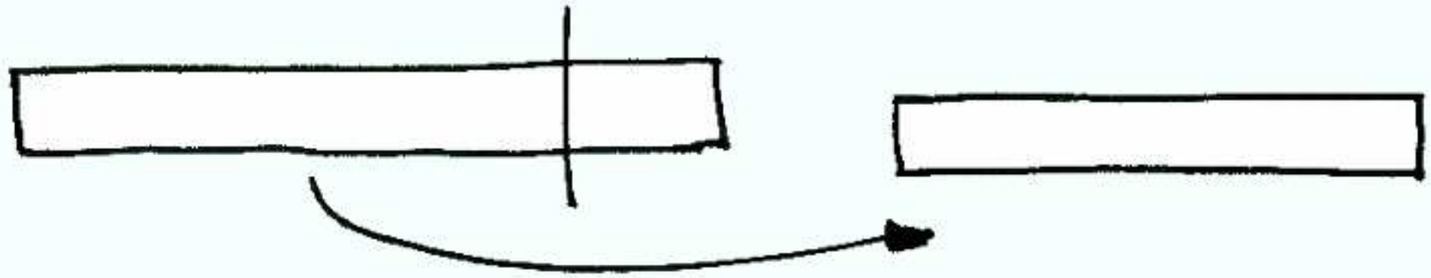
Let's construct a page table

- size of virtual address : 14 bits
- size of physical address : 12 bits
- page size : 64 bits

Page tables

Solution 1

We know that more bits are in the virtual address.

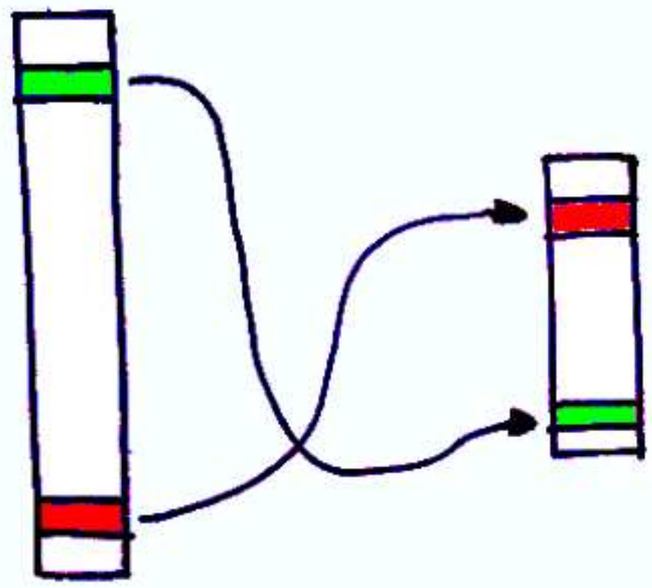


We simply cut off the extra bits to form the physical address.

Page tables

Solution 2

We can associate any physical address to a given virtual address:

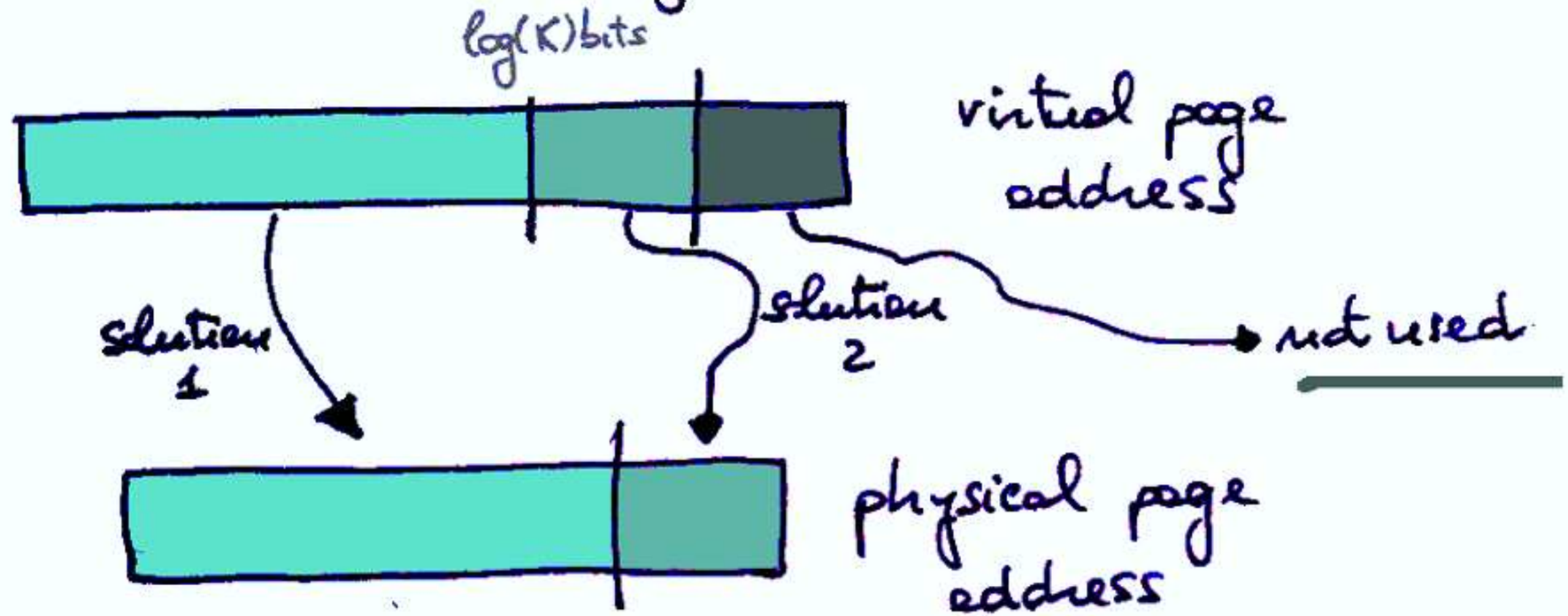


High flexibility

Computationally expensive

Page tables

Best solution: K-way associative tables



Best trade-off between the previous two solutions.