

# Dirty COW

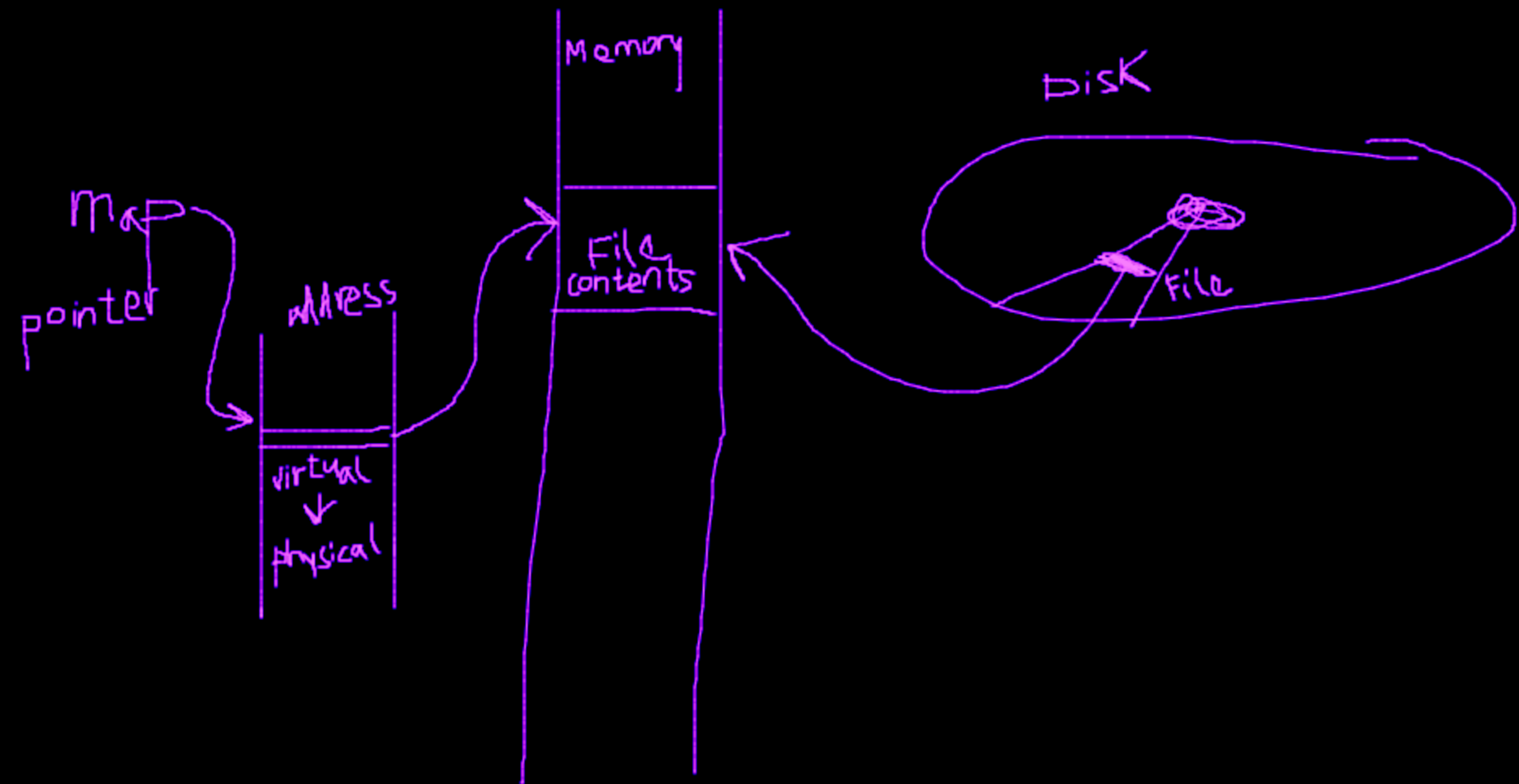
Race condition vulnerability in Linux  
memory management kernel code

# Dirty COW?

- Dirty = page in memory has been modified
- COW = copy-on-write...
- Patched and fixed in October 2016 in majority of Linux distributions and Linux kernel
- But... Android...

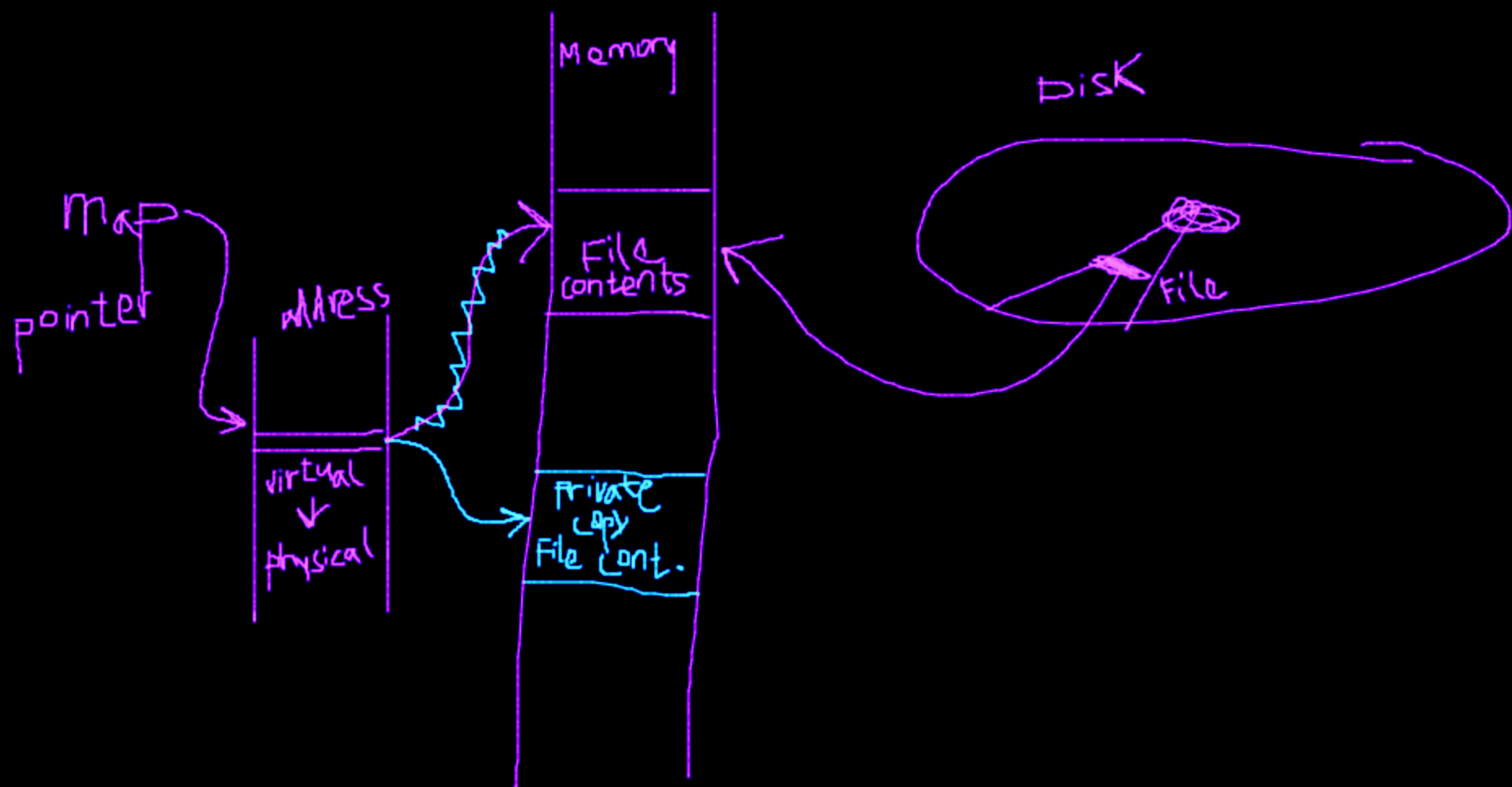
# Copy-on-write?

- User can directly map a file into memory using a system call `mmap()`
- User changes file in the memory, the page gets marked "dirty" and the operating system stores changes back to the disk upon getting rid of the page



# So what?

- User can also map read-only files from the disk to the memory
- We can write into the file but the kernel will provide us with a private copy (**hence, copy-on-write**) with the same virtual address but different physical address
- If we made changed to this private page and the kernel wants to get rid of it, nothing happens because this page is not directly mapped to the read-only file



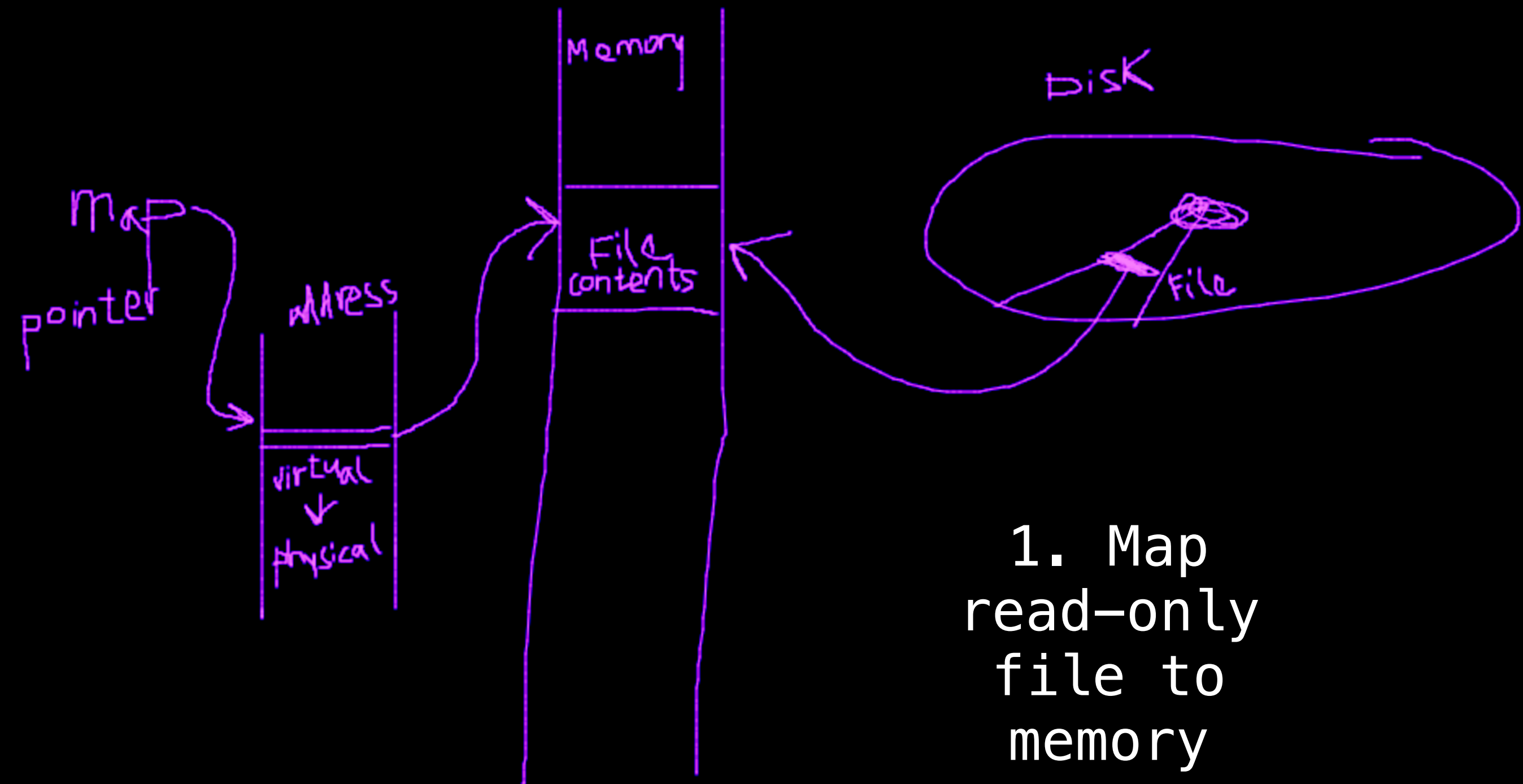
# Race condition

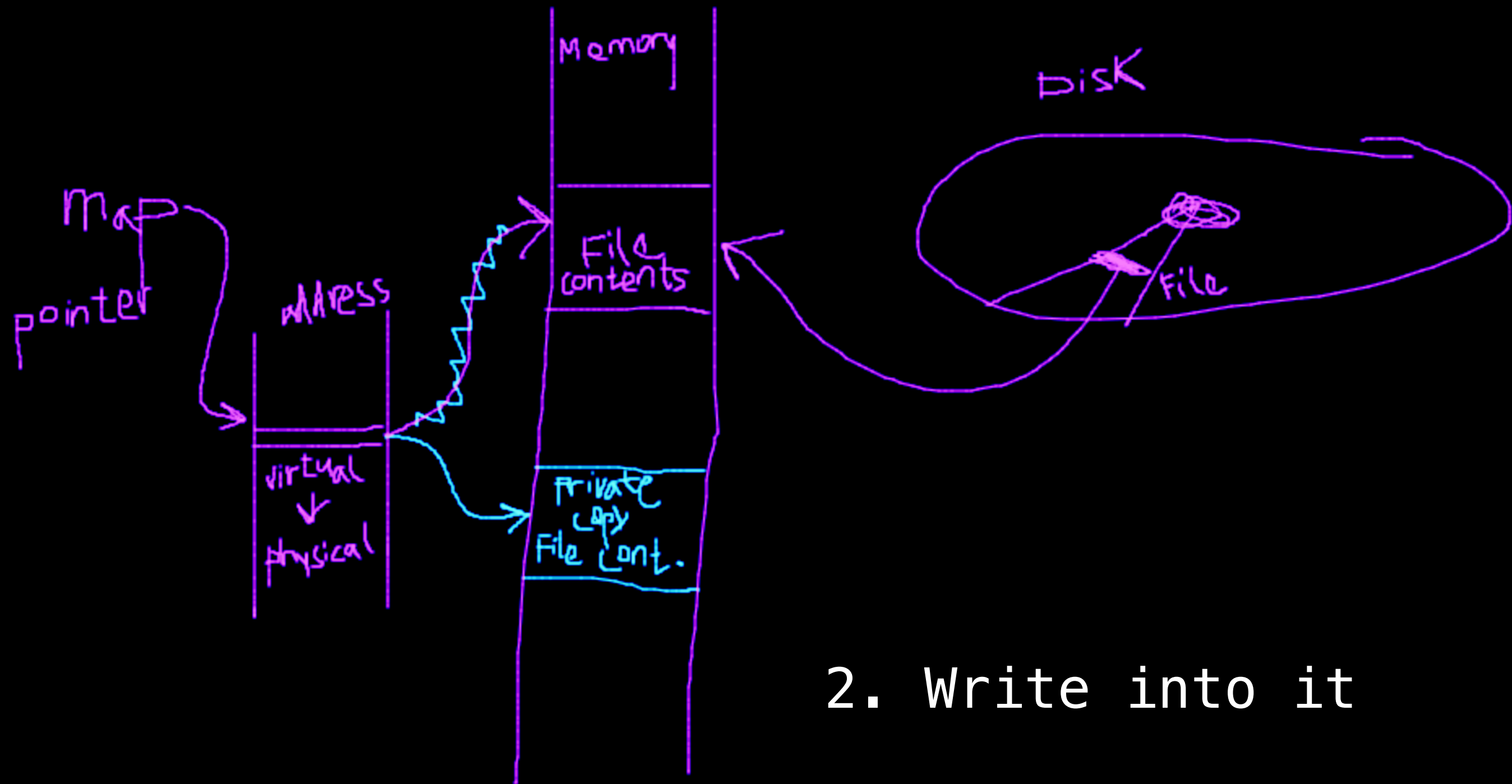
- We can also utilize a system call **madvise()** to let the kernel know what we intend to do with the memory mapping
- This system call aids the page replacement algorithm make better decisions
- **madvise(MADV\_DONTNEED)**: tell kernel the memory map is no longer needed (get rid off the page in the future)
- If we call **madvise()** on a read-only mapping, it first deallocates the private copy, then changes the mapping to the original mapping and then deallocates it

# Two threads

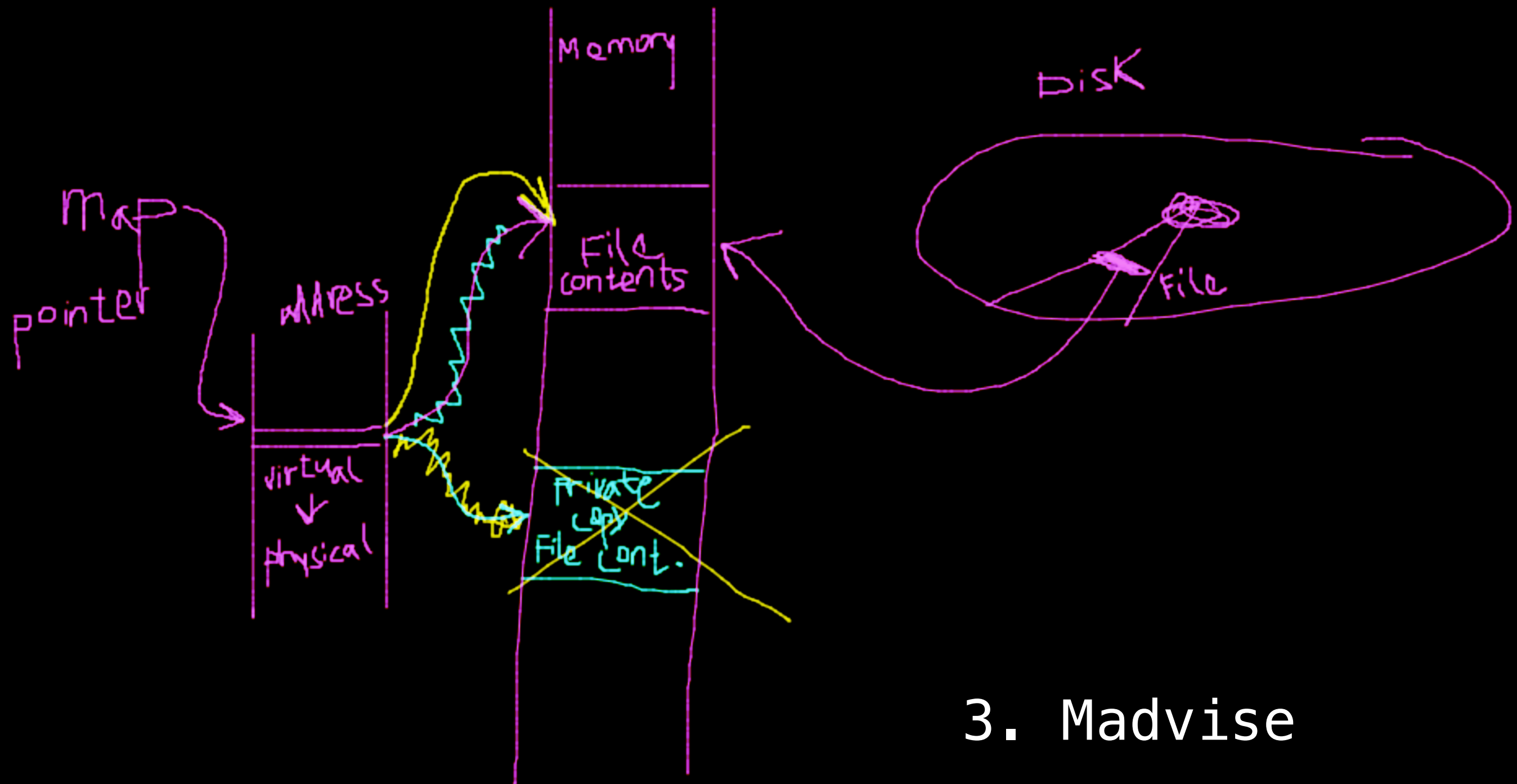
- Map a read-only file to the memory
- Create 2 threads:
  - **Write thread:** constantly writes into the mapped memory
  - **Madvise thread:** constantly tells the kernel the mapped memory will not be needed in the future







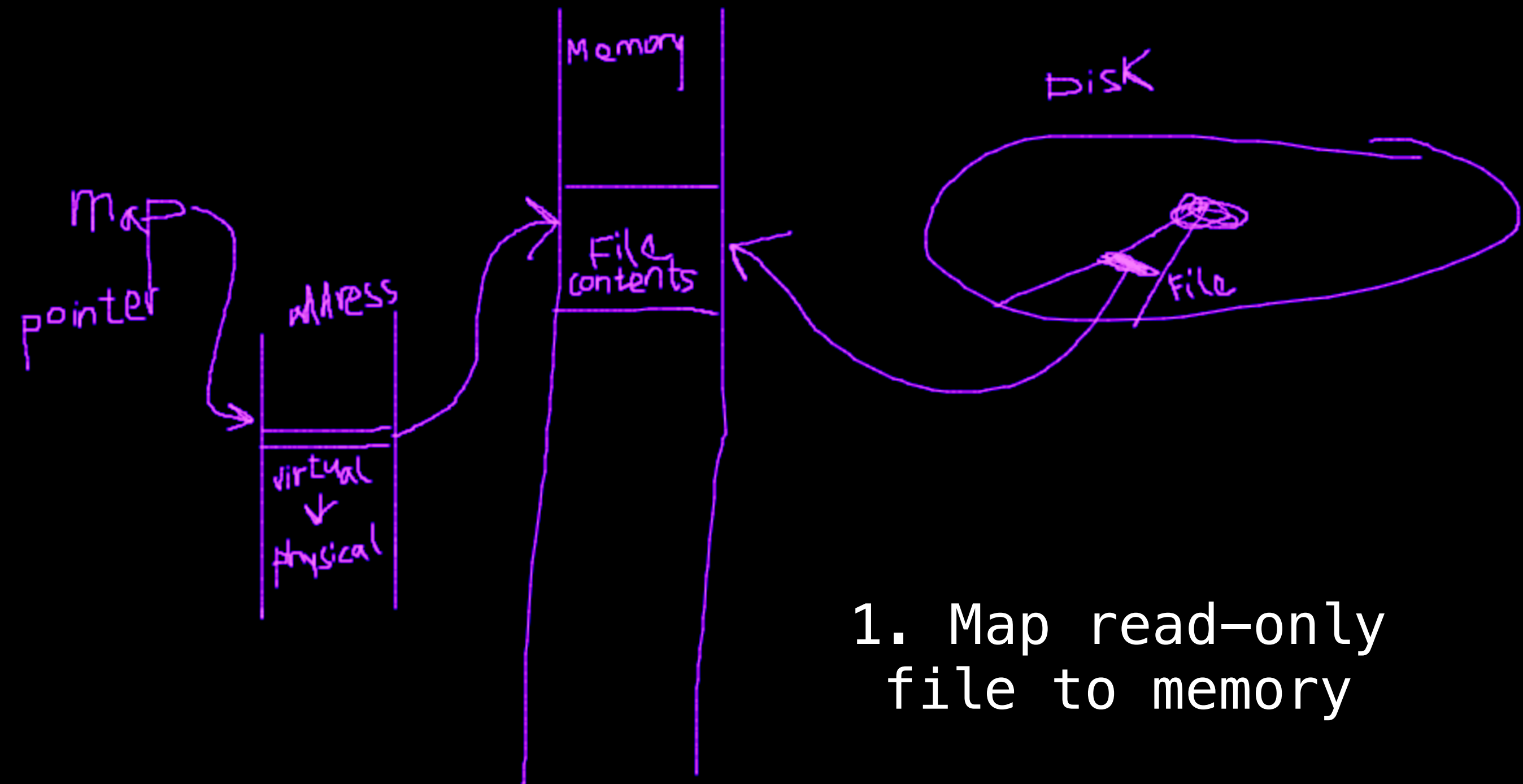
2. Write into it

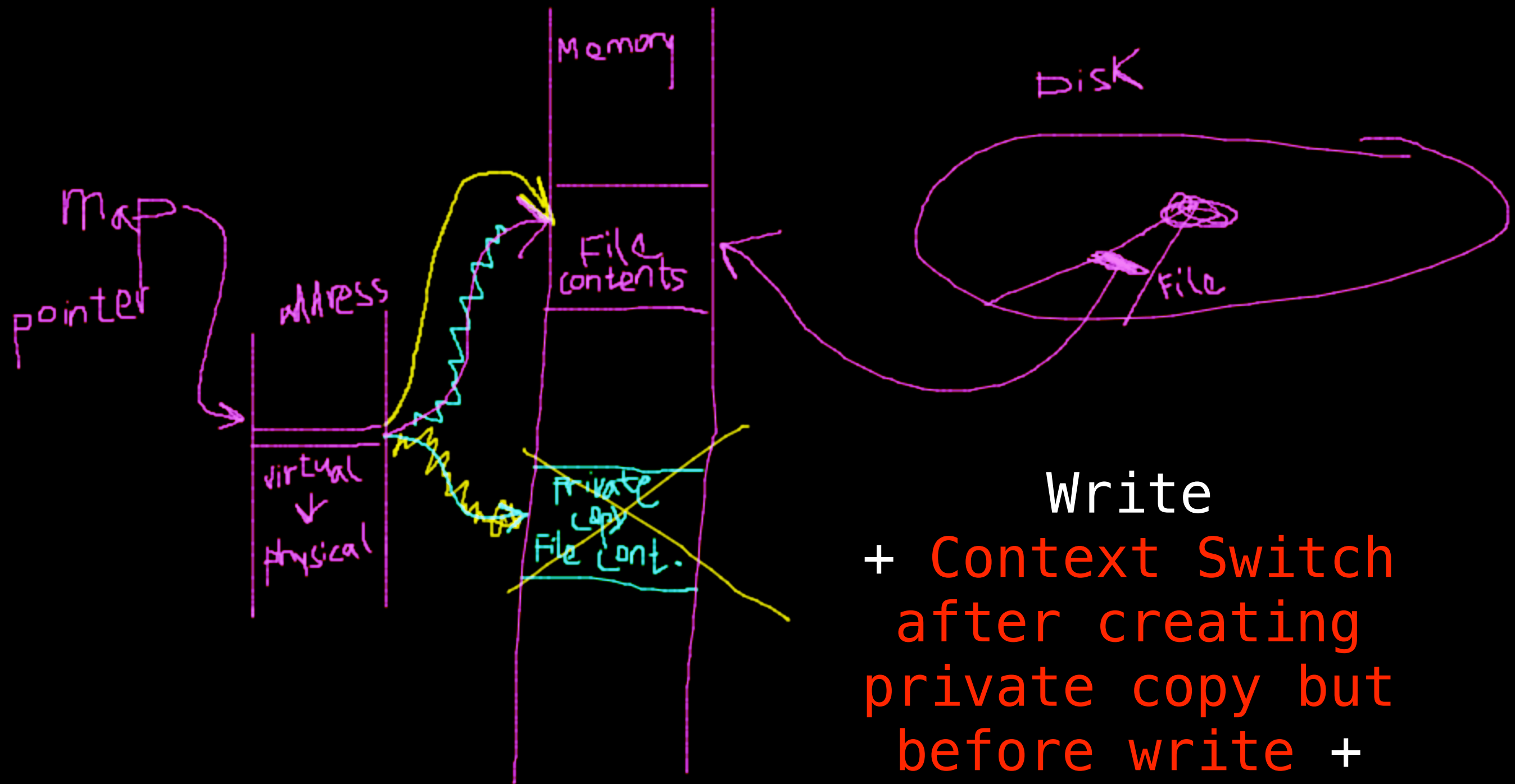


### 3. Madvise

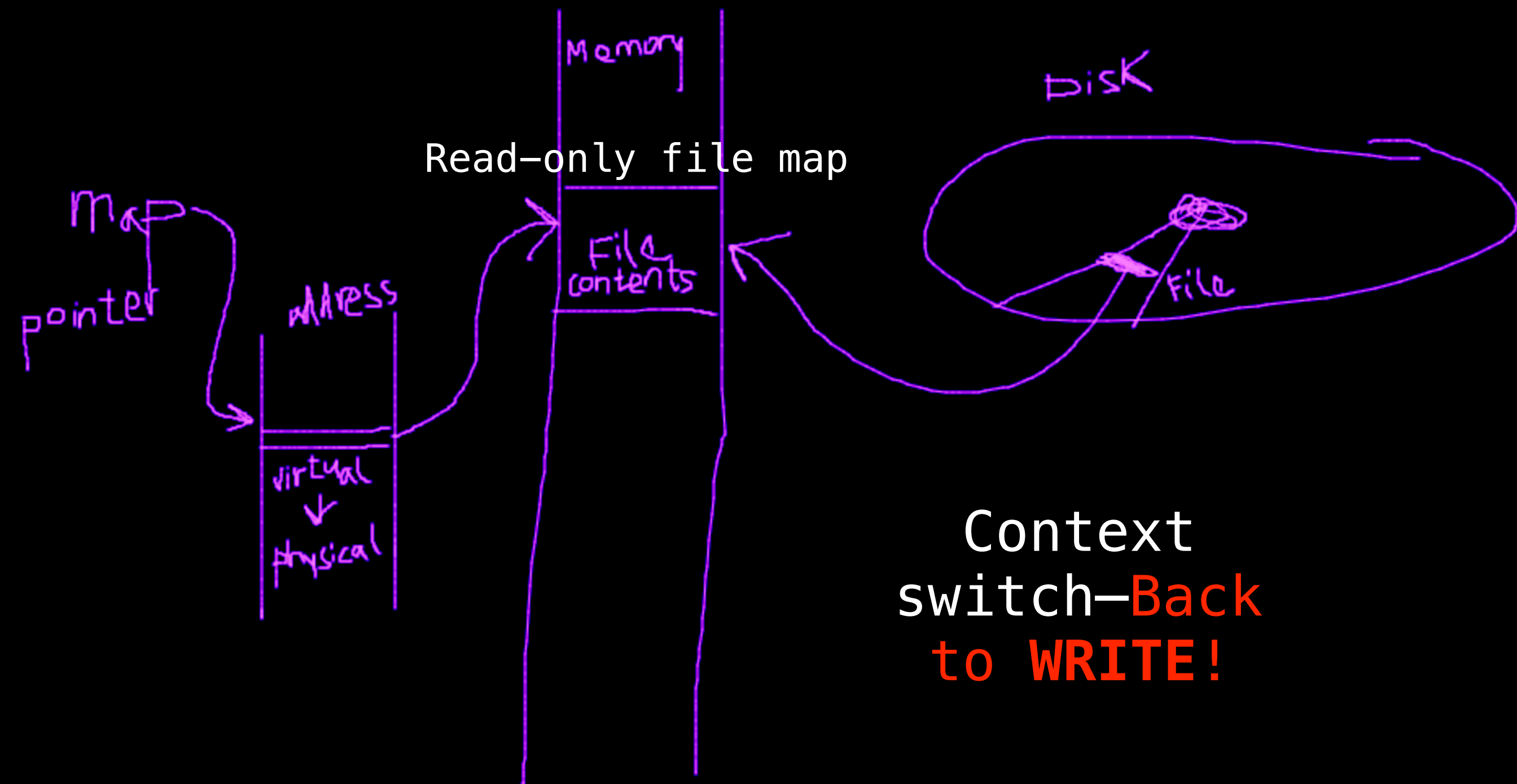
# So what's the problem?

- Well, since the `write()` and `advise()` operations are not atomic, we can get a race condition where `advise()` gets executed after we create a private copy and we end up writing into the original mapped memory which should have been read-only
- Later, the operating file sees the dirty flag on the page and writes changes to the disk
- Effectively, we eliminate the copy-on-write operation and we write into the original read-only file





Write  
+ Context Switch  
after creating  
private copy but  
before write +  
Madvise



“Boom.”

–*Jakub Hladik*



# How to carry out the attack?

- Need access to the device as a regular user and execute code which runs the two racing threads
- Malicious software that the target willingly downloads (including Android apps)
- Other attacks that will get us into the device as a regular user

# Demo

[https://youtu.be/\\_e6FU\\_NaM9U](https://youtu.be/_e6FU_NaM9U)