Memory mapping

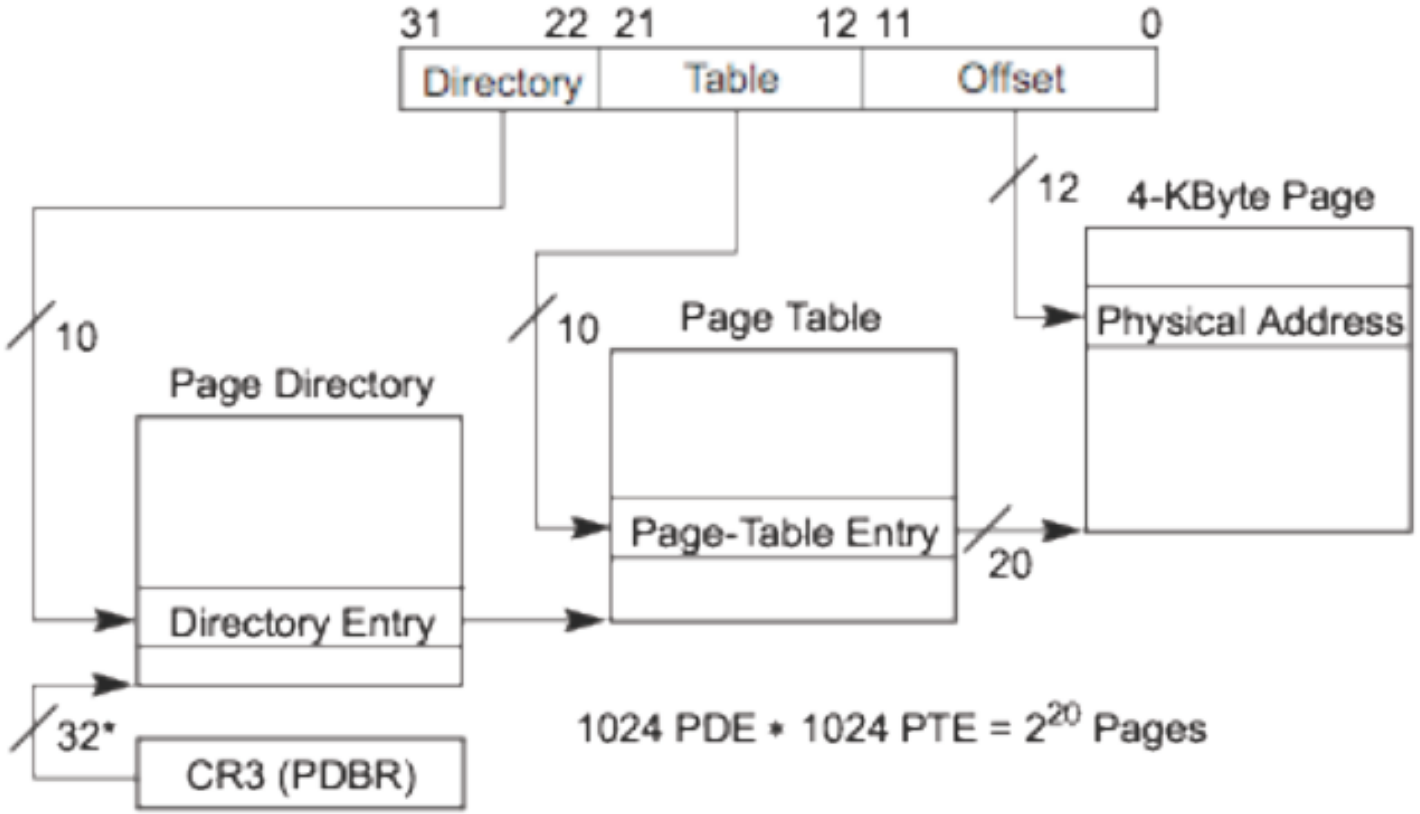
DEMO Time

- Hacking the virtual memory system of Linux
  - A journey through the page directories and tables of Linux kernel 32bits

By Renato Mancuso

Two level page table hierarchy

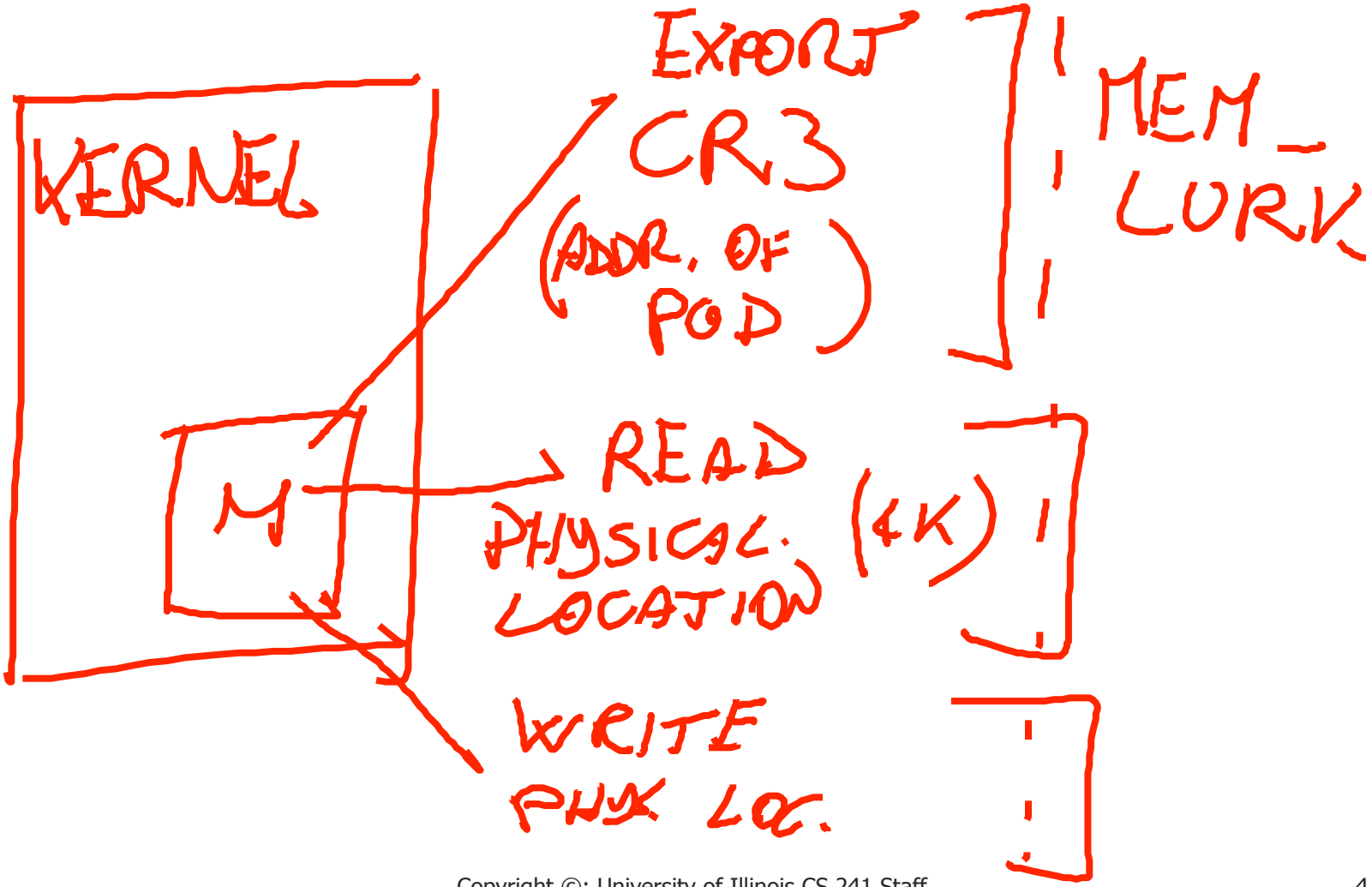
virtual address (32 bits)➔ addresses a byte in VM



addresses a byte in Physical Mem.

*32 bits aligned onto a 4-KByte boundary.

OVERALL STRUCTURE



TRANSLATION

TARGET: 0x0804A01C

OFFSET

PID: 7840

CR3: 0x25E13000

0x0804A01C

↳ 0x20 OFFSET → 0x80

PT ADDRESS

PT: 0xB18430 67

ADDR. FLAGS

PT OFFSET: 0x0A4 ×4 → 0x128

PAGE ADDR.: 0xF3B20067

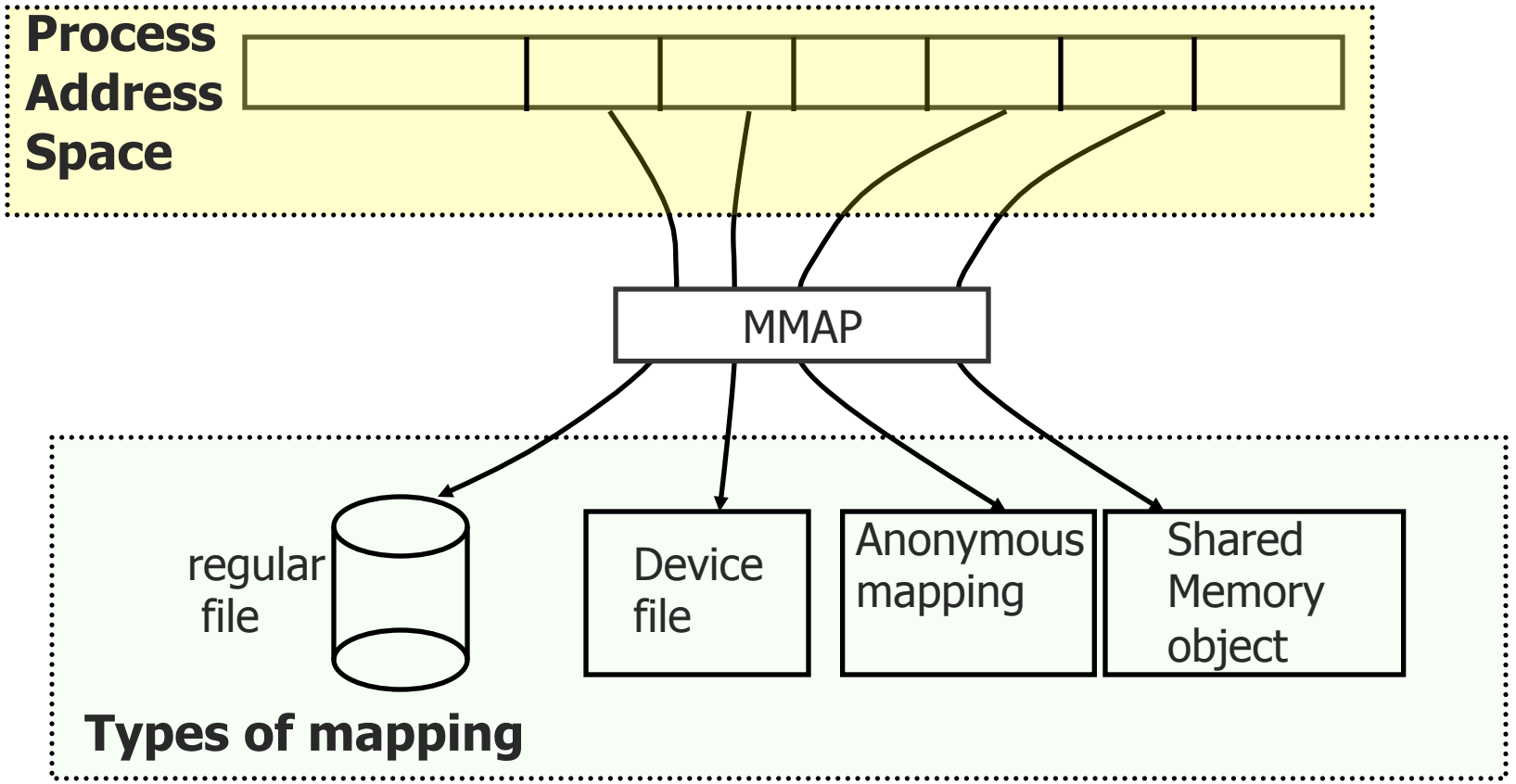
FLAGS

Concept of memory mapping

- If the virtual memory sub-system is integrated with the file-system, it enables a simple and efficient mechanism to load programs and data into memory

- If disk I/O requires the transfer of **large amounts of data (one or more pages)**, mmap significantly speeds up I/O by mapping a disk file directly into user-space memory
  - It does not suffer the overhead of syscalls like read/write
  - User-process has direct access to kernel disk cache

MMAP: a powerful syscall



- MMAP is used for mapping differing sorts of objects

MMAP

memory address of the mapping

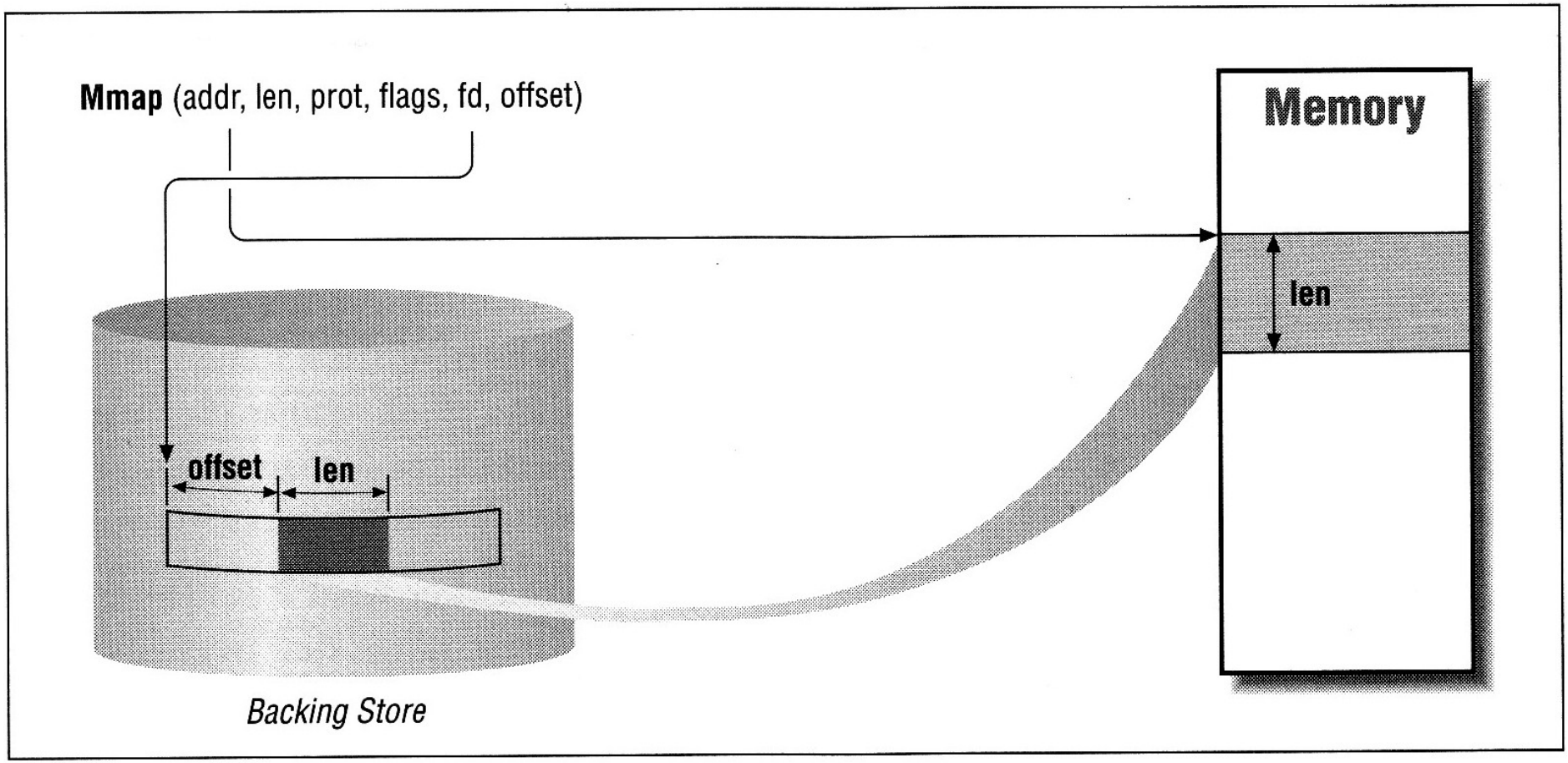
read/write/exec permissions for the mapping

MAP_SHARED, MAP_PRIVATE, MAP_ANONYMOUS

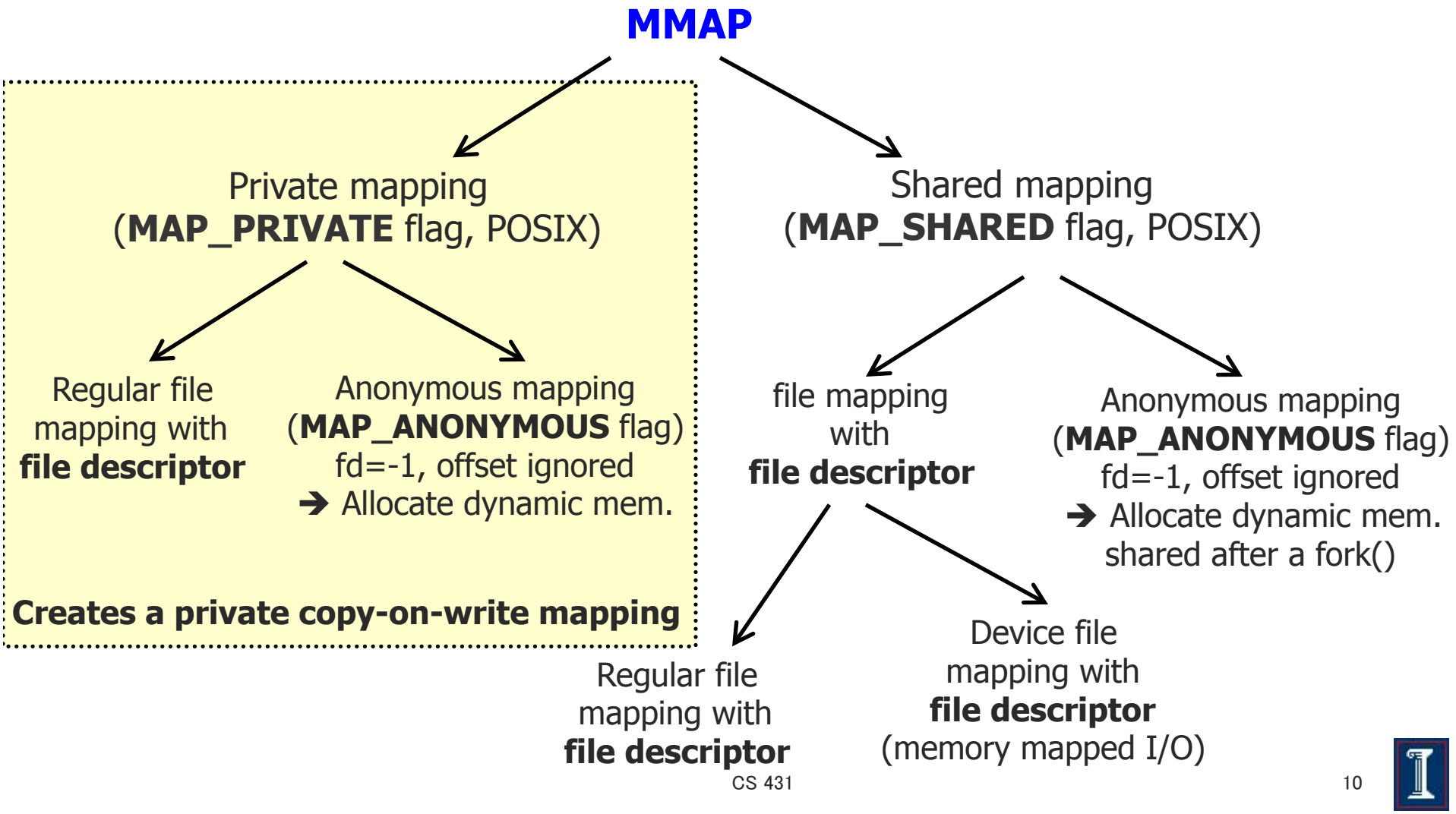
file descriptor of object to map

```
#include <sys/mman.h>

void *  mmap (void *addr, size_t len, int prot , int flags, int fd, off_t offset);
```



Types of mapping with MMAP



Private vs Shared mapping

- **MAP_PRIVATE**
  - Updates to the mapping are not visible to other processes mapping the same file, and are not carried through to the underlying file.
- **MAP_SHARED**
  - Updates to the mapping are visible to other processes that map a shared file, and are carried through to the underlying file. The file may not actually be updated until **msync** or **munmap**() is called
  - Further discussion later in the semester when covering inter-process communication (IPC)

Portable use of MMAP

- Set addr = NULL
  - the kernel chooses the address at which to create the mapping
  - MMAP always returns a page aligned address
- Virtual memory always allocates entire pages
  - offset must be a multiple of the page size
  - len must be a multiple of the page size

MUNMAP

int **munmap**(void *addr, size_t length);

- **munmap**() system call
  - It deletes the mappings for the specified address range.
  - It can unmap a smaller number of pages among those allocated by mmap (partial unmapping)
  - addr argument must be page aligned
  - len must be a multiple of the page size

Example: private mapping of regular file

```
#include <sys/mman.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#define  PAGESIZE   4096
int main(int argc, char **argv)
{
  int fd;
  char string[] ="CS241 takeaway: mmap can be used to map files in memory";
  char *ptr;

  //open a regular file, write a string, and map it into memory
  fd = open(argv[1],O_RDWR|O_CREAT,S_IRWXU);
  write(fd, string, sizeof(string) - 1);
  ptr = (char*) mmap(NULL,PAGESIZE,PROT_READ|PROT_WRITE,MAP_PRIVATE,fd,0);
  close(fd);

  printf("pointer to memory mapped file: %p \n", ptr);
  printf("%s \n", ptr);
  ptr[2]='4'; ptr[3]='3'; ptr[4]='1'; // it triggers a copy-on-write
  printf("%s \n", ptr);
}
```

Example: shared mapping of device file

```
#include <sys/mman.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#define  PAGESIZE   4096
int main(int argc, char **argv)
{
  int fd;
  char string[]="CS241 takeaway:mmap can be used to map dev. files in memory";
  char *ptr;

  //open a raw UNUSED disk partition, write a string, and map it into memory
  fd = open("/dev/sda3", O_RDWR, S_IRWXU);
  write(fd, string, sizeof(string) - 1);
  ptr = (char*) mmap(NULL,PAGESIZE,PROT_READ|PROT_WRITE,MAP_SHARED,fd,0);
  close(fd);

  printf("pointer to memory mapped file: %p \n", ptr);
  printf("%s \n", ptr);
  ptr[2]='4'; ptr[3]='3'; ptr[4]='1'; // it modifies the disk partition cache
  printf("%s \n", ptr);
}
```

Example: shared mapping of device file

copy the first 256bytes of disk partition /dev/sda3 to sda3.bin and check its content!

sudo dd bs=256 count=1 if=/dev/sda3 of=./sda3.bin

ghex sda3.bin