Linux Storage Stack

ECE 469, April 15

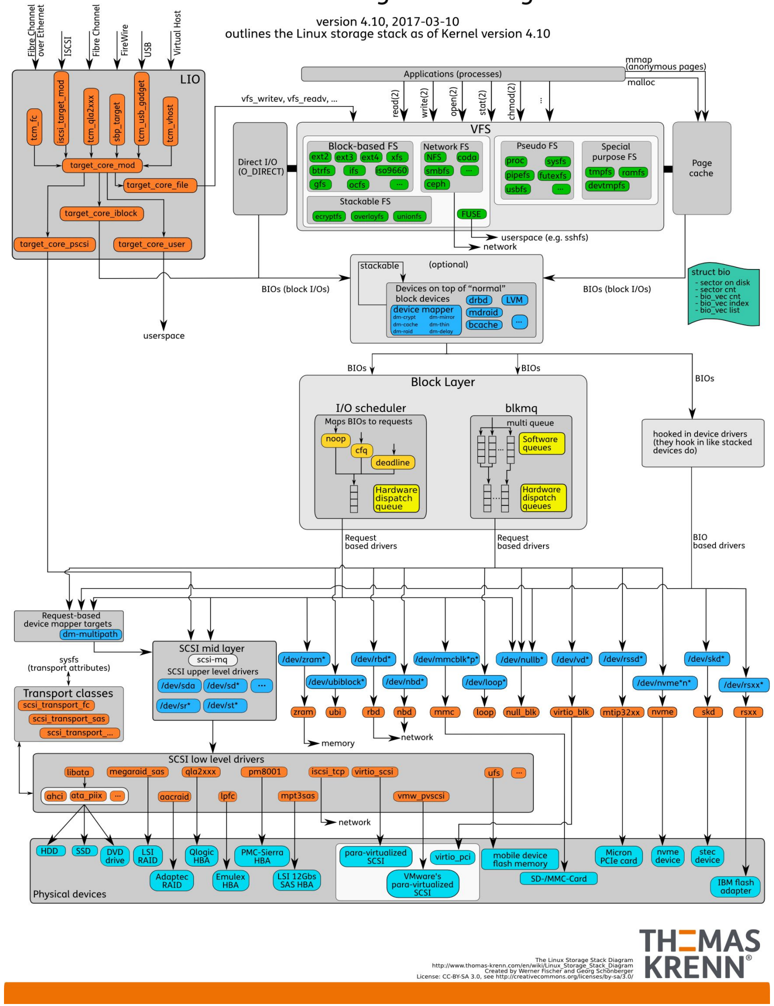
Aravind Machiry

Linux Storage Stack

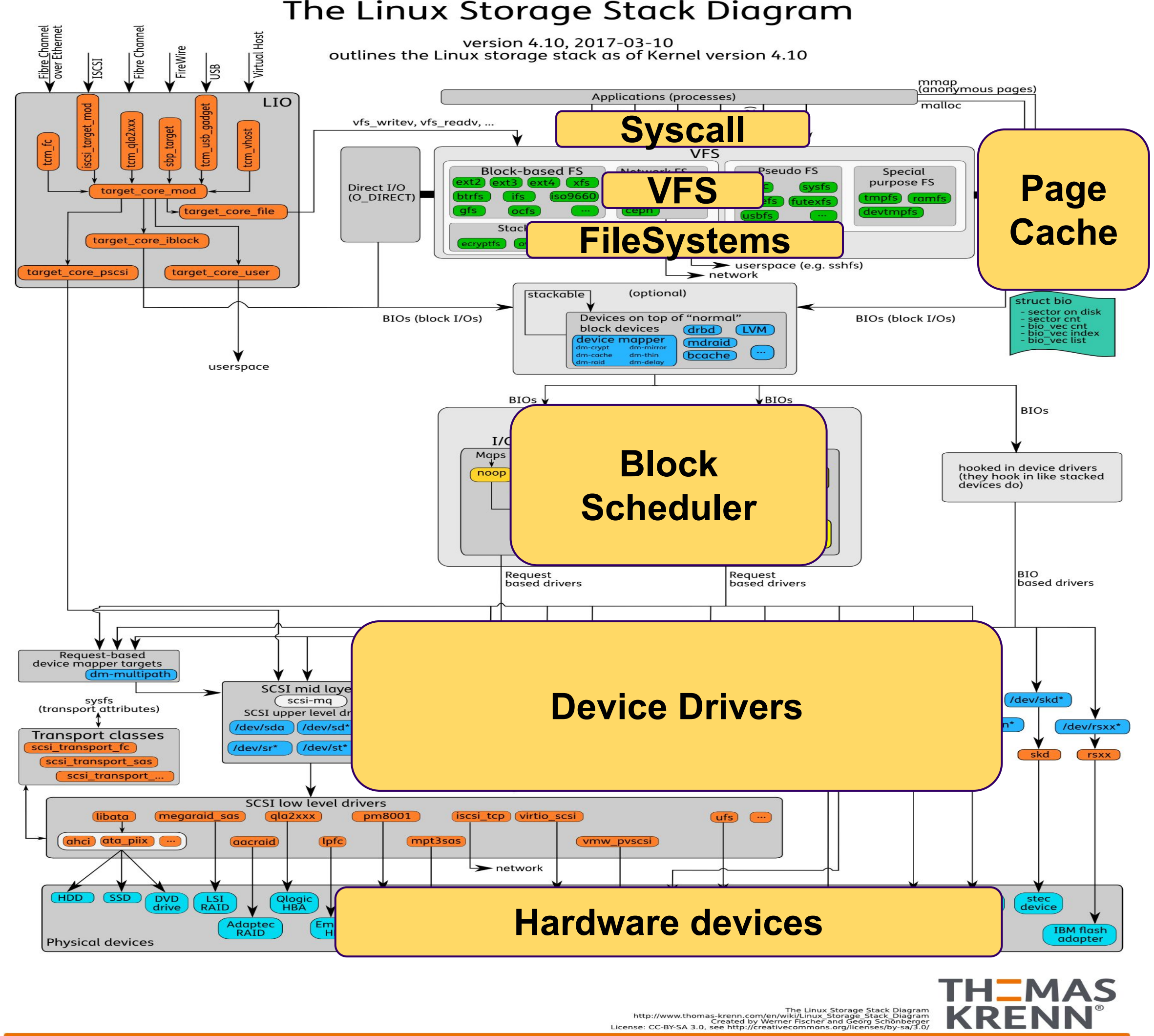
- Exhaustive and Modular

The Linux Storage Stack Diagram

version 4.10, 2017-03-10
outlines the Linux storage stack as of Kernel version 4.10



The Linux Storage Stack Diagram
http://www.thomas-krenn.com/en/wiki/Linux_Storage_Stack_Diagram
Created by Werner Fischer and Georg Schönberger



Syscall

VFS

FileSystems

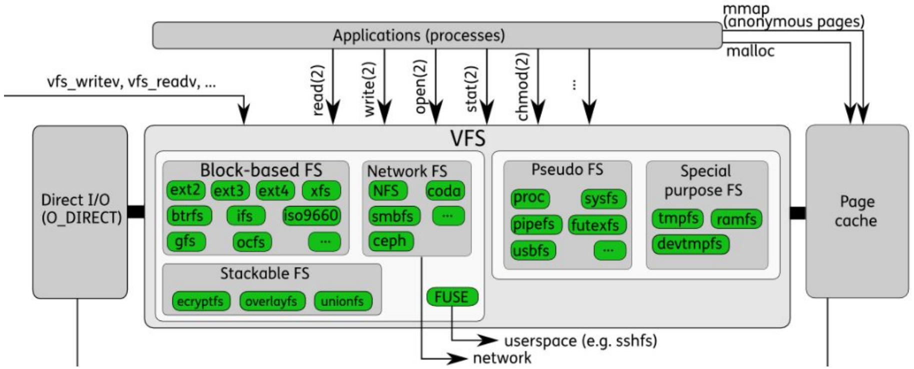
Page Cache

Block Scheduler

Device Drivers

Hardware devices

VFS

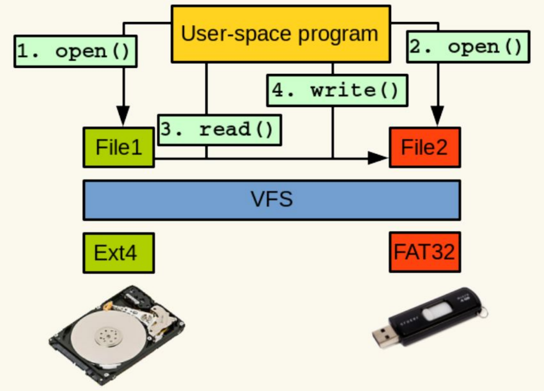


- Virtual File System (~22K SLOC).

- Everything is a File!!
  - E.g., Network file system! sshfs!?

- ~42 File Systems supported in Linux!!

VFS to Applications



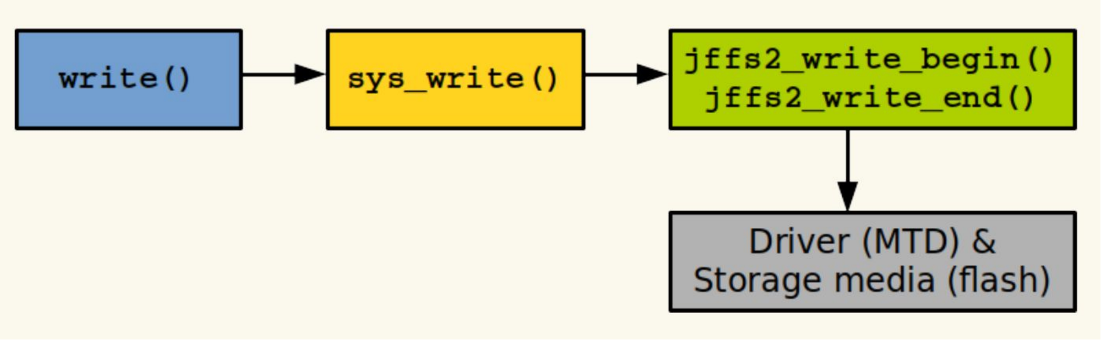
- Common interface for accessing files irrespective of file systems.

- File systems no need to worry about interface to user.

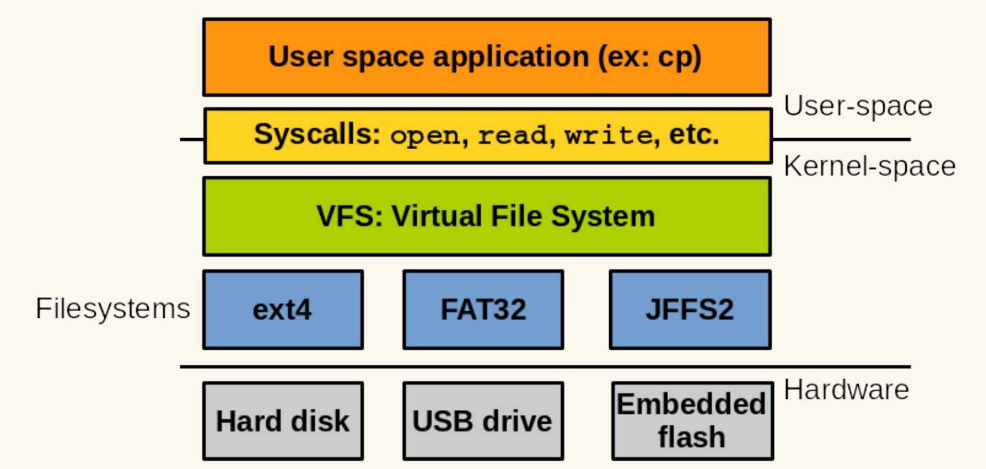
VFS to File System Implementers

- Exposes common optimization logic. E.g., Page cache, Path lookup.

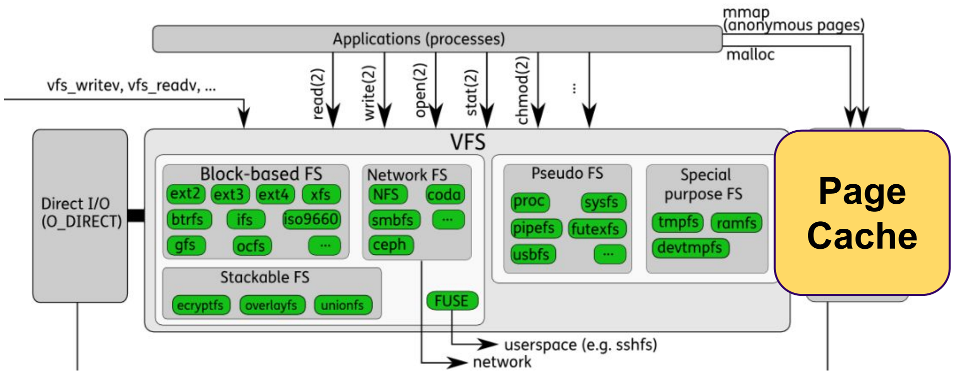
- Define functions to be implemented by the filesystems.



What does File System Implementers do?



Page Cache



- Reduce Disk IO
- Memory pages maintained by the kernel for storing contents to/from disks.
- Disk block <-> Page

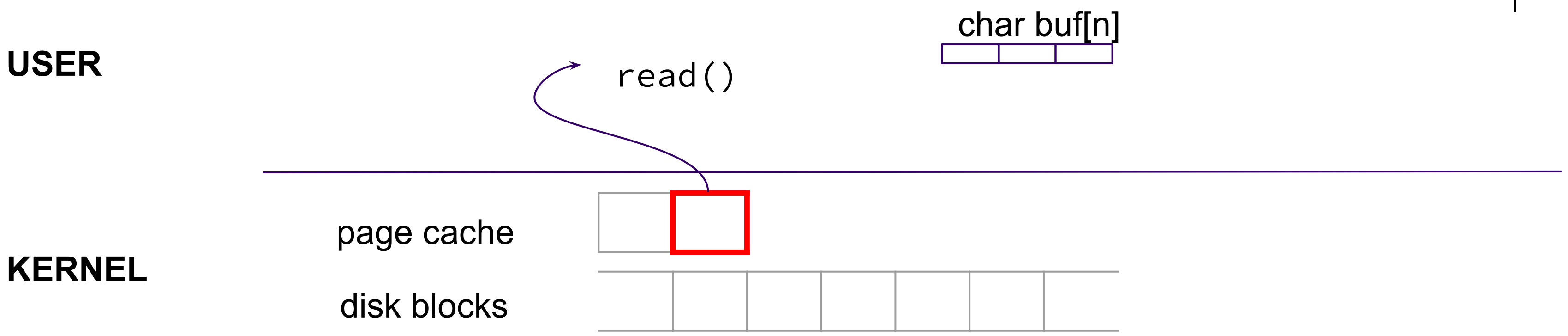
File IO with Page Cache

- *`read()`*: Serviced by Page Cache!
  - Optimization: Read ahead!

- *`write()`*: Dirty pages; will be written to disk later!
  - Can loose data!?

- *`sync()`*: Flush all writes to files.
  - Synchronous

File IO with Page Cache



USER

read()

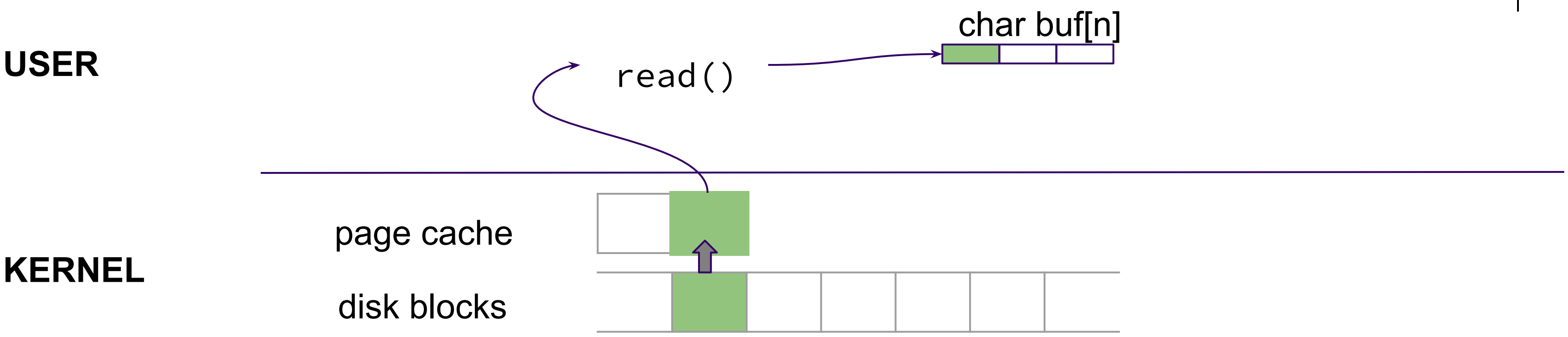
char buf[n]

KERNEL

page cache

disk blocks

File IO with Page Cache



Page Cache Implementation

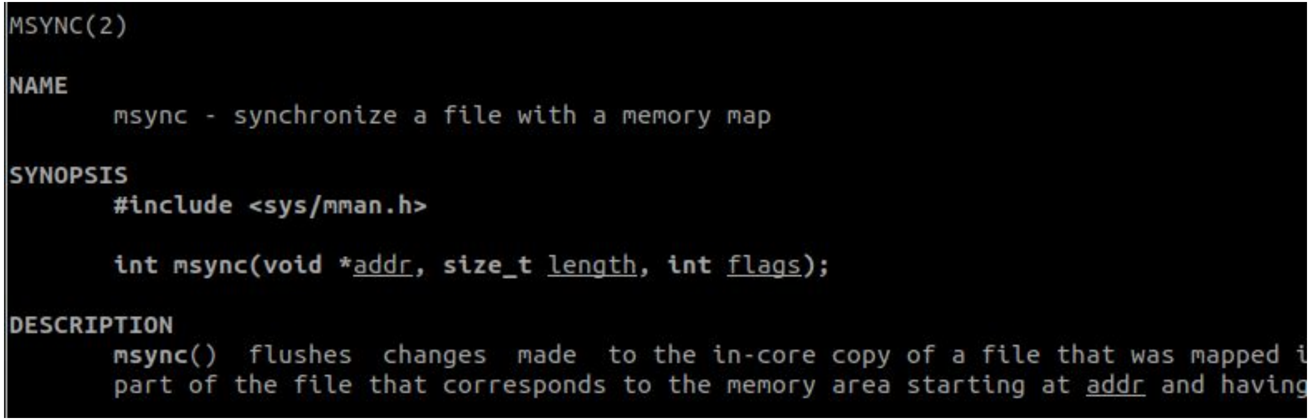
- For each file (inode):
  - Has addr space.
  - File offset -> Page cache.
- For each page:
  - A reference to the file/process.
  - The offset with in the file.

The mmap system call

- Bind virtual memory to file blocks.

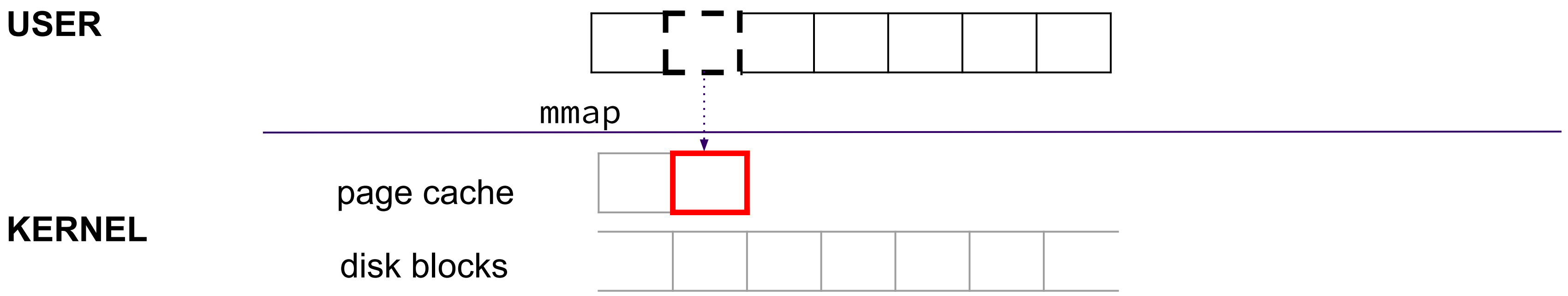
```
fd = open(“hello.txt”, O_RDWR);

// map 4k from offset 0 into virtual address space of the
process.
char *data = mmap(..,fd, 0);

// read 7th character from file.
char c = data[6];

// write 101th character into file.
data[100] = ‘a’
```

Flushing mmap region to file

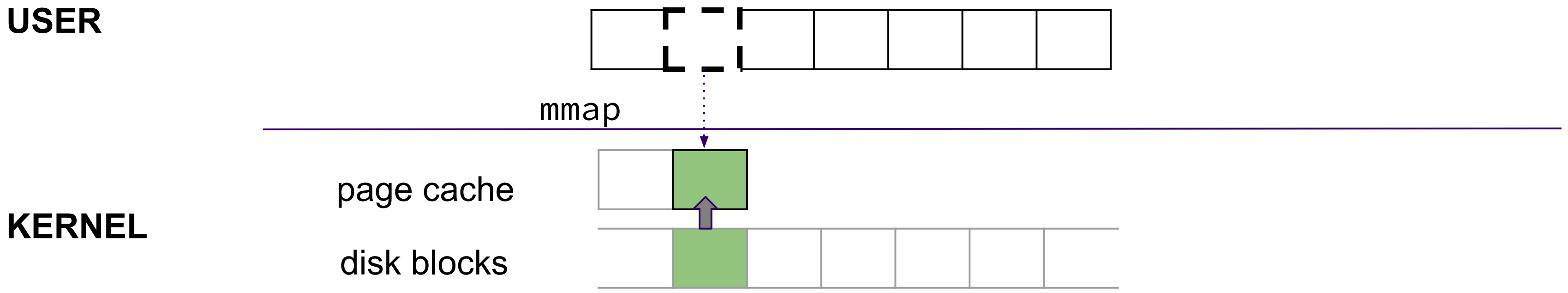


```
MSYNC(2)

NAME
       msync - synchronize a file with a memory map

SYNOPSIS
       #include <sys/mman.h>

       int msync(void *addr, size_t length, int flags);

DESCRIPTION
       msync()  flushes  changes  made  to the in-core copy of a file that was mapped i
       part of the file that corresponds to the memory area starting at addr and having
```

Memory RW with Page Cache

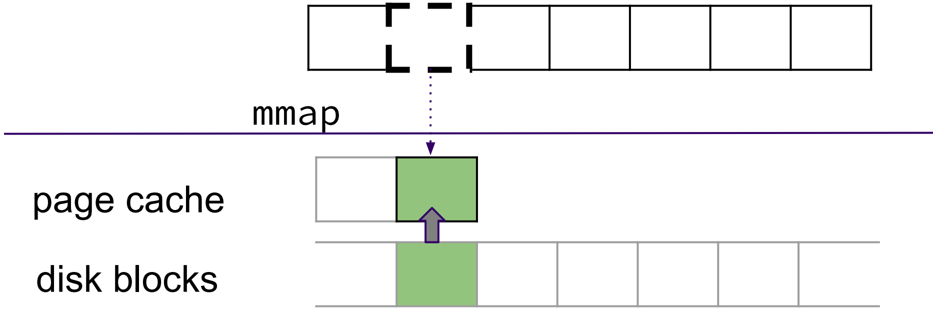


Memory RW with Page Cache

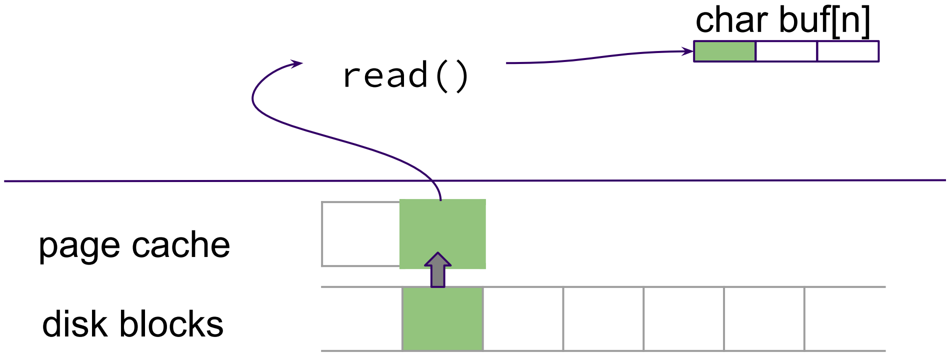


Mmap v/s Explicit IO

- Mmap:
  - No syscalls on each access.
  - Page cache <-> Disk.
  - Dynamic paging.
  - Extra PTEs.
  - Mapping large files? IO Errors?



- File IO
  - Universal.
  - app buffer <-> page cache <-> Disk.


char buf[n]
read()
page cache
disk blocks