University of BRISTOL

Computer Systems B

COMS20012

Introduction to Operating Systems and Security

File Systems

bristol.ac.uk

University of BRISTOL

I/O Devices

bristol.ac.uk

Devices

- **Devices** is how computer receive inputs and outputs
  - **Keyboard** is an input device
  - **Printer** is an output device
  - **Touch Screen** is both input and output

- Sys161 have the following devices
  - Timer/clock
  - Disk
  - Serial Console
  - Text Screen
  - Network interface

Terminology

- **Bus**: communication pathway between devices in a computer
  - **Internal bus**: bus between the CPU and the RAM. Relatively fast!
  - **Peripheral**: or extension bus, allow devices within the computer to communicate
- **Bridge**: connects two different buses

Device Register

- Communication with devices carried through **device registers**

- Three primary types of registers:
  - **Status**: tells you about the state of a device
  - **Command**: issue a command to the device by writing a particular value
  - **Data**: used to transfer larger block of data

- Some device registers can be combination of primary types:
  - **Status and command**: read for device state, write for command

Device register: Sys161 example **clock**

Offset	Size	Type	Description
0	4	status	current time (seconds)
4	4	status	current time (nanoseconds)
8	4	command	restart-on-expiry
12	4	status and command	interrupt (reading clears)
16	4	status and command	countdown time (microseconds)
20	4	command	speaker (causes beeps)

Device register: Sys161 example **serial console**

Offset	Size	Type	Description
0	4	command and data	character buffer
4	4	status	Read IRQ
8	4	status	Write IRQ

Device driver

- Part of the kernel that interface with a device
- Example write a character to the serial console

wait(console_semaphore) # only one write at a time

write to character buffer

while(writeIRQ!=completed)

write writeIRQ to acknowledge completion

signal(console_semaphore)

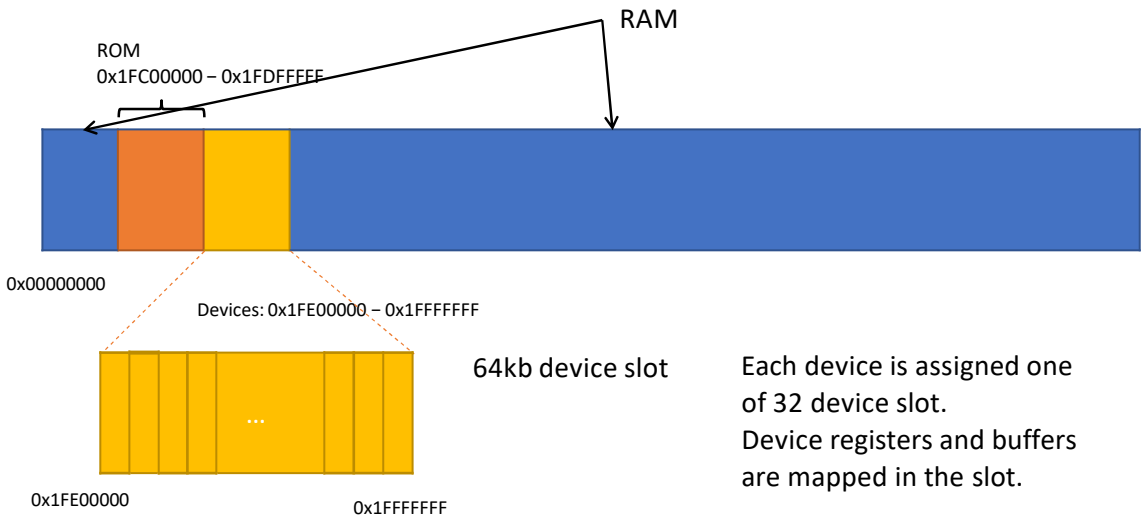
Device driver

- **Polling** approach
  - Check repeatedly the status of the device
  - Polling is bad (waste CPU cycles)
- Instead we should rely on **interrupts**
  - An interrupt is simply a signal that the hardware can send when it wants the processor's attention.

Accessing device registers

- How can our driver access device registers?
  - Option 1: **port-mapped I/O** with special instructions
    - Device are assigned port numbers which corresponds to an address in a separate smaller address space
    - Special instruction to read/write to this address space (in/out on x86)
  - Option 2: **memory-mapped I/O**
    - Each device registers associated to a physical memory address
    - This is not mapped to user space virtual addresses!
    - Read/write using normal load/store instructions (as reading/writing to normal memory)
  - An architecture can have both

Sys161 example

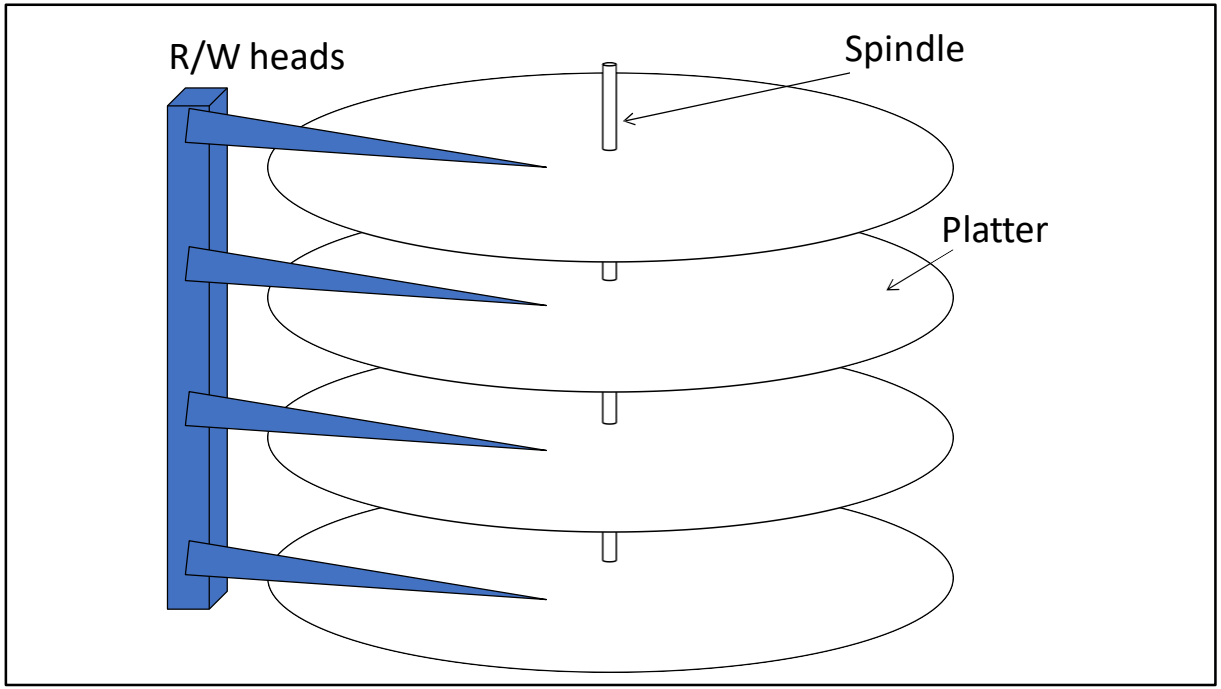


64kb device slot

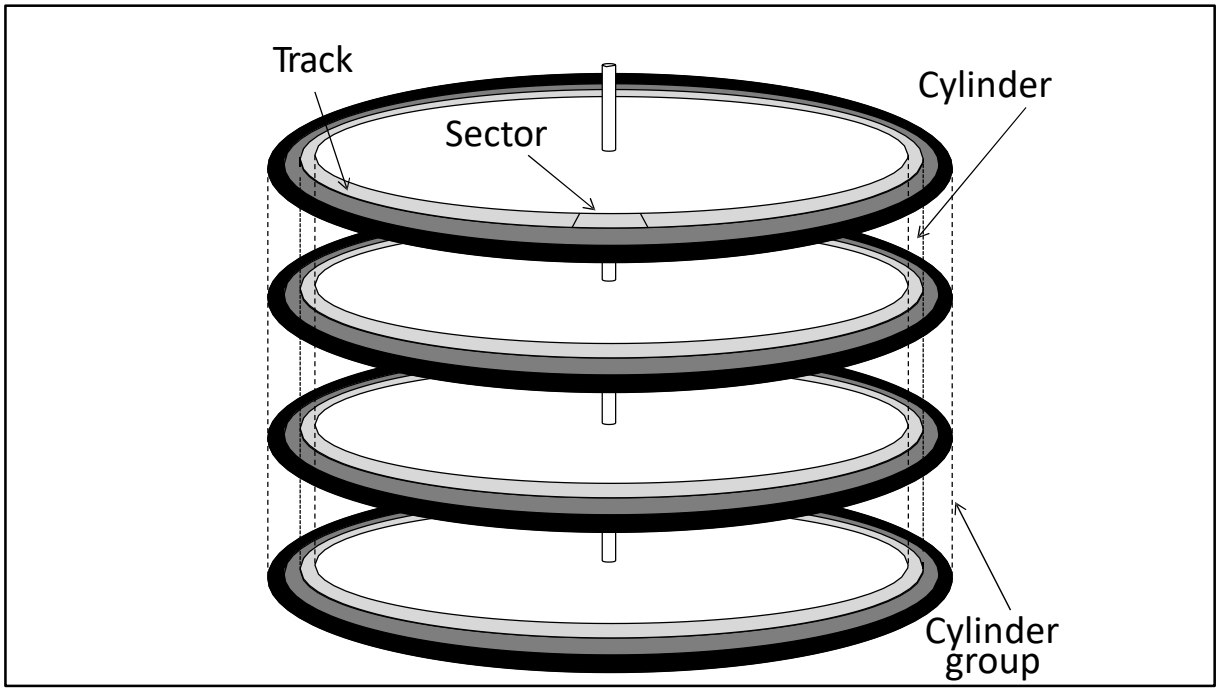
Each device is assigned one of 32 device slot.
Device registers and buffers are mapped in the slot.

Large data transfer

- Write bytes one by one in register won't be very efficient
  - Think of a hard drive
- Buffer in memory
- Two strategy for transfer
  - Program-controlled I/O
    - The device driver move data between the CPU and I/O device
    - The CPU is Busy
  - Direct memory access DMA
    - The device itself copy the data from memory to itself
    - The CPU is not busy while this happen
    - The device will trigger an interrupt when done
- Sys161 disks use program-controlled I/O



Note that the platters are double-sided, i.e., they store data on both sides. Also note that all of the read/write heads move together, in unison.



For a long time, hard disks used a sector size of 512. However, modern disks now use a sector size of 4K.

Cylinder group to blocks

- Cylinder groups are divided into blocks
- Blocks can be addressed to read/write from disk
- You can check the textbook for discussion on optimization around reading/writing from hard drive
- – 6.1.2 (page 223)
  - –6.1.3 (page 226), first finish all videos
  - –Not mandatory, just if you are curious

Device register: Sys161 **disk controller**

Offset	Size	Type	Description
0	4	status	number of sectors
4	4	status and command	status
8	4	command	sector number
12	4	status	rotational speed
32768	512	data	transfer buffer

Writing to a Sys161 disk

- Device driver

wait(disk_semaphore)
copy data from memory to transfer buffer
write target sector to sector register
write "write" command to disk status register
wait(disk_completion) signal(disk_semaphore)

- Interrupt handler

write disk status register to acknowledge completion
signal(disk_completion)

Reading from a Sys161 disk

- Device driver

wait(disk_semaphore)

write target sector to sector register
write "read" command to disk status register
wait(disk_completion)
copy data from buffer to memory
signal(disk_semaphore)

- Interrupt handler

write disk status register to acknowledge completion
signal(disk_completion)

University of BRISTOL

Files on disk

bristol.ac.uk

It is all about abstractions

- OS see storage as a large addressable array of bytes
- User space wants better abstraction
  - –Naming: /pics/meme.jpg instead of bytes between 24,048 to 28,156
  - –Performance optimization
    - Caching
    - Pre-fetching
  - –Transparent sector/block management
  - –Reliability in case of crash/power failure

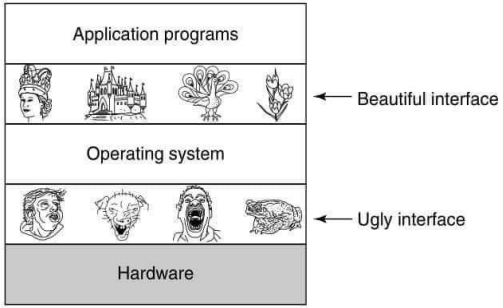


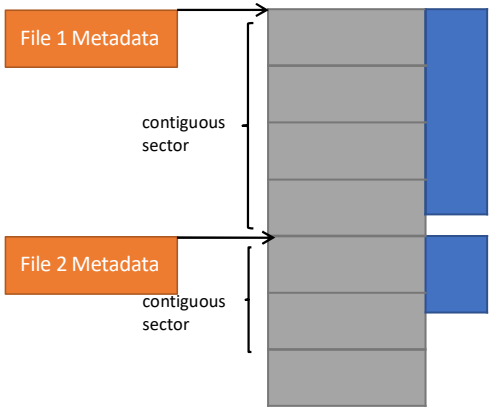
Figure 1-2. Operating systems turn ugly hardware into beautiful abstractions.

Modern Operating Systems, by Andrew Tanenbaum, Herbert Bos, Pearson

Files and Directories

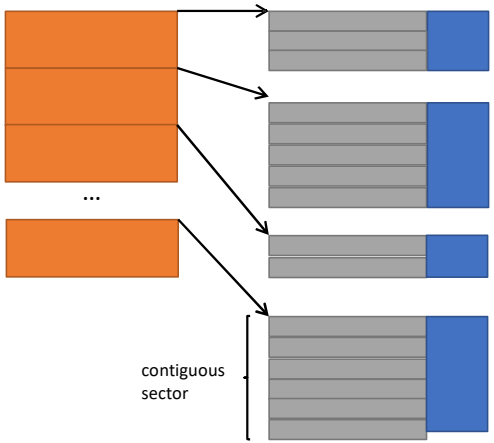
- File: linear region of bytes that can grow and shrink
  - Associated with metadata
    - A name (e.g., meme.jpg)
    - Size in bytes
    - Access permissions (read/write/execute)
    - Statistics (e.g., creation and access dates)
  - OS is agnostic to the content of the file (userspace is to interpret it)
- Directory: container for files and other directories
  - Associate with a name + metadata
  - Nested directories can create a hierarchy (e.g., /home/bob/pictures/meme.jpg)

File as a single extent (1960's file systems)



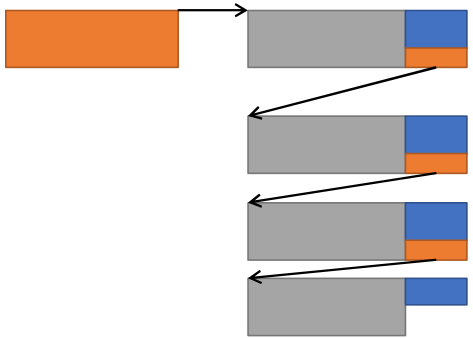
- File metadata
  - Starting sector
  - Length in byte (last sector may not be fully used)
- Advantages
  - Simple
  - Small metadata
  - Good sequential and random I/O
- Problems
  - How much space to allocate to new files
  - What to do if a file grow beyond its allocation? Or Shrink?
  - External fragmentation

File as a collection of extents (IBM 360, ext4)



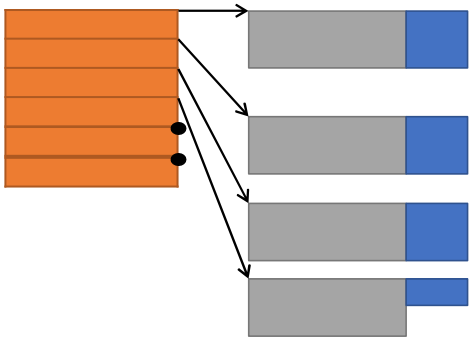
- Advantage
  - Metadata remain relatively small
  - Almost as good sequential I/O
    - Sequential offset calculation a bit tricky
  - Almost as good random I/O
- Challenges
  - How large the initial extent should be?
  - What to do if a file grows or shrink?
  - Improve on fragmentation!

Files as linked list: (FAT)



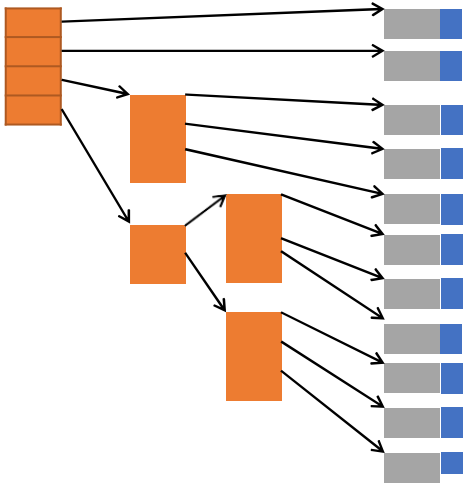
- Advantages
  - Easy to shrink and grow
  - Low internal and external fragmentation
  - Sequential offset calculation is easy
- Disadvantages
  - Need to go through the list to find the part ones need
  - Some metadata at the end of each data block
  - Sequential I/O requires lots of seeks (on hard drive mechanical movement)

Files as flat indices



- Array mapping ranges to a block
- Advantages
  - Offset is easy to calculate
  - Low fragmentation
- Disadvantages
  - Maximum file size is fixed by number of entries in an index
  - Sequential I/O requires lots of seeks (on hard drive mechanical movement)

Files as hybrid indices (FFS, ext2, ext3)



- Top level index contains: direct pointers, indirect pointers, doubly indirect pointers etc.
- Advantages
  - Efficient for small files (do not materialize unused indirect list)
  - Big maximum file size (function of depth and index size)
  - Low fragmentation
- Disadvantages
  - Sometimes multiple disk access for a single read/write (need to fetch indirections)
  - Still require a large number of seek

University of BRISTOL

Naming

bristol.ac.uk

Naïve naming

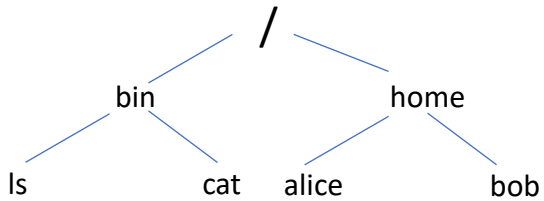
- One directory for the entire disk
- Small maximum name size
- Maximum number of files at creation
- Implementation
  - Allocate space for the directory
  - Directory structure is a big map
    - key: char file_name[MAX_SIZE]
    - value: file representation (see video 2) or an ID that easily map to it

- Pros:
  - Simple to implement
- Cons:
  - Hard to organize data
  - No two objects with the same name
  - Collision likely in multi-users systems
  - Names are limited

Hierarchical structure

- Tree structure
- Directories are normal files with a specific format
- Bit indicate if file or directory
- Contains directory entries
  - –Map name to some ID
  - –User can read
  - –Only kernel can write

- Pros
  - –Much better organization
  - –Reuse file implementation
- Cons
  - –File look up is a bit more complicated
    - ➢Need to traverse directories



Directory Implementation

- Directory are implemented like files
- Content of directory entries:
  - Name
  - inode number (the name come from the original UNIX)
  - Type
- Root directory has a designated inode (so we know where to start!)

Root directory “/”

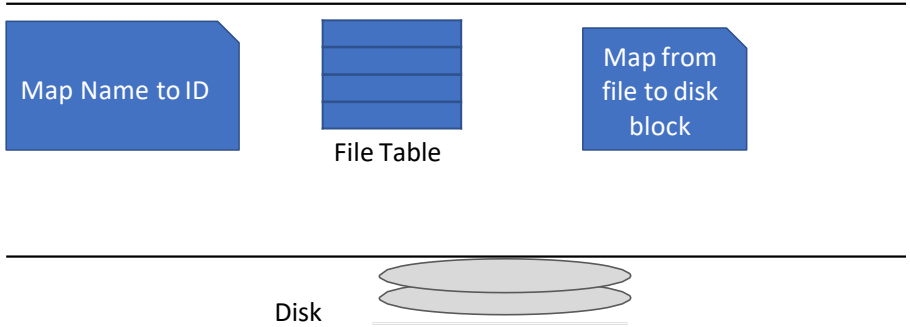
Name	inumber
home	6
etc	2254
net	3
sbin	4512
var	25615
tmp	14525
…	…

University of BRISTOL

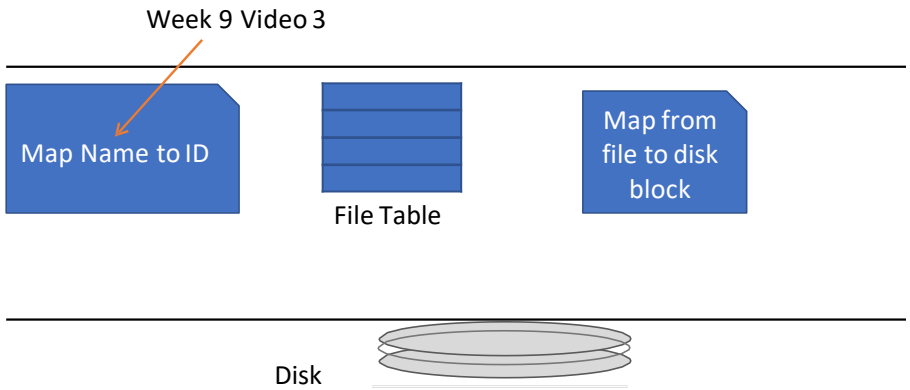
Files and System Calls

bristol.ac.uk

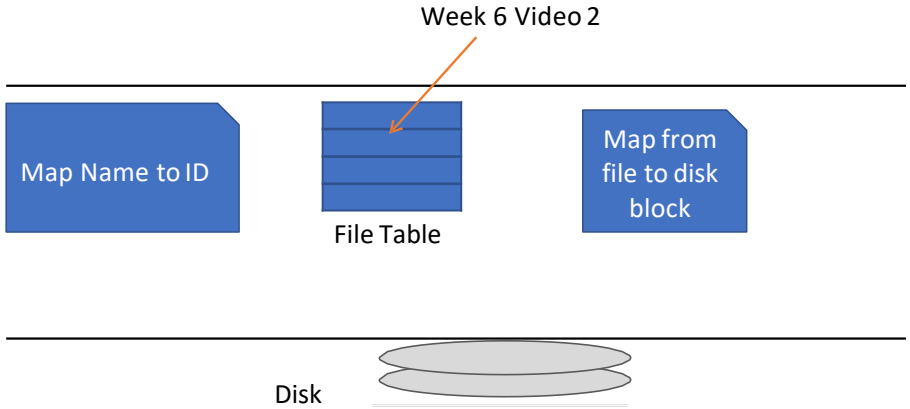
From System Call API to disk



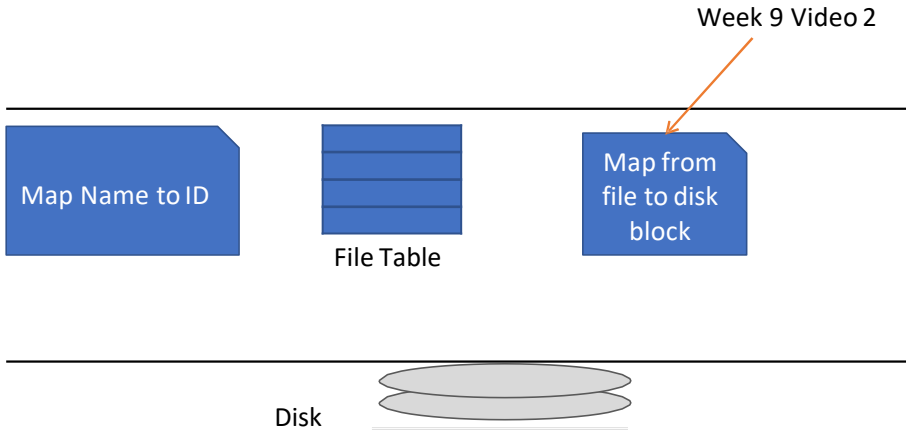
From System Call API to disk



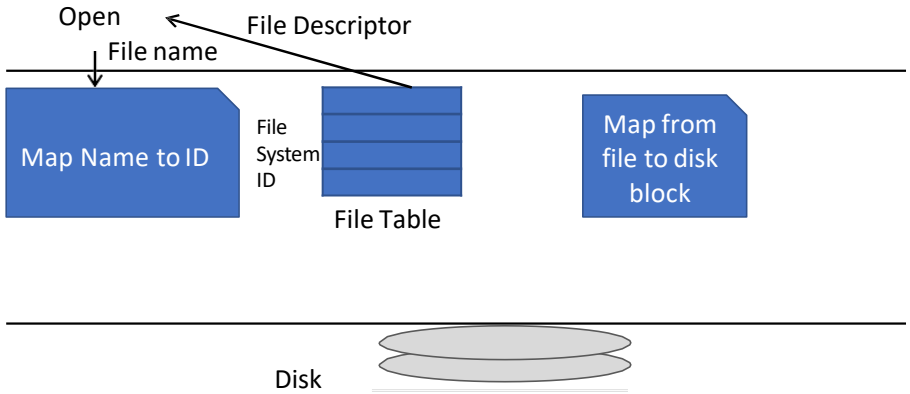
From System Call API to disk



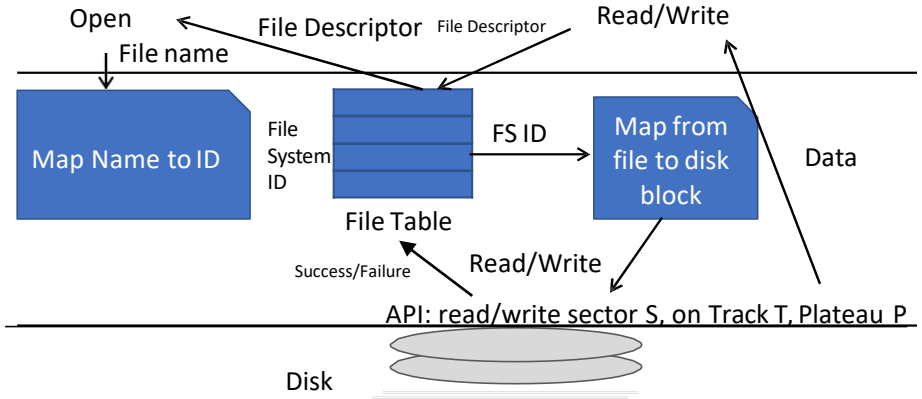
From System Call API to disk



From System Call API to disk



From System Call API to disk



University of BRISTOL

Problem?

bristol.ac.uk

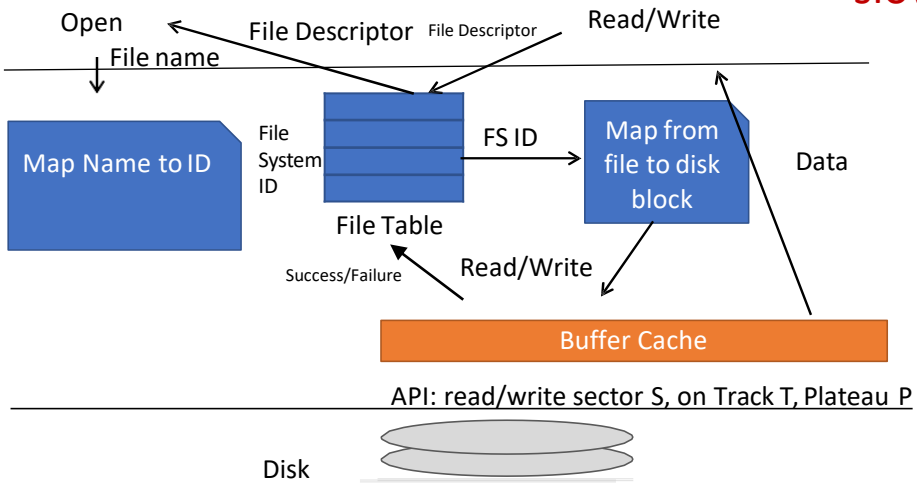
University of BRISTOL

Problem?
Disk are very slow!

bristol.ac.uk

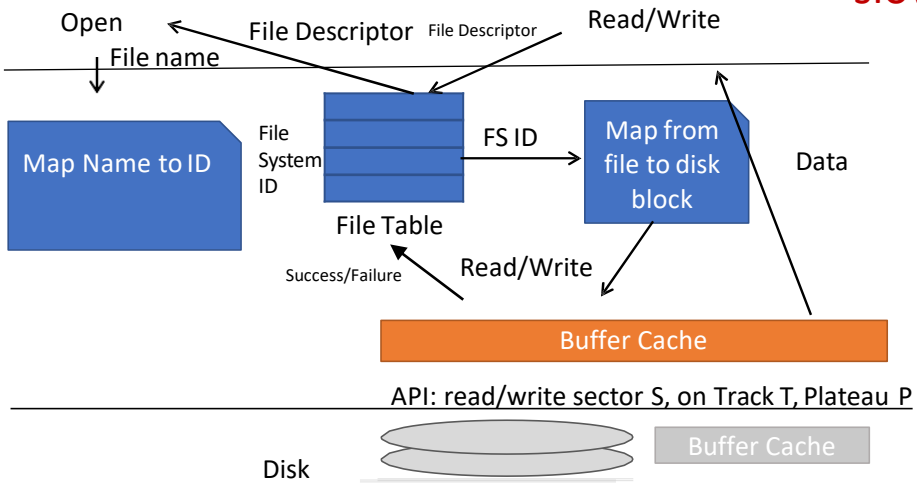
From System Call API to disk

Typical CS solution to fix
"slow" add some cache!



From System Call API to disk

Typical CS solution to fix
"slow" add some cache!



University of BRISTOL

Thank you

bristol.ac.uk