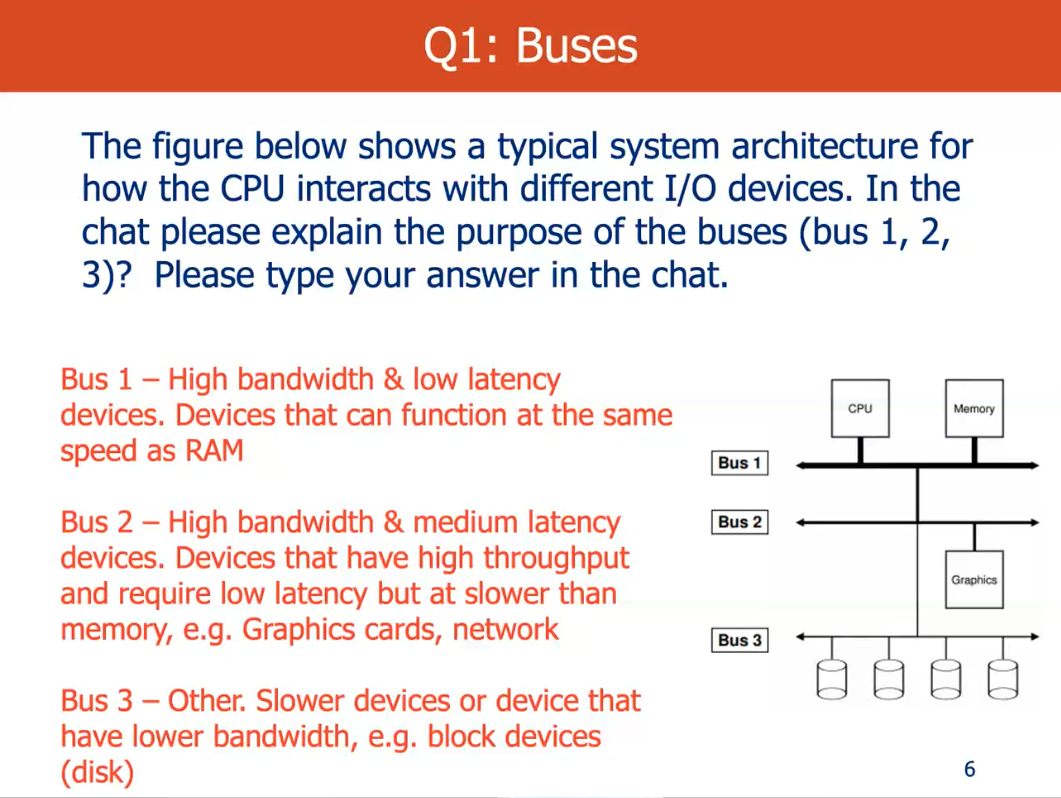
2

FOR Q1 SEE LIVE SESSION RECORDING (from live session 2020 [here](https://imperial.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=4522a879-df22-49f7-93a4-ac7900fc340d))

1a) Multiple buses exist since devices on faster buses can be processed quicker but they provide less plugs and have a larger cost. That is not needed often. The purpose of the bus 1 in the picture is to exchange data between the memory and CPU. The purpose of the second and third bus is to connect devices to the CPU and memory with faster devices being on the second bus and slower on the third bus.



(From live session 2020 [here](https://imperial.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=4522a879-df22-49f7-93a4-ac7900fc340d) at 15:00)

b)

1. Fair access to shared devices
2. Exploit parallelism of I/O devices for multiprogramming
3. Hide complexity of device handling
4. Give uniform naming and error handling

A

c) Character devices - store/retrieve data as a stream of characters(bytes)

Block devices - store/retrieve data in fixed size blocks. So, block devices tend to be faster than the character device as they transfer more data within a single interrupt handler (causes overhead).

Block devices are buffered/cached in memory whereas character devices are unbuffered.

d) Block devices are faster so they can be on the second bus. However, some are still connected to the 3rd bus such as a disk. The character devices are connected to the 3rd bus.

1e)

i) To bus 2. Its performance is between that of RAM (connected to bus 1) and a disk (connected to bus 3)

ii) It should be considered as a block device since transferring data from/to memory one byte at a time would be extremely inefficient

iii) To design the I/O device abstraction we need to implement the interface common to all devices. In a UNIX-style system devices are represented as files so the interface is similar to that of files (read, write, seek). Most of these requests would simply be passed on to the device driver which knows the device specifics allowing it to satisfy those requests.

2

2a) 1. Type and Access Control

2. Number of links

3. User ID

4. Group iD

5. Access time

b)

Direct, Indirect, Double Indirect, Triple Indirect

For different file sizes

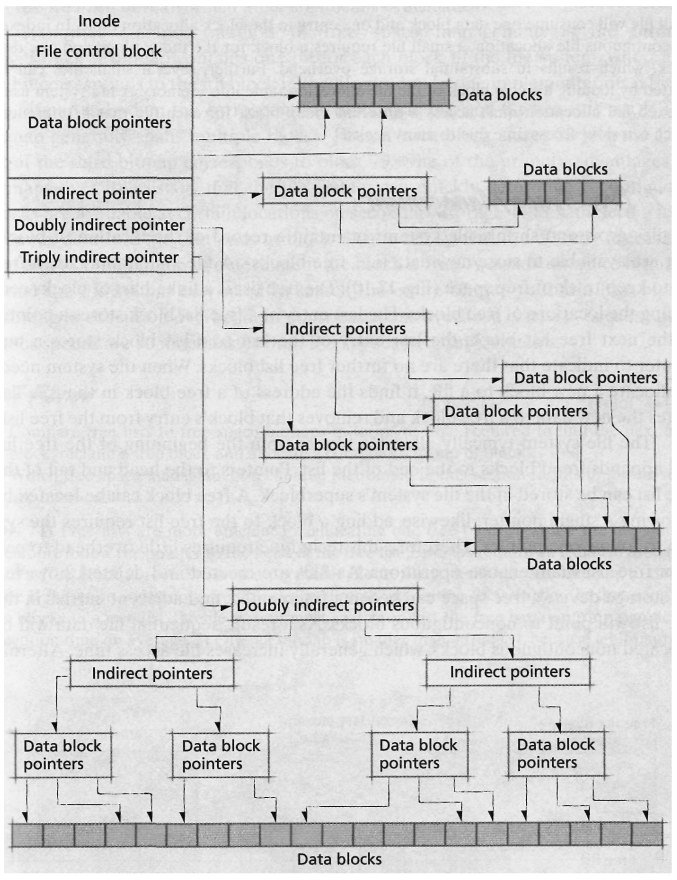
c) TODO

Pointers 1-4 should be direct pointers and pointers 5 & 6 should be indirect pointers. This means that for the file wombat.txt, it'll be able to be stored all in the first four pointers as 4 \* 4096 = 16384 > 16000 = 16Kb. The file dingo.txt could be stored in the 2 indirect pointers as 1 block is 4096 Bytes which is 1024 pointers, so 2 blocks = 2048 pointers. 2048 Blocks = 8,388,608 > 8,209,000 so the file would fit.

For both files the pointers are stored sequentially, apart from the split in the data blocks, which improves efficiency (I think). Also means that space isn't wasted with double/triple indirect pointers when they aren't required? -> Need to state that double/triple indirect pointers are reserved for very large files, should the file system hold larger files than dingo.txt/ wombat.txt. Tree-like structure means exponential scalability & also logarithmic, main memory/ cache access time for the pointer to a specific block in a large file. Different pointers types to make small files essentially have random access time but also accomodate large files. (Question is about a general file system)

Note that if the file sizes are actually KiB, not KB, then since 8029 = 4096\*2 + 17, a double-indirect pointer is needed (or three indirect pointers and three direct, but i think that takes more lookups)

d) TODO



e) TODO

Assuming the file system doesn't also have to include dingo.txt & wombat.txt

Reduce the block size to 16/32/64 Bytes, increase the pointer size accordingly so every space is mapped to. Then change the inode to include only 1 direct pointer and 1 indirect pointer so the odd large file can still fit but there's no wasted space for unused double/triple indirect pointers.

One direct pointer, the smaller, the better. Too bad if you want to store a bigger file.

Keep a fixed size array (eg 16 bytes) in the inode struct itself, so that for files < 16B you don’t need any blocks at all as the file contents can be stored within the inode.

Thank you Tanuj

Yw b

4\*direct + 2\*indirect but also store 10 B file on inode support:

When file size <= 20 B: inode.direct\_1 = NULL and union is used to store up to 20B of data.

When file size > 20 B: inode.direct\_1 != NULL and union used to store other block pointers.  
Struct inode {

…

void \*direct\_1;

Union {

Uint8\_t data[20];

Struct {

Void \*direct\_2;

Void \*direct\_3;

Void \*direct\_4;

Void \*indirect\_1;

Void \*indrect\_2;

} blocks;

} tail;

}