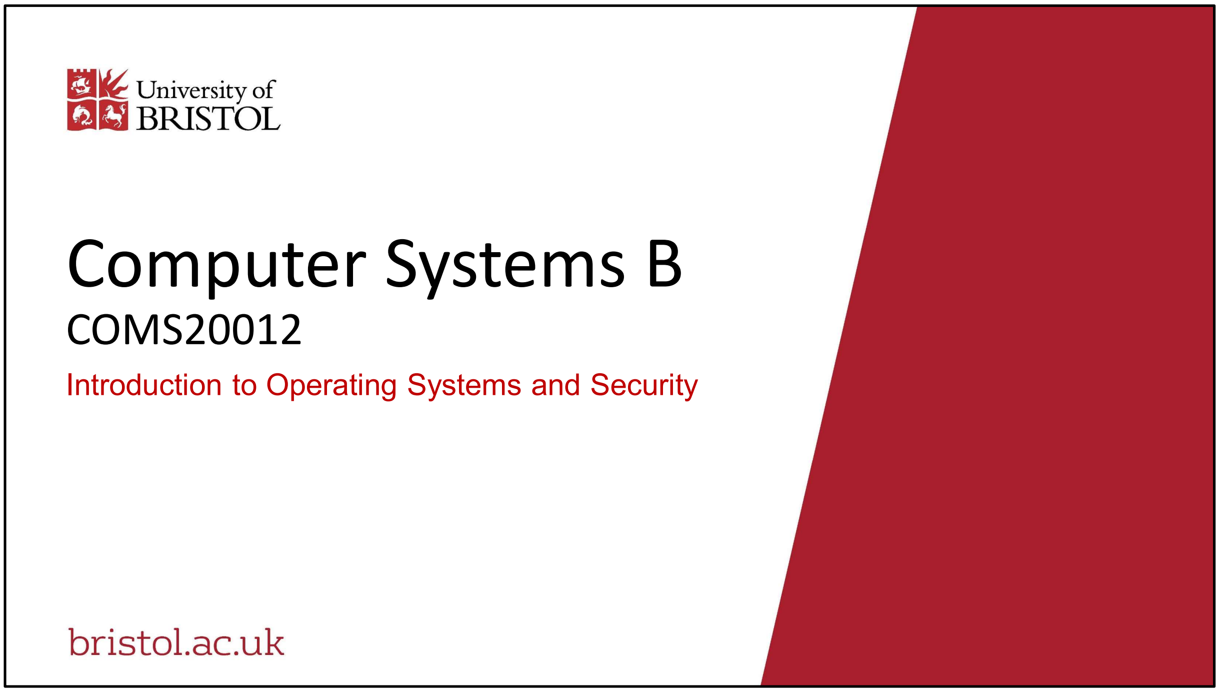


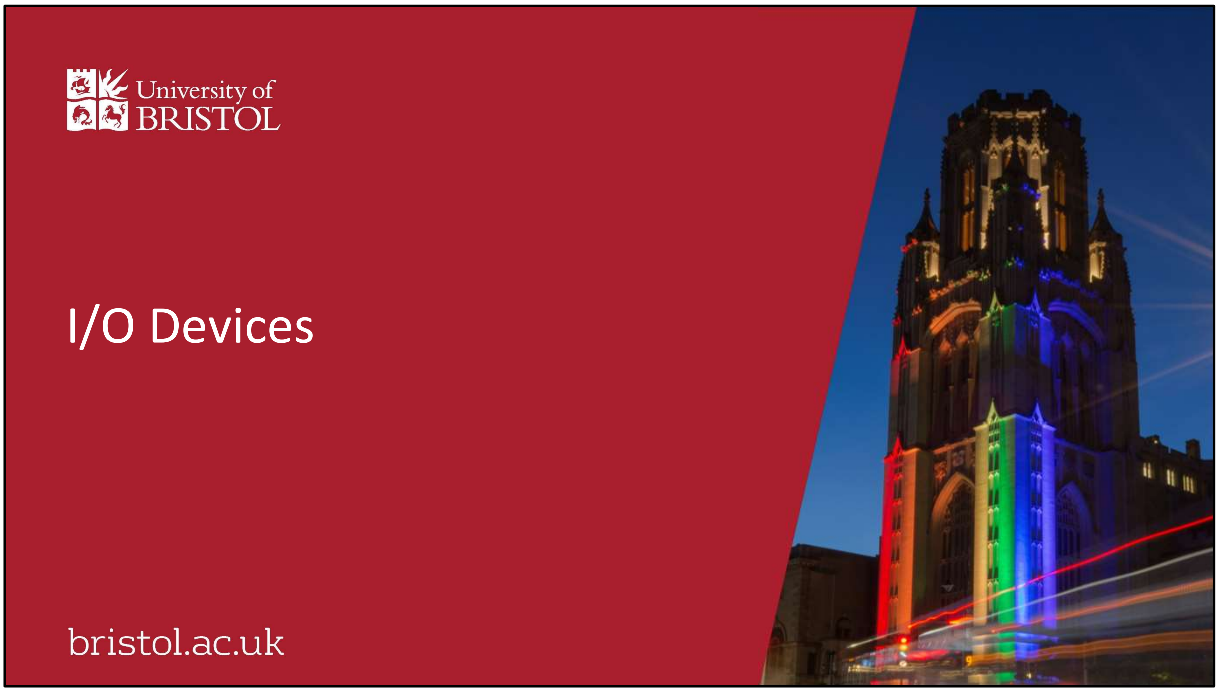
University of BRISTOL

Computer Systems B

COMS20012

Introduction to Operating Systems and Security

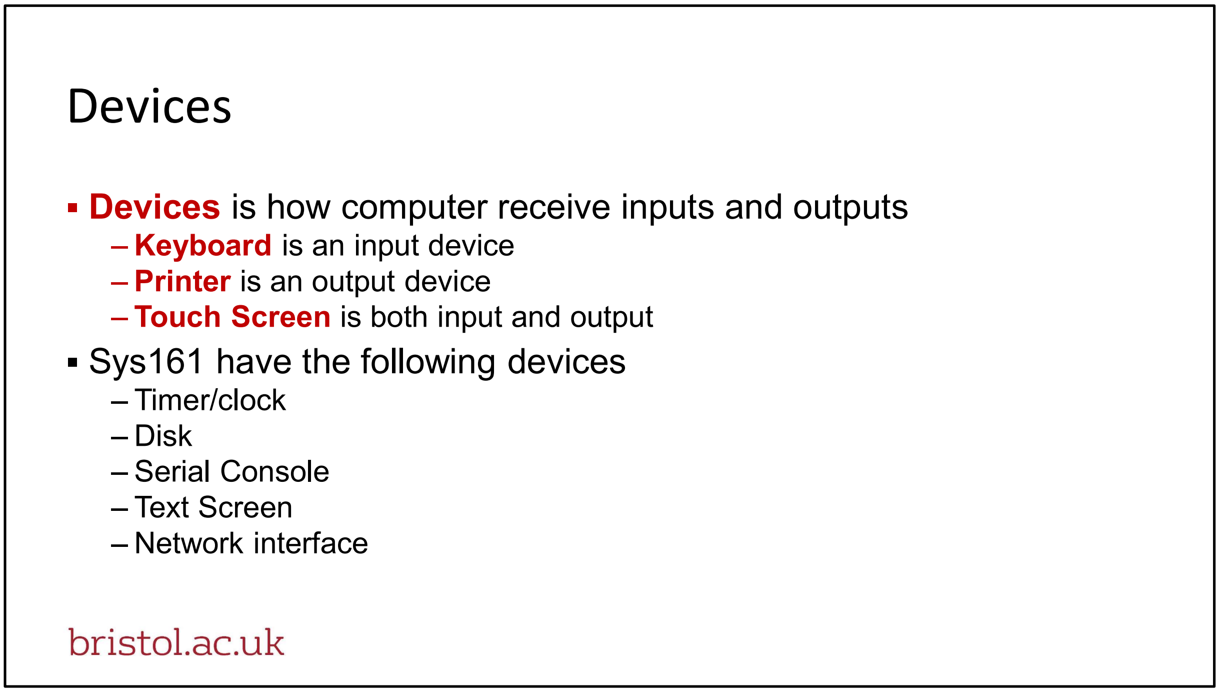
bristol.ac.uk



University of BRISTOL

I/O Devices

bristol.ac.uk



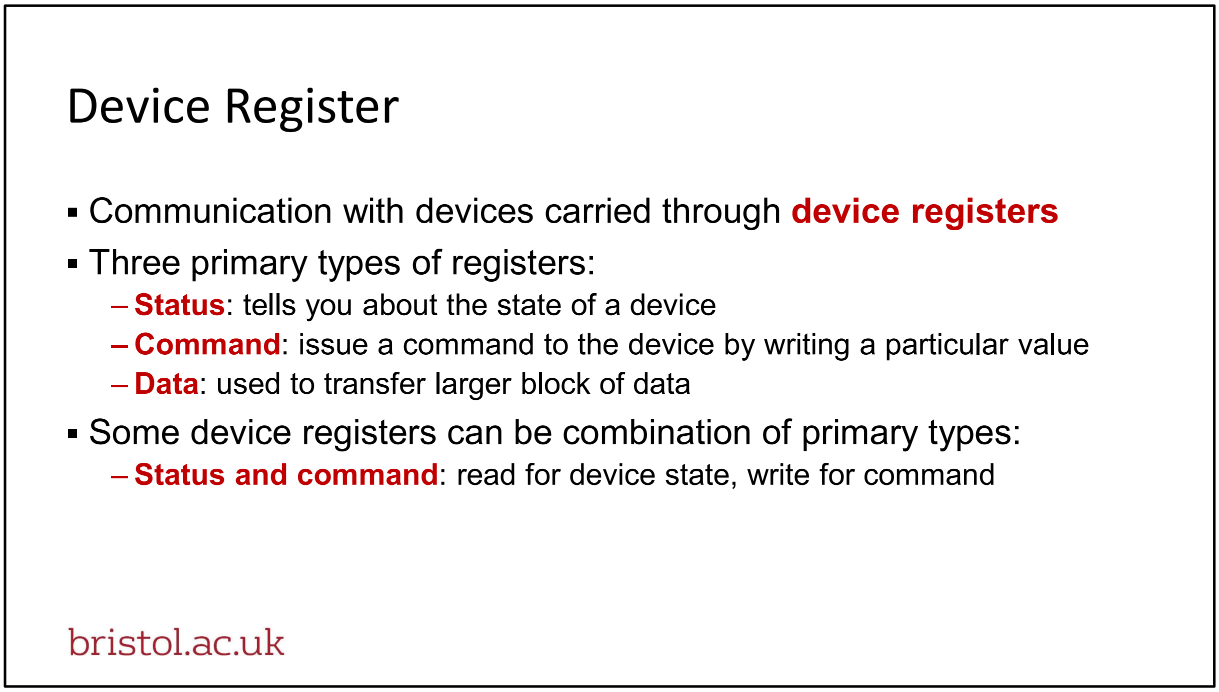
Devices

- **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

bristol.ac.uk

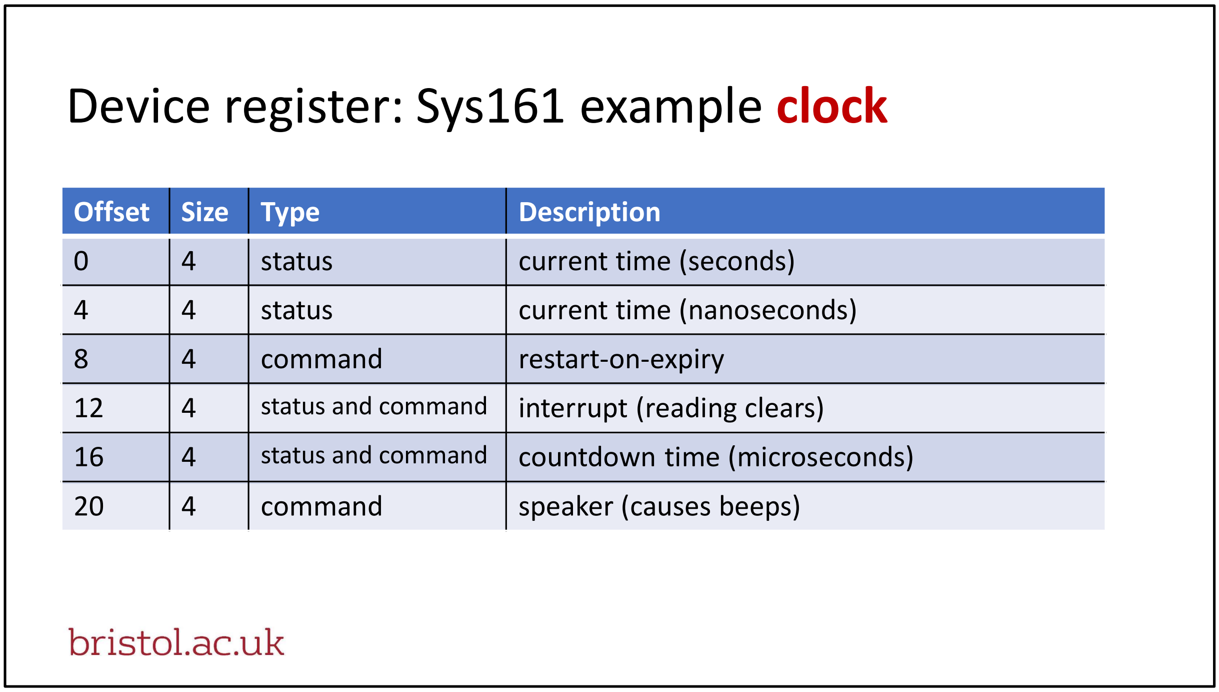
Terminology

- **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



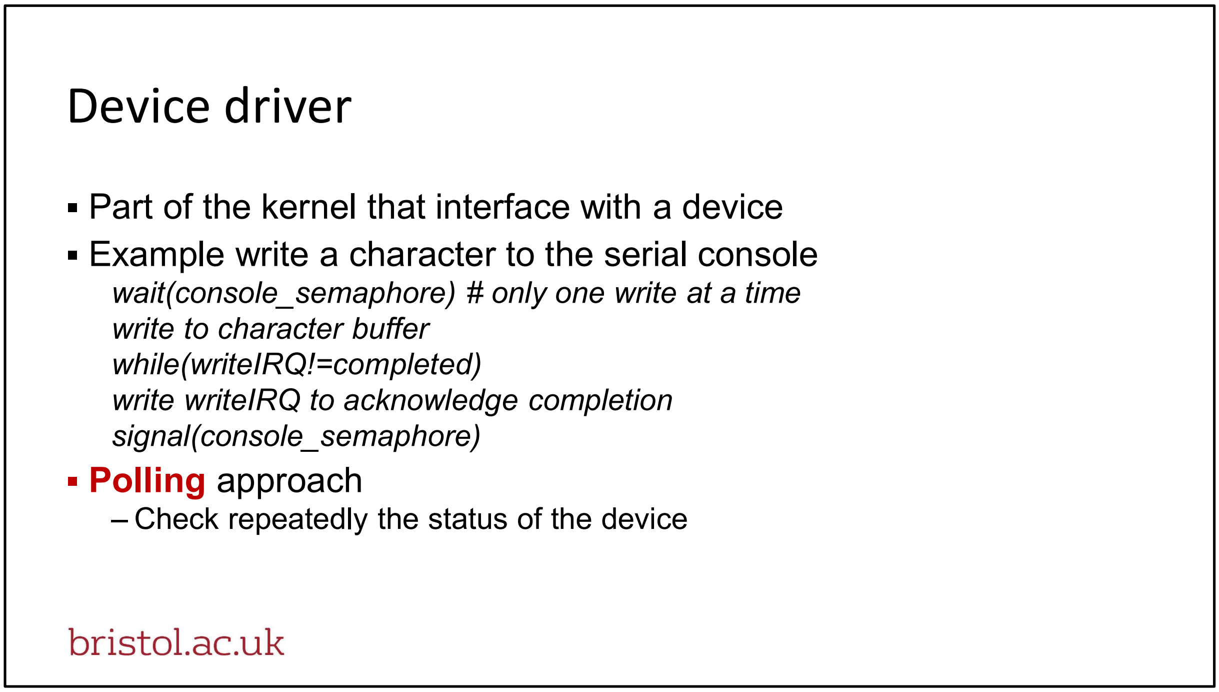
Device register: Sys161 example **clock**

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

IRQ: interrupt request

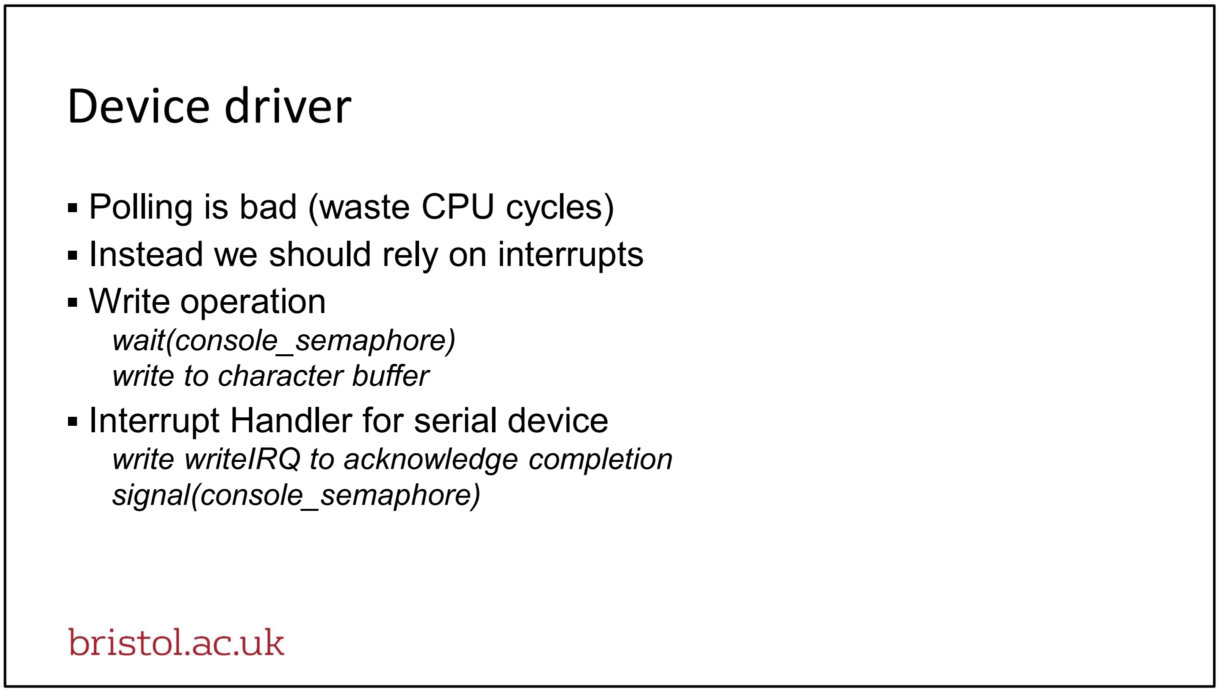


Device driver

- 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)
```

- **Polling** approach
  - Check repeatedly the status of the device



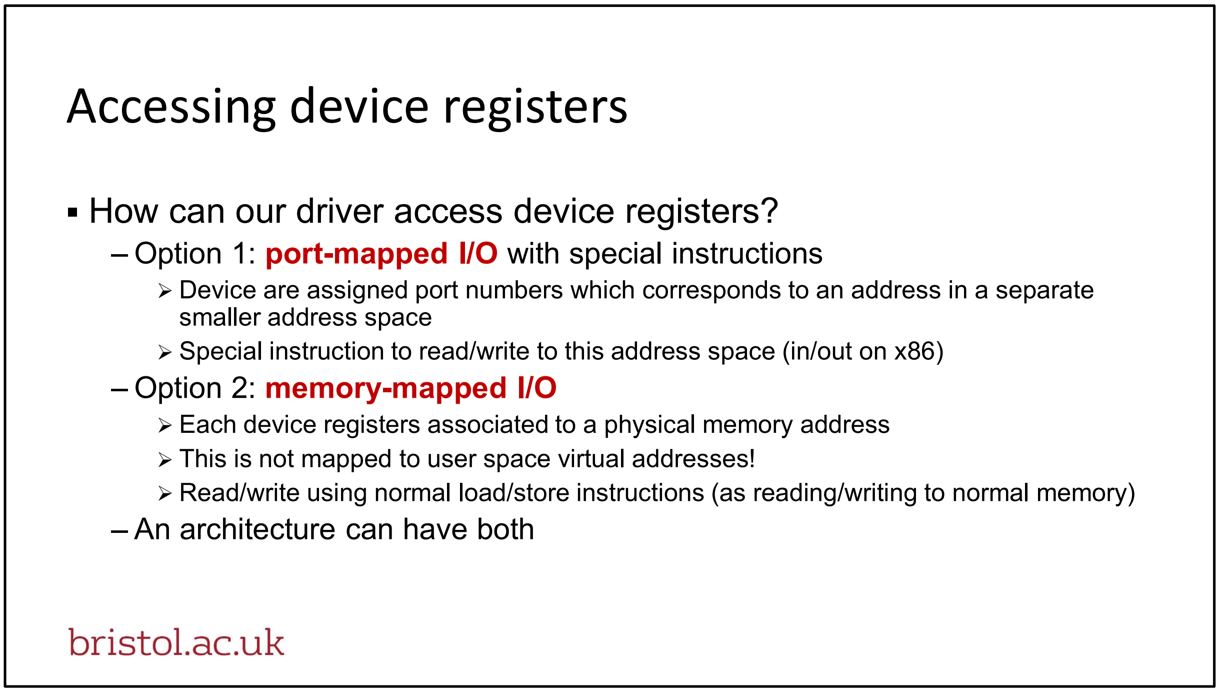
Device driver

- Polling is bad (waste CPU cycles)
- Instead we should rely on interrupts
- Write operation

  wait(console_semaphore)
  write to character buffer
- Interrupt Handler for serial device

  write writeIRQ to acknowledge completion
  signal(console_semaphore)

bristol.ac.uk

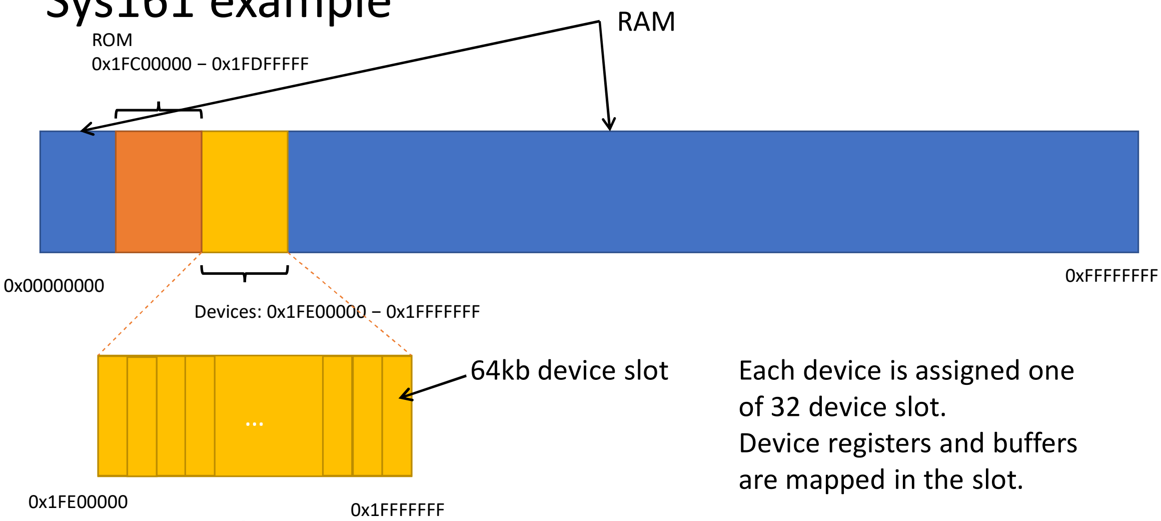


Accessing device registers

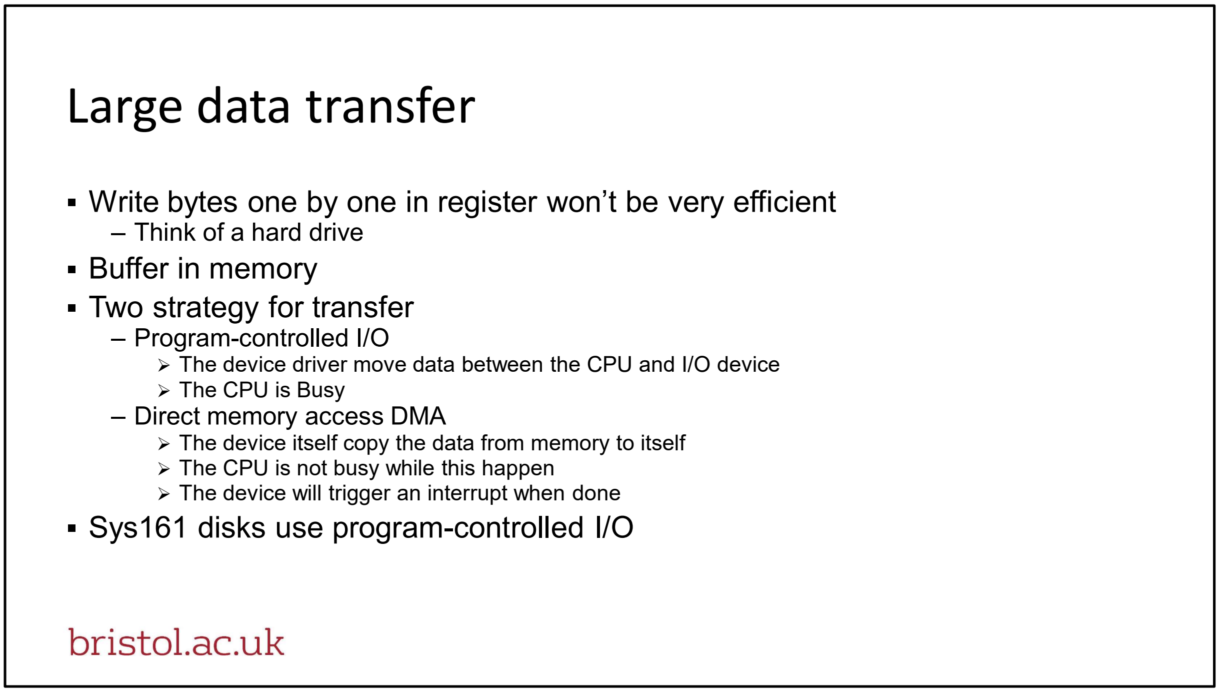
- How can our driver access device registers?
  - Option 1: **port-mapped I/O** with special instructions
    - Device are assigned port numbers which corresponds to an address in a separate smaller address space
    - Special instruction to read/write to this address space (in/out on x86)
  - Option 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

bristol.ac.uk

Sys161 example



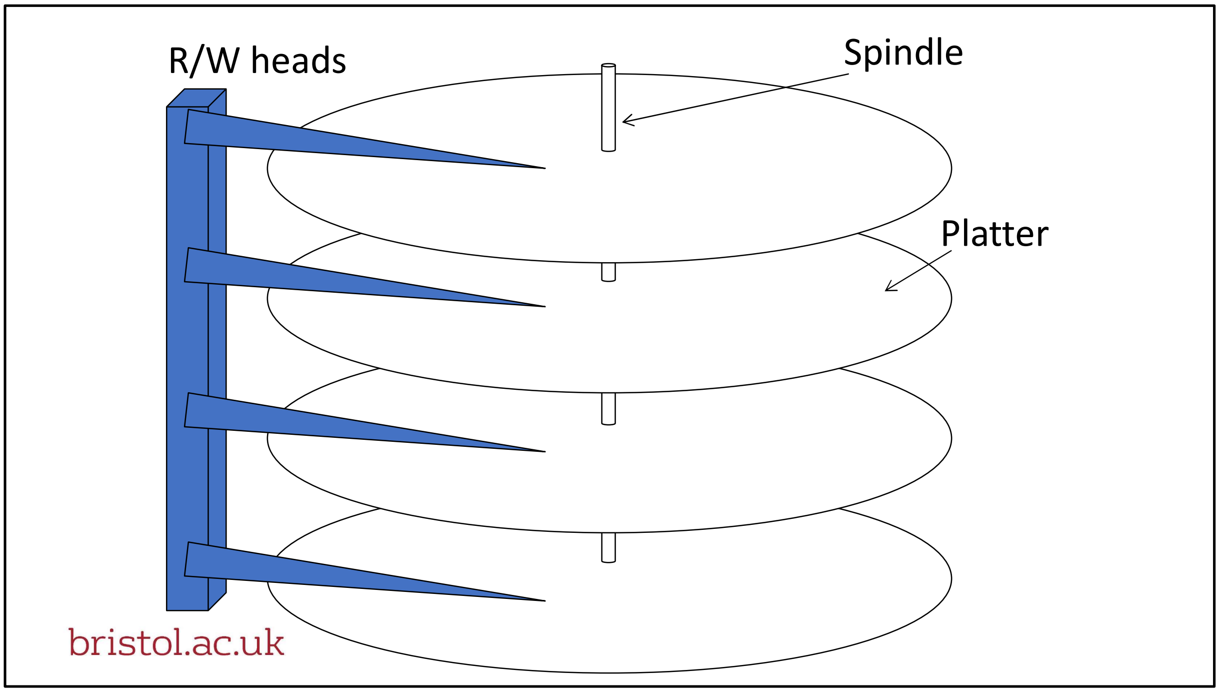
Each device is assigned one of 32 device slot.
Device registers and buffers are mapped in the slot.



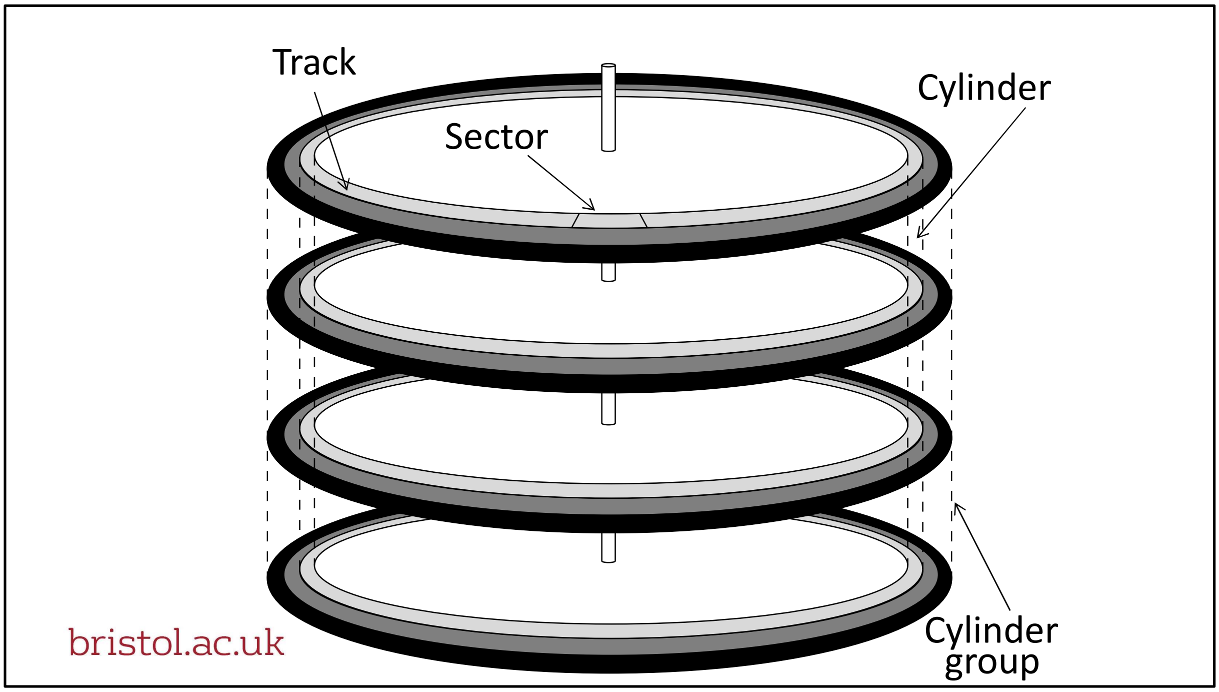
Large data transfer

- 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

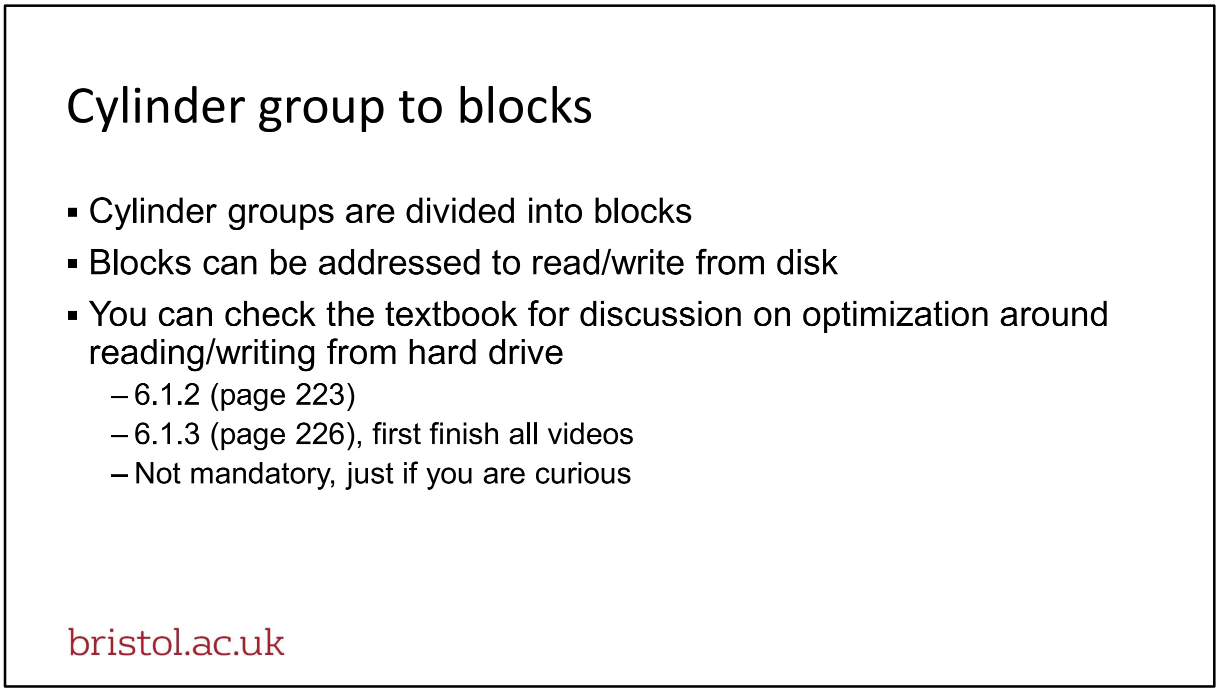
bristol.ac.uk



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.



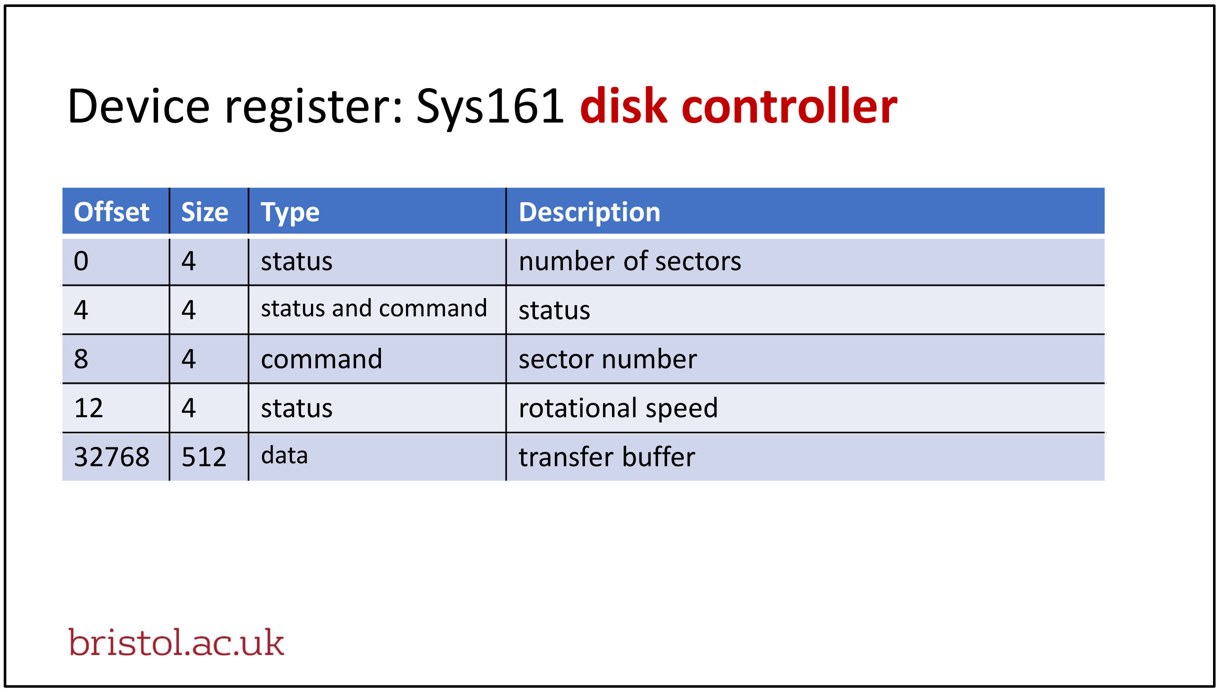
For a long time, hard disks used a sector size of 512. However, modern disks now use a sector size of 4K.



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

bristol.ac.uk



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

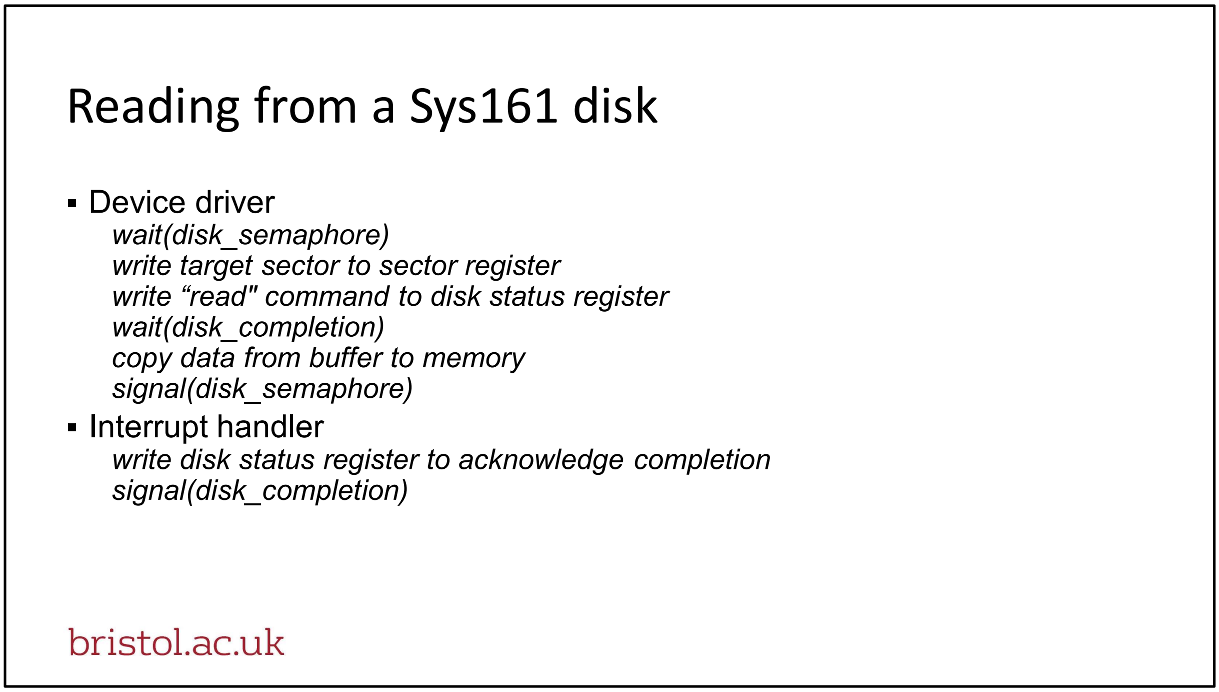
Writing to a Sys161 disk

- 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)
```



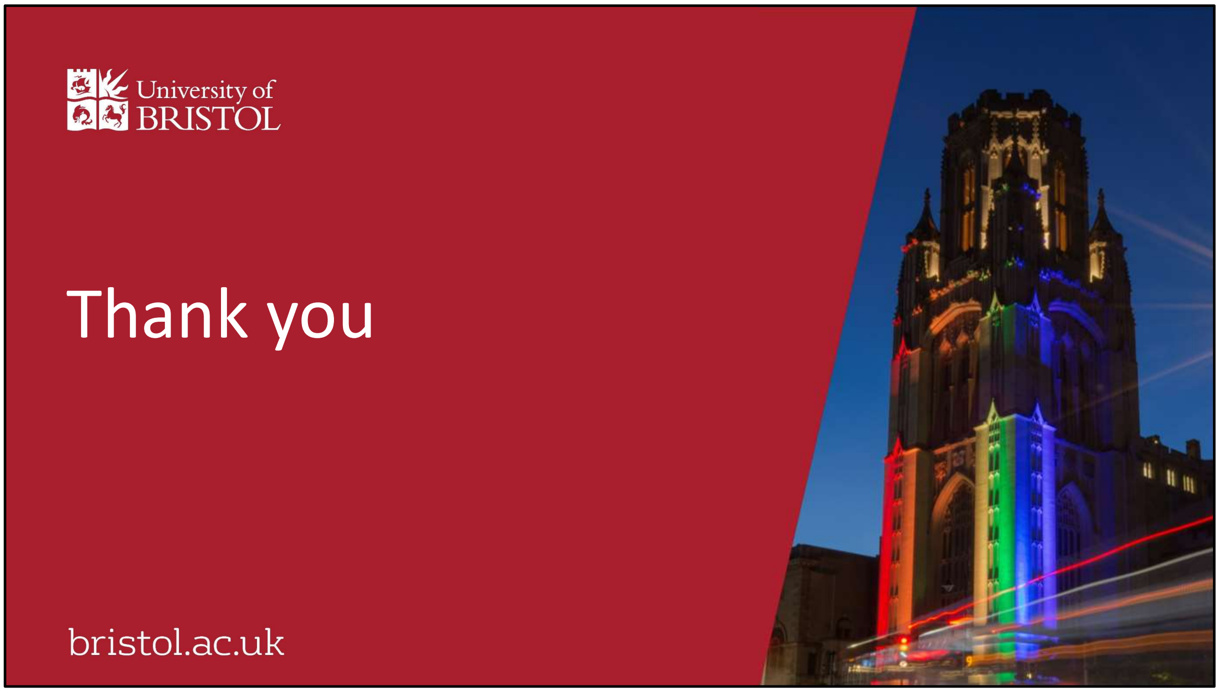
Reading from a Sys161 disk

- 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

Thank you

bristol.ac.uk