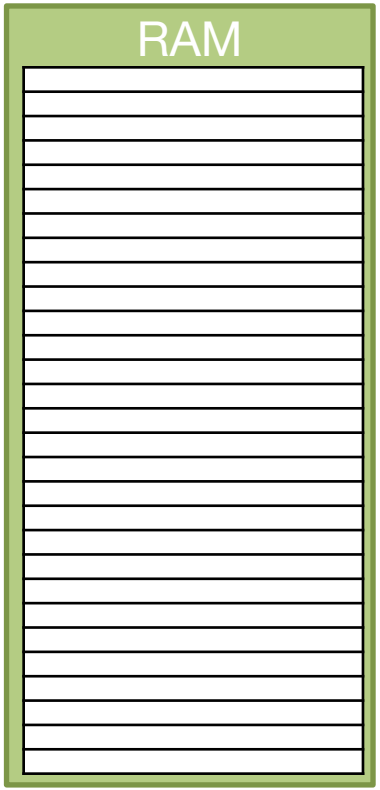
Virtual Memory III and I/O

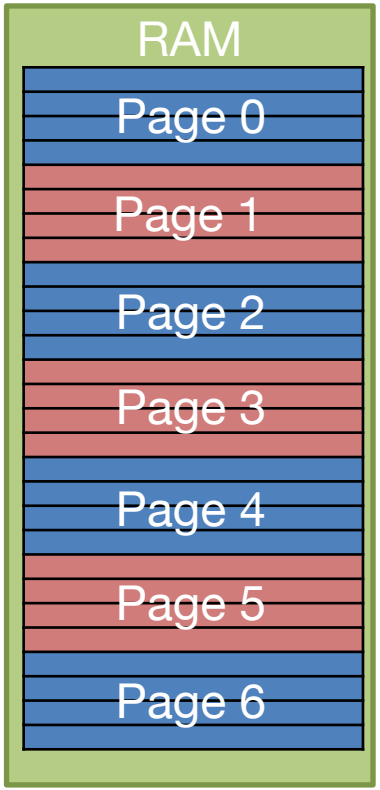
Review

- RAM = Main Memory
- The RAM can contain code and data
- To access a piece of data or code, we need to load it from disk into the RAM
- The size of our RAM is smaller than our disk, so we may not be able to fit all of the data that we need in RAM at the same time
- This means that we sometimes need to move data between the RAM and the disk

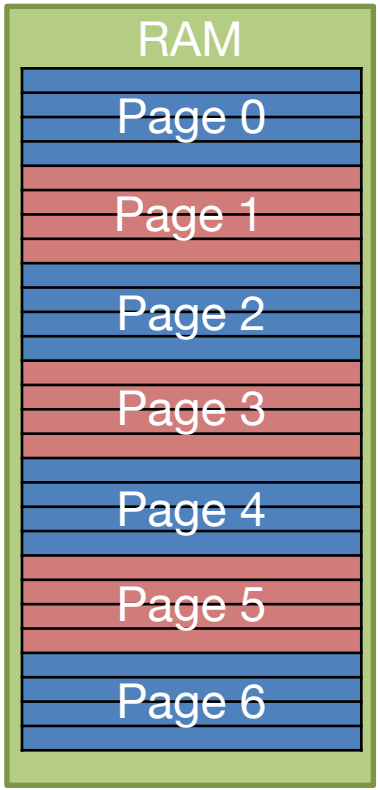


Review

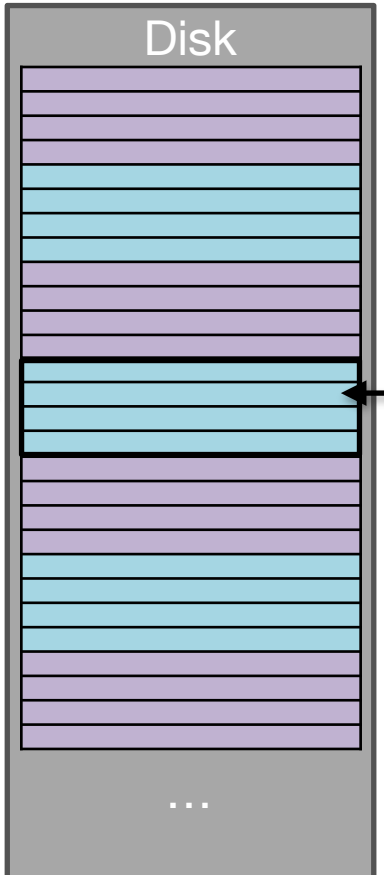
- To make it easy to transfer data between the disk and the RAM, we standardize the granularity at which we can move data
- The granularity that we use is called a page
- The size of a page is typically 4KB
- This means that if I want to access a piece of data that is not in the RAM (let's say I want to access an integer), then I will need to bring in the page where the integer resides to the RAM



Review



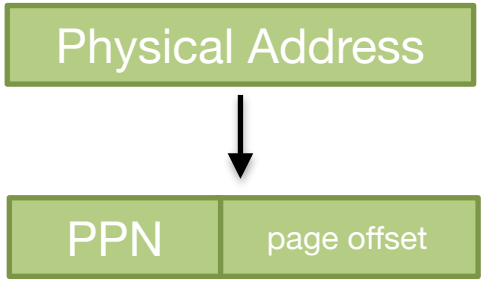
I will bring this whole page into RAM

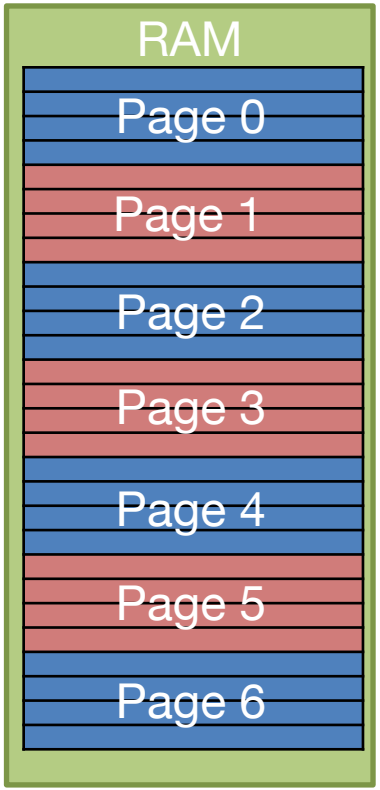


If the integer that I want to access is here

Review

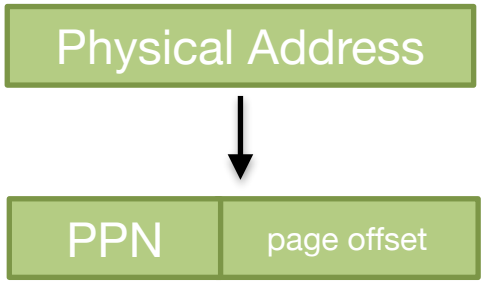
- Each of these pages has a page number (aka Physical Page Number or PPN)
- When we want to access a piece of data, we go to the page where it is located
- Then we use an offset to determine where in the page the data is located

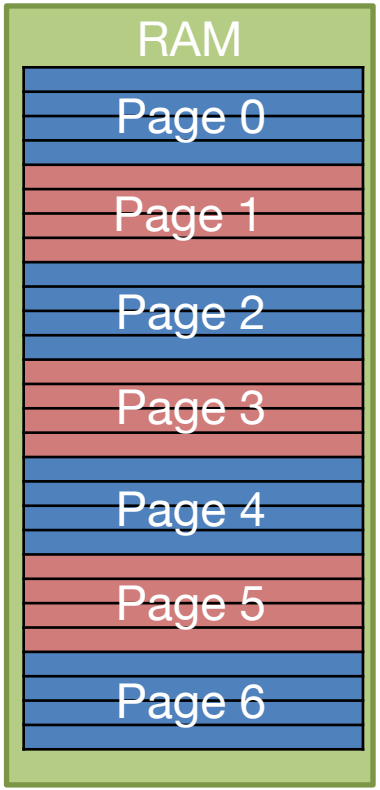




Review

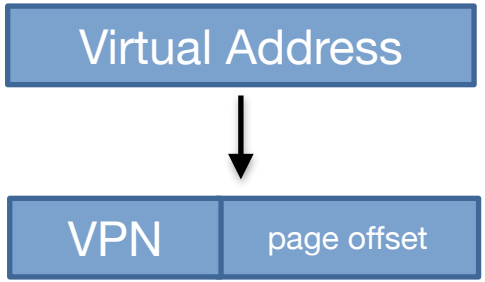
- How many bits are in the PPN?
  - $\log_2$(num pages)
- How many bits are in the page offset?
  - $\log_2$(page size)

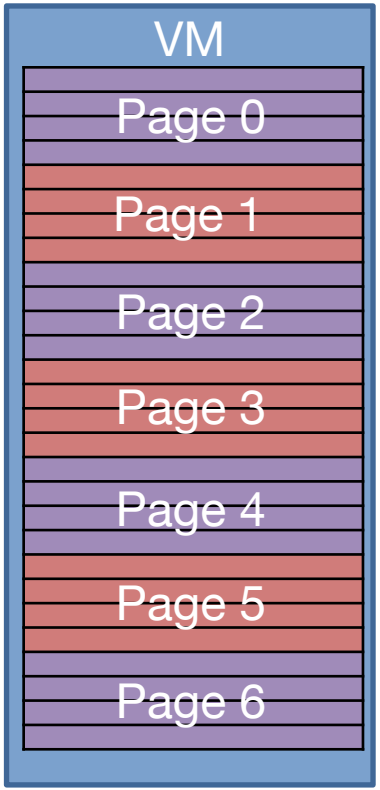




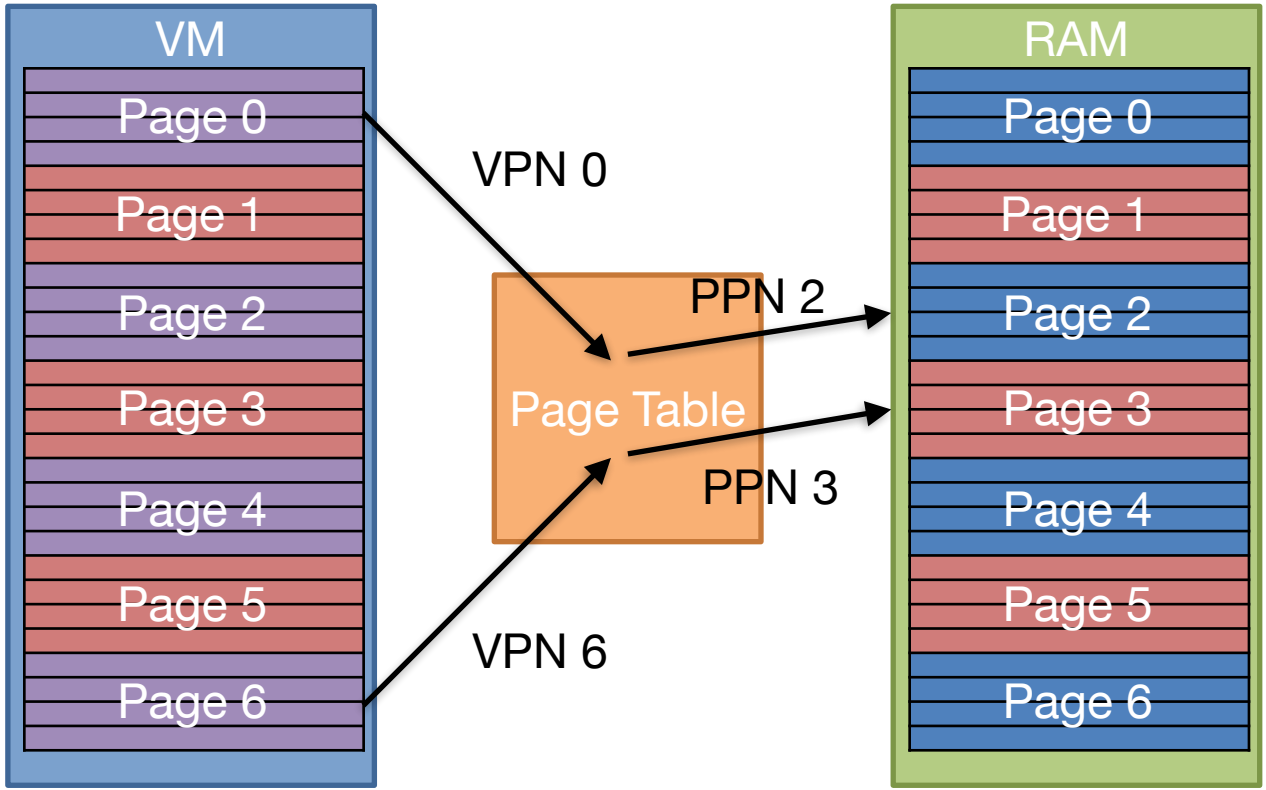
Review

- Virtual memory is imaginary
- It is the way that a program **thinks** that the RAM is laid out
- It is also split up into pages (because this is how the RAM is split up)
- We use a map to assign virtual pages to physical ones
- The map is called a page table





Review

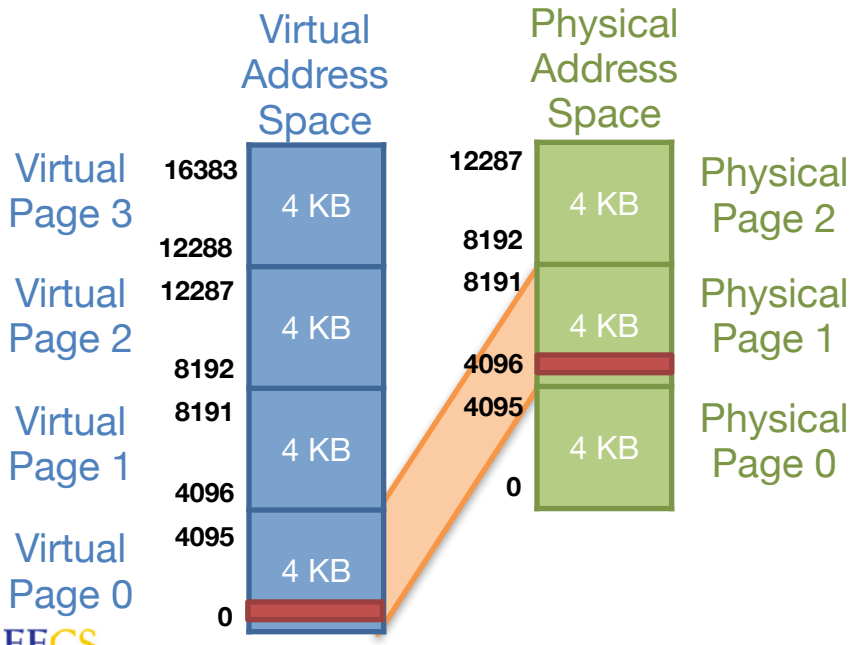

VM
Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
VPN 0
VPN 6
Page Table
PPN 2
PPN 3
RAM
Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6

Review

- Why do we have virtual memory?
  1. Our RAM is not big enough to hold all of the data that we need at one time
  2. Our program needs to “think” that the data is contiguous
     - When you are writing a program, you don’t know where it will be stored in memory
     - So we just pretend like the memory is contiguous and then the virtual memory takes care of everything in the background
  3. Protection between processes

Review

- The page table holds the mapping of VPNs to PPNs
  - It does not hold any data!



Page Table

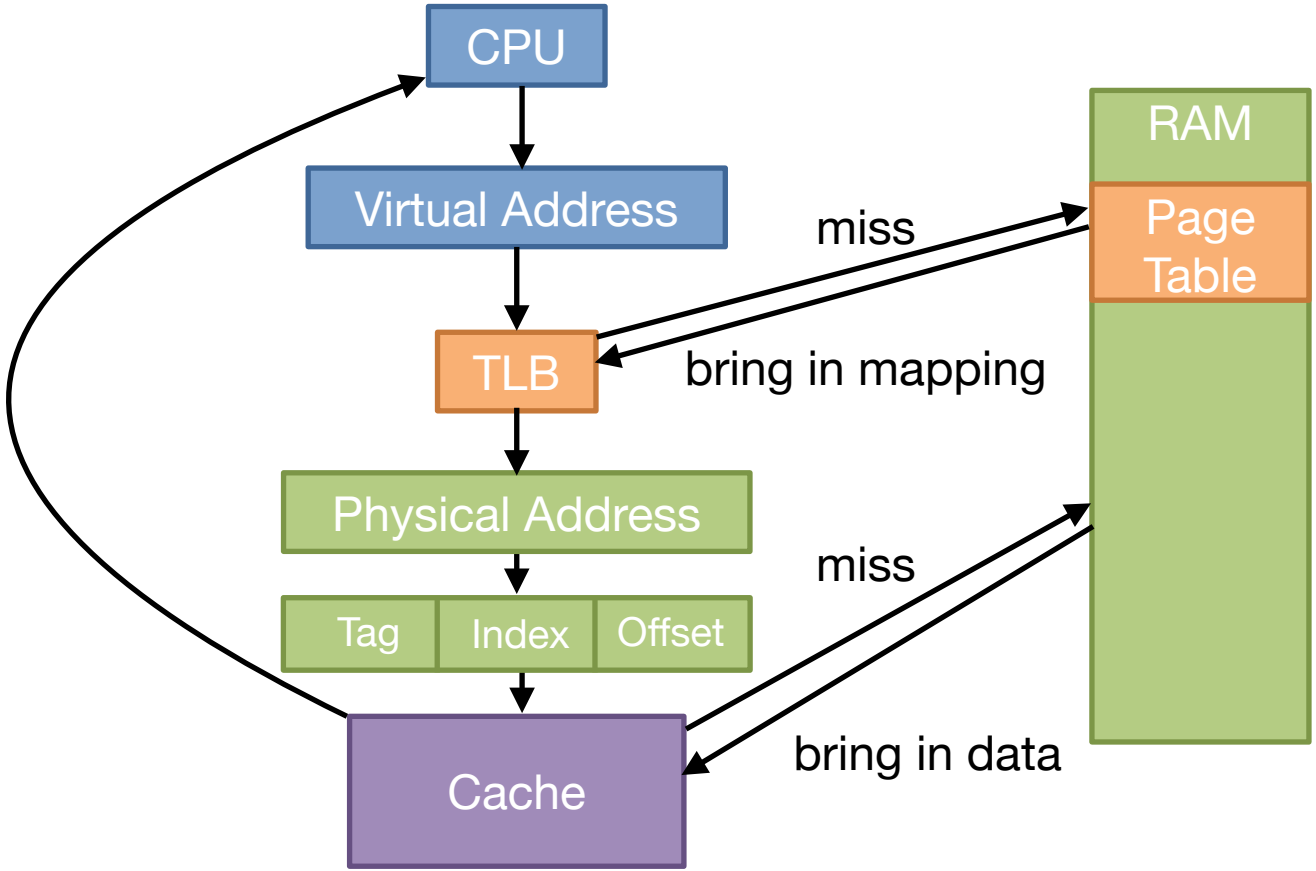
VPN	PPN
0b00	0b01
0b01	Disk
0b10	0b11
0b11	0b10

Review: Overall Idea

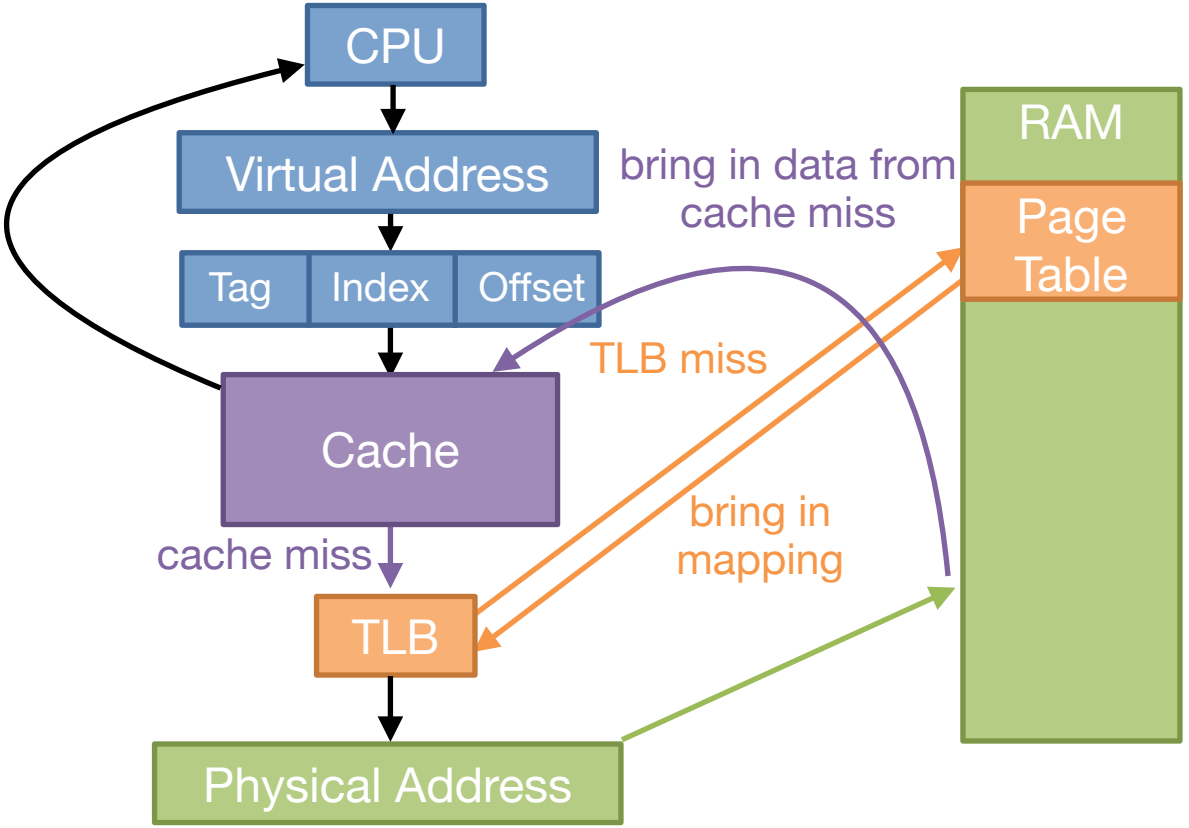
- A program consists of a bunch of memory that stores the code and data
- This memory is split up into chunks called pages
- These pages can be found in either the RAM (main memory) or the disk
  - Why is it not always in the RAM?
    - Not enough space

Caches and Virtual Memory

Physically Indexed, Physically Tagged Cache

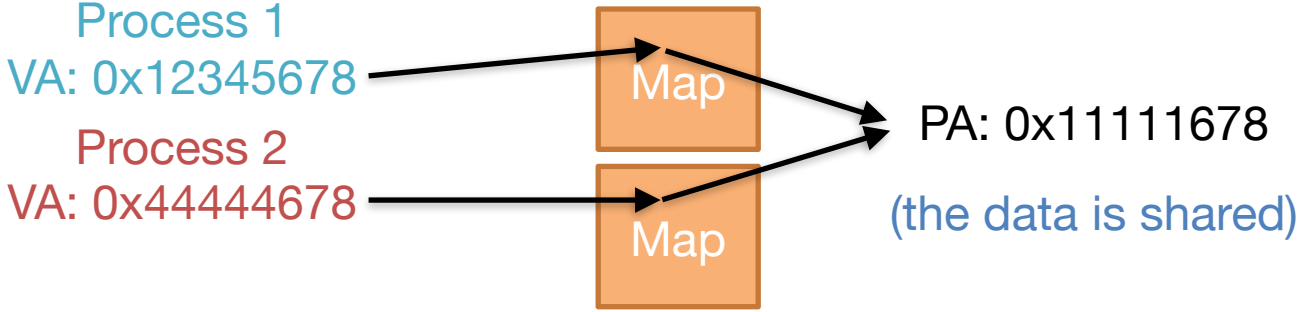


Virtually Indexed, Virtually Tagged Caches

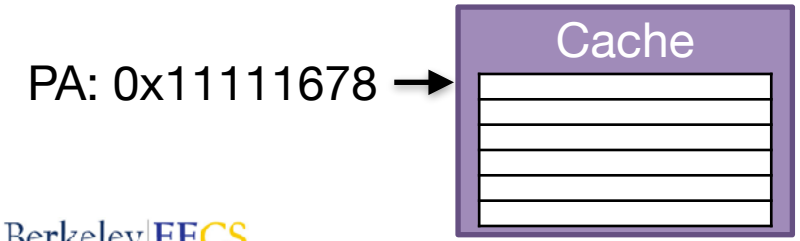


Synonyms

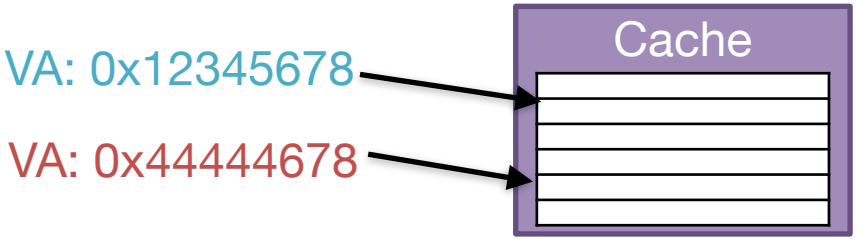
Different virtual addresses can map to the same physical address



PIPT Cache: No problem!

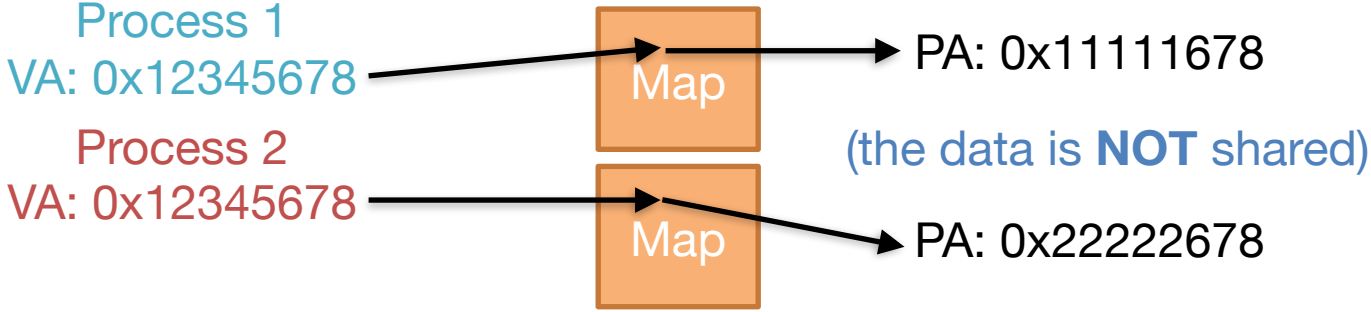


VIVT Cache: two inconsistent copies

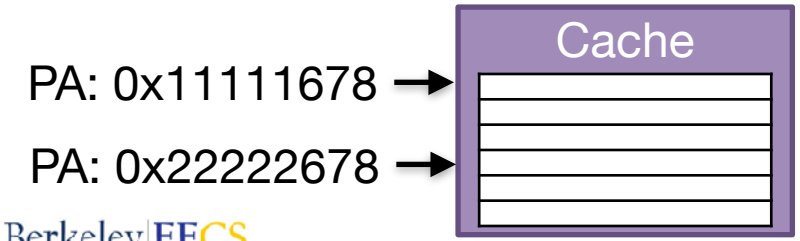


Homonyms

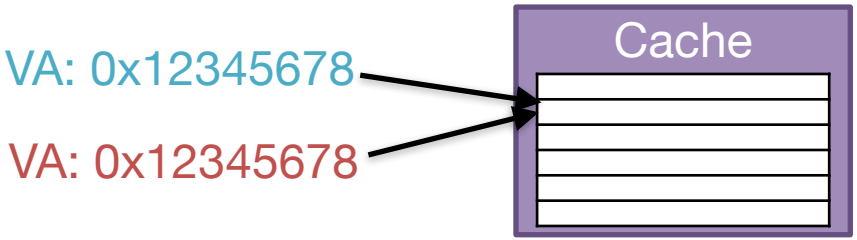
The same virtual address can map to different physical addresses



PIPT Cache: No problem!



VIVT Cache: the processes are overriding each other's data



PIPT vs VIVT

PIPT

- Multiple processes can share the same cache
- Must translate the address before accessing the cache

VIVT

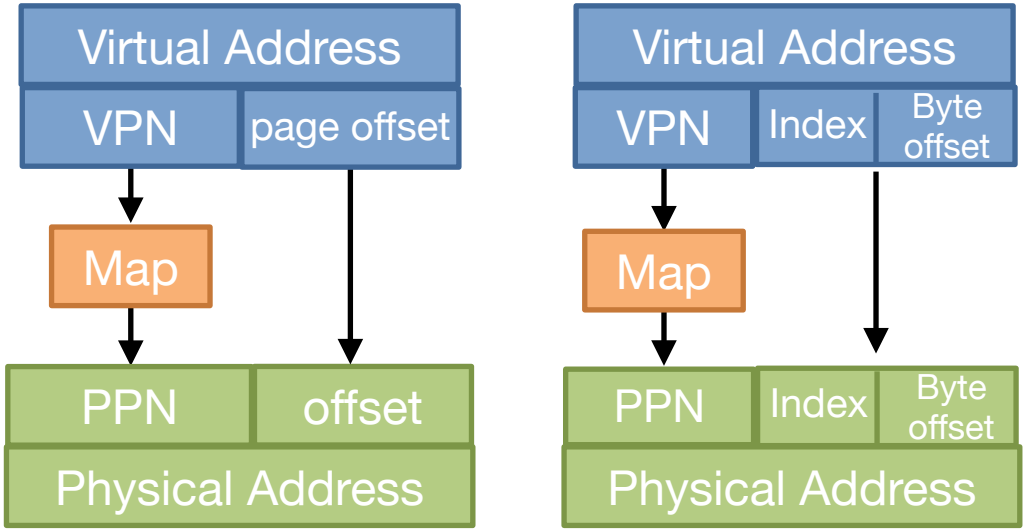
- Multiple processes cannot share the same cache
  - must flush cache on context switch
- Do not need to translate the address before accessing the cache

Virtually Indexed, Physically Tagged Caches

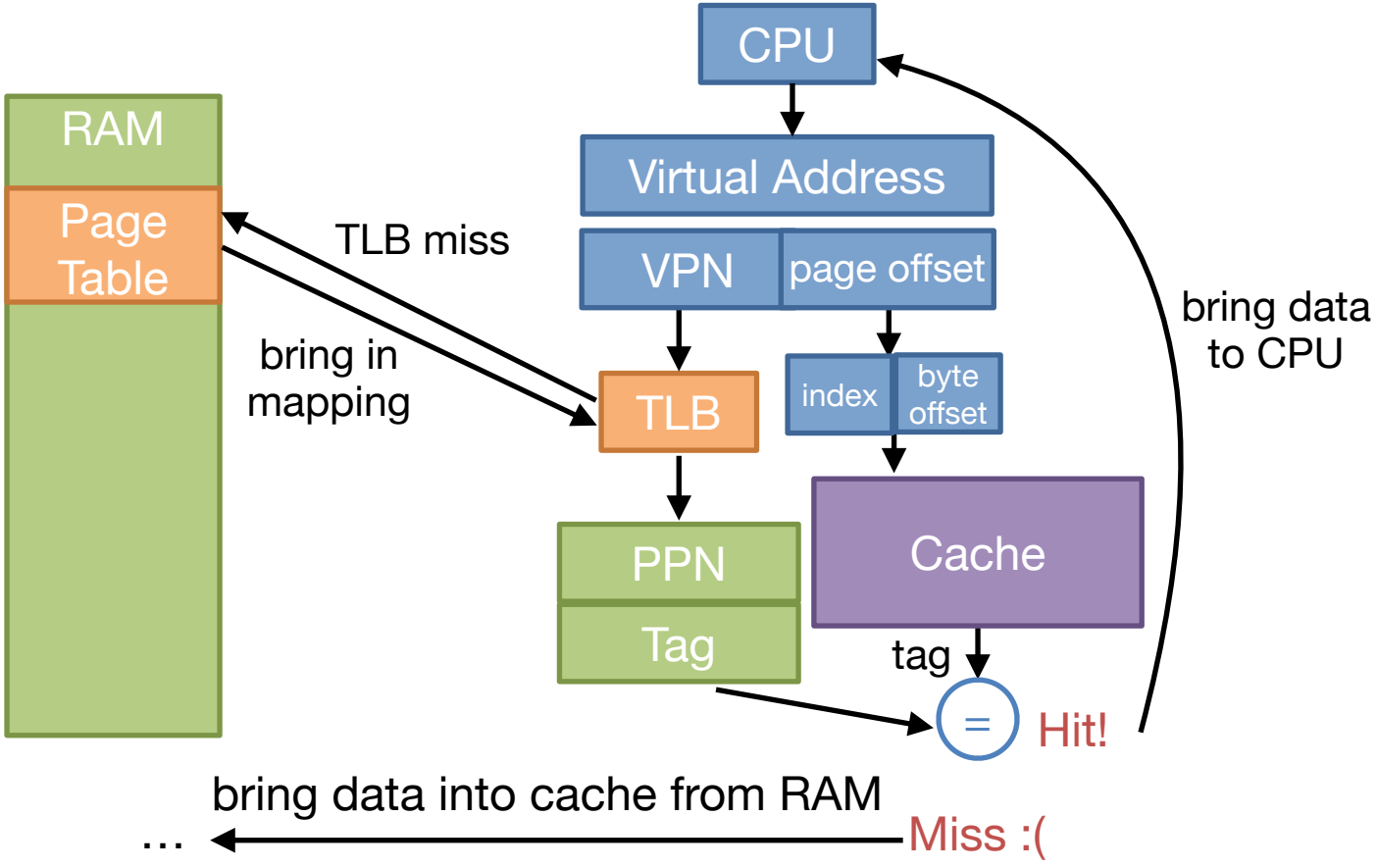
- We want to have a physical cache so that we don't have to flush it on a context switch
- Translating the address takes extra time
- We would like to speed up the access time
- How can we prevent translating the address from being in the critical path?

How to make this work?

- Which bits remain the same after the translation?
  - The page offset bits



Virtually Indexed, Physically Tagged (VIPT) Caches



VIPT Cache Size

- The size of our cache is limited by the number of page offset bits
- Q: With 4kB pages, how many bytes can a direct-mapped (1-way) VIPT cache store?

VIPT Cache Size

- The size of our cache is limited by the number of page offset bits
- Q: With 4kB pages, how many bytes can a direct-mapped (1-way) VIPT cache store?
  - 4kB
  - We can only use the page offset bits (12 bits for 4kB pages) to index into the cache. So the index can only address 12 bits of address, or 4kB of data

VIPT Cache Size

- The size of our cache is limited by the number of page offset bits
- Q: With 4kB pages, how many bytes can a direct-mapped (1-way) VIPT cache store?
  - 4kB
  - We can only use the page offset bits (12 bits for 4kB pages) to index into the cache. So the index can only address 12 bits of address, or 4kB of data
- How can we make our cache larger with the same page size?

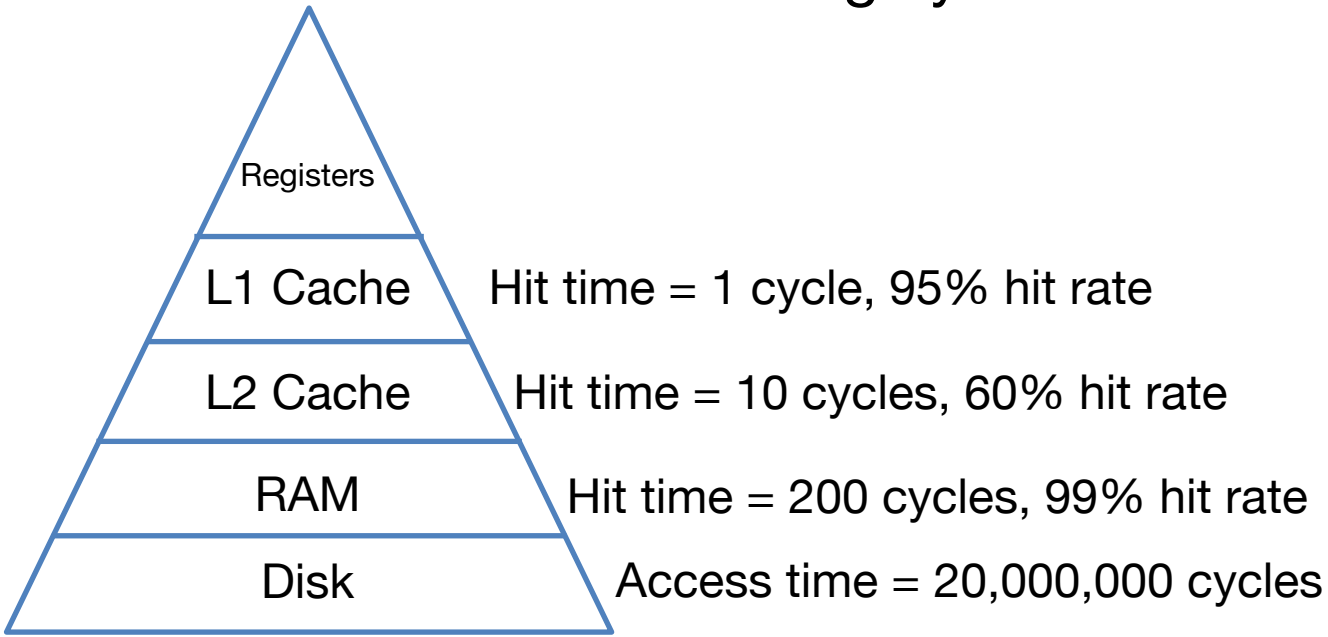
VIPT Cache Size

- The size of our cache is limited by the number of page offset bits
- Q: With 4kB pages, how many bytes can a direct-mapped (1-way) VIPT cache store?
  - 4kB
  - We can only use the page offset bits (12 bits for 4kB pages) to index into the cache. So the index can only address 12 bits of address, or 4kB of data
- How can we make our cache larger with the same page size?
  - Increase the associativity

Memory Access Time

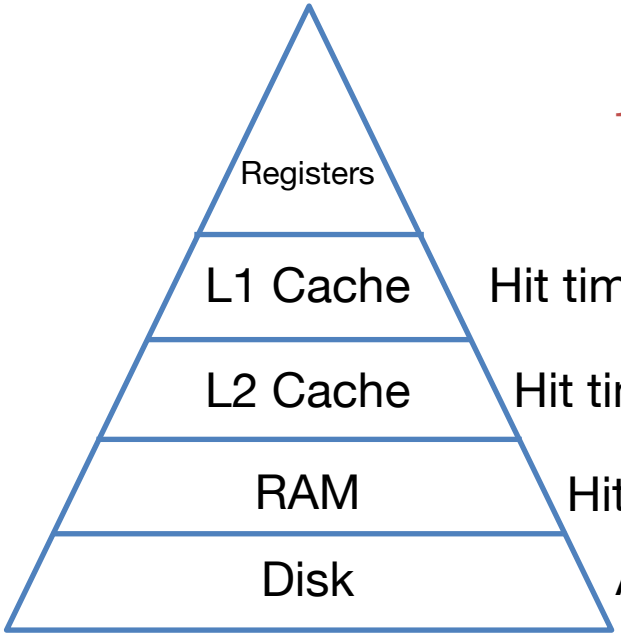
Average Memory Access Time (AMAT)

- AMAT = hit time + miss rate * miss penalty
- What is the AMAT of the following system?



Average Memory Access Time (AMAT)

- AMAT = hit time + miss rate * miss penalty
- What is the AMAT of the following system?



Hit time = 1 cycle, 95% hit rate

Hit time = 10 cycles, 60% hit rate

Hit time = 200 cycles, 99% hit rate

Access time = 20,000,000 cycles

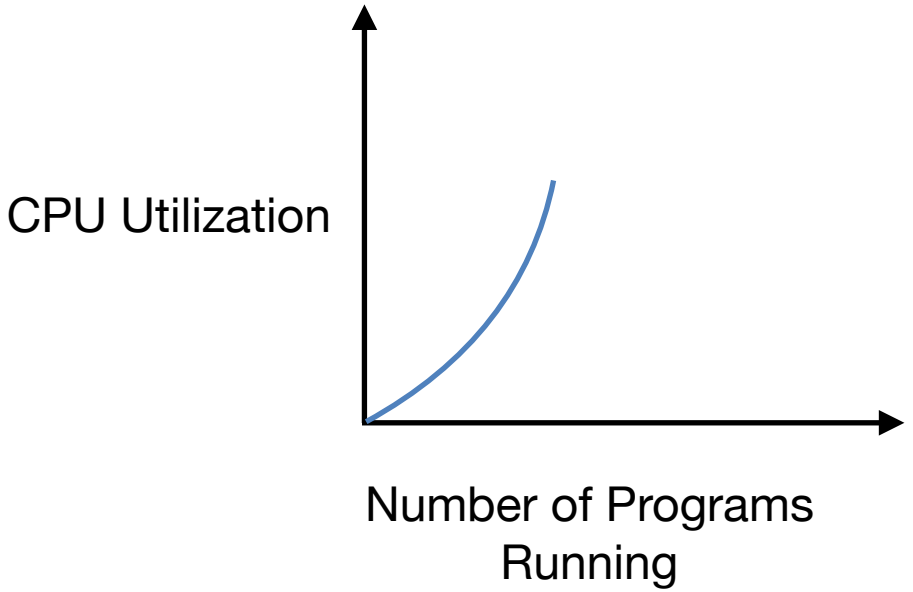
1 + .05(10 + .4(200 + .01(20M))) = 4005.5 cycles

this is why we cortex switch if data is in disk

Thrashing

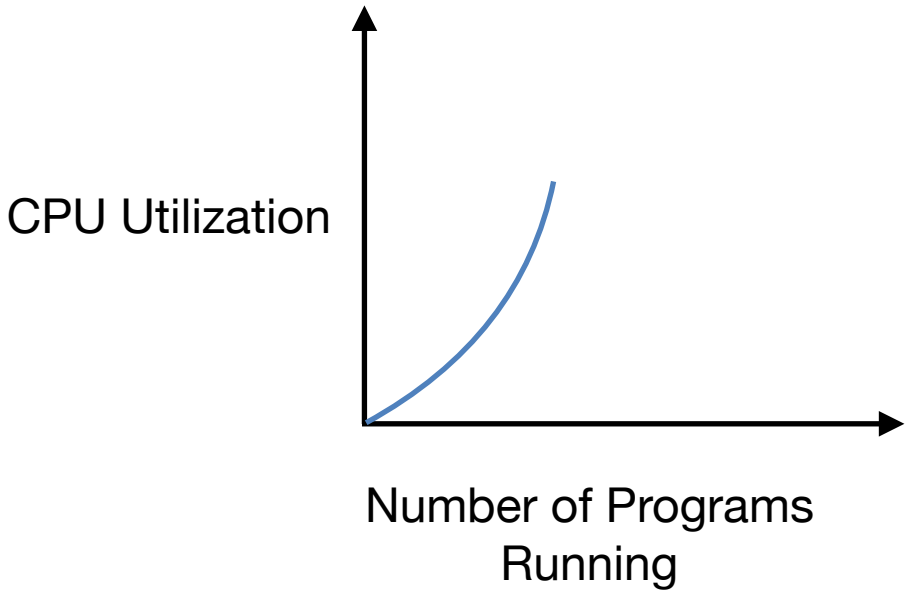
CPU Utilization

- When processes perform I/O, it can take a long time
- During this time, the CPU can schedule another process, so that it is not idle
- The more processes there are, the better CPU utilization



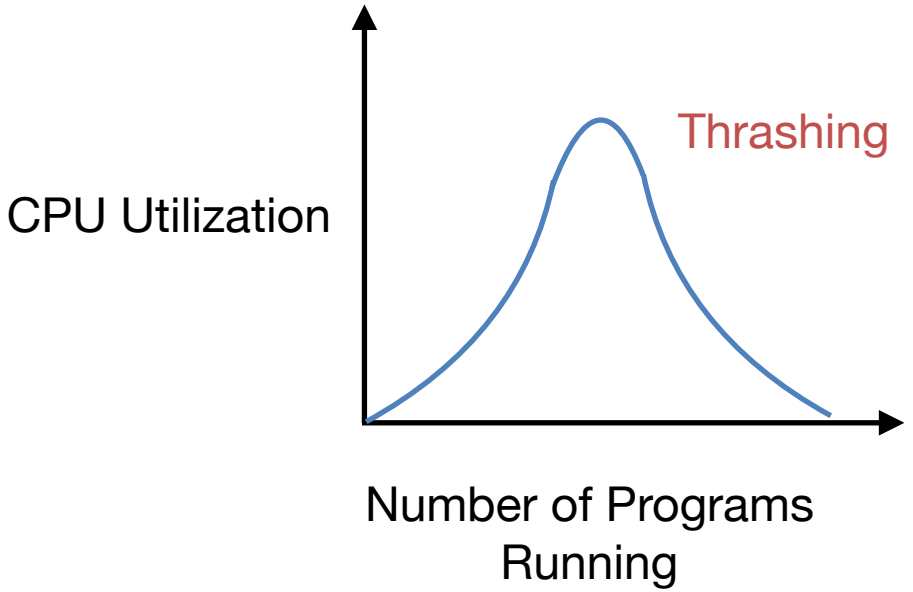
CPU Utilization

- For example, if we need to access a page of a process that is not in RAM, we need to bring it in from the disk
- This takes a long time
- So the CPU can schedule another process while this is happening



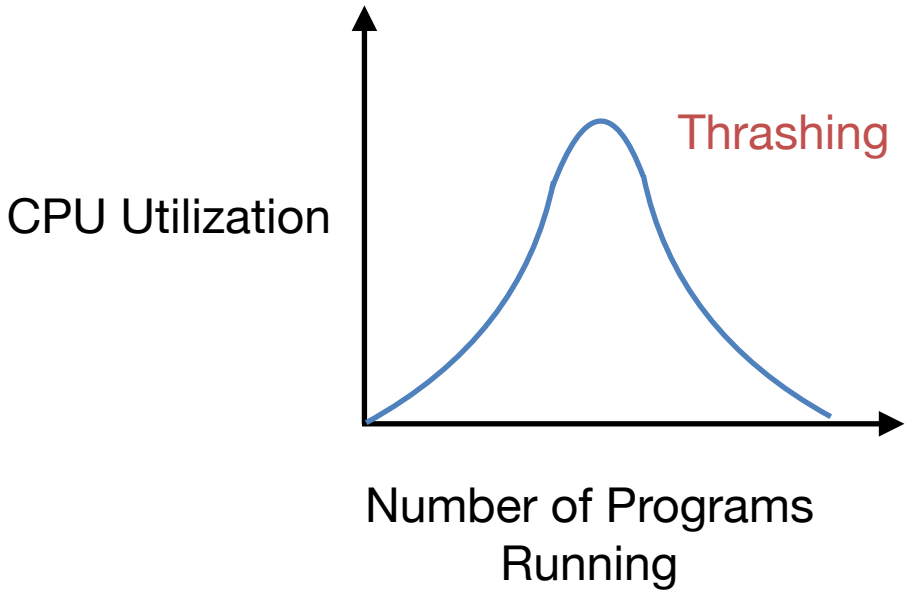
CPU Utilization

- When adding more processes, you will eventually reach a large number of accesses are page faults
- The CPU utilization goes down because the processes it tries to run page fault more frequently



CPU Utilization

- Thrashing
  - When the processor encounters page faults so frequently, that the time spent resolving the page faults overwhelms the time spent performing the computations



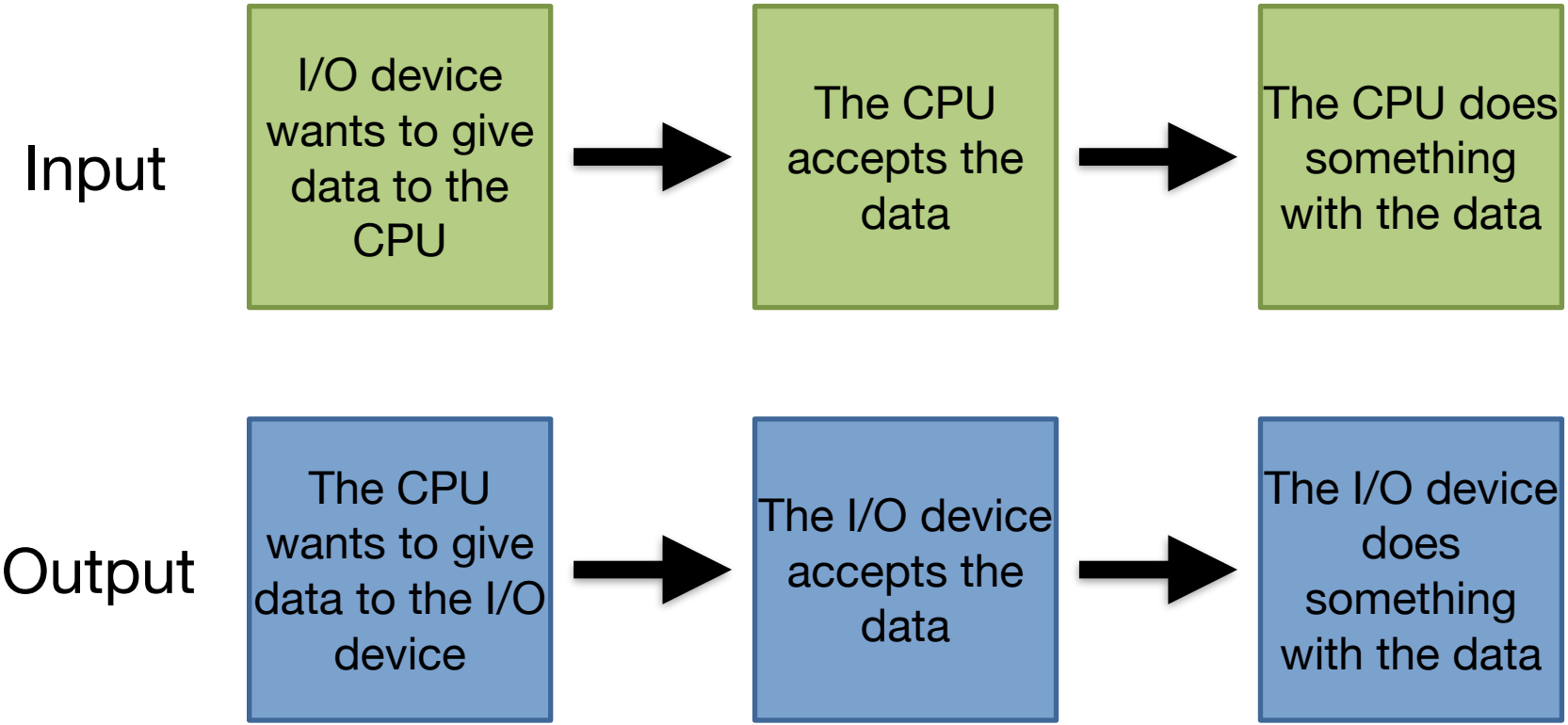
Cat Break

Input/Output

I/O Device Examples

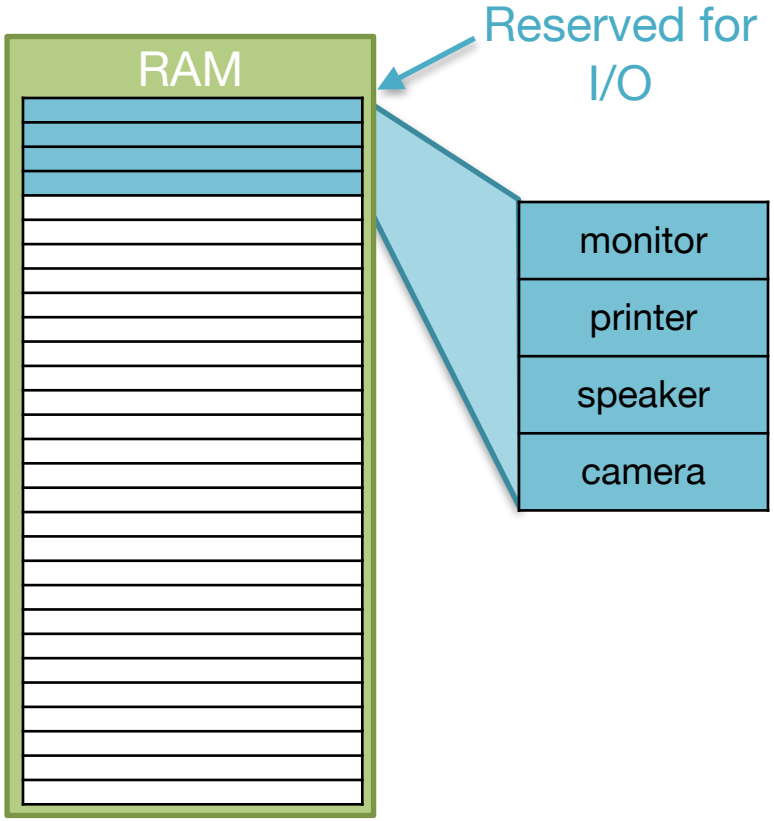
Name	*Type*	*Where*
magnetic disk drive	in/out	servers, desktops
solid-state drive (SSD)	in/out	servers, desktops, laptops, handheld
display	out	desktops, laptops, handheld
keyboard	in	desktops, laptops
speakers / headphones	out	desktops, laptops, handheld
microphone	in	desktops, laptops, handheld, embedded
mouse	in	laptops
video camera	in	desktops, laptops, handheld, embbed
printer	out	desktops, laptops
touch screen	in/out	laptops, handheld

Communication with the CPU



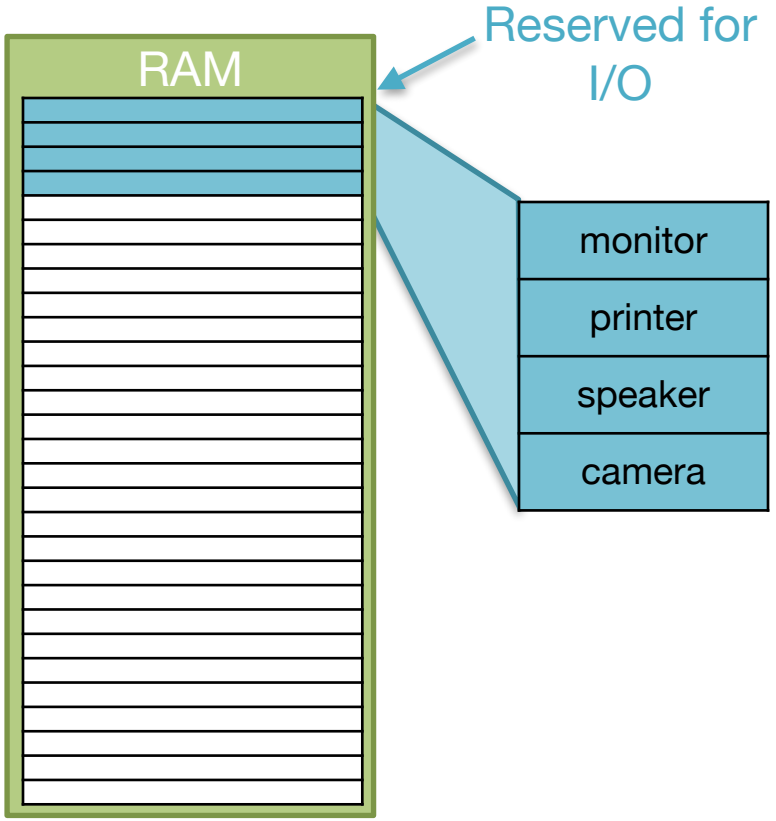
Memory Mapped I/O

- A portion of memory is dedicated for communicating with I/O devices
- The addresses in this area correspond to different I/O devices
- When the CPU wants to send information to an I/O device, it "writes" to this location
- This will write the data to the device that corresponds to that address

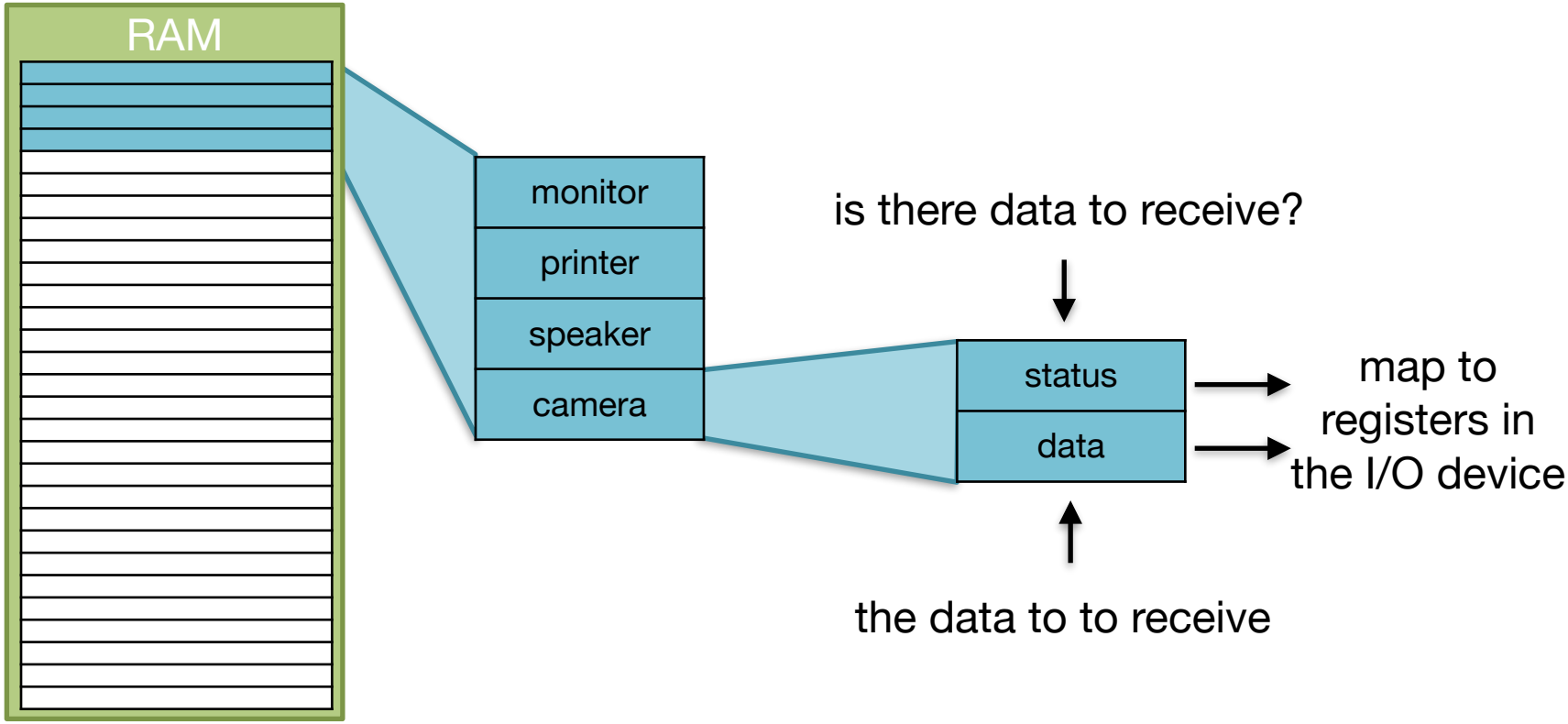


Memory Mapped I/O

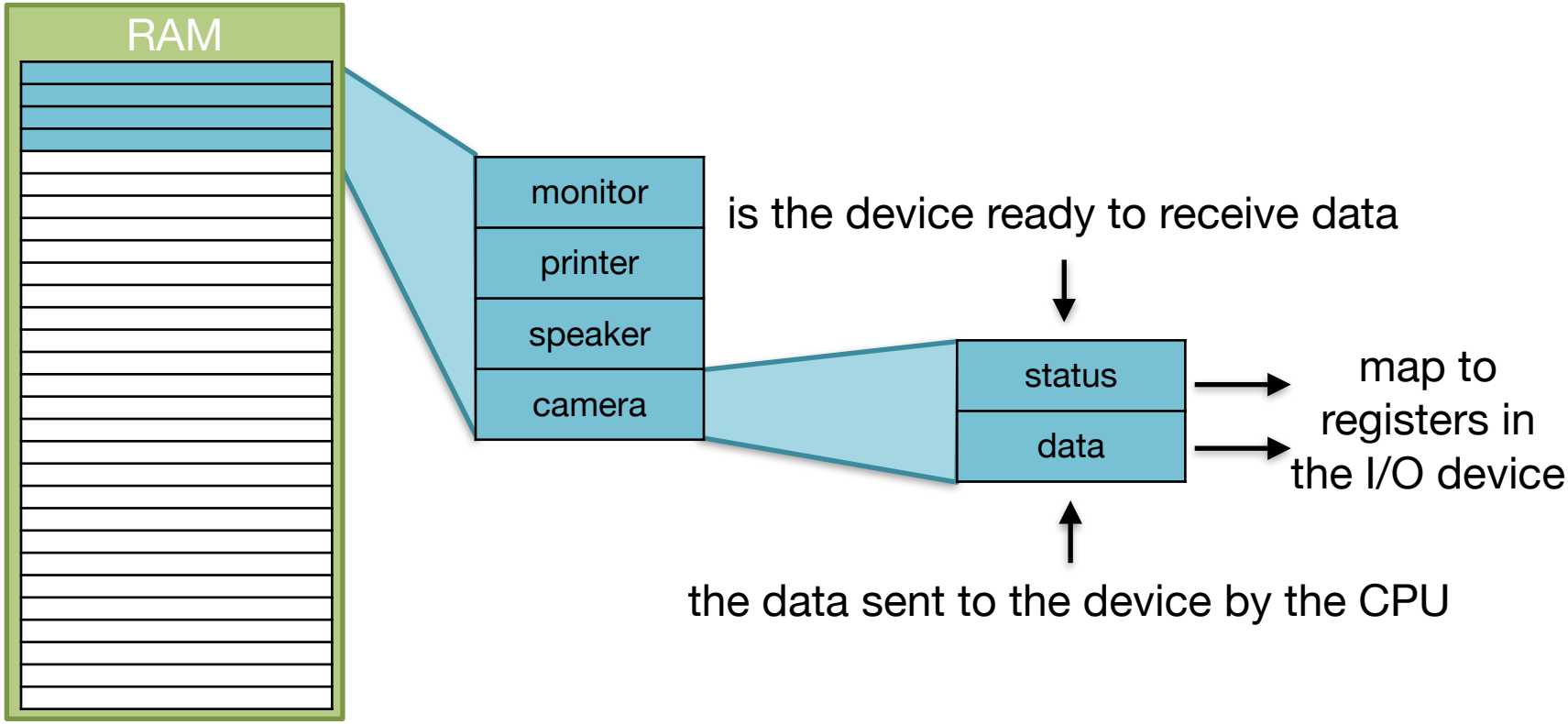
- When the CPU wants to send information to an I/O device, “writes” to this location
- The hardware will intercept this write and will write the data to the device that corresponds to that address
- No actual memory access is performed



Memory Mapped I/O (Inputs)

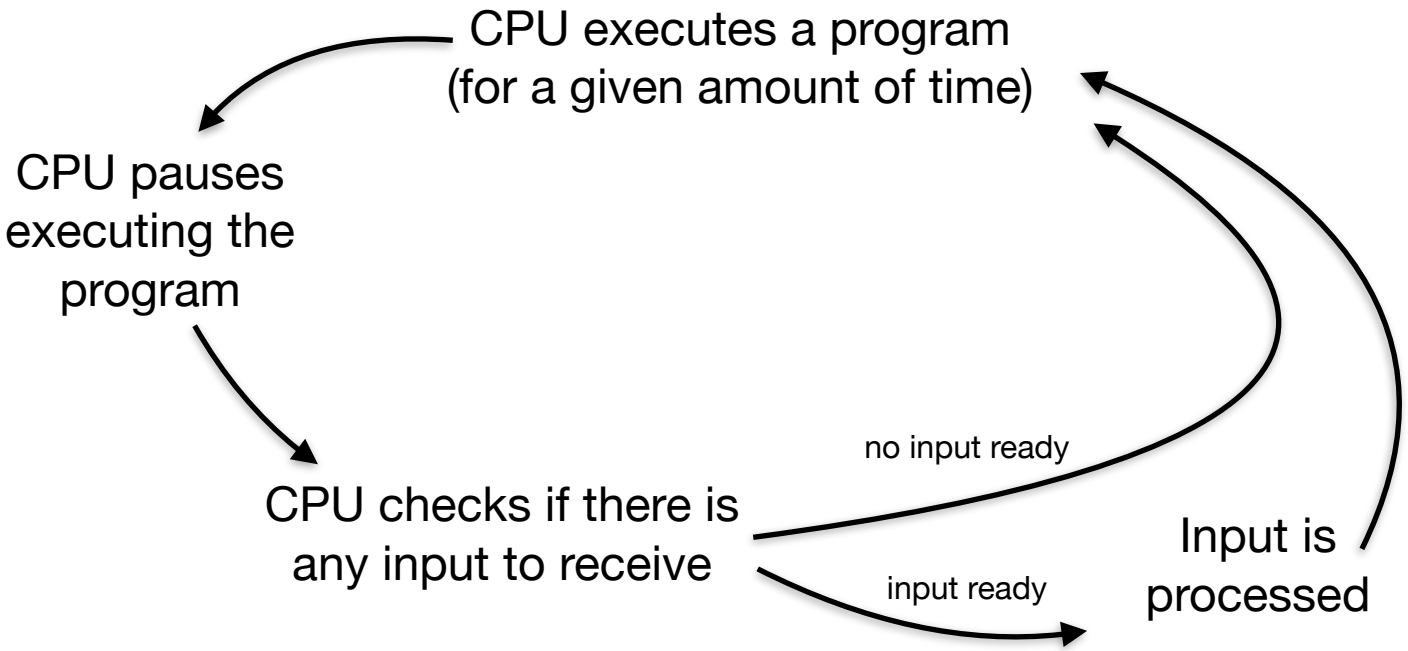


Memory Mapped I/O (Outputs)



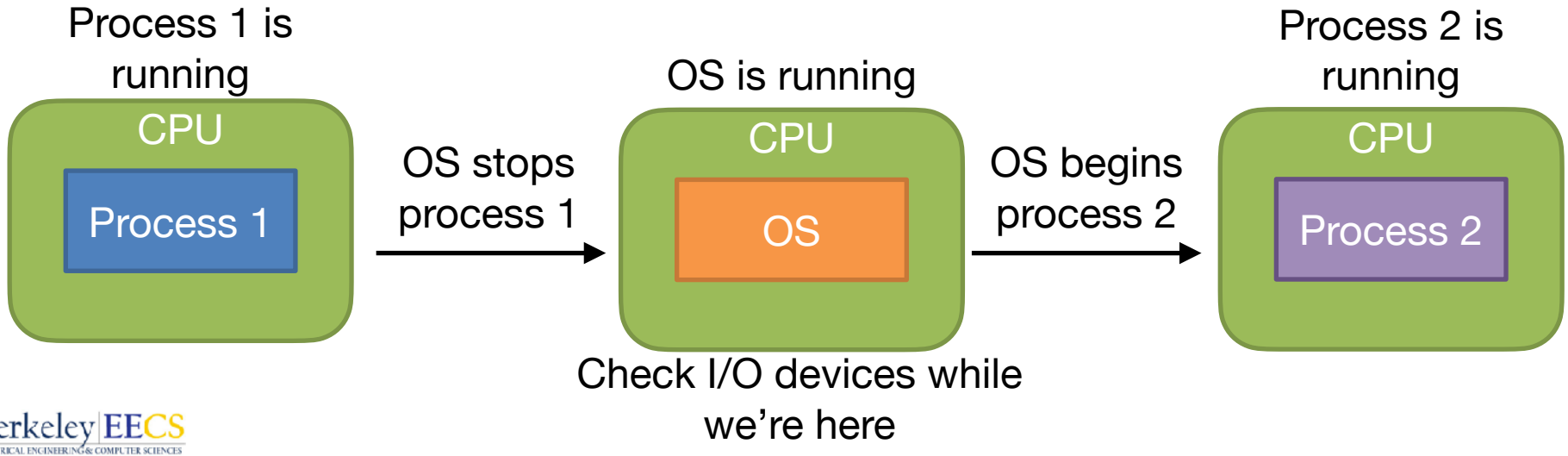
Polling

- The CPU periodically checks the status register to see if there is any data to receive



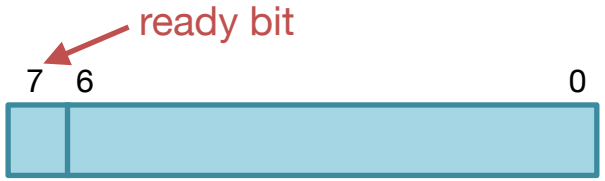
Polling

- When does polling occur?
  - Usually while we are already performing a context switch
- We periodically switch between processes every few milliseconds
- Every time we switch, we can check our I/O devices

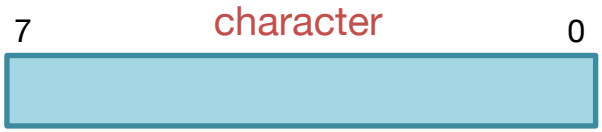


Basic Keyboard and Display Example

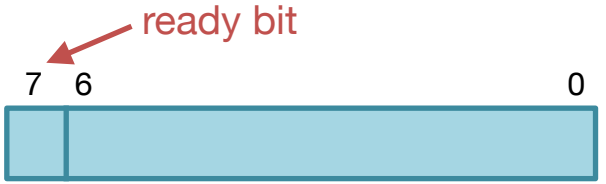
Keyboard Status Register



Keyboard Data Register



Display Status Register



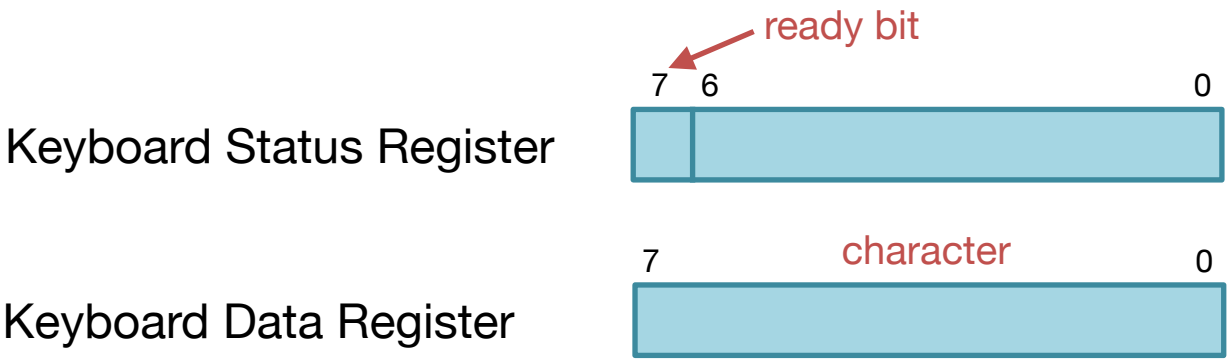
Display Data Register

Basic Keyboard and Display Example

Location	Register	Function
0xFFFFFE00	Keyboard Status Register	Bit 7 is 1 when keyboard has received a new character
0xFFFFFE01	Keyboard Data Register	Contains the last character typed on the keyboard
0xFFFFFE02	Display Status Register	Bit 7 is 1 when device is ready to display another character
0xFFFFFE03	Display Data Register	The character in this register is the data to be displayed

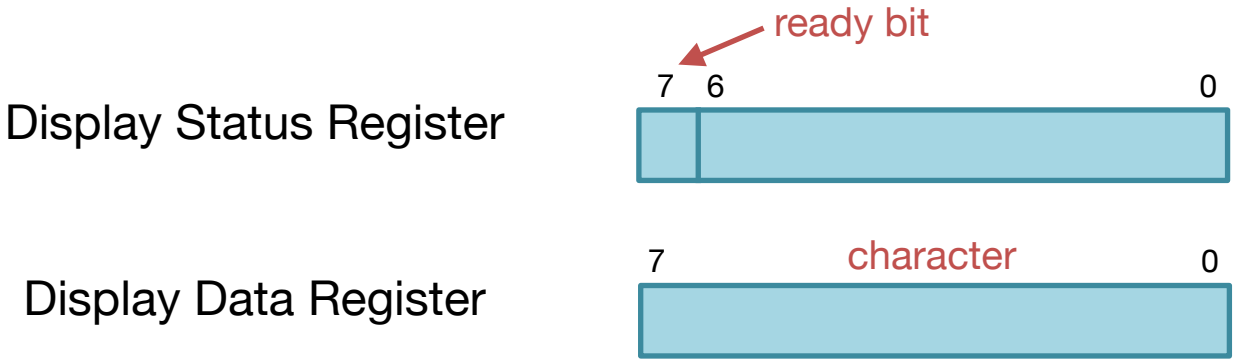
Basic Keyboard and Display Example

- When a key is pressed
  - It's ASCII code is placed in the keyboard data register
  - The keyboard sets the ready bit to 1
- Once CPU accepts the keyboard input
  - The CPU sets the ready bit to zero



Basic Keyboard and Display Example

- When the display is available to display a new character
  - It sets its ready bit to 1
- When data is written to the display register
  - The ready bit is set to 0
  - The character in the display register is written to the screen



Cost of Polling

- Let's say it takes 400 cycles to complete a polling operation
- We have to poll a mouse 30 times per second. Our clock runs at 1 GHz (1 billion cycle/second).
- How much of our processor time is spent polling?

Cost of Polling

- Let's say it takes 400 cycles to complete a polling operation
- We have to poll a mouse 30 times per second. Our clock runs at 1 GHz (1 billion cycle/second).
- How much of our processor time is spent polling?

Number of cycles per second spent polling

30 polls/sec * 400 cycles/poll = 12K cycles/sec

12K cycles/sec / 10^9 cycles/sec = 0.0012%

1 GHz

Cost of Polling

- I want to read from a disk that can transfer data at a rate of 16 MB/sec
- I can only accept 16B per poll
- The clock runs at 1 GHz (1 billion cycle/second).
- What percent of my processor time is spent polling?

Cost of Polling

- I want to read from a disk that can transfer data at a rate of 16 MB/sec
- I can only accept 16B per poll
- The clock runs at 1 GHz (1 billion cycle/second).
- What percent of my processor time is spent polling?

Number of polls needed per second to ensure that no data is missed → (16 MB/sec) / (16 B/poll) = 1M polls/sec

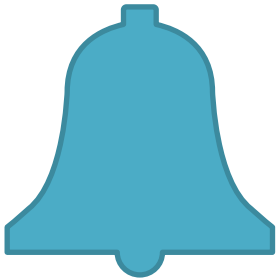
Number of cycles per second spent polling → 1M polls/sec * 400 cycles/poll = 400M cycles/sec

Unacceptable

400M cycles/sec / 10^9 cycles/sec = 40%

Doorbell Analogy

- Polling
  - Checking the door every two minutes to see if someone is there
- Interrupts
  - Only check the door when the door bell rings



Alternative to Polling: Interrupts

- Polling wastes processor resources when no data is available from the I/O device
  - Akin to frequently checking the door for guests
- Interrupts are like the doorbell
  - Occur only when I/O needs attention
- When an interrupt occurs
  - Stop the current program
  - Transfer control to the trap handler in the OS

Interrupts vs Polling

	Interrupts	Polling
If no I/O activity	No wasted cycles	Lots of wasted cycles
If lots of I/O activity	Expensive – saving/restoring state	Less expensive - only poll when we are already context switching
Better suited for events that are	• asynchronous (unsure when event will occur) • urgent • infrequent	• synchronous (occurs at fixed intervals) • not urgent • frequent (majority of polls are a hit)
Examples	Disk	Keyboard, mouse

Programmed I/O

- CPU is in charge of moving all data to/from devices
- Moving data is a pretty easy task, so it's wasteful to make the CPU do it
- Also, the speed of I/O devices is much slower than the CPU…

I/O Device Examples

name	*type*	*approx. max data-rate*	*where*
magnetic disk drive	in/out	200 MB/s	servers, desktops
solid-state drive (SSD)	in/out	550 MB/s	servers, desktops, laptops, handheld
display	out	100 MB/s	desktops, laptops, handheld
keyboard	in	10 B/s	desktops, laptops
speakers / headphones	out	200 KB/s	desktops, laptops, handheld
microphone	in	200 KB/s	desktops, laptops, handheld, embedded
mouse	in	100 B/s	laptops
video camera	in	100 MB/s	desktops, laptops, handheld, embbed
printer	out	2 KB/s	desktops, laptops
touch screen	in/out	100 B/s, 100MB/s	laptops, handheld

Direct Memory Access (DMA)

- Introduce a new piece of hardware: the DMA engine
- Controls the movement of data between the memory and the I/O devices
- Allows the CPU to do other work

Direct Memory Access (DMA)

- To initiate the transfer, the CPU will set up the following in the DMA controller
  - Memory address to place data
  - # of bytes
  - I/O device #
  - direction of transfer
  - …

Direct Memory Access (DMA)

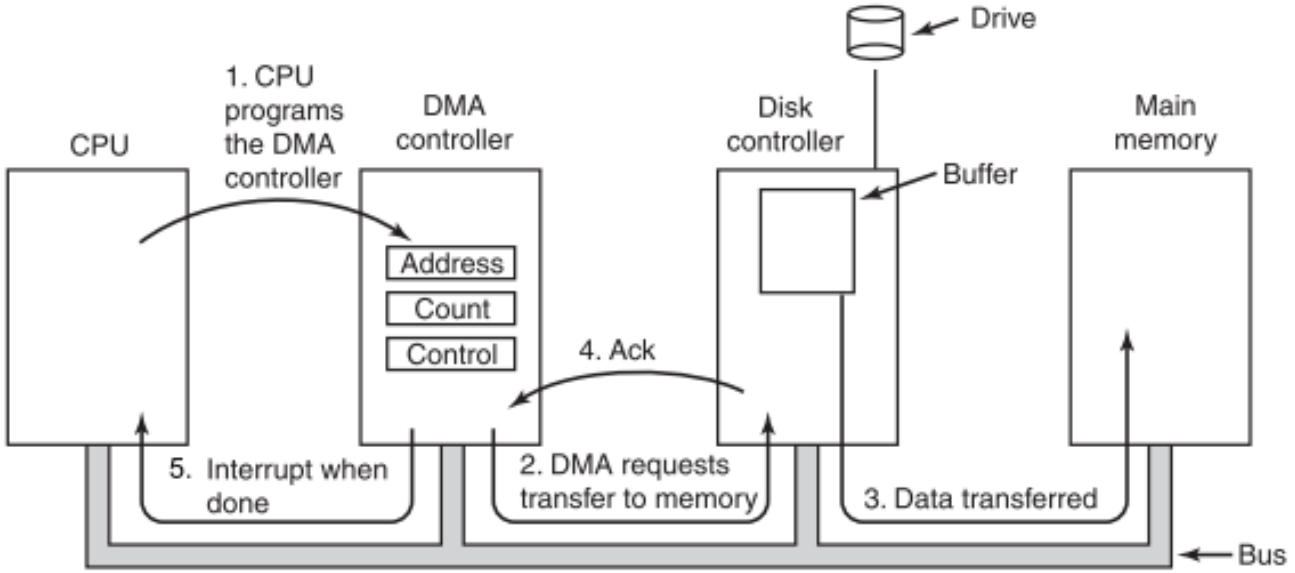


Figure 5-4. Operation of a DMA transfer.

DMA: Incoming Data

1. Receive interrupt from device
2. CPU takes interrupt, initiates transfer
   - Instructs DMA engine to place data @ certain address
3. DMA engine handle the transfer
   - CPU is free to execute other things
4. Upon completion, DMA engine interrupt the CPU again

DMA: Outgoing Data

1. CPU decides to initiate transfer, confirms that external device is ready
2. CPU initiates transfer
   - Instructs DMA engine that data is available @ certain address
3. DMA engine handle the transfer
   - CPU is free to execute other things
4. DMA engine interrupt the CPU again to signal completion

Coming up

- Networking
- Dependability
- Warehouse scale computing