CMSC421: Principles of Operating Systems

Nilanjan Banerjee

Assistant Professor, University of Maryland
Baltimore County
nilanb@umbc.edu
http://www.csee.umbc.edu/~nilanb/teaching/421/

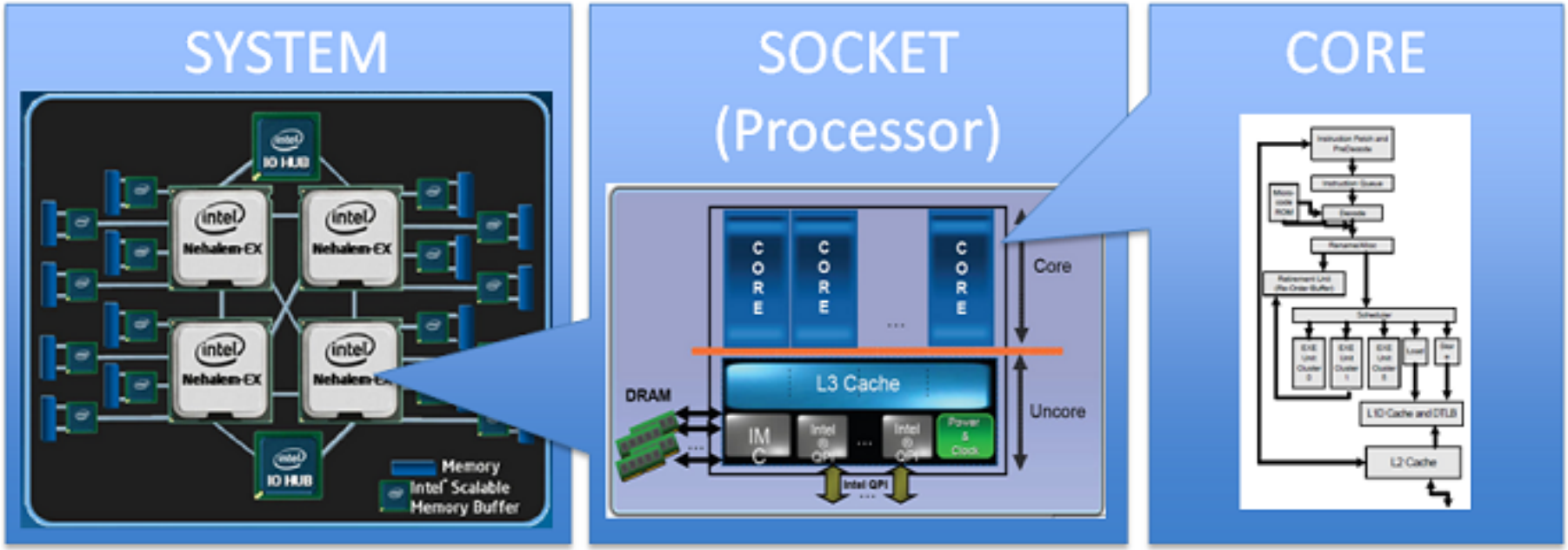
Acknowledgments: Some of the slides are adapted from Prof. Mark Corner and Prof. Emery Berger's OS course at Umass Amherst

Announcements

- Midterm (29th of October in class)
- Project 2 is out (design document is due 26th October)
- Readings from Silberchatz [8th chapter]

Memory Hierarchy

- Registers
- Caches – L1, L2, L3 caches and a special cache called the TLB (Translation lookaside buffer)
- Main Memory
- Network Memory
- Hard Drive
- Tape (nobody uses tape any more)



Registers

- Fastest piece of memory
- On the chip
- 1 clock cycle to read of write from a register
  - 3 GHz machine --- 0.33 Nanoseconds
- Pros: extremely fast
- Cons:
  - Limited space (8 – 16 registers, each 16 bit or 32 bit)
  - Cannot store a lot of information
  - Should use it whenever possible
  - No direct access from a language like C

L1 cache

- L1 cache geometrically close to the CPU
  - 8 KB – 32 KB
  - 3-4 clock cycles (1 ns to read/write from L1 cache)
  - You cannot address the cache directly (upto the hardware, OS to put data into a cache)
  - Cache is associated with cache lines (64 Bytes)
  - You can populate 64 Bytes of data in the cache at a time
  - Also, you cannot address all locations of a cache (associativity)
  - Four way associative --- 4 locations in the cache that can be addressed independently
  - Fully associative --- address all cache locations

L1 cache

- L1 cache geometrically close to the CPU
  - 8 KB – 32 KB
  - 3-4 clock cycles (1-2 ns to read/write from L1 cache)
  - Cumulative overhead = 5 clock cycles
  - You cannot address the cache directly (upto the hardware, OS to put data into a cache)
  - Cache is associated with cache lines (64 Bytes)
  - You can populate 64 Bytes of data in the cache at a time
  - Also, you cannot address all locations of a cache (associativity)
  - Four way associative --- 4 locations in the cache that can be addressed independently
  - Fully associative --- address all cache locations

L2 cache

- L2 cache geometrically close to the CPU
  - 256 KB
  - 6 clock cycles (2-3 ns to read/write)
  - Cumulative overhead = 5 clock cycles
  - 4-way associative caches

L3 Cache and Main memory

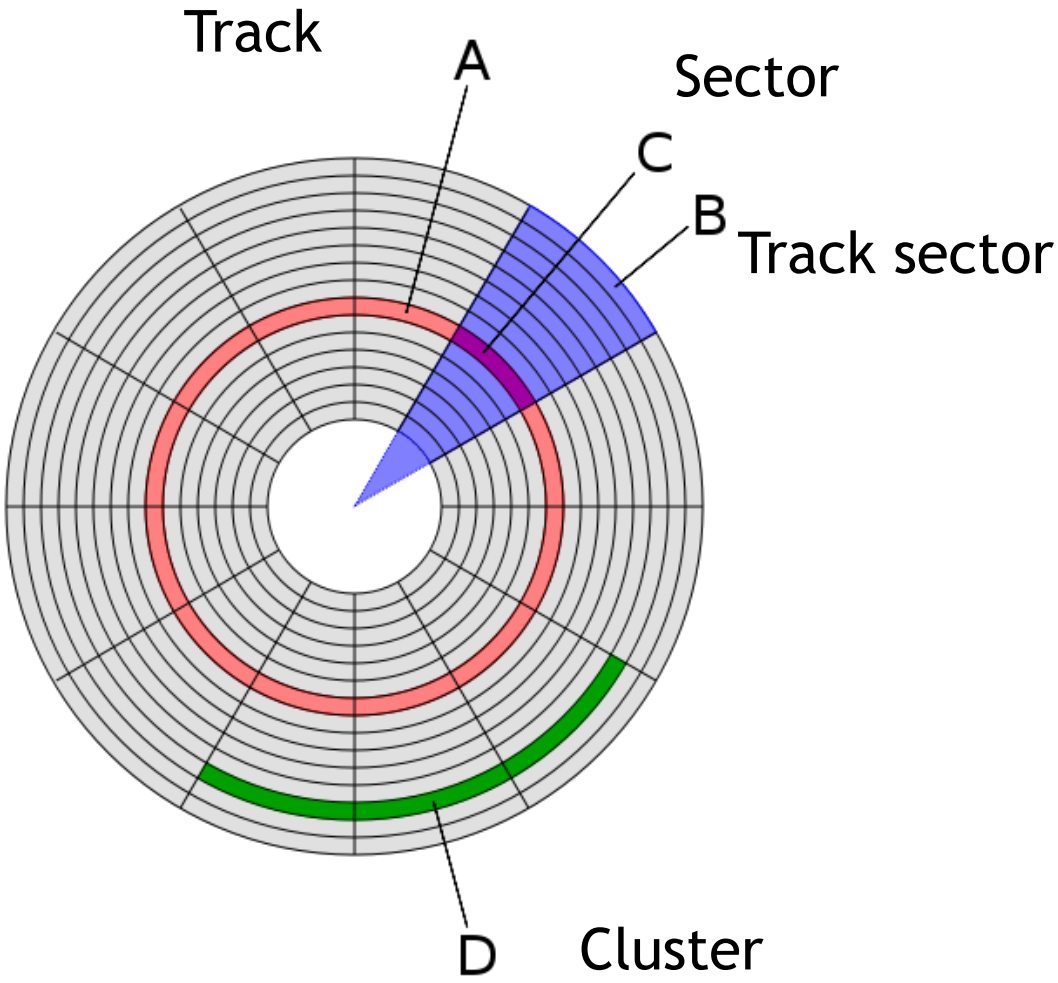
- L3 cache is shared between cores on the same chip
  - 12 Mbytes
  - 10 cycles to access (3-5 ns)
  - Overall cumulative ~ 20 cycles
- Main Memory
  - SDRAM (Synchronous dynamic random access memory)
  - Characterized by FSB (front side bus)
  - FSB– memory bus whose speed would be 1.3 GHz
  - 7-7-7-21 would a specification for Main memory
  - Time to access <row>-<column>-<cleanup> -- total latency
  - 21 FSB clock cycles ~= 60 CPU clock cycles
  - 18 ns?

Hard drive

- Permanent storage
- Use the harddrive instead of memory

Hard drive

- Head moves up and down and the disc spins



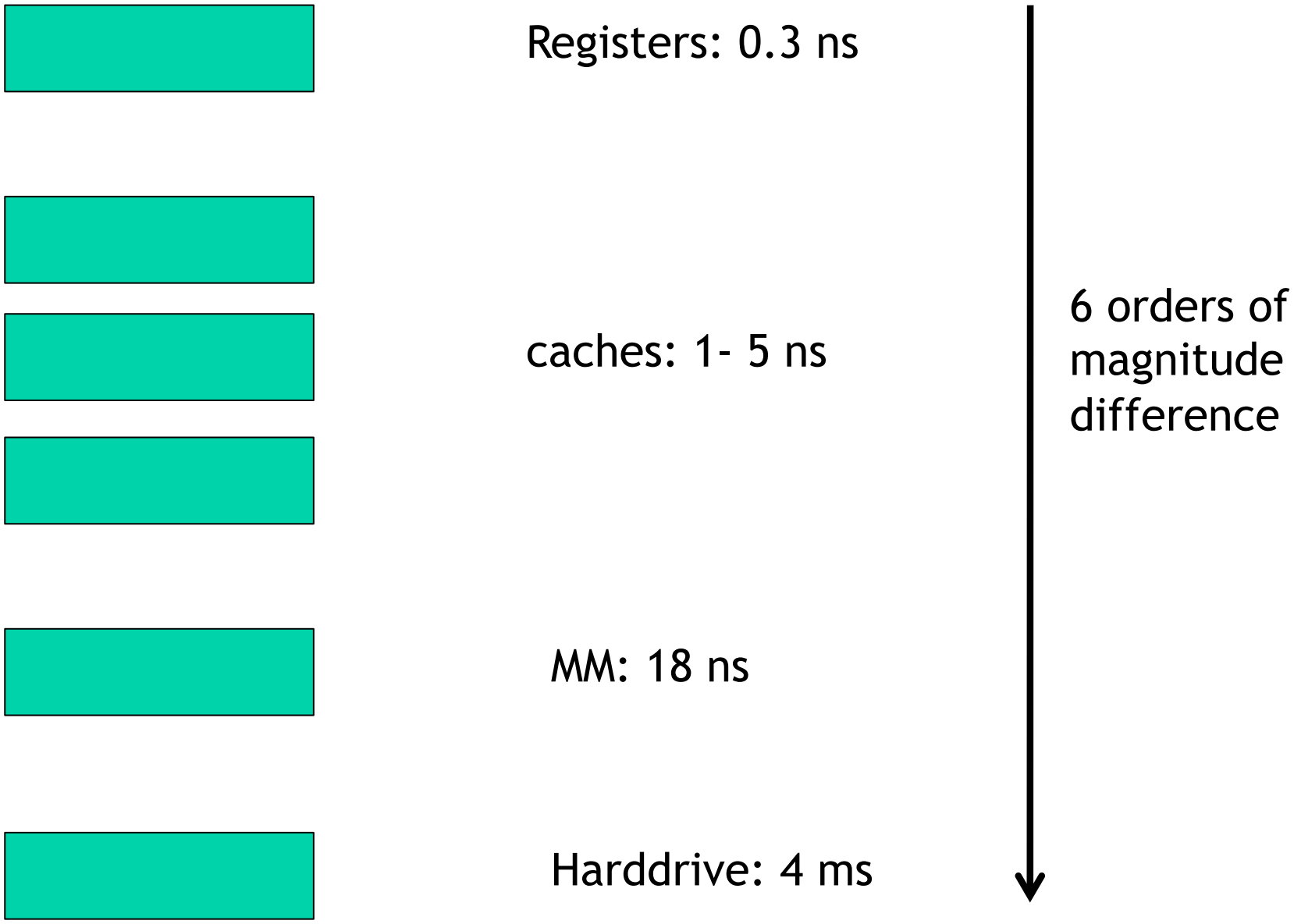
Acknowledgments: Wikipedia

Hard drive

- How fast can be read/write to the hard drive
- 7200 rpms --- 7200 revolutions/minute
- = 120 revolutions/second
- = 8 ms for one revolution
- On expectation = 4 ms to read/write from a sector
- Compared to L1 cache == 1ns (4 million times slower)

Acknowledgments: Wikipedia

Lets put this in perspective



There is an intermediary to Harddrive

Network memory
How fast is network memory?

How does all of this work?

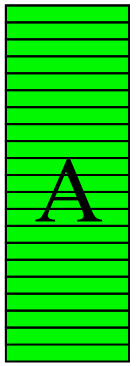
- A process asks for memory to read/write/copy
  - Does not really care where the data came from
  - Registers, L1, L2, L3, Main memory, Network memory or the disk
- Key question: How do we make this transparent from the process

Virtual vs. Physical Memory

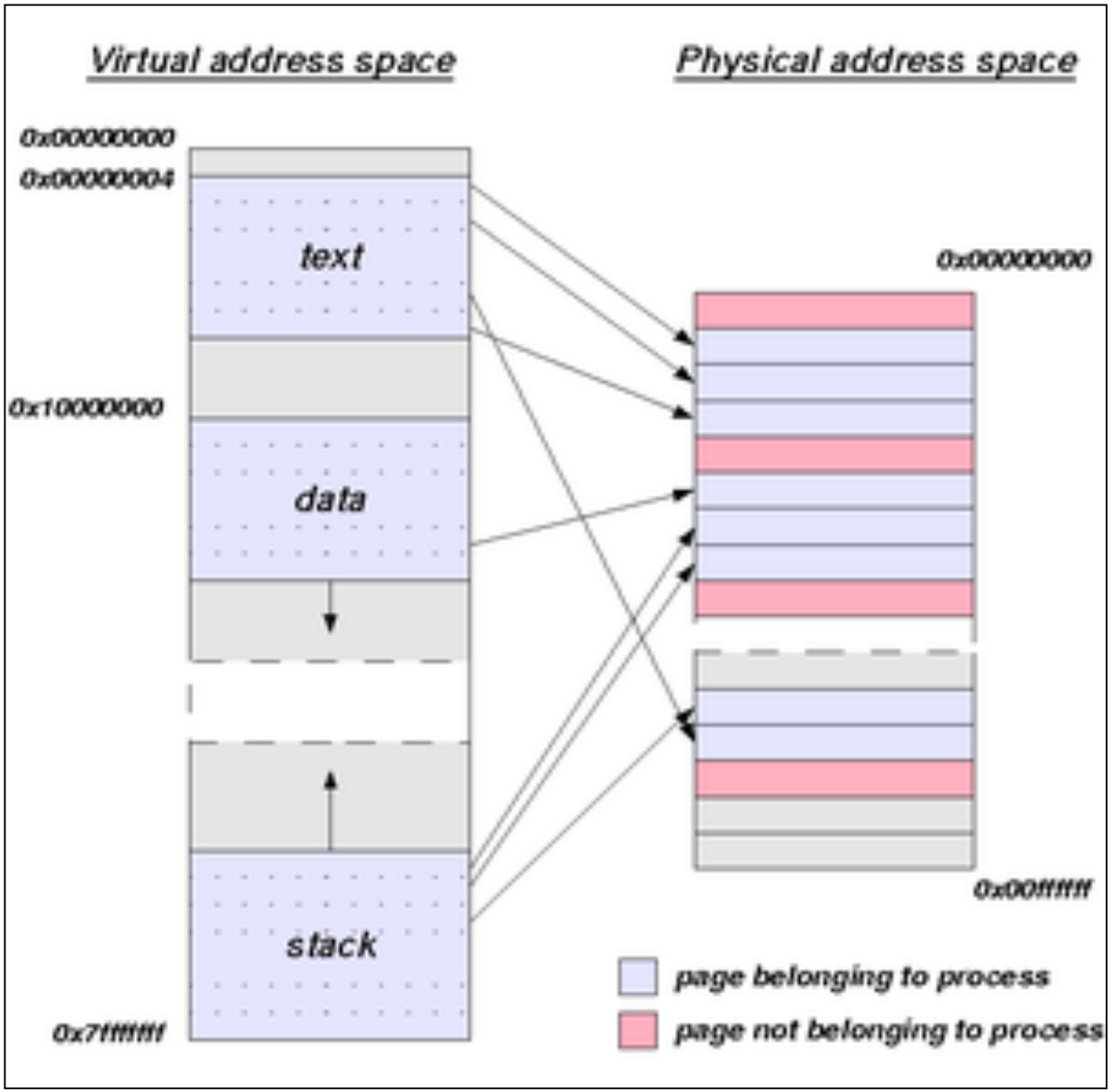
- Processes don't access physical memory
  - Well, not directly
- Apps use **virtual memory**
  - Addresses start at 0
  - One level of **indirection**
  - Address you see is not "real" address

Memory Pages

- Programs use memory as individual bytes
- OS manages groups of bytes: **pages**
  - typically 4kB, 8kB
  - Applies this to virtual and physical memory
    - Physical pages usually called **frames**

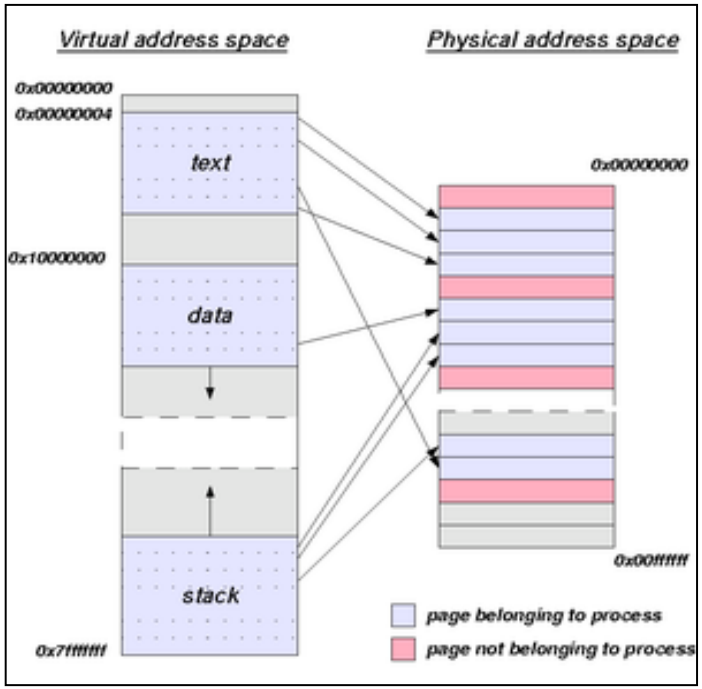


Mapping Virtual to Physical



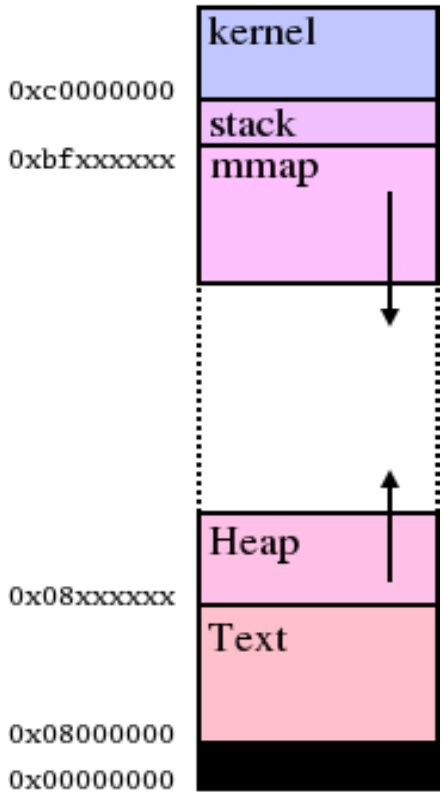
Why Virtual Memory?

- Why?
  - Simpler
    - Everyone gets illusion of whole address space
  - Isolation
    - Every process protected from every other
  - Optimization
    - Reduces space requirements



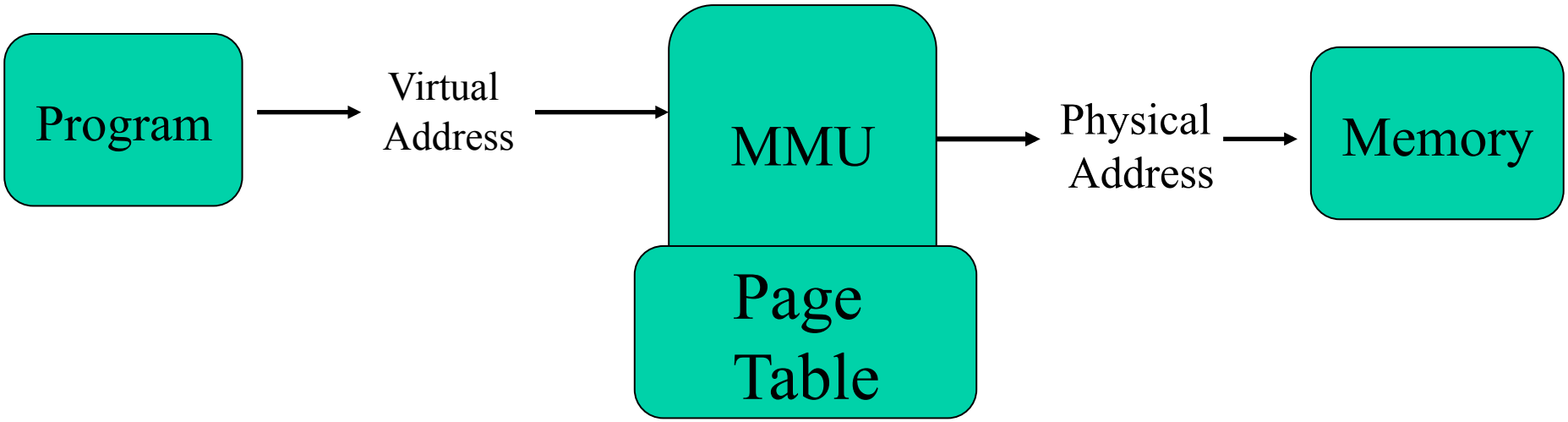
Typical Virtual Memory Layout

- Some things grow
  - Must leave room!
- Mmap and heap spaces
  - Mmap increases mmap
  - Brk increases heap
- Other layouts possible



Memory Management Unit

- Programs issue loads and stores
- What kind of addresses are these?
- MMU Translates virtual to physical addresses
  - Maintains page table (big hash table):
  - Almost always in HW… Why?



Page Tables

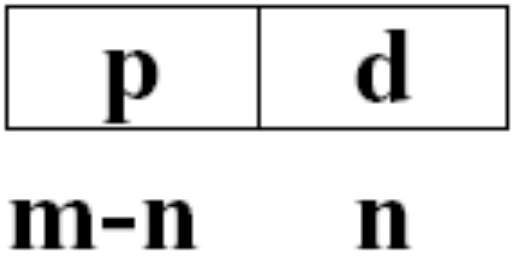
- Table of translations
  - virtual pages -> physical pages
- One page table per process
- One **page table entry** per virtual page
- How?
  - Programs issue virtual address
  - Find virtual page (how?)
  - Lookup physical page, add offset

Page Table Entries

- Do all virtual pages -> physical page?
  - **Valid** and **Invalid** bits
- PTEs have lots of other information
  - For instance some pages can only be read

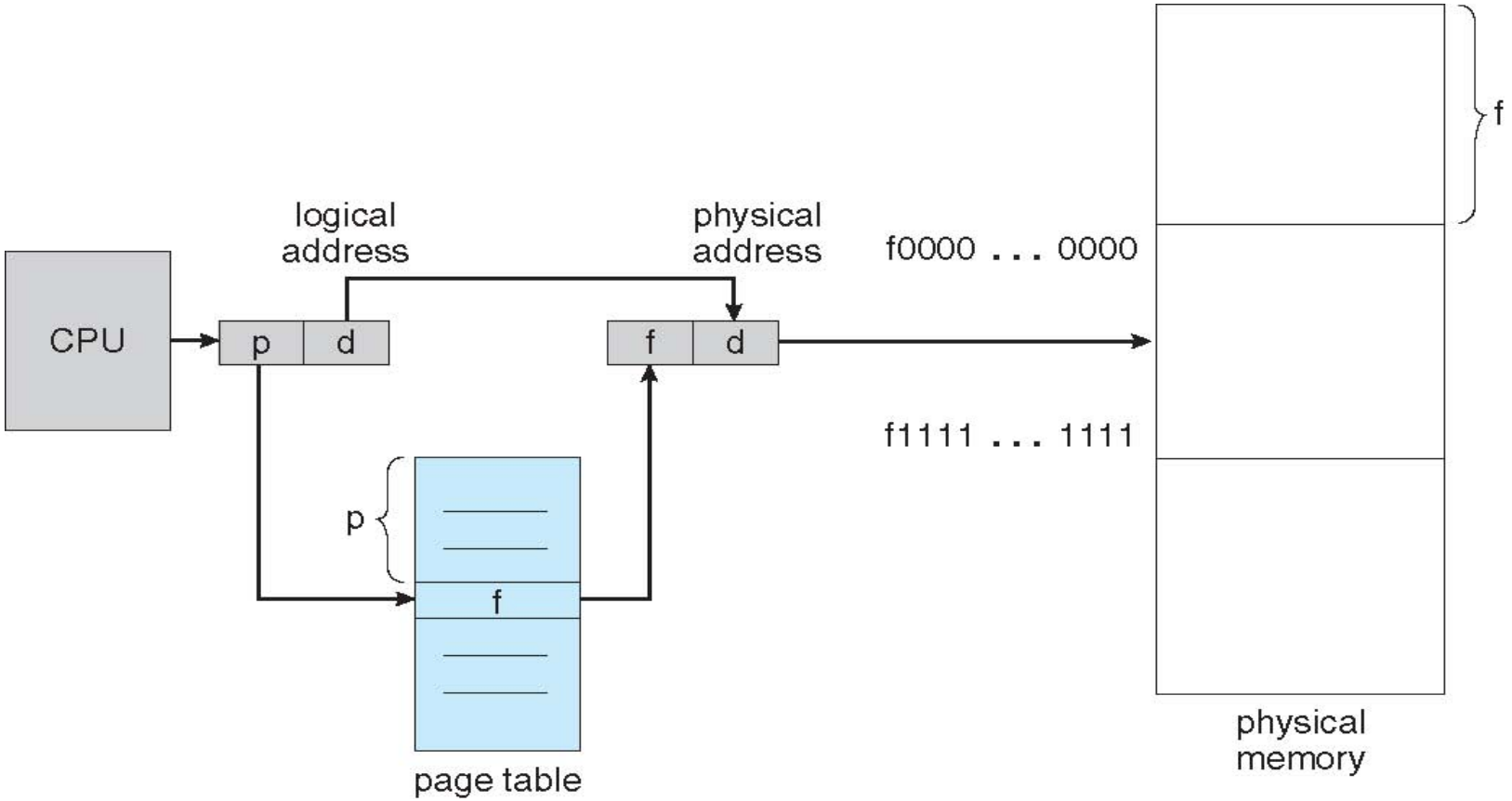
Address Translation

- Powers of 2:
  - Virtual address space: size 2^m
  - Page size 2^n
- Page#: High m-n bits of virtual address
- Lower n bits select offset in page



p: page number
d: page offset

Paging Hardware



Quick Activity

- How much mem does a page table need?
  - 4kB pages, 32 bit address space
  - page table entry (PTE) uses 4 bytes
- 2^32/2^12*4=2^22 bytes=4MB
  - Is this a problem?
  - Isn't this per process?
  - What about a 64 bit address space?
- Any ideas how to fix this?

In-class discussion