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 design document is due 26th October
- We will do a review at the end of the class

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

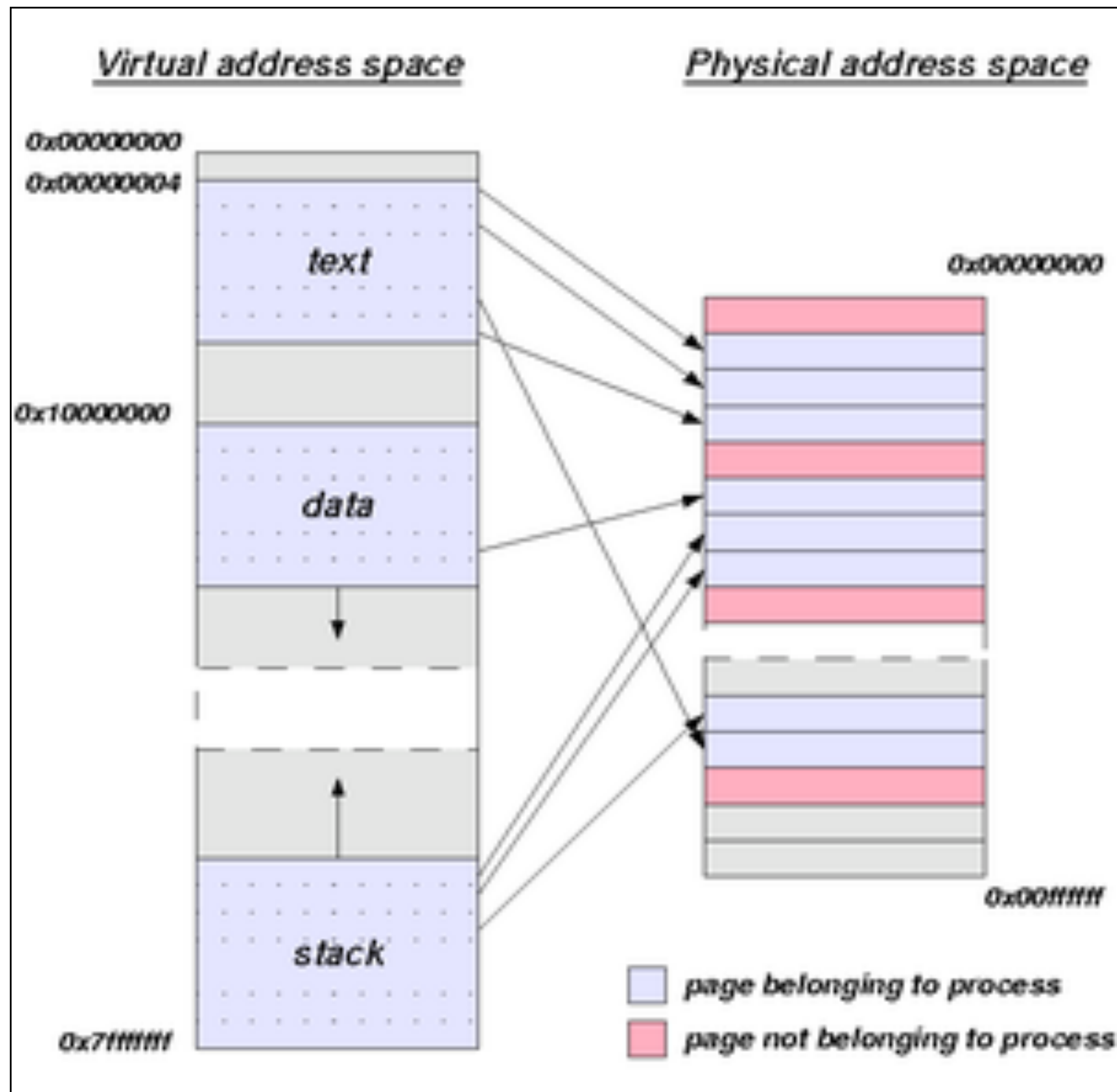
Virtual vs. Physical Memory

- 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

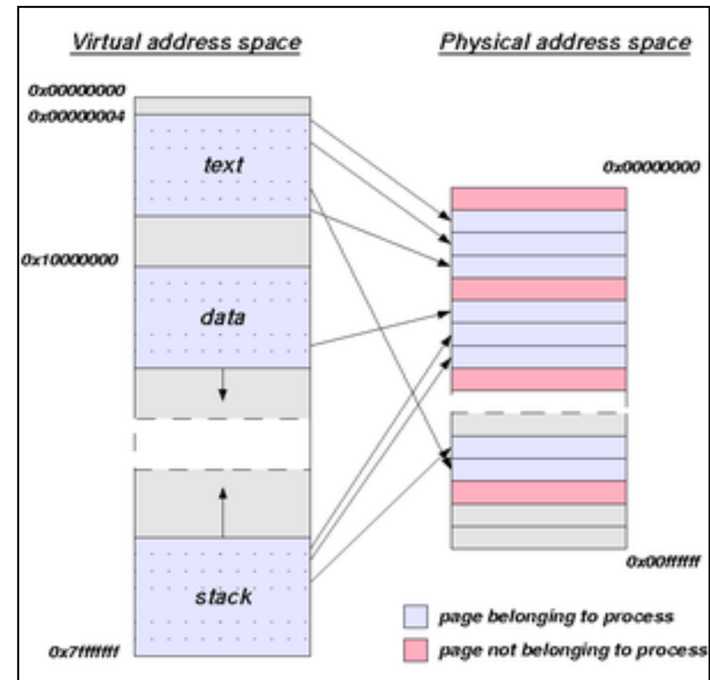
- Programs use memory as individual bytes
- OS manages groups of bytes: **pages**
  - typically 4kB, 8kB
  - Applies this to virtual and physical memory
    - Physical pages usually called **frames**

Mapping Virtual to Physical



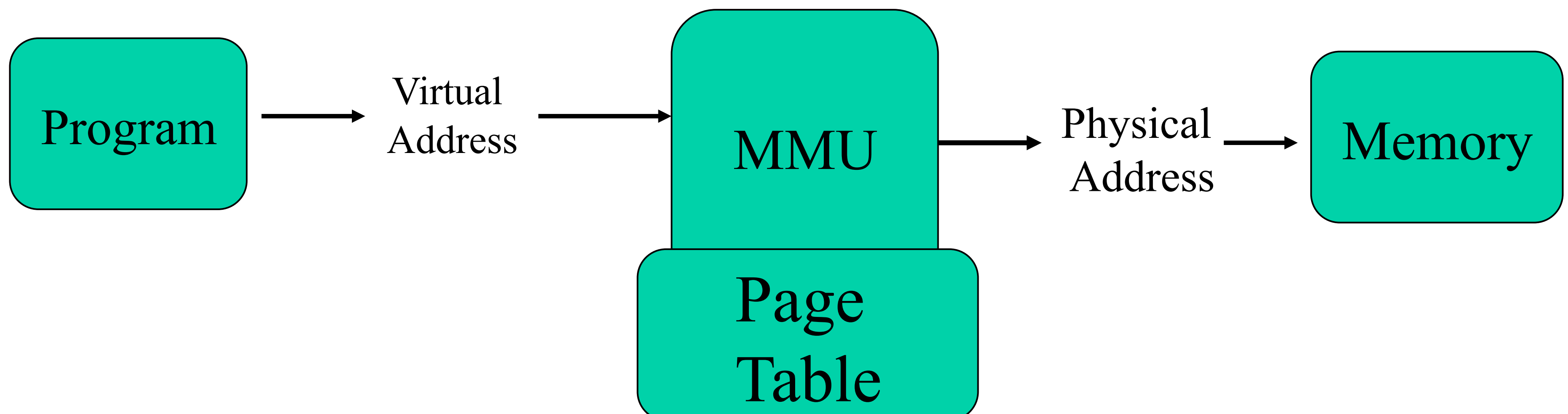
Why Virtual Memory?

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



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

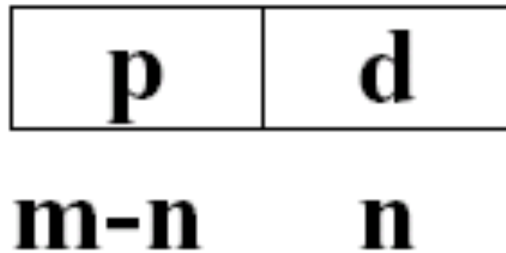
- Table of translations
  - virtual pages -> physical pages
- One page table per process
- One **page table entry** per virtual page
- How?
  - Programs issue virtual address
  - Find virtual page (how?)
  - Lookup physical page, add offset

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

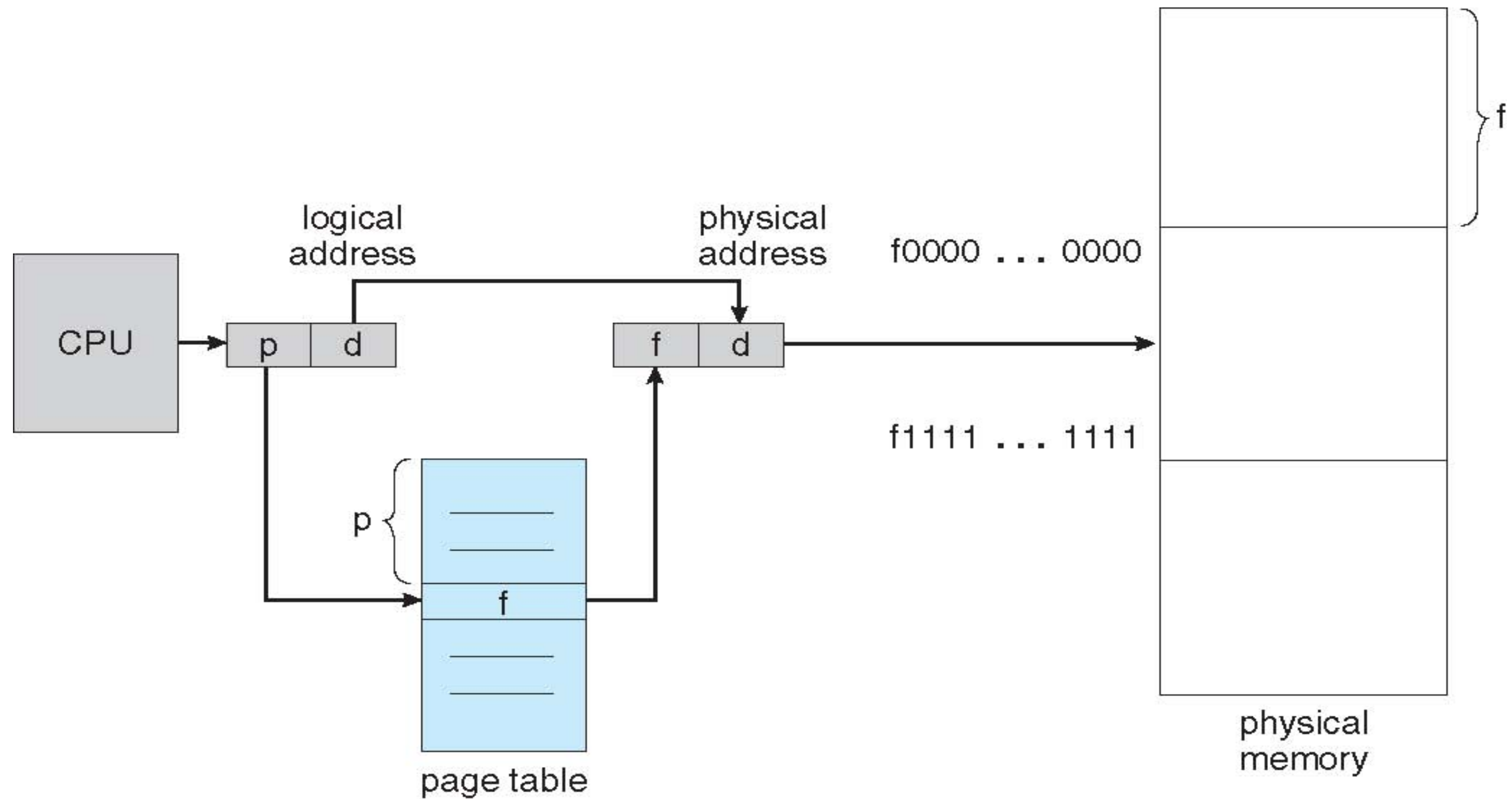
- Powers of 2:
  - Virtual address space: size 2^m
  - Page size 2^n
- Page#: High m-n bits of virtual address
- Lower n bits select offset in page



p: page number
d: page offset

Lets take an example

Paging Hardware

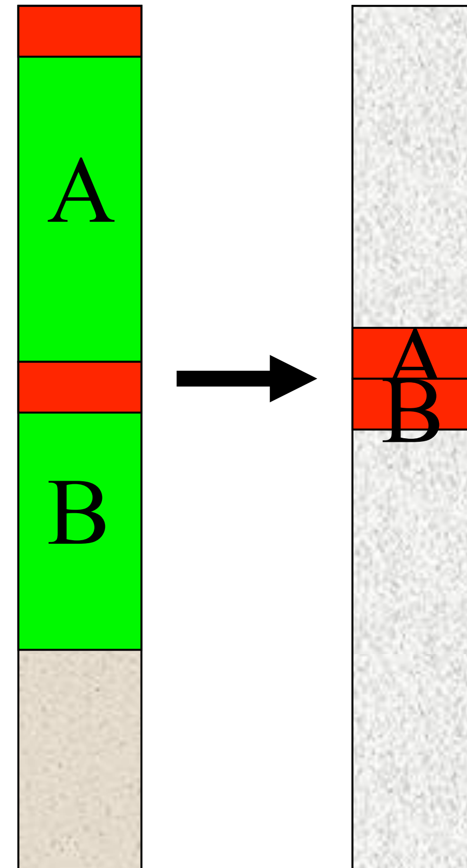


Quick Activity

- 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?

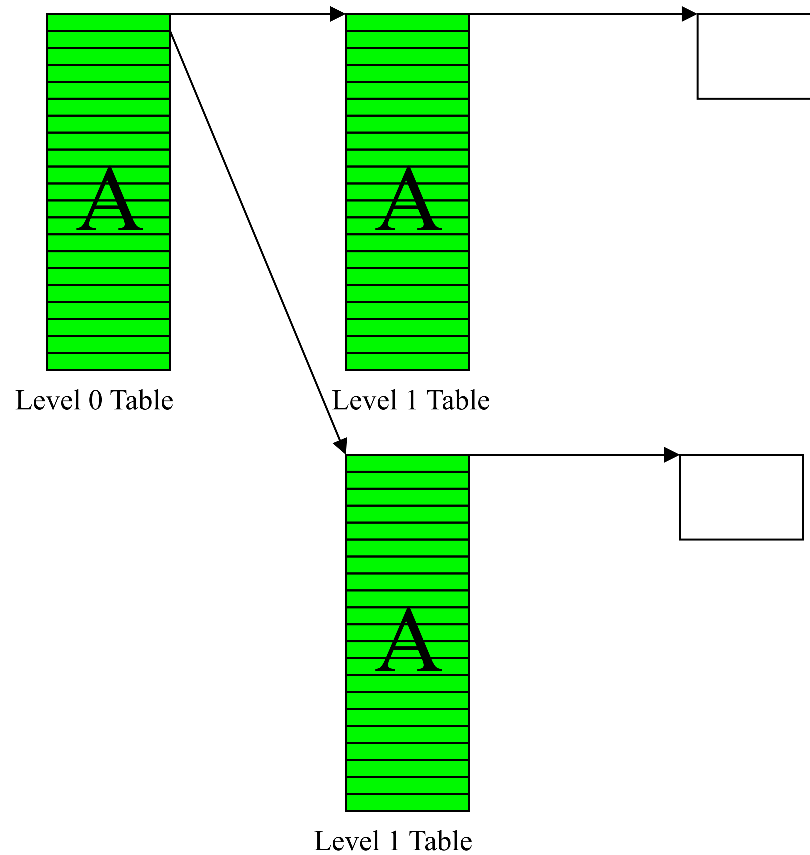
Locality

- Most programs obey 90/10 "rule"
  - 90% of time spent accessing 10% of memory
- Exploit this rule:
  - Only keep "live" parts of process in memory



Multi-Level Page Tables

- Use a multi-level page table



Quick Activity

- How much mem does a page table need?
  - 4kB pages, 32 bit address space
  - Two level page table
  - 20bits = 10 bits each level
  - page table entry (PTE) uses 4 bytes
  - Only first page of program is valid
- 2^10*4+2^10*4=2^13 bytes=8kB

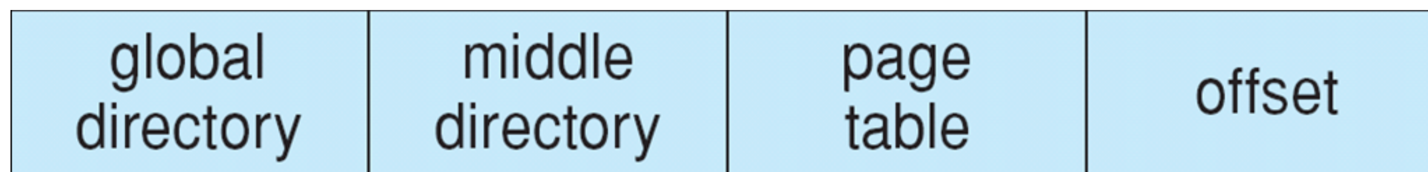
- Isn’t this slow?

Translation Lookaside Buffer (TLB)

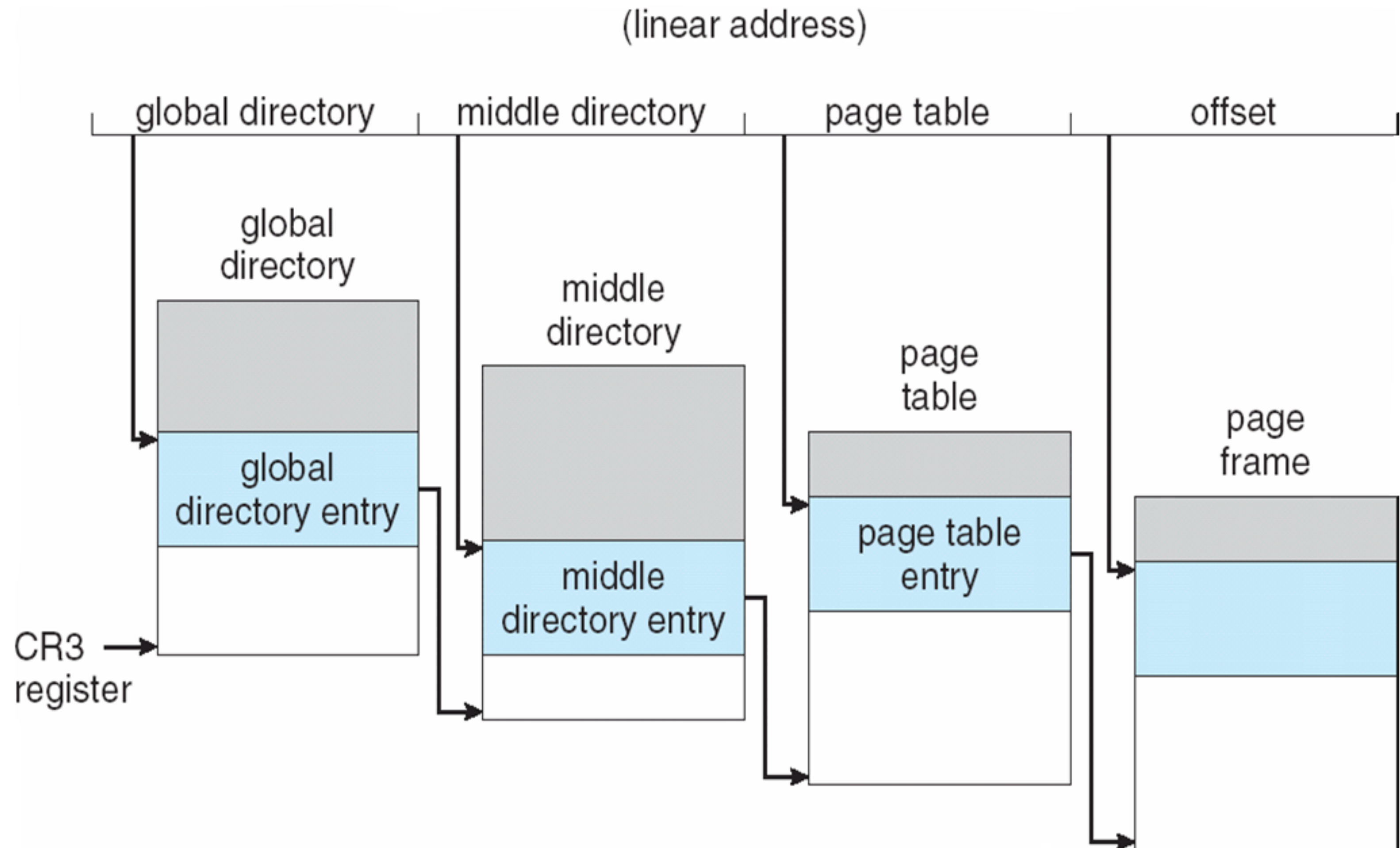
- TLB: fast, fully associative memory
  - Caches page table entries
  - Stores page numbers (key) and frame (value) in which they are stored
- Assumption: locality of reference
  - Locality in memory accesses = locality in address translation
- TLB sizes: 8 to 2048 entries
  - Powers of 2 simplifies translation of virtual to physical addresses

Linear Address in Linux

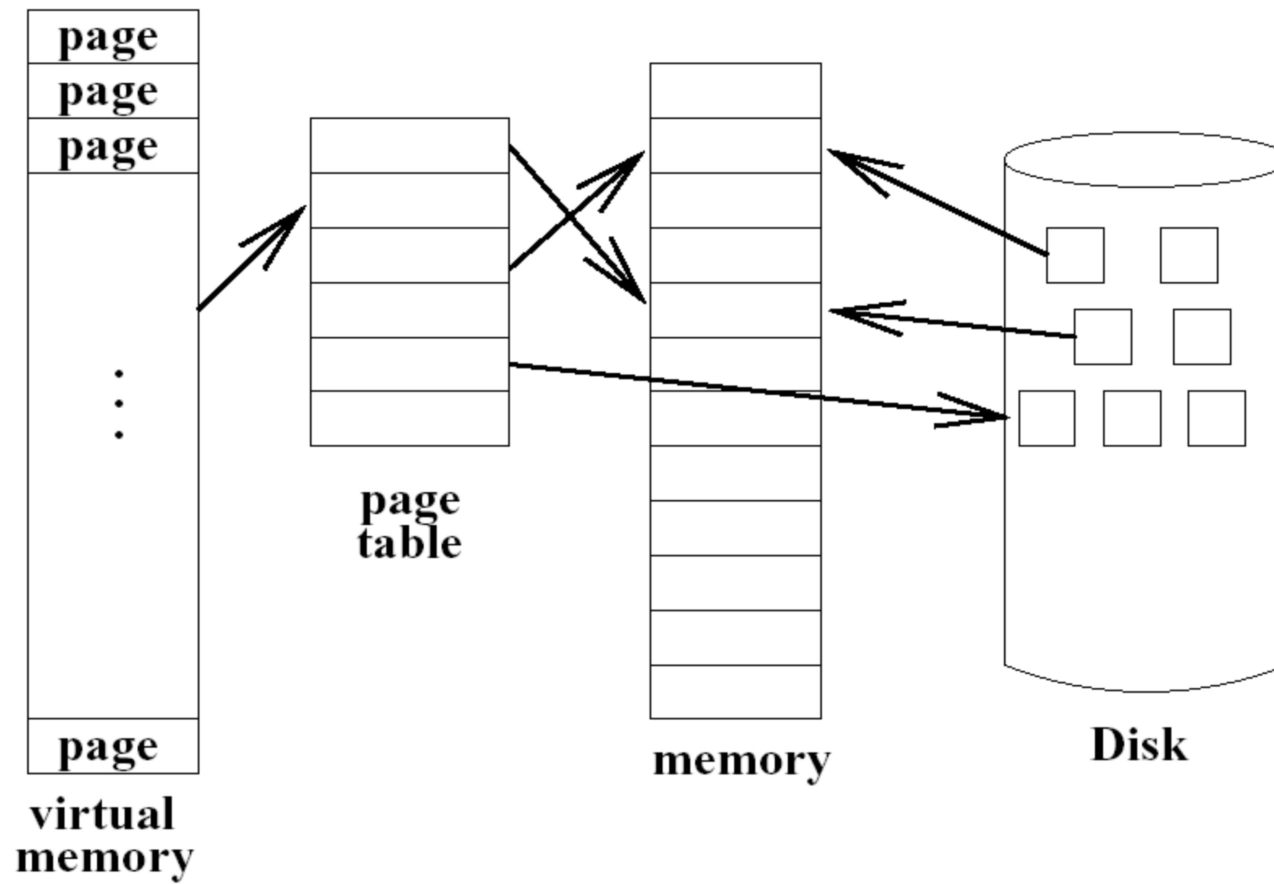
- Uses a three-level paging strategy that works well for 32-bit and 64-bit systems
- Linear address broken into four parts:



Three-level Paging in Linux



Paging



Midterm review