# Memory Management

## *How does the Operating System manage memory?*

Nick Urbanik <nicku(at)nicku.org>



A computing department

# Why memory management?

- Memory gets cheaper
- Programs get bigger:

  "Programs expand to fill the memory available to hold them"
- Memory is nearly always precious

# Issues of memory management

- If run out of memory
  - cannot start a new process, or
  - process cannot continue
- Often, all memory is used.
- How solve this?
  - Buy more memory
  - Use the hard disk to simulate more memory

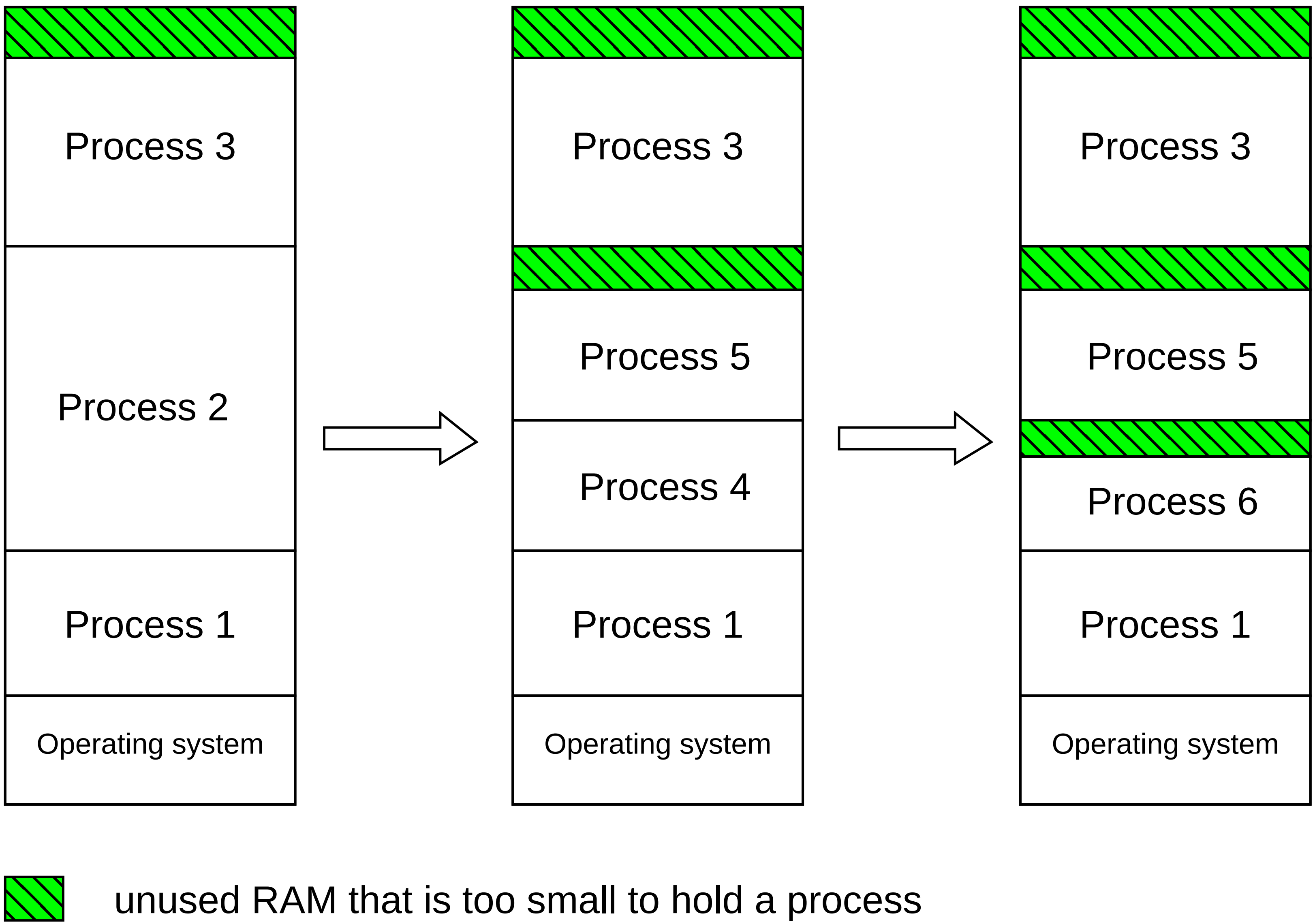
# Virtual memory

- *Virtual memory* uses the hard disk to simulate RAM
- Windows:
  - swap file
- Linux can use:
  - swap file(s) or
  - swap partition(s)

# Swapping

- Where the *entire* memory used by each process is:
  - swapped in (from disk to RAM), or
  - swapped out (from RAM to disk)
- The unit that is written to or from the disk is the process
- Problems:
  - Gaps between used RAM may be too small to use, so they are wasted
    - This problem is called *fragmentation*
    - A consequence of units written to or from the hard disk having *different sizes*
    - Gaps are called “holes”
  - Swapping a big process is too inefficient, it takes too long

# Fragmentation of RAM



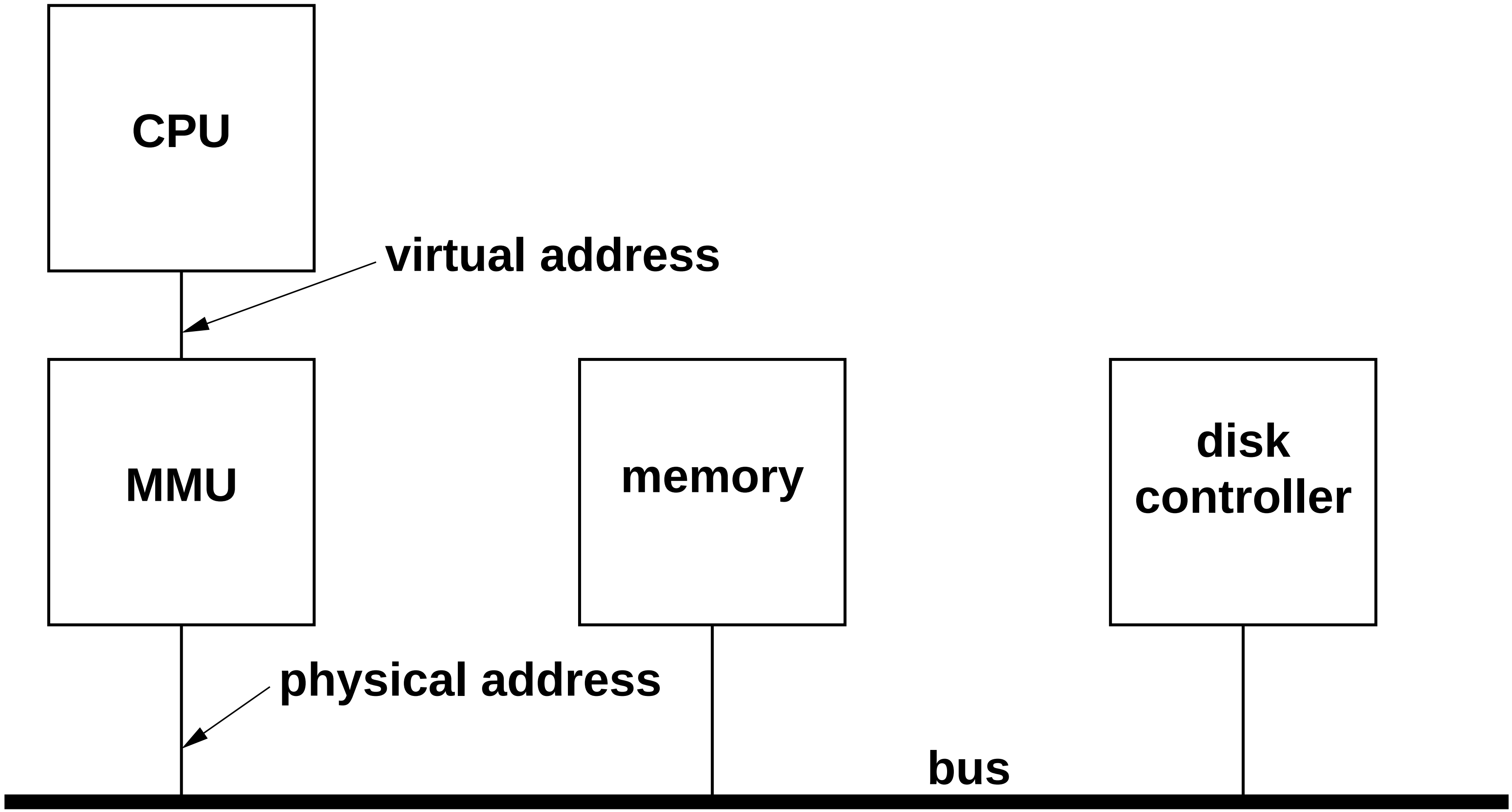
# Paging

- Modern OS uses *paging*
- Each process uses fixed sized chunks of memory called *pages*
- *All* the virtual memory is divided into pages
- The pages do not have to be contiguous (all physically next to each other), so no holes
- Each process has its own *virtual address space*
- Hardware maps from virtual addresses to real addresses

# MMU: Memory Management Unit

- All desktop systems use hardware called a *MMU*
- Quickly maps *virtual addresses* to real addresses (corresponding to voltages on the physical RAM address pins)
- *Hardware* specific to CPU organisation

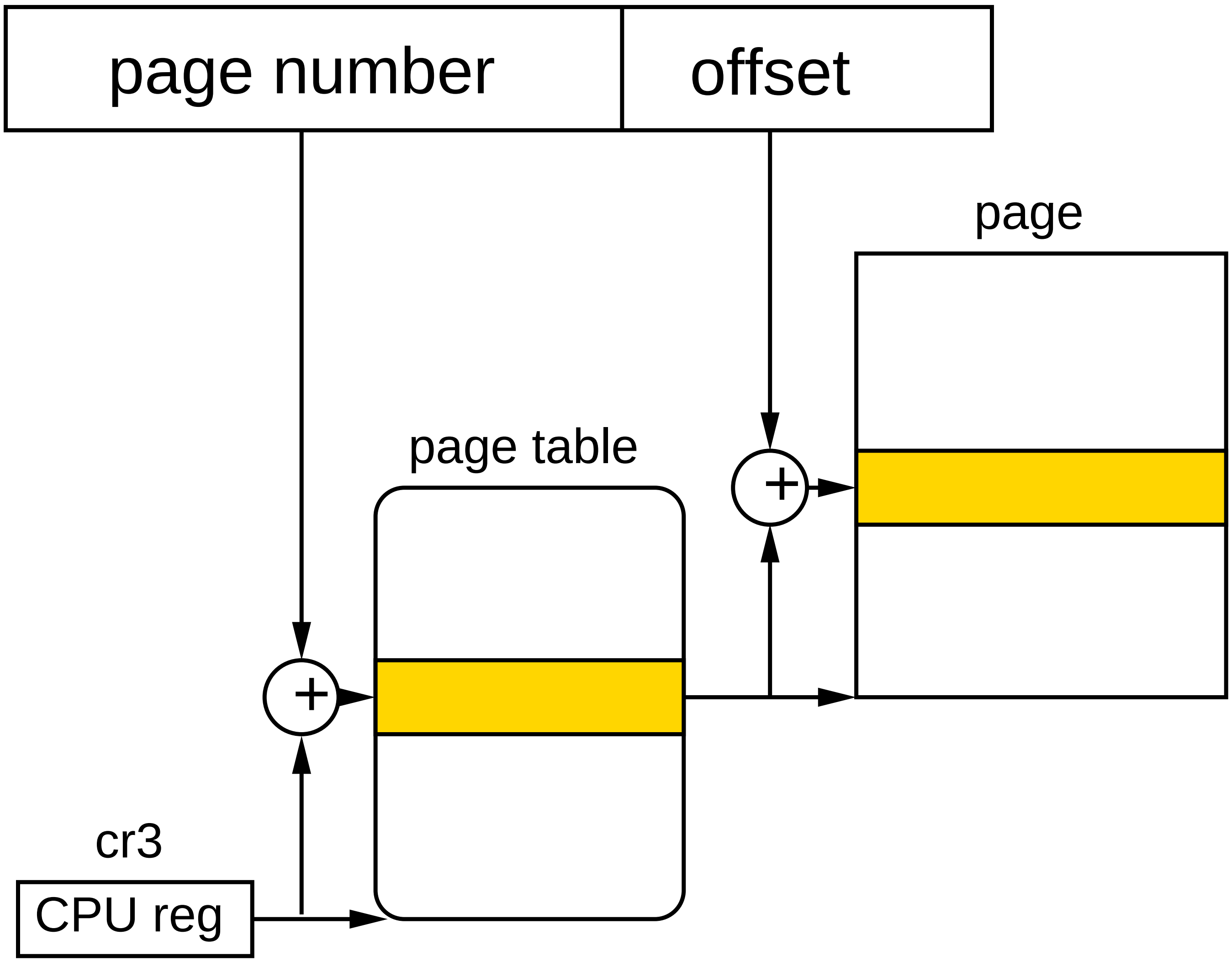
# Memory Management Unit



# Basic idea of paging: page tables

- Virtual address has two parts:
- *Page number*
- *Page offset*
- Each *page table entry* is physical address of start of page

# Page tables



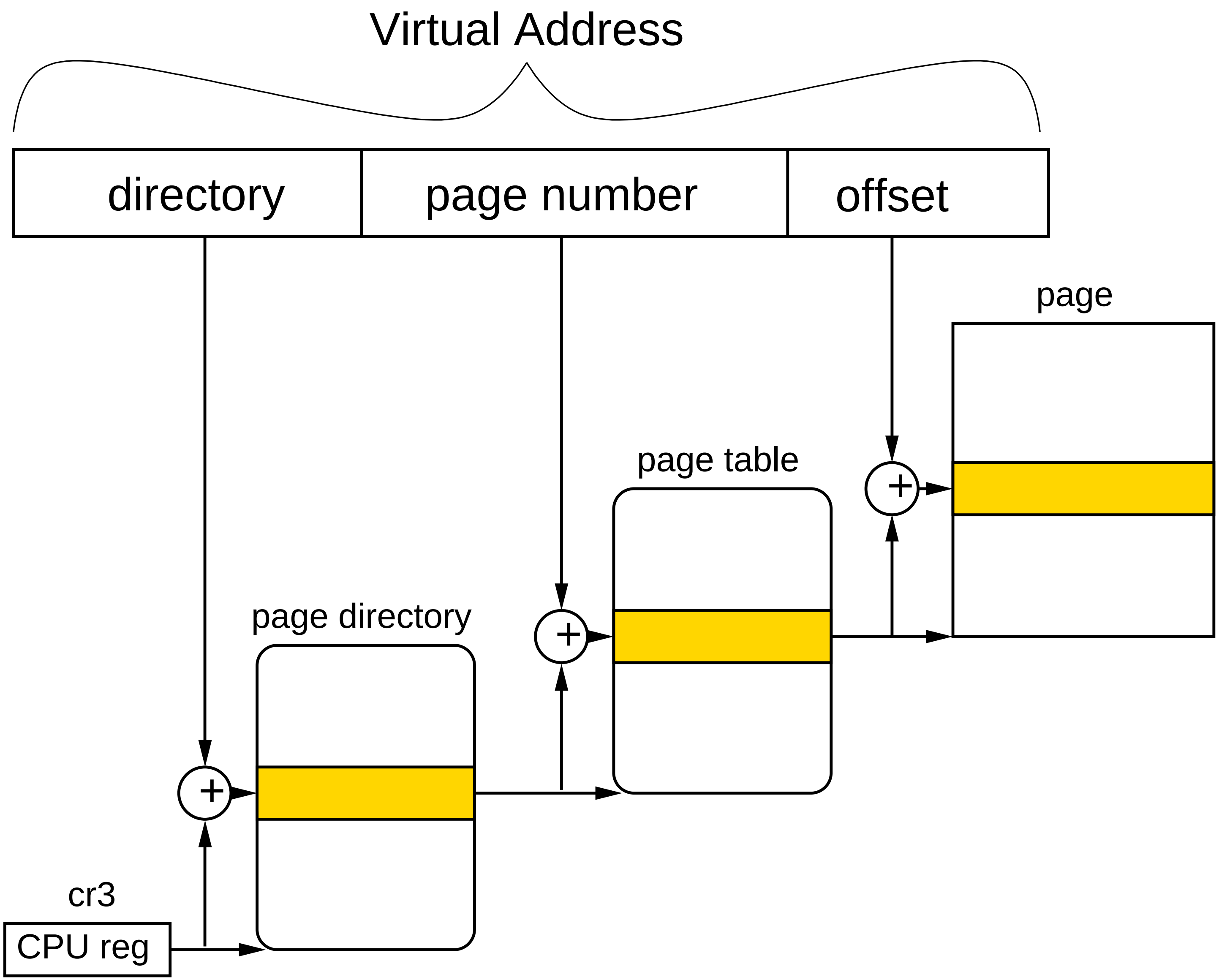
# The MMU does the arithmetic

- The Memory Management Unit performs the arithmetic very quickly
- Contains special registers to store page table entries

# Problems

- If:
  - each page is 4 KB, and
  - virtual memory addresses are 32 bits,
- Then need page tables with $32 - 12 = 20$ bit indexes
- Need page tables to contain $2^{20}$ (more than a million) entries
- Require lots more RAM!

# Multilevel paging



# Page faults

- What if application asks for a page not in RAM?
- CPU gets a *page fault exception*
- Operating system decides how to handle this
- Could also occur if process tries to access RAM outside of its allocation of pages

# Paging on Intel x86 — 1

- | | | |
  |---|---|---|
  | Bits 31...22: | Directory: | 10 bits |
  | Bits 21...12: | page table: | 10 bits |
  | Bits 11...0: | offset: | 12 bits |

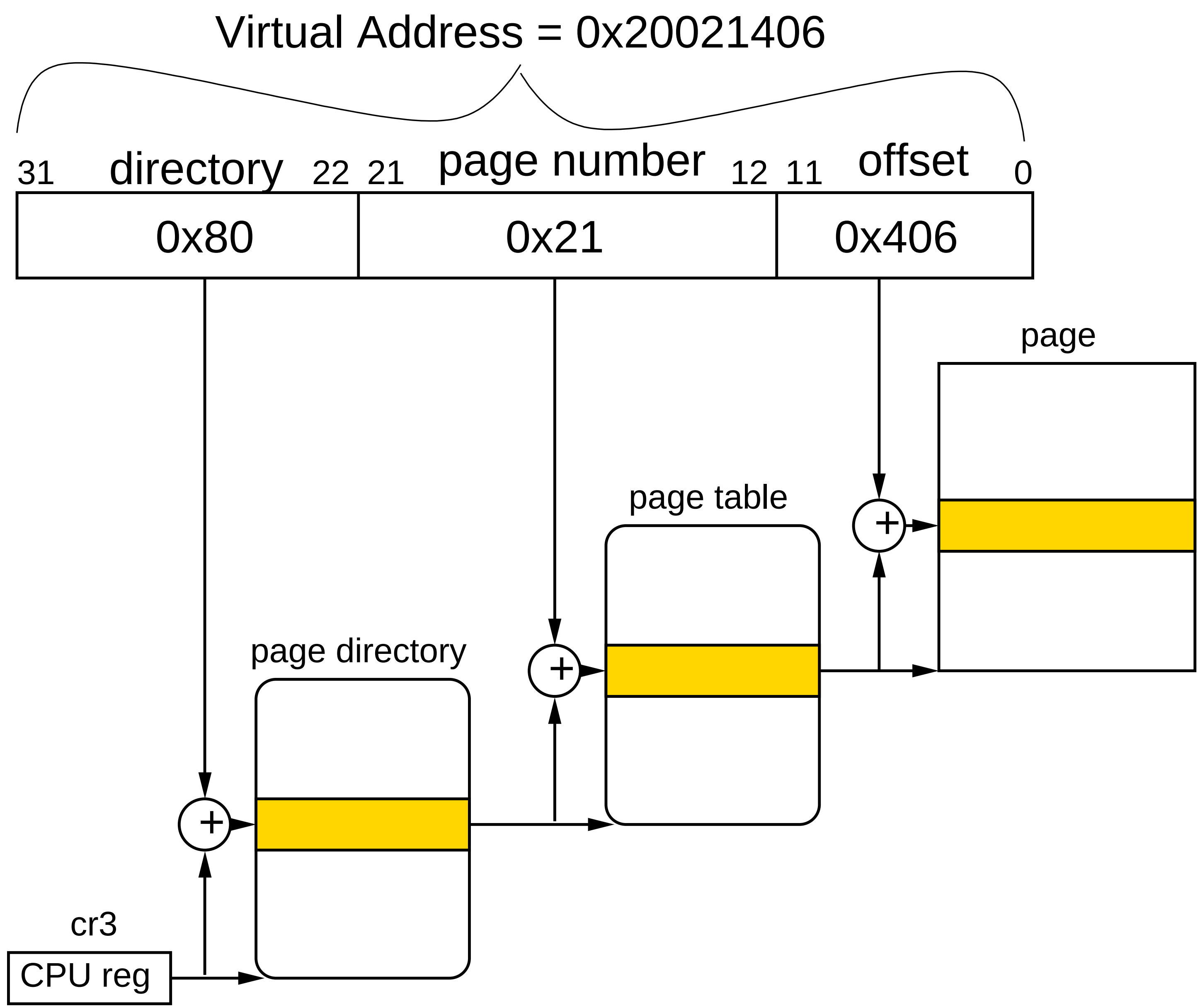
# Paging on Intel x86 — 2

- Page directory, page table entries contain:
  - Present flag: 1 if in RAM, 0 if paged out
  - 20 MS bits of page frame physical address
  - Dirty flag (page table only): set when the page is written to
  - Access rights info

# Example of paging: Intel x86

- Dir: 10 bits, table: 10 bits, offset 12
- Say kernel assigns 64 pages to a process: 0x20000000 to 0x2003ffff
- We don't care about the physical address
- Virtual address = 0x20021406
- We are determining the offset within the page for this address, the page number, and the value of the directory entry
- 10 most significant bits = 0x080
- Middle 10 bits = 0x21 = 33
  - Note: can only be in range 0 to 63 decimal
- Offset 0x406

# Intel Paging Example



# The Arithmetic

- Note: 0xnnnn is the notation in the C programming language for a hexadecimal number $\text{nnnn}_{16}$
- Most significant 12 bits is 0x200
- in binary: 0010 0000 0000
- the ten MS bits are 00 1000 0000
- In Hex, that's 0x80
- 0x20021406 =
  0010 0000 0000 0010 0001 0100 0000 0110

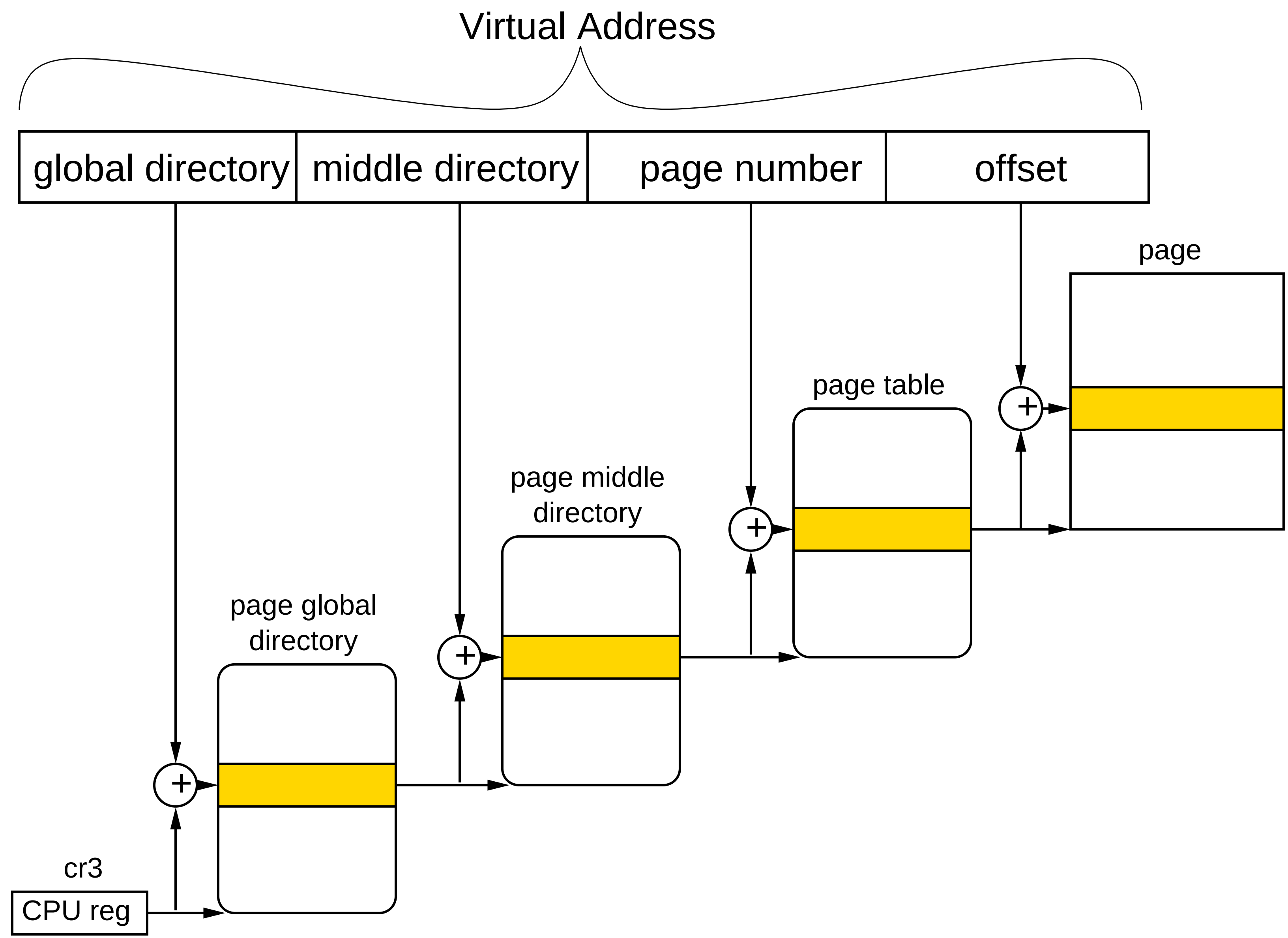
# Paging Example (continued)

- If present flag is 0, page not in RAM
- *Page exception* is generated
- Operating system then decides what to do:
  - Start to read page into RAM
  - (Probably) schedule new process

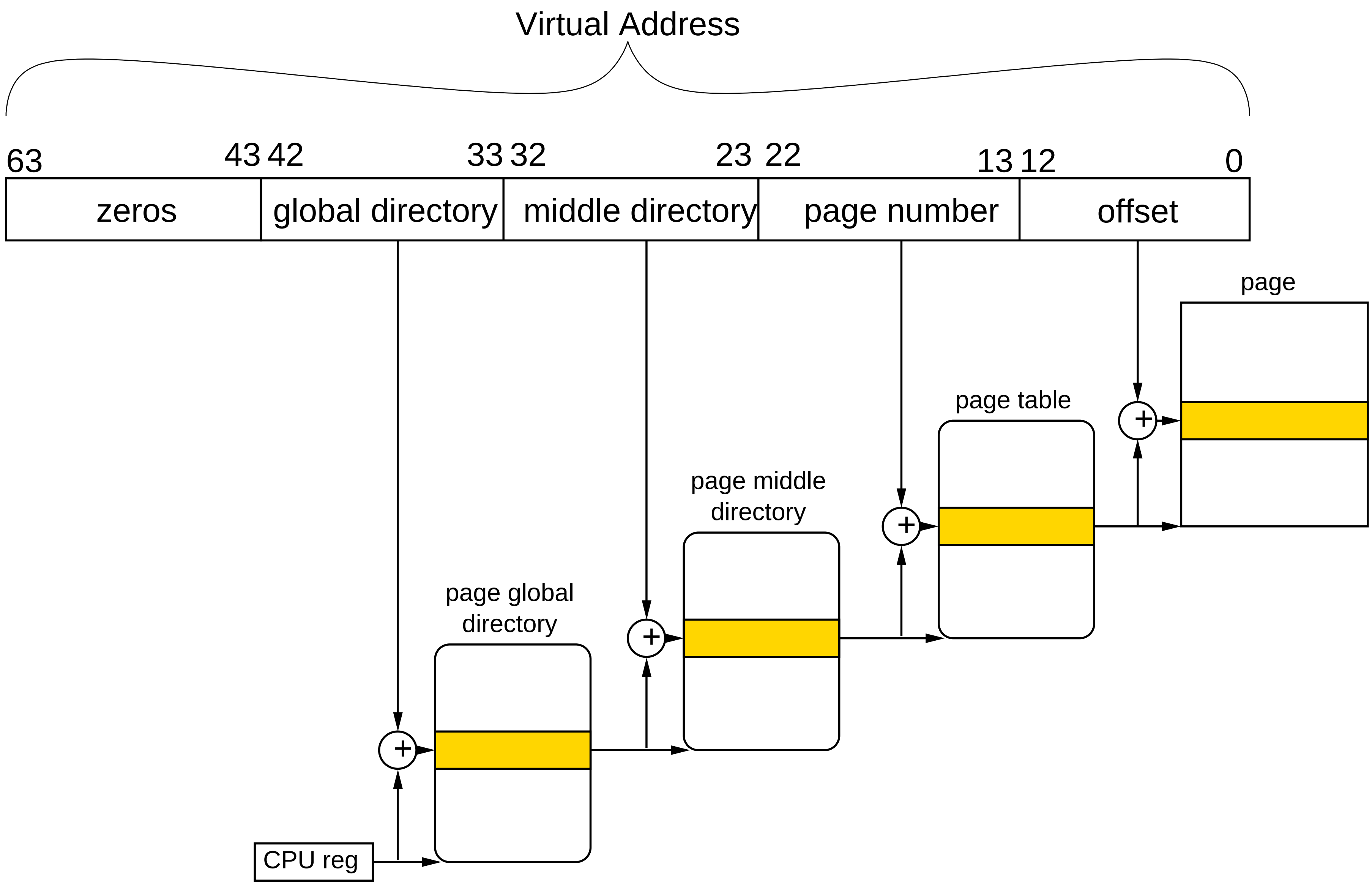
# Linux uses three level paging

- Linux runs on many other architectures than Intel
- Alpha hardware implements 3 level paging
- Linux uses 3 level paging for ease of porting to other architectures

# Three Level Paging



# Alpha Memory Management



# Hewlett-Packard Alpha

- The Alpha is a 64-bit CPU
- page frames are 8 KB long, offset is 13 bits
- Only least significant 43 bits of address are used (most significant bits all zero)
- Three level of page tables, so remaining 30 bits of address split into three 10-bit fields. So page tables each hold $2^{10} = 1024$ entries

# Memory management policies

- If low on RAM, which pages should be written to disk?
- How decide which pages to read from disk for a process just starting to run again?