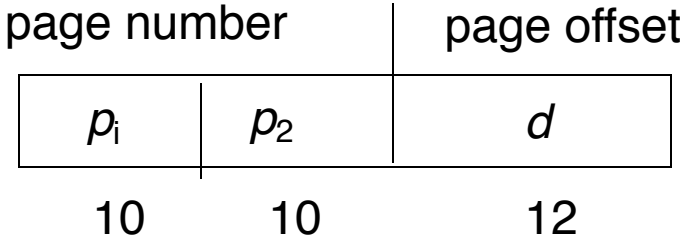
Paging in x86 and TLB

COMS W4118

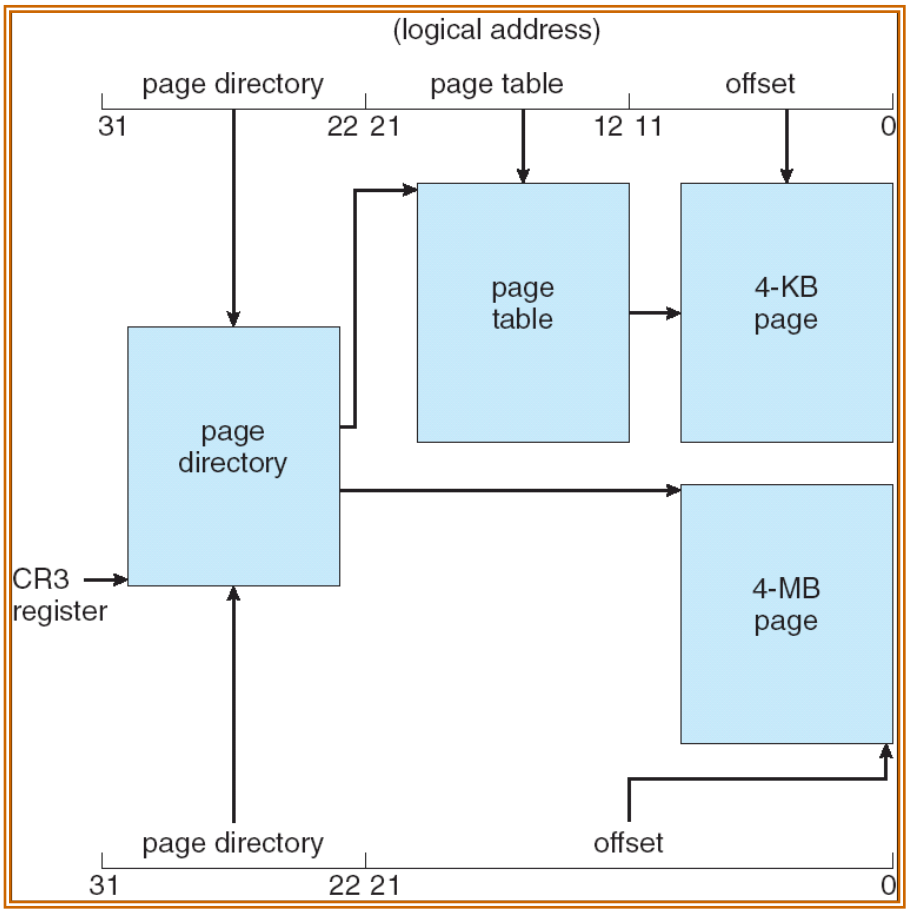
References: Operating Systems Concepts (9e), Linux Kernel Development, previous W4118s

x86 page translation with 4KB pages

- 32-bit address space, 4 KB page
  - 4KB page ➔ 12 bits for page offset
- How many bits for 2nd-level page table?
  - Desirable to fit a 2nd-level page table in one page
  - 4KB/4B = 1024 ➔ 10 bits for 2nd-level page table
- Address bits for top-level page table: 32 – 10 – 12 = 10



x86 paging architecture

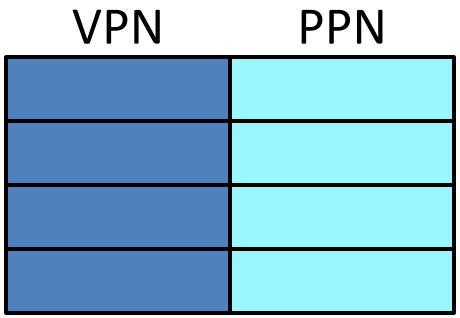


- A better picture here (page 26):
  - http://pdos.csail.mit.edu/6.828/2012/xv6/book-rev7.pdf

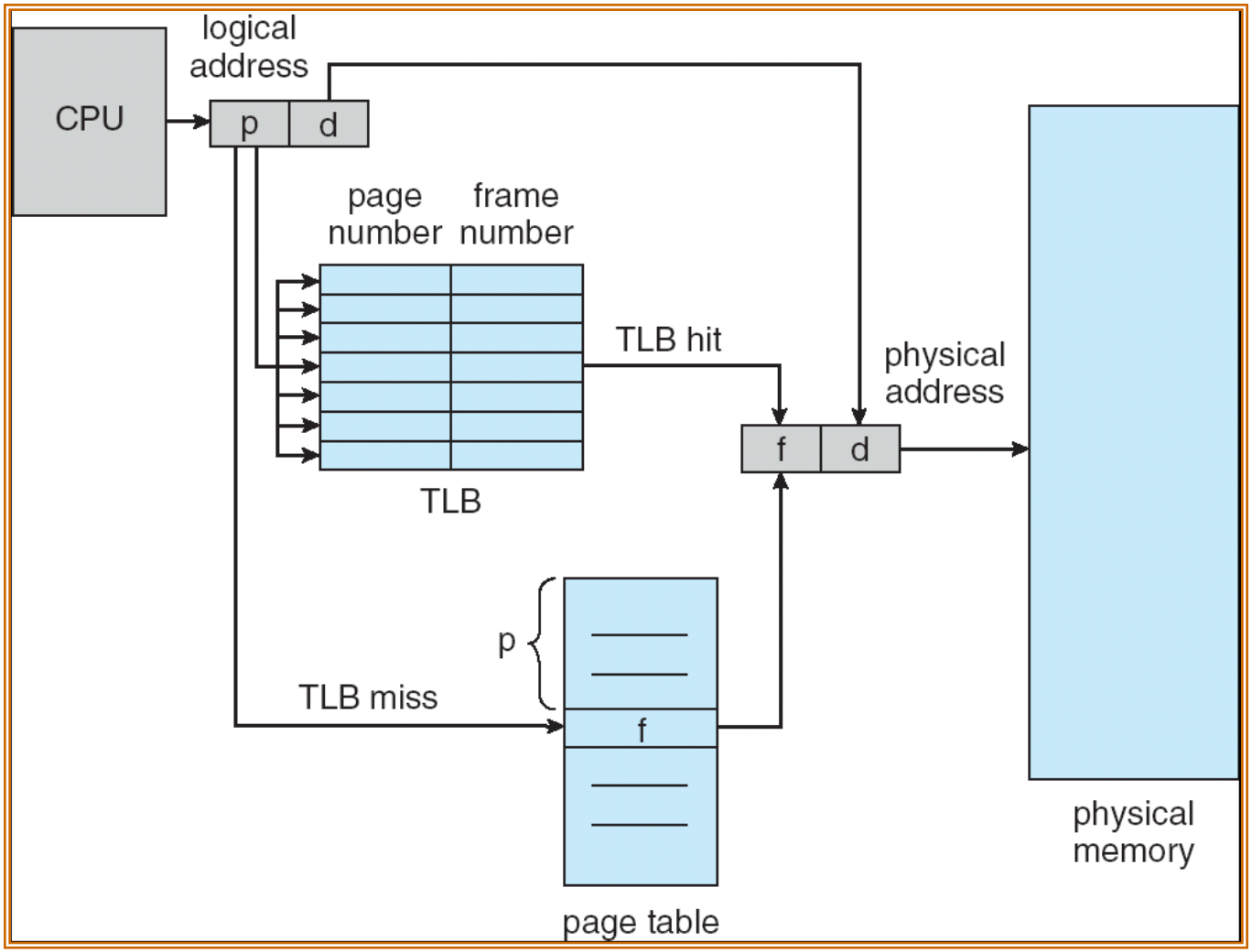
Avoiding extra memory accesses

- Observation: locality
  - Temporal: access locations accessed just now
  - Spatial: access locations adjacent to locations accessed just now
  - Process often needs only a small number of vpn➔ppn mappings at any moment!

- Fast-lookup hardware cache called associative memory or translation look-aside buffers (TLBs)
  - Fast parallel search (CPU speed)
  - Small



Paging hardware with TLB



Effective access time with TLB

- Assume memory cycle time is 1 unit time
- TLB Lookup time = ε
- TLB Hit ratio = α
  - Percentage of times that a vpn➔ppn mapping is found in TLB

- Effective Access Time (EAT)

$$EAT = (1 + \varepsilon)\,\alpha + (2 + \varepsilon)(1 - \alpha)$$
$$= \alpha + \varepsilon\alpha + 2 + \varepsilon - \varepsilon\alpha - 2\alpha$$
$$= 2 + \varepsilon - \alpha$$

TLB and context switches

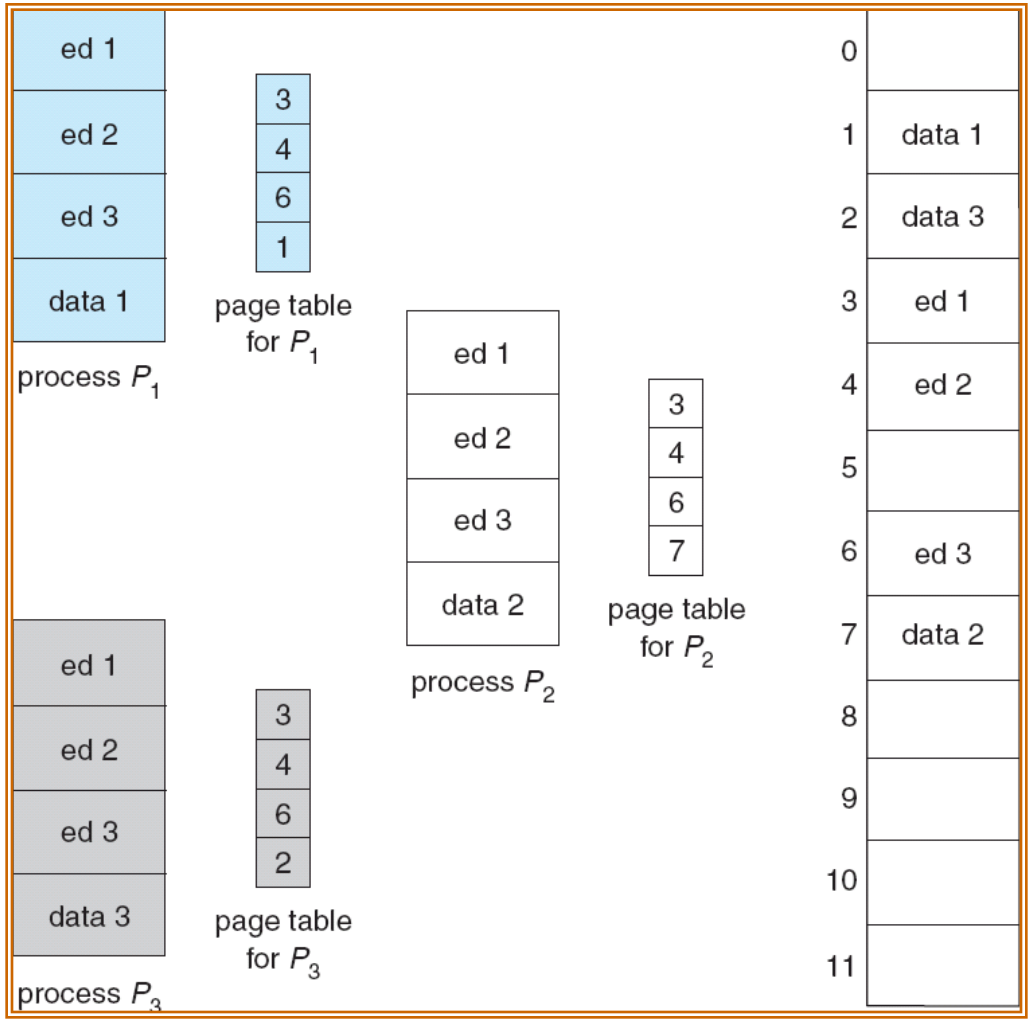
- What happens to TLB on context switches?
- Option 1: flush entire TLB
  - x86
    - "load cr3" (load page table base) flushes TLB
- Option 2: attach process ID to TLB entries
  - ASID: Address Space Identifier
  - MIPS, SPARC
- x86 "INVLPG addr" invalidates one TLB entry

Motivation for page sharing

- Efficient communication. Processes communicate by write to shared pages

- Memory efficiency. One copy of read-only code/data shared among processes
  - Example 1: multiple instances of the shell program
  - Example 2: copy-on-write fork. Parent and child processes share pages right after fork; copy only when either writes to a page

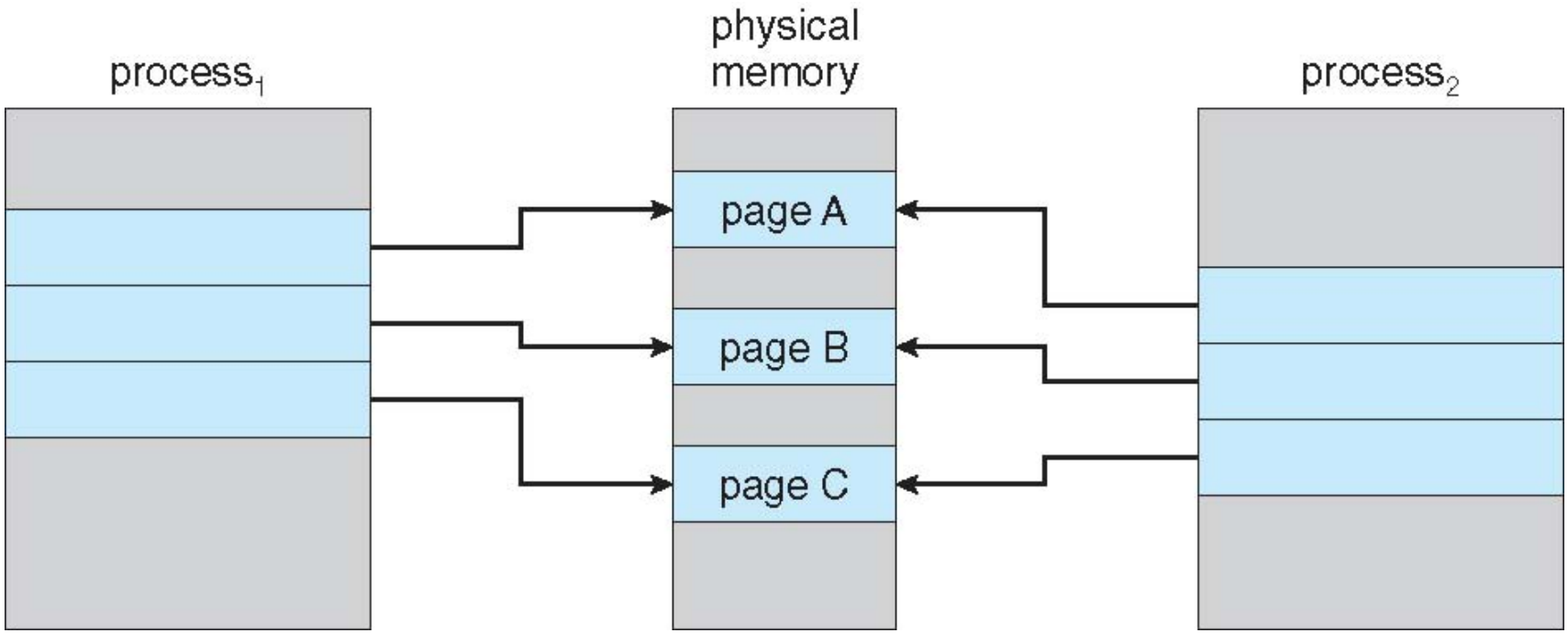
Page sharing example



A cool trick: copy-on-write

- In fork(), parent and child often share significant amount of memory
  - Expensive to copy all pages
- COW Idea: exploit VA to PA indirection
  - Instead of copying all pages, share them
  - If either process writes to shared pages, only then is the page copied
- Real: used in virtually all modern OS

Before Process 1 Modifies Page C



After Process 1 Modifies Page C