Paging in x86 and TLB

COMS W4118

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

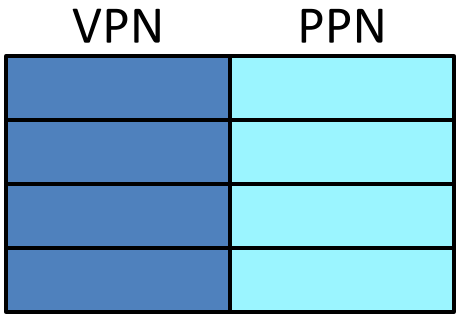
x86 paging at a glance

- 1-level paging in x86-32 with 4KB pages possible?
  - 20-bit page number → 1 million page table entries → 4MB page table per process → way too big!
- 2-level paging in x86-32
  - Default x86-32 paging mode: 10 + 10 + 12
  - http://pdos.csail.mit.edu/6.828/2012/xv6/book-rev7.pdf (page 26)
- 3-level paging in x86-32
  - Physical Address Extension (PAE): 2 + 9 + 9 + 12
  - Still limited to 4GB per process but up to 64GB RAM
- 4-level paging in x86-64
  - Default x86-64 paging mode: 9 + 9 + 9 + 9 + 12
- 5-level paging in x86-64
  - Available in Intel Ice Lake processors: 9 + 9 + 9 + 9 + 9 + 12
  - CONFIG_X86_5LEVEL kernel option from Linux 4.14

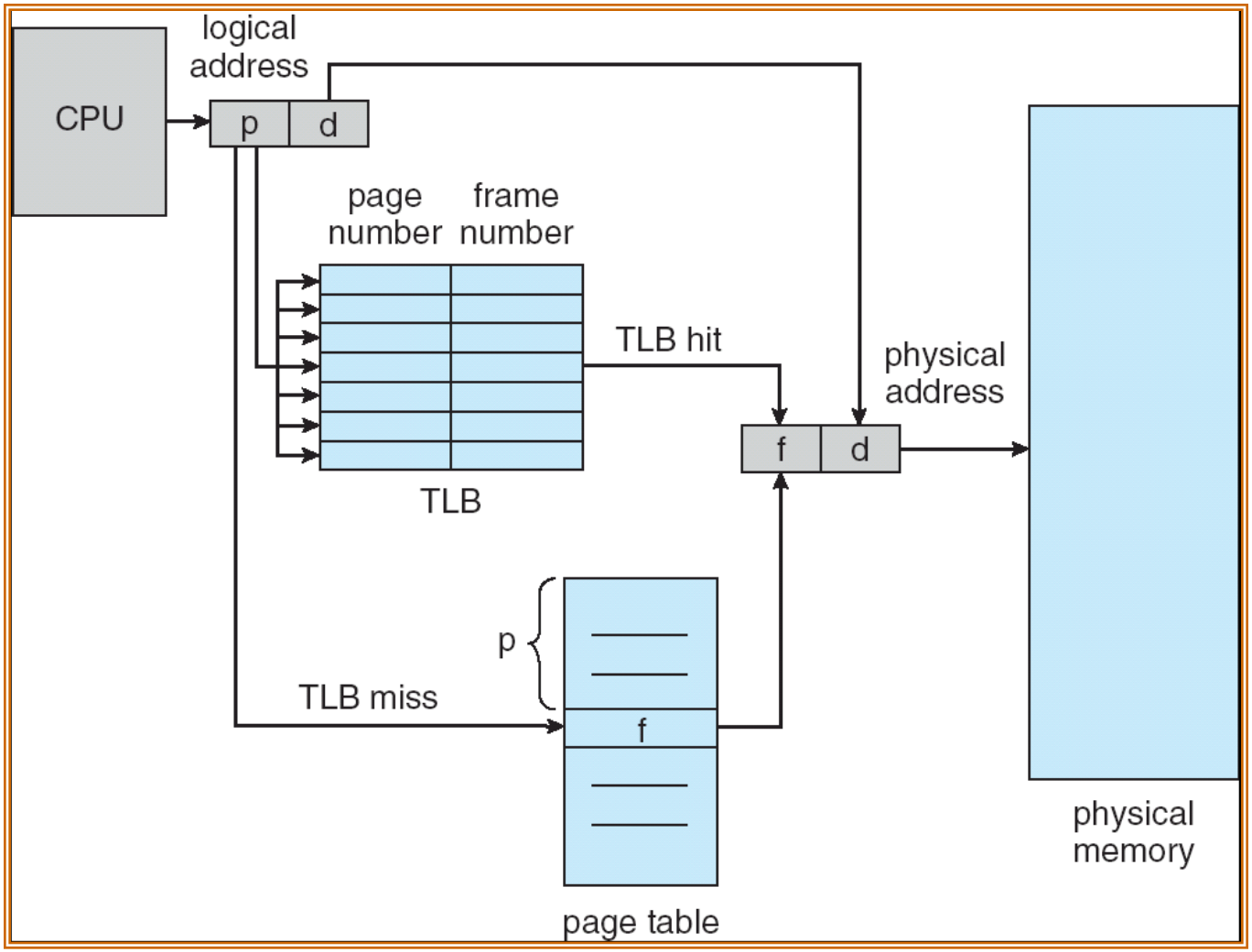
Avoiding extra memory accesses

- 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

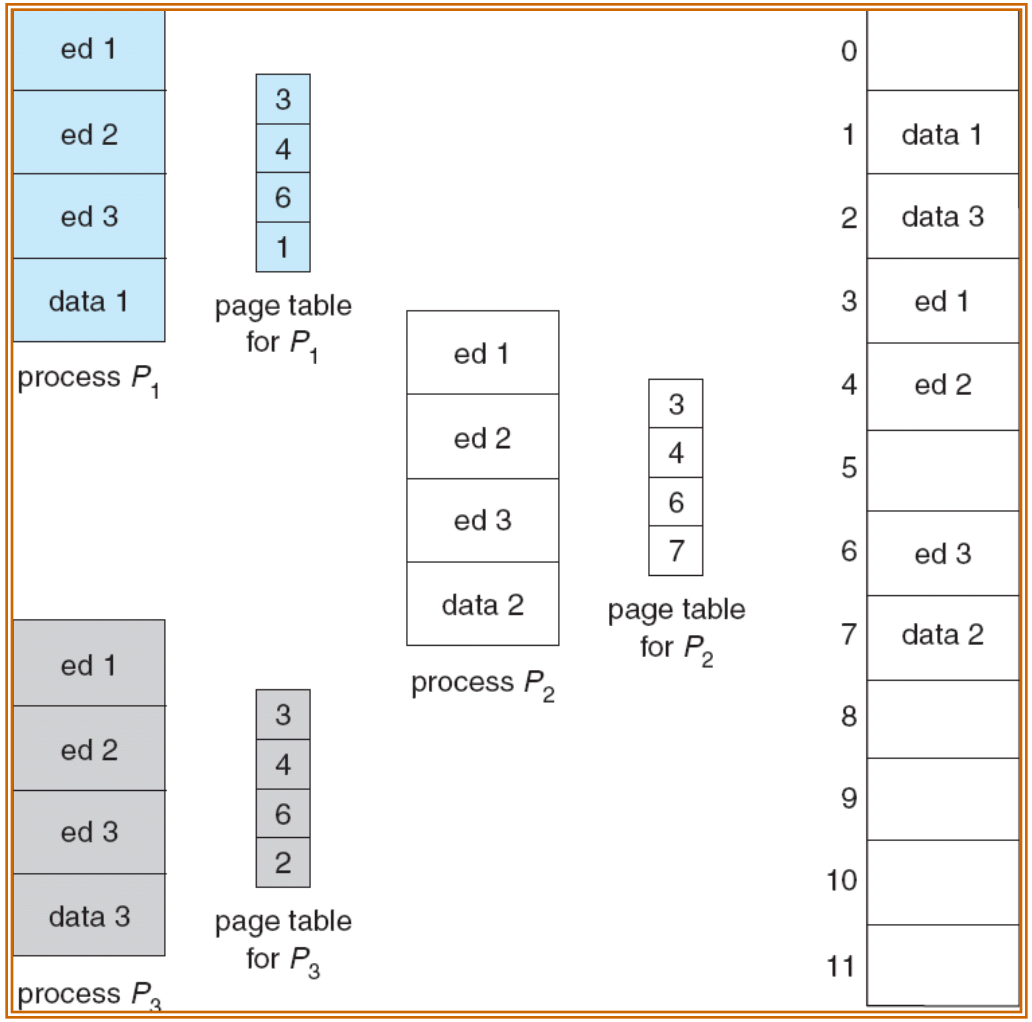
- Effective Access Time (EAT)

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

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

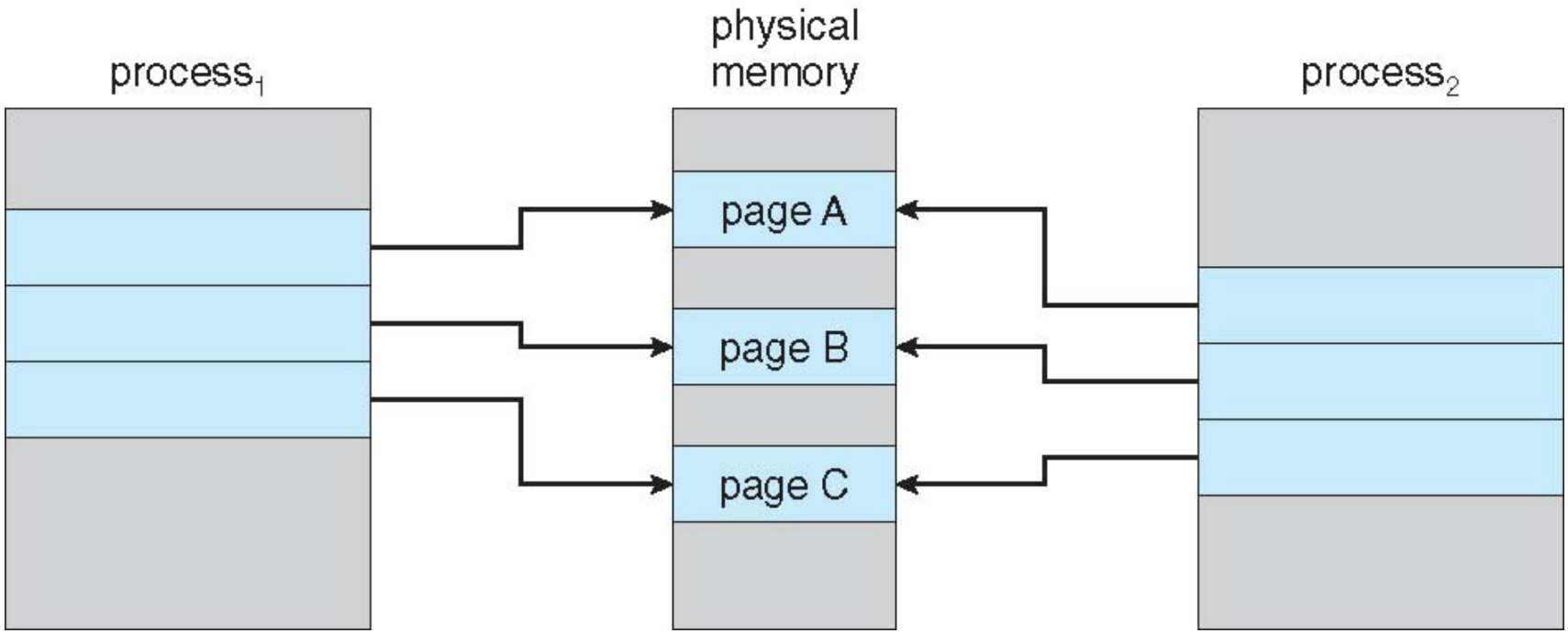
Page sharing



Copy-On-Write (COW)

- 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
- Used in virtually all modern OS

Before Process 1 Modifies Page C



After Process 1 Modifies Page C