CS 492: Operating Systems

Memory Management (2) – Virtual Memory

Instructor: Iraklis Tsekourakis

Email: itsekour@stevens.edu

Some Facts

- A n-bit virtual address space
  - The virtual address consists of n bits.
  - There are 2^n virtual addresses

- A n-bit virtual address space with k bits for offset
  - Each page consists of 2^k virtual addresses
  - The virtual page number takes (n-k) bits
  - There are $2^{(n-k)}$ virtual pages
  - There are $2^{(n-k)}$ entries in the page table

- A m-bit physical address space
  - The physical address consists of m bits.
  - There are 2^m physical addresses

- A m-bit physical address space with k bits for offset
  - Each page frame consists of 2^k physical addresses
  - The page frame number takes (m-k) bits
  - There are $2^{(m-k)}$ physical pages

Exercise 1: Page Table Size

Consider a 32-bit virtual address space. Each page consists of 4096 virtual addresses. Assume each page table entry (PTE) takes 4 bytes.

Question:

1. What is the size of the page table for one process?
2. What is the total size of the page tables for 100 running processes?

Page Tables: Design Issues

- Two issues:
  - Address translation must be fast
  - Large virtual address results in large page table
- In fact
  - All modern computers use virtual address of at least 32 bits. 64 bits is becoming common
  - For 4KB page size, a 32-bit address space has 1 million pages (Recall Exercise 1)
  - Namely, each process has a page table of 1 million entries!
- We need implementation of page tables for
  - Speeding up paging
  - Handling large virtual address spaces.

Motivation of Implementing Page Tables in Hardware

- 1st issue: **speed up** translation from virtual address to physical address
  - Problem: translating an address requires **two** memory accesses
    - 1st access reads page table entry (PTE)
    - 2nd access reads the data / instruction from memory

- Can we reduce the number of accesses?

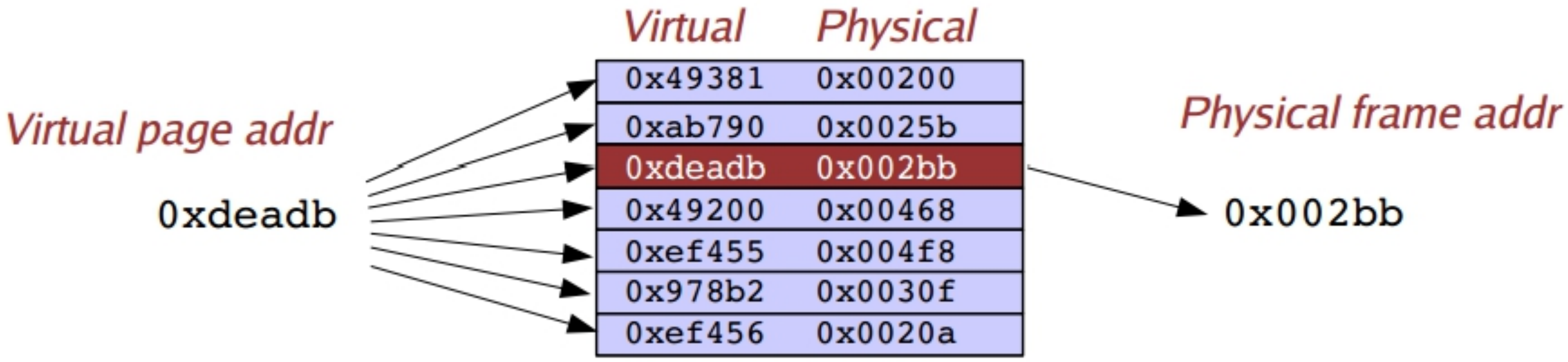
Translation Lookaside Buffer (TLB)

Solution: eliminate the 1st access by caching recent mappings in *translation lookaside buffer* (TLB)

– Why does it work? Because most programs often make a large number of references to a small number of pages

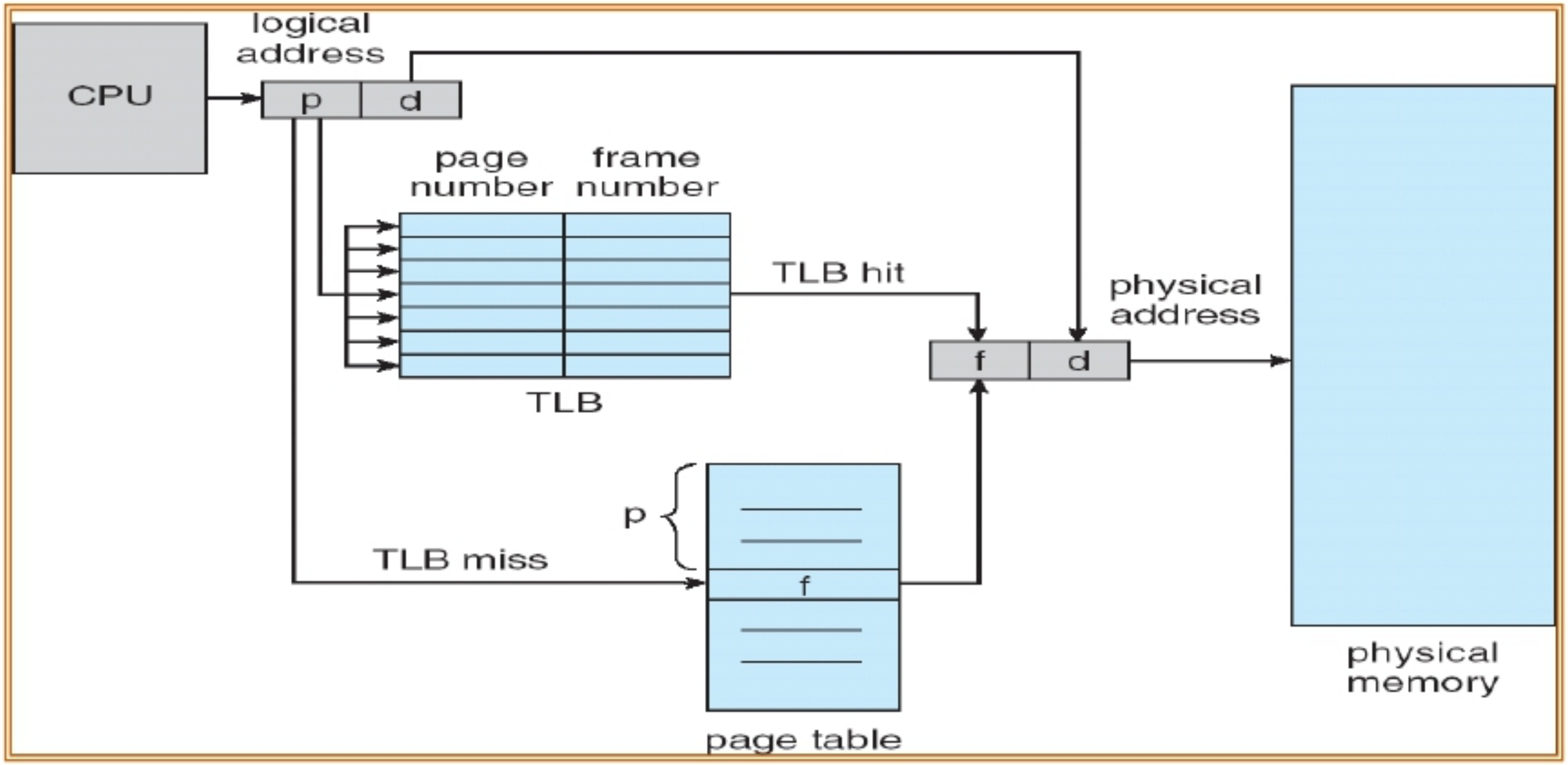
Translation Lookaside Buffer (TLB)

- TLB is part of MMU
  - Where does it reside?
- TLB is very fast
- TLB is small: approximately 64-256 PTE's in TLB
- Caches most recent virtual to physical address translations



TLB

How to Use TLB for Address Translation?



A TLB *miss* requires the MMU to do address translation

TLB Example

	Valid	Virtual page	Modified	Protection	Page frame
Index used in loop	1	140	1	RW	31
	1	20	0	R X	38
Data	1	130	1	RW	29
	1	129	1	RW	62
	1	19	0	R X	50
	1	21	0	R X	45
Stack	1	860	1	RW	14
	1	861	1	RW	75

A TLB to speed up paging for a loop that spans virtual pages 19, 20 and 21

How long do memory accesses take?

- Assume the following times:
  - TLB lookup time = *a* (often zero - overlapped in CPU)
  - Memory access time = *m*
- Hit ratio (*h*) is percentage of time that a logical page number is found in the TLB
  - Larger TLB usually means higher h
  - TLB structure can affect h as well
- What is the effective access time (EAT)?
  - $\text{EAT} = \underset{\substack{\uparrow \\ \text{Hit pages}}}{(m + a)h} + \underset{\substack{\uparrow \\ \text{Non-hit pages}}}{(m + m + a)(1\text{-}h)} = a + (2\text{-}h)m$

Outline

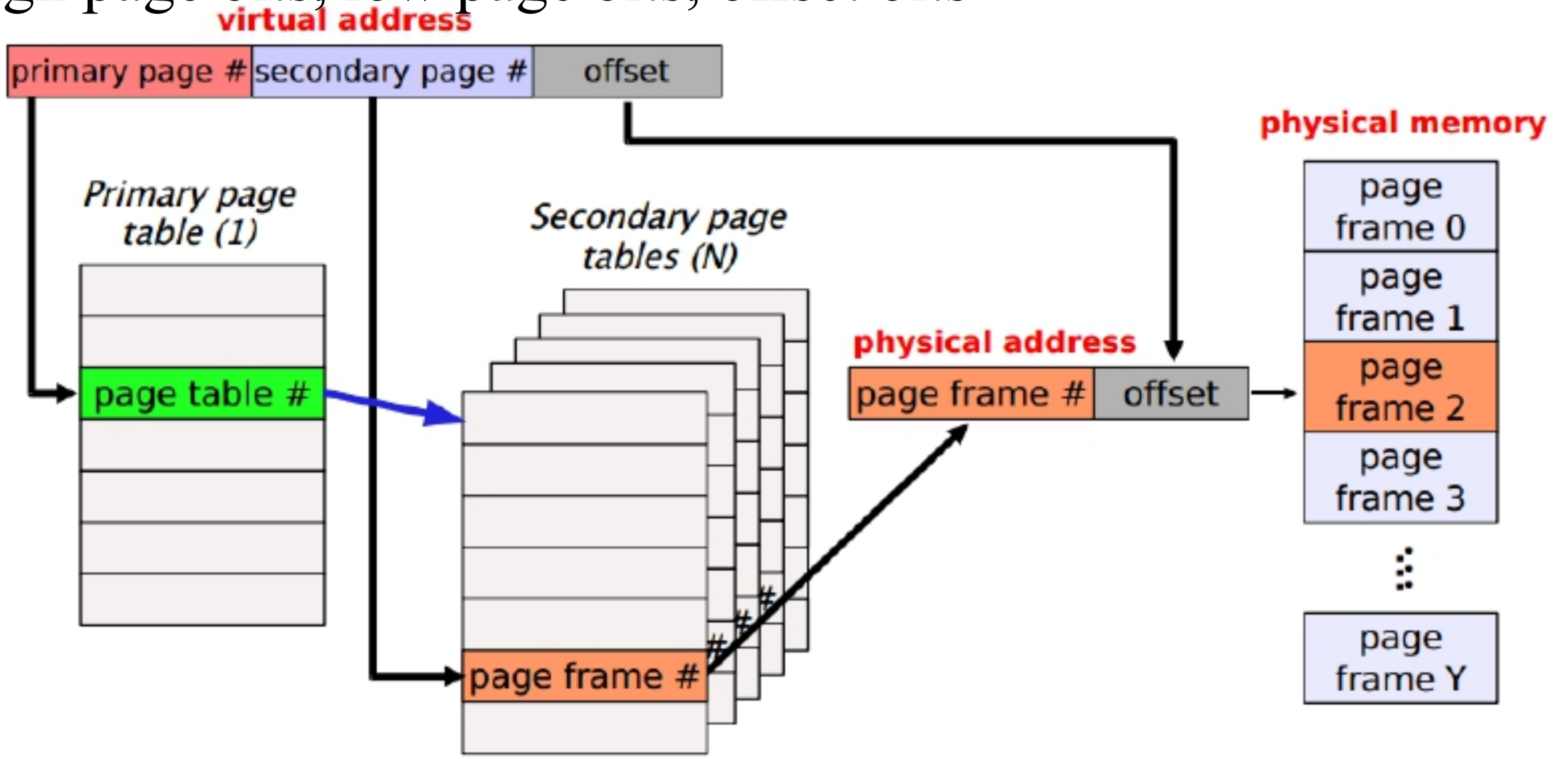
Recall

- Two design issues of page tables:
  - Address translation must be fast
  - Large virtual address results in large page table

- We need implementation of page tables for
  - Speeding up paging
    - Solution: TLB
  - Handling large virtual address spaces (next)

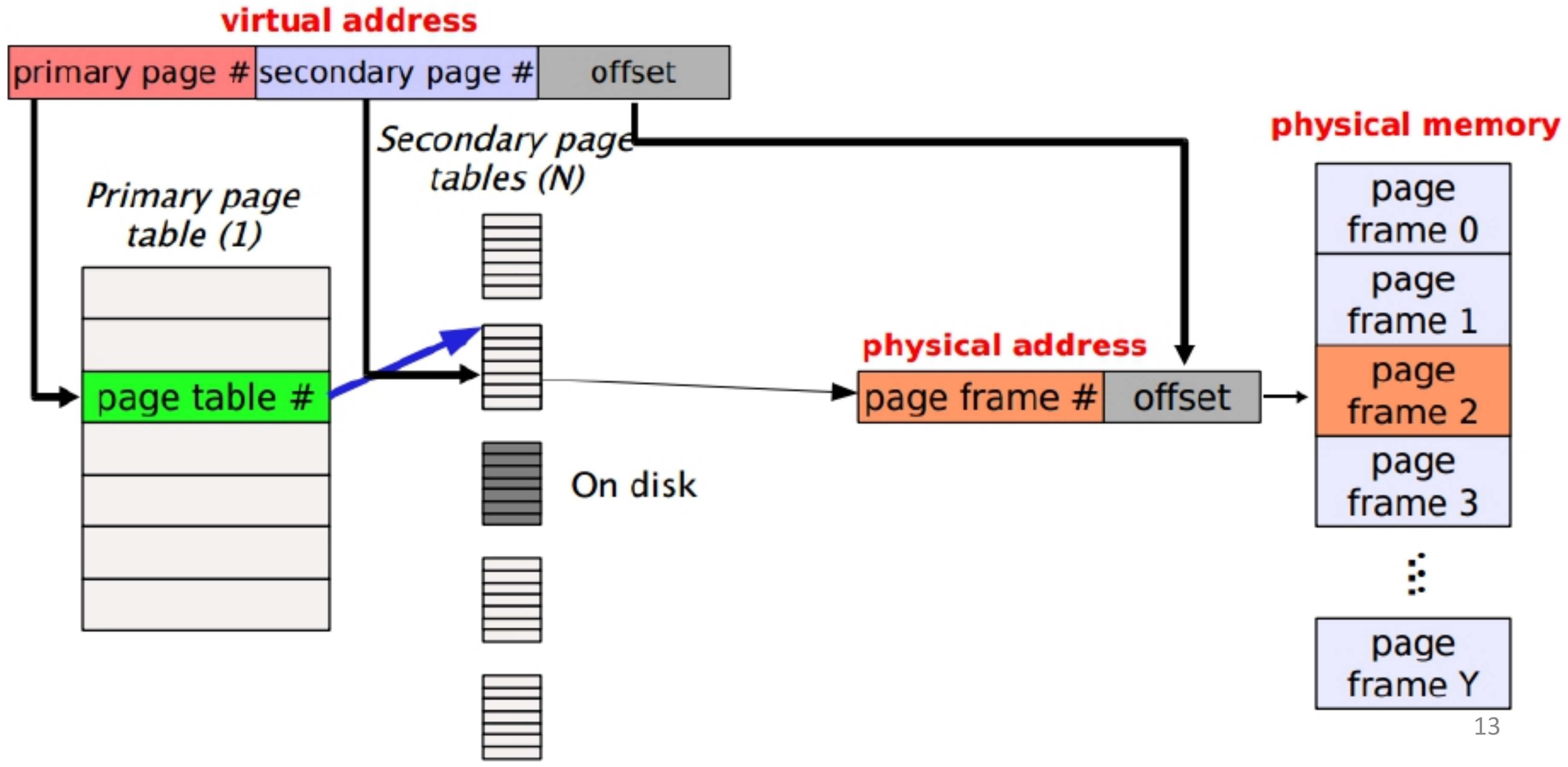
Solution 1: Multi-level Page Tables

- Problem: Can't hold all of the page tables in memory
- Basic idea: *page the page tables*
- Multi-level page tables: divide virtual address into 3 portions – high page bits, low page bits, offset bits



How Does It Work?

- Multi-level page tables allow portions of the page tables to be kept in memory at one time



How Big Is Multi-level Page Table?

Consider a 32-bit virtual address space. Each page consists of 4K virtual addresses. Also assume each page table entry (PTE) takes 4 bytes.

- Recall (Exercise 1): A 1-level page table takes 4MB!
- Question: assume the 32-bit address is allocated as following: 10 bits to the primary page, 10 bits to the secondary page, 12 bits to the page offset.
  - What is the size of each 2-level page table?
  - What is the total size of the page tables (including 1-level and 2-level ones)?
  - How much memory is needed for one virtual address translation?

Multi-level Page Tables (Cont.)

- Multi-level page tables can be extended to N levels as the address space gets larger
  - X86-64 (AMD64 and Intel64) support up to 4 levels

- However, multi-level page tables still can be of large size

Solution 2: Inverted Page Table

- Each entry in the page table is for a *real* page of memory (page frame), rather than for a *virtual* page
  - Size of page table = # of page frames (much smaller than # of pages)
- Mapping: page frame -> virtual page

Inverted Page Tables (Cont.)

- Increases time needed to search the table to translate virtual addresses to physical addresses
  - Have to search the whole table to find the mapping

- Is there a better way?