



University of
Nottingham

UK | CHINA | MALAYSIA

Operating Systems and Concurrency

Lecture 18:
Memory Management 4

Edited by: Dr Qian Zhang
University of Nottingham, Ningbo China
2023



Goals for Today

Overview

- Principles behind **virtual memory**
- Complex/large **page tables**



- The **principles of paging** are:
 - Main memory is divided into small equal sized **frames**
 - Each process is divided into **pages** of equal size
 - A page table contains multiple “relocation registers” (**page table**) to map the pages on to frames
- The **benefits** of paging include:
 - Reduced **internal fragmentation**
 - No **external fragmentation**



Virtual Memory

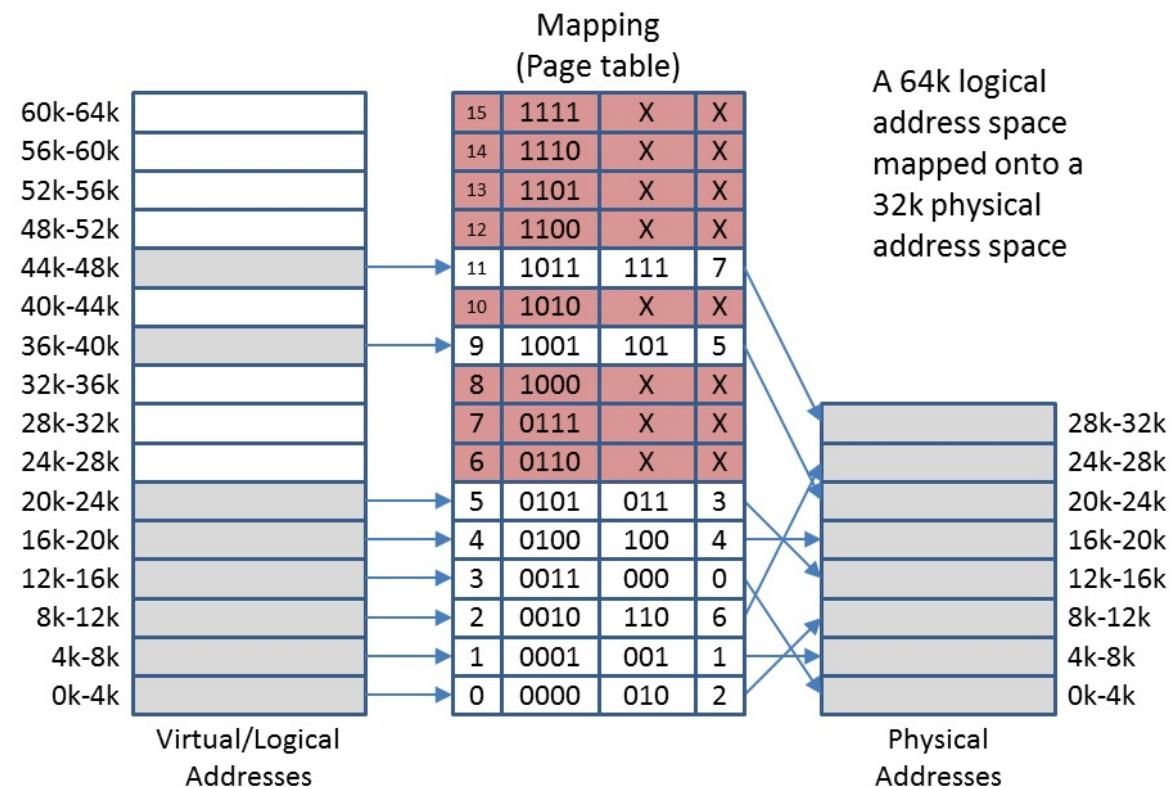
Principle of Locality

- **Principle of Locality:** the program and data references within a process tend to **cluster**.
 - **localities** constitute **groups of pages** that are **used together**, e.g., related to a function (code, data, etc.)
 - Code execution and data manipulation are usually **restricted to a small subset** (i.e. limited number of pages) at any point in time
 - I.e. **code and data references** within a process are usually **clustered** => This is called the **principle of locality**
- **Not all pages** have to be **loaded** in memory at the same time => **virtual memory**
 - Loading an entire set of pages for an entire program/data set into memory is **wasteful**
 - Desired blocks could be **loaded on demand**

Virtual Memory

Definition

- Virtual Memory: A storage allocation scheme in which **secondary memory** can be addressed as though it were part of main memory.





Simple Paging	Virtual Memory Paging
Main memory partitioned into small fixed-size chunks called frames.	
Program broken into pages by the compiler or memory management system.	
Internal fragmentation within frames.	
No external fragmentation.	
Operating system must maintain a page table for each process showing which frame each page occupies.	
Operating system must maintain a free-frame list.	
Processor uses page number, offset to calculate absolute address.	
All the pages of a process must be in main memory for process to run, unless overlays are used.	Not all pages of a process need be in main memory frames for the process to run. Pages may be read in as needed.
	Reading a page into main memory may require writing a page out to disk.



Virtual Memory

Page Faults

- The **resident set** refers to the pages that are loaded in main memory
- A **page fault** is generated if the processor accesses a page that is **not in memory**
 - A page fault results in an **interrupt** (process enters **blocked state**)
 - An **I/O operation** is started to bring the missing page into main memory
 - A **context switch** (may) take place
 - An **interrupt signals** that the I/O operation is complete (process enters **ready state**)

Virtual Memory

The Benefits

- Being able to maintain **more processes** in main memory through the use of virtual memory **improves CPU utilisation**
 - Individual processes take up less memory since they are only **partially** loaded
- Virtual memory allows the **logical address space** (i.e processes) to be **larger than physical address space** (i.e. main memory)
 - 64 bit machine --> 2^{64} logical addresses (theoretically)

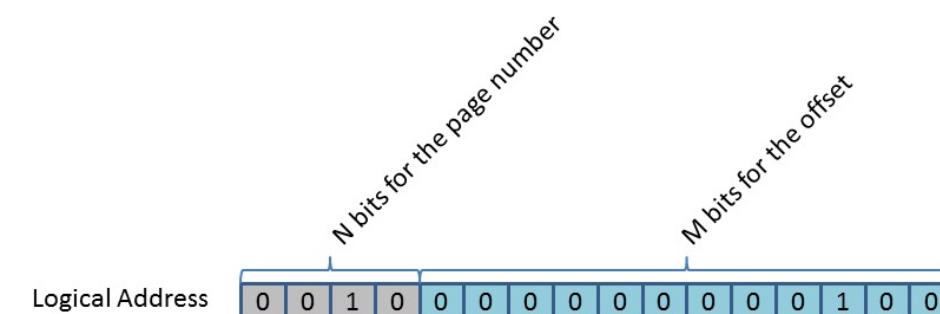


Figure: Logical Address --> physical address

Virtual Memory

Page Tables Revisited: Contents of a Page Entry

- A “**present/absent bit**” that is set if the page/frame is in memory (page fault)
- A “**modified bit**” that is set if the page/frame has been modified (only modified pages have to be written back to disk when evicted) (page usage)
- A “**referenced bit**” that is set if the page is in use (page usage)
- **Protection and sharing bits:** read, write, execute or combinations thereof

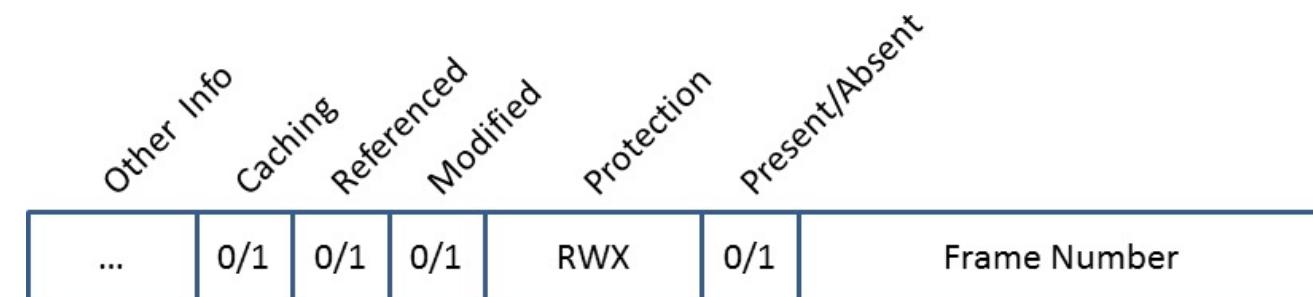


Figure: Page Table Entry



Virtual Memory

Page Tables Revisited: Page Table Size

- On a **16 bit machine**, the total address space is 2^{16}
 - Assuming that 10 bits are used for the offset (2^{10})
 - 6 bits can be used to number the pages
 - I.e., $2^6 = 64$ pages can be maintained
- In a **32 bit machine**, total address space is 2^{32}
 - Assuming pages of 2^{12} bits (4kb)
 - 20 bits can be used to number the pages
 - I.e. 2^{20} pages (approx. 1 million) can be maintained
- On a **64 bit machine** . . .



- How do we deal with **the increasing size of page tables**, i.e., where do we store them?
 - Their size prevents them from being **stored in registers**
 - They have to be stored in (virtual) **main memory**:
 - **Multi-level** page tables
 - **Inverted page tables** (for large virtual address spaces)
- How can we maintain **acceptable speeds**: address translation happens at every memory reference, it has to be fast!
 - Accessing main memory results in **memory stalls**

- **Solution: Page the page table!**
- We keep tree-like structures to hold page tables
- Divide the page number into
 - An index to a page table of second level
 - A page within a second level page table
- No need to keep all page tables in memory all time

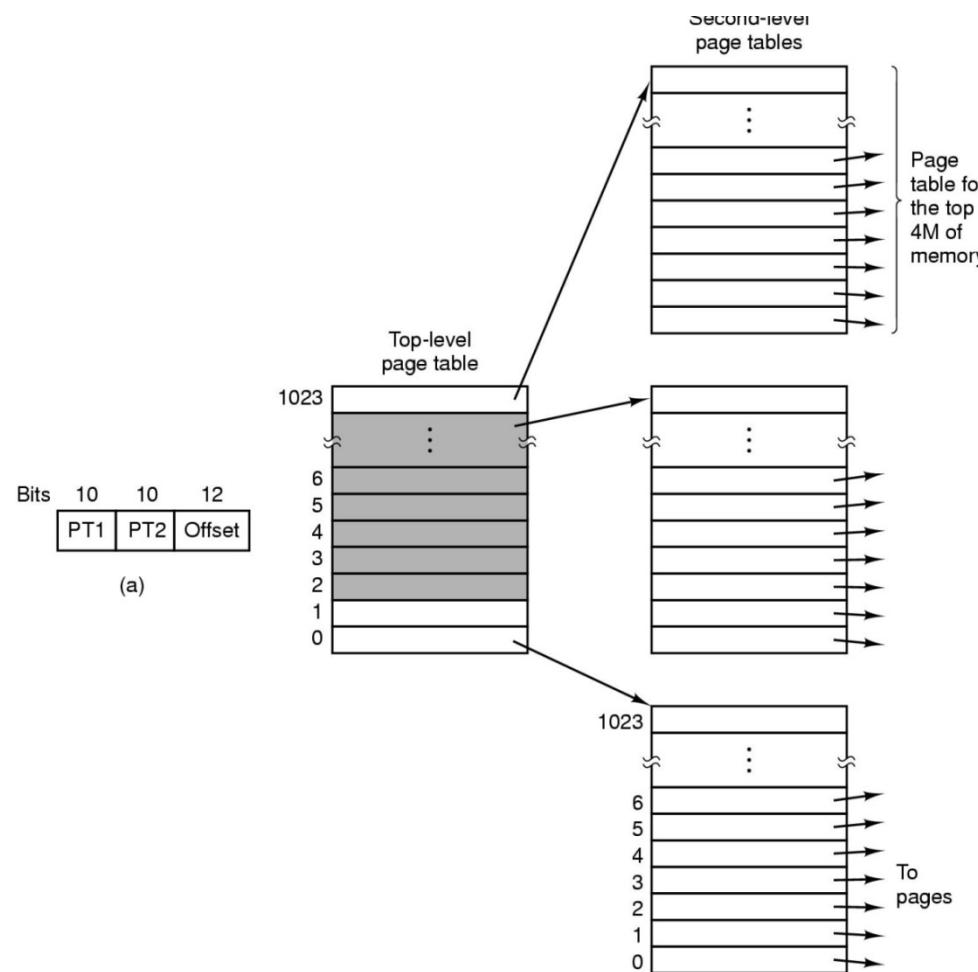


Figure: Multi-level page tables (from Tanenbaum)

Virtual Memory

Page Table Revisited: Multi-level Page Tables

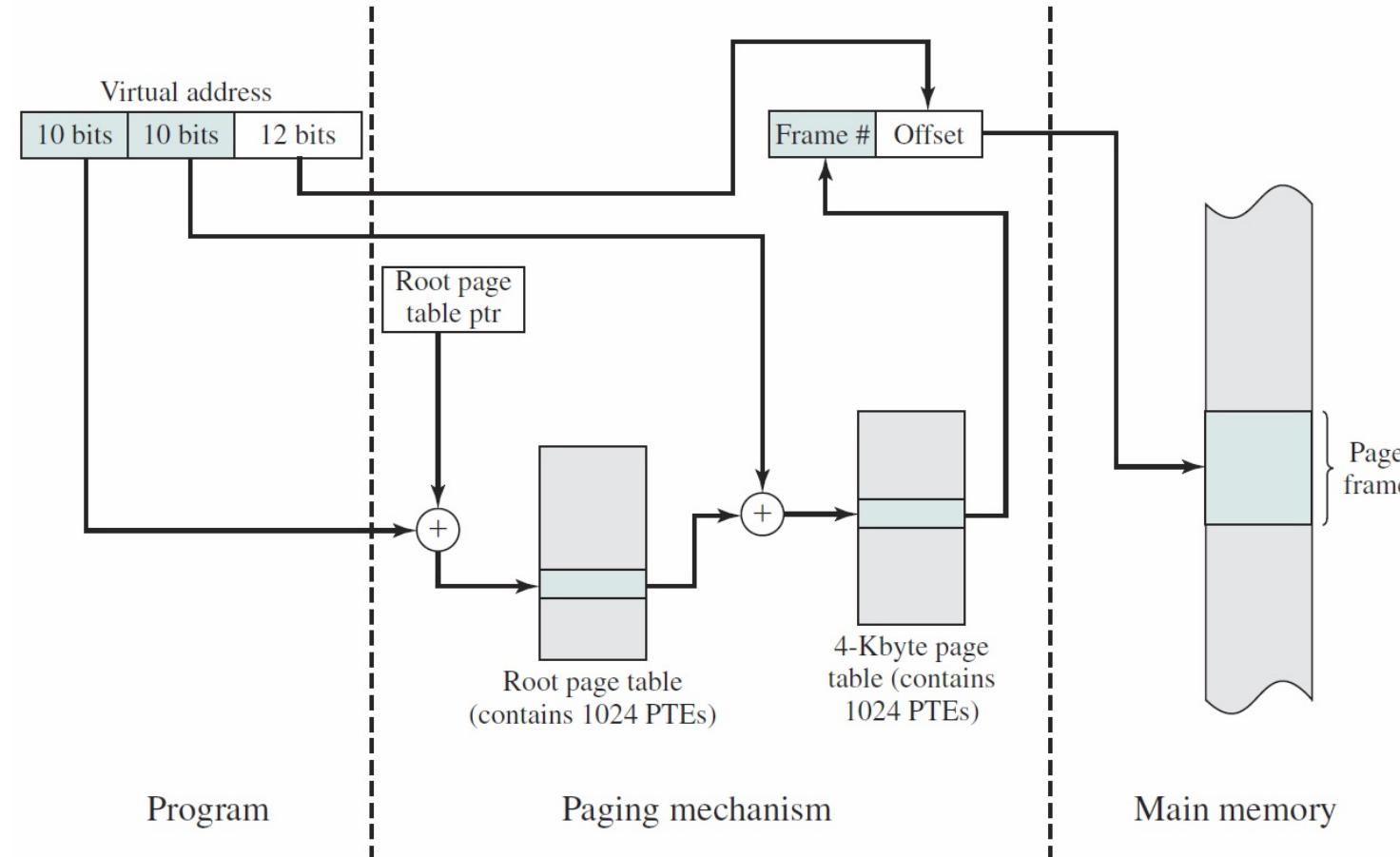


Figure: Multi-level Address Translation (from Stallings)



Virtual Memory

Page Tables Revisited: Access Speed

- **Memory organisation** of multi-level page tables:
 - The **root page table** is always maintained in memory
 - Page tables themselves are maintained in **virtual memory** due to their size
 - Page table size is proportional to that of the virtual address space
- Assume that a **fetch** from main memory (\Rightarrow memory access) takes T nano seconds
 - With a **single page table level**, access is $2T$
 - With **two page table levels**, access is $3T$
 - . . .



- **Translation look aside buffers (TLBs)** are (usually) located inside the memory management unit
 - They **cache** the most **frequently** used page table entries
 - They can be searched **in parallel**
- The principle behind TLBs is similar to other types of **caching in operating systems**
- Remember: **locality** states that processes make a large number of references to a **small number of pages**

Virtual Memory

Translation Look Aside Buffers (TLBs)

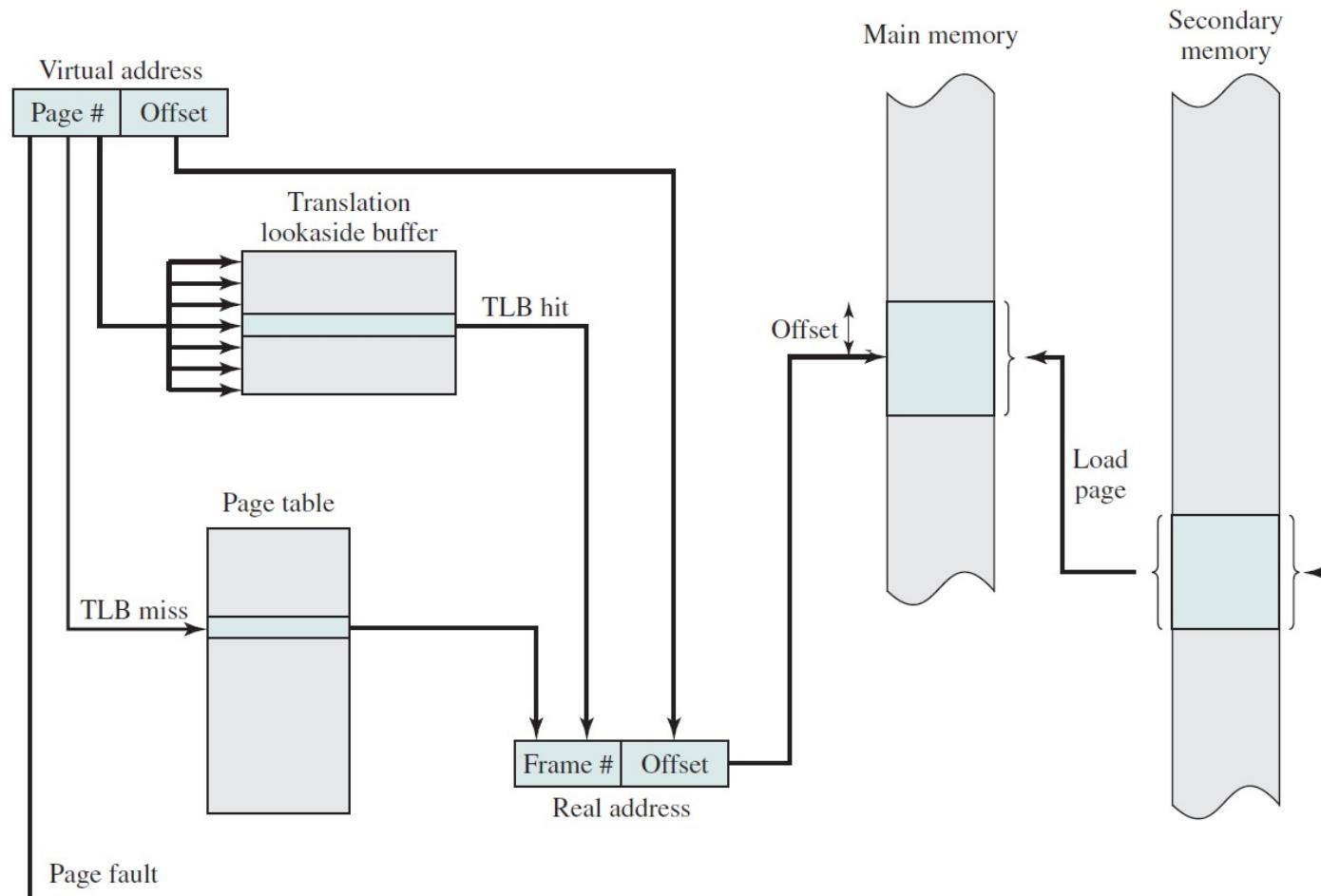
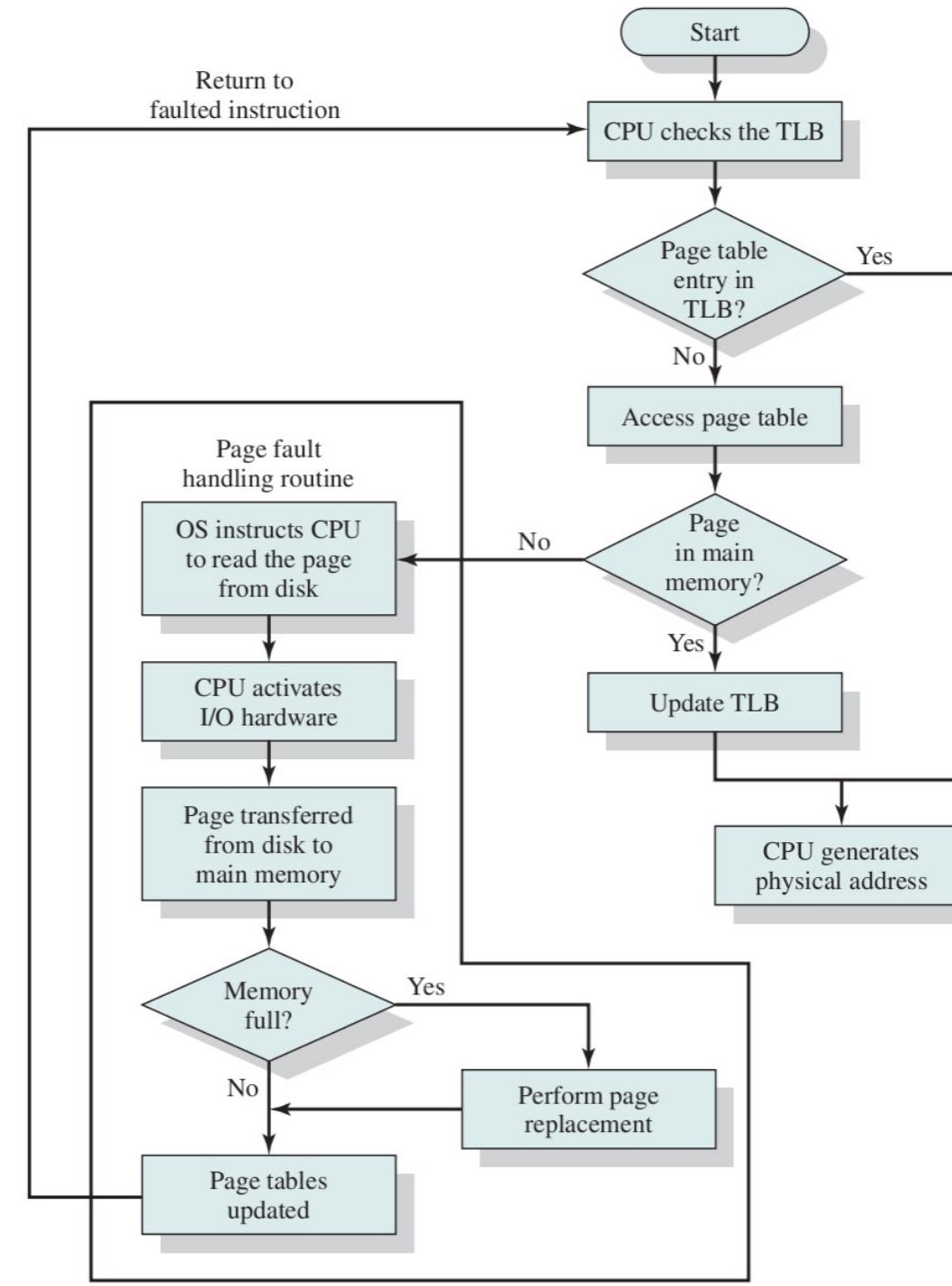


Figure: TLB Address Translation (from Stallings)



- Memory access with TLBs:
 - Assume a 20ns associative **TLB lookup**
 - Assume a 100ns **memory access time** and with a single level page table
 - **TLB Hit** → $20 + 100 = 120$ ns
 - **TLB Miss** → $20 + 100 + 100 = 220$ ns
- Performance evaluation of TLBs:
 - For an 80% hit rate, the estimated access time is:
 - $120 \times 0.8 + 220 \times (1 - 0.8) = 140$ ns (i.e. 40% slowdown)
 - For a 98% hit rate, the estimated access time is:
 - $120 \times 0.98 + 220 \times (1 - 0.98) = 122$ ns (i.e. 22% slowdown)
- Note that **page tables** can be **held in virtual memory** → further (initial) slow down due to page faults



- A “**normal**” page table’s size is proportional to the number of pages in the virtual address space (drawback of multi-level/single-level page table)
- An “**inverted**” page table’s size is proportional to the size of main memory
 - The inverted table contains one **entry for every frame** (i.e. not for every page)
 - A **hash function** based on the page number is used to index the inverted page table
 - The inverted table **indexes entries by frame number**, not by page number
- The OS maintains a **single inverted page table** for all processes

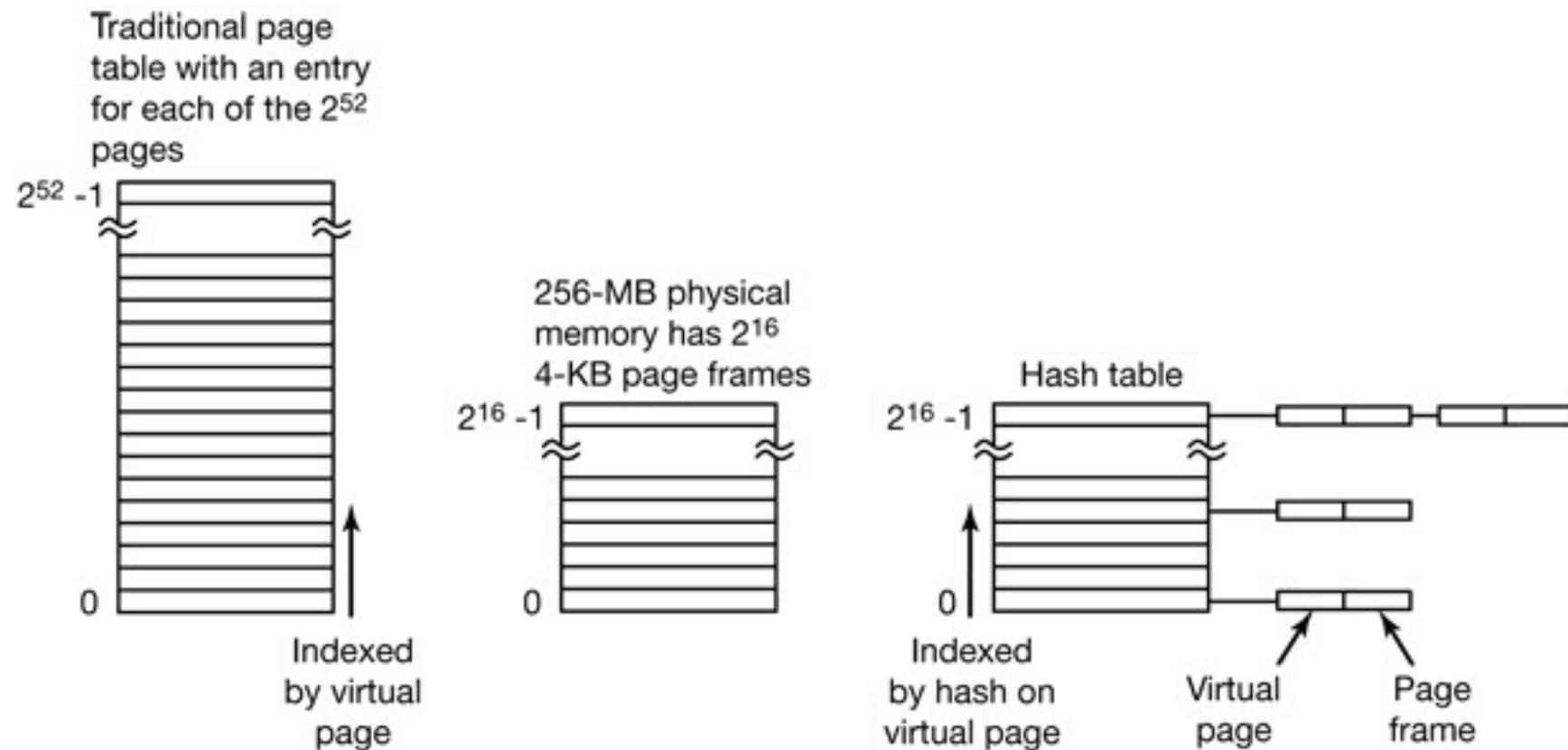


Figure: Inverted Page Table (from Stallings)



- Advantages:
 - The OS maintains a **single inverted page table** for all processes
 - It saves lots of space (especially when the virtual address space is much larger than the physical memory)
- Disadvantages:
 - Virtual-to-physical **translations becomes much harder**. We need to use hash tables to avoid searching the whole inverted table (be aware of potential collisions)
- It is used in combination with TLBs to speed up the search.
- Commonly used on 64-bit machines (e.g. Windows 10)



Summary

Take-Home Message

- **Paging** splits logical and physical address spaces into small **pages/frames** to reduce internal and external fragmentation
- **Virtual memory** exploits the principle of **locality** and allows for processes to be **loaded only partially** into memory, **large logical address spaces** require “different” approaches



University of
Nottingham

UK | CHINA | MALAYSIA

Quiz!



Exercise

- Assume a machine that has a 42-bit virtual address space, and a 32-bit physical address space (i.e., the maximum amount of physical memory for a 32-bit address space is present). How many entries would you expect a single level page table to have when the page size is 4 kilobytes?



Exercise

- Briefly explain the principle behind address relocation. Why it is a necessity on modern day interactive/multi-tasking machines. Explain how address relocation works in virtual memory with paging. Include an illustration.



Exercise

- When creating a child process in Linux using fork(), would the child process have its own page table, or would its page table be shared with the parent process. Briefly explain your answer.