# CS420/520
# Computer Architecture I

## Virtual Memory

**Dr. Xiaobo Zhou**
**Department of Computer Science**



## Review: The Principle of Locality


- **The Principle of Locality:**
  - **Program access a relatively small portion of the address space at any instant of time.**
  - **Example: 90% of time in 10% of the code**

## Re: Summary

- The Principle of Locality: Temporal Locality vs Spatial Locality
- Four Questions For Any Cache
  - Where to place in the cache
  - How to locate a block in the cache
  - Replacement: Random, LRU, NRU, LFU
  - Write policy: Write through vs Write back
    - Write miss: Write Allocate vs. Write Not Allocate
    - Write buffer
- Three Major Categories of Cache Misses:
  - Compulsory Misses: sad facts of life. Example: cold start misses.
  - Conflict Misses: increase cache size and/or associativity.
    Nightmare Scenario: ping pong effect!
  - Capacity Misses: increase cache size
- Three general options to improve cache performance
  - Reduce the miss rate / increase the hit rate,
  - Reduce the miss penalty, or
  - Reduce the time to hit in the cache



## Today's Topic --- Virtual Memory

**Provides *illusion* of very large memory**
– sum of the memory of many jobs greater than physical memory
– address space of each job larger than physical memory

**Allows available (fast and expensive) physical memory to be very well utilized**

**Simplifies memory management (*main reason today*) – do we need VM if the main memory is already as large as the virtual address of a program?**

**Exploits memory hierarchy to keep average access time low.**

**Involves at least two storage levels: *main* and *secondary (Disk)***

***Virtual Address* -- address used by the programmer**

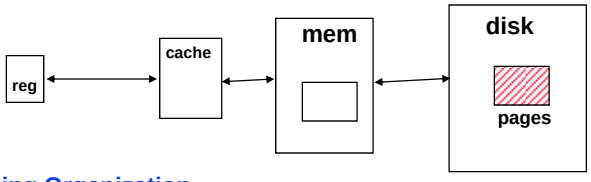
***Virtual Address Space* -- collection of all such addresses**

***Memory Address* -- address of word in physical memory**
**also known as "physical address" or "real address"**

***Memory Address Space* - collection of all such addresses**

## Basic Issues in VM System Design

- What is the size of information blocks that are transferred from secondary to main storage → *fixed-size page*; segmentation
- Which region of M is to hold the new block --> *placement policy*
- How is a block found if it is in main memory -> *location policy*
- If block of information brought into M, and M is full, then some region of M must be released to make room for the new block --> *replacement policy*
- Missing item fetched from secondary memory only on the occurrence of a fault --> *fetch/load policy;* and what happens on a *write?*

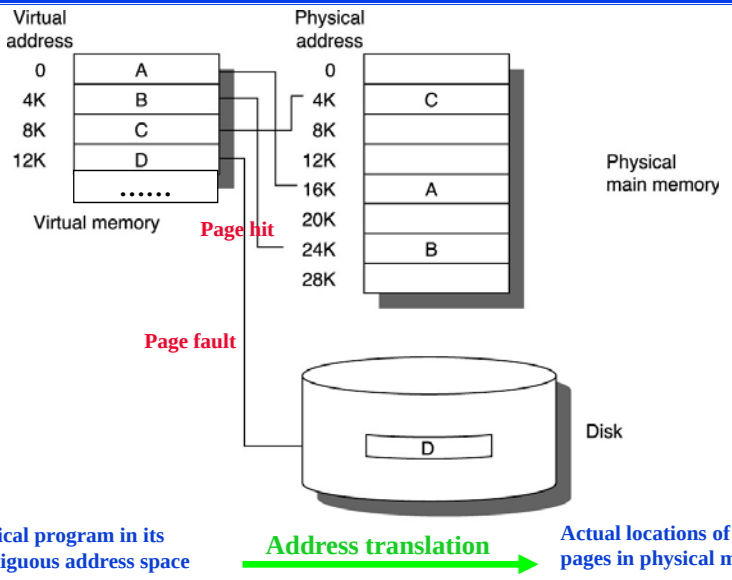



Paging Organization

virtual and physical address space partitioned into blocks of equal size



## Mapping of Virtual addresses to Physical addresses



## Optimal Page Size

° Most machines at 4K to 16K byte pages today, with page sizes likely to increase towards 32KB and 64KB

- Size of the page table is inversely proportional to the page size
- Pages should be large enough to amortize the high access time, transferring larger pages is more efficient
  - Access time vs. transfer time
- But small page size means less waste (*internal fragmentation*), more flexibility of dealing with different sizes of code



## Typical Ranges of Parameters for Cache and VM

| Parameter | First-level Cache | Virtual Memory |
|---|---|---|
| Block (page) size | 16-128 Bytes | 2^12 – 2^16 bytes |
| Hit time | 1-3 clock cycles | 50 – 150 clock cycles |
| Miss Penalty | 8-150 clock cycles | 1M-10M clock cycles |
| (access time) | 6-130 clock cycles | 0.8M-8M clock cycles |
| (transfer time) | 2-20 clock cycles | 0.2M–2M clock cycles |
| Miss rate | 0.1 – 10 % | 10^ - 4 – 10^-5% |
| Address mapping | 25-45 bit physical addr. → 14-20 bit cache addr. | 32-64 bit virtual addr. → 20-45 bit physical addr. |

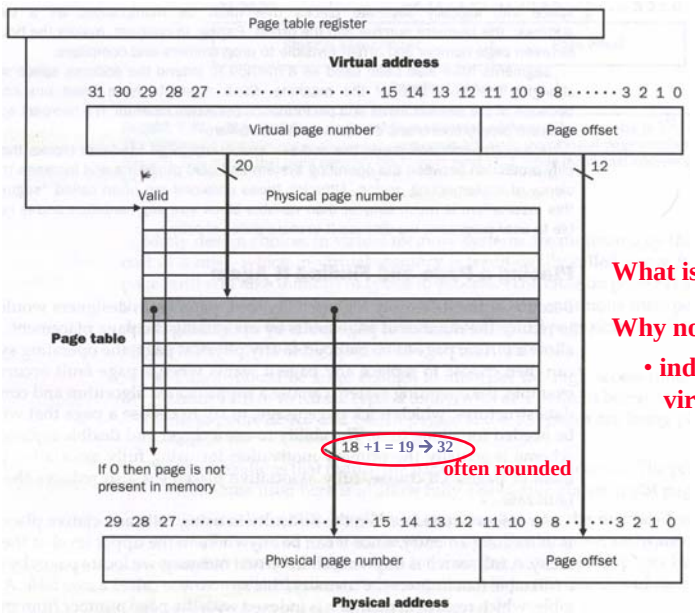
Hit time << miss penalty

## More Differences between Cache and VM

- Replacement on misses
  - In cache, is primarily controlled by hardware
  - In VM, is controlled by operating system (OS)
    - Longer miss penalty means the value of good decision
    - OS can afford to use sophisticated algorithms and complex data structures
- Size:
  - In cache: independent of the processor address size
  - In VM: size of the processor address determines VM' size
- Sharing
  - In cache: all space dedicates to caching purpose
  - In VM: the secondary storage is also used for file system



## Q1: Placing a Page in MM (Fully Associative)

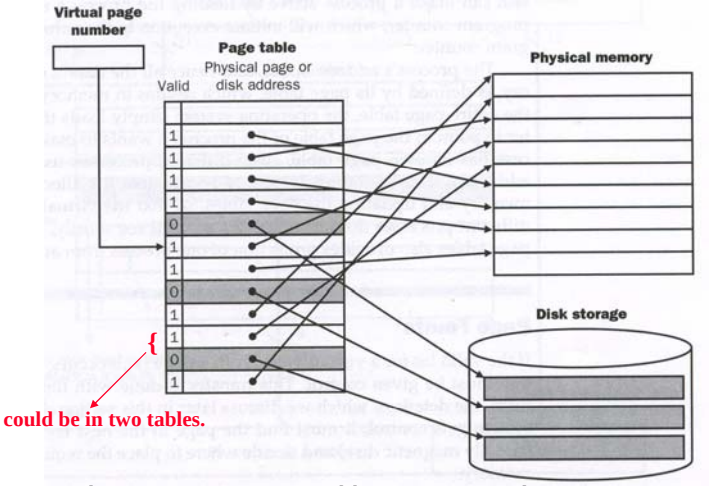



What is page size?

Why no tags?

- indexed with the virtual page #

## Q2: Finding a Page in Memory (or in Disk)




- Two data structures created by OS on creating a process
  - To record where each virtual page is stored on disk (in PT or not )
  - To track which virtual address(es) use each physical page (for replacement)



## Page Table Size

Given a 32-bit virtual address,
4 KB pages,
4 bytes per page table entry (memory addr. or disk addr.)

What is the size of the page table?

The number of page table entries:

2^32 / 2^12 = 2^20

The total size of page table:

2^20 * 2^2 = 2^22 (4 MB)

When we calculate Page Table size, the index itself (virtual page number) is often NOT included!

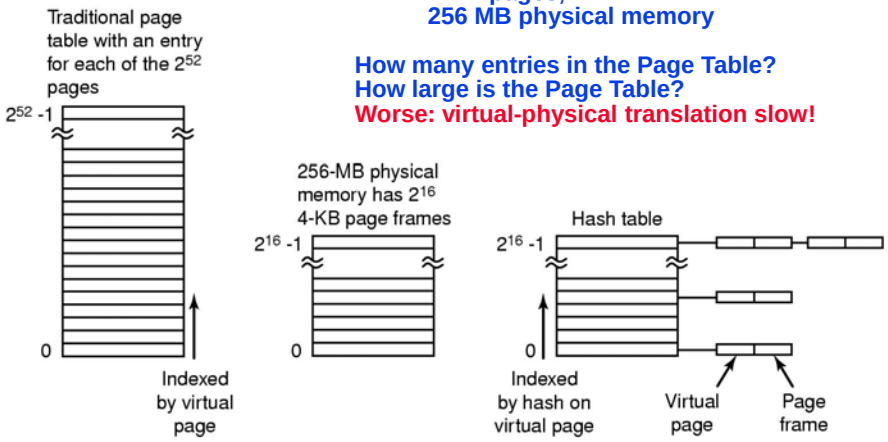
What if the virtual memory address is 64-bit?

## 64-bit Machines: Inverted Page Tables

° **Inverted page table: one entry per page frame in physical memory, instead of one entry per page of virtual address space.**

**Given a 64-bit virtual address,**
**4 KB pages,**
**256 MB physical memory**

**How many entries in the Page Table?**
**How large is the Page Table?**
**Worse: virtual-physical translation slow!**




**Comparison of a traditional page table with an inverted page table**



## Q3: Page Replacement Algorithms

° **Just like cache block replacement!**

° **Least Recently Used (LRU)**

- **Selects the least recently used page for replacement**
- **Good performance, recognizes principle of locality**
- **Expensive to implement**
  - **Requires updating a data structure on every memory reference**
    - entries from most recently referenced to least recently referenced; when a page is referenced it is placed at the head of the list; the end of the list is the page to replace
- **Alternative**
  - **approximate LRU by keeping track of which pages have and which pages have not been recently used**
    - Use/reference bit; periodically cleared by OS' clock interrupt
    - to find a page among the Not Recently Used ones to replace

## Example:

**Suppose the most recent page references (in order) were**
**10, 12, 9, 7, 11, 10**
**When page 8 is referenced, which was not present in memory, and the memory is full.**

**Which page should be replace in LRU?**



## Re: Page Replacement Algorithms (Continued)

***Not Recently Used (NRU):***

**Associated with each page is a reference flag (use/reference bit) such that reference flag = 1 if the page has been referenced in recent past**
**= 0 otherwise**

**-- if replacement is necessary, choose any page such that its reference bit is 0. This is a page that has not been referenced in the recent past period**

**In practice, NRU is more complex, depending on both R and W bits; see OS.**

## Demand Paging and Pre-fetching Pages

### *Fetch Policy*

when is the page brought into memory?
if pages are loaded solely in response to page faults, then the policy is *demand paging*

### An alternative is *pre-fetching (pre-paging):*

anticipate future references and load such pages before their actual use, usually for the *working set* of specific processes

\+ reduces page transfer overhead

\- removes pages already in page frames, which could adversely affect the page fault rate

\- predicting future references usually difficult (but remember what happened before is not)

Most systems implement demand paging without pre-paging,
but pre-paging is useful in multiprogramming (OS)

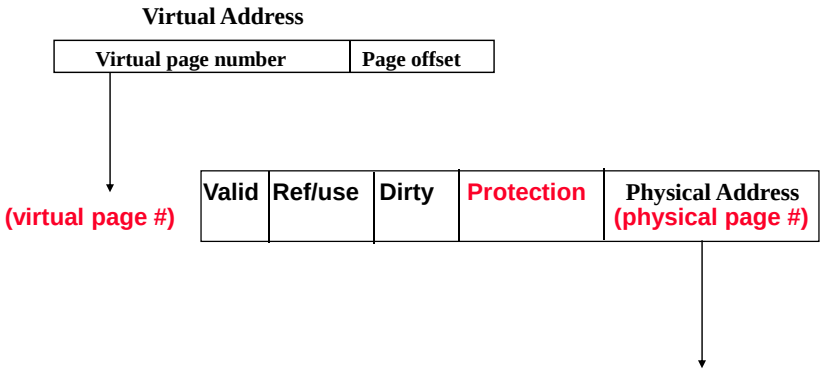


## Q4: Write Issue

° What happens on a write?

- Great access time in the disk, none has yet built a VM that write though main memory to disk
  - Write strategy is always write back w/ dirty bit

## Entry in Page Table






## Translation Look-Aside Buffers

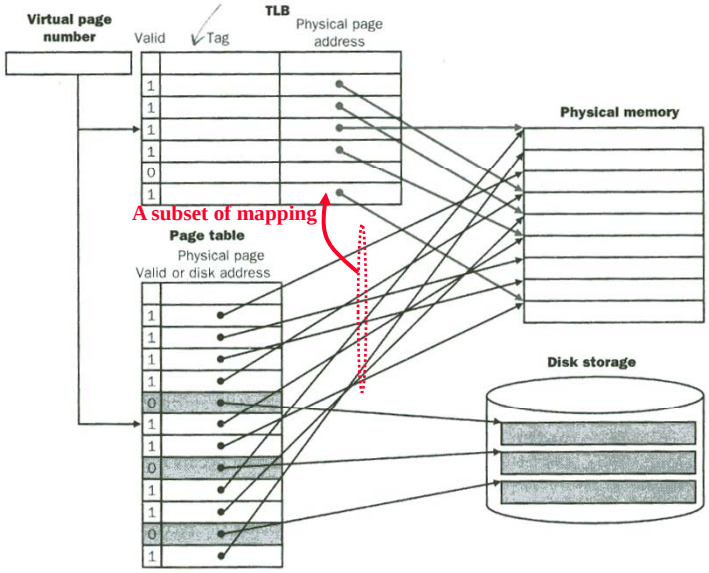
A way to speed up address translation is to use a special cache of recently used page table entries -- this has many names, but the most frequently used is *Translation Lookaside Buffer* or *TLB*

| Virtual page number (virtual page #) | Valid | Ref/use | Dirty | Protection | Physical Address (physical page #) |
|---|---|---|---|---|---|
| | | | | | |

TLB access time comparable to cache access time;
much less than Page Table (usually in main memory) access time

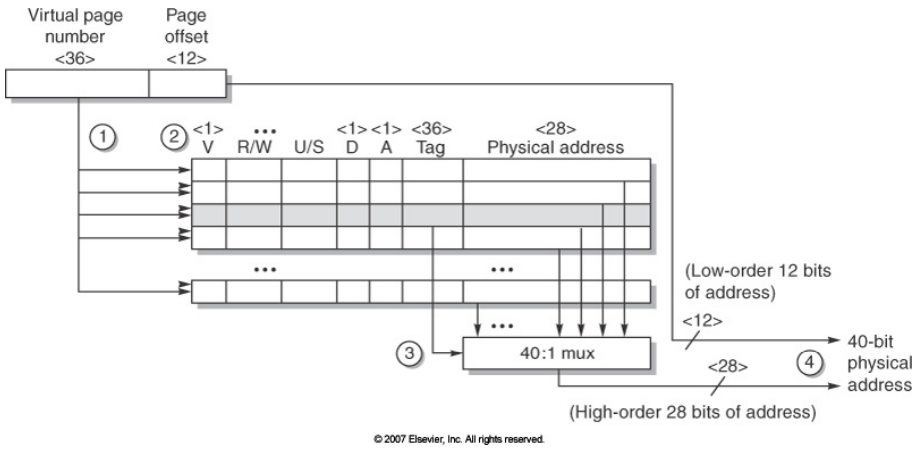
Traditionally, TLB management and handling were done by MMU Hardware, today, more in software

## Acting of Translation Look-Aside Buffers






## Real World: Operation of the Opteron data TLB





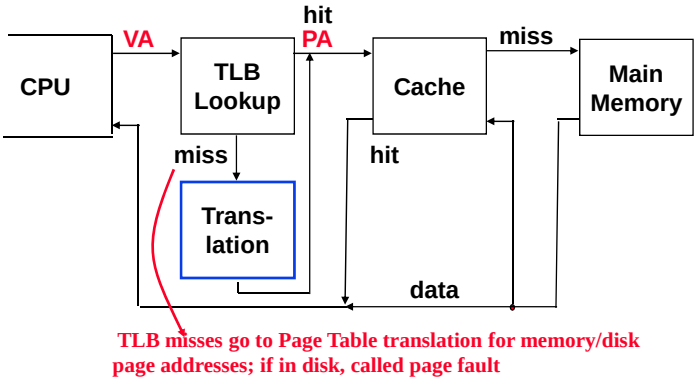

**The TLB uses fully associative placement.**

## Integrating TLB, Cache, and VM

Just like any other cache, the TLB can be organized as fully associative, set associative, or direct mapped

TLBs are usually small, typically not more than 128 - 256 entries even on high end machines. This permits fully associative lookup on these machines. Most mid-range machines use small n-way set associative organizations.

*Translation with a TLB*




TLB misses go to Page Table translation for memory/disk page addresses; if in disk, called page fault

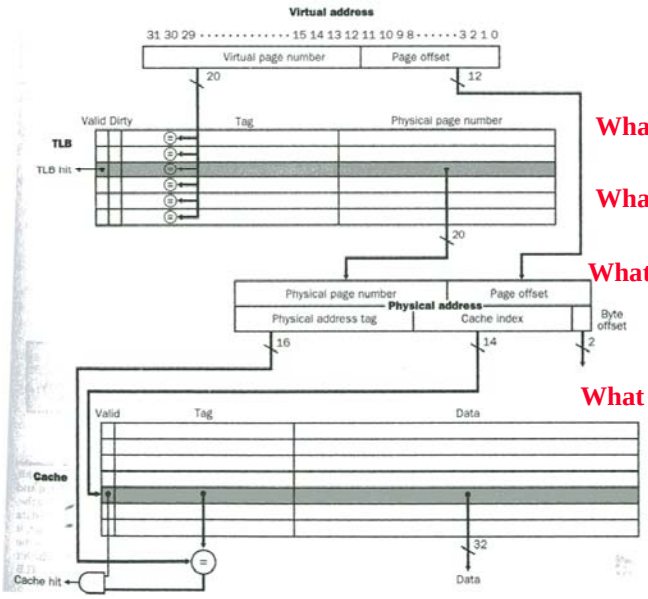


## Hardware/Software Interface

° Process: *state* of a program

- Page table + program counter (PC) + the registers
- Active and inactive processes
  - Possession of the CPU or not
- To make a process active, OS loads the process's state, including PC
  - Execution starts at the value of the saved PC
  - Process migration

## An Example – DECStation 3100




What is TLB scheme?

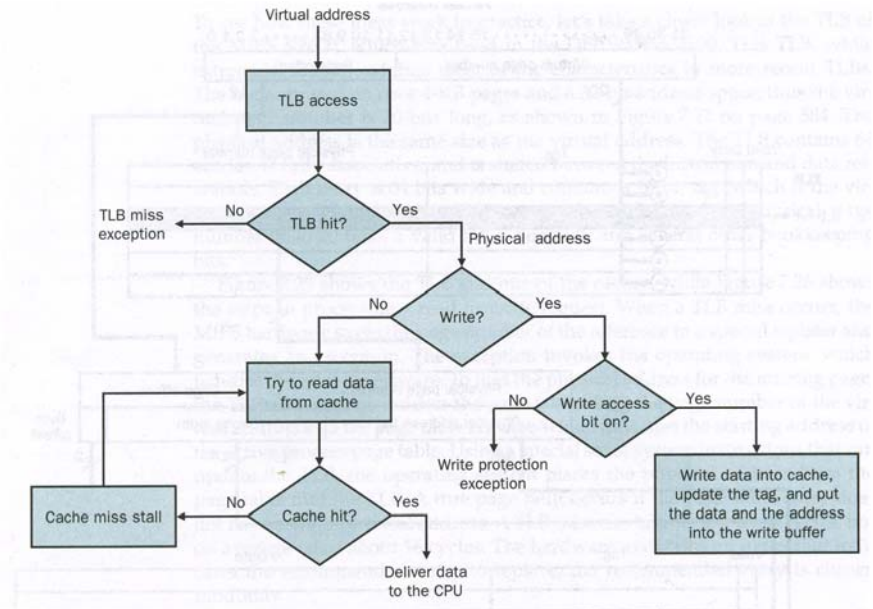
What if TLB miss?

What is page/block size?

What is cache scheme?



## Read and Write in DECStation 3100



## Common Framework for Memory Hierarchy

- Question 1: Where can a Block be Placed
  - Cache:
    - direct mapped, n-way set associative
  - VM:
    - fully associative – to reduce page fault rate
- Question 2: How is a block found
  - index,
  - index the set and search among elements
  - search all cache entries or separate lookup table
  - VM: a data structure (and TLB)
- Question 3: Which block be replaced
  - Random, LRU, NRU, FIFO
- Question 4: What happens on a write
  - write through vs write back
  - VM: write-back; write-through takes too long!



## Summary of Virtual Memory

- Virtual Memory invented as another level of the hierarchy
- Today VM allows many processes to share single memory without having to swap all processes to disk, protection is more important
- (Multi-level) page tables to map virtual address to physical address
- TLBs are important for fast translation
- TLB misses are significant in performance
- More information
  CO 4: Chapter 5
  CA 5: Appendix B

## Homework, due one week later

- See course Web site