香港中文大學
The Chinese University of Hong Kong

CENG3420

Lecture 09: Virtual Memory & Performance

Bei Yu

(Latest update: March 19, 2020)

Spring 2020

Overview

Introduction

Virtual Memory

- $VA \rightarrow PA$
- TLB

Performance Issues

Overview

Introduction

Virtual Memory

- VA $\rightarrow$ PA
- TLB

Performance Issues

Motivations

Physical memory may not be as large as "possible address space" spanned by a processor, e.g.

- A processor can address 4G bytes with 32-bit address
- But installed main memory may only be 1GB

How if we want to simultaneously run many programs which require a total memory consumption greater than the installed main memory capacity?

Terminology:

- A running program is called a process or a thread
- Operating System (OS) controls the processes

Virtual Memory

- Use main memory as a "cache" for secondary memory
- Each program is compiled into its own virtual address space
- What makes it work? Principle of Locality

Virtual Memory

- Use main memory as a "cache" for secondary memory
- Each program is compiled into its own virtual address space
- What makes it work? Principle of Locality

Why virtual memory?

- During run-time, virtual address is translated to a physical address
- Efficient & safe sharing memory among multiple programs
- Ability to run programs larger than the size of physical memory
- Code relocation: code can be loaded anywhere in main memory

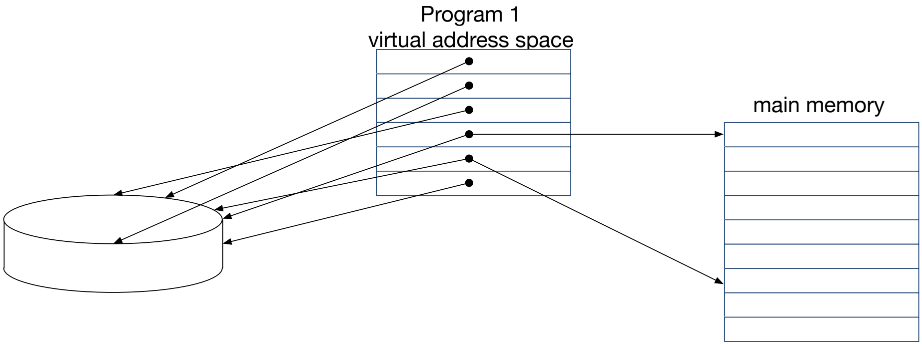
Bottom of the Memory Hierarchy

Consider the following example:

- Suppose we hit the limit of 1GB in the example, and we suddenly need some more memory on the fly.
- We move some main memory chunks to the harddisk, say, 100MB.
- So, we have 100MB of "free" main memory for use.
- What if later on, those instructions / data in the saved 100MB chunk are needed again?
- We have to "free" some other main memory chunks in order to move the instructions / data back from the harddisk.

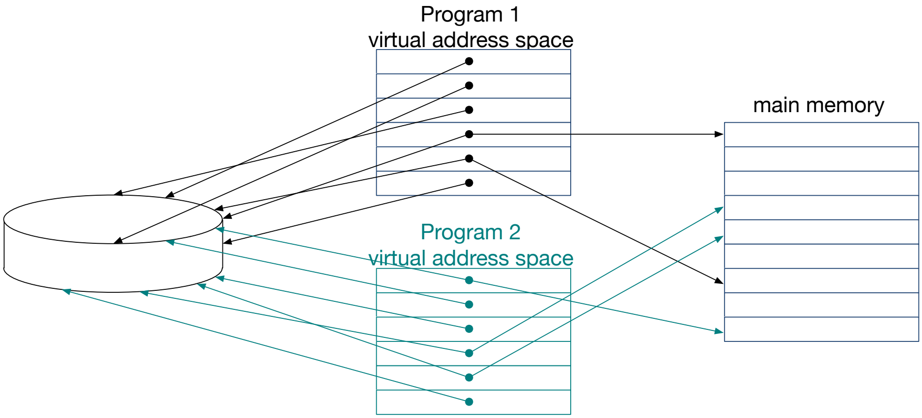
Two Programs Sharing Physical Memory

- A program's address space is divided into pages (fixed size) or segments (variable sizes)



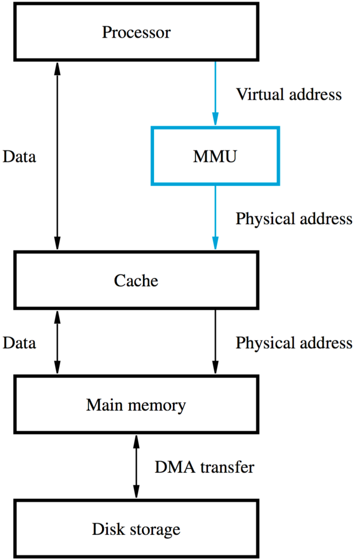
Two Programs Sharing Physical Memory

- A program's address space is divided into pages (fixed size) or segments (variable sizes)



Virtual Memory Organization

- Part of process(es) are stored temporarily on harddisk and brought into main memory as needed
- This is done automatically by the OS, application program does not need to be aware of the existence of virtual memory (VM)
- Memory management unit (MMU) translates virtual addresses to physical addresses



Overview

Introduction

Virtual Memory

- $\text{VA} \rightarrow \text{PA}$
- TLB

Performance Issues

Address Translation

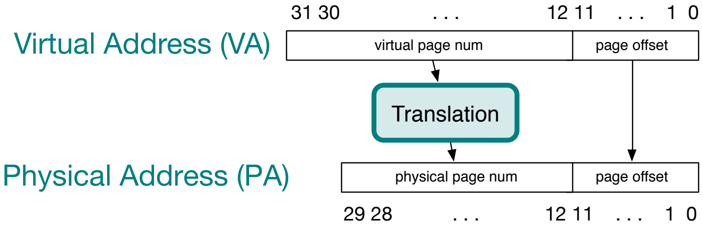
- Memory divided into pages of size ranging from 2KB to 16KB
  - Page too small: too much time spent getting pages from disk
  - Page too large: a large portion of the page may not be used
  - This is similar to cache block size issue (discussed earlier)
- For harddisk, it takes a considerable amount of time to locate a data on the disk but once located, the data can be transferred at a rate of several MB per second.
- If pages are too large, it is possible that a substantial portion of a page is not used but it will occupy valuable space in the main memory.

Address Translation

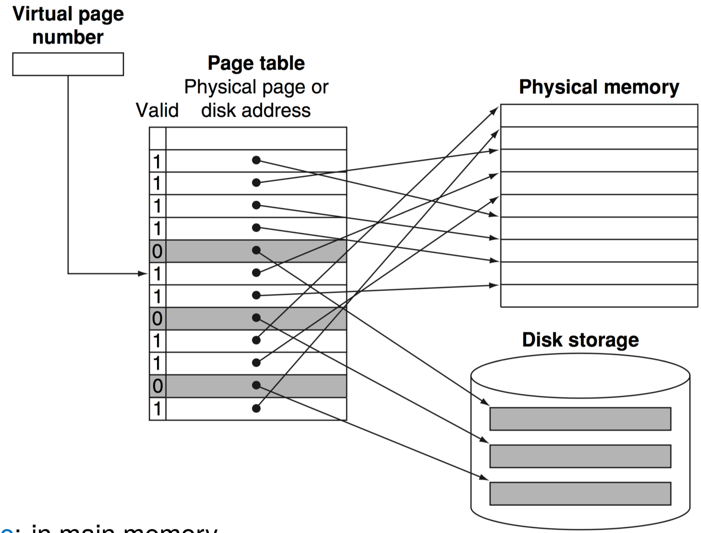
- An area in the main memory that can hold one page is called a page frame.
- Processor generates virtual addresses
  - MS (high order) bits are the virtual page number
  - LS (low order) bits are the offset
- Information about where each page is stored is maintained in a data structure in the main memory called the **page table**
  - Starting address of the page table is stored in a page table base register
  - Address in physical memory is obtained by indexing the virtual page number from the page table base register

Address Translation

- Virtual address $\rightarrow$ physical address by combination of HW/SW
- Each memory request needs first an address translation
- Page Fault: a virtual memory miss



Address Translation Mechanisms

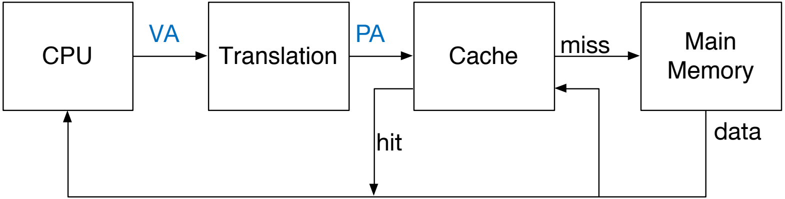


- Page Table: in main memory
- Process: page table + program counter + registers

Virtual Addressing with a Cache

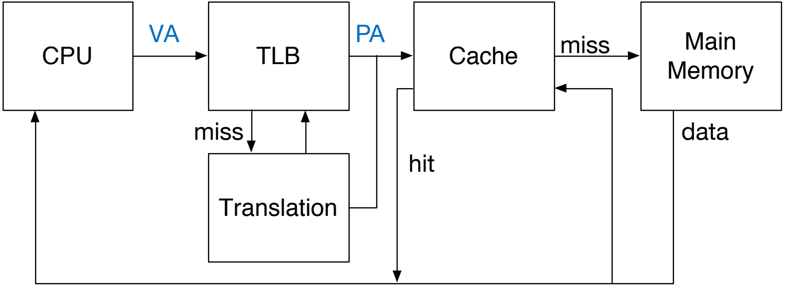
Disadvantage of virtual addressing:

- One extra memory access to translate a VA to a PA
- memory (cache) access very expensive...

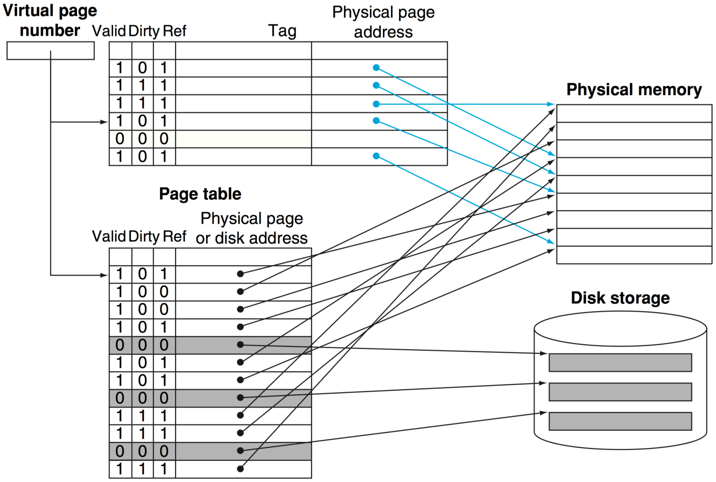


Translation Look-aside Buffer (TLB)

- A small cache: keeps track of recently used address mappings
- Avoid page table lookup



Translation Look-aside Buffer (TLB)



- Dirty bit:
- Ref bit:

More about TLB

Organization:

- Just like any other cache, can be fully associative, set associative, or direct mapped.

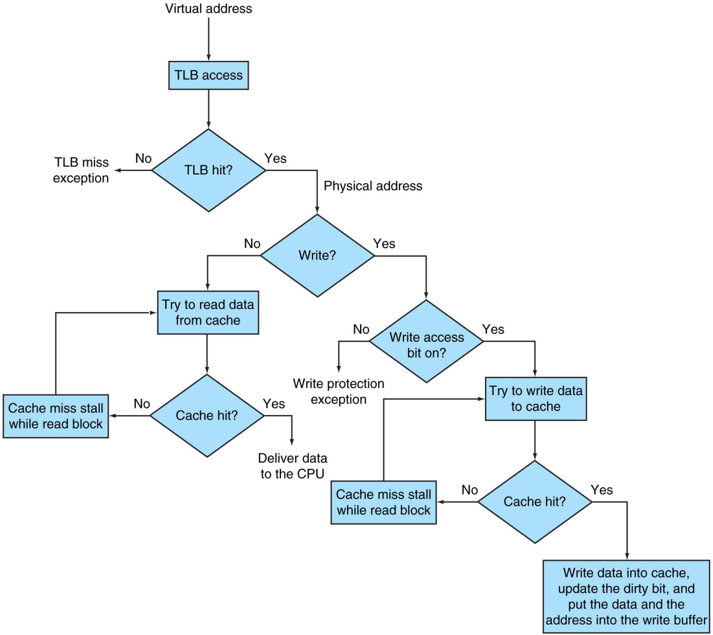
Access time:

- Faster than cache: due to smaller size
- Typically not more than 512 entries even on high end machines

A TLB miss:

- If the page is in main memory: miss can be handled; load translation info from page table to TLB
- If the page is NOT in main memory: page fault

Cooperation of TLB & Cache



TLB Event Combinations

- TLB / Cache miss: page / block not in "cache"
- Page Table miss: page NOT in memory

TLB	Page Table	Cache	Possible? Under what circumstances?
Hit	Hit	Hit	
Hit	Hit	Miss	
Miss	Hit	Hit	
Miss	Hit	Miss	
Miss	Miss	Miss	
Hit	Miss	Miss / Hit	
Miss	Miss	Hit	

TLB Event Combinations

- TLB / Cache miss: page / block not in "cache"
- Page Table miss: page NOT in memory

TLB	Page Table	Cache	Possible? Under what circumstances?
Hit	Hit	Hit	Yes – what we want!
Hit	Hit	Miss	Yes – although page table is not checked if TLB hits
Miss	Hit	Hit	Yes – TLB miss, PA in page table
Miss	Hit	Miss	Yes – TLB miss, PA in page table but data not in cache
Miss	Miss	Miss	Yes – page fault
Hit	Miss	Miss / Hit	
Miss	Miss	Hit	

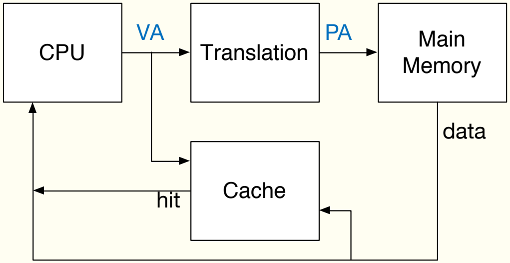
TLB Event Combinations

- TLB / Cache miss: page / block not in "cache"
- Page Table miss: page NOT in memory

TLB	Page Table	Cache	Possible? Under what circumstances?
Hit	Hit	Hit	Yes – what we want!
Hit	Hit	Miss	Yes – although page table is not checked if TLB hits
Miss	Hit	Hit	Yes – TLB miss, PA in page table
Miss	Hit	Miss	Yes – TLB miss, PA in page table but data not in cache
Miss	Miss	Miss	Yes – page fault
Hit	Miss	Miss / Hit	Impossible – TLB translation not possible if page is not in memory
Miss	Miss	Hit	Impossible – data not allowd in cache if page is not in memory

QUESTION: Why Not a Virtually Addressed Cache?

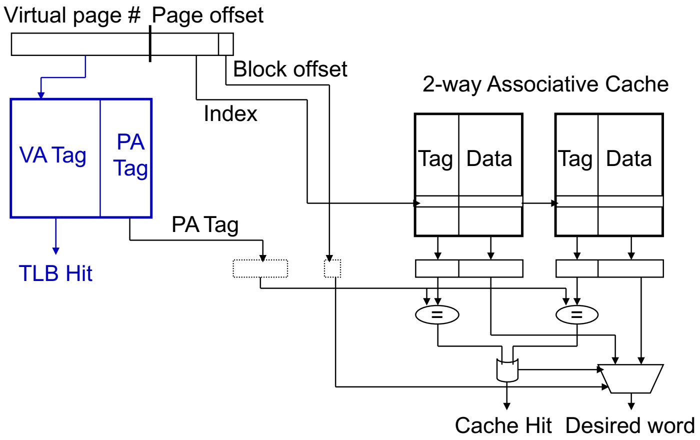
- Access Cache using virtual address (VA)
- Only address translation when cache misses



Answer:

Overlap Cache & TLB Accesses

- High order bits of VA are used to access TLB
- Low order bits of VA are used as index into cache



The Hardware / Software Boundary

Which part of address translation is done by hardware?

- TLB that caches recent translations:
  - TLB access time is part of cache hit time
  - May allot extra stage in pipeline
- Page Table storage, fault detection and updating
  - Dirty & Reference bits
  - Page faults result in interrupts
- Disk Placement:

The Hardware / Software Boundary

Which part of address translation is done by hardware?

- TLB that caches recent translations: (Hardware)
  - TLB access time is part of cache hit time
  - May allot extra stage in pipeline
- Page Table storage, fault detection and updating
  - Dirty & Reference bits (Hardware)
  - Page faults result in interrupts (Software)
- Disk Placement: (Software)

Overview

Introduction

Virtual Memory

- VA $\rightarrow$ PA
- TLB

Performance Issues

Q1: Where A Block Be Placed in Upper Level?

Scheme name	# of sets	Blocks per set
Direct mapped	# of blocks	1
Set associative	$\frac{\text{\# of blocks}}{\text{Associativity}}$	Associativity
Fully associative	1	# of blocks

Q1: Where A Block Be Placed in Upper Level?

Scheme name	# of sets	Blocks per set
Direct mapped	# of blocks	1
Set associative	$\frac{\text{\# of blocks}}{\text{Associativity}}$	Associativity
Fully associative	1	# of blocks

Q2: How Is Entry Be Found?

Scheme name	Location method	# of comparisons
Direct mapped	Index	1
Set associative	Index the set; compare set's tags	Degree of associativity
Fully associative	Compare all tags	# of blocks

Q3: Which Entry Should Be Replaced on a Miss?

- Direct mapped: only one choice
- Set associative or fully associative:
  - Random
  - LRU (Least Recently Used)

Note that:

- For a 2-way set associative, random replacement has a miss rate $1.1\times$ than LRU
- For high level associativity (4-way), LRU is too costly

Q4: What Happen On A Write?

- Write-Through:
  - The information is written in both the block in cache & the block in lower level of memory
  - Combined with write buffer, so write waits can be eliminated
  - ⊕:
  - ⊕:

- Write-Back:
  - The information is written only to the block in cache
  - The modification is written to lower level, only when the block is replaced
  - Need dirty bit: tracks whether the block is clean or not
  - Virtual memory always use write-back
  - ⊕:
  - ⊕:

Q4: What Happen On A Write?

- Write-Through:
  - The information is written in both the block in cache & the block in lower level of memory
  - Combined with write buffer, so write waits can be eliminated
  - ⊕: read misses don't result in writes
  - ⊕: easier to implement

- Write-Back:
  - The information is written only to the block in cache
  - The modification is written to lower level, only when the block is replaced
  - Need dirty bit: tracks whether the block is clean or not
  - Virtual memory always use write-back
  - ⊕:
  - ⊕:

Q4: What Happen On A Write?

- Write-Through:
  - The information is written in both the block in cache & the block in lower level of memory
  - Combined with write buffer, so write waits can be eliminated
  - ⊕: read misses don't result in writes
  - ⊕: easier to implement

- Write-Back:
  - The information is written only to the block in cache
  - The modification is written to lower level, only when the block is replaced
  - Need dirty bit: tracks whether the block is clean or not
  - Virtual memory always use write-back
  - ⊕: write with speed of cache
  - ⊕: repeated writes require only one write to lower level

Performance Consideration

Performance

How **fast** machine instructions can be brought into the processor and how **fast** they can be executed.

- Two key factors are performance and cost, i.e., price/performance ratio.
- For a hierarchical memory system with cache, the processor is able to access instructions and data more quickly when the data wanted are in the cache.
- Therefore, the impact of a cache on performance is dependent on the hit and miss rates.

Cache Hit Rate and Miss Penalty

- High hit rates over 0.9 are essential for high-performance computers.
- A penalty is incurred because extra time is needed to bring a block of data from a slower unit to a faster one in the hierarchy.
- During that time, the processor is stalled.
- The waiting time depends on the details of the cache operation.

Miss Penalty

Total access time seen by the processor when a miss occurs.

Miss Penalty

Example: Consider a computer with the following parameters:

Access times to the cache and the main memory are t and 10t respectively. When a cache miss occurs, a block of 8 words will be transferred from the MM to the cache. It takes 10t to transfer the first word of the block and the remaining 7 words are transferred at a rate of one word per t seconds.

- Miss penalty = $t + 10t + 7 \times t + t$
- First t: Initial cache access that results in a miss.
- Last t: Move data from the cache to the processor.

Average Memory Access Time

$$h \times C + (1 - h) \times M$$

- h: hit rate
- M: miss penalty
- C: cache access time

- High cache hit rates ($> 90\%$) are essential
- Miss penalty must also be reduced

Question: Memory Access Time Example

- Assume 8 cycles to read a single memory word;
- 15 cycles to load a 8-word block from main memory (previous example);
- cache access time = 1 cycle
- For every **100** instructions, statistically **30** instructions are data read/ write
- Instruction fetch: 100 memory access: assume hit rate = 0.95
- Data read/ write: 30 memory access: assume hit rate = 0.90

Calculate: (1) Execution cycles without cache; (2) Execution cycles with cache.

Caches on Processor Chips

- In high-performance processors, two levels of caches are normally used, L1 and L2.
- L1 must be very fast as they determine the memory access time seen by the processor.
- L2 cache can be slower, but it should be much larger than the L1 cache to ensure a high hit rate. Its speed is less critical because it only affects the miss penalty of the L1 cache.
- Average access time on such a system:

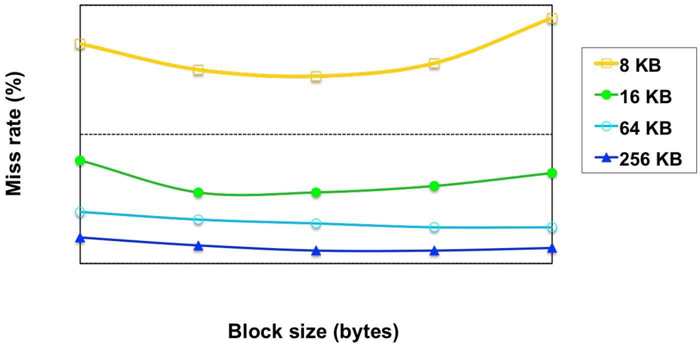
$$h_1 \cdot C_1 + (1 - h_1) \cdot [h_2 \cdot C_2 + (1 - h_2) \cdot M]$$

- h_1 (h_2): the L1 (L2) hit rate
- C_1 the access time of L1 cache,
- C_2 the miss penalty to transfer data from L2 cache to L1
- M: the miss penalty to transfer data from MM to L2 and then to L1.

Larger Block Size

- Take advantage of spatial locality.
- ☺ If all items in a larger block are needed in a computation, it is better to load these items into the cache in a single miss.
- ☹ Larger blocks are effective only up to a certain size, beyond which too many items will remain unused before the block is replaced.
- ☹ Larger blocks take longer time to transfer and thus increase the miss penalty.
- Block sizes of 16 to 128 bytes are most popular.

Miss Rate v.s. Block Size v.s. Cache Size



Miss rate goes up if the block size becomes a significant fraction of the cache size because the number of blocks that can be held in the same size cache is smaller (increasing capacity misses)

Enhancement

Write buffer:

- Read request is served first.
- Write request stored in write buffer first and sent to memory whenever there is no read request.
- The addresses of a read request should be compared with the addresses of the write buffer.

Prefetch:

- Prefetch data into the cache before they are needed, while the processor is busy executing instructions that do not result in a read miss.
- Prefetch instructions can be inserted by the programmer or the compiler.

Load-through Approach

- Instead of waiting the whole block to be transferred, the processor resumes execution as soon as the required word is loaded in the cache.