

CENG3420 Computer Organization & Design

Lecture 09: Virtual Memory

Bei Yu

Spring 2016

byu@cse.cuhk.edu.hk



香港中文大學

The Chinese University of Hong Kong

Overview

Review: Memory Hierarchy

Virtual Memory

1. VA \rightarrow PA
2. TLB

Questions for Memory Hierarchy

Secondary Memory

Overview

Review: Memory Hierarchy

Virtual Memory

1. VA \rightarrow PA
2. TLB

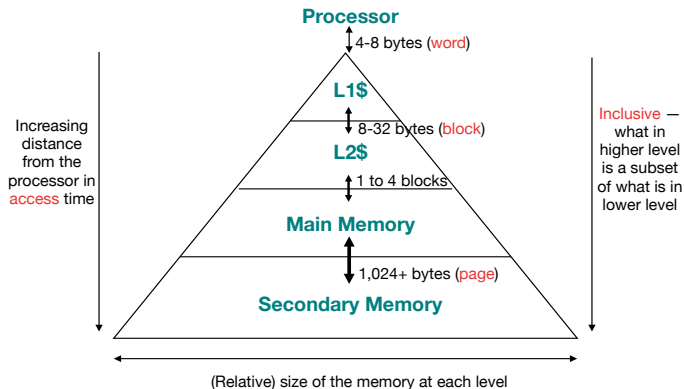
Questions for Memory Hierarchy

Secondary Memory

Review: Memory Hierarchy

Take advantage of **principle of locality**, present the user:

- ▶ as much memory as is available
- ▶ cheapest technology
- ▶ at the speed offered by the fastest technology



Review: Reducing Cache Miss Rates #1

Direct mapped cache:

- ▶ a memory block maps to **exactly one** cache block

Fully associative cache:

- ▶ a memory block maps to **any** cache block

Review: Reducing Cache Miss Rates #1

Direct mapped cache:

- ▶ a memory block maps to **exactly one** cache block

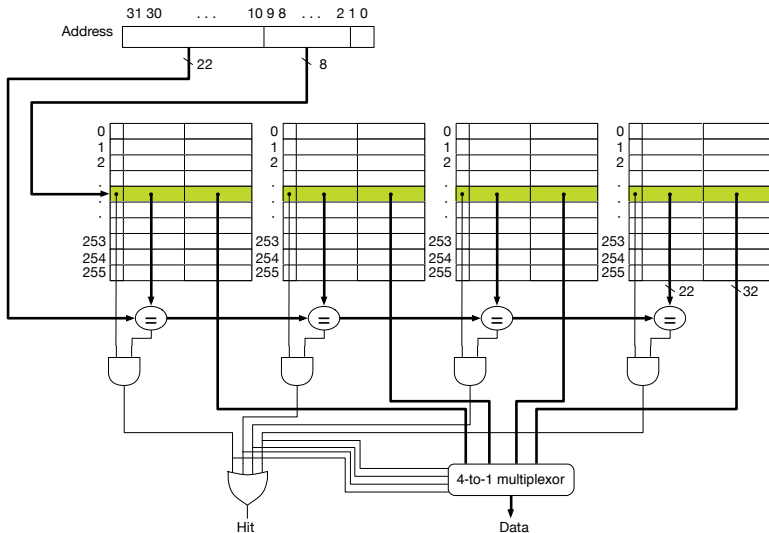
Fully associative cache:

- ▶ a memory block maps to **any** cache block

N-Way Set Associative Cache:

- ▶ A compromise is to divide the cache into **sets**
- ▶ `index` field maps a memory block to a **unique** set
- ▶ can be placed in any way of that set

Review: 4-Way Set Associative Cache



- ▶ $2^8 = 256$ sets each with four ways (each with one block)

Overview

Review: Memory Hierarchy

Virtual Memory

1. VA \rightarrow PA
2. TLB

Questions for Memory Hierarchy

Secondary Memory

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

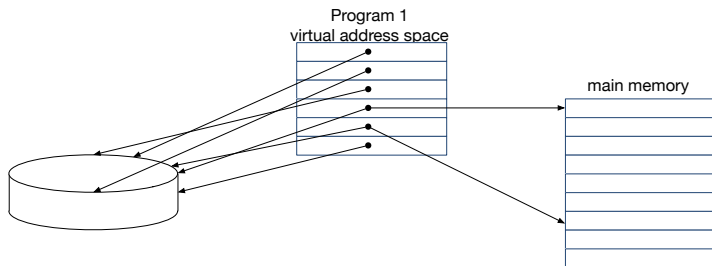
- ▶ Use main memory as a “cache” for secondary memory
- ▶ Each program is compiled into its own **virtual** address space
- ▶ What makes it work? [Principle of Locality](#)

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

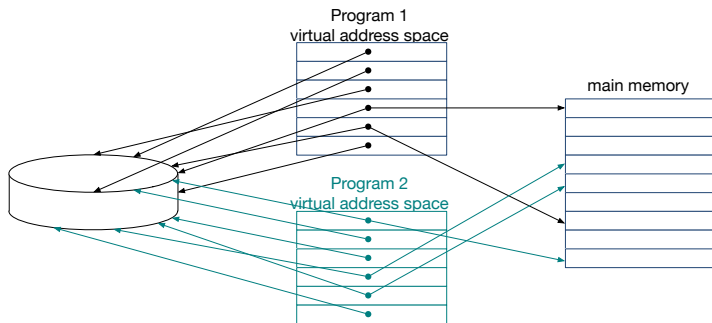
Two Programs Sharing Physical Memory

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



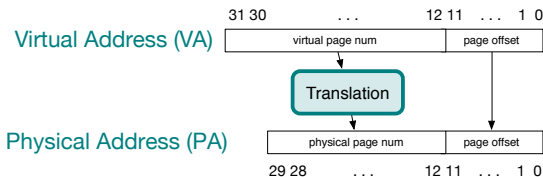
Two Programs Sharing Physical Memory

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

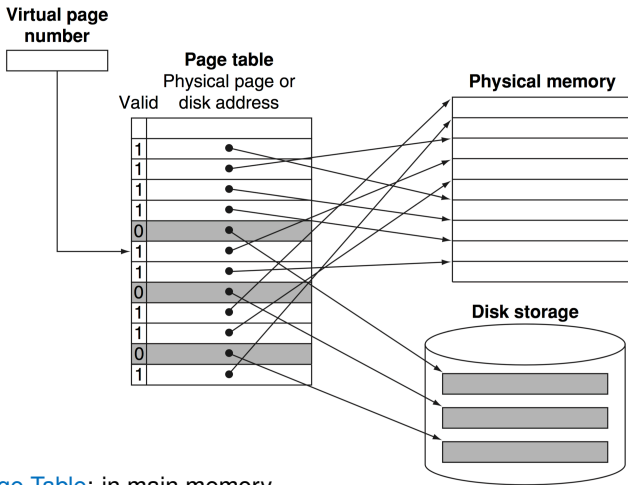


Address Translation

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



Address Translation Mechanisms

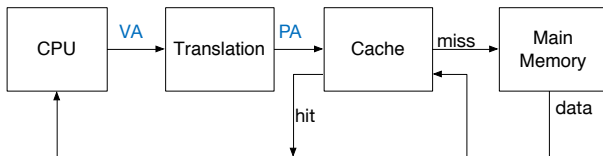


- ▶ **Page Table**: in main memory
- ▶ **Process**: page table + program counter + registers

Virtual Addressing with a Cache

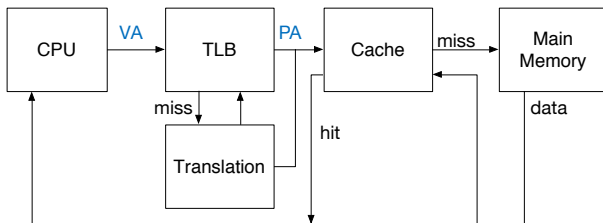
Disadvantage of virtual addressing:

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

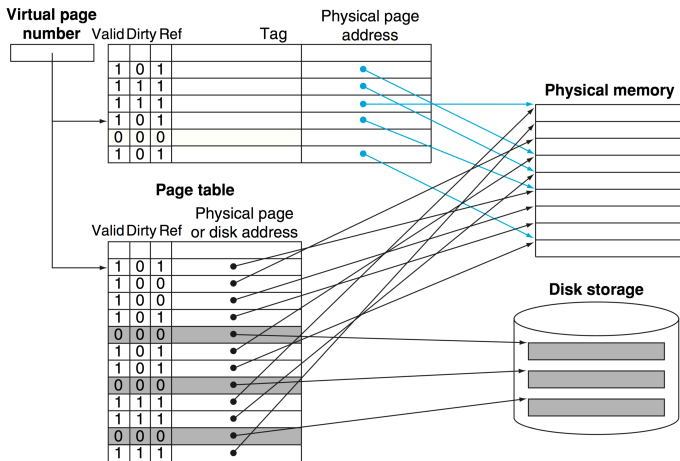


Translation Look-aside Buffer (TLB)

- ▶ 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.

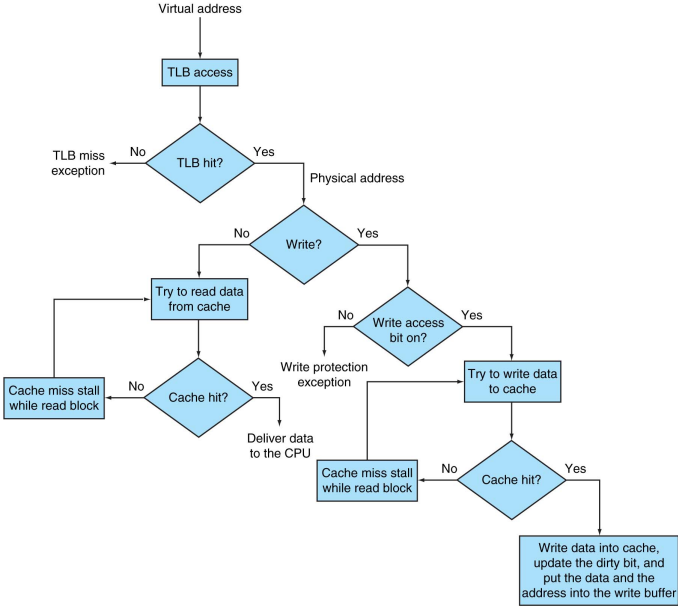
Access time:

- ▶ **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	

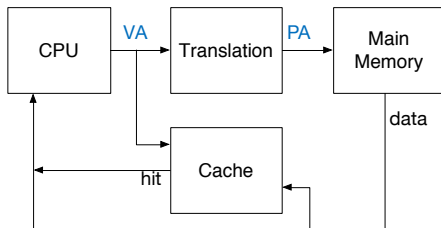
TLB Event Combinations

- ▶ 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 allowed in cache if page is not in memory

Question: Why Not a Virtually Addressed Cache?

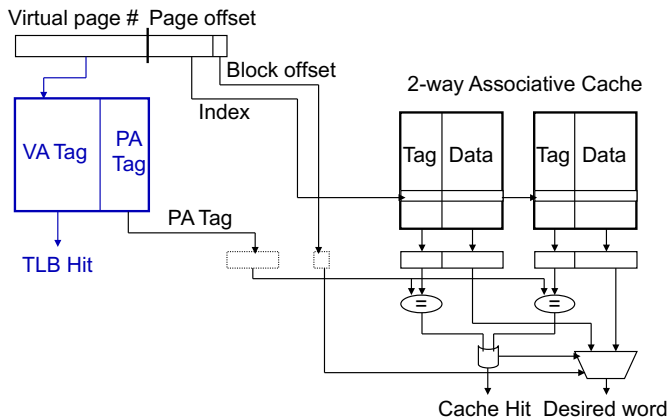
- ▶ Access Cache using virtual address (VA)
- ▶ Only address translation when cache misses



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

Review: Memory Hierarchy

Virtual Memory

1. VA \rightarrow PA
2. TLB

Questions for Memory Hierarchy

Secondary Memory

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
	Separate page tables	0

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

Overview

Review: Memory Hierarchy

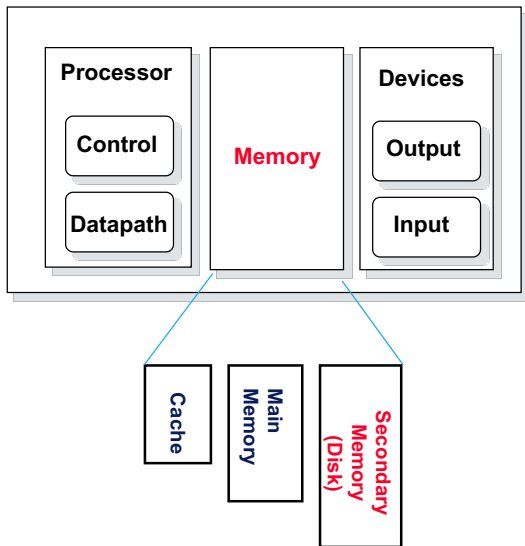
Virtual Memory

1. VA \rightarrow PA
2. TLB

Questions for Memory Hierarchy

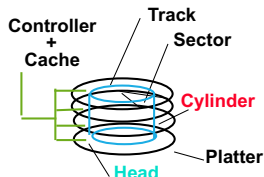
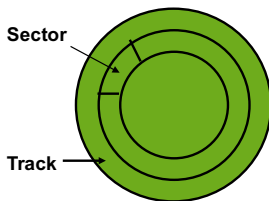
Secondary Memory

Major Components of A Computer



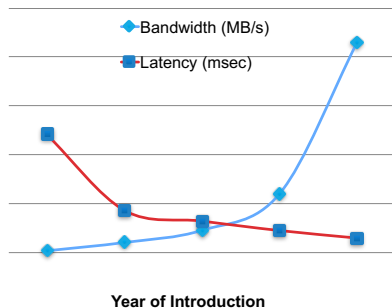
Magnetic Disk

- ▶ Long term, **nonvolatile** storage
- ▶ Lowest level memory: slow; large; inexpensive
- ▶ A rotating platter coated with a magnetic surface
- ▶ A moveable read/write head to access the information



Magnetic Disk (cont.)

- ▶ **Latency**: average seek time plus the rotational latency
- ▶ **Bandwidth**: peak transfer time of formatted data from the media (not from the cache)



- ▶ In the time the bandwidth doubles, latency improves by a factor of only around 1.2

Flash Storage

- ▶ First credible challenger to disks
- ▶ Nonvolatile, and $100 \times - 1000 \times$ faster than disks
- ▶ **Wear leveling** to overcome **wear out** problem

