dce 2011

ADVANCED COMPUTER ARCHITECTURE

Khoa Khoa học và Kỹ thuật Máy tính
BM Kỹ thuật Máy tính

Trần Ngọc Thịnh
http://www.cse.hcmut.edu.vn/~tnthinh

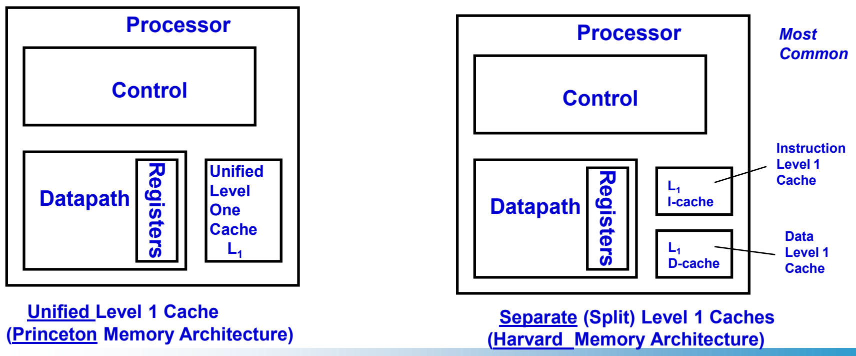
dce 2011

Memory Hierarchy Design (part2)

2

Unified vs. Separate Level 1 Cache

- Unified Level 1 Cache (Princeton Memory Architecture).
  A single level 1 (L_1) cache is used for both instructions and data.
- Separate instruction/data Level 1 caches (Harvard Memory Architecture):
  The level 1 (L_1) cache is split into two caches, one for instructions (instruction cache, L_1 I-cache) and the other for data (data cache, L_1 D-cache).


Processor
Control
Datapath
Registers
Unified Level One Cache L1
Unified Level 1 Cache
(Princeton Memory Architecture)
Processor
Most Common
Control
Datapath
Registers
L1 I-cache
L1 D-cache
Instruction Level 1 Cache
Data Level 1 Cache
Separate (Split) Level 1 Caches
(Harvard Memory Architecture)

Memory Hierarchy Performance (1/2)

- The Average Memory Access Time (AMAT): The number of cycles required to complete an average memory access request by the CPU.
- Memory stall cycles per memory access: The number of stall cycles added to CPU execution cycles for one memory access.

$$\boxed{\text{Memory stall cycles per average memory access} = (\text{AMAT} - 1)}$$

- For ideal memory: AMAT = 1 cycle, this results in zero memory stall cycles.

Memory Hierarchy Performance (2/2)

- Memory stall cycles per average instruction =
  Number of memory accesses per instruction
  x Memory stall cycles per average memory access
  = (1 + fraction of loads/stores) x (AMAT -1)

Instruction Fetch

Base CPI = $CPI_{execution}$ = CPI with ideal memory

$$\boxed{CPI = CPI_{execution} + \text{Mem Stall cycles per instruction}}$$

Cache Performance:Single Level L1 Princeton (Unified) Memory Architecture (1/2)

CPUtime = Instruction count x CPI x Clock cycle time

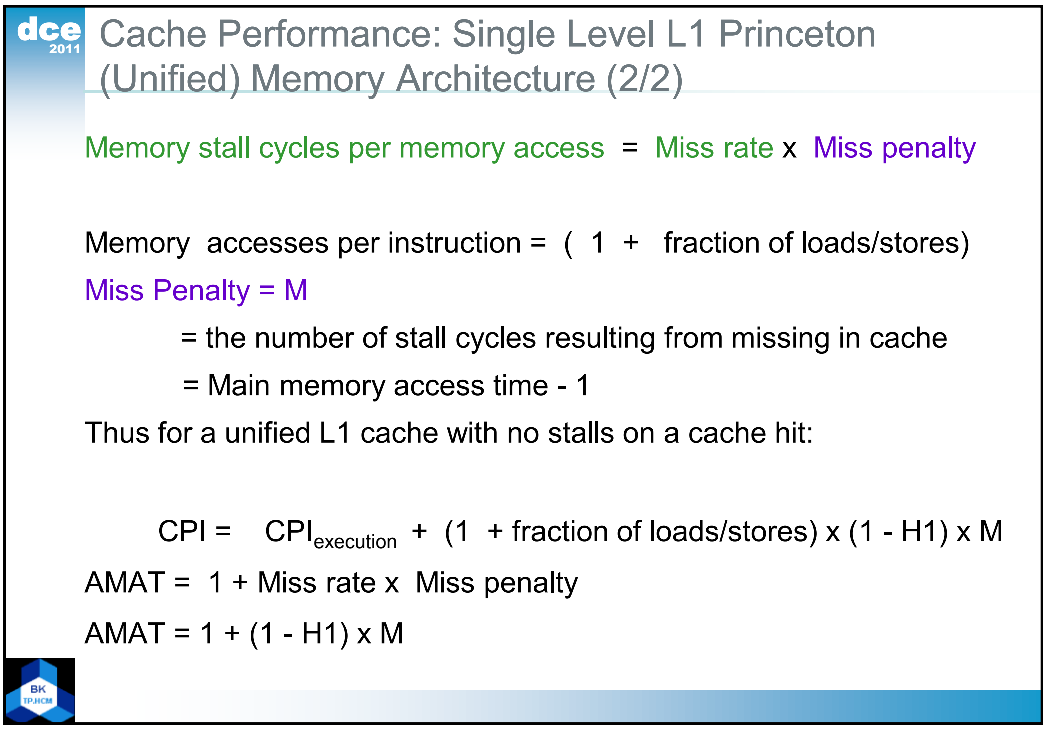
$CPI_{execution}$ = CPI with ideal memory

$$\boxed{CPI = CPI_{execution} + \text{Mem Stall cycles per instruction}}$$

Mem Stall cycles per instruction =
Memory accesses per instruction x Memory stall cycles per access

Assuming no stall cycles on a cache hit (cache access time = 1 cycle, stall = 0)

Cache Hit Rate = H1   Miss Rate = 1- H1



Cache Performance: Single Level L1 Princeton (Unified) Memory Architecture (2/2)

Memory stall cycles per memory access = Miss rate x Miss penalty

Memory accesses per instruction = (1 + fraction of loads/stores)

Miss Penalty = M

= the number of stall cycles resulting from missing in cache

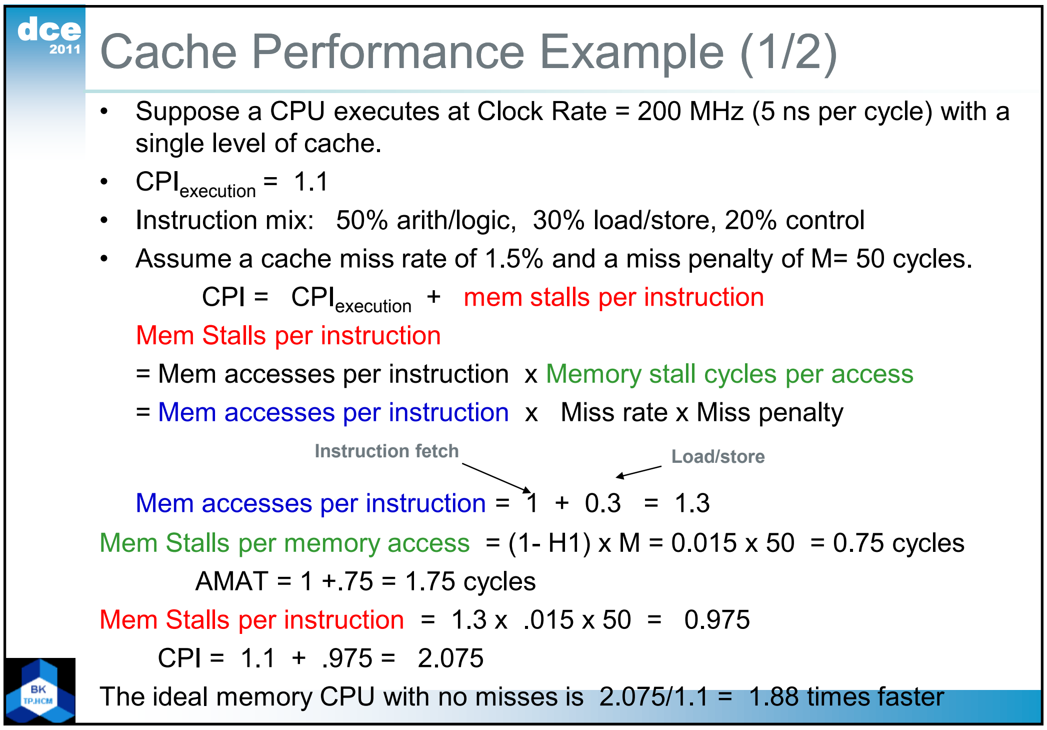
= Main memory access time - 1

Thus for a unified L1 cache with no stalls on a cache hit:

$$CPI = CPI_{execution} + (1 + \text{fraction of loads/stores}) \times (1 - H1) \times M$$

AMAT = 1 + Miss rate x Miss penalty

AMAT = 1 + (1 - H1) x M



Cache Performance Example (1/2)

- Suppose a CPU executes at Clock Rate = 200 MHz (5 ns per cycle) with a single level of cache.
- $CPI_{execution}$ = 1.1
- Instruction mix: 50% arith/logic, 30% load/store, 20% control
- Assume a cache miss rate of 1.5% and a miss penalty of M= 50 cycles.

$CPI = CPI_{execution}$ + mem stalls per instruction

Mem Stalls per instruction

= Mem accesses per instruction x Memory stall cycles per access

= Mem accesses per instruction x Miss rate x Miss penalty

Instruction fetch Load/store

Mem accesses per instruction = 1 + 0.3 = 1.3

Mem Stalls per memory access = (1- H1) x M = 0.015 x 50 = 0.75 cycles

AMAT = 1 +.75 = 1.75 cycles

Mem Stalls per instruction = 1.3 x .015 x 50 = 0.975

CPI = 1.1 + .975 = 2.075

The ideal memory CPU with no misses is 2.075/1.1 = 1.88 times faster

Cache Performance Example (2/2)

- Suppose for the previous example we double the clock rate to 400 MHz, how much faster is this machine, assuming similar miss rate, instruction mix?
- Since memory speed is not changed, the miss penalty takes more CPU cycles:

Miss penalty = M = 50 x 2 = 100 cycles.

CPI = 1.1 + 1.3 x .015 x 100 = 1.1 + 1.95 = 3.05

Speedup = $(CPI_{old} \times C_{old}) / (CPI_{new} \times C_{new})$

= 2.075 x 2 / 3.05 = 1.36

- The new machine is only 1.36 times faster rather than 2 times faster due to the increased effect of cache misses.

→ *CPUs with higher clock rate, have more cycles per cache miss and more memory impact on CPI.*

Cache Performance Harvard Memory Architecture

For a CPU with separate or split level one (L1) caches for instructions and data (Harvard memory architecture) and no stalls for cache hits:

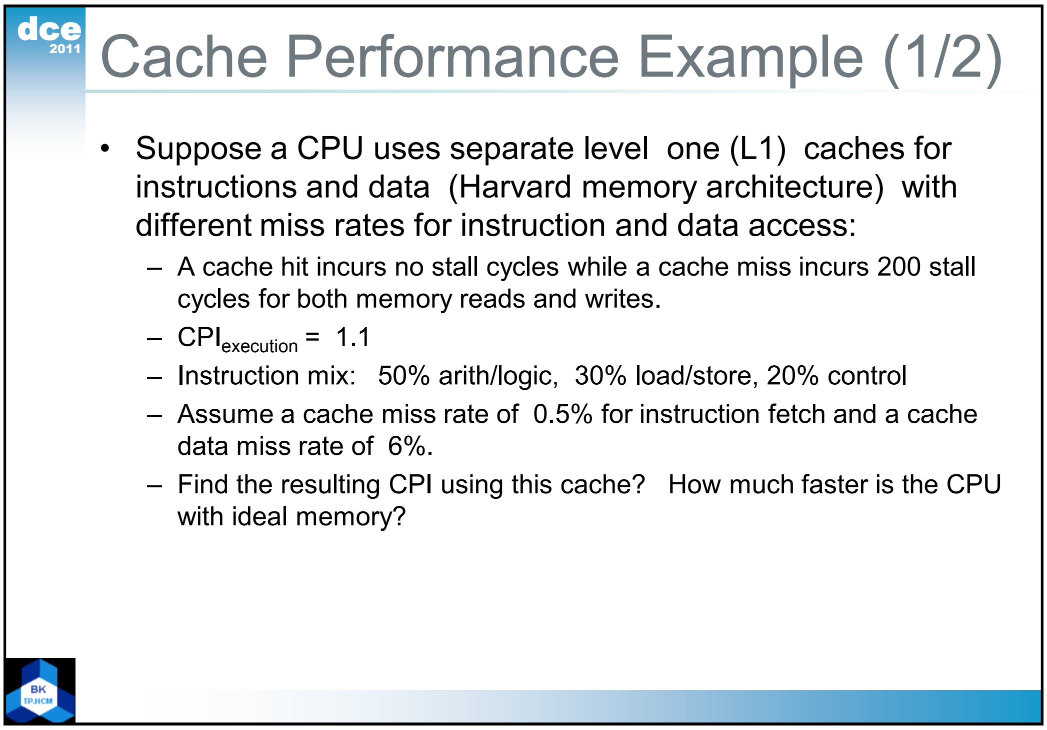
CPUtime = Instruction count x CPI x Clock cycle time

$\boxed{CPI = CPI_{execution} + \text{Mem Stall cycles per instruction}}$

Mem Stall cycles per instruction =

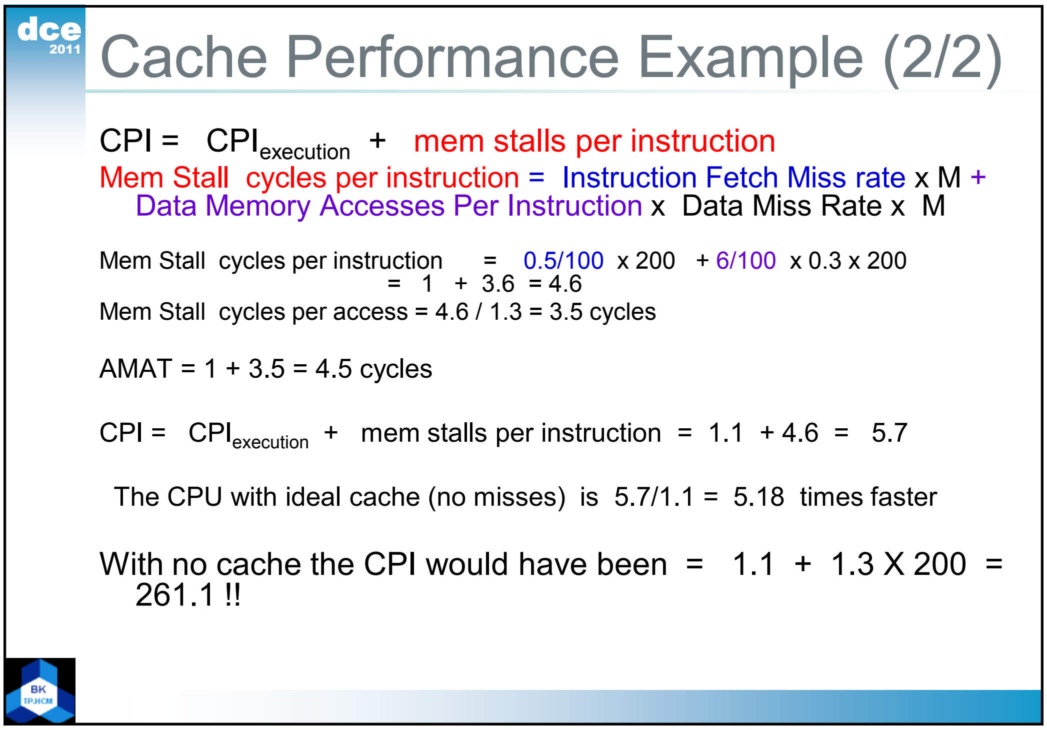
Instruction Fetch Miss rate x M +

Data Memory Accesses Per Instruction x Data Miss Rate x M



Cache Performance Example (1/2)

- Suppose a CPU uses separate level one (L1) caches for instructions and data (Harvard memory architecture) with different miss rates for instruction and data access:
  - A cache hit incurs no stall cycles while a cache miss incurs 200 stall cycles for both memory reads and writes.
  - $CPI_{execution} = 1.1$
  - Instruction mix: 50% arith/logic, 30% load/store, 20% control
  - Assume a cache miss rate of 0.5% for instruction fetch and a cache data miss rate of 6%.
  - Find the resulting CPI using this cache? How much faster is the CPU with ideal memory?



Cache Performance Example (2/2)

CPI = $CPI_{execution}$ + mem stalls per instruction

Mem Stall cycles per instruction = Instruction Fetch Miss rate x M + Data Memory Accesses Per Instruction x Data Miss Rate x M

Mem Stall cycles per instruction = 0.5/100 x 200 + 6/100 x 0.3 x 200
= 1 + 3.6 = 4.6

Mem Stall cycles per access = 4.6 / 1.3 = 3.5 cycles

AMAT = 1 + 3.5 = 4.5 cycles

CPI = $CPI_{execution}$ + mem stalls per instruction = 1.1 + 4.6 = 5.7

The CPU with ideal cache (no misses) is 5.7/1.1 = 5.18 times faster

With no cache the CPI would have been = 1.1 + 1.3 X 200 = 261.1 !!

Virtual Memory

- Some facts of computer life…
  - Computers run lots of processes simultaneously
  - No full address space of memory for each process
  - Must share smaller amounts of physical memory among many processes
- Virtual memory is the answer!
  - Divides physical memory into blocks, assigns them to different processes

Virtual Memory

- Virtual memory (VM) allows main memory (DRAM) to act like a cache for secondary storage (magnetic disk).
- VM address translation a provides a mapping from the virtual address of the processor to the physical address in main memory or on disk.

Compiler assigns data to a “virtual” address.
VA translated to a real/physical somewhere in memory…

(allows any program to run anywhere;
where is determined by a particular machine, OS)

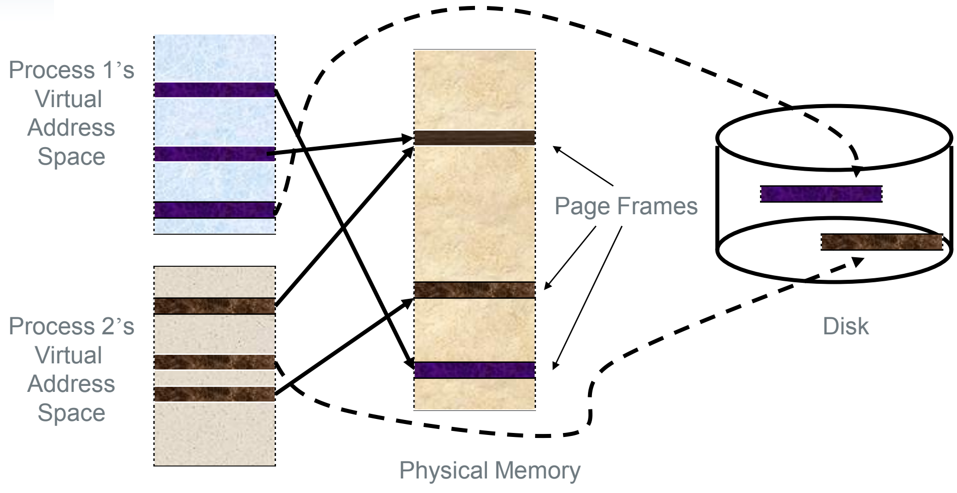
VM Benefit

- VM provides the following benefits
  - Allows multiple programs to share the same physical memory
  - Allows programmers to write code as though they have a very large amount of main memory
  - Automatically handles bringing in data from disk

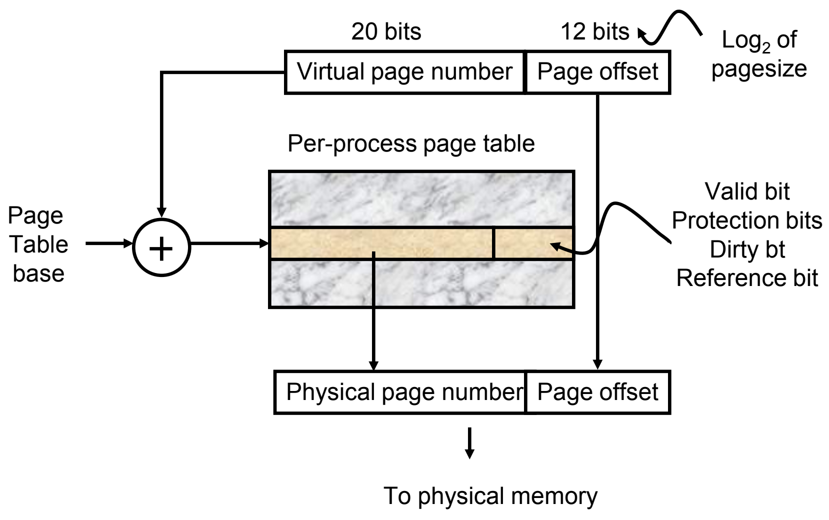
Virtual Memory Basics

- Programs reference "virtual" addresses in a non-existent memory
  - These are then translated into real "physical" addresses
  - Virtual address space may be bigger than physical address space
- Divide physical memory into blocks, called pages
  - Anywhere from 512 to 16MB (4k typical)
- Virtual-to-physical translation by indexed table lookup
  - Add another cache for recent translations (the TLB)
- Invisible to the programmer
  - Looks to your application like you have a lot of memory!
  - Anyone remember overlays?

VM: Page Mapping

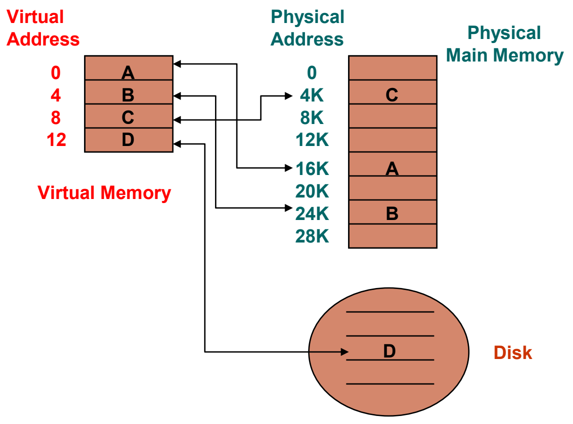

Process 1's Virtual Address Space
Process 2's Virtual Address Space
Page Frames
Disk
Physical Memory

VM: Address Translation



Example of virtual memory

- Relieves problem of making a program that was too large to fit in physical memory – well….fit!
- Allows program to run in any location in physical memory
  - (called relocation)
  - Really useful as you might want to run same program on lots machines…



Logical program is in contiguous VA space; here, consists of 4 pages: A, B, C, D;
The physical location of the 3 pages – 3 are in main memory and 1 is located on the disk

Cache terms vs. VM terms

So, some definitions/"analogies"

- A "*page*" or "*segment*" of memory is analogous to a "block" in a cache
- A "*page fault*" or "*address fault*" is analogous to a cache miss

so, if we go to main memory and our data isn't there, we need to get it from disk…

"real"/physical memory

More definitions and cache comparisons

- These are more definitions than analogies…
  - With VM, CPU produces "*virtual addresses*" that are translated by a combination of HW/SW to "*physical addresses*"
  - The "*physical addresses*" access main memory
    - The process described above is called "*memory mapping*" or "*address translation*"

Cache VS. VM comparisons (1/2)

Parameter	First-level cache	Virtual memory
Block (page) size	12-128 bytes	4096-65,536 bytes
Hit time	1-2 clock cycles	40-100 clock cycles
Miss penalty (Access time) (Transfer time)	8-100 clock cycles (6-60 clock cycles) (2-40 clock cycles)	700,000 – 6,000,000 clock cycles (500,000 – 4,000,000 clock cycles) (200,000 – 2,000,000 clock cycles)
Miss rate	0.5 – 10%	0.00001 – 0.001%
Data memory size	0.016 – 1 MB	4MB – 4GB

- It's a lot like what happens in a cache
  - But everything (except miss rate) is a LOT worse

Cache VS. VM comparisons (2/2)

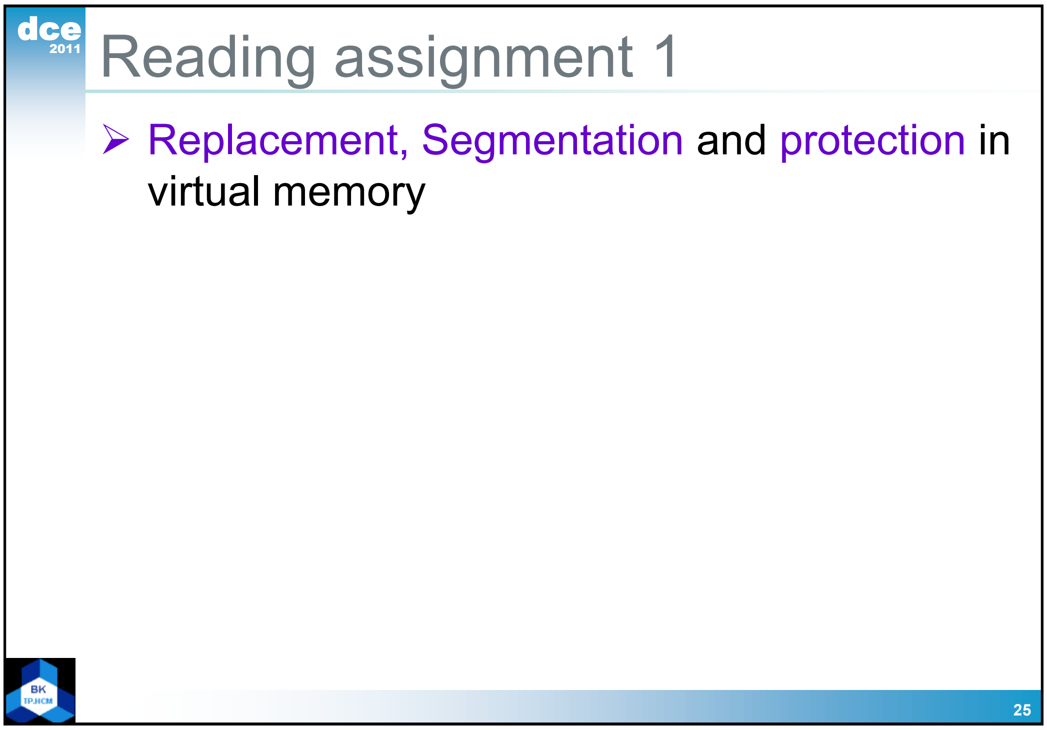
- Replacement policy:
  - Replacement on cache misses primarily controlled by hardware
  - Replacement with VM (i.e. which page do I replace?) usually controlled by OS
    - Because of bigger miss penalty, want to make the right choice
- Sizes:
  - Size of processor address determines size of VM
  - Cache size independent of processor address size

Virtual Memory

- Timing's tough with virtual memory:

  - AMAT = T_{mem} + (1-h) * T_{disk}
  - = 100nS + (1-h) * 25,000,000nS

- h (hit rate) had to be *incredibly* (almost unattainably) close to perfect to work



Reading assignment 1

- Replacement, Segmentation and protection in virtual memory

25