

# Lecture 5

## Memory Hierarchies

- **Cache memory (Ch. 4.3)**
  - ✓ Classification of cache misses (Ch 4.3.5)
  - ✓ Cache hierarchy performance (Ch 4.3.4)
  - ✓ Memory inclusion (Ch. 4.3.3)
  - ✓ Non-blocking (Lock-up free) caches (Ch. 4.3.6)
  - ✓ Cache prefetching and preloading (Ch. 4.3.7)
- **Virtual memory (Ch. 4.4)**

# Cache Miss Classification

## (Ch 4.3.5)

# Classification of Cache Misses

## The 3 C's miss classification model

- **Compulsory (Cold) misses**: On the first reference to a block
- **Capacity misses**: Space is not sufficient to host data or code
- **Conflict misses**: Happen when two memory blocks map the same cache block in direct-mapped or set-associative caches
- (Related to communication: **Coherence misses**, the 4<sup>th</sup> C.)

## How to find out?

- **Cold misses**: Simulate infinite cache size
- **Capacity misses**: Simulate fully associative cache, then deduct cold misses
- **Conflict misses**: Simulate cache, then deduct cold and capacity misses

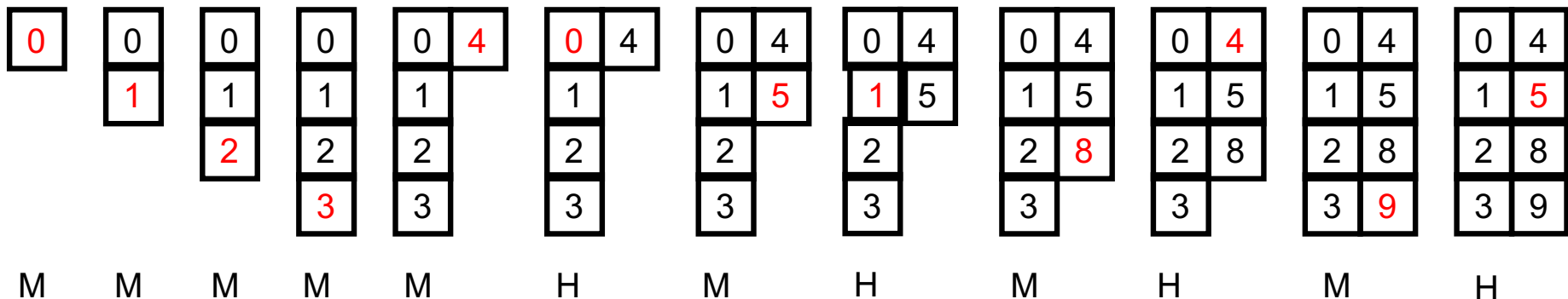
**Classification is useful to understand how to eliminate misses**

## Assumptions:

# Example

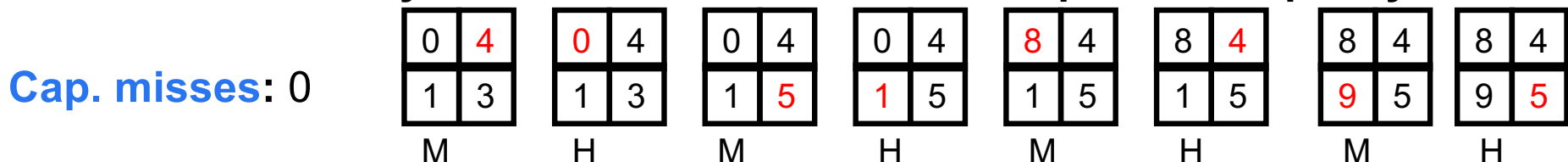
- A 2-way set-associative cache with 4 blocks
- Access sequence:** 0 1 2 3 4 0 5 1 8 4 9 5

## Simulation of infinite cache:



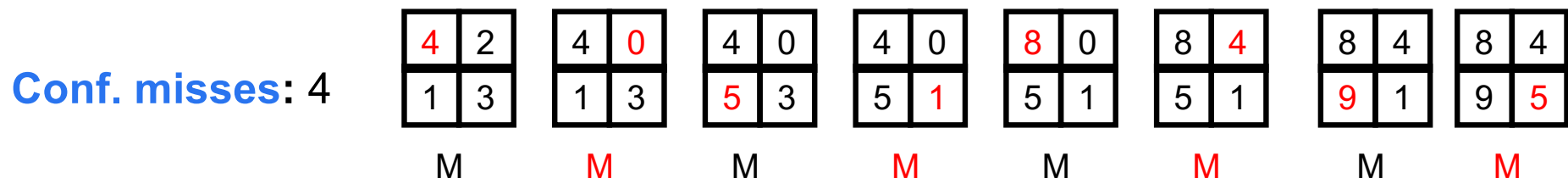
**Cold misses:** 8

## Simulation of a fully associative cache with OPT replacement policy:



**Cap. misses:** 0

## Simulation of a 2-way set-associative cache with LRU replacement policy:



**Conf. misses:** 4

## Quiz 5.1

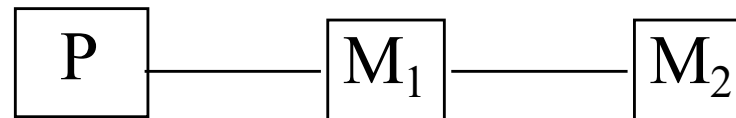
Given the following access sequence (block addresses):  
0,1,8,0,1,9,5,4,0,8,1

How many cold misses are there?

- a) 10
- b) 5
- c) 6

# Impact of Cache Organization on Performance (Ch 4.3.4)

# Memory System Performance



Average access time =  $h_1T_1 + m_1(T_1+T_2)$

- $h_x$  = *hit rate* = probability of finding data in level x
- $(1-h_x) = m_x$  = *miss rate* for level x
- $T_x$  = *hit time* = access time for level x
- $(T_x+T_{x+1})$  = *miss penalty* for level x (assuming processor stalls for the entire duration of a miss)

Impact on performance:

$$\text{CPI} = \text{CPI}_0 + \text{MPI}_1 \times T_2$$

where  $\text{MPI}_1$  is the number of *misses per instruction* at level 1

# Effect of Cache Parameters

## Larger caches

- Slower
- Less capacity misses

## Larger block size

- Exploit spatial locality
- Too big a block increases capacity misses
- Big blocks also increase miss penalty

## Higher associativity

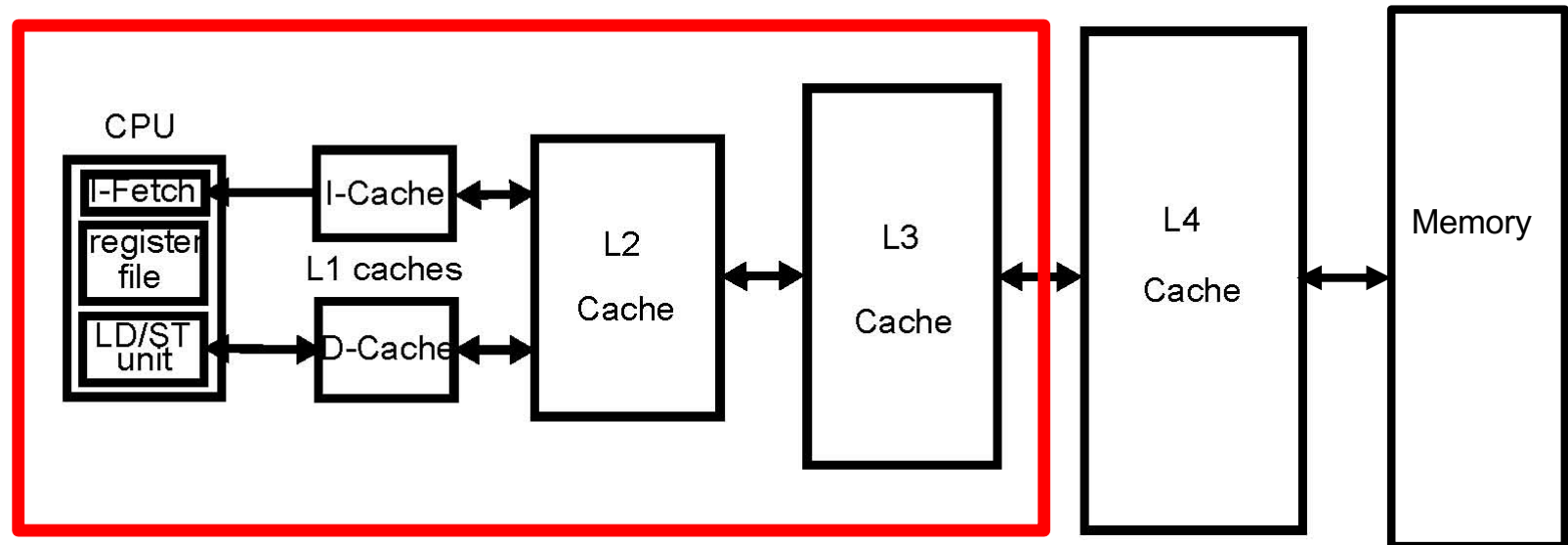
- Reduces number of conflict misses
- 8-16 way set-assoc. as good as fully assoc.
- 2-way set-assoc. cache of size  $N$  has similar miss rate as a direct-mapped cache of size  $2N$
- Higher hit time



# Multi-level Cache Hierarchies

## (Ch 4.2.3)

# Multi-level Cache Hierarchies



Today, 1<sup>st</sup>, 2<sup>nd</sup> and 3<sup>rd</sup> level are on-chip and 4<sup>th</sup> level is off-chip

**Cache inclusion** is maintained:

- • When a block misses in L1 then it must be brought into all levels
- • When a block is replaced in any level  $L_i$ , it must be removed from all levels  $L_j$  where  $j < i$

# Cache Exclusion

**A block can only exist at one level**

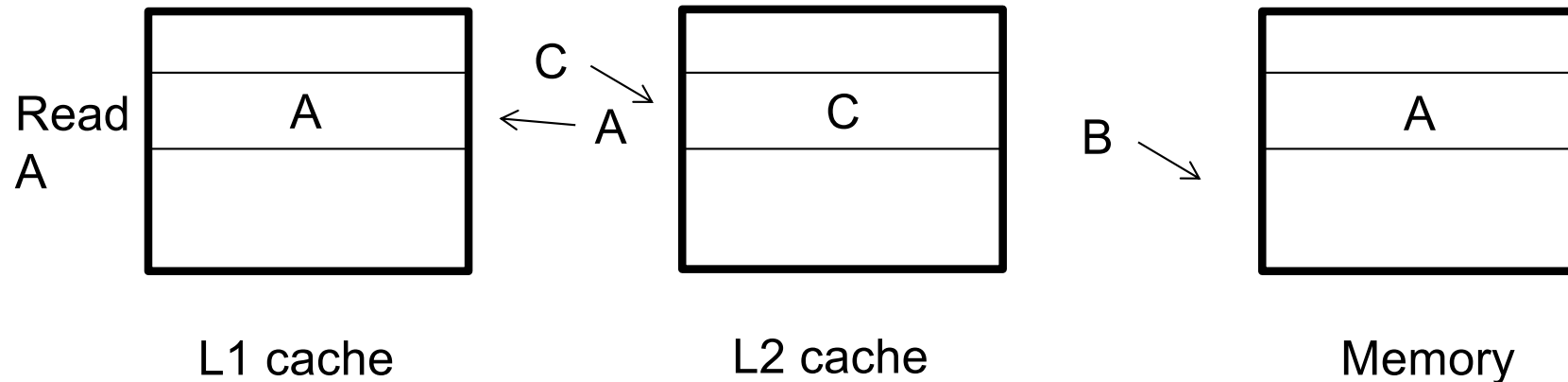
**Benefits:** improved utilization of cache space

**Disadvantages:** Complicates locating a block

**Conditions:**

- If a block is in  $L_i$  then it is not in  $L_j, j > i$
- If a block is in  $L_1$  then all copies are removed from all  $L_j$ 's,  $j > 1$
- If a block is replaced in  $L_i$  then it is allocated in  $L_{i+1}$

# Example of Exclusion Management

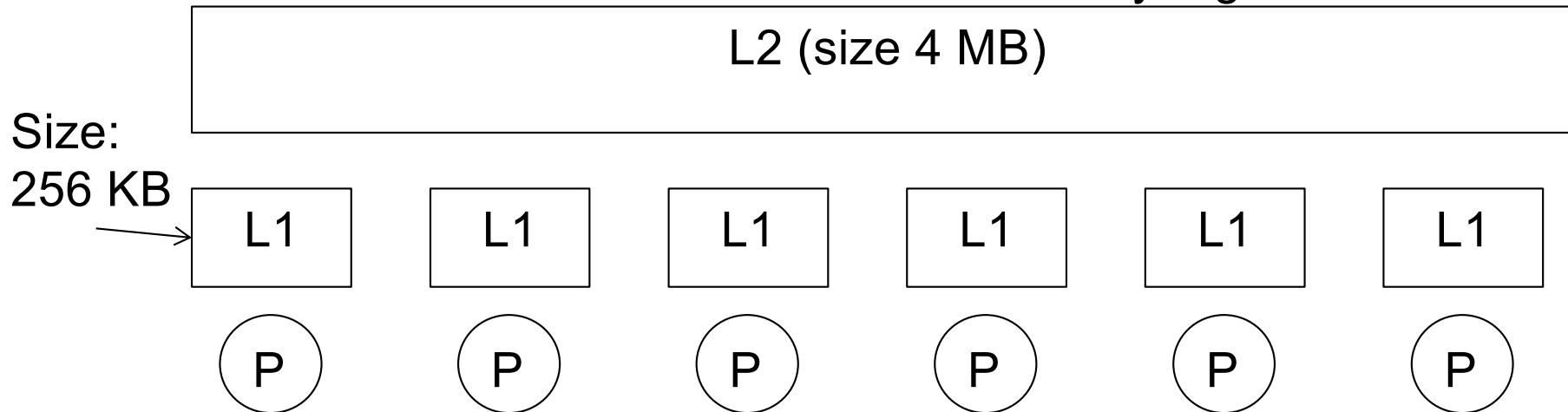


## Note:

- Blocks B and C have to be moved to lower levels even if they are clean
- In an inclusive cache hierarchy, replacements are “silent” (no write-back needed) for clean blocks

## Quiz 5.2

A multicore cache memory organization



Which of the following statements are correct?

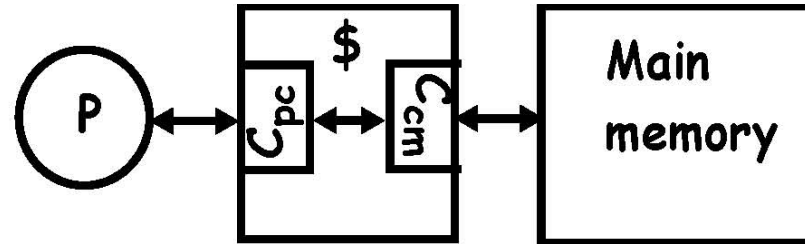
- a) The effective cache capacity is  $4 \text{ MB} + 6 \times 256 \text{ KB}$  if L2 **includes** L1
- b) The effective cache capacity is  $4 \text{ MB} - 6 \times 256 \text{ KB}$  if L2 **includes** L1
- c) The effective cache capacity is 4 MB if L2 **includes** L1
- d) The effective cache capacity is  $4 \text{ MB} + 6 \times 256 \text{ KB}$  if L2 **excludes** L1

# Non-Blocking (Lockup-free) Caches (Ch 4.3.6)

# Non-blocking (Lockup-free) Cache

$C_{cm}$ : cache to memory interface

$C_{pc}$ : processor to cache interface



- **If the cache misses, it does not block**
  - Rather, it handles the miss and keeps accepting accesses from the processor
  - Permits the concurrent processing of multiple misses and hits
- **Cache has to bookkeep all pending misses**
  - **MSHRs (Miss Status Handling Registers)** contain the address of pending miss, the destination block in cache and the destination register
  - Number of MSHRs limits the number of pending misses
- **Data dependencies eventually block the processor**
- **Non-blocking caches are required in dynamically scheduled processors and for prefetches (to be considered next)**

# Lockup-free Caches: Primary and Secondary Misses

**Primary:** The first miss to a block

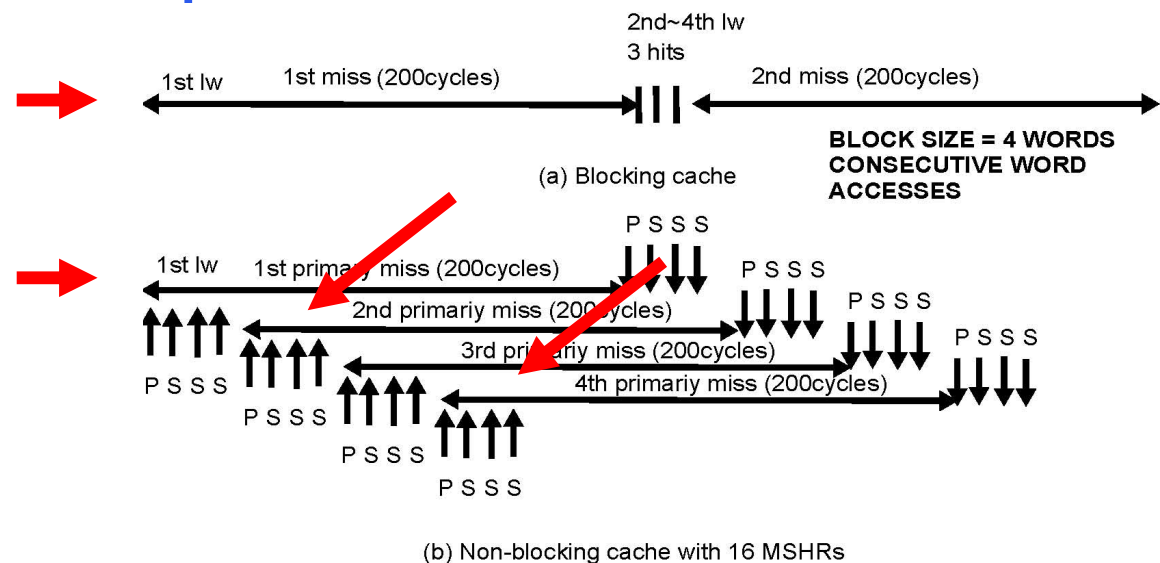
**Secondary:** Following accesses to blocks pending due to primary miss

- Lot more misses (blocking cache only has primary misses)
- Needs MSHRs for both primary and secondary misses

**Misses are overlapped with computation and with other miss**

Example:

```
TOY: LW R1,0(R2)
      ADDI R2,R2,#4
      BNEZ R2,R4, TOY
```



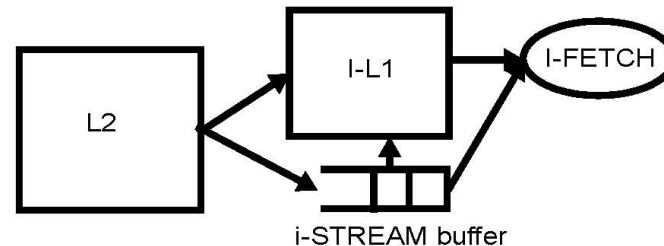


# Cache Prefetching

## (Ch 4.3.7)

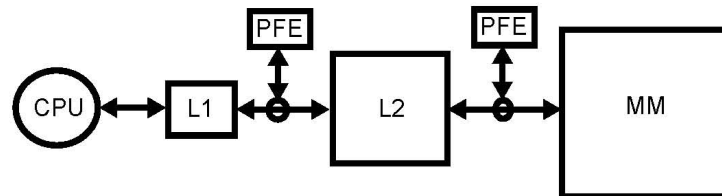
# Hardware Prefetching

- Sequential prefetching of instructions



- ➔ – On an I-fetch miss, fetch two blocks instead of one
- ➔ – Second block is stored in an I-stream buffer
- ➔ – If I-stream-buffer hits, block is moved to L1
- ➔ – I-stream buffer blocks are overlaid if not accessed
- ➔ – Also applicable to data, but less effective

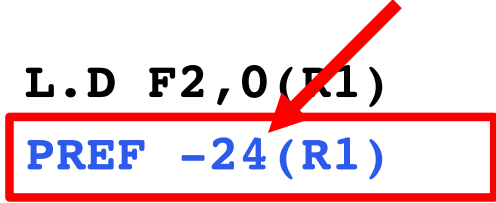
- Hardware prefetching: detect strides in streams of accesses and fetch ahead



# Compiler-controlled Prefetching 1(2)

- Insert prefetch instructions in code
- These are non-binding, load in cache only
- In a loop, we may insert prefetch instructions in the body of the loop to prefetch data needed in future loop iterations

```
LOOP L.D F2,0(R1)
      PREF -24(R1)
      ADD.D F4,F2,F0
      S.D F4,0(R1)
      SUBI R1,R1,#8
      BNEZ R1, LOOP
```



# Compiler-controlled Prefetching 2(2)

**What is required to make it work:**

- **Can work for both Loads and Stores**
- **Requires a non-blocking (lockup-free) cache**
- **Causes instruction overhead**
- **Data must be prefetched on time, so that they are present in cache at the time of access**
- **Data may not be prefetched too early so that they are still in cache at the time of the access**
- **Can easily be done for arrays, but it is also possible to act on pointer accesses**

## Quiz 5.3

Consider software prefetching applied to the following loop and assume that the block size is 8 bytes.

```
LOOP L.D F2,0(R1)
      PREF -24(R1)
      ADD.D F4,F2,F0
      S.D F4,0(R1)
      SUBI R1,R1,#8
      BNEZ R1, LOOP
```

How many miss handling status registers are needed to support it?

- a) 1
- b) 2
- c) 3
- d) 8

# Virtual Memory

## (Ch 4.4)

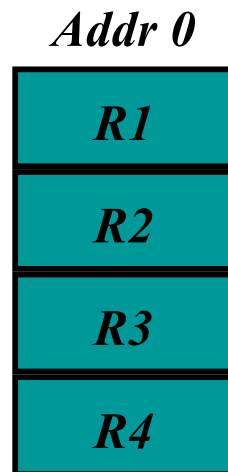
# Motivation: Virtual Memory

## Main motivation for using virtual memory:

- Memory protection between processes
- More flexible software:
  - Possible to allocate larger virtual memory than physical memory available
  - Relocation easier

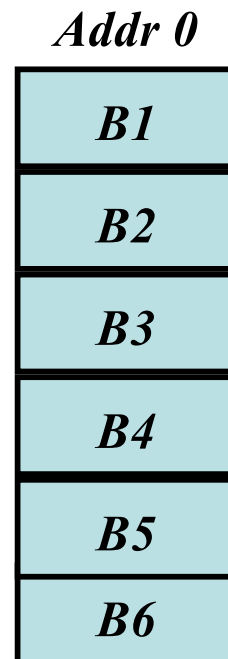
# Virtual Memory 1(3)

*PROGRAM "R"*



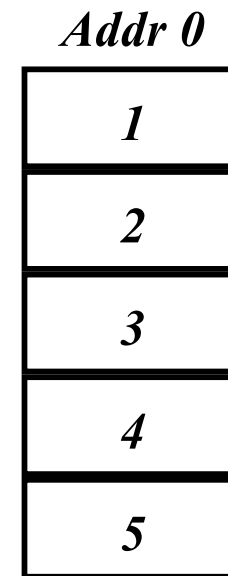
*Each program uses  
its own "virtual"  
address space*

*PROGRAM "B"*



*Main memory is  
shared and uses a  
physical address  
space*

*MAIN MEMORY*



*Virtual Addresses*

*Physical Addresses*

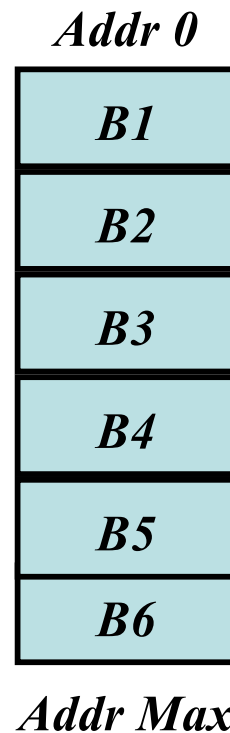


# Virtual Memory 2(3)

*PROGRAM “R”*

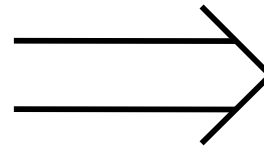


*PROGRAM “B”*

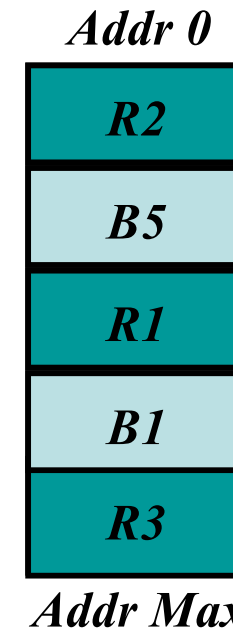


*MAIN MEMORY*

*Virtual addresses  
get translated to  
physical addresses*



*Different  
translations for  
different programs*



*Virtual Addresses*

*Physical Addresses*

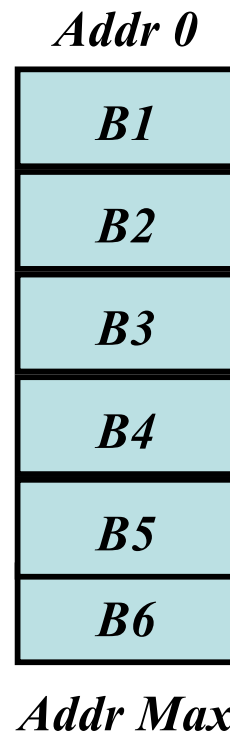
# Virtual Memory 3(3)

*SECONDARY  
MEMORY  
(disk storage)*

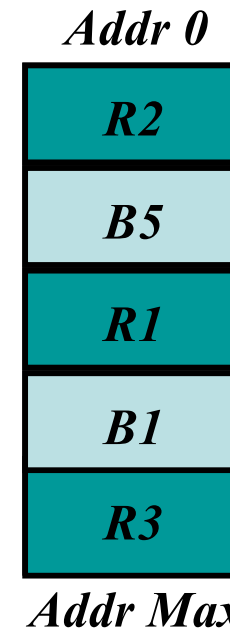
*PROGRAM “R”*



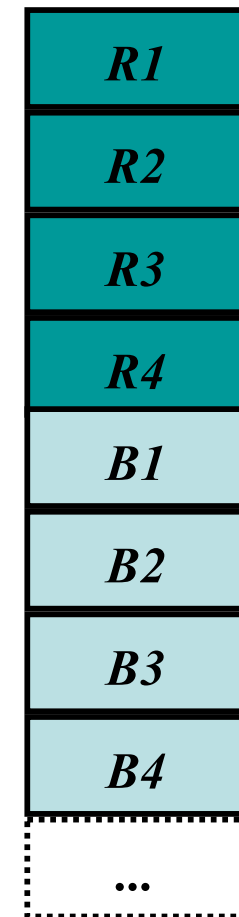
*PROGRAM “B”*



*MAIN MEMORY*



*Main memory acts as a cache for  
secondary memory*



*Virtual Addresses*

*Physical Addresses*

# The Four Questions for Virtual Memory

## Where to place a block (in main memory)?

- fully associative, a page can be placed anywhere

## How to find a block (in main memory)?

- look up in the page table
- on page fault, interrupt CPU and OS fetches from disk

## Which block to replace?

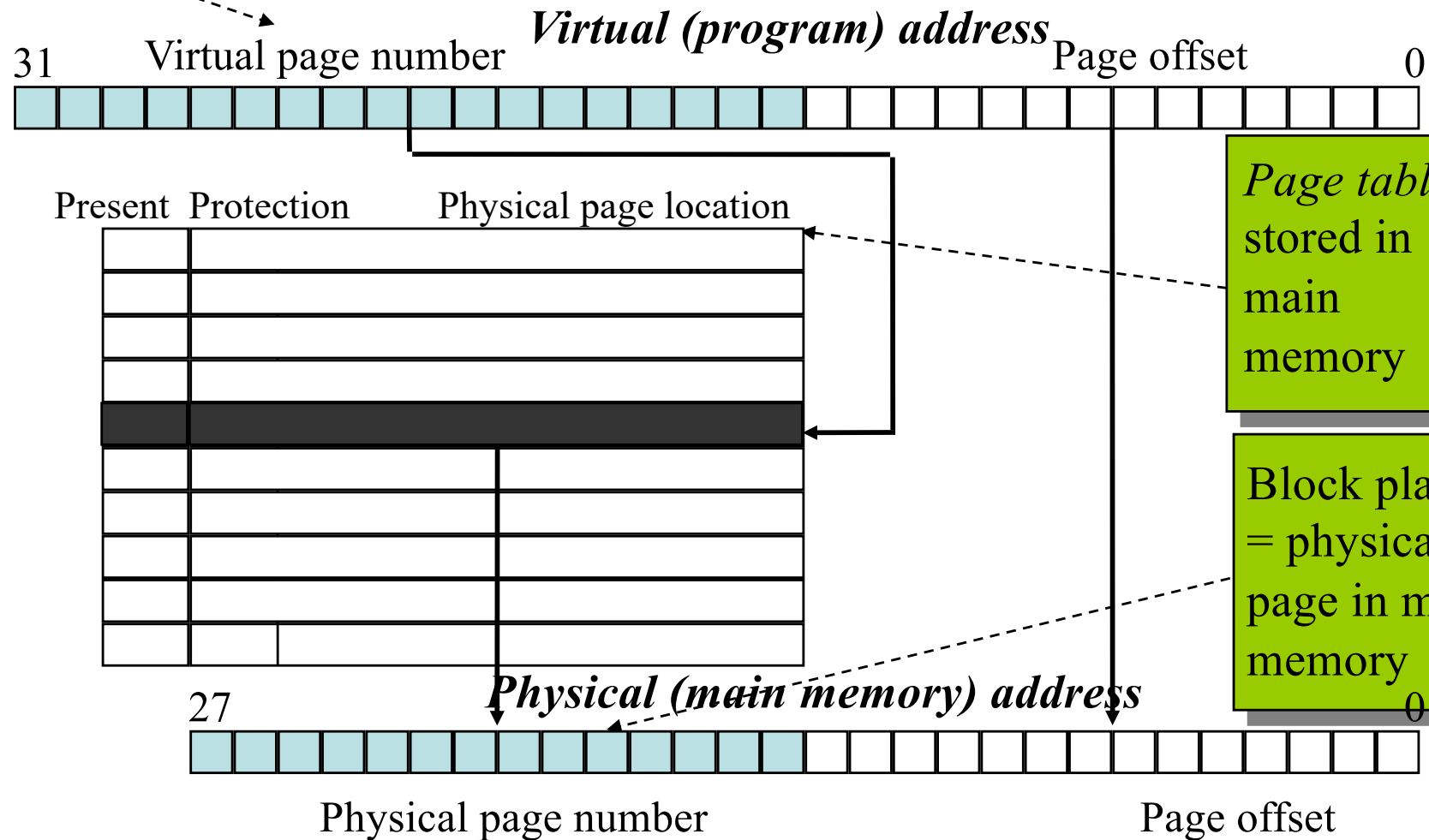
- always LRU (or some approximation)

## What happens on a write?

- always write back (copy back)

# Virtual Address Translation

Block called *page*  
Typically 4-16 KB



## Quiz 4.4

Assuming 64-bit virtual addresses and 8-KB pages and that each page-table entry occupies 4 bytes. How much space does the page table entry occupy?

- a)  $2^{64}$  bytes
- b)  $2^{53}$  bytes
- c)  $2^{44}$  bytes
- d)  $2^{34}$  bytes

# PTE, Page Table Entry

**One entry in the page table:**

- Presence bit
- Dirty bit
- Protection bits
- Reference (accessed) bits
- Physical page location (page number)

*Present in memory or  
must be fetched from  
disk?*

*Page fault  
if not present*

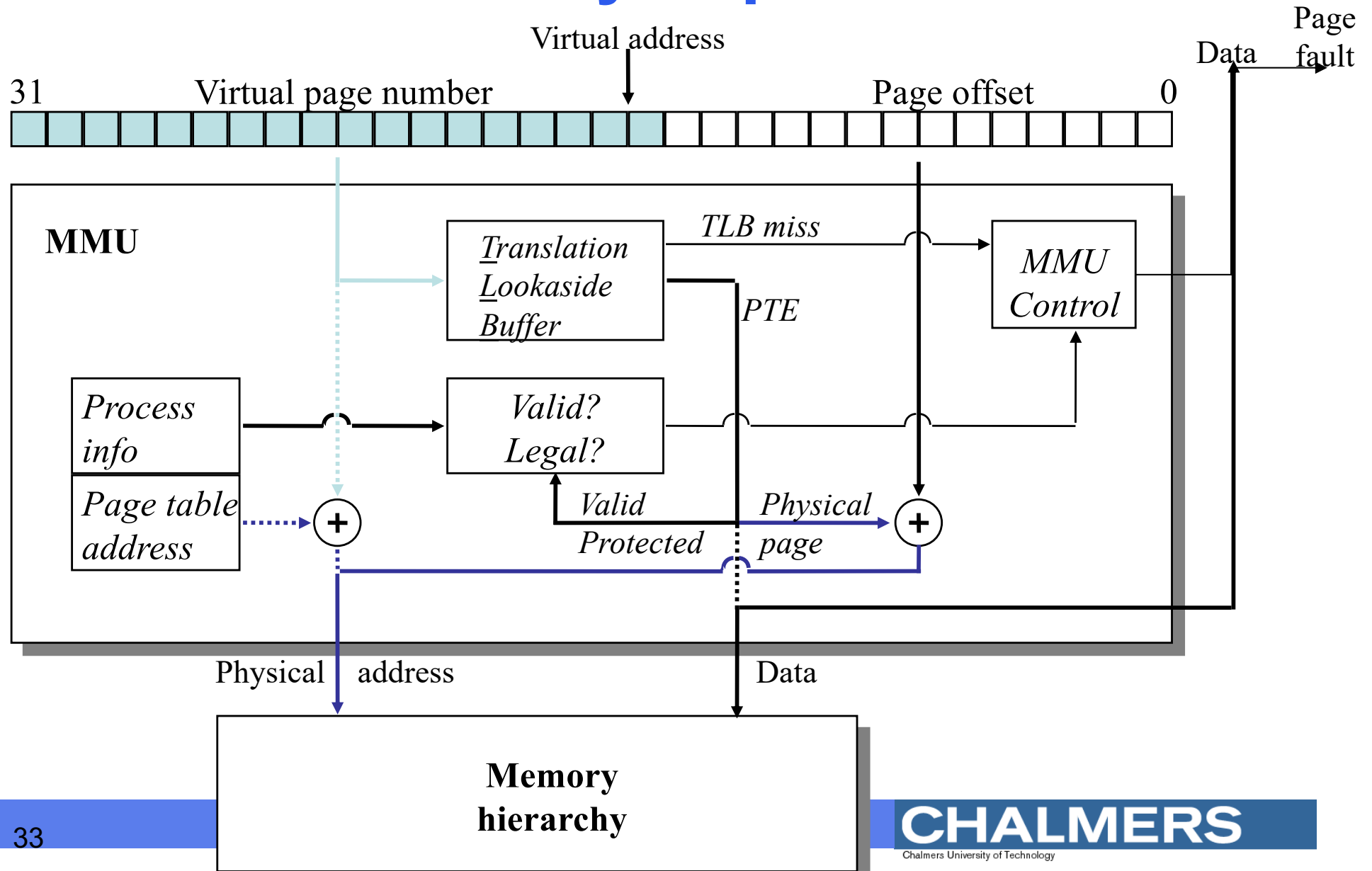
# Translation Lookaside Buffer (TLB)

# Translation Lookaside Buffer (TLB)

- PTEs are cached in a small buffer called the **Translation Lookaside Buffer (TLB for short)**.
- A TLB entry:
  - The same as a PTE but instead of the Presence bit we have a Valid bit (like in a cache)
- A translation not found in the TLB causes a TLB miss



# Virtual Memory Implementation



# TLB miss handling

- TLB misses are handled in hardware or software
- When PTE not found in TLB:
  - Look up entry in page table (main memory)
  - If page not present generate Page Fault
  - If page not mapped generate Exception (invalid adress)
  - Insert entry in TLB
- Stall while handling or restart instruction after handling

# Page Fault Handling

Page fault handled by software (OS). Takes very long time!

1. Mark current process as not executable
2. Execute replacement algorithm
3. If modified, initiate write of chosen page to secondary memory
4. Switch to other tasks during write
5. Write finish causes exception that restarts memory manager
6. Initiate read of requested page from secondary memory
7. Switch to other tasks during read
8. Read finish causes exception that restarts memory manager
9. Update page table (and TLB)
10. Mark process that caused page fault as executable
11. Eventually, original process is restarted

# What you should know by now

- **Cache performance equation**
  - Impact of miss penalty on execution time
- **Classification of cache misses – the 3C model**
  - Impact of cache organizational parameters on miss type
- **Multi-level memory hierarchies**
  - Inclusion and exclusion
- **Non-blocking or lockup-free caches**
  - Primary/secondary misses and MSHRs
- **Hardware and software prefetching**
  - Sequential/stride pref. and pref. instructions
- **Virtual memory and hardware support (TLB)**