COM1006 Devices and Networks (Autumn)
COM1090 Computer Architectures

Lecture #13

- Computer memory

Dr Dirk Sudholt
Department of Computer Science
University of Sheffield

d.sudholt@sheffield.ac.uk
http://staffwww.dcs.shef.ac.uk/~dirk/campus_only/com1006/

Based on Chapters 8, 12, 13 in Clements, Principles of Computer Hardware

u Aims of this lecture

- To explain the concept of memory hierarchy.
- To explain the way in which cache memory accelerates computer performance.
- To describe the three main strategies for cache memory.
- To explain logical-to-physical mapping in memory management.
- To introduce the concept of virtual memory.
- To point out the similarity between strategies for cache memory and virtual memory.

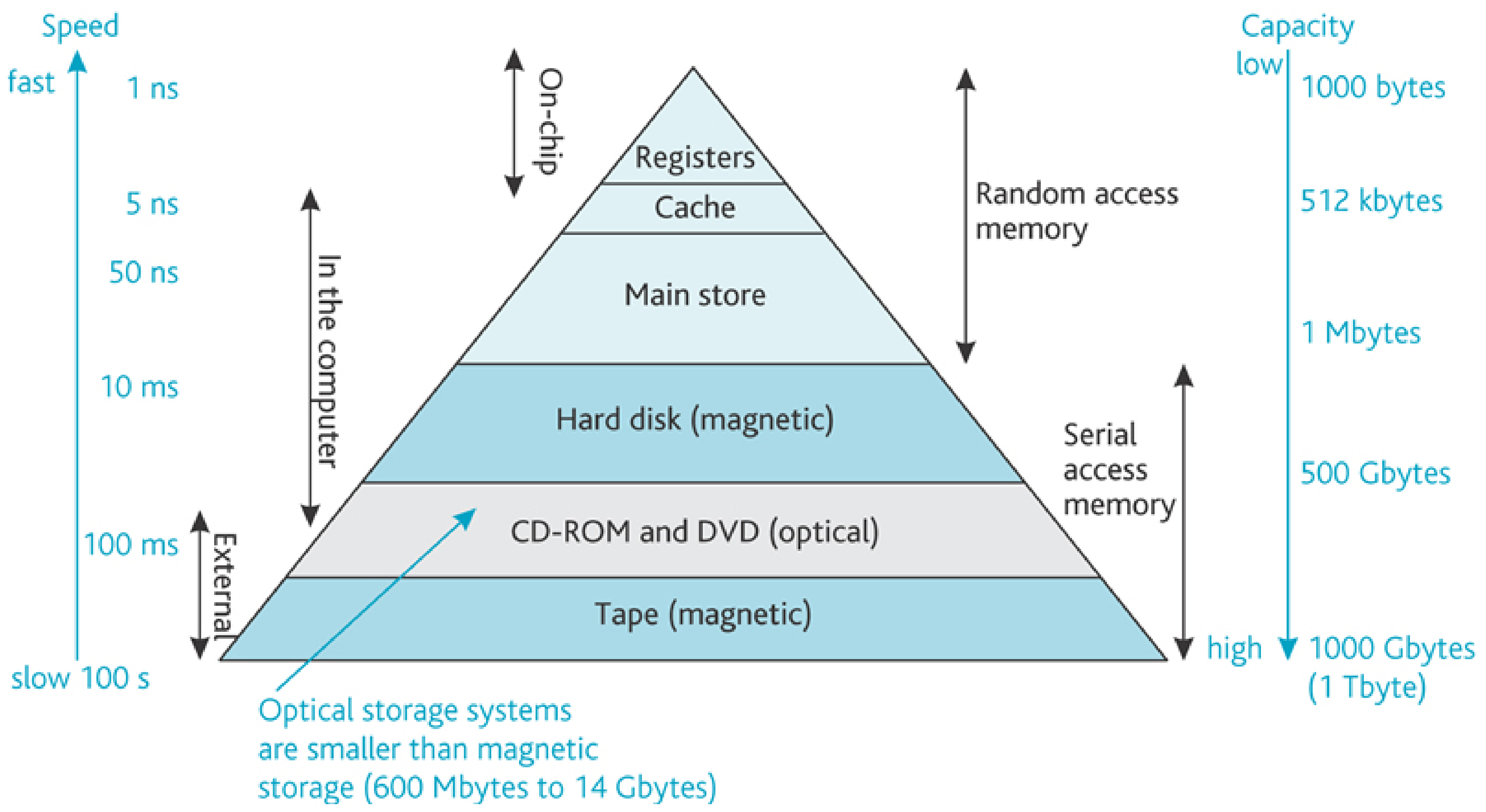
u Computer memory

- Computer memory systems are not homogeneous.
- Many kinds of memory technology (semiconductor, optical, magnetic) – why do we need them?

❓ What are the characteristics of ideal computer memory?

- In practice, memory technologies can be arranged into a hierarchy with fast, expensive, low capacity at the top and slow, cheap, large capacity at the bottom.
- Combining different technologies allows memory systems that are low-cost but close in performance to a system built only with high-speed random-access memory.

u Memory hierarchy

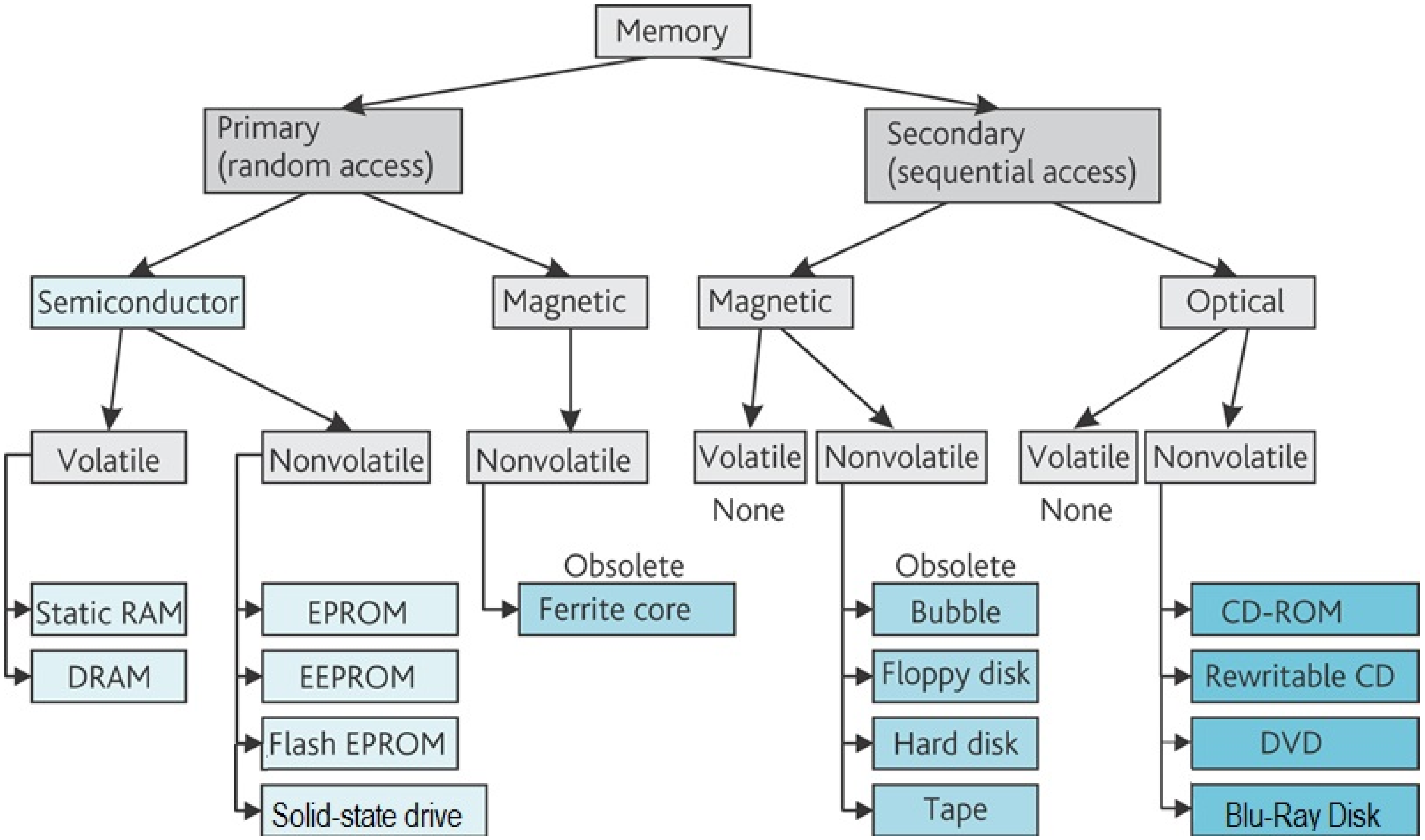


u Memory hierarchy design

- The goal of the hierarchy design is to have the CPU see the entire hierarchy as having the capacity of the bottom level while being accessible at the speed of the top level.
- Hierarchy is controlled by the hardware (for cache and main store) or by operating system (virtual memory). Use of registers is determined by compiler.
- Hierarchy design is influenced by the phenomenon of locality of reference:

The collection of data locations referenced in a short period of time in a running computer often consists of relatively predictable clusters

u Classes of memory



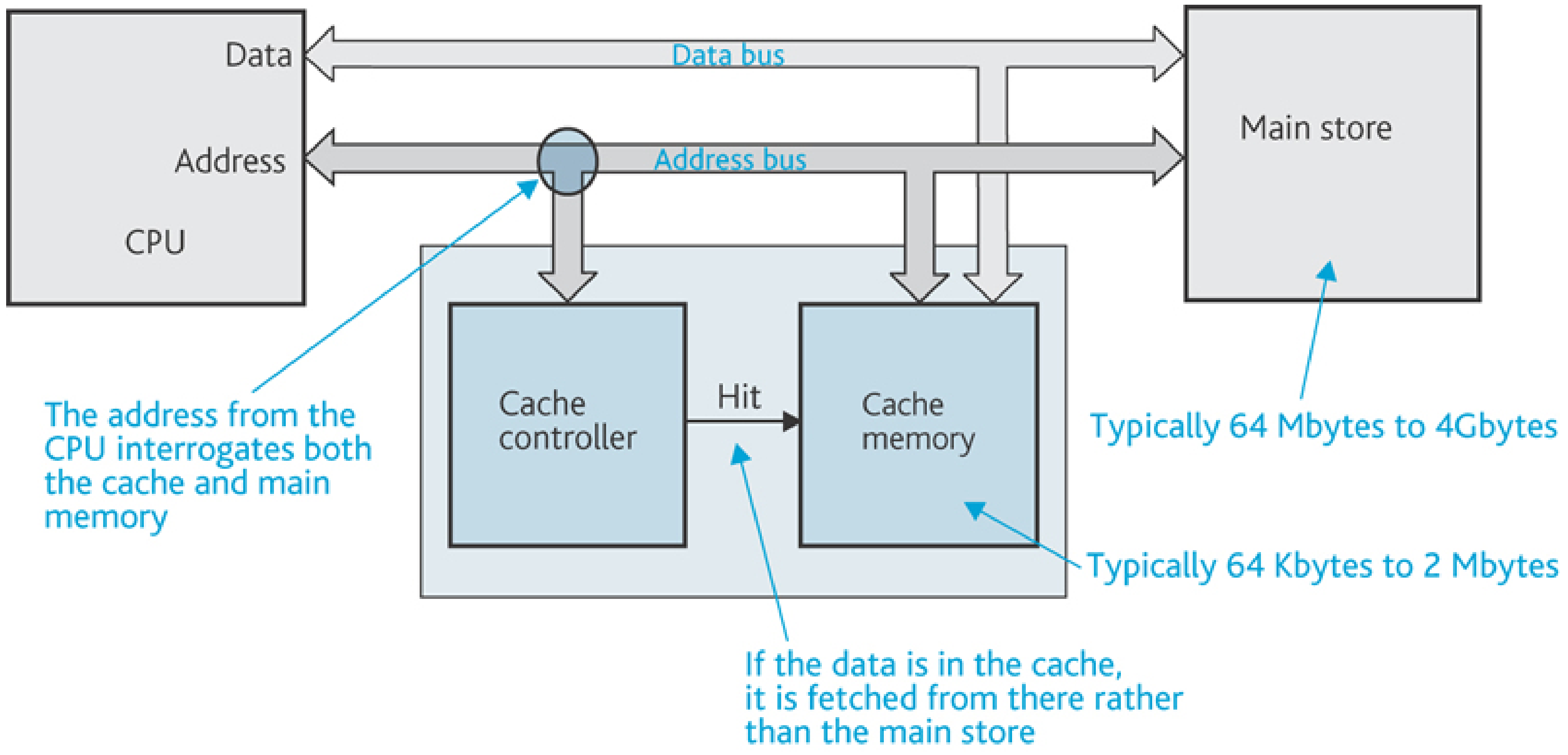
u Classes of memory

- Random access memory (RAM)
  - static (SRAM): based on flip-flops, require power
  - dynamic (DRAM): capacitor stores charge to represent 0/1 state, must be refreshed regularly, compact, low power
- Read-only memory (ROM)
  - EPROM: erasable programmable memory, erased by UV light
  - EEPROM: electrically erasable programmable memory
  - Flash EEPROM (and also flash RAM) : nonvolatile memory in which a section of memory cells is erased in a single step (or ‘flash’). Basis of many ‘solid state’ drives

u Cache memory

- Cache memory (from French ‘cacher’ = ‘to hide’) can dramatically increase performance at relatively little cost.
- High speed memory, can be accessed rapidly by CPU.
- Analogy: cache is like a personal address book as opposed to telephone directory.
- Cache memory connected in parallel with main memory on address bus (data held in cache is also in main store).
- Cache controller determines whether data accessed by CPU is in the cache, or must be obtained from main store.
- Controller returns a hit if required data is in the cache.

u Structure of cache memory


Data
Address
CPU
Data bus
Address bus
Main store
Cache controller
Hit
Cache memory
The address from the CPU interrogates both the cache and main memory
Typically 64 Mbytes to 4Gbytes
Typically 64 Kbytes to 2 Mbytes
If the data is in the cache, it is fetched from there rather than the main store

u Effect of cache memory on performance

- Key parameter of cache memory is the hit ratio (*h*), defined as ratio of hits to all accesses.
- Hit ratio determined by statistical observation of real system, depends on nature of program being executed and cannot readily be calculated.
- To quantify effect of cache on performance, define the following terms:

Access time of main store	t_m
Access time of cache memory	t_c
Hit ratio	h
Speedup ratio	S

u Cache and performance

- *N* accesses to system without cache memory requires Nt_m seconds.
- *N* accesses to system with cache requires $N(ht_c+(1-h)t_m)$ seconds (where $(1-h)$ is the miss ratio).
- The figure of merit for a computer with cache is the speedup ratio, *S*, which is ratio of memory access time without cache to access time with cache:

$$S = \frac{Nt_m}{N(ht_c + (1-h)t_m)} = \frac{t_m}{ht_c + (1-h)t_m}$$

u Cache and performance (cont'd)

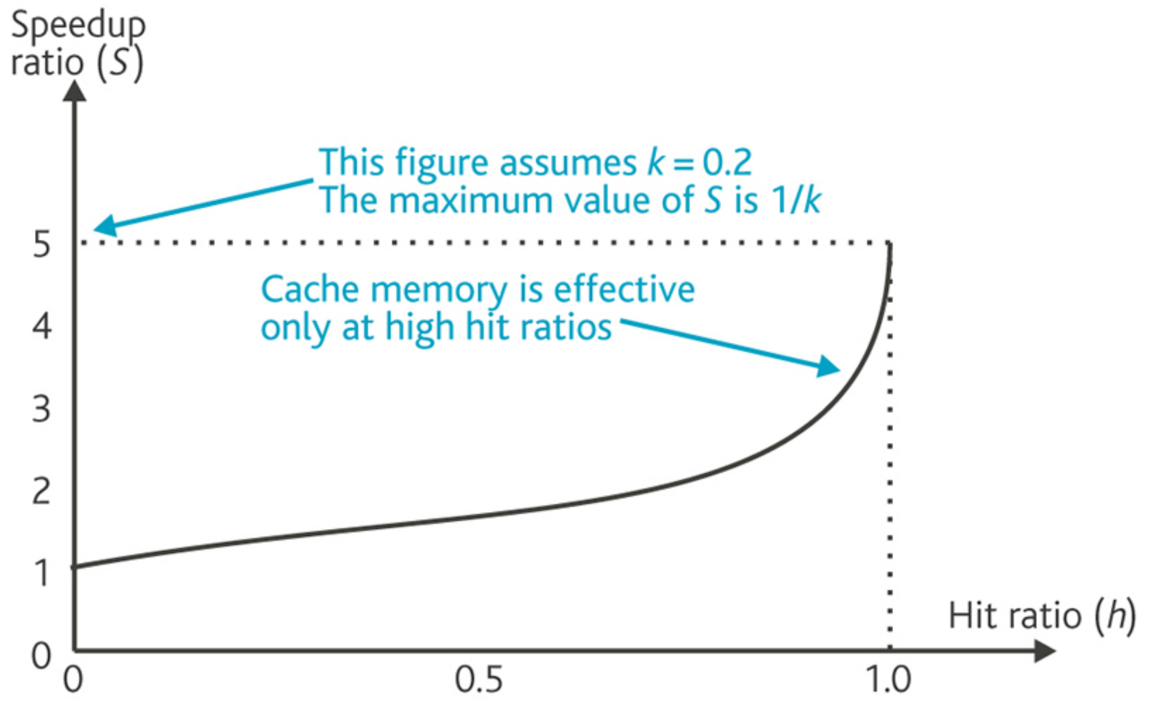
- Convenient to introduce parameter $k = t_c / t_m$.
- We can now write the expression for S as:

$$S = \frac{t_m / t_m}{ht_c / t_m + (1 - h)t_m / t_m} = \frac{1}{hk + (1 - h)}$$

- Typical values are $t_c = 10$ ns and $t_m = 50$ ns, giving $k = 0.2$.
- Note that $S = 1$ when $h = 0$ (all accesses to main memory).
- When $h = 1$, $S = 1/k$ (all accesses to cache).

u Speedup as a function of hit ratio

- Only when hit ratio near to 90% does the effect of cache become really significant.
- Fortunately effect of locality of reference means that hit ratio is usually very high – data required by processor is highly clustered.

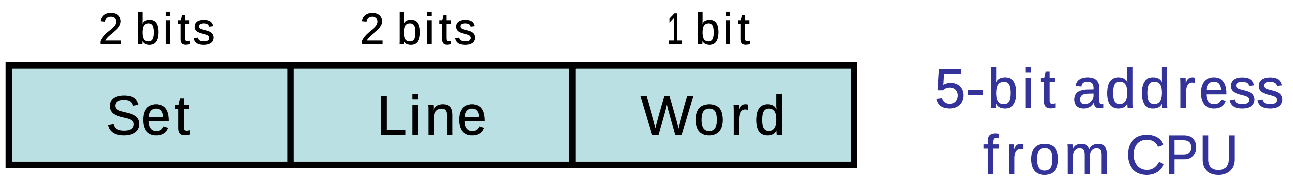


u Cache organisation

- Three main ways of organising cache memory:
  - direct-mapped
  - associative-mapped.
  - set associative-mapped
- Each has different performance/cost trade-offs.
- The approaches differ in how they choose the subset of main memory that is represented in cache.
- The basic unit of data transferred between cache and main store is a line (also called a ‘block’), usually between 4 and 32 bytes.

Direct-mapped cache

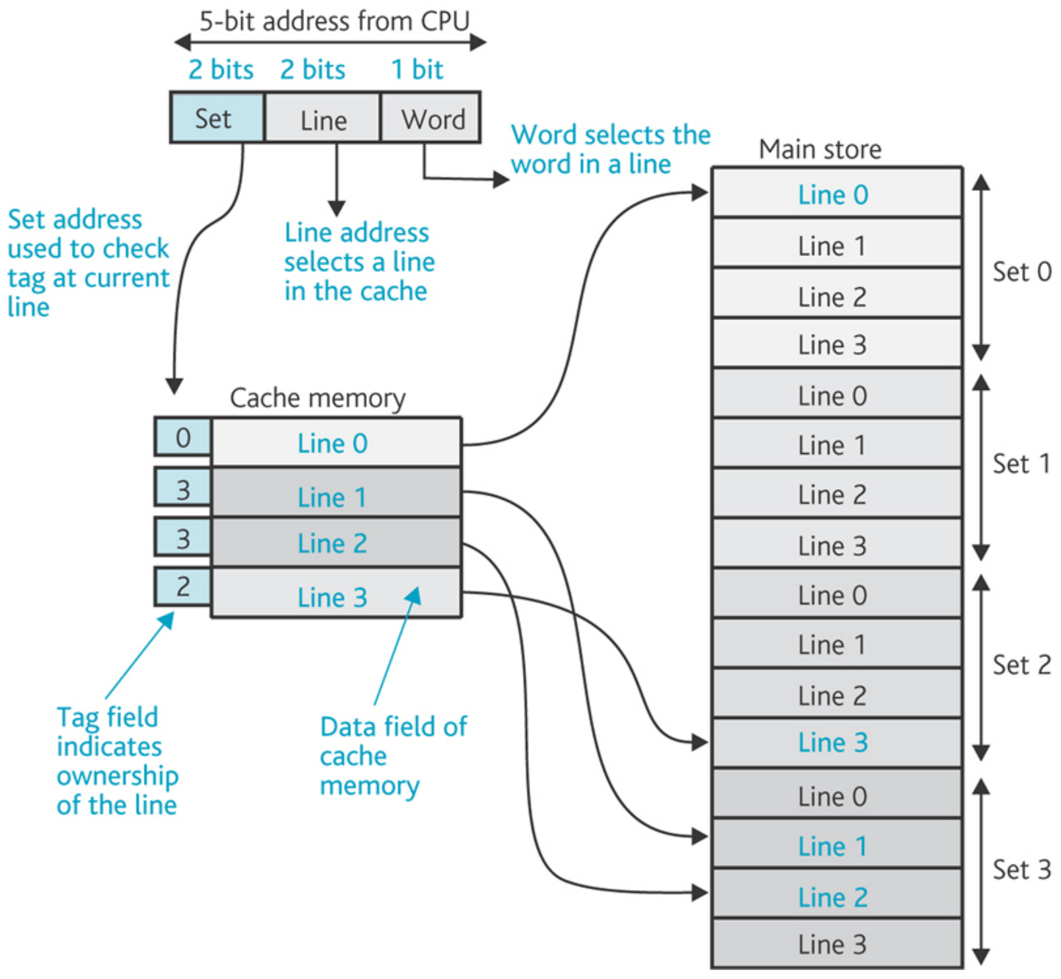
- *N* Lines are organised into *M* sets. Cache memory contains *N*/*M* lines, which may correspond to any set.
- Example: Consider main memory of 32 words accessed by a 5-bit address bus from the CPU. Each line contains two words, 4 sets containing 4 lines each (4x4x2=32)



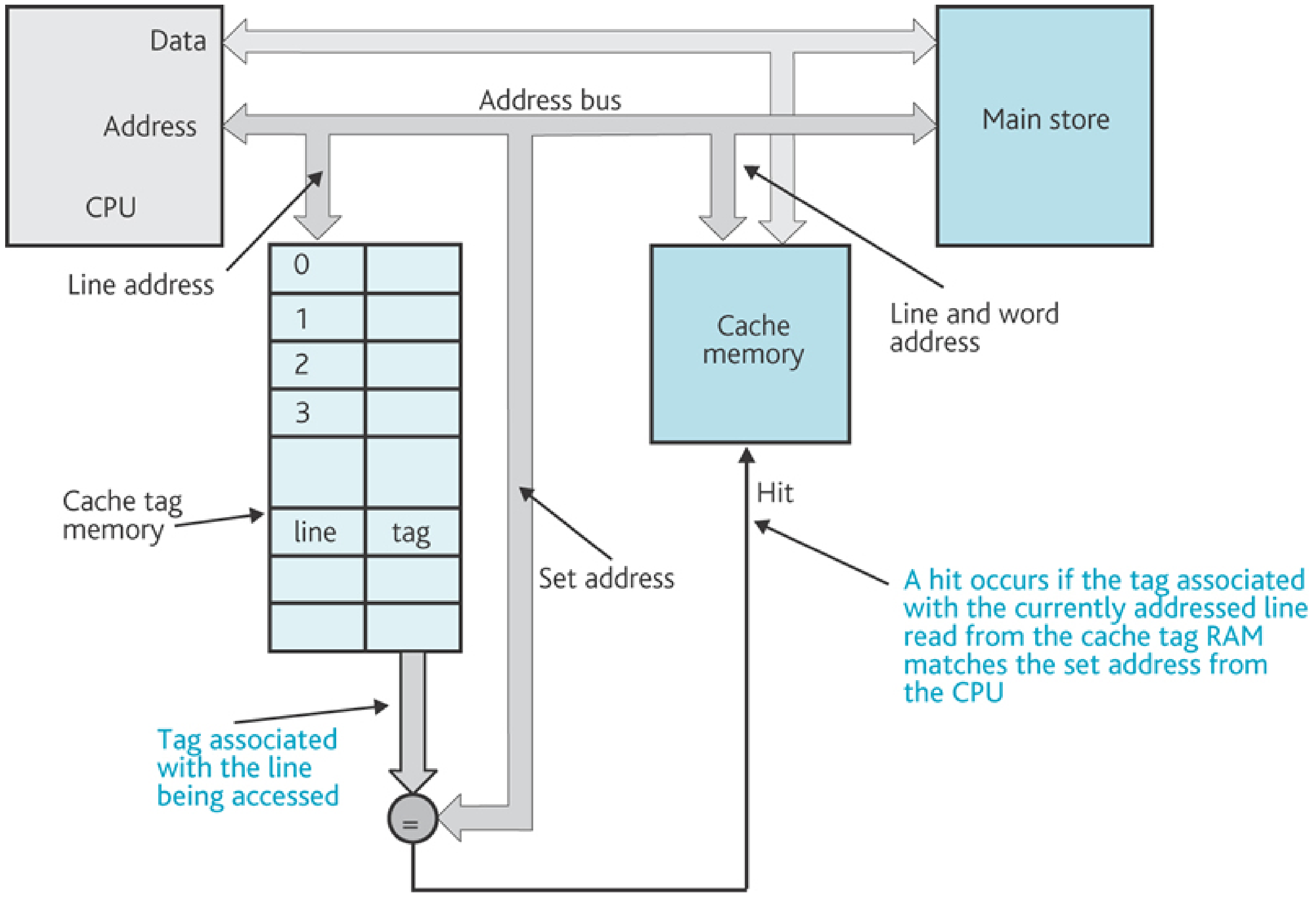
- Cache is indexed by line, so contains 4 lines in this case.
- Tag field in cache records set that the line came from.
- Set in address cross-checked against tag by comparator.

u Direct-mapped cache (cont'd)

- Example 1: address 11010_2 is word 0 of line 1 in set 3
- Hit: line 1 of cache is used.
- Example 2: address 10101_2 is word 1 of line 2 in set 2
- Miss: cache is updated by copying contents of address 10101_2 into line 2 of the cache and setting its tag to 2.


5-bit address from CPU
2 bits
2 bits
1 bit
Set
Line
Word
Word selects the word in a line
Set address used to check tag at current line
Line address selects a line in the cache
Main store
Line 0
Line 1
Line 2
Line 3
Set 0
Set 1
Set 2
Set 3
Cache memory
0
3
3
2
Line 0
Line 1
Line 2
Line 3
Tag field indicates ownership of the line
Data field of cache memory

u Implementation of direct-mapped cache



Data
Address
CPU
Address bus
Main store
Line address
0
1
2
3
line
tag
Cache tag memory
Cache memory
Line and word address
Set address
Hit
=
A hit occurs if the tag associated with the currently addressed line read from the cache tag RAM matches the set address from the CPU
Tag associated with the line being accessed

u Pros and cons of direct-mapped cache

- Direct-mapped cache is just high-speed memory with built-in comparator, inexpensive and easy to build.
- Cache memory and tag memory are independent and can function in parallel.
- Performance can be poor in some cases. Consider this:

```
repeat
    get data from set i line j
    compare with data from set k line j
until done
```

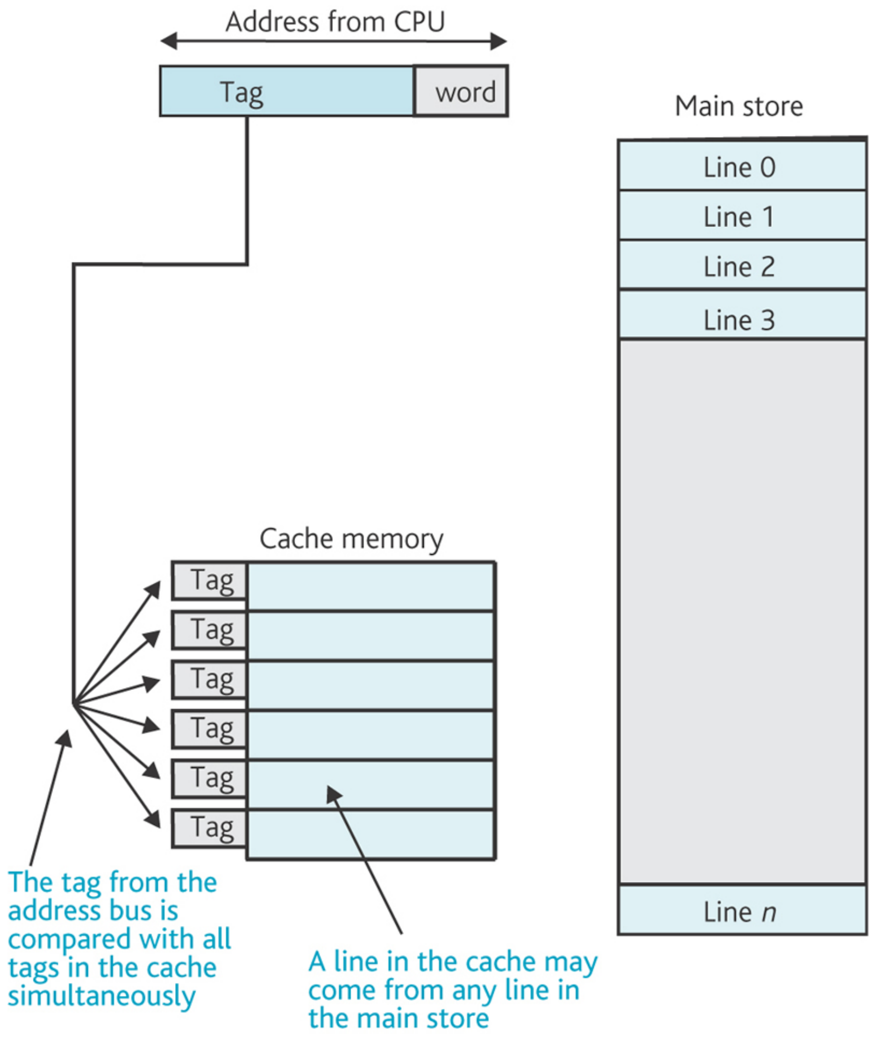
- Data in line *j* must be loaded into the cache twice on each iteration of the loop. In practice such cases have little impact on average performance.

u Associative-mapped cache

- Associative-mapped cache places no restrictions on the data it can contain (cf. direct-mapped, which cannot contain the same line from two different sets).
- Address from CPU divided into tag and word fields.
- Example: Consider a system with 1MB (2^{20} bytes) of main store and 64 KB (2^{16} bytes) of associatively-mapped cache.
  - Assume size of line is four 32-bit words (16 bytes)
  - Cache composed of $2^{16}/16 = 4096$ lines
  - Main memory composed of $2^{20}/16 = 65{,}536 = 2^{16}$ lines
  - Any line in main memory may be stored in cache, so a 16-bit cache tag is required to identify the line.

u Implementing associative-mapped cache

- Associate-mapped cache employs a special associative memory that simultaneously compares the address tag to all the tags in cache.
- Hit if input tag matches one of the cache tags, and corresponding line returned from cache.
- Miss if no matching tag in cache.


Address from CPU
Tag
word
Main store
Line 0
Line 1
Line 2
Line 3
Line n
Cache memory
Tag
Tag
Tag
Tag
Tag
Tag
The tag from the address bus is compared with all tags in the cache simultaneously
A line in the cache may come from any line in the main store

u Pros and cons of associative-mapped cache

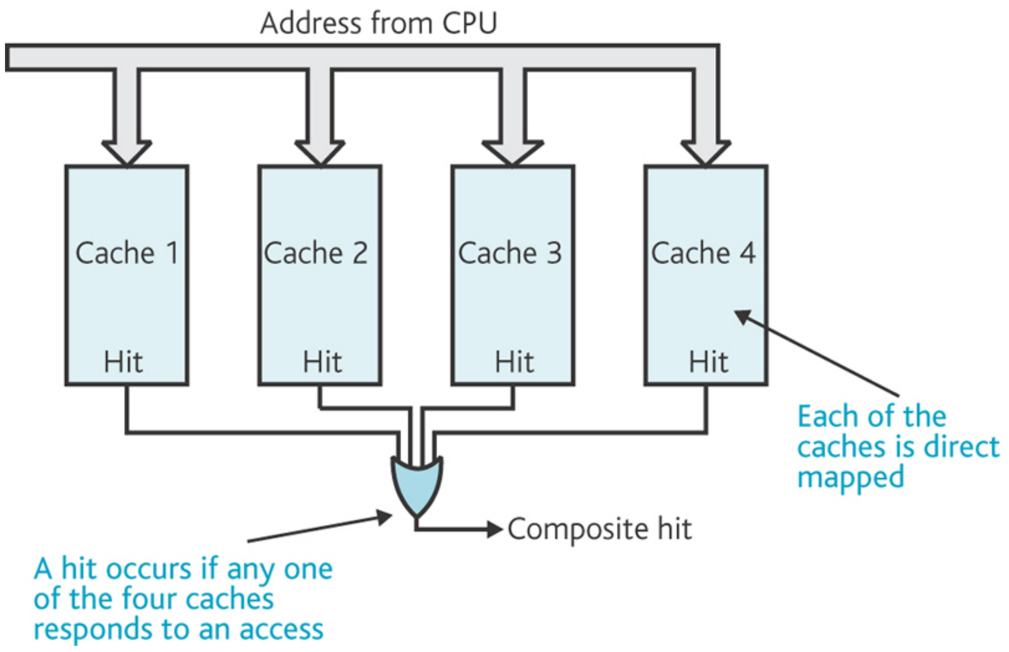
- Associative memory is specialised and is not addressed in the same way as conventional memory:
  - conventional: refer to a specified location
  - associative: do you have this item stored somewhere?
- Large associative memories currently not cost effective.
- When associative cache is full, an existing line must be over-written and this requires a (possibly complex) line replacement policy.

u Set-associative mapped cache

- Most computers use a compromise between direct-mapped cache and associative-mapped cache.
- A set-associative mapped cache consists of a number of direct-mapped caches operating in parallel.
- If cache has *M* parallel sets, an *M*-way comparison is performed on all members of the set.
- Simplest arrangement is two-way set-associative cache in which each line is duplicated.
- This allows line *j* from set *i* and line *j* from set *k* to be stored at the same time in cache memory (see earlier example).

u Implementation of set-associative cache

- Hit occurs if the requested line and tag occur in any of the caches (4 in this example).
- Logic is not complex because number of caches is small.
- Hit indicated by ORing the hit outputs of all caches.

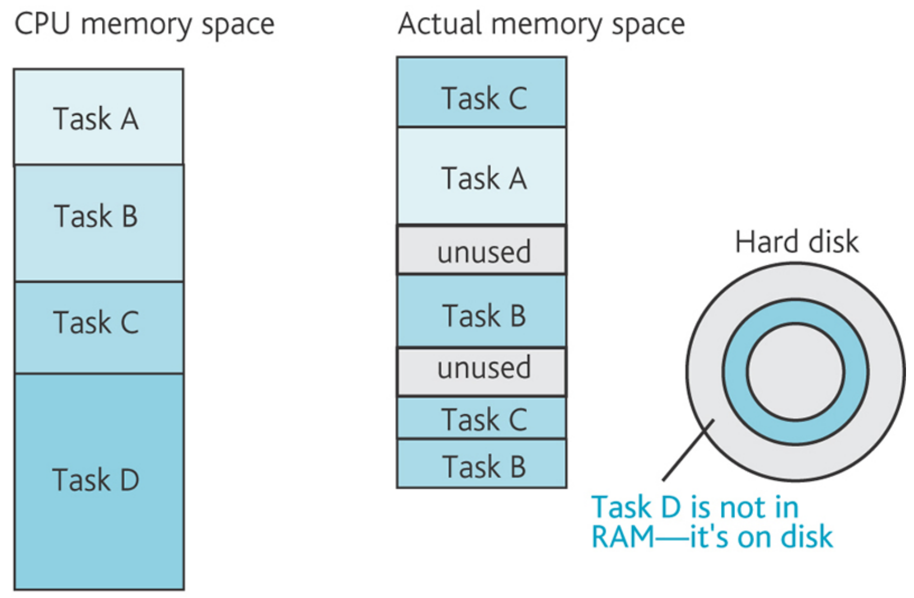

Address from CPU
Cache 1
Cache 2
Cache 3
Cache 4
Hit
Hit
Hit
Hit
Each of the caches is direct mapped
Composite hit
A hit occurs if any one of the four caches responds to an access

u Memory management

- In simple computer systems, address generated by CPU corresponds to location in memory.
- Modern operating systems use memory management to translate logical address generated by CPU to physical address (actual memory location).
  - Physical address may correspond to memory contents that are temporarily stored on hard disk, so-called virtual memory. Allows running programs larger than the main memory.
  - Used in multitasking operating systems to give the impression that each process has sole control of the CPU.
  - Used to protect one process from being corrupted by another process (memory protection).

u Why is logical to physical mapping needed?

- As the operating system creates new processes and removes old ones, memory becomes fragmented.
- Logical address space appears contiguous but physical memory is not.



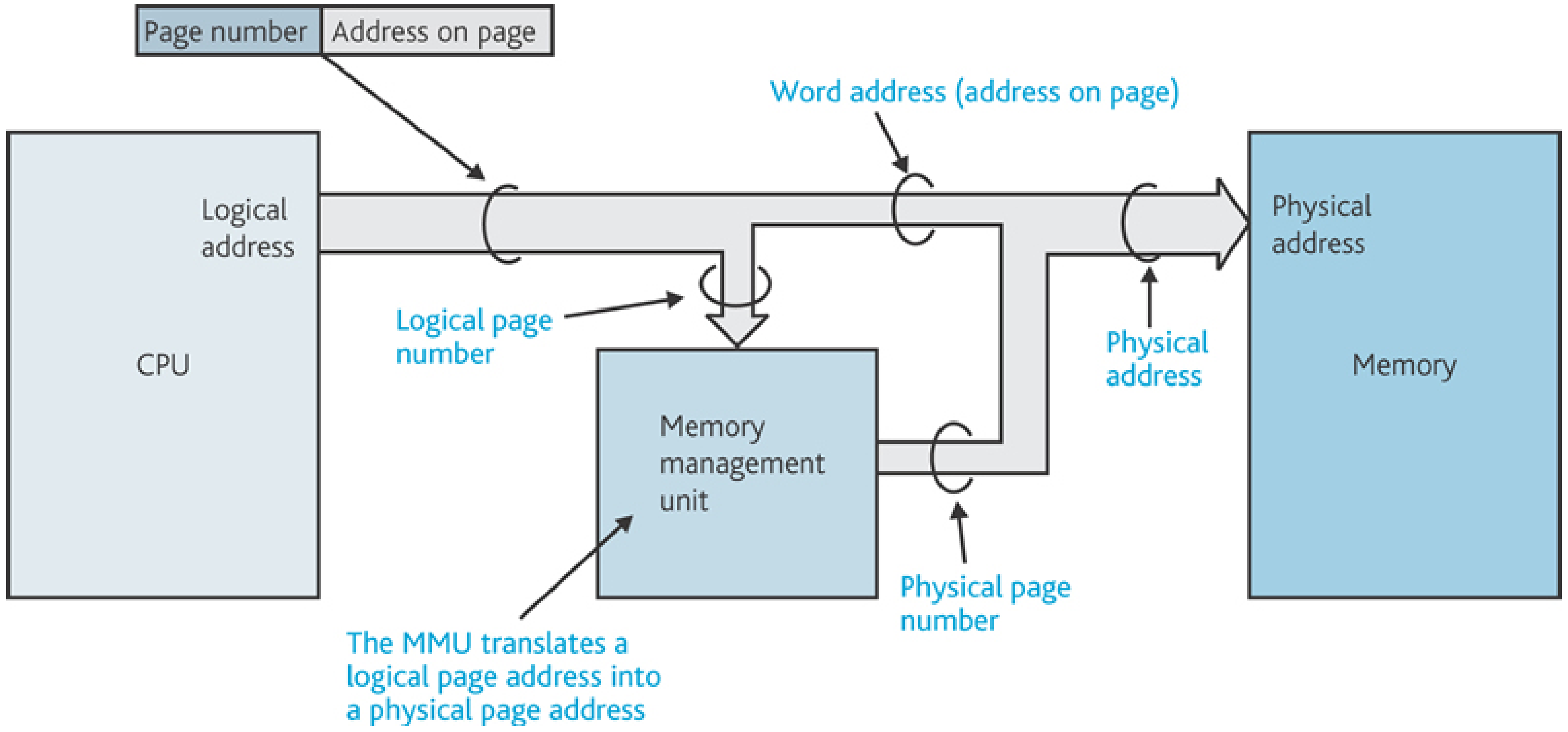
(a) Logical address space. (b) Physical address space.

- If logical address space is larger than physical address space, some memory contents will be temporarily swapped to disk (virtual memory).

u Memory management unit (MMU)

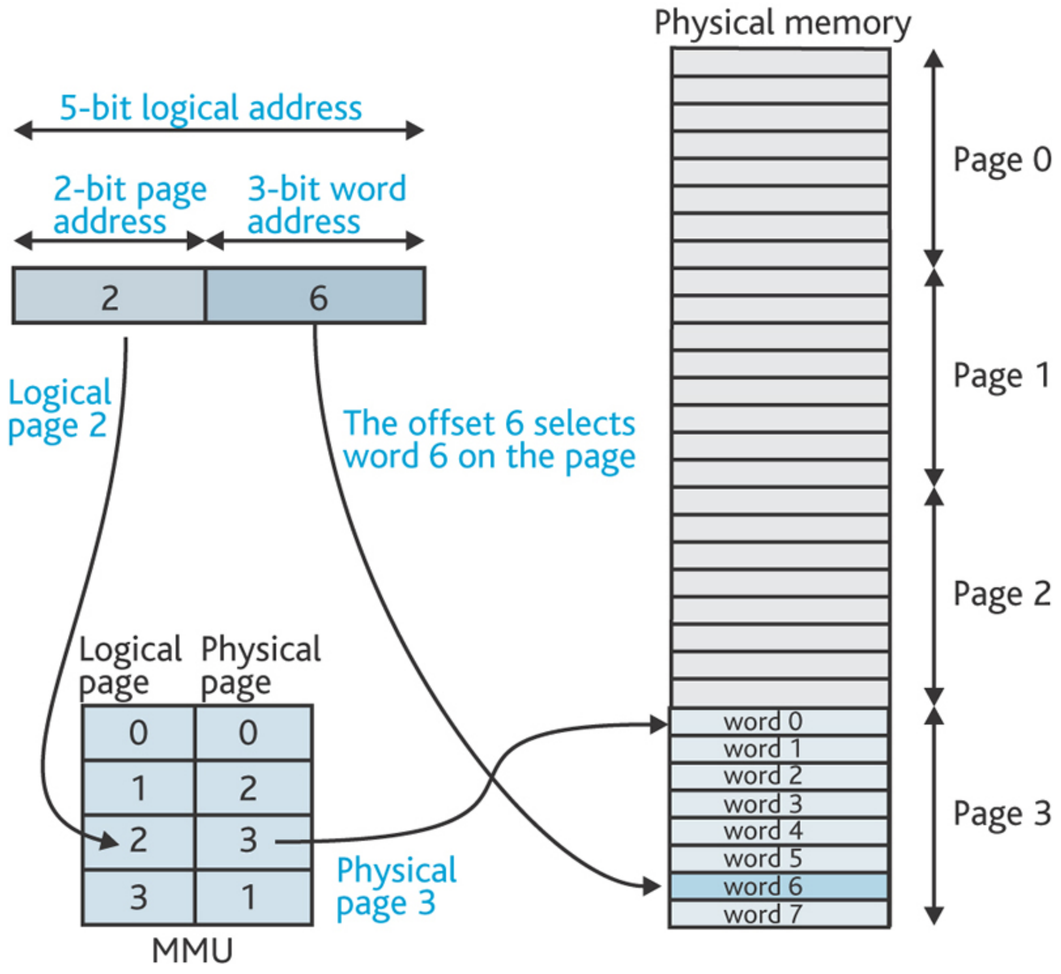
- Modern chips have an integrated memory management unit (MMU) (cf. cache controller).
- MMU maps logical address to physical address and keeps track of where data are stored (disk or main store).
- Physical memory is divided into blocks called pages (cf. lines).
- Logical address has two parts:
  - page address
  - word address
- Word address goes directly to memory but page address goes to the MMU.

u Memory management unit schematic



u Logical to physical mapping

- Consider 5-bit logical address consisting of 2-bit page address and 3-bit word address.
- MMU maps request for word 6 in logical page 2 to word 6 in physical page 3.



u Dealing with page faults

- Principles governing virtual memory are essentially the same as those governing cache memory.
- If requested page is present in RAM, the logical to physical translation occurs and information is accessed.
- If requested page is not in RAM, then:
  - MMU sends a special interrupt to the processor called a page fault.
  - Operating system copies page of data from disc to RAM.
  - Operating system updates page mapping table.
- Only works effectively if data is in RAM most of the time.
- Fortunately the 80:20 rule applies: “80% of the time the processor accesses only 20% of the program”.

u Summary

- The goal of the memory hierarchy is to provide both capacity and speed.
- Three main ways of organising cache memory (direct-mapped, associative-mapped, set associative-mapped) with different performance/cost trade-offs.
- Computers use memory management to translate logical address generated by CPU to physical address.
- In virtual memory, physical memory consists of small high-speed RAM and larger low-speed disc store.
- Similar principles underlie the operation of cache and virtual memory.