CHAPTER 7
LARGE AND FAST: EXPLOITING MEMORY HIERARCHY

Topics to be covered

- Principle of locality
- Memory hierarchy
- Cache concepts and cache organization
- Virtual memory concepts

PRINCIPLE OF LOCALITY

Principle of Locality states that programs access a relatively small portion of their addresses at any instant in time. It creates the illusion of a large memory that we can access as fast as a very small memory.

Two types of locality inherent in programs are:

1. **Temporal locality**: Locality in time

   If an item is referenced, it will tend to be referenced again soon.

   Example: most programs contains loops, so instructions and data are likely to be accessed repeatedly, showing high amounts of temporal locality.

2. **Spatial locality**: Locality in space

   If an item is referenced, items whose addresses are close by will tend to be referenced soon.

   Example: access to elements of arrays naturally have high degrees of spatial locality.

The principle of locality allows the implementation of memory hierarchy.

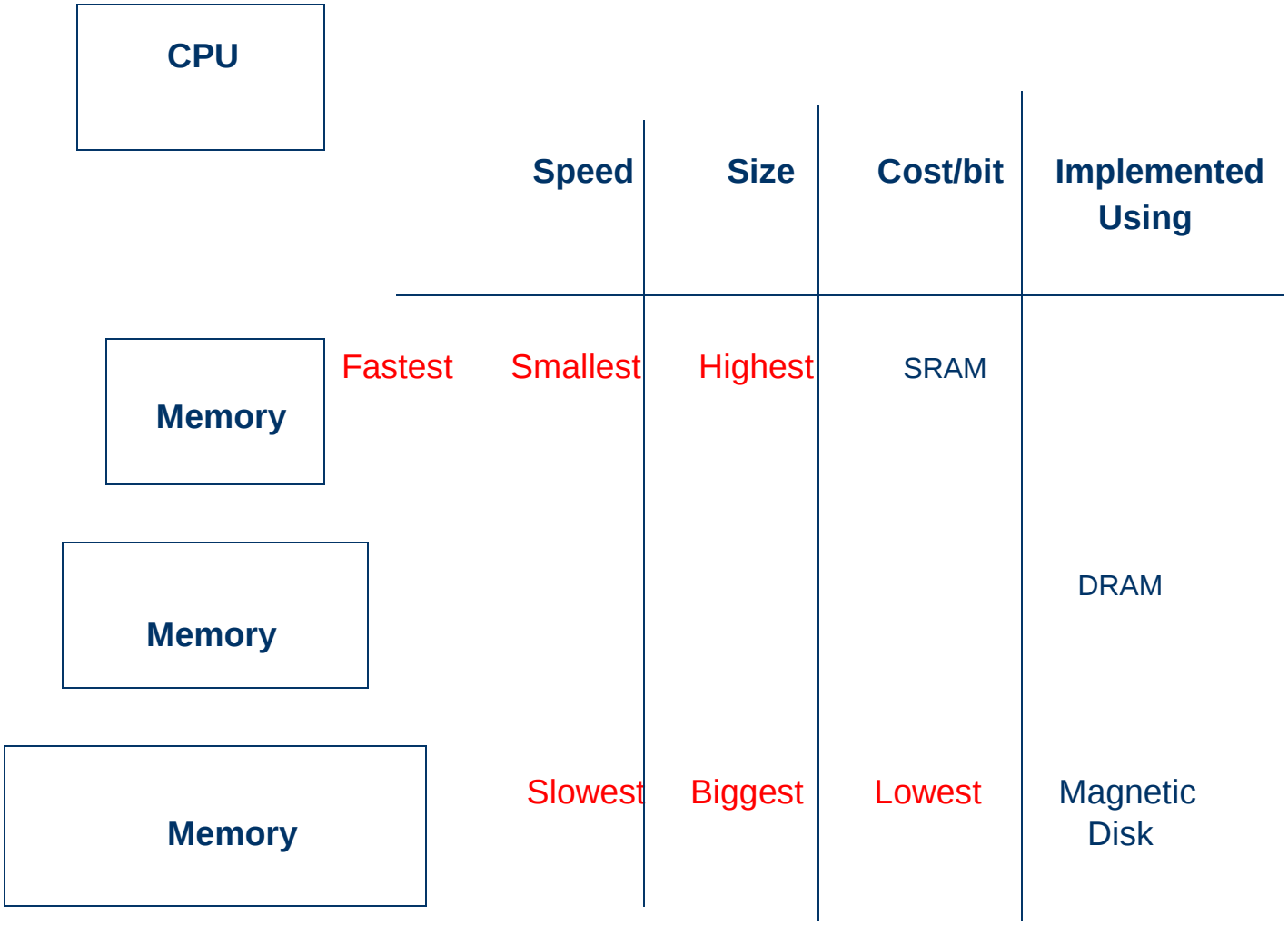
MEMORY HIERARCHY

- Consists of multiple levels of memory with different **speeds and sizes.**

- Goal is to provide the user with **much memory at a low cost,** while **providing access at the speed offered by the fastest memory.**

MEMORY HIERARCHY (Continued)

Memory hierarchy consists of multiple levels but data is copied between only two adjacent levels at a time.
1.Upper level
2.Lower level



Memory hierarchy in a computer

CACHE MEMORY

Cache represents the level of memory hierarchy between the main memory and the CPU.

→ upper level of the memory

Terms associated with cache

Hit: The item/data requested by the processor **is found** in some block in the cache.

Miss: The item requested by the processor **is not found** in the cache.

Terms associated with cache (Continued)

Hit rate: The fraction of the memory access **found in the cache.**
Used as a measure of performance of the cache.

Hit rate = (Number of hits) / Number of access
= (Number of hits) / (# hits + # misses)

Miss rate: The fraction of memory access **not found in cache.**

Miss rate = 1.0 - Hit rate

Terms associated with cache (Continued)

Hit time: **Time to access the cache memory**

Includes the time needed to determine whether the access is a hit or miss.

Miss penalty:

Time to replace a cache block with the corresponding block from the memory + the time to deliver this block to the processor

Cache Organizations

Three types of cache organizations available

- Direct-mapped cache
- Set associative cache
- Fully associative cache

DIRECT MAPPED CACHE

Each main memory block is directly mapped to exactly one location in the cache. (It is assumed for right now that 1 block = 1word)

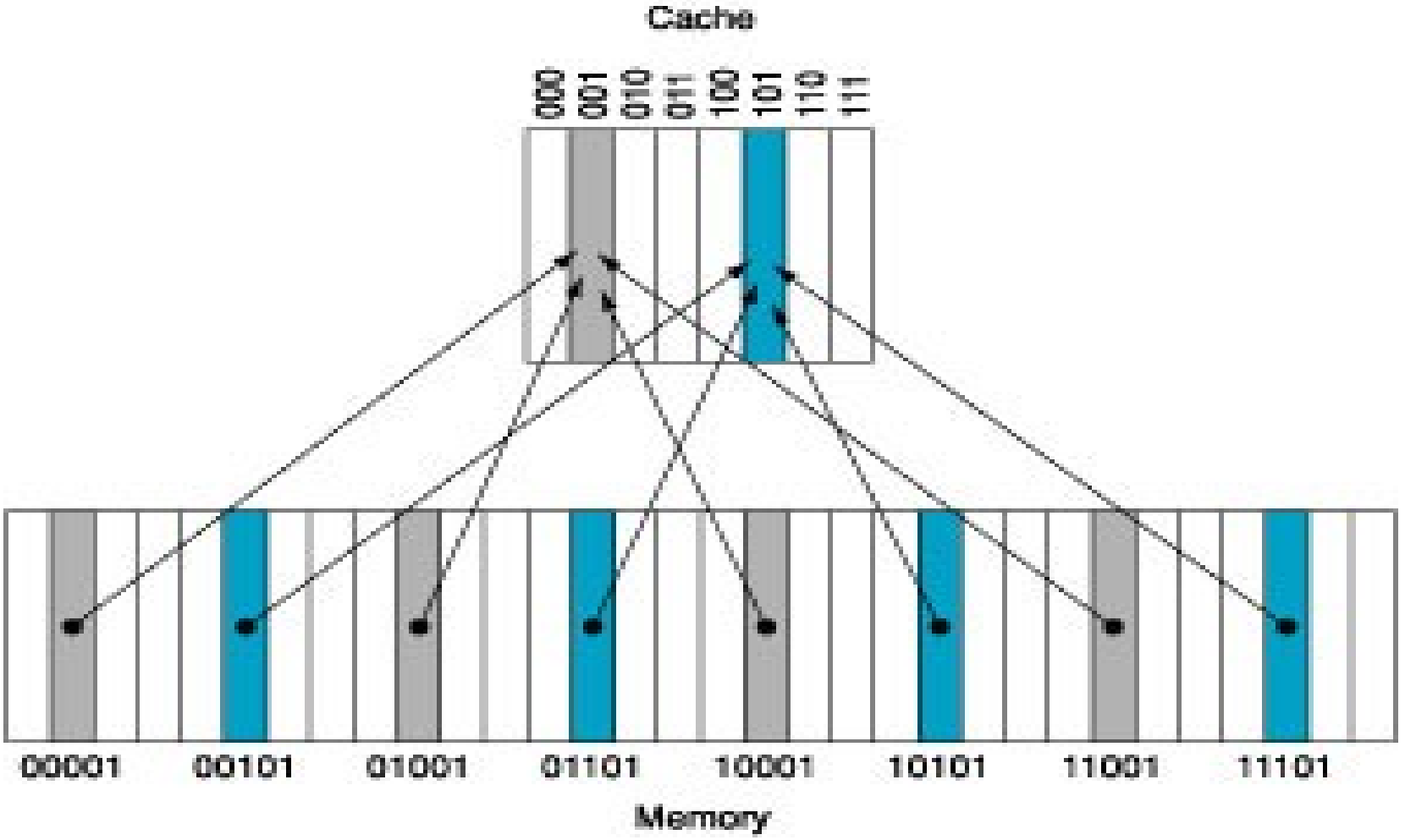
For each block in the main memory, a corresponding cache location is assigned based on the address of the block in the main memory.

Mapping used in a direct-mapped cache:

Cache index = (Memory block address) *modulo* (Number of blocks in the cache)

Example of a Direct-Mapped Cache

Figure 7.5 A direct-mapped cache of 8 blocks



Accessing a Cache Location and Identifying a Hit

We need to know

1. Whether a cache block has valid information
   - done using *valid bit*

and

2. Whether the cache block corresponds to the requested word
   - done using *tags*

CONTENTS OF A CACHE MEMORY BLOCK

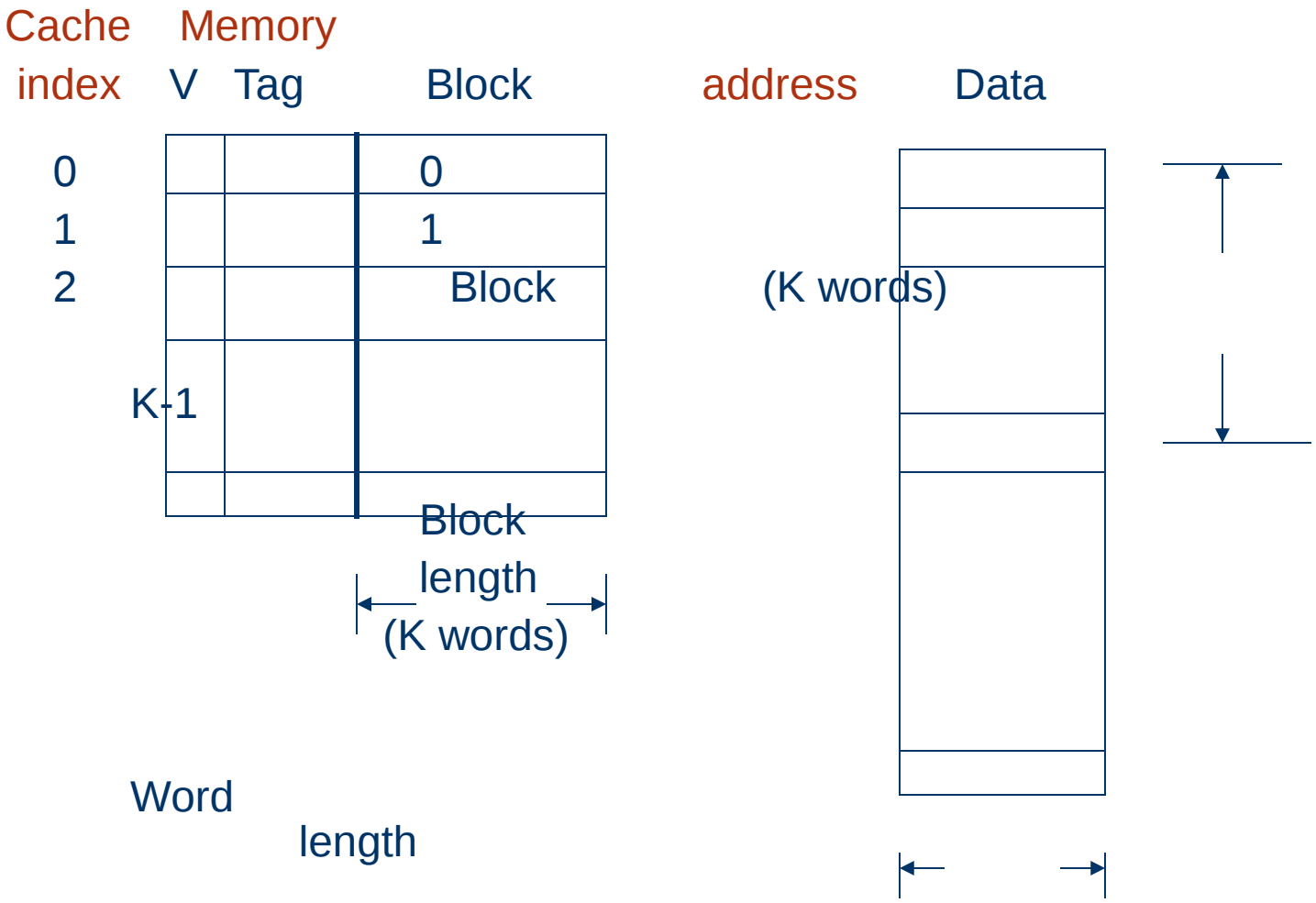
A cache memory block consists of the **data bits, tag bits and a valid (V)bit.**

The index of a cache block and the tag contents of that block uniquely specify the memory address of the word contained in the cache block.

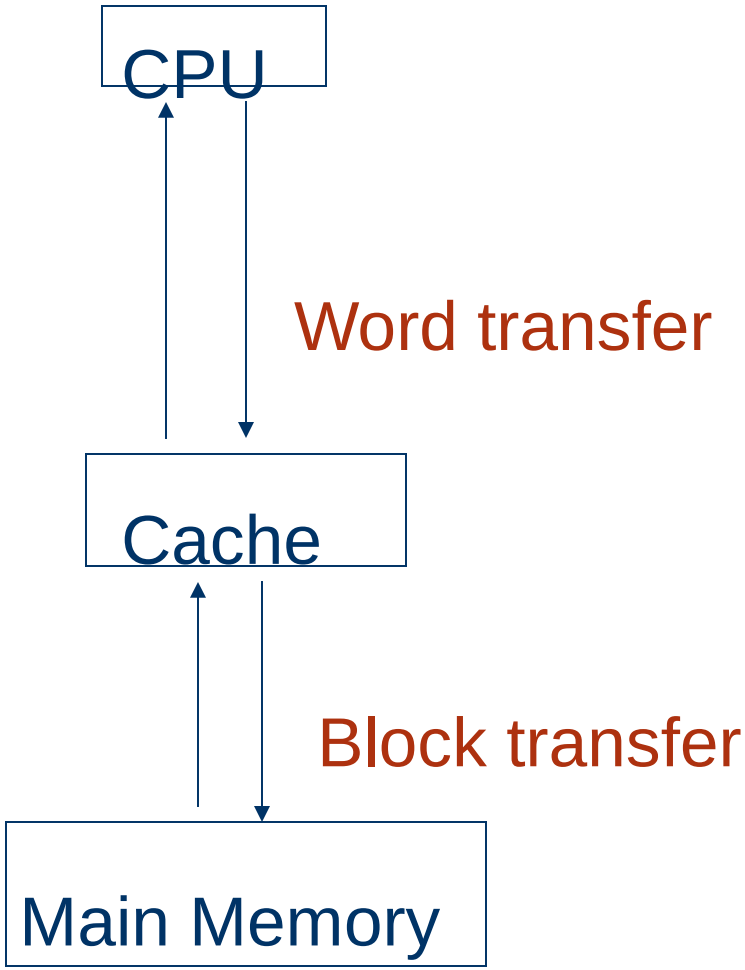
V bit is set *only if* the cache block has valid information.

Cache index	V	Tag	Data

CACHE AND MAIN MEMORY STRUCTURE



CACHE CONCEPT



A Cache Example

Q. The series of memory address references given as word addresses are 22, 26, 22, 26, 16, 3, 16, and 18. Assume a direct-mapped cache with 8 one-word blocks that is initially empty. Label each reference in the list as a hit or miss and show the contents of the cache after each reference.

Answer:

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss (7.6b)	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	miss (7.6c)	$(11010_{two} \bmod 8) = 010_{two}$
22	10110_{two}	hit	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	hit	$(11010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	miss (7.6d)	$(10000_{two} \bmod 8) = 000_{two}$
3	00011_{two}	miss (7.6e)	$(00011_{two} \bmod 8) = 011_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$
18	10010_{two}	miss (7.6f)	$(10010_{two} \bmod 8) = 010_{two}$

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

a. The initial state of the cache after power-on

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory(10110_{two})
111	N		

b. After handling a miss of address (10110_{two})

Index	V	Tag	Dlata
000	N		
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memorlly (10110_{two})
111	N		

c. After handling a miss of address (11010_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

d. After handling a miss of address (10000_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

e. After handling a miss of address (00011_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	10_{two}	Memory (10010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

f. After handling a miss of address (10010_{two})

Cache example (2)

- If each instruction 2^n blocks,
- Each block contains 2^m words
- So, tag field size = 32 – (n+m+2)
- Total number of bits in a direct mapped cache,
- = 2^n *(block size + tag size + valid field size)

How many total bits are required for a direct-mapped cache with 16 KB of data and 4-word blocks, assuming a 32-bit address?

We know that 16 KB is 4K words, which is 2^{12} words, and, with a block size of 4 words (2^2), 2^{10} blocks. Each block has 4 × 32 or 128 bits of data plus a tag, which is 32 – 10 – 2 – 2 bits, plus a valid bit. Thus, the total cache size is

$$2^{10} \times (128 + (32 - 10 - 2 - 2) + 1) = 2^{10} \times 147 = 147 \text{ Kbits}$$

or 18.4 KB for a 16 KB cache. For this cache, the total number of bits in the cache is about 1.15 times as many as needed just for the storage of the data.

HANDLING CACHE MISSES FOR INSTRUCTIONS

- Decrement PC by 4
- Fetch the block containing the missed instruction from the main memory
- Write the block into the cache
  - Write the instruction block into the data portion of the referenced cache block
  - Copy the upper bits of the referenced memory address into the tag field of the cache memory
  - Turn the valid (V) bit on
- Restart the fetch cycle - this will refetch the instruction, this time finding it in cache

HANDLING CACHE MISSES FOR DATA

- Read the block containing the missed data from the main memory
- Write the block into the cache
  - Write the data into the data portion of the referenced cache block
  - Copy the upper bits of the referenced memory address into the tag field of the cache memory
  - Turn the valid (V) bit on

- The requested word may be forwarded immediately to the processor as the block is being updated

or

- The requested word may be delayed until the entire block has been stored in the cache

CHOOSING WHICH BLOCK TO REPLACE

Direct-mapped Cache

When a miss occurs, the requested block can go in exactly one position. So the block occupying that position must be replaced.

Set associative or fully associative Cache

When a miss occurs we have a choice of where to place the requested block, and therefore a choice of which block to replace.

- Set associative cache:

  All the blocks in the selected set are candidates for replacement.

- Fully associative cache:

  All blocks in the cache are candidates for replacement.

VIRTUAL MEMORY

Virtual memory permits each process to use the main memory as if it were the only user, and to extend the apparent size of accessible memory beyond its actual physical size**.(Advantages)**

Virtual memory does not exist in physical memory. The virtual address generated by the CPU is translated into a physical address, which in turn can be used to access the main memory. The process of translating the virtual address into a physical address is called **memory mapping or address translation.**

Page: A virtual memory block

Page fault: A virtual memory miss

Figure 7.20 The virtual addressed memory with pages mapped to the main memory

Figure7.21 Mapping from a virtual to a physical address

Figure 7.19 The virtual addressed memory with pages mapped to the main memory

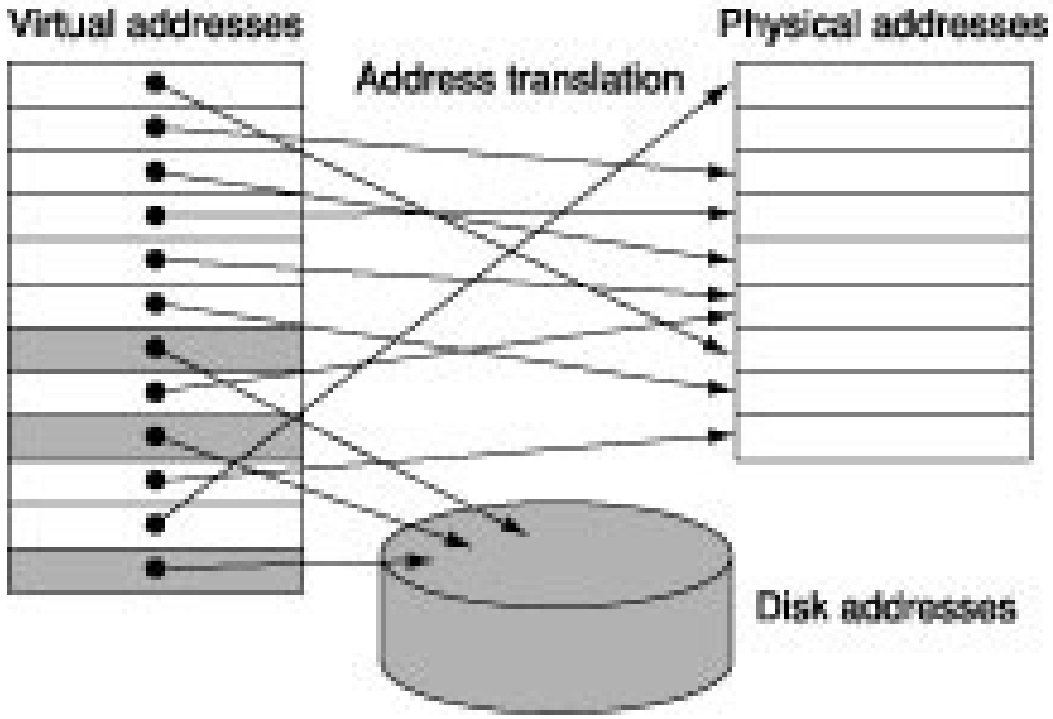
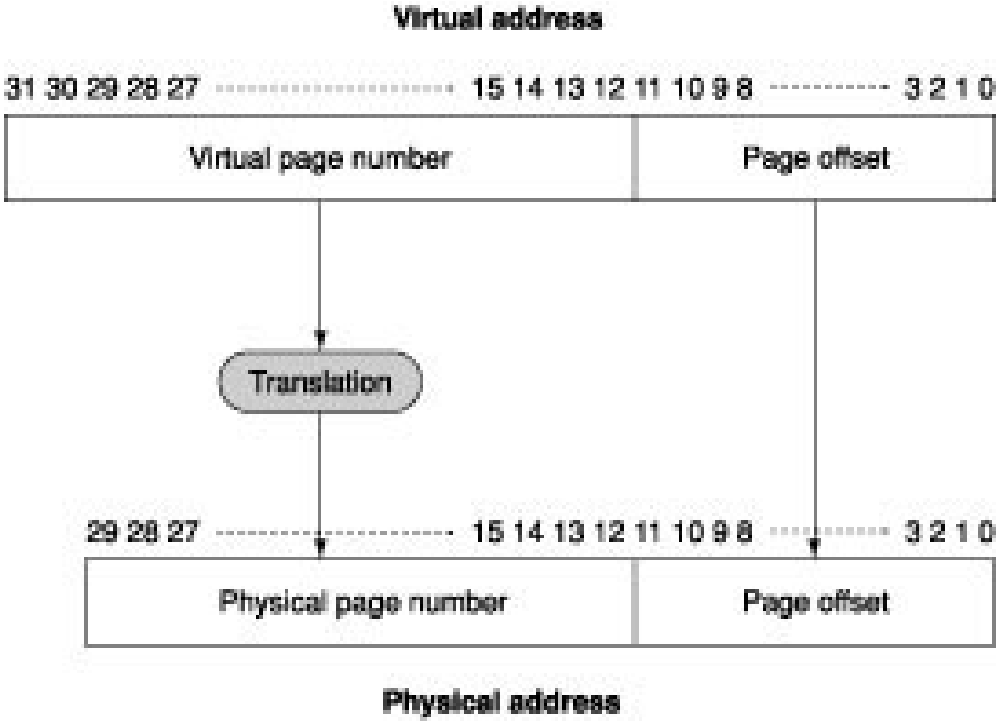


Figure7.20 Mapping from a virtual to a physical address



PLACING A PAGE AND FINDING IT AGAIN

Operating system must maintain a **page table.**

Page Table

- **Maps virtual pages to physical pages or else to locations in the secondary memory**
- Resides in memory
- Indexed with the page number from the virtual address

Each program has its own page table, which maps the virtual address space of that program to main memory.

No tags are required in the page table because the page table contains a mapping for every possible virtual page.

Page table register

- **Indicates the location of the page table in the memory**
- **Points to the start of the page table.**

PAGE FAULTS

If the valid bit for a virtual page is off, a page fault occurs.

- Operating system is given control at this point (exception mechanism)
- OS finds the page in the next level of the hierarchy (magnetic disc for example)
- OS decide where to place the requested page in main memory

OS also creates a data structure that tracks which processes and which virtual addresses use each physical page.

When a page fault occurs, if all the pages in the main memory are in use, the **OS has to choose a page to replace**. The algorithm usually employed is **the LRU replacement algorithm**.

WRITES (self)

In a virtual memory system, writes to the disk take hundreds of thousands of cycles. Hence write-through is impractical. The strategy employed is called write-back (copy back).

Write-back technique

- Individual writes are accumulated into a page
- When the page is replaced in the memory, it is copied back into the disk.

MAKING ADDRESS TRANSLATION FAST: THE TRANSLATION-LOOKASIDE BUFFER (TLB)

If a CPU has to access a page table resident in the memory to translate every memory access, the virtual memory would have too much overhead. Instead a TLB cache can be used to implement the page table.

Figure 7.24 TLB acts as a cache for page table references

Figure 7.23 TLB acts as a cache for page table references