Topic #7

Memory Organization

Major Goals

- To introduce the **memory hierarchy** and the **principle of locality**.
- To introduce basic cache concepts and **cache organizations**.
- To introduce basic concepts of **virtual memory** and its implementation.
- To explain memory hierarchy design challenges in modern processors.

Memory vs. Processor Improvements

- Memory improvement has not kept up with the improvement of processors -> **processors are much faster than memories**.

- It is impractical and not economical to include a large amount of fast memory inside the processor.

- As a result, **memory access can be a bottleneck** -> the processor has to wait for memory access for every instruction.

- **Solution**: Using a **memory hierarchy** that exploits the **principle of locality**.

Principle of Locality

- Programs usually access a relatively small portion of their address space (for instructions or data) at any instant of time.

- **Two types of locality**:
  - **Temporal locality**:
    - If an instruction or data item is referenced, it will tend to be referenced again soon.
    - **Loops in programs** contribute to temporal locality.
  - **Spatial locality**:
    - If an instruction or data item is referenced, items whose addresses are close by will tend to be referenced soon.
    - **Sequential statements in programs** and **data arrays** contribute to spatial locality.

Example

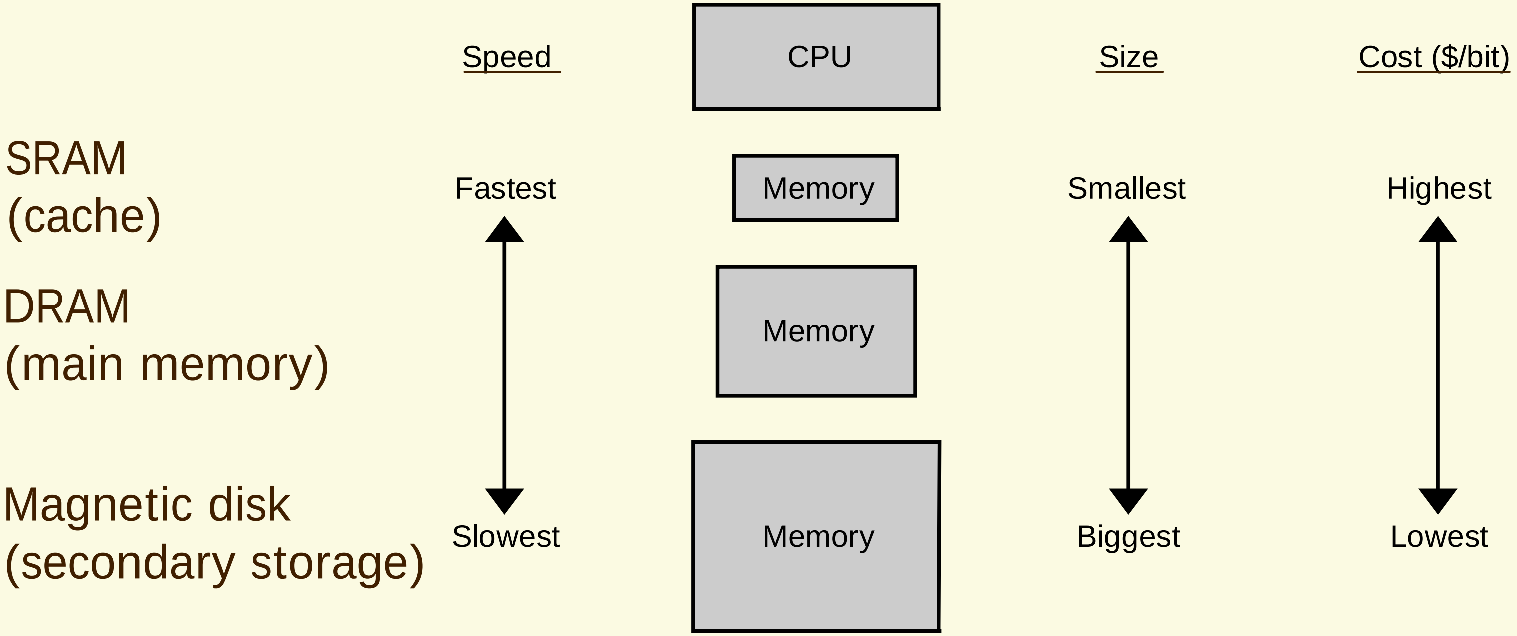
- **Summing up 100 values stored in memory**:

```
        addi   $s0, $zero, 0      # $s0: accumulator
        addi   $s1, $zero, 100    # $s1: counter
  L1:   lw     $t1, 0($s2)        # $s2: memory addr
        add    $s0, $s0, $t1
        addi   $s2, $s2, 4
        subi   $s1, $s1, 1
        bne    $s1, $zero, L1
```

- **Temporal and spatial locality** can be observed in instructions and data.

Memory Hierarchy

- A **memory hierarchy** consists of multiple levels of memory. The user has the **illusion** of a memory that is as large as the largest level.



Speed

CPU

Size

Cost ($/bit)

SRAM (cache)

Fastest

Memory

Smallest

Highest

DRAM (main memory)

Memory

Magnetic disk (secondary storage)

Slowest

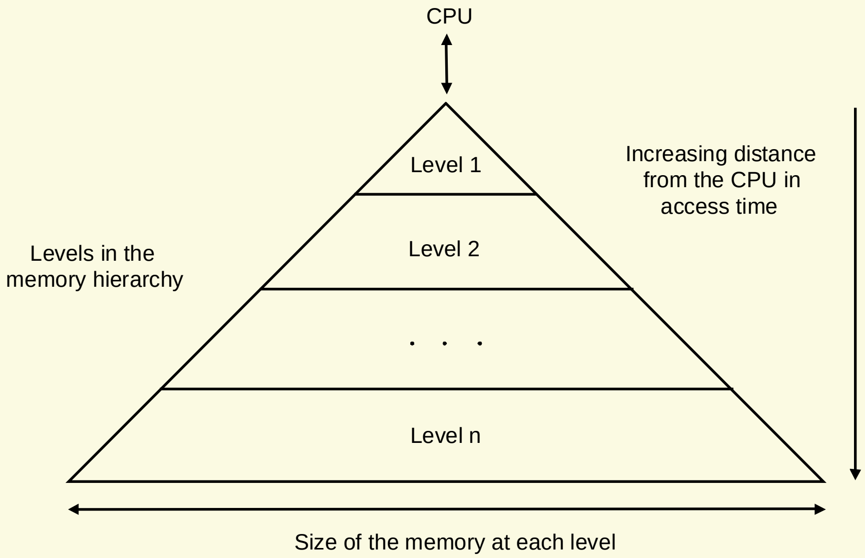
Memory

Biggest

Lowest

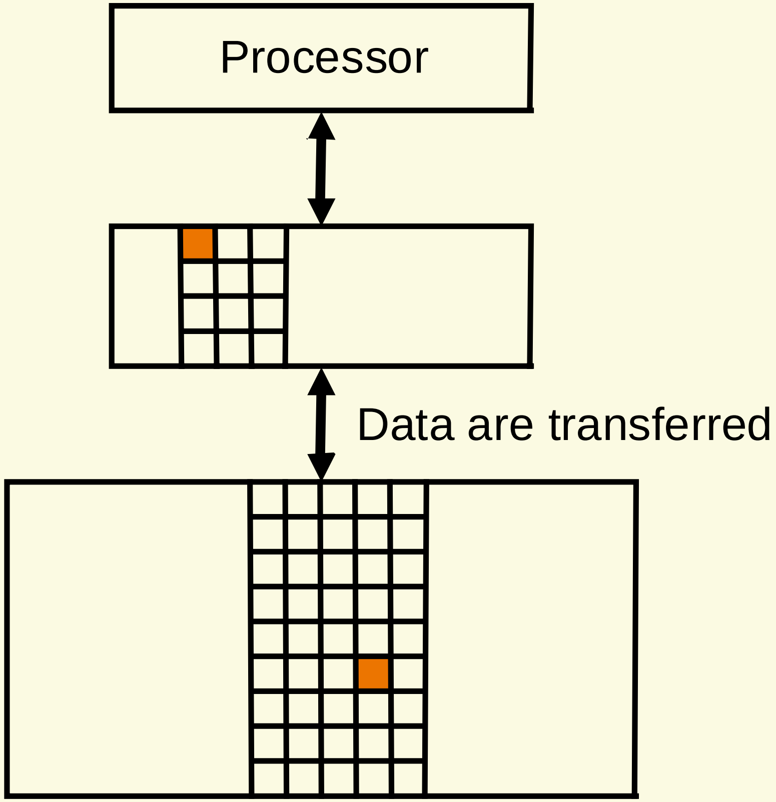
Memory Hierarchy

- With the appropriate operating mechanisms, the processor can have an access time that is determined primarily by level 1 (fastest level) of the hierarchy and yet can have a memory as large as level n (largest level).


CPU
Level 1
Level 2
. . .
Level n
Levels in the memory hierarchy
Increasing distance from the CPU in access time
Size of the memory at each level

- **Memory hierarchy** and **principle of locality**:
  - Memory hierarchy takes advantage of **temporal locality** by keeping more recently accessed data items closer to the processor.
  - Memory hierarchy takes advantage of **spatial locality** by moving blocks consisting of multiple contiguous words in memory to upper levels of the hierarchy.

Data Transfer Between Levels



- A level closer to the processor is a subset of any level further away.
- Data are copied in **blocks** between only two adjacent levels at a time.
- **Hit**: when the data item requested by the processor appears in some block in the upper level.
- **Miss**: when the data item requested by the processor is not in the upper level -> data transfer occurs from the lower level to the upper level.

Performance Measures

- **Hit rate** (or **hit ratio**): fraction of memory accesses found in the upper level

- **Miss rate**: 1 - hit rate

- **Hit time**:

  time to access a data item which is in the upper level

  (= time to determine miss or hit + time to access data item)

- **Miss penalty**:

  time to replace a block in the upper level with the corresponding block from the lower level + time to deliver the block to the processor

- Hit time << miss penalty

Caches

- **Cache** was the name chosen to represent the level of the memory hierarchy between the CPU and the main memory in the first commercial machine.

- Today, although this remains the dominant use of the word cache, the term is also used to refer to any storage managed to take advantage of locality of access.

- **Cache inside processor**:
  - Takes advantage of **temporal locality** by storing instructions and data recently fetched from the main memory.
  - Takes advantage of **spatial locality** by fetching instructions and data from the main memory in blocks.

Issues to Consider

- **Block placement**:
  - Where is a block placed in the cache?
- **Block identification**:
  - How can a block be found if it is in the cache?
- **Block replacement**:
  - When a miss occurs, how can a block in the cache be selected for replacement?
- **Write strategy**:
  - When a write occurs, is the information written only to the cache?

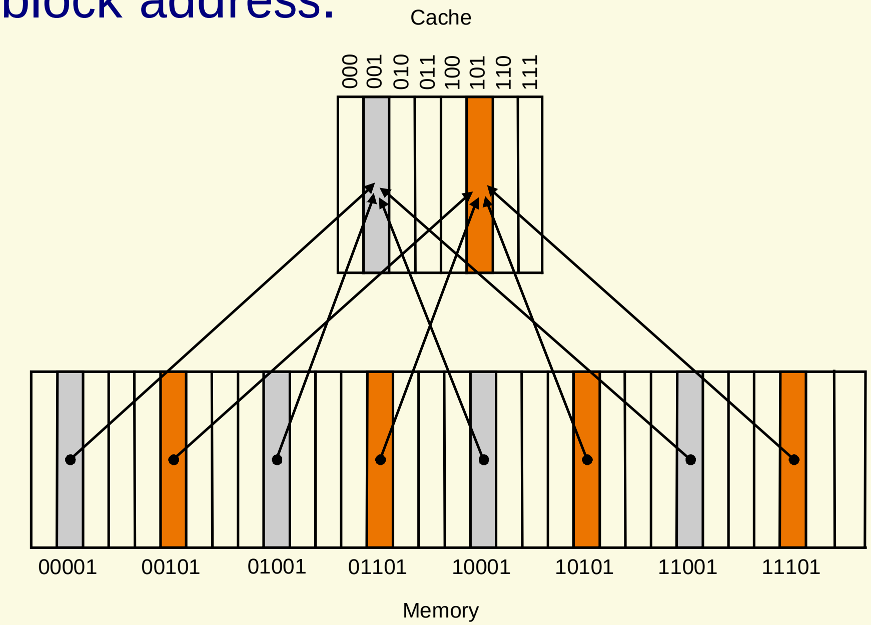
A Simple Block Placement Scheme: Direct Mapped

- ❑ Each memory location is mapped to one location in the cache.

- ❑ **A common mapping strategy**:

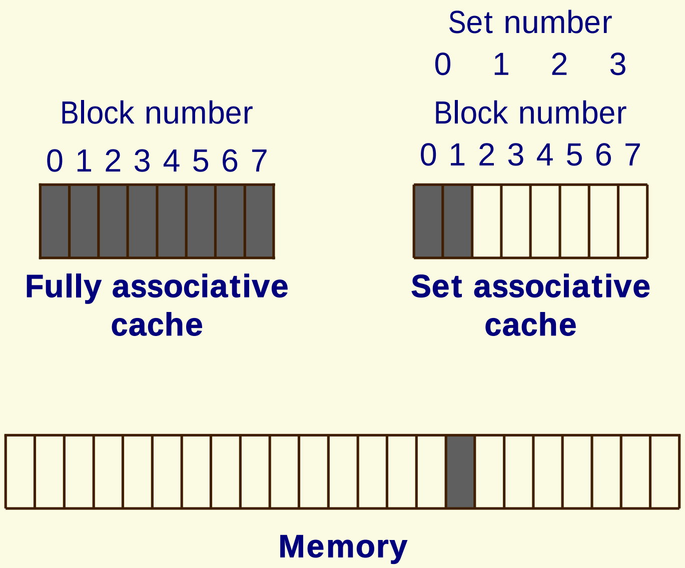
  cache_location = block_address MOD number_of_blocks_in_cache

  - ❍ If the number of cache blocks (N) is a power of 2, then MOD can be computed simply by using only the low-order $\log_2 N$ bits of the block address.



Other Block Placement Schemes

- Disadvantage of direct mapped scheme: Blocks with the same location in the cache cannot be present simultaneously.



- **Fully associative**:
  - Each block can be placed anywhere in the cache.
  - Disadvantage: quite costly (hardware and time) to search for a block in the cache.

- **Set associative**:
  - Each block can be placed in a certain number of locations in the cache.
  - A good **compromise** between direct mapped and fully associative schemes.

Set Associative Schemes

- An **N-way set associative cache** consists of a number of sets, each of which consists of N blocks.

- **Mapping strategy**:

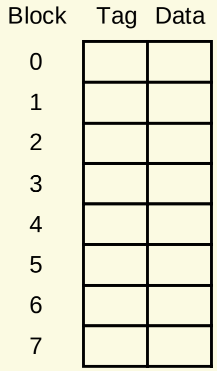
  cache_location = block_address MOD number_of_sets_in_cache

- **Special cases**:
  - A **direct mapped cache** can be considered as a one-way set associative cache.
  - A **fully associative cache** with M blocks can be considered as an M-way set associative cache.

Possible Associativity Structures

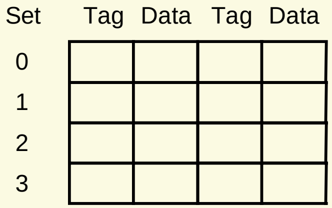
- **Increase** in degree of associativity =>
  - **Decrease** in miss rate (advantage)
  - **Increase** in hit time (disadvantage)

One-way set associative
(direct mapped)



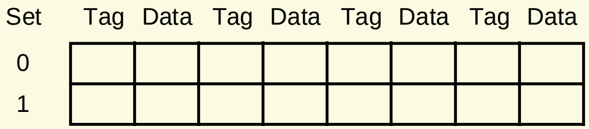
Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative



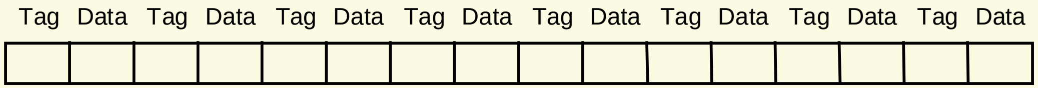
Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative



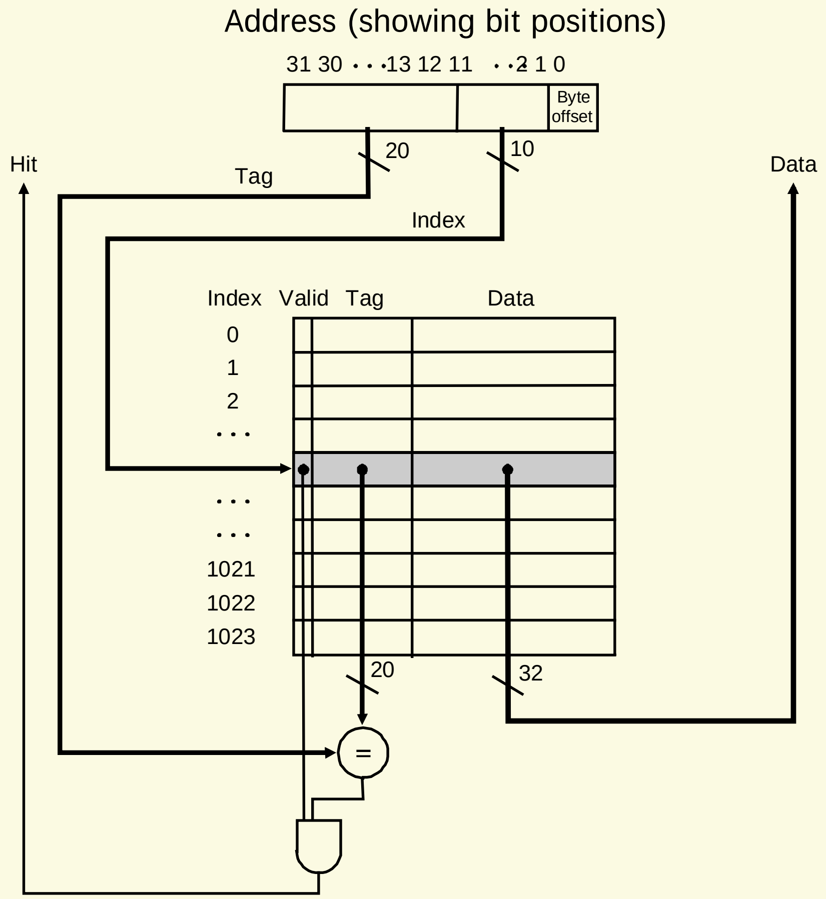
Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)



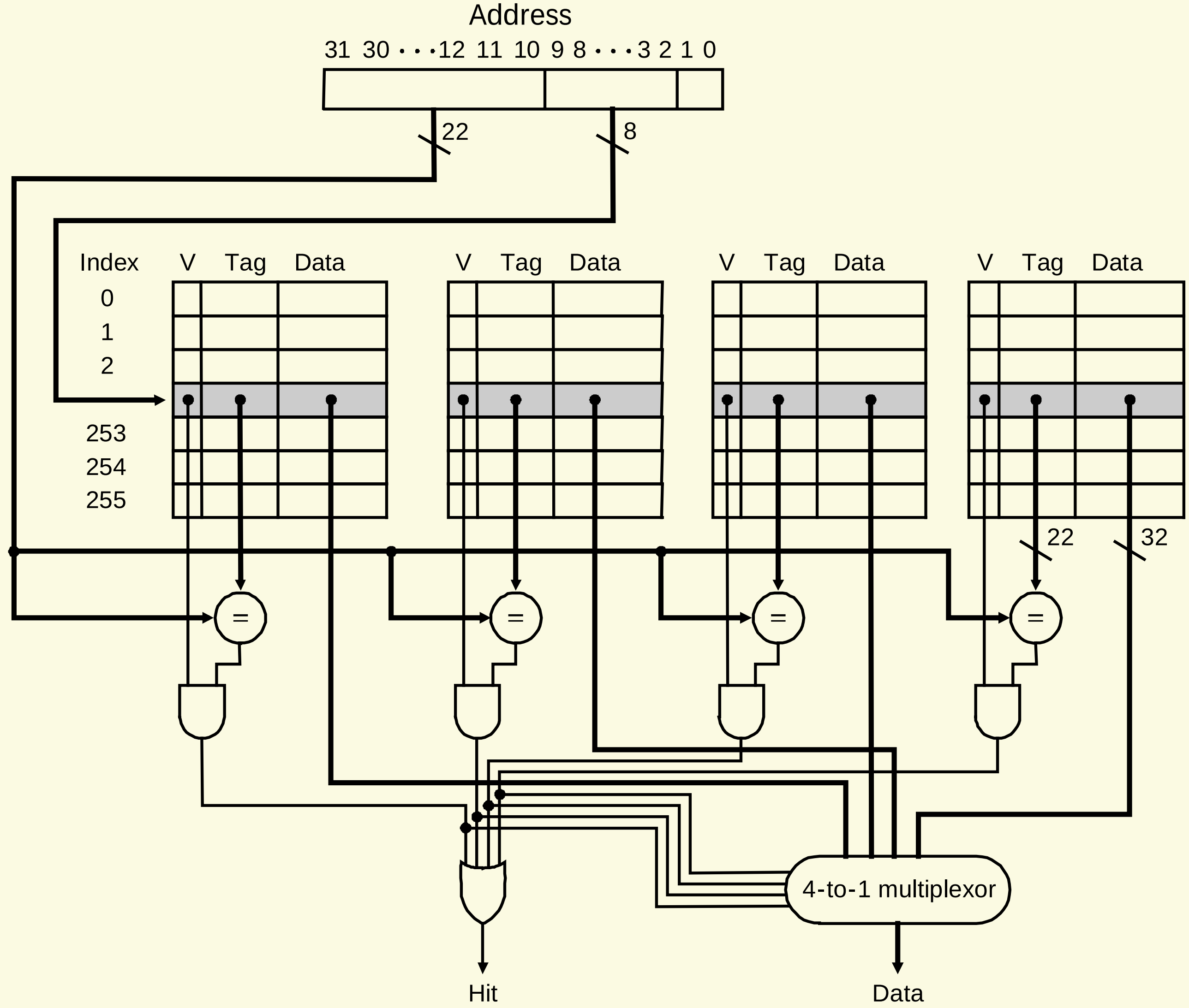
Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Block Identification



- ❑ Each cache location can contain a block from a number of different memory locations.
- ❑ A **tag** is used to store the address information. The tag needs only to contain those high-order bits that are not used as an index into the cache.
- ❑ A **valid bit** is needed to indicate whether a cache block contains valid information.

Block Identification in N-Way Set Associative Cache as Parallel Search



Block Replacement

- When a miss occurs in a cache, we must decide which block to replace.
  - **Direct mapped**: only one candidate (trivial case)
  - **Fully associative**: all blocks are candidates
  - **Set associative**: only blocks within a particular set are candidates

- **Two primary replacement strategies for associative caches**:
  - **Random**: To spread allocation uniformly, candidate blocks are randomly selected, possibly with hardware assistance.
  - **Least recently used** (LRU): The block replaced is the one that has not been used for the longest time.
    - Can be costly to implement for a degree of associativity higher than 2 or 4.

Write-Through Strategy

- The information is written to both the block in the cache and to the block in the main memory.

- **Advantages**:
  - Misses are simpler and cheaper because they do not require a block to be written back to the main memory.
  - Write-through is easier to implement than write-back, although a write buffer is needed for a high-speed system.

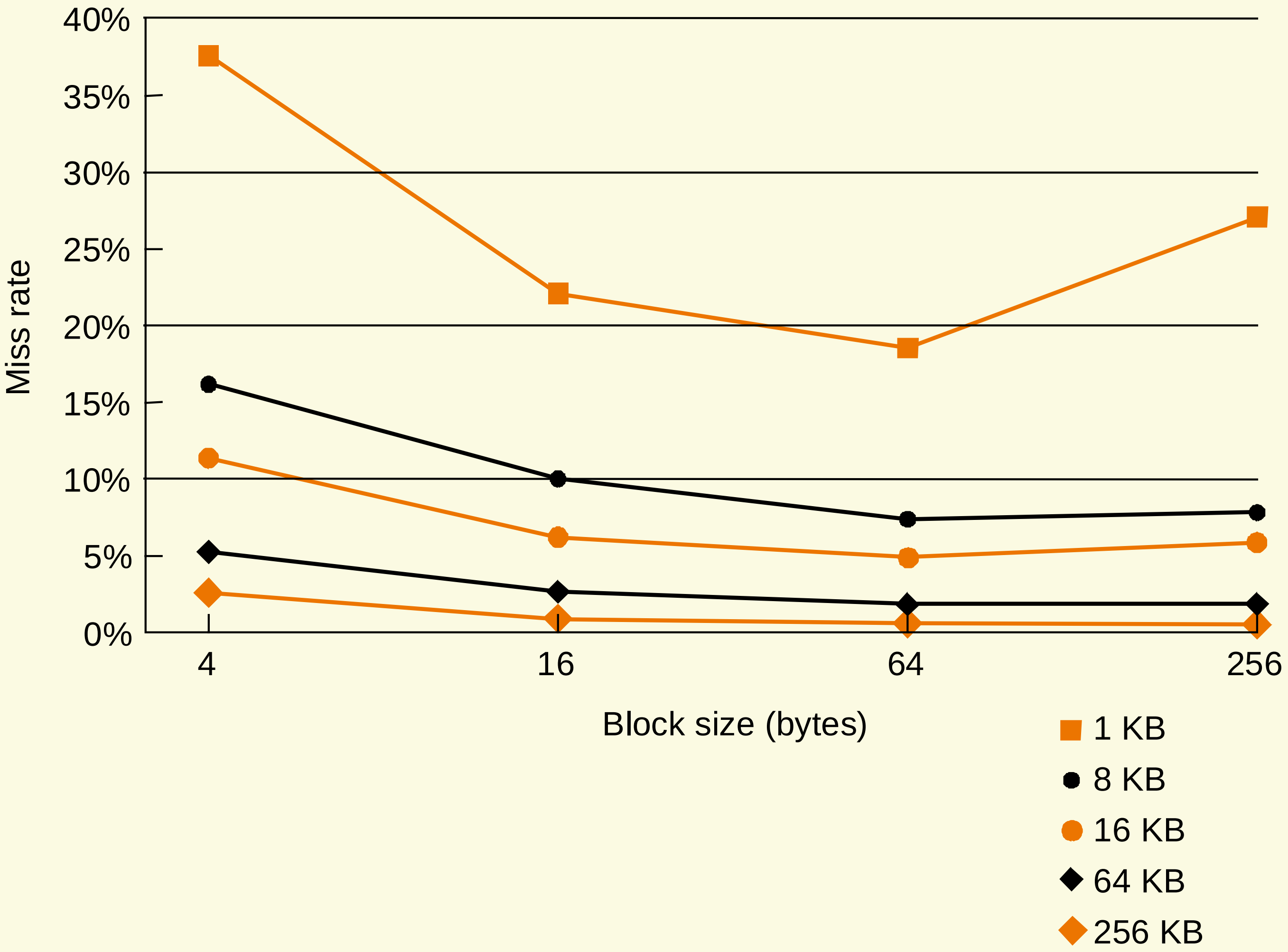
Write-Back Strategy

- The information is written only to the block in the cache. The modified block is written to the main memory only when it is replaced.

- **Advantages**:
  - Individual words can be written by the processor at the rate that the cache, rather than the memory, can accept them.
  - Multiple writes within a block require only one write to the main memory.
  - When blocks are written back, the system can make effective use of a high bandwidth transfer since the entire block is written.

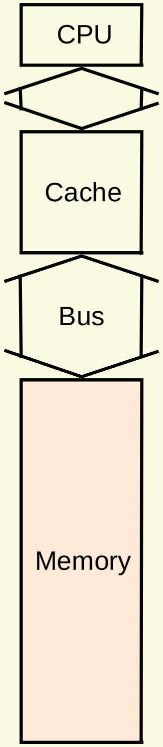
- As CPU performance increases at a rate faster than DRAM-based main memory, more and more caches use the write-back strategy.

Miss Rate vs. Block Size

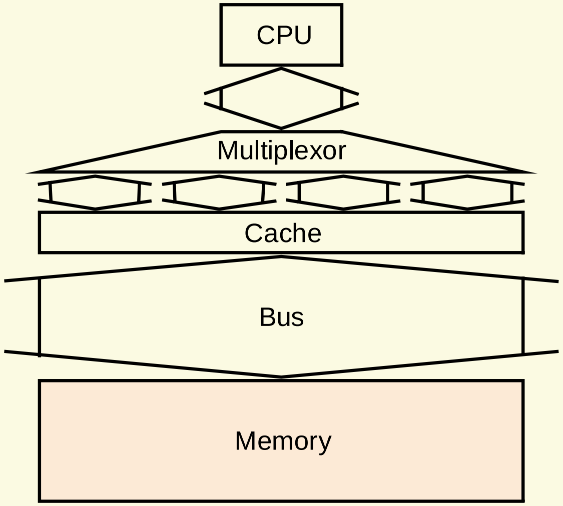


Increasing Memory Bandwidth

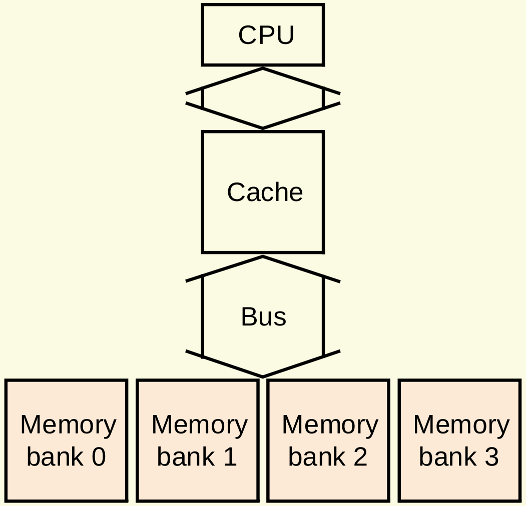
- Either the **physical** or **logical** width can be increased.



One-word-wide memory organization



Wide memory organization



Interleaved memory organization

Virtual Memory

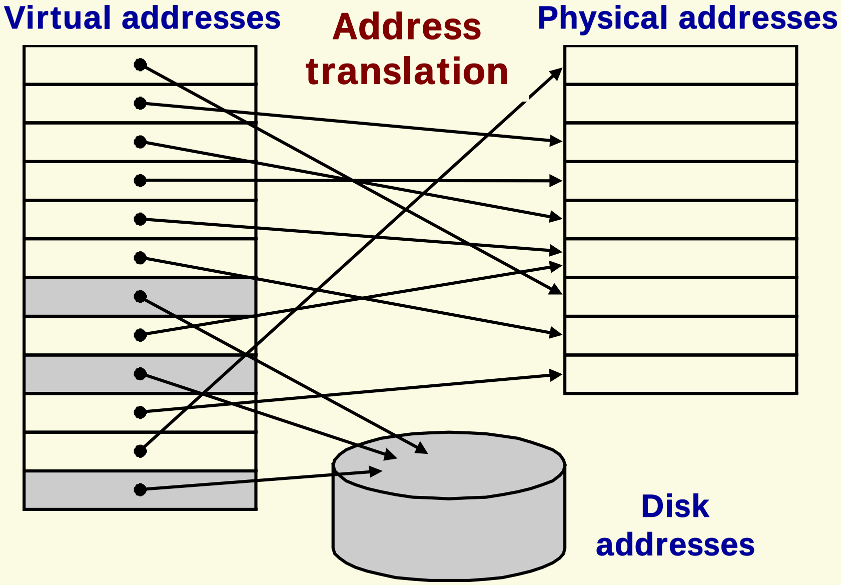
- **Virtual memory** is the technique that makes use of the main memory as a "cache" for magnetic disks (secondary storage).

- **Motivations**:
  - Sharing of memory between programs:
    - The total memory required by all programs running on a machine may be much larger than the amount of main memory available.
    - Only a fraction of the memory is being actively used.
  - Allowing a program to exceed the size of the main memory:
    - Automatic mapping of virtual addresses to physical addresses can relieve the otherwise substantial burden of the programmers.

Terminology

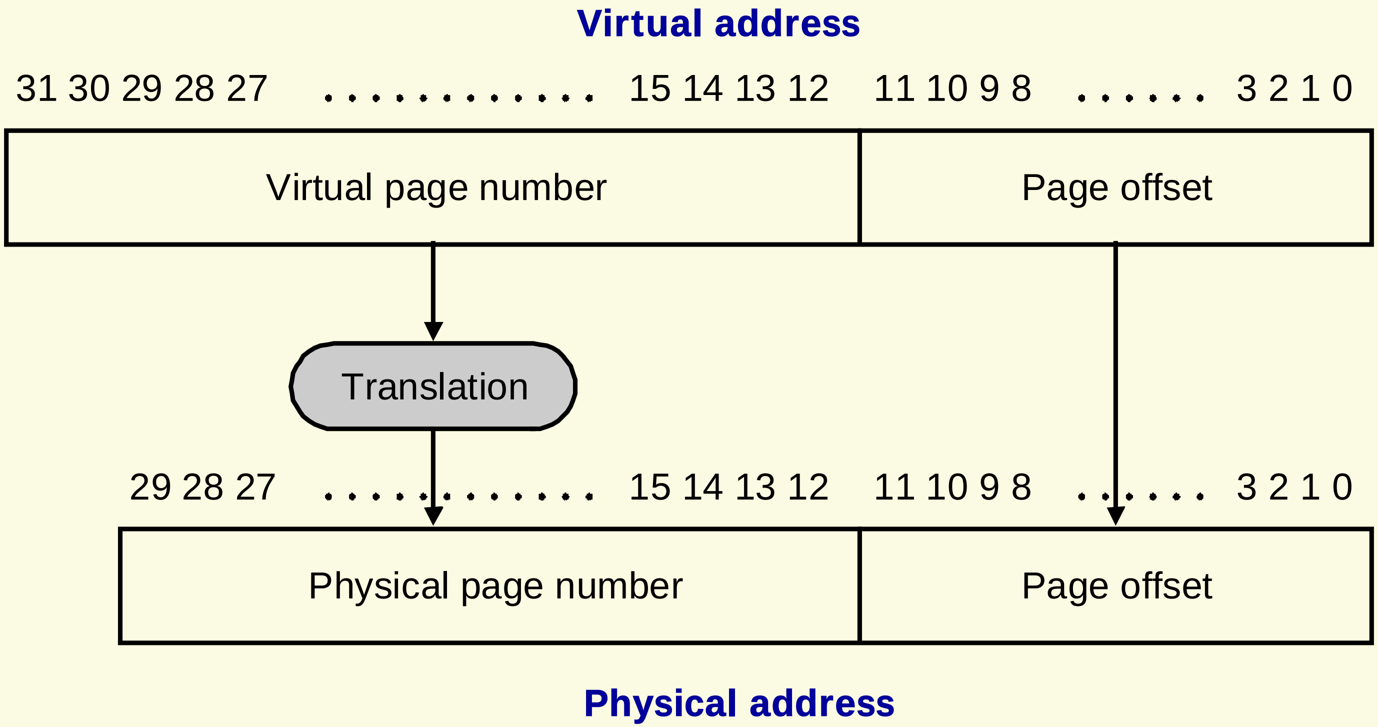
- Although the concepts at work in virtual memory and in caches are the same, their differing historical roots have led to the use of different terminology.
  - **Page** in virtual memory (cf. **block** in cache)
  - **Page fault** in virtual memory (cf. **miss** in cache)
- The processor generates virtual addresses while the memory is accessed using physical addresses. Sharing of physical pages is necessary.

- **Address translation or memory mapping**:



Address Translation Example

- Virtual address space: 4 GB
- Maximum main memory size: 1 GB
- Page size: 4 KB



Page Faults

- A page fault usually has an enormous penalty dominated by the time it takes to get the first word for typical page sizes. It can take millions of clock cycles to process.

- **Design choices resulted**:
  - Pages should be large enough.
  - Flexible page placement (e.g., fully associative scheme) should be used.
  - Page faults can be handled in software because the overhead is small compared to the disk access time. Software can also afford to use clever page placement algorithms.
  - Write-back should be used as write-through would take too long.

Page Placement and Identification

- Fully associative placement is used to reduce the page fault rate.

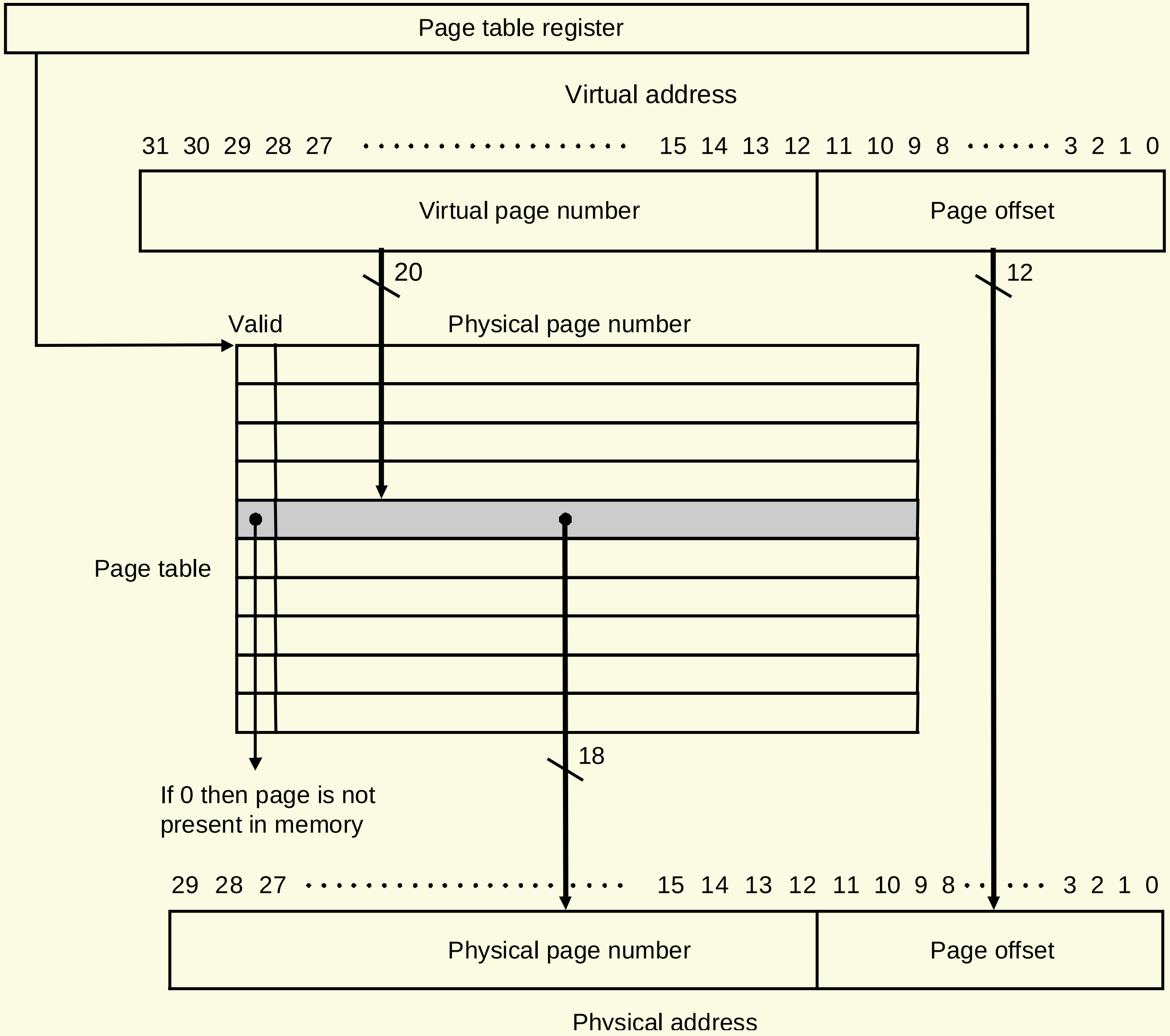
- Mapping of virtual addresses to physical addresses is done through a **page table**, which is a structure that resides in the memory.

- The starting address of the page table is stored in the **page table register**.

- Each page table entry stores a **valid bit** and the corresponding **physical page number**.

- Since every possible virtual page is represented in the page table, there is no need to have a tag field.

Page Placement and Identification



Page Replacement and Write Strategy

- **Page replacement**:
  - The LRU scheme is used.
  - The OS is responsible for keeping track of page usage information. However, a completely accurate LRU scheme would be too expensive to implement as it would require updating a data structure on every memory reference. Instead, only approximate schemes are used in practice.

- **Write strategy**:
  - Only the write-back scheme is practical because of the long latency of a disk write.

Key Concepts to Remember

- Ordinary programs exhibit two different notions of locality: **temporal locality** and **spatial locality**.

- Multilevel memory organizations exploit the **principle of locality** to achieve cost/performance tradeoff.

- Two important levels: **cache** and **virtual memory**

- Data are transferred in **blocks** from the main memory to the cache when **misses** occur; data are transferred in **pages** from the disk to the main memory when **page faults** occur.

Key Concepts to Remember

- **Set associative** placement is a good compromise of the **direct mapped** and **fully associative** placement schemes.

- A **valid bit** is used in both a cache and a **page table** to indicate whether the corresponding entry is filled with valid information.

- Block replacement uses either **random** or **least recently used** (LRU) replacement scheme, while page replacement usually uses the LRU scheme.

- The write strategy for caches is either **write-through** or **write-back**, while virtual memory can only use write-back.