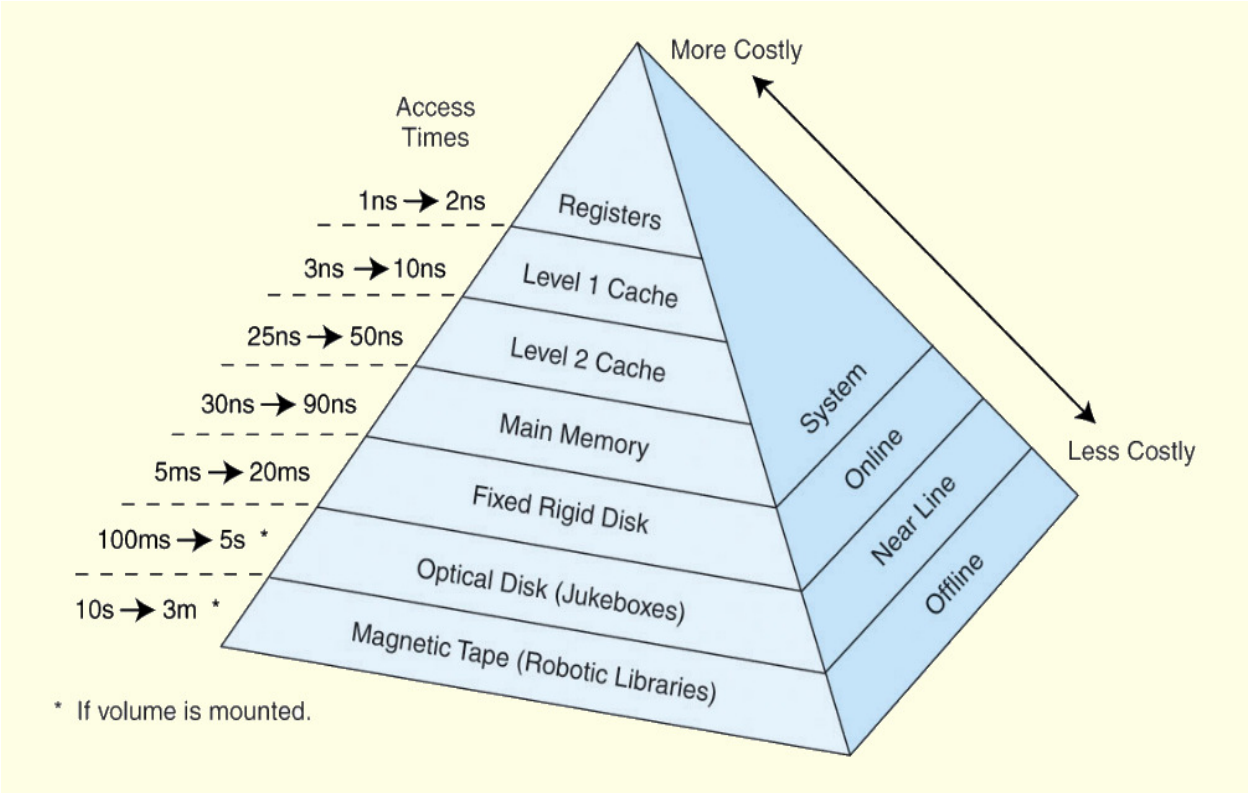
Memory Hierarchy

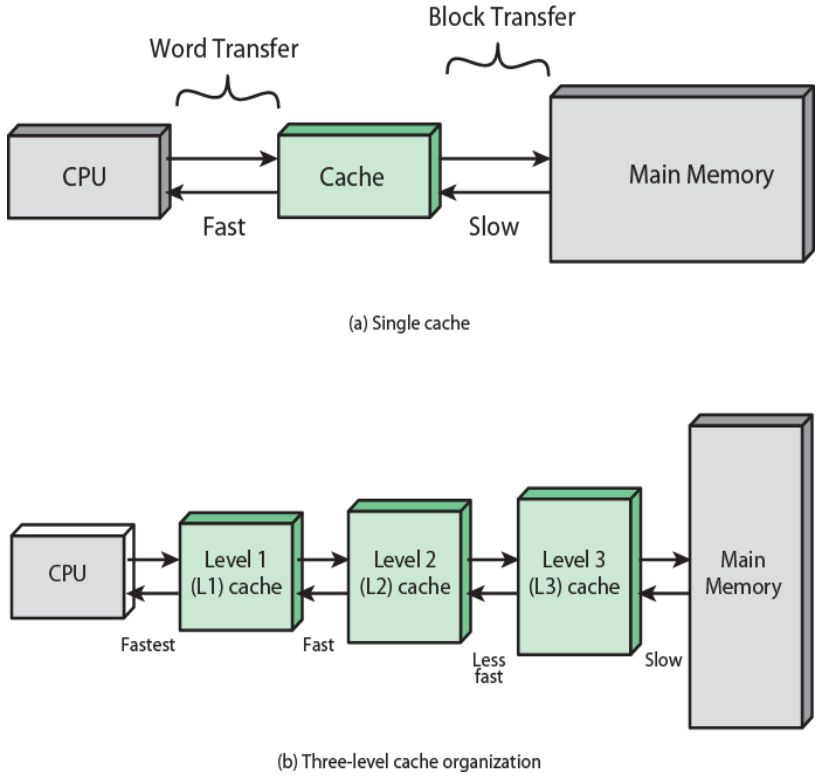
Goal: "Fast", "unlimited" storage at a reasonable cost per bit.



Recall the von Neumann bottleneck - single, relatively slow path between the CPU and main memory.

Typical system view of the memory hierarchy

Figure 4.3 Cache and Main Memory



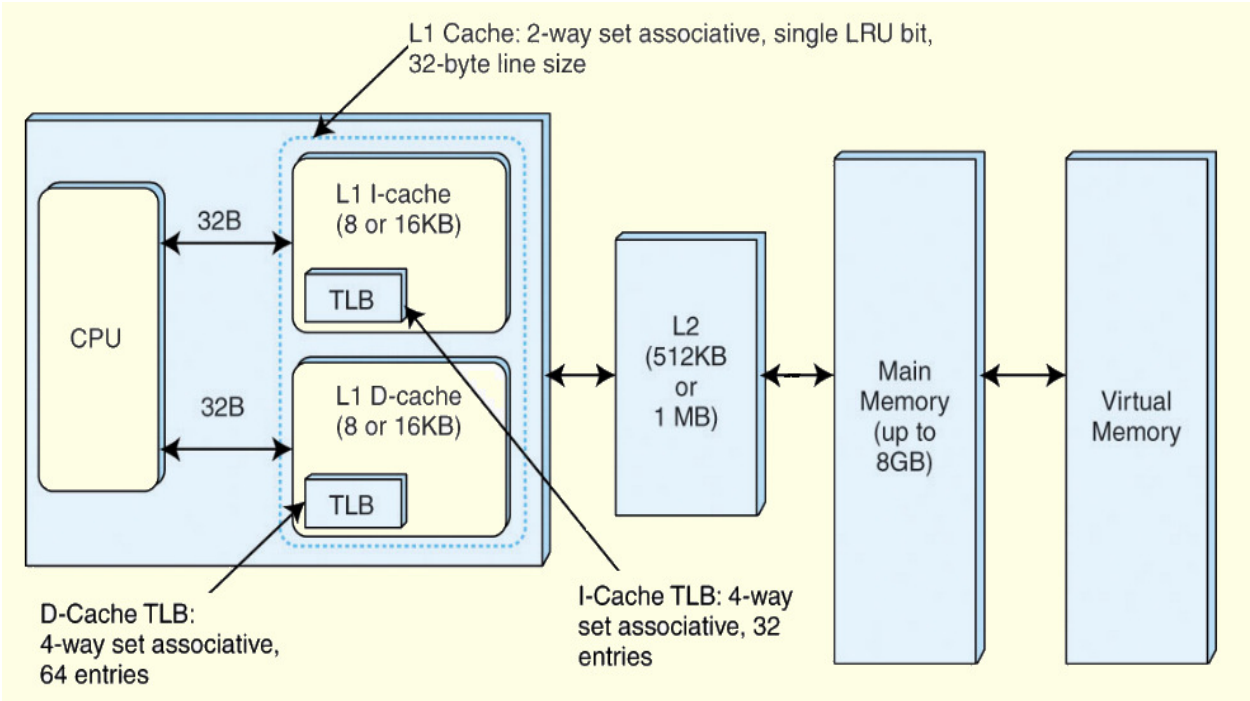
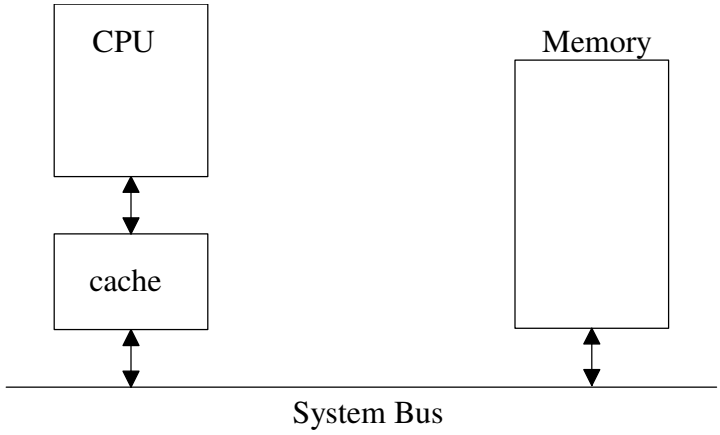


Table 4.3 Cache Sizes of Some Processors

Processor	Type	Year of Introduction	L1 cache	L2 cache	L3 cache
IBM 360/85	Mainframe	1968	16 to 32 KB	—	—
PDP-11/70	Minicomputer	1975	1 KB	—	—
VAX 11/780	Minicomputer	1978	16 KB	—	—
IBM 3033	Mainframe	1978	64 KB	—	—
IBM 3090	Mainframe	1985	128 to 256 KB	—	—
Intel 80486	PC	1989	8 KB	—	—
Pentium	PC	1993	8 KB/8 KB	256 to 512 KB	—
PowerPC 601	PC	1993	32 KB	—	—
PowerPC 620	PC	1996	32 KB/32 KB	—	—
PowerPC G4	PC/server	1999	32 KB/32 KB	256 KB to 1 MB	2 MB
IBM S/390 G4	Mainframe	1997	32 KB	256 KB	2 MB
IBM S/390 G6	Mainframe	1999	256 KB	8 MB	—
Pentium 4	PC/server	2000	8 KB/8 KB	256 KB	—
IBM SP	High-end server/ supercomputer	2000	64 KB/32 KB	8 MB	—
CRAY MTAb	Supercomputer	2000	8 KB	2 MB	—
Itanium	PC/server	2001	16 KB/16 KB	96 KB	4 MB
SGI Origin 2001	High-end server	2001	32 KB/32 KB	4 MB	—
Itanium 2	PC/server	2002	32 KB	256 KB	6 MB
IBM POWER5	High-end server	2003	64 KB	1.9 MB	36 MB
CRAY XD-1	Supercomputer	2004	64 KB/64 KB	1MB	—

Main Idea of a Cache - keep a copy of frequently used information as "close" (w.r.t access time) to the processor as possible.



Steps when the CPU generates a memory request:
1) check the (faster) cache first
2) If the addressed memory value is in the cache (called a ***hit***), then no need to access memory
3) If the addressed memory value is NOT in the cache (called a ***miss***), then transfer the block of memory containing the reference to cache. (The CPU is stalled waiting while this occurs)
4) The cache supplies the memory value from the cache.

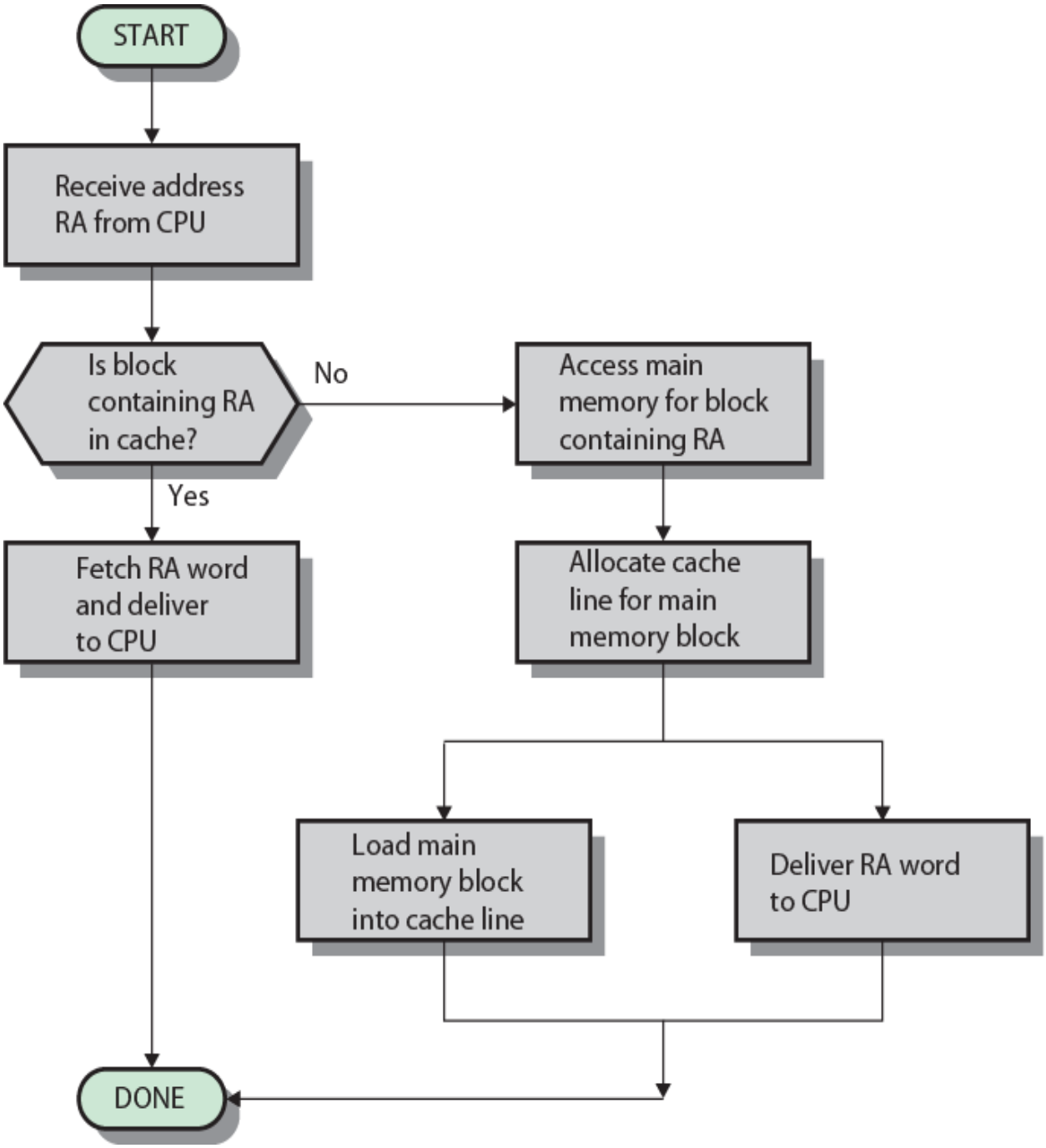
Effective Memory Access Time
Suppose that the hit time is 5 ns, the cache miss penalty is 160 ns, and the hit rate is 99%.

Effective Access Time ≈ (hit time * hit probability) + (miss penalty * miss probability)

Effective Access Time = 5 * 0.99 + 160 * (1 - 0.99) = 4.95 + 1.6 = 6.55 ns

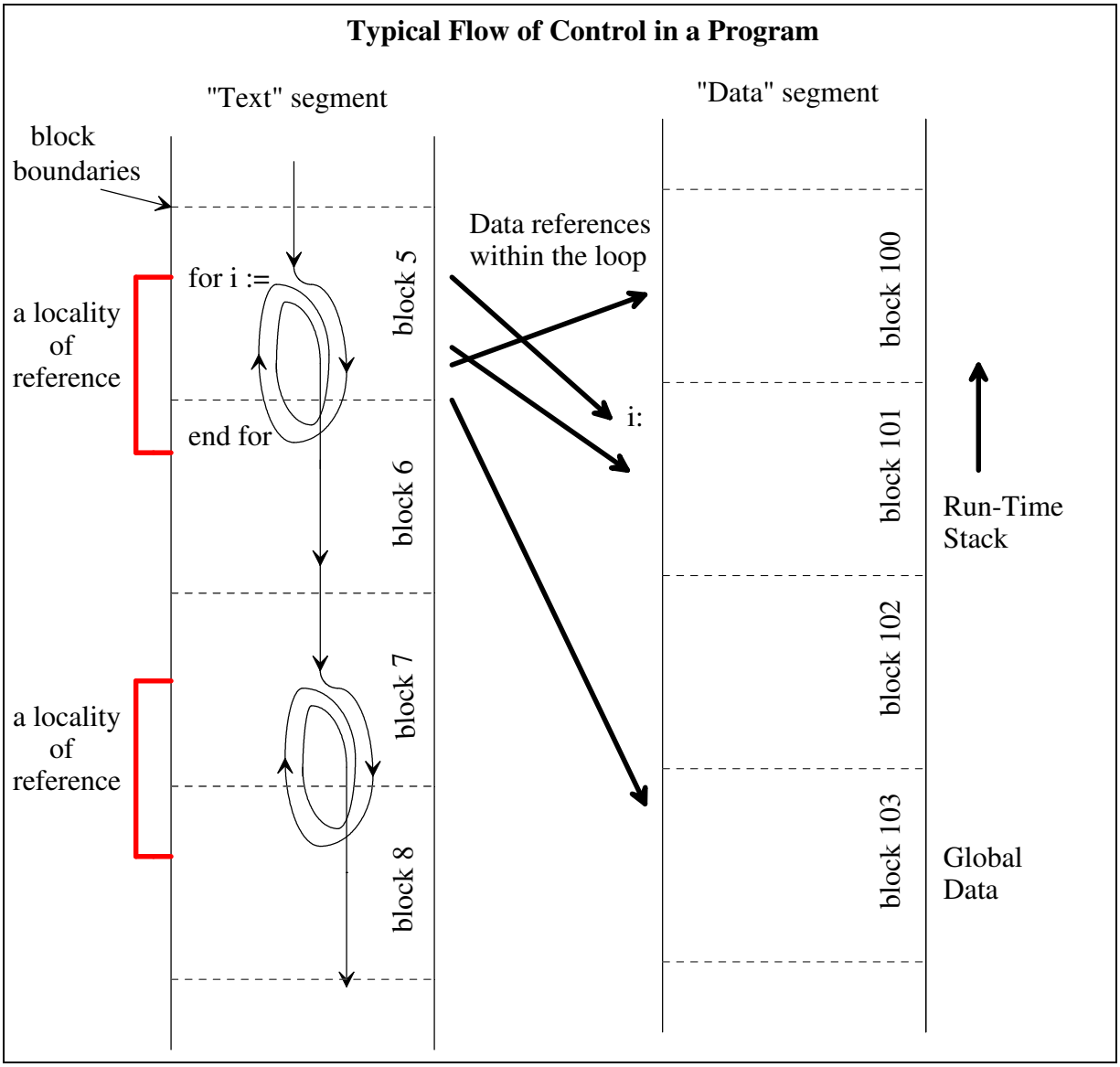
One way to reduce the miss penalty is to not have the cache wait for the whole block to be read from memory before supplying the accessed memory word.

Figure 4.5 Cache Read Operation



Fortunately, programs exhibit ***locality of reference*** that helps achieve high hit-rates:
1) ***spatial locality*** - if a (logical) memory address is referenced, nearby memory addresses will tend to be referenced soon.

2) ***temporal locality*** - if a memory address is referenced, it will tend to be referenced again soon.

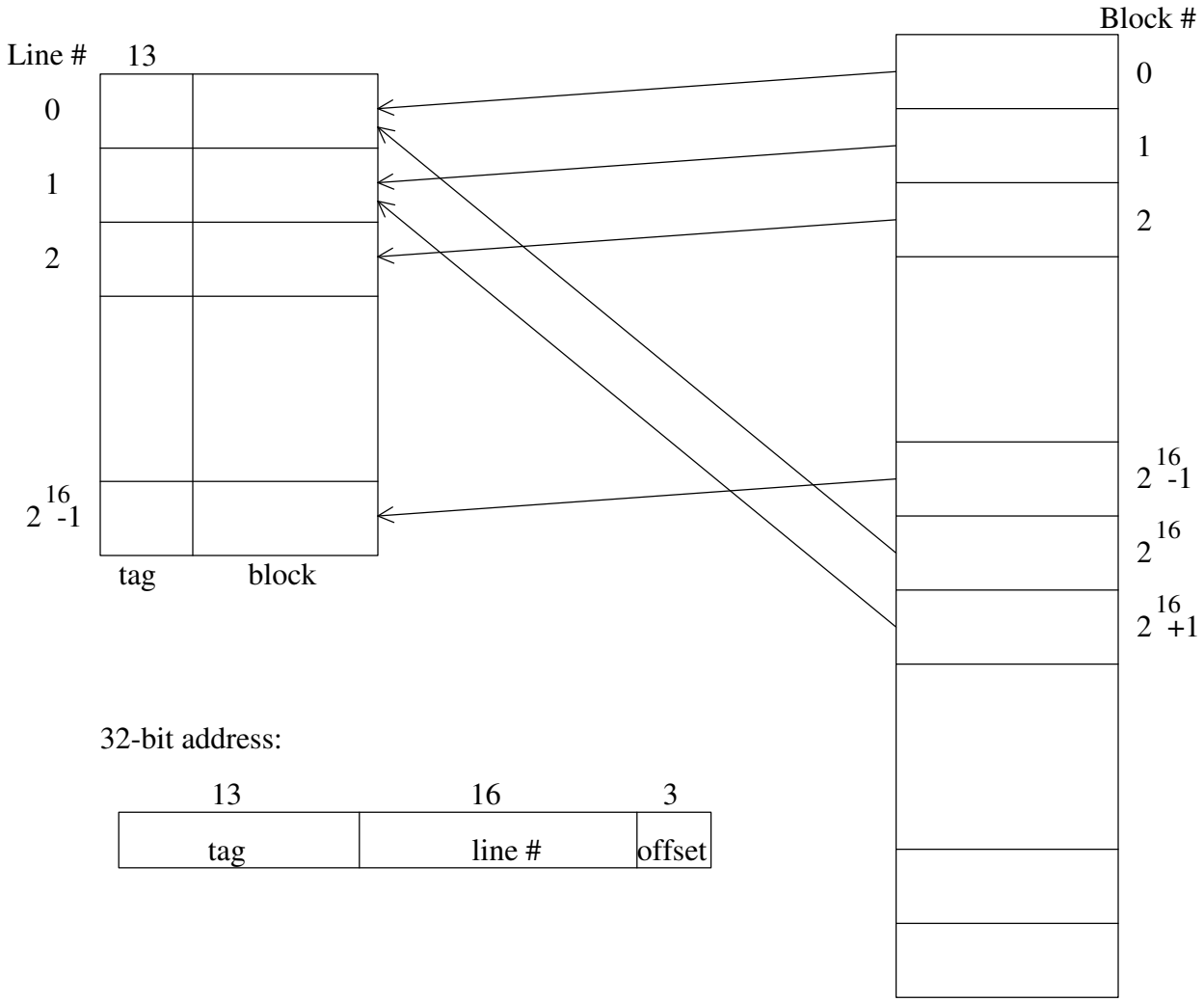


Cache - Small fast memory between the CPU and RAM/Main memory.

Example:
- 32-bit address
- 512 KB (2^{19})
- 8 byte per block/line
- byte-addressable memory

$$\text{Number of Cache Line} = \frac{\text{size of cache}}{\text{size of line}} = \frac{2^{19}}{2^3} = 2^{16}$$

Three Types of Cache:
1) Direct-mapped - a memory block maps to a single cache line



32-bit address:

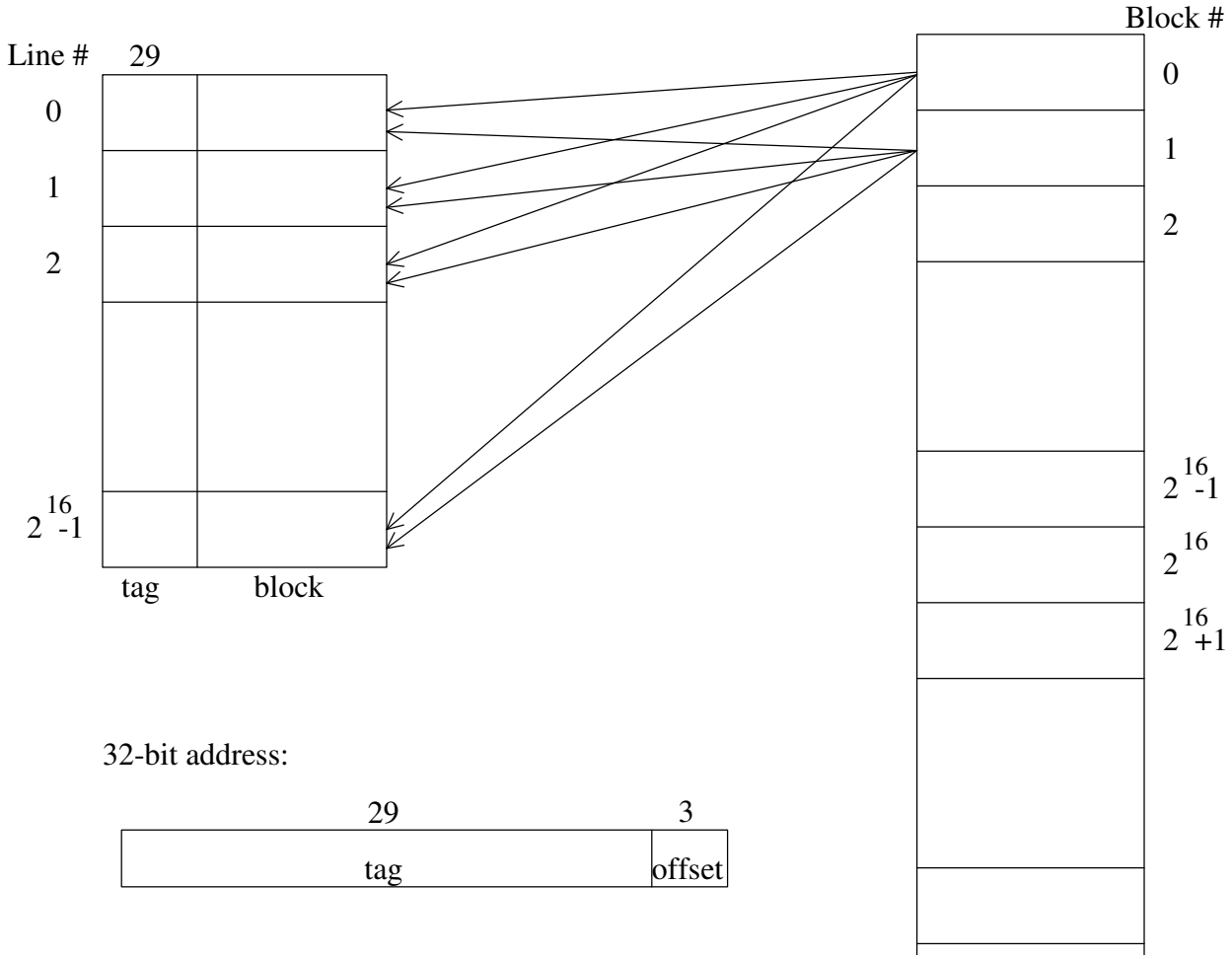
13	16	3
tag	line #	offset

Cache - Small fast memory between the CPU and RAM/Main memory.

Example:
- 32-bit address, byte-addressable memory
- 512 KB (2^{19})
- 8 byte per block/line

Number of Cache Line $= \frac{\text{size of cache}}{\text{size of line}} = \frac{2^{19}}{2^3} = 2^{16}$

2) Fully-Associative Cache - a memory block can map to any cache line



Advantage: Flexibility on what's in the cache

Disadvantage: Complex circuit to compare all tags of the cache with the tag in the target address

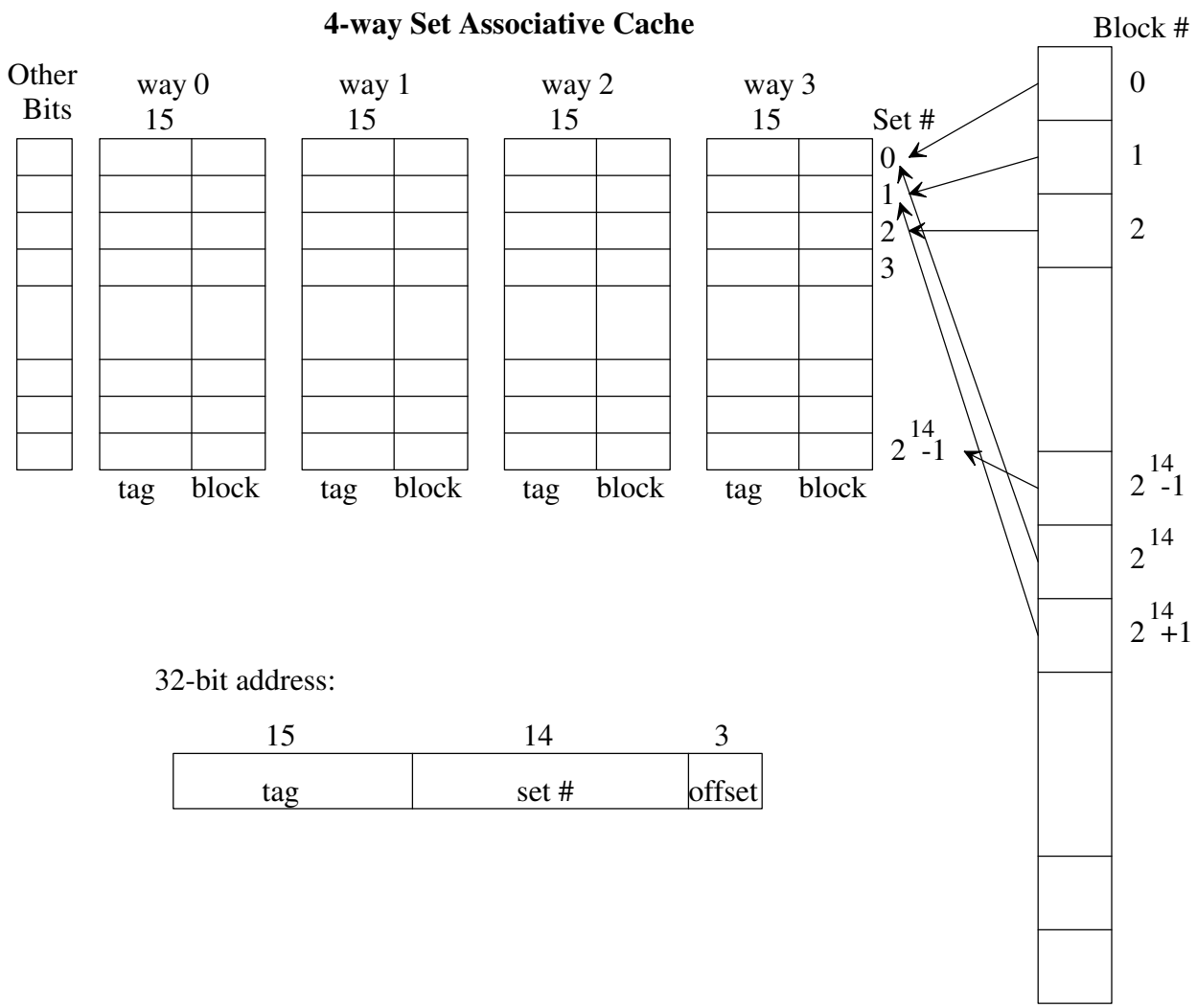
Therefore, they are expensive and slower so use only for small caches (say 8-64 lines)

Replacement algorithms - on a miss of a full cache, we must select a block in the cache to replace
- LRU - replace the cache block that has not been used for the longest time (need additional bits)
- Random - select a block randomly (only slightly worse that LRU)
- FIFO - select the block that has been in the cache for the longest time (slightly worse that LRU)

3) Set-Associative Cache - a memory block can map to a small (2, 4, or 8) set of cache lines
Common Possibilities:

- 2-way set associative - each memory block can map to either of two lines in the cache
- 4-way set associative - each memory block can map to either of four lines in the cache

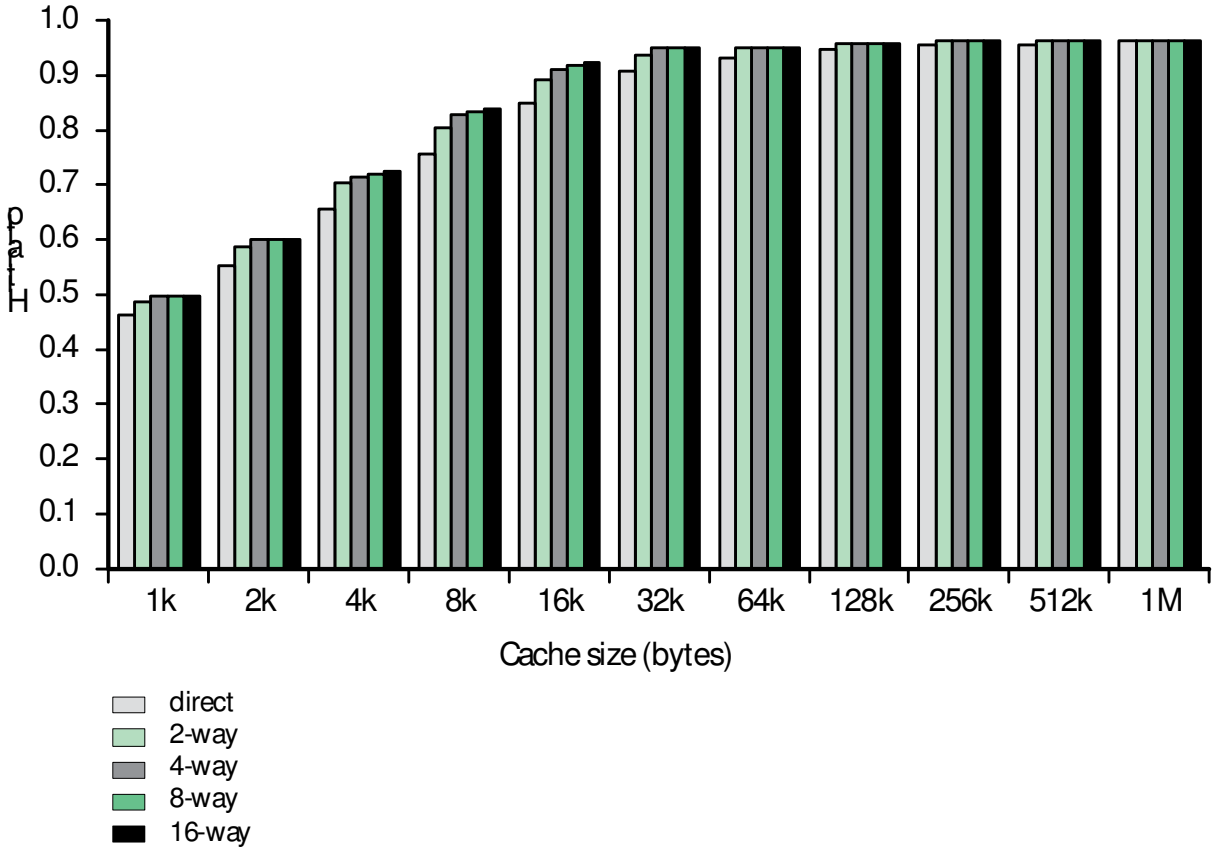
$$\text{Number of Sets} = \frac{\text{number of cache lines}}{\text{size of each set}} = \frac{2^{16}}{4} = \frac{2^{16}}{2^2} = 2^{14}$$



32-bit address:

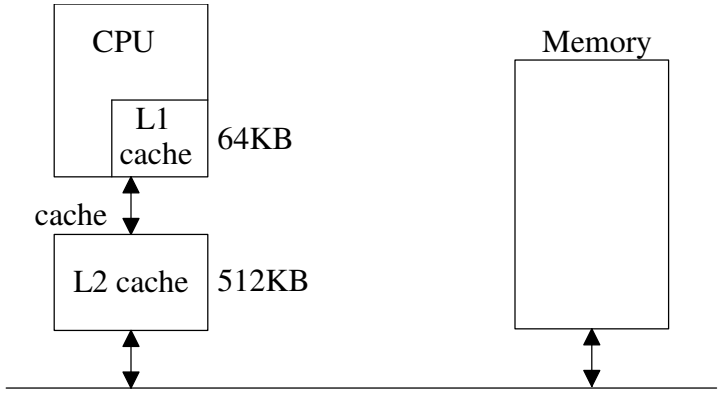
15	14	3
tag	set #	offset

Figure 4.16 Varying Associativity over Cache Size



Block/Line Size
The larger the line size:
- fewer cache line for the same size cache
- improves hit rate since larger blocks are read when a miss occurs
- larger miss penalty since more words are read from memory when a miss occurs

Number of Caches:
Issues:
- Number of cache levels

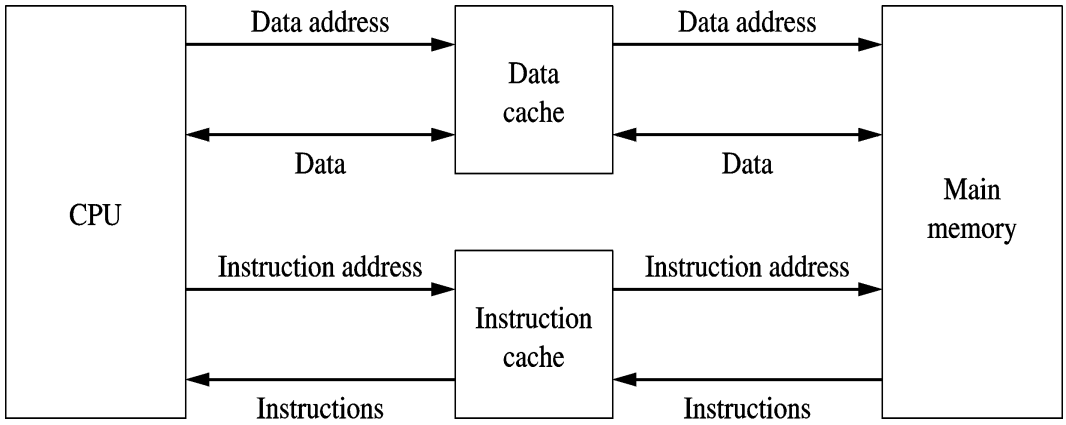


- unified vs. split caches

split caches - separate smaller caches for data and instructions
unified cache - data and instructions in the same cache

Advantages of each:
split caches - reduces contention for "memory" between instruction and data accesses

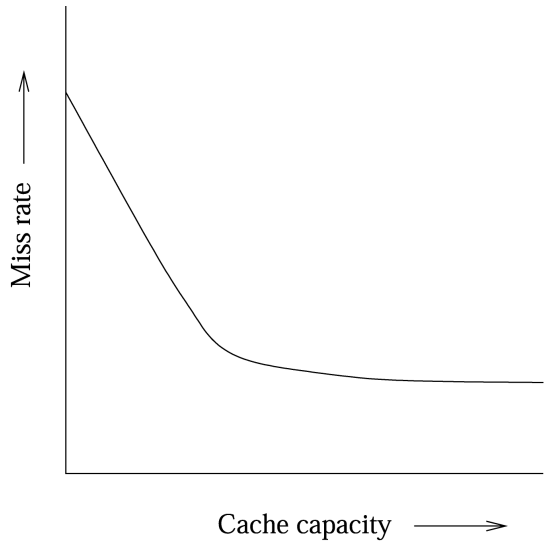
unified caches - balances the load between instructions and data automatically
(e.g., a tight loop might need more data blocks than instruction blocks)

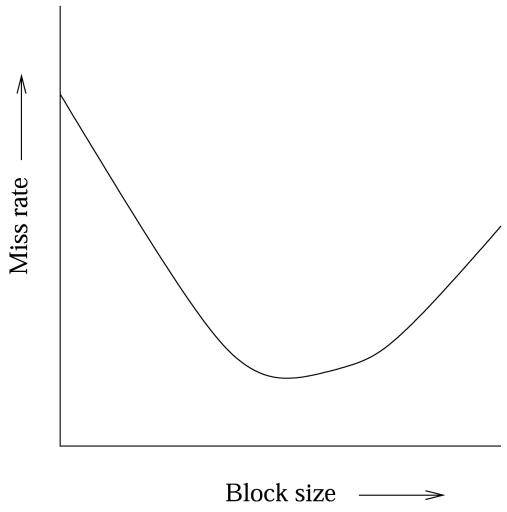


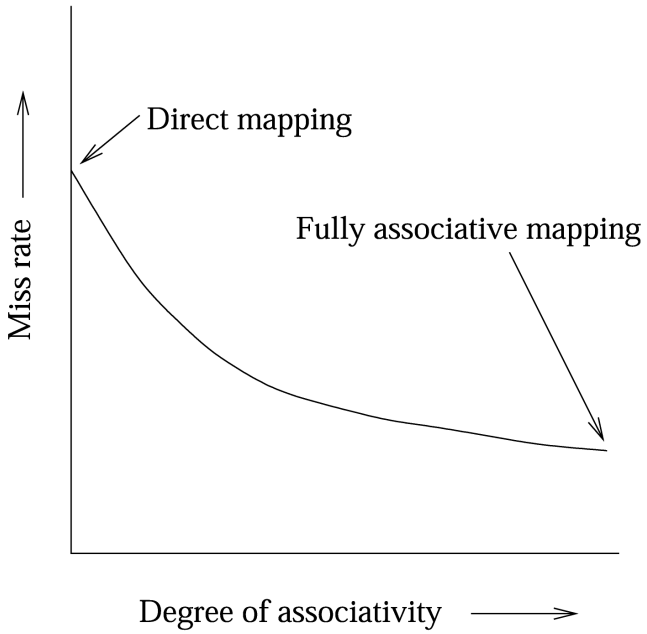
Types of Cache Misses:

- Compulsory Misses: misses due to the first access to a block
- Capacity Misses: misses on blocks that were in the cache, but were forced out due to the capacity of the cache.
- Conflict/Collision Misses: misses due to conflict caused by the direct or set-associated mapping that are eliminated by using fully-associative mapping

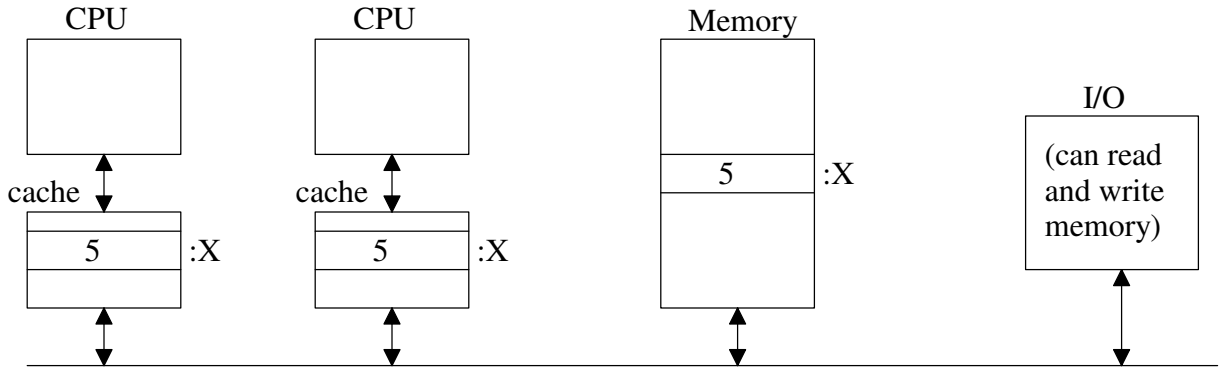
Studying the performance of a cache vs. characteristics of the cache:



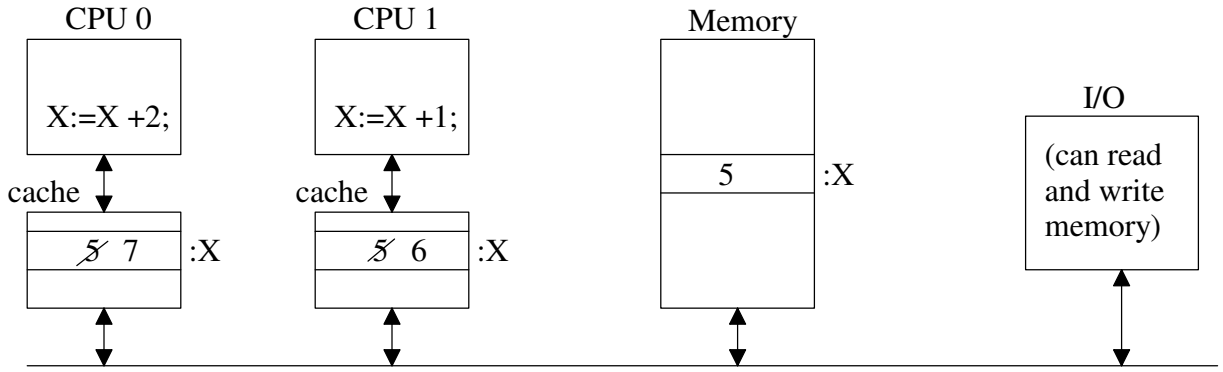




Write Policy - do we keep the cache and memory copy of a block identical???



Just reading a shared variable causes no problems - all caches have the same value
Writing can cause a "***cache-coherency problem***"



Write Policies
write back - CPU only changes local cache copy until that block is replaced, then it is written back to memory (a UPDATE/DIRTY bit is associated with each cache line to indicate if it has been modified). If we assume that CPU 0 writes the block back to memory before CPU 1, then X's resulting value will be 6. Thereby, discarding the effect of "X:=X+2".

Disadvantage(s) of writeback?

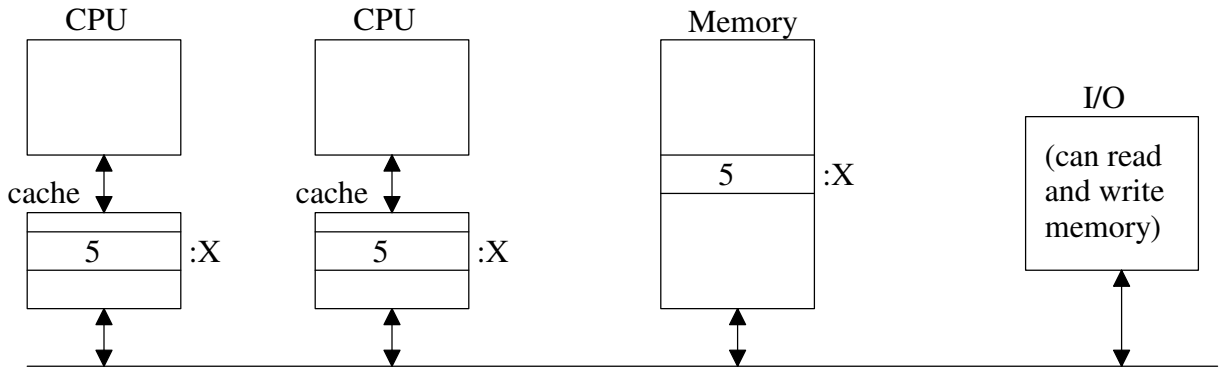
Advantage(s) of writeback?

write through - on a write to a cache block, write to the main memory copy to keep it up to date
To avoid stalling the CPU on a write, a write buffer can be used to allow the CPU to continue execution without stalling to wait for the write.

write miss options:
write allocate - the block written to is read into the cache before updating
no-write allocate - no block is allocated in the cache and only the lower-level memory is modified

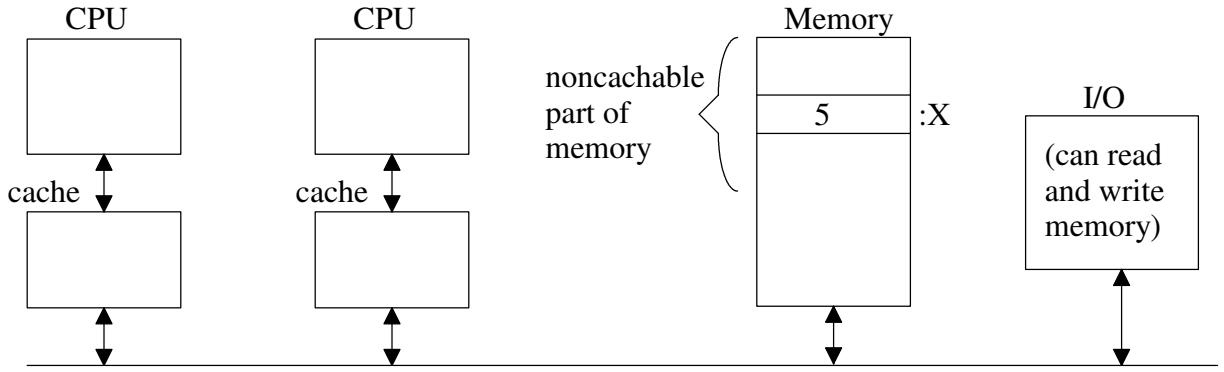
Cache Coherency Solutions

a) bus watching with write through / Snoopy caches - caches eavesdrop on the bus for other caches write requests. If the cache contains a block written by another cache, it take some action such as invalidating it's cache copy.



The MESA protocol is a common cache-coherency protocol.

b) noncachable memory - part of memory is designated as noncachable, so the memory copy is the only copy. Access to this part of memory always generate a "miss".



c) Hardware transparency - additional hardware is added to update all caches and memory at once on a write.

Figure 4.18 Pentium 4 Block Diagram