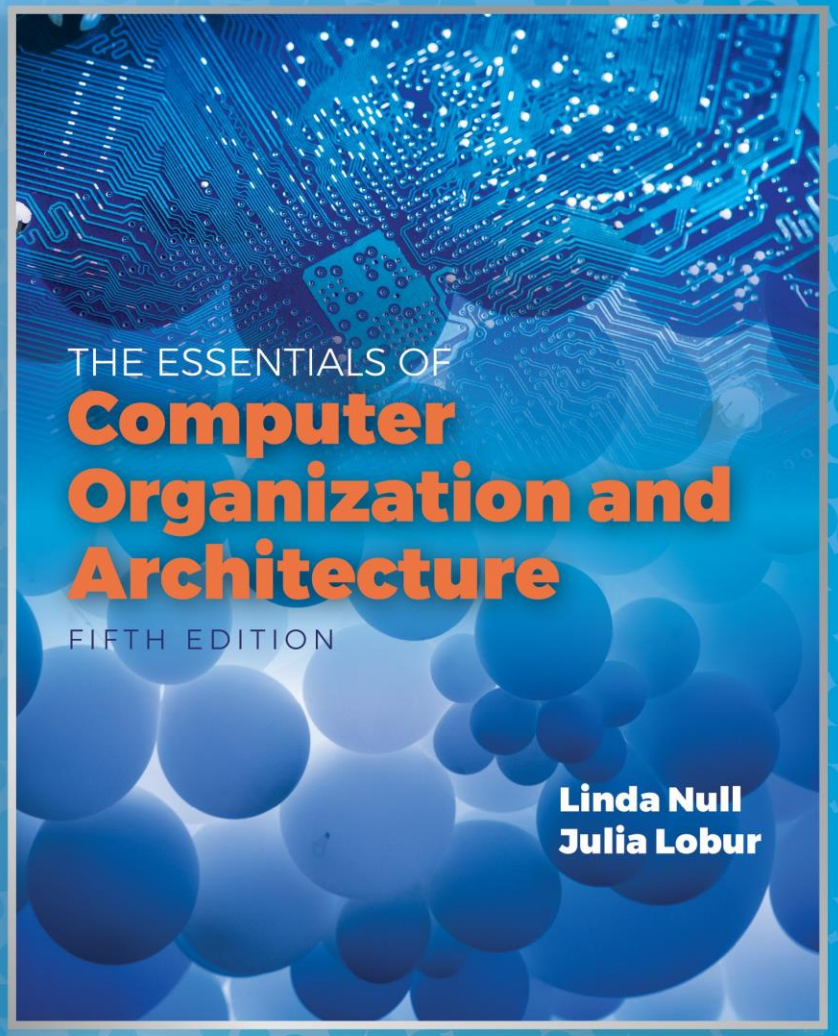
This is the fourth lecture of Chapter 6

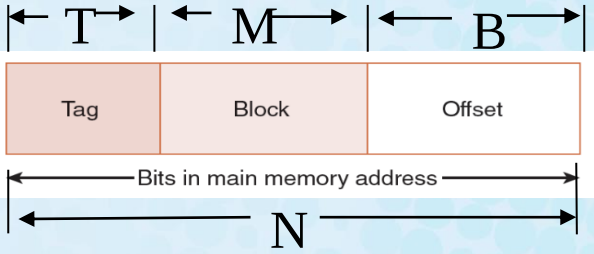
Chapter 6

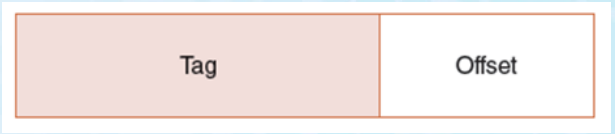
Memory (D)



Quick review of last lecture

- Cache Placement
  - Direct Mapping
    - Memory Address format
    - Memory_size = Block_size * #_of_blocks
  - Fully associative mapping
    - Memory Address format
    - Cache replacement
  - N-way set associative mapping





6.4 Cache Memory (32 of 45)

- With fully associative and set associative cache, a *replacement policy* is invoked when it becomes necessary to evict a block from cache.
- An *optimal* replacement policy would be able to look into the future to see which blocks won't be needed for the longest period of time.
- Although it is impossible to implement an optimal replacement algorithm, it is instructive to use it as a benchmark for assessing the efficiency of any other scheme we come up with.

6.4 Cache Memory (33 of 45)

- The replacement policy that we choose depends upon the locality that we are trying to optimize—usually, we are interested in temporal locality.
- A *least recently used* (LRU) algorithm keeps track of the last time that a block was assessed and evicts the block that has been unused for the longest period of time.
- The disadvantage of this approach is its complexity: LRU has to maintain an access history for each block, which ultimately slows down the cache.

6.4 Cache Memory (34 of 45)

- *First-in, first-out* (FIFO) is a popular cache replacement policy.
- In FIFO, the block that has been in the cache the longest, regardless of when it was last used.
- A *random* replacement policy does what its name implies: It picks a block at random and replaces it with a new block.
- Random replacement can certainly evict a block that will be needed often or needed soon, but it never thrashes.

Cache Replacement

Block tracing experiments

LRU

Slot	0	1	2	4	2	3	7	2	1	3	1	Hit Rate
a	0	0	0	4	4	4	7	7	7	3	3	
b		1	1	1	1	3	3	3	1	1	1	3/11
c			2	2	2	2	2	2	2	2	2	
Misses	*	*	*	*		*	*		*	*		

block trace

FIFO

Slot	0	1	2	4	2	3	7	2	1	3	1	Hit Rate
a	0	0	0	4	4	4	4	2	2	2	2	
b		1	1	1	1	3	3	3	1	1	1	2/11
c			2	2	2	2	7	7	7	3	3	
Misses	*	*	*	*		*	*	*	*	*		

block trace

6.4 Cache Memory (35 of 45)

- The performance of hierarchical memory is measured by its *effective access time* (EAT).
- EAT is a weighted average that takes into account the hit ratio and relative access times of successive levels of memory.
- The EAT for a two-level memory is given by:

$$\mathbf{EAT} = \boldsymbol{H} \times \mathbf{Access}_{\boldsymbol{C}} + (1 - \boldsymbol{H}) \times \mathbf{Access}_{\boldsymbol{MM}}$$

  - where H is the cache hit rate and Access_C and Access_{MM} are the access times for cache and main memory, respectively.

6.4 Cache Memory (36 of 45)

- For example, consider a system with a main memory access time of 200ns supported by a cache having a 10ns access time and a hit rate of 99%.
- Suppose access to cache and main memory occurs concurrently (the accesses overlap).
- The EAT is:

$$0.99(10ns) + 0.01(200ns) = 9.9ns + 2ns = 11ns$$

6.4 Cache Memory (37 of 45)

- For example, consider a system with a main memory access time of 200ns supported by a cache having a 10ns access time and a hit rate of 99%.
- If the accesses do not overlap, the EAT is:

$$0.99(10\text{ns}) + 0.01(10\text{ns} + 200\text{ns})$$

$$= 9.9\text{ns} + 2.01\text{ns} = 12\text{ns}$$

- This equation for determining the effective access time can be extended to any number of memory levels, as we will see in later sections.

6.4 Cache Memory (38 of 45)

- Caching is depends upon programs exhibiting good locality.
  - Some object-oriented programs have poor locality owing to their complex, dynamic structures.
  - Arrays stored in column-major rather than row-major order can be problematic for certain cache organizations.
- With poor locality, caching can actually cause performance degradation rather than performance improvement.

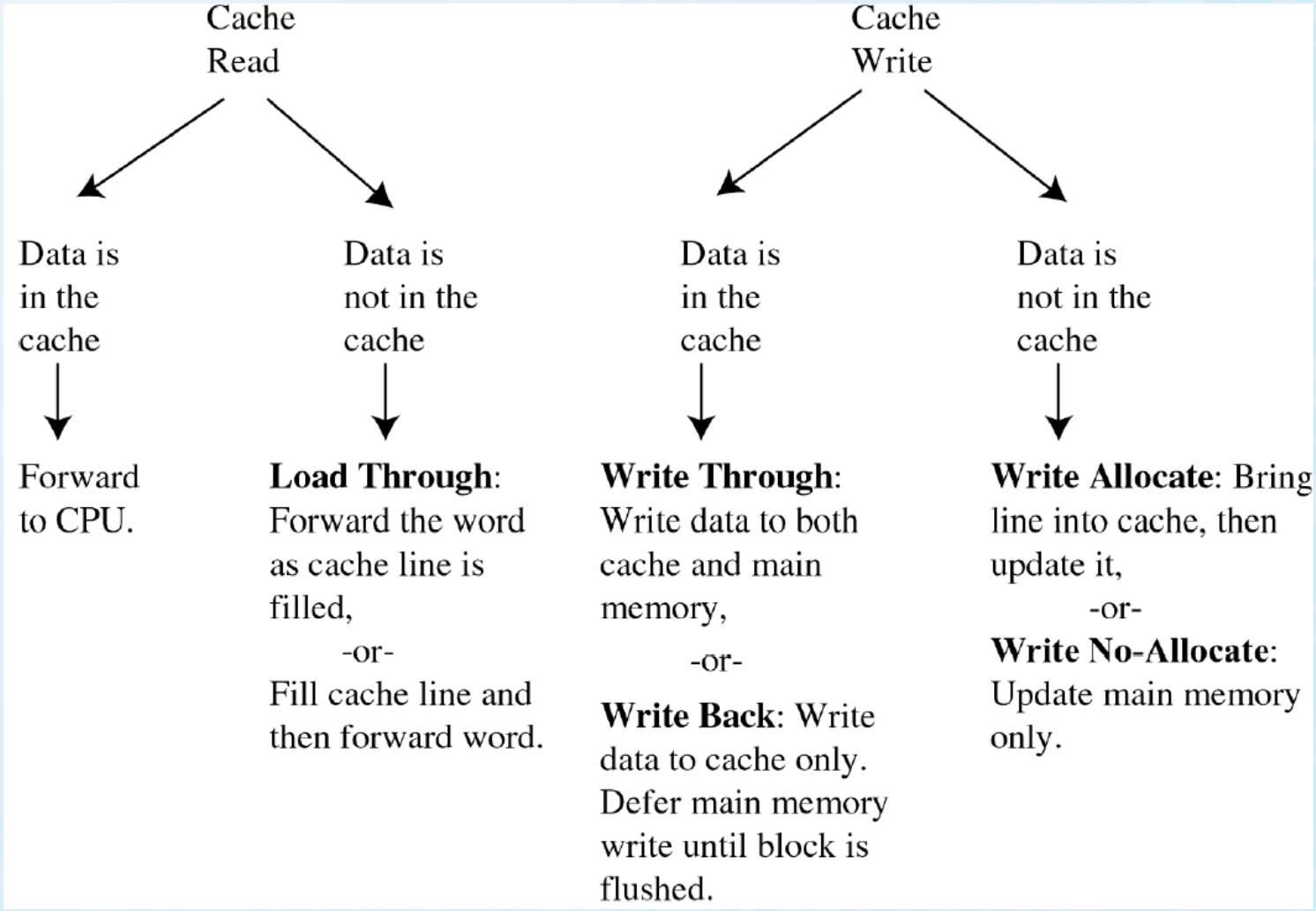
6.4 Cache Memory (39 of 45)

- Cache replacement policies must take into account *dirty blocks*, those blocks that have been updated while they were in the cache.
- Dirty blocks must be written back to memory. A *write policy* determines how this will be done.
- There are two types of write policies, *write through* and *write back*.
- Write through updates cache and main memory simultaneously on every write.
- Write back (also called *copyback*) updates memory only when the block is selected for replacement.

6.4 Cache Memory (40 of 45)

- The disadvantage of write through is that memory must be updated with each cache write, which slows down the access time on updates. This slowdown is usually negligible, because the majority of accesses tend to be reads, not writes.
- The advantage of write back is that memory traffic is minimized, but its disadvantage is that memory does not always agree with the value in cache, causing problems in systems with many concurrent users.

Cache Read and Write Policies



6.4 Cache Memory (41 of 45)

- The cache we have been discussing is called a *unified* or *integrated* cache where both instructions and data are cached.
- Many modern systems employ separate caches for data and instructions.
  - This is called a *Harvard* cache.
- The separation of data from instructions provides better locality, at the cost of greater complexity.
  - Simply making the cache larger provides about the same performance improvement without the complexity.

6.4 Cache Memory (42 of 45)

- Cache performance can also be improved by adding a small associative cache to hold blocks that have been evicted recently.
  - This is called a *victim cache*.
- A trace cache is a variant of an instruction cache that holds decoded instructions for program branches, giving the illusion that noncontiguous instructions are really contiguous.

6.4 Cache Memory (43 of 45)

- Most of today's small systems employ multilevel cache hierarchies.
- The levels of cache form their own small memory hierarchy.
- Level 1 cache (8KB to 64KB) is situated on the processor itself.
  - Access time is typically about 4ns.
- Level 2 cache (64KB to 2MB) may be on the motherboard, or on an expansion card.
  - Access time is usually around 15–20ns.

6.4 Cache Memory (44 of 45)

- In systems that employ three levels of cache, the Level 2 cache is placed on the same die as the CPU (reducing access time to about 10ns).
- Accordingly, the Level 3 cache (2MB to 256MB) refers to cache that is situated between the processor and main memory.
- Once the number of cache levels is determined, the next thing to consider is whether data (or instructions) can exist in more than one cache level.

6.4 Cache Memory (45 of 45)

- If the cache system used an *inclusive* cache, the same data may be present at multiple levels of cache.
- *Strictly inclusive* caches guarantee that all data in a smaller cache also exists at the next higher level.
- *Exclusive* caches permit only one copy of the data.
- The tradeoffs in choosing one over the other involve weighing the variables of access time, memory size, and circuit complexity.

Next Class

- Next class will give examples to show how to compute
  - Hit rate
  - Miss rate
  - EAT

  For a given program (sequence of instructions).