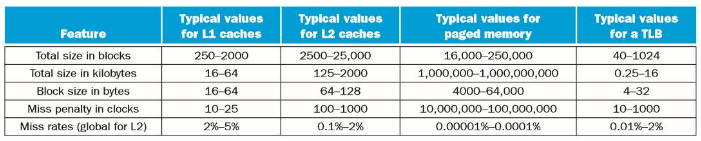
5.8 A common framework for memory hierarchy

This section has been set as optional by your instructor.

By now, you've recognized that the different types of memory hierarchies have a great deal in common. Although many of the aspects of memory hierarchies differ quantitatively, many of the policies and features that determine how a hierarchy functions are similar qualitatively. The figure below shows how some of the quantitative characteristics of memory hierarchies can differ. In the rest of this section, we will discuss the common operational alternatives for memory hierarchies, and how these determine their behavior. We will examine these policies in a series of four questions that apply between any two levels of a memory hierarchy, although for simplicity, we will primarily use terminology for caches.

Figure 5.8.1: The key quantitative design parameters that characterize the major elements of memory hierarchy in a computer (COD Figure 5.34).

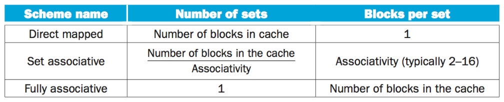
These are typical values for these levels as of 2012. Although the range of values is wide, this is partially because many of the values that have shifted over time are related; for example, as caches become larger to overcome larger miss penalties, block sizes also grow. While not shown, server microprocessors today also have L3 caches, which can be 2 to 8 MiB and contain many more blocks than L2 caches. L3 caches lower the L2 miss penalty to 30 to 40 clock cycles.



Feature	Typical values for L1 caches	Typical values for L2 caches	Typical values for paged memory	Typical values for a TLB
Total size in blocks	250–2000	2500–25,000	16,000–250,000	40–1024
Total size in kilobytes	16–64	125–2000	1,000,000–1,000,000,000	0.25–16
Block size in bytes	16–64	64–128	4000–64,000	4–32
Miss penalty in clocks	10–25	100–1000	10,000,000–100,000,000	10–1000
Miss rates (global for L2)	2%–5%	0.1%–2%	0.00001%–0.0001%	0.01%–2%

Question 1: Where can a block be placed?

We have seen that block placement in the upper level of the hierarchy can use a range of schemes, from direct mapped to set associative to fully associative. As mentioned above, this entire range of schemes can be thought of as variations on a set-associative scheme where the number of sets and the number of blocks per set varies:

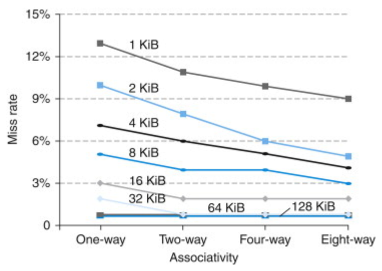


Scheme name	Number of sets	Blocks per set
Direct mapped	Number of blocks in cache	1
Set associative	$\frac{\text{Number of blocks in the cache}}{\text{Associativity}}$	Associativity (typically 2–16)
Fully associative	1	Number of blocks in the cache

The advantage of increasing the degree of associativity is that it usually decreases the miss rate. The improvement in miss rate comes from reducing misses that compete for the same location. We will examine these in more detail shortly. First, let's look at how much improvement is gained. The figure below shows the miss rates for several cache sizes as associativity varies from direct mapped to eight-way set associative. The largest gains are obtained in going from direct mapped to two-way set associative, which yields between a 20% and 30% reduction in the miss rate. As cache sizes grow, the relative improvement from associativity increases only slightly; since the overall miss rate of a larger cache is lower, the opportunity for improving the miss rate decreases and the absolute improvement in the miss rate from associativity shrinks significantly. The potential disadvantages of associativity, as we mentioned earlier, are increased cost and slower access time.

Figure 5.8.2: The data cache miss rates for each of eight cache sizes improve as the associativity increases (COD Figure 5.35).

While the benefit of going from one-way (direct mapped) to two-way set associative is significant, the benefits of further associativity are smaller (e.g., 1—10% improvement going from two-way to four-way versus 20—30% improvement going from one-way to two-way). There is even less improvement in going from four-way to eight-way set associative, which, in turn, comes very close to the miss rates of a fully associative cache. Smaller caches obtain a significantly larger absolute benefit from associativity because the base miss rate of a small cache is larger. COD Figure 5.16 (The data cache miss rates for an organization like the Intrinsity FastMATH ...) explains how these data were collected.



PARTICIPATION ACTIVITY 5.8.1: Where to place a block?

1) The largest reduction in misses occurs when going from a _____.

- direct mapped to a two-way set associative cache
- two-way to a four-way set associative cache

2) What kind of cache benefits the greatest from a fully associative cache?

○ large cache

○ small cache

Question 2: How is a block found?

The choice of how we locate a block depends on the block placement scheme, since that dictates the number of possible locations. We can summarize the schemes as follows:

Scheme name	Number of sets	Blocks per set
Direct mapped	Number of blocks in cache	1
Set associative	$\frac{\text{Number of blocks in the cache}}{\text{Associativity}}$	Associativity (typically 2–16)
Fully associative	1	Number of blocks in the cache

The choice among direct-mapped, set-associative, or fully associative mapping in any memory hierarchy will depend on the cost of a miss versus the cost of implementing associativity, both in time and in extra hardware. Including the L2 cache on the chip enables much higher associativity, because the hit times are not as critical and the designer does not have to rely on standard SRAM chips as the building blocks. Fully associative caches are prohibitive except for small sizes, where the cost of the comparators is not overwhelming and where the absolute miss rate improvements are greatest.

In virtual memory systems, a separate mapping table—the page table—is kept to index the memory. In addition to the storage needed for the table, using an index table requires an extra memory access. The choice of full associativity for page placement and the extra table is motivated by these facts:

1. Full associativity is beneficial, since misses are very expensive.
2. Full associativity allows software to use sophisticated replacement schemes that are designed to reduce the miss rate.
3. The full map can be easily indexed with no extra hardware and no searching required.

Therefore, virtual memory systems almost always use fully associative placement.

Set-associative placement is often used for caches and TLBs, where the access combines indexing and the search of a small set. A few systems have used direct-mapped caches because of their advantage in access time and simplicity. The advantage in access time occurs because finding the requested block does not depend on a comparison. Such design choices depend on many details of the implementation, such as whether the cache is on-chip, the technology used for implementing the cache, and the critical role of cache access time in determining the processor cycle time.

PARTICIPATION ACTIVITY 5.8.2: How to find a block.

1) Locating a block in memory depends on the block placement scheme.

○ True

○ False

2) Full associativity is not best for virtual memory systems because of a larger access latency.

○ True

○ False

3) For caches and TLBs, direct mapping tends to have faster placement times than set-associative mapping.

○ True

○ False

Question 3: Which block should be replaced on a cache miss?

When a miss occurs in an associative cache, we must decide which block to replace. In a fully associative cache, all blocks are candidates for replacement. If the cache is set associative, we must choose among the blocks in the set. Of course, replacement is easy in a direct-mapped cache because there is only one candidate.

There are the two primary strategies for replacement in set-associative or fully associative caches:

- *Random*: Candidate blocks are randomly selected, possible using some hardware assistance.
- *Least recently used* (LRU): The block replaced is the one that has been unused for the longest time.

In practice, LRU is too costly to implement for hierarchies with more than a small degree of associativity (two to four, typically), since tracking the usage information is expensive. Even for four-way set associativity, LRU is often approximated—for example, by keeping track of which pair of blocks is LRU (which requires 1 bit), and then tracking which block in each pair is LRU (which requires 1 bit per pair).

For larger associativity, either LRU is approximated or random replacement is used. In caches, the replacement algorithm is in hardware, which means that the scheme should be easy to implement. Random replacement is simple to build in hardware, and for a two-way set-associative cache, random replacement has a miss rate about 1.1 times higher than LRU replacement. As the caches become larger, the miss rate for both replacement strategies falls, and the absolute difference becomes small. In fact, random replacement can sometimes be better than the simple LRU approximations that are easily implemented in hardware.

In virtual memory, some form of LRU is always approximated, since even a tiny reduction in the miss rate can be important when the cost of a miss is enormous. Reference bits or equivalent functionality are often provided to make it easier for the operating system to track a set of less recently used pages. Because misses are so expensive and relatively infrequent, approximating this information primarily in software is acceptable.

PARTICIPATION ACTIVITY 5.8.3: How to replace a block.

1) In a direct-mapped cache, how many blocks are candidates for replacement?

Check Show answer

2) The primary replacement schemes for set and fully-associative caches are _____ and LRU.

Check Show answer

3) Which of the two replacement schemes (random or LRU) is better for caches with a smaller degree of associativity?

Check Show answer

Question 4: What happens on a write?

A key characteristic of any memory hierarchy is how it deals with writes. We have already seen the two basic options:

- *Write-through*: The information is written to both the block in the cache and the block in the lower level of the memory hierarchy (main memory for a cache). The caches in COD Section 5.3 (The basics of caches) used this scheme.
- *Write-back*: The information is written just to the block in the cache. The modified block is written to the lower level of the hierarchy only when it is replaced. Virtual memory systems always use write-back, for the reasons discussed in COD Section 5.7 (Virtual memory).

Both write-back and write-through have their advantages. The key advantages of write-back are the following:

- 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 lower level in the hierarchy.
- When blocks are written back, the system can make effective use of a high-bandwidth transfer, since the entire block is written.

Write-through has these advantages:

- Misses are simpler and cheaper because they never require a block to be written back to the lower level.
- Write-through is easier to implement than write-back, although to be realistic, a write-through cache will still need to use a write buffer.

PARTICIPATION ACTIVITY 5.8.4: Performing a write.

1) The two memory system write methods are write-back and write-off.

○ True
○ False

2) A benefit of write-back is that multiple writes to a block only require one write to main memory.

○ True
○ False

3) Misses are less costly when using the write-through method.

○ True
○ False

The Big Picture

Caches, TLBs, and virtual memory may initially look very different, but they rely on the same two principles of locality, and they can be understood by their answers to four questions:

Question 1: Where can a block be placed?
Answer: One place (direct mapped), a few places (set associative), or any place (fully associative).

Question 2: How is a block found?
Answer: There are four methods: indexing (as in a direct-mapped cache), limited search (as in a set-associative cache), full search (as in a fully associative cache), and a separate lookup table (as in a page table).

Question 3: What block is replaced on a miss?
Answer: Typically, either the least recently used or a random block.

Question 4: How are writes handled?
Answer: Each level in the hierarchy can use either write-through or write-back.

In virtual memory systems, only a write-back policy is practical because of the long latency of a write to the lower level of the hierarchy. The rate at which writes are generated by a processor generally exceeds the rate at which the memory system can process them, even allowing for physically and logically wider memories and burst modes for DRAM. Consequently, today lowest-level caches typically use write-back.

The three Cs: An intuitive model for understanding the behavior of memory hierarchies

In this subsection, we look at a model that provides insight into the sources of misses in a memory hierarchy and how the misses will be affected by changes in the hierarchy. We will explain the ideas in terms of caches, although the ideas carry over directly to any other level in the hierarchy. In this model, all misses are classified into one of three categories (the *three Cs*):

- *Compulsory misses*: These are cache misses caused by the first access to a block that has never been in the cache. These are also called *cold-start misses*.
- *Capacity misses*: These are cache misses caused when the cache cannot contain all the blocks needed during execution of a program. Capacity misses occur when blocks are replaced and then later retrieved.
- *Conflict misses*: These are cache misses that occur in set-associative or direct-mapped caches when multiple blocks compete for the same set. Conflict misses are those misses in a direct-mapped or set-associative cache that are eliminated in a fully associative cache of the same size. These cache misses are also called *collision misses*.

Three Cs model: A cache model in which all cache misses are classified into one of three categories: compulsory misses, capacity misses, and conflict misses.

Compulsory miss: Also called ***cold-start miss***. A cache miss caused by the first access to a block that has never been in the cache.

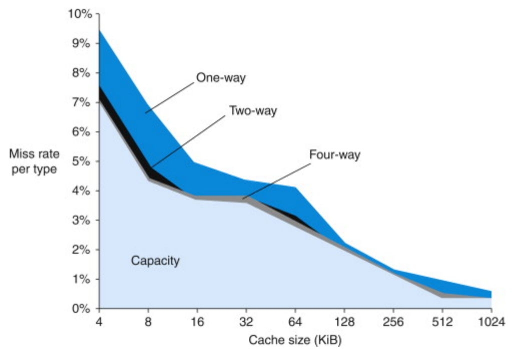
Capacity miss: A cache miss that occurs because the cache, even with full associativity, cannot contain all the blocks needed to satisfy the request.

Conflict miss: Also called ***collision miss***. A cache miss that occurs in a set-associative or direct-mapped cache when multiple blocks compete for the same set and that are eliminated in a fully associative cache of the same size.

The figure below shows how the miss rate divides into the three sources. These sources of misses can be directly attacked by changing some aspect of the cache design. Since conflict misses arise straight from contention for the same cache block, increasing associativity reduces conflict misses. Associativity, however, may slow access time, leading to lower overall performance.

Figure 5.8.3: The miss rate can be broken into three sources of misses (COD Figure 5.36).

This graph shows the total miss rate and its components for a range of cache sizes. These data are for the SPEC CPU2000 integer and floating-point benchmarks and are from the same source as the data in COD Figure 5.35 (The data cache miss rates for each of eight cache sizes ...). The compulsory miss component is 0.006% and cannot be seen in this graph. The next component is the capacity miss rate, which depends on cache size. The conflict portion, which depends both on associativity and on cache size, is shown for a range of associativities from one-way to eight-way. In each case, the labeled section corresponds to the increase in the miss rate that occurs when the associativity is changed from the next higher degree to the labeled degree of associativity. For example, the section labeled *two-way* indicates the additional misses arising when the cache has associativity of two rather than four. Thus, the difference in the miss rate incurred by a direct-mapped cache versus a fully associative cache of the same size is given by the sum of the sections marked *four-way*, *two-way*, and *one-way*. The difference between eight-way and four-way is so small that it is difficult to see on this graph.



Capacity misses can easily be reduced by enlarging the cache; indeed, second-level caches have been growing steadily bigger for many years. Of course, when we make the cache larger, we must also be careful about increasing the access time, which could lead to lower overall performance. Thus, first-level caches have been growing slowly, if at all.

Because compulsory misses are generated by the first reference to a block, the primary way for the cache system to reduce the number of compulsory misses is to increase the block size. This will reduce the number of references required to touch each block of the program once, because the program will consist of fewer cache blocks. As mentioned above, increasing the block size too much can have a negative effect on performance because of the increase in the miss penalty.

The decomposition of misses into the three Cs is a useful qualitative model. In real cache designs, many of the design choices interact, and changing one cache characteristic will often affect several components of the miss rate. Despite such shortcomings, this model is a useful way to gain insight into the performance of cache designs.

PARTICIPATION ACTIVITY 5.8.5: The three Cs.

Conflict miss Capacity miss Compulsory miss

A miss caused when multiple blocks attempt to occupy the same set.

A miss caused by attempting to access

a block that has never been in the cache.

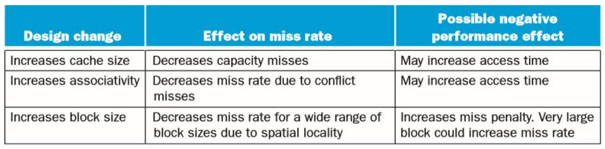
A miss caused when a cache does not have the space to hold all of the required blocks.

Reset

The Big Picture

The challenge in designing memory hierarchies is that every change that potentially improves the miss rate can also negatively affect overall performance, as the figure below summarizes. This combination of positive and negative effects is what makes the design of a memory hierarchy interesting.

Figure 5.8.4: Memory hierarchy design challenges (COD Figure 5.37).



Design change	Effect on miss rate	Possible negative performance effect
Increases cache size	Decreases capacity misses	May increase access time
Increases associativity	Decreases miss rate due to conflict misses	May increase access time
Increases block size	Decreases miss rate for a wide range of block sizes due to spatial locality	Increases miss penalty. Very large block could increase miss rate

PARTICIPATION ACTIVITY 5.8.6: Check yourself.

Which of the following statements (if any) are generally true?

1) There is no way to reduce compulsory misses.
   - True
   - False

2) Fully associative caches have no conflict misses.
   - True
   - False

3) In reducing misses, associativity is more important than capacity.
   - True
   - False

Provide feedback on this section