5.1 Introduction

> “ Ideally one would desire an indefinitely large memory capacity such that any particular … word would be immediately available. … We are … forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible.
> *A. W. Burks, H. H. Goldstine, and J. von Neumann, Preliminary Discussion of the Logical Design of an Electronic Computing Instrument, 1946*

From the earliest days of computing, programmers have wanted unlimited amounts of fast memory. The topics in this chapter aid programmers by creating that illusion. Before we look at creating the illusion, let's consider a simple analogy that illustrates the key principles and mechanisms that we use.

Suppose you were a student writing a term paper on important historical developments in computer hardware. You are sitting at a desk in a library with a collection of books that you have pulled from the shelves and are examining. You find that several of the important computers that you need to write about are described in the books you have, but there is nothing about the EDSAC. Therefore, you go back to the shelves and look for an additional book. You find a book on early British computers that covers the EDSAC. Once you have a good selection of books on the desk in front of you, there is a high probability that many of the topics you need can be found in them, and you may spend most of your time just using the books on the desk without returning to the shelves. Having several books on the desk in front of you saves time compared to having only one book there and constantly having to go back to the shelves to return it and take out another.

The same principle allows us to create the illusion of a large memory that we can access as fast as a very small memory. Just as you did not need to access all the books in the library at once with equal probability, a program does not access all of its code or data at once with equal probability. Otherwise, it would be impossible to make most memory accesses fast and still have large memory in computers, just as it would be impossible for you to fit all the library books on your desk and still find what you wanted quickly.

This underlies both the way in which you did your work in the library and the way that programs operate. The principle of locality states that programs access a relatively small portion of their address space at any instant of time, just as you accessed a very small portion of the library's collection. There are two different types of locality:

- *Temporal locality* (locality in time): if an item is referenced, it will tend to be referenced again soon. If you recently brought a book to your desk to look at, you will probably need to look at it again soon.
- *Spatial locality* (locality in space): if an item is referenced, items whose addresses are close by will tend to be referenced soon. For example, when you brought out the book on early English computers to learn about the EDSAC, you also noticed that there was another book shelved next to it about early mechanical computers, so you likewise brought back that book and, later on, found something useful in that book. Libraries put books on the same topic together on the same shelves to increase spatial locality. We'll see how memory hierarchies use spatial locality a little later in this chapter.

Temporal locality: The locality principle stating that if a data location is referenced then it will tend to be referenced again soon.

Spatial locality: The locality principle stating that if a data location is referenced, data locations with nearby addresses will tend to be referenced soon.

Just as accesses to books on the desk naturally exhibit locality, locality in programs arises from simple and natural program structures. For example, most programs contain loops, so instructions and data are likely to be accessed repeatedly, showing large temporal locality. Since instructions are normally accessed sequentially, programs also show high spatial locality. Accesses to data also exhibit a natural spatial locality. For example, sequential accesses to elements of an array or a record will naturally have high degrees of spatial locality.

PARTICIPATION ACTIVITY 5.1.1: Principle of locality.

1) Bob is building a fence behind his house. He uses a hammer to attach a board to the rail. Bob then measures and cuts the next board.

The likelihood that Bob will need the hammer again is an example of _____ locality.

○ spatial
○ temporal

2) Bob is building a fence behind his house. He grabs a hammer from the garage. Bob will likely need additional tools stored in the garage, so Bob also grabs nails, a shovel, and a level.

The likelihood that Bob will need resources stored together in the garage is an example of _____ locality.

○ spatial
○ temporal

3) Given the following loop, the high likelihood of accessing multiple elements within array A is an example of _____ locality.

```
while (i < 10){
    A[i] = A[i] + 2;
    i = i + 1;
}
```

○ spatial

- temporal

4) Given the following loop, the high likelihood of accessing `i = i + 1` repeatedly is an example of ____ locality.

```
while (i < 10){
    A[i] = A[i] + 2;
    i = i + 1;
}
```

- spatial
- temporal

5) Instructions may exhibit temporal locality, but not spatial locality.
- True
- False

6) Data may exhibit spatial locality, but not temporal locality.
- True
- False

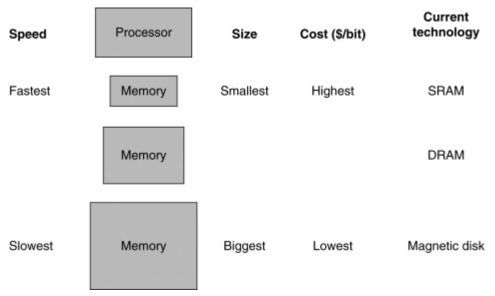
We take advantage of the principle of locality by implementing the memory of a computer as a *memory hierarchy*. A memory hierarchy consists of multiple levels of memory with different speeds and sizes. The faster memories are more expensive per bit than the slower memories and thus are smaller.

Memory hierarchy: A structure that uses multiple levels of memories; as the distance from the processor increases, the size of the memories and the access time both increase.

The figure below shows the faster memory is close to the processor and the slower, less expensive memory is below it. The goal is to present the user with as much memory as is available in the cheapest technology, while providing access at the speed offered by the fastest memory.

Figure 5.1.1: The basic structure of a memory hierarchy (COD Figure 5.1).

By implementing the memory system as a hierarchy, the user has the illusion of a memory that is as large as the largest level of the hierarchy, but can be accessed as if it were all built from the fastest memory. Flash memory has replaced disks in many personal mobile devices, and may lead to a new level in the storage hierarchy for desktop and server computers; see COD Section 5.2 (Memory technologies).

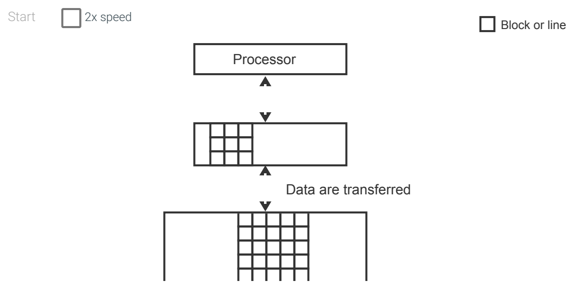


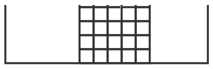
The data are similarly hierarchical: a level closer to the processor is generally a subset of any level further away, and all the data are stored at the lowest level. By analogy, the books on your desk form a subset of the library you are working in, which is in turn a subset of all the libraries on campus. Furthermore, as we move away from the processor, the levels take progressively longer to access, just as we might encounter in a hierarchy of campus libraries.

A memory hierarchy can consist of multiple levels, but data are copied between only two adjacent levels at a time, so we can focus our attention on just two levels. The upper level—the one closer to the processor—is smaller and faster than the lower level, since the upper level uses technology that is more expensive. The figure below shows that the minimum unit of information that can be either present or not present in the two-level hierarchy is called a *block* or a *line*; in our library analogy, a block of information is one book.

Block (or line): The minimum unit of information that can be either present or not present in a cache.

PARTICIPATION ACTIVITY 5.1.2: Every pair of levels in the memory hierarchy can be thought of as having an upper and lower level (COD Figure 5.2).





PARTICIPATION ACTIVITY 5.1.3: Memory hierarchy.

Find the error in each of the following statements.

1) A memory hierarchy consists of multiple levels of memory with the same speed and size.

2) Data are hierarchical. All of the data are stored in the memory closest to the processor Subsets of the data can be copied between adjacent levels

3) The minimum unit of information that can be either present or not present in the memory hierarchy is a bit / block .

If the data requested by the processor appear in some block in the upper level, this is called a *hit* (analogous to your finding the information in one of the books on your desk). If the data are not found in the upper level, the request is called a *miss*. The lower level in the hierarchy is then accessed to retrieve the block containing the requested data. (Continuing our analogy, you go from your desk to the shelves to find the desired book.) The *hit rate*, or *hit ratio*, is the fraction of memory accesses found in the upper level; it is often used as a measure of the performance of the memory hierarchy. The *miss rate* (1 - hit rate) is the fraction of memory accesses not found in the upper level.

Hit rate: The fraction of memory accesses found in a level of the memory hierarchy.

Miss rate: The fraction of memory accesses not found in a level of the memory hierarchy.

Since performance is the major reason for having a memory hierarchy, the time to service hits and misses is important. *Hit time* is the time to access the upper level of the memory hierarchy, which includes the time needed to determine whether the access is a hit or a miss (that is, the time needed to look through the books on the desk). The *miss penalty* is the time to replace a block in the upper level with the corresponding block from the lower level, plus the time to deliver this block to the processor (or the time to get another book from the shelves and place it on the desk). Because the upper level is smaller and built using faster memory parts, the hit time will be much smaller than the time to access the next level in the hierarchy, which is the major component of the miss penalty. (The time to examine the books on the desk is much smaller than the time to get up and get a new book from the shelves.)

Hit time: The time required to access a level of the memory hierarchy, including the time needed to determine whether the access is a hit or a miss.

Miss penalty: The time required to fetch a block into a level of the memory hierarchy from the lower level, including the time to access the block, transmit it from one level to the other, insert it in the level that experienced the miss, and then pass the block to the requestor.

PARTICIPATION ACTIVITY 5.1.4: Cache terminology.

A memory hierarchy is composed of an upper level and a lower level. Data are requested by the processor. 9 out of 10 requests find the data in the upper level and returns the data in 0.4 ns. The remaining requests require 0.7 ns to return the data.

Determine the corresponding values for the upper level memory.

0.4 0.1 0.7 0.9

Hit rate

Miss rate

Hit time

Miss penalty

Reset

As we will see in this chapter, the concepts used to build memory systems affect many other aspects of a computer, including how the operating system manages memory and I/O, how compilers generate code, and even how applications use the computer. Of course, because all programs spend much of their time accessing memory, the memory system is necessarily a major factor in determining performance. The reliance on memory hierarchies to achieve performance has meant that programmers, who used to be able to think of memory as a flat, random access storage device, now need to understand that memory is a hierarchy to get good performance. We show how important this understanding is in later examples, such as COD Figure 5.18 (The implementation of a four-way set-associative cache ...), and COD Section 5.15 (Going faster: cache blocking and matrix multiply), which shows how to double matrix multiply performance.

Since memory systems are critical to performance, computer designers devote a great deal of attention to these systems and develop sophisticated mechanisms for improving the performance of the memory system. In this chapter, we discuss the major conceptual ideas, although we use many simplifications and abstractions to keep the material manageable in length and complexity. (The hardware models in this chapter have been sourced by the authors and do not imply ARM-endorsed architectures.)

The Big Picture

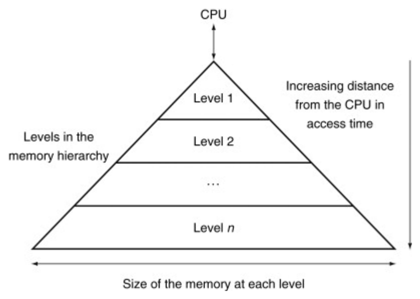
Programs exhibit both temporal locality, the tendency to reuse recently accessed data items, and spatial locality, the tendency to reference data items that are close to other recently accessed items. Memory hierarchies take advantage of temporal locality by keeping more recently accessed data items closer to the processor. Memory hierarchies take advantage of spatial locality by moving blocks consisting of multiple contiguous words in memory to upper levels of the hierarchy.

The figure below shows that a memory hierarchy uses smaller and faster memory technologies close to the processor. Thus, accesses that hit in the highest level of the hierarchy can be processed quickly. Accesses that miss go to lower levels of the hierarchy, which are larger but slower. If the hit rate is high enough, the memory hierarchy has an effective access time close to that of the highest (and fastest) level and a size equal to that of the lowest (and largest) level.

In most systems, the memory is a true hierarchy, meaning that data cannot be present in level *i* unless they are also present in level *i* + 1.

Figure 5.1.2: This diagram shows the structure of a memory hierarchy: as the distance from the processor increases, so does the size (COD Figure 5.3).

This structure, with the appropriate operating mechanisms, allows the processor to have an access time that is determined primarily by level 1 of the hierarchy and yet have a memory as large as level *n*. Maintaining this illusion is the subject of this chapter. Although the local disk is normally the bottom of the hierarchy, some systems use tape or a file server over a local area network as the next levels of the hierarchy.


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

PARTICIPATION ACTIVITY 5.1.5: Check yourself: Memory hierarchy.

Which of the following statements are generally true?

1) Memory hierarchies take advantage of temporal locality.
   - True
   - False

2) On a read, the value returned depends on which blocks are in the cache.
   - True
   - False

3) Most of the cost of the memory hierarchy is at the highest level.
   - True
   - False

4) Most of the capacity of the memory hierarchy is at the lowest level.
   - True
   - False

Provide feedback on this section