

<https://youtu.be/mY3j60e4ZIs>

Problems with physical memory

Virtual memory is a very useful concept in computer architecture because it helps with making your software work well given the configuration of the respective hardware on the computer it is running on.

The idea of virtual memory stems back from a (not so long ago) time, when the random access memory (RAM) of most computers was severely limited. Programmers needed to treat memory as a precious resource and use it most efficiently. Also, they wanted to be able to run programs even if there was not enough RAM available. At the time of writing (August 2019), the amount of RAM is no longer a large concern for most computers and programs usually have enough memory available to them. But in some cases, for example when trying to do video editing or when running multiple large programs at the same time, the RAM memory can be exhausted. In such a case, the computer can slow down drastically.

- There are several other memory-related problems, that programmers need to know about:
- Holes in address space** : If several programs are started one after the other and then shortly afterwards some of these are terminated again, it must be ensured that the freed-up space in between the remaining programs does not remain unused. If memory becomes too fragmented, it might not be possible to allocate a large block of memory due to a large-enough free contiguous block not being available any more.
 - Programs writing over each other** : If several programs are allowed to access the same memory address, they will overwrite each others' data at this location. In some cases, this might even lead to one program reading sensitive information (e.g. bank account info) that was written by another program. This problem is of particular concern when writing concurrent programs which run several threads at the same time.

The basic idea of virtual memory is to separate the addresses a program may use from the addresses in physical computer memory. By using a mapping function, an access to (virtual) program memory can be redirected to a real address which is guaranteed to be protected from other programs.

In the following, you will see, how virtual memory solves the problems mentioned above and you will also learn about the concepts of memory pages, frames and mapping. A sound knowledge on virtual memory will help you understand the C++ memory model, which will be introduced in the next lesson of this course.

Quiz

On a 32-bit machine, each program has its own 32-bit address space. When a program wants to access a memory location, it must specify a 32-bit address, which directs it to the byte stored at this location. On a hardware level, this address is transported to the physical memory via a parallel bus with 32 cables, i.e. each cable can either have the information 'high voltage', and 'low voltage' (or '1' and '0').

QUIZ QUESTION

How large is the address space on a 32-bit system? What is the upper limit for program memory in GB?

☐ 1 GB

☐ 2 GB

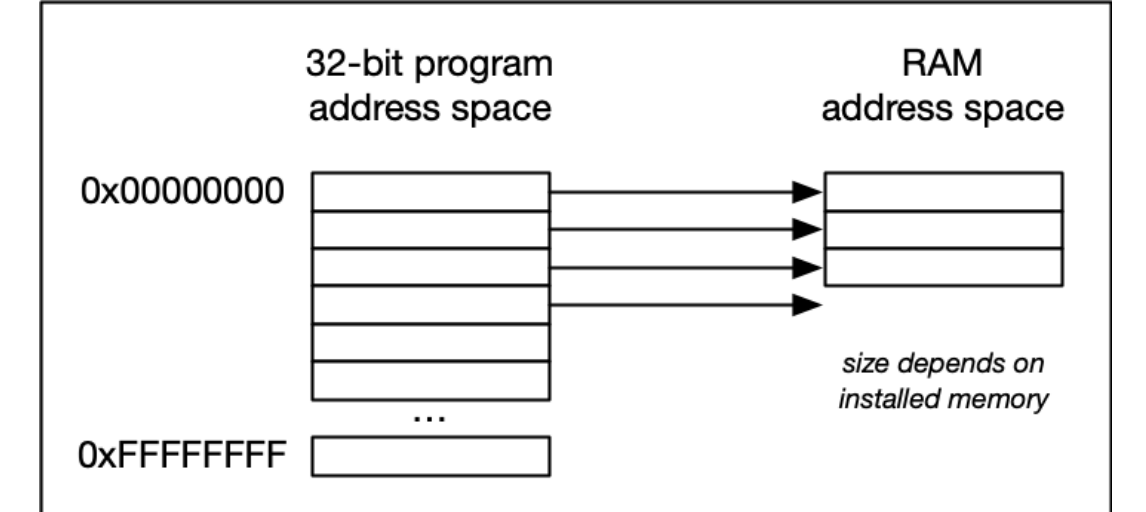
☒ 4 GB

☐ 8 GB

SUBMIT

Expanding the available memory

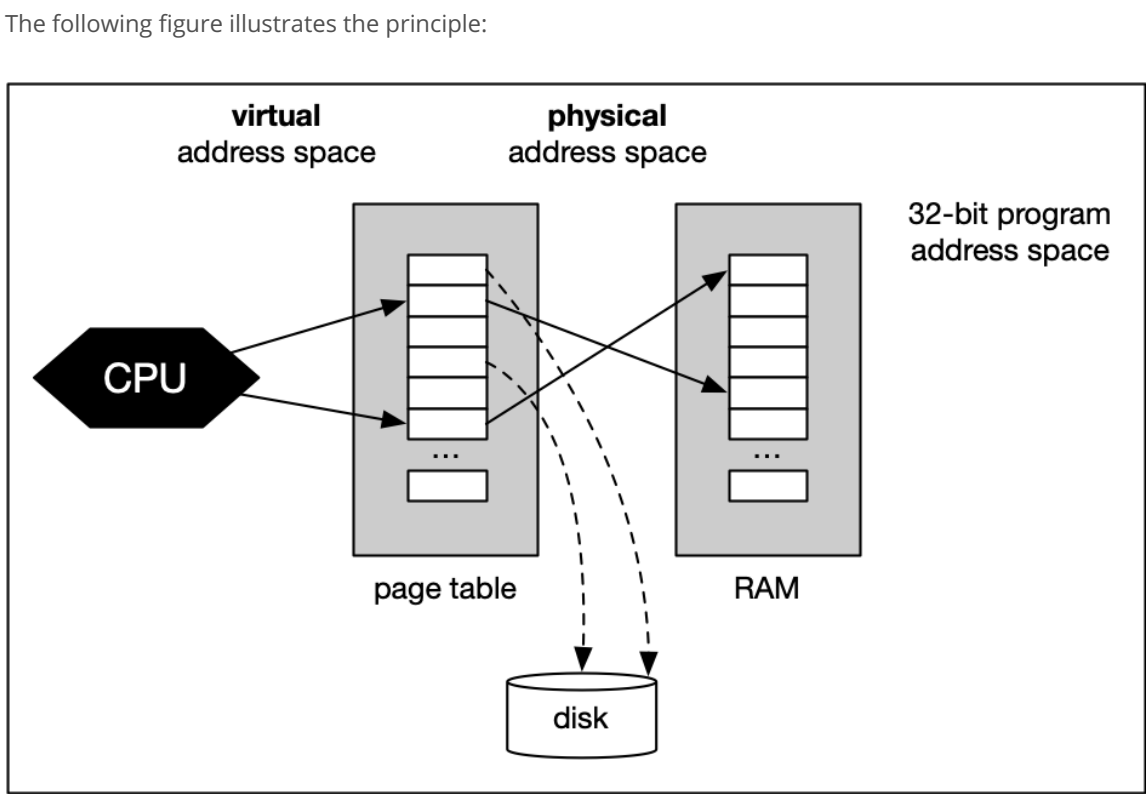
As you have just learned in the quiz, the total amount of addressable memory is limited and depends on the architecture of the system (e.g. 32-bit). But what would happen if the available physical memory was below the upper bound imposed by the architecture? The following figure illustrates the problem for such a case:



In the image above, the available physical memory is less than the upper bound provided by the 32-bit address space.

On a typical architecture such as MIPS ("Microprocessor without interlocked pipeline stages"), each program is promised to have access to an address space ranging from 0x00000000 up to 0xFFFFFFFF. If however, the available physical memory is only 1GB in size, a 1-on-1 mapping would lead to undefined behavior as soon as the 3b-bit RAM address space were exceeded.

With virtual memory however, a mapping is performed between the virtual address space a program sees and the physical addresses of various storage devices such as the RAM but also the hard disk. Mapping makes it possible for the operating system to use physical memory for the parts of a process that are currently being used and back up the rest of the virtual memory to a secondary storage location such as the hard disk. With virtual memory, the size of RAM is not the limit anymore as the system hard disk can be used to store information as well.



With virtual memory, the RAM acts as a cache for the virtual memory space which resides on secondary storage devices. On Windows systems, the file `pagefile.sys` is such a virtual memory container of varying size. To speed up your system, it makes sense to adjust the system settings in a way that this file is stored on an SSD instead of a slow magnetic hard drive, thus reducing the latency. On a Mac, swap files are stored in `/private/var/vm/`.

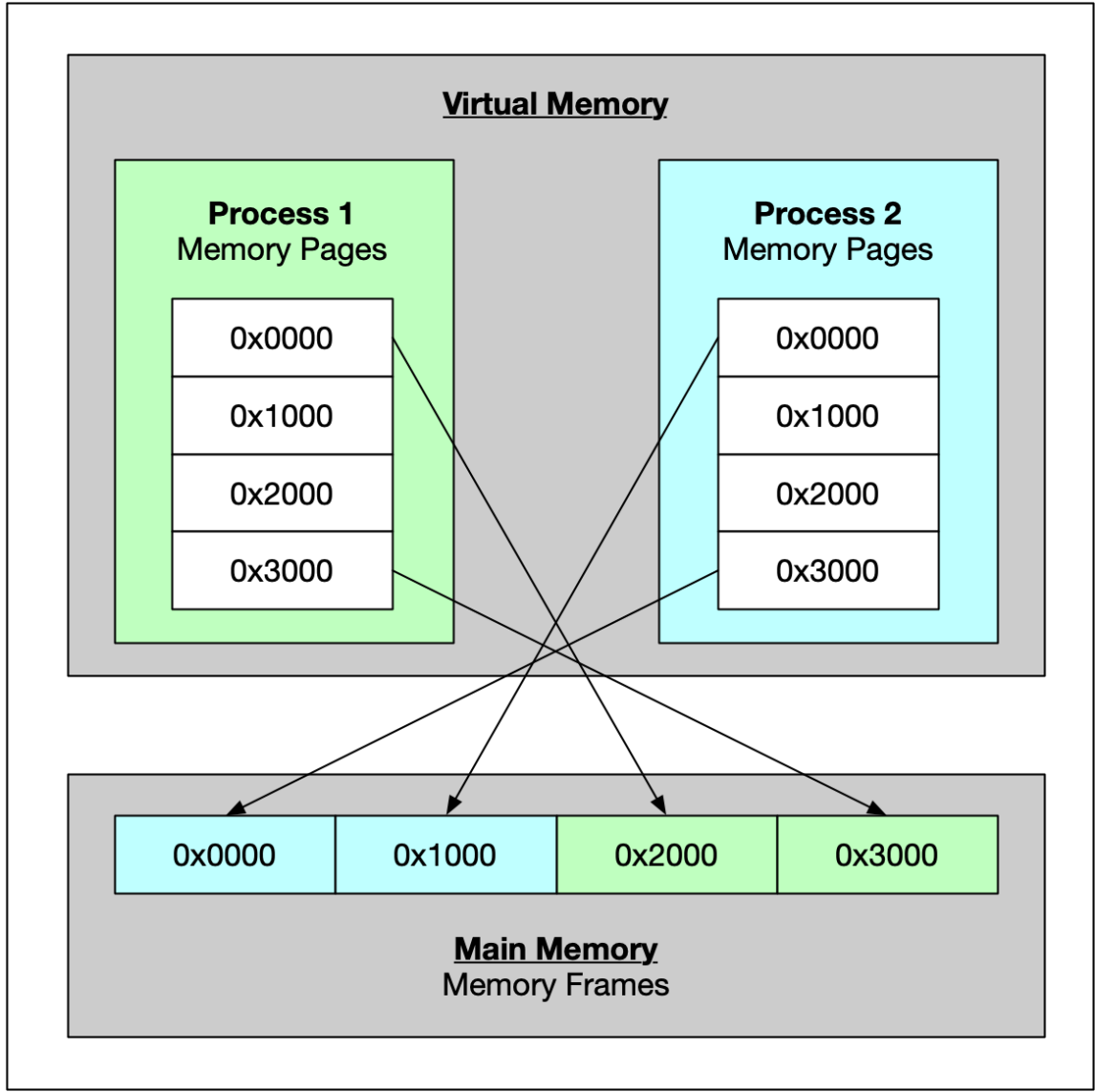
In a nutshell, virtual memory guarantees us a fixed-size address space which is largely independent of the system configuration. Also, the OS guarantees that the virtual address spaces of different programs do not interfere with each other.

The task of mapping addresses and of providing each program with its own virtual address space is performed entirely by the operating system, so from a programmer's perspective, we usually don't have to bother much about memory that is being used by other processes.

Before we take a closer look at an example though, let us define two important terms which are often used in the context of caches and virtual memory:

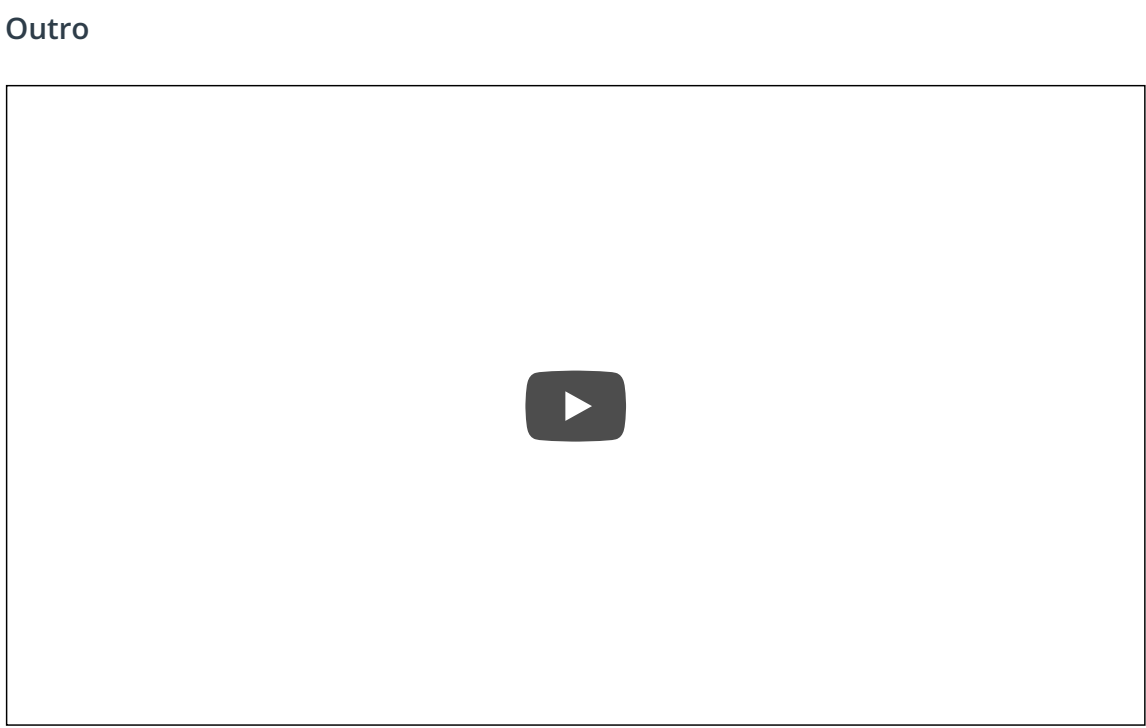
- A **memory page** is a number of directly successive memory locations in virtual memory defined by the computer architecture and by the operating system. The computer memory is divided into memory pages of equal size. The use of memory pages enables the operating system to perform virtual memory management. The entire working memory is divided into tiles and each address in this computer architecture is interpreted by the Memory Management Unit (MMU) as a logical address and converted into a physical address.
- A **memory frame** is mostly identical to the concept of a memory page with the key difference being its location in the physical main memory instead of the virtual memory.

The following diagram shows two running processes and a collection of memory pages and frames:



As can be seen, both processes have their own virtual memory space. Some of the pages are mapped to frames in the physical memory and some are not. If process 1 needs to use memory in the memory page that starts at address 0x1000, a page fault will occur if the required data is not there. The memory page will then be mapped to a vacant memory frame in physical memory. Also, note that the virtual memory addresses are not the same as the physical addresses. The first memory page of process 1, which starts at the virtual address 0x0000, is mapped to a memory frame that starts at the physical address 0x2000.

In summary, virtual memory management is performed by the operating system and programmers do usually not interfere with this process. The major benefit is a unique perspective on a chunk of memory for each program that is only limited in its size by the architecture of the system (32 bit, 64 bit) and by the available physical memory, including the hard disk.



https://youtu.be/yaDdy3j_IYE