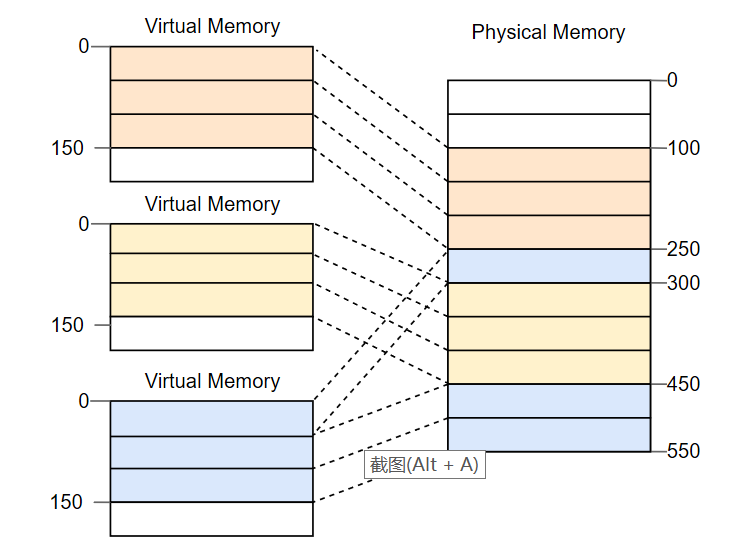
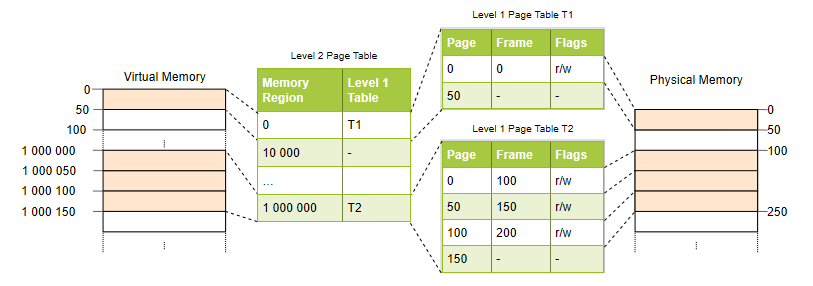
OS: Memory allocation

The memory allocator in OS takes the responsibility of properly distribute, release and protect the memory zoom. Nowadays, there are several frequently used memory allocation strategies, containing relocation, segmentation, paging and virtual memory. There are plenty of memory allocation structures deduced by those strategies, all of them have their pros and cons. In this project, we mainly focus on the **multi-level page tables** as well as **virtual memory** to implement our allocator. The advantages of such model are stated below.

One main task of our allocator is to isolate programs from each other, which means we need to ensure that the memory area of one process will never be accessible for other process while it is still working. To achieve this goal we need a memory protection technique. Two powerful techniques are segmentation and paging. While segmentation uses variable-sized memory regions and suffers from external fragmentation, the paging uses fixed-sized pages and allows much more fine-grained control over access permissions. Though paging strategy still have the problem of inner fragmentation, it is not compatible with the external fragmentation caused by segmentation, as well as the memory relocation problems. While a linear one-level page table suffers the problem of its large size, the multi-level page tables can minimize the memory occupied by the table itself by dividing and indexing. The other aspect is the usage of virtual memory, Virtual memory is a technique that allows the execution of processes that are not completely in memory. One major advantage of this scheme is that programs can be larger than physical memory. Since the execution of a program needs continuous memory, we can allocate them with continuous virtual memory then map the virtual memory to the physical memory where the mapping can be discrete, in this way we make full use of memory space, as shown in the diagram below.



The mechanism of our model can be interpreted by the diagram below:

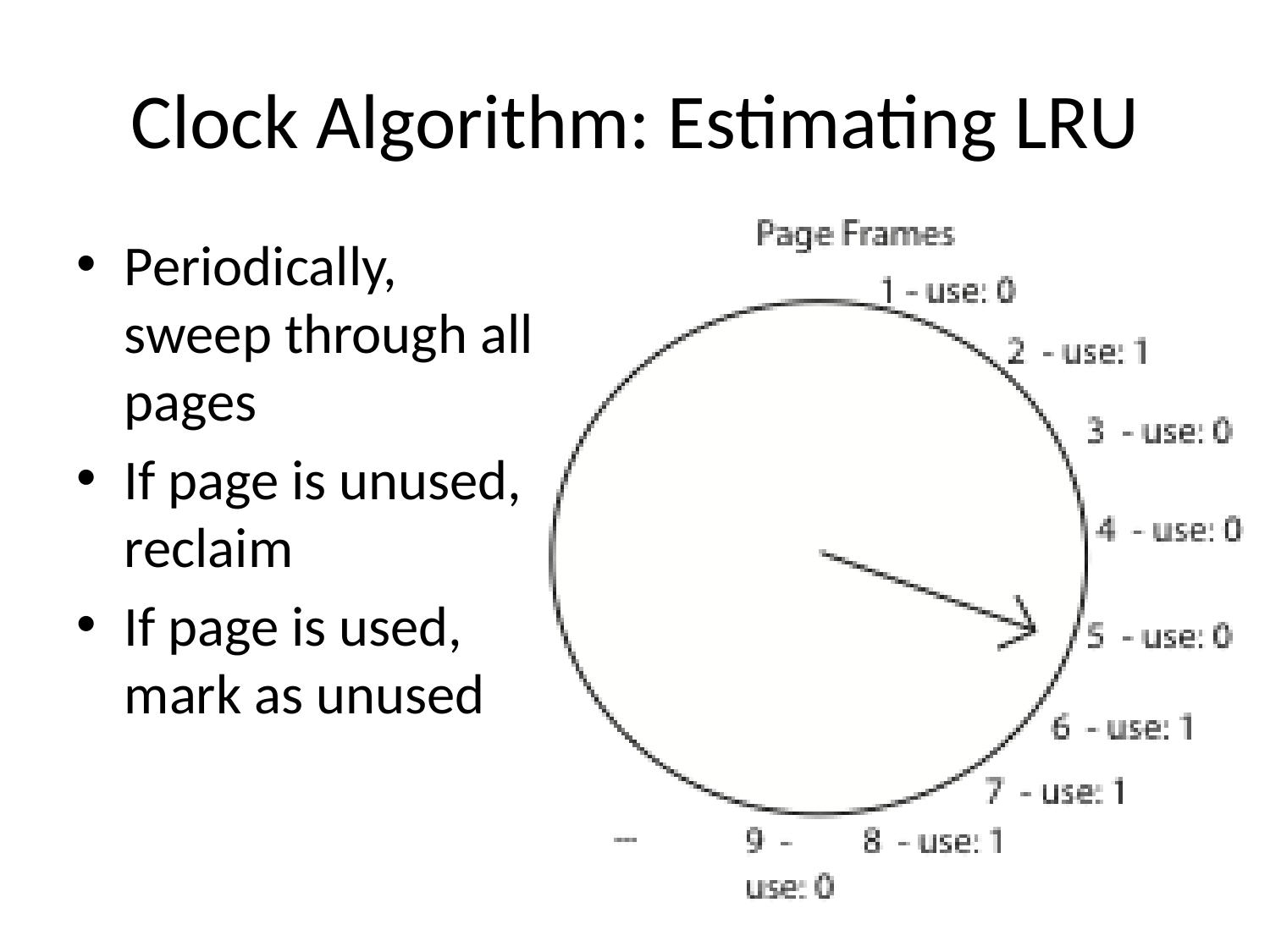


The core idea of the model is we allocate virtual memory to programs then use the multi-level page table iteratively to access its mapping physical memory. When we access a page but can not find its corresponding physical memory (page fault), we swap some pages in physical memory with the pages we need in hard disk based on our paging swapping algorithm.

The implementation of this memory allocation model can be mainly divided into three parts:

1. The implementation of virtual memory
2. The MMU (Memory Management Unit) system, which has the function of translating the virtual address to physical address.
3. Paging system with swapping technique

We will use some powerful data structures like double linked list and heap. The final outcome depends on their performance. Moreover, an essential part in the implementation is about the paging swapping algorithm. We now decide to use Clock page replacement algorithm to implement this function, the implementation diagram of this algorithm shows below.



There are other swapping algorithms like Optimal Page Replacement Algorithm, Not Recently Used Replacement Algorithm, First-In,First-Out Page Replacement Algorithm, Second Chance Page Replacement Algorithm and LRU Page Replacement Algorithm. Any further adjustment could be done according to the performance of our algorithm.

Reference

1. <https://os.phil-opp.com/paging-introduction/> Introduction to paging
2. <https://blog.csdn.net/qq_32635069/article/details/74838187>

Figures cited from

1. <https://os.phil-opp.com/paging-introduction/> Introduction to paging
2. https://slide-finder.com/view/Caching-and-Virtual-Memory.254456.html