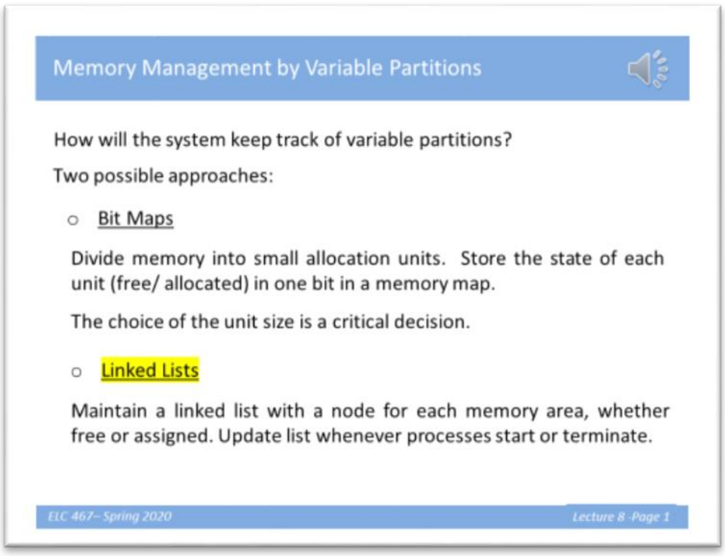


Memory Management by Variable Partitions

How will the system keep track of variable partitions?

Two possible approaches:

- Bit Maps

  Divide memory into small allocation units. Store the state of each unit (free/ allocated) in one bit in a memory map.

  The choice of the unit size is a critical decision.

- Linked Lists

  Maintain a linked list with a node for each memory area, whether free or assigned. Update list whenever processes start or terminate.

ELC 467– Spring 2020 Lecture 8 -Page 1

Slide 1:

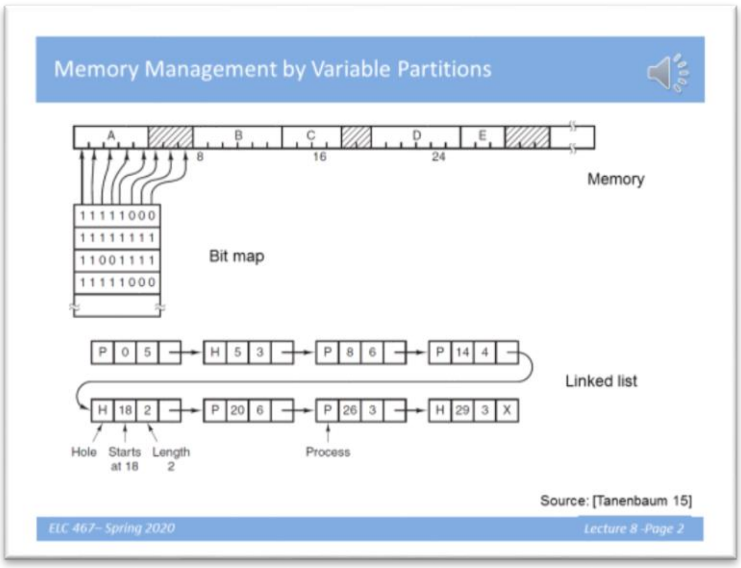
In fixed partitions, the system needs only to maintain a fixed table with start and end addresses of partitions. When partitions vary dynamically, a more complex data structure is needed.

Slide 2:

Here, the system will maintain a map of memory indicating free and assigned areas. If this map shows the status of each address (byte), its size will be too large, taking itself a large portion of memory. Thus, map deals with larger units (blocks of addresses) to make its size more compact. Process is assigned an integer number of these allocation units, as map does not handle fractions of units. Unit should not be too large, as process may waste memory if assigned a large unit not used completely.

Slide 4:

In this alternative method system uses a linked list with a node for every memory partition, whether free or assigned. As partitions are created, split, and merged, the linked list is updated to reflect these changes.



Memory Management by Variable Partitions

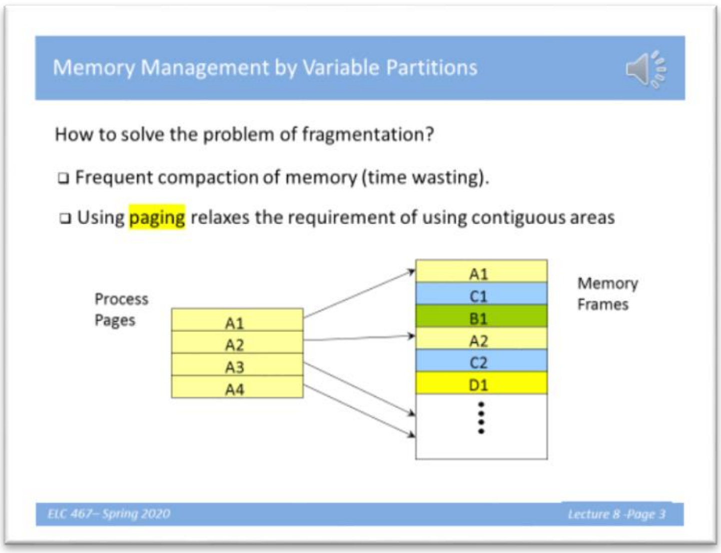
Slide 4:

Slide 3:

Note here how memory is divided into units, with a bit in map corresponding to each unit. Process A is assigned the first five units. This is reflected in map by five 1 bits. There is next three free units shown in map by three zero bits,… etc. A search of a free partition with a given size will be a search for a specific number of adjacent zero bits in map.

Slide 5:

Linked list has a node for each area assigned to process (P) or free or hole (H). Node indicates length of partition and where it starts. Note that units are not necessary for the linked list method and used here for illustration only. If for example process B terminates, the area it occupies will be merged with the preceding hole, forming a single hole of length 9.



Memory Management by Variable Partitions

How to solve the problem of fragmentation?

- ❑ Frequent compaction of memory (time wasting).
- ❑ Using paging relaxes the requirement of using contiguous areas

Process Pages
A1
A2
A3
A4
A1
C1
B1
A2
C2
D1
Memory Frames

Slide 6:

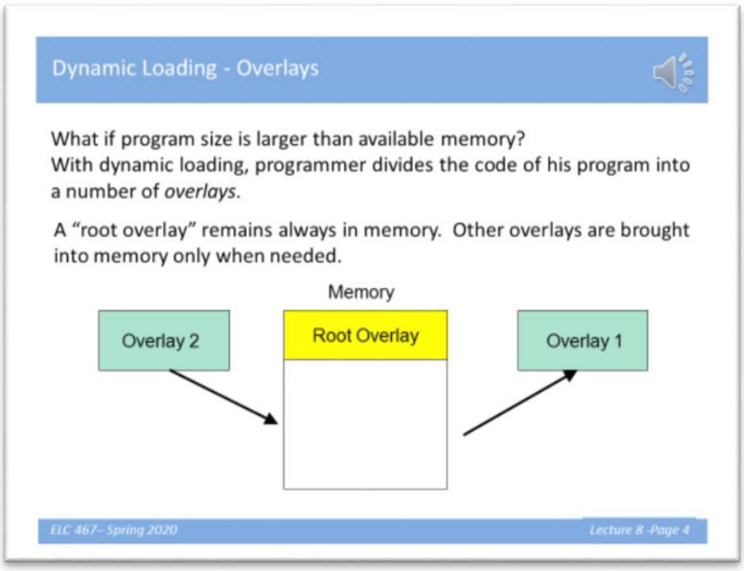
Fragmentation means that after operating for a while, free memory available becomes divided into small areas. Each of these areas cannot be used alone to run any process, since we still assume that process needs to run in an undivided area.

Slide 7:

This means to stop system operation occasionally and rewrite the memory contents into adjacent areas, thus grouping the free areas together. This usually results in long delays that slows down the system operation.

Slide 8:

With paging, memory contents are divided into fixed size pages. Physical RAM is also considered as a number of frames each having the size of a page. The page contents can be loaded into any free frame. For example, in the shown diagram first page of process A is loaded in first frame. However, second page is loaded in fourth frame since second and third frames are loaded with pages of processes B and C. Thus, memory area used by process no longer need to be undivided. We will consider paging in more detail later.



Dynamic Loading - Overlays

What if program size is larger than available memory?
With dynamic loading, programmer divides the code of his program into a number of *overlays*.

A "root overlay" remains always in memory. Other overlays are brought into memory only when needed.

Memory
Overlay 2
Root Overlay
Overlay 1

ELC 467– Spring 2020 Lecture 8 -Page 4

Slide 9:

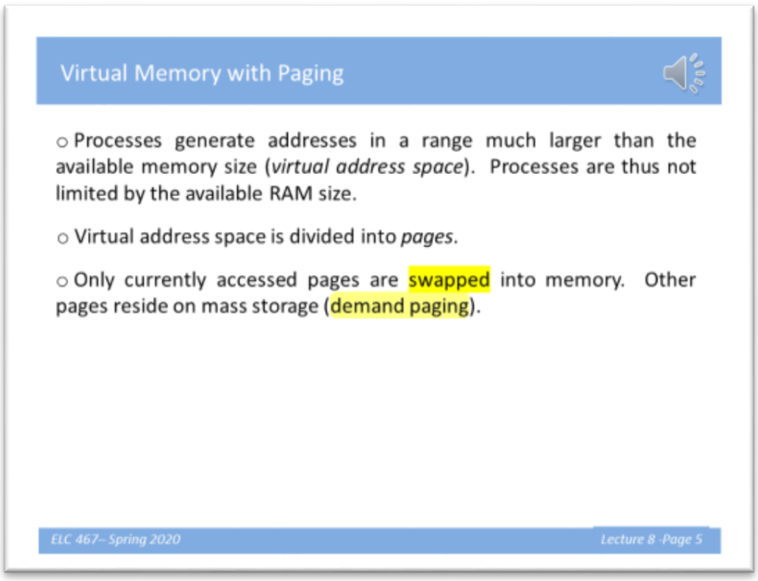
We also assume so far that process cannot run unless all the instructions and data it needs are loaded into memory. Of course, this limits the number of processes that can run concurrently. In fact, typically the process needs only a subset of this information at any time during its execution. It only accesses some instructions and some variables for some time, then moves to other instruction and data,…etc. This is known as the "principle of locality of access".

Slide 10:

One method that made use of this principle is the method of overlays used in some old systems (e.g. DOS). Compilers under these systems allow user to compile his program into a number of files used one at a time, instead of compiling it into a single large executable file.

Slide 11:

For example, a word processor program may have an overlay file for editing functions, and another overlay file for spelling check. Since these will not be used together, this allows process to run from a smaller memory area. However, this method depends on the programmer for dividing the code and does not do this automatically. Also, it applies for instructions and not data. These disadvantages are solved in the method of virtual memory considered next.



Virtual Memory with Paging

- Processes generate addresses in a range much larger than the available memory size (*virtual address space*). Processes are thus not limited by the available RAM size.
- Virtual address space is divided into *pages*.
- Only currently accessed pages are swapped into memory. Other pages reside on mass storage (demand paging).

ELC 467– Spring 2020 Lecture 8 -Page 5

Slide 12:

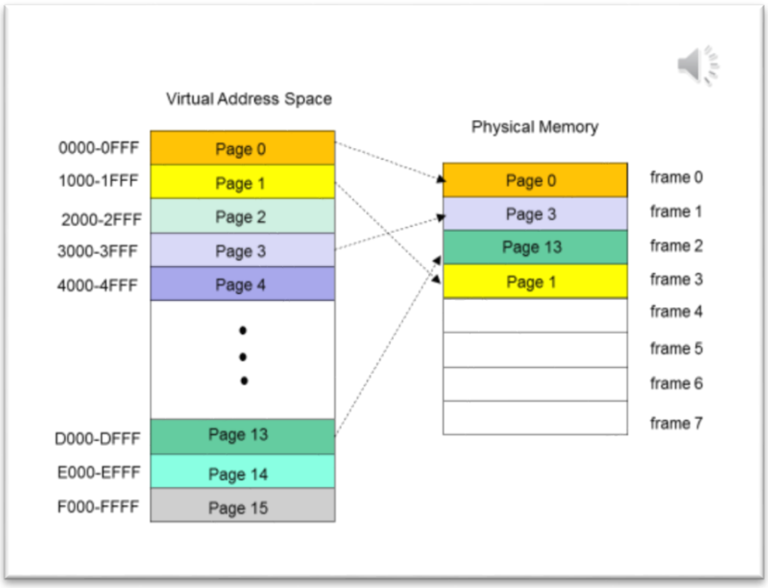
We consider the method of virtual memory combined with paging since this is how it is used in current processors. The method of virtual memory allows processes to run without restrictions resulting from the available physical RAM size.

Slide 13:

Each process can access addresses in a very large address space (typically of terabytes) irrespective of actual RAM size, and number of processes concurrently sharing the RAM. This is known as the virtual address space.

Slide 14:

Demand paging means that page will be brought into a RAM frame (or swapped in it) only when process starts to access its contents. If page is not accessed, it remains on disk (in a special swapping file). Thus, in a sense, a part of disk acts as an extension to RAM.



Slide 15:

We consider this example with very small sizes for illustration only. Here, each process behaves as if it has access to 64K of memory, although actual RAM size is 32K. Page size is 4K, thus virtual space has 16 pages and physical memory has 8 frames.

Slide 16:

Process starts to access addresses in page 0 (e.g. to fetch instructions). Thus, the contents of this page are swapped into a free frame, for example frame 0. Addresses used by the process will be from 0000 to 0FFF. These are the same physical addresses of frame 0, so no translation from virtual to physical addresses will be necessary.

Slide 17:

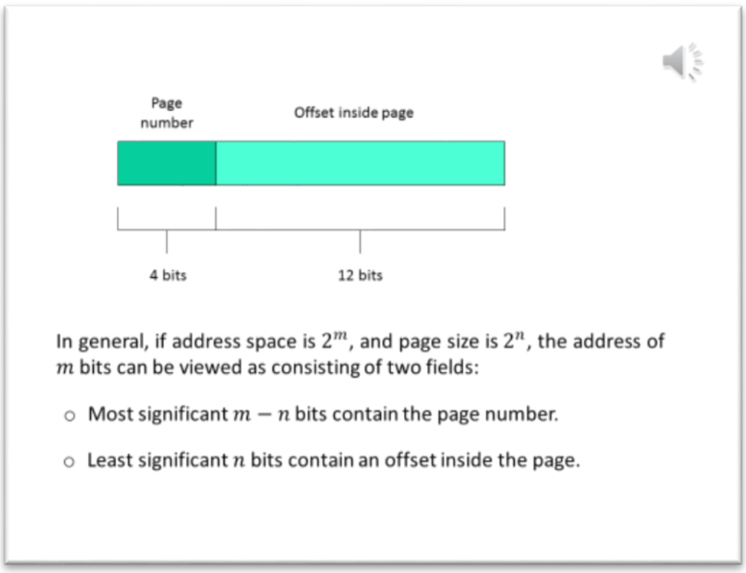
Now process reads data from the address range of page 3. This is now swapped into frame 1. Note that address translation from the virtual range of page 3 (3000-3FFF) to the physical range of frame 1 (1000-1FFF) is now necessary with each access.

Slide 18:

Process now accesses the stack while the stack pointer is pointing to an address in page 13. Thus, page 13 is brought into frame 2. Again, page translation will be necessary.

Slide 19:

Process now fetched all instruction in page 0 and starts fetching instructions from page 1. This is swapped into frame 3. The other eleven virtual pages will not be brought into RAM frames until demanded by the process.

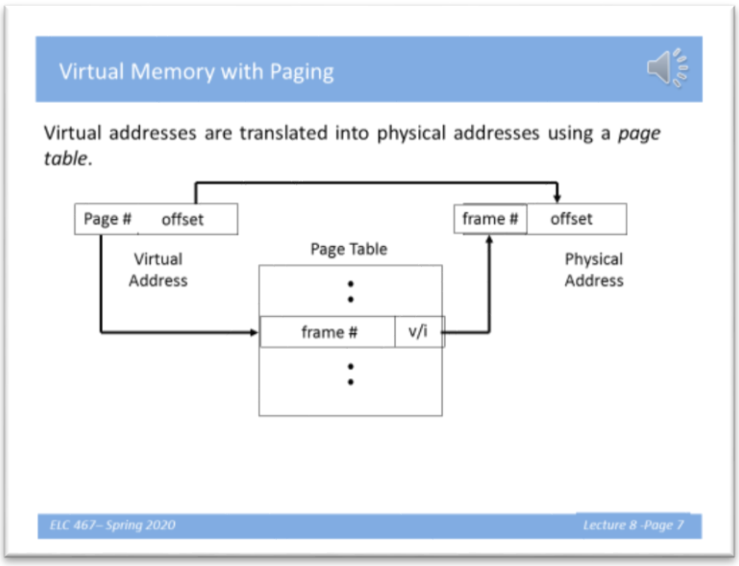


In general, if address space is 2^m, and page size is 2^n, the address of m bits can be viewed as consisting of two fields:

- Most significant $m - n$ bits contain the page number.
- Least significant n bits contain an offset inside the page.

Slide 20:

By inspection of addresses in the previous simplified example, it is seen that most significant hexadecimal digit of address (4 bits) is actually the page number. Other digits give a range of offsets that is repeated in each page. This can be generalized as shown here.



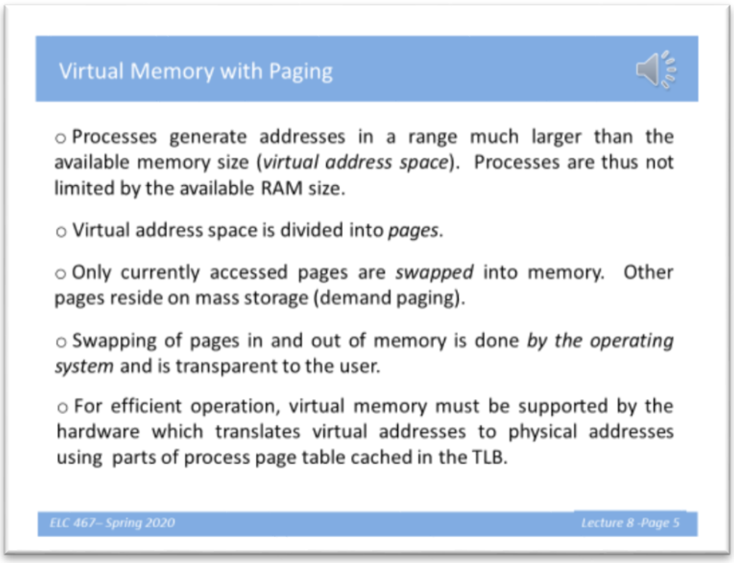
Virtual Memory with Paging

Virtual addresses are translated into physical addresses using a *page table*.

ELC 467– Spring 2020 Lecture 8 -Page 7

Slide 21:

Translation will be in fact simple. To translate virtual address into physical address, we need to know the number of frame in which page is loaded, then simply replace the page number with the frame number, keeping offset the same. More on this slide later.



Virtual Memory with Paging

- Processes generate addresses in a range much larger than the available memory size (*virtual address space*). Processes are thus not limited by the available RAM size.
- Virtual address space is divided into *pages*.
- Only currently accessed pages are *swapped* into memory. Other pages reside on mass storage (demand paging).
- Swapping of pages in and out of memory is done *by the operating system* and is transparent to the user.
- For efficient operation, virtual memory must be supported by the hardware which translates virtual addresses to physical addresses using parts of process page table cached in the TLB.

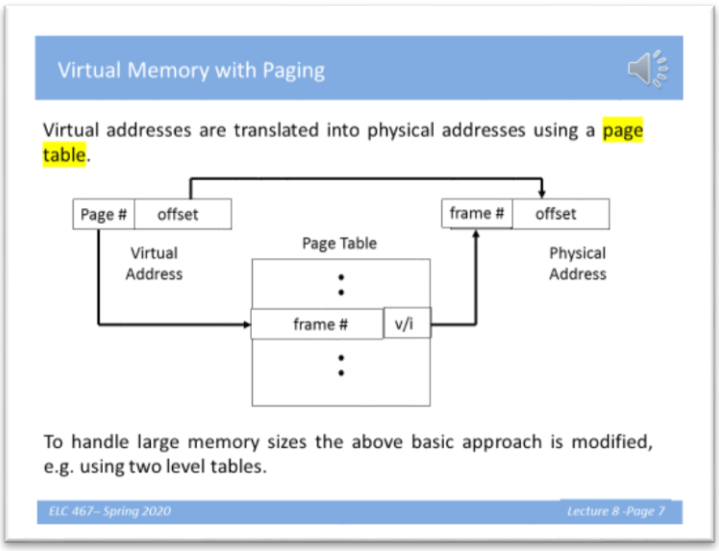
ELC 467– Spring 2020 Lecture 8 -Page 5

Slide 22:

The user needs not be concerned by the process of swapping of demanded pages. This will be the responsibility of the operating system. In fact, user will not be aware of this process unless heavy swapping causes the system to be somewhat slower.

Slide 23:

If continuous translation of addresses is done by software, system will be too slow to be practical. Thus, processor hardware must support this translation process. A page table holds where each virtual page is loaded in physical RAM. This is stored (at least partially) inside processor in the so called TLB, which stands for Translation Lookaside Buffer.



Virtual Memory with Paging

Virtual addresses are translated into physical addresses using a page table.

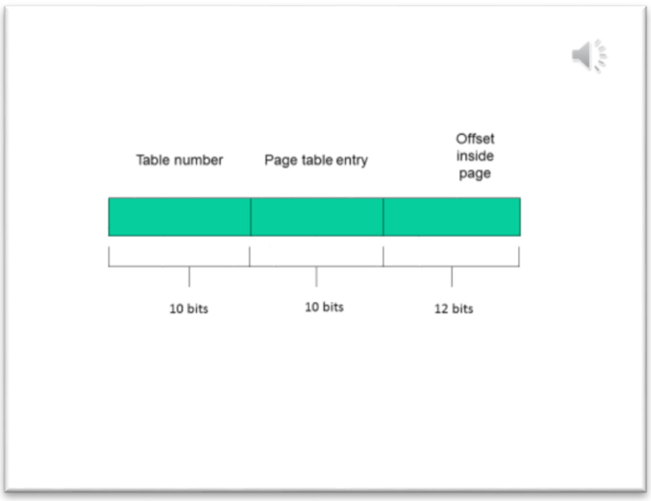
To handle large memory sizes the above basic approach is modified, e.g. using two level tables.

Slide 24:

A straightforward structure of page table will have an entry for each page. A valid/invalid bit is set if page is present in memory and is 0 otherwise. In the first case, the number of frame in which page is loaded is written in table.

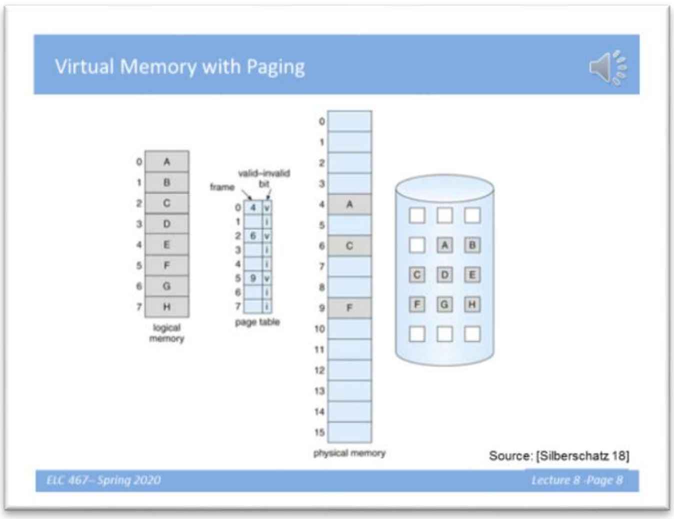
Slide 25:

With actual huge address spaces with large number of pages, table constructed in this way will be too large to be stored within processor. Thus this simple form need to be modified.



Slide 26:

For example, consider the case of 32-bit virtual addresses and 4K pages. The 4G address space will have million virtual pages, and page table should have million entries. Instead, we may consider that we have 1024 tables, with 1024 entries each. This corresponds to the shown fields of address. Benefiting again from the locality of access, we hold one table at a time within the processor.

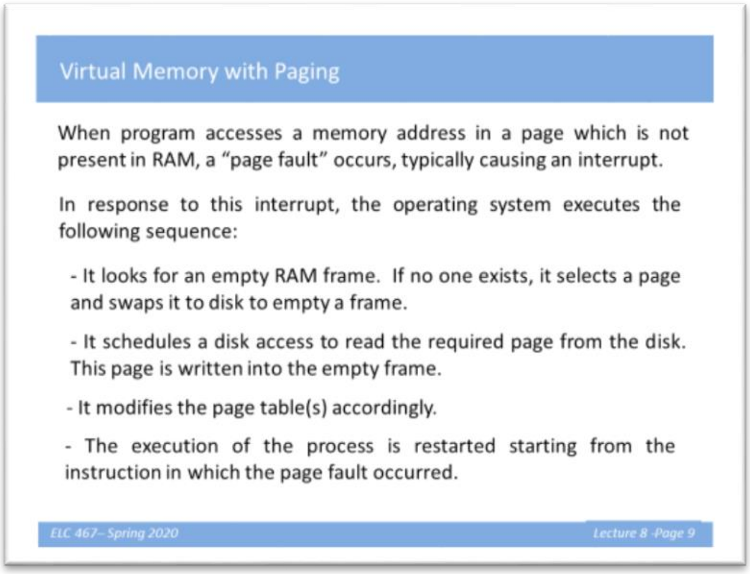


Virtual Memory with Paging

Source: [Silberschatz 18]

Slide 27:

This is another illustration of the previous discussion. Virtual memory can also be called logical memory, as opposed to physical memory. Here, process has 8 pages of which three pages A,C, and F are swapped from disk shown on the right into physical memory frames. Note that there are two copies of the contents of each of the three demanded pages: one on disk and one in RAM.



Virtual Memory with Paging

When program accesses a memory address in a page which is not present in RAM, a "page fault" occurs, typically causing an interrupt.

In response to this interrupt, the operating system executes the following sequence:

- It looks for an empty RAM frame. If no one exists, it selects a page and swaps it to disk to empty a frame.
- It schedules a disk access to read the required page from the disk. This page is written into the empty frame.
- It modifies the page table(s) accordingly.
- The execution of the process is restarted starting from the instruction in which the page fault occurred.

Slide 28:

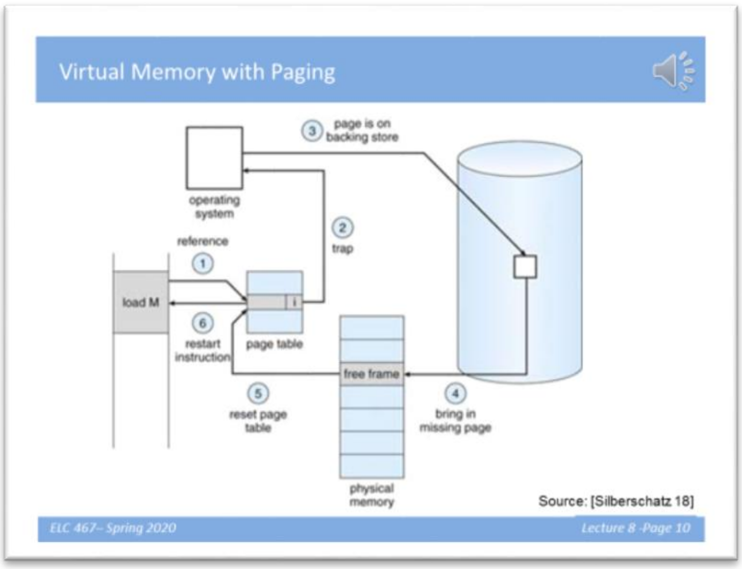
Now we explain how page swapping is handled. If an address in a page not present in physical memory is accessed, a page fault interrupt occurs.

Slide 29:

We will consider later which page should be swapped out of RAM if no frame is available.

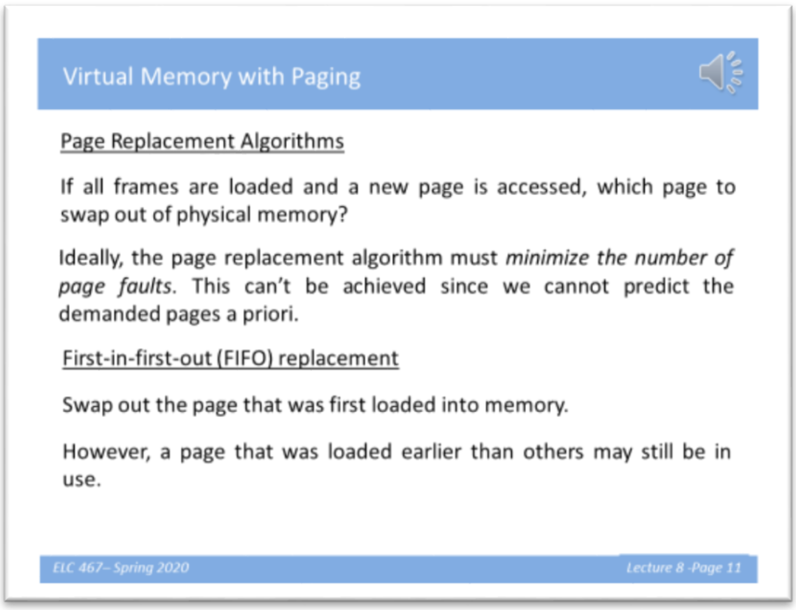
Slide 30:

The term "schedules an access" is used since probably other disk access operation is currently taking place. We will understand this more when we study file management.



Slide 32:

You should be able now to understand the steps in this diagram. The term “trap” in step 2 is usually used to refer to interrupts resulting from errors, a page fault in this case.



Virtual Memory with Paging

Page Replacement Algorithms

If all frames are loaded and a new page is accessed, which page to swap out of physical memory?

Ideally, the page replacement algorithm must *minimize the number of page faults*. This can’t be achieved since we cannot predict the demanded pages a priori.

First-in-first-out (FIFO) replacement

Swap out the page that was first loaded into memory.

However, a page that was loaded earlier than others may still be in use.

ELC 467– Spring 2020 Lecture 8 -Page 11

Slide 34:

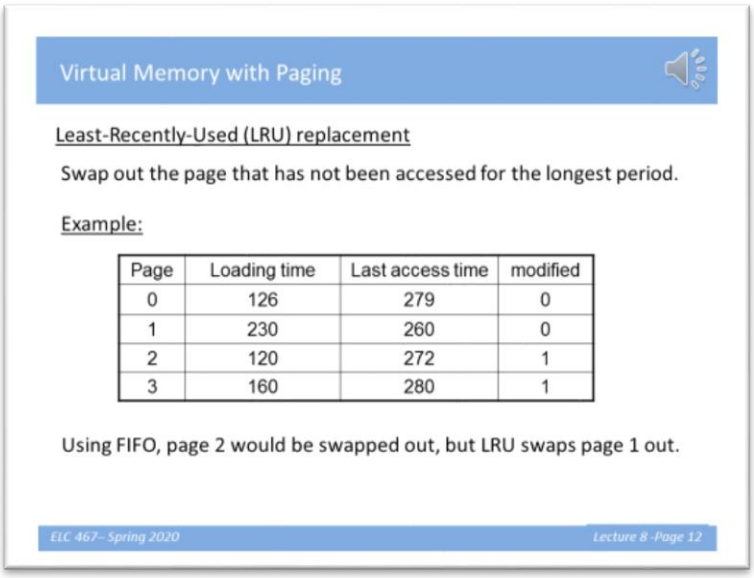
Ideally, the page swapped out should be a page that will not be accessed or demanded again. Unfortunately, system will not be able to know exactly which addresses the process will access in the future. Note that if page contents were modified while in RAM, copy of page on disk need to be updated when swapped out causing an additional disk write operation.

Slide 35:

FIFO is based on the assumption that the page that was first brought into memory was there for a long time, and is expected to be no longer needed. On the other hand, a page just swapped in is most probably still in use.

Slide 36:

This assumption is not necessarily true. For example, a page containing the instructions of a loop with many iterations will remain in use for long time, although loaded earlier than other pages.



Virtual Memory with Paging

Least-Recently-Used (LRU) replacement

Swap out the page that has not been accessed for the longest period.

Example:

Page	Loading time	Last access time	modified
0	126	279	0
1	230	260	0
2	120	272	1
3	160	280	1

Using FIFO, page 2 would be swapped out, but LRU swaps page 1 out.

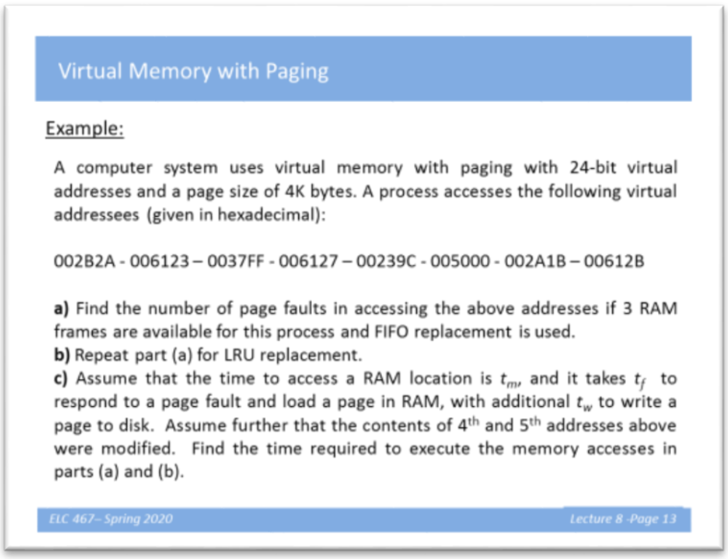
ELC 467– Spring 2020 Lecture 8 -Page 12

Slide 37:

This method takes into account whether page is still in use or not. It swaps out the page which was not used for a longer time.

Slide 38:

Note however that LRU can be implemented without storing the time of each access, which is not practical. We will consider some alternative methods in the problem set.



Virtual Memory with Paging

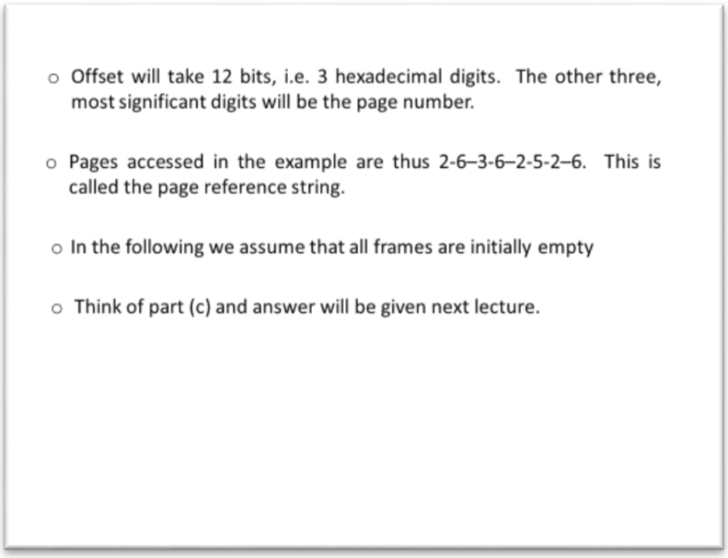
Example:

A computer system uses virtual memory with paging with 24-bit virtual addresses and a page size of 4K bytes. A process accesses the following virtual addressees (given in hexadecimal):

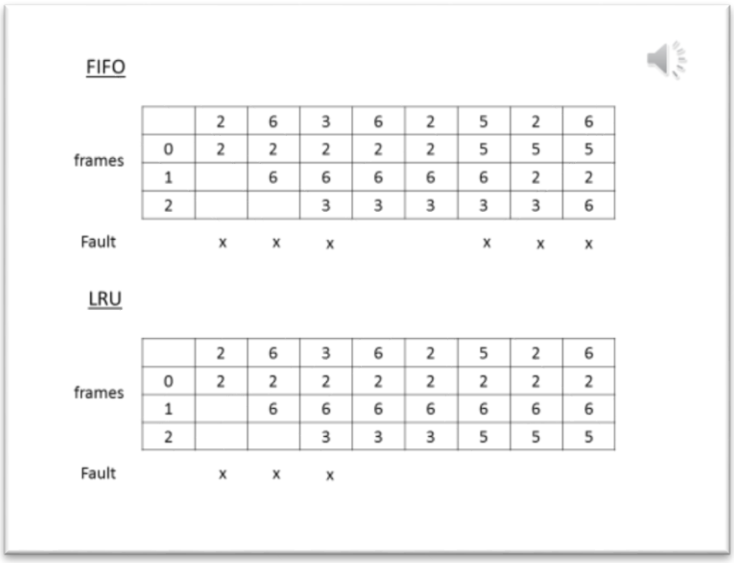
002B2A - 006123 – 0037FF - 006127 – 00239C - 005000 - 002A1B – 00612B

a) Find the number of page faults in accessing the above addresses if 3 RAM frames are available for this process and FIFO replacement is used.
b) Repeat part (a) for LRU replacement.
c) Assume that the time to access a RAM location is t_m, and it takes t_f to respond to a page fault and load a page in RAM, with additional t_w to write a page to disk. Assume further that the contents of 4th and 5th addresses above were modified. Find the time required to execute the memory accesses in parts (a) and (b).

ELC 467– Spring 2020 Lecture 8 -Page 13



- Offset will take 12 bits, i.e. 3 hexadecimal digits. The other three, most significant digits will be the page number.
- Pages accessed in the example are thus 2-6–3-6–2-5-2–6. This is called the page reference string.
- In the following we assume that all frames are initially empty
- Think of part (c) and answer will be given next lecture.



FIFO

		2	6	3	6	2	5	2	6
frames	0	2	2	2	2	2	5	5	5
	1		6	6	6	6	6	2	2
	2			3	3	3	3	3	6
Fault		x	x	x			x	x	x

LRU

		2	6	3	6	2	5	2	6
frames	0	2	2	2	2	2	2	2	2
	1		6	6	6	6	6	6	6
	2			3	3	3	5	5	5
Fault		x	x	x					

Slide 41:

In LRU, when page 5 was accessed the page that was not accessed for the longest time was page 3. Thus, this is the one swapped out, although loaded after pages 2 and 6, which were accessed after it.