Operating Systems
Memory Management

[2] **Memory Management**

What is expected by programmers from the system memory:

- to be huge,
- to be fast,
- to be nonvolatile.

Memory hierarchy:

- small fast expensive memory (e.g. cache),
- average in size, speed and expense memory (e.g. operating memory),
- huge, slow and cheap memory (e.g. disk/tape memory).

Memory Management (MM) type on the level of operating system is dictated by the computer system architecture.

[3] **Memory Management Organization**

Memory management depends on:

- structure of address field of instruction arguments,
- position of address field in instruction word,
- hardware capabilities of address field transformation.

Depending on the length of the address field **the address space** may be the same in size, smaller or bigger than the scope of operating memory addresses.

[4] **Operating System Functions**

Functions of the operating systems from the point of view of memory management:

- address space management through address translation mechanisms,
- memory contents protection,
- shared areas access control,

- effective organization of operating memory management.

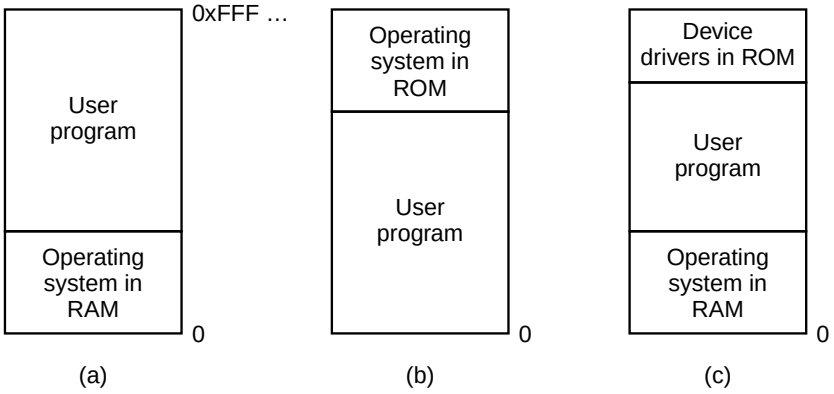
The size of the operating memory for operating system code is usually fixed, but the allocation process for user processes is usually much more complicated.

[5] **Memory Allocation Methods**

Methods of memory allocation to user processes:

1. **without division**, all the memory allocated to one process. Multiprogramming may be implemented by swapping,
2. **memory division**, free operating memory divided into blocks allocated to particular user processes,
3. usage of **virtual memory**, there exist one or more virtual address spaces allocated to user processes and without any direct obvious mapping to the physical address space.

[6] **M.A.M. Without Division**



Three simple ways of organizing memory for an operating system with one user process.

[7] **M.A.M. With Division**

The goals of the memory division:

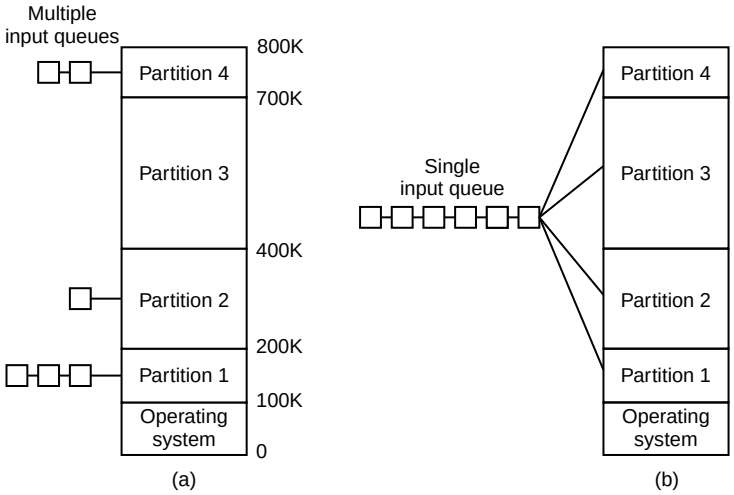
- better utilization of hardware elements of the system, mainly processor and memory,

- enabling fast context switching.

The following types of systems with memory division may be distinguished:

- systems with **static division** - either fixed partitions with different length or blocks with fixed length called **frames**.
- systems with **dynamic division** - implemented by structures describing **free blocks of memory** with variable length and by swapping mechanism.

[8] Static Division Based on Constant Size Partitions



a. fixed memory partitions with separate input queues for each partition,

b. fixed memory partitions with a single input queue.

[9] Dynamic Division

For implementation of dynamic memory allocation the following hypothetical functions:

- **allocate**(size, address)
  - choosing from the list of free blocks that one which covers demand,
  - returning the address of the chosen block and removing allocated block from the free block lists as a result of allocation,
  - **malloc**(size), **calloc**(n, size), **realloc**(ptr, size).

- **free**(address) - adding pointed block to the free block list, e.g. **free**(ptr),
- **msize**() - return the size of the biggest currently available free block, **msize**() (but: races possible).

[10] **MM in Multiprogramming Context**

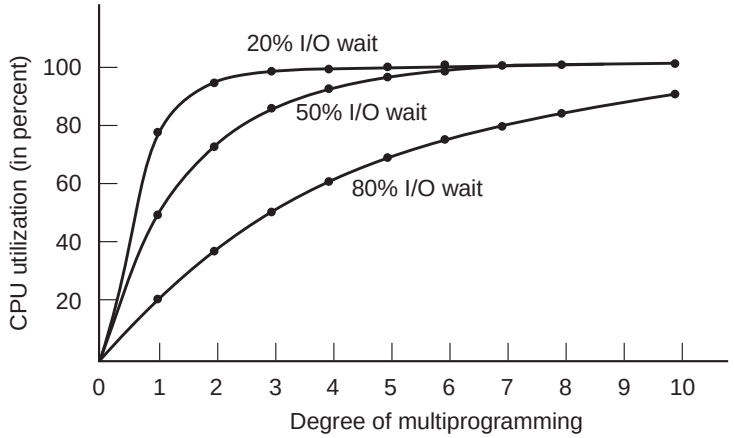
Multiprogramming has introduced two essential problems to be solved:

- relocation service - cannot be sure where program will be loaded in memory,
- mutual memory protection.

Possible solution: use of base and limit values:

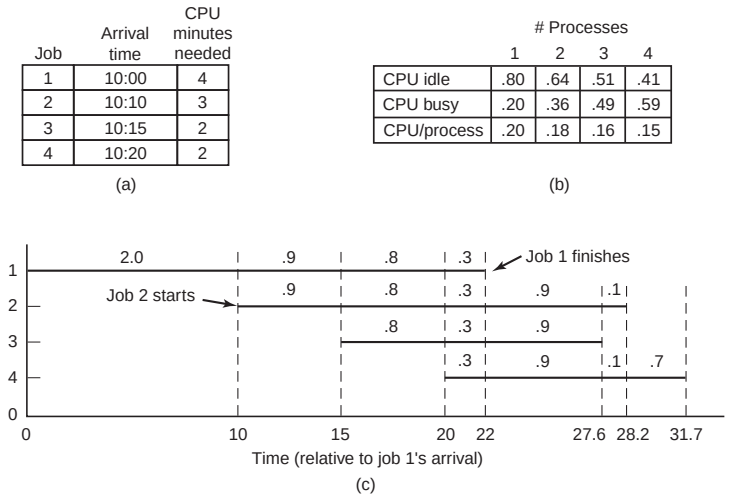
- address locations added to base value to map to physical address,
- address locations larger than limit value treated as an error.

[11] **Multiprogramming Modelling**



CPU utilization as a function of number of processes in memory (ignoring operating system overhead).

[12] **Multiprogramming Effectiveness Analisys**

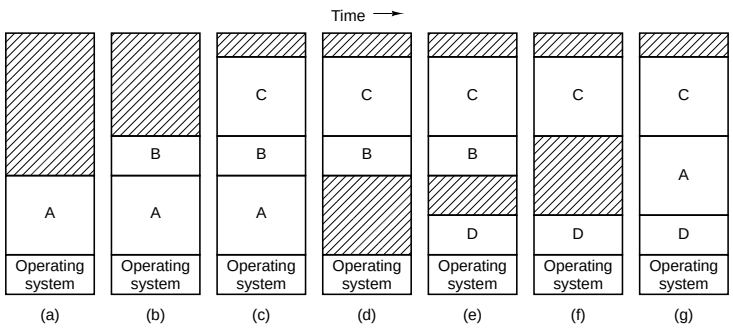


a. arrival and work requirements of 4 jobs,

b. CPU utilization for 1–4 jobs with 80% I/O wait,

c. sequence of events as jobs arrive and finish, numbers show amout of CPU time jobs get in each interval.

[13] **Swapping**



Memory allocation changes as processes come into memory and leave memory. Shaded regions are unused memory.

[14] **Fragmentation**

Choosing allocation algorithm the following aspects should be considered:

- effectiveness (speed),

- simplicity,
- fragmentation effect.

Internal fragmentation - situation when some pieces of memory although allocated as part of some bigger structure (partition, frame) are not to be used by user processes.

External fragmentation - situation when some pieces of memory although unallocated cannot be allocated for some reason (e.g. are too small) for user processes.

[15] Allocation Algorithms

The role of the allocation algorithm is to chose appropriate free block in order to allocate it to some user process.

Some allocation algorithms:

- **First Fit** algorithm, first which matches from the list,
- **Best Fit** algorithm, the optimal in size from the list,
- **Worst Fit** algorithm, the biggest from the list,
- **Buddies** algorithm, memory (with size 2^k) division into two equal in size blocks; dividing by two till obtaining minimal yet big enough block.

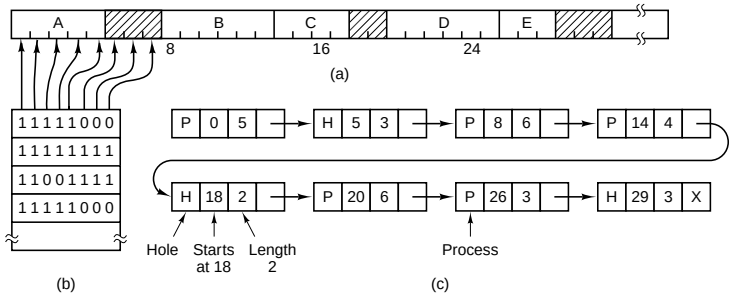
[16] Anti-fragmentation Countermeasure

In the system there may be defined value Δn as a minimal allowed size of allocated block, which makes easier management and improves effectiveness but may lead into fragmentation.

Methods of countermeasure against fragmentation:

- deallocation and merging,
- rellocation and compaction,
- paging mechanism.

[17] Memory Management - Bitmaps and Lists



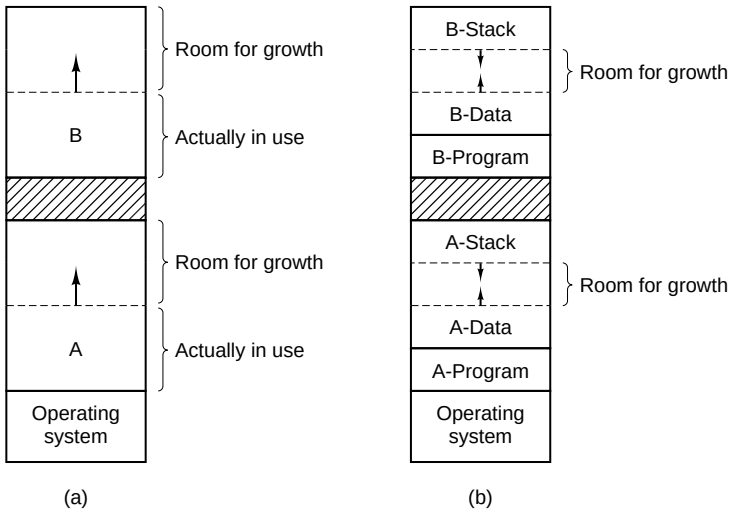
a. part of memory with 5 processes and 3 holes,

   – tick marks show allocation units,
   – shaded regions are free,
   – the smaller the allocation unit, the larger the bitmap.

b. corresponding bitmap,

c. same information as a list.

[18] **Dynamic Allocation Problem**



a. allocating space for growing data segment,

b. allocating space for growing stack and data segment.

[19] **Virtual Memory**

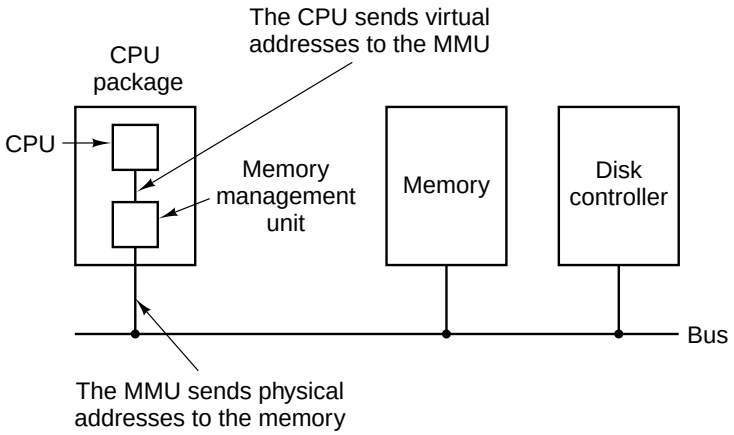
Virtual memory memory system consisting of at least two memory types: *small and fast* (e.g. **operating memory**) and *big but slow* (e.g. **disk memory**), and moreover of additional hardware and software which enable automatic moving of memory pieces between those memory types.

Virtual memory should be almost as fast as the faster from above-mentioned memories and almost as huge as the bigger from the above-mentioned memories.

Methods of virtual memory implementation:

- paging,
- segmentation,
- paging with segmentation.

[20] **Memory Management Unit Role**



MMU (ang. *Memory Management Unit*) may be integrated with processor (as it is common now), but may be independent (as it used to be earlier).

[21] **Paging**

Paging is based on fixed memory division. Units of division:

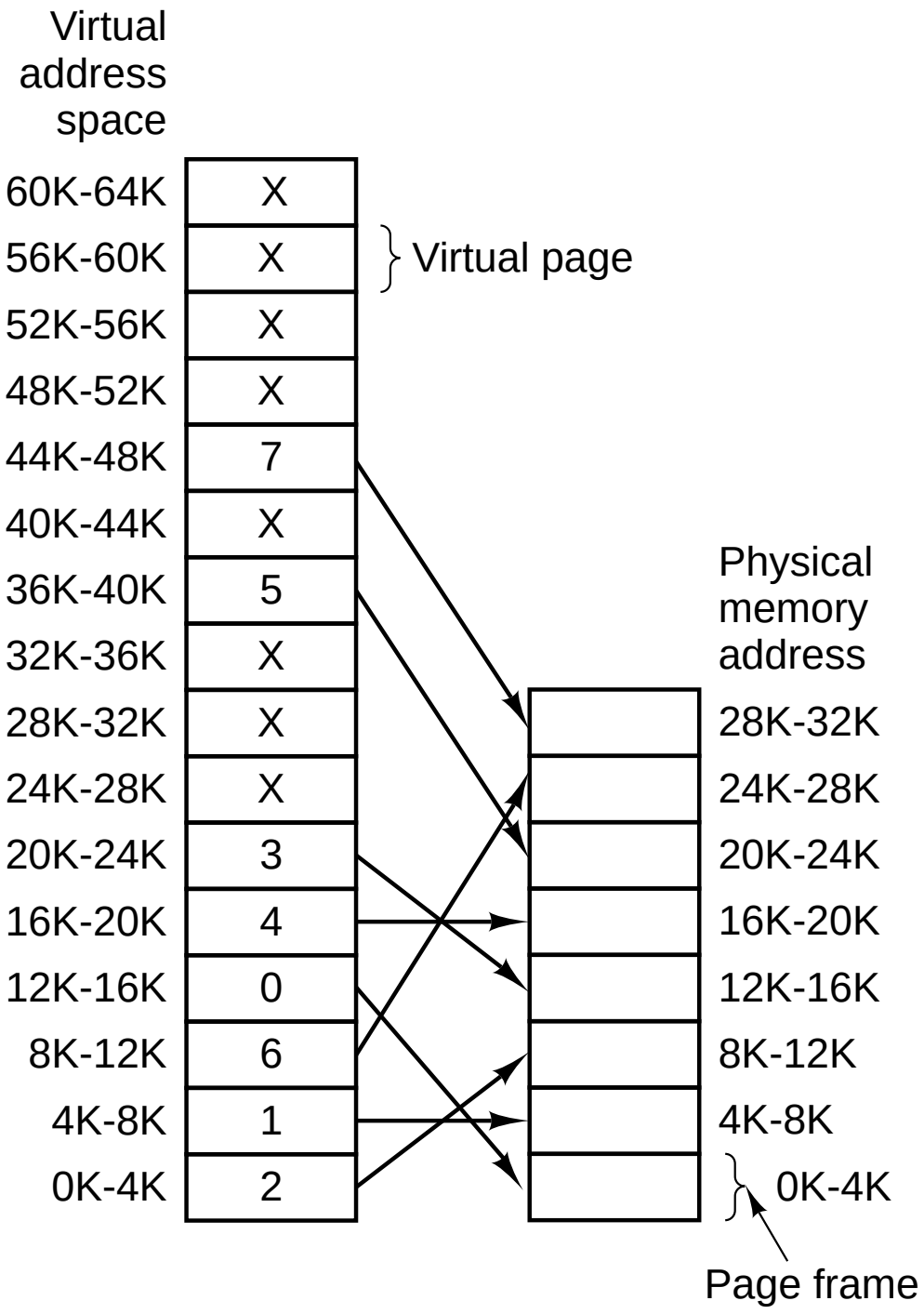
- **frames** for physical memory,

- **pages** for process virtual address space.

Paging mechanism is responsible for:

- mapping virtual address into physical address:
  1. locating the page referenced by the address in the program,
  2. finding the frame currently used by that page.
- sending - depending on requests - pages from external memory to operating memory and sending back those pages which are no longer required.

[22] **Translation Example**



[23] **Paging Implementation**

Usage of paging does not require anything specific from the user. Memory allocation is implemented by the system with usage of **page tables** and/ or **frame**

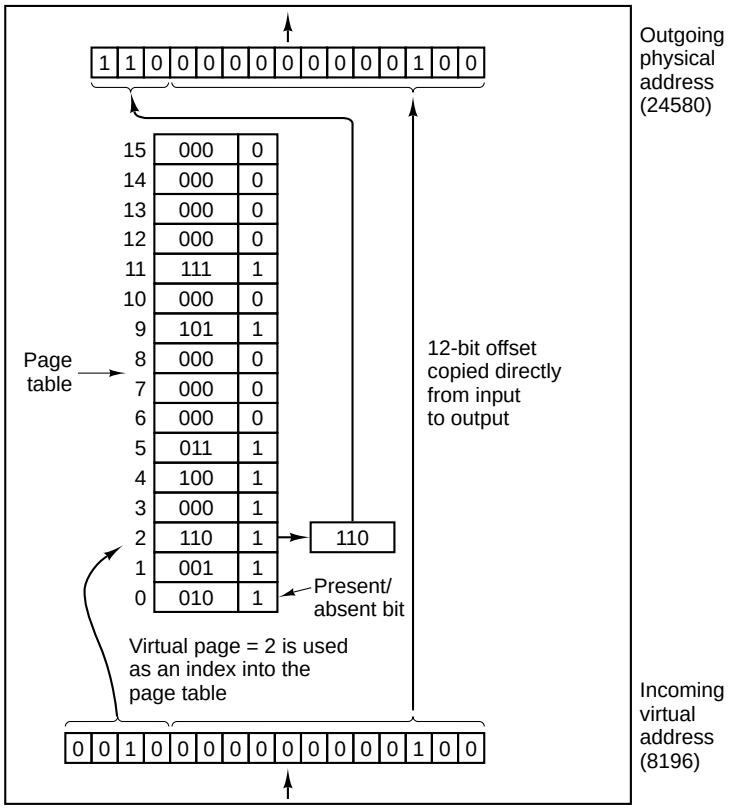
tables.

In order to enforce transparency of the solution, the following is required:

- handling system interrupt called **page fault**, which signals referencing to the page which is currently absent in the main memory,
- page fault handling has, due to some established page replacement algorithm, allocate a frame and download the required page from the external memory,
- in order to minimize the cost of multiple memory accesses associative memory is used for address transalation.

[24] **Exemplary Page Table**

Internal operation of MMU with 16 4 KB pages.

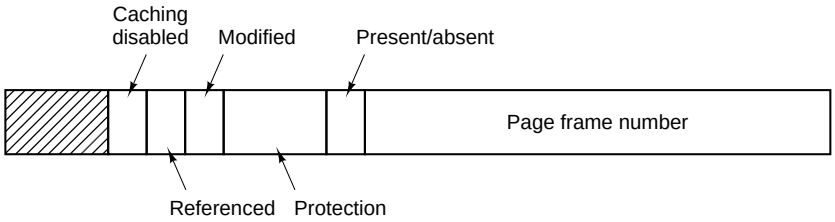


The following aspects should be considered:

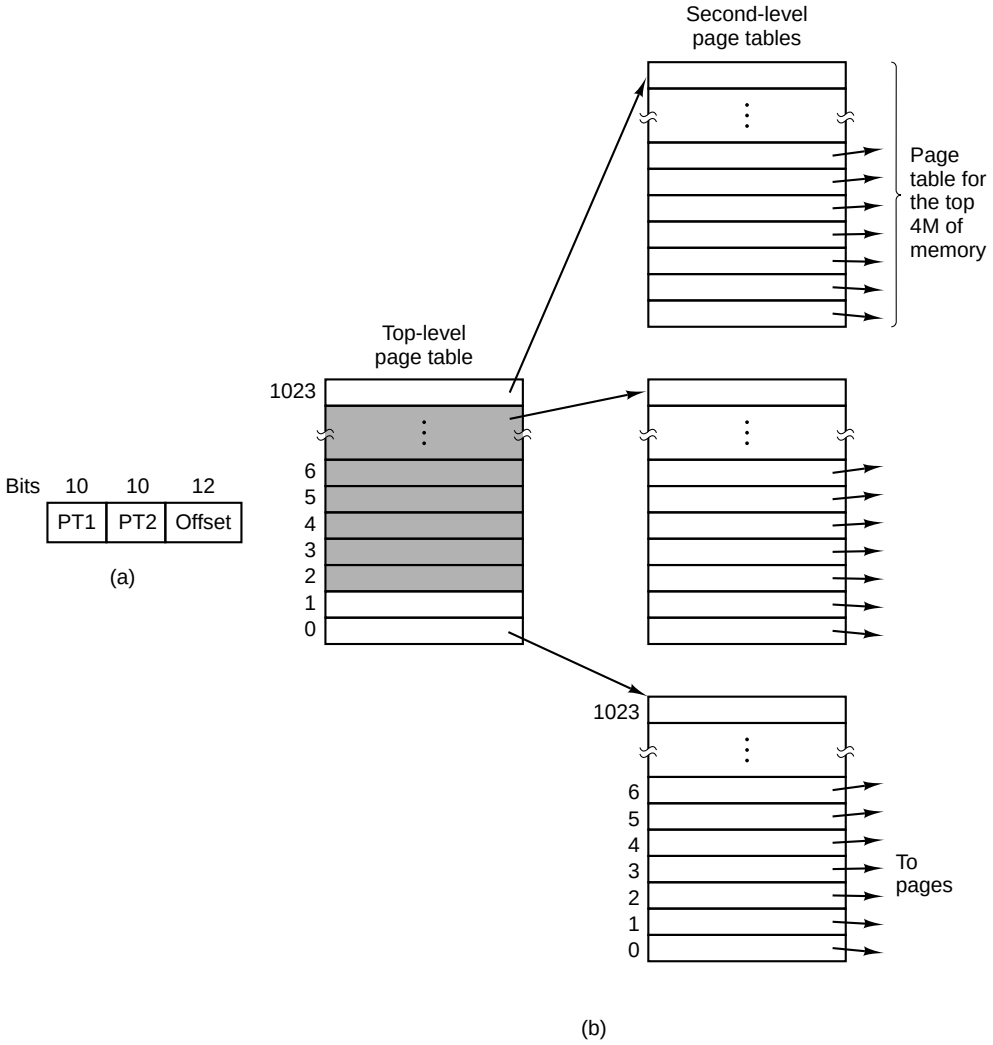
- page table may be extremely huge,

- translation process should be extremely fast.

[25] **Typical Page Table Entry**



[26] **Multilevel Page Tables**



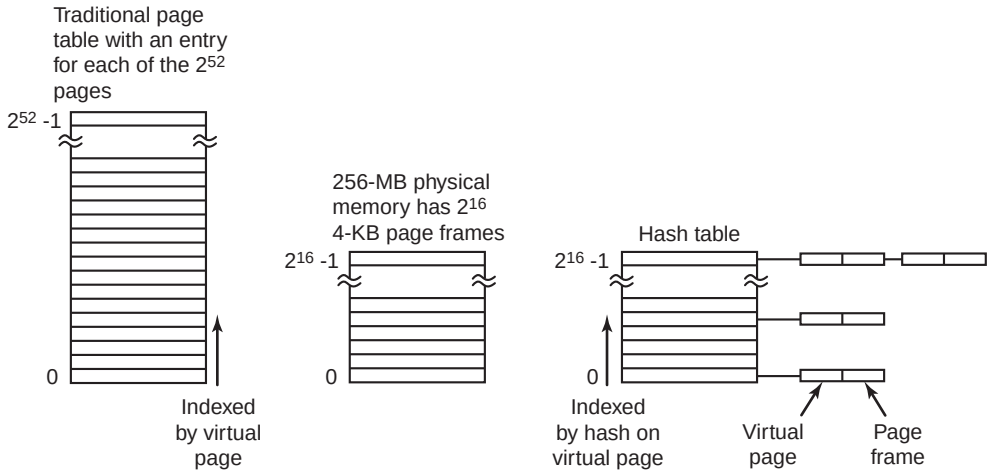
[27] **Translation Lookaside Buffer**

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

The main goal: to minimize the number of memory references required for successful mapping. Aspects related to TLB:

- cost of context switching,
- cost of multilevel page table,
- implemetation: in hardware/ in software.

[28] **Inverted Page Tables**



Comparison of a traditional page table with an inverted page table.

[29] **Optimal Page Table Size**

The size of a page influences internal fragmentation level and size of the page table.

Let:

s the average process size (in bytes),

p page size,

e size of one item in the page table.

then $overhead = s * e/p + p/2$

The first derivative with respect to p and equating to 0:

$-se/p^2 + 1/2 = 0$

Therefore, the optimal page size is equal to $p = \sqrt{2se}$

[30] **Page Fault Handling**

Page fault handling has the following sequence of events:

1. choosing the frame for required page,
2. freeing pointed frame,
3. the disk operation scheduling to bring the page in the frame,
4. page table updating.

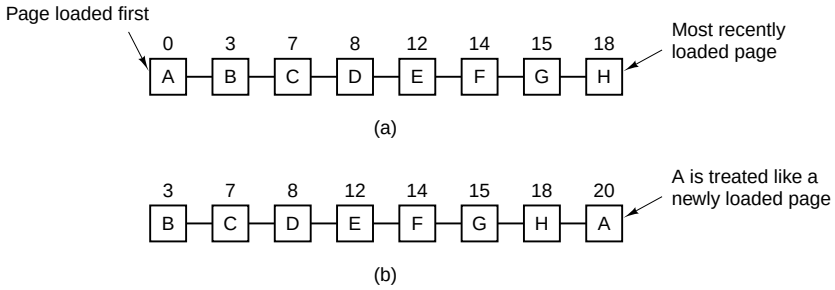
For choosing the frame with page to be evicted is responsible algorithm called page replacement algorithm. The optimal solution would be to evict the page which will be never again required or required in the most distant moment in the future.

[31] **Page Replacement Algorithms**

Some of page replacement algorithms:

- optimal solution,
- NRU algorithm (not recently used) (R and M bits),
- FIFO algorithm,
- the second chance algorithm,
- clock algorithm,
- LRU algorithm (least recently used).

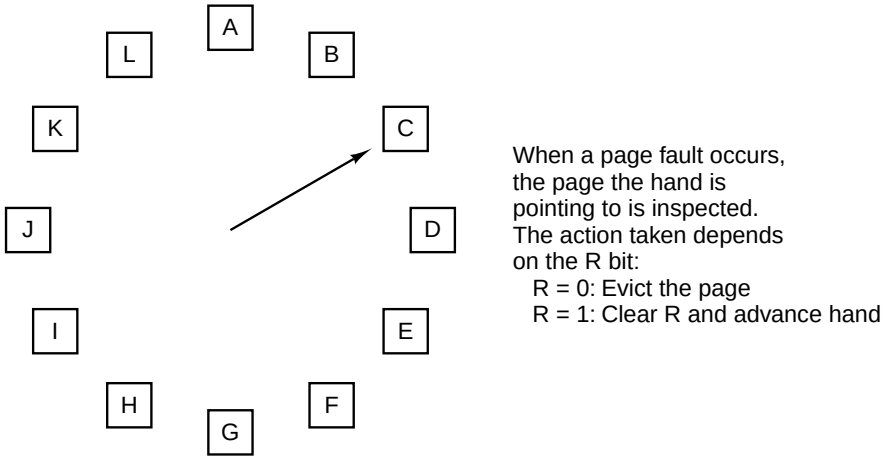
[32] **Second Chance Algorithm**



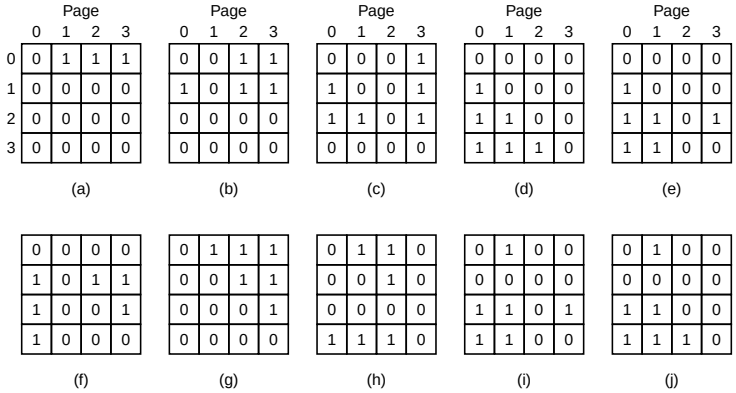
a. pages sorted in FIFO order,

b. new list, after page fault occurrence at time 20 and *A* had its *R* bit set.

[33] **Clock Page Replacement Algorithm**



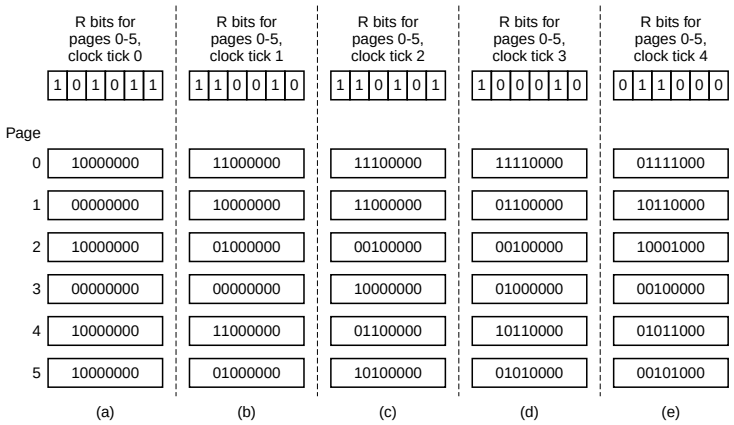
[34] **Hardware LRU Algorithm**



Pages were referenced in the following order:

0, 1, 2, 3, 2, 1, 0, 3, 2, 3.

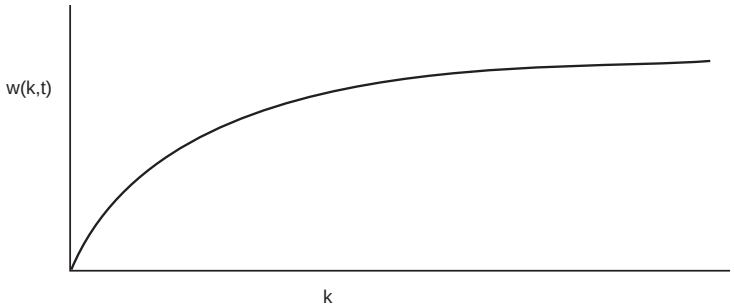
[35] **Software Simulation of LRU**



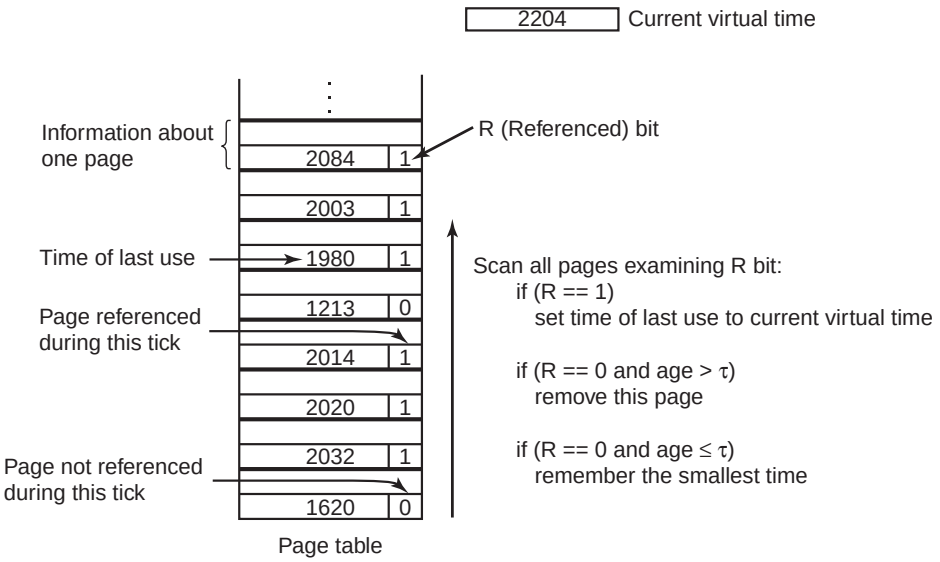
The aging algorithm simulates LRU in software. Six pages for five clock ticks is shown. The five clock ticks are represented by (a) to (e).

[36] **Working Set**

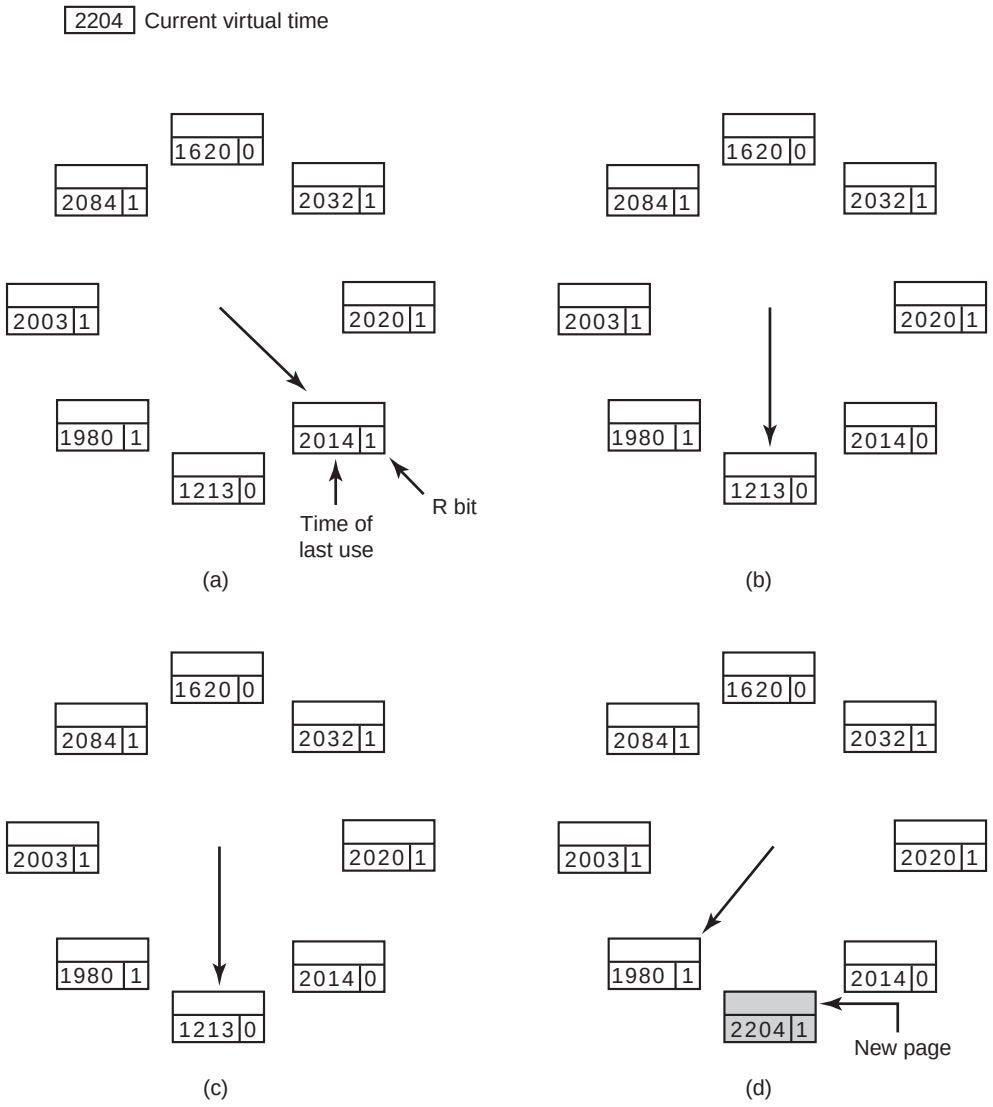
The **working set** is the set of pages used by the k most recent memory references. The function $w(k, t)$ is the size of the working set at time t.



[37] **Working Set Algorithm**



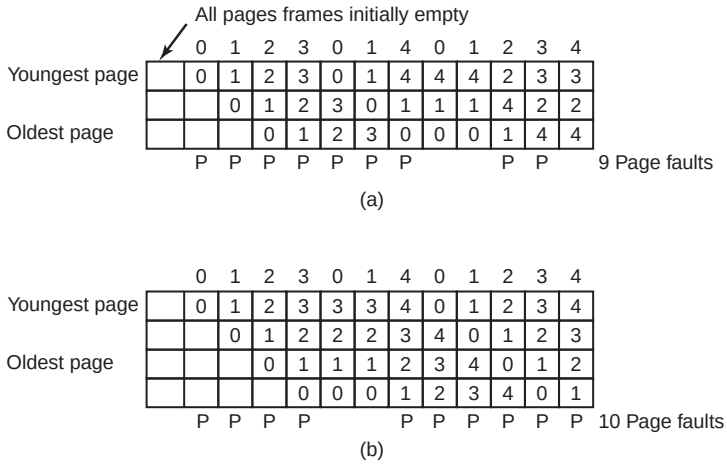
[38] **Clock Working Set Algorithm**



[39] **Page Replacement Algorithms Comparison**

Algorithm	**Comment**
Optimal	not implementable, but useful as a benchmark
NRU	very crude
FIFO	might throw out important pages
Second chance	big improvement over FIFO
Clock	realistic
LRU	excellent, but difficult to implement exactly
NFU	fairly crude approximation to LRU
Aging	efficient algorithm that approximates LRU well
Working set	somewhat expensive to implement
WSClock	good efficient algorithm

[40] **Belady's Anomaly**

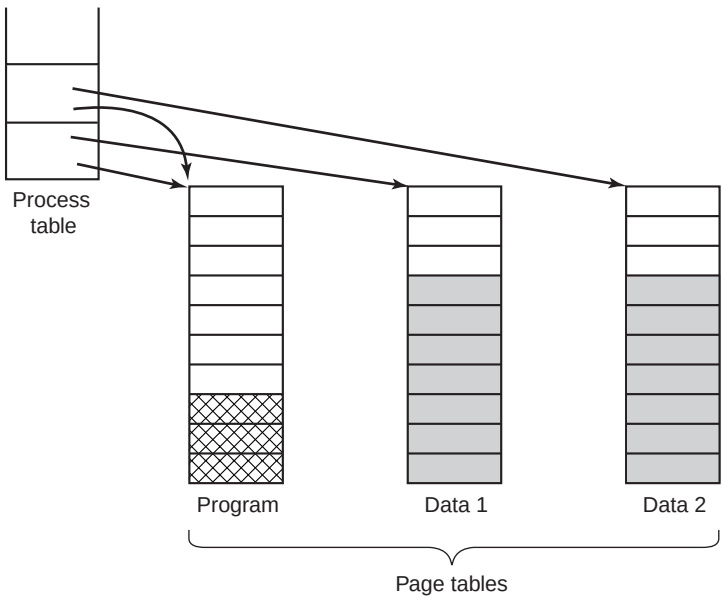


a. FIFO algorithm with three page frames,

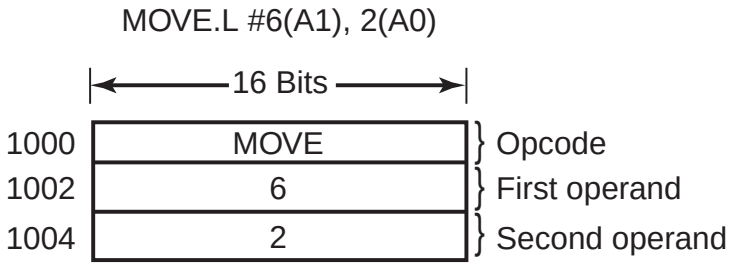
b. FIFO algorithm with four page frames,

- more page faults when we have more page frames. The P's show which page references cause page faults.

[41] **Shared Pages**

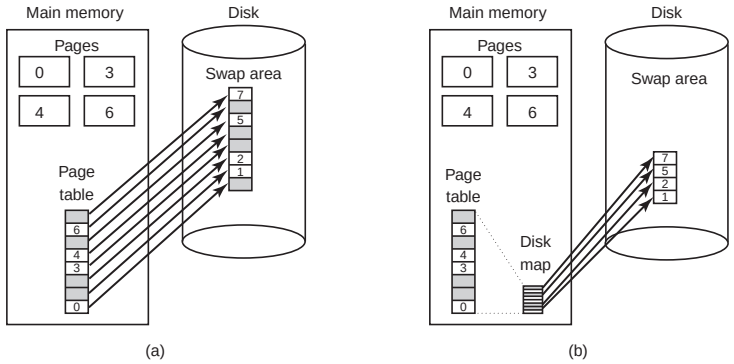


[42] **Page Fault Handling**



- an instruction causing a page fault,
- where to start the reexecution of the instruction?

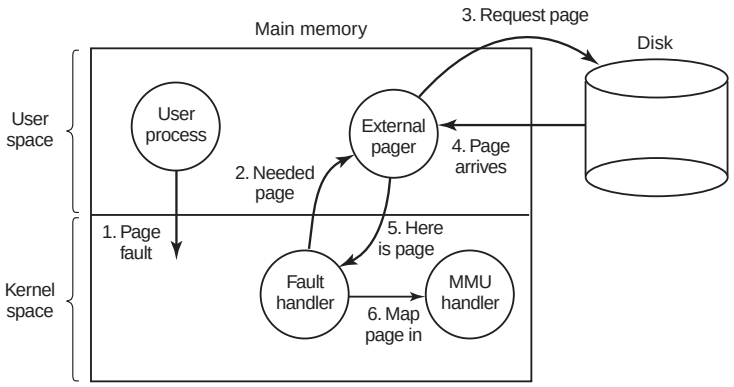
[43] **Swap Organization**



a. paging to a static swap area,

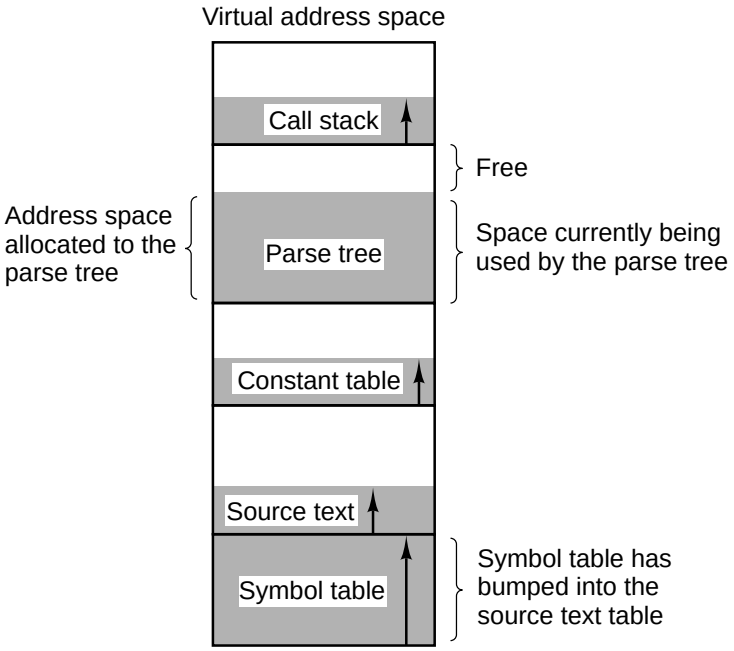
b. backing up pages dynamically.

[44] **Policy and Mechanism Separation**



- page fault handling with an external pager.

[45] **Single Address Space**



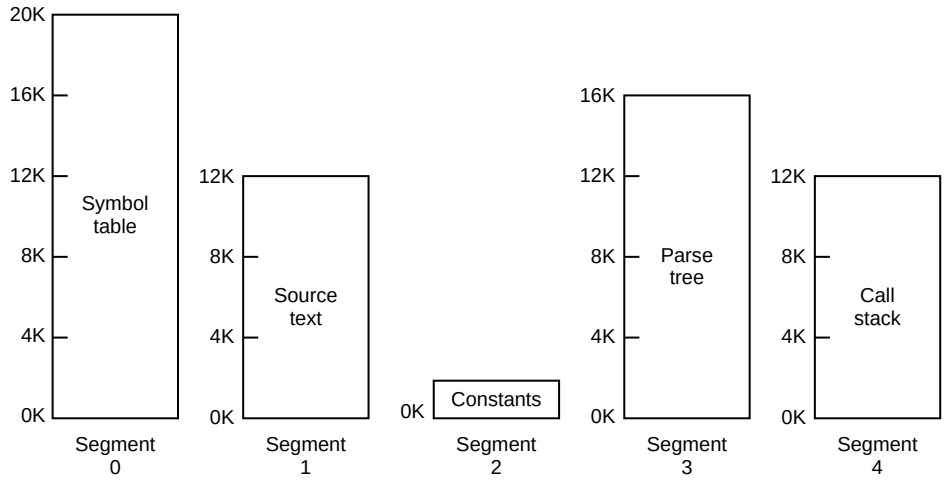
In a one-dimensional address space with growing tables, one table may bump into another - compiler process as an example.

[46] **Segmentation Features**

- the goal of segmentation is to organize the memory address space in a way corresponding to the logical information distribution,
- possibility of usage of many named (by the programmer) segments during the process of virtual address space organization,
- two-dimensional address space because of address identification by the pair: *segment name* + *offset*,
- address translation organized usually by separate for each process **segment table**,
- entries in the segment table called **segment descriptors**,

- each segment descriptor contains at least **base address** and **size** (limit) of the segment.

[47] **Segment Size**



A segmented memory allows each table to grow or shrink independently of the other tables.

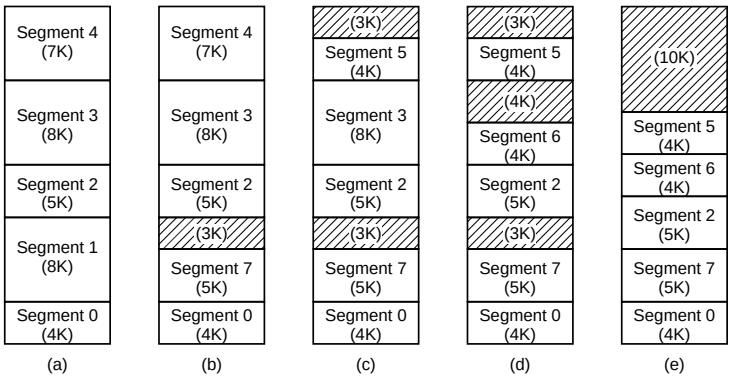
[48] **Paging and Segmentation (I)**

- the aim of segmentation: **logical** division of the operating memory, the aim of paging **physical** division of the memory,
- paging as a low-level mechanism, transparent to the programmer; segmentation as a high-level mechanism, visible to the programmer,
- size of a page fixed, derived from the system architecture; size of a segment not fixed, set by the programmer,
- both with paging and segmentation the total address space may be bigger than available physical memory,
- segmentation enables better protection by opportunity to distinguish different segments which logically group process' elements.

[49] **Paging and Segmentation (II)**

- segmentation enables better management of elements with non-constant size (like stack or heap),
- segmentation provides sharing of procedures among processes (shared segments, shared libraries),
- the reason of paging introduction: obtaining more address space without need to buy more physical memory,
- the reason of segmentation introduction: enabling division and distinction of programs and data into separate logically independent address spaces with support for sharing some elements and better protection,
- segmentation - external fragmentation possible,
- paging - internal segmentation possible.

[50] **Pure Segmentation Implementation**



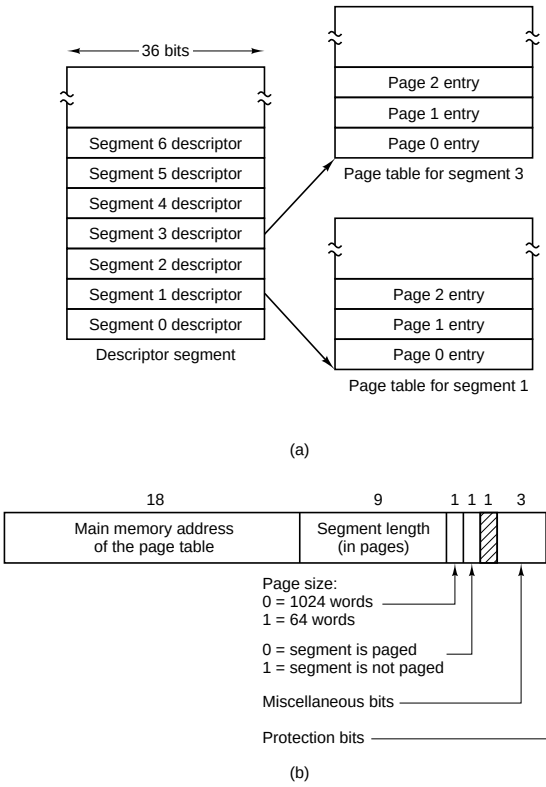
- phenomenon of **external fragmentation** (*checkerboarding*),
- (a)-(d) development of checkerboarding,
- (e) removal of the checkerboarding by compaction.
- usage of paging as a countermeasure against external fragmentation.

[51] **MULTICS: Segmentation and Paging**

- Honeywell 6000 and following models,

- 2^{18} segments possible, each segment might have up till 65536 36-bits words,
- paged segments, for each process segment table, the very segment table implemented as a paged segment as well,
- physical addresses 24-bits, pages aligned to 64 (2^6) bytes, thus 18 bits for page table address,
- eventual segment address in cache in a separate table used by the page fault handler.

[52] **MULTICS: Virtual Memory**

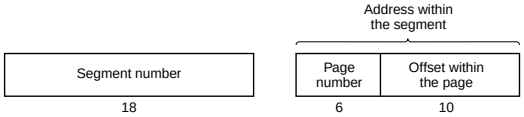


The MULTICS virtual memory.

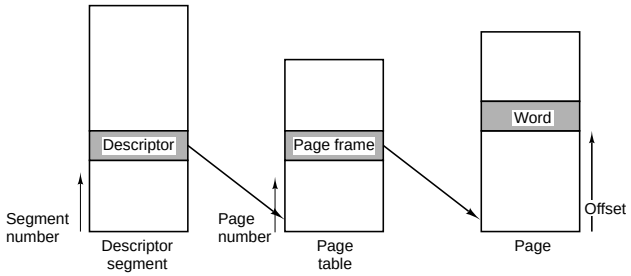
a. the descriptor segment points to the page tables,

b. a segment descriptor, the numbers are the field lengths.

[53] **MULTICS: Physical Address Evaluation**



A 32-bit MULTICS virtual address.



Conversion of a two-part MULTICS address into a main memory address. For simplicity it is omitted that segment of descriptors is itself paged.

[54] **MULTICS: TLB Buffers**

Comparison field					Is this entry used?
Segment number	Virtual page	Page frame	Protection	Age	
4	1	7	Read/write	13	1
6	0	2	Read only	10	1
12	3	1	Read/write	2	1
					0
2	1	0	Execute only	7	1
2	2	12	Execute only	9	1

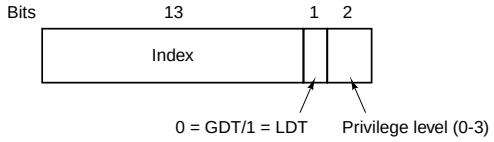
A simplified version of the MULTICS TLB. 16 entries for which segment number and frame number are compared in parallel. The existance of two page sizes makes the actual TLB more complicated.

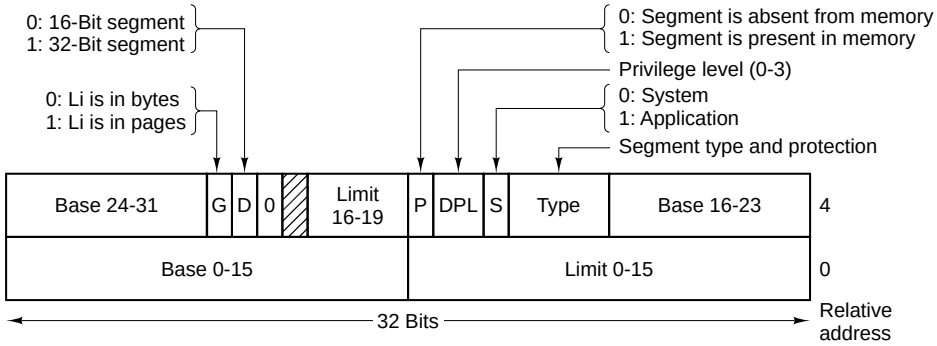
[55] **Intel Pentium: Segmentation with Paging**

- Multics: 256K independent segments, each up to 64K 36-bit words,
- Pentium: 16K independent segments, each up to 1 billion 32-bit words,
- single **Global Descriptor Table** in the system,
- **Local Descriptor Table** for each process.

[56] **Intel Pentium: Selector and Segment Descriptor**

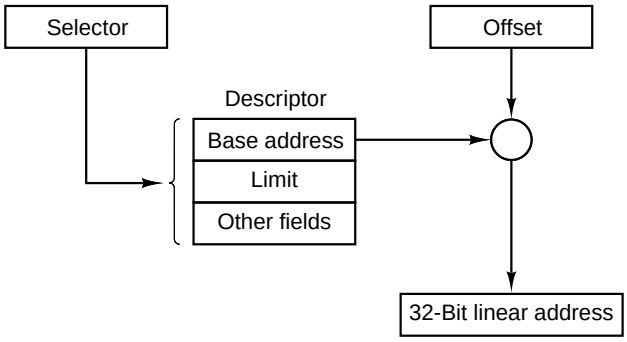
A Pentium selector.





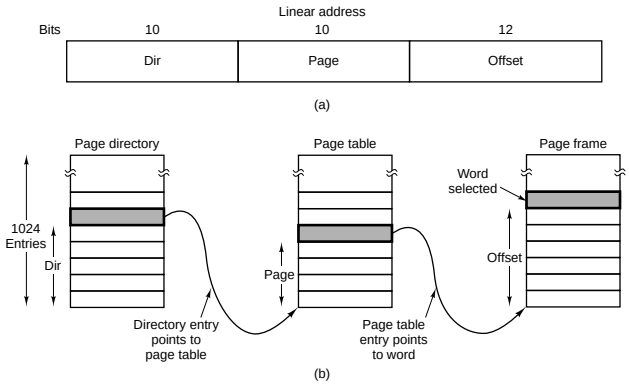
Pentium code segment descriptor. Data segments differ slightly.

[57] **Intel Pentium: Physical Address Evaluation (I)**



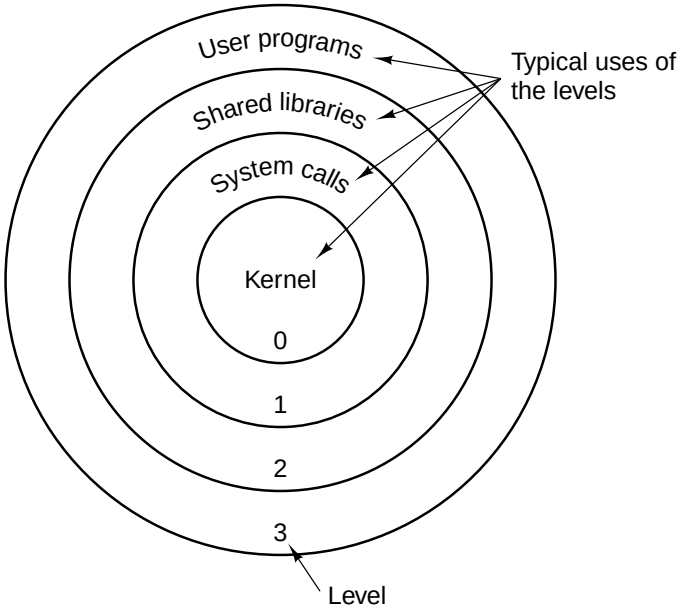
- conversion of a (selector, offset) pair to a linear address.
- if paging is turned off (pure segmentation), obtained address is a physical one,
- with paging on, obtained address is a logical one,
- usage of TLB buffers.

[58] **Intel Pentium: Physical Address Evaluation (II)**



- 32-bits logical address space, each page has a size 4kB,
- each process contains page directory with 2^{10} elements, each of them points to page table with 2^{10} elements as well.
- single paged 32-bits address space possible as well.

[59] **Intel Pentium: Protection**



- four levels of protection, segments from the same and higher levels accessible for reading/writing,
- to call a procedure from different level, CALL has as an argument not an address but selector of so called **call gate**.