5th Slide Set
Operating Systems

Prof. Dr. Christian Baun

Frankfurt University of Applied Sciences
(1971–2014: Fachhochschule Frankfurt am Main)
Faculty of Computer Science and Engineering
christianbaun@fb2.fra-uas.de

Learning Objectives of this Slide Set

- At the end of this slide set You know/understand...
  - fundamental concepts of **memory management**
    - **Static partitioning**
    - **Dynamic partitioning**
    - **Buddy memory allocation**
  - how operating systems **access the memory** (address it!)
    - **Real mode**
    - **Protected mode**
  - components and concepts to implement **virtual memory**
    - Memory Management Unit (**MMU**)
    - Paged memory management (**paging**)
    - ~~Segmented memory management (**segmentation**)~~
  - the possible results if memory is requested (**Hit** and **Miss**)
  - the functioning and characteristics of common **replacement strategies**

Exercise sheet 5 repeats the contents of this slide set which are relevant for these learning objectives

Memory Management

- An essential function of operating systems
- Required for allocating portions of memory to programs at their request
- Also frees memory portions, which are allocated to programs, when they are not needed any longer

Intellectual Game...

How would you implement a memory management ?!

- 3 concepts for memory management:
  1. **Static partitioning**
  2. **Dynamic partitioning**
  3. **Buddy memory allocation**

These concepts are already somewhat older...

Image source: unknown (perhaps IBM)

A good description of the memory management concepts provide...

- **Operating Systems – Internals and Design Principles**, *William Stallings*, 4th edition, Prentice Hall (2001), P.305-315
- **Moderne Betriebssysteme**, *Andrew S. Tanenbaum*, 3rd edition, Pearson (2009), P.232-240

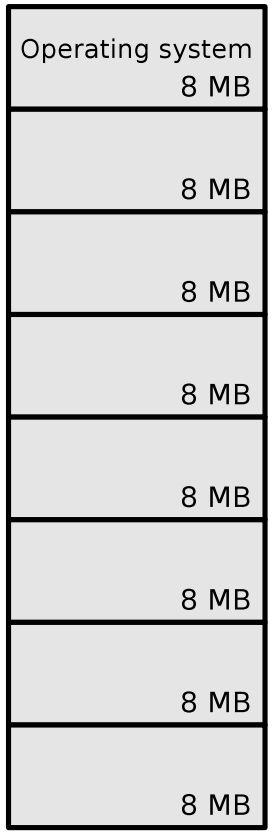
Concept 1: Static Partitioning

- The main memory is split into **partitions of equal size** or of **different sizes**
- Drawbacks:
  - **Internal fragmentation** occurs in any case $\Longrightarrow$ inefficient
    - The problem is moderated by partitions of different sizes, but not solved
  - The number of partitions limits the number of possible processes
- Challenge: A process requires more memory than a partition is of size
  - Then the process must be implemented in a way that only a part of its program code is stored inside the main memory
    - When program code (modules) are loaded into the main memory *Overlay* occurs
      $\Longrightarrow$ modules and data may become overwritten

IBM OS/360 MFT in the 1960s implemented static partitioning

Static Partitioning (1/2)

- If **partitions of equal size** are used, it does not matter which free partition is allocated to a process
  - If no partition is free, a process from main memory need to be replaced
    - The decision of which process will be replaced depends on the scheduling method ($\Longrightarrow$ slide set 8) used

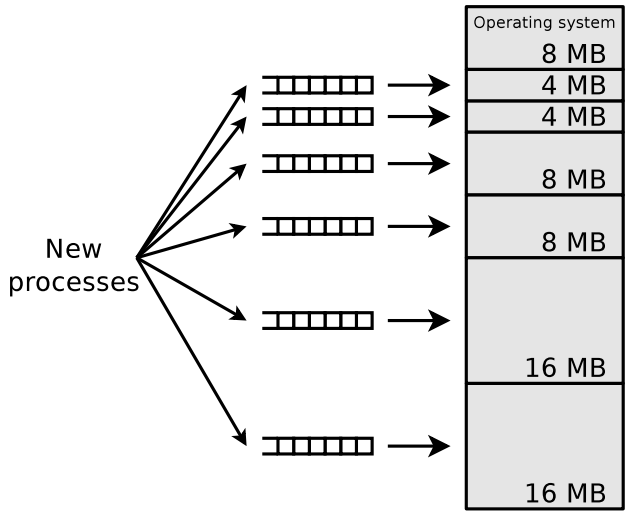


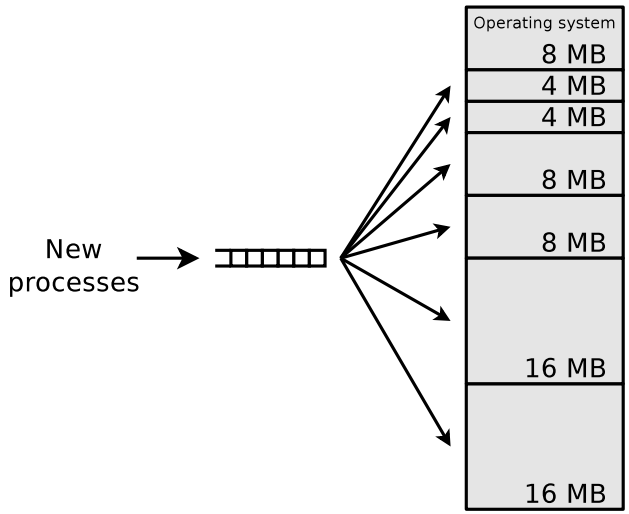
Partitions of equal size

Source:
Wiliam Stallings.
Operating Systems.
Prentice Hall. 2001

Static Partitioning (2/2)

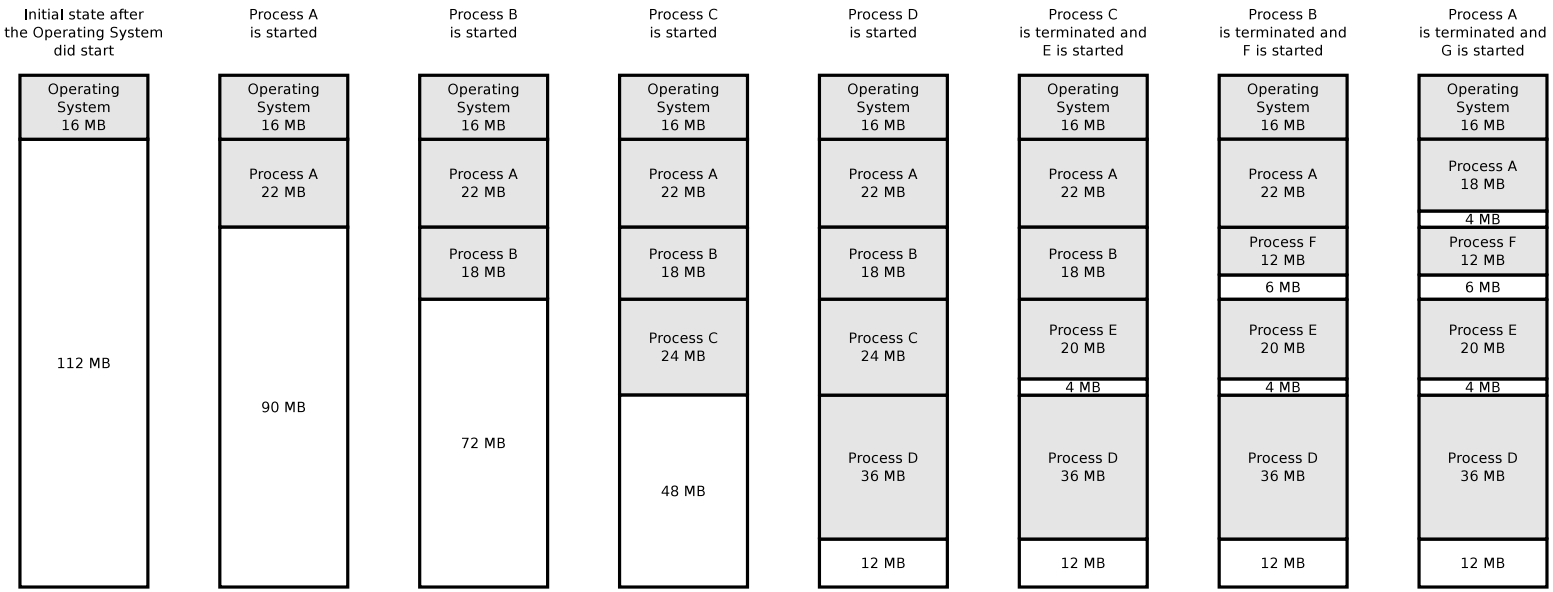
- Processes should get a partition allocated, which fits as precise as possible
  - Objective: Little internal fragmentation
- If **partitions of different sizes** are used, 2 possible ways exist to allocate partitions to processes:
  1. **A separate process queue for each partition**
     - Drawback: Some partitions may never used
  2. **A single queue for all partitions**
     - The allocation of partitions to processes can be carried out in a flexible way
     - To changed requirements of processes can be reacted quickly





Concept 2: Dynamic Partitioning

- Each process gets a gapless main memory partition with the exact required size allocated



- **External fragmentation** occurs in any case $\Longrightarrow$ inefficient
  - Possible solution: Defragmentation
    - Requirement: Relocation of memory blocks must be supported
    - References in processes must not become invalid by relocating partitions

IBM OS/360 MVT in the 1960s implemented dynamic partitioning

Implementation Concepts for Dynamic Partitioning

- **First Fit**
  - Searches for a free block, starting from the beginning of the address space
  - Quick method
- **Next Fit**
  - Searches for a free block, starting from the latest allocation
  - Fragments quickly the large area of free space at the end of the address space
- **Best Fit**
  - Searches for the free block, which fits best
  - Produces many mini-fragments and is slow

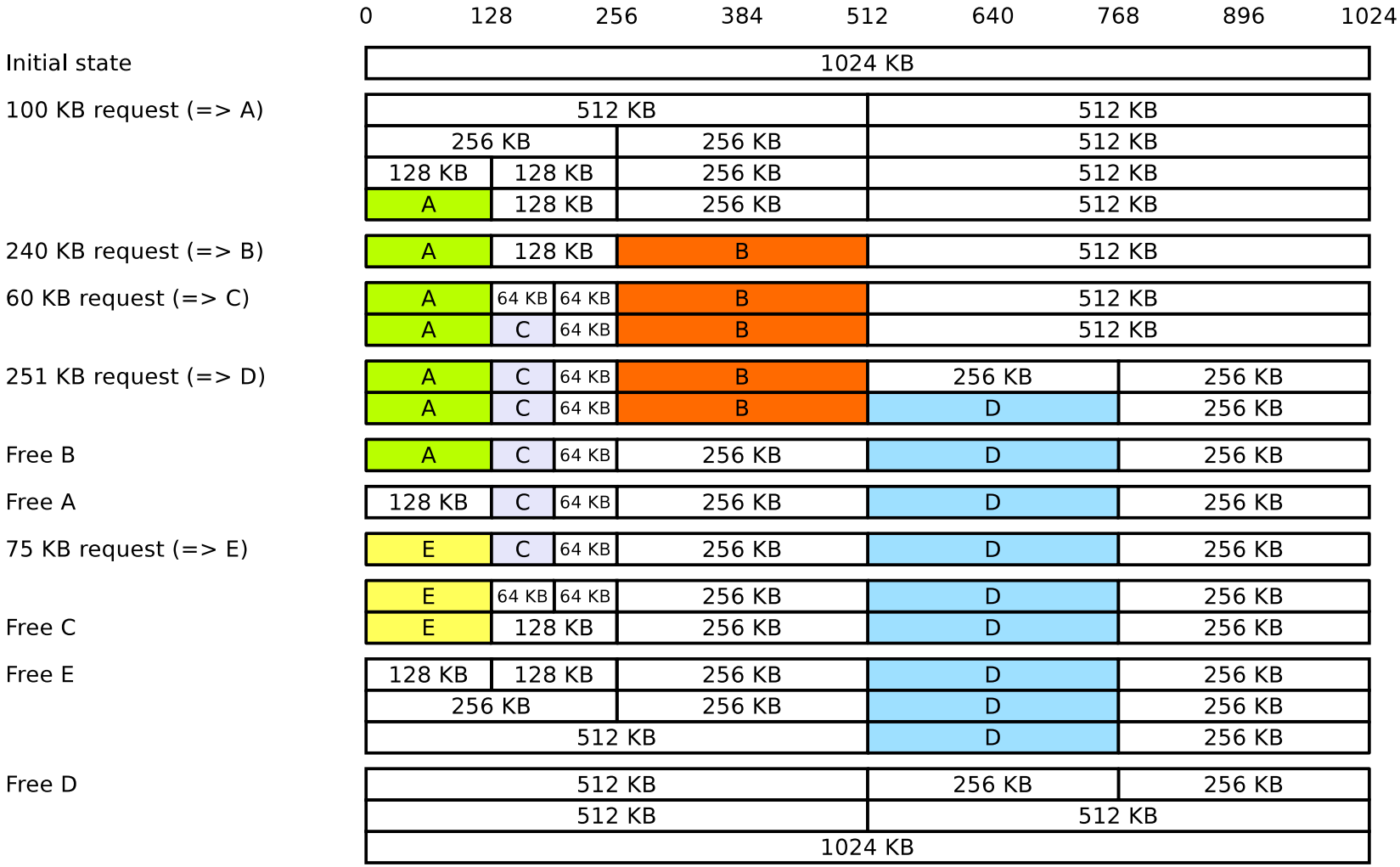
Concept 3: Buddy Memory Allocation of Donald Knuth

- Initially, a single block covers the entire memory
- If a process requires memory, the request is rounded up to the next higher power of two and a matching free block is searched
  - If no block of this size exists, a block of double size is searched and this block is split into 2 halves (so-called *buddies*)
    - The first block is then allocated to the process
  - If no block of double size exists, a block of four times size is searched, etc. . .
- If memory is freed, it is checked whether 2 halves of the same size can be recombined to a larger block
  - Only previously made subdivisions are reversed!

Buddy memory management in practice

- The Linux kernel implements a variant of the buddy memory management for the page allocation
- The operating system maintains for each possible block size a list of free blocks

Buddy Memory Allocation Example



- Drawback: Internal and external fragmentation

Information about the Memory Fragmentation

- The `DMA` row shows the first 16 MB of the system
  - The size of the address bus of the Intel 80286 is 2^{24} => 16 MB memory can be addressed maximum
- The `DMA32` row shows all memory $>$ 16 MB and $<$ 4 GB of the system
  - The address bus size of the Intel 80386, 80486, Pentium I/II/III/IV, ... is 2^{32} => 4 GB memory can be addressed
- The `Normal` row shows all memory $>$ 4 GB of the system
  - The size of the address bus of modern computer systems is usually 36, 44 or 48 bits

Further information about the rows: `https://utcc.utoronto.ca/~cks/space/blog/linux/KernelMemoryZones`

```
$ cat /proc/buddyinfo
Node 0, zone      DMA      1      1      1      0      2      1      1      0      1      1      3
Node 0, zone    DMA32    208    124   1646    566    347    116    139    115     17      4    212
Node 0, zone   Normal     43     62    747    433    273    300    254    190     20      8    287
```

- column 1 $\Longrightarrow$ number of free memory chunks („*buddies*“) of size $2^0 * PAGESIZE \Longrightarrow 4\,\text{kB}$
- column 2 $\Longrightarrow$ number of free memory chunks („*buddies*“) of size $2^1 * PAGESIZE \Longrightarrow 8\,\text{kB}$
- column 3 $\Longrightarrow$ number of free memory chunks („*buddies*“) of size $2^2 * PAGESIZE \Longrightarrow 16\,\text{kB}$
- ...
- column 11 $\Longrightarrow$ number of free memory chunks („*buddies*“) of size $2^{10} * PAGESIZE \Longrightarrow 4096\,\text{kB} = 4\,\text{MB}$

PAGESIZE = 4096 Bytes = 4 kB
The pagesize of a Linux system can be checked via the command: `$ getconf PAGESIZE`

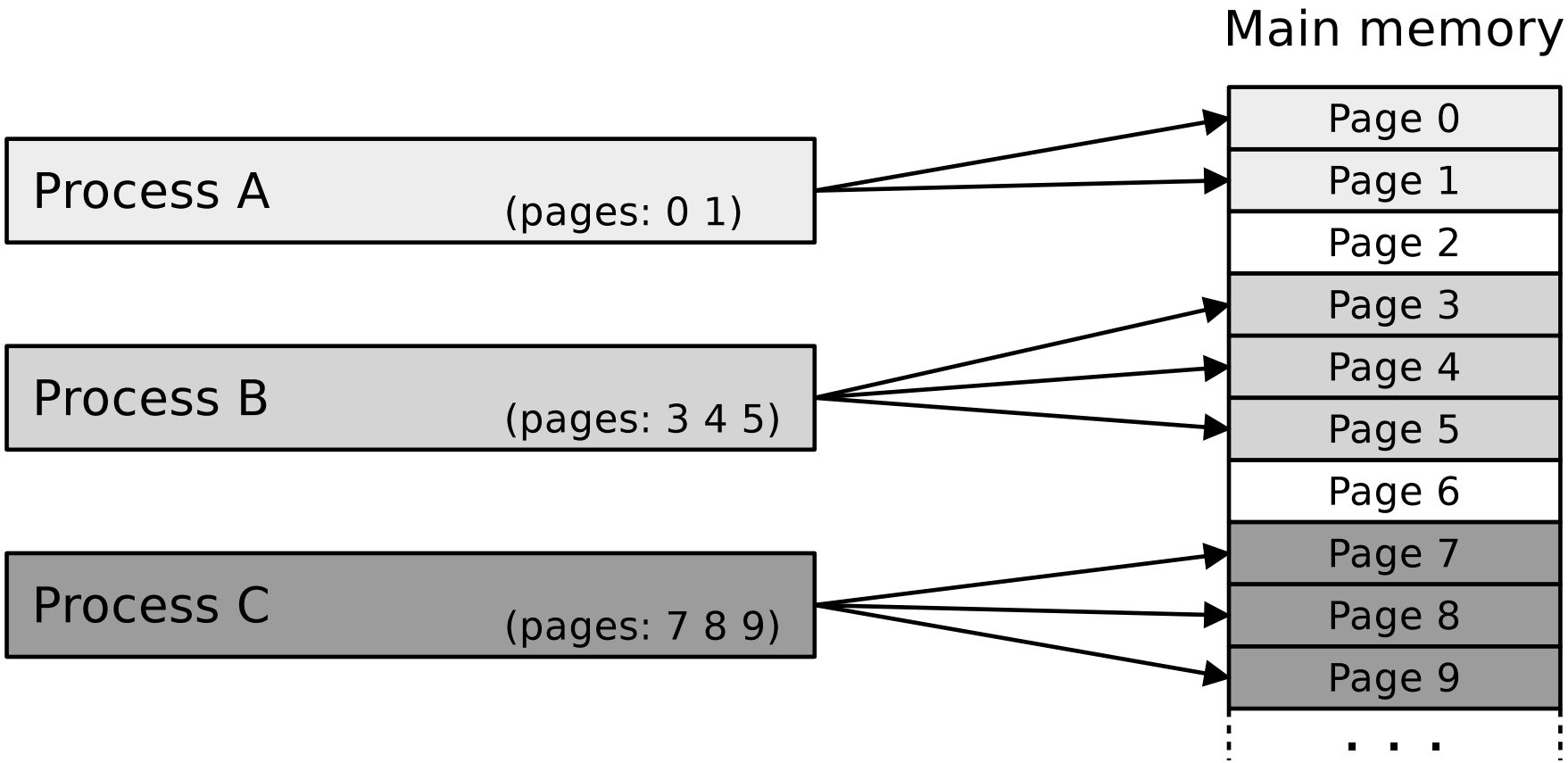
Accessing Memory

! ! ! Question ! ! !

How do processes access (allocate) memory?

- With 16-bit architectures, 2^{16} memory addresses and therefore up to $65,536$ Bytes can be addressed
- With 32-bit architectures, 2^{32} memory addresses and therefore up to $4,294,967,296$ Bytes $=$ **4 GB** can be addressed
- With 64-bit architectures, 2^{64} memory addresses and therefore up to $18,446,744,073,709,551,616$ Bytes $=$ **16 Exabyte** can be addressed

Idea: Direct Memory Access



- Most obvious idea: Direct memory access by the processes
  $\Longrightarrow$ **Real Mode**

Real Mode (Real Address Mode)

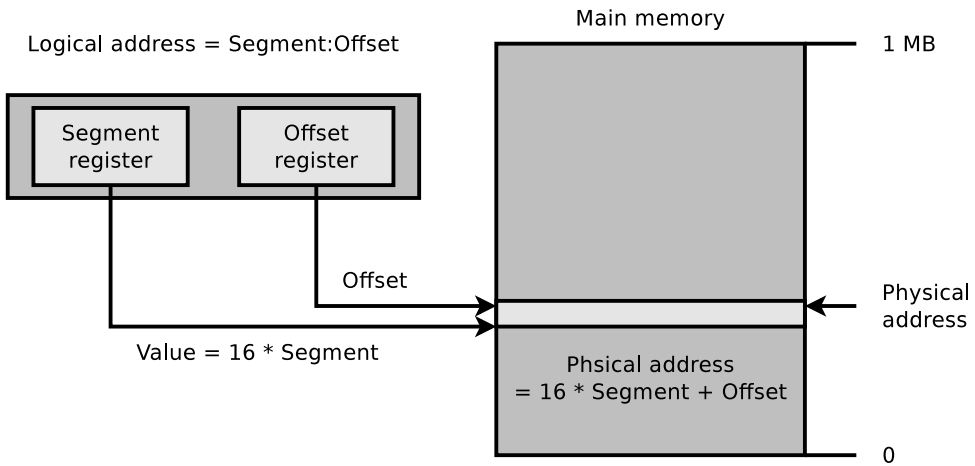
- Operating mode of x86-compatible CPUs
- **No memory protection**
  - Each process can access the entire memory, which can be addressed
    - Unacceptable for multitasking operating systems
- **A maximum of 1 MB main memory can be addressed**
  - Maximum main memory of an Intel 8086
  - Reason: The address bus of the 8088 contains only 20 lines
    - 20 lines $\Longrightarrow$ 20 Bits long memory addresses $\Longrightarrow$ 2^{20} = approx. 1 MB memory can be addressed by the CPU
  - Only the first 640 kB (*lower memory*) can be used by the operating system (MS-DOS) and the applications
    - The remaining 384 kB (*upper memory*) contains the BIOS of the graphics card, the memory window to the graphics card memory and the BIOS ROM of the motherboard
- The term „real mode“ was introduced with the Intel 80286
  - In real mode, a CPU accesses the main memory equal to a 8086
  - Each x86-compatible CPU starts in real mode

Helpful source: `http://www.internals.com/articles/protmode/realmode.htm`

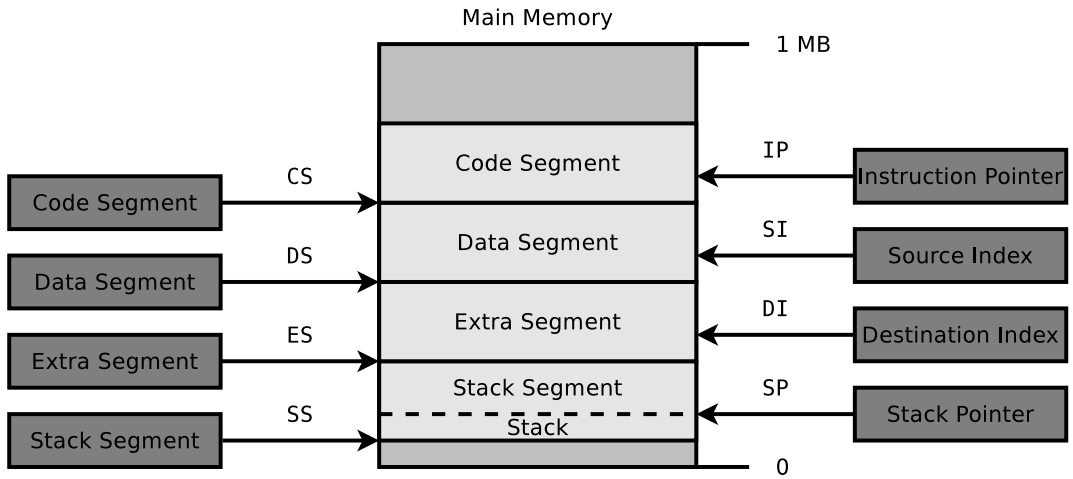
Real Mode – Addressing

- The main memory is split into 65,536 segments
  - The memory adress length is 16 Bits
  - The size of each segment is 64 Bytes ($= 2^{16} = 65,536$ bits)
- Main memory addressing is implemented via segment and offset
  - Two 16 bits long values, which are separated by a colon `Segment:Offset`
  - Segment and offset are stored in the two 16-bit large registers **segment register** (= **base address register**) and **offset register** (= **index register**)

- The **segment register** stores the segments number
- The **offset register** points to an address between 0 and 2^{16} (=65,536), relative to the address in the segment register



Real Mode – Segment Registers since the 8086

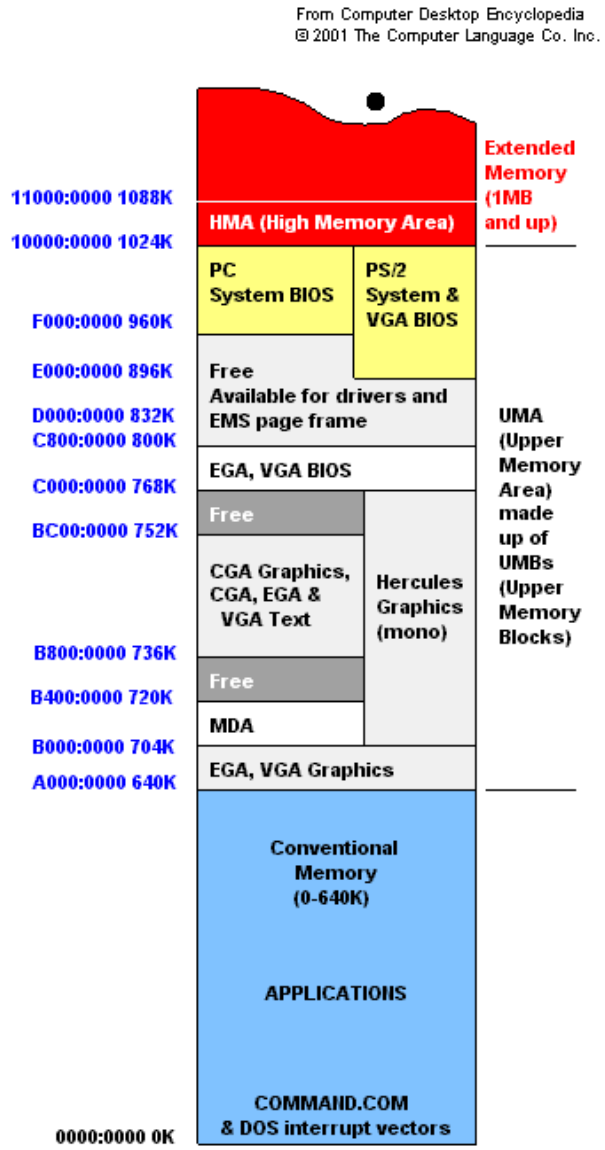


- The 8086 has 4 **segment registers**
- `CS` (Code Segment)
  - Contains the source code of the program
- `DS` (Data Segment)
  - Contains the global data of the current program
- `SS` (Stack Segment)
  - Contains the stack for the local data of the program
- `ES` (Extra Segment)
  - Segment for further data
- Since the Intel 80386, 2 addition segment registers (`FS`, and `GS`) for additional extra segments exist
- The segments implement a simple form of **memory protection**

Real Mode in MS-DOS

Image Source: Google Image Search

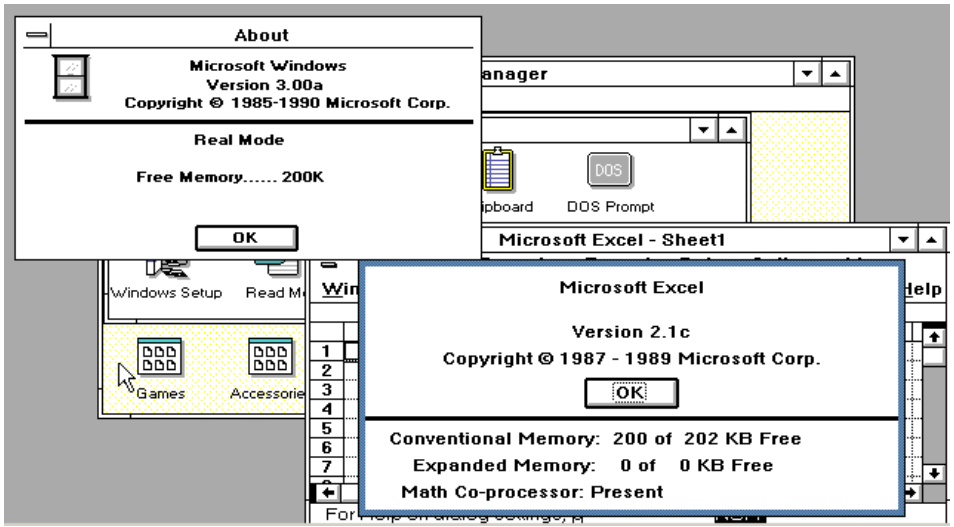
```
A:\>mem

Memory Type        Total       Used       Free
---------------  --------   --------   --------
Conventional         640K        92K       548K
Upper                  0K         0K         0K
Reserved             384K       384K         0K
Extended (XMS)   742,400K        64K   742,336K
---------------  --------   --------   --------
Total memory     743,424K       540K   742,884K

Total under 1 MB     640K        92K       548K

Largest executable program size       548K (561,552 bytes)
Largest free upper memory block         0K       (0 bytes)
MS-DOS is resident in the high memory area.
```

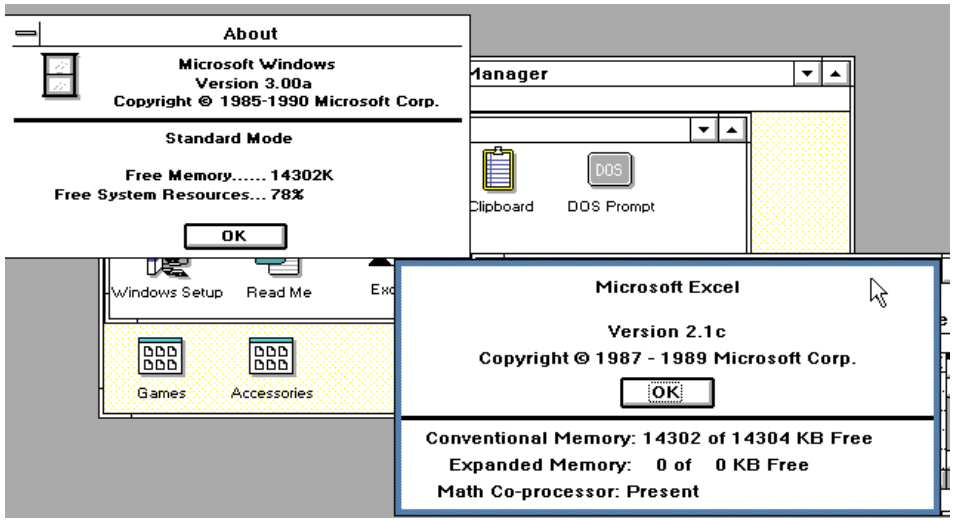


- Real mode is the default mode of MS-DOS and compatible operating systems (e.g. PC-DOS, DR-DOS and FreeDOS)

Real Mode in Microsoft Windows

- Newer operating systems only use it during the start phase and then switch to the **protected mode**





- Windows 2.0 runs only in real mode
- Windows 2.1 and 3.0 can run either in real mode or protected mode
- Windows 3.1 and later revisions run only in protected mode

Image Source: `http://virtuallyfun.superglobalmegacorp.com`

Memory Management Demands

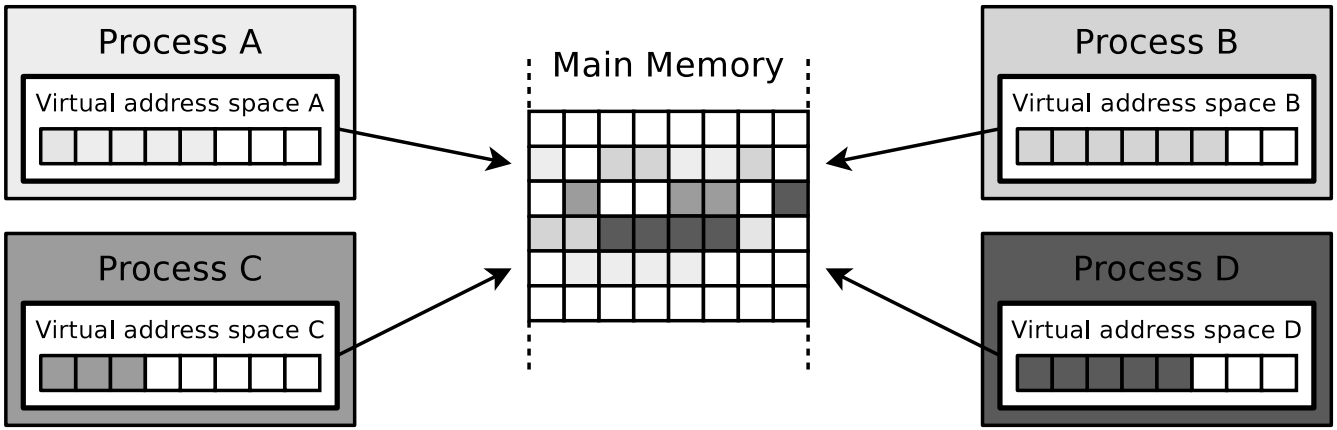
- **Relocation**
  - If processes are replaced from the main memory, it is unknown at which address they will be inserted later into the main memory again
  - Finding: Processes must not refer to physical memory addresses
- **Protection**
  - Memory areas must be protected against accidental or unauthorized access by other processes
  - Finding: Access attempts must be verified (by the CPU)
- **Shared use**
  - Despite memory protection, it must be possible for processes to collaborate via shared memory $\Longrightarrow$ slide set 10
- **Increased capacity**
  - 1 MB is not enough
  - It should be possible to use more memory as physically exists
  - Finding: If the main memory is full, parts of the data can be swapped
- Solution: **Protected mode** and **virtual memory**

Protected Mode

- Operating mode of x86-compatible CPUs
  - Introduced with the Intel 80286
- Increases the amount of memory, which can be addressed
  - 16-bit protected mode at 80286 $\Longrightarrow$ 16 MB main memory
  - 32-bit protected mode at 80386 $\Longrightarrow$ 4 GB main memory
  - For later processors, the amount of addressable memory depends on the number of bus lines in the address bus
- Implements the **virtual memory** concept
  - Processes do not use physical memory addresses
    - This would cause issues in multitasking systems
  - Instead, each process has a separate **address space**
    - It implements **virtual memory**
    - It is independent from the storage technology used and the given expansion capabilities
    - It consists of logical memory addresses, which are numbered from address 0 upwards

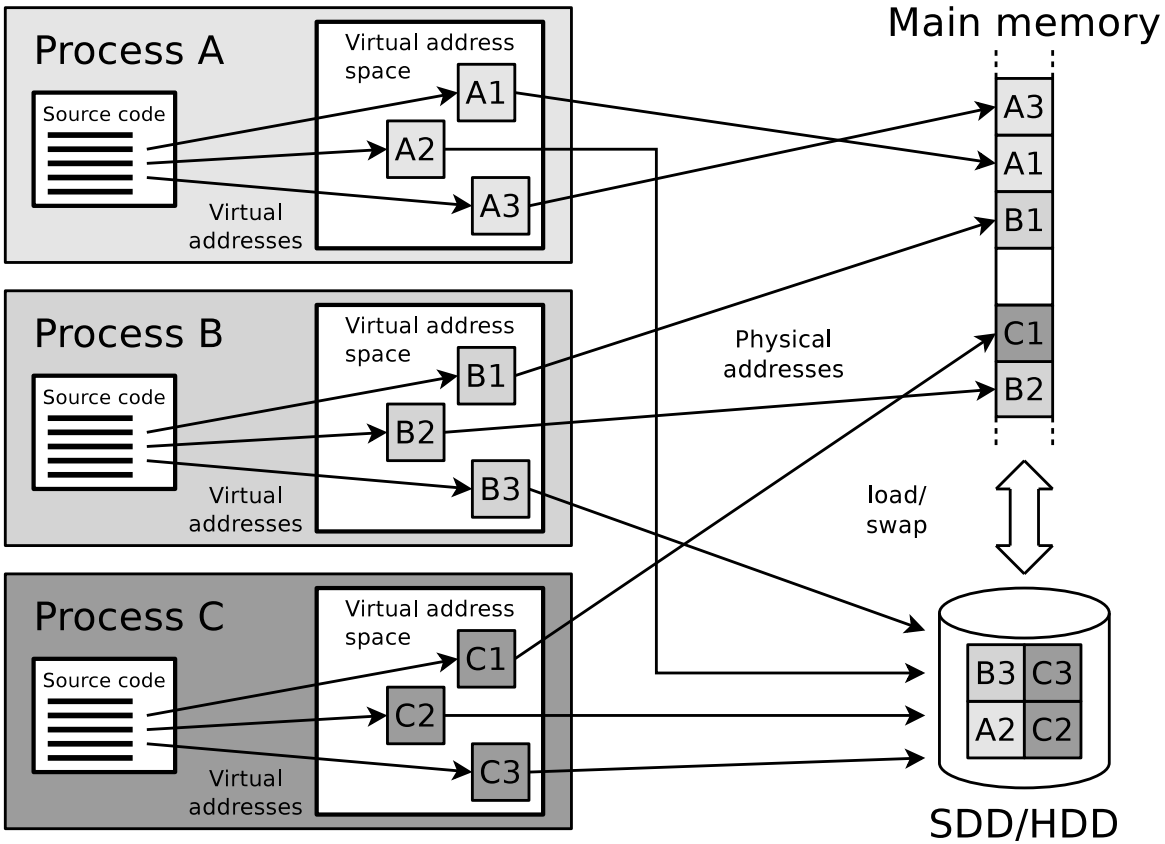
Virtual Memory (1/2)

- Address spaces can be created or erased as necessary and they are protected
  - No process can access the address space of another process without prior agreement
- The virtual memory is **mapped** to the physical memory



- With virtual memory, the main memory is utilized better
  - Processes do not need to be located in one piece inside the main memory
  - Therefore, the fragmentation of the main memory is not a problem

Virtual Memory (2/2)



- Thanks to virtual memory, more memory can be addressed and used, as is physically present in the system
- **Swapping** is performed transparently for users and processes

The topic Virtual Memory is clearly explained by...

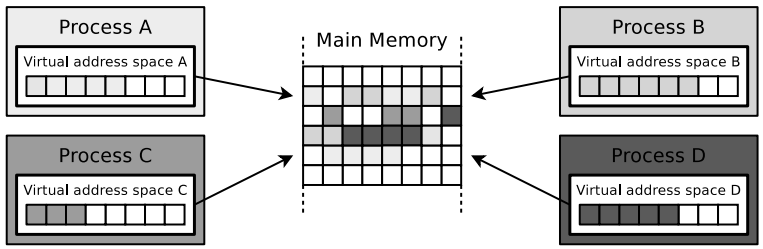
Betriebssysteme, *Carsten Vogt*, 1st edition, Spektrum Akademischer Verlag (2001), P. 152

In protected mode, the CPU supports 2 memory management methods

- ~~**Segmentation**~~ exists since the 80286
- **Paging** (siehe Folie 23) exists since the 80386
- Both methods are implementation variants of the **virtual memory** concept

Paging: Paged Memory Management

- **Virtual pages** of the processes are mapped to **physical pages** in the main memory
  - All pages have the same length
    - The page size is usually 4 kb (at the Alpha architecture: 8 kB)
- Benefits:
  - External fragmentation is irrelevant
  - Internal fragmentation can only occur in the last page of each process



- The operating system maintains **for each process a page table**
  - It stores the locations of the individual pages of the process
- Processes only work with **virtual memory addresses**
  - Virtual memory addresses consist of 2 parts
    - The more significant part is the page number
    - The lower significant part is the offset (address inside a page)
  - The length of the virtual addresses is architecture dependent (depends on the number of bus lines in the address bus), and is 16, 32, or 64 bits

Allocation of Process Pages to free Physical Pages

- Processes do not need to be located in a row inside the main memory
  $\Longrightarrow$ No external fragmentation

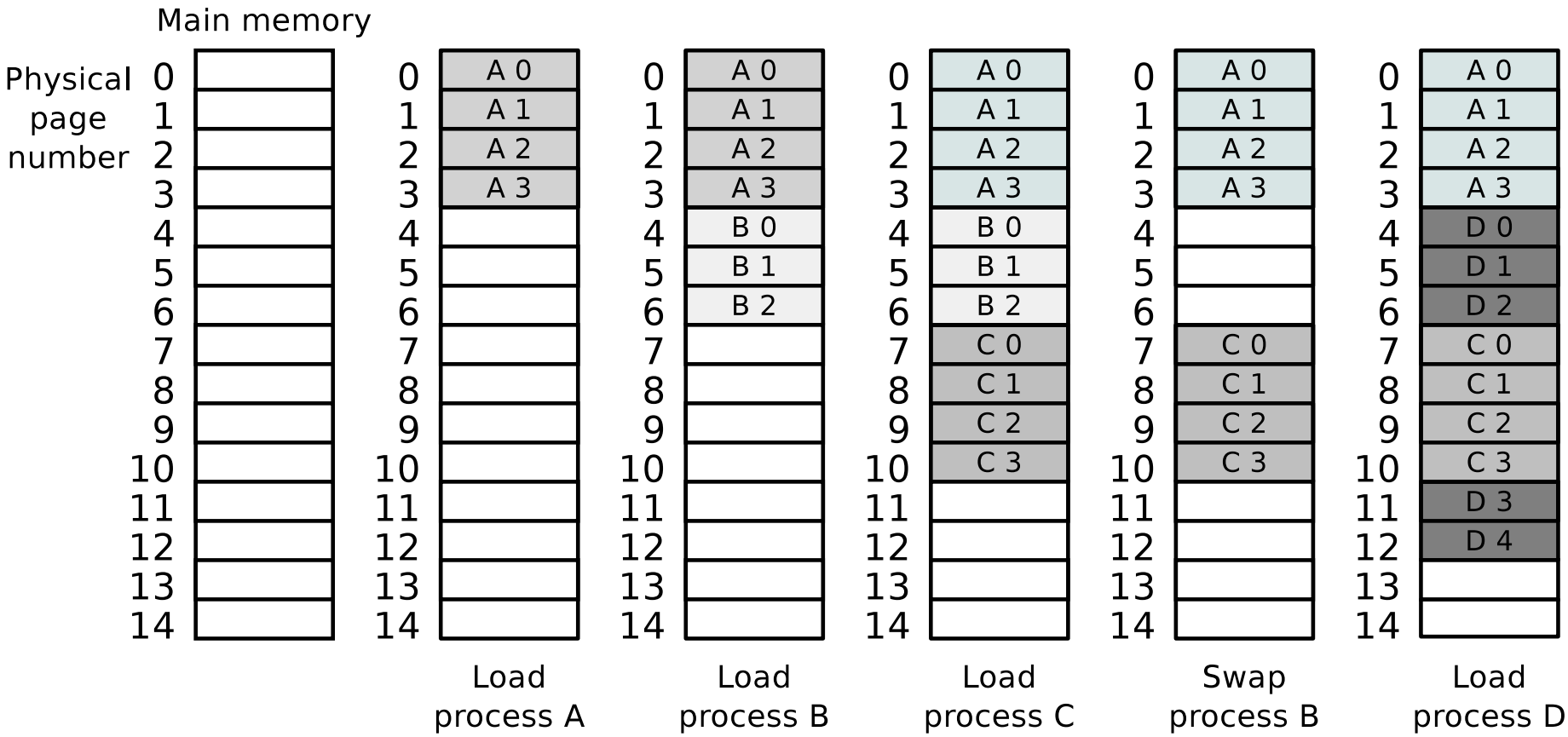
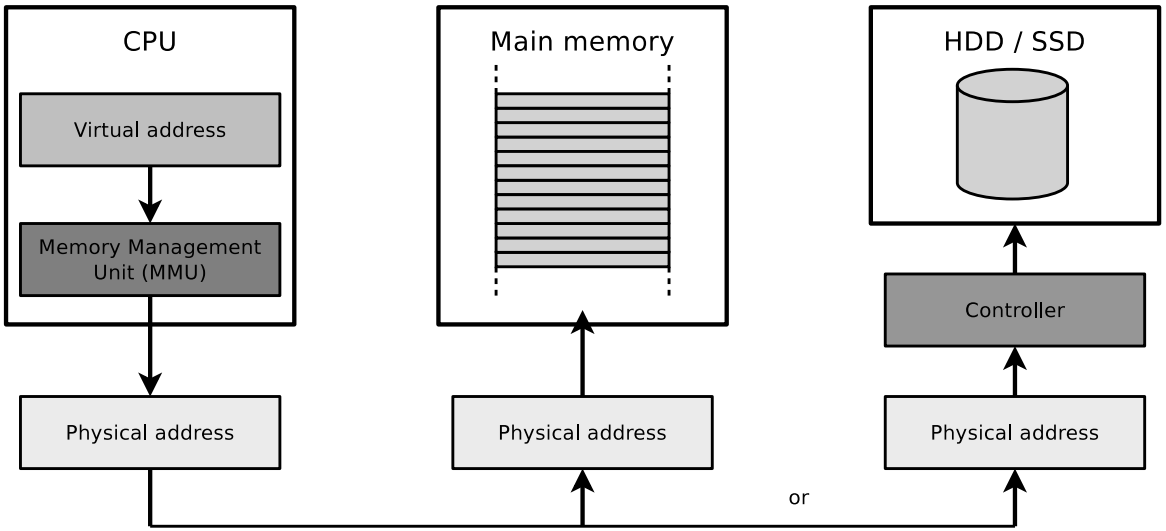


Image source: **Operating Systems**, *William Stallings*, 4th edition, Prentice Hall (2001)

Address Translation by the Memory Management Unit

- Virtual memory addresses translates the CPU with the **MMU** and the page table into physical addresses
  - The operating system determines whether the physical address belongs to the main memory or to a SSD/HDD



- If the desired data is located on the SSD/HDD, the operating system must copy the data into the main memory
- If the main memory has no more free capacity, the operating system must relocate (*swap*) data from the main memory to the SDD/HDD

The topic MMU is clearly explained by...

- **Betriebssysteme**, *Carsten Vogt*, 1st edition, Spektrum Akademischer Verlag (2001), P. 152-153
- **Moderne Betriebssysteme**, *Andrew S. Tanenbaum*, 2nd edition, Pearson (2009), P. 223-226

Implementation of the Page Table

- Impact of the page length:
  - **Short pages**: Less capacity loss caused by internal fragmentation, but bigger page table
  - **Long pages**: Shorter page table, but more capacity loss caused by internal fragmentation
- Page tables are stored inside the main memory

$$\text{Maximum page table size} = \frac{\text{Virtual address space}}{\text{Page size}} * \text{Size of each page table entry}$$

- Maximum page table size with 32 bit operating systems:

$$\frac{4\,\text{GB}}{4\,\text{kB}} * 4\,\text{Bytes} = \frac{2^{32}\,\text{Bytes}}{2^{12}\,\text{Bytes}} * 2^2\,\text{Bytes} = 2^{22}\,\text{Bytes} = 4\,\text{MB}$$

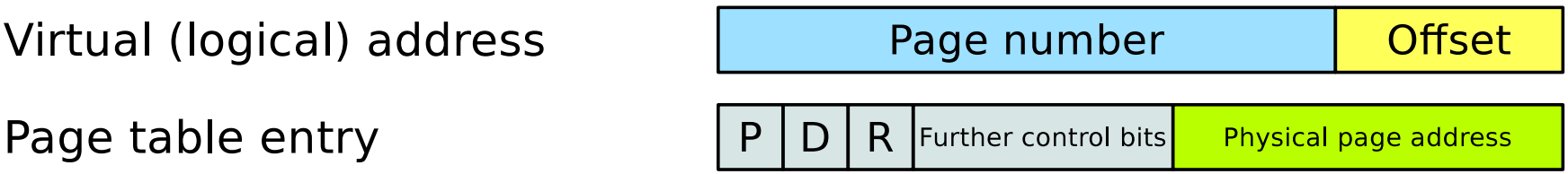
- Each process in a multitasking operating system requires a page table

In 64 bit operating systems, the page tables of the individual processes can be significantly larger

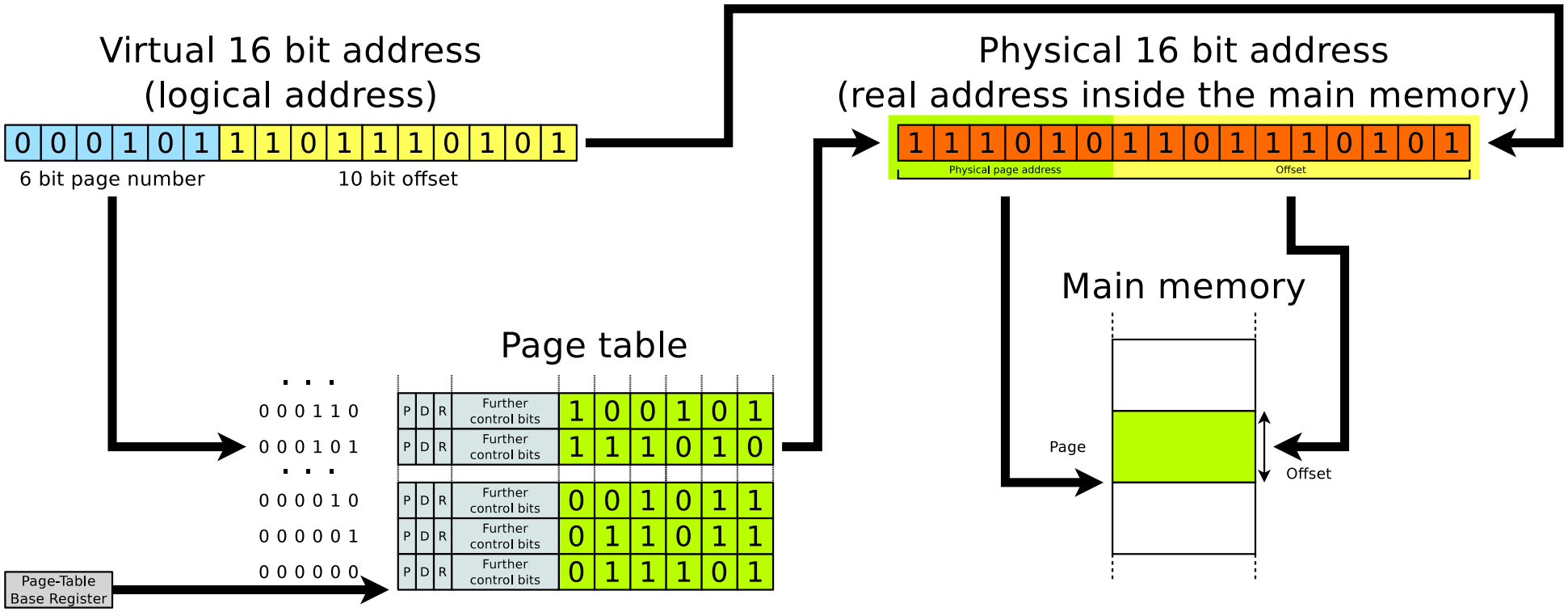
However, since most everyday processes do not require several gigabytes of memory, the overhead of managing the page tables on modern computers is low

Page Table Structure

- Each page table record contains among others:
  - **Present bit**: Specifies whether the page in stored inside the main memory
  - **Dirty bit** (*Modified-Bit*): Specifies whether the page has been modified
  - **Reference bit**: Specifies whether the page was referenced (even read operations!) $\Longrightarrow$ this is eventually relevant for the page replacement strategy used
  - **Further control bits**: Here is among others specified whether...
    - User mode processes have only read access to the page or write access too (**read/write bit**)
    - User-mode processes are allowed to access the page (**user/supervisor bit**)
    - Modifications are immediately passed down (**write-through**) or when the page is removed (**write-back**) from main memory (**write-through bit**)
    - The page may be loaded into the cache or not (**cache-disable bit**)
  - **Physical page address**: Is concatenated with the offset of the virtual address



Address Translation with Paging (single level)

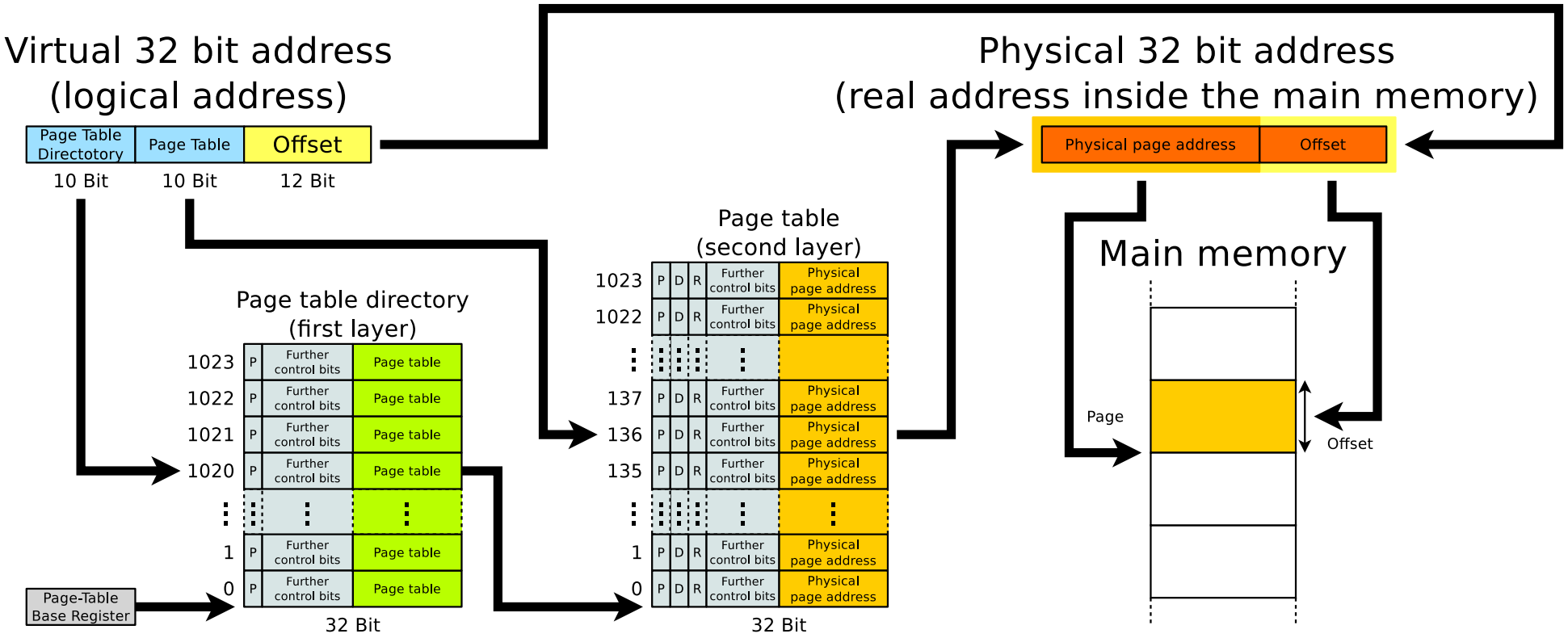


- Single level paging is sufficient in 16 bit architectures
- For architectures $\geq$ 32 bit the operating systems implement multi-level paging

2 registers enable the MMU to access the page table

- **Page-Table Base Register** (PTBR): Address where the page table of the current process starts
- **Page-Table Length Register** (PTLR): Length of the page table of the current process

Address Translation with Paging (2 levels)



The topic Paging is clearly explained by...

- **Betriebssysteme**, *Eduard Glatz*, 2nd edition, dpunkt (2010), P.450-457
- **Betriebssysteme**, *William Stallings*, 4th edition, Pearson (2003), S.394-399
- `http://wiki.osdev.org/Paging`

Why multi-level Paging?

We already know...

- In 32 bit operating systems with 4 kB page length, the page table of each process can be 4 MB in size (see slide 26)
- In 64 bit operating systems, the page tables can be much larger

- Multi-level paging reduces the main memory usage
  - When calculating a physical address, the operating system scans the pages of the different levels step by step
  - If required, individual pages of the different levels can be relocated to the swap storage to free up storage capacity in the main memory

Architecture	Page Table	Virtual Address Length	Partitioning[a]
IA32 (x86-32)	2 levels	32 Bits	10+10+12
IA32 with PAE[b]	3 levels	32 Bits	2+9+9+12
PPC64	3 levels	41 Bits	10+10+9+12
AMD64 (x86-64)	4 levels	48 Bits	9+9+9+9+12

[a] The last number indicates the length of the offset in bits. The remaining numbers indicate the lengths of the page tables.

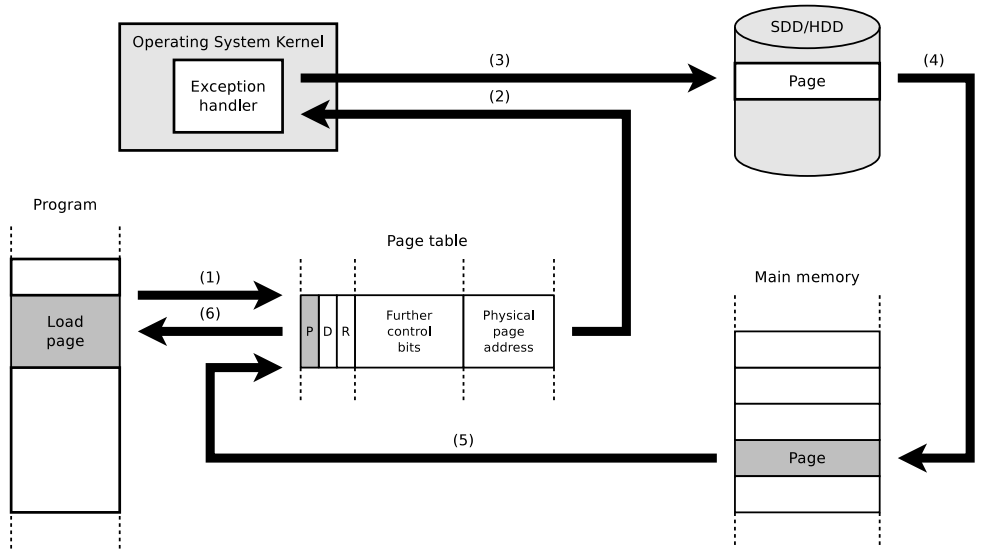
[b] PAE = Physical Address Extension. With this paging extension of the Pentium Pro processor, more than 4 GB of RAM can be addressed by the operating system. However, the memory usable per process is still limited to 4 GB.

A good description of this topic provides...

Architektur von Betriebssystemen, *Horst Wettstein*, Hanser (1984), P.249

Page Fault Exception

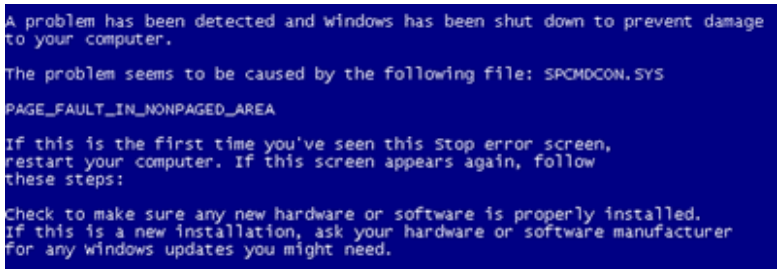
- A process tries (1) to access a page, which is not located in the physical main memory
  - The **present bit** in each page table record indicates whether the page is located inside main memory or not



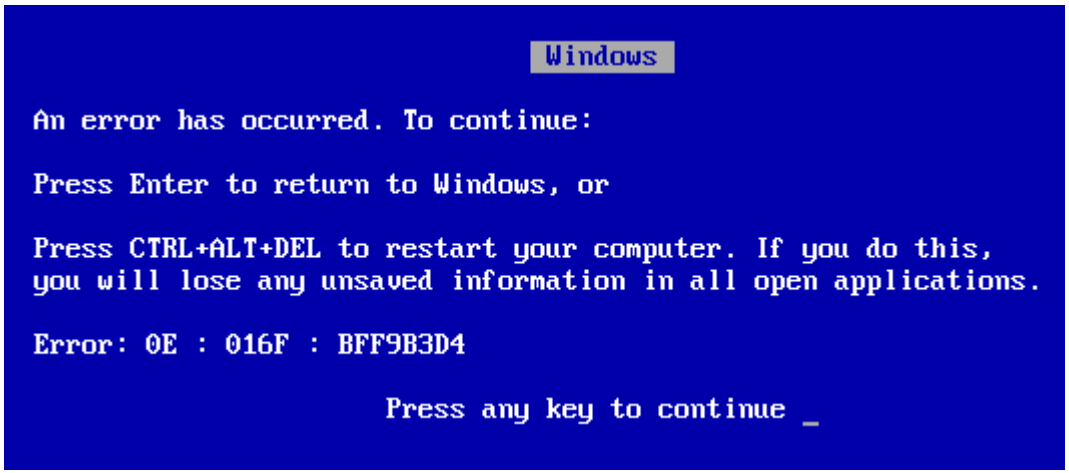
- A software interrupt (exception) is triggered (2) to switch from user mode to kernel mode
- The operating system...
  - allocates (3) the page by using the controller and the device driver on the swap memory (SSD/HDD)
  - copies (4) the page into a free page of the main memory
  - updates (5) the page table
  - returns control to the process (6)
    - The process again executes the instruction that caused the page fault

Access Violation Exception or General Protection Fault Exception

- Also called **Segmentation fault** or **Segmentation violation**
  - A paging issue, which has nothing to do with segmentation!



- A process tries to access a virtual memory address, which it is not allowed to access



- Result: Legacy Windows systems crash (blue screen), Linux returns the signal `SIGSEGV`
- Example: A process tries to carry out a write access to a read-only page

Source: Herold H. (1996) *UNIX-Systemprogrammierung*. 2. Auflage. Addison-Wesley
Image source: Wikipedia and `http://www.dtec-computers.com/images/jpg/computer_repair/blue-screen-of-death.gif`

Summary: Real Mode and Protected Mode

- **Real mode**
  - Operating mode of x86-compatible CPUs
  - The CPU accesses the main memory equal to an Intel 8086 CPU
  - No memory protection
    - Each process can access the entire main memory
- **Protected mode**
  - Modern operating systems (for x86) operate in protected mode and implement paging

Hit Rate and Miss Rate

An efficient memory management method for the main memory and cache...

- keeps those pages inside the memory that are accessed frequently
- identifies those pages that are unlikely to be accessed in the near future and replaces them if capacity is needed

- In case of a request to a computer memory, 2 results are possible:
  - **Hit**: Requested data is available
  - **Miss**: Requested data is missing
- 2 Key figures are used to evaluate the efficiency of a computer memory
  - **Hit rate**: The number of requests to the computer memory, with result in hit, divided by the total number of requests
    - Result is between 0 and 1
    - The greater the value, the better is the efficiency of the computer memory
  - **Miss rate**: The number of requests to the computer memory, with result in miss, divided by the total number of requests
    - Miss rate $= 1 -$ hit rate

Page Replacement Strategies

- It makes sense to keep the data ($\Longrightarrow$ **pages**) inside main memory, which is frequently accessed
- Some **replacement strategies**:
  - **OPT** (Optimal strategy)
  - **LRU** (Least Recently Used)
  - **LFU** (Least Frequently Used)
  - **FIFO** (First In First Out)
  - **Clock / Second Chance**
  - **TTL** (Time To Live)
  - **Random**

A well understandable explanation of the page seplacement strategies...

- OPT, FIFO, LRU and Clock provides **Operating Systems**, *William Stallings*, 4th edition, Prentice Hall (2001), P.355-363
- FIFO, LRU, LFU and Clock provides **Betriebssysteme**, *Carsten Vogt*, 1st edition, Spektrum Verlag (2001), P.162-163
- FIFO, LRU and Clock provides **Moderne Betriebssysteme**, *Andrew S. Tanenbaum*, 2nd edition, Pearson (2009), P.237-242
- FIFO, LRU, LFU and Clock provides **Betriebssysteme**, *Eduard Glatz*, 2nd edition, dpunkt (2010), P.471-476

Optimal strategy (OPT)

Image Source: Lukasfilm Games

- Replaces the page, which is **not accessed for the longest time in the future**
- Impossible to implement!
  - Reason: Nobody can predict the future
    - Therefore, the operating system must take into account the past
- OPT is used to evaluate the efficiency of other replacement strategies

Requests:	**1**	**2**	**3**	**4**	**1**	**2**	**5**	**1**	**2**	**3**	**4**	**5**
Page 1:	1	1	1	1	1	1	1	1	1	3	3	3
Page 2:		2	2	2	2	2	2	2	2	2	4	4
Page 3:			3	4	4	4	5	5	5	5	5	5

→ 7 Miss

The **requests** are requests for pages inside the virtual address space of a process. If the requested page is not inside the cache, it is read from the main memory or the swap

Least Recently Used (LRU)

- Replaces the page, which was **not accessed for the longest time**
- All pages are referenced in a queue
  - If a page is loaded into memory or referenced, it is moved to the front of the queue
  - If the memory has no more free capacity and a miss occurs, the page at the end of the queue is replaced
- Drawback: Ignores the number of accesses

Requests:	**1**	**2**	**3**	**4**	**1**	**2**	**5**	**1**	**2**	**3**	**4**	**5**
Page 1:	1	1	1	2	3	4	1	2	5	1	2	3
Page 2:		2	2	3	4	1	2	5	1	2	3	4
Page 3:			3	4	1	2	5	1	2	3	4	5

→ 10 Miss

Least Frequently Used (LFU)

- Replaces the page, which was **least often accessed**
- For each page inside the memory, a reference counter exists in the page table, in which the operating system stores the number of accesses
  - If the memory has no more free capacity and a miss occurs, the page is replaced, which has the lowest value in its reference counter
- Benefit: Takes into account the number of times pages are accessed
- Drawback: Pages which have been accessed often in the past, may block the memory

Requests:	**1**	**2**	**3**	**4**	**1**	**2**	**5**	**1**	**2**	**3**	**4**	**5**
Page 1:	$_1 1$	$_1 1$	$_1 1$	$_1 4$	$_1 4$	$_1 4$	$_1 5$	$_1 5$	$_1 5$	$_1 3$	$_1 4$	$_1 5$
Page 2:		$_1 2$	$_1 2$	$_1 2$	$_1 1$	$_1 1$	$_1 1$	$_2 1$	$_2 1$	$_2 1$	$_2 1$	$_2 1$
Page 3:			$_1 3$	$_1 3$	$_1 3$	$_1 2$	$_1 2$	$_1 2$	$_2 2$	$_2 2$	$_2 2$	$_2 2$

→ 10 Miss

First In First Out (FIFO)

- Replaces the page, which is stored **in memory for the longest time**
- Common assumption: increasing the memory results in fewer or, at worst, the same miss number
- Problem: Laszlo Belady demonstrated in 1969 that for certain access patterns, FIFO causes with an expanded memory capacity more miss events ($\Longrightarrow$ **Belady's anomaly**)
  - Until the discovery of Belady's Anomaly, FIFO was considered a good replacement strategy

Belady's Anomaly (1969)

Requests:	**1**	**2**	**3**	**4**	**1**	**2**	**5**	**1**	**2**	**3**	**4**	**5**	
Page 1:	1	1	1	4	4	4	5	5	5	5	5	5	
Page 2:		2	2	2	1	1	1	1	1	3	3	3	
Page 3:			3	3	3	2	2	2	2	2	4	4	→ 9 Miss

Page 1:	1	1	1	1	1	1	5	5	5	5	4	4	
Page 2:		2	2	2	2	2	2	1	1	1	1	5	
Page 3:			3	3	3	3	3	3	2	2	2	2	
Page 4:				4	4	4	4	4	4	3	3	3	→ 10 Miss

More information about Belady's anomaly

Belady, Nelson and Shedler. *An Anomaly in Space-time Characteristics of Certain Programs Running in a Paging Machine*. Communications of the ACM. Volume 12 Issue 6. June 1969

Clock / Second Chance

- This strategy uses the *reference bit* (see slide 27), which exists in the page table for each page
  - If a page is loaded into memory $\Longrightarrow$ reference bit $= 0$
  - If a page is accessed $\Longrightarrow$ reference bit $= 1$
- A pointer indicates the last accessed page
- In case of a miss, the memory is searched from the position of the pointer for the first page, whose reference bit has value 0
  - This page is replaced
  - For all pages, which are examined during the searching, where the reference bit has value 1, it is set to value 0

Requests:	**1**	**2**	**3**	**4**	**1**	**2**	**5**	**1**	**2**	**3**	**4**	**5**
Page 1:	$_0 1^X$	$_0 1$	$_0 1$	$_0 4^X$	$_0 4$	$_0 4$	$_0 5^X$	$_0 5$	$_0 5$	$_0 3^X$	$_0 4^X$	$_0 4$
Page 2:		$_0 2^X$	$_0 2$	$_0 2$	$_0 1^X$	$_0 1$	$_0 1$	$_1 1^X$	$_1 1$	$_1 1$	$_0 1$	$_0 5^X$
Page 3:			$_0 3^X$	$_0 3$	$_0 3$	$_0 2^X$	$_0 2$	$_0 2$	$_1 2^X$	$_1 2$	$_0 2$	$_0 2$

$\longrightarrow$ 10 Miss

Linux, BSD-UNIX, VAX/VMS (originally from Digital Equipment Corporation) and Windows NT 4.0 on uniprocessors systems implement the clock replacement strategy or variants of this strategy

Further Replacement Strategies

- **TTL** (Time To Live): Each page gets a time to live value, when it is stored in the memory
  - If the TTL has exceeded, the page can be replaced

This concept is not used in operating systems but it is useful for the caching of Web pages (Internet contents)

Interesting source: **Caching with expiration times**. *Gopalan P, Harloff H, Mehta A, Mihail M, Vishnoi N* (2002) `https://www.cc.gatech.edu/~mihail/www-papers/soda02.pdf`

- **Random**: Random pages are replaced
  - Benefits: Simple and resource-saving replacement strategy
    - Reason: No need to store information about the access behavior

The random replacement strategy is (was) used in practice

- The operating systems IBM OS/390 and Windows NT 4.0 on SMP systems use the random replacement strategy (Source OS/390: Pancham P, Chaudhary D, Gupta R. (2014) *Comparison of Cache Page Replacement Techniques to Enhance Cache Memory Performance*. International Journal of Computer Applications. Volume 98, Number 19) (Source NT4: `http://www.itprotoday.com/management-mobility/inside-memory-management-part-2`)
- The Intel i860 RISC CPU uses the Random replacement strategy for the cache (Source: Rhodehamel M. (1989) *The Bus Interface and Paging Units of the i860 Microprocessor*. Proceedings of the IEEE International Conference on Computer Design. S. 380-384)