

```

Sep 19 14:27:41 amd64 syslog-ng[7653]: STATS:
Sep 20 01:00:01 amd64 /usr/sbin/cron[2937]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 20 01:00:01 amd64 syslog-ng[7653]:
Sep 20 02:00:01 amd64 /usr/sbin/cron[2937]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 20 02:00:01 amd64 syslog-ng[7653]:
Sep 20 12:46:44 amd64 sshd[651]: Accepted publickey for esser from ::ffff:87.234.201.207 port 62004
Sep 20 12:48:41 amd64 syslog-ng[7653]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62105
Sep 20 12:54:44 amd64 sshd[651]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62514
Sep 20 15:27:35 amd64 syslog-ng[7653]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64242
Sep 20 15:27:35 amd64 syslog-ng[7653]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63375
Sep 20 16:11:11 amd64 sshd[20998]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63546
Sep 20 02:00:01 amd64 /usr/sbin/cron[2937]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 20 02:00:01 amd64 /usr/sbin/cron[2937]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 20 02:00:01 amd64 /usr/sbin/cron[2937]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 23 02:00:01 amd64 /usr/sbin/cron[2937]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 23 18:04:05 amd64 sshd[6554]: Accepted publickey for esser from ::ffff:192.168.1.5 port 59771 ssh2
Sep 23 18:04:05 amd64 syslog-ng[7653]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62093
Sep 23 18:04:34 amd64 sshd[6569]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62093
Sep 24 01:00:01 amd64 /usr/sbin/cron[12436]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 24 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 02:00:01 amd64 /usr/sbin/cron[13251]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 24 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 11:15:48 amd64 sshd[20998]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64456
Sep 24 13:49:08 amd64 sshd[23197]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61330
Sep 24 13:49:08 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: snd_seq_oss: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: snd_seq_oss: unsupported module, tainting kernel.
Sep 24 20:25:31 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 01:00:02 amd64 /usr/sbin/cron[662]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 25 01:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 02:00:01 amd64 /usr/sbin/cron[1484]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 25 02:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:25 amd64 sshd[8889]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64183
Sep 25 10:59:25 amd64 syslog-ng[7653]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64253
Sep 25 11:30:02 amd64 sshd[9372]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62029
Sep 25 11:59:05 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:05:37 amd64 sshd[11554]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62822
Sep 25 14:05:37 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:06:10 amd64 sshd[11566]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62951
Sep 25 14:07:17 amd64 sshd[11608]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63392
Sep 25 14:08:33 amd64 sshd[11630]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63709
Sep 25 15:25:33 amd64 sshd[12930]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62778

```



# Memory Management (2)

## Allocation of free memory

A specific amount of requested memory can be allocated as follows:

- **First Fit Method**
  - First sufficiently large free memory area is allocated.
- **Best Fit Method**
  - Smallest sufficiently large free memory area is allocated.
  - More complex, since whole list or bitmap must be searched.
  - Strong fragmentation of free memory into many small areas.

## Allocation of free memory

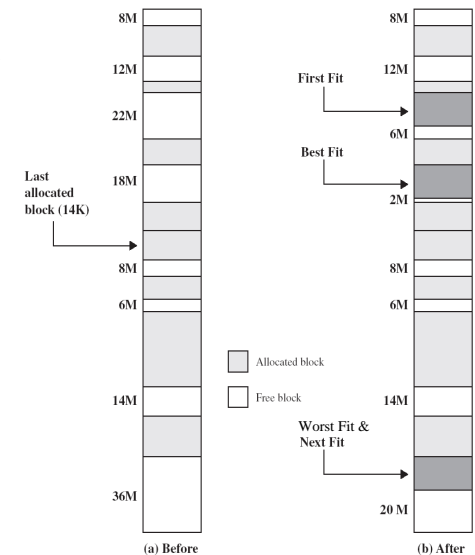
- **Worst Fit Method**
  - Largest free memory area is allocated.
  - Rather large free areas remain.
- **Quick Fit Method**
  - Keeping several lists of free memory areas for specific standard sizes (e.g. 4 KB, 8 KB, 12 KB etc.) leads to quick allocation.
  - After freeing an area it must be joined with its free neighbors (if there are any). This is more work when there are several lists.
  - Special variant: **Buddy System**

## Allocation of free memory: Example

Example for memory usage

- Before
- after

allocation of a 16 Mbyte block



## Memory Management: Buddy System

- Separate lists of free areas with sizes 1, 2, 4, 8, 16 etc. KByte (up to memory size).
- When freeing a memory area, only search one list to find out whether area can be joined with another free area.
- Block size always a power of 2 – not very efficient memory usage.

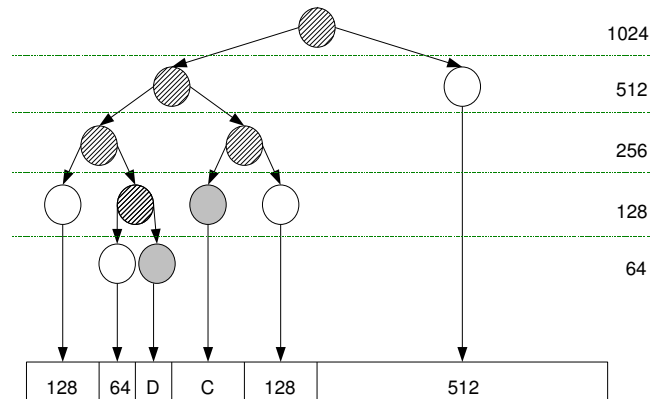
	0	128 K	256 K	384 K	512 K	640 K	768 K	896 K	1 M
Initial state	1024								
Request 70	A	128	256		512				
Request 35	A	B 64	256		512				
Request 80	A	B 64	C	128	512				
Release A	128	B 64	C	128	512				
Request 60	128	B	D	C	128	512			
Release B	128	64	D	C	128	512			
Release D	256		C	128	512				
Release C	1024								

## Segmentation (1)

- Partition Programs into several **segments**, e.g. code segment, data segment(s), stack segment(s) etc.
- Each Segment corresponds to a **linear address space** from 0 to a maximum value.
- Addresses are composed of:
  - a **segment address**
  - a **relative address within the segment**
 (two-dimensional address space)

## Buddy System: Tree Representation

Situation before release of process D



## Segmentation (2)

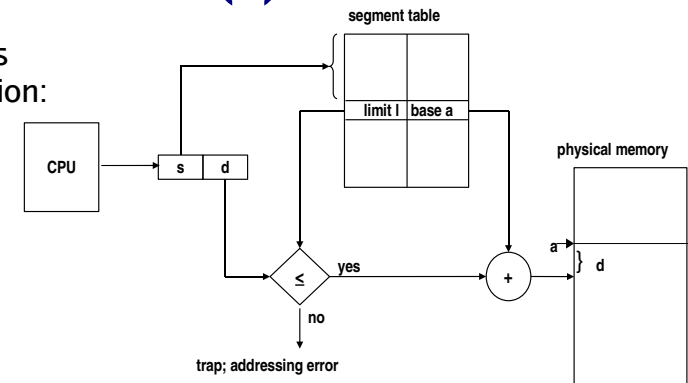
- **Advantages** of segmentation:
  - Easier to find free space for the smaller segments.
  - Segments can grow independently.
  - Not all segments have to be inside the main memory (**Swapping segments**).
  - Segments can have separate levels of protection (read, write, ...).
  - Differentiate between process-private segments and segments that are shared between several processes.

## Segmentation (3)

- Hardware must allow for segmentation.
- For each process there is a **segment table** in the main memory, holding at least three values for each segment:
  - Is the segment loaded in main memory?
  - Segment's start address in main memory
  - Length of segment

## Segmentation (5)

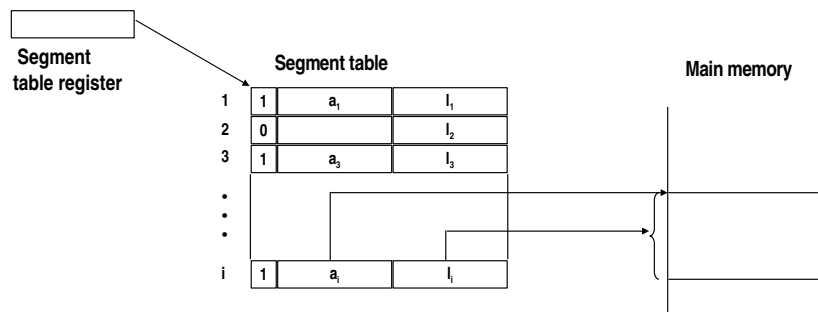
- Address translation:



- Each memory access requires an additional memory access (segment table). Special hardware registers must speed-up this process.

## Segmentation (4)

- Address of the segment table saved in the **segment table register**.



## Segmentation (6)

- Accessing a segment that is not inside the main memory causes a **Segment Fault**: Operating system must fetch segment into memory.
- Protection of a segment: Entries in segment table contain additional **protection code** (e.g. for read, write, execute accesses). Code generated by programmer (or compiler/linker).

## Segmentation (7)

- **Sharing** of a segment in main memory is **simple**:  
Let entries in segment tables of several processes point to the same main memory address.
- **Fragmentation**:  
Since every segment is contiguous inside the main memory, there is **external fragmentation**.

## Starting position

- Not enough memory for large programs  
-> Overlay programming
- Load parts of the program dynamically (from disk) when they are needed
- Programmer must care about creation of overlays

```

Sep 19 14:27:41 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 01:00:01 amd64 /usr/sbin/cron[29278]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 20 01:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 02:00:01 amd64 /usr/sbin/cron[30103]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 20 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:46:44 amd64 sshd[6516]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62004
Sep 20 12:46:44 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:48:41 amd64 sshd[6609]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62105
Sep 20 12:54:44 amd64 sshd[6694]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62514
Sep 20 15:27:35 amd64 sshd[9071]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64242
Sep 20 15:27:35 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:37:11 amd64 sshd[10102]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63375
Sep 20 16:37:11 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:38:10 amd64 sshd[10140]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63546
Sep 21 01:00:01 amd64 /usr/sbin/cron[17055]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 21 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 02:00:01 amd64 /usr/sbin/cron[17878]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 21 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:43:26 amd64 sshd[31088]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63397
Sep 21 17:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:53:39 amd64 sshd[31269]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64391
Sep 21 18:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 19:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 01:00:01 amd64 /usr/sbin/cron[4674]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 22 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 02:00:01 amd64 /usr/sbin/cron[5499]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 22 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 20:23:21 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 01:00:01 amd64 /usr/sbin/cron[24720]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 23 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 02:00:01 amd64 /usr/sbin/cron[25555]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 23 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:05 amd64 sshd[6554]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64456
Sep 23 18:04:05 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:06 amd64 sshd[6606]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64456
Sep 24 01:00:01 amd64 /usr/sbin/cron[12436]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 24 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 02:00:01 amd64 /usr/sbin/cron[13252]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 24 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 11:15:48 amd64 sshd[20998]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64456
Sep 24 11:15:48 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 13:49:08 amd64 sshd[21971]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61330
Sep 24 13:49:08 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: end_seq_mid_event: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: end_seq_oss: unsupported module, tainting kernel.
Sep 24 20:25:31 amd64 sshd[29399]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62566
Sep 24 20:25:31 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 01:00:02 amd64 /usr/sbin/cron[662]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 25 02:00:01 amd64 /usr/sbin/cron[1484]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 25 02:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:25 amd64 sshd[8889]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64183
Sep 25 10:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:47 amd64 sshd[8921]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64253
Sep 25 11:30:02 amd64 sshd[9372]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62029
Sep 25 11:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:05:37 amd64 sshd[11541]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62822
Sep 25 14:05:37 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:06:10 amd64 sshd[11586]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62951
Sep 25 14:07:17 amd64 sshd[11608]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63392
Sep 25 14:08:33 amd64 sshd[11630]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63709
Sep 25 15:25:33 amd64 sshd[12930]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62778
    
```

**Non-contiguous  
(virtual)  
memory partitioning**

## Overlay Programming

Turbo Pascal, about 1985-90:

```
program bigproject;
```

```
overlay procedure customerdata;
```

```
...
```

```
overlay procedure storedata;
```

```
...
```

```
{ main program }
```

```
begin
```

```
  while input <> "exit" do begin
```

```
    case input of
```

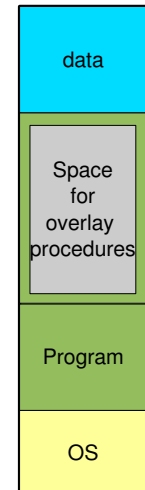
```
      "customer": customerdata;
```

```
      "storage": storedate;
```

```
    end;
```

```
  end;
```

```
end.
```



## Solution of the problem

- Virtual memory that is big enough for the whole program
- Program sees memory area that was allocated for it – how much "real" memory the system has, does not matter (for the program)

## Virtual Memory Management (Paging)

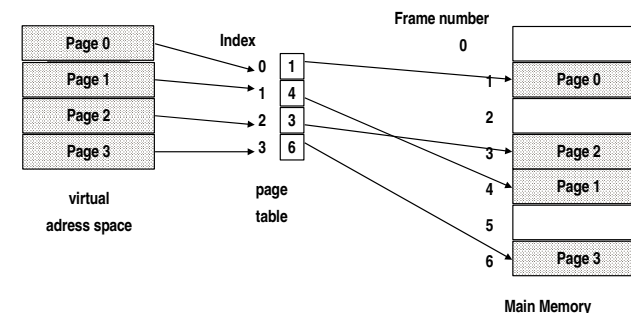
- Calculation of **physical** memory address from the **virtual** address given by the program
  - happens at the program's runtime,
  - is transparent for the program,
  - must be hardware-assisted.
- **Advantages** of virtual memory management:
  - Allocation of main memory is simple.
  - No external fragmentation, small internal fragmentation.
  - Nothing to do for the programmer.

## Virtual Memory Management (Paging)

- Partition the address space into **pages** of **fixed size** and the main memory into **page frames** of **equal size**.
  - Typical page sizes: 512 Byte – 8192 Byte (always a power of 2).
- The linear contiguous address space of a process ("**virtual**" **address space**) is mapped onto non-contiguous page-frames.
- Operating system manages **one** single list of free page frames.

## Virtual Address Space (1)

- With Paging, the relationship between program addresses and physical (main memory) addresses is only established at runtime by using the page tables.



## Virtual Address Space (2)

- The addresses used by the program are therefore called **virtual addresses**.
- The **virtual address space** of a program is the **linear, contiguous address space** that can be used by the program.

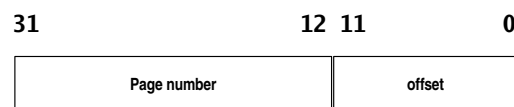
## Address translation with Paging (2)

- For each process there is a **page table**. It contains, for each process page:
  - the information whether the page is in main memory,
  - the number of the page frame in main memory, which holds the page.
- A special Register contains the start address of the page table for den current (running) process.
- Page number is used as index into the page table.

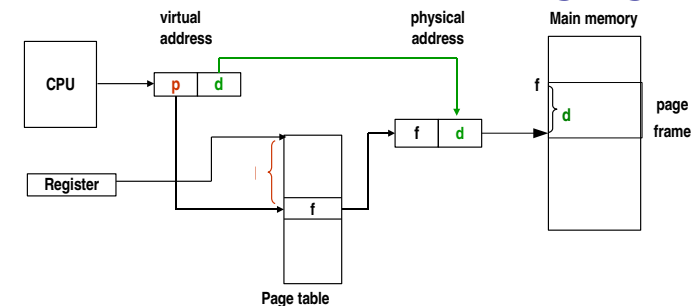
## Address translation with Paging (1)

- Program address is split into two parts:
  - a page number
  - a relative address (offset) into the page

Example: 32 bit address, page size of 4096 (=2<sup>12</sup>) bytes:



## Address translation with Paging (3)



- Each memory access requires an additional memory access (page table). Special caches (hardware) must speed-up this process!
- Page not in main memory -> special exception, so called **page fault**.

## Virtual memory in general (1)

- Several processes can be kept in memory effectively  
--> **better system utilization**
- A process can request much **more memory** than **physically available**

## Virtual memory in general (3)

- „*thrashing*“ (see later): Processor spends most of its time swapping process parts in and out instead of running process code
- **Locality principle:**
  - Data and program code accesses often locally grouped;  
--> it is ok to assume, that only few process parts must be kept in memory simultaneously during a short period of time

## Virtual memory in general (2)

- General procedure:
  - Only parts of the process reside in the physical (main) memory
  - When accessing an address that is not in memory:
    - OS sets process state to *blocked*
    - OS executes Disk-I/O read command
    - After loading the missing part (page or segment) an I/O interrupt occurs
    - OS sets process state back to *ready*