



Prof. Dr. Florian Künzner

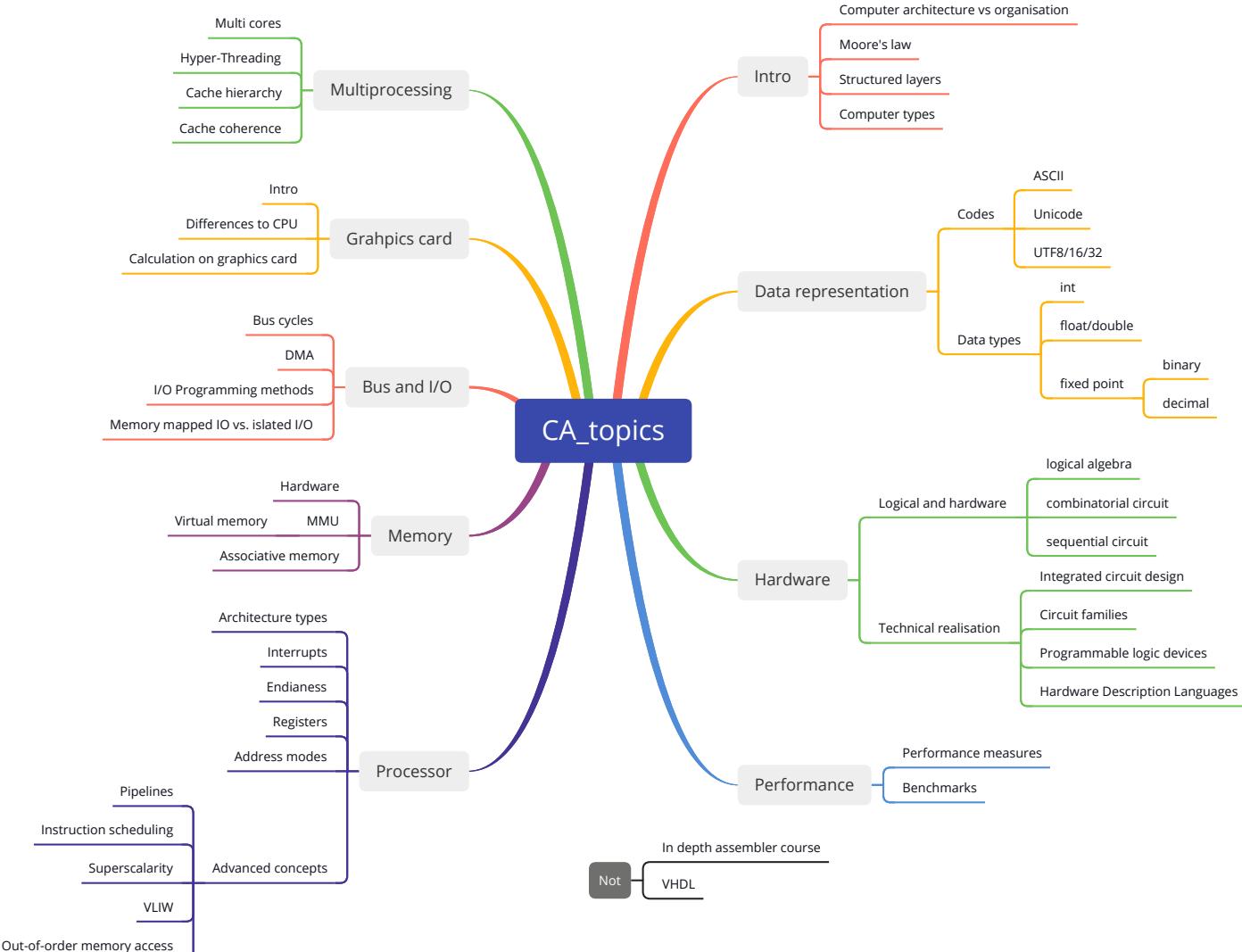
Technical University of Applied Sciences Rosenheim, Computer Science

CA 9 – MMU

The lecture is based on the work and the documents of Prof. Dr. Theodor Tempelmeier



Goal





Goal

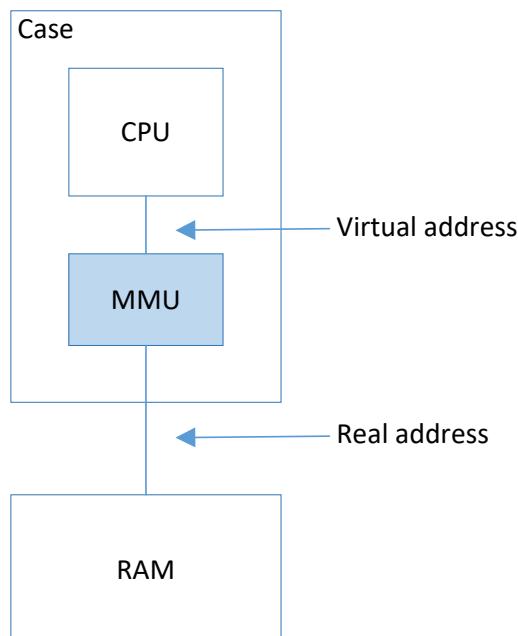
CA::MMU

- MMU
- Virtual addresses
- 1 level page table
- 2 level page table
- N level page table



Overview

MMU - Memory management unit



- Special hardware inside the CPU case
- Dynamic address conversion at runtime



Overview

Goals of virtual memory management:

- Aggregation of different memory sources
- Suitable abstraction for developers
 - Only one type of addresses
 - Linearly accessible memory
- Use of full theoretical available memory
- Multiple processes: Multiprogramming
- Shared libraries (DLL, so)
- Memory protection
- Parallelism (multiple cores: parallel processes)
- High performance memory access



Overview

Goals of virtual memory management:

- Aggregation of different memory sources
- Suitable abstraction for developers
 - Only one type of addresses
 - Linearly accessible memory
- Use of full theoretical available memory
- Multiple processes: Multiprogramming
- Shared libraries (DLL, so)
- Memory protection
- Parallelism (multiple cores: parallel processes)
- High performance memory access



Overview

Goals of virtual memory management:

- Aggregation of different memory sources
- Suitable abstraction for developers
 - Only one type of addresses
 - Linearly accessible memory
- Use of full theoretical available memory
- Multiple processes: Multiprogramming
- Shared libraries (DLL, so)
- Memory protection
- Parallelism (multiple cores: parallel processes)
- High performance memory access



Overview

Goals of virtual memory management:

- Aggregation of different memory sources
- Suitable abstraction for developers
 - Only one type of addresses
 - Linearly accessible memory
- Use of full theoretical available memory
- Multiple processes: Multiprogramming
- Shared libraries (DLL, so)
- Memory protection
- Parallelism (multiple cores: parallel processes)
- High performance memory access



Overview

Goals of virtual memory management:

- Aggregation of different memory sources
- Suitable abstraction for developers
 - Only one type of addresses
 - Linearly accessible memory
- Use of full theoretical available memory
- Multiple processes: Multiprogramming
- Shared libraries (DLL, so)
- Memory protection
- Parallelism (multiple cores: parallel processes)
- High performance memory access



Overview

Goals of virtual memory management:

- Aggregation of different memory sources
- Suitable abstraction for developers
 - Only one type of addresses
 - Linearly accessible memory
- Use of full theoretical available memory
- Multiple processes: Multiprogramming
- Shared libraries (DLL, so)
- Memory protection
- Parallelism (multiple cores: parallel processes)
- High performance memory access



Overview

Goals of virtual memory management:

- Aggregation of different memory sources
- Suitable abstraction for developers
 - Only one type of addresses
 - Linearly accessible memory
- Use of full theoretical available memory
- Multiple processes: Multiprogramming
- Shared libraries (DLL, so)
- Memory protection
- Parallelism (multiple cores: parallel processes)
- High performance memory access



Overview

Goals of virtual memory management:

- Aggregation of different memory sources
- Suitable abstraction for developers
 - Only one type of addresses
 - Linearly accessible memory
- Use of full theoretical available memory
- Multiple processes: Multiprogramming
- Shared libraries (DLL, so)
- Memory protection
- Parallelism (multiple cores: parallel processes)
- High performance memory access



Overview

Goals of virtual memory management:

- Aggregation of different memory sources
- Suitable abstraction for developers
 - Only one type of addresses
 - Linearly accessible memory
- Use of full theoretical available memory
- Multiple processes: Multiprogramming
- Shared libraries (DLL, so)
- Memory protection
- Parallelism (multiple cores: parallel processes)
- High performance memory access



Situation without virtual addresses

We try to recap the situation without virtual addresses.

C example

```
1 int x;  
2 int y;  
3  
4 x = x + y;
```

Compiled assembler example

```
1 ;CODE of ADD  
2 ;real address of x  
3 ;real address of y  
4  
5 X: ;value of x  
6 Y: ;value of y
```



Situation without virtual addresses

We try to recap the situation without virtual addresses.

C example

```
1 int x;  
2 int y;  
3  
4 x = x + y;
```

Compiled assembler example

```
1 ;CODE of ADD  
2 ;real address of x  
3 ;real address of y  
4  
5 X: ;value of x  
6 Y: ;value of y
```



Situation without virtual addresses

Absolute addresses

Compiled assembler

```
1 ...
2 1000: ;code of ADD
3 1004: ;real address of x
4 1008: ;real address of y
5 ...
6 1024: ;value of x
7 1028: ;value of y
```

Facts

- An *.elf/*.exe with exactly this code and addresses (called image) is saved on the harddisk
- The *.elf/*.exe has to be loaded exactly at this memory addresses (starting with 1000)

Problems

- Program can't be started twice at the same time
- Different programs can't use the same address



Situation without virtual addresses

Absolute addresses

Compiled assembler

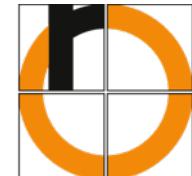
```
1 ...
2 1000: ;code of ADD
3 1004: ;real address of x
4 1008: ;real address of y
5 ...
6 1024: ;value of x
7 1028: ;value of y
```

Facts

- An *.elf/*.exe with exactly this code and addresses (called image) is saved on the harddisk
- The *.elf/*.exe has to be loaded exactly at this memory addresses (starting with 1000)

Problems

- Program can't be started twice at the same time
- Different programs can't use the same address



Situation without virtual addresses

Absolute addresses

Compiled assembler

```
1 ...  
2 1000: ;code of ADD  
3 1004: ;real address of x  
4 1008: ;real address of y  
5 ...  
6 1024: ;value of x  
7 1028: ;value of y
```

Facts

- An *.elf/*.exe with exactly this code and addresses (called image) is saved on the harddisk
- The *.elf/*.exe has to be loaded exactly at this memory addresses (starting with 1000)

Problems

- Program can't be started twice at the same time
- Different programs can't use the same address



Situation without virtual addresses

Absolute addresses

Compiled assembler

```
1 ...  
2 1000: ;code of ADD  
3 1004: ;real address of x  
4 1008: ;real address of y  
5 ...  
6 1024: ;value of x  
7 1028: ;value of y
```

Facts

- An *.elf/* .exe with exactly this code and addresses (called image) is saved on the harddisk
- The *.elf/* .exe has to be loaded exactly at this memory addresses (starting with 1000)

Problems

- Program can't be started twice at the same time
- Different programs can't use the same address



Situation without virtual addresses

Absolute addresses

Compiled assembler

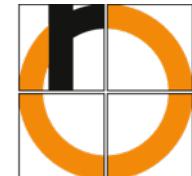
```
1 ...  
2 1000: ;code of ADD  
3 1004: ;real address of x  
4 1008: ;real address of y  
5 ...  
6 1024: ;value of x  
7 1028: ;value of y
```

Facts

- An *.elf/* .exe with exactly this code and addresses (called image) is saved on the harddisk
- The *.elf/* .exe has to be loaded exactly at this memory addresses (starting with 1000)

Problems

- Program can't be started twice at the same time
- Different programs can't use the same address



Situation without virtual addresses

Absolute addresses

Compiled assembler

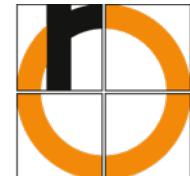
```
1 ...  
2 1000: ;code of ADD  
3 1004: ;real address of x  
4 1008: ;real address of y  
5 ...  
6 1024: ;value of x  
7 1028: ;value of y
```

Facts

- An *.elf/* .exe with exactly this code and addresses (called image) is saved on the harddisk
- The *.elf/* .exe has to be loaded exactly at this memory addresses (starting with 1000)

Problems

- Program can't be started twice at the same time
- Different programs can't use the same address



Situation without virtual addresses

Absolute addresses

Compiled assembler

```
1 ...  
2 1000: ;code of ADD  
3 1004: ;real address of x  
4 1008: ;real address of y  
5 ...  
6 1024: ;value of x  
7 1028: ;value of y
```

Facts

- An *.elf/* .exe with exactly this code and addresses (called image) is saved on the harddisk
- The *.elf/* .exe has to be loaded exactly at this memory addresses (starting with 1000)

Problems

- Program can't be started twice at the same time
- Different programs can't use the same address



Situation without virtual addresses

Position independent code (i.e. PC relative)

Compiled assembler

```
1 ...
2 1000: ;code of ADD
3 1004: 20 ;(offset of x to PC)
4 1008: 20 ;(offset of y to PC)
5 ...
6 1024: ;value of x
7 1028: ;value of y
```

Facts

- The operands are addressed with an relative offset to the PC
- The program can be loaded on any address in the memory
- It is also possible to relatively address to the SP

Problems

- Address translation required to obtain addresses of x and y at runtime



Situation without virtual addresses

Position independent code (i.e. PC relative)

Compiled assembler

```
1 ...
2 1000: ;code of ADD
3 1004: 20 ;(offset of x to PC)
4 1008: 20 ;(offset of y to PC)
5 ...
6 1024: ;value of x
7 1028: ;value of y
```

Facts

- The operands are addressed with an relative offset to the PC
- The program can be loaded on any address in the memory
- It is also possible to relatively address to the SP

Problems

- Address translation required to obtain addresses of x and y at runtime



Situation without virtual addresses

Position independent code (i.e. PC relative)

Compiled assembler

```
1 ...  
2 1000: ;code of ADD  
3 1004: 20 ;(offset of x to PC)  
4 1008: 20 ;(offset of y to PC)  
5 ...  
6 1024: ;value of x  
7 1028: ;value of y
```

Facts

- The operands are addressed with an relative offset to the PC
- The program can be loaded on any address in the memory
- It is also possible to relatively address to the SP

Problems

- Address translation required to obtain addresses of x and y at runtime



Situation without virtual addresses

Position independent code (i.e. PC relative)

Compiled assembler

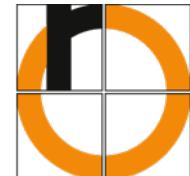
```
1 ...  
2 1000: ;code of ADD  
3 1004: 20 ;(offset of x to PC)  
4 1008: 20 ;(offset of y to PC)  
5 ...  
6 1024: ;value of x  
7 1028: ;value of y
```

Facts

- The operands are addressed with a relative offset to the PC
- The program can be loaded on any address in the memory
- It is also possible to relatively address to the SP

Problems

- Address translation required to obtain addresses of x and y at runtime



Situation without virtual addresses

Position independent code (i.e. PC relative)

Compiled assembler

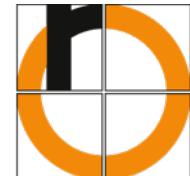
```
1 ...  
2 1000: ;code of ADD  
3 1004: 20 ;(offset of x to PC)  
4 1008: 20 ;(offset of y to PC)  
5 ...  
6 1024: ;value of x  
7 1028: ;value of y
```

Facts

- The operands are addressed with an relative offset to the PC
- The program can be loaded on any address in the memory
- It is also possible to relatively address to the SP

Problems

- Address translation required to obtain addresses of x and y at runtime



Situation without virtual addresses

Position independent code (i.e. PC relative)

Compiled assembler

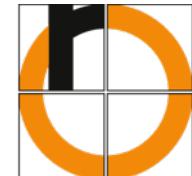
```
1 ...  
2 1000: ;code of ADD  
3 1004: 20 ;(offset of x to PC)  
4 1008: 20 ;(offset of y to PC)  
5 ...  
6 1024: ;value of x  
7 1028: ;value of y
```

Facts

- The operands are addressed with an relative offset to the PC
- The program can be loaded on any address in the memory
- It is also possible to relatively address to the SP

Problems

- Address translation required to obtain addresses of x and y at runtime



Situation without virtual addresses

Position independent code (i.e. PC relative)

Compiled assembler

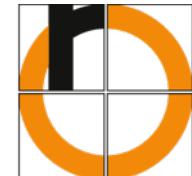
```
1 ...  
2 1000: ;code of ADD  
3 1004: 20 ;(offset of x to PC)  
4 1008: 20 ;(offset of y to PC)  
5 ...  
6 1024: ;value of x  
7 1028: ;value of y
```

Facts

- The operands are addressed with an relative offset to the PC
- The program can be loaded on any address in the memory
- It is also possible to relatively address to the SP

Problems

- Address translation required to obtain addresses of x and y at runtime



Situation without virtual addresses

Relocation

Compiled assembler

```
1 ...  
2 2000: ;code of ADD  
3 2004: 2024 ;real address of x  
4 2008: 2028 ;real address of y  
5 ...  
6 2024: ;value of x  
7 2028: ;value of y
```

Facts

- The program is compiled with absolute addresses
- When the program is loaded into memory, the addresses are changed.
- The relocation table contains the information which addresses have to be changed

Problems

- Address translation required on startup



Situation without virtual addresses

Relocation

Compiled assembler

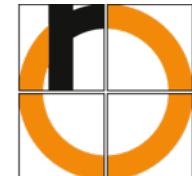
```
1 ...  
2 2000: ;code of ADD  
3 2004: 2024 ;real address of x  
4 2008: 2028 ;real address of y  
5 ...  
6 2024: ;value of x  
7 2028: ;value of y
```

Facts

- The program is compiled with absolute addresses
- When the program is loaded into memory, the addresses are changed.
- The relocation table contains the information which addresses have to be changed

Problems

- Address translation required on startup



Situation without virtual addresses

Relocation

Compiled assembler

```
1 ...  
2 2000: ;code of ADD  
3 2004: 2024 ;real address of x  
4 2008: 2028 ;real address of y  
5 ...  
6 2024: ;value of x  
7 2028: ;value of y
```

Facts

- The program is compiled with absolute addresses
- When the program is loaded into memory, the addresses are changed.
- The relocation table contains the information which addresses have to be changed

Problems

- Address translation required on startup



Situation without virtual addresses

Relocation

Compiled assembler

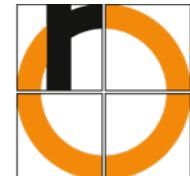
```
1 ...  
2 2000: ;code of ADD  
3 2004: 2024 ;real address of x  
4 2008: 2028 ;real address of y  
5 ...  
6 2024: ;value of x  
7 2028: ;value of y
```

Facts

- The program is compiled with absolute addresses
- When the program is loaded into memory, the addresses are changed.
- The relocation table contains the information which addresses have to be changed

Problems

- Address translation required on startup



Situation without virtual addresses

Relocation

Compiled assembler

```
1 ...  
2 2000: ;code of ADD  
3 2004: 2024 ;real address of x  
4 2008: 2028 ;real address of y  
5 ...  
6 2024: ;value of x  
7 2028: ;value of y
```

Facts

- The program is compiled with absolute addresses
- When the program is loaded into memory, the addresses are changed.
- The relocation table contains the information which addresses have to be changed

Problems

- Address translation required on startup



Situation without virtual addresses

Relocation

Compiled assembler

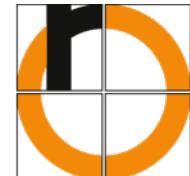
```
1 ...  
2 2000: ;code of ADD  
3 2004: 2024 ;real address of x  
4 2008: 2028 ;real address of y  
5 ...  
6 2024: ;value of x  
7 2028: ;value of y
```

Facts

- The program is compiled with absolute addresses
- When the program is loaded into memory, the addresses are changed.
- The relocation table contains the information which addresses have to be changed

Problems

- Address translation required on startup



Situation without virtual addresses

Relocation

Compiled assembler

```
1 ...  
2 2000: ;code of ADD  
3 2004: 2024 ;real address of x  
4 2008: 2028 ;real address of y  
5 ...  
6 2024: ;value of x  
7 2028: ;value of y
```

Facts

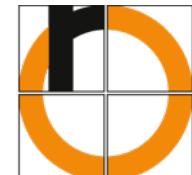
- The program is compiled with absolute addresses
- When the program is loaded into memory, the addresses are changed.
- The relocation table contains the information which addresses have to be changed

Problems

- Address translation required on startup

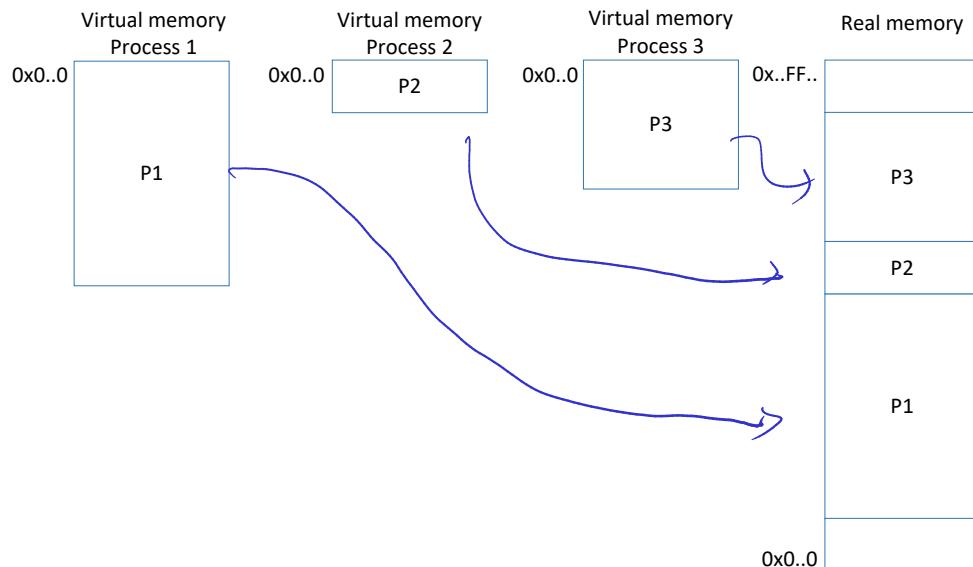


Towards virtual addresses



Towards virtual addresses

Idea 1: Separation of virtual and real address space

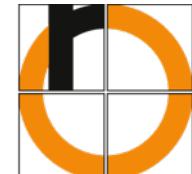


Facts

- Every process starts with address 0x0.0 until its upper limit
- The process needs to know its upper limit at compile time

Problems

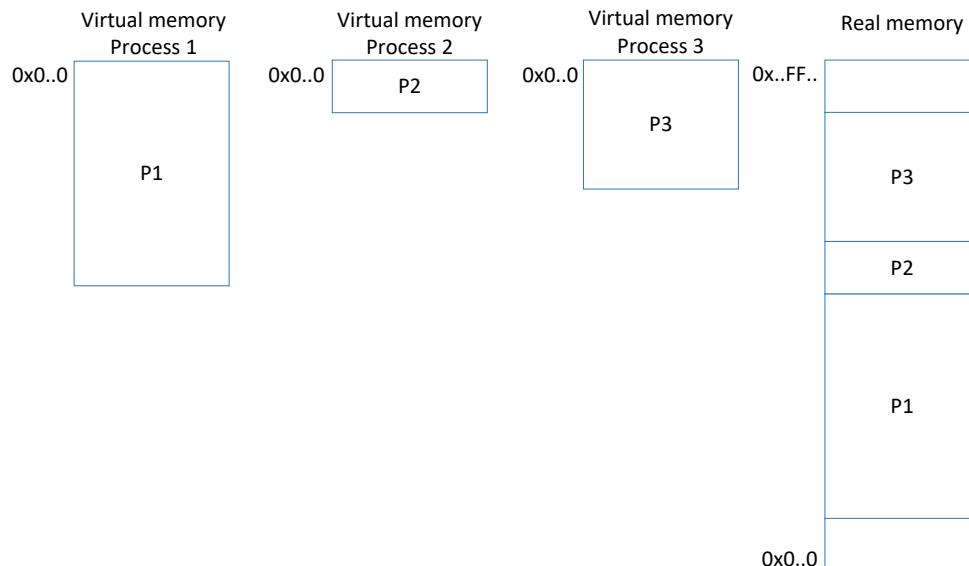
- Real memory fragmentation
- Maximum address space is limited by the real memory



Towards virtual addresses

Idea 1: Separation of virtual and real address space

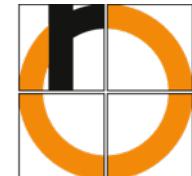
Facts



- Every process starts with address 0x0.0 until its upper limit
- The process needs to know its upper limit at compile time

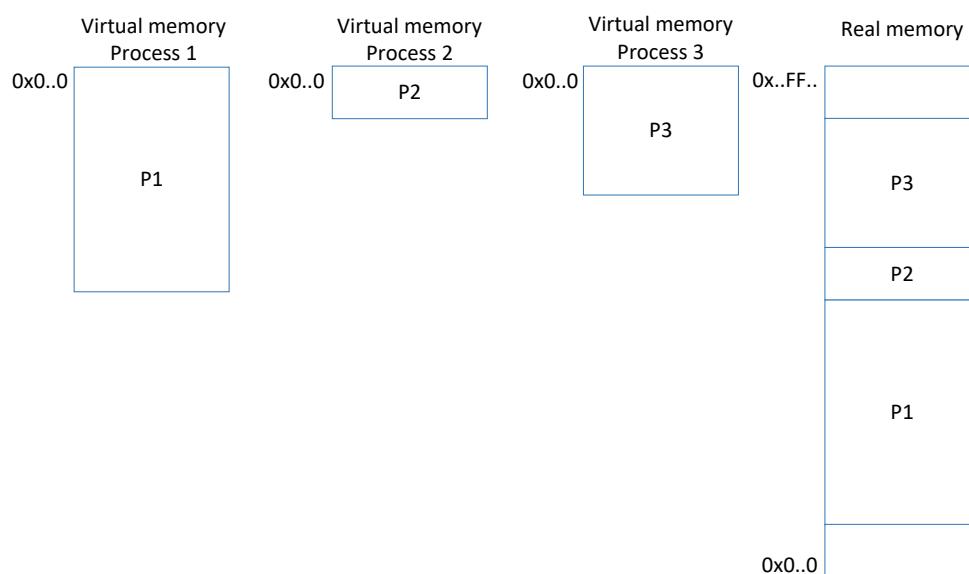
Problems

- Real memory fragmentation
- Maximum address space is limited by the real memory



Towards virtual addresses

Idea 1: Separation of virtual and real address space

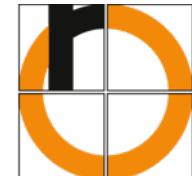


Facts

- Every process starts with address 0x0..0 until its upper limit
- The process needs to know its upper limit at compile time

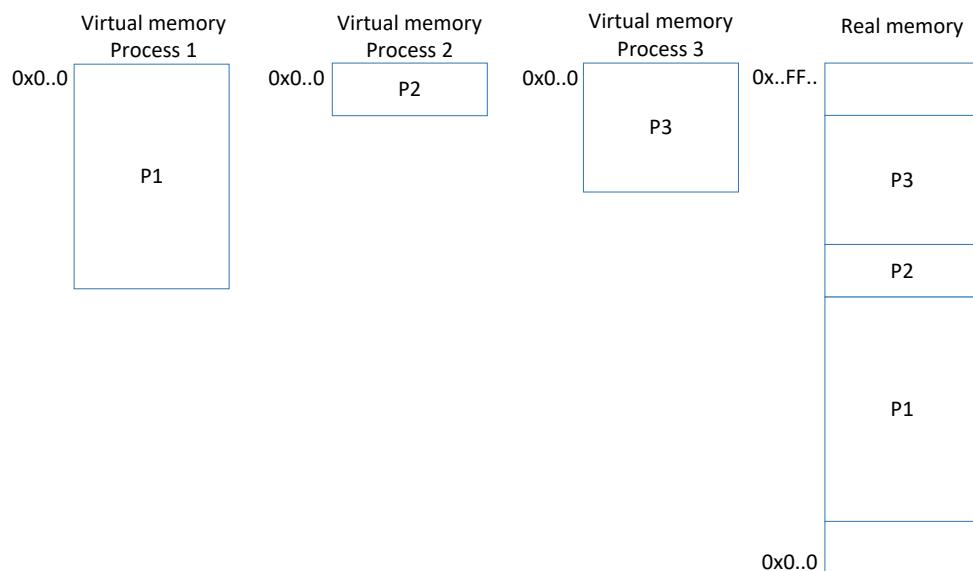
Problems

- Real memory fragmentation
- Maximum address space is limited by the real memory



Towards virtual addresses

Idea 1: Separation of virtual and real address space

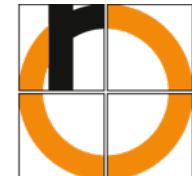


Facts

- Every process starts with address 0x0..0 until its upper limit
- The process needs to know its upper limit at compile time

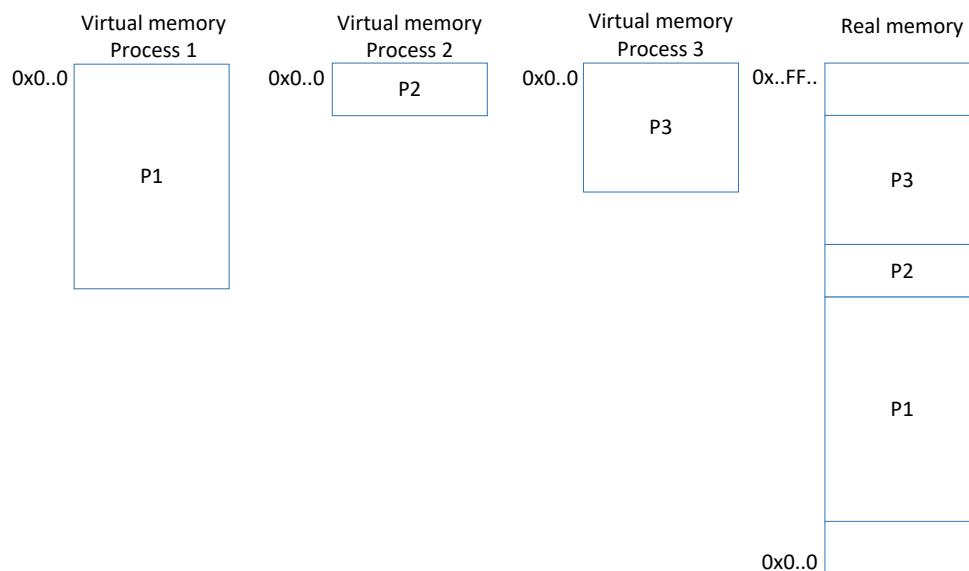
Problems

- Real memory fragmentation
- Maximum address space is limited by the real memory



Towards virtual addresses

Idea 1: Separation of virtual and real address space

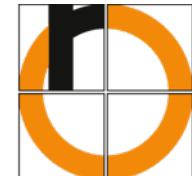


Facts

- Every process starts with address 0x0..0 until its upper limit
- The process needs to know its upper limit at compile time

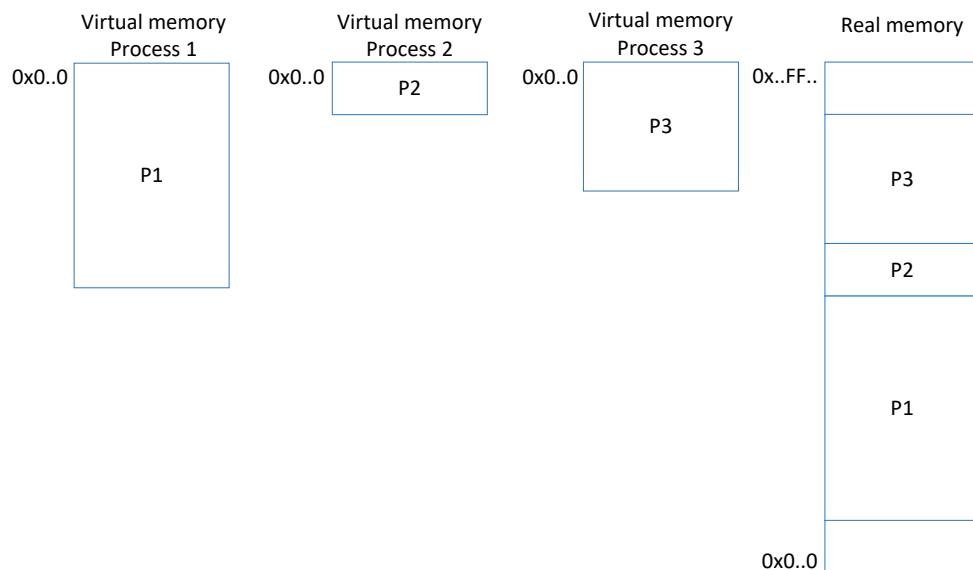
Problems

- Real memory fragmentation
- Maximum address space is limited by the real memory



Towards virtual addresses

Idea 1: Separation of virtual and real address space



Facts

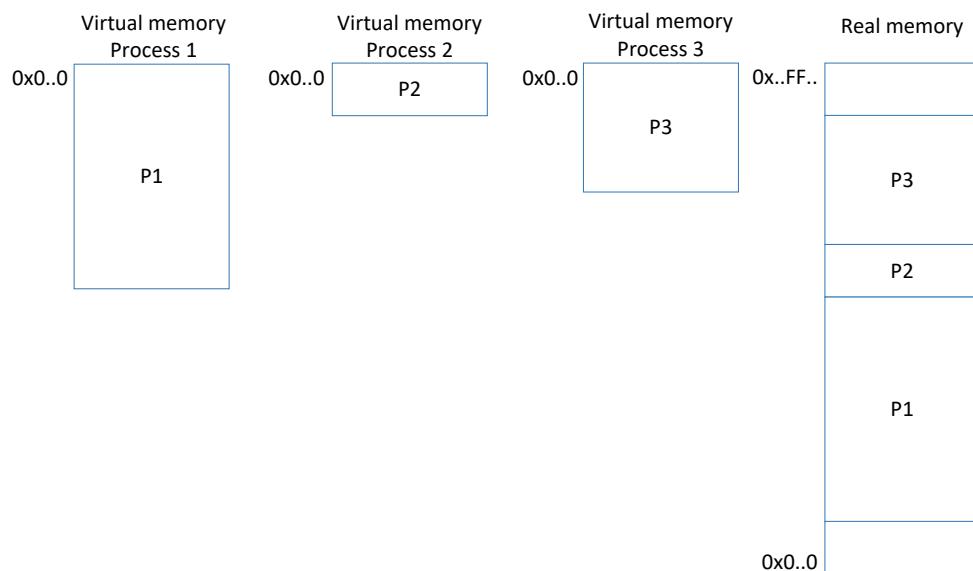
- Every process starts with address 0x0..0 until its upper limit
- The process needs to know its upper limit at compile time

Problems

- Real memory fragmentation
- Maximum address space is limited by the real memory

Towards virtual addresses

Idea 1: Separation of virtual and real address space

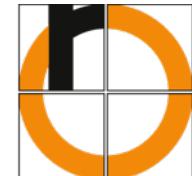


Facts

- Every process starts with address 0x0..0 until its upper limit
- The process needs to know its upper limit at compile time

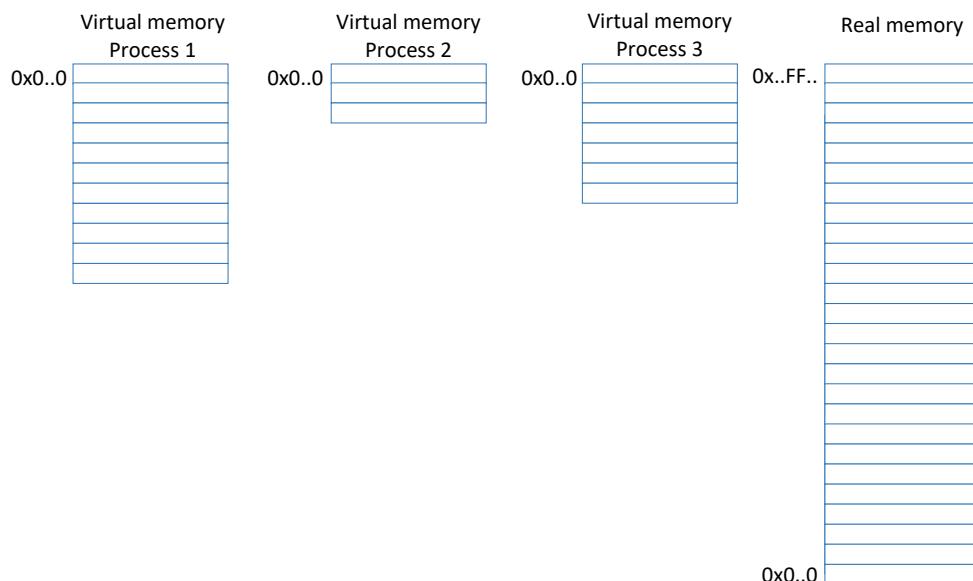
Problems

- Real memory fragmentation
- Maximum address space is limited by the real memory



Towards virtual addresses

Idea 2: Introduce pages and frames



Facts

- The memory is divided into small pieces of equal size called pages and frames
- Page: virtual address space
- Frame: real address space

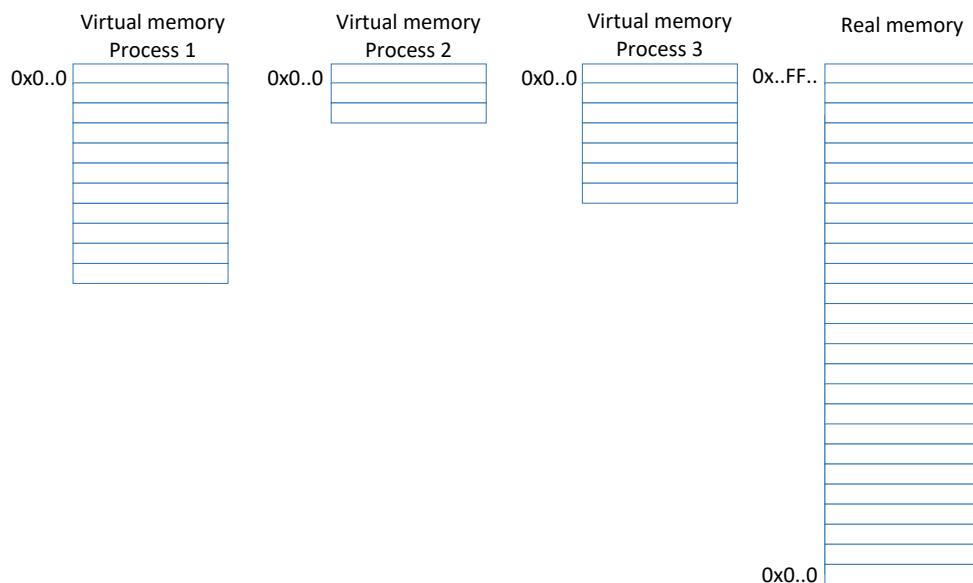
Advantages

- No (little) real memory fragmentation
- No upper memory limit required at compile time
- Virtual address space > real address space



Towards virtual addresses

Idea 2: Introduce pages and frames



Facts

- The memory is divided into small pieces of equal size called pages and frames
- Page: virtual address space
- Frame: real address space

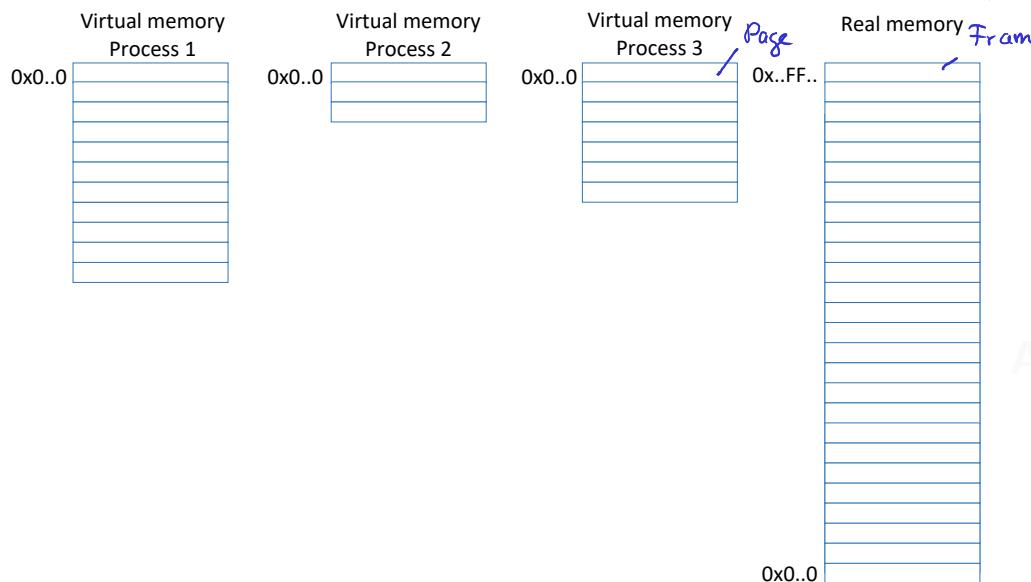
Advantages

- No (little) real memory fragmentation
- No upper memory limit required at compile time
- Virtual address space > real address space



Towards virtual addresses

Idea 2: Introduce pages and frames



Physical
Page

Facts

- The memory is divided into small pieces of equal size called pages and frames

- Page: virtual address space
- Frame: real address space

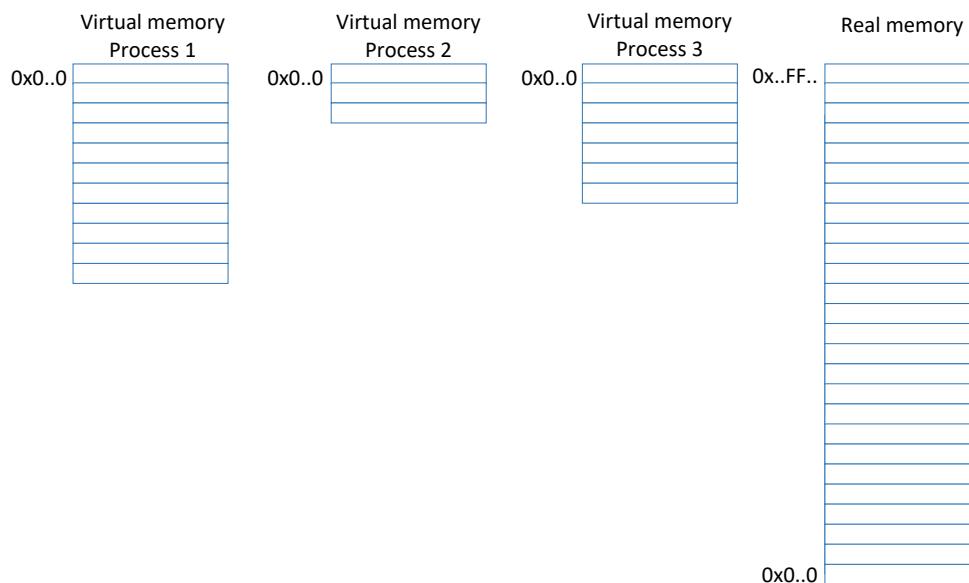
Advantages

- No (little) real memory fragmentation
- No upper memory limit required at compile time
- Virtual address space > real address space



Towards virtual addresses

Idea 2: Introduce pages and frames



Facts

- The memory is divided into small pieces of equal size called pages and frames
- Page: virtual address space
- Frame: real address space

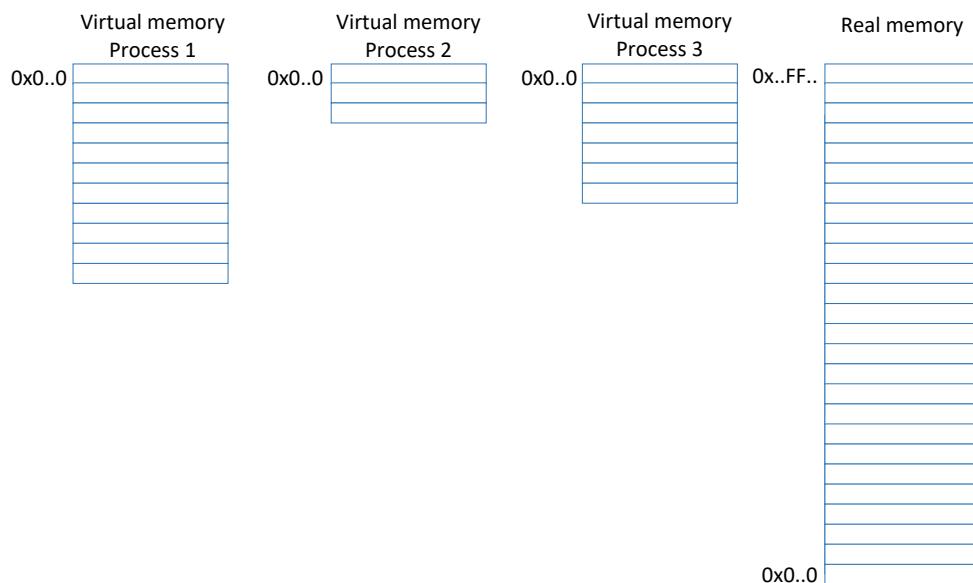
Advantages

- No (little) real memory fragmentation
- No upper memory limit required at compile time
- Virtual address space > real address space



Towards virtual addresses

Idea 2: Introduce pages and frames



Facts

- The memory is divided into small pieces of equal size called pages and frames
- Page: virtual address space
- Frame: real address space

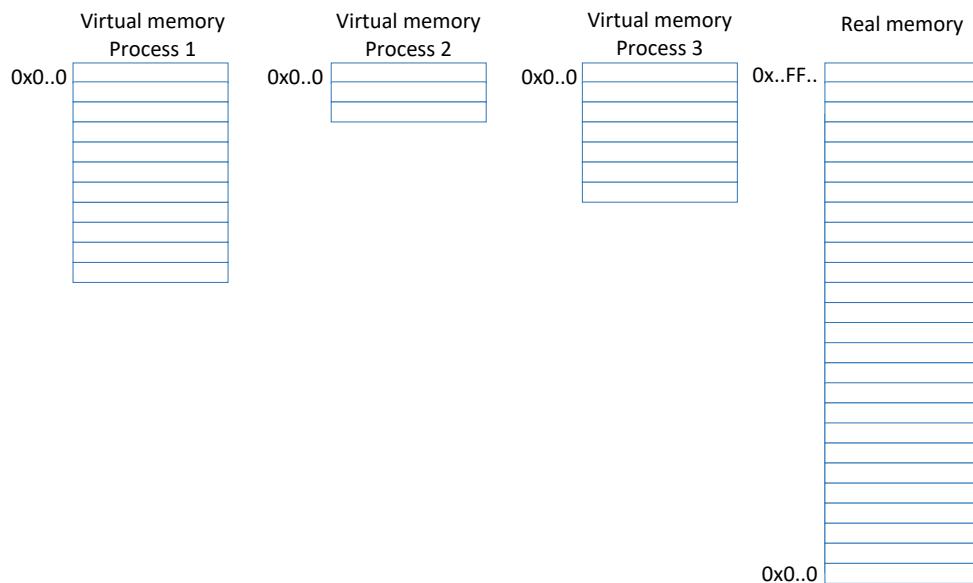
Advantages

- No (little) real memory fragmentation
- No upper memory limit required at compile time
- Virtual address space > real address space



Towards virtual addresses

Idea 2: Introduce pages and frames



Facts

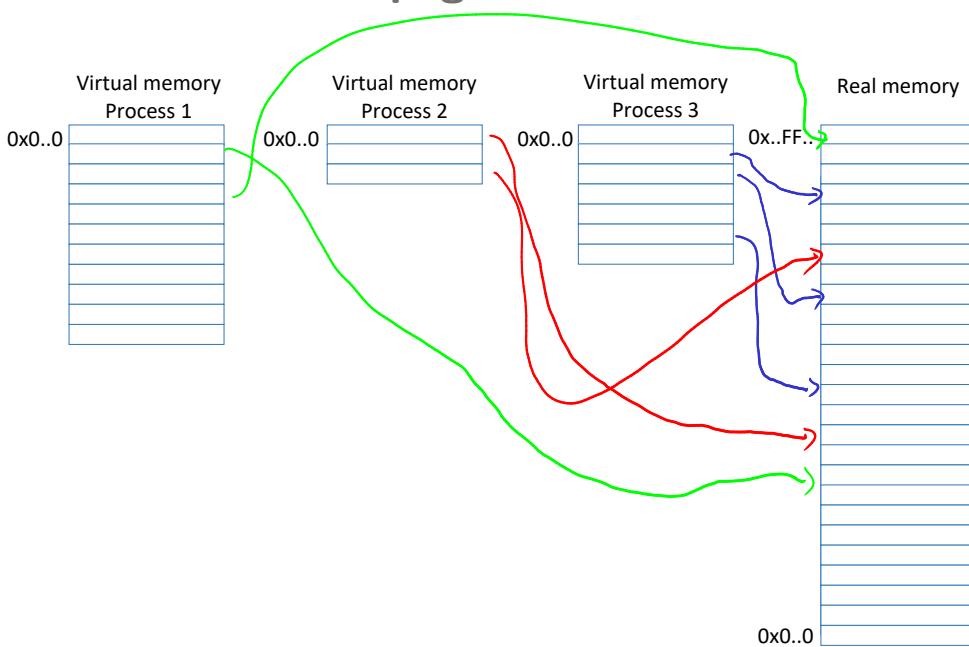
- The memory is divided into small pieces of equal size called pages and frames
- Page: virtual address space
- Frame: real address space

Advantages

- No (little) real memory fragmentation
- No upper memory limit required at compile time
- Virtual address space > real address space

Towards virtual addresses

Idea 2: Introduce pages and frames



Facts

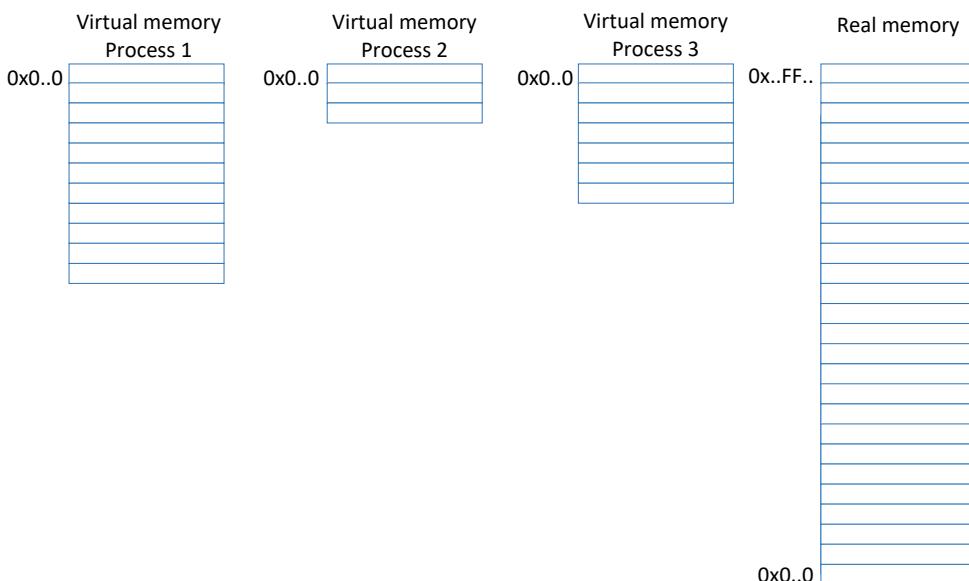
- The memory is divided into small pieces of equal size called pages and frames
- Page: virtual address space
- Frame: real address space

Advantages

- No (little) real memory fragmentation
- No upper memory limit required at compile time
- Virtual address space > real address space

Towards virtual addresses

Idea 2: Introduce pages and frames



Facts

- The memory is divided into small pieces of equal size called pages and frames
- Page: virtual address space
- Frame: real address space

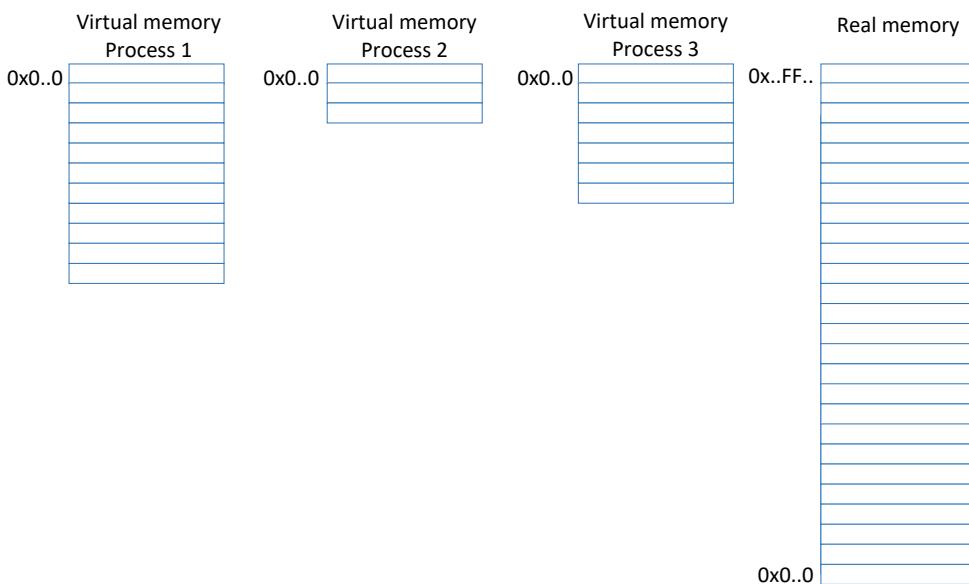
Advantages

- No (little) real memory fragmentation
- No upper memory limit required at compile time

■ Virtual address space > real address space

Towards virtual addresses

Idea 2: Introduce pages and frames



Facts

- The memory is divided into small pieces of equal size called pages and frames
- Page: virtual address space
- Frame: real address space

Advantages

- No (little) real memory fragmentation
- No upper memory limit required at compile time
- Virtual address space > real address space



Goal

Understand in detail virtual addresses and its translation to real addresses

Procedure

- Introduction of terms
- 1 level page table: with a small memory 64 KiB (virtual) and 32 KiB (real)
- 2 level page table: Intel x86/32 bit architecture
- 4 level page table: Intel x86/64 architecture (AMD/Intel)

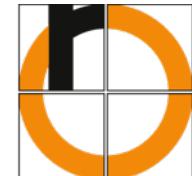


Goal

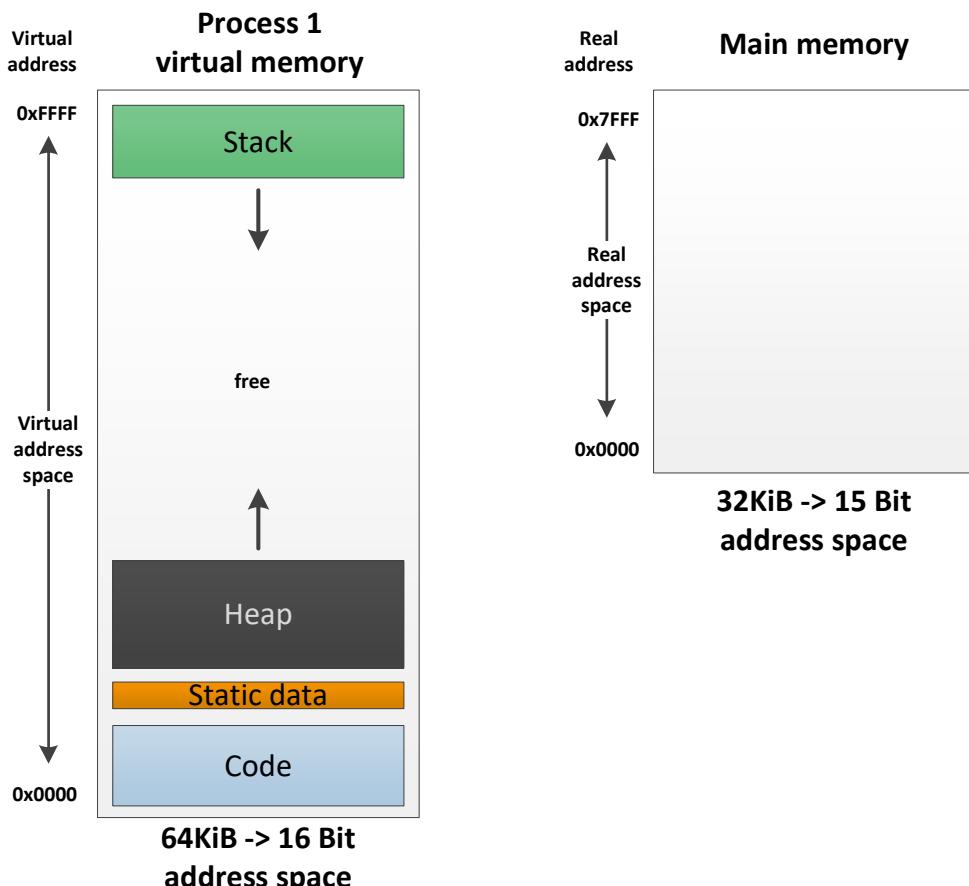
Understand in detail virtual addresses and its translation to real addresses

Procedure

- Introduction of terms
- 1 level page table: with a small memory 64 KiB (virtual) and 32 KiB (real)
- 2 level page table: Intel x86/32 bit architecture
- 4 level page table: Intel x86/64 architecture (AMD/Intel)



Virtual address space



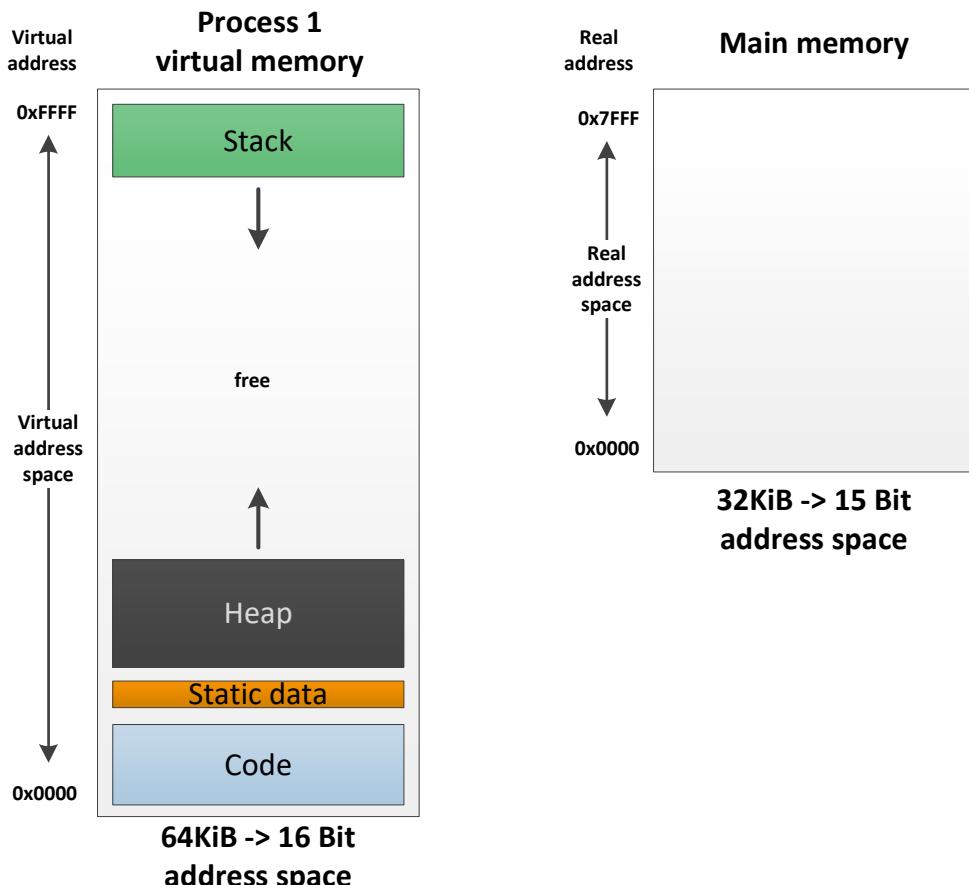
Virtual address space

- Linear address space
- Starting from 0x0..0 to 0x..FF..
- All virtual addresses => virtual address space
- Every process has its own virtual address space

Real address space

- All physically available addresses
- Size depends on available memory

Virtual address space



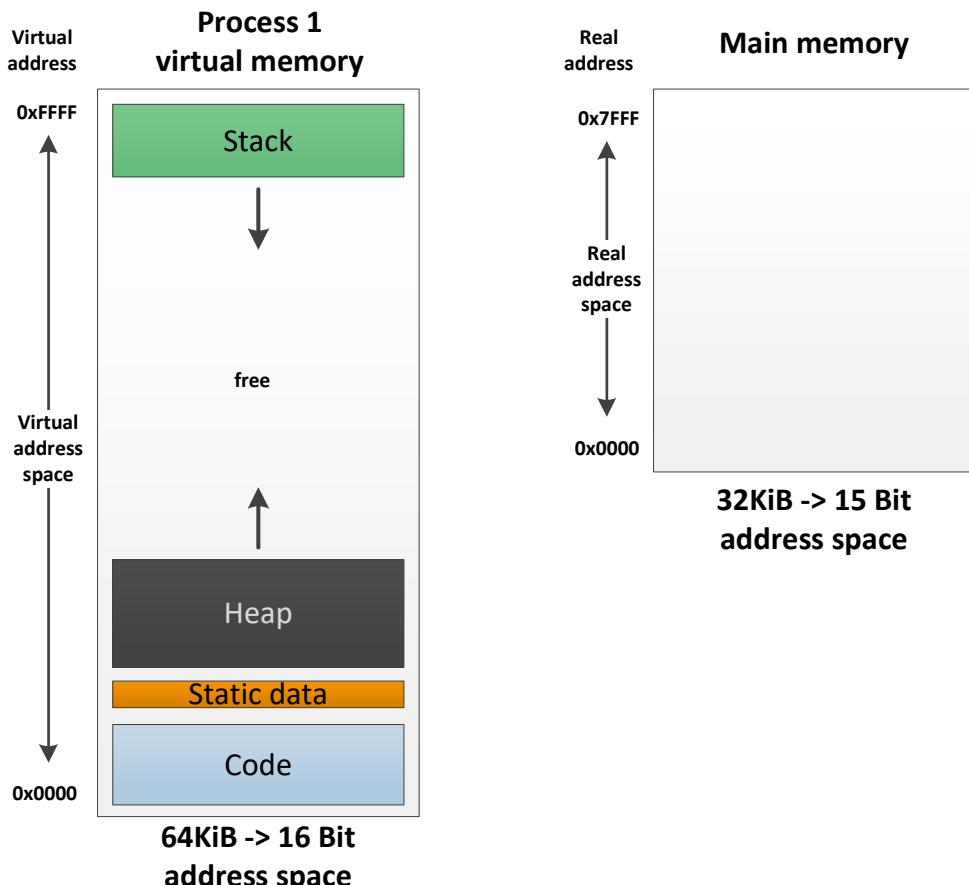
Virtual address space

- Linear address space
- Starting from 0x0..0 to 0x..FF..
- All virtual addresses => virtual address space
- Every process has its own virtual address space

Real address space

- All physically available addresses
- Size depends on available memory

Virtual address space

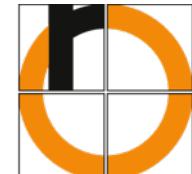


Virtual address space

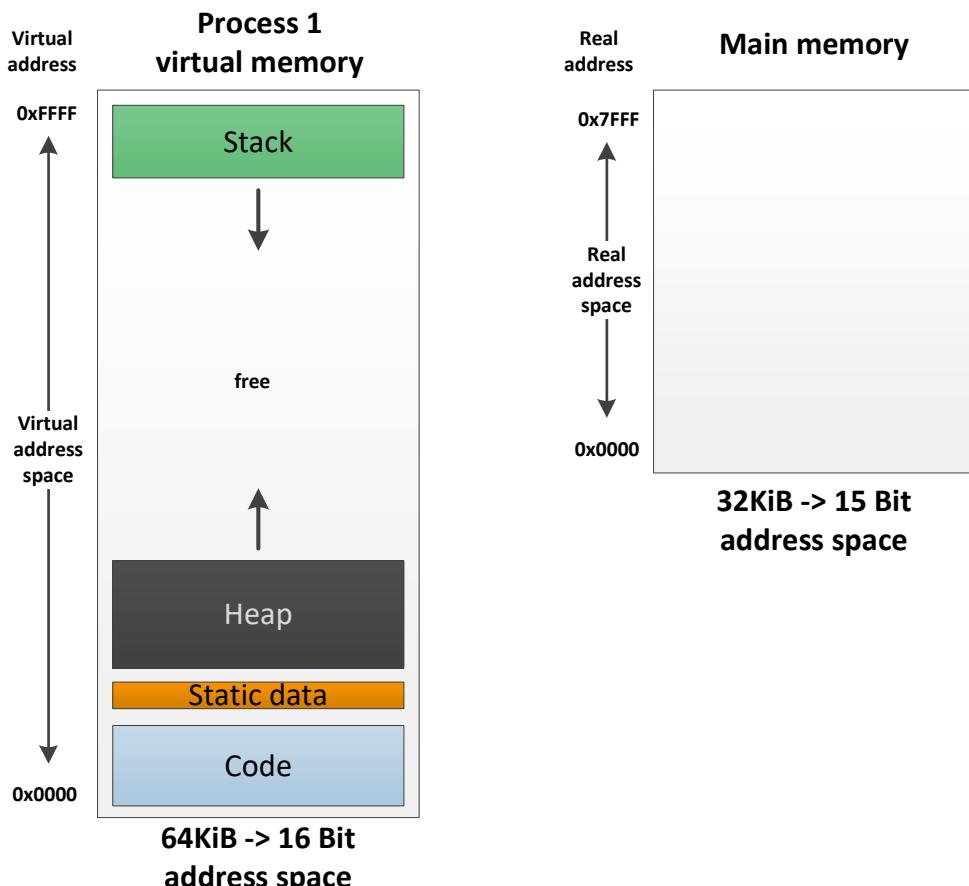
- Linear address space
- Starting from 0x0..0 to 0x..FF..
- All virtual addresses => virtual address space
- Every process has its own virtual address space

Real address space

- All physically available addresses
- Size depends on available memory



Virtual address space



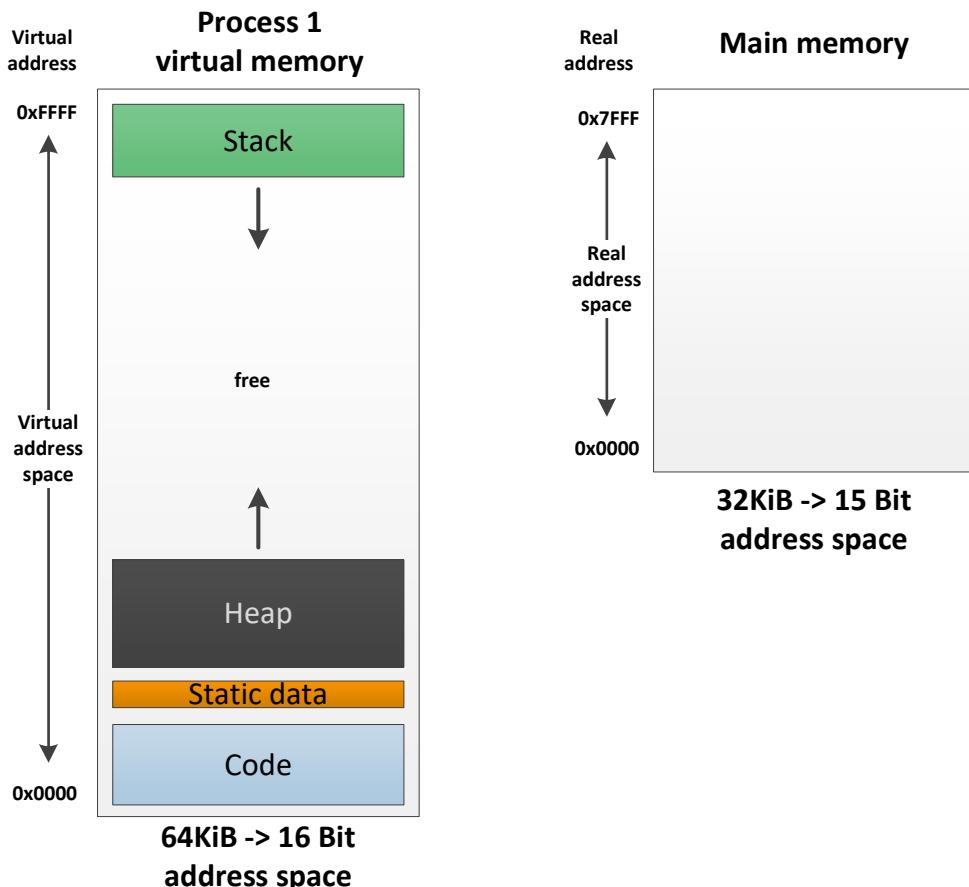
Virtual address space

- Linear address space
- Starting from 0x0..0 to 0x..FF..
- All virtual addresses => virtual address space
- Every process has its own virtual address space

Real address space

- All physically available addresses
- Size depends on available memory

Virtual address space



Virtual address space

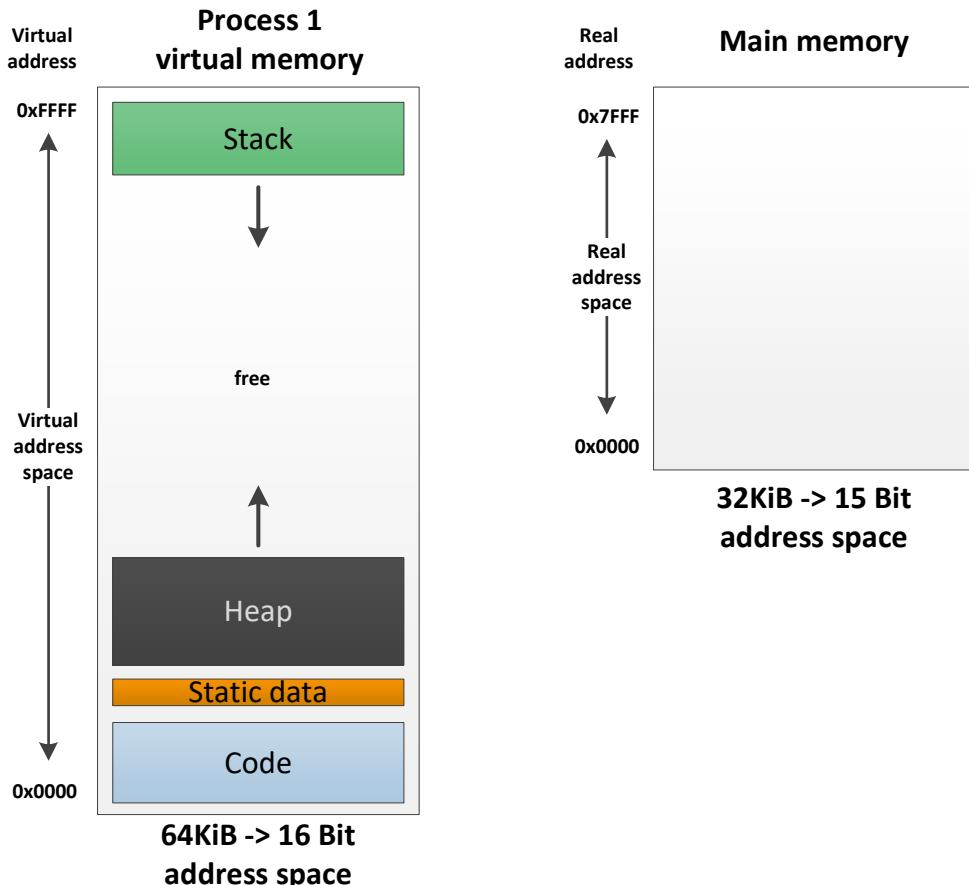
- Linear address space
- Starting from 0x0..0 to 0x..FF..
- All virtual addresses => virtual address space
- Every process has its own virtual address space

Real address space

- All physically available addresses
- Size depends on available memory



Virtual address space



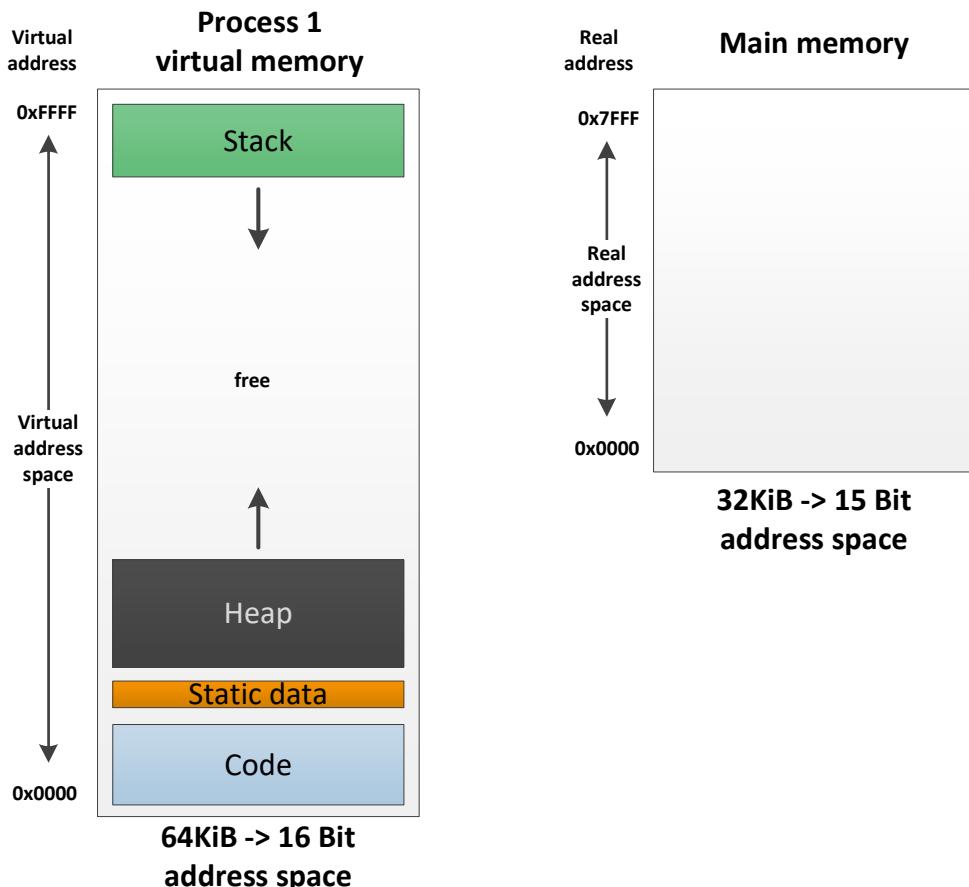
Virtual address space

- Linear address space
- Starting from 0x0..0 to 0x..FF..
- All virtual addresses => virtual address space
- Every process has its own virtual address space

Real address space

- All physically available addresses
- Size depends on available memory

Virtual address space

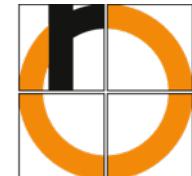


Virtual address space

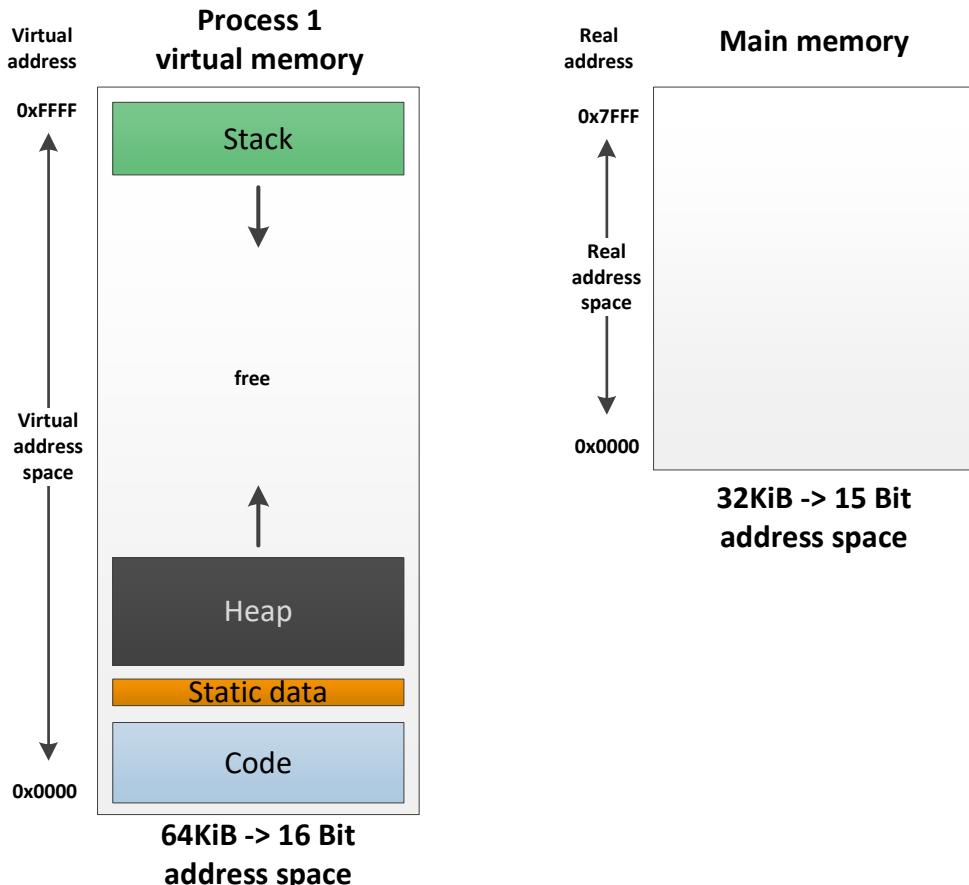
- Linear address space
- Starting from 0x0..0 to 0x..FF..
- All virtual addresses => virtual address space
- Every process has its own virtual address space

Real address space

- All physically available addresses
- Size depends on available memory



Virtual address space

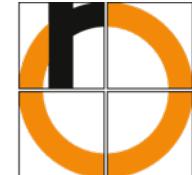


Virtual address space

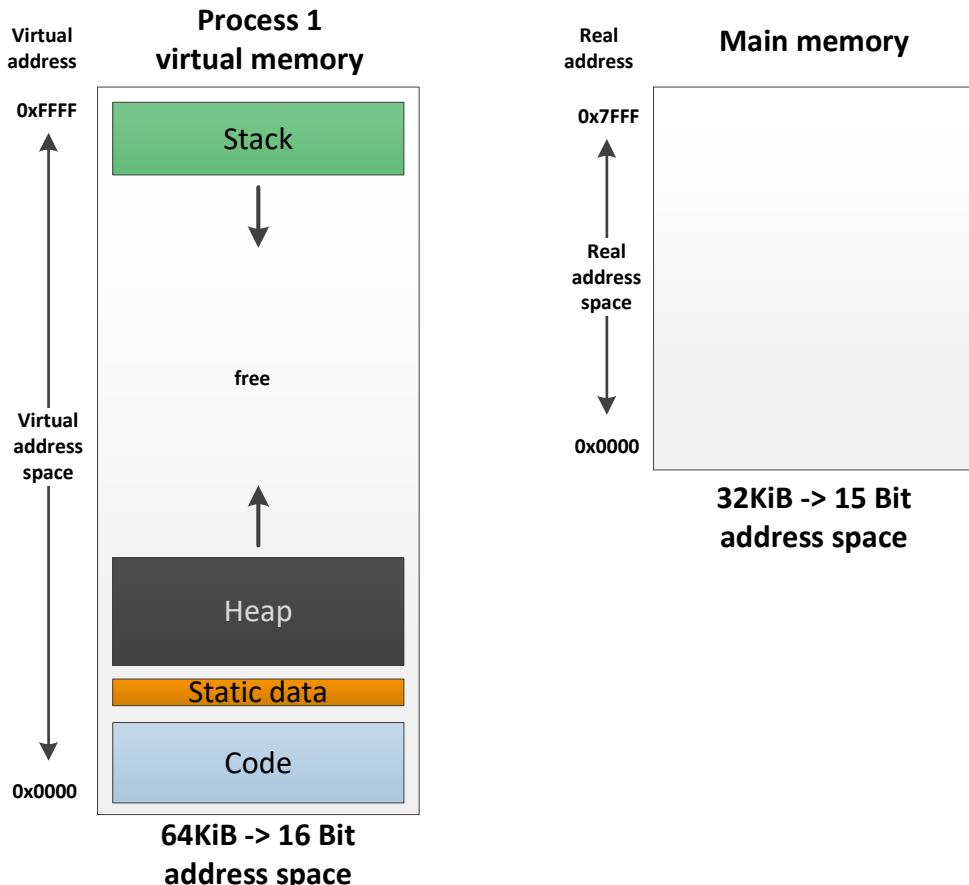
- Linear address space
- Starting from 0x0..0 to 0x..FF..
- All virtual addresses => virtual address space
- Every process has its own virtual address space

Real address space

- All physically available addresses
- Size depends on available memory



Virtual address space



Virtual address space

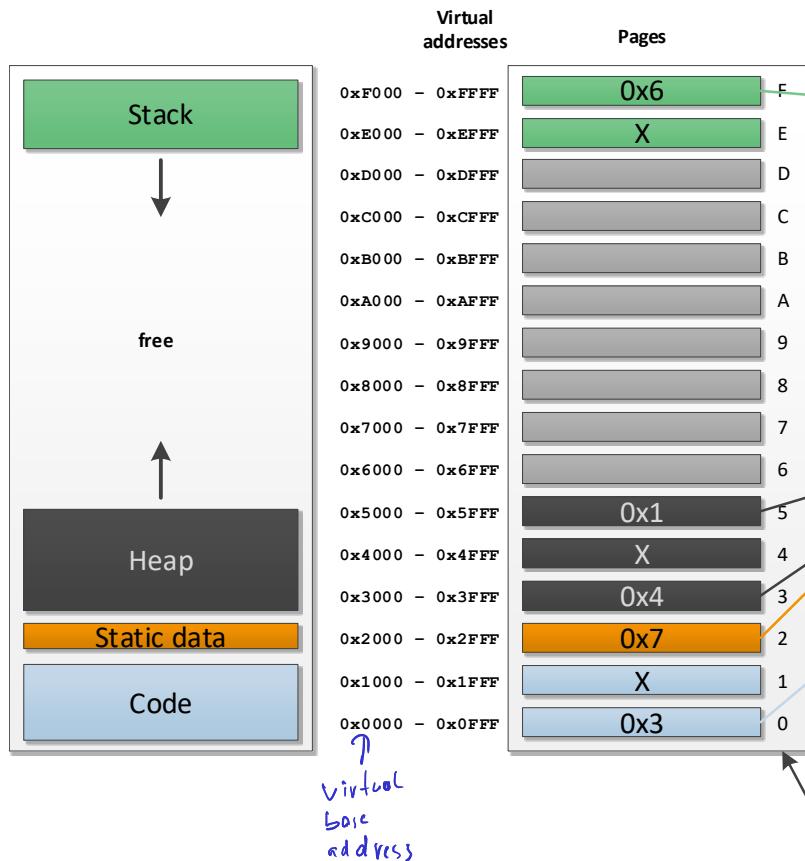
- Linear address space
- Starting from 0x0..0 to 0x..FF..
- All virtual addresses => virtual address space
- Every process has its own virtual address space

Real address space

- All physically available addresses
- Size depends on available memory

Pages and frames

Virtual memory



Main memory



Pages and frames have same size (here 4KiB)

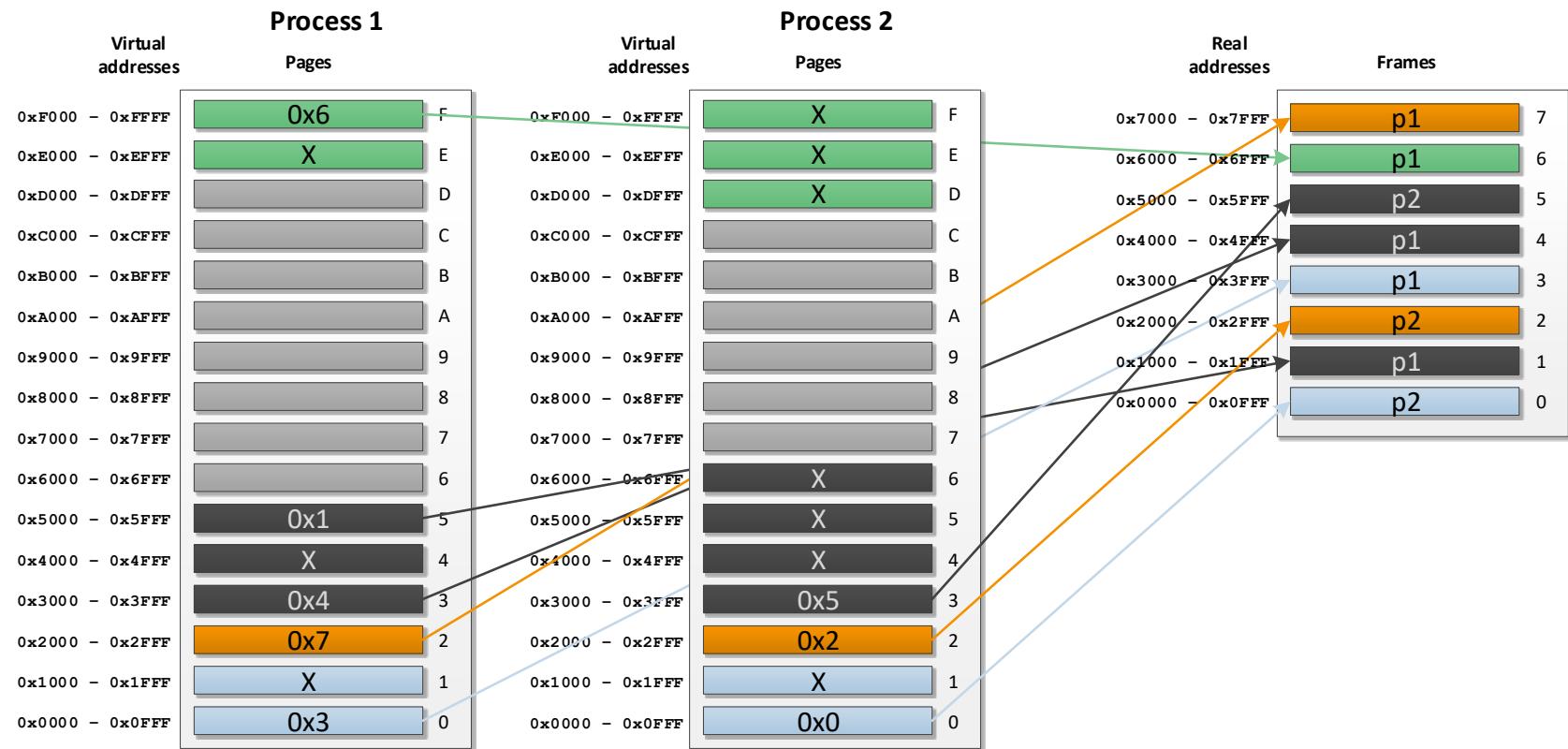
4096 byte

$$\log_2(4 \text{ KiB}) = \log_2(2^2 \cdot 2^{10} = 2^{12}) = 12$$

Multiprogramming

Virtual memory

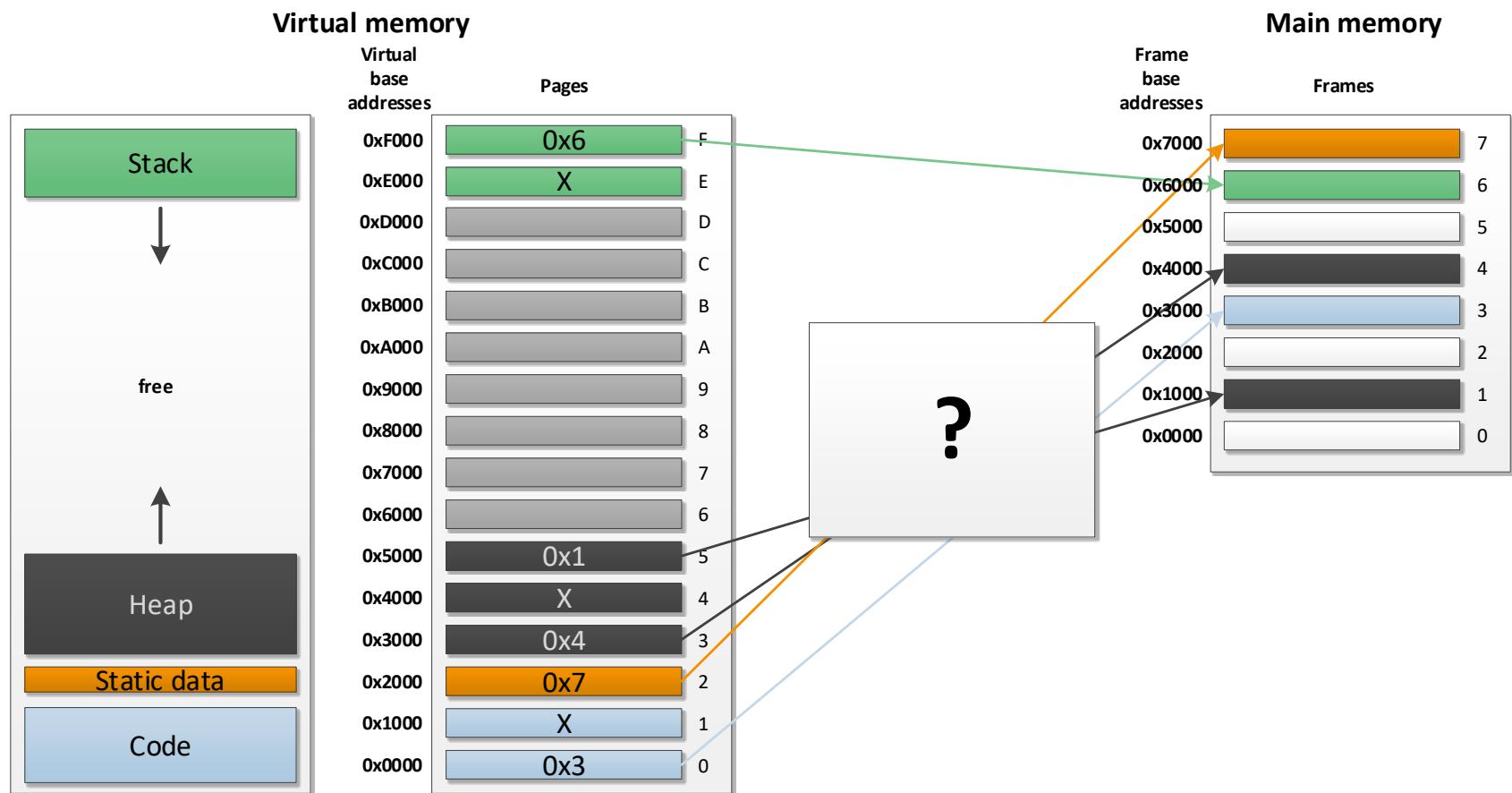
Main memory



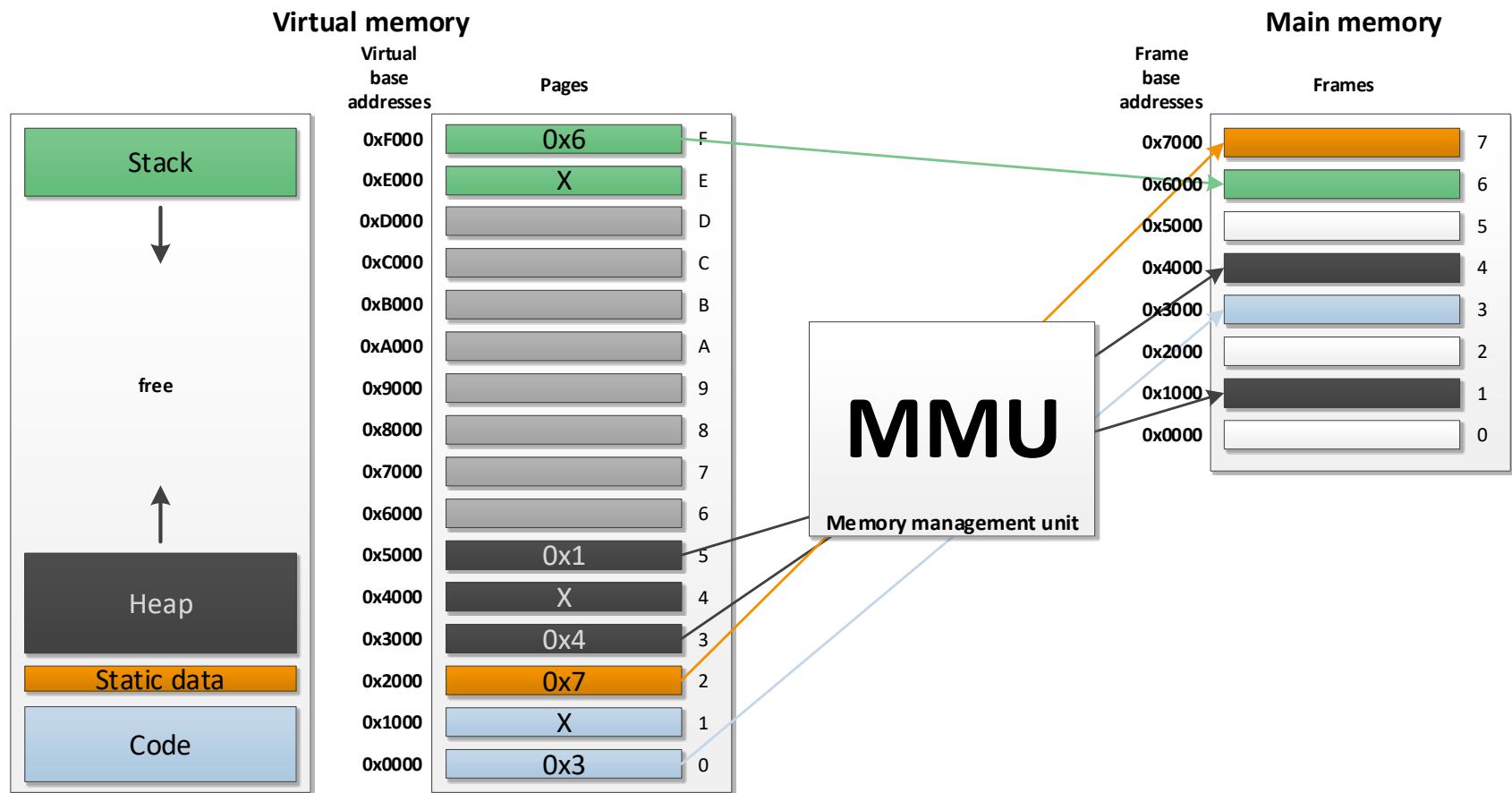


Let's start with a simple
1 level page table.

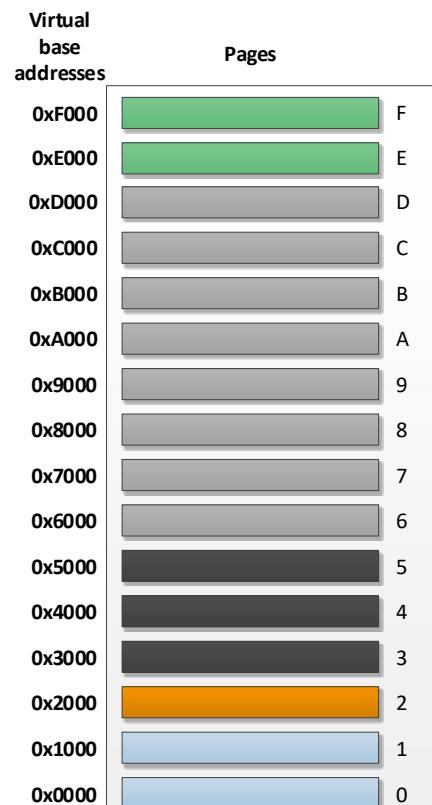
Virtual address translation



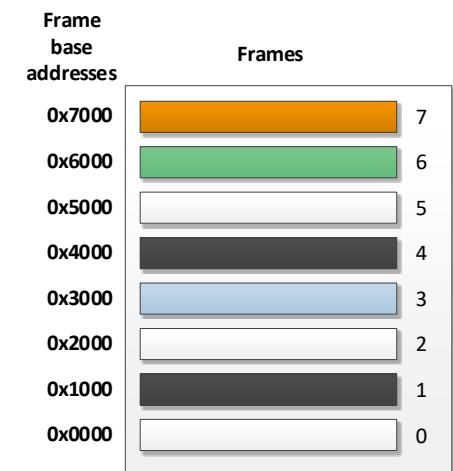
Virtual address translation



Page table

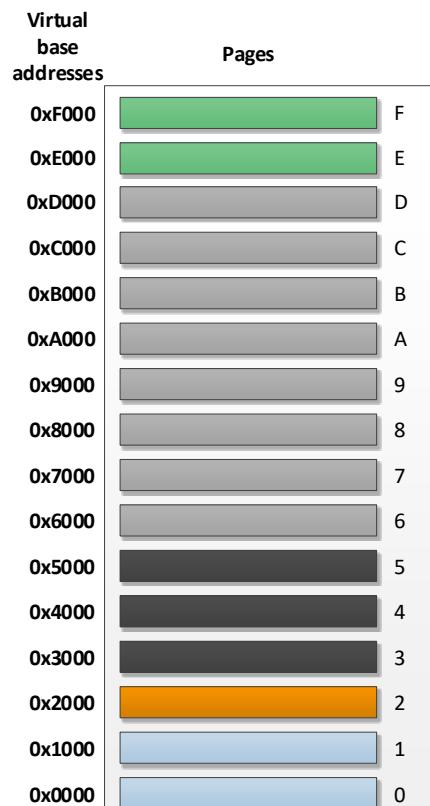
Virtual memory

Each process has its own virtual memory

Main memory

Page table

Virtual memory

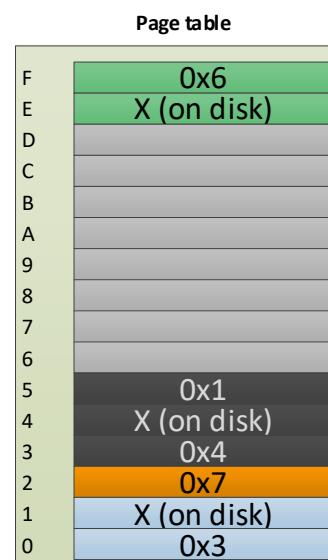
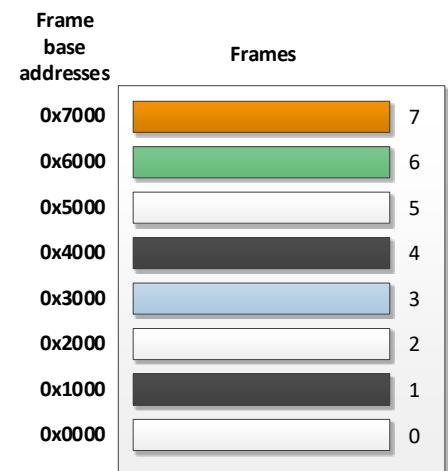


Each process has its own virtual memory

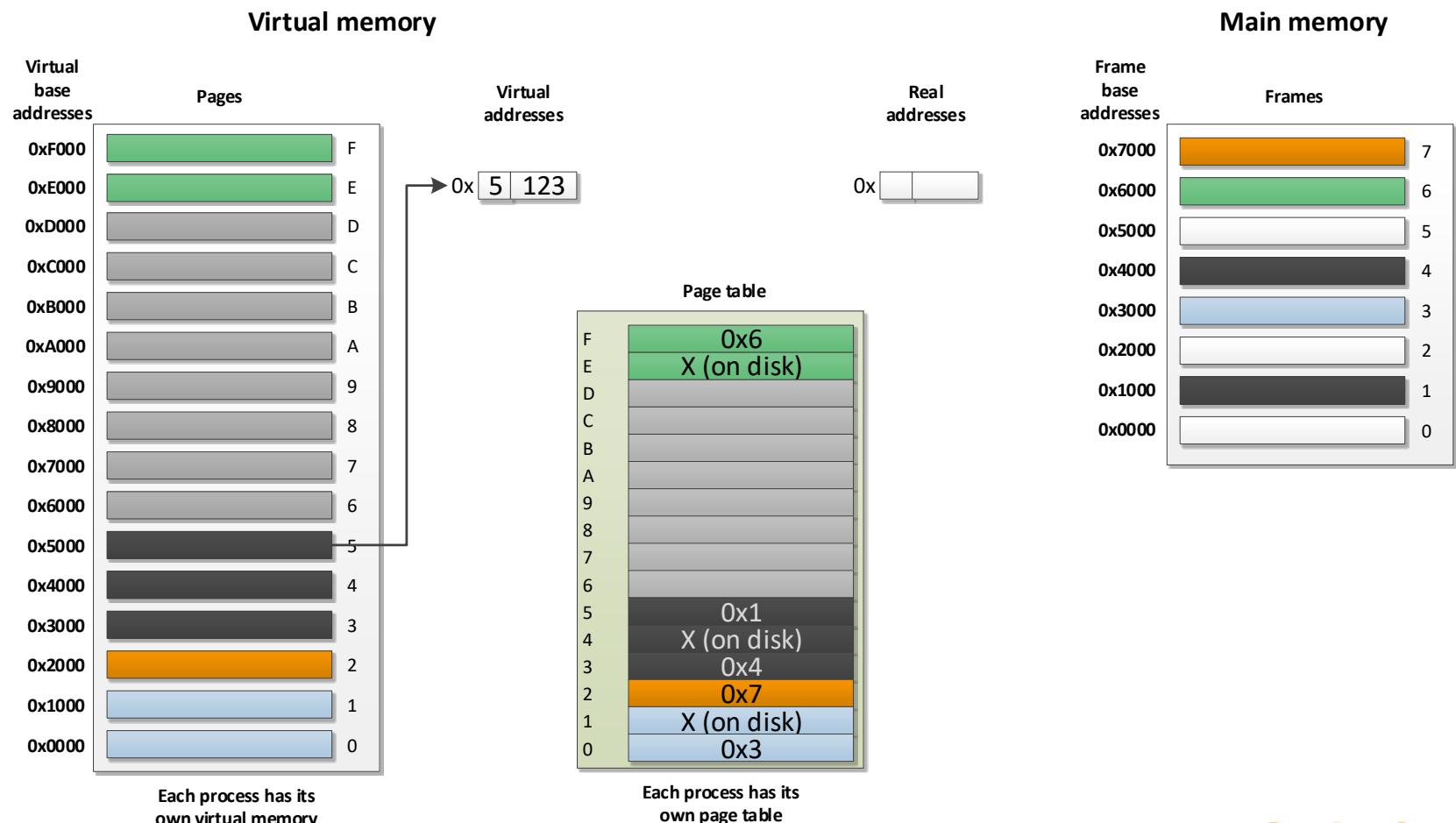
Each process has its own page table

CA 9 – MMU

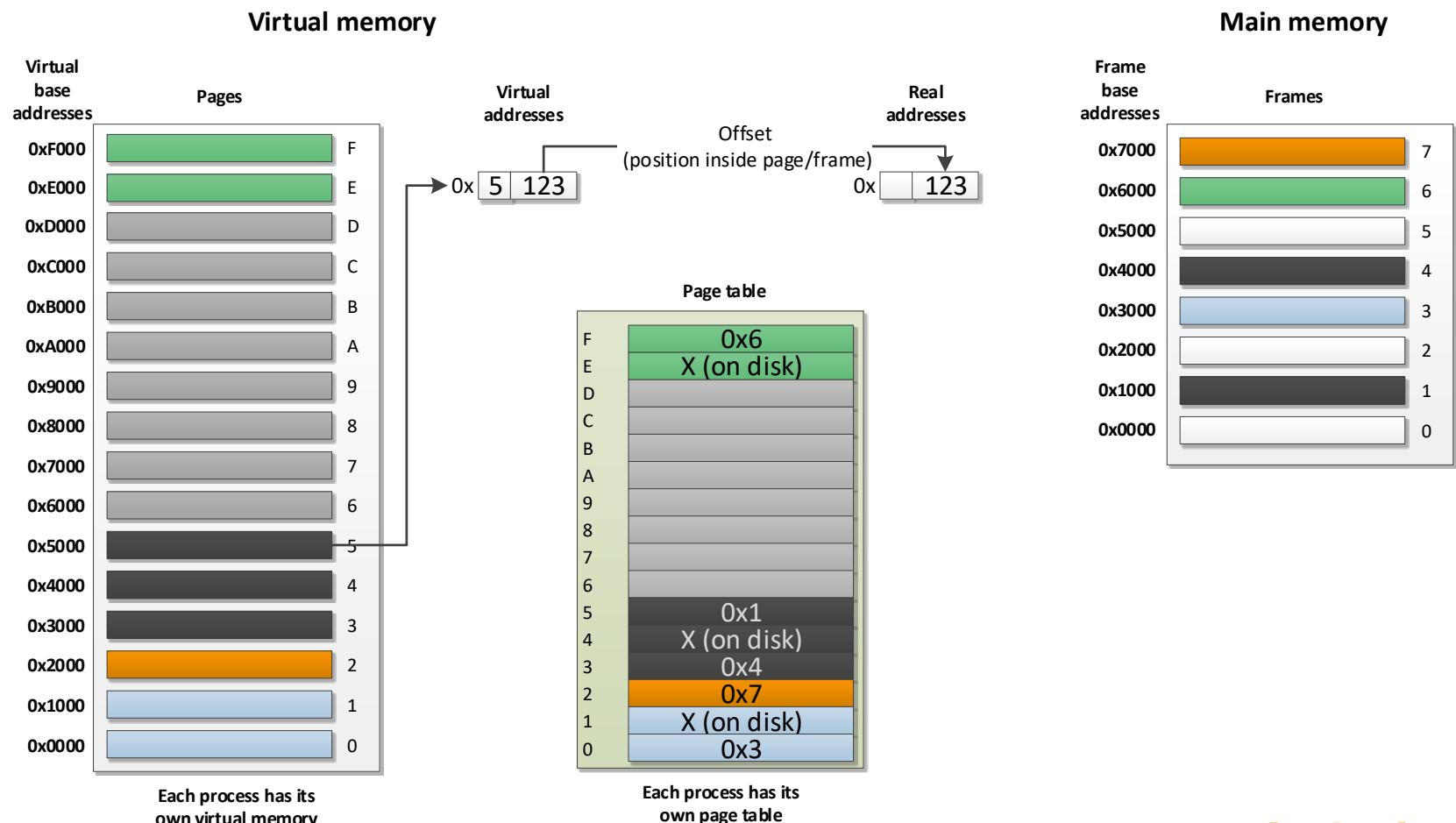
Main memory



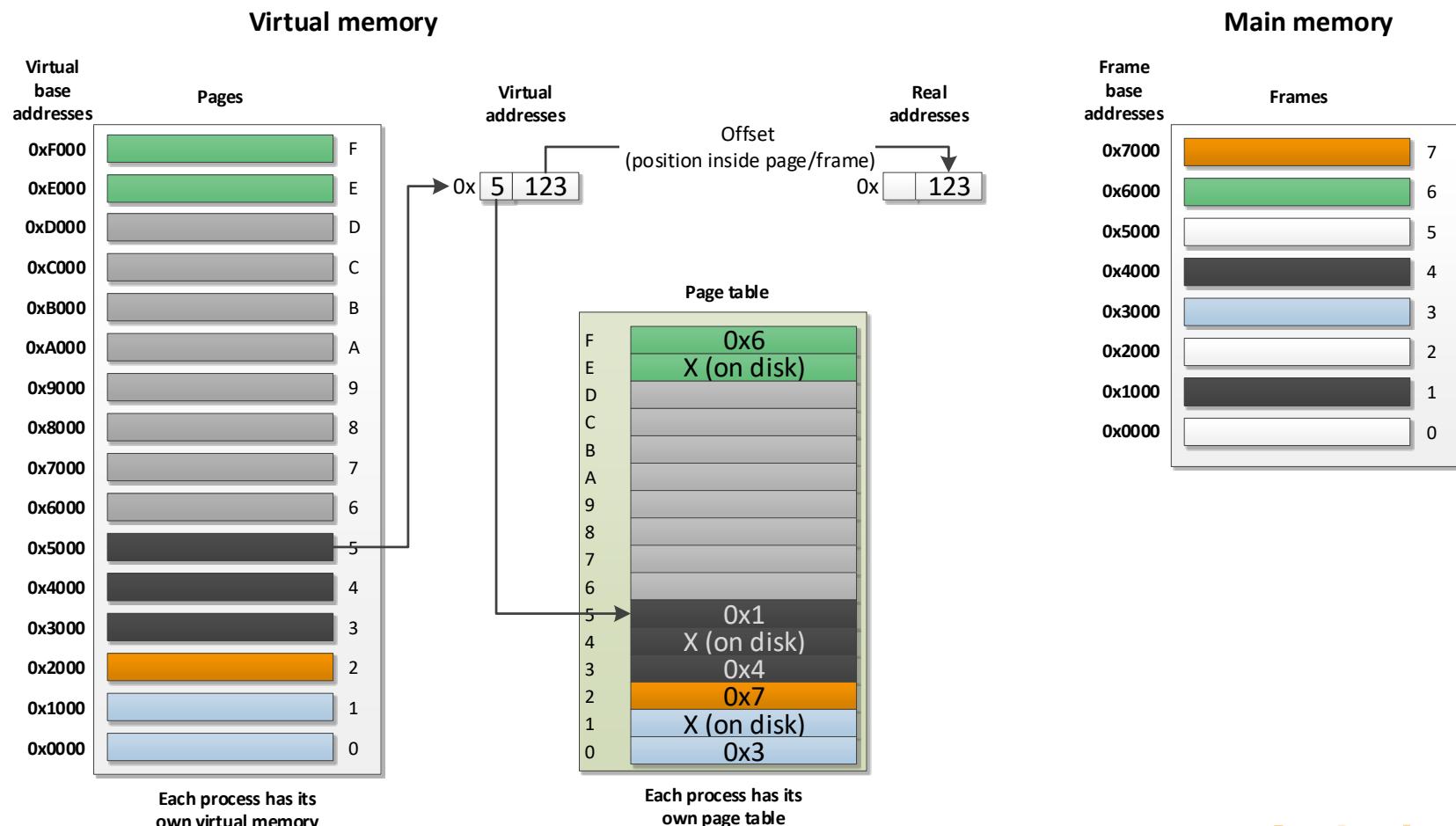
Page table



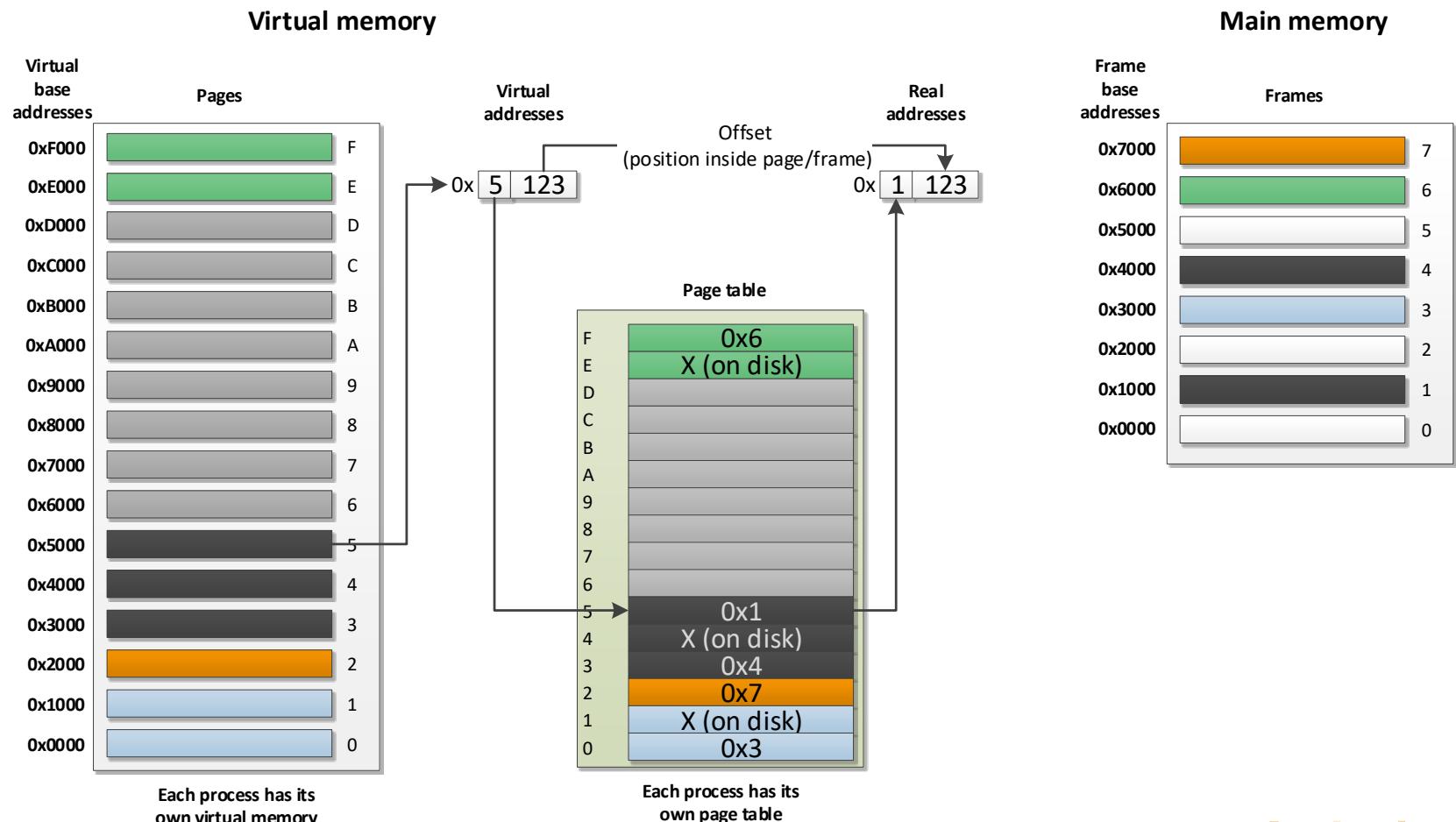
Page table



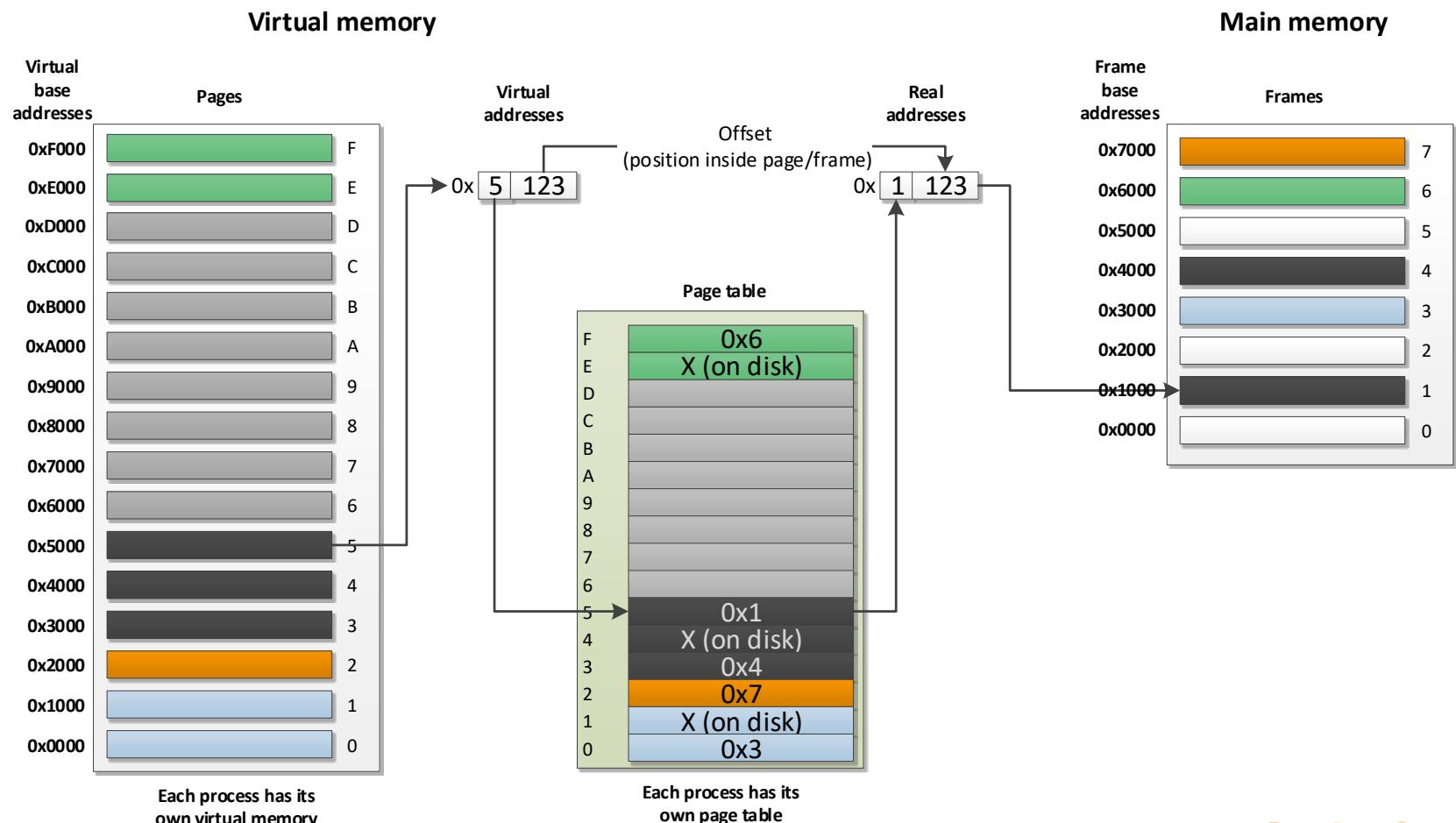
Page table



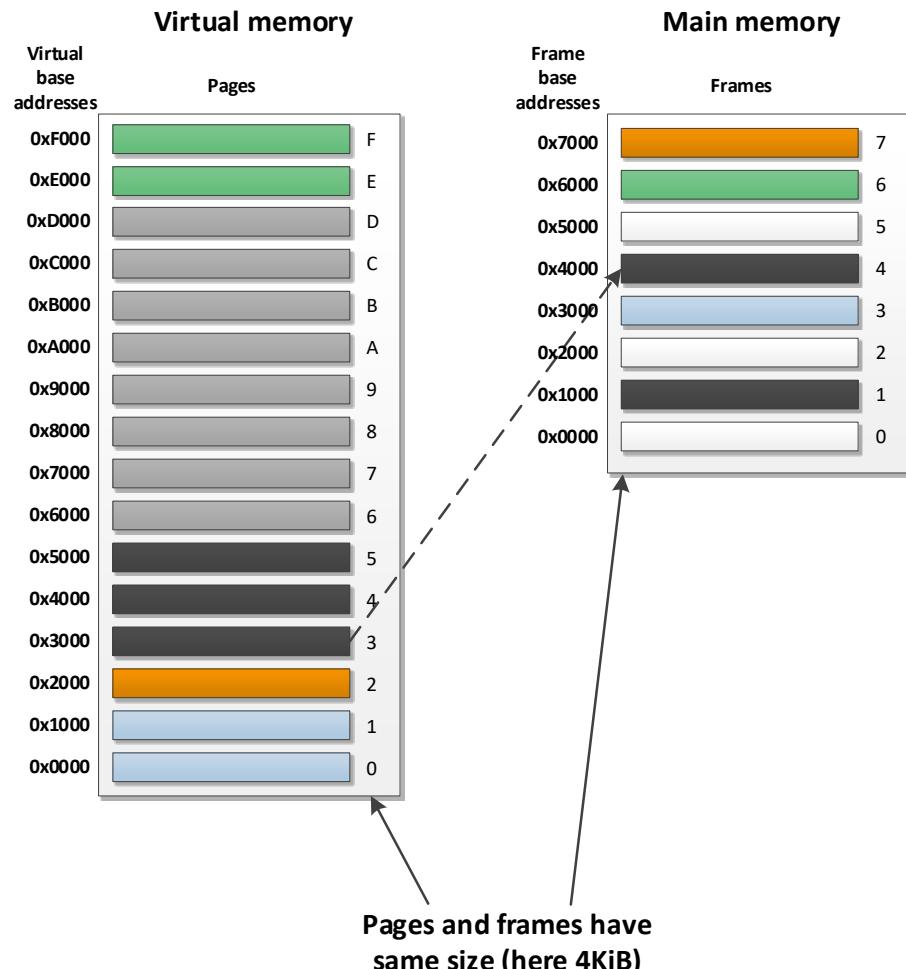
Page table



Page table



Virtual address translation: example

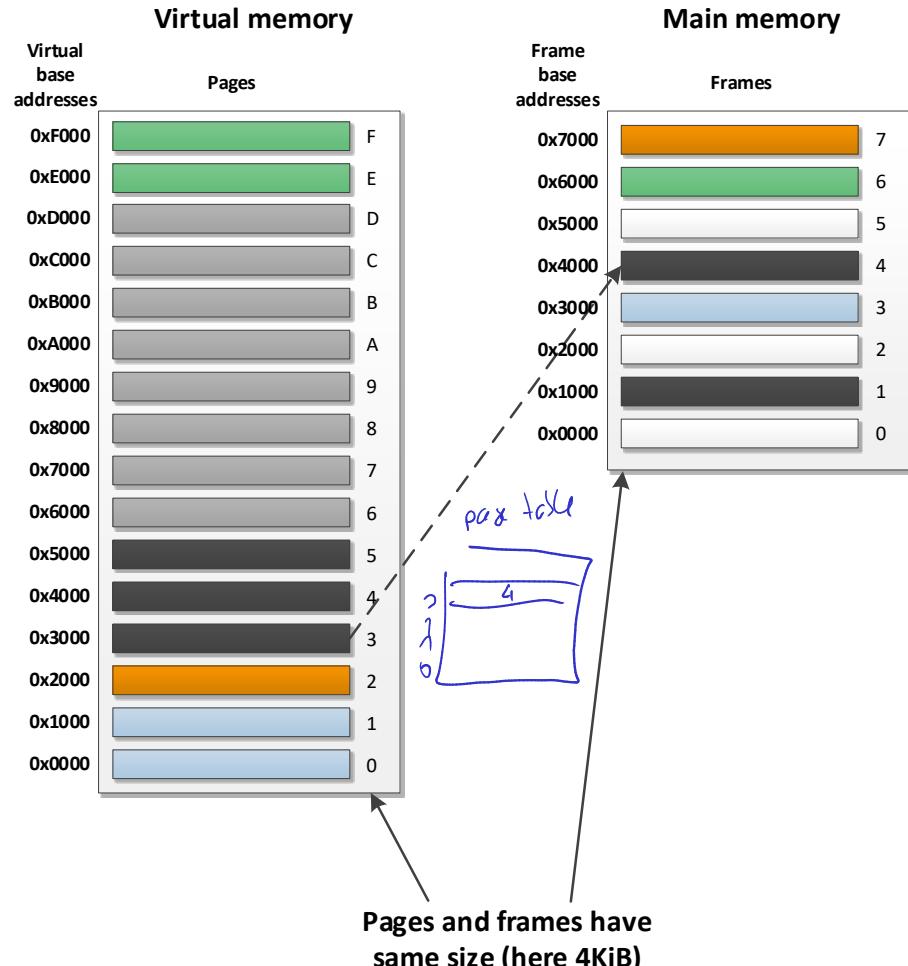

Given:

- Virtual address space: 64 KiB
- Real address space: 32 KiB
- Virtual address: 0x3D05

Questions:

- 1 Address width (virtual and real)?
- 2 Offset: number of bits?
- 3 Page/frame numbers: number of bits?
- 4 Number of pages and frames?
- 5 Real address?

Virtual address translation: example



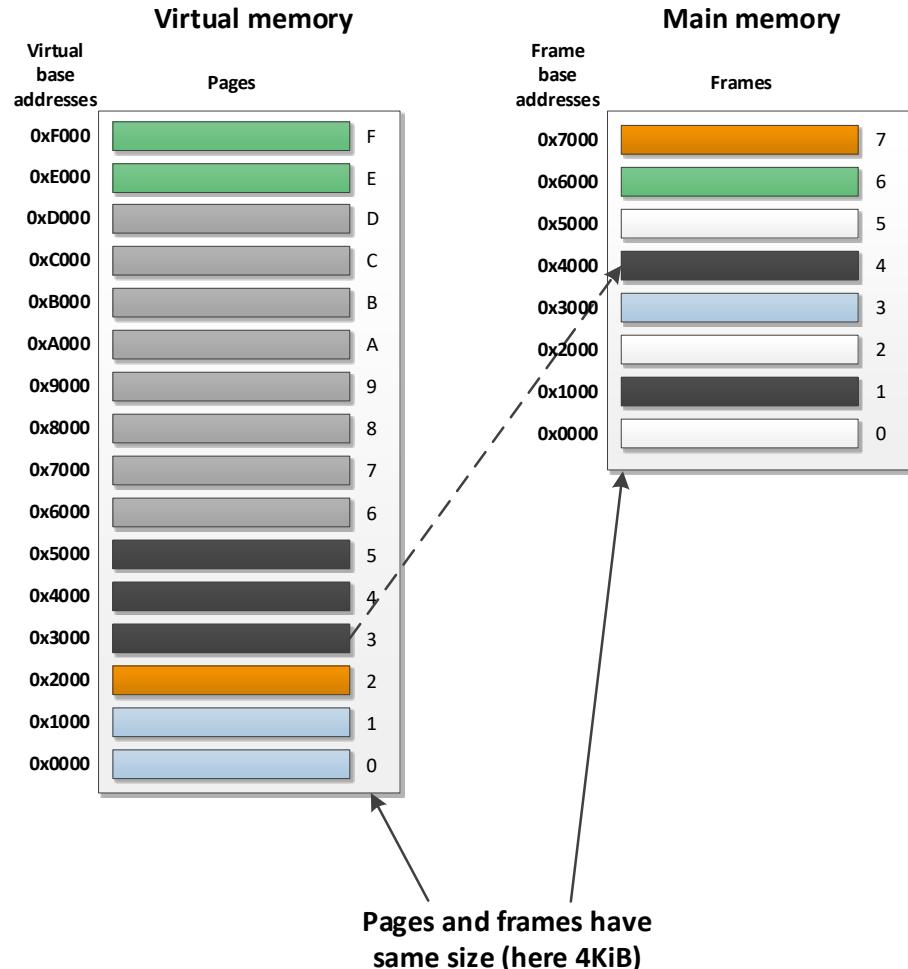
Given:

- Virtual address space: 64 KiB
- Real address space: 32 KiB
- Virtual address: 0x3D05

Questions:

- 1 Address width (virtual and real)?
- 2 Offset: number of bits?
- 3 Page/frame numbers: number of bits?
- 4 Number of pages and frames?
- 5 Real address?

Virtual address translation: example



Given:

- Virtual address space: 64 KiB
- Real address space: 32 KiB
- Virtual address: 0x3D05

Questions:

- 1 Address width (virtual and real)?
- 2 Offset: number of bits?
- 3 Page/frame numbers: number of bits?
- 4 Number of pages and frames?
- 5 Real address?

Results:

- 1 virtual: 16 bit/real: 15 bit
- 2 offset: 12 bit
- 3 pages: 4 bit/frames: 3 bit
- 4 pages: $2^4 = 16$ /frames: $2^3 = 8$
- 5 real address: 0x4D05



1 level page table

Any problems with a 1 level page table?

- How many number of page table entries are required on an 32 bit system?
- How much space would the page table use?

Possible solutions

- Increase page/frame size
 - i.e. 1 MiB, 2 MiB, ...
 - => less pages, less entries in page table, smaller page table
- Multilevel page table
 - 32 bit: 2 level page table with 4 KiB page size
 - 64 bit: 4 level page table with 4 KiB page size
- Swap page tables to disk



1 level page table

$$2^{32} / 2^{12} = 1 \text{ Mi}$$

Any problems with a 1 level page table?

- How many number of page table entries are required on an 32 bit system?
- How much space would the page table use?

Possible solutions

- Increase page/frame size
 - i.e. 1 MiB, 2 MiB, ...
 - => less pages, less entries in page table, smaller page table
- Multilevel page table
 - 32 bit: 2 level page table with 4 KiB page size
 - 64 bit: 4 level page table with 4 KiB page size
- Swap page tables to disk



1 level page table

Any problems with a 1 level page table?

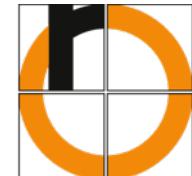
- How many number of page table entries are required on an 32 bit system?
- How much space would the page table use?

$$32 \text{ bit} = 4 \text{ byte}$$

$$1 \text{ Mi} - 4 \text{ byte} = 4 \text{ MiB}$$

Possible solutions

- Increase page/frame size
 - i.e. 1 MiB, 2 MiB, ...
 - => less pages, less entries in page table, smaller page table
- Multilevel page table
 - 32 bit: 2 level page table with 4 KiB page size
 - 64 bit: 4 level page table with 4 KiB page size
- Swap page tables to disk



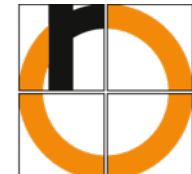
1 level page table

Any problems with a 1 level page table?

- How many number of page table entries are required on an 32 bit system?
- How much space would the page table use?

Possible solutions

- Increase page/frame size
 - i.e. 1 MiB, 2 MiB, ...
 - => less pages, less entries in page table, smaller page table
- Multilevel page table
 - 32 bit: 2 level page table with 4 KiB page size
 - 64 bit: 4 level page table with 4 KiB page size
- Swap page tables to disk



1 level page table

Any problems with a 1 level page table?

- How many number of page table entries are required on an 32 bit system?
- How much space would the page table use?

Possible solutions

- Increase page/frame size
 - i.e. 1 MiB, 2 MiB, ...
 - => less pages, less entries in page table, smaller page table
- Multilevel page table
 - 32 bit: 2 level page table with 4 KiB page size
 - 64 bit: 4 level page table with 4 KiB page size
- Swap page tables to disk

1 level page table

Any problems with a 1 level page table?

- How many number of page table entries are required on an 32 bit system?
- How much space would the page table use?

Possible solutions

- Increase page/frame size
 - i.e. 1 MiB, 2 MiB, ...
 - => less pages, less entries in page table, smaller page table
- Multilevel page table
 - 32 bit: 2 level page table with 4 KiB page size
 - 64 bit: 4 level page table with 4 KiB page size
- Swap page tables to disk

1 level page table

Any problems with a 1 level page table?

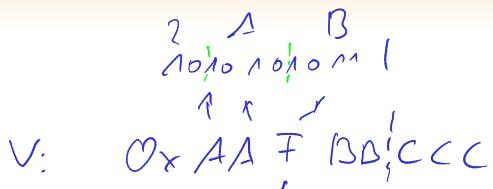
- How many number of page table entries are required on an 32 bit system?
- How much space would the page table use?

Possible solutions

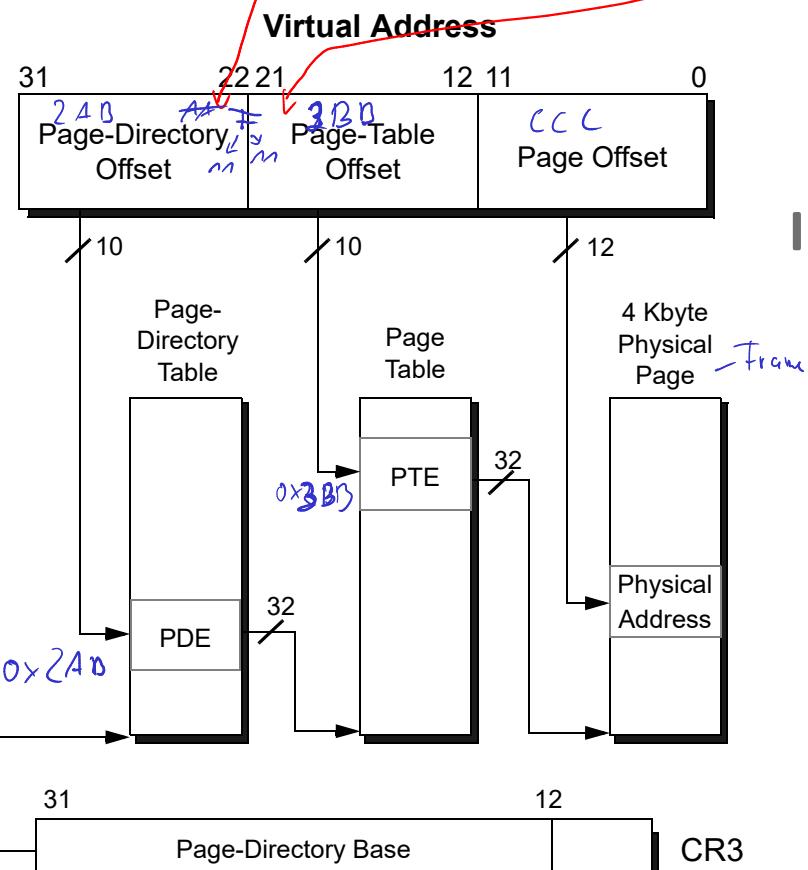
- Increase page/frame size
 - i.e. 1 MiB, 2 MiB, ...
 - => less pages, less entries in page table, smaller page table
- Multilevel page table
 - 32 bit: 2 level page table with 4 KiB page size
 - 64 bit: 4 level page table with 4 KiB page size
- Swap page tables to disk



Let's proceed with a
2 level page table.



2 level page table



Intel x86/32 bit

- Usually uses 2 level page tables
- Offset: 12 bit
- $2^{12} = 4 \text{ KiB}$ page size
- 1 page directory per process
- $2^{10} = 1024$ page tables per process

[source: AMD64 Architecture Programmer's Manual Volume 2: System Programming]

2 level page table entries

PDE - page directory entry

- Page-Table Base Address (20 bit)
= address of page table

PTE - page table entry

- Physical-Page Base Address (20 bit)
= address of frame

Fields

- P = present (loaded into memory)
- R/W = read/write
- U/S = user/supervisor
- PWT = page level writethrough
- PCD = page level cache disable
- A = accessed
- D = dirty
- AVL = available to software (for OS)

31

	12	11	9	8	7	6	5	4	3	2	1	0
Page-Table Base Address	AVL		I G N	0	I G N	A	P C D	P W T	U S	R W	P	

Figure 5-5. 4-Kbyte PDE—Non-PAE Paging Legacy-Mode

31

	12	11	9	8	7	6	5	4	3	2	1	0
Physical-Page Base Address	AVL		G A T	P A D	D A	A	P C D	P W T	U S	R W	P	

Figure 5-6. 4-Kbyte PTE—Non-PAE Paging Legacy-Mode

[source: AMD64 Architecture Programmer's Manual Volume 2: System Programming]

2 level page table entries

PDE - page directory entry

- Page-Table Base Address (20 bit)
= address of page table

PTE - page table entry

- Physical-Page Base Address (20 bit)
= address of frame

Fields

- P = present (loaded into memory)
- R/W = read/write
- U/S = user/supervisor
- PWT = page level writethrough
- PCD = page level cache disable
- A = accessed
- D = dirty
- AVL = available to software (for OS)

31

	12	11	9	8	7	6	5	4	3	2	1	0
Page-Table Base Address	AVL	G N	0	I G N	A	P C D	P W T	U S	R W	/	/	P

Figure 5-5. 4-Kbyte PDE—Non-PAE Paging Legacy-Mode

31

	12	11	9	8	7	6	5	4	3	2	1	0
Physical-Page Base Address	AVL	G A T	P A D	D A	P C D	P W T	U S	R W	/	/	P	

Figure 5-6. 4-Kbyte PTE—Non-PAE Paging Legacy-Mode

[source: AMD64 Architecture Programmer's Manual Volume 2: System Programming]

2 level page table entries

PDE - page directory entry

- Page-Table Base Address (20 bit)
= address of page table

PTE - page table entry

- Physical-Page Base Address (20 bit)
= address of frame

31		12 11	9	8	7	6	5	4	3	2	1	0
	Page-Table Base Address	AVL	I G N	0	I G N	A	P C D	P W T	U /	R /	S W	P

Figure 5-5. 4-Kbyte PDE—Non-PAE Paging Legacy-Mode

31		12 11	9	8	7	6	5	4	3	2	1	0
	Physical-Page Base Address	AVL	G A T	P A D	D A	A	P C D	P W T	U /	R /	S W	P

Figure 5-6. 4-Kbyte PTE—Non-PAE Paging Legacy-Mode

[source: AMD64 Architecture Programmer's Manual Volume 2: System Programming]

Fields

- P = present (loaded into memory)
- R/W = read/write
- U/S = user/supervisor
- PWT = page level writethrough
- PCD = page level cache disable
- A = accessed
- D = dirty
- AVL = available to software (for OS)

2 level page table entries

PDE - page directory entry

- Page-Table Base Address (20 bit)
= address of page table

PTE - page table entry

- Physical-Page Base Address (20 bit)
= address of frame

Fields

- P = present (loaded into memory)
- R/W = read/write
- U/S = user/supervisor
- PWT = page level writethrough
- PCD = page level cache disable
- A = accessed
- D = dirty
- AVL = available to software (for OS)

31



Figure 5-5. 4-Kbyte PDE—Non-PAE Paging Legacy-Mode

31



Figure 5-6. 4-Kbyte PTE—Non-PAE Paging Legacy-Mode

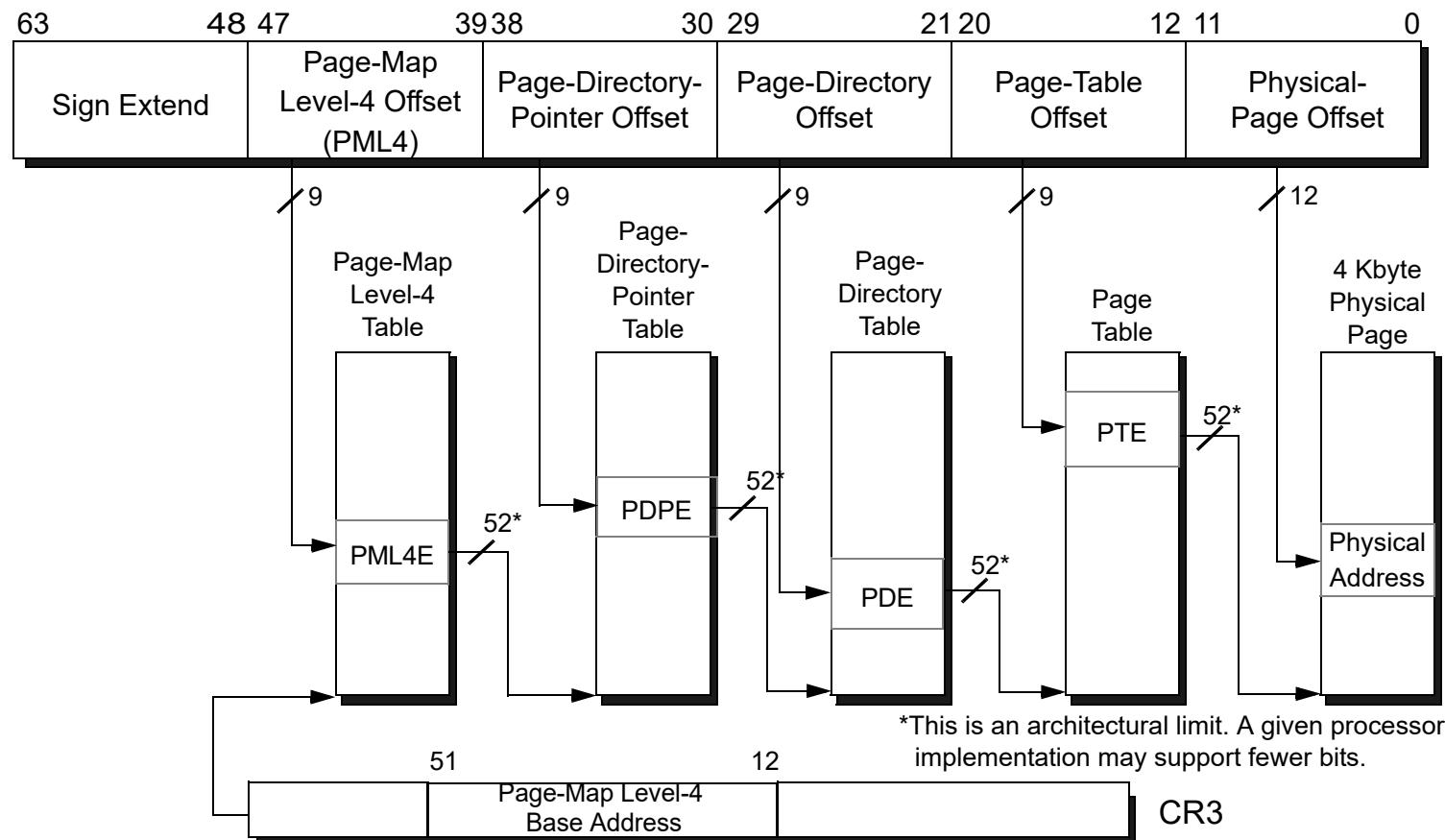
[source: AMD64 Architecture Programmer's Manual Volume 2: System Programming]



And finally with
N level page tables.

4 level page table

Virtual Address





N level page table

More details:

**AMD64 Architecture Programmer's Manual Volume 2:
System Programming** ([link](#))

Lets have a look on page 131 (PDF: 191), table 5-1.



Summary and outlook

Summary

- MMU
- Virtual addresses
- 1/2/N level page table

Outlook

- Memory hierarchy
- Associative memory
- Translation lookaside buffer
- Cache
- Memory protection



Summary and outlook

Summary

- MMU
- Virtual addresses
- 1/2/N level page table

Outlook

- Memory hierarchy
- Associative memory
- Translation lookaside buffer
- Cache
- Memory protection