Sistemi Operativi I

Corso di Laurea in Informatica

2025-2026

Gabriele Tolomei

Dipartimento di Informatica

Sapienza Università di Roma

tolomei@di.uniroma1.it

Problems Seen So Far

- Contiguous allocation
  - Hard to grow or shrink process memory
- Fragmentation
  - Frequent compaction needed
- Process entirely loaded
  - Swapping helps but it may be too inefficient

Paging

- A memory management scheme that addresses the problems above

Paging

- A memory management scheme that addresses the problems above
- The logical address space of a process is still **contiguous** but it is divided into **fixed-size blocks**, called **pages**

Paging

- A memory management scheme that addresses the problems above
- The logical address space of a process is still **contiguous** but it is divided into **fixed-size blocks**, called **pages**
- Contiguous allocation is no longer required as logical pages can be mapped to **non-contiguous** physical **frames**

Paging

- A memory management scheme that addresses the problems above
- The logical address space of a process is still **contiguous** but it is divided into **fixed-size blocks**, called **pages**
- Contiguous allocation is no longer required as logical pages can be mapped to **non-contiguous** physical **frames**
- External fragmentation is eliminated because pages have fixed size
  - Internal fragmentation may still occur though

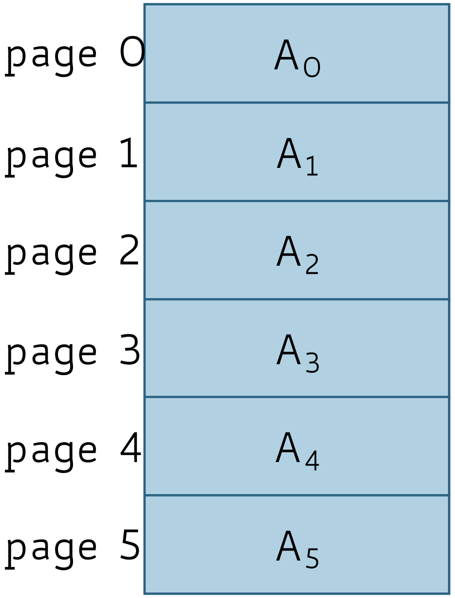
Paging

- A memory management scheme that addresses the problems above
- The logical address space of a process is still **contiguous** but it is divided into **fixed-size blocks**, called **pages**
- Contiguous allocation is no longer required as logical pages can be mapped to **non-contiguous** physical **frames**
- External fragmentation is eliminated because pages have fixed size
  - Internal fragmentation may still occur though

90/10 Rule

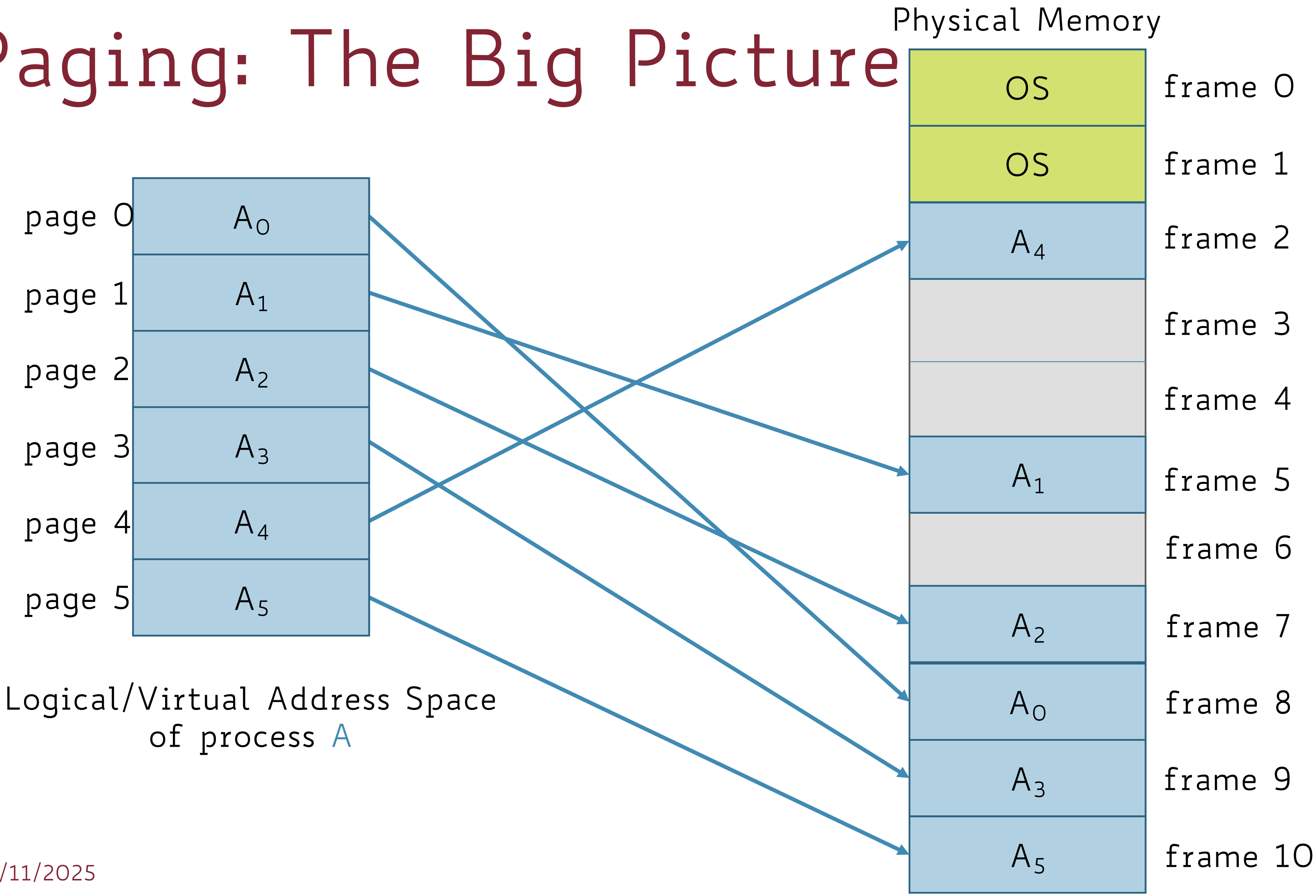
Processes spend 90% of their time accessing only 10% of their allocated memory space

Paging: The Big Picture



Logical/Virtual Address Space
of process A

Paging: The Big Picture



Basic OS Responsibilities for Paging

- The OS has 2 main responsibilities:
  - mapping between logical pages and physical frames
  - translating logical addresses to physical addresses

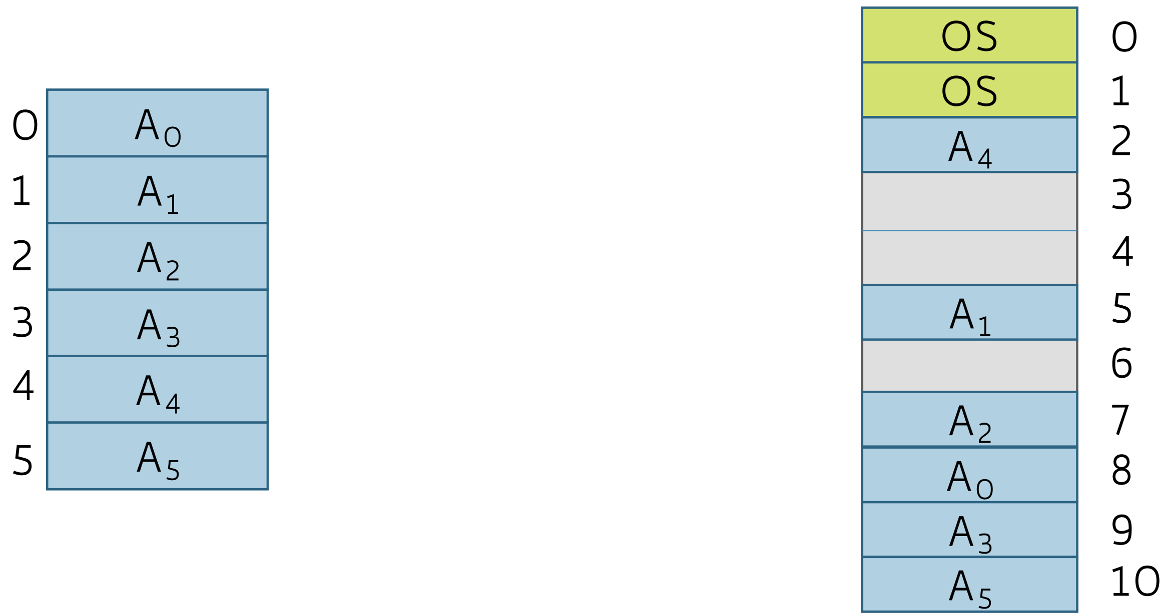
Basic OS Responsibilities for Paging

- The OS has 2 main responsibilities:
  - mapping between logical pages and physical frames
  - translating logical addresses to physical addresses
- All of this must be done efficiently!
  - Remember, memory addresses are referenced all the time

Basic OS Responsibilities for Paging

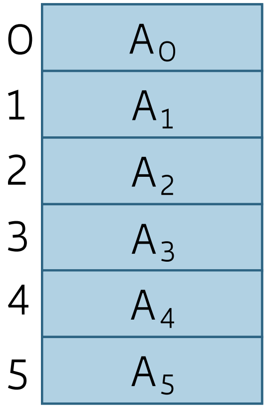
- The OS has 2 main responsibilities:
  - mapping between logical pages and physical frames
  - translating logical addresses to physical addresses
- All of this must be done efficiently!
  - Remember, memory addresses are referenced all the time
- OS needs dedicated support for doing it → **Page Table**

Page Table: Mapping Pages to Frames

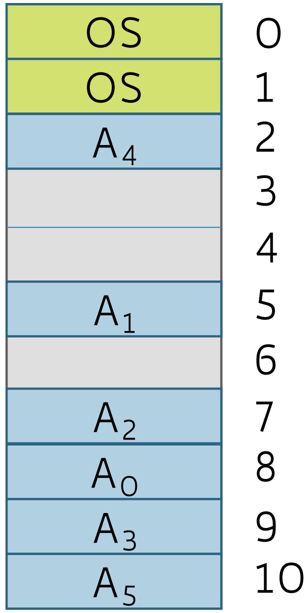


Page Table: Mapping Pages to Frames

Lookup table to retrieve what frame a page is stored in

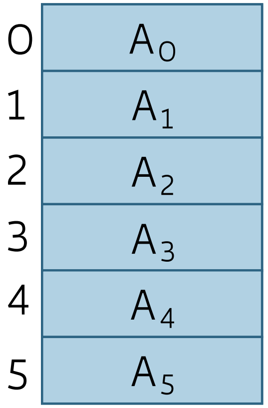


Page	Frame
0	8
1	5
2	7
3	9
4	2
5	10

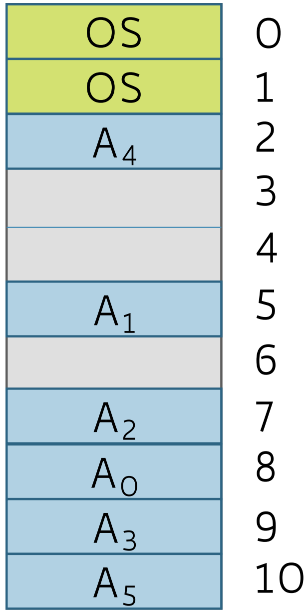


Page Table: Mapping Pages to Frames

Lookup table to retrieve what frame a page is stored in



Page	Frame
0	8
1	5
2	7
3	9
4	2
5	10



We have assumed all pages of a process are mapped to physical frames, but this is not always the case

Page Table: Virtual to Physical Address

- Processes use virtual (logical) addresses to refer to memory (not page number!)

Page Table: Virtual to Physical Address

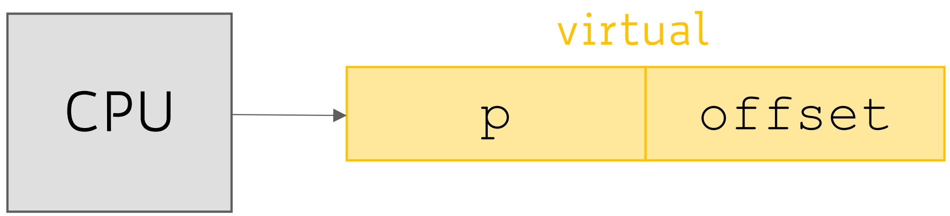
- Processes use virtual (logical) addresses to refer to memory (not page number!)
- Virtual (logical) address space is still contiguous starting from 0

Page Table: Virtual to Physical Address

- Processes use virtual (logical) addresses to refer to memory (not page number!)
- Virtual (logical) address space is still contiguous starting from 0
- Page table must ultimately translate virtual address to physical address

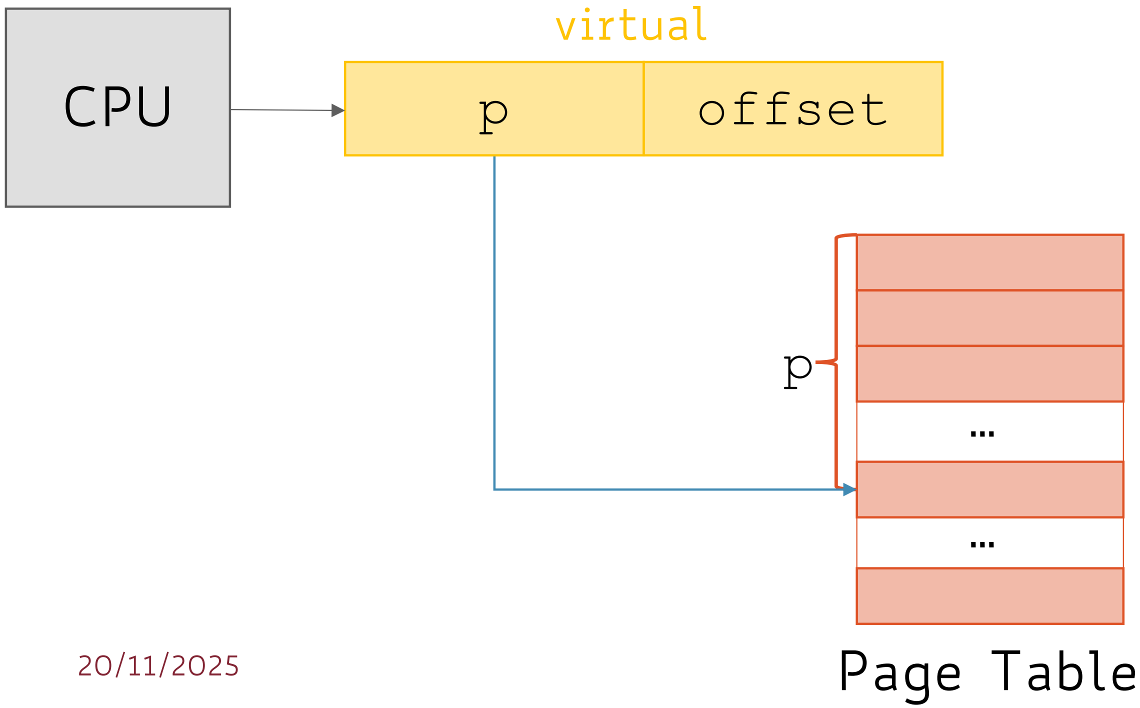
Page Table: Virtual to Physical Address

virtual address consists of 2 parts:
- `p`: page number where the address resides
- `offset`: relative from the beginning of the page



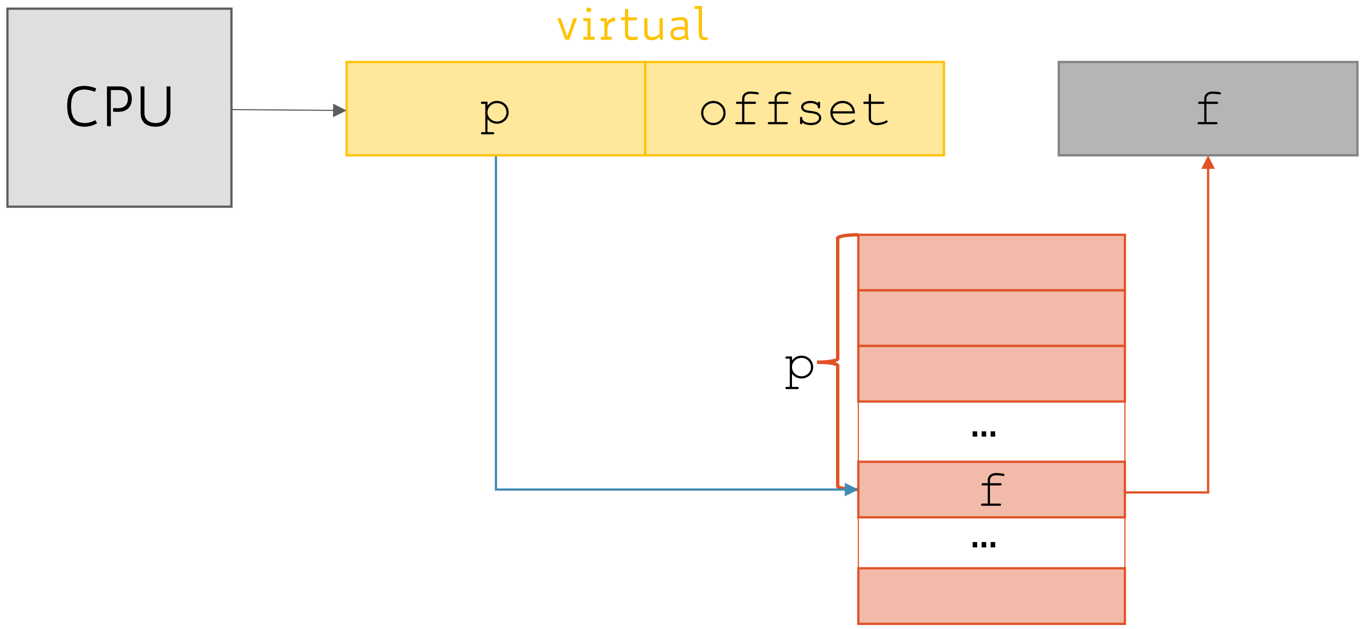
Page Table: Virtual to Physical Address

virtual address consists of 2 parts:
- `p`: page number where the address resides
- `offset`: relative from the beginning of the page



Page Table: Virtual to Physical Address

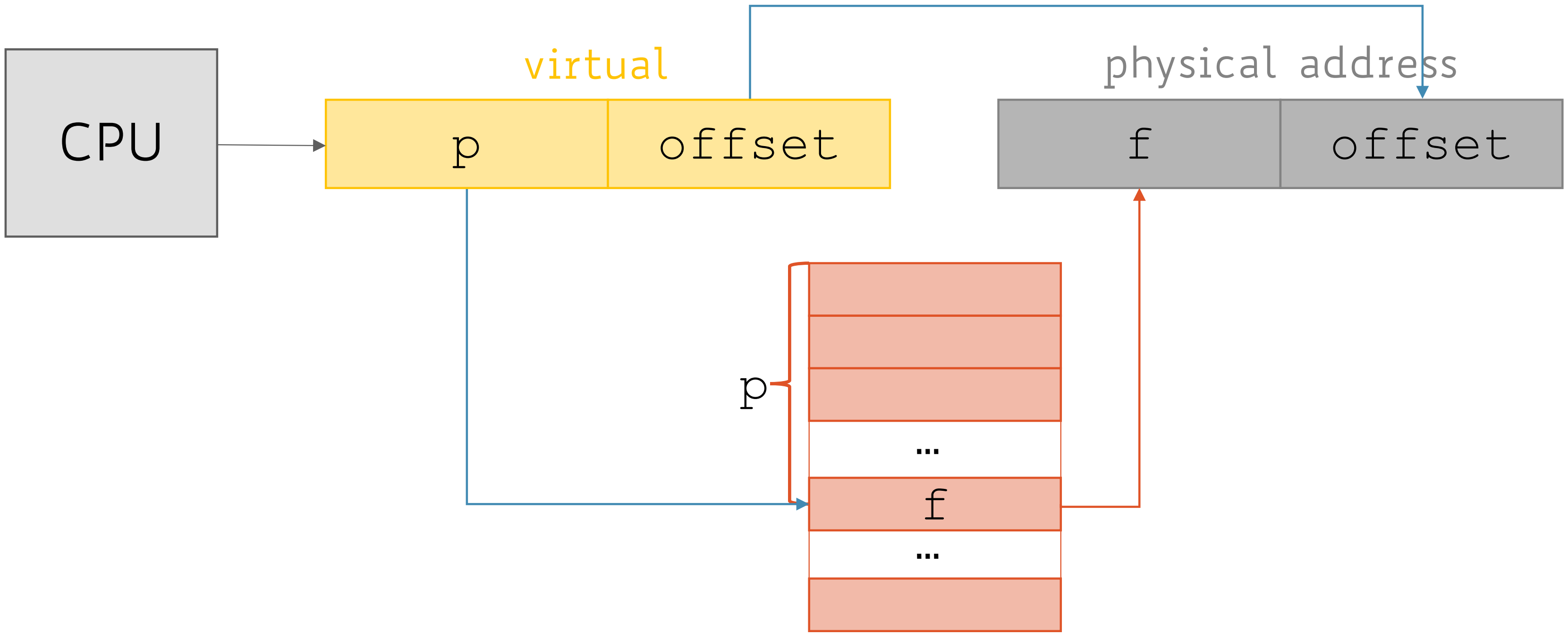
virtual address consists of 2 parts:
- `p`: page number where the address resides
- `offset`: relative from the beginning of the page



Page Table

Page Table: Virtual to Physical Address

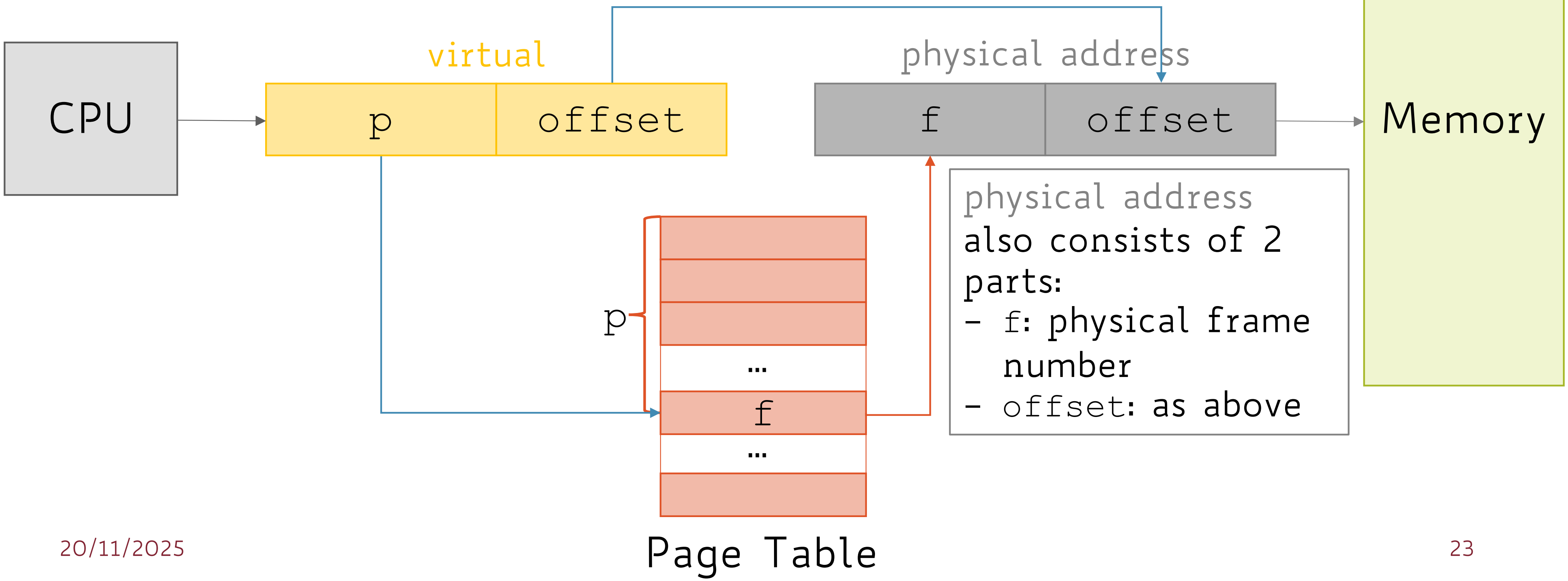
virtual address consists of 2 parts:
- `p`: page number where the address resides
- `offset`: relative from the beginning of the page



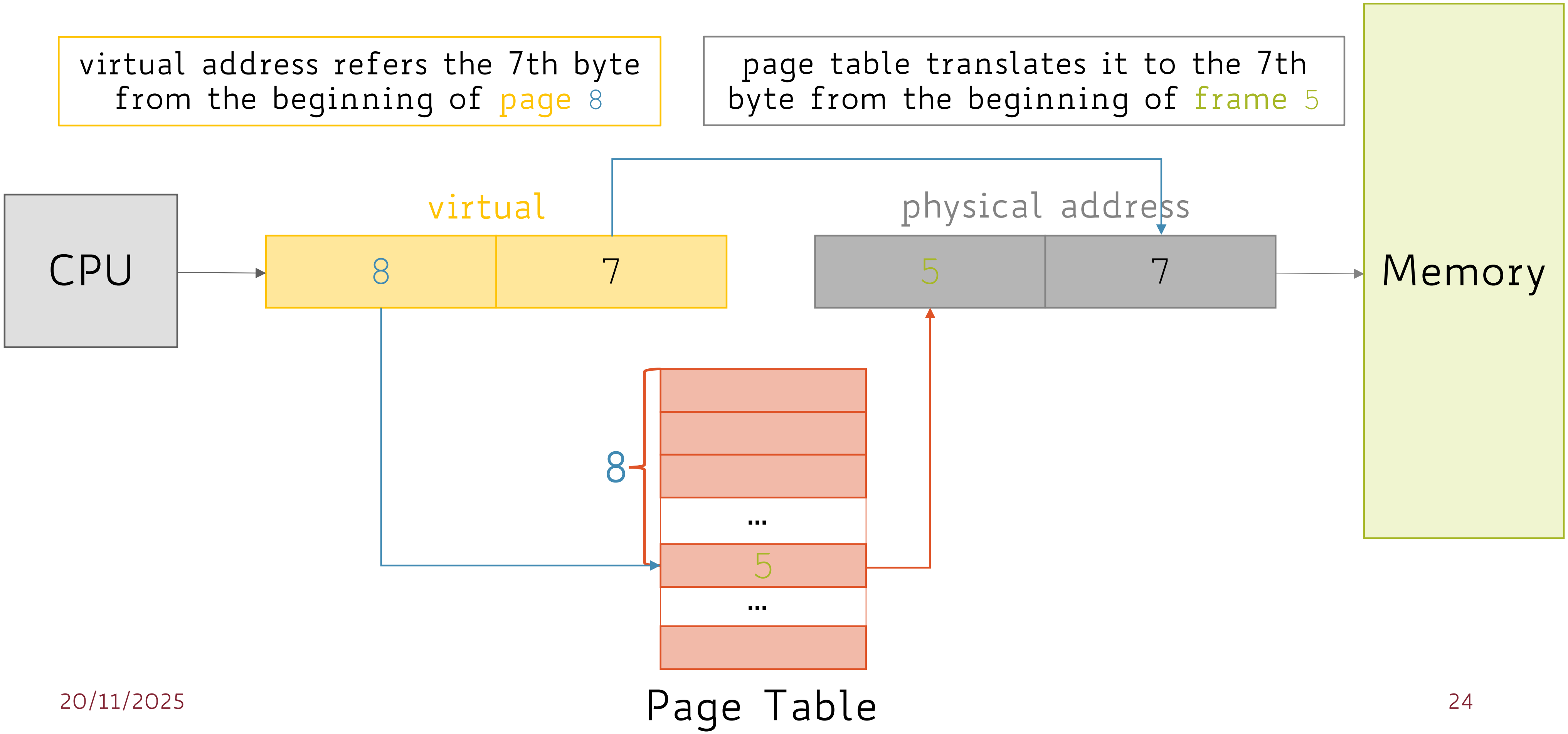
Page Table

Page Table: Virtual to Physical Address

virtual address consists of 2 parts:
- `p`: page number where the address resides
- `offset`: relative from the beginning of the page



Page Table: Example of Address Translation



Paging as Dynamic Relocation

- Paging is an advanced form of dynamic relocation
- Each virtual address is bound by the page table to a physical address
- Page table can be seen just as a set of base (relocation) registers, one for each frame
- Mapping is invisible to the user process: the OS maintains the page table and translation happens in hardware (via the MMU)
- Protection is provided similarly to dynamic relocation (limit register)

Paging: Steps

How does MMU use page table to translate a virtual address **`x`** into a physical address **`y`**?

Paging: Steps

How does MMU use page table to translate a virtual address **`x`** into a physical address **`y`**?

1. Get the page number (**`p`**) and the **`offset`** where the virtual address **`x`** resides

Paging: Steps

How does MMU use page table to translate a virtual address **`x`** into a physical address **`y`**?

1. Get the page number (**`p`**) and the **`offset`** where the virtual address **`x`** resides

2. Use **`p`** to index into the page table to retrieve the frame number **`f`**

Paging: Steps

How does MMU use page table to translate a virtual address **`x`** into a physical address **`y`**?

1. Get the page number (**`p`**) and the **`offset`** where the virtual address **`x`** resides

2. Use **`p`** to index into the page table to retrieve the frame number **`f`**

3. Combine **`f`** with **`offset`** to obtain the physical address **`y`**

Paging: Get `p` and `offset` from `x`

Suppose we have **`50B`** of physical memory available for user processes

Paging: Get `p` and `offset` from `x`

Suppose we have **`50B`** of physical memory available for user processes

Assume we use paging with page (frame) size **`S`** = **`10B`**

Paging: Get `p` and `offset` from `x`

Suppose we have **`50B`** of physical memory available for user processes

Assume we use paging with page (frame) size **`S`** = **`10B`**

Each process can generate virtual addresses in the range **`[0, 49]`**

Paging: Get **`p`** and **`offset`** from **`x`**

Suppose we have **`50B`** of physical memory available for user processes

Assume we use paging with page (frame) size **`S = 10B`**

Each process can generate virtual addresses in the range **`[0, 49]`**

Suppose a process generates virtual address **`x = 27`**

Paging: Get **p** and **offset** from **x**

Suppose we have **50B** of physical memory available for user processes

Assume we use paging with page (frame) size **S** = **10B**

Each process can generate virtual addresses in the range **[0, 49]**

Suppose a process generates virtual address **x** = **27**

p = **x** **div** **S**

page number

Paging: Get **p** and **offset** from **x**

Suppose we have **50B** of physical memory available for user processes

Assume we use paging with page (frame) size **S** = **10B**

Each process can generate virtual addresses in the range **[0, 49]**

Suppose a process generates virtual address **x** = **27**

p = **x div S**

page number

p = **27 div 10 = 2**

Paging: Get `p` and `offset` from `x`

Suppose we have **50B** of physical memory available for user processes

Assume we use paging with page (frame) size **S** = **10B**

Each process can generate virtual addresses in the range **[0, 49]**

Suppose a process generates virtual address **x** = **27**

p = **x** **div** **S**

page number

offset = **x** **mod** **S**

offset

p = **27** **div** **10** **= 2**

Paging: Get `p` and `offset` from `x`

Suppose we have **50B** of physical memory available for user processes

Assume we use paging with page (frame) size **S = 10B**

Each process can generate virtual addresses in the range **[0, 49]**

Suppose a process generates virtual address **x = 27**

p = x div S

page number

offset = x mod S

offset

p = 27 div 10 = 2

offset = 27 mod 10 = 7

Paging: Get `p` and `offset` from `x`

Suppose we have **50B** of physical memory available for user processes

Assume we use paging with page (frame) size **S = 10B**

Each process can generate virtual addresses in the range **[0, 49]**

Suppose a process generates virtual address **x = 27**

p = x div S

page number

offset = x mod S

offset

p = 27 div 10 = 2

offset = 27 mod 10 = 7

Address translation requires a **div** and a **mod** operation

Paging: Implementation Details

- Page/frame numbers and page/frame sizes are determined by the architecture

Paging: Implementation Details

- Page/frame numbers and page/frame sizes are determined by the architecture
- Page/frame sizes are typically a **power of 2**, ranging between 512B and 8192B (i.e., 8KiB)

Paging: Implementation Details

- Page/frame numbers and page/frame sizes are determined by the architecture
- Page/frame sizes are typically a **power of 2**, ranging between 512B and 8192B (i.e., 8KiB)
- Powers of 2 make the translation from virtual to physical address easy (i.e., no need for **`div`** and **`mod`**)

Paging: Implementation Details

- Page/frame numbers and page/frame sizes are determined by the architecture
- Page/frame sizes are typically a **power of 2,** ranging between 512B and 8192B (i.e., 8KiB)
- Powers of 2 make the translation from virtual to physical address easy (i.e., no need for **`div`** and **`mod`**)

Why?

Paging: Why Power of 2?

- Virtual address is made of m bits

Paging: Why Power of 2?

- Virtual address is made of m bits
  - Then, virtual address space (i.e., the set of bytes addressable by each user process) is 2^m long, and ranges between $[0, 2^m-1]$

Paging: Why Power of 2?

- Virtual address is made of m bits
  - Then, virtual address space (i.e., the set of bytes addressable by each user process) is 2^m long, and ranges between $[0, 2^m-1]$
- Assume page (frame) size is 2^n, $n < m$

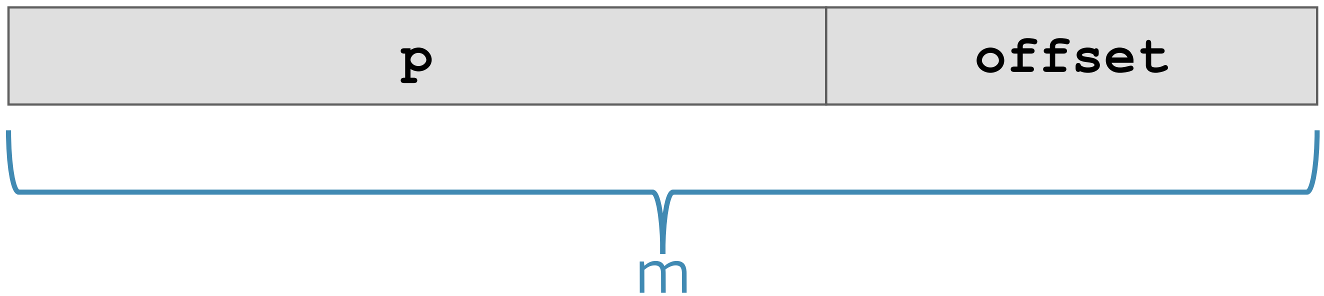
Paging: Why Power of 2?

- Virtual address is made of m bits
  - Then, virtual address space (i.e., the set of bytes addressable by each user process) is 2^m long, and ranges between $[0, 2^m-1]$
- Assume page (frame) size is 2^n, $n < m$
- The higher $m-n$ bits of the virtual address indicates the page number

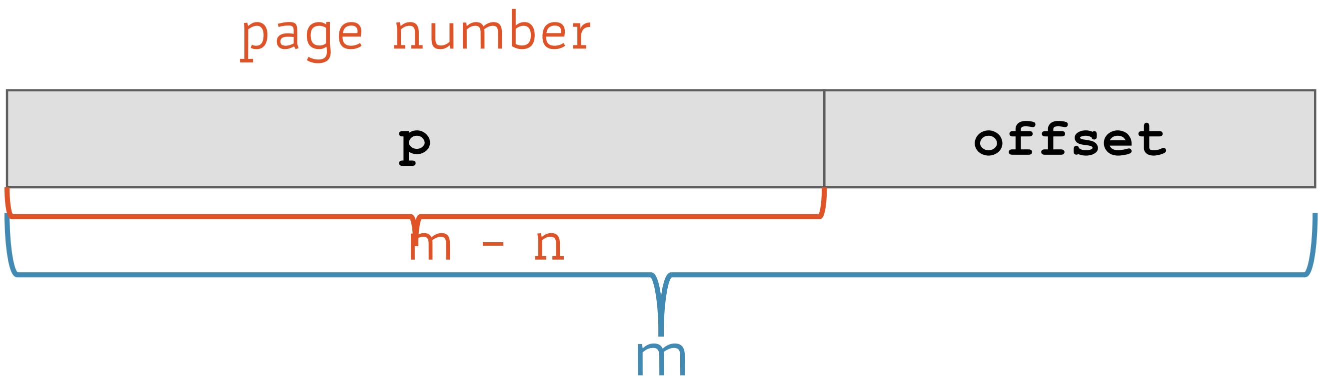
Paging: Why Power of 2?

- Virtual address is made of m bits
  - Then, virtual address space (i.e., the set of bytes addressable by each user process) is 2^m long, and ranges between $[0, 2^m-1]$
- Assume page (frame) size is 2^n, $n < m$
- The higher $m-n$ bits of the virtual address indicates the **page number**
- The low order n bits represent the **offset**

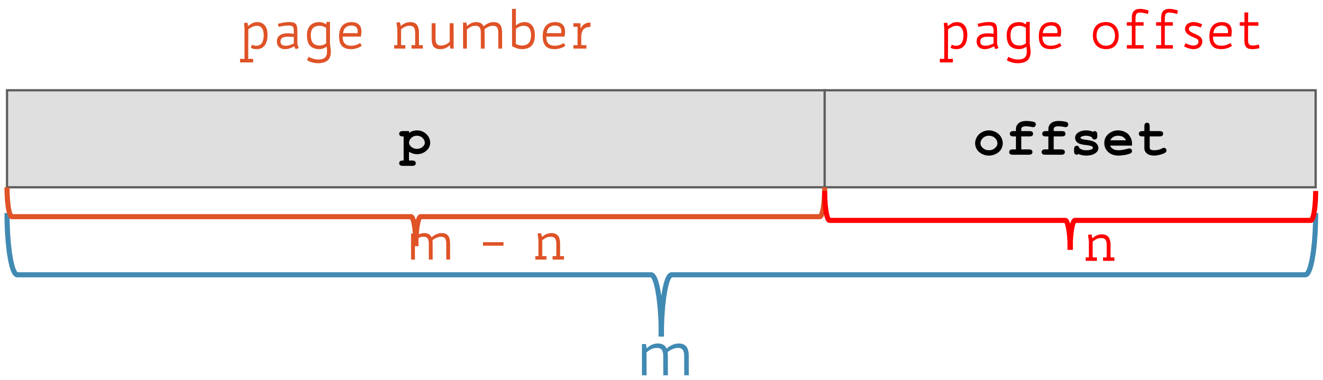
Paging: Why Power of 2?



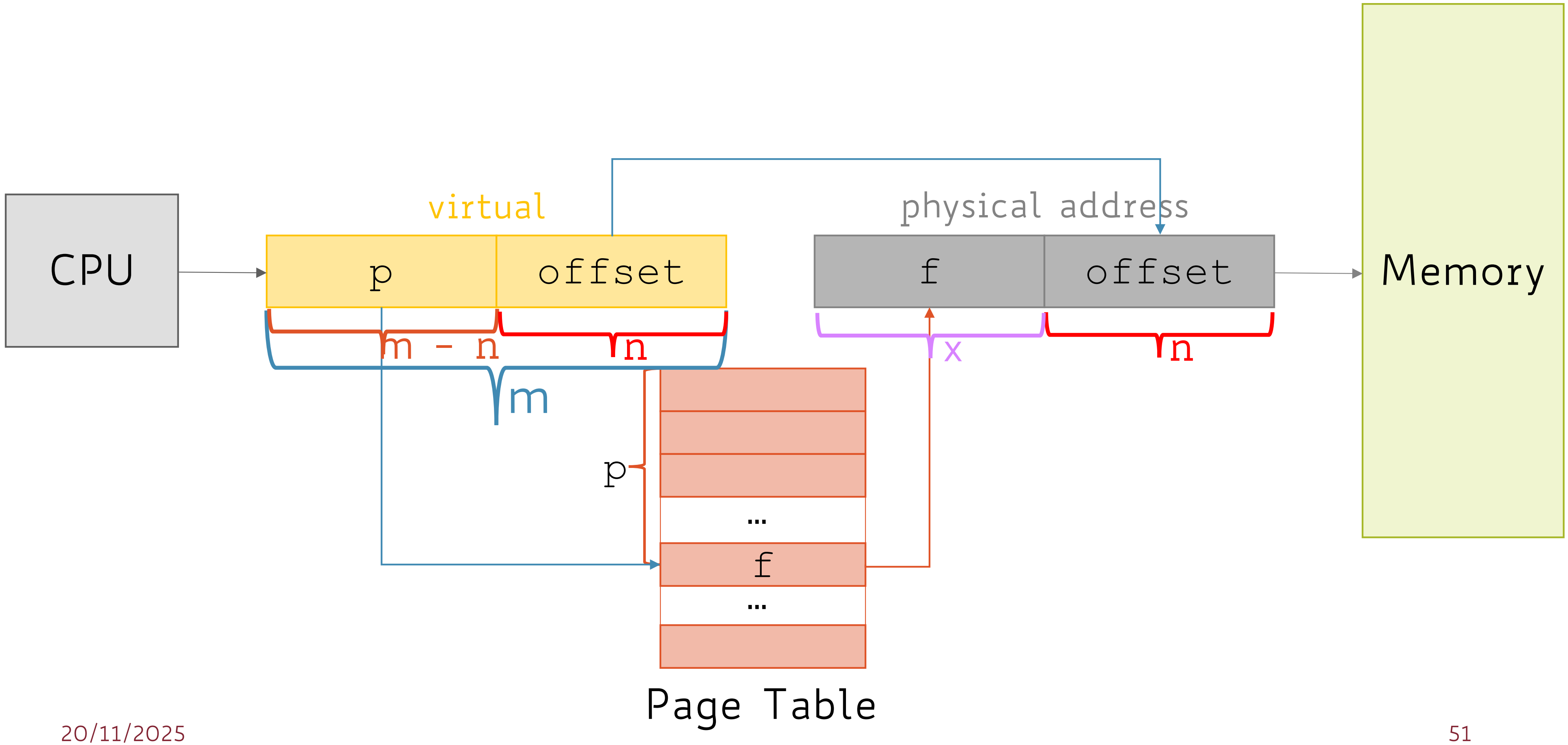
Paging: Why Power of 2?



Paging: Why Power of 2?



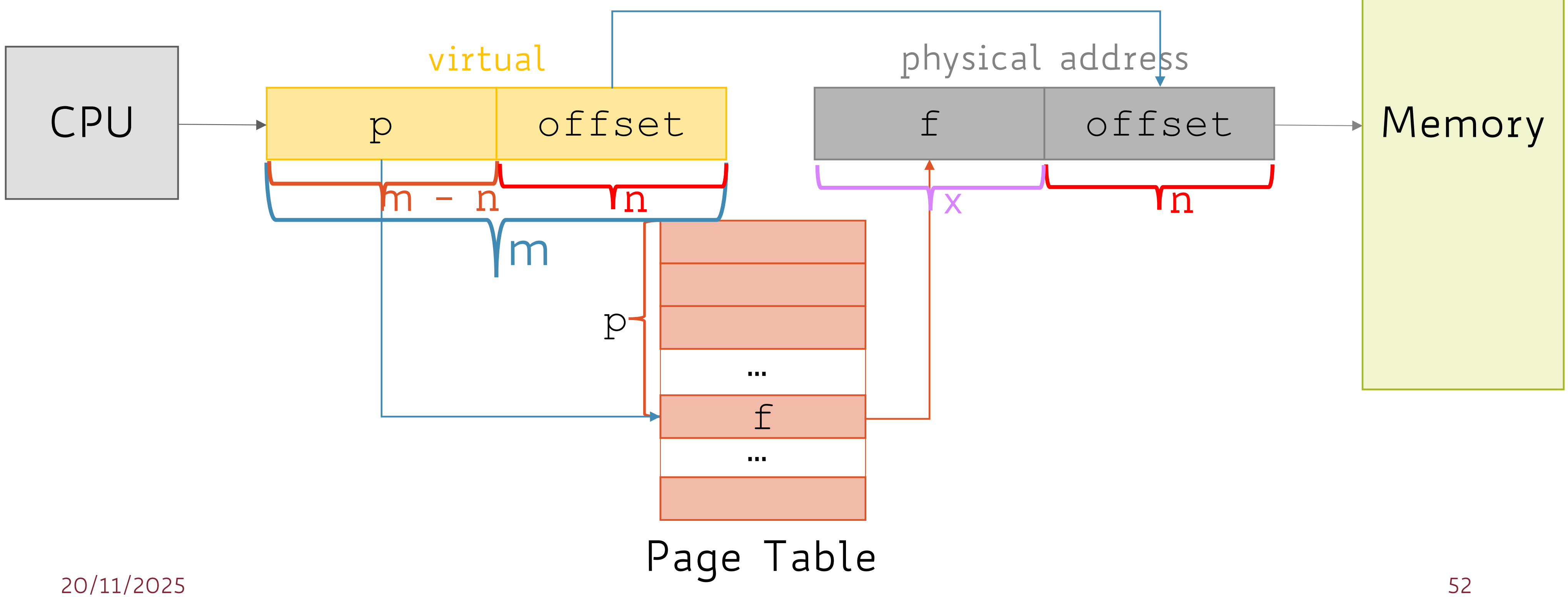
Paging: Practical Details



Paging: Practical Details

NOTE

m-n doesn't necessarily have to be equal to x



Paging: Practical Details

- Typical values of virtual address size is $m = 32$ or 64 bits
  - The virtual address space is 2^{32} = 4GiB or 2^{64} = 16EiB

Paging: Practical Details

- Typical values of virtual address size is m = 32 or 64 bits
  - The virtual address space is 2^{32} = 4GiB or 2^{64} = 16EiB
- Typical values of page/frame sizes is n = 12 bits
  - Each page/frame is 2^{12} = 4KiB

Paging: Practical Details

- Typical values of virtual address size is m = 32 or 64 bits
  - The virtual address space is 2^{32} = 4GiB or 2^{64} = 16EiB
- Typical values of page/frame sizes is n = 12 bits
  - Each page/frame is 2^{12} = 4KiB
- Assuming m = 32 bits, there are $2^{m-n} = 2^{20}$ = ~1M pages/frames
  - The page table has 2^{20} entries (i.e., one for each page/frame)

Paging: Practical Example

Suppose we have a virtual memory and a physical memory, both of size M = 1024B (1KiB)

Q1

How many bits are needed for a virtual/physical address (assuming single-byte addressing)

Paging: Practical Example

Suppose we have a virtual memory and a physical memory, both of size M = 1024B (1KiB)

Q1

How many bits are needed for a virtual/physical address (assuming single-byte addressing)

R1

10 bits to address M = 1024 bytes (both for virtual and physical address)

Paging: Practical Example

Now, assume we use paging with page/frame size S = 16B

Q2

How big is the page table? (i.e., how many pages/entries does it have to index?)

Paging: Practical Example

Now, assume we use paging with page/frame size S = 16B

Q2

How big is the page table? (i.e., how many pages/entries does it have to index?)

R2

T = M / S = 1024 memory bytes / 16 bytes per page = 64 pages

Paging: Practical Example

Q3

What is `p` and `offset` (i.e., how many bits for `p` and `offset`?)

Paging: Practical Example

Q3

What is `p` and `offset` (i.e., how many bits for `p` and `offset`?)

R3

Our logical address is made of $m = 10$ bits

$n = 4$ bits are used to represent the `offset`, as each page/frame is $S = 16$ bytes

$m-n = 6$ bits are used to represent page number `p`, as there are $T = 64$ pages

Paging: Practical Example

Q4

Translate the virtual address $x = 42$, assuming the following page table

page	frame
0	12
1	5
2	37
3	0
…	..
63	29

Paging: Practical Example

Q4

Translate the virtual address $x = 42$, assuming the following page table

page	frame
0	12
1	5
2	37
3	0
…	..
63	29

R4

$\mathbf{p} = x \ \mathbf{div} \ S = 42 \ \mathbf{div} \ 16 = \mathbf{2}$

Paging: Practical Example

Q4

Translate the virtual address $x = 42$, assuming the following page table

page	frame
0	12
1	5
2	37
3	0
…	..
63	29

R4

$\mathbf{p} = x\ \mathbf{div}\ S = 42\ \mathbf{div}\ 16 = \mathbf{2}$

Paging: Practical Example

Q4

Translate the virtual address `x = 42`, assuming the following page table

page	frame
0	12
1	5
2	37
3	0
…	..
63	29

R4

p = x **div** S = 42 **div** 16 = **2**

offset = x **mod** S = 42 **mod** 16 = **10**

10th byte from the beginning of frame 37

Paging: Practical Example 2

Suppose we still have a virtual memory and a physical memory, both of size $M = 1024B$

Paging: Practical Example 2

Suppose we still have a virtual memory and a physical memory, both of size M = 1024B

Modern computers however operate natively on multiple of bytes (i.e., words) rather than single-byte. Typical values of word length is: 16, 32 or 64 bits.
If we assume 32-bit architecture (i.e., word = 32 bits = 4 bytes), virtual addresses refer to words instead of bytes

Paging: Practical Example 2

Suppose we still have a virtual memory and a physical memory, both of size M = 1024B

Modern computers however operate natively on multiple of bytes (i.e., words) rather than single-byte. Typical values of word length is: 16, 32 or 64 bits.
If we assume 32-bit architecture (i.e., word = 32 bits = 4 bytes), virtual addresses refer to words instead of bytes

Q1
How many bits are therefore needed to address the number of words available on M?

Paging: Practical Example 2

Suppose we still have a virtual memory and a physical memory, both of size M = 1024B

Modern computers however operate natively on multiple of bytes (i.e., words) rather than single-byte. Typical values of word length is: 16, 32 or 64 bits.
If we assume 32-bit architecture (i.e., word = 32 bits = 4 bytes), virtual addresses refer to words instead of bytes

Q1
How many bits are therefore needed to address the number of words available on M?

R1
8 bits to address M = 1024/4 = 256 4-byte words

Paging: Practical Example 2

Now, assume we still use paging with page/frame size S = 16B

Q2

How big is the page table? (i.e., how many pages/entries does it have to index?)

Paging: Practical Example 2

Now, assume we still use paging with page/frame size S = 16B

Q2

How big is the page table? (i.e., how many pages/entries does it have to index?)

R2

T = M / S = 1024 memory bytes / 16 bytes per page = 64 pages

Paging: Practical Example 2

Q3

What is `p` and `offset` (i.e., how many bits for `p` and `offset`?)

Paging: Practical Example 2

Q3

What is p and offset (i.e., how many bits for p and offset?)

R3

Our logical address is now made of m = 8 bits

n = 2 bits are used to represent the offset, as each page/frame is:

S = 16 bytes = 4 * 4-byte words

m-n = 6 bits are used to represent page number p, as there are still

T = 64 pages

Paging: Practical Example 2

Q4

Translate the virtual address $x = 7$, assuming the following page table

page	frame
0	12
1	5
2	37
3	0
…	..
63	29

Paging: Practical Example 2

Q4

Translate the virtual address $x = 7$, assuming the following page table

page	frame
0	12
1	5
2	37
3	0
…	..
63	29

Remember: now virtual address refers to a 4-byte word!

Paging: Practical Example 2

Q4
Translate the virtual address x = 7, assuming the following page table

page	frame
0	12
1	5
2	37
3	0
…	..
63	29

S = 16 bytes = 4 * 4-byte words
Must be expressed in terms of number of words

R4
p = x **div** S = 7 **div** 4 = **1**

Paging: Practical Example 2

Q4

Translate the virtual address x = 7, assuming the following page table

page	frame
0	12
1	5
2	37
3	0
…	..
63	29

R4

p = x **div** S = 7 **div** 4 = **1**

offset = x **mod** S = 7 **mod** 4 = **3**

3rd word from the beginning of frame 5

Summary

- Paging solves the problem of external fragmentation by relaxing the assumption of contiguous allocation

Summary

- Paging solves the problem of external fragmentation by relaxing the assumption of contiguous allocation
- It decouples logical from physical memory layout

Summary

- Paging solves the problem of external fragmentation by relaxing the assumption of contiguous allocation
- It decouples logical from physical memory layout
- An advanced form of dynamic relocation that requires OS intervention (**Page Tables**)

Summary

- Paging solves the problem of external fragmentation by relaxing the assumption of contiguous allocation
- It decouples logical from physical memory layout
- An advanced form of dynamic relocation that requires OS intervention (**Page Tables**)
- Standard paging alone can be inefficient…