# Sistemi Operativi I

Corso di Laurea in Informatica

2023-2024

Gabriele Tolomei

Dipartimento di Informatica

Sapienza Università di Roma

tolomei@di.uniroma1.it

SAPIENZA
UNIVERSITÀ DI ROMA

## A Quick Step Back: Segmentation

- Most users (programmers) do not think of their programs as existing in one continuous linear address space

## A Quick Step Back: Segmentation

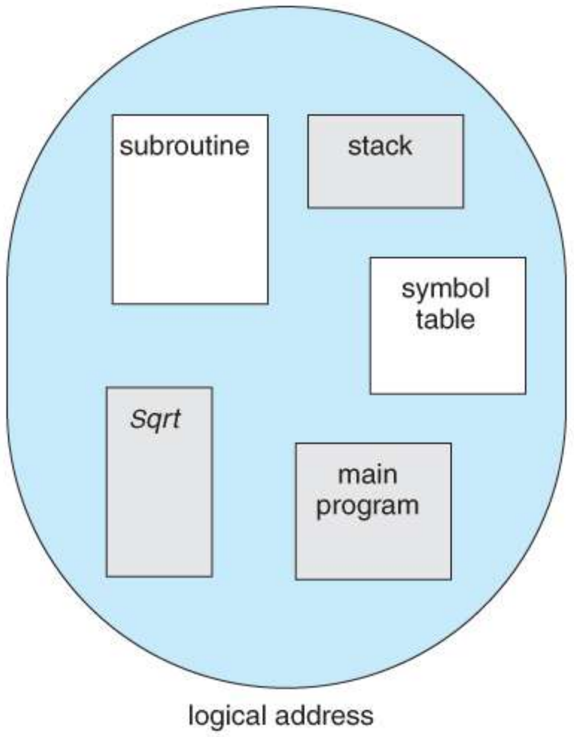
- Most users (programmers) do not think of their programs as existing in one continuous linear address space
- Rather they think of memory divided in multiple **segments**, each dedicated to a specific use, such as code, data, stack, heap, etc.

## A Quick Step Back: Segmentation

- Most users (programmers) do not think of their programs as existing in one continuous linear address space
- Rather they think of memory divided in multiple **segments**, each dedicated to a specific use, such as code, data, stack, heap, etc.
- Memory segmentation supports this view by providing addresses with a **segment number** (mapped to a segment base address) and an **offset**

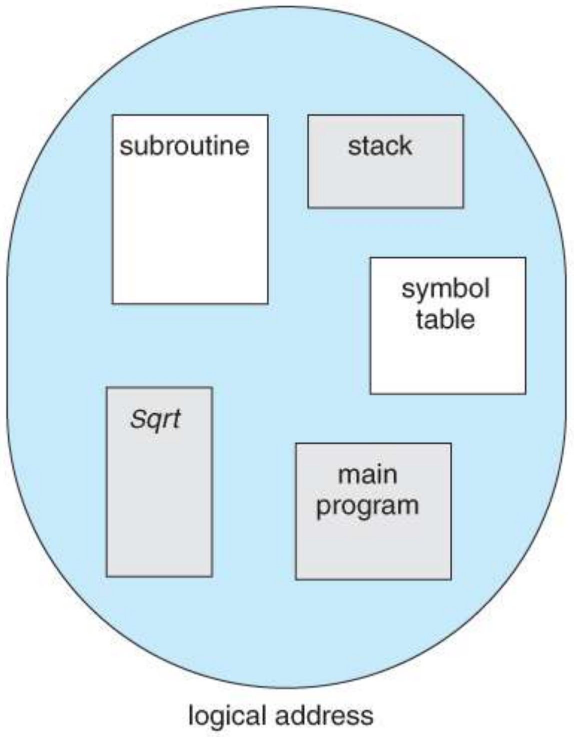
# Segmentation: Example

A C compiler generating 5 segments for the user code, library code, global (static) variables, the stack, and the heap



## Segmentation: Example

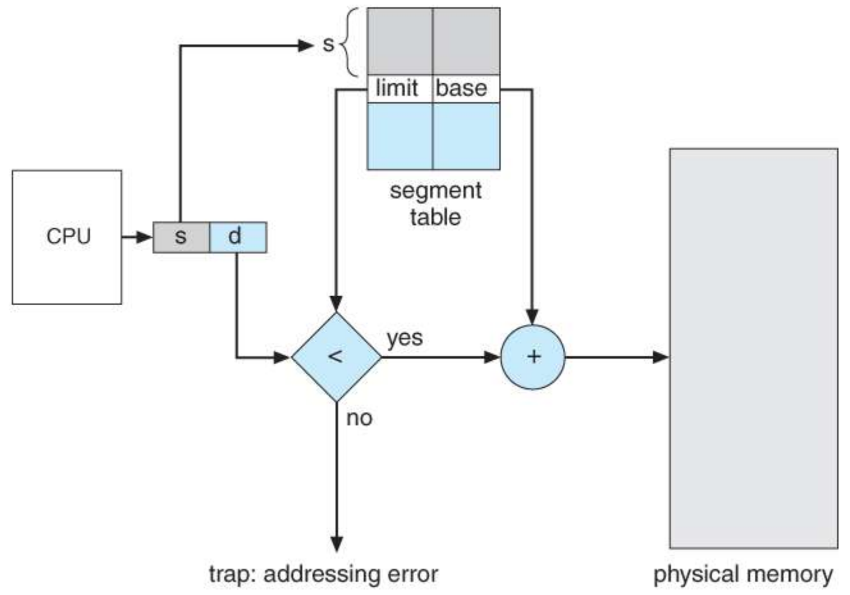
A C compiler generating 5 segments for the user code, library code, global (static) variables, the stack, and the heap



The compiler generates addresses identifying segments and offset

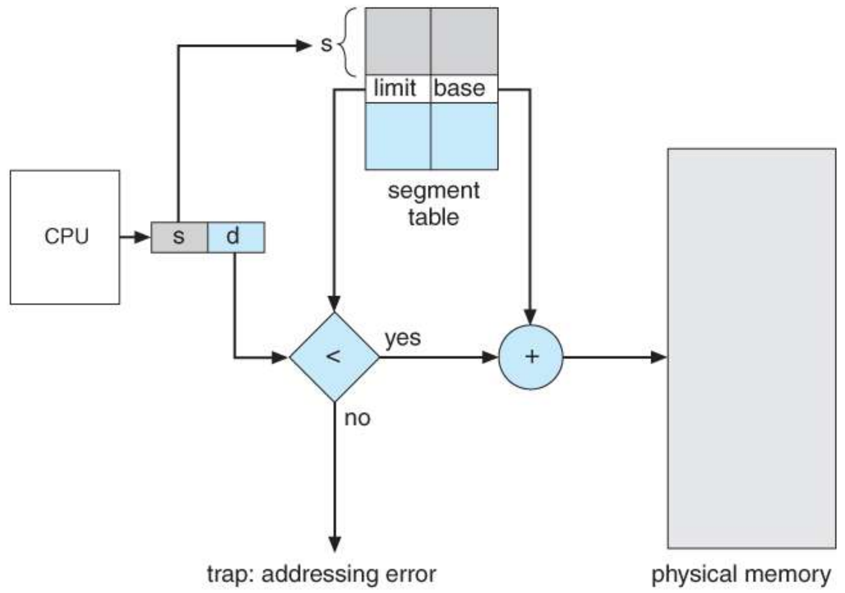
## Segmentation Hardware

A segment table maps segment-offset addresses to physical addresses, and simultaneously checks for invalid addresses, using a system similar to the page tables and relocation base registers discussed previously



## Segmentation Hardware

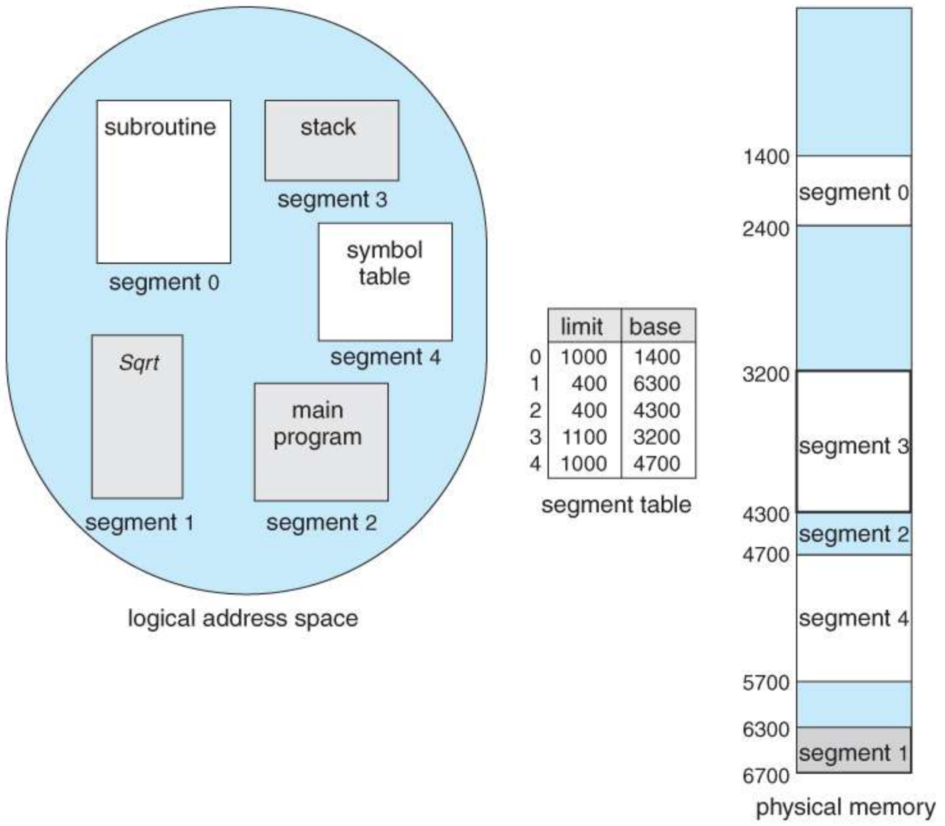
A segment table maps segment-offset addresses to physical addresses, and simultaneously checks for invalid addresses, using a system similar to the page tables and relocation base registers discussed previously



Note that we came back to the assumption that each segment is kept in contiguous memory and may be of different size…

# Segment Table

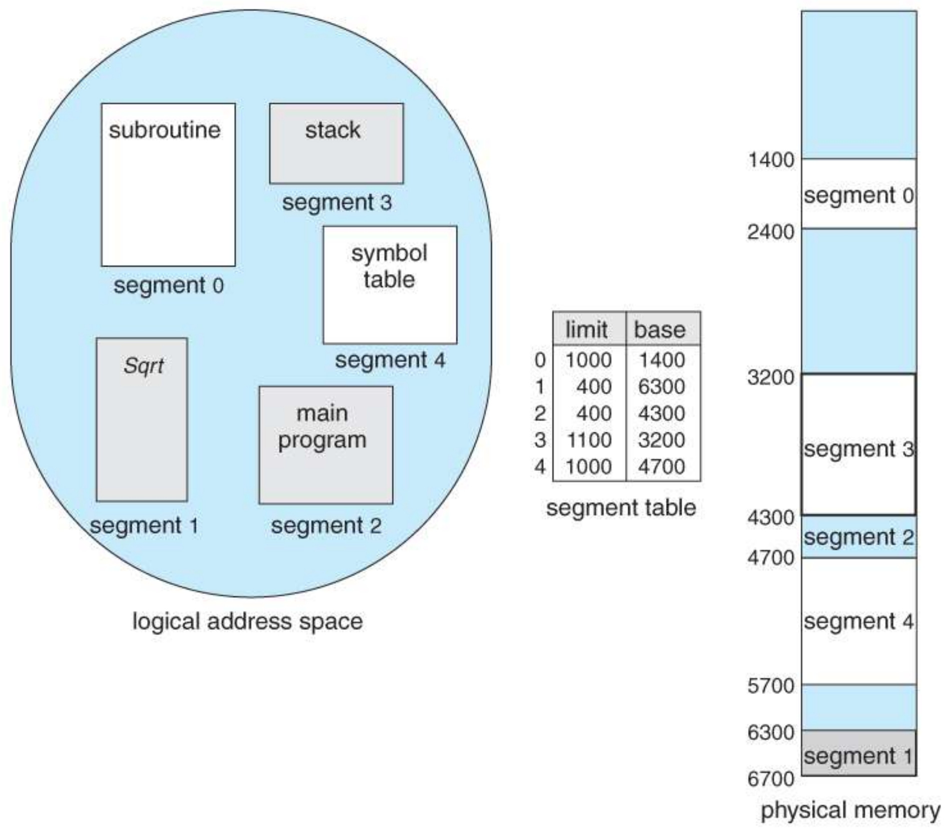
Each entry contains a base address in memory, the length of the segment, plus additional protection information (e.g., sharing, read/write permissions)



| | limit | base |
|---|---|---|
| 0 | 1000 | 1400 |
| 1 | 400 | 6300 |
| 2 | 400 | 4300 |
| 3 | 1100 | 3200 |
| 4 | 1000 | 4700 |

segment table

# Segment Table

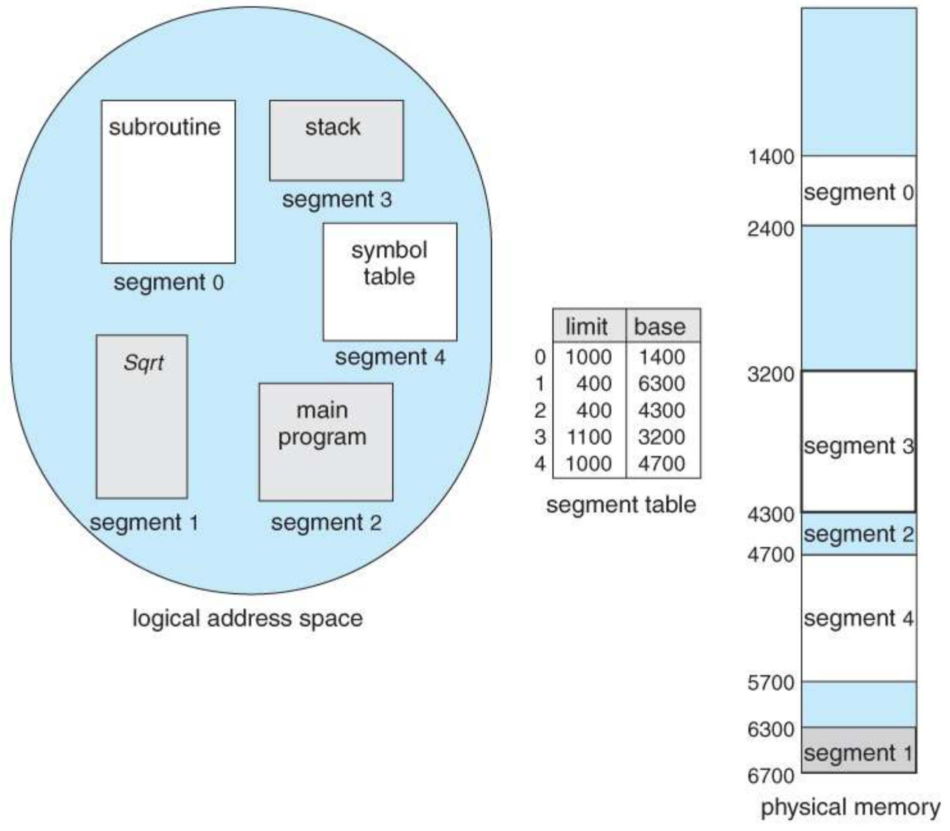


| | limit | base |
|---|---|---|
| 0 | 1000 | 1400 |
| 1 | 400 | 6300 |
| 2 | 400 | 4300 |
| 3 | 1100 | 3200 |
| 4 | 1000 | 4700 |

Each entry contains a base address in memory, the length of the segment, plus additional protection information (e.g., sharing, read/write permissions)

Segment Table can be stored using few base/limit hardware registers

# Segment Table



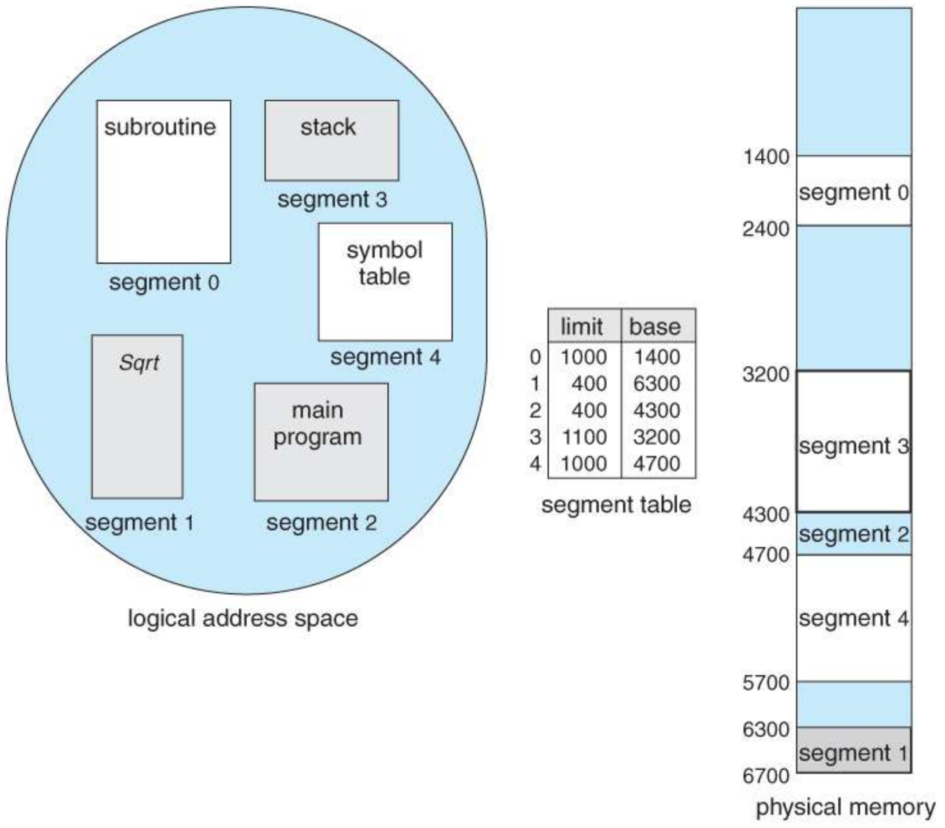
| | limit | base |
|---|---|---|
| 0 | 1000 | 1400 |
| 1 | 400 | 6300 |
| 2 | 400 | 4300 |
| 3 | 1100 | 3200 |
| 4 | 1000 | 4700 |

Each entry contains a base address in memory, the length of the segment, plus additional protection information (e.g., sharing, read/write permissions)

Segment Table can be stored using few base/limit hardware registers

Page Table cannot be stored using hw registers as there might be potentially too many page entries

# Segment Table



|   | limit | base |
|---|---|---|
| 0 | 1000 | 1400 |
| 1 | 400 | 6300 |
| 2 | 400 | 4300 |
| 3 | 1100 | 3200 |
| 4 | 1000 | 4700 |

Each entry contains a base address in memory, the length of the segment, plus additional protection information (e.g., sharing, read/write permissions)

Segment Table can be stored using few base/limit hardware registers

Page Table cannot be stored using hw registers as there might be potentially too many page entries

Segment Table, instead, must store a very limited amount of segments per process (3÷5)

## Implementing (Basic) Segmentation

- Compiler generates virtual addresses whose top-most significant bits indicate the segment number

## Implementing (Basic) Segmentation

- Compiler generates virtual addresses whose top-most significant bits indicate the segment number
- Segmentation can be combined both with static or dynamic relocation

# Implementing (Basic) Segmentation

- Compiler generates virtual addresses whose top-most significant bits indicate the segment number
- Segmentation can be combined both with static or dynamic relocation
- Each segment is allocated a contiguous block of memory
  - External fragmentation can be an issue again!

## Implementing (Basic) Segmentation

- Compiler generates virtual addresses whose top-most significant bits indicate the segment number
- Segmentation can be combined both with static or dynamic relocation
- Each segment is allocated a contiguous block of memory
    - External fragmentation can be an issue again!
- Additional HW (like TLB cache) might be needed if programs use many logical segments

## Combine Segmentation with Paging

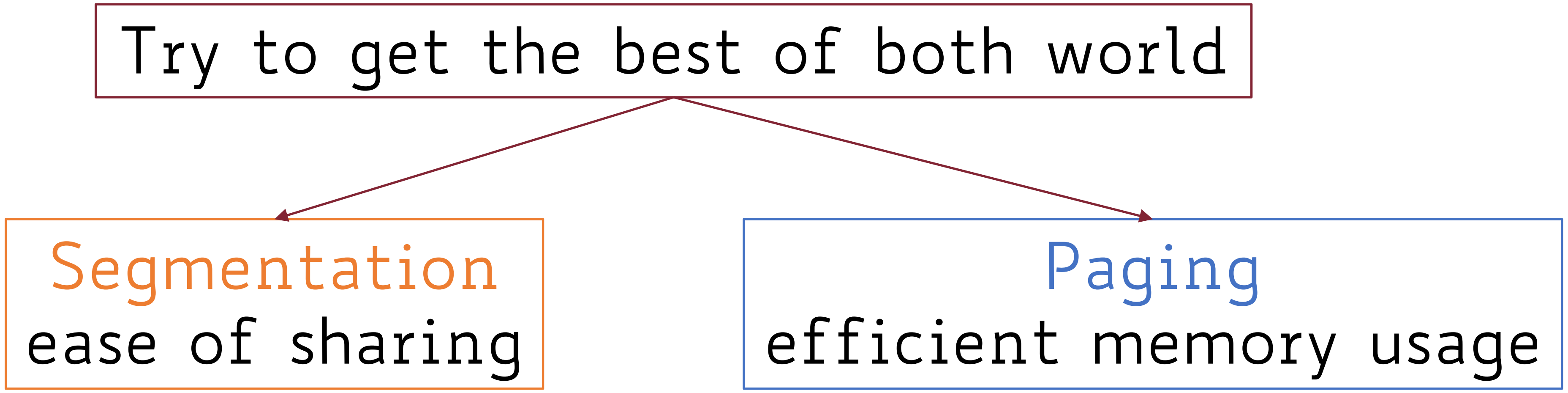
Try to get the best of both world

## Combine Segmentation with Paging

Try to get the best of both world

Segmentation
ease of sharing

# Combine Segmentation with Paging



## Combine Segmentation with Paging

Try to get the best of both world

Segmentation
ease of sharing

Paging
efficient memory usage

How?

## Combine Segmentation with Paging

Try to get the best of both world

Segmentation
ease of sharing

Paging
efficient memory usage

How?

Apply paging to segments!

## Combine Segmentation with Paging

- Each process' virtual address space is seen as a collection of segments (i.e., logical units of arbitrary size)

## Combine Segmentation with Paging

- Each process' virtual address space is seen as a collection of segments (i.e., logical units of arbitrary size)
- Physical address space is still seen as a sequence of fixed-size frames

## Combine Segmentation with Paging

- Each process' virtual address space is seen as a collection of segments (i.e., logical units of arbitrary size)
- Physical address space is still seen as a sequence of fixed-size frames
- Segments are usually larger than physical page frames

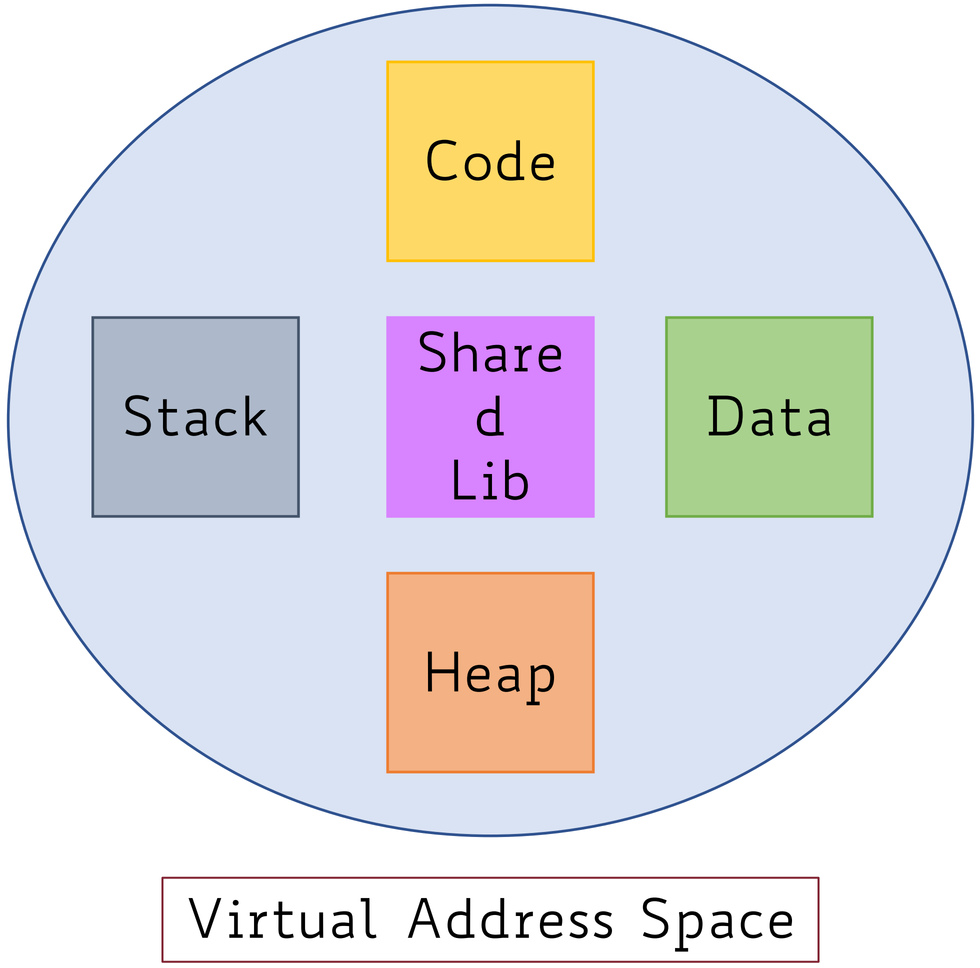
## Combine Segmentation with Paging

- Each process' virtual address space is seen as a collection of segments (i.e., logical units of arbitrary size)
- Physical address space is still seen as a sequence of fixed-size frames
- Segments are usually larger than physical page frames

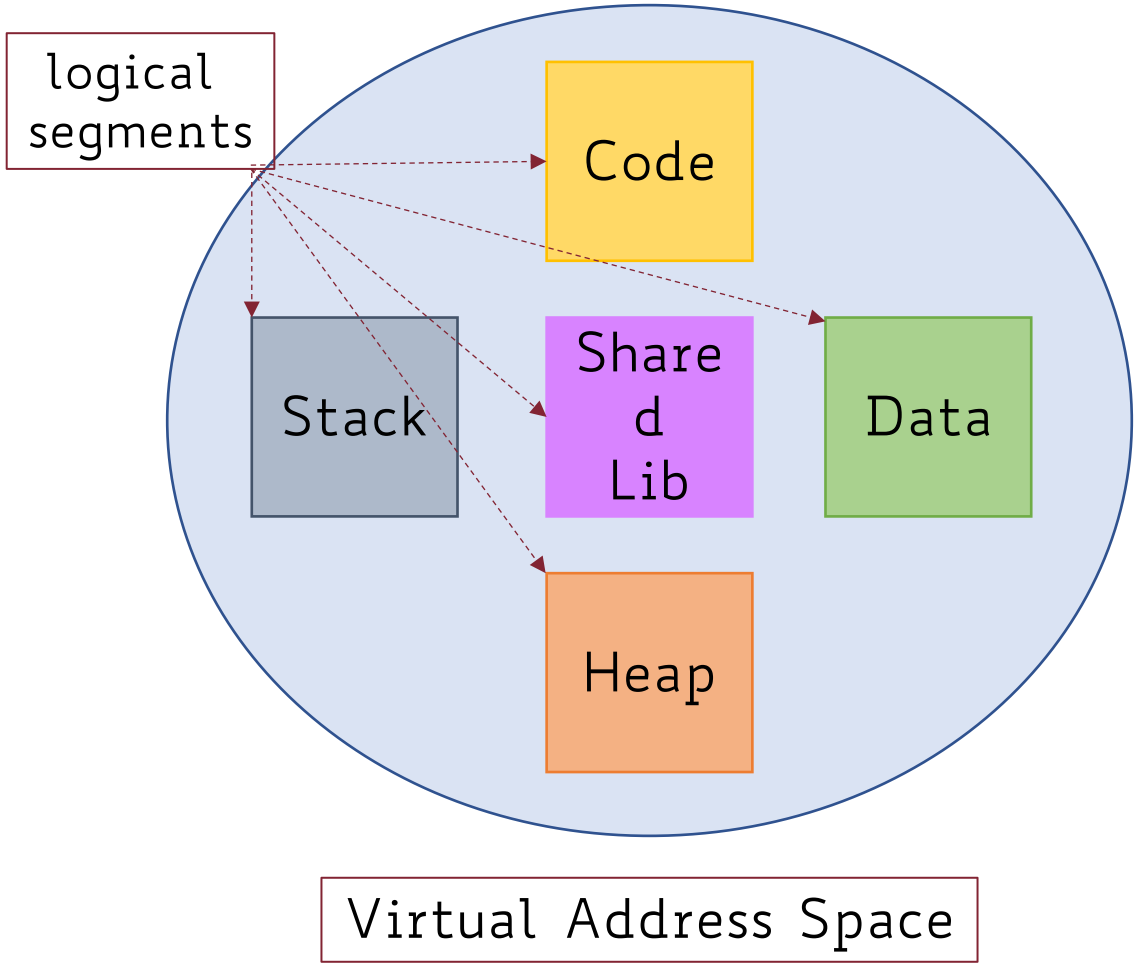
$\Downarrow$

Map a logical segment onto multiple page frames

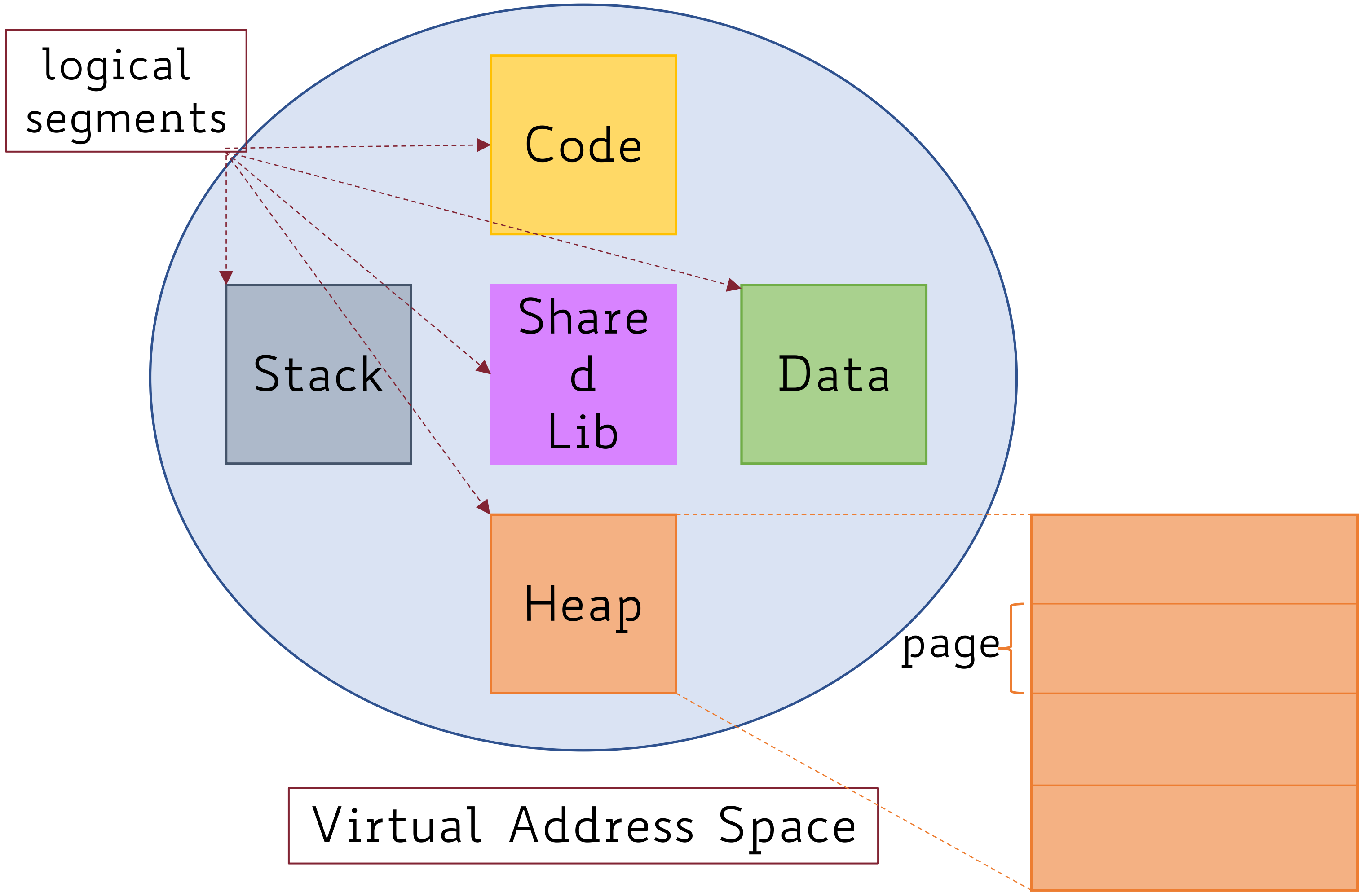
# Paging Logical Segments



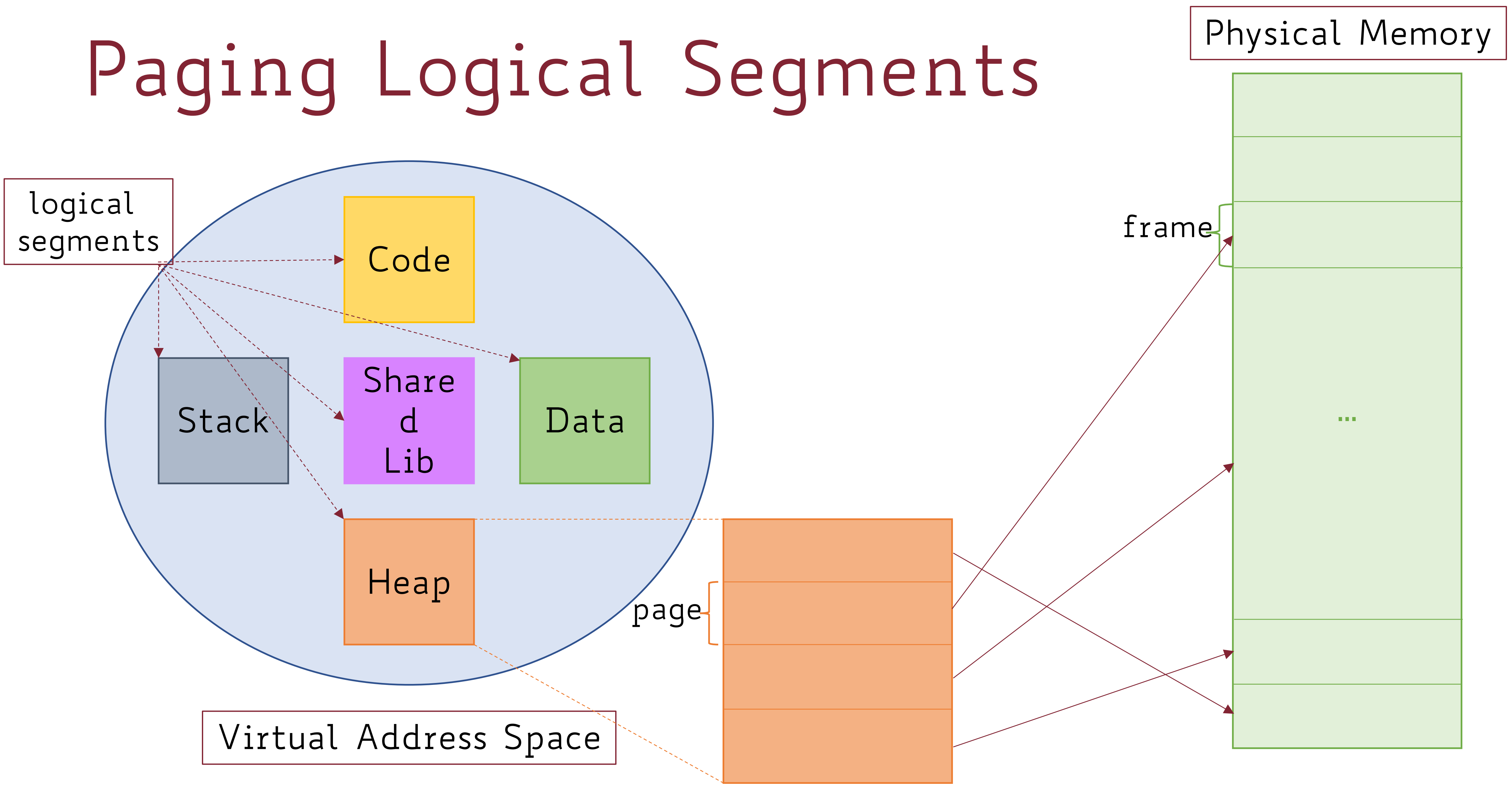
# Paging Logical Segments



# Paging Logical Segments

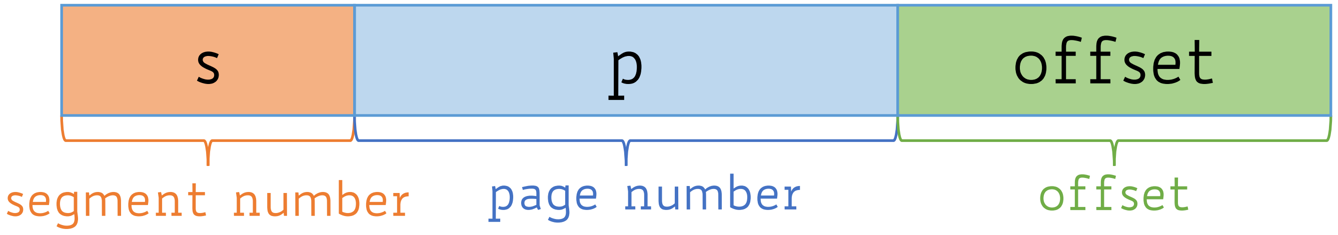


# Paging Logical Segments



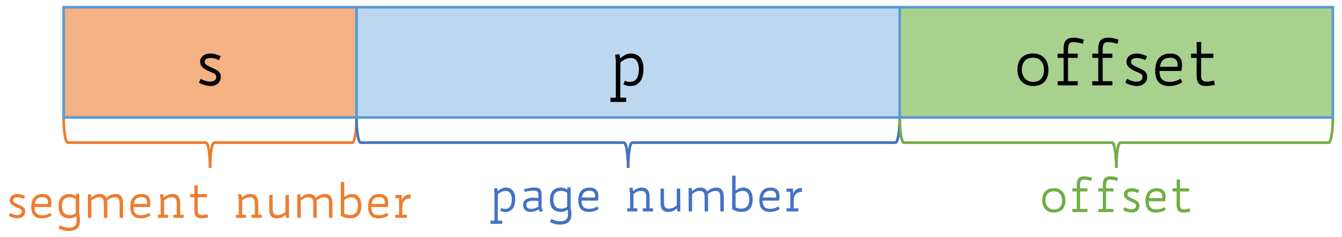
# Address Translation with Segmented Paging

A virtual address now becomes:



# Address Translation with Segmented Paging

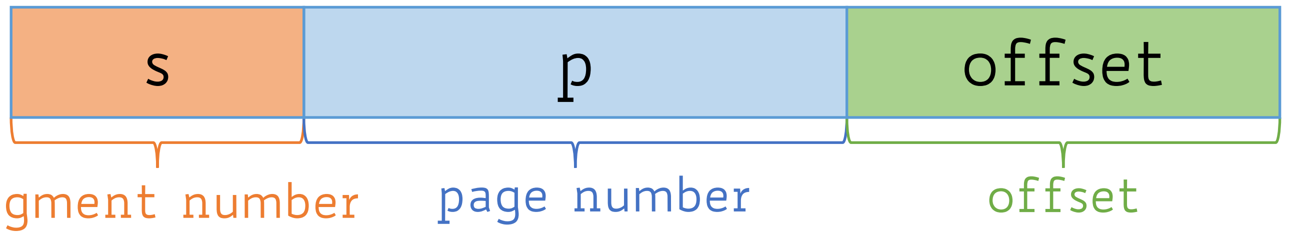
A virtual address now becomes:



- The segment number indexes into the **segment table**, which contains the base address of the **page table** for *that* segment

# Address Translation with Segmented Paging

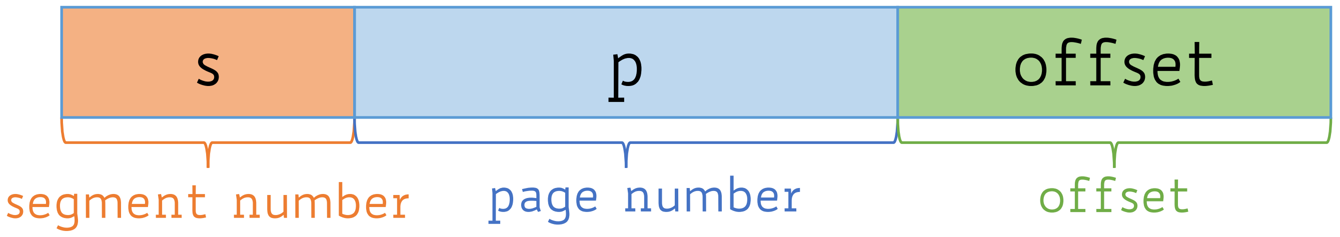
A virtual address now becomes:



- The segment number indexes into the **segment table**, which contains the base address of the **page table** for *that* segment
- Check the page number + offset against the limit of the segment

# Address Translation with Segmented Paging

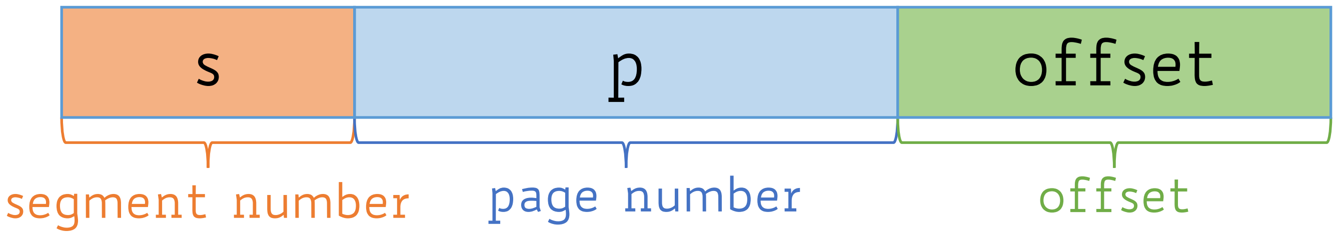
A virtual address now becomes:



- Use the page number to index the page table to get the physical frame number

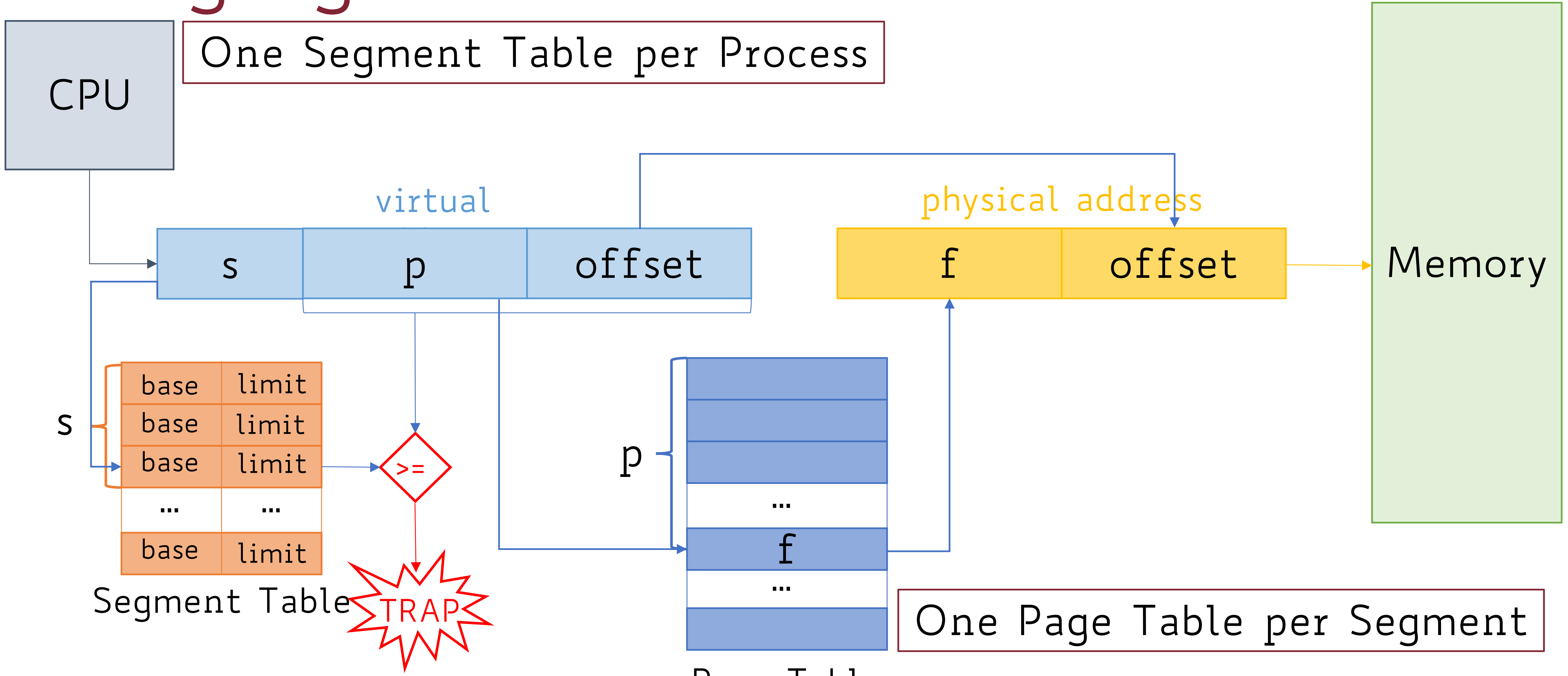
# Address Translation with Segmented Paging

A virtual address now becomes:



- Use the page number to index the page table to get the physical frame number
- Add the frame number to the offset to get the physical address

## Address Translation with Segmented Paging



## Segmented Paging: Implementation

Where are segment tables and page tables stored?

## Segmented Paging: Implementation

Where are segment tables and page tables stored?

Option 1

segment tables in a small number of registers

page tables in main memory with TLB cache

## Segmented Paging: Implementation

Where are segment tables and page tables stored?

Option 2

both segment tables and page tables in main memory with TLB cache

TLB lookup done using segment index and page index

# Segmented Paging: Implementation

Where are segment tables and page tables stored?

**Option 1**

segment tables in a small number of registers

page tables in main memory with TLB cache

Faster but the number of segments is limited

**Option 2**

both segment tables and page tables in main memory with TLB cache

TLB lookup done using segment index and page index

## Segmented Paging: Implementation

Where are segment tables and page tables stored?

**Option 1**

segment tables in a small number of registers
page tables in main memory with TLB cache

Faster but the number of segments is limited

**Option 2**

both segment tables and page tables in main memory with TLB cache
TLB lookup done using segment index and page index

Slower but more flexible

## Segmented Paging Hardware: Practical Example 3

Suppose a physical memory of 1024 addressable words (assuming 1 word = 1 byte)

Frame size is 64 words (i.e., 64 bytes)

Page table size (i.e., number of entries) is thus 1024 bytes/64 bytes per frame = 16

8 logical segments

## Segmented Paging Hardware: Practical Example 3

Suppose a physical memory of 1024 addressable words (assuming 1 word = 1 byte)

Frame size is 64 words (i.e., 64 bytes)

Page table size (i.e., number of entries) is thus 1024 bytes/64 bytes per frame = 16

8 logical segments

Q1

How many bits are therefore needed for the physical address?

# Segmented Paging Hardware: Practical Example 3

Suppose a physical memory of 1024 addressable words (assuming 1 word = 1 byte)

Frame size is 64 words (i.e., 64 bytes)

Page table size (i.e., number of entries) is thus 1024 bytes/64 bytes per frame = 16

8 logical segments

Q1

How many bits are therefore needed for the physical address?

R1

10 bits to address M = 1024/1 = 1024 1-byte words

## Segmented Paging Hardware: Practical Example 3

Suppose a physical memory of 1024 addressable words (assuming 1 word = 1 byte)

Frame size is 64 words (i.e., 64 bytes)

Page table size (i.e., number of entries) is thus 1024 bytes/64 bytes per frame = 16

8 logical segments

Q2

How many bits are therefore needed for the virtual address?

# Segmented Paging Hardware: Practical Example 3

Q2

How many bits are therefore needed for the virtual address?

## Segmented Paging Hardware: Practical Example 3

Q2

How many bits are therefore needed for the virtual address?

R2

3 bits to address 8 logical segments (s)

4 bits to address 16 entries of the page table

6 bits to address 64 individual words (i.e., bytes) within each page

# Segmented Paging Hardware: Practical Example 3

Q2

How many bits are therefore needed for the virtual address?

R2

3 bits to address 8 logical segments (s)

4 bits to address 16 entries of the page table

6 bits to address 64 individual words (i.e., bytes) within each page

13 bits (virtual address) vs. 10 bits (physical address)

## Sharing Pages and Segments

- We already showed individual pages can be shared across processes by copying page entries

## Sharing Pages and Segments

- We already showed individual pages can be shared across processes by copying page entries
- Segments can also be shared by sharing segment table entries:

## Sharing Pages and Segments

- We already showed individual pages can be shared across processes by copying page entries
- Segments can also be shared by sharing segment table entries:
  - Two or more processes map the shared segment on its own segment table (one segment table per process) to the same page table

## Sharing Pages and Segments

- We already showed individual pages can be shared across processes by copying page entries
- Segments can also be shared by sharing segment table entries:
  - Two or more processes map the shared segment on its own segment table (one segment table per process) to the same page table
  - Since there is one page table for each segment, we can share a segment by simply sharing the page table this points to

## Sharing Pages and Segments

- We already showed individual pages can be shared across processes by copying page entries
- Segments can also be shared by sharing segment table entries:
  - Two or more processes map the shared segment on its own segment table (one segment table per process) to the same page table
  - Since there is one page table for each segment, we can share a segment by simply sharing the page table this points to
- Even more flexible!

## Segmented Paging: Benefits and Costs

- Benefits:
  - Merge compiler and OS view of memory
  - Flexibility
  - No external fragmentation
  - Sharing memory between processes

## Segmented Paging: Benefits and Costs

- Benefits:
  - Merge compiler and OS view of memory
  - Flexibility
  - No external fragmentation
  - Sharing memory between processes
- Costs:
  - Slower context switches (why?)
  - Slower address translation (why?)

## Internal Fragmentation Still There…

- Paging allows getting rid of external fragmentation

# Internal Fragmentation Still There…

- Paging allows getting rid of external fragmentation
- Internal fragmentation is still an issue

# Internal Fragmentation Still There...

- Paging allows getting rid of external fragmentation
- Internal fragmentation is still an issue
- On pure paging (no segmented), assuming process' memory footprint is random, internal fragmentation amounts to 0.5 page per process (on average)

## Internal Fragmentation Still There...

- Paging allows getting rid of external fragmentation
- Internal fragmentation is still an issue
- On pure paging (no segmented), assuming process' memory footprint is random, internal fragmentation amounts to 0.5 page per process (on average)
- On segmented paging, we can lose 0.5 page per process' segment

# Internal Fragmentation Still There...

- Paging allows getting rid of external fragmentation
- Internal fragmentation is still an issue
- On pure paging (no segmented), assuming process' memory footprint is random, internal fragmentation amounts to 0.5 page per process (on average)
- On segmented paging, we can lose 0.5 page per process' segment
- The larger the page size the higher the chance of internal fragmentation

## Advanced Paging

Most modern computer systems support logical address spaces of $2^{32}$ to $2^{64}$ (i.e., 32-/64-bit machines)

## Advanced Paging

Most modern computer systems support logical address spaces of $2^{32}$ to $2^{64}$ (i.e., 32-/64-bit machines)

With a $2^{32}$ address space and 4KiB ($2^{12}$) page size, there are $2^{20}$ ~ 1 million entries in the page table

## Advanced Paging

Most modern computer systems support logical address spaces of $2^{32}$ to $2^{64}$ (i.e., 32-/64-bit machines)

With a $2^{32}$ address space and 4KiB ($2^{12}$) page size, there are $2^{20}$ ~ 1 million entries in the page table

At 4 bytes per entry, this amounts to a 4 MiB page table, which may be too large to reasonably keep in contiguous memory

# Advanced Paging

Most modern computer systems support logical address spaces of $2^{32}$ to $2^{64}$ (i.e., 32-/64-bit machines)

With a $2^{32}$ address space and 4KiB ($2^{12}$) page size, there are $2^{20}$ ~ 1 million entries in the page table

At 4 bytes per entry, this amounts to a 4 MiB page table, which may be too large to reasonably keep in contiguous memory

Note that the page table itself must be paged, and with 4KiB page size this would take $2^{10}$ = 1024 pages just to hold the page table!

## Advanced Paging

Most modern computer systems support logical address spaces of $2^{32}$ to $2^{64}$ (i.e., 32-/64-bit machines)

With a $2^{32}$ address space and 4KiB ($2^{12}$) page size, there are $2^{20}$ ~ 1 million entries in the page table

At 4 bytes per entry, this amounts to a 4 MiB page table, which may be too large to reasonably keep in contiguous memory

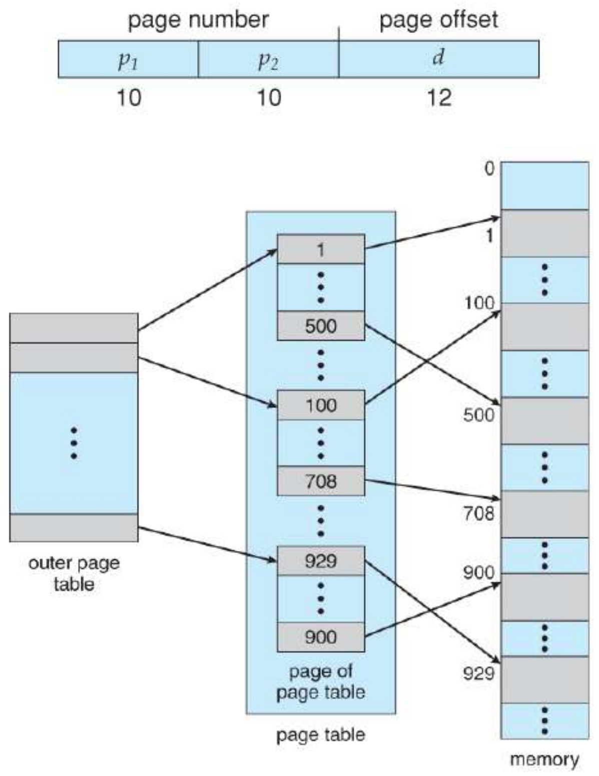
Note that the page table itself must be paged, and with 4KiB page size this would take $2^{10}$ = 1024 pages just to hold the page table!

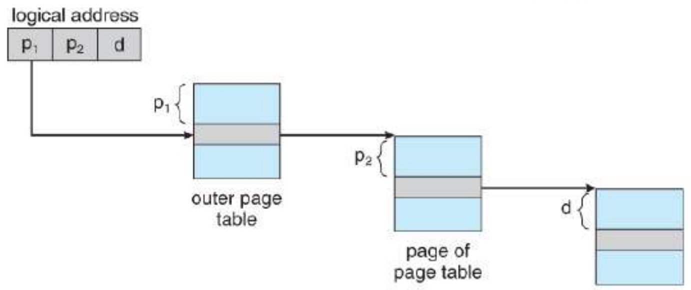
$\Downarrow$

**More advanced paging structures are needed!**

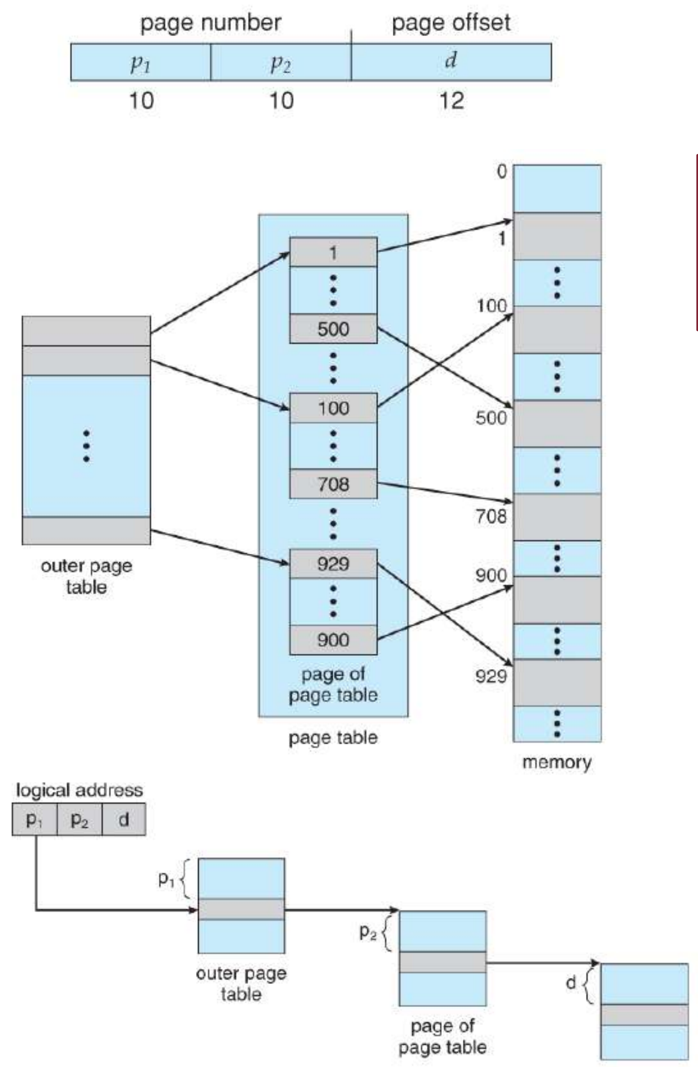
# Advanced Paging: Two-Tier Page Table

Let's page the page table!





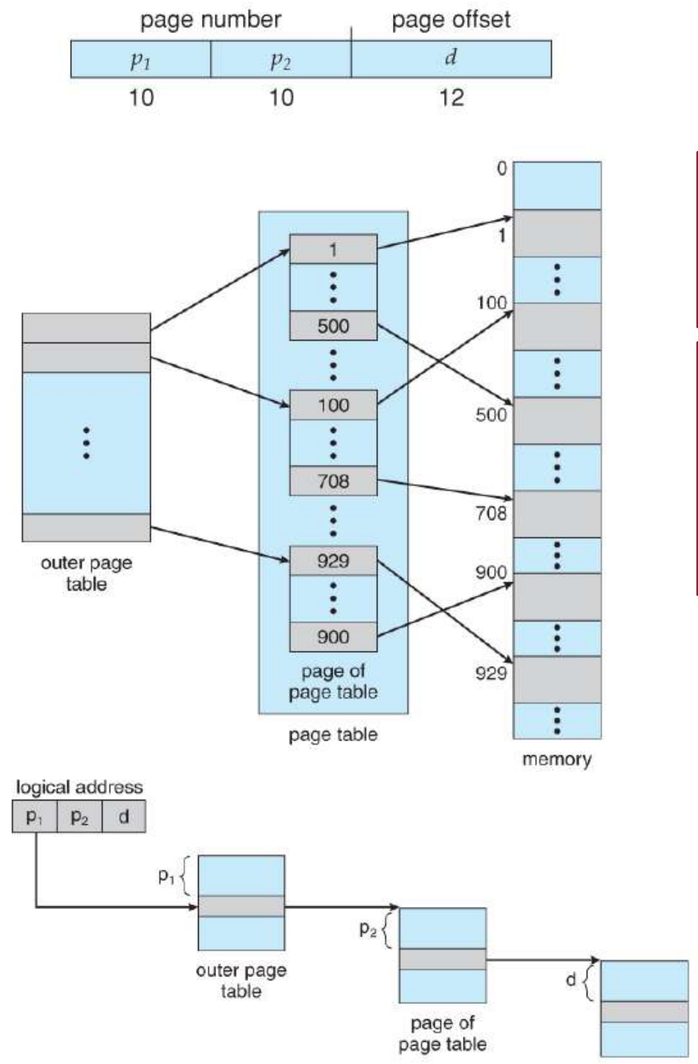
# Advanced Paging: Two-Tier Page Table



**Let's page the page table!**

20-bit page number broken into 2 10-bit page numbers

# Advanced Paging: Two-Tier Page Table

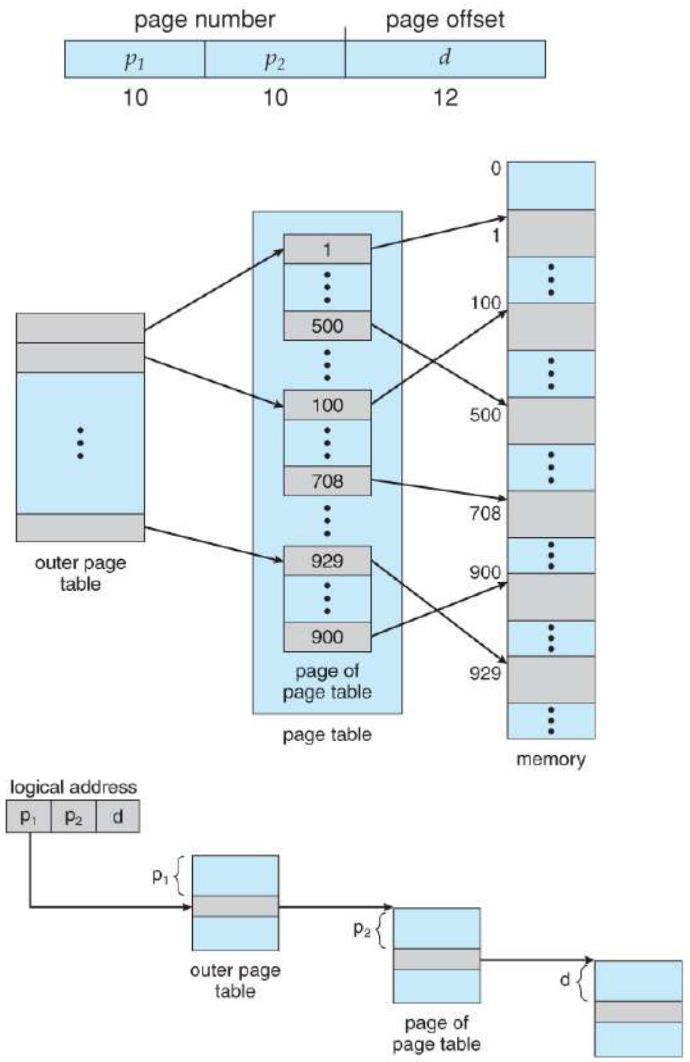


**Let's page the page table!**

20-bit page number broken into 2 10-bit page numbers

The most significant 10 bits represent an entry in the outer page table, used to find one page of the inner page table

# Advanced Paging: Two-Tier Page Table



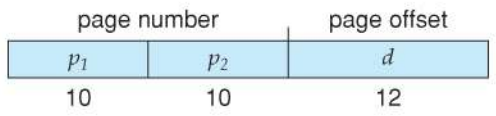
**Let's page the page table!**

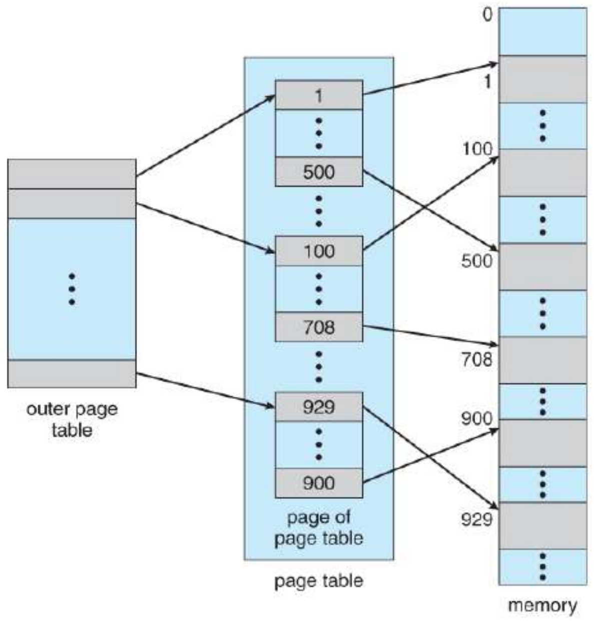
20-bit page number broken into 2 10-bit page numbers

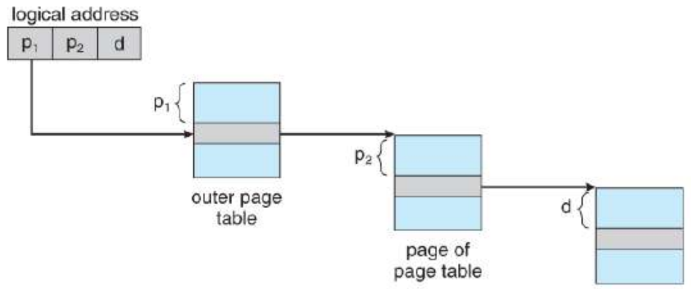
The most significant 10 bits represent an entry in the outer page table, used to find one page of the inner page table

The second 10 bits finds a specific entry in that inner page table, which maps to a frame in physical memory

# Advanced Paging: Two-Tier Page Table







**Let's page the page table!**

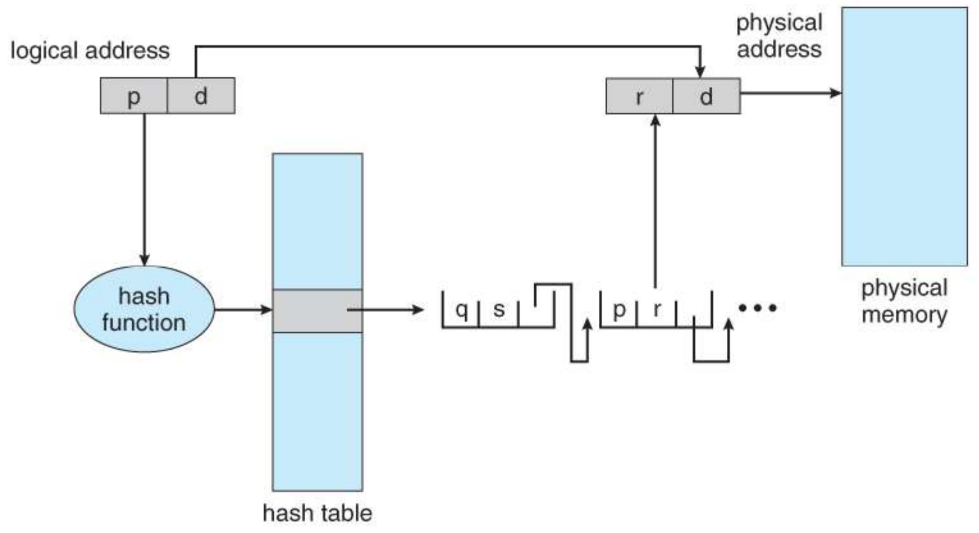
20-bit page number broken into 2 10-bit page numbers

The most significant 10 bits represent an entry in the outer page table, used to find one page of the inner page table

The second 10 bits finds a specific entry in that inner page table, which maps to a frame in physical memory

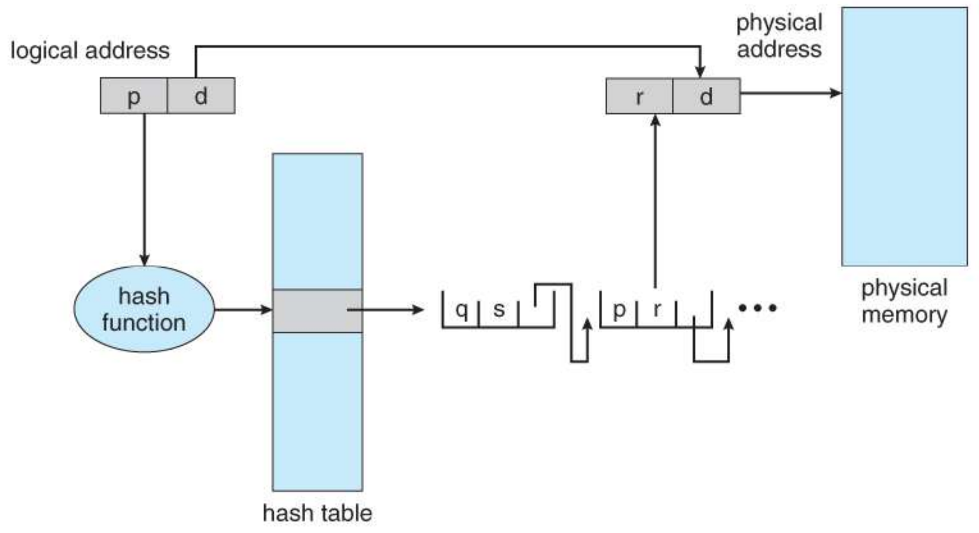
The remaining 12 bits of the 32-bit logical address are still the offset within the 4KiB frame

# Advanced Paging: Hashed Page Table



Use hash tables to store highly sparse page tables

# Advanced Paging: Hashed Page Table



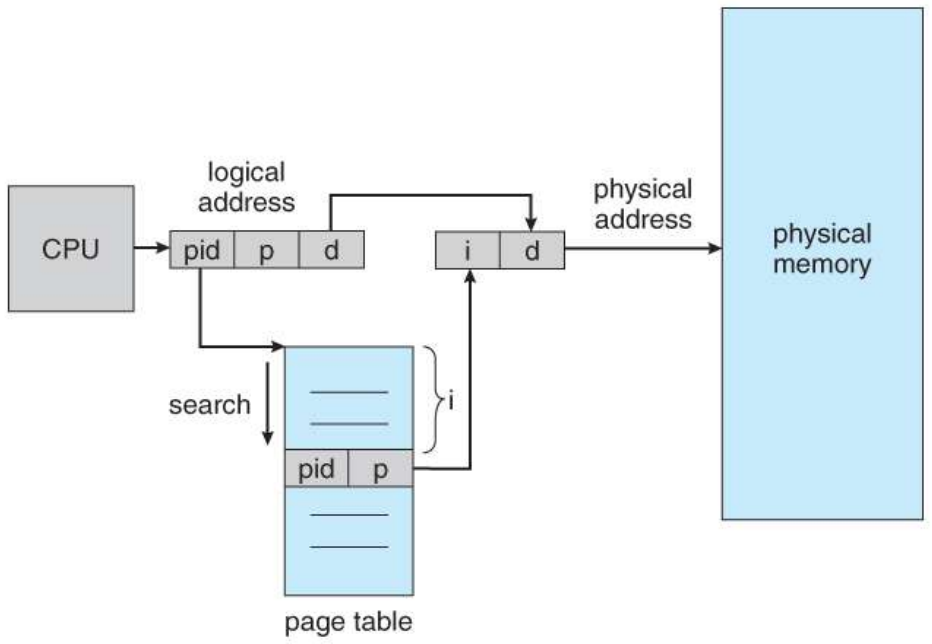
Use hash tables to store highly sparse page tables

Indexing via hash function rather than integers

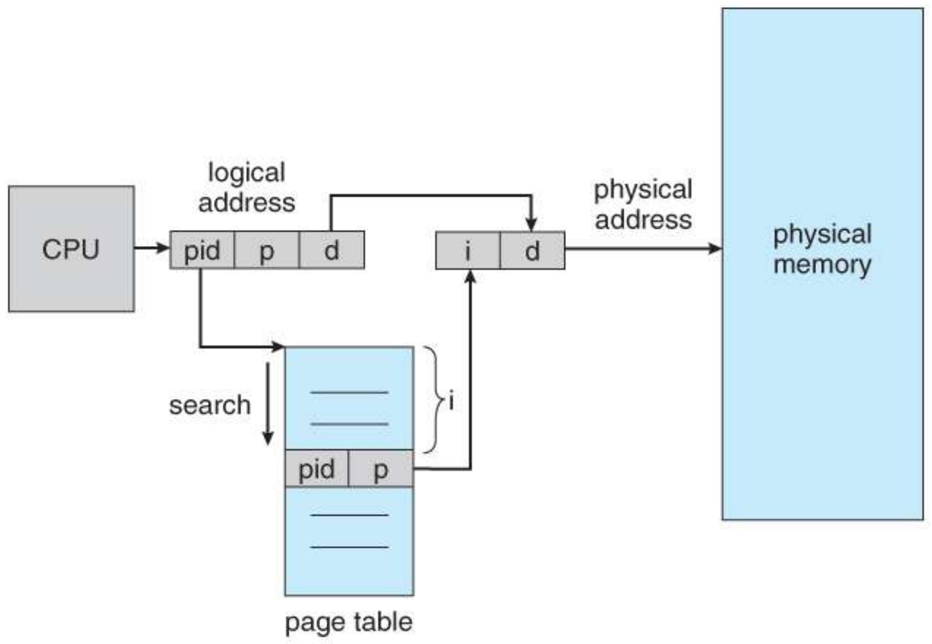
# Advanced Paging: Inverted Page Table

An inverted page table lists all of the frames currently loaded in memory, for all processes

Instead of a table listing all of the pages for a particular process



# Advanced Paging: Inverted Page Table



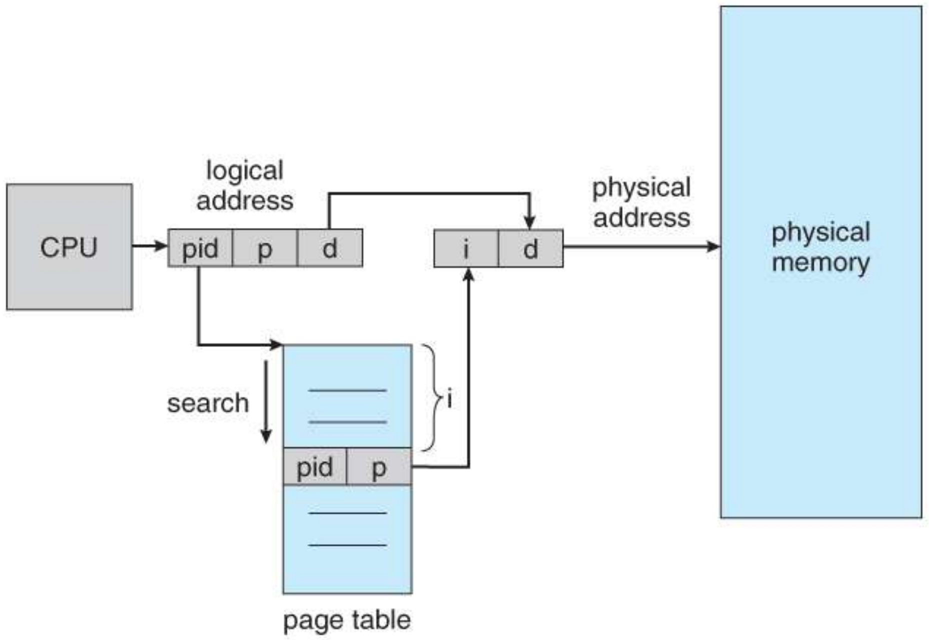
An inverted page table lists all of the frames currently loaded in memory, for all processes

Instead of a table listing all of the pages for a particular process

Access to an inverted page table can be slow (linear search)

Hashing the table speeds up the search process

## Advanced Paging: Inverted Page Table



An inverted page table lists all of the frames currently loaded in memory, for all processes

Instead of a table listing all of the pages for a particular process

Access to an inverted page table can be slow (linear search)

Hashing the table speeds up the search process

Inverted page tables do not easily allow mapping multiple logical pages to a common physical frame (page sharing)

Each frame is mapped to exactly one process

## Summary

- **Relocation** using base and limit registers
  - Simple yet inflexible

## Summary

- Relocation using base and limit registers
    - Simple yet inflexible
- Segmentation
    - Compiler's logical view of memory presented to the OS
    - Segment tables tend to be small enough to be stored in registers
    - Contiguous memory allocation is expensive and complicated (first-fit, best-fit, or worst-fit)
    - Compaction is needed to solve external fragmentation

## Summary

- Paging
  - Simplifies memory allocation by relaxing contiguous assumption
  - Each logical page can be allocated to any physical frame
  - Page tables can be extremely large

# Summary

- Segmentation + Paging
  - Only need to allocate as many page table entries an needed
  - Sharing either at the segment or at the page level
  - Might increase internal fragmentation over pure paging
  - 2 lookups per memory reference are needed