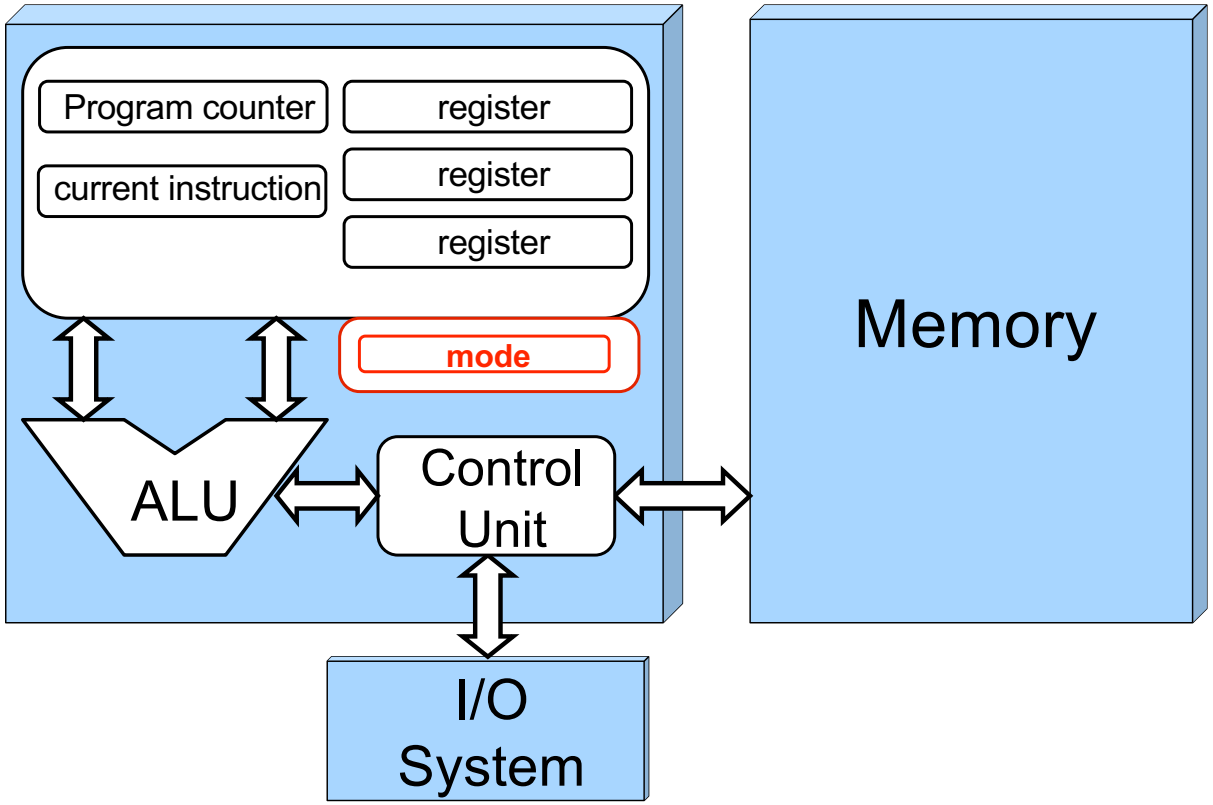
Main Memory

Operating Systems
Wenbo Shen

Acknowledgement: some slides are from Yajin Zhou@ZJU and Xiaosong Ma from NCSU.

Computer Architecture



Outline

- Memory allocation evolution
  - How to move a process
  - Partition
- Memory partition – fixed and variable
  - first, best, worst fit
  - fragmentation: internal/external
- Segmentation
  - Logical address vs physical address
- MMU: address translation + protection

Background

- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage that CPU can access directly
- Memory unit only sees a stream of:
  - addresses + read requests, or
  - addresses + data and write requests
- Register access is done in one CPU clock (or less)
- Main memory can take many cycles, causing a **stall**
- **Cache** sits between main memory and CPU registers
- **Protection** of memory is required to ensure correct operation
  - hardware vs software

Conceptual View of Memory

address	content
0000 0000 0000 0000	0110 1110
0000 0000 0000 0001	1111 0100
0000 0000 0000 0010	0000 0000
0000 0000 0000 0011	0000 0000
0000 0000 0000 0100	0101 1110
000(	
000(	
0000 0000 0000 0111	0100 0000
0000 0000 0000 1000	1111 0101
...	...

At address 0000 0000 0000 0010 the content is 0000 0000

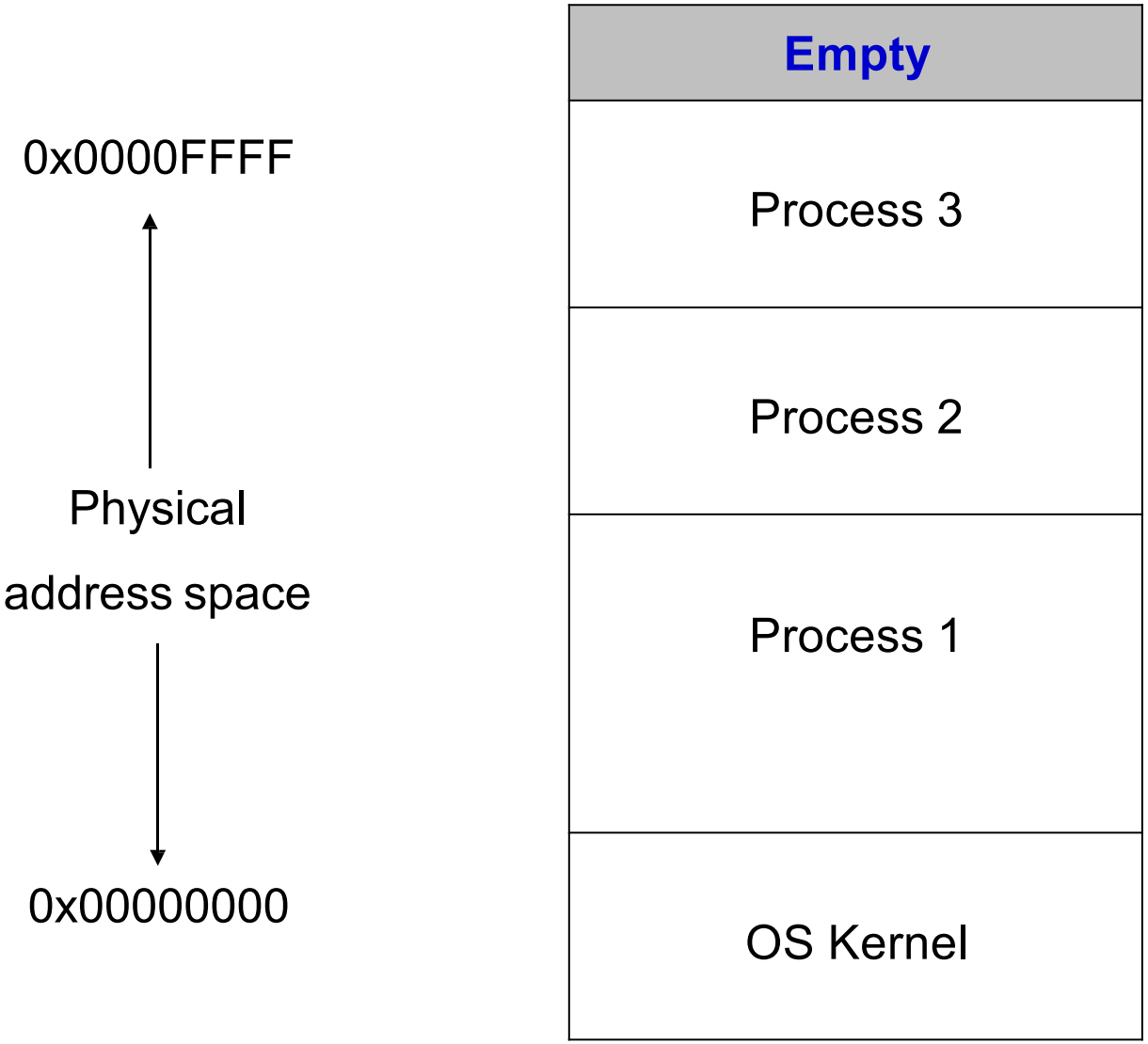
In the Beginning (prehistory) …

- Batch systems
  - One program loaded in physical memory
  - Runs to completion
- What if the job is larger than physical memory?
  - Divide and conquer
  - Identify sections of program that
    - Can run to generate result
    - Can fit into available memory
  - Add statement after result to load new section
  - Like passes in compiler

In the Beginning (multi-programming) …

- Multiple processes in physical memory at the same time
  - Allows fast switching to a ready process
  - Divide physical memory into multiple pieces – partitioning
- Partition requirements
  - Protection – keep processes from smashing each other
  - Fast execution – memory accesses can't be slowed by protection mechanisms
  - Fast context switch – can't take forever to setup mapping of addresses

Physical Memory

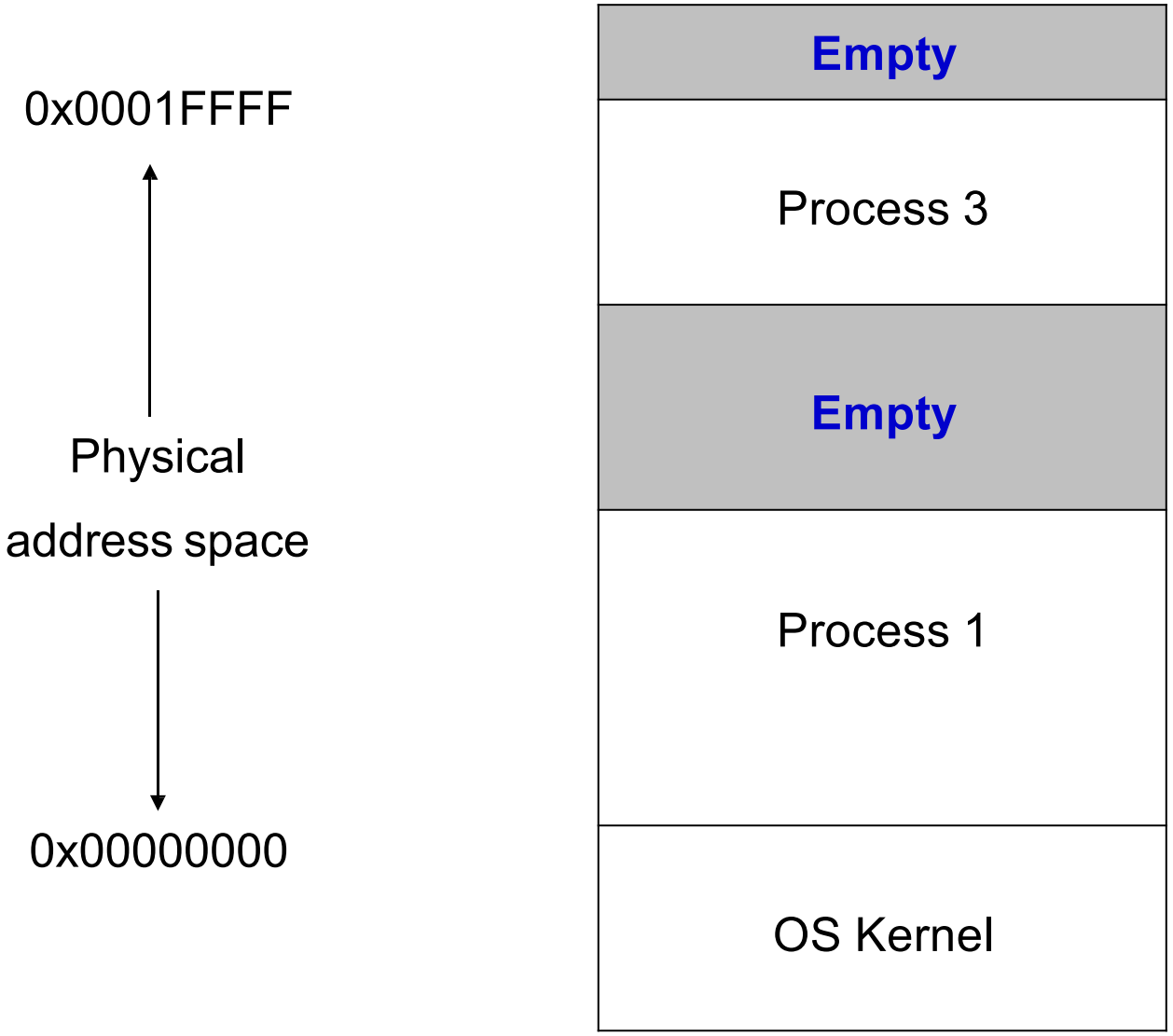


Loading a Process

- Relocate all addresses relative to start of partition

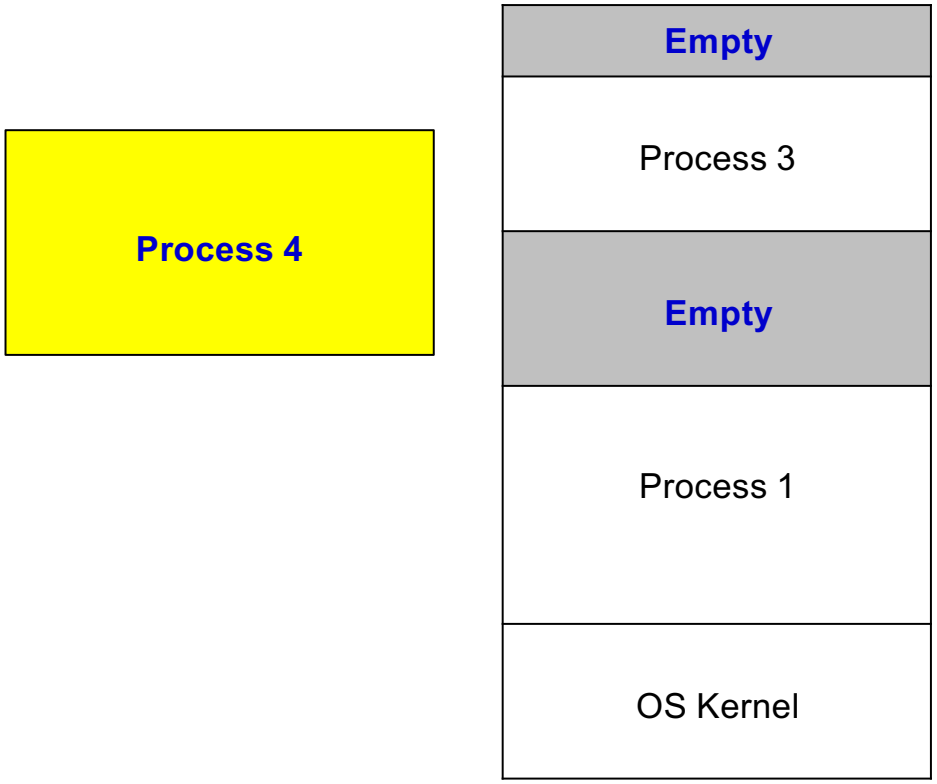
- Memory protection assigned by OS
  - Block-by-block to physical memory
- Once process starts
  - Partition cannot be moved in memory
  - Why?

Physical Memory



Problem

- What happens when Process 4 comes along and requires space larger than the largest empty partition?
  - Wait
  - Potential starvation



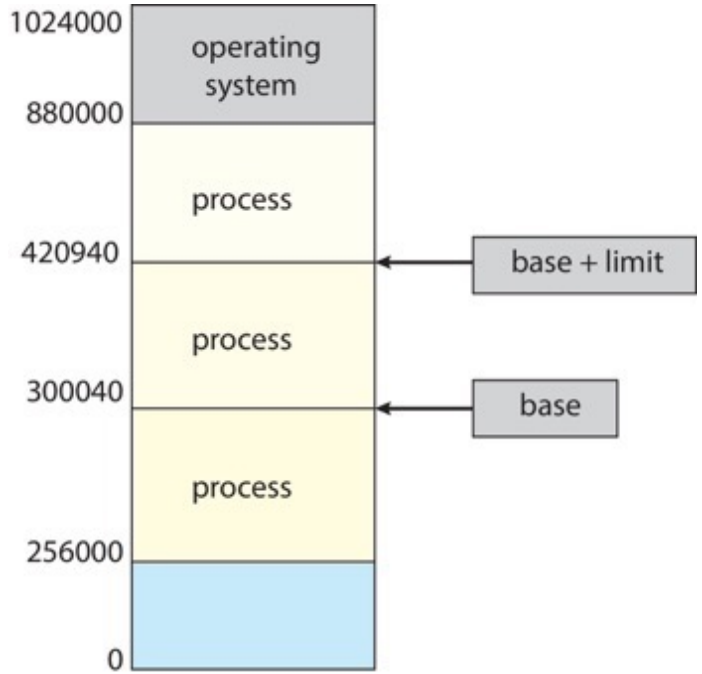
- Solution
  - Move process 3
  - Hard to move
    - Why?

Solution

- Logical Address
  - Instead of physical address
  - Used by the process
  - Translated into physical address at runtime
- E.g., when program utters `0x00346`
  - Machine accesses `0x14346` instead
- How to move process now?

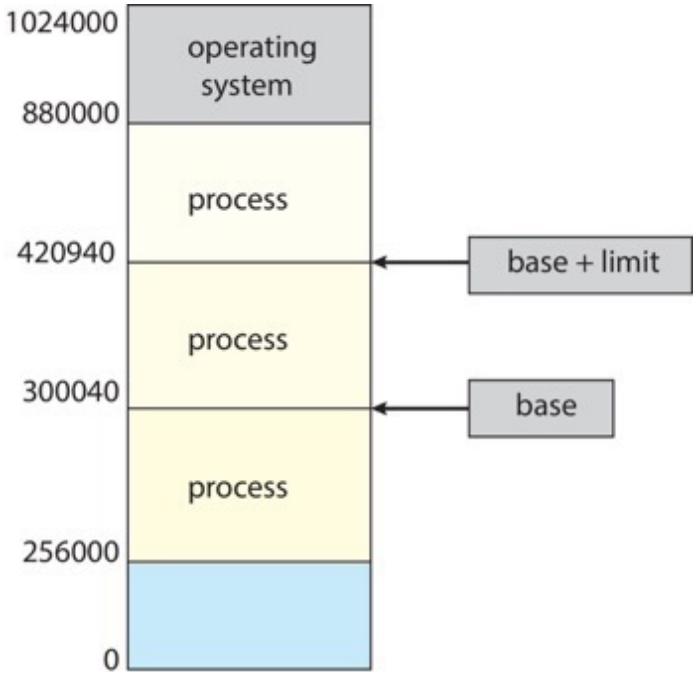
Simplest Implementation - Partition

- Base and Limit registers
  - Base added to all addresses
  - Limit checked on all memory references
  - Loaded by OS at each context switch
    - Why?



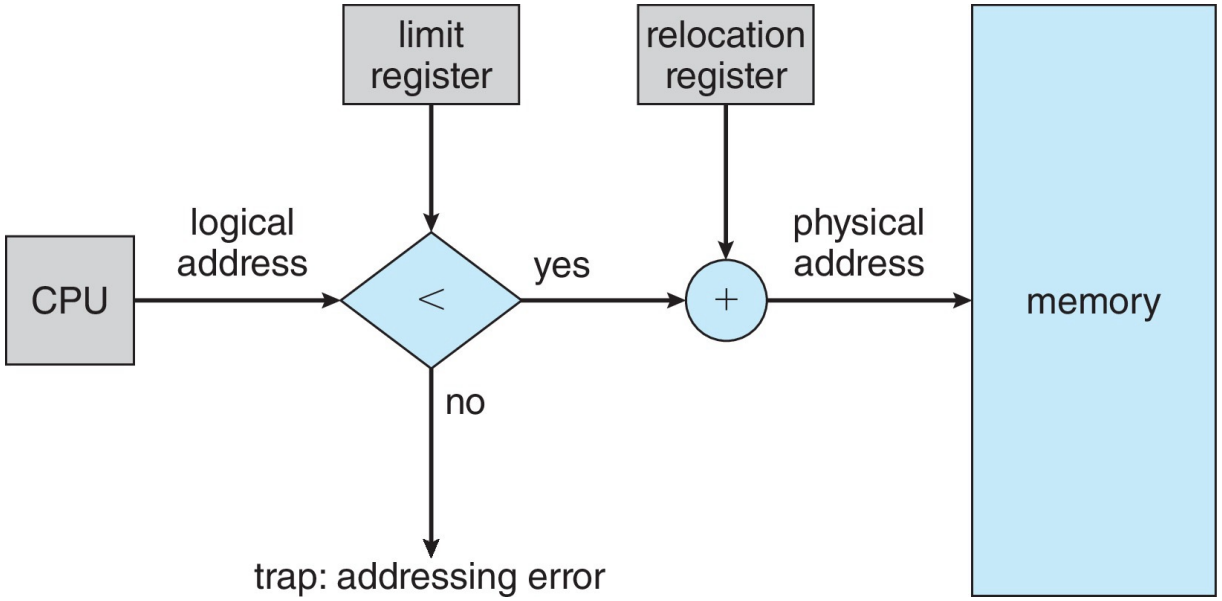
Protection

- Need to ensure that a process can access and only access those addresses in its address space.
- We can provide this protection by using a pair of **base** and **limit** registers define the logical address space of a process



Hardware Address Protection

- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user
- The instructions that load the base and limit registers are privileged

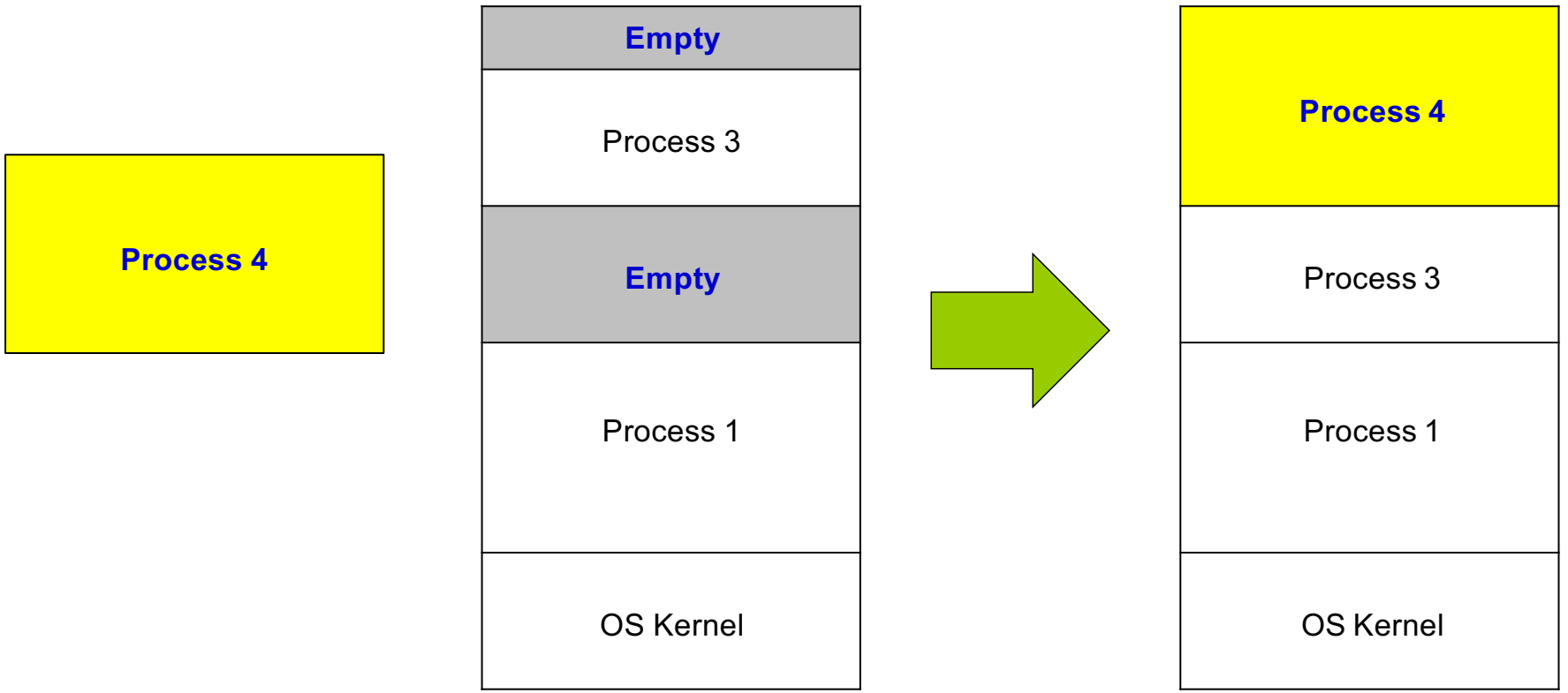


Advantages

- Built-in protection provided by Limit
  - No physical protection per page or block
- Fast execution
  - Addition and limit check at hardware speeds within each instruction
- Fast context switch
  - Need only change base and limit registers
- No relocation of program addresses at load time
  - All addresses relative to zero
- Partition can be suspended and moved at any time
  - Process is unaware of change
  - Expensive for large processes

Simplest Implementation - Partition

- Copy process 3 to the empty slots and update its base and limit



Memory Allocation Strategies

- Fixed partitions
- Variable partitions

Partitioning Strategies – Fixed

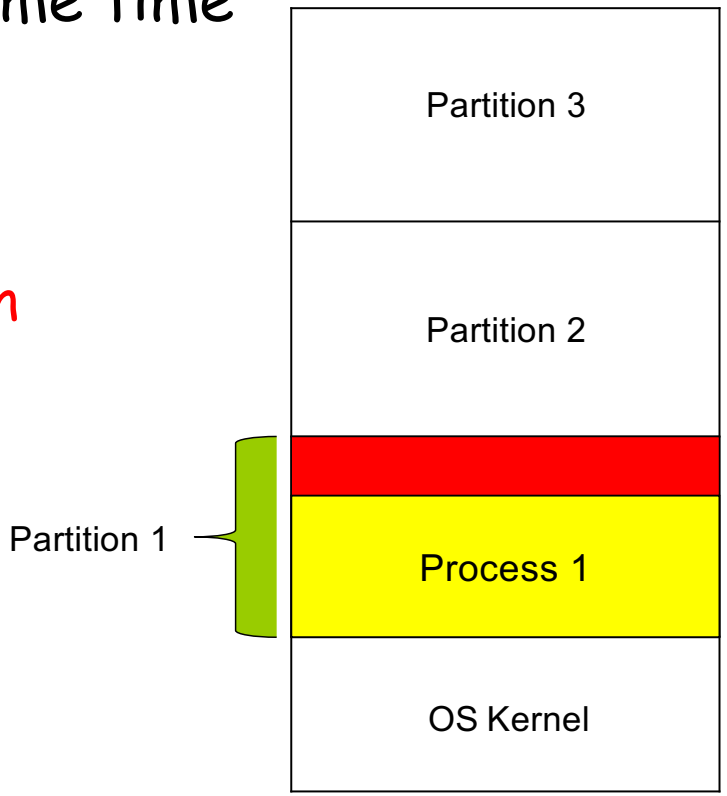
- Fixed Partitions – divide memory into equal sized pieces (except for OS)
  - Degree of multiprogramming = number of partitions
  - Simple policy to implement
    - All processes must fit into partition space
    - Find any free partition and load process
- Problem – Internal Fragmentation
  - Unused memory in partition not available to other processes

Question:
What is the "right" partition size?

Partitioning Strategies - Fixed

- Same size
- Support 3 processes at the same time

- Internal Fragmentation
  - Unused memory within partition
  - Big waste of memory



Internal fragmentation

- Memory allocated may be larger than the requested size
  - This size difference is memory internal to a partition, but not being used
  - Sophisticated algorithms are designed to avoid fragmentation

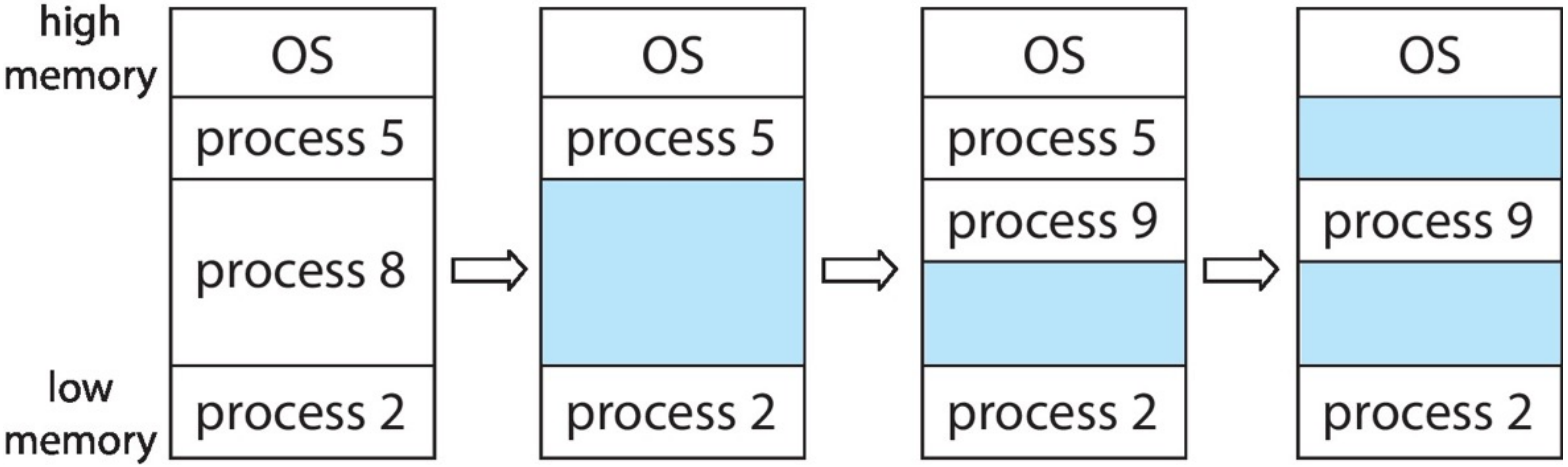
Partitioning Strategies – Variable

- Memory is dynamically divided into partitions based on process needs
  - More complex management problem
    - Need data structures to track free and used memory
    - New process allocated memory from hole large enough to fit it
- Problem – External Fragmentation
  - Unused memory between partitions too small to be used by any processes

Partitioning Strategies – Variable

- Multiple-partition allocation
  - Degree of multiprogramming limited by number of partitions
  - **Variable-partition** sizes for efficiency (sized to a given process' needs)
  - **Hole** – block of available memory; holes of various size are scattered throughout memory

- Allocation process
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it
  - Process exiting frees its partition, adjacent free partitions combined
  - Operating system maintains information about:
    - a) allocated partitions b) free partitions (hole)

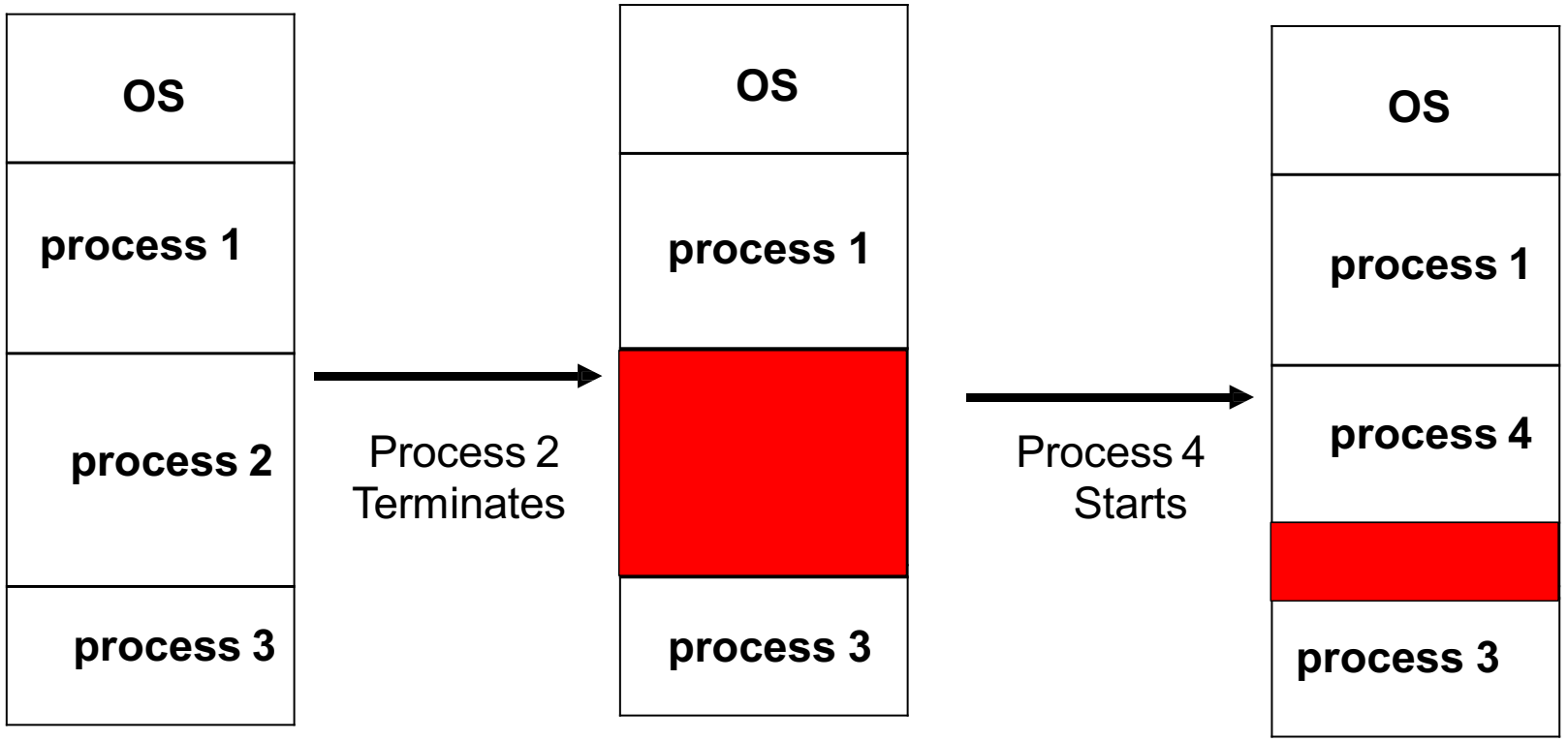


Partitioning Strategies – Variable

- How to satisfy a request of size *n* from a list of free memory blocks?
  - **first-fit**: allocate from the first block that is big enough
  - **best-fit**: allocate from the smallest block that is big enough
    - must search entire list, unless ordered by size
    - produces the smallest leftover hole
  - **worst-fit**: allocate from the largest hole
    - must also search entire list
    - produces the largest leftover hole
- **Fragmentation** is big problem for all three methods
  - first-fit and best-fit usually perform better than worst-fit

Partitioning Strategies – Variable

- External Fragmentation



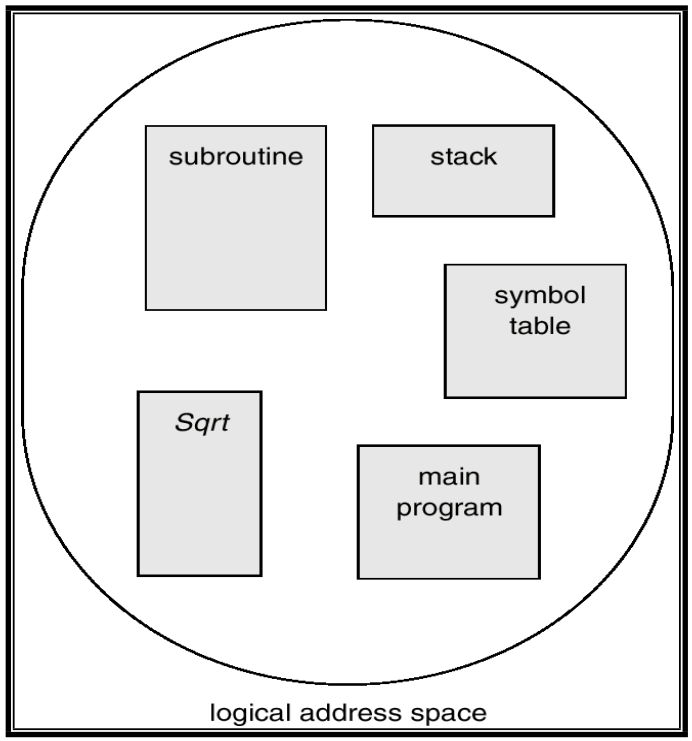
External fragmentation

- Unusable memory between allocated memory blocks
  - **total amount** of free memory space **is larger than a request**
  - the request cannot be fulfilled because the free memory is not contiguous
- External fragmentation can be reduced by **compaction**
  - shuffle memory contents to place all free memory in one large block
  - program needs to be **relocatable** at runtime
  - performance overhead, timing to do this operation

Segmentation

How about with a program?

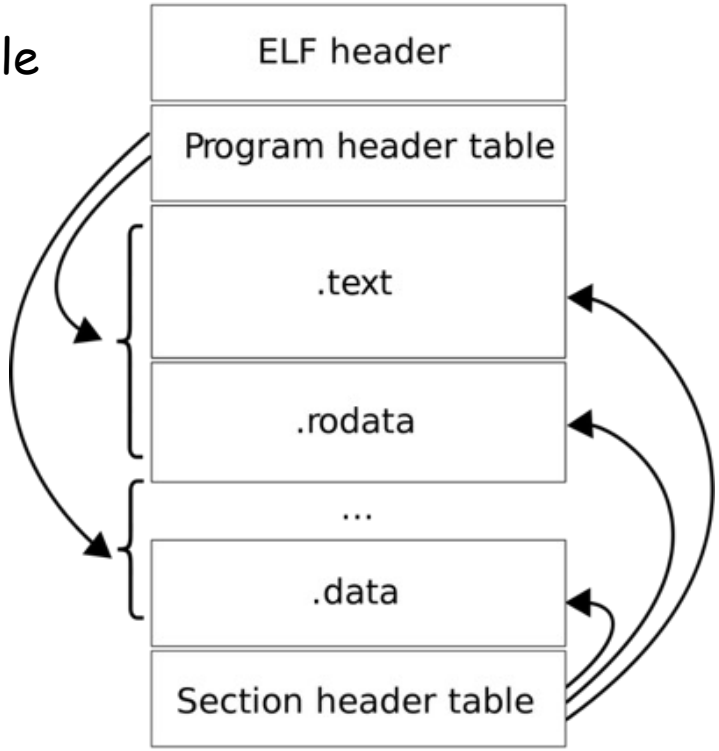
- User's View of a Program



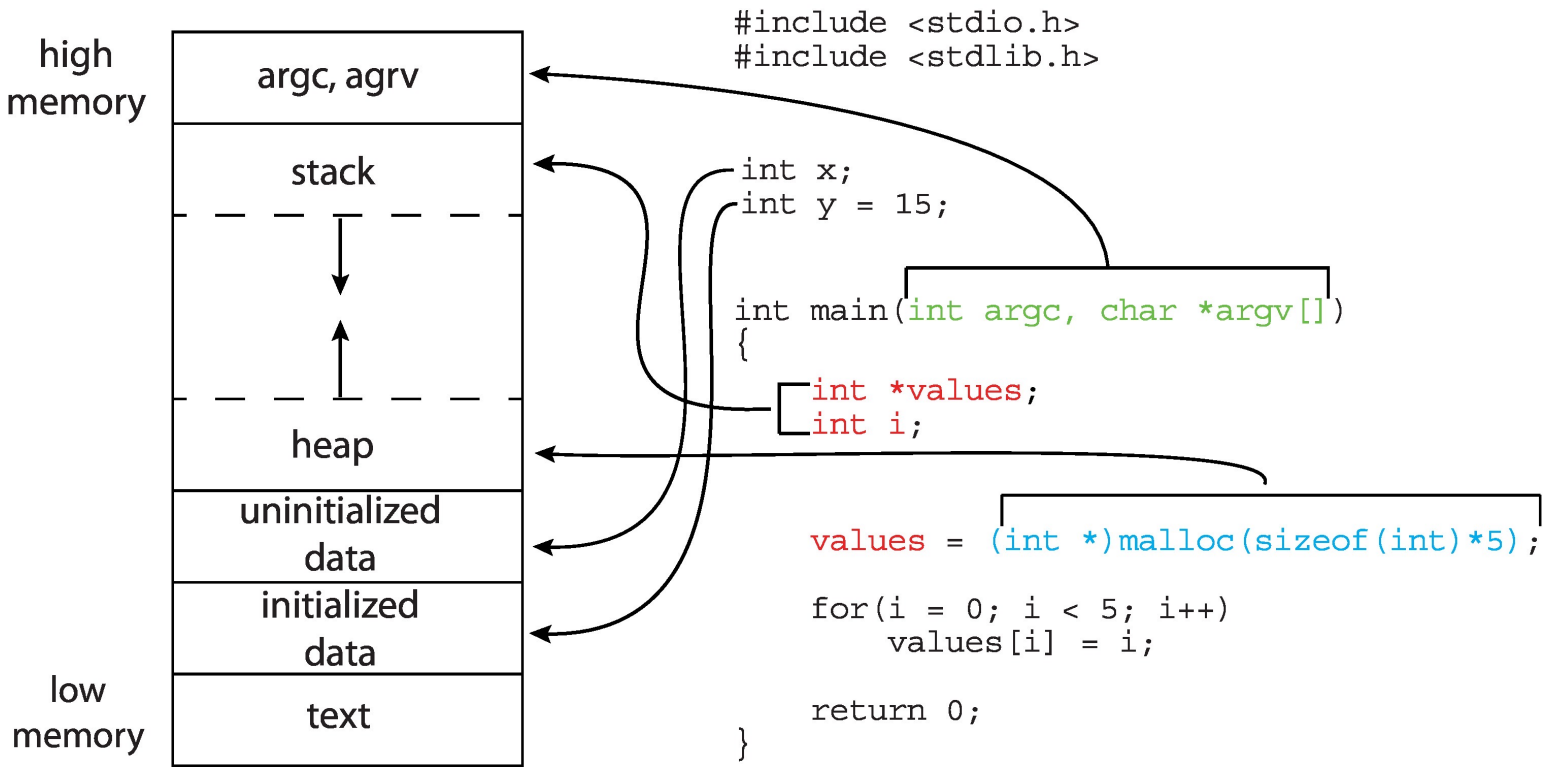
Question:
Can we have multiple sets of "base and limit" registers?

ELF binary basics

- What's main (a.out)
  - Executable and Linkable Format - ELF
  - Program header table and section header table
    - For Linker and Loader
  - .text: code
  - .rodata: initialized read-only data
  - .data: initialized data
  - .bss: uninitialized data



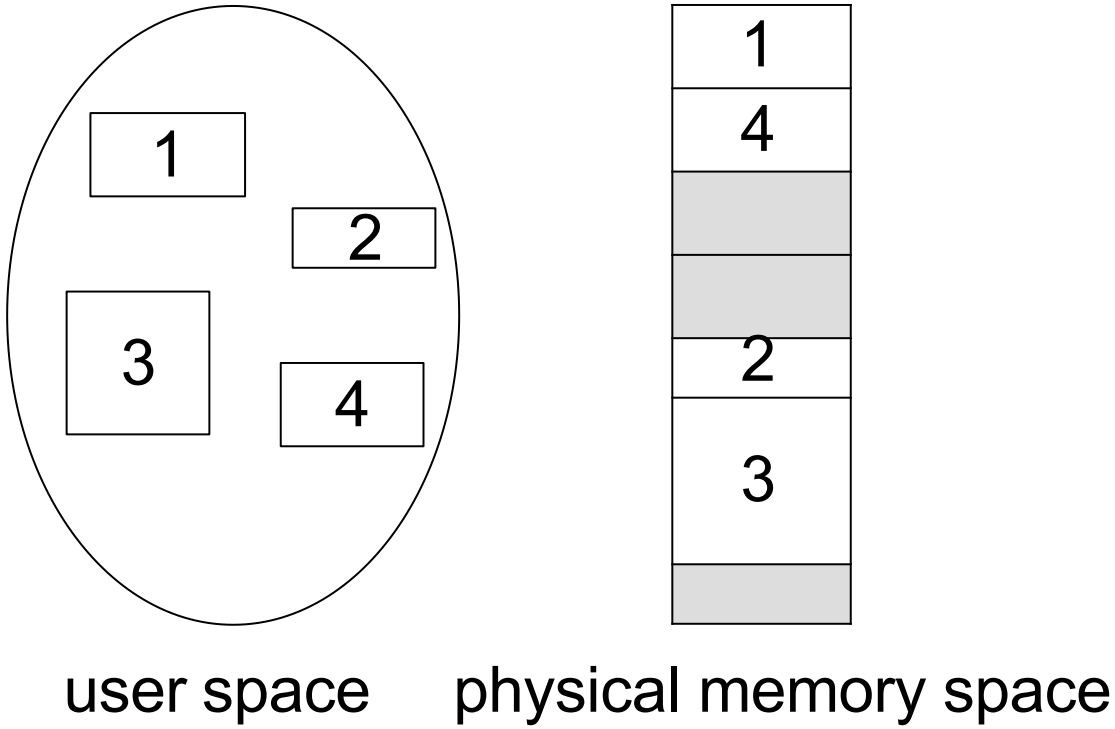
Memory Layout of a C Program



Process Address Space

```
wenbo@parallels: ~
wenbo@parallels: ~ 107x30
7ffc75a5f000-7ffc75a80000 rw-p 00000000 00:00 0                          [stack]
7ffc75aa7000-7ffc75aaa000 r--p 00000000 00:00 0                          [vvar]
7ffc75aaa000-7ffc75aac000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
wenbo@parallels:~$ which cat
/bin/cat
wenbo@parallels:~$ file /bin/cat
/bin/cat: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/l,
 for GNU/Linux 3.2.0, BuildID[sha1]=747e524bc20d33ce25ed4aea108e3025e5c3b78f, stripped
wenbo@parallels:~$ cat /proc/self/maps
55b793b79000-55b793b81000 r-xp 00000000 08:01 1048601                    /bin/cat
55b793d80000-55b793d81000 r--p 00007000 08:01 1048601                    /bin/cat
55b793d81000-55b793d82000 rw-p 00008000 08:01 1048601                    /bin/cat
55b794d33000-55b794d54000 rw-p 00000000 00:00 0                          [heap]
7f1974b90000-7f197555f000 r--p 00000000 08:01 662494                     /usr/lib/locale/locale-archive
7f197555f000-7f1975746000 r-xp 00000000 08:01 267596                     /lib/x86_64-linux-gnu/libc-2.27.so
7f1975746000-7f1975946000 ---p 001e7000 08:01 267596                     /lib/x86_64-linux-gnu/libc-2.27.so
7f1975946000-7f197594a000 r--p 001e7000 08:01 267596                     /lib/x86_64-linux-gnu/libc-2.27.so
7f197594a000-7f197594c000 rw-p 001eb000 08:01 267596                     /lib/x86_64-linux-gnu/libc-2.27.so
7f197594c000-7f1975950000 rw-p 00000000 00:00 0
7f1975950000-7f1975977000 r-xp 00000000 08:01 267568                     /lib/x86_64-linux-gnu/ld-2.27.so
7f1975b3c000-7f1975b60000 rw-p 00000000 00:00 0
7f1975b77000-7f1975b78000 r--p 00027000 08:01 267568                     /lib/x86_64-linux-gnu/ld-2.27.so
7f1975b78000-7f1975b79000 rw-p 00028000 08:01 267568                     /lib/x86_64-linux-gnu/ld-2.27.so
7f1975b79000-7f1975b7a000 rw-p 00000000 00:00 0
7ffc73010000-7ffc73031000 rw-p 00000000 00:00 0                          [stack]
7ffc73148000-7ffc7314b000 r--p 00000000 00:00 0                          [vvar]
7ffc7314b000-7ffc7314d000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
wenbo@parallels:~$
```

Logical View of Segmentation

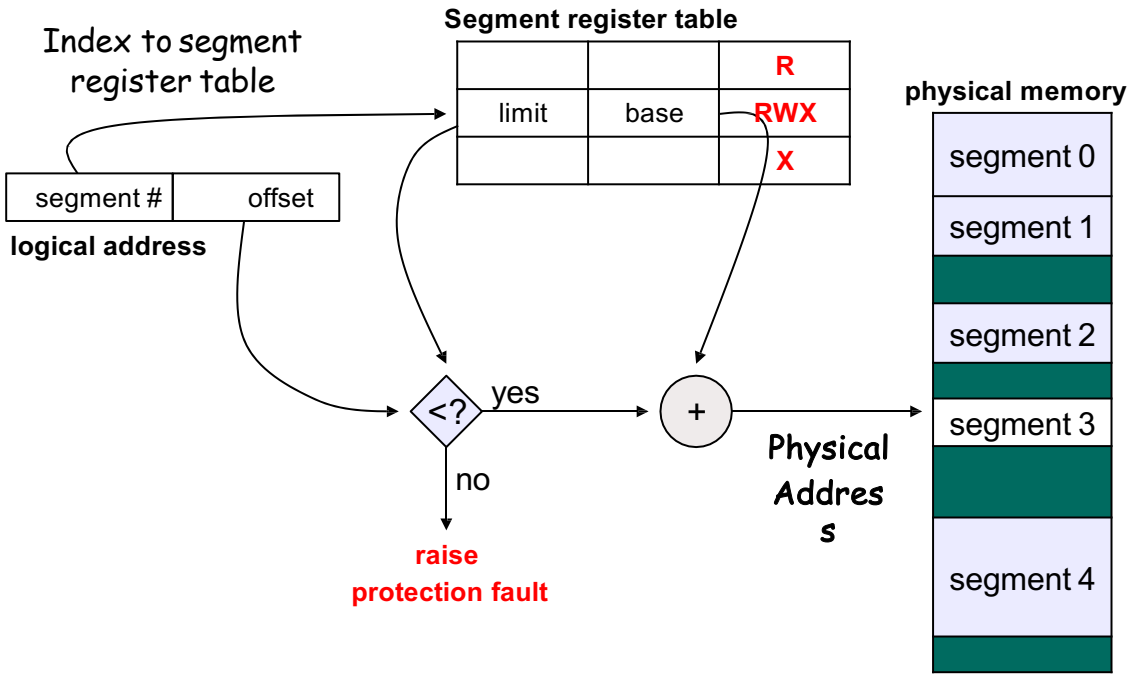


Segmentation

- Logical address consists of a pair:
  - <segment-number, offset>
  - Offset is the address offset within the segment

- Segment table where each entry has:
  - Base: starting physical address
  - Limit: length of segment

Segment Lookup



Segmentation

- Common in early minicomputers
  - Small amount of additional hardware
  - segments
  - Used effectively in Unix
- Good idea that has persisted and supported in current hardware and OSs
  - X86 supports segments
  - Linux supports segments

Question:

Do we still have external fragmentation problem? If yes, can we further improve it?

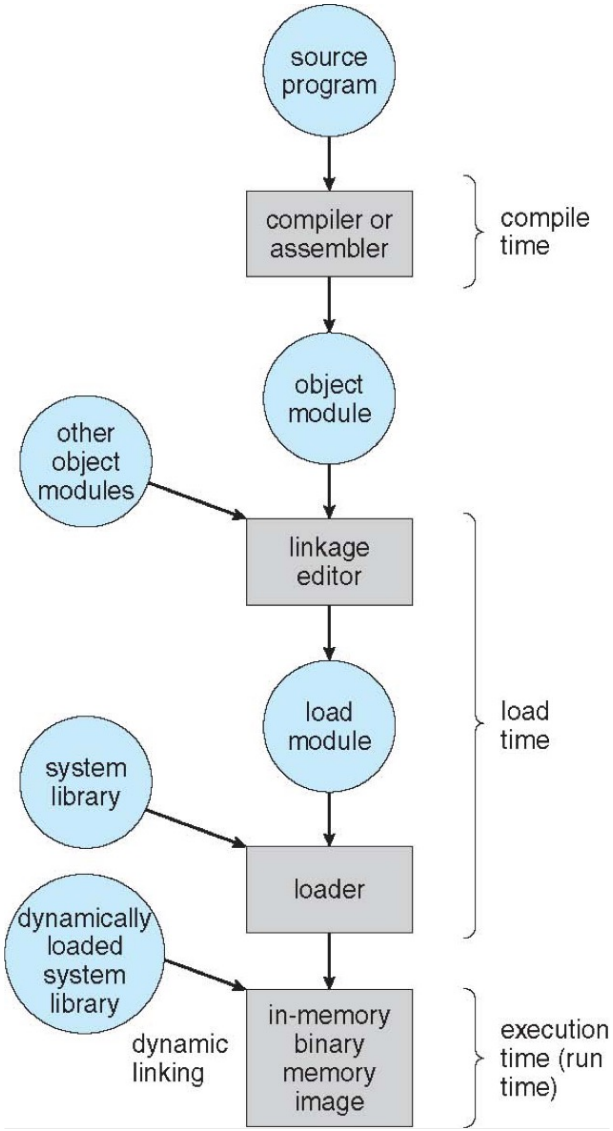
Address Binding

- Addresses are represented in different ways at different stages of a program's life
  - **source code** addresses are usually **symbolic** (e.g., variable name)
  - **compiler** binds symbols to **relocatable addresses**
    - e.g., "14 bytes from beginning of this module"
  - **linker** (or loader) binds relocatable addresses to **absolute addresses**
    - e.g., 0x0e74014
  - Each binding maps one address space to another

Binding of Instructions and Data to Memory

- Address binding of instructions and data to memory addresses can happen at three different stages
  - **Compile time**: If memory location known a priori, **absolute code** can be generated; must recompile code if starting location changes
  - **Load time**: Must generate relocatable code if memory location is not known at compile time
  - **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another
    - Need **hardware support** for address maps (e.g., base and limit registers)

Multi-step Processing of a User Program

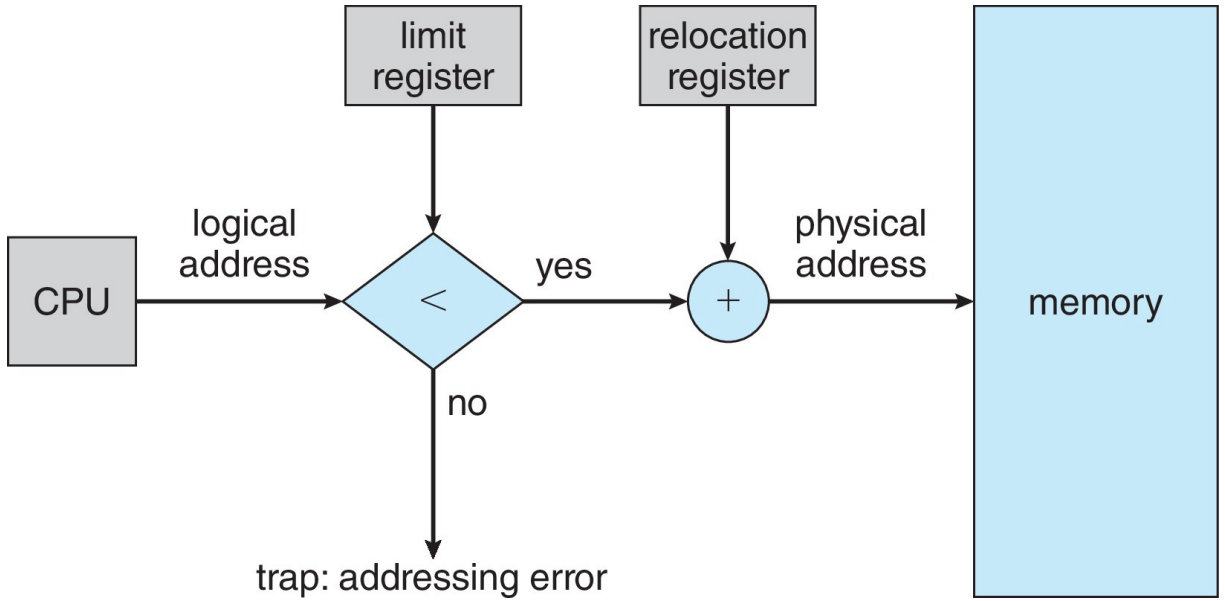


Logical vs. Physical Address

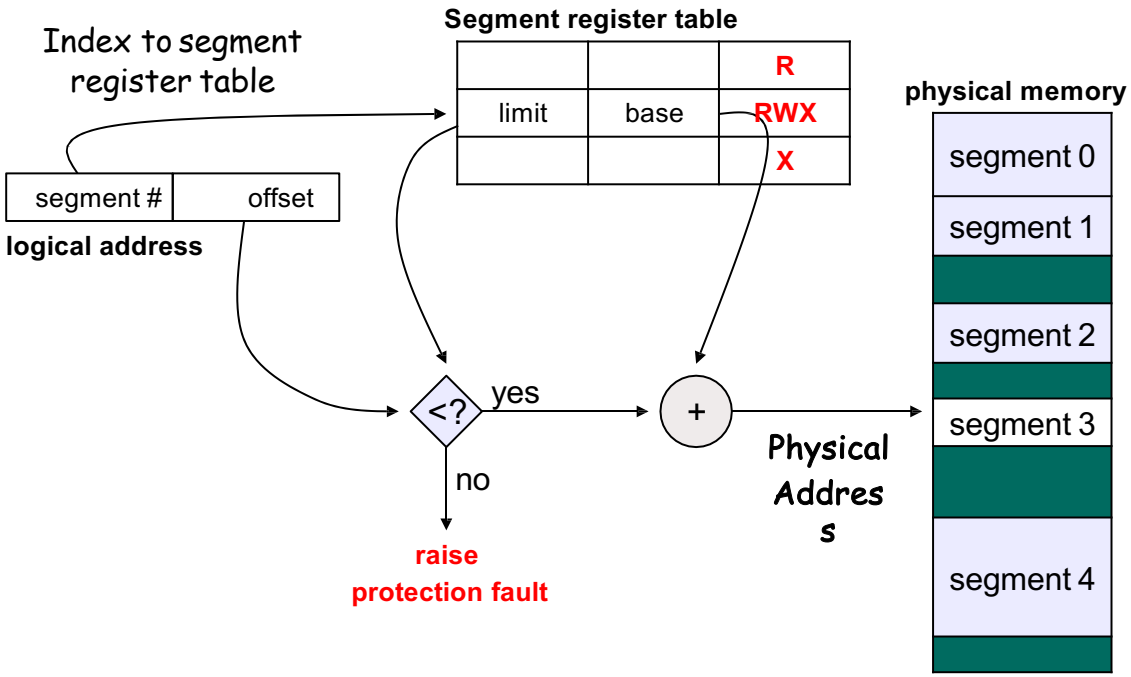
- The concept of a logical address space that is bound to a **separate physical address** space is central to proper memory management
  - **Logical address** – generated by the CPU; also referred to as virtual address
  - **Physical address** – address seen by the memory unit
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme
- **Logical address space** is the set of all logical addresses generated by a program
- **Physical address space** is the set of all physical addresses generated by a program

Logical vs. Physical Address

- What are the logical and physical address

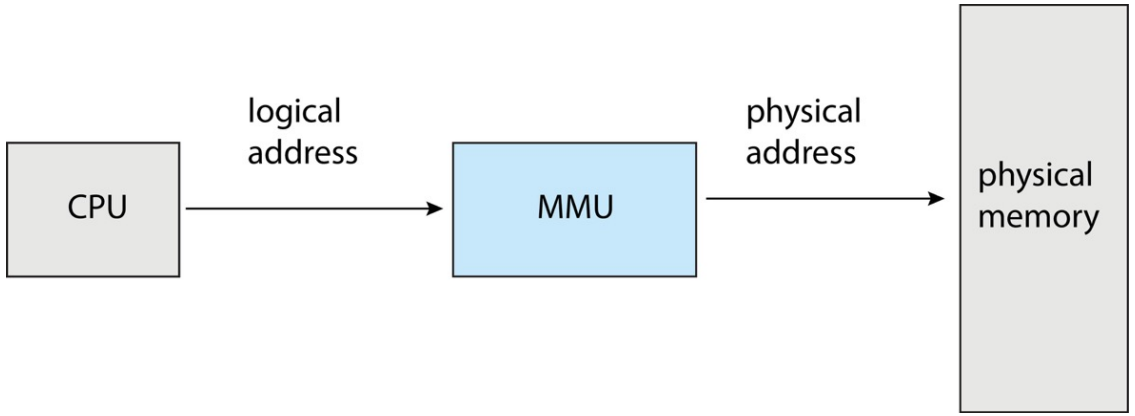


Review-Segment Lookup



Memory-Management Unit (MMU)

- Hardware device that at run time maps logical to physical address
  - CPU uses logical addresses
  - Memory unit uses physical address
  - Like speaking "different languages", MMU does the translation

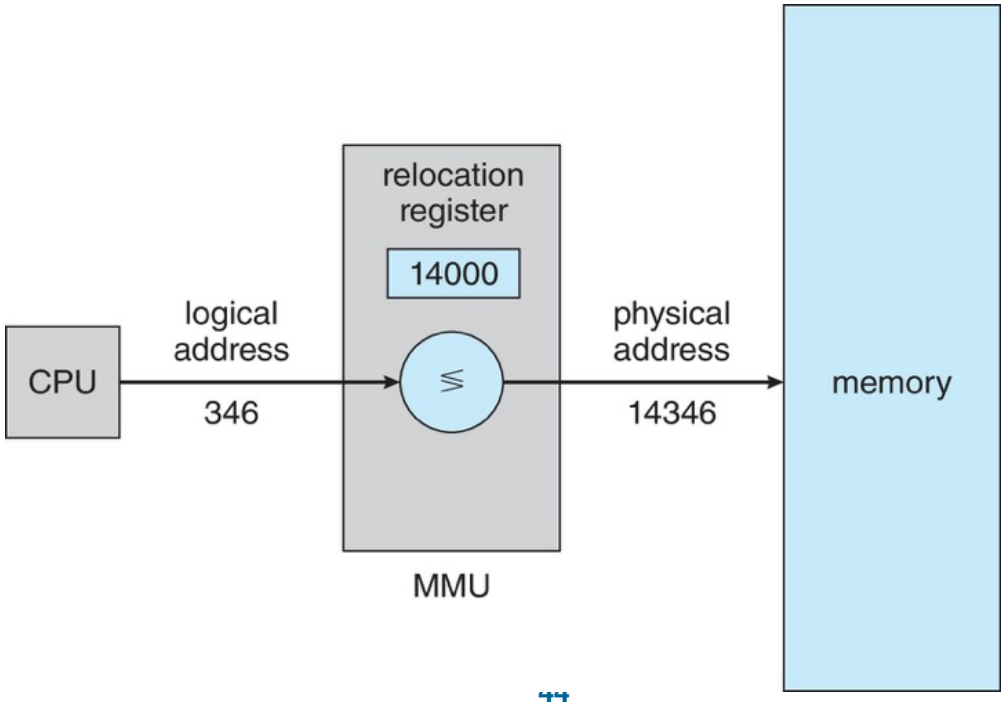


Memory-Management Unit

- Consider a simple scheme which is a generalization of the base-register scheme.
  - The base register now called **relocation register**
  - The value in the relocation register is added to every address generated by a user process at the time it is sent to memory
  - The user program deals with logical addresses; it never sees the real physical addresses
    - Execution-time binding occurs when reference is made to location in memory
    - Logical address bound to physical addresses

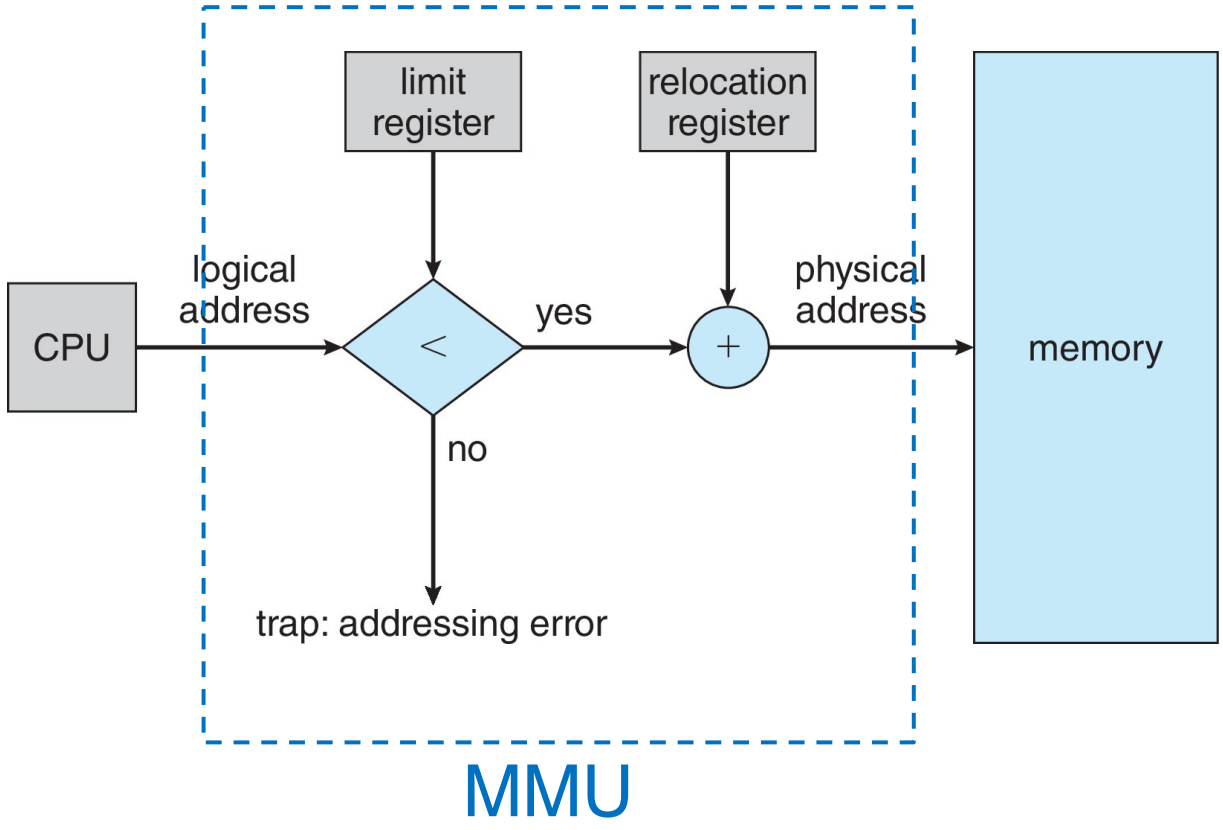
Memory-Management Unit

- Consider simple scheme. which is a generalization of the base-register scheme.
- The base register now called **relocation register**
- The value in the relocation register is added to every address generated by a user process at the time it is sent to memory
- Logical address 0 to max, physical address: R, R+max.

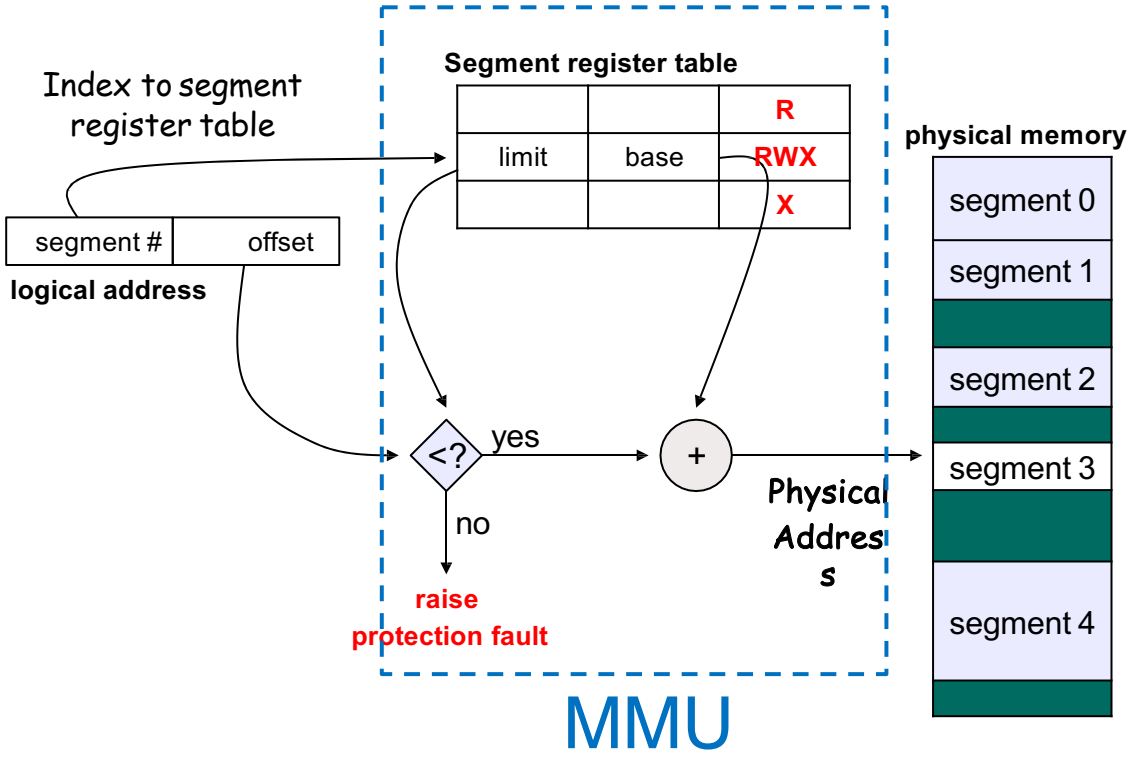


Logical vs. Physical Address

- What are the logical and physical address



Segment Lookup



Takeaway

- Partition evolution
- Memory partition – fixed and variable
  - first, best, worst fit
  - fragmentation: internal/ external
- Segmentation
  - Logical address vs physical address
- MMU: address translation + protection