# Memory Management

CS 416: Operating Systems Design, Spring 2011

Department of Computer Science
Rutgers University

Rutgers Sakai: 01:198:416 Sp11
(https://sakai.rutgers.edu)

# Memory Management

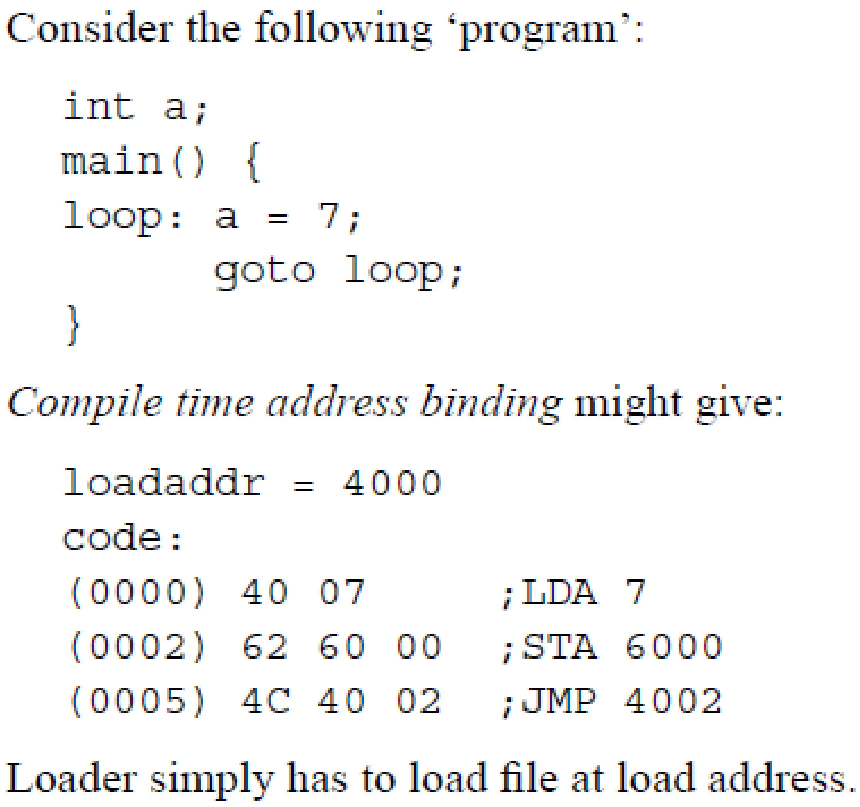
- Goals of Memory Management
  - Convenient abstraction for programming
  - Provide isolation for different processes
  - Allocate scarce physical memory to different processes
- Mechanism
  - Virtual Address Translation
  - Paging and TLB
  - Page Table Management
- Policies
  - Page replacement policies

# Address Binding

- 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)

# Compile time address binding

- If the location where the program would be loaded is known at compile time, the compiler generates a code with actual physical address.
- What are the advantages and disadvantages ?



Consider the following 'program':

```
int a;
main() {
loop: a = 7;
      goto loop;
}
```

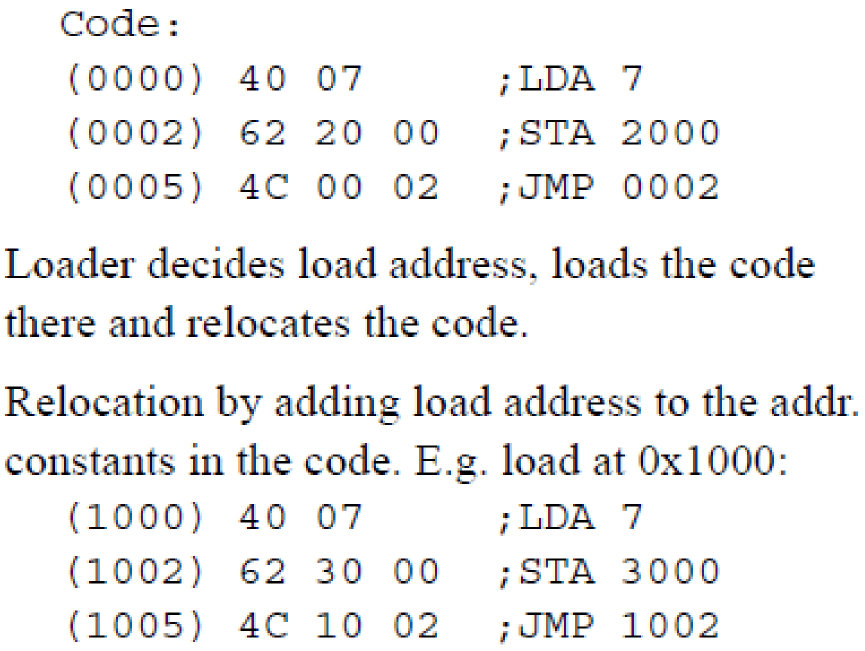
*Compile time address binding* might give:

```
loadaddr = 4000
code:
(0000) 40 07       ;LDA 7
(0002) 62 60 00    ;STA 6000
(0005) 4C 40 02    ;JMP 4002
```

Loader simply has to load file at load address.

# Load time address binding

- If the address where the program would be loaded is NOT known at compile time, Compiler generates a Relocatable address (e.g., 14 bytes from the beginning of this module)
- At load time, the relative addresses are converted to physical addresses depending on where it is loaded.



```
Code:
 (0000) 40 07      ;LDA 7
 (0002) 62 20 00   ;STA 2000
 (0005) 4C 00 02   ;JMP 0002
```

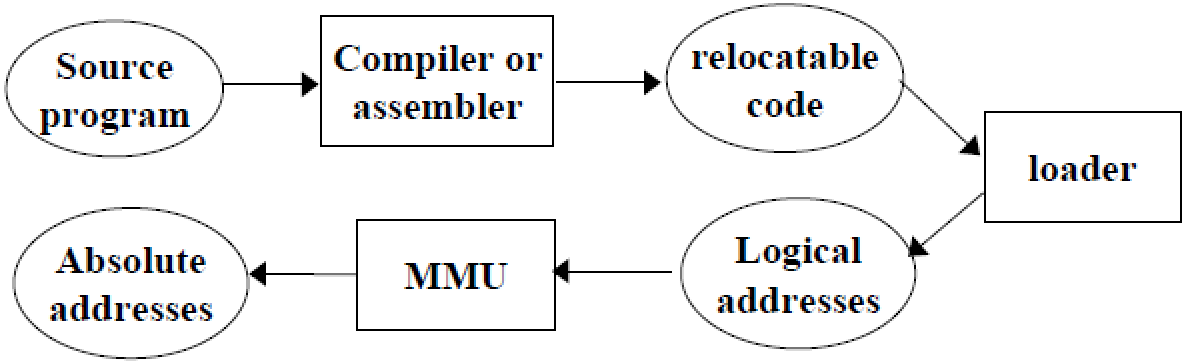
Loader decides load address, loads the code there and relocates the code.

Relocation by adding load address to the addr. constants in the code. E.g. load at 0x1000:

```
 (1000) 40 07      ;LDA 7
 (1002) 62 30 00   ;STA 3000
 (1005) 4C 10 02   ;JMP 1002
```

# Execution time address binding

- Load code at any physical address
- Physically move the code after loading
  - Requires special hardware support (MMU)



Source program

Compiler or assembler

relocatable code

loader

Logical addresses

MMU

Absolute addresses

# Virtual Memory

- The basic abstraction provided by the OS for memory management

- VM enables programs to run without requiring their entire address space to be resident in physical memory

- Observation: Many programs don’t use all of their code or data
  - e.g., branches may never be taken or variables not accessed
  - Therefore, no need to allocate memory for it until its used
  - OS should adjust amount of Physical memory allocated based on the program’s run-time behavior.

# Virtual Memory

- VM also isolates processes from each other
  - One process cannot access the memory addresses in other processes
  - Each process has its own address space

- VM requires both hardware and OS support
  - Hardware Support: Memory Management Unit (MMU) and Translation Look-aside Buffer(TLB)
  - OS support: virtual memory system to control the MMU and TLB

# Memory Management Requirements

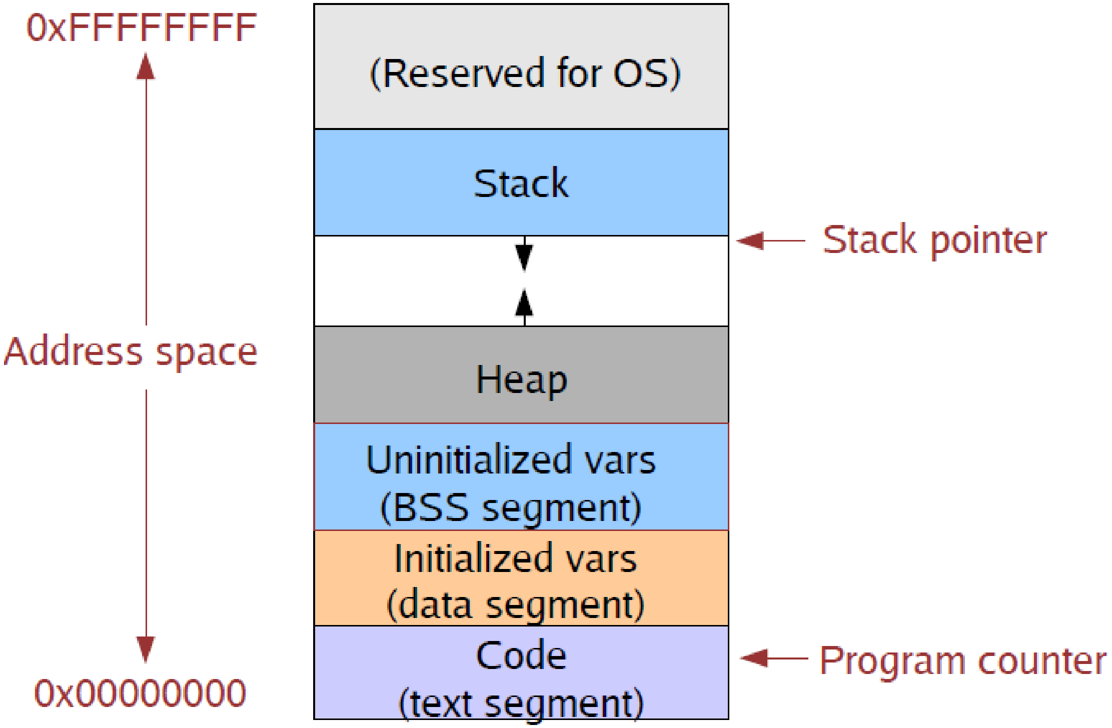
- Protection
  - Restricts which addresses processes can use so that they don’t overlap
- Fast Translation
  - Accessing a memory should be fast regardless of the protection scheme

    (It would be a bad idea to call into the OS for every memory access)
- Fast Context Switch
  - Overhead of updating the memory hardware on a context switch should be low

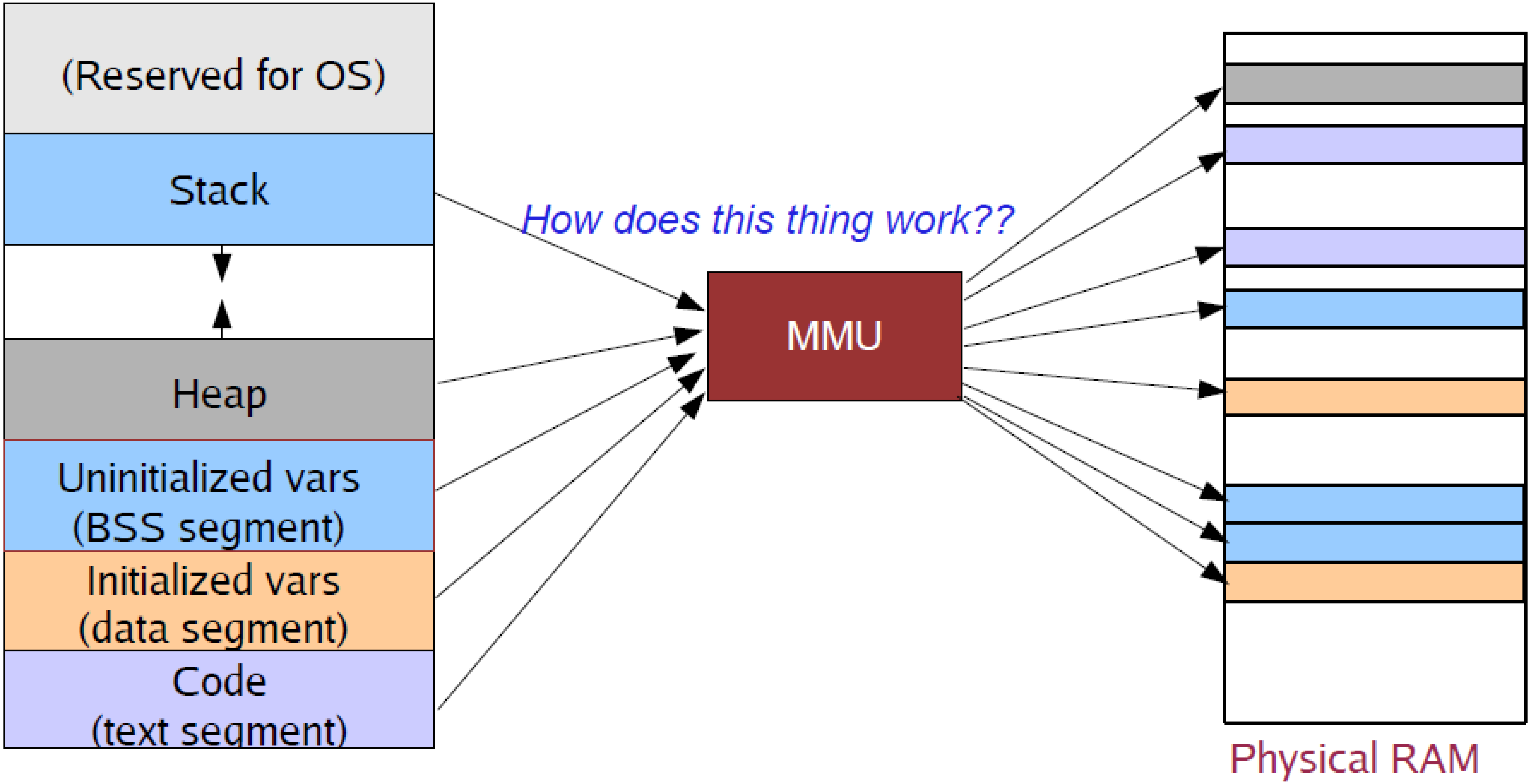
    (For ex, it would be a bad idea to write back all of the process’s memory out to the disk on every context switch)

# Virtual Addresses

- A *virtual address* is a memory address that a process uses to access its own memory
  - The virtual address is ***not the same*** as the physical RAM address in which it is stored
  - When a process accesses a virtual address, the MMU hardware ***translates*** the virtual address into a physical address
  - The OS determines the ***mapping*** from virtual address to physical address



# Virtual Addresses

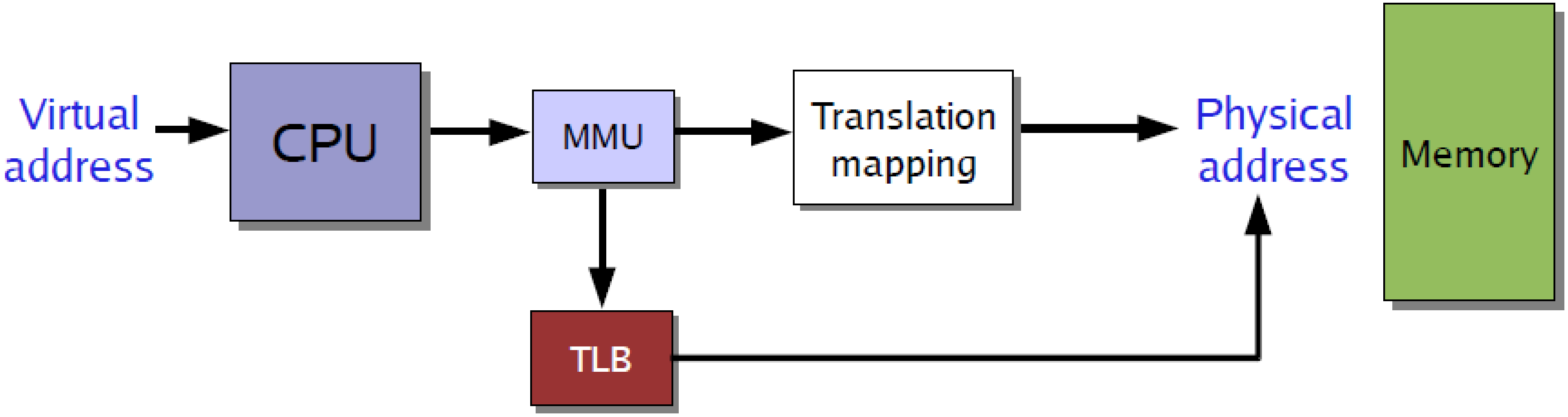


# Virtual Addresses

- A *virtual address* is a memory address that a process uses to access its own memory
  - The virtual address is ***not the same*** as the physical RAM address in which it is stored
  - When a process accesses a virtual address, the MMU hardware ***translates*** the virtual address into a physical address
  - The OS determines the ***mapping*** from virtual address to physical address
- Virtual Addresses allow *isolation*
  - Virtual addresses in one process refer to different physical memory than virtual addresses in another
    - Exception: shared memory regions.
- Virtual addresses allow *relocation*
  - A program does not need to know which physical addresses it will use when it is run
    - This however is a bad idea !! Why ?
  - Compilers generate relocatable code: Code that is independent of physical location in memory

# MMU and TLB

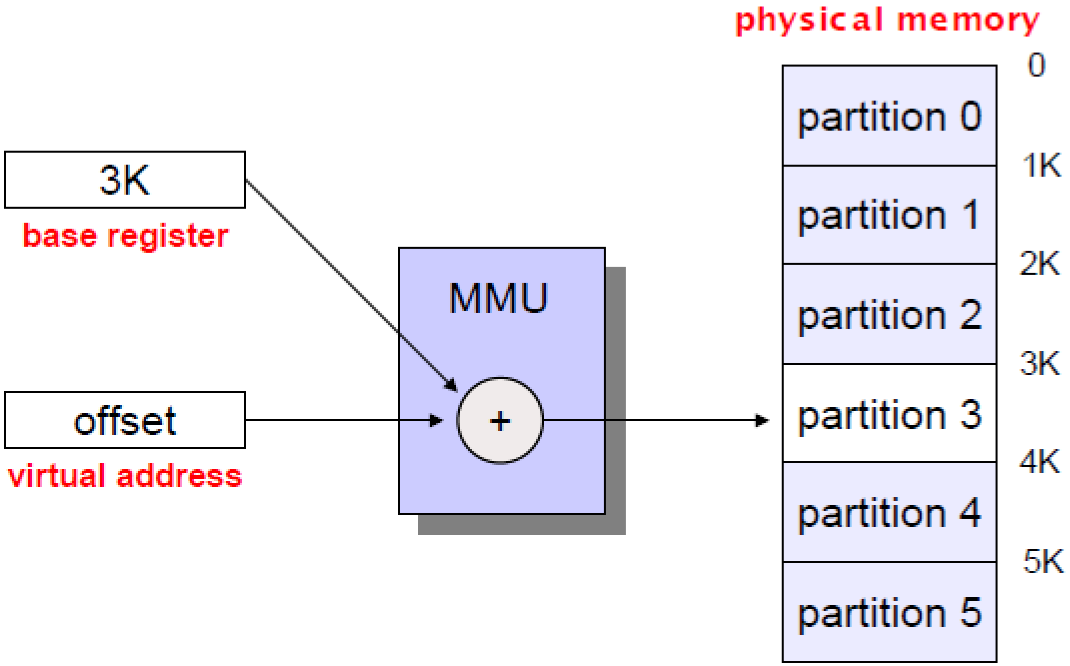
- Memory Management Unit (MMU)
  - Hardware that translates a virtual address to a physical address
  - Each memory reference is passed through the MMU
  - Translate a virtual address to a physical address – Lots of way to do this
- Translation Lookaside Buffer (TLB)
  - **Cache** for MMU virtual-to-physical address translations
  - Just an optimization – but an important one!



Cache of translations

# Simple Approach : Fixed Partitions

- Break Memory into FIXED partitions.
  - Each Partition contains exactly one process
  - Hardware requirement: base register
- Translation from virtual to physical address: Simply add base re



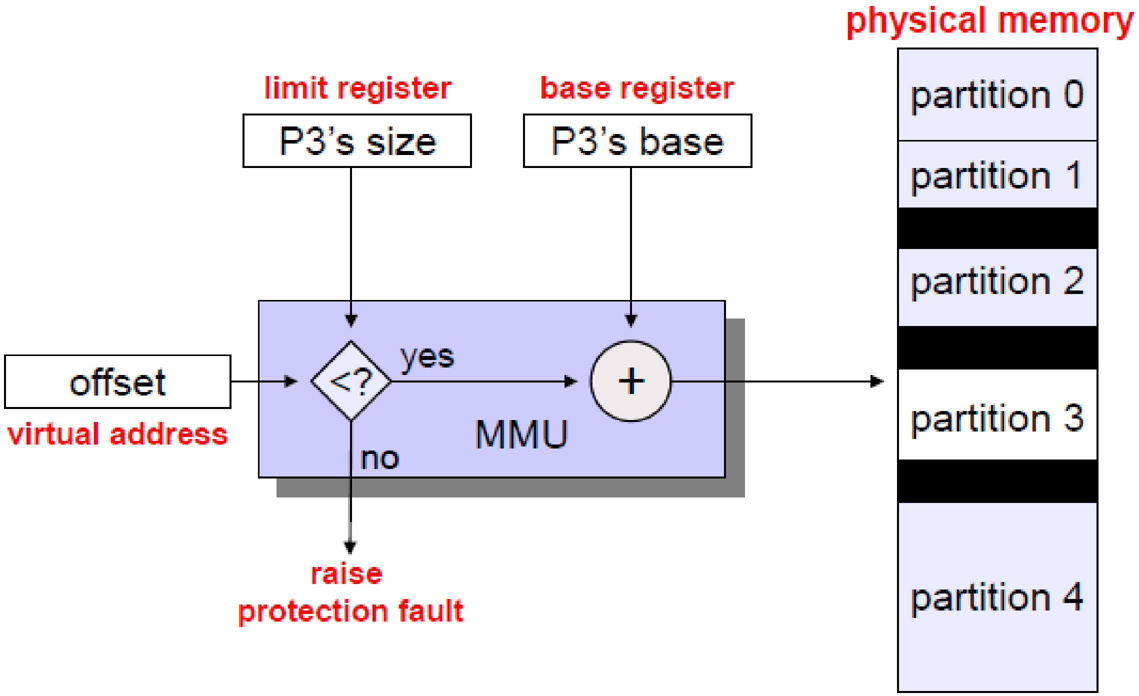
What are the advantages and disadvantages of this approach ?

# Simple Approach : Fixed Partitions

- Advantages:
  - Fast Context Switch – Only need to update the base register
  - Simple memory management code: Locate empty partition when running new process
- Disadvantages
  - Internal Fragmentation
    - If the entire partition is not consumed, there is a wastage
  - Static partition sizes
    - No single size is appropriate for all programs

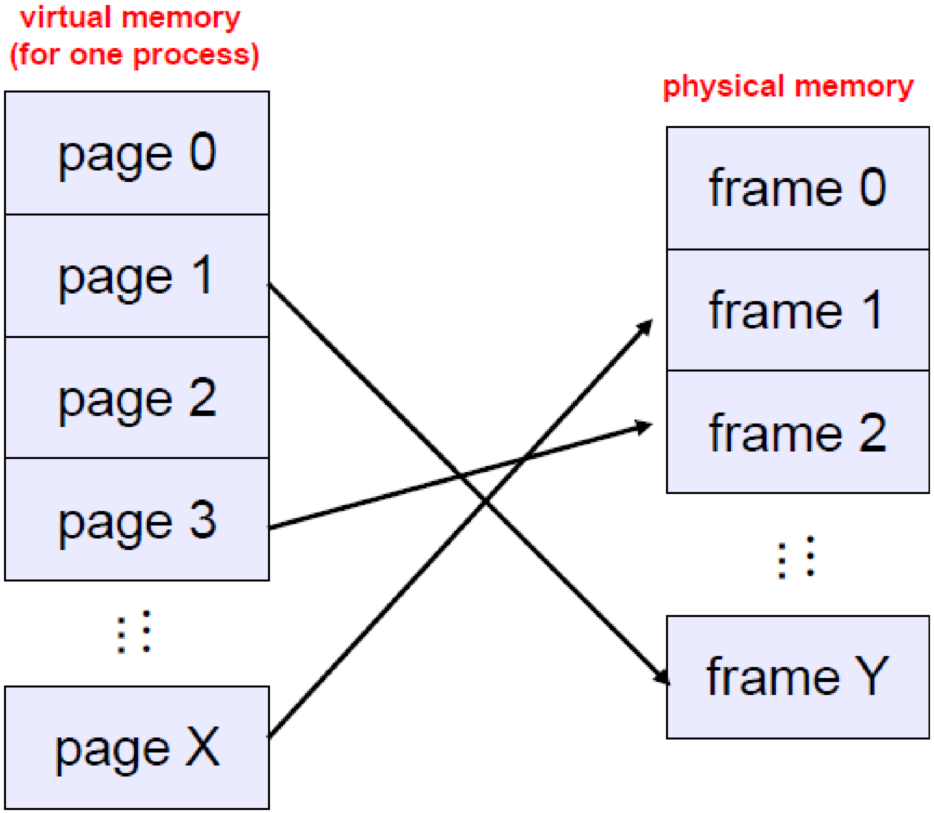
# Variable Partitions

- Allow variable sized partitions
  - Now requires both and *base* and a *limit* register
  - Solves the internal fragmentation problem: The partitions size is based on process needs
- New Problem: External Fragmentation



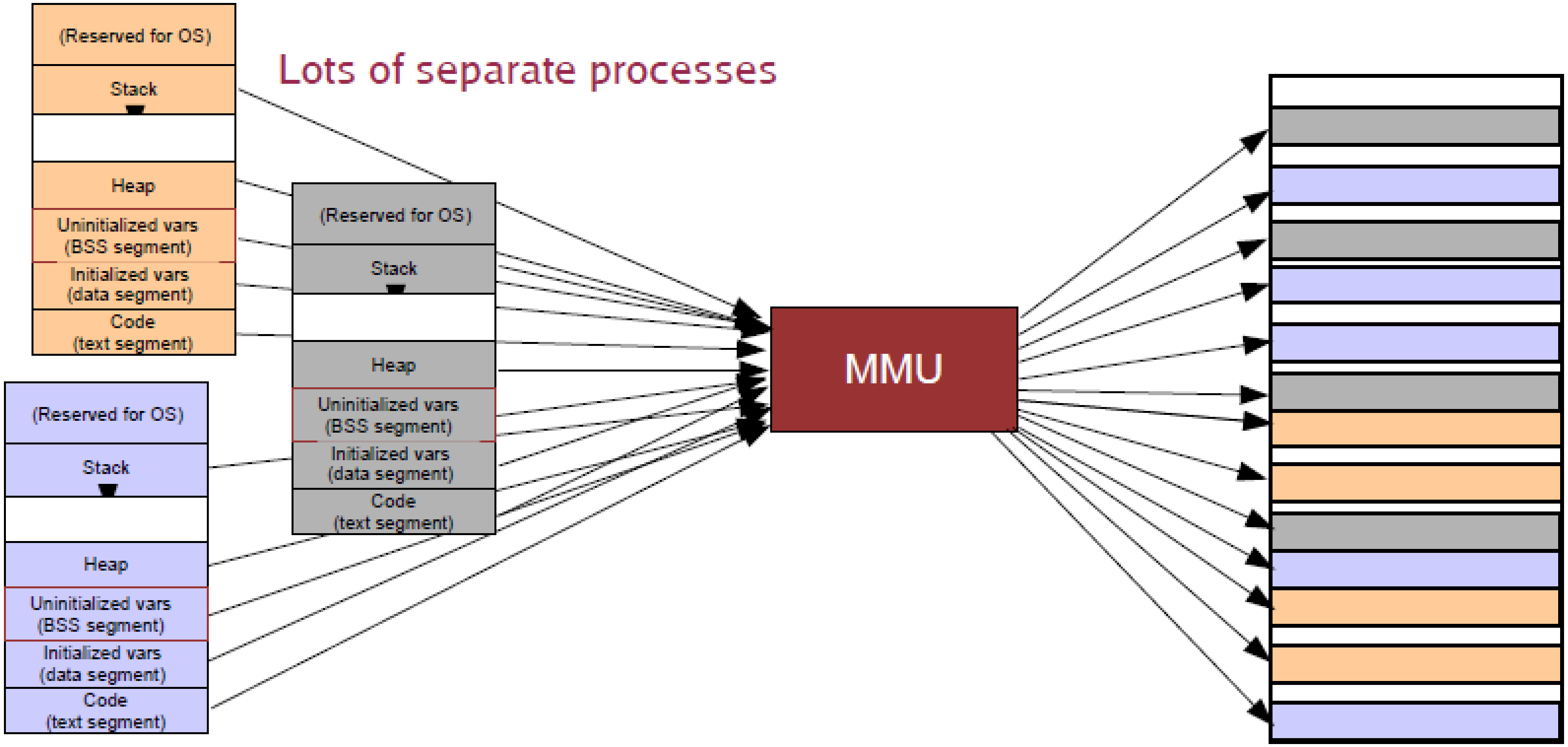
# Modern Technique: Paging

- Solve external fragmentation by fixed size chunks of virtual and physical memory
  - Virtual memory unit is called a *Page*
  - Physical memory unit is called a *frame* ( or a page frame)



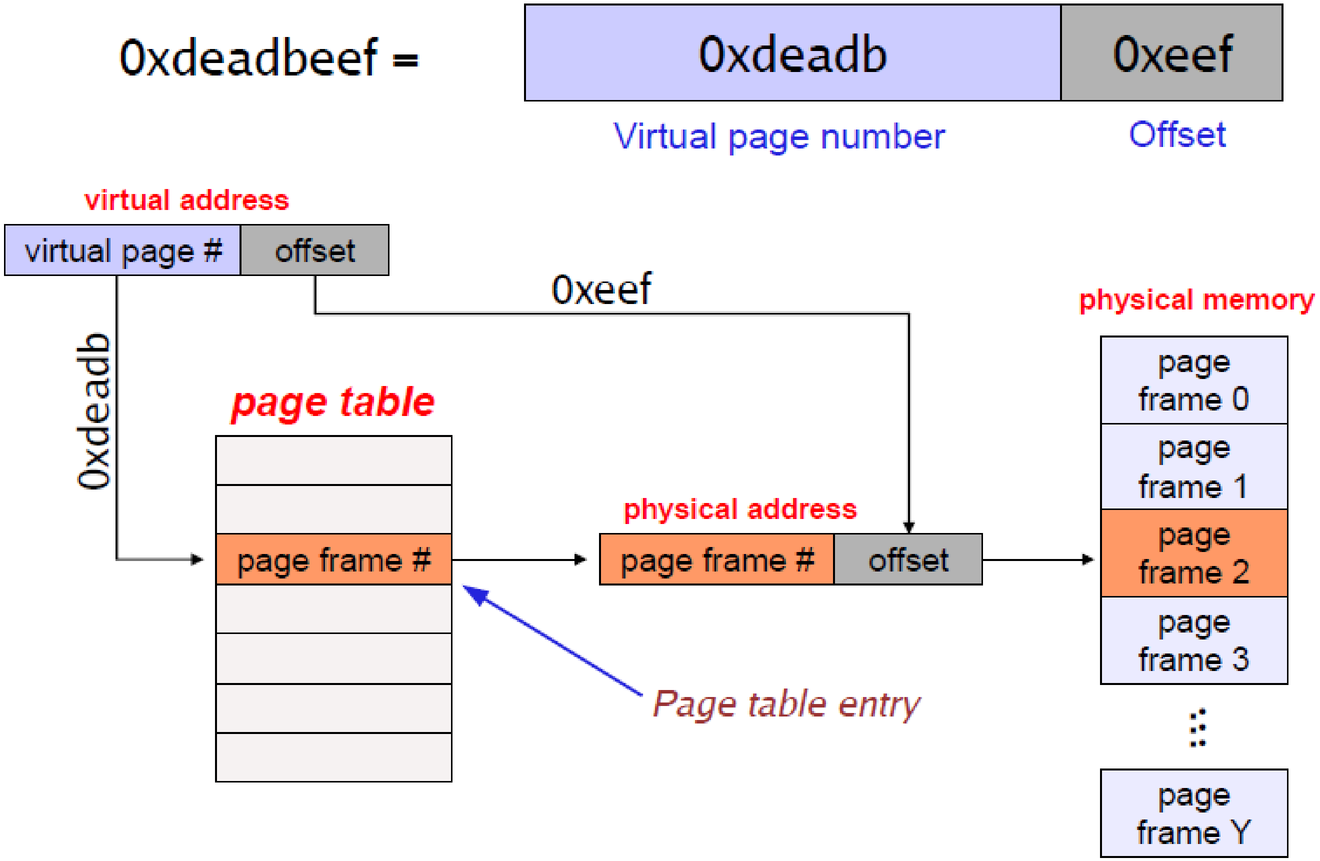
# Application Perspective

- Application believes it has a single contiguous address space ranging from 0 to $2^p - 1$ bytes
  - Where p is the number of number of bits in the pointer (e.g 32 bits)
- In reality, virtual pages are scattered across physical memory



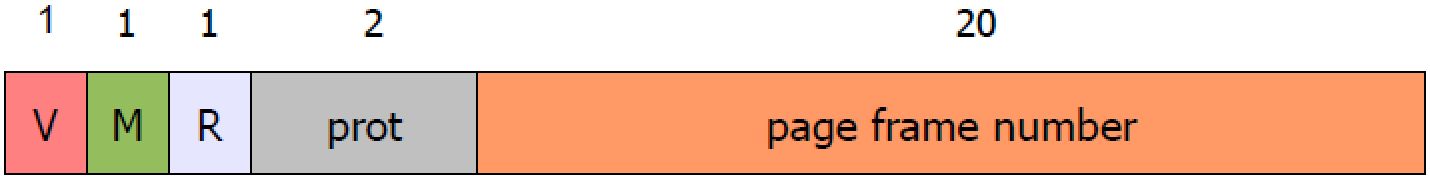
# Virtual Address Translation

- Virtual to Physical address translation performed by MMU
  - Virtual address is broken into two parts: virtual page number and *offset*
  - The mapping between the Page to Frame is maintained by *Page Table*



# Page Table Entries (PTEs)

- Typical PTE format (depends on CPU architecture)



| 1 | 1 | 1 | 2 | 20 |
|---|---|---|---|---|
| V | M | R | prot | page frame number |

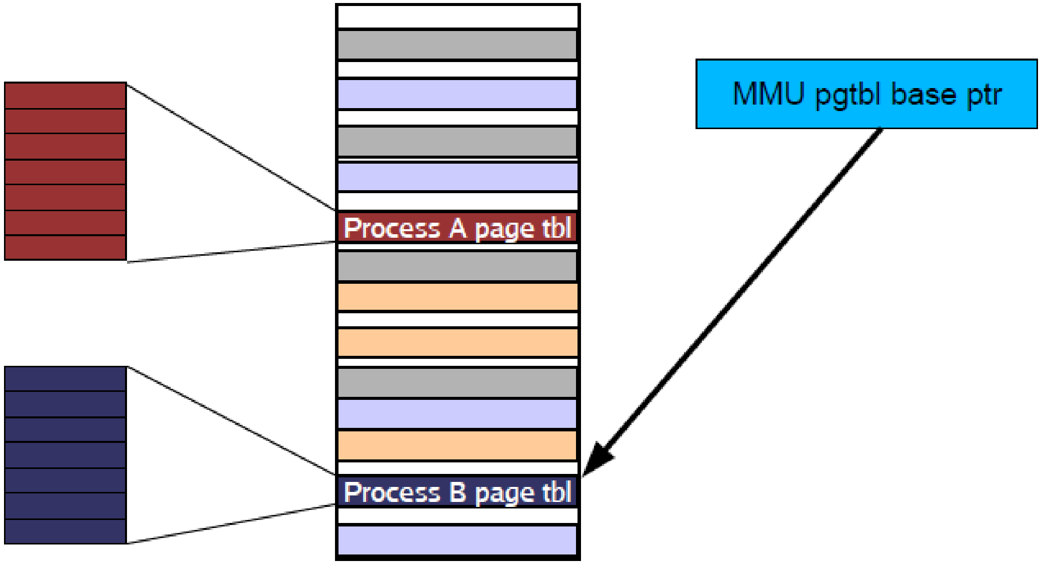
- Various bits accessed by MMU on each page access:
  - Valid bit (V): whether the corresponding page is in memory
  - Modify bit (M): Indicates whether a page is “dirty” (Modified)
  - Reference bit (R): Indicates whether a page has been accessed (read/write)
    - **Useful for page replacement algorithms.**
  - Protection bits: Specify if  page is readable, writable or executable
  - Page frame number: Physical location of page in RAM

# Advantages of Paging

- Simplifies physical memory management
  - OS maintains a list of free physical page frames
  - To allocate a physical page, just remove an entry from this list
- No External fragmentation
  - No need to allocate pages contiguously
  - Therefore, pages from diff. processes can be interspersed.
- Allocation of memory can be performed at a fine granularity
  - Only allocate physical memory to those parts of the address space that requires it
  - Can swap out unused pages out of memory when low on memory

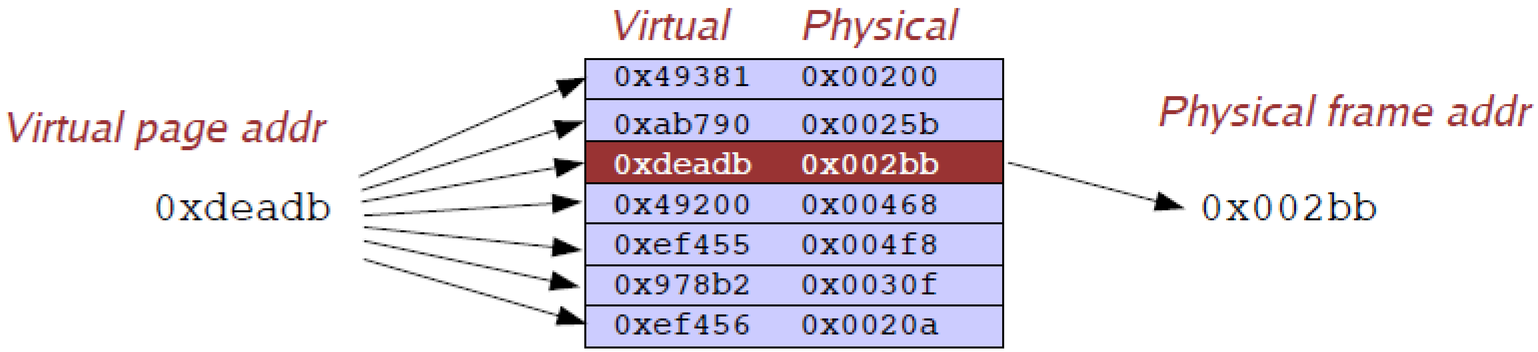
# Page Tables

- Page Tables stores the virtual-to-physical address mappings
- Where is the page table located ? ***In Memory***
- How does the MMU access them ?
  - The MMU has a special register called the page table base pointer
  - This points to the physical memory address of the top of the page table for this currently running process.



# The TLB

- Now, we have introduced a high overhead for address translation
  - On every memory access, must have a *separate* access to consult the page table
- Solution: Translation Lookaside Buffer (TLB)
  - Very fast cache directly on the CPU
  - Caches most recent virtual to physical address translations
  - Implemented as fully associative cache
  - A *TLB miss* requires that the MMU actually try to do the address translation



| Virtual | Physical |
|---|---|
| 0x49381 | 0x00200 |
| 0xab790 | 0x0025b |
| 0xdeadb | 0x002bb |
| 0x49200 | 0x00468 |
| 0xef455 | 0x004f8 |
| 0x978b2 | 0x0030f |
| 0xef456 | 0x0020a |

# Page Table Size

How big are the page tables **per process** ?

We need one page table entry per page

Lets say, we have a 32 bit address and a 4KB page size

How many pages do we have ? (Virtual Address space)

2^32 == 4GB / 4KB per page = 1,048,576 (i.e 1M Pages)

How big is each page table entry?

Depends on the CPU architecture. On x86, it’s 4 bytes

So, the total page table size is : 1M pages * 4 bytes/PTE  == 4Mbytes

If we have 100 processes, then 400MBytes of Page Tables needs to be kept.

How do we deal with this ? – Next lecture !

# Memory Management Requirements - Revisted

- Protection
  - Process can only refer to its own virtual addresses.
  - O/S responsible for ensuring that each process uses disjoint **physical pages**
- Fast Translation
  - MMU (on the CPU) translates each virtual address to a physical address.
  - TLB caches recent virtual->physical translations
- Fast Context Switch
  - Only need to swap pointer to current page tables when context switching!