Virtual Memory

Jinkyu Jeong (jinkyu@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

Virtual Memory: Goals

- Transparency
  - Processes should not be aware that memory is shared
  - Provides a convenient abstraction for programming (i.e. a large, contiguous memory space)
- Efficiency
  - Minimizes fragmentation due to variable-sized requests (space)
  - Gets some hardware support (time)
- Protection
  - Protect processes and the OS from another process
  - Isolation: a process can fail without affecting other processes
  - Cooperating processes can share portions of memory

Accessing Memory

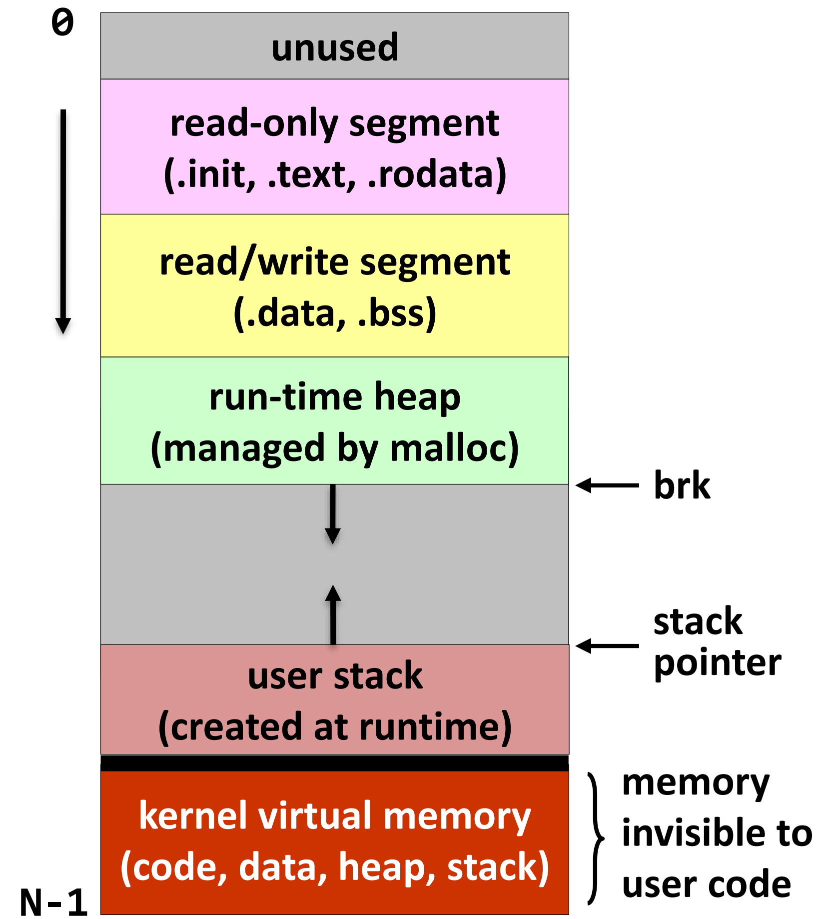
- Example

```
#include <stdio.h>

int n = 0;

int main ()
{
    n++;
    printf ("&n = %p, n = %d\n", &n, n);
}

% ./a.out
&n = 0x0804a024, n = 1
% ./a.out
&n = 0x0804a024, n = 1
```

  – What happens if two users simultaneously run this program?

(Virtual) Address Space

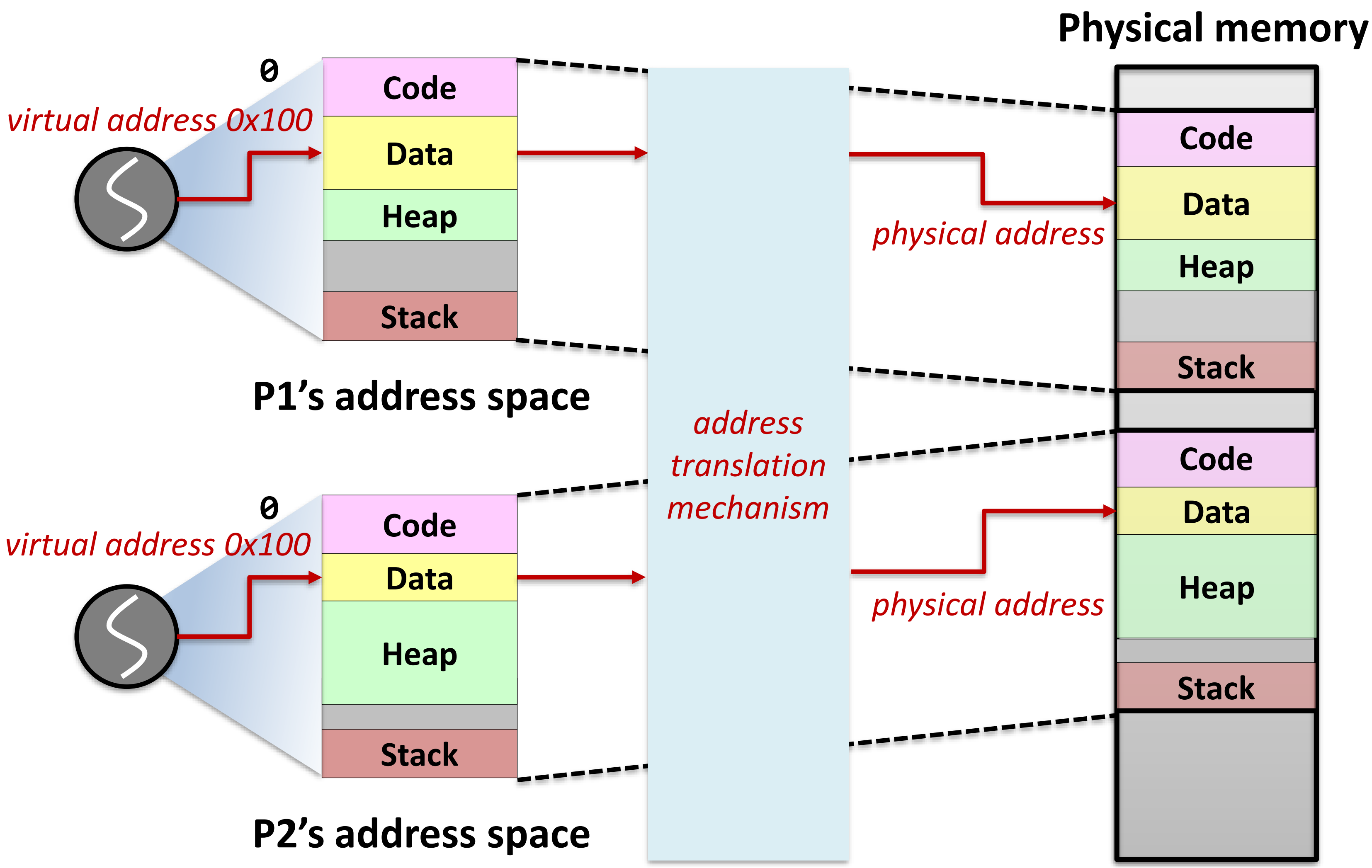
- Process' abstract view of memory
  - OS provides illusion of private address space to each process
  - Contains all of the memory state of the process
  - Static area
    - Allocated on exec()
    - Code & Data
  - Dynamic area
    - Allocated at runtime
    - Can grow or shrink
    - Heap & Stack



Virtual Memory

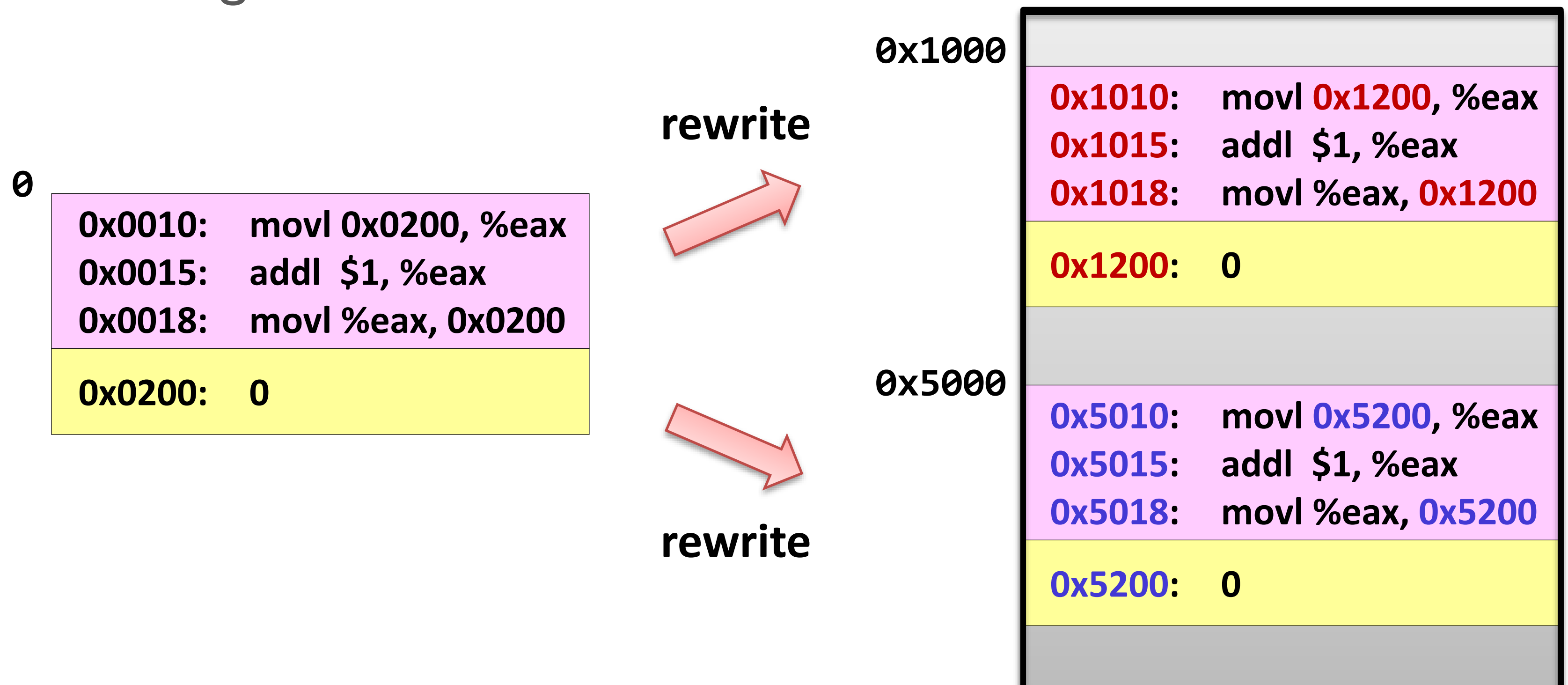
- Each process has its own virtual address space
  - Large and contiguous
  - Use virtual addresses for memory references
  - Virtual addresses are private to each process
- Address translation is performed at run time
  - From a virtual address to the corresponding physical address
- Supports lazy allocation
  - Physical memory is dynamically allocated or released on demand
  - Programs execute without requiring their entire address space to be resident in physical memory

Virtual Memory



Static Relocation (1)

- Software-based relocation
  - OS rewrites each program before loading it in memory
  - Changes addresses of static data and functions



Static Relocation (2)

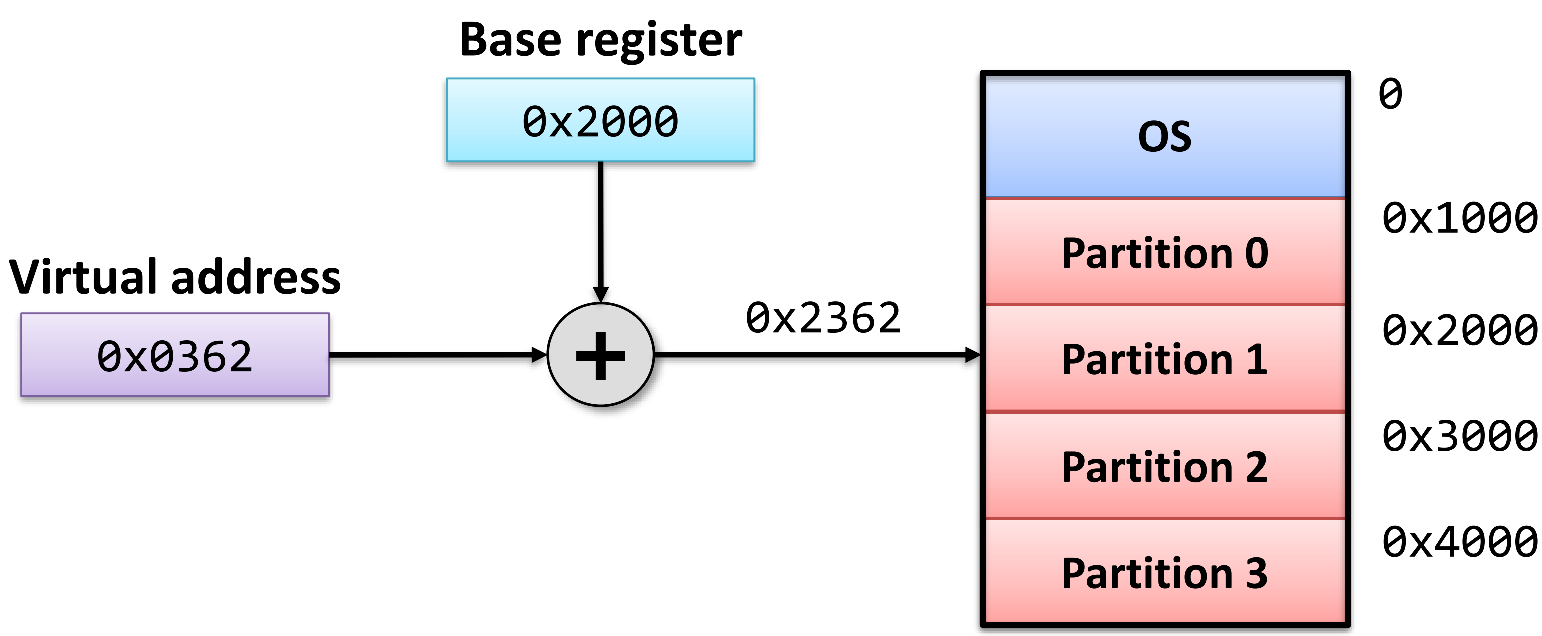
- Pros
  - No hardware support is required
- Cons
  - No protection enforced
    - A process can destroy memory regions of the OS or other processes
    - No privacy: can read any memory address
  - Cannot move address space after it has been placed
    - May not be able to allocate a new process due to external fragmentation

Dynamic Relocation

- Hardware-based relocation
  - MMU (Memory Management Unit) performs address translation on every memory reference instructions
  - Protection is enforced by hardware: if the virtual address is invalid, the MMU raises an exception
  - OS passes the information about the valid address space of the current process to the MMU
- Implementations
  - Fixed or variable partitions
  - Segmentation
  - Paging

Fixed Partitions (1)

- Physical memory is broken up into fixed partitions
  - Size of each partition is the same and fixed
  - The number of partitions = degree of multiprogramming



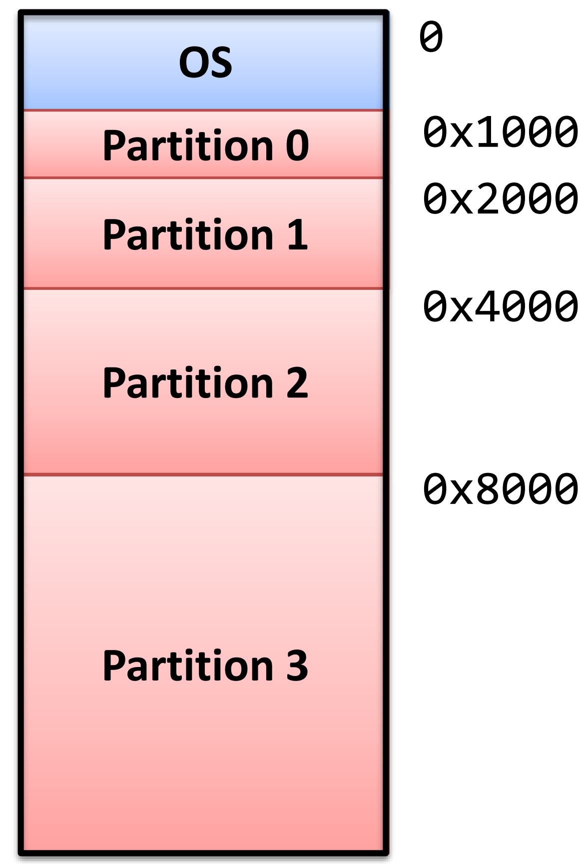
Fixed Partitions (2)

- Hardware requirements: base register
  - Physical address = virtual address + base register
  - Base register loaded by OS on context switch

- Pros
  - Easy to implement
  - Fast context switch
- Cons
  - Internal fragmentation: unused area in a partition is wasted
  - Partition size: one size does not fit all

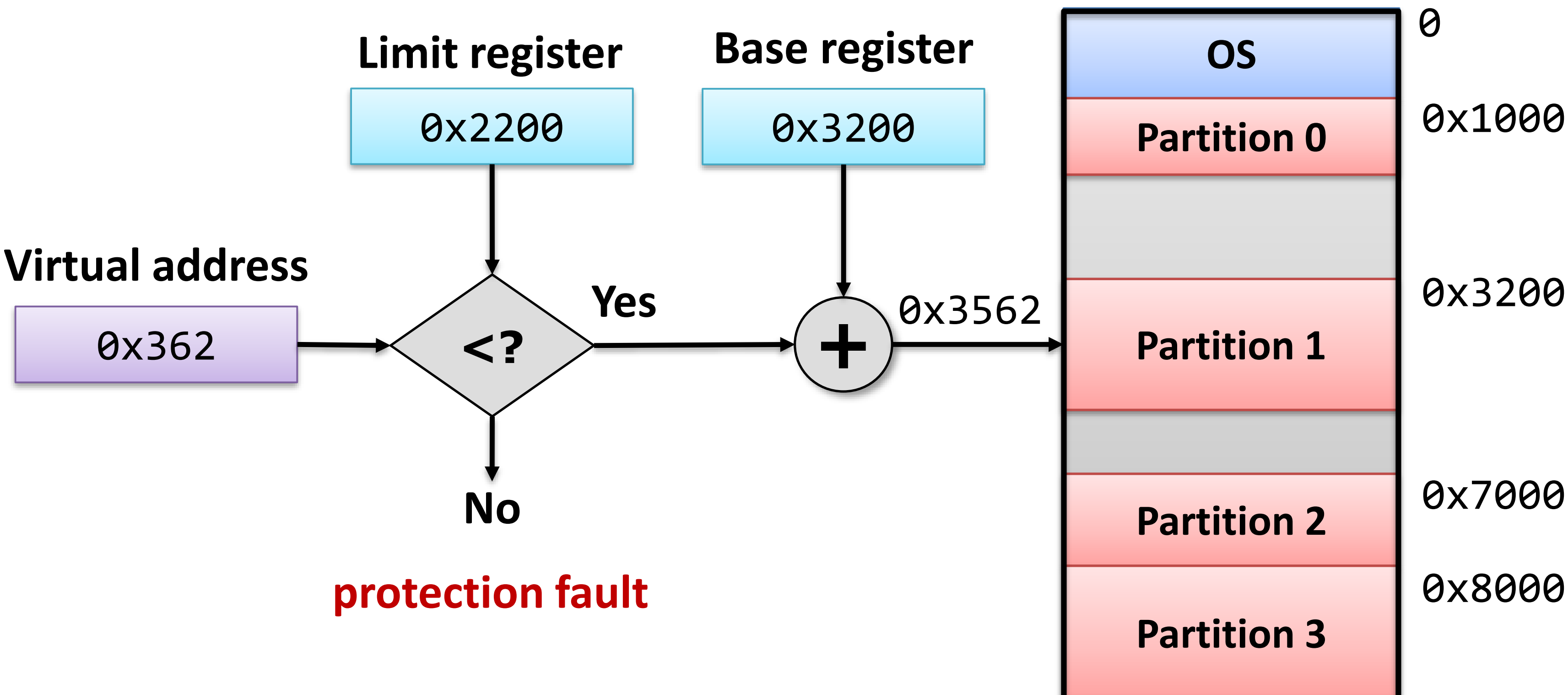
Fixed Partitions (3)

- Improvement
  - Partition size needs not be equal
  - Allocation strategies
    - A separate queue for each partition size
    - A single queue + first fit
    - A single queue + best fit
  - Used in IBM OS/MFT (Multiprogramming with a Fixed number of Tasks)



Variable Partitions (1)

- Physical memory is divided into variable-sized partitions
  - Used in IBM OS/MVT



Variable Partitions (2)

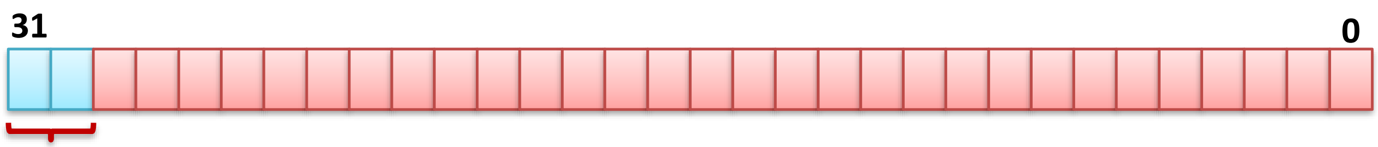
- Hardware requirements: base register + limit register
  - The role of limit register: protection
- Pros
  - Simple, inexpensive implementation
  - No internal fragmentation
- Cons
  - Each process must be allocated contiguously in physical mem
  - External fragmentation:
    - Holes are left scattered throughout physical memory
    - Compaction can be used to reduce external fragmentation
  - No partial sharing: cannot share parts of address space

Segmentation

- Divide address space into logical segments
  - Each segment corresponds to logical entity in address space
    - Code, data, stack, heap, etc.
  - Users view memory as a collection of variable-sized segments, with no necessary ordering among them
    - Virtual address: <Segment #, Offset>
  - Each segment can independently
    - be placed separately in physical memory
    - grow or shrink
    - be protected (separate read/write/execute protection bits)
  - Natural extension of variable partitions
    - Variable partitions: 1 segment / process
    - Segmentation: many segments / process

Segmentation: Addressing

- Explicit approach
  - Use a part of virtual address as a segment number
  - The remaining bits mean the offset within the segment
  - e.g. VAX/VMS system



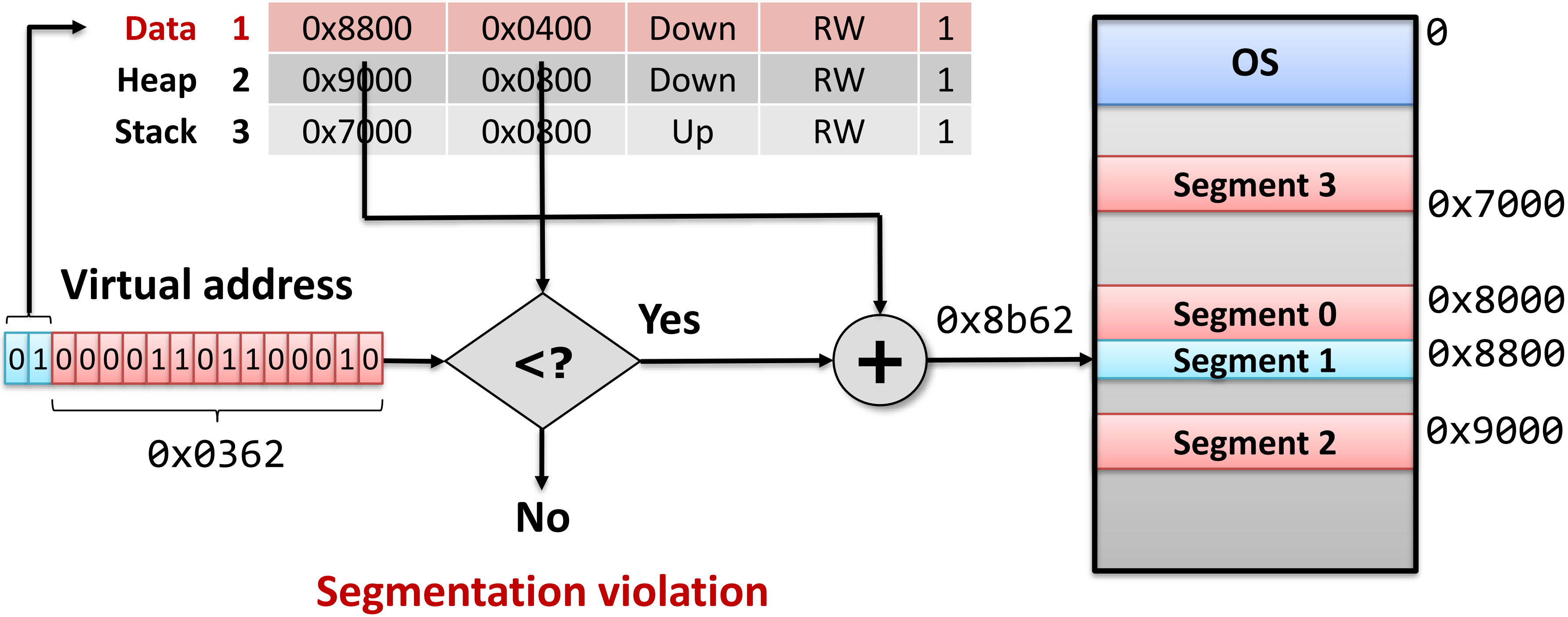
00: P0 (User code, data, heap)
01: P1 (User stack)
1X: S (System)

- Implicit approach
  - Determines the segment by the type of memory reference
    - PC-based addressing: code segment
    - SP- or BP-based addressing: stack segment

Segmentation: Implementation

- Segment registers or table (per process)

		Base	Limit	Dir	Prot	V
Code	**0**	0x8000	0x0800	Down	RO-X	1
Data	**1**	0x8800	0x0400	Down	RW	1
Heap	**2**	0x9000	0x0800	Down	RW	1
Stack	**3**	0x7000	0x0800	Up	RW	1



Segmentation: Pros

- Enables sparse allocation of address space
  - Stack and heap can grow independently
- Easy to protect segments
  - Valid bit
  - Different protection bits for different segments
    - e.g. Read-only status for code, Kernel-mode-only for system segment
- Easy to share segments
  - Put the same translation into base/limit pair
  - Code/data sharing at segment level (e.g. shared libraries)
- Supports dynamic relocation of each segment

Segmentation: Cons

- Each segment must be allocated contiguously
  - External fragmentation
  - May not have sufficient physical memory for large segments
- Large segment table
  - Keep in main memory
  - Use hardware cache for speed
- Cross-segment addresses
  - Segments need to have same segment number for pointers to them to be shared among processes
  - Otherwise, use indirect addressing only

Summary

- Separates user's virtual memory from physical memory
  - Abstracts main memory into a large, uniform array of bytes
  - Frees programmers from the concerns of memory limitations
  - Physical memory locations can be moved transparently
- The virtual address space is overcommitted
  - Allows the execution of processes that may not be completely in memory
  - Physical memory is allocated on demand
  - Views the physical memory as a cache for the disk
- Easy to protect and share memory regions among processes