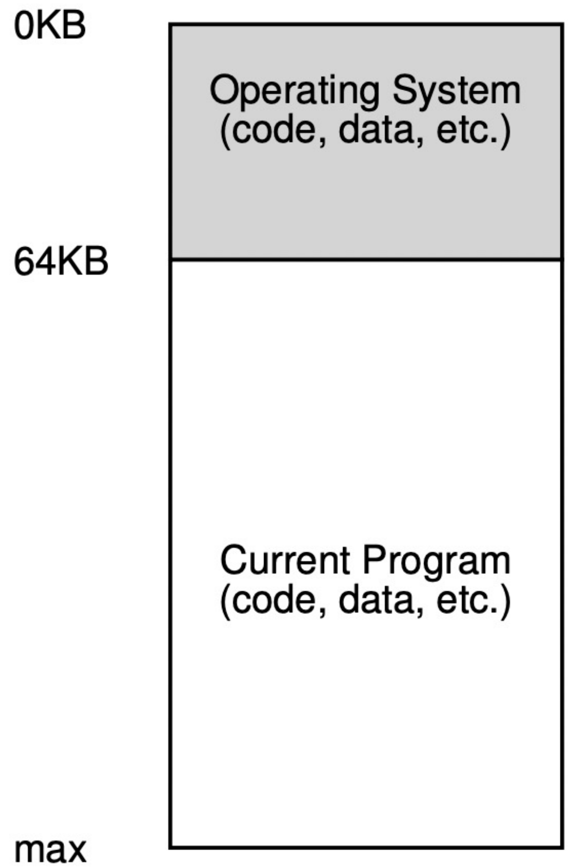
Lecture 6
Address Translation

Prof. Yinqian Zhang

Spring 2023

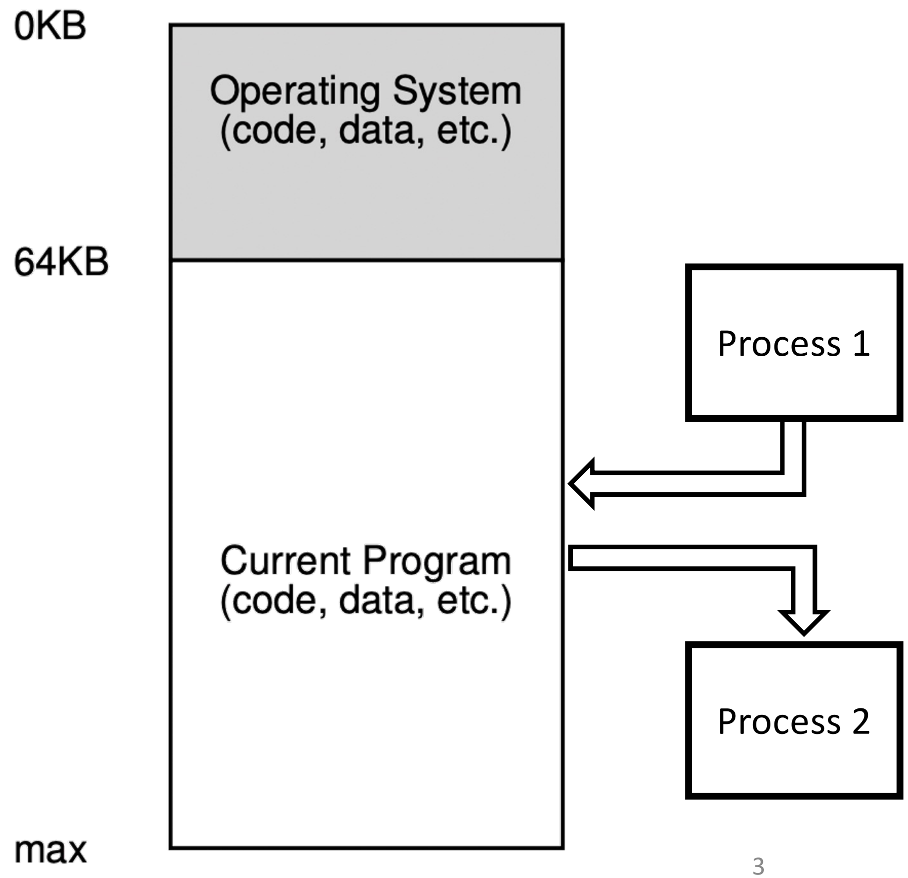
Operating System in Early Days

- The OS is a set of routines (a library) that uses lower memory
  - Starting at physical address 0 in this example
- One running program uses the rest of memory
  - Starting at physical address 64k in this example



Multiprogramming and Time Sharing

- Multiprogramming [DV66]
  - Multiple processes ready to run at a given time
  - OS switches between them, e.g., when one decided to perform I/O.
- Benefit of multiprogramming
  - Time sharing of computer resources
  - More effective use of CPU
- What about physical memory?
  - Moving data in/out of memory is slow

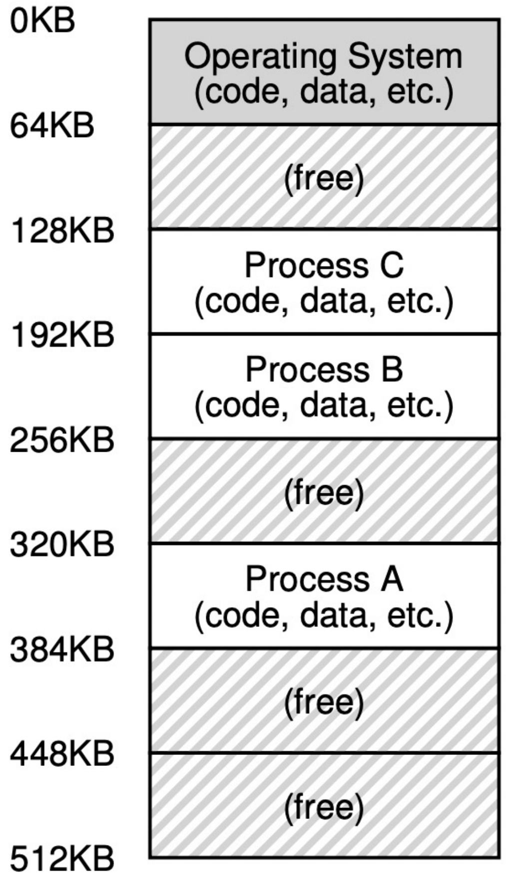


[DV66] Jack B. Dennis, Earl C. Van Horn. “Programming Semantics for Multiprogrammed Computations”. 1966

Multiprogramming with Memory Partition

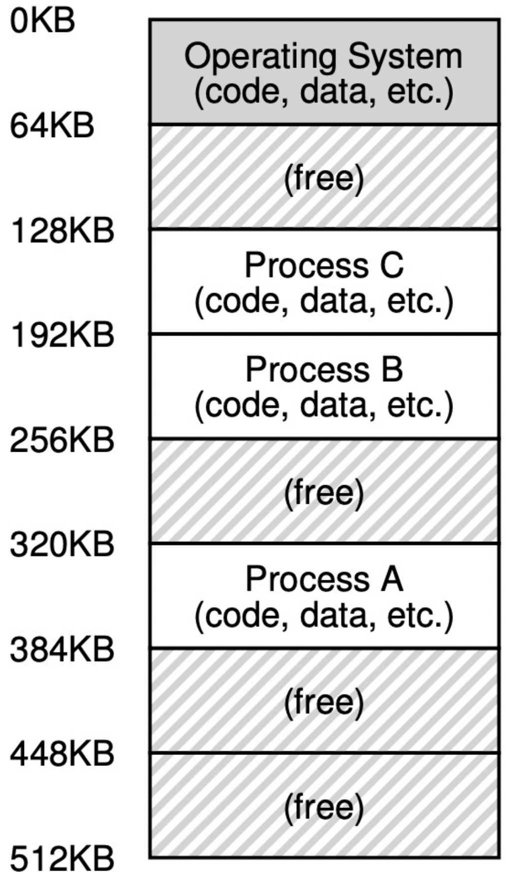
- Solution:
  - Leave processes in memory when switching
  - Each process owns a small part of the physical memory that is carved out for them.

- New demand for complex memory management



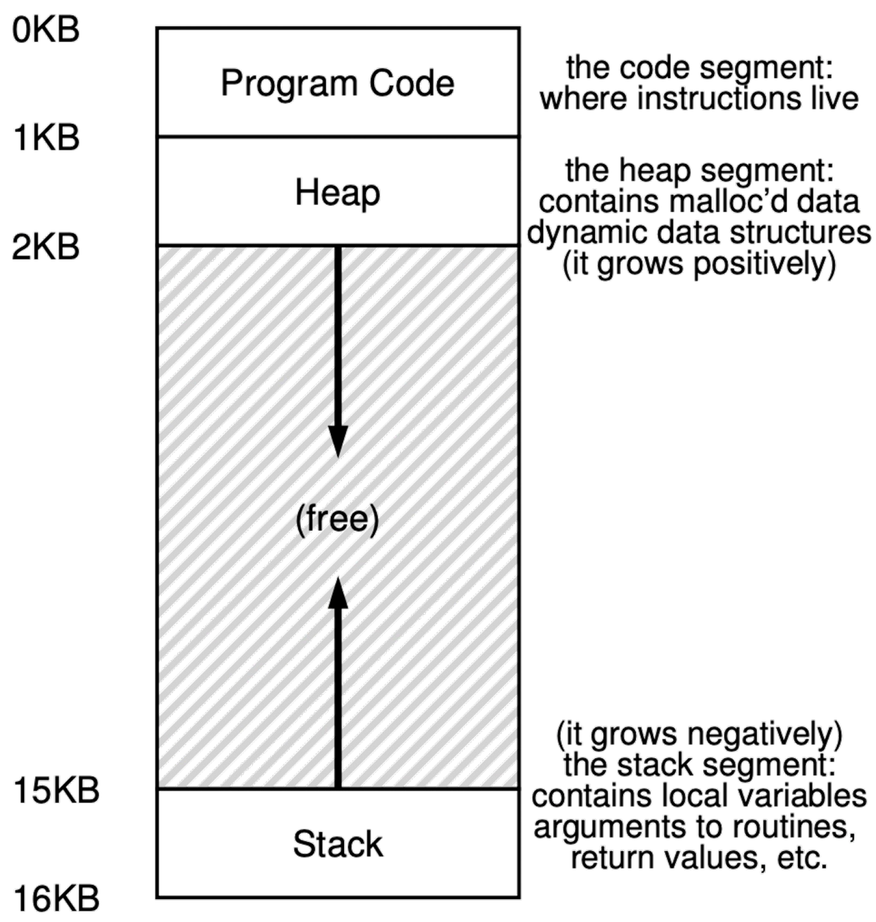
Multiprogramming with Memory Partition

- Potential issues:
  - What happens when Process C needs more memory?
  - How to compile Program B so that it knows it will run at 192KB?
  - What if Process C has an error and writes to address at 1KB or 330KB?



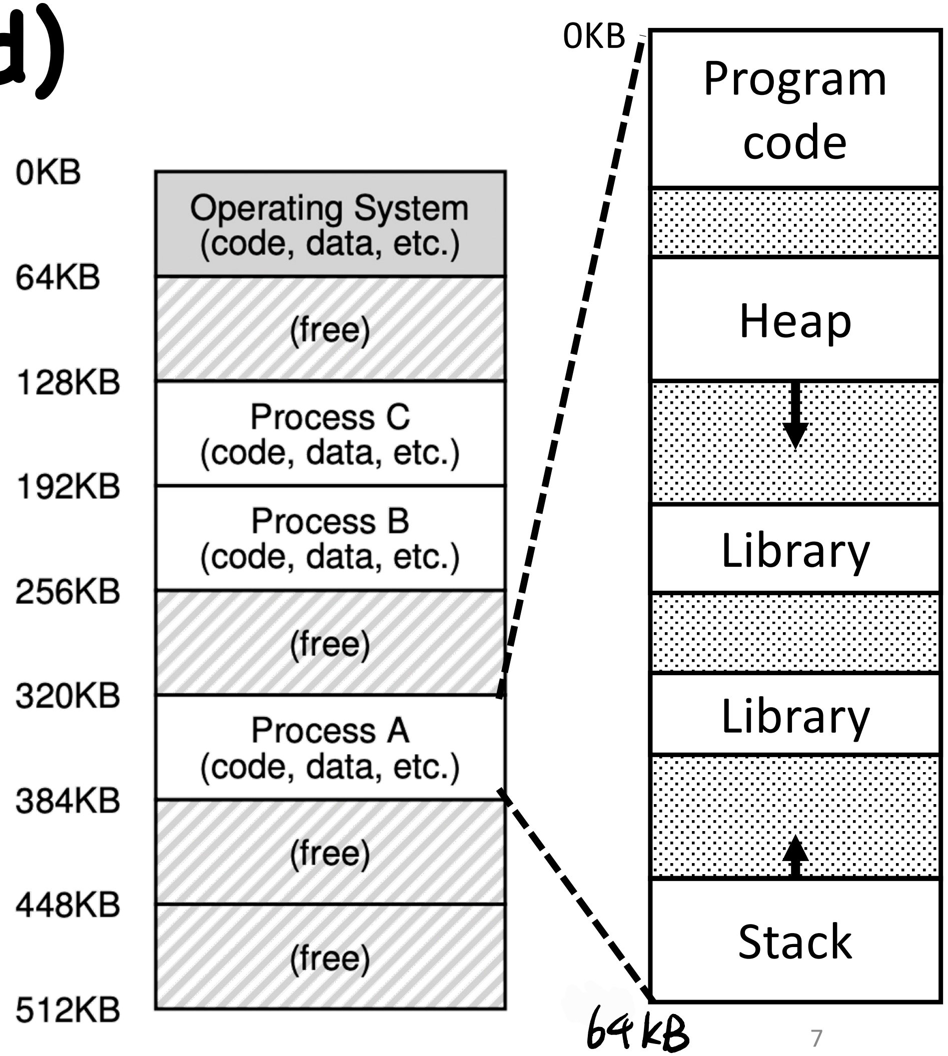
Address Space

- Address space is an important OS abstraction
  - Address space is a process' **view** of memory in the computer system
- Segments in an address space
  - Code segment: instructions at the bottom
  - Stack segment: local variables, arguments, return values
  - Heap: malloc
  - Stack and Heap need to grow



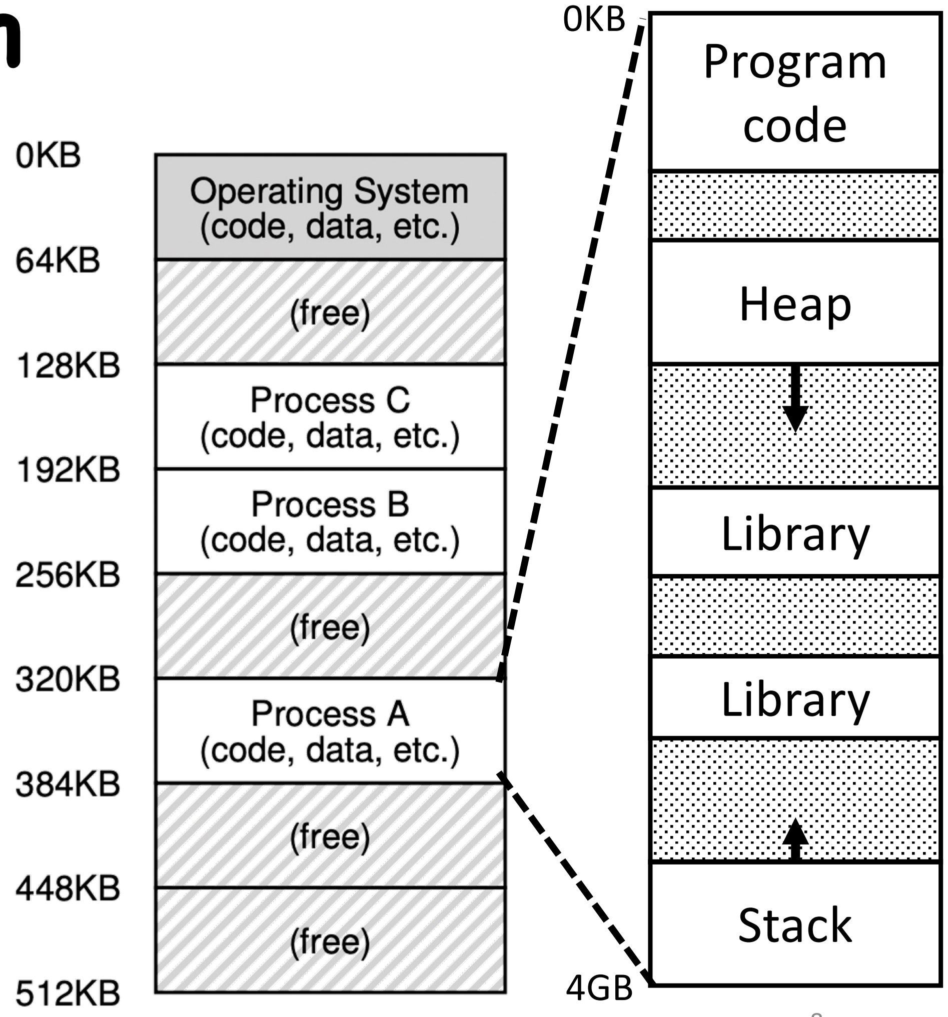
Address Space (Cont'd)

- This 16KB address space is just an abstraction
  - 0KB in the address space is not 0KB of physical memory
- This 16KB address space is just an illustration
  - 32-bit CPU supports up to 2^32 Byte (4GB) address space
  - 64-bit CPU supports up to 2^64 (4EB) Byte
    - But most CPU would reserve higher address bits
    - x86-64 supports only 2^48 Bytes (256TB) address space



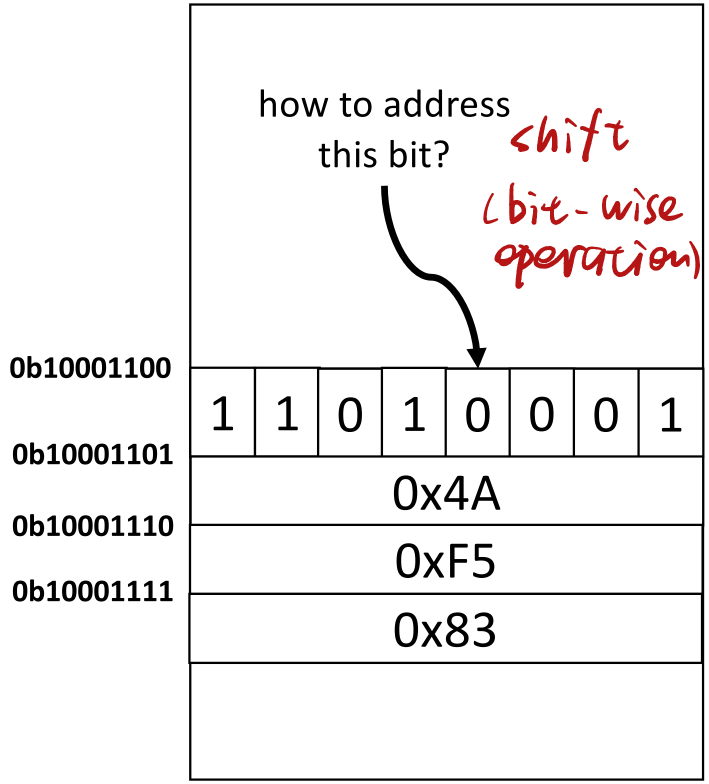
Memory Virtualization

- An abstraction of a private, large address space for multiple running processes on top of a single, physical memory
- Virtual address
  - Address in a process' own address space
- Physical address
  - Address of the physical memory
- Address translation
  - Virtual to physical address translation
  - example: 0KB -> 320KB



Aside: Addressing Memory

- Memory address is the address of a BYTE
  - 1 byte = 8 bit
  - how to address a bit?
- Address representation
  - hexidecimal: 0x8c
  - decimal : 140
  - binary: 0b10001100
- Big endian or little endian
  - 32-bit int at 0x8c
  - big endian: 0x d1 4a f5 83
  - little endian: 0x 83 f5 4a d1



Memory Virtualization (Cont'd)

- A mechanism that virtualize memory should
  - Be **transparent**
    - Memory virtualization should be invisible to processes
    - Processes run as if on a single private memory
  - Be **efficient**
    - Time: translation is fast
    - Space: not too space consuming
  - Provide **protection**
    - Enable **memory isolation**
    - One process may not access memory of another process or the OS kernel
    - Isolation is a key principle in building reliable systems

Virtual Address v.s. Physical Address

- Process uses virtual addresses

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
    printf("code : %p\n", main);
    printf("heap : %p\n", malloc(100e6));
    int x = 3;
    printf("stack: %p\n", &x);
    return x;
}
```

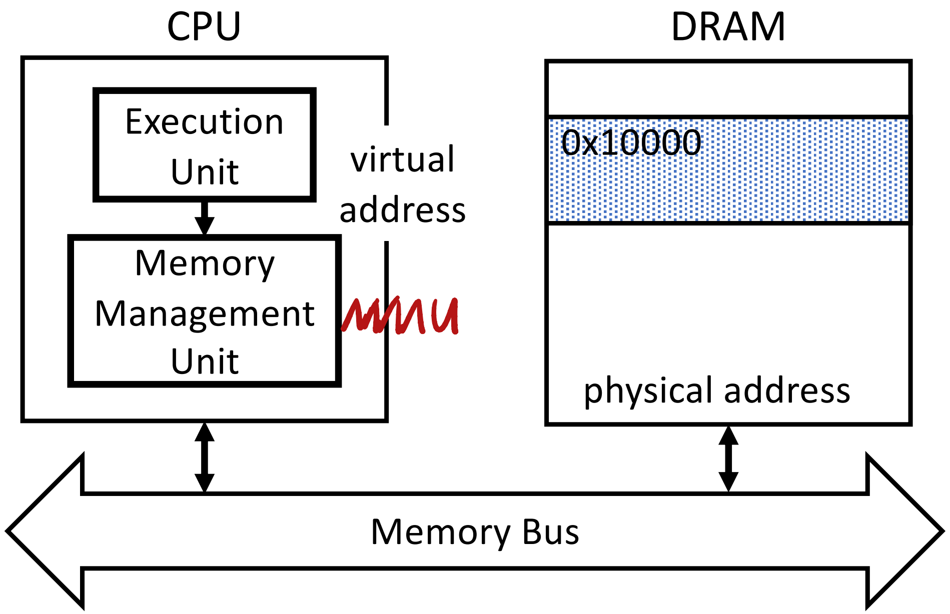
virtual addresses

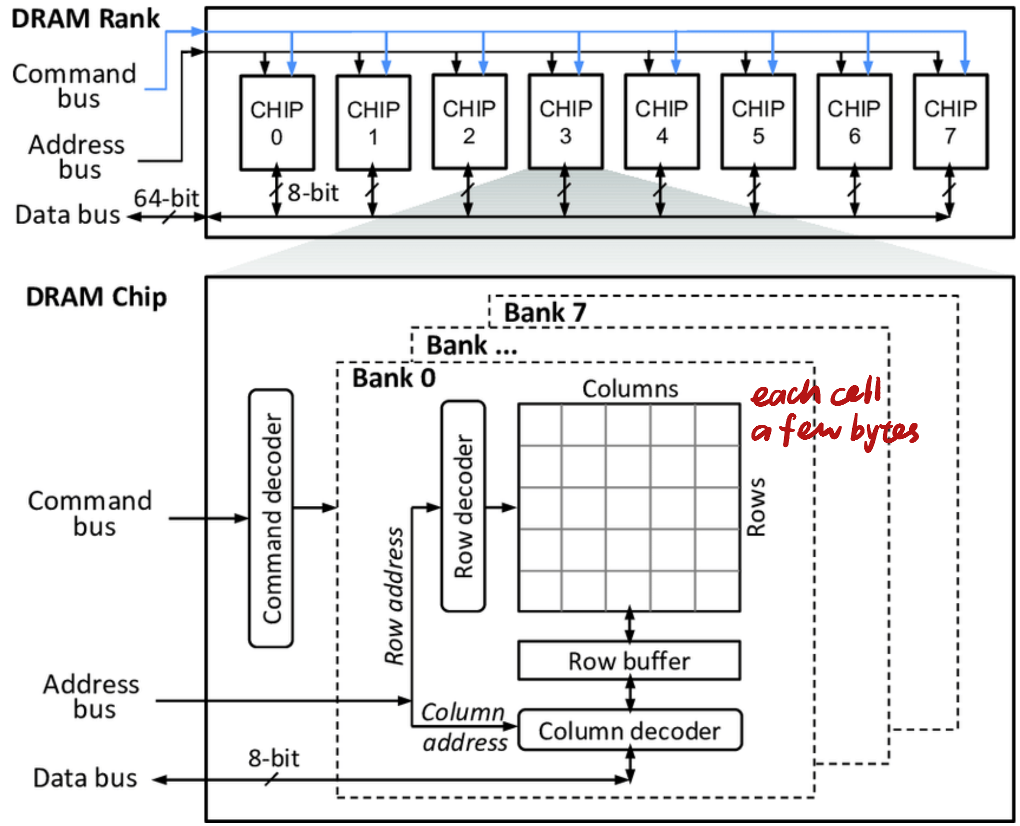
```
$ ./mem_layout
code : 0x1095afe50
heap : 0x1096008c0
stack: 0x7fff691aea64
```

begin mem

Virtual Address v.s. Physical Address

- CPU uses physical addresses to access DRAM





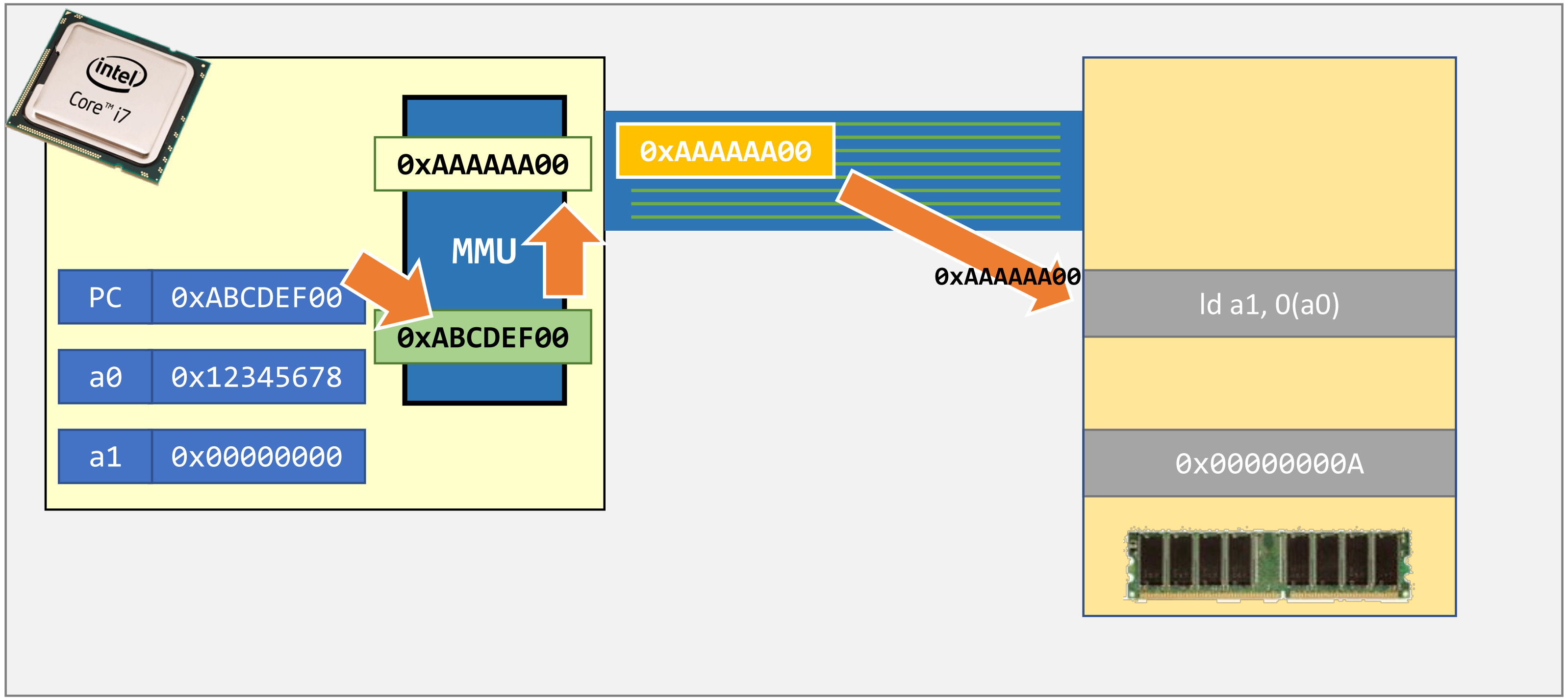
Address Translation

MMU is usually on the CPU chip, but may also be off-chip or pure software

- Coordination between CPU hardware and OS software
- Memory management unit (MMU) in CPU
  - Translate virtual address used by instruction to physical address understood by DRAM
  - CPU interposes every memory access
    - Interposition: a generic and powerful technique used in computer systems for better tranparency
- Operating system
  - Set up hardware for correct translation
  - Keep track of which locations are free and which are in use
  - Maintain control of how memory is used

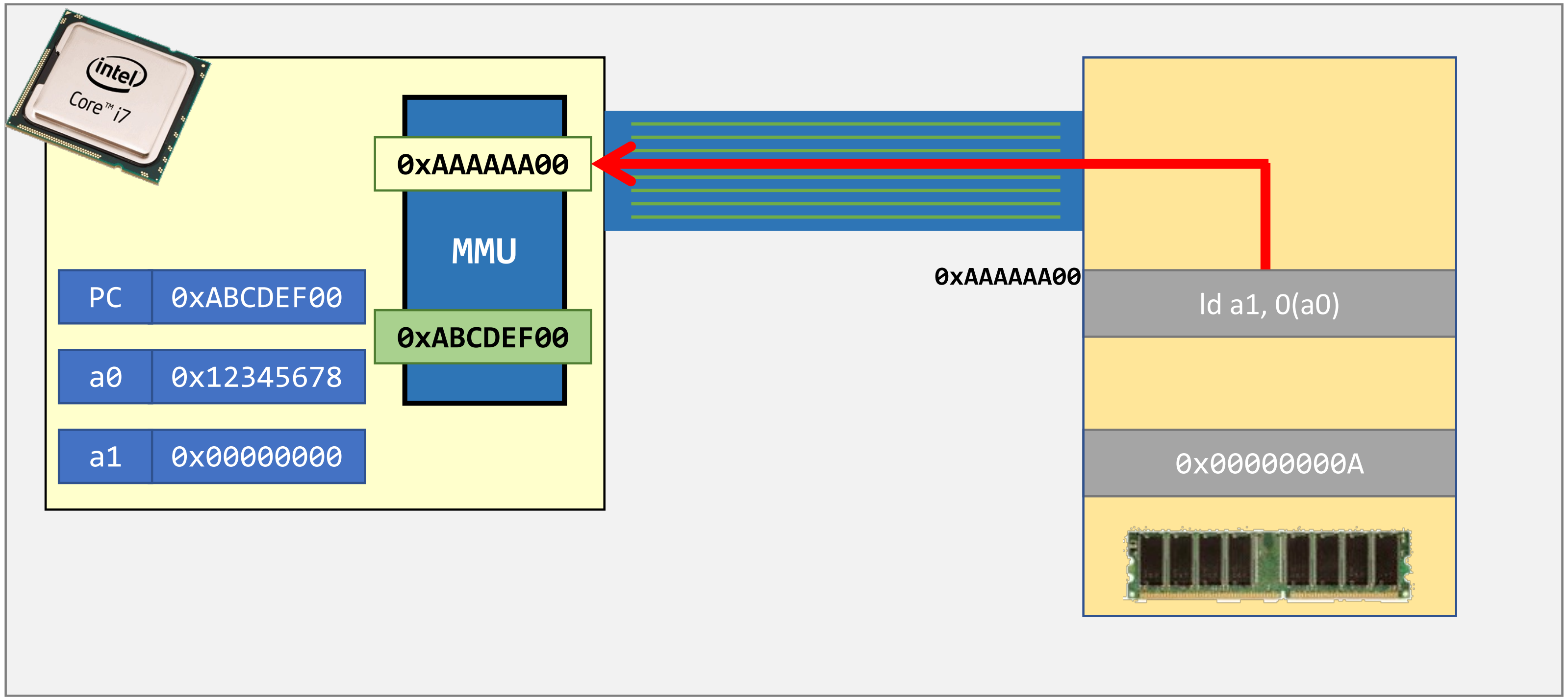
Address Translation in Action

- Step 1: Fetch instruction at virtual address 0xabcdef00



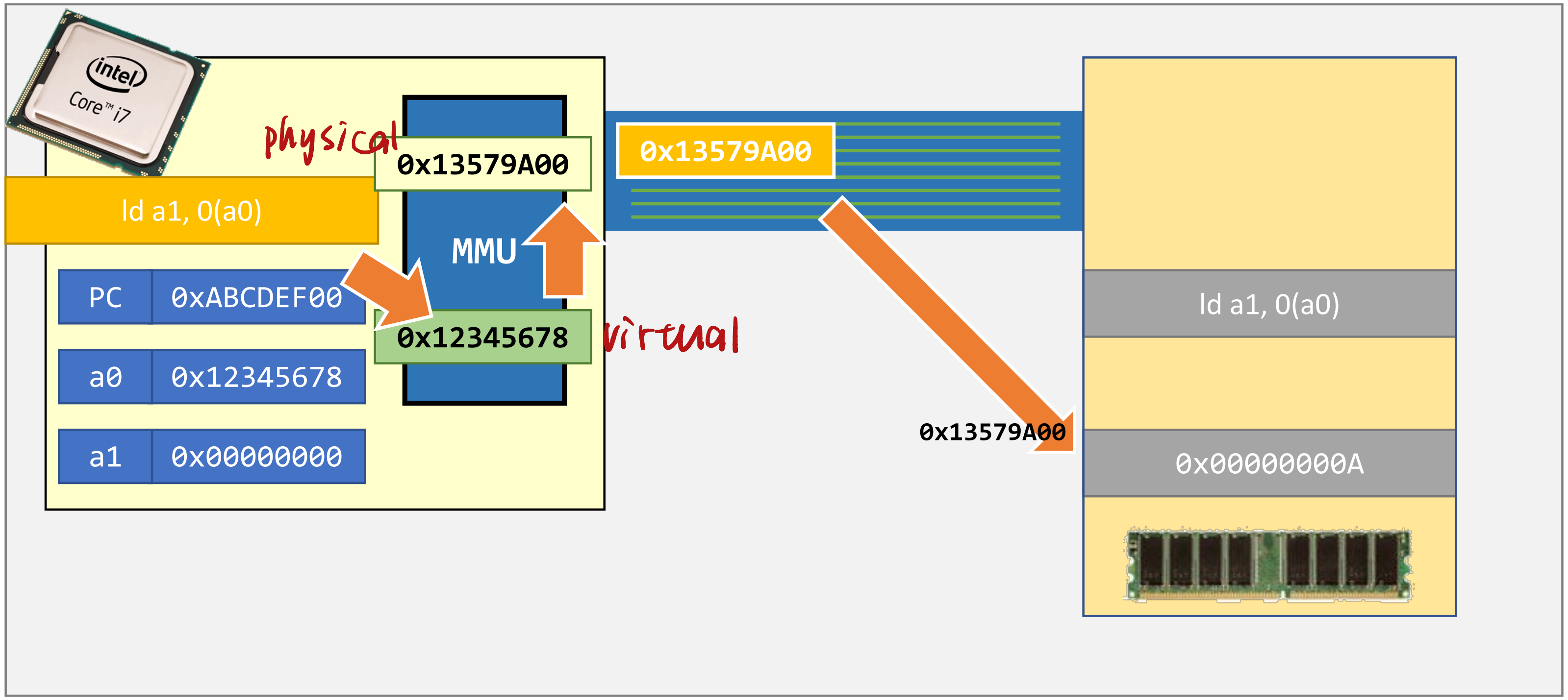
Address Translation in Action

- Step 2: Instruction fetched from physical address 0xaaaaaa00



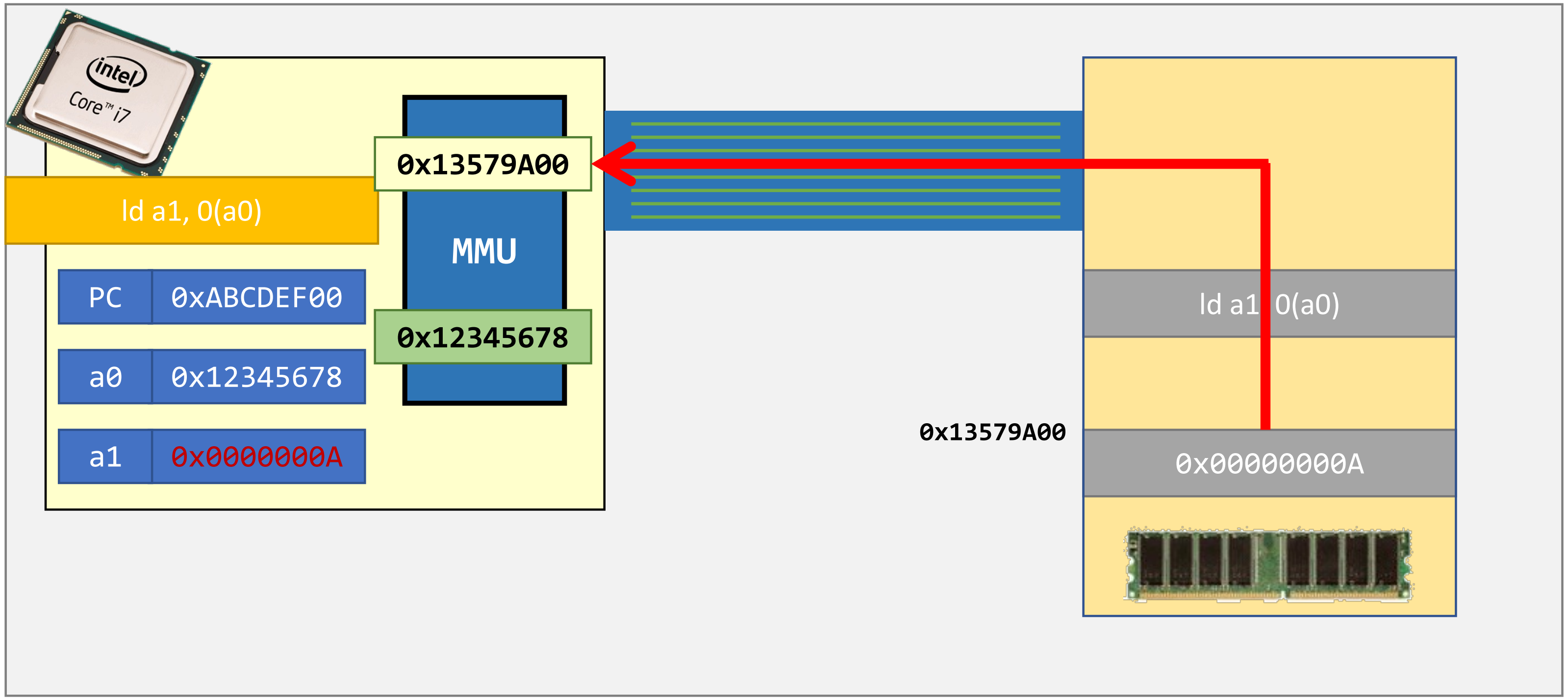
Address Translation in Action

- Step 3. CPU executes the instruction and access virtual address at 0x12345678



Address Translation in Action

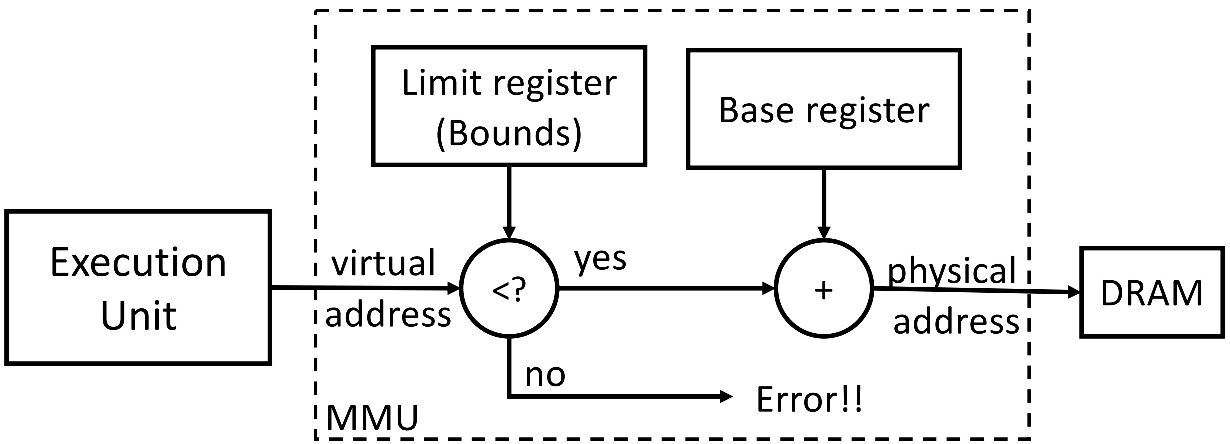
- Step 4. Data retrieved from physical address 0x13579a00 into EAX

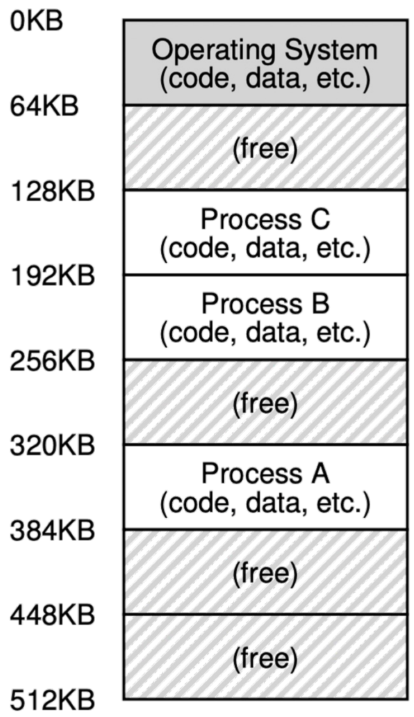


How to Translation Virtual Address to Physical Address

Base & Bounds: Dynamic Relocation

- Two hardware registers [SS74]
  - base register
  - bounds register (also called a limit register).
  - Process A, e.g., base 320KB, bounds 64KB





[SS74] "The Protection of Information in Computer Systems" by J. Saltzer and M. Schroeder. CACM, July 1974.

Hardware & OS Coordination

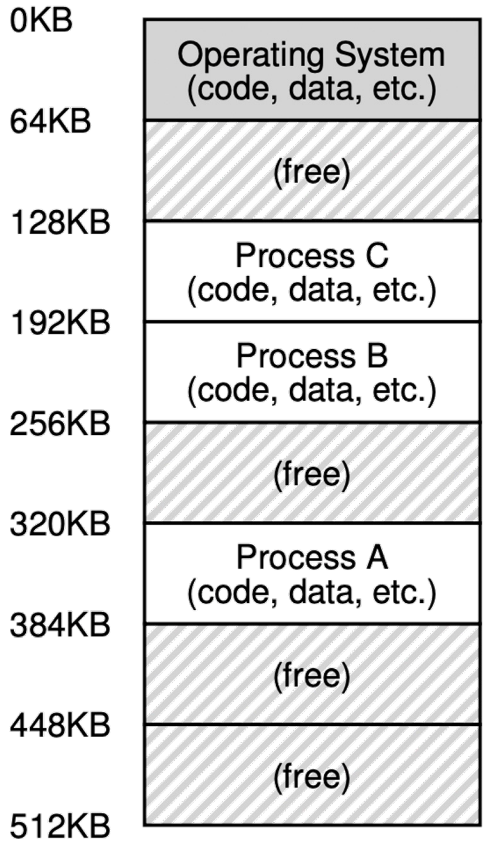
Hardware Support	Explanation
Privileged mode to update base/bounds	Needed to prevent user-mode processes from executing privileged operations to update base/bounds
Base/bounds registers	Need pair of registers per CPU to support address translation and bounds checks
Privileged instruction(s) to register exception handlers	Need to allow OS, but not the processes, to tell hardware what exception handlers code to run if exception occurs

OS Support	Explanation
Memory management	Need to allocate memory for new processes; Reclaim memory from terminated processes; manage memory via free list
Base/bounds management	Must set base/bounds properly upon context switch
Exception handling	Code to run when exceptions arise; likely action is to terminate offending process

Limitations of Base & Bounds

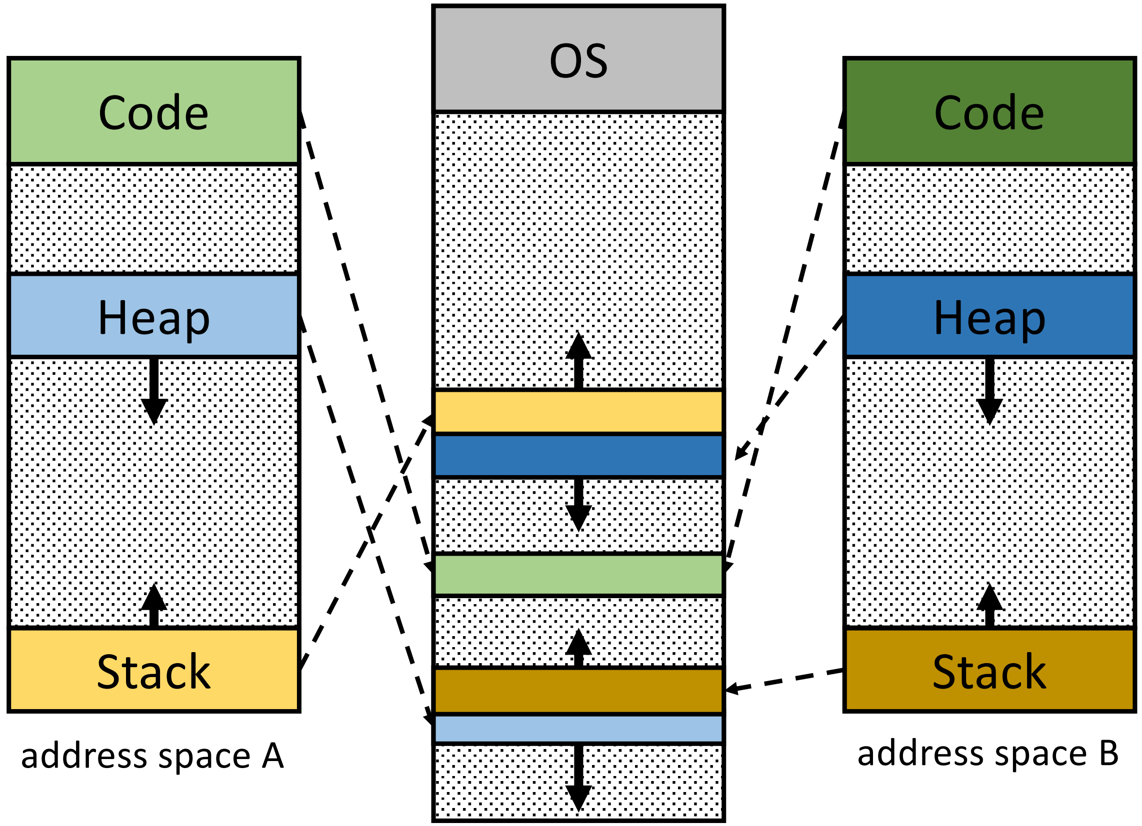
- Internal fragmentation
  - wasted memory between heap and stack
- Cannot support larger address space
  - Address space equals the allocated slot in memory
  - example: Process C's address space is at most 64KB
- Hard to do inter-process sharing
  - Want to share code segments when possible
  - Want to share memory between processes
  - example: Process A & C cannot share memory

?



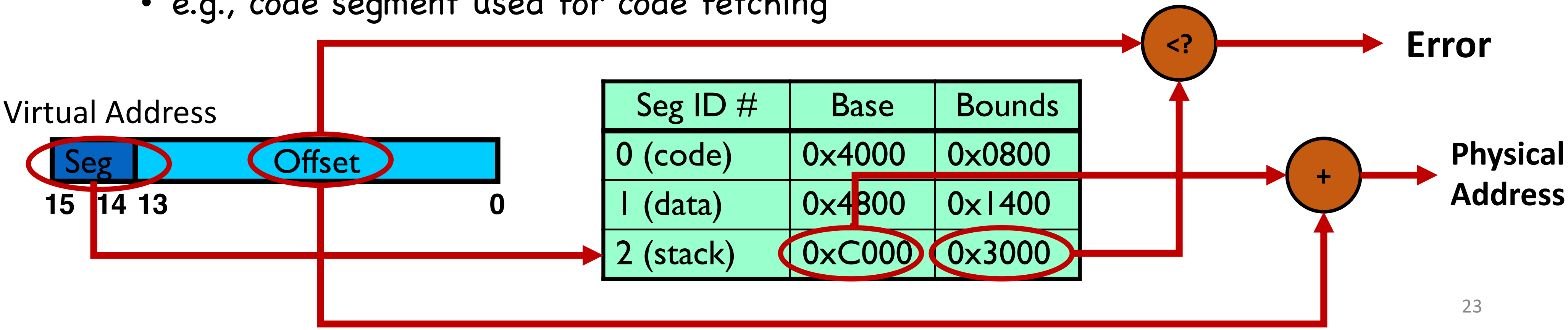
Segmentation: Generalized Base/Bounds

- A pair of base/bounds registers for each segment
  - code, stack, heap
- Each segment mapped to a different region of the physical memory
  - internal fragmentation?
  - larger address space?
  - inter-process sharing?



Segmentation: Implementation

- Base/bounds registers organized as a table
  - Segment ID used to index the base/bounds pair
  - Base added to offset (of virtual address) to generate physical address
  - Error check catches offset (of virtual address) out of range
- Use segments explicitly
  - **Segment addressed by top bits of virtual address**
  - or, x86-32 mov [es:bx],ax.
- Use segments implicitly
  - e.g., code segment used for code fetching



Seg ID #	Base	Bounds
0 (code)	0x4000	0x0800
1 (data)	0x4800	0x1400
2 (stack)	0xC000	0x3000

More about Segmentation

- Memory sharing with segmentation
  - Code sharing on modern OS is very common
  - If multiple processes use the same program code or library code
    - Their address space may overlap in the physical memory
    - The cooresponding segments have the same base/bounds
  - Memory sharing needs memory protection
- Memory protection with segmentation
  - Extend base/bounds register pair
  - Read/Write/Execute permission

Seg ID	Base	Bounds	protection
0 (code)	0x4000	0x0800	Read-Execute
1 (data)	0x4800	0x1400	Read-Write
2 (stack)	0xC000	0x3000	Read-Write

Problems with Segmentation

- OS context switch must also save and restore all pairs of segment registers

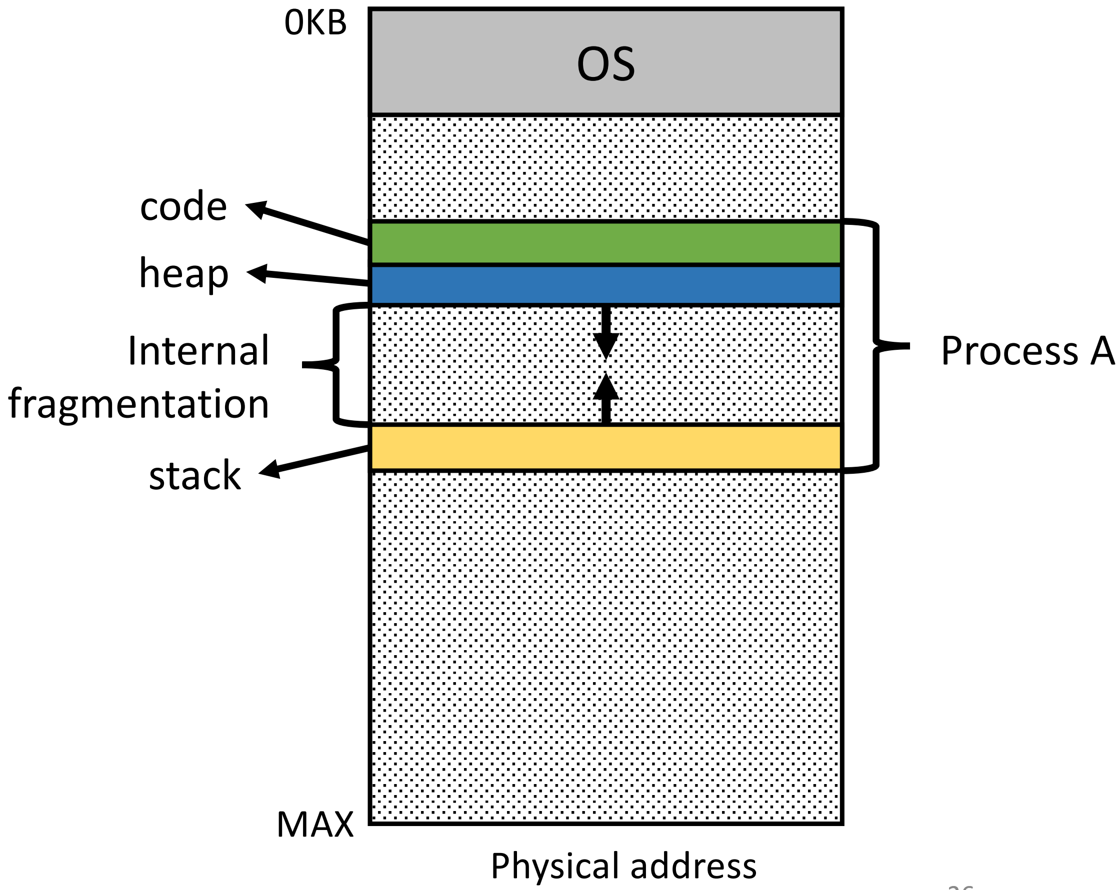
- A segment may grow, which may or may not be possible

- Management of free spaces of physical memory with variable-sized segments

- **External fragmentation**: free gaps between allocated segments
  - Segmentation may also have internal fragmentation if more space allocated than needed.

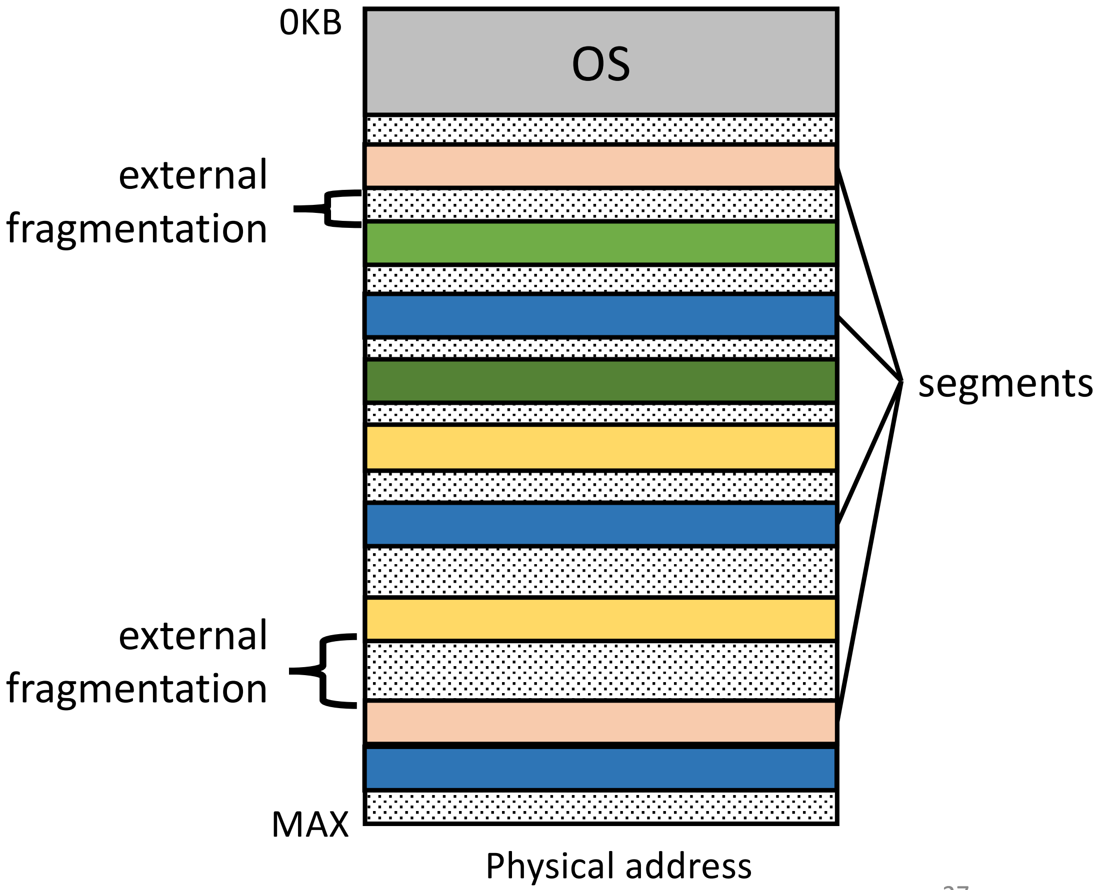
Fragmentation Illustrated

- Internal fragmentation with Base & Bounds
- Space between heap and stack may be wasted



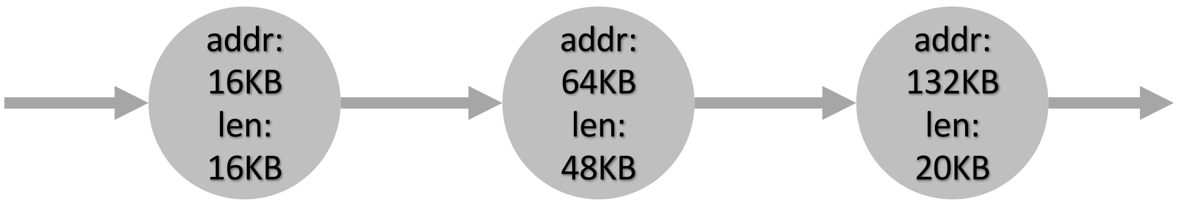
Fragmentation Illustrated (Cont'd)

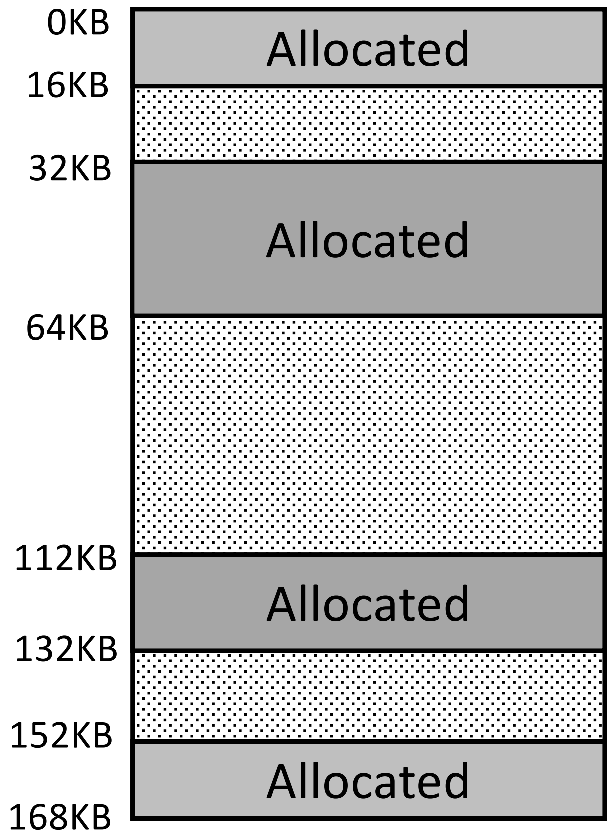
- External fragmentation with segmentation
- free spaces are curved into small chunks
  - each is too small for further allocation
  - added together could be a huge waste



Memory Allocation

- OS needs to manage all free physical memory regions
- A basic solution is to maintain a linked list of free slots
- An ideal allocation algorithm is both **fast** and **minimizes fragmentation**.

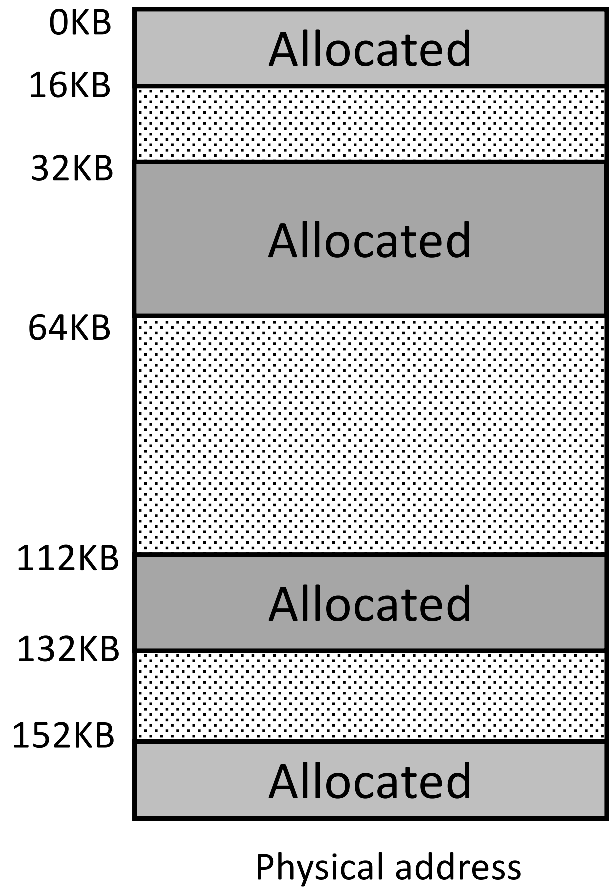




Physical address

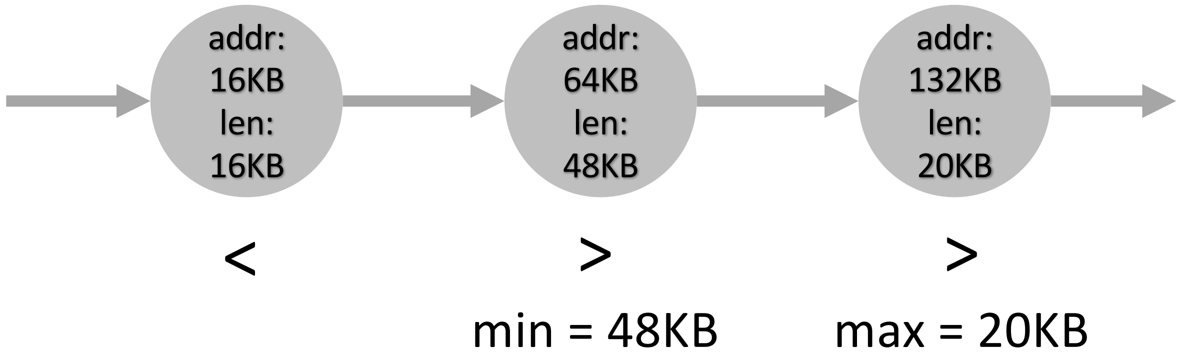
Basic Strategies: Best Fit

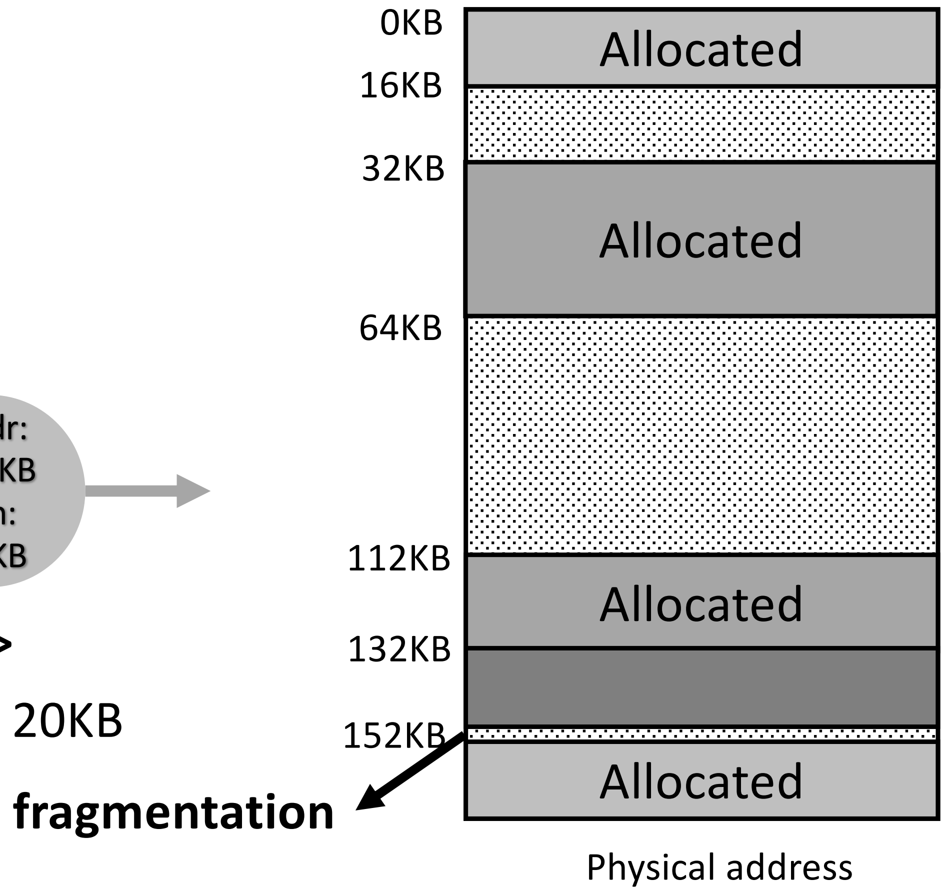
- Idea
  - search through the free list and find chunks of free memory that are as big or bigger than the requested size.
  - return the one that is the **smallest** in that group of candidates;
- Pros
  - Satisfy the request with minimal external fragmentation
- Cons
  - exhaustive search is slow



Basic Strategies: Best Fit (Cont'd)

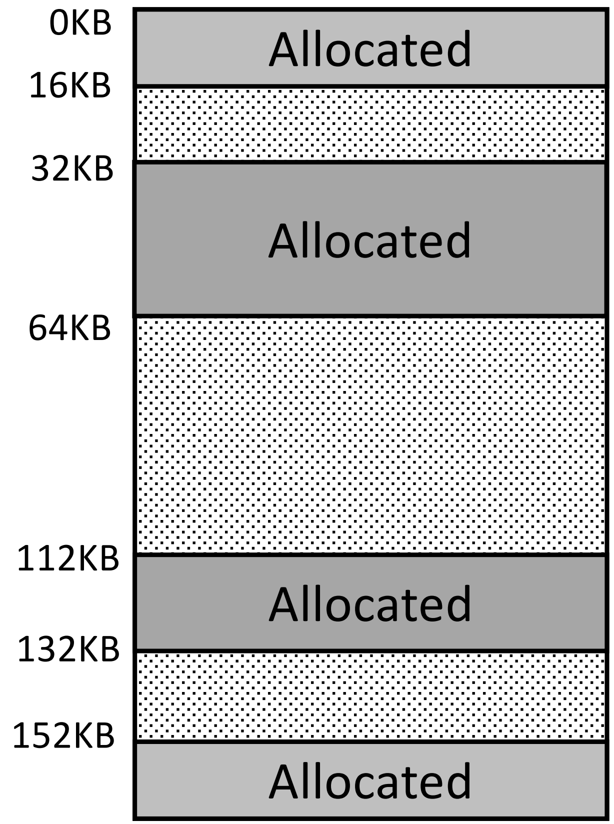
- Example
  - Requested space is 18KB
  - Allocated at 132KB





Basic Strategies: Worst Fit

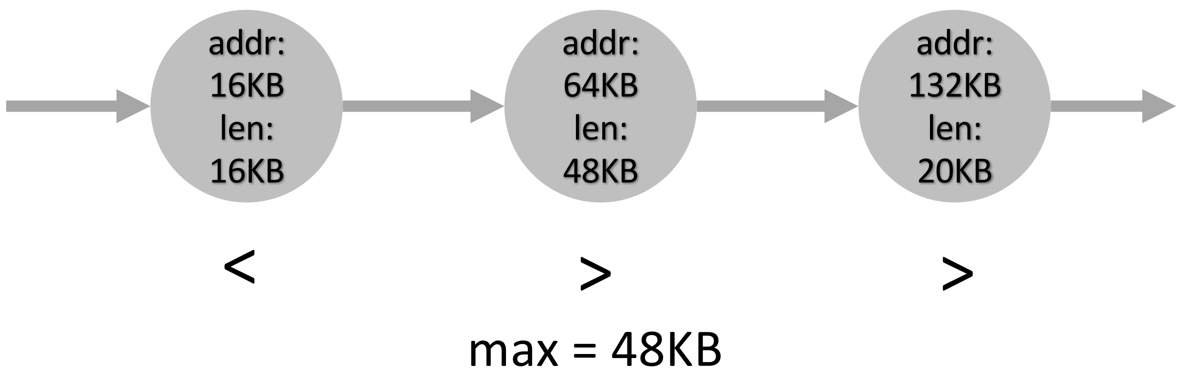
- Idea
  - search through the free list and find chunks of free memory that are as big or bigger than the requested size.
  - return the one that is the **largest** in that group of candidates;
- Pros
  - Leaves larger "holes" in physical memory
- Cons
  - exhaustive search is slow
  - severe fragmentation in practice

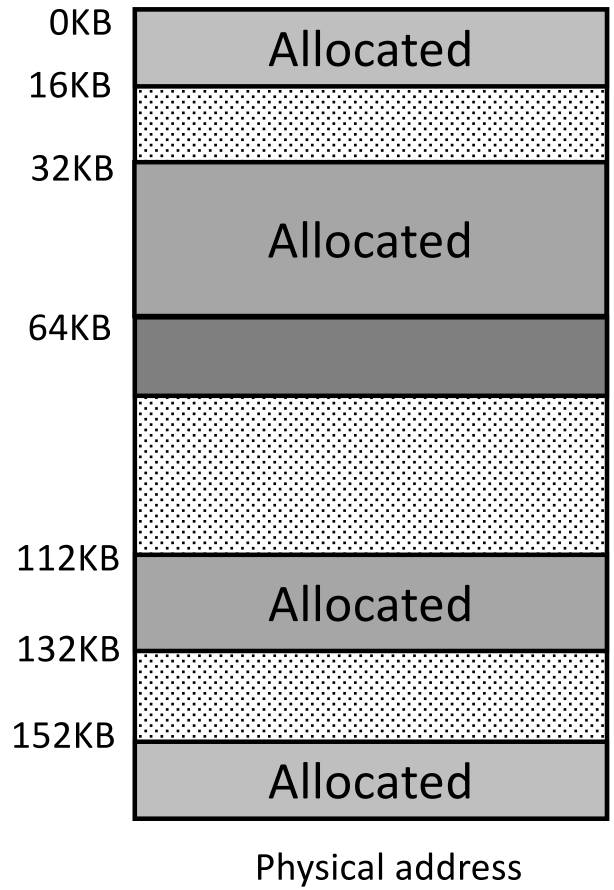


Physical address

Basic Strategies: Worst Fit (Cont'd)

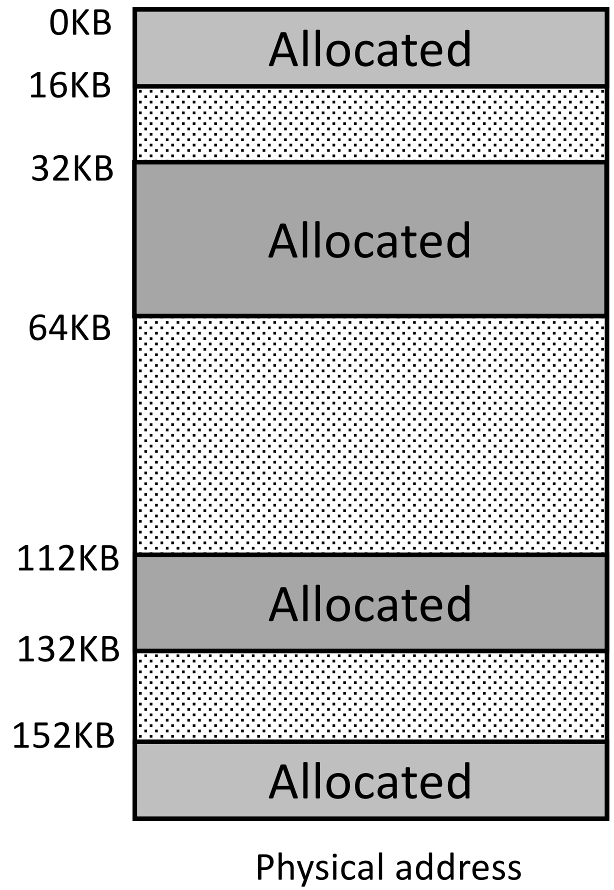
- Example
  - Requested space is 18KB
  - Allocated at 64KB





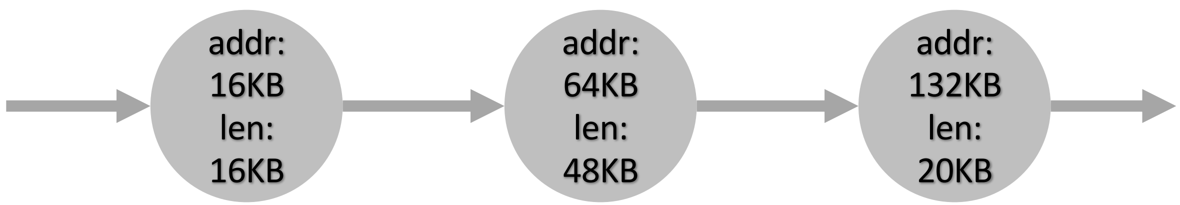
Basic Strategies: First Fit

- Idea
  - find the first block that is big enough and returns the requested size
- Pros
  - Fast
- Cons
  - pollutes the beginning of the free list with small chunks

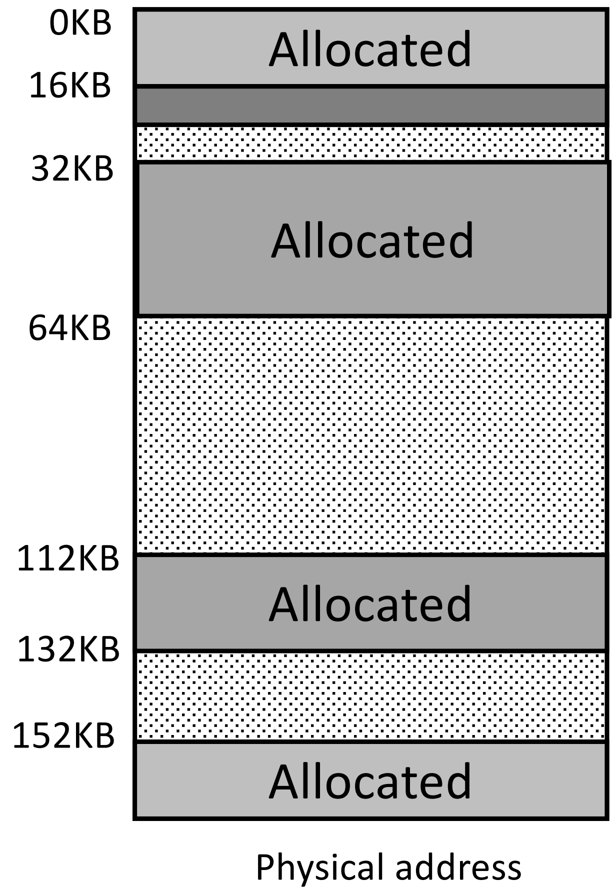


Basic Strategies: First Fit (Cont'd)

- Example
  - Requested space is 8KB
  - Allocated at 16KB



>, stop



Summary

- Address space is a key abstraction of OS
- Address translation requires hardware/software coorperation
- Two schemes so far: (1) Base & Bounds (2) segmentation
- Internal/external fragmentation is an issue
- Best/worst/first fit, no best option

Thank you!