Memory Management

Memory Management

» Paraphrase of Parkinson's Law, "Programs expand to fill the memory available to hold them."

» Average home computer nowadays has 10,000 times more memory than the IBM 7094, the largest computer in the world in the early 1960s

Memory Hierarchy

- How does the operating system create abstractions from memory, and how does it manage them?
- Memory hierarchy:
  - Few megabytes of very fast, very expensive, volatile cache memory
  - A few gigabytes of medium-speed, medium priced, volatile main memory
  - A few terabytes of slow, cheap, nonvolatile magnetic or solid state disk storage.

Memory Hierarchy

- Memory Manager manages it
  - Tracks which parts of memory are in use
  - Allocates memory to processes
  - Deallocates memory when processes are done.

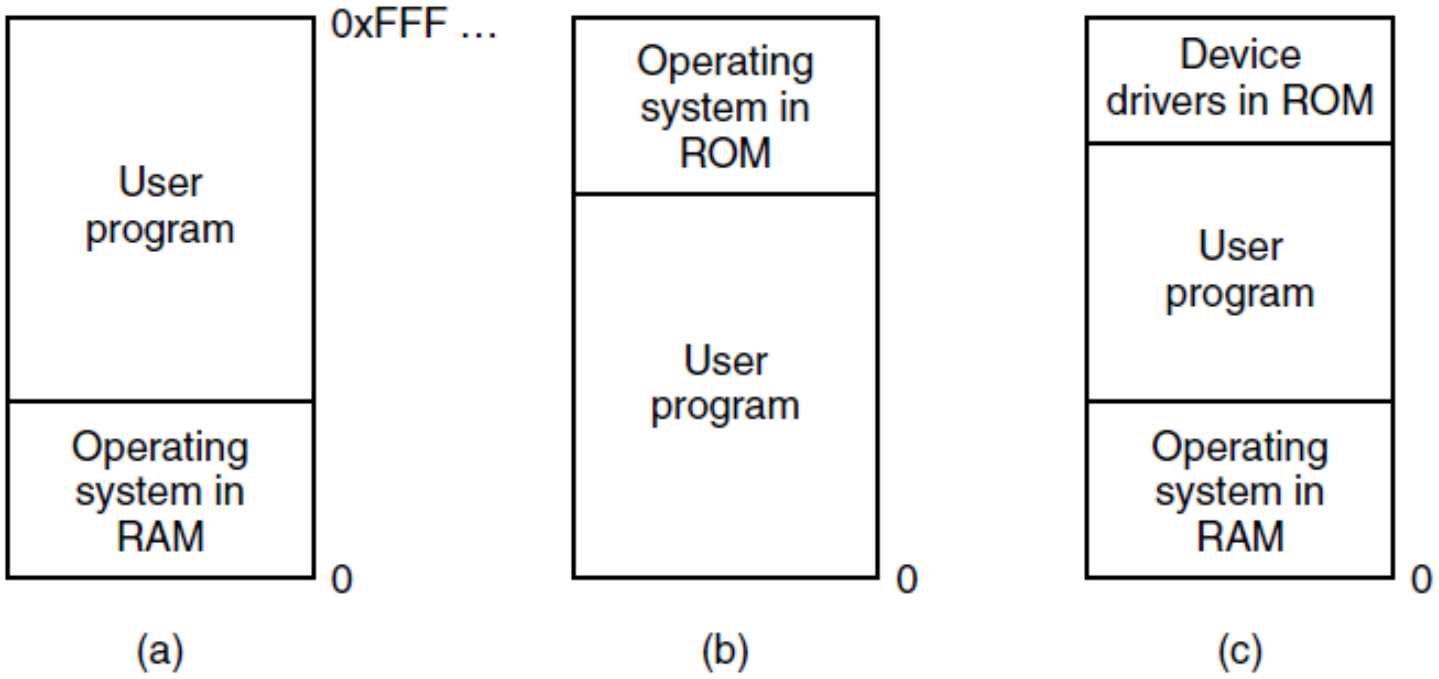
No Memory Abstraction

- Early mainframes (before 1960), early minicomputers (before 1970), early personal computers (before 1980) had no memory abstraction
- Every program saw physical memory.
- Programmers saw a set of addresses from 0 to some maximum.

No Memory Abstraction

- Two programs could not be resident in memory at the same time.
  - Address conflicts
  - Could run multi-threaded since threads share address space.
  - Limited use. Users want unrelated programs to be running at once.

Memory Layout

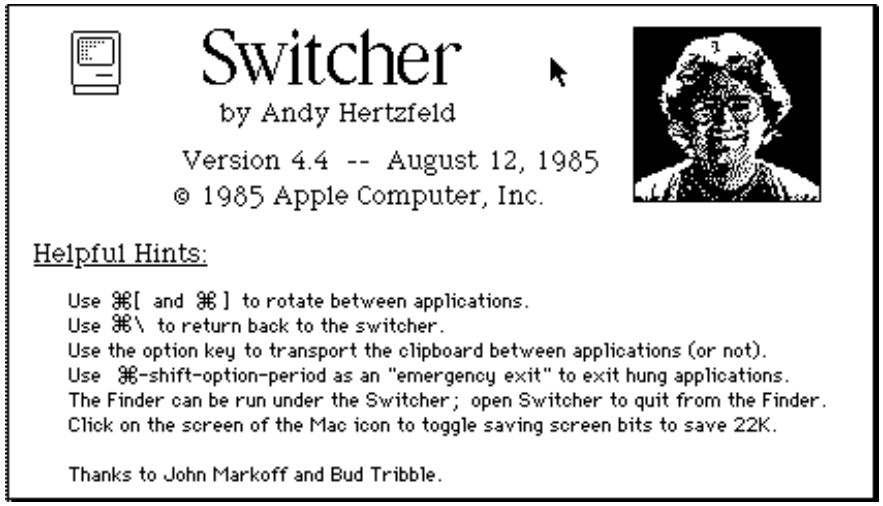


No Memory Abstraction

- Multiple programs possible
  - Save entire contents of memory to a disk file
  - Bring in new and run it
  - Swapping.
- Additional hardware will allow concurrency without swapping.

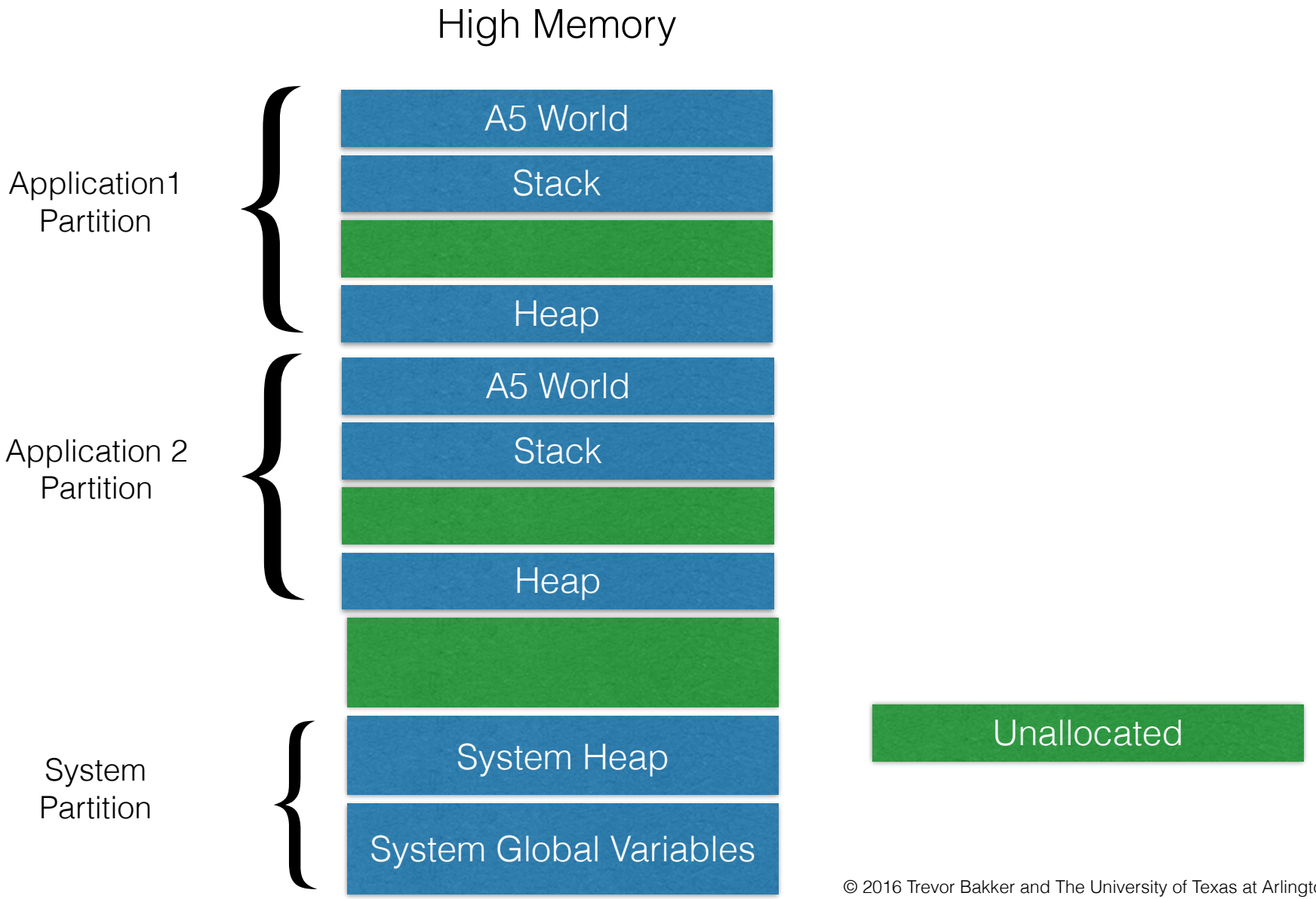
Switcher

- Switcher worked by designating a number of fixed "slots" in memory
    - Applications could be loaded into those slots
    - The user could then switch between these applications by clicking a button above the menu bar. The current application would horizontally slide out of view, and the next one would slide in.
    - Worked with existing memory management so changes were transparent to the OS and to the running programs



http://www.folklore.org/StoryView.py?project=Macintosh&story=Switcher.txt

Switcher Memory Layout



Address Spaces

- Exposing memory to processes have several drawbacks
  - If user programs can address every byte of memory, the OS can be trashed intentionally or by accident.
  - Difficult to have multiple programs running at once

Address Spaces

- Abstract memory
  - Process creates a virtual CPU abstraction
  - Address space is abstract memory for a process to live in
- Address space: Set of all addresses that a process can see to address memory
- Each process has an independent address space

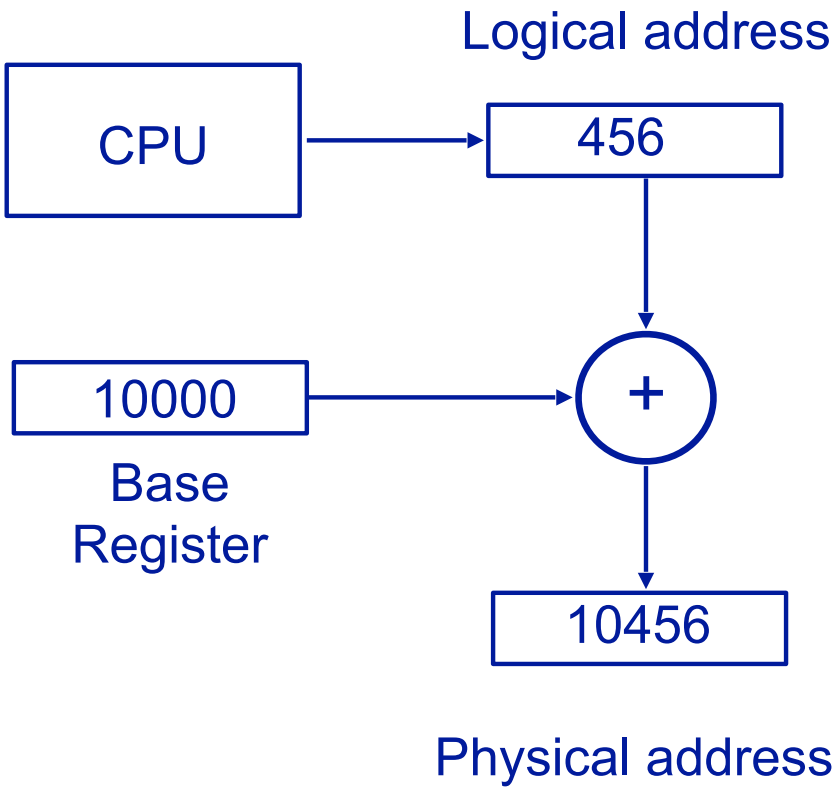
Address Spaces

- Need to map address 28 in one process and address 28 in another process to a separate physical address.
- Dynamic relocation
- Older solution: two special registers
  - Base register
  - Limit register

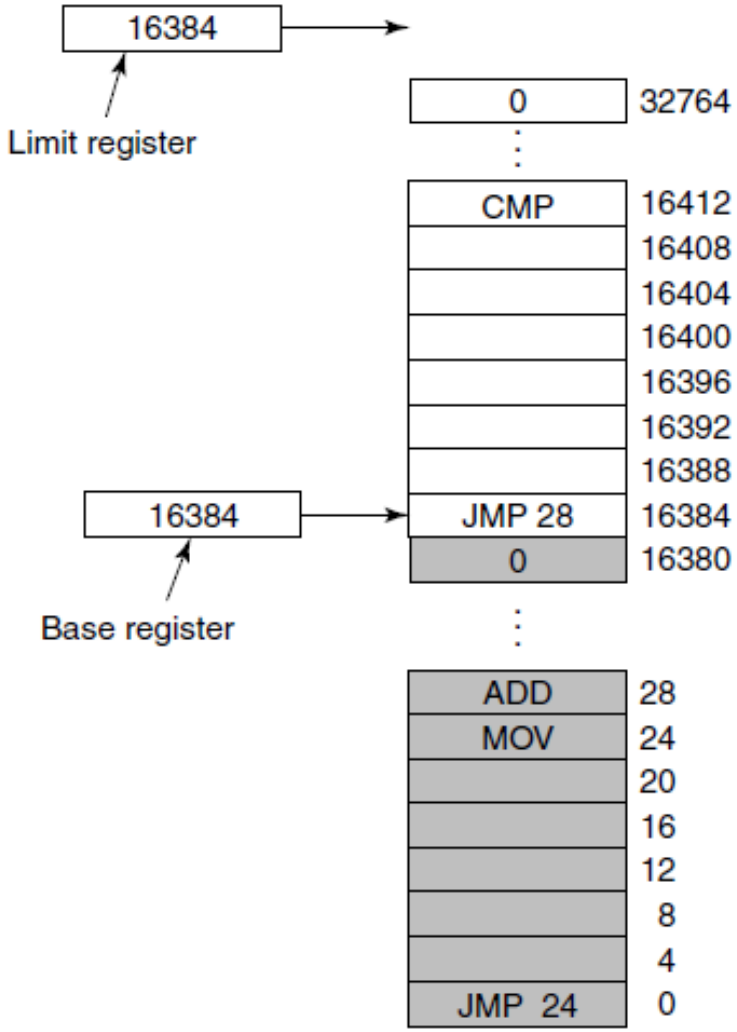
Physical v. Logical Addresses

- Originally programs were compiled to reference addresses that corresponded one to one with physical memory addresses.
- Unknowingly using concept of logical and physical memory
  - Logical addresses - Set of addresses the CPU generates as the program is executed.
  - Physical addresses - Set of addresses used to reference physical memory.

Dynamic Relocation



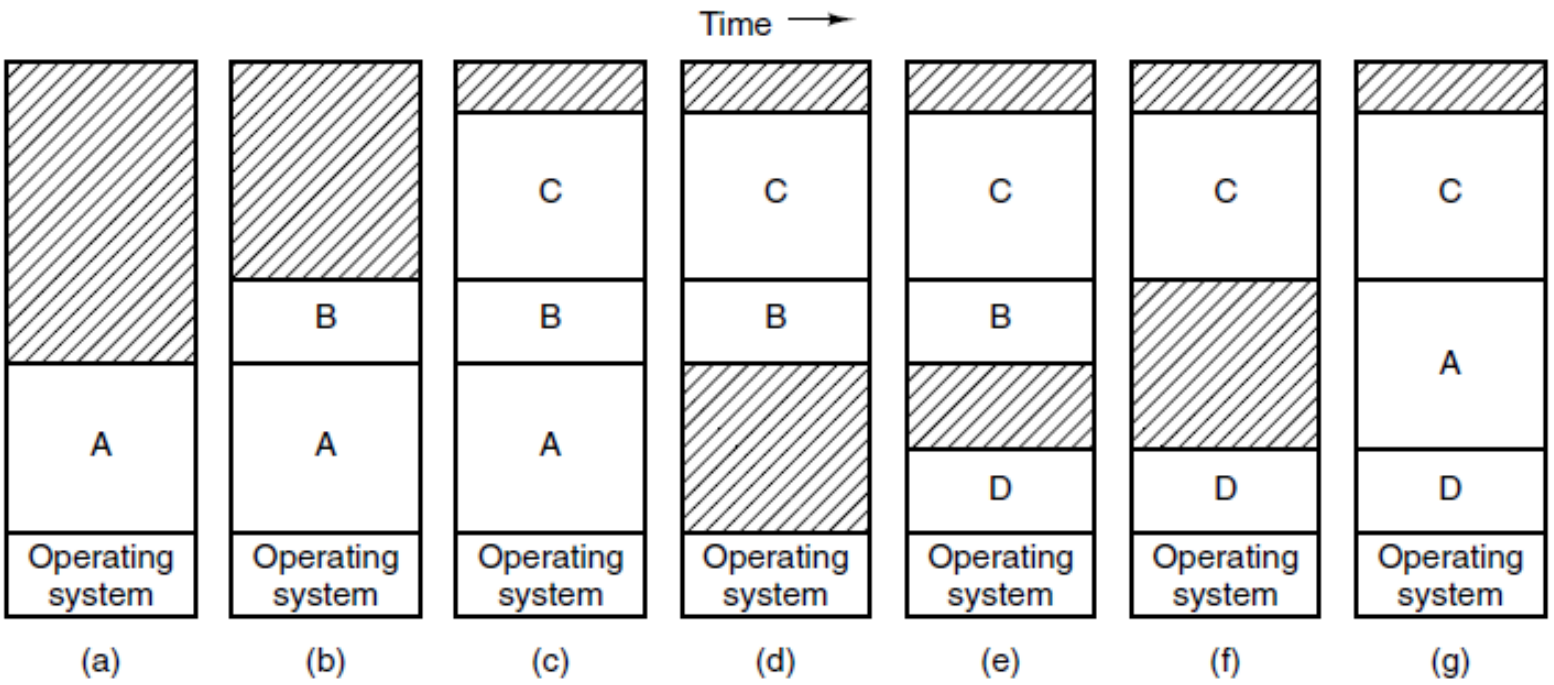
Dynamic Relocation



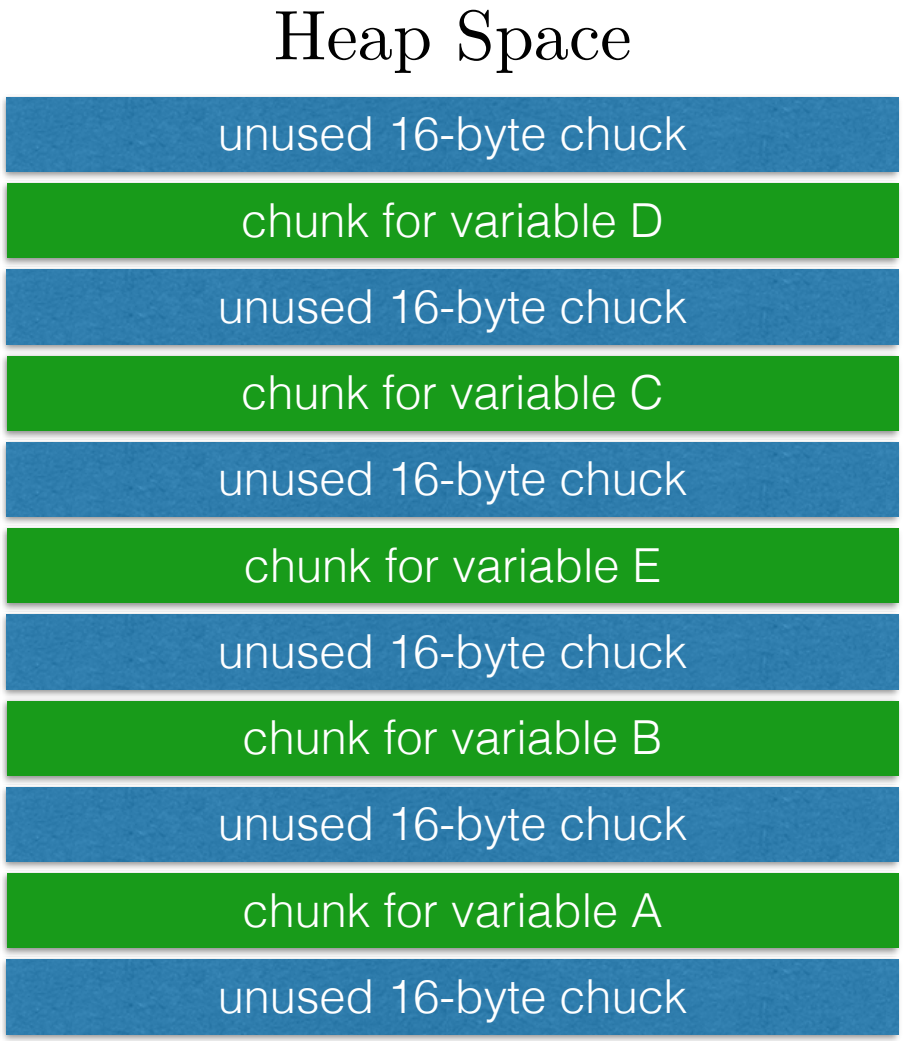
Swapping

- If physical memory is large enough to hold all the processes, the previous schemes will do
- In practice, total amount of RAM is more that can fit in memory.
- Two approaches:
  - Swapping: Bring in each process in its entirety, run it for a while, then put it back onto disk
  - Virtual Memory: Run processes when they are partially in memory

Swapping


Time
C
B
A
D
Operating system
(a)
(b)
(c)
(d)
(e)
(f)
(g)
Software or Hardware relocation needed

Heap Fragmentation



96 bytes of free memory but we can't allocate any chuck larger than 16 bytes

External Fragmentation

Compaction

- The operating system can reorganize the fragmented space so that the free space is contiguous.
- Expensive

Allocation

- How much should the OS allocate?
  - If processes can't grow, then allocate exactly the amount needed
  - If the process can grow, and there is a hole next to the process, it can be allocated and the process grow into it.

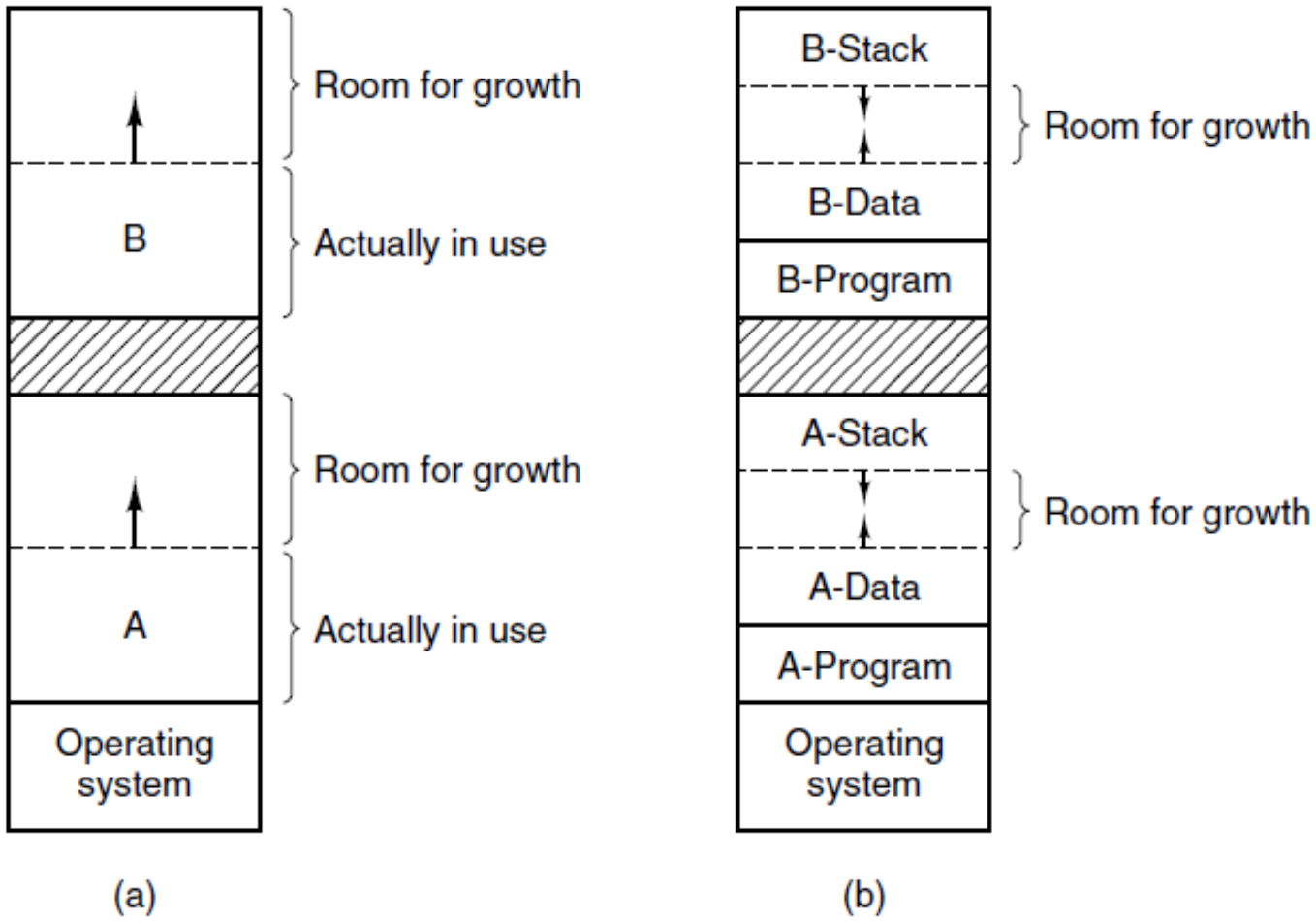
Allocation

- How much should the OS allocate?
  - If processes doesn't have an adjacent hole, it will have to move to a hole in memory large enough or
  - Swap out one or more processes.
  - If it can't grow and no free swap space, suspend or kill the process

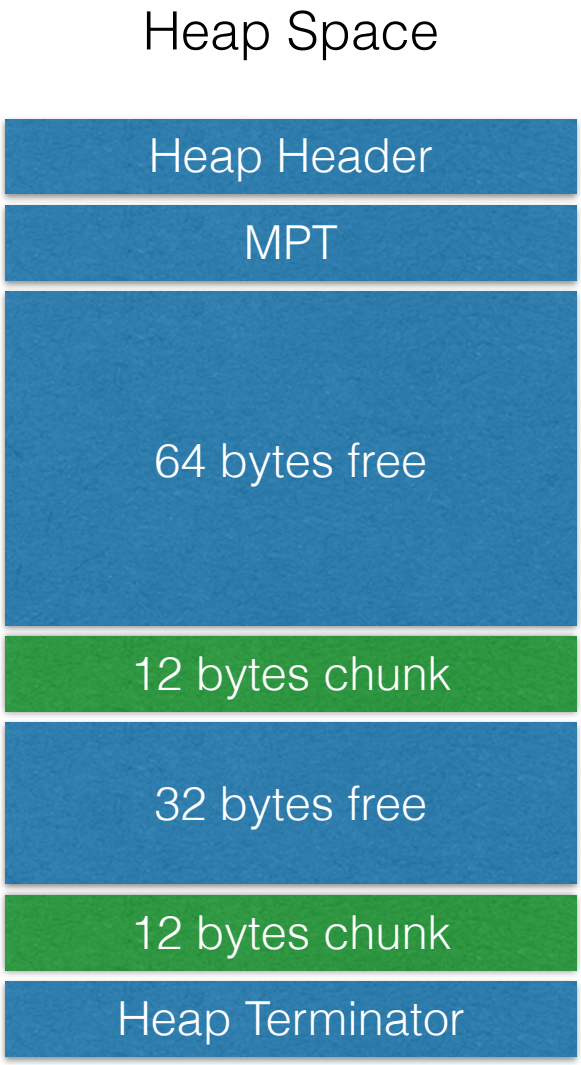
Allocation

- Good idea to allocate a little extra when a process is swapped in to reduce the overheard with moving or swapping processes.
- Only memory used should be swapped, otherwise wasteful.

Allocation


Room for growth
B
Actually in use
Room for growth
A
Actually in use
Operating system
(a)
B-Stack
Room for growth
B-Data
B-Program
A-Stack
Room for growth
A-Data
A-Program
Operating system
(b)

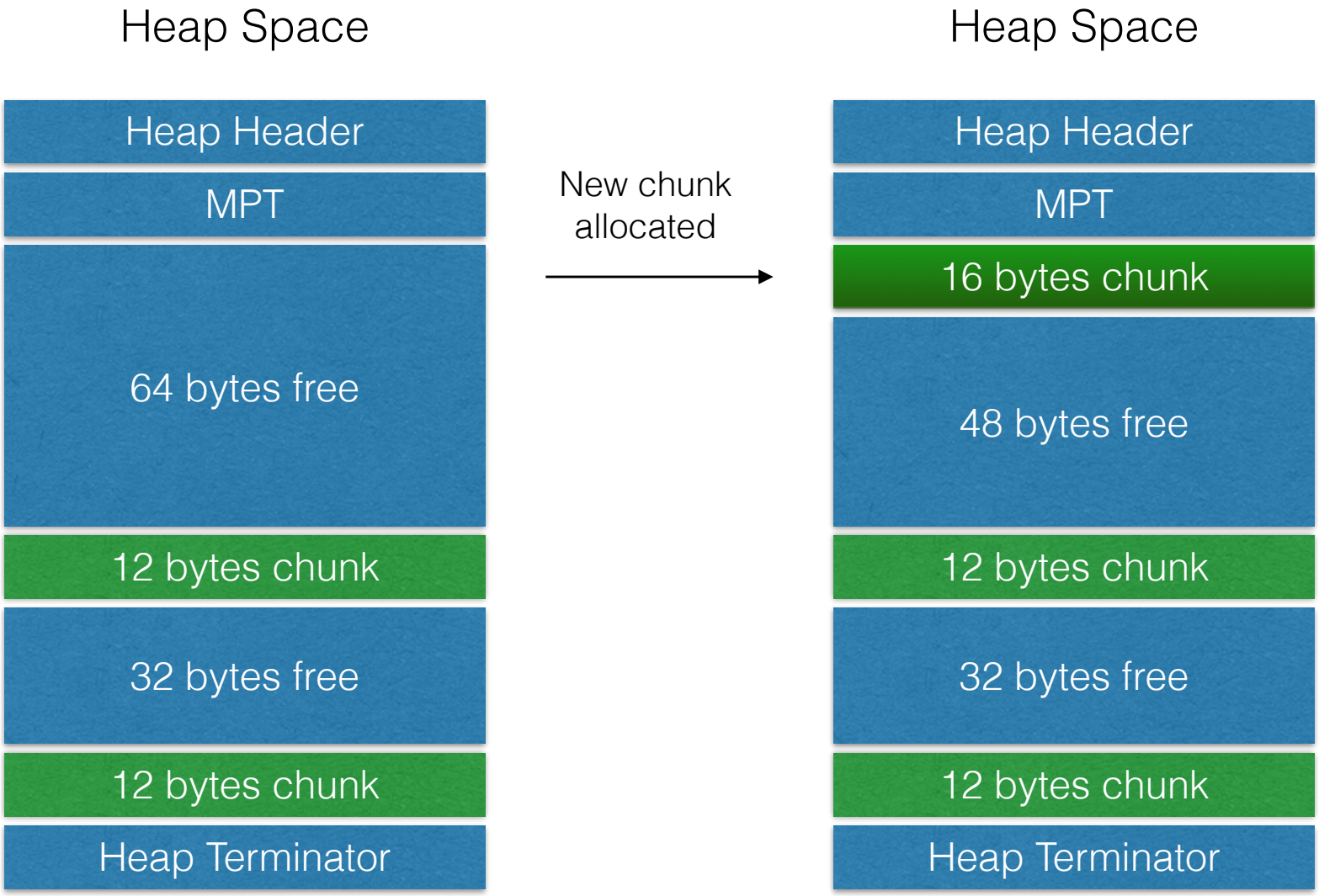
Allocating Memory



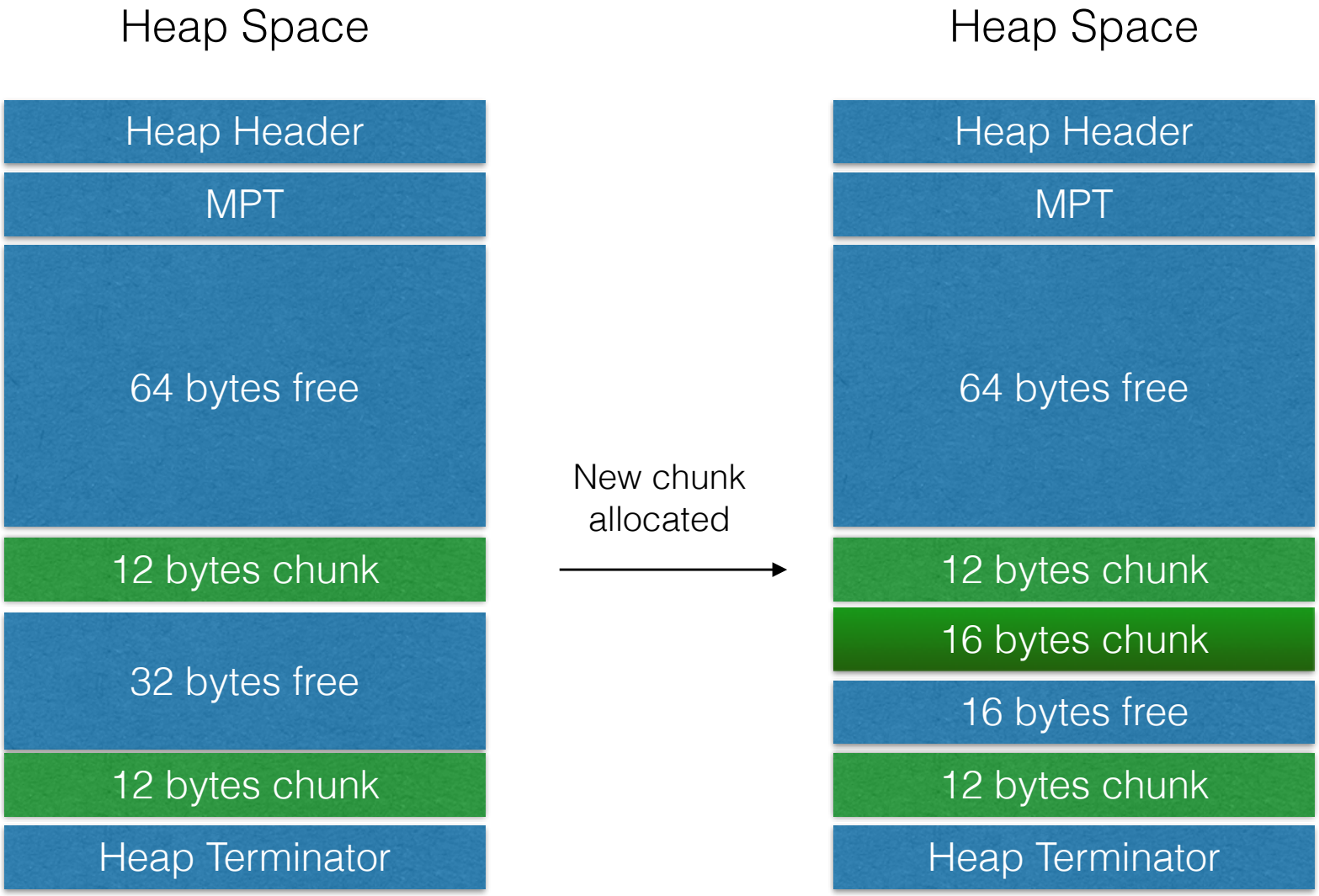
Example: We want to allocate a new 16 byte block.

Which free block does the OS choose?

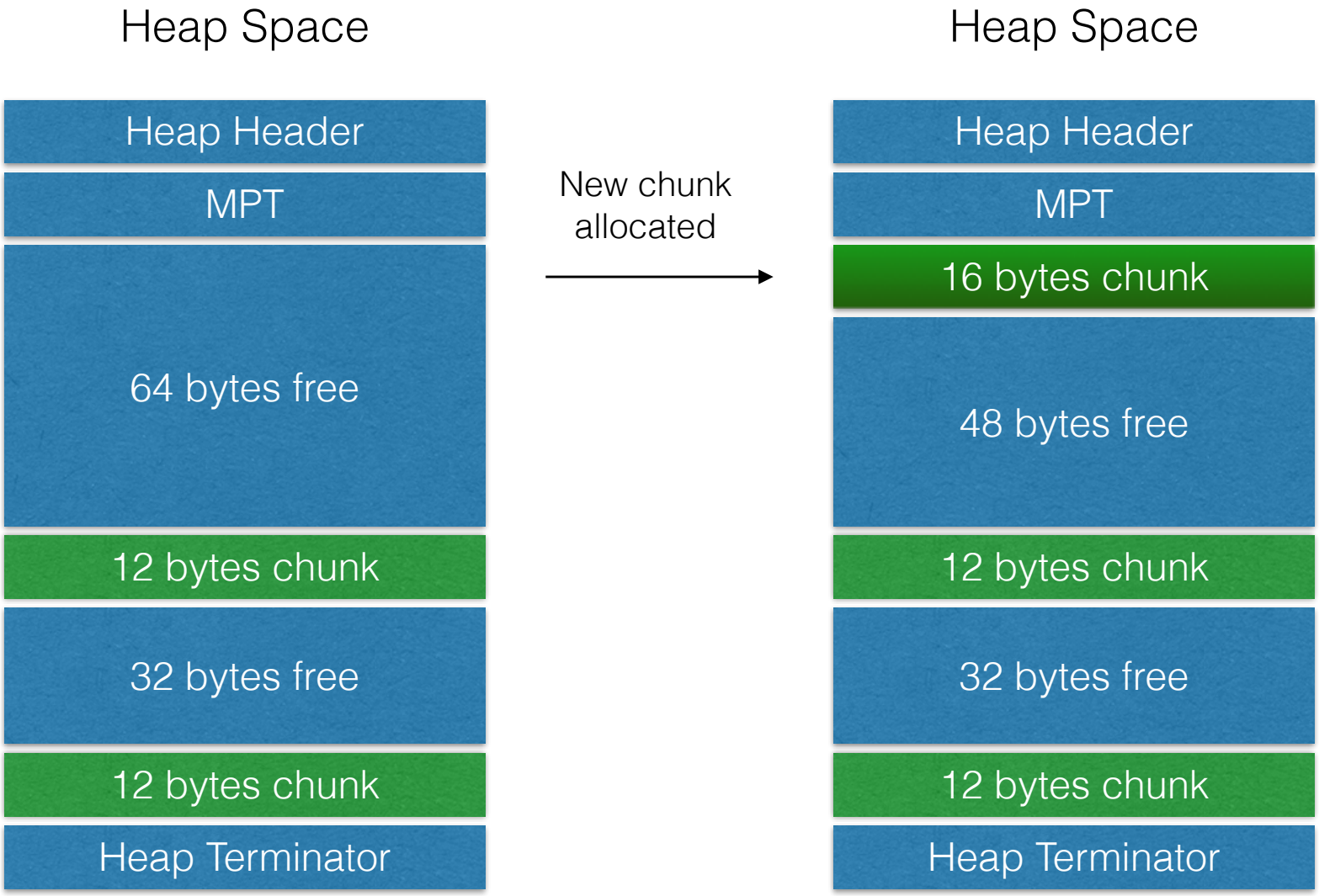
First / Next Fit



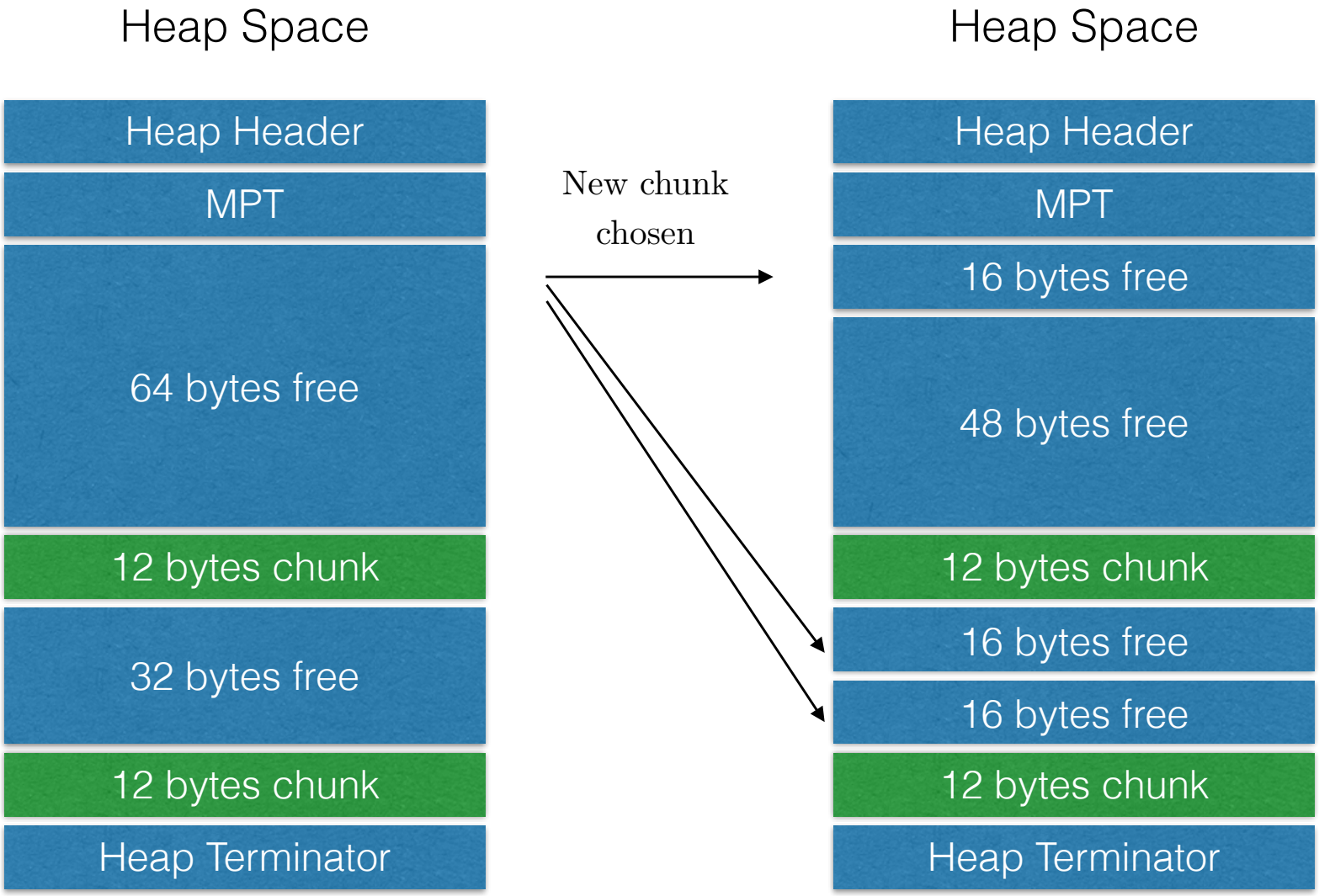
Best Fit


Heap Space
Heap Header
MPT
64 bytes free
12 bytes chunk
32 bytes free
12 bytes chunk
Heap Terminator
New chunk allocated
Heap Space
Heap Header
MPT
64 bytes free
12 bytes chunk
16 bytes chunk
16 bytes free
12 bytes chunk
Heap Terminator

Worst Fit


Heap Space
Heap Header
MPT
64 bytes free
12 bytes chunk
32 bytes free
12 bytes chunk
Heap Terminator
New chunk allocated
Heap Space
Heap Header
MPT
16 bytes chunk
48 bytes free
12 bytes chunk
32 bytes free
12 bytes chunk
Heap Terminator

Quick Fit



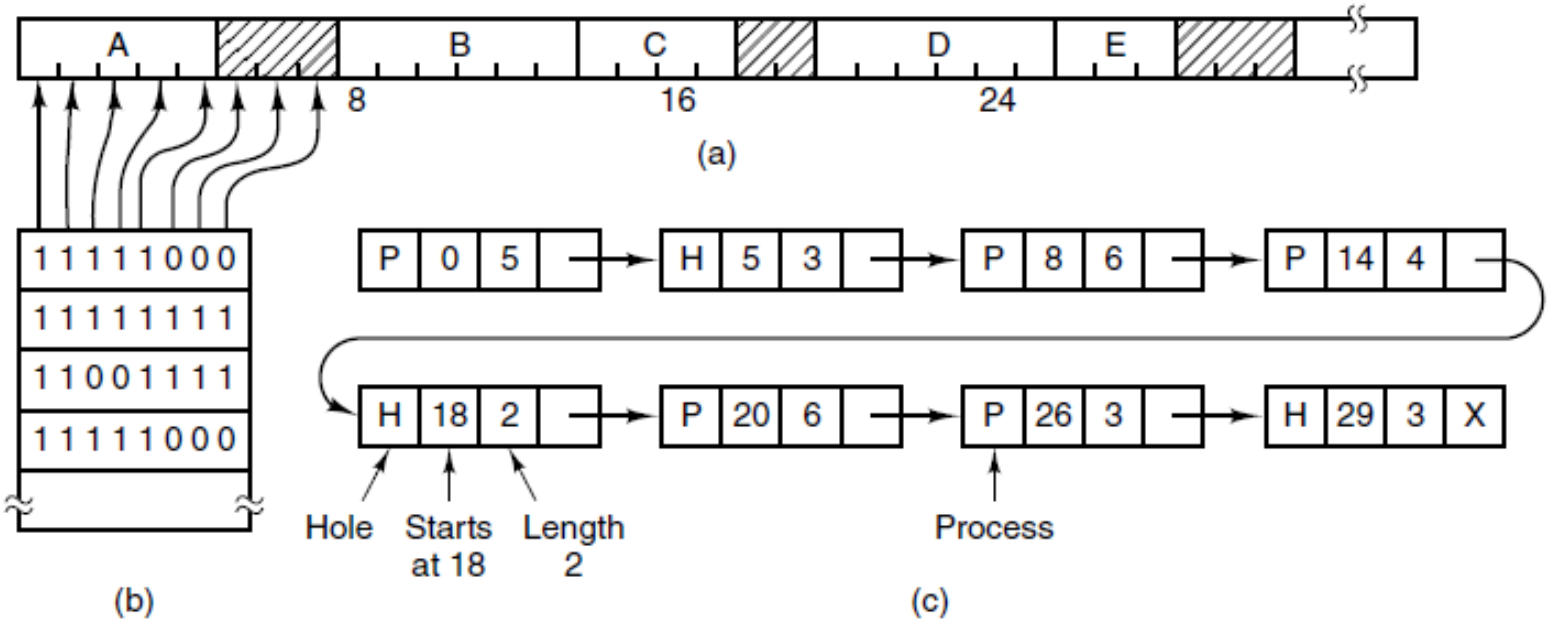
Maintain list for common sizes

Managing Free Memory

- Two ways to track memory usage
  - Bit maps
  - Free lists

- Chapter 10: Linux memory allocators such as buddy and slab.

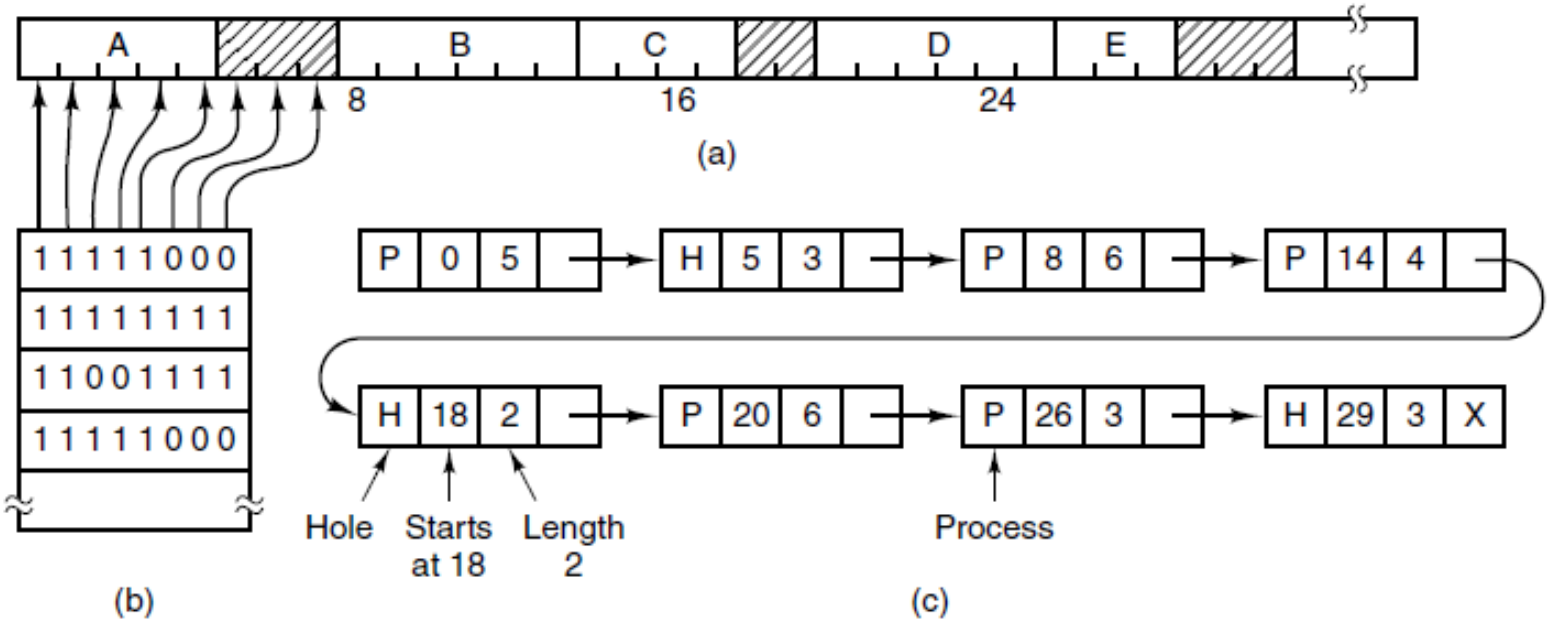
Bitmap



» With bitmap, memory divided into allocation units as small as a few words to a couple kilobytes.

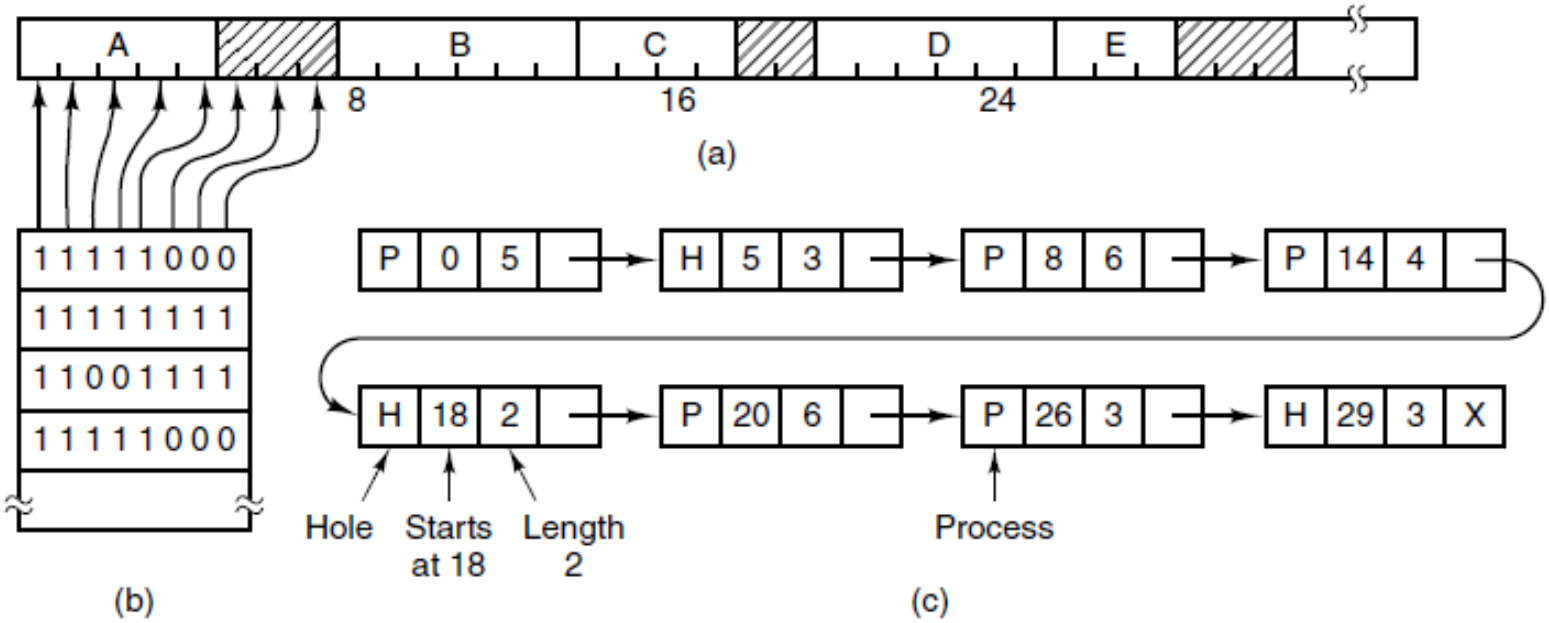
» Each allocation unit corresponds to a bit

Bitmap



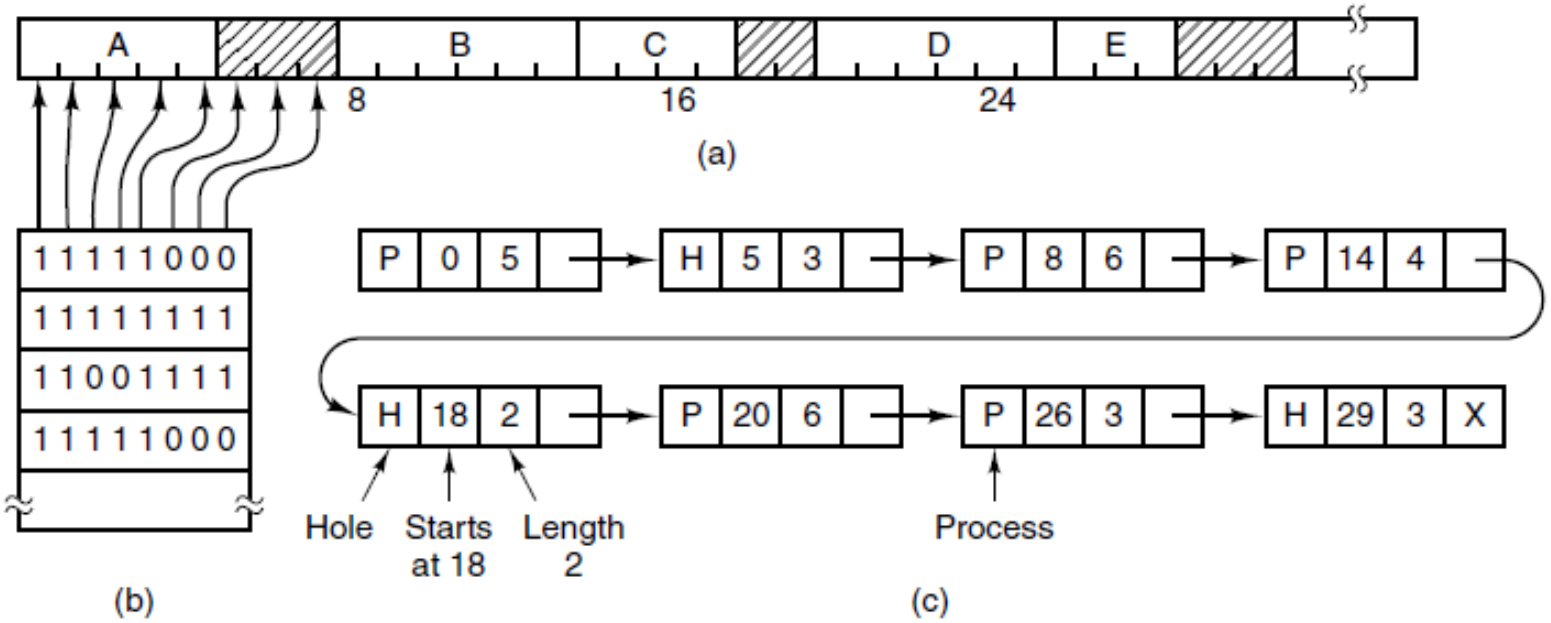
» Problem with bitmap: When bringing in K-units, the memory manager must search the bitmap for a run of k consecutive 0's

Bitmap



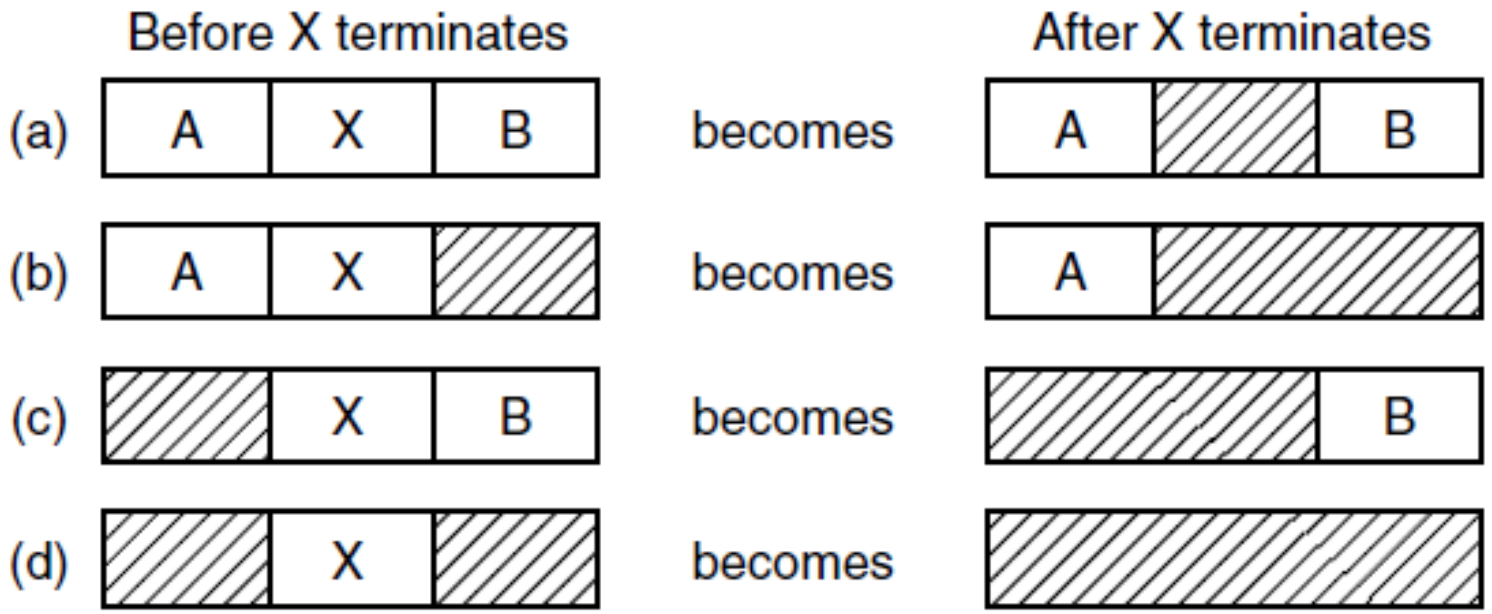
» Problem with bitmap: When bringing in K-units, the memory manager must search the bitmap for a run of k consecutive 0's

Linked List



» Linked list: list of allocated and free memory segments.

Linked List


Before X terminates
After X terminates
(a)
A
X
B
becomes
A
B
(b)
A
X
becomes
A
(c)
X
B
becomes
B
(d)
X
becomes

Memory Allocation

- Can be optimized by keeping separate lists for processes and holes.
- If separate lists kept then the hole list should be sorted by size

What if the program size exceeds physical RAM?

Overlays

» Memory wasn't always cheap. Could be a large part of total system price.

» How do you write large programs for small memories?

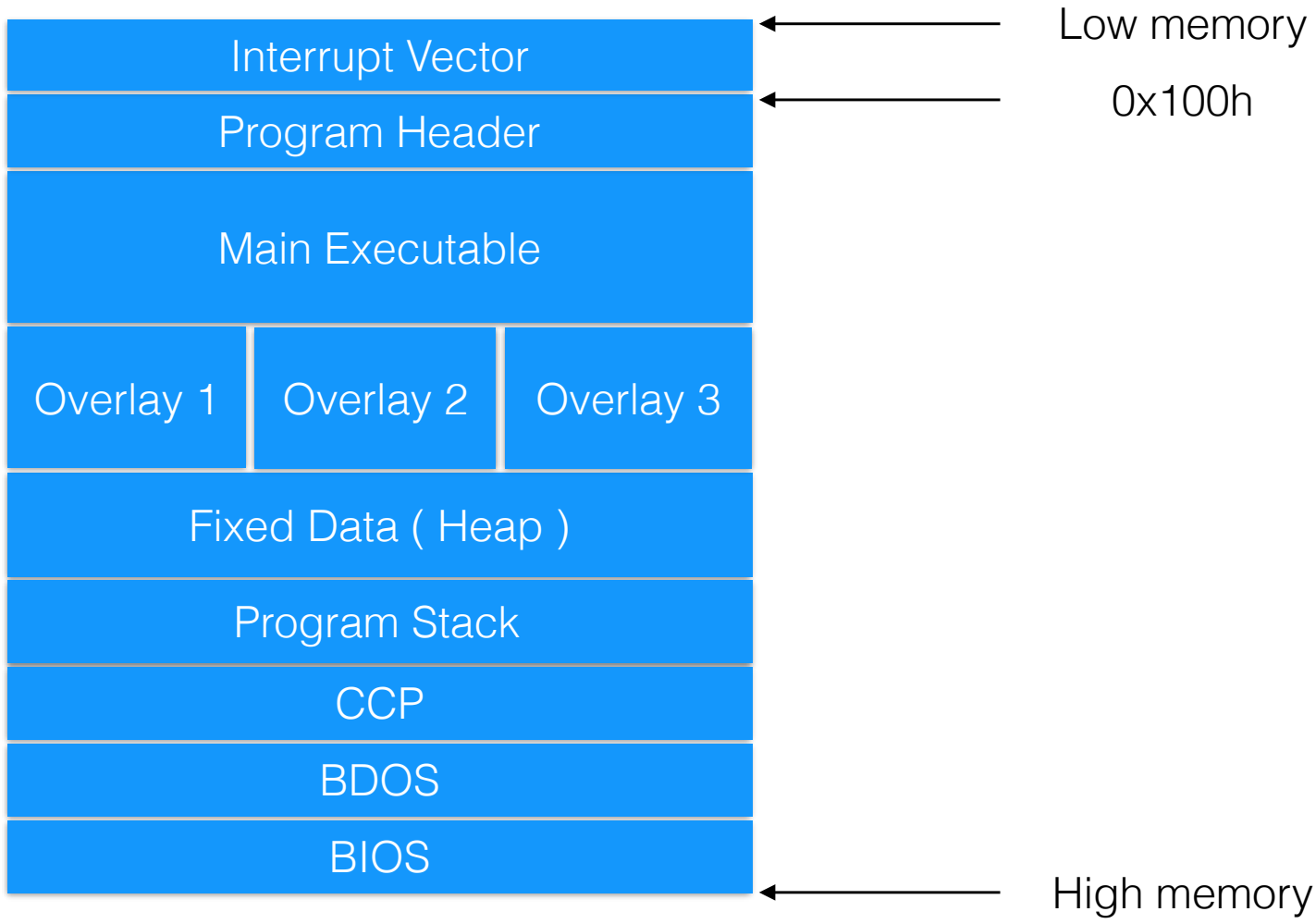
» init, main, finalize - common programming pattern

  – Don't have to be in memory at the same time.

Overlays

- The programmer specifies the overlays
- An overlay can not make calls into another overlay.
- Actual loading of overlays into memory would be done by the runtime library

Program Overlay Layout



We need help, but why?

» Multiprocessing causes external fragmentation

» Compaction wastes resources

» Solution – break RAM into fixed parts

» Do not need to be contiguous

» Map from logical to physical spaces

» Need hardware help