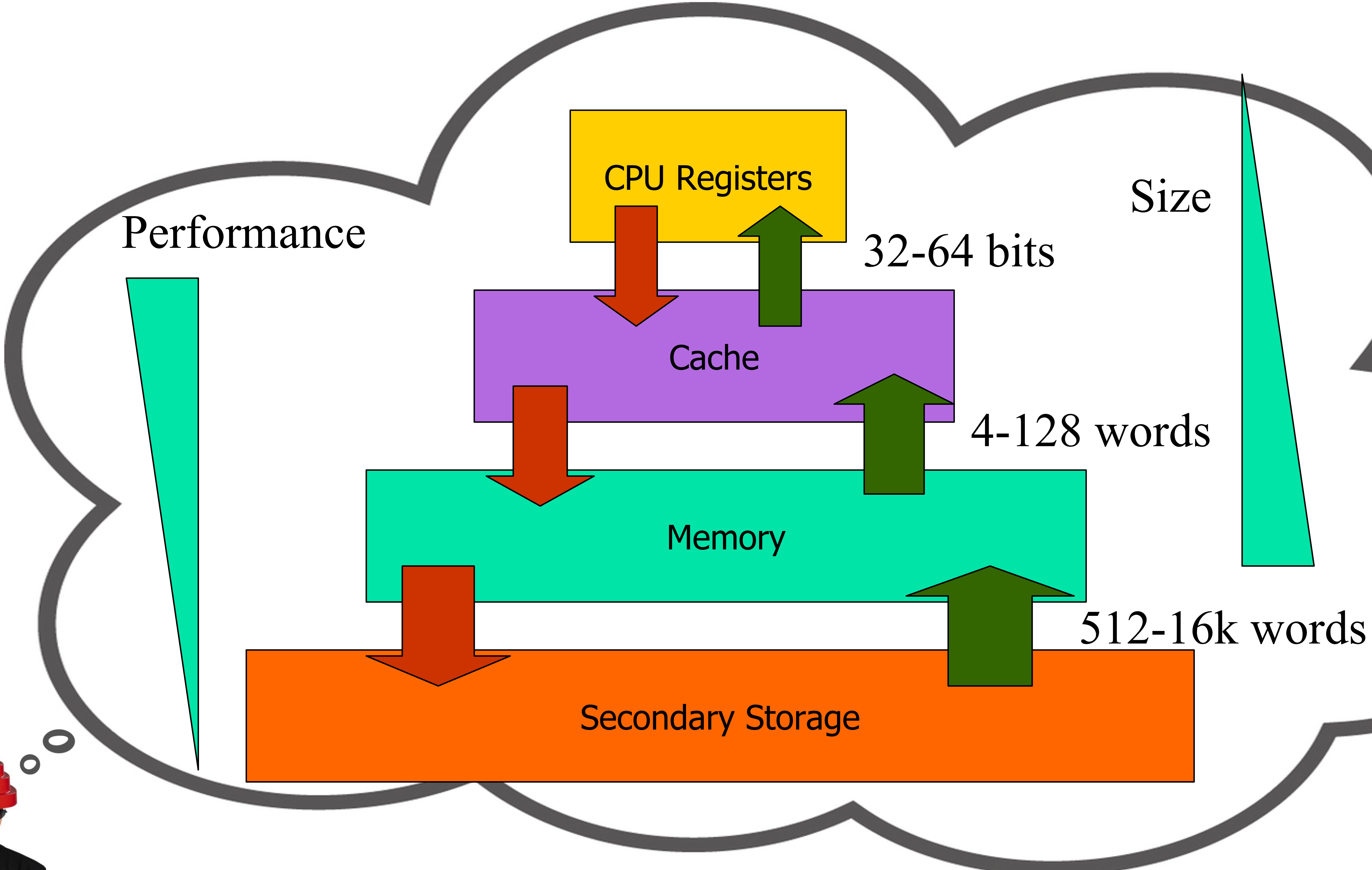
CS 423
Operating System Design: Historical Memory Management

Tianyin Xu (MIC)

* Thanks for Prof. Adam Bates for the slides.

Storage Hierarchy

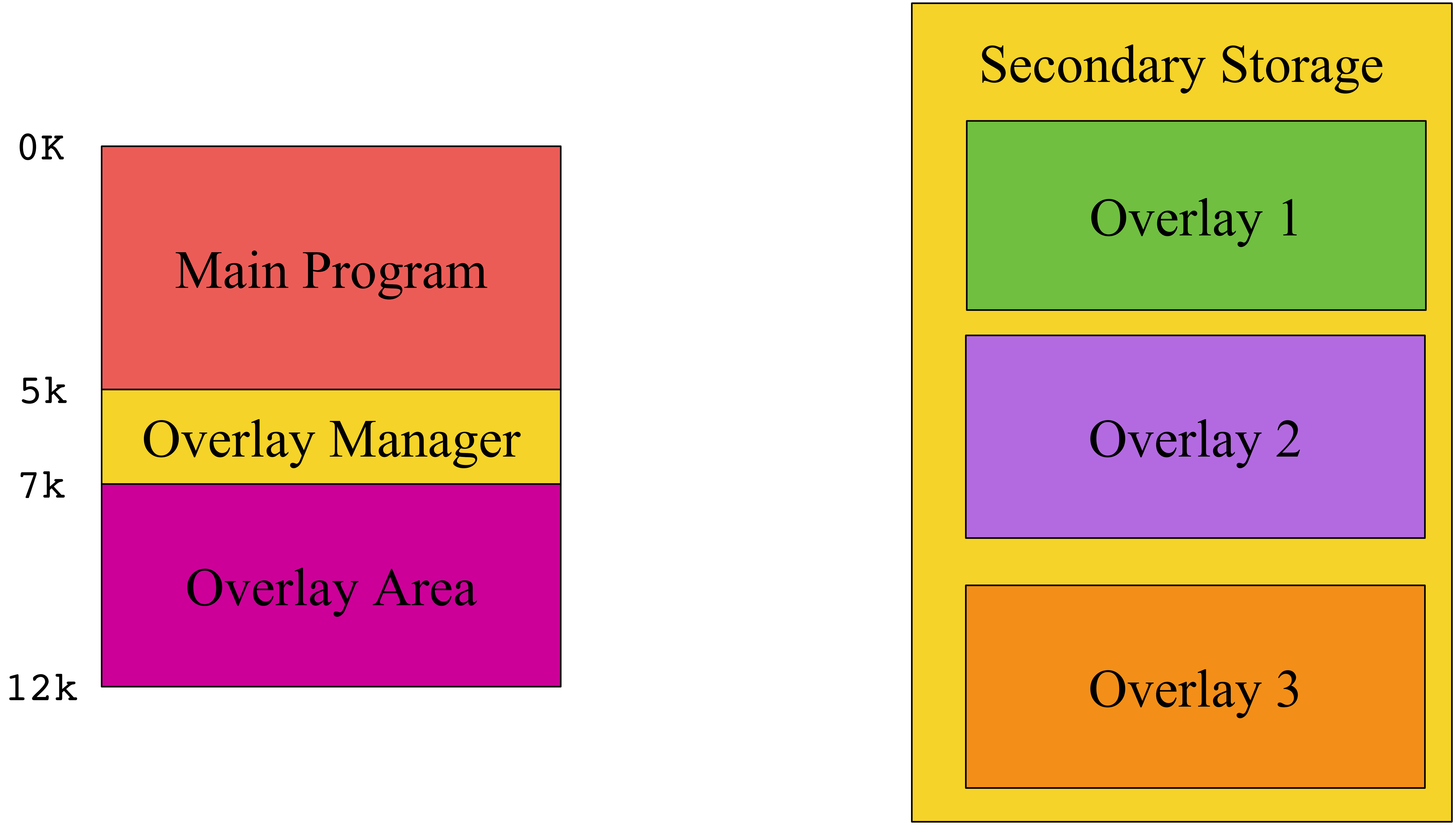


Problem Statement

We have limited amounts of fast resources,
and large amounts of slower resources…

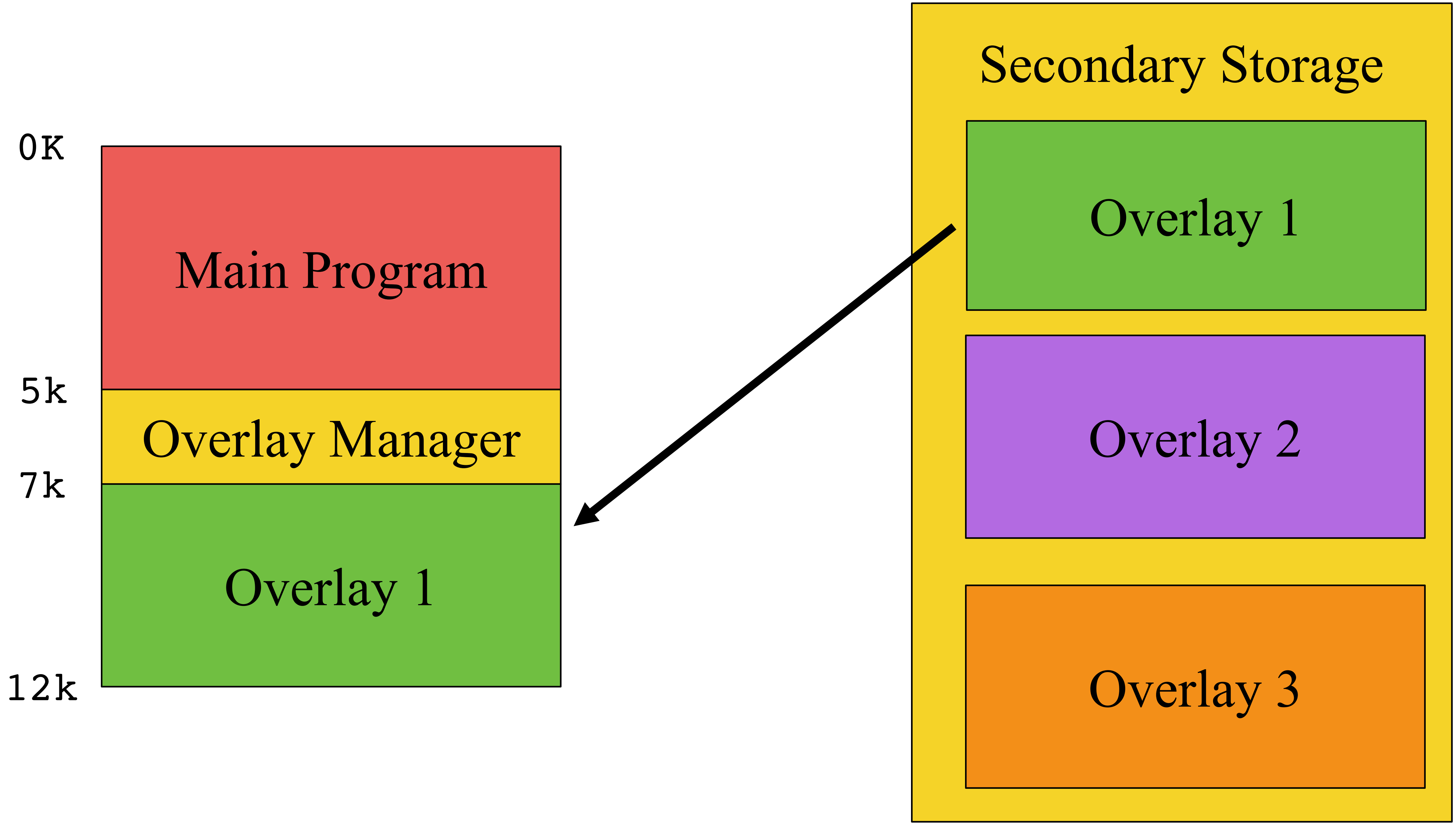
<u>How to create the illusion of an abundant fast resource?</u>

History: Mem Overlays



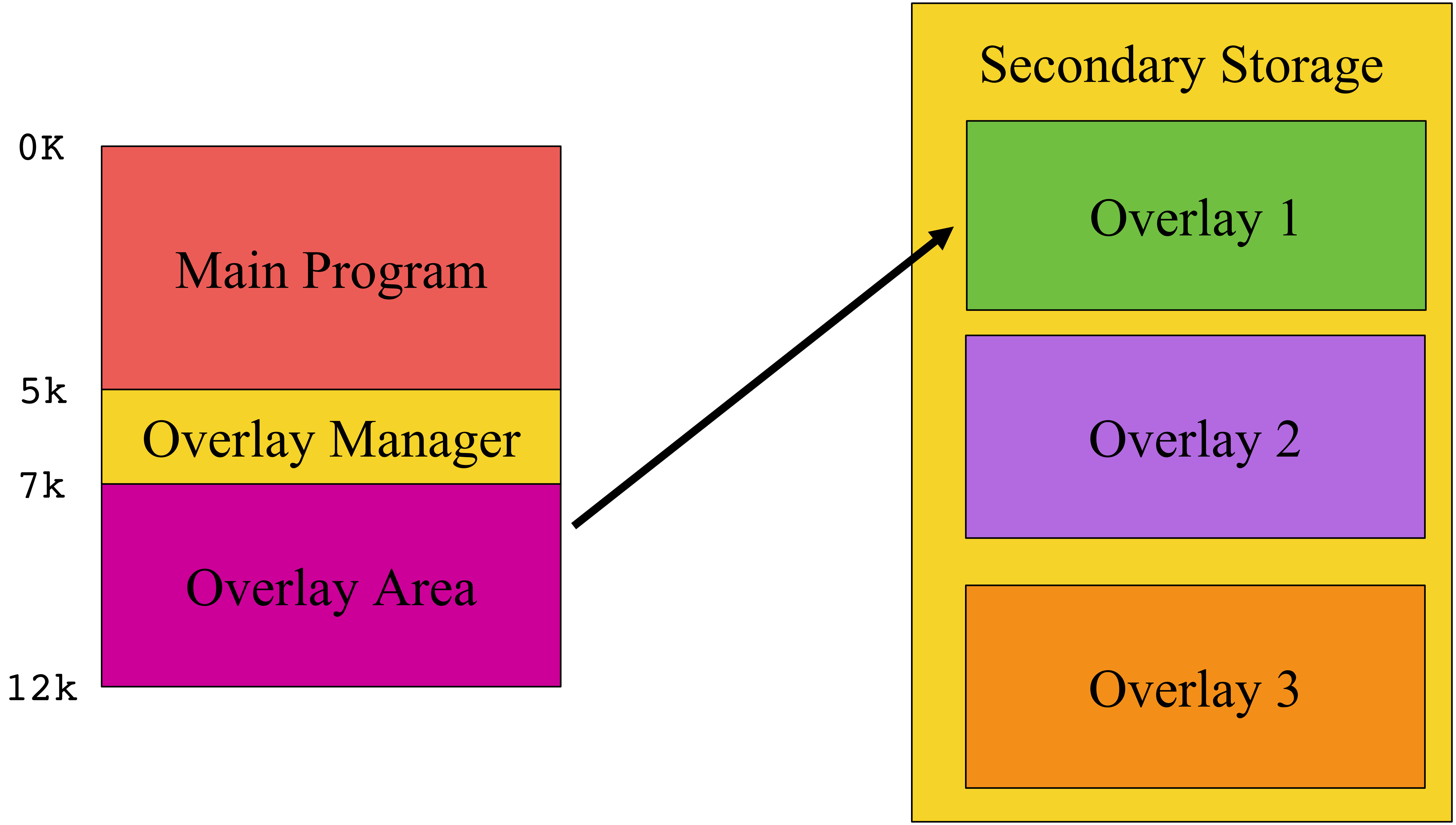
Used when process memory requirement exceeded the physical memory space

History: Mem Overlays



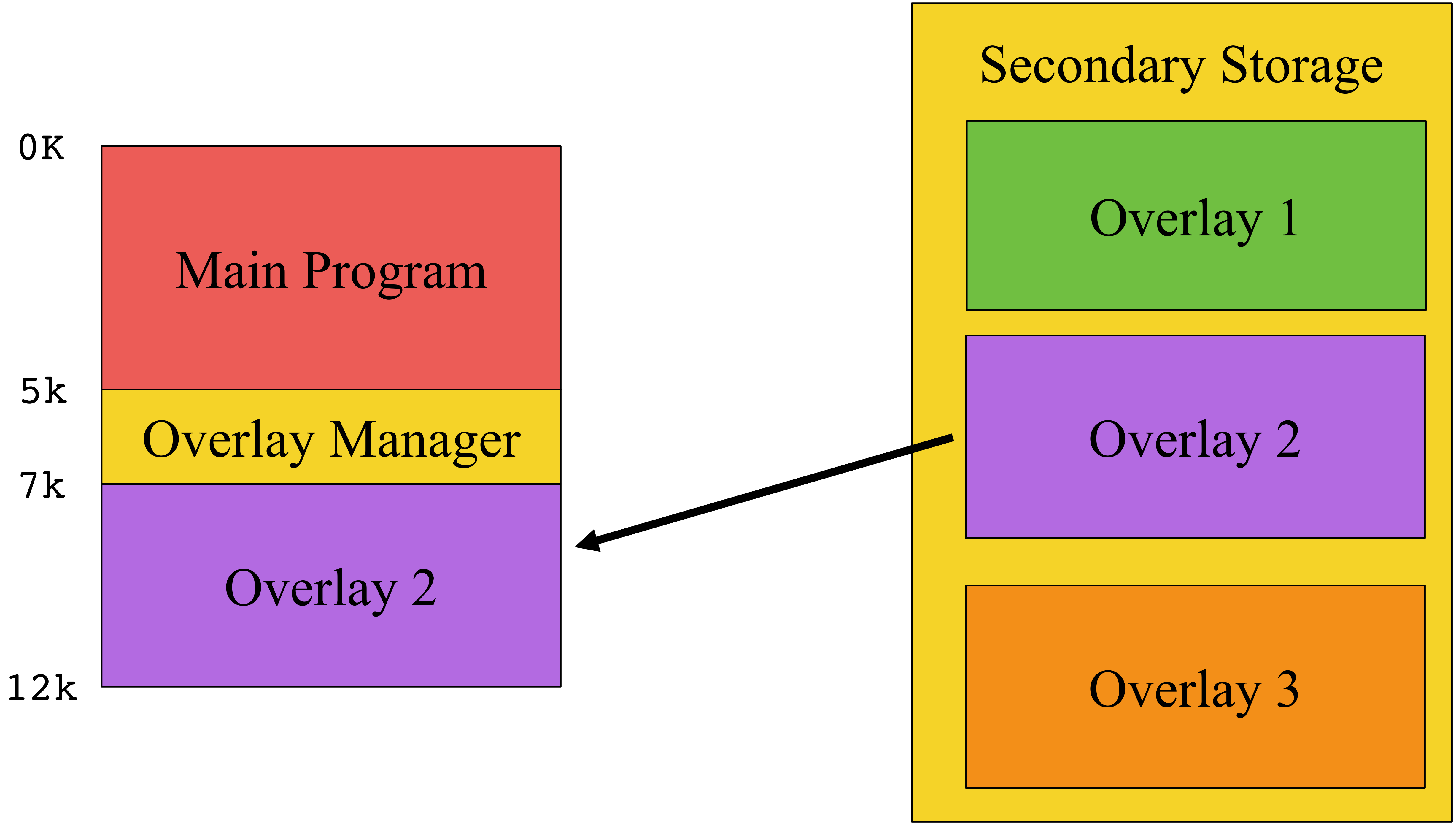
Used when process memory requirement exceeded the physical memory space

History: Mem Overlays



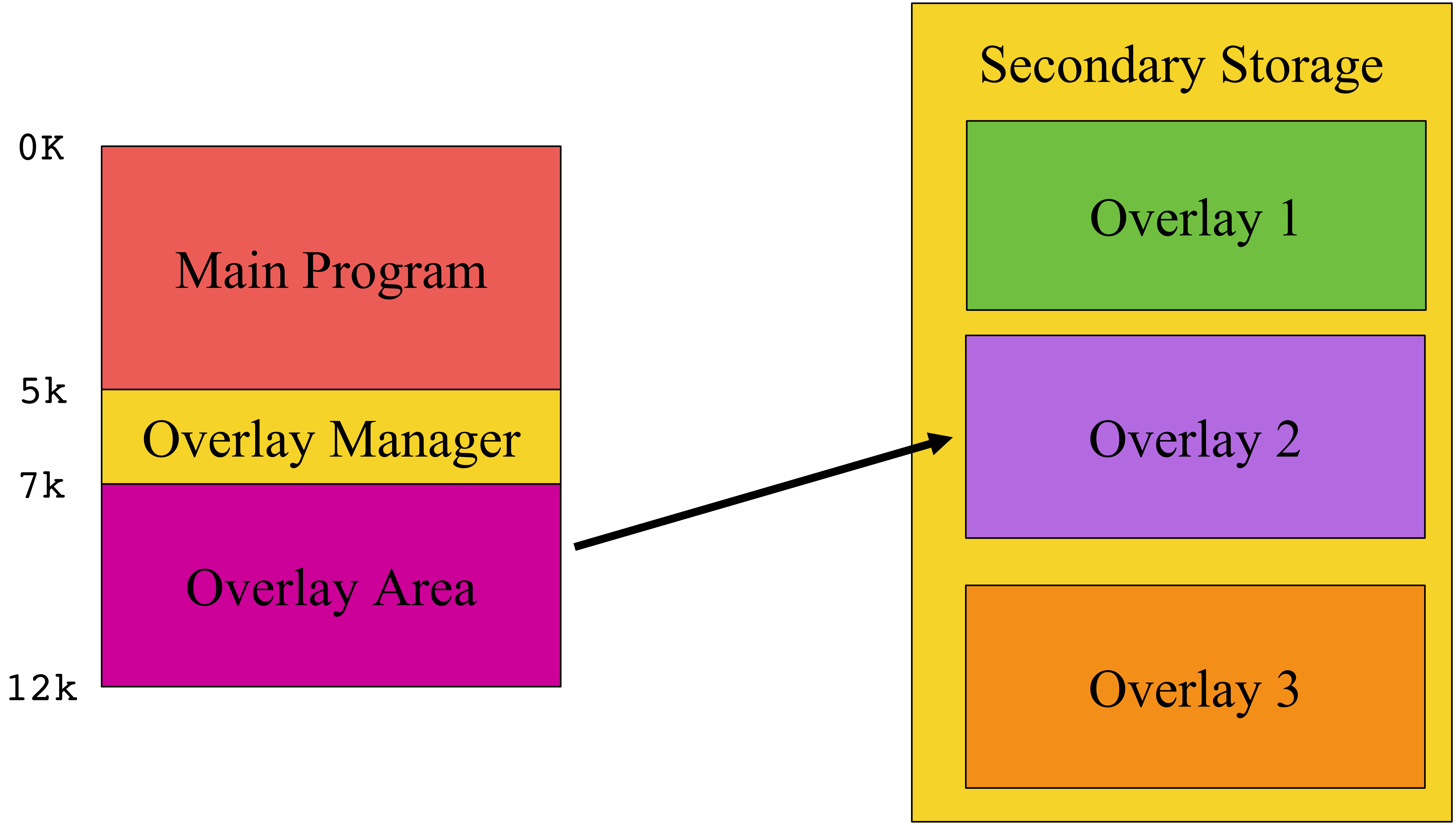
Used when process memory requirement exceeded the physical memory space

History: Mem Overlays



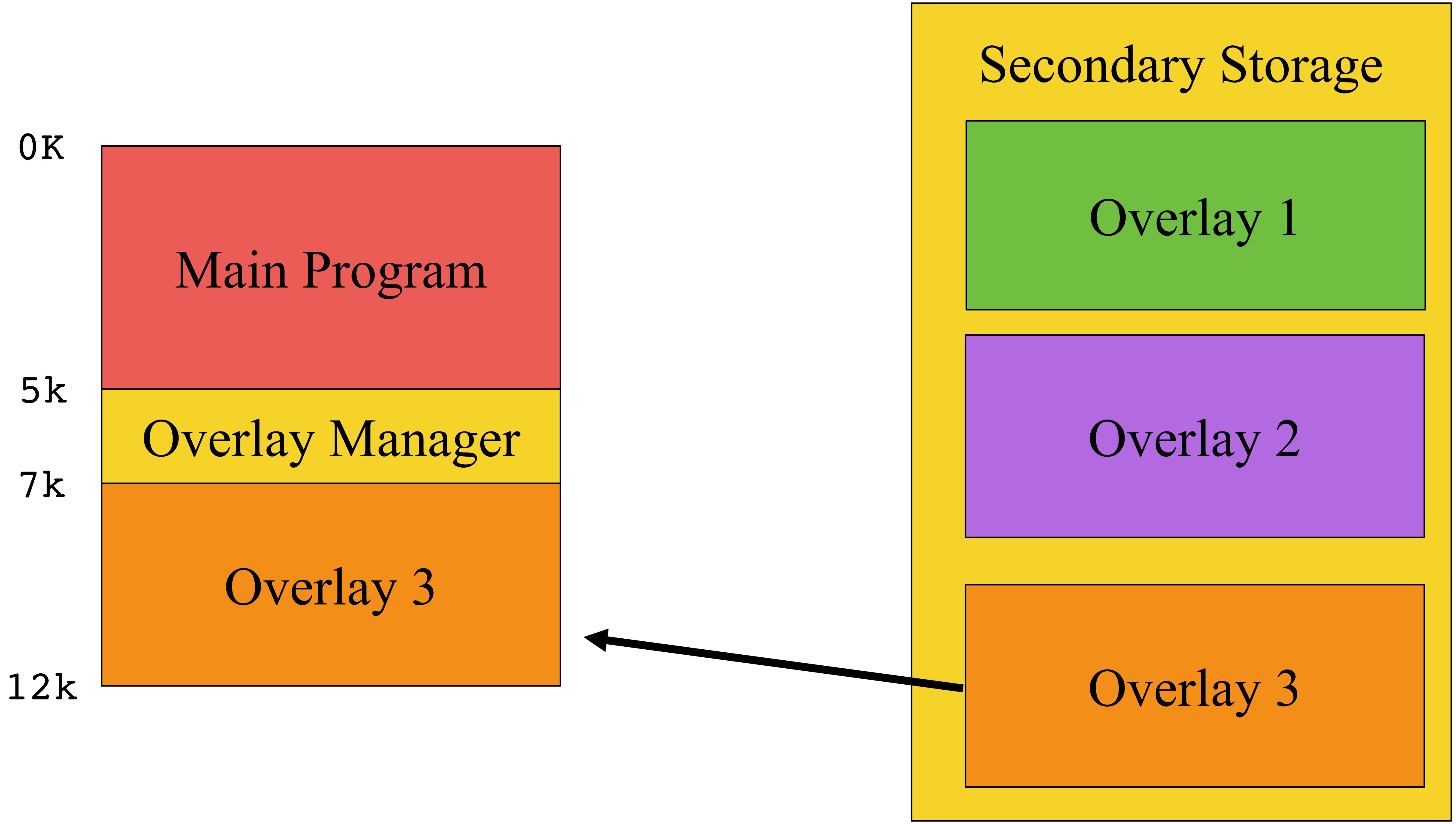
Used when process memory requirement exceeded the physical memory space

History: Mem Overlays



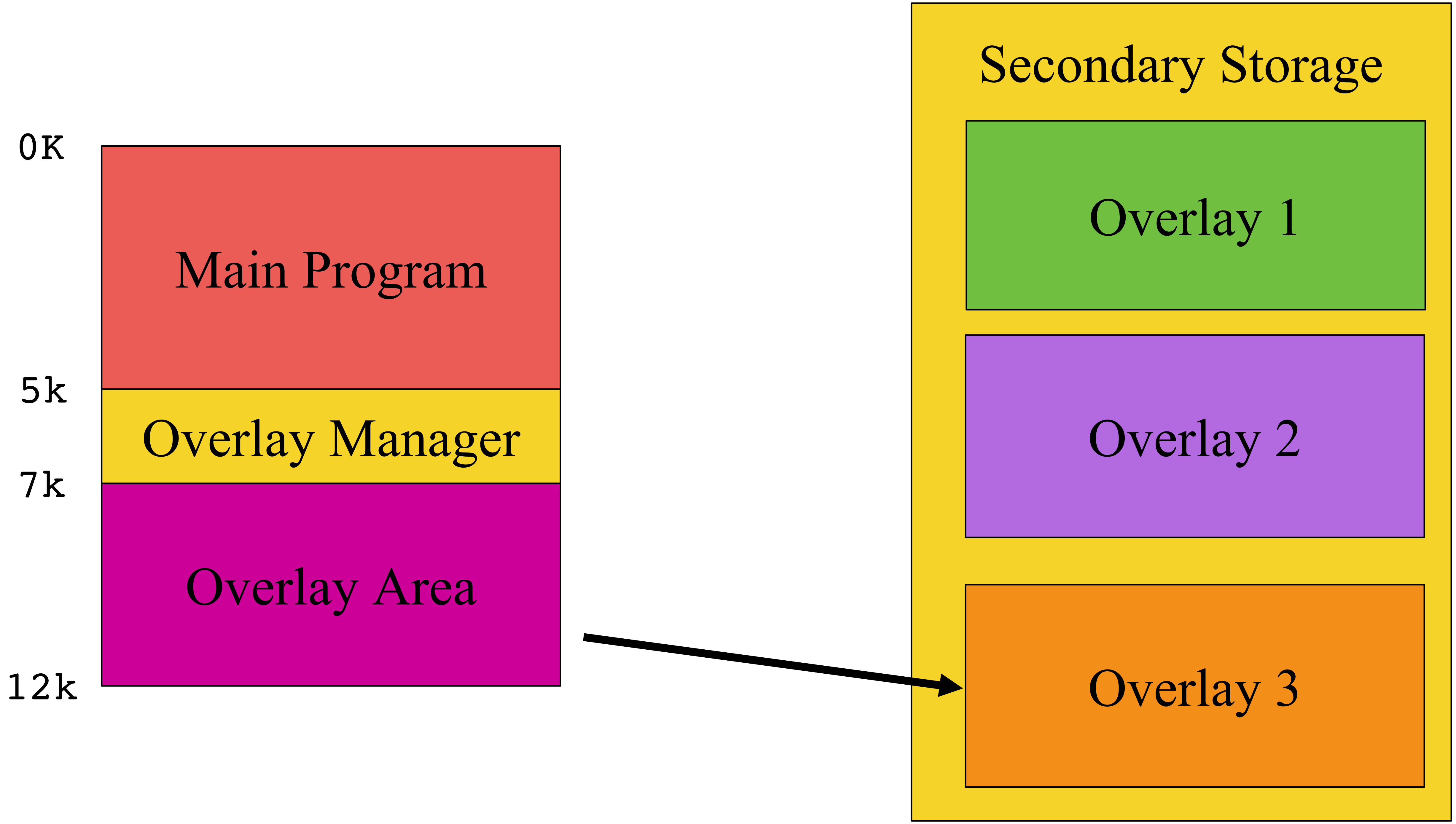
Used when process memory requirement exceeded the physical memory space

History: Mem Overlays



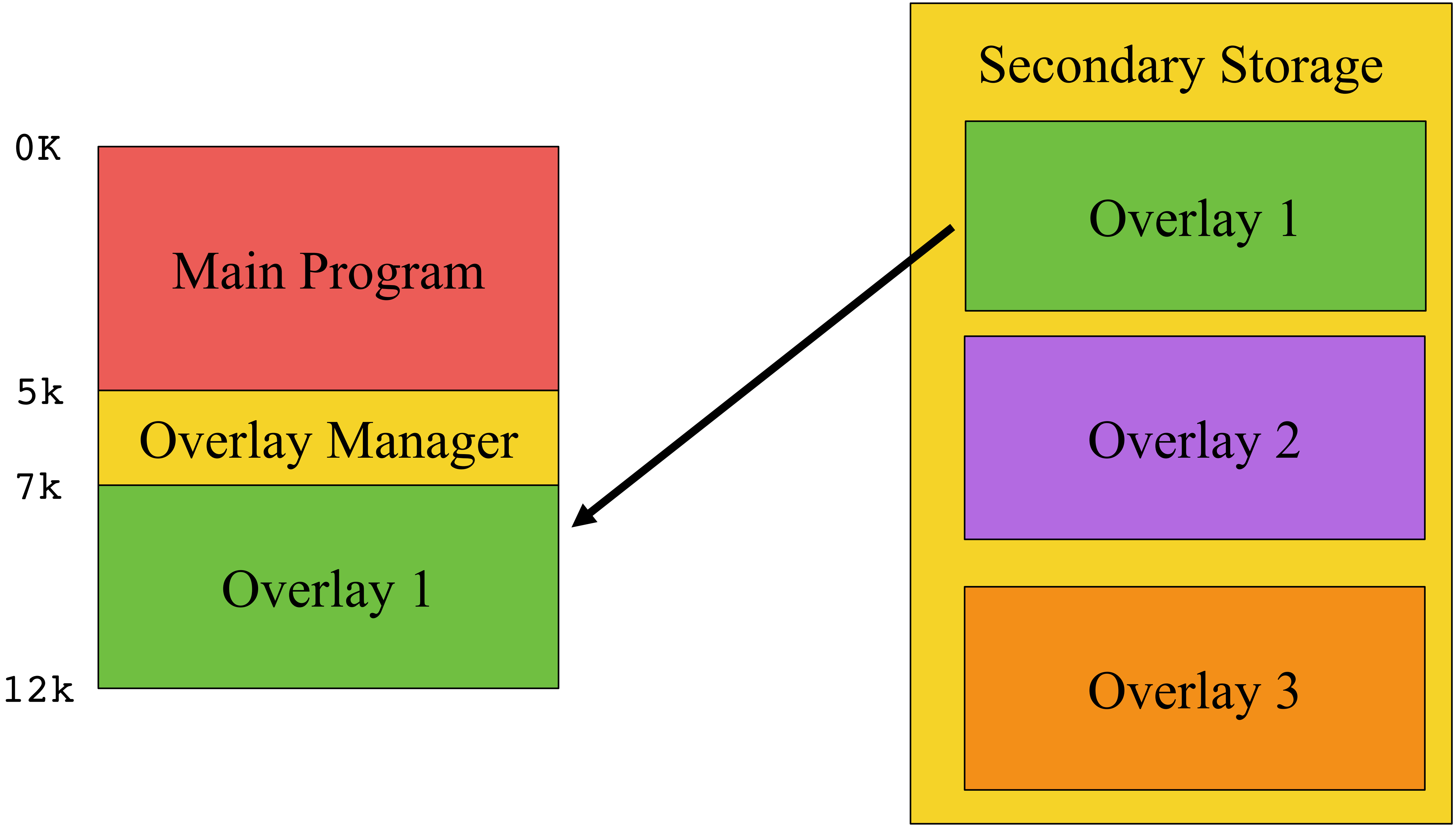
Used when process memory requirement exceeded the physical memory space

History: Mem Overlays



Used when process memory requirement exceeded the physical memory space

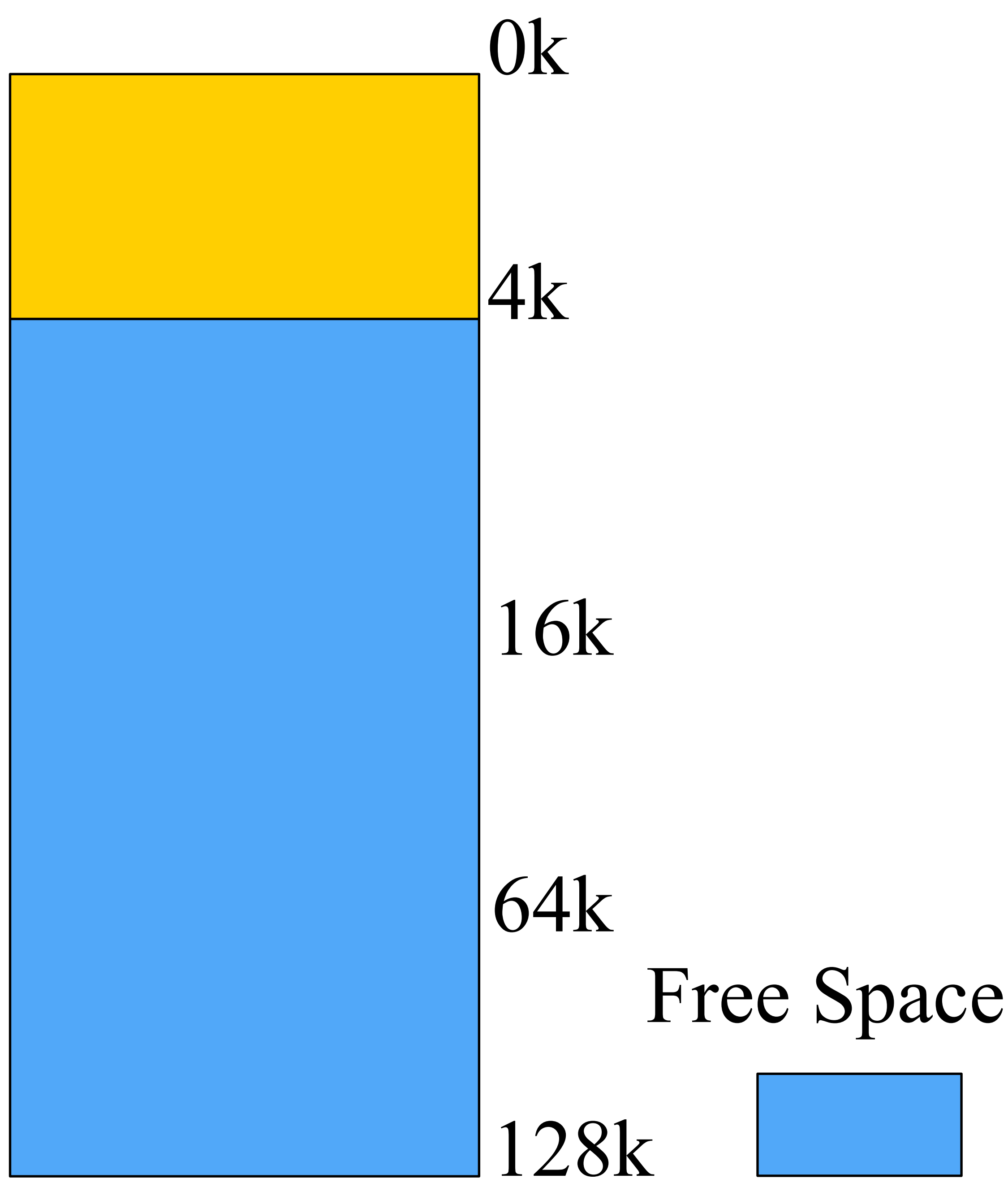
History: Mem Overlays



Used when process memory requirement exceeded the physical memory space

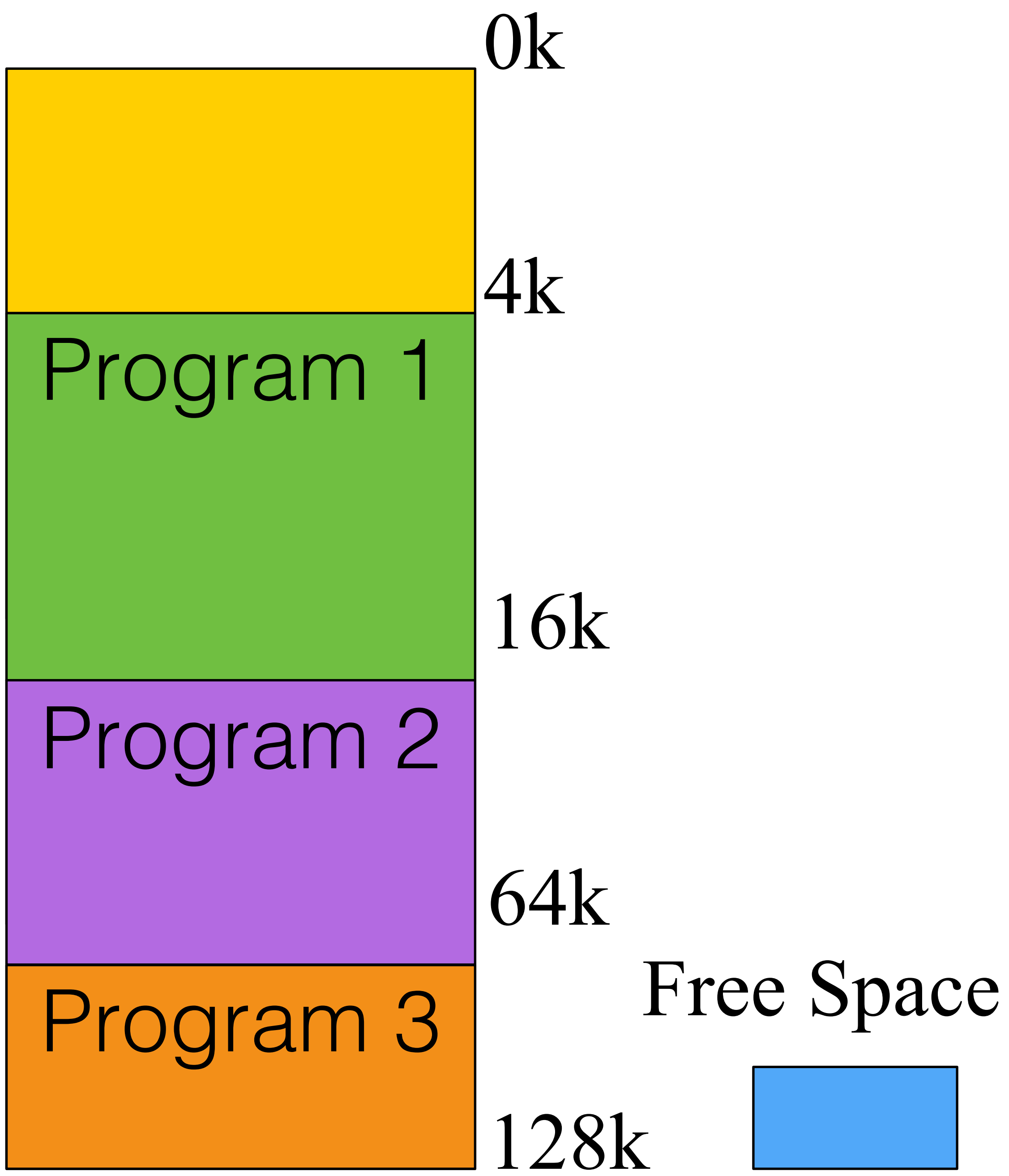
History: Fixed Partition Allocation

- Approach: Multiprogramming with fixed memory partitions
- Divides memory into ***n*** fixed partitions (possibly unequal)
- Problem?



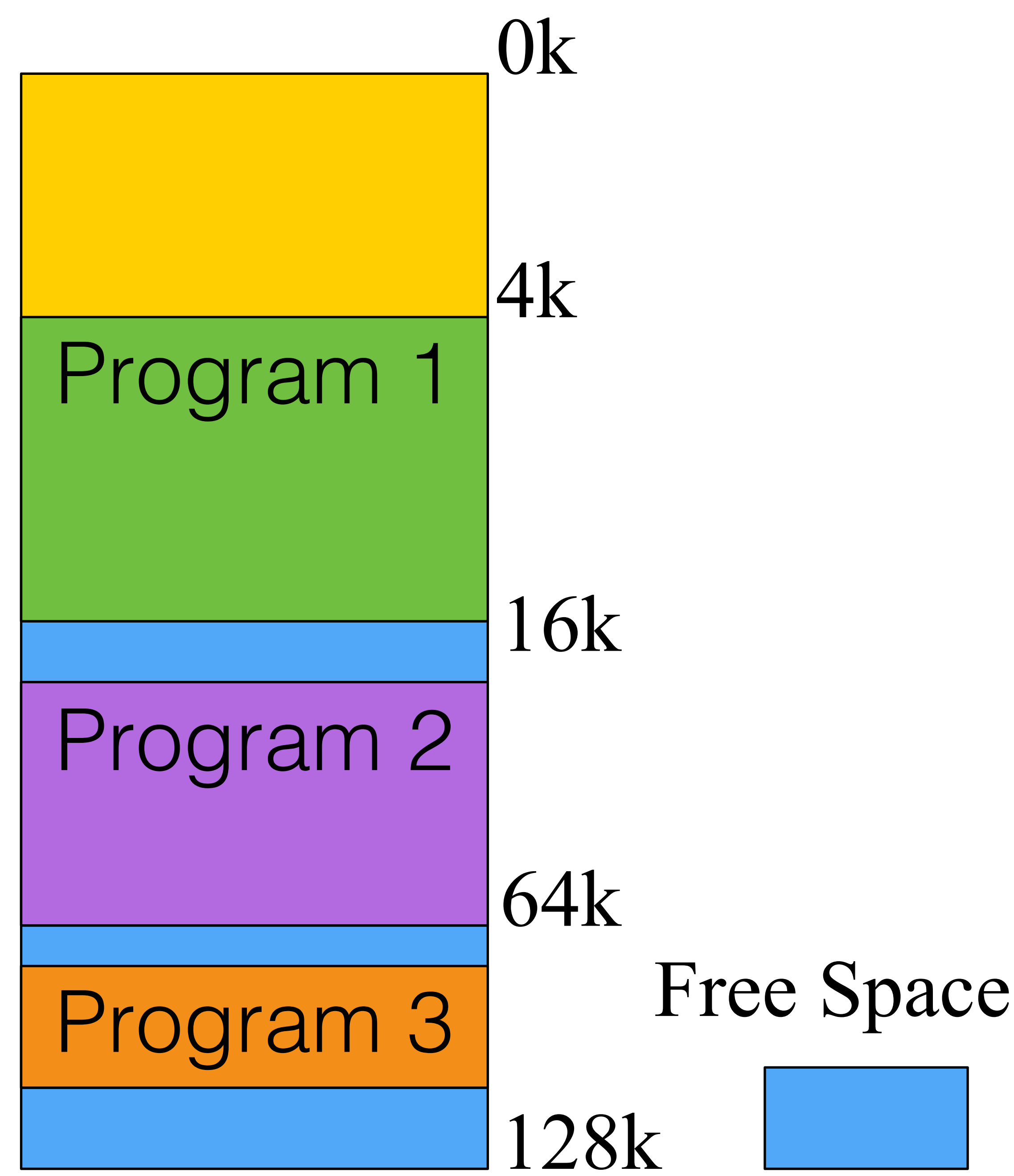
History: Fixed Partition Allocation

- Approach: Multiprogramming with fixed memory partitions
- Divides memory into ***n*** fixed partitions (possible unequal)
- Problem?



History: Fixed Partition Allocation

- Approach: Multiprogramming with fixed memory partitions
- Divides memory into ***n*** fixed partitions (possible unequal)
- Problem?
  - Internal Fragmentation



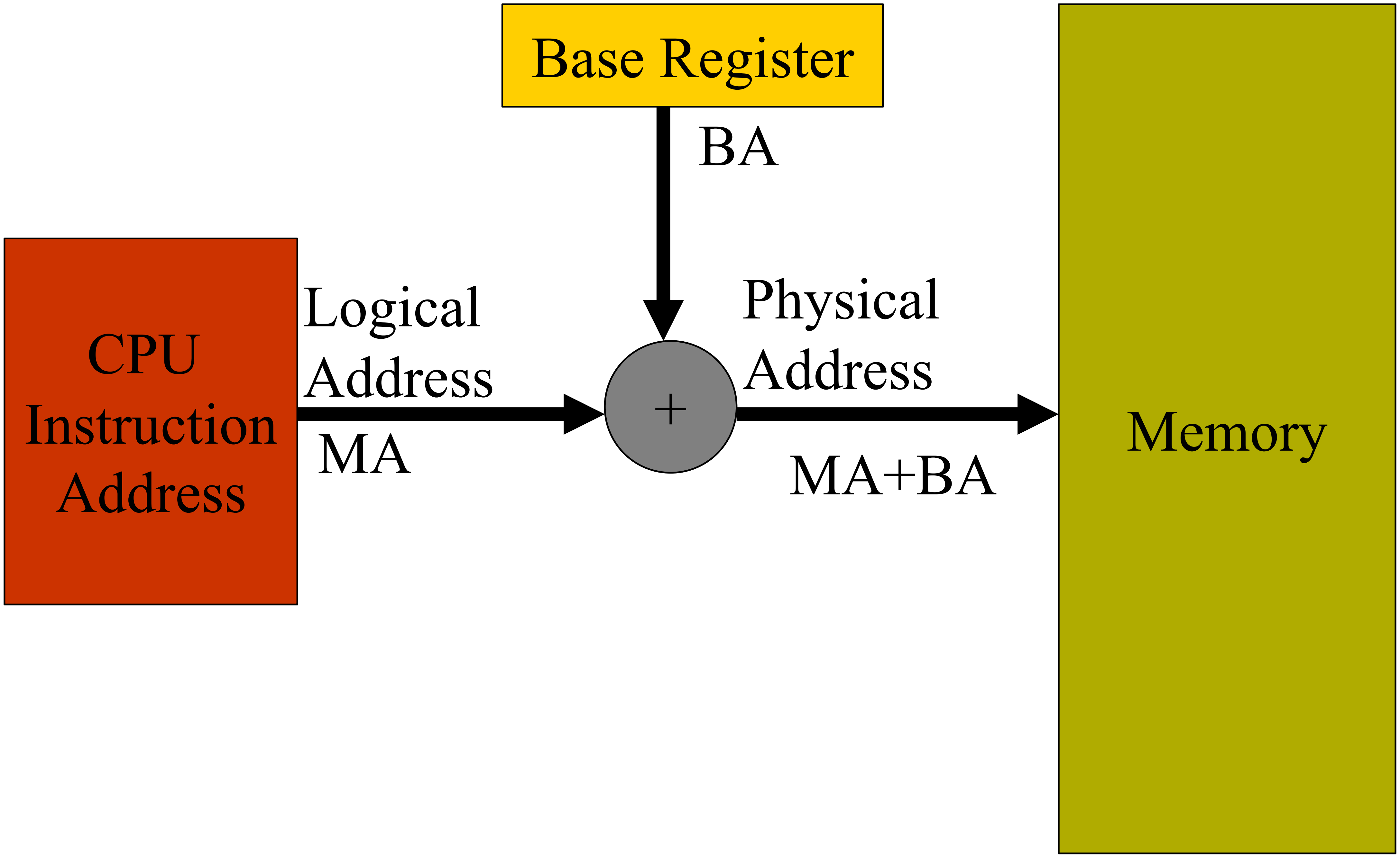
History: Fixed Partition Allocation

- Separate input queue for each partition
  - Sorting incoming jobs into separate queues
  - Inefficient utilization of memory
    - when the queue for a large partition is empty but the queue for a small partition is full. Small jobs have to wait to get into memory even though plenty of memory is free.
- One single input queue for all partitions.
  - Allocate a partition where the job fits in.

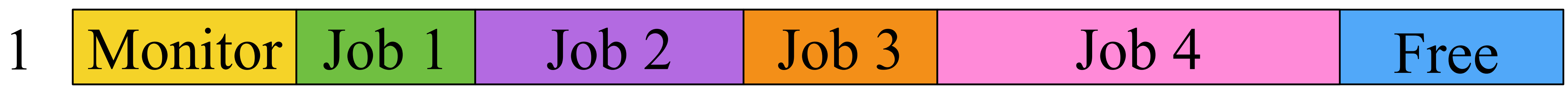
History: Relocation

- Correct starting address when a program should start in the memory
- Different jobs will run at different addresses
  - When a program is linked, the linker must know at what address the program will begin in memory.
- Enter "Logical addresses"
  - Logical address space , range (0 to max)
  - Physical addresses, Physical address space range (R+0 to R+max) for base value R.
  - User program never sees the real physical addresses
- Relocation register
  - Mapping requires hardware with the base register

History: Relocation Register



History: Variable Partition Allocation



Memory wasted by <u>External</u> Fragmentation

History: Storage Placement Strategy

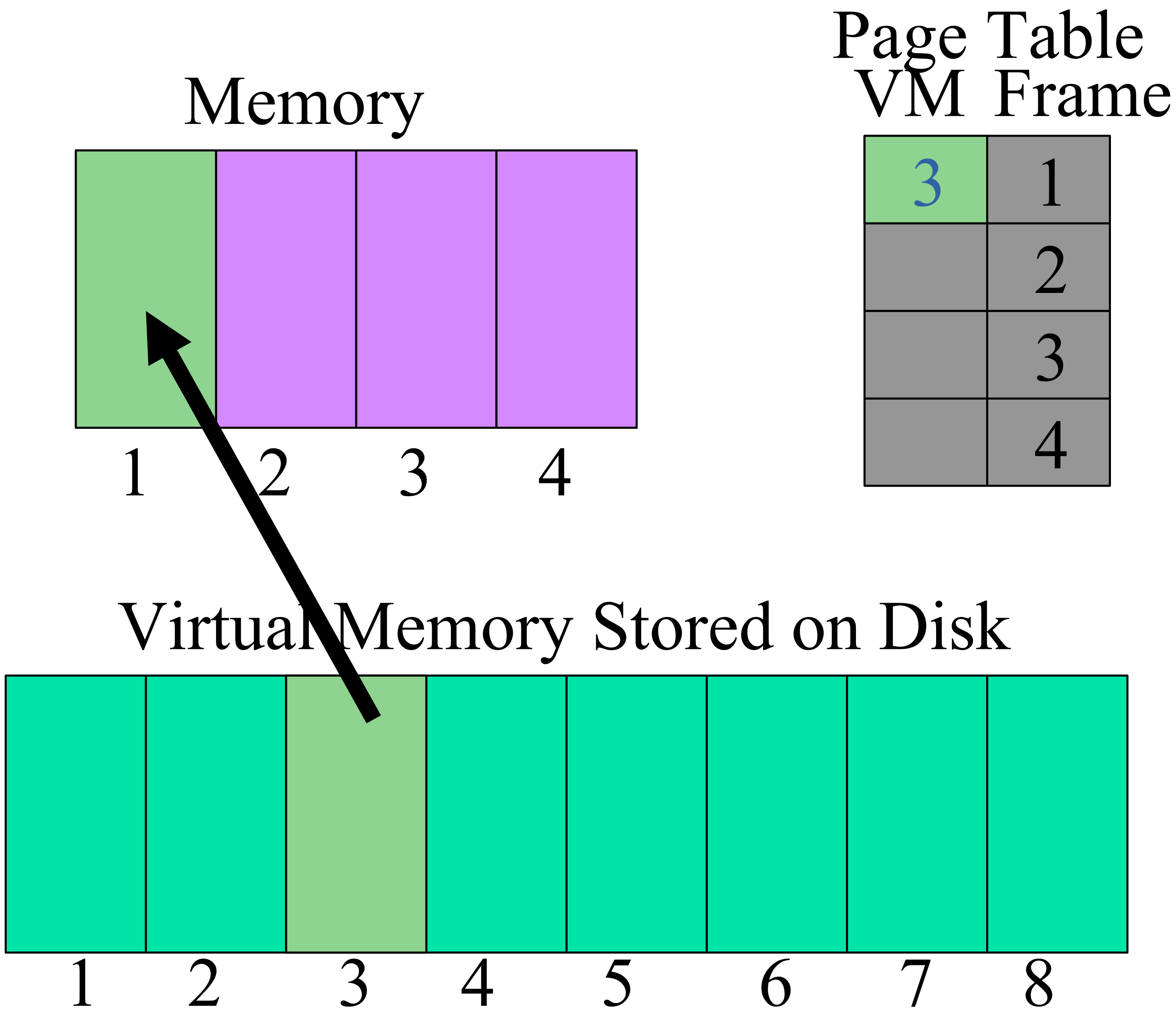
- **Best Fit?**
- **First Fit?**
- **Next Fit?**
- **Worst Fit?**

Virtual Memory

- Provide user with virtual memory that is as big as user needs
- Store virtual memory on disk
- Cache parts of virtual memory being used in real memory
- Load and store cached virtual memory without user program intervention

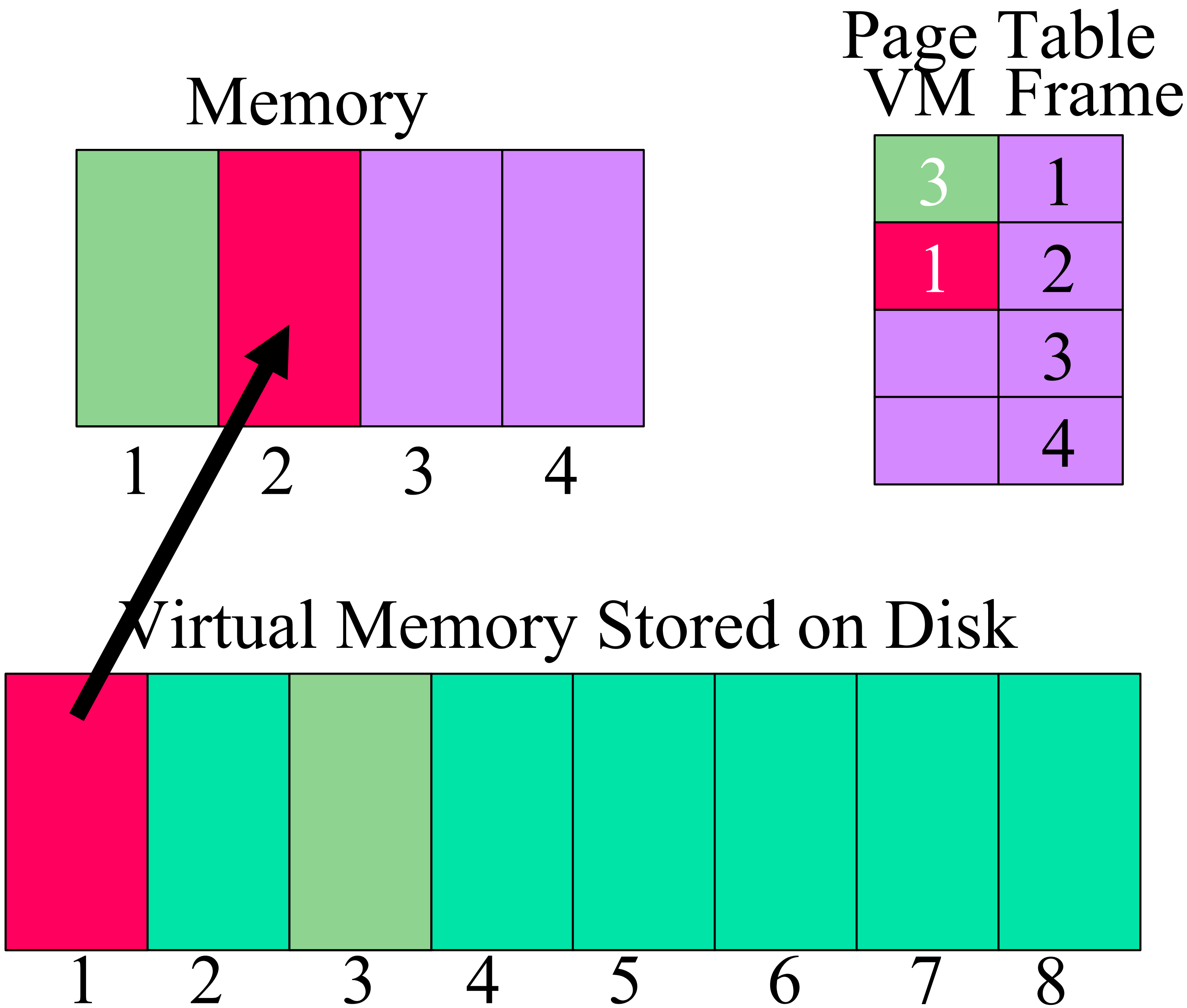
Paging

Request Page 3…



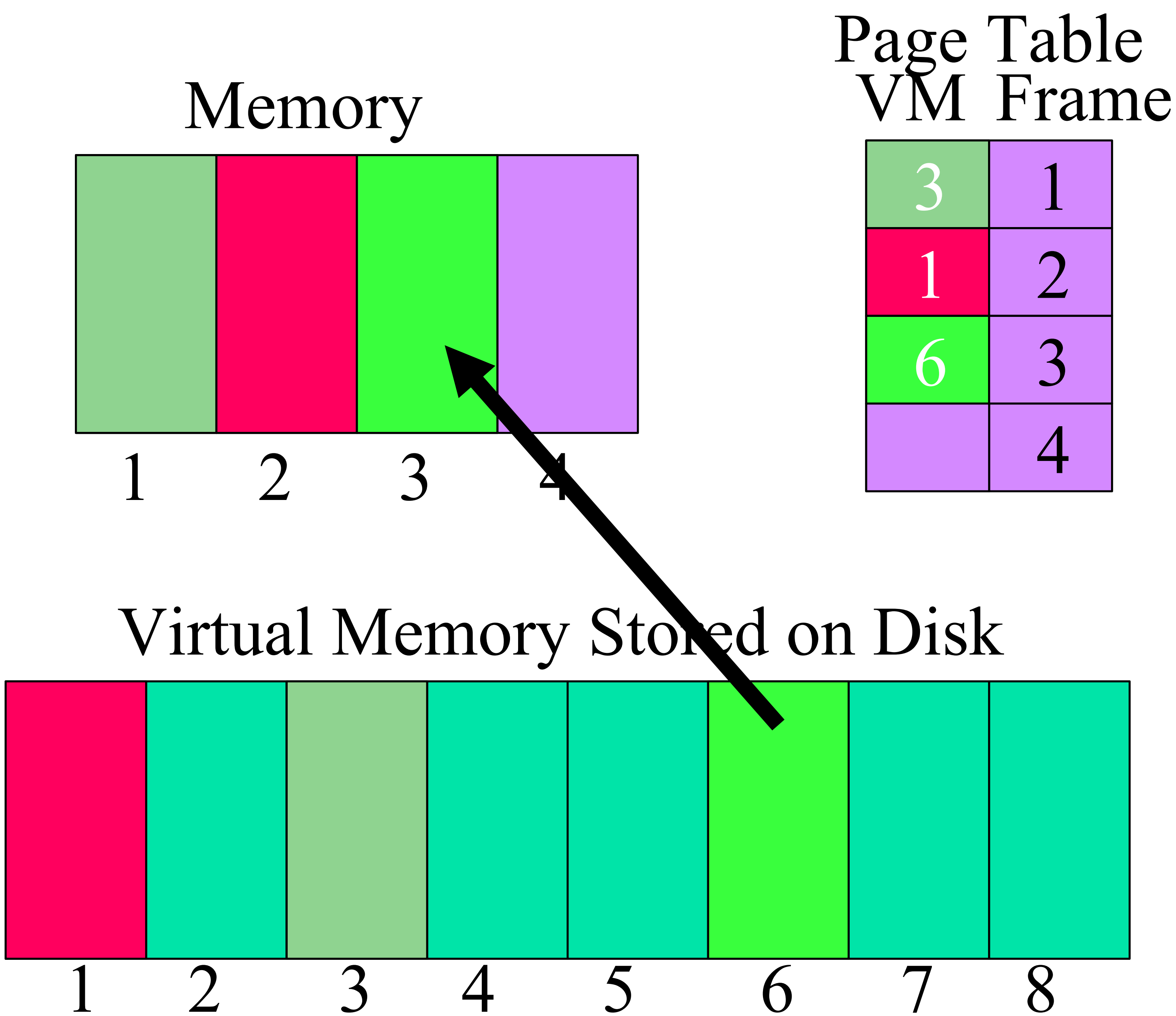
Paging

Request Page 1…



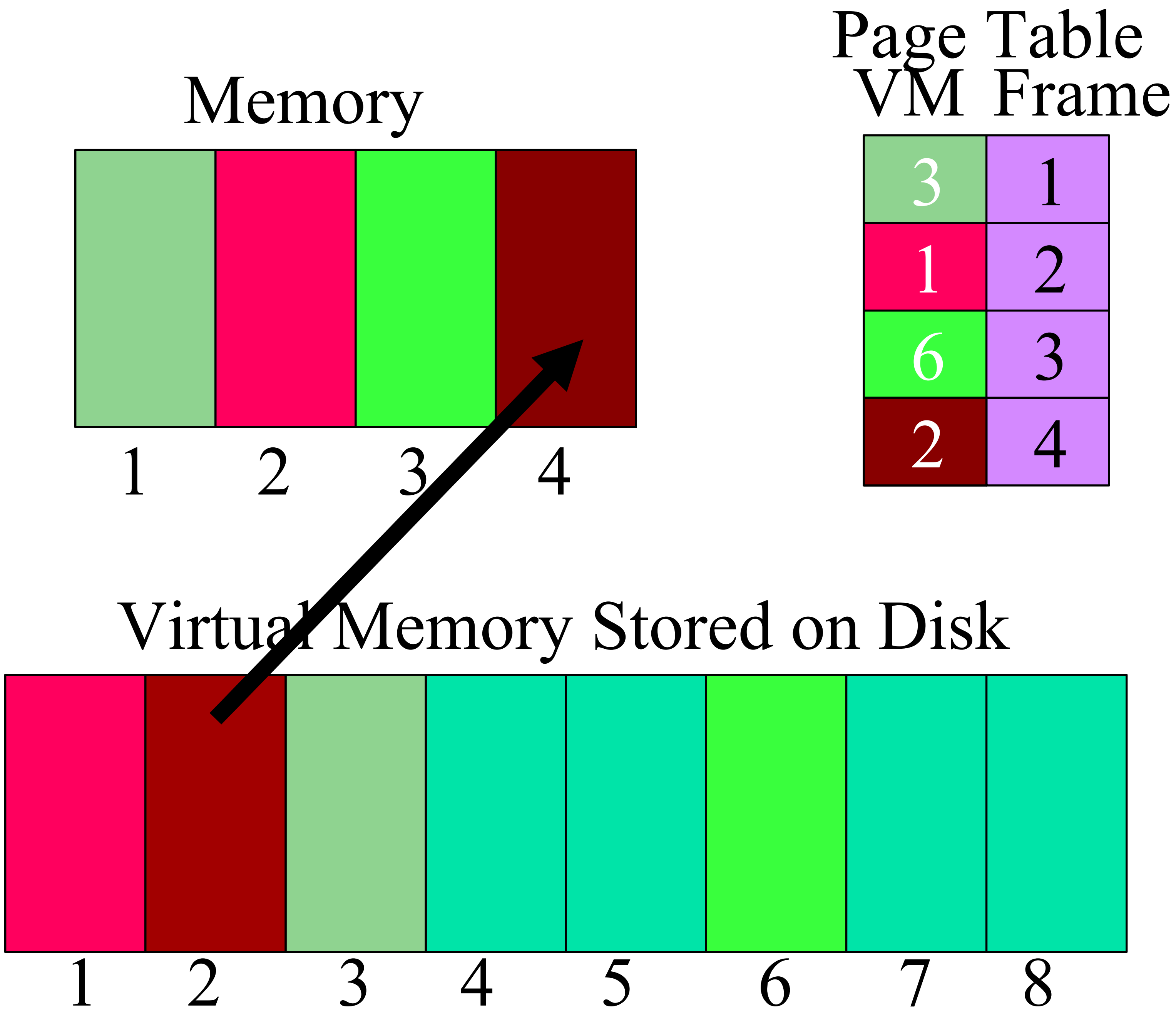
Paging

Request Page 6…



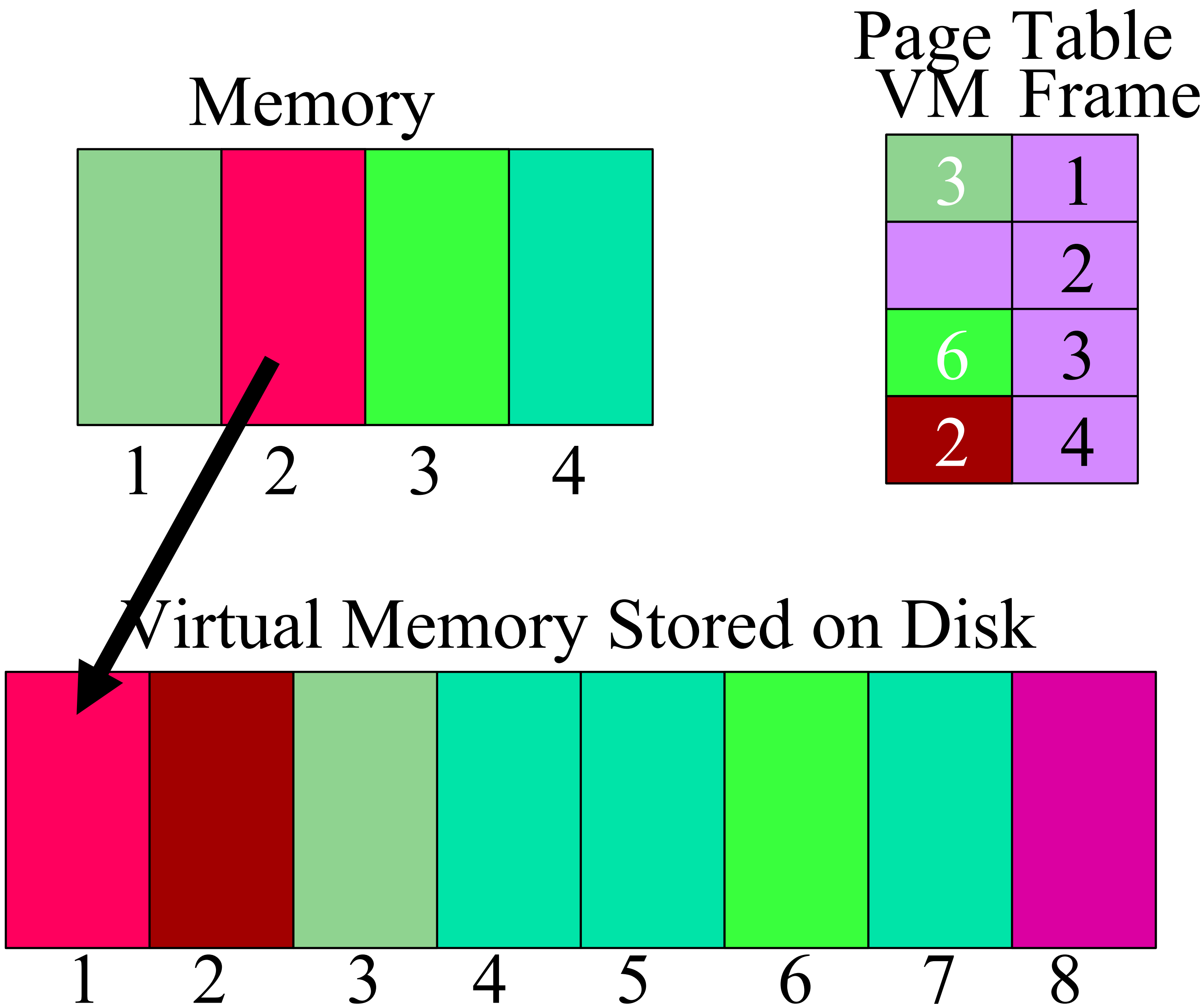
Paging

Request Page 2…



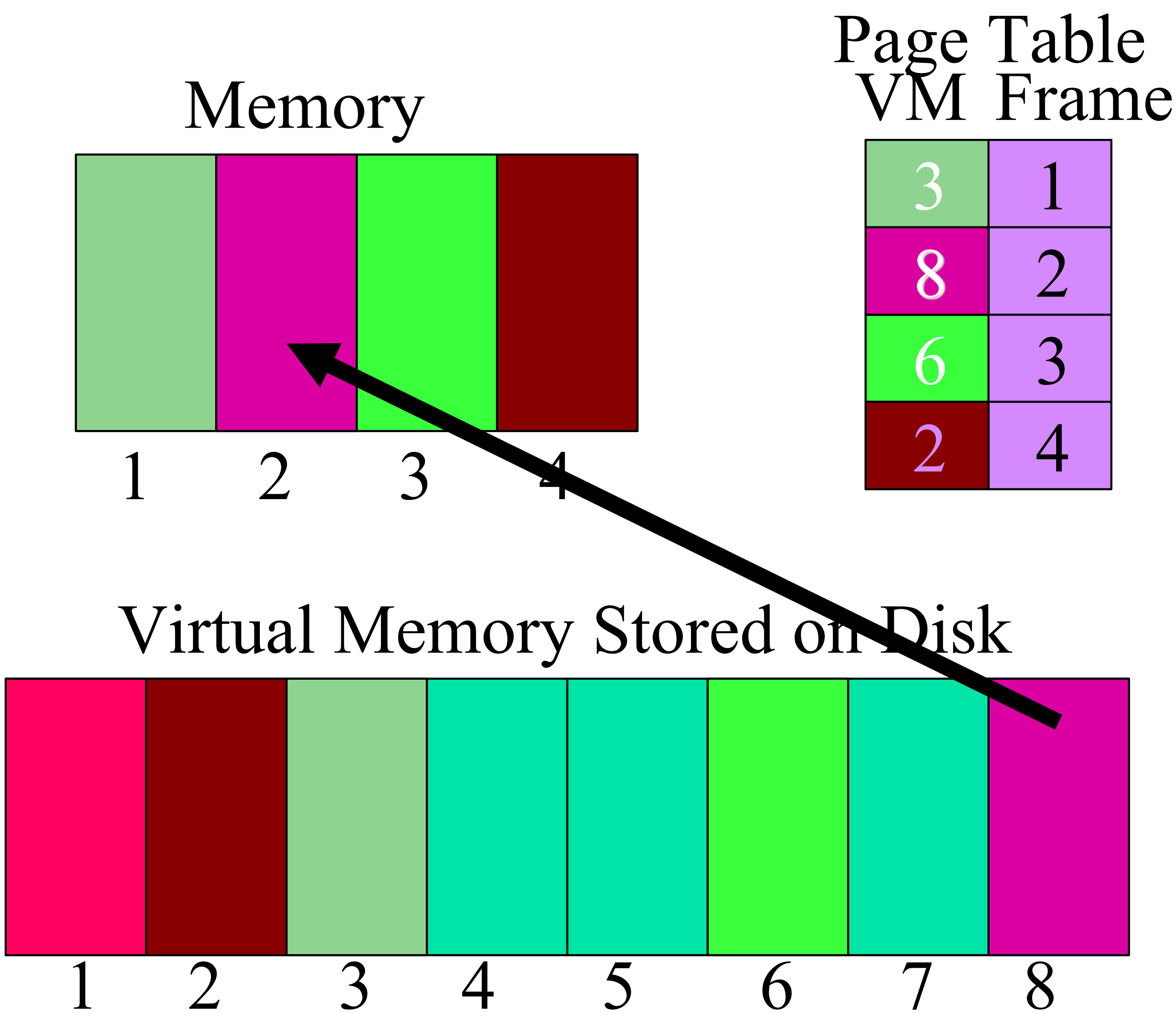
Paging

Request Page 8. Swap Page 1 to Disk First…

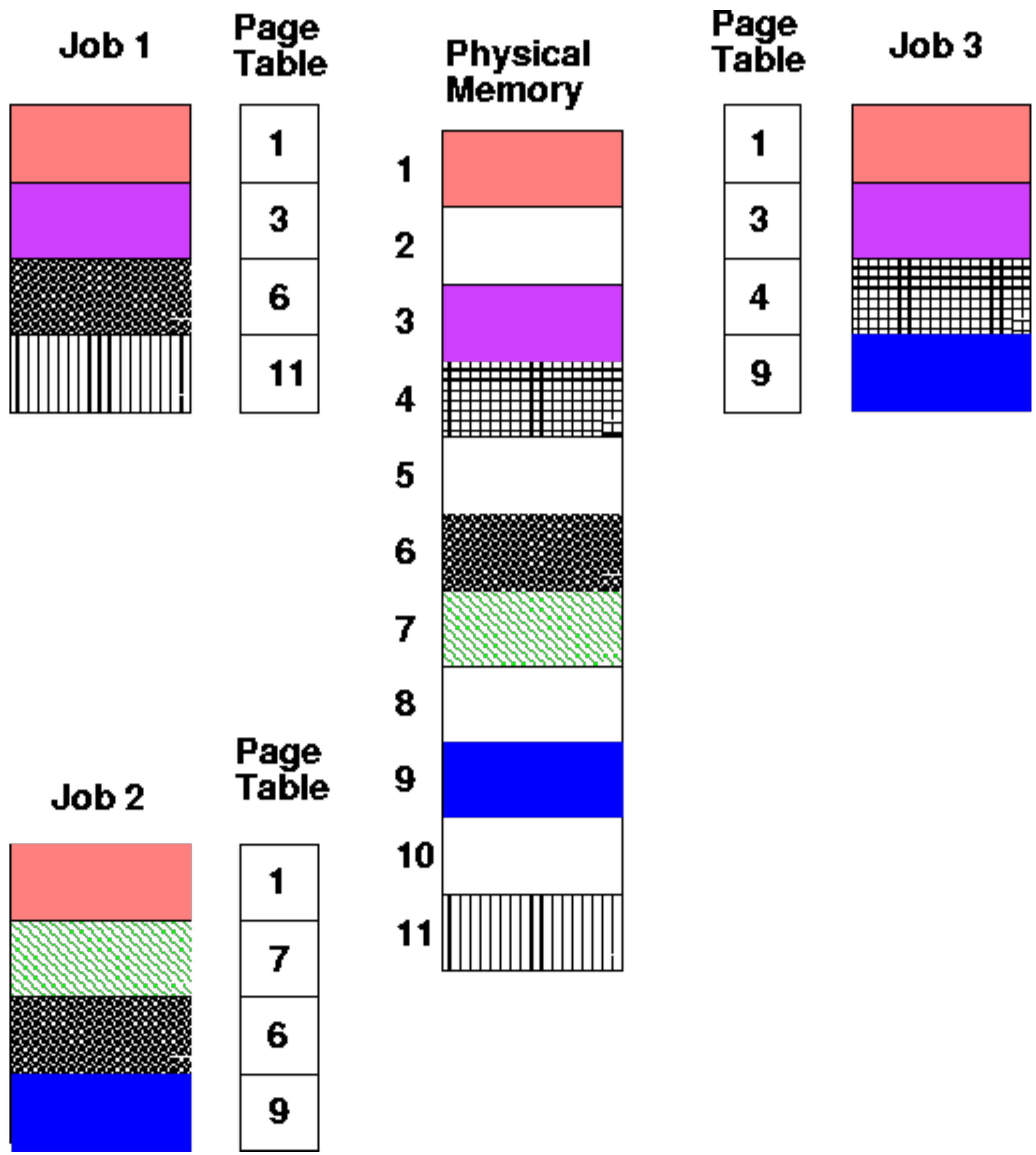


Paging

Request Page 8. … now load Page 8 into Memory.

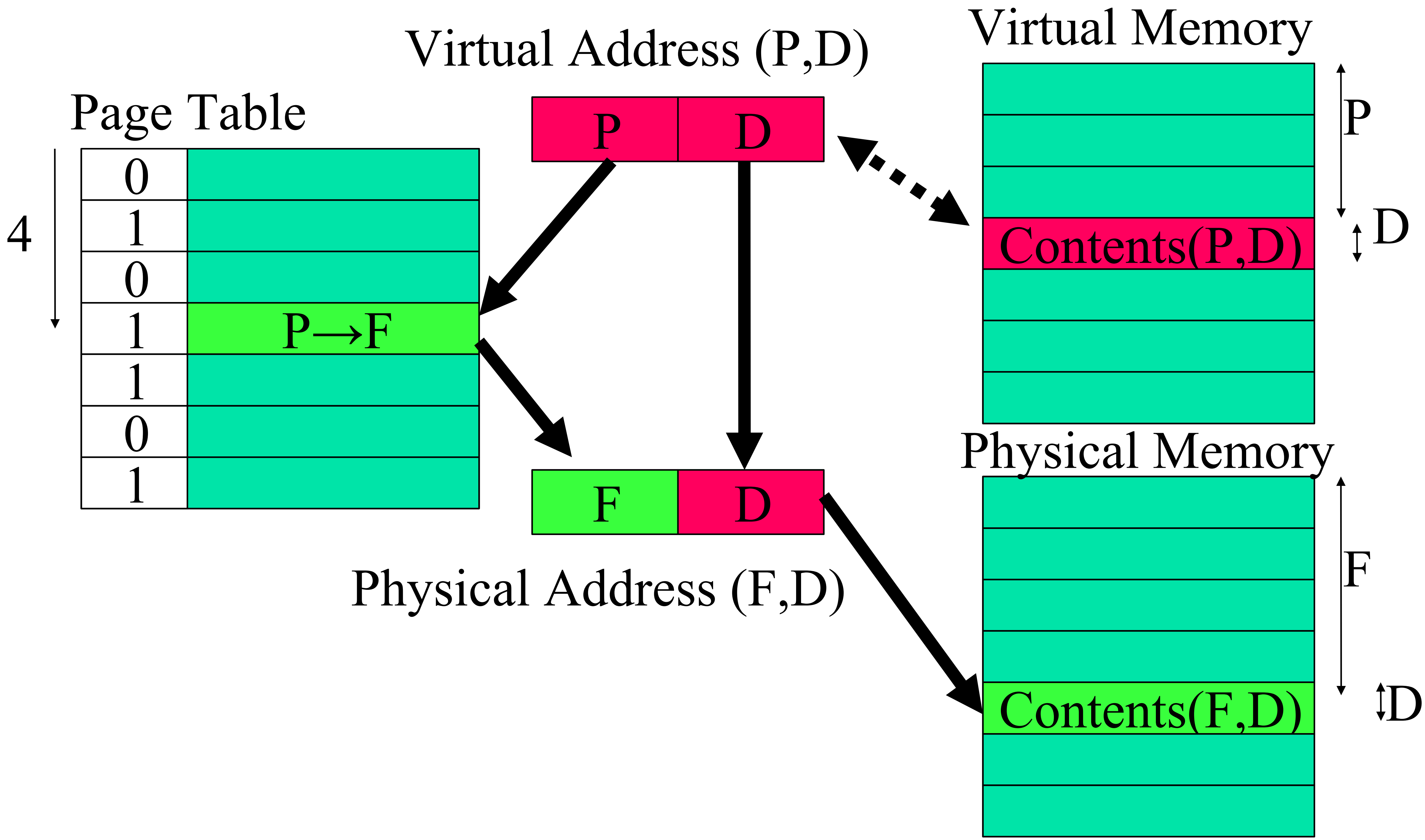


Shared Pages

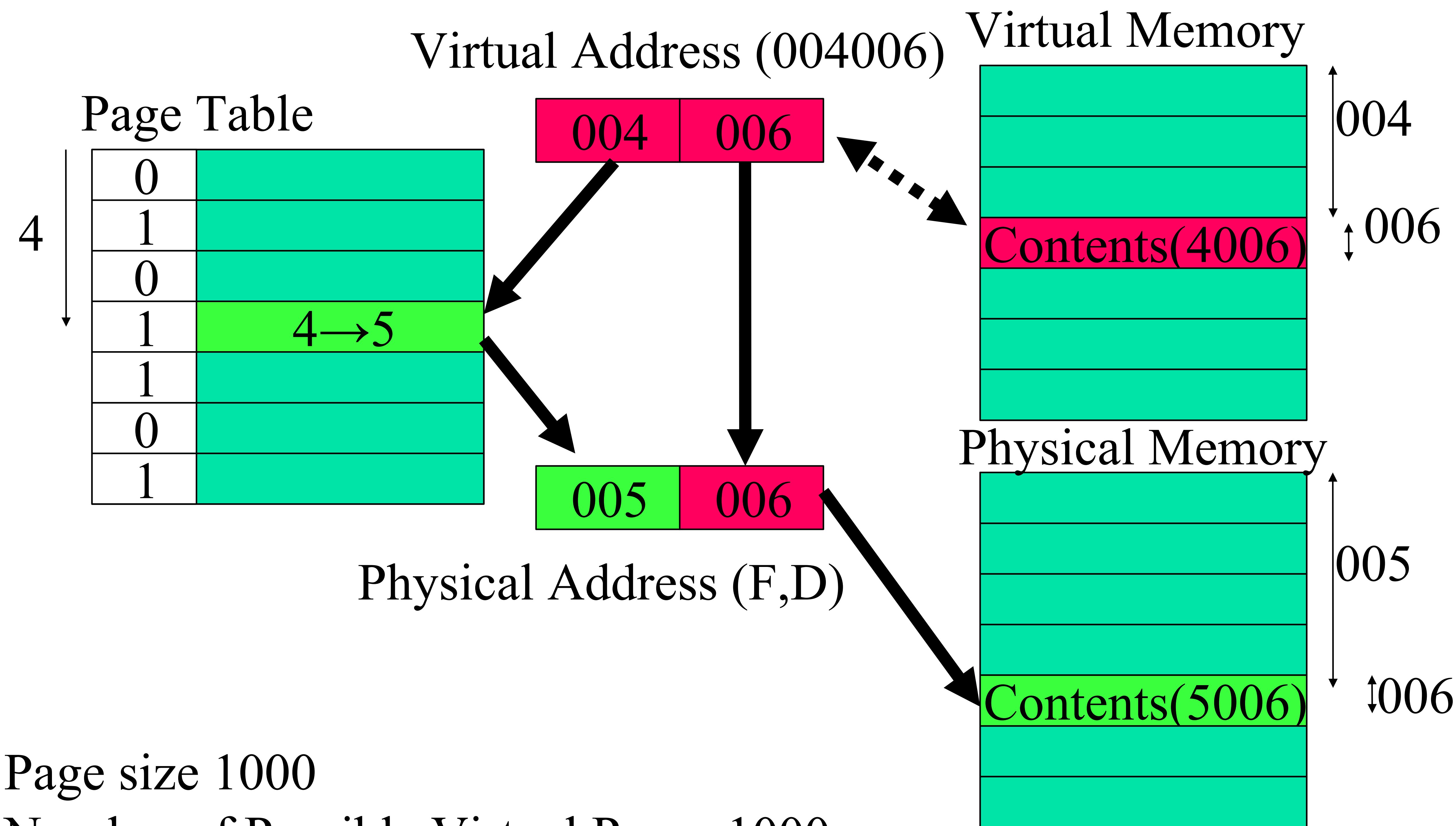


Note: Virtual Memory also supports shared pages.

Page Mapping Hardware



Page Mapping Hardware



Page size 1000

Number of Possible Virtual Pages 1000

Number of Page Frames 8

Page Faults

- Occur when we access a virtual page that is not mapped into any physical page
  - A fault is triggered by hardware
- Page fault handler (in OS's VM subsystem)
  - Find if there is any free physical page available
    - If no, evict some resident page to disk (swapping space)
  - Allocate a free physical page
  - Load the faulted virtual page to the prepared physical page
  - Modify the page table

Reasoning about Page Tables

- On a 32 bit system we have 2^32 B virtual address space
  - i.e., a 32 bit register can store 2^32 values
- # of pages are 2^n (e.g., 512 B, 1 KB, 2 KB, 4 KB…)
- Given a page size, how many pages are needed?
  - e.g., If 4 KB pages (2^12 B), then 2^32/2^12=…
    - 2^20 pages required to represent the address space
- **But**! each page entry takes more than 1 Byte of space to represent.
  - suppose page size is 4 bytes (Why?)
  - (2*2) * 2^ 20 = 4 MB of space required to represent our page table in physical memory.
- What is the consequence of this?

Paging Issues

- Page size is 2^n
  - usually 512 bytes, 1 KB, 2 KB, 4 KB, or 8 KB
  - E.g. 32 bit VM address may have 2^{20} (1 MB) pages with 4k (2^{12}) bytes per page
- Page table:
  - 2^{20} page entries take 2^{22} bytes (4 MB)
  - Must map into real memory
  - Page Table base register must be changed for context switch
- No external fragmentation; internal fragmentation on last page only