An Introduction to Memory Management: appendix

Memory partitioning

- Nowadays memory management is based on a sophisticated technique known as (paged) virtual memory.
- Before studying virtual memory, we will review simpler but outdated memory management mechanisms for multiprogramming systems:
  - Fixed partitioning
  - Dynamic partitioning

Copyright ©: Nahrstedt, Angrave, Abdelzaher

Fixed partitioning

- A simple scheme to manage the available user memory is to partition it in several regions of equal size (e.g., 8Mb partitions)

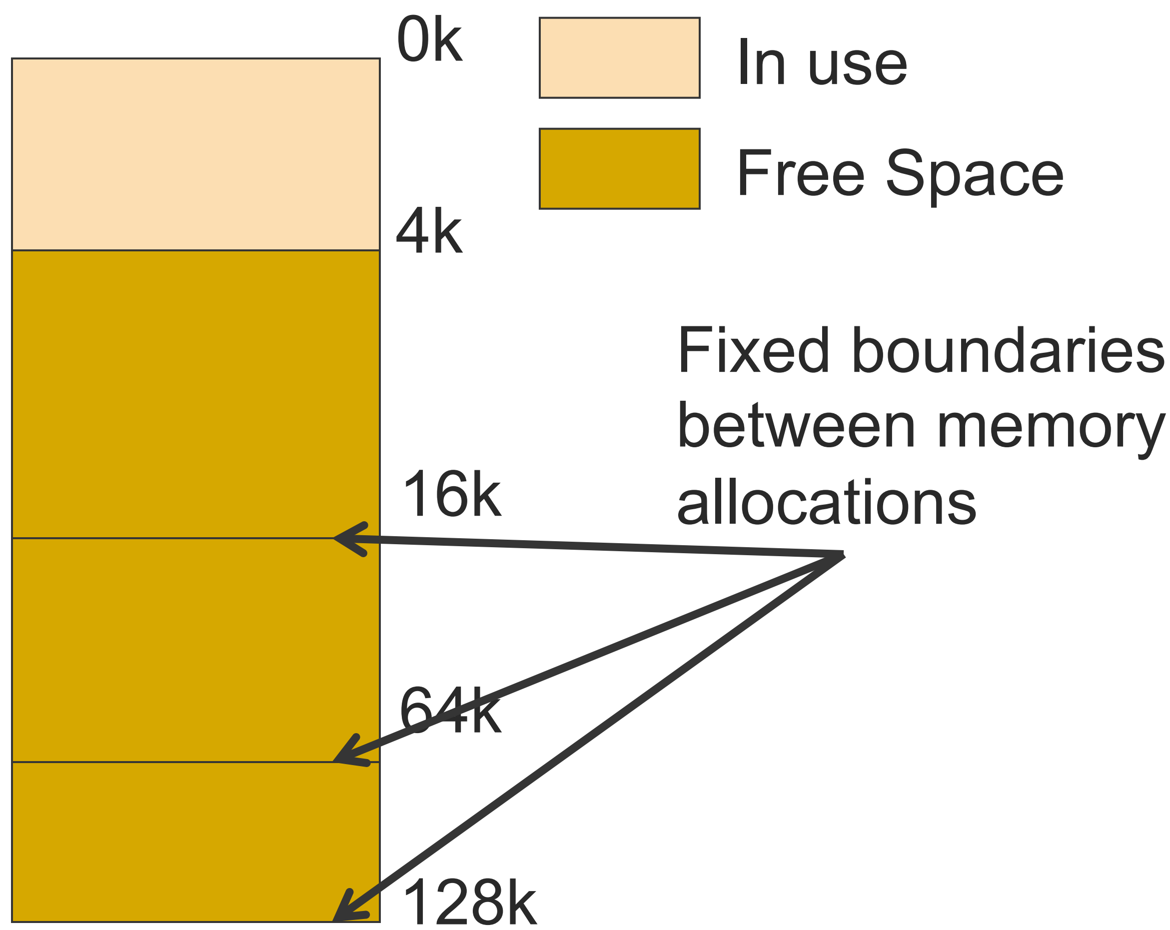
- As another option the memory can be partitioned in fixed regions of different sizes (e.g., 2Mb,4Mb,6Mb,8Mb partitions)

- Problems:
  - If a program does not fit in the available fixed size, the program must be designed by using overlays (only a part of the program needs to be in main memory at any given time)
  - Internal fragmentation
  - Not all processes may fit in memory

Copyright ©: Nahrstedt, Angrave, Abdelzaher

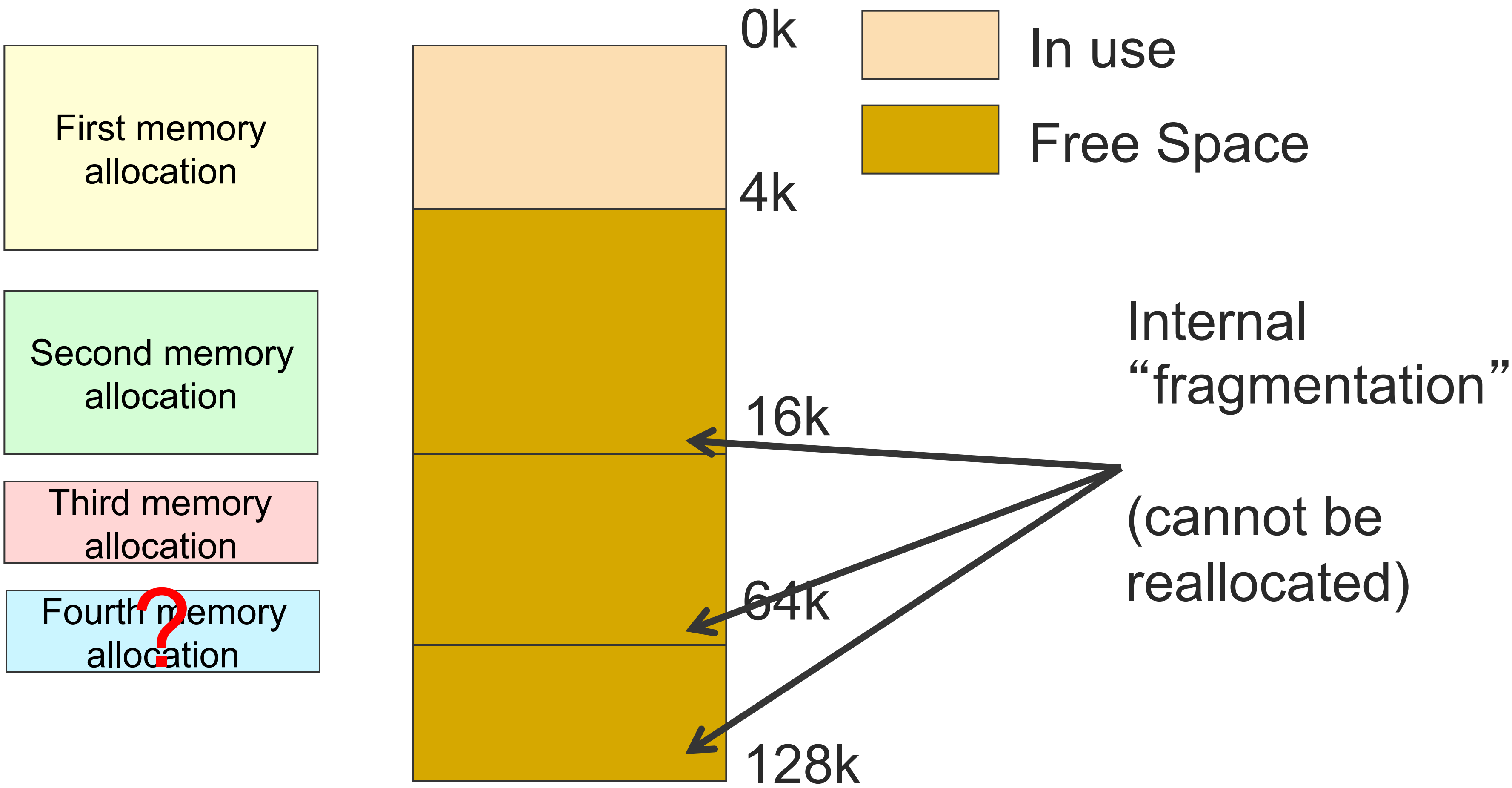
Multiple Fixed Partitions

Divide memory into *n* (possibly unequal) partitions.

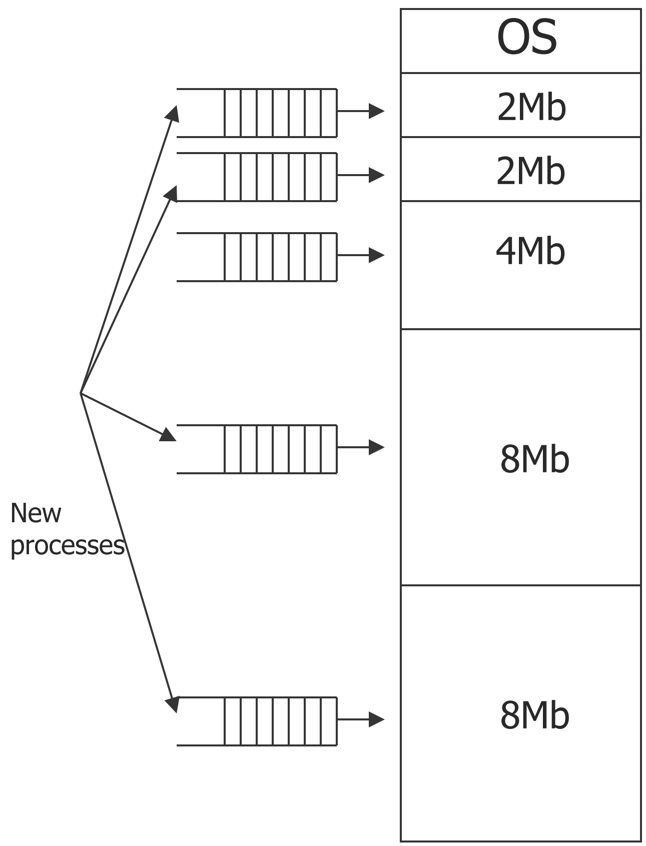


Copyright ©: University of Illinois CS 241 Staff

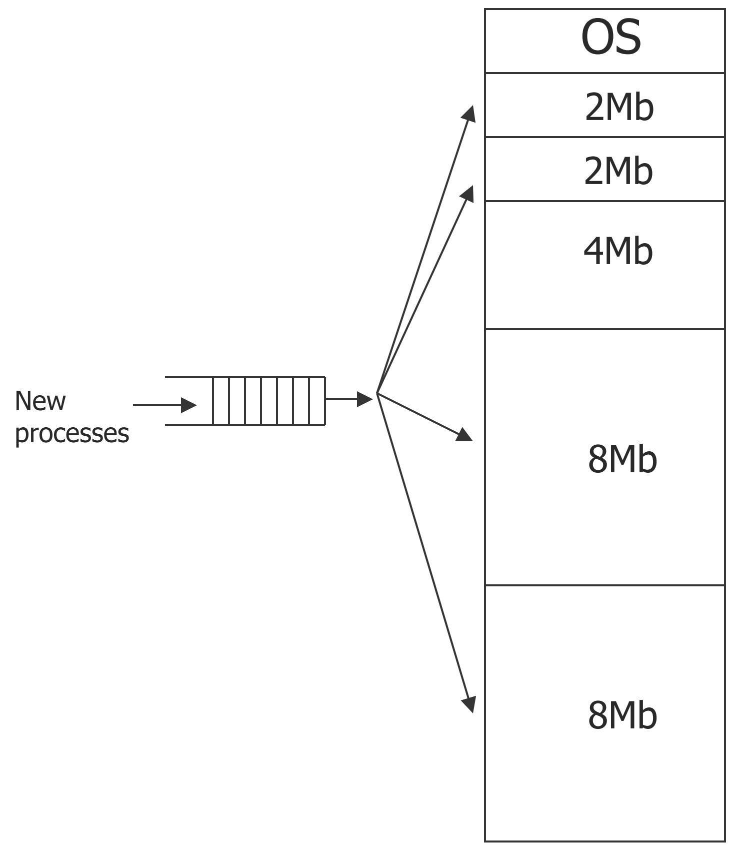
Multiple Fixed Partitions



Memory assignment for fixed partitioning



A process is always reloaded
in the same partition
(relocation is not needed)



A process can be reloaded
in any partition
(relocation is needed)

Memory assignment for fixed partitioning

- Separate input queue for each partition
  - Requires sorting the incoming jobs and putting them 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 memory has plenty of free space.

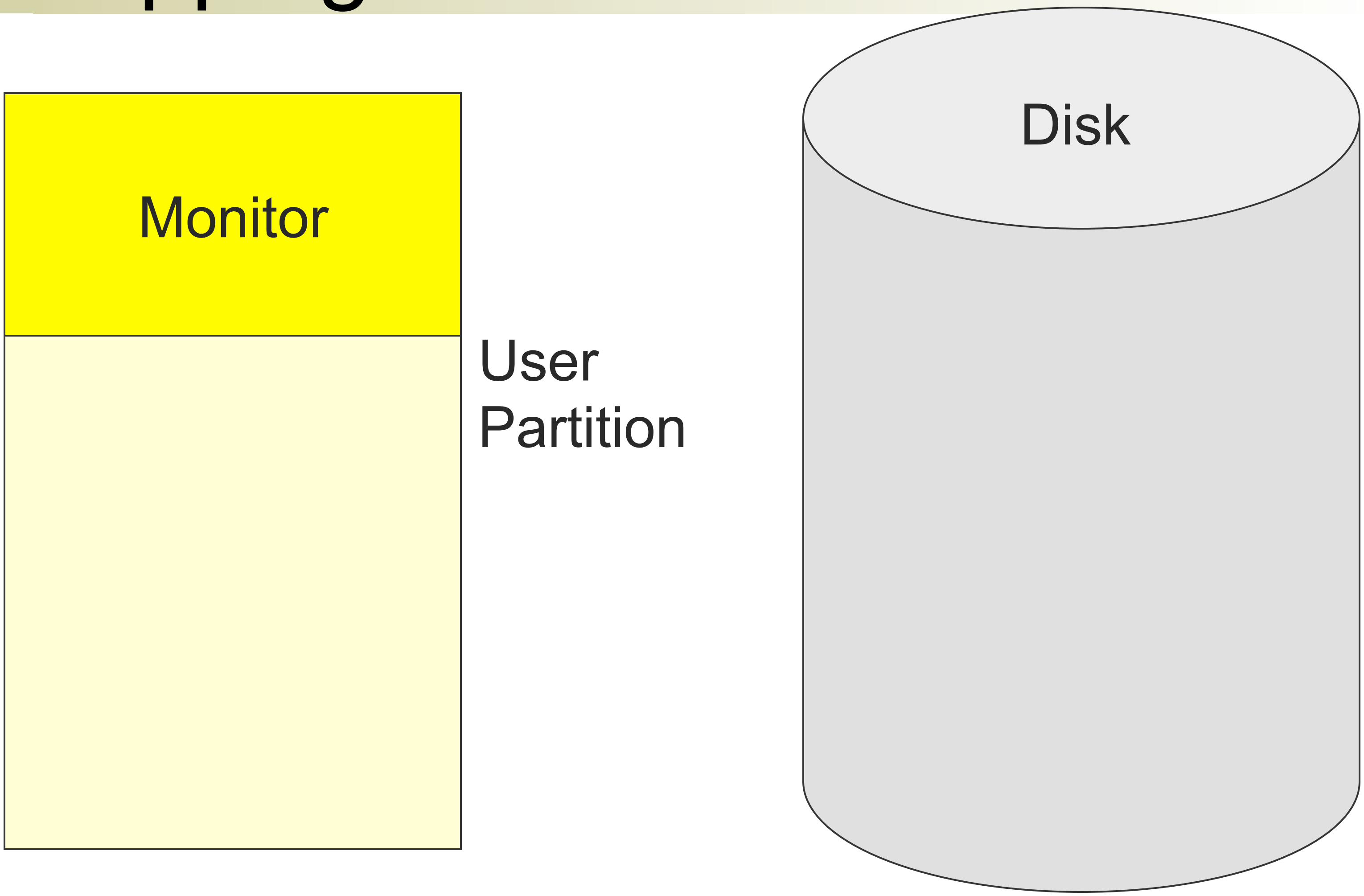
- One single input queue for all partitions.
  - Allocate a partition where the job fits in.
    - Best Fit
    - Worst Fit
    - First Fit

Copyright ©: Nahrstedt, Angrave, Abdelzaher

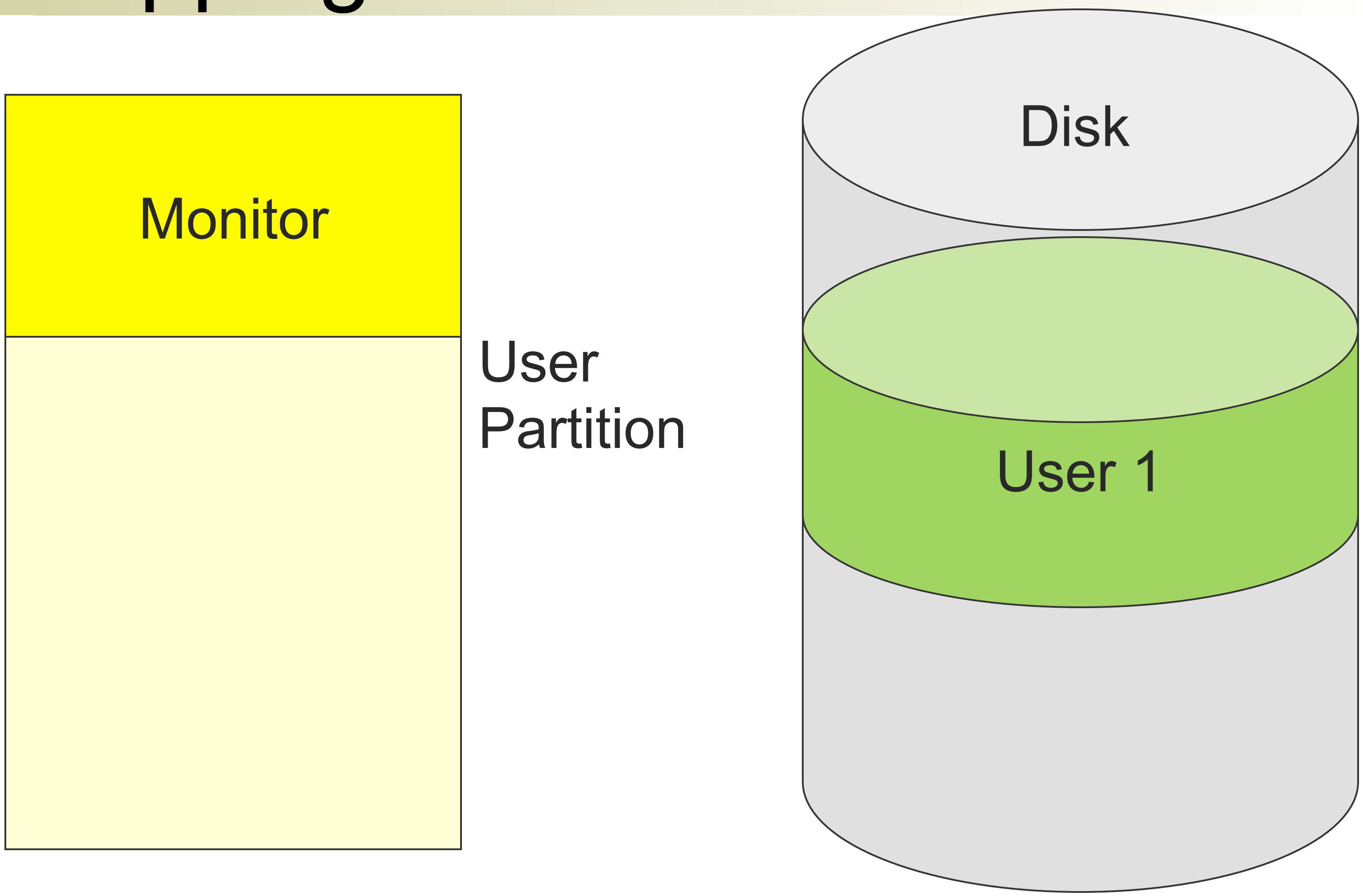
Problem: Insufficient Memory

- What if there are more processes than could fit into the memory?
- Swapping

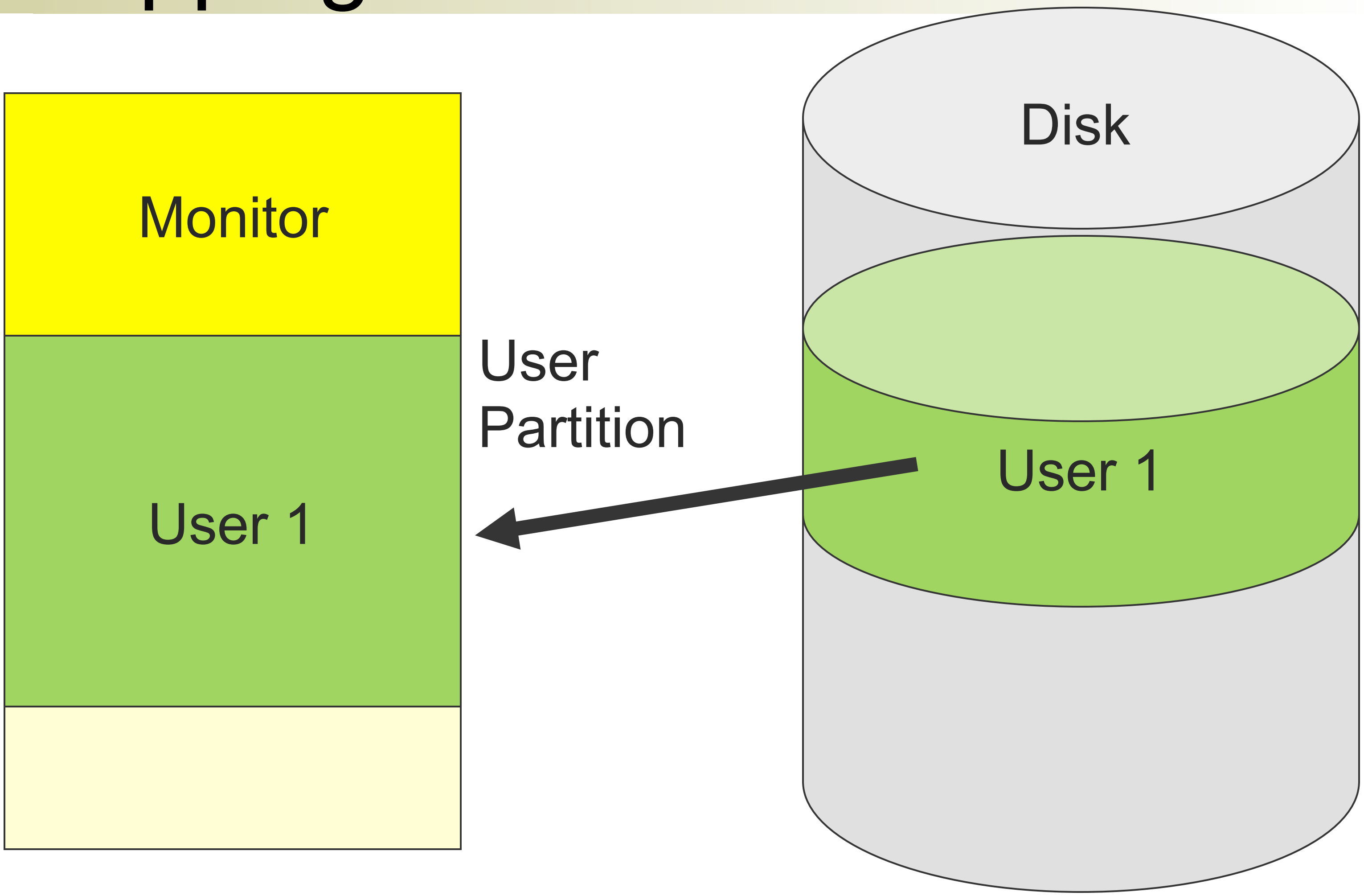
Swapping



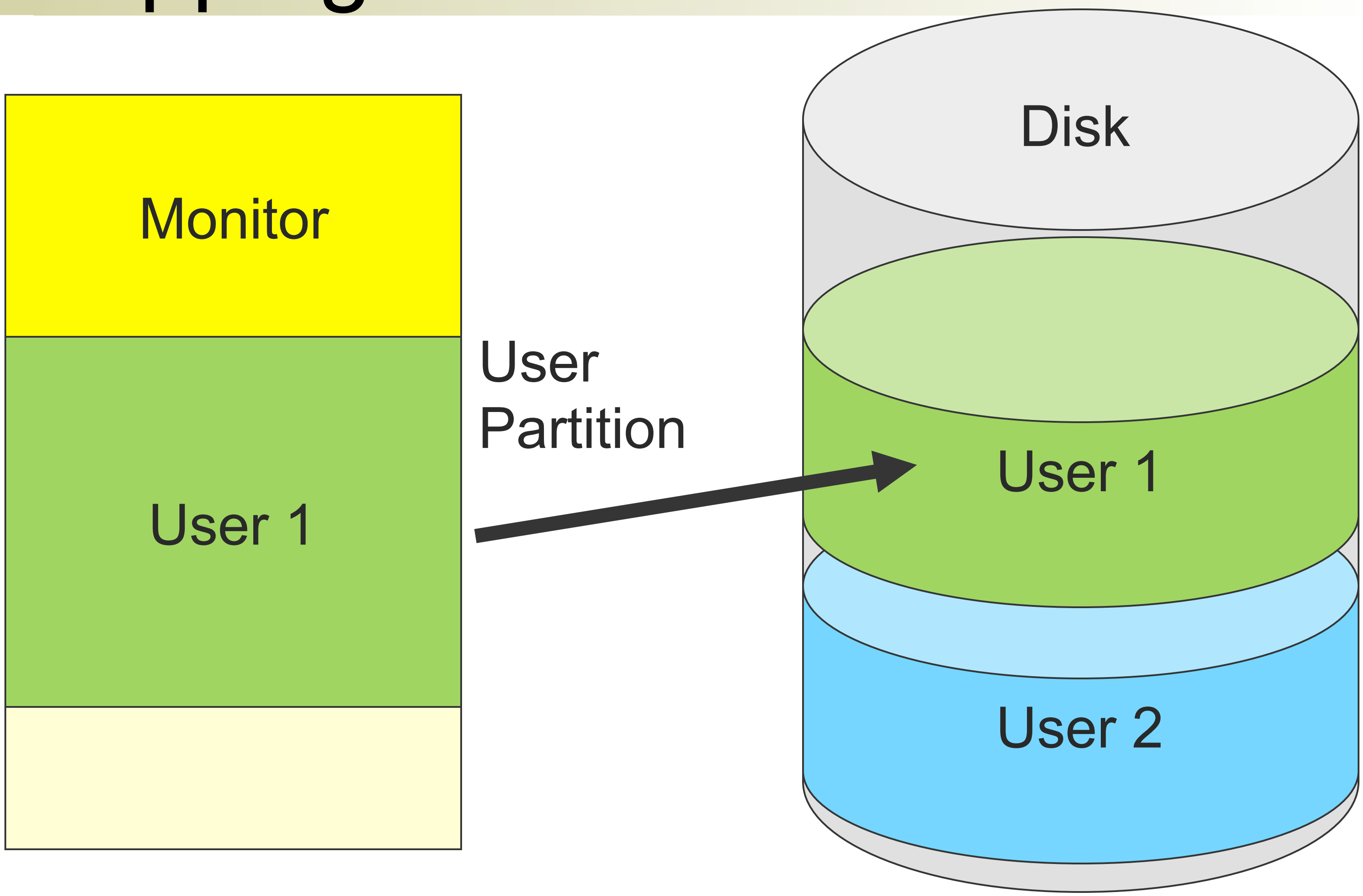
Swapping



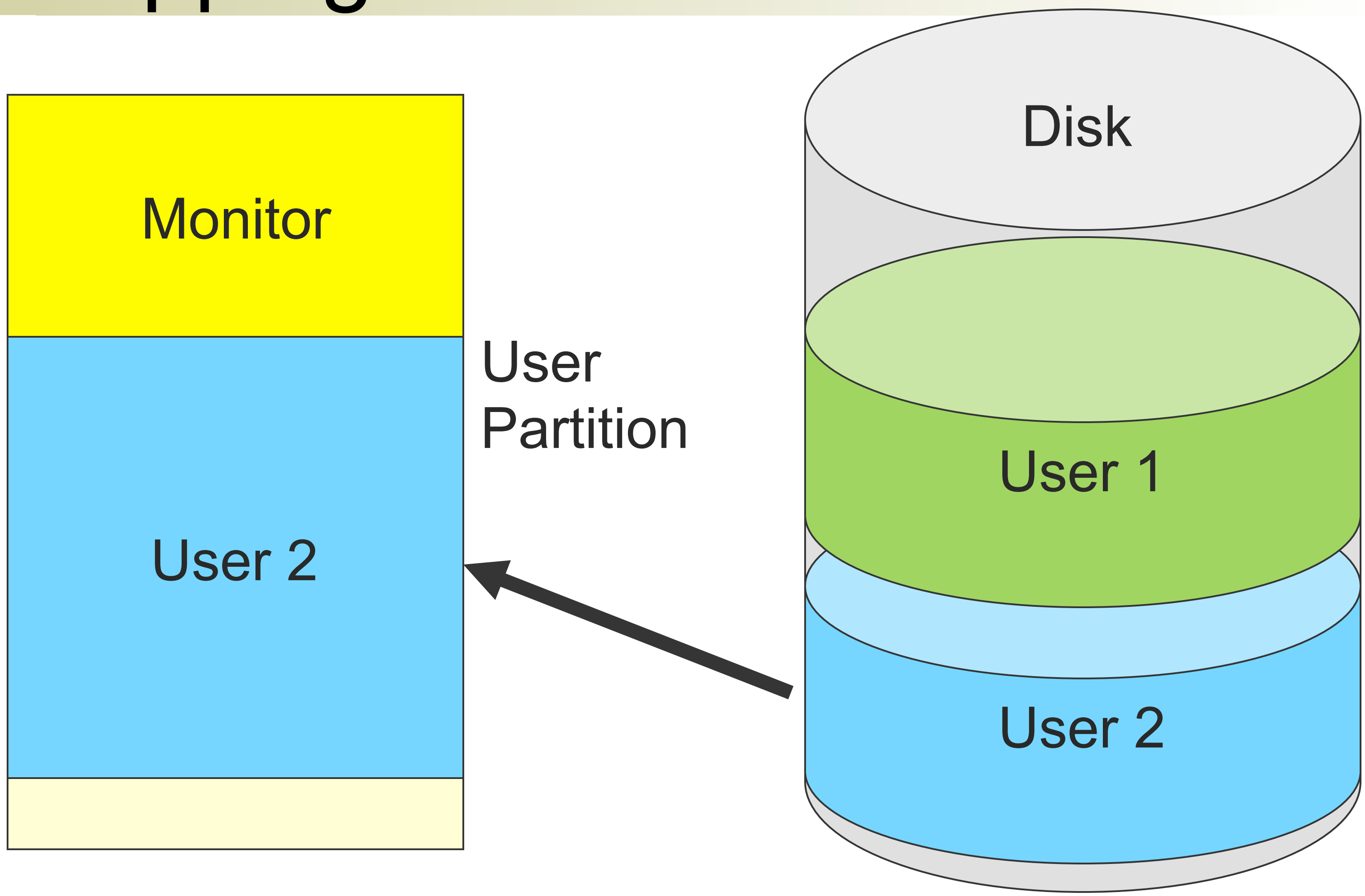
Swapping



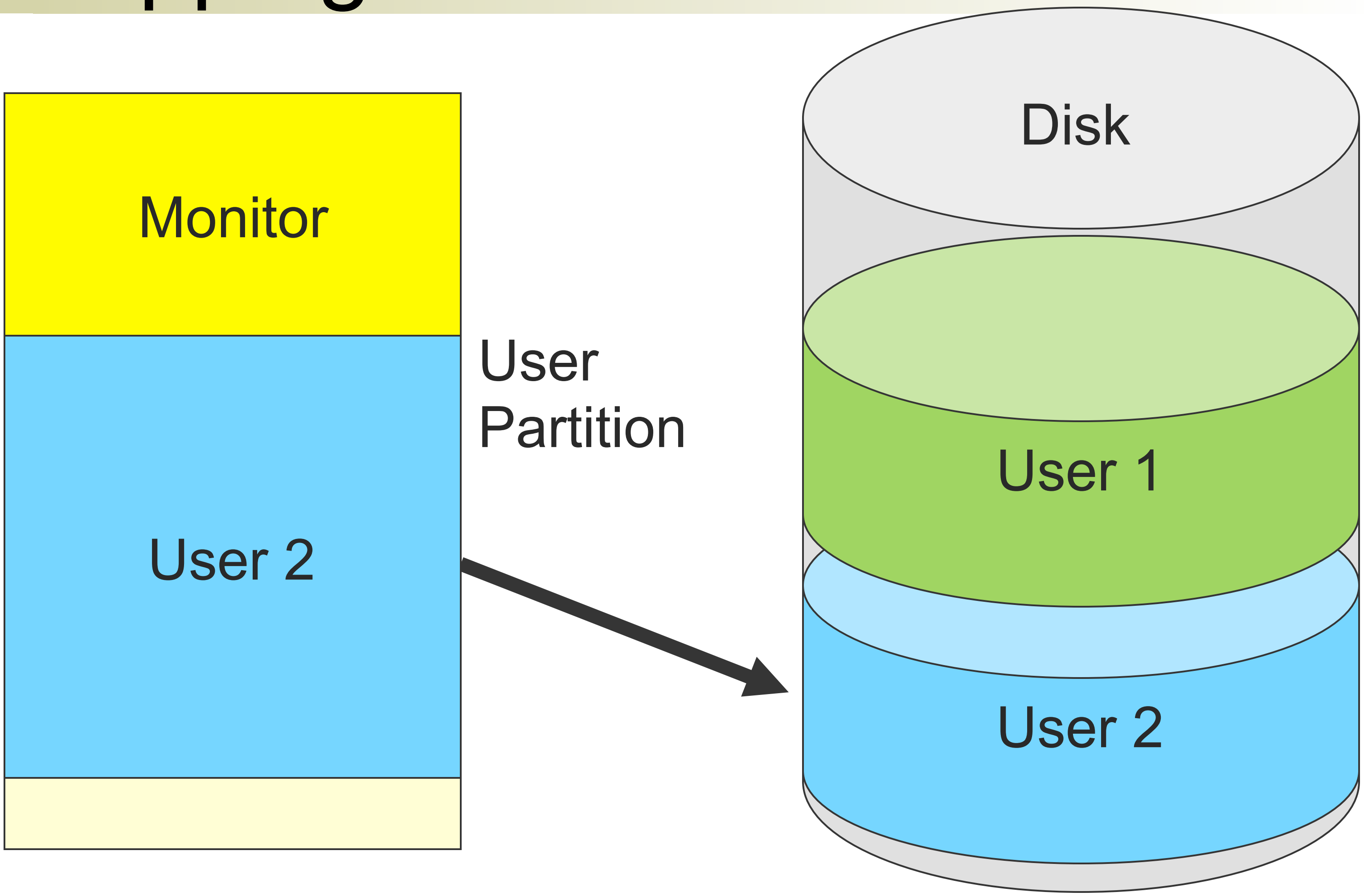
Swapping



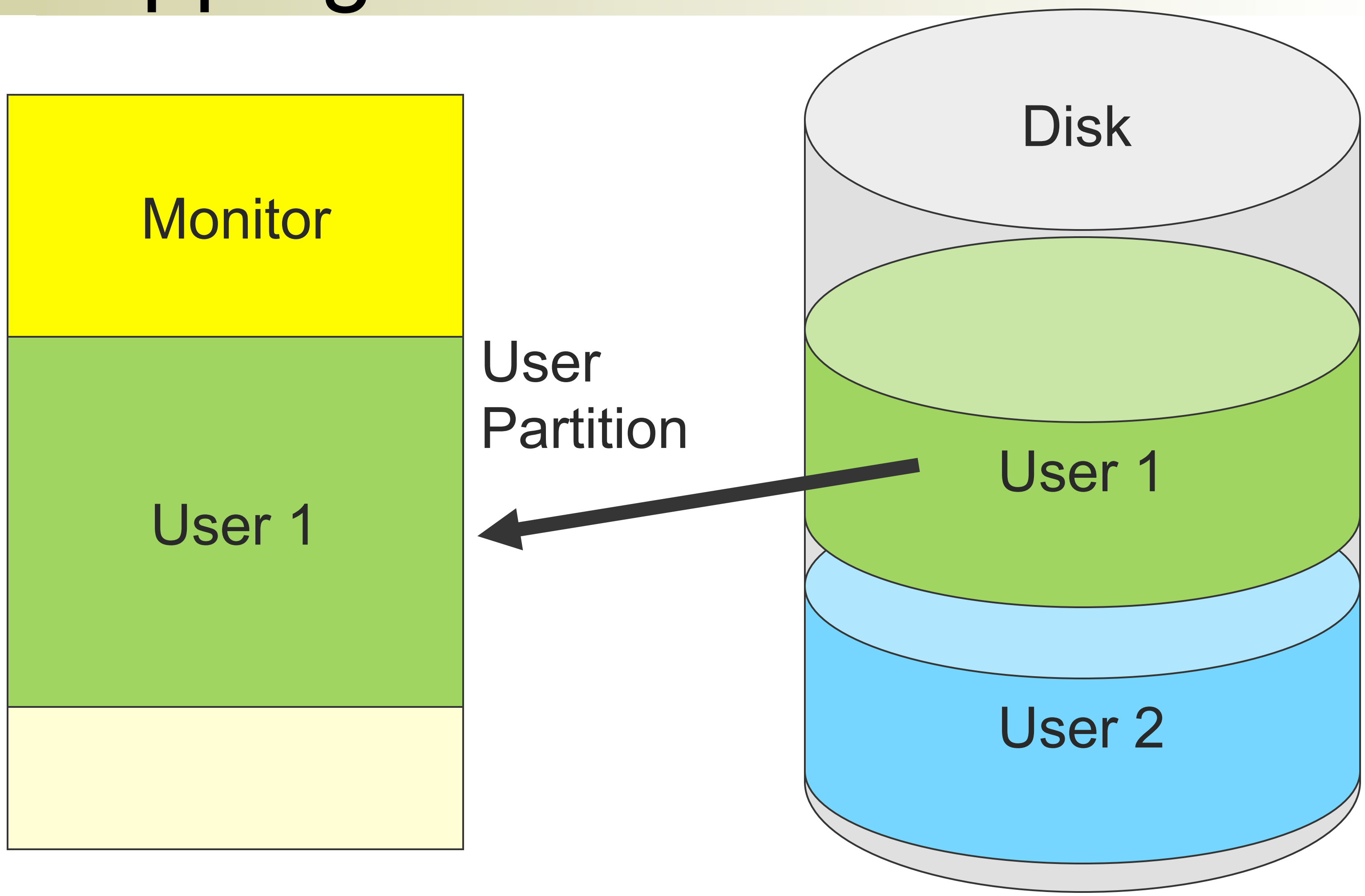
Swapping



Swapping

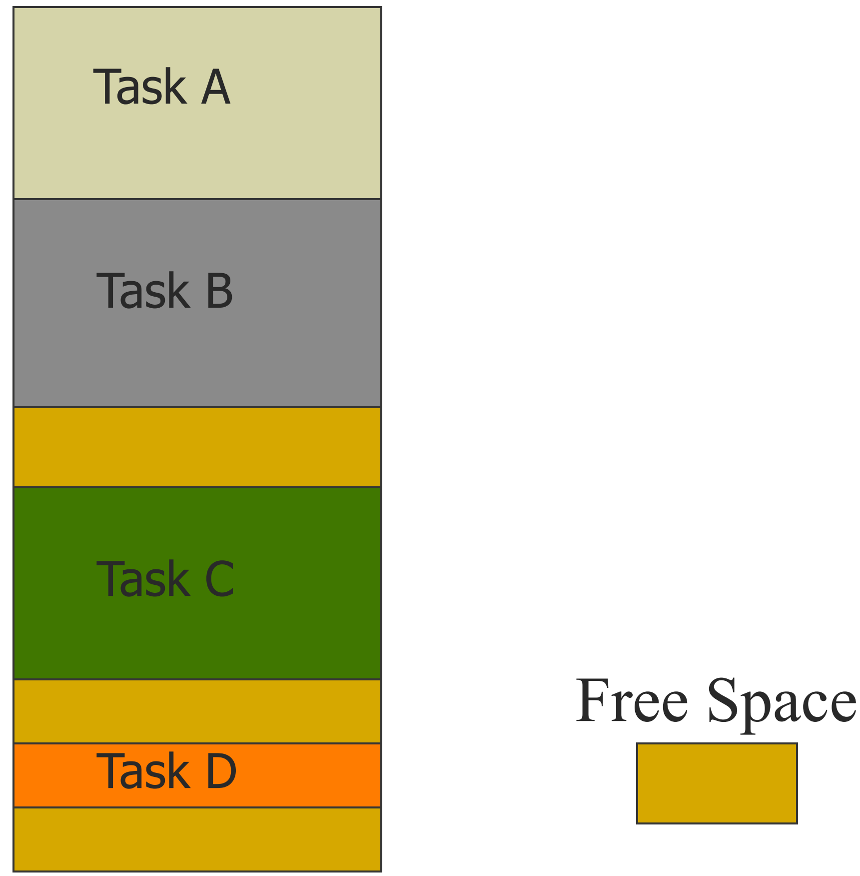


Swapping



Dynamic partitions

- Partitions are of variable length and number
- When a process is loaded, it is allocated exactly as much memory as it needs and no more
- Problems:
  - External fragmentation
  - It requires periodic compaction and task relocation



Copyright ©: Nahrstedt, Angrave, Abdelzaher

16

Storage Placement Strategies

- First fit
  - Use the first available hole whose size is sufficient to meet the need
- Best fit
  - Use the hole whose size is equal to the need, or if none is equal, the hole that is larger but closest in size
- Worst fit
  - Use the largest available hole

Copyright ©: University of Illinois CS 241 Staff

Example

- Consider a system in which memory consists of the following hole sizes in memory order:
  - 10K, 4K, 20K, 18K, 7K, 9K, 12K, and 15K.
  - Which hole is taken for successive segment requests of:
    - 12K
    - 10K
    - 9K

Example

- Consider a system in which memory consists of the following hole sizes in memory order:
  - 10K, 4K, 20K, 18K, 7K, 9K, 12K, and 15K.
  - Which hole is taken for successive segment requests of:
    - 12K
    - 10K
    - 9K

First fit:	Best fit:	Worst fit:
20K, 10K, 18K.	12K, 10K, 9K.	20K, 18K, and 15K.

Storage Placement Strategies

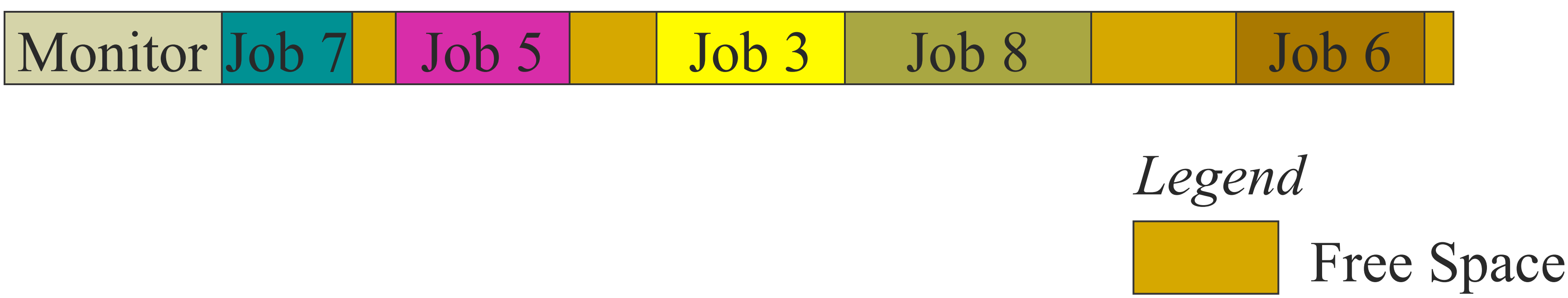
- Best fit
  - Produces the smallest leftover hole
  - Creates small holes that cannot be used
- Worst Fit
  - Produces the largest leftover hole
  - Difficult to run large programs
- First Fit
  - Creates average size holes

- First-fit and best-fit better than worst-fit in terms of speed and storage utilization

Copyright ©: University of Illinois CS 241 Staff

Fragmentation

- Internal Fragmentation
  - When an allocated block is larger than data it holds
- External Fragmentation
  - When *aggregate* free space would be large enough to satisfy request but no *single* free block is large enough



Copyright ©: Nahrstedt, Angrave, Abdelzaher

Fragmentation

- Internal Fragmentation
  - Allocated memory may be larger than requested memory
  - The extra memory is internal to a partition and it cannot be used
- External Fragmentation
  - Memory space exists to satisfy a request, but it is not contiguous

How Bad Is Fragmentation?

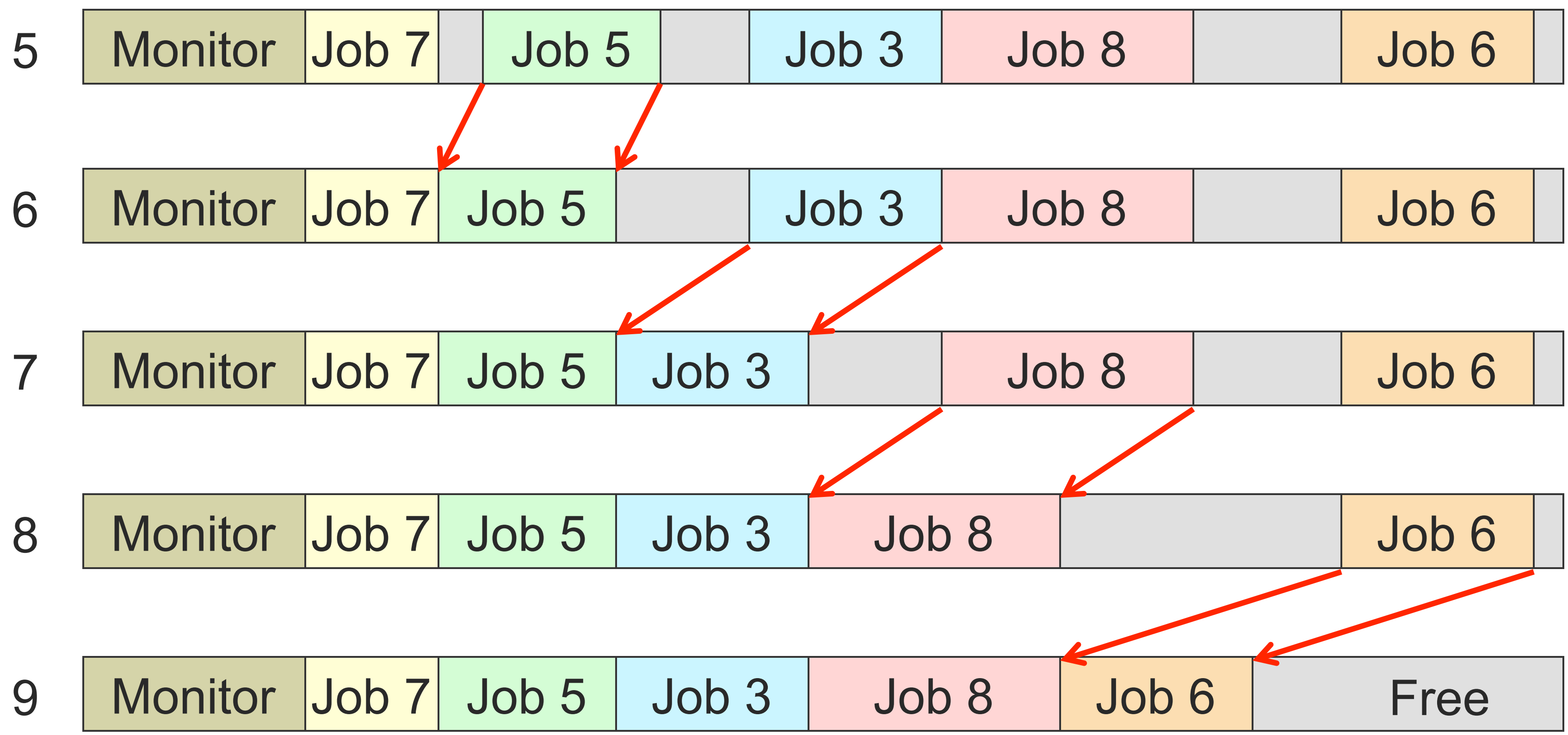
- Statistical analysis - Random job sizes
- First-fit
  - Given N allocated blocks
  - 0.5*N blocks will be wasted *on average*, because of internal fragmentation

- Known as 50% RULE

Compaction

- Reduce external fragmentation by compaction
  - Move jobs in memory to place all free memory together in one large block
  - Compaction is possible only if run-time process relocation is supported

Compaction



Copyright ©: University of Illinois CS 241 Staff

Storage Management Problems

- Fixed partitions suffer from
- Dynamic partitions suffer from
- Compaction suffers from

Copyright ©: University of Illinois CS 241 Staff

Storage Management Problems

- Fixed partitions suffer from
  - Internal fragmentation
- Dynamic partitions suffer from
  - External fragmentation
- Compaction suffers from
  - Overhead

Copyright ©: University of Illinois CS 241 Staff

Relocation

- Assume relocation is not supported: when the process is first loaded, all memory references in the code are replaced by absolute main memory addresses
  - Different processes will be loaded at different absolute addresses or swapping is necessary
  - It is a strong limitation since a swapped process must always be reloaded in the same partition

- Relocation allows a process to occupy different partitions during its lifetime. It relies on notion of logical and physical addresses (it requires hardware support)
  - Logical addresses: range from 0 to max
  - Physical addresses: range from R+0 to R+max (given base value R).
  - User program never sees the real physical addresses

28

Relocation: Logical vs. Physical Addresses

- Logical address
  - An address meaningful to the user process
  - A translation must be made to a physical address before the memory access can be achieved
- Physical address
  - It is an actual location in main (physical) memory
- Different processes run at different physical addresses
  - But logical address can be the same
  - Program never sees physical addresses

Copyright ©: University of Illinois CS 241 Staff

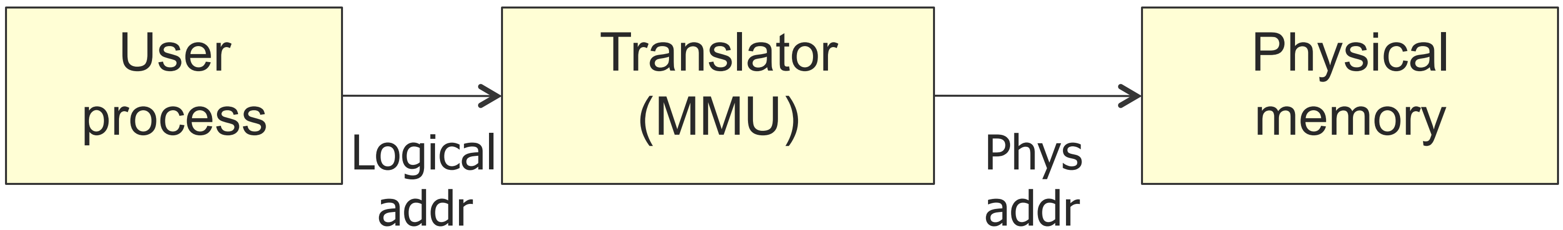
Relocation: Dynamic Address Translation

- Load each process into contiguous regions of physical memory

- Logical addresses
  - Logical address space
  - Range: 0 to MAX

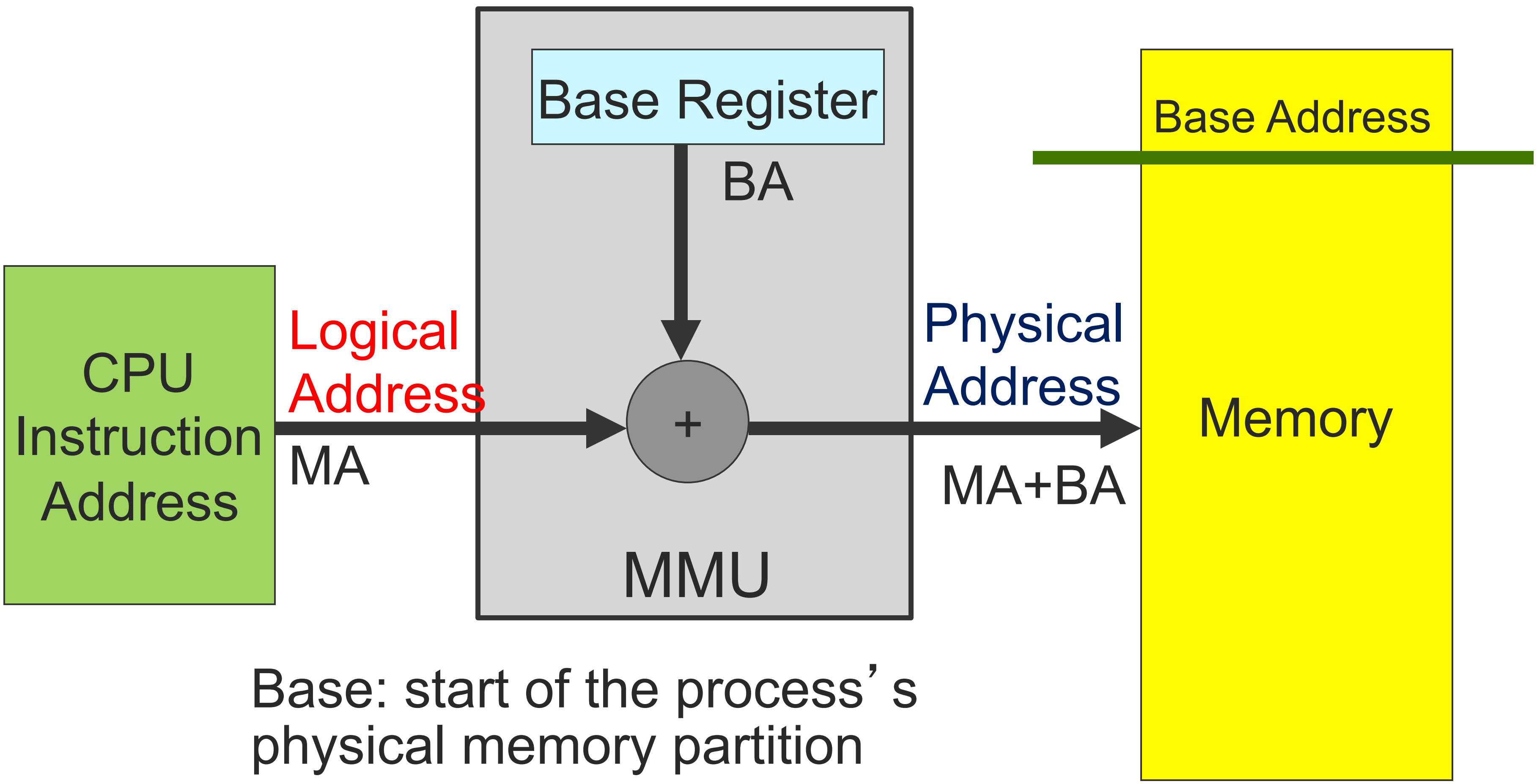
- Physical addresses
  - Physical address space
  - Range: R+0 to R+MAX for base value R

Dynamic Address Translation



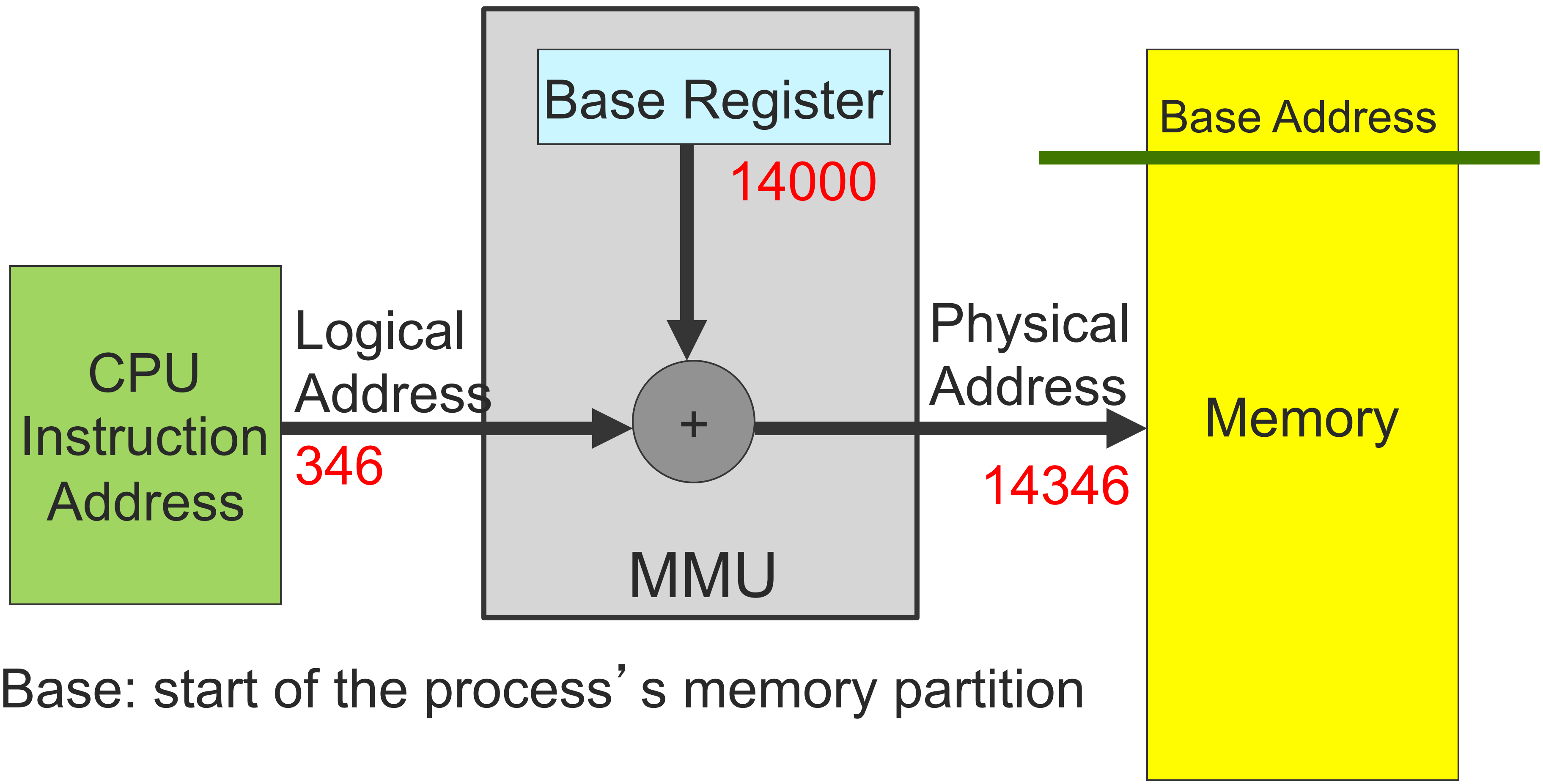
- Translation enforces protection
  - One process can't even refer to another process's physical address space
- Translation enables relocation and protection

Base Register



Base: start of the process's
physical memory partition

Base Register



Base: start of the process' s memory partition

Protection

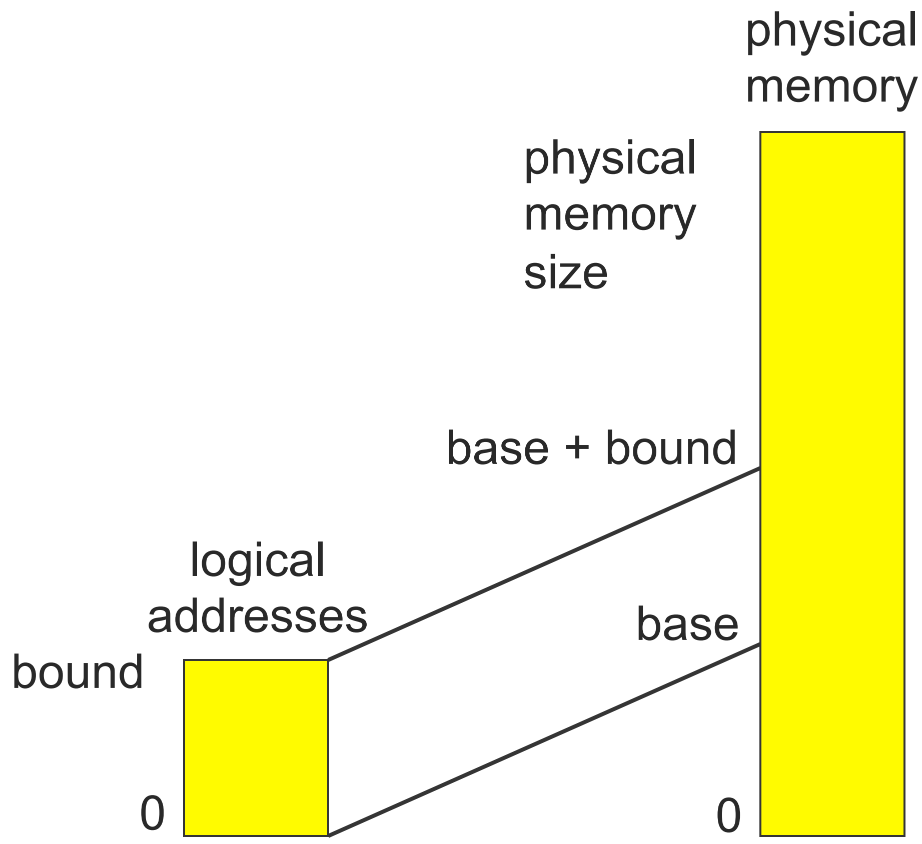
- Problem
  - How to prevent a malicious process from writing or jumping into another user's or OS physical partitions
- Solution
  - Base bounds register

Copyright ©: University of Illinois CS 241 Staff

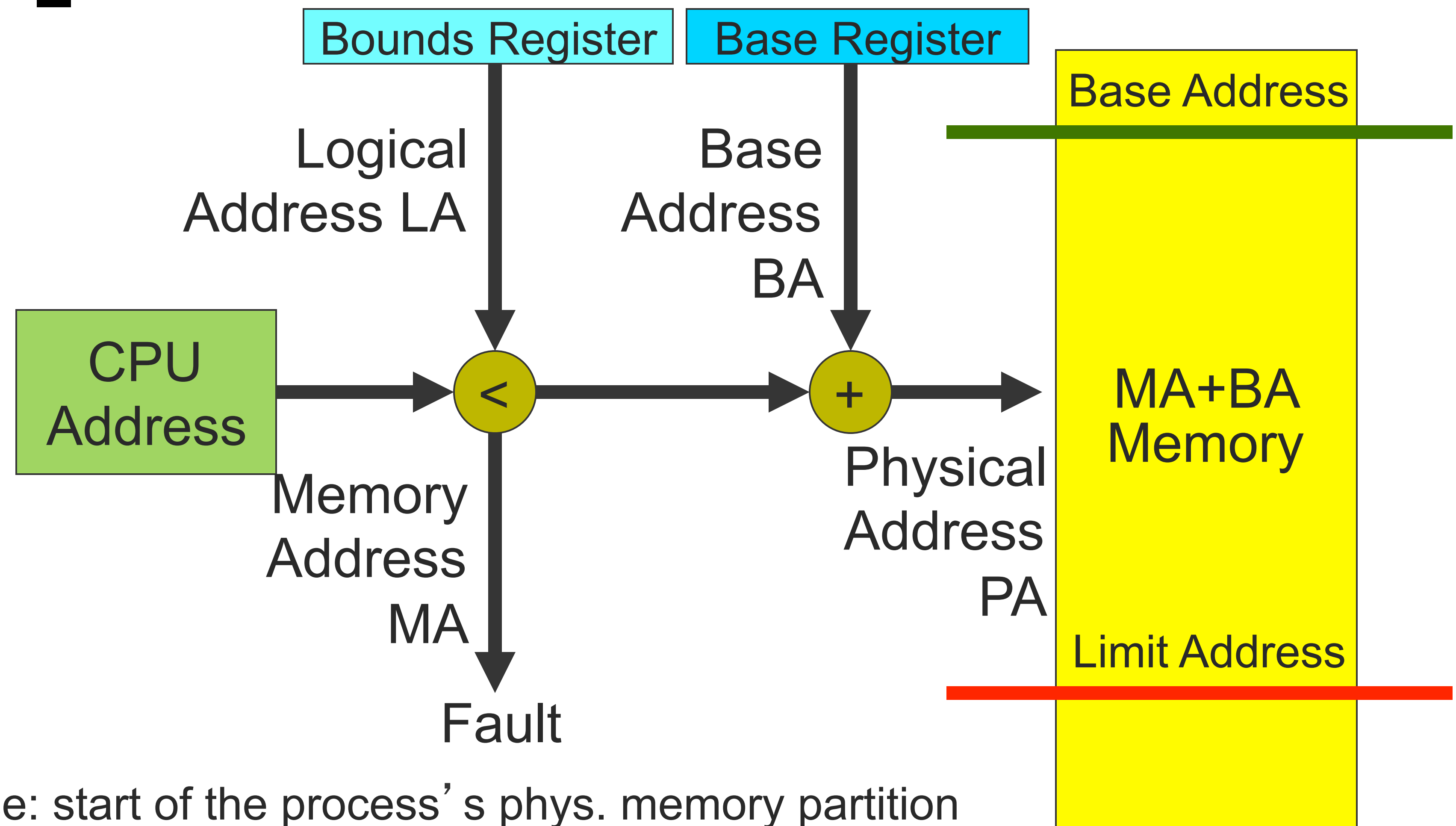
Base and bounds

```
if (logical_addr > bound)
    trap to kernel
} else {
    phys_addr =
      logical_addr + base
}
```

- Process can be relocated at run-time
- Provides protection from other processes also currently in memory



Base and Bounds Registers



Base: start of the process's phys. memory partition
Limit: max address in the process's phys. memory partition

Copyright ©: University of Illinois CS 241 Staff

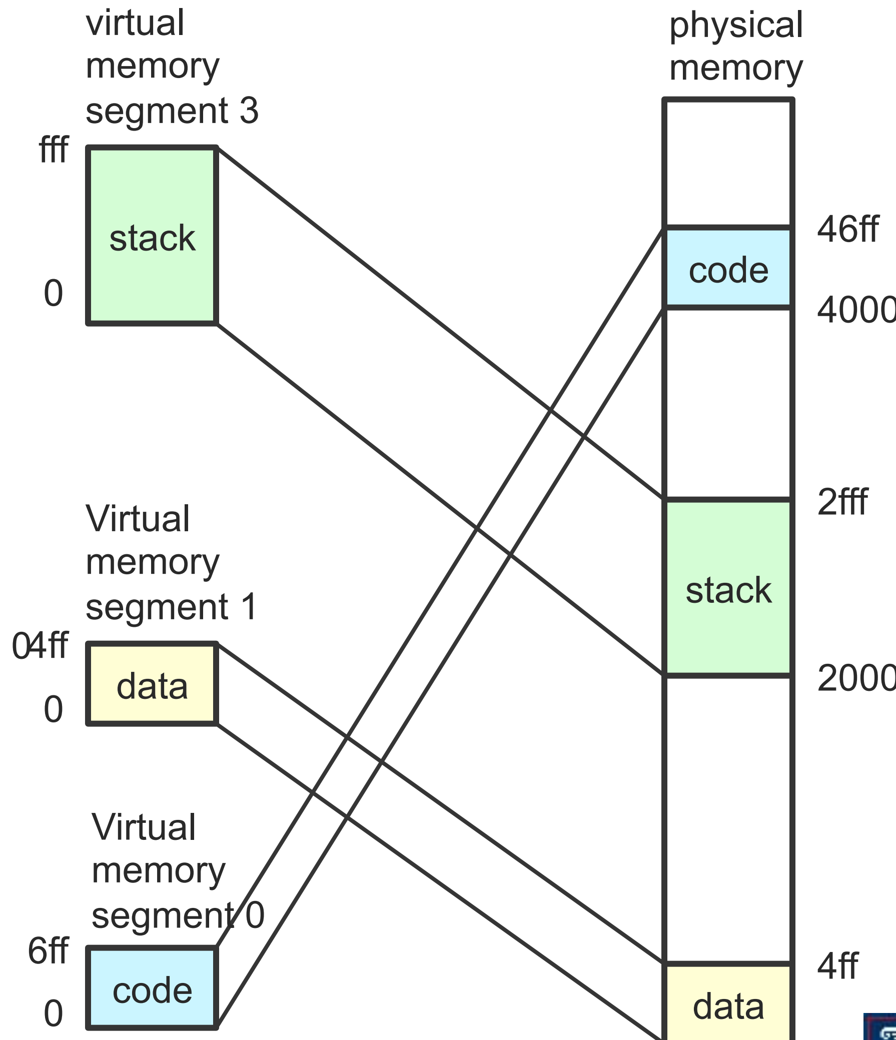
Another memory management technique: Segmentation

- Segment
  - Region of contiguous memory
- Segmentation
  - Generalized base and bounds with support for multiple segments at once

Copyright ©: University of Illinois CS 241 Staff

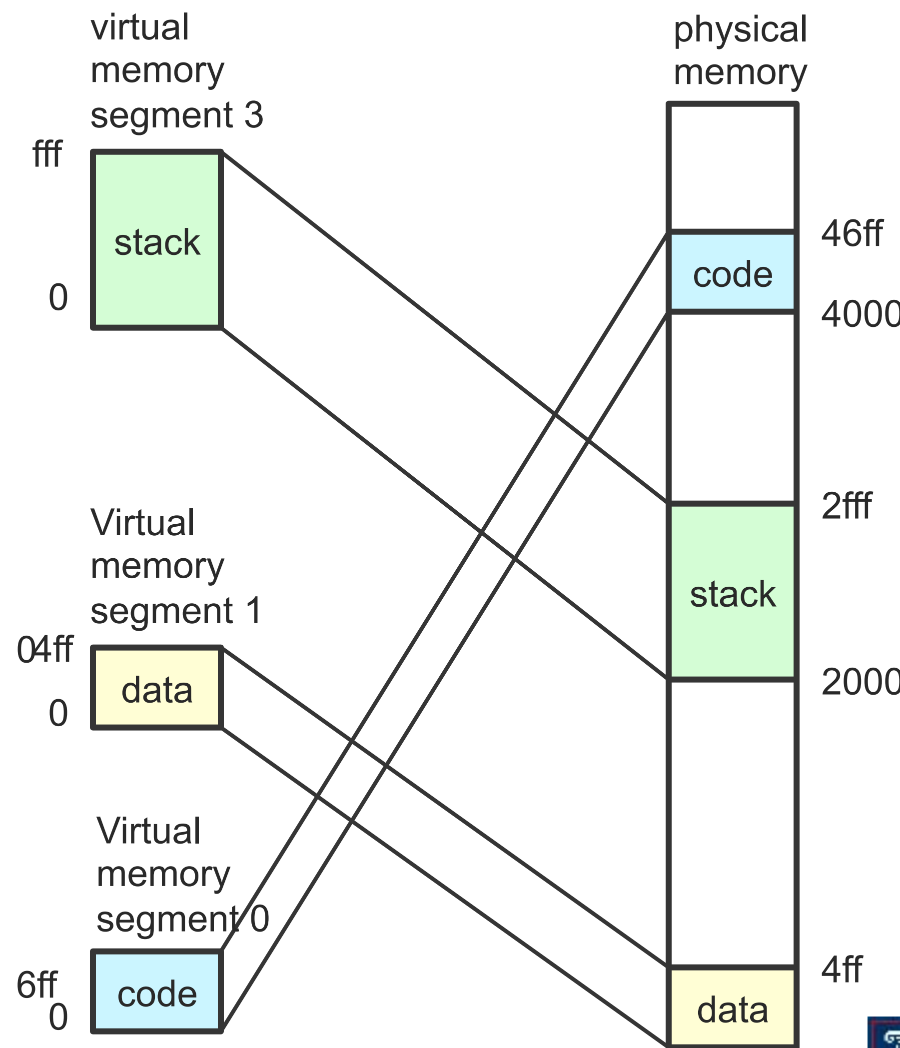
Segmentation

Seg #	Base	Bound	Description
0	4000	700	Code segment
1	0	500	Data segment
2	Unused		
3	2000	1000	Stack segment



Segmentation

- Segments: advantages over base and bounds?
- Protection
  - Different segments can have different protections
- Flexibility
  - It can separately grow both a stack and heap
  - Enables sharing of code and other segments if needed



Segmentation

- What abstraction is not supported well by segmentation and by B&B?
  - Supporting an address space larger than the size of physical memory

- Note: x86 used to support segmentation, **now effectively deprecated with x86-64**

Copyright ©: University of Illinois CS 241 Staff