Memory

Address Space Abstraction

- Address space
  - All memory data
  - i.e., program code, stack, data segment
- Hardware interface (physical reality)
  - Computer has one small, shared memory
- Application interface (illusion)
  - Each process wants private, large memory

How can we close this gap?

Copyright ©: University of Illinois CS 241 Staff

Address Space Illusions

- Address independence
  - Same address can be used in different address spaces yet remain logically distinct
- Protection
  - One address space cannot access data in another address space
- Virtual memory
  - Address space can be larger than the amount of physical memory on the machine

Copyright ©: University of Illinois CS 241 Staff

Address Space Illusions

Illusion

Giant address space
Protected from others
(Unless you want to share)
More whenever you want it

Reality

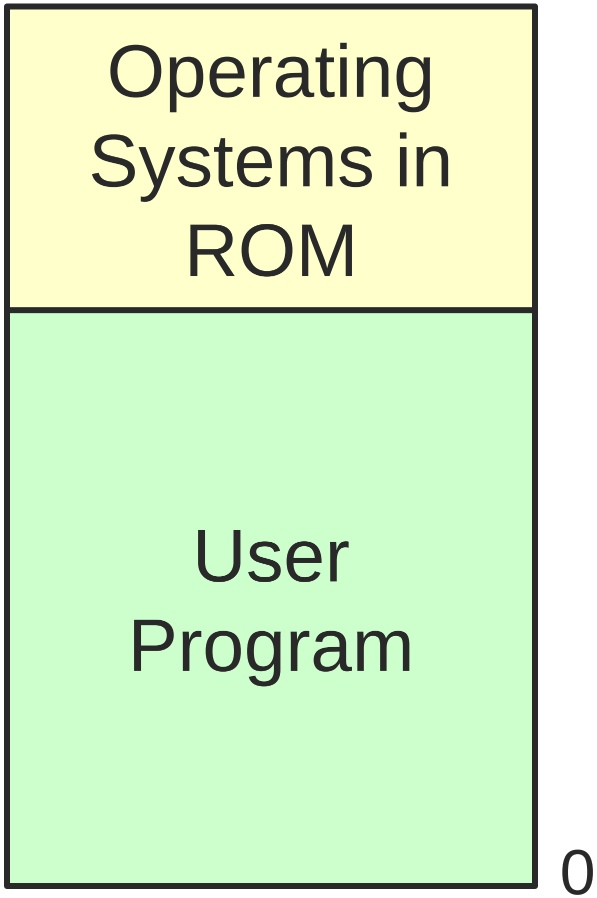
Many processes sharing
One address space
Limited memory

Today:
The story of the Illusion

Copyright ©: University of Illinois CS 241 Staff

Uni-programming

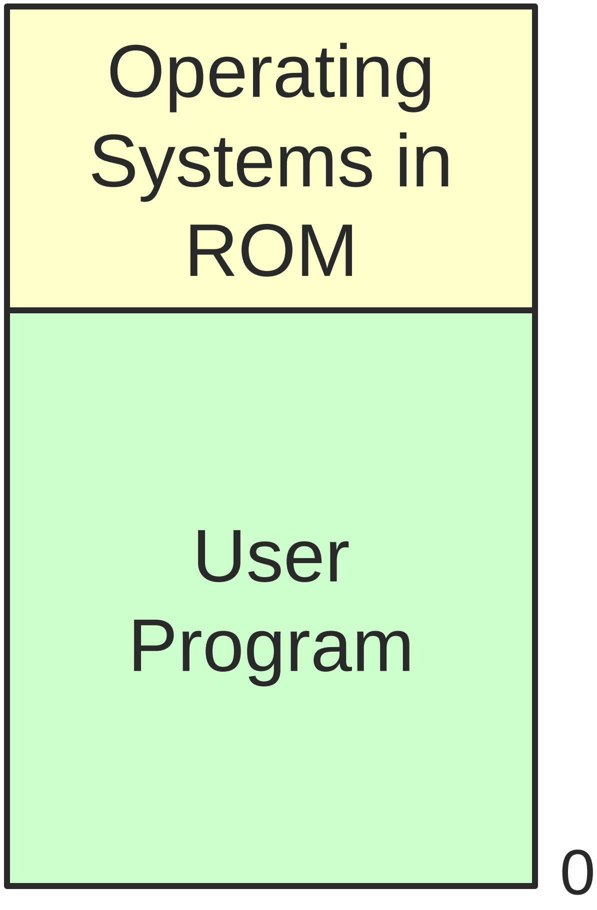
- 1 process runs at a time
- Always load process into the same spot
- How do you switch processes?
- What illusions does this provide?
  - Independence, protection, virtual memory?



Copyright ©: University of Illinois CS 241 Staff

Uni-programming

- 1 process runs at a time
- Always load process into the same spot
- How do you switch processes?
- What illusions does this provide?
  - Independence, protection, virtual memory?
- Problems?
  - Slow, large time slices



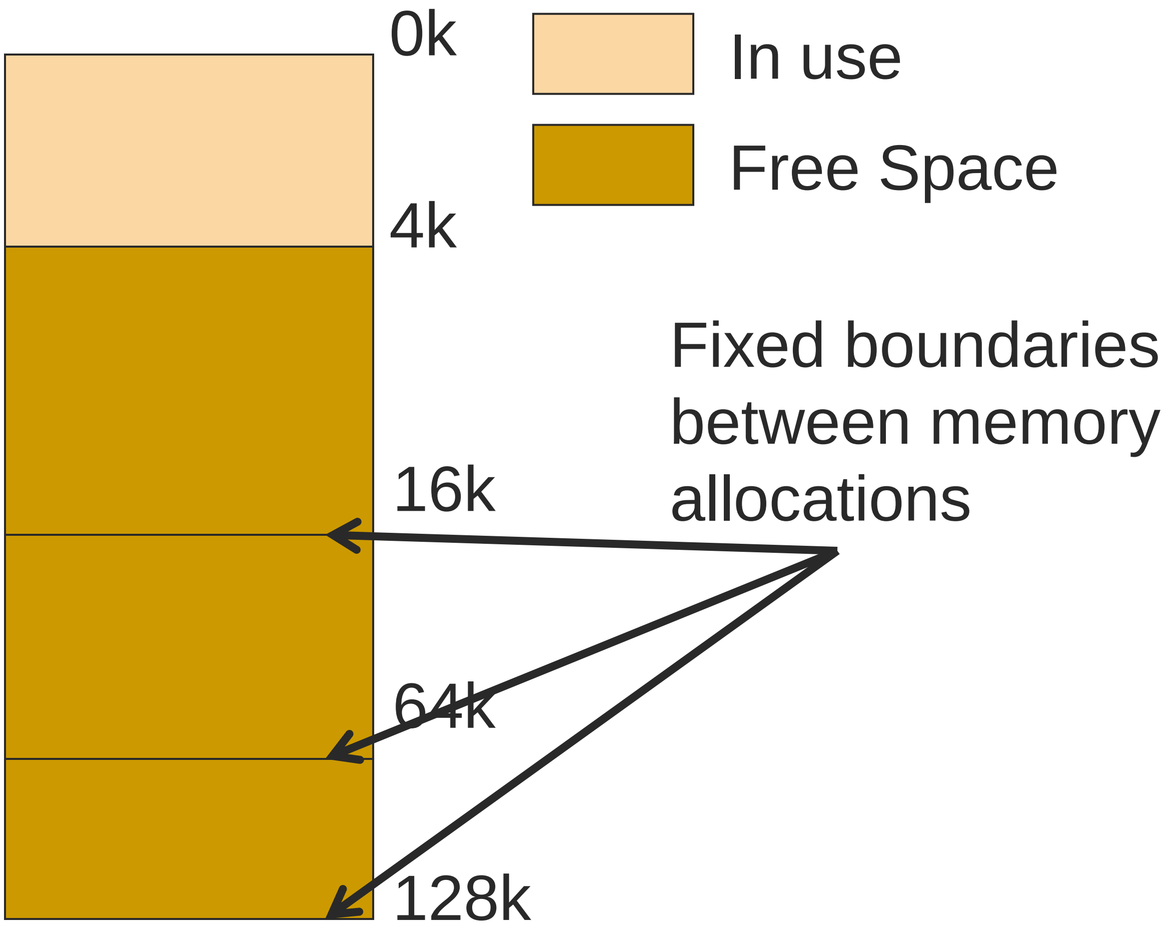
Multi-Programming

- Multiple processes in memory at the same time
- Goals
  1. Layout processes in memory as needed
  2. Protect each process's memory from accesses by other processes
  3. Minimize performance overheads
  4. Maximize memory utilization

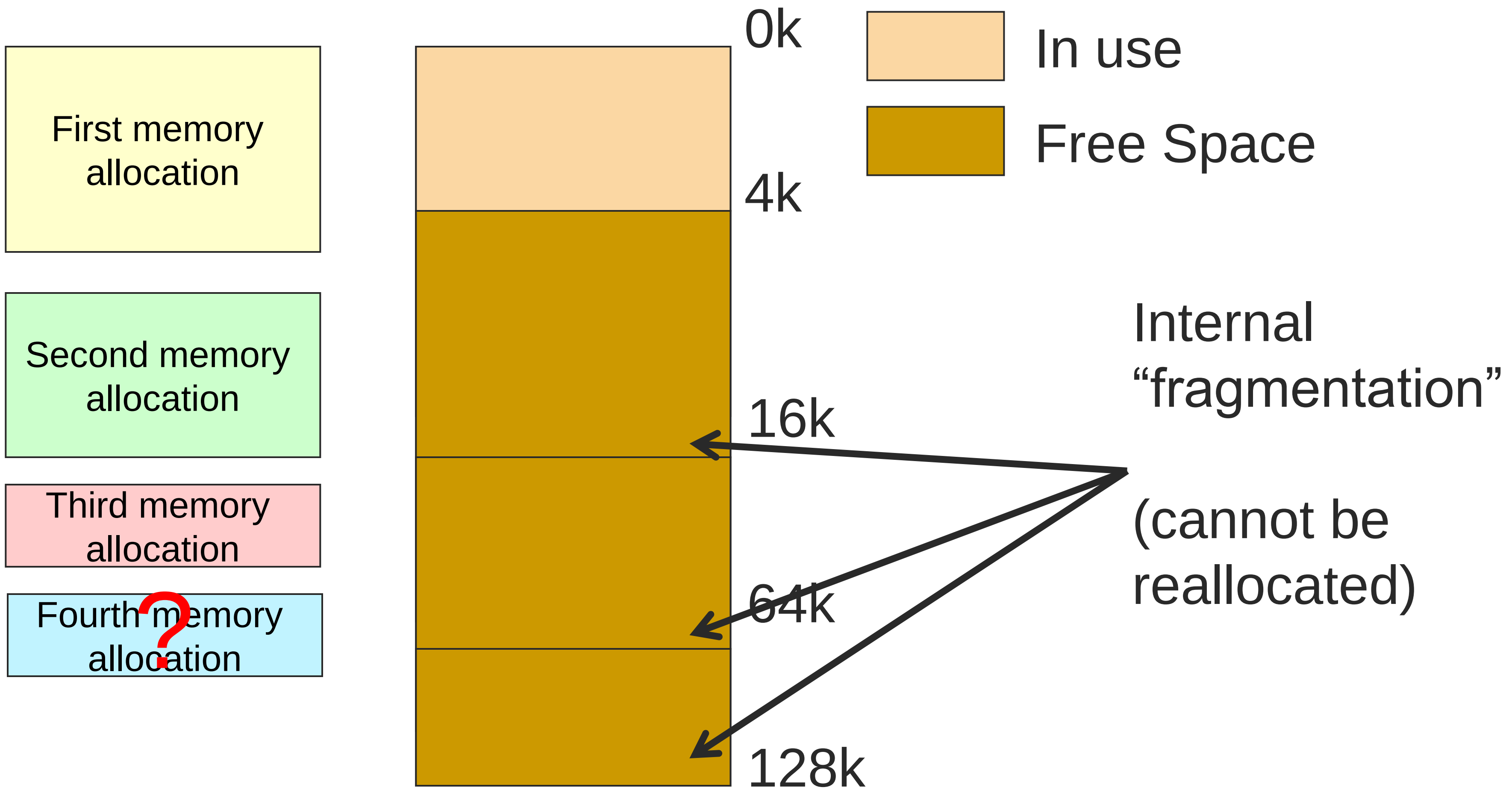
Copyright ©: University of Illinois CS 241 Staff

Multiple Fixed Partitions

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



Multiple Fixed Partitions



Problems with Fixed Partitions

1. Program addresses vary across runs
2. Internal fragmentation
3. Not all processes may fit in memory

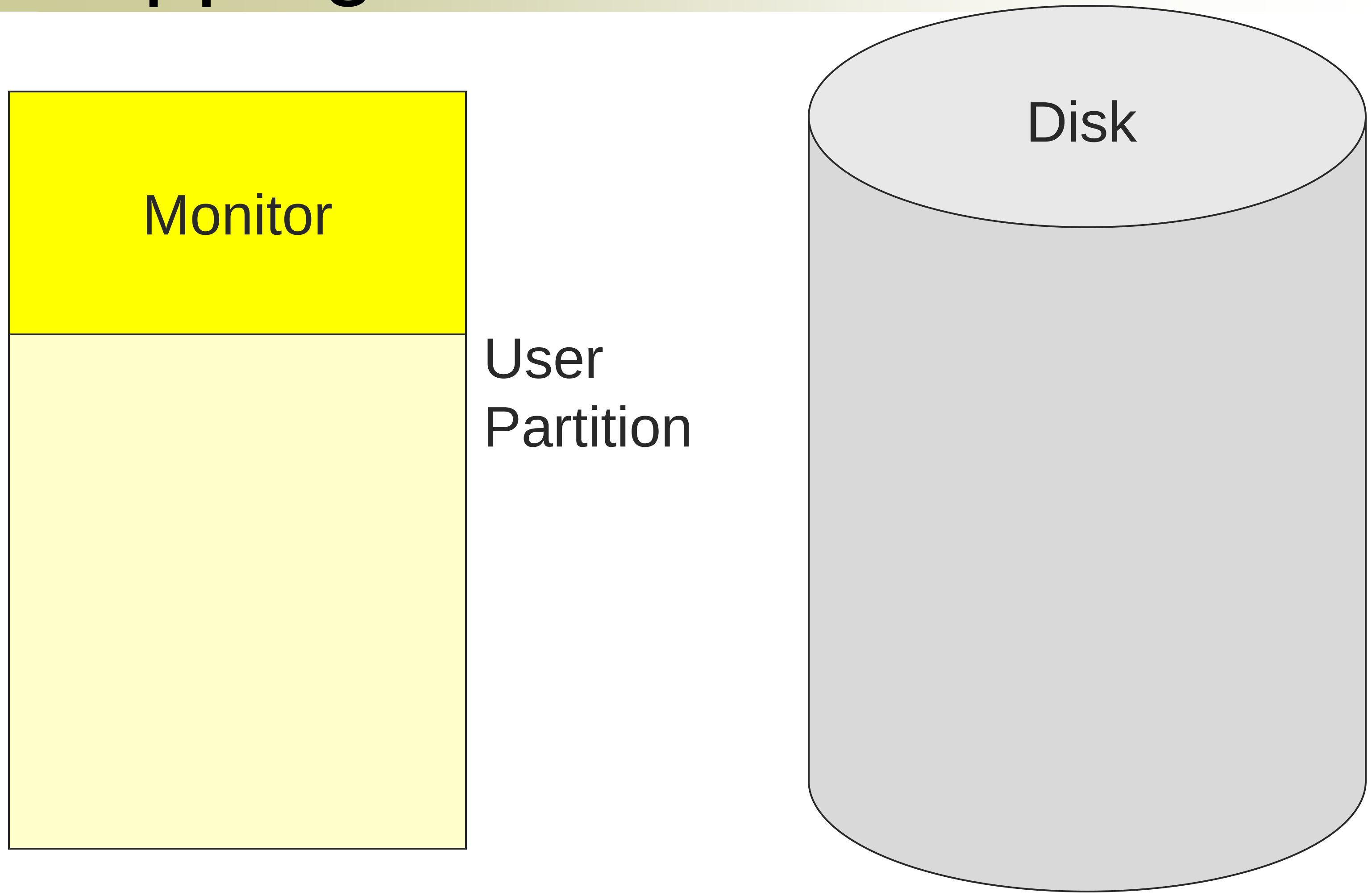
Copyright ©: University of Illinois CS 241 Staff

Problem 1: Insufficient Memory

- What if there are more processes than could fit into the memory?
- Swapping
- Impact: Memory allocation changes as
  - Processes come into memory
  - Processes leave memory
    - Swapped to disk
    - Complete execution

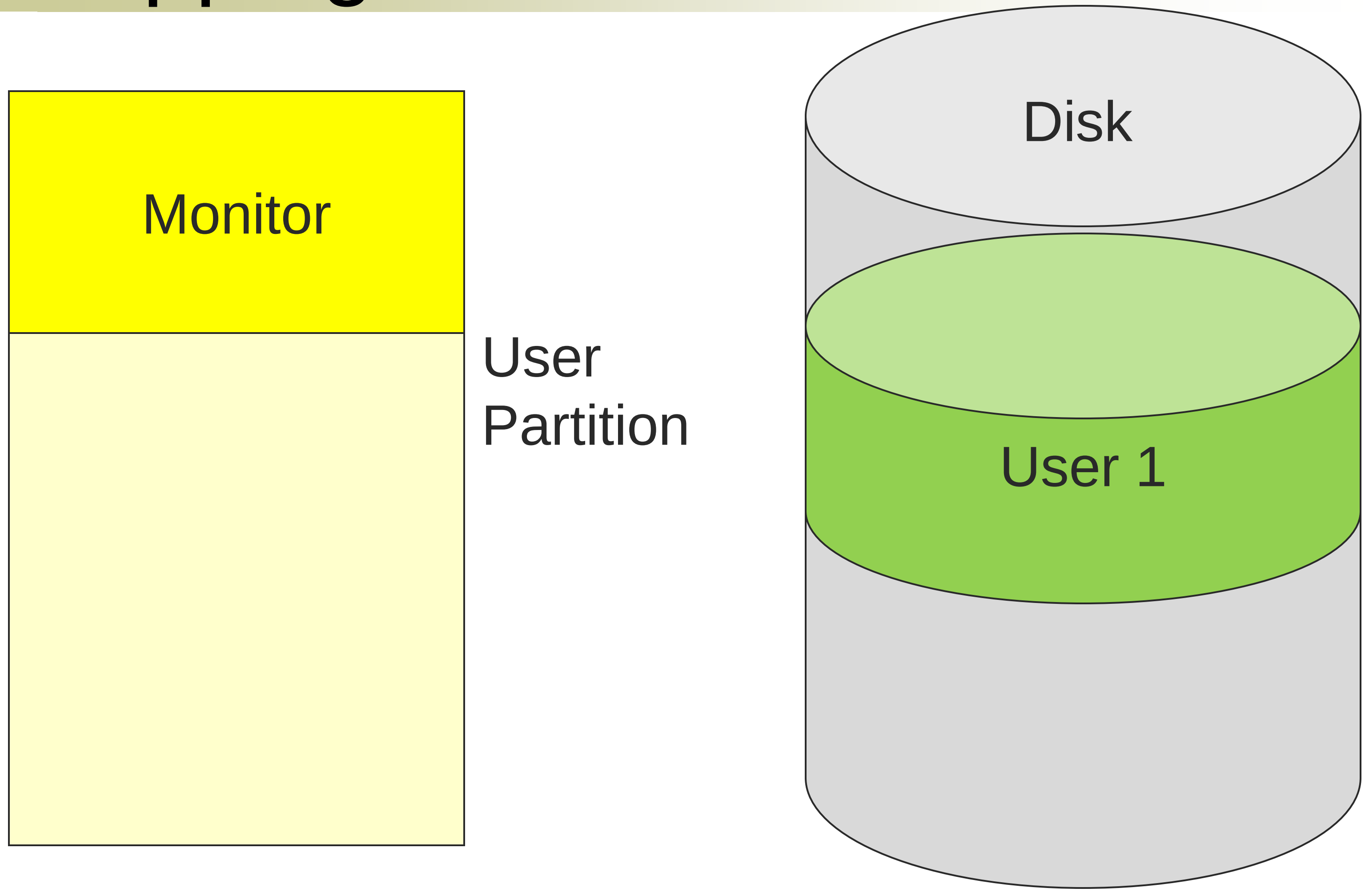
Copyright ©: University of Illinois CS 241 Staff

Swapping

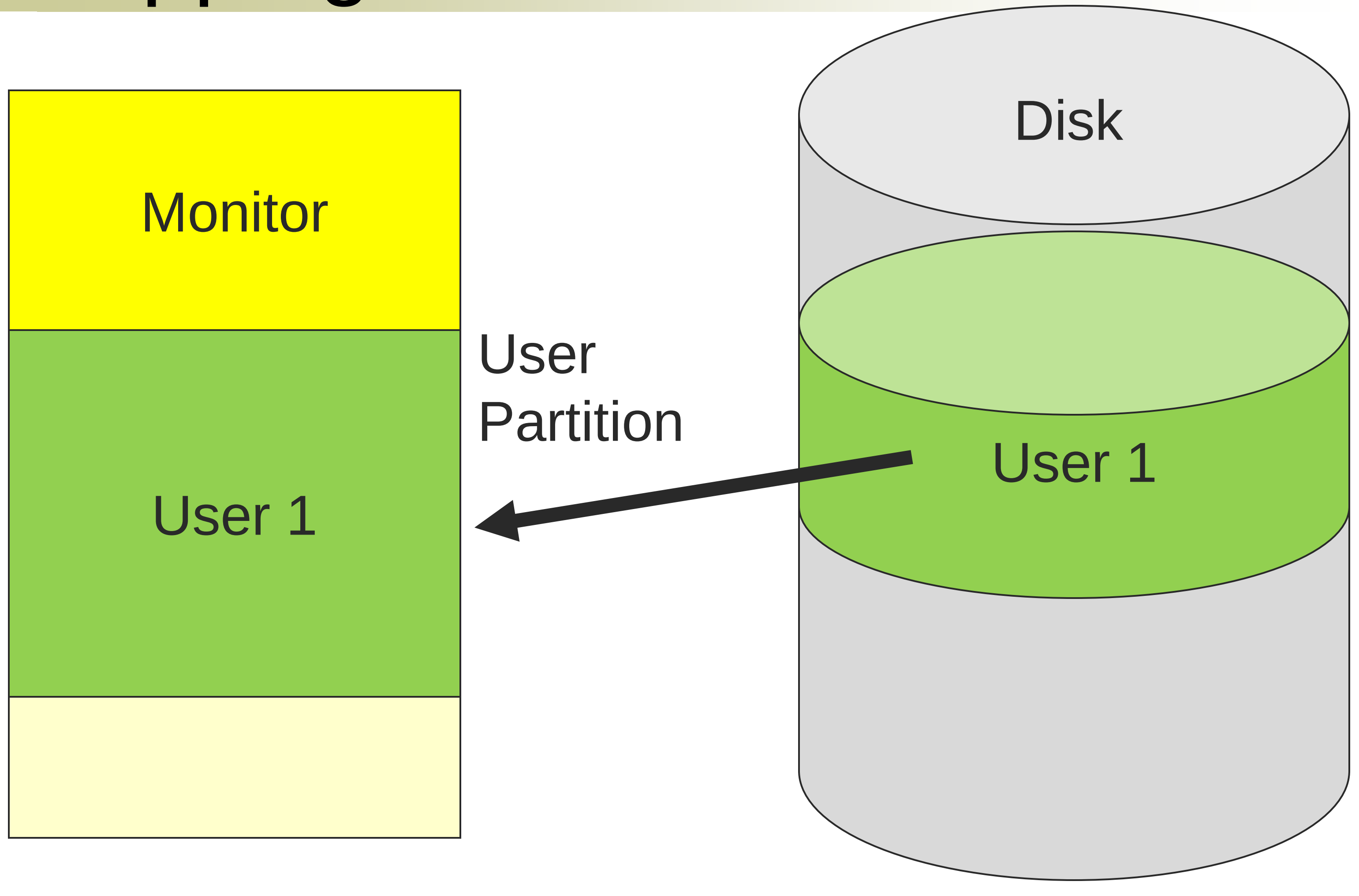


Copyright ©: University of Illinois CS 241 Staff

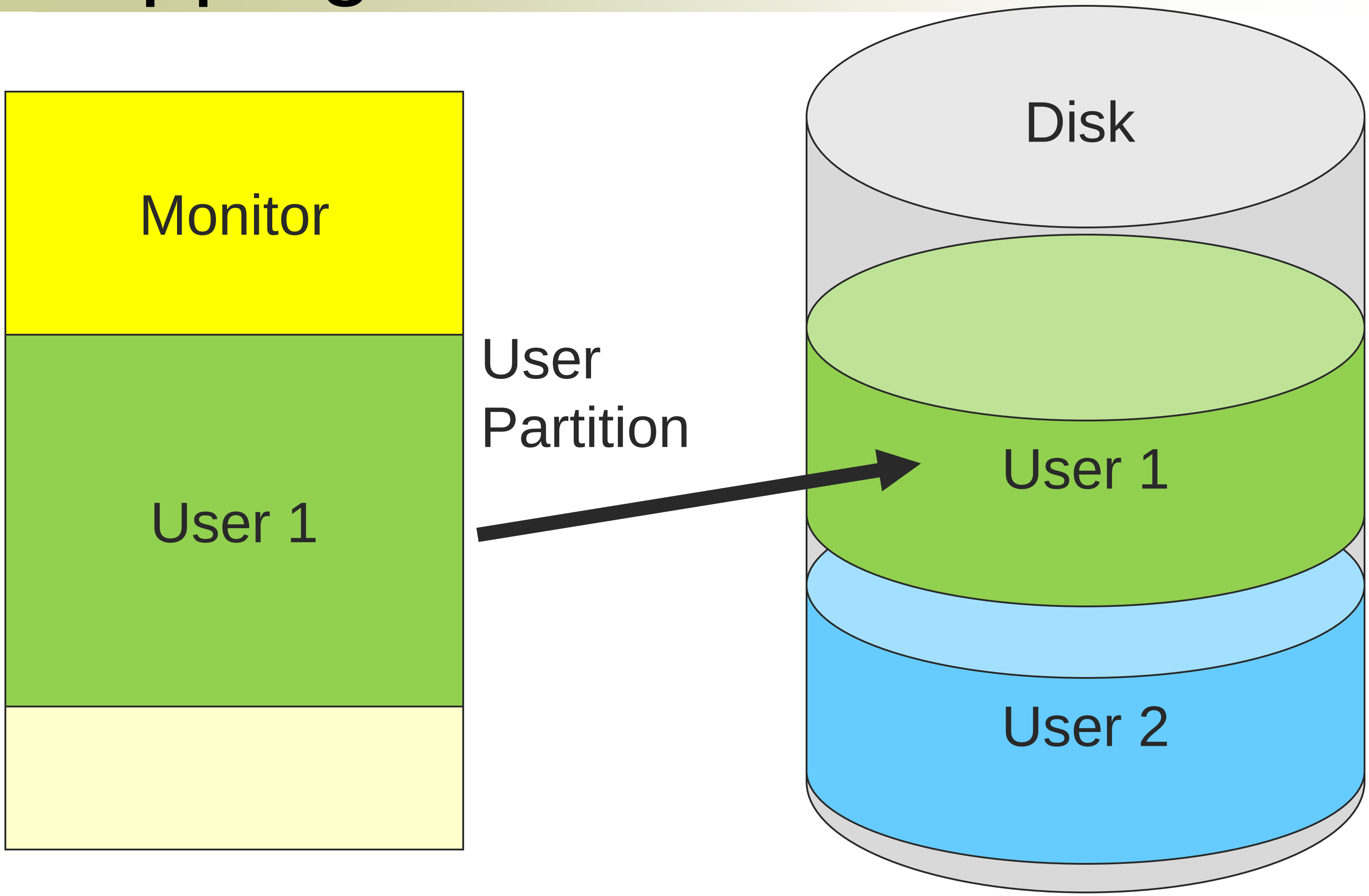
Swapping



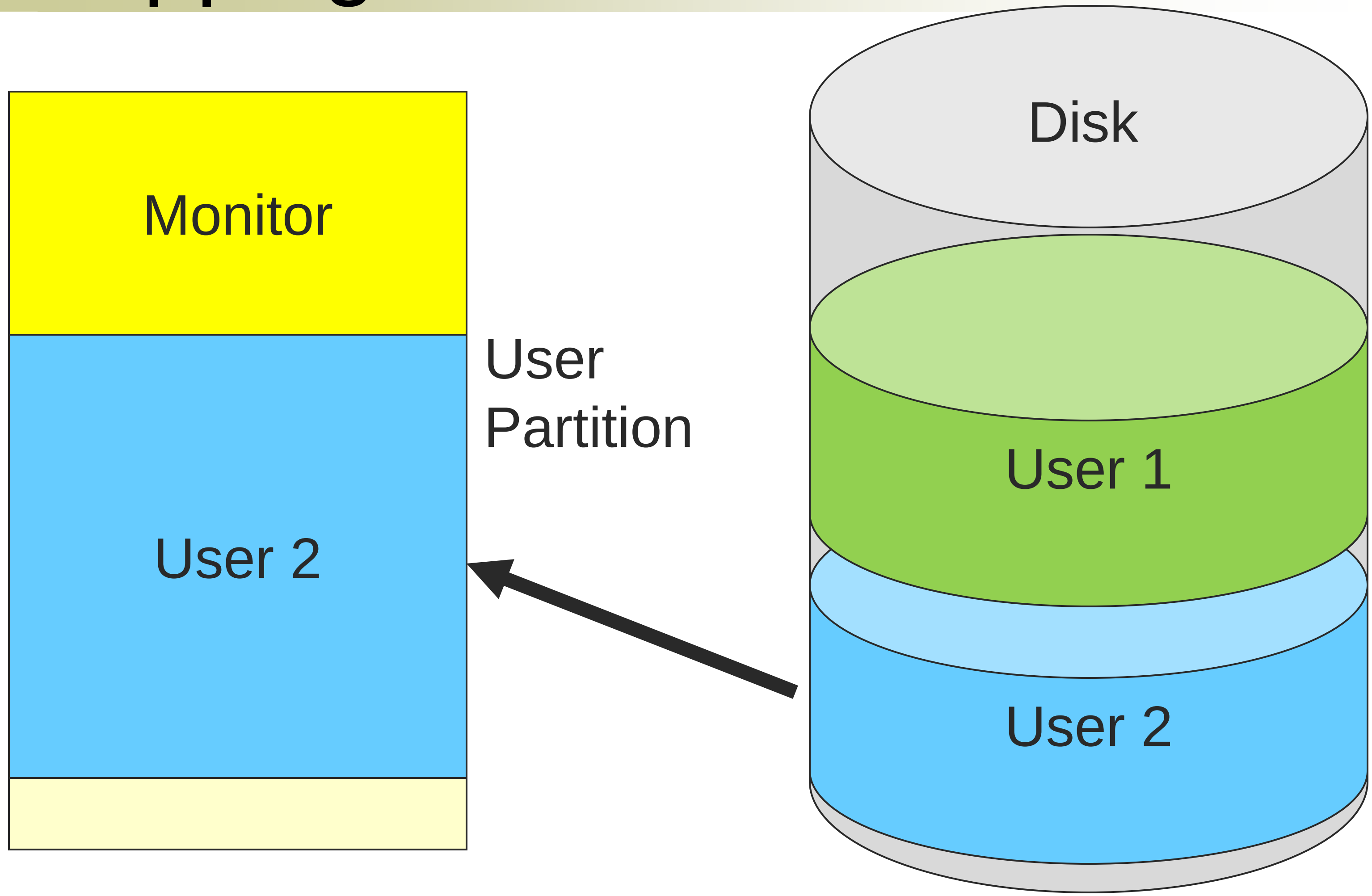
Swapping



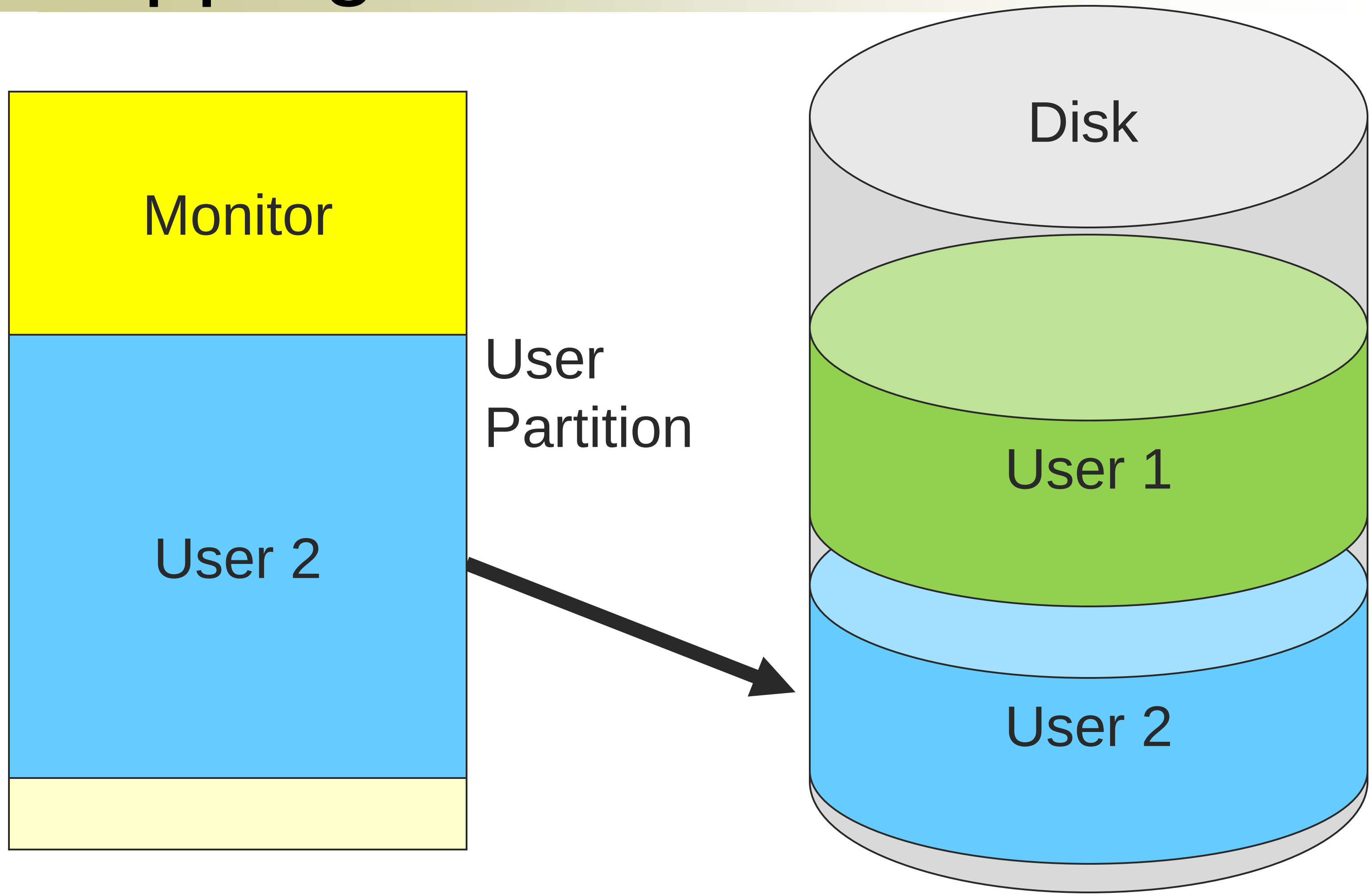
Swapping



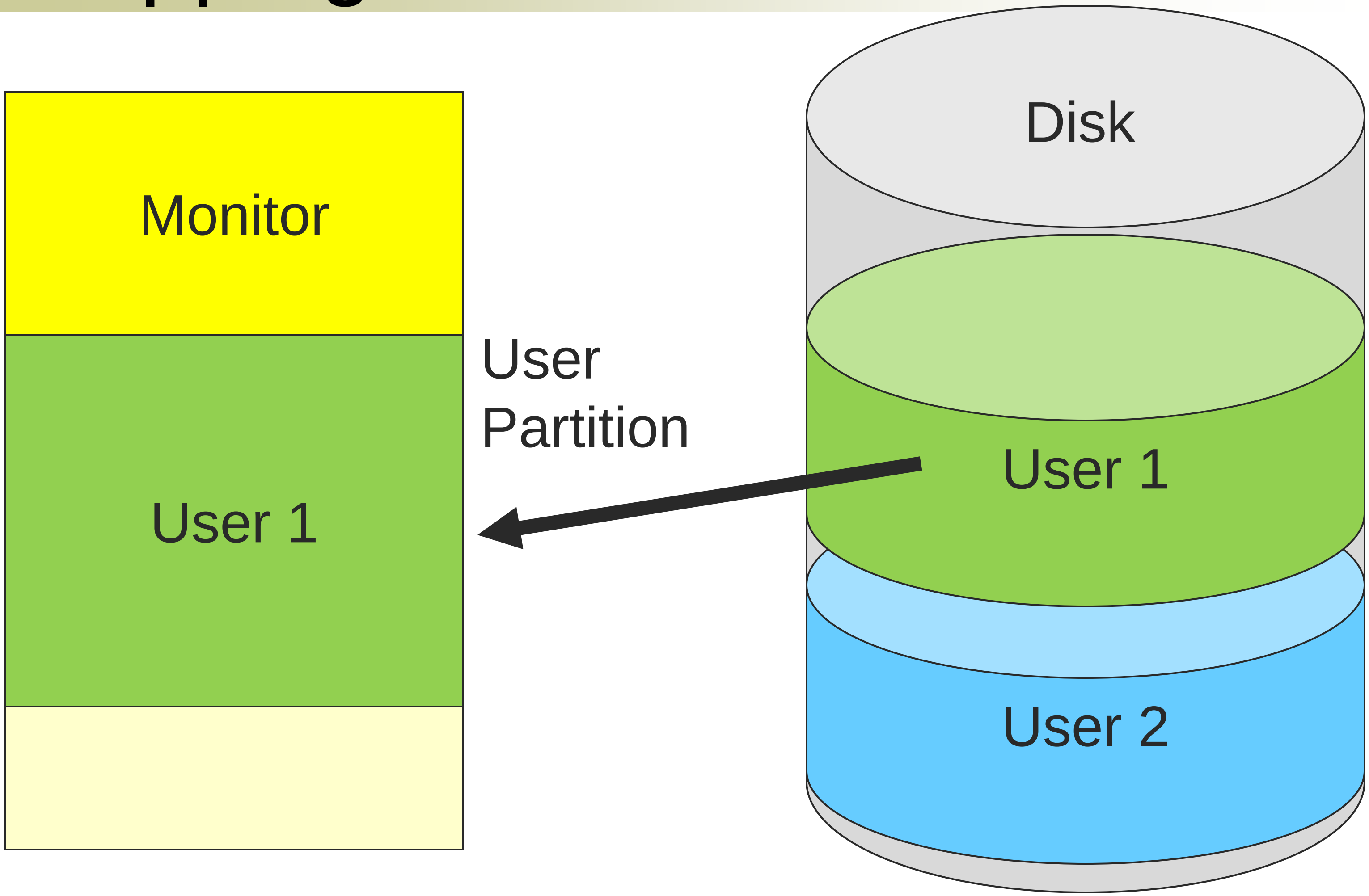
Swapping



Swapping



Swapping



Storage Placement Strategies

- First fit
  - Use the first available hole whose size is sufficient to meet the need
  - Rationale?
- 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
  - Rationale?
- Worst fit
  - Use the largest available hole
  - Rationale?

Example

- Consider a swapping 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

Copyright ©: University of Illinois CS 241 Staff

Example

- Consider a swapping 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: 20K, 10K, 18K.	Best fit: 12K, 10K, 9K.	Worst fit: 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

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

How Bad Is Fragmentation?

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

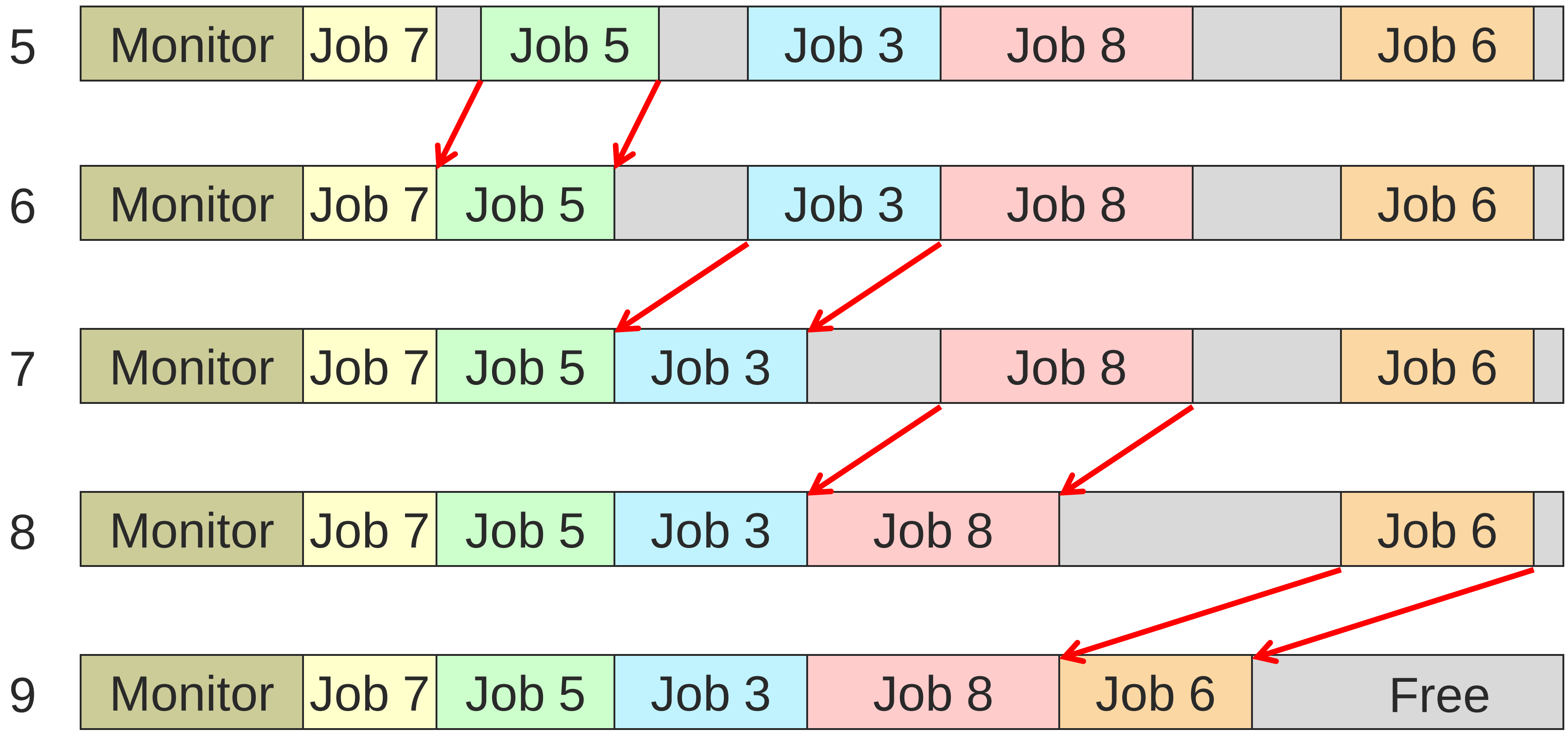
- Known as 50% RULE

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 relocation is dynamic, and is done at execution time

Copyright ©: University of Illinois CS 241 Staff

Compaction



Copyright ©: University of Illinois CS 241 Staff

Storage Management Problems

- Fixed partitions suffer from
- Variable partitions suffer from
- Compaction suffers from

Copyright ©: University of Illinois CS 241 Staff

Storage Management Problems

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

Copyright ©: University of Illinois CS 241 Staff

Limitations of Swapping

- Problems with swapping under Partitioning
  - Process must fit into physical memory (impossible to run larger processes)
  - Memory becomes fragmented
    - External fragmentation
      - Lots of small free areas
    - Need compaction
      - Reassemble larger free areas
  - Processes are either in memory or on disk
    - Half and half doesn't do any good

Copyright ©: University of Illinois CS 241 Staff

Problem 2: Varying Addresses

- Problem addresses for a job are unknown until start time
  - At link-time, linker must know memory address at which the program will begin
  - These addresses must be adjusted at run time

Solution?

Virtual Memory

- Basic idea
  - Allow the OS to hand out more memory than exists on the system
  - Keep recently used stuff in physical memory
  - Move less recently used stuff to disk
  - Keep all of this hidden from processes
- Process view
  - Processes still see an address space from 0 – max address
  - Actual physical location (and movement) of memory handled by the OS without process help

Copyright ©: University of Illinois CS 241 Staff

Virtual Addresses

- Virtual address
  - An address meaningful to the user process
- Physical address
  - An address meaningful to the physical memory
- Different jobs run at different phy. addresses
  - But virtual address can be the same
  - Program never sees physical address
  - Linker must know program's starting memory address

Copyright ©: University of Illinois CS 241 Staff

Indirection

"Any programming problem can be solved by adding a level of indirection …

…except for the problem of too many layers of indirection."

David Wheeler

Copyright ©: University of Illinois CS 241 Staff

Multi-programming

- Multiple processes in memory at the same time
- What do we really need?
  - Address translation
    - Translate every memory reference from virtual address to physical address
    - Static before execution, or dynamic during execution?
  - Protection
    - Support independent addresses spaces

Dynamic Address Translation

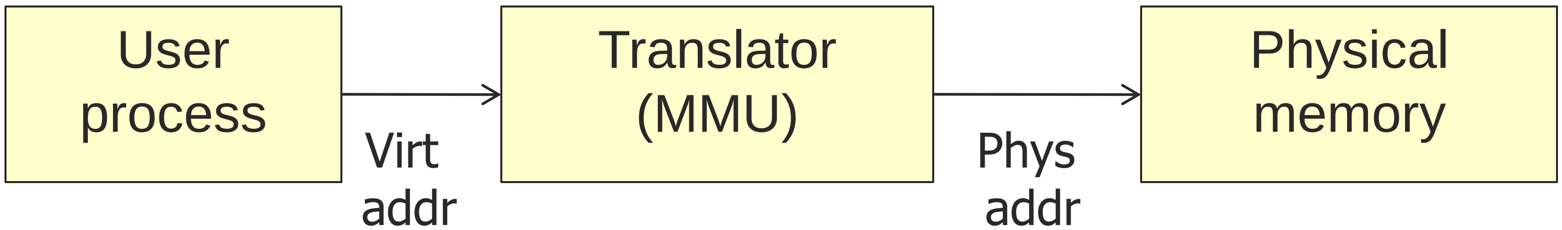
- Load each process into contiguous regions of physical memory

- Logical or "Virtual" addresses
  - Logical address space
  - Range: 0 to MAX

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

Copyright ©: University of Illinois CS 241 Staff

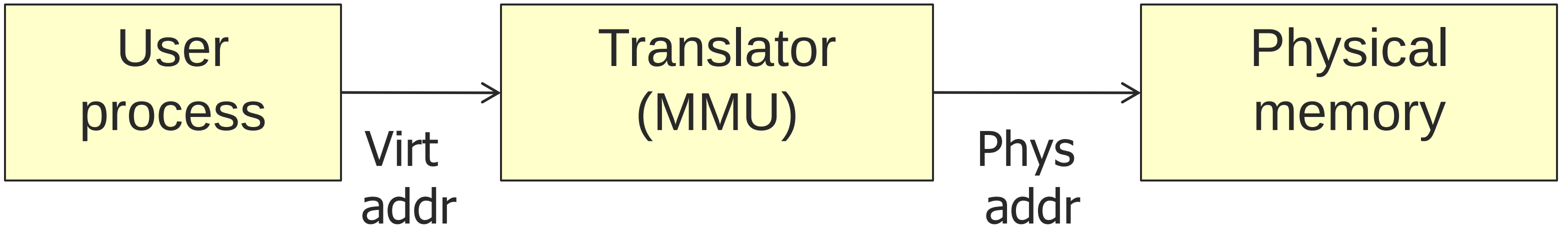
Dynamic Address Translation



- Translation enforces protection
  - One process can't even refer to another process's address space
- Translation enables virtual memory
  - A virtual address only needs to be in physical memory when it is being accessed
  - Change translations on the fly as different virtual addresses occupy physical memory

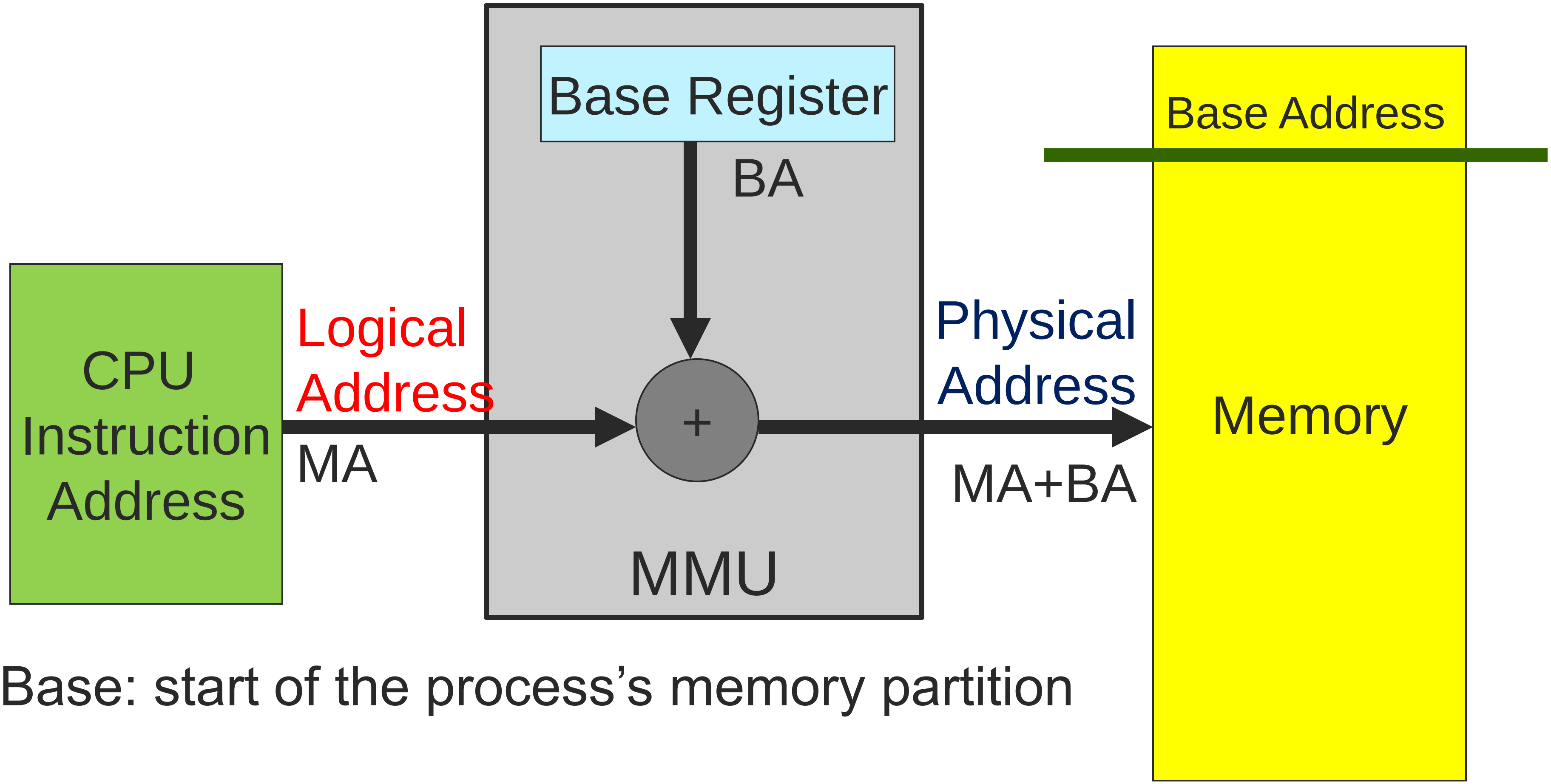
Copyright ©: University of Illinois CS 241 Staff

Dynamic Address Translation



- Implementation tradeoffs
  - Flexibility (e.g., sharing, growth, virtual memory)
  - Size of translation data
  - Speed of translation

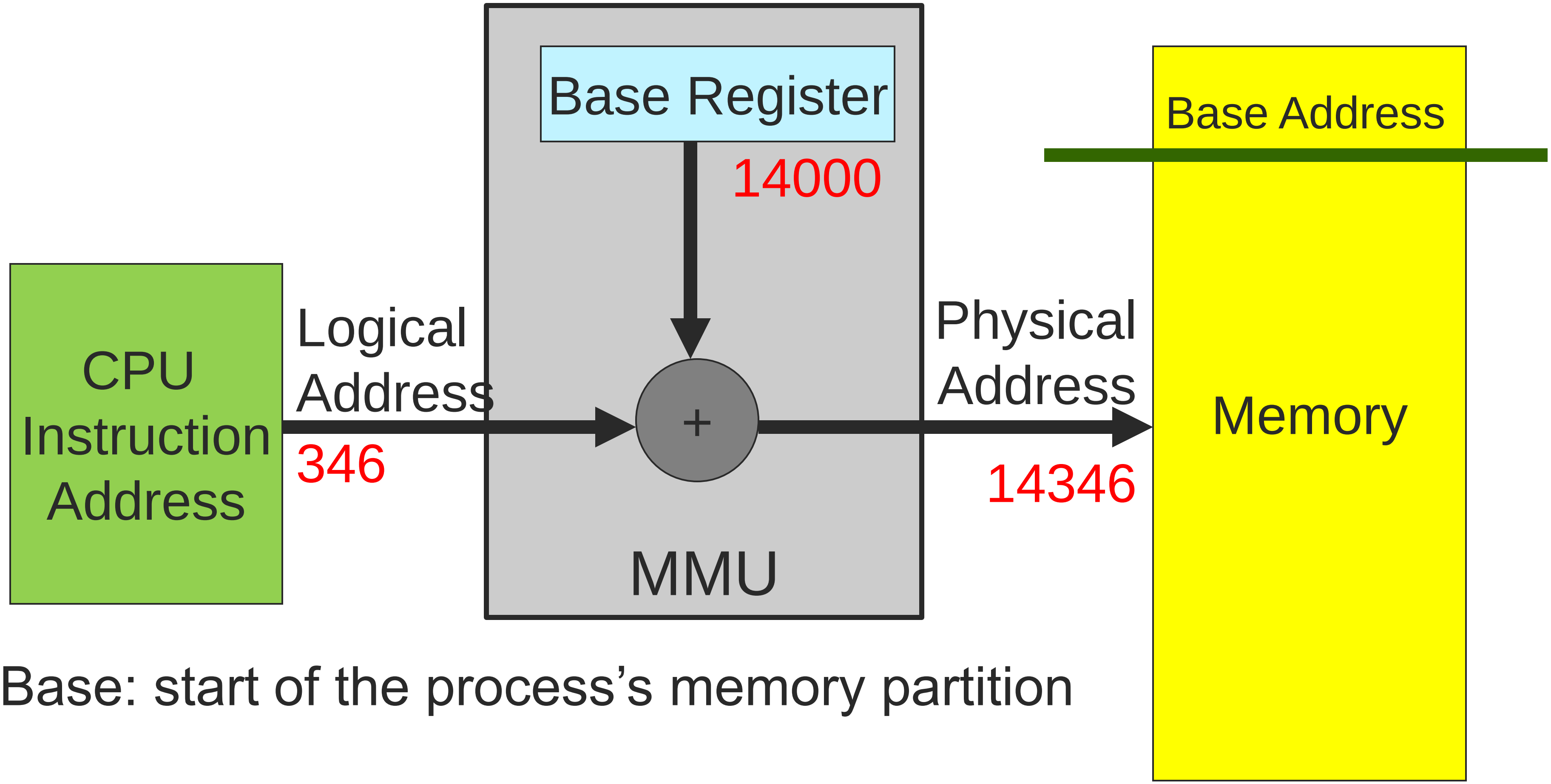
Base Register



Base: start of the process's memory partition

Copyright ©: University of Illinois CS 241 Staff

Base Register



Base: start of the process's memory partition

Copyright ©: University of Illinois CS 241 Staff

Protection

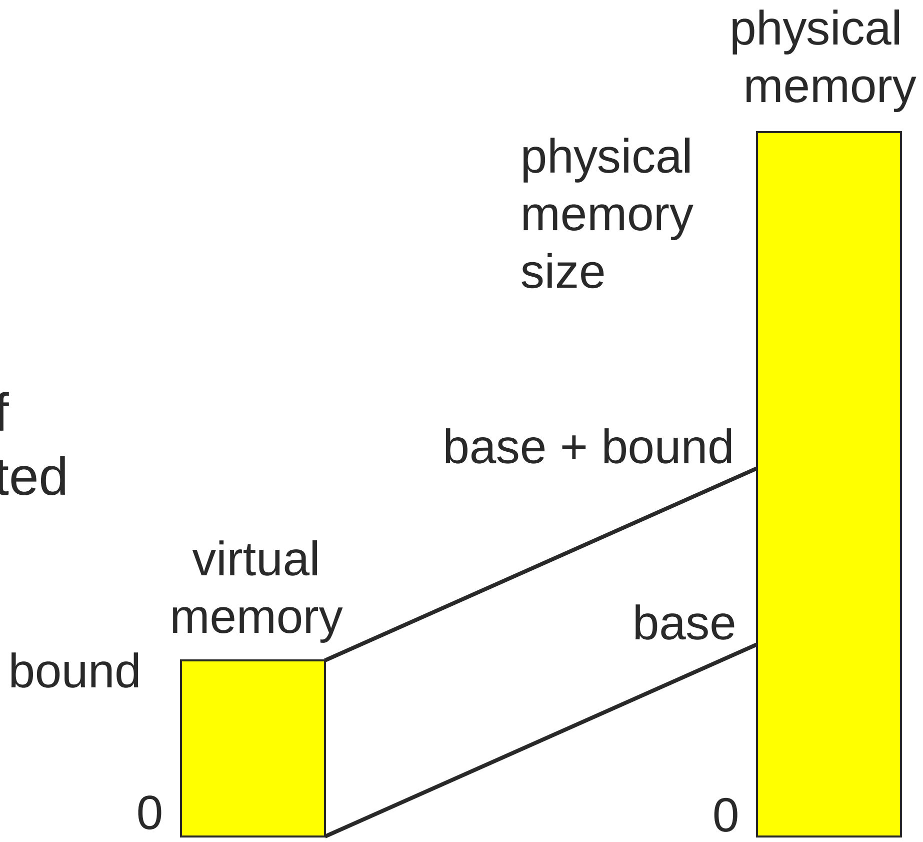
- Problem
  - How to prevent a malicious process from writing or jumping into another user's or OS partitions
- Solution
  - Base bounds register

Copyright ©: University of Illinois CS 241 Staff

Base and bounds

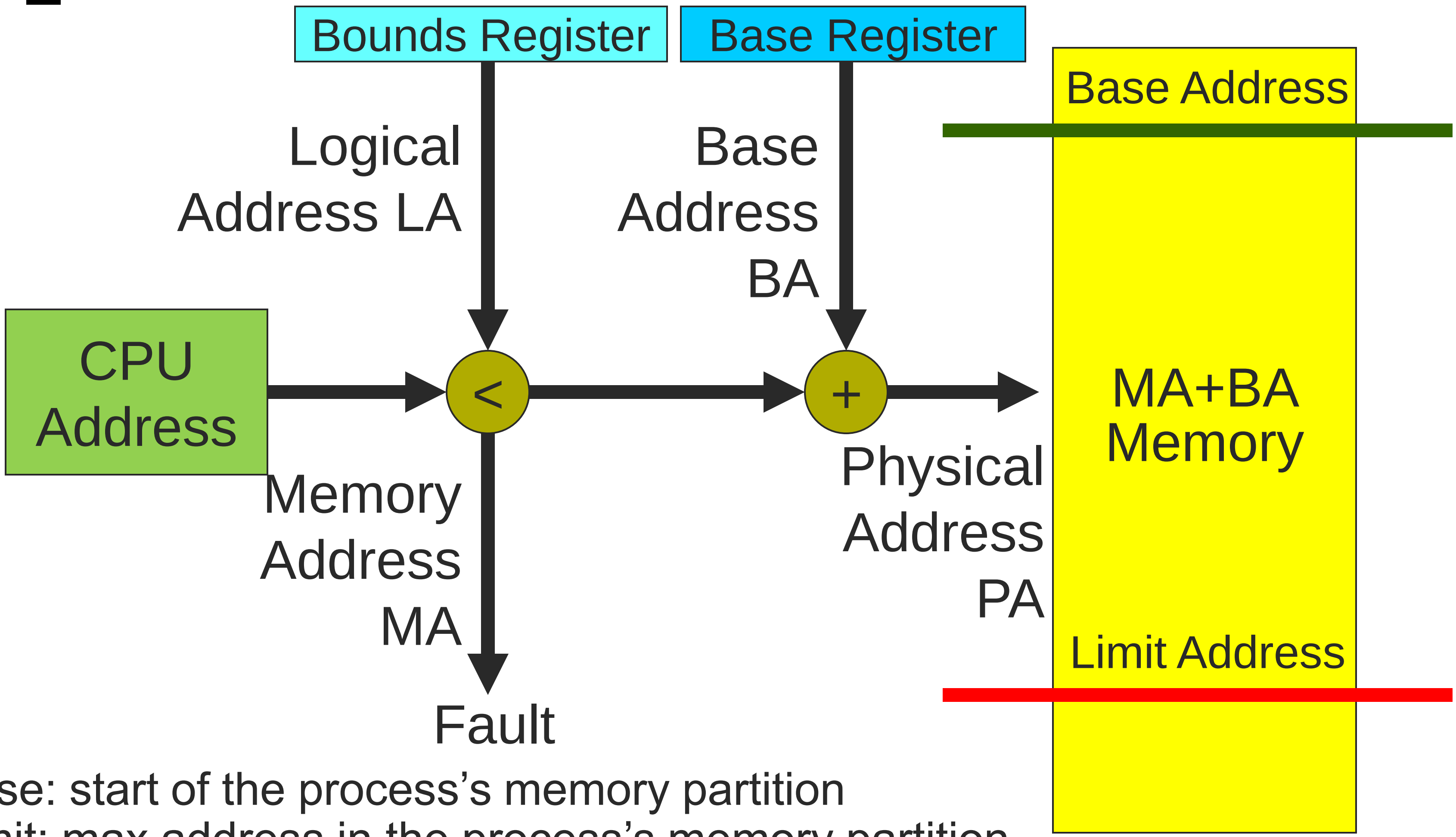
```
if (virt addr > bound)
    trap to kernel
} else {
    phys addr =
      virt addr + base
}
```

- Process has the illusion of running on its own dedicated machine with memory [0,bound)
- Provides protection from other processes also currently in memory



Copyright ©: University of Illinois CS 241 Staff

Base and Bounds Registers



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

Copyright ©: University of Illinois CS 241 Staff

Base and bounds

- What must change during a context switch?
  - The base and the bounds registers
- Can a process change its own base and bound?
  - No, only the OS can change these registers
  - The program can do it indirectly (e.g., ask for more memory in stack)

Base and bounds

- Problem: Process needs more memory over time
- How does the kernel handle the address space growing?
  - You are the OS designer
  - Design algorithm for allowing processes to grow