Memory

CS 241

January 30, 2012

Announcements

- MP1 due Tuesday 11:59 pm via svn
- MP2 out Wednesday
- Research opportunity
  - With Nitin Vaidya
  - Related to networks / distributed systems
  - See Piazza for details

Address Spaces and Memory

- Process
  - One or more thread
  - One address space
- Thread
  - Stream of execution
  - Unit of concurrency
- Address space
  - Memory space that threads use
  - Unit of data

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?

Address Space Illusions

- Address independence
- Protection
- Virtual memory

Address Space Illusions

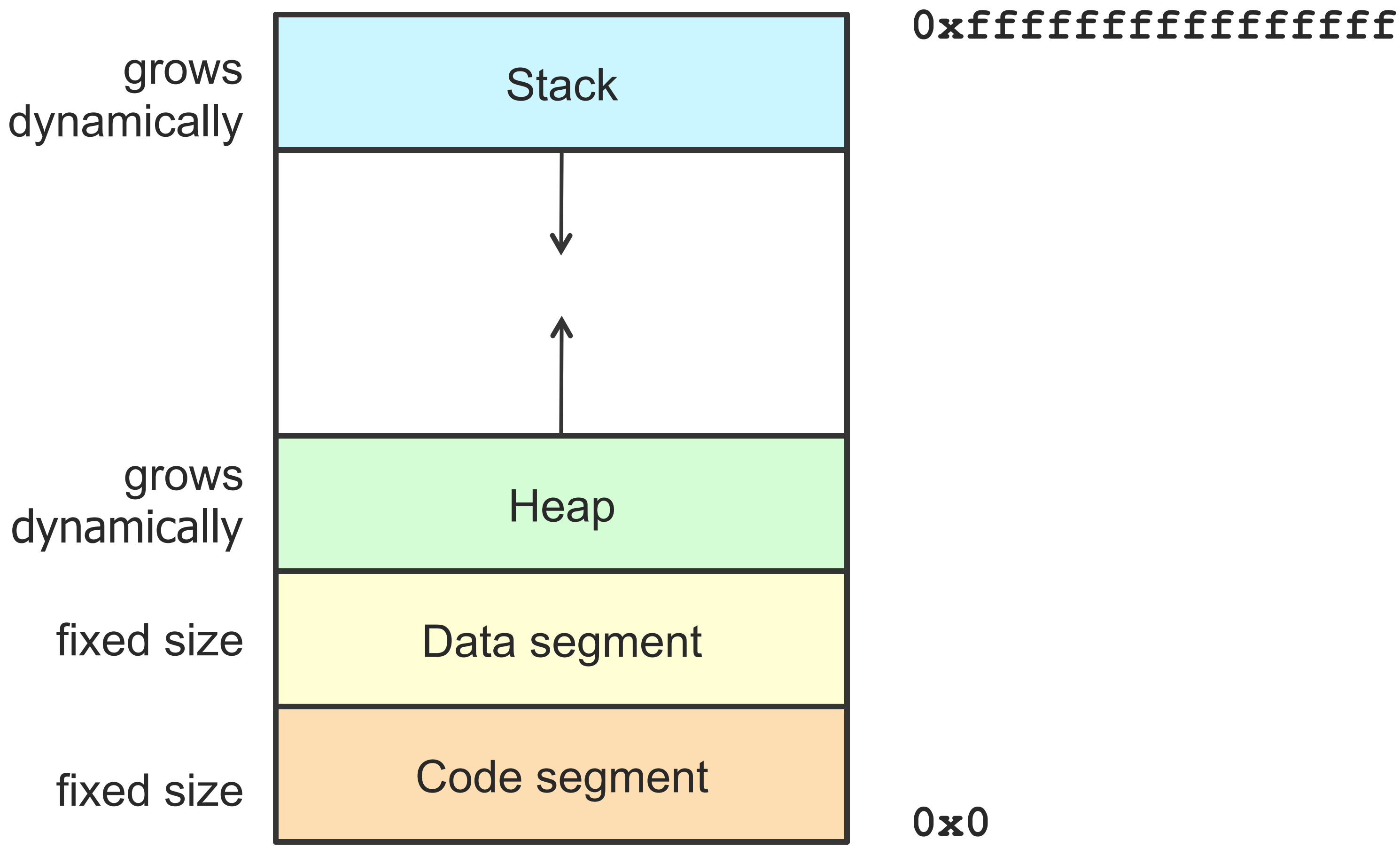
- Address independence
  - Same address can be used in different address spaces yet remain logically distinct
- Protection
  - One address space cannot access data in another address space
- Virtual memory
  - Address space can be larger than the amount of physical memory on the machine

Address Space Illusions

Illusion	**Reality**
Giant address space Protected from others (Unless you want to share) More whenever you want it	Many processes sharing One address space Limited memory

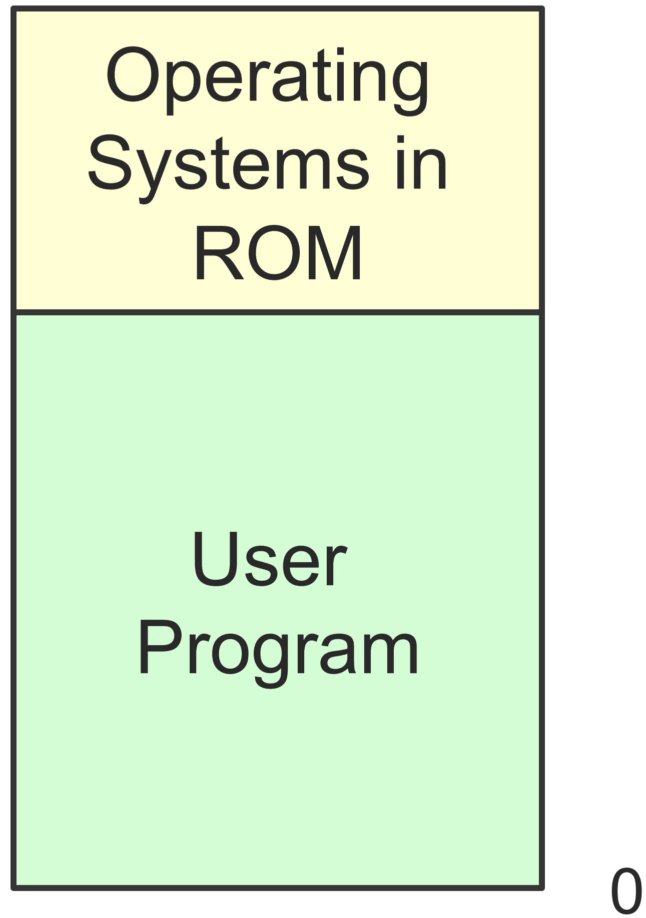
Today:
The story of the Illusion

Address Space



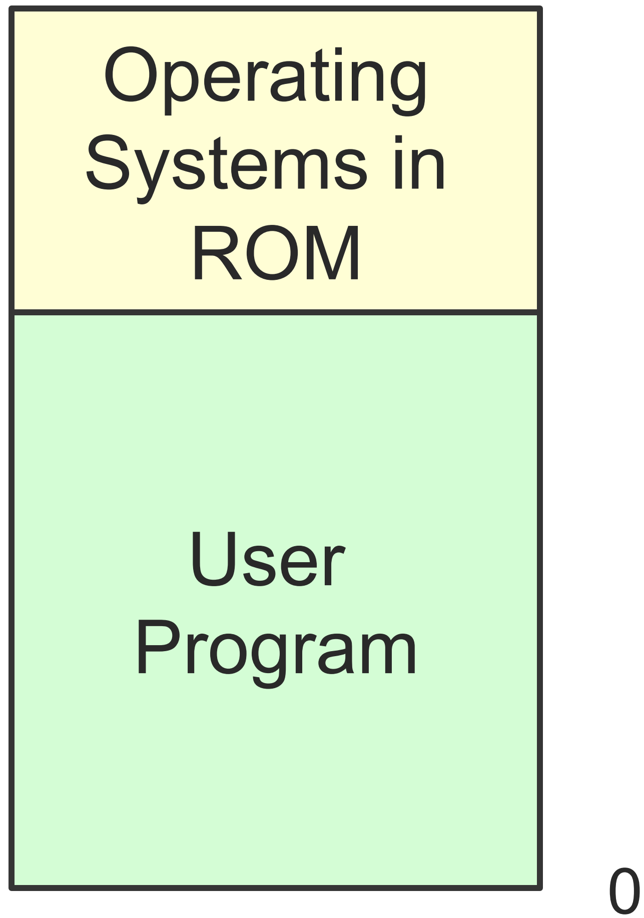
Uni-programming

- 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?



Uni-programming

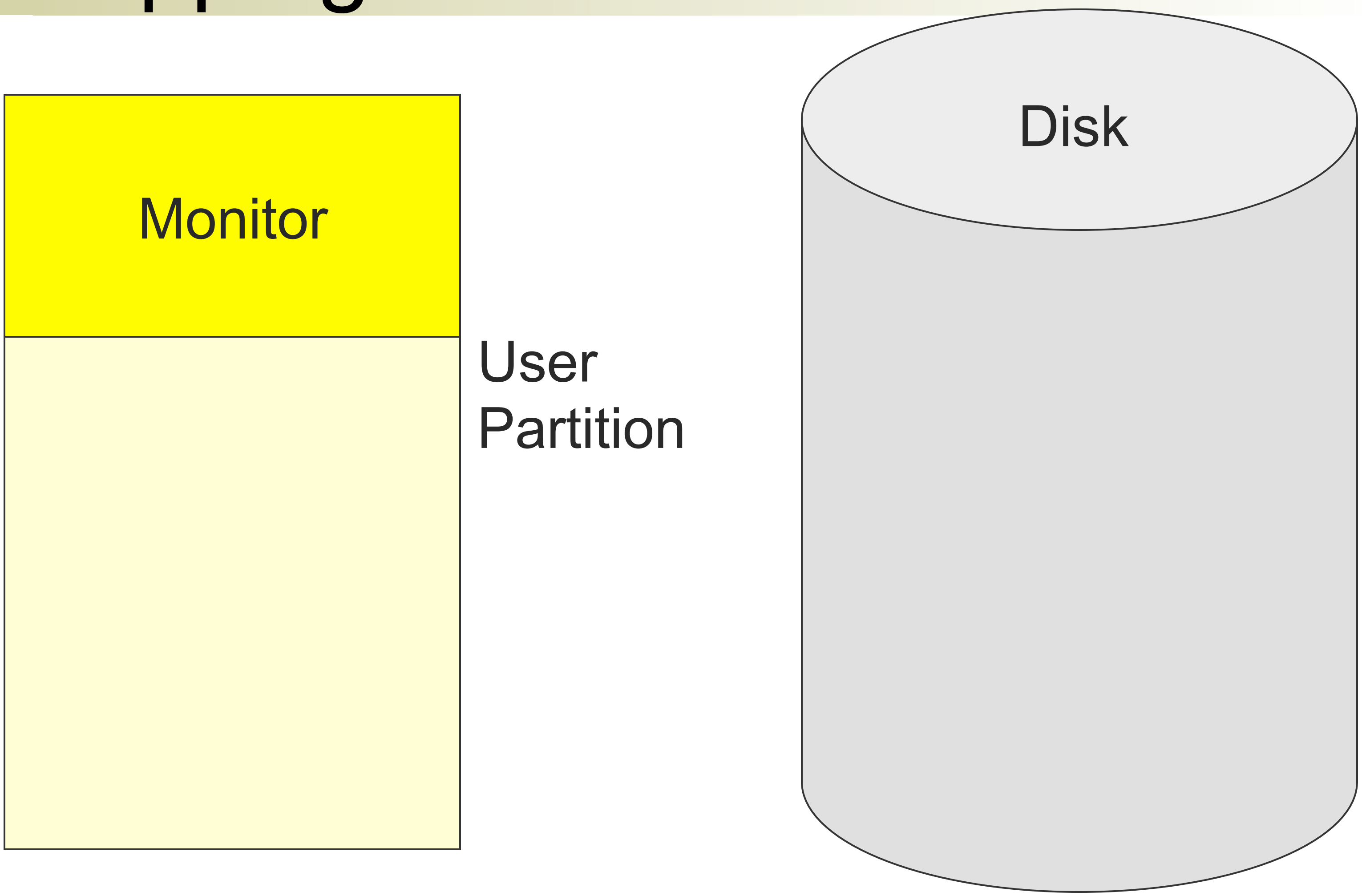
- 1 process runs at a time
- Always load process into the same spot
- How do you switch processes?
- What illusions does this provide?
  - Independence, protection, virtual memory
- Problems?
  - Slow, large time slices



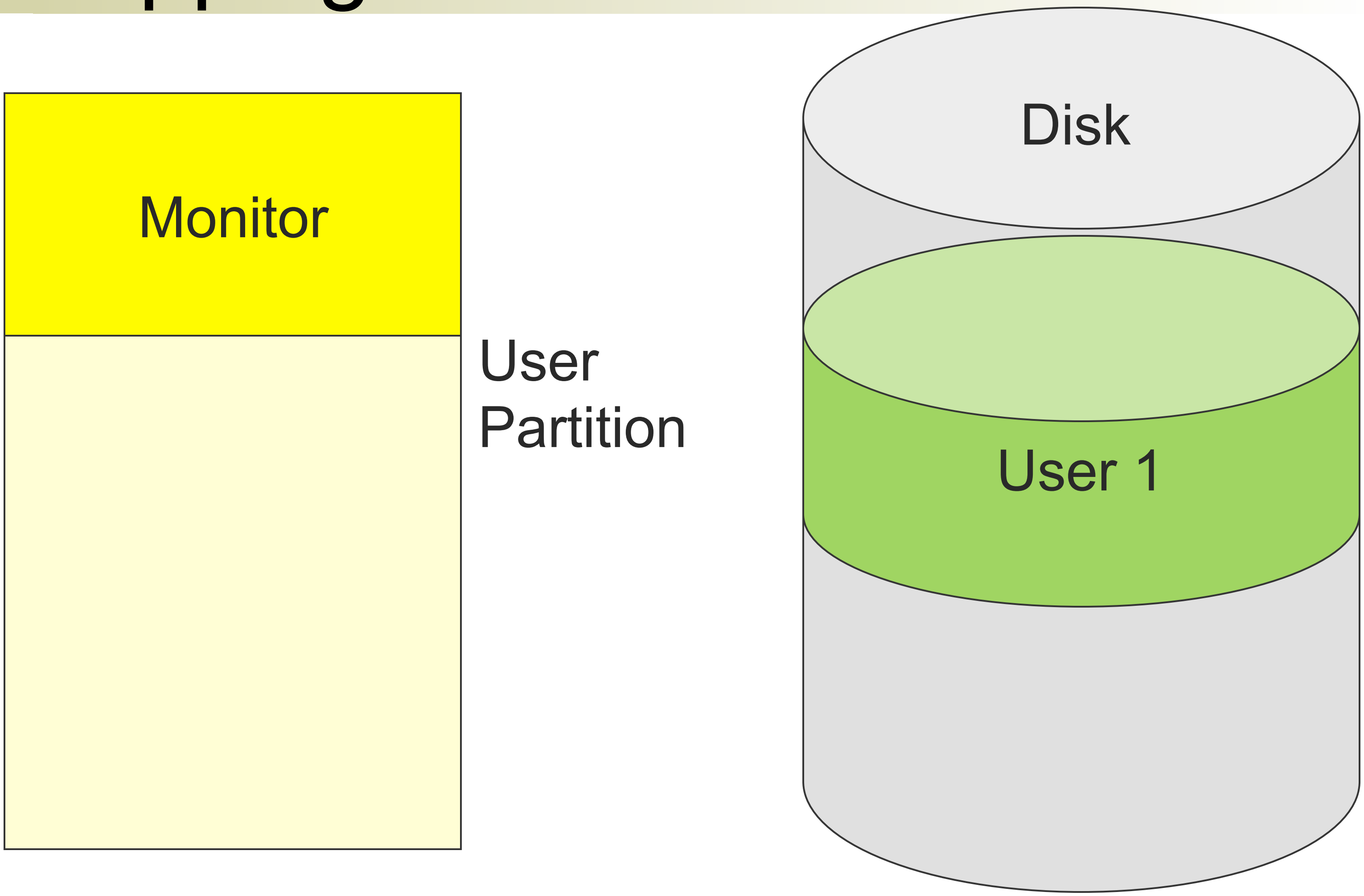
Multi-Programming

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

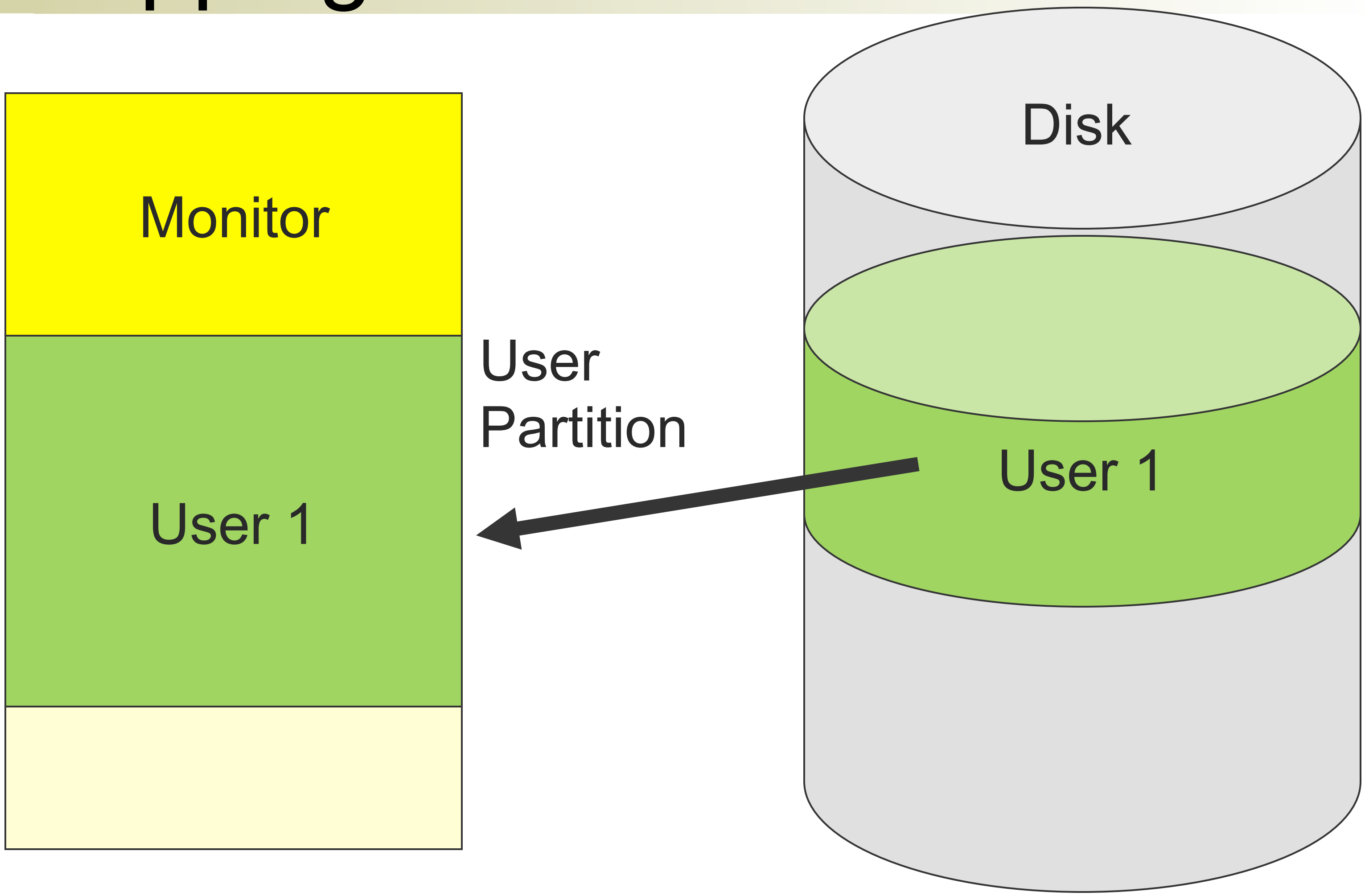
Swapping



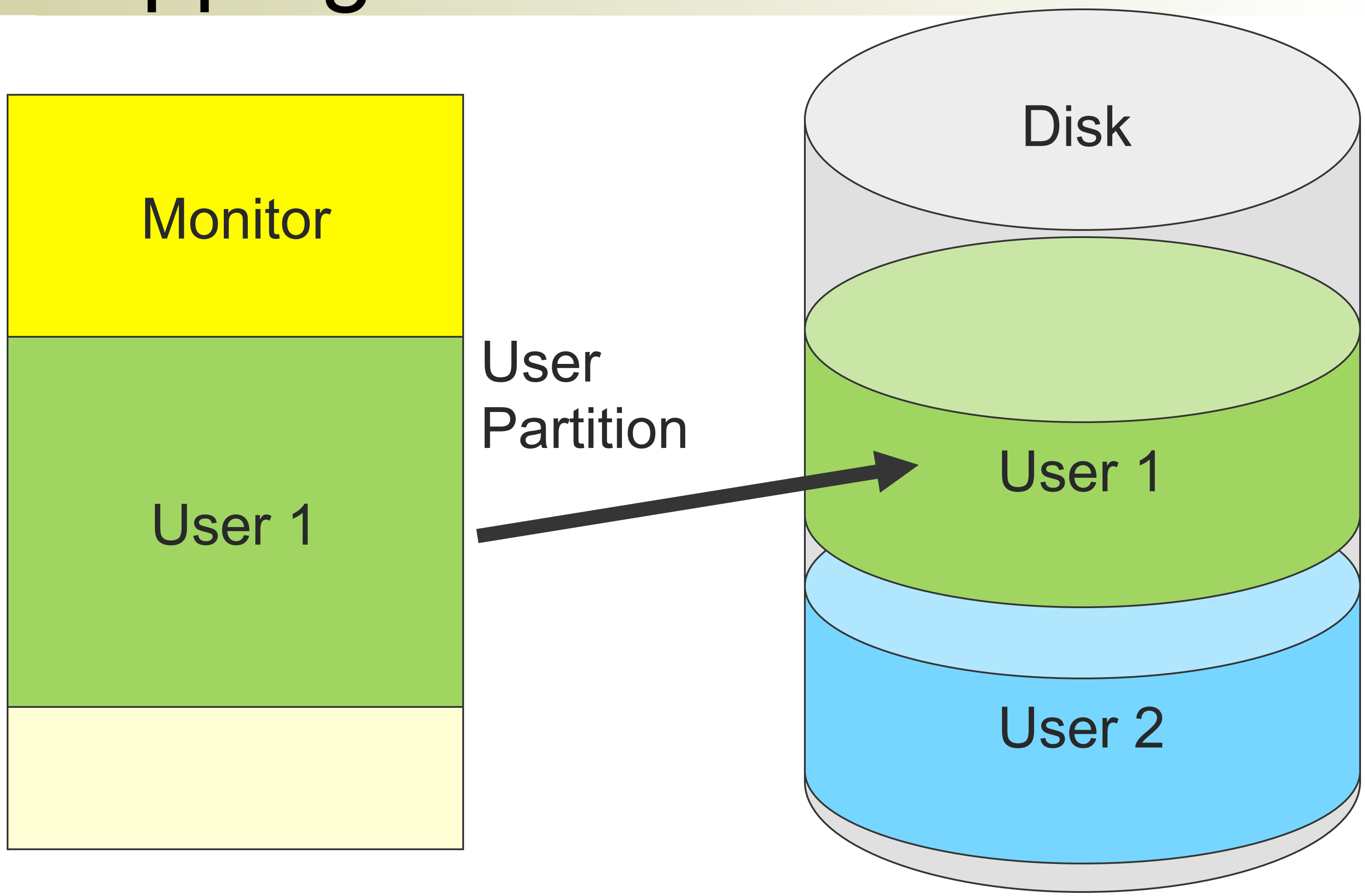
Swapping



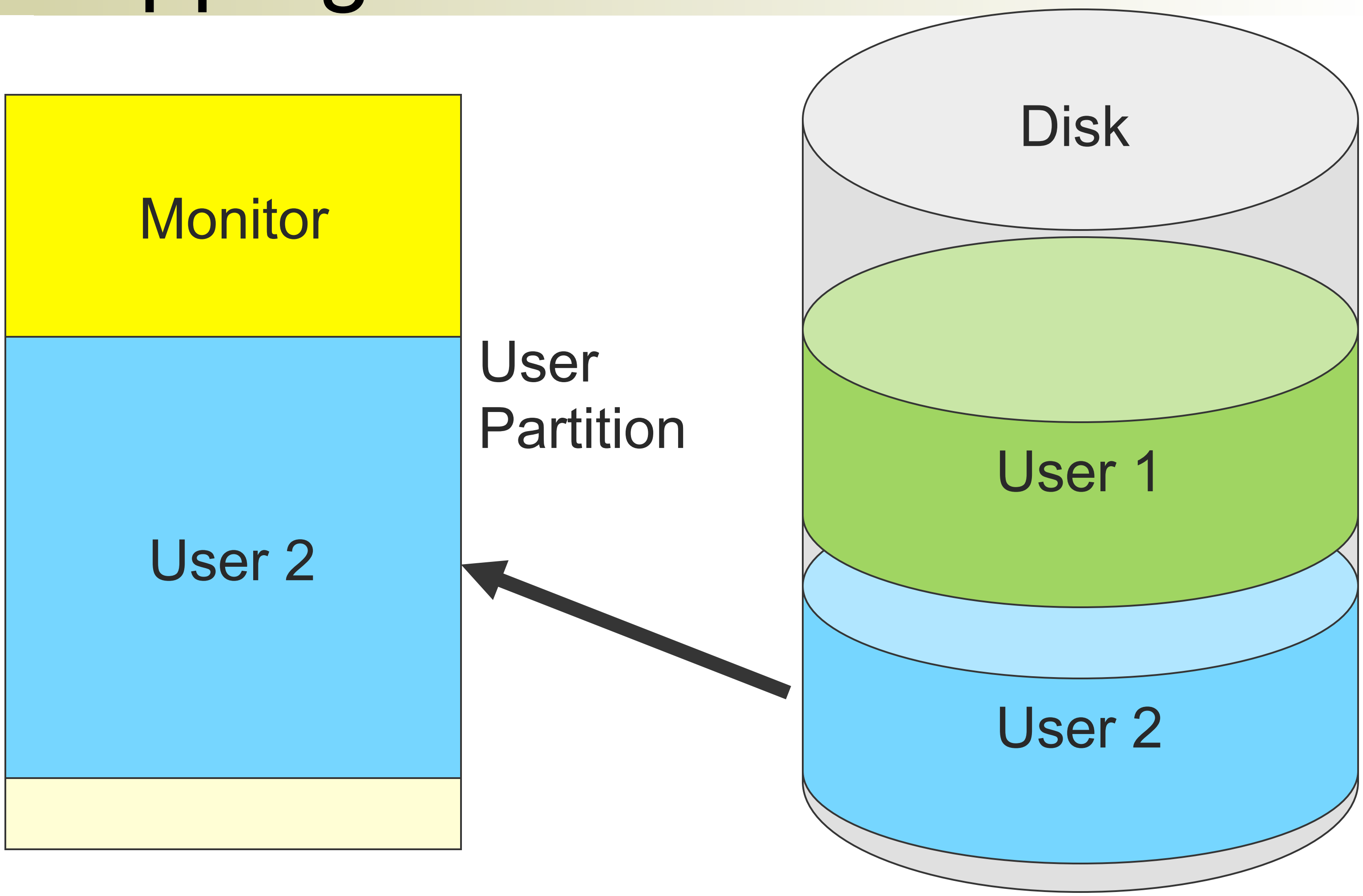
Swapping



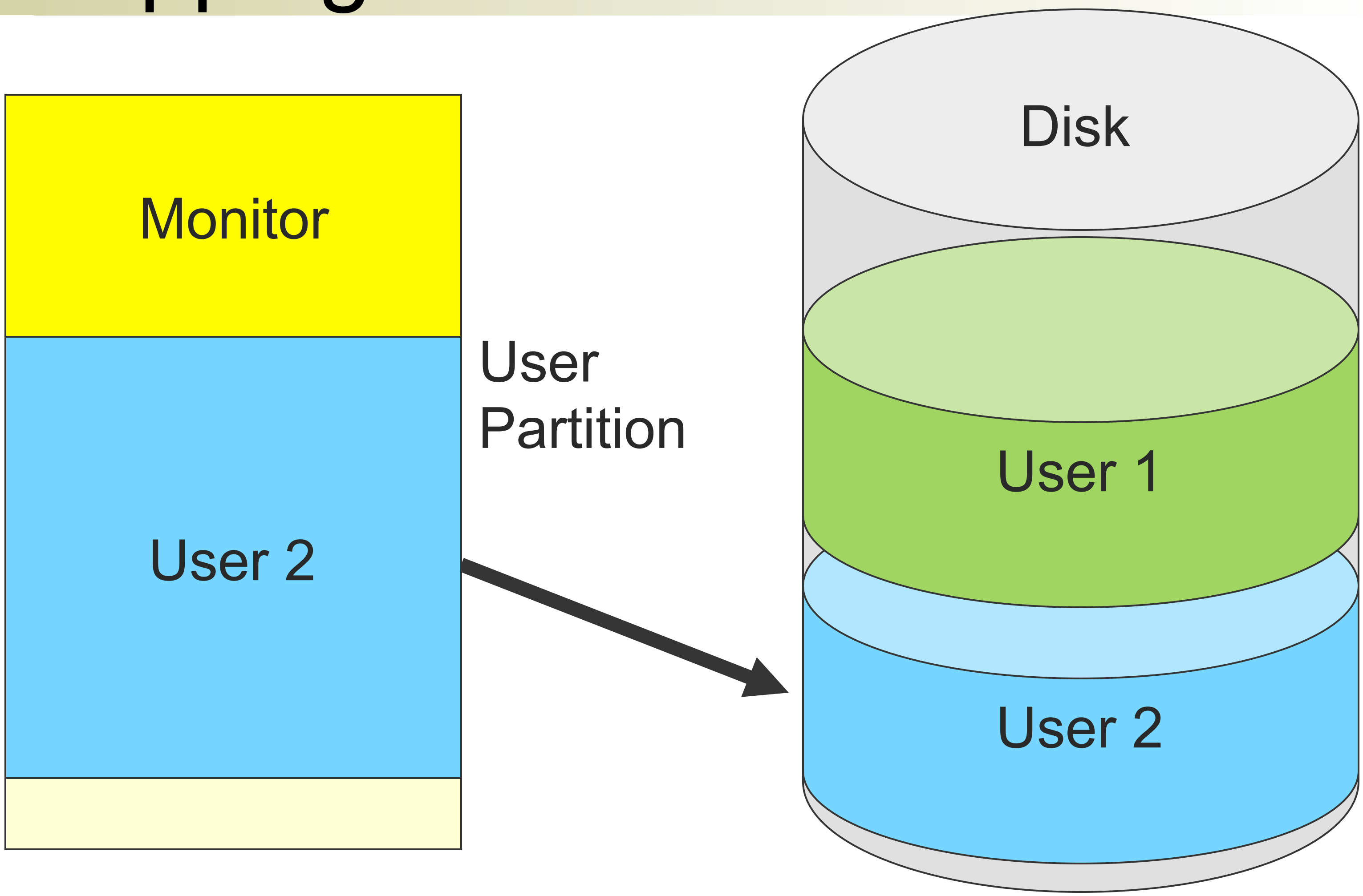
Swapping



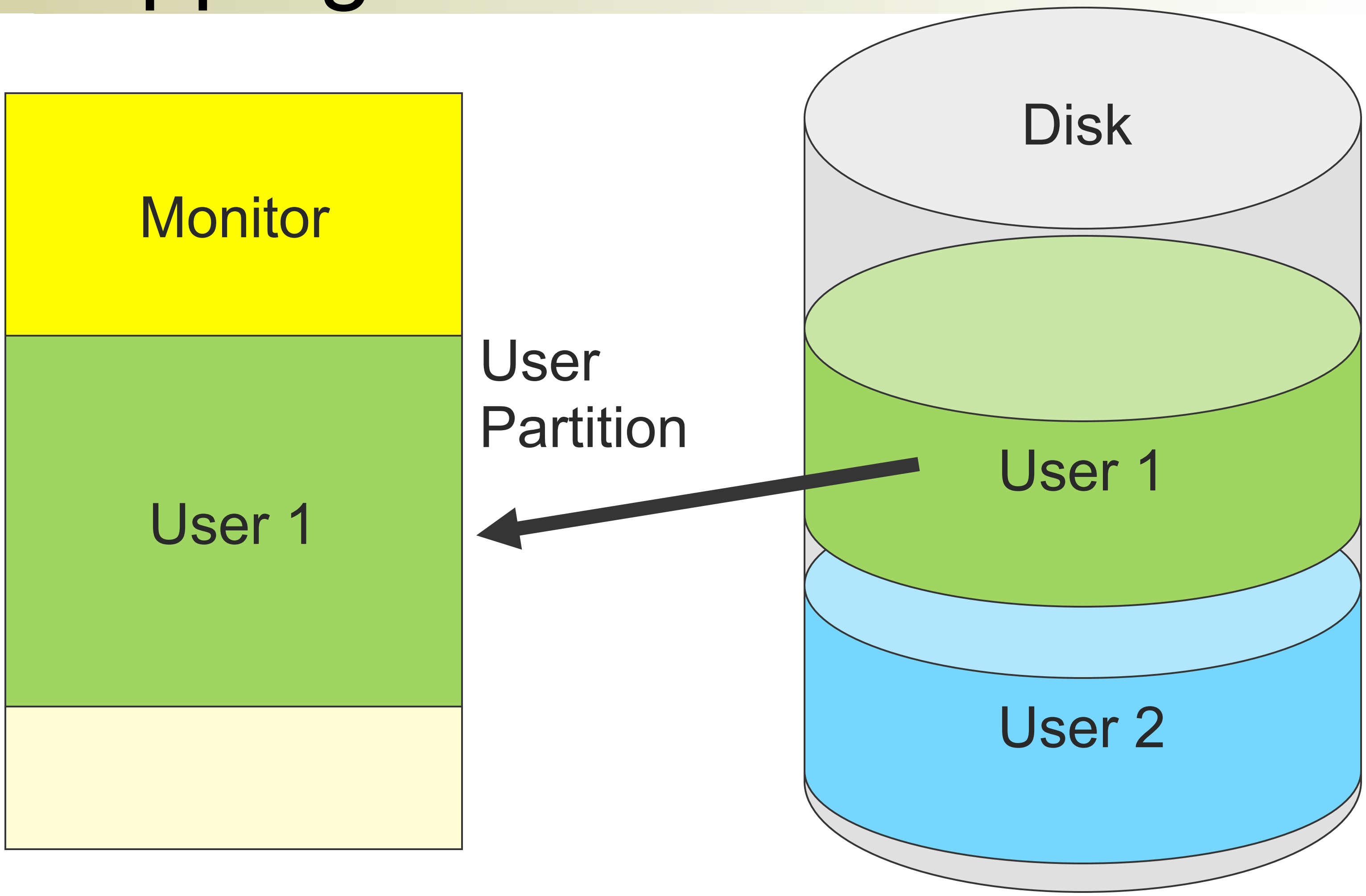
Swapping



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 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 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 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

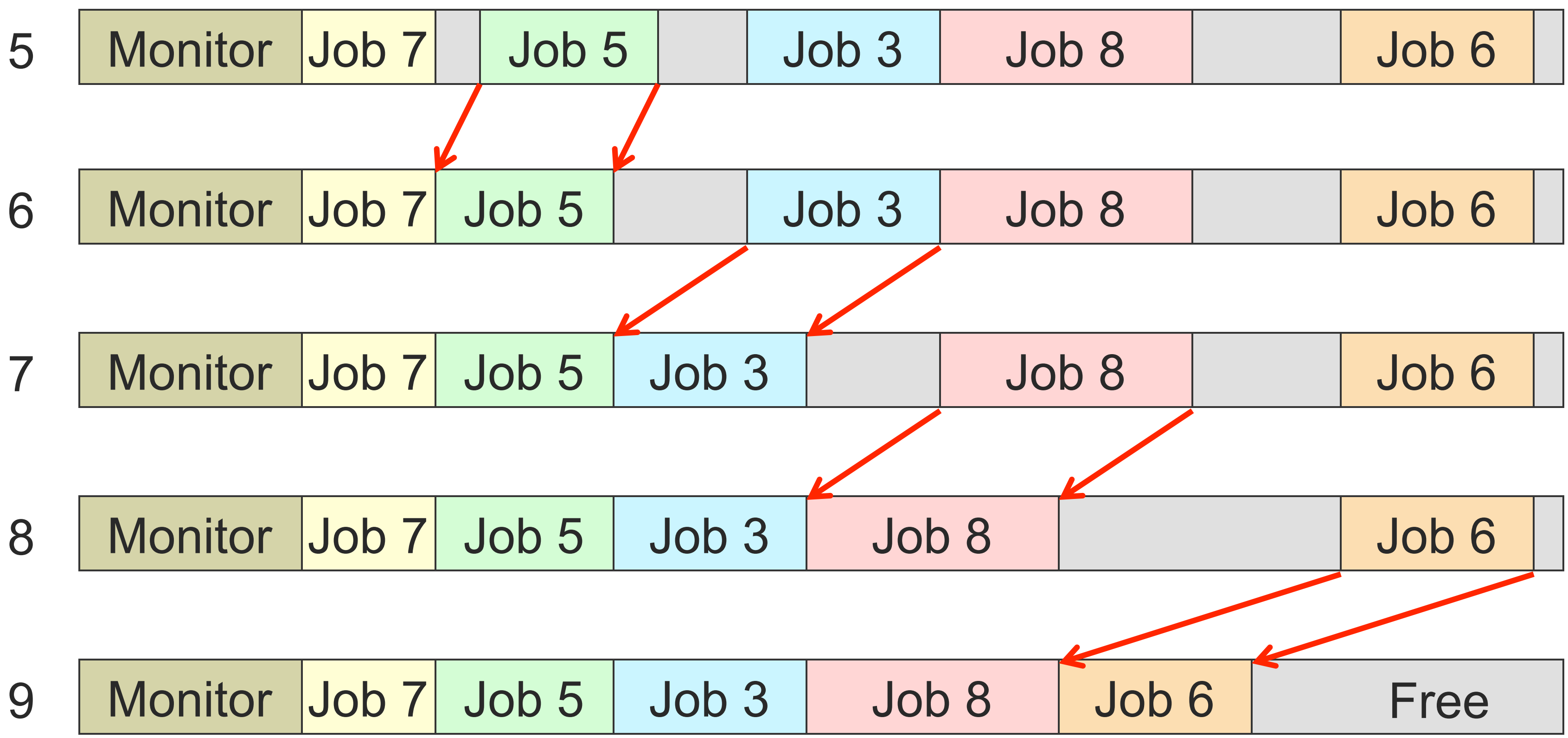
Fragmentation

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

Compaction

- Reduce external fragmentation by compaction
  - Shuffle memory contents to place all free memory together in one large block
  - Compaction is possible only if relocation is dynamic, and is done at execution time

Solve Fragmentation w. Compaction



Limitations of Swapping

- Problems with swapping
  - Process must fit into physical memory (impossible to run larger processes)
  - Memory becomes fragmented
  - Processes are either in memory or on disk
    - Half and half doesn't do any good

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

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

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

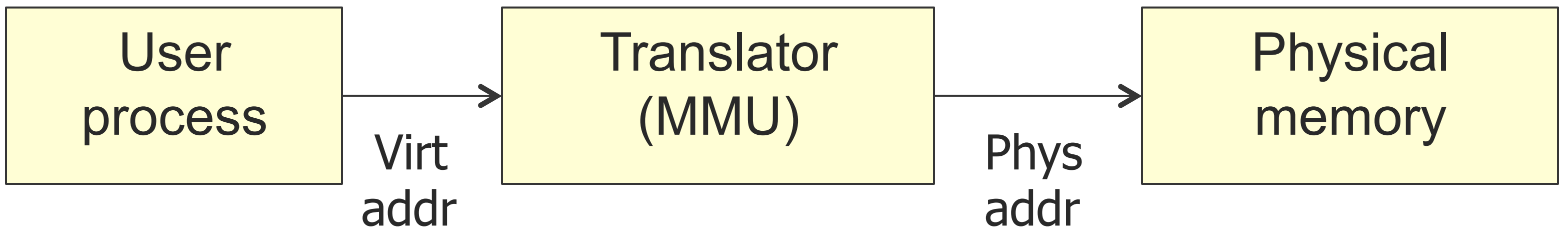
Dynamic Address Translation

- 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

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

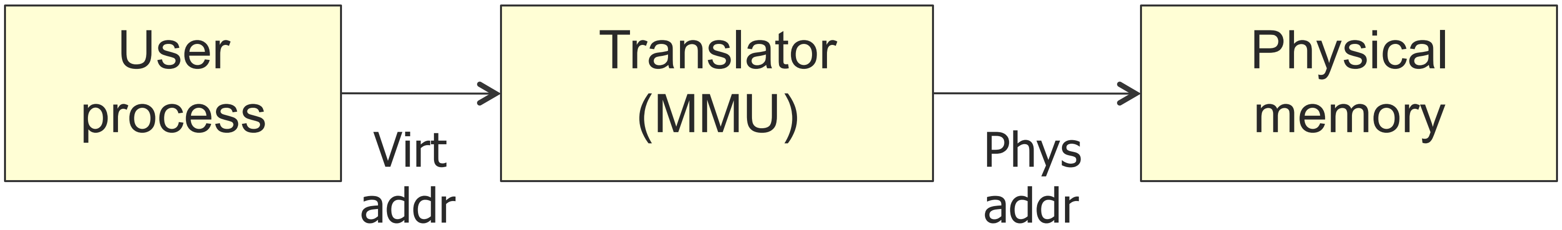
Wheeler On Indirection

"Any problem in computer science can be solved with another level of indirection…

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

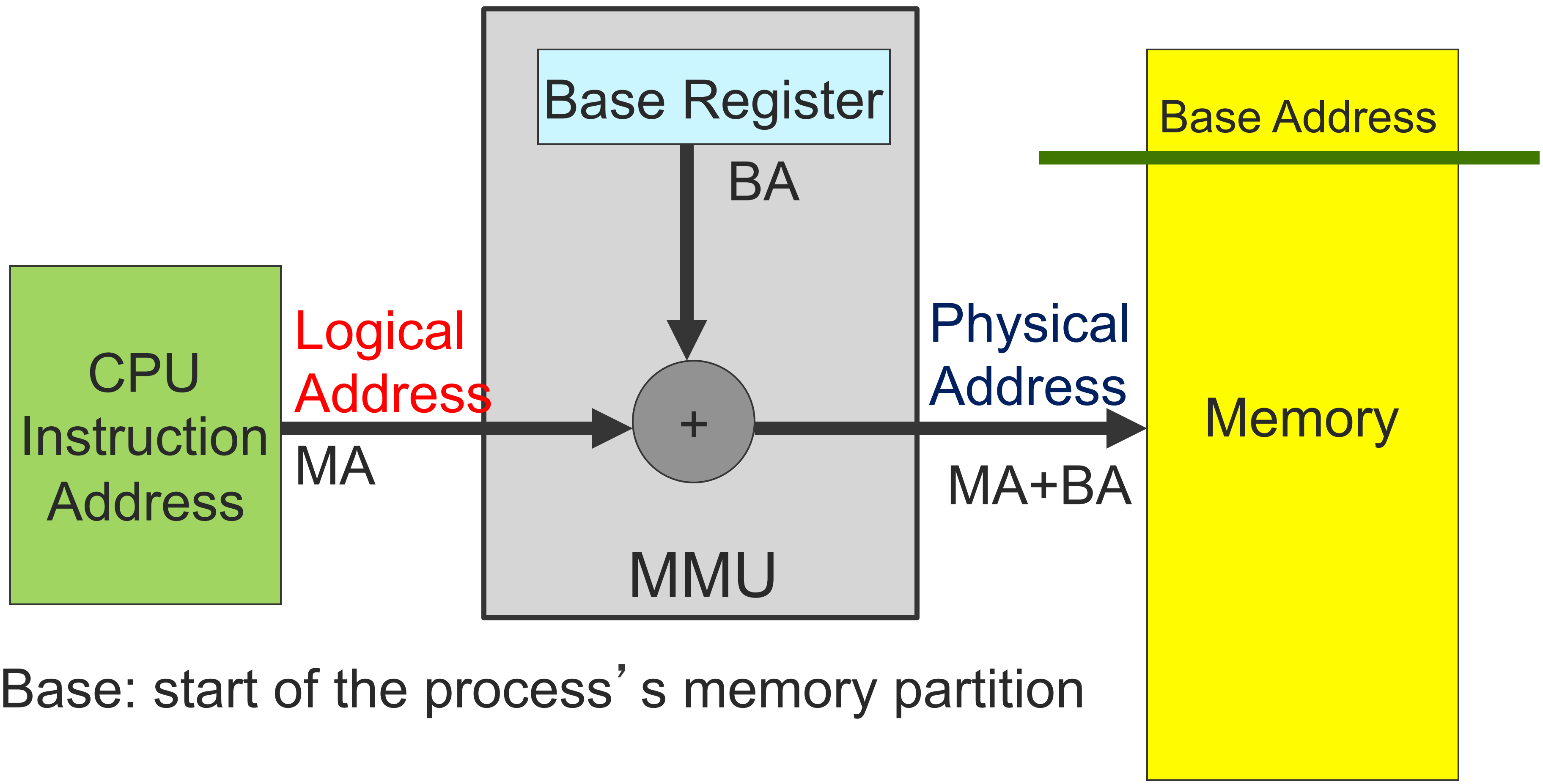
David Wheeler

Dynamic Address Translation



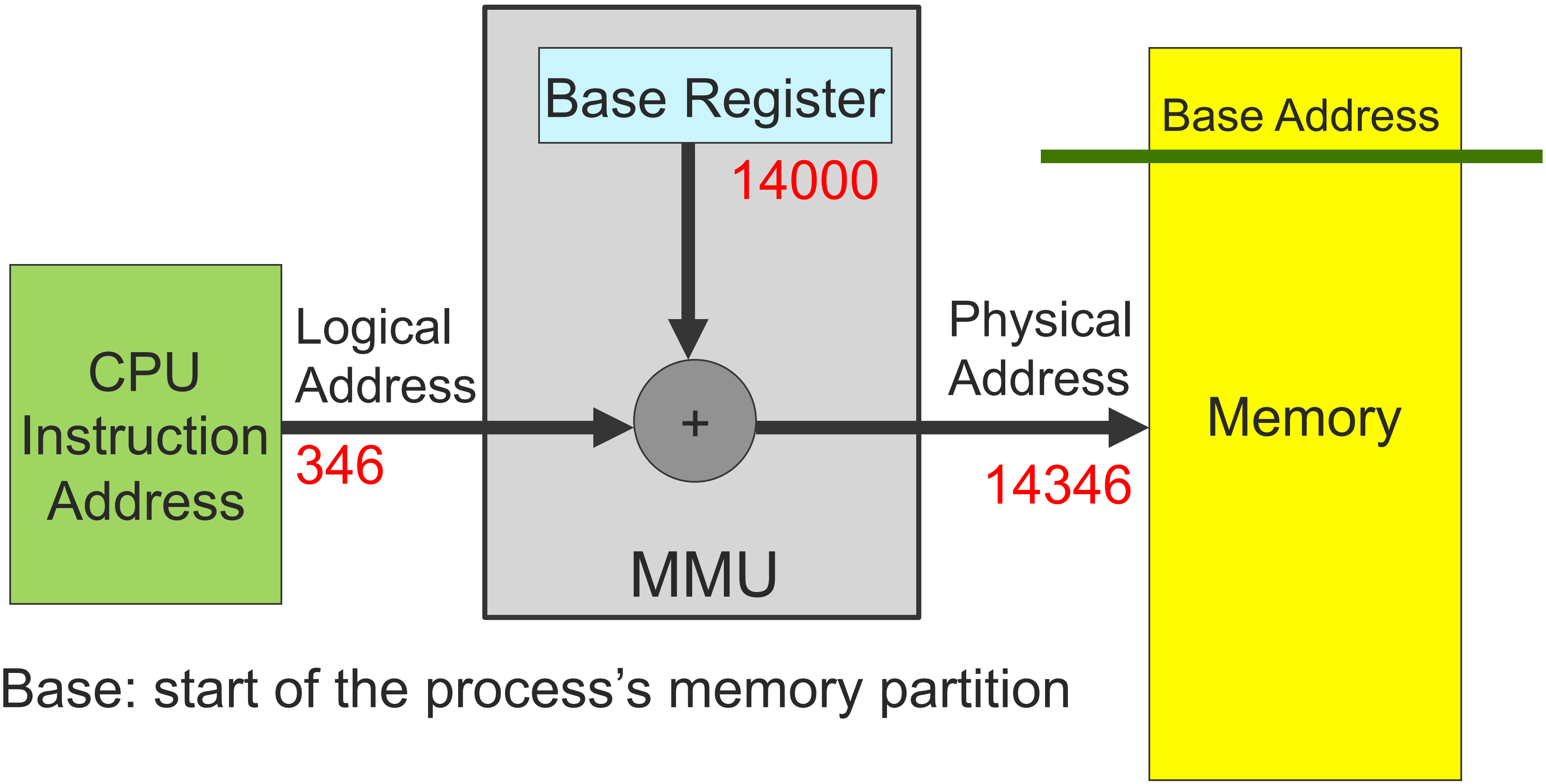
- Implementation tradeoffs
  - Flexibility (e.g., sharing, growth, virtual memory)
  - Size of translation data
  - Speed of translation

Base Register



Base: start of the process's memory partition

Base Register



Base: start of the process's memory partition

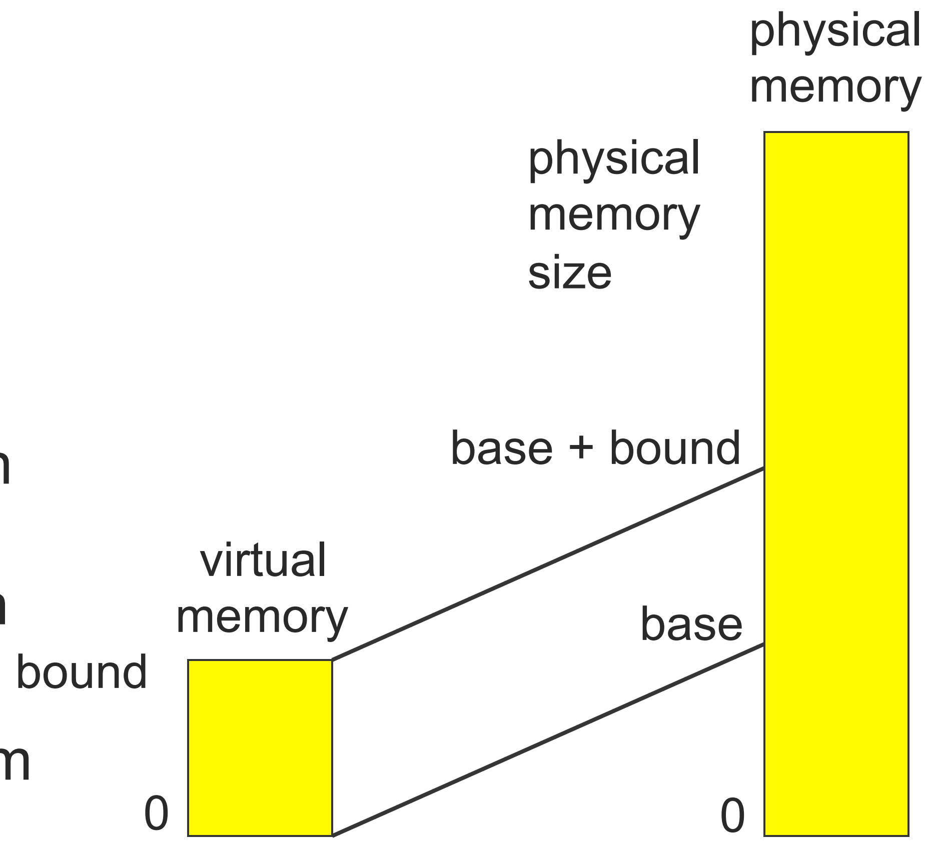
Protection

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

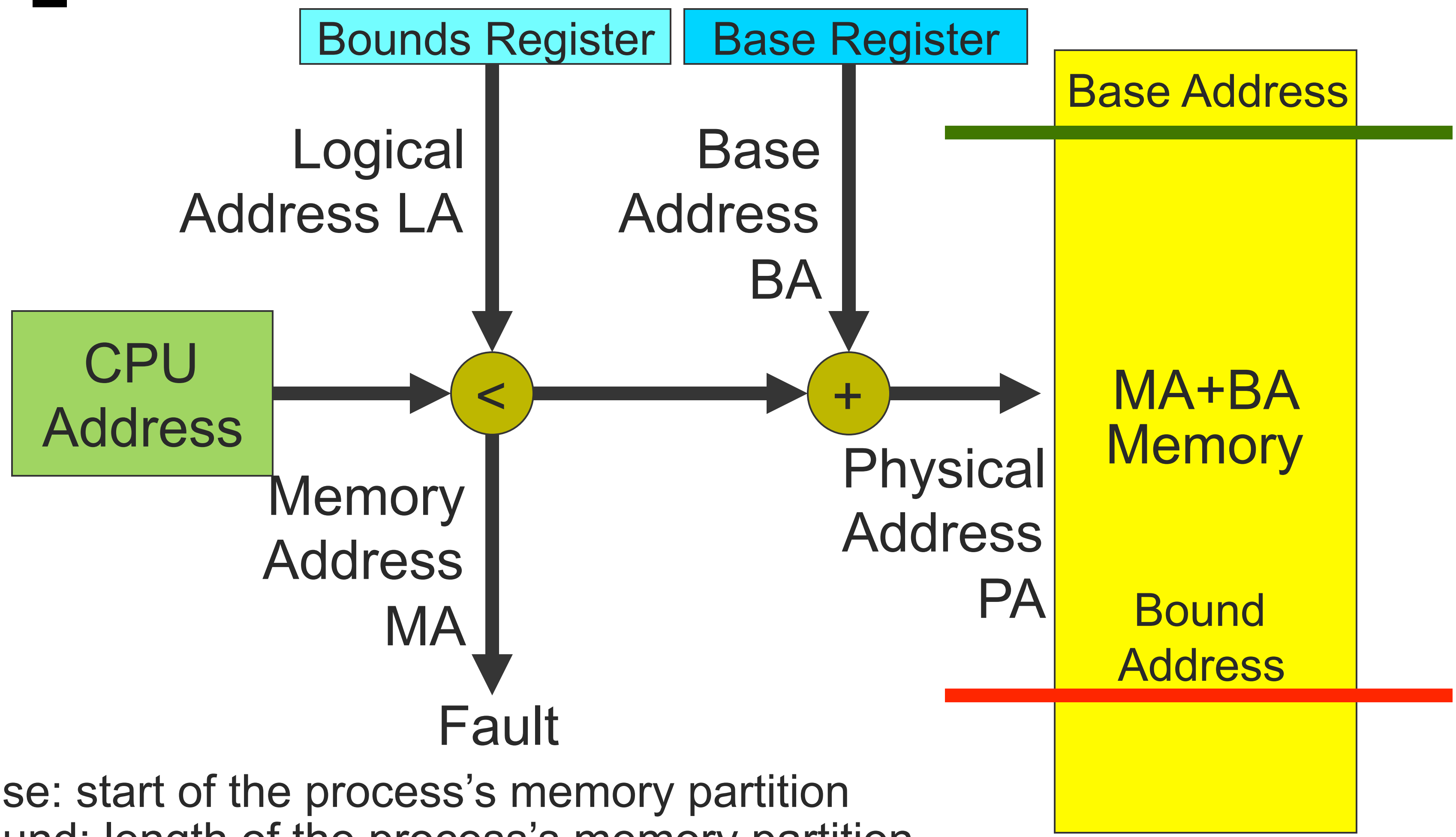
Base and bounds

```
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



Base and bounds



Base: start of the process's memory partition
Bound: length of the process's memory partition

Base and bounds

- What must change during a context switch?
- Can a process change its own base and bound?
- Can you share memory with another process?

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