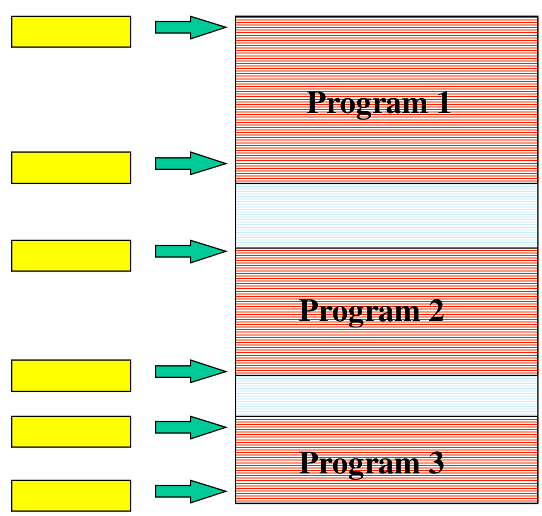
Dynamic Partition Allocation

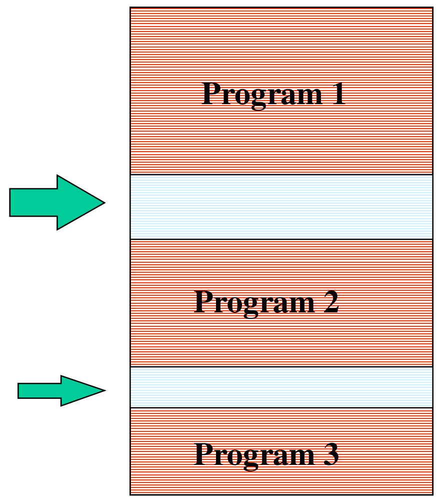
- Allocate memory depending on requirements
- Partitions adjust depending on memory size
- Requires relocatable code
  – Works best with relocation registers

Dynamic Partition Allocation



External Fragmentation

These fragments are not allocated to any program, and they are wasted.

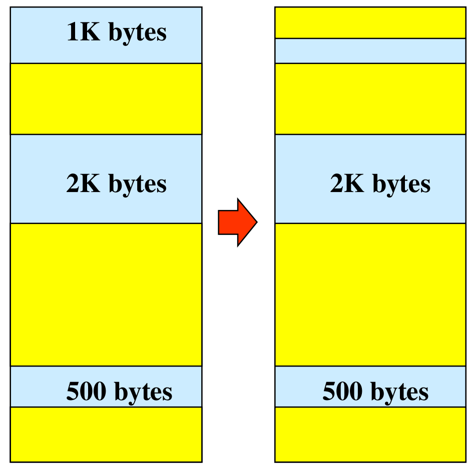


Allocation Policies

- Best Fit
- First Fit
- Worst Fit
- Rotating First Fit

First Fit Allocation

To allocate *n* bytes, use the *first* available free block such that the block size is larger than *n*.



To allocate 400 bytes, we use the 1st free block available

Rationale & Implementation

- Simplicity of implementation

Requires:

- Free block list sorted by address
- Allocation requires a search for a suitable partition
- Deallocation requires a check to see if the freed partition could be merged with adjacent free partitions (if any)

First Fit Allocation

Advantages

- Simple
- Tends to produce larger free blocks toward the end of the address space

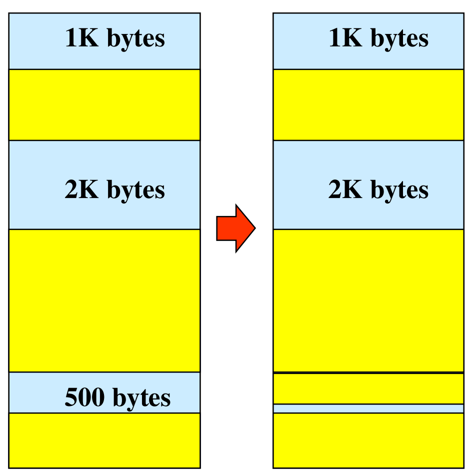
Disadvantages

- External fragmentation

Best Fit Allocation

To allocate *n* bytes, use the *smallest* available free block such that the block size is larger than *n*.

To allocate 400 bytes, we use the 3rd free block available (smallest)



Rationale & Implementation

- To avoid fragmenting big free blocks
- To minimize the size of external fragments produced

Requires:

- Free block list sorted by size
- Allocation requires search for a suitable partition
- Deallocation requires search + merge with adjacent free partitions, if any

Best Fit Allocation

Advantages

- Works well when most allocations are of small size
- Relatively simple

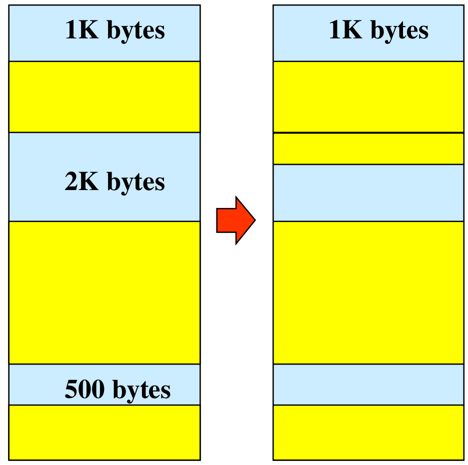
Disadvantages

- External fragmentation
- Slow allocation
- Slow deallocation
- Tends to produce many useless tiny fragments (not really great)

Worst Fit Allocation

To allocate *n* bytes, use the *largest* available free block such that the block size is larger than *n*.

To allocate 400 bytes, we use the 2nd free block available (largest)



Rationale & Implementation

- To avoid having too many tiny fragments

Requires:

- Free block list sorted by size
- Allocation is fast (get the largest partition)
- Deallocation requires merge with adjacent free partitions, if any, and then adjusting the free block list

Worst Fit Allocation

Advantages

- Works best if allocations are of medium sizes

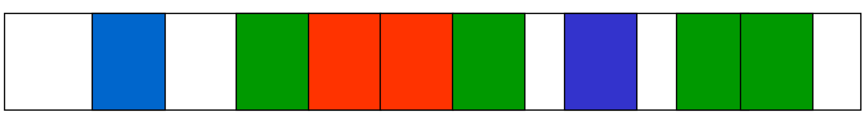
Disadvantages

- External fragmentation
- Tends to break large free blocks such that large partitions cannot be allocated

How bad is external fragmentation? (Knuth)

H: # of holes

S: # of used segments



blue: when released, H is decreased by 1
green: when released, H is unchanged
red: when released, H is increased

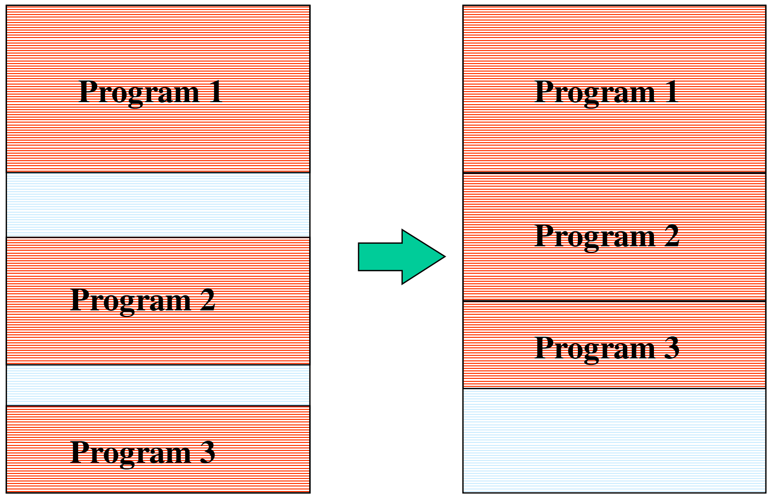
$S = b+g+r \quad H = (2b+g+\varepsilon)/2$

At equilibrium, $b = r$

$S = b+g+r = 2b+g = 2H$ or $H = S/2$

number of holes is half of number of segments!

Compaction (Burping)



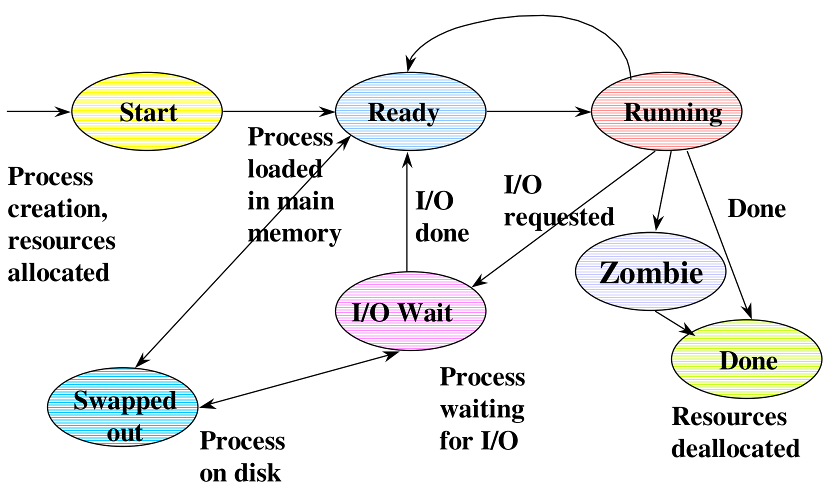
Multiprogramming

- Multiprogramming: Ability to run more than one program simultaneously
- Early systems: degree of multiprogramming had to be large to maximize the return on hardware investment
- Multiprogramming requires all programs to be resident in memory simultaneously
  - Can be done by partitioned memory allocation, but
  - Size of memory limits the number of programs that can run simultaneously

Swapping

- Allows to schedule more programs simultaneously than the memory can accommodate
  - Use a partition on disk to “extend” the main memory
  - Processes are added until memory is full
  - If more programs need to be loaded, pick one program and save its partition on disk (the process is swapped out, cannot run)
  - Later, bring the process back from disk (process is swapped in) and swap out another process
- Done on an all-or-nothing basis
  - The entire process is either in memory or on swap space (cannot be between both),

Memory & Scheduling



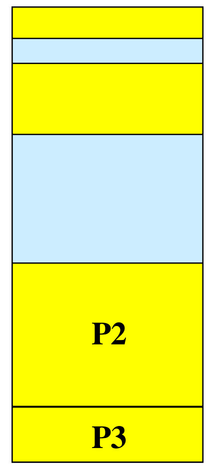
Memory & Scheduling (cont'd)

- Need two levels of scheduling:
  - Upper level decides on swapping:
    - When
    - Who
    - For how long
  - Lower level decides on who actually runs on the CPU (what we have covered so far)
- Upper level is invoked if there are processes swapped out, and whenever we need to load more programs than can fit in main memory

Paging

Memory Partitioning Troubles

- Fragmentation
- Need for compaction/swapping
- A process size is limited by the available physical memory
- Dynamic growth of partition is troublesome
- No winning policy on allocation/deallocation



The Basic Problem

- A process needs a contiguous partition(s)

But

- Contiguity is difficult to manage

Contiguity mandates the use of *physical* memory addressing (so far)

The Basic Solution

- Give illusion of a contiguous address space
- The actual allocation need not be contiguous
- Use a Memory Management Unit (MMU) to translate from the illusion to reality

A solution: Virtual Addresses

- Use n-bit to represent *virtual* or *logical* addresses
- A process perceives an address space extending from address 0 to 2^n-1
- MMU translates from virtual addresses to real ones
- Processes no longer see real or physical addresses

Paged Memory

- Subdivide the address space (both virtual and physical) to "pages" of equal size
- Use MMU to map from virtual pages to physical ones
- Physical pages are called frames

Paging: Example