- No. and sizes of partitions are "fixed"
- Find a free partition of size ≥ job's size
- Static ~~partition~~ specification table keeps status of all partitions

Binding a logical address to a physical address

Static binding (early binding) (binding before execution - at the time of compile/load/link)
eg. - static partition mm
- dynamic partition mm

dynamic binding (late binding) (takes place at the execution time)
eg. - relocatable dynamic partition mm
- paging
* segment/paging m.m.

- Static binding in Static Partition

eg.

Some of the Advantages and Disadvantages of Static Partitioning

Adv: - Very efficient because it only has to maintain the specification

Disadvs: - If the partition size > the job size, then the remaining space ("frag mentation") in the partition cannot be used by any other job. As a consequence we will have limited degree of multiprogramming

- The job's ultimate memory requirement will have to be decided (or predicted) at the loading. So we're overestimating the job size.

(X) - Job size is limited to the size of the biggest partition

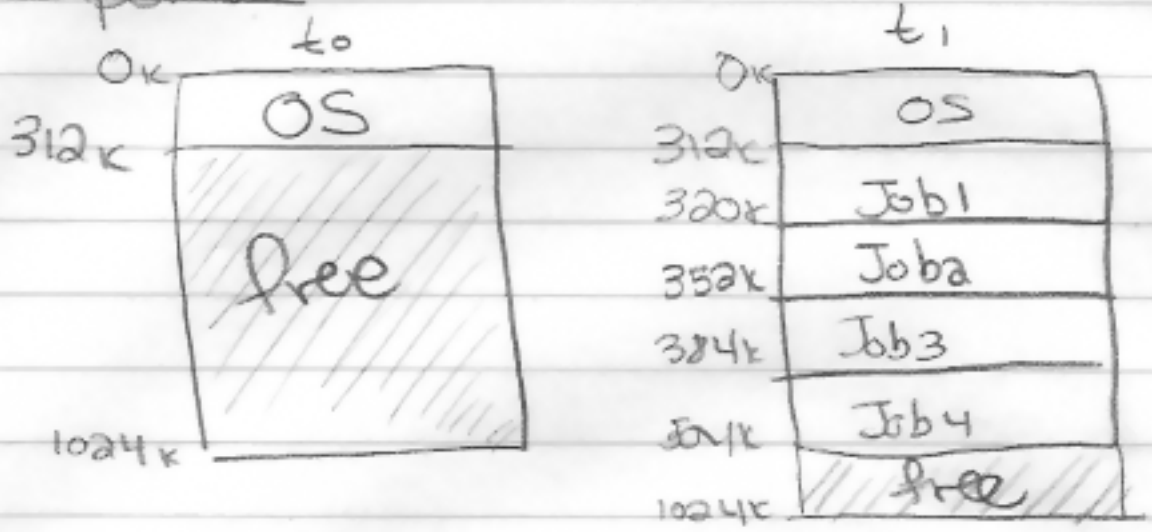
- Due to early binding, if a job has to be swapped out (temporarily) it has to be brought back in exactly the same order.

Dynamic Partition MM

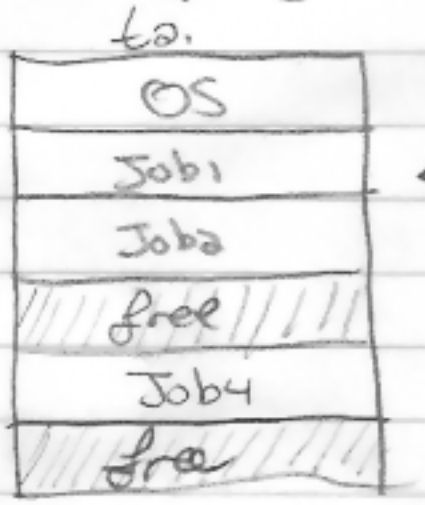
- Uses early binding

But - the sizes + the numbers of partitions are not fixed!

Snapshot



At time t_1, Jobs 1, 2, 3, 4 multiprogramming.



← Job3 is done.

Book keeping

- Status table (similar to static partitioning)

↑ A dynamic data structure

- free table (maintain the list of free partitions)

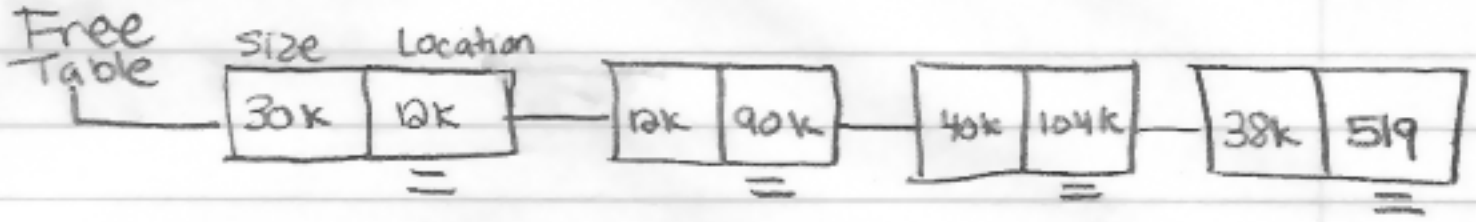
Q// ① which free partition?
② find adjacent partitions so they could be "collapsed" together.

Two Strategies (for maintaining free tables)

① First-fit Algorithm
② Best-fit Algorithm

First Fit
- Free table is kept sorted by the Location

eg. At time t



For Allocation
→ Find the first free partition when size ≥ the job's size
→ So in the worst case it may have to "look at" the entire table, because we are looking for size, while the free table is kept sorted by location

For Deallocation
→ Since we are searching for a neighboring free partitions, an efficient binary search routine can be used.

Adv: - Deallocation ratio is fast
- Since allocations are biased towards the low mem addr, it usually leaves large free space in the higher order addrs

Disadv: - Allocation routine could be linear worst case
- Sensitive to order at which requests come in.

eg. Job1 20k
Job 2 30k

will take first 20k, even if it blocks Job 2.