5. Dynamic Partitions

- There is no partition when the computer is turned on.
- Available memory is still kept in contiguous blocks but jobs are given only as much memory as they request.
- It still does not solve the memory-wasting problem completely.

external fragmentation: within dynamic partition, the allocation of memory creates fragments of free memory between blocks of allocated memory which are wasted.

- **External Fragmentation**: the dynamic allocation of memory creates fragments of free memory between allocated memory, which might be wasted.
- Notice that when memory is allocated, we can either use **first-fit** (first partition fitting the requirements) or **best-fit** (closest fit, the smallest partition fitting the requirements).
- The **first-fit** allocation method keeps a list of free/busy memory fragments ordered by memory address.
- The **best-fit** allocation method keeps a list of free/busy memory fragments ordered by memory size.
- What is the advantage/drawback of first-fit?

  - usually faster
  - might not use memory efficiently.

- What is the advantage/drawback of best-fit?

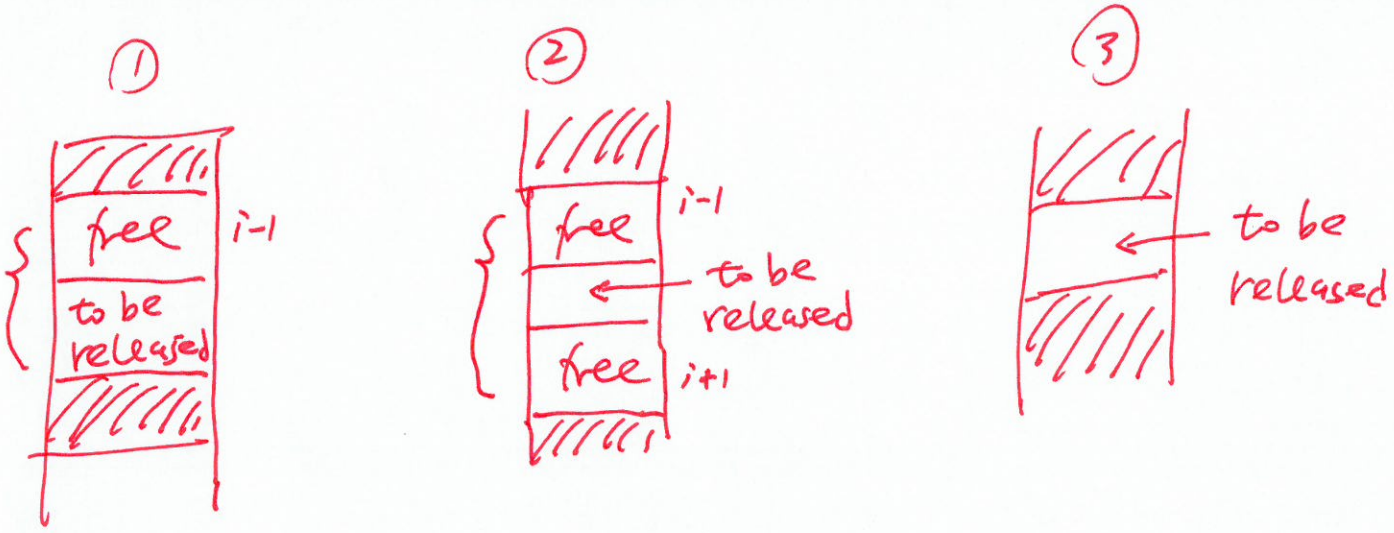
  - slower
  - space-efficient

- Algorithm: Dynamic partition–first fit
  - 1. i = 1. //i = counter
  - 2. While i<= number of blocks in memory
  - if job_size > memory_size(i)
  - then i = i + 1.
  - else
  - load job into memory_size(i)
  - adjust free/busy memory lists
  - go to step 4.
  - 3. Put job in waiting queue.
  - 4. Process next job.

- **Algorithm: Dynamic partition–best fit**
  - 1. memory_block(0)=9999999. //largest memory address
  - 2. initial_memory_waste=memory_block(0)-job_size.
  - 3. j = 0. //j is the subscript.
  - 4. i = 1. //i is the counter.
  - 5. While i<= number of blocks in memory
  - if job_size > memory_size(i)
  - then i = i + 1.
  - else
  - memory_waste=memory_size(i)-job_size
  - if initial_memory_waste > memory_waste
  - then j = i
  - initial_memory_waste = memory_waste
  - i = i + 1.
  - 6. If j==0 then put job in waiting queue
  - else load job into memory_size(j)
  - adjust free/busy memory lists
  - 7. Process next job.

6. Memory Deallocation

- When memory spaces are not used anymore they must be released (returned back to) the system.
- Fixed partition: easy! just reset the 'free/busy' status variable.
- Dynamic partition: we need to combine free areas of memory.
- Think of 3 situations when a block is to be released:



- **Algorithm: Dynamic partition–memory deallocation**

```
– 1.  If job_location is adjacent to one or more
  free blocks
–     then
–       if job_location is between 2 free blocks
–          then merge all three blocks into one.
–               memory_size(i-1)=memory_size(i-1)
–                    +job_size+memory_size(i+1).
–               set the status of memory_size(i+1)
–               to NULL.
–          else merge both blocks into one
–               memory_size(i-1)=memory_size(i-1)
–                              +job_size.
–     else search for NULL entry in free memory
–          list.
–          enter job_size and beginning_address
–          in the entry list.
–          set its status to FREE.
```

7. Relocatable Dynamic Partitions

- Both of the previous partitions introduce internal/external fragmentations.
- Relocatable dynamic partition is a natural solution.
- The basic idea is **garbage collection (memory compaction)**—reuse unused memory blocks.
- How to do garbage collection?
  - 1. Bounds register is used to store the highest (or lowest) location in memory accessible by each program.
  - 2. The relocation register contains the value, which could be positive or negative, and that must be added to each address referenced in the program so that the memory addresses are correctly accessed after relocation.

8. The drawback of relocatable dynamic partition

- **Overhead!**
- When should we use relocatable dynamic partition?
  - 1. When memory is busy.
  - 2. When many jobs are waiting.
  - 3. When the waiting time is too long.
- What can we learn from relocatable dynamic partition?
  - 1. Programs do not have to be stored completely in memory.
  - 2. Programs do not have to be stored contiguously in memory.

CSCI 460—Operating Systems

Lecture 3

Memory Management–recent systems

Textbook: Operating Systems
by William Stallings

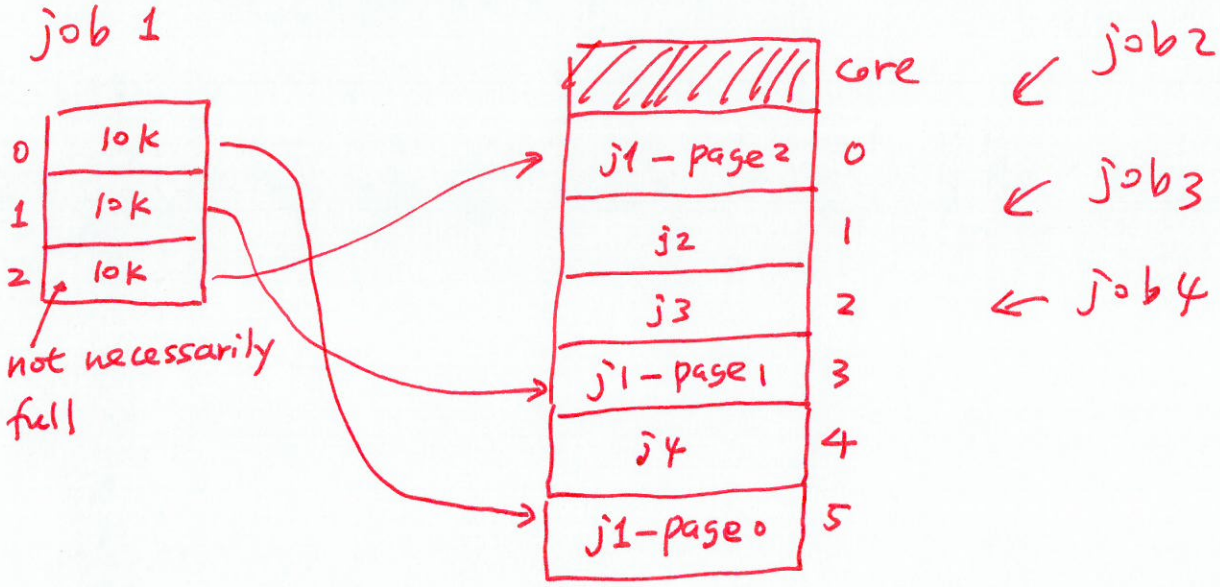
1. Paging (Paged memory allocation)

- Does a program have to be resided completely and contiguously in the main memory for execution? NO!
- IDEA: Dividing an incoming job into memory blocks (frames) of equal size, which are called **pages**.
- To execute a program, Memory Manager must do the following
  - 1. Decide # of pages in the program.
  - 2. Have enough empty page frames in main memory.
  - 3. Load all the program's pages into them.
- Advantage
  - 1. Memory is certainly used efficiently.
  - 2. No external fragmentation.
  - 3. Almost no internal fragmentation.

Fragmentation
- no external fragmentation
- internal fragmentation can possibly occur in the last page of a job.

- Drawback? *Overhead is increased significantly.* An OS nowadays has to be designed by experts and by substantial teamwork.

- How do we manage paging?
  - 1. Job Table.
  - 2. Page Map Table (for each job).
  - 3. Memory Map Table.



Job Table

job #	size	PMT location
1	26K	xxxxx

pointer

PMT (for job 1)

page #	page frame number in memory
0	5
1	3
2	0

Memory Map Table

	page frame #	status
0	0	busy
1	1	..
2	2	free
3	3	busy
4	4	..
5	5	busy

if only consider job 1.