Module - III

Memory & File Management

=> **Memory Managmnt:**

* Mly managmnt module of OS is concerned with the managmnt primary mly.
* Manages primary mly & moves processes back & forth b/o main mly & disk during execution.
* It checks how much mly is to be allocated to processes.
* CPU fetches instrn from mly according to the value of the prgm countr.
* Main aim of this is to achieve efficient utilization of mly.
* It is required
  - to minimize fragmentn issues
  - to maintain data integrity
  - to proper utilization of main mly.
* divided into 2 —

a) **Contiguous Mly managmnt:** assumes a prgm's data & instrn to occupy a single contiguous mly area (mly blocks having consecutive add).

b) **Non-contiguous Mly.M:** a prgm's data & instrn may occupy many non-contiguous mly area (mly blocks hving non-consecutive add)

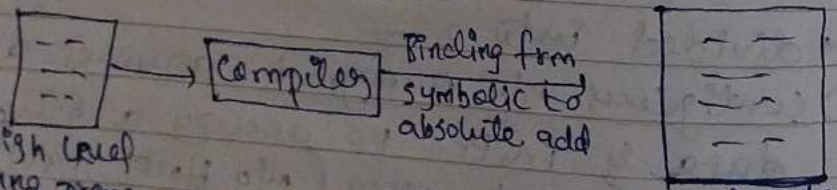
=> Background :

I Add Binding :

* means mapping comp instrn & data to loc in RAM.
* high level prgms are loaded into m/y for their execution in the CPU.
* m/y managmnt unit (MMU) translates logical add into physical add.
Mapping a logical add to a physical add → add binding / add mapping.
* 3 types —

a) compile-time Add. B :

- If the compiler is responsible for performing add binding then it is c-t. add. B.
- It will be done before loading the prgm into m/y.



b) Load time Binding :

- It will be done after loading the prgm into m/y
- This type of add binding will be done by the OS m/y managr.

c) Execution time binding :

- It will be postponed even after loading the prgm into m/y
- It is done by the processor at the time of prgm execution.

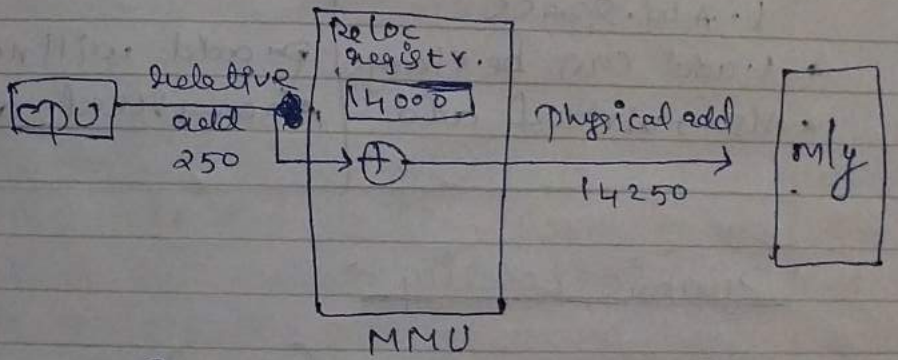


fig : Load time binding.

II Logical vs Physical add space :

Logical add. S	Physical add. S
* It is a virtual add generated by CPU * user can view the logical add of a prgm * user uses l. add to	It is a loc in m/y unit. — never view Physical add of prgms user can not directly

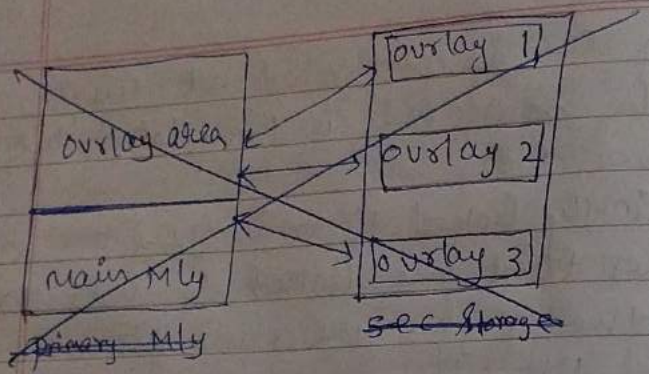
access the physical add.	access physical add.
* l.add is generated by the CPU	p.add is computed by MMU.
* Set of all l.add generated by CPU in refrnce to a prgm is referred as l.Add.Space	set of p.add mapped to the corresponding l.add is referred as p.add.space.
* l.add can be change	p.add will not change
* Also → virtual add	Also → real add.

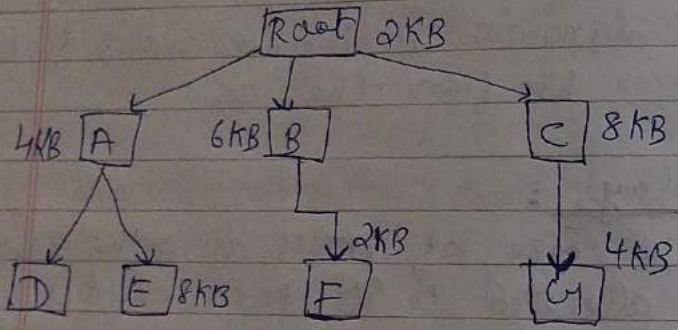
III Dynamic Loading :

* Also → Load on Call (LOCAL)
* A routine is not loaded until it is called. All routines are kept on disk in a relocatable load format.
* Adv → unused routine is never loaded.
* does not require special sprt from the OS.
* Dynamic linking : process of collecting & combining various pieces of code & data into a single executable file from multiple obj files.
* Linking can be performed at compile time, when the source code is translated into machine code.
* Dynamically linked libraries (DLL) are system libraries that are linked to user prgms when the prgms are run. Also → shared libraries.
* unlike dynamic loading, dynamic linking requires hlp from the OS.

IV overlays :

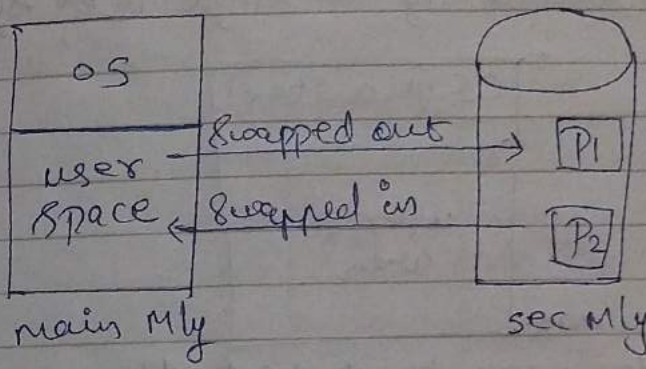
* If the size of process is > the amount of mly allocated to it, then overlays are used to execute such prgms
* An overlay is a part of prgm, which has the same load origin as some other parts of the prgm.
* used to reduce the main mly req of a prgm.
* A prgm containing overlays is → an overlay-structured prgm → consists of a permanently resident portion → root & set of overlays.
* Adv → using less mly
* disadv → Extensive prgming req.



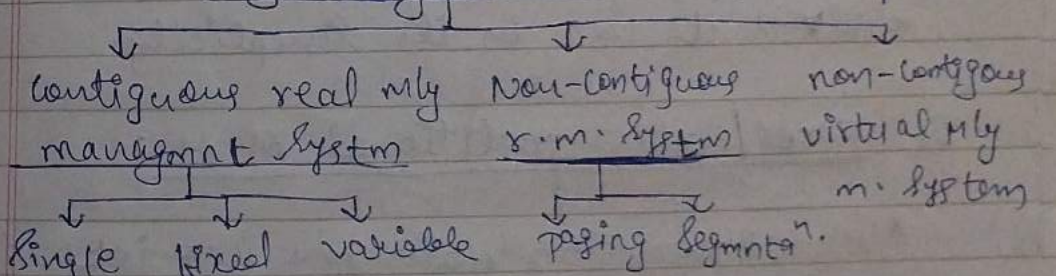


V Swapping :

* Temporary removal of suspended processes from mly to sec storage disk & their subsequent bringing back → (S).
* Its responsibilities —
  - Selection of Ⓟ to swap out
  - " " swap in
  - Allocation & managmnt of swap space
* Size of main mly is limited.
* once a Ⓟ is selected for swap out, the swapper keeping to swap it out.
* It has traditionally been used to implemnt multiprgming systms.



* Mly managmnt schemes / techniques

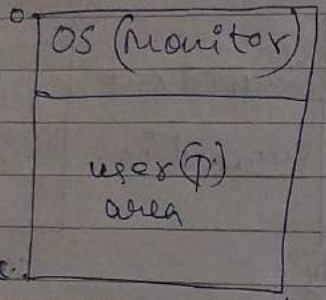


I Contiguous Mly Allocation :

1) Single c.M. managmnt :

* Here, the physical mly is ÷ into 2 contiguous areas.
* It is simple & easy. Basically the OS needs to keep track of the first & last

loc available for allocation to user (p).
* A new user (p) may be activated upon termination of the running (p).
* M/y partitioning in single user system —

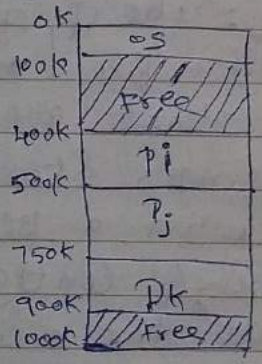


* A single way to protect the OS code frm user prgm is to place the OS in ROM.
* Adv → it lies in its simplicity.

2) Fixed/static partitioned M.M :

* In c.p. allocation, main m/y is ÷ into several partitions. Depending on ¢ how partitions are created ¢ modified, m/y partitioning may be static / dynamic.
* In static p. m/y managmnt scheme, the available m/y is logically ÷ into several partitions during system generation (p). These partitions can be of diffrnt sizes ¢ cannot be changed.

* 1 (p) may be executed out of a given partition at any time, the no. of distinct partitions represnts an upper limit on the no. of active (p)s in a system. This figure → degree of multiprgming.



3) Dynamic/variable p. m/y. M :

* It was developed to overcome some of prbms associated with fixed p. m/y. M like prbms of internal fragmen", restrictions of degree of mp.
* Here, the no. of partitions ¢ their sizes are variable. partitions are not defined at the time of system genera".
* partitions are defined dynamically in accordance with the req of incoming (p)s.
* Here, OS must keep track of both partitions ¢ free m/y.
* Common (al) for selection of a free area

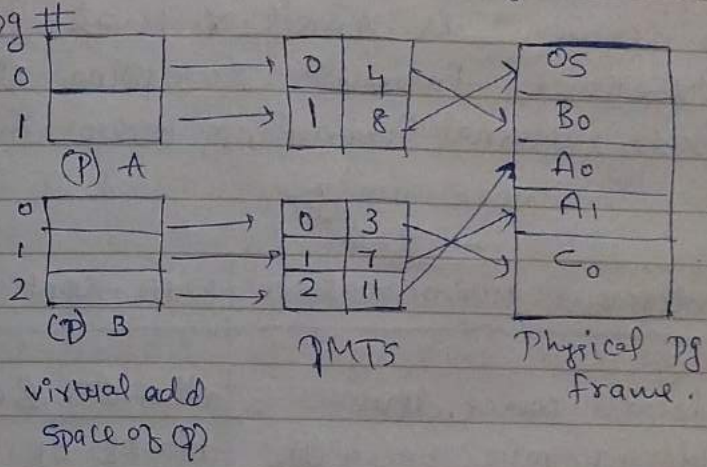
of mly for creation of a partitions -
First-fit, next-fit
Best-fit
worst-fit.

4) Fragmentation: The ⓟ of dividing a comp file such as data file / an executble pg file into fragmts that are stored in diff parts of a comp's storage medium.
* This can hpn whn a file is too large to fit into a single contigous block of free space on the medium
* 2 types -
a) Internal (f): occurs whn there is unused space within a mly block.
b) External (f): occurs whn a storage medium has many small blocks scattered throughout it.

II Non-Contiguous M/y m:

1) Paging:
* Mly m. schene that removes the req of contiguous allocation of physical mly.
* Add mapping is used to maintain the illusion of contiguity of virtual add space of a (p).

* Basically, physical mly is ÷ into nom of fixed-size slots → page frames
* virtual add space of a (p) is also split into fixed size blocks of same size → pgs
*



* Adv → reduced fragmn
↳ Efficient use of mly
Disadv → Used complexity
↳ mly thrashing

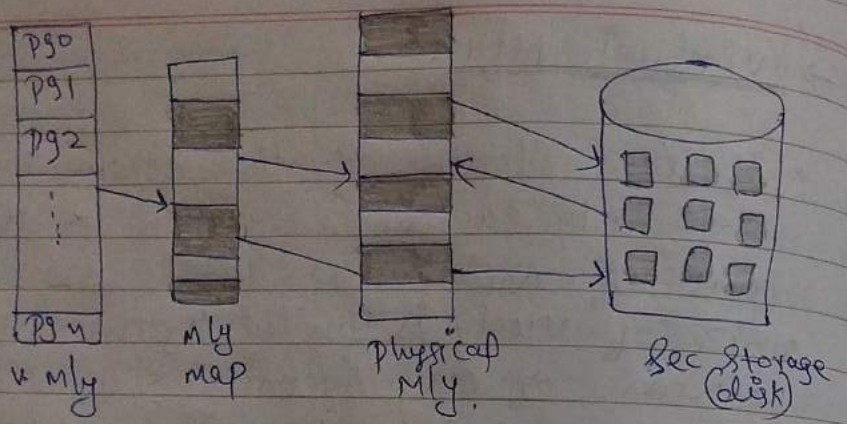
2) Segmentation:
* Allows breaking of the virtual add space of a single (p) into variable sized logical blocks → Segmnts.
* A Segmnt can be defined as a logical grping of info.
* Segmnts are formed at prgm translation time by grping togthr the logically related items.

* The os maintains a list of free m/y areas for the purpose of allocation & deallocation of m/y.
* The dynamic add translation mechanism in segmentn is similar that in pging sys
* Adv → no ex fragmn, swapping is easy.
* disadv → internal fragmn, pg tables may consume more m/y.

* contiguous m/y allocation	non-contn m/y allocn
* allocates consecutive blks of m/y to a (p).	allocates separate blocks of m/y to a (p)
* faster in execution	slower
* easier for the os to control.	difficult
* Both internal & ex fragmn occurs here	only ex fragmn occurs
* include single & multi partition allocation	include pging & segmentn.
* degree of multipsming is bound	not bound.

→ virtual m/y:

* It is a storage allocation scheme in which sec m/y can be addressed as it is a part of main m/y.
* Size of v.m/y is limited by the addressing scheme of comp system
* It is a technique that is implemented using both hardware & s/w.
* It maps m/y add used by a prgm → virtual add into physical add in comp m/y.
* Demand pging:
  process of loading the pg into m/y on demand → d.p.
  As the no. of (p)s & no. of pgs in main m/y for each (p) use, at some point of time, all pg frames become occupied.
  MMT (m/y map table)
  PMT (pg " ")
  Dirty Bit
  FMT (file mangmnt table)
  } os requires this data str to implemnt the v.m/y managmnt with demand pging.
* adv → no fragmn, more efficient use of m/y
* dis → complex hardware & s/w required.
  → possibility of thrashing must be handled at design stage.



⇒ Page Replacemnt :-

* In an os that uses pging for mly managmnt a pg replacmnt (al) is needed to decide which pg needs to be replaced whn a new pg comes in.
* A pg fault hpns whn a running pgm accesses a mly pg that is mapped into the virtual add space bt not loaded in physical mly since actual physical mly is much smaller than virtual mly, Pg faults hpn.
* (al) —

I FIFO pg.R :

Simplest P.R (al). here, os keeps track of all pgs in the mly in a queue, the oldest pg is in the front of the queue. whn a pg needs to be replaced pg in the front of the queue is selected for removal.

Pg fault → (*) → pg which is not prnt in the main mly

Pg hit → (hit) → " is prnt

refrnce no	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
refrnce str	7	0	1	2	0	3	0	4	2	3	0	3	1	2	0
Pg framed	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0
" 1		0	0	0	0	3	3	3	2	2	2	2	1	1	1
" 2			1	1	1	1	0	0	0	3	3	3	3	2	2
	*	*	*	*	*	*	*	*	*	*	*	*	hit	*	hit * hit

pg fault ① → main mly full so pg.R hpns, so according to FIFO, 7 swaped to disk

② → here 0 is already prnt on rfrnce no 4 so write like this. (i.e) hit

∴ no. of pg hits → 3

hit ratio = $\frac{\text{no of hits}}{\text{total no of rfrnces}} \times 100 = \frac{3}{15} \times 100$

= 20 %

no of pg faults → 12

Pg faults ratio = $\frac{\text{no of pg faults}}{\text{total no of rfrnces}}$

$= \frac{12}{15} \times 100 = 80\%$.

II LRU (least recently used) pg.R :

* Here, pg will be replaced which is least recently used.
* It belongs to a large cls of stack replacemnt (al).
* eg →

R.no	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R.Str	(7)	0	1	2	0	3	0	4	2	3	0	3	1	2	0
PF 0	[7]	7	7	2	2	2	[2]	4	4	[4]	0	0	[0]	2	2
" 1		0	0	0	0	0	0	0	[0]	3	3	3	3	[3]	0
" 2			1	1	[1]	3	3	[3]	2	2	2	[2]	[illegible]	1	1
	*	*	*	①*	hit	②*	hit	③*	④	*	*	hit	*	*	*

① 2 → here 7 is least recently used pg & 1 is more recently used pg.

② 3 → ⟵ 1 2 0, here LRU is 1 & MRU is 0. so replace 1 with 3

③ 4 → ⟵ 2 0 3 0, LRU → 2 & MRU → 0.
→ R.Str-[illegible] 2 0 3, 0 [illegible] 3 [illegible]
[illegible] R.Str-[illegible] there 4 2030

∴ no. of hits = 3
no. of fault = 12

always look previous R.Str nums for pg

(iii) optimal pg. R:

* Here, pgs are replaced which would not be user for the longest duration of time in the future.
* It is the best possible policy, bt it is not realizable.
* use of this pg. R (al) guarentees the lowest possible pg fault rate for a fixed no. of frames.

* eg →

R.no	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
R.Str	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7
P.F 0	7	7	[7]	2	2	2	2	2	2	2	2	2	2	2	2	2	[2]	7
" 1		0	0	0	0	0	[0]	4	4	[4]	0	0	0	0	0	0	0	0
" 2			1	1	[1]	3	3	3	3	3	3	3	[3]	1	1	1	1	1
	*	*	*	①*	hit	②*	hit	*	hit	hit	③*	hit	hit	*	hit	hit	hit	④*

In FIFO & LRU we check pg on left, bt here it is on right →.

① 2 → longest timing future [illegible] (ie)
7 → R.no is 18 ✓, 0 → R.no is 5 &
1 → R.no is 14.

② 3 → 2 → R.no is 9
0 → R.no is 7
1 → R.no is 14 ✓

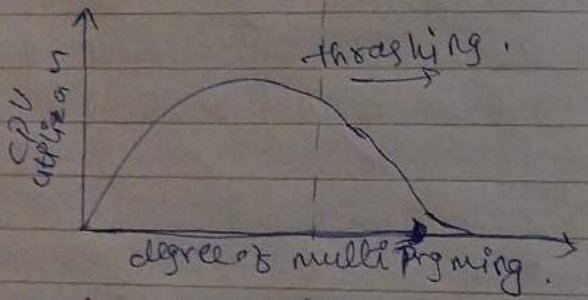
③ 0 → 2 → R.no is 13
4 → 4 is not used, so take 4 ✓
3 → R.no is 12

④ 7 → replace 2 with 7 (bcz 2 [illegible])

∴ no. of pg hit = 9
no. of pg fault = 9

→ **Thrashing:**

* A situation when system is spending most of the time in servicing pg faults (swapping) than executing the actual process
* occurs when a comp's virtual m/y resources are overused, leading to pg faults
* Bcz of thrashing, cpu utilization is going to be reduced



* (a) during thrashing → global pg·R
  → Local pg·R.

* **causes of thrashing:**
  - If cpu utiliz^{n} is too low, we ↑se the degree of multipgming by introducing a new system. A global pg·R (a) is used.
  - cpu utiliz^{n} is plotted against degree of m·p
  - As degree of m·p ↑se, cpu utiliz^{n} also ↑s
* **Techniques** to prevnt thrashing:
  1) Locality model (3) pg fault freq
  2) wrking-set model

⇒ **Files:**

* The info managmnt modules of OS → **F·system**
  It provides the resource abstractions typically associated with sec storage.
* It hides all the device specific aspects of file managmnt frm user.
* F·System maps files to physical media & accesses these files via storage devices.
* A file is a named collection of logically related info & it is the smallest allotmnt unit of logical sec storage.
* objectives → minimizes the chances of lost/destroyed data.
  → provide I/O spt for multiple users
  → provide I/O spt for a variety of storage device types.
* () → create file, file space on disk
  → write to file
  → read frm file
  → dlt directory entry
  → reposition: move read/write position.

1 **File System Str:**
needs to be predefined format in such a way that an OS understands.

* 3 types in OS → txt file → Series of Char
  → obj file → " of bytes
  → source file → " of ()s & (P)s.

user prgms
↕
Logical file System
File organiza^n module
Basic file System
I/O control.
↕
Storage devices

a) I/O control level consist of device drivrs & interrupt handlers to transfr info b/w the main m/y & the disk System.

b) Basic filesystm deals with blocks of data that are exchanged with disk | tape System.

c) File organiza^n module knows about files & their logical blocks. Each file's logical blocks are numbered from 0 to N.

d) logical file System manages metadata info. Metadata includes all of the file System str except the actual data.

11 File organization :

* Refers to logical structuring of the records as determined by the way in which they are accessed.

* imp file organiza^n techniques are—

1) pile file org → data are collected in the order in which they arrive. The purpose of pile is simply to accumulate the mass of data & save it.

2) sequential file org → here, a fixed format is used for rec. The key field uniquely identifies the rec, thus key values for diff rec are always diff.

3) Indexed sequential file org → maintains the key characterstic of the sequential file, rec are organized in sequence based on a key field.

4) Indexed file org → used mostly in appli^n where timelines of info is critical & where data are rarely processed.

5) Direct / Hashed file org → direct file makes use of hashing on the key value. This () was explained in Appendix 8A. They used where rapid access is required.

III) File Allocation:

* on sec storage, a file consists of a collection of blocks.
* The OS is responsible for allocating blocks to files. This raises 2 managmnt issues.
  1) space on sec storage must be allocated to files
  2) It is necessary to keep track of the space available for allocation.
* with variable-size portions, we need to be concerned with fragmⁿ of free space, possible alternative strategies are first fit, best fit & nearst fit.
* 3 major methods for allocating sec storage space are-

1) Contiguous F.A:

- here each file occupies a contiguous set of blocks on the disk.
- directory entry for a file with C.F.A contains - add of starting block
  - length of allocated portion.
- Adv → both sequential & direct access are supted by this.
- dis → Kping file size is difficult

2) Linked list / chained non C.F.A:

- here, each file is a linked list of disk blocks which need not be contiguous.
- The disk blocks can be scattered anywhere on the disk.
- The directory entry contains a pointer to the starting & the ending file block.
- adv → very flexible in terms of size.

3) Indexed F.A:

- here, a spl block known as indx block contains the pointers to all the blocks occupied by a file.
- Each file has its own indx block.
- adv → It overcomes the prbm of ex fragmⁿ
- dis → pointer ovrhead for indexed A is > linked allocation.
- Allocation may be on the basis of either fixed-size blocks / variable-size portions.

IV) Free-Space Managmnt:

* It is a critical aspect of OS as it involves managing the available storage space on the hard disk / sec devices.
* To keep track of free-disk space,

the os maintains a free-space list.
* whn a file is dltd, its disk space is added to the free-space list.
* free-space list can be maintained by os by follow ways —

1) Bit vector:
- here, each block is represented by 1 bit.
- If the block is free, the bit is 1, if the block is allocated the bit is 0.
- A bit table has the adv that it is easy to find 1 a contiguous grp of free blocks.

2) Linked list:
- It is to link togthr all the free disk blocks, keeping a pointer to the first free block in a spl loc on the disk.
- This scheme is not efficient, to traverse the list, we must read each block.

3) Grping:
- Stores the addl of the free blocks in the 1st free block.
- 1st free block stores the addl of some, say n free blocks. out of these n blocks, 1st n-1 blocks are free & last block contains the addl of next free

4) Counting:
- Stores the addl of 1st free disk block & a no. n of free contiguous disk blocks that follow the 1st block.
- Every entry in the list contain —
  1) addl of 1st free disk block
  2) A no. n.

* adv → faster access to files
  → easy to implemnt
- dis → limited scalability.
  → ovrhead & risk of data loss.