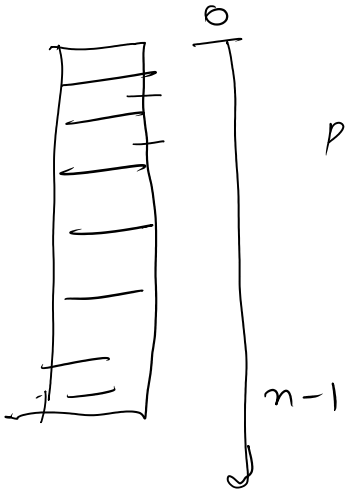
Memory Management

Monday, January 11, 2021 7:40 PM

Physical Memory — array of bytes



Physical addresses — Index

Memory Management – custom solution
allocation, de-allocation, re-allocation

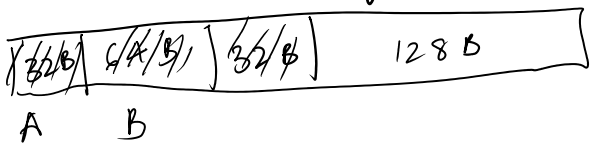
Why new & delete are not enough ?
- No control over programs
- No debugging
- no statistics

Debugging
Statistics
control

1) Memory Manager – allocates the memory & divides the memory into Blocks

BLOCK – small chunk of memory

256 Bytes

#g 

FREELIST – Linked list of Blocks → unallocated / free

Allocation – Block from freelist, allocated, in-use

De-allocation – Block is released back & added to the freelist

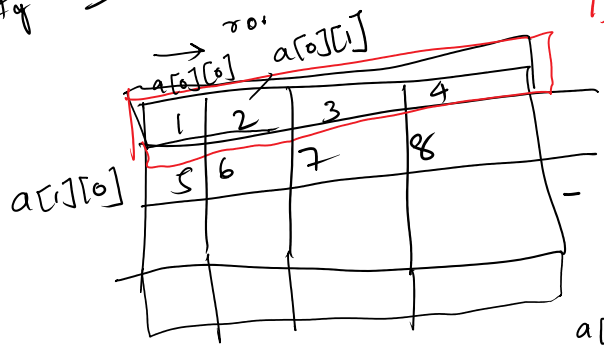
→ Attributes of Memory Manager

- Ease of use – frontend Users
  Garbage collection
- Performance – Locality of reference
  Speed vs consistency
  Allocation & Dellocation

Locality of Reference

Temporal
Same Data used multiple times
same book

Spatial
Data near / surrounding will be used again
Book in Politics
1, 2, 3, 4, 5, 6, 7, 8, 9, 10



faster

```
for(i=0; i<n)
  for(j=0, j<n)
    a[i][j]
```

// a[0][0], a[0][1], a[0][2]

Slower

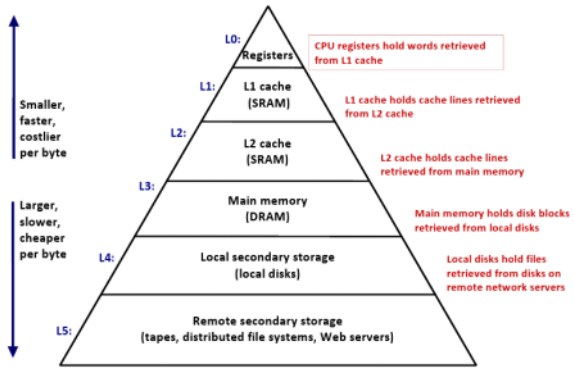
a[j][i]

a[0,0], a[1][0], a[2][0]

Print the contents of the array

Block of memory for the print

Speed vs consistency



Allocation Policies

I Sequential Fits

Roving pointer
Circular chain
Start from the last allocated mem location

First Fit - Start from beg., first empty block, large block will be split & assigned to Freelist

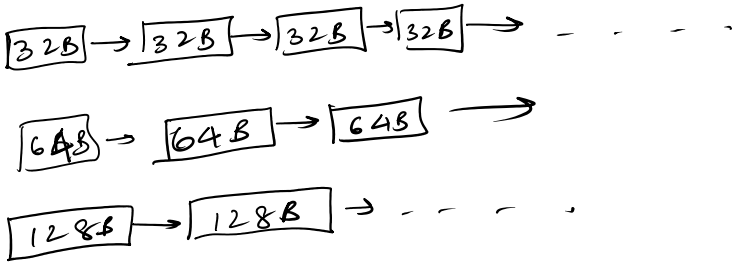
- Next Fit

Best Fit

↳ Best suitable memory block

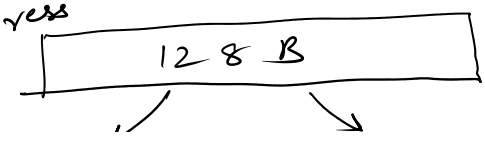
II Segregated Free List - Set of free list

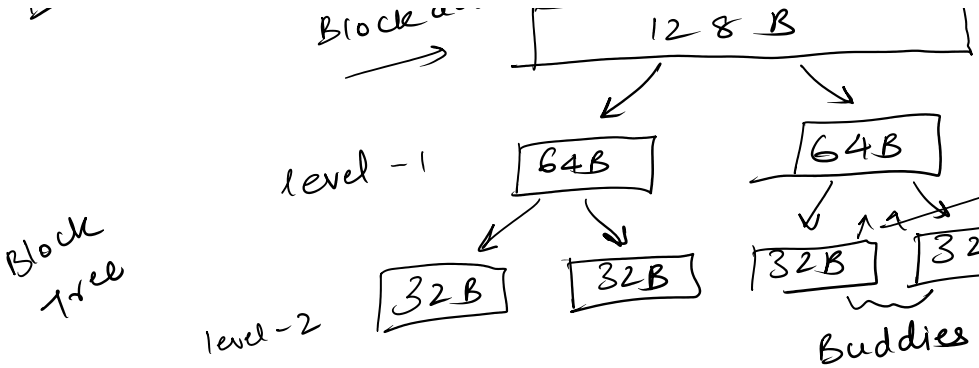
Each list holds blocks of a particular size



III Buddy System

Block address





Block tree

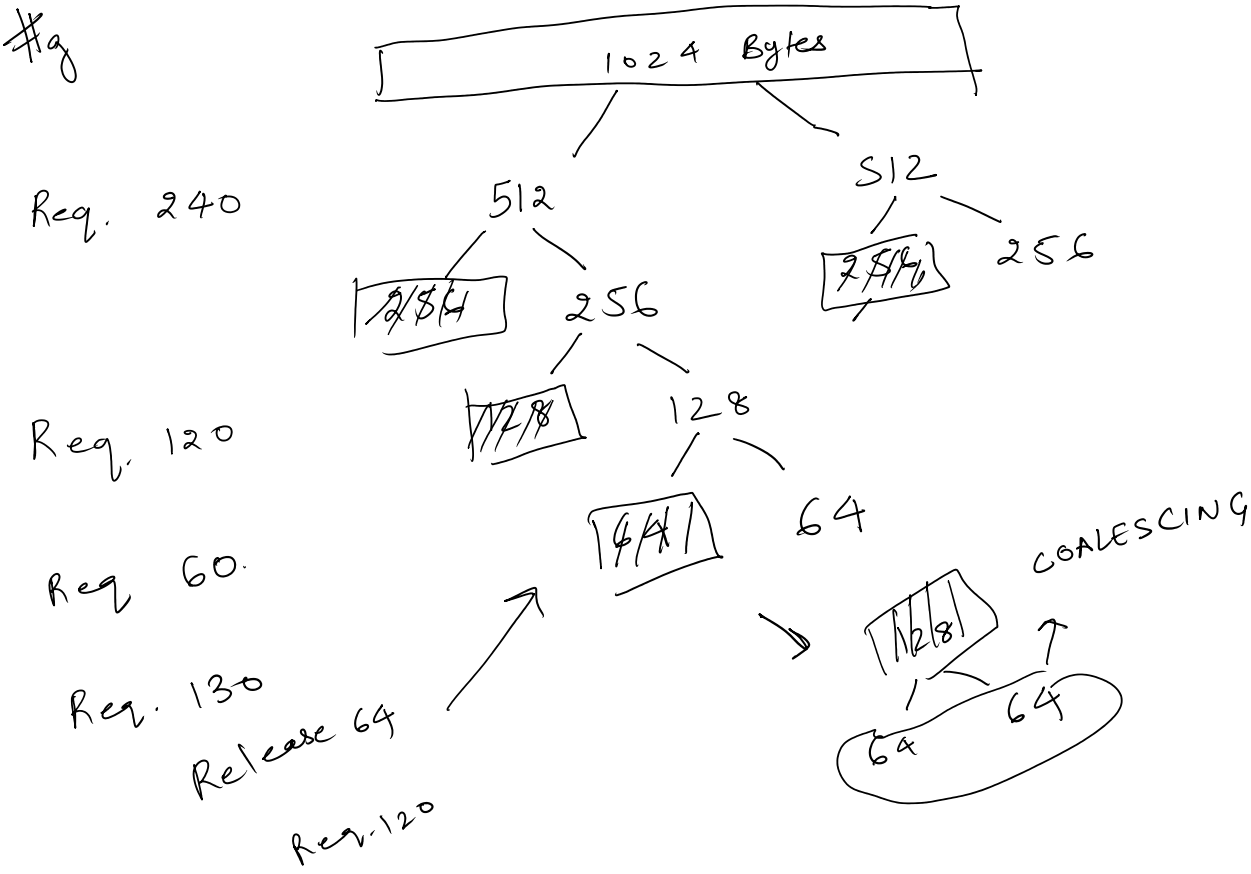
coalescing when a large block size is required, buddies can be merged together

$$\text{Buddy Address} = \text{Block Addr} + \frac{\text{Original size}}{2^{\text{level}}}$$

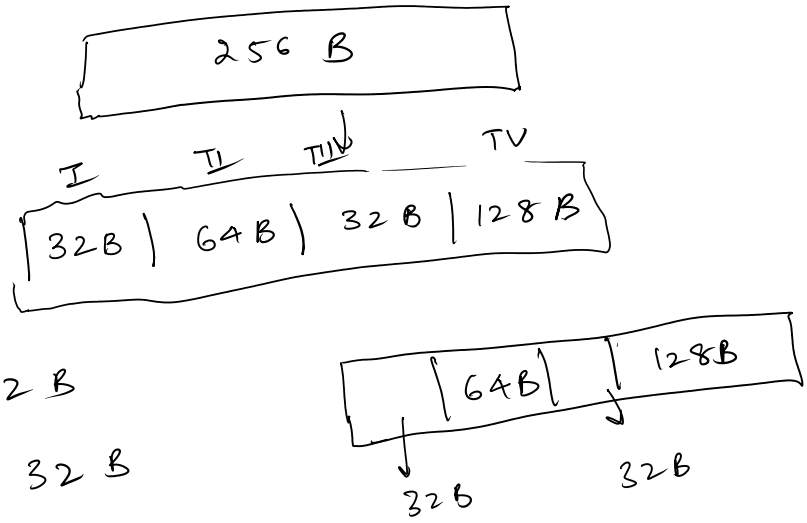
Each size is associated with a level

↓ memory.

#g



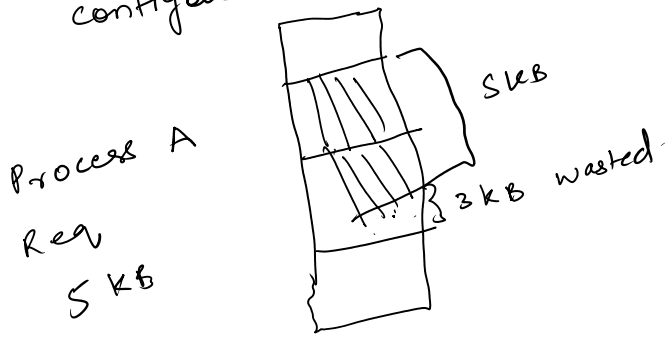
FRAGMENTATION



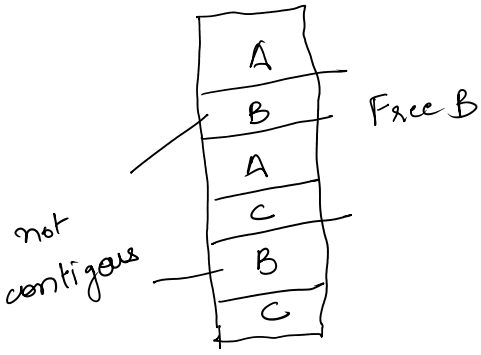
Req for 64 B → no contigous memory

Internal

contigous- each 4KB pages



External



Allocators — Stack
— Pool
— Fixed sized
— Variable size

Assignment 1

#g struct student {

unit age; // 4 bytes

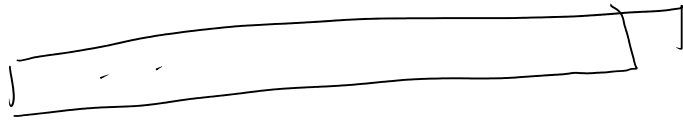
unit year;

int GPA;
int ID;
};

Student * s1 = new student()

Main Idea :—

1. Alloc a junk



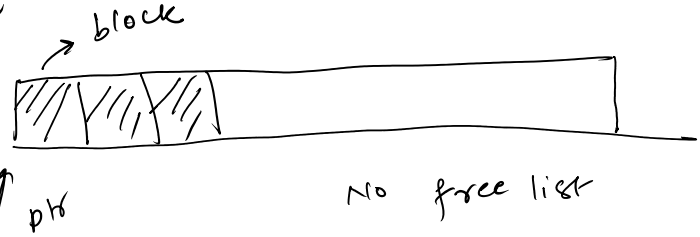
2. allocate() — return a block of memory.

3. free (obj) — put the block back

4. Reuse blocks.

Types of Allocators
- Linear
- Stack
- Pool
- FreeList

I Linear



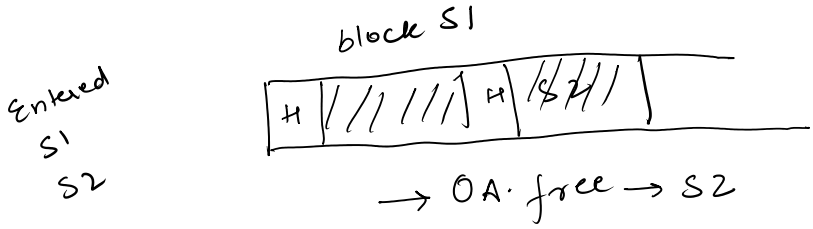
1. faster/simpler
2. low overhead memory
3. No single free()

II Stack — LIFO

1. Still faster

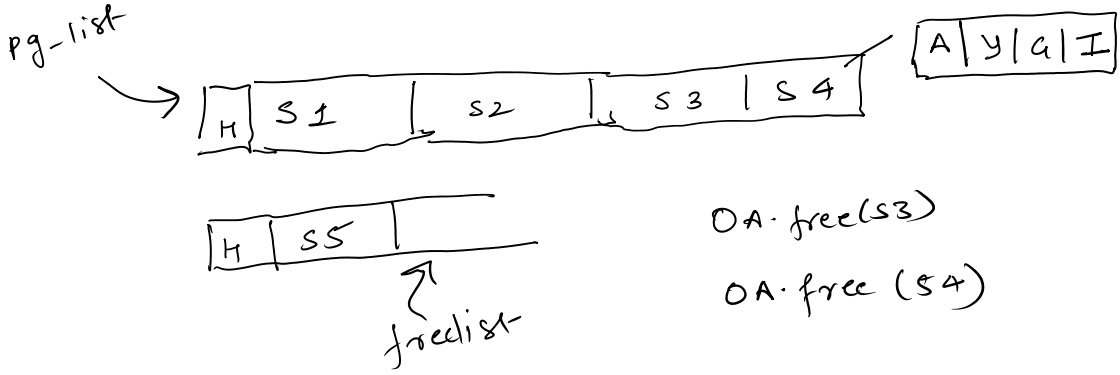
II Stack - LIFO

1. Still faster
2. Header + ptr
3. free LIFO()



III POOL - fixed size block

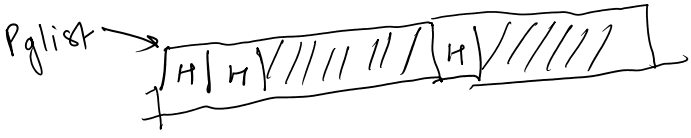
PageList, FreeList → LL



OA. free(S3)

OA. free (S4)

IV Free List - variable sized blocks



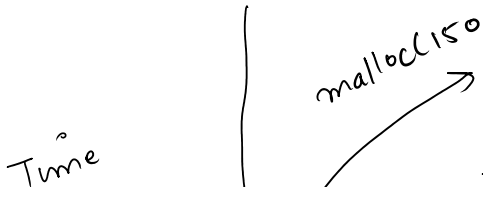
general purpose allocator

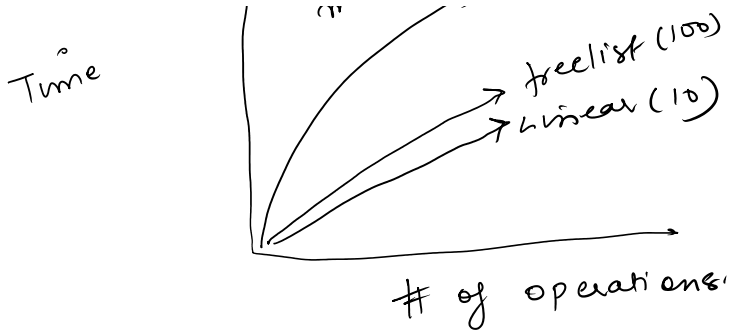
alloc/ free in any order

OA. allocate (size) → Allocation Policy — Next — Best — First

OA. free (obj) → merge empty blocks

gamesdev.net




Time
freelist (100)
Linear (10)
# of operations