

MEMORY MANAGEMENT

WHY MEM MGMT in DS?

- see "Motivation to DS" again

WHAT exactly is MEM MGMT?

- overriding C++ new/delete

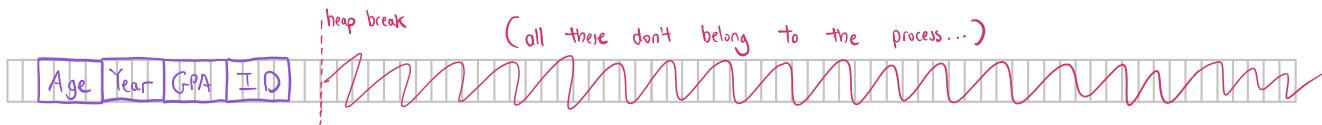
THE MS QNS:

WHY do we want to reinvent the ?????ing C++ new/delete ???

THE M  ANS:
consider the struct Student {

```
int Age;  
long Year;  
float GPA;  
long ID;
```

```
Student* s1 = new Student();
```



```
Student* s2 = new Student();
```

this will trigger...

① allocate() from proc heap

- find empty slot
- reduce fragmentation
- etc...

② OUT_OF_MEMORY_ERR

③ SYS_CALL_TRAP

only SYS has oversight across procs
(Context switch) USER MODE

KERNEL MODE

④ lock()

- prevent 2 procs alloc same mem

⑤ allocate() more mem to proc



⑥ BUT maybe unlucky... hit OUT_OF_RAM_ERROR

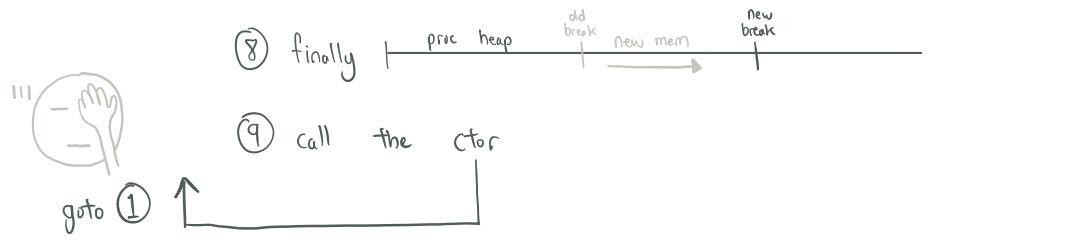


⑦ swap old/unused pages to HDD



} tl;dr

THIS is very BAD



this is WHY we have ...

CUSTOM ALLOCATORS

MAIN IDEA :

- ① alloc a "chunk" (a.k.a. pages) of mem at one go
`ObjectAllocator* oa = new ObjectAllocator(sizeof(Student), config);`

$$\left. \begin{array}{l} \text{Objectsize_ = 16 bytes} \\ \text{ObjectsPerPage_ = 4} \end{array} \right\} \begin{array}{l} \text{PageSize_ = } 4 * 16 + \text{sizeof (void*)} \\ \qquad\qquad\qquad = 68 \text{ bytes} \end{array}$$

- ② when user $\text{Student}^* \text{ pStudent} = \text{static_cast} < \text{Student}^* > (\text{oa} \rightarrow \text{Allocate}());$

③ when user $0a \rightarrow$ Free (pStudent)

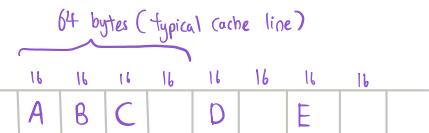
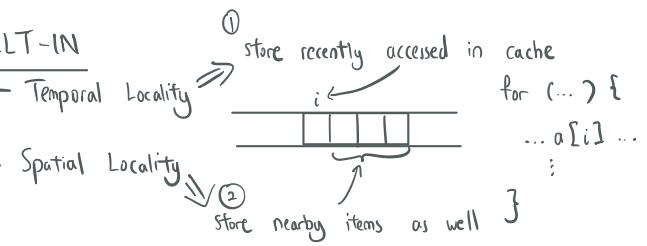
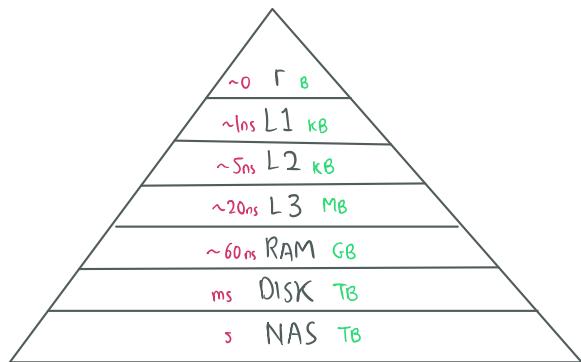
④ ... reuse blocks by adhering to ② and ③

Q: Show what happens to the ptrs in mem.

now let's look deeper at

WHY CUSTOM OAs CAN BE BETTER THAN BUILT-IN

① performance boost with LOCALITY OF REFERENCE



Q: Describe what happens when
do(A), do(C), do(A), do(B), do(D)

now consider this bmp operation ...

	0	1	2	3
0	a	b	c	d
1	e	f	g	h
2	i	j	k	l
3	m	n	o	p

// $\text{bmp}[1][2] = g$

```
Pixel pixel = bmp[4][4]; // sizeof(Pixel)=16, cacheline = 64
for (auto i=0; i<4; ++i)
    for (auto j=0; j<4; ++j)
        clear(bmp[i][j]); // 1 vs clear(bmp[j][i]); // 2
```

Q: Does it matter if we choose 1 or 2?



② customized to specific app needs



- fixed entity types
- max objs in scene

ALLOC PATTERNS

- same sequence between levels
- fixed durations

③ MEM DEBUGGING

- count allocs / deallocs (detect leaks)
- track alloc sizes
- time between allocs
- etc...

So what are some common

CUSTOM ALLOCATOR TYPES ?



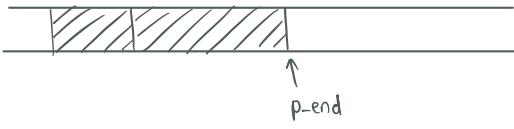
LINEAR ALLOCATOR

oa → Allocate (size);

oa → FreeAll();

Q: Show what happens in mem after the above stnts are ran...

(state of curr mem)



✓ fast

✓ insignificant space overhead

✗ no individual frees

} Suitable for e.g. glx screen buffer
- clrScr() every frame

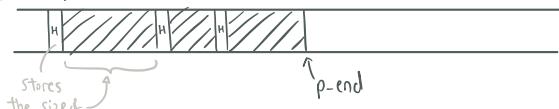
STACK ALLOCATOR

oa → Allocate (size);

oa → Free ();

Q: Show what happens in mem after the above stnts are ran...

(state of curr mem)



✓ still fast

✗ slight space overhead

✓ can individually Free() BUT only LIFO

} Suitable for e.g. hierarchical game levels
- 02 → 02.1 → 02.2

POOL ALLOCATOR

after alloc "chunk", pre-split into blocks, ie. populate the free list

Student* p.s1 = static_cast<Student*>(oa → Allocate());

Student* p.s2 = static_cast<Student*>(oa → Allocate());

oa → Free(p.s1); // return to free list for reuse

Q: Show what happens in mem after the above stnts are ran...



✓ still fast

✓ fixed-size blocks

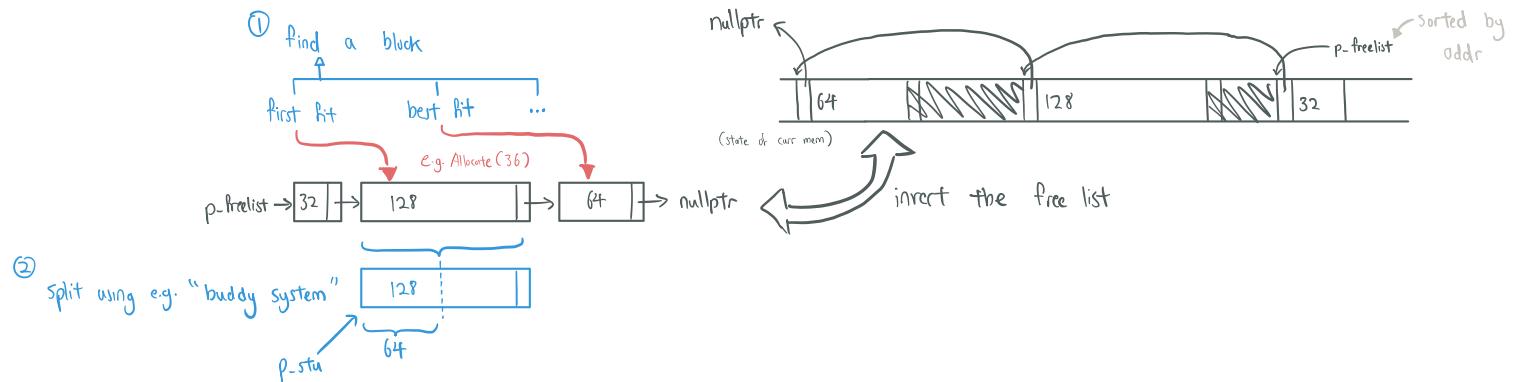
✗ some space overhead

} Suitable for e.g. game entities
- roughly same size
- frequent arbitrary alloc/frees

FREE LIST ALLOCATOR (a.k.a. pooling with dynamic sizing)

(ignoring the cuts...)

$p_stu = \text{oa} \rightarrow \text{Allocate}(\text{size})$;



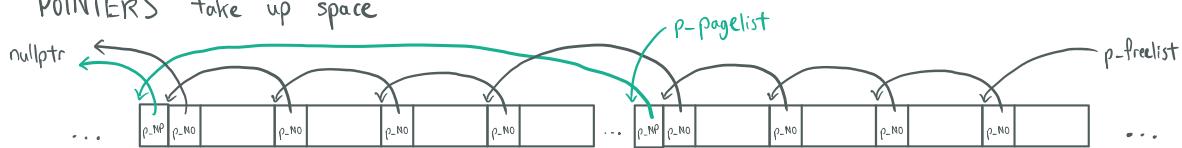
oa → Free(p_stu) ① link back to free list (in order of address)
② merge if contiguous (coalescence)

✓ general purpose	Suitable for ANYTHING - BUT always consider more specialized list - we can also note that as it gets more generic, speed decreases.
✗ not as fast	
(still faster than malloc)	

visit [github.com / mtrebi / memory-allocators](https://github.com/mtrebi/memory-allocators) to see some benchmarks of the above

MEM ALLOCATOR OVERHEAD

POINTERS take up space



STATISTICS take up space

OAStats → Freeobjects_ ObjectsInuse_ PageInuse_ MostObjects_ Allocations_ Deallocations_

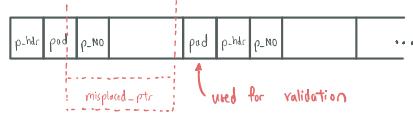
DEBUGGING

- e.g. if ($\text{Allocations}_- == \text{Deallocations}_-$) noLeaks = true;

IMPROVE OA DESIGN

- e.g. if ($\text{MostObjects}_- > \text{threshold}$) $\text{++ ObjectsPerPage}_-$;

and then padding (for validation)



and then there're HEADERS...



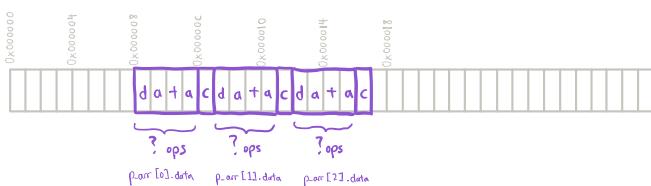
ALIGNMENT also takes up space

e.g. struct Intcy {
 int data; //4
 char c; //1
}

Intcy* p-arr = new Intcy[3];

Q: Show the above array in mem below... (assume 32-bit archi)

Q: How many atomic operations to access each element of the array?



Q: WHAT is the sol? to this problem?

lastly, a note about FRAGMENTATION

2 types

INTERNAL

e.g. a fixed-size pool allocator
of 16-byte blocks

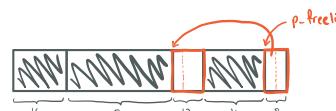


p1 = static_cast<sixteenByte*>(oa->Allocate());
p2 = static_cast<sixteenByte*>(oa->Allocate());
p3 = static_cast<fourByte*>(oa->Allocate());
p4 = static_cast<sixteenByte*>(oa->Allocate());

(fragment is within allocated mem)

EXTERNAL

e.g. a freelist allocator



what if now we want

p-0 = static_cast<sixteenByte*>(oa->Allocate());
(fragment is in free mem)