CS280 – Data Structures

Assignment 1: Object Allocator

Recap

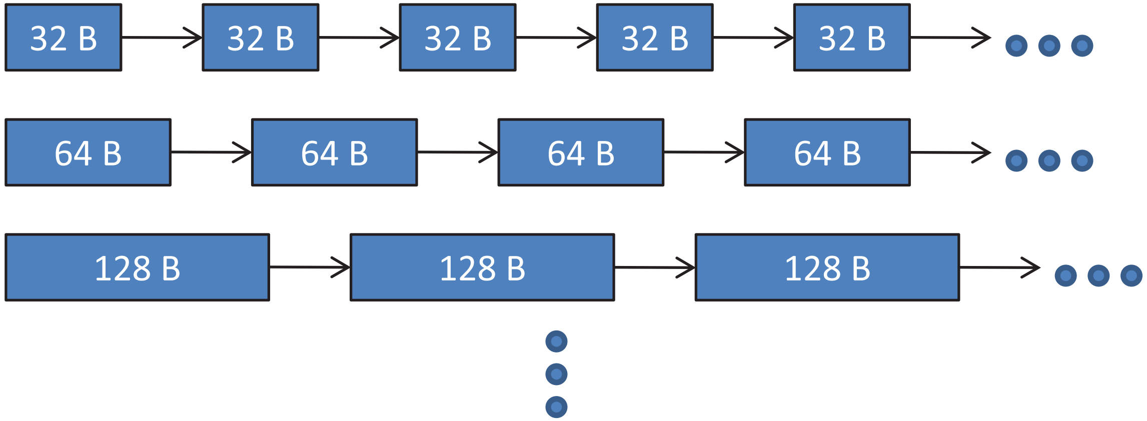
- What is memory management?
- Why memory management?
- Automatic memory management
- Fragmentation
- Allocation techniques
- Alignment

Recap

- Recall different allocation policies
  - Sequential fits: first fit, next fit, best fit, etc.
  - Segregated free lists
  - Buddy systems

Segregated Free Lists

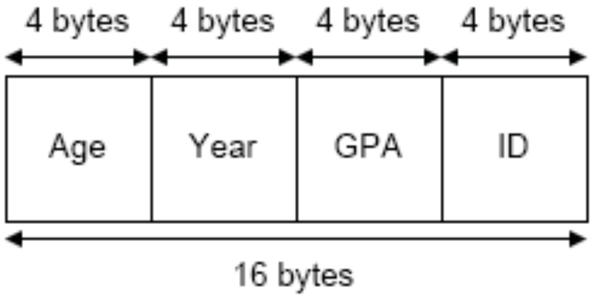
- The allocator maintains a set of free lists where each list holds free blocks of a *particular* size.
- Can group each object according to its size and assign it to a particular list



Page Allocation

```
struct Student{
  int Age;
  long Year;
  float GPA;
  long ID;
};
```

`sizeof(Student)`?



Page Allocation

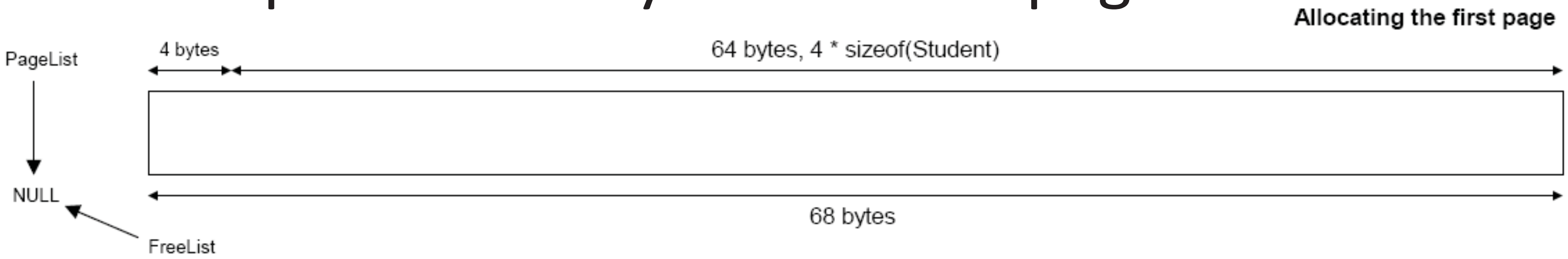
- Suppose we set
  - the maximum number of pages to 2
  - the maximum number of objects per page to 4…
- Let’s keep two pointers
  - PageList
  - FreeList

```
studentObjectMgr = new
  ObjectAllocator(sizeof(Student), config);
```

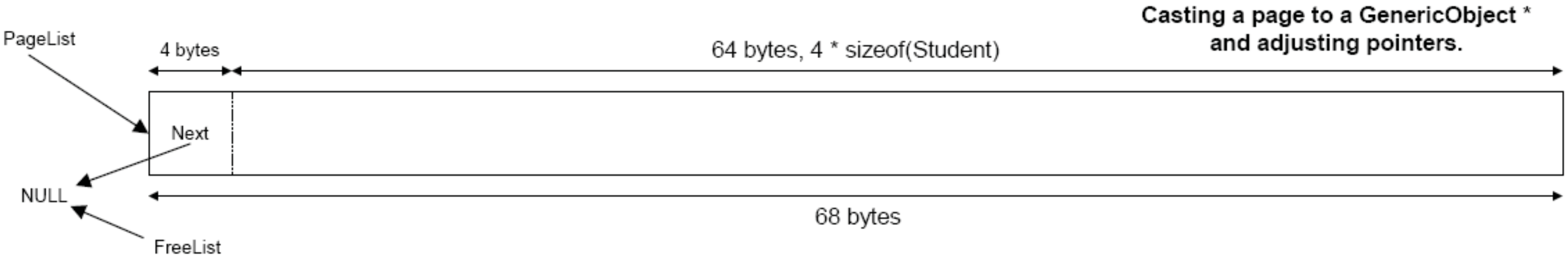
Steps for Page Allocation

Steps for Page Allocation

1. Request memory for the first page.

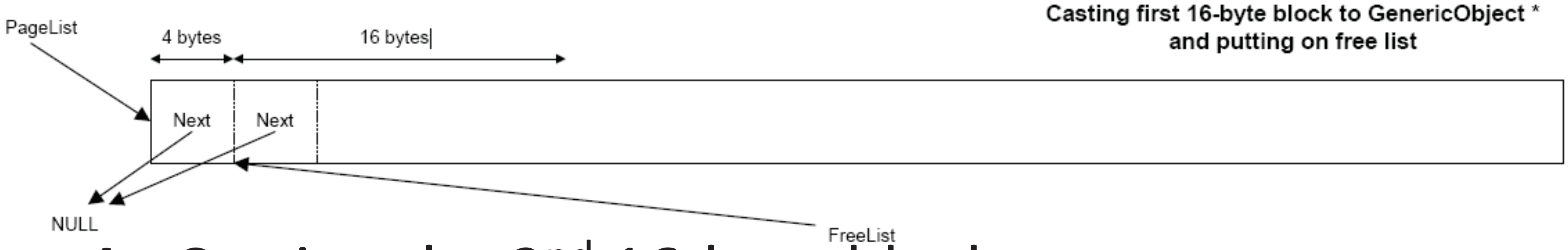


2. Casting the page to a `GenericObject*` and adjusting pointers.

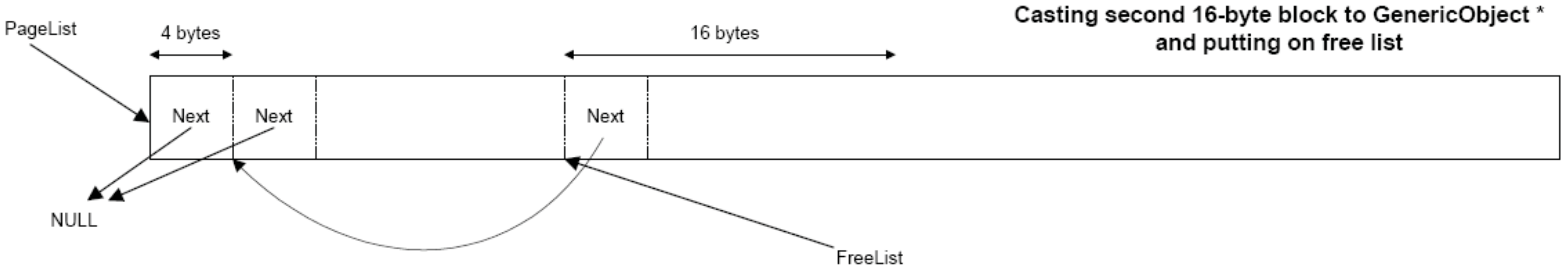

Casting a page to a GenericObject * and adjusting pointers.
PageList
4 bytes
64 bytes, 4 * sizeof(Student)
Next
NULL
68 bytes
FreeList

Steps for Page Allocation

3. Casting the 1st 16-byte block to `GenericObject*` and putting on free list

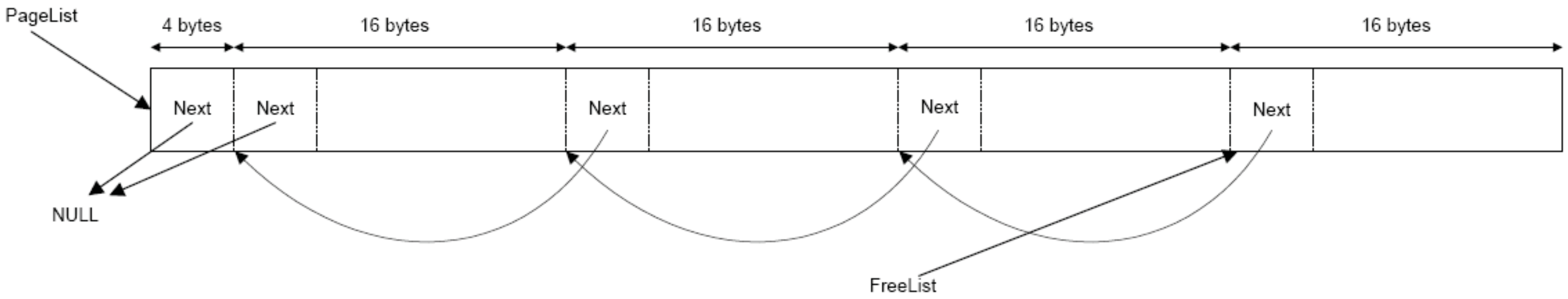


4. Casting the 2nd 16-byte block to `GenericObject*` and putting on free list



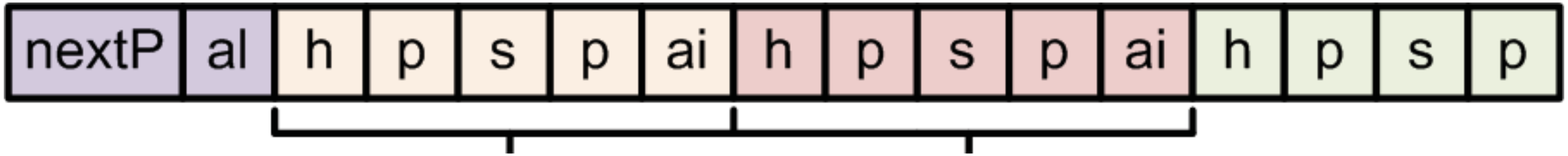
Steps for Page Allocation

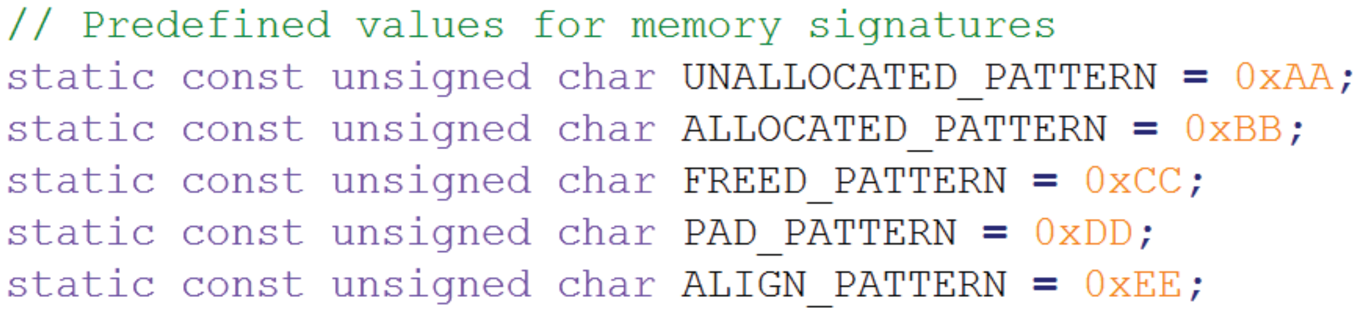
5. Do the same for the 3rd and 4th blocks



How to compute the page size?

- Let number of objects per page=3
- Let s be object’s size in bytes
- Let p be number of pad bytes
- Let h be size of the head block in bytes
- Let al be left alignment
- Let ai be inter alignment
- Let ps be the page size we are interested
- Page size = sizeof(nextP) +al+3*size(mid block)-ai





```
// Predefined values for memory signatures
static const unsigned char UNALLOCATED_PATTERN = 0xAA;
static const unsigned char ALLOCATED_PATTERN = 0xBB;
static const unsigned char FREED_PATTERN = 0xCC;
static const unsigned char PAD_PATTERN = 0xDD;
static const unsigned char ALIGN_PATTERN = 0xEE;
```

Configurations

Recall

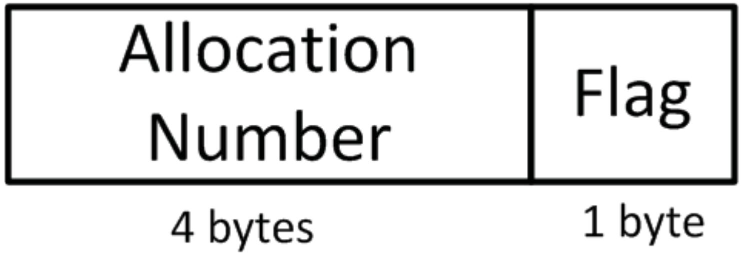
```
studentObjectMgr = new
ObjectAllocator(sizeof(Student),
config);
```

OAConfig

- Header block
  - Basic
  - Extended
  - External
- Padding
- Alignment

Header Blocks

Basic Header Block



Allocation Number | Flag

4 bytes | 1 byte

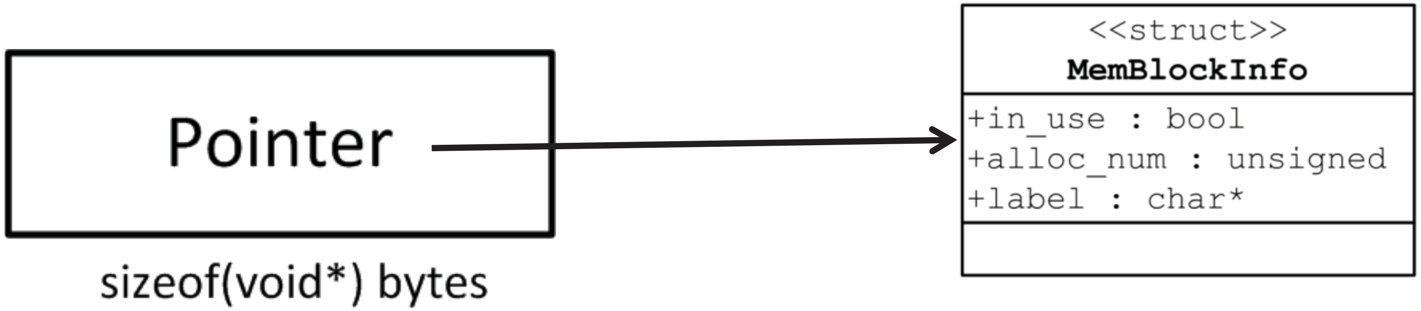
- Allocation Number: increments each time the allocator successfully returns a block of a memory.
- Flag: the right-most bit indicates whether the block is free (0000000**0**) or not (0000000**1**).

Extended Header Block

User-defined	Use-count	Allocation Number	Flag
x bytes	2 bytes	4 bytes	1 byte

- User-defined: user-defined field of x bytes.
- Use-count: count how many times this block has been used.

External Header Block

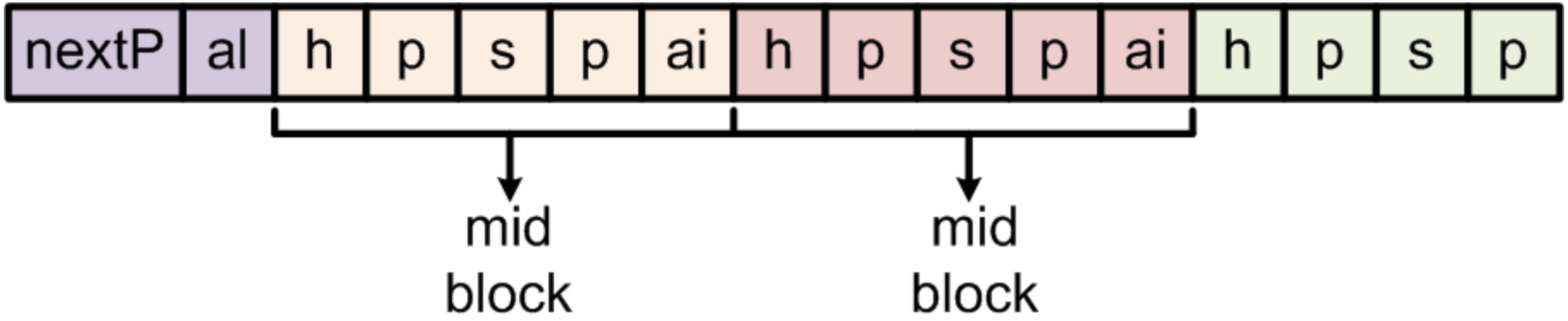


- A pointer to a chunk of memory outside of the block itself.
- More flexible and easier to be extended

```
void *Allocate(const char *label = 0) throw(OAException);
```

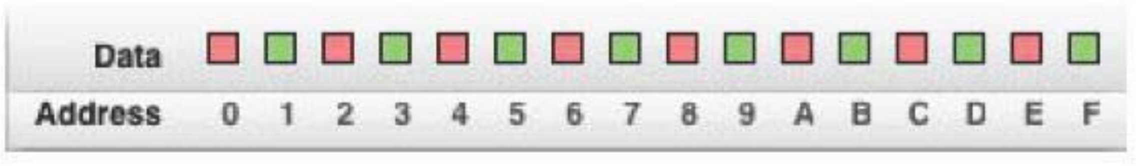
Padding

- Padding is used for validation

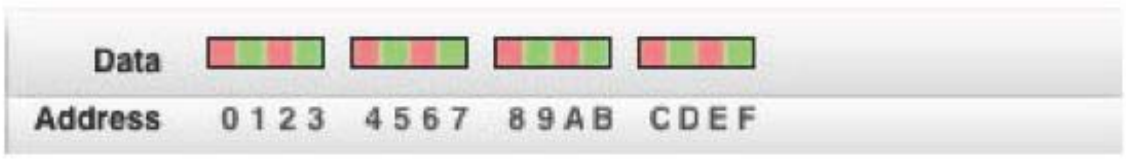


Alignment

- How programmers see memory?

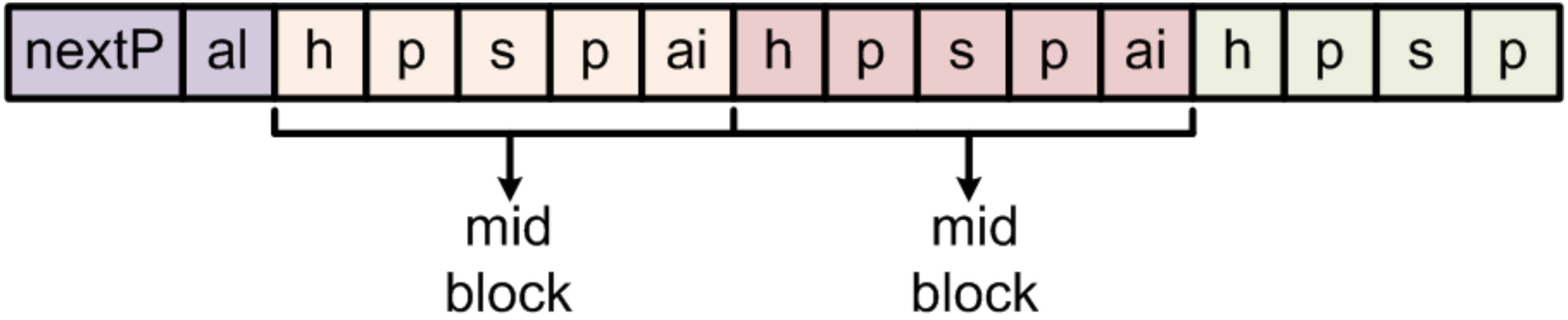


- How processors see memory? (in a 32-bit machine)



Alignment

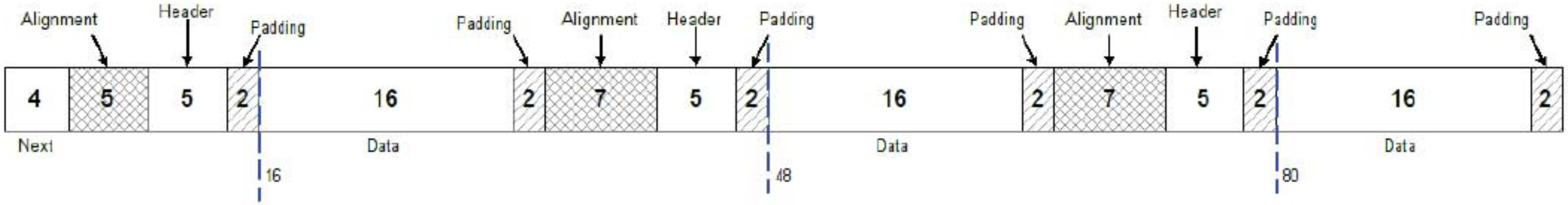
- Each pointer (nextP, nextO) must be aligned
- Two computations required
  - – Left alignment
  - – Internal alignment



Alignment

Example 4: 16-byte data, 2-byte padding (left/right), basic header blocks (5 bytes), 16-byte alignment.

Field	Size
Next pointer	4 bytes
Padding	2 bytes
Basic header block	5 bytes
Data	16 bytes
Alignment	5/7 bytes (left/interblock)
Page size	98 bytes



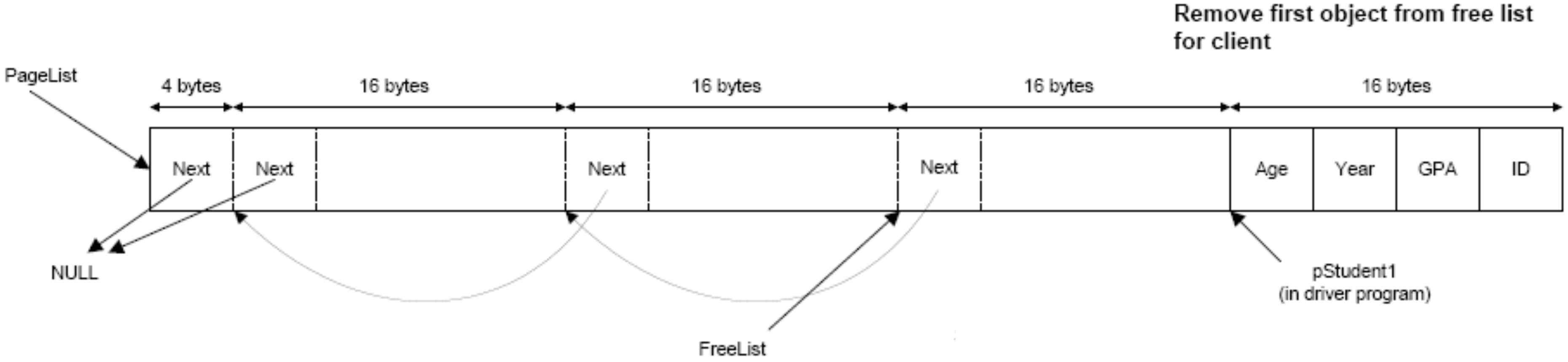
Allocation/Deallocation

Allocation/Deallocation

- Client's allocation request:

```
Student *pStudent1 =
reinterpret_cast<Student
*>(studentObjectManager->Allocate());
```

- Remove first object from free list for client:

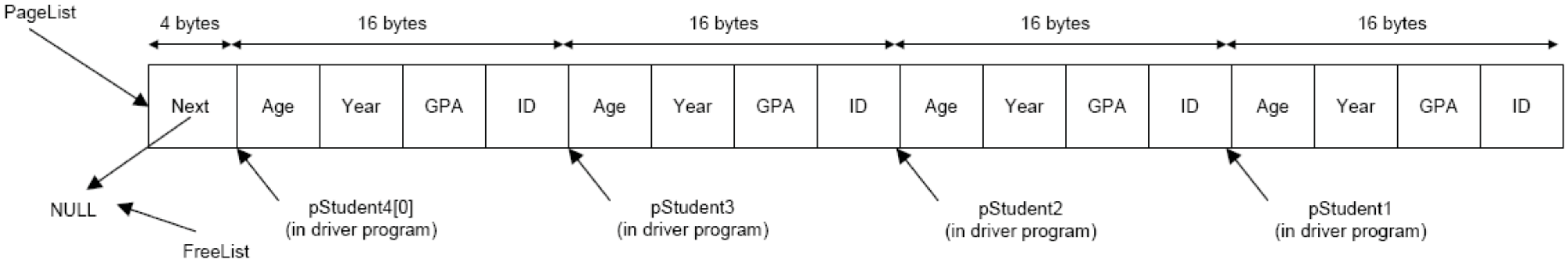


Allocation/Deallocation

- Update the accounting information
  - Total number of allocations
  - Total number of objects in use by the clients
  - Total number of free objects
  - etc.
- Update the header block information
  - Allocation number
  - Flag
  - Use-count
  - etc.
- etc.

Allocation/Deallocation

- After another few allocations:



PageList

4 bytes | 16 bytes | 16 bytes | 16 bytes | 16 bytes

Next | Age | Year | GPA | ID | Age | Year | GPA | ID | Age | Year | GPA | ID | Age | Year | GPA | ID

NULL

FreeList

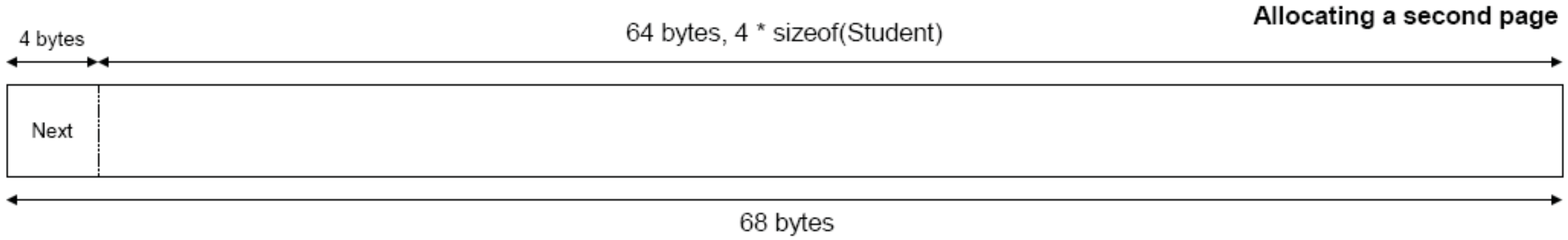
pStudent4[0]
(in driver program)

pStudent3
(in driver program)

pStudent2
(in driver program)

pStudent1
(in driver program)

- Need to create another page for more allocation requests:



Allocating a second page

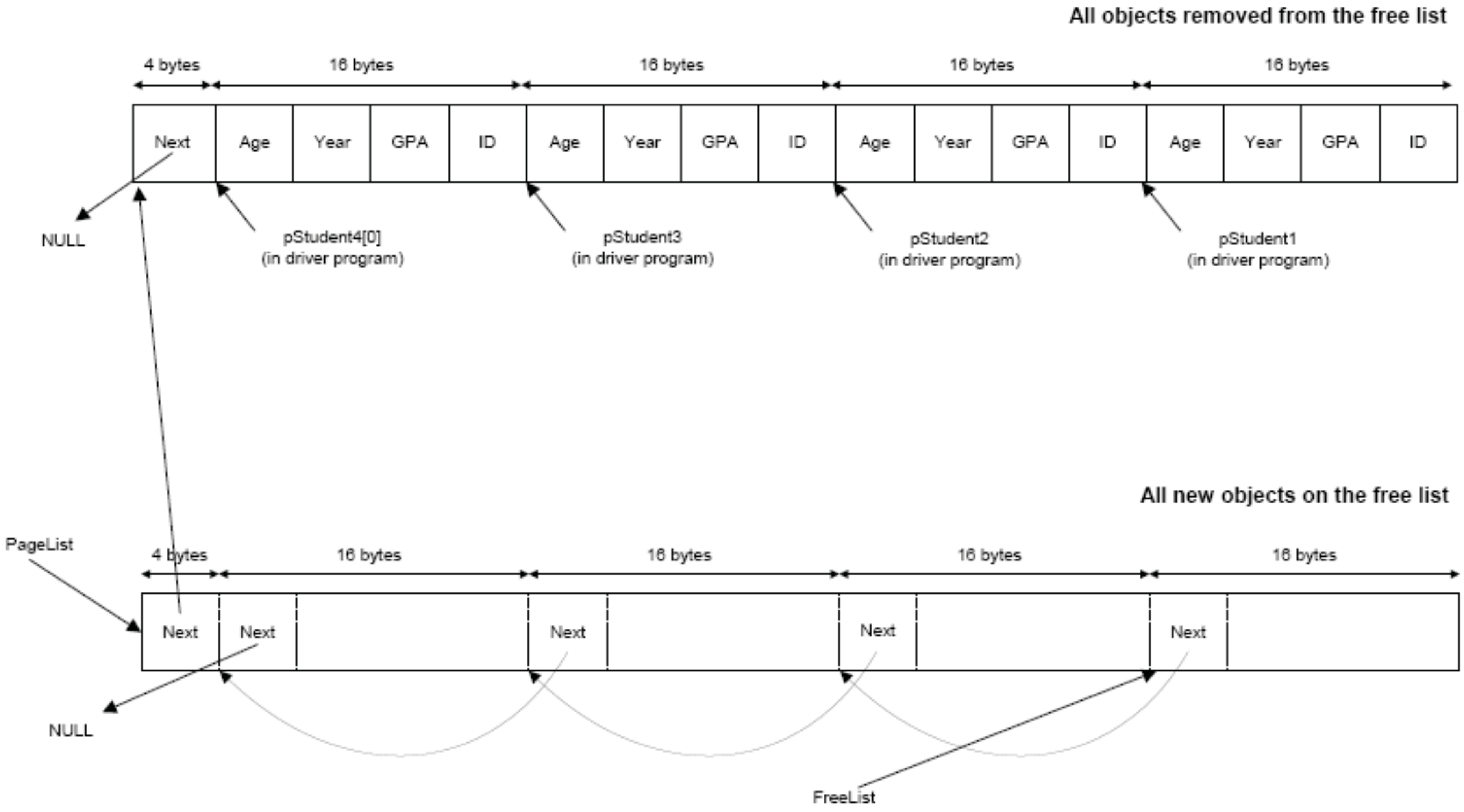
4 bytes

64 bytes, 4 * sizeof(Student)

Next

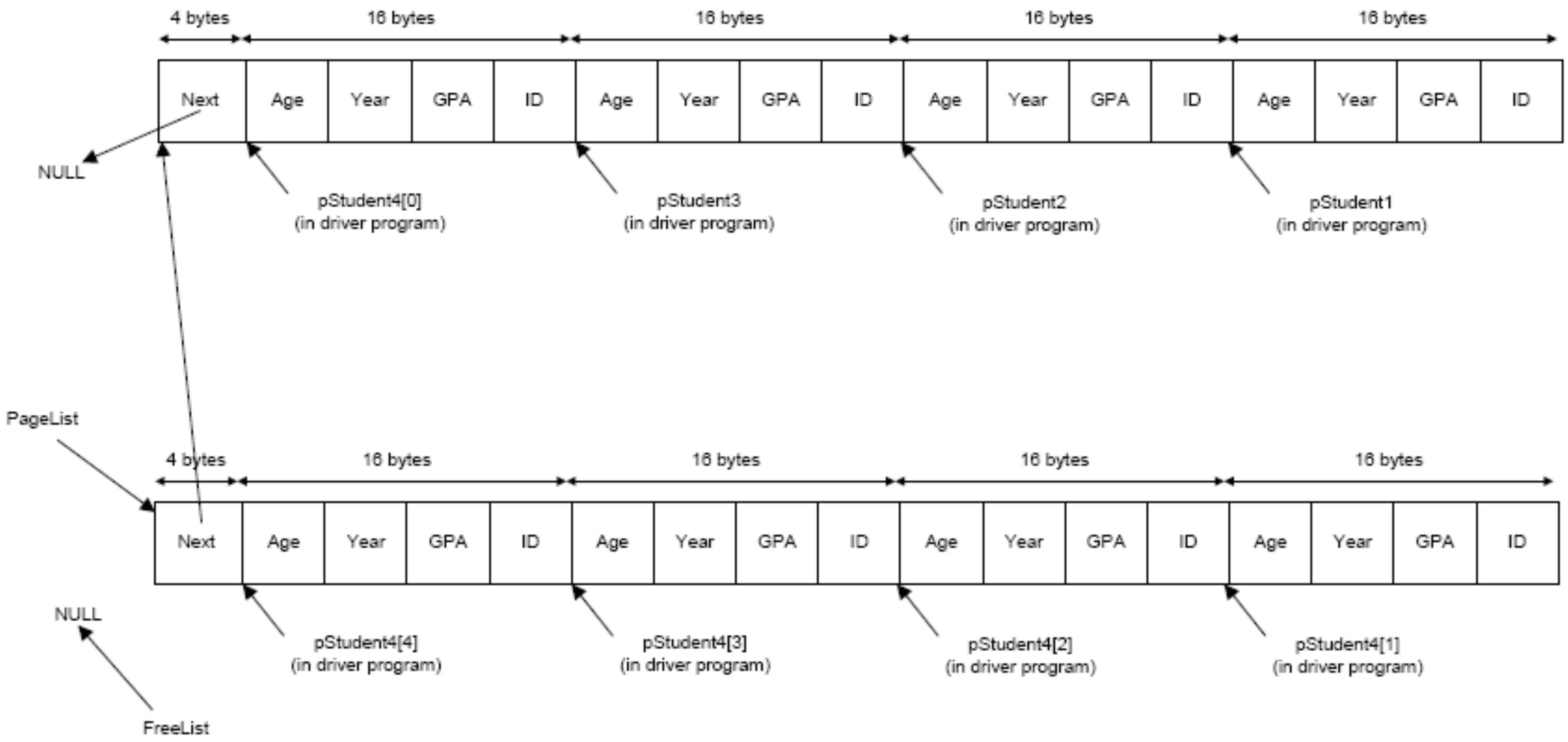
68 bytes

Allocation/Deallocation



Allocation/Deallocation

Both pages have supplied objects to the client. No more objects on the free list.



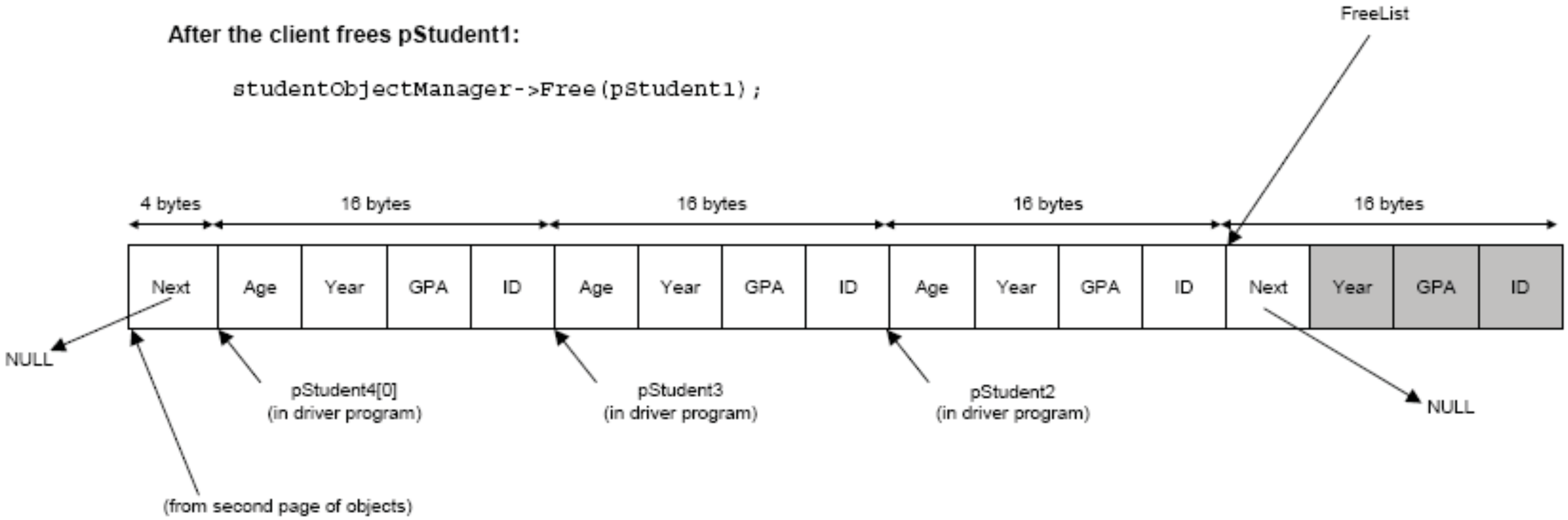
Allocation/Deallocation

- Client's deallocation request:

```
studentObjectManager->Free(pStudent1)
```

After the client frees pStudent1:

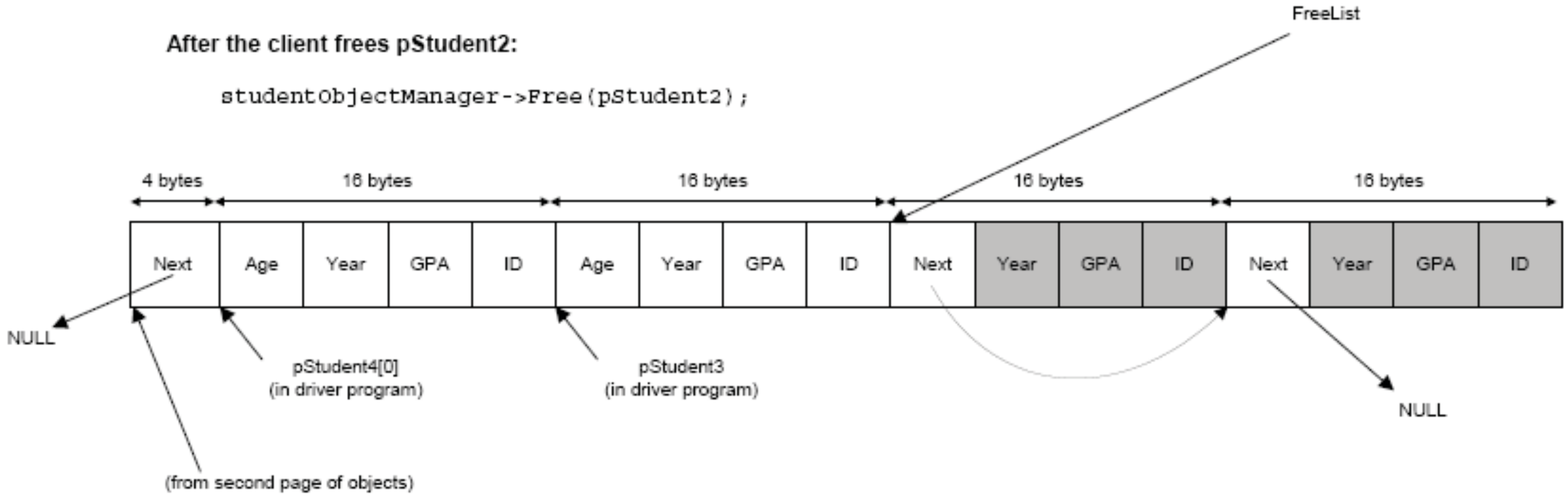
```
studentObjectManager->Free(pStudent1);
```



Allocation/Deallocation

- Check for “double” free
  - Compare the object with all objects in the current freelist
- Check for “bad boundary”
  - The object to be deleted is not in the range of all pages
  - Validate address: check if the object is aligned
- Validate padding
- Add the object to free list
- etc.

Allocation/Deallocation

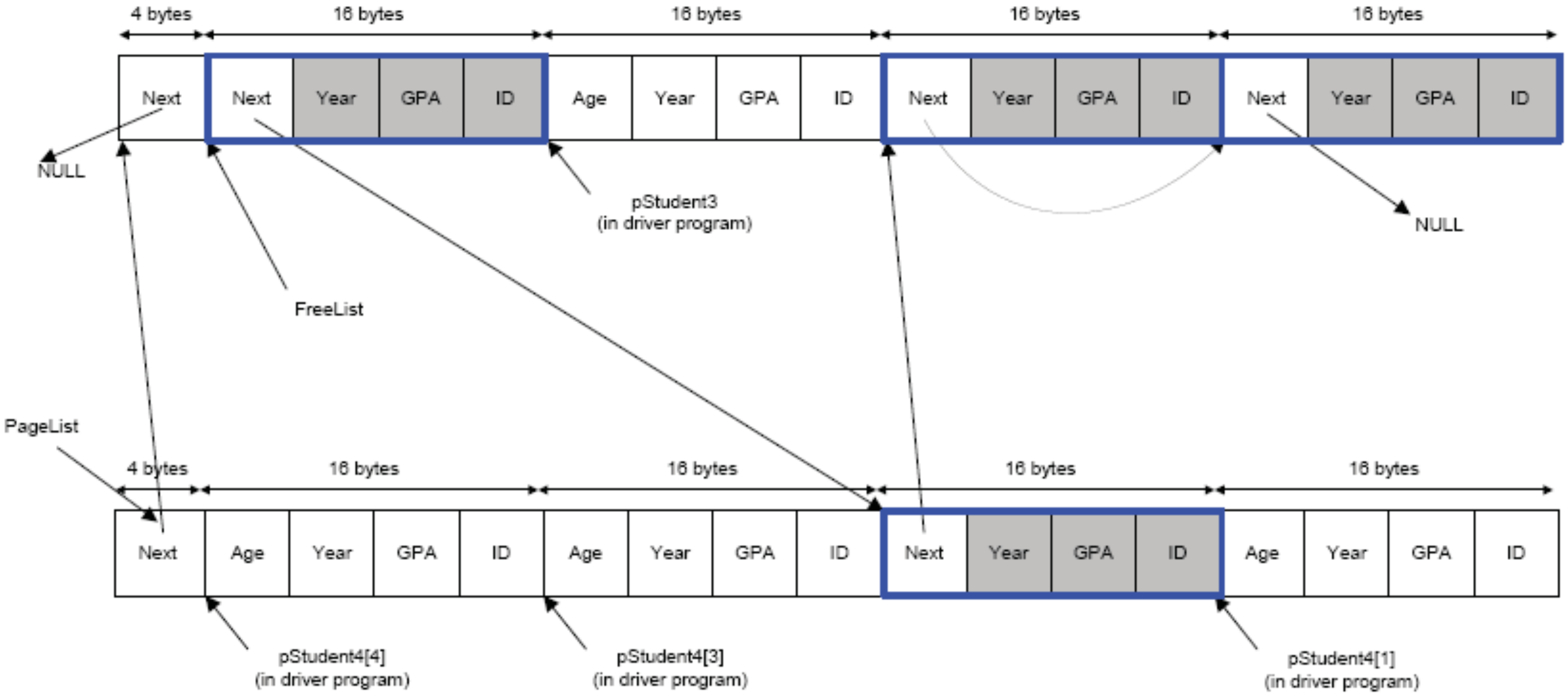


After the client frees pStudent2:

```
studentObjectManager->Free(pStudent2);
```

```
studentObjectManager->Free(pStudent1);
studentObjectManager->Free(pStudent2);
studentObjectManager->Free(pStudent4[2]);
studentObjectManager->Free(pStudent4[0]);
```

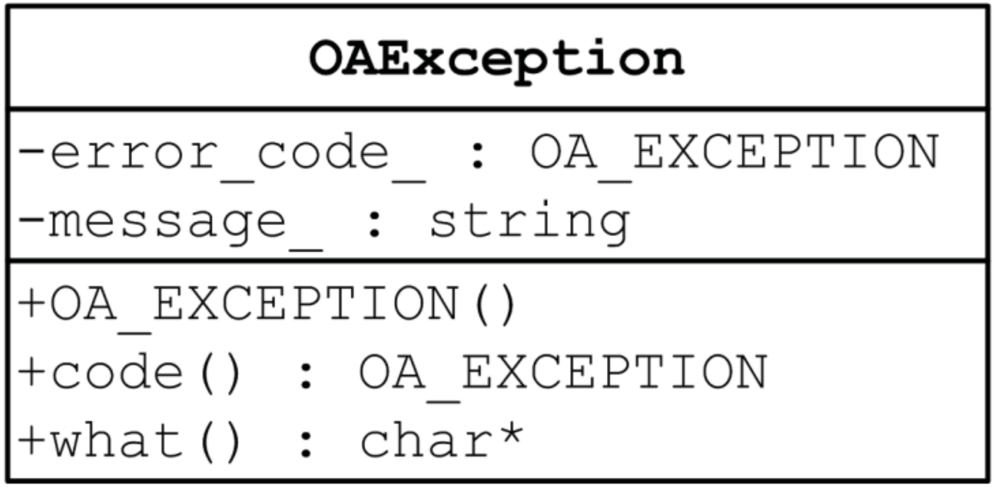
Allocation/Deallocation



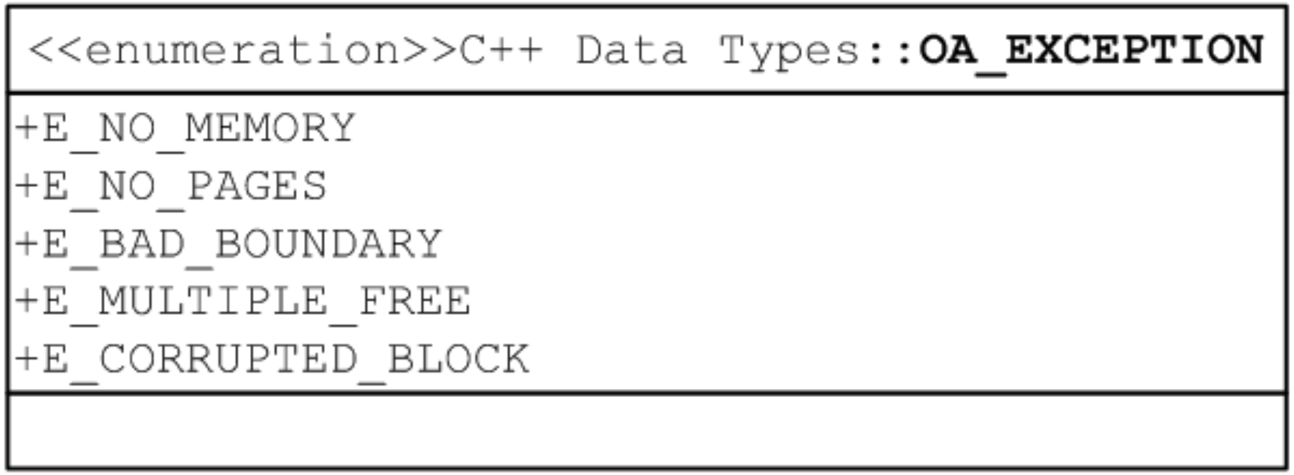
FreeEmptyPages

- Remove pages, which are not used by the clients.
- Returns the number of empty pages

Exceptions

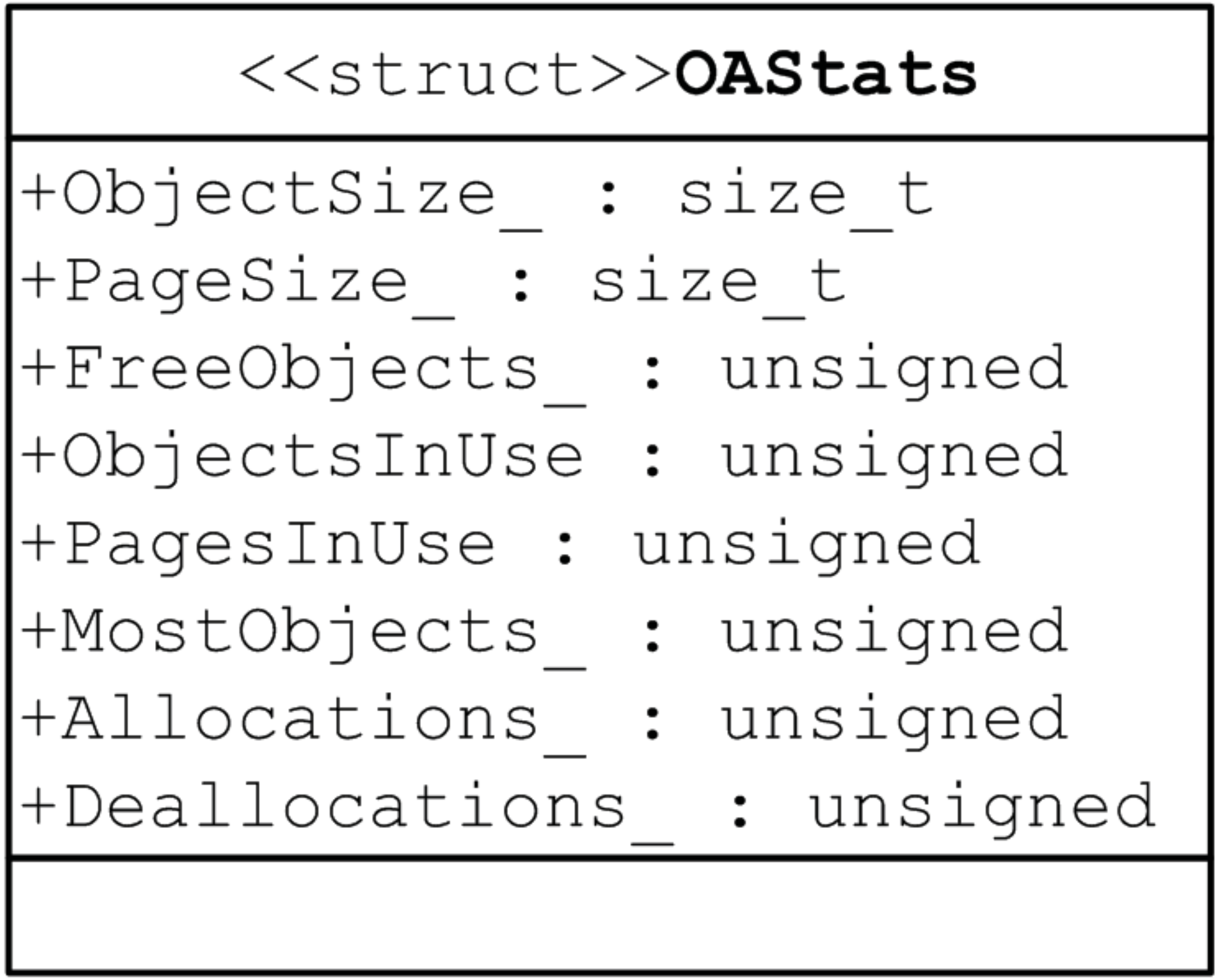


```
OAException
-error_code_ : OA_EXCEPTION
-message_ : string
+OA_EXCEPTION()
+code() : OA_EXCEPTION
+what() : char*
```



```
<<enumeration>>C++ Data Types::OA_EXCEPTION
+E_NO_MEMORY
+E_NO_PAGES
+E_BAD_BOUNDARY
+E_MULTIPLE_FREE
+E_CORRUPTED_BLOCK
```

Statistics



```
<<struct>>OAStats
+ObjectSize_ : size_t
+PageSize_ : size_t
+FreeObjects_ : unsigned
+ObjectsInUse : unsigned
+PagesInUse : unsigned
+MostObjects_ : unsigned
+Allocations_ : unsigned
+Deallocations_ : unsigned
```