Nginx memory management

Memory management in Nginx:

1. In Nginx memory management is implemented using memory pools.
2. A pool is a sequence of pre allocated memory blocks.
3. Memory pool is a one-way linked list of fixed size memory block.
4. A pool is tied to an object. This object can be a request or event.

Why to use pools?

- To minimize the overhead of the allocation of small objects.
- To reduce the number of malloc(),calloc() system calls. Using the malloc and free cause several problems: the demand of web server memory is small, system operation for a long time leads to memory fragmentation, causing memory operations more time until there is no memory available, causing the server to go down.
- To minimize memory leaks. When the connection is closed all memory is released and doesn't leave behind any memory.
- To reduce fragmentation.
- To improve memory access : In the memory pool the memory address are aligned and there is a memory paging mechanism that improve the CPU page hits and efficient memory access.

The memory management in Nginx is implemented using the following files:

- src/core/ngx_palloc.{h,c}
- src/os/unix/ngx_alloc.{h,c}

What is there in "src/os/unix/ngx_alloc.{h,c}" ?

These files are operating system related files which encapsulated the basic memory allocation functions like malloc(), calloc() and free().
This encapsulation just give a new identity to these functions nothing else.

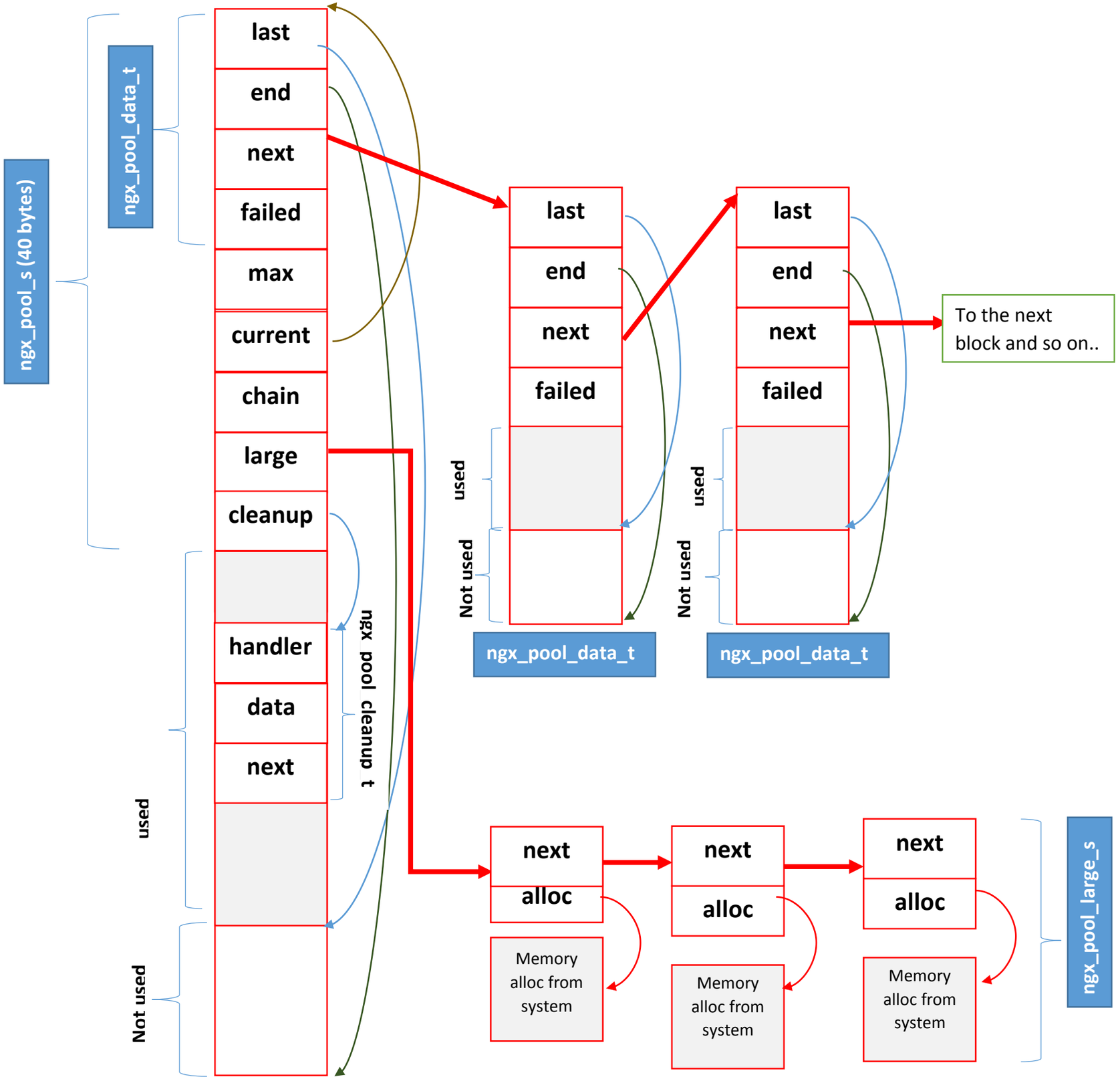
- free() function is encapsulated as ngx_free()
- malloc() function is encapsulated as ngx_alloc()
- calloc() function is encapsulated as ngx_calloc()
- posix_memalign() function is encapsulated as ngx_memalign()

What is there in "src/core/ngx_palloc.{h,c}" ?

These are the two files which actually implement memory management in Nginx.
ngx_palloc.c -> Provides the functionality to create, delete, resize, reset memory pools.
ngx_palloc.h -> Defines the pool structure and the associated function definitions.

Major Pool Data Structures :

In nginx a memory pool is implemented using two basic structures ngx_pool_data_t and ngx_pool_s. They are used for maintaining data portion and header portion of the memory pool.



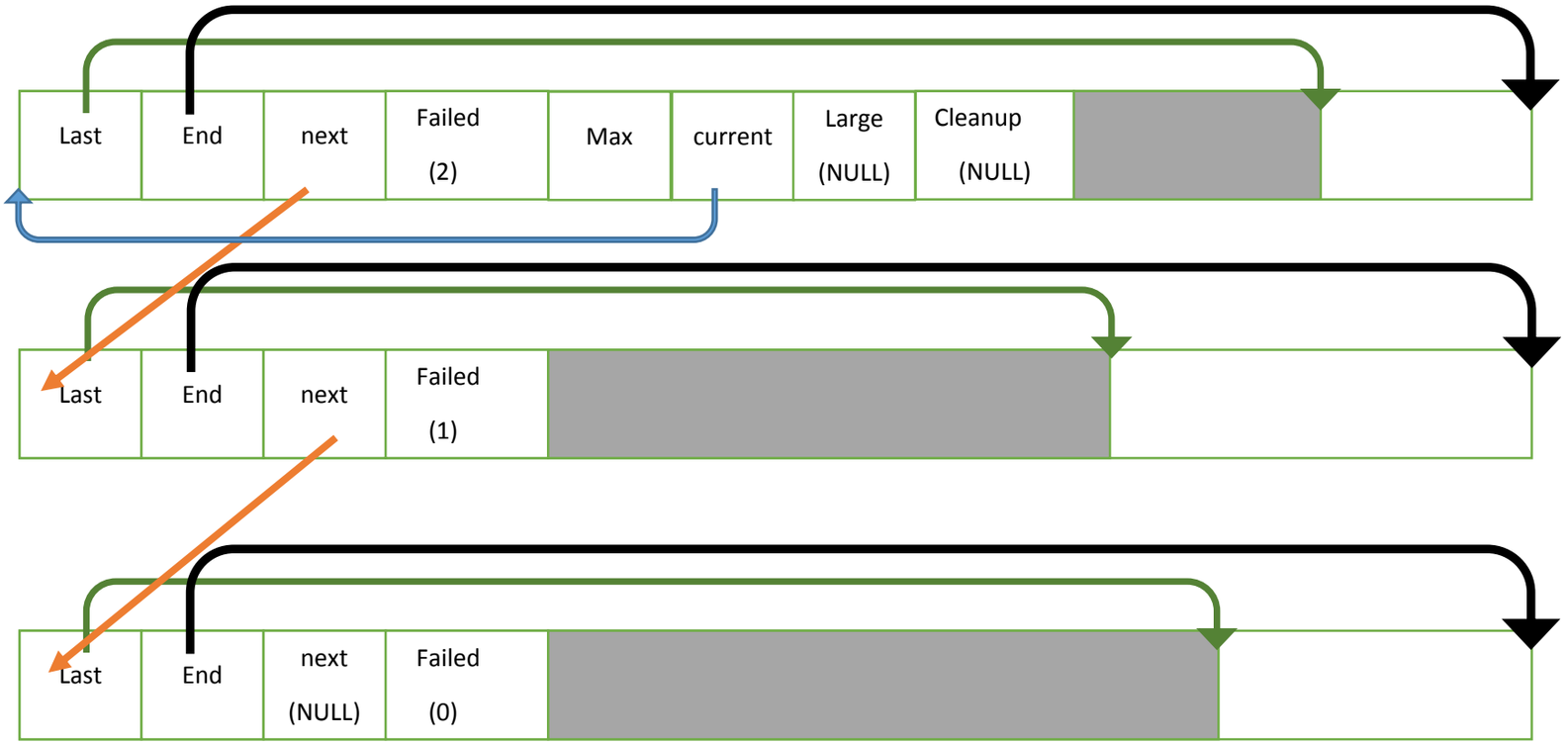
Types of Memory Allocation :

Nginx uses the memory paging mechanism to improve the number of page hits thus leading to efficient and faster access to memory. Thus Nginx sets an upper bound to the maximum amount of memory that can be allocated using the pool. If the memory is within this limited size then the allocation takes place inside the pool and this is called small memory allocation. If the memory to be allocated is more than size then memory is allocated outside the pool in the main memory using normal malloc or alloc() functions and this allocation is called large memory allocation.
Small memory allocation is handled by ngx_pool_data_t while large memory allocation is handled by ngx_pool_large_t.
There are different types of memory allocations that can take place in nginx depending on the size of the memory asked for. The types are:

1. Small Memory Allocation (size <= max)

   - one way linked list maintained by structure ngx_pool_data_t

   Distrubution of small memory :



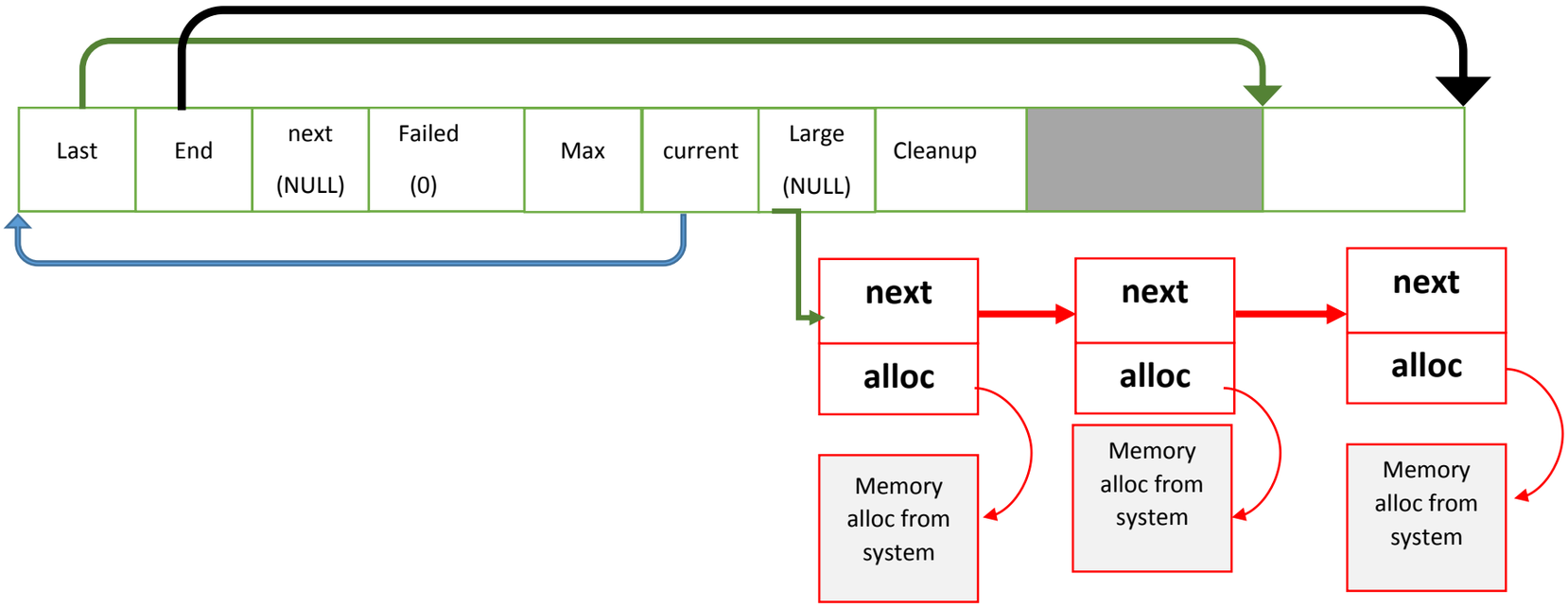
Brief Explanation of distribution of small memory :

We specify the size of the pool while allocating it.In the above example if the pool has enough free memory available then the new data block will be allocated in the current pool itself but if it doesn't have enough memory then a new small memory data element is created to satisfy the requirements.

Failed represents that the current memory pool's available memory can't meet the user allocation request, that is to say: an allocation request arrives, but the memory pool does not have enough free memory, then failed will increase 1; the allocation request will be handed over to the pool's next to deal with, if the memory pool next also cannot be satisfied, then its failed will add 1, will then be asked to continue down by transferring, until it meets the requirements.

2. Large Memory Allocation: (When size>max)

- Responsible for the bulk allocation of data memory pool (size> max) management.
- one way linked list maintained but structure ngx_pool_large_s



This is the large memory allocation.Each large memory allocation is linked to the pool via the large pointer.Each memory allocation has a next and alloc pointer.The next pointer points to the next node in the list while alloc points to the allocated memory.

3. Abstract Memory data (E.g.: cache data or file data)

- one way linked list maintained but structure ngx_pool_chain_t

Various Structures implementing Memory pools:

ngx_pool_data_t

This structure is the memory pool data block. Responsible for small memory allocation.
It contain pointers to maintain data block allocated by the user.
Code definition in source code :

```
typedef struct{
        u_char          *last;
        u_char          *end;
        ngx_pool_t  *next;
        ngx_unit_t   failed;
}ngx_pool_data_t;
```

From the structure we can observe that :

- last is a pointer to unsigned char which holds the last used address in the pool and next assignment starts from here if needed.
- end is a pointer to unsigned char which points to the end of the pool. It is different from last which points the last allocated address and not the end address of the pool.
- In memory pool there are many blocks which are connected in form of a linked list. The next pointer is a pointer to the structure which points to the next memory block in this list.
- failed will store information about failures which occurred during memory pool allocation.

==========

ngx_pool_s

ngx_pool_s is the memory pool head structure. The header information for the entire pool is stored in this structure. In turn it contains ngx_pool_data_t (data block) and other things like size, current block etc.

```
struct ngx_pool_s {                                  /* Memory pool Data block */
    ngx_pool_data_t     d;
        size_t                      max;
        ngx_pool_t              *current;
        ngx_chain_t            *chain;
        ngx_pool_large_t  *large;
}
```

From the structure definition we can see that:

- d is the memory pool data block as explained above
- max is size of data block
- current points to current pool of memory
- chain is the pointer to connect a ngx_chain_t structure
- Point to large memory allocation i.e when size>max

==========

ngx_pool_t

The memory pool structure ngx_pool_s is encapsulated as ngx_pool_t.This means that ngx_pool_t will be referred while create, destroying memory pool. Thus memory pool is implemented using ngx_pool_t.
This encapsulation of ngx_pool_s as ngx_pool_t is defined in the file src/core/ngx_core.h as:

```
typedef struct ngx_pool_s    ngx_pool_t ;
```

==========

ngx_pool_large_t:

Responsible for the bulk allocation or large allocation of data memory pool (i.e when size> max) management.

Source code :

```
typedef struct ngx_pool_large_s  ngx_pool_large_t;
struct ngx_pool_large_s {
    ngx_pool_large_t     *next;
    void                 *alloc;
};
```

==========

ngx_pool_cleanup_t:

Responsible for memory pool data recovery.This structure consists of a handler which tell how to release the resources.This makes the memory pool very flexible.Using this structure the ngx_pool_t can not only manage memory but can also manage resources for e.g. closing or opening of files etc. This resource management is implemented using this structure.

Source code :

```
typedef void (*ngx_pool_cleanup_pt)(void *data);
typedef struct ngx_pool_cleanup_s  ngx_pool_cleanup_t;
struct ngx_pool_cleanup_s {
   ngx_pool_cleanup_pt   handler; // Function pointer
   void                 *data; // The actual data
   ngx_pool_cleanup_t   *next;
};
```

Here :
Handler is a function pointer. It points to a custom function which tells ngx_pool_t how to release the resources.
Data is the pointer to the data to be handled.
Next is the pointer to next node in the linked list structure.