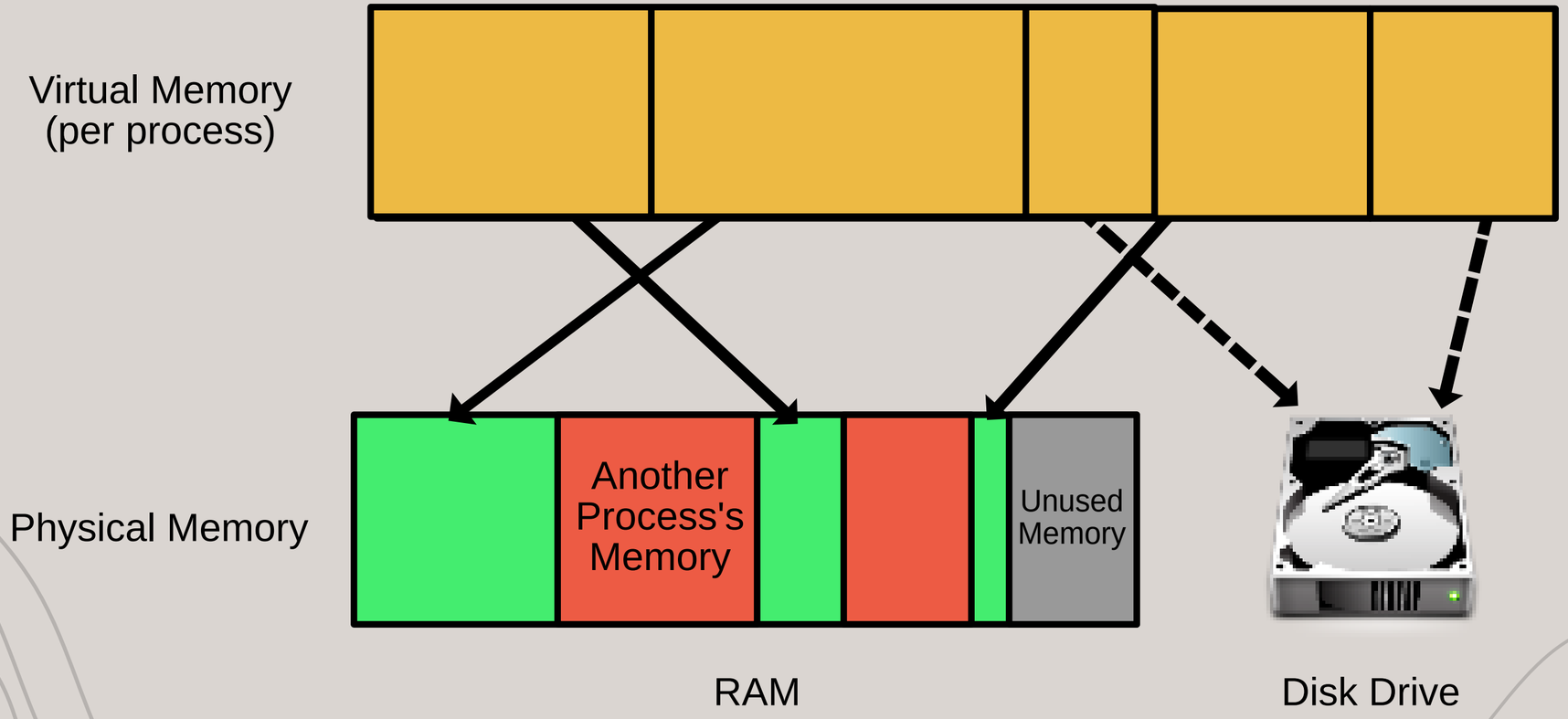
M’ Allocators

Pramish Aryal

A Refresher on Virtual Memory

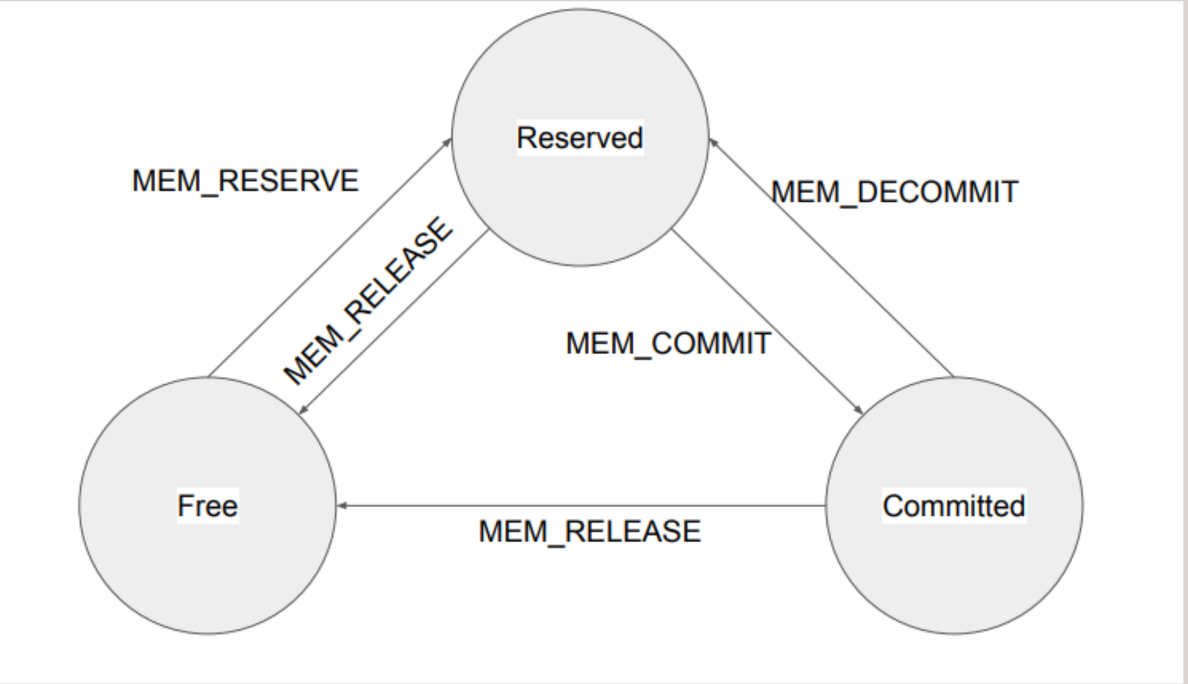


Allocation & Deallocation

- Allocation: Requesting the OS for memory using various syscalls.
- OS specific functions:
  - mmap (linux)
  - VirtualAlloc (windows)
- Deallocation: Returning the memory back to the OS.
- OS specific functions:
  - munmap (linux)
  - VirtualFree (windows)

Stages of allocation and deallocation

- Reserving (MEM_RESERVE)
- Committing (MEM_COMMIT)
- Decommitting (MEM_DECOMMIT)
- Releasing (MEM_RELEASE)



Allocation in Windows

```
LPVOID VirtualAlloc (
  [in, optional] LPVOID lpAddress,
  [in]           SIZE_T dwSize,
  [in]           DWORD flAllocationType,
  [in]           DWORD  flProtect
);
```

```
BOOL VirtualFree (
  [in] LPVOID lpAddress,
  [in] SIZE_T dwSize,
  [in] DWORD dwFreeType
);
```

Remember Alignment?

```
uint64_t align_up_power_2 ( uint64_t size, uint64_t align ) {
            return ( size + ( align - 1 ) ) & ~(align-1);
}
```

The above functions “aligns” the value of size to the power of 2 given by align

- align_up_power_2(23, 8) = 24
- align_up_power_2(27, 8) = 32
- align_up_power_2(16, 8) = 16

Onto the actual topic

Allocators

But before that

Lifetime and Size categorization of memory

Allocation Categories	**Size Known**	**Size Unknown**
Lifetime known	95%	~4%
Lifetime Unknown	~1%	< 1%

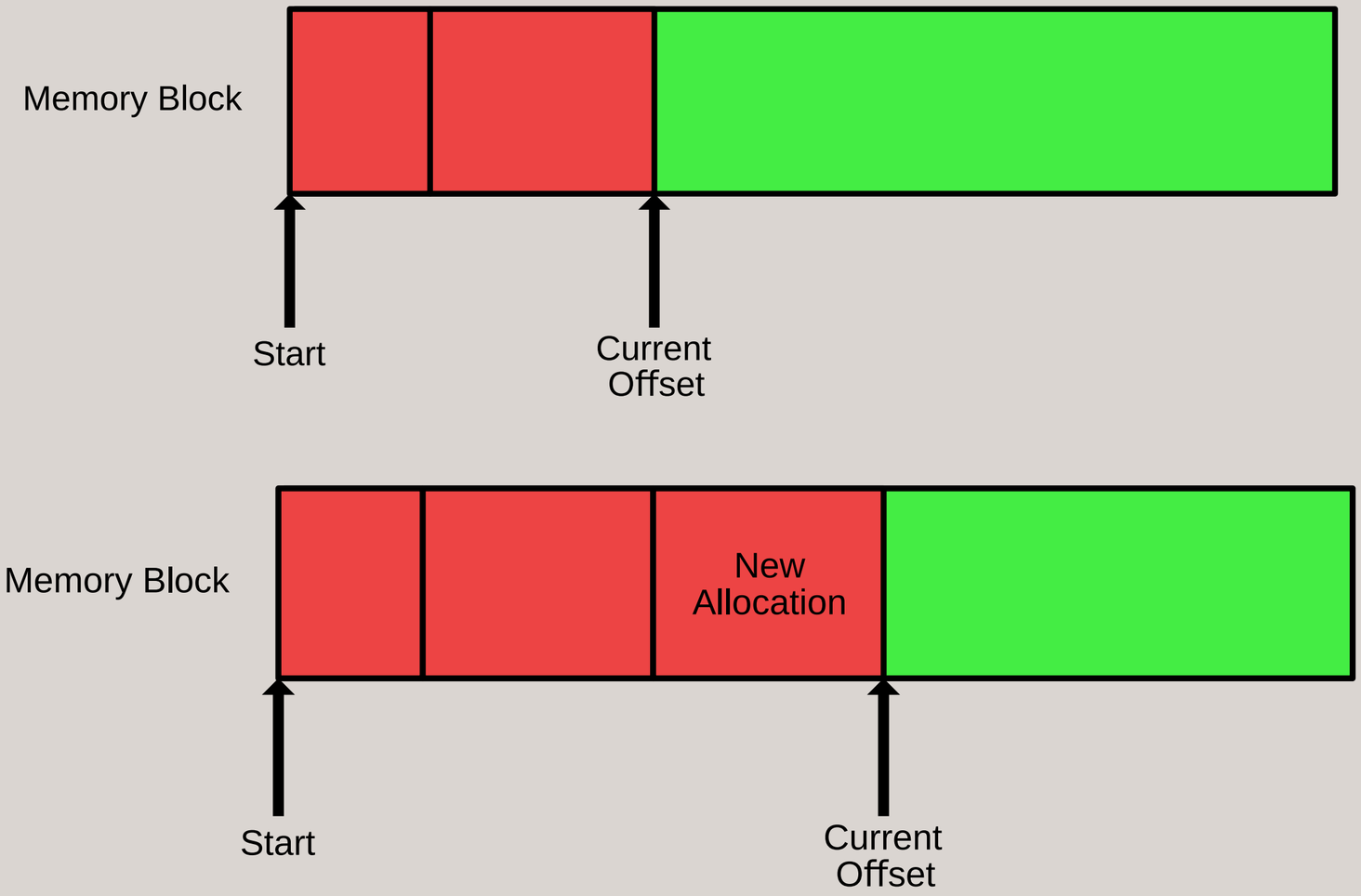
Allocators

Lots of types, but generally following are used:

- Arena (or Bump) Allocator
- Stack Allocator
- Free List Allocator

Each with its own set of caveats in terms of complexity and flexibility

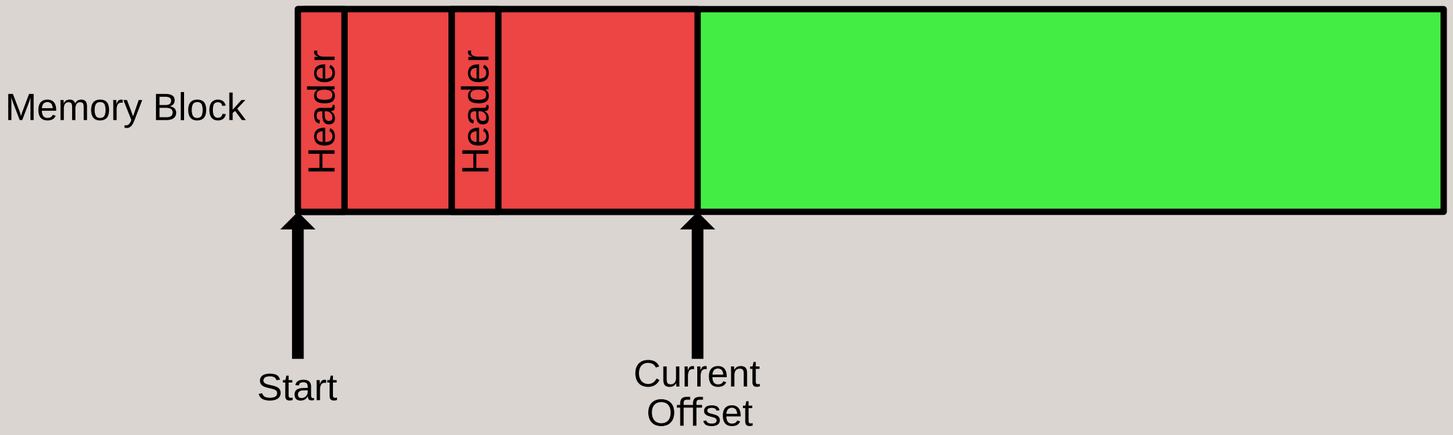
Arena Allocator

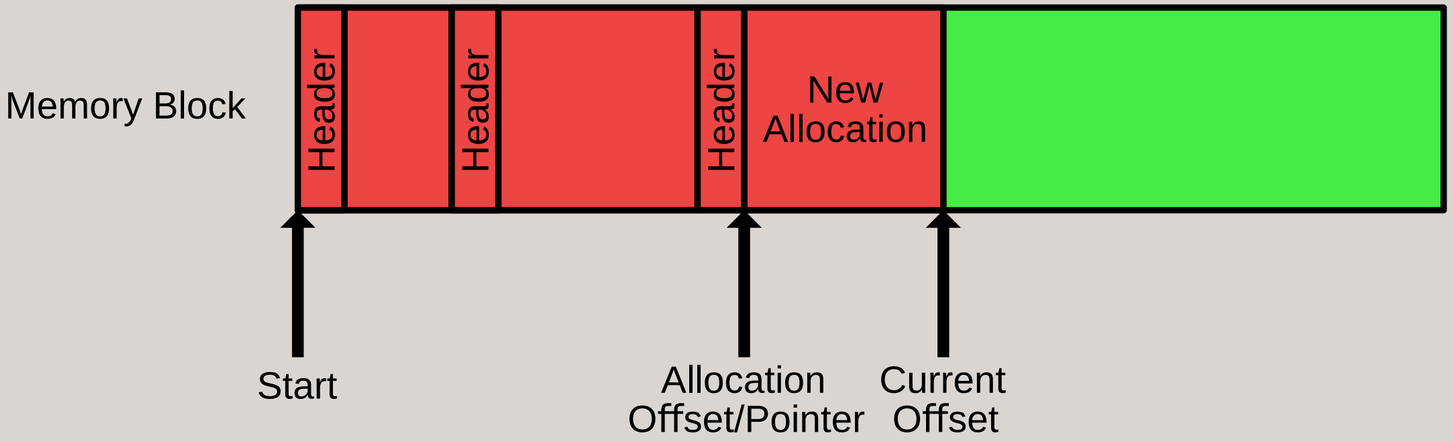


Arena Allocator

```
typedef struct Arena Arena;
struct Arena {
    uint8_t *buf;
    size_t  buf_len;
    size_t  prev_offset;
    size_t  curr_offset;
};

void arena_init(Arena *a, void *backing_buffer, size_t backing_buffer_length);
void *arena_alloc(Arena *a, size_t size);
void arena_free_all(Arena *a);
```

Stack Allocator

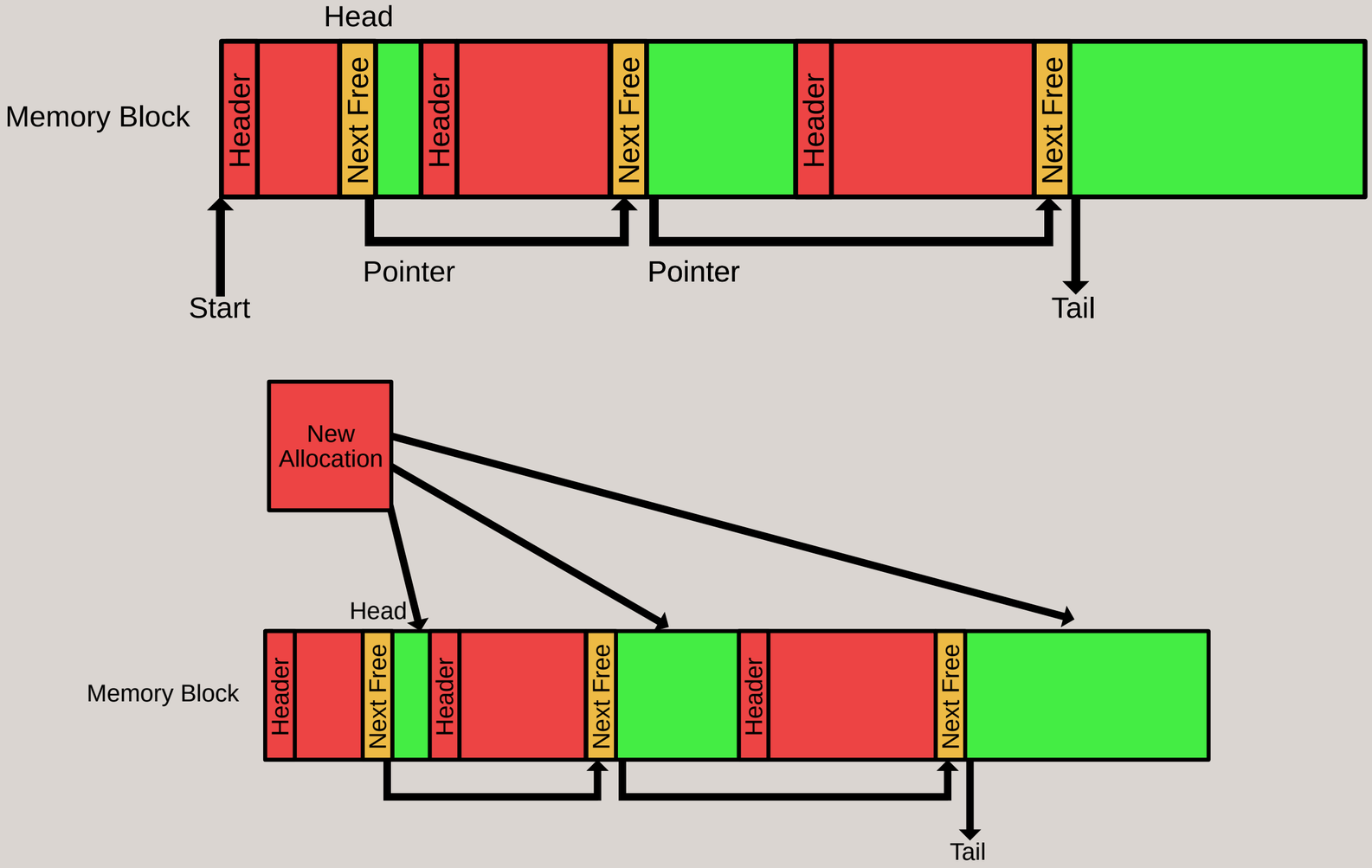




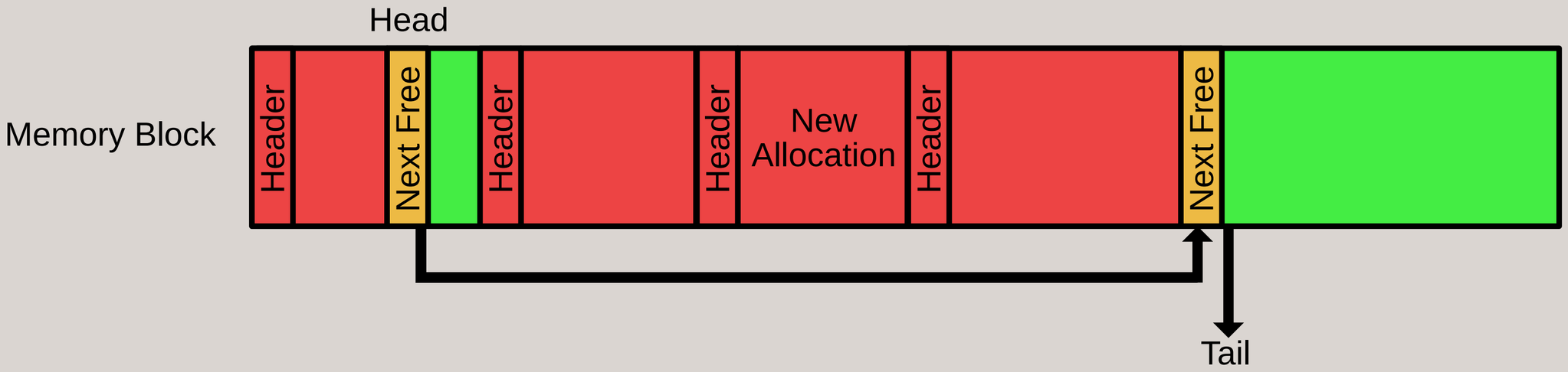
Stack Allocator

```
typedef struct Stack Stack;
struct Stack {
   uint8_t *buf;
   size_t   buf_len;
   size_t   curr_offset;
   size_t   prev_offset;
};

typedef struct Stack_Allocation_Header Stack_Allocation_Header;
struct Stack_Allocation_Header {
    size_t prev_offset;
};

void stack_init(Stack *s, void *backing_buffer, size_t backing_buffer_length);
void *stack_alloc(Stack *s, size_t size);
void stack_free(Stack *s, void* ptr);
void stack_free_all(Stack *s);
```

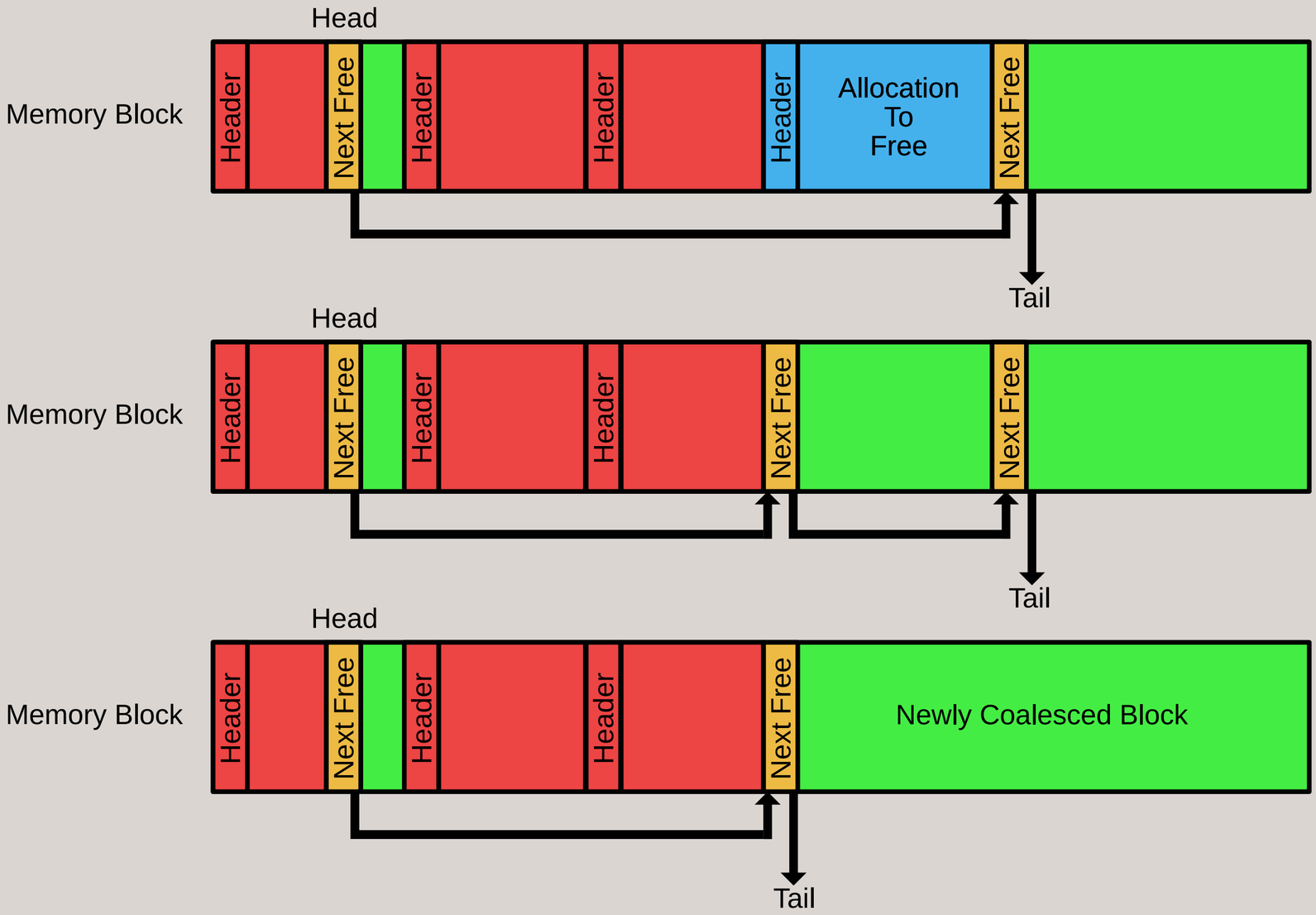
FreeList Allocator



FreeList Allocator



FreeList Allocator




That's all Folks!