

```
0xb7154f2f 2919 arena_lookup(ar_ptr);
  0xb7154f22 <__GI___libc_malloc+50>: 8b 93 78 fe ff ff mov  edx,DWORD PTR [ebx-0x188]
  0xb7154f28 <__GI___libc_malloc+56>: 65 8b 0d 00 00 00 00 mov  ecx,DWORD PTR gs:0x0
=> 0xb7154f2f <__GI___libc_malloc+63>: 8b 3c 11 mov  edi,DWORD PTR [ecx+edx*1]
```

```
(gdb) i r
```

```
eax      0x0  0
ecx      0xb67f5840  -1233168320
edx      0xfffffd8  -40
ebx      0xb7283ff4  -1222098956
esp      0xbffc2f70  0xbffc2f70
ebp      0x24  0x24
```

```
(gdb) x /x $ecx+$edx
0xb67f5818: 0xb7284440
```

```
(gdb) p &main_arena
$16 = (struct malloc_state *) 0xb7284440
```

这次分配使用的刚好是主场地

```
#0 new_heap (size=3168, top_pad=131072) at arena.c:523
#1 0xb79c227f in _int_new_arena (size=2040) at arena.c:712
#2 arena_get2 (avoid_arena=0x0, size=2040, a_tsd=<optimized out>) at arena.c:856
#3 arena_get2 (a_tsd=<optimized out>, size=2040, avoid_arena=0x0) at arena.c:823
#4 0xb79c422d in __libc_calloc (n=1, elem_size=2040) at malloc.c:3291
#5 0xb7397753 in ?? () from /lib/i386-linux-gnu/libglib-2.0.so.0
#6 0xb7397e6b in g_malloc0 () from /lib/i386-linux-gnu/libglib-2.0.so.0
#7 0xb7360215 in ?? () from /lib/i386-linux-gnu/libglib-2.0.so.0
#8 0xb73ac445 in g_slice_alloc () from /lib/i386-linux-gnu/libglib-2.0.so.0
#9 0xb73ac796 in g_slice_alloc0 () from /lib/i386-linux-gnu/libglib-2.0.so.0
#10 0xb73a004b in g_queue_new () from /lib/i386-linux-gnu/libglib-2.0.so.0
#11 0xb73912d5 in g_main_context_push_thread_default () from /lib/i386-linux-gnu/libglib-2.0.so.0
#12 0xb6e5e11c in ?? () from /usr/lib/i386-linux-gnu/gio/modules/libdconfsettings.so
#13 0xb73b5673 in ?? () from /lib/i386-linux-gnu/libglib-2.0.so.0
#14 0xb7afcd4c in start_thread (arg=0xb6c2fb40) at pthread_create.c:308
#15 0xb7a3b87e in clone () at ../sysdeps/unix/sysv/linux/i386/clone.S:130
```

(gdb) info threads

```
Id Target Id      Frame
```

```
* 2 Thread 0xb6c2fb40 (LWP 13642) "dconf worker" new_heap (size=3168, top_pad=131072) at arena.c:5
```

```
1 Thread 0xb7063840 (LWP 13639) "gemalloc" __GI__dl_debug_state () at dl-debug.c:77
```

(gdb) frame 2

#2 arena_get2 (avoid_arena=0x0, size=2040, a_tsd=<optimized out>) at arena.c:856

856 #else

(gdb) frame 4

#4 0xb79c422d in __libc_calloc (n=1, elem_size=2040) at malloc.c:3291

3291 av = arena_get2(av->next ? av : 0, sz);

```
if (mem == 0) {
```

```
    /* Maybe the failure is due to running out of mmapped areas. */
```

```
    if(av != &main_arena) {
```

```
        (void)mutex_lock(&main_arena.mutex);
```

```
        mem = _int_malloc(&main_arena, sz);
```

```
        (void)mutex_unlock(&main_arena.mutex);
```

```
    } else {
```

```
        /* ... or sbrk() has failed and there is still a chance to mmap() */
```

```
        (void)mutex_lock(&main_arena.mutex);
```

```
        av = arena_get2(av->next ? av : 0, sz);
```

```
        (void)mutex_unlock(&main_arena.mutex);
```

```
        if(av) {
```

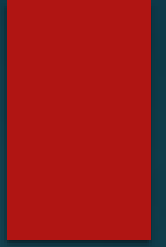
```
            mem = _int_malloc(av, sz);
```

```
            (void)mutex_unlock(&av->mutex);
```

```
        }
```

Calloc的错误处理代码中触发创建

不忘初心



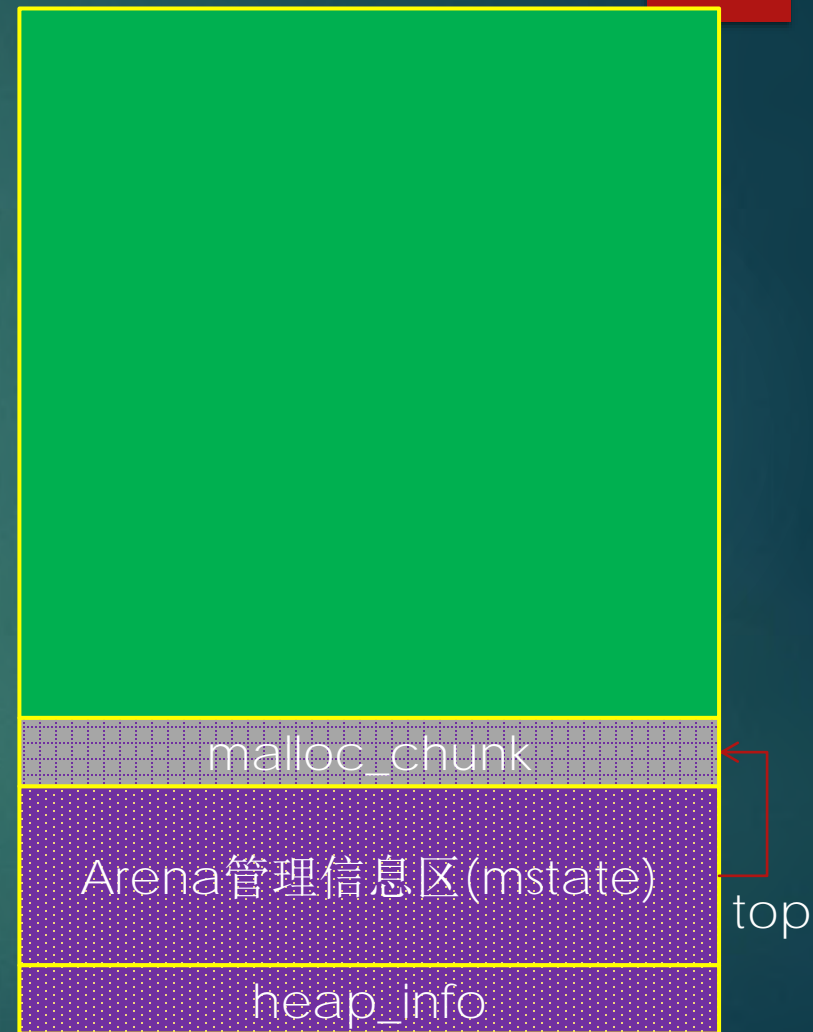
```
/* Malloc implementation for multiple threads without lock contention.  
Copyright (C) 2001-2006, 2007, 2008, 2009, 2011 Free Software Foundation, Inc.  
This file is part of the GNU C Library.  
Contributed by Wolfram Gloger <wg@malloc.de>, 2001.
```

* Why use this malloc?

This is **not the fastest, most space-conserving, most portable, or most tunable** malloc ever written. However it is among the fastest while also being among the most space-conserving, portable and tunable.

Consistent **balance** across these factors results in a good general-purpose allocator for malloc-intensive programs.

创建过程



源代码

mstate

`_int_new_arena`(size_t size)

```
{
    mstate a;
    heap_info *h;
    char *ptr;
    unsigned long misalign;

    h = new_heap(size + (sizeof(*h) + sizeof(*a) + MALLOC_ALIGNMENT),
                mp_.top_pad);
    if(!h) {
        /* Maybe size is too large to fit in a single heap. So, just try
           to create a minimally-sized arena and let _int_malloc() attempt
           to deal with the large request via mmap_chunk(). */
        h = new_heap(sizeof(*h) + sizeof(*a) + MALLOC_ALIGNMENT, mp_.top_pad);
        if(!h)
            return 0;
    }
}
```

```
a = h->ar_ptr = (mstate)(h+1);
malloc_init_state(a);
/*a->next = NULL;*/
a->system_mem = a->max_system_mem = h->size;
arena_mem += h->size;
#ifdef NO_THREADS
if((unsigned long)(mp_.mmapped_mem + arena_mem + main_arena.system_mem) >
    mp_.max_total_mem)
    mp_.max_total_mem = mp_.mmapped_mem + arena_mem + main_arena.system_mem;
#endif

/* Set up the top chunk, with proper alignment. */
ptr = (char*)(a + 1);
misalign = (unsigned long)chunk2mem(ptr) & MALLOC_ALIGN_MASK;
if (misalign > 0)
    ptr += MALLOC_ALIGNMENT - misalign;
top(a) = (mchunkptr)ptr;
set_head(top(a), (((char*)h + h->size) - ptr) | PREV_INUSE);

return a;
}
```


Arena小结

- ▶ Main_Arena是静态定义的
- ▶ 根据需要动态创建更多的Arena
 - ▶ 直到找到一个可用的
 - ▶ 最多每个线程一个，但可以通过配置参数设置上限
- ▶ 单CPU系统，常常有三个
- ▶ 增加CPU个数后（并发度增大），多线程的gemalloc小程序创建10个工作线程后，有6个Arena

他山之石，可以为错

Windows	Linux/Glibc
堆 (Heap)	Arena
段 (Segment)	Heap (子堆)
Entry	Chunk





概览

Arena

批发

分配和释
放

Valgrind



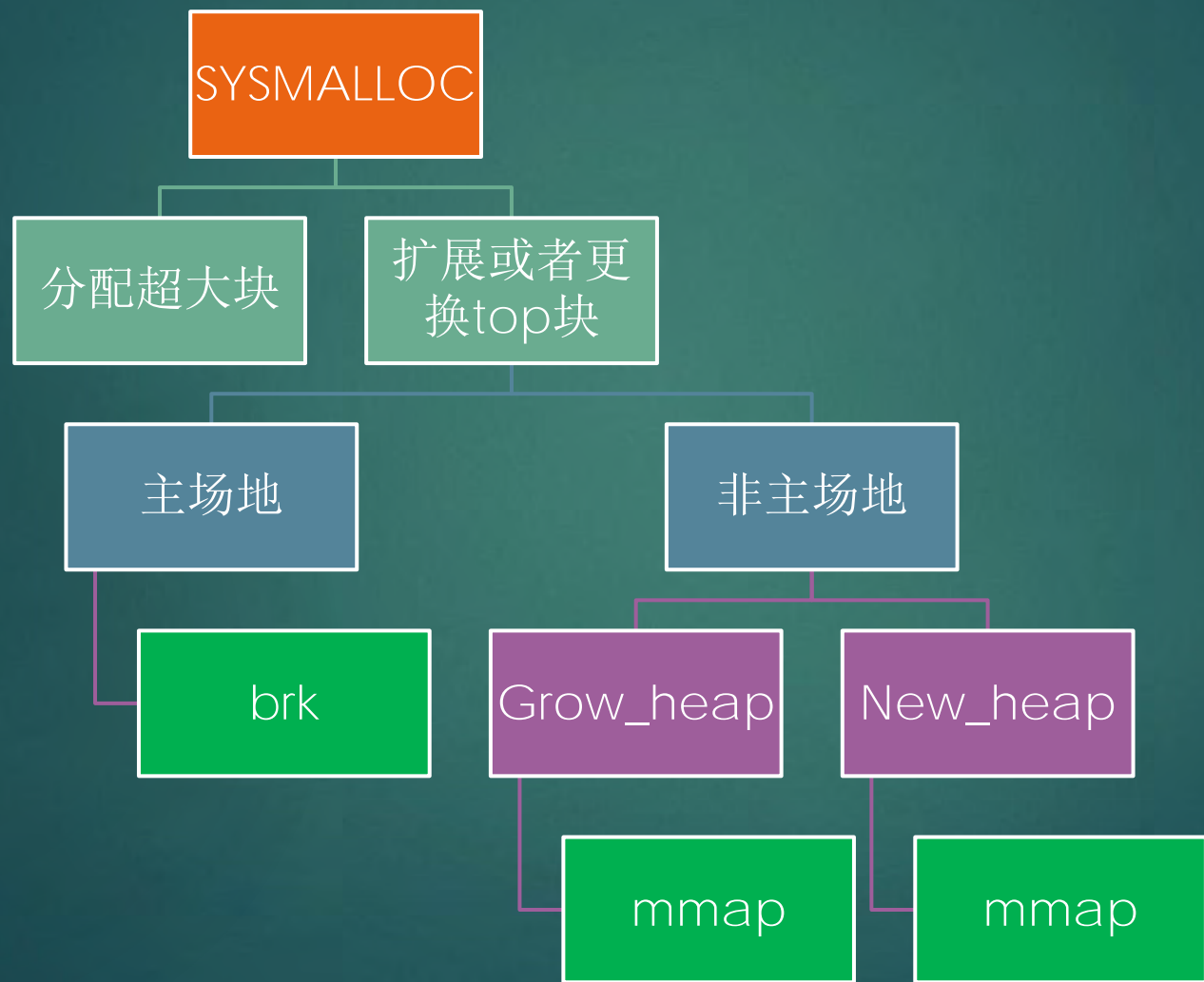
/*

sysmalloc handles malloc cases requiring more memory from the system.

On entry, it is assumed that `av->top` does not have enough space to service request for `nb` bytes, thus requiring that `av->top` be extended or replaced.

*/

三大块代码



主arena被触发扩容

(gdb) bt

#0 __brk (addr=0x0) at ../sysdeps/unix/sysv/linux/i386/brk.c:52

#1 0xb7a33571 in __GI___sbrk (increment=135168) at sbrk.c:44

#2 0xb79c4f4f in __GI___default_ **morecore** (increment=135168) at morecore.c:49

#3 0xb79c0e82 in **sYSMALLOC** (av=0xfffffd0, nb=360) at malloc.c:2535

#4 **_int_malloc** (av=0xfffffd0, bytes=352) at malloc.c:3932

#5 0xb79c2f5c in __GI___libc_malloc (bytes=352) at malloc.c:2924

#6 0xb79c30c7 in __GI___libc_malloc (bytes=352) at malloc.c:2917

#7 0xb79aee08 in __fopen_internal (filename=0xb710ef86 "/proc/filesystems", mode=0xb710ee7b "r", is3

#8 0xb79b163b in _IO_fopen64 (filename=0xb710ef86 "/proc/filesystems", mode=0xb710ee7b "r") at iofo

#9 0xb70fd2d3 in ?? () from /lib/i386-linux-gnu/libselinux.so.1

#10 0xb7fece5b in call_init (env=0xbffff41c, argv=0xbffff414, argc=1, l=<optimized out>) at dl-init.c:85

#11 call_init (l=<optimized out>, argc=1, argv=0xbffff414, env=0xbffff41c) at dl-init.c:35

#12 0xb7fecf44 in _dl_init (main_map=<optimized out>, argc=1, argv=0xbffff414, env=0xbffff41c) at dl-in

#13 0xb7fdf20f in _dl_start_user () from /lib/ld-linux.so.2


```
/*  
MORECORE is the name of the routine to call to obtain more memory  
from the system. See below for general guidance on writing  
alternative MORECORE functions, as well as a version for WIN32 and a  
sample version for pre-OSX macos.
```

```
*/
```

```
#ifndef MORECORE
```

```
#define MORECORE sbrk
```

```
#endif
```



brk

```
#include <unistd.h>
```

```
int brk(void *addr);
```

```
void *sbrk(intptr_t increment);
```

- ▶ change data segment size

NAME

brk, sbrk - change space allocation (**LEGACY**)

SYNOPSIS

```
#include <unistd.h>

int brk(void *addr);
void *sbrk(intptr_t incr);
```

DESCRIPTION

The *brk()* and *sbrk()* functions are used to change the amount of space allocated for the calling process. The change is made by resetting the process' break value and allocating the appropriate amount of space. The amount of allocated space increases as the break value increases. The newly-allocated space is set to 0. However, if the application first decrements and then increments the break value, the contents of the reallocated space are unspecified.

The *brk()* function sets the break value to *addr* and changes the allocated space accordingly.

更替top块

Hardware watchpoint 9: main_arena->top

Old value = (mchunkptr) 0xb7af2470

New value = (mchunkptr) 0x804c000


Hardware watchpoint 10: main_arena->top

Old value = (mchunkptr) 0xb7af2470


New value = (mchunkptr) 0x804c000

sYSMALLOc (av=0xfffffd0, nb=360) at malloc.c:2697

2697 set_head(av->top, (snd_brk - aligned_brk + correction) | PREV_INUSE);



```
ge@gewubox:~$ cat /proc/2748/maps | grep "\[heap\  
0804c000-0806d000 rw-p 00000000 00:00 0      [heap]
```



```
#0 __GI__sbrk (increment=143360) at sbrk.c:55
#1 0xb79c4f4f in __GI__default_morecore (increment=143360) at morecore.c:49
#2 0xb79c0e82 in sYSMALLOC (av=0xffffefd0, nb=32800) at malloc.c:2535
#3 _int_malloc (av=0xffffefd0, bytes=32796) at malloc.c:3932
#4 0xb79c2f5c in __GI__libc_malloc (bytes=32796) at malloc.c:2924
#5 0xb79fffc8 in __alloc_dir (fd=8, close_fd=<optimized out>, flags=<optimized out>, statp=0x0) at ../sysde
#6 0xb7a000f7 in __opendirat (dfd=<optimized out>, name=<optimized out>) at ../sysdeps/unix/opendir.c
#7 0xb7a0013d in __opendir (name=0xb75eed88 "/usr/lib/i386-linux-gnu/gio/modules") at ../sysdeps/unix/
#8 0xb737d0dc in g_dir_open () from /lib/i386-linux-gnu/libglib-2.0.so.0
.....
#18 0xb6e1bc00 in os_utils_is_blacklisted () from /usr/lib/liboverlay-scrollbar-0.2.so.0
#19 0xb7c962cc in ?? () from /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0
#20 0xb739db12 in g_option_context_parse () from /lib/i386-linux-gnu/libglib-2.0.so.0
#21 0xb7c96988 in gtk_parse_args () from /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0
#22 0xb7c96a13 in gtk_init_check () from /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0
#23 0xb7c96a53 in gtk_init () from /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0
#24 0x08049332 in main (argc=1, argv=0xbffff414) at gemalloc.c:145
```

0xb7af2780, 0xb7af2780, 0xb7af2788, 0xb7af2788...}, binmap = {20, 33554432, 16, 0}, next = 0xb7af2440, r

0x0, **system_mem = 135168**, max_system_mem = 135168}

(gdb) watch main_arena->system_mem

Hardware watchpoint 15: main_arena->system_mem

(gdb) c

Continuing.

Hardware watchpoint 15: main_arena->system_mem

Old value = 135168

New value = 278528

sYSMALLOC (av=0xffffefd0, nb=32800) at malloc.c:2591

2591 if (brk == old_end && snd_brk == (char*)(MORECORE_FAILURE))

(gdb) p 135168+143360

\$30 = 278528

#0 sYSMALLOC (av=0xffffefd0, nb=32800) at malloc.c:2591

#1 _int_malloc (av=0xffffefd0, bytes=32796) at malloc.c:3932

#2 0xb79c2f5c in __GI___libc_malloc (bytes=32796) at malloc.c:2924


```
ge@gewubox:~/eglibc-2.15$ cat /proc/13795/maps | grep "heap"
08048000-0804a000 r-xp 00000000 08:01 1835228 /home/ge/work/geheap/gemalloc
0804a000-0804b000 r--p 00001000 08:01 1835228 /home/ge/work/geheap/gemalloc
0804b000-0804c000 rw-p 00002000 08:01 1835228 /home/ge/work/geheap/gemalloc
0804c000-0806d000 rw-p 00000000 00:00 0 [heap]
0806d000-08090000 rw-p 00000000 00:00 0 [heap]
```

所以，maps文件看到的heap块数与arena个数不是一会事，可能不一致

辅Arena用mmap从内核批发内存

```
#include <sys/mman.h>
```

```
void *mmap(void *addr, size_t length, int prot, int flags,  
           int fd, off_t offset);  
int munmap(void *addr, size_t length);
```

- ▶ 历史上，比brk机制要出现的晚

mmap and associated systems calls were designed as part of the Berkeley Software Distribution (BSD) version of Unix. Their API was already described in the 4.2BSD System Manual, even though it was neither implemented in that release, nor in 4.3BSD.[1] Sun Microsystems had implemented this very API, though, in their SunOS operating system. The BSD developers at U.C. Berkeley requested Sun to donate its implementation, but these talks never led to any transfer of code; 4.3BSD-Reno was shipped instead with an implementation based on the virtual memory system of Mach.[2] -- wikipedia

创建第一个子堆

```
#0 mmap () at ../sysdeps/unix/sysv/linux/i386/mmap.S:98
#1 0xb79be6c7 in new_heap (size=<optimized out>, top_pad=<optimized out>) at arena.c:554
#2 0xb79c227f in _int_new_arena (size=2040) at arena.c:712
#3 arena_get2 (avoid_arena=0x0, size=2040, a_tsd=<optimized out>) at arena.c:856
#4 arena_get2 (a_tsd=<optimized out>, size=2040, avoid_arena=0x0) at arena.c:823
#5 0xb79c422d in __libc_calloc (n=1, elem_size=2040) at malloc.c:3291
```



```
/* Create a new heap. size is automatically rounded up to a multiple
of the page size. */

static heap_info *
internal_function
new_heap(size_t size, size_t top_pad)
{
    size_t page_mask = GLRO(dl_pagesize) - 1;
    char *p1, *p2;
    unsigned long ul;
    heap_info *h;

    if(size+top_pad < HEAP_MIN_SIZE)
        size = HEAP_MIN_SIZE;
    else if(size+top_pad <= HEAP_MAX_SIZE)
        size += top_pad;
    else if(size > HEAP_MAX_SIZE)
        return 0;
    else
        size = HEAP_MAX_SIZE;
    size = (size + page_mask) & ~page_mask;

    /* A memory region aligned to a multiple of HEAP_MAX_SIZE is needed.
    No swap space needs to be reserved for the following large
    mapping (on Linux, this is the case for all non-writable mappings
    anyway). */
    p2 = MAP_FAILED;
    if(aligned_heap_area) {
        p2 = (char *)MMAP(aligned_heap_area, HEAP_MAX_SIZE, PROT_NONE,
            MAP_PRIVATE|MAP_NORESERVE);
        aligned_heap_area = NULL;
        if (p2 != MAP_FAILED && ((unsigned long)p2 & (HEAP_MAX_SIZE-1))) {
            munmap(p2, HEAP_MAX_SIZE);
            p2 = MAP_FAILED;
        }
    }
}
```

创建第二个

New_heap()函数返回后，观察

```
(gdb) p *heap
```

```
$27 = {ar_ptr = 0xb4f00010, prev = 0xb4f00000, size = 266240,  
      mprotect_size = 266240, pad = 0xb3100010 ""}
```

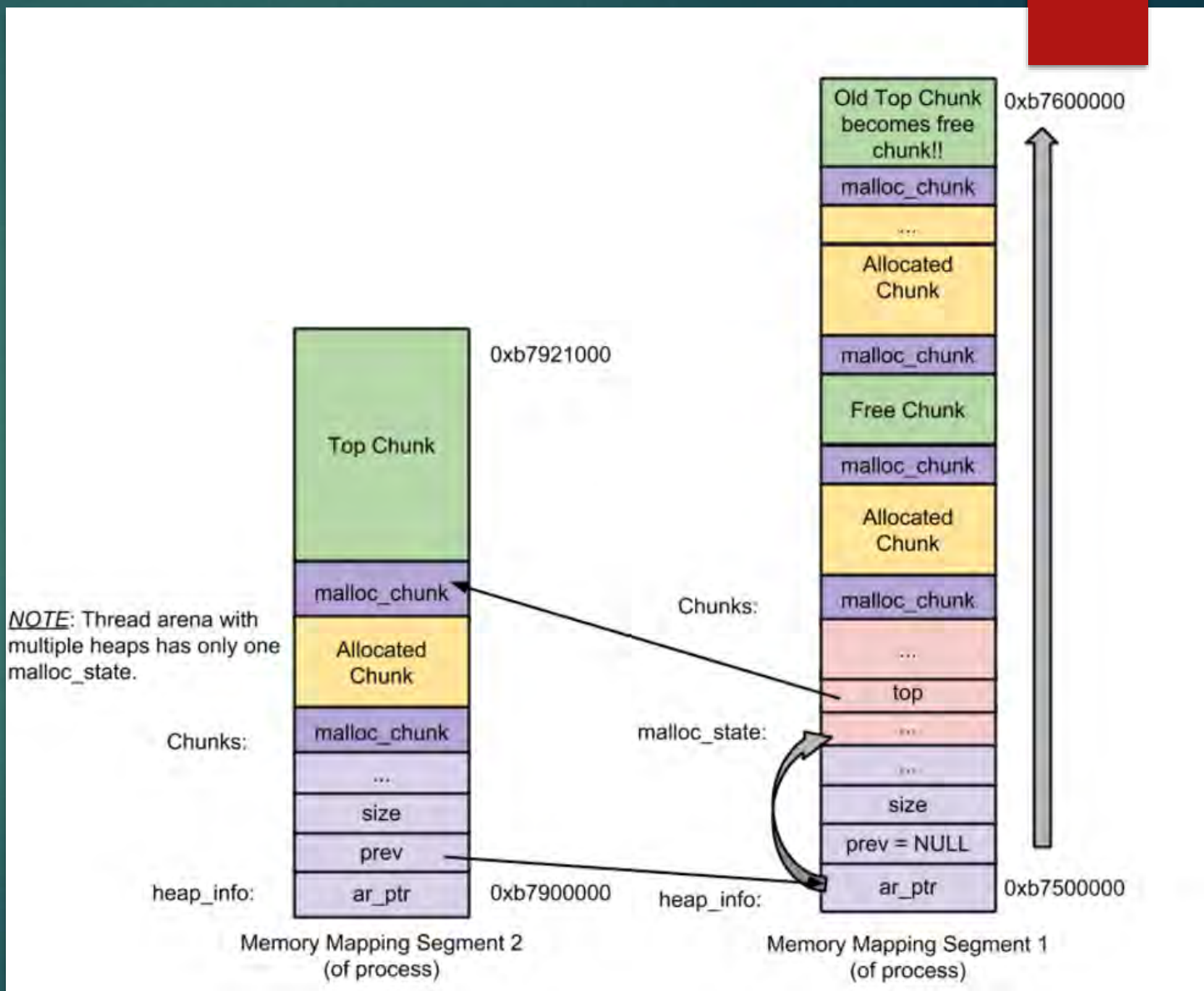
```
(gdb) bt
```

```
#0 sYSMALLOC (av=0xb4f00010, nb=131064) at malloc.c:2476  
#1 _int_malloc (av=0xb4f00010, bytes=131060) at malloc.c:3932  
#2 0xb7818f5c in __GI___libc_malloc (bytes=131060) at malloc.c:2924  
#3 0x080492e4 in do_malloc (nsize=131060, no=10, threadno=1) at gemalloc.c:49  
#4 0x0804947c in thread_func (arg=0x1) at gemalloc.c:111  
#5 0xb7952d4c in start_thread (arg=0xb5810b40) at pthread_create.c:308  
#6 0xb789187e in clone () at ../sysdeps/unix/sysv/linux/i386/clone.S:130
```

创建新的heap，此后，一个场地具有多个heap

一个arena 的两个子堆

- ▶ 老的top块变成空闲块
- ▶ 新分配出的运营区划出一块给当前的请求，余下的成为新的top块
- ▶ 两个子堆共用一个mstate结构体




```
(gdb) p *((heap_info*)0xb2f00000)
$40 = {ar_ptr = 0xb4f00010, prev = 0xb3100000, size = 1048576,
      mprotect_size = 1048576, pad = 0xb2f00010 ""}
(gdb) p *((heap_info*)0xb2f00000)->prev
$41 = {ar_ptr = 0xb4f00010, prev = 0xb4f00000, size = 1048576,
      mprotect_size = 1048576, pad = 0xb3100010 ""}
(gdb) p *((heap_info*)0xb2f00000)->prev->prev
$42 = {ar_ptr = 0xb4f00010, prev = 0x0, size = 978944, mprotect_size = 978944,
      pad = 0xb4f00010 ""}
```

后两个子堆的
大小都为1MB，
第一个略小，
arena结构体有
1104字节

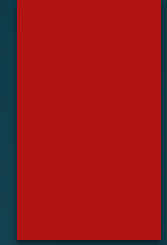
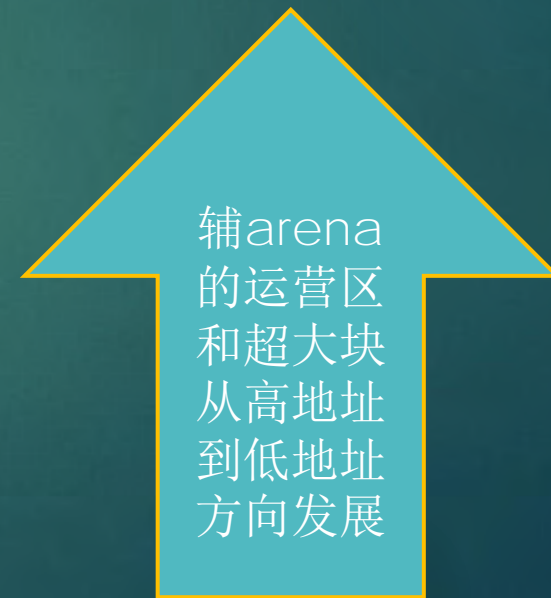
分配超大块

```
/*  
If have mmap, and the request size meets the mmap threshold, and  
the system supports mmap, and there are few enough currently  
allocated mmapped regions, try to directly map this request  
rather than expanding top.  
*/  
  
if ((unsigned long)(nb) >= (unsigned long)(mp_.mmap_threshold) &&  
    (mp_.n_mmmaps < mp_.n_mmmaps_max)) {
```

分配超大块($\geq 128\text{KB}$)

```
#0 mmap () at ../sysdeps/unix/sysv/linux/i386/mmap.S:35
#1 0xb7817328 in sYSMALLOC (av=0xb4f00010, nb=131080) at malloc.c:2400
#2 _int_malloc (av=0xb4f00010, bytes=131072) at malloc.c:3932
#3 0xb7818f5c in __GI___libc_malloc (bytes=131072) at malloc.c:2924
#4 0x080492e4 in do_malloc (nsize=131072, no=1, threadno=1) at gemalloc.c:49
#5 0x0804947c in thread_func (arg=0x1) at gemalloc.c:111
#6 0xb7952d4c in start_thread (arg=0xb5810b40) at pthread_create.c:308
#7 0xb789187e in clone () at ../sysdeps/unix/sysv/linux/i386/clone.S:130
```


08725000-08745000	rw-p	00000000	00:00	0	[heap]
08745000-08785000	rw-p	00000000	00:00	0	[heap]
08785000-0879b000	rw-p	00000000	00:00	0	[heap]
0879b000-087db000	rw-p	00000000	00:00	0	[heap]
087db000-0881b000	rw-p	00000000	00:00	0	[heap]
0881b000-0885b000	rw-p	00000000	00:00	0	[heap]
0885b000-0889b000	rw-p	00000000	00:00	0	[heap]
0889b000-088db000	rw-p	00000000	00:00	0	[heap]
088db000-0891b000	rw-p	00000000	00:00	0	[heap]
0891b000-0895b000	rw-p	00000000	00:00	0	[heap]
0895b000-0899b000	rw-p	00000000	00:00	0	[heap]
0899b000-089db000	rw-p	00000000	00:00	0	[heap]
089db000-08a1b000	rw-p	00000000	00:00	0	[heap]
b2b00000-b2c41000	rw-p	00000000	00:00	0	
b2c41000-b2d00000	---p	00000000	00:00	0	
b2d00000-b2f00000	rw-p	00000000	00:00	0	
b2f00000-b3100000	rw-p	00000000	00:00	0	
b3100000-b3200000	rw-p	00000000	00:00	0	
b32f8000-b3319000	rw-p	00000000	00:00	0	
b3319000-b333a000	rw-p	00000000	00:00	0	
b333a000-b3fbb000	rw-p	00000000	00:00	0	
b3fbb000-b4c3c000	rw-p	00000000	00:00	0	
b4c3c000-b4d7d000	rw-p	00000000	00:00	0	
b4d7d000-b4ebe000	rw-p	00000000	00:00	0	
b4ebe000-b4edf000	rw-p	00000000	00:00	0	
b4edf000-b4f00000	rw-p	00000000	00:00	0	
b4f00000-b4fef000	rw-p	00000000	00:00	0	
b4fef000-b5000000	---p	00000000	00:00	0	





概览


Arena

批发

分配和释
放

Valgrind

从外部接口到内部实现

```
/*----- Public wrappers. -----*/  
  
void*  
public_mALLOc(size_t bytes)  
{  
    mstate ar_ptr;  
    void *victim;  
  
    __malloc_ptr_t (*hook) (size_t, __const __malloc_ptr_t)  
    = force_reg (__malloc_hook);  
    if (__builtin_expect (hook != NULL, 0))  
        return (*hook)(bytes, RETURN_ADDRESS (0));  
  
    arena_lookup(ar_ptr);  
  
    arena_lock(ar_ptr, bytes);  
    if(!ar_ptr)  
        return 0;  
    victim = _int_malloc(ar_ptr, bytes);  
    if(!victim) {  
        /* Maybe the failure is due to running out of mmaped areas. */  
        if(ar_ptr != &main_arena) {  
                    }  
    }  
}
```

#0 _int_malloc (av=0xb7fc7440, bytes=10) at malloc.c:3485

#1 0xb7e97f5c in **__GI___libc_malloc** (bytes=10) at malloc.c:2924

#2 0xb7e980c7 in __GI___libc_malloc (bytes=10) at malloc.c:2917

#3 0x08048401 in main (argc=1, argv=0xbffff414) at geheap.c:9

内部分配函数

```
/*
----- malloc -----
*/
Void_t*
_int_malloc(mstate av, size_t bytes)
{
INTERNAL_SIZE_T nb;          /* normalized request size */
unsigned int  idx;          /* associated bin index */
mbinptr      bin;          /* associated bin */
mfastbinptr* fb;           /* associated fastbin */

mchunkptr    victim;       /* inspected/selected chunk */
INTERNAL_SIZE_T size;      /* its size */
int          victim_index; /* its bin index */

mchunkptr    remainder;    /* remainder from a split */
unsigned long remainder_size; /* its size */

unsigned int  block;        /* bit map traverser */
unsigned int  bit;          /* bit map traverser */
unsigned int  map;          /* current word of binmap */

mchunkptr    fwd;          /* misc temp for linking */
mchunkptr    bck;          /* misc temp for linking */
```

malloc_chunk

```
/*
  This struct declaration is misleading (but accurate and necessary).
  It declares a "view" into memory allowing access to necessary
  fields at known offsets from a given base. See explanation below.
*/

struct malloc_chunk {

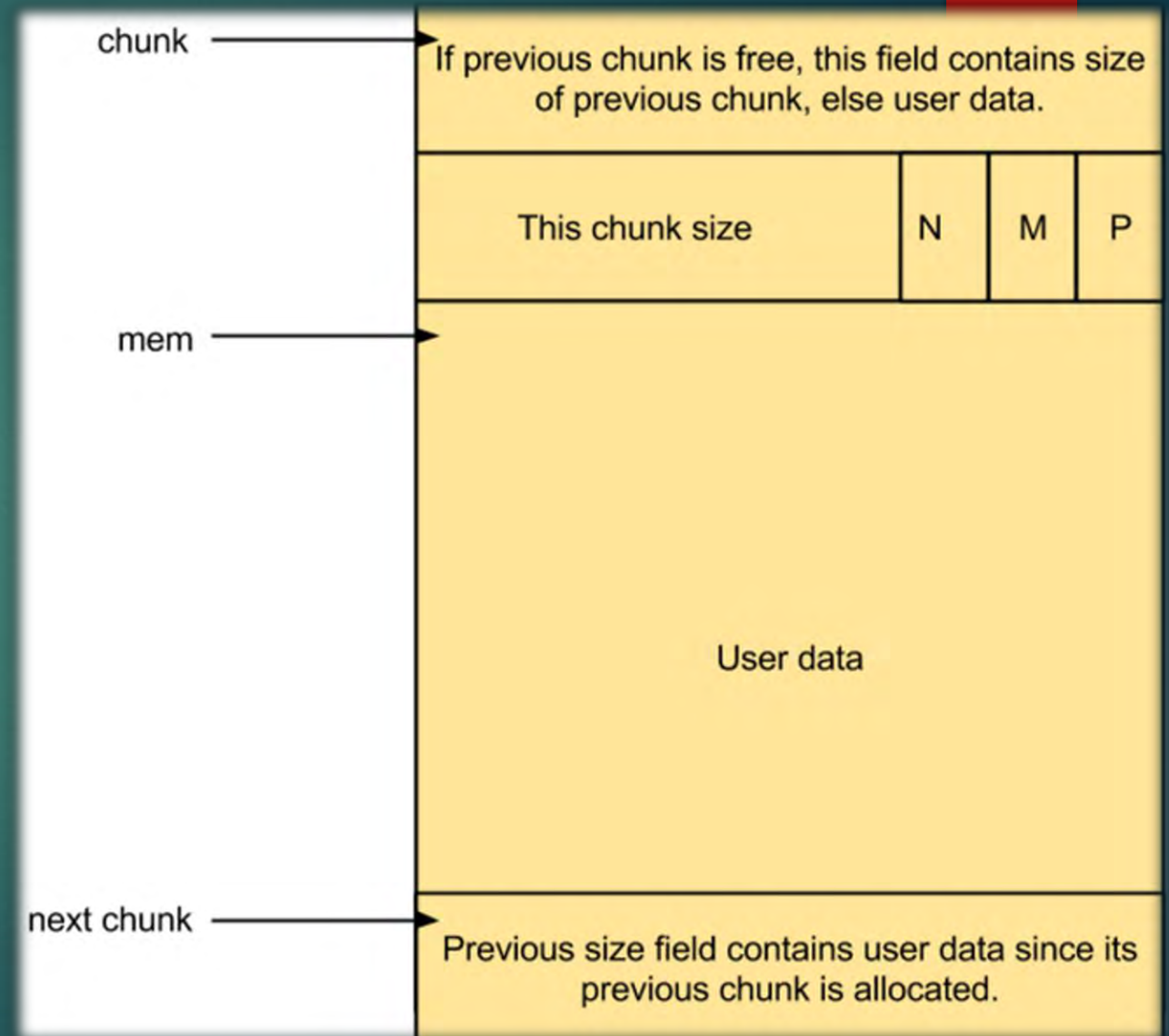
  INTERNAL_SIZE_T    prev_size; /* Size of previous chunk (if free). */
  INTERNAL_SIZE_T    size;      /* Size in bytes, including overhead. */

  struct malloc_chunk* fd;      /* double links -- used only if free. */
  struct malloc_chunk* bk;

  /* Only used for large blocks: pointer to next larger size. */
  struct malloc_chunk* fd_nextsize; /* double links -- used only if free. */
  struct malloc_chunk* bk_nextsize;
};
```

Allocated chunk

- ▶ `prev_size`: If the previous chunk is free, this field contains the size of previous chunk. Else if previous chunk is allocated, this field contains previous chunk's user data.
- ▶ `size`: This field contains the size of this allocated chunk. Last 3 bits of this field contains flag information.
 - ▶ `PREV_INUSE (P)` – This bit is set when previous chunk is allocated.
 - ▶ `IS_MMAPPED (M)` – This bit is set when chunk is `mmap`'d.
 - ▶ `NON_MAIN_ARENA (N)` – This bit is set when this chunk belongs to a thread arena.




```
/* size field is or'ed with PREV_INUSE when previous adjacent chunk in use */
```

```
#define PREV_INUSE 0x1
```

三个标志位

```
/* extract inuse bit of previous chunk */
```

```
#define prev_inuse(p) ((p)->size & PREV_INUSE)
```

```
/* size field is or'ed with IS_MMAPPED if the chunk was obtained with mmap() */
```

```
#define IS_MMAPPED 0x2
```

```
/* check for mmap()'ed chunk */
```

```
#define chunk_is_mmapped(p) ((p)->size & IS_MMAPPED)
```

```
/* size field is or'ed with NON_MAIN_ARENA if the chunk was obtained  
from a non-main arena. This is only set immediately before handing  
the chunk to the user, if necessary. */
```

```
#define NON_MAIN_ARENA 0x4
```

```
/* check for chunk from non-main arena */
```

```
#define chunk_non_main_arena(p) ((p)->size & NON_MAIN_ARENA)
```

看人下菜碟儿

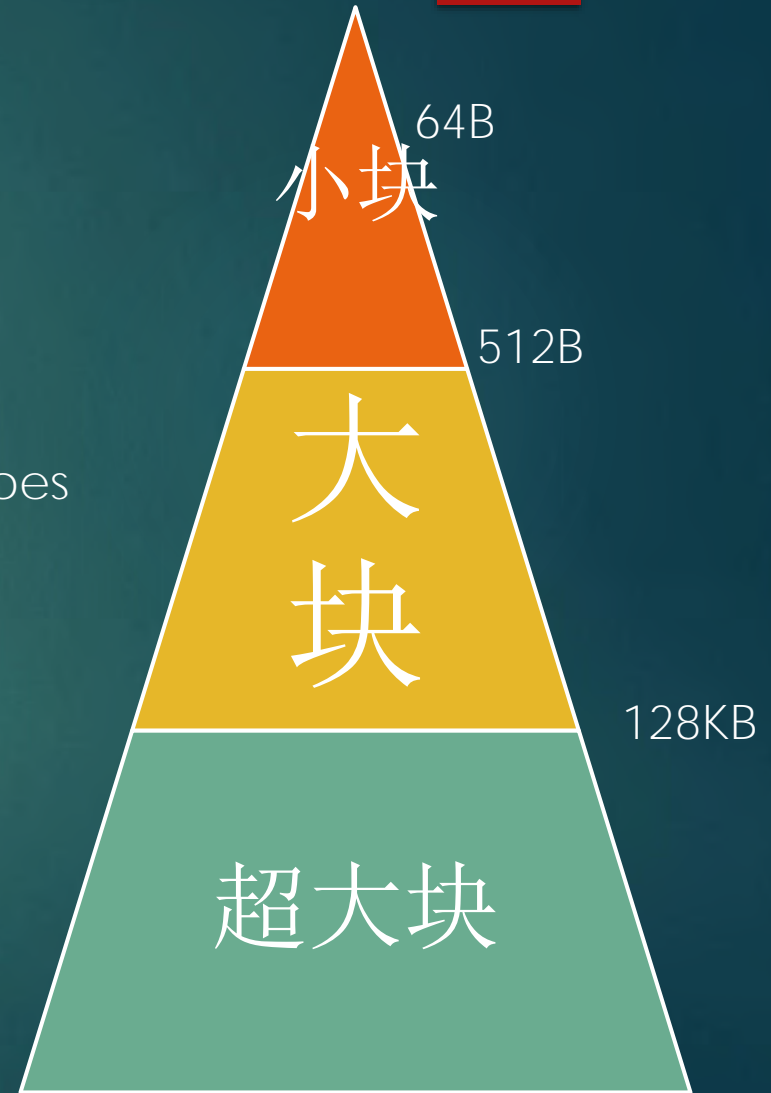
The main properties of the algorithms are:

- * For large (≥ 512 bytes) requests, it is a pure best-fit allocator, with ties normally decided via FIFO (i.e. least recently used).
- * For small (≤ 64 bytes by default) requests, it is a caching allocator, that maintains pools of quickly recycled chunks.
- * In between, and for combinations of large and small requests, it does the best it can trying to meet both goals at once.
- * For very large requests ($\geq 128\text{KB}$ by default), it relies on system memory mapping facilities, if supported.

-- malloc.c

赵姨娘也不答话，走上来便将粉照着芳官脸上撒来，指着芳官骂道：“我家里下三等奴才也比你高贵些的，你都会看人下菜碟儿。宝玉要给东西，你拦在头里，莫不是要了你的了？拿这个哄他，你只当他不认得呢！”

--清·曹雪芹《红楼梦》第六十回 茉莉粉替去蔷薇硝 玫瑰露引来茯苓霜





图书馆目录的怀旧情结

- ▶ 在有目录的时代，读者见到的都是**读者目录**，放在目录室里，有一个专门的目录组负责，定期维护。记得先后担任《国家图书馆学刊》和《中国图书馆学报》常务副主编的蒋弘女士曾经在目录室工作过，南京图书馆《新世纪图书馆》常务副主编彭飞老师也当过目录组组长。采编工作人员负责的是**公务目录有三套**，包括**采访目录、编目目录、书库目录**，所有目录加起来至少**有四大套**，里面还包括著者目录、书名目录（或者著者和书名混合的排架目录）和分类目录等，卡片有7-8套，分别维护，很有难度。

节选自书蠹精的博客文章

http://blog.sina.com.cn/s/blog_495d62640102v50l.html

细看卡片箱



卡片箱中的 卡片





Bin（卡片箱）

Bin	卡片箱
索引空闲块	索引可借的书
有时也被称为空闲链表(free list)	读者卡片
为了加快释放和分配速度，释放时放入bin，分配时先在bin里寻找	提高查找书的速度
分成两大类，多种bin	“所有目录加起来至少有四大套，里面还包括著者目录、书名目录（或者著者和书名混合的排架目录）和分类目录等，卡片有7-8套”

两类bin

fastbin

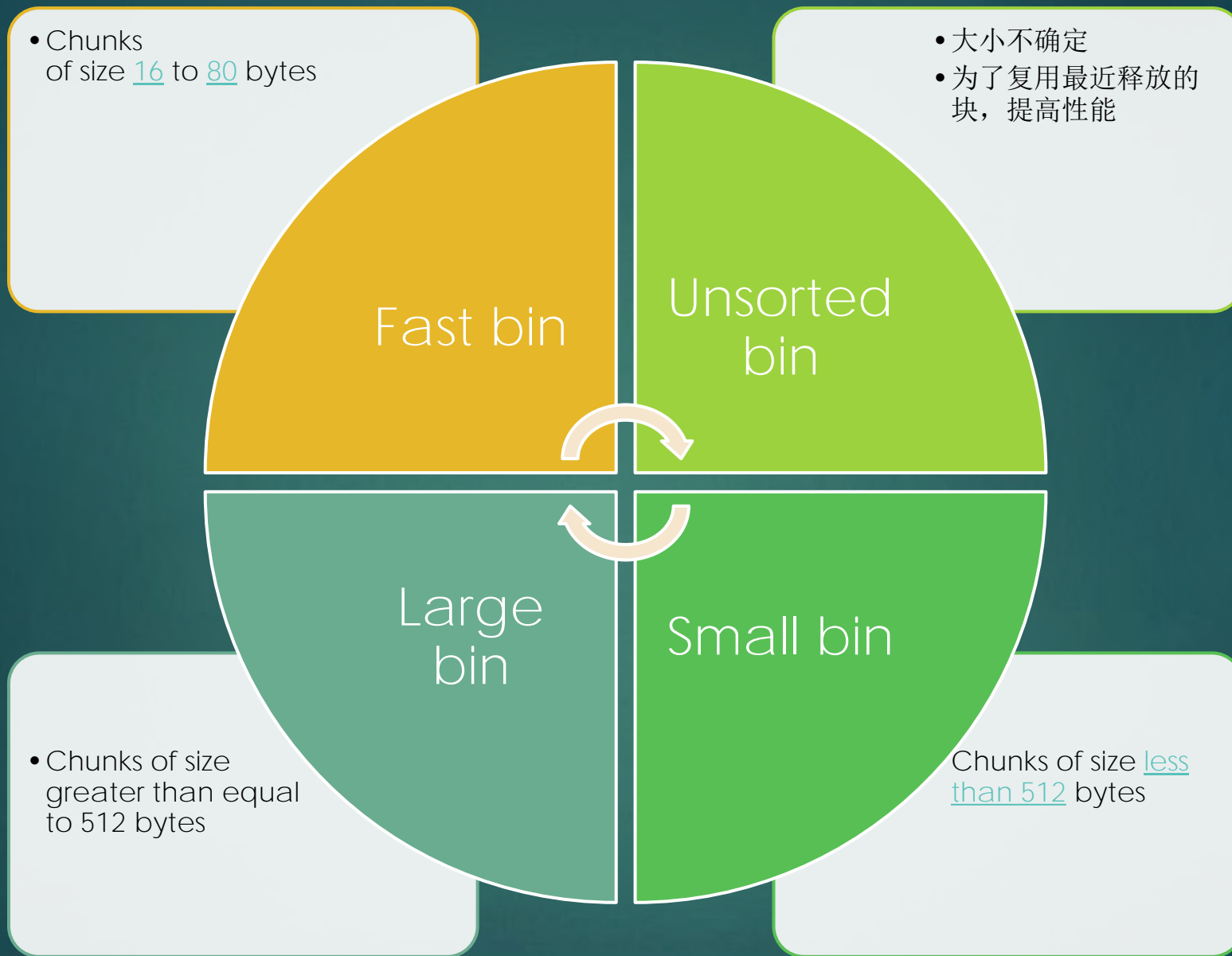
- 共有10个
- 只记录小于80字节的小块
- 每个bin里的块大小相同
- 单向链表



普通bin

- 分成三种
 - 无序，小块，大块
- 以双向链表索引

Bin总览

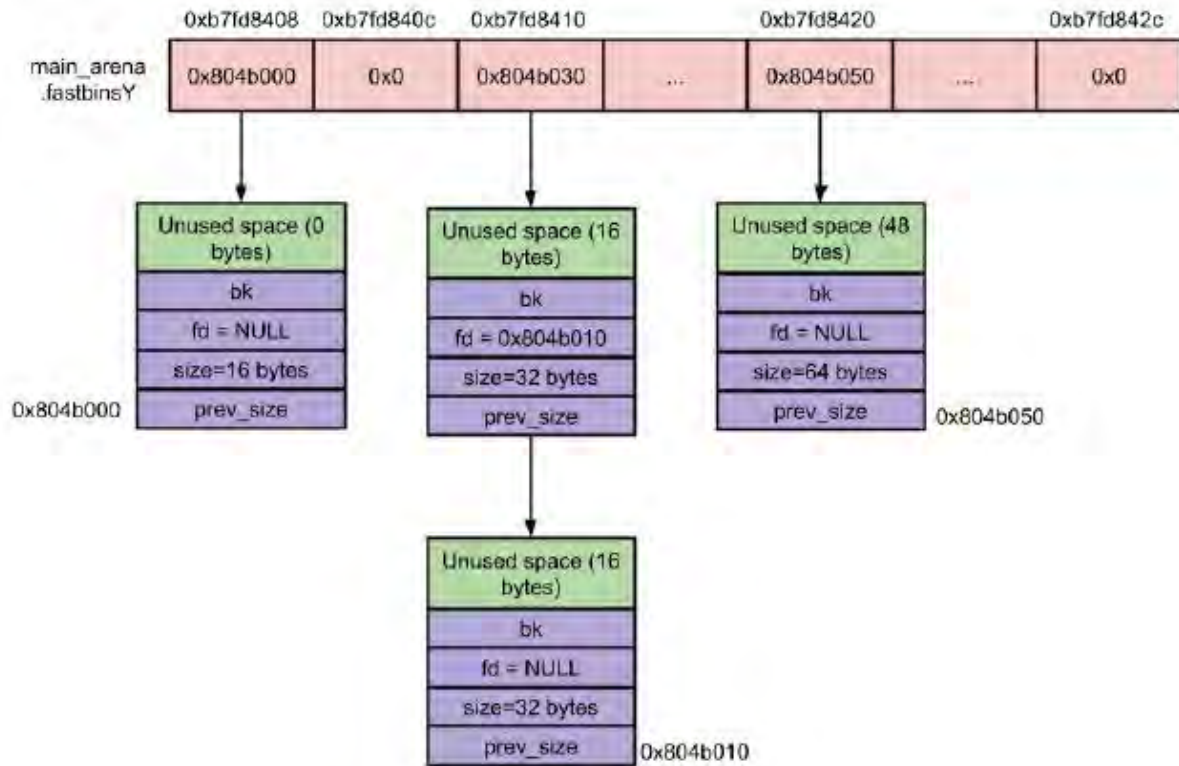


fastbin

- ▶ 里面的块不标记为空闲，保持busy状态
 - ▶ 释放和分配都快
- ▶ 单向链表
- ▶ 添加和删除都在头尾端进行，先进先出
- ▶ 按8字节单位划分
 - ▶ bin 0存放的都是16字节的块（用户区8字节）
 - ▶ Bin 1... 24
 - ▶ Bin 2... $(2+2)*8 = 32$
 - ▶ Bin 9... $(9+2)*8 = 88$ (用户区80字节)

```
/* The maximum fastbin request size we support */  
#define MAX_FAST_SIZE 80
```

```
#define NFASTBINS (fastbin_index(request2size(MAX_FAST_SIZE))+1)
```



Fast Bin Snapshot

Fastbin快照

从fastbin中分配堆块

```
/*
  If the size qualifies as a fastbin, first check corresponding bin.
  This code is safe to execute even if av is not yet initialized, so we
  can try it without checking, which saves some time on this fast path.
*/

if ((unsigned long)(nb) <= (unsigned long)(av->max_fast)) {
  fb = &(av->fastbins[(fastbin_index(nb))]); //把fb指向对应的bin（卡片箱）
  if ( (victim = *fb) != 0) { //把箱子里的内容给victim，如果不为空说明有内容
    *fb = victim->fd; //把箱子里的内容更新为下一个
    check_reallocated_chunk(av, victim, nb);
    return chunk2mem(victim);
  }
}

#define fastbin(ar_ptr, idx) ((ar_ptr)->fastbinsY[idx])

/* offset 2 to use otherwise unindexable first 2 bins */
#define fastbin_index(sz) \
  (((unsigned int)(sz)) >> (SIZE_SZ == 8 ? 4 : 3)) - 2
```

Fastbin的块上限是可配置的

/*

M_MXFAST is the maximum request size used for "fastbins", special bins that hold returned chunks without consolidating their spaces. This enables future requests for chunks of the same size to be handled very quickly, but **can increase fragmentation**, and thus increase the overall memory footprint of a program.

This malloc manages fastbins very conservatively yet still efficiently, so fragmentation is rarely a problem for values less than or equal to the default. The maximum supported value of MXFAST is 80. You wouldn't want it any higher than this anyway. Fastbins are designed especially for use with many small structs, objects or strings -- the default handles structs/objects/arrays with sizes up

to 8 4byte fields, or small strings representing words, tokens, etc. Using fastbins for larger objects normally worsens fragmentation without improving speed.

M_MXFAST is set in REQUEST size units. It is internally used in chunksize units, which adds padding and alignment. You can reduce M_MXFAST to 0 to disable all use of fastbins. This causes the malloc algorithm to be a closer approximation of fifo-best-fit in all cases, not just for larger requests, but will generally cause it to be slower.

*/

av->max_fast为什么是72而不是88?

```
(gdb) p av->max_fast  
$46 = 72
```

```
#ifndef DEFAULT_MXFAST  
#define DEFAULT_MXFAST 64  
#endif
```

```
(gdb) p global_max_fast  
$8 = 64
```


格物

```
#0 _int_malloc (av=0x80521a0, bytes=8) at malloc.c:3851
#1 0x0804cf49 in malloc (bytes=8) at malloc.c:3350
#2 0xb7269474 in ?? () from /usr/lib/i386-linux-gnu/libxcb.so.1
#3 0xb72697db in ?? () from /usr/lib/i386-linux-gnu/libxcb.so.1
#4 0xb76a1a8a in ?? () from /usr/lib/i386-linux-gnu/libX11.so.6
#5 0xb76a1b8f in ?? () from /usr/lib/i386-linux-gnu/libX11.so.6
#6 0xb76a239f in _XEventsQueued () from /usr/lib/i386-linux-gnu/libX11.so.6
#7 0xb7693138 in XPending () from /usr/lib/i386-linux-gnu/libX11.so.6
#8 0xb7b07f3c in ?? () from /usr/lib/i386-linux-gnu/libgdk-x11-2.0.so.0
#9 0xb79adb3b in g_main_context_check () from /lib/i386-linux-gnu/libglib-2.0.so.0
#10 0xb79ae002 in ?? () from /lib/i386-linux-gnu/libglib-2.0.so.0
#11 0xb79ae52b in g_main_loop_run () from /lib/i386-linux-gnu/libglib-2.0.so.0
#12 0xb7c96bff in gtk_main () from /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0
#13 0x0804a650 in main (argc=1, argv=0xbffff404) at gemalloc.c:301
```

```
(gdb) p nb
$55 = 16
```

块总大小

```
(gdb) p fb
```

```
$56 = (mfastbinptr *) 0x80521bc
```

把fb指向bin

```
(gdb) p av->fastbins[0]
```

```
$57 = (mfastbinptr) 0x80f8f18
```

```
(gdb) p &av->fastbins[0]
```

```
$58 = (mfastbinptr *) 0x80521bc
```

```
(gdb) p *av->fastbins[0]
```

```
$59 = {prev_size = 1145324612, size = 17, fd = 0x80ce370, bk = 0x80f4f30}
```

```
(gdb) p *fb
```

```
$60 = (mfastbinptr) 0x80f8f18
```

```
(gdb) x /xw 0x80521bc
```

```
0x80521bc <main_arena+28>: 0x080f8f18
```

```
(gdb) p /t 17
```

```
$61 = 10001
```

```
(gdb) p victim
```

```
$62 = (mchunkptr) 0x80f8f18
```

```
(gdb) l
```

```
3848     if ((unsigned long)(nb) <= (unsigned long)(av->max_fast)) {
```

```
3849         fb = &(av->fastbins[(fastbin_index(nb))]);
```

```
3850         if ( (victim = *fb) != 0) {
```

```
3851             *fb = victim->fd;
```

将要更新卡片箱

```
3852             check_reallocated_chunk(av, victim, nb);
```

```
3853             return chunk2mem(victim);
```

总大小为16字节，注意，第三位是标志

Bin中目前的块



```
(gdb) p nb
$70 = 40
(gdb) macro expand fastbin_index(40)
expands to: (((unsigned int)(40)) >> 3) - 2
(gdb) p (((unsigned int)(40)) >> 3) - 2
$71 = 3
(gdb) p &av->fastbins[3]
$72 = (mfastbinptr *) 0x80521c8
(gdb) p fb
$73 = (mfastbinptr *) 0x80521c8
(gdb) p av->fastbins[3]
$74 = (mfastbinptr) 0x80f12d8
(gdb) p *fb
$75 = (mfastbinptr) 0x80f12d8
(gdb) p victim
$76 = (mchunkptr) 0x80f12d8
(gdb) p *victim
$77 = {prev_size = 0, size = 41, fd = 0x80cdb60, bk = 0x0}
(gdb) l
3848     if ((unsigned long)(nb) <= (unsigned long)(av->max_fast)) {
3849         fb = &(av->fastbins[(fastbin_index(nb))]);
3850         if ( (victim = *fb) != 0) {
3851             *fb = victim->fd;
3852             check_reallocated_chunk(av, victim, nb);
3853             return chunk2mem(victim);
```

```
#0  _int_malloc (av=0x80521a0, bytes=36) at malloc.c:3350
#1  0x0804cf49 in malloc (bytes=36) at malloc.c:3350
#2  0xb7269394 in ?? () from /usr/lib/i386-linux-gnu/libxc
```

```
(gdb) n
3853         return chunk2mem(victim);
(gdb) p av->fastbins[3]
$79 = (mfastbinptr) 0x80cdb60
```

单步一行后，更新了bin里的内容

常规bin

- 共128个
 - 双向链表管理，每两个元素对应一个bin
 - Bin 1 – Unsorted bin
 - Bin 2 to Bin 63 – Small bin
 - Bin 64 to Bin 126 – Large bin

```
#define NBINS          128
#define NSMALLBINS    64
```

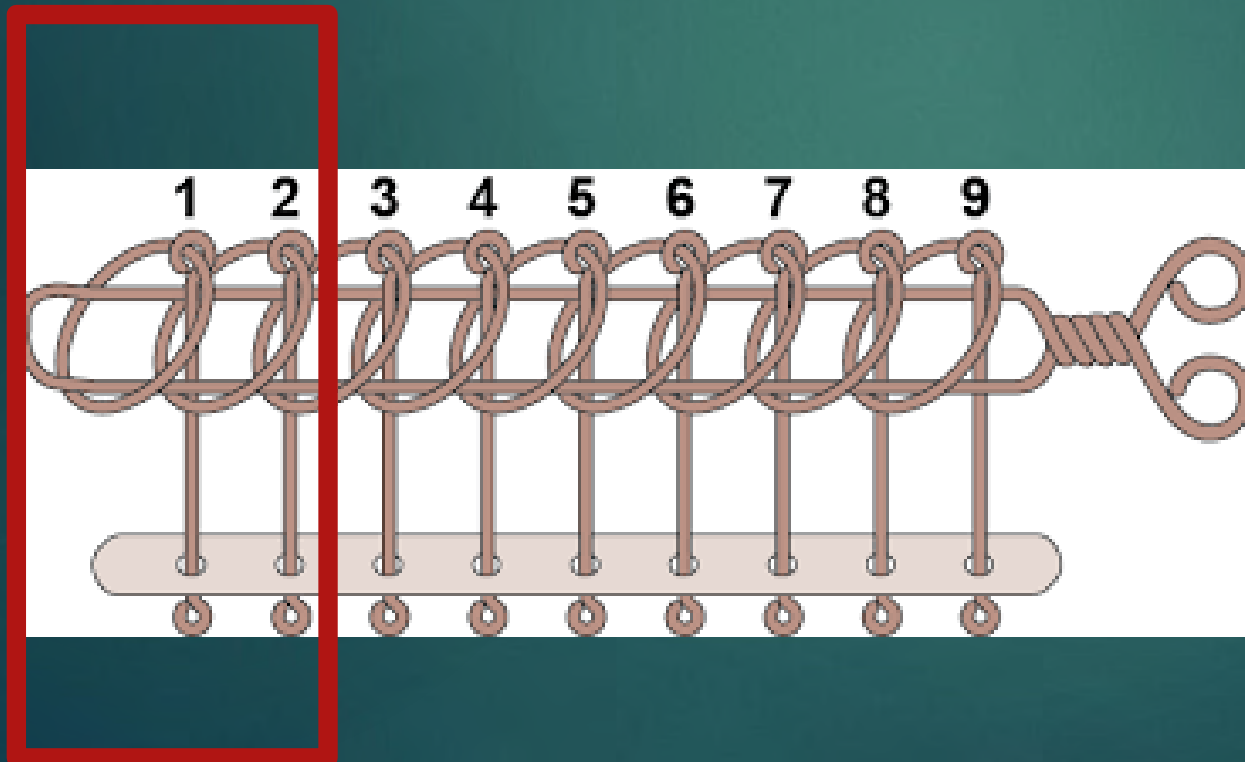
```
(gdb) pt main_arena
type = struct malloc_state {
    mutex_t mutex;
    long int stat_lock_direct;
    long int stat_lock_loop;
    long int stat_lock_wait;
    long int pad0_[1];
    size_t max_fast;

    mfastbinptr fastbins[10];
    mchunkptr top;
    mchunkptr last_remainder;

    mchunkptr bins[256];
    unsigned int binmap[4];
    struct malloc_state *next;
    size_t system_mem;
    size_t max_system_mem;
}
```

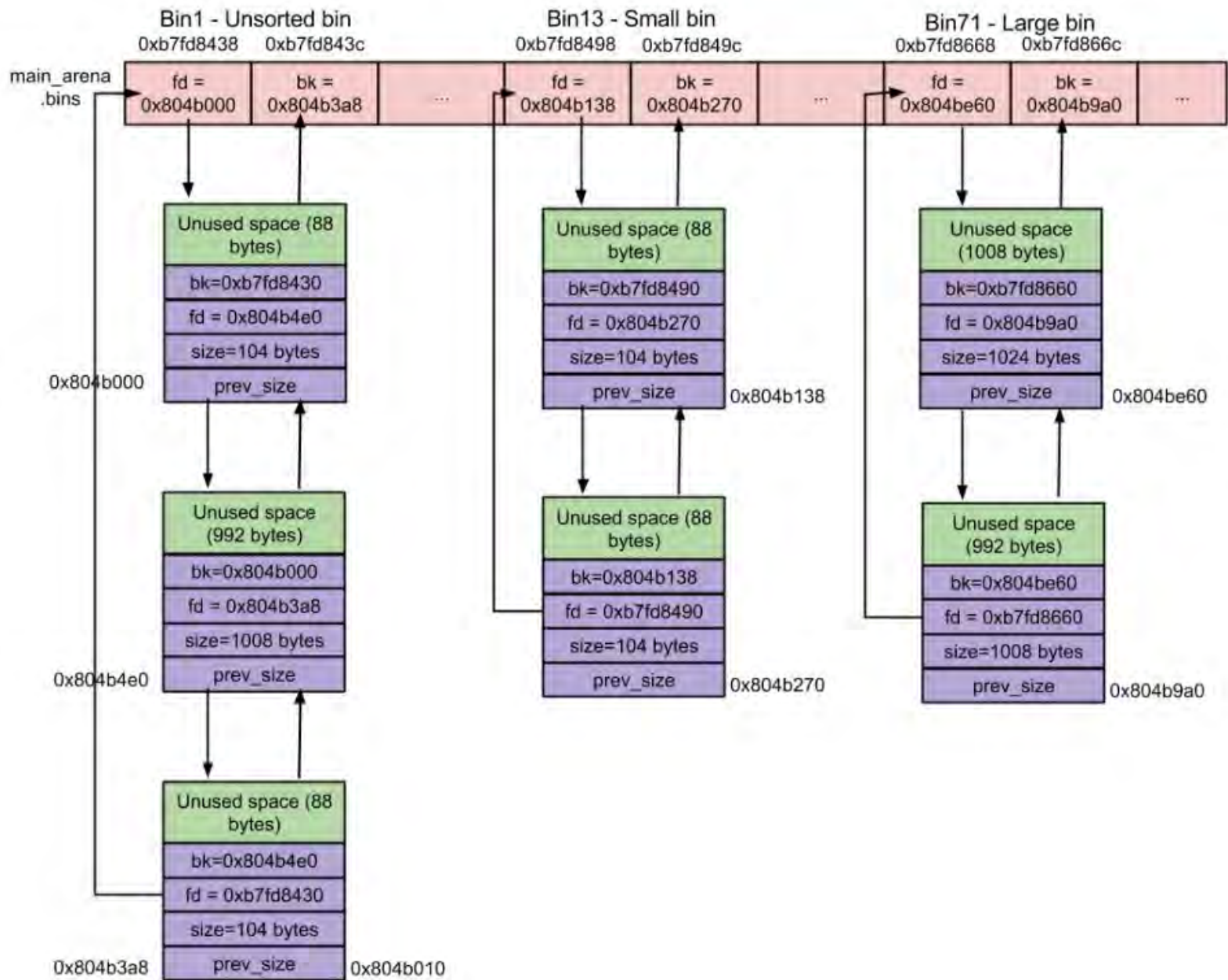

普通bin (regular bin)

- ▶ 双向链表
- ▶ 以数组形式定义，256元素，每个元素是malloc_chunk *
- ▶ 两个数组元素组成一个bin，与其它空闲块的fd和bk指针手拉手串起来



```
(gdb) ptype main_arena.bins[1]
type = struct malloc_chunk {
    size_t prev_size;
    size_t size;
    struct malloc_chunk *fd;
    struct malloc_chunk *bk;
}
```

```
(gdb) p sizeof(main_arena.bins[1])
$8 = 4
(gdb) p sizeof(main_arena.bins)
$9 = 1024
```



从小箱分配

```
if (in_smallbin_range(nb)) {
    idx = smallbin_index(nb);
    bin = bin_at(av,idx);

    if ( (victim = last(bin)) != bin) {
        if (victim == 0) /* initialization check */
            malloc_consolidate(av);
        else {
            bck = victim->bk;
            set_inuse_bit_at_offset(victim, nb);
            bin->bk = bck;
            bck->fd = bin;

            if (av != &main_arena)
                victim->size |= NON_MAIN_ARENA;
            check_malloced_chunk(av, victim, nb);
            return chunk2mem(victim);
        }
    }
}
```

/*

If a small request, check regular bin. Since these "smallbins" hold one size each, no searching within bins is necessary. (For a large request, we need to wait until unsorted chunks are processed to find best fit. But for small ones, fits are exact anyway, so we can check now, which is faster.)

*/

格物

```
(gdb) p nb
$80 = 40
(gdb) p idx
$81 = 5
```

```
(gdb) p av->bins[10]
$85 = (mchunkptr) 0x805220c
(gdb) p av->bins[11]
$86 = (mchunkptr) 0x805220c
```

```
#define SMALLBIN_WIDTH 8
#define MIN_LARGE_SIZE 512
```

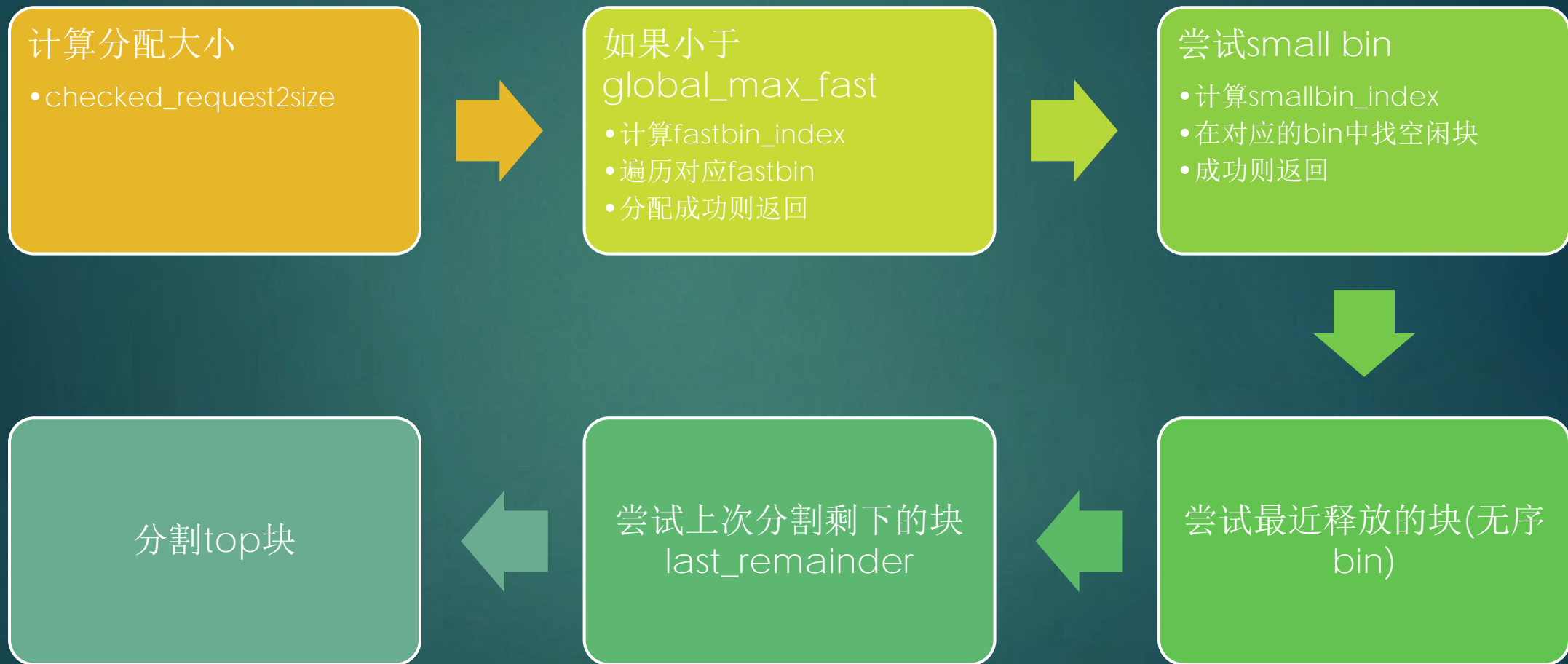
```
#define in_smallbin_range(sz) \
  ((unsigned long)(sz) < (unsigned long)MIN_LARGE_SIZE)
```

```
#define smallbin_index(sz) (((unsigned)(sz)) >> 3)
```

```
#0 _int_malloc (av=0x80521a0, bytes=36) at malloc.c:3918
#1 0x0804cf49 in malloc (bytes=36) at malloc.c:3350
#2 0xb7269394 in ?? () from /usr/lib/i386-linux-gnu/libxcb.so.1
#3 0xb72697db in ?? () from /usr/lib/i386-linux-gnu/libxcb.so.1
```

请求36字节，fastbin不能满足请求，则尝试smallbin，计算出是5号箱，但是为空，于是会尝试无序bin

小块分配过程



跟踪malloc

(gdb) bt

```
#0 _int_malloc (av=0xb7fc7440, bytes=2000) at malloc.c:3902
#1 0xb7e9af2c in __GI___libc_malloc (bytes=2000) at malloc.c:2924
#2 0xb7e9b097 in __GI___libc_malloc (bytes=2000) at malloc.c:2917
#3 0x08048730 in main (argc=3, argv=0xbffff404) at geheap.c:54
```

```
ge@gewubox: ~/work/heap
(gdb) n
3790      assert((unsigned long)(size) >= (unsigned long)(nb));
(gdb) n
3794      /* unlink */
(gdb) n
3791
(gdb) n
3792      remainder_size = size - nb;
(gdb) n
3793
(gdb) n
3799      set_inuse_bit_at_offset(victim, size);
(gdb) n
3801      victim->size |= NON_MAIN_ARENA;
(gdb) n
3803
(gdb) n
3801      victim->size |= NON_MAIN_ARENA;
(gdb) n
3899
```



```
===== After allocating blocks =====
Total non-mmapped bytes (arena):    135168
# of free chunks (ordblks):         1
# of free fastbin blocks (smblocks): 0
# of mapped regions (hblks):        0
Bytes in mapped regions (hblkhd):    0
Max. total allocated space (usmblocks): 0
Free bytes held in fastbins (fsmblocks): 0
Total allocated space (uordblks):    2016
Total free space (fordblks):         133152
Topmost releasable block (keepcost): 133152
```

调用malloc(1000)两次，块头占8字节

```
===== After freeing blocks =====
Total non-mmapped bytes (arena):    135168
# of free chunks (ordblks):         1
# of free fastbin blocks (smblocks): 0
# of mapped regions (hblks):        0
Bytes in mapped regions (hblkhd):    0
Max. total allocated space (usmblocks): 0
Free bytes held in fastbins (fsmblocks): 0
Total allocated space (uordblks):    0
Total free space (fordblks):         135168
Topmost releasable block (keepcost): 135168
```

通过mallinfo接口获取信息

<https://www.systutorials.com/docs/linux/man/3-mallinfo/>

```
(gdb) p *ar_ptr
```

```
$5 = {mutex = 1, flags = 1, fastbinsY = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, top = 0x804b7d8, last_remainder = 0x0, bins = {0xb7fc7470, 0xb7fc7470, 0xb7fc7478, 0xb7fc7478, 0xb7fc7480, 0xb7fc7480, 0xb7fc7488, 0xb7fc7488, 0xb7fc7490, 0xb7fc7490, 0xb7fc7498, 0xb7fc7498, 0xb7fc74a0, 0xb7fc74a0, 0xb7fc74a8, 0xb7fc74a8, 0xb7fc74b0, 0xb7fc74b0, 0xb7fc74b8, 0xb7fc74b8, 0xb7fc74c0, 0xb7fc74c0, 0xb7fc74c8, 0xb7fc74c8, 0xb7fc74d0, 0xb7fc74d0, 0xb7fc74d8, 0xb7fc74d8, 0xb7fc74e0, 0xb7fc74e0, 0xb7fc74e8, 0xb7fc74e8, 0xb7fc74f0, 0xb7fc74f0, 0xb7fc74f8, 0xb7fc74f8, 0xb7fc7500, 0xb7fc7500, 0xb7fc7508, 0xb7fc7508, 0xb7fc7510, 0xb7fc7510, 0xb7fc7518, 0xb7fc7518, 0xb7fc7520, 0xb7fc7520, 0xb7fc7528, 0xb7fc7528, 0xb7fc7530, 0xb7fc7530, 0xb7fc7538, 0xb7fc7538, 0xb7fc7540, 0xb7fc7540, 0xb7fc7548, 0xb7fc7548, 0xb7fc7550, 0xb7fc7550, 0xb7fc7558, 0xb7fc7558, 0xb7fc7560, 0xb7fc7560, 0xb7fc7568, 0xb7fc7568, 0xb7fc7570, 0xb7fc7570, 0xb7fc7578, 0xb7fc7578, 0xb7fc7580, 0xb7fc7580, 0xb7fc7588, 0xb7fc7588, 0xb7fc7590, 0xb7fc7590, 0xb7fc7598, 0xb7fc7598, 0xb7fc75a0, 0xb7fc75a0, 0xb7fc75a8, 0xb7fc75a8, 0xb7fc75b0, 0xb7fc75b0, 0xb7fc75b8, 0xb7fc75b8, 0xb7fc75c0, 0xb7fc75c0, 0xb7fc75c8, 0xb7fc75c8, 0xb7fc75d0, 0xb7fc75d0, 0xb7fc75d8, 0xb7fc75d8, 0xb7fc75e0, 0xb7fc75e0, 0xb7fc75e8, 0xb7fc75e8, 0xb7fc75f0, 0xb7fc75f0, 0xb7fc75f8, 0xb7fc75f8, 0xb7fc7600, 0xb7fc7600, 0xb7fc7608, 0xb7fc7608, 0xb7fc7610, 0xb7fc7610, 0xb7fc7618, 0xb7fc7618, 0xb7fc7620, 0xb7fc7620, 0xb7fc7628, 0xb7fc7628, 0xb7fc7630, 0xb7fc7630, 0xb7fc7638, 0xb7fc7638, 0xb7fc7640, 0xb7fc7640, 0xb7fc7648, 0xb7fc7648, 0xb7fc7650, 0xb7fc7650, 0xb7fc7658, 0xb7fc7658, 0xb7fc7660, 0xb7fc7660, 0xb7fc7668, 0xb7fc7668, 0xb7fc7670, 0xb7fc7670, 0xb7fc7678, 0xb7fc7678, 0xb7fc7680, 0xb7fc7680, 0xb7fc7688, 0xb7fc7688, 0xb7fc7690, 0xb7fc7690, 0xb7fc7698, 0xb7fc7698, 0xb7fc76a0, 0xb7fc76a0, 0xb7fc76a8, 0xb7fc76a8, 0xb7fc76b0, 0xb7fc76b0, 0xb7fc76b8, 0xb7fc76b8, 0xb7fc76c0, 0xb7fc76c0, 0xb7fc76c8, 0xb7fc76c8, 0xb7fc76d0, 0xb7fc76d0, 0xb7fc76d8, 0xb7fc76d8, 0xb7fc76e0, 0xb7fc76e0, 0xb7fc76e8, 0xb7fc76e8, 0xb7fc76f0, 0xb7fc76f0, 0xb7fc76f8, 0xb7fc76f8, 0xb7fc7700, 0xb7fc7700, 0xb7fc7708, 0xb7fc7708, 0xb7fc7710, 0xb7fc7710, 0xb7fc7718, 0xb7fc7718, 0xb7fc7720, 0xb7fc7720, 0xb7fc7728, 0xb7fc7728, 0xb7fc7730, 0xb7fc7730, 0xb7fc7738, 0xb7fc7738, 0xb7fc7740, 0xb7fc7740, 0xb7fc7748, 0xb7fc7748, 0xb7fc7750, 0xb7fc7750, 0xb7fc7758, 0xb7fc7758, 0xb7fc7760, 0xb7fc7760, 0xb7fc7768, 0xb7fc7768, 0xb7fc7770, 0xb7fc7770, 0xb7fc7778, 0xb7fc7778, 0xb7fc7780, 0xb7fc7780, 0xb7fc7788, 0xb7fc7788...}, binmap = {0, 0, 0, 0}, next = 0xb7fc7440, next_free = 0x0, system_mem = 135168, max_system_mem = 135168}
```

free

看人下菜碟儿

小于max_fast

- 放入fastbin
- 快速释放
- 嫌弃态度：就这么一点点空间，值得我弄一会么？

超大块

- 直接还给内核
- 大客户是偶尔的，不要搞不清自己的身份

介于中间的

- 标记为free
- 放入无序bin
- 顺便合并空闲块
- 如果临近top，则更新top
- 根据需要修剪

```
#0 free (mem=0x80f3d48) at malloc.c:3385
#1 0xb72682dd in ?? () from /usr/lib/i386-linux-gnu/libxcb.so.1
#2 0xb72697e6 in ?? () from /usr/lib/i386-linux-gnu/libxcb.so.1
#3 0xb76a1a8a in ?? () from /usr/lib/i386-linux-gnu/libX11.so.6
#4 0xb76a1b8f in ?? () from /usr/lib/i386-linux-gnu/libX11.so.6
...
#8 0xb79adb3b in g_main_context_check () from /lib/i386-linux-gnu/libglib-2.0.so.0
#9 0xb79ae002 in ?? () from /lib/i386-linux-gnu/libglib-2.0.so.0
#10 0xb79ae52b in g_main_loop_run () from /lib/i386-linux-gnu/libglib-2.0.so.0
#11 0xb7c96bff in gtk_main () from /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0
#12 0x0804a650 in main (argc=1, argv=0xbffff404) at gemalloc.c:301
```

```
(gdb) n
```

```
3387     if (hook != NULL) {
```

```
(gdb)
```

```
3392     if (mem == 0)                /* free(0) has no effect */
```

```
(gdb)
```

```
3395     p = mem2chunk(mem);
```

```
(gdb)
```

```
3398     if (chunk_is_mmapped(p))      /* release mmapped memory. */
```

```
(gdb) p p
```

```
$90 = (mchunkptr) 0x80f3d40
```

```
(gdb) p *p
```

```
$91 = {prev_size = 55, size = 17, fd = 0x80f33c8, bk = 0x80ce818}
```

续

(gdb) n

```
3405     ar_ptr = arena_for_chunk(p);
```

(gdb) s

```
3407     if(!mutex_trylock(&ar_ptr->mutex))
```

(gdb) macro expand arena_for_chunk(p)

expands to: (((p)->size & 0x4) ? ((heap_info *)((unsigned long)(p) & ~((1024*1024)-1)))->ar_ptr : &main_a

(gdb) n

```
3408     ++(ar_ptr->stat_lock_direct);
```

(gdb)

```
3416     _int_free(ar_ptr, mem);
```


_int_free – 释放到fastbin

```
(gdb) s
_int_free (av=0x80521a0, mem=0x80f3d48) at malloc.c:4206
4206     if (mem != 0) {
(gdb) n
4207     p = mem2chunk(mem);
(gdb)
4208     size = chunksize(p);
(gdb)
4217     if ((unsigned long)(size) <= (unsigned long)(av->max_fast)
(gdb)
4228     set_fastchunks(av);
(gdb)
4229     fb = &(av->fastbins[fastbin_index(size)]);
(gdb)
4230     p->fd = *fb;
(gdb)
4231     *fb = p;
(gdb)
4347 }
```

链表更新好就
OK了，不必设
置为空闲块

set_fastchunks

```
(gdb) macro expand set_fastchunks(av)
expands to: ((av)->max_fast &= ~(1U))
(gdb) p av->max_fast
$95 = 72
(gdb) n
4228      set_fastchunks(av);
(gdb)
4229      fb = &(av->fastbins[fastbin_index(size)]);
(gdb) p av->max_fast
$96 = 72
```

Max_fast的最低两位是标志位

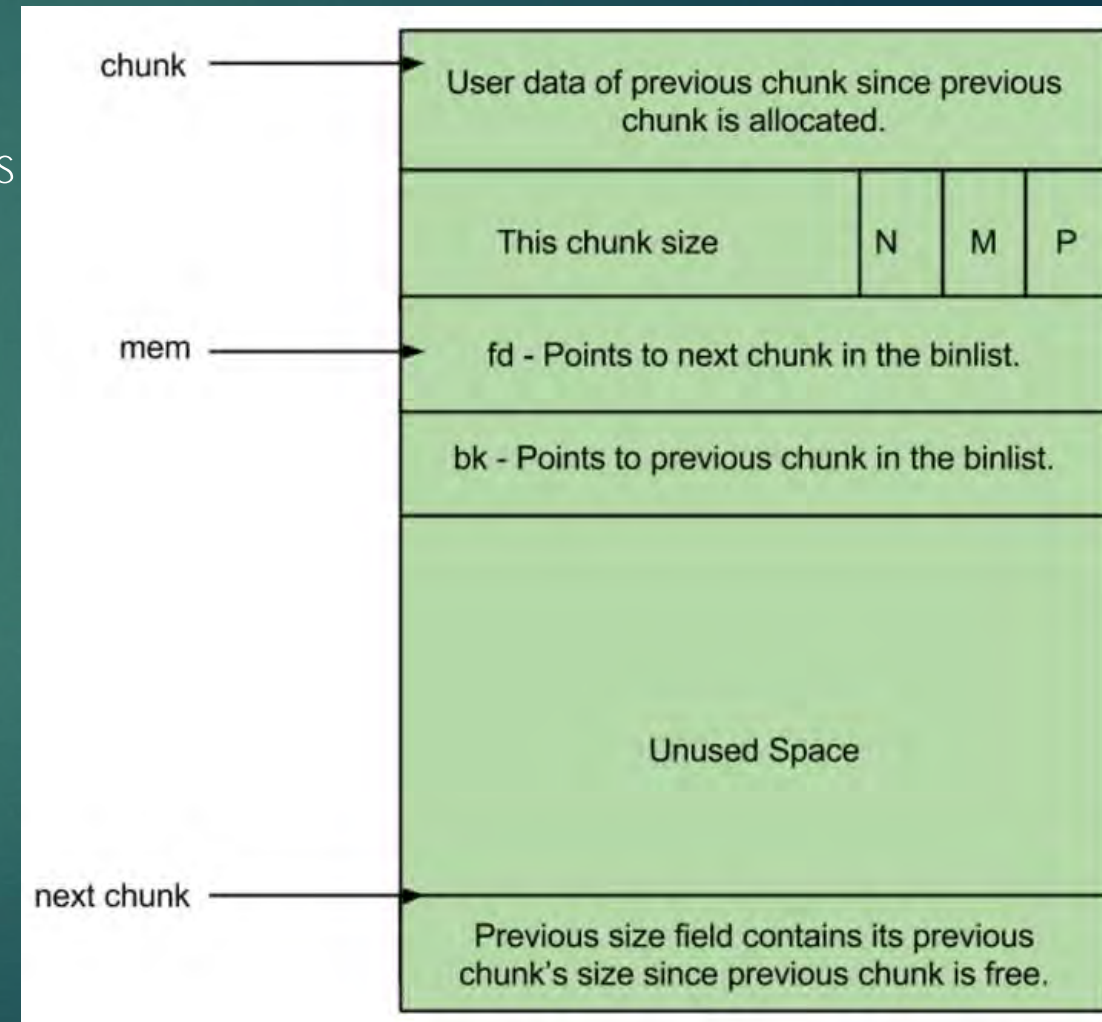
直接返还超大块

```
/*
  If the chunk was allocated via mmap, release via munmap(). Note
  that if HAVE_MMAP is false but chunk_is_mmapped is true, then
  user must have overwritten memory. There's nothing we can do to
  catch this error unless MALLOC_DEBUG is set, in which case
  check_inuse_chunk (above) will have triggered error.
*/

else {
#ifdef HAVE_MMAP
  int ret;
  INTERNAL_SIZE_T offset = p->prev_size;
  mp_.n_mmaps--;
  mp_.mmapped_mem -= (size + offset);
  ret = munmap((char*)p - offset, size + offset);
  /* munmap returns non-zero on failure */
  assert(ret == 0);
#endif
}
```


Free Chunk

- ▶ prev_size: No two free chunks can be adjacent together. When both the chunks are free, its gets combined into one single free chunk. Hence always previous chunk to this freed chunk would be allocated and therefore prev_size contains previous chunk's user data.
- ▶ size: This field contains the size of this free chunk.
- ▶ fd: Forward pointer – Points to next chunk in the same bin (and NOT to the next chunk present in physical memory).
- ▶ bk: Backward pointer – Points to previous chunk in the same bin (and NOT to the previous chunk present in physical memory).



```

nextchunk-> +-----+
|           Size of chunk           |
+-----+

```

Where "chunk" is the front of the chunk for the purpose of most of the malloc code, but "mem" is the pointer that is returned to the user. "Nextchunk" is the beginning of the next contiguous chunk.

Chunks always begin on even word boundaries, so the mem portion (which is returned to the user) is also on an even word boundary, and thus at least double-word aligned.

Free chunks are stored in circular doubly-linked lists, and look like this:

```

chunk-> +-----+
|           Size of previous chunk   |
+-----+
`head:' |           Size of chunk, in bytes           |P|
mem->   +-----+
|           Forward pointer to next chunk in list     |
+-----+
|           Back pointer to previous chunk in list    |
+-----+
|           Unused space (may be 0 bytes long)        |
:                                                     |
:                                                     |
nextchunk-> +-----+
`foot:' |           Size of chunk, in bytes           |
+-----+

```

The P (PREV_INUSE) bit, stored in the unused low-order bit of the chunk size (which is always a multiple of two words), is an in-use bit for the *previous* chunk. If that bit is *clear*, then the word before the current chunk size contains the previous chunk size, and can be used to find the front of the previous chunk. The very first chunk allocated always has this bit set, preventing access to non-existent (or non-owned) memory. If

▶ malloc.c