Inside ptmalloc2

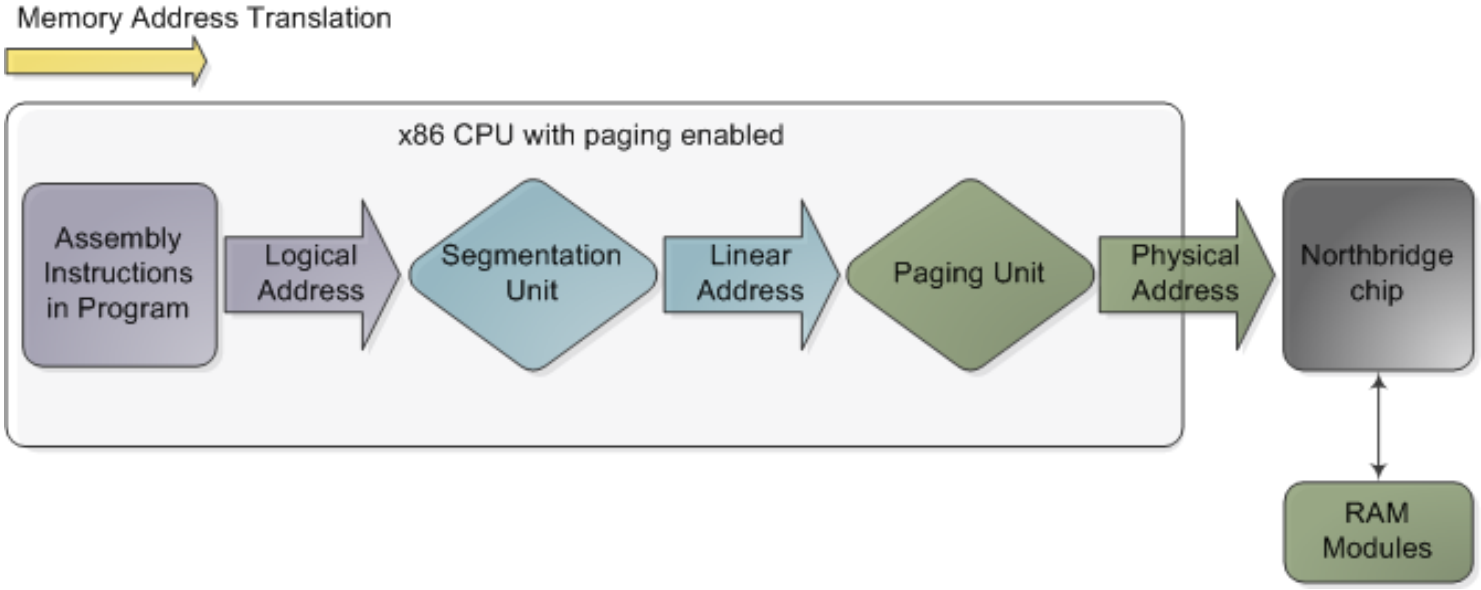
Peng Xu
peng.p.xu@ericsson.com

Sep 14, 2013

Part I

basic concept and data structure

Memory Translation



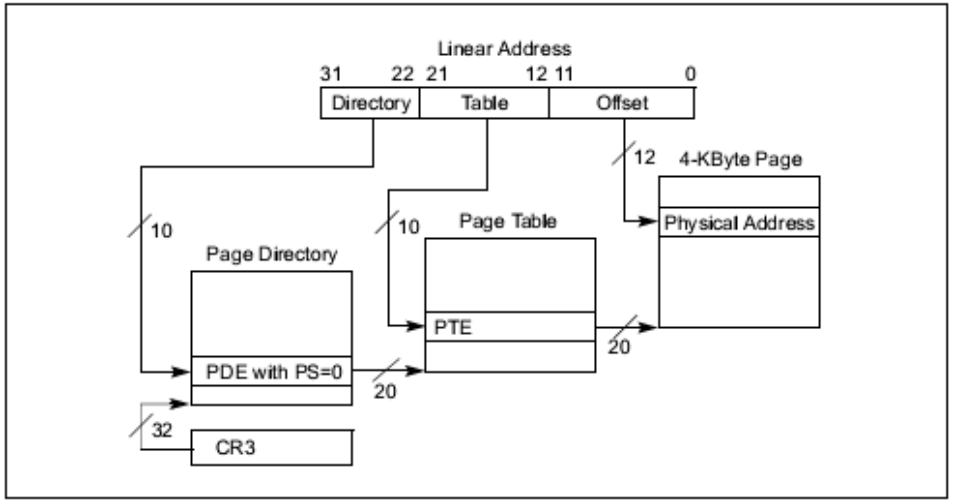
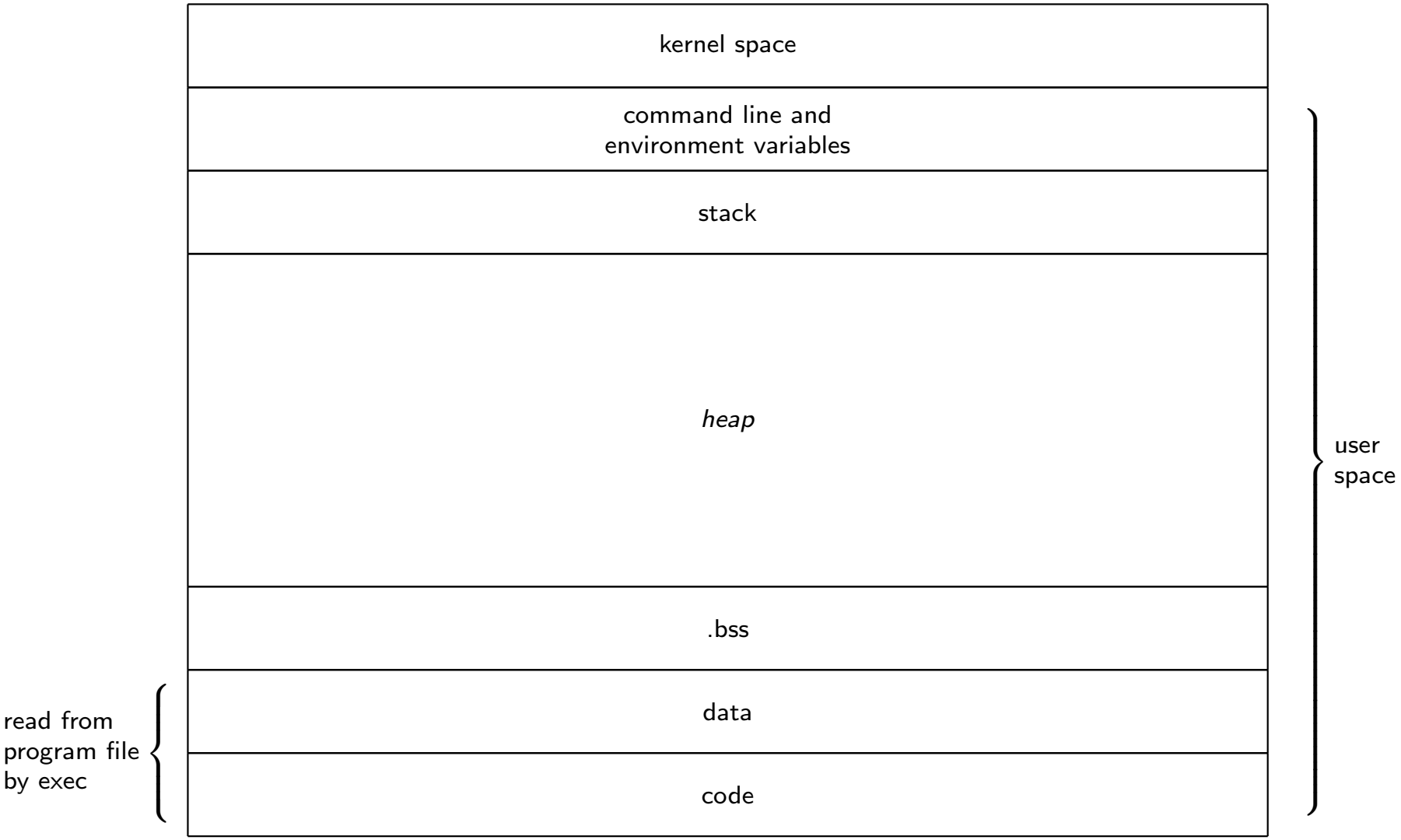


Figure 4-2. Linear-Address Translation to a 4-KByte Page using 32-Bit Paging

process memory layout



System call for memory management

system call list

1. brk
2. sbrk is a wrapper of brk
3. mmap

why it's not a good idea to call these system call in app directly

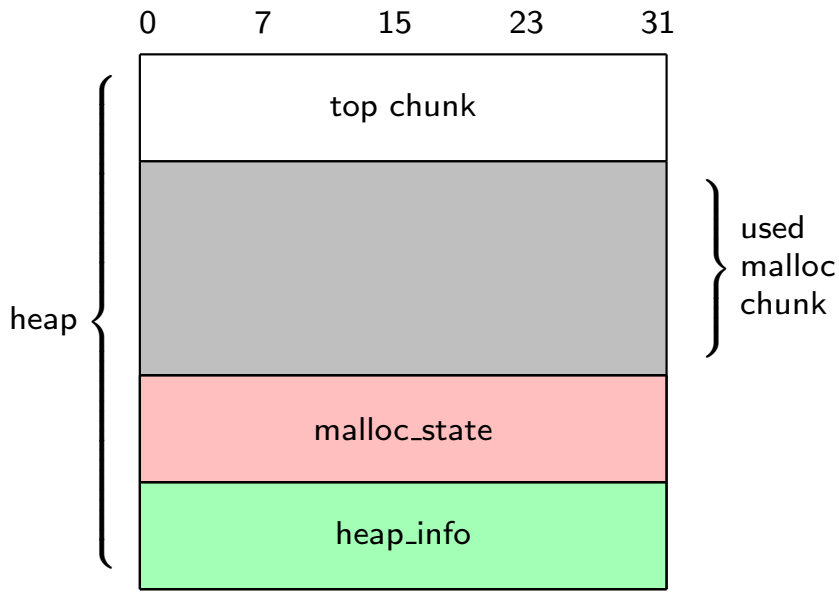
1. System call is very expensive.
2. Glibc can balance the time and space complexity very well.

test.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(void) {
  char* orig_heap_end = (char*)sbrk(0);
  char* cur_heap_end = (char*)sbrk(20);
  sprintf(orig_heap_end,"%s","hello");
  return 0;
}
```

heap layout after first malloc

```
1  #include <stdlib.h>
2  int main(void) {
3    char* ptr = (char*)malloc(1);
4    free(ptr);
5    return 0;
6  }
```

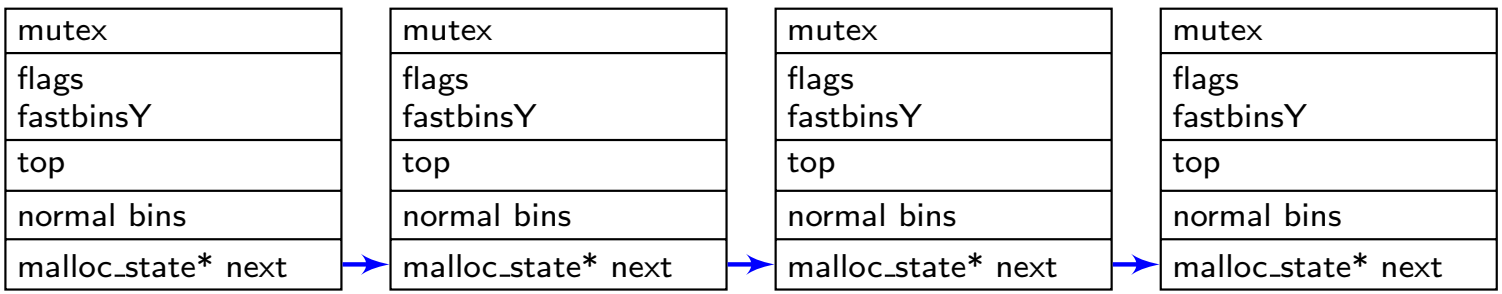


strace output

```
execve("./test", ["./test"], [/* 47 vars */]) = 0
brk(0)                                  = 0x9dd9000
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7714000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=132397, ...}) = 0
mmap2(NULL, 132397, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb76f3000
close(3)                                = 0
open("/usr/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\3\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0000\233\1\0004\0\0\0"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=9127208, ...}) = 0
mmap2(NULL, 1763972, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xb7544000
mprotect(0xb76ec000, 4096, PROT_NONE)   = 0
mmap2(0xb76ed000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1a8000) = 0xb76ec
mmap2(0xb76f0000, 10884, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xb76f0000
close(3)                                = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7543000
set_thread_area({entry_number:-1 -> 6, base_addr:0xb7543700, limit:1048575, seg_32bit:1, contents:0, read
mprotect(0xb76ed000, 8192, PROT_READ)   = 0
mprotect(0xb7736000, 4096, PROT_READ)   = 0
munmap(0xb76f3000, 132397)              = 0
brk(0)                                  = 0x9dd9000
brk(0x9dfa000)                          = 0x9dfa000
exit_group(0)                           = ?
+++ exited with 0 +++
```

data structure of malloc_state

```
struct malloc_state {
  mutex_t mutex;
  int flags;
  mfastbinptr      fastbinsY[NFASTBINS];
  /* Base of the topmost chunk -- not otherwise kept in a bin */
  mchunkptr        top;
  /* The remainder from the most recent split of a small request */
  mchunkptr        last_remainder;
  /* Normal bins packed as described above */
  mchunkptr        bins[NBINS * 2 - 2];
  unsigned int     binmap[BINMAPSIZE];
  struct malloc_state *next;
  /* Memory allocated from the system in this arena.  */
  INTERNAL_SIZE_T system_mem;
  INTERNAL_SIZE_T max_system_mem;
};
```



data structure of heap_info

```
typedef struct _heap_info {
  mstate ar_ptr; /* Arena for this heap. */
  struct _heap_info *prev; /* Previous heap. */
  size_t size;   /* Current size in bytes. */
  size_t mprotect_size; /* Size in bytes that has been mprotected
   PROT_READ|PROT_WRITE.  */
  /* Make sure the following data is properly aligned, particularly
     that sizeof (heap_info) + 2 * SIZE_SZ is a multiple of
     MALLOC_ALIGNMENT. */
  char pad[-6 * SIZE_SZ & MALLOC_ALIGN_MASK];
} heap_info;
```

Question

why are there prev in the data structure?

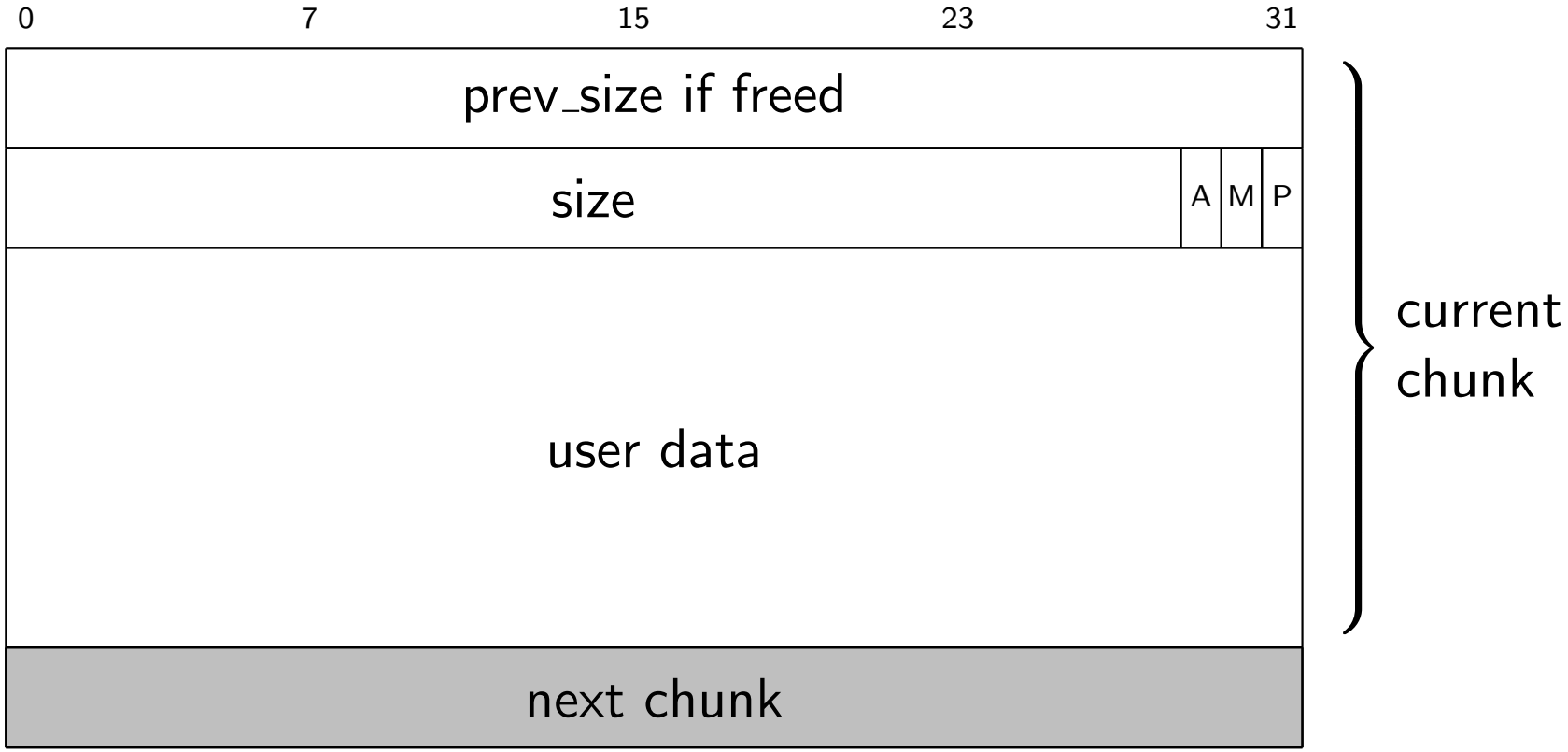
data structure of malloc_chunk

```
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;
};
```

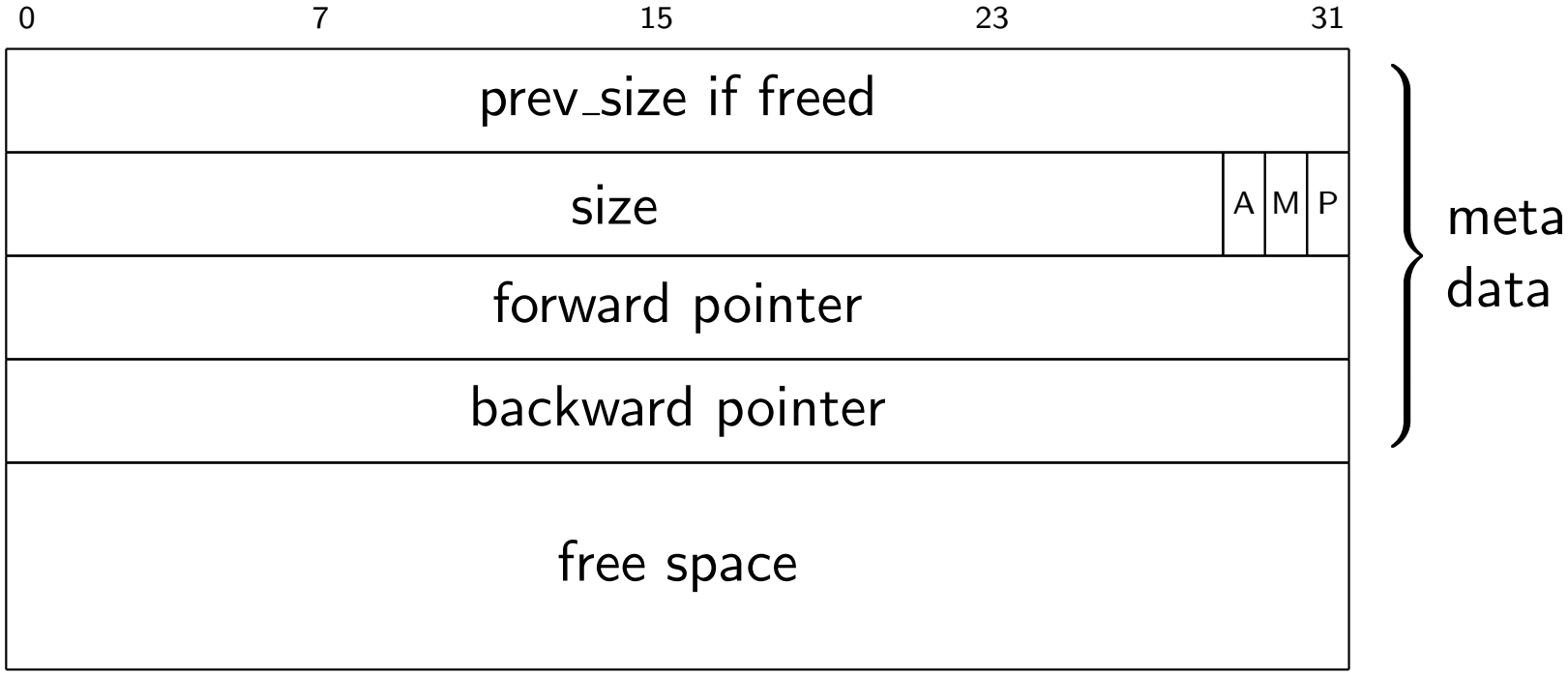
chunk structure of allocated one



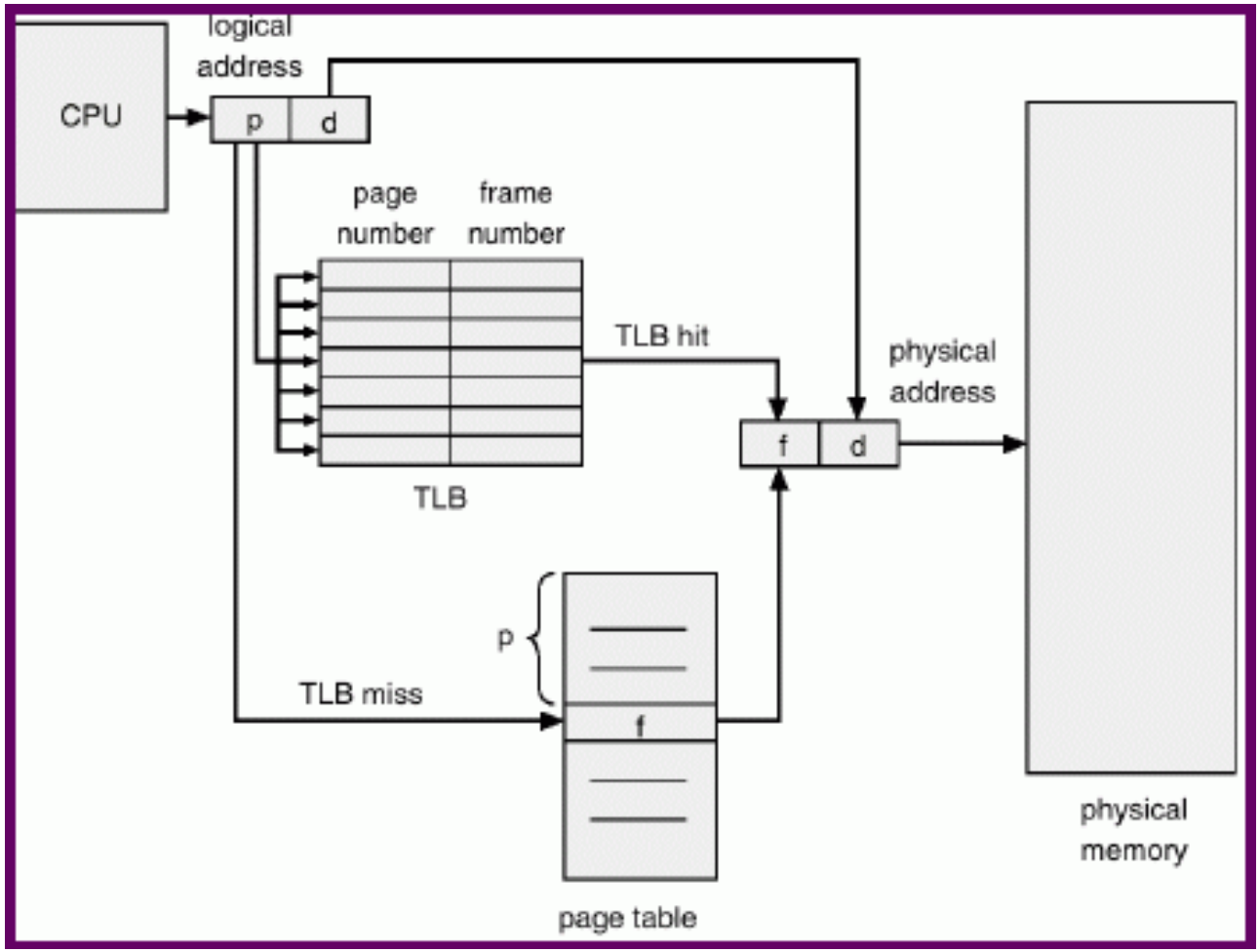
prev_size

If the previous chunk is freed, the value of prev_size field states the previous chunk size in byte. Otherwise, the value in previous field is *just one part of user data* in the previous chunk.

chunk structure of freed one



TLB and Cache

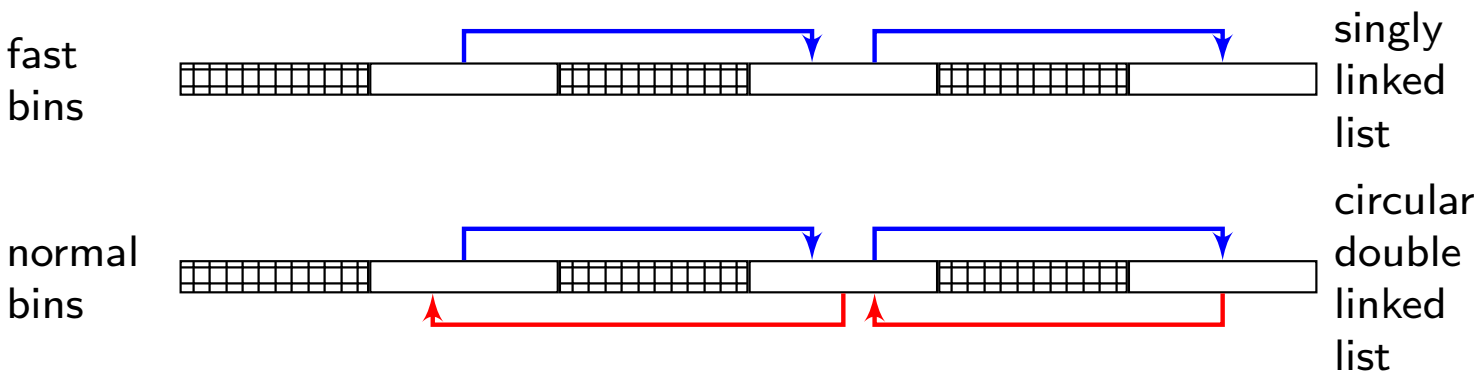


Organization of the freed chunk

The freed chunk will be cached. The pointer will be saved in suitable bins. There are three types of bins.

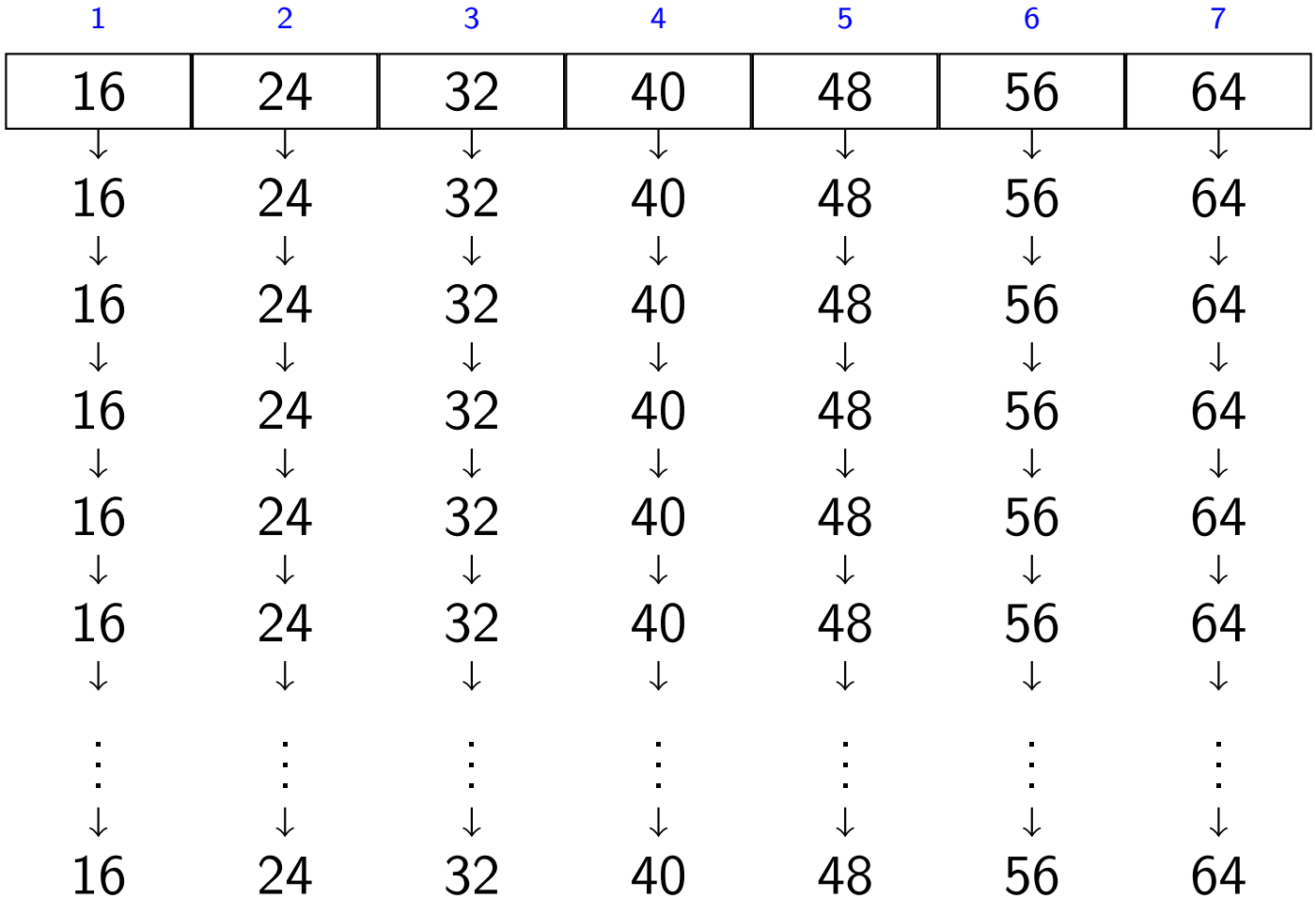
- fast bin $<$ 64 bytes
- small bin $<$ 512 bytes
- large bin $>$ 512 bytes

Small bin and large bin are also called *normal bins*.

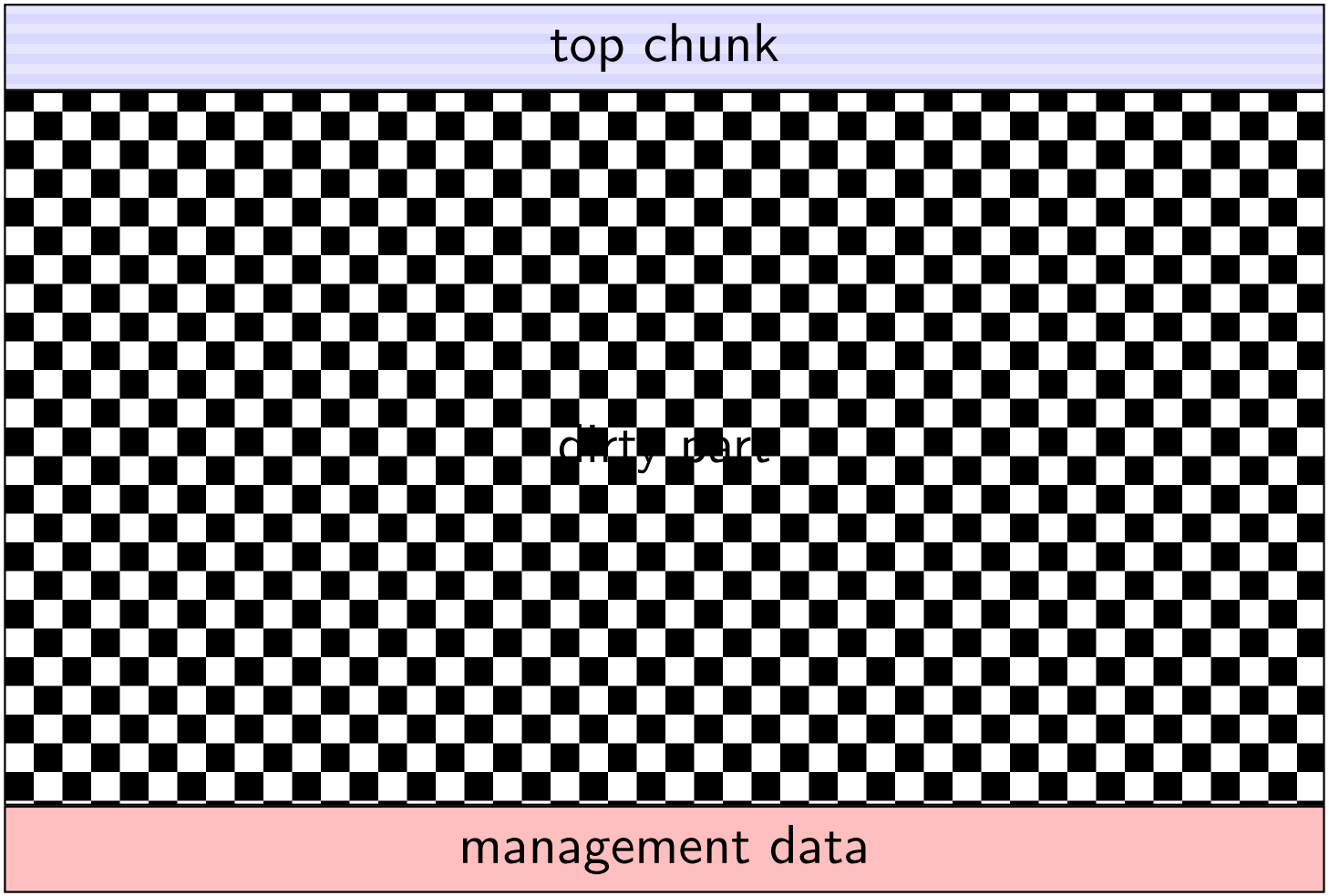


Fastbins

Fastbin is a singly linked list.



Summary of heap layout



Part II

Algorithm

algorithm for malloc

Version is glibc-2.18

basic steps for malloc

1. fetch an arena
   - one area is free
   - create a new one if non-area is in idle
2. allocate requested size in the fetched area
   - return the requested memory
   - return NULL if non-memory available

malloc function

```
__libc_malloc(size_t bytes)
{
  mstate ar_ptr;
  void *victim;
  void *(*hook) (size_t, const void *)
    = 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) {
    ar_ptr = arena_get_retry(ar_ptr, bytes);
    if (__builtin_expect(ar_ptr != NULL, 1)) {
      victim = _int_malloc(ar_ptr, bytes);
      (void)mutex_unlock(&ar_ptr->mutex);
    }
  } else
    (void)mutex_unlock(&ar_ptr->mutex);
  assert(!victim || chunk_is_mmapped(mem2chunk(victim)) ||
 ar_ptr == arena_for_chunk(mem2chunk(victim)));
  return victim;
}
```

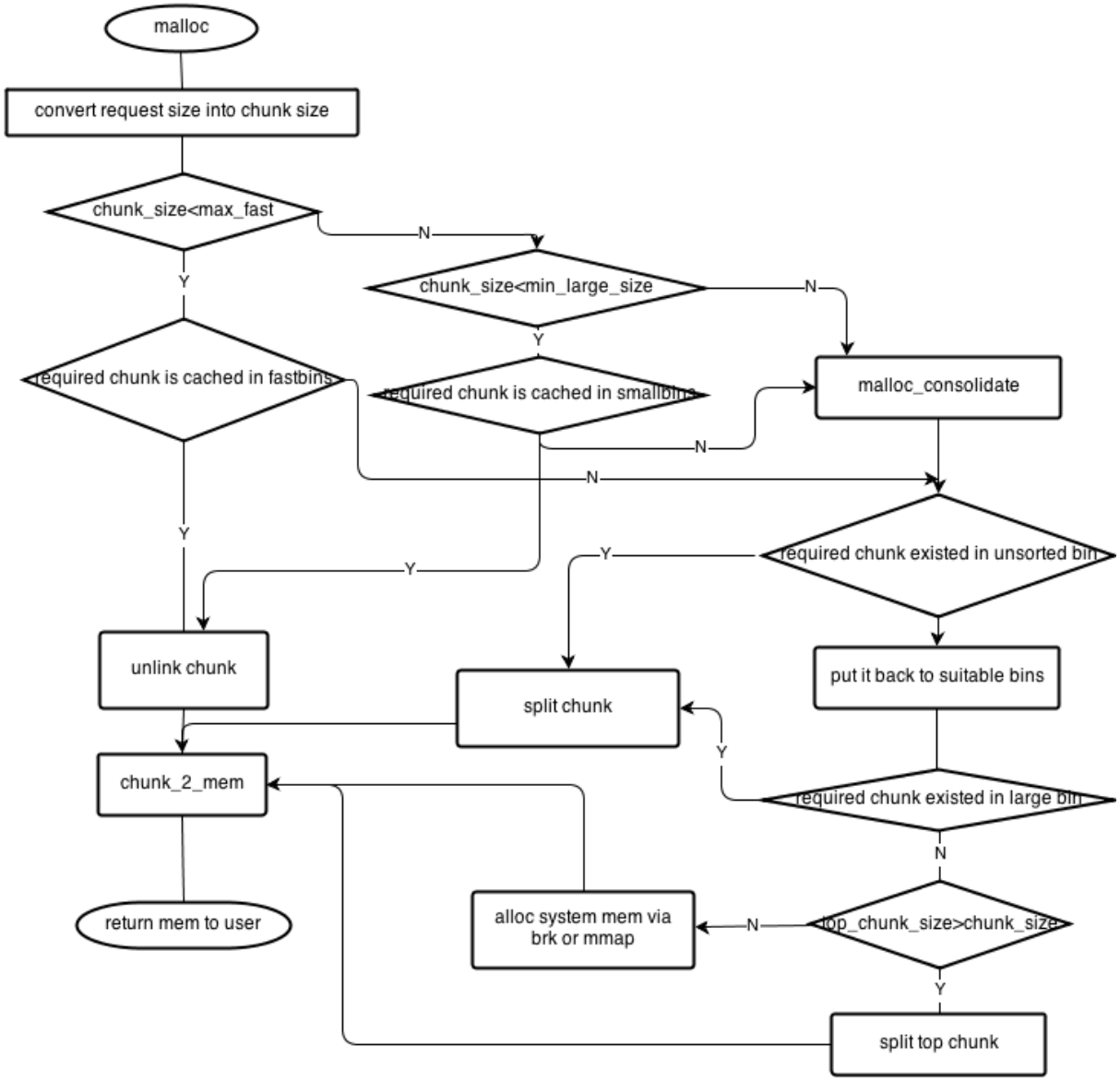
malloc procedure in one specific area

exact match fastbins and smallbins

closest match largebins

- fetch freed chunk in the matched bins if the request size falls into the fastbins or smallbins
- largebin
  - do malloc_consolidate, consolidated memory is put into unsorted bins
  - do alloation
- if both of above steps failed, try to apply a new memory block from system

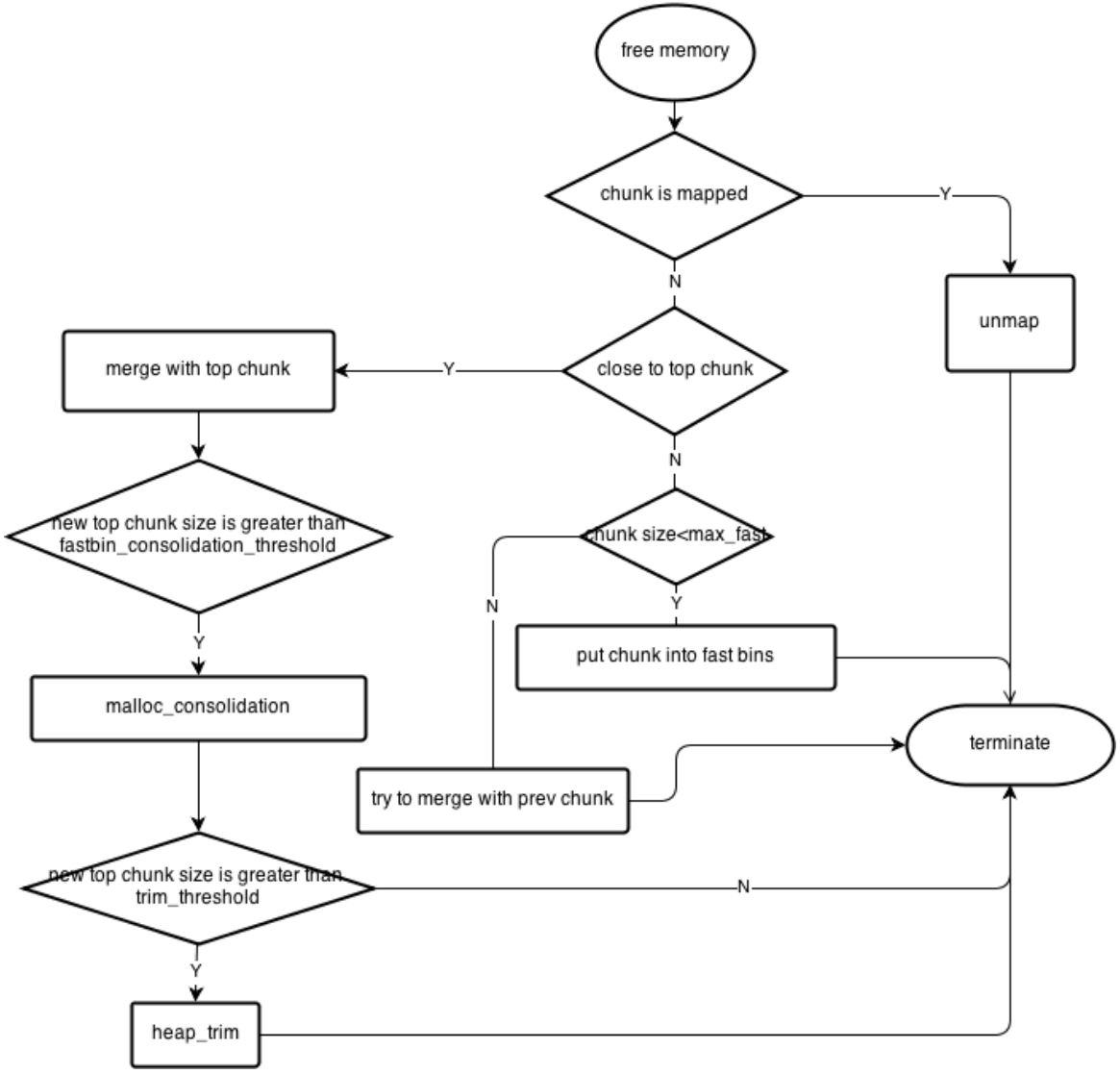
malloc flow chart



algorithm for free memory

1. free to system directory if the chunk is allocated using mmap
2. free to fastbins directory and return
3. free to normal bins and do malloc_consolidate
4. if the top chunk is big enough, do heap trim or heap shrink

memory free flow chart



function list

- libc_malloc
- _int_malloc
- _int_free
- malloc_consolidate
- sysmalloc

Part III

common memory error

why memory heap corruption

The basic reason is that the meta data is corrupted.
Overflow will be the main reason.
Memory restriction condition is not matched.

how to detect and solve memory leak

1. purify
2. valgrind
3. tcmalloc