Tips of malloc & free

Making your own malloc library for troubleshooting

2013.2.22 Embedded Linux Conference

Tetsuyuki Kobayashi

- The latest version of this slide will be available from here
- http://www.slideshare.net/tetsu.koba/presentations

Who am I?

- 20+ years involved in embedded systems
  - 10 years in real time OS, such as iTRON
  - 10 years in embedded Java Virtual Machine
  - Now GCC, Linux, QEMU, Android, …
- Blogs
  - http://d.hatena.ne.jp/embedded/ (Personal)
  - http://blog.kmckk.com/ (Corporate)
  - http://kobablog.wordpress.com/(English)
- Twitter
  - @tetsu_koba

Today's topics

- Prologue: Making your own malloc library for troubleshooting
- System calls to allocate memory in user space
- Tips of glibc's malloc
- How to hook and replace malloc (and pitfalls I fell)
- dlmalloc

Prologue:
Making your own malloc library for troubleshooting

Typical troubles of heap memory

- Corruption
  - crashed by SEGV at malloc or free. looks malloc bug, but **NOT**
  - Who actually destroy heap?
- Leaking
  - malloc'ed but not free'ed
  - damages silently
- You want additional checking and logging in malloc/free

Wrapping macro/fuction

- #define malloc(x) debug_malloc(x)
- Useful. But you can't cover all malloc calling because ...

Explicit call for malloc

- many standard library functions use malloc internally
  - example) sprintf
- C++ new operator uses malloc internelly

Modify glibc(libc.so) directly?

- libc source package is quite large
- If you replace libc.so, it affects whole system
  - not only for the debugee process

So I did was

- making my own malloc library
  - easy to modify
  - use this only for the debugee process

System calls to allocate memory in user space

System calls to allocate memory in user space

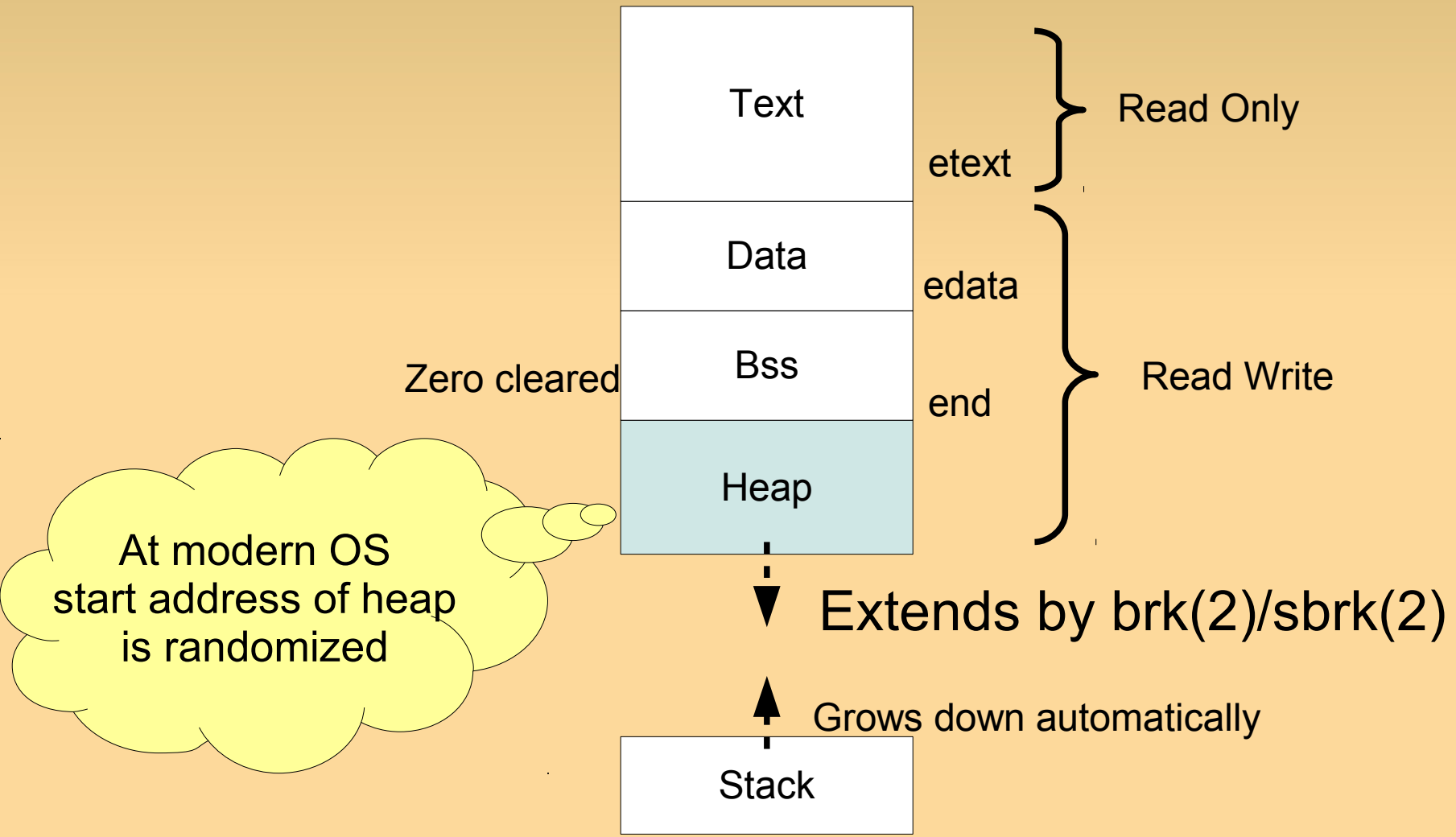
- You need system call to allocate in user space when you make your own malloc library
- There are 2 types of them
  - brk/sbrk
  - mmap/munmap/mremap

brk/sbrk

- exists from ancient Unix
  - before virtual memory system
- extends data segment
- standard malloc library use these system calls
- You should not use these system calls if your own malloc library co-exist with standard malloc library

brk/sbrk extends data segment

Memory map of user process on old simple UNIX



cat /proc/self/maps

You see memory map of 'cat /proc/self/maps' itself

This is heap area

```
$ cat /proc/self/maps
00400000-0040d000 r-xp 00000000 08:01 1048675                            /bin/cat
0060d000-0060e000 r--p 0000d000 08:01 1048675                            /bin/cat
0060e000-0060f000 rw-p 0000e000 08:01 1048675                            /bin/cat
01a7a000-01a9b000 rw-p 00000000 00:00 0                                  [heap]
7f10f05d0000-7f10f074d000 r-xp 00000000 08:01 316763                     /lib/libc-2.11.1.so
7f10f074d000-7f10f094c000 ---p 0017d000 08:01 316763                     /lib/libc-2.11.1.so
7f10f094c000-7f10f0950000 r--p 0017c000 08:01 316763                     /lib/libc-2.11.1.so
7f10f0950000-7f10f0951000 rw-p 00180000 08:01 316763                     /lib/libc-2.11.1.so
7f10f0951000-7f10f0956000 rw-p 00000000 00:00 0
7f10f0956000-7f10f0976000 r-xp 00000000 08:01 272407                     /lib/ld-2.11.1.so
7f10f09fa000-7f10f0a39000 r--p 00000000 08:01 1580725                    /usr/lib/locale/en_US.utf8/LC_CTYPE
7f10f0a39000-7f10f0b57000 r--p 00000000 08:01 1580503                    /usr/lib/locale/en_US.utf8/LC_COLLATE
7f10f0b57000-7f10f0b5a000 rw-p 00000000 00:00 0
7f10f0b62000-7f10f0b63000 r--p 00000000 08:01 1580587                    /usr/lib/locale/en_US.utf8/LC_NUMERIC
7f10f0b63000-7f10f0b64000 r--p 00000000 08:01 1583228                    /usr/lib/locale/en_US.utf8/LC_TIME
7f10f0b64000-7f10f0b65000 r--p 00000000 08:01 1583229                    /usr/lib/locale/en_US.utf8/LC_MONETARY
7f10f0b65000-7f10f0b66000 r--p 00000000 08:01 1583230                    /usr/lib/locale/en_US.utf8/LC_MESSAGES/SYS_LC_MESSAGES
7f10f0b66000-7f10f0b67000 r--p 00000000 08:01 1580575                    /usr/lib/locale/en_US.utf8/LC_PAPER
7f10f0b67000-7f10f0b68000 r--p 00000000 08:01 1580573                    /usr/lib/locale/en_US.utf8/LC_NAME
7f10f0b68000-7f10f0b69000 r--p 00000000 08:01 1583231                    /usr/lib/locale/en_US.utf8/LC_ADDRESS
7f10f0b69000-7f10f0b6a000 r--p 00000000 08:01 1583232                    /usr/lib/locale/en_US.utf8/LC_TELEPHONE
7f10f0b6a000-7f10f0b6b000 r--p 00000000 08:01 1580571                    /usr/lib/locale/en_US.utf8/LC_MEASUREMENT
7f10f0b6b000-7f10f0b72000 r--s 00000000 08:01 1623537                    /usr/lib/gconv/gconv-modules.cache
7f10f0b72000-7f10f0b73000 r--p 00000000 08:01 1583233                    /usr/lib/locale/en_US.utf8/LC_IDENTIFICATION
7f10f0b73000-7f10f0b75000 rw-p 00000000 00:00 0
7f10f0b75000-7f10f0b76000 r--p 0001f000 08:01 272407                     /lib/ld-2.11.1.so
7f10f0b76000-7f10f0b77000 rw-p 00020000 08:01 272407                     /lib/ld-2.11.1.so
7f10f0b77000-7f10f0b78000 rw-p 00000000 00:00 0
7fff80929000-7fff8093e000 rw-p 00000000 00:00 0                          [stack]
7fff809ff000-7fff80a00000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

mmap/munmap/mremap

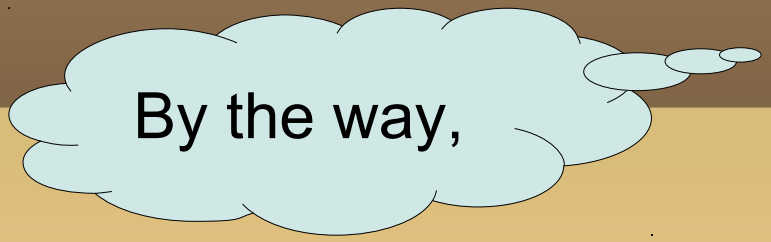
- newer system calls than brk/sbrk
  - integrate memory and file mapping
- Glibc's malloc also use these when large chunk (>= 128KB: default) required
- Use these when you implement your own malloc library

Usage of mmap(2)

```
addr = mmap(NULL, size, PROT_READ|PROT_WRITE,
            MAP_PRIVATE|MAP_ANONYMOUS, 0, 0);
if (MAP_FAILED == addr) {
        perror("mmap");
        abort();
}
```

You don't have to specify address. (set NULL)
Then kernel allocate memory from free space.

alloca(3)



- allocates memory in caller's stack frame
- frees automatically when the function that called alloca() returns
  - same as local variables
  - machine and compiler dependent
  - be careful when stack size is small
    - especially multi-thread

Tips of glibc's malloc

mallopt

- int mallopt(int param, int value)
- configures glibc malloc such as
  - M_CHECK_ACTION
  - M_MMAP_THRESHOLD
  - M_TOP_PAD
  - M_TRIM_THRESHOLD
- see man 3 mallopt

malloc_stats

- void malloc_stats(void)
- prints (on standard error) statistics about heap like this

```
Arena 0:
system bytes     =     135168
in use bytes     =        128
Total (incl. mmap):
system bytes     =     139264
in use bytes     =       4224
max mmap regions =          1
max mmap bytes   =     569344
```

malloc_usable_size

- size_t malloc_usable_size(void *__ptr)
  - reports the number of usable allocated bytes associated with allocated chunk __ptr
  - This size may be a bit bigger than the size specified at malloc()
    - because of alignment of next data
- This is useful when counting allocated total size
  - increment size in hooked malloc
  - decrement size in hooked free

MALLOC_CHECK_

- easy way to enable additional checking in glibc malloc
  - with some overhead
- environment variable MALLOC_CHECK_
  - 0: no check at all (no overhead)
  - 1: check and print message if error
  - 2: check and abort if error

__malloc_hook

- glibc's malloc has its own hook mechanism
- global variables of function pointers
  - __malloc_hook
  - __realloc_hook
  - __memalign_hook
  - __free_hook
  - __malloc_initialize_hook
- man malloc_hook for detail

mtrace

- easy way to enable logging in glibc malloc
  - see man 3 mtrace
- There is tool to check log and find leaking memory
  - see man 1 mtrace
- implemented using __malloc_hook
  - This seems not thread safe

How to hook and replace malloc

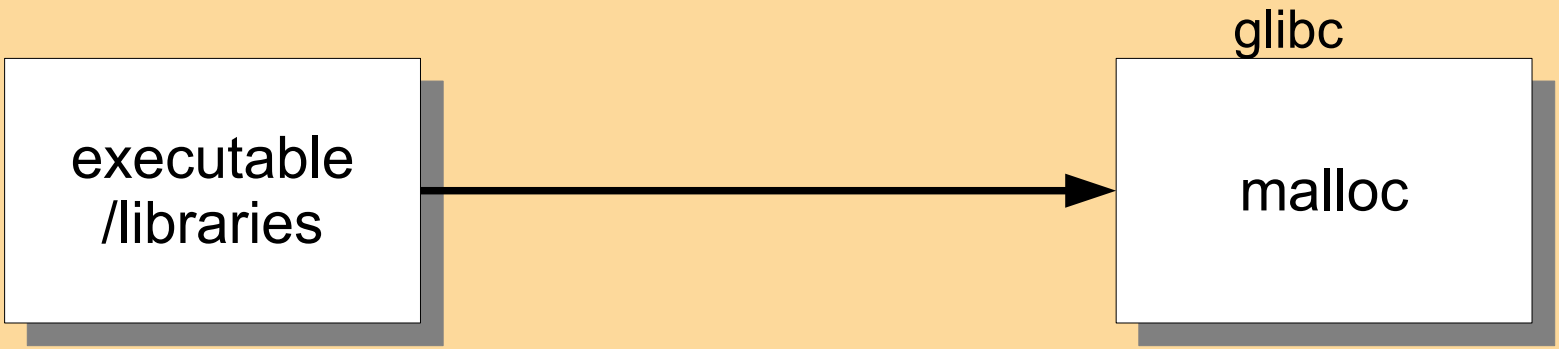
Hook and replace malloc

- 2 methods to hook malloc
  - LD_PRELOAD & dlsym
  - __malloc_hook
- These do not require to recompile other program and libraries

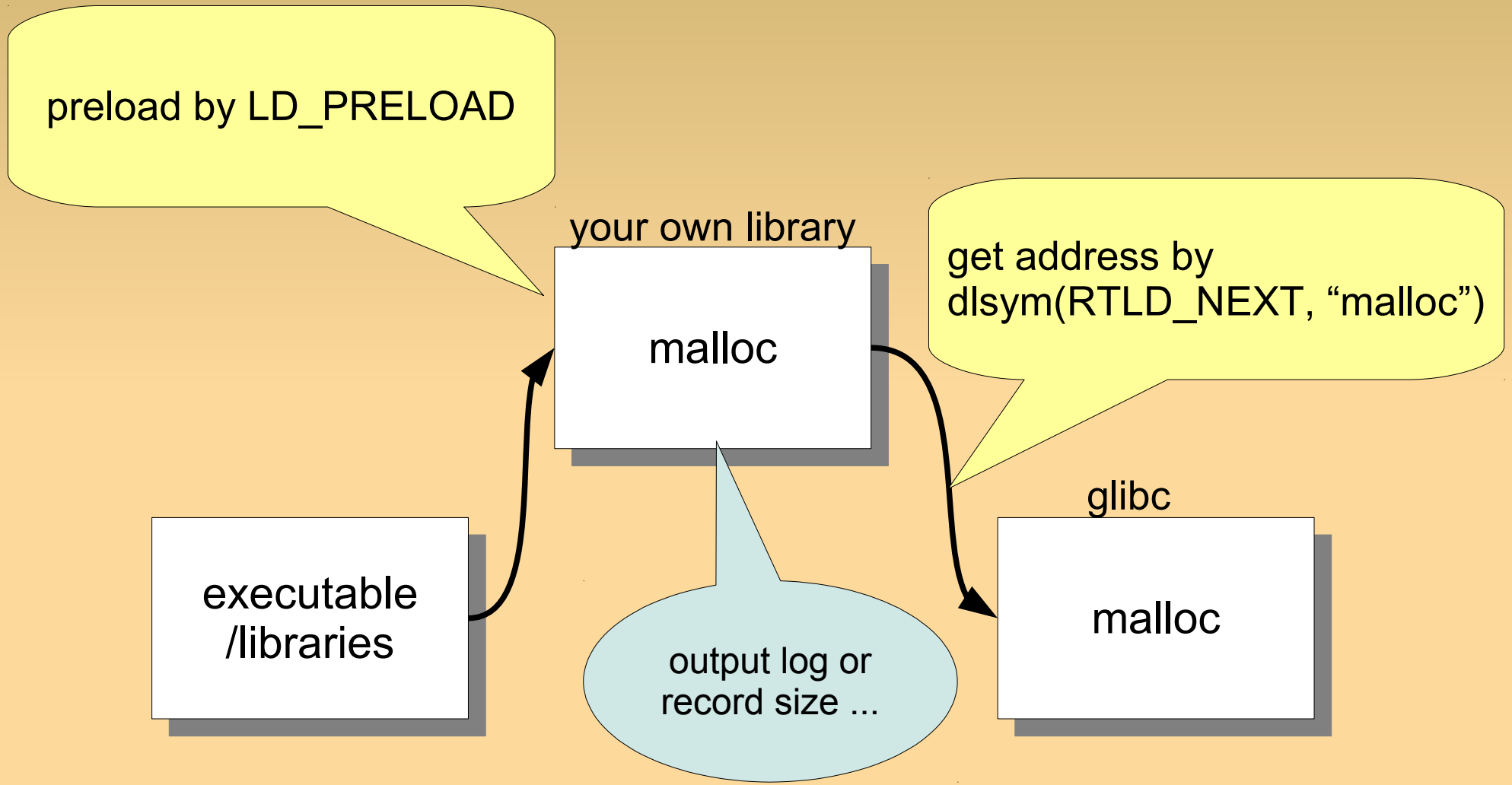
Using LD_PRELOAD & dlsym to hook malloc

- Use dynamic link mechanism
  - can not use when static linking
- Make your own malloc dynamic link library and set it to environment variable LD_PRELOAD
- Then your malloc is used prior to glibc's malloc
- You can get glibc's malloc address by dlsym(3)

Usual call for malloc



Hooking malloc by LD_PRELOAD



minimum sample code

```
static void __attribute__((constructor)) init(void)
{
        callocp = (void *(*) (size_t, size_t)) dlsym (RTLD_NEXT, "calloc");
        mallocp = (void *(*) (size_t)) dlsym (RTLD_NEXT, "malloc");
        reallocp = (void *(*) (void *, size_t)) dlsym (RTLD_NEXT, "realloc");
        memalignp = (void *(*)(size_t, size_t)) dlsym (RTLD_NEXT, "memalign");
        freep = (void (*) (void *)) dlsym (RTLD_NEXT, "free");
```

```
 void *malloc (size_t len)
{
        void *ret;

        ret = (*mallocp)(len);
        return ret;
}
```

Pitfall #1

- If you use printf to output logs, it causes recursive call of malloc. Because printf uses malloc internally.

Avoid infinite recursive call

```
static __thread int no_hook;
```

TLS (Thread Local Storage)

```
void *malloc (size_t len)
{
        void *ret;
        void *caller;

        if (no_hook) {
                return (*mallocp)(len);
        }

        no_hook = 1;
        caller = RETURN_ADDRESS(0);
        fprintf(logfp, "%p malloc(%zu", caller, len);
        ret = (*mallocp)(len);
        fprintf(logfp, ") -> %p\n", ret);
        no_hook = 0;
        return ret;
}
```

Pitfall #2

- When compile with -pthread, it crashes at the beginning. Why?
- In multi-thread mode, dlsym() uses calloc() at the first time.
  - calloc() requires dlsym()
    dlsym() requires calloc() … !!
  - prepare special calloc() for the first call of calloc().

Call special calloc at the 1st time

```
void *calloc (size_t n, size_t len)
{
        void *ret;
        void *caller;

        if (no_hook) {
                if (callocp == NULL) {
                        ret = my_calloc(n, len);
                        return ret;
                }
                return (*callocp)(n, len);
        }
        ...
```

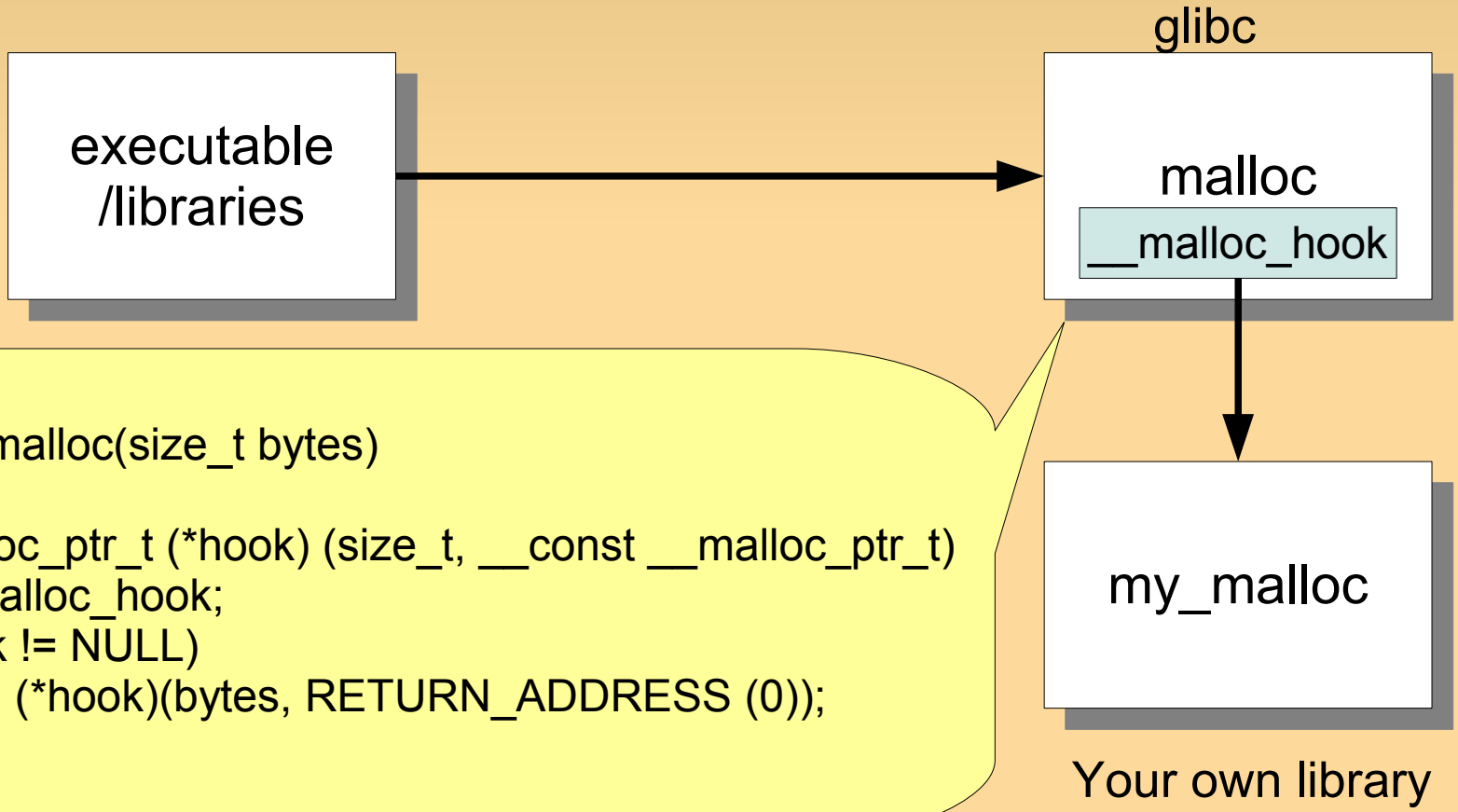
Just returns some static allocated memory

Using __malloc_hook variable to hook malloc

- function pointer variables for hooking
  - void *(*__malloc_hook)(size_t, const void*)
  - void*(*__realloc_hook)(void*, size_t, const void*)
  - void*(*__memalign_hook)(size_t, size_t, const void*)
  - void (*__free_hook)(void)
  - void (*__malloc_initialize_hook)(void)

Usual call for malloc

Hooking malloc by __malloc_hook




```
void_t *malloc(size_t bytes)
{
  __malloc_ptr_t (*hook) (size_t, __const __malloc_ptr_t)
    =__malloc_hook;
  if (hook != NULL)
    return (*hook)(bytes, RETURN_ADDRESS (0));
```

Thread unsafe example

```
static void *
my_malloc_hook(size_t size, const void *caller)
{
        void *result;

        /* Restore all old hooks */
        __malloc_hook = old_malloc_hook;

        /* Call recursively */
        result = malloc(size);

        /* Save underlying hooks */
        old_malloc_hook = __malloc_hook;

        /* printf() might call malloc(), so protect it too. */
        printf("malloc(%u) called from %p returns %p\n",
                (unsigned int) size, caller, result);

        /* Restore our own hooks */
        __malloc_hook = my_malloc_hook;
        return result;
}
```

__malloc_hook is not locked at all

In this moment malloc from other thread does not hook.

Workaround

- Changing __malloc_hook variable is not thread safe. (Actually these variables are marked as 'deprecated')
- Set once these hook variables at initial time and don't touch after that.
  - You can not call back glibc's malloc.
  - link and replace to other malloc.
    - dlmalloc is good for this.

Which ?

- If you replace malloc
  - You can use __malloc_hook with care
- Otherwise
  - use LD_PRELOAD & dlsym

Another pitfall

- Almost program works fine with my own malloc library. But some game app. causes SEGV accessing null pointer.
- At first I doubt that malloc returns NULL because heap runs out …

Behavior of malloc(size=0)

- I thought malloc(0) returns NULL.
- man malloc says:
  - *“If size is 0, then malloc() returns either NULL, or a unique pointer value that can later be successfully passed to free().”*
- glibc's malloc does the latter.

- The game app. calls malloc(0) and use the pointer without check!
  - so it causes null pointer access
- I modified my malloc returns a unique pointer even if size == 0
- Then the game app. works fine with my malloc library.

dlmalloc

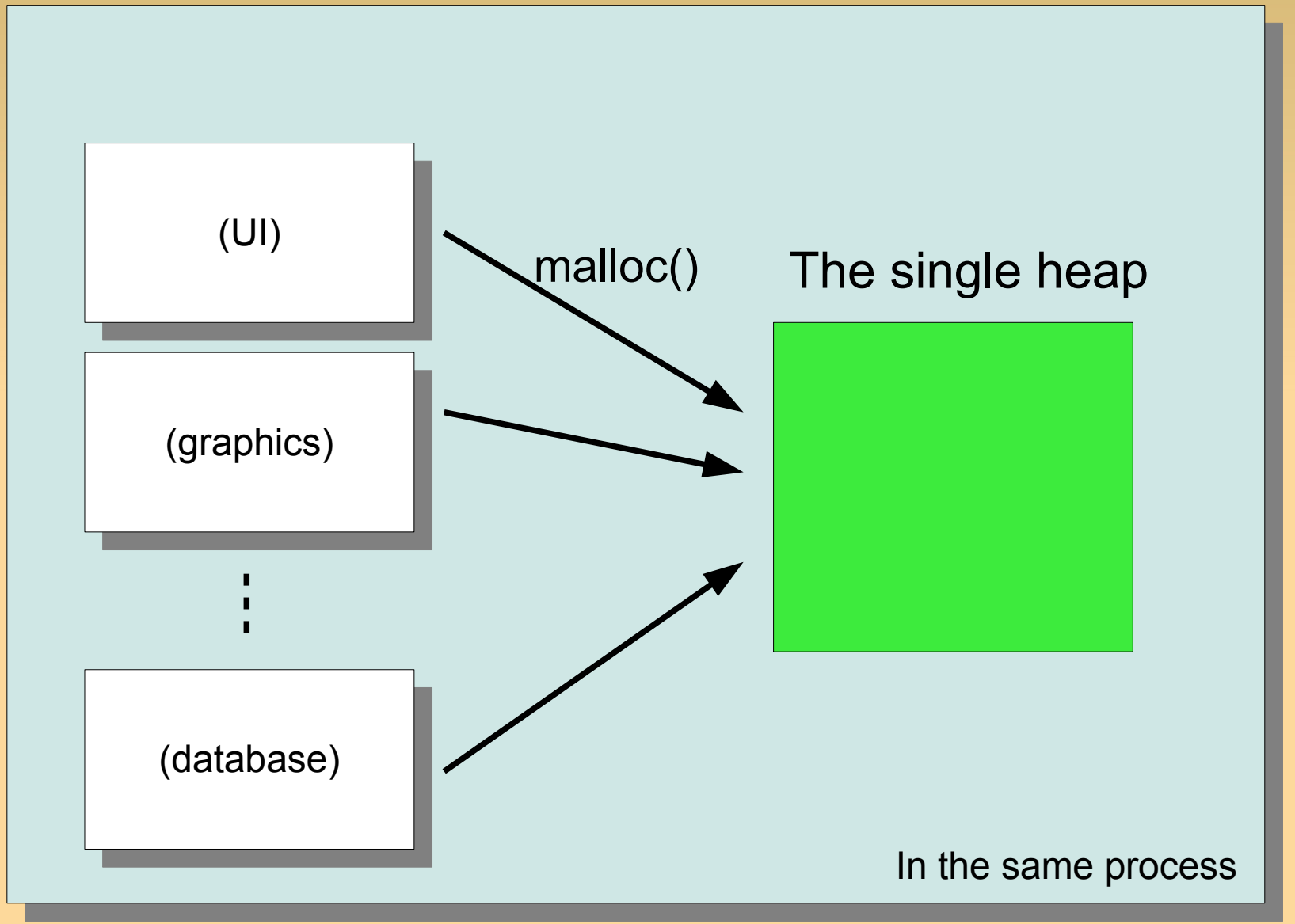
dlmalloc

- by Doug Lea
- http://g.oswego.edu/dl/html/malloc.html
- easy to compile and use
  - can add prefix to all function names to avoid conflict to standard malloc functions (-DUSE_DL_PREFIX)
  - add -DUSE_LOCKS=1 for thread safe
- Actually glibc's malloc is based on this

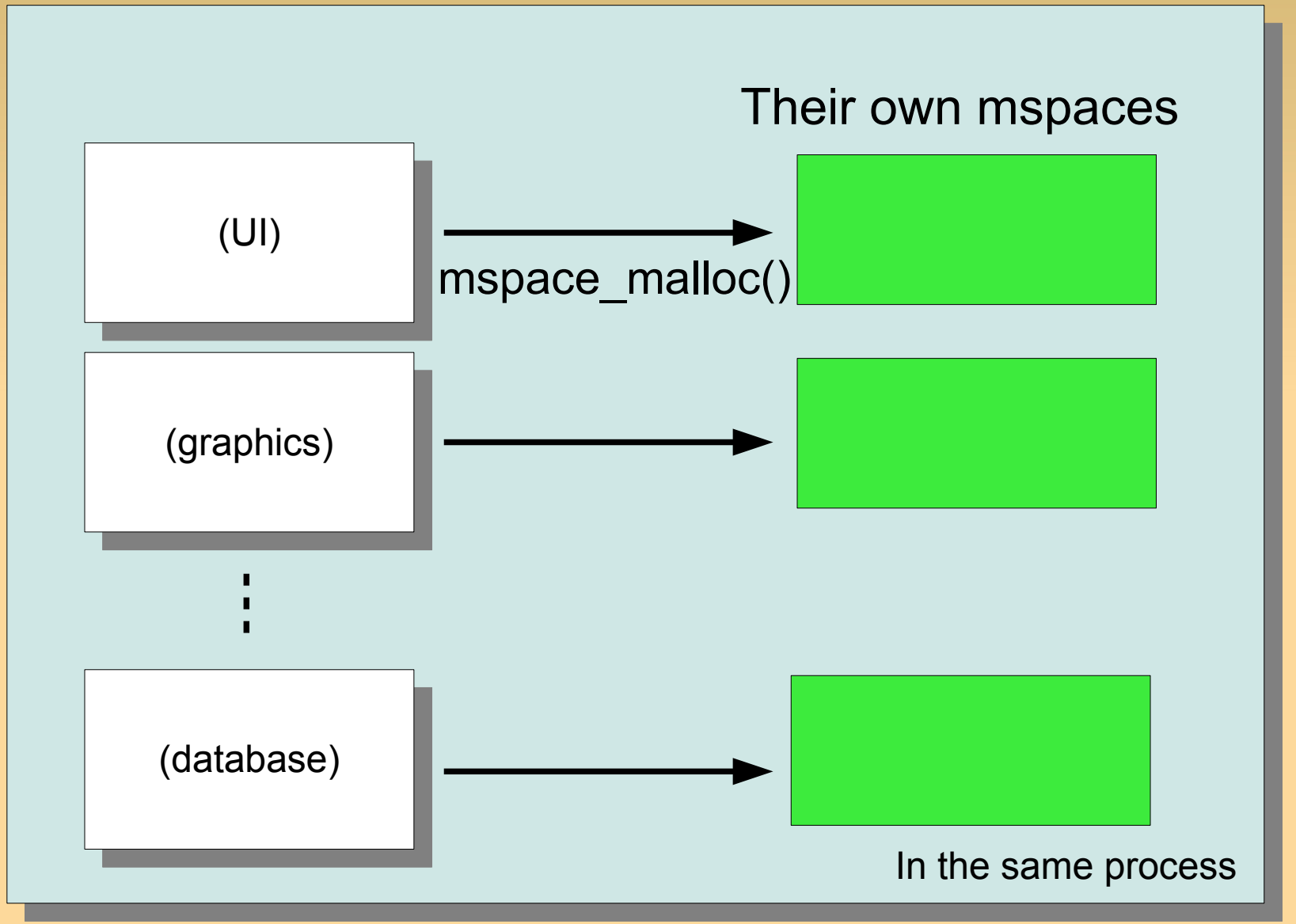
mspace of dlmalloc

- can have multiple separate memory spaces for heap
  - per thread, per functional module, ...
- Good for troubleshooting
  - isolate heap of module in question

Usual single heap



Using mspaces



Summary

- Make your own malloc library rather than modify glibc (libc.so).
- Use mmap(2) to get memory.
- __malloc_hook is not thread safe and deprecated.
- Use LD_PRELOAD & dlsym(3) to hook glibc's malloc.

Q & A

Thank you for listening!

@tetsu_koba