CTSRD

CRASH-worthy Trustworthy Systems Research and Development

Everything you ever wanted to know about "hello, world"*

(*but were afraid to ask)

Brooks Davis
SRI International

November 17th, 2017
Whitman College
Walla Walla, WA

Approved for public release; distribution is unlimited. This research is sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-10-C-0237. The views, opinions, and/or findings contained in this article/presentation are those of the author(s)/presenter(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

SRI International

University of Cambridge

Introduction

- Understanding the process ABI:
  - So I can change it!
- CHERI CPU is a MIPS64 compatible CPU
  - With C compatible memory safety extensions
  - We replace integer pointers with unforgeable capabilities
  - Prevent buffer overflows and make compartmentalization cheap

K&R: *The C Programming Language*

```
#include <stdio.h>


main()
{
  printf("hello, world\n");
}
```

K&R: *The C Programming Language*

```
#include <stdio.h>

void
main(void)
{
  printf("hello, world\n");
}
```

Today's version

```
int
main(void)
{
  const char hello[] =
    “hello, world";

  printf("%s %d\n", hello, 123);

  return (0);
}
```

Minimal C version

```
void
main(void)
{
  const char hello[] =
    "hello, world 123\n";

  write(1, hello, sizeof(hello));

  exit(0);
}
```

Minimal (MIPS) assembly version

```
        .text
        .global __start
        .ent __start
__start:
        li $a0,  1
        dla $a1, hello
        li $a2, 17
        li $v0, 4
        syscall         # write(1, "hello, world 123\n", 17)
        li $a0, 0
        li $v0, 1
        syscall         # _exit(0)
        .end __start

        .data
hello:
        .ascii "hello, world 123\n"
```


SRI International
UNIVERSITY OF CAMBRIDGE

Size comparison

- Assembly
  - Compiles to 9 instructions
  - Stripped binary less than 1K
    - Mostly ELF headers, MIPS ABI bits
- Minimal C
  - Stripped binary over 550K!
    - Mostly `malloc()` and localization

Program linkage

```
$ cc -static -o helloworld helloworld.o
```

```
$ ld -EB -melf64btsmip_fbsd -Bstatic \
  -o helloworld /usr/lib/crt1.o \
  /usr/lib/crti.o /usr/lib/crtbeginT.o \
  -L/usr/lib helloworld.o \
  --start-group -lgcc -lgcc_eh –lc
  --end-group \
  /usr/lib/crtend.o /usr/lib/crtn.o
```

Compiler runtime support

File	Purpose
`crt1.o`	Contains `__start()` function which initializes process environment and calls `main()`.
`crti.o`	Entry points for old style `_init()` and `_fini()` functions.

`crtbegin.o`
`crtbeginS.o`
`crtbeginT.o`
	Declares `.ctor` and `.dtor` constructor and destructor sections.
	Declares functions to call constructors and destructors.

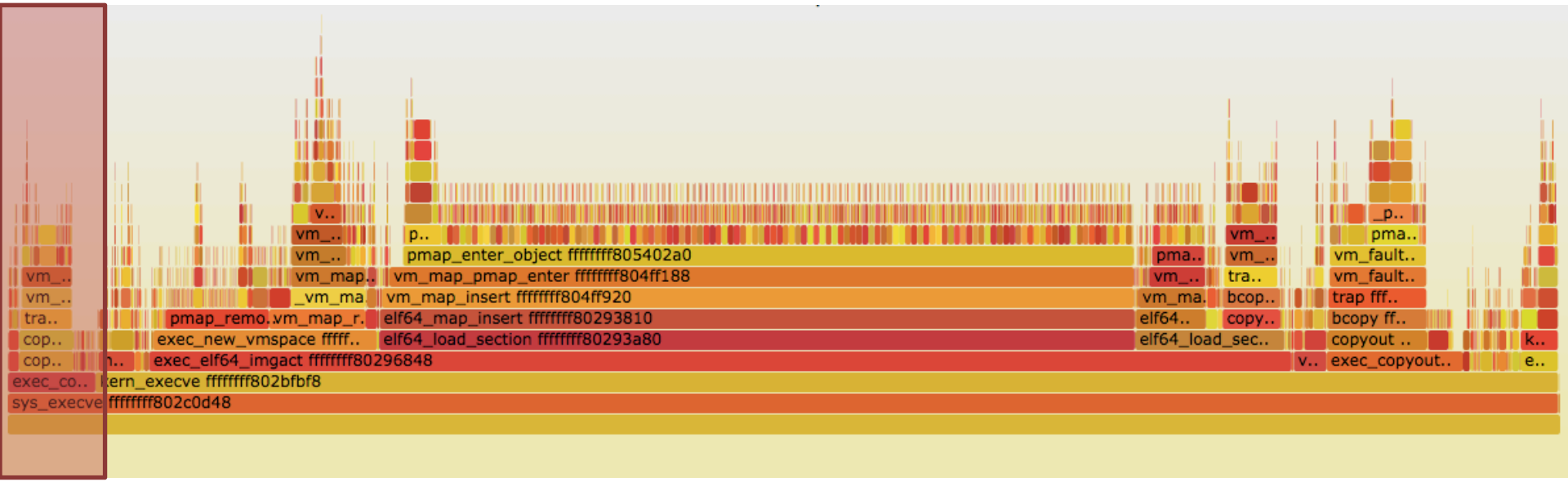
`crtend.o`	NULL terminates `.ctor` and `.dtor` sections.
`crtn.o`	Trailers for `_init()` and `_fini()` functions.

On FreeBSD: built in `gnu/lib/csu` and `lib/csu/`*ARCH*.

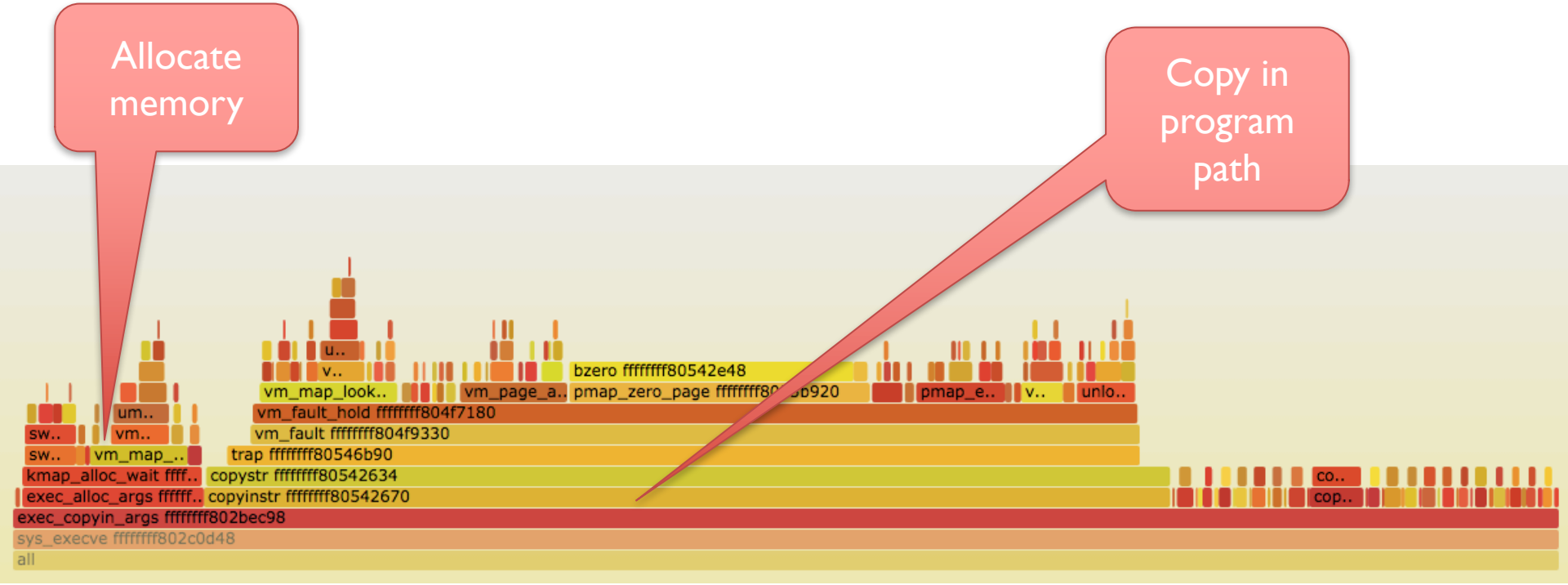
Code and images online

https://people.freebsd.org/~brooks/talks/whitman2017-helloworld

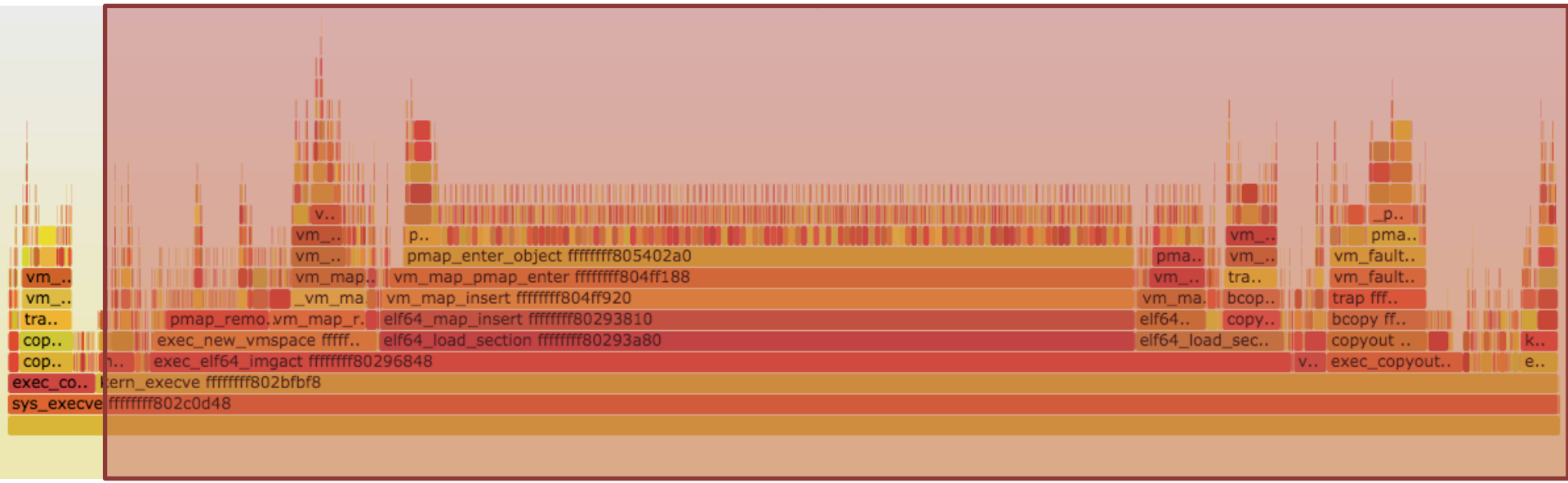
execve()


v..
vm_..
vm_..
vm_..
vm_map..
vm_..
_vm_ma
tra..
pmap_remo..
vm_map_r..
cop..
exec_new_vmspace fffff..
cop..
exec_elf64_imgact ffffffff80296848
exec_co..
kern_execve ffffffff802bfbf8
sys_execve ffffffff802c0d48
p..
pmap_enter_object ffffffff805402a0
vm_map_pmap_enter ffffffff804ff188
vm_map_insert ffffffff804ff920
elf64_map_insert ffffffff80293810
elf64_load_section ffffffff80293a80
pma..
vm_..
vm_..
vm_ma..
elf64..
elf64_load_sec..
vm_..
tra..
bcop..
copy..
v..
_p..
pma..
vm_fault..
vm_fault..
trap fff..
bcopy ff..
copyout ..
exec_copyout..
k..
e..

exec_copyin_args()

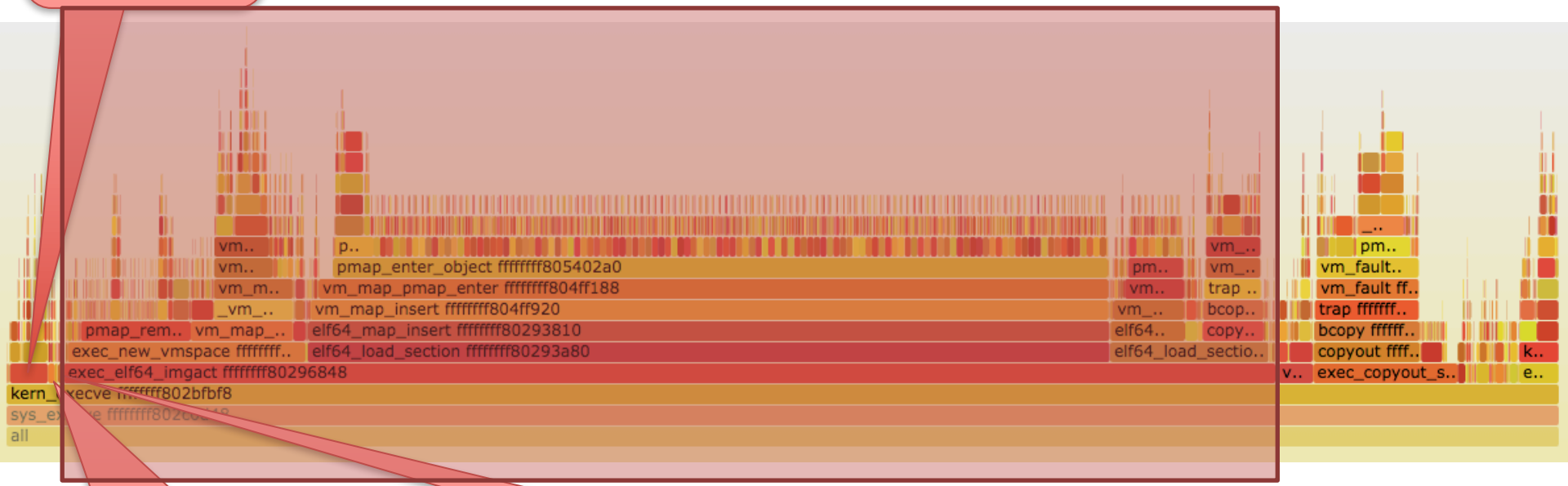


sys_execve()


v..
vm_..
p..
vm_..
_p..
vm_..
vm_..
pmap_enter_object ffffffff805402a0
pma..
vm_..
pma..
vm_..
vm_map..
vm_map_pmap_enter ffffffff804ff188
vm_..
tra..
vm_fault..
vm_..
_vm_ma
vm_map_insert ffffffff804ff920
vm_ma.
bcop..
vm_fault..
tra..
pmap_remo.
vm_map_r.
elf64_map_insert ffffffff80293810
elf64..
copy..
trap fff..
bcopy ff..
cop..
exec_new_vmspace fffff..
elf64_load_section ffffffff80293a80
elf64_load_sec..
copyout ..
k..
cop..
exec_elf64_imgact ffffffff80296848
v..
exec_copyout..
e..
exec_co..
kern_execve ffffffff802bfbf8
sys_execve
ffffffff802c0d48

kern_execve()

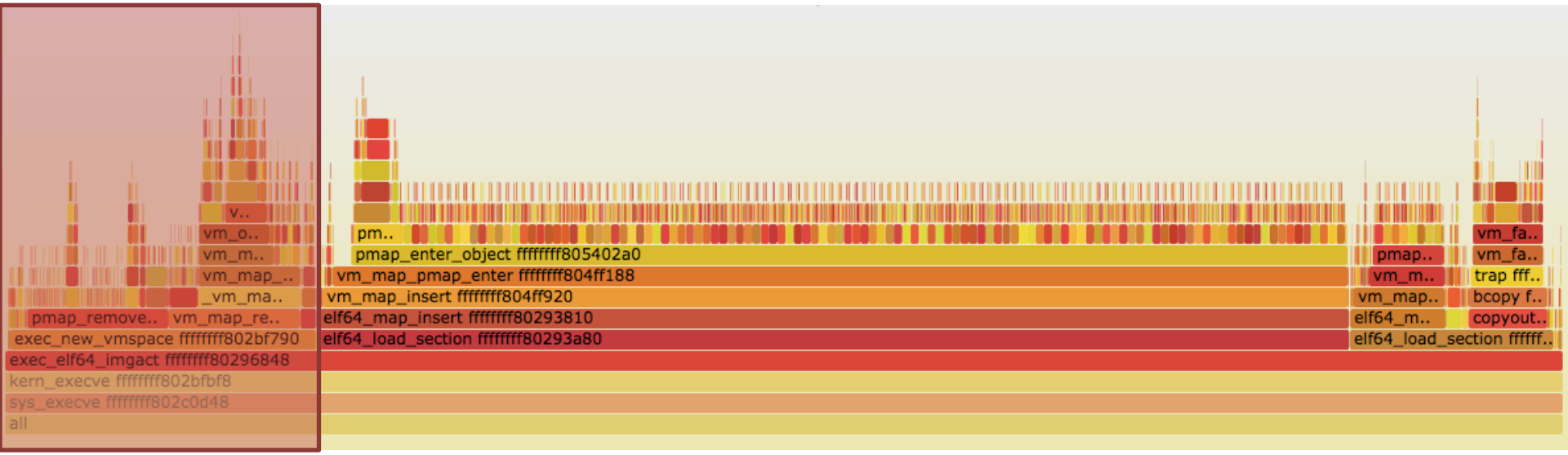
`namei()`
Resolve path



`exec_check_permissions()`
Check that the file has the right permissions and open it.

`exec_map_first_page()`
Map the header into kernel memory.

exec_elf64_imgact()



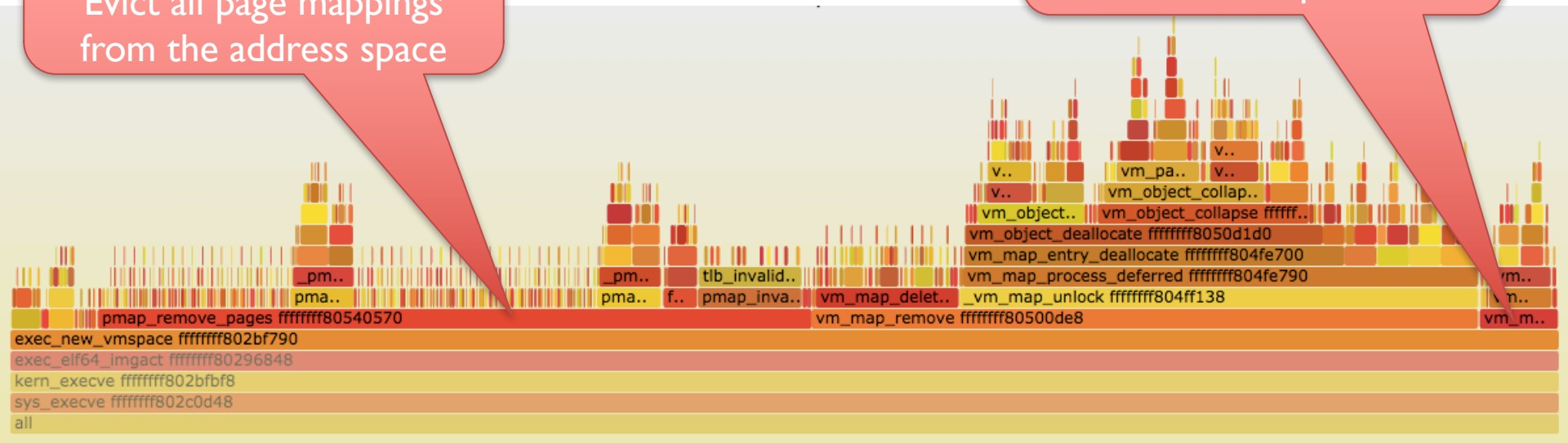
Defined with macros:

```
__CONCAT(exec_, __elfN(imgact))
    (struct image_params *imgp)
```

exec_new_vmspace()

pmap_remove_pages()
vm_map_remove()
Evict all page mappings from the address space

vm_map_stack()
Map a stack into the address space

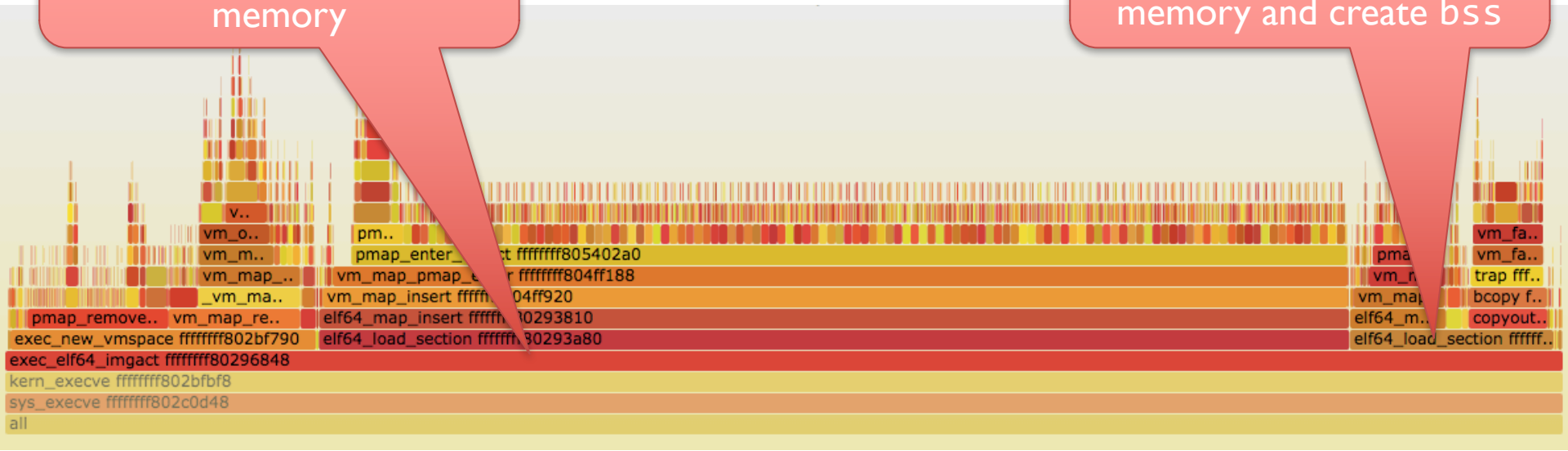


Stack

exec_elf64_imgact()

elf_load_section()
Map .text section into memory

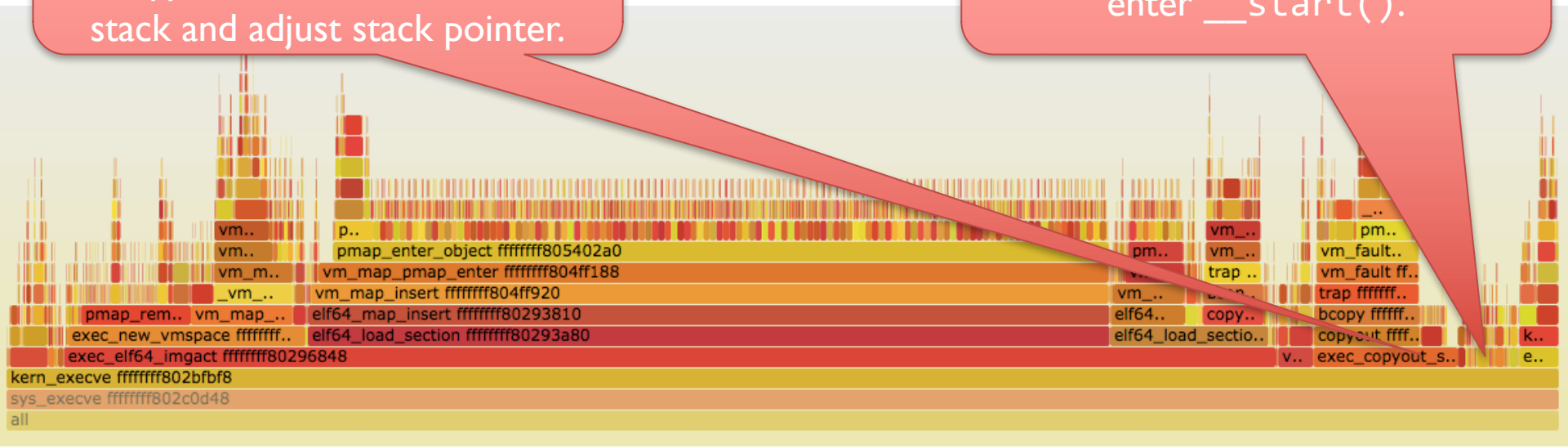
elf_load_section()
Map .data section into memory and create bss



kern_execve()

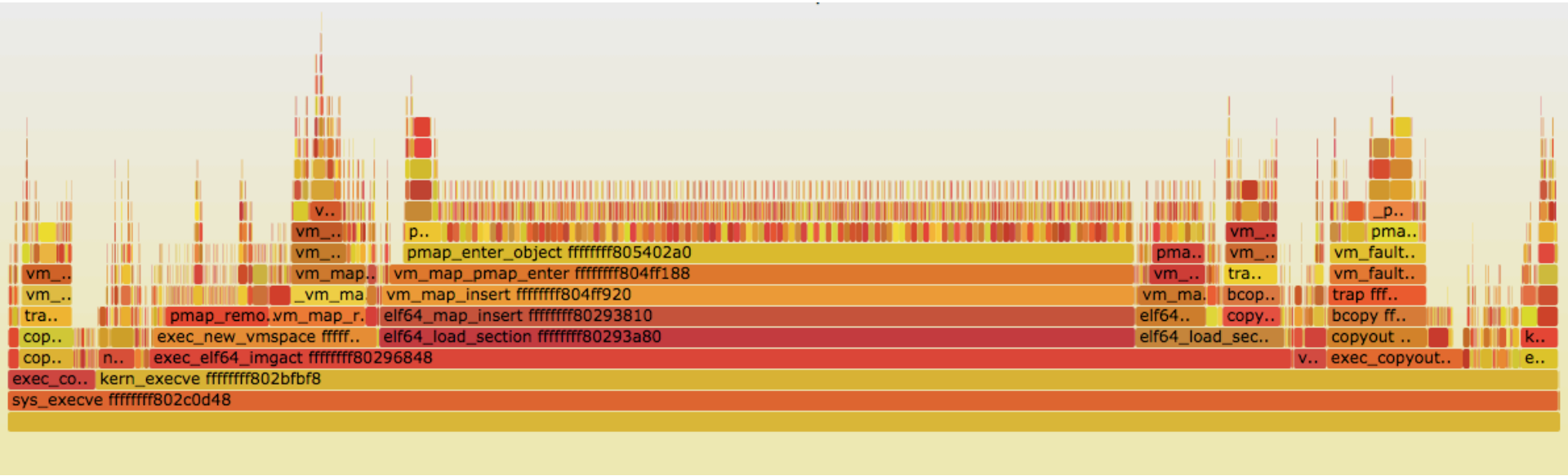
`exec_copyout_strings()`
`elf64_freebsd_fixup()`
Copy argv, envp, etc to the stack and adjust stack pointer.

`exec_setregs()`
Set initial register context to enter `__start()`.



SRI International
UNIVERSITY OF CAMBRIDGE

sys_execve()

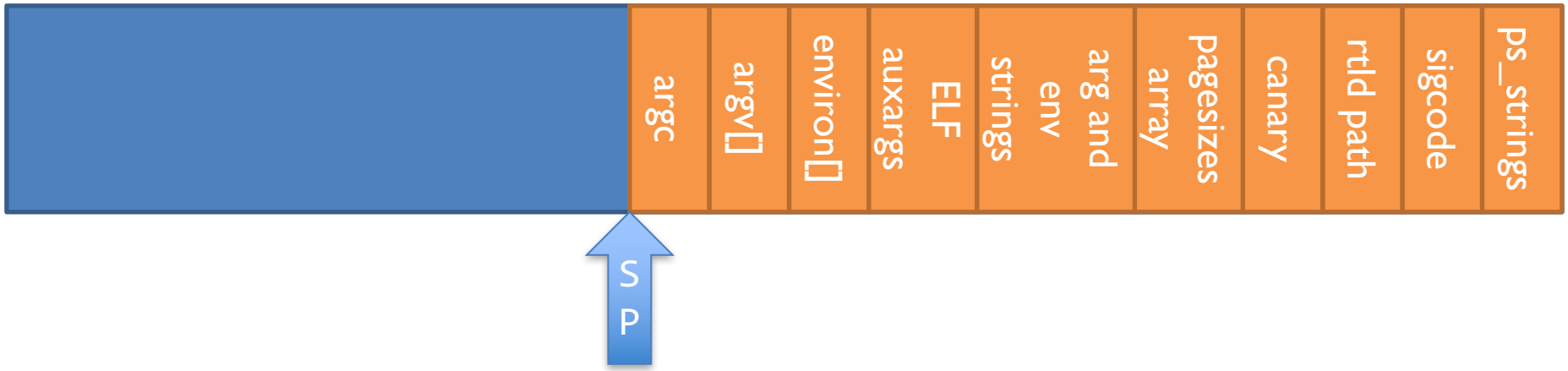


SRI International
UNIVERSITY OF CAMBRIDGE

Returning to userspace

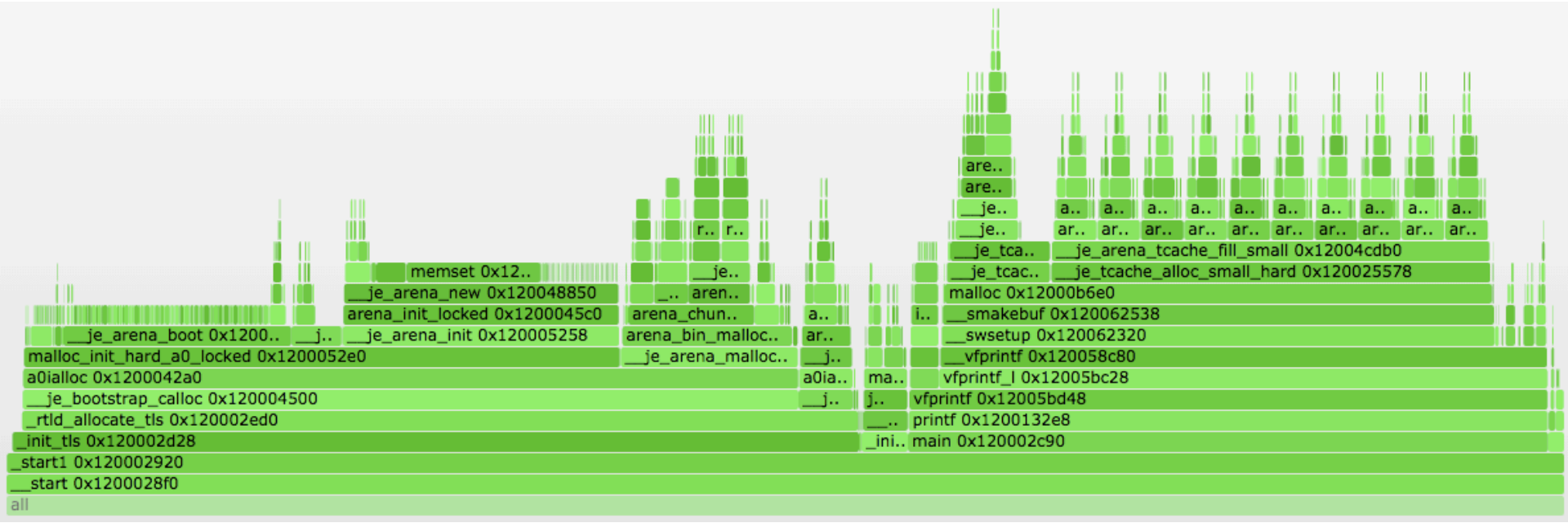
- Stack is mapped into address space
- Program is mapped into address space
- Strings, `argv`, `envp`, signal handler, etc are on the top of the stack
- Register state is set up to call `__start()`

SCO i386 ABI stack




```
__start(char **ap, …) {
  ...
        argc = * (long *) ap;
        argv = ap + 1;
        env  = ap + 2 + argc;
  ...
```

__start()



Most cycles spent in `malloc()`

SRI International
UNIVERSITY OF CAMBRIDGE

__start() 1/2

```
void __start(char **ap)
{
  int argc;
  char **argv, **env;

  argc = * (long *) ap;
  argv = ap + 1;
  env  = ap + 2 + argc;
…
```

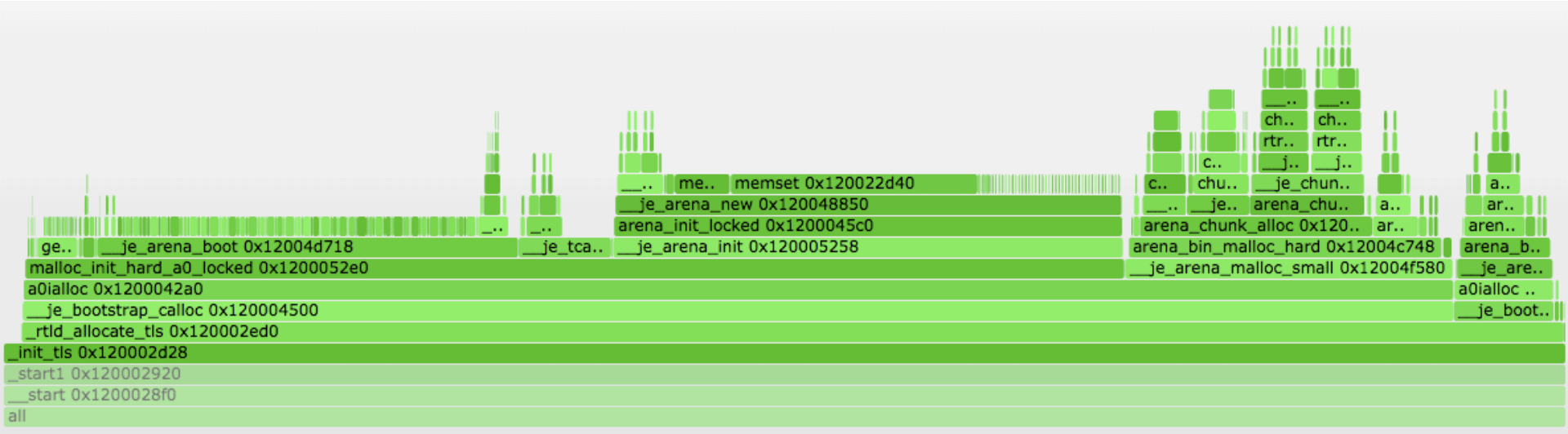
__start() 2/2

```
…
    handle_argv(argc, argv, env);
    _init_tls();
    handle_static_init(argc, argv,
      env);

    exit(main(argc, argv, env));
}
```

Set environ and __progname variables.

_init_tls()



Most cycles spent in `malloc()`

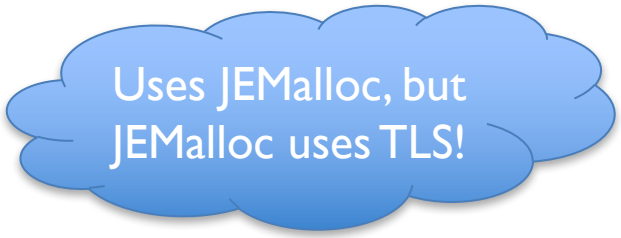
_init_tls()

- Find the ELF auxargs vector

```
Elf_Addr *sp;
sp = (Elf_Addr *) environ;
while (*sp++ != 0)
   ;
aux = (Elf_Auxinfo *) sp;
```

`_init_tls()`

- Find the ELF auxargs vector
- Use that to find the program headers
- Use those to find the PT_TLS section (initial values)
- Call `__libc_allocate_tls()` (as `_rtld_allocate_tls()`)
  - Allocates space
  - Copies initial values
- Set the TLS pointer



__start() 2/2

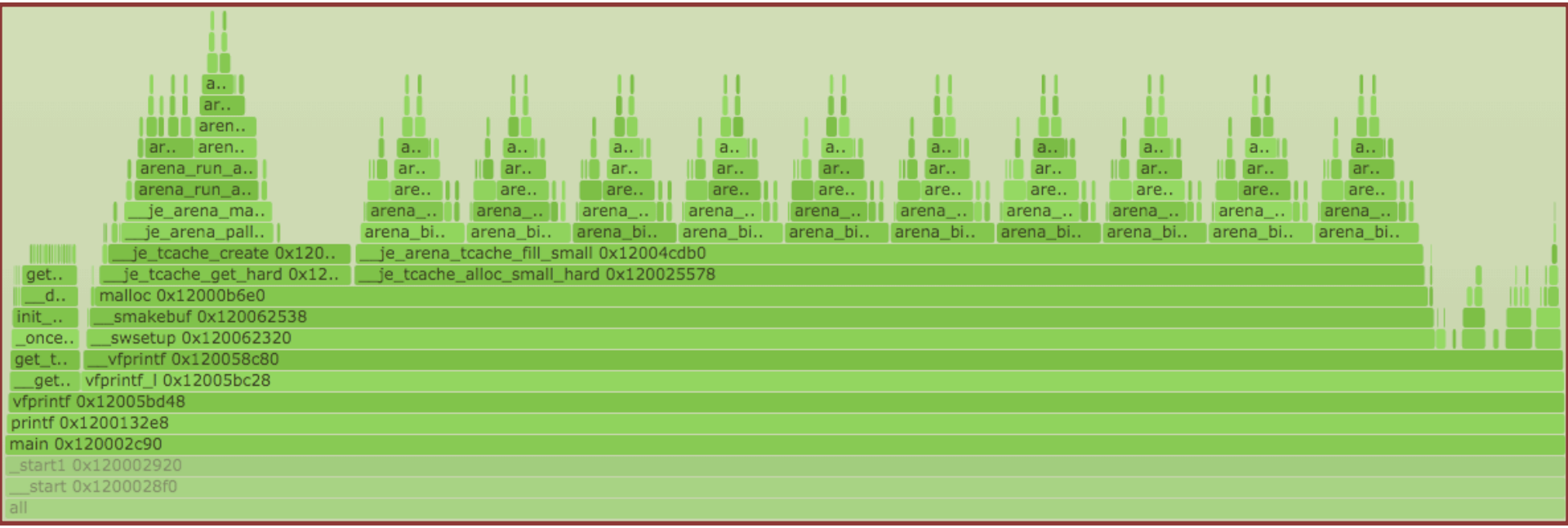
```
…
    handle_argv(argc, argv, env);
    _init_tls();
    handle_static_init(argc, argv,
      env);

    exit(main
}
```

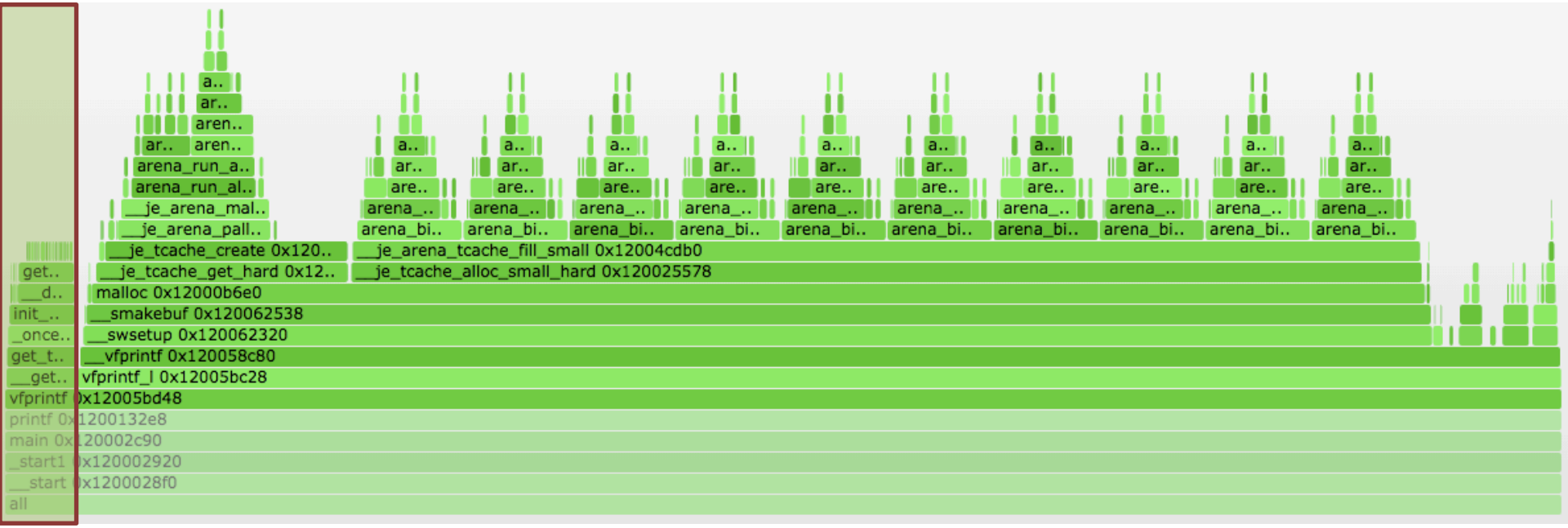
Calls constructors and registers destructors. Four types supported:

- `.pre_init_array` section
- `_init()` function
- `.ctors` section (via `_init()`)
- `.init_array` section

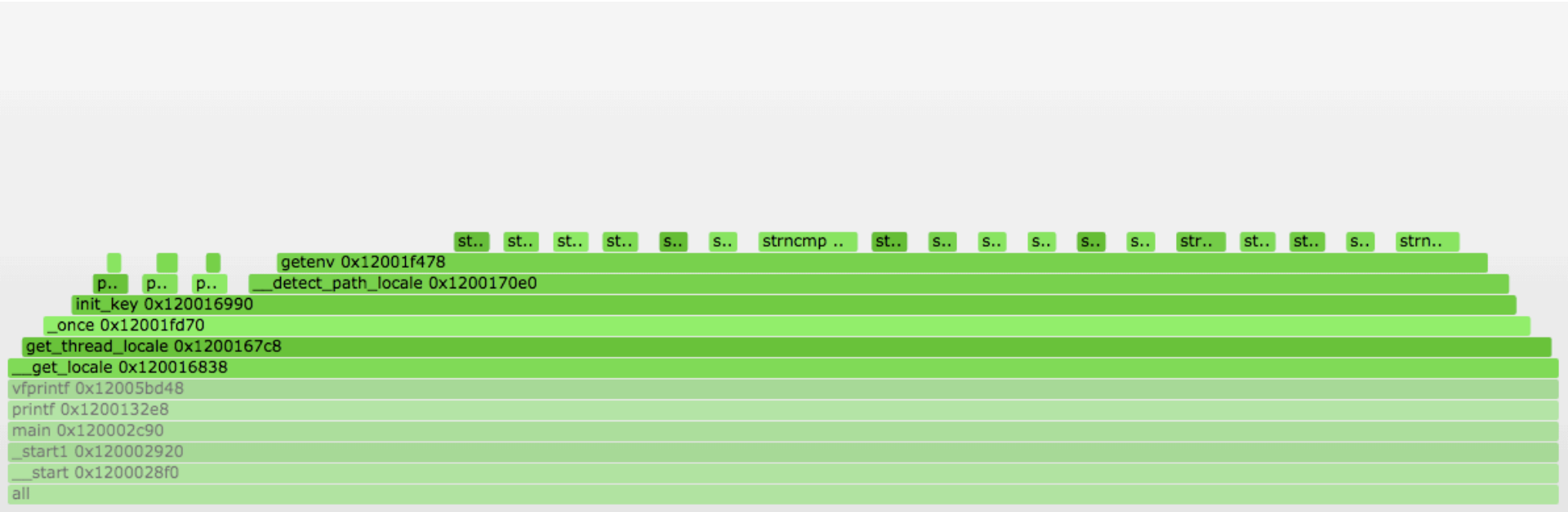
main()


a..
ar..
aren..
ar.. aren..
arena_run_a..
arena_run_a..
__je_arena_ma..
__je_arena_pall..
__je_tcache_create 0x120..
__je_arena_tcache_fill_small 0x12004cdb0
get..
__je_tcache_get_hard 0x12..
__je_tcache_alloc_small_hard 0x120025578
__d..
malloc 0x12000b6e0
init_..
__smakebuf 0x120062538
_once..
__swsetup 0x120062320
get_t..
__vfprintf 0x120058c80
__get..
vfprintf_l 0x12005bc28
vfprintf 0x12005bd48
printf 0x1200132e8
main 0x120002c90
_start1 0x120002920
__start 0x1200028f0
all

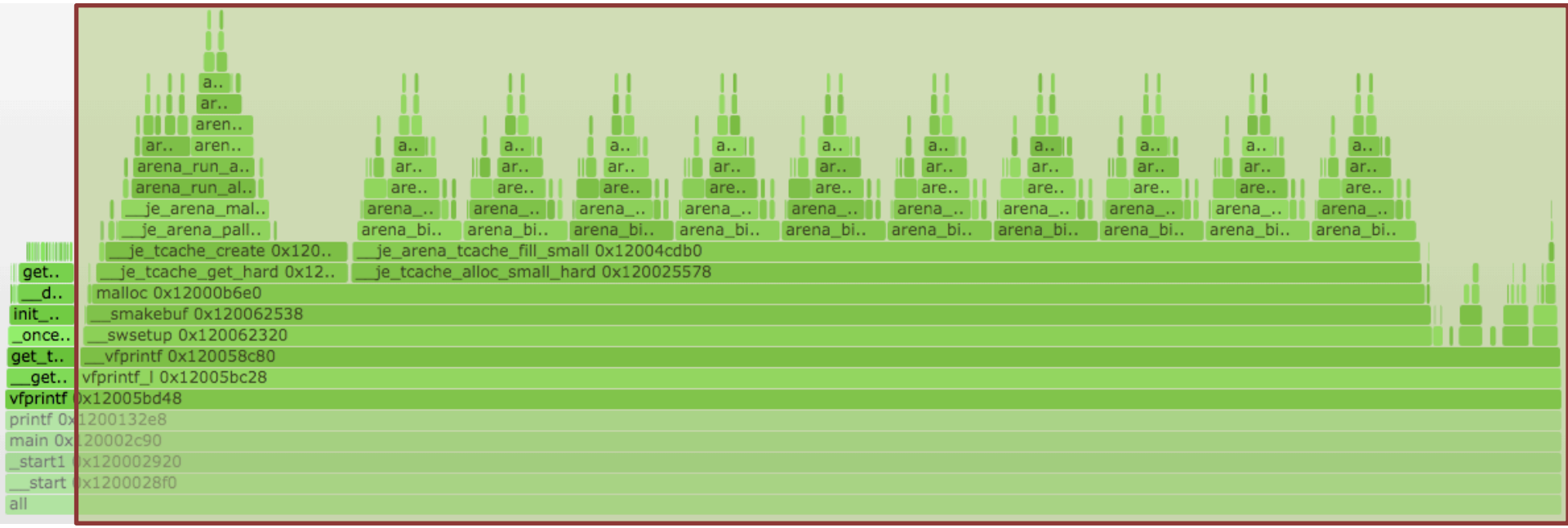
vfprintf()


a..
ar..
aren..
ar..
aren..
arena_run_a..
arena_run_al..
__je_arena_mal..
__je_arena_pall..
__je_tcache_create 0x120..
__je_arena_tcache_fill_small 0x12004cdb0
__je_tcache_get_hard 0x12..
__je_tcache_alloc_small_hard 0x120025578
malloc 0x12000b6e0
__smakebuf 0x120062538
__swsetup 0x120062320
__vfprintf 0x120058c80
vfprintf_l 0x12005bc28
0x12005bd48
printf 0x1200132e8
main 0x120002c90
_start1 0x120002920
__start 0x1200028f0
all
get..
__d..
init_..
_once..
get_t..
__get..
vfprintf
are..
arena_..
arena_bi..

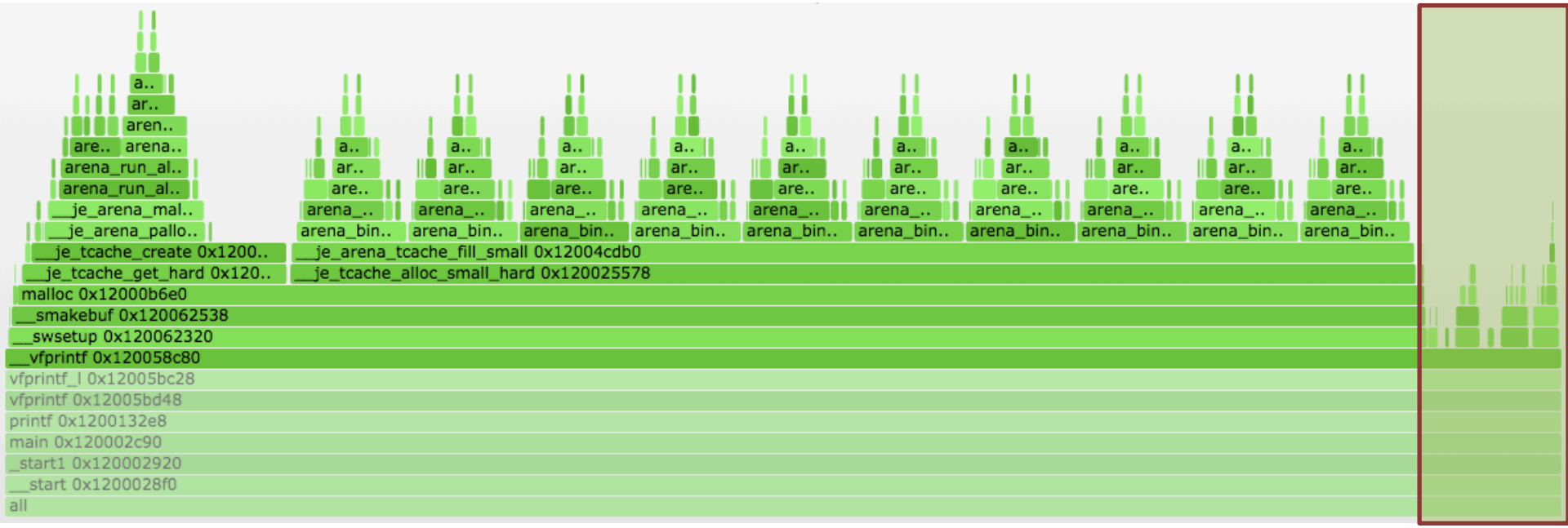
__get_locale()


st..
st..
st..
st..
s..
s..
strncmp ..
st..
s..
s..
s..
s..
s..
str..
st..
st..
s..
strn..
getenv 0x12001f478
p..
p..
p..
__detect_path_locale 0x1200170e0
init_key 0x120016990
_once 0x12001fd70
get_thread_locale 0x1200167c8
__get_locale 0x120016838
vfprintf 0x12005bd48
printf 0x1200132e8
main 0x120002c90
_start1 0x120002920
__start 0x1200028f0
all

vfprintf()

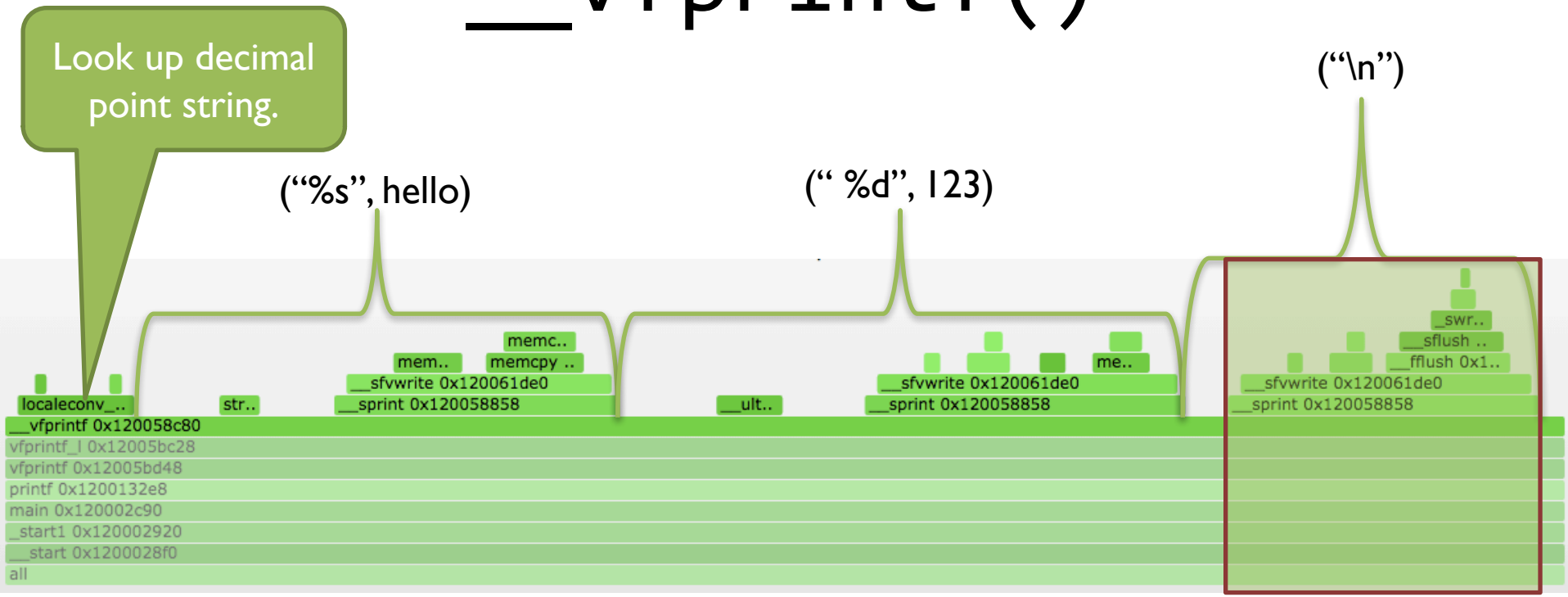

__je_tcache_create 0x120..
__je_arena_tcache_fill_small 0x12004cdb0
__je_tcache_get_hard 0x12..
__je_tcache_alloc_small_hard 0x120025578
malloc 0x12000b6e0
__smakebuf 0x120062538
__swsetup 0x120062320
__vfprintf 0x120058c80
vfprintf_l 0x12005bc28
vfprintf 0x12005bd48
printf 0x1200132e8
main 0x120002c90
_start1 0x120002920
__start 0x1200028f0
all

__vfprintf()

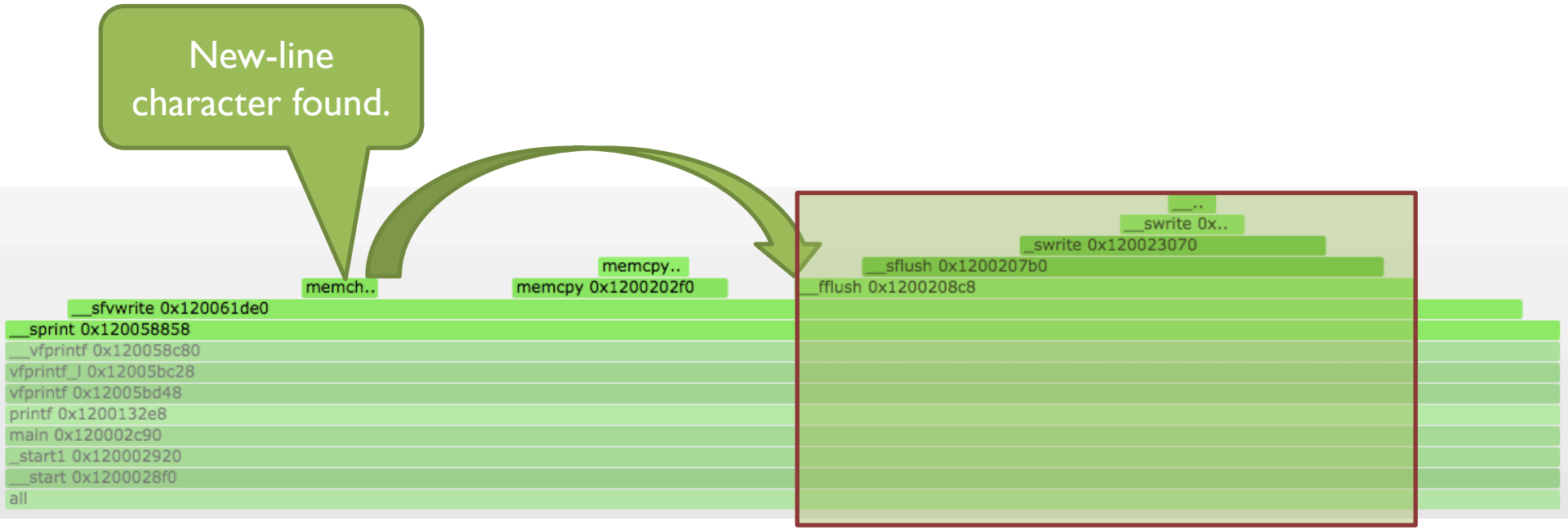


/* This code is large and complicated... */

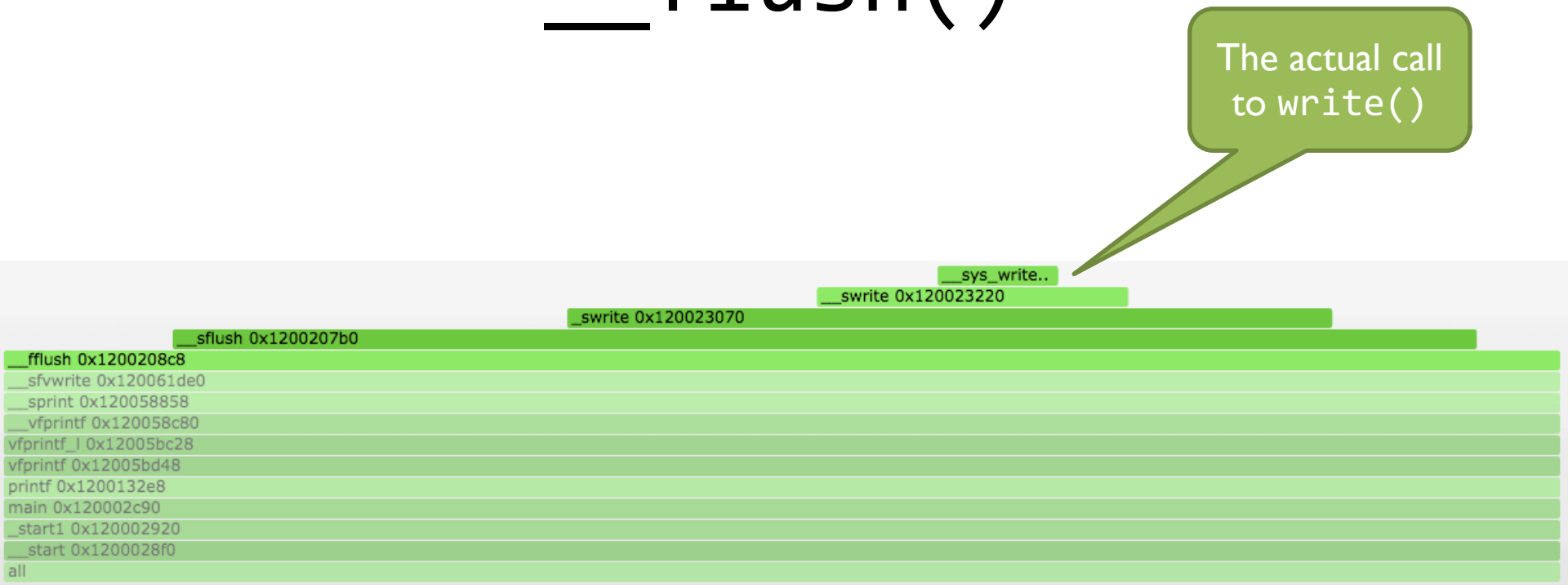
__vfprintf()


Look up decimal point string.
("%s", hello)
(" %d", 123)
("\n")
memc..
mem..
memcpy ..
__sfvwrite 0x120061de0
__sprint 0x120058858
localeconv_..
str..
__ult..
me..
__sfvwrite 0x120061de0
__sprint 0x120058858
_swr..
__sflush ..
__fflush 0x1..
__sfvwrite 0x120061de0
__sprint 0x120058858
__vfprintf 0x120058c80
vfprintf_l 0x12005bc28
vfprintf 0x12005bd48
printf 0x1200132e8
main 0x120002c90
_start1 0x120002920
__start 0x1200028f0
all

__sprint()

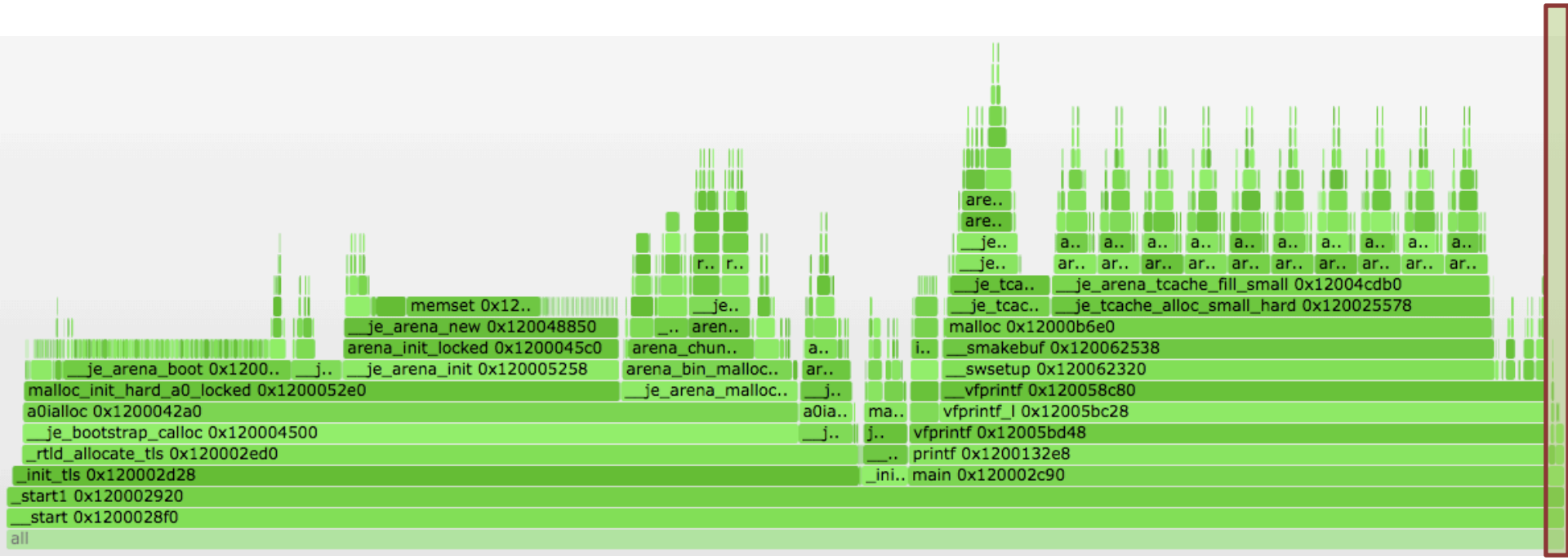

New-line character found.
memch..
memcpy..
memcpy 0x1200202f0
__..
__swrite 0x..
_swrite 0x120023070
__sflush 0x1200207b0
__fflush 0x1200208c8
__sfvwrite 0x120061de0
__sprint 0x120058858
__vfprintf 0x120058c80
vfprintf_l 0x12005bc28
vfprintf 0x12005bd48
printf 0x1200132e8
main 0x120002c90
_start1 0x120002920
__start 0x1200028f0
all

__flush()


The actual call to write()
__sys_write..
__swrite 0x120023220
_swrite 0x120023070
__sflush 0x1200207b0
__fflush 0x1200208c8
__sfvwrite 0x120061de0
__sprint 0x120058858
__vfprintf 0x120058c80
vfprintf_l 0x12005bc28
vfprintf 0x12005bd48
printf 0x1200132e8
main 0x120002c90
_start1 0x120002920
__start 0x1200028f0
all

```
hello, world 123
```

__start()

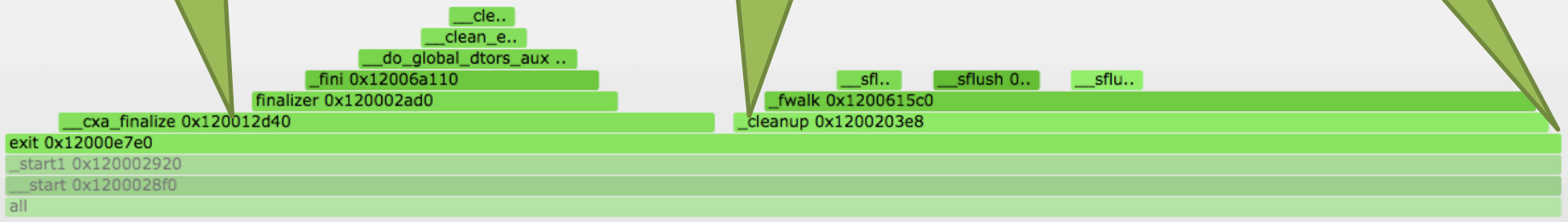

are..
are..
__je..
__je..
__je_tca..
__je_tcac..
a..
ar..
__je_arena_tcache_fill_small 0x12004cdb0
__je_tcache_alloc_small_hard 0x120025578
malloc 0x12000b6e0
__smakebuf 0x120062538
__swsetup 0x120062320
__vfprintf 0x120058c80
vfprintf_l 0x12005bc28
vfprintf 0x12005bd48
printf 0x1200132e8
main 0x120002c90
memset 0x12..
__je_arena_new 0x120048850
arena_init_locked 0x1200045c0
__je_arena_boot 0x1200..
__je_arena_init 0x120005258
malloc_init_hard_a0_locked 0x1200052e0
a0ialloc 0x1200042a0
__je_bootstrap_calloc 0x120004500
_rtld_allocate_tls 0x120002ed0
_init_tls 0x120002d28
_start1 0x120002920
__start 0x1200028f0
all

exit()

Call destructors registered with `atexit()`

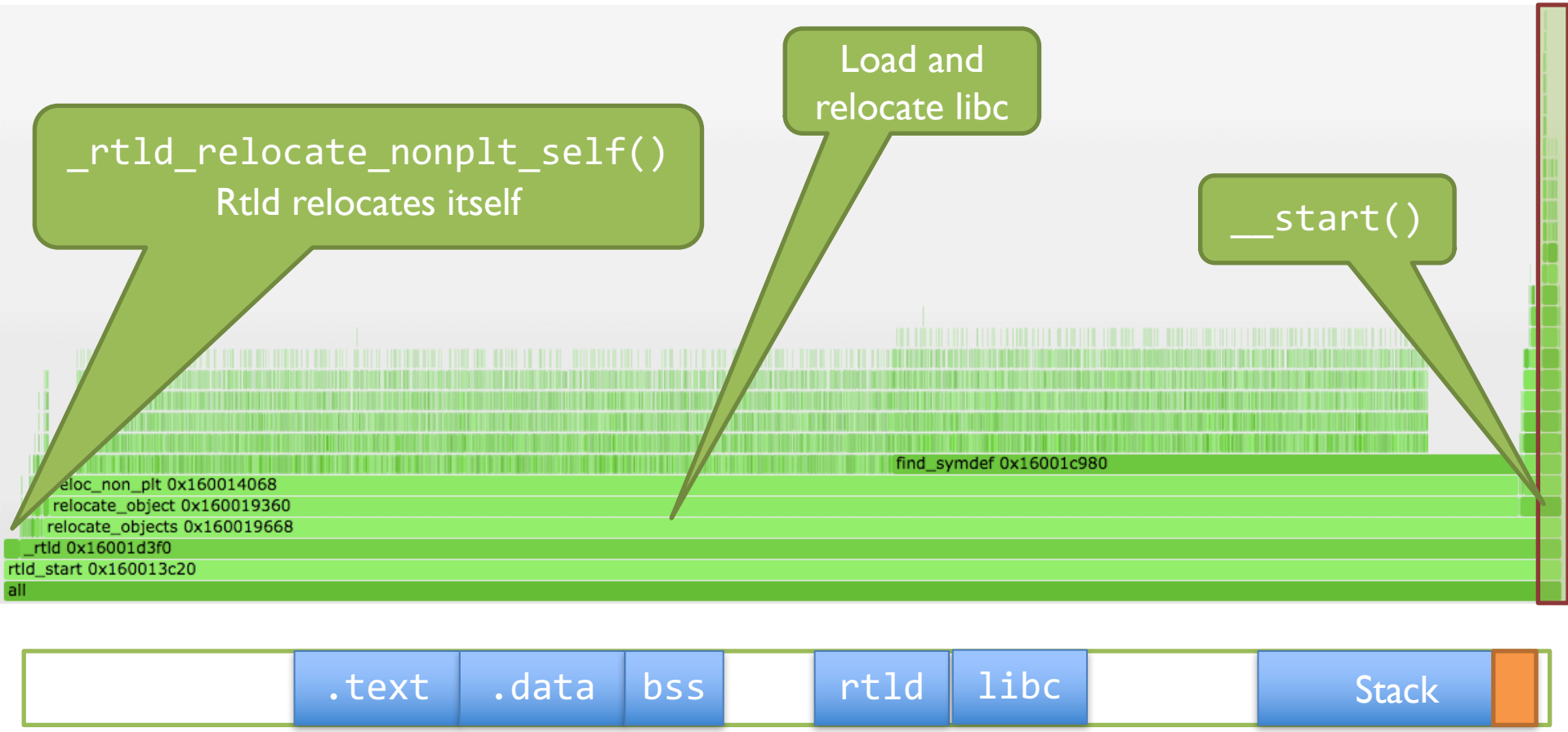
Flush any unflushed FILEs

Call `_exit()`

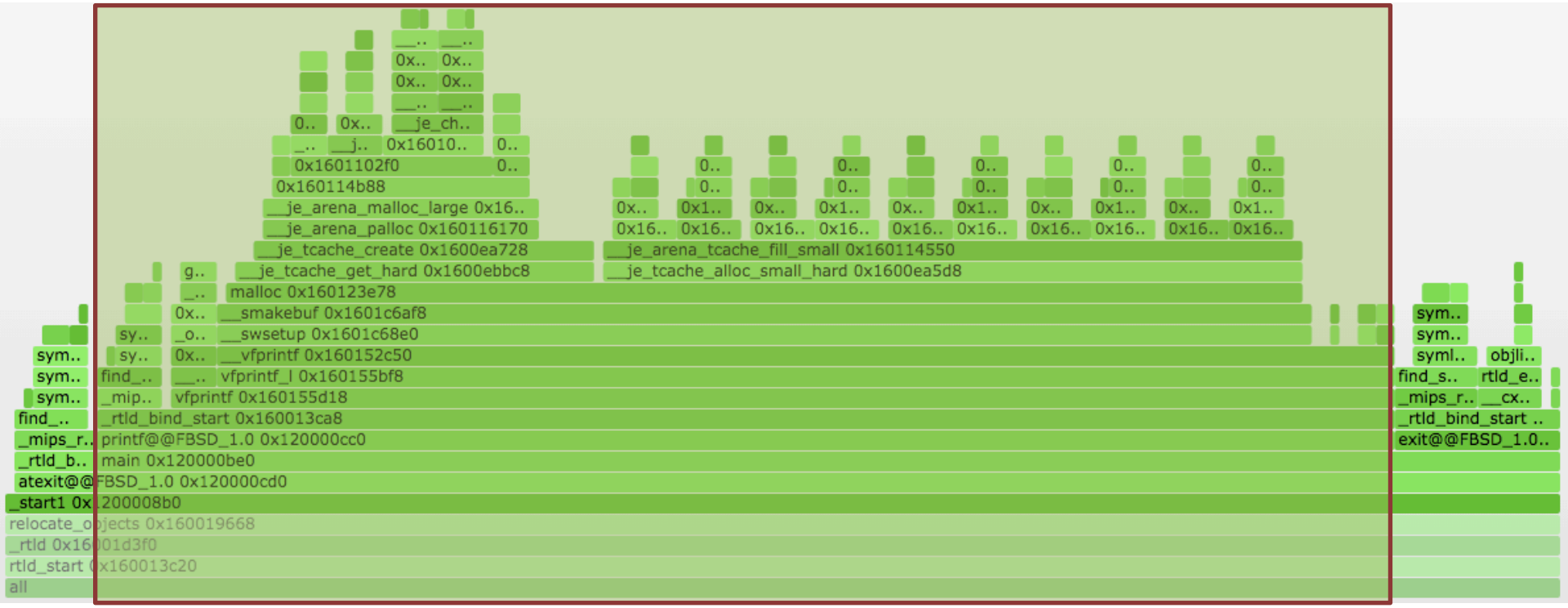


SRI International
UNIVERSITY OF CAMBRIDGE

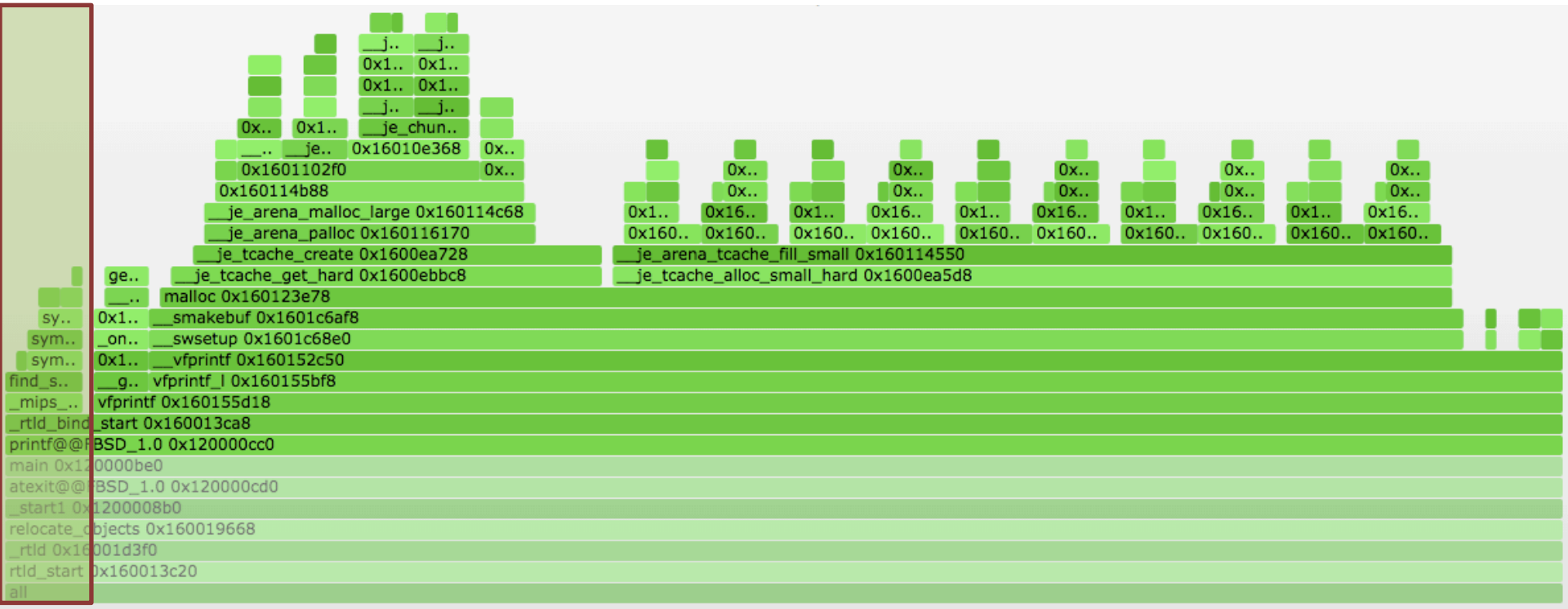
Dynamic binary



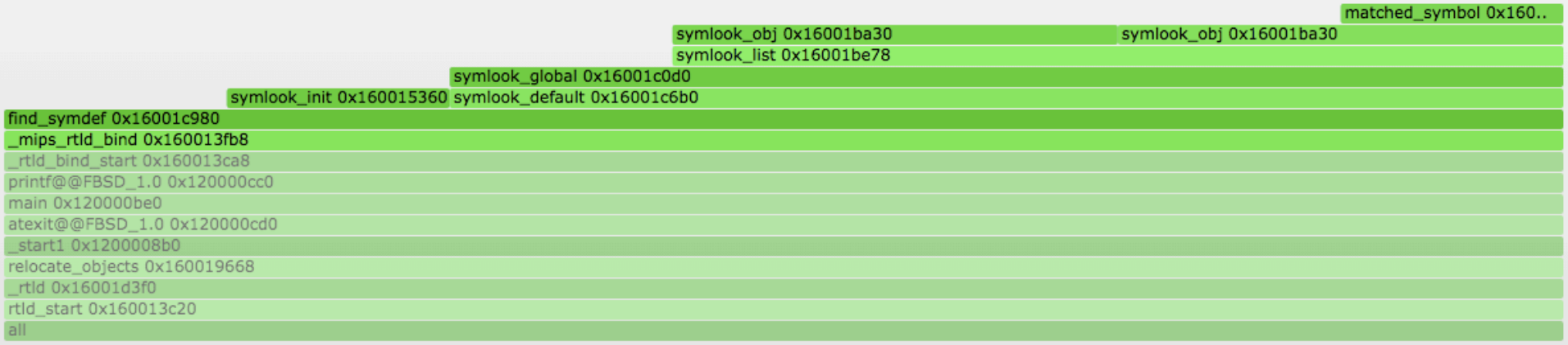
__start()


__.. __..
0x.. 0x..
0x.. 0x..
__.. __..
0.. 0x.. __je_ch..
_.. __j.. 0x16010.. 0..
0x1601102f0 0..
0x160114b88
__je_arena_malloc_large 0x16..
__je_arena_palloc 0x160116170
__je_tcache_create 0x1600ea728
__je_arena_tcache_fill_small 0x160114550
g.. __je_tcache_get_hard 0x1600ebbc8
__je_tcache_alloc_small_hard 0x1600ea5d8
_.. malloc 0x160123e78
0x.. __smakebuf 0x1601c6af8
sy.. _o.. __swsetup 0x1601c68e0
sym.. sy.. 0x.. __vfprintf 0x160152c50
sym.. find_.. __.. vfprintf_l 0x160155bf8
sym.. _mip.. vfprintf 0x160155d18
find_.. _rtld_bind_start 0x160013ca8
_mips_r.. printf@@FBSD_1.0 0x120000cc0
_rtld_b.. main 0x120000be0
atexit@@FBSD_1.0 0x120000cd0
_start1 0x120008b0
relocate_objects 0x160019668
_rtld 0x160001d3f0
rtld_start 0x160013c20
all
sym..
sym..
syml.. objli..
find_s.. rtld_e..
_mips_r.. __cx..
_rtld_bind_start ..
exit@@FBSD_1.0..

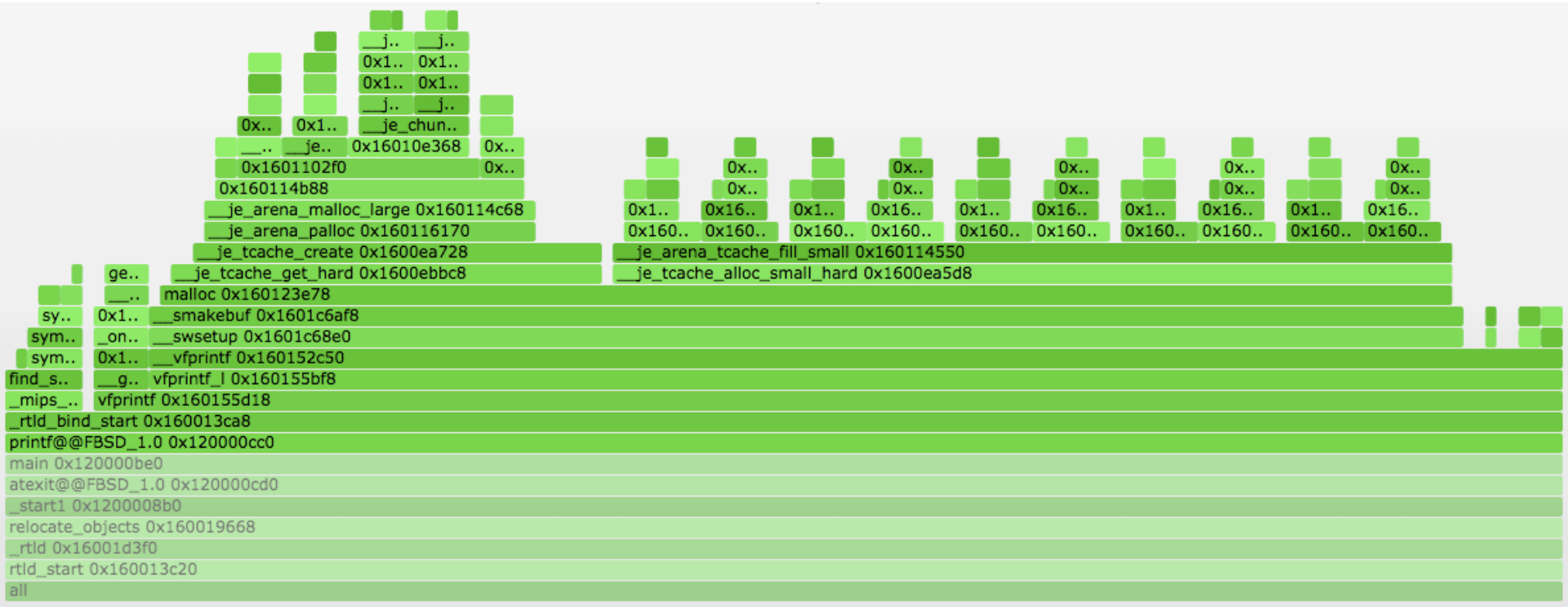
printf()


__j.. __j..
0x1.. 0x1..
0x1.. 0x1..
__j.. __j..
0x.. 0x1.. __je_chun..
__.. __je.. 0x16010e368 0x..
0x1601102f0 0x..
0x160114b88
__je_arena_malloc_large 0x160114c68
__je_arena_palloc 0x160116170
__je_tcache_create 0x1600ea728
__je_tcache_get_hard 0x1600ebbc8
malloc 0x160123e78
__smakebuf 0x1601c6af8
__swsetup 0x1601c68e0
__vfprintf 0x160152c50
vfprintf_l 0x160155bf8
vfprintf 0x160155d18
_start 0x160013ca8
printf@@FBSD_1.0 0x120000cc0
main 0x120000be0
atexit@@FBSD_1.0 0x120000cd0
_start1 0x1200008b0
relocate_objects 0x160019668
_rtld 0x1601d3f0
rtld_start 0x160013c20
all
0x1.. 0x16.. 0x160..
__je_arena_tcache_fill_small 0x160114550
__je_tcache_alloc_small_hard 0x1600ea5d8

_mips_rtld_bind()



printf()


__j.. __j..
0x1.. 0x1..
0x1.. 0x1..
__j.. __j..
0x.. 0x1.. __je_chun..
__.. __je.. 0x16010e368 0x..
0x1601102f0 0x..
0x160114b88
__je_arena_malloc_large 0x160114c68
__je_arena_palloc 0x160116170
__je_tcache_create 0x1600ea728
ge.. __je_tcache_get_hard 0x1600ebbc8
__.. malloc 0x160123e78
sy.. 0x1.. __smakebuf 0x1601c6af8
sym.. _on.. __swsetup 0x1601c68e0
sym.. 0x1.. __vfprintf 0x160152c50
find_s.. __g.. vfprintf_l 0x160155bf8
mips.. vfprintf 0x160155d18
_rtld_bind_start 0x160013ca8
printf@@FBSD_1.0 0x120000cc0
main 0x120000be0
atexit@@FBSD_1.0 0x120000cd0
_start1 0x1200008b0
relocate_objects 0x160019668
_rtld 0x16001d3f0
rtld_start 0x160013c20
all
0x..
0x1..
0x16..
0x160..
__je_arena_tcache_fill_small 0x160114550
__je_tcache_alloc_small_hard 0x1600ea5d8

QUESTIONS?

Feedback requested

- Was the talk interesting and/or helpful?
- What didn't make sense?
- What would you like have learned more (or less) about?

- brooks.davis@sri.com