



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

February 4<sup>th</sup>, 2017  
FOSDEM 2017  
Brussels, Belgium



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.



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"
```

# 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
```

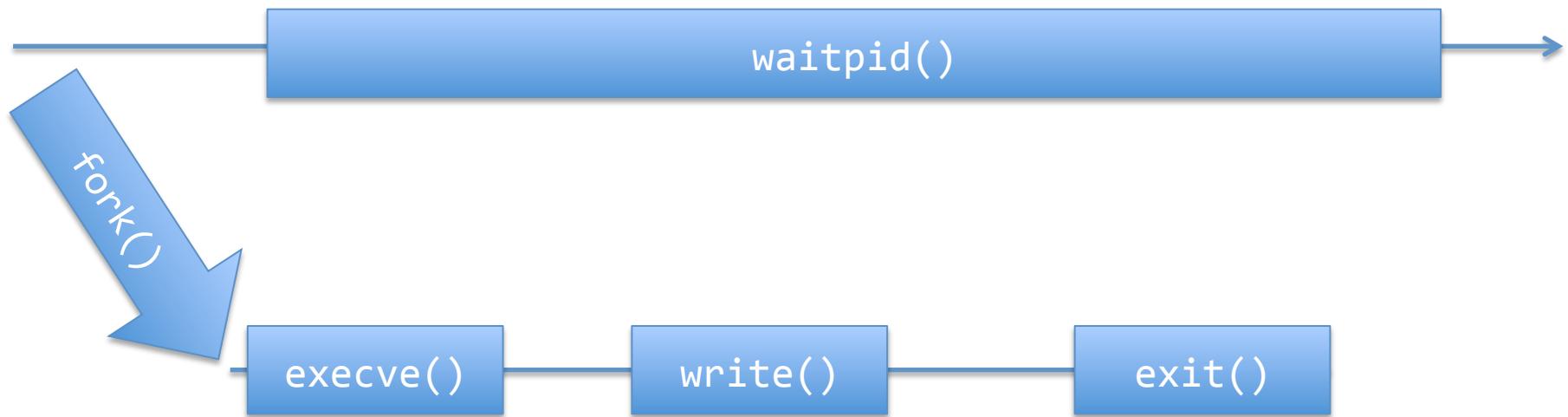
```
$ ld -EB -melf64btsmip_fbsd -Bstatic \
-o helloworld /usr/lib/crt1.o \
/usr/lib/crti.o /usr/lib/crtbeginT.o \
-L/usr/lib helloworld.o \
--start-group -lgcc -lgcc_eh -lc
--end-group \
/usr/lib/crtn.o /usr/lib/crtn.o
```

# Compiler runtime support

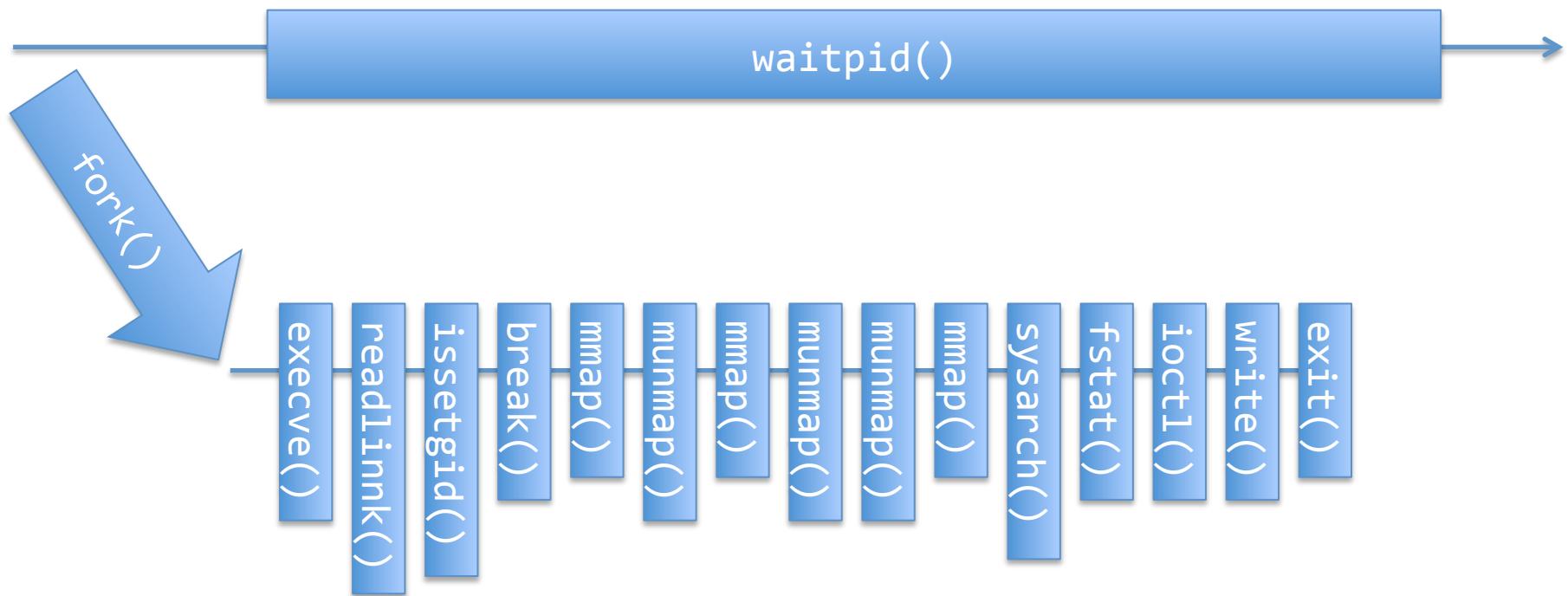
File	Purpose
crt1.o	Contains <code>__start()</code> function which initializes process environment and calls <code>main()</code> .
crti.o	Entry points for old style <code>_init()</code> and <code>_fini()</code> functions.
crtbegin.o crtbeginS.o crtbeginT.o	Declares <code>.ctor</code> and <code>.dtor</code> constructor and destructor sections. Declares functions to call constructors and destructors.
crtend.o	NULL terminates <code>.ctor</code> and <code>.dtor</code> sections.
crtn.o	Trailers for <code>_init()</code> and <code>_fini()</code> functions.

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

# Execution lifecycle (notional)



# Execution lifecycle (static)



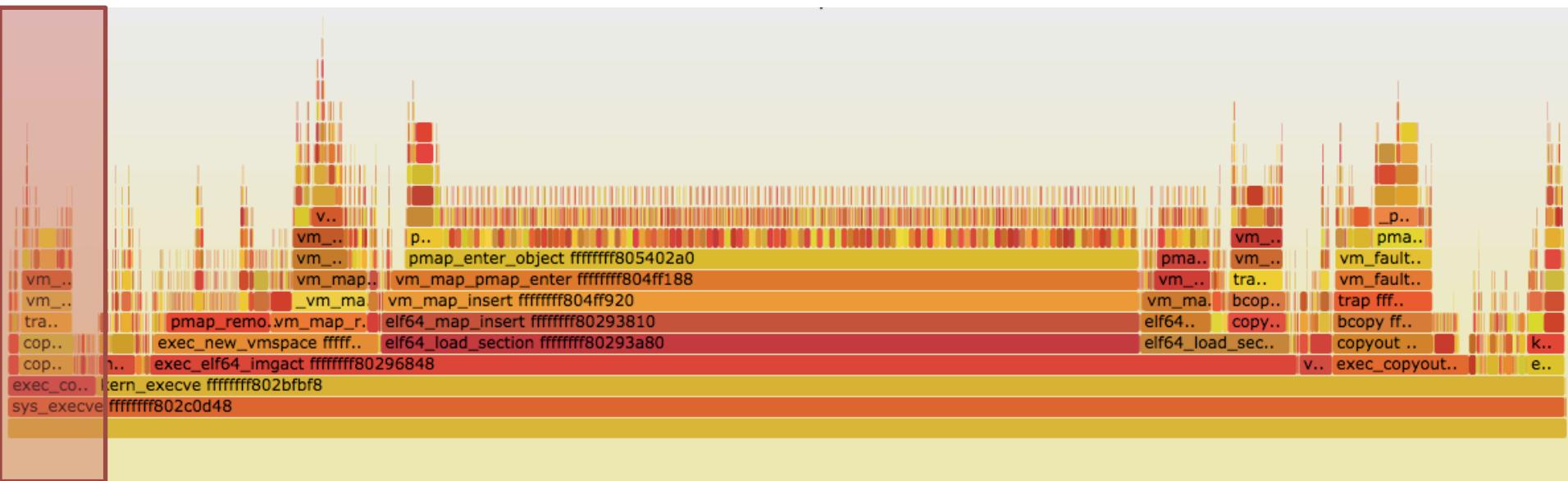
# Code and images online

<https://people.freebsd.org/~brooks/talks/fosdem2017-helloworld>

or

[\*\*http://bit.ly/helloworld-fosdem\*\*](http://bit.ly/helloworld-fosdem)

# execve()



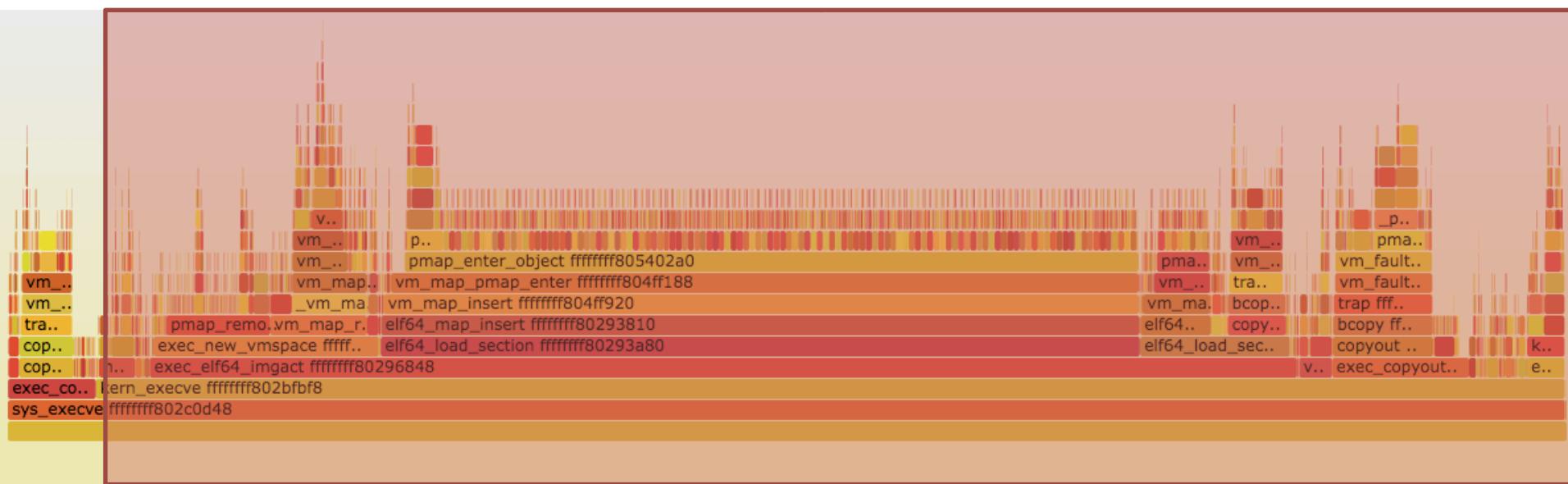
# exec\_copyin\_args()

Allocate  
memory

Copy in  
program  
path

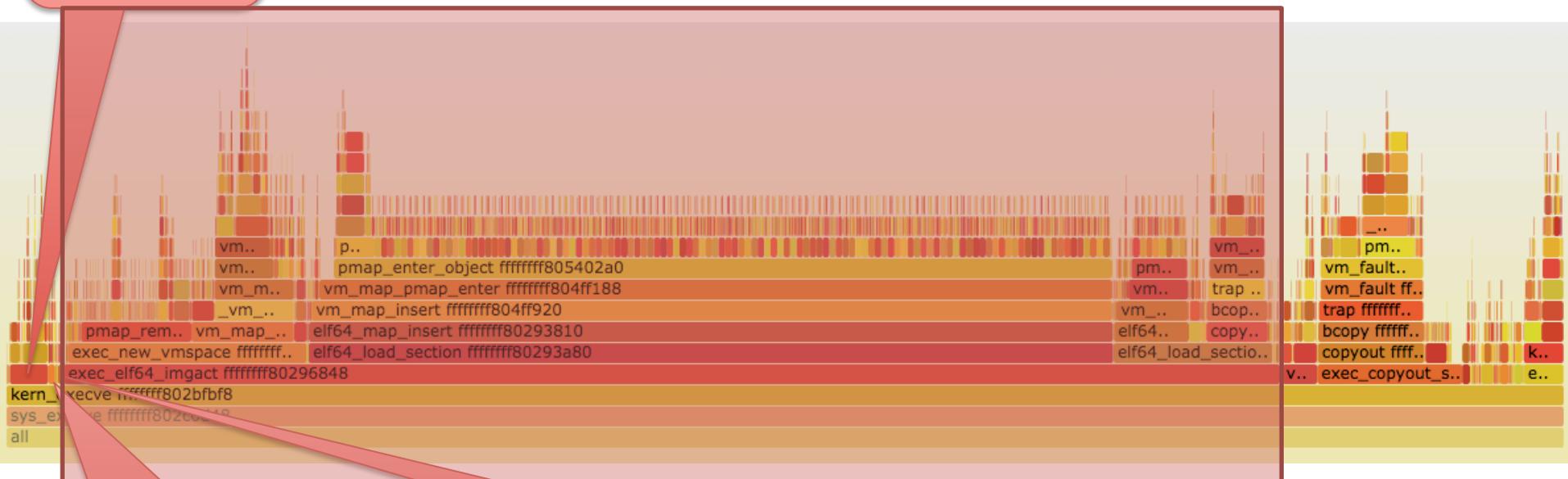


# sys\_execve()



# kern\_execve()

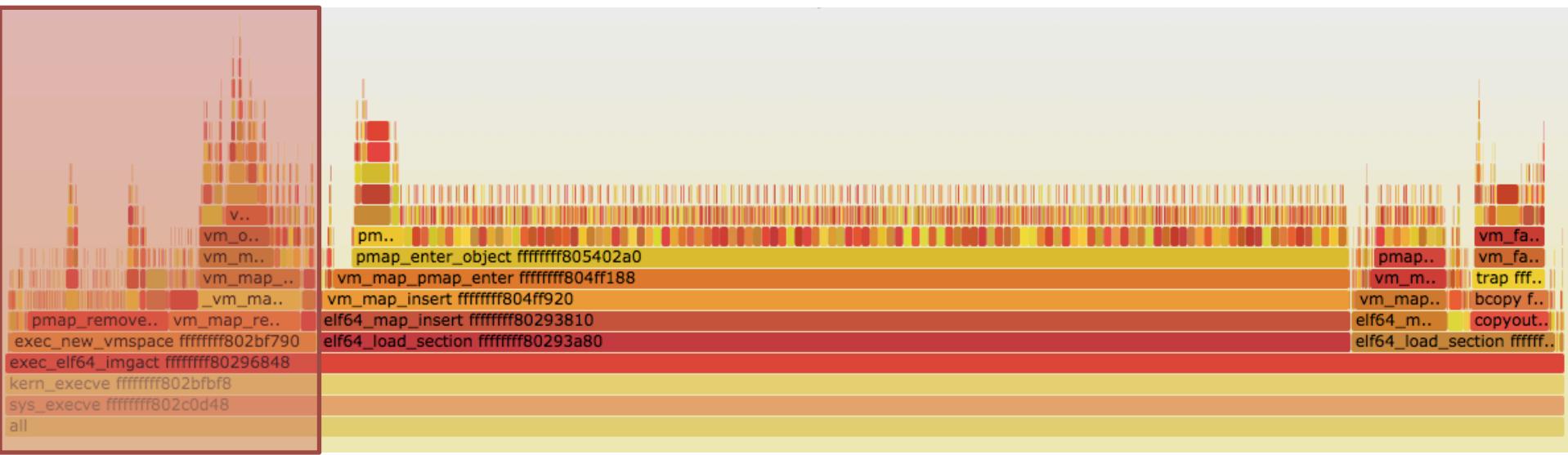
namei()  
Resolve  
path



`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()



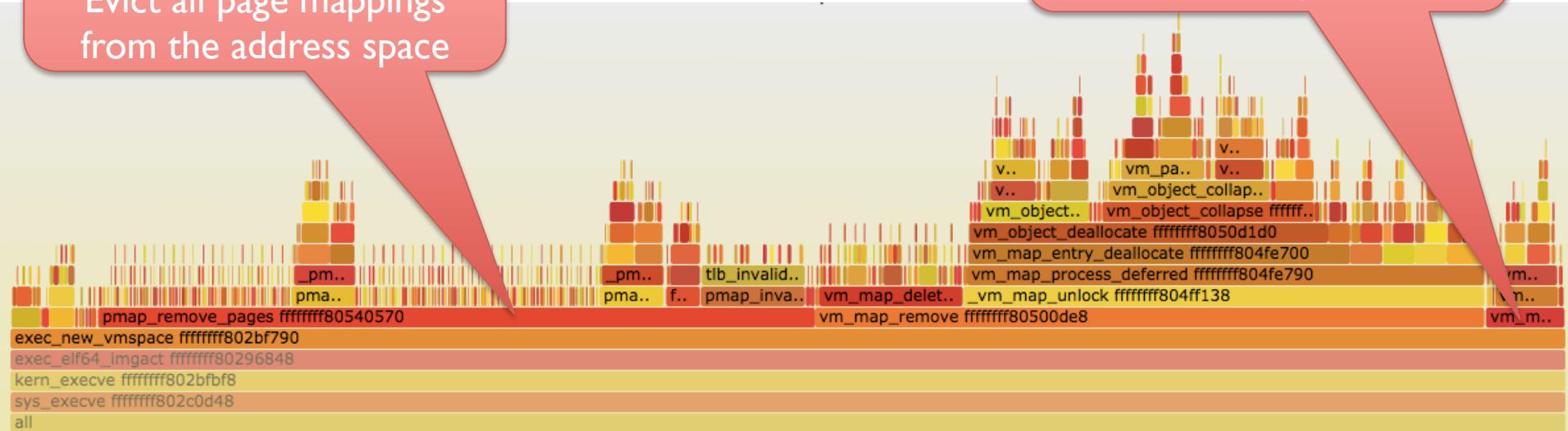
Defined with macros:

```
__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

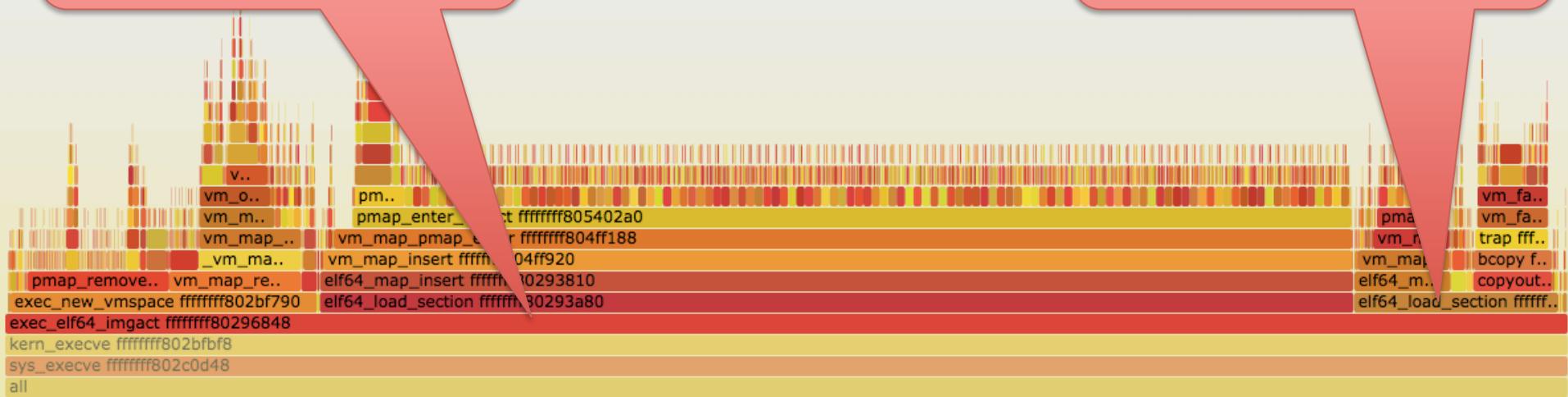


Stack

# exec\_elf64\_imgact()

`elf_load_section()`  
Map .text section into  
memory

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



.text

.data

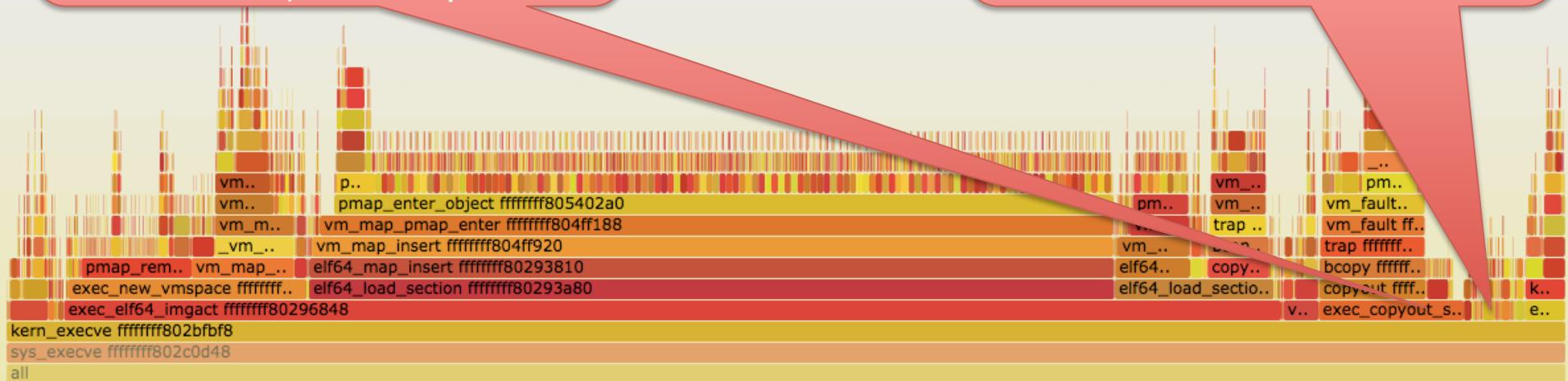
bss

Stack

# 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()`.



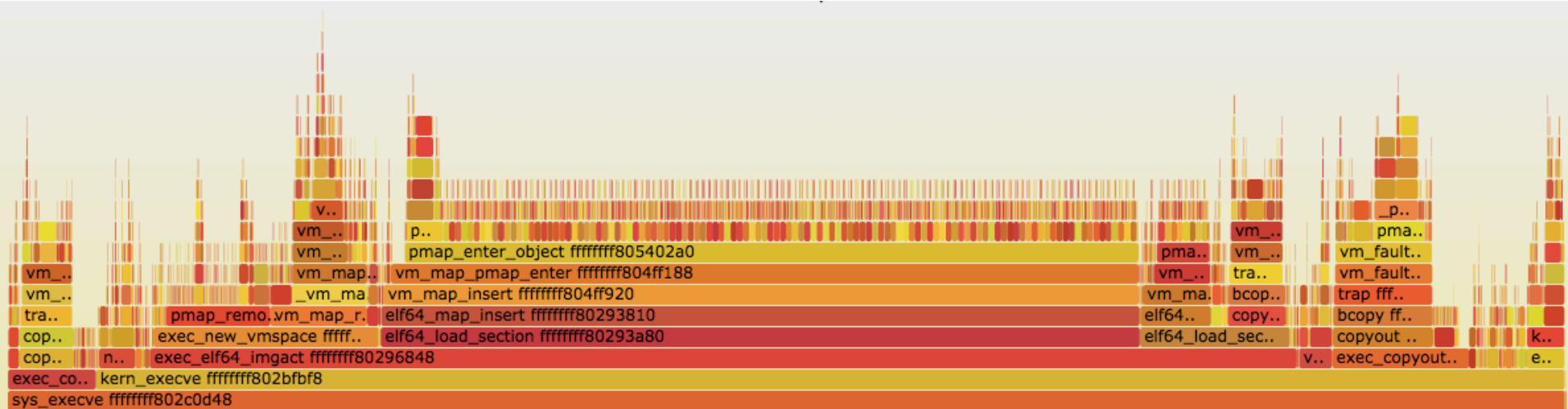
.text

.data

bss

Stack

# sys\_execve()



.text

.data

bss

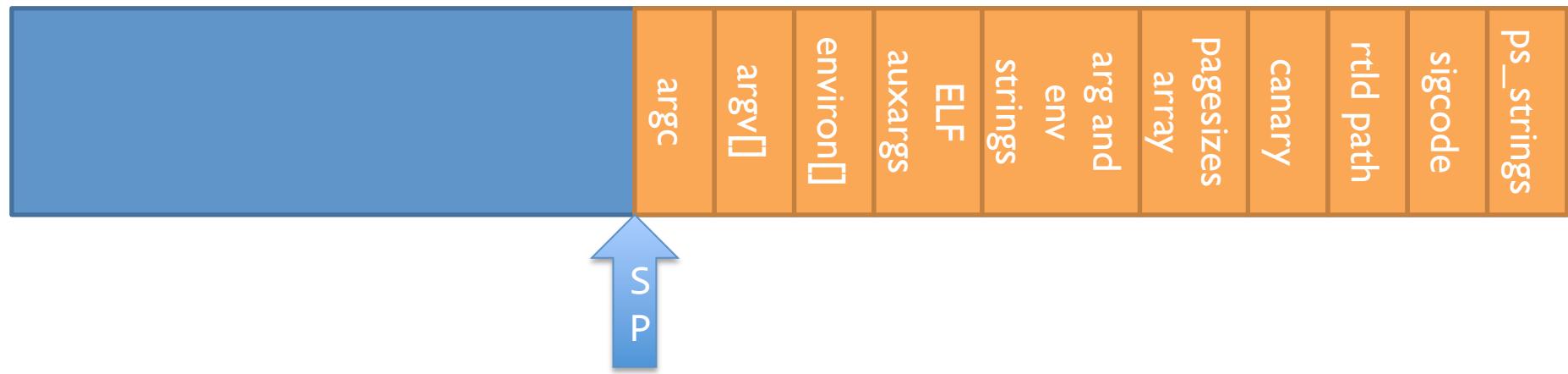
Stack

# 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()`

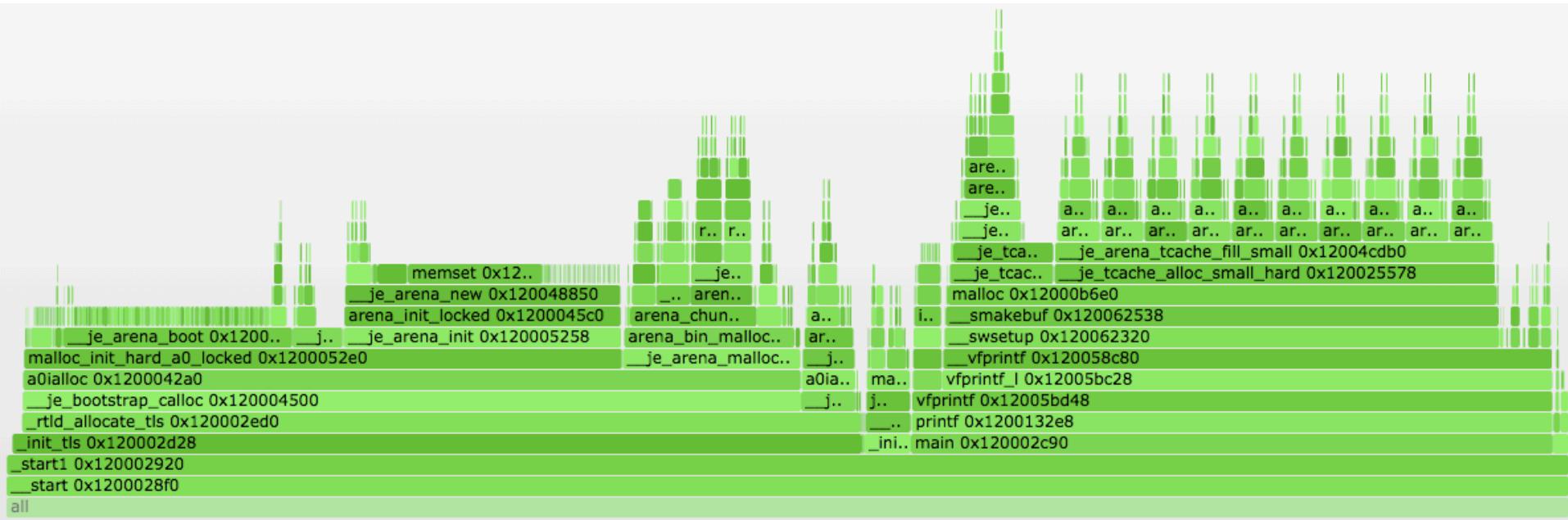


# SCO i386 ABI stack



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

# \_\_start()



## Most cycles spent in malloc()

# \_\_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

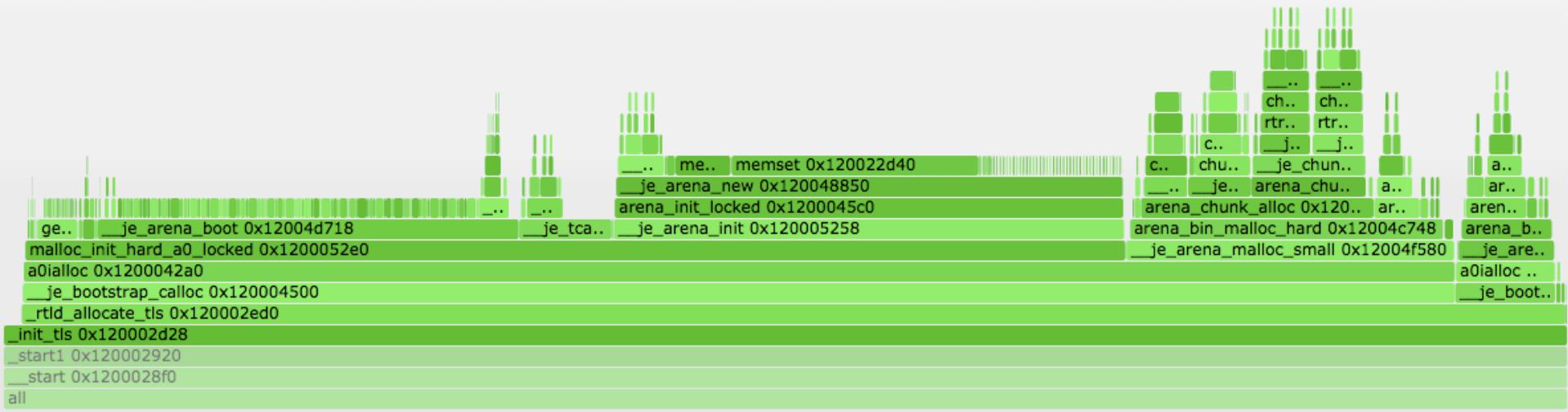
...

Set environ and  
\_\_progname variables.

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

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

# \_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



Uses JEMalloc, but  
JEMalloc uses TLS!

# \_\_start() 2/2

...

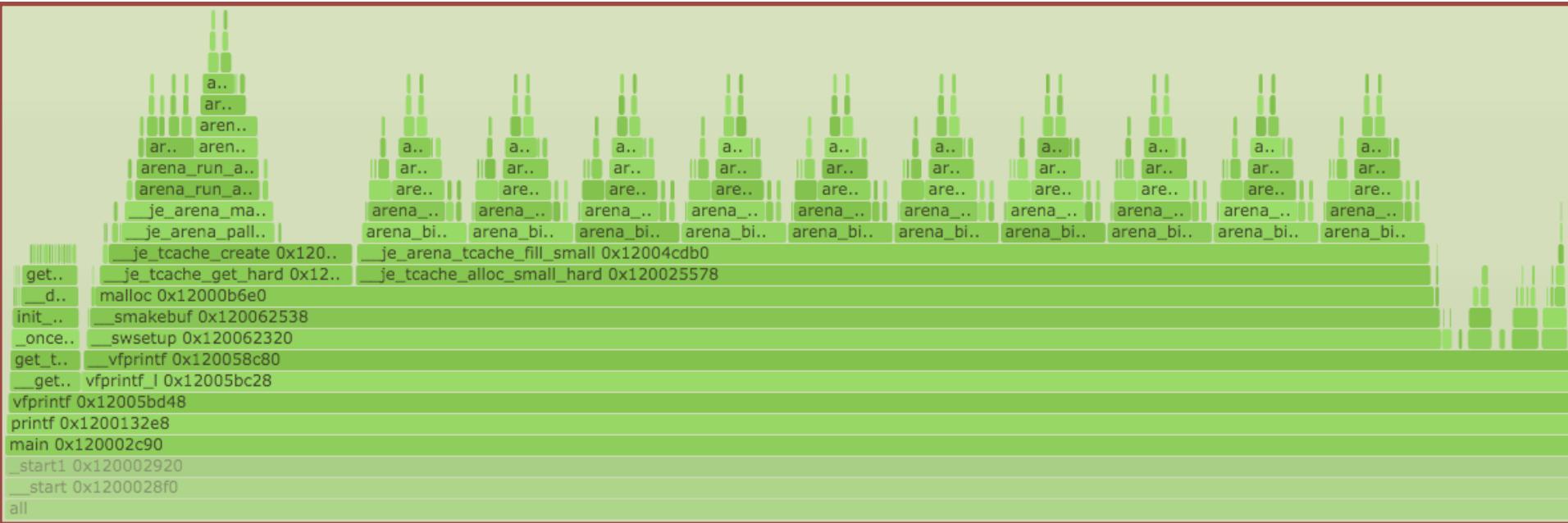
```
handle_argv(argc, argv, env);  
_init_tls();  
handle_static_init(argc, argv,  
env);
```

```
exit(main  
}
```

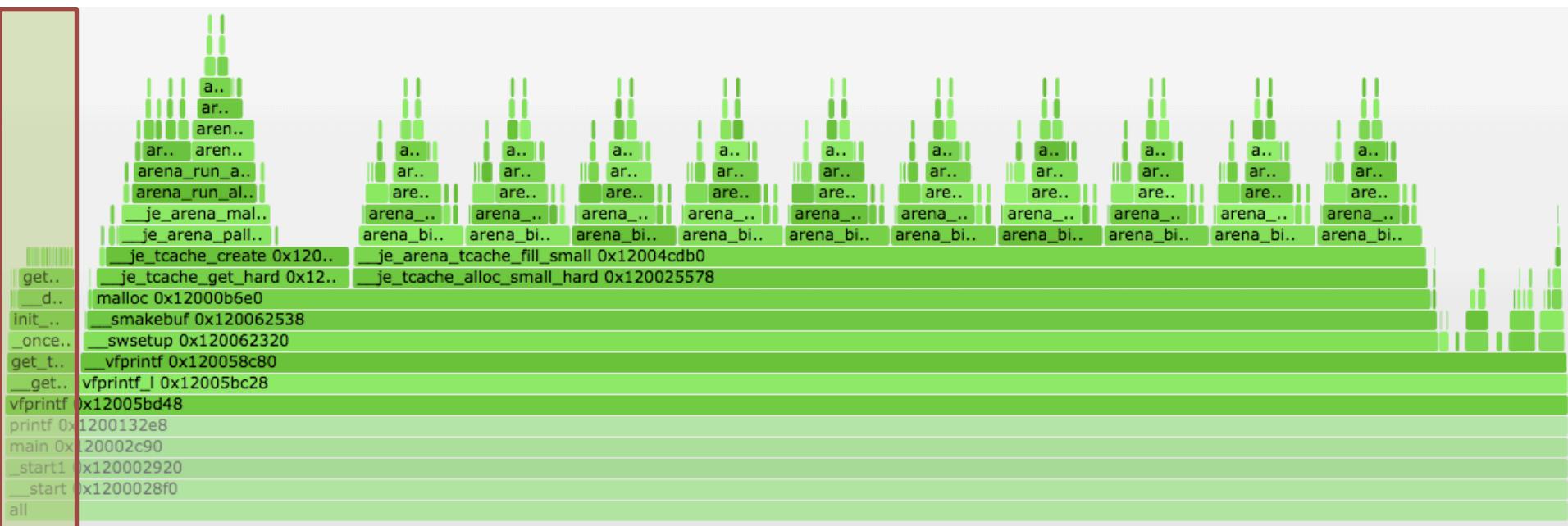
Calls constructors and registers  
destructors. Four types supported:

- .pre\_init\_array section
- \_\_init() function
- .ctors section (via \_\_init())
- .init\_array section

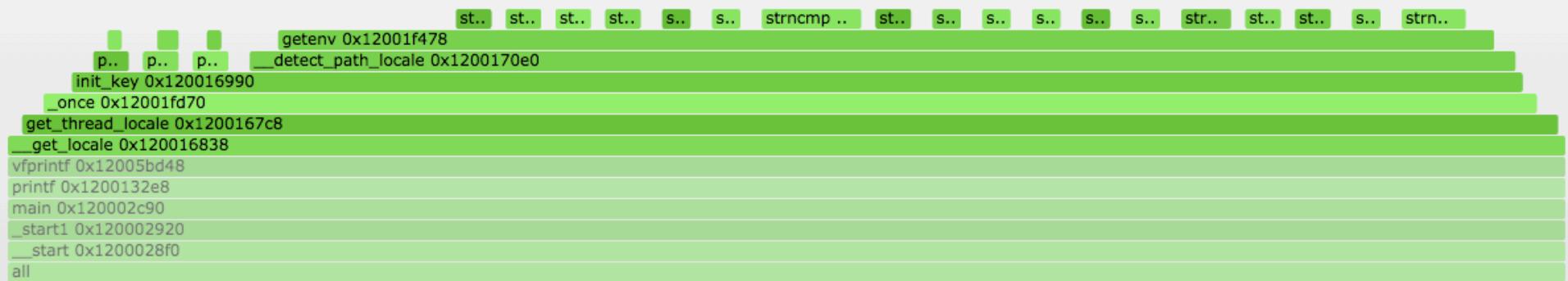
# main()



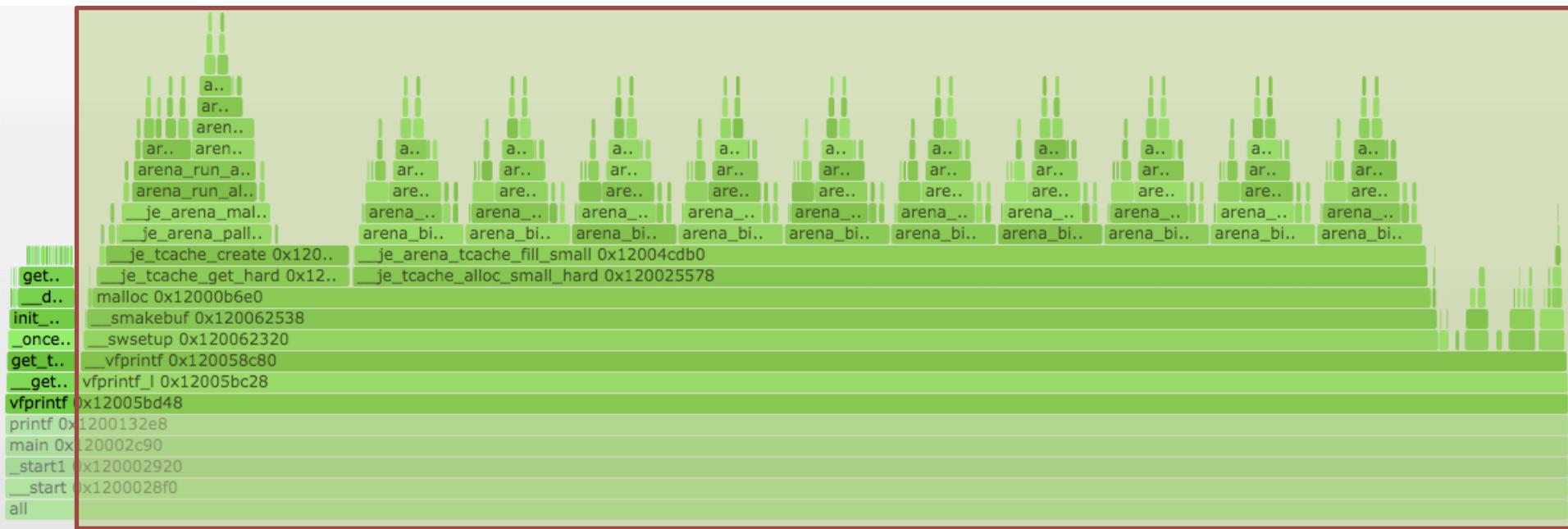
# vfprintf()



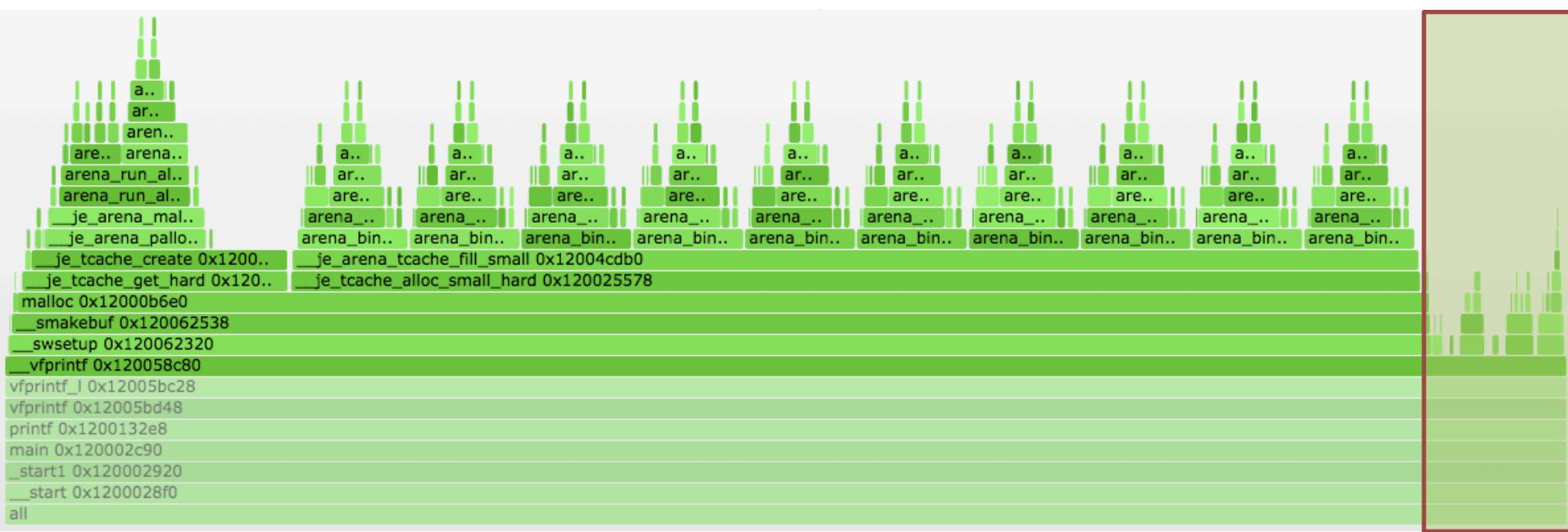
# \_\_get\_locale()



# vfprintf()



# \_\_vfprintf()



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

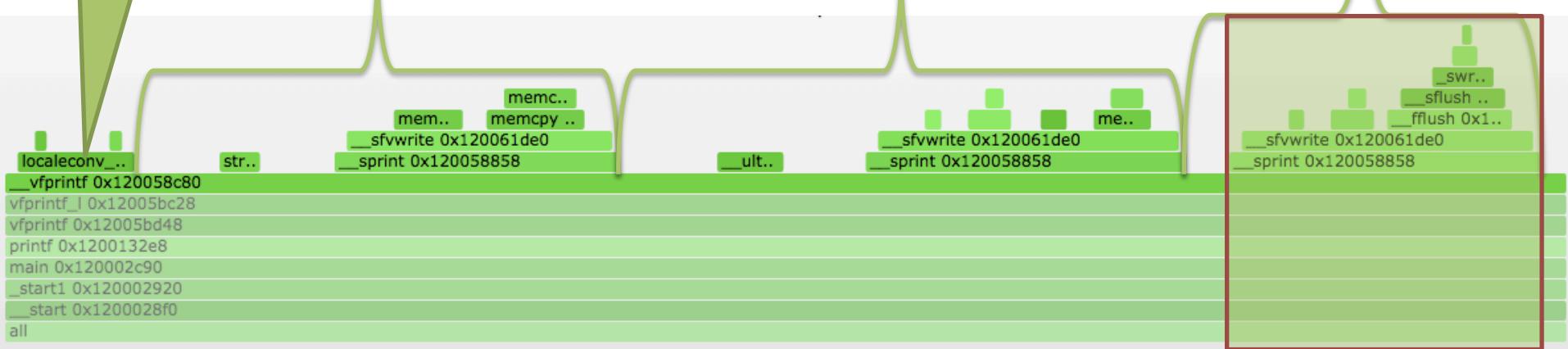
# vprintf()

Look up decimal  
point string.

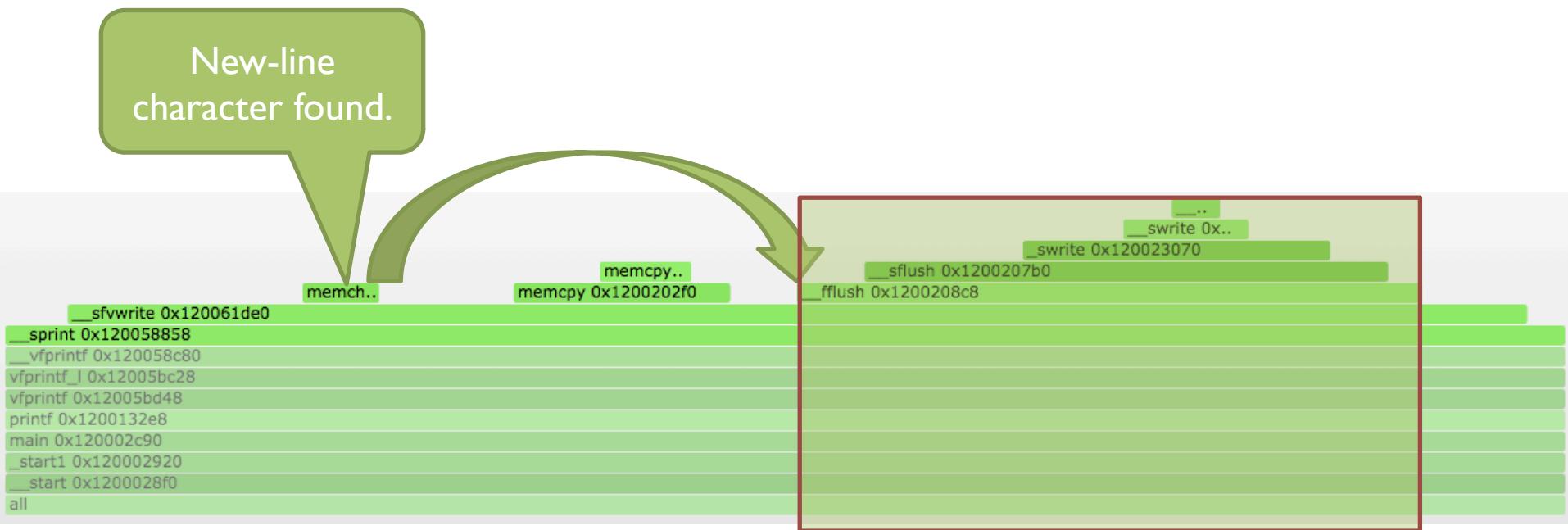
(“%s”, hello)

(“ %d”, 123)

(“\n”)

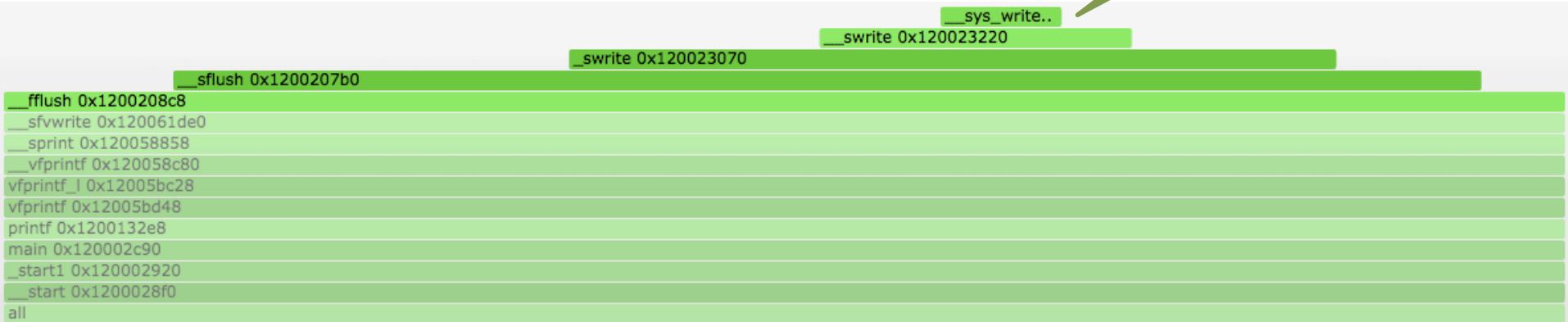


# \_\_sprint()



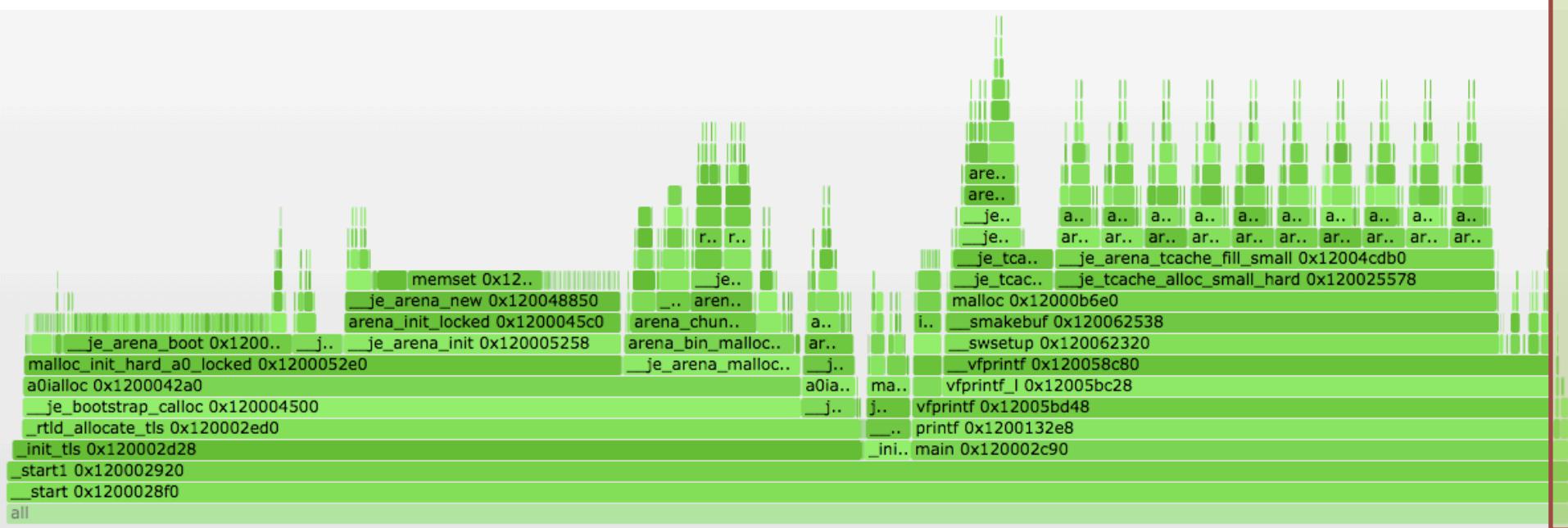
# \_\_flush()

The actual call  
to write()



hello, world 123

# \_\_start()

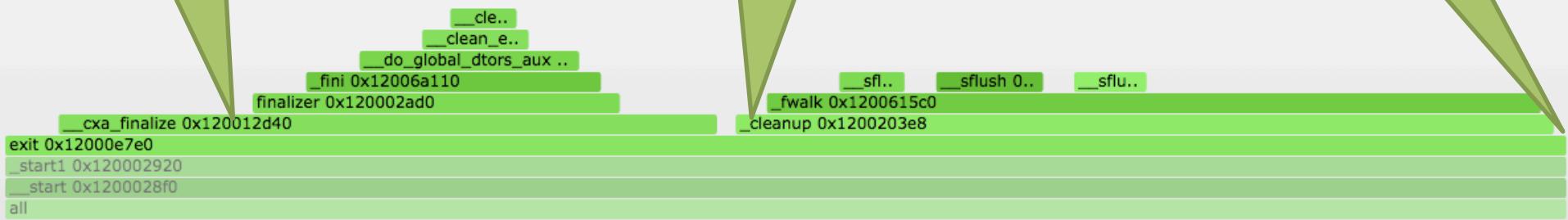


# exit()

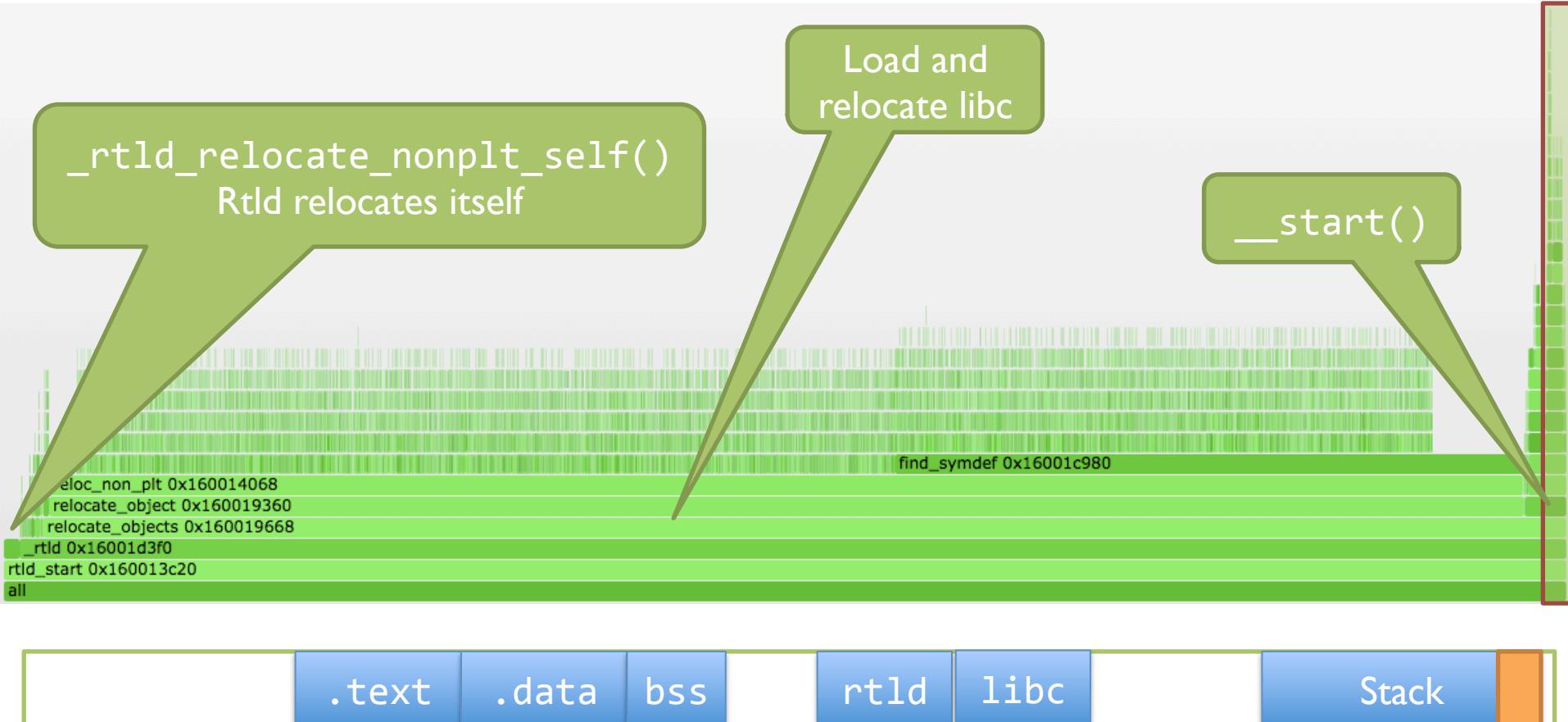
Call destructors registered with atexit()

Flush any unflushed FILEs

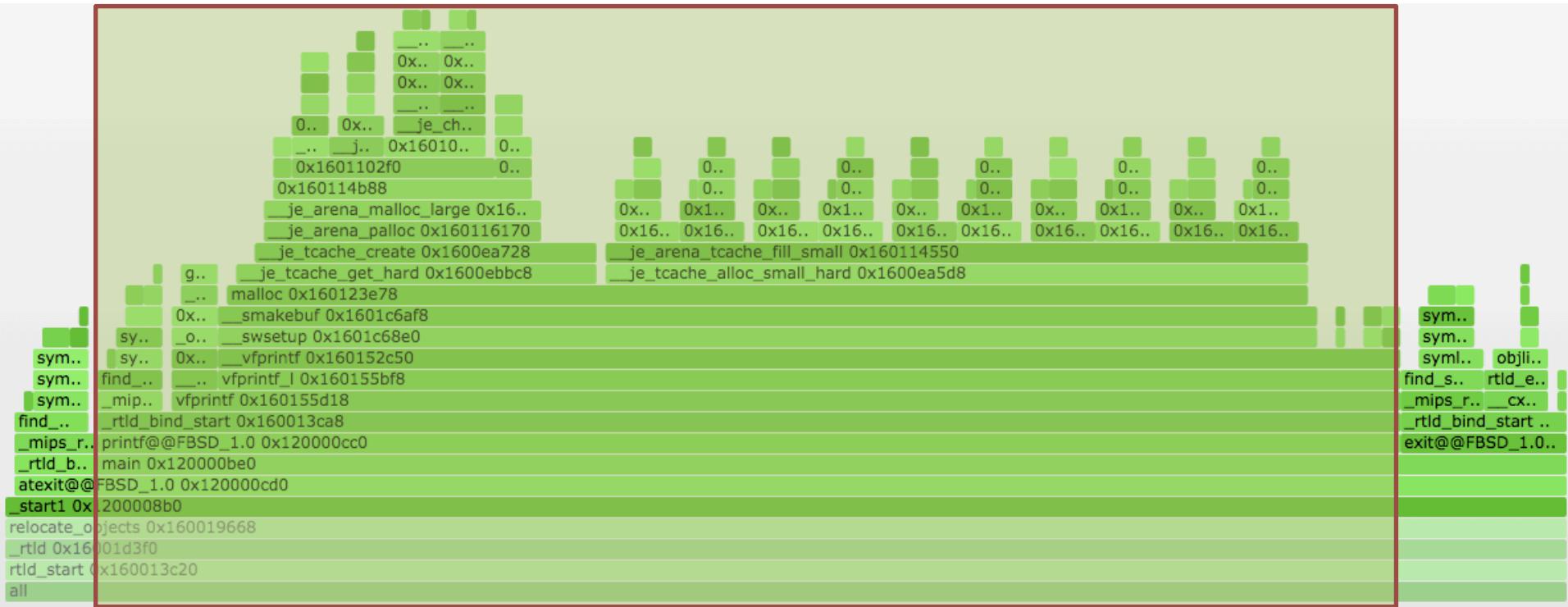
Call \_exit()



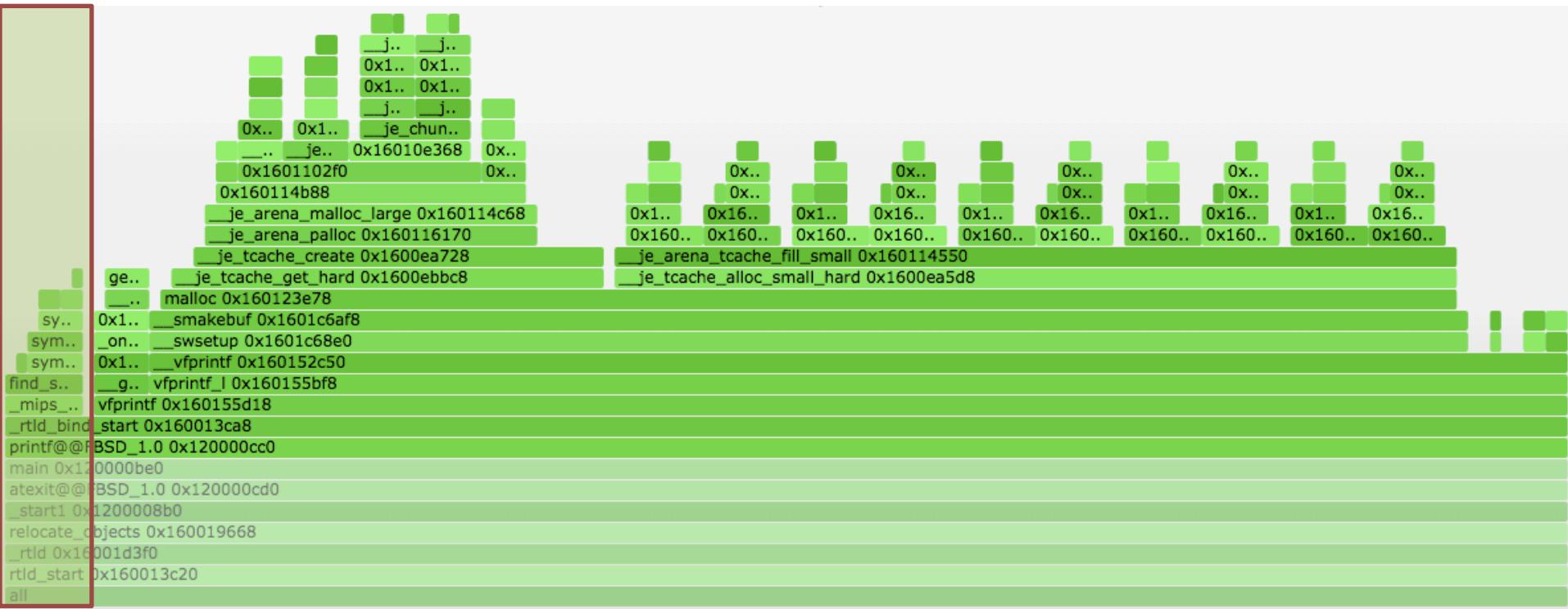
# Dynamic binary



# \_\_start()



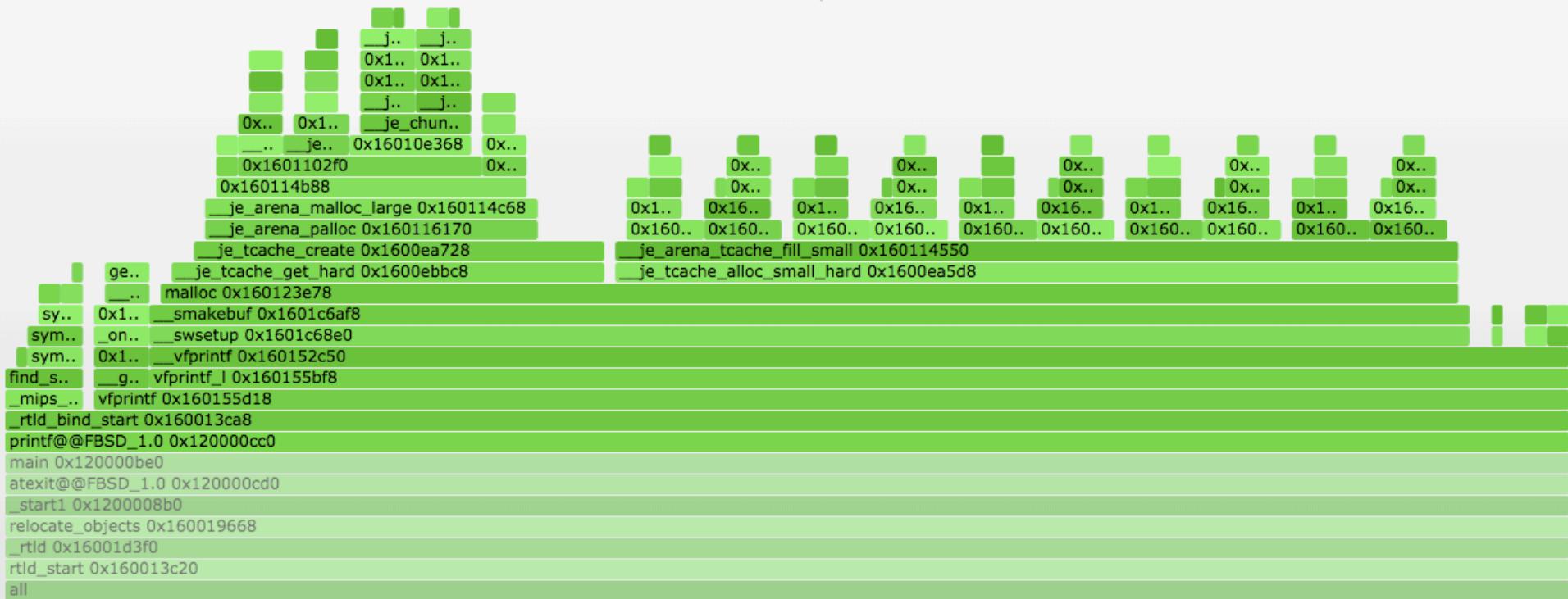
# printf()



# \_mips\_rtld\_bind()

```
matched_symbol 0x160...
symlook_obj 0x16001ba30
symlook_list 0x16001be78
symlook_global 0x16001c0d0
symlook_init 0x160015360 symlook_default 0x16001c6b0
find_symdef 0x16001c980
_mips_rtld_bind 0x160013fb8
_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
```

# printf()



# 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](mailto:brooks.davis@sri.com)