ftrace

Where modifying a running kernel all started!

Steven Rostedt
Open Source Engineer
rostedt@goodmis.org / srostedt@vmware.com

Ftrace Function hooks

- Allows attaching to a function in the kernel
  - Function Tracer
  - Function Graph Tracer
  - Perf
  - Stack Tracer
  - Kprobes
  - SystemTap
  - Pstore

Function Tracing

```
# cd /sys/kernel/tracing
# echo function > current_tracer
# cat trace
# tracer: function
#
# entries-in-buffer/entries-written: 159693/4101675   #P:4
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _----=> need-resched
#                            || / _---=> hardirq/softirq
#                            ||| / _--=> preempt-depth
#                            ||||/     delay
#           TASK-PID   CPU#  |||||   TIMESTAMP  FUNCTION
#              | |       |   |||||      |         |
            cat-3432  [002] d..... 60071.538270: __rcu_read_unlock <-__is_insn_slot_addr
            cat-3432  [002] d..... 60071.538270: is_bpf_text_address <-kernel_text_address
            cat-3432  [002] d..... 60071.538270: __rcu_read_lock <-is_bpf_text_address
            cat-3432  [002] d..... 60071.538271: bpf_prog_kallsyms_find <-is_bpf_text_address
            cat-3432  [002] d..... 60071.538271: __rcu_read_unlock <-is_bpf_text_address
            cat-3432  [002] d..... 60071.538271: init_object <-alloc_debug_processing
            cat-3432  [002] d..... 60071.538271: deactivate_slab.isra.74 <-___slab_alloc
            cat-3432  [002] d..... 60071.538272: preempt_count_add <-deactivate_slab.isra.74
            cat-3432  [002] d...1.. 60071.538272: preempt_count_sub <-deactivate_slab.isra.74
            cat-3432  [002] d..... 60071.538272: preempt_count_add <-deactivate_slab.isra.74
            cat-3432  [002] d...1.. 60071.538272: preempt_count_sub <-deactivate_slab.isra.74
            cat-3432  [002] d..... 60071.538273: preempt_count_add <-deactivate_slab.isra.74
            cat-3432  [002] d...1.. 60071.538273: preempt_count_sub <-deactivate_slab.isra.74
            cat-3432  [002] d..... 60071.538273: _raw_spin_lock <-deactivate_slab.isra.74
            cat-3432  [002] d..... 60071.538273: preempt_count_add <-_raw_spin_lock
            cat-3432  [002] d...1.. 60071.538273: do_raw_spin_trylock <-_raw_spin_lock
```

Function Graph Tracing

```
# cd /sys/kernel/tracing
# echo function_graph > current_tracer
# cat trace
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
 3)   0.868 us    |                  } /* rt_spin_lock_slowlock_locked */
 3)               |                  _raw_spin_unlock_irqrestore() {
 3)   0.294 us    |                    do_raw_spin_unlock();
 3)   0.374 us    |                    preempt_count_sub();
 3)   1.542 us    |                  }
 3)   0.198 us    |                  put_pid();
 3)   5.727 us    |                } /* rt_spin_lock_slowlock */
 3) + 18.867 us   |              } /* rt_spin_lock */
 3)               |              rt_spin_unlock() {
 3)               |                rt_mutex_futex_unlock() {
 3)               |                  _raw_spin_lock_irqsave() {
 3)   0.224 us    |                    preempt_count_add();
 3)   0.376 us    |                    do_raw_spin_trylock();
 3)   1.767 us    |                  }
 3)   0.264 us    |                  __rt_mutex_unlock_common();
 3)               |                  _raw_spin_unlock_irqrestore() {
 3)   0.278 us    |                    do_raw_spin_unlock();
 3)   0.249 us    |                    preempt_count_sub();
 3)   1.421 us    |                  }
 3)   4.565 us    |                }
 3)               |                migrate_enable() {
 3)   0.275 us    |                  preempt_count_add();
```

Dynamic Function Tracing

```
# cd /sys/kernel/tracing
# echo '*sched*' > set_ftrace_filter
# echo function > current_tracer
# cat trace
# tracer: function
#
# entries-in-buffer/entries-written: 35104/35104   #P:4
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _----=> need-resched
#                            || / _---=> hardirq/softirq
#                            ||| / _--=> preempt-depth
#                            ||||/     delay
#           TASK-PID   CPU#  |||||   TIMESTAMP  FUNCTION
#              | |       |   |||||      |         |
            bash-1294  [000] d..h... 60276.948739: tick_sched_timer <-__hrtimer_run_queues
            bash-1294  [000] d..h... 60276.948741: tick_sched_do_timer <-tick_sched_timer
            bash-1294  [000] d..h... 60276.948743: tick_sched_handle <-tick_sched_timer
            bash-1294  [000] d..h... 60276.948745: rcu_sched_clock_irq <-update_process_times
            bash-1294  [000] d..h... 60276.948745: scheduler_tick <-update_process_times
            bash-1294  [000] d...2.. 60276.948754: resched_curr_lazy <-check_preempt_wakeup
            bash-1294  [000] d.L.... 60276.948756: preempt_schedule_irq <-
restore_regs_and_return_to_kernel
     ksoftirqd/0-9     [000] ....... 60276.948769: schedule <-smpboot_thread_fn
            bash-1294  [000] d...311 60276.948908: resched_curr <-check_preempt_curr
            bash-1294  [000] d...311 60276.948908: native_smp_send_reschedule <-check_preempt_curr
          <idle>-0     [003] dn..1.. 60276.948922: smp_reschedule_interrupt <-reschedule_interrupt
          <idle>-0     [003] dn..1.. 60276.948923: scheduler_ipi <-reschedule_interrupt
```

How does it work?

- gcc’s profiler option: -pg
  - Adds a special “mcount” call to all non-inlined functions
  - mcount is a trampoline to jump to C code
  - All non-inlined functions call mcount near the beginning (after frame setup)
  - Requires frame pointers

How does it work?

- gcc's profiler option: -pg
  - Adds a special "mcount" call to all non-inlined functions
  - mcount is a trampoline to jump to C code
  - All non-inlined functions call mcount near the beginning (after frame setup)
  - Requires frame pointers
- x86 now only uses: -pg -mfentry
  - Adds a special "__fentry__" call to all non-inlined functions
  - __fentry__ is also a trampoline to jump to C code
  - All non-inlined function call __fentry__ **at** the beginning of the function
  - No need to have frame pointers

A Function Call

```
asmlinkage __visible void __sched schedule(void)
{
        struct task_struct *tsk = current;

        sched_submit_work(tsk);
        do {
                preempt_disable();
                __schedule(false);
                sched_preempt_enable_no_resched();
        } while (need_resched());
        sched_update_worker(tsk);
}
```

WARNING!

The following slides
may not be suitable for
some audiences

WARNING!

The next slide contains
ASSEMBLY!

Disassembled Function Call

```
<schedule>:
       53                      push   %rbx
       65 48 8b 1c 25 00 61    mov    %gs:0x16100,%rbx
       01 00
       ffffffff819dbce6: R_X86_64_32S  current_task
       48 8b 43 10             mov    0x10(%rbx),%rax
       48 85 c0                test   %rax,%rax
       74 10                   je     ffffffff819dbd03 <schedule+0x23>
       f6 43 24 20             testb  $0x20,0x24(%rbx)
       75 49                   jne    ffffffff819dbd42 <schedule+0x62>
       48 83 bb 20 0c 00 00    cmpq   $0x0,0xc20(%rbx)
       00
       74 1f                   je     ffffffff819dbd22 <schedule+0x42>
       31 ff                   xor    %edi,%edi
       e8 a6 f8 ff ff          callq  ffffffff819db5b0 <__schedule>
       65 48 8b 04 25 00 61    mov    %gs:0x16100,%rax
       01 00
```

Disassembled Function Call

With -pg -mfentry options

```
<schedule>:
        e8 1b d0 1e 00          callq  ffffffff81c01930 <__fentry__>
        ffffffff81a14911: R_X86_64_PLT32        __fentry__-0x4
        53                      push   %rbx
        65 48 8b 1c 25 00 61    mov    %gs:0x16100,%rbx
        01 00
        ffffffff81a1491b: R_X86_64_32S  current_task
        48 8b 43 10             mov    0x10(%rbx),%rax
        48 85 c0                test   %rax,%rax
        74 10                   je     ffffffff81a14938 <schedule+0x28>
        f6 43 24 20             testb  $0x20,0x24(%rbx)
        75 49                   jne    ffffffff81a14977 <schedule+0x67>
        48 83 bb 20 0c 00 00    cmpq   $0x0,0xc20(%rbx)
        00
        74 1f                   je     ffffffff81a14957 <schedule+0x47>
        31 ff                   xor    %edi,%edi
        e8 a1 f8 ff ff          callq  ffffffff81a141e0 <__schedule>
        65 48 8b 04 25 00 61    mov    %gs:0x16100,%rax
        01 00
```

At Kernel Boot Up

Where are all the __fentry__ callers?

Can’t just leave them there

- Too much overhead
- Just calling and doing a return adds 13% overhead!

Need to convert them to nops at boot up

Need to know where they are

Best to find them at compile time!

recordmcount

scripts/recordmcount.c (and there’s a perl version too!)

Reads the object files one at a time

Reads the relocation tables

- Finds all the calls to __fentry__
- Creates a table (array)
- Links them back into the object file
- New section called **__mcount_loc**
  - Even for __fentry__ locations

recordmcount

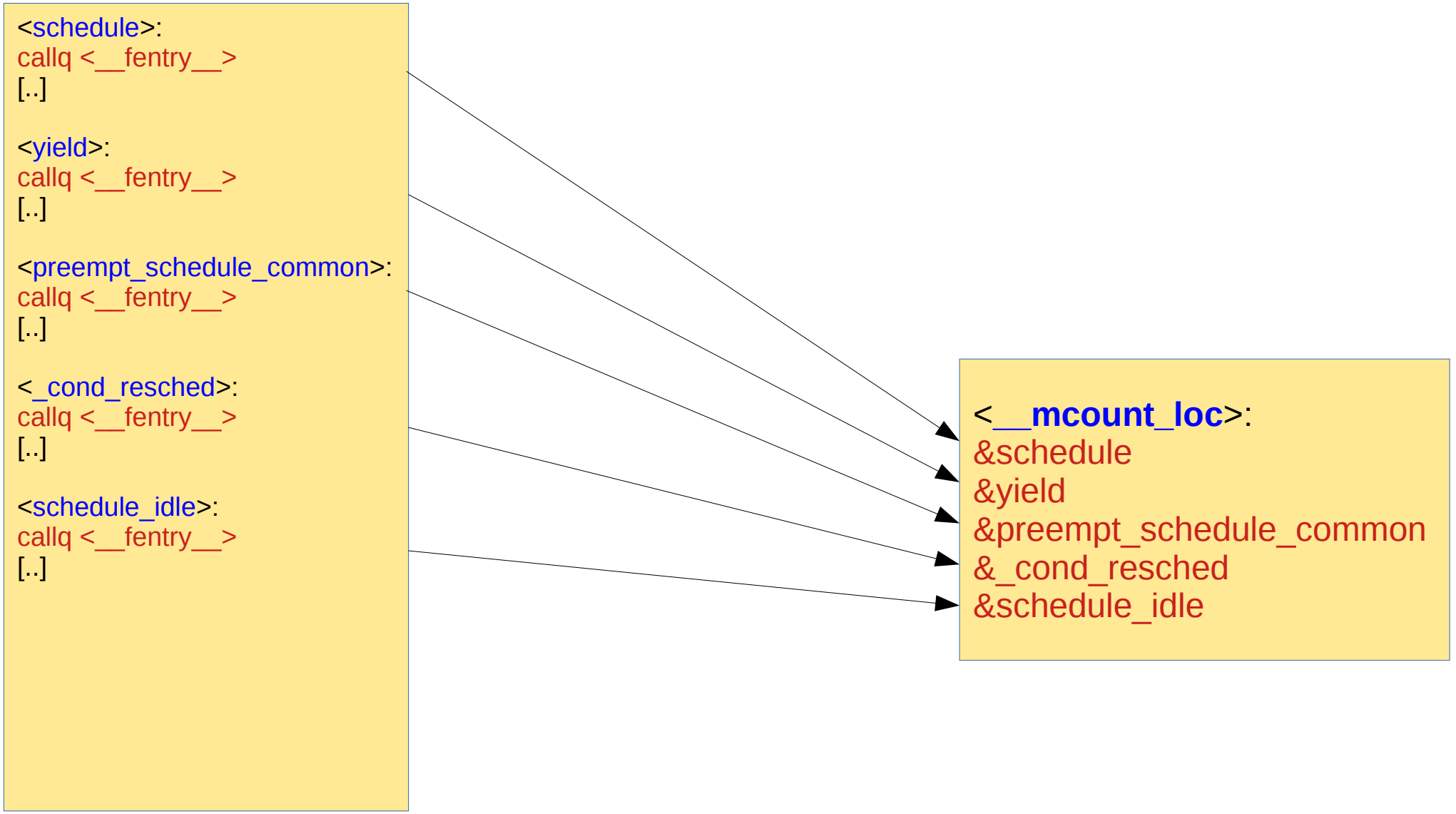
scripts/recordmcount.c (and there's a perl version too!)

Reads the object files one at a time

Reads the relocation tables

- Finds all the calls to __fentry__
- Creates a table (array)
- Links them back into the object file
- New section called **__mcount_loc**
  - Even for __fentry__ locations
- gcc 5 added **-mrecord-mcount** (to do this for us)

recordmcount (kernel/sched/core.o)



recordmcount (kernel/sched/core.o)

<schedule>:
callq <__fentry__>
[..]

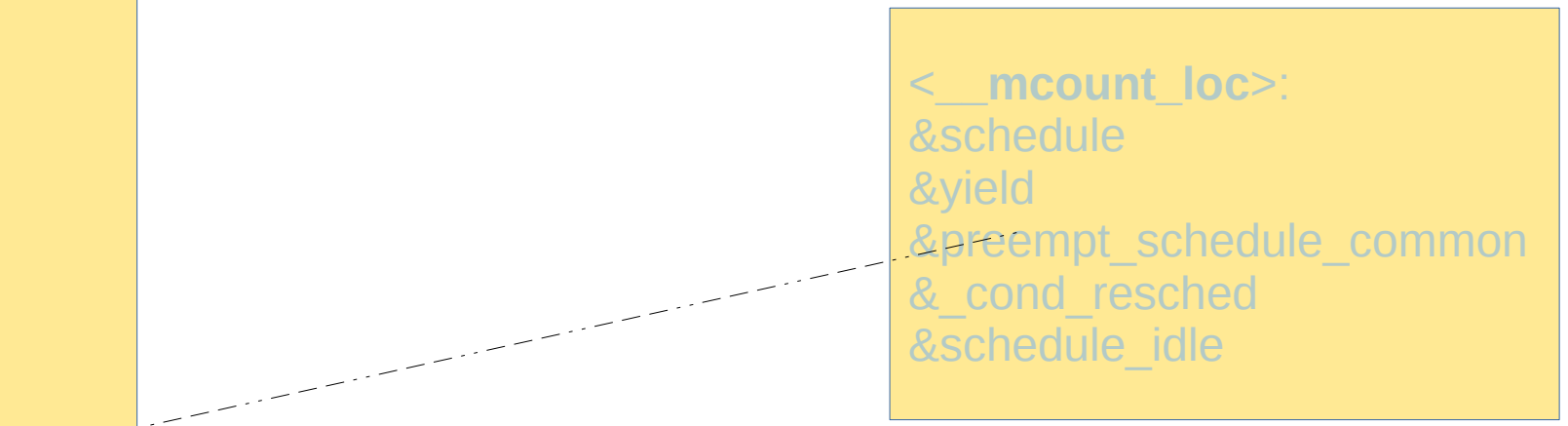
<yield>:
callq <__fentry__>
[..]

<preempt_schedule_common>:
callq <__fentry__>
[..]

<_cond_resched>:
callq <__fentry__>
[..]

<schedule_idle>:
callq <__fentry__>
[..]

<__**mcount_loc**>:
&schedule
&yield
&preempt_schedule_common
&_cond_resched
&schedule_idle



Linker Magic!

vmlinux.lds

- include/asm-generic/vmlinux.lds.h

Magic Variables

- __start_mcount_loc
- __stop_mcount_loc

```
#ifdef CONFIG_FTRACE_MCOUNT_RECORD
#ifdef CC_USING_PATCHABLE_FUNCTION_ENTRY
#define MCOUNT_REC()    . = ALIGN(8);                           \
                        __start_mcount_loc = .;                 \
                        KEEP(*(__patchable_function_entries))   \
                        __stop_mcount_loc = .;
#else
#define MCOUNT_REC()    . = ALIGN(8);                           \
                        __start_mcount_loc = .;                 \
                        KEEP(*(__mcount_loc))                   \
                        __stop_mcount_loc = .;
#endif
#else
#define MCOUNT_REC()
#endif
```

Linker Magic!

vmlinux.lds

- include/asm-generic/vmlinux.lds.h

Magic Variables

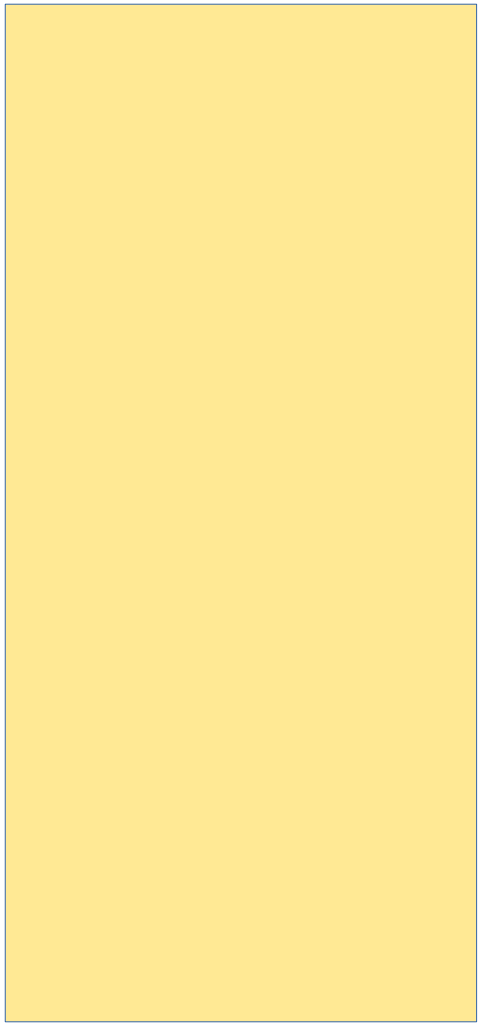
- __start_mcount_loc
- __stop_mcount_loc

parisc architecture

```
#ifdef CONFIG_FTRACE_MCOUNT_RECORD
#ifdef CC_USING_PATCHABLE_FUNCTION_ENTRY
#define MCOUNT_REC()    . = ALIGN(8);                           \
                        __start_mcount_loc = .;                 \
                        KEEP(*(__patchable_function_entries))   \
                        __stop_mcount_loc = .;

#else
#define MCOUNT_REC()    . = ALIGN(8);                           \
                        __start_mcount_loc = .;                 \
                        KEEP(*(__mcount_loc))                   \
                        __stop_mcount_loc = .;

#endif
#else
#define MCOUNT_REC()
#endif
```

Linker Magic

vmlinux:



kernel/sched/core.o:

```
<__mcount_loc>:
&schedule
&yield
&preempt_schedule_common
&_cond_resched
&schedule_idle
```

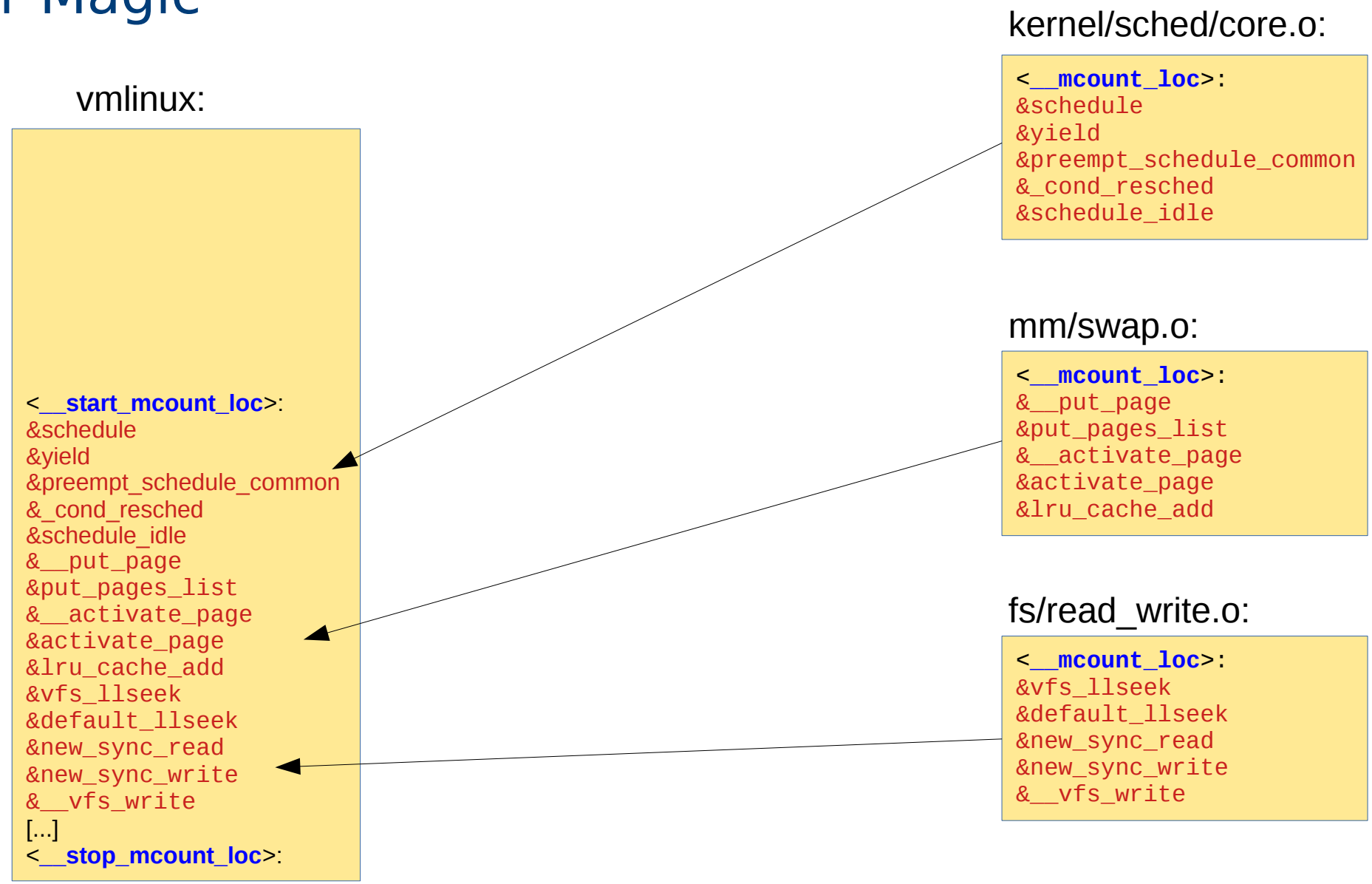
mm/swap.o:

```
<__mcount_loc>:
&__put_page
&put_pages_list
&__activate_page
&activate_page
&lru_cache_add
```

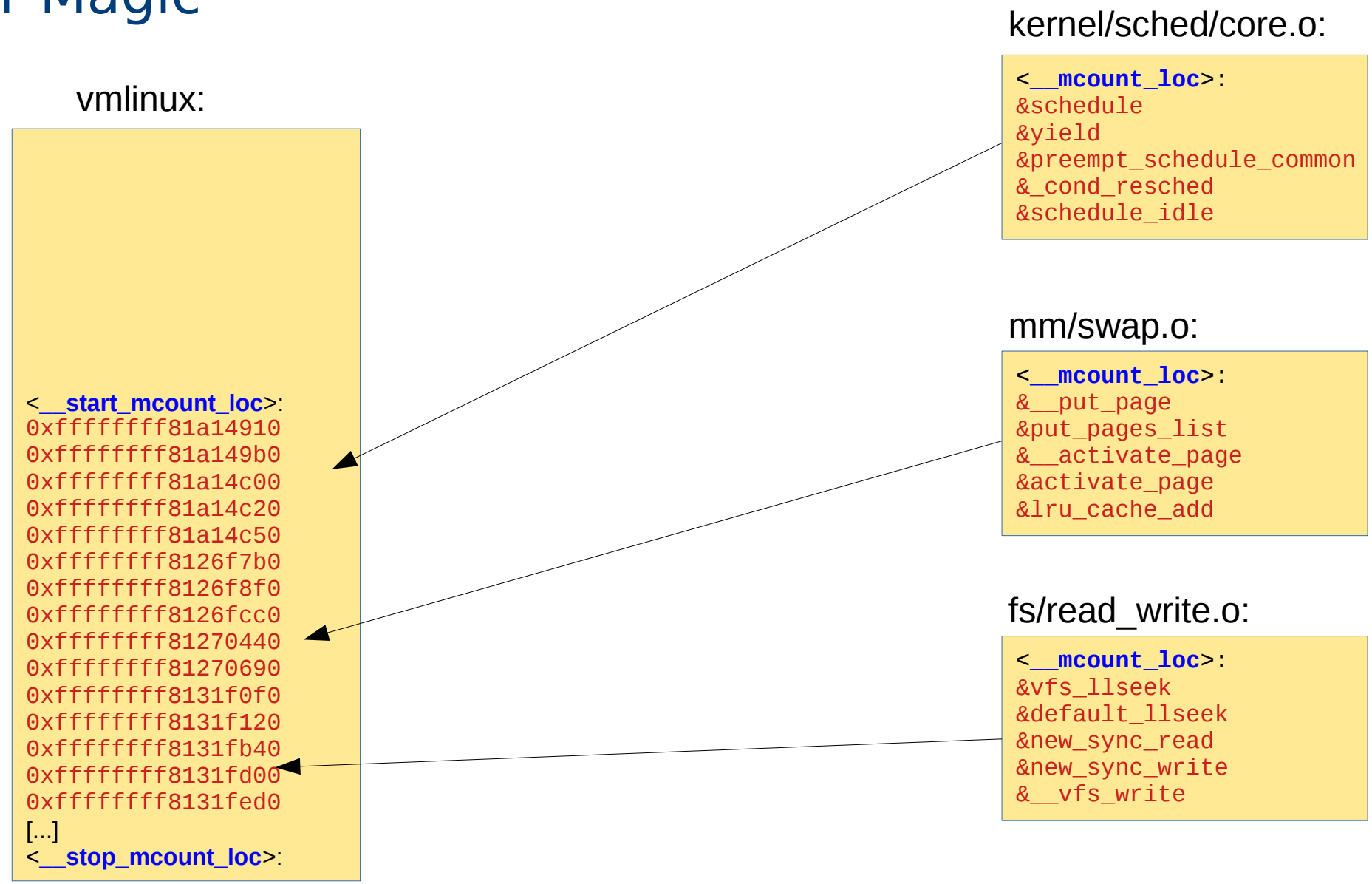
fs/read_write.o:

```
<__mcount_loc>:
&vfs_llseek
&default_llseek
&new_sync_read
&new_sync_write
&__vfs_write
```

Linker Magic



Linker Magic



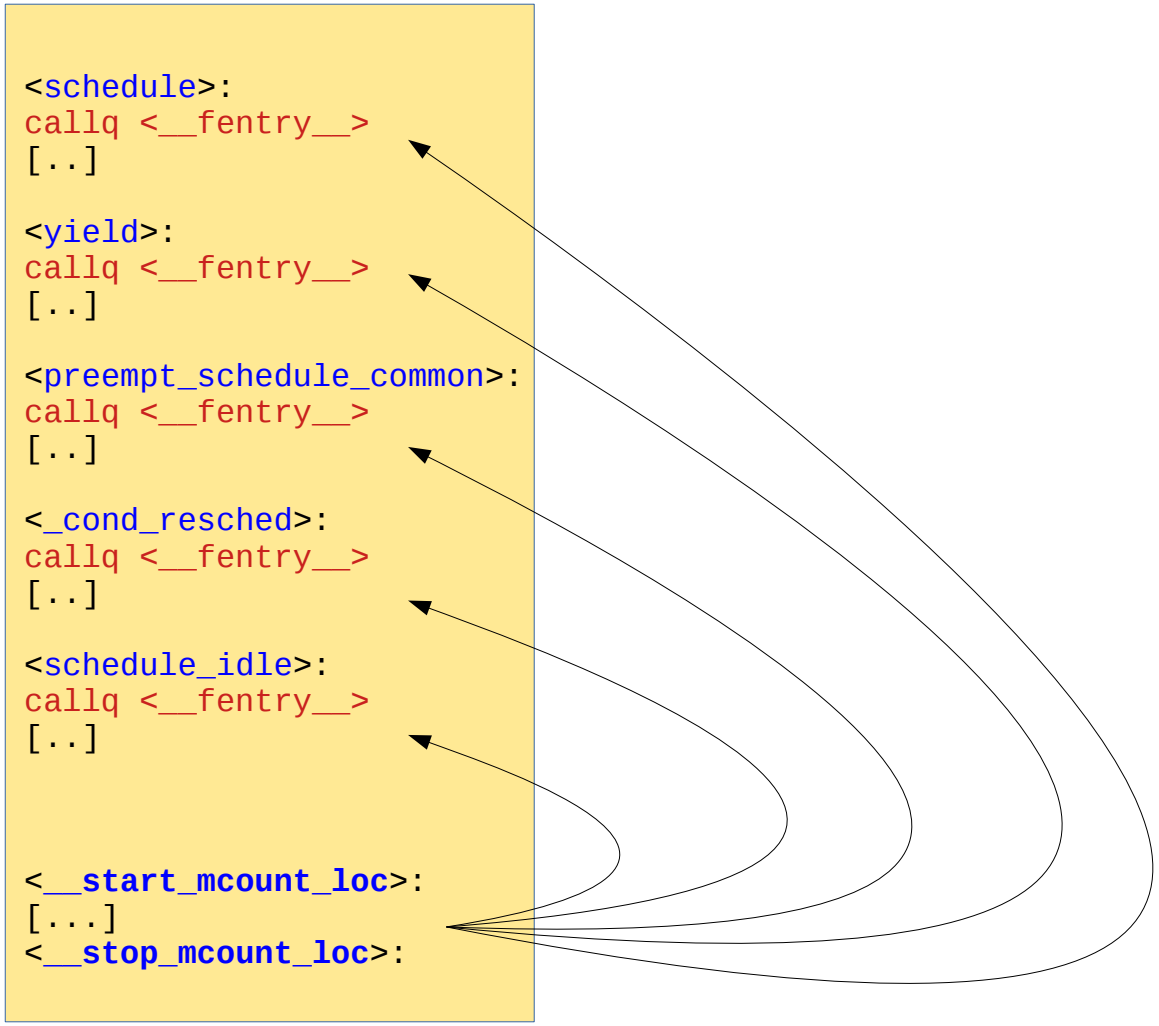
Finding __fentry__

vmlinux:

```
<schedule>:
callq <__fentry__>
[..]

<yield>:
callq <__fentry__>
[..]

<preempt_schedule_common>:
callq <__fentry__>
[..]

<_cond_resched>:
callq <__fentry__>
[..]

<schedule_idle>:
callq <__fentry__>
[..]



<__start_mcount_loc>:
[...]
<__stop_mcount_loc>:
```

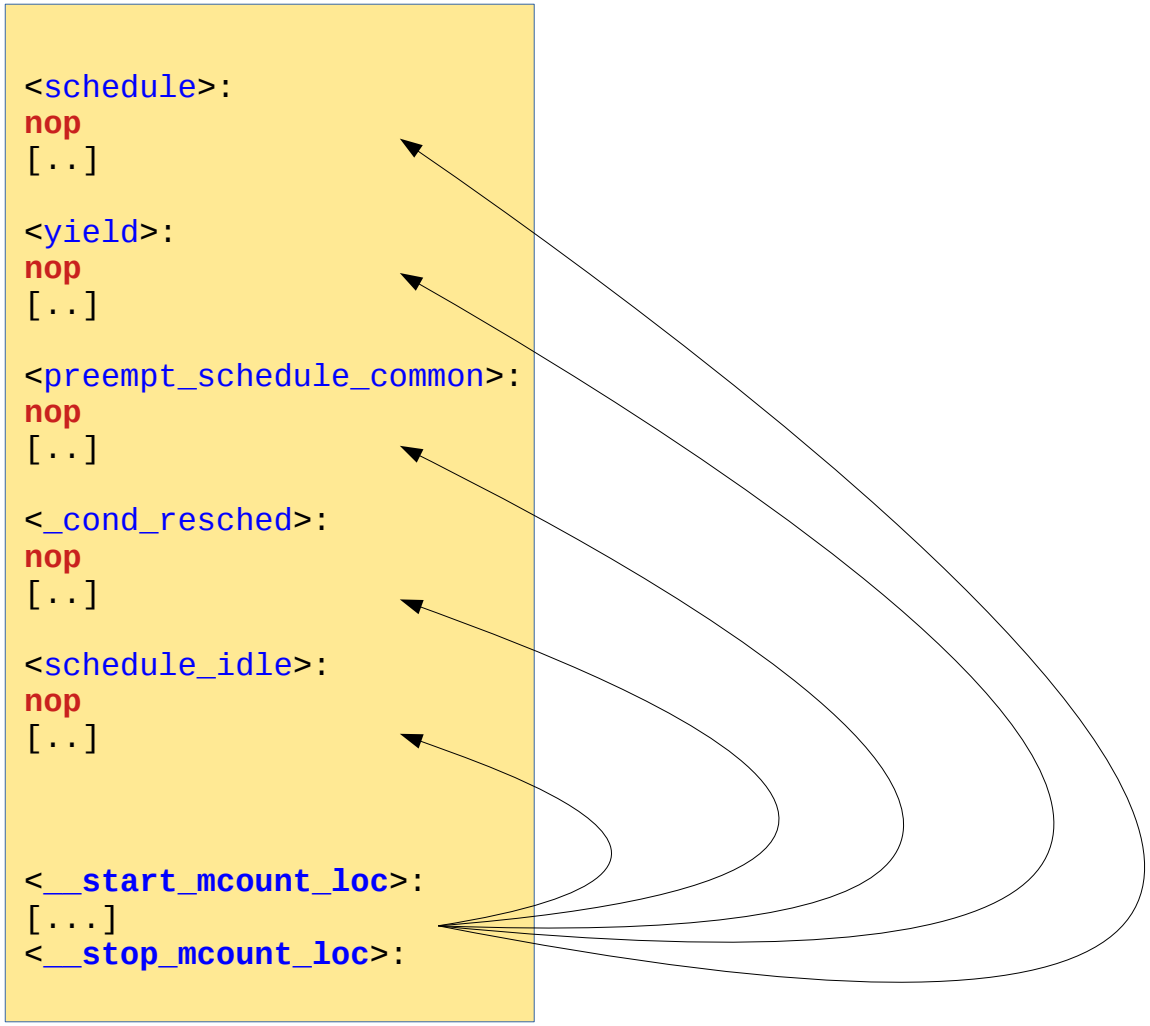
Finding __fentry__

vmlinux:



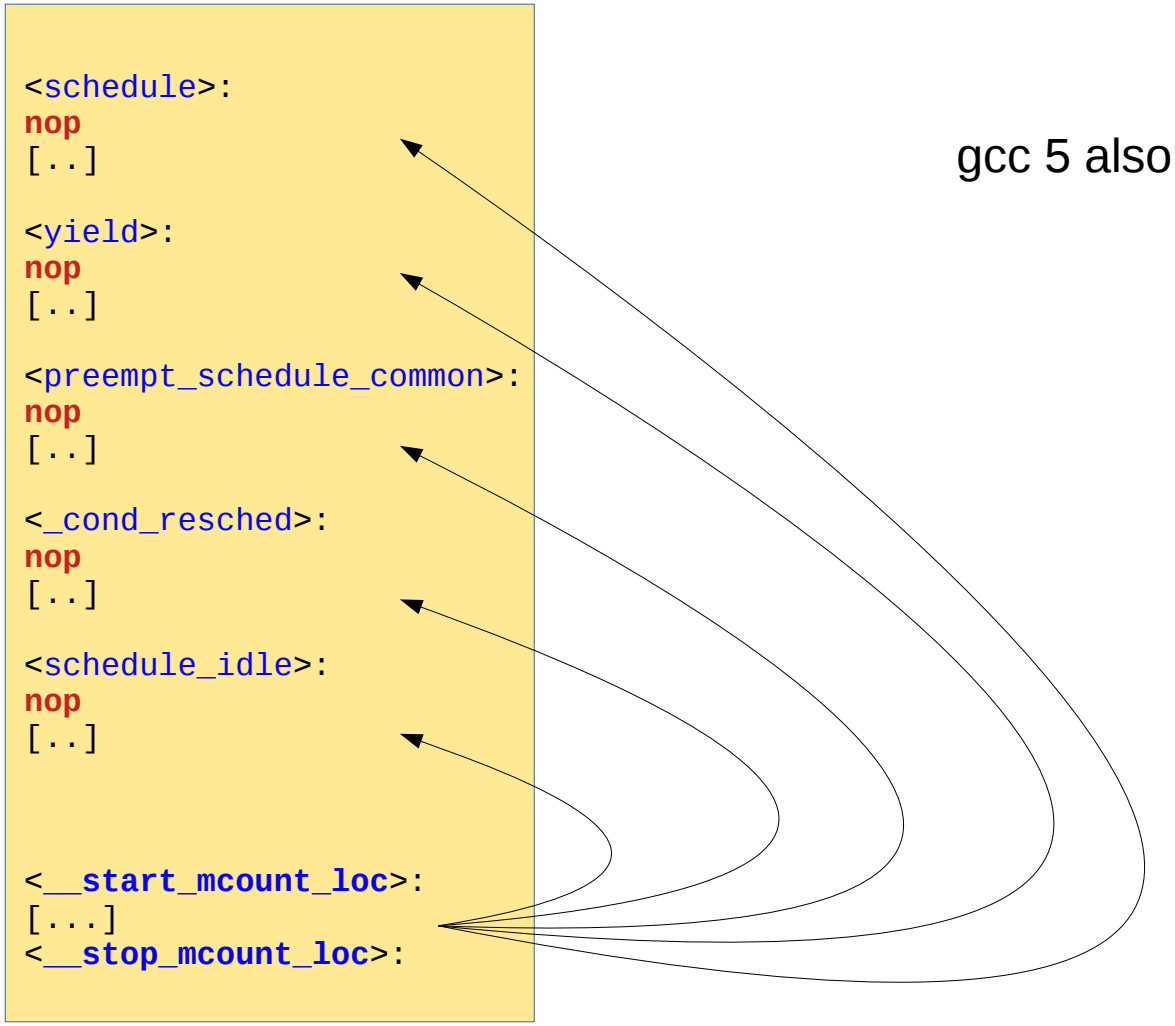
Finding __fentry__

vmlinux:



Finding __fentry__

vmlinux:



gcc 5 also added -mnop-mcount

Finding __fentry__

vmlinux:

```
<schedule>:
nop
[..]

<yield>:
nop
[..]

<preempt_schedule_common>:
nop
[..]

<_cond_resched>:
nop
[..]

<schedule_idle>:
nop
[..]



<__start_mcount_loc>:
[...]
<__stop_mcount_loc>:
```

Finding __fentry__

vmlinux:

```
<schedule>:
nop
[..]

<yield>:
nop
[..]

<preempt_schedule_common>:
nop
[..]

<_cond_resched>:
nop
[..]

<schedule_idle>:
nop
[..]
```

What about Tracing?

Need to know where to enable tracing

We threw away the __mcount_loc section

What about Tracing?

Need to know where to enable tracing

We threw away the __mcount_loc section

- The __mcount_loc section isn’t enough for us
- Tracing requires saving state

struct dyn_ftrace

```
struct dyn_ftrace {
        unsigned long           ip; /* address of mcount call-site */
        unsigned long           flags;
        struct dyn_arch_ftrace  arch;
};
```

struct dyn_ftrace

```
struct dyn_ftrace {
        unsigned long           ip; /* address of mcount call-site */
        unsigned long           flags;
        struct dyn_arch_ftrace  arch;
};

arch/x86/include/asm/ftrace.h:
struct dyn_arch_ftrace {
        /* No extra data needed for x86 */
};
```

struct dyn_ftrace

```
struct dyn_ftrace {
        unsigned long           ip; /* address of mcount call-site */
        unsigned long           flags;
        struct dyn_arch_ftrace  arch;
};

arch/powerpc/include/asm/ftrace.h:
struct dyn_arch_ftrace {
        struct module *mod;
};
```

Tracing data

Copy from __mcount_loc before deleting that section

Sorted for quick lookup

Allocated in groups of pages

- details out of scope for this talk

Data reported at boot up

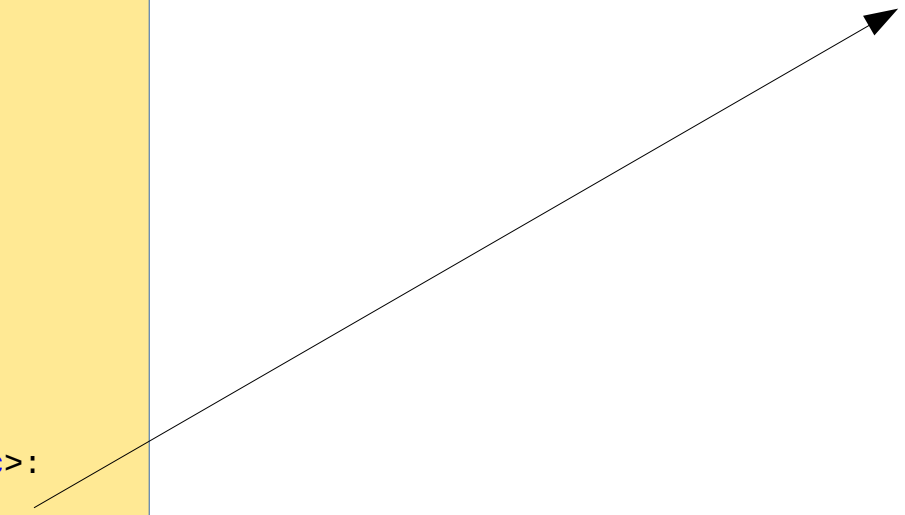
```
$ dmesg |grep ftrace
[    0.528844] ftrace: allocating 39317 entries in 154 pages
$ uname -r
5.1.11-200.fc29.x86_64
```

  - Allocated 39,317 dyn_ftrace structures
  - Used up 154 (4K) pages
  - Total of 630,784 bytes of memory

Finding __fentry__

vmlinux:

```
<schedule>:
nop
[..]

<yield>:
nop
[..]

<preempt_schedule_common>:
nop
[..]

<_cond_resched>:
nop
[..]

<schedule_idle>:
nop
[..]



<__start_mcount_loc>:
[...]
<__stop_mcount_loc>:
```



<ftrace_pages>

```
ip    = 0xffffffff81a14910
flags = 0
ip    = 0xffffffff81a149b0
flags = 0
ip    = 0xffffffff81a14c00
flags = 0
ip    = 0xffffffff81a14c20
flags = 0
ip    = 0xffffffff81a14c50
flags = 0
ip    = 0xffffffff8126f7b0
flags = 0
ip    = 0xffffffff8126f8f0
flags = 0
ip    = 0xffffffff8126fcc0
flags = 0
ip    = 0xffffffff81270440
flags = 0
ip    = 0xffffffff81270690
flags = 0
ip    = 0xffffffff8131f0f0
flags = 0
ip    = 0xffffffff8131f120
flags = 0
ip    = 0xffffffff8131fb40
flags = 0
ip    = 0xffffffff8131fd00
flags = 0
ip    = 0xffffffff8131fed0
flags = 0
```

Finding __fentry__

<ftrace_pages>

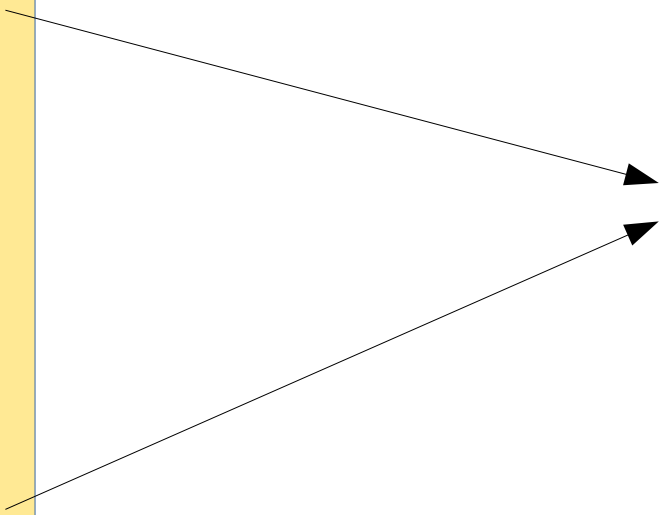
```
ip    = 0xffffffff81a14910
flags = 0
ip    = 0xffffffff81a149b0
flags = 0
ip    = 0xffffffff81a14c00
flags = 0
ip    = 0xffffffff81a14c20
flags = 0
ip    = 0xffffffff81a14c50
flags = 0
ip    = 0xffffffff8126f7b0
flags = 0
ip    = 0xffffffff8126f8f0
flags = 0
ip    = 0xffffffff8126fcc0
flags = 0
ip    = 0xffffffff81270440
flags = 0
ip    = 0xffffffff81270690
flags = 0
ip    = 0xffffffff8131f0f0
flags = 0
ip    = 0xffffffff8131f120
flags = 0
ip    = 0xffffffff8131fb40
flags = 0
ip    = 0xffffffff8131fd00
flags = 0
```

```
# cat available_filter_functions
schedule
yield
preempt_schedule_common
_cond_resched
schedule_idle
__put_page
put_pages_list
__activate_page
activate_page
lru_cache_add
vfs_llseek
default_llseek
new_sync_read
new_sync_write
```

Finding __fentry__

<ftrace_pages>

```
ip    = 0xffffffff81a14910
flags = 0
ip    = 0xffffffff81a149b0
flags = 0
ip    = 0xffffffff81a14c00
flags = 0
ip    = 0xffffffff81a14c20
flags = 0
ip    = 0xffffffff81a14c50
flags = 0
ip    = 0xffffffff8126f7b0
flags = 0
ip    = 0xffffffff8126f8f0
flags = 0
ip    = 0xffffffff8126fcc0
flags = 0
ip    = 0xffffffff81270440
flags = 0
ip    = 0xffffffff81270690
flags = 0
ip    = 0xffffffff8131f0f0
flags = 0
ip    = 0xffffffff8131f120
flags = 0
ip    = 0xffffffff8131fb40
flags = 0
ip    = 0xffffffff8131fd00
flags = 0
```



```
# echo default_llseek > set_ftrace_filter
# echo sched_idle >> set_ftrace_filtre
# cat set_ftrace_filter
schedule_idle
default_llseek
```

dyn_ftrace.flags

Bits 0-24: Counter for number of callbacks registered to function

Bit 25: Function is being initialized and not ready to touch

- module init

Bit 26: Return from callback may modify IP address

- kprobe or live patching

Bit 27: Has unique trampoline and its enabled

Bit 28: Has unique trampoline

Bit 29: Saves regs is enabled (see bit 30)

Bit 30: Needs to call ftrace_regs_caller (to save all regs like int3 does)

Bit 31: The function is being traced

dyn_ftrace.flags

Bits 0-24: Counter for number of callbacks registered to function

Bit 25: Function is being initialized and not ready to touch

- module init

Bit 26: Return from callback may modify IP address

- kprobe or live patching

Bit 27: Has unique trampoline and its enabled

Bit 28: Has unique trampoline

Bit 29: Saves regs is enabled (see bit 30)

Bit 30: Needs to call ftrace_regs_caller (to save all regs like int3 does)

Bit 31: The function is being traced

Finding __fentry__

vmlinux:

```
<schedule>:
nop
[..]

<yield>:
nop
[..]

<preempt_schedule_common>:
nop
[..]

<_cond_resched>:
nop
[..]

<schedule_idle>:
nop
[..]
```

<ftrace_pages>

```
ip    = 0xffffffff81a14910
flags = 0
ip    = 0xffffffff81a149b0
flags = 0
ip    = 0xffffffff81a14c00
flags = 0
ip    = 0xffffffff81a14c20
flags = 0
ip    = 0xffffffff81a14c50
flags = 0
ip    = 0xffffffff8126f7b0
flags = 0
ip    = 0xffffffff8126f8f0
flags = 0
ip    = 0xffffffff8126fcc0
flags = 0
ip    = 0xffffffff81270440
flags = 0
ip    = 0xffffffff81270690
flags = 0
ip    = 0xffffffff8131f0f0
flags = 0
ip    = 0xffffffff8131f120
flags = 0
ip    = 0xffffffff8131fb40
flags = 0
ip    = 0xffffffff8131fd00
flags = 0
ip    = 0xffffffff8131fed0
flags = 0
```

Finding __fentry__

vmlinux:

```
<schedule>:
nop
[..]

<yield>:
nop
[..]

<preempt_schedule_common>:
nop
[..]

<_cond_resched>:
nop
[..]

<schedule_idle>:
nop
[..]
```

<ftrace_pages>

```
ip    = 0xffffffff81a14910
flags = 0x40000001
ip    = 0xffffffff81a149b0
flags = 0
ip    = 0xffffffff81a14c00
flags = 0
ip    = 0xffffffff81a14c20
flags = 0
ip    = 0xffffffff81a14c50
flags = 0
ip    = 0xffffffff8126f7b0
flags = 0x00000001
ip    = 0xffffffff8126f8f0
flags = 0
ip    = 0xffffffff8126fcc0
flags = 0
ip    = 0xffffffff81270440
flags = 0
ip    = 0xffffffff81270690
flags = 0
ip    = 0xffffffff8131f0f0
flags = 0
ip    = 0xffffffff8131f120
flags = 0
ip    = 0xffffffff8131fb40
flags = 0
ip    = 0xffffffff8131fd00
flags = 0
ip    = 0xffffffff8131fed0
flags = 0
```

bit 30
count = 1

count = 1

Finding __fentry__

vmlinux:

```
<schedule>:
call ftrace_regs_caller
[..]

<yield>:
nop
[..]

<preempt_schedule_common>:
nop
[..]

<_cond_resched>:
nop
[..]

<schedule_idle>:
call ftrace_caller
[..]
```

<ftrace_pages>

```
ip    = 0xffffffff81a14910
flags = 0xe0000001
ip    = 0xffffffff81a149b0
flags = 0
ip    = 0xffffffff81a14c00
flags = 0
ip    = 0xffffffff81a14c20
flags = 0
ip    = 0xffffffff81a14c50
flags = 0
ip    = 0xffffffff8126f7b0
flags = 0x80000001
ip    = 0xffffffff8126f8f0
flags = 0
ip    = 0xffffffff8126fcc0
flags = 0
ip    = 0xffffffff81270440
flags = 0
ip    = 0xffffffff81270690
flags = 0
ip    = 0xffffffff8131f0f0
flags = 0
ip    = 0xffffffff8131f120
flags = 0
ip    = 0xffffffff8131fb40
flags = 0
ip    = 0xffffffff8131fd00
flags = 0
ip    = 0xffffffff8131fed0
flags = 0
```

bit 29,30,31
count = 1

bit 31
count = 1

Modifying code at runtime!

Not the same as at boot up

SMP boxes need to take extra care

Other CPUs may be executing the code you change

x86 has non uniform instruction (different sizes)

Instructions may cross cache and page boundaries

Modifying code at runtime!

```
<schedule>:
       0f 1f 44 00 00          nop
       53                      push   %rbx
       65 48 8b 1c 25 00 61    mov    %gs:0x16100,%rbx
       01 00
       ffffffff81a1491b: R_X86_64_32S  current_task
       48 8b 43 10             mov    0x10(%rbx),%rax
       48 85 c0                test   %rax,%rax
       74 10                   je     ffffffff81a14938 <schedule+0x28>
       f6 43 24 20             testb  $0x20,0x24(%rbx)
       75 49                   jne    ffffffff81a14977 <schedule+0x67>
       48 83 bb 20 0c 00 00    cmpq   $0x0,0xc20(%rbx)
       00
       74 1f                   je     ffffffff81a14957 <schedule+0x47>
       31 ff                   xor    %edi,%edi
       e8 a1 f8 ff ff          callq  ffffffff81a141e0 <__schedule>
       65 48 8b 04 25 00 61    mov    %gs:0x16100,%rax
       01 00
```

Modifying code at runtime!

```
<schedule>:
       e8 1b d0 1e 00          callq  ffffffff81c01930 <__fentry__>
       53                      push   %rbx
       65 48 8b 1c 25 00 61    mov    %gs:0x16100,%rbx
       01 00
       ffffffff81a1491b: R_X86_64_32S  current_task
       48 8b 43 10             mov    0x10(%rbx),%rax
       48 85 c0                test   %rax,%rax
       74 10                   je     ffffffff81a14938 <schedule+0x28>
       f6 43 24 20             testb  $0x20,0x24(%rbx)
       75 49                   jne    ffffffff81a14977 <schedule+0x67>
       48 83 bb 20 0c 00 00    cmpq   $0x0,0xc20(%rbx)
       00
       74 1f                   je     ffffffff81a14957 <schedule+0x47>
       31 ff                   xor    %edi,%edi
       e8 a1 f8 ff ff          callq  ffffffff81a141e0 <__schedule>
       65 48 8b 04 25 00 61    mov    %gs:0x16100,%rax
       01 00
```

Modifying code at runtime!

Modifying code at runtime!

CPU 0

```
<schedule>:
        e8 1b d0 1e 00
        53
        65 48 8b 1c 25 00 61
        01 00
        48 8b 43 10
        48 85 c0
```

CPU 1

```
<schedule>:
        0f 1f 44 00 00
        53
        65 48 8b 1c 25 00 61
        01 00
        48 8b 43 10
        48 85 c0
```

Modifying code at runtime!

CPU 0

```
<schedule>:
        e8 1b d0 1e 00
        53
        65 48 8b 1c 25 00 61
        01 00
        48 8b 43 10
        48 85 c0
```

CPU 1

```
<schedule>:
        0f 1f d0 1e 00
        53
        65 48 8b 1c 25 00 61
        01 00
        48 8b 43 10
        48 85 c0
```

0f 1f d0 1e 00 ???

```
0f 1f d0 1e 00
```

0f 1f d0 1e 00 ???

BOOM!

CRASH!

General Protection Fault!

REBOOT!

How to go from this!

```
<schedule>:
       0f 1f 44 00 00          nop
       53                      push   %rbx
       65 48 8b 1c 25 00 61    mov    %gs:0x16100,%rbx
       01 00
       ffffffff81a1491b: R_X86_64_32S  current_task
       48 8b 43 10             mov    0x10(%rbx),%rax
       48 85 c0                test   %rax,%rax
       74 10                   je     ffffffff81a14938 <schedule+0x28>
       f6 43 24 20             testb  $0x20,0x24(%rbx)
       75 49                   jne    ffffffff81a14977 <schedule+0x67>
       48 83 bb 20 0c 00 00    cmpq   $0x0,0xc20(%rbx)
       00
       74 1f                   je     ffffffff81a14957 <schedule+0x47>
       31 ff                   xor    %edi,%edi
       e8 a1 f8 ff ff          callq  ffffffff81a141e0 <__schedule>
       65 48 8b 04 25 00 61    mov    %gs:0x16100,%rax
       01 00
```

To this?

```
<schedule>:
        e8 1b d0 1e 00          callq  ffffffff81c01930 <__fentry__>
        53                      push   %rbx
        65 48 8b 1c 25 00 61    mov    %gs:0x16100,%rbx
        01 00
        ffffffff81a1491b: R_X86_64_32S  current_task
        48 8b 43 10             mov    0x10(%rbx),%rax
        48 85 c0                test   %rax,%rax
        74 10                   je     ffffffff81a14938 <schedule+0x28>
        f6 43 24 20             testb  $0x20,0x24(%rbx)
        75 49                   jne    ffffffff81a14977 <schedule+0x67>
        48 83 bb 20 0c 00 00    cmpq   $0x0,0xc20(%rbx)
        00
        74 1f                   je     ffffffff81a14957 <schedule+0x47>
        31 ff                   xor    %edi,%edi
        e8 a1 f8 ff ff          callq  ffffffff81a141e0 <__schedule>
        65 48 8b 04 25 00 61    mov    %gs:0x16100,%rax
        01 00
```

Breakpoints!

Breakpoints!

```
<schedule>:
        0f 1f 44 00 00          nop
        53                      push   %rbx
        65 48 8b 1c 25 00 61    mov    %gs:0x16100,%rbx
        01 00
        ffffffff81a1491b: R_X86_64_32S  current_task
        48 8b 43 10             mov    0x10(%rbx),%rax
        48 85 c0                test   %rax,%rax
        74 10                   je     ffffffff81a14938 <schedule+0x28>
        f6 43 24 20             testb  $0x20,0x24(%rbx)
        75 49                   jne    ffffffff81a14977 <schedule+0x67>
        48 83 bb 20 0c 00 00    cmpq   $0x0,0xc20(%rbx)
        00
        74 1f                   je     ffffffff81a14957 <schedule+0x47>
        31 ff                   xor    %edi,%edi
        e8 a1 f8 ff ff          callq  ffffffff81a141e0 <__schedule>
        65 48 8b 04 25 00 61    mov    %gs:0x16100,%rax
        01 00
```

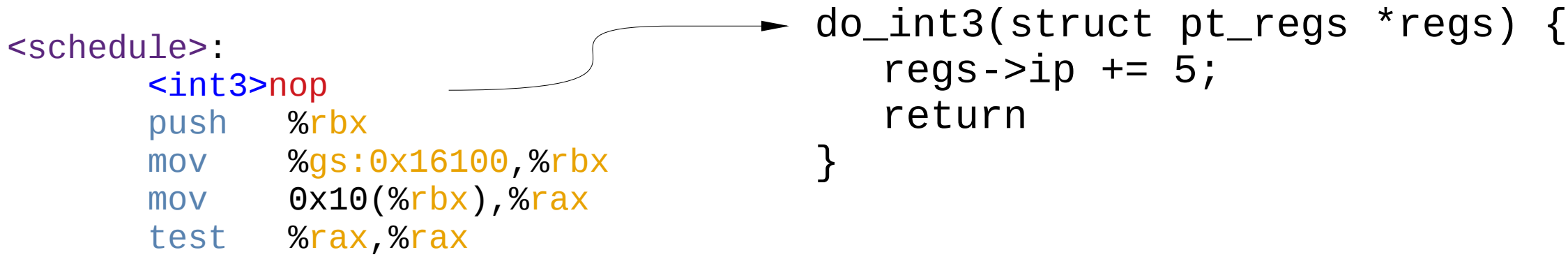
Breakpoints!

```
<schedule>:
        <cc> 1f 44 00 00          <int3>nop
        53                        push   %rbx
        65 48 8b 1c 25 00 61      mov    %gs:0x16100,%rbx
        01 00
        ffffffff81a1491b: R_X86_64_32S  current_task
        48 8b 43 10               mov    0x10(%rbx),%rax
        48 85 c0                  test   %rax,%rax
        74 10                     je     ffffffff81a14938 <schedule+0x28>
        f6 43 24 20               testb  $0x20,0x24(%rbx)
        75 49                     jne    ffffffff81a14977 <schedule+0x67>
        48 83 bb 20 0c 00 00      cmpq   $0x0,0xc20(%rbx)
        00
        74 1f                     je     ffffffff81a14957 <schedule+0x47>
        31 ff                     xor    %edi,%edi
        e8 a1 f8 ff ff            callq  ffffffff81a141e0 <__schedule>
        65 48 8b 04 25 00 61      mov    %gs:0x16100,%rax
        01 00
```

How this works

```
<schedule>:
  ──► <int3>nop
      push   %rbx
      mov    %gs:0x16100,%rbx
      mov    0x10(%rbx),%rax
      test   %rax,%rax
```

How this works



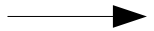
How this works

```
<schedule>:
        <int3>nop
        push   %rbx
        mov    %gs:0x16100,%rbx
        mov    0x10(%rbx),%rax
        test   %rax,%rax
```

```
do_int3(struct pt_regs *regs) {
    regs->ip += 5;
    return
}
```

How this works

```
<schedule>:
        <int3>nop
        push   %rbx
        mov    %gs:0x16100,%rbx
        mov    0x10(%rbx),%rax
        test   %rax,%rax
```



```
do_int3(struct pt_regs *regs) {
   regs->ip += 5;
   return
}
```

How this works

```
<schedule>:
        <int3>nop
        push   %rbx
        mov    %gs:0x16100,%rbx
        mov    0x10(%rbx),%rax
        test   %rax,%rax
```

```
do_int3(struct pt_regs *regs) {
   regs->ip += 5;
   return
}
```

Breakpoints!

```
<schedule>:
        <cc> 1f 44 00 00          <int3>nop
        53                        push   %rbx
        65 48 8b 1c 25 00 61      mov    %gs:0x16100,%rbx
        01 00
        ffffffff81a1491b: R_X86_64_32S  current_task
        48 8b 43 10               mov    0x10(%rbx),%rax
        48 85 c0                  test   %rax,%rax
        74 10                     je     ffffffff81a14938 <schedule+0x28>
        f6 43 24 20               testb  $0x20,0x24(%rbx)
        75 49                     jne    ffffffff81a14977 <schedule+0x67>
        48 83 bb 20 0c 00 00      cmpq   $0x0,0xc20(%rbx)
        00
        74 1f                     je     ffffffff81a14957 <schedule+0x47>
        31 ff                     xor    %edi,%edi
        e8 a1 f8 ff ff            callq  ffffffff81a141e0 <__schedule>
        65 48 8b 04 25 00 61      mov    %gs:0x16100,%rax
        01 00
```

Breakpoints!

```
<schedule>:
        <cc>1b d0 1e 00          <int3>callq  ffffffff81c01930 <__fentry__>
        53                       push   %rbx
        65 48 8b 1c 25 00 61     mov    %gs:0x16100,%rbx
        01 00
        ffffffff81a1491b: R_X86_64_32S  current_task
        48 8b 43 10              mov    0x10(%rbx),%rax
        48 85 c0                 test   %rax,%rax
        74 10                    je     ffffffff81a14938 <schedule+0x28>
        f6 43 24 20              testb  $0x20,0x24(%rbx)
        75 49                    jne    ffffffff81a14977 <schedule+0x67>
        48 83 bb 20 0c 00 00     cmpq   $0x0,0xc20(%rbx)
        00
        74 1f                    je     ffffffff81a14957 <schedule+0x47>
        31 ff                    xor    %edi,%edi
        e8 a1 f8 ff ff           callq  ffffffff81a141e0 <__schedule>
        65 48 8b 04 25 00 61     mov    %gs:0x16100,%rax
        01 00
```

Breakpoints!

```
<schedule>:
        e8 1b d0 1e 00          callq  ffffffff81c01930 <__fentry__>
        53                      push   %rbx
        65 48 8b 1c 25 00 61    mov    %gs:0x16100,%rbx
        01 00
        ffffffff81a1491b: R_X86_64_32S  current_task
        48 8b 43 10             mov    0x10(%rbx),%rax
        48 85 c0                test   %rax,%rax
        74 10                   je     ffffffff81a14938 <schedule+0x28>
        f6 43 24 20             testb  $0x20,0x24(%rbx)
        75 49                   jne    ffffffff81a14977 <schedule+0x67>
        48 83 bb 20 0c 00 00    cmpq   $0x0,0xc20(%rbx)
        00
        74 1f                   je     ffffffff81a14957 <schedule+0x47>
        31 ff                   xor    %edi,%edi
        e8 a1 f8 ff ff          callq  ffffffff81a141e0 <__schedule>
        65 48 8b 04 25 00 61    mov    %gs:0x16100,%rax
        01 00
```

Registering a callback with ftrace

Call register_ftrace_function()

Takes a ftrace_ops descriptor

Static ftrace_ops (allocated at build time)

- Top level ftrace tracers
  - function
  - function_graph
  - stack tracer
  - latency tracers

Dynamic ftrace_ops (allocated via kmalloc())

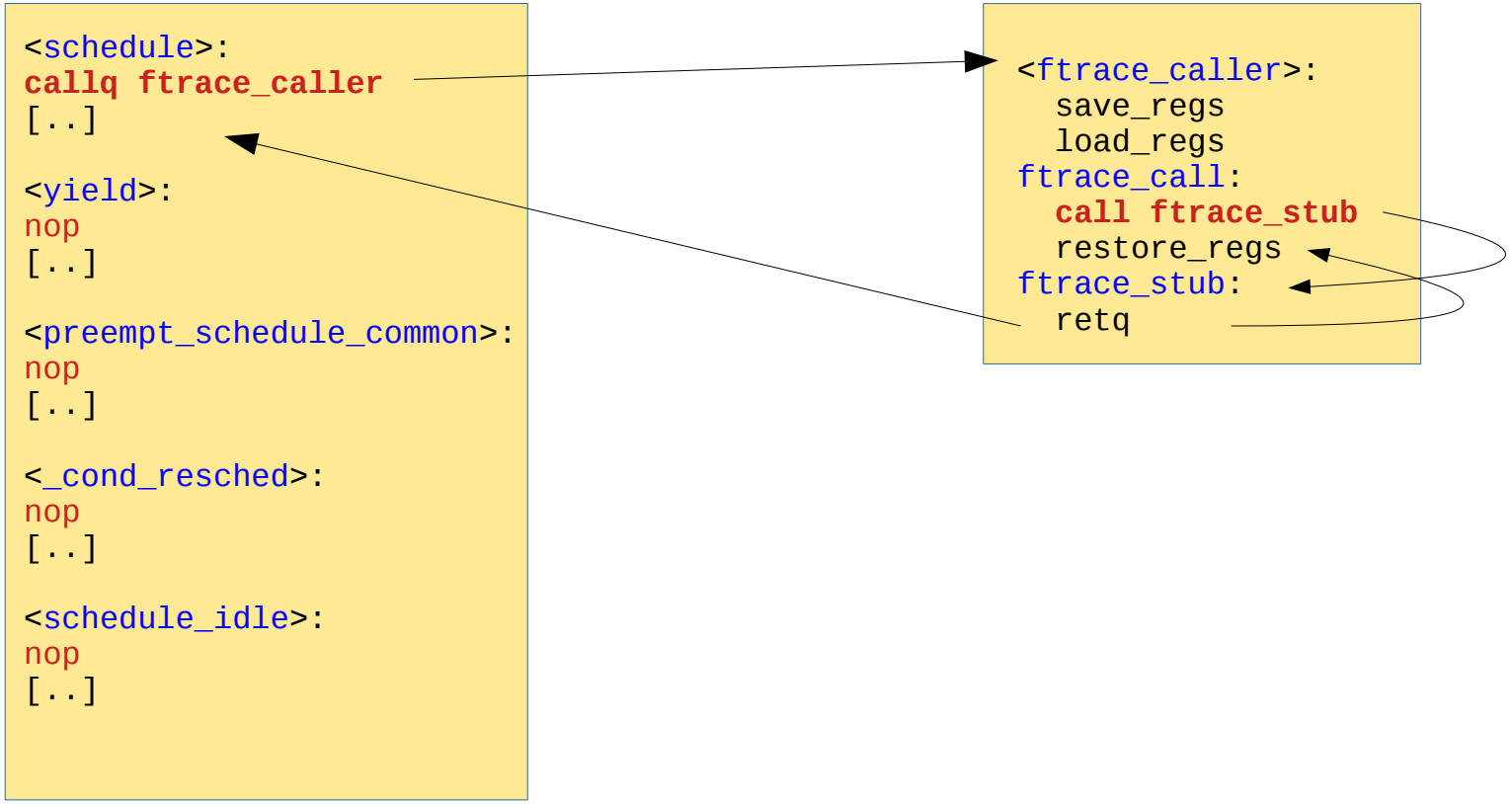
- perf
- kprobes
- ftrace instances (sub buffers)

ftrace_ops structure

```
struct ftrace_ops {
        ftrace_func_t                   func;
        struct ftrace_ops __rcu         *next;
        unsigned long                   flags;
        void                            *private;
        ftrace_func_t                   saved_func;
#ifdef CONFIG_DYNAMIC_FTRACE
        struct ftrace_ops_hash          local_hash;
        struct ftrace_ops_hash          *func_hash;
        struct ftrace_ops_hash          old_hash;
        unsigned long                   trampoline;
        unsigned long                   trampoline_size;
#endif
};
```

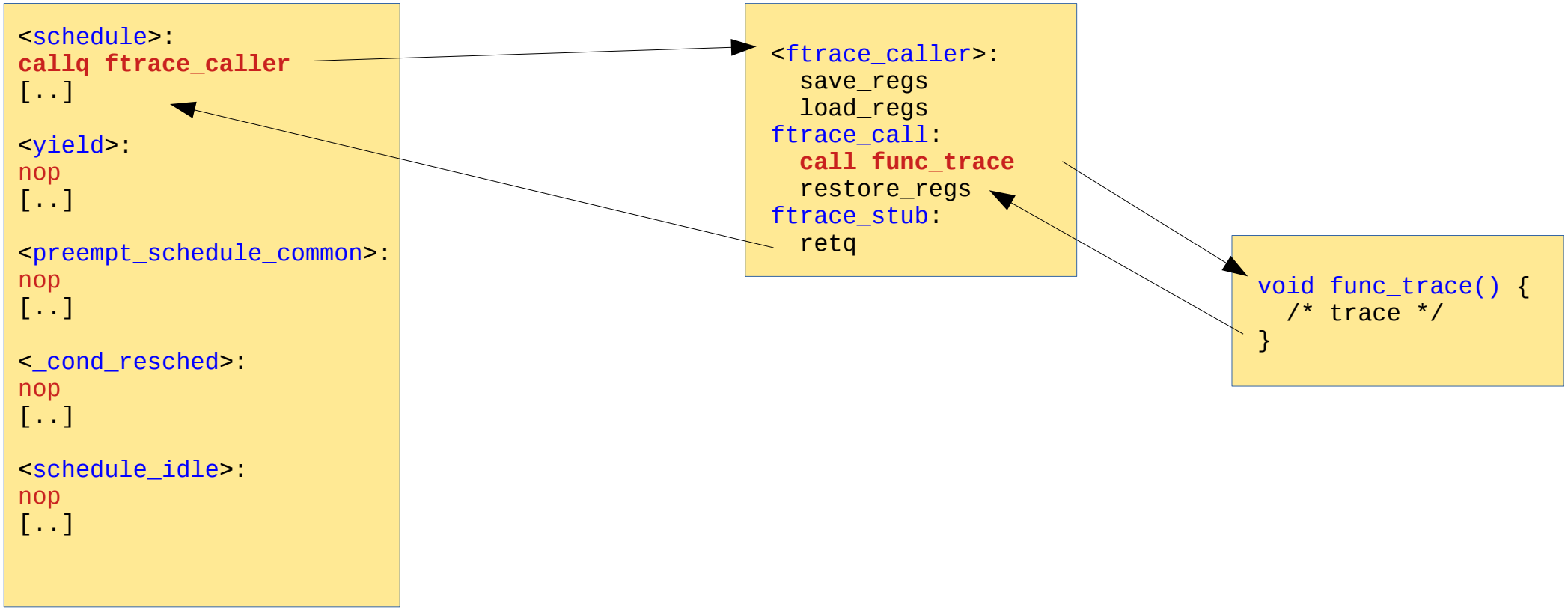
ftrace_caller trampoline

vmlinux:



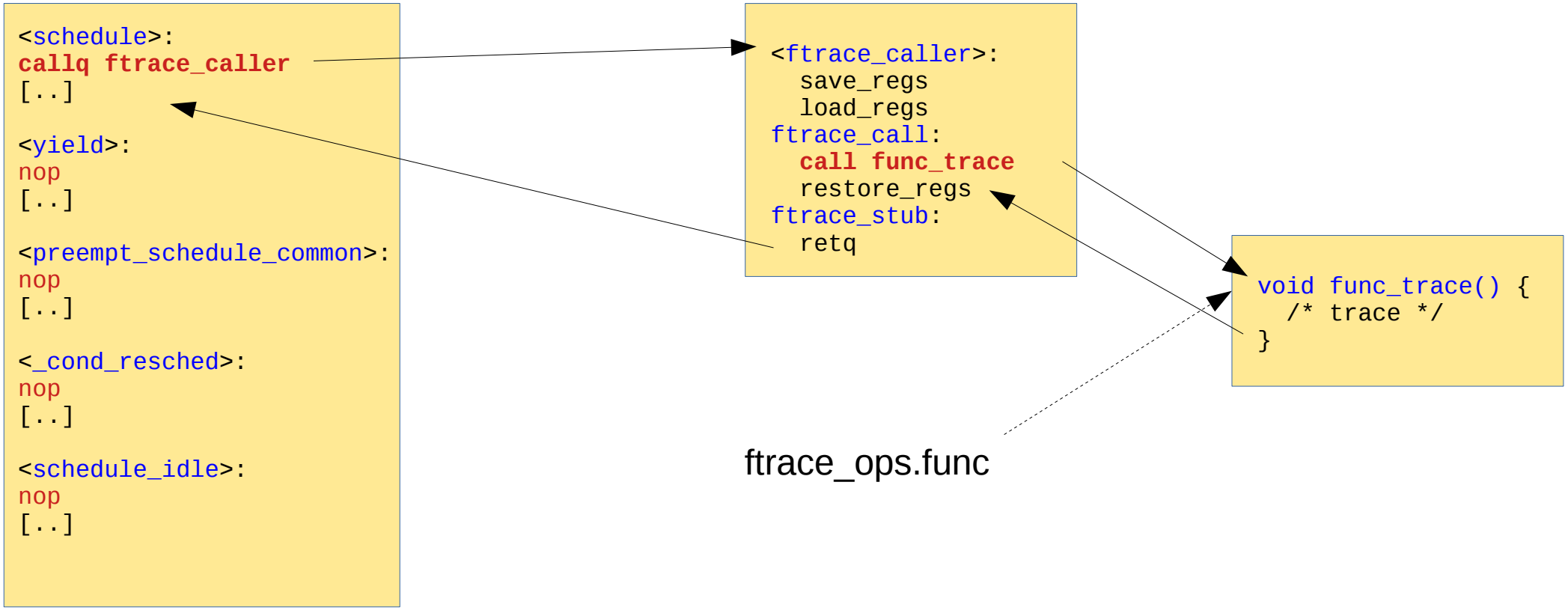
ftrace_caller trampoline

vmlinux:



ftrace_caller trampoline

vmlinux:



Calling more that one callback on a function?

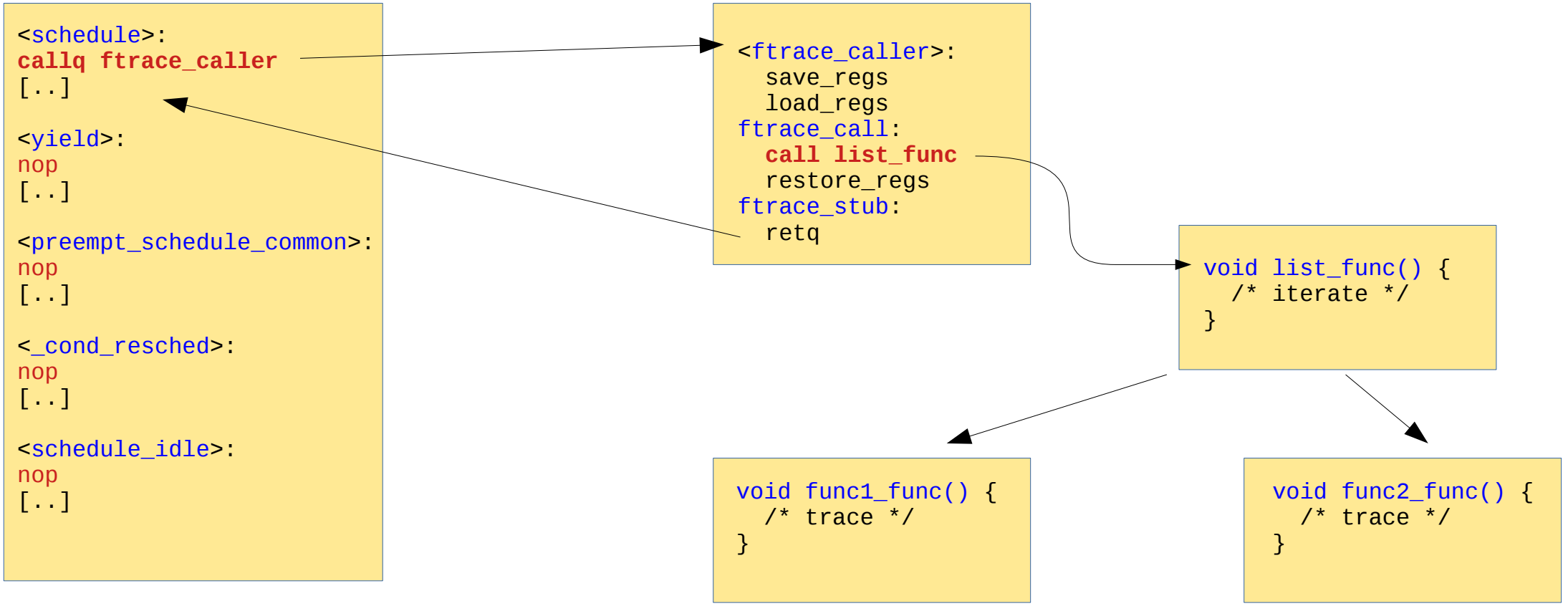
Direct calls to a single function are easy

Handling more than one, requires a list operation

But then all functions being traced will go through a list!

ftrace_caller trampoline

vmlinux:

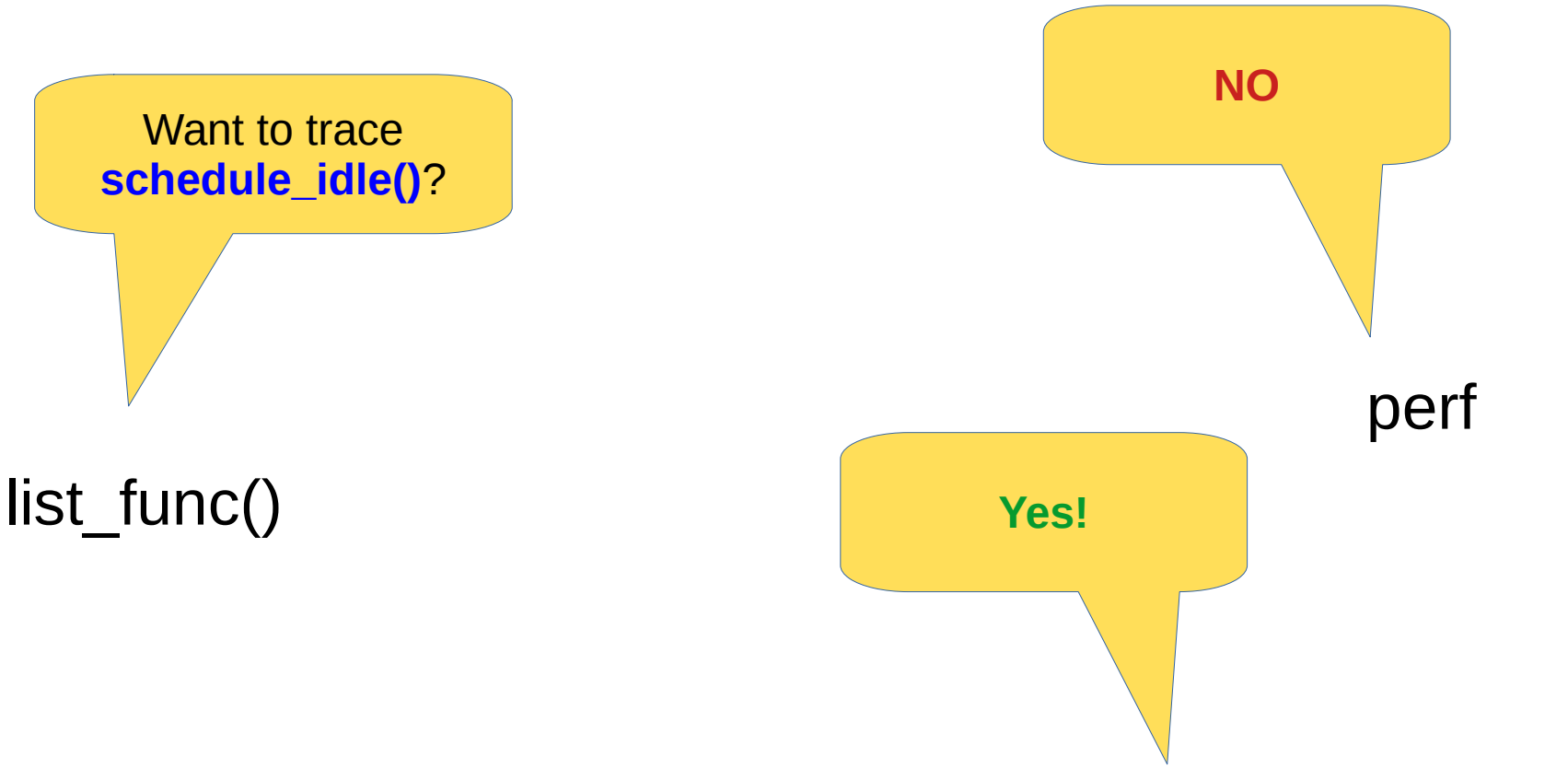


Multiple function callback example

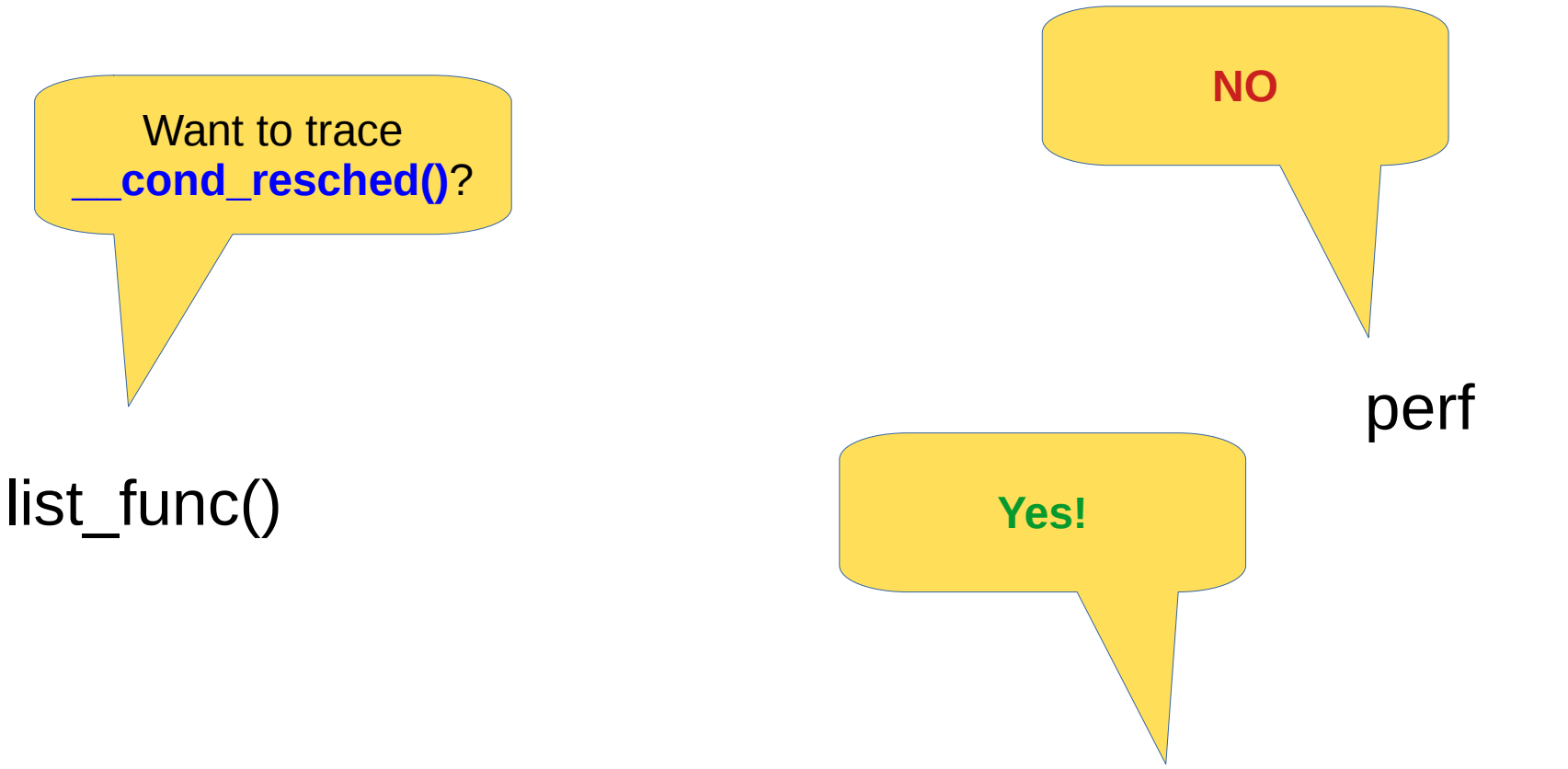
Run function tracer on all functions

Run perf on just the scheduler

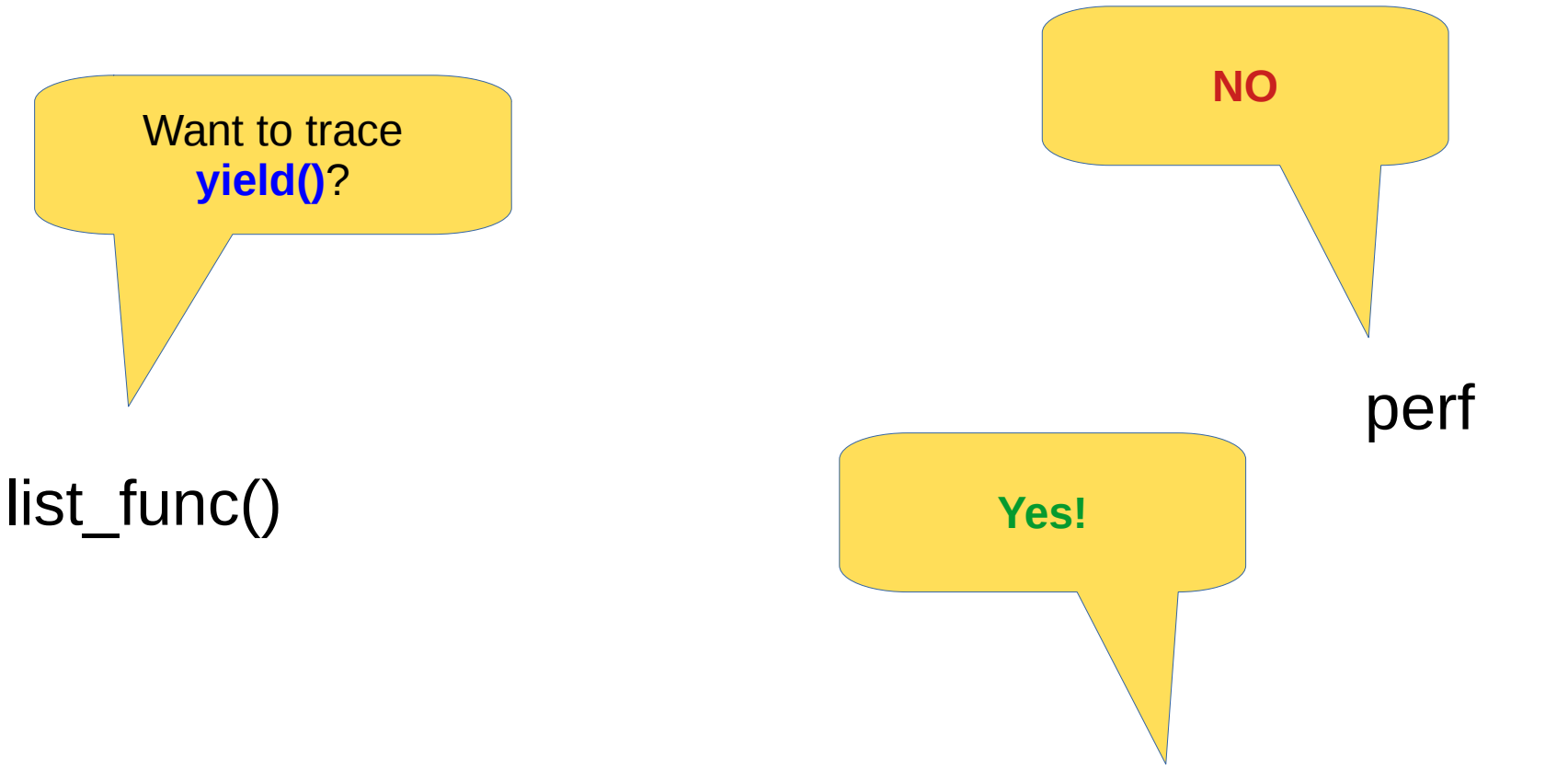
Multiple function callback example



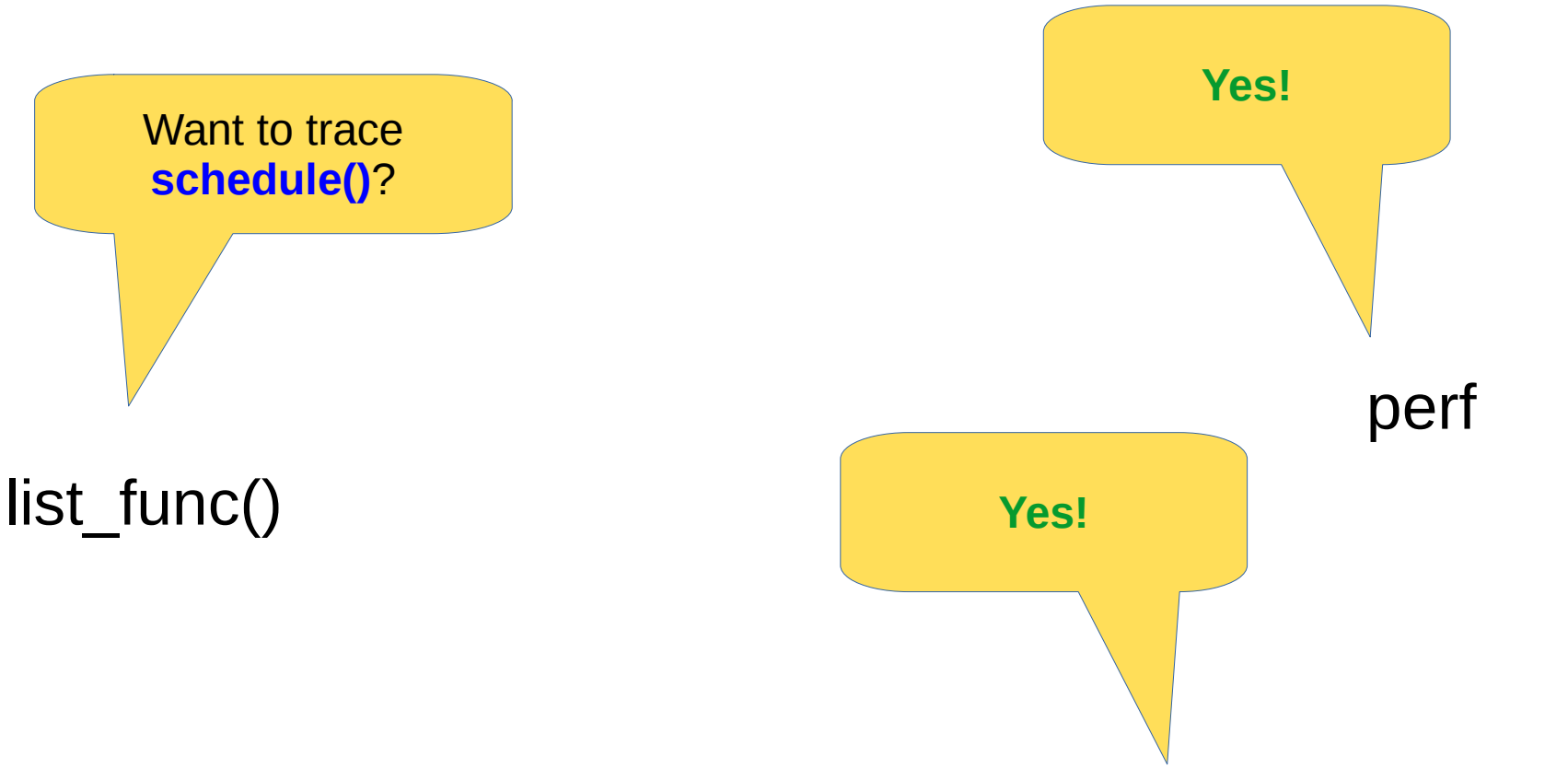
Multiple function callback example



Multiple function callback example

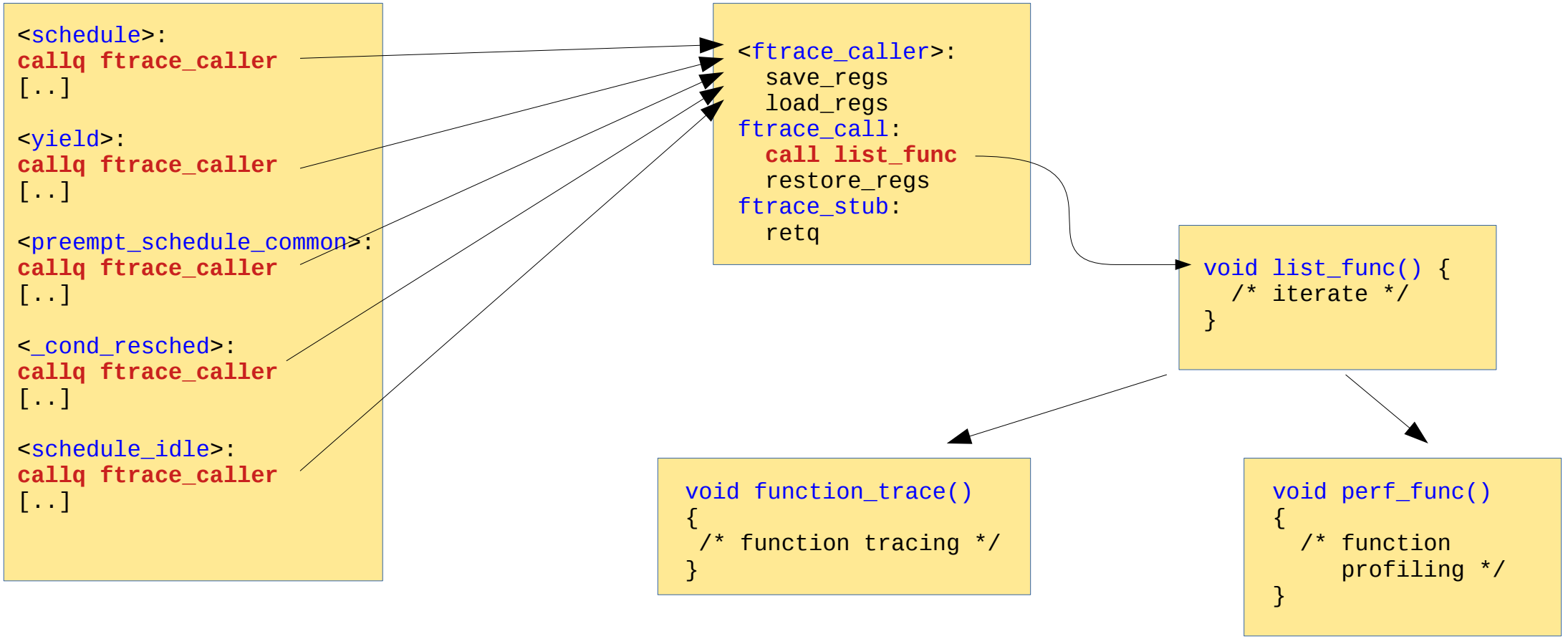


Multiple function callback example



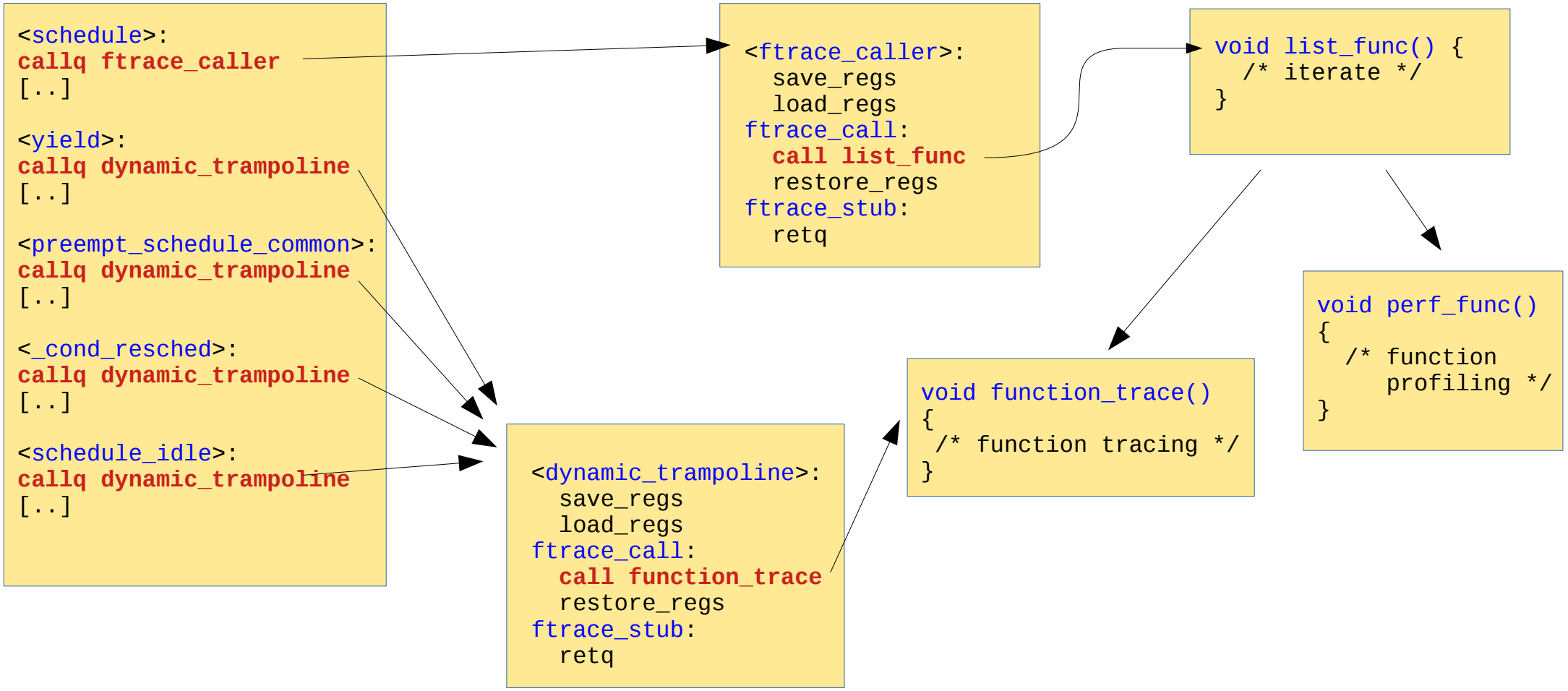
ftrace_caller trampoline

vmlinux:



ftrace_caller trampoline

vmlinux:



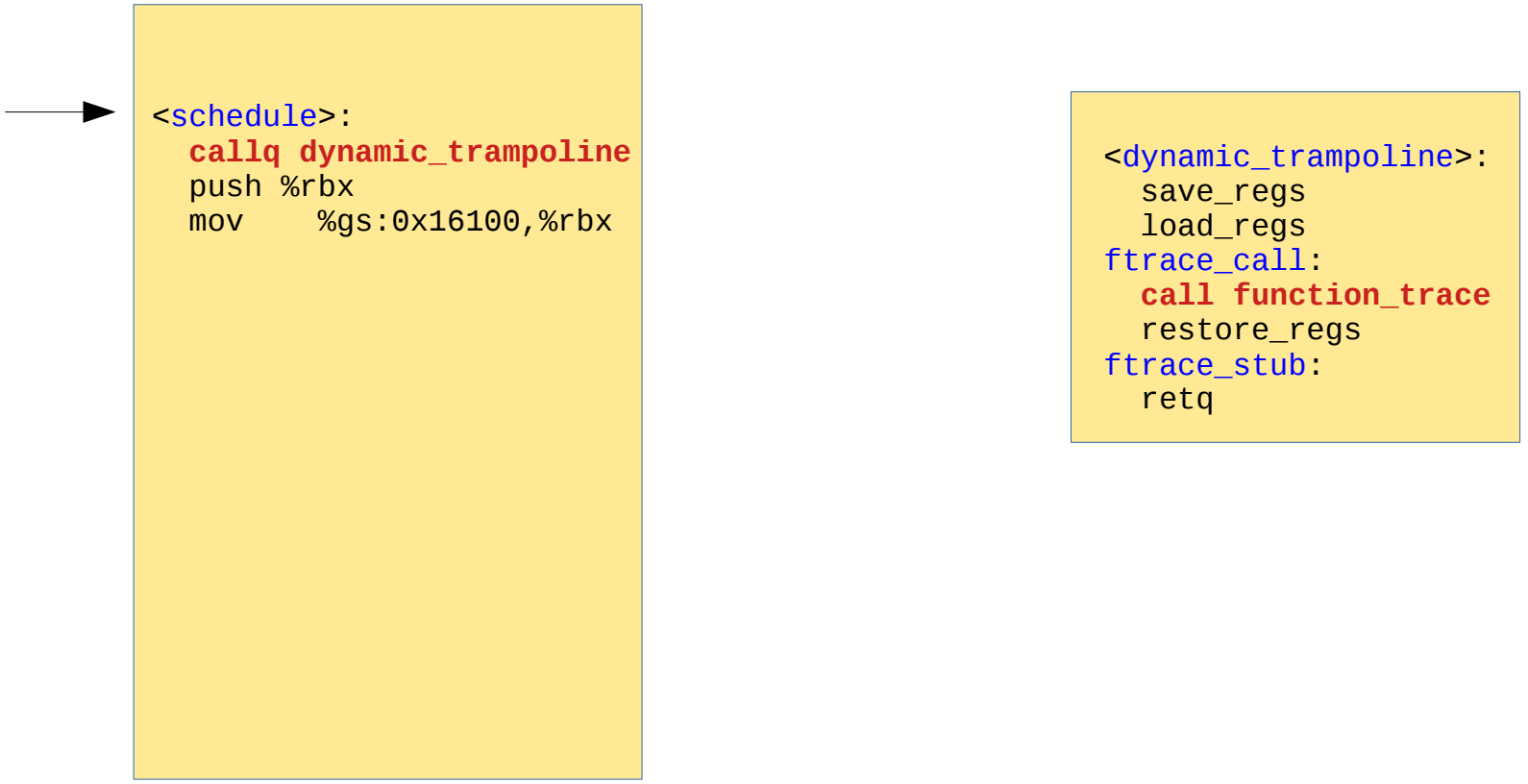
Problems with dynamic trampolines

When can you free them?

How do you know they are still not in use?

Dynamic Trampoline Problem

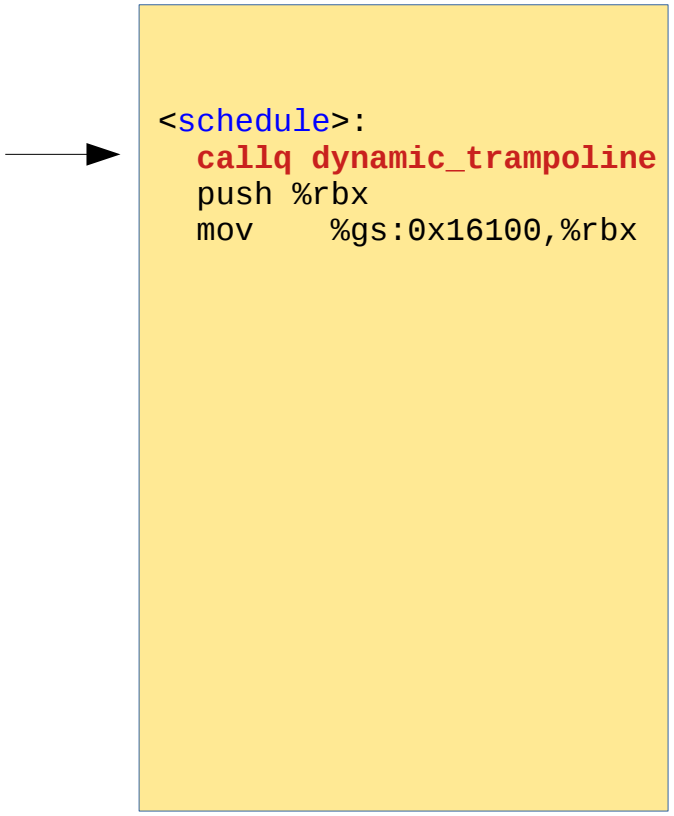
vmlinux:




```
<dynamic_trampoline>:
  save_regs
  load_regs
ftrace_call:
  call function_trace
  restore_regs
ftrace_stub:
  retq
```

Dynamic Trampoline Problem

vmlinux:




```
<dynamic_trampoline>:
  save_regs
  load_regs
ftrace_call:
  call function_trace
  restore_regs
ftrace_stub:
  retq
```

Dynamic Trampoline Problem

vmlinux:

```
<schedule>:
  callq dynamic_trampoline
  push %rbx
  mov    %gs:0x16100,%rbx
```

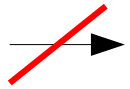
```
<dynamic_trampoline>:
  save_regs
  load_regs
ftrace_call:
  call function_trace
  restore_regs
ftrace_stub:
  retq
```

Dynamic Trampoline Problem

vmlinux:

```
<schedule>:
  callq dynamic_trampoline
  push %rbx
  mov    %gs:0x16100,%rbx
```

Preempted!



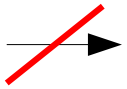
```
<dynamic_trampoline>:
  save_regs
  load_regs
ftrace_call:
  call function_trace
  restore_regs
ftrace_stub:
  retq
```

Dynamic Trampoline Problem

vmlinux:

```
<schedule>:
  nop
  push %rbx
  mov    %gs:0x16100,%rbx
```

Preempted!



```
<dynamic_trampoline>:
  save_regs
  load_regs
ftrace_call:
  call function_trace
  restore_regs
ftrace_stub:
  retq
```

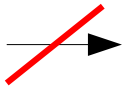
Dynamic Trampoline Problem

vmlinux:

```
<schedule>:
  nop
  push %rbx
  mov    %gs:0x16100,%rbx
```

`kfree(dynamic_trampoline)`

Preempted!



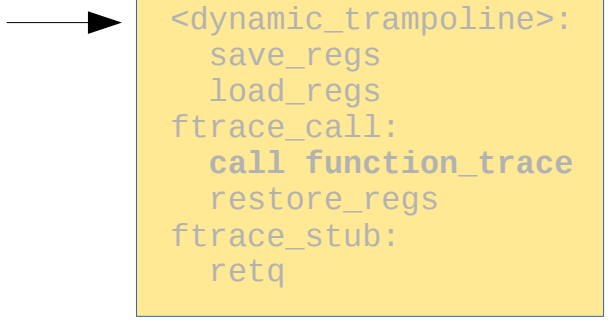
```
<dynamic_trampoline>:
  save_regs
  load_regs
ftrace_call:
  call function_trace
  restore_regs
ftrace_stub:
  retq
```

Dynamic Trampoline Problem

vmlinux:

```
<schedule>:
  nop
  push %rbx
  mov    %gs:0x16100,%rbx
```

Scheduled



Dynamic Trampoline Problem

vmlinux:

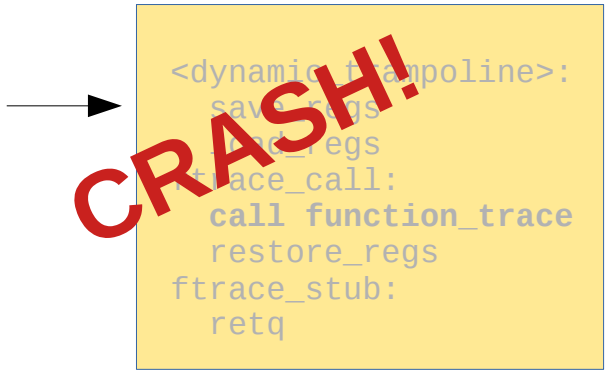
```
<schedule>:
  nop
  push %rbx
  mov    %gs:0x16100,%rbx
```

```
<dynamic_trampoline>:
  save_regs
  load_regs
ftrace_call:
  call function_trace
  restore_regs
ftrace_stub:
  retq
```

Dynamic Trampoline Problem

vmlinux:

```
<schedule>:
  nop
  push %rbx
  mov    %gs:0x16100,%rbx
```



Problems with dynamic trampolines

When can you free them?

How do you know they are still not in use?

Problems with dynamic trampolines

When can you free them?

How do you know they are still not in use?

Use RCU!

call_rcu_tasks()

Added in Linux v3.18

- Commit 8315f42295d2667 by Paul E. McKenney

synchronize_rcu_tasks()

- Waits for all tasks to voluntary schedule
- We do not allow ftrace callbacks to schedule
- The trampoline will not schedule

call_rcu_tasks()

Added in Linux v3.18

- Commit 8315f42295d2667 by Paul E. McKenney

synchronize_rcu_tasks()

- Waits for all tasks to voluntary schedule
- We do not allow ftrace callbacks to schedule
- The trampoline will not schedule

Used by ftrace in v4.12

call_rcu_tasks()

Added in Linux v3.18

- Commit 8315f42295d2667 by Paul E. McKenney

synchronize_rcu_tasks()

- Waits for all tasks to voluntary schedule
- We do not allow ftrace callbacks to schedule
- The trampoline will not schedule

Used by ftrace in v4.12

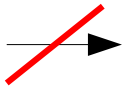
- Yes Steven was lazy
- Added with the threat that Paul was going to remove it

Dynamic Trampoline Solution

vmlinux:

```
<schedule>:
  nop
  push %rbx
  mov    %gs:0x16100,%rbx
```

Preempted!



```
<dynamic_trampoline>:
  save_regs
  load_regs
ftrace_call:
  call function_trace
  restore_regs
ftrace_stub:
  retq
```

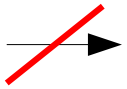
Dynamic Trampoline Solution

vmlinux:

```
<schedule>:
  nop
  push %rbx
  mov    %gs:0x16100,%rbx
```

`call_rcu_tasks(dynamic_trampoline)`

Preempted!



```
<dynamic_trampoline>:
  save_regs
  load_regs
ftrace_call:
  call function_trace
  restore_regs
ftrace_stub:
  retq
```

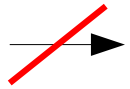
Dynamic Trampoline Solution

vmlinux:

```
<schedule>:
  nop
  push %rbx
  mov    %gs:0x16100,%rbx
```

`call_rcu_tasks(dynamic_trampoline)`

Preempted!



```
<dynamic_trampoline>:
  save_regs
  load_regs
ftrace_call:
  call function_trace
  restore_regs
ftrace_stub:
  retq
```

Wait’s for all tasks to voluntarily schedule

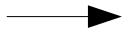
Dynamic Trampoline Solution

vmlinux:

```
<schedule>:
  nop
  push %rbx
  mov    %gs:0x16100,%rbx
```

`call_rcu_tasks(dynamic_trampoline)`

Scheduled



```
<dynamic_trampoline>:
  save_regs
  load_regs
ftrace_call:
  call function_trace
  restore_regs
ftrace_stub:
  retq
```

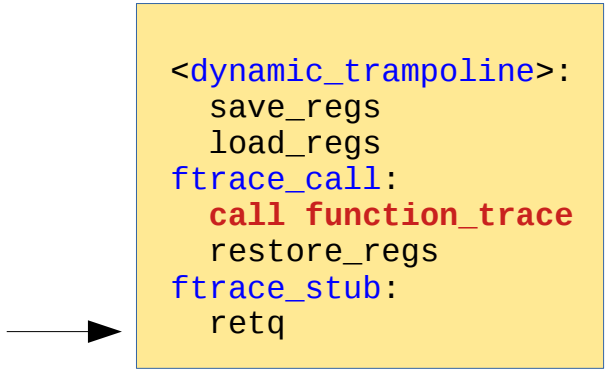
Wait’s for all tasks to voluntarily schedule

Dynamic Trampoline Solution

vmlinux:

```
<schedule>:
  nop
  push %rbx
  mov    %gs:0x16100,%rbx
```

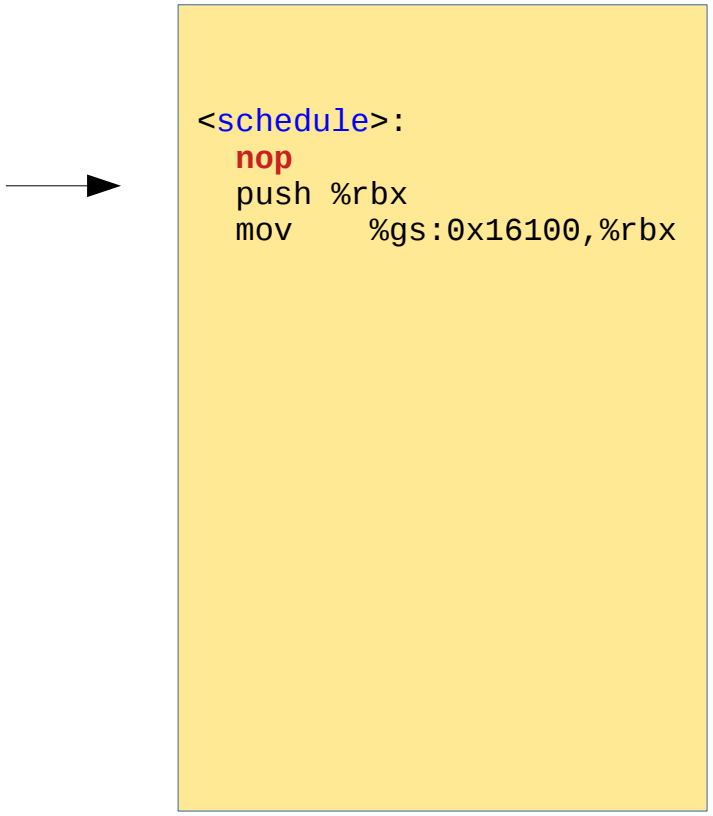
`call_rcu_tasks(dynamic_trampoline)`



Wait's for all tasks to voluntarily schedule

Dynamic Trampoline Solution

vmlinux:



call_rcu_tasks(dynamic_trampoline)

```
<dynamic_trampoline>:
  save_regs
  load_regs
ftrace_call:
  call function_trace
  restore_regs
ftrace_stub:
  retq
```

Wait's for all tasks to voluntarily schedule

Dynamic Trampoline Solution

vmlinux:

```
<schedule>:
  nop
  push %rbx
  mov    %gs:0x16100,%rbx
```

`call_rcu_tasks(dynamic_trampoline)`

```
<dynamic_trampoline>:
  save_regs
  load_regs
ftrace_call:
  call function_trace
  restore_regs
ftrace_stub:
  retq
```

All tasks have scheduled

Dynamic Trampoline Solution

vmlinux:

```
<schedule>:
  nop
  push %rbx
  mov    %gs:0x16100,%rbx
```

kfree(dynamic_trampoline)

```
<dynamic_trampoline>:
  save_regs
  load_regs
ftrace_call:
  call function_trace
  restore_regs
ftrace_stub:
  retq
```

More uses of the function callback code

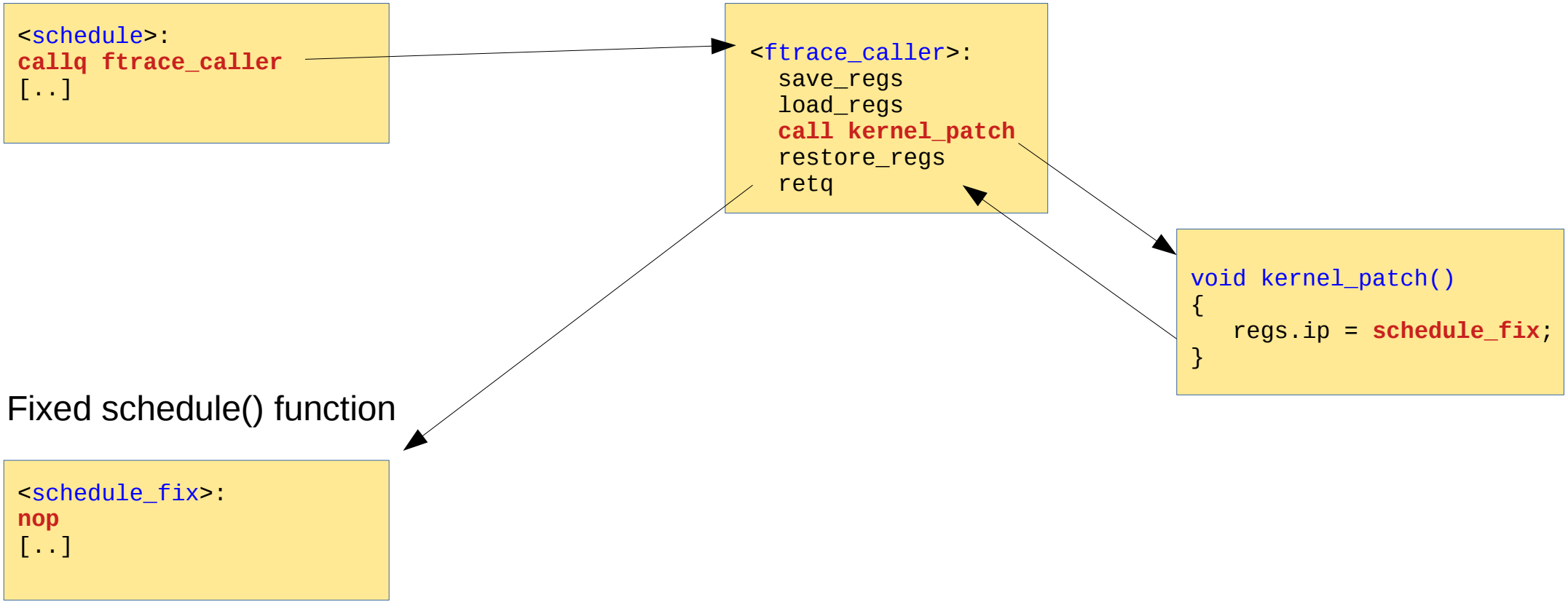
ftrace_regs_caller() gives all registers

A callback can modify any register

- Needs a flag in ftrace_ops to modify the instruction pointer (ip)

Live Kernel Patching!

Buggy schedule() function



Thank You

vmware®