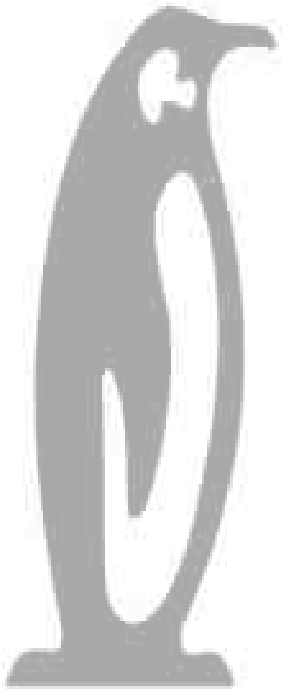


Linux Plumbers Conference

Dublin, Ireland September 12-14, 2022

kprobes/trampolines batch attachment

jiri olsa / isovalent

BATCH ATTACHMENT

speed up attachment of bpf program

to multiple probes

BATCH ATTACHMENT

speed up attachment of bpf program to multiple probes

```
register/unregister(bpf_program, ips, cookies)
```

```
bpf_program(ctx)
{
  cookie = bpf_get_attach_cookie(ctx)
  ip = bpf_get_func_ip(ctx)

  err = get_func_arg(ctx, n, &value)
  err = get_func_ret(ctx, &value)
  cnt = get_func_arg_cnt(ctx)
}
```

BATCH ATTACHMENT

kprobes and bpf trampolines

started for trampolines, took kprobe detour

KPROBE

not multi attach friendly interface

probe anywhere

```
int register_kprobe(struct kprobe *p);

void unregister_kprobe(struct kprobe *p);
```

KPROBE

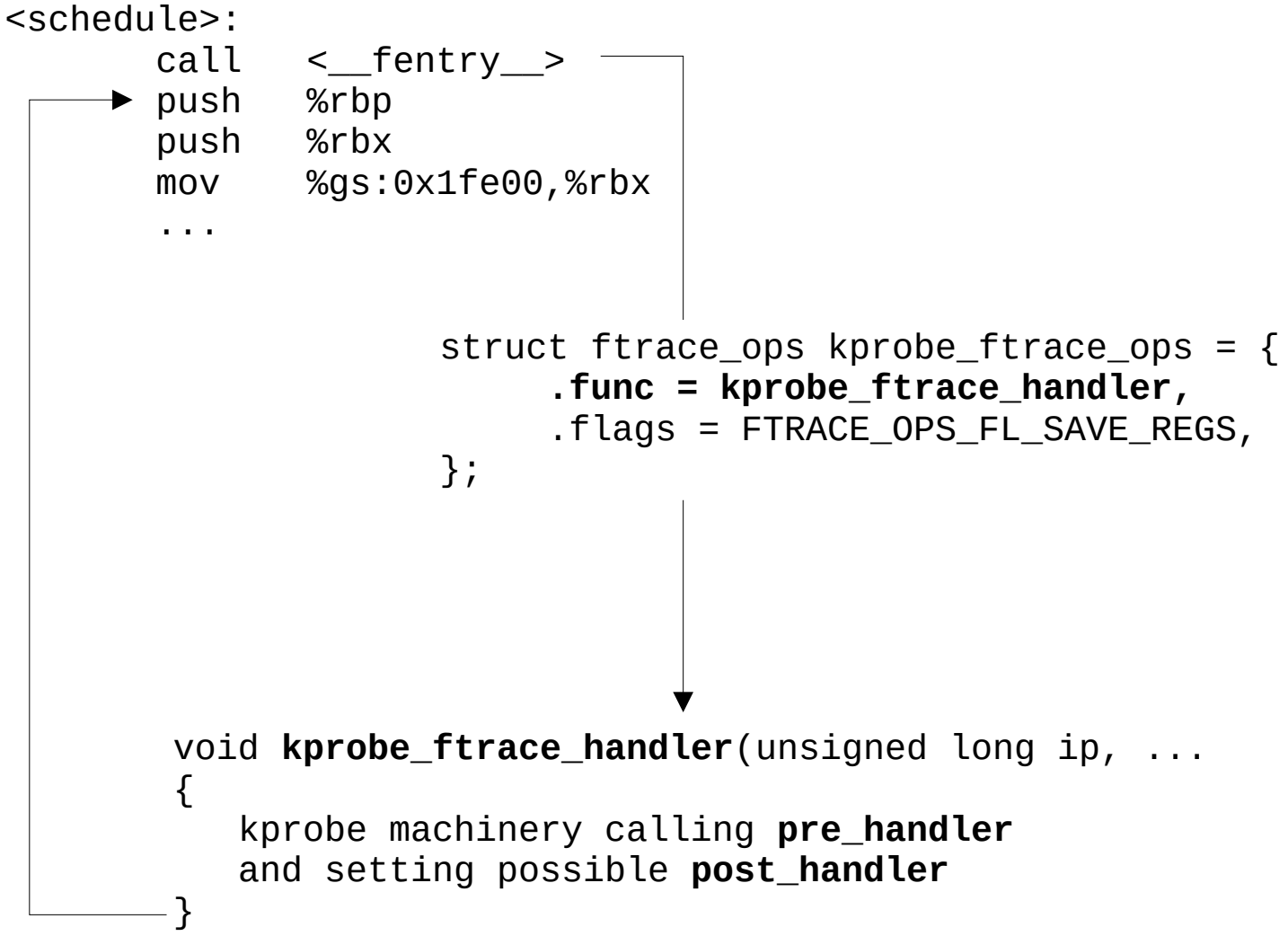
not multi attach friendly interface

probe anywhere

```
int register_kprobe(struct kprobe *p);
void unregister_kprobe(struct kprobe *p);
```

probe on function entry

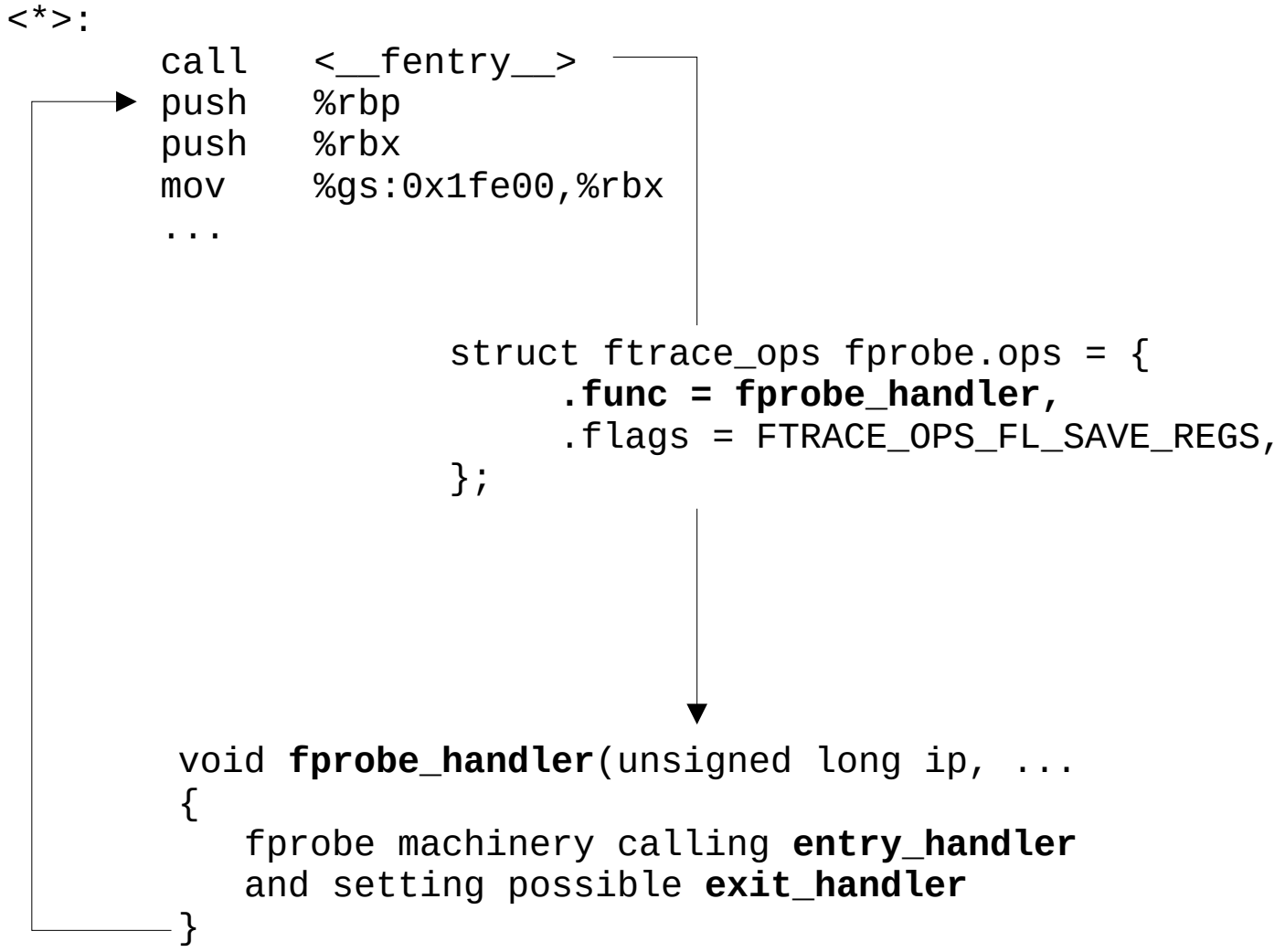
ftrace based kprobes


<schedule>:
call <__fentry__>
push %rbp
push %rbx
mov %gs:0x1fe00,%rbx
...
struct ftrace_ops kprobe_ftrace_ops = {
.func = kprobe_ftrace_handler,
.flags = FTRACE_OPS_FL_SAVE_REGS,
};
void kprobe_ftrace_handler(unsigned long ip, ...
{
kprobe machinery calling pre_handler
and setting possible post_handler
}

FPROBE

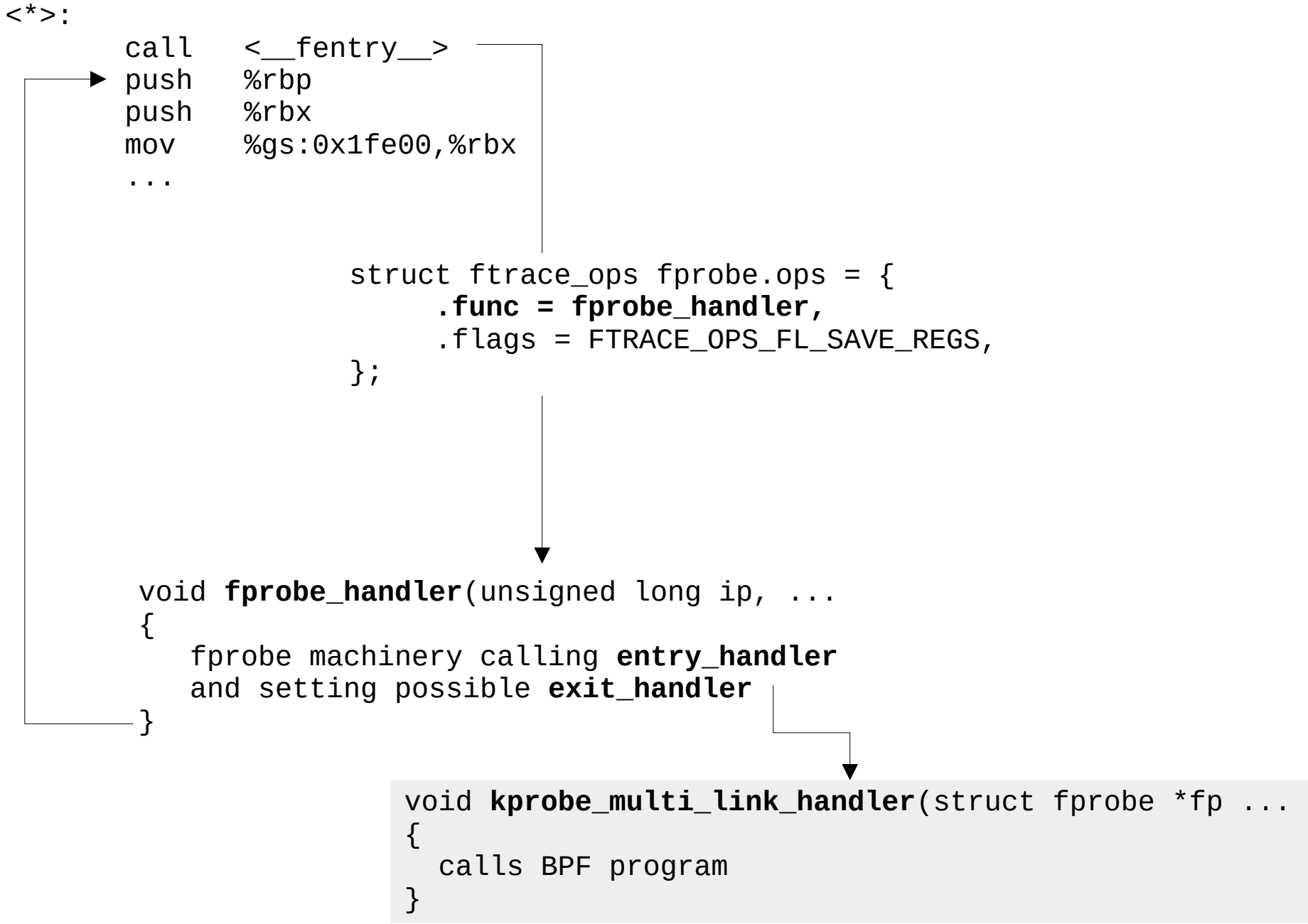
- added by Masami Hiramatsu
- based on ftrace
  - wrapper API for ftrace function tracer
- multi ip attach friendly
  - ftrace does fast attach
  - and multiplexing

```
int register_fprobe(struct fprobe *fp, const char *filter, ...
int register_fprobe_ips(struct fprobe *fp, unsigned long *addrs, ...
int register_fprobe_syms(struct fprobe *fp, const char **syms, ...

int unregister_fprobe(struct fprobe *fp);
```


<*>:
call <__fentry__>
push %rbp
push %rbx
mov %gs:0x1fe00,%rbx
...
struct ftrace_ops fprobe.ops = {
.func = fprobe_handler,
.flags = FTRACE_OPS_FL_SAVE_REGS,
};
void fprobe_handler(unsigned long ip, ...
{
fprobe machinery calling entry_handler
and setting possible exit_handler
}

KPROBE MULTI

- bpf link
- use fprobe as attach layer
- provides user space interface


<*>:
call <__fentry__>
push %rbp
push %rbx
mov %gs:0x1fe00,%rbx
...
struct ftrace_ops fprobe.ops = {
.func = fprobe_handler,
.flags = FTRACE_OPS_FL_SAVE_REGS,
};
void fprobe_handler(unsigned long ip, ...
{
fprobe machinery calling entry_handler
and setting possible exit_handler
}
void kprobe_multi_link_handler(struct fprobe *fp ...
{
calls BPF program
}

```
union bpf_attr {
     struct { /* struct used by BPF_LINK_CREATE command */
          ...
          struct {
               __u32           flags;
               __u32           cnt;
               __aligned_u64   syms;
               __aligned_u64   addrs;
               __aligned_u64   cookies;
          } kprobe_multi;
     };
     ...
};
```

kernel

---

libbpf

```
struct bpf_link_create_opts {
    ...
    struct {
            __u32 flags;
            __u32 cnt;
            const char **syms;
            const unsigned long *addrs;
            const __u64 *cookies;
    } kprobe_multi;
}

link_fd = bpf_link_create(prog_fd, 0, BPF_TRACE_KPROBE_MULTI, opts);
```

```
struct bpf_kprobe_multi_opts {
        /* size of this struct, for forward/backward compatibility */
        size_t sz;
        /* array of function symbols to attach */
        const char **syms;
        /* array of function addresses to attach */
        const unsigned long *addrs;
        /* array of user-provided values fetchable through bpf…
        const __u64 *cookies;
        /* number of elements in syms/addrs/cookies arrays */
        size_t cnt;
        /* create return kprobes */
        bool retprobe;
        size_t :0;
};


link_fd = bpf_program__attach_kprobe_multi_opts(prog, "sched_*", opts);
```

IT’S FAST ;-)

```
# ./test_progs -n 86/8 -v
Loading bpf_testmod.ko...
Successfully loaded bpf_testmod.ko.
test_kprobe_multi_test:PASS:load_kallsyms 0 nsec
test_bench_attach:PASS:get_syms 0 nsec
test_bench_attach:PASS:kprobe_multi_empty__open_and_load 0 nsec
test_bench_attach:PASS:bpf_program__attach_kprobe_multi_opts 0 nsec
test_bench_attach: found 49893 functions
test_bench_attach: attached in   0.837s
test_bench_attach: detached in   0.353s
#86/8    kprobe_multi_test/bench_attach:OK
#86      kprobe_multi_test:OK
Summary: 1/1 PASSED, 0 SKIPPED, 0 FAILED
Successfully unloaded bpf_testmod.ko.
```

USERS

- bpftrace
- tetragon
  - https://github.com/cilium/ebpf/pull/716
  - https://github.com/cilium/tetragon/pull/79
- retsnoop
- probably others.. ;-)

TRAMPOLINE BATCH ATTACHMENT

fun..

TRAMPOLINE BATCH ATTACHMENT

- helpers to get function arguments
- ftrace direct interface to manage multiple ips at once
- mixing trampolines

TRAMPOLINE BATCH ATTACHMENT

- helpers to get function arguments
- ftrace direct interface to manage multiple ips at once
- mixing trampolines

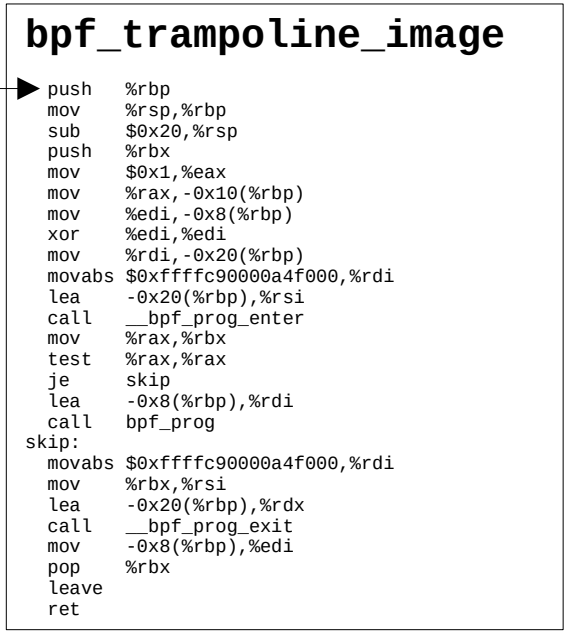
HELPERS

```
long bpf_get_func_arg_cnt(void *ctx);

long bpf_get_func_arg(void *ctx, u32 n, u64 *value);

long bpf_get_func_ret(void *ctx, u64 *value);

u64 bpf_get_func_ip(void *ctx);

u64 bpf_get_attach_cookie(void *ctx);
```

FTRACE DIRECT API

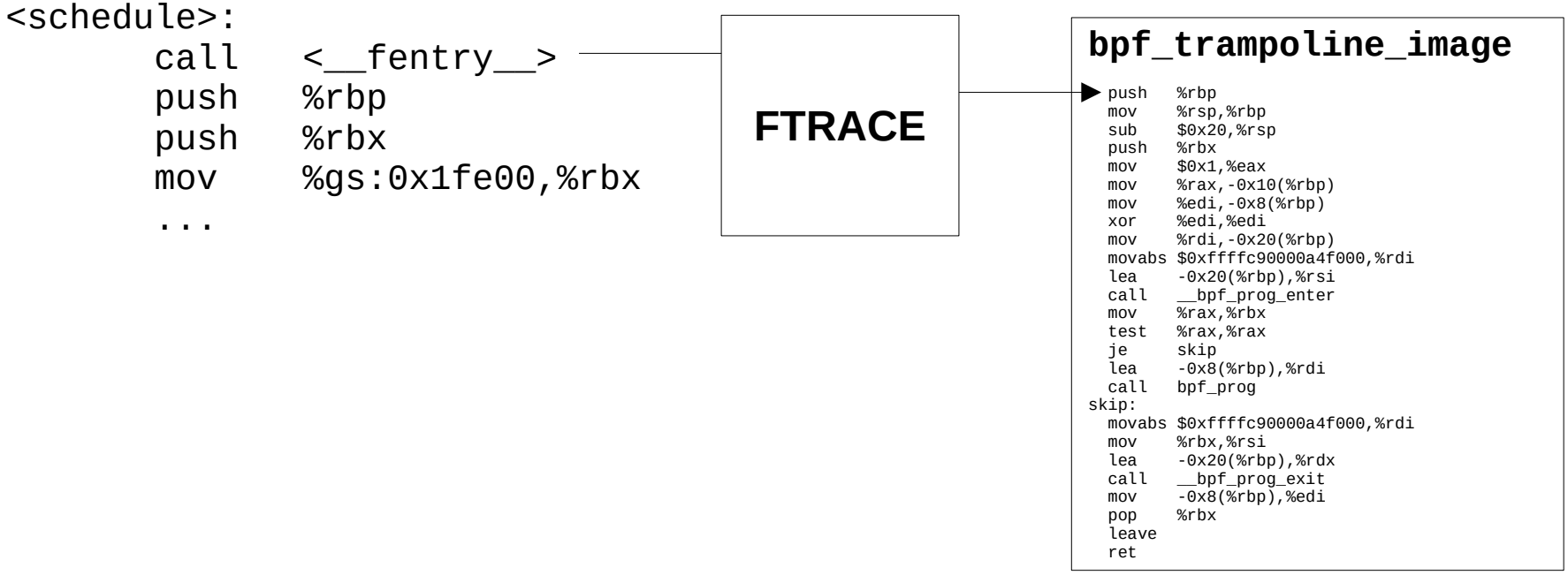
```
int register_ftrace_direct(unsigned long ip, unsigned long addr);
int unregister_ftrace_direct(unsigned long ip, unsigned long addr);
int modify_ftrace_direct(unsigned long ip, unsigned long old_addr,
                         unsigned long new_addr);
```

```
<schedule>:
      call   <__fentry__>
      push   %rbp
      push   %rbx
      mov    %gs:0x1fe00,%rbx
      ...
```



FTRACE DIRECT API

```
int register_ftrace_direct(unsigned long ip, unsigned long addr);
int unregister_ftrace_direct(unsigned long ip, unsigned long addr);
int modify_ftrace_direct(unsigned long ip, unsigned long old_addr,
                         unsigned long new_addr);
```



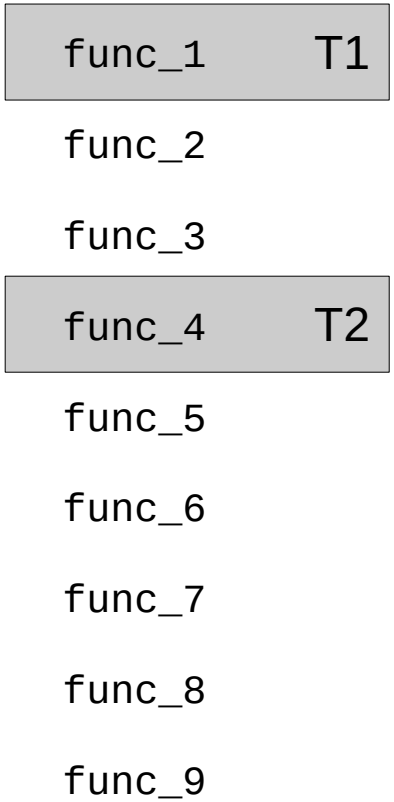
FTRACE DIRECT API

```
int register_ftrace_direct(unsigned long ip, unsigned long addr);
int unregister_ftrace_direct(unsigned long ip, unsigned long addr);
int modify_ftrace_direct(unsigned long ip, unsigned long old_addr,
                         unsigned long new_addr);
```

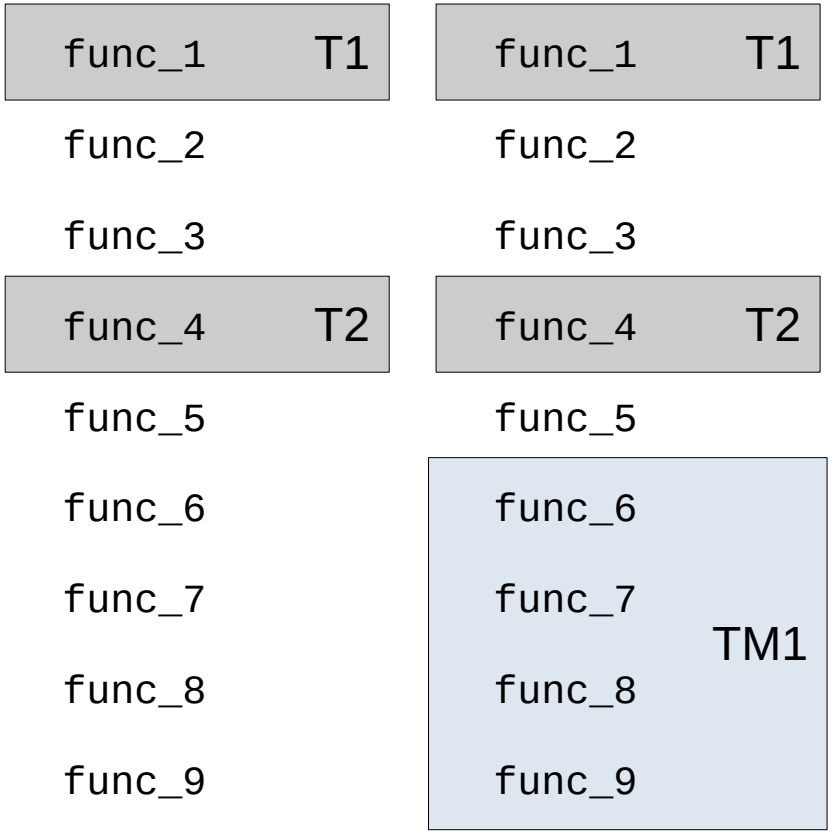
```
int register_ftrace_direct_multi(struct ftrace_ops *ops,
                                 unsigned long addr);
int unregister_ftrace_direct_multi(struct ftrace_ops *ops,
                                   unsigned long addr);
int modify_ftrace_direct_multi(struct ftrace_ops *ops,
                               unsigned long addr);
```

```
int ftrace_set_filter_ip(struct ftrace_ops *ops, unsigned long ip,
                         int remove, int reset);
int ftrace_set_filter_ips(struct ftrace_ops *ops, unsigned long *ips,
                          unsigned int cnt, int remove, int reset);
```

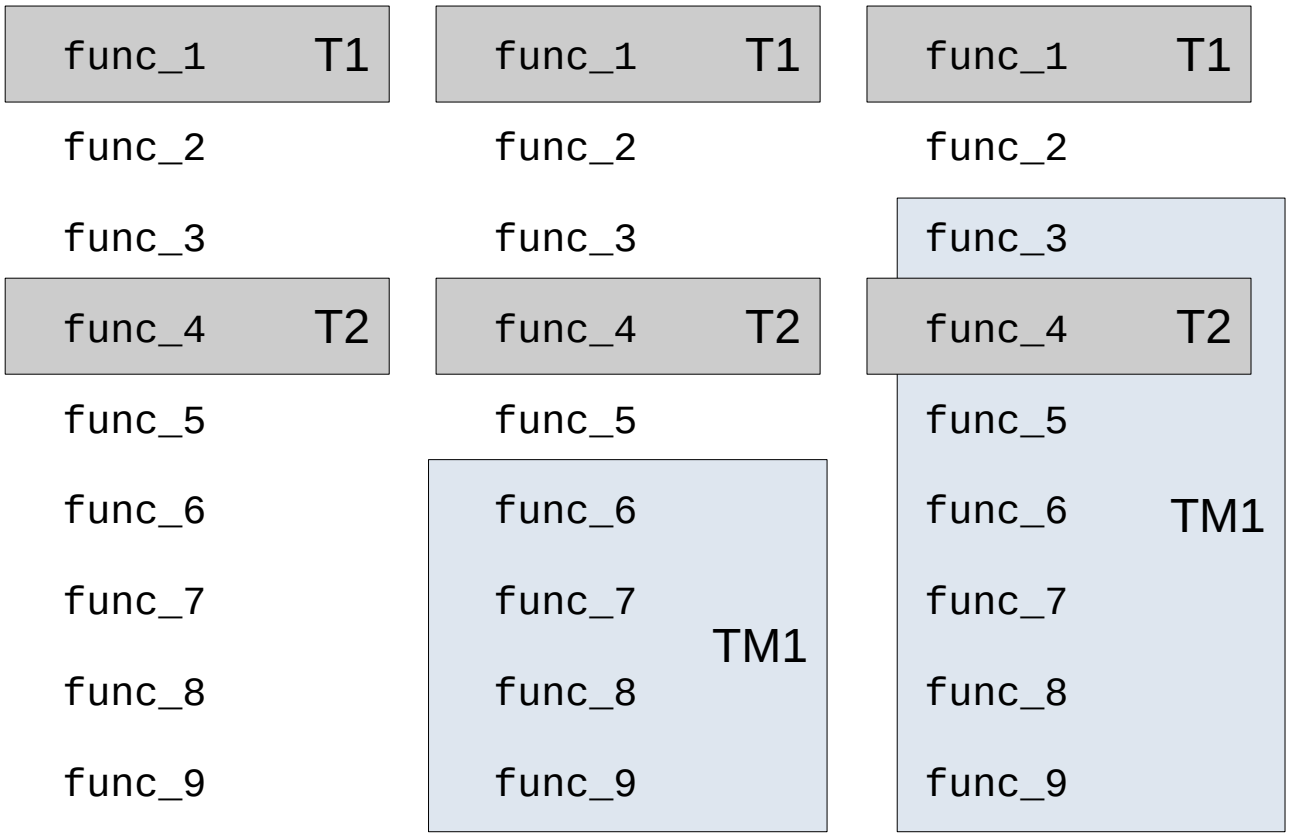
MIXING TRAMPOLINES



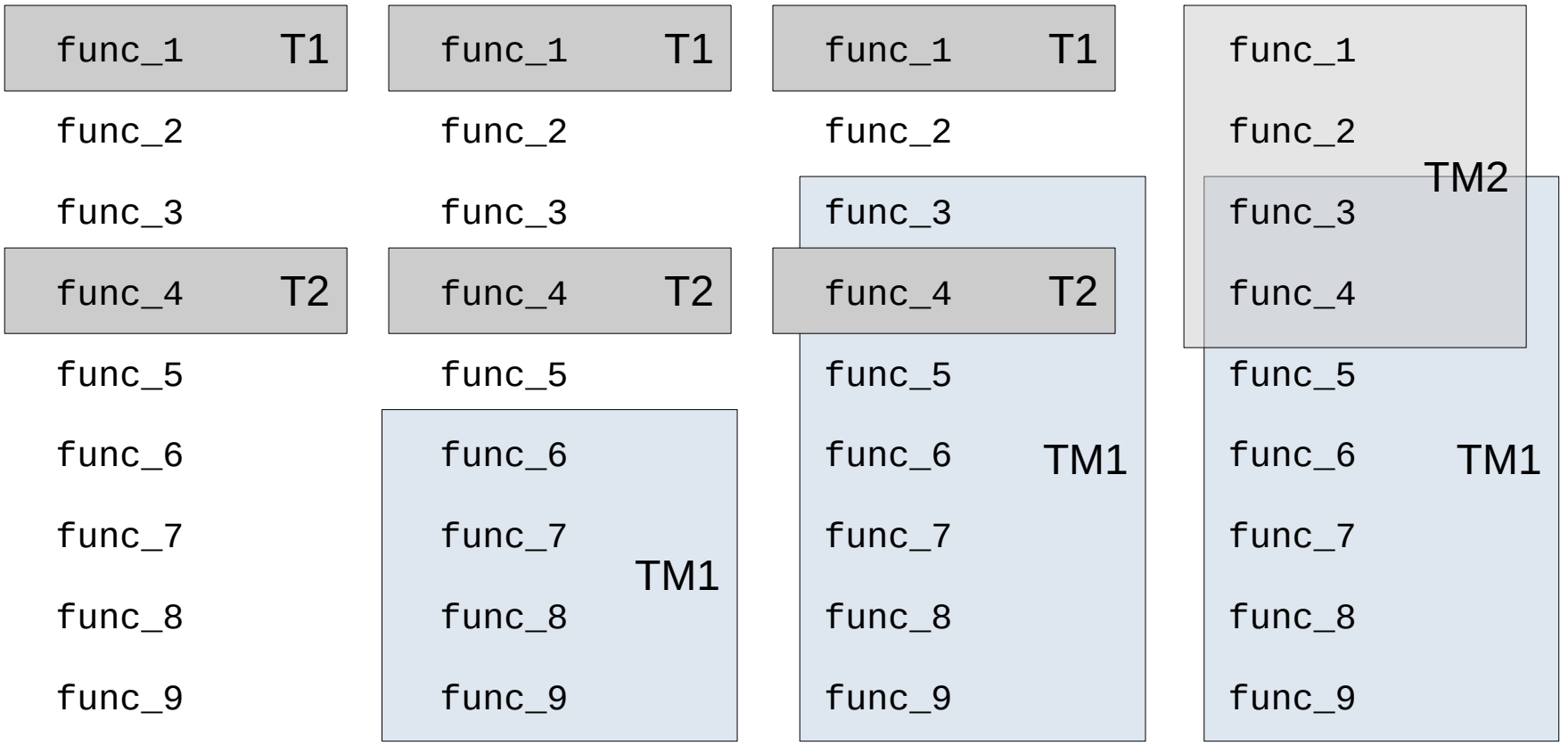
MIXING TRAMPOLINES



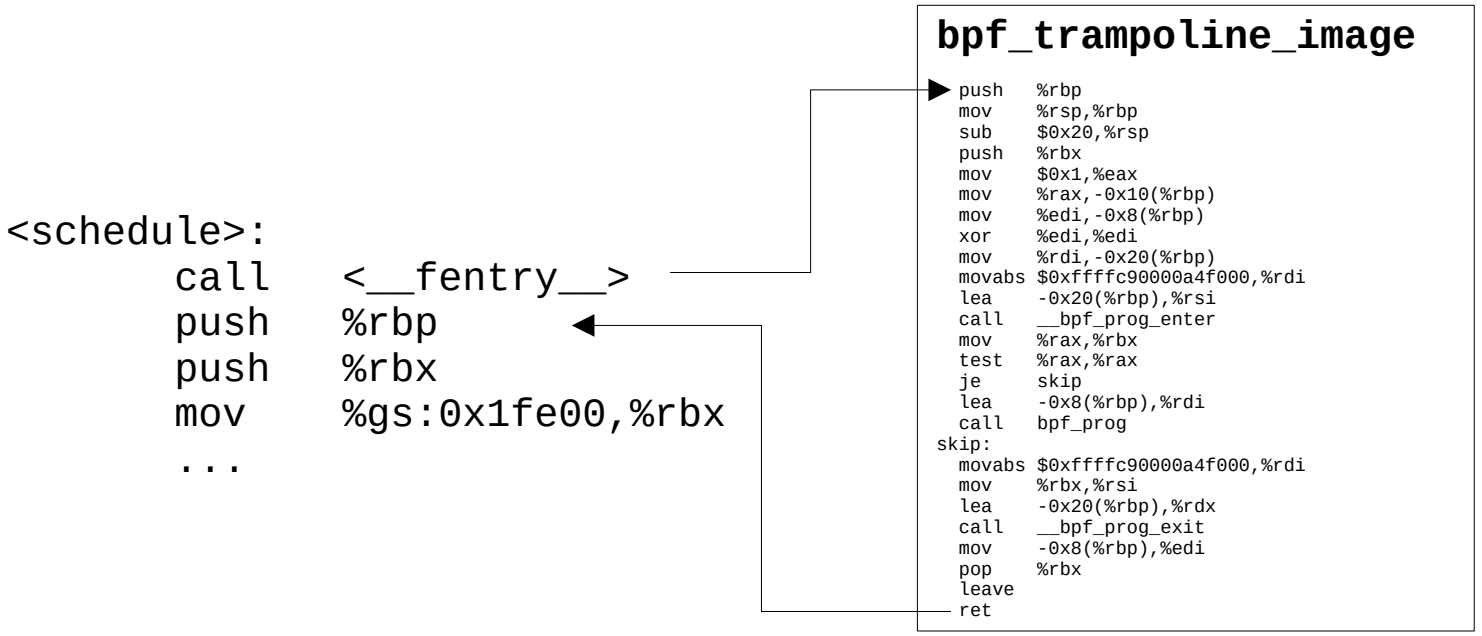
MIXING TRAMPOLINES



MIXING TRAMPOLINES

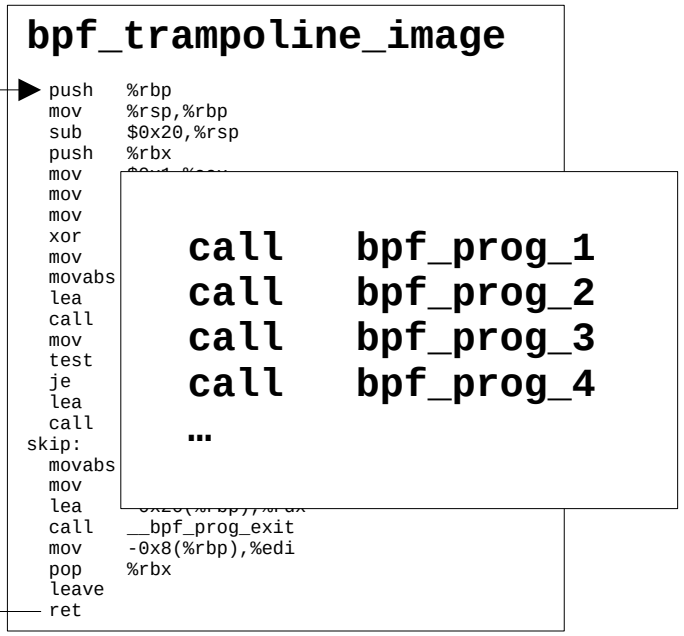


MIXING TRAMPOLINES

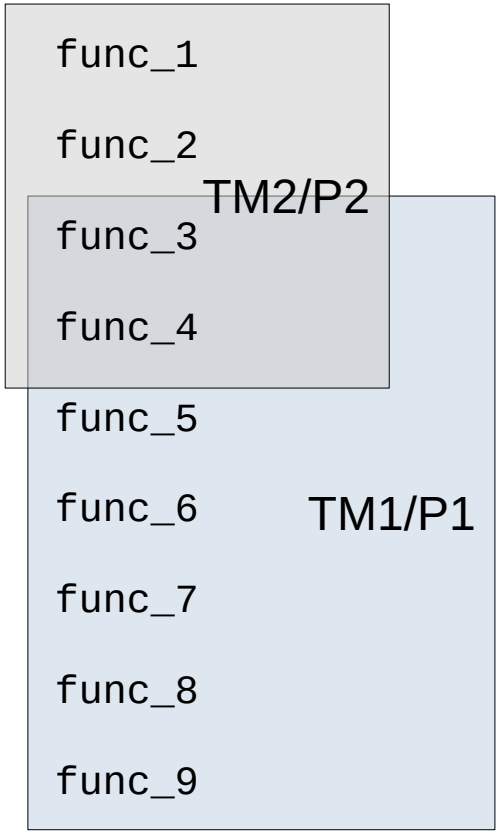


MIXING TRAMPOLINES

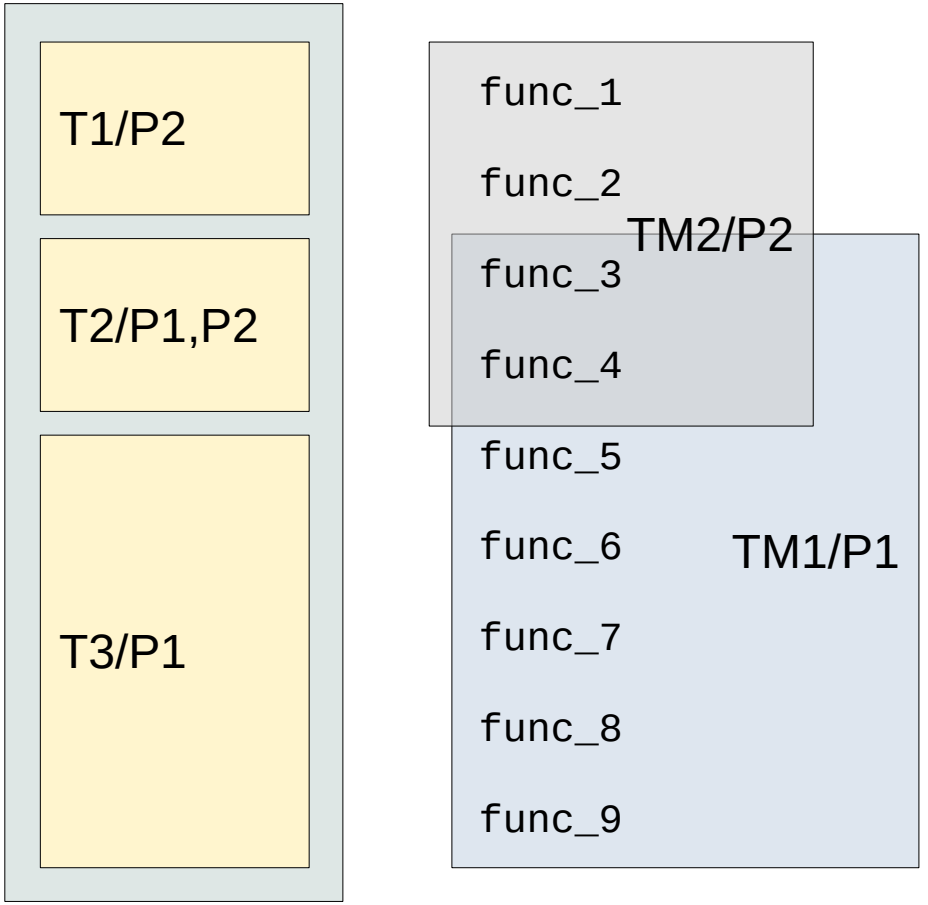
```
<schedule>:
        call    <__fentry__>
        push    %rbp
        push    %rbx
        mov     %gs:0x1fe00,%rbx
        ...
```



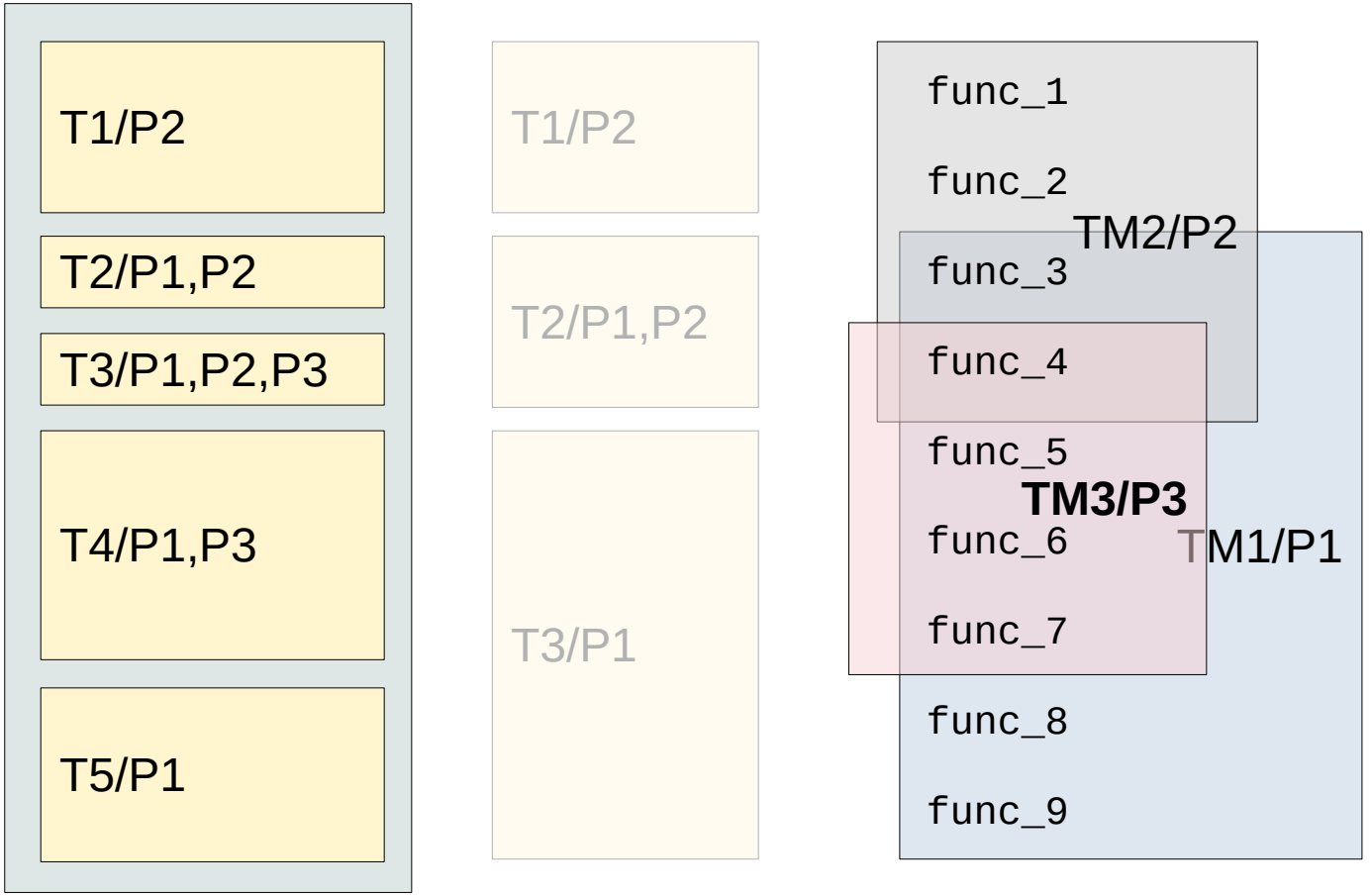
MIXING TRAMPOLINES



MIXING 2 MULTI TRAMPOLINES



MIXING 3 MULTI TRAMPOLINES



PATCHSETs

mixing multi/multi:

https://lore.kernel.org/bpf/20211118112455.475349-1-jolsa@kernel.org/

mixing multi/single:

https://lore.kernel.org/bpf/20220808140626.422731-1-jolsa@kernel.org/

thanks, questions..