# Live Kernel Patching vs BPF

How to play along

# Problem statement

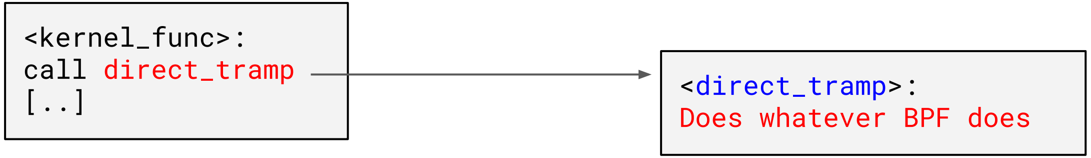
Live kernel patching uses ftrace to “hijack” the function call to call a updated function. In doing so, it sets the IP_MODIFY flag because it modifies where the ftrace trampoline will return to which can only be done by one user attached to a specific function.

BPF uses direct calls which must also set the IP_MODIFY flag as the return of the ftrace trampoline will go to the BPF direct trampoline. This prevents BPF and Live Kernel patching from operating on the same function.

# BPF direct trampoline

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

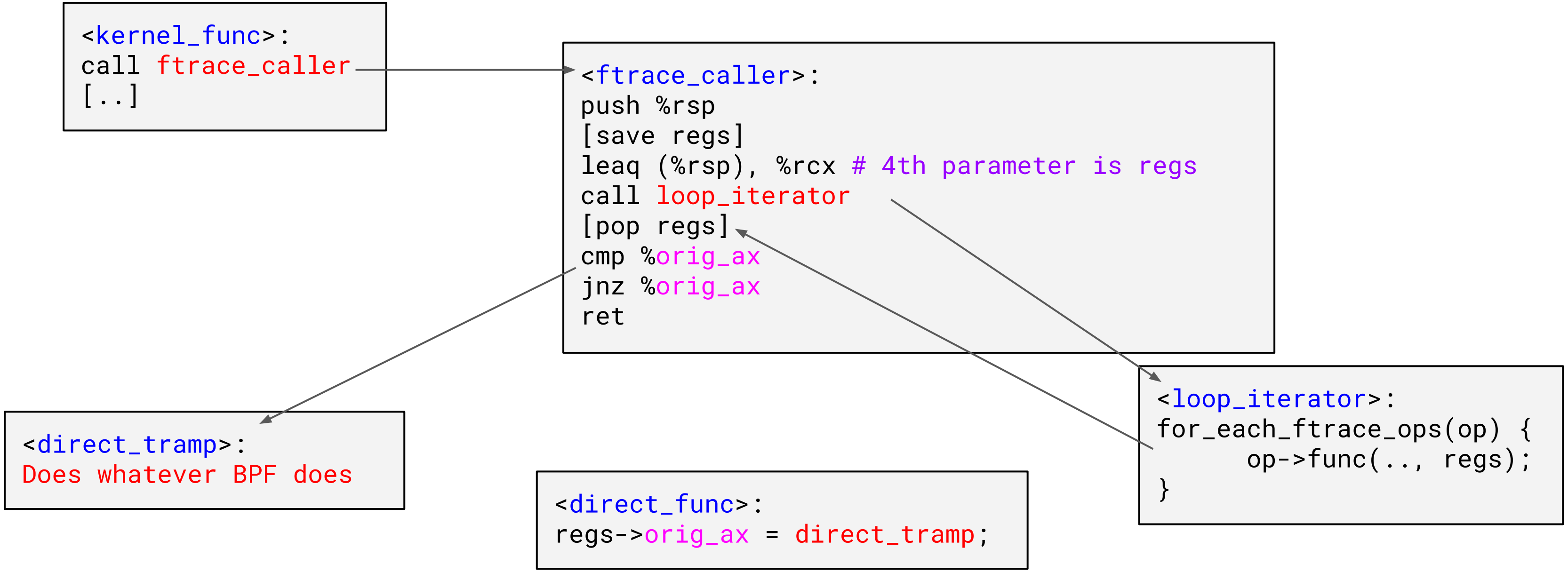
# BPF direct trampoline (solo)



```
<kernel_func>:
call direct_tramp
[..]
```

```
<direct_tramp>:
Does whatever BPF does
```

# BPF direct trampoline (shared)



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

```
<ftrace_caller>:
push %rsp
[save regs]
leaq (%rsp), %rcx # 4th parameter is regs
call loop_iterator
[pop regs]
cmp %orig_ax
jnz %orig_ax
ret
```

```
<loop_iterator>:
for_each_ftrace_ops(op) {
      op->func(.., regs);
}
```

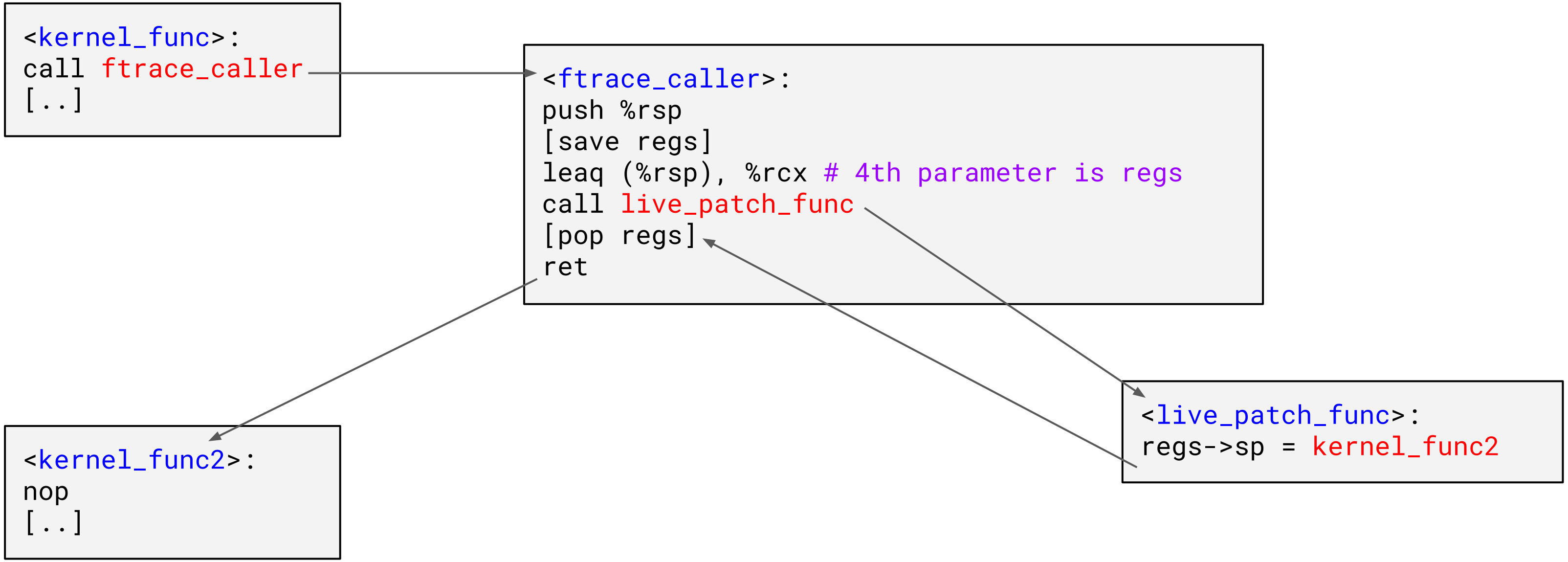
```
<direct_tramp>:
Does whatever BPF does
```

```
<direct_func>:
regs->orig_ax = direct_tramp;
```

# Live kernel patching

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

# Live kernel patching



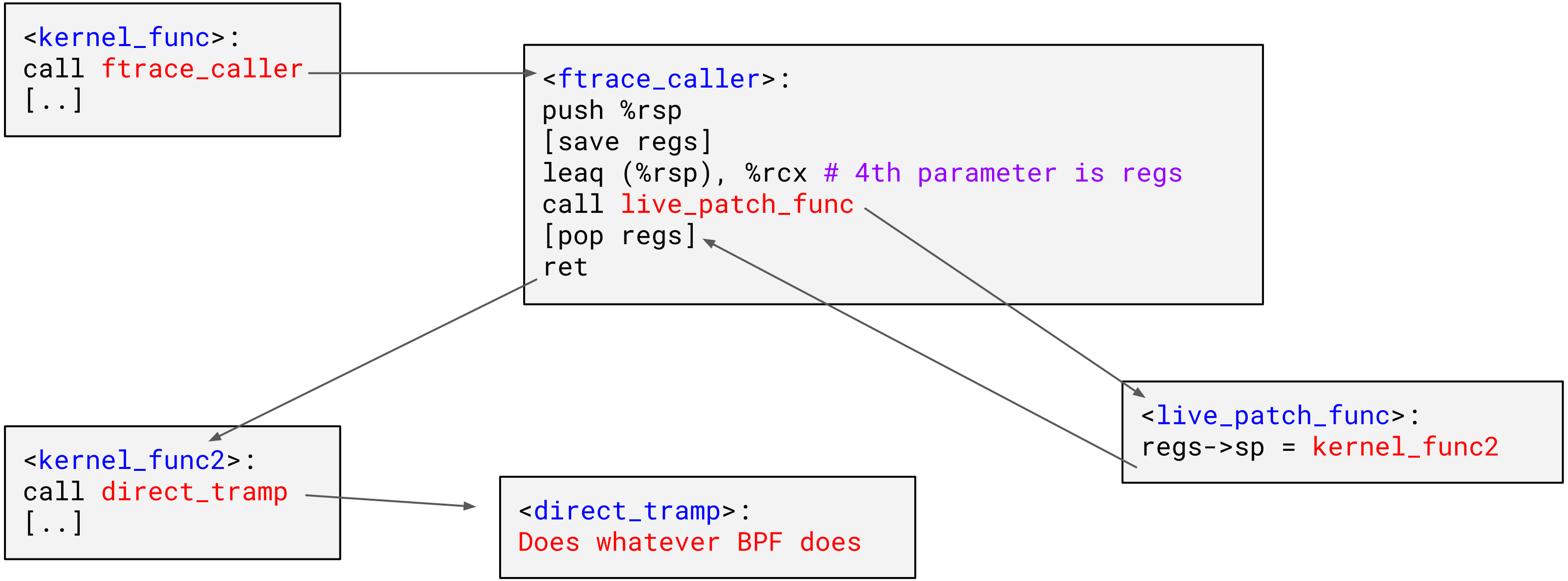
```
<kernel_func>:
call ftrace_caller
[..]
```

```
<ftrace_caller>:
push %rsp
[save regs]
leaq (%rsp), %rcx # 4th parameter is regs
call live_patch_func
[pop regs]
ret
```

```
<live_patch_func>:
regs->sp = kernel_func2
```

```
<kernel_func2>:
nop
[..]
```

# Live kernel patching with BPF direct trampolines



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

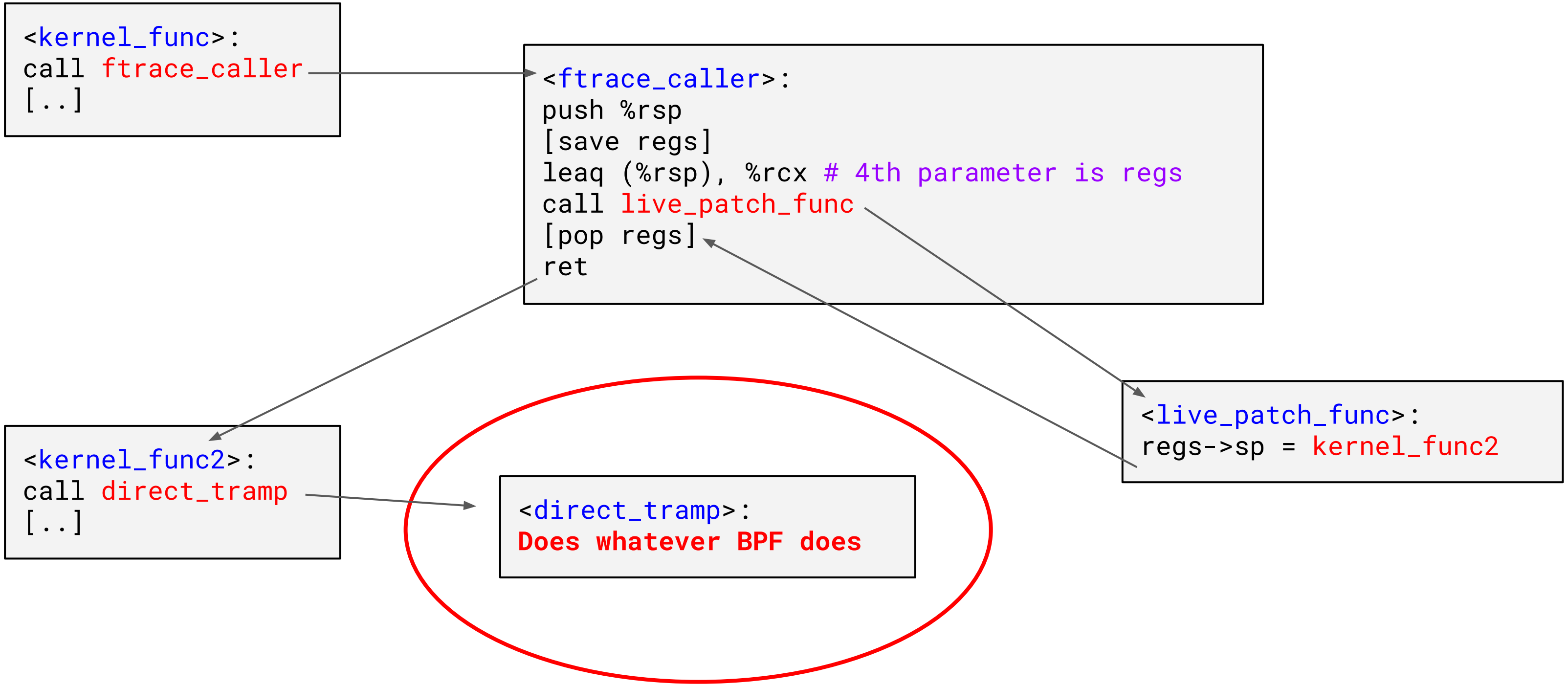
```
<ftrace_caller>:
push %rsp
[save regs]
leaq (%rsp), %rcx # 4th parameter is regs
call live_patch_func
[pop regs]
ret
```

```
<live_patch_func>:
regs->sp = kernel_func2
```

```
<kernel_func2>:
call direct_tramp
[..]
```

```
<direct_tramp>:
Does whatever BPF does
```

# Live kernel patching with BPF direct trampolines



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

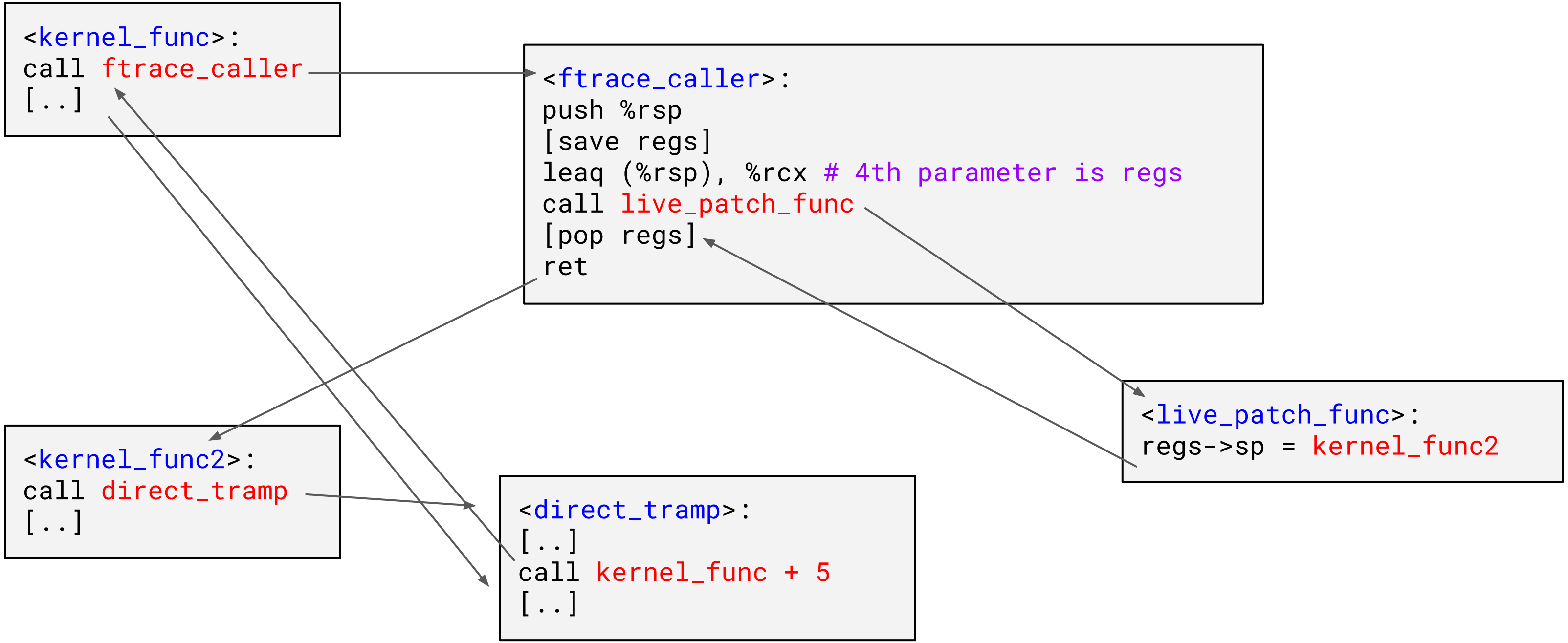
```
<ftrace_caller>:
push %rsp
[save regs]
leaq (%rsp), %rcx # 4th parameter is regs
call live_patch_func
[pop regs]
ret
```

```
<live_patch_func>:
regs->sp = kernel_func2
```

```
<kernel_func2>:
call direct_tramp
[..]
```

```
<direct_tramp>:
Does whatever BPF does
```

# Live kernel patching with BPF direct trampolines



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

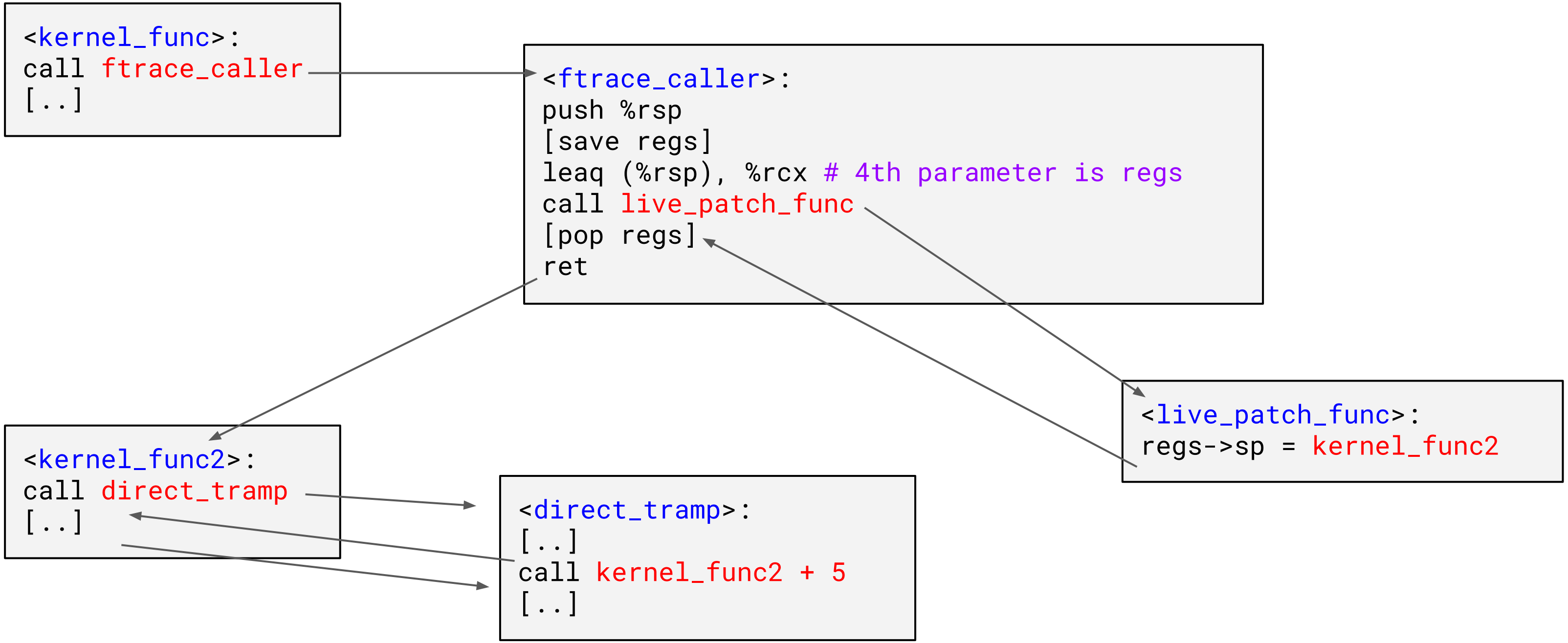
```
<ftrace_caller>:
push %rsp
[save regs]
leaq (%rsp), %rcx # 4th parameter is regs
call live_patch_func
[pop regs]
ret
```

```
<live_patch_func>:
regs->sp = kernel_func2
```

```
<kernel_func2>:
call direct_tramp
[..]
```

```
<direct_tramp>:
[..]
call kernel_func + 5
[..]
```

# Live kernel patching with BPF direct trampolines



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

```
<ftrace_caller>:
push %rsp
[save regs]
leaq (%rsp), %rcx # 4th parameter is regs
call live_patch_func
[pop regs]
ret
```

```
<live_patch_func>:
regs->sp = kernel_func2
```

```
<kernel_func2>:
call direct_tramp
[..]
```

```
<direct_tramp>:
[..]
call kernel_func2 + 5
[..]
```