kprobe multi updates

jiri olsa / isovalent at cisco

SESSION

- new attach-type on top of kprobe-multi
- allows attach to function entry/exit
- why?
- now we need 2 links
- waste and no control

SESSION

- one program attached for function entry and return
- conditional program execution on return probe
- session cookie
- for both kprobe/uprobe_multi links

```
extern bool bpf_session_is_return(void) __ksym;

extern __u64 *bpf_session_cookie(void) __ksym;
```

RETURN PROGRAM EXECUTION

```
SEC("kprobe.session/foo")
int test_kprobe(struct pt_regs *ctx)
{
   if (bpf_session_is_return()) {
     // do return probe logic
   } else {
     // do entry probe logic/filtering
     return should_executed_return_probe ? 0 : 1;
  }
}
```

BPF_GET_ATTACH_COOKIE

```
SEC("kprobe.session/foo")
int test_kprobe(struct pt_regs *ctx)
{
   __u64 cookie = bpf_get_attach_cookie();
   ...
}
```

BPF_SESSION_COOKIE

```
SEC("kprobe.session/foo")
int test_kprobe(struct pt_regs *ctx)
{
   long val, *cookie = bpf_session_cookie();

   if (bpf_session_is_return()) {
     val = *cookie;
     ...
   } else {
     *cookie = (long) 0xWHATEVER;
}
```

KPROBE MULTI SESSION

- merged
- current kprobe/fprobe support
- fprobe-on-graph support

UPROBE MULTI SESSION

- uprobe lacks both:
  - ‘do not execute return probe’ logic and
  - session data support
- uprobe entry handler can return 0 or 1

  and 1 means remove the uprobe
- new version of entry consumer handler
- RFC soon

FPROBE ON FGRAPH

- implement fprobe on top of fgraph
- ongoing patchset development by

  Masami Hiramatsu, Steven Rostedt

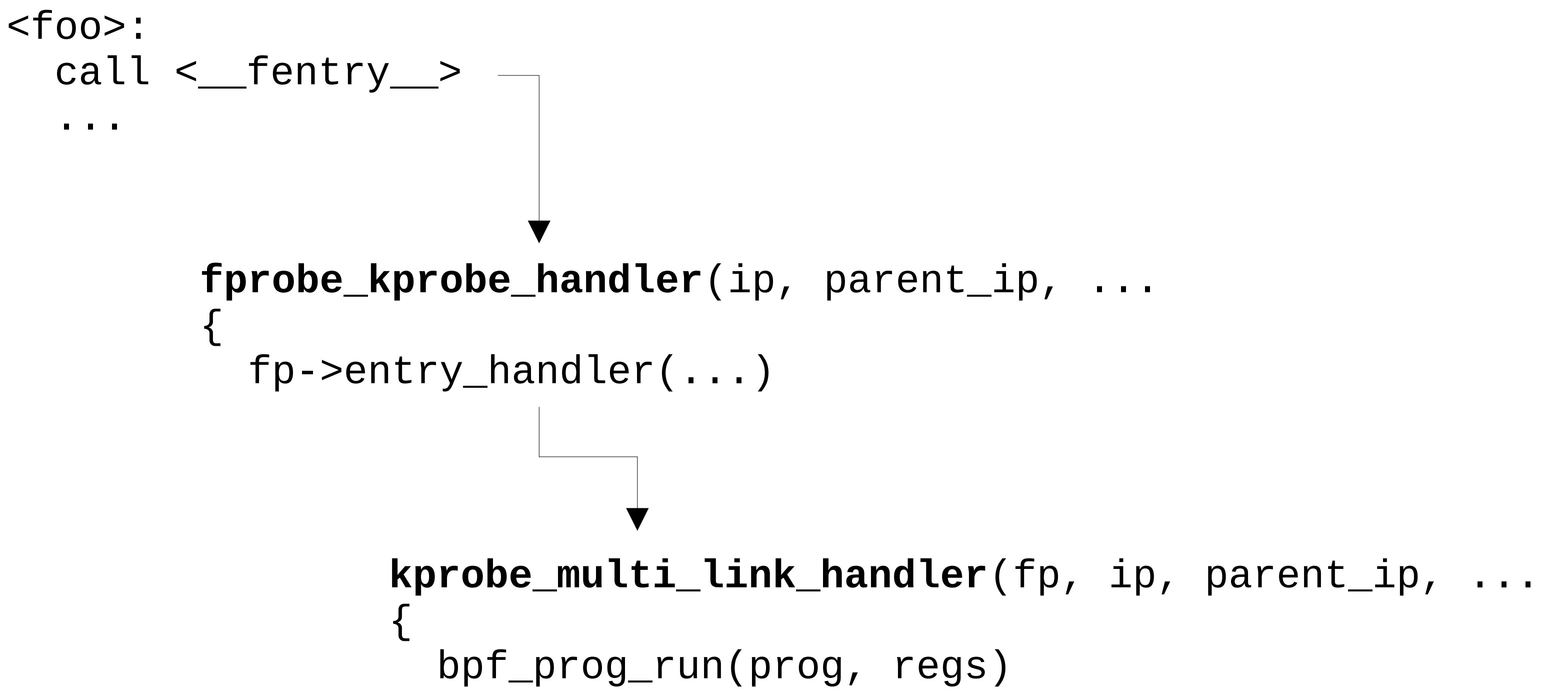
  https://lore.kernel.org/bpf/171318533841.254850.15841395205784342850.stgit@devnote2/
- fprobe is the base of kprobe_multi

OBJECTIVES

- get rid of rethook

  (shadow stack per process)
- tracers consolidation
- future use in kretprobe

CURRENT FPROBE

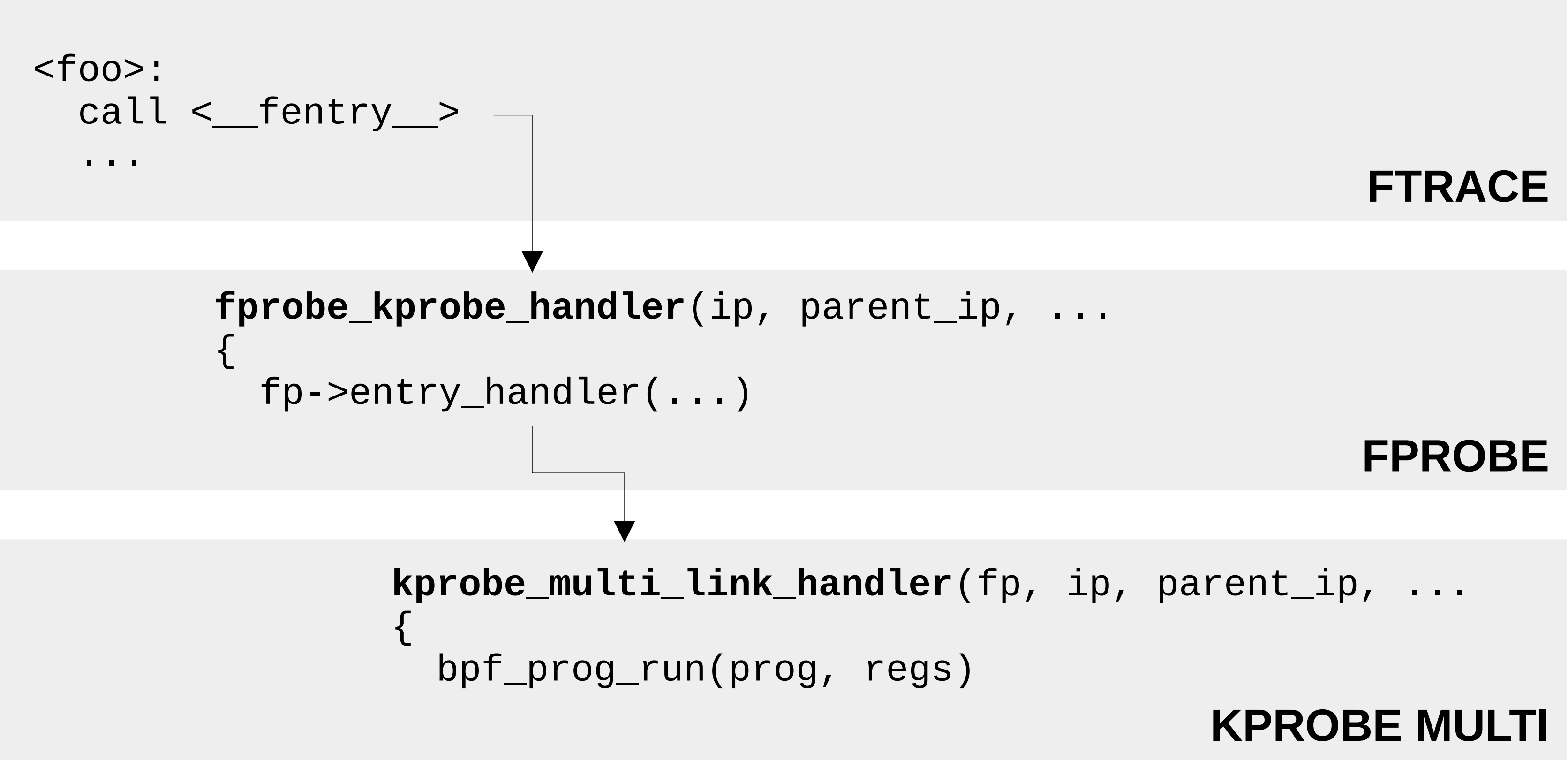


```
<foo>:
  call <__fentry__>
  ...
```

```
fprobe_kprobe_handler(ip, parent_ip, ...
{
  fp->entry_handler(...)
```

```
kprobe_multi_link_handler(fp, ip, parent_ip, ...
{
  bpf_prog_run(prog, regs)
```

CURRENT FPROBE



```
<foo>:
  call <__fentry__>
  ...
```

FTRACE

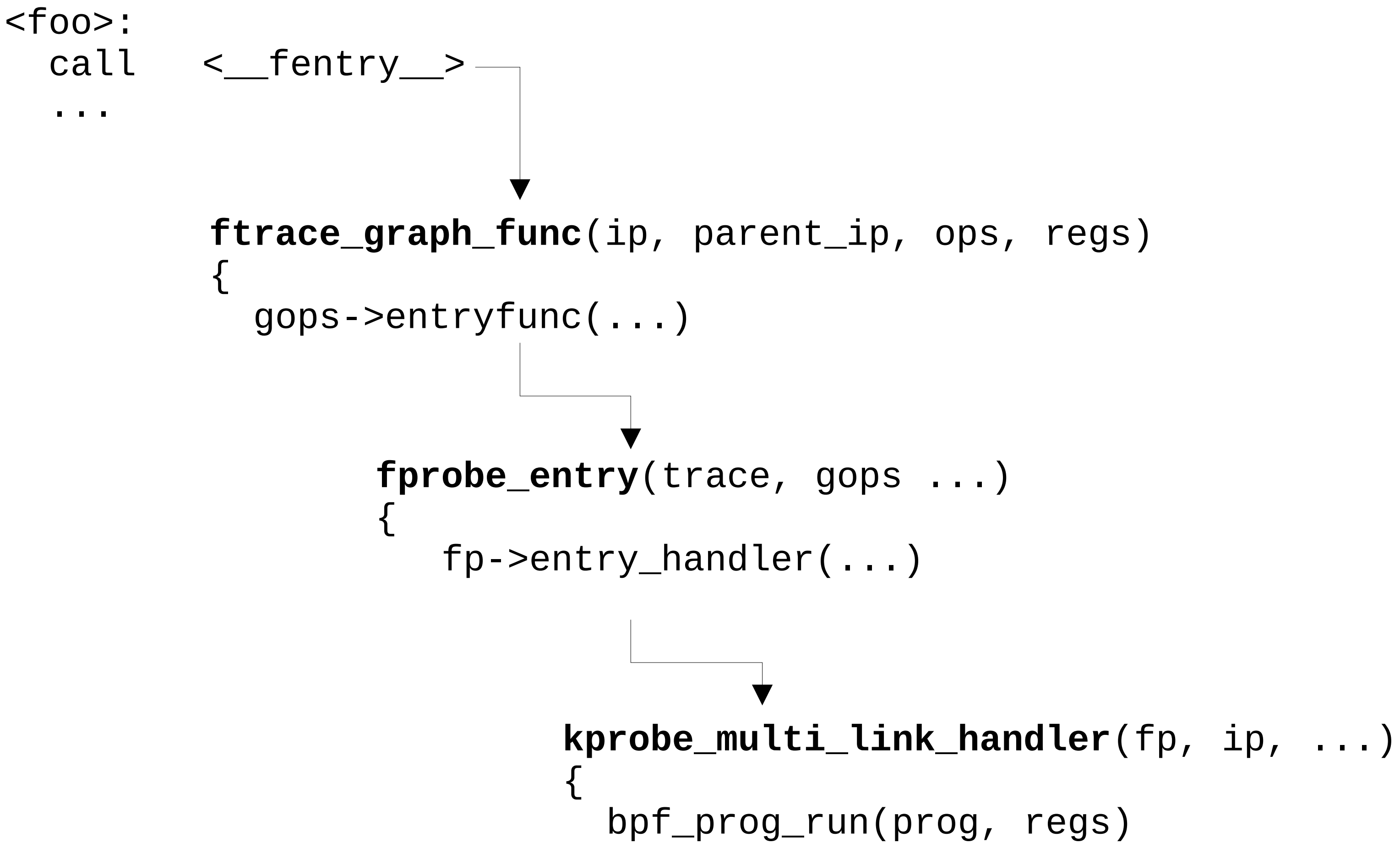
```
fprobe_kprobe_handler(ip, parent_ip, ...
{
  fp->entry_handler(...)
```

FPROBE

```
kprobe_multi_link_handler(fp, ip, parent_ip, ...
{
  bpf_prog_run(prog, regs)
```

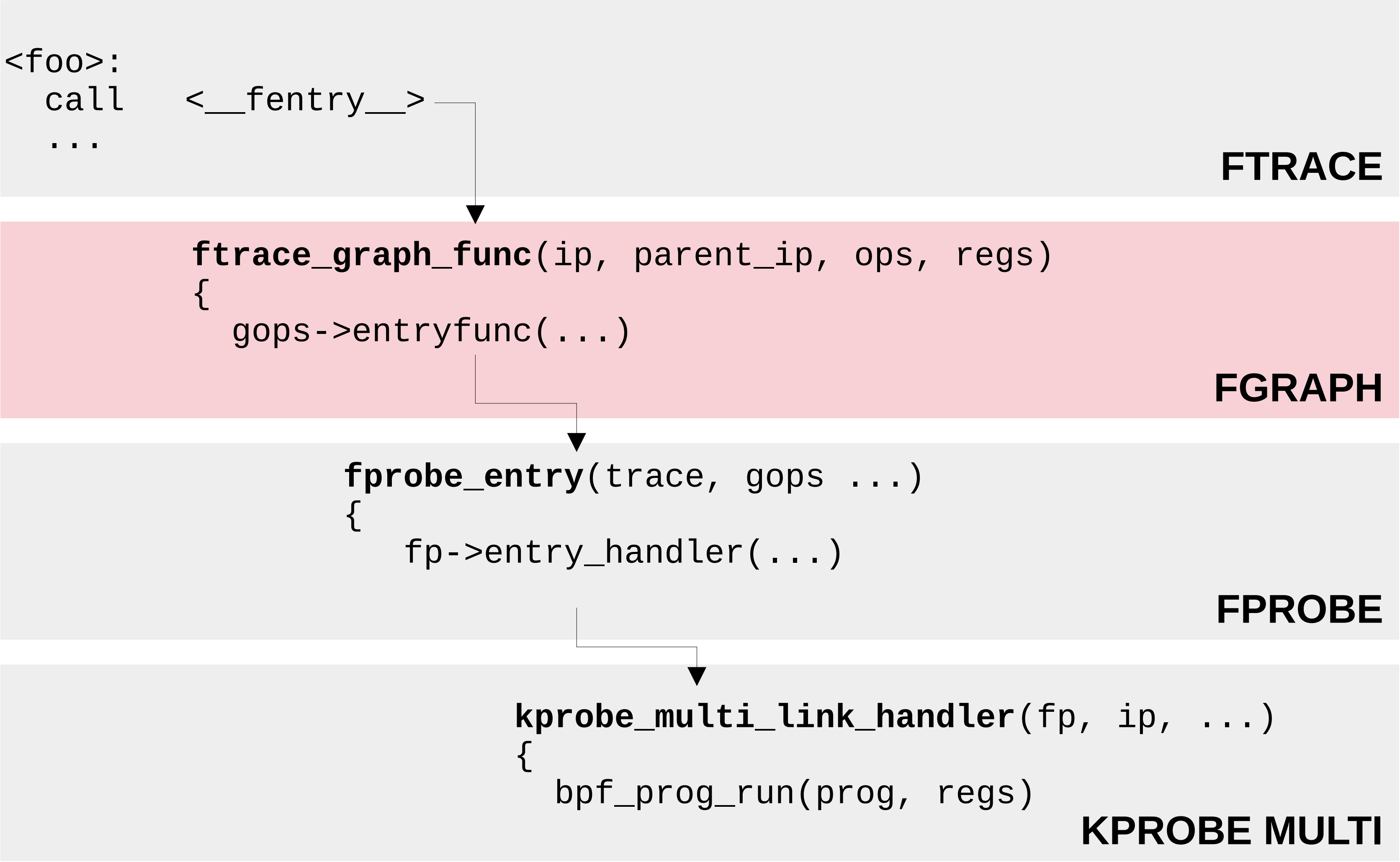
KPROBE MULTI

FPROBE ON FGRAPH



```
<foo>:
  call   <__fentry__>
  ...

         ftrace_graph_func(ip, parent_ip, ops, regs)
         {
           gops->entryfunc(...)

                 fprobe_entry(trace, gops ...)
                 {
                    fp->entry_handler(...)

                          kprobe_multi_link_handler(fp, ip, ...)
                          {
                            bpf_prog_run(prog, regs)
```

FPROBE ON FGRAPH



```
<foo>:
  call   <__fentry__>
  ...
```

FTRACE

```
ftrace_graph_func(ip, parent_ip, ops, regs)
{
  gops->entryfunc(...)
```

FGRAPH

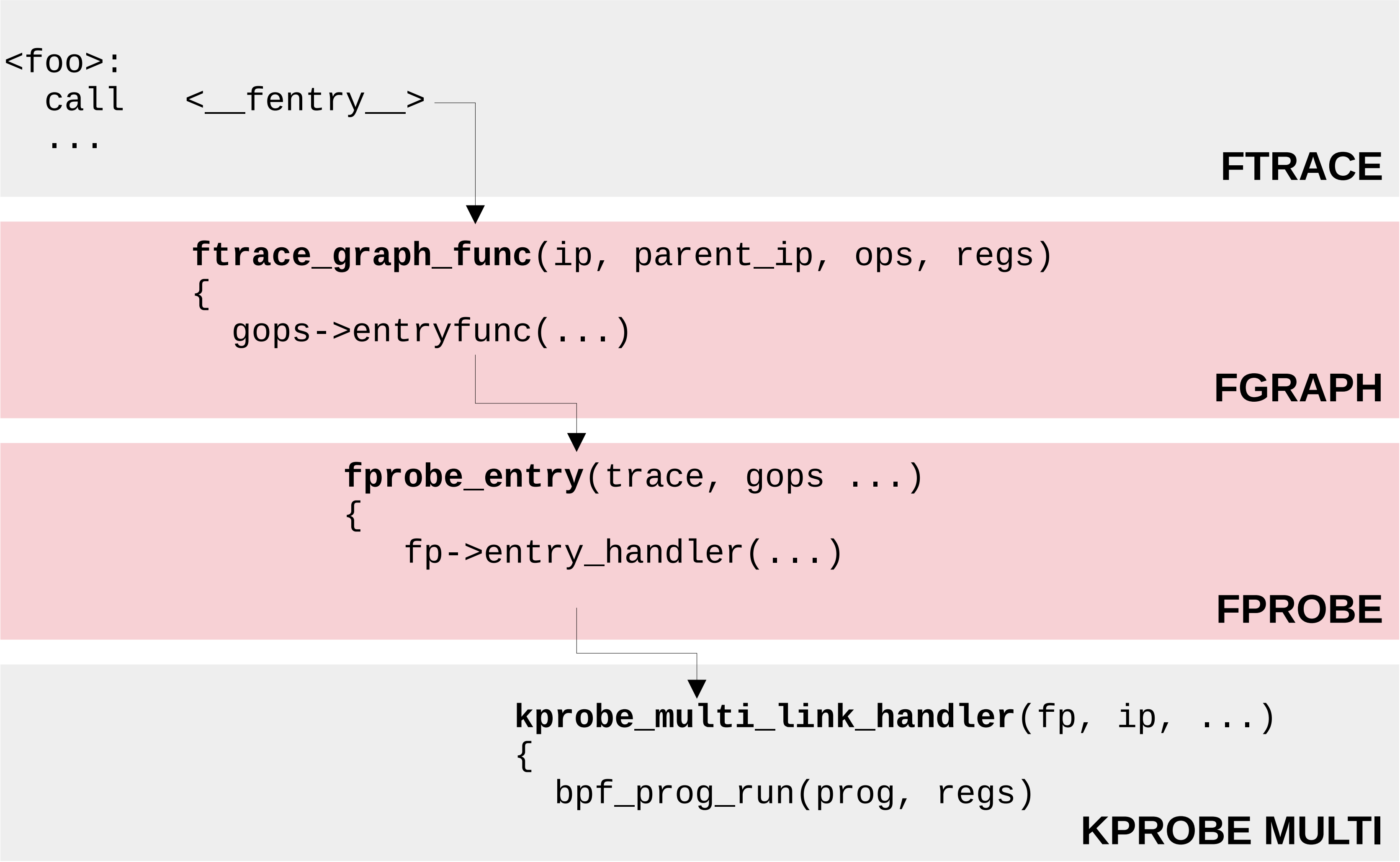
```
fprobe_entry(trace, gops ...)
{
   fp->entry_handler(...)
```

FPROBE

```
kprobe_multi_link_handler(fp, ip, ...)
{
  bpf_prog_run(prog, regs)
```

KPROBE MULTI

FPROBE ON FGRAPH



```
<foo>:
  call   <__fentry__>
  ...
```

FTRACE

```
ftrace_graph_func(ip, parent_ip, ops, regs)
{
  gops->entryfunc(...)
```

FGRAPH

```
fprobe_entry(trace, gops ...)
{
   fp->entry_handler(...)
```

FPROBE

```
kprobe_multi_link_handler(fp, ip, ...)
{
  bpf_prog_run(prog, regs)
```

KPROBE MULTI

FGRAPH TRACER

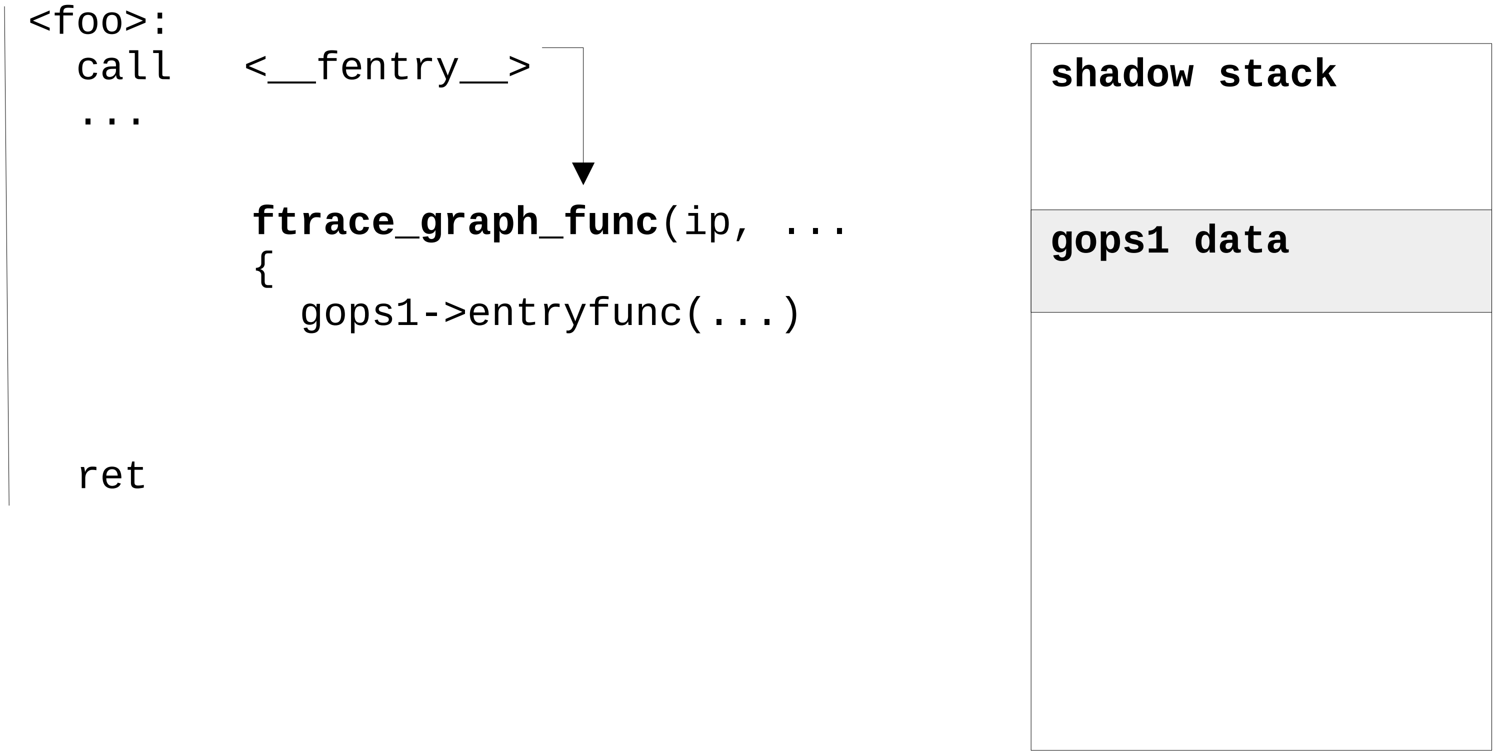
- max 16 of them
- fprobe registers 1 graph tracer
- ftrace_opts user (fgraph_ops)
- maintains shadow stack per task

```
static struct fgraph_ops fprobe_graph_ops = {
        .entryfunc      = fprobe_entry,
        .retfunc        = fprobe_return,
        .skip_timestamp = true,
};

ret = register_ftrace_graph(&fprobe_graph_ops);
```

SHADOW STACK

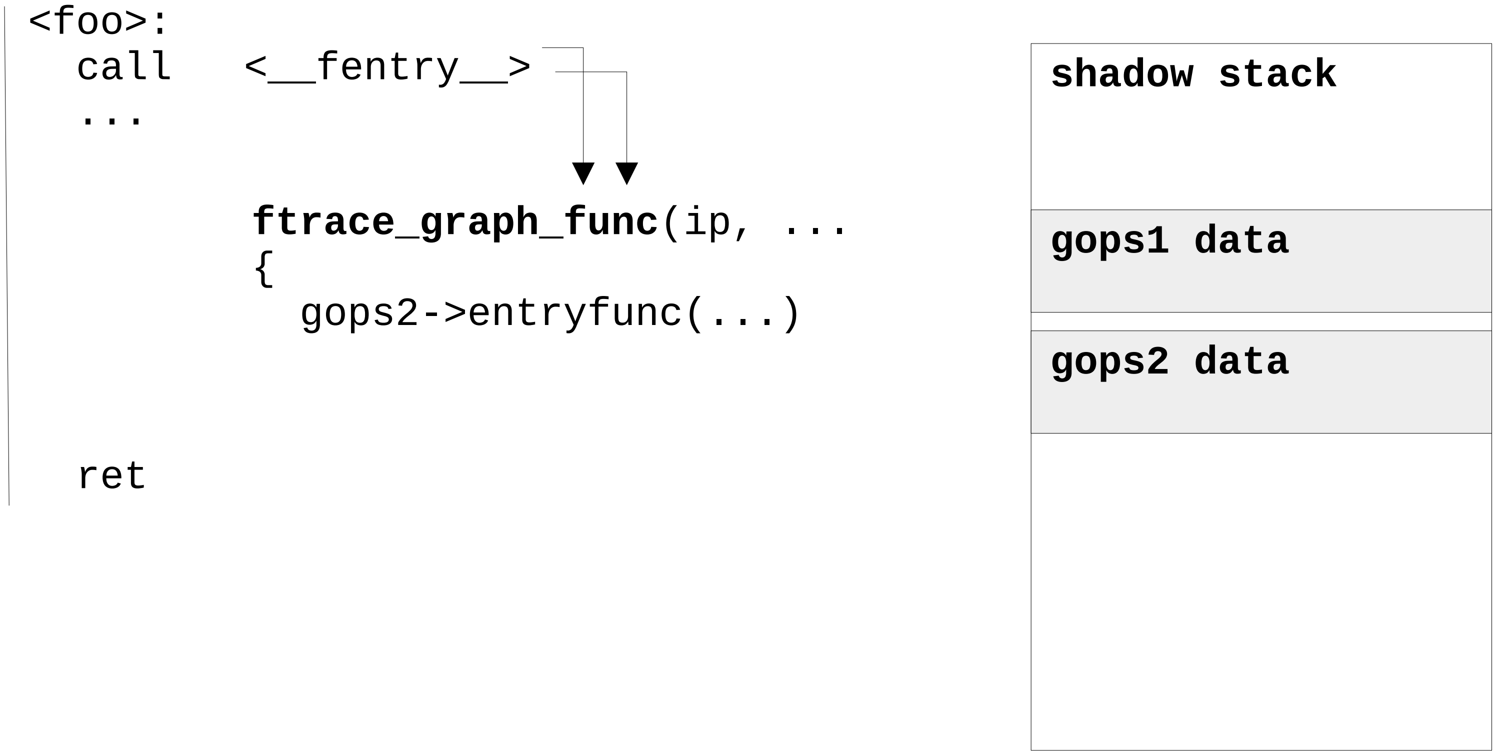
- per task (1 page)
- control data for return probe
- each tracer can request data on stack

SHADOW STACK



```
<foo>:
  call   <__fentry__>
  ...


         ftrace_graph_func(ip, ...
         {
           gops1->entryfunc(...)



  ret
```

SHADOW STACK



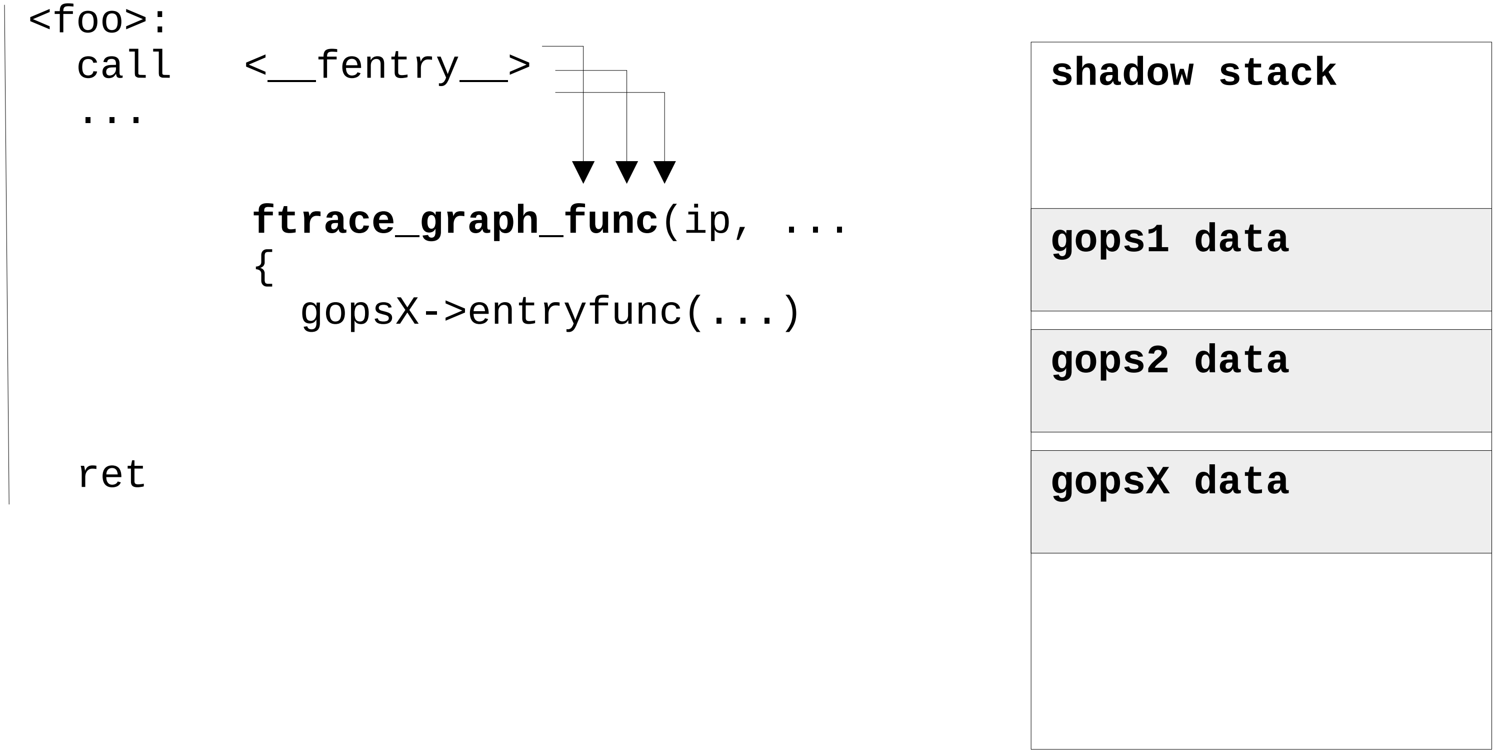
```
<foo>:
  call   <__fentry__>
  ...

         ftrace_graph_func(ip, ...
         {
           gops2->entryfunc(...)



  ret
```

shadow stack

gops1 data

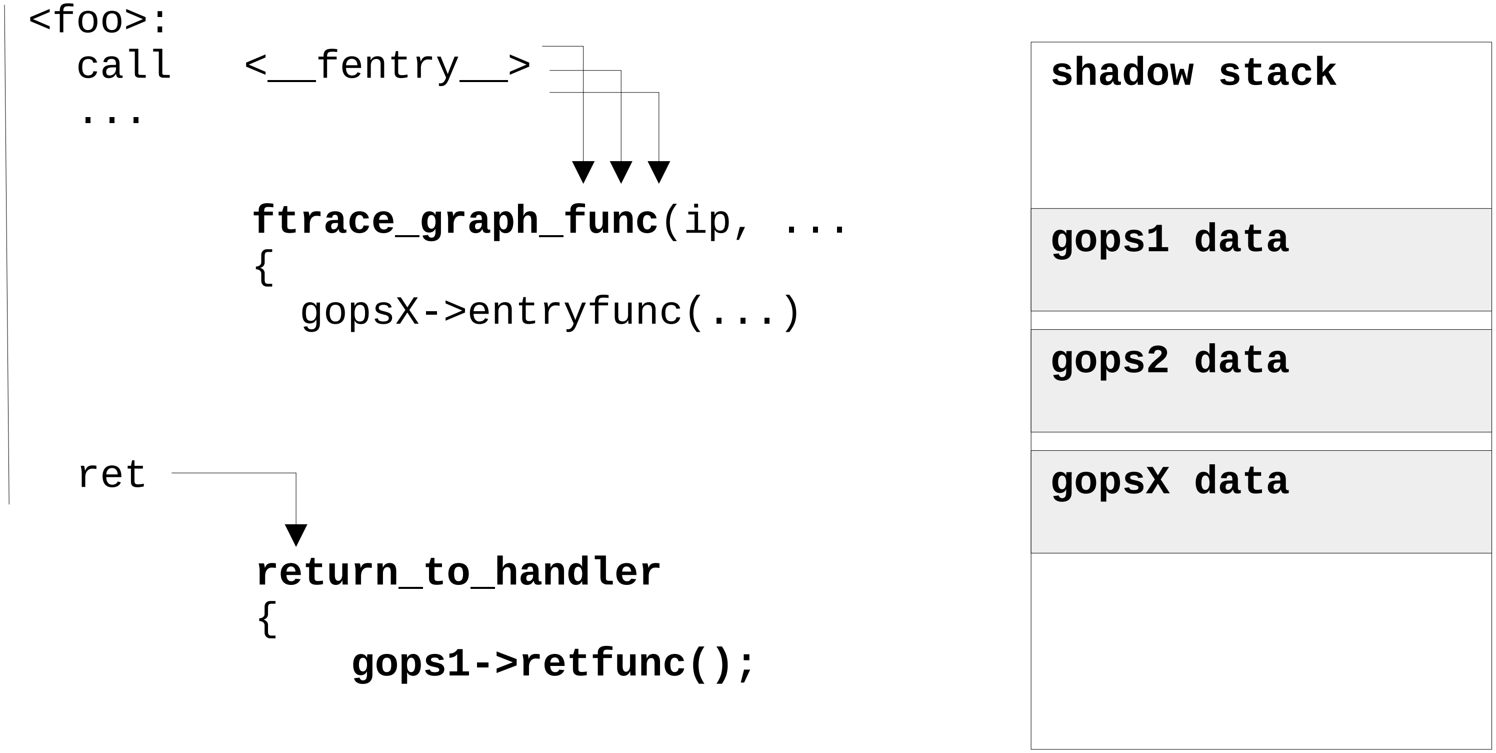
gops2 data

SHADOW STACK



```
<foo>:
  call   <__fentry__>
  ...


         ftrace_graph_func(ip, ...
         {
           gopsX->entryfunc(...)



  ret
```

SHADOW STACK



```
<foo>:
  call   <__fentry__>
  ...

         ftrace_graph_func(ip, ...
         {
           gopsX->entryfunc(...)


  ret

         return_to_handler
         {
             gops1->retfunc();
```

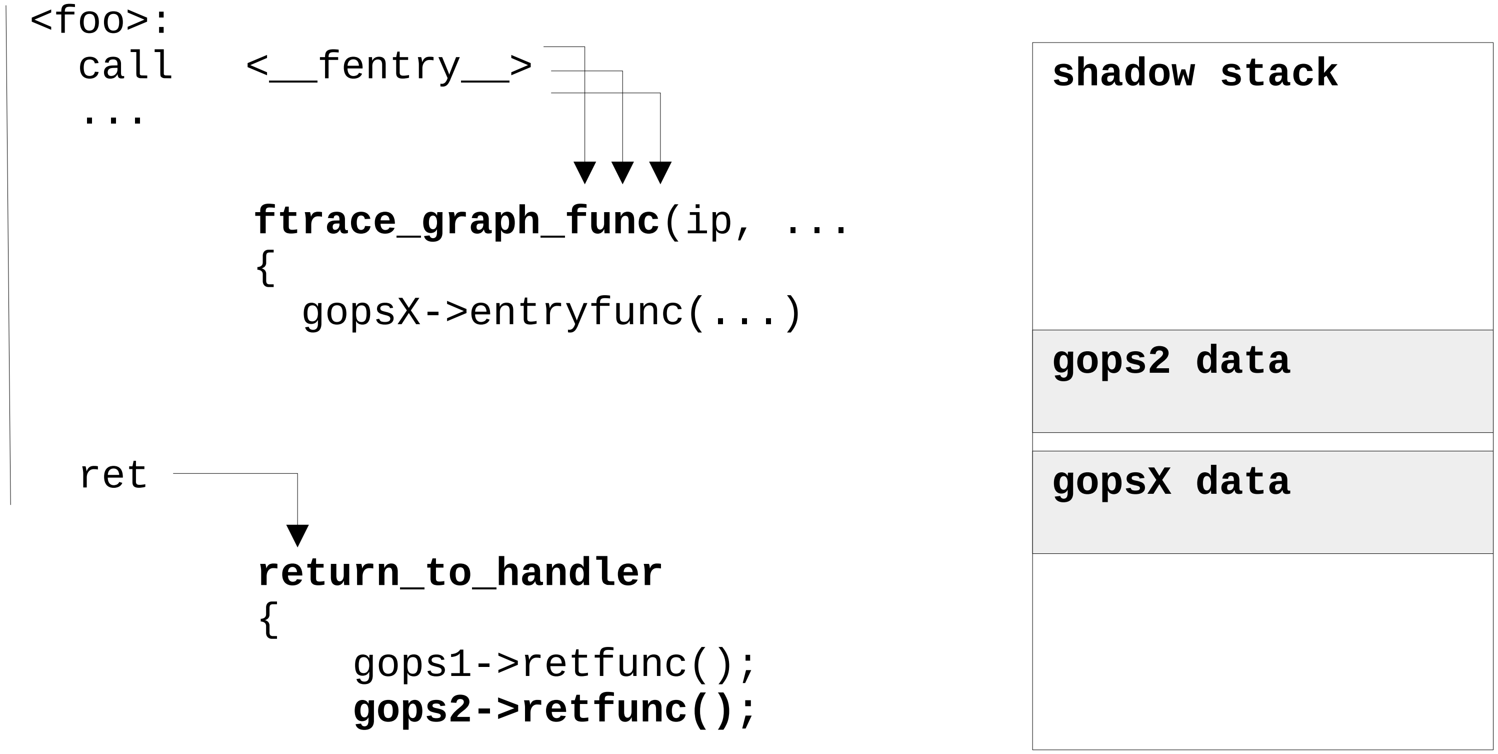
shadow stack

gops1 data

gops2 data

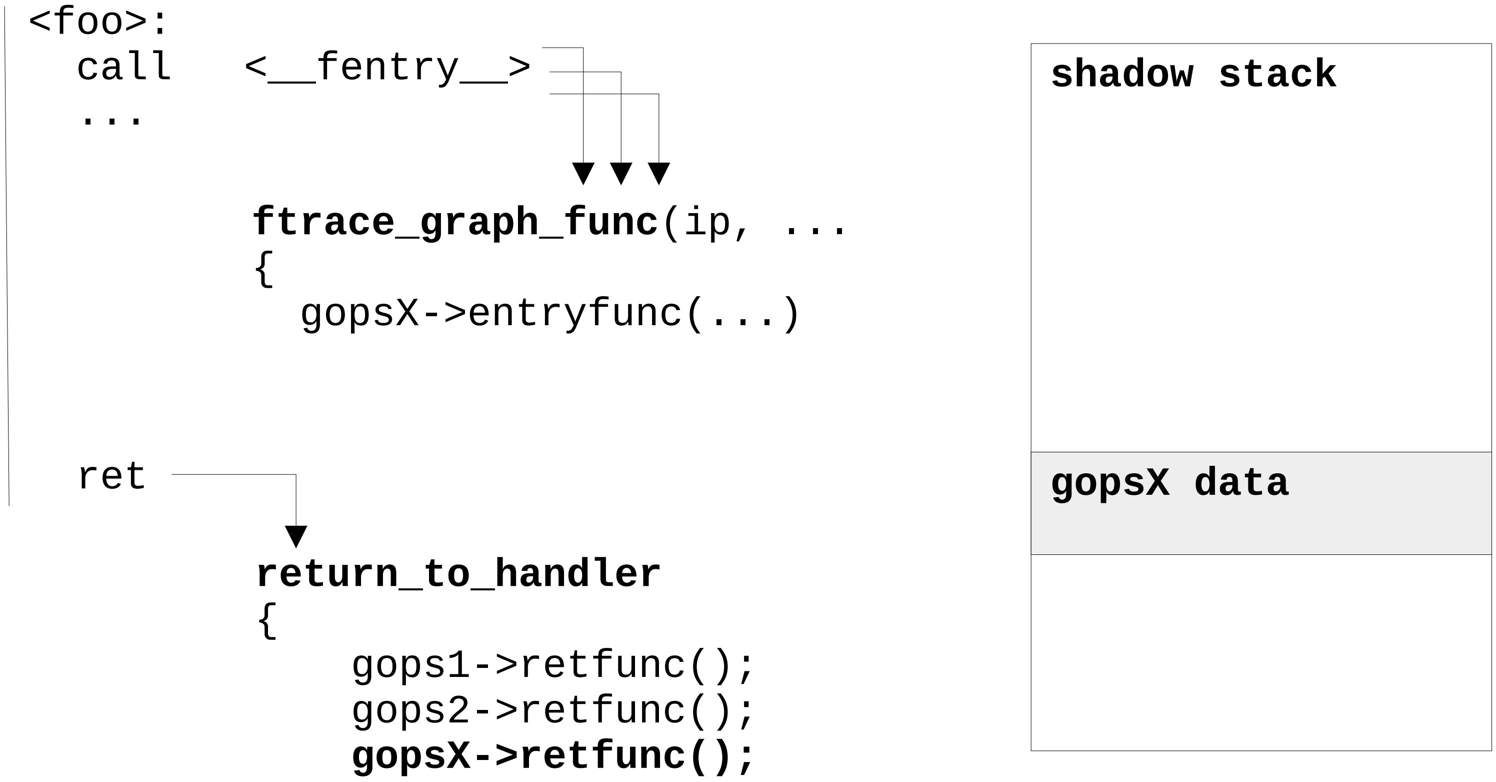
gopsX data

SHADOW STACK



```
<foo>:
  call   <__fentry__>
  ...

         ftrace_graph_func(ip, ...
         {
           gopsX->entryfunc(...)


  ret

         return_to_handler
         {
             gops1->retfunc();
             gops2->retfunc();
```

SHADOW STACK



```
<foo>:
  call   <__fentry__>
  ...
```

```
ftrace_graph_func(ip, ...
{
  gopsX->entryfunc(...)
```

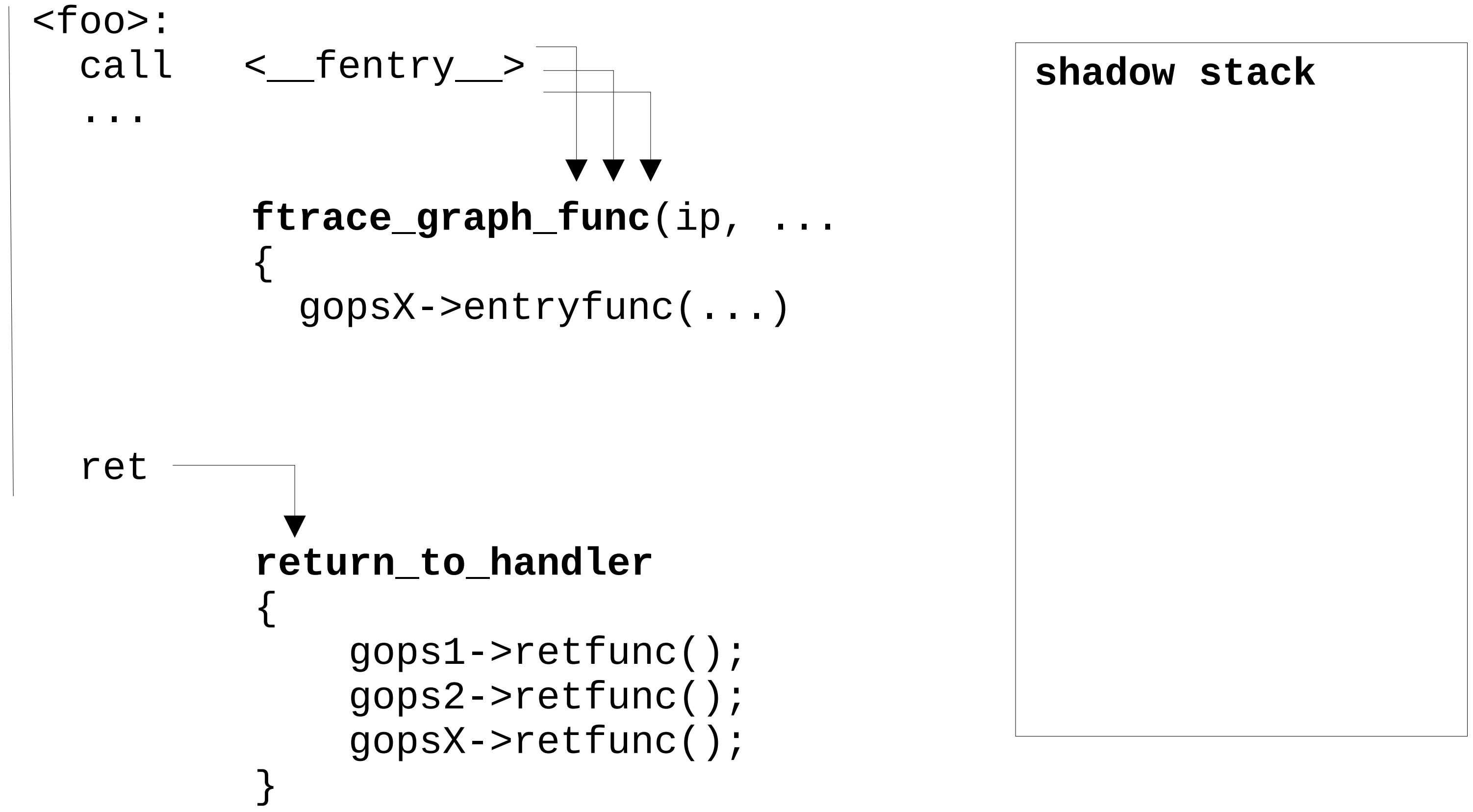
```
  ret
```

```
return_to_handler
{
    gops1->retfunc();
    gops2->retfunc();
    gopsX->retfunc();
```

shadow stack

gopsX data

SHADOW STACK



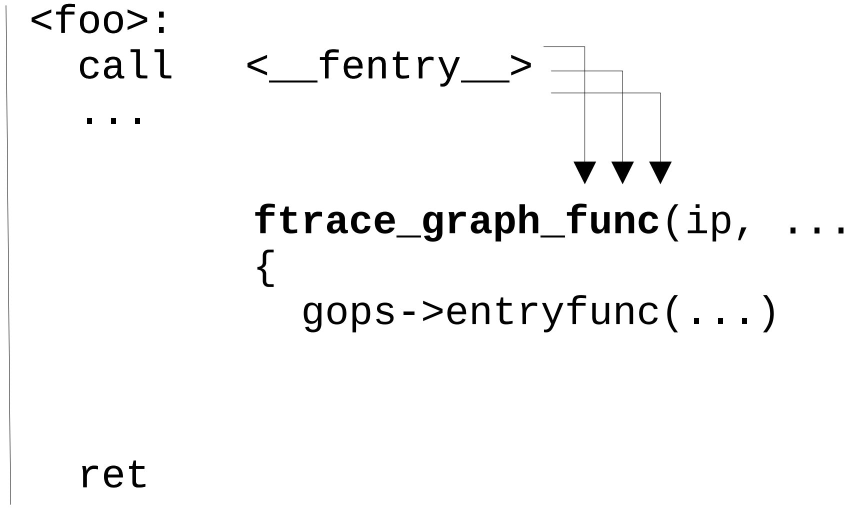
```
<foo>:
  call   <__fentry__>
  ...

         ftrace_graph_func(ip, ...
         {
           gopsX->entryfunc(...)



  ret

         return_to_handler
         {
             gops1->retfunc();
             gops2->retfunc();
             gopsX->retfunc();
         }
```

shadow stack

FPROBE

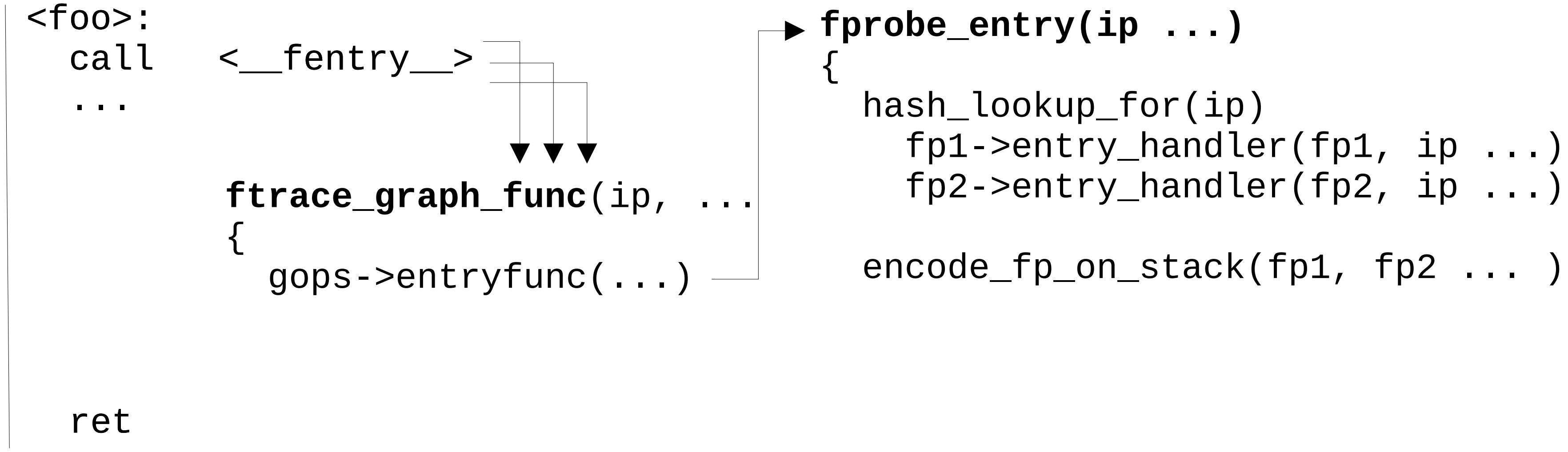
- fgraph_ops user
- 1 entry point for all fprobe users
- hash table to get fprobe object from ip

FPROBE



```
<foo>:
  call   <__fentry__>
  ...

         ftrace_graph_func(ip, ...
         {
           gops->entryfunc(...)



  ret
```

FPROBE

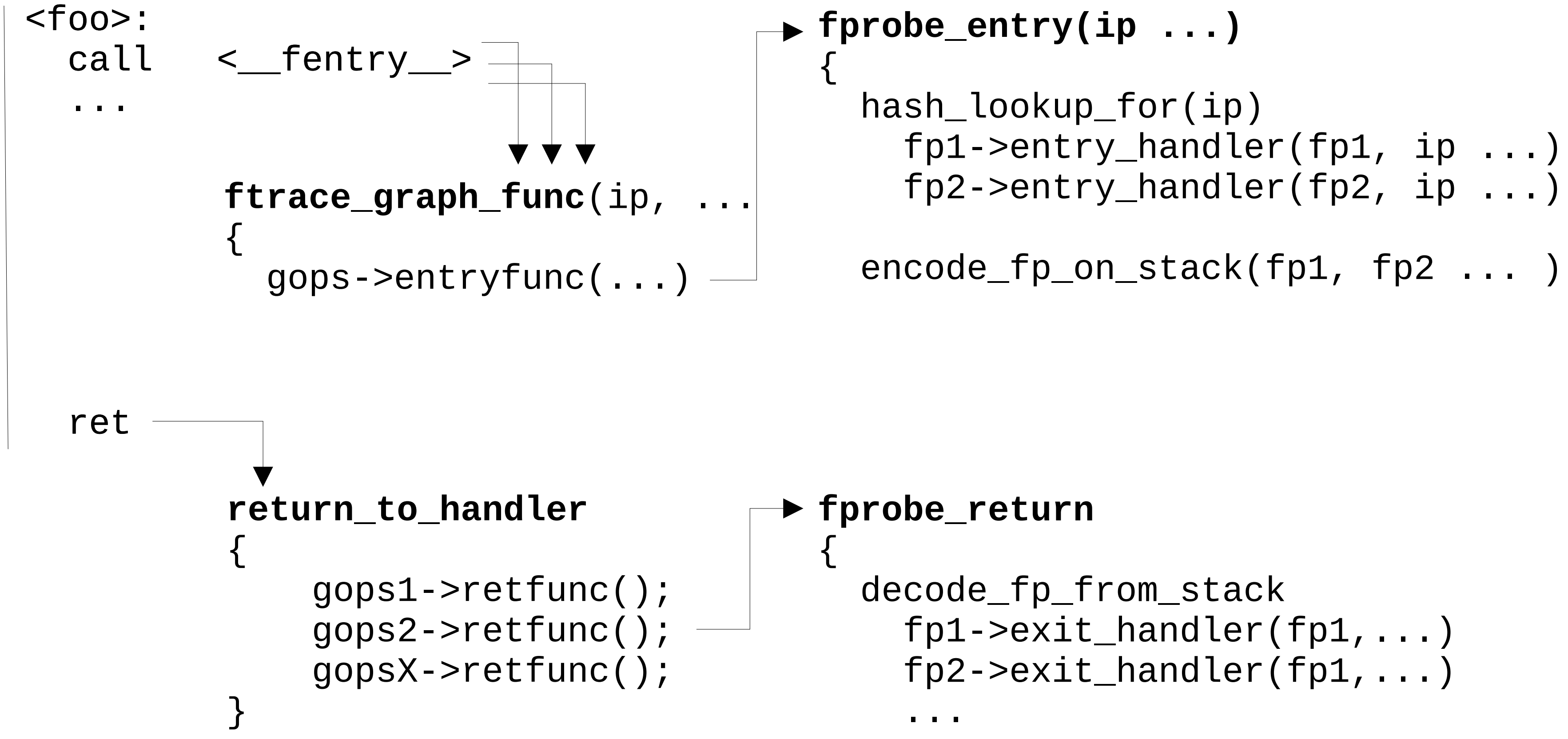


```
<foo>:
  call   <__fentry__>
  ...

  ret
```

```
ftrace_graph_func(ip, ...
{
  gops->entryfunc(...)
```

```
fprobe_entry(ip ...)
{
  hash_lookup_for(ip)
    fp1->entry_handler(fp1, ip ...)
    fp2->entry_handler(fp2, ip ...)

  encode_fp_on_stack(fp1, fp2 ... )
```

FPROBE



```
<foo>:
  call   <__fentry__>
  ...

  ret
```

```
ftrace_graph_func(ip, ...
{
  gops->entryfunc(...)
```

```
fprobe_entry(ip ...)
{
  hash_lookup_for(ip)
    fp1->entry_handler(fp1, ip ...)
    fp2->entry_handler(fp2, ip ...)

  encode_fp_on_stack(fp1, fp2 ... )
```

```
return_to_handler
{
    gops1->retfunc();
    gops2->retfunc();
    gopsX->retfunc();
}
```

```
fprobe_return
{
  decode_fp_from_stack
    fp1->exit_handler(fp1,...)
    fp2->exit_handler(fp1,...)
    ...
```

BENCHMARKs

before:

```
kernel-count   :  108.006 ± 0.096M/s
kprobe         :   31.276 ± 0.058M/s
kprobe-multi   :   37.821 ± 0.143M/s
kretprobe      :   12.038 ± 0.048M/s
kretprobe-multi:   12.997 ± 0.044M/s
```

after:

```
kernel-count   :  109.064 ± 0.165M/s
kprobe         :   32.127 ± 0.180M/s
kprobe-multi   :   36.242 ± 0.134M/s
kretprobe      :   12.299 ± 0.030M/s
kretprobe-multi:   15.364 ± 0.047M/s  (+18.2%)
```

BENCHMARKs (Andrii)

before:

```
kprobe         :   24.634 ± 0.205M/s
kprobe-multi   :   28.898 ± 0.531M/s
kretprobe      :   10.478 ± 0.015M/s
kretprobe-multi:   11.012 ± 0.063M/s
```

after:

```
kprobe         :   25.144 ± 0.027M/s (+2%)
kprobe-multi   :   28.909 ± 0.074M/s
kretprobe      :    9.482 ± 0.008M/s (-9.5%)
kretprobe-multi:   13.688 ± 0.027M/s (+24%)
```

thanks, questions..

PER PROGRAM RE-ENTRY CHECKS

- no progress so far..