第三届 eBPF开发者大会
www.ebpftravel.com

TRAMPOLINE & TRACING MULTI-LINK 技术探索

天翼云科技有限公司 董梦龙

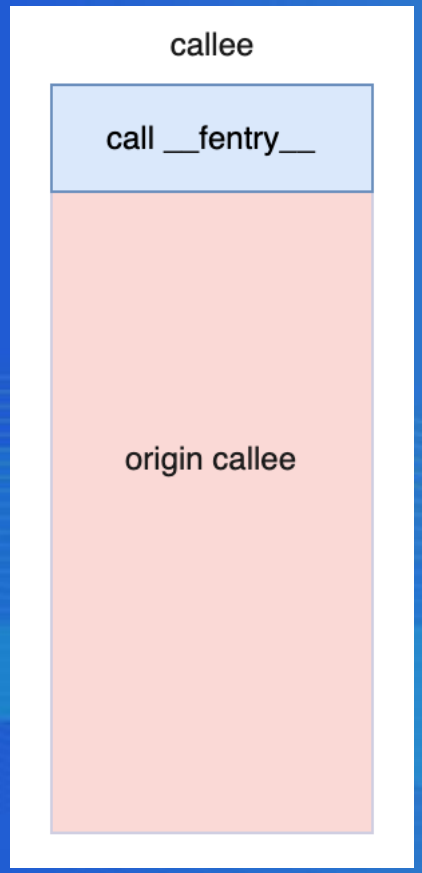
中国·西安

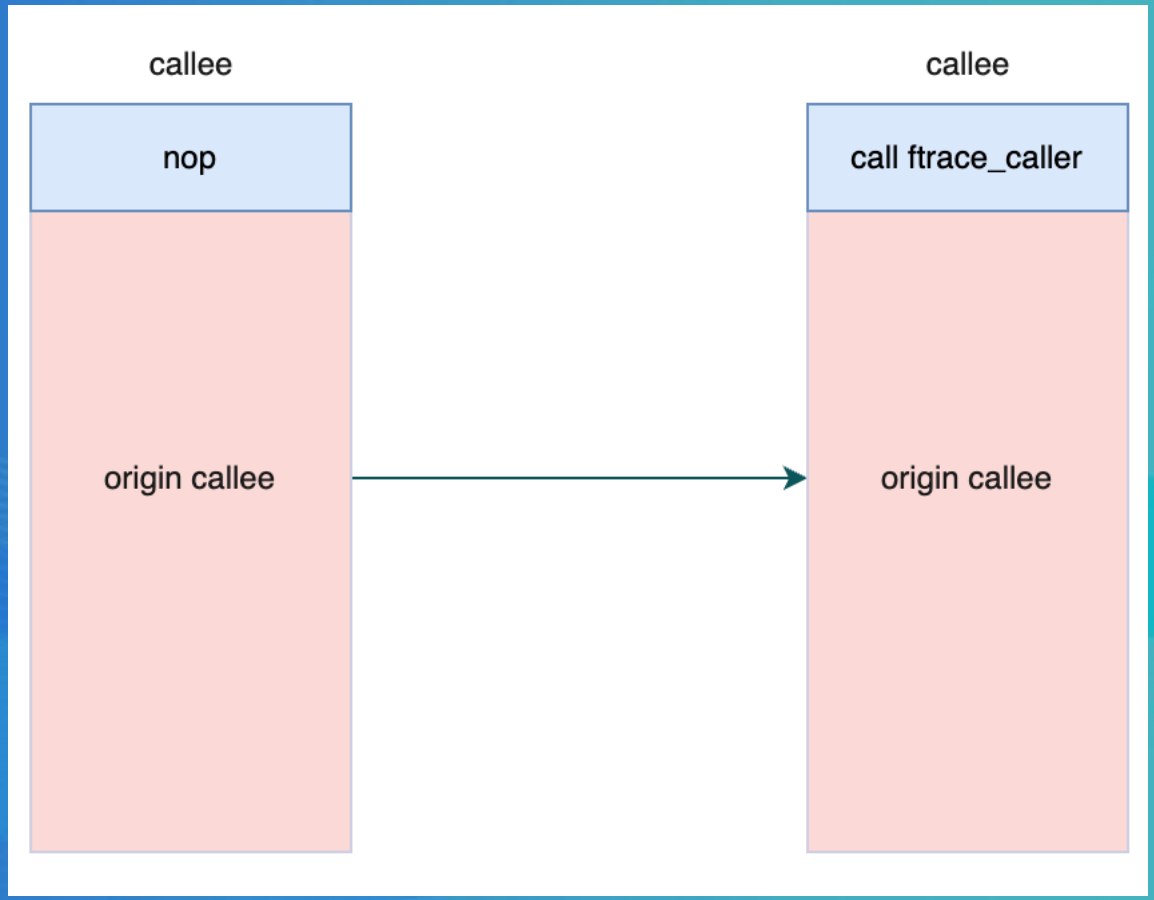
① TRAMPOLINE基本原理

天翼云科技有限公司

① FTRACE基本原理

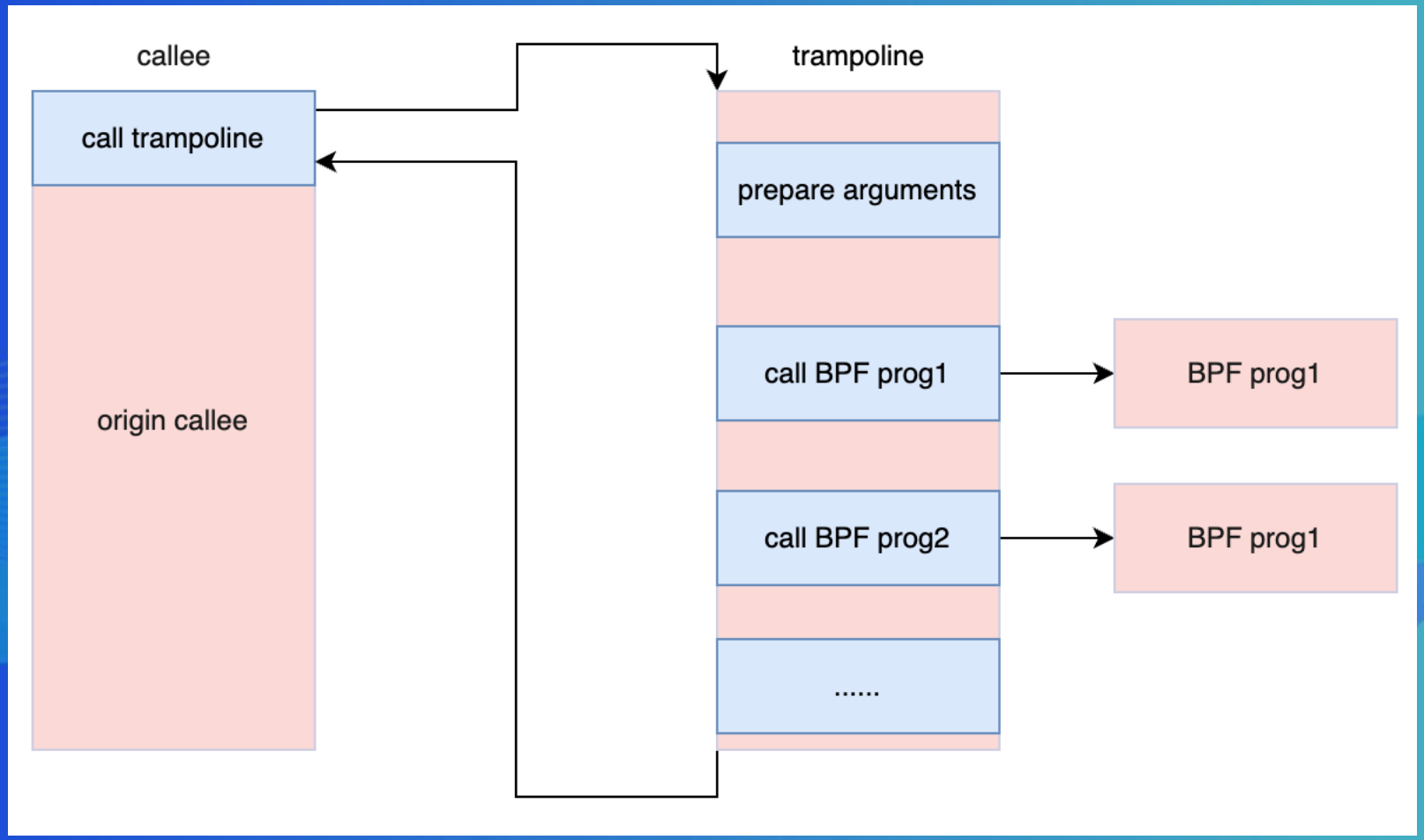
-mfentry
-fpatchable-function-entry





① TRAMPOLINE基本原理

目标函数和BPF之间的桥梁



① TRAMPOLINE基本原理

优势：

- direct call，性能高
- 更简洁的方式来获取被跟踪的函数的参数
- 函数返回值的同时，获取函数的参数
- 修改被跟踪函数的返回值

用户：

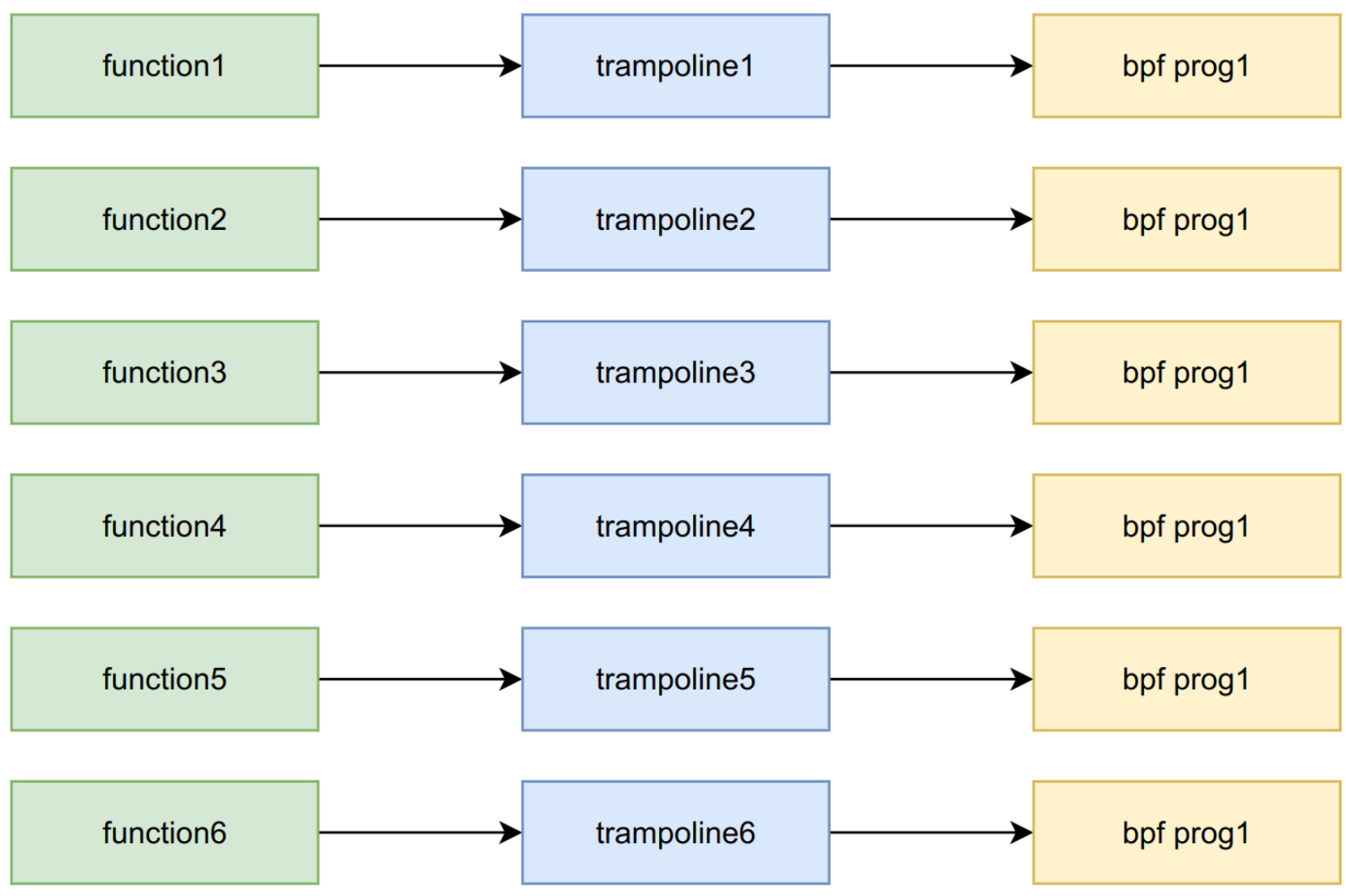
BPF_PROG_TYPE_TRACING

BPF_PROG_TYPE_EXT

BPF_PROG_TYPE_LSM

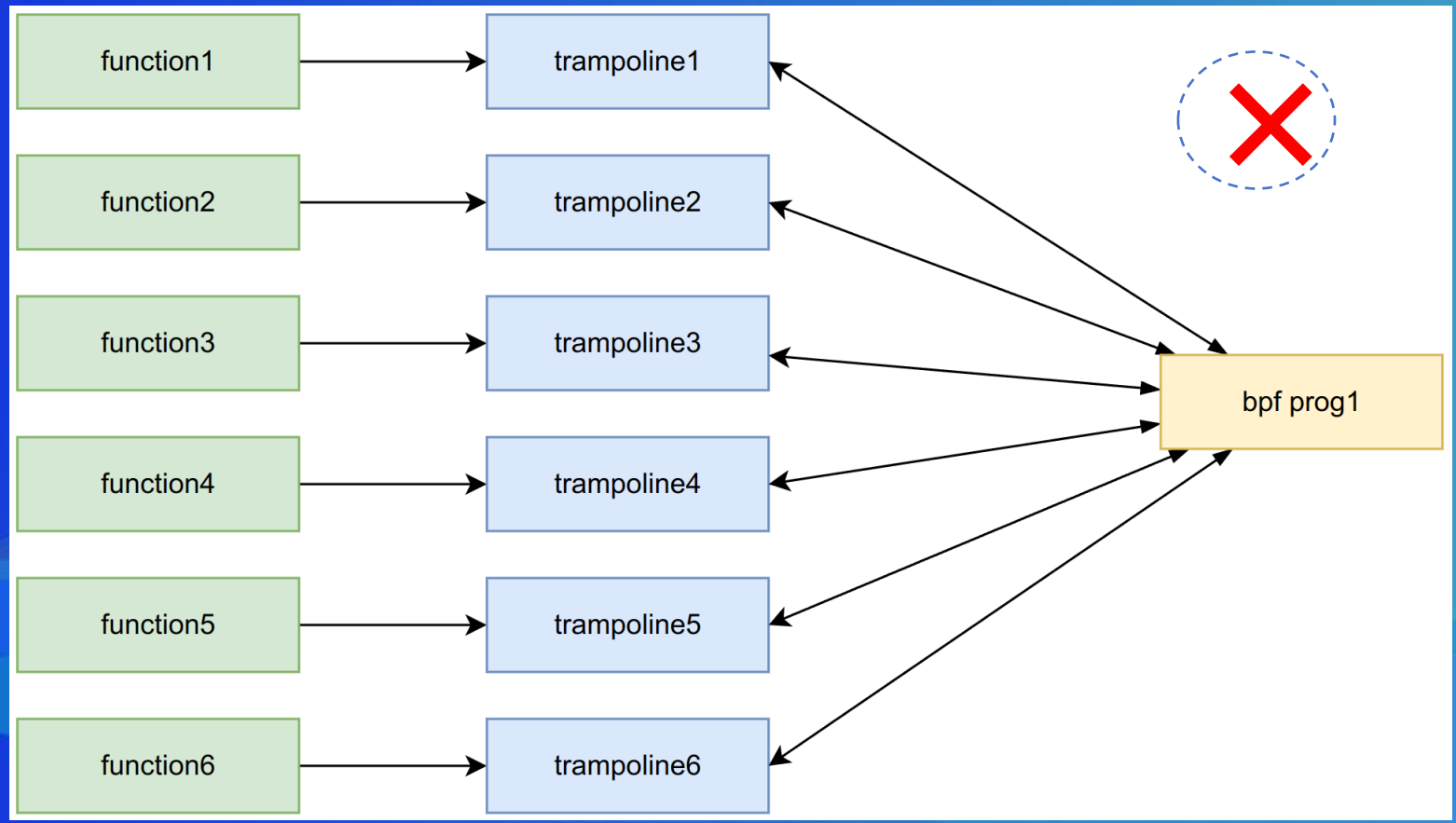
BPF_LSM_CGROUP

① TRACING劣势



- 资源开销大，创建耗时，不适合大批量场景
- 无法一个BPF程序实现HOOK多个内核函数
- 每秒只能完成几十个BPF程序的attach，超过一千个目标函数，要以分钟为单位
- 代码 & 数据耦合

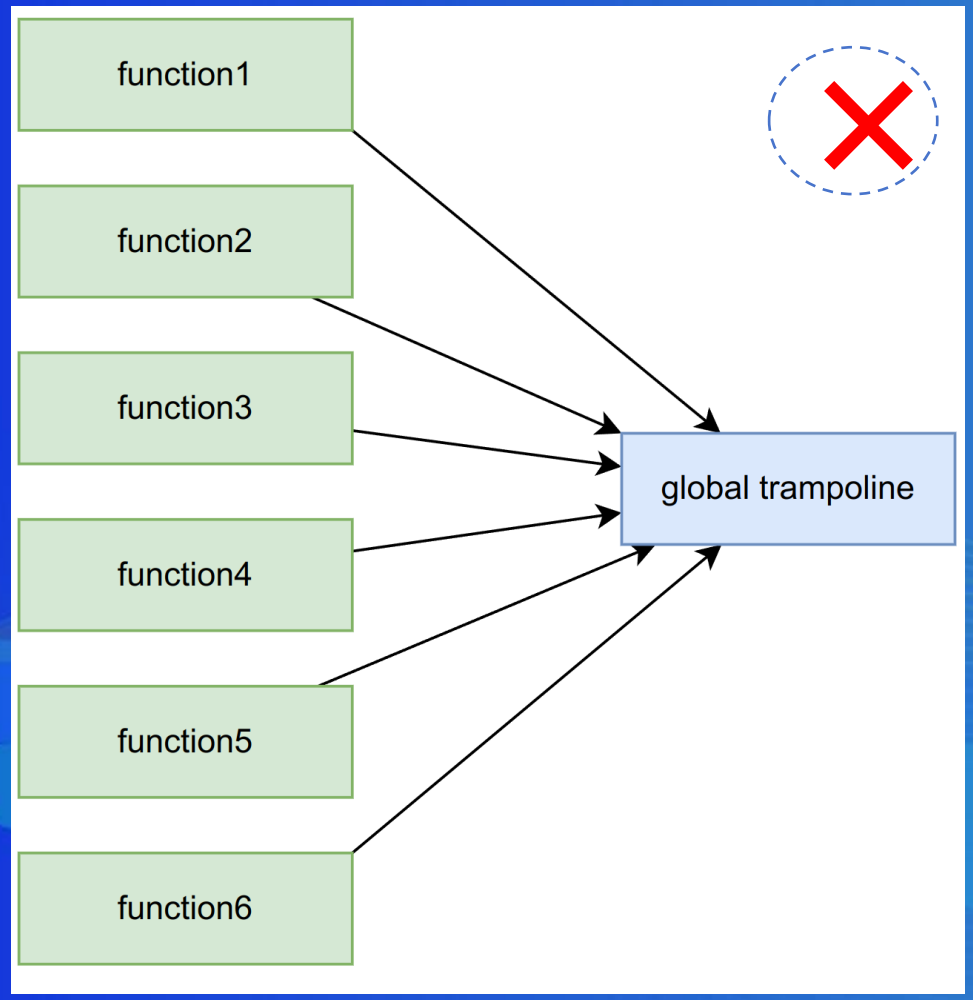
②TRACING MULTI-LINK(V1)



- 实现了tracing multi-link，可以将单个bpf程序attach到多个内核函数。
- 没有从根本上解决加载慢、资源浪费的问题
- patch series：https://lore.kernel.org/netdev/20240311093526.1010158-1-dongmenglong.8@bytedance.com/

核心思路：检查被访问过的参数类型是否一致

②TRACING MULTI-LINK(V1)



global trampoline的逻辑：

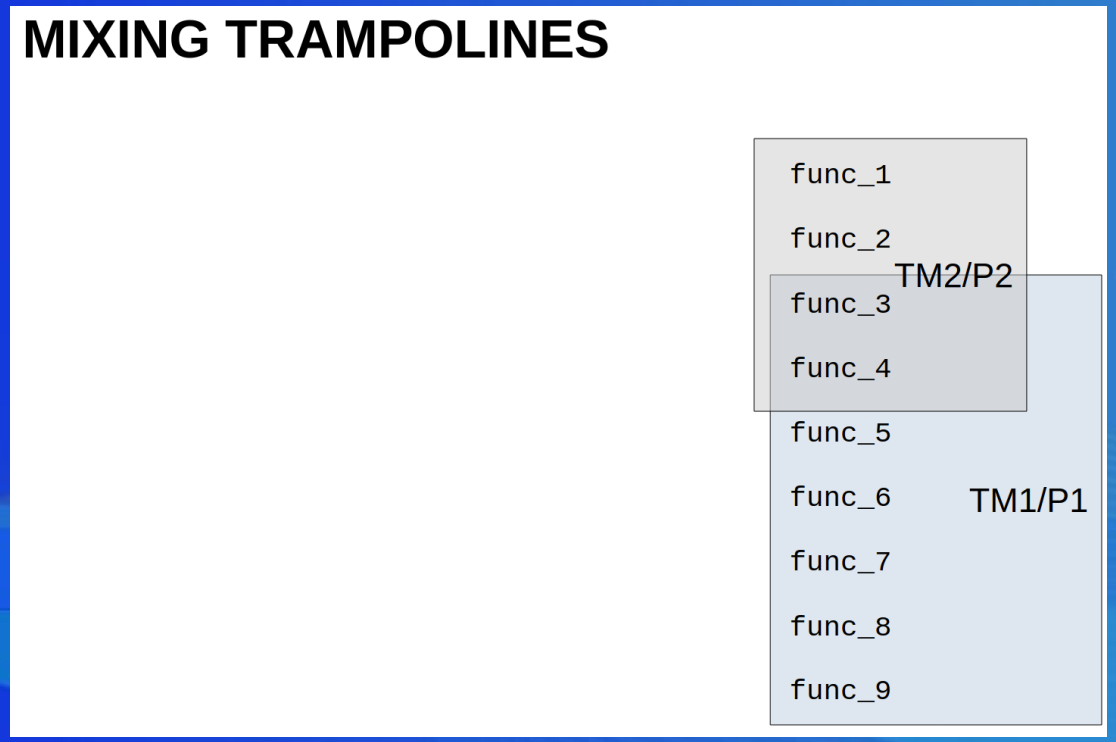
```
save regs
bpfs = trampoline_lookup_ip(ip)
fentry = bpfs->fentries
while fentry:
  fentry(ctx)
  fentry = fentry->next

call origin
save return value

fexit = bpfs->fexits
while fexit:
  fexit(ctx)
  fexit = fexit->next
```

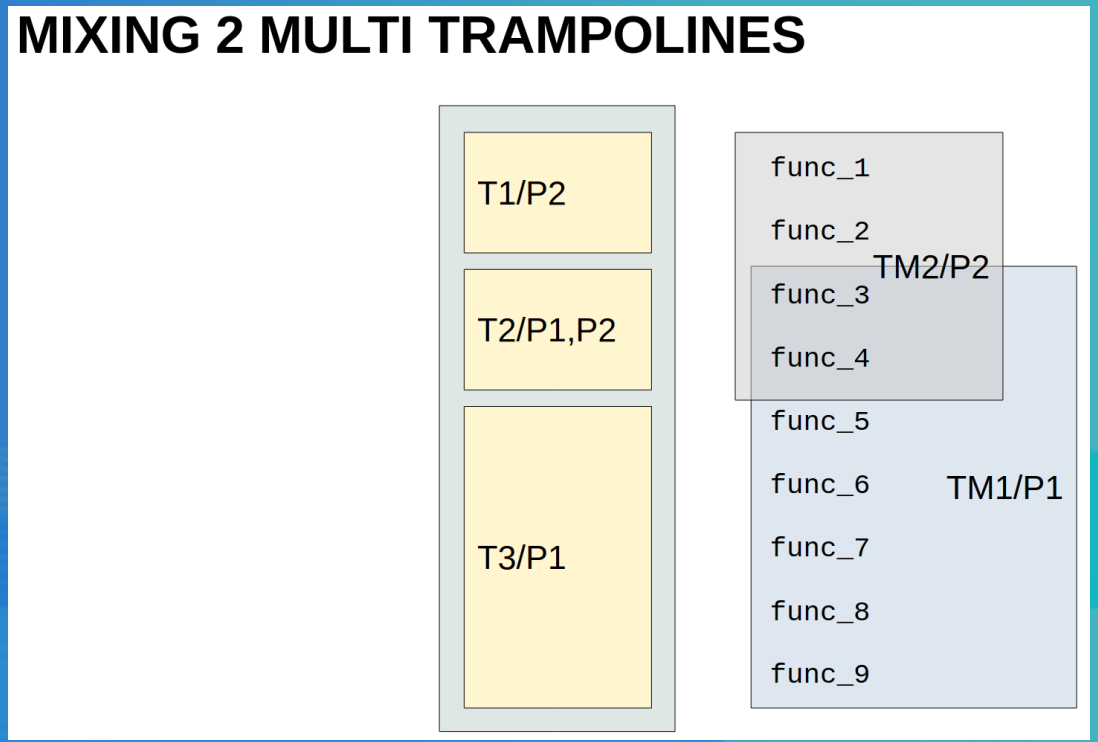
将每个函数上面挂载的BPF信息，以函数地址为key，保存到全局哈希表中，通过trampoline_lookup_ip函数进行查找，在global trampoline中调用。

```
struct bpf_array {
  struct bpf_prog
*fentries;
  struct bpf_prog *fexits;
  struct bpf_prog
*modify_returns;
}
```

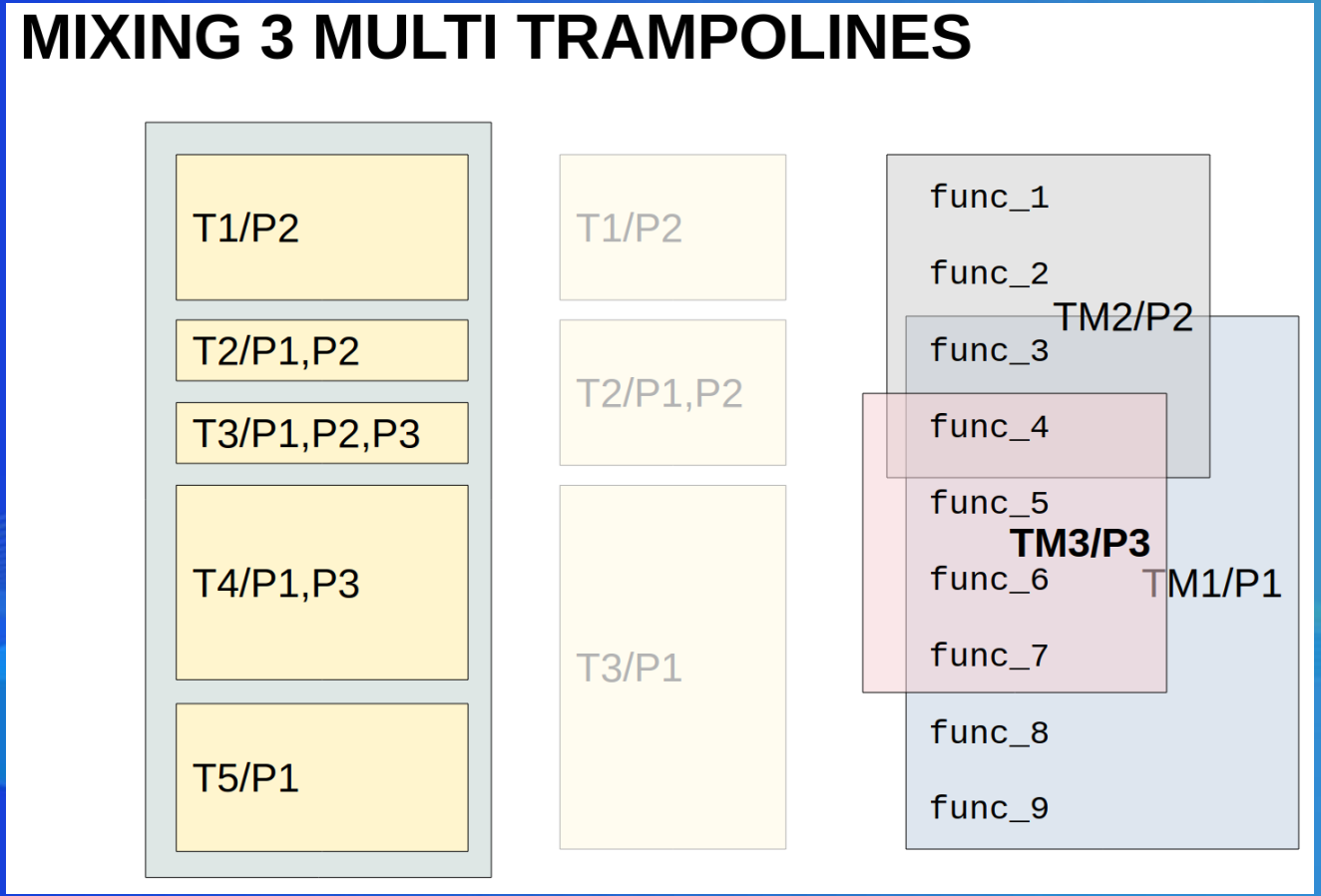
性能太差，不行！！！

③MIXING TRAMPOLINES（Jirka）





③MIXING TRAMPOLINES（Jirka）

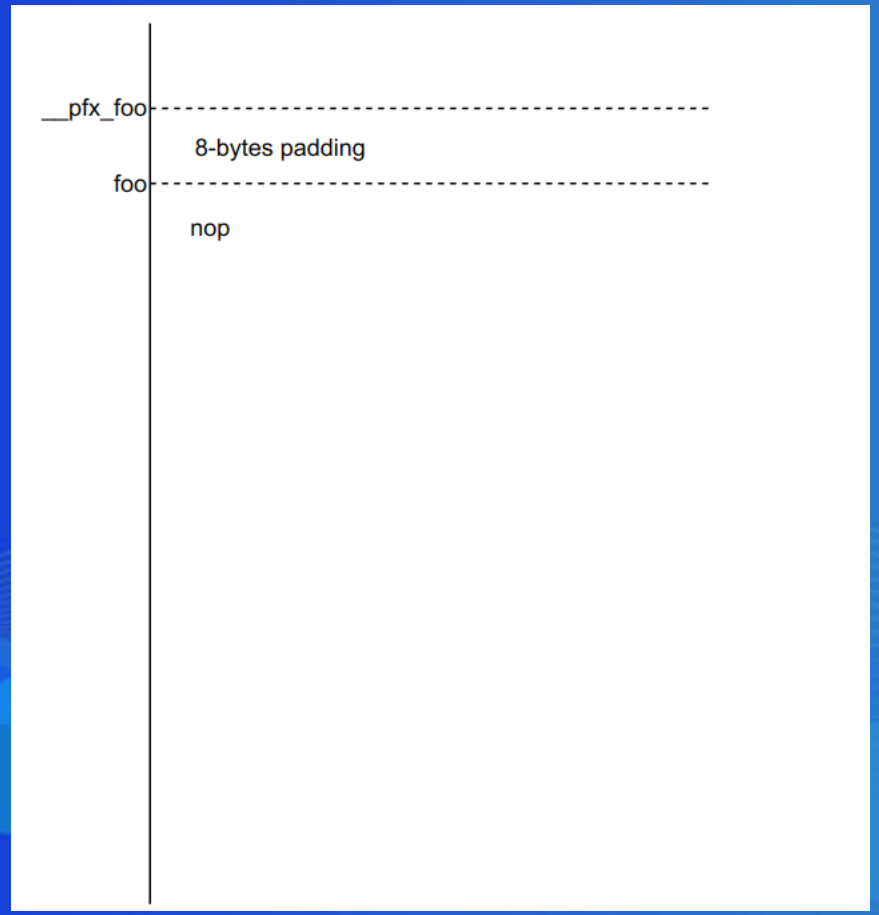


- https://lpc.events/event/16/contributions/1350/attachments/1033/1983/plumbers.pdf
- https://git.kernel.org/pub/scm/linux/kernel/git/jolsa/perf.git/log/?h=bpf/batch
- 29个patch
- 不行，太过复杂

④TRACING MULTI-LINK(V1)

零开销的“per-function storage”？

④function meta data



核心思想：
在每个函数前面预留一定的padding空间（8字节），将我们需要的信息存储到这个padding空间中。

该方式需要编译器支持。所幸，编译器提供了-fpatchable-function-entry参数可以实现这一功能。

零开销：
(struct bpf_array *)(ip - 8)

④function meta data

挑战：

1. 每个函数前面预留padding空间会导致vmlinux的text变大
2. padding空间可能已经被使用了，需要进行适配

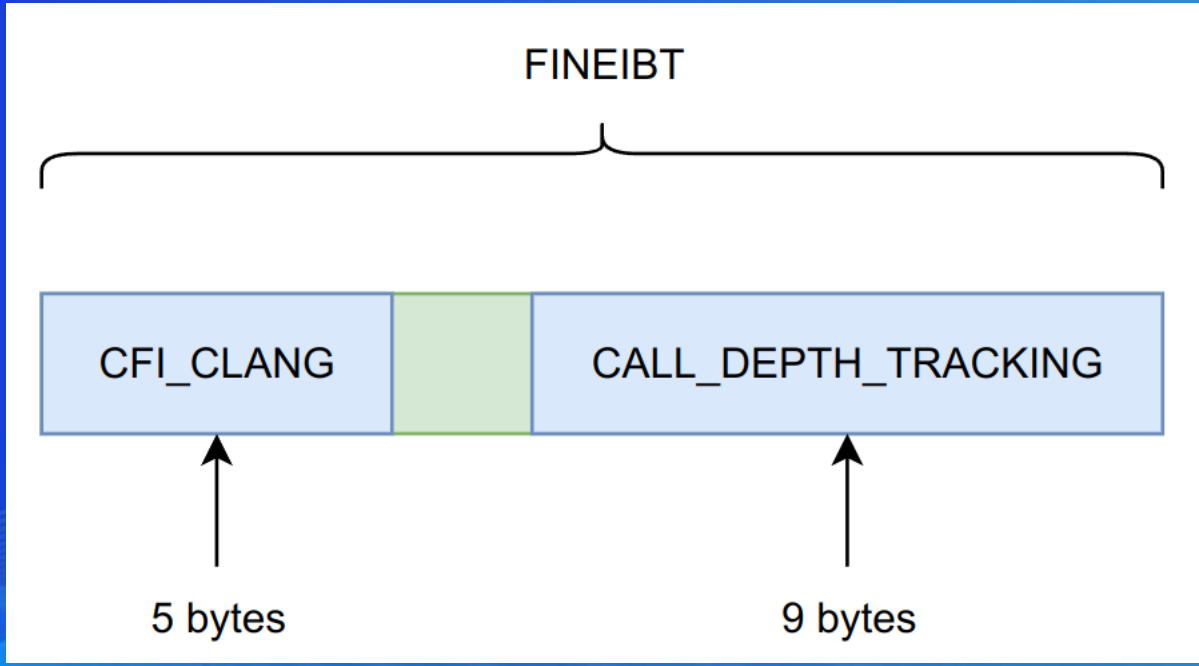
ARM64：

CONFIG_DYNAMIC_FTRACE_WITH_CALL_OPS已经使用了类似的功能，其将callback指针存储到了每个函数的padding space中，适配起来较简单。（padding无法和CFI_CLANG共存）

X86_64：

MITIGATION_CALL_DEPTH_TRACKING & CFI_CLANG & FINEIBT，X86适配起来较为困难

④function meta data



X86_64:

存在可用padding空间的时候，才启用本功能。即：

!CFI_CALNG

|| !CALL_DEPTH_TRACKING

④function meta data

function padding不受支持怎么办?

—— fallback到以hash table的方式来实现

具体function padding的实现:

维护一个全局数组，将meta data的index存储到对应函数的function padding里。

对于x86_64，占用5字节（封装为mov指令）padding空间；对于arm64，占用4字节。

⑤global trampoline

```
SYM_FUNC_START(bpf_global_caller)
        ANNOTATE_NOENDBR

        pushq %rbp
        movq %rsp, %rbp
        subq $STACK_SIZE, %rsp

        tramp_save_regs

        CALL_DEPTH_ACCOUNT

        /* save rbx and r12, which will be used later */
        movq %rbx, RBX_OFFSET(%rbp)
        movq %r12, R12_OFFSET(%rbp)

        /* get the function address */
        movq 8(%rbp), %rdi
SYM_INNER_LABEL(bpf_global_caller_real_func, SYM_L_GLOBAL)
        ANNOTATE_NOENDBR
        /* subq $0, %rdi */
        andq $0xfffffffffffffff0, %rdi
```

⑤global trampoline

```
                /* save the function ip */
                movq %rdi, FUNC_ORIGIN_IP(%rbp)

SYM_INNER_LABEL(bpf_global_caller_kfunc_md, SYM_L_GLOBAL)
                ANNOTATE_NOENDBR
                call kfunc_md_find
                testq %rax, %rax
                jz do_bpf_out
                movq %rax, KFUNC_MD_OFFSET(%rbp)

                /* fentry bpf progs */
                movq (KFUNC_MD_PROGS)(%rax), %rbx
                bpf_caller_prog_run fentry

                movq KFUNC_MD_OFFSET(%rbp), %r12
                /* load the modify_return prog list to rbx */
                movq (KFUNC_MD_PROGS + 16)(%r12), %rbx
                bpf_caller_prog_run modify_return 1
                ....................................
```

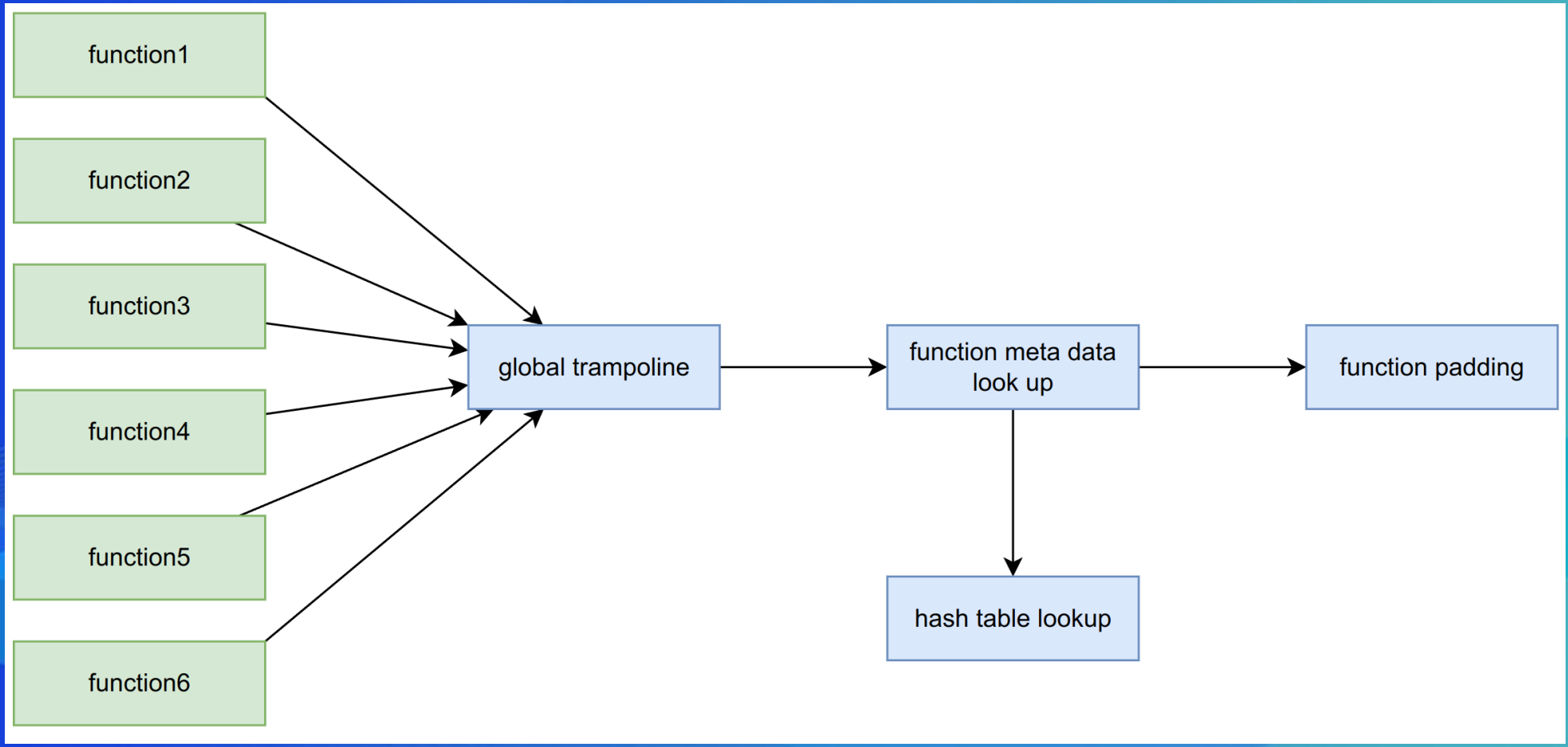
⑤global trampoline

缺陷：indirect call

当产生到indirect call的时候，会涉及到很多东西，最大的问题在于MITIGATION机制而导致的性能问题。初步测试，indirect call会使得global trampoline的overhead相比于普通的trampoline高50%左右。（不知道为啥，Alexei说indirect call不是个问题，那就先不管了~）

即使这样，global trampoline的性能依然比kprobe_multi高出一个数量级（kprobe_multi是基于哈希表的方式来实现ops查找的）。

⑥tracing multi-link



⑥tracing multi-link

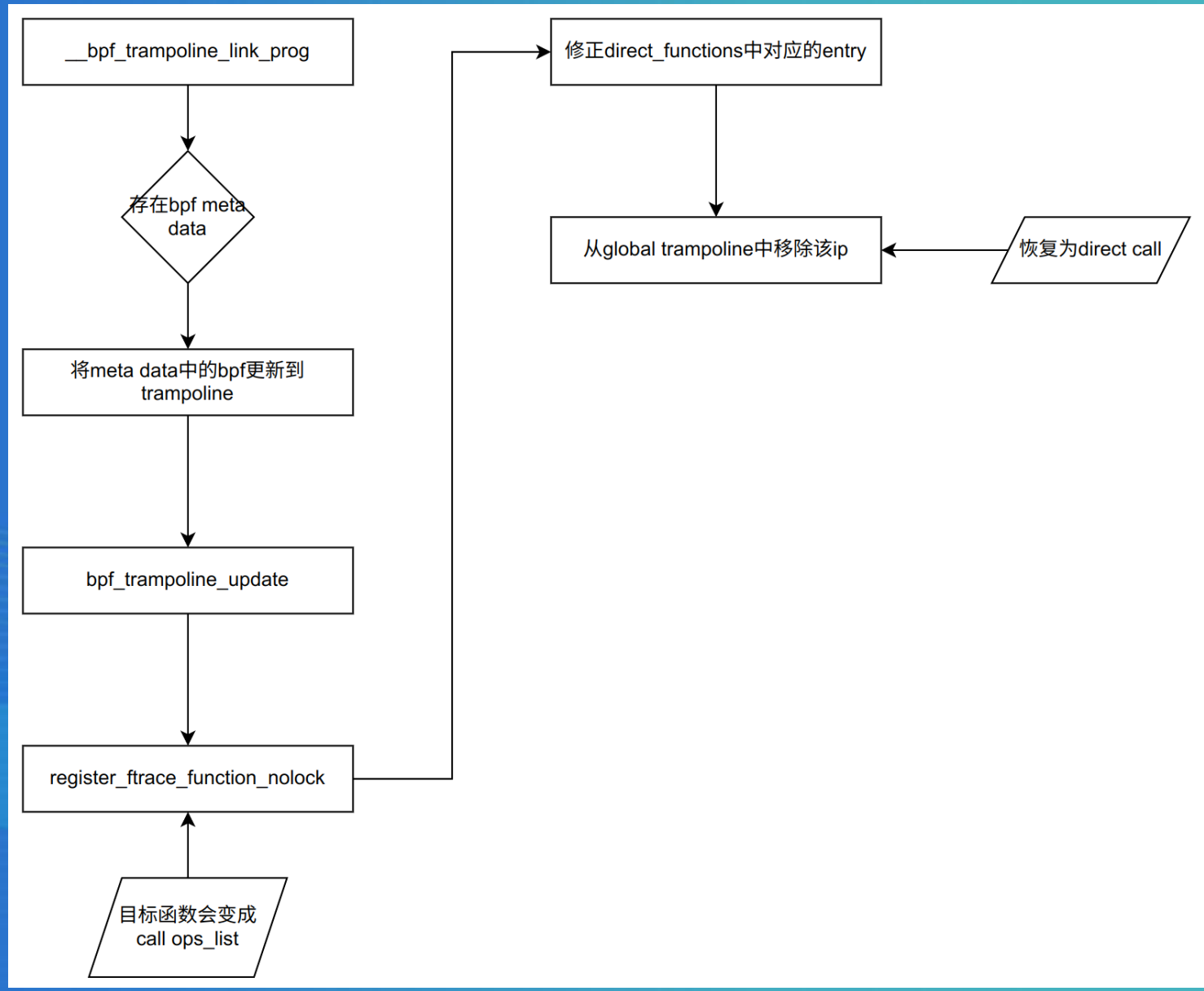
兼容性：

attach trampoline -> 已经存在global trampoline -> 将function metadata中的BPF更新到trampoline，使用trampoline替换global trampoline

attach global trampoline -> 已经存在trampoline -> 使用当前BPF程序更新trampoline

⑥tracing multi-link

trampoline替换过程：



⑥tracing multi-link

当前进展：

完成度80%（目前仅支持x86），代码量新增3500行

https://lore.kernel.org/netdev/20250303065345.229298-1-dongml2@chinatelecom.cn/