BPF programming hands-on tutorials

Performance Analysis in Linux Storage Stack with BPF

Taeung Song
KossLab

Daniel T. Lee
The University of Soongsil

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

What is BPF ?

What is BPF ?

“BPF in the kernel is similar with V8 in Chrome browser”

What is BPF ?

“Linux kernel **code execution** engine”

What is BPF ?

“Run **code** in the kernel”

What is BPF ?

“Run **code** in the kernel”

```
$ readelf -h bpf-prog.o  | grep Machine
  Machine:    Linux BPF
```

What is BPF ?

“Run **code** in the kernel”

```
$ readelf -h bpf-prog.o  | grep Machine
  Machine:    Linux BPF
  Machine:    Advanced Micro Devices X86-64
```

What is BPF ?

“Run **code** in the kernel”

↓

```
$ readelf -h bpf-prog.o  | grep Machine
  Machine:    Linux BPF
  Machine:    Advanced Micro Devices X86-64
```

writing C program

↓

clang / llc

↓

BPF instruction

What is BPF ?

“Run **code** in the kernel”

↓

```
$ readelf -h bpf-prog.o  | grep Machine
  Machine:     Linux BPF
  Machine:     Advanced Micro Devices X86-64
```

writing C program

but, restricted

↓

clang / llc

↓

BPF instruction

What is BPF ?

“Run BPF **code** in the kernel”

```
$ readelf -h bpf-prog.o  | grep Machine
  Machine:     Linux BPF
  Machine:     Advanced Micro Devices X86-64
```

writing C program

but, restricted

clang / llc

BPF instruction

Is it **safe** ?

the image from http://www.tunisiaonline.com/travel-to-tunisia-in-2017/

Is it **safe** ?

“BPF verifier(in-kernel) guarantees”

Is it safe ?

“BPF verifier(in-kernel) guarantees”

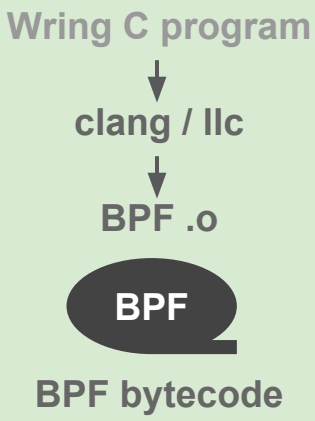
Check BPF program & Prevent problems
before the injection

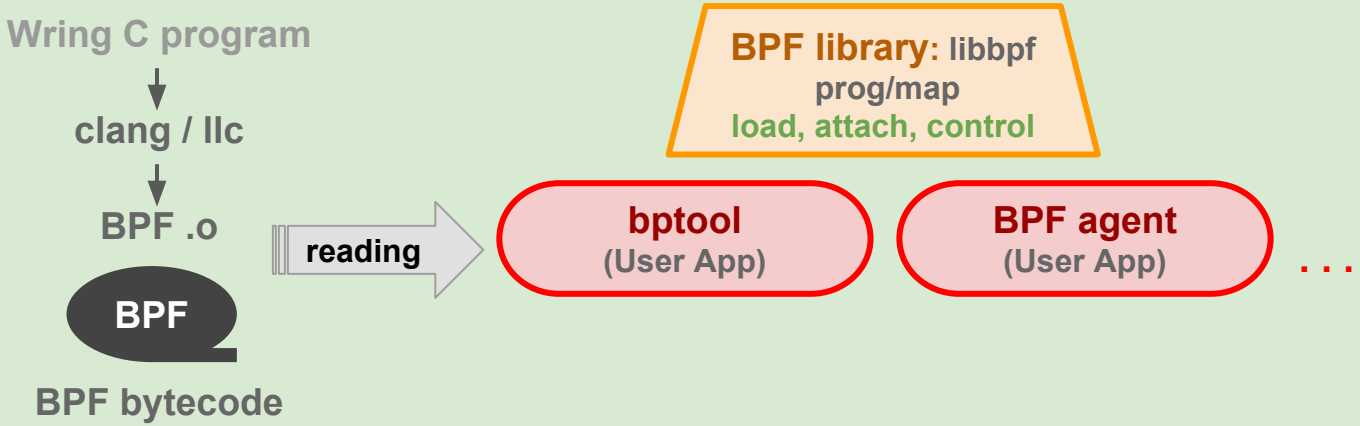
How to execute BPF program ?

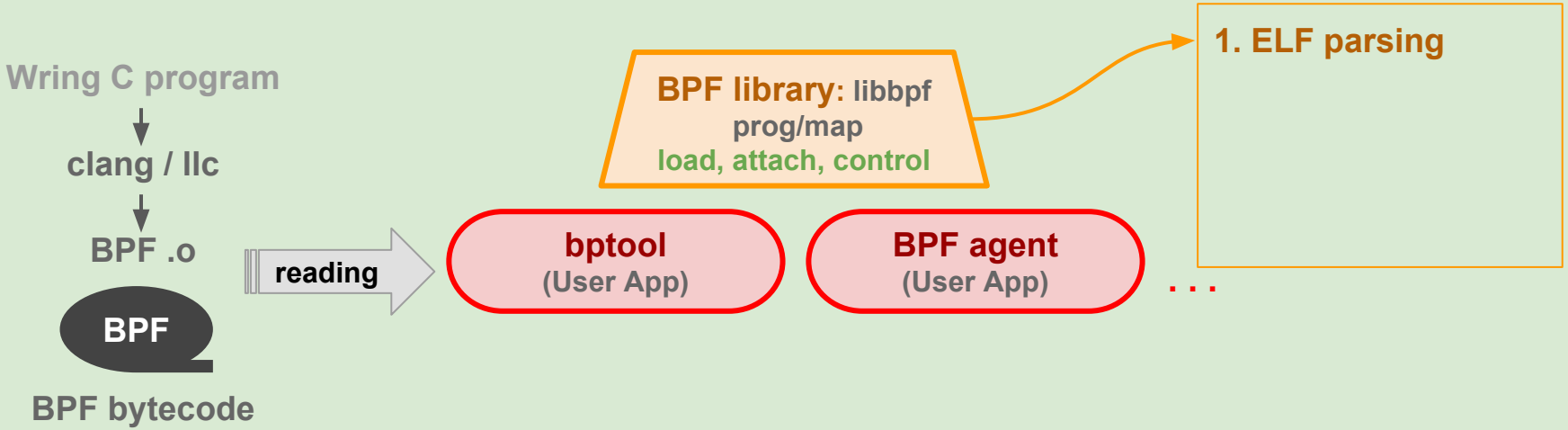
LOAD

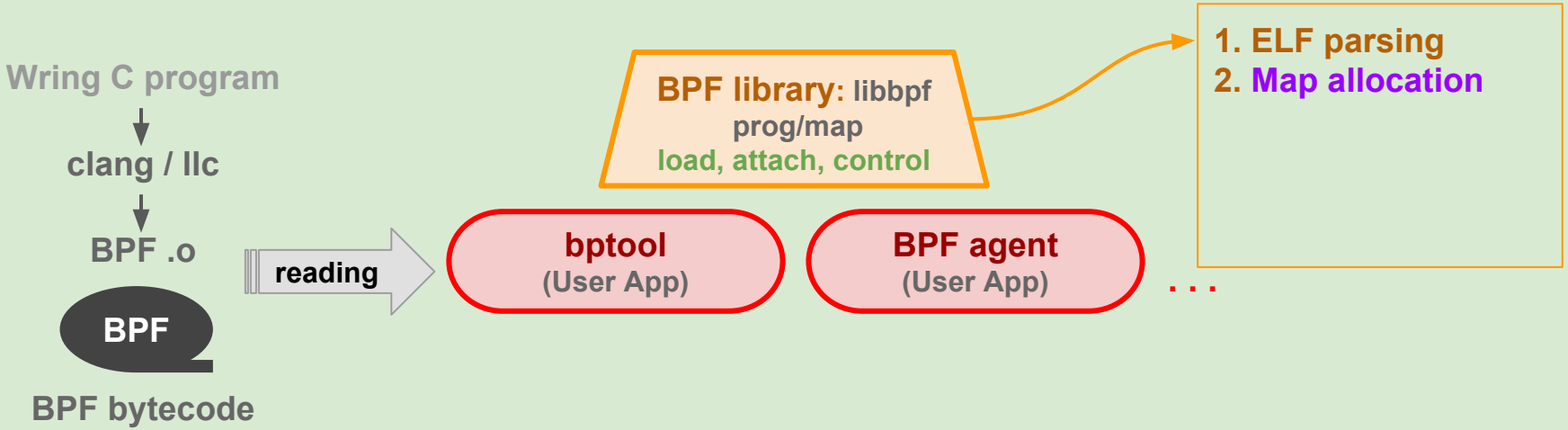
ATTACH

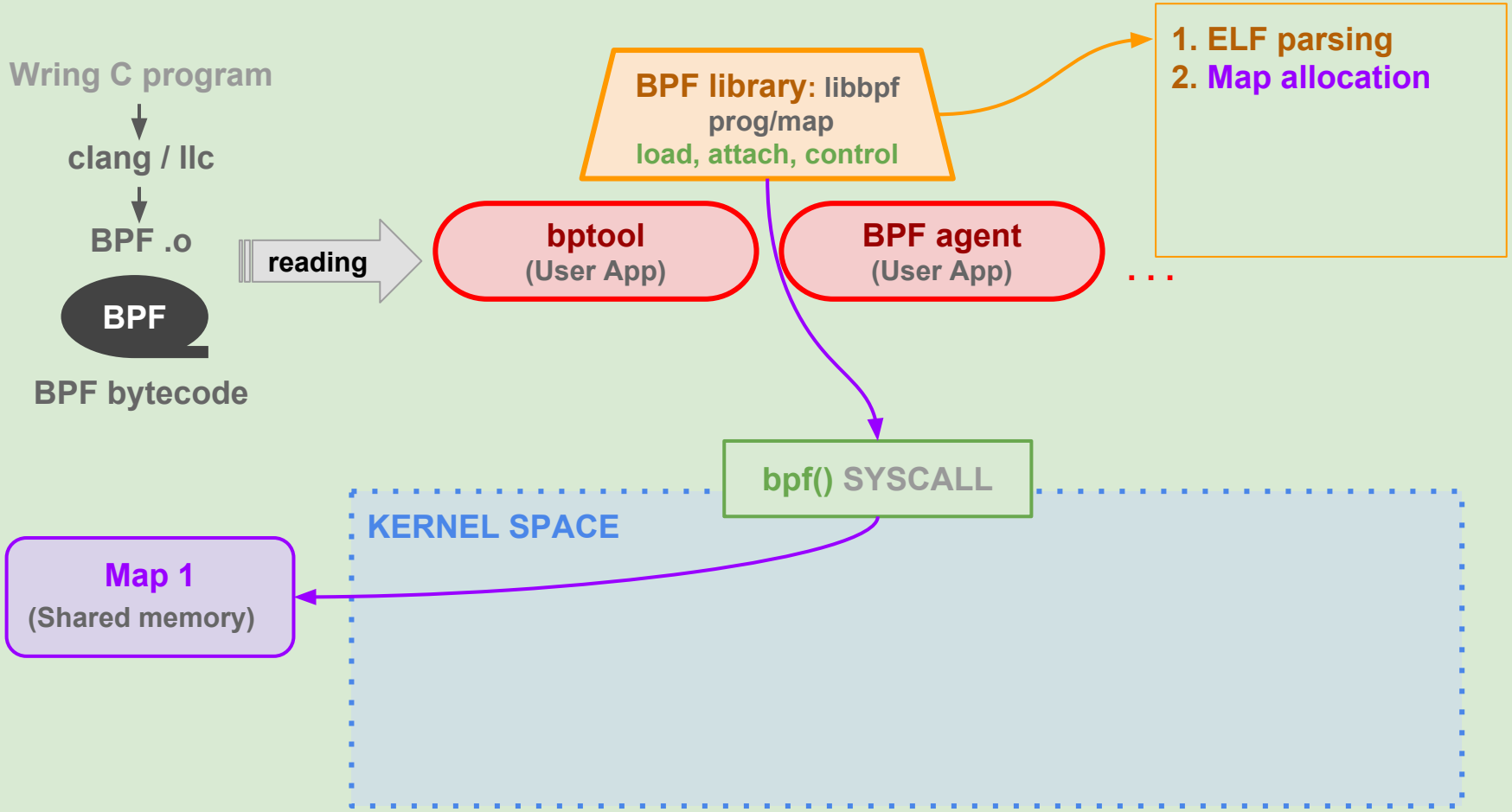
CALLBACK

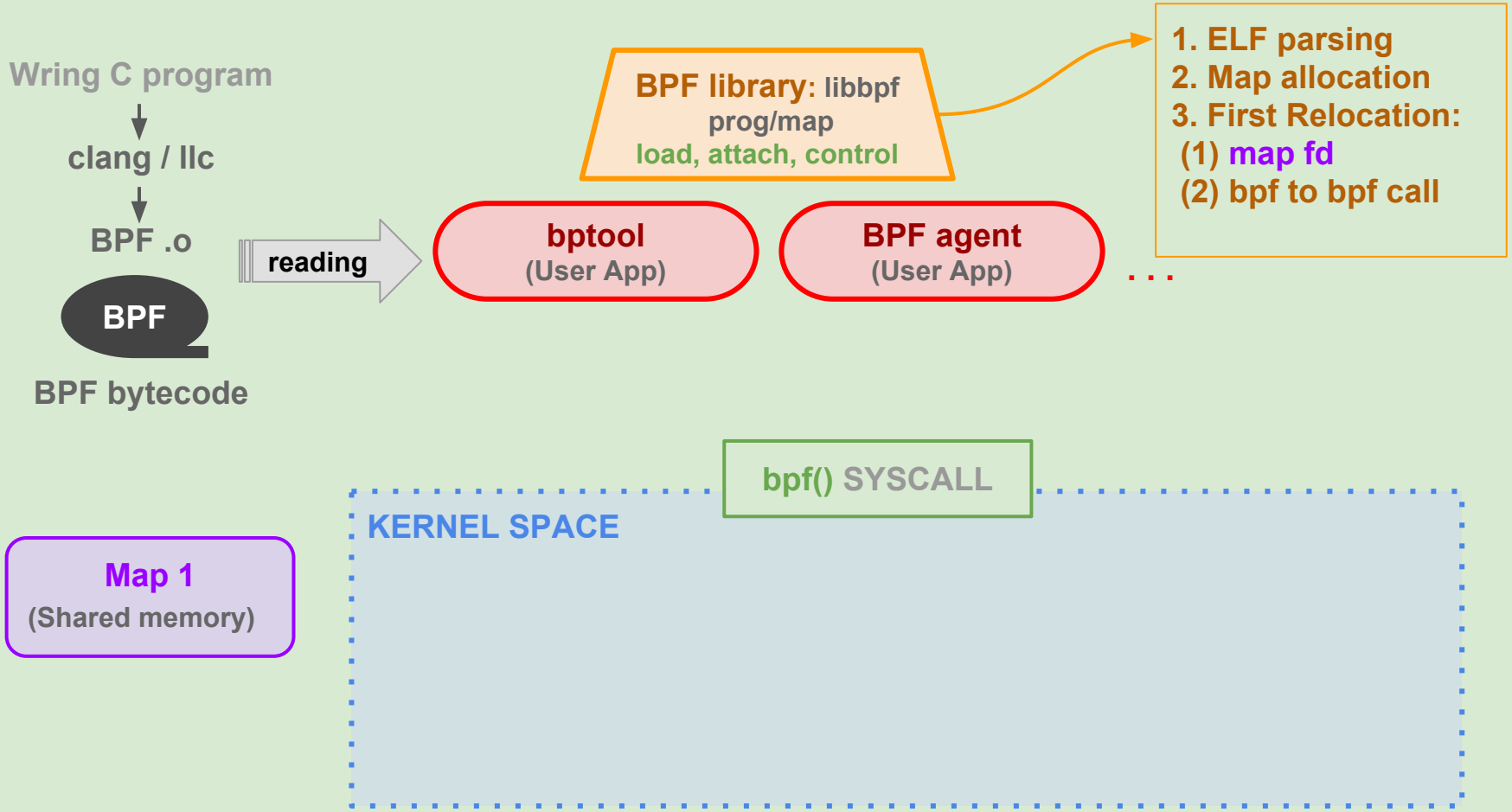

Wring C program
clang / llc
BPF .o
BPF
BPF bytecode

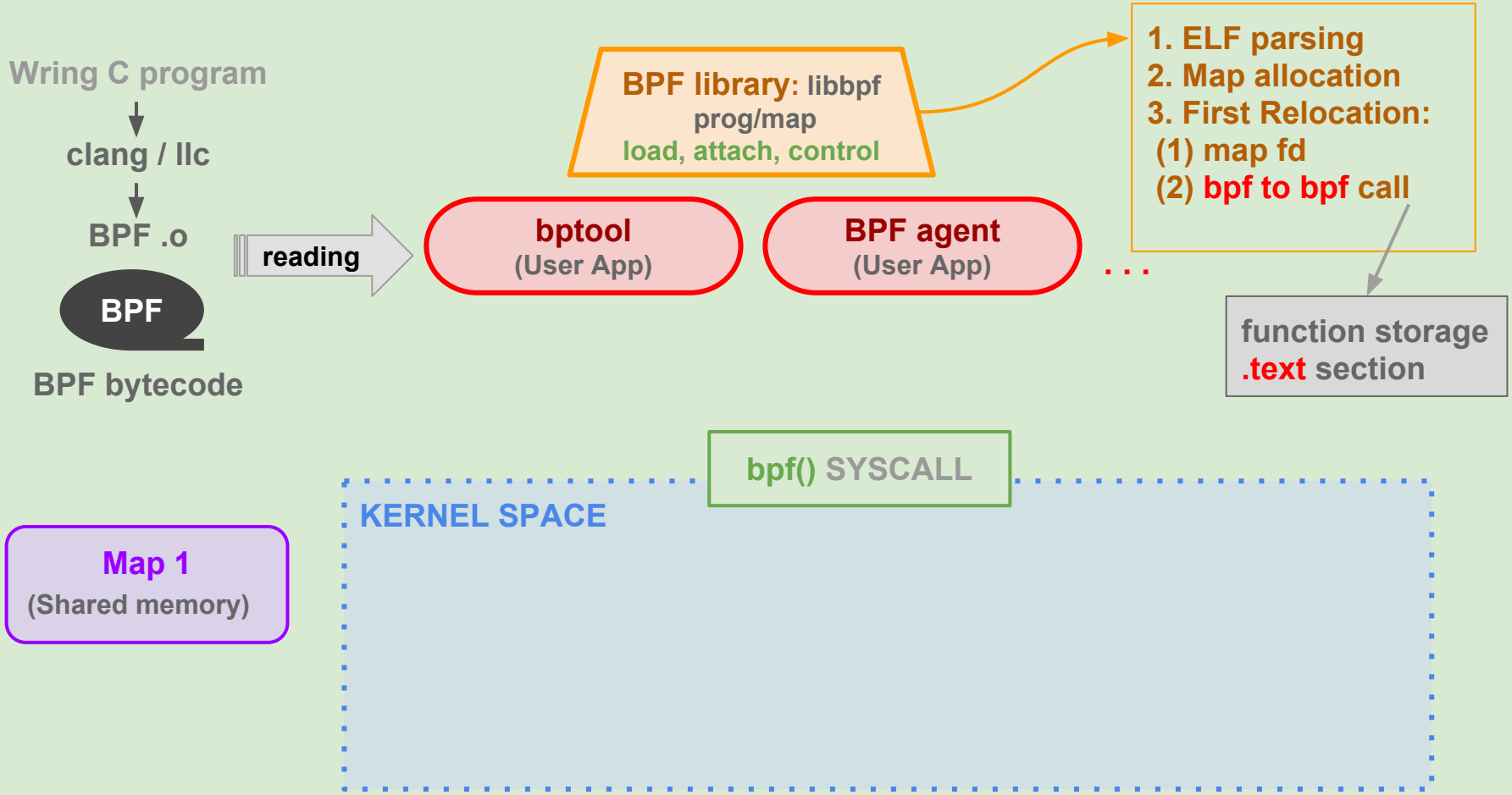

Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
BPF library: libbpf
prog/map
load, attach, control
bptool
(User App)
BPF agent
(User App)
. . .

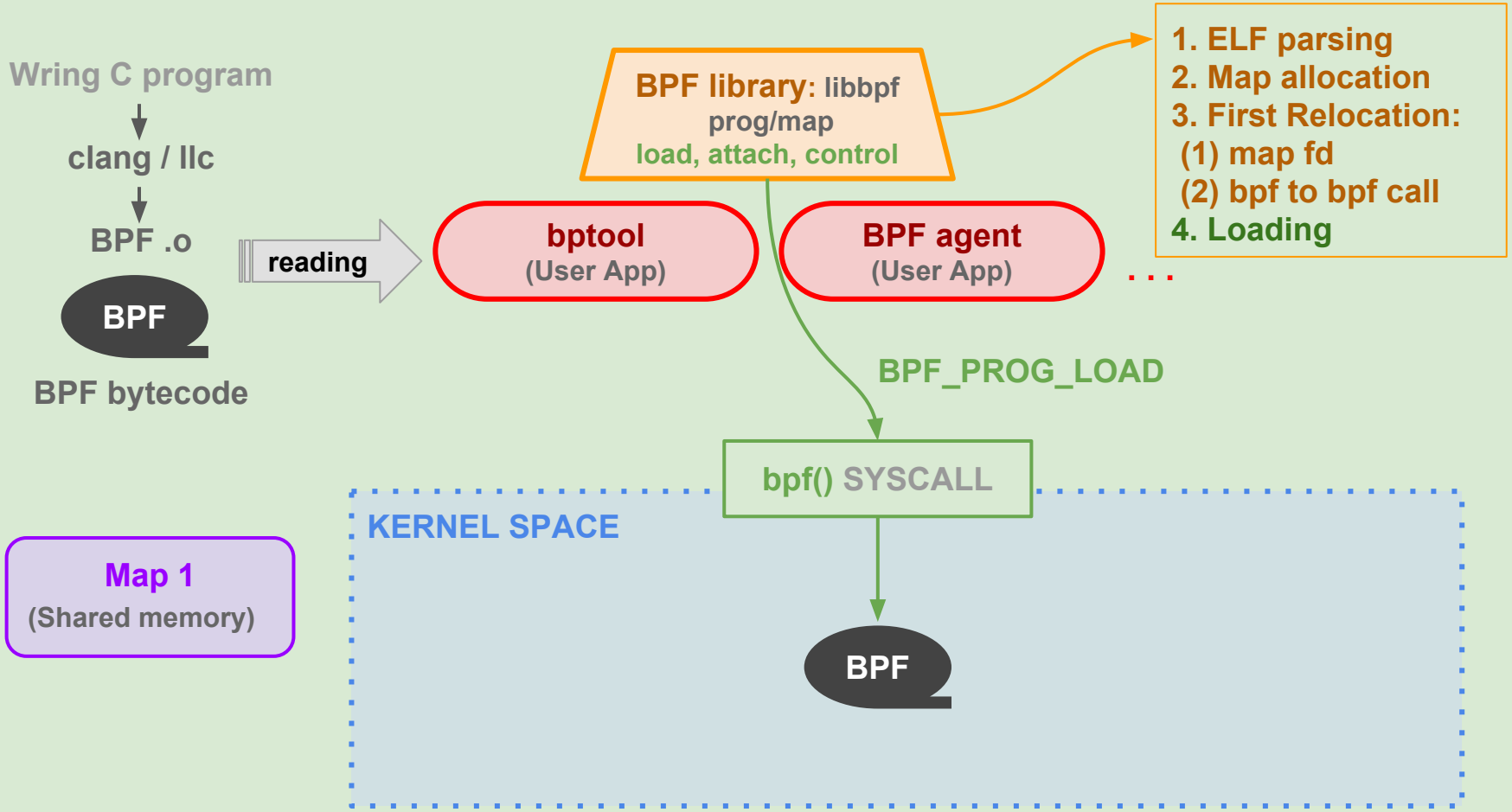

Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
bptool
(User App)
BPF agent
(User App)
. . .
BPF library: libbpf
prog/map
load, attach, control
1. ELF parsing

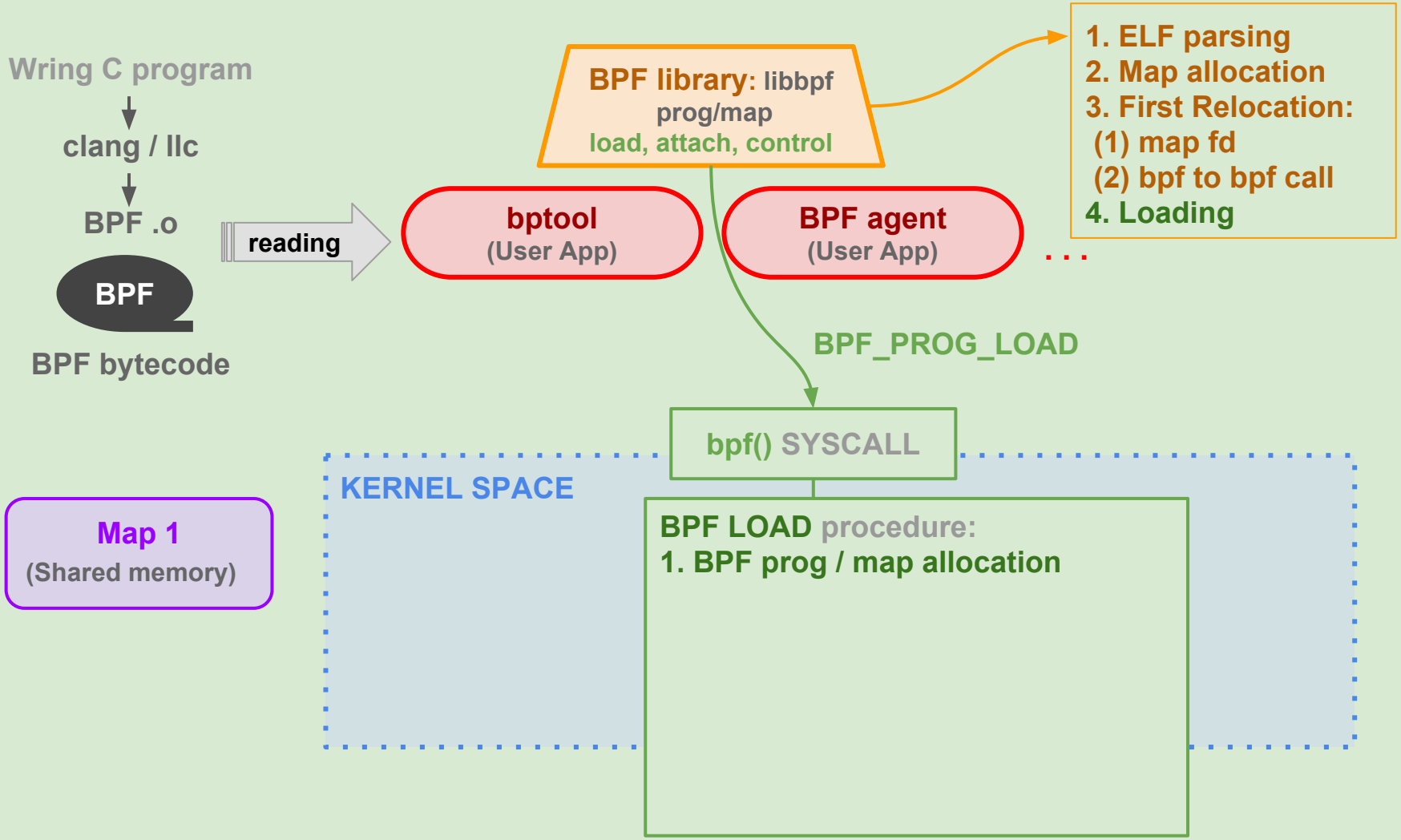

Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
bptool
(User App)
BPF agent
(User App)
. . .
BPF library: libbpf
prog/map
load, attach, control
1. ELF parsing
2. Map allocation

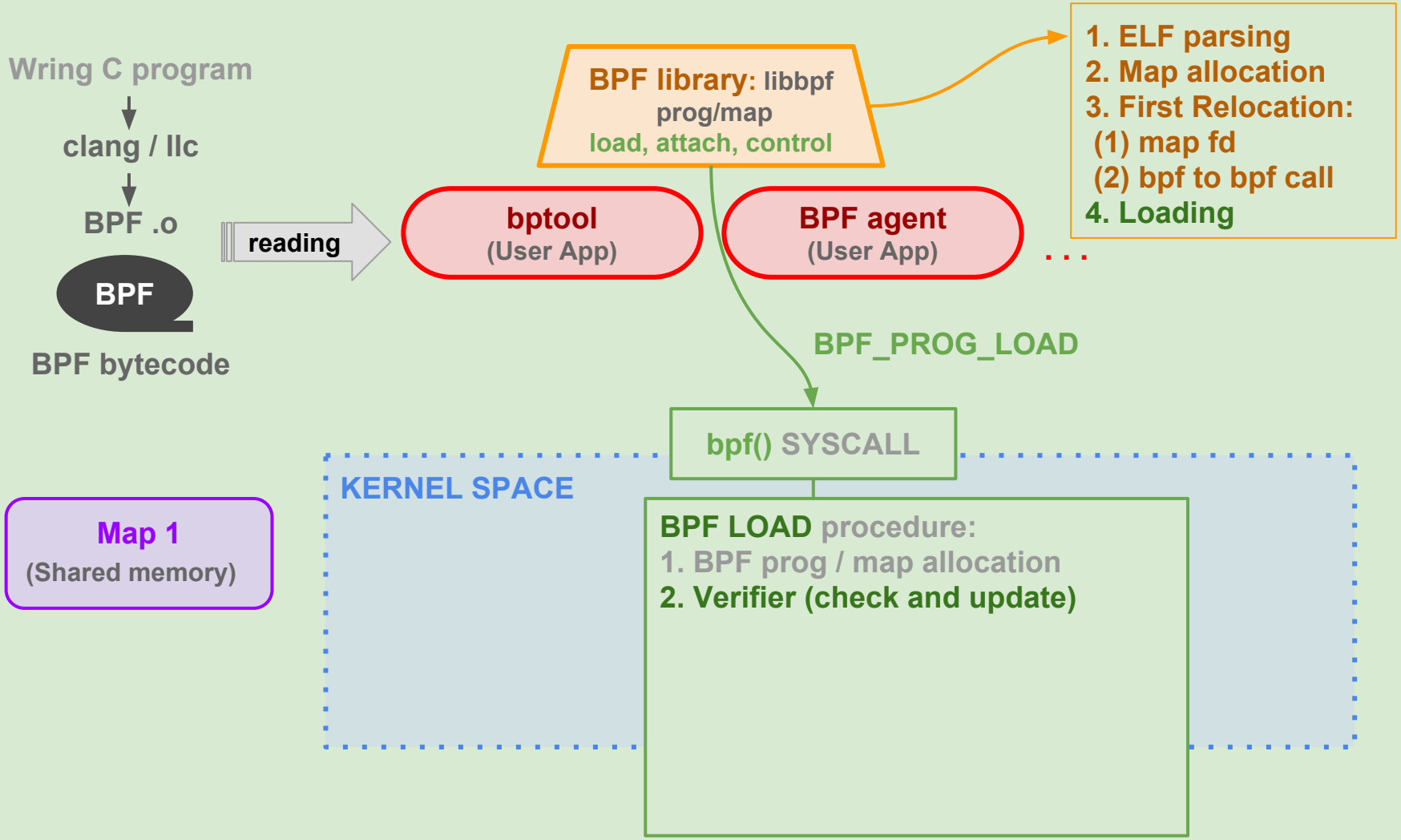

Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
bptool
(User App)
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
. . .
1. ELF parsing
2. Map allocation
bpf() SYSCALL
KERNEL SPACE
Map 1
(Shared memory)

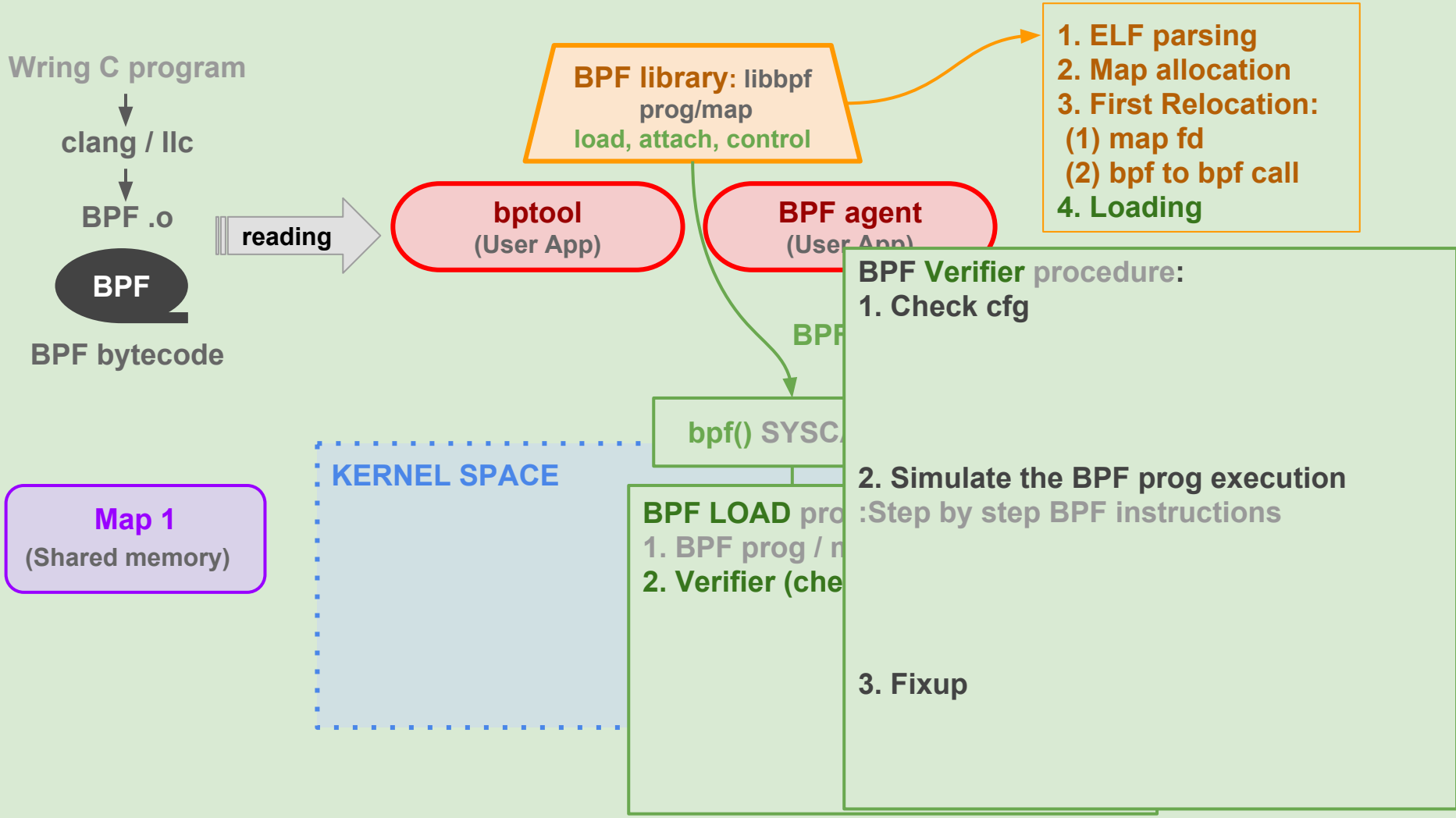

Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
bptool
(User App)
BPF agent
(User App)
. . .
BPF library: libbpf
prog/map
load, attach, control
1. ELF parsing
2. Map allocation
3. First Relocation:
(1) map fd
(2) bpf to bpf call
bpf() SYSCALL
KERNEL SPACE
Map 1
(Shared memory)

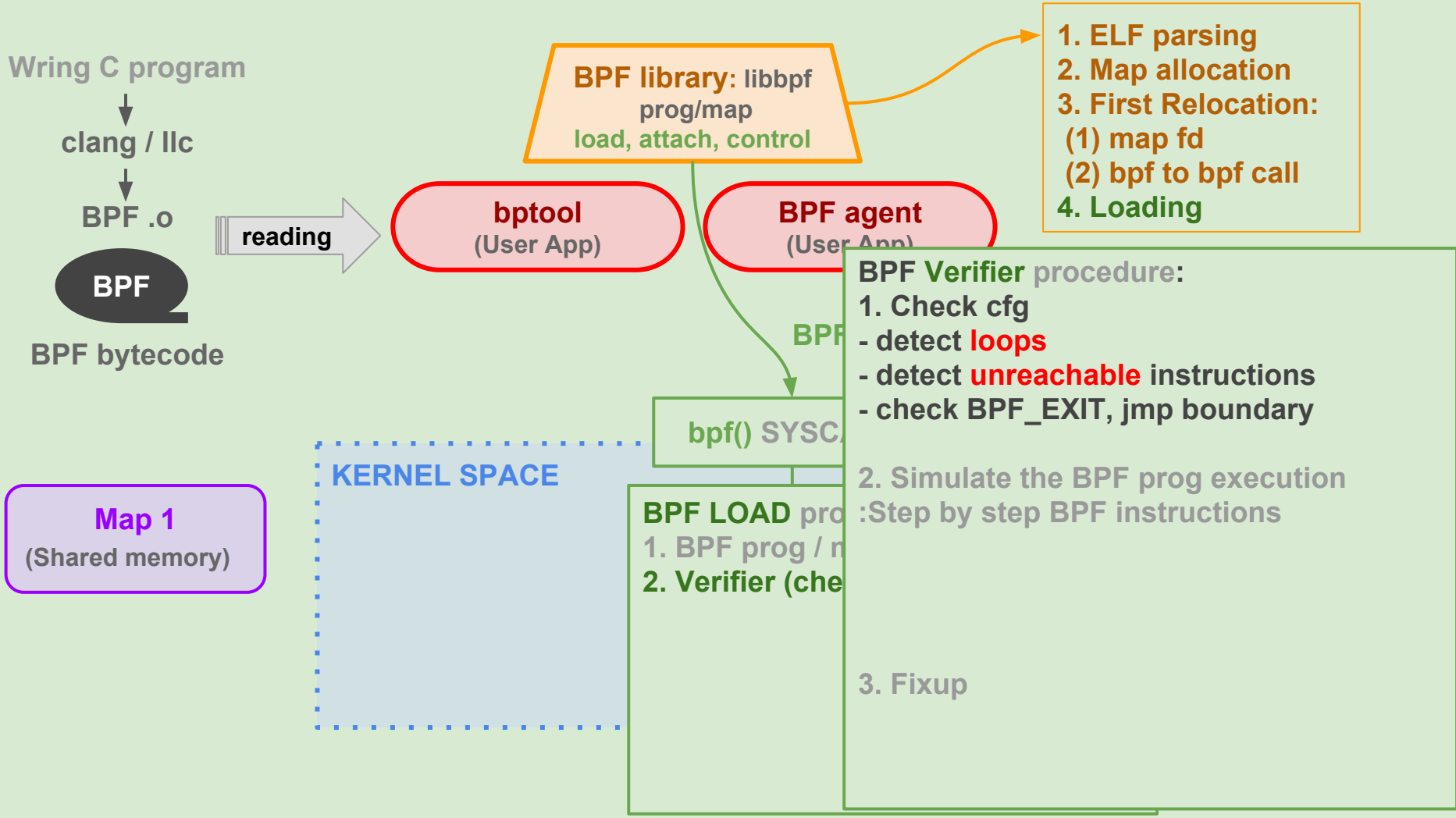

Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
bptool
(User App)
BPF agent
(User App)
. . .
BPF library: libbpf
prog/map
load, attach, control
1. ELF parsing
2. Map allocation
3. First Relocation:
(1) map fd
(2) bpf to bpf call
function storage
.text section
bpf() SYSCALL
KERNEL SPACE
Map 1
(Shared memory)

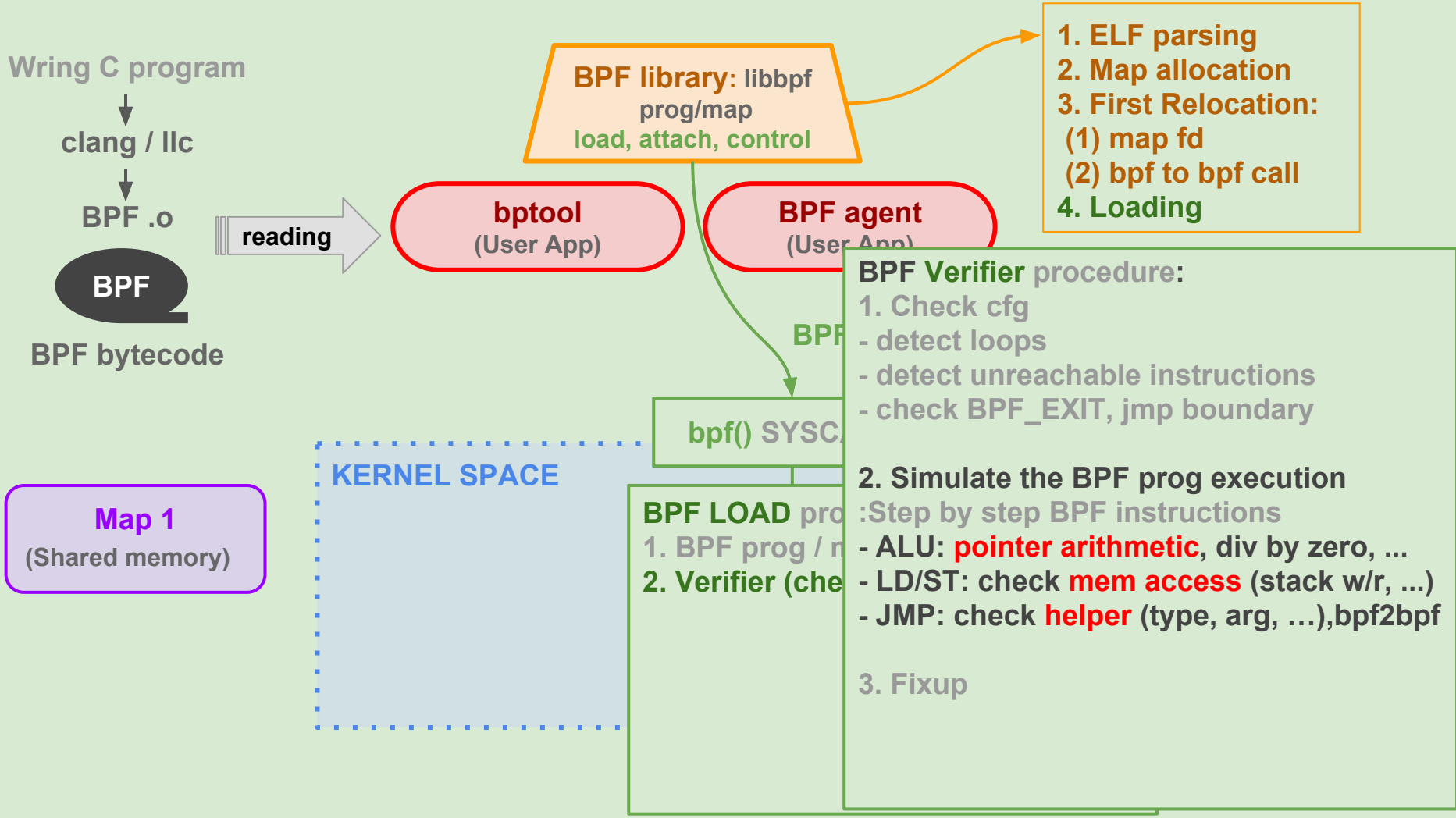

Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
bptool
(User App)
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
. . .
1. ELF parsing
2. Map allocation
3. First Relocation:
(1) map fd
(2) bpf to bpf call
4. Loading
BPF_PROG_LOAD
bpf() SYSCALL
KERNEL SPACE
BPF
Map 1
(Shared memory)

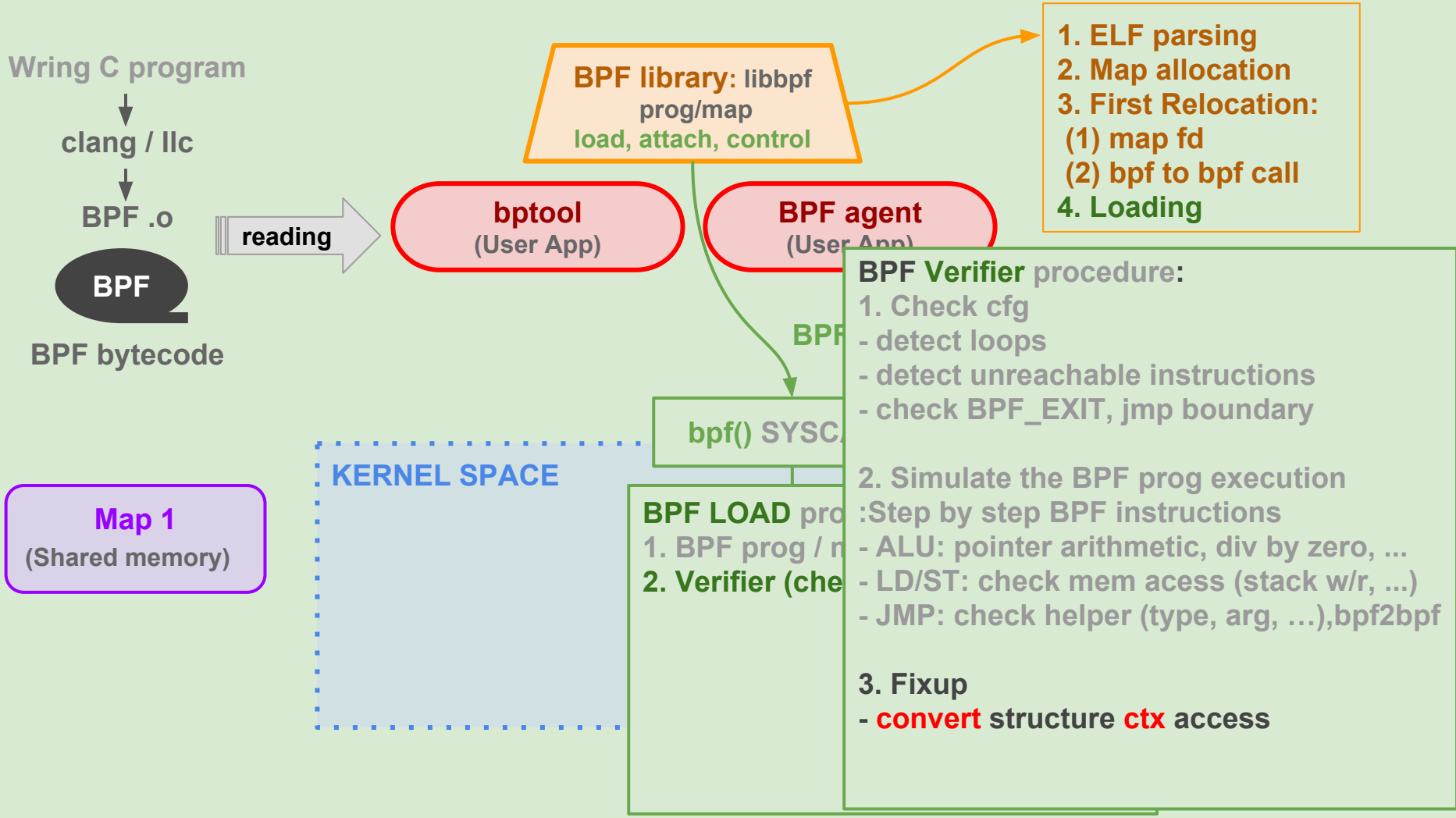

Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
bptool
(User App)
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
. . .
1. ELF parsing
2. Map allocation
3. First Relocation:
(1) map fd
(2) bpf to bpf call
4. Loading
BPF_PROG_LOAD
bpf() SYSCALL
KERNEL SPACE
BPF LOAD procedure:
1. BPF prog / map allocation
Map 1
(Shared memory)

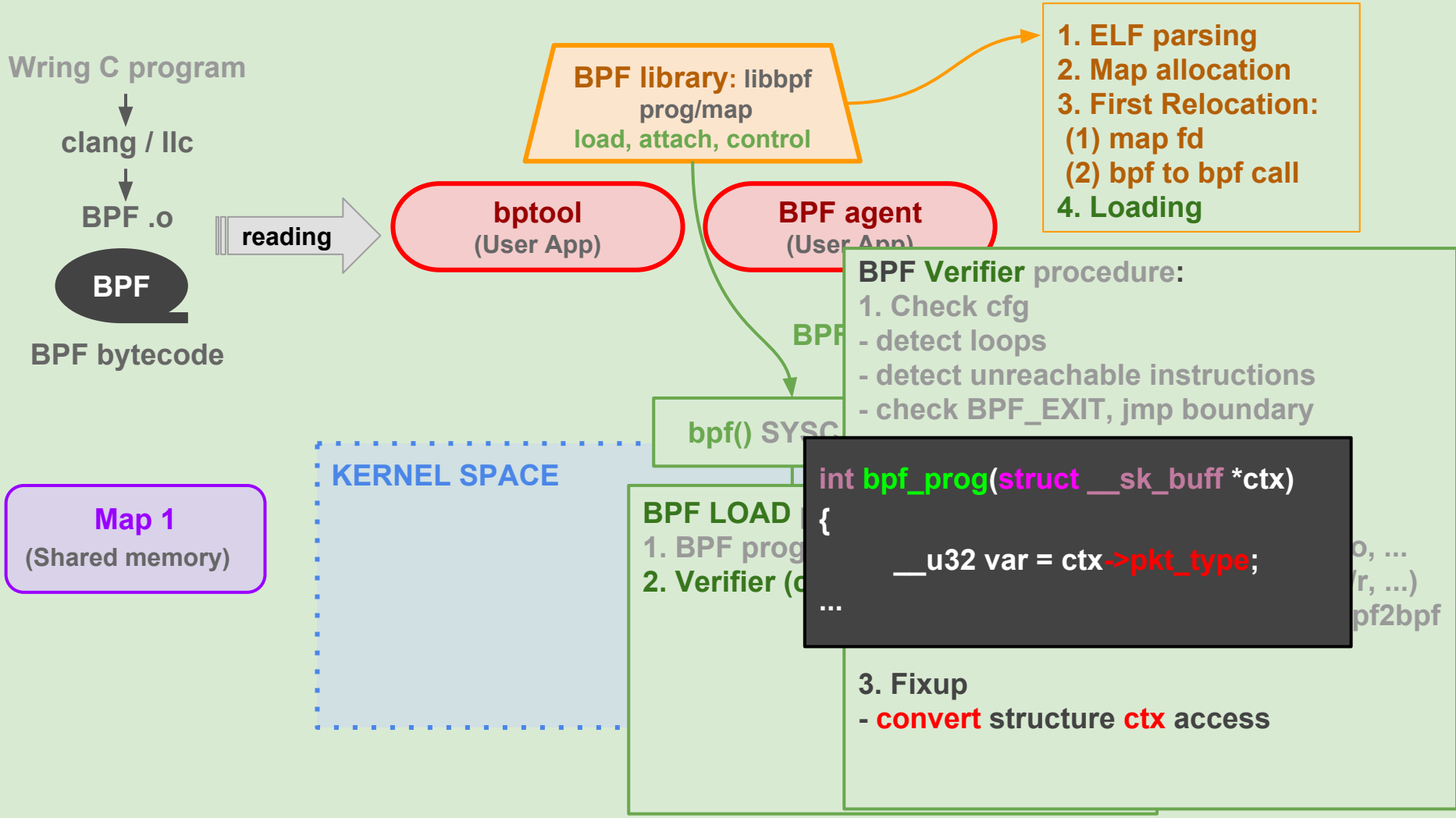

Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
bptool
(User App)
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
. . .
1. ELF parsing
2. Map allocation
3. First Relocation:
(1) map fd
(2) bpf to bpf call
4. Loading
BPF_PROG_LOAD
bpf() SYSCALL
KERNEL SPACE
BPF LOAD procedure:
1. BPF prog / map allocation
2. Verifier (check and update)
Map 1
(Shared memory)

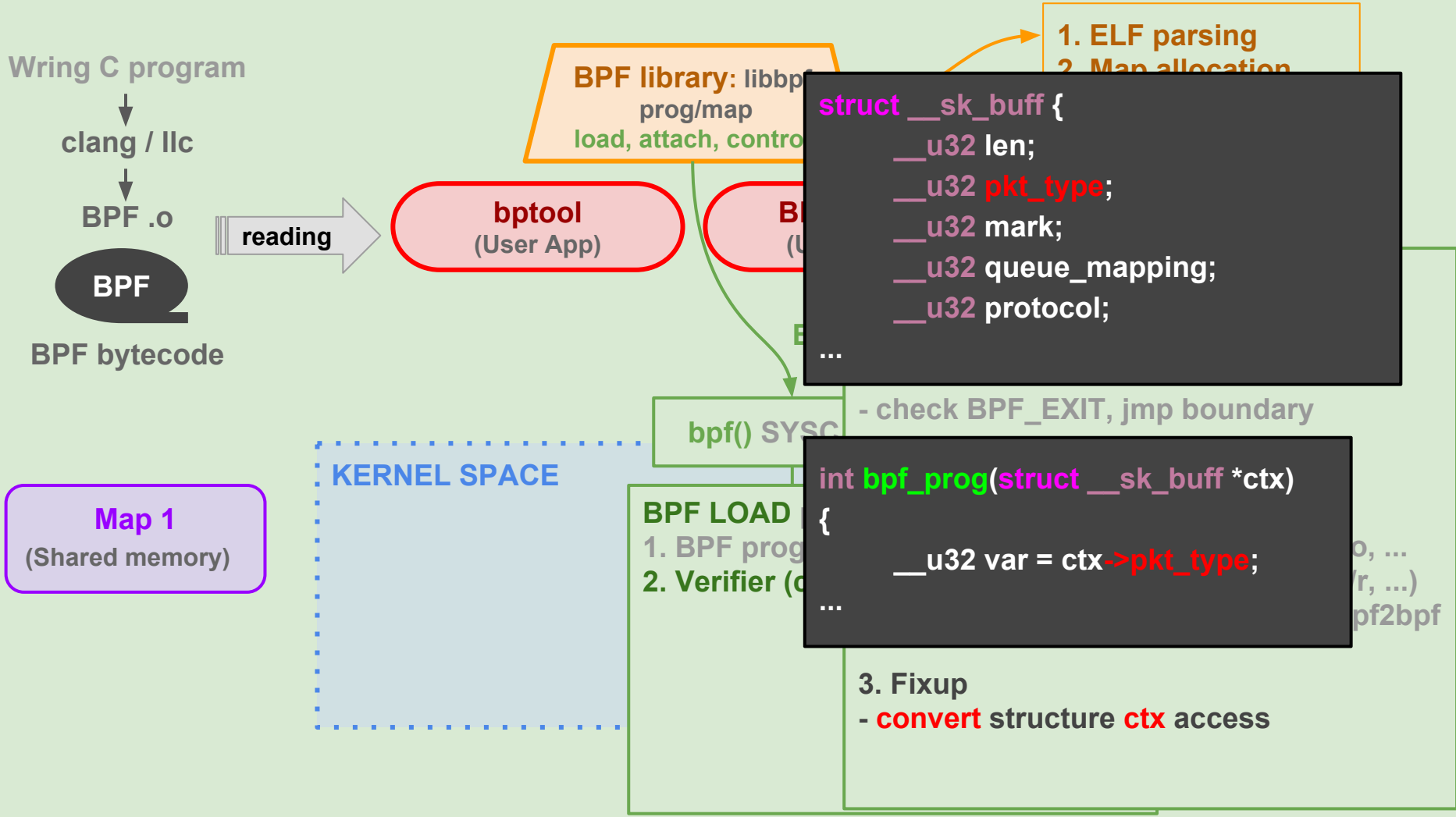

Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
bptool
(User App)
BPF library: libbpf
prog/map
load, attach, control
1. ELF parsing
2. Map allocation
3. First Relocation:
(1) map fd
(2) bpf to bpf call
4. Loading
BPF agent
(User App)
BPF Verifier procedure:
1. Check cfg
2. Simulate the BPF prog execution
:Step by step BPF instructions
3. Fixup
BPF
bpf() SYSC
KERNEL SPACE
Map 1
(Shared memory)
BPF LOAD pro
1. BPF prog / m
2. Verifier (che


Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
bptool
(User App)
BPF library: libbpf
prog/map
load, attach, control
1. ELF parsing
2. Map allocation
3. First Relocation:
(1) map fd
(2) bpf to bpf call
4. Loading
BPF agent
(User App)
BPF Verifier procedure:
1. Check cfg
- detect loops
- detect unreachable instructions
- check BPF_EXIT, jmp boundary
2. Simulate the BPF prog execution
:Step by step BPF instructions
3. Fixup
bpf() SYSC
KERNEL SPACE
Map 1
(Shared memory)
BPF LOAD pro
1. BPF prog / m
2. Verifier (che


Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
bptool
(User App)
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
1. ELF parsing
2. Map allocation
3. First Relocation:
(1) map fd
(2) bpf to bpf call
4. Loading
BPF Verifier procedure:
1. Check cfg
- detect loops
- detect unreachable instructions
- check BPF_EXIT, jmp boundary
2. Simulate the BPF prog execution
:Step by step BPF instructions
- ALU: pointer arithmetic, div by zero, ...
- LD/ST: check mem access (stack w/r, ...)
- JMP: check helper (type, arg, …),bpf2bpf
3. Fixup
KERNEL SPACE
Map 1
(Shared memory)


Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
bptool
(User App)
BPF library: libbpf
prog/map
load, attach, control
1. ELF parsing
2. Map allocation
3. First Relocation:
(1) map fd
(2) bpf to bpf call
4. Loading
BPF agent
(User App)
BPF
bpf() SYSC
KERNEL SPACE
Map 1
(Shared memory)
BPF LOAD pro
1. BPF prog / m
2. Verifier (che
BPF Verifier procedure:
1. Check cfg
- detect loops
- detect unreachable instructions
- check BPF_EXIT, jmp boundary
2. Simulate the BPF prog execution
:Step by step BPF instructions
- ALU: pointer arithmetic, div by zero, ...
- LD/ST: check mem acess (stack w/r, ...)
- JMP: check helper (type, arg, …),bpf2bpf
3. Fixup
- convert structure ctx access


Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
bptool
(User App)
BPF library: libbpf
prog/map
load, attach, control
1. ELF parsing
2. Map allocation
3. First Relocation:
(1) map fd
(2) bpf to bpf call
4. Loading
BPF agent
(User App)
BPF Verifier procedure:
1. Check cfg
- detect loops
- detect unreachable instructions
- check BPF_EXIT, jmp boundary
int bpf_prog(struct __sk_buff *ctx)
{
__u32 var = ctx->pkt_type;
...
3. Fixup
- convert structure ctx access
KERNEL SPACE
Map 1
(Shared memory)
BPF LOAD
1. BPF prog
2. Verifier (c


Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
bptool
(User App)
BPF library: libbp
prog/map
load, attach, contro
1. ELF parsing
2. Map allocation
struct __sk_buff {
__u32 len;
__u32 pkt_type;
__u32 mark;
__u32 queue_mapping;
__u32 protocol;
...
- check BPF_EXIT, jmp boundary
bpf() SYSC
KERNEL SPACE
Map 1
(Shared memory)
BPF LOAD
1. BPF prog
2. Verifier (c
int bpf_prog(struct __sk_buff *ctx)
{
__u32 var = ctx->pkt_type;
...
3. Fixup
- convert structure ctx access

include/linux/skbuff.h:

```
struct sk_buff {
        union {
                struct {
                        /* These two members must be first. */
                        struct sk_buff          *next;
                        struct sk_buff          *prev;

                        union {
                                struct net_device       *dev;
                                /* Some protocols might use this space to store information,
                                 * while device pointer would be NULL.
                                 * UDP receive path is one user.
                                 */
                                unsigned long           dev_scratch;
...
                        __u8                    __pkt_type_offset[0];
                        __u8                    pkt_type:3;
```

1. ELF parsing
2. Map allocation

```
_sk_buff {
u32 len;
u32 pkt_type;
u32 mark;
u32 queue_mapping;
u32 protocol;
```

ck BPF_EXIT, jmp boundary

```
prog(struct __sk_buff *ctx)

u32 var = ctx->pkt_type;
```

o, ...
r, ...)
pf2bpf

3. Fixup
- convert structure ctx access

include/linux/skbuff.h:

```
struct sk_buff {
        union {
                struct {
                        /* These two members must be first. */
                        struct sk_buff          *next;
                        struct sk_buff          *prev;

                        union {
                                struct net_device       *dev;
                                /*
                                 *
                                 *
                                 *
                                u
...
                        __u8
                        __u8
```

```
SEC("kprobe/submit_bio")
int kprobe_type_bpf_prog (struct pt_regs *ctx)
{
        struct bio *bio = (struct bio *)PT_REGS_PARM1(ctx);
...
```

1. ELF parsing
2. Map allocation

```
_sk_buff {
u32 len;
u32 pkt_type;
u32 mark;
u32 queue_mapping;
u32 protocol;
```

boundary

```
uff *ctx)

u32 var = ctx->pkt_type;
```

o, ...
r, ...)
pf2bpf

3. Fixup
- convert structure ctx access

include/linux/skbuff.h:

```
struct sk_buff {
        union {
                struct {
                        /* These two members must be first. */
                        struct sk_buff          *next;
                        struct sk_buff          *prev;

                        union {
                                struct net_device       *dev;
                                /*
                                 *
                                 *
                                 *
                                u
...
                        __u8
                        __u8
```

Various BPF prog types:
BPF_PROG_TYPE_KPROBE,
BPF_PROG_TYPE_TRACEPOINT,
BPF_PROG_TYPE_PERF_EVENT
BPF_PROG_TYPE_XDP,
BPF_PROG_TYPE_SK_SKB
...

```
SEC("kprobe/submit_bio")
int kprobe_type_bpf_prog (struct pt_regs *ctx)
{
        struct bio *bio = (struct bio *)PT_REGS_PARM1(ctx);
...
```

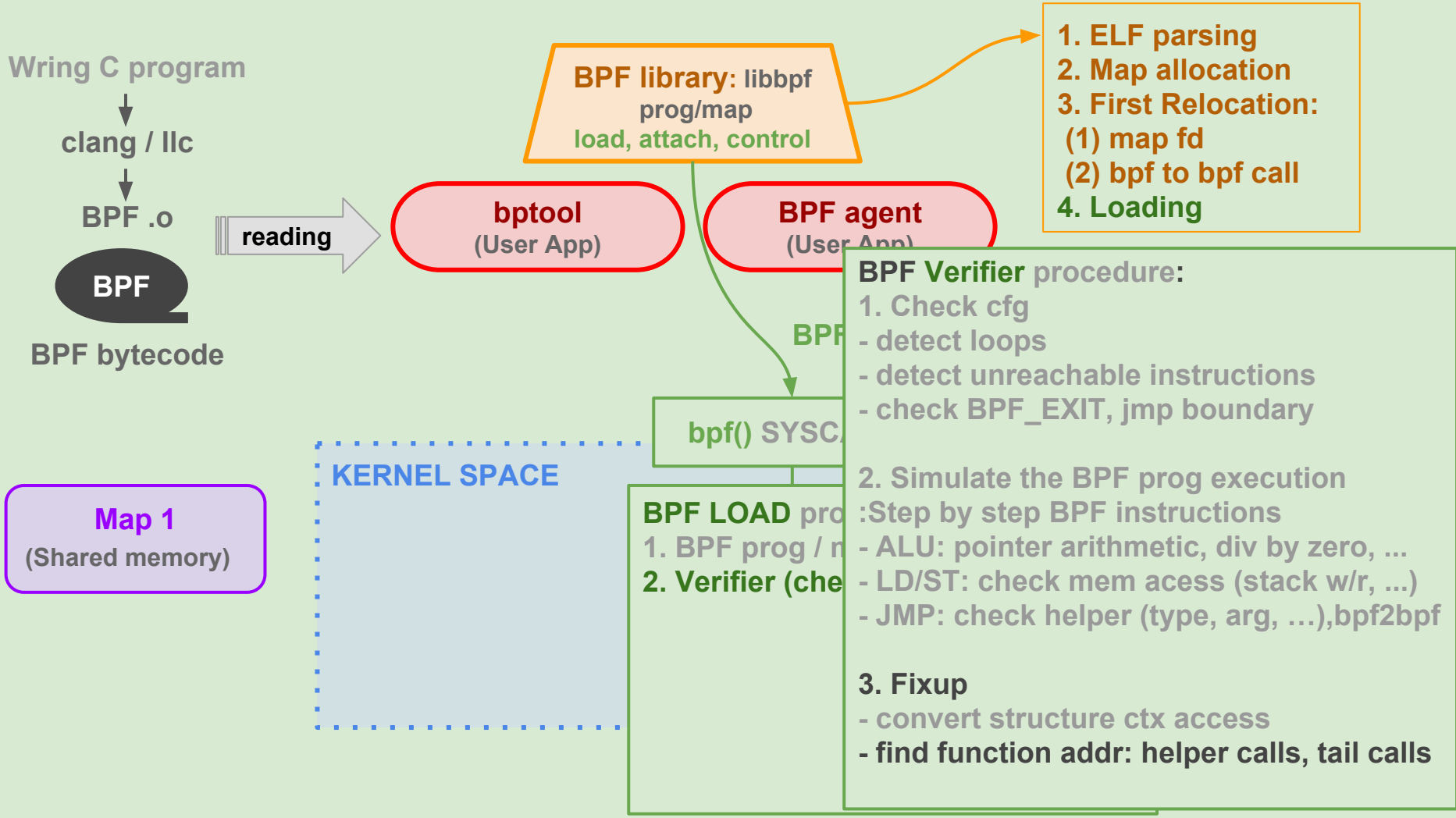
```
u32 protocol;
```

boundary

```
uff *ctx)

u32 var = ctx->pkt_type;
```

o, ...
r, ...)
pf2bpf

3. Fixup
- convert structure ctx access


Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
bptool
(User App)
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
1. ELF parsing
2. Map allocation
3. First Relocation:
(1) map fd
(2) bpf to bpf call
4. Loading
BPF Verifier procedure:
1. Check cfg
- detect loops
- detect unreachable instructions
- check BPF_EXIT, jmp boundary
2. Simulate the BPF prog execution
:Step by step BPF instructions
- ALU: pointer arithmetic, div by zero, ...
- LD/ST: check mem acess (stack w/r, ...)
- JMP: check helper (type, arg, …),bpf2bpf
3. Fixup
- convert structure ctx access
- find function addr: helper calls, tail calls
KERNEL SPACE
Map 1
(Shared memory)

Wring C program

clang / llc

BPF .o

reading

BPF

BPF byte

BPF library: libbpf
prog/map
load, attach, control

bptool
(User App)

BPF agent
(User App)

1. ELF parsing
2. Map allocation
3. First Relocation:
   (1) map fd
   (2) bpf to bpf call
4. Loading

Common Helpers:
bpf_map_lookup_elem()
bpf_map_update_elem()
bpf_map_delete_elem()
bpf_get_smp_processor_id()
bpf_ktime_get_ns()
...

Leveraging kernel functions

Special Helpers:
bpf_trace_printk()
bpf_probe_read()
bpf_perf_event_read()
bpf_xdp_adjust_meta()
bpf_skb_change_head()
...

Map
(Shared me

BPF

SYSC

D pro
og / n
(che

BPF Verifier procedure:

1. Check cfg
- detect loops
- detect unreachable instructions
- check BPF_EXIT, jmp boundary

2. Simulate the BPF prog execution
:Step by step BPF instructions
- ALU: pointer arithmetic, div by zero, ...
- LD/ST: check mem acess (stack w/r, ...)
- JMP: check helper (type, arg, …),bpf2bpf

3. Fixup
- convert structure ctx access
- find function addr: helper calls, tail calls

Wring C program

clang / llc

BPF .o

reading

bptool
(User App)

BPF agent
(User App)

BPF library: libbpf
prog/map
load, attach, control

1. ELF parsing
2. Map allocation
3. First Relocation:
   (1) map fd
   (2) bpf to bpf call
4. Loading

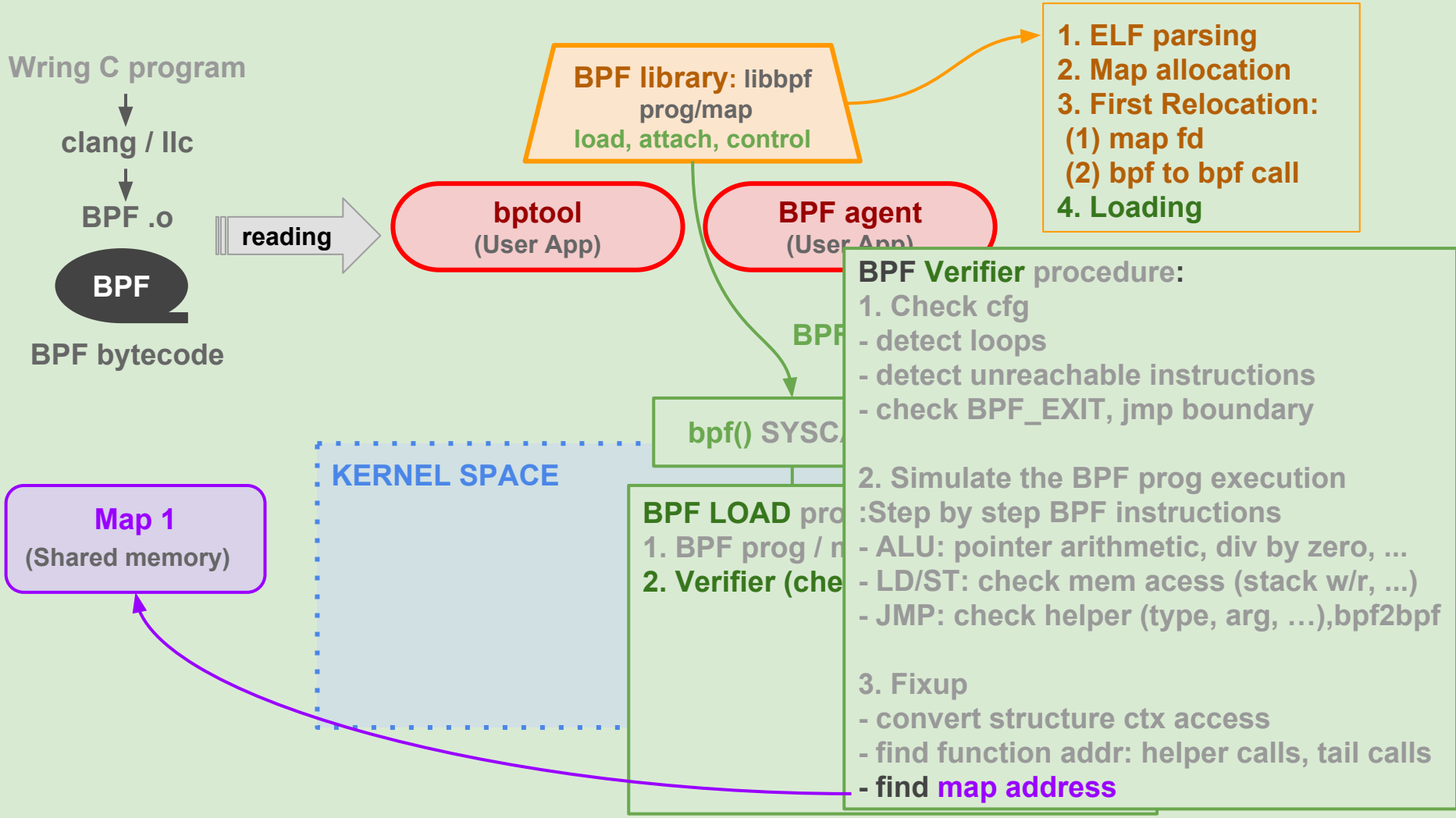
1) Allows one BPF program to call another
2) Implemented as a long jump,
3) Has minimal overhead as unlike function calls
4) Reusing stack frame

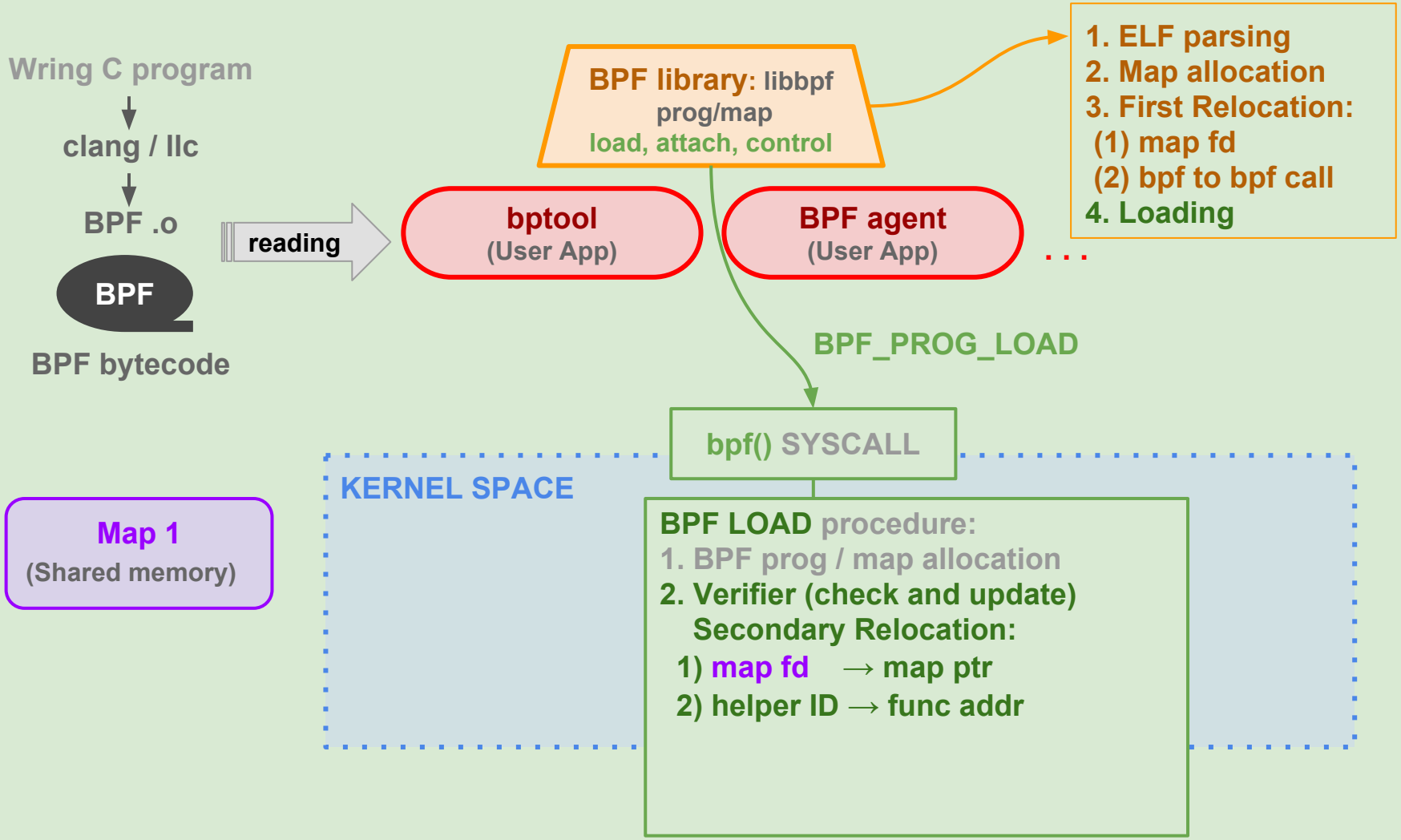
For ...
1) Different parsers depending on packet format: parse_tcp(), parse_udp(), ...
2) Splitting complex programs into small things
3) Dispatching routine (ex. syscall entry tracing)
e.g. bpf programs attached to __seccomp_filter()
...

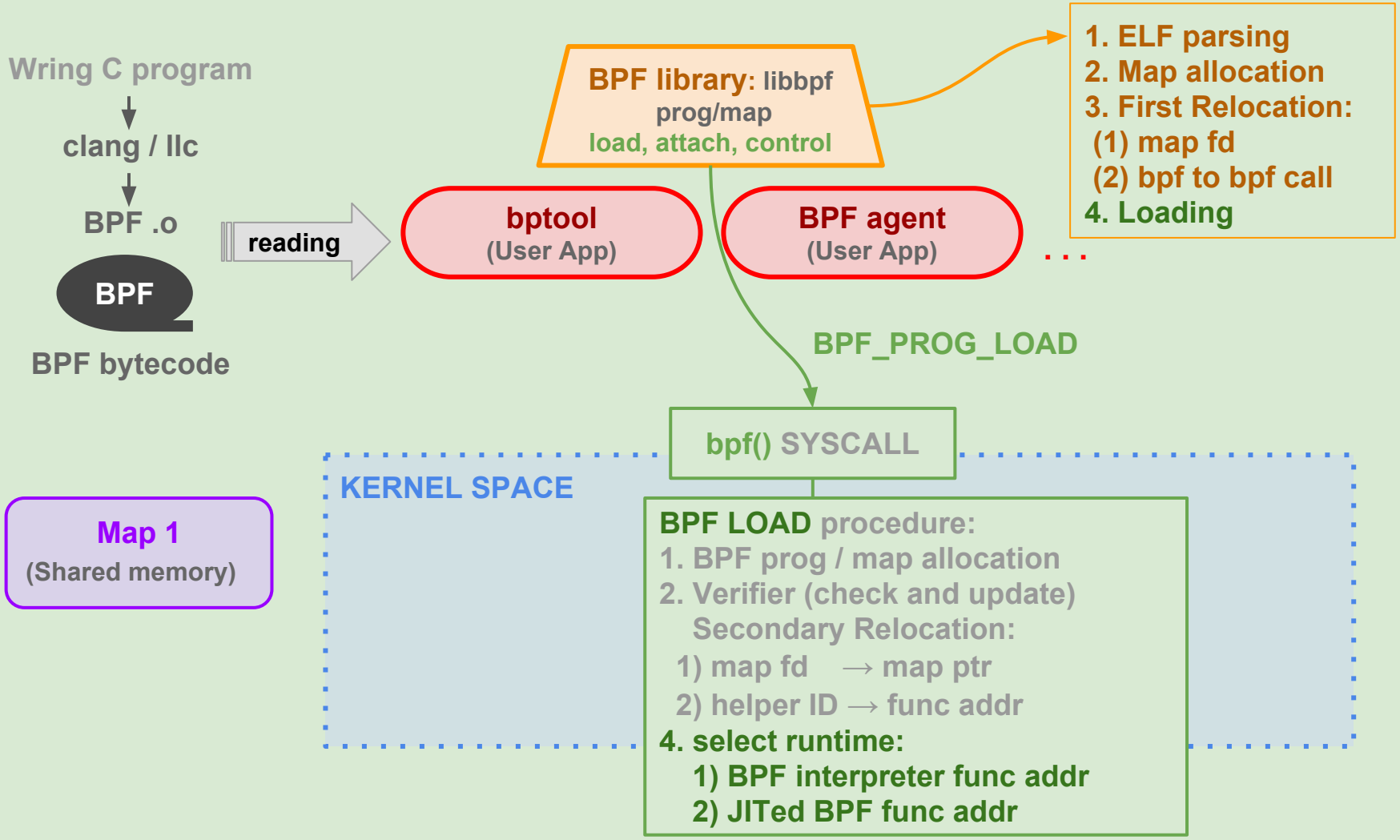
BPF Verifier procedure:
1. Check cfg
- detect loops
- detect unreachable instructions
- check BPF_EXIT, jmp boundary

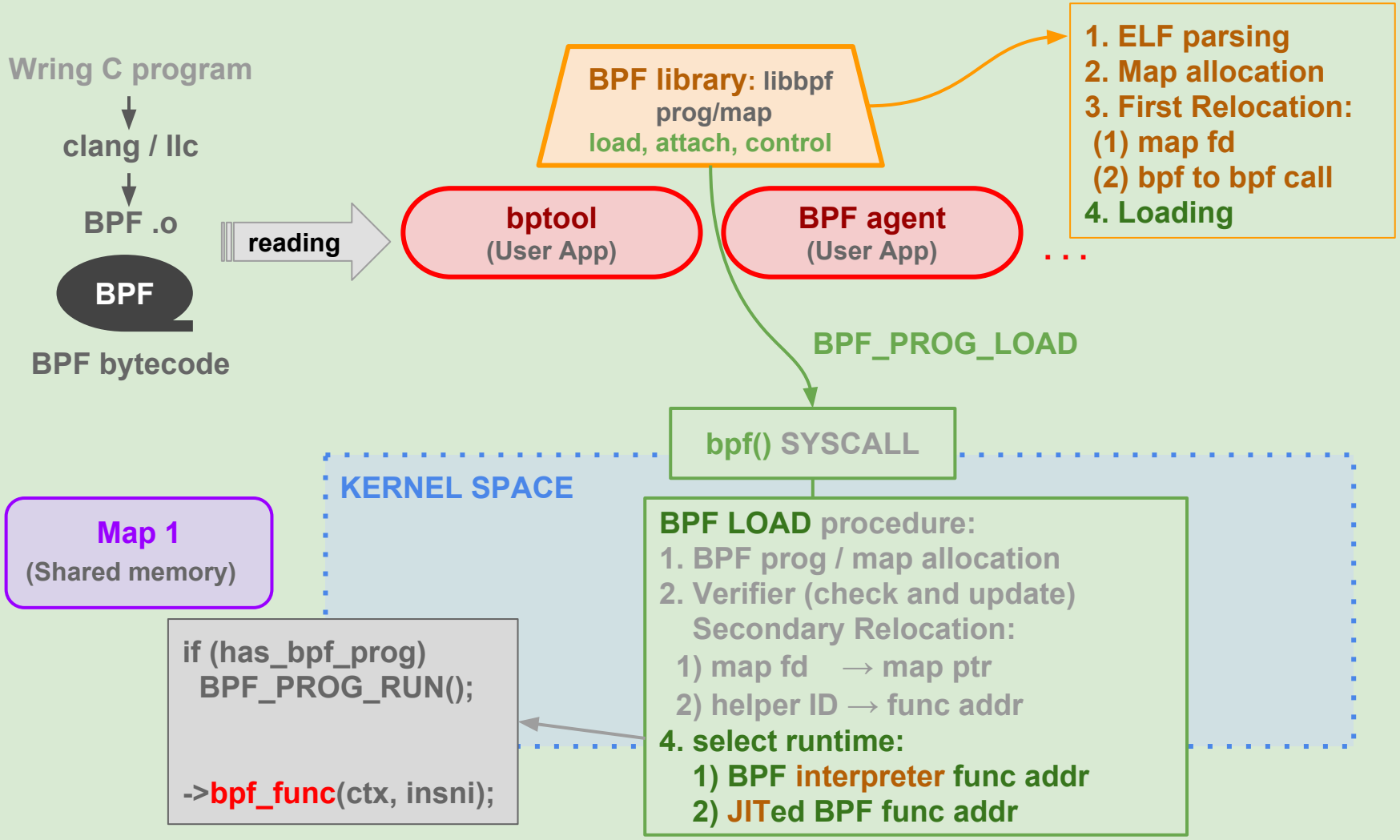
2. Simulate the BPF prog execution
:Step by step BPF instructions
- ALU: pointer arithmetic, div by zero, ...
- LD/ST: check mem acess (stack w/r, ...)
- JMP: check helper (type, arg, …),bpf2bpf

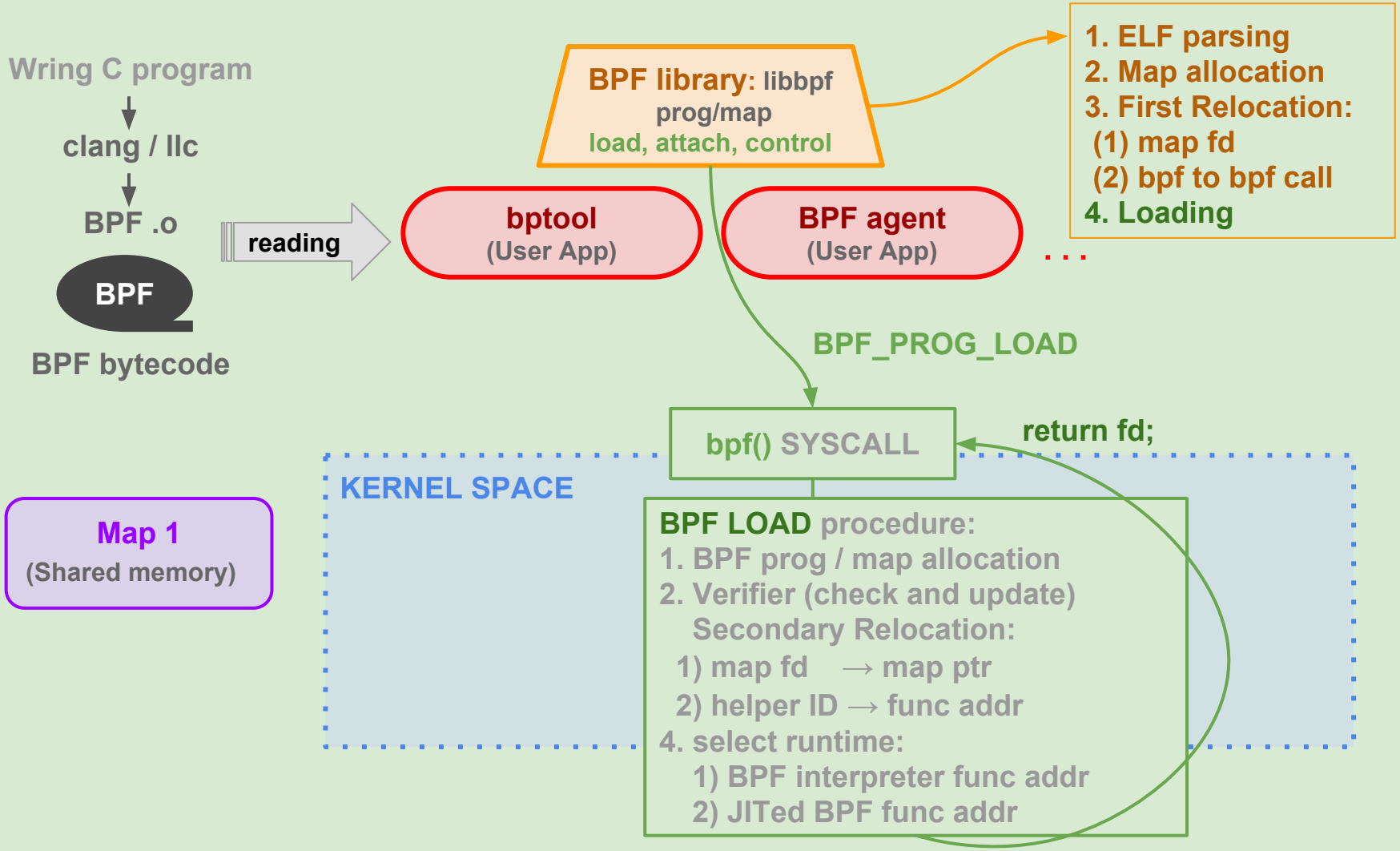
3. Fixup
- convert structure ctx access
- find function addr: helper calls, tail calls

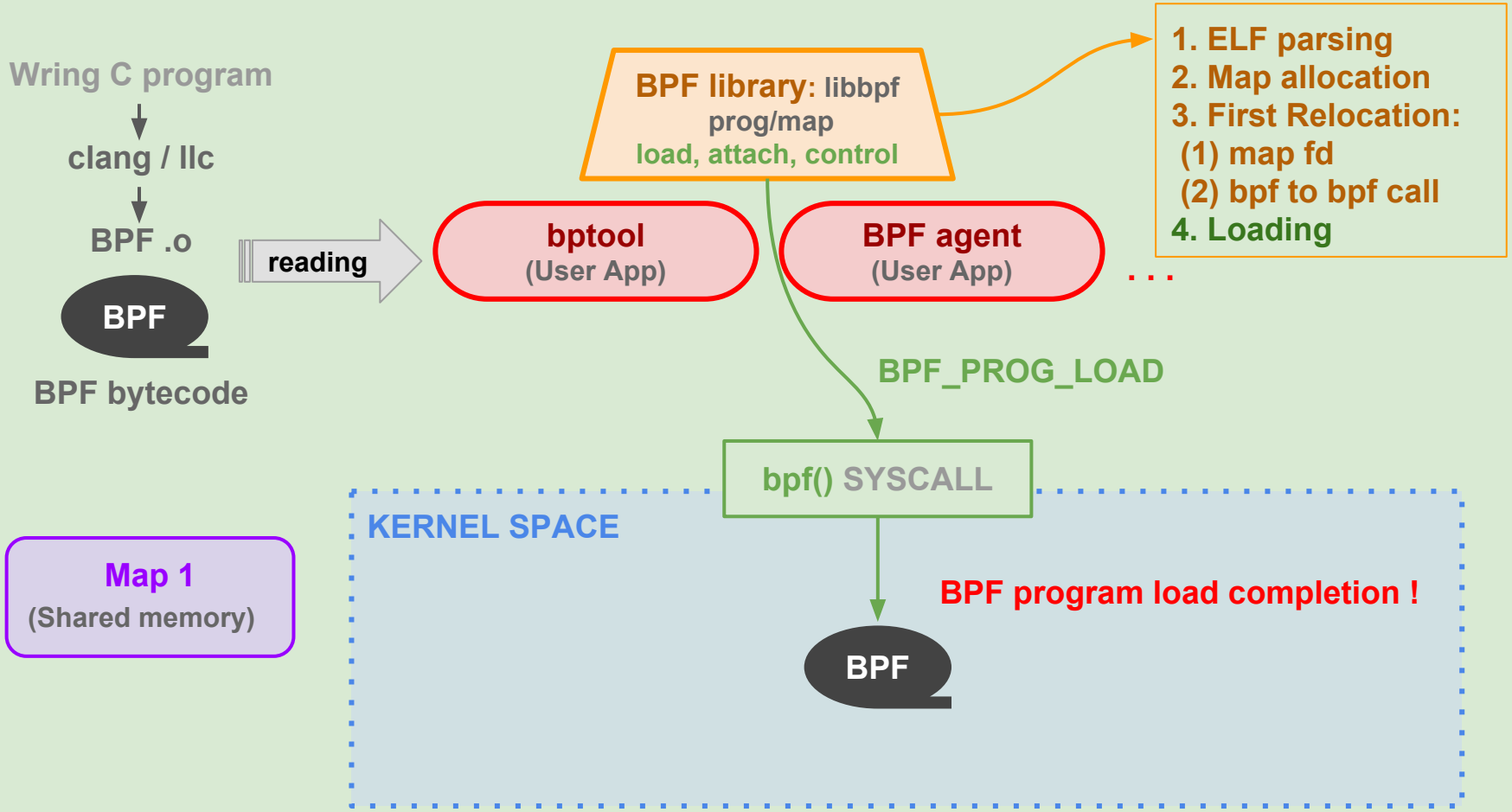

Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
bptool
(User App)
BPF library: libbpf
prog/map
load, attach, control
1. ELF parsing
2. Map allocation
3. First Relocation:
(1) map fd
(2) bpf to bpf call
4. Loading
BPF agent
(User App)
BPF
bpf() SYSC
KERNEL SPACE
BPF LOAD pro
1. BPF prog / m
2. Verifier (che
Map 1
(Shared memory)
BPF Verifier procedure:
1. Check cfg
- detect loops
- detect unreachable instructions
- check BPF_EXIT, jmp boundary
2. Simulate the BPF prog execution
:Step by step BPF instructions
- ALU: pointer arithmetic, div by zero, ...
- LD/ST: check mem acess (stack w/r, ...)
- JMP: check helper (type, arg, …),bpf2bpf
3. Fixup
- convert structure ctx access
- find function addr: helper calls, tail calls
- find map address

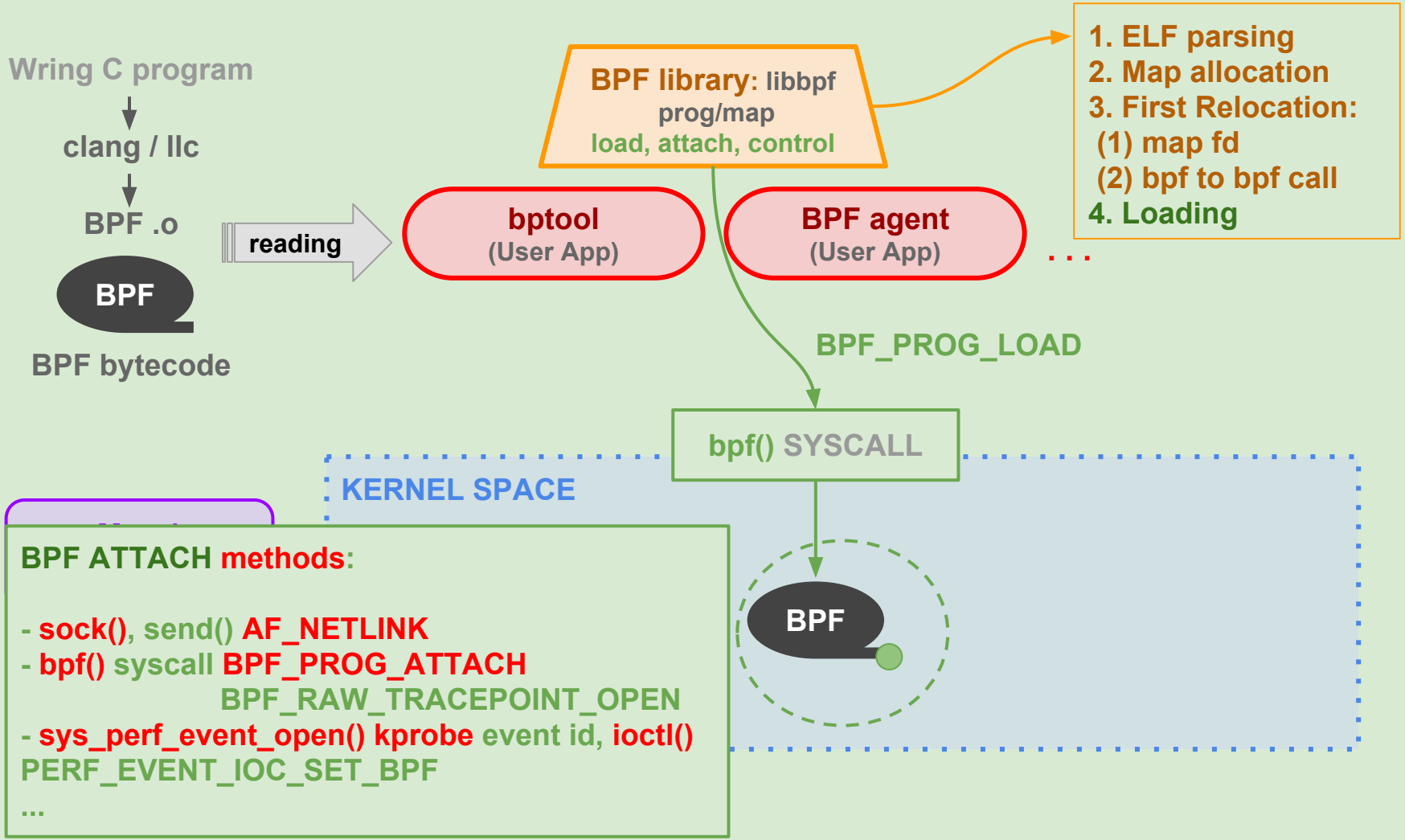

Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
bptool
(User App)
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
. . .
1. ELF parsing
2. Map allocation
3. First Relocation:
(1) map fd
(2) bpf to bpf call
4. Loading
BPF_PROG_LOAD
bpf() SYSCALL
KERNEL SPACE
Map 1
(Shared memory)
BPF LOAD procedure:
1. BPF prog / map allocation
2. Verifier (check and update)
Secondary Relocation:
1) map fd → map ptr
2) helper ID → func addr

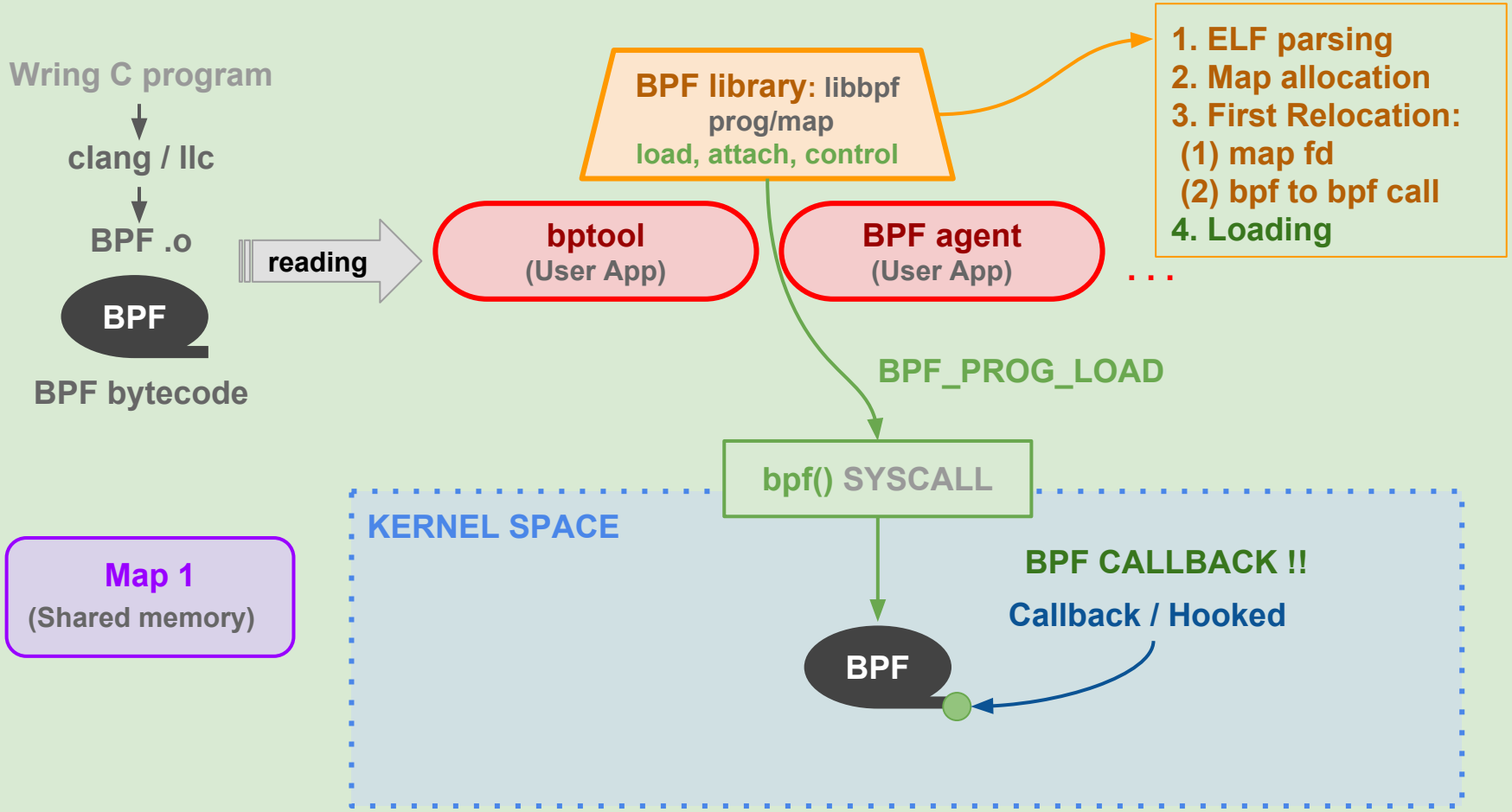

Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
bptool
(User App)
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
. . .
1. ELF parsing
2. Map allocation
3. First Relocation:
(1) map fd
(2) bpf to bpf call
4. Loading
BPF_PROG_LOAD
bpf() SYSCALL
KERNEL SPACE
Map 1
(Shared memory)
BPF LOAD procedure:
1. BPF prog / map allocation
2. Verifier (check and update)
Secondary Relocation:
1) map fd → map ptr
2) helper ID → func addr
4. select runtime:
1) BPF interpreter func addr
2) JITed BPF func addr

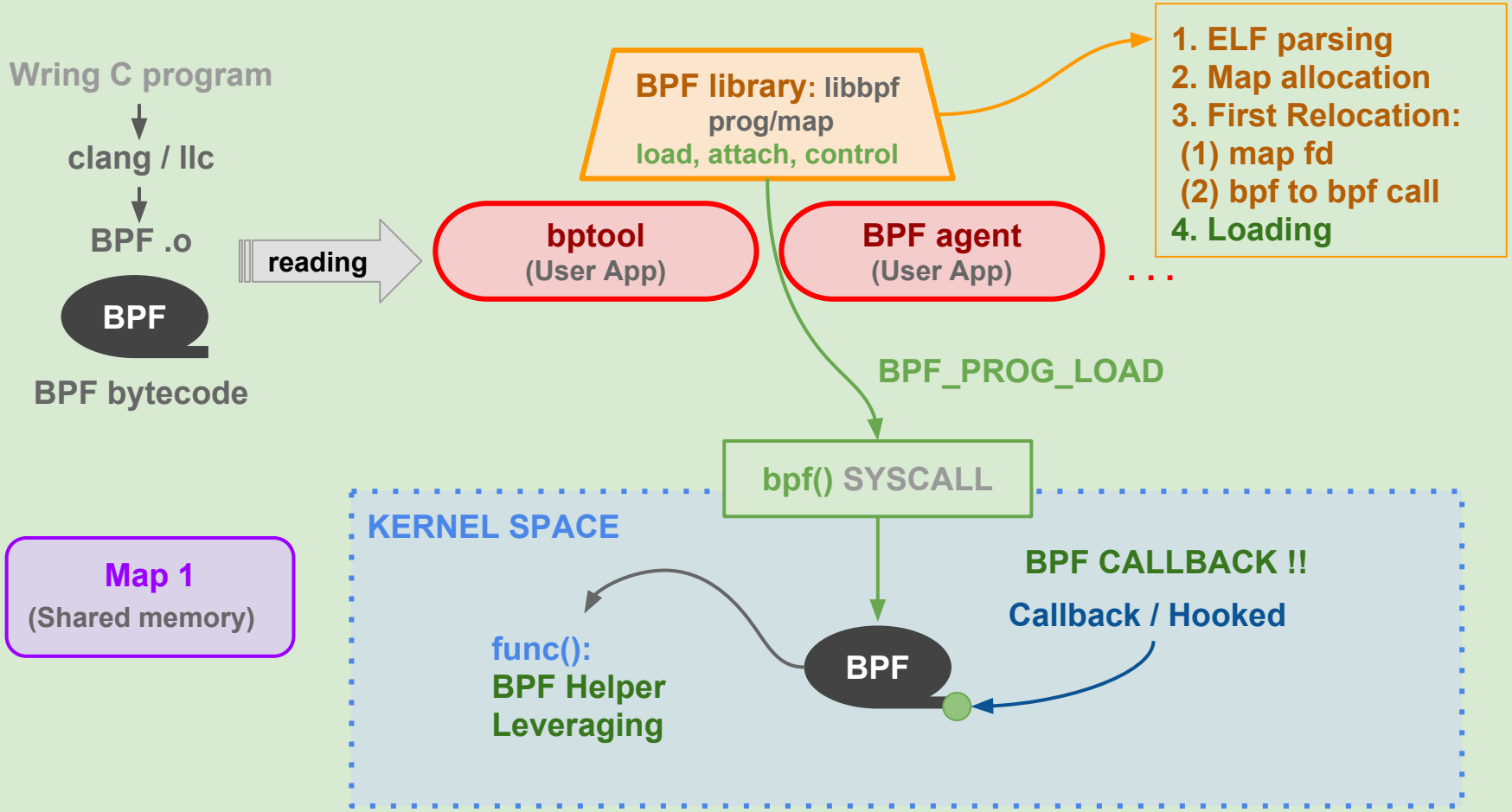

Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
bptool
(User App)
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
. . .
1. ELF parsing
2. Map allocation
3. First Relocation:
(1) map fd
(2) bpf to bpf call
4. Loading
BPF_PROG_LOAD
bpf() SYSCALL
KERNEL SPACE
Map 1
(Shared memory)
BPF LOAD procedure:
1. BPF prog / map allocation
2. Verifier (check and update)
Secondary Relocation:
1) map fd → map ptr
2) helper ID → func addr
4. select runtime:
1) BPF interpreter func addr
2) JITed BPF func addr
if (has_bpf_prog)
BPF_PROG_RUN();
->bpf_func(ctx, insni);

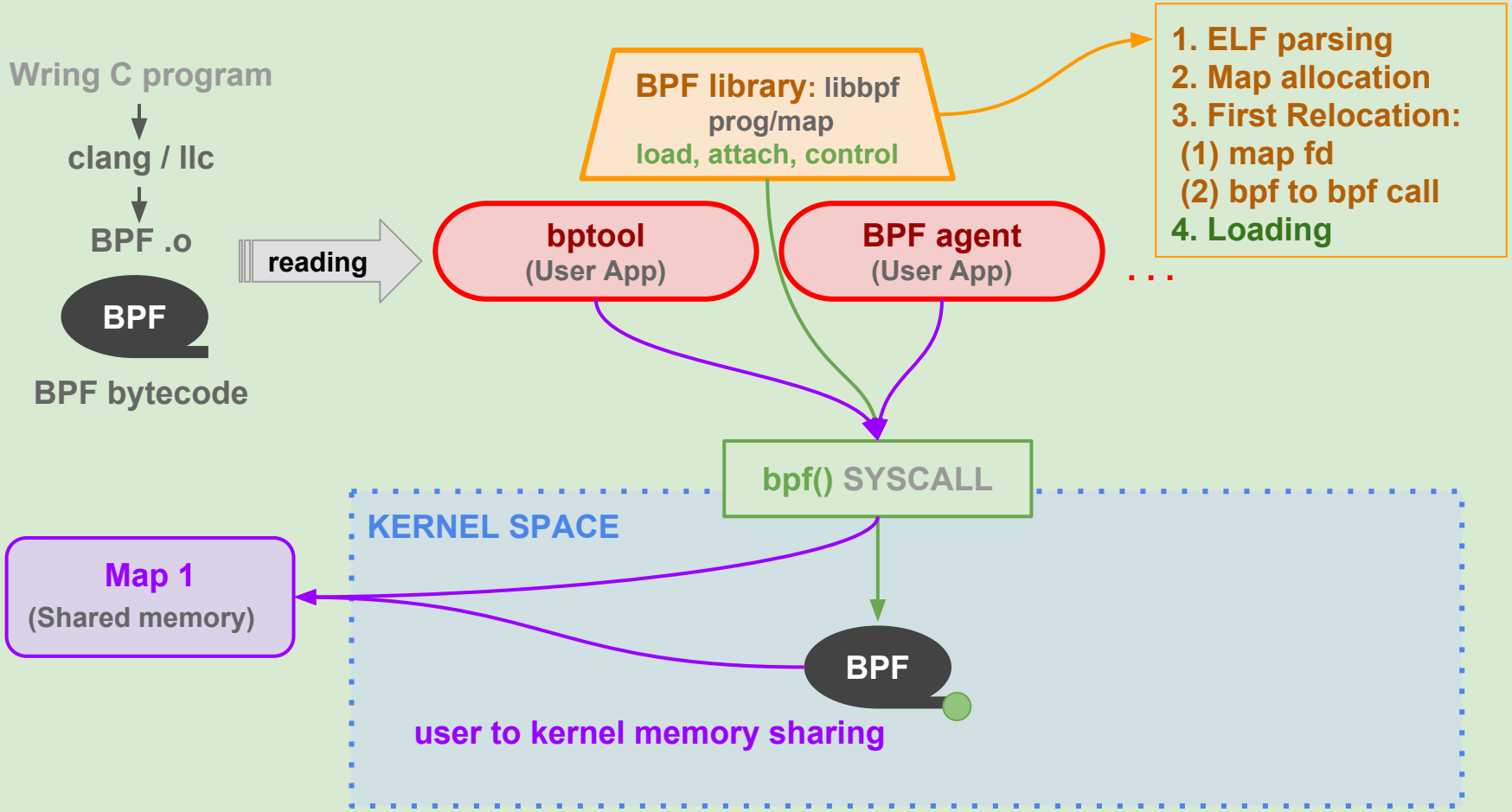

Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
bptool
(User App)
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
. . .
1. ELF parsing
2. Map allocation
3. First Relocation:
(1) map fd
(2) bpf to bpf call
4. Loading
BPF_PROG_LOAD
bpf() SYSCALL
return fd;
KERNEL SPACE
Map 1
(Shared memory)
BPF LOAD procedure:
1. BPF prog / map allocation
2. Verifier (check and update)
Secondary Relocation:
1) map fd → map ptr
2) helper ID → func addr
4. select runtime:
1) BPF interpreter func addr
2) JITed BPF func addr

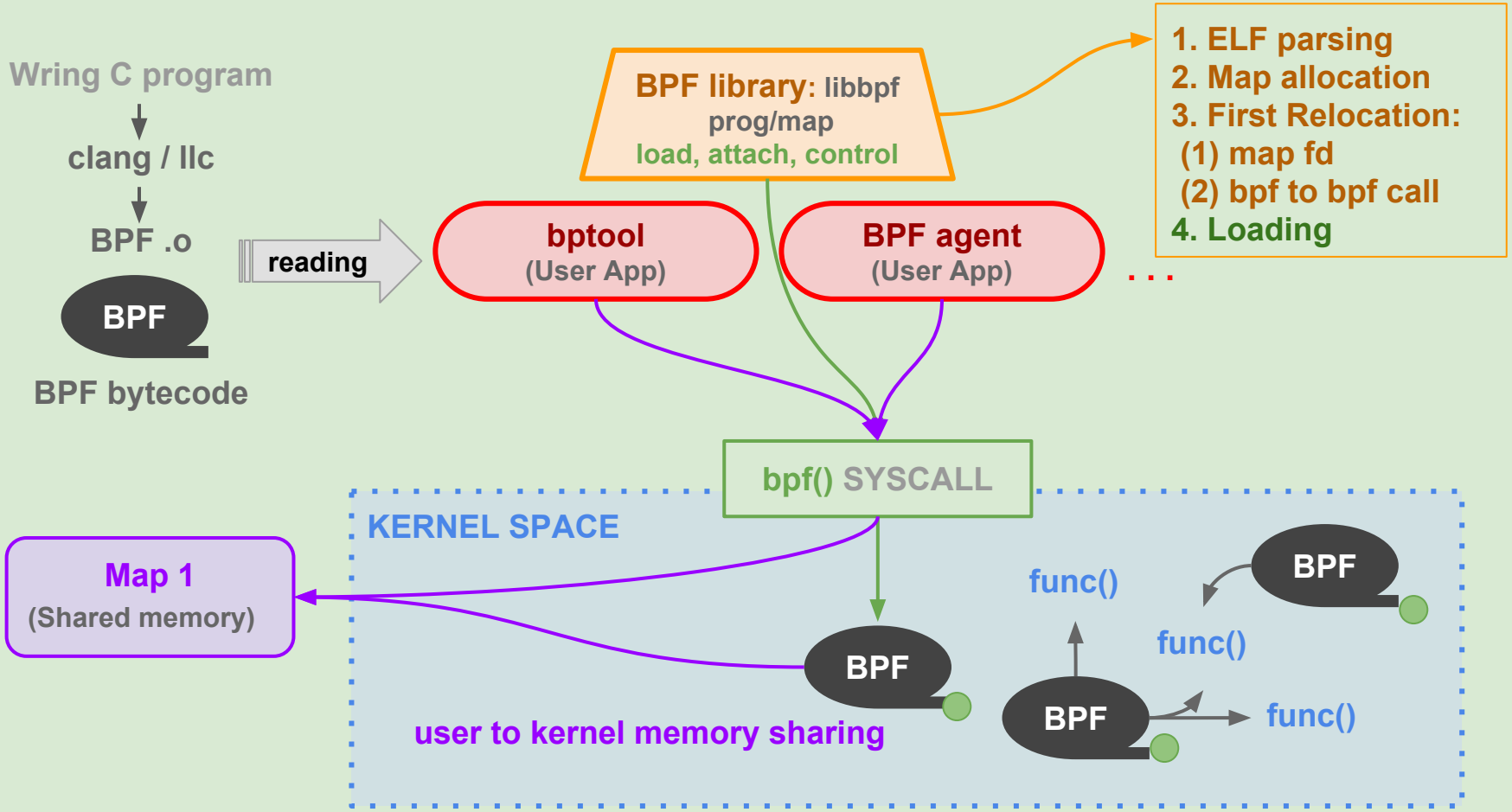

Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
bptool
(User App)
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
. . .
1. ELF parsing
2. Map allocation
3. First Relocation:
(1) map fd
(2) bpf to bpf call
4. Loading
BPF_PROG_LOAD
bpf() SYSCALL
KERNEL SPACE
BPF program load completion !
BPF
Map 1
(Shared memory)


Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
bptool
(User App)
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
. . .
1. ELF parsing
2. Map allocation
3. First Relocation:
(1) map fd
(2) bpf to bpf call
4. Loading
BPF_PROG_LOAD
bpf() SYSCALL
KERNEL SPACE
BPF
BPF ATTACH methods:
- sock(), send() AF_NETLINK
- bpf() syscall BPF_PROG_ATTACH
BPF_RAW_TRACEPOINT_OPEN
- sys_perf_event_open() kprobe event id, ioctl()
PERF_EVENT_IOC_SET_BPF
...

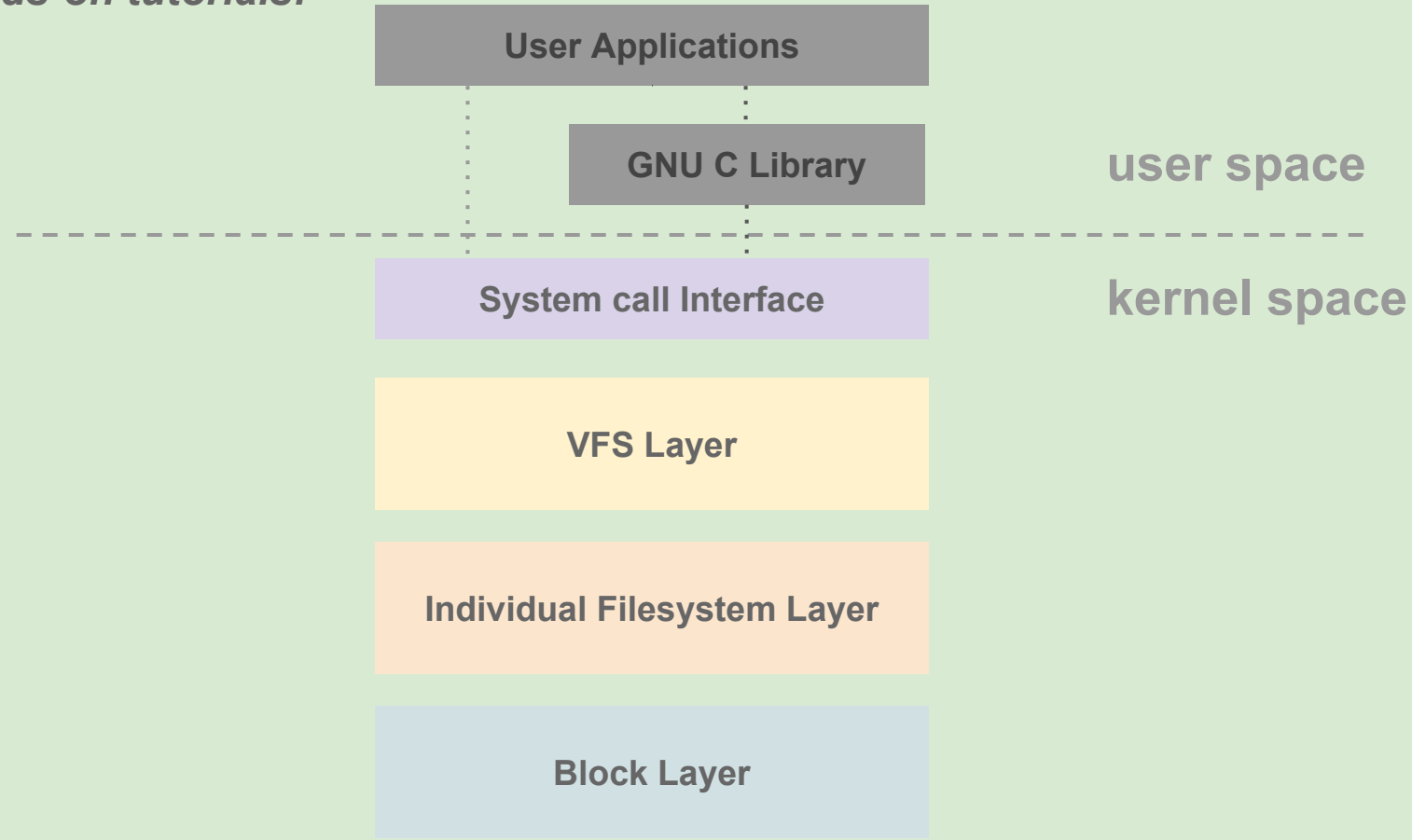

Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
bptool
(User App)
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
. . .
1. ELF parsing
2. Map allocation
3. First Relocation:
(1) map fd
(2) bpf to bpf call
4. Loading
BPF_PROG_LOAD
bpf() SYSCALL
KERNEL SPACE
BPF CALLBACK !!
Callback / Hooked
BPF
Map 1
(Shared memory)

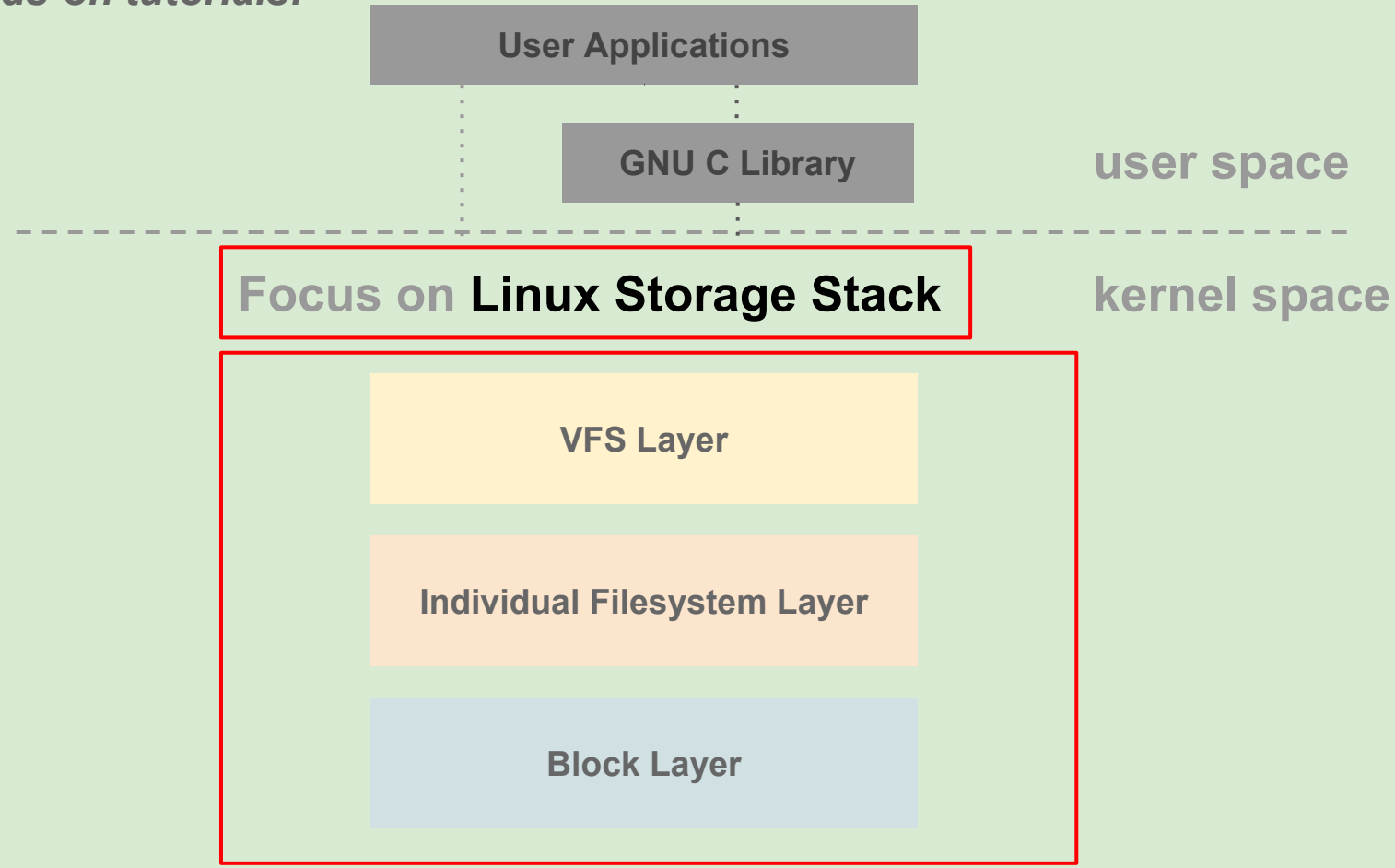

Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
bptool
(User App)
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
. . .
1. ELF parsing
2. Map allocation
3. First Relocation:
(1) map fd
(2) bpf to bpf call
4. Loading
BPF_PROG_LOAD
bpf() SYSCALL
KERNEL SPACE
BPF CALLBACK !!
Callback / Hooked
BPF
func():
BPF Helper
Leveraging
Map 1
(Shared memory)


Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
BPF library: libbpf
prog/map
load, attach, control
1. ELF parsing
2. Map allocation
3. First Relocation:
(1) map fd
(2) bpf to bpf call
4. Loading
bptool
(User App)
BPF agent
(User App)
. . .
bpf() SYSCALL
KERNEL SPACE
Map 1
(Shared memory)
BPF
user to kernel memory sharing


Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
BPF library: libbpf
prog/map
load, attach, control
1. ELF parsing
2. Map allocation
3. First Relocation:
(1) map fd
(2) bpf to bpf call
4. Loading
bptool
(User App)
BPF agent
(User App)
. . .
bpf() SYSCALL
KERNEL SPACE
Map 1
(Shared memory)
BPF
func()
BPF
func()
BPF
func()
user to kernel memory sharing

Before BPF hands-on tutorials,


User Applications
GNU C Library
user space
System call Interface
kernel space
VFS Layer
Individual Filesystem Layer
Block Layer


User Applications
GNU C Library
user space
Focus on Linux Storage Stack
kernel space
VFS Layer
Individual Filesystem Layer
Block Layer

Sure,

There are already many tracing tools

(blktrace, tracepoints, ...)

But,

BPF contains not only those advantages

(blktrace, tracepoints, ...)

but also

But,
BPF contains not only those advantages
(blktrace, tracepoints, ...)

but also + α: low-overhead profiling and tracing,
rich introspection,
detailedly tracking kernel data,
...

Before BPF hands-on tutorials:

Stage 00: Traces the latency between blk_mq_start_request() -> blk_account_io_completion()

```
$ cd ~/git/linux/samples/bpf

$ sudo ./tracex3
failed to create kprobe 'blk_start_request' error 'No such file or directory'

# blk_start_request() don't exist anymore in latest kernel version
# Edit tracex3_kern.c to replace 'blk_start_request' to 'blk_mq_start_request'
$ vim tracex3_kern.c
$ make
# Retry to run it
$ sudo ./tracex3
```

Before BPF hands-on tutorials:

Stage 00: Traces the latency between blk_mq_start_request() -> blk_account_io_completion()

```
$ ls tracex3*
tracex3  tracex3_kern.c  tracex3_kern.o  tracex3_user.c  tracex3_user.o
$ ls bpf_load.o
bpf_load.o

$ readelf -h tracex3_kern.o | grep Machine
  Machine:                           Linux BPF

$ readelf -h tracex3_user.o | grep Machine
  Machine:                           Advanced Micro Devices X86-64

$ readelf -h tracex3 | grep Machine
  Machine:                           Advanced Micro Devices X86-64
```

Before BPF hands-on tutorials:

Stage 00: Traces the latency between blk_mq_start_request() -> blk_account_io_completion()

```
$ ls tracex3*
tracex3  tracex3_kern.c  tracex3_kern.o  tracex3_user.c  tracex3_user.o
$ ls bpf_load.o
bpf_load.o

$ readelf -h tracex3_kern.o | grep Machine
  Machine:                           Linux BPF

$ readelf -h tracex3_user.o | grep Machine
  Machine:                           Advanced Micro Devices X86-64

$ readelf -h tracex3 | grep Machine
  Machine:                           Advanced Micro Devices X86-64
```

Before BPF hands-on tutorials:

Stage 00: Traces the latency between blk_mq_start_request() -> blk_account_io_completion()

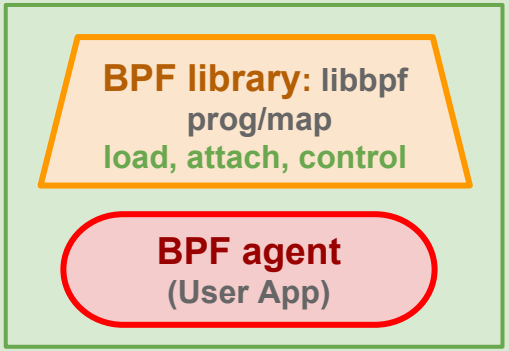
```
$ ls tracex3*
tracex3  tracex3_kern.c  tracex3_kern.o  tracex3_user.c  tracex3_user.o
$ ls bpf_load.o
bpf_load.o

$ readelf -h tracex3_kern.o | grep Machine
  Machine:                           Linux BPF

$ readelf -h tracex3_user.o | grep Machine
  Machine:                           Advanced Micro Devices X86-64

$ readelf -h tracex3 | grep Machine
  Machine:                           Advanced Micro Devices X86-64
```

Before BPF hands-on tutorials:

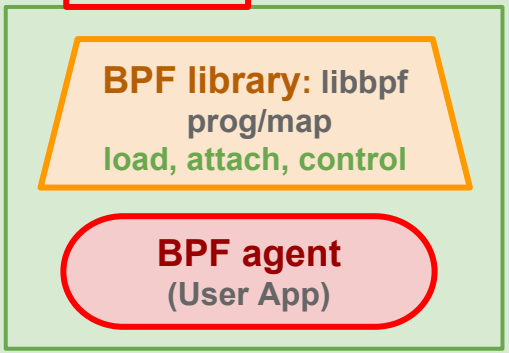
Stage 00: Traces the latency between blk_mq_start_request() -> blk_account_io_completion()

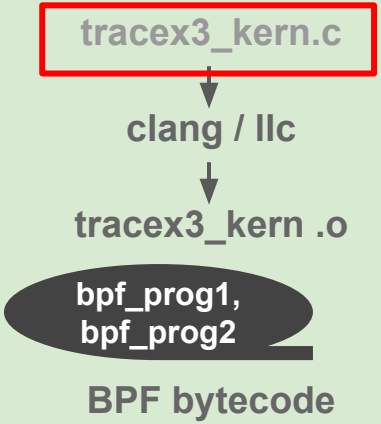
```
$ ls tracex3*
tracex3  tracex3_kern.c  tracex3_kern.o  tracex3_user.c  tracex3_user.o
$ ls bpf_load.o
bpf_load.o

$ readelf -h tracex3_kern.o | grep Machine
  Machine:                           Linux BPF

$ readelf -h tracex3_user.o | grep Machine
  Machine:                           Advanced Micro Devices X86-64

$ readelf -h tracex3 | grep Machine
  Machine:                           Advanced Micro Devices X86-64
```

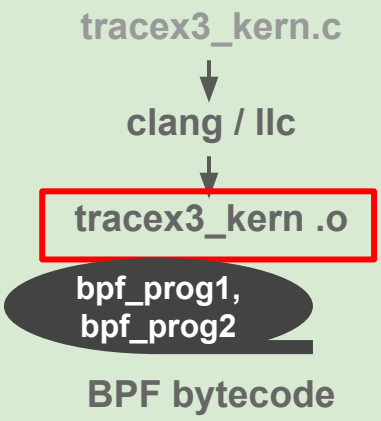
bpf_load.o + tracex3_user.o
=> tracex3

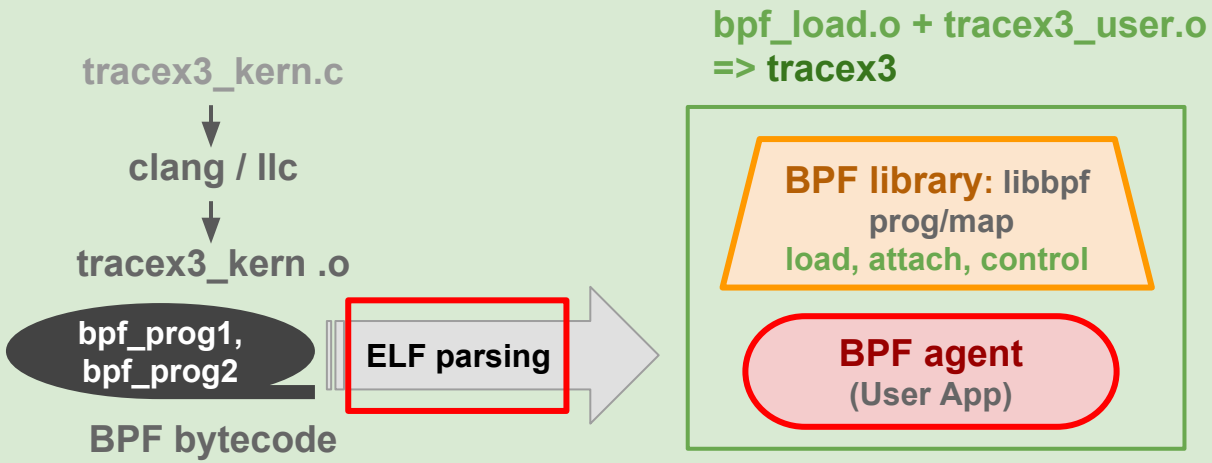


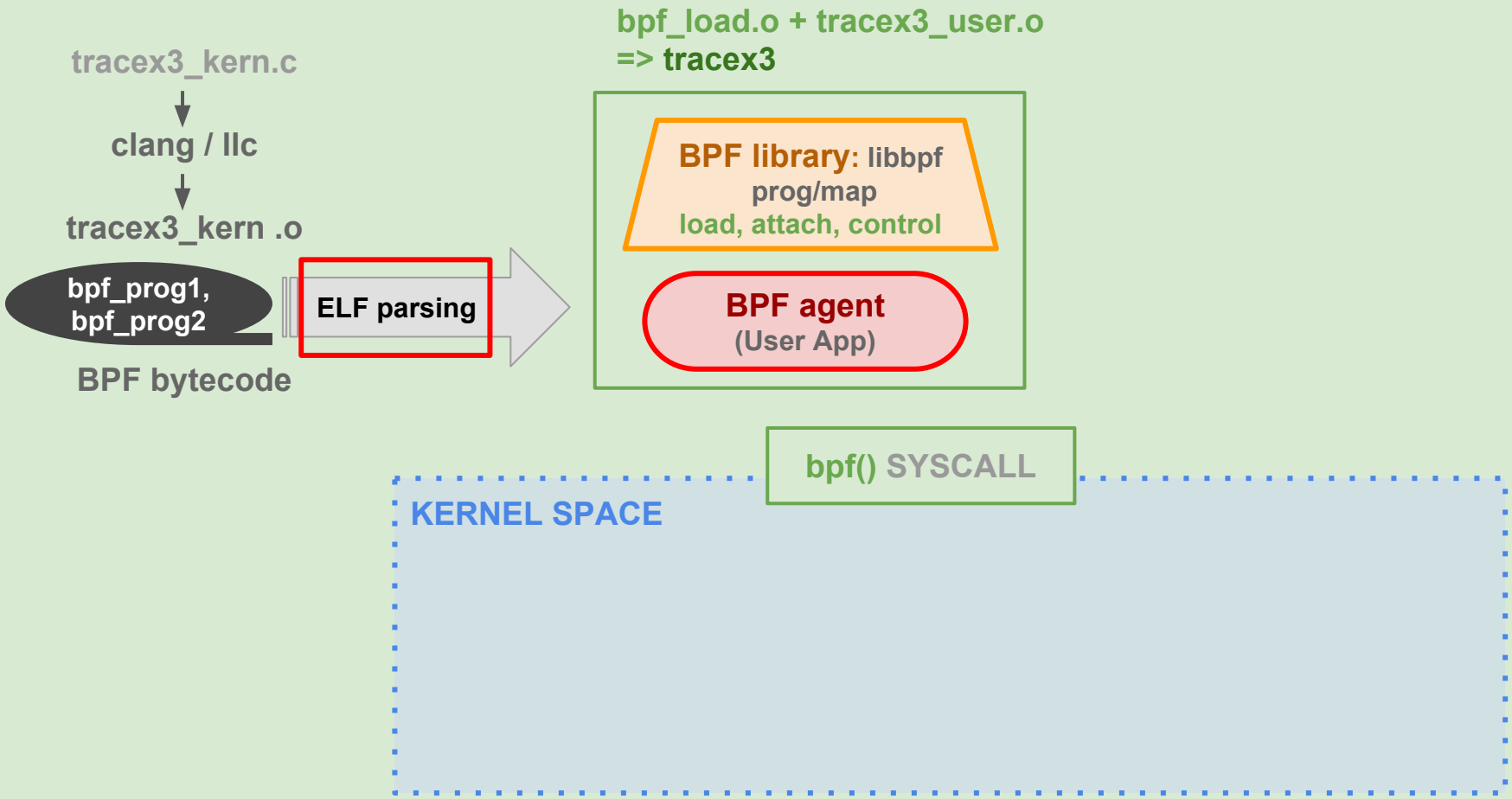
bpf_load.o + tracex3_user.o
=> tracex3

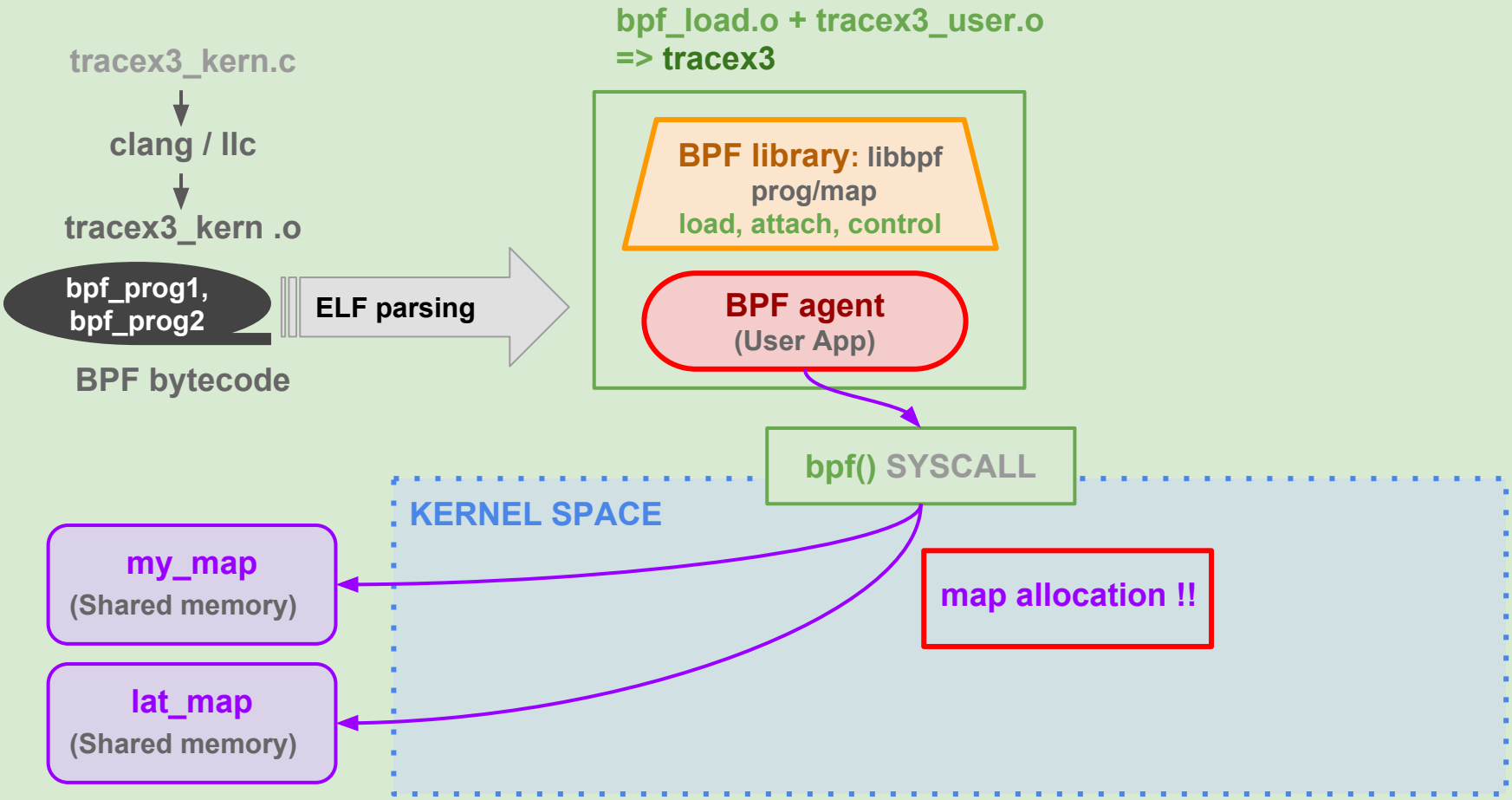


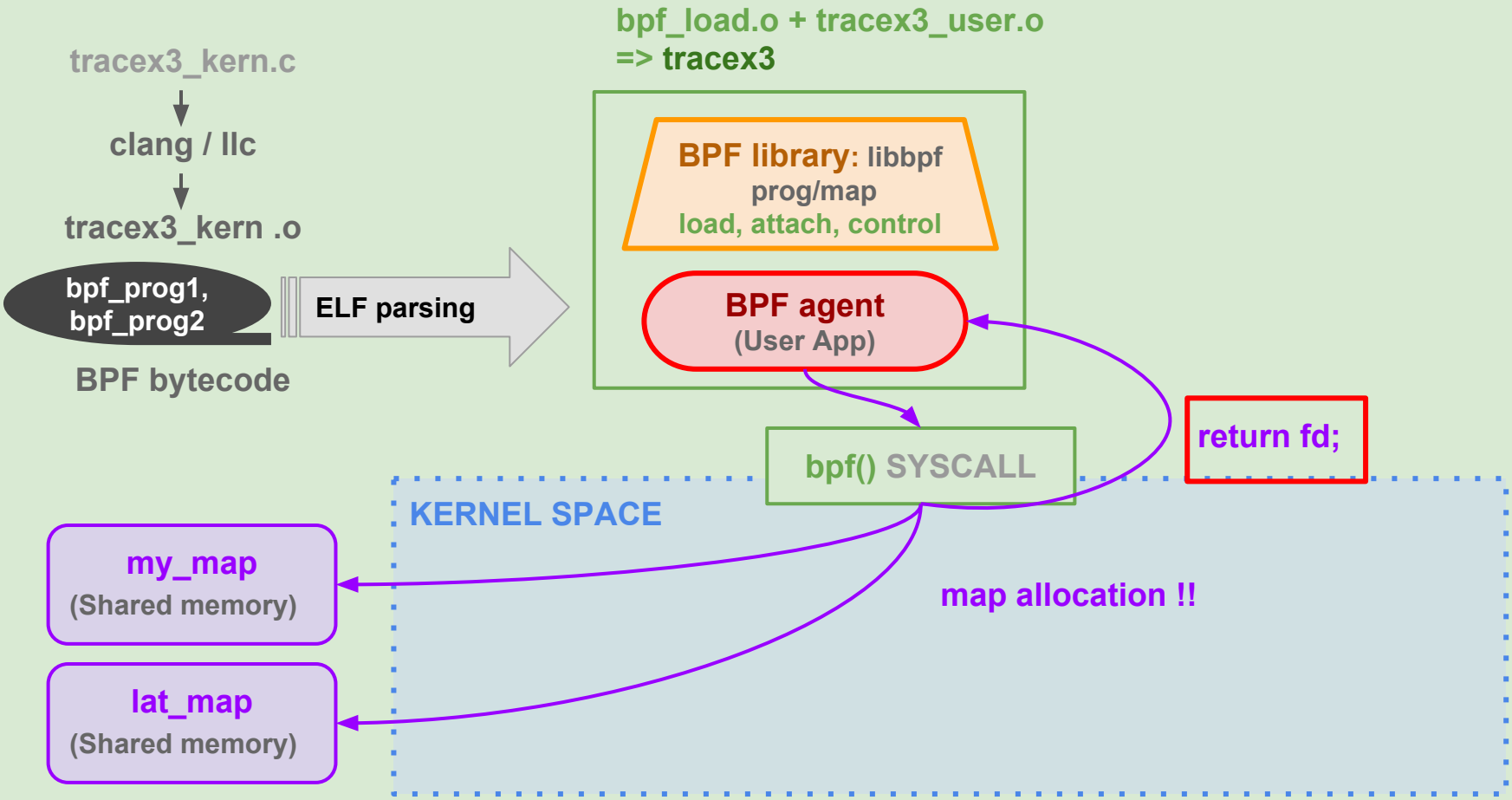

tracex3_kern.c
clang / llc
tracex3_kern .o
bpf_prog1,
bpf_prog2
BPF bytecode

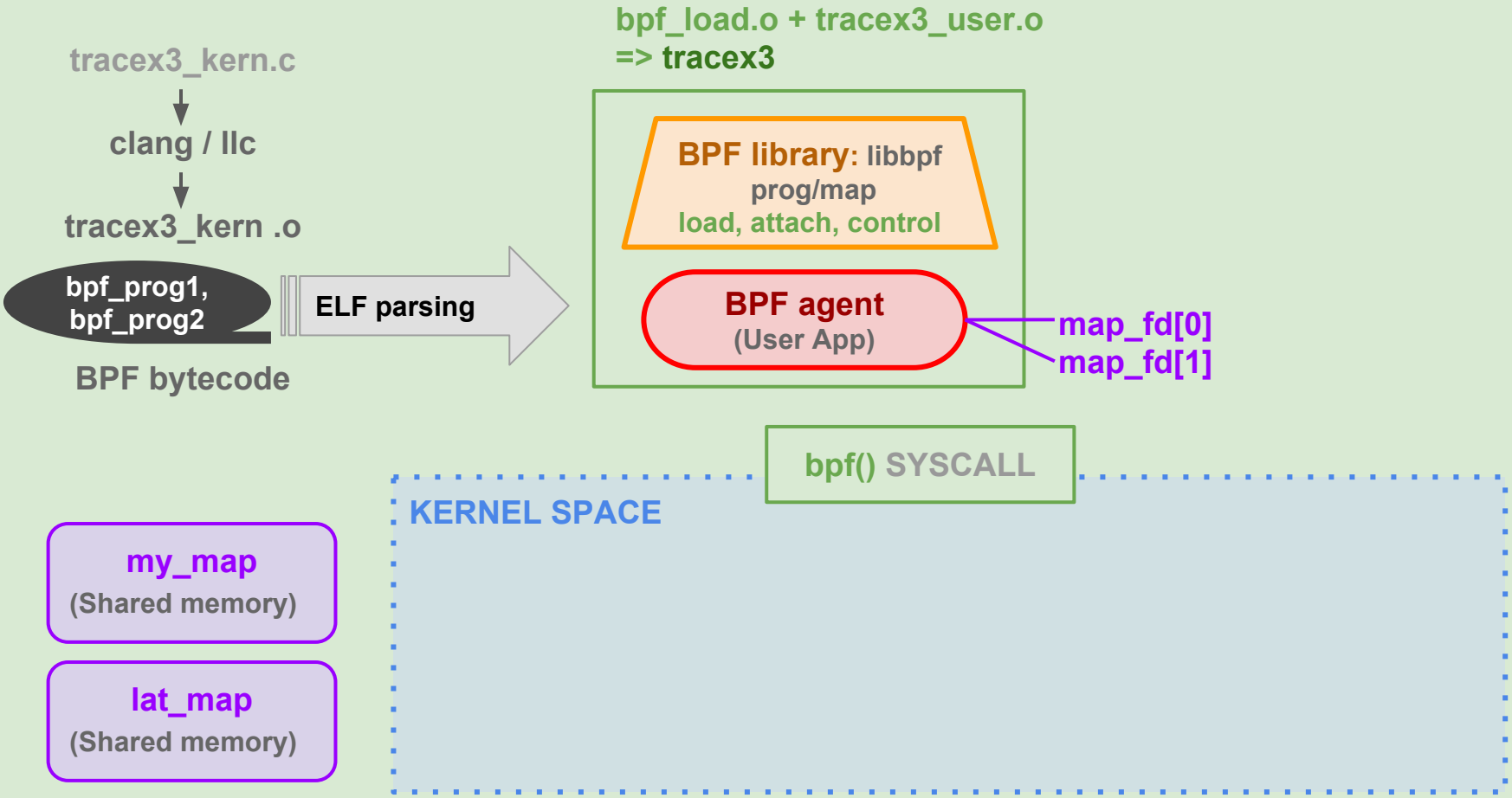

tracex3_kern.c
clang / llc
tracex3_kern .o
bpf_prog1,
bpf_prog2
BPF bytecode

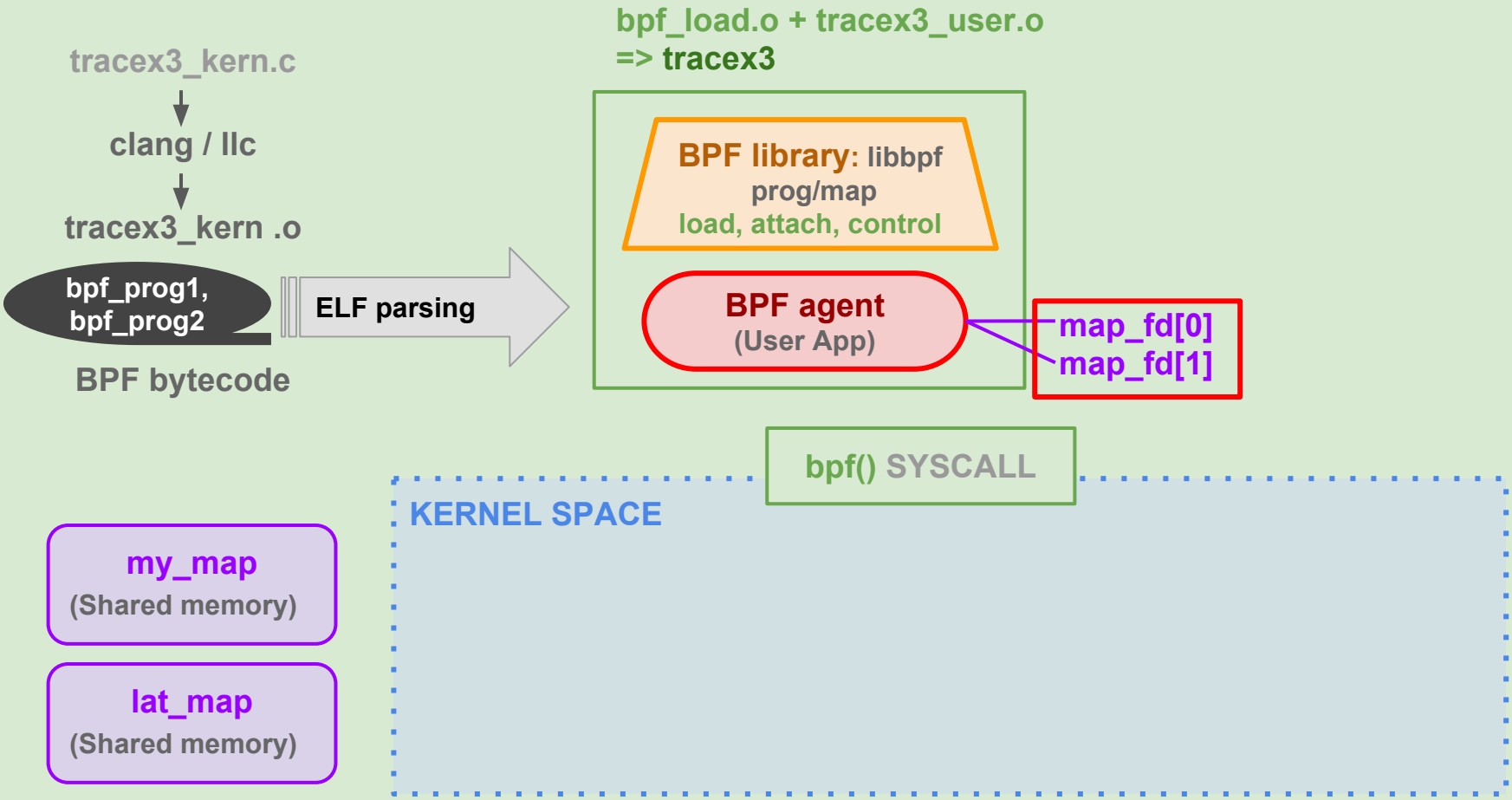

tracex3_kern.c
clang / llc
tracex3_kern .o
bpf_prog1,
bpf_prog2
BPF bytecode
ELF parsing
bpf_load.o + tracex3_user.o
=> tracex3
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)

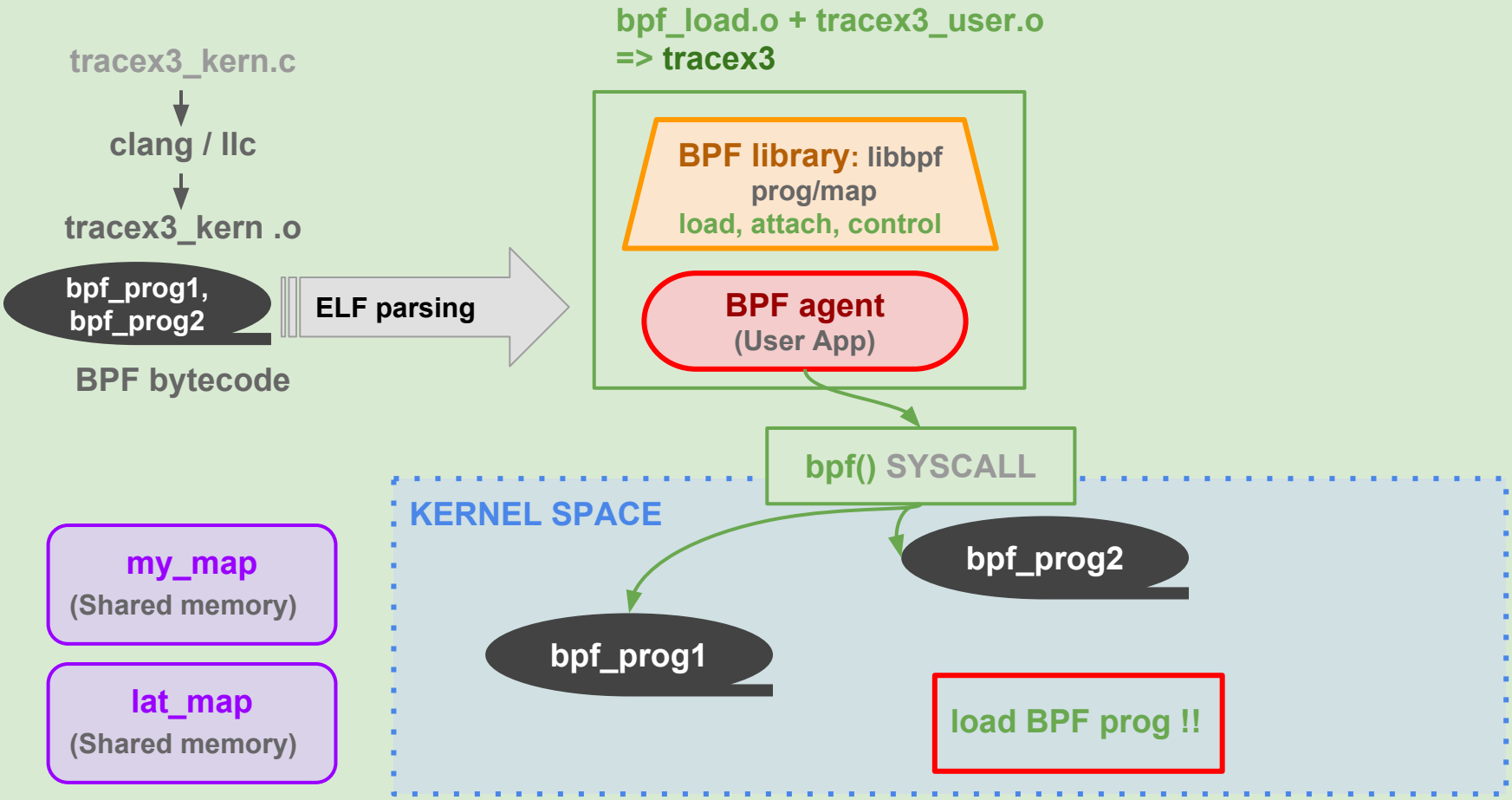

tracex3_kern.c
clang / llc
tracex3_kern .o
bpf_prog1,
bpf_prog2
BPF bytecode
ELF parsing
bpf_load.o + tracex3_user.o
=> tracex3
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
bpf() SYSCALL
KERNEL SPACE

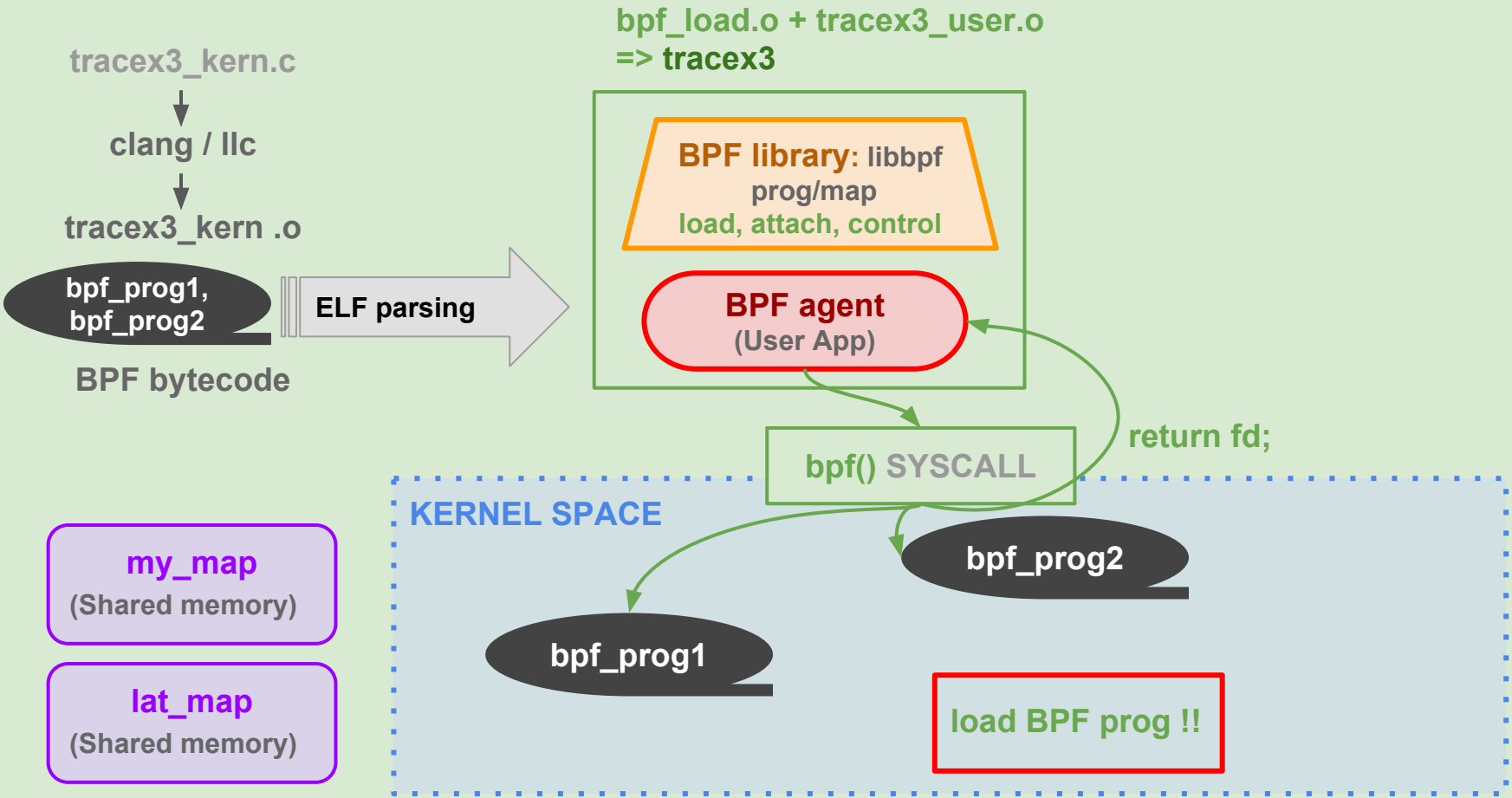

tracex3_kern.c
clang / llc
tracex3_kern .o
bpf_prog1,
bpf_prog2
BPF bytecode
ELF parsing
bpf_load.o + tracex3_user.o
=> tracex3
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
bpf() SYSCALL
KERNEL SPACE
map allocation !!
my_map
(Shared memory)
lat_map
(Shared memory)

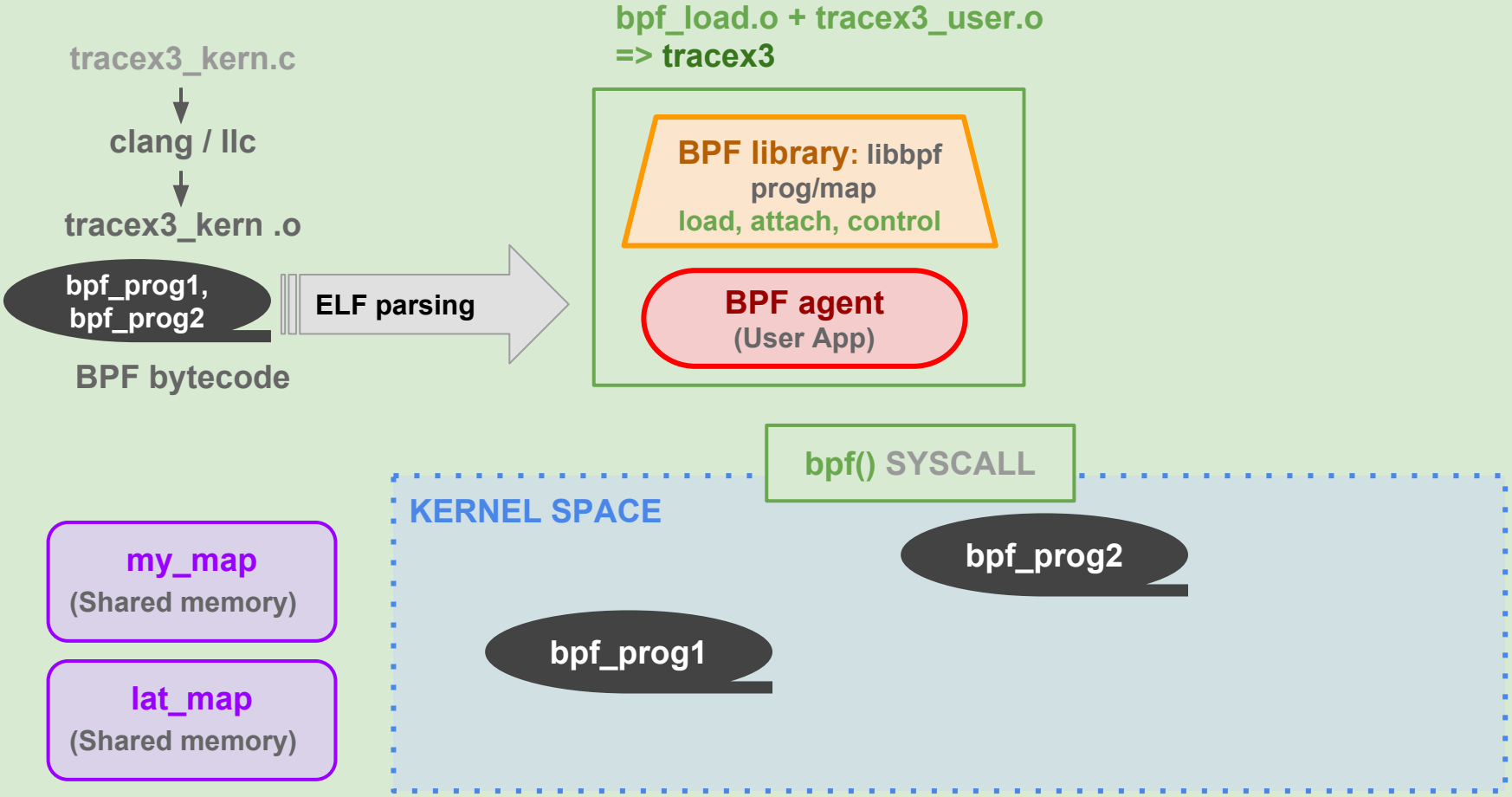

tracex3_kern.c
clang / llc
tracex3_kern .o
bpf_prog1,
bpf_prog2
BPF bytecode
ELF parsing
bpf_load.o + tracex3_user.o
=> tracex3
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
bpf() SYSCALL
return fd;
KERNEL SPACE
map allocation !!
my_map
(Shared memory)
lat_map
(Shared memory)

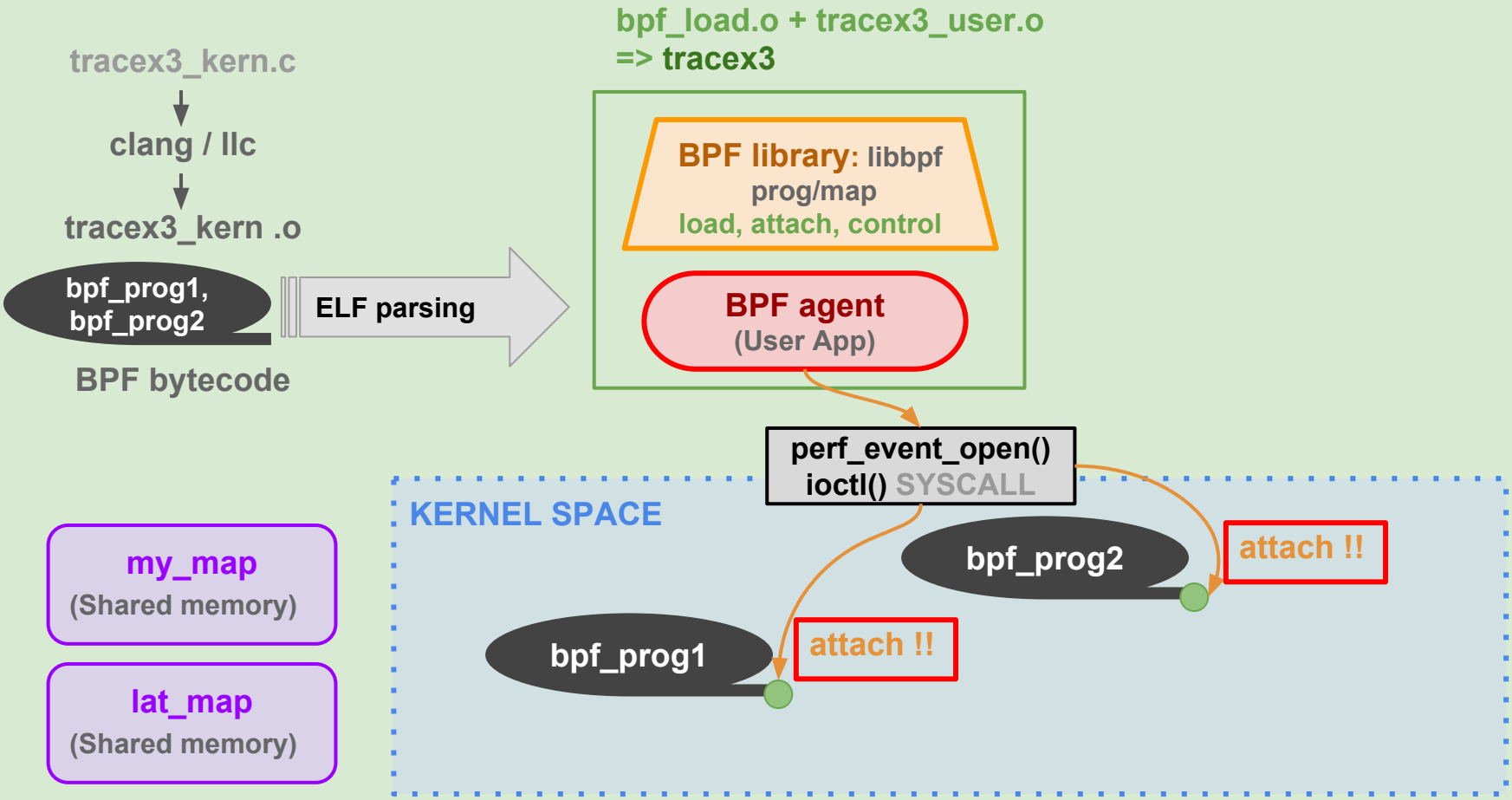

tracex3_kern.c
clang / llc
tracex3_kern .o
bpf_prog1,
bpf_prog2
BPF bytecode
ELF parsing
bpf_load.o + tracex3_user.o
=> tracex3
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
map_fd[0]
map_fd[1]
bpf() SYSCALL
KERNEL SPACE
my_map
(Shared memory)
lat_map
(Shared memory)

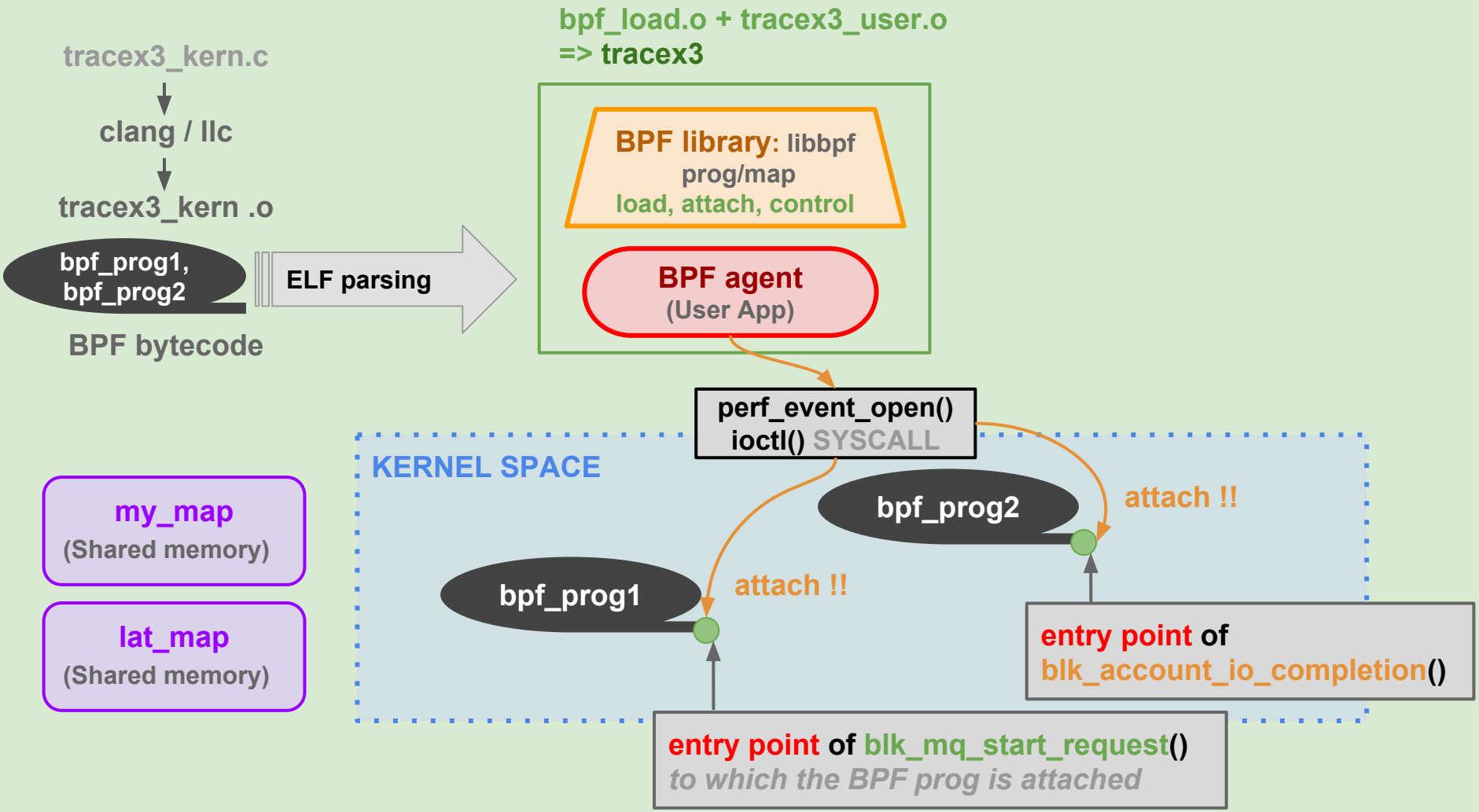

tracex3_kern.c
clang / llc
tracex3_kern .o
bpf_prog1,
bpf_prog2
BPF bytecode
ELF parsing
bpf_load.o + tracex3_user.o
=> tracex3
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
map_fd[0]
map_fd[1]
bpf() SYSCALL
KERNEL SPACE
my_map
(Shared memory)
lat_map
(Shared memory)

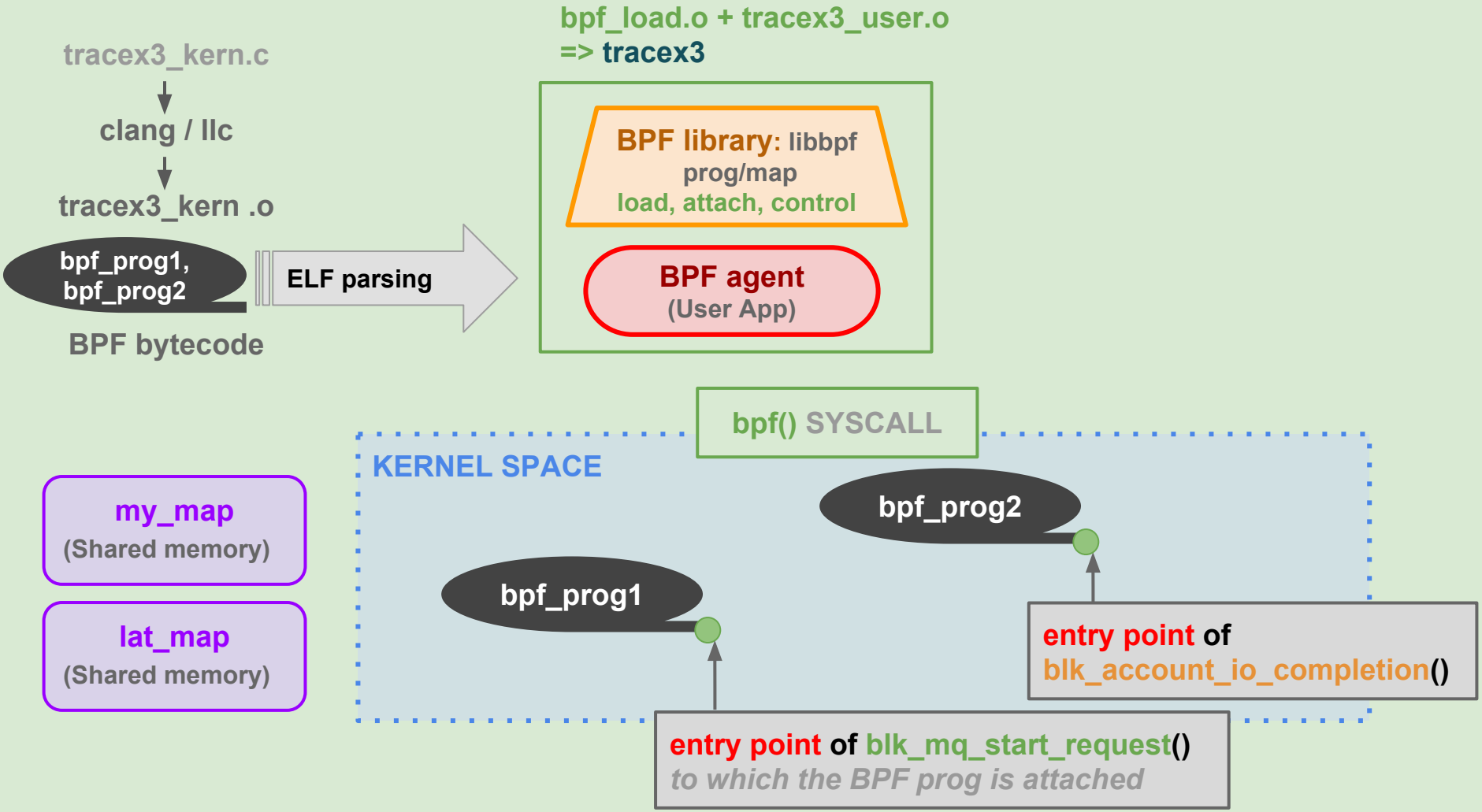

tracex3_kern.c
clang / llc
tracex3_kern .o
bpf_prog1,
bpf_prog2
BPF bytecode
ELF parsing
bpf_load.o + tracex3_user.o
=> tracex3
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
bpf() SYSCALL
KERNEL SPACE
bpf_prog2
bpf_prog1
load BPF prog !!
my_map
(Shared memory)
lat_map
(Shared memory)

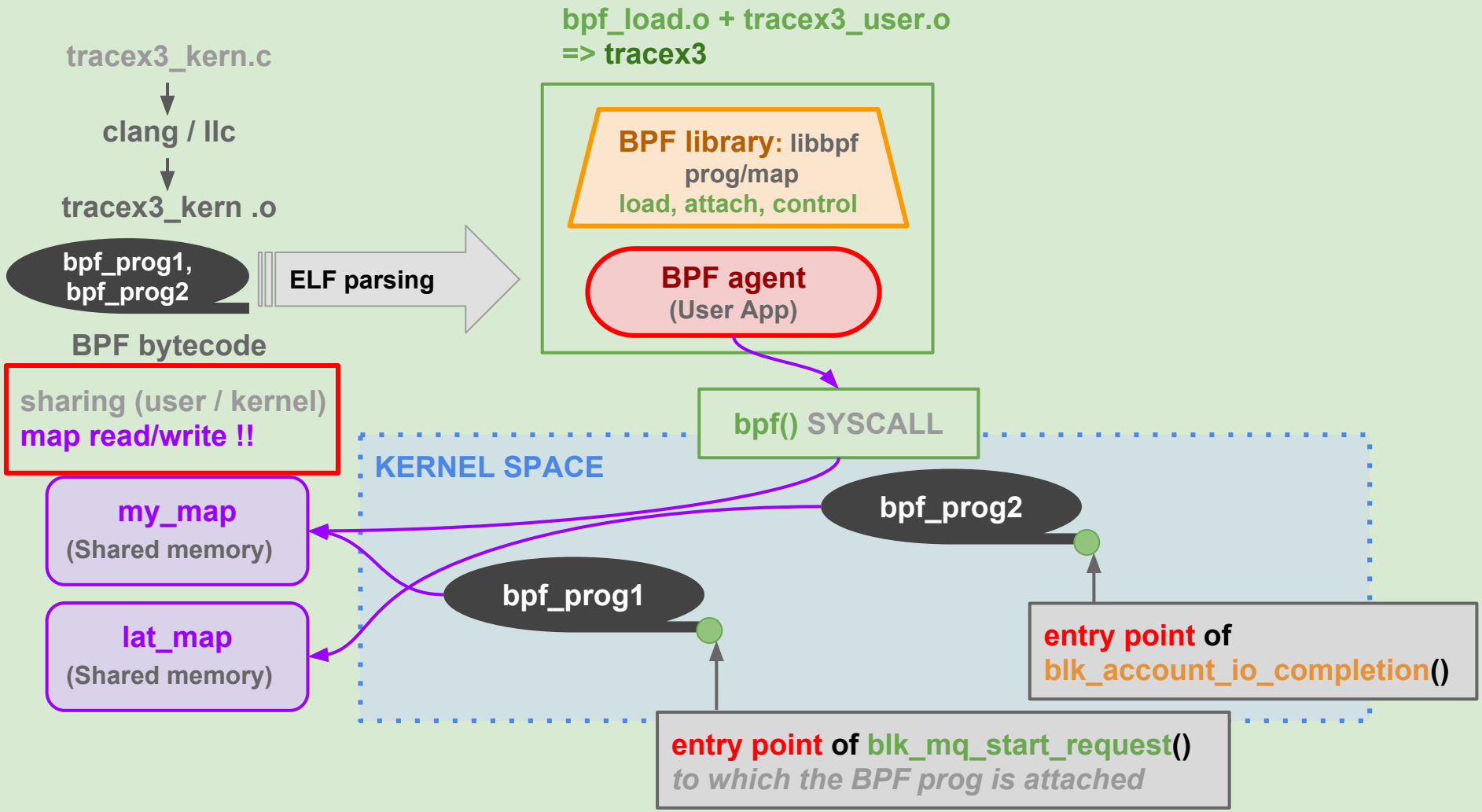

tracex3_kern.c
clang / llc
tracex3_kern .o
bpf_prog1,
bpf_prog2
BPF bytecode
ELF parsing
bpf_load.o + tracex3_user.o
=> tracex3
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
bpf() SYSCALL
return fd;
KERNEL SPACE
bpf_prog2
bpf_prog1
load BPF prog !!
my_map
(Shared memory)
lat_map
(Shared memory)


tracex3_kern.c
clang / llc
tracex3_kern .o
bpf_prog1,
bpf_prog2
BPF bytecode
ELF parsing
bpf_load.o + tracex3_user.o
=> tracex3
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
bpf() SYSCALL
KERNEL SPACE
bpf_prog2
bpf_prog1
my_map
(Shared memory)
lat_map
(Shared memory)


tracex3_kern.c
clang / llc
tracex3_kern .o
bpf_prog1,
bpf_prog2
BPF bytecode
ELF parsing
bpf_load.o + tracex3_user.o
=> tracex3
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
perf_event_open()
ioctl() SYSCALL
KERNEL SPACE
bpf_prog2
attach !!
bpf_prog1
attach !!
my_map
(Shared memory)
lat_map
(Shared memory)


tracex3_kern.c
clang / llc
tracex3_kern .o
bpf_prog1,
bpf_prog2
BPF bytecode
ELF parsing
bpf_load.o + tracex3_user.o
=> tracex3
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
perf_event_open()
ioctl() SYSCALL
KERNEL SPACE
bpf_prog2
attach !!
bpf_prog1
attach !!
entry point of
blk_account_io_completion()
entry point of blk_mq_start_request()
to which the BPF prog is attached
my_map
(Shared memory)
lat_map
(Shared memory)


tracex3_kern.c
clang / llc
tracex3_kern .o
bpf_prog1,
bpf_prog2
BPF bytecode
ELF parsing
bpf_load.o + tracex3_user.o
=> tracex3
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
bpf() SYSCALL
KERNEL SPACE
bpf_prog2
bpf_prog1
entry point of
blk_account_io_completion()
entry point of blk_mq_start_request()
to which the BPF prog is attached
my_map
(Shared memory)
lat_map
(Shared memory)


tracex3_kern.c
clang / llc
tracex3_kern .o
bpf_prog1,
bpf_prog2
ELF parsing
BPF bytecode
bpf_load.o + tracex3_user.o
=> tracex3
BPF library: libbpf
prog/map
load, attach, control
BPF agent
(User App)
sharing (user / kernel)
map read/write !!
bpf() SYSCALL
KERNEL SPACE
my_map
(Shared memory)
lat_map
(Shared memory)
bpf_prog2
bpf_prog1
entry point of
blk_account_io_completion()
entry point of blk_mq_start_request()
to which the BPF prog is attached

BPF hands-on tutorials

Installations & Settings

(FYI, It isn't needed in the given vbox image)

vbox image(.ova) download link:
https://www.dropbox.com/s/z3b9faj7ng2wkjy/vault19_BPF_tutorials.ova?dl=1

ID: kosslab PW: kosslab SSH: ip:127.0.0.1 port:2222

Installations: BPF development environment

```
# Ubuntu 17.04 or later:
#
$ sudo apt-get install -y make gcc libssl-dev bc libelf-dev libcap-dev \
  clang gcc-multilib llvm libncurses5-dev git pkg-config libmnl bison flex \
  graphviz


# Or,
# Fedora 25 or later:
#
$ sudo dnf install -y git gcc ncurses-devel elfutils-libelf-devel bc\
  openssl-devel libcap-devel clang llvm graphviz bison flex glibc-static
```

https://cilium.readthedocs.io/en/latest/bpf/#development-environment

Installations: This BPF tutorial environment

```
# For parsing debug info of 'vmlinux'
#
$ sudo apt-get install dwarves


# Download BPF examples 'vault19_bpf_tutorial'
#
$ git clone https://github.com/kernel-digging/vault19_bpf_tutorial.git
```

BPF hands-on tutorials:

Examples: see ‘vault19_bpf_tutorial’

```
$ cd ~/git/vault19-bpf-tutorial
$ ls
linux-samples-bpf-Makfile.patch  stage06_max_latency_stacktrace
stage01_submit_bio_call_time     stage07_latency_other_section
stage02_bio_endio_call_time      stage08_pagecache_miss_counts
stage03_bio_latency_map          stage09_pagecache_miss_pid_filter
stage04_max_latency              stage10_ctracer-data+func_tracing
stage05_max_latency_sector
```

Preparation: modify samples/bpf/Makefile

```
# just check the diff of Makefile
# see both BPF agents(user) and programs(kernel)
$ cat ~/git/vault19-bpf-tutorial/linux-samples-bpf-Makfile.patch

# The Makefile is already modified in given vbox image
$ cd ~/git/linux/samples/bpf

$ vim Makefile
```

```
diff --git a/samples/bpf/Makefile b/samples/bpf/Makefile
index db1a91dfa702..55c90a7d92d8 100644
--- a/samples/bpf/Makefile
+++ b/samples/bpf/Makefile
@@ -53,6 +53,12 @@ hostprogs-y += xdpsock
 hostprogs-y += xdp_fwd
 hostprogs-y += task_fd_query
 hostprogs-y += xdp_sample_pkts
+#
+# Vault19 BPF programming tutorial: BPF agent(user)
+#
+hostprogs-y += vault19_bio_trace
+hostprogs-y += vault19_pagecache_trace
+hostprogs-y += vault19_ctracer
...
+#
+# Vault19 BPF programming tutorial: BPF program (kernel)
+#
+always += vault19_bio_trace_kern.o
+always += vault19_pagecache_trace_kern.o
...
```

```
diff --git a/samples/bpf/Makefile b/samples/bpf/Makefile
index db1a91dfa702..55c90a7d92d8 100644
--- a/samples/bpf/Makefile
+++ b/samples/bpf/Makefile
@@ -53,6 +53,12 @@ hostprogs-y += xdpsock
 hostprogs-y += xdp_fwd
 hostprogs-y += task_fd_query
 hostprogs-y += xdp_sample_pkts
+#
+# Vault19 BPF programming tutorial: BPF agent(user)
+#
+hostprogs-y += vault19_bio_trace
+hostprogs-y += vault19_pagecache_trace
+hostprogs-y += vault19_ctracer
...
+#
+# Vault19 BPF programming tutorial: BPF program (kernel)
+#
+always += vault19_bio_trace_kern.o
+always += vault19_pagecache_trace_kern.o
...
```

Stage 01: Record the time submit_bio() is called

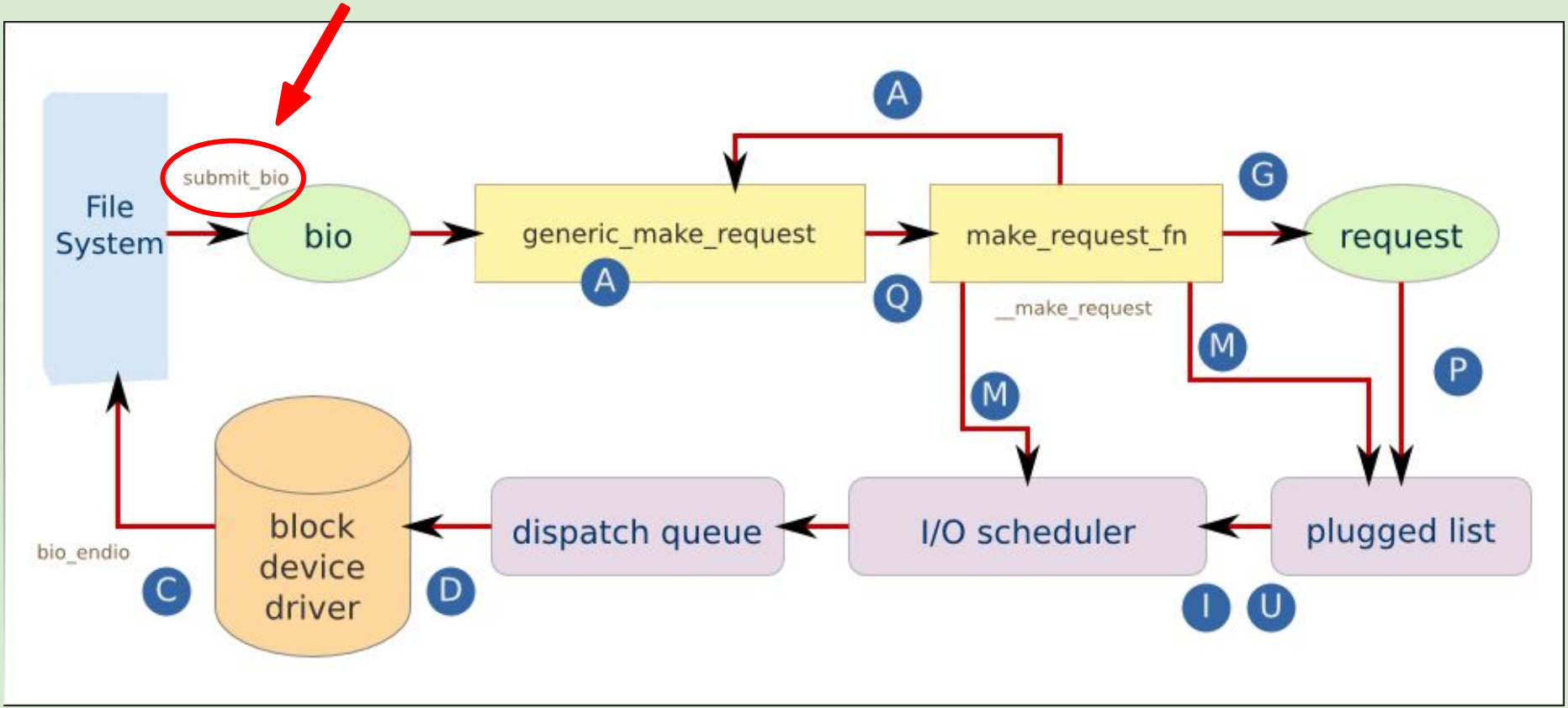


image from http://studyfoss.egloos.com/5585801

```
# Stage 01: submit_bio() call time
$
$ sudo ./vault19_bio_trace


     jbd2/sda2-8-1270  [000] ....629943140500: 0: submit_bio() called: 629943132973
     jbd2/sda2-8-1270  [000] ....629943240881: 0: submit_bio() called: 629943238515
     jbd2/sda2-8-1270  [000] ....629943246169: 0: submit_bio() called: 629943244798
     jbd2/sda2-8-1270  [000] ....629943249928: 0: submit_bio() called: 629943248578
     jbd2/sda2-8-1270  [000] ....629943253554: 0: submit_bio() called: 629943252215

            sshd-4287  [000] ....629944127921: 0: submit_bio() called: 629944122505

    kworker/u2:1-4141  [000] ....630966317623: 0: submit_bio() called: 630966309278
    kworker/u2:1-4141  [000] ....630966420007: 0: submit_bio() called: 630966417869
    kworker/u2:1-4141  [000] ....630966428512: 0: submit_bio() called: 630966426710
    kworker/u2:1-4141  [000] ....630966438142: 0: submit_bio() called: 630966436711
    kworker/u2:1-4141  [000] ....630966444203: 0: submit_bio() called: 630966442804
    kworker/u2:1-4141  [000] ....630966503472: 0: submit_bio() called: 630966501795
    kworker/u2:1-4141  [000] ....630966511920: 0: submit_bio() called: 630966510488
```

BPF hands-on tutorials:

Just run it !!
(copy & make & run)

Stage 01: Record the time submit_bio() is called

```
$ cd ~/git/vault19_bpf_tutorial/stage01_submit_bio_call_time
$ ls
vault19_bio_trace_kern.c  vault19_bio_trace_user.c

$ cp ./vault19_bio_trace_kern.c ~/git/linux/samples/bpf/
$ cp ./vault19_bio_trace_user.c ~/git/linux/samples/bpf/

$ cd ~/git/linux/samples/bpf/

$ make
$ sudo ./vault19_bio_trace
```

BPF hands-on tutorials:

Check the src files !!
(How does it work ?)

Stage 01: Record the time submit_bio() is called

```
$ cd ~/git/vault19_bpf_tutorial/stage01_submit_bio_call_time
$ ls
vault19_bio_trace_kern.c  vault19_bio_trace_user.c

$ less vault19_bio_trace_kern.c

$ less vault19_bio_trace_user.c
```

```diff
diff /dev/null stage01_submit_bio_call_time/vault19_bio_trace_kern.c
--- /dev/null
+++ stage01_*/vault19_bio_trace_kern.c
@@ -1 +1,17 @@
+#include <linux/ptrace.h>
+#include <linux/version.h>
+#include <uapi/linux/bpf.h>
+#include "bpf_helpers.h"

+SEC("kprobe/submit_bio")
+int submit_bio_entry(struct pt_regs *ctx)
+{
+        char fmt [] = "submit_bio() called: %llu\n";
+        u64 start_time = bpf_ktime_get_ns();
...

+        bpf_trace_printk(fmt, sizeof(fmt), start_time);
+        return 0;
+}

...
```

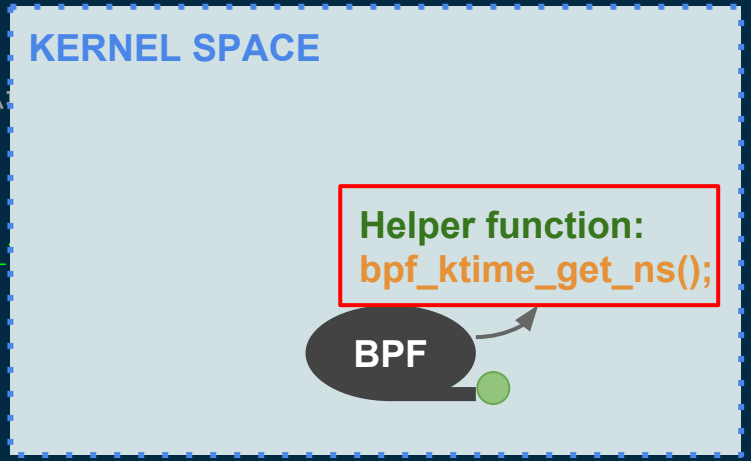
```
diff /dev/null stage01_submit_bio_call_time/vault19_bio_trace_kern.c
--- /dev/null
+++ stage01_*/vault19_bio_trace_kern.c
@@ -1 +1,17 @@
+#include <linux/ptrace.h>
+#include <linux/version.h>
+#include <uapi/linux/bpf.h>
+#include "bpf_helpers.h"

+SEC("kprobe/submit_bio")
+int submit_bio_entry(struct pt_regs *ctx)
+{
+        char fmt [] = "submit_bio() called: %llu\n";
+        u64 start_time = bpf_ktime_get_ns();
...

+        bpf_trace_printk(fmt, sizeof(fmt), start_time);
+        return 0;
+}

...
```

```
diff /dev/null stage01_submit_bio_call_time/vault19_bio_trace_kern.c
--- /dev/null
+++ stage01_*/vault19_bio_trace_kern.c
@@ -1 +1,17 @@
+#include <linux/ptrace.h>
+#include <linux/version.h>
+#include <uapi/linux/bpf.h>
+#include "bpf_helpers.h"

+SEC("kprobe/submit_bio")
+int submit_bio_entry(struct pt_regs *ctx)
+{
+        char fmt [] = "submit_bio() called: %llu\n";
+        u64 start_time = bpf_ktime_get_ns();
...

+        bpf_trace_printk(fmt, sizeof(fmt), start_time);
+        return 0;
+}

...
```

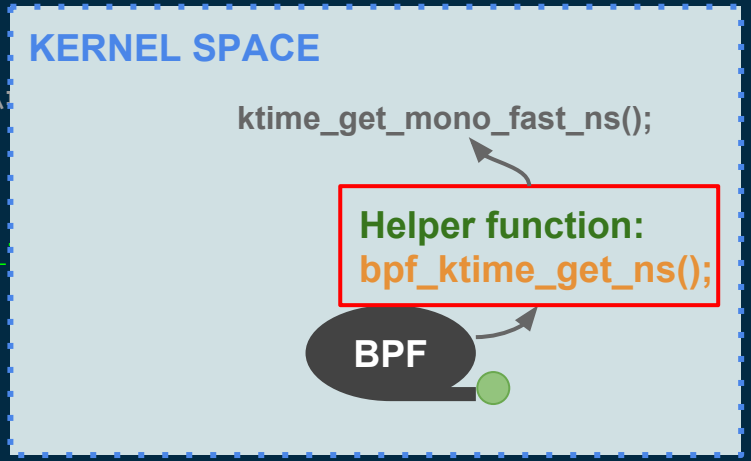
Attached to the entry point of submit_bio() function

```diff
diff /dev/null stage01_submit_bio_call_time/vault19_bio_trace_kern.c
--- /dev/null
+++ stage01_*/vault19_bio_trace_kern.c
@@ -1 +1,17 @@
+#include <linux/ptrace.h>
+#include <linux/version.h>
+#include <uapi/linux/bpf.h>
+#include "bpf_helpers.h"

+SEC("kprobe/submit_bio")
+int submit_bio_entry(struct pt_regs *ctx)
+{
+        char fmt [] = "submit_bio() called: %llu\n";
+        u64 start_time = bpf_ktime_get_ns();
...

+        bpf_trace_printk(fmt, sizeof(fmt), start_time);
+        return 0;
+}

...
```

```
diff /dev/null stage01_submit_bio_call_time/vault19_bio_trace_kern.c
--- /dev/null
+++ stage01_*/vault19_bio_trace_kern.c
@@ -1 +1,17 @@
+#include <linux/ptrace.h>
+#include <linux/version.h>
+#include <uapi/linux/bpf.h>
+#include "bpf_helpers.h"

+SEC("kprobe/submit_bio")
+int submit_bio_entry(struct pt_regs *ctx)
+{
+        char fmt [] = "submit_bio() called: %llu\
+        u64 start_time = bpf_ktime_get_ns();
...

+        bpf_trace_printk(fmt, sizeof(fmt), start_
+        return 0;
+}

...
```



```
diff /dev/null stage01_submit_bio_call_time/vault19_bio_trace_kern.c
--- /dev/null
+++ stage01_*/vault19_bio_trace_kern.c
@@ -1 +1,17 @@
+#include <linux/ptrace.h>
+#include <linux/version.h>
+#include <uapi/linux/bpf.h>
+#include "bpf_helpers.h"

+SEC("kprobe/submit_bio")
+int submit_bio_entry(struct pt_regs *ctx)
+{
+        char fmt [] = "submit_bio() called: %llu\
+        u64 start_time = bpf_ktime_get_ns();
...

+        bpf_trace_printk(fmt, sizeof(fmt), start_
+        return 0;
+}

...
```



```diff
diff /dev/null stage01_submit_bio_call_time/vault19_bio_trace_kern.c
--- /dev/null
+++ stage01_*/vault19_bio_trace_kern.c
@@ -1 +1,17 @@
+#include <linux/ptrace.h>
+#include <linux/version.h>
+#include <uapi/linux/bpf.h>
+#include "bpf_helpers.h"

+SEC("kprobe/submit_bio")
+int submit_bio_entry(struct pt_regs *ctx)
+{
+        char fmt [] = "submit_bio() called: %llu\n";
+        u64 start_time = bpf_ktime_get_ns();
...

+        bpf_trace_printk(fmt, sizeof(fmt), start_time);
+        return 0;
+}

...
```

```
diff /dev/null stage01_submit_bio_call_time/vault19_bio_trace_kern.c
--- /dev/null
+++ stage01_*/vault19_bio_trace_kern.c
@@ -1 +1,17 @@
+#include <linux/ptrace.h>
+#include <linux/version.h>
+#include <uapi/linux/bpf.h>
+#include "bpf_helpers.h"

+SEC("kprobe/submit_bio")
+int submit_bio_entry(struct pt_regs *ctx)
+{
+        char fmt [] = "submit_bio() called: %llu\n";
+        u64 start_time = bpf_ktime_get_ns();
...

+        bpf_trace_printk(fmt, sizeof(fmt), start_time);
+        return 0;
+}

...
```

__trace_printk();

Helper function

```diff
diff /dev/null stage01_submit_bio_call_time/vault19_bio_trace_user.c
--- /dev/null
+++ stage01_*/vault19_bio_trace_user.c
@@ -0,0 +1,21 @@
+#include <linux/bpf.h>
+#include <bpf/bpf.h>
+#include "bpf_load.h"
+
+int main(int argc, char **argv)
+{
...
+    snprintf(filename, sizeof(filename), "%s_kern.o", argv[0]);
+
+    if (load_bpf_file(filename)) {
+        printf("%s", bpf_log_buf);
+        return 1;
+    }
...
+    read_trace_pipe();
+
+    return 0;
+}
```

```diff
diff /dev/null stage01_submit_bio_call_time/vault19_bio_trace_user.c
--- /dev/null
+++ stage01_*/vault19_bio_trace_user.c
@@ -0,0 +1,21 @@
+#include <linux/bpf.h>
+#include <bpf/bpf.h>
+#include "bpf_load.h"
+
+int main(int argc, char **argv)
+{
...
+    snprintf(filename, sizeof(filename), "%s_kern.o", argv[0]);
+
+    if (load_bpf_file(filename)) {
+        printf("%s", bpf_log_buf);
+        return 1;
+    }
...
+    read_trace_pipe();
+
+    return 0;
+}
```

```
diff /dev/null stage01_submit_bio_call_time/vault19_bio_trace_user.c
--- /dev/null
+++ stage01_*/vault19_bio_trace_user.c
@@ -0,0 +1,21 @@
+#include <linux/bpf.h>
+#include <bpf/bpf.h>
+#include "bpf_load.h"
+
+int main(int argc, char **argv)
+{
...
+    snprintf(filename, sizeof(filename), "%s_kern.o", argv[0]);
+
+    if (load_bpf_file(filename)) {
+        printf("%s", bpf_log_buf);
+        return 1;
+    }
...
+    read_trace_pipe();
+
+    return 0;
+}
```

vault19_bio_trace_kern.o

```diff
diff /dev/null stage01_submit_bio_call_time/vault19_bio_trace_user.c
--- /dev/null
+++ stage01_*/vault19_bio_trace_user.c
@@ -0,0 +1,21 @@
+#include <linux/bpf.h>
+#include <bpf/bpf.h>
+#include "bpf_load.h"
+
+int main(int argc, char **argv)
+{
...
+    snprintf(filename, sizeof(filename), "%s_kern.o", argv[0]);
+
+    if (load_bpf_file(filename)) {
+        printf("%s", bpf_log_buf);
+        return 1;
+    }
...
+    read_trace_pipe();
+
+    return 0;
+}
```

```
diff /dev/null stage01_submit_bio_call_time/vault19_bio_trace_user.c
--- /dev/null
+++ stage01_*/vault19_bio_trace_user.c
@@ -0,0 +1,21 @@
+#include <linux/bpf.h>
+#include <bpf/bpf.h>
+#include "bpf_load.h"
+
+int main(int argc, char **argv)
+{
...
+    snprintf(filename, sizeof(filename), "%s_kern.o", argv[0]);
+
+    if (load_bpf_file(filename)) {
+        printf("%s", bpf_log_buf);
+        return 1;
+    }
...
+    read_trace_pipe();
+
+    return 0;
+}
```

read & print

/sys/kernel/debug/tracing/trace_pipe (ftrace ring-buffer)

```diff
diff /dev/null stage01_submit_bio_call_time/vault19_bio_trace_user.c
--- /dev/null
+++ stage01_*/vault19_bio_trace_user.c
@@ -0,0 +1,21 @@
+#include <linux/bpf.h>
+#include <bpf/bpf.h>
+#include "bpf_load.h"
+
+int main(int argc, char **argv)
+{
...
+    snprintf(filename, sizeof(filename), "%s_kern.o", argv[0]);
+
+    if (load_bpf_file(filename)) {
+        printf("%s", bpf_log_buf);
+        return 1;
+    }
...
+    read_trace_pipe();
+
+    return 0;
+}
```

read & print

/sys/kernel/debug/tracing/trace_pipe (ftrace ring-buffer)

```
bpf_trace_printk(fmt, sizeof(fmt), start_time);
```

write

Stage 02: Record the time bio_endio() is called

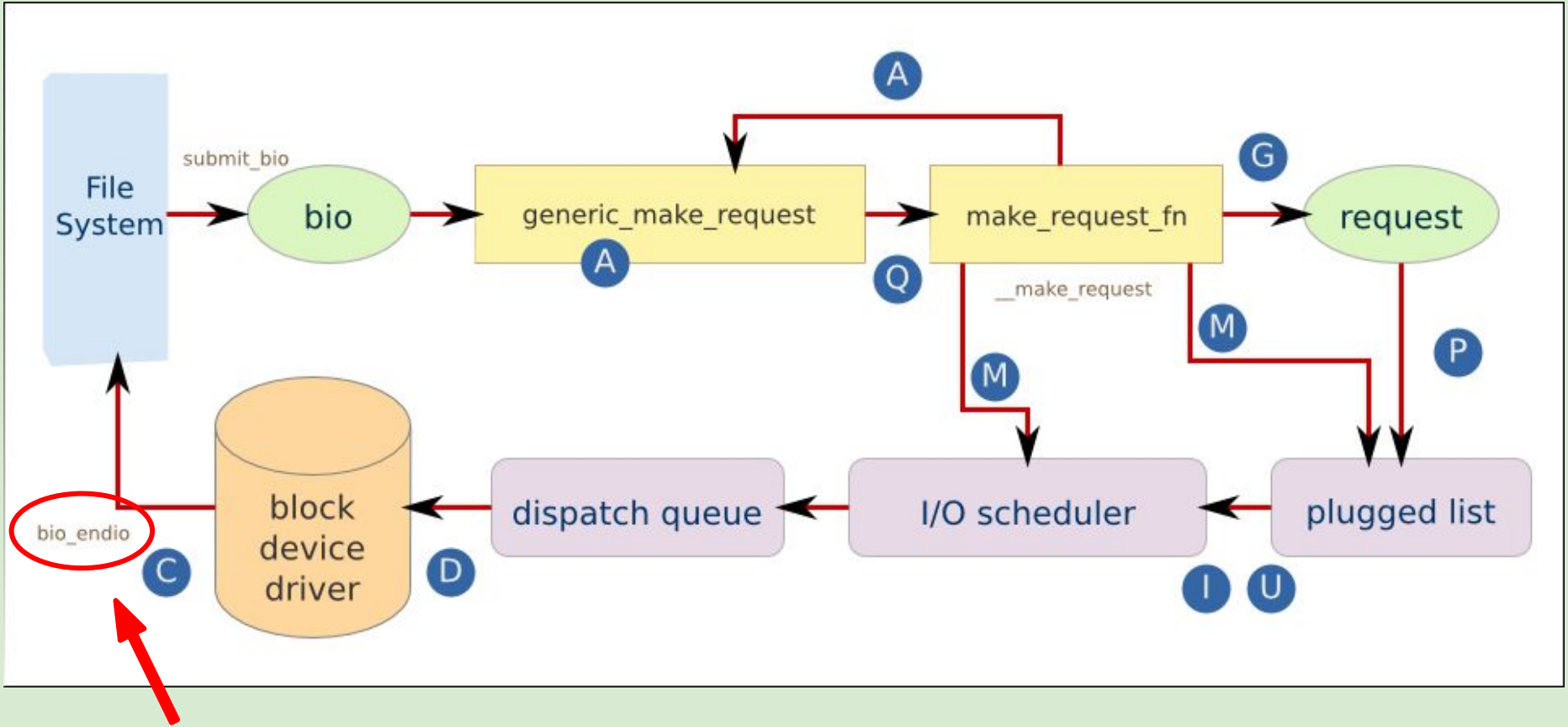


image from http://studyfoss.egloos.com/5585801

```
# Stage 02: bio_endio() call time
$
$ sudo ./vault19_bio_trace

          <idle>-0     [000] ..s.1095991863011: 0: bio_endio() called: 1095991854621
          <idle>-0     [000] ..s.1097974692821: 0: bio_endio() called: 1097974683065
          <idle>-0     [000] ..s.1098039309591: 0: bio_endio() called: 1098039294726

     jbd2/sda2-8-1270  [000] ....1098934216034: 0: submit_bio() called: 1098934210231
     jbd2/sda2-8-1270  [000] ....1098934287488: 0: submit_bio() called: 1098934285488
     jbd2/sda2-8-1270  [000] ....1098934292797: 0: submit_bio() called: 1098934291386
     jbd2/sda2-8-1270  [000] ....1098934296491: 0: submit_bio() called: 1098934295135

            sshd-1824  [000] ..s.1098934602276: 0: bio_endio() called: 1098934599687
            sshd-1824  [000] .Ns.1098934672300: 0: bio_endio() called: 1098934670634
            sshd-1824  [000] .Ns.1098934675405: 0: bio_endio() called: 1098934674029
            sshd-1824  [000] .Ns.1098934678226: 0: bio_endio() called: 1098934676931

     jbd2/sda2-8-1270  [000] ....1098934763515: 0: submit_bio() called: 1098934761525
          <idle>-0     [000] d.s.1098937113471: 0: bio_endio() called: 1098937108802
```

BPF hands-on tutorials:

Just run it !!
(copy & make & run)

Stage 02: Record the time bio_endio() is called

```
$ cd ~/git/vault19_bpf_tutorial/stage02_bio_endio_call_time
$ ls
vault19_bio_trace_kern.c  vault19_bio_trace_user.c

$ cp ./vault19_bio_trace_kern.c ~/git/linux/samples/bpf/

$ cd ~/git/linux/samples/bpf/

$ make
$ sudo ./vault19_bio_trace
```

Check the src files !!
(How does it work ?)

Stage 02: Record the time bio_endio() is called

```
$ cd ~/git/vault19_bpf_tutorial/stage02_bio_endio_call_time
$ ls
vault19_bio_trace_kern.c  vault19_bio_trace_user.c

$ diff stage01_submit_bio_call_time/vault19_bio_trace_kern.c  \
  ./vault19_bio_trace_kern.c
```

```diff
diff stage01_*/vault19_bio_trace_kern.c stage02_bio_endio_call_time/vault19_bio_trace_kern.c
--- stage01_*/vault19_bio_trace_kern.c
+++ stage02_*/vault19_bio_trace_kern.c
@@ -6,12 +6,22 @@
 SEC("kprobe/submit_bio")
 int submit_bio_entry(struct pt_regs *ctx)
 {
 ...
         return 0;
 }

+SEC("kprobe/bio_endio")
+int bio_endio_entry(struct pt_regs *ctx)
+{
+        char fmt [] = "bio_endio() called: %llu\n";
+        u64 end_time = bpf_ktime_get_ns();
+
+        bpf_trace_printk(fmt, sizeof(fmt), end_time);
+        return 0;
+}
+
```

```diff
diff stage01_*/vault19_bio_trace_kern.c stage02_bio_endio_call_time/vault19_bio_trace_kern.c
--- stage01_*/vault19_bio_trace_kern.c
+++ stage02_*/vault19_bio_trace_kern.c
@@ -6,12 +6,22 @@
 SEC("kprobe/submit_bio")
 int submit_bio_entry(struct pt_regs *ctx)
 {
 ...
         return 0;
 }

+SEC("kprobe/bio_endio")
+int bio_endio_entry(struct pt_regs *ctx)
+{
+        char fmt [] = "bio_endio() called: %llu\n";
+        u64 end_time = bpf_ktime_get_ns();
+
+        bpf_trace_printk(fmt, sizeof(fmt), end_time);
+        return 0;
+}
+
```

```
diff stage01_*/vault19_bio_trace_kern.c stage02_bio_endio_call_time/vault19_bio_trace_kern.c
--- stage01_*/vault19_bio_trace_kern.c
+++ stage02_*/vault19_bio_trace_kern.c
@@ -6,12 +6,22 @@
 SEC("kprobe/submit_bio")
 int submit_bio_entry(struct pt_regs *ctx)
 {
 ...
         return 0;
 }

+SEC("kprobe/bio_endio")
+int bio_endio_entry(struct pt_regs *ctx)
+{
+        char fmt [] = "bio_endio() called: %llu\n";
+        u64 end_time = bpf_ktime_get_ns();
+
+        bpf_trace_printk(fmt, sizeof(fmt), end_time);
+        return 0;
+}
+
```

Attached to the entry point of bio_endio() function

```diff
diff stage01_*/vault19_bio_trace_kern.c stage02_bio_endio_call_time/vault19_bio_trace_kern.c
--- stage01_*/vault19_bio_trace_kern.c
+++ stage02_*/vault19_bio_trace_kern.c
@@ -6,12 +6,22 @@
 SEC("kprobe/submit_bio")
 int submit_bio_entry(struct pt_regs *ctx)
 {
 ...
         return 0;
 }

+SEC("kprobe/bio_endio")
+int bio_endio_entry(struct pt_regs *ctx)
+{
+        char fmt [] = "bio_endio() called: %llu\n";
+        u64 end_time = bpf_ktime_get_ns();
+
+        bpf_trace_printk(fmt, sizeof(fmt), end_time);
+        return 0;
+}
+
```

ktime_get_mono_fast_ns();

Helper function

```diff
diff stage01_*/vault19_bio_trace_kern.c stage02_bio_endio_call_time/vault19_bio_trace_kern.c
--- stage01_*/vault19_bio_trace_kern.c
+++ stage02_*/vault19_bio_trace_kern.c
@@ -6,12 +6,22 @@
 SEC("kprobe/submit_bio")
 int submit_bio_entry(struct pt_regs *ctx)
 {
 ...
         return 0;
 }

+SEC("kprobe/bio_endio")
+int bio_endio_entry(struct pt_regs *ctx)
+{
+        char fmt [] = "bio_endio() called: %llu\n";
+        u64 end_time = bpf_ktime_get_ns();
+
+        bpf_trace_printk(fmt, sizeof(fmt), end_time);
+        return 0;
+}
+
```

__trace_printk();

Helper function

Stage 03: Calculate the bio latency between submit_bio() -> bio_endio()

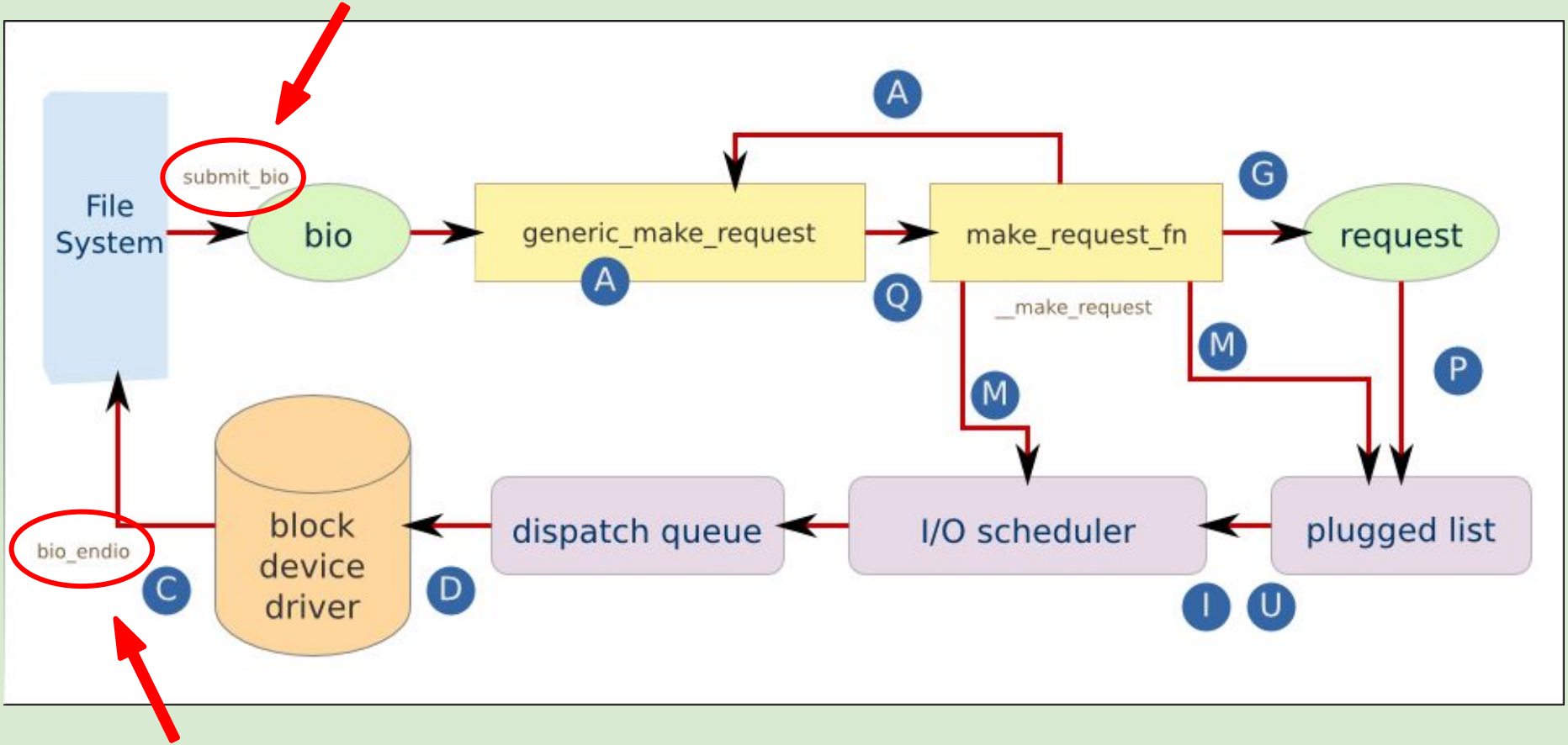


image from http://studyfoss.egloos.com/5585801

Calculate the bio latency: How does it work?

```
1 blk-core.c
/**
 * submit_bio - submit a bio to the block device layer for I/O
 * @bio: The &struct bio which describes the I/O
 *
 * submit_bio() is very similar in purpose to generic_make_request(), and
 * uses that function to do most of the work. Both are fairly rough
 * interfaces; @bio must be presetup and ready for I/O.
 *
 */
blk_qc_t submit_bio(struct bio *bio)
{
        /*
         * If it's a regular read/write or a barrier with data attached,
         * go through the normal accounting stuff before submission.
         */
        if (bio_has_data(bio)) {
                unsigned int count;

                if (unlikely(bio_op(bio) == REQ_OP_WRITE_SAME))
@
NORMAL  bpf   blk-core.c                                  64%   1167:1
"~/git/linux/block/blk-core.c" 1822L, 49756C
```

```
1 bio.c
 *   bio_endio() will end I/O on the whole bio. bio_endio() is the preferred
 *   way to end I/O on a bio. No one should call bi_end_io() directly on a
 *   bio unless they own it and thus know that it has an end_io function.
 *
 *   bio_endio() can be called several times on a bio that has been chained
 *   using bio_chain().  The ->bi_end_io() function will only be called the
 *   last time.  At this point the BLK_TRACE_COMPLETE tracing event will be
 *   generated if BIO_TRACE_COMPLETION is set.
 **/
void bio_endio(struct bio *bio)
{
again:
        if (!bio_remaining_done(bio))
                return;
        if (!bio_integrity_endio(bio))
                return;

        if (bio->bi_disk)
                rq_qos_done_bio(bio->bi_disk->queue, bio);
NORMAL  bpf   bio.c                                       81%   1759:1
"~/git/linux/block/bio.c" 2149L, 53553C
```

submit_bio (struct bio* bio)

block/blk-core.c: 1822L

bio_endio (struct bio* bio)

block/bio.c: 2149L

```
# Stage 03: bio latency map
$
$ sudo ./vault19_bio_trace

...
    kworker/u2:2-6375  [000] ....3627666075886: 0: submit_bio(bio=0xffff9f8b3ac80600) called: 3627666070384
    kworker/u2:2-6375  [000] ....3627666090408: 0: submit_bio(bio=0xffff9f8b3ae01f00) called: 3627666087021
    kworker/u2:2-6375  [000] ....3627666100463: 0: submit_bio(bio=0xffff9f8b3ac80780) called: 3627666097231
    kworker/u2:2-6375  [000] ....3627666111629: 0: submit_bio(bio=0xffff9f8b3ac800c0) called: 3627666107789

    kworker/u2:2-6375  [000] .Ns.3627666627687: 0: bio_endio (bio=0xffff9f8b3ac80600) called: 3627666622336
    kworker/u2:2-6375  [000] .Ns.3627666630359: 0: submit_bio() -> bio_endio() time duration: 551952 ns

    kworker/u2:2-6375  [000] .Ns.3627666646824: 0: bio_endio (bio=0xffff9f8b3ae01f00) called: 3627666641524
    kworker/u2:2-6375  [000] .Ns.3627666649432: 0: submit_bio() -> bio_endio() time duration: 554503 ns

    kworker/u2:2-6375  [000] .Ns.3627666662687: 0: bio_endio (bio=0xffff9f8b3ac80780) called: 3627666658135
    kworker/u2:2-6375  [000] .Ns.3627666665235: 0: submit_bio() -> bio_endio() time duration: 560904 ns

    kworker/u2:2-6375  [000] .Ns.3627666679470: 0: bio_endio (bio=0xffff9f8b3ac800c0) called: 3627666674895
    kworker/u2:2-6375  [000] .Ns.3627666682287: 0: submit_bio() -> bio_endio() time duration: 567106 ns

     jbd2/sda2-8-1270  [000] ....1435319638182: 0: submit_bio(bio=0xffff9f8b3ae01840) called: 1435319630175

            sshd-1824  [000] d.s11435322241050: 0: bio_endio (bio=0xffff9f8b3ae01840) called: 1435322230252
            sshd-1824  [000] d.s11435322281220: 0: submit_bio() -> bio_endio() time duration: 2600077 ns
```

BPF hands-on tutorials:

Just run it !!
(copy & make & run)

Stage 03: Calculate the bio latency between submit_bio() -> bio_endio()

```
$ cd ~/git/vault19_bpf_tutorial/stage03_bio_latency_map
$ ls
vault19_bio_trace_kern.c  vault19_bio_trace_user.c

$ cp ./vault19_bio_trace_kern.c ~/git/linux/samples/bpf/

$ cd ~/git/linux/samples/bpf/

$ make
$ sudo ./vault19_bio_trace
```

BPF hands-on tutorials:

Check the src files !!
(How does it work ?)

Stage 03: Calculate the bio latency between submit_bio() -> bio_endio()

```
$ cd ~/git/vault19_bpf_tutorial/stage03_bio_latency_map
$ ls
vault19_bio_trace_kern.c  vault19_bio_trace_user.c

$ diff stage02_bio_endio_call_time/vault19_bio_trace_kern.c  \
  ./vault19_bio_trace_kern.c
```

```diff
diff stage02_*/vault19_bio_trace_kern.c stage03_bio_latency_map/vault19_bio_trace_kern.c
--- stage02_*/vault19_bio_trace_kern.c
+++ stage03_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
+struct called_info {
+        u64 start;
+        u64 end;
+};

+struct bpf_map_def SEC("maps") called_info_map = {
+        .type = BPF_MAP_TYPE_HASH,
+        .key_size = sizeof(long),
+        .value_size = sizeof(struct called_info),
+        .max_entries = 4096,
+};

 SEC("kprobe/submit_bio")
 int submit_bio_entry(struct pt_regs *ctx)
...      u64 start_time = bpf_ktime_get_ns();
+        long bio_ptr = PT_REGS_PARM1(ctx);
+        struct called_info called_info = {
+                .start = start_time,
+                .end = 0
+        };

+        bpf_map_update_elem(&called_info_map, &bio_ptr, &called_info, BPF_ANY);
...
```

```diff
diff stage02_*/vault19_bio_trace_kern.c stage03_bio_latency_map/vault19_bio_trace_kern.c
--- stage02_*/vault19_bio_trace_kern.c
+++ stage03_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
+struct called_info {
+        u64 start;
+        u64 end;
+};

+struct bpf_map_def SEC("maps") called_info_map = {
+        .type = BPF_MAP_TYPE_HASH,
+        .key_size = sizeof(long),
+        .value_size = sizeof(struct called_info),
+        .max_entries = 4096,
+};

 SEC("kprobe/submit_bio")
 int submit_bio_entry(struct pt_regs *ctx)
...      u64 start_time = bpf_ktime_get_ns();
+        long bio_ptr = PT_REGS_PARM1(ctx);
+        struct called_info called_info = {
+                .start = start_time,
+                .end = 0
+        };

+        bpf_map_update_elem(&called_info_map, &bio_ptr, &called_info, BPF_ANY);
...
```

```diff
diff stage02_*/vault19_bio_trace_kern.c stage03_bio_latency_map/vault19_bio_trace_kern.c
--- stage02_*/vault19_bio_trace_kern.c
+++ stage03_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
+struct called_info {
+        u64 start;
+        u64 end;
+};

+struct bpf_map_def SEC("maps") called_info_map = {
+        .type = BPF_MAP_TYPE_HASH,
+        .key_size = sizeof(long),
+        .value_size = sizeof(struct called_info),
+        .max_entries = 4096,
+};

 SEC("kprobe/submit_bio")
 int submit_bio_entry(struct pt_regs *ctx)
...      u64 start_time = bpf_ktime_get_ns();
+        long bio_ptr = PT_REGS_PARM1(ctx);
+        struct called_info called_info = {
+                .start = start_time,
+                .end = 0
+        };

+        bpf_map_update_elem(&called_info_map, &bio_ptr, &called_info, BPF_ANY);
...
```

```diff
diff stage02_*/vault19_bio_trace_kern.c stage03_bio_latency_map/vault19_bio_trace_kern.c
--- stage02_*/vault19_bio_trace_kern.c
+++ stage03_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
+struct called_info {
+        u64 start;
+        u64 end;
+};

+struct bpf_map_def SEC("maps") called_info_map = {
+        .type = BPF_MAP_TYPE_HASH,
+        .key_size = sizeof(long),
+        .value_size = sizeof(struct called_info),
+        .max_entries = 4096,
+};

 SEC("kprobe/submit_bio")
 int submit_bio_entry(struct pt_regs *ctx)
...      u64 start_time = bpf_ktime_get_ns();
+        long bio_ptr = PT_REGS_PARM1(ctx);
+        struct called_info called_info = {
+                .start = start_time,
+                .end = 0
+        };

+        bpf_map_update_elem(&called_info_map, &bio_ptr, &called_info, BPF_ANY);
...
```

```
1 blk-core.c                                                          X
/**
 * submit_bio - submit a bio to the block device layer for I/O
 * @bio: The &struct bio which describes the I/O
 *
 * submit_bio() is very similar in purpose to generic_make_request(), and
 * uses that function to do most of the work. Both are fairly rough
 * interfaces; @bio must be presetup and ready for I/O.
 *
 */
blk_qc_t submit_bio(struct bio *bio)
{
        /*
         * If it's a regular read/write or a barrier with data attached,
         * go through the normal accounting stuff before submission.
         */
        if (bio_has_data(bio)) {
                unsigned int count;

                if (unlikely(bio_op(bio) == REQ_OP_WRITE_SAME))
@
NORMAL  bpf   blk-core.c                                   64%   1167:1
"~/git/linux/block/blk-core.c" 1822L, 49756C
```

```diff
diff stage02_*/vault19_bio_trace_kern.c stage03_bio_latency_map/vault19_bio_trace_kern.c
--- stage02_*/vault19_bio_trace_kern.c
+++ stage03_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
+struct called_info {
+        u64 start;
+        u64 end;
+};

+struct bpf_map_def SEC("maps") called_info_map = {
+        .type = BPF_MAP_TYPE_HASH,
+        .key_size = sizeof(long),
+        .value_size = sizeof(struct called_info),
+        .max_entries = 4096,
+};

 SEC("kprobe/submit_bio")
 int submit_bio_entry(struct pt_regs *ctx)
...      u64 start_time = bpf_ktime_get_ns();
+        long bio_ptr = PT_REGS_PARM1(ctx);
+        struct called_info called_info = {
+                .start = start_time,
+                .end = 0
+        };

+        bpf_map_update_elem(&called_info_map, &bio_ptr, &called_info, BPF_ANY);
...
```

```diff
diff stage02_*/vault19_bio_trace_kern.c stage03_bio_latency_map/vault19_bio_trace_kern.c
--- stage02_*/vault19_bio_trace_kern.c
+++ stage03_*/vault19_bio_trace_kern.c
+struct called_info { u64 start; u64 end; };

+struct bpf_map_def SEC("maps") called_info_map = {
+        .type = BPF_MAP_TYPE_HASH,
+        .key_size = sizeof(long),
+        .value_size = sizeof(struct called_info),
+        .max_entries = 4096,
+};

 SEC("kprobe/bio_endio")
 int bio_endio_entry(struct pt_regs *ctx)
...      u64 end_time = bpf_ktime_get_ns();
+        long bio_ptr = PT_REGS_PARM1(ctx);
+        struct called_info *called_info;
+        u64 time_duration;
+
+        called_info = bpf_map_lookup_elem(&called_info_map, &bio_ptr);
+        if (!called_info)
+                return 0;
+
+        called_info->end = end_time;
+        time_duration = called_info->end - called_info->start;
...
```

```diff
diff stage02_*/vault19_bio_trace_kern.c stage03_bio_latency_map/vault19_bio_trace_kern.c
--- stage02_*/vault19_bio_trace_kern.c
+++ stage03_*/vault19_bio_trace_kern.c
+struct called_info { u64 start; u64 end; };

+struct bpf_map_def SEC("maps") called_info_map = {
+        .type = BPF_MAP_TYPE_HASH,
+        .key_size = sizeof(long),
+        .value_size = sizeof(struct called_info),
+        .max_entries = 4096,
+};

 SEC("kprobe/bio_endio")
 int bio_endio_entry(struct pt_regs *ctx)
...      u64 end_time = bpf_ktime_get_ns();
+        long bio_ptr = PT_REGS_PARM1(ctx);
+        struct called_info *called_info;
+        u64 time_duration;
+
+        called_info = bpf_map_lookup_elem(&called_info_map, &bio_ptr);
+        if (!called_info)
+                return 0;
+
+        called_info->end = end_time;
+        time_duration = called_info->end - called_info->start;
...
```

```diff
diff stage02_*/vault19_bio_trace_kern.c stage03_bio_latency_map/vault19_bio_trace_kern.c
--- stage02_*/vault19_bio_trace_kern.c
+++ stage03_*/vault19_bio_trace_kern.c
+struct called_info { u64 start; u64 end; };

+struct bpf_map_def SEC("maps") called_info_map = {
+        .type = BPF_MAP_TYPE_HASH,
+        .key_size = sizeof(long),
+        .value_size = sizeof(struct called_info),
+        .max_entries = 4096,
+};

 SEC("kprobe/bio_endio")
 int bio_endio_entry(struct pt_regs *ctx)
...      u64 end_time = bpf_ktime_get_ns();
+        long bio_ptr = PT_REGS_PARM1(ctx);
+        struct called_info *called_info;
+        u64 time_duration;
+
+        called_info = bpf_map_lookup_elem(&called_info_map
+        if (!called_info)
+                return 0;
+
+        called_info->end = end_time;
+        time_duration = called_info->end - called_info->start;
...
```

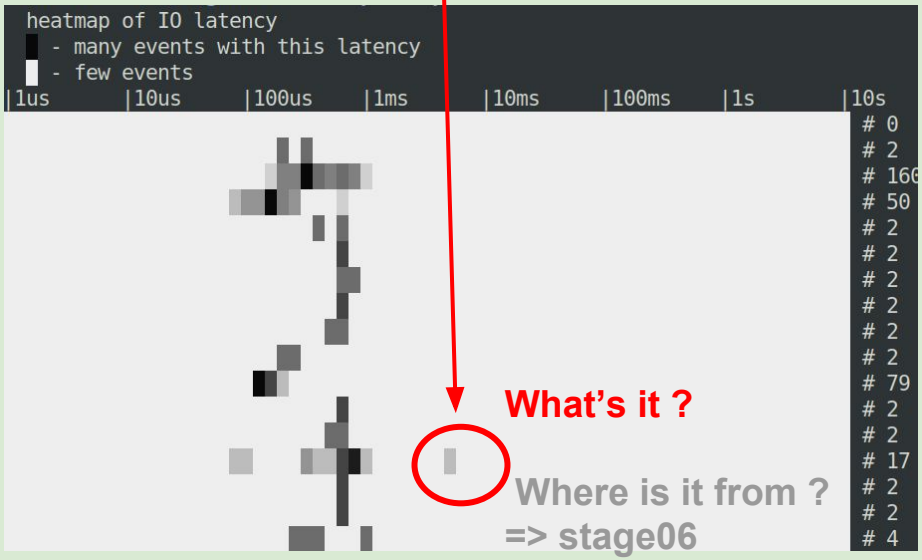
```
1 bio.c                                                                    X
 *   bio_endio() will end I/O on the whole bio. bio_endio() is the preferred
 *   way to end I/O on a bio. No one should call bi_end_io() directly on a
 *   bio unless they own it and thus know that it has an end_io function.
 *
 *   bio_endio() can be called several times on a bio that has been chained
 *   using bio_chain().  The ->bi_end_io() function will only be called the
 *   last time.  At this point the BLK_TA_COMPLETE tracing event will be
 *   generated if BIO_TRACE_COMPLETION is set.
 **/
void bio_endio(struct bio *bio)
{
again:
        if (!bio_remaining_done(bio))
                return;
        if (!bio_integrity_endio(bio))
                return;

        if (bio->bi_disk)
                rq_qos_done_bio(bio->bi_disk->queue, bio);
NORMAL  bpf   bio.c                                          81%   1759:1
"~/git/linux/block/bio.c" 2149L, 53553C
```

```diff
diff stage02_*/vault19_bio_trace_kern.c stage03_bio_latency_map/vault19_bio_trace_kern.c
--- stage02_*/vault19_bio_trace_kern.c
+++ stage03_*/vault19_bio_trace_kern.c
+struct called_info { u64 start; u64 end; };

+struct bpf_map_def SEC("maps") called_info_map = {
+        .type = BPF_MAP_TYPE_HASH,
+        .key_size = sizeof(long),
+        .value_size = sizeof(struct called_info),
+        .max_entries = 4096,
+};

 SEC("kprobe/bio_endio")
 int bio_endio_entry(struct pt_regs *ctx)
...      u64 end_time = bpf_ktime_get_ns();
+        long bio_ptr = PT_REGS_PARM1(ctx);
+        struct called_info *called_info;
+        u64 time_duration;
+
+        called_info = bpf_map_lookup_elem(&called_info_map, &bio_ptr);
+        if (!called_info)
+                return 0;
+
+        called_info->end = end_time;
+        time_duration = called_info->end - called_info->start;
...
```

```diff
diff stage02_*/vault19_bio_trace_kern.c stage03_bio_latency_map/vault19_bio_trace_kern.c
--- stage02_*/vault19_bio_trace_kern.c
+++ stage03_*/vault19_bio_trace_kern.c
+struct called_info { u64 start; u64 end; };

+struct bpf_map_def SEC("maps") called_info_map = {
+        .type = BPF_MAP_TYPE_HASH,
+        .key_size = sizeof(long),
+        .value_size = sizeof(struct called_info),
+        .max_entries = 4096,
+};

 SEC("kprobe/bio_endio")
 int bio_endio_entry(struct pt_regs *ctx)
...      u64 end_time = bpf_ktime_get_ns();
+        long bio_ptr = PT_REGS_PARM1(ctx);
+        struct called_info *called_info;
+        u64 time_duration;
+
+        called_info = bpf_map_lookup_elem(&called_info_map, &bio_ptr);
+        if (!called_info)
+                return 0;
+
+        called_info->end = end_time;
+        time_duration = called_info->end - called_info->start;
...
```

Stage 04: Get the max latency between submit_bio() -> bio_endio()

Stage 04: Get the max latency between submit_bio() -> bio_endio()



```
# Stage 04: max latency
$
$ sudo ./vault19_bio_trace

...
     jbd2/sda2-8-1270  [000] ....3971353869415: 0: submit_bio(bio=0xffff9f8b3b27fb40) called: 3971356075458
     jbd2/sda2-8-1270  [000] ....3971353958480: 0: submit_bio(bio=0xffff9f8b3ac80840) called: 3971353956214
     jbd2/sda2-8-1270  [000] ....3971353964818: 0: submit_bio(bio=0xffff9f8b3ae01f00) called: 3971353963729

            sshd-1824  [000] ..s.3971354074711: 0: bio_endio (bio=0xffff9f8b3b27fb40) called: 3971356932073
            sshd-1824  [000] ..s.3971354091868: 0: submit_bio() -> bio_endio() time duration: 1148038 ns

            sshd-1824  [000] ..s.3971354095221: 0: bio_endio (bio=0xffff9f8b3ac80840) called: 3971354094035
            sshd-1824  [000] ..s.3971354095854: 0: submit_bio() -> bio_endio() time duration: 137821 ns

            sshd-1824  [000] ..s.3971354103669: 0: bio_endio (bio=0xffff9f8b3ae01f00) called: 3971354102638
            sshd-1824  [000] ..s.3971354104304: 0: submit_bio() -> bio_endio() time duration: 138909 ns

     jbd2/sda2-8-1270  [000] ....3971354121261: 0: submit_bio(bio=0xffff9f8b3ac80d80) called: 3971354119379

          <idle>-0     [000] d.s.3971355270572: 0: bio_endio (bio=0xffff9f8b3ac80d80) called: 3971355267417
          <idle>-0     [000] d.s.3971355282406: 0: submit_bio() -> bio_endio() time duration: 1148038 ns

^C
=====================================================
From: submit_bio(bio=0xffff9f8b3b27fb40) 3971356075458
To  : bio_endio (bio=0xffff9f8b3b27fb40) 3971356932073
Max latency 1148038 ns
=====================================================
```

BPF hands-on tutorials:

Just run it !!
(copy & make & run)

Stage 04: Get the max latency between submit_bio() -> bio_endio()

```
$ cd ~/git/vault19_bpf_tutorial/stage04_max_latency
$ ls
vault19_bio_trace_kern.c  vault19_bio_trace_user.c

$ cp ./vault19_bio_trace_kern.c ~/git/linux/samples/bpf/
$ cp ./vault19_bio_trace_user.c ~/git/linux/samples/bpf/

$ cd ~/git/linux/samples/bpf/

$ make
$ sudo ./vault19_bio_trace
```

BPF hands-on tutorials:

Check the src files !!
(How does it work ?)

Stage 04: Get the max latency between submit_bio() -> bio_endio()

```
$ cd ~/git/vault19_bpf_tutorial/stage04_max_latency
$ ls
vault19_bio_trace_kern.c  vault19_bio_trace_user.c

$ diff stage03_bio_latency_map/vault19_bio_trace_kern.c \
  ./vault19_bio_trace_kern.c

$ diff stage03_bio_latency_map/vault19_bio_trace_user.c \
  ./vault19_bio_trace_user.c
```

```diff
diff stage03_*/vault19_bio_trace_kern.c stage04_max_latency/vault19_bio_trace_kern.c
--- stage03_*/vault19_bio_trace_kern.c
+++ stage04_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
+/* max_latency_info[0]: key == bio_ptr,
+ * max_latency_info[1]: val == max_time_duration */
+struct bpf_map_def SEC("maps") max_latency_info = {
+        .type = BPF_MAP_TYPE_ARRAY,
+        .key_size = sizeof(u32),
+        .value_size = sizeof(u64),
+        .max_entries = 2
+}
 SEC("kprobe/bio_endio")
 int bio_endio_entry(struct pt_regs *ctx)
         long bio_ptr = PT_REGS_PARM1(ctx);
         called_info = bpf_map_lookup_elem(&called_info_map, &bio_ptr);
         time_duration = called_info->end - called_info->start;
...
+        u64 *max_time_duration;
+        max_time_duration = bpf_map_lookup_elem(&max_latency_info, &val_idx);
+
+        if (max_time_duration && (time_duration <= *max_time_duration))
+                return 0;
+
+        bpf_map_update_elem(&max_latency_info, &key_idx, &bio_ptr, BPF_ANY);
+        bpf_map_update_elem(&max_latency_info, &val_idx, &time_duration, BPF_ANY);
```

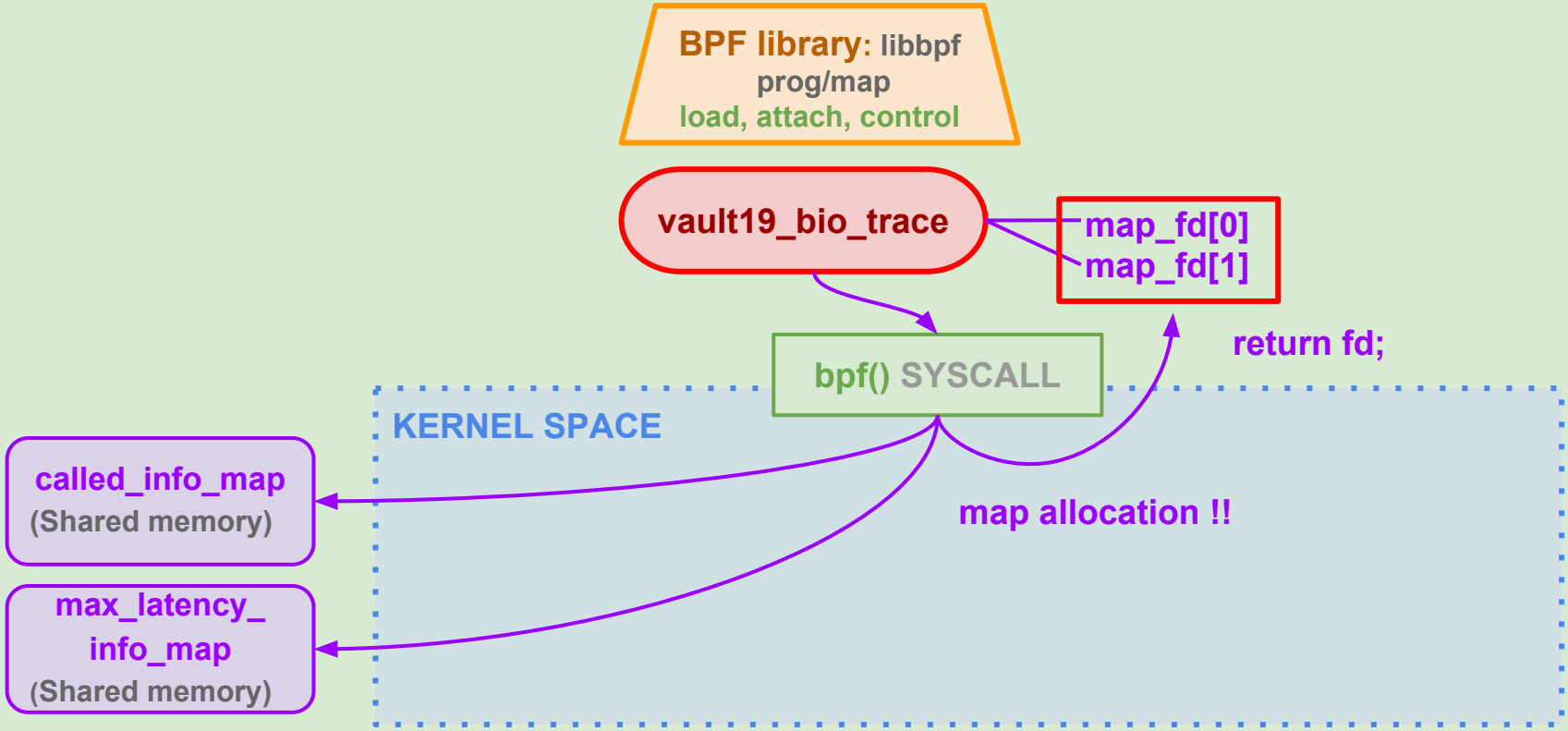
```diff
diff stage03_*/vault19_bio_trace_kern.c stage04_max_latency/vault19_bio_trace_kern.c
--- stage03_*/vault19_bio_trace_kern.c
+++ stage04_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
+/* max_latency_info[0]: key == bio_ptr,
+ * max_latency_info[1]: val == max_time_duration */
+struct bpf_map_def SEC("maps") max_latency_info = {
+        .type = BPF_MAP_TYPE_ARRAY,
+        .key_size = sizeof(u32),
+        .value_size = sizeof(u64),
+        .max_entries = 2
+}
 SEC("kprobe/bio_endio")
 int bio_endio_entry(struct pt_regs *ctx)
         long bio_ptr = PT_REGS_PARM1(ctx);
         called_info = bpf_map_lookup_elem(&called_info_map, &bio_ptr);
         time_duration = called_info->end - called_info->start;
...
+        u64 *max_time_duration;
+        max_time_duration = bpf_map_lookup_elem(&max_latency_info, &val_idx);
+
+        if (max_time_duration && (time_duration <= *max_time_duration))
+                return 0;
+
+        bpf_map_update_elem(&max_latency_info, &key_idx, &bio_ptr, BPF_ANY);
+        bpf_map_update_elem(&max_latency_info, &val_idx, &time_duration, BPF_ANY);
```

```diff
diff stage03_*/vault19_bio_trace_kern.c stage04_max_latency/vault19_bio_trace_kern.c
--- stage03_*/vault19_bio_trace_kern.c
+++ stage04_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
+/* max_latency_info[0]: key == bio_ptr,
+ * max_latency_info[1]: val == max_time_duration */
+struct bpf_map_def SEC("maps") max_latency_info = {
+        .type = BPF_MAP_TYPE_ARRAY,
+        .key_size = sizeof(u32),
+        .value_size = sizeof(u64),
+        .max_entries = 2
+}
 SEC("kprobe/bio_endio")
 int bio_endio_entry(struct pt_regs *ctx)
         long bio_ptr = PT_REGS_PARM1(ctx);
         called_info = bpf_map_lookup_elem(&called_info_map, &bio_ptr);
         time_duration = called_info->end - called_info->start;
...
+        u64 *max_time_duration;
+        max_time_duration = bpf_map_lookup_elem(&max_latency_info, &val_idx);
+
+        if (max_time_duration && (time_duration <= *max_time_duration))
+                return 0;
+
+        bpf_map_update_elem(&max_latency_info, &key_idx, &bio_ptr, BPF_ANY);
+        bpf_map_update_elem(&max_latency_info, &val_idx, &time_duration, BPF_ANY);
```

```diff
diff stage03_*/vault19_bio_trace_kern.c stage04_max_latency/vault19_bio_trace_kern.c
--- stage03_*/vault19_bio_trace_kern.c
+++ stage04_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
+/* max_latency_info[0]: key == bio_ptr,
+ * max_latency_info[1]: val == max_time_duration */
+struct bpf_map_def SEC("maps") max_latency_info = {
+        .type = BPF_MAP_TYPE_ARRAY,
+        .key_size = sizeof(u32),
+        .value_size = sizeof(u64),
+        .max_entries = 2
+}
 SEC("kprobe/bio_endio")
 int bio_endio_entry(struct pt_regs *ctx)
         long bio_ptr = PT_REGS_PARM1(ctx);
         called_info = bpf_map_lookup_elem(&called_info_map, &bio_ptr);
         time_duration = called_info->end - called_info->start;
...
+        u64 *max_time_duration;
+        max_time_duration = bpf_map_lookup_elem(&max_latency_info, &val_idx);
+
+        if (max_time_duration && (time_duration <= *max_time_duration))
+                return 0;
+
+        bpf_map_update_elem(&max_latency_info, &key_idx, &bio_ptr, BPF_ANY);
+        bpf_map_update_elem(&max_latency_info, &val_idx, &time_duration, BPF_ANY);
```

```diff
diff stage03_*/vault19_bio_trace_user.c stage04_max_latency/vault19_bio_trace_user.c
--- stage03_*/vault19_bio_trace_user.c
+++ stage04_*/vault19_bio_trace_user.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
+struct called_info {
+        __u64 start;
+        __u64 end;
+};

+static void int_exit(int sig)
+{
+        print_max_latency_info(map_fd[0], map_fd[1]);
+        exit(0);
+}

+static void print_max_latency_info(int called_info_map, int max_latency_info_map)
+{
+        struct called_info called_info = {};
+        __u32 key_idx = 0, val_idx = 1;
+        __u64 bio_ptr, max_time_duration;
+
+        bpf_map_lookup_elem(max_latency_info_map, &key_idx, &bio_ptr);
+        bpf_map_lookup_elem(max_latency_info_map, &val_idx, &max_time_duration);
...
+        printf("Max latency %llu ns\n", max_time_duration);
+}
```

```diff
diff stage03_*/vault19_bio_trace_user.c stage04_max_latency/vault19_bio_trace_user.c
--- stage03_*/vault19_bio_trace_user.c
+++ stage04_*/vault19_bio_trace_user.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
+struct called_info {
+        __u64 start;
+        __u64 end;
+};

+static void int_exit(int sig)
+{
+        print_max_latency_info(map_fd[0], map_fd[1]);
+        exit(0);
+}

+static void print_max_latency_info(int called_info_map, int max_latency_info_map)
+{
+        struct called_info called_info = {};
+        __u32 key_idx = 0, val_idx = 1;
+        __u64 bio_ptr, max_time_duration;
+
+        bpf_map_lookup_elem(max_latency_info_map, &key_idx, &bio_ptr);
+        bpf_map_lookup_elem(max_latency_info_map, &val_idx, &max_time_duration);
...
+        printf("Max latency %llu ns\n", max_time_duration);
+}
```


BPF library: libbpf
prog/map
load, attach, control
vault19_bio_trace
map_fd[0]
map_fd[1]
return fd;
bpf() SYSCALL
KERNEL SPACE
called_info_map
(Shared memory)
max_latency_
info_map
(Shared memory)
map allocation !!

```diff
diff stage03_*/vault19_bio_trace_user.c stage04_max_latency/vault19_bio_trace_user.c
--- stage03_*/vault19_bio_trace_user.c
+++ stage04_*/vault19_bio_trace_user.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
+struct called_info {
+        __u64 start;
+        __u64 end;
+};

+static void int_exit(int sig)
+{
+        print_max_latency_info(map_fd[0], map_fd[1]);
+        exit(0);
+}

+static void print_max_latency_info(int called_info_map, int max_latency_info_map)
+{
+        struct called_info called_info = {};
+        __u32 key_idx = 0, val_idx = 1;
+        __u64 bio_ptr, max_time_duration;
+
+        bpf_map_lookup_elem(max_latency_info_map, &key_idx, &bio_ptr);
+        bpf_map_lookup_elem(max_latency_info_map, &val_idx, &max_time_duration);
...
+        printf("Max latency %llu ns\n", max_time_duration);
+}
```

```diff
diff stage03_*/vault19_bio_trace_user.c stage04_max_latency/vault19_bio_trace_user.c
--- stage03_*/vault19_bio_trace_user.c
+++ stage04_*/vault19_bio_trace_user.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
+struct called_info {
+        __u64 start;
+        __u64 end;
+};

+static void int_exit(int sig)
+{
+        print_max_latency_info(map_fd[0], map_fd[1]);
+        exit(0);
+}

+static void print_max_latency_info(int called_info_map, int max_latency_info_map)
+{
+        struct called_info called_info = {};
+        __u32 key_idx = 0, val_idx = 1;
+        __u64 bio_ptr, max_time_duration;
+
+        bpf_map_lookup_elem(max_latency_info_map, &key_idx, &bio_ptr);
+        bpf_map_lookup_elem(max_latency_info_map, &val_idx, &max_time_duration);
...
+        printf("Max latency %llu ns\n", max_time_duration);
+}
```

```diff
diff stage03_*/vault19_bio_trace_user.c stage04_max_latency/vault19_bio_trace_user.c
--- stage03_*/vault19_bio_trace_user.c
+++ stage04_*/vault19_bio_trace_user.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
+struct called_info {
+        __u64 start;
+        __u64 end;
+};

+static void int_exit(int sig)
+{
+        print_max_latency_info(map_fd[0], map_fd[1]);
+        exit(0);
+}

+static void print_max_latency_info(int called_info_map, int max_latenc
+{
+        struct called_info called_info = {};
+        __u32 key_idx = 0, val_idx = 1;
+        __u64 bio_ptr, max_time_duration;
+
+        bpf_map_lookup_elem(max_latency_info_map, &key_idx, &bio_ptr);
+        bpf_map_lookup_elem(max_latency_info_map, &val_idx, &max_time_duration);
...
+        printf("Max latency %llu ns\n", max_time_duration);
+}
```

```
=======================================
From: submit_bio(bio=0xffff9f8b3b27fb40)
To  : bio_endio (bio=0xffff9f8b3b27fb40)
Max latency 1148038 ns
=======================================
```

Stage 05: Getting sector of the max latency

submit_bio() -> bio_endio()

Getting sector of the max latency: How does it work?

```
1 blk_types.h
/*
 * main unit of I/O for the block layer and lower layers (ie drivers and
 * stacking drivers)
 */
struct bio {
        struct bio              *bi_next;       /* request queue link */
        struct gendisk          *bi_disk;
+-- 21 lines: unsigned int  bi_opf;  bottom bits req flags,----------

        struct bvec_iter        bi_iter;

        atomic_t                __bi_remaining;
        bio_end_io_t            *bi_end_io;

        void                    *bi_private;
#ifdef CONFIG_BLK_CGROUP
        /*
         * Represents the association of the css and request_queue for the bio.
         * If a bio goes direct to device, it will not have a blkg as it will
         * not have a request_queue associated with it.  The reference is put
 VISUAL  bpf   blk_types.h                                   37%   169:41
"~/git/linux/include/linux/blk_types.h" 451 lines --37%--
-- VISUAL --
```

```
1 bvec.h
/*
 * was unsigned short, but we might as well be ready for > 64kB I/O pages
 */
struct bio_vec {
        struct page     *bv_page;
        unsigned int    bv_len;
        unsigned int    bv_offset;
};

struct bvec_iter {
        sector_t                bi_sector;      /* device address in 512 byte
                                                   sectors */
        unsigned int            bi_size;        /* residual I/O count */

        unsigned int            bi_idx;         /* current index into
bvl_vec */

        unsigned int            bi_bvec_done;   /* number of bytes completed in
                                                   current bvec */
};
 VISUAL  bpf   bvec.h                                        27%   37:42
"~/git/linux/include/linux/bvec.h" 134 lines --26%--
-- VISUAL --
```

struct bio

include/linux/blk_types.h: 169L

bvec_iter -> bi_secor

block/bio.c: 2149L

```
# Stage 05: max latency sector
$
$ sudo ./vault19_bio_trace

...
     jbd2/sda2-8-1270  [000] ....5936311043684: 0: submit_bio(bio=0xffff9f8b3ac80000) called: 5936311040322
     jbd2/sda2-8-1270  [000] ....5936311049264: 0: submit_bio(bio=0xffff9f8b3ac80900) called: 5936311046015
     jbd2/sda2-8-1270  [000] ....5936311054851: 0: submit_bio(bio=0xffff9f8b3ac80480) called: 5936312677138

            sshd-1824  [000] ..s.3971354074711: 0: bio_endio (bio=0xffff9f8b3ac80000) called: 5936311717960
            sshd-1824  [000] ..s.3971354091868: 0: submit_bio() -> bio_endio() time duration: 920088 ns

            sshd-1824  [000] ..s.3971354095221: 0: bio_endio (bio=0xffff9f8b3ac80840) called: 5936311787989
            sshd-1824  [000] ..s.3971354095854: 0: submit_bio() -> bio_endio() time duration: 137821 ns

            sshd-1824  [000] ..s.5936312029728: 0: bio_endio (bio=0xffff9f8b3ac80480) called: 5936315594656
            sshd-1824  [000] ..s.5936312031498: 0: submit_bio() -> bio_endio() time duration: 2917518 ns

     jbd2/sda2-8-1270  [000] ....5936312684251: 0: submit_bio(bio=0xffff9f8b3ac80000) called: 5936312677138
            sshd-1824  [000] d.s15936315631455: 0: bio_endio (bio=0xffff9f8b3ac80000) called: 5936315594656
            sshd-1824  [000] d.s15936315660844: 0: submit_bio() -> bio_endio() time duration: 965205 ns

^C
=====================================================
From: submit_bio(bio=0xffff9f8b3ac80480) 5936312677138
To  : bio_endio (bio=0xffff9f8b3ac80480) 5936315594656
Bio Info : Sector (8675104)
Max latency 2917518 ns
=====================================================
```

Just run it !!
(copy & make & run)

Stage 05: Getting sector of the max latency
submit_bio() -> bio_endio()

```
$ cd ~/git/vault19_bpf_tutorial/stage05_max_latency_sector
$ ls
vault19_bio_trace_kern.c  vault19_bio_trace_user.c

$ cp ./vault19_bio_trace_kern.c ~/git/linux/samples/bpf/
$ cp ./vault19_bio_trace_user.c ~/git/linux/samples/bpf/

$ cd ~/git/linux/samples/bpf/

$ make
```

BPF hands-on tutorials:

Check the src files !!
(How does it work ?)

Stage 05: Getting sector of the max latency
submit_bio() -> bio_endio()

```
$ cd ~/git/vault19_bpf_tutorial/stage05_max_latency_sector
$ ls
vault19_bio_trace_kern.c  vault19_bio_trace_user.c

$ diff stage04_max_latency/vault19_bio_trace_kern.c  \
  ./vault19_bio_trace_kern.c

$ diff stage04_max_latency/vault19_bio_trace_user.c  \
  ./vault19_bio_trace_user.c
```

```diff
diff stage04_*/vault19_bio_trace_kern.c stage05_max_latency_sector/vault19_bio_trace_kern.c
--- stage04_*/vault19_bio_trace_kern.c
+++ stage05_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>


 SEC("kprobe/bio_endio")
 int bio_endio_entry(struct pt_regs *ctx)
         long bio_ptr = PT_REGS_PARM1(ctx);
         called_info = bpf_map_lookup_elem(&called_info_map, &bio_ptr);
         time_duration = called_info->end - called_info->start;

...
+        struct bio *bio;

+        bio = (struct bio *) bio_ptr;
+        bi_iter = _(bio->bi_iter);

+        curr.bio_ptr = bio_ptr;
+        curr.time_duration = time_duration;
+        curr.bi_sector = bi_iter.bi_sector;

         return 0;
 }
```

```diff
diff stage04_*/vault19_bio_trace_kern.c stage05_max_latency_sector/vault19_bio_trace_kern.c
--- stage04_*/vault19_bio_trace_kern.c
+++ stage05_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>


 SEC("kprobe/bio_endio")
 int bio_endio_entry(struct pt_regs *ctx)
         long bio_ptr = PT_REGS_PARM1(ctx);
         called_info = bpf_map_lookup_elem(&called_info_map, &bio_ptr);
         time_duration = called_info->end - called_info->start;

...
+        struct bio *bio;

+        bio = (struct bio *) bio_ptr;
+        bi_iter = _(bio->bi_iter);

+        curr.bio_ptr = bio_ptr;
+        curr.time_duration = time_duration;
+        curr.bi_sector = bi_iter.bi_sector;

         return 0;
 }
```

```diff
diff stage04_*/vault19_bio_trace_kern.c stage05_max_latency_sector/vault19_bio_trace_kern.c
--- stage04_*/vault19_bio_trace_kern.c
+++ stage05_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>


 SEC("kprobe/bio_endio")
 int bio_endio_entry(struct pt_regs *ctx)
         long bio_ptr = PT_REGS_PARM1(ctx);
         called_info = bpf_map_lookup_elem(&called_info_ma
         time_duration = called_info->end - called_info->s

...
+        struct bio *bio;

+        bio = (struct bio *) bio_ptr;
+        bi_iter = _(bio->bi_iter);

+        curr.bio_ptr = bio_ptr;
+        curr.time_duration = time_duration;
+        curr.bi_sector = bi_iter.bi_sector;

         return 0;
 }
```

```
1 blk_types.h                                                          X
/*
 * main unit of I/O for the block layer and lower layers (ie drivers and
 * stacking drivers)
 */
struct bio {
        struct bio              *bi_next;       /* request queue link */
        struct gendisk          *bi_disk;
++-- 21 lines: unsigned int  bi_opf;  bottom bits req flags,-------------------

        struct bvec_iter        bi_iter;

        atomic_t                __bi_remaining;
        bio_end_io_t            *bi_end_io;

        void                    *bi_private;
#ifdef CONFIG_BLK_CGROUP
        /*
         * Represents the association of the css and request_queue for the bio.
         * If a bio goes direct to device, it will not have a blkg as it will
         * not have a request_queue associated with it.  The reference is put
 VISUAL  bpf   blk_types.h                                   37%   169:41
"~/git/linux/include/linux/blk_types.h" 451 lines --37%--
-- VISUAL --
```

```diff
diff stage04_*/vault19_bio_trace_kern.c stage05_max_latency_sector/vault19_bio_trace_kern.c
--- stage04_*/vault19_bio_trace_kern.c
+++ stage05_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>


 SEC("kprobe/bio_endio")
 int bio_endio_entry(struct pt_regs *ctx)
         long bio_ptr = PT_REGS_PARM1(ctx);
         called_info = bpf_map_lookup_elem(&called_info_ma
         time_duration = called_info->end - called_info->s

...
+        struct bio *bio;

+        bio = (struct bio *) bio_ptr;
+        bi_iter = _(bio->bi_iter);

+        curr.bio_ptr = bio_ptr;
+        curr.time_duration = time_duration;
+        curr.bi_sector = bi_iter.bi_sector;

         return 0;
 }
```

```
1 bvec.h                                                                X
/*
 * was unsigned short, but we might as well be ready for > 64kB I/O pages
 */
struct bio_vec {
        struct page     *bv_page;
        unsigned int    bv_len;
        unsigned int    bv_offset;
};

struct bvec_iter {
        sector_t                bi_sector;      /* device address in 512 byte
                                                   sectors */
        unsigned int            bi_size;        /* residual I/O count */

        unsigned int            bi_idx;         /* current index into
bvec_vec */

        unsigned int            bi_bvec_done;   /* number of bytes completed in
                                                   current bvec */
};
VISUAL   bpf   bvec.h                                   27%    37:42
"~/git/linux/include/linux/bvec.h" 134 lines --26%--
-- VISUAL --
```

```diff
diff stage04_*/vault19_bio_trace_kern.c stage05_max_latency_sector/vault19_bio_trace_kern.c
--- stage04_*/vault19_bio_trace_kern.c
+++ stage05_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>


 SEC("kprobe/bio_endio")
 int bio_endio_entry(struct pt_regs *ctx)
         long bio_ptr = PT_REGS_PARM1(ctx);
         called_info = bpf_map_lookup_elem(&called_info_map, &bio_ptr);
         time_duration = called_info->end - called_info->start;

...
+        struct bio *bio;

+        bio = (struct bio *) bio_ptr;
+        bi_iter = _(bio->bi_iter);

+        curr.bio_ptr = bio_ptr;
+        curr.time_duration = time_duration;
+        curr.bi_sector = bi_iter.bi_sector;

         return 0;
 }
```

```
=====================================================
From: submit_bio(bio=0xffff9f8b3ac80480) 5936312677138
To  : bio_endio (bio=0xffff9f8b3ac80480) 5936315594656
Bio Info : Sector (8675104)
Max latency 2917518 ns
=====================================================
```

Stage 06: Getting stacktrace of the max latency

submit_bio() -> bio_endio()

```
# Stage 06: max latency stacktrace
$
$ sudo ./vault19_bio_trace

...
     jbd2/sda2-8-1270   [000] ....6661305624701: 0: submit_bio(bio=0xffff9f8af768f480) called: 6661305606354

          <idle>-0      [000] d.s.6661306555091: 0: bio_endio (bio=0xffff9f8af768f480) called: 6661306552485
          <idle>-0      [000] d.s.6661306566848: 0: submit_bio() -> bio_endio() time duration: 946131 ns
^C
=====================================================
From: submit_bio(bio=0xffff9f8af768f480) 6661305606354
To  : bio_endio (bio=0xffff9f8af768f480) 6661306552485
Bio Info : Sector (8682768)
Max latency 946131 ns
=====================================================
=> entry_SYSCALL_64_after_hwframe()
=> do_syscall_64()
=> __x64_sys_sync()
=> ksys_sync()
=> iterate_bdevs()
=> __filemap_fdatawrite_range()
=> do_writepages()
=> generic_writepages()
=> write_cache_pages()
=> __writepage()
=> __block_write_full_page()
=> submit_bh_wbc()
=> submit_bio()
```

BPF hands-on tutorials:

Just run it !!
(copy & make & run)

Stage 06: Getting stacktrace of the max latency
submit_bio() -> bio_endio()

```
$ cd ~/git/vault19_bpf_tutorial/stage06_max_latency_stacktrace
$ ls
vault19_bio_trace_kern.c  vault19_bio_trace_user.c

$ cp ./vault19_bio_trace_kern.c ~/git/linux/samples/bpf/
$ cp ./vault19_bio_trace_user.c ~/git/linux/samples/bpf/

$ cd ~/git/linux/samples/bpf/

$ make
$ sudo ./vault19_bio_trace
```

Check the src files !!
(How does it work ?)

Stage 06: Getting stacktrace of the max latency
submit_bio() -> bio_endio()

```
$ cd ~/git/vault19_bpf_tutorial/stage06_max_latency_stacktrace
$ ls
vault19_bio_trace_kern.c  vault19_bio_trace_user.c

$ diff stage05_max_latency_sector/vault19_bio_trace_kern.c  \
  ./vault19_bio_trace_kern.c

$ diff stage05_max_latency_sector/vault19_bio_trace_user.c  \
  ./vault19_bio_trace_user.c
```

```diff
diff stage05_*/vault19_bio_trace_kern.c stage06_max_latency_stacktrace/vault19_bio_trace_kern.c
--- stage05_*/vault19_bio_trace_kern.c
+++ stage06_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
  struct called_info {
+        u64 stack_id;
 };

+struct bpf_map_def SEC("maps") stacktrace_map = {
+        .type = BPF_MAP_TYPE_STACK_TRACE,
+        .key_size = sizeof(__u32),
+        .value_size = sizeof(__u64) * PERF_MAX_STACK_DEPTH,
+        .max_entries = 1024,
+};

 SEC("kprobe/submit_bio")
 int submit_bio_entry(struct pt_regs *ctx)
         struct called_info called_info = {
                 .start = start_time,
                 .end = 0,
+                .stack_id = 0
         };

+        stack_id = bpf_get_stackid(ctx, &stacktrace_map, 0);
+        if (stack_id)
+                called_info.stack_id = stack_id;
```

```diff
diff stage05_*/vault19_bio_trace_kern.c stage06_max_latency_stacktrace/vault19_bio_trace_kern.c
--- stage05_*/vault19_bio_trace_kern.c
+++ stage06_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
  struct called_info {
+        u64 stack_id;
 };

+struct bpf_map_def SEC("maps") stacktrace_map = {
+        .type = BPF_MAP_TYPE_STACK_TRACE,
+        .key_size = sizeof(__u32),
+        .value_size = sizeof(__u64) * PERF_MAX_STACK_DEPTH,
+        .max_entries = 1024,
+};

 SEC("kprobe/submit_bio")
 int submit_bio_entry(struct pt_regs *ctx)
         struct called_info called_info = {
                 .start = start_time,
                 .end = 0,
+                .stack_id = 0
         };

+        stack_id = bpf_get_stackid(ctx, &stacktrace_map, 0);
+        if (stack_id)
+                called_info.stack_id = stack_id;
```

```diff
diff stage05_*/vault19_bio_trace_kern.c stage06_max_latency_stacktrace/vault19_bio_trace_kern.c
--- stage05_*/vault19_bio_trace_kern.c
+++ stage06_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
  struct called_info {
+        u64 stack_id;
 };

+struct bpf_map_def SEC("maps") stacktrace_map = {
+        .type = BPF_MAP_TYPE_STACK_TRACE,
+        .key_size = sizeof(__u32),
+        .value_size = sizeof(__u64) * PERF_MAX_STACK_DEPTH,
+        .max_entries = 1024,
+};

 SEC("kprobe/submit_bio")
 int submit_bio_entry(struct pt_regs *ctx)
         struct called_info called_info = {
                 .start = start_time,
                 .end = 0,
+                .stack_id = 0
         };

+        stack_id = bpf_get_stackid(ctx, &stacktrace_map, 0);
+        if (stack_id)
+                called_info.stack_id = stack_id;
```

```diff
diff stage05_*/vault19_bio_trace_kern.c stage06_max_latency_stacktrace/vault19_bio_trace_kern.c
--- stage05_*/vault19_bio_trace_kern.c
+++ stage06_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
  struct called_info {
+        u64 stack_id;
 };

+struct bpf_map_def SEC("maps") stacktrace_map = {
+        .type = BPF_MAP_TYPE_STACK_TRACE,
+        .key_size = sizeof(__u32),
+        .value_size = sizeof(__u64) * PERF_MAX_STACK_DEPTH,
+        .max_entries = 1024,
+};

 SEC("kprobe/submit_bio")
 int submit_bio_entry(struct pt_regs *ctx)
         struct called_info called_info = {
                 .start = start_time,
                 .end = 0,
+                .stack_id = 0
         };

+        stack_id = bpf_get_stackid(ctx, &stacktrace_map, 0);
+        if (stack_id)
+                called_info.stack_id = stack_id;
```

```diff
diff stage05_*/vault19_bio_trace_kern.c stage06_max_latency_stacktrace/vault19_bio_trace_kern.c
--- stage05_*/vault19_bio_trace_kern.c
+++ stage06_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
  struct called_info {
+        u64 stack_id;
 };

+struct bpf_map_def SEC("maps") stacktrace_map = {
+        .type = BPF_MAP_TYPE_STACK_TRACE,
+        .key_size = sizeof(__u32),
+        .value_size = sizeof(__u64) * PERF_MAX_STACK_DEPTH,
+        .max_entries = 1024,
+};

 SEC("kprobe/submit_bio")
 int submit_bio_entry(struct pt_regs *ctx)
         struct called_info called_info = {
                 .start = start_time,
                 .end = 0,
+                .stack_id = 0
         };

+        stack_id = bpf_get_stackid(ctx, &stacktrace_map, 0);
+        if (stack_id)
+                called_info.stack_id = stack_id;
```

get_perf_callchain();

Helper function:
bpf_get_stackid() fill
the stacktrace in stacktrace_map

```diff
diff stage05_*/vault19_bio_trace_user.c stage06_max_latency_stacktrace/vault19_bio_trace_user.c
--- stage05_*/vault19_bio_trace_user.c
+++ stage06_*/vault19_bio_trace_user.c
 int main(int argc, char **argv) {
         if (load_bpf_file(filename)) {
                 printf("%s", bpf_log_buf);
                 return 1;
         }

+        if (load_kallsyms()) {
+                printf("failed to process /proc/kallsyms\n");
+                return 2;
+        }

 static void print_max_latency_info(int called_info_map, int max_latency_info_max, int stacktrace_map)
         struct called_info called_info = {};
         struct max_latency_bio_info max_info;
+        __u64 ip[PERF_MAX_STACK_DEPTH] = {};

         bpf_map_lookup_elem(max_latency_info_map, &one_idx, &max_info);
         bpf_map_lookup_elem(called_info_map, &max_info.bio_ptr, &called_info);
...

+        if (bpf_map_lookup_elem(stacktrace_map, &called_info.stack_id, ip) != 0) {
```

```diff
diff stage05_*/vault19_bio_trace_user.c stage06_max_latency_stacktrace/vault19_bio_trace_user.c
--- stage05_*/vault19_bio_trace_user.c
+++ stage06_*/vault19_bio_trace_user.c
 int main(int argc, char **argv) {
         if (load_bpf_file(filename)) {
                 printf("%s", bpf_log_buf);
                 return 1;
         }

+        if (load_kallsyms()) {
+                printf("failed to process /proc/kallsyms\n");
+                return 2;
+        }

 static void print_max_latency_info(int called_info_map, int max_latency_info_max, int stacktrace_map)
         struct called_info called_info = {};
         struct max_latency_bio_info max_info;
+        __u64 ip[PERF_MAX_STACK_DEPTH] = {};

         bpf_map_lookup_elem(max_latency_info_map, &one_idx, &max_info);
         bpf_map_lookup_elem(called_info_map, &max_info.bio_ptr, &called_info);
...

+        if (bpf_map_lookup_elem(stacktrace_map, &called_info.stack_id, ip) != 0) {
```

```diff
diff stage05_*/vault19_bio_trace_user.c stage06_max_latency_stacktrace/vault19_bio_trace_user.c
--- stage05_*/vault19_bio_trace_user.c
+++ stage06_*/vault19_bio_trace_user.c
 int main(int argc, char **argv) {
         if (load_bpf_file(filename)) {
                 printf("%s", bpf_log_buf);
                 return 1;
         }

+        if (load_kallsyms()) {
+                printf("failed to process /proc/kallsyms\n");
+                return 2;
+        }

 static void print_max_latency_info(int called_info_map, int max_latency_info_max, int stacktrace_map)
         struct called_info called_info = {};
         struct max_latency_bio_info max_info;
+        __u64 ip[PERF_MAX_STACK_DEPTH] = {};

         bpf_map_lookup_elem(max_latency_info_map, &one_idx, &max_info);
         bpf_map_lookup_elem(called_info_map, &max_info.bio_ptr, &called_info);
...

+        if (bpf_map_lookup_elem(stacktrace_map, &called_info.stack_id, ip) != 0) {
```

```diff
diff stage05_*/vault19_bio_trace_user.c stage06_max_latency_stacktrace/vault19_bio_trace_user.c
--- stage05_*/vault19_bio_trace_user.c
+++ stage06_*/vault19_bio_trace_user.c
 static void print_max_latency_info(int called_info_map, int max_latency_info_max, int stacktrace_map)
+        __u64 ip[PERF_MAX_STACK_DEPTH] = {};

         bpf_map_lookup_elem(max_latency_info_map, &one_idx, &max_info);
         bpf_map_lookup_elem(called_info_map, &max_info.bio_ptr, &called_info);
...
+        if (bpf_map_lookup_elem(stacktrace_map, &called_info.stack_id, ip) != 0) {
+                printf("Stack info not found !!\n");
+        } else {
+                for (i = PERF_MAX_STACK_DEPTH - 1; i >= 0; i--)
+                        print_ksym(ip[i]);
+        }

+static void print_ksym(__u64 addr) {
+        struct ksym *sym;
+
+        if (!addr)
+                return;
+
+        sym = ksym_search(addr);
+        printf("=> %s()\n", sym->name);
+}
```

```diff
diff stage05_*/vault19_bio_trace_user.c stage06_max_latency_stacktrace/vault19_bio_trace_user.c
--- stage05_*/vault19_bio_trace_user.c
+++ stage06_*/vault19_bio_trace_user.c
 static void print_max_latency_info(int called_info_map, int max_latency_info_max, int stacktrace_map)
+        __u64 ip[PERF_MAX_STACK_DEPTH] = {};

         bpf_map_lookup_elem(max_latency_info_map, &one_idx, &max_info);
         bpf_map_lookup_elem(called_info_map, &max_info.bio_ptr, &called_info);
...
+        if (bpf_map_lookup_elem(stacktrace_map, &called_info.stack_id, ip) != 0) {
+                printf("Stack info not found !!\n");
+        } else {
+                for (i = PERF_MAX_STACK_DEPTH - 1; i >= 0; i--)
+                        print_ksym(ip[i]);
+        }

+static void print_ksym(__u64 addr) {
+        struct ksym *sym;
+
+        if (!addr)
+                return;
+
+        sym = ksym_search(addr);
+        printf("=> %s()\n", sym->name);
+}
```

```diff
diff stage05_*/vault19_bio_trace_user.c stage06_max_latency_stacktrace/vault19_bio_trace_user.c
--- stage05_*/vault19_bio_trace_user.c
+++ stage06_*/vault19_bio_trace_user.c
 static void print_max_latency_info(int called_info_map, int max_latency_info_max, int stacktrace_map)
+        __u64 ip[PERF_MAX_STACK_DEPTH] = {};

         bpf_map_lookup_elem(max_latency_info_map, &one_idx, &max_info);
         bpf_map_lookup_elem(called_info_map, &max_info.bio_ptr, &called_info);
...
+        if (bpf_map_lookup_elem(stacktrace_map, &called_info.stack_id, ip) != 0) {
+                printf("Stack info not found !!\n");
+        } else {
+                for (i = PERF_MAX_STACK_DEPTH - 1; i >= 0; i--)
+                        print_ksym(ip[i]);
+        }

+static void print_ksym(__u64 addr) {
+        struct ksym *sym;
+
+        if (!addr)
+                return;
+
+        sym = ksym_search(addr);
+        printf("=> %s()\n", sym->name);
+}
```

```diff
diff stage05_*/vault19_bio_trace_user.c stage06_max_latency_stacktrace/vault19_bio_trace_user.c
--- stage05_*/vault19_bio_trace_user.c
+++ stage06_*/vault19_bio_trace_user.c
 static void print_max_latency_info(int called_info_map, int max_latency_info_max, int stacktrace_map)
+        __u64 ip[PERF_MAX_STACK_DEPTH] = {};

         bpf_map_lookup_elem(max_latency_info_map, &one_idx, &max_info);
         bpf_map_lookup_elem(called_info_map, &max_info.bio_ptr, &called_info);
...
+        if (bpf_map_lookup_elem(stacktrace_map, &called_info.stack_id, ip) != 0) {
+                printf("Stack info not found !!\n");
+        } else {
+                for (i = PERF_MAX_STACK_DEPTH - 1; i >= 0; i--)
+                        print_ksym(ip[i]);
+        }

+static void print_ksym(__u64 addr) {
+        struct ksym *sym;
+
+        if (!addr)
+                return;
+
+        sym = ksym_search(addr);
+        printf("=> %s()\n", sym->name);
+}
```

```
=========================================
=> entry_SYSCALL_64_after_hwframe()
=> do_syscall_64()
=> __x64_sys_sync()
=> ksys_sync()
=> iterate_bdevs()
=> __filemap_fdatawrite_range()
=> do_writepages()
=> generic_writepages()
=> write_cache_pages()
=> __writepage()
=> __block_write_full_page()
=> submit_bh_wbc()
=> submit_bio()
```

Stage 07: Calculate the request latency between blk_mq_start_request() -> blk_account_io_completion()

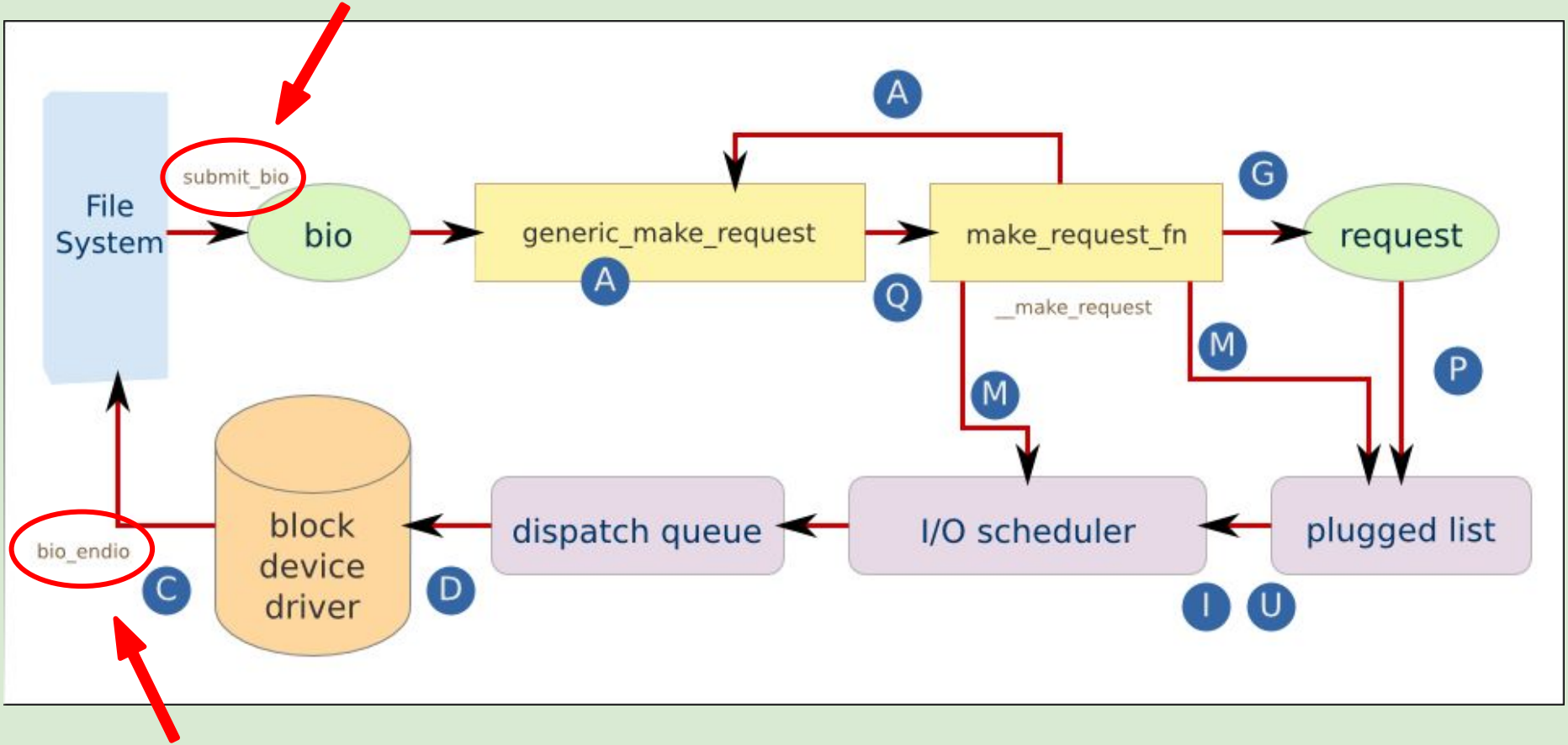


image from http://studyfoss.egloos.com/5585801

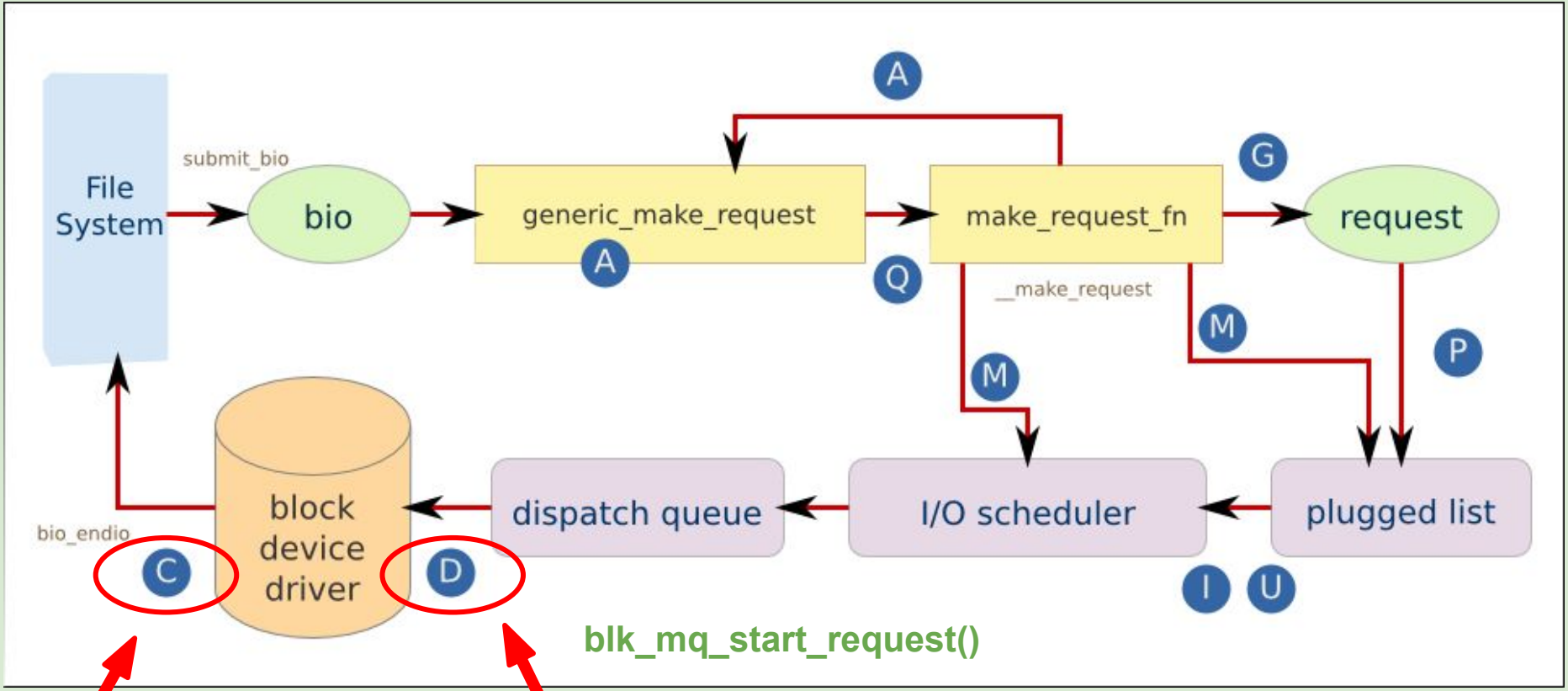


image from http://studyfoss.egloos.com/5585801

blk_account_io_completion()

Calculate the request latency: How does it work?

```
1 blk-mq.c
}
EXPORT_SYMBOL(blk_mq_complete_request);

int blk_mq_request_started(struct request *rq)
{
        return blk_mq_rq_state(rq) != MQ_RQ_IDLE;
}
EXPORT_SYMBOL_GPL(blk_mq_request_started);

void blk_mq_start_request(struct request *rq)
{
        struct request_queue *q = rq->q;

        blk_mq_sched_started_request(rq);

        trace_block_rq_issue(q, rq);

        if (test_bit(QUEUE_FLAG_STATS, &q->queue_flags)) {
                rq->io_start_time_ns = ktime_get_ns();
#ifdef CONFIG_BLK_DEV_THROTTLING_LOW
NORMAL  bpf   blk-mq.c                                  19%   666:1
"~/git/linux/block/blk-mq.c" 3482L, 85908C
```

```
1 blk-core.c
                bytes += bio->bi_iter.bi_size;
        }

        /* this could lead to infinite loop */
        BUG_ON(blk_rq_bytes(rq) && !bytes);
        return bytes;
}
EXPORT_SYMBOL_GPL(blk_rq_err_bytes);

void blk_account_io_completion(struct request *req, unsigned int bytes)
{
        if (blk_do_io_stat(req)) {
                const int sgrp = op_stat_group(req_op(req));
                struct hd_struct *part;

                part_stat_lock();
                part = req->part;
                part_stat_add(part, sectors[sgrp], bytes >> 9);
                part_stat_unlock();
        }
NORMAL  bpf   blk-core.c                                71%   1311:1
"~/git/linux/block/blk-core.c" 1822L, 49756C
```

blk_mq_start_request

(struct request* rq)

block/blk-mq.c: 3482L

blk_account_io_completion

(struct request* rq)

block/blk-core.c: 1822L

```
# Stage 07: latency other section
$
$ sudo ./vault19_bio_trace

...

     kworker/0:2-975   [000] ....6981879095488: 0: blk_mq_start_request(rq=0xffff9f8af7054000) is called!
            sshd-1824  [000] ..s.6981879829511: 0: blk_account_io_completion(rq=0xffff9f8af7054000) is called!
            sshd-1824  [000] ..s.6981879882239: 0: blk_mq_start_request() -> blk_account_io_completion() time
duration: 737423 ns

     kworker/0:2-975   [000] ....6981942334663: 0: blk_mq_start_request(rq=0xffff9f8b3aef12c0) is called!
 vault19_bio_tra-19703 [000] ..s.6981942888506: 0: blk_account_io_completion(rq=0xffff9f8b3aef12c0) is called!
 vault19_bio_tra-19703 [000] ..s.6981942932035: 0: blk_mq_start_request() -> blk_account_io_completion() time
duration: 559523 ns


     kworker/0:2-975   [000] ....6983927192671: 0: blk_mq_start_request(rq=0xffff9f8af70552c0) is called!
            sshd-1824  [000] ..s.6983927893060: 0: blk_account_io_completion(rq=0xffff9f8af70552c0) is called!
            sshd-1824  [000] ..s.6983927947056: 0: blk_mq_start_request() -> blk_account_io_completion() time
duration: 703701 ns

     kworker/0:2-975   [000] ....6983990342270: 0: blk_mq_start_request(rq=0xffff9f8b3aef0000) is called!
 vault19_bio_tra-19703 [000] ..s.6983991154089: 0: blk_account_io_completion(rq=0xffff9f8b3aef0000) is called!
 vault19_bio_tra-19703 [000] ..s.6983991205574: 0: blk_mq_start_request() -> blk_account_io_completion() time
duration: 815129 ns
```

BPF hands-on tutorials:

Just run it !!
(copy & make & run)

Stage 07: Calculate the request latency between blk_mq_start_request() -> blk_account_io_completion()

```
$ cd ~/git/vault19_bpf_tutorial/stage07_latency_other_section
$ ls
vault19_bio_trace_kern.c  vault19_bio_trace_user.c

$ cp ./vault19_bio_trace_kern.c ~/git/linux/samples/bpf/
$ cp ./vault19_bio_trace_user.c ~/git/linux/samples/bpf/

$ cd ~/git/linux/samples/bpf/

$ make
$ sudo ./vault19_bio_trace
```

BPF hands-on tutorials:

Check the src files !!
(How does it work ?)

Stage 07: Calculate the request latency between blk_mq_start_request() -> blk_account_io_completion()

```
$ cd ~/git/vault19_bpf_tutorial/stage07_latency_other_section
$ ls
vault19_bio_trace_kern.c  vault19_bio_trace_user.c

$ diff stage03_bio_latency_map/vault19_bio_trace_kern.c  \
  ./vault19_bio_trace_kern.c

$ diff stage03_bio_latency_map/vault19_bio_trace_user.c  \
  ./vault19_bio_trace_user.c
```

```diff
diff stage03_*/vault19_bio_trace_kern.c stage07_latency_other_section/vault19_bio_trace_kern.c
--- stage03_*/vault19_bio_trace_kern.c
+++ stage07_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>

-SEC("kprobe/submit_bio")
+SEC("kprobe/blk_mq_start_request")
 int submit_bio_entry(struct pt_regs *ctx)
 {
         u64 start_time = bpf_ktime_get_ns();
-        long bio_ptr = PT_REGS_PARM1(ctx);
+        long rq_ptr = PT_REGS_PARM1(ctx);
         struct called_info called_info = {
                 .start = start_time,
                 .end = 0

-SEC("kprobe/bio_endio")
+SEC("kprobe/blk_account_io_completion")
 int bio_endio_entry(struct pt_regs *ctx)
 {
         u64 end_time = bpf_ktime_get_ns();
-        long bio_ptr = PT_REGS_PARM1(ctx);
+        long rq_ptr = PT_REGS_PARM1(ctx);
         struct called_info *called_info;
         u64 time_duration;
```

```diff
diff stage03_*/vault19_bio_trace_kern.c stage07_latency_other_section/vault19_bio_trace_kern.c
--- stage03_*/vault19_bio_trace_kern.c
+++ stage07_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>

-SEC("kprobe/submit_bio")
+SEC("kprobe/blk_mq_start_request")
 int submit_bio_entry(struct pt_regs *ctx)
 {
         u64 start_time = bpf_ktime_get_ns();
-        long bio_ptr = PT_REGS_PARM1(ctx);
+        long rq_ptr = PT_REGS_PARM1(ctx);
         struct called_info called_info = {
                 .start = start_time,
                 .end = 0

-SEC("kprobe/bio_endio")
+SEC("kprobe/blk_account_io_completion")
 int bio_endio_entry(struct pt_regs *ctx)
 {
         u64 end_time = bpf_ktime_get_ns();
-        long bio_ptr = PT_REGS_PARM1(ctx);
+        long rq_ptr = PT_REGS_PARM1(ctx);
         struct called_info *called_info;
         u64 time_duration;
```

```diff
diff stage03_*/vault19_bio_trace_kern.c stage07_latency_other_section/vault19_bio_trace_kern.c
--- stage03_*/vault19_bio_trace_kern.c
+++ stage07_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>

-SEC("kprobe/submit_bio")
+SEC("kprobe/blk_mq_start_request")
 int submit_bio_entry(struct pt_regs *ctx)
 {
         u64 start_time = bpf_ktime_get_ns();
-        long bio_ptr = PT_REGS_PARM1(ctx);
+        long rq_ptr = PT_REGS_PARM1(ctx);
         struct called_info called_info = {
                 .start = start_time,
                 .end = 0

-SEC("kprobe/bio_endio")
+SEC("kprobe/blk_account_io_completion")
 int bio_endio_entry(struct pt_regs *ctx)
 {
         u64 end_time = bpf_ktime_get_ns();
-        long bio_ptr = PT_REGS_PARM1(ctx);
+        long rq_ptr = PT_REGS_PARM1(ctx);
         struct called_info *called_info;
         u64 time_duration;
```

```
1 blk-mq.c                                                                X
}
EXPORT_SYMBOL(blk_mq_complete_request);

int blk_mq_request_started(struct request *rq)
{
        return blk_mq_rq_state(rq) != MQ_RQ_IDLE;
}
EXPORT_SYMBOL_GPL(blk_mq_request_started);

void blk_mq_start_request(struct request *rq)
{
        struct request_queue *q = rq->q;

        blk_mq_sched_started_request(rq);

        trace_block_rq_issue(q, rq);

        if (test_bit(QUEUE_FLAG_STATS, &q->queue_flags)) {
                rq->io_start_time_ns = ktime_get_ns();
#ifdef CONFIG_BLK_DEV_THROTTLING_LOW
NORMAL  bpf   blk-mq.c                                   19%   666:1
"~/git/linux/block/blk-mq.c" 3482L, 85908C
```

```diff
diff stage03_*/vault19_bio_trace_kern.c stage07_latency_other_section/vault19_bio_trace_kern.c
--- stage03_*/vault19_bio_trace_kern.c
+++ stage07_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>

-SEC("kprobe/submit_bio")
+SEC("kprobe/blk_mq_start_request")
 int submit_bio_entry(struct pt_regs *ctx)
 {
         u64 start_time = bpf_ktime_get_ns();
-        long bio_ptr = PT_REGS_PARM1(ctx);
+        long rq_ptr = PT_REGS_PARM1(ctx);
         struct called_info called_info = {
                 .start = start_time,
                 .end = 0

-SEC("kprobe/bio_endio")
+SEC("kprobe/blk_account_io_completion")
 int bio_endio_entry(struct pt_regs *ctx)
 {
         u64 end_time = bpf_ktime_get_ns();
-        long bio_ptr = PT_REGS_PARM1(ctx);
+        long rq_ptr = PT_REGS_PARM1(ctx);
         struct called_info *called_info;
         u64 time_duration;
```

```diff
diff stage03_*/vault19_bio_trace_kern.c stage07_latency_other_section/vault19_bio_trace_kern.c
--- stage03_*/vault19_bio_trace_kern.c
+++ stage07_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>

-SEC("kprobe/submit_bio")
+SEC("kprobe/blk_mq_start_request")
 int submit_bio_entry(struct pt_regs *ctx)
 {
         u64 start_time = bpf_ktime_get_ns();
-        long bio_ptr = PT_REGS_PARM1(ctx);
+        long rq_ptr = PT_REGS_PARM1(ctx);
         struct called_info called_info = {
                 .start = start_time,
                 .end = 0

-SEC("kprobe/bio_endio")
+SEC("kprobe/blk_account_io_completion")
 int bio_endio_entry(struct pt_regs *ctx)
 {
         u64 end_time = bpf_ktime_get_ns();
-        long bio_ptr = PT_REGS_PARM1(ctx);
+        long rq_ptr = PT_REGS_PARM1(ctx);
         struct called_info *called_info;
         u64 time_duration;
```

```
1 blk-core.c                                                                    X
                bytes += bio->bi_iter.bi_size;
        }

        /* this could lead to infinite loop */
        BUG_ON(blk_rq_bytes(rq) && !bytes);
        return bytes;
}
EXPORT_SYMBOL_GPL(blk_rq_err_bytes);

void blk_account_io_completion(struct request *req, unsigned int bytes)
{
        if (blk_do_io_stat(req)) {
                const int sgrp = op_stat_group(req_op(req));
                struct hd_struct *part;

                part_stat_lock();
                part = req->part;
                part_stat_add(part, sectors[sgrp], bytes >> 9);
                part_stat_unlock();
        }
NORMAL  bpf   blk-core.c                                        71%   1311:1
"~/git/linux/block/blk-core.c" 1822L, 49756C
```

```diff
diff stage03_*/vault19_bio_trace_kern.c stage07_latency_other_section/vault19_bio_trace_kern.c
--- stage03_*/vault19_bio_trace_kern.c
+++ stage07_*/vault19_bio_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>

-SEC("kprobe/submit_bio")
+SEC("kprobe/blk_mq_start_request")
 int submit_bio_entry(struct pt_regs *ctx)
 {
         u64 start_time = bpf_ktime_get_ns();
-        long bio_ptr = PT_REGS_PARM1(ctx);
+        long rq_ptr = PT_REGS_PARM1(ctx);
         struct called_info called_info = {
                 .start = start_time,
                 .end = 0

-SEC("kprobe/bio_endio")
+SEC("kprobe/blk_account_io_completion")
 int bio_endio_entry(struct pt_regs *ctx)
 {
         u64 end_time = bpf_ktime_get_ns();
-        long bio_ptr = PT_REGS_PARM1(ctx);
+        long rq_ptr = PT_REGS_PARM1(ctx);
         struct called_info *called_info;
         u64 time_duration;
```

```
blk_mq_start_request(rq=0xffff9f8af7054000) is called!
blk_account_io_completion(rq=0xffff9f8af7054000) is called!
blk_mq_start_request() -> blk_account_io_completion()
time duration: 737423 ns
```

Stage 08: Count the number of pagecache miss

Scenario ? read()

```
1 read.c
#include <stdio.h>
#include <stdlib.h>

//hello linux filesystem\n
#define SIZE 24

void main()
{
    FILE *fp = fopen("/home/kosslab/hello.txt","r");
    char buf[BUFSIZ];

    if (fp) {
        fread(buf, SIZE, 1, fp);
        printf("%s", buf);
        fclose(fp);
    }
}
~
~
~
NORMAL  read.c                                  5%   1:1
"read.c" 17L, 246C
```

Scenario ?

Stage 08-1: First read case

vs

Stage 08-2: Secondary read case

Scenario ?

Stage 08-1: First read case

vs

Stage 08-2: Secondary read case

using uftrace tool

Stage 08-1: First read case

```
# recorded call-graph by: "uftrace record -d read.uftrace.data -K 30 ./read"
$ cd ~/uftrace-data
$ cd read.uftrace.data
$ uftrace graph
  596.832 us :     |         +-(2)  __vfs_read
  595.923 us :     |         | (2) ext4_file_read_iter
  595.100 us :     |         | (2) generic_file_read_iter
    1.863 us :     |         |  +-(4) _cond_resched
    0.598 us :     |         |  | (4) rcu_all_qs
             :     |         |  |
    2.032 us :     |         |  +-(4) pagecache_get_page
    0.720 us :     |         |  | (4) find_get_entry
             :     |         |  |
  101.849 us :     |         |  +-(1) page_cache_sync_readahead
  101.525 us :     |         |  | (1) ondemand_readahead
  101.166 us :     |         |  | (1) __do_page_cache_readahead
    3.221 us :     |         |  |  +-(1) __page_cache_alloc
    2.882 us :     |         |  |  | (1) alloc_pages_current
...
    1.869 us :     |         |  |  |  +-(1) __alloc_pages_nodemask
    0.464 us :     |         |  |  |     +-(1) _cond_resched
    0.152 us :     |         |  |  |     | (1) rcu_all_qs
             :     |         |  |  |     |
    0.872 us :     |         |  |  |     +-(1) get_page_from_freelist
    0.309 us :     |         |  |  |       (2) __inc_numa_state
             :     |         |  |  |
   97.124 us :     |         |  |  +-(1) read_pages
    0.149 us :     |         |  |     +-(1) blk_start_plug
```

pagecache Miss !!

and allocate it

Stage 08-2: Secondary read case

```
# recorded call-graph by: "uftrace record -d read2.uftrace.data -K 30 ./read"
$ cd ~/uftrace-data
$ cd read2.uftrace.data
$ uftrace graph
   10.344 us :      |         +-(2) __vfs_read
    9.601 us :      |         | (2) ext4_file_read_iter
    8.828 us :      |         | (2) generic_file_read_iter
    1.797 us :      |         |  +-(4) _cond_resched
    0.590 us :      |         |  | (4) rcu_all_qs
             :      |         |  |
    1.642 us :      |         |  +-(3) pagecache_get_page
    0.685 us :      |         |  | (3) find_get_entry
             :      |         |  |
    0.158 us :      |         |  +-(1) mark_page_accessed
             :      |         |  |
    2.777 us :      |         |  +-(2) touch_atime
    2.170 us :      |         |    (2) atime_needs_update
    1.529 us :      |         |    (2) current_time
    0.301 us :      |         |     +-(2) ktime_get_coarse_real_ts64
             :      |         |     |
    0.301 us :      |         |     +-(2) timespec64_trunc
             :      |         |
    0.146 us :      |         +-(1) __fsnotify_parent
             :      |         |
    0.151 us :      |         +-(1) fsnotify
             :      |
  649.414 us :      +-(1) printf
```

So, pagecache Hit !!

no allocation

Count Pagecache miss: How does it work?

```
1 filemap.c +
 *
 * If there is a page cache page, it is returned with an increased refcount.
 */
struct page *pagecache_get_page(struct address_space *mapping, pgoff_t offset,
        int fgp_flags, gfp_t gfp_mask)
{
+--  3 lines: struct page *page;-----------------------------------------------
        page = find_get_entry(mapping, offset);
        if (!page)
                goto no_page;
+--- 32 lines: if (fgp_flags & FGP_LOCK) {-------------------------------------
                if (!page)
                        return NULL;

                if (WARN_ON_ONCE(!(fgp_flags & FGP_LOCK)))
                        fgp_flags |= FGP_LOCK;

                /* Init accessed so avoid atomic mark_page_accessed later */
                if (fgp_flags & FGP_ACCESSED)
                        __SetPageReferenced(page);
VISUAL  bpf   filemap.c   +                                   47%  1615:37
"~/git/linux/mm/filemap.c" [Modified] 3398 lines --47%--
-- VISUAL --
```

pagecache_get_page

(return page*)

mm/filemap.c: 1617L

```
# Stage 08: pagecache miss counts
$
$ sudo ./vault19_pagecache_trace

...

 systemd-journal-1358  [000] d...103910198012: 0: pagecache_get_page (retval=0xfffffa9641e20840)

     jbd2/sda2-8-1284  [000] d...107092903930: 0: pagecache_get_page (retval=0x0)
     jbd2/sda2-8-1284  [000] d...107092982708: 0: pagecache_get_page (retval=0x0)
     jbd2/sda2-8-1284  [000] d...107093001200: 0: pagecache_get_page (retval=0xfffffa9641c474c0)
     jbd2/sda2-8-1284  [000] d...107093011427: 0: pagecache_get_page (retval=0xfffffa9641c474c0)

     jbd2/sda2-8-1284  [000] d...107095024422: 0: pagecache_get_page (retval=0x0)
     jbd2/sda2-8-1284  [000] d...107095091620: 0: pagecache_get_page (retval=0x0)
     jbd2/sda2-8-1284  [000] d...107095104781: 0: pagecache_get_page (retval=0xfffffa9641aefa00)
     jbd2/sda2-8-1284  [000] d...107095112094: 0: pagecache_get_page (retval=0xfffffa9641aefa00)

     jbd2/sda2-8-1284  [000] d...107097172039: 0: pagecache_get_page (retval=0xfffffa9641f08500)
     jbd2/sda2-8-1284  [000] d...107097244841: 0: pagecache_get_page (retval=0xfffffa9641f08a40)
     jbd2/sda2-8-1284  [000] d...107097253557: 0: pagecache_get_page (retval=0xfffffa9641f08500)
     jbd2/sda2-8-1284  [000] d...107097258230: 0: pagecache_get_page (retval=0xfffffa9641f08a40)
     jbd2/sda2-8-1284  [000] d...107097265320: 0: pagecache_get_page (retval=0xfffffa9641f08500)
     jbd2/sda2-8-1284  [000] d...107097269918: 0: pagecache_get_page (retval=0xfffffa9641f08a40)

^C
=====================================================
[Total 15 Hit 11 miss 4]
=====================================================
```

BPF hands-on tutorials:

Just run it !!
(copy & make & run)

Stage 08: Count the number of pagecache miss

```
$ cd ~/git/vault19_bpf_tutorial/stage08_pagecache_miss_counts
$ ls
vault19_pagecache_trace_kern.c  vault19_pagecache_trace_user.c

$ cp ./vault19_pagecache_trace_kern.c ~/git/linux/samples/bpf/
$ cp ./vault19_pagecache_trace_user.c ~/git/linux/samples/bpf/

$ cd ~/git/linux/samples/bpf/

$ make
$ sudo ./vault19_pagecache_trace
```

Check the src files !!
(How does it work ?)

Stage 08: Count the number of pagecache miss

```
$ cd ~/git/vault19_bpf_tutorial/stage08_pagecache_miss_counts
$ ls
vault19_pagecache_trace_kern.c  vault19_pagecache_trace_user.c

$ less vault19_pagecache_trace_kern.c

$ less vault19_pagecache_trace_user.c
```

```diff
diff /dev/null stage08_pagecache_miss_counts/vault19_pagecache_trace_kern.c
--- /dev/null
+++ stage06_*/vault19_pagecache_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>

+struct bpf_map_def SEC("maps") pagecache_retval_map = {
+        .type = BPF_MAP_TYPE_HASH,
+        .key_size = sizeof(u64),
+        .value_size = sizeof(long),
+        .max_entries = 4096,
+};

+SEC("kretprobe/pagecache_get_page")
+int pagecache_get_page_retval(struct pt_regs *ctx)
+{
+        char fmt[] = "pagecache_get_page (retval=0x%lx)\n";
+        long pagecache_retval = PT_REGS_RC(ctx);
+        u64 start_time = bpf_ktime_get_ns();
+
+        bpf_trace_printk(fmt, sizeof(fmt), pagecache_retval);
+        bpf_map_update_elem(&pagecache_retval_map, &start_time, &pagecache_retval, BPF_ANY);
+        return 0;
+}
```

```diff
diff /dev/null stage08_pagecache_miss_counts/vault19_pagecache_trace_kern.c
--- /dev/null
+++ stage06_*/vault19_pagecache_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>

+struct bpf_map_def SEC("maps") pagecache_retval_map = {
+        .type = BPF_MAP_TYPE_HASH,
+        .key_size = sizeof(u64),
+        .value_size = sizeof(long),
+        .max_entries = 4096,
+};

+SEC("kretprobe/pagecache_get_page")
+int pagecache_get_page_retval(struct pt_regs *ctx)
+{
+        char fmt[] = "pagecache_get_page (retval=0x%lx)\n";
+        long pagecache_retval = PT_REGS_RC(ctx);
+        u64 start_time = bpf_ktime_get_ns();
+
+        bpf_trace_printk(fmt, sizeof(fmt), pagecache_retval);
+        bpf_map_update_elem(&pagecache_retval_map, &start_time, &pagecache_retval, BPF_ANY);
+        return 0;
+}
```

```diff
diff /dev/null stage08_pagecache_miss_counts/vault19_pagecache_trace_kern.c
--- /dev/null
+++ stage06_*/vault19_pagecache_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>

+struct bpf_map_def SEC("maps") pagecache_retval_map = {
+        .type = BPF_MAP_TYPE_HASH,
+        .key_size = sizeof(u64),
+        .value_size = sizeof(long),
+        .max_entries = 4096,
+};

+SEC("kretprobe/pagecache_get_page")
+int pagecache_get_page_retval(struct pt_regs *ctx)
+{
+        char fmt[] = "pagecache_get_page (retval=0x%lx)\n";
+        long pagecache_retval = PT_REGS_RC(ctx);
+        u64 start_time = bpf_ktime_get_ns();
+
+        bpf_trace_printk(fmt, sizeof(fmt), pagecache_retval);
+        bpf_map_update_elem(&pagecache_retval_map, &start_time, &pagecache_retval, BPF_ANY);
+        return 0;
+}
```

```diff
diff /dev/null stage08_pagecache_miss_counts/vault19_pagecache_trace_kern.c
--- /dev/null
+++ stage06_*/vault19_pagecache_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>

+struct bpf_map_def SEC("maps") pagecache_retval_map = {
+        .type = BPF_MAP_TYPE_HASH,
+        .key_size = sizeof(u64),
+        .value_size = sizeof(long),
+        .max_entries = 4096,
+};

+SEC("kretprobe/pagecache_get_page")
+int pagecache_get_page_retval(struct pt_regs *ctx)
+{
+        char fmt[] = "pagecache_get_page (retval=0x%lx)\n";
+        long pagecache_retval = PT_REGS_RC(ctx);
+        u64 start_time = bpf_ktime_get_ns();
+
+        bpf_trace_printk(fmt, sizeof(fmt), pagecache_retval);
+        bpf_map_update_elem(&pagecache_retval_map, &start_time, &pagecache_retval, BPF_ANY);
+        return 0;
+}
```

Attached to the return point of pagecache_get_page() function

```diff
diff /dev/null stage08_pagecache_miss_counts/vault19_pagecache_trace_kern.c
--- /dev/null
+++ stage06_*/vault19_pagecache_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>

+struct bpf_map_def SEC("maps") pagecache_retval_map = {
+        .type = BPF_MAP_TYPE_HASH,
+        .key_size = sizeof(u64),
+        .value_size = sizeof(long),
+        .max_entries = 4096,
+};

+SEC("kretprobe/pagecache_get_page")
+int pagecache_get_page_retval(struct pt_regs *ctx)
+{
+        char fmt[] = "pagecache_get_page (retval=0x%lx)\n";
+        long pagecache_retval = PT_REGS_RC(ctx);
+        u64 start_time = bpf_ktime_get_ns();
+
+        bpf_trace_printk(fmt, sizeof(fmt), pagecache_retval);
+        bpf_map_update_elem(&pagecache_retval_map, &start_time, &pagecache_retval, BPF_ANY);
+        return 0;
+}
```

```diff
diff /dev/null stage08_pagecache_miss_counts/vault19_pagecache_trace_kern.c
--- /dev/null
+++ stage06_*/vault19_pagecache_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>

+struct bpf_map_def SEC("maps") pagecache_retval_map = {
+        .type = BPF_MAP_TYPE_HASH,
+        .key_size = sizeof(u64),
+        .value_size = sizeof(long),
+        .max_entries = 4096,
+};

+SEC("kretprobe/pagecache_get_page")
+int pagecache_get_page_retval(struct pt_regs *ctx)
+{
+        char fmt[] = "pagecache_get_page (retval=0x%lx)\n
+        long pagecache_retval = PT_REGS_RC(ctx);
+        u64 start_time = bpf_ktime_get_ns();
+
+        bpf_trace_printk(fmt, sizeof(fmt), pagecache_retv
+        bpf_map_update_elem(&pagecache_retval_map, &start
+        return 0;
+}
```

```
1 filemap.c +                                                          X
 *
 * If there is a page cache page, it is returned with an increased refcount.
 */
struct page *pagecache_get_page(struct address_space *mapping, pgoff_t offset,
        int fgp_flags, gfp_t gfp_mask)
{
+--  3 lines: struct page *page;-----------------------------------------
        page = find_get_entry(mapping, offset);
        if (!page)
                goto no_page;
+--- 32 lines: if (fgp_flags & FGP_LOCK) {-------------------------------
                if (!page)
                        return NULL;

                if (WARN_ON_ONCE(!(fgp_flags & FGP_LOCK)))
                        fgp_flags |= FGP_LOCK;

                /* Init accessed so avoid atomic mark_page_accessed later */
                if (fgp_flags & FGP_ACCESSED)
                        __SetPageReferenced(page);
VISUAL  bpf   filemap.c  +                                   47%   1615:37
"~/git/linux/mm/filemap.c" [Modified] 3398 lines --47%--
-- VISUAL --
```

```diff
diff /dev/null stage08_pagecache_miss_counts/vault19_pagecache_trace_kern.c
--- /dev/null
+++ stage06_*/vault19_pagecache_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>

+struct bpf_map_def SEC("maps") pagecache_retval_map = {
+        .type = BPF_MAP_TYPE_HASH,
+        .key_size = sizeof(u64),
+        .value_size = sizeof(long),
+        .max_entries = 4096,
+};

+SEC("kretprobe/pagecache_get_page")
+int pagecache_get_page_retval(struct pt_regs *ctx)
+{
+        char fmt[] = "pagecache_get_page (retval=0x%lx)\n";
+        long pagecache_retval = PT_REGS_RC(ctx);
+        u64 start_time = bpf_ktime_get_ns();
+
+        bpf_trace_printk(fmt, sizeof(fmt), pagecache_retval);
+        bpf_map_update_elem(&pagecache_retval_map, &start_time, &pagecache_retval, BPF_ANY);
+        return 0;
+}
```

```diff
diff /dev/null stage08_pagecache_miss_counts/vault19_pagecache_trace_user.c
--- /dev/null
+++ stage06_*/vault19_pagecache_trace_user.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>

+static void int_exit(int sig) {
+        print_pagecache_retval_stats(map_fd[0]);
+        exit(0);
+}

+static void print_pagecache_retval_stats(int pagecache_retval_map) {
+        __u64 key = -1, next_key;
+        long pagecache_retval;
+        int hit = 0, miss = 0;
+
+        while (bpf_map_get_next_key(pagecache_retval_map, &key, &next_key) == 0) {
+                bpf_map_lookup_elem(pagecache_retval_map, &next_key, &pagecache_retval);
+
+                if (pagecache_retval)
+                        hit++;
+                else
+                        miss++;
+
+                key = next_key;
+        }
```

```diff
diff /dev/null stage08_pagecache_miss_counts/vault19_pagecache_trace_user.c
--- /dev/null
+++ stage06_*/vault19_pagecache_trace_user.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>

+static void int_exit(int sig) {
+        print_pagecache_retval_stats(map_fd[0]);
+        exit(0);
+}

+static void print_pagecache_retval_stats(int pagecache_retval_map) {
+        __u64 key = -1, next_key;
+        long pagecache_retval;
+        int hit = 0, miss = 0;
+
+        while (bpf_map_get_next_key(pagecache_retval_map, &key, &next_key) == 0) {
+                bpf_map_lookup_elem(pagecache_retval_map, &next_key, &pagecache_retval);
+
+                if (pagecache_retval)
+                        hit++;
+                else
+                        miss++;
+
+                key = next_key;
+        }
```

```diff
diff /dev/null stage08_pagecache_miss_counts/vault19_pagecache_trace_user.c
--- /dev/null
+++ stage06_*/vault19_pagecache_trace_user.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>

+static void int_exit(int sig) {
+        print_pagecache_retval_stats(map_fd[0]);
+        exit(0);
+}

+static void print_pagecache_retval_stats(int pagecache_retval_map) {
+        __u64 key = -1, next_key;
+        long pagecache_retval;
+        int hit = 0, miss = 0;
+
+        while (bpf_map_get_next_key(pagecache_retval_map, &key, &next_key) == 0) {
+                bpf_map_lookup_elem(pagecache_retval_map, &next_key, &pagecache_retval);
+
+                if (pagecache_retval)
+                        hit++;
+                else
+                        miss++;
+
+                key = next_key;
+        }
```

```diff
diff /dev/null stage08_pagecache_miss_counts/vault19_pagecache_trace_user.c
--- /dev/null
+++ stage06_*/vault19_pagecache_trace_user.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>

+static void int_exit(int sig) {
+        print_pagecache_retval_stats(map_fd[0]);
+        exit(0);
+}

+static void print_pagecache_retval_stats(int pagecache_retval_map) {
+        __u64 key = -1, next_key;
+        long pagecache_retval;
+        int hit = 0, miss = 0;
+
+        while (bpf_map_get_next_key(pagecache_retval_map, &key, &next_key) == 0) {
+                bpf_map_lookup_elem(pagecache_retval_map, &next_key, &pagecache_retval);
+
+                if (pagecache_retval)
+                        hit++;
+                else
+                        miss++;
+
+                key = next_key;
+        }
```

```
=====================================================
[Total 15 Hit 11 miss 4]
=====================================================
```

Stage 09: Count the number of pagecache miss with Filter PID

```
# Stage 09: pagecache miss pid filter
$
$ sudo ./vault19_pagecache_trace 1284

...

==================================
pid_filter_map update: (pid=1284)
==================================

     jbd2/sda2-8-1284  [000] d...363092899392: 0: pagecache_get_page (retval=0x0)
     jbd2/sda2-8-1284  [000] d...363092974631: 0: pagecache_get_page (retval=0x0)
     jbd2/sda2-8-1284  [000] d...363092993253: 0: pagecache_get_page (retval=0xfffffa96418e9640)
     jbd2/sda2-8-1284  [000] d...363093004199: 0: pagecache_get_page (retval=0xfffffa96418e9640)

     jbd2/sda2-8-1284  [000] d...363094053231: 0: pagecache_get_page (retval=0x0)
     jbd2/sda2-8-1284  [000] d...363094053327: 0: pagecache_get_page (retval=0x0)
     jbd2/sda2-8-1284  [000] d...363094053391: 0: pagecache_get_page (retval=0xfffffa964191bdc0)
     jbd2/sda2-8-1284  [000] d...363094053455: 0: pagecache_get_page (retval=0xfffffa964191bdc0)

     jbd2/sda2-8-1284  [000] d...363097106202: 0: pagecache_get_page (retval=0xfffffa9641f26340)
     jbd2/sda2-8-1284  [000] d...363097172629: 0: pagecache_get_page (retval=0xfffffa9641f29440)

^C
=====================================================
Filtered PID : 1284
[Total 10 Hit 6 miss 4]
=====================================================
```

Just run it !!
(copy & make & run)

Stage 09: Count the number of pagecache miss with Filter PID

```
$ cd ~/git/vault19_bpf_tutorial/stage09_pagecache_miss_pid_filter
$ ls
vault19_pagecache_trace_kern.c  vault19_pagecache_trace_user.c

$ cp ./vault19_pagecache_trace_kern.c ~/git/linux/samples/bpf/
$ cp ./vault19_pagecache_trace_user.c ~/git/linux/samples/bpf/

$ cd ~/git/linux/samples/bpf/

$ make
$ sudo ./vault19_pagecache_trace
```

Check the src files !!
(How does it work ?)

Stage 09: Count the number of pagecache miss with Filter PID

```
$ cd ~/git/vault19_bpf_tutorial/stage09_pagecache_miss_pid_filter
$ ls
vault19_pagecache_trace_kern.c  vault19_pagecache_trace_user.c

$ diff stage08_pagecache_miss_counts/vault19_pagecache_trace_kern.c  \
  ./vault19_pagecache_trace_kern.c

$ diff stage08_pagecache_miss_counts/vault19_pagecache_trace_user.c  \
  ./vault19_pagecache_trace_user.c
```

```diff
diff stage08_*/vault19_pagecache_trace_kern.c stage09_pagecache_miss_pid_filter/vault19_pagecache_trace_kern.c
--- stage08_*/vault19_pagecache_trace_kern.c
+++ stage09_*/vault19_pagecache_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
+struct bpf_map_def SEC("maps") filter_pid_map = {
+        .type = BPF_MAP_TYPE_HASH,
+        .key_size = sizeof(u32),
+        .value_size = sizeof(u32),
+        .max_entries = 1,
+};

 SEC("kretprobe/pagecache_get_page")
 int pagecache_get_page_retval(struct pt_regs *ctx) {
         long pagecache_retval = PT_REGS_RC(ctx);
         u64 start_time = bpf_ktime_get_ns();
+        u32 pid = bpf_get_current_pid_tgid();
+        u32 *filter_pid;
+        u32 one_idx = 0;
+
+        filter_pid = bpf_map_lookup_elem(&filter_pid_map, &one_idx);

+        if (!(filter_pid && (*filter_pid != pid))) {
+                bpf_trace_printk(fmt, sizeof(fmt), pagecache_retval);
+                bpf_map_update_elem(&pagecache_retval_map, &start_time, &pagecache_retval, BPF_ANY);
+        }
```

```diff
diff stage08_*/vault19_pagecache_trace_kern.c stage09_pagecache_miss_pid_filter/vault19_pagecache_trace_kern.c
--- stage08_*/vault19_pagecache_trace_kern.c
+++ stage09_*/vault19_pagecache_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
+struct bpf_map_def SEC("maps") filter_pid_map = {
+        .type = BPF_MAP_TYPE_HASH,
+        .key_size = sizeof(u32),
+        .value_size = sizeof(u32),
+        .max_entries = 1,
+};

 SEC("kretprobe/pagecache_get_page")
 int pagecache_get_page_retval(struct pt_regs *ctx) {
         long pagecache_retval = PT_REGS_RC(ctx);
         u64 start_time = bpf_ktime_get_ns();
+        u32 pid = bpf_get_current_pid_tgid();
+        u32 *filter_pid;
+        u32 one_idx = 0;
+
+        filter_pid = bpf_map_lookup_elem(&filter_pid_map, &one_idx);

+        if (!(filter_pid && (*filter_pid != pid))) {
+                bpf_trace_printk(fmt, sizeof(fmt), pagecache_retval);
+                bpf_map_update_elem(&pagecache_retval_map, &start_time, &pagecache_retval, BPF_ANY);
+        }
```

```diff
diff stage08_*/vault19_pagecache_trace_kern.c stage09_pagecache_miss_pid_filter/vault19_pagecache_trace_kern.c
--- stage08_*/vault19_pagecache_trace_kern.c
+++ stage09_*/vault19_pagecache_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
+struct bpf_map_def SEC("maps") filter_pid_map = {
+        .type = BPF_MAP_TYPE_HASH,
+        .key_size = sizeof(u32),
+        .value_size = sizeof(u32),
+        .max_entries = 1,
+};

 SEC("kretprobe/pagecache_get_page")
 int pagecache_get_page_retval(struct pt_regs *ctx) {
         long pagecache_retval = PT_REGS_RC(ctx);
         u64 start_time = bpf_ktime_get_ns();
+        u32 pid = bpf_get_current_pid_tgid();
+        u32 *filter_pid;
+        u32 one_idx = 0;
+
+        filter_pid = bpf_map_lookup_elem(&filter_pid_map, &one_idx);

+        if (!(filter_pid && (*filter_pid != pid))) {
+                bpf_trace_printk(fmt, sizeof(fmt), pagecache_retval);
+                bpf_map_update_elem(&pagecache_retval_map, &start_time, &pagecache_retval, BPF_ANY);
+        }
```

```diff
diff stage08_*/vault19_pagecache_trace_kern.c stage09_pagecache_miss_pid_filter/vault19_pagecache_trace_kern.c
--- stage08_*/vault19_pagecache_trace_kern.c
+++ stage09_*/vault19_pagecache_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
+struct bpf_map_def SEC("maps") filter_pid_map = {
+        .type = BPF_MAP_TYPE_HASH,
+        .key_size = sizeof(u32),
+        .value_size = sizeof(u32),
+        .max_entries = 1,
+};

 SEC("kretprobe/pagecache_get_page")
 int pagecache_get_page_retval(struct pt_regs *ctx) {
         long pagecache_retval = PT_REGS_RC(ctx);
         u64 start_time = bpf_ktime_get_ns();
+        u32 pid = bpf_get_current_pid_tgid();
+        u32 *filter_pid;
+        u32 one_idx = 0;
+
+        filter_pid = bpf_map_lookup_elem(&filter_pid_map, &one_idx);

+        if (!(filter_pid && (*filter_pid != pid))) {
+                bpf_trace_printk(fmt, sizeof(fmt), pagecache_retval);
+                bpf_map_update_elem(&pagecache_retval_map, &start_time, &pagecache_retval, BPF_ANY);
+        }
```

current->tgid << 32 | current->pid

```diff
diff stage08_*/vault19_pagecache_trace_kern.c stage09_pagecache_miss_pid_filter/vault19_pagecache_trace_kern.c
--- stage08_*/vault19_pagecache_trace_kern.c
+++ stage09_*/vault19_pagecache_trace_kern.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
+struct bpf_map_def SEC("maps") filter_pid_map = {
+        .type = BPF_MAP_TYPE_HASH,
+        .key_size = sizeof(u32),
+        .value_size = sizeof(u32),
+        .max_entries = 1,
+};

 SEC("kretprobe/pagecache_get_page")
 int pagecache_get_page_retval(struct pt_regs *ctx) {
         long pagecache_retval = PT_REGS_RC(ctx);
         u64 start_time = bpf_ktime_get_ns();
+        u32 pid = bpf_get_current_pid_tgid();
+        u32 *filter_pid;
+        u32 one_idx = 0;
+
+        filter_pid = bpf_map_lookup_elem(&filter_pid_map, &one_idx);

+        if (!(filter_pid && (*filter_pid != pid))) {
+                bpf_trace_printk(fmt, sizeof(fmt), pagecache_retval);
+                bpf_map_update_elem(&pagecache_retval_map, &start_time, &pagecache_retval, BPF_ANY);
+        }
```

```diff
diff stage08_*/vault19_pagecache_trace_user.c stage09_pagecache_miss_pid_filter/vault19_pagecache_trace_user.c
--- stage08_*/vault19_pagecache_trace_user.c
+++ stage09_*/vault19_pagecache_trace_user.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
 #include <bpf/bpf.h>
 #include "bpf_load.h"

+__u32 pid_filter;

 int main(int argc, char **argv) {
         if (load_bpf_file(filename)) {
                 printf("%s", bpf_log_buf);
                 return 1;
         }

+        pid_filter_map = map_fd[1];
+        if (argc > 1) {
+                pid_filter = atoi(argv[1]);
+                bpf_map_update_elem(pid_filter_map, &one_idx, &pid_filter, BPF_ANY);
+                printf("pid_filter_map update: (pid=%u)\n", pid_filter);
+        }
+
```

```diff
diff stage08_*/vault19_pagecache_trace_user.c stage09_pagecache_miss_pid_filter/vault19_pagecache_trace_user.c
--- stage08_*/vault19_pagecache_trace_user.c
+++ stage09_*/vault19_pagecache_trace_user.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
 #include <bpf/bpf.h>
 #include "bpf_load.h"

+__u32 pid_filter;

 int main(int argc, char **argv) {
         if (load_bpf_file(filename)) {
                 printf("%s", bpf_log_buf);
                 return 1;
         }

+        pid_filter_map = map_fd[1];
+        if (argc > 1) {
+                pid_filter = atoi(argv[1]);
+                bpf_map_update_elem(pid_filter_map, &one_idx, &pid_filter, BPF_ANY);
+                printf("pid_filter_map update: (pid=%u)\n", pid_filter);
+        }
+
```

```diff
diff stage08_*/vault19_pagecache_trace_user.c stage09_pagecache_miss_pid_filter/vault19_pagecache_trace_user.c
--- stage08_*/vault19_pagecache_trace_user.c
+++ stage09_*/vault19_pagecache_trace_user.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
 #include <bpf/bpf.h>
 #include "bpf_load.h"

+__u32 pid_filter;

 int main(int argc, char **argv) {
         if (load_bpf_file(filename)) {
                 printf("%s", bpf_log_buf);
                 return 1;
         }

+        pid_filter_map = map_fd[1];
+        if (argc > 1) {
+                pid_filter = atoi(argv[1]);
+                bpf_map_update_elem(pid_filter_map, &one_idx, &pid_filter, BPF_ANY);
+                printf("pid_filter_map update: (pid=%u)\n", pid_filter);
+        }
+
```

```diff
diff stage08_*/vault19_pagecache_trace_user.c stage09_pagecache_miss_pid_filter/vault19_pagecache_trace_user.c
--- stage08_*/vault19_pagecache_trace_user.c
+++ stage09_*/vault19_pagecache_trace_user.c
@@ -3,23 +3,53 @@ #include <linux/ptrace.h>
 #include <bpf/bpf.h>
 #include "bpf_load.h"

+__u32 pid_filter;

 int main(int argc, char **argv) {
         if (load_bpf_file(filename)) {
                 printf("%s", bpf_log_buf);
                 return 1;
         }

+        pid_filter_map = map_fd[1];
+        if (argc > 1) {
+                pid_filter = atoi(argv[1]);
+                bpf_map_update_elem(pid_filter_map, &one_idx, &pid_filter, BPF_ANY);
+                printf("pid_filter_map update: (pid=%u)\n", pid_filter);
+        }
+
```

```
=====================================================
Filtered PID : 1284
[Total 10 Hit 6 miss 4]
=====================================================
```

Stage 10: Class(Data + Functions) Tracing

```
# Let’s get functions that have ‘struct bio’ pointer parameters using ‘pfunct’ of dwarves
#
$ pfunct --help
Usage: pfunct [OPTION...] FILE

  -c, --class=CLASS          functions that have CLASS pointer parameters
...
  -P, --prototypes           show function prototypes
```

```
# It is able to get functions that have 'struct bio' pointer parameters
#
$ pfunct --help
Usage: pfunct [OPTION...] FILE

  -c, --class=CLASS          functions that have CLASS pointer parameters
...
  -P, --prototypes           show function prototypes
```

```
struct bio {
    struct bio * bi_next;
...
    struct bvec_iter bi_iter;
...
};
```

```
# It is able to get all function prototypes
#
$ pfunct --help
Usage: pfunct [OPTION...] FILE

  -c, --class=CLASS          functions that have CLASS pointer parameters
...
  -P, --prototypes           show function prototypes
```

```
# It is able to get all function prototypes
#
$ pfunct --help
Usage: pfunct [OPTION...] FILE

  -c, --class=CLASS          functions that have CLASS pointer parameters
...
  -P, --prototypes           show function prototypes
```

```
blk_qc_t submit_bio (struct bio * bio);
```

```
# For example,
#
$ pfunct --class=bio blk-core.o | grep -v trace_
blk_rq_bio_prep
submit_bio
direct_make_request
generic_make_request
generic_make_request_checks
should_fail_bio
handle_bad_sector
blk_init_request_from_bio
blk_attempt_plug_merge
bio_attempt_discard_merge
bio_attempt_back_merge
bio_attempt_front_merge
```

```
# For example,
#
$ pfunct --prototypes blk-core.o | grep -ve "trace_\|(void)\|inline" | grep “struct bio”
...
void req_bio_endio(struct request * rq, struct bio * bio, unsigned int nbytes, ...
int should_fail_bio(struct bio * bio);
bool generic_make_request_checks(struct bio * bio);
blk_qc_t generic_make_request(struct bio * bio);
blk_qc_t direct_make_request(struct bio * bio);
blk_qc_t submit_bio(struct bio * bio);
...
```

How to trace Data + Functions

focusing on ‘struct bio’ ?

Just run it !!

Stage 10: Class(Data + Functions) Tracing 'struct bio'

```
$ ls ~/git/vault19_bpf_tutorial/stage10_ctracer-data+func_tracing
# collecting kernel function debug info from 'vmlinux'
# and build 'vmlinux.debuginfo/kfunc.json'
$ ./setup.py vmlinux

# generate BPF programs(kernel) based on 'struct bio' using bpf/utils/kern/bio-bpf.c
# with functions that have the struct type parameters
$ ./gen-BPF-cfiles.py bio

# check generated BPF c files
$ ls bpf/bpf-kern-progs/bpf-kprogs-0/
```

```
# compile generated BPF c files in ~/git/linux/samples/bpf/
$ ./ctracer-compile.py bpf/bpf-kern-progs/bpf-kprogs-0/

# load and pin the BPF programs to /sys/fs/bpf/
$ sudo ./ctracer-load.py ~/git/linux/samples/bpf

# Data + Function tracing:
# Collect all 'struct bio' data with call trace of functions that has its parameters
$ sudo ./vault19_ctracer

Stop by 'Ctrl + c'

# check recorded ctracer data: ftrace.data and ctracer.json
$ ls /tmp

# build ftrace.data.arg and ctracer.json.srcline with arg type / srcline info
$ ./ctracer-finish.py

$ scp /tmp/ftrace.data.arg <username>@<ip>:<path>
$ scp /tmp/ctracer.json <username>@<ip>:<path>
#or
$ scp /tmp/ctracer.json.srcline <username>@<ip>:<path>

# open webview: 127.0.0.1:5000 and upload the two files
```

BPF hands-on tutorials: **Ctracer webview: Data (struct bio) trace**

"class trace"

+ Functions(that have struct bio parameter) trace

Code View | Clear Select | Select Between | Show Only > ☑ data ☐ select | Filter CPU > ☑ 0 ☑ X

25190542114013	0		submit_bio(struct bio* bio) {
	0		generic_make_request(struct bio* bio) {
	0		generic_make_request_checks(struct bio* bio) {
	0	0.794 us	should_fail_bio(struct bio* bio) {
	0		blk_mq_make_request(struct request_queue* q, struct bio* bio) {
	0		blk_attempt_plug_merge(struct request_queue* q, struct bio* bio, struct request** same_queue_rq) {
	0	0.968 us	blk_rq_merge_ok(struct request* rq, struct bio* bio) {
	0	0.825 us	blk_try_merge(struct request* rq, struct bio* bio) {
	0		__blk_mq_sched_bio_merge(struct request_queue* q, struct bio* bio) {
	0		dd_bio_merge(struct blk_mq_hw_ctx* hctx, struct bio* bio) {
	0		blk_mq_sched_try_merge(struct request_queue* q, struct bio* bio, struct request** merged_request) {
	0		elv_merge(struct request_queue* q, struct request** req, struct bio* bio) {
	0		dd_request_merge(struct request_queue* q, struct request** rq, struct bio* bio) {
	0		blk_mq_get_request(struct request_queue* q, struct bio* bio, struct blk_mq_alloc_data* data) {
25190542254964	0	0.822 us	dd_prepare_request(struct request* rq, struct bio* bio) {
25190542263728	0		blk_init_request_from_bio(struct request* req, struct bio* bio) {
25190542271221	0		blk_rq_bio_prep(struct request_queue* q, struct request* rq, struct bio* bio) {
25190542278984	0	0.801 us	bio_phys_segments(struct request_queue* q, struct bio* bio) {
25190542310967	0		bio_associate_blkg(struct bio* bio) {
25190542319894	0		bio_associate_blkg_from_css(struct bio* bio, struct cgroup_subsys_state* css) {
25190542338322	0		bio_add_page(struct bio* bio, struct page* page, unsigned int len, unsigned int offset) {
25190542352810	0	0.792 us	__bio_try_merge_page(struct bio* bio, struct page* page, unsigned int len, unsigned int off) {
25190542368122	0	0.794 us	__bio_add_page(struct bio* bio, struct page* page, unsigned int len, unsigned int off) {

magic	1700885511
sequence	0
time	25190542096911 -> 25190542108727
sector	0
bytes	4096
action	4294967295
pid	0
device	0
cpu	0
error	0
pdu_len	0

Thanks

Appendix

uftrace

Function tracer for C/C++ programs

- created by Namhyung Kim
  - LG Electronics open-source contribution team
  - Linux kernel developer (since 2010)
    - perf, ftrace, …

- inspired by ftrace framework in the kernel
- **record** and **replay** model

Search or jump to… / Pull requests Issues Marketplace Explore

namhyung / uftrace

Unwatch 79 | Unstar 996 | Fork 153

Code | Issues 62 | Pull requests 19 | Wiki | Insights

Function (graph) tracer for user-space https://uftrace.github.io/slide/

trace function tracer

2,986 commits | 13 branches | 13 releases | 28 contributors | GPL-2.0

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

namhyung Merge pull request #575 from gy741/fix_code ··· Latest commit 63f73ed 2 days ago

arch	mcount: Fix Unchecked Return Value in mcount_setup_trampoline()	a month ago
check-deps	build: removed stringop_truncation supression	5 days ago
cmds	record: fix GCC8 string truncation warning	6 days ago

README.md

build passing coverity passed

uftrace

The uftrace tool is to trace and analyze execution of a program written in C/C++. It was heavily inspired by the ftrace framework of the Linux kernel (especially function graph tracer) and supports userspace programs. It supports various kind of commands and filters to help analysis of the program execution and performance.

```
$ sudo uftrace -k a.out
[sudo] password for honggyu:
Hello World!
# DURATION    TID     FUNCTION
   1.116 us [ 6267] | __monstartup();
   0.603 us [ 6267] | __cxa_atexit();
            [ 6267] | main() {
            [ 6267] |   printf() {
            [ 6267] |     sys_newfstat() {
```

```
$ gcc -pg -o fibonacci tests/s-fibonacci.c

$ uftrace -A fib@arg1 -R fib@retval fibonacci 5
# DURATION    TID     FUNCTION
   0.633 us [ 2851] | __monstartup();
   0.480 us [ 2851] | __cxa_atexit();
            [ 2851] | main() {
   0.546 us [ 2851] |   atoi();
            [ 2851] |   fib(5) {
            [ 2851] |     fib(4) {
            [ 2851] |       fib(3) {
   1.146 us [ 2851] |         fib(2) = 1;
   0.077 us [ 2851] |         fib(1) = 1;
   1.823 us [ 2851] |       } = 2; /* fib */
   0.062 us [ 2851] |       fib(2) = 1;
   2.199 us [ 2851] |     } = 3; /* fib */
            [ 2851] |     fib(3) {
   0.061 us [ 2851] |       fib(2) = 1;
   0.067 us [ 2851] |       fib(1) = 1;
   0.474 us [ 2851] |     } = 2; /* fib */
   3.317 us [ 2851] |   } = 5; /* fib */
   4.343 us [ 2851] | } /* main */
```

uftrace

“C/C++ execution flow”

```
# Default == record + replay

$ uftrace -A fib@arg1 -R fib@retval fibonacci 5
# DURATION    TID     FUNCTION
   0.633 us [ 2851] | __monstartup();
   0.480 us [ 2851] | __cxa_atexit();
            [ 2851] | main() {
   0.546 us [ 2851] |   atoi();
            [ 2851] |   fib(5) {
            [ 2851] |     fib(4) {
            [ 2851] |       fib(3) {
   1.146 us [ 2851] |         fib(2) = 1;
   0.077 us [ 2851] |         fib(1) = 1;
   1.823 us [ 2851] |       } = 2; /* fib */
   0.062 us [ 2851] |       fib(2) = 1;
   2.199 us [ 2851] |     } = 3; /* fib */
            [ 2851] |     fib(3) {
   0.061 us [ 2851] |       fib(2) = 1;
   0.067 us [ 2851] |       fib(1) = 1;
   0.474 us [ 2851] |     } = 2; /* fib */
   3.317 us [ 2851] |   } = 5; /* fib */
   4.343 us [ 2851] | } /* main */
```

uftrace

“C/C++ execution flow”

Function Call Trace

```
# Default == record + replay

$ uftrace -A fib@arg1 -R fib@retval fibonacci 5
# DURATION     TID     FUNCTION
   0.633 us [ 2851] | __monstartup();
   0.480 us [ 2851] | __cxa_atexit();
            [ 2851] | main() {
   0.546 us [ 2851] |   atoi();
            [ 2851] |   fib(5) {
            [ 2851] |     fib(4) {
            [ 2851] |       fib(3) {
   1.146 us [ 2851] |         fib(2) = 1;
   0.077 us [ 2851] |         fib(1) = 1;
   1.823 us [ 2851] |       } = 2; /* fib */
   0.062 us [ 2851] |       fib(2) = 1;
   2.199 us [ 2851] |     } = 3; /* fib */
            [ 2851] |     fib(3) {
   0.061 us [ 2851] |       fib(2) = 1;
   0.067 us [ 2851] |       fib(1) = 1;
   0.474 us [ 2851] |     } = 2; /* fib */
   3.317 us [ 2851] |   } = 5; /* fib */
   4.343 us [ 2851] | } /* main */
```

uftrace

“C/C++ each function

Execution time”

```
# Default == record + replay

$ uftrace -A fib@arg1 -R fib@retval fibonacci 5
# DURATION     TID     FUNCTION
   0.633 us [ 2851] | __monstartup();
   0.480 us [ 2851] | __cxa_atexit();
            [ 2851] | main() {
   0.546 us [ 2851] |   atoi();
            [ 2851] |   fib(5) {
            [ 2851] |     fib(4) {
            [ 2851] |       fib(3) {
   1.146 us [ 2851] |         fib(2) = 1;
   0.077 us [ 2851] |         fib(1) = 1;
   1.823 us [ 2851] |       } = 2; /* fib */
   0.062 us [ 2851] |       fib(2) = 1;
   2.199 us [ 2851] |     } = 3; /* fib */
            [ 2851] |     fib(3) {
   0.061 us [ 2851] |       fib(2) = 1;
   0.067 us [ 2851] |       fib(1) = 1;
   0.474 us [ 2851] |     } = 2; /* fib */
   3.317 us [ 2851] |   } = 5; /* fib */
   4.343 us [ 2851] | } /* main */
```

uftrace

“Arguments”

based on
Function Call Trace

```
# Default == record + replay

$ uftrace -A fib@arg1 -R fib@retval fibonacci 5
# DURATION     TID     FUNCTION
   0.633 us [ 2851] | __monstartup();
   0.480 us [ 2851] | __cxa_atexit();
            [ 2851] | main() {
   0.546 us [ 2851] |   atoi();
            [ 2851] |   fib(5) {
            [ 2851] |     fib(4) {
            [ 2851] |       fib(3) {
   1.146 us [ 2851] |         fib(2) = 1;
   0.077 us [ 2851] |         fib(1) = 1;
   1.823 us [ 2851] |       } = 2; /* fib */
   0.062 us [ 2851] |       fib(2) = 1;
   2.199 us [ 2851] |     } = 3; /* fib */
            [ 2851] |     fib(3) {
   0.061 us [ 2851] |       fib(2) = 1;
   0.067 us [ 2851] |       fib(1) = 1;
   0.474 us [ 2851] |     } = 2; /* fib */
   3.317 us [ 2851] |   } = 5; /* fib */
   4.343 us [ 2851] | } /* main */
```

uftrace

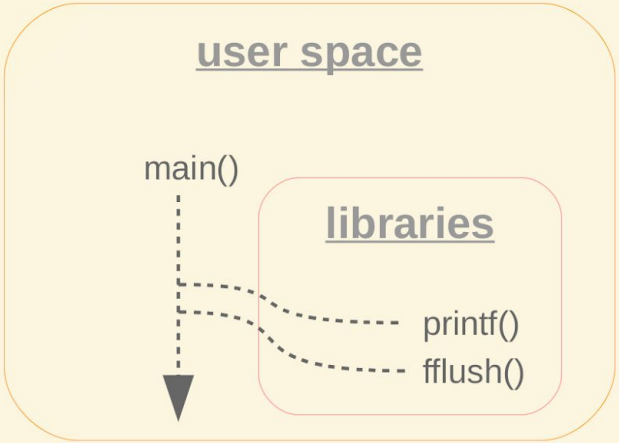
“Return values”

based on
Function Call Trace

uftrace can trace User + Lib + Kernel

showing the execution flow

```
$ uftrace record hello
Hello OSSEU17 !!

$ uftrace replay
# DURATION    TID     FUNCTION
   0.710 us [13588] | __monstartup();
   0.713 us [13588] | __cxa_atexit();
            [13588] | main() {
   4.107 us [13588] |   printf();
   4.046 us [13588] |   fflush();
   8.815 us [13588] | } /* main */
```



```
$ uftrace record -k hello
Hello OSSEU17 !!

$ uftrace replay
# DURATION    TID     FUNCTION
   1.060 us [13565] | __monstartup();
   1.113 us [13565] | __cxa_atexit();
            [13565] | main() {
            [13565] |   printf() {
   3.173 us [13565] |     sys_newfstat();
   6.107 us [13565] |     __do_page_fault();
  17.713 us [13565] |   } /* printf */
            [13565] |   fflush() {
   7.198 us [13565] |     sys_write();
  12.270 us [13565] |   } /* fflush */
  30.661 us [13565] | } /* main */
```

Integrated tracer !