BPF infrastructure in kernel

Taeung Song

KossLab

SOSCON 2018

SAMSUNG OPEN SOURCE CONFERENCE 2018

BPF 란 ?

1. Berkeley Packet Filter since 1992
2. Kernel Infrastructure

BPF 란 ?

1. Berkeley Packet Filter since 1992
2. **Kernel Infrastructure**
   - **Interpreter** in-kernel virtual machine
   - **Hook points** in-kernel callback point
   - **Map**
   - **Helper**

BPF 란 ?

1. Berkeley Packet Filter since 1992
2. **Kernel Infrastructure**

"**Safe** dynamic programs and tools"

BPF 란 ?

1. Berkeley Packet Filter since 1992
2. **Kernel Infrastructure**

“**Safe** dynamic programs and tools”

"런타임중 안전하게 커널코드를 삽입하는 기술"

BPF Infrastructure:
안전한 code injection 작전

1) Native 머신코드 대신 **BPF instruction** 을 활용하자

2) **Verifier** 를 통해 위험요소를 미리검사하자

3) (기존)커널함수가 필요할때 **Helper** 함수를 통해서만 호출하자

BPF Infrastructure:
안전한 code injection 작전

1) Native 머신코드 대신 **BPF instruction** 을 활용하자

BPF Infrastructure:
안전한 code injection 작전

2) Verifier 를 통해 위험요소를 미리검사하자

BPF Infrastructure:
안전한 code injection 작전

3) (기존)커널함수가 필요할때 Helper 함수를 통해서만 호출하자

BPF Infrastructure:

안전한 code injection 의 활용 case ?

- Networking (XDP, ovs, tc, offload, ...)
- Tracing, Monitoring, Debugging
- BPF-ization: from daemon to BPF programs

...

BPF Infrastructure:
안전한 code injection 위한 기반기술

Kernel += BPF Interpreter in-kernel virtual machine

\+ Verifier

\+ BPF Helper 함수 추가 leveraging kernel func

\+ BPF syscall prog/map: loading & attaching 등

BPF Instruction set

BPF Instruction set:

1) 주니어 x86 Instruction set 'simplified x86'

(참고: PLUMgrind의 x86 bytecode verifier 실패)'

2) BPF = classic BPF:10% + x86:70% + arm64:25% + risc:5%

3) Instruction encoding 사이즈 고정

(for high interpreter speed)

4) 간소화 -> 위험을 예측하고 예방하기 수월

(Verifier를 통한 loop, memory access 범위 점검 등)

5) Architecture-independent

참조: bpf collabsummit 2015

BPF calling convention:

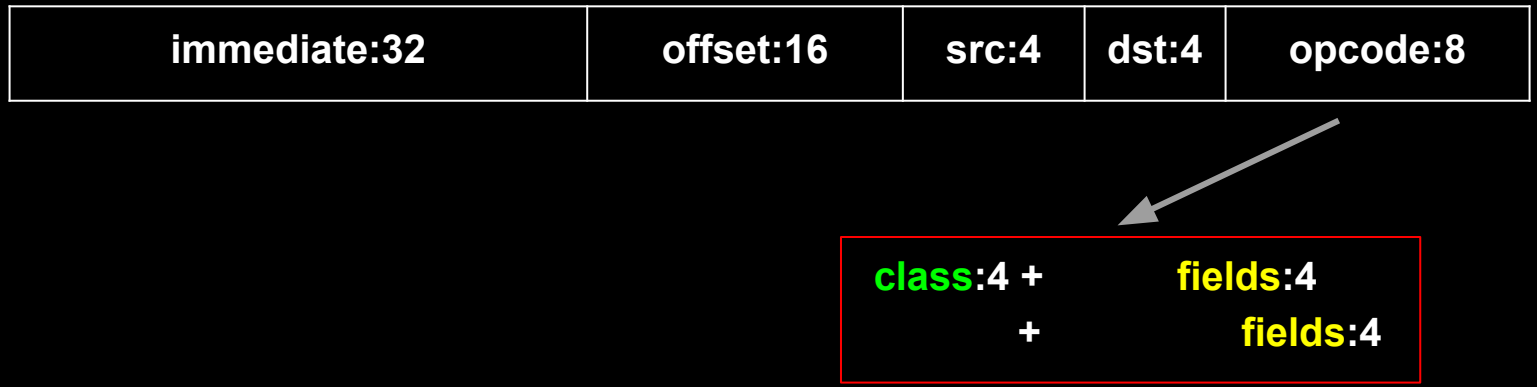
- R0: return value (ex. rax)
- R1 ~ R5: function arguments (ex. rdi, rsi, rdx, ...)
- R6..R9: callee saved
- R10: frame pointer (ex. rbp)

BPF Instruction set:

immediate:32	offset:16	src:4	dst:4	opcode:8

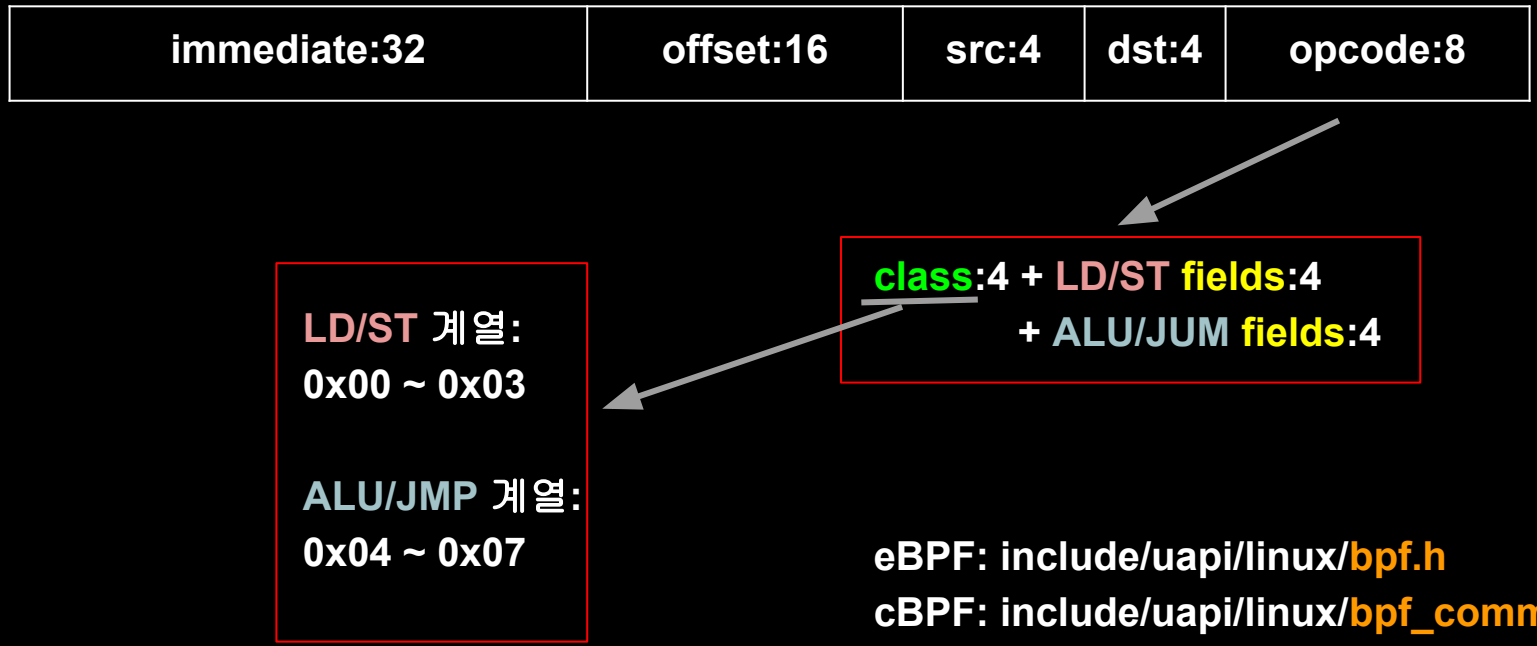
```
$ cat include/uapi/linux/bpf.h
[...]
struct bpf_insn {
        __u8    code;           /* opcode */
        __u8    dst_reg:4;      /* dest register */
        __u8    src_reg:4;      /* source register */
        __s16   off;            /* signed offset */
        __s32   imm;            /* signed immediate constant */
};
[...]
```

BPF Instruction set:



eBPF: include/uapi/linux/bpf.h
cBPF: include/uapi/linux/bpf_common.h

BPF Instruction set:



BPF Instruction set:

class:4 + LD/ST fields:4
+ ALU/JUM fields:4

LD/ST 계열:
0x00 ~ 0x03

ALU/JMP 계열:
0x04 ~ 0x07

eBPF: include/uapi/linux/bpf.h
cBPF: include/uapi/linux/bpf_common.h

BPF programming:

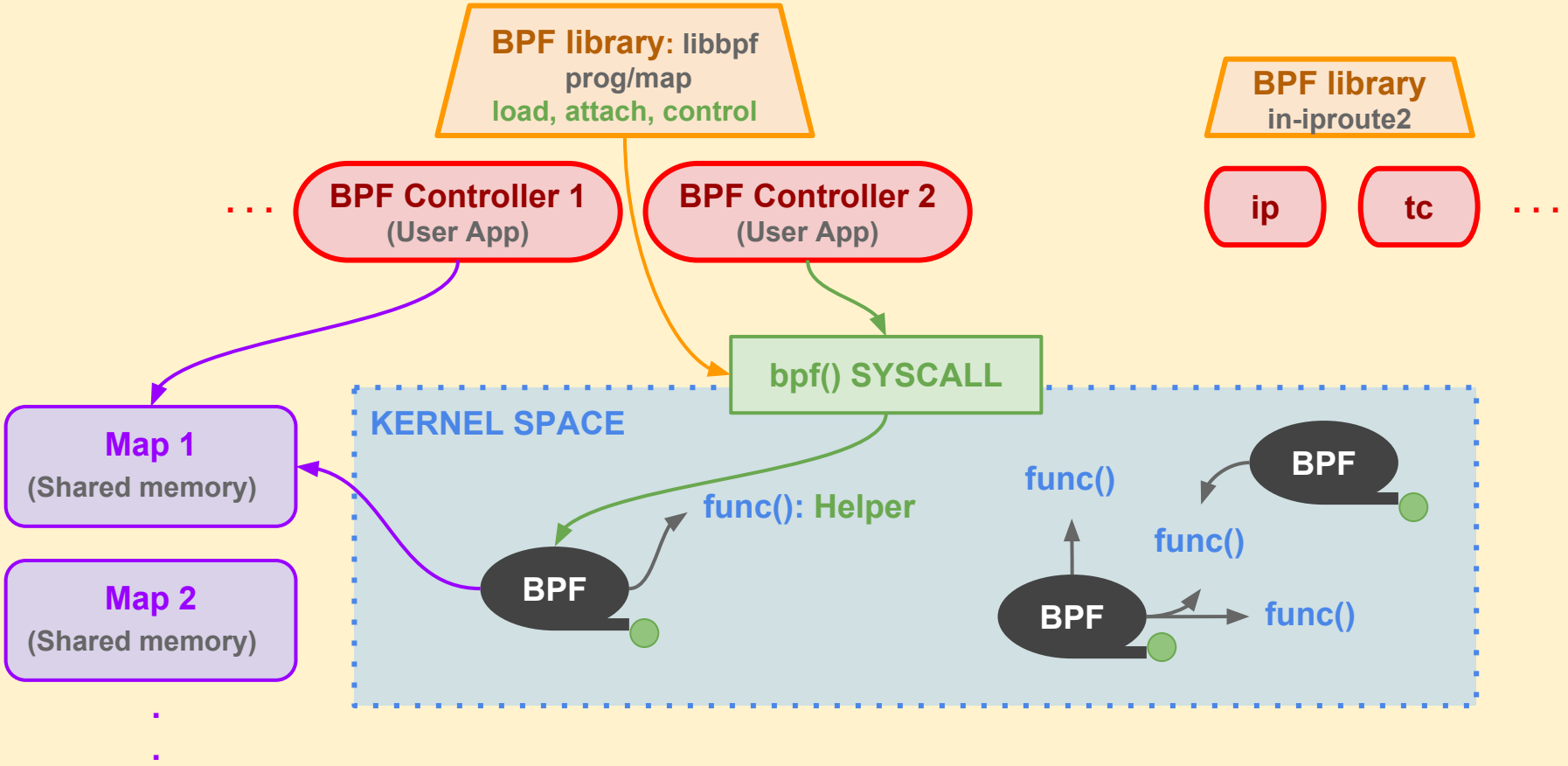
```
struct bpf_insn prog[] = {
BPF_MOV64_REG(BPF_REG_6, BPF_REG_1),
BPF_LD_ABS(BPF_B, ETH_HLEN + offsetof(struct iphdr, protocol) /* R0 = ip->proto */),
BPF_STX_MEM(BPF_W, BPF_REG_10, BPF_REG_0, -4), /* *(u32 *)(fp - 4) = r0 */
BPF_MOV64_REG(BPF_REG_2, BPF_REG_10),
BPF_ALU64_IMM(BPF_ADD, BPF_REG_2, -4), /* r2 = fp - 4 */
BPF_LD_MAP_FD(BPF_REG_1, map_fd),
BPF_RAW_INSN(BPF_JMP | BPF_CALL, 0, 0, 0, BPF_FUNC_map_lookup_elem),
BPF_JMP_IMM(BPF_JEQ, BPF_REG_0, 0, 2),
BPF_MOV64_IMM(BPF_REG_1, 1), /* r1 = 1 */
BPF_RAW_INSN(BPF_STX | BPF_XADD | BPF_DW, BPF_REG_0, BPF_REG_1, 0, 0), /* xadd r0 += r1 */
BPF_MOV64_IMM(BPF_REG_0, 0), /* r0 = 0 */
BPF_EXIT_INSN(),
};
```

https://git.kernel.org/pub/scm/linux/kernel/git/bpf/bpf.git/tree/samples/bpf/sock_example.c

BPF Helper 함수:

```
$ grep BPF_CALL
kernel/bpf/helpers.c:
BPF_CALL_2(bpf_map_lookup_elem, struct bpf_map *, map, void *, key)
BPF_CALL_4(bpf_map_update_elem, struct bpf_map *, map, void *, key,
[...]
kernel/trace/bpf_trace.c:
BPF_CALL_2(bpf_override_return, struct pt_regs *, regs, unsigned long, rc)
BPF_CALL_3(bpf_probe_read, void *, dst, u32, size, const void *, unsafe_ptr)
BPF_CALL_3(bpf_probe_write_user, void *, unsafe_ptr, const void *, src,
BPF_CALL_5(bpf_trace_printk, char *, fmt, u32, fmt_size, u64, arg1,
[...]
net/core/filter.c:
BPF_CALL_1(bpf_skb_get_pay_offset, struct sk_buff *, skb)
BPF_CALL_3(bpf_skb_get_nlattr, struct sk_buff *, skb, u32, a, u32, x)
[...]
```

BPF Architecture:



BPF internals

BPF Infrastructure:

1) Verifier / Interpreter / JIT
2) Hook points in-kernel callback point
3) Map user-to-kernel shared memory
4) helper를 통한 커널함수호출 leveraging
5) Object pinning /sys/fs/bpf/…

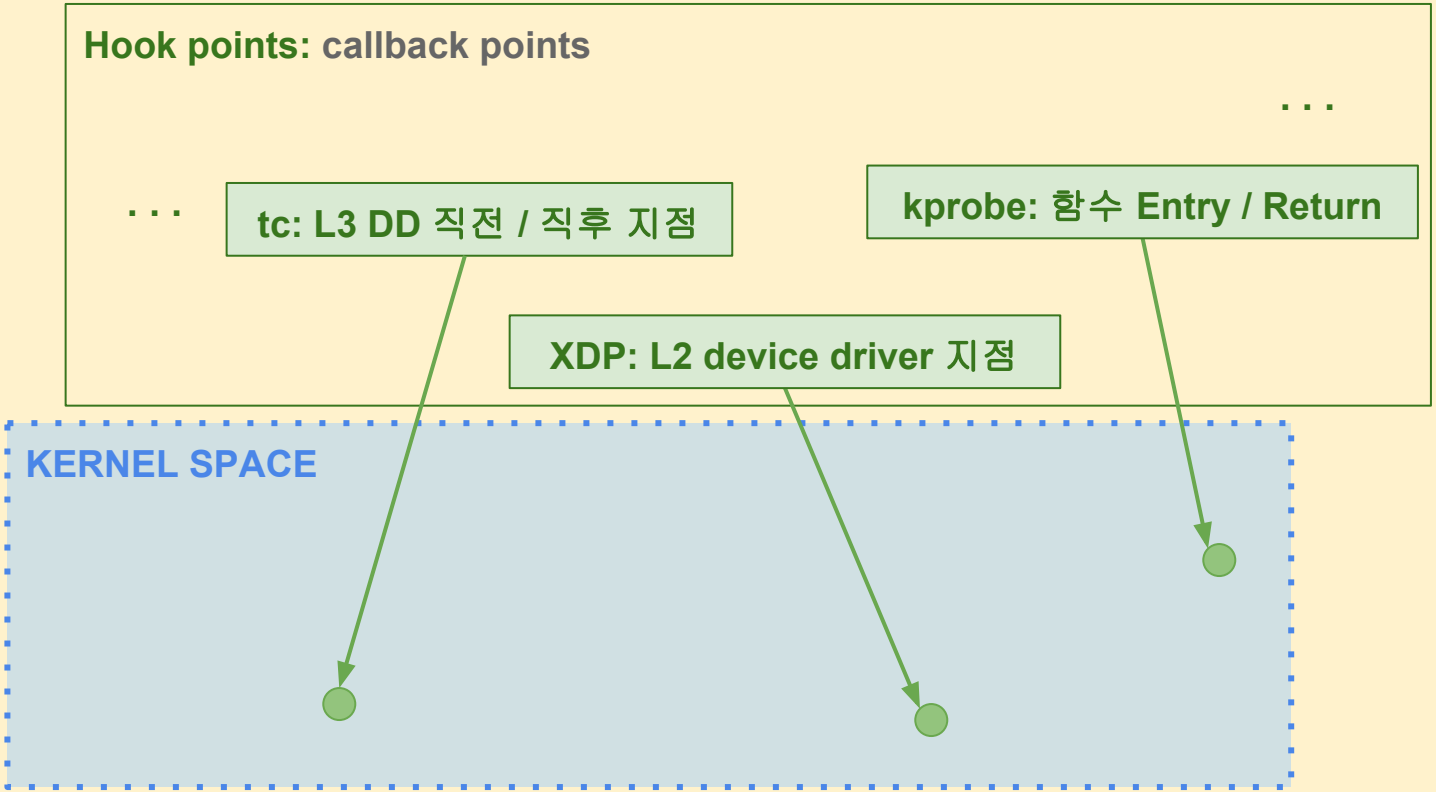
...

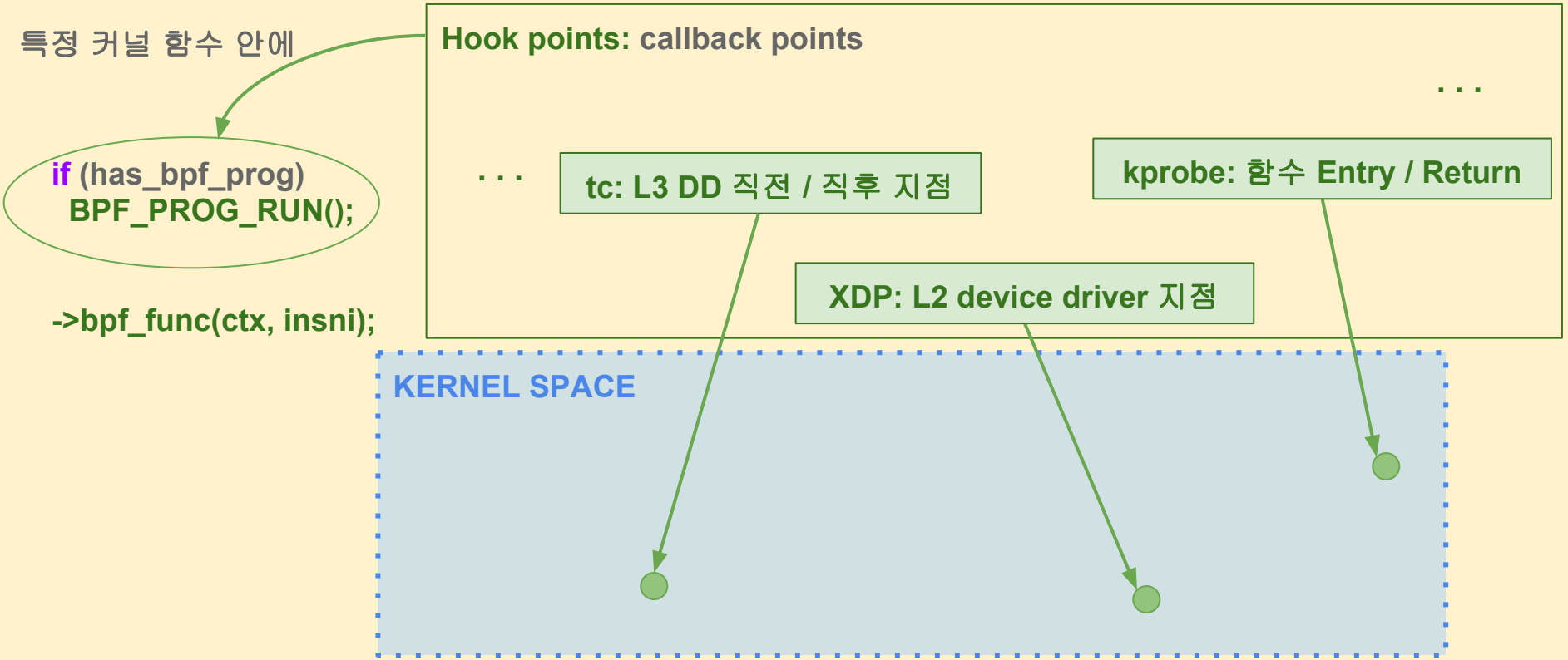
BPF Infrastructure:

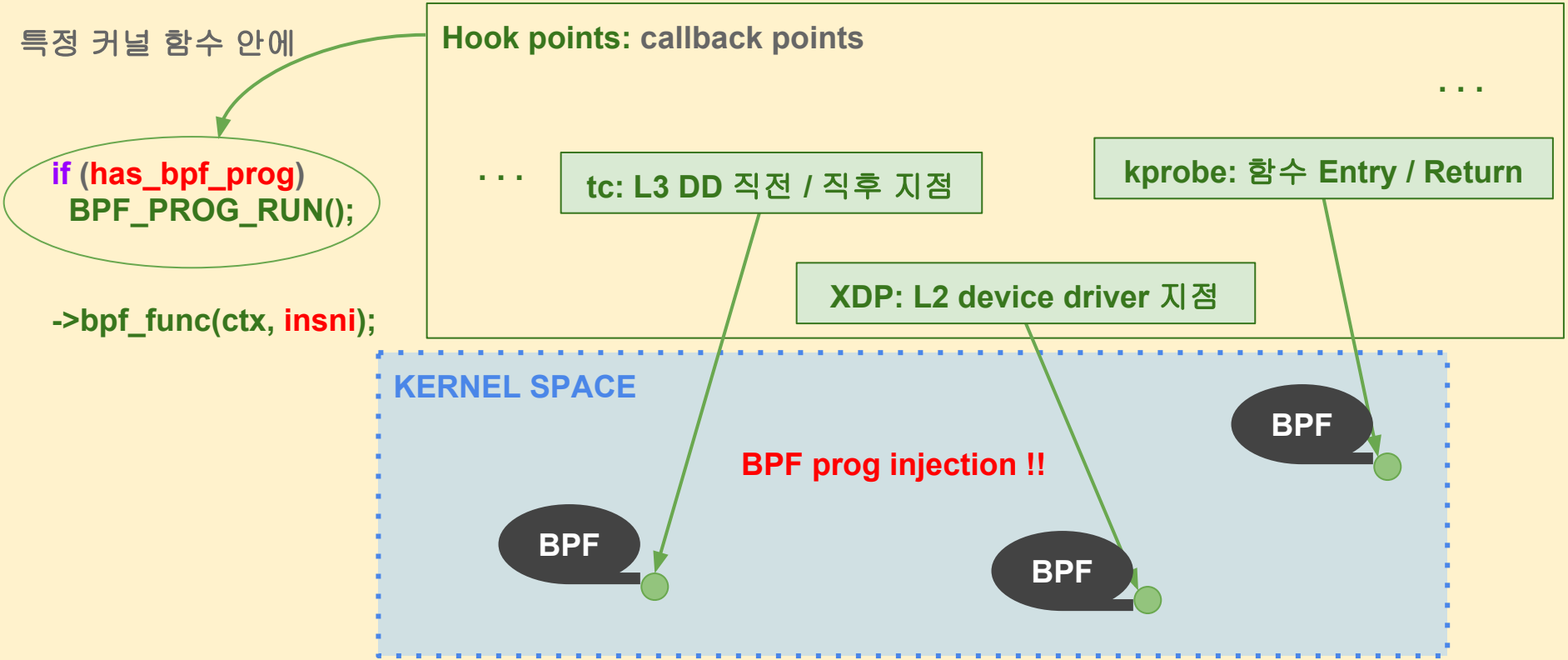
LOAD → ATTACH → CALLBACK

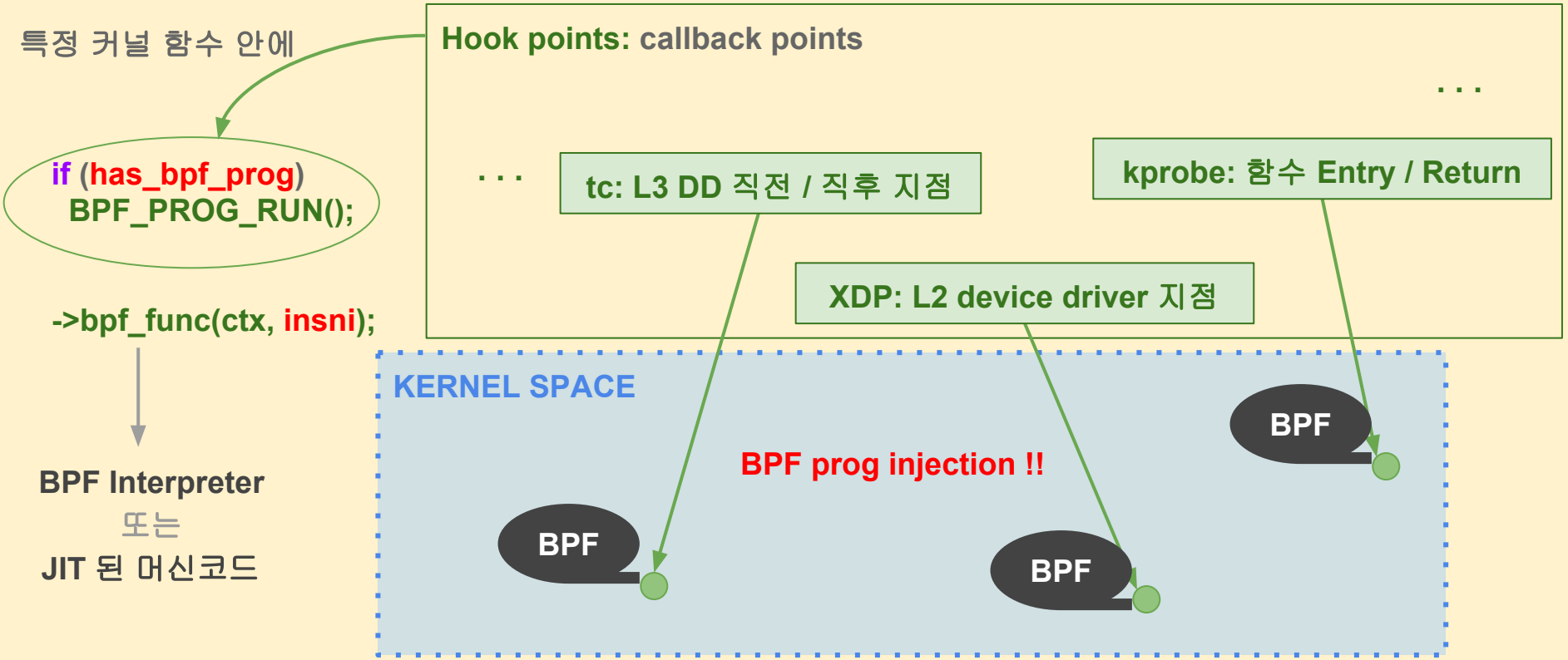
1) Verifier / Interpreter / JIT
2) Hook points in-kernel callback point
3) Map user-to-kernel shared memory
4) helper를 통한 커널함수호출 leveraging
5) Object pinning /sys/fs/bpf/…

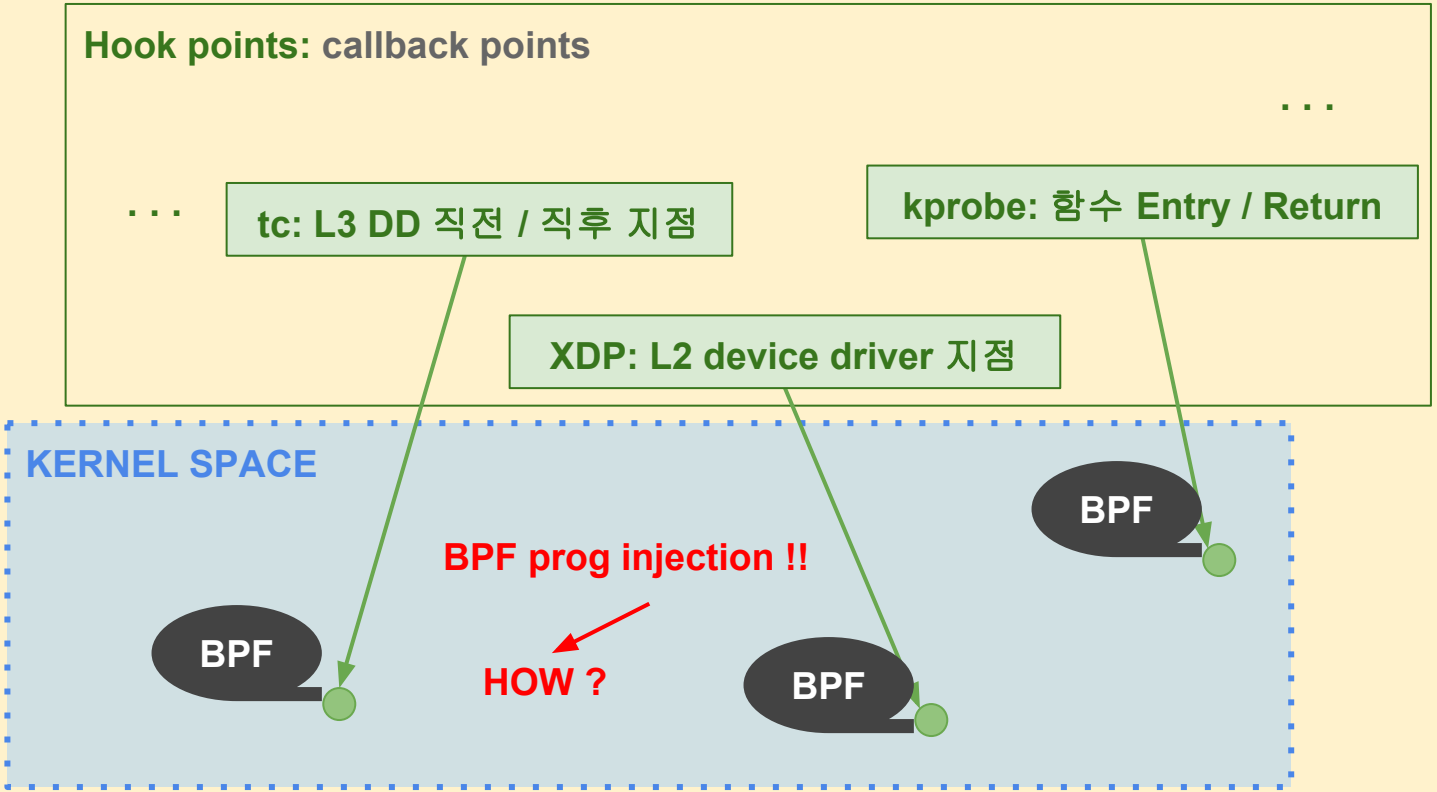
...

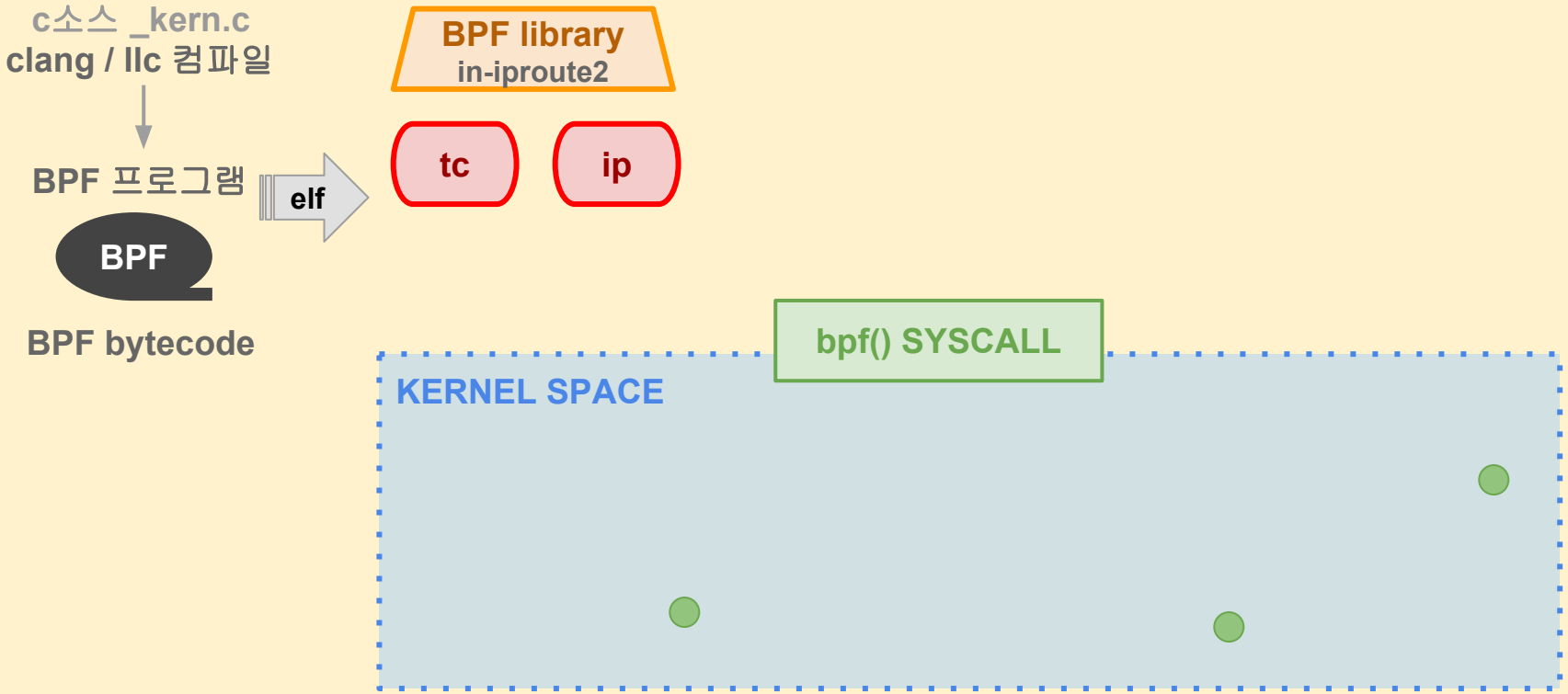

Hook points: callback points
. . .
. . .
tc: L3 DD 직전 / 직후 지점
kprobe: 함수 Entry / Return
XDP: L2 device driver 지점
KERNEL SPACE

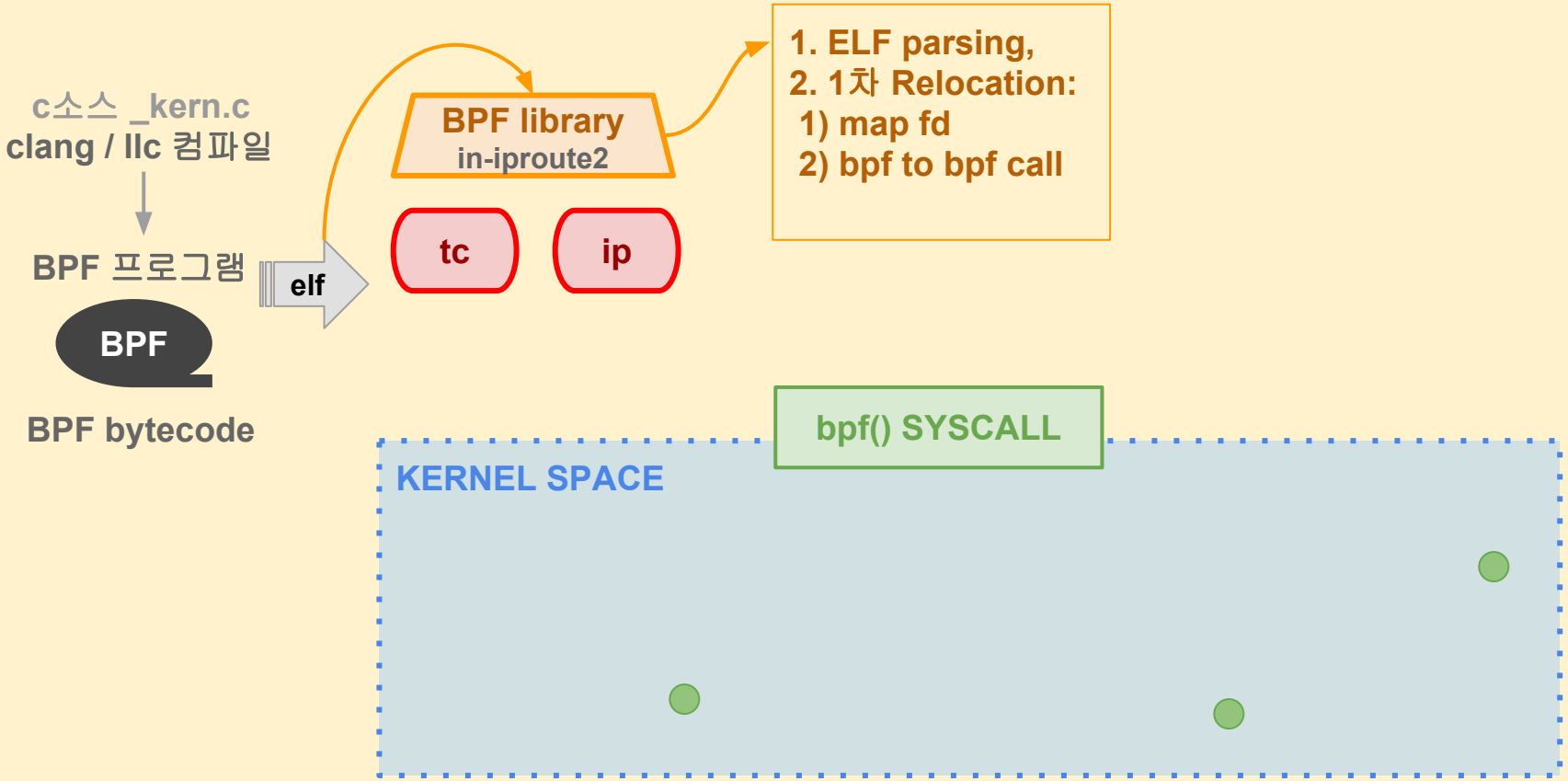

특정 커널 함수 안에
if (has_bpf_prog)
BPF_PROG_RUN();
->bpf_func(ctx, insni);
Hook points: callback points
. . .
. . .
tc: L3 DD 직전 / 직후 지점
kprobe: 함수 Entry / Return
XDP: L2 device driver 지점
KERNEL SPACE

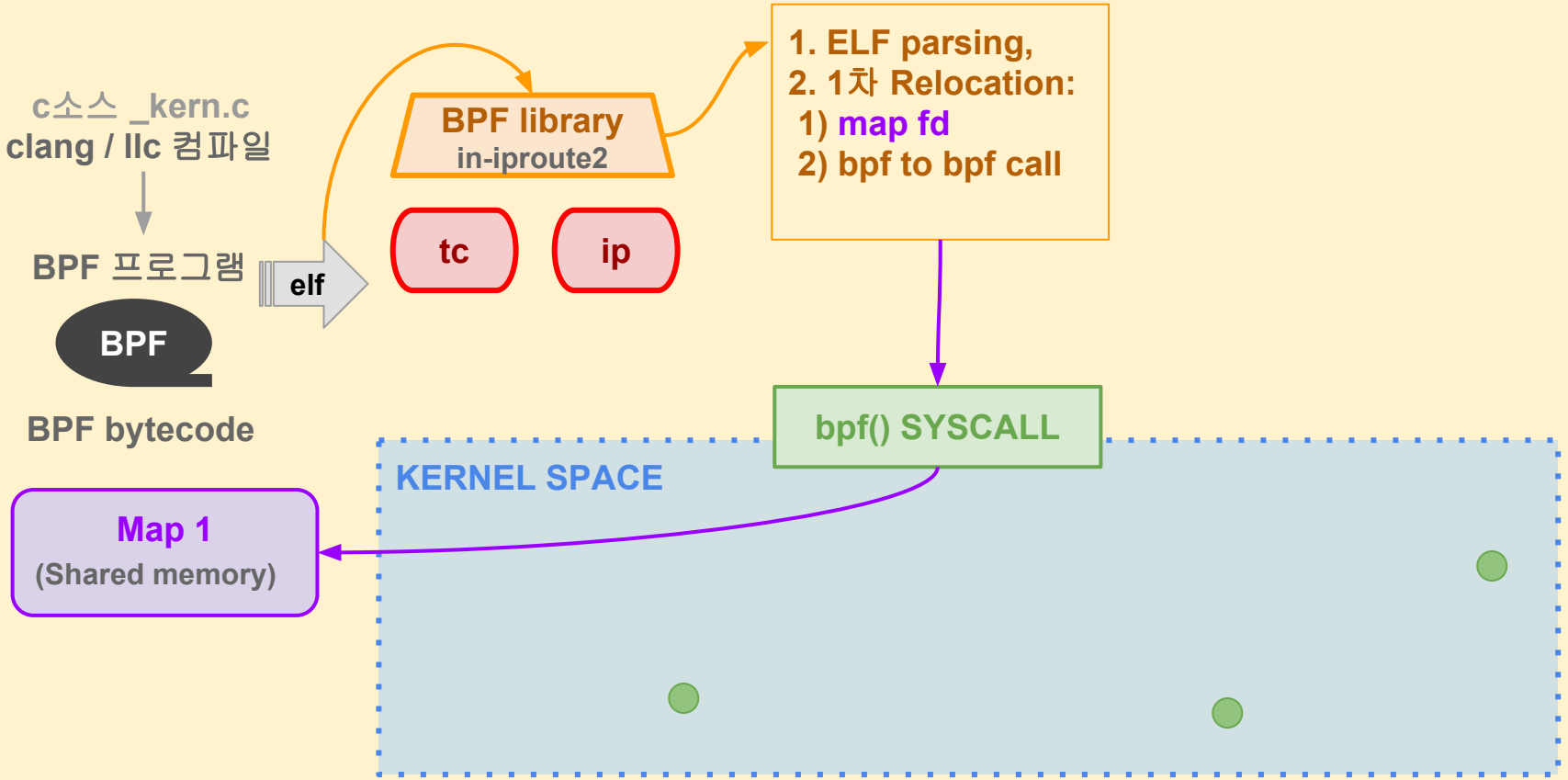

특정 커널 함수 안에
if (has_bpf_prog)
BPF_PROG_RUN();
->bpf_func(ctx, insni);
Hook points: callback points
. . .
. . .
tc: L3 DD 직전 / 직후 지점
kprobe: 함수 Entry / Return
XDP: L2 device driver 지점
KERNEL SPACE
BPF prog injection !!
BPF
BPF
BPF

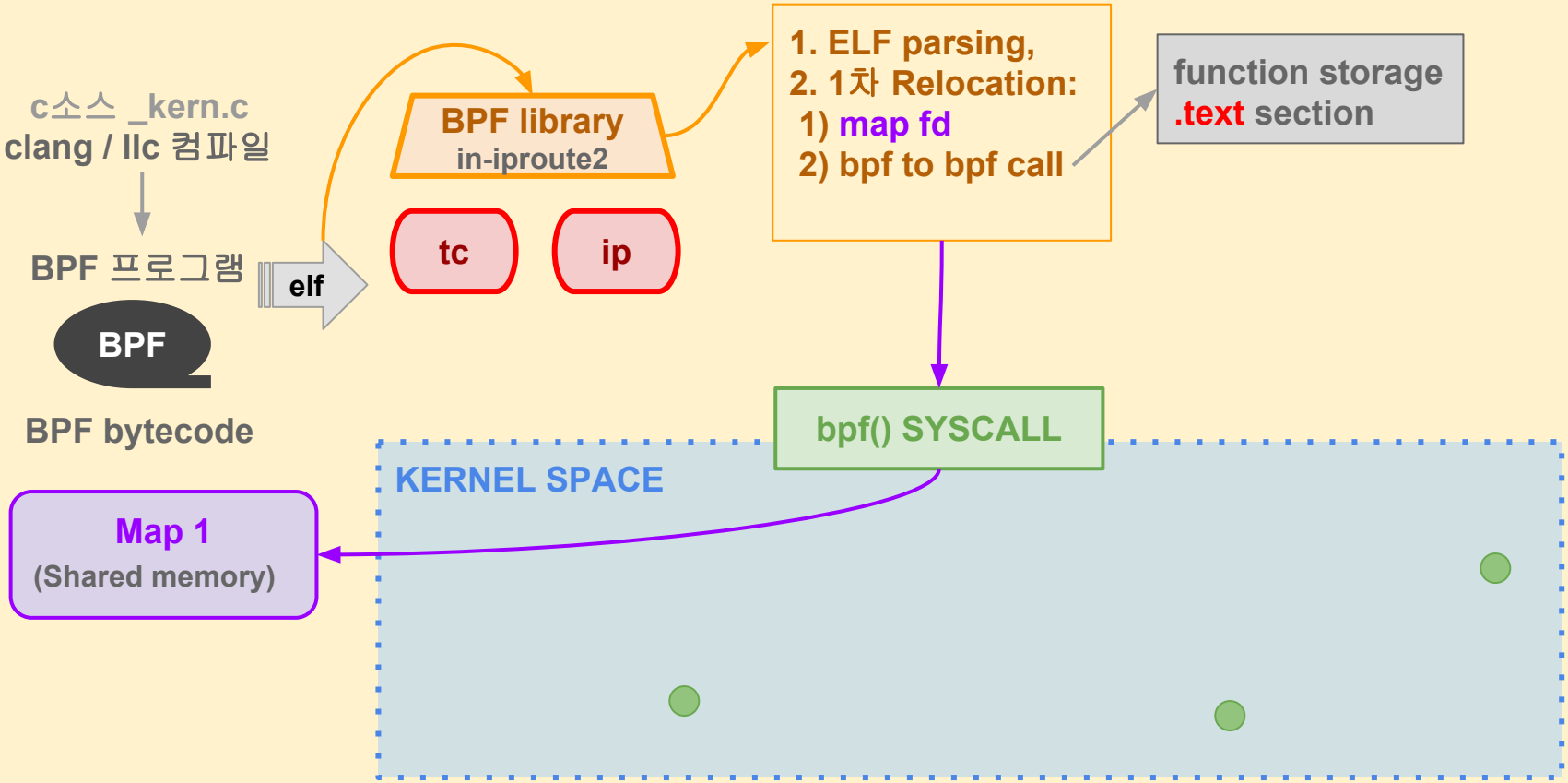

특정 커널 함수 안에
if (has_bpf_prog)
BPF_PROG_RUN();
->bpf_func(ctx, insni);
BPF Interpreter
또는
JIT 된 머신코드
Hook points: callback points
. . .
. . .
tc: L3 DD 직전 / 직후 지점
kprobe: 함수 Entry / Return
XDP: L2 device driver 지점
KERNEL SPACE
BPF prog injection !!
BPF
BPF
BPF

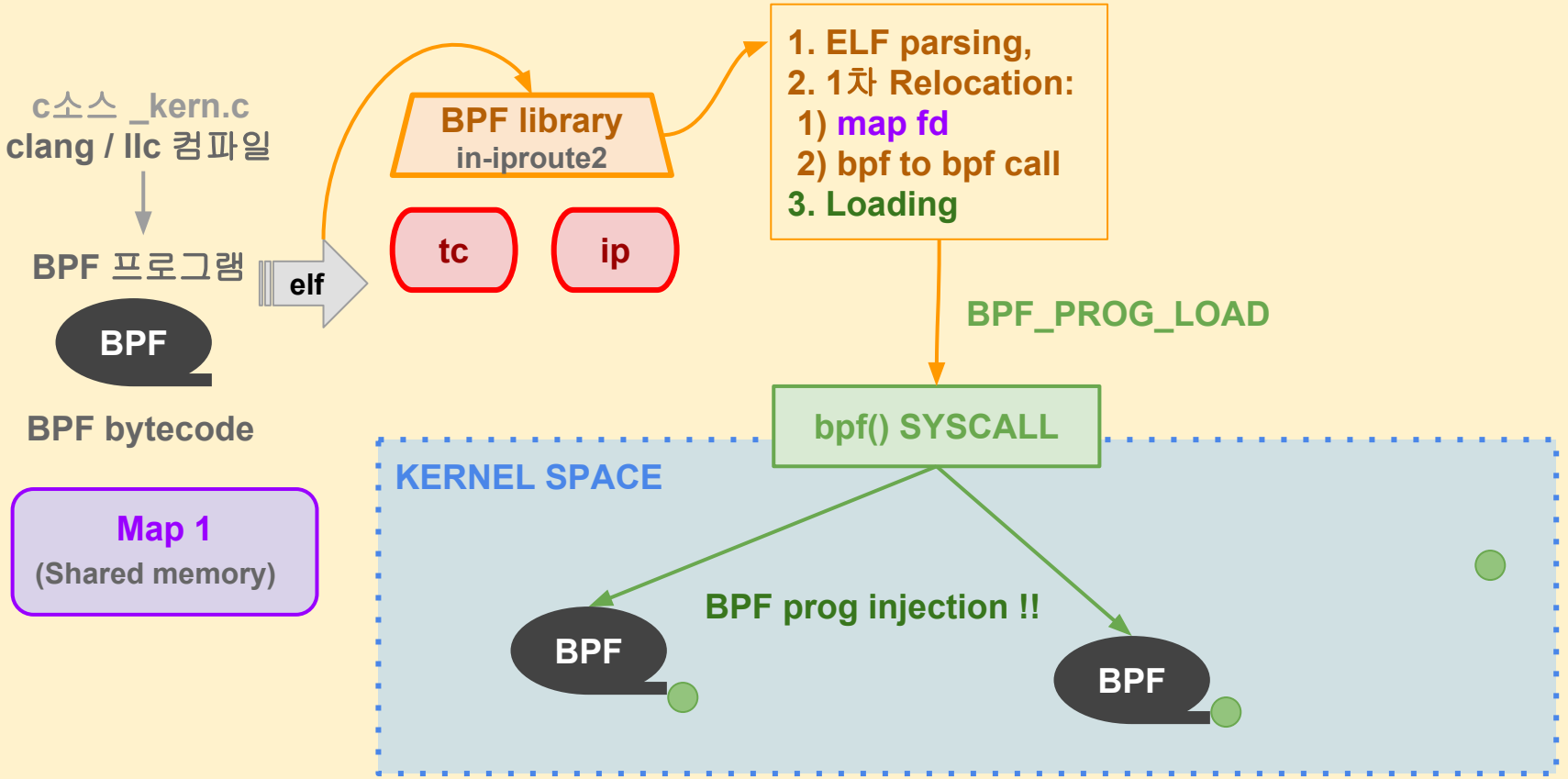

Hook points: callback points
. . .
. . .
tc: L3 DD 직전 / 직후 지점
kprobe: 함수 Entry / Return
XDP: L2 device driver 지점
KERNEL SPACE
BPF
BPF prog injection !!
HOW ?
BPF
BPF

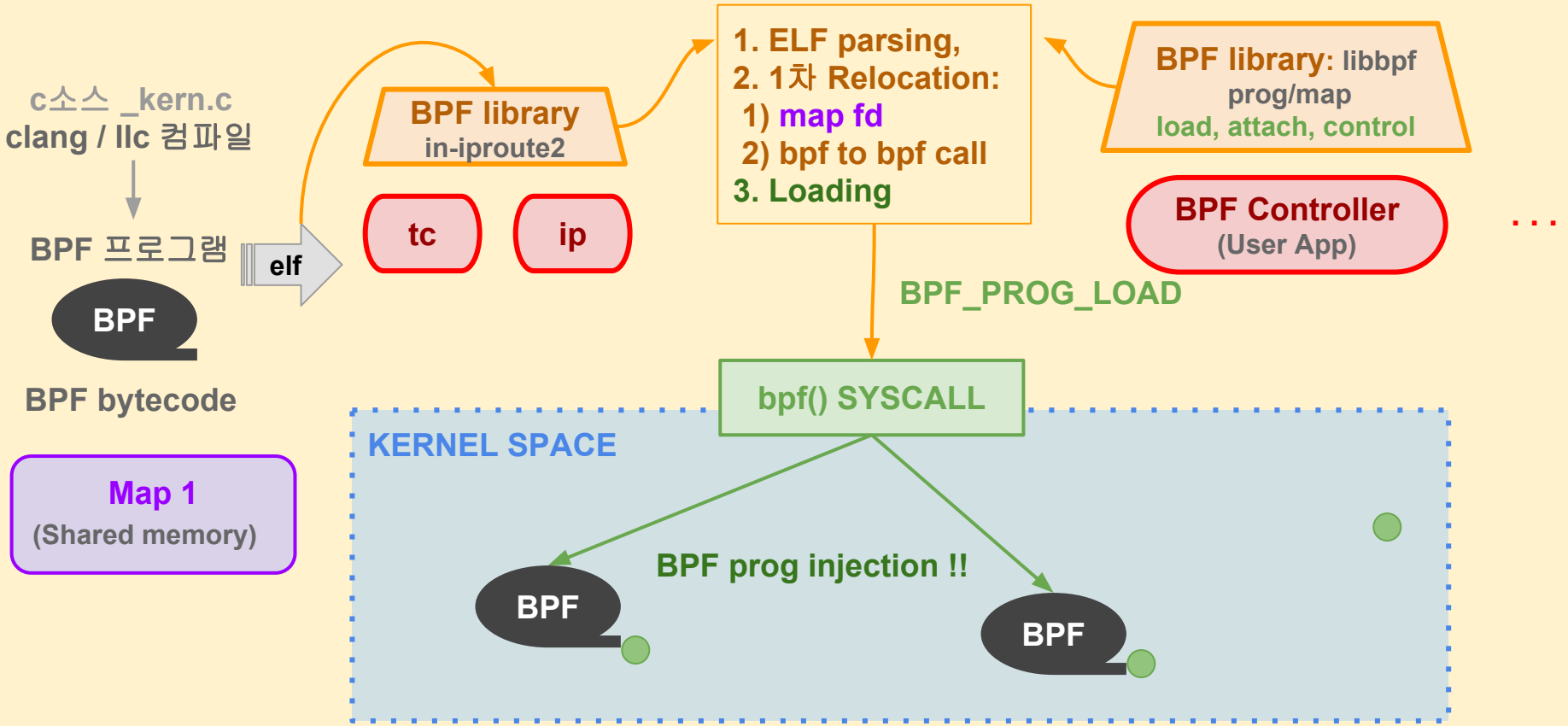

c소스 _kern.c
clang / llc 컴파일
BPF 프로그램
elf
BPF
BPF bytecode
BPF library
in-iproute2
tc
ip
bpf() SYSCALL
KERNEL SPACE

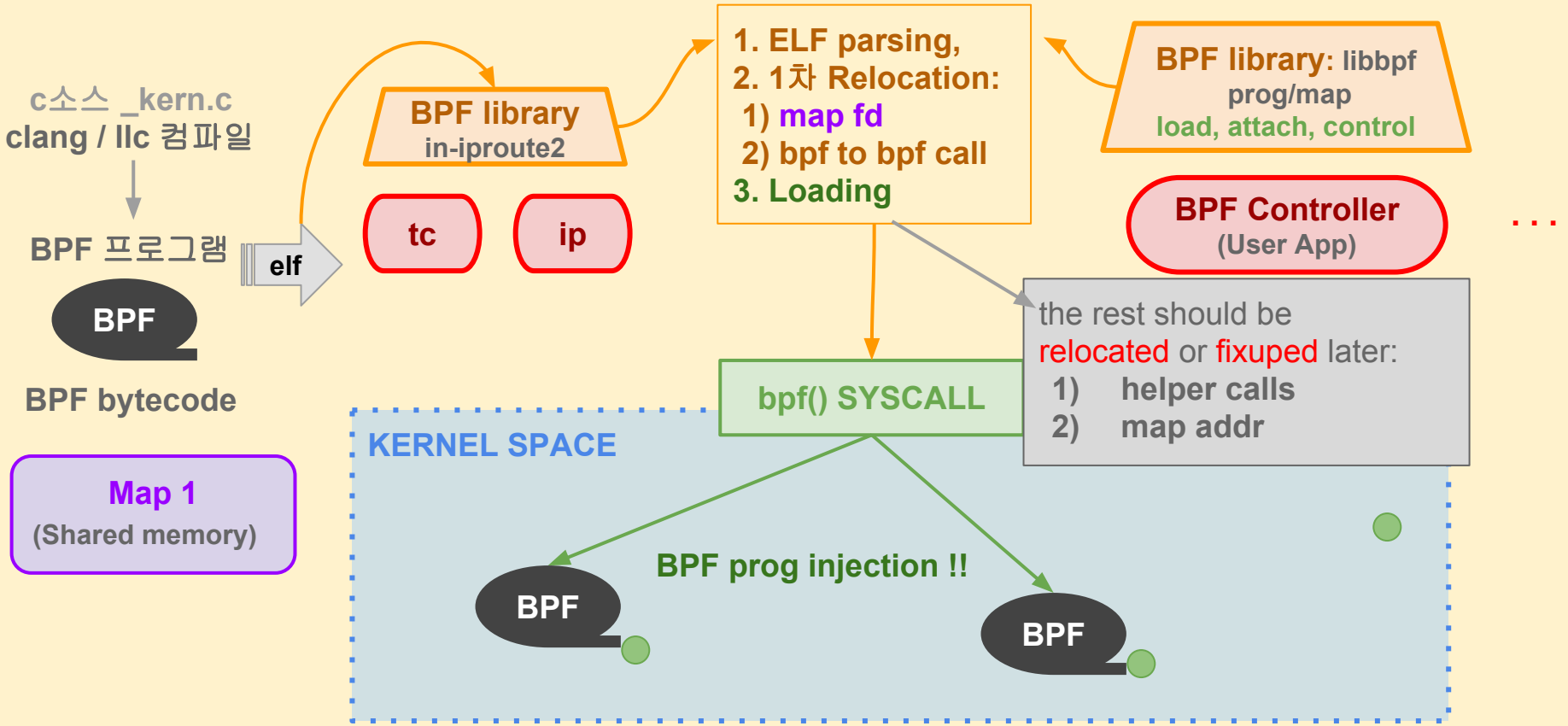

c소스 _kern.c
clang / llc 컴파일
BPF 프로그램
BPF
BPF bytecode
elf
BPF library
in-iproute2
tc
ip
1. ELF parsing,
2. 1차 Relocation:
1) map fd
2) bpf to bpf call
bpf() SYSCALL
KERNEL SPACE

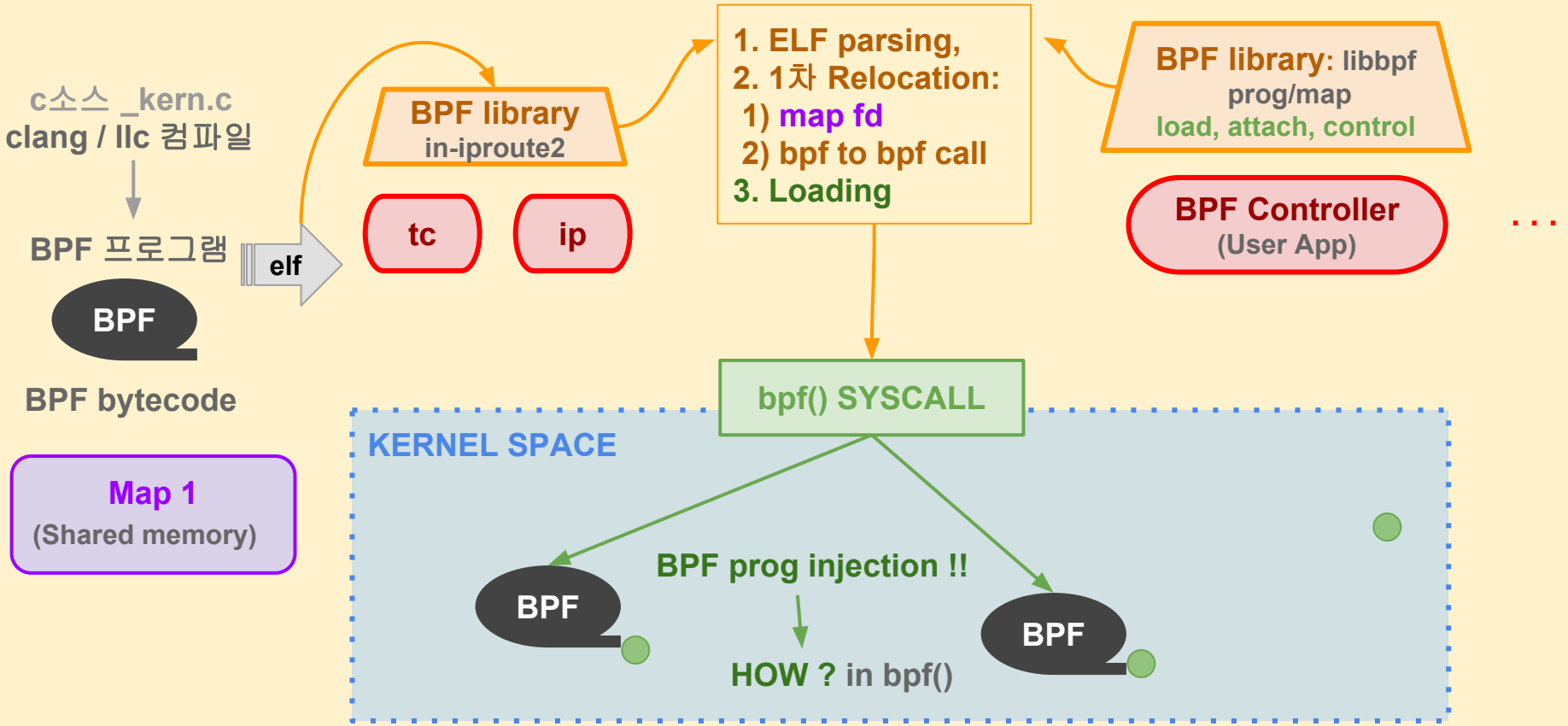

c소스 _kern.c
clang / llc 컴파일
BPF 프로그램
BPF
BPF bytecode
elf
BPF library
in-iproute2
tc
ip
1. ELF parsing,
2. 1차 Relocation:
1) map fd
2) bpf to bpf call
bpf() SYSCALL
KERNEL SPACE
Map 1
(Shared memory)

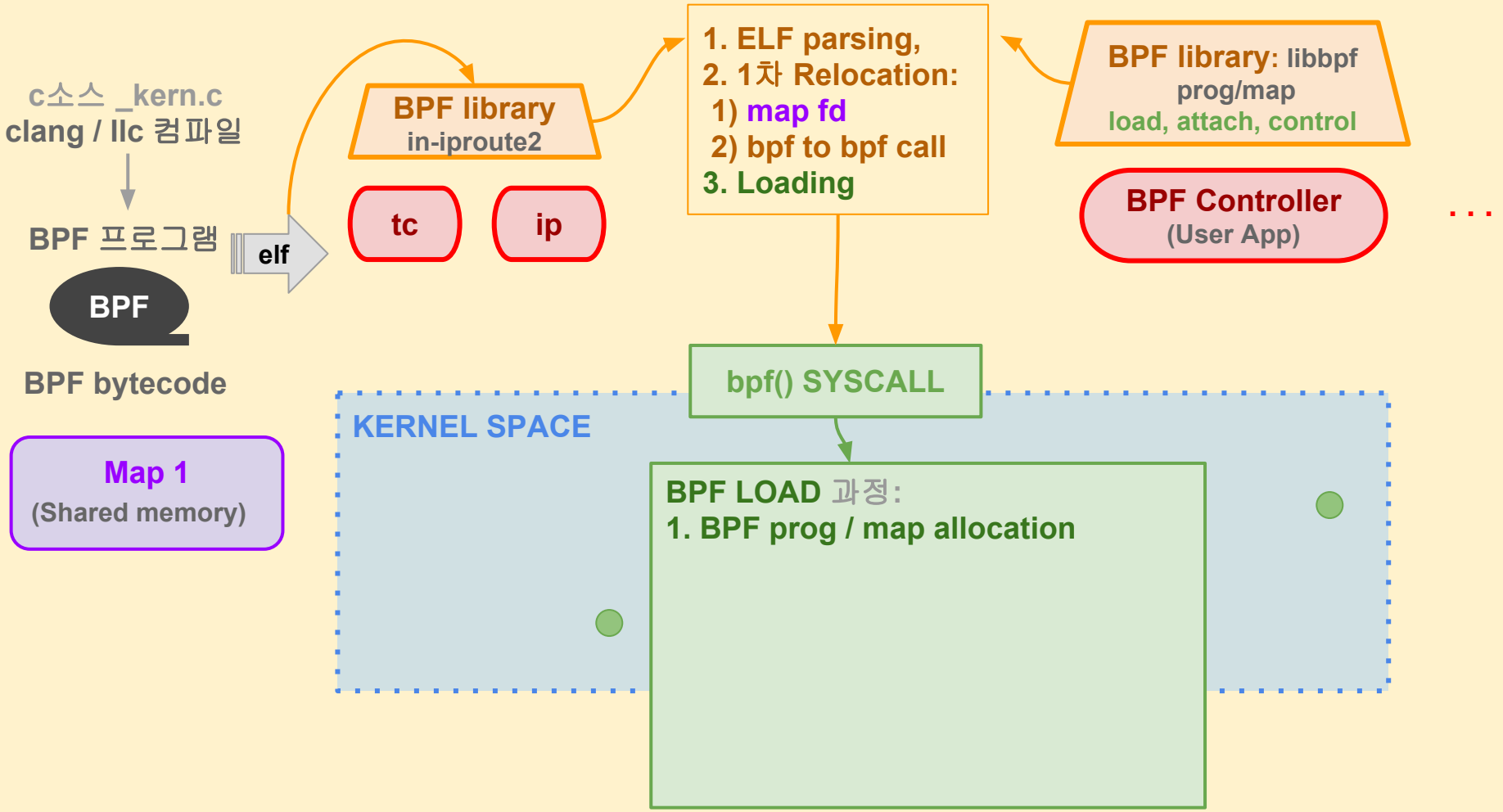

c소스 _kern.c
clang / llc 컴파일
BPF 프로그램
BPF
BPF bytecode
elf
BPF library
in-iproute2
tc
ip
1. ELF parsing,
2. 1차 Relocation:
1) map fd
2) bpf to bpf call
function storage
.text section
bpf() SYSCALL
KERNEL SPACE
Map 1
(Shared memory)

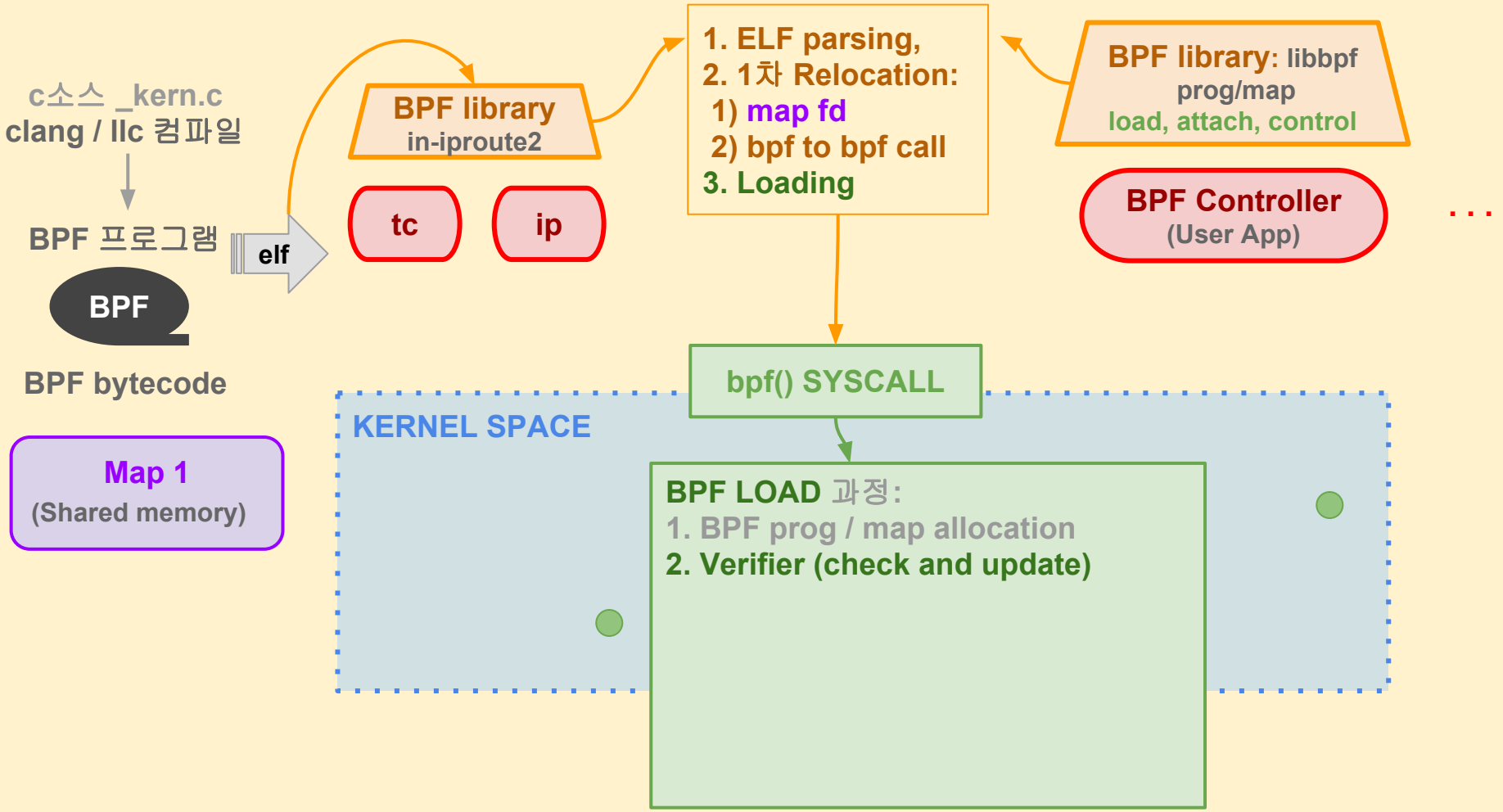

c소스 _kern.c
clang / llc 컴파일
BPF 프로그램
BPF
BPF bytecode
elf
BPF library
in-iproute2
tc
ip
1. ELF parsing,
2. 1차 Relocation:
1) map fd
2) bpf to bpf call
3. Loading
BPF_PROG_LOAD
bpf() SYSCALL
Map 1
(Shared memory)
KERNEL SPACE
BPF prog injection !!
BPF
BPF

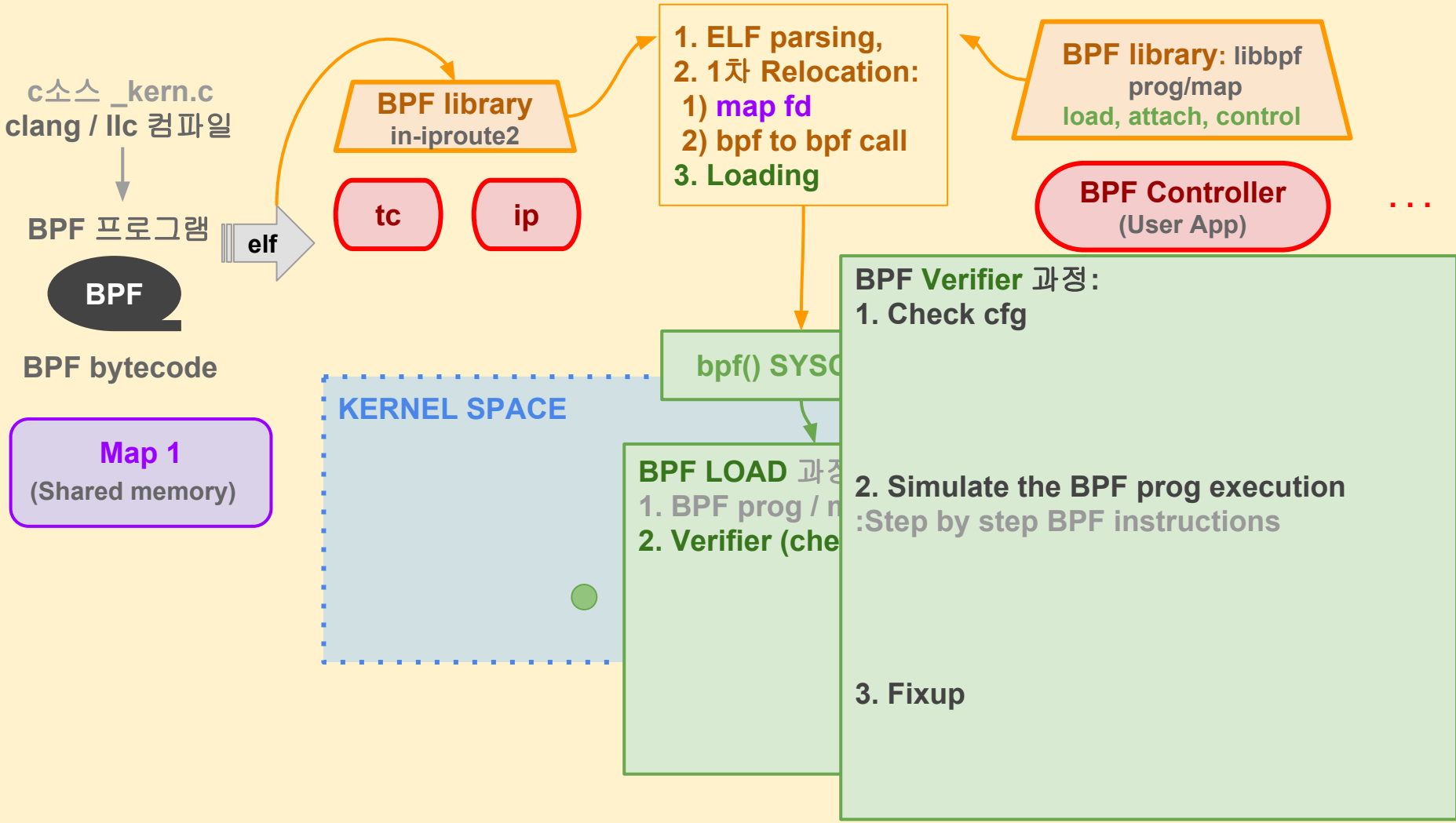

c소스 _kern.c
clang / llc 컴파일
BPF 프로그램
BPF
BPF bytecode
elf
BPF library
in-iproute2
tc
ip
1. ELF parsing,
2. 1차 Relocation:
1) map fd
2) bpf to bpf call
3. Loading
BPF library: libbpf
prog/map
load, attach, control
BPF Controller
(User App)
. . .
BPF_PROG_LOAD
bpf() SYSCALL
KERNEL SPACE
BPF prog injection !!
BPF
BPF
Map 1
(Shared memory)

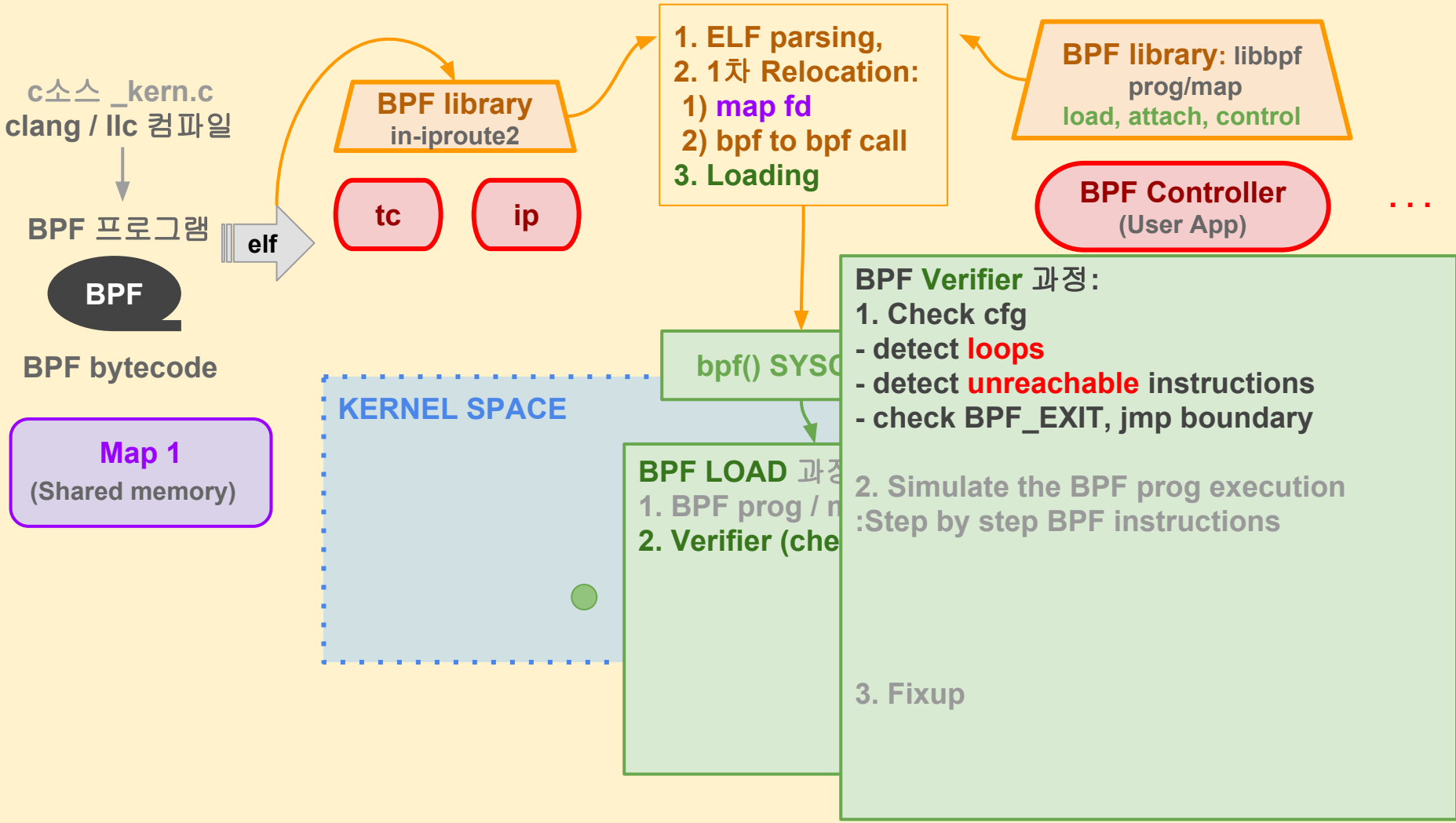

c소스 _kern.c
clang / llc 컴파일
BPF 프로그램
BPF
BPF bytecode
elf
BPF library
in-iproute2
tc
ip
1. ELF parsing,
2. 1차 Relocation:
1) map fd
2) bpf to bpf call
3. Loading
BPF library: libbpf
prog/map
load, attach, control
BPF Controller
(User App)
. . .
the rest should be
relocated or fixuped later:
1) helper calls
2) map addr
bpf() SYSCALL
KERNEL SPACE
BPF prog injection !!
BPF
BPF
Map 1
(Shared memory)

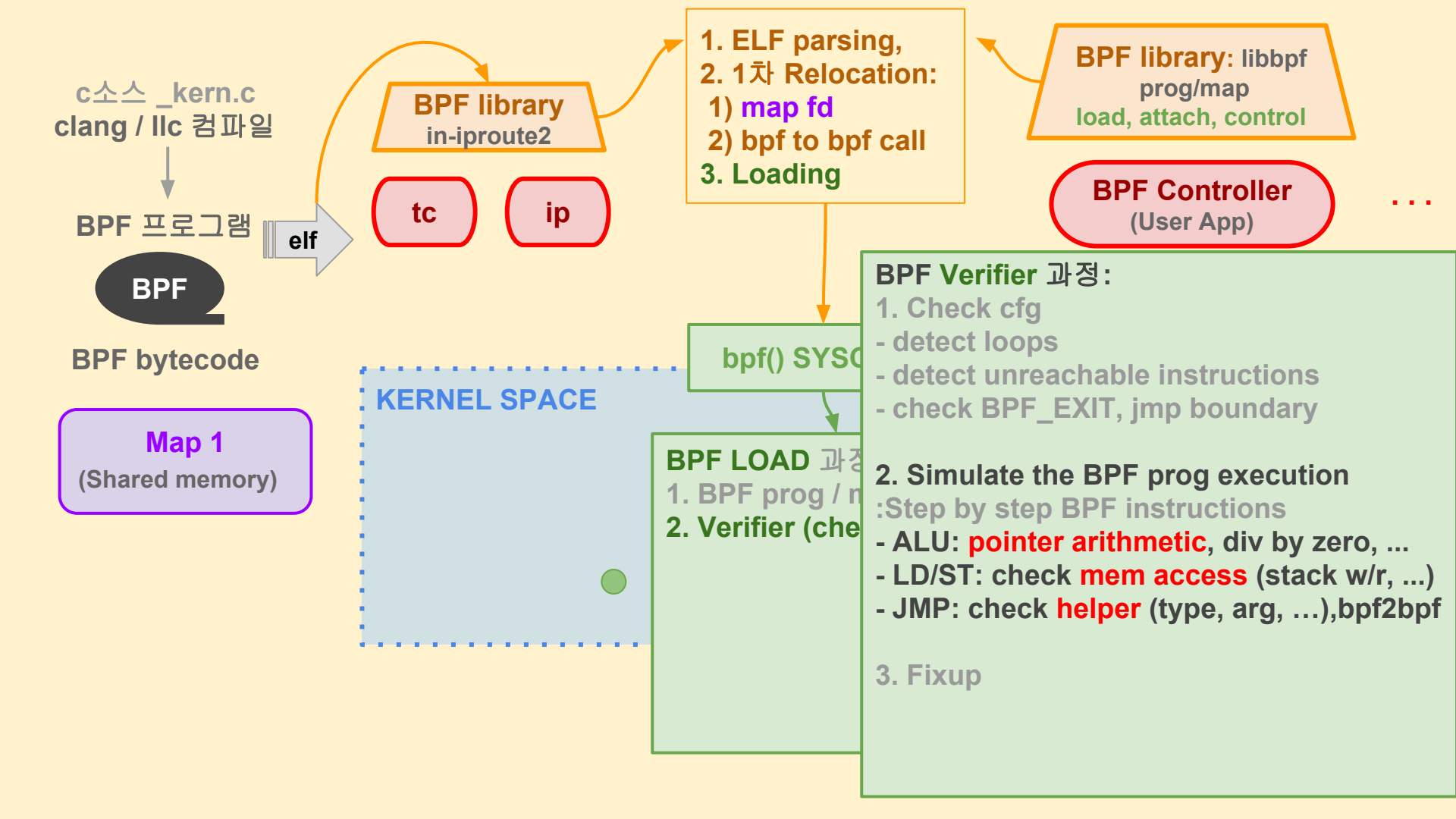

c소스 _kern.c
clang / llc 컴파일
BPF 프로그램
BPF
BPF bytecode
elf
BPF library
in-iproute2
tc
ip
1. ELF parsing,
2. 1차 Relocation:
1) map fd
2) bpf to bpf call
3. Loading
BPF library: libbpf
prog/map
load, attach, control
BPF Controller
(User App)
. . .
Map 1
(Shared memory)
bpf() SYSCALL
KERNEL SPACE
BPF prog injection !!
HOW ? in bpf()
BPF
BPF

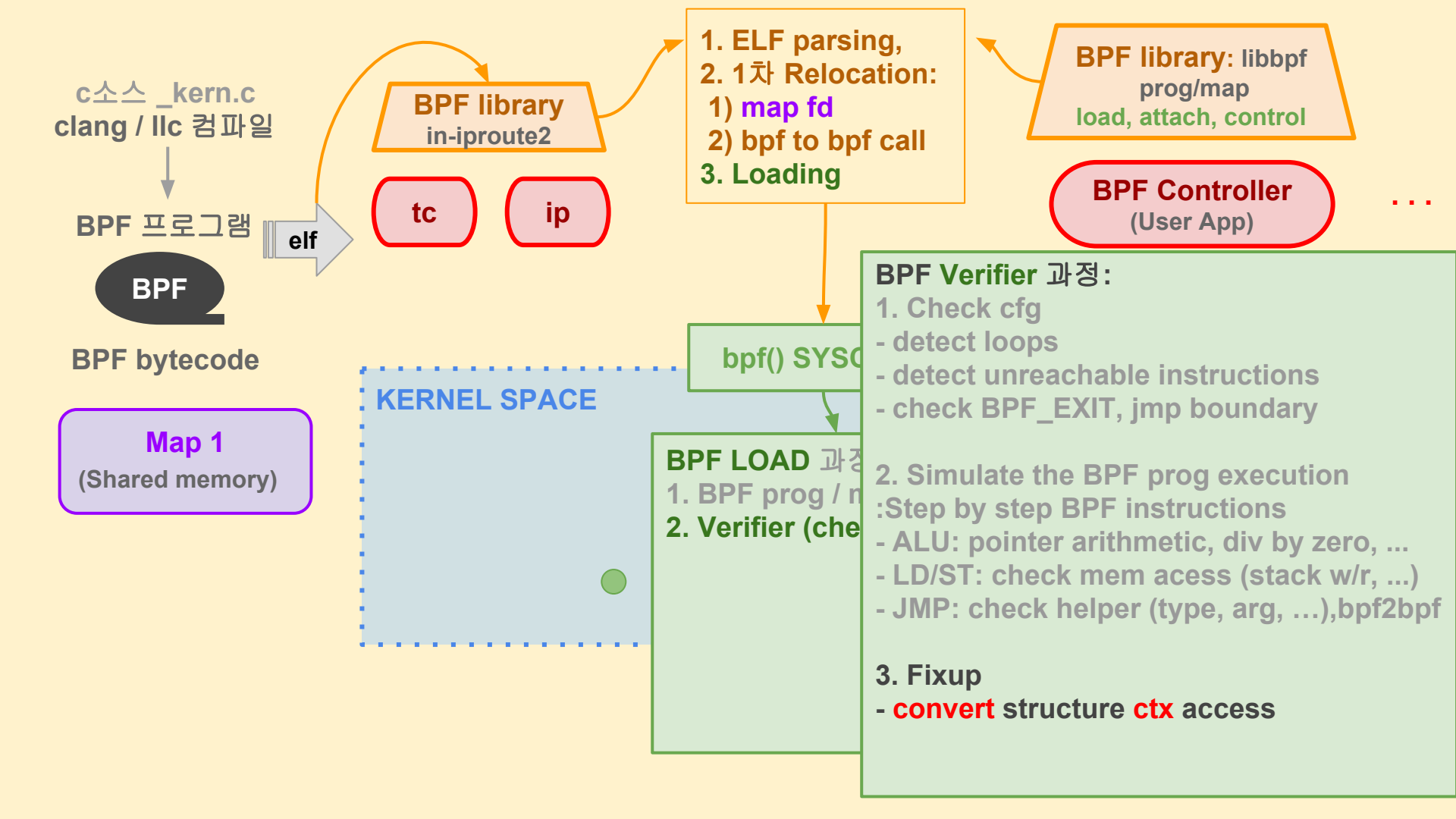

c소스 _kern.c
clang / llc 컴파일
BPF 프로그램
BPF
BPF bytecode
elf
BPF library
in-iproute2
tc
ip
1. ELF parsing,
2. 1차 Relocation:
1) map fd
2) bpf to bpf call
3. Loading
BPF library: libbpf
prog/map
load, attach, control
BPF Controller
(User App)
. . .
bpf() SYSCALL
KERNEL SPACE
BPF LOAD 과정:
1. BPF prog / map allocation
Map 1
(Shared memory)

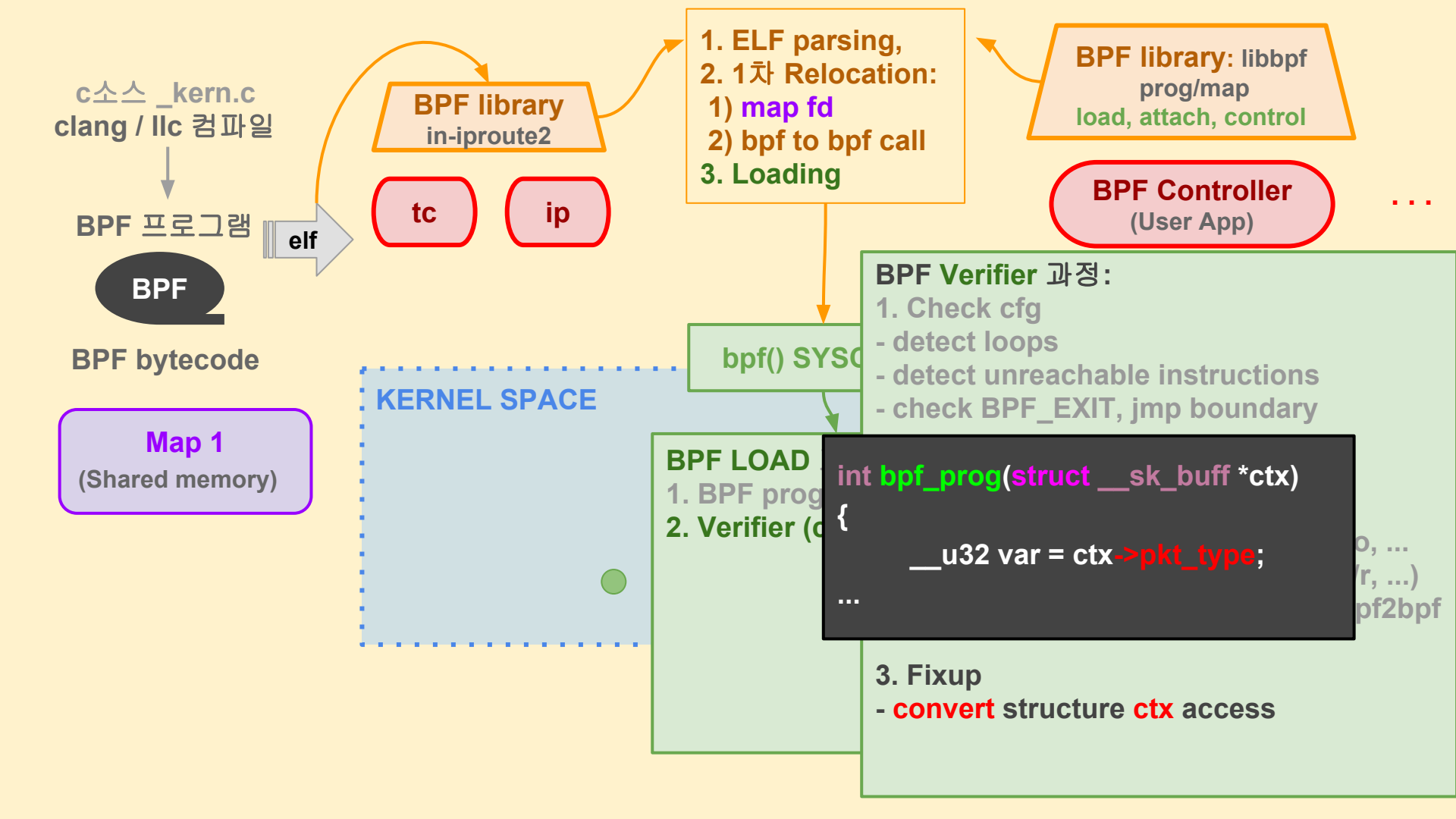

c소스 _kern.c
clang / llc 컴파일
BPF 프로그램
BPF
BPF bytecode
elf
BPF library
in-iproute2
tc
ip
1. ELF parsing,
2. 1차 Relocation:
1) map fd
2) bpf to bpf call
3. Loading
BPF library: libbpf
prog/map
load, attach, control
BPF Controller
(User App)
. . .
bpf() SYSCALL
KERNEL SPACE
BPF LOAD 과정:
1. BPF prog / map allocation
2. Verifier (check and update)
Map 1
(Shared memory)

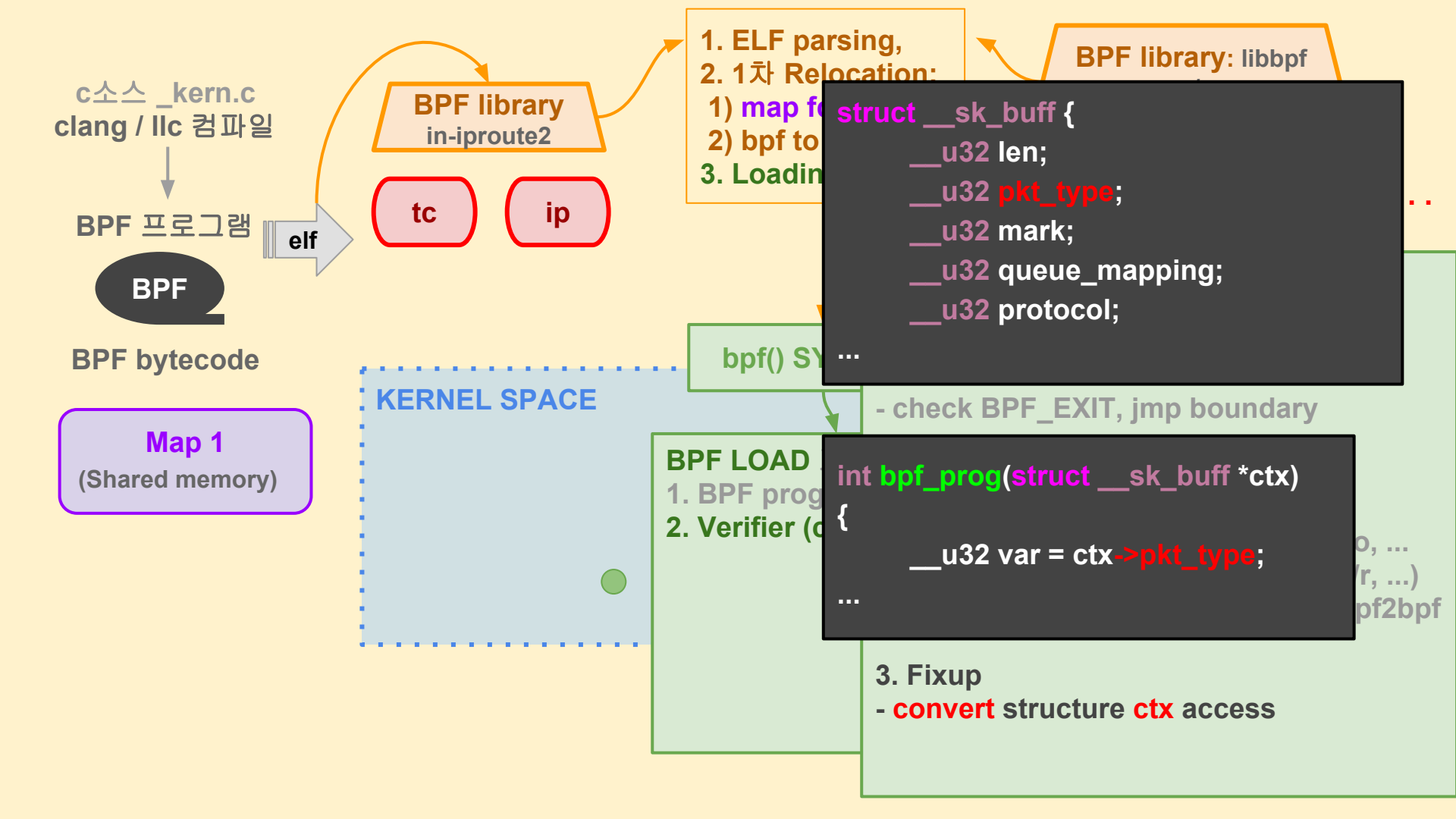

c소스 _kern.c
clang / llc 컴파일
BPF 프로그램
BPF
BPF bytecode
elf
BPF library
in-iproute2
tc
ip
1. ELF parsing,
2. 1차 Relocation:
1) map fd
2) bpf to bpf call
3. Loading
BPF library: libbpf
prog/map
load, attach, control
BPF Controller
(User App)
. . .
Map 1
(Shared memory)
KERNEL SPACE
bpf() SYSC
BPF LOAD 과정
1. BPF prog / m
2. Verifier (che
BPF Verifier 과정:
1. Check cfg
2. Simulate the BPF prog execution
:Step by step BPF instructions
3. Fixup


c소스 _kern.c
clang / llc 컴파일
BPF 프로그램
BPF
BPF bytecode
elf
BPF library
in-iproute2
tc
ip
1. ELF parsing,
2. 1차 Relocation:
1) map fd
2) bpf to bpf call
3. Loading
BPF library: libbpf
prog/map
load, attach, control
BPF Controller
(User App)
. . .
KERNEL SPACE
bpf() SYSC
BPF LOAD 과정
1. BPF prog / m
2. Verifier (che
Map 1
(Shared memory)
BPF Verifier 과정:
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


c소스 _kern.c
clang / llc 컴파일
BPF 프로그램
BPF
BPF bytecode
elf
BPF library
in-iproute2
tc
ip
1. ELF parsing,
2. 1차 Relocation:
1) map fd
2) bpf to bpf call
3. Loading
BPF library: libbpf
prog/map
load, attach, control
BPF Controller
(User App)
. . .
Map 1
(Shared memory)
KERNEL SPACE
bpf() SYSC
BPF LOAD 과정
1. BPF prog / m
2. Verifier (che
BPF Verifier 과정:
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


c소스 _kern.c
clang / llc 컴파일
BPF 프로그램
BPF
BPF bytecode
elf
BPF library
in-iproute2
tc
ip
1. ELF parsing,
2. 1차 Relocation:
1) map fd
2) bpf to bpf call
3. Loading
BPF library: libbpf
prog/map
load, attach, control
BPF Controller
(User App)
. . .
KERNEL SPACE
bpf() SYSC
BPF LOAD 과정
1. BPF prog / m
2. Verifier (che
Map 1
(Shared memory)
BPF Verifier 과정:
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


c소스 _kern.c
clang / llc 컴파일
BPF 프로그램
BPF
BPF bytecode
elf
BPF library
in-iproute2
tc
ip
1. ELF parsing,
2. 1차 Relocation:
1) map fd
2) bpf to bpf call
3. Loading
BPF library: libbpf
prog/map
load, attach, control
BPF Controller
(User App)
. . .
Map 1
(Shared memory)
KERNEL SPACE
bpf() SYSC
BPF LOAD
1. BPF prog
2. Verifier (c
BPF Verifier 과정:
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;
...
o, ...
r, ...)
pf2bpf
3. Fixup
- convert structure ctx access


c소스 _kern.c
clang / llc 컴파일
BPF 프로그램
BPF
BPF bytecode
elf
BPF library
in-iproute2
tc
ip
1. ELF parsing,
2. 1차 Relocation:
1) map f
2) bpf to
3. Loadin
BPF library: libbpf
struct __sk_buff {
__u32 len;
__u32 pkt_type;
__u32 mark;
__u32 queue_mapping;
__u32 protocol;
...
bpf() SY
KERNEL SPACE
- check BPF_EXIT, jmp boundary
BPF LOAD
1. BPF prog
2. Verifier (c
int bpf_prog(struct __sk_buff *ctx)
{
__u32 var = ctx->pkt_type;
...
o, ...
r, ...)
pf2bpf
3. Fixup
- convert structure ctx access
Map 1
(Shared memory)

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 inform
                                 * 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;
```

BPF library: libbpf

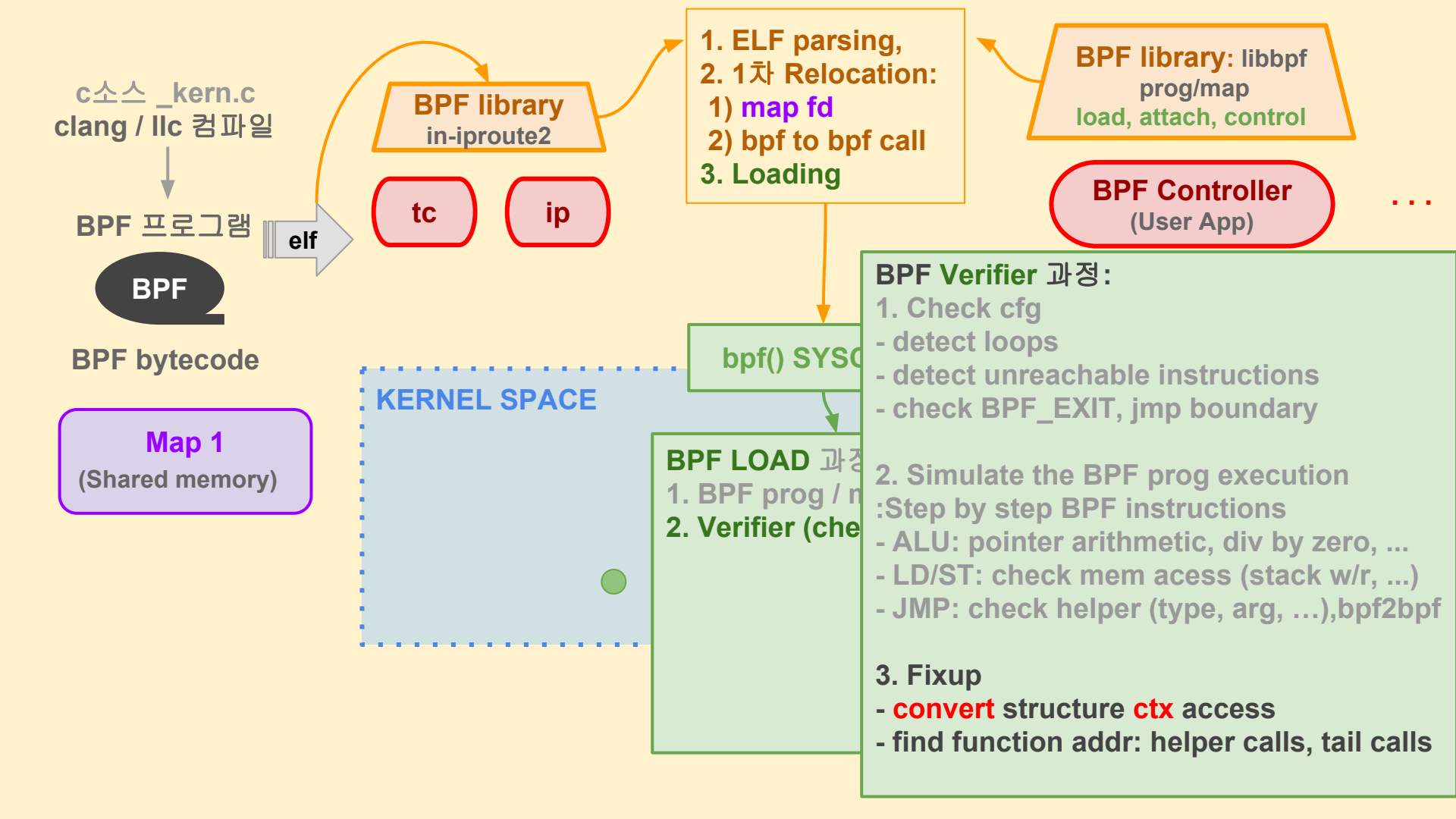
```
struct __sk_buff {
        __u32 len;
        __u32 pkt_type;
        __u32 mark;
        __u32 queue_mapping;
        __u32 protocol;
...
```

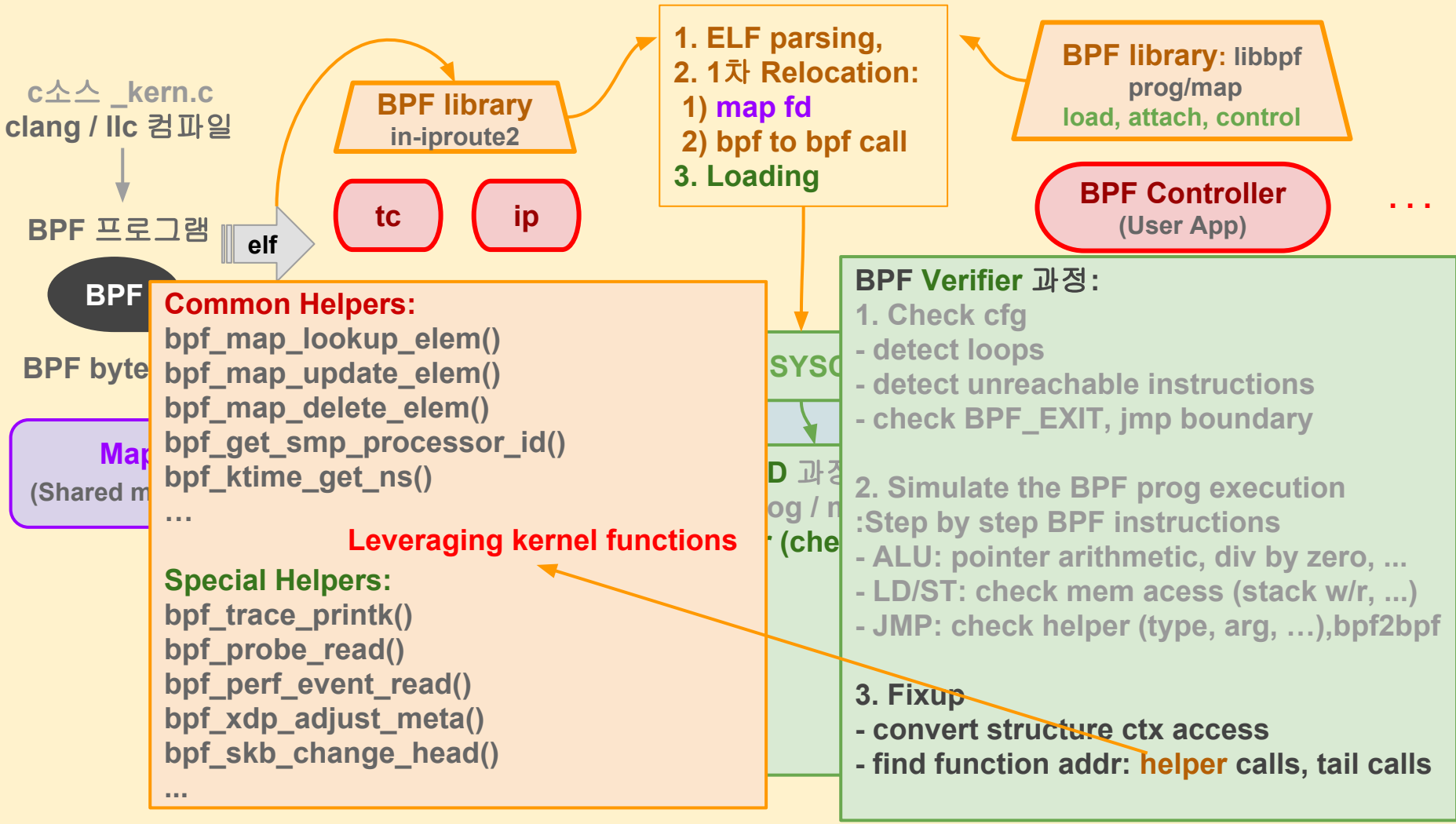
ck BPF_EXIT, jmp boundary

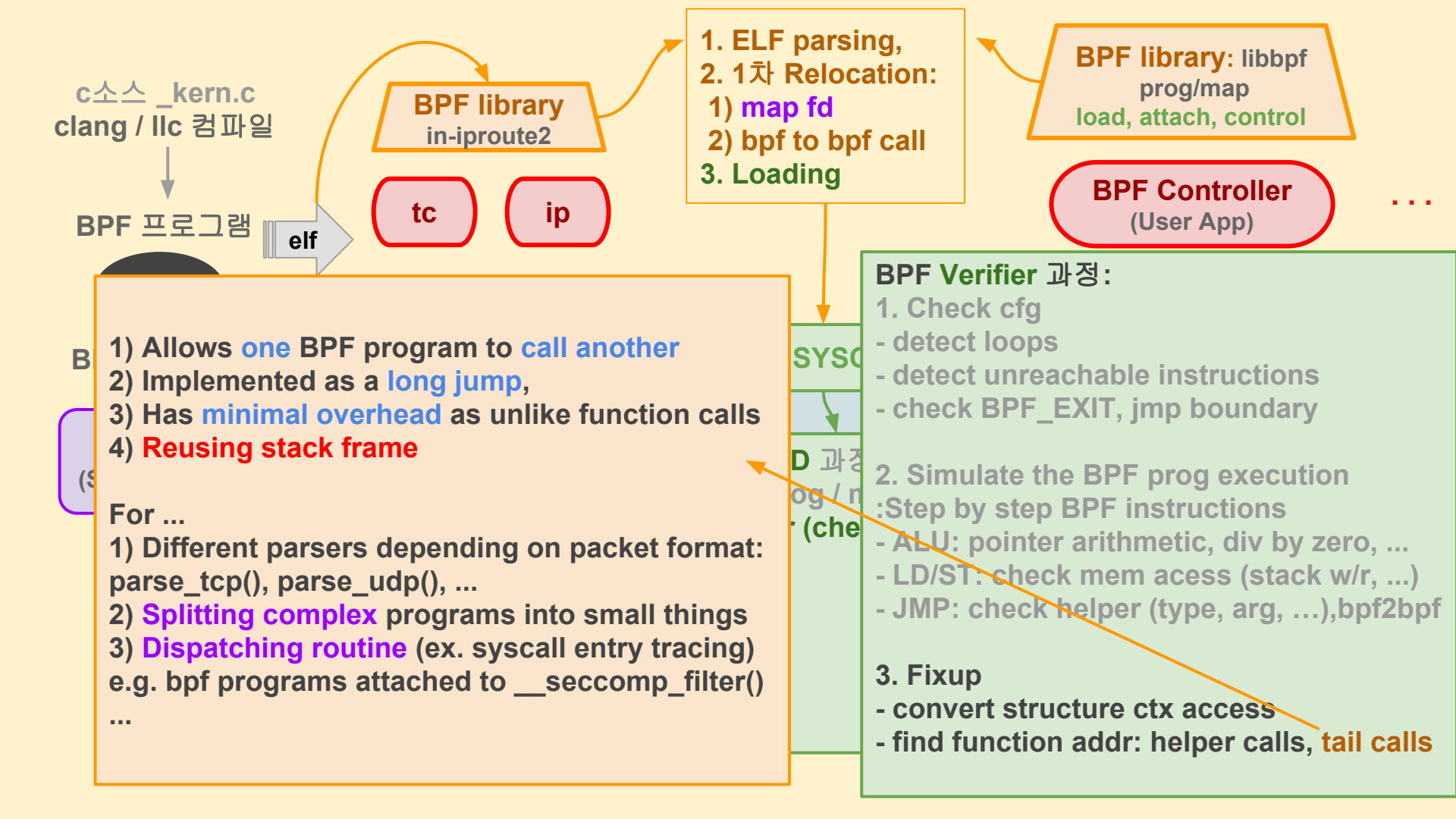
```
int bpf_prog(struct __sk_buff *ctx)
{
        __u32 var = ctx->pkt_type;
...
```

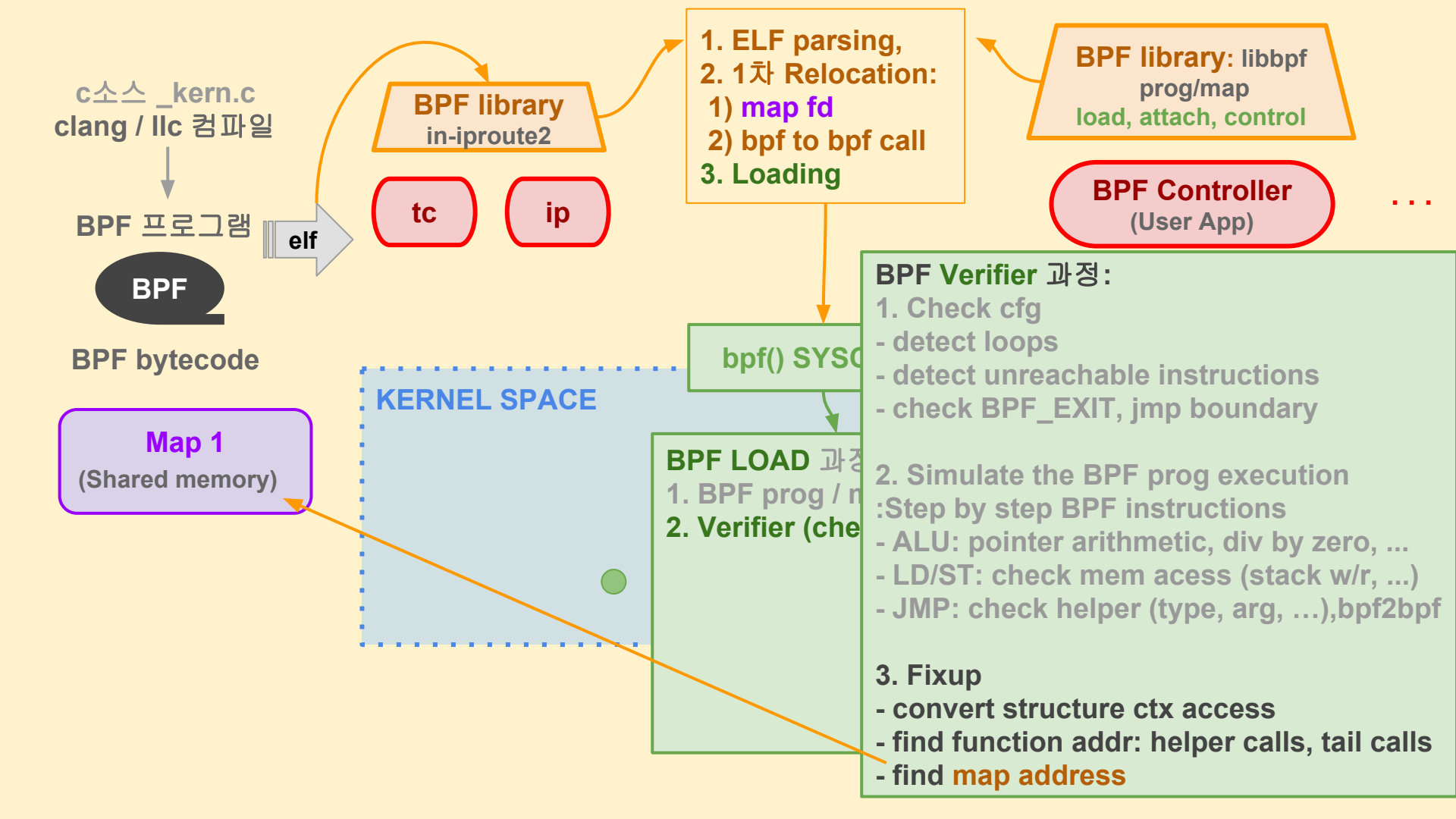
o, ...
r, ...)
pf2bpf

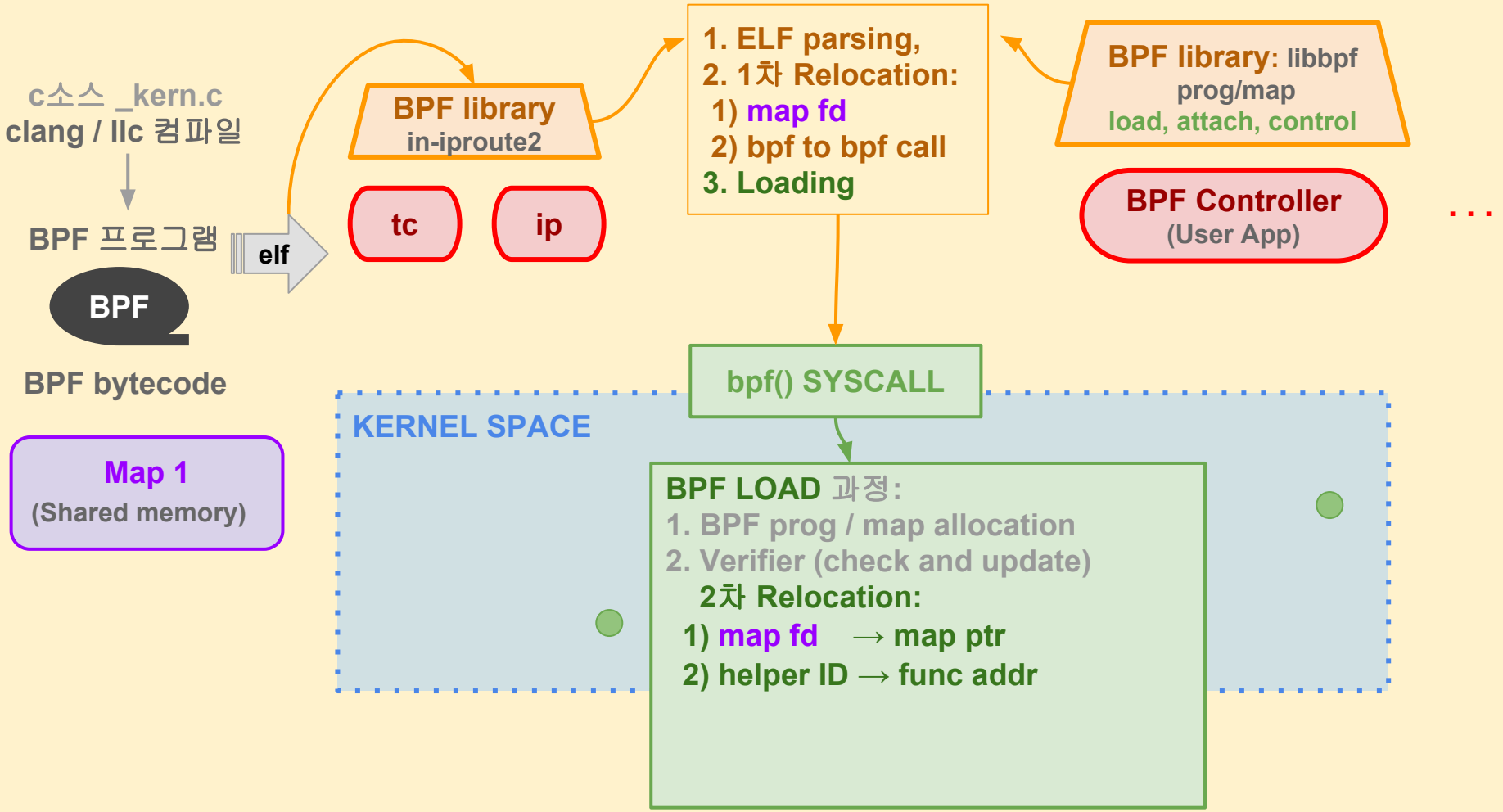
3. Fixup
- convert structure ctx access

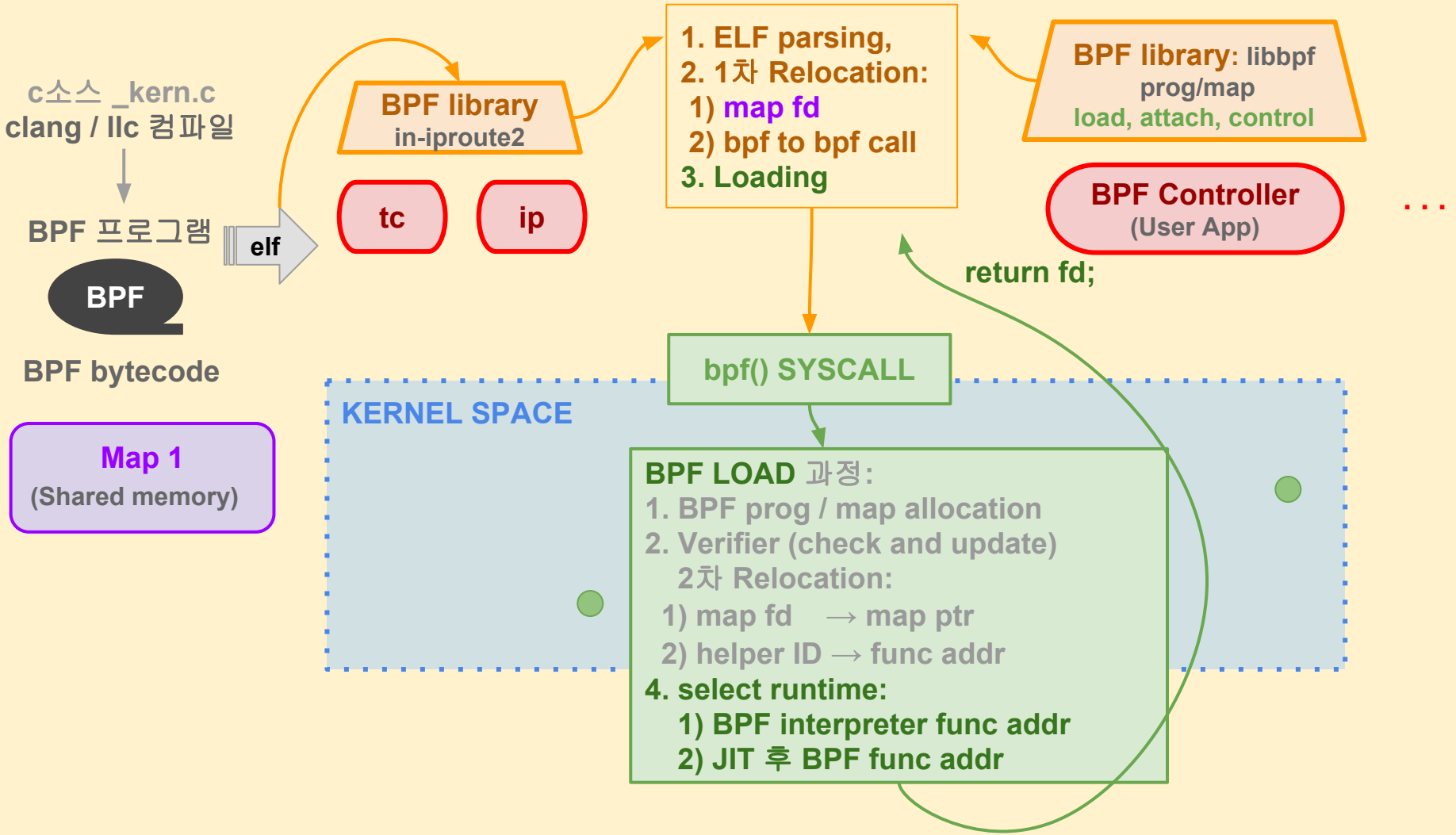

c소스 _kern.c
clang / llc 컴파일
BPF 프로그램
BPF
BPF bytecode
elf
BPF library
in-iproute2
tc
ip
1. ELF parsing,
2. 1차 Relocation:
1) map fd
2) bpf to bpf call
3. Loading
BPF library: libbpf
prog/map
load, attach, control
BPF Controller
(User App)
. . .
KERNEL SPACE
bpf() SYSC
BPF LOAD 과정
1. BPF prog / m
2. Verifier (che
Map 1
(Shared memory)
BPF Verifier 과정:
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

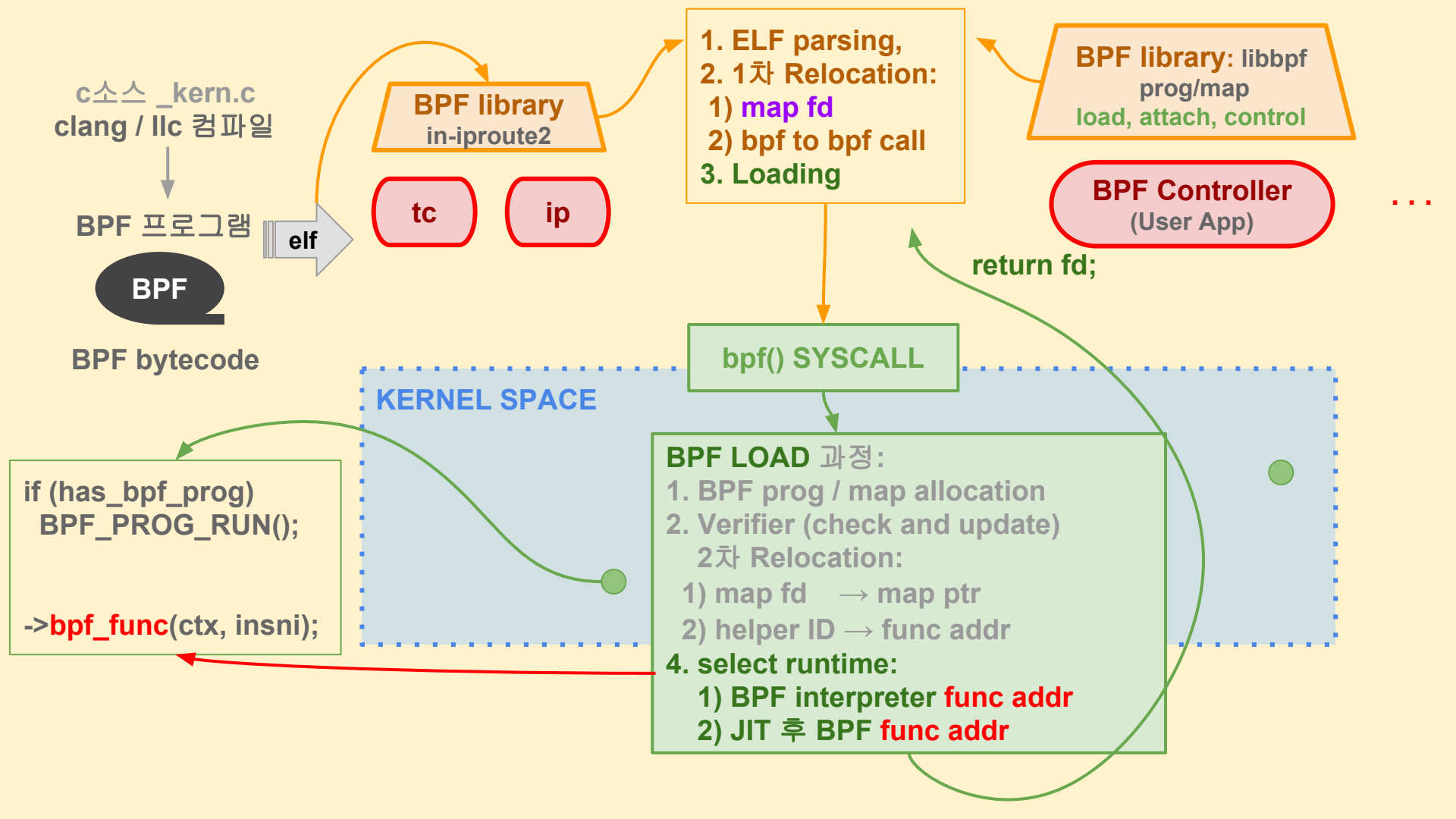

c소스 _kern.c
clang / llc 컴파일
BPF 프로그램
elf
BPF
BPF byte
Map
(Shared m
BPF library
in-iproute2
tc
ip
1. ELF parsing,
2. 1차 Relocation:
1) map fd
2) bpf to bpf call
3. Loading
BPF library: libbpf
prog/map
load, attach, control
BPF Controller
(User App)
. . .
Common Helpers:
bpf_map_lookup_elem()
bpf_map_update_elem()
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()
...
SYSC
D 과정
og / m
r (che
BPF Verifier 과정:
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

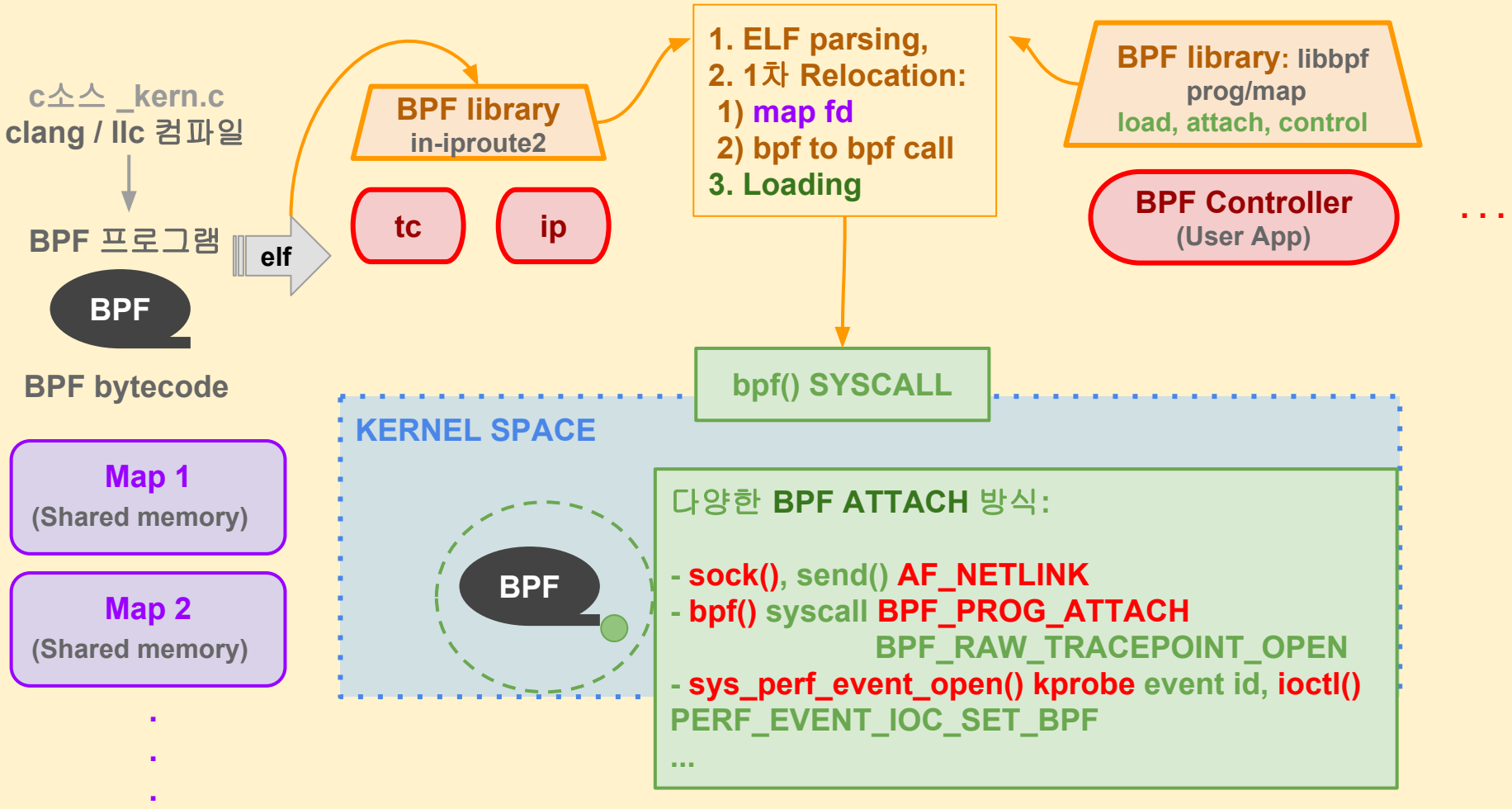

c소스 _kern.c
clang / llc 컴파일
BPF 프로그램
elf
BPF library
in-iproute2
tc
ip
1. ELF parsing,
2. 1차 Relocation:
1) map fd
2) bpf to bpf call
3. Loading
BPF library: libbpf
prog/map
load, attach, control
BPF Controller
(User App)
. . .
1) Allows one BPF program to call another
2) Implemented as a long jump,
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 과정:
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

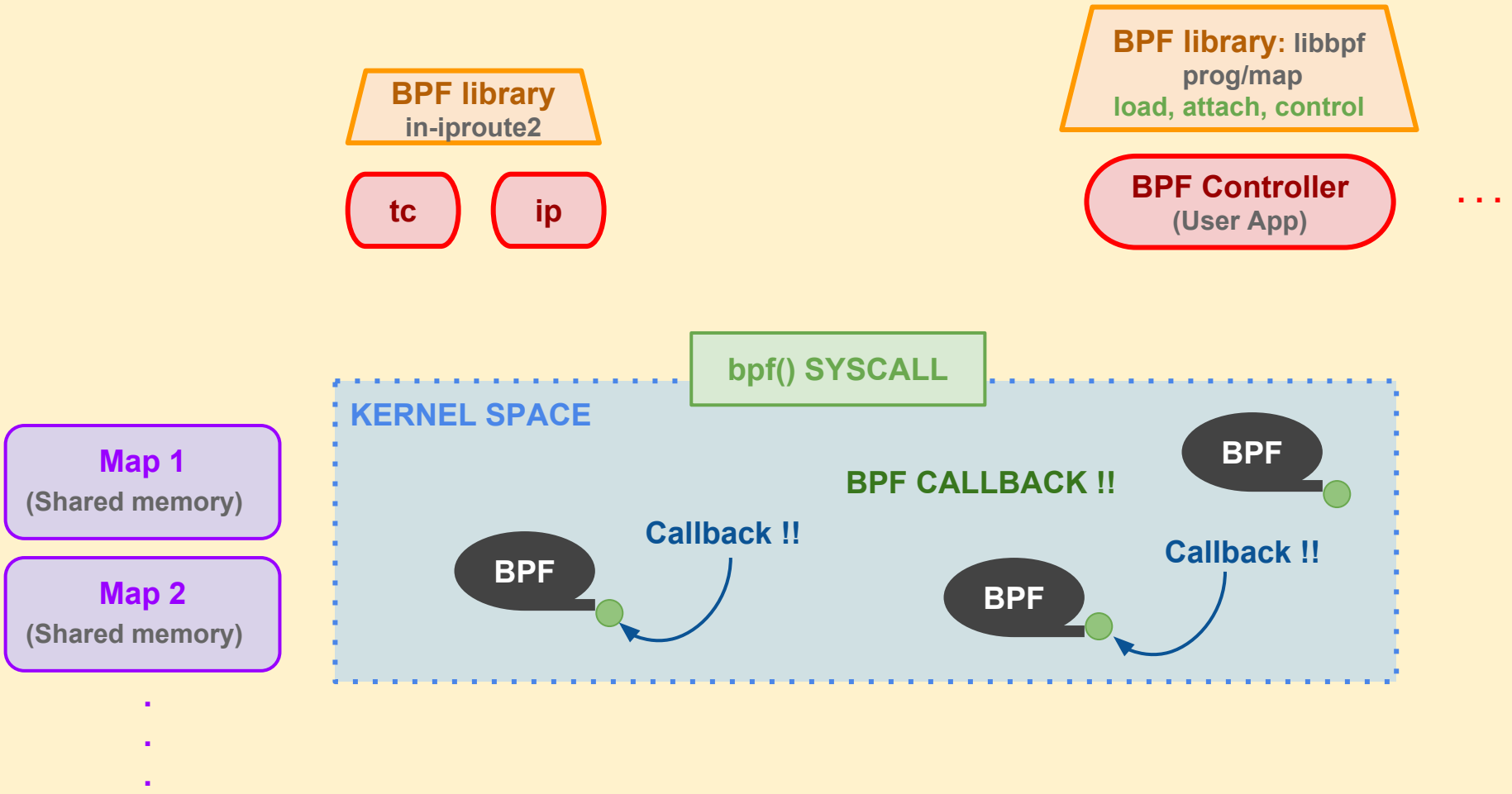

c소스 _kern.c
clang / llc 컴파일
BPF 프로그램
BPF
BPF bytecode
elf
BPF library
in-iproute2
tc
ip
1. ELF parsing,
2. 1차 Relocation:
1) map fd
2) bpf to bpf call
3. Loading
BPF library: libbpf
prog/map
load, attach, control
BPF Controller
(User App)
. . .
KERNEL SPACE
Map 1
(Shared memory)
bpf() SYSC
BPF LOAD 과정
1. BPF prog / m
2. Verifier (che
BPF Verifier 과정:
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


c소스 _kern.c
clang / llc 컴파일
BPF 프로그램
BPF
BPF bytecode
elf
BPF library
in-iproute2
tc
ip
1. ELF parsing,
2. 1차 Relocation:
1) map fd
2) bpf to bpf call
3. Loading
BPF library: libbpf
prog/map
load, attach, control
BPF Controller
(User App)
. . .
bpf() SYSCALL
KERNEL SPACE
Map 1
(Shared memory)
BPF LOAD 과정:
1. BPF prog / map allocation
2. Verifier (check and update)
2차 Relocation:
1) map fd → map ptr
2) helper ID → func addr


c소스 _kern.c
clang / llc 컴파일
BPF 프로그램
BPF
BPF bytecode
elf
BPF library
in-iproute2
tc
ip
1. ELF parsing,
2. 1차 Relocation:
1) map fd
2) bpf to bpf call
3. Loading
BPF library: libbpf
prog/map
load, attach, control
BPF Controller
(User App)
return fd;
bpf() SYSCALL
KERNEL SPACE
BPF LOAD 과정:
1. BPF prog / map allocation
2. Verifier (check and update)
2차 Relocation:
1) map fd → map ptr
2) helper ID → func addr
4. select runtime:
1) BPF interpreter func addr
2) JIT 후 BPF func addr
Map 1
(Shared memory)


c소스 _kern.c
clang / llc 컴파일
BPF 프로그램
BPF
BPF bytecode
elf
BPF library
in-iproute2
tc
ip
1. ELF parsing,
2. 1차 Relocation:
1) map fd
2) bpf to bpf call
3. Loading
BPF library: libbpf
prog/map
load, attach, control
BPF Controller
(User App)
. . .
return fd;
bpf() SYSCALL
KERNEL SPACE
BPF LOAD 과정:
1. BPF prog / map allocation
2. Verifier (check and update)
2차 Relocation:
1) map fd → map ptr
2) helper ID → func addr
4. select runtime:
1) BPF interpreter func addr
2) JIT 후 BPF func addr
if (has_bpf_prog)
BPF_PROG_RUN();
->bpf_func(ctx, insni);


c소스 _kern.c
clang / llc 컴파일
BPF 프로그램
BPF
BPF bytecode
elf
BPF library
in-iproute2
tc
ip
1. ELF parsing,
2. 1차 Relocation:
1) map fd
2) bpf to bpf call
3. Loading
BPF library: libbpf
prog/map
load, attach, control
BPF Controller
(User App)
. . .
bpf() SYSCALL
KERNEL SPACE
Map 1
(Shared memory)
Map 2
(Shared memory)
BPF
다양한 BPF ATTACH 방식:
- 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
...


BPF library: libbpf
prog/map
load, attach, control
BPF library
in-iproute2
tc
ip
BPF Controller
(User App)
. . .
bpf() SYSCALL
KERNEL SPACE
Map 1
(Shared memory)
Map 2
(Shared memory)
BPF
BPF CALLBACK !!
Callback !!
BPF
Callback !!

BPF library
in-iproute2

tc

ip

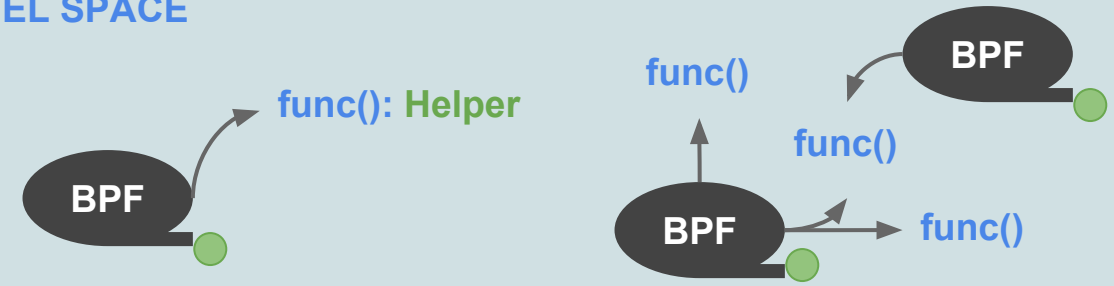
BPF library: libbpf
prog/map
load, attach, control

BPF Controller
(User App)

. . .

bpf() SYSCALL

KERNEL SPACE

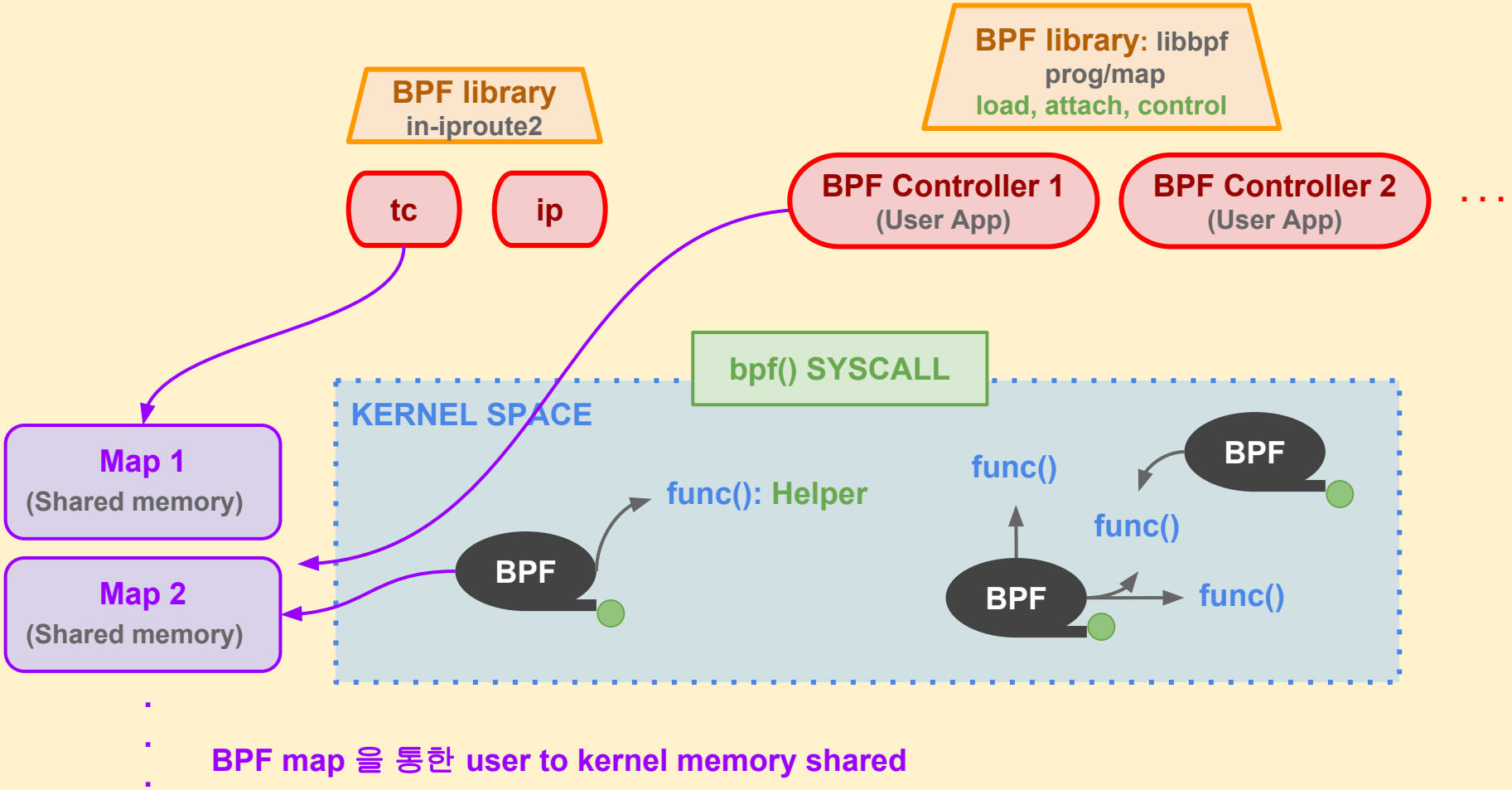


Map 1
(Shared memory)

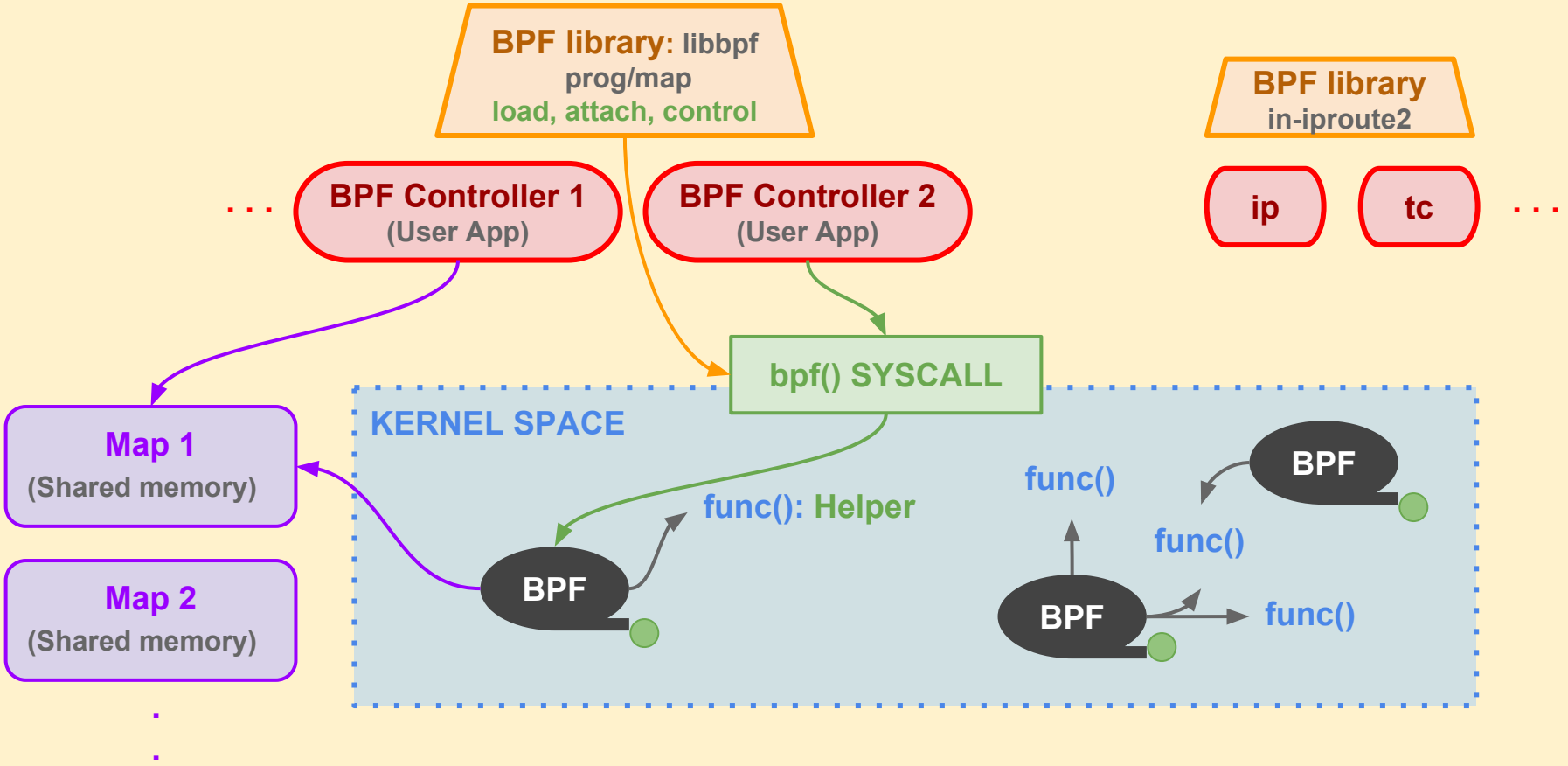
Map 2
(Shared memory)

. . .

BPF Helper 함수를 통한 커널함수 호출 leveraging !!


BPF library: libbpf
prog/map
load, attach, control
BPF library
in-iproute2
tc
ip
BPF Controller 1
(User App)
BPF Controller 2
(User App)
. . .
bpf() SYSCALL
KERNEL SPACE
Map 1
(Shared memory)
Map 2
(Shared memory)
func(): Helper
BPF
func()
BPF
func()
BPF
func()
BPF map 을 통한 user to kernel memory shared

BPF Architecture:



BPF use cases:

1. Facebook - Katran : A high performance layer 4 load balancer
https://github.com/facebookincubator/katran

2. Cilium: Security and Networking for Containers with BPF and XDP
https://github.com/cilium/cilium

3. BCC: BPF Compiler Collection & Kernel Tracing tools
https://github.com/iovisor/bcc

4. IR decoding with BPF "BPF-ization: from daemon to BPF prog"
https://lwn.net/Articles/759188/

- Restricted C: BPF programming
- Helper Functions: leveraging kernel functions
- Hook points: the location it'd be called

Thanks

SOSCON 2018

SAMSUNG OPEN SOURCE CONFERENCE 2018