uftrace 컨트리뷰톤 모임

2019.09.17

김홍규

Quick Installation

```
$ git clone https://github.com/namhyung/uftrace && cd uftrace
```

```
$ ./configure
uftrace detected system features:
...          prefix: /usr/local
...          libelf: [ OFF ] - more flexible ELF data handling
...           libdw: [ OFF ] - DWARF debug info support
...    libpython2.7: [ OFF ] - python scripting support
...     libncursesw: [ OFF ] - TUI support
...    cxa_demangle: [ OFF ] - full demangler support with libstdc++
...      perf_event: [ OFF ] - perf (PMU) event support
...        schedule: [ OFF ] - scheduler event support
...        capstone: [ OFF ] - full dynamic tracing support

$ make
$ sudo make install
```

모든 기능을 optional 하게 사용해 OFF 인 경우 해당 기능을 끄고 실행 가능

Quick Installation

```
$ git clone https://github.com/namhyung/uftrace && cd uftrace
```

```
$ ./configure
uftrace detected system features:
...          prefix: /usr/local
...          libelf: [ on  ] - more flexible ELF data handling
...           libdw: [ OFF ] - DWARF debug info support
...    libpython2.7: [ OFF ] - python scripting support
...     libncursesw: [ OFF ] - TUI support
...    cxa_demangle: [ on  ] - full demangler support with libstdc++
...      perf_event: [ on  ] - perf (PMU) event support
...        schedule: [ on  ] - scheduler event support
...        capstone: [ OFF ] - full dynamic tracing support

$ make
$ sudo make install
```

일부 기능이 on 인경우 관련 라이브러리의 도움으로 더 많은 기능 사용 가능

Quick Installation

```
$ git clone https://github.com/namhyung/uftrace && cd uftrace

# To install required packages
$ sudo ./misc/install-deps.sh

$ ./configure
uftrace detected system features:
...          prefix: /usr/local
...          libelf: [ on  ] - more flexible ELF data handling
...           libdw: [ on  ] - DWARF debug info support
...    libpython2.7: [ on  ] - python scripting support
...     libncursesw: [ on  ] - TUI support
...    cxa_demangle: [ on  ] - full demangler support with libstdc++
...      perf_event: [ on  ] - perf (PMU) event support
...        schedule: [ on  ] - scheduler event support
...        capstone: [ on  ] - full dynamic tracing support

$ make
$ sudo make install
```

./misc/install-deps.sh 스크립트로 관련 라이브러리들 전부 설치 가능

Quick Installation

```
$ git clone https://github.com/namhyung/uftrace && cd uftrace

# To install required packages
$ sudo ./misc/install-deps.sh

$ ./configure --prefix=/home/honggyu/usr
uftrace detected system features:
...          prefix: /home/honggyu/usr
...          libelf: [ on  ] - more flexible ELF data handling
...           libdw: [ on  ] - DWARF debug info support
...    libpython2.7: [ on  ] - python scripting support
...     libncursesw: [ on  ] - TUI support
...    cxa_demangle: [ on  ] - full demangler support with libstdc++
...      perf_event: [ on  ] - perf (PMU) event support
...        schedule: [ on  ] - scheduler event support
...        capstone: [ on  ] - full dynamic tracing support

$ make
$ make install
```

--prefix 로 설치 경로를 바꿀 수 있어서 root 권한이 없는 서버에도 설치 가능

```
$ tree -d uftrace
uftrace
├── arch            # architecture 특화된 코드를 관리
│   ├── aarch64
│   ├── arm
│   ├── i386
│   └── x86_64
├── check-deps      # configure 스크립트 실행시 기능 테스트
├── cmds            # 명령어(record, replay 등) 코드 관리
├── doc             # 문서 관련 코드 (man page, slide 문서)
├── gdb             # gdb 디버깅을 위한 편의 스크립트
│   └── uftrace
├── libmcount       # libmcount.so로 컴파일되어 대상 프로그램과 함께 동작
├── libtraceevent    # 커널 함수 정보가 기록된 버퍼를 읽는 코드
│   └── include
│       ├── asm
│       └── linux
├── misc            # 기타 등등...
├── scripts         # -S 옵션이나 script 명령어로 실행할 예제 스크립트
├── tests           # 테스트 관련 코드들
│   └── arch
│       ├── arm
│       └── x86_64
└── utils           # libmcount.so와 uftrace에서 공통으로 사용하는 코드
```

```
$ tree -d uftrace
uftrace
├── arch            # architecture 특화된 코드를 관리
│   ├── aarch64
│   ├── arm
│   ├── i386
│   └── x86_64
├── check-deps      # configure 스크립트 실행시 기능 테스트
├── cmds            # 명령어(record, replay 등) 코드 관리
├── doc             # 문서 관련 코드 (man page, slide 문서)
├── gdb             # gdb 디버깅을 위한 편의 스크립트
│   └── uftrace
├── libmcount       # libmcount.so로 컴파일되어 대상 프로그램과 함께 동작
├── libtraceevent    # 커널 함수 정보가 기록된 버퍼를 읽는 코드
│   └── include
│       ├── asm
│       └── linux
├── misc            # 기타 등등...
├── scripts         # -S 옵션이나 script 명령어로 실행할 예제 스크립트
├── tests           # 테스트 관련 코드들
│   └── arch
│       ├── arm
│       └── x86_64
└── utils           # libmcount.so와 uftrace에서 공통으로 사용하는 코드
```

```
$ tree -d uftrace
uftrace
├── arch          # architecture 특화된 코드를 관리
│   ├── aarch64
│   ├── arm
│   ├── i386
│   └── x86_64
├── cmds          # 명령어(record, replay 등) 코드 관리
├── doc           # 문서 관련 코드 (man page, slide 문서)
├── libmcount     # libmcount.so로 컴파일되어 대상 프로그램과 함께 동작
├── misc          # 기타 등등...
├── scripts       # -S 옵션이나 script 명령어로 실행할 예제 스크립트
├── tests         # 테스트 관련 코드들
└── utils         # libmcount.so와 uftrace에서 공통으로 사용하는 코드
```

uftrace 소개

- **uftrace 가 분석할 수 있는 것들**
  - C/C++ 사용자(user-space) 함수
    - 컴파일 시 **-pg** 나 **-finstrument-functions** 옵션 필요
    - 또는 **-finstrument-functions-after-inlining** (clang only)
    - 또는 **-fxray-instrument** (clang only)
  - 라이브러리 함수 (library functions)
  - 리눅스 커널(kernel-space) 내부 함수
  - 시스템 이벤트들

```
$ gcc hello.c
```

```
$ gcc -pg hello.c
```

```
$ gcc -pg hello.c

$ ./a.out
```

```
$ gcc -pg hello.c

$ ./a.out
Hello, World!
```

```
$ gcc -pg hello.c

$ uftrace ./a.out
```

```
$ gcc -pg hello.c

$ uftrace ./a.out
Hello, World!
```

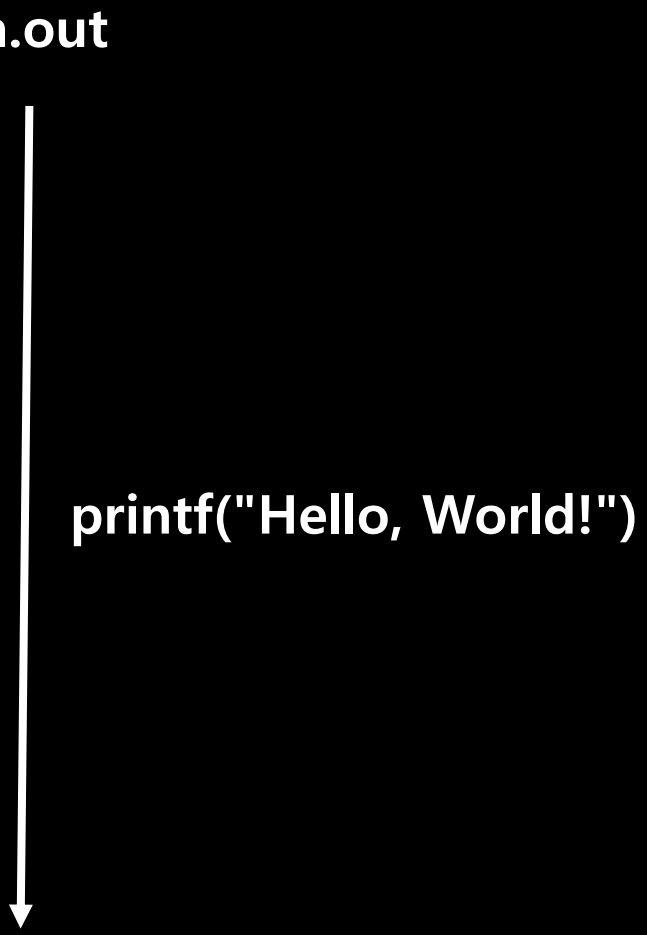
```
$ gcc -pg hello.c

$ uftrace ./a.out
Hello, World!
# DURATION     TID     FUNCTION
   1.447 us [120218] | __monstartup();
   0.997 us [120218] | __cxa_atexit();
            [120218] | main() {
   7.214 us [120218] |   printf();
   8.246 us [120218] | } /* main */
```

```
$ gcc hello.c
$ ./a.out
```

```
$ gcc hello.c
$ ./a.out
```

a.out

```
$ gcc hello.c
$ ./a.out
```



```
$ gcc -pg hello.c
```

```
$ gcc -pg hello.c
$ ./a.out
```

```
$ gcc -pg hello.c
$ ./a.out
```

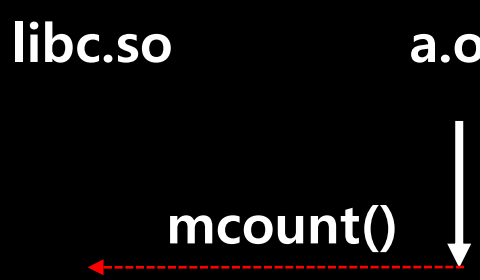
libc.so a.out

```
$ gcc -pg hello.c
$ ./a.out
```

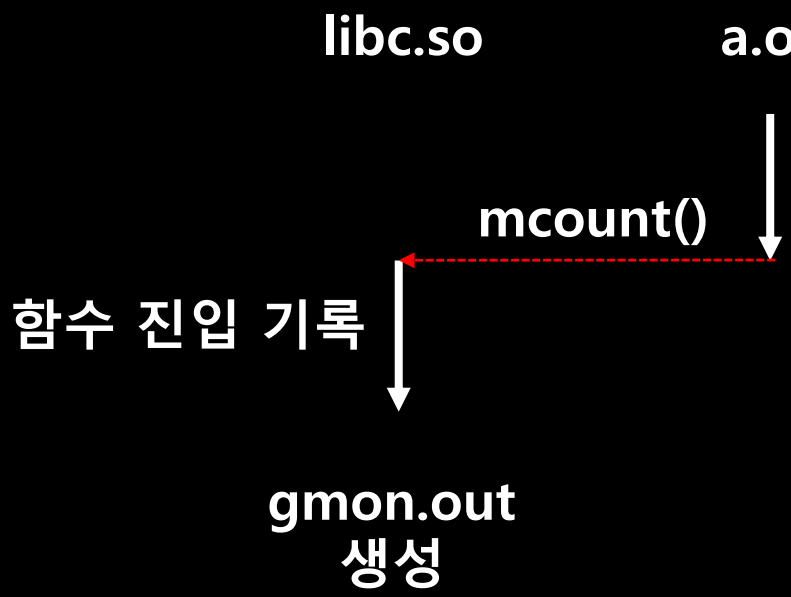
libc.so

a.out

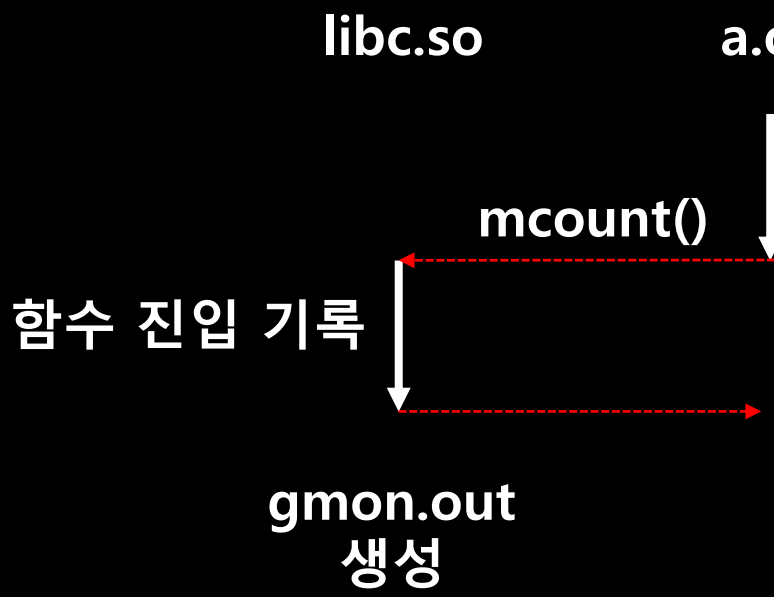
```
$ gcc -pg hello.c
$ ./a.out
```



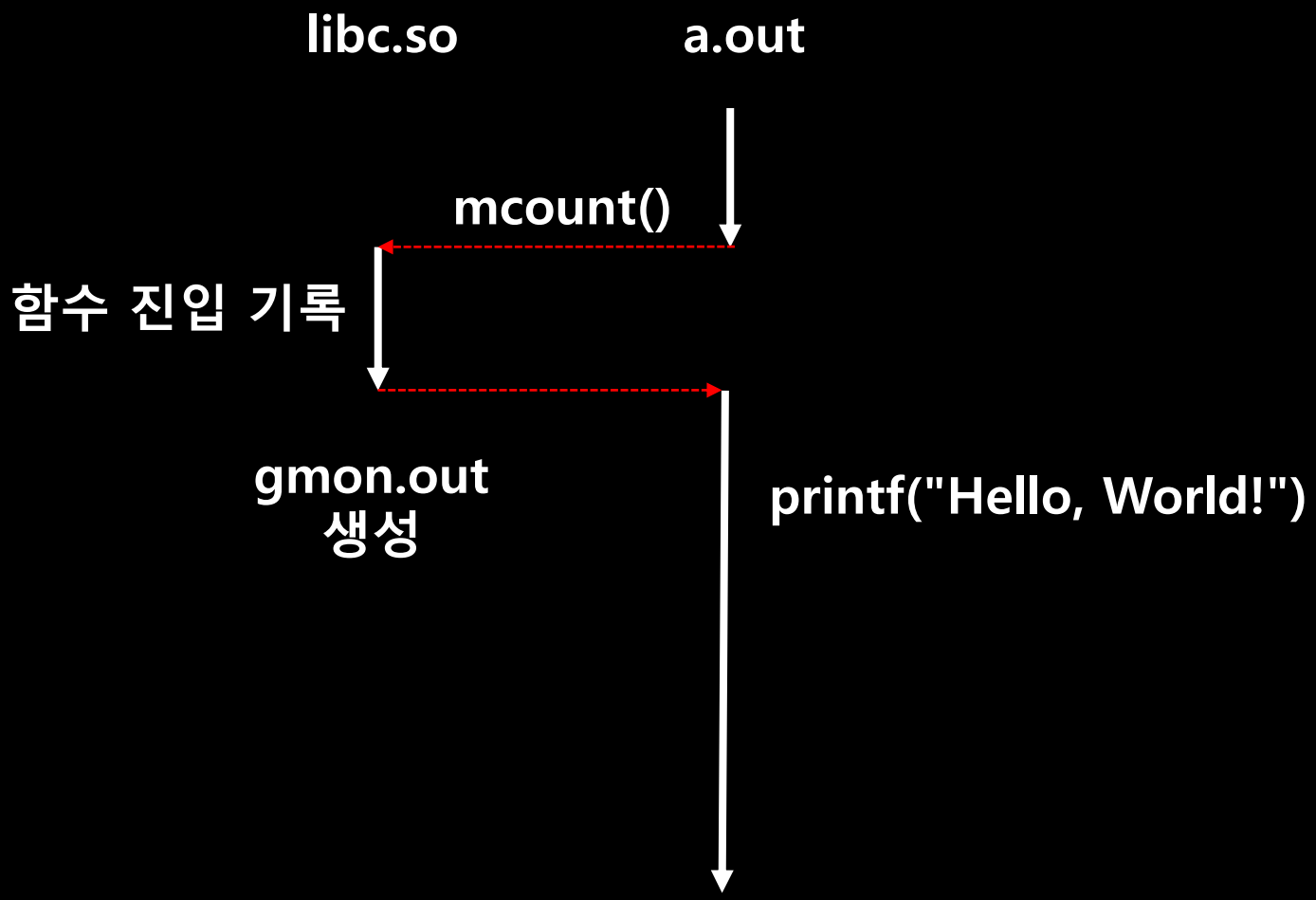
```
$ gcc -pg hello.c
$ ./a.out
```



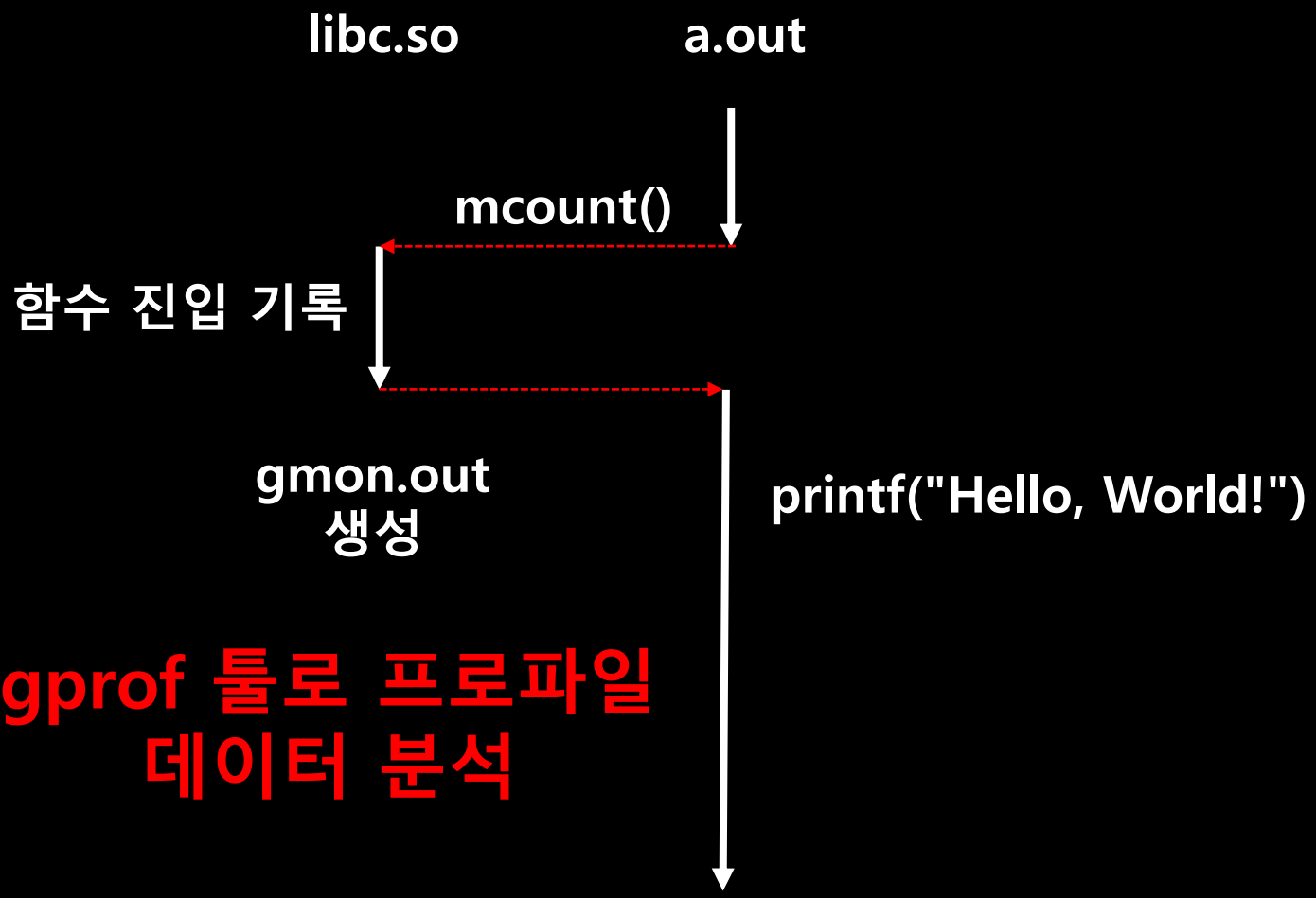
```
$ gcc -pg hello.c
$ ./a.out
```



```
$ gcc -pg hello.c
$ ./a.out
```



```
$ gcc -pg hello.c
$ ./a.out
```



gprof 툴로 프로파일 데이터 분석

```
$ uftrace ./a.out
```

```
$ uftrace ./a.out
```

LD_PRELOAD=libmcount.so a.out

```
$ uftrace ./a.out
```

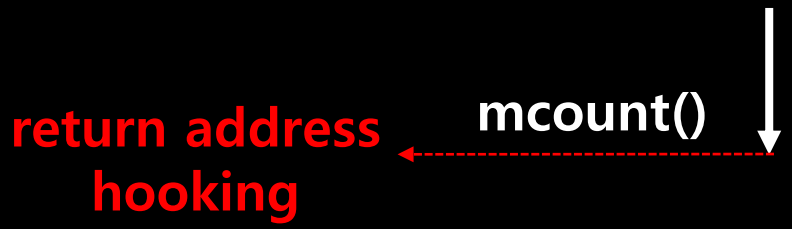
LD_PRELOAD=libmcount.so a.out

```
$ uftrace ./a.out
```

LD_PRELOAD=libmcount.so a.out

```
$ uftrace ./a.out
```

LD_PRELOAD=libmcount.so a.out



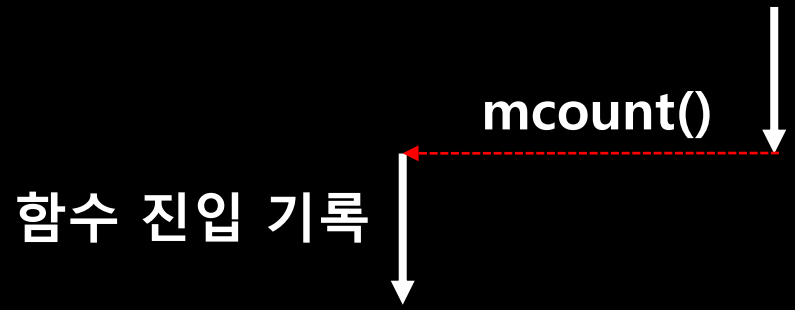
원래 함수의 반환 주소를 별도의 공간에 보관해두고
실제 반환 주소를 libmcount.so 로 변경해 함수가 리턴될때
실행 흐름을 얻을 수 있도록 함.

```
static int __mcount_entry(unsigned long *parent_loc, unsigned long child,
                struct mcount_regs *regs)
{
        ...
    /* fixup the parent_loc in an arch-dependant way (if needed) */
    parent_loc = mcount_arch_parent_location(&symtabs, parent_loc, child);

    rstack = &mtdp->rstack[mtdp->idx++];

    rstack->depth      = mtdp->record_idx;
    rstack->dyn_idx    = MCOUNT_INVALID_DYNIDX;
    rstack->parent_loc = parent_loc;
    rstack->parent_ip  = *parent_loc;
    rstack->child_ip   = child;
    rstack->start_time = mcount_gettime();
    rstack->end_time   = 0;
    rstack->flags      = 0;
    rstack->nr_events  = 0;
    rstack->event_idx  = ARGBUF_SIZE;

    /* hijack the return address of child */
    *parent_loc = mcount_return_fn;    /* 일반적으로 mcount_return */

    /* restore return address of parent */
    if (mcount_auto_recover)
        mcount_auto_restore(mtdp);
        ...
}
```

uftrace/libmcount/mcount.c

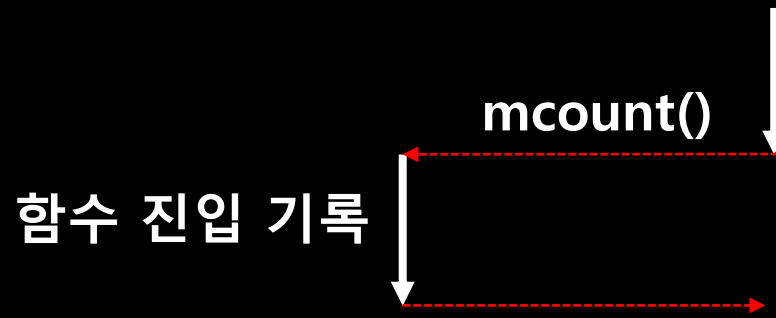
```
$ uftrace ./a.out
```

LD_PRELOAD=libmcount.so a.out



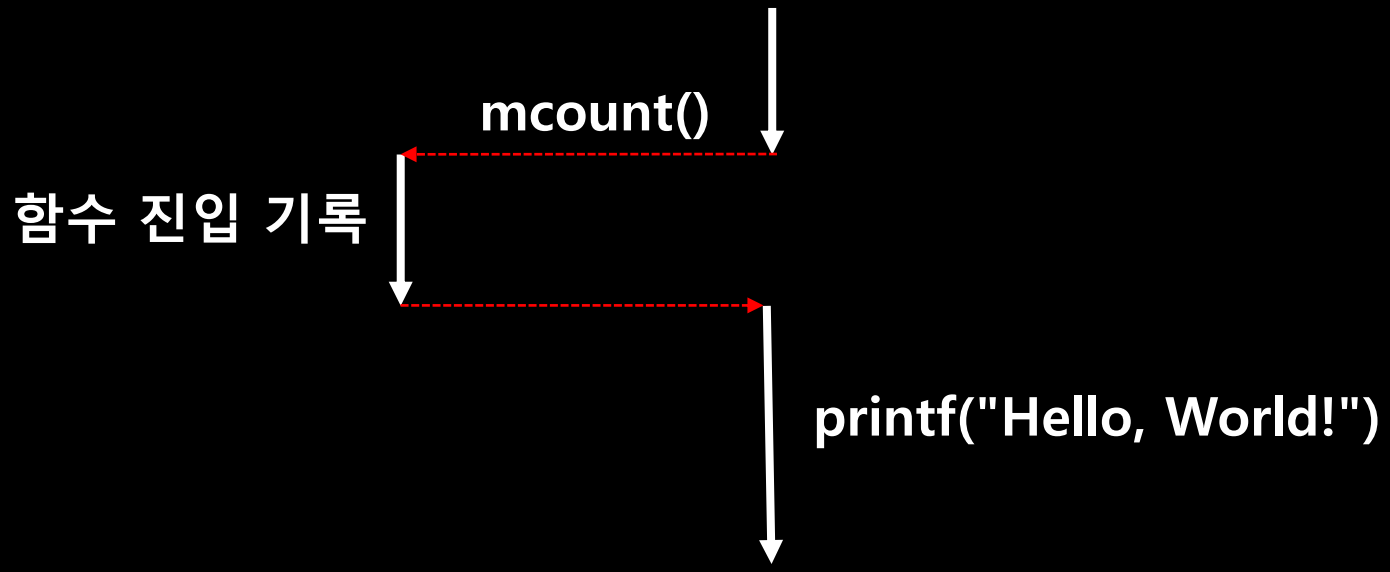
```
$ uftrace ./a.out
```

LD_PRELOAD=libmcount.so a.out



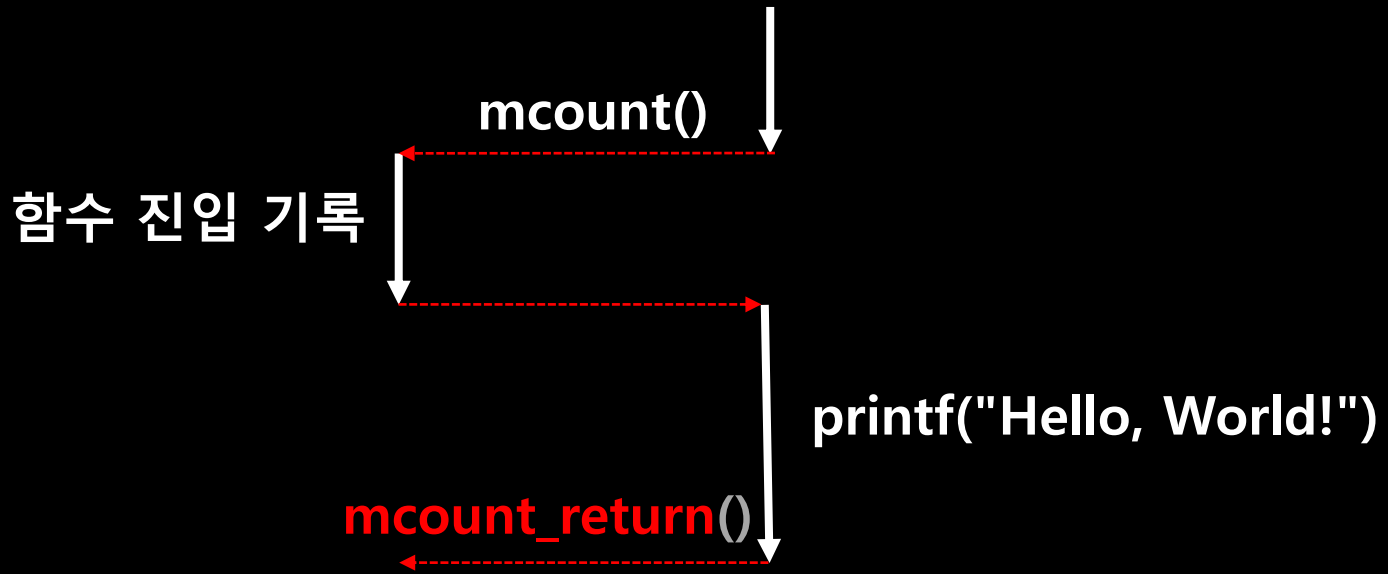
```
$ uftrace ./a.out
```

LD_PRELOAD=libmcount.so a.out



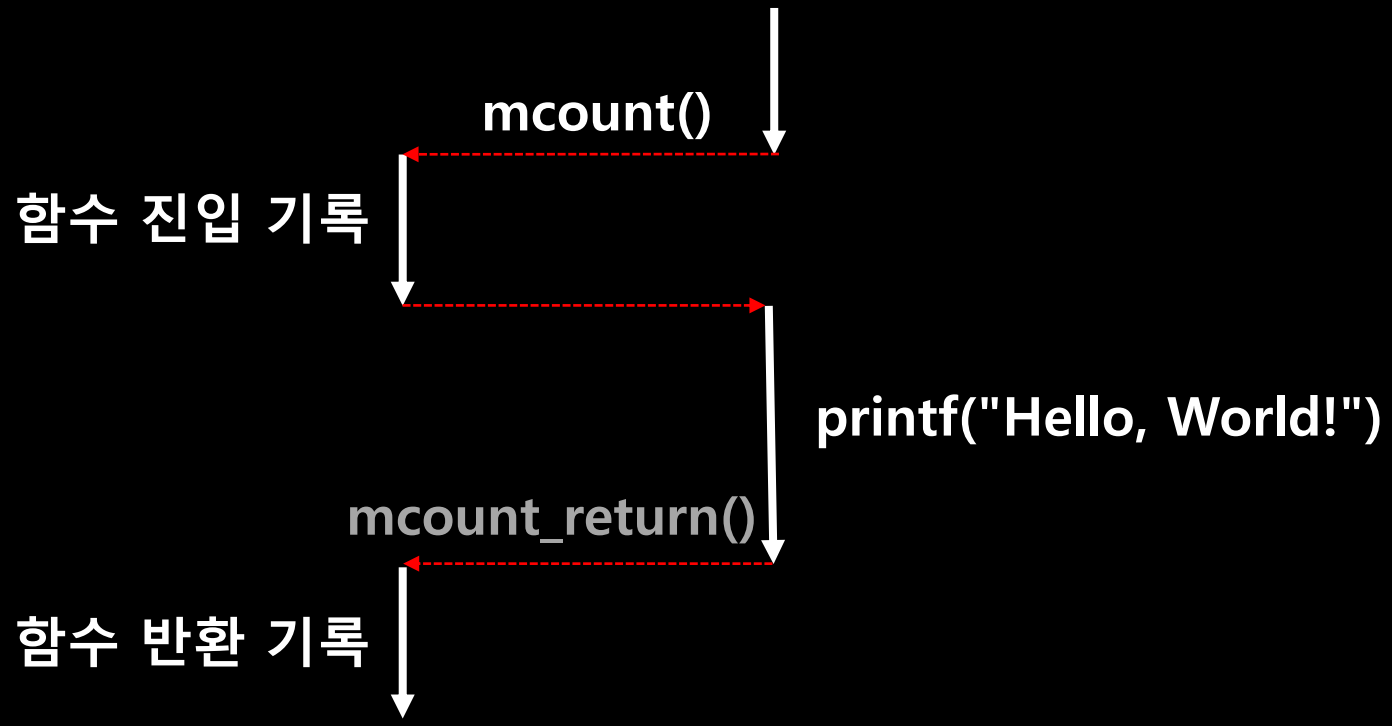
```
$ uftrace ./a.out
```

LD_PRELOAD=libmcount.so a.out



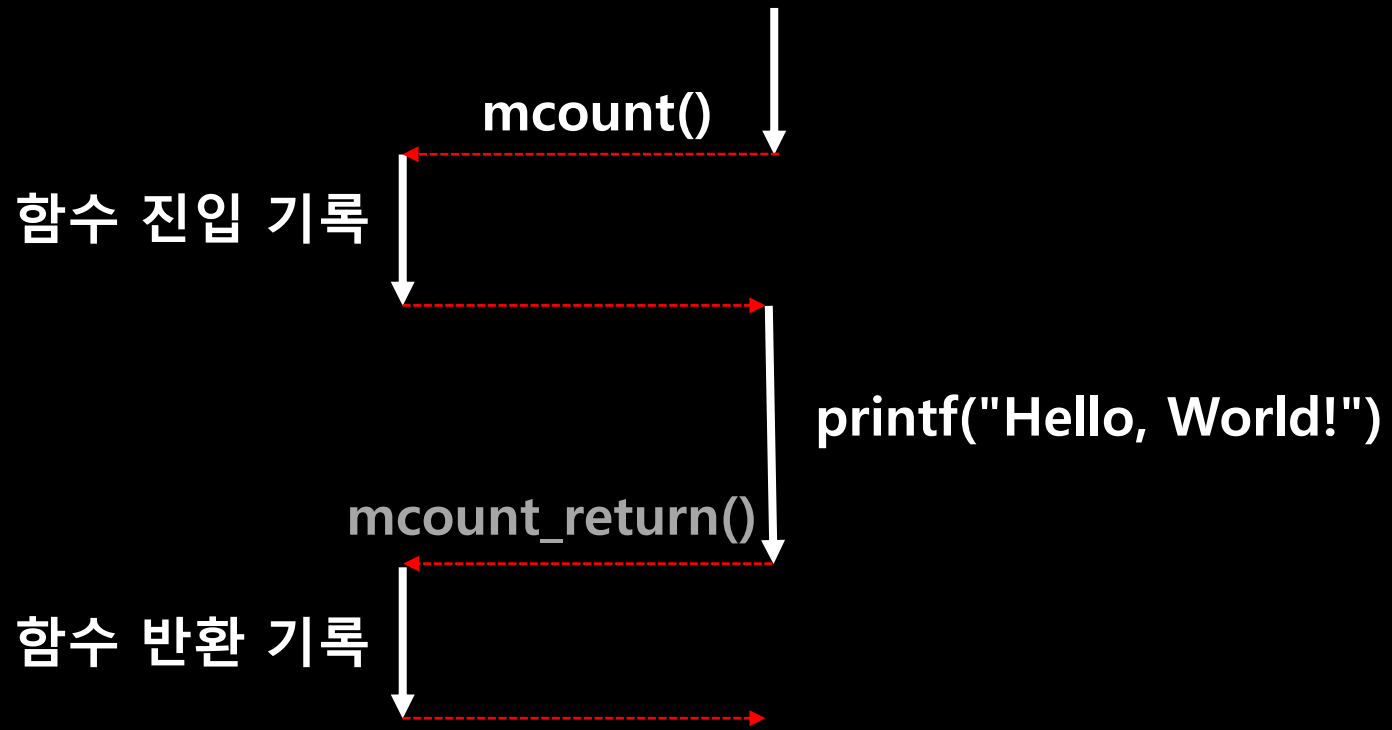
```
$ uftrace ./a.out
```

LD_PRELOAD=libmcount.so a.out



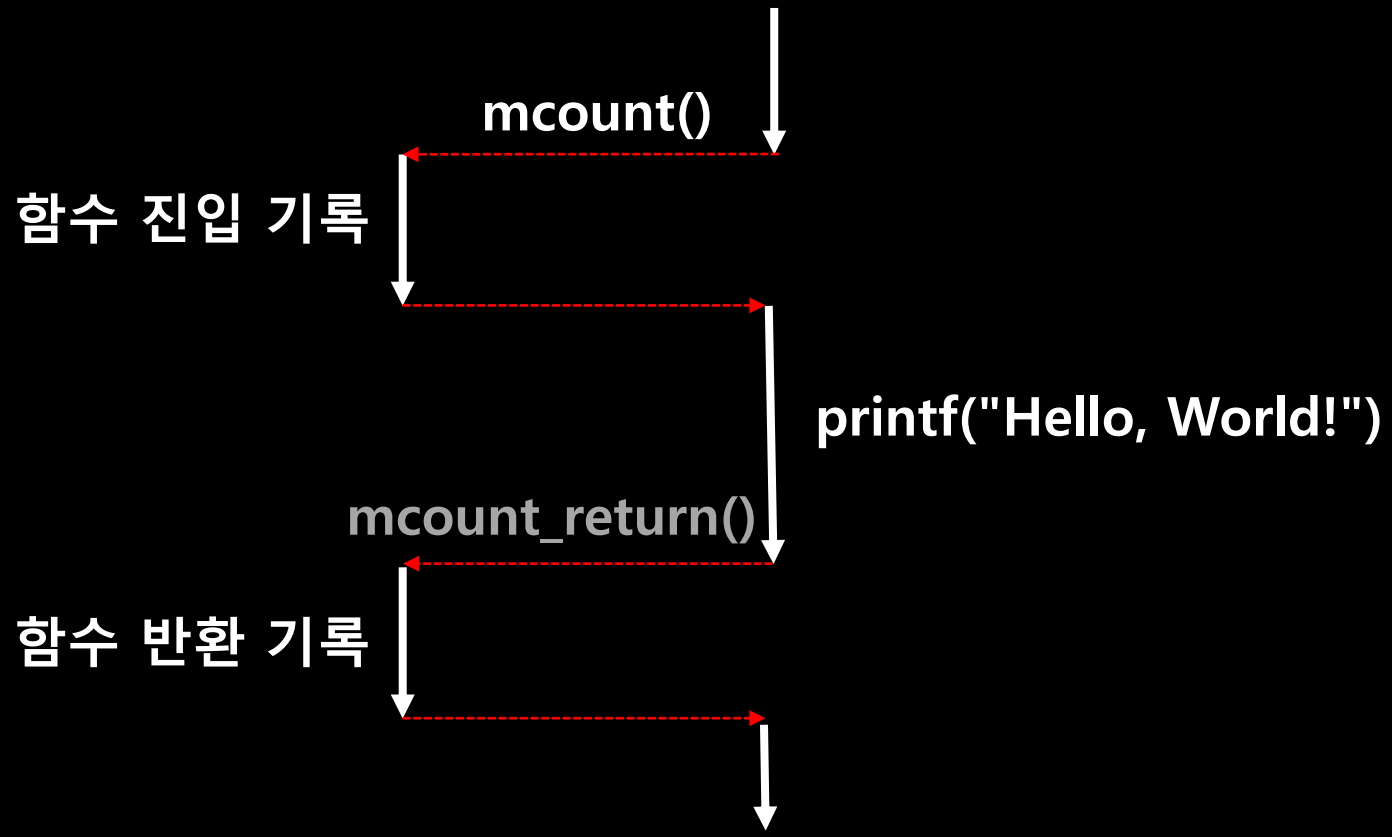
```
$ uftrace ./a.out
```

LD_PRELOAD=libmcount.so a.out

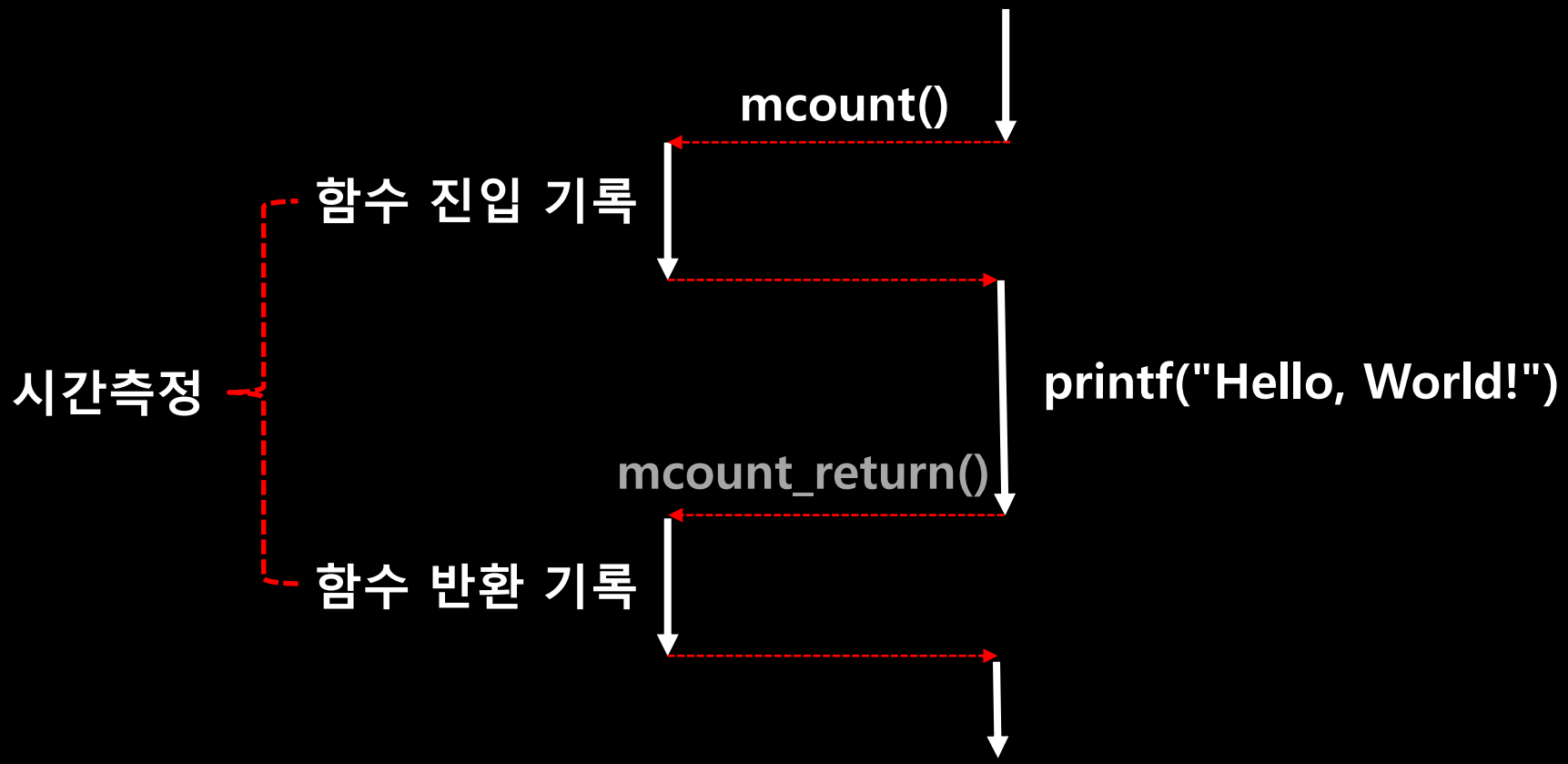


```
$ uftrace ./a.out
```

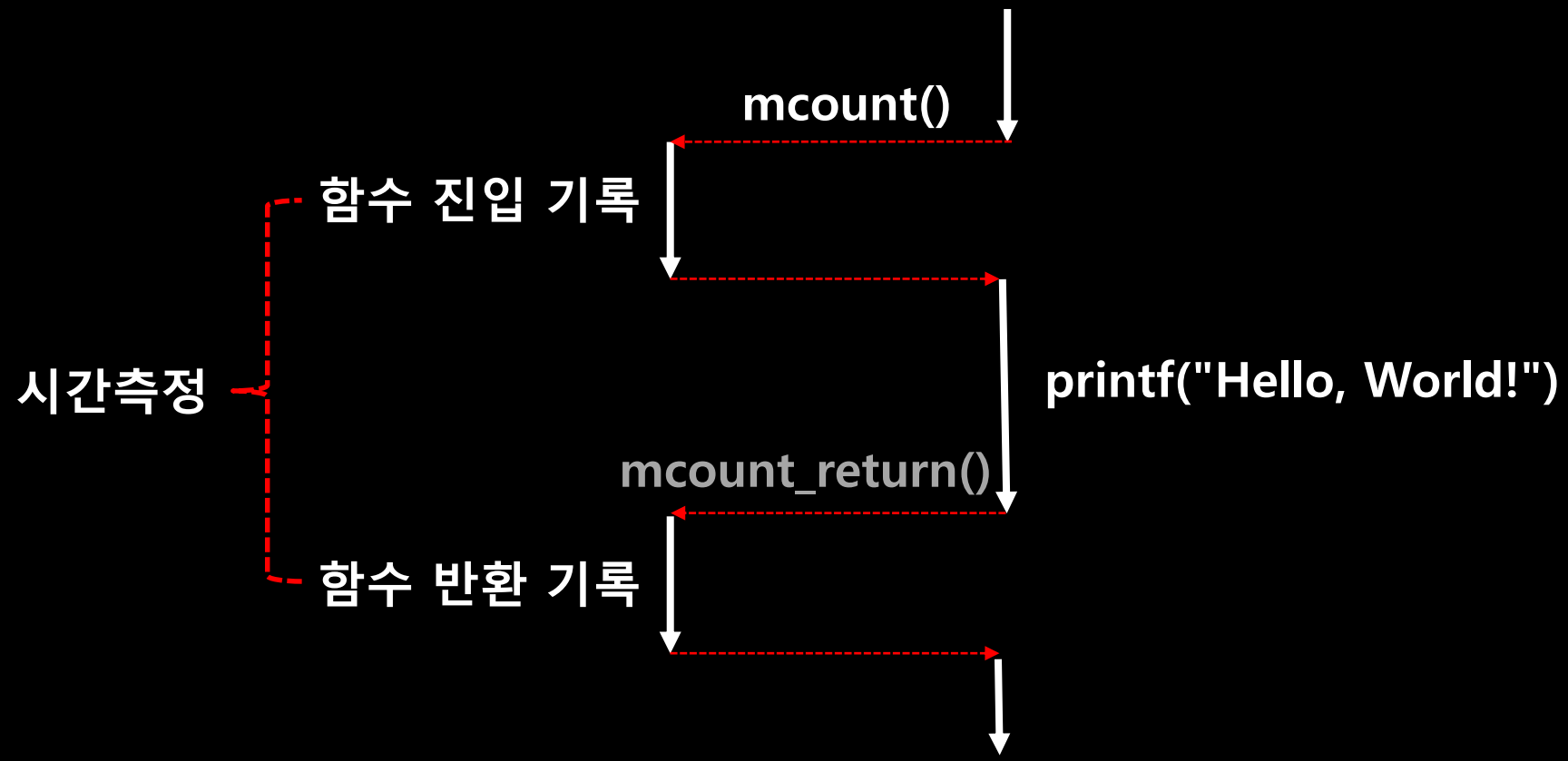
LD_PRELOAD=libmcount.so a.out



```
$ uftrace ./a.out
```



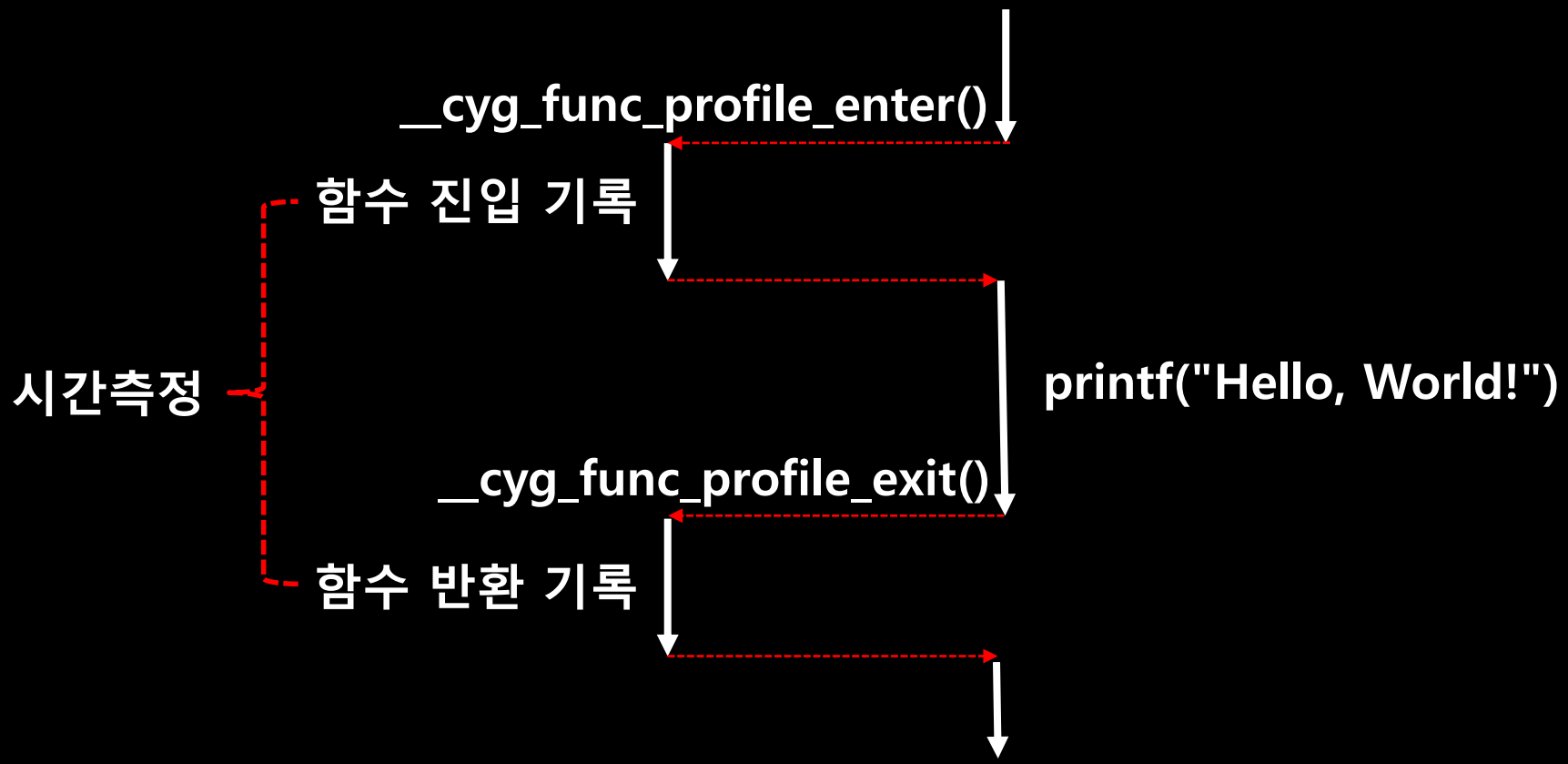
```
$ uftrace ./a.out
```



함수 진입 및 반환 기록을 위해 16바이트씩 shared memory 에 저장

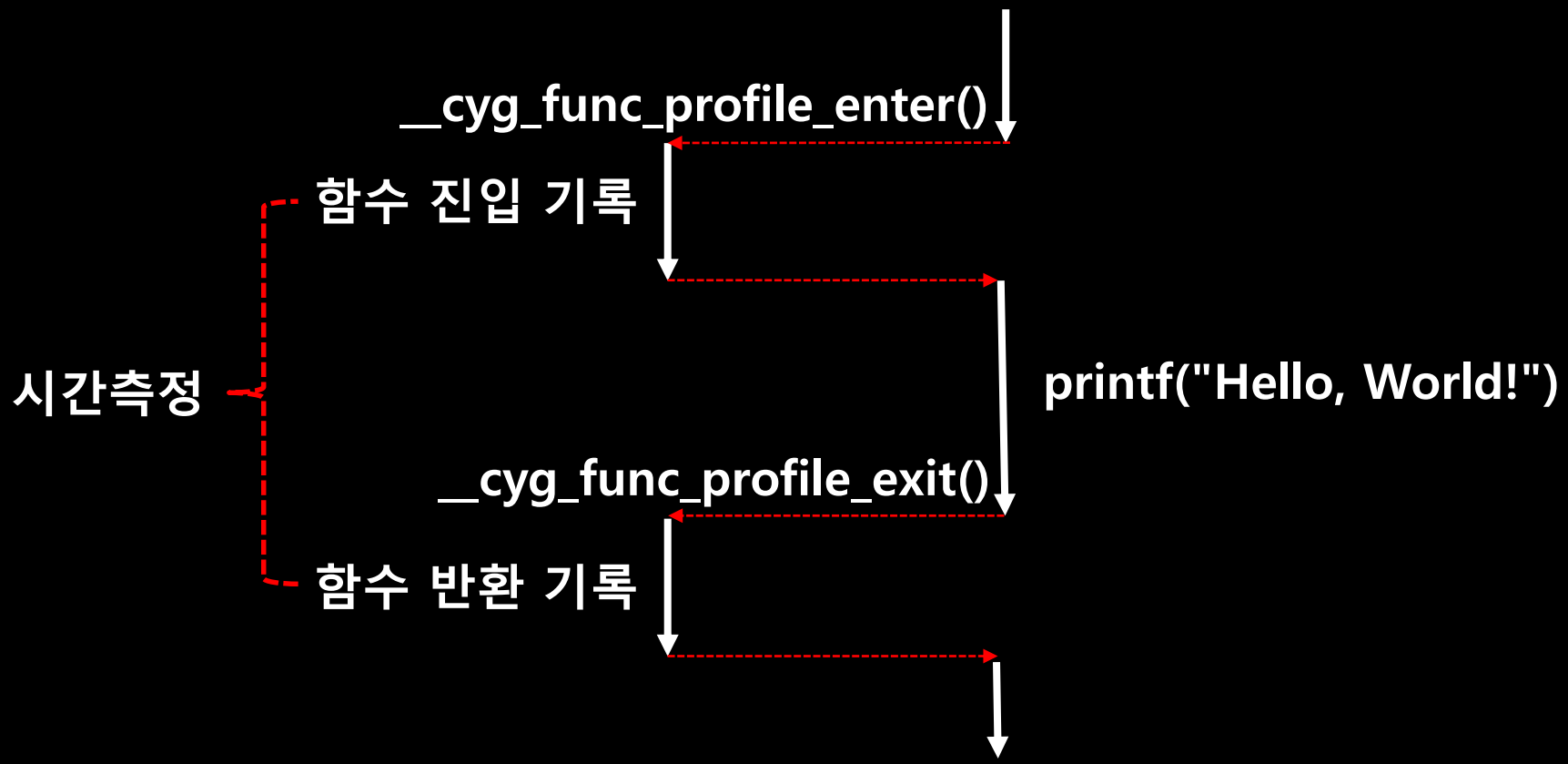
```
$ gcc -finstrument-funtions -fno-omit-frame-pointer hello.c
```

```
$ uftrace ./a.out
```



```
$ gcc -finstrument-funtions -fno-omit-frame-pointer hello.c
```

```
$ uftrace ./a.out
```



ARM/AArch64 환경에서는 return 주소 조작이 없는 이 방식이 더 안전함

Library Function Tracing

- PLT hooking 을 통해 라이브러리 함수 호출 추적 가능

```
void bar() {
  getpid();
}
void foo() {
  bar();
}
int main() {
  foo();
}
```

Library Function Tracing

- PLT hooking 을 통해 라이브러리 함수 호출 추적 가능

```
$ gcc -pg test.c
```

```
void bar() {
  getpid();
}
void foo() {
  bar();
}
int main() {
  foo();
}
```

Library Function Tracing

- PLT hooking 을 통해 라이브러리 함수 호출 추적 가능

```
$ gcc -pg test.c
```

```
void bar() {
  getpid();
}
void foo() {
  bar();
}
int main() {
  foo();
}
```

```
<bar>:
  call <mcount@plt>
  call <getpid@plt>  # indirect call in PLT
  ret

<foo>:
  call <mcount@plt>
  call <bar>
  ret

<main>:
  call <mcount@plt>
  call <foo>
  ret
```

Library Function Tracing

```
$ gcc -pg test.c

$ uftrace a.out
Hello
# DURATION    TID     FUNCTION
   1.087 us [12411] | __monstartup();
   0.790 us [12411] | __cxa_atexit();
            [12411] | main() {
            [12411] |   foo() {
            [12411] |     bar() {
   6.263 us [12411] |       getpid();
   7.016 us [12411] |     } /* bar */
   7.443 us [12411] |   } /* foo */
   7.826 us [12411] | } /* main */
```

Library Function Tracing

```
$ uftrace tests/t-fork
# DURATION    TID     FUNCTION
            [14528] | main() {
 127.033 us [14528] |   fork();
            [14528] |   wait() {
            [14540] |   } /* fork */
            [14540] |   a() {
            [14540] |     b() {
            [14540] |       c() {
   1.507 us [14540] |         getpid();
   2.987 us [14540] |       } /* c */
   3.464 us [14540] |     } /* b */
   3.854 us [14540] |   } /* a */
  13.394 us [14540] | } /* main */
 799.270 us [14528] |   } /* wait */
            [14528] |   a() {
            [14528] |     b() {
            [14528] |       c() {
   2.410 us [14528] |         getpid();
   3.470 us [14528] |       } /* c */
   3.833 us [14528] |     } /* b */
   4.144 us [14528] |   } /* a */
 952.797 us [14528] | } /* main */
```

Prebuilt Binary Tracing

```
$ uftrace -la /bin/dd if=/dev/zero of=out bs=1k count=3
...
  29.792 us [  1364] | open("/dev/zero", O_RDONLY) = 4;
   8.125 us [  1364] | dup2();
   0.625 us [  1364] | __errno_location();
   5.364 us [  1364] | close(4) = 0;
   5.573 us [  1364] | lseek(0, 0, SEEK_CUR) = 0;
 224.218 us [  1364] | open("out", O_TRUNC|O_CREAT|O_WRONLY) = 4;
   2.760 us [  1364] | dup2();
   0.677 us [  1364] | __errno_location();
   2.917 us [  1364] | close(4) = 0;
   5.312 us [  1364] | clock_gettime(CLOCK_MONOTONIC, 0xffffe3fee7b8) = 0;
  17.604 us [  1364] | malloc(9219) = 0xaaaad08250e0;
  22.865 us [  1364] | read(0, 0xaaaad0826000, 1024) = 1024;
  65.468 us [  1364] | write(1, 0xaaaad0826000, 1024) = 1024;
   5.312 us [  1364] | read(0, 0xaaaad0826000, 1024) = 1024;
  19.687 us [  1364] | write(1, 0xaaaad0826000, 1024) = 1024;
   4.532 us [  1364] | read(0, 0xaaaad0826000, 1024) = 1024;
  17.447 us [  1364] | write(1, 0xaaaad0826000, 1024) = 1024;
   9.063 us [  1364] | close(0) = 0;
 228.957 us [  1364] | close(1) = 0;
...
```

Dynamic Relocation

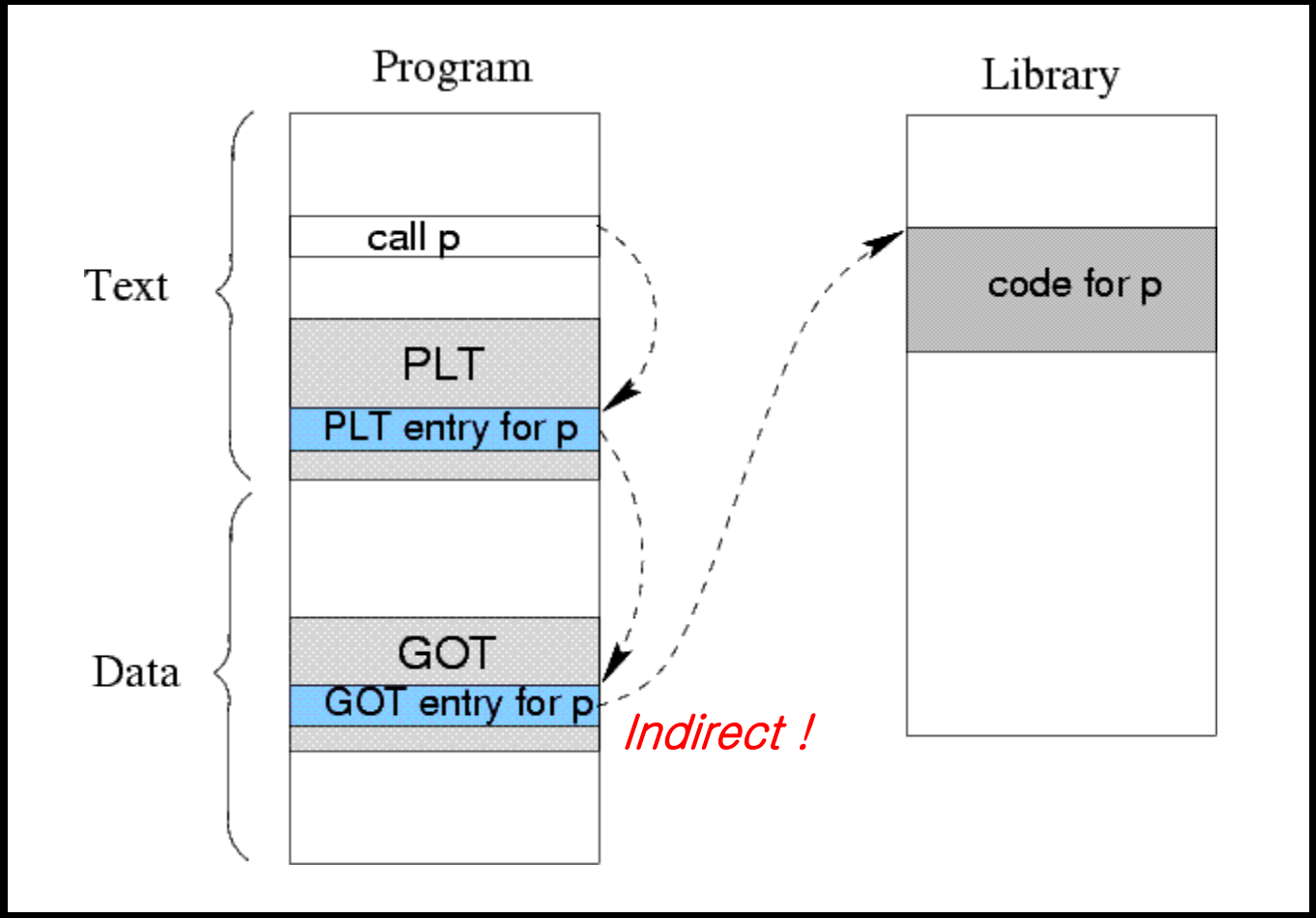
- **Position Independent Code (PIC)**
  - used when the load address for a program is not fixed
    - shared libraries
  - PC-relative addressing can give PIC references.
    - offset from the code to the data
  - the linker creates a global offset table(GOT)
    - contains offsets to all global data
- **Pros.**
  - code doesn’t have to be relocated when loaded.
  - different processes can share the memory page of code.
- **Cons.**
  - additional load-time GOT relocation overhead
  - bigger and slower code

Dynamic Relocation (cont.)

- **ELF shared libraries use PIC**
  - no relocation for text sections
- **Global Offset Table (GOT)**
  - for global data references
  - each global symbol has a relocatable pointer in GOT
  - dynamic linker relocates these pointers at runtime.
- **Procedure Linkage Table (PLT)**
  - PLT invokes dynamic linker at the first reference of each function and relocates the address in the GOT.
  - to reduces program startup time, PLT invokes dynamic linker lazily on demand.

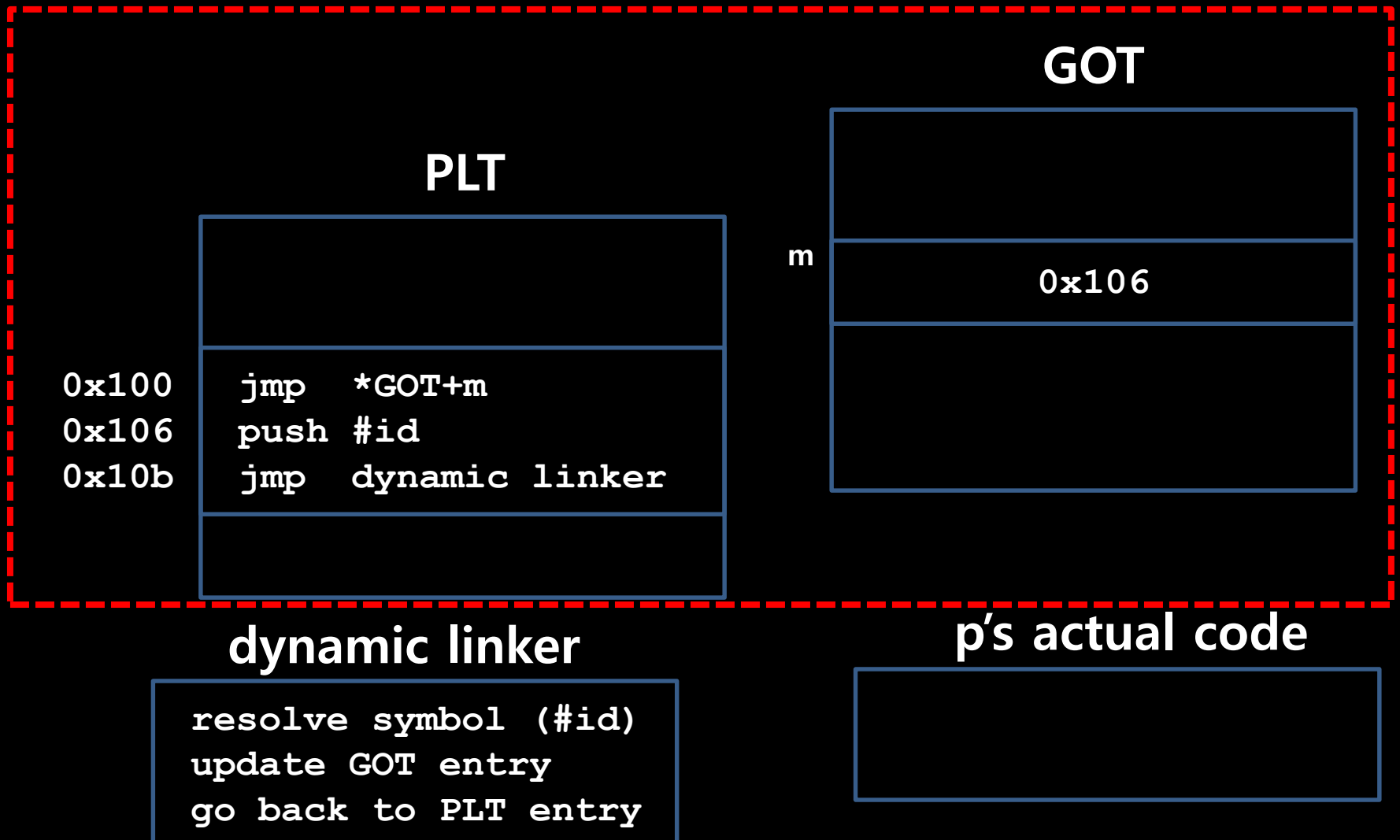
Dynamic Relocation (cont.)

- PLT and GOT



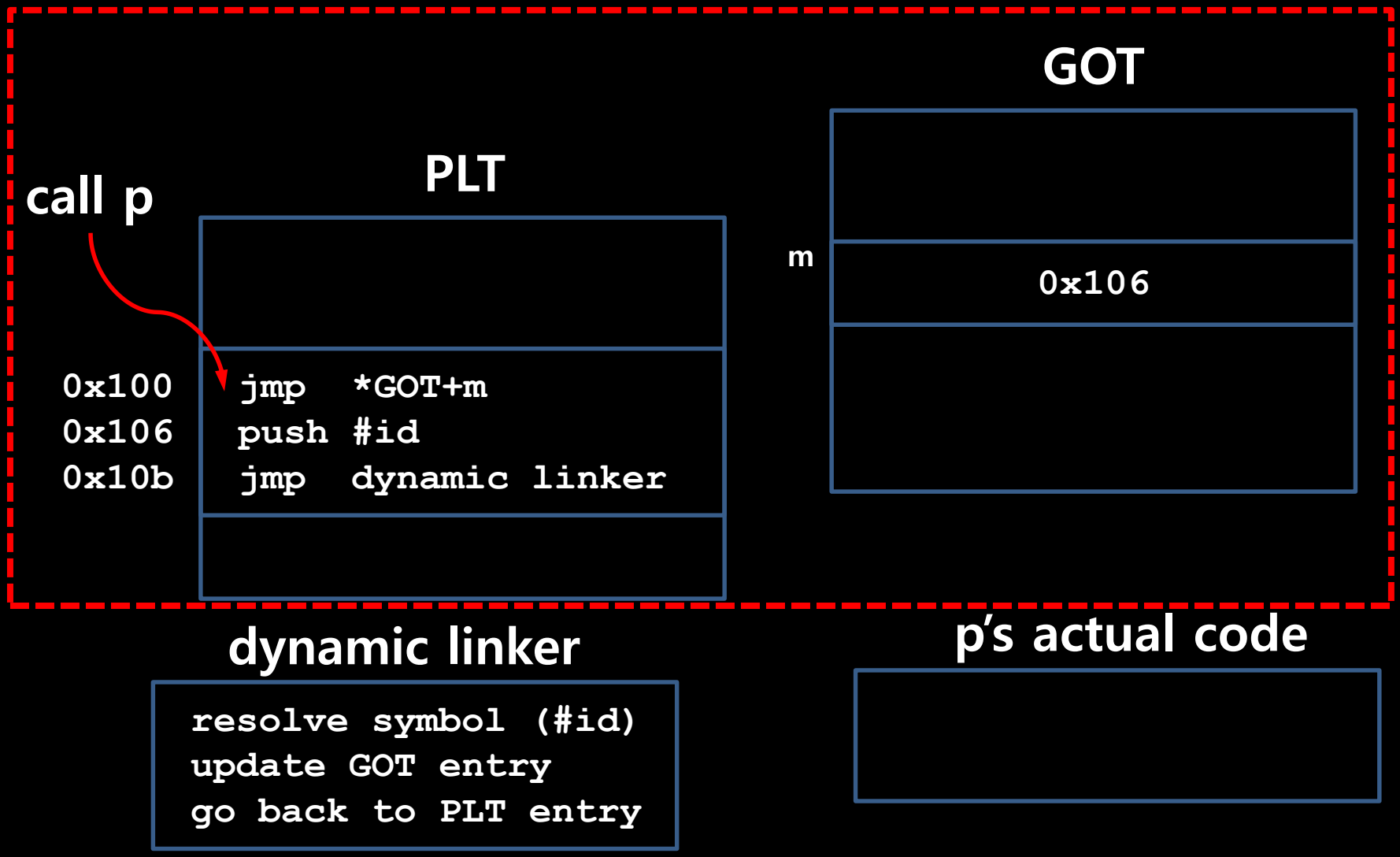
Dynamic Linking

- First Reference



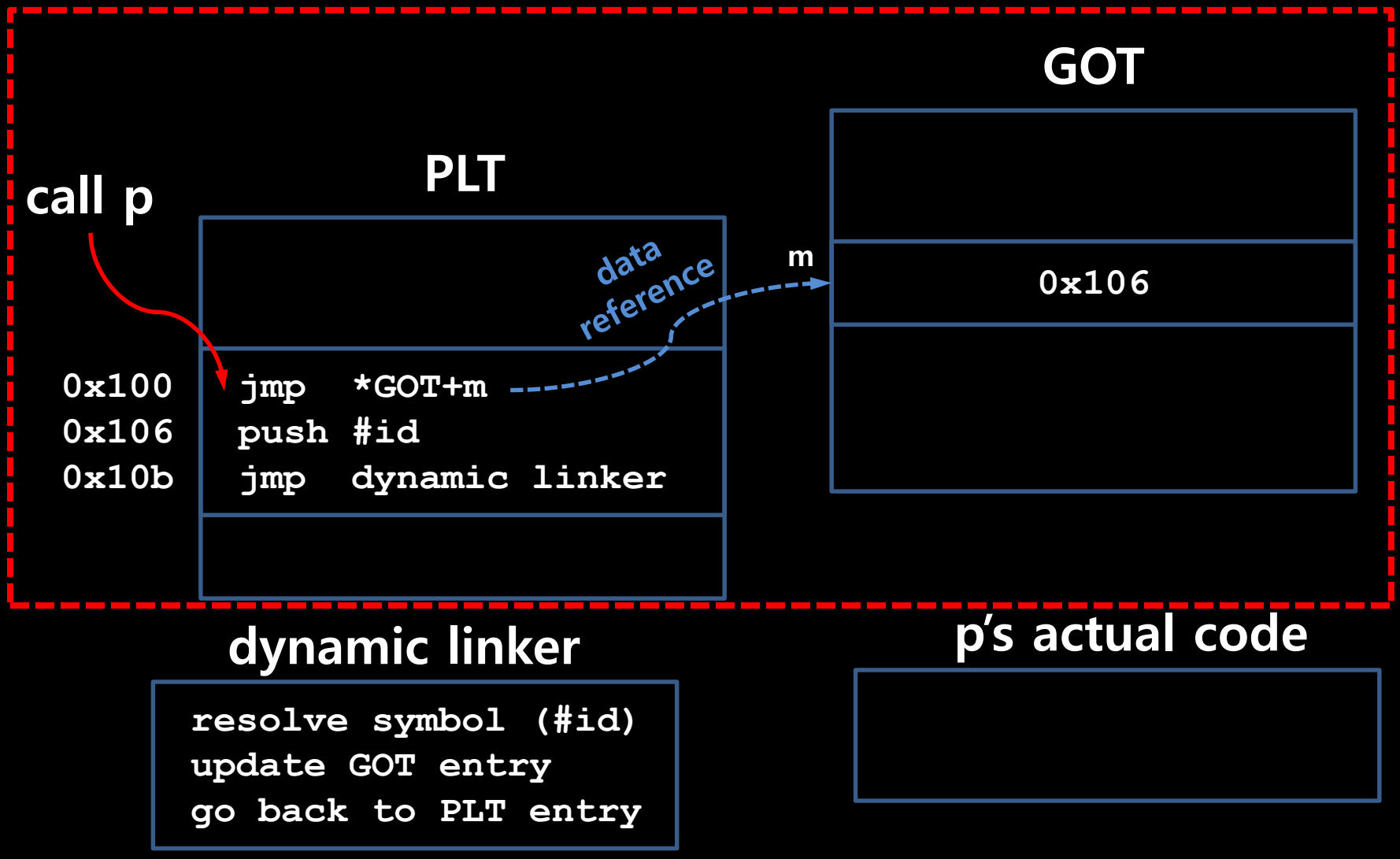
Dynamic Linking

- First Reference



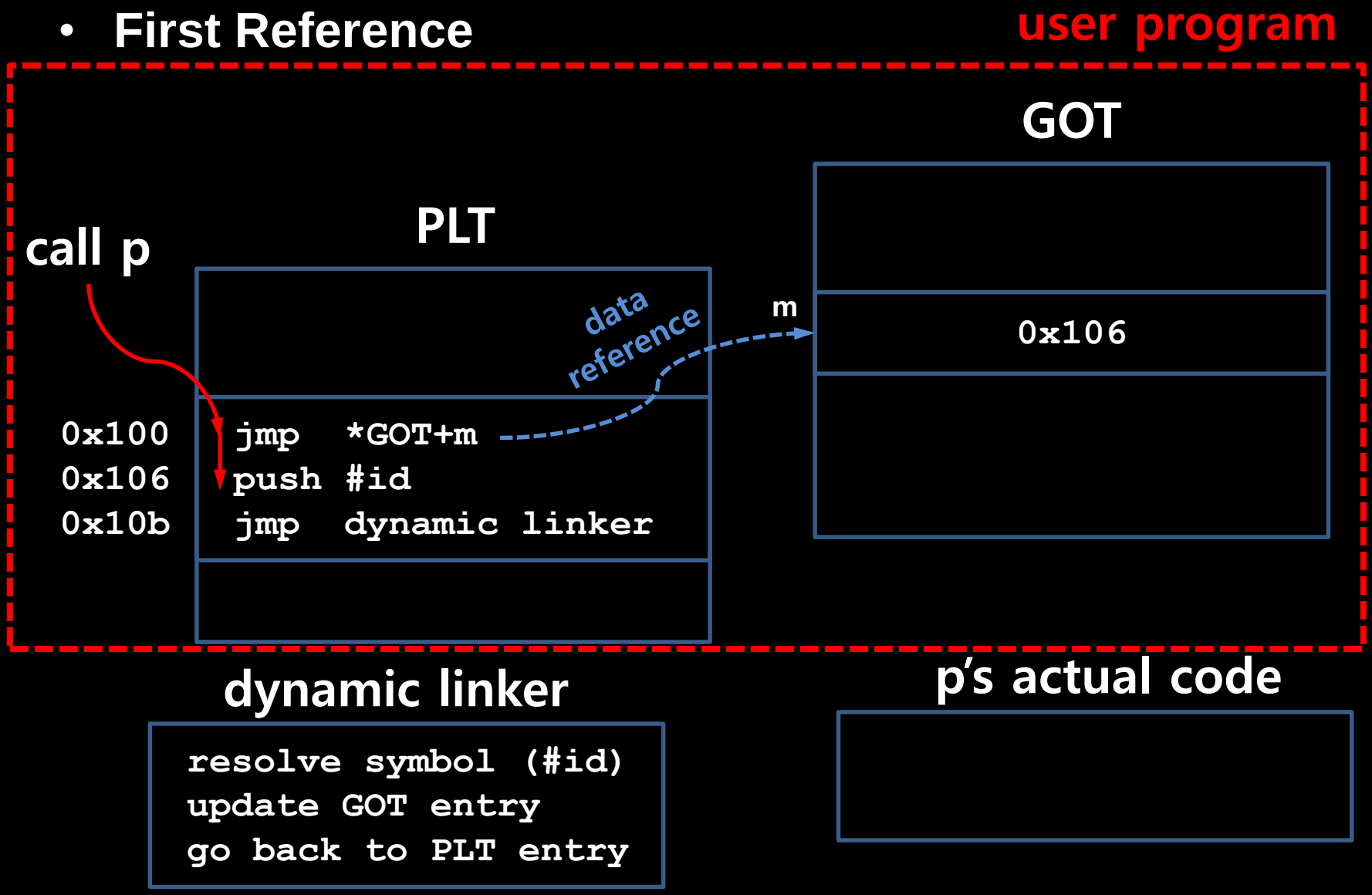
Dynamic Linking

- First Reference



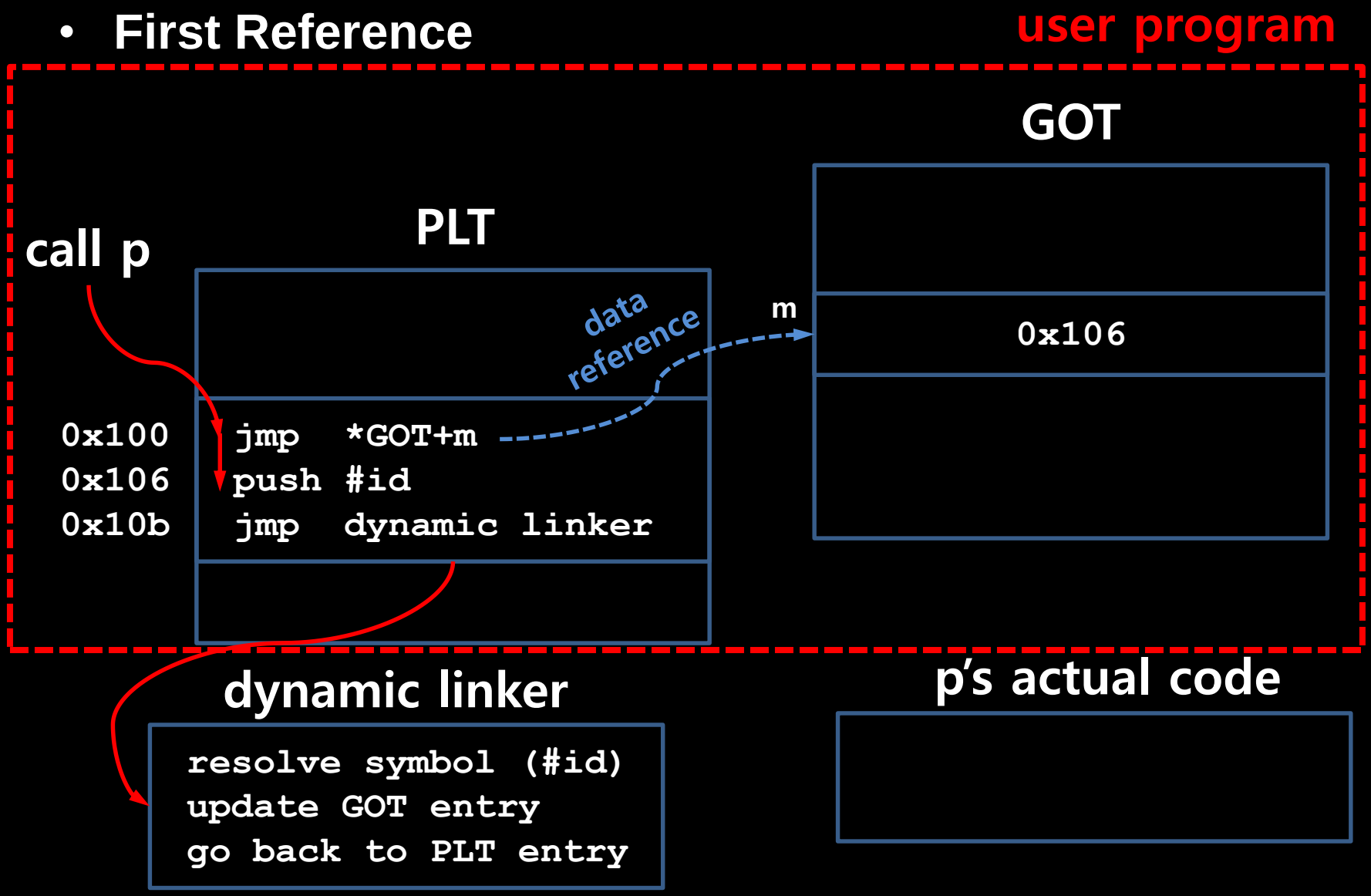
Dynamic Linking

- First Reference



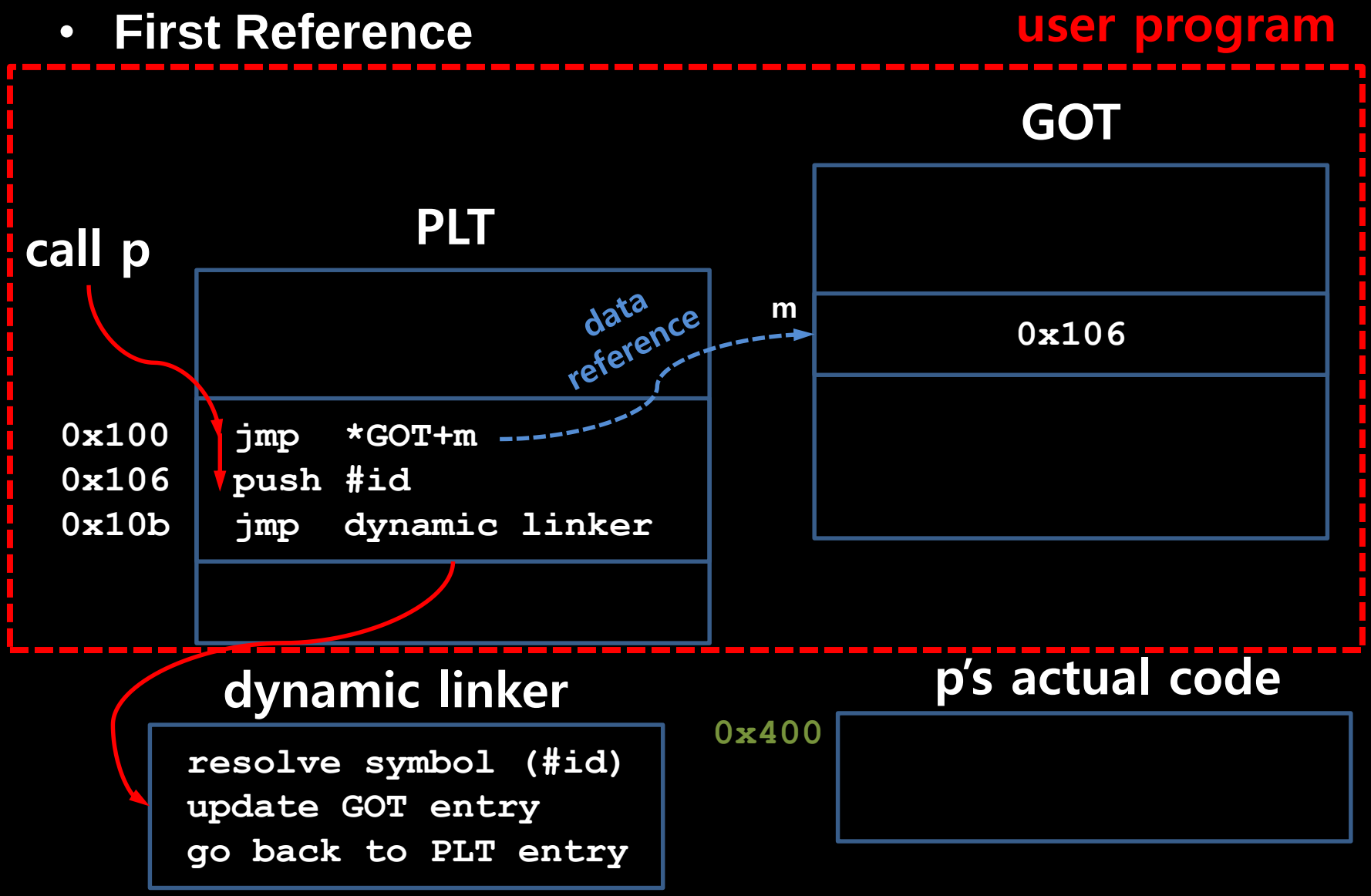
Dynamic Linking

- First Reference



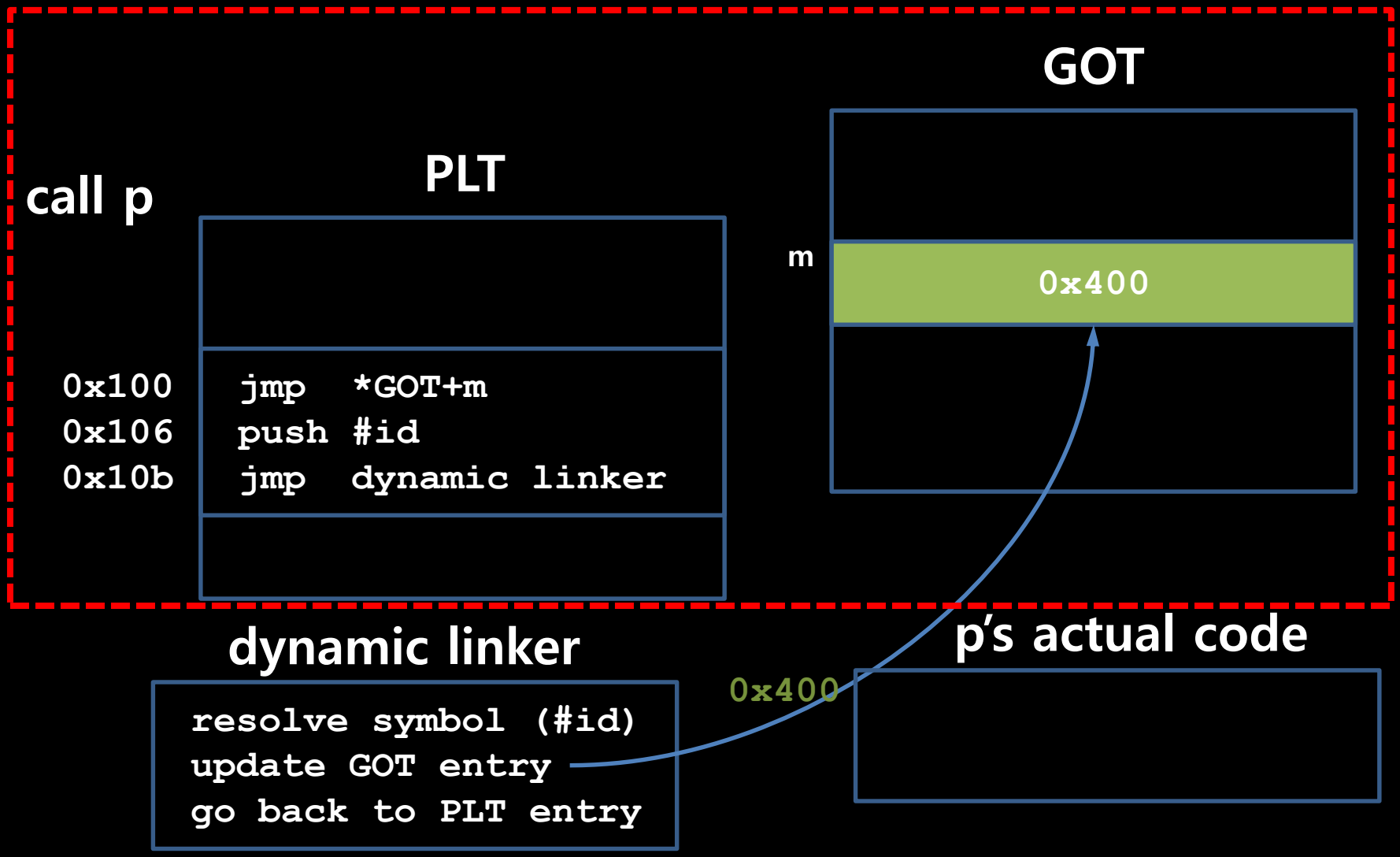
Dynamic Linking

- First Reference



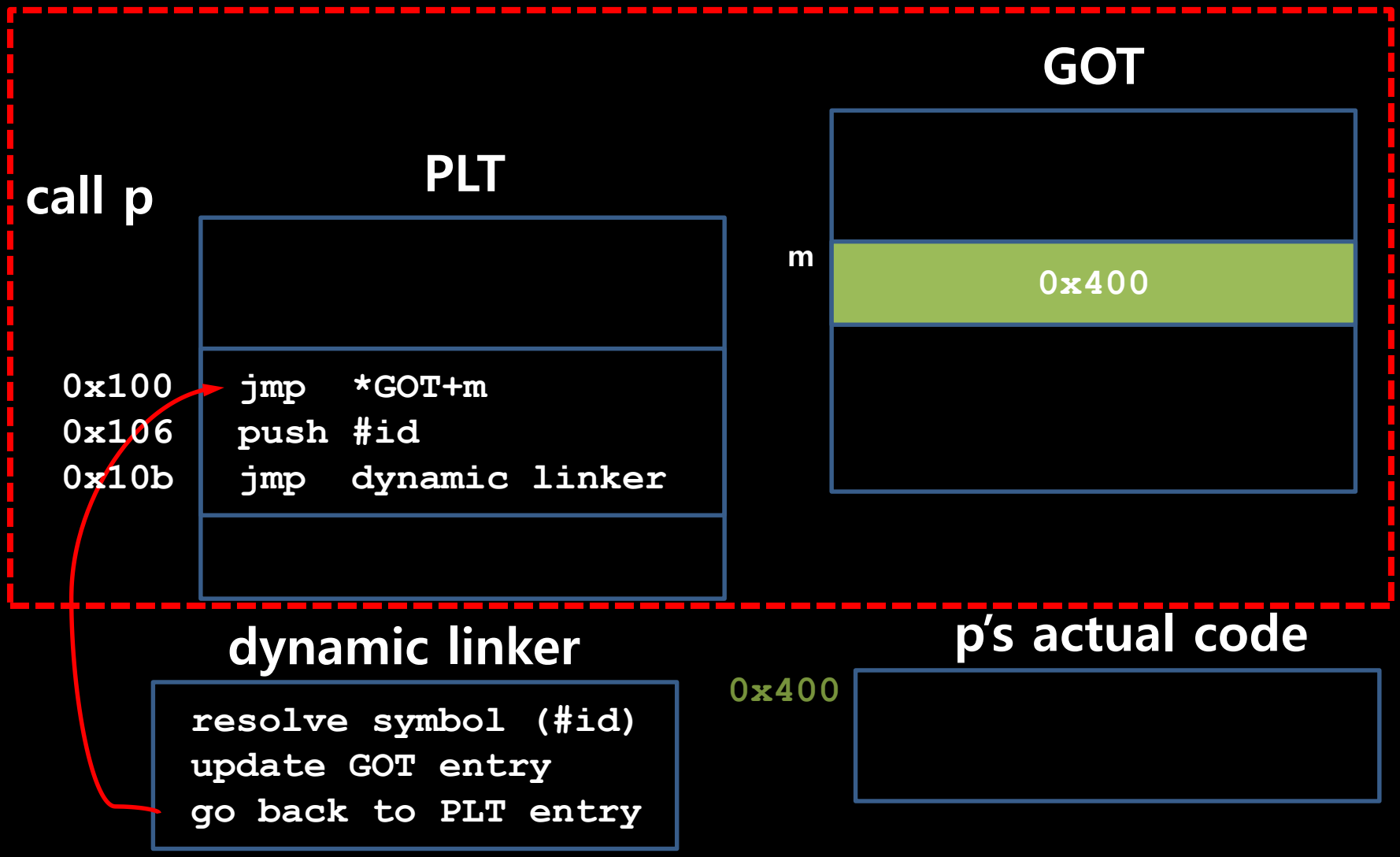
Dynamic Linking

- First Reference



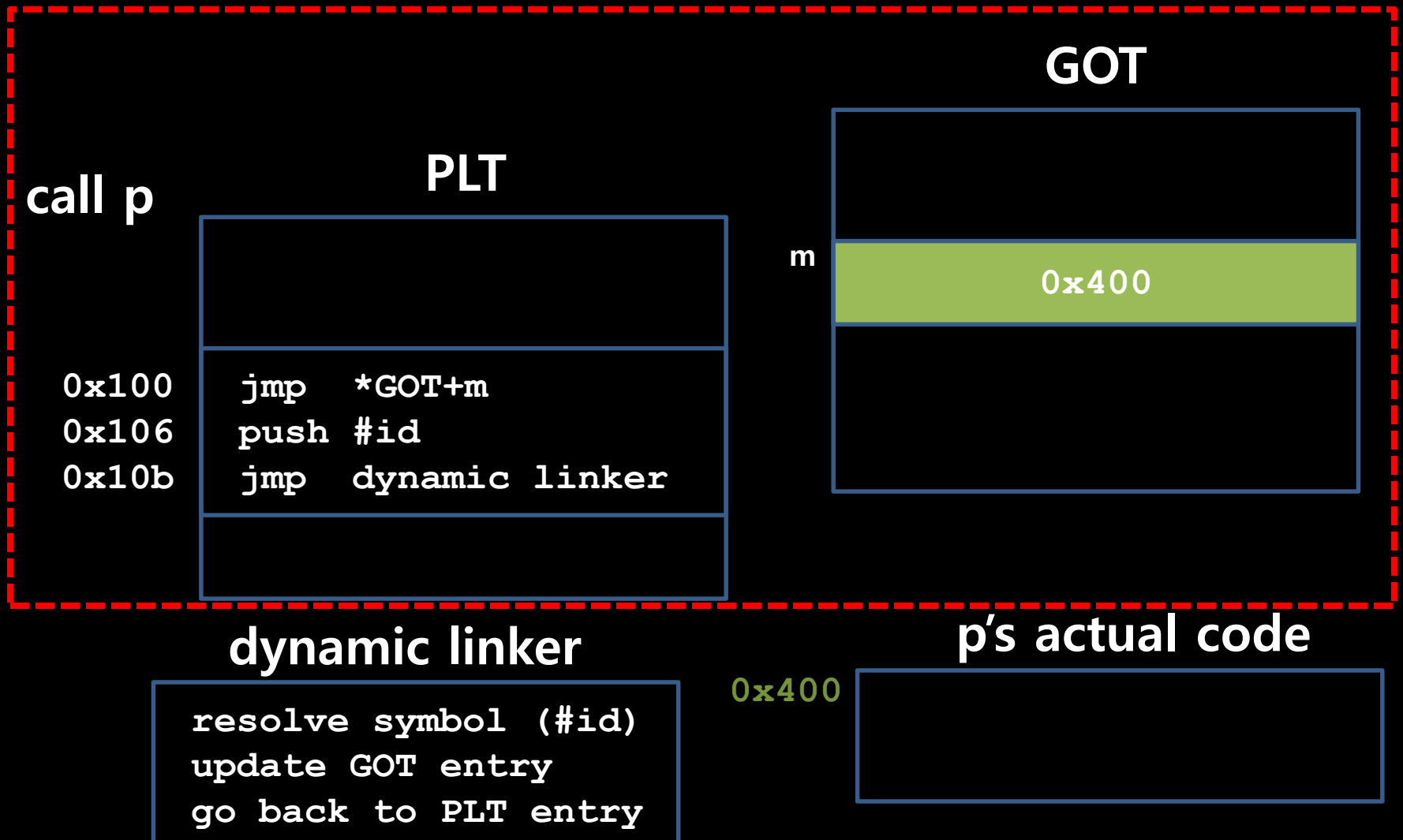
Dynamic Linking

- First Reference



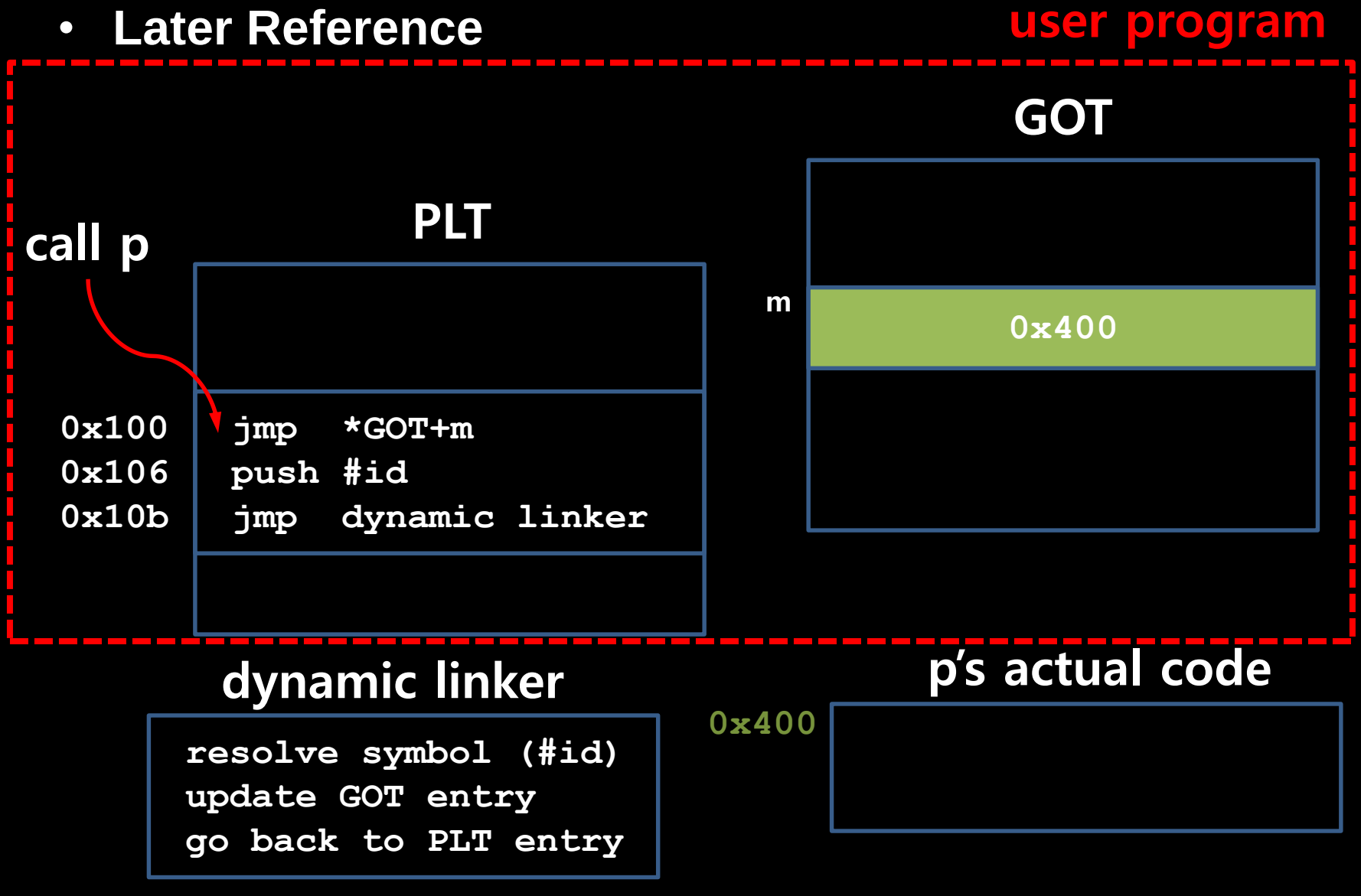
Dynamic Linking

- Later Reference



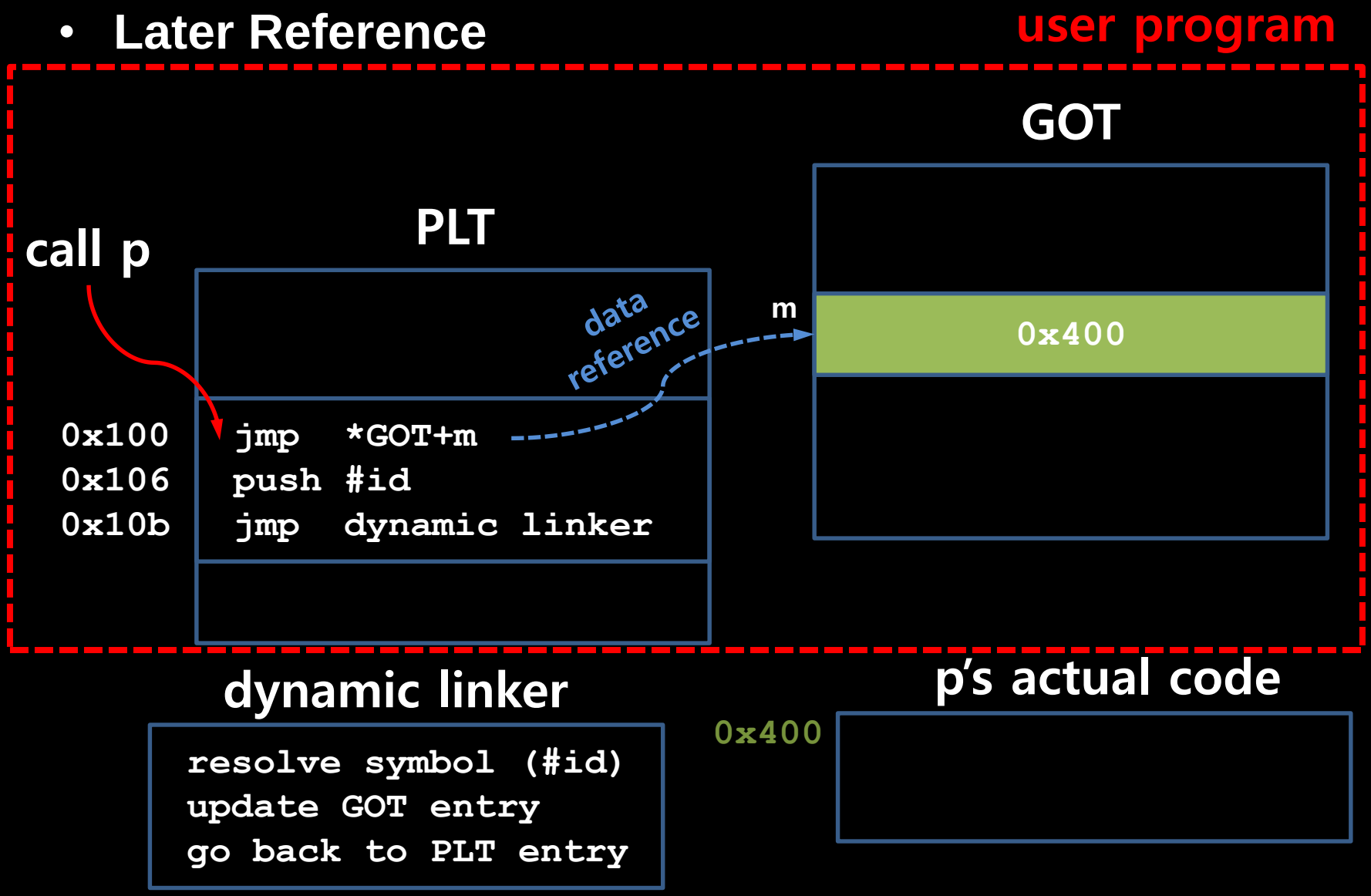
Dynamic Linking

- Later Reference



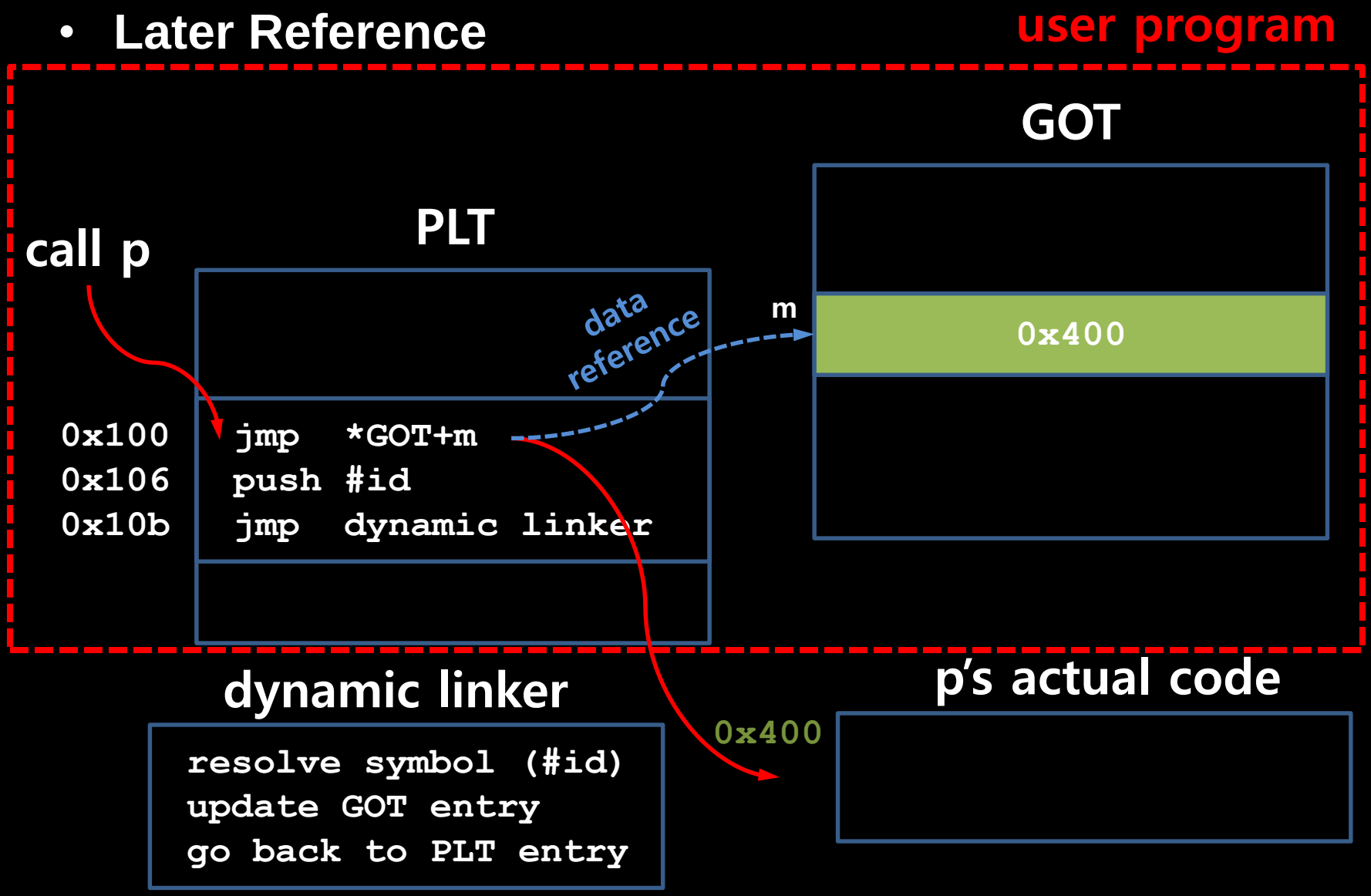
Dynamic Linking

- Later Reference



Dynamic Linking

- Later Reference



```
/*
 * mcount_arch_plthook_addr() returns the address of GOT entry.
 * The initial value for each GOT entry redirects the execution to
 * the runtime resolver. (_dl_runtime_resolve in ld-linux.so)
 *
 * The GOT entry is updated by the runtime resolver to the resolved address of
 * the target library function for later reference.
 *
 * However, uftrace gets this address to update it back to the initial value.
 * Even if the GOT entry is resolved by runtime resolver, uftrace restores the
 * address back to the initial value to watch library function calls.
 *
 * Before doing this work, GOT[2] is updated from the address of runtime
 * resolver(_dl_runtime_resolve) to uftrace hooking routine(plt_hooker).
 *
 * This address depends on the PLT structure of each architecture so this
 * function is implemented differently for each architecture.
 */
__weak unsigned long mcount_arch_plthook_addr(struct plthook_data *pd, int idx)
{
        struct sym *sym;

        sym = &pd->dsymtab.sym[idx];
        return sym->addr + ARCH_PLTHOOK_ADDR_OFFSET;
}
```

https://github.com/namhyung/uftrace/commit/4b283f7430d7c817cdf6d2999dc5a1608293c544

```
static int find_got(struct uftrace_elf_data *elf,
                    struct uftrace_elf_iter *iter,
                    const char *modname,
                    unsigned long offset)
{
        bool plt_found = false;
        unsigned long pltgot_addr = 0;
        unsigned long plt_addr = 0;
        struct plthook_data *pd;
                ...
        pd = xmalloc(sizeof(*pd));
        pd->mod_name   = xstrdup(modname);
        pd->pltgot_ptr = (void *)pltgot_addr;
        pd->module_id  = pd->pltgot_ptr[1];
        pd->base_addr  = offset;
        pd->plt_addr   = plt_addr;
                ...
        pd->resolved_addr = xcalloc(pd->dsymtab.nr_sym, sizeof(long));
        pd->special_funcs = NULL;
        pd->nr_special    = 0;

        mcount_arch_plthook_setup(pd, elf);
        list_add_tail(&pd->list, &plthook_modules);
                ...
        overwrite_pltgot(pd, 2, plt_hooker);
                ...
        return 0;
}
```

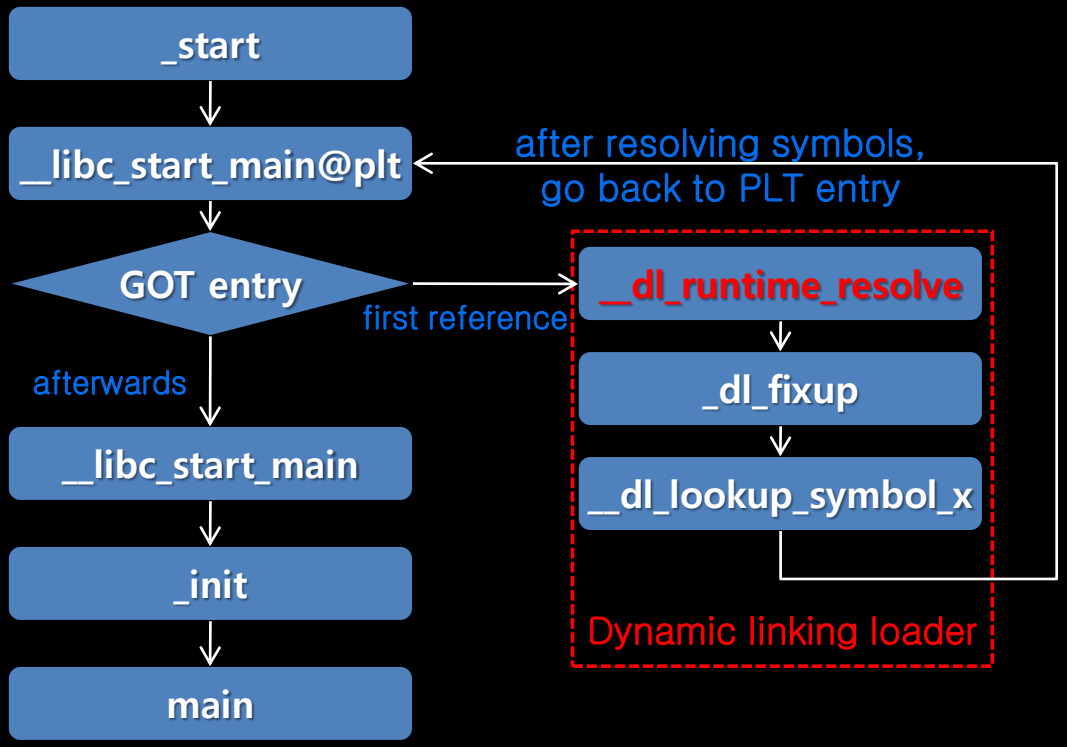
uftrace/libmcount/plthook.c

```
static unsigned long __plthook_entry(unsigned long *ret_addr,
                                     unsigned long child_idx,
                                     unsigned long module_id,
                                     struct mcount_regs *regs)
{
                ...
        rstack = &mtdp->rstack[mtdp->idx++];

        rstack->depth      = mtdp->record_idx;
        rstack->pd         = pd;
        rstack->dyn_idx    = child_idx;
        rstack->parent_loc = ret_addr;
        rstack->parent_ip  = *ret_addr;
        rstack->child_ip   = sym->addr;
        rstack->start_time = skip ? 0 : mcount_gettime();
        rstack->end_time   = 0;
        rstack->flags      = skip ? MCOUNT_FL_NORECORD : 0;
        rstack->nr_events  = 0;
        rstack->event_idx  = ARGBUF_SIZE;

        /* hijack the return address of child */
        *ret_addr = (unsigned long)plthook_return;

        /* restore return address of parent */
        if (mcount_auto_recover)
                mcount_auto_restore(mtdp);
}
```

uftrace/libmcount/plthook.c

Dynamic Relocation (cont.)

- Procedure of entering C main function in Linux



Nested Library Tracing

```
$ uftrace --nest-libcall --auto-args \
        /usr/bin/clang hello.c
```

```
-l, --nest-libcall
  Trace function calls between libraries.
  By default, uftrace only record library call from
  the main executable.
```

라이브러리 내부에서 다른 라이브러리를
호출하는 것도 tracing 해야 할 때 사용

Nested Library Tracing

```
$ uftrace -l -a /usr/bin/clang hello.c
```

```
-l, --nest-libcall
  Trace function calls between libraries.
  By default, uftrace only record library call from
  the main executable.
```

라이브러리 내부에서 다른 라이브러리를
호출하는 것도 tracing 해야 할 때 사용

Nested Library Tracing

```
$ uftrace -la /usr/bin/clang hello.c
```

```
-l, --nest-libcall
  Trace function calls between libraries.
  By default, uftrace only record library call from
  the main executable.
```

라이브러리 내부에서 다른 라이브러리를
호출하는 것도 tracing 해야 할 때 사용

Nested Library Tracing

```
  0.284 us [175968] | strlen("/usr/bin/ld") = 11;
         [175968] | llvm::sys::commandLineFitsWithinSystemLimits() {
        ...
 21.584 us [175968] | } /* llvm::sys::commandLineFitsWithinSystemLimits */
  0.197 us [175968] | llvm::opt::ArgList::getLastArg();
  0.420 us [175968] | memcpy(0x7ffc7ba7a020, 0x28a07d0, 384) = 0x7ffc7ba7a020;
  0.323 us [175968] | strlen("/usr/lib/llvm-3.8/bin/clang") = 27;
         [175968] | llvm::sys::ExecuteAndWait() {
  0.360 us [175968] |   memcpy(0x7ffc7ba79b18, 0x2883dc0, 27) = 0x7ffc7ba79b18;
  3.093 us [175968] |   access();
  0.153 us [175968] |   std::_V2::system_category();
         [175968] |   std::__cxx11::basic_string::_M_create() {
         [175968] |     operator new() {
  0.490 us [175968] |       malloc(28) = 0x28a1150;
  1.053 us [175968] |     } /* operator new */
  1.566 us [175968] |   } /* std::__cxx11::basic_string::_M_create */
  0.253 us [175968] |   memcpy(0x28a1150, 0x2883dc0, 27) = 0x28a1150;
247.286 us [175968] |   posix_spawn();
         [175968] |   operator delete() {
  0.590 us [175968] |     free(0x28a1150);
  1.500 us [175968] |   } /* operator delete */
         [175968] |   waitpid(175980, 0x7ffc7ba79bfc, 0) {
        ...
```

재컴파일 없이 시스템에 배포된 clang 이미지에 적용해도
일부 라이브러리 함수 호출 확인 가능

Linux Kernel Function Tracing

```
$ gcc -pg hello.c
```

Linux Kernel Function Tracing

```
$ gcc -pg hello.c

$ sudo uftrace -k a.out
Hello World!
```

Linux Kernel Function Tracing

```
$ gcc -pg hello.c

$ sudo uftrace -k a.out
Hello World!
# DURATION    TID     FUNCTION
   0.395 us [ 8926] | __monstartup();
   0.354 us [ 8926] | __cxa_atexit();
            [ 8926] | main() {
            [ 8926] |   printf() {
   0.572 us [ 8926] |     sys_newfstat();
   1.316 us [ 8926] |     __do_page_fault();
   4.123 us [ 8926] |   } /* puts */
            [ 8926] |   fflush() {
   5.229 us [ 8926] |     sys_write();
   6.454 us [ 8926] |   } /* fflush */
  11.171 us [ 8926] | } /* main */
```

리눅스 커널
내부 함수들

Event Tracing (sched event)

```
$ uftrace t-fork
# DURATION    TID     FUNCTION
            [14983] | main() {
 225.620 us [14983] |   fork();
            [14983] |   wait() {
            [14983] |     /* linux:sched-out */
            [14995] |   } /* fork */
            [14995] |   a() {
            [14995] |     b() {
            [14995] |       c() {
   1.033 us [14995] |         getpid();
   2.280 us [14995] |       } /* c */
   2.677 us [14995] |     } /* b */
   3.020 us [14995] |   } /* a */
  11.131 us [14995] | } /* main */
 676.988 us [14983] |     /* linux:sched-in */
 695.312 us [14983] |   } /* wait */
            [14983] |   a() {
            [14983] |     b() {
            [14983] |       c() {
   2.067 us [14983] |         getpid();
   3.067 us [14983] |       } /* c */
   3.444 us [14983] |     } /* b */
   3.841 us [14983] |   } /* a */
 950.334 us [14983] | } /* main */
```

Event Tracing (sched event)

```
$ uftrace --no-event t-fork
# DURATION    TID     FUNCTION
            [14983] | main() {
 225.620 us [14983] |   fork();
            [14983] |   wait() {

            [14995] |   } /* fork */
            [14995] |   a() {
            [14995] |     b() {
            [14995] |       c() {
   1.033 us [14995] |         getpid();
   2.280 us [14995] |       } /* c */
   2.677 us [14995] |     } /* b */
   3.020 us [14995] |   } /* a */
  11.131 us [14995] | } /* main */

 695.312 us [14983] |   } /* wait */
            [14983] |   a() {
            [14983] |     b() {
            [14983] |       c() {
   2.067 us [14983] |         getpid();
   3.067 us [14983] |       } /* c */
   3.444 us [14983] |     } /* b */
   3.841 us [14983] |   } /* a */
 950.334 us [14983] | } /* main */
```

Event Recording

- perf_event_paranoid 값에 의해 읽지 못하는 event 를 읽을 수 없는 경우

- 아래와 같이 값이 3인 경우 권한 없음

```
$ cat /proc/sys/kernel/perf_event_paranoid
3
```

- 값을 아래와 같이 1로 설정해야 함

```
$ echo 1 | sudo tee /proc/sys/kernel/perf_event_paranoid
1
```

man pages

- **uftrace 의 세부 옵션 설명은 manual 페이지에서 참고 가능**
  - man uftrace
  - man uftrace record
  - man uftrace replay
  - man uftrace live
  - man uftrace report
  - man uftrace dump
  - man uftrace graph
  - man uftrace script
  - man uftrace tui
  - man uftrace info
  - man uftrace recv

Architecture 지원 현황

- **현재 지원 가능한 Architecture**
  - x86_64
    - 주요 테스트 및 개발 환경으로 대부분의 주요 기능 동작.
    - chrome 과 같은 대규모 프로그램을 대상으로도 record 가능.
    - -pg 와 같은 재컴파일 없이 내부 disassemble 엔진으로 직접적인 live patching 후 tracing 가능
  - i386
    - 테스트가 부족하지만 아직 특별한 문제는 발견되지 않음
    - 현재는 사용 빈도가 상당히 낮은 환경
  - ARM / AArch64
    - 많은 테스트 결과 라즈베리파이 환경에서는 잘 동작하나 컴파일러 및 링커 설정에 따라 일부 버그 존재.
    - 주요 임베디드 시스템에서 사용되는 환경으로 중요도 높음.

C++ Case Study with uftrace

uftrace 를 활용한 C++ 내부에 대한 분석

std::unique_ptr

new / delete 와 성능 차이가 있을까?

(Zero-Cost Abstractions)

new and delete

```
$ cat new-delete.cc
int main()
{
    int* p = new int;
    delete p;
}
```

new and delete

```
$ cat new-delete.cc
int main()
{
    int* p = new int;
    delete p;
}

$ g++ -pg -g -O2 new-delete.cc
```

new and delete

```
$ cat new-delete.cc
int main()
{
    int* p = new int;
    delete p;
}

$ g++ -pg -g -O2 new-delete.cc

$ uftrace -a --nest-libcall a.out
# DURATION     TID     FUNCTION
            [165267] | main() {
            [165267] |   operator new(4) {
   0.976 us [165267] |     malloc(4) = 0x21ceaa0;
   7.847 us [165267] |   } = 0x21ceaa0; /* operator new */
            [165267] |   operator delete(0x21ceaa0) {
   1.807 us [165267] |     free(0x21ceaa0);
   5.261 us [165267] |   } /* operator delete */
  18.154 us [165267] | } = 0; /* main */
```

std::unique_ptr

```
$ cat unique_ptr.cc
#include <memory>
int main()
{
    std::unique_ptr<int> p(new int);
}
```

std::unique_ptr

```
$ cat unique_ptr.cc
#include <memory>
int main()
{
    std::unique_ptr<int> p(new int);
}

$ g++ -std=c++11 -pg -g -O2 unique_ptr.cc
```

std::unique_ptr

```
$ cat unique_ptr.cc
#include <memory>
int main()
{
    std::unique_ptr<int> p(new int);
}

$ g++ -std=c++11 -pg -g -O2 unique_ptr.cc

$ uftrace -a --nest-libcall a.out
# DURATION     TID     FUNCTION
            [165318] | main() {
            [165318] |   operator new(4) {
   1.076 us [165318] |     malloc(4) = 0x1d7b290;
   7.824 us [165318] |   } = 0x1d7b290; /* operator new */
            [165318] |   operator delete(0x1d7b290) {
   1.716 us [165318] |     free(0x1d7b290);
   5.097 us [165318] |   } /* operator delete */
  18.260 us [165318] | } = 0; /* main */
```

std::unique_ptr

```
$ cat unique_ptr.cc
#include <memory>
int main()
{
    std::unique_ptr<int> p(new int);
}

$ g++ -std=c++11 -pg -g -O2 unique_ptr.cc

$ uftrace -a --nest-libcall a.out
# DURATION     TID     FUNCTION
            [165318] | main() {
            [165318] |   operator new(4) {
   1.076 us [165318] |     malloc(4) = 0x1d7b290;
   7.824 us [165318] |   } = 0x1d7b290; /* operator new */
            [165318] |   operator delete(0x1d7b290) {
   1.716 us [165318] |     free(0x1d7b290);
   5.097 us [165318] |   } /* operator delete */
  18.260 us [165318] | } = 0; /* main */
```

new 와 delete 를 직접 사용한 경우와 동일함

```
# DURATION     TID     FUNCTION
            [168276] | main() {
            [168276] |   operator new(4) {
   1.094 us [168276] |     malloc(4) = 0xbdbb40;
   8.114 us [168276] |   } = 0xbdbb40; /* operator new */
            [168276] |   std::unique_ptr::unique_ptr(0x7ffded6fd110, 0xbdbb40) {
            [168276] |     std::tuple::tuple(0x7ffded6fd110) {
            [168276] |       std::_Tuple_impl::_Tuple_impl(0x7ffded6fd110) {
            [168276] |         std::_Tuple_impl::_Tuple_impl(0x7ffded6fd110) {
            [168276] |           std::_Head_base::_Head_base(0x7ffded6fd110) {
   0.266 us [168276] |             std::default_delete::default_delete(0x7ffded6fd110);
   1.674 us [168276] |           } /* std::_Head_base::_Head_base */
   6.320 us [168276] |         } /* std::_Tuple_impl::_Tuple_impl */
   0.227 us [168276] |         std::_Head_base::_Head_base(0x7ffded6fd110);
   7.850 us [168276] |       } /* std::_Tuple_impl::_Tuple_impl */
   8.500 us [168276] |     } /* std::tuple::tuple */
            [168276] |     std::get(0x7ffded6fd110) {
            [168276] |       std::__get_helper(0x7ffded6fd110) {
            [168276] |         std::_Tuple_impl::_M_head(0x7ffded6fd110) {
   0.226 us [168276] |           std::_Head_base::_M_head(0x7ffded6fd110) = 0x7ffded6fd110;
   1.290 us [168276] |         } = 0x7ffded6fd110; /* std::_Tuple_impl::_M_head */
   2.113 us [168276] |       } = 0x7ffded6fd110; /* std::__get_helper */
   2.816 us [168276] |     } = 0x7ffded6fd110; /* std::get */
  12.870 us [168276] |   } /* std::unique_ptr::unique_ptr */
            [168276] |   std::unique_ptr::~unique_ptr(0x7ffded6fd110) {
            [168276] |     std::get(0x7ffded6fd110) {
            [168276] |       std::__get_helper(0x7ffded6fd110) {
            [168276] |         std::_Tuple_impl::_M_head(0x7ffded6fd110) {
   0.210 us [168276] |           std::_Head_base::_M_head(0x7ffded6fd110) = 0x7ffded6fd110;
   1.093 us [168276] |         } = 0x7ffded6fd110; /* std::_Tuple_impl::_M_head */
   1.646 us [168276] |       } = 0x7ffded6fd110; /* std::__get_helper */
   2.220 us [168276] |     } = 0x7ffded6fd110; /* std::get */
            [168276] |     std::unique_ptr::get_deleter(0x7ffded6fd110) {
            [168276] |       std::get(0x7ffded6fd110) {
            [168276] |         std::__get_helper(0x7ffded6fd110) {
            [168276] |           std::_Tuple_impl::_M_head(0x7ffded6fd110) {
   0.213 us [168276] |             std::_Head_base::_M_head(0x7ffded6fd110) = 0x7ffded6fd110;
   1.094 us [168276] |           } = 0x7ffded6fd110; /* std::_Tuple_impl::_M_head */
   1.747 us [168276] |         } = 0x7ffded6fd110; /* std::__get_helper */
   2.391 us [168276] |       } = 0x7ffded6fd110; /* std::get */
   3.223 us [168276] |     } = 0x7ffded6fd110; /* std::unique_ptr::get_deleter */
            [168276] |     std::default_delete::operator()(0x7ffded6fd110, 0xbdbb40) {
            [168276] |       operator delete(0xbdbb40) {
   1.810 us [168276] |         free(0xbdbb40);
   5.230 us [168276] |       } /* operator delete */
   6.213 us [168276] |     } /* std::default_delete::operator() */
  13.271 us [168276] |   } /* std::unique_ptr::~unique_ptr */
  40.624 us [168276] | } = 0; /* main */
```

```
# DURATION     TID     FUNCTION
            [168276] | main() {
            [168276] |   operator new(4) {
   1.094 us [168276] |     malloc(4) = 0xbdbb40;
   8.114 us [168276] |   } = 0xbdbb40; /* operator new */
            [168276] |   std::unique_ptr::unique_ptr(0x7ffded6fd110, 0xbdbb40) {
            [168276] |     std::tuple::tuple(0x7ffded6fd110) {
            [168276] |       std::_Tuple_impl::_Tuple_impl(0x7ffded6fd110) {
            [168276] |         std::_Tuple_impl::_Tuple_impl(0x7ffded6fd110) {
            [168276] |           std::_Head_base::_Head_base(0x7ffded6fd110) {
   0.266 us [168276] |             std::default_delete::default_delete(0x7ffded6fd110);
   1.674 us [168276] |           } /* std::_Head_base::_Head_base */
   6.320 us [168276] |         } /* std::_Tuple_impl::_Tuple_impl */
   0.227 us [168276] |         std::_Head_base::_Head_base(0x7ffded6fd110);
   7.850 us [168276] |       } /* std::_Tuple_impl::_Tuple_impl */
   8.500 us [168276] |     } /* std::tuple::tuple */
            [168276] |     std::get(0x7ffded6fd110) {
            [168276] |       std::__get_helper(0x7ffded6fd110) {
            [168276] |         std::_Tuple_impl::_M_head(0x7ffded6fd110) {
   0.226 us [168276] |           std::_Head_base::_M_head(0x7ffded6fd110) = 0x7ffded6fd110;
   1.290 us [168276] |         } = 0x7ffded6fd110; /* std::_Tuple_impl::_M_head */
   2.113 us [168276] |       } = 0x7ffded6fd110; /* std::__get_helper */
   2.816 us [168276] |     } = 0x7ffded6fd110; /* std::get */
  12.870 us [168276] |   } /* std::unique_ptr::unique_ptr */
            [168276] |   std::unique_ptr::~unique_ptr(0x7ffded6fd110) {
            [168276] |     std::get(0x7ffded6fd110) {
            [168276] |       std::__get_helper(0x7ffded6fd110) {
            [168276] |         std::_Tuple_impl::_M_head(0x7ffded6fd110) {
   0.210 us [168276] |           std::_Head_base::_M_head(0x7ffded6fd110) = 0x7ffded6fd110;
   1.093 us [168276] |         } = 0x7ffded6fd110; /* std::_Tuple_impl::_M_head */
   1.646 us [168276] |       } = 0x7ffded6fd110; /* std::__get_helper */
   2.220 us [168276] |     } = 0x7ffded6fd110; /* std::get */
            [168276] |     std::unique_ptr::get_deleter(0x7ffded6fd110) {
            [168276] |       std::get(0x7ffded6fd110) {
            [168276] |         std::__get_helper(0x7ffded6fd110) {
            [168276] |           std::_Tuple_impl::_M_head(0x7ffded6fd110) {
   0.213 us [168276] |             std::_Head_base::_M_head(0x7ffded6fd110) = 0x7ffded6fd110;
   1.094 us [168276] |           } = 0x7ffded6fd110; /* std::_Tuple_impl::_M_head */
   1.747 us [168276] |         } = 0x7ffded6fd110; /* std::__get_helper */
   2.391 us [168276] |       } = 0x7ffded6fd110; /* std::get */
   3.223 us [168276] |     } = 0x7ffded6fd110; /* std::unique_ptr::get_deleter */
            [168276] |     std::default_delete::operator()(0x7ffded6fd110, 0xbdbb40) {
            [168276] |       operator delete(0xbdbb40) {
   1.810 us [168276] |         free(0xbdbb40);
   5.230 us [168276] |       } /* operator delete */
   6.213 us [168276] |     } /* std::default_delete::operator() */
  13.271 us [168276] |   } /* std::unique_ptr::~unique_ptr */
  40.624 us [168276] | } = 0; /* main */
```

최적화를 안한 경우의
전체 내부 함수 호출

std::endl

std::endl 을 사용하지 말아야 하는 이유

```
$ cat endl.cc
#include <iostream>
#include <vector>
#include <cstdlib>
using namespace std;

int main(int argc, char* argv[])
{
    int iter = atoi(argv[1]);
    vector<int> v(iter, 0);

    for (int i = 0; i < v.size(); i++)
        cout << v[i] << endl;
}
```

```
$ cat no-endl.cc
#include <iostream>
#include <vector>
#include <cstdlib>
using namespace std;

int main(int argc, char* argv[])
{
    int iter = atoi(argv[1]);
    vector<int> v(iter, 0);

    for (int i = 0; i < v.size(); i++)
        cout << v[i] << '\n';
}
```

```
$ gcc -O2 -o endl endl.cc
$ time ./endl 50000000 > /dev/null

real    0m32.594s
user    0m19.433s
sys     0m13.161s


$ gcc -O2 -o no-endl no-endl.cc
$ time ./no-endl 50000000 > /dev/null

real    0m4.250s
user    0m4.138s
sys     0m0.112s
```

std::endl

- std::endl 은 내부에서 어떤 일을 하는가?

```
std::cout << std::endl;
```

std::endl

- std::endl 은 내부에서 어떤 일을 하는가?

```
std::cout << std::endl;
```

```
std::cout << '\n' << std::flush;
```

```
$ cat test-endl.cpp
#include <iostream>
#include <vector>

int main()
{
    std::vector<int> v = { 10, 20, 30 };
    for (auto a : v)
        std::cout << a << std::endl;
}


$ g++ -O2 -pg -o endl test-endl.cpp
```

```
$ uftrace endl
10
20
30
```

std::endl 호출시 반복된 flush 발생

```
$ uftrace endl
10
20
30
# DURATION     TID     FUNCTION
            [122233] | _GLOBAL__sub_I_main() {
 156.463 us [122233] |   std::ios_base::Init::Init();
   0.426 us [122233] |   __cxa_atexit();
 164.050 us [122233] | } /* _GLOBAL__sub_I_main */
            [122233] | main() {
   2.336 us [122233] |   operator new();
  21.507 us [122233] |   std::basic_ostream::operator<<();
  15.691 us [122233] |   std::basic_ostream::put();
   2.920 us [122233] |   std::basic_ostream::flush();
   0.907 us [122233] |   std::basic_ostream::operator<<();
   2.394 us [122233] |   std::basic_ostream::put();
   0.344 us [122233] |   std::basic_ostream::flush();
   0.313 us [122233] |   std::basic_ostream::operator<<();
   2.064 us [122233] |   std::basic_ostream::put();
   0.234 us [122233] |   std::basic_ostream::flush();
   3.050 us [122233] |   operator delete();
  58.105 us [122233] | } = 0; /* main */
   3.837 us [122233] | std::ios_base::Init::~Init();
```

```
$ cat test-endl.cpp
#include <iostream>
#include <vector>

int main()
{
    std::vector<int> v = { 10, 20, 30 };
    for (auto a : v)
        std::cout << a << '\n';
}


$ g++ -O2 -pg -o no-endl test-endl.cpp
```

```
$ uftrace no-endl
10
20
30
```

불필요한 flush 없음

```
$ uftrace no-endl
10
20
30
# DURATION     TID     FUNCTION
            [122346] | _GLOBAL__sub_I_main() {
 170.023 us [122346] |   std::ios_base::Init::Init();
   0.380 us [122346] |   __cxa_atexit();
 178.013 us [122346] | } /* _GLOBAL__sub_I_main */
            [122346] | main() {
   2.184 us [122346] |   operator new();
  14.537 us [122346] |   std::basic_ostream::operator<<();
  16.204 us [122346] |   std::__ostream_insert();
   0.720 us [122346] |   std::basic_ostream::operator<<();
   2.430 us [122346] |   std::__ostream_insert();
   0.353 us [122346] |   std::basic_ostream::operator<<();
   2.150 us [122346] |   std::__ostream_insert();
   3.170 us [122346] |   operator delete();
  46.781 us [122346] | } = 0; /* main */
   5.284 us [122346] | std::ios_base::Init::~Init();
```