Demystifying Linux Kernel initcalls

Mylène Josserand

mylene.josserand@collabora.com

Open First

Demystifying Linux Kernel initcalls

Introduction:

Purpose & debugging

initcalls

initcalls

- Implemented early in Linux Kernel – v2.4 ~ 2001

initcalls

- Implemented early in Linux Kernel – v2.4 ~ 2001
- No big changes since

initcalls

- Implemented early in Linux Kernel – v2.4 ~ 2001
- No big changes since
- 2018: Tracing support from Steven Rostedt

Purpose

- Call **functions** at different stages during **boot** process

Purpose

- Call **functions** at different stages during **boot** process
- **Helpers** to define the type used (i.e. levels)

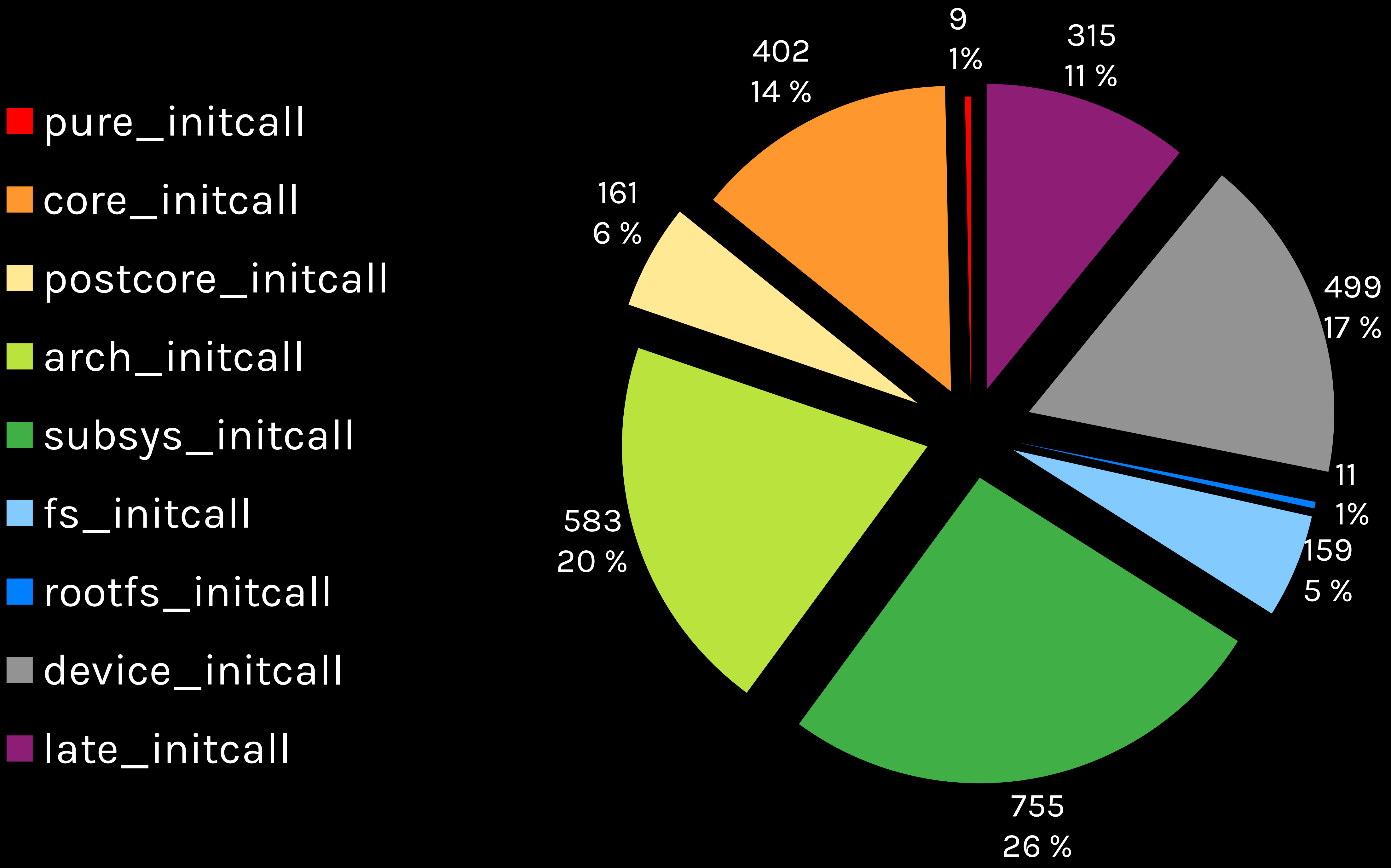
Purpose

- Call **functions** at different stages during **boot** process

- **Helpers** to define the type used (i.e. levels)
  - `pure_initcall`
  - `core_initcall`
  - `postcore_initcall`
  - `arch_initcall`
  - `subsys_initcall`
  - `fs_initcall`
  - `rootfs_initcall`
  - `device_initcall`
  - `late_initcall`

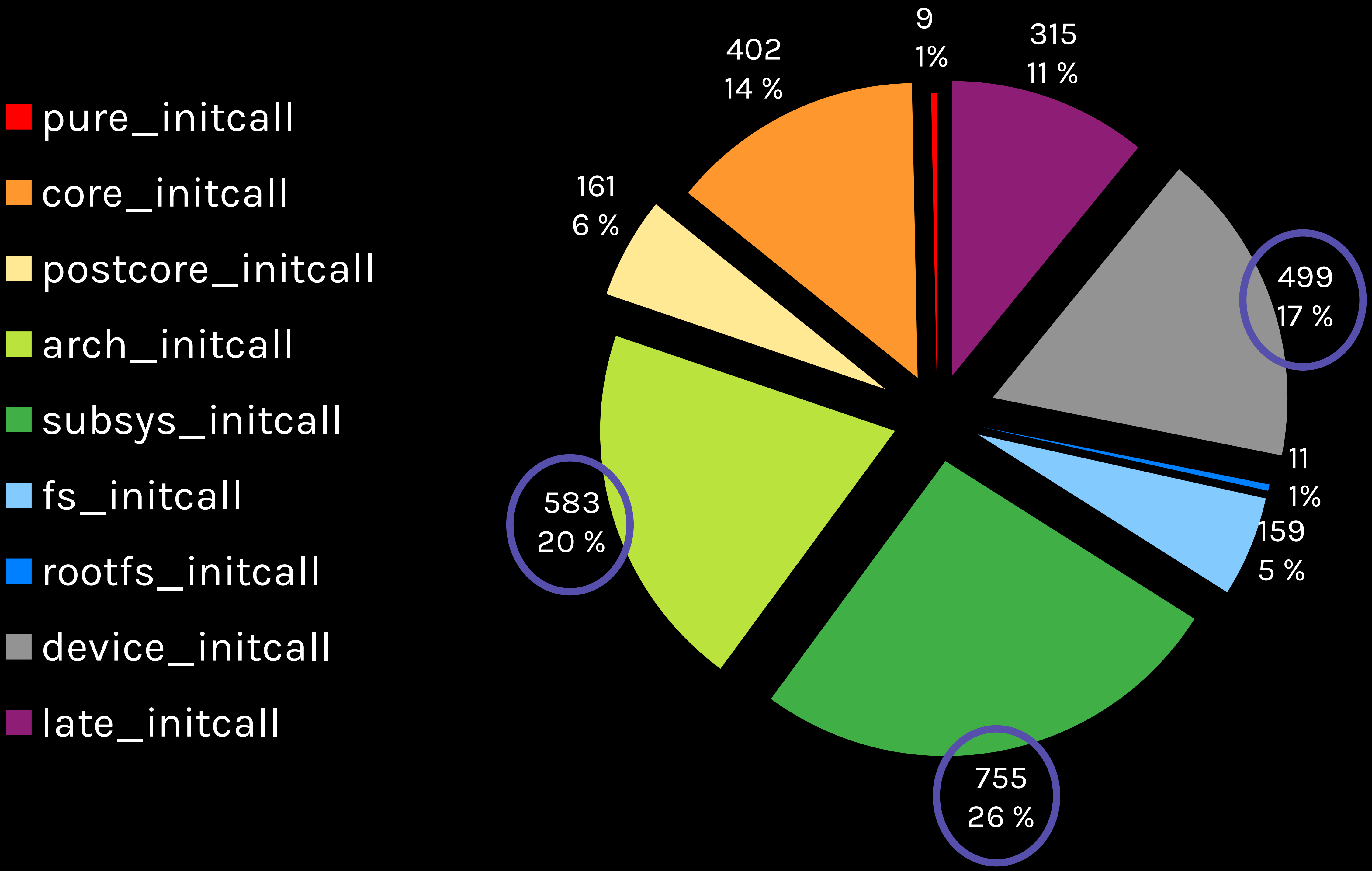
Distribution

In Linux Kernel v5.8



Distribution

In Linux Kernel v5.8



Example

Example

```
static int __init foo_init(void)
{
    return 0;
}
postcore_initcall(foo_init);
```

Example

```
static int __init foo_init(void)
{
    return 0;
}
postcore_initcall(foo_init);
```

- Executed at postcore stage

Example

```
static int __init foo_init(void)
{
    return 0;
}
postcore_initcall(foo_init);
```

- Executed at postcore stage
- It is like marking the execution of a function at a specific level

Example

```
static int __init foo_init(void)
{
    return 0;
}
postcore_initcall(foo_init);
```

- Executed at postcore stage
- It is like marking the execution of a function at a specific level
- Names of helpers reflect the order of the execution

Debugging

- Introduced in 2.5.67

Debugging

- Introduced in 2.5.67
- `initcall_debug` in command-line

Debugging

- Introduced in 2.5.67
- `initcall_debug` in command-line

```
[...]
[    0.040357] calling  net_ns_init+0x0/0x140 @ 1
[    0.040635] initcall net_ns_init+0x0/0x140 returned 0 after 0 usecs
[    0.040740] calling  vfp_init+0x0/0x1d0 @ 1
[    0.040777] initcall vfp_init+0x0/0x1d0 returned 0 after 0 usecs
[    0.040786] calling  ptrace_break_init+0x0/0x2c @ 1
[    0.040796] initcall ptrace_break_init+0x0/0x2c returned 0 after 0 usecs
[    0.040808] calling  register_cpufreq_notifier+0x0/0x10 @ 1
[    0.040819] initcall register_cpufreq_notifier+0x0/0x10 returned 0
                                                             after 0 usecs
[...]
```

Debugging

- Increase the boot time during the debug

Debugging

- Increase the boot time during the debug
- Difficult to retrieve specific data

Debugging

- Increase the boot time during the debug
- Difficult to retrieve specific data
- Ftrace support introduced by S. Rostedt

Debugging

- Increase the boot time during the debug
- Difficult to retrieve specific data
- Ftrace support introduced by S. Rostedt

```
# mount -t debugfs nodev /sys/kernel/debug
```

Debugging

- Increase the boot time during the debug
- Difficult to retrieve specific data
- Ftrace support introduced by S. Rostedt

```
# mount -t debugfs nodev /sys/kernel/debug

# cat /sys/kernel/debug/tracing/available_events | grep initcall
initcall:initcall_finish
initcall:initcall_start
initcall:initcall_level
```

Debugging

- Increase the boot time during the debug
- Difficult to retrieve specific data
- Ftrace support introduced by S. Rostedt

```
# mount -t debugfs nodev /sys/kernel/debug

# cat /sys/kernel/debug/tracing/available_events | grep initcall
initcall:initcall_finish
initcall:initcall_start
initcall:initcall_level

# cat /proc/cmdline
console=ttyS0,115200 earlyprintk root=/dev/mmcblk0p2 rootwait \
trace_event=initcall:initcall_level,initcall:initcall_start,
           initcall:initcall_finish
```

Debugging

```
# cat /sys/kernel/debug/tracing/trace
# tracer: nop
#
# entries-in-buffer/entries-written: 1090/1090   #P:4
#
#                       _-----=> irqs-off
#                      / _----=> need-resched
#                     | / _---=> hardirq/softirq
#                     || / _--=> preempt-depth
#                     ||| /     delay
#    TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#       | |       |   ||||       |         |
   <idle>-0     [000] ....   0.000125: initcall_level:level=console
   <idle>-0     [000] ....   0.000136: initcall_start:func=con_init+0x0/0x220
   <idle>-0     [000] ....   0.000232: initcall_finish:func=con_init+0x0/0x220 ret=0
   <idle>-0     [000] ....   0.000235: initcall_start:func=univ8250_console_init+0x0/0x3c
   <idle>-0     [000] ....   0.000246: initcall_finish:func=univ8250_console_init+0x0/0x3c
                                                                                ret=0
swapper/0-1     [000] ....   0.002016: initcall_level:level=early
swapper/0-1     [000] ....   0.002026: initcall_start:func=trace_init_flags_sys_exit+0x0/0x24
swapper/0-1     [000] ....   0.002029: initcall_finish:func=trace_init_flags_sys_exit+0x0/0x24
                                                                                         ret=0
[...]
```

Demystifying Linux Kernel initcalls

Implementation

- ☐ General
- ☐ Ordering
  - ☐ For a particular level
  - ☐ Between all initcalls
- ☐ Execution
- ☐ Modules

Implementation

- Disclaimer:
  - ELF understanding
  - Not an expert!

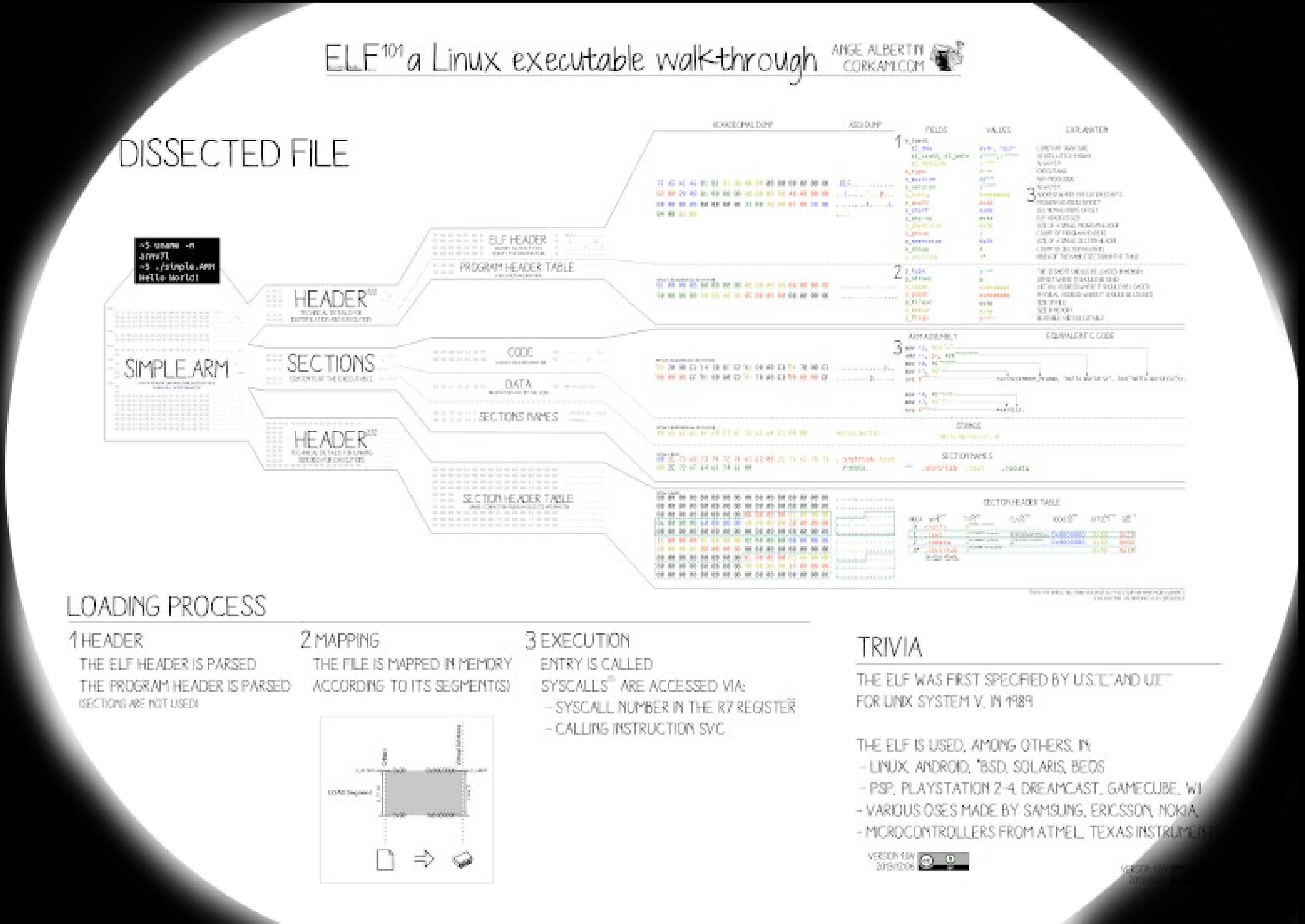
Implementation

- Disclaimer:
  - ELF understanding
  - Not an expert!

- Interesting resources
  - Wikipedia about ELF
  - Kernel-newbies article
  - corkami.github.io/

Implementation

- Disclaimer:
  - ELF understanding
  - Not an expert!
- Interesting resources
  - [Wikipedia about ELF]
  - [Kernel-newbies article]
  - [corkami.github.io/]


ELF101 a Linux executable walk-through
ANGE ALBERTINI
CORKAMI.COM
DISSECTED FILE
~$ uname -m
armv7l
~$ ./simple.ARM
Hello World!
SIMPLE.ARM
HEADER
SECTIONS
HEADER
ELF HEADER
PROGRAM HEADER TABLE
CODE
DATA
SECTION NAMES
SECTION HEADER TABLE
LOADING PROCESS
1 HEADER
THE ELF HEADER IS PARSED
THE PROGRAM HEADER IS PARSED
(SECTIONS ARE NOT USED)
2 MAPPING
THE FILE IS MAPPED IN MEMORY
ACCORDING TO ITS SEGMENT(S)
3 EXECUTION
ENTRY IS CALLED
SYSCALLS ARE ACCESSED VIA:
- SYSCALL NUMBER IN THE R7 REGISTER
- CALLING INSTRUCTION SVC
TRIVIA
THE ELF WAS FIRST SPECIFIED BY U.S.L. AND UI
FOR UNIX SYSTEM V. IN 1989
THE ELF IS USED, AMONG OTHERS, IN:
- LINUX, ANDROID, *BSD, SOLARIS, BEOS
- PSP, PLAYSTATION 2-4, DREAMCAST, GAMECUBE, WII
- VARIOUS OSES MADE BY SAMSUNG, ERICSSON, NOKIA
- MICROCONTROLLERS FROM ATMEL, TEXAS INSTRUMENTS

Implementation

- `include/linux/init.h`

Implementation

- include/linux/init.h

```
#define pure_initcall(fn)      __define_initcall(fn, 0)
#define core_initcall(fn)      __define_initcall(fn, 1)
#define postcore_initcall(fn)  __define_initcall(fn, 2)
#define arch_initcall(fn)      __define_initcall(fn, 3)
#define subsys_initcall(fn)    __define_initcall(fn, 4)
#define fs_initcall(fn)        __define_initcall(fn, 5)
#define rootfs_initcall(fn)    __define_initcall(fn, rootfs)
#define device_initcall(fn)    __define_initcall(fn, 6)
#define late_initcall(fn)      __define_initcall(fn, 7)
```

Implementation

- include/linux/init.h

```
#define pure_initcall(fn)        __define_initcall(fn, 0)
#define core_initcall(fn)        __define_initcall(fn, 1)
#define postcore_initcall(fn)    __define_initcall(fn, 2)
#define arch_initcall(fn)        __define_initcall(fn, 3)
#define subsys_initcall(fn)      __define_initcall(fn, 4)
#define fs_initcall(fn)          __define_initcall(fn, 5)
#define rootfs_initcall(fn)      __define_initcall(fn, rootfs)
#define device_initcall(fn)      __define_initcall(fn, 6)
#define late_initcall(fn)        __define_initcall(fn, 7)
```

- `__define_initcall(fn, id)`
  - Function name
  - ID: order initcalls

Implementation in our example

```
static int __init foo_init(void)
{
    return 0;
}
postcore_initcall(foo_init);
```

Implementation in our example

```
static int __init foo_init(void)
{
    return 0;
}
postcore_initcall(foo_init);
```

```
#define postcore_initcall(fn)                    __define_initcall(fn, id)
```

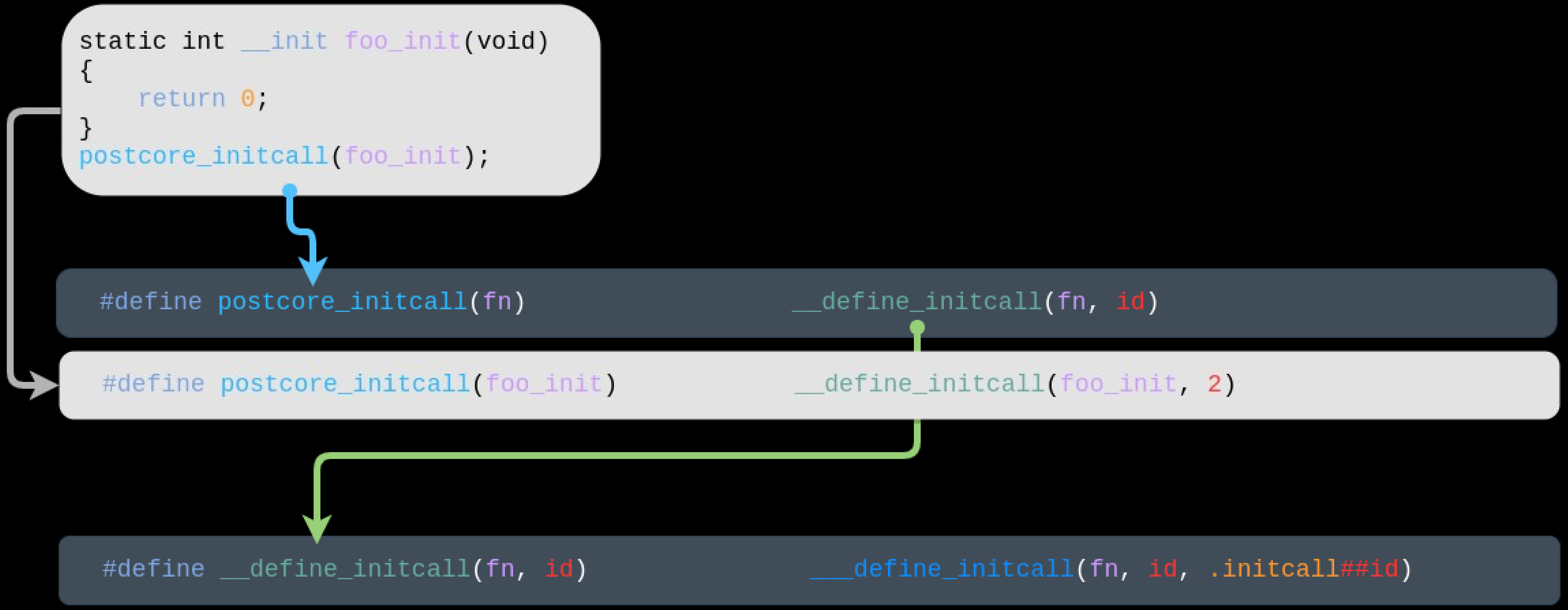
Implementation in our example

```c
static int __init foo_init(void)
{
    return 0;
}
postcore_initcall(foo_init);
```

```c
#define postcore_initcall(fn)                   __define_initcall(fn, id)
```

```c
#define postcore_initcall(foo_init)             __define_initcall(foo_init, 2)
```

Implementation in our example



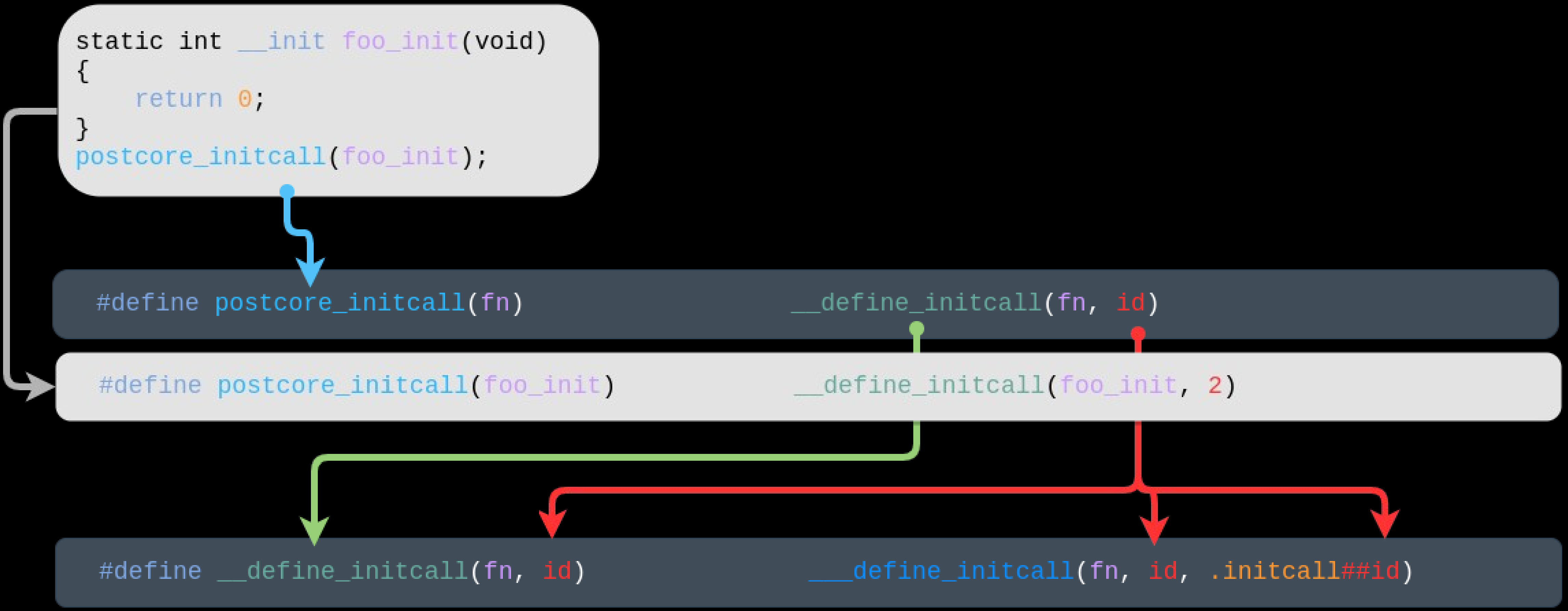
```
static int __init foo_init(void)
{
    return 0;
}
postcore_initcall(foo_init);
```

```
#define postcore_initcall(fn)                 __define_initcall(fn, id)
```

```
#define postcore_initcall(foo_init)           __define_initcall(foo_init, 2)
```

```
#define __define_initcall(fn, id)             ___define_initcall(fn, id, .initcall##id)
```

Implementation in our example



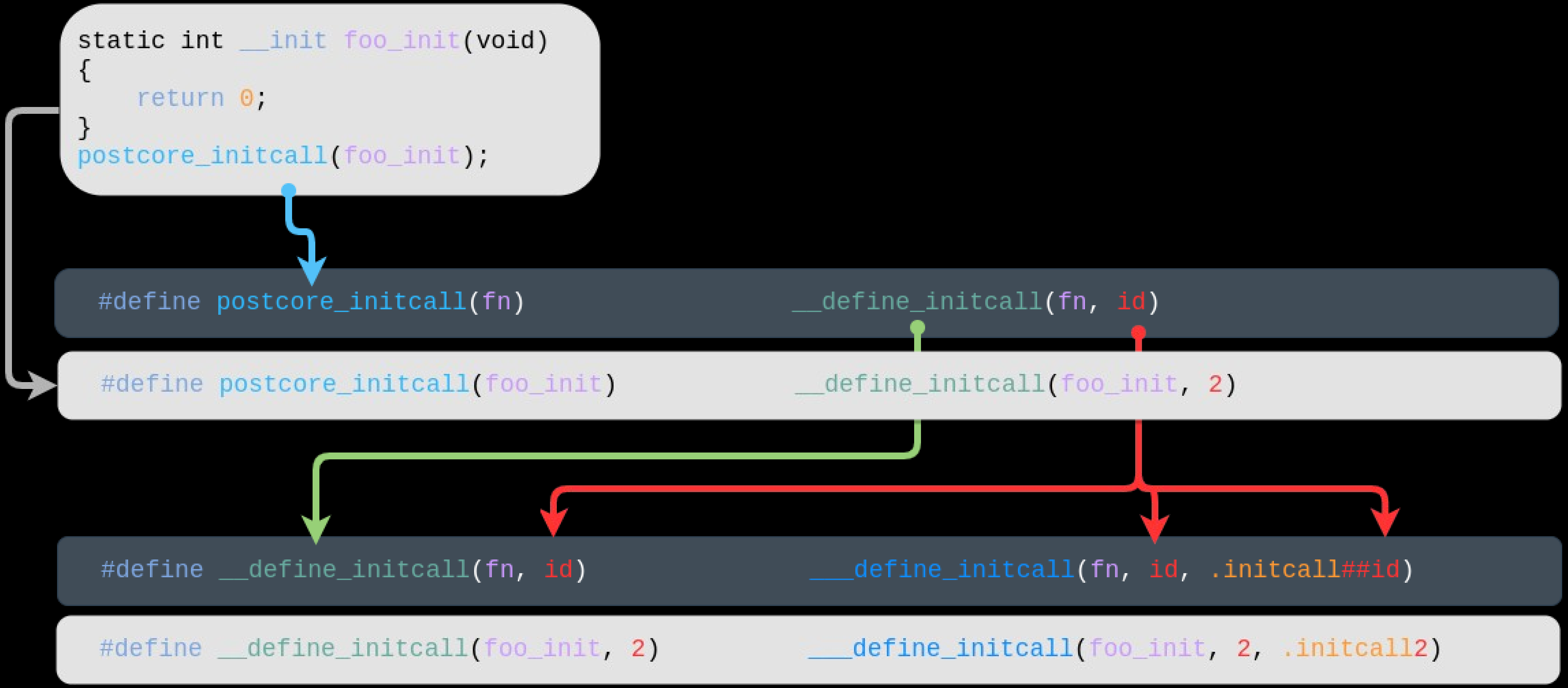
```
static int __init foo_init(void)
{
    return 0;
}
postcore_initcall(foo_init);
```

```
#define postcore_initcall(fn)                __define_initcall(fn, id)
```

```
#define postcore_initcall(foo_init)          __define_initcall(foo_init, 2)
```

```
#define __define_initcall(fn, id)            ___define_initcall(fn, id, .initcall##id)
```

Implementation in our example

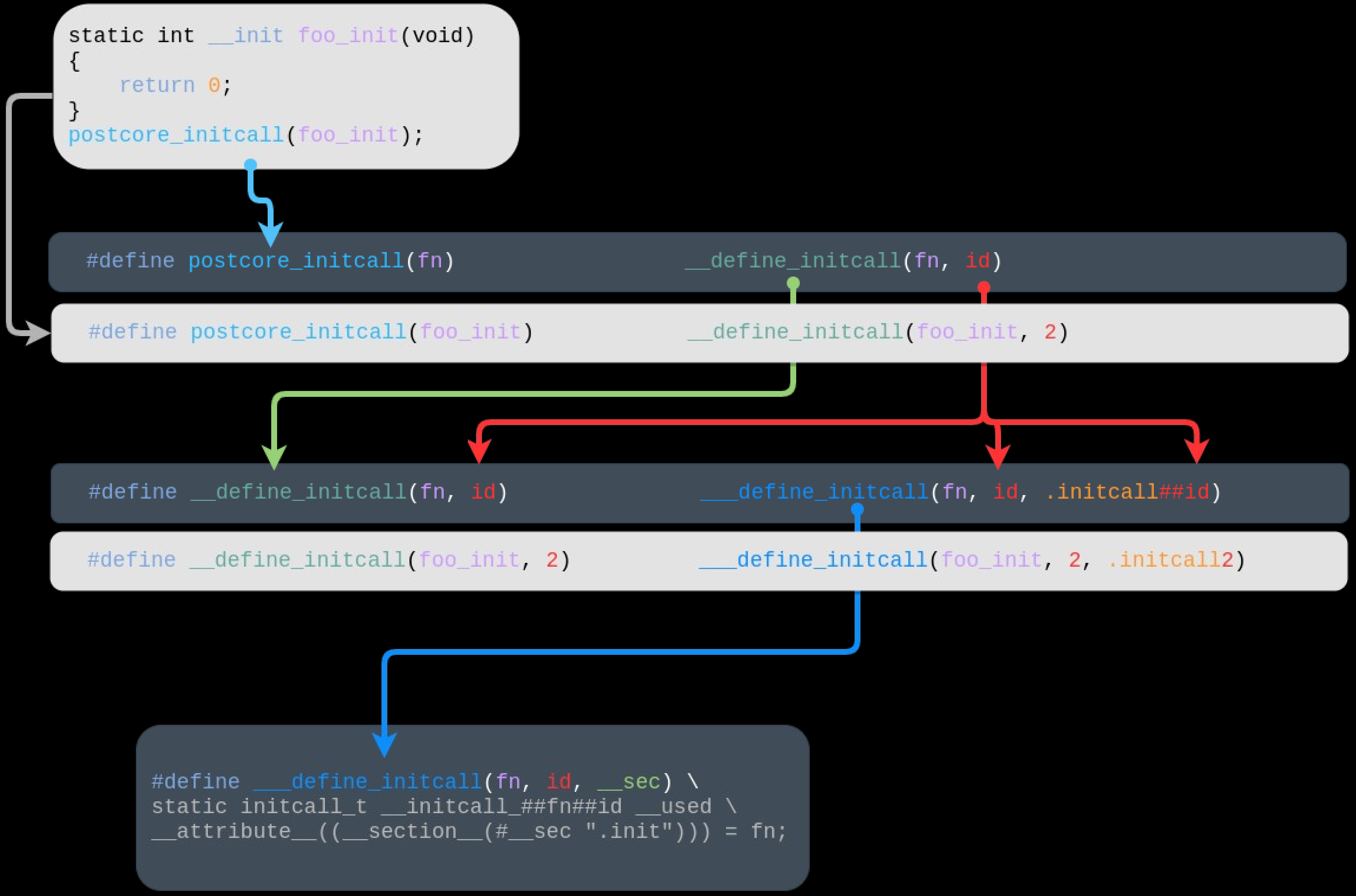


```
static int __init foo_init(void)
{
    return 0;
}
postcore_initcall(foo_init);
```

```
#define postcore_initcall(fn)              __define_initcall(fn, id)
#define postcore_initcall(foo_init)        __define_initcall(foo_init, 2)
```

```
#define __define_initcall(fn, id)              ___define_initcall(fn, id, .initcall##id)
#define __define_initcall(foo_init, 2)         ___define_initcall(foo_init, 2, .initcall2)
```

Implementation in our example



```
static int __init foo_init(void)
{
    return 0;
}
postcore_initcall(foo_init);
```

```
#define postcore_initcall(fn)                   __define_initcall(fn, id)
```

```
#define postcore_initcall(foo_init)             __define_initcall(foo_init, 2)
```

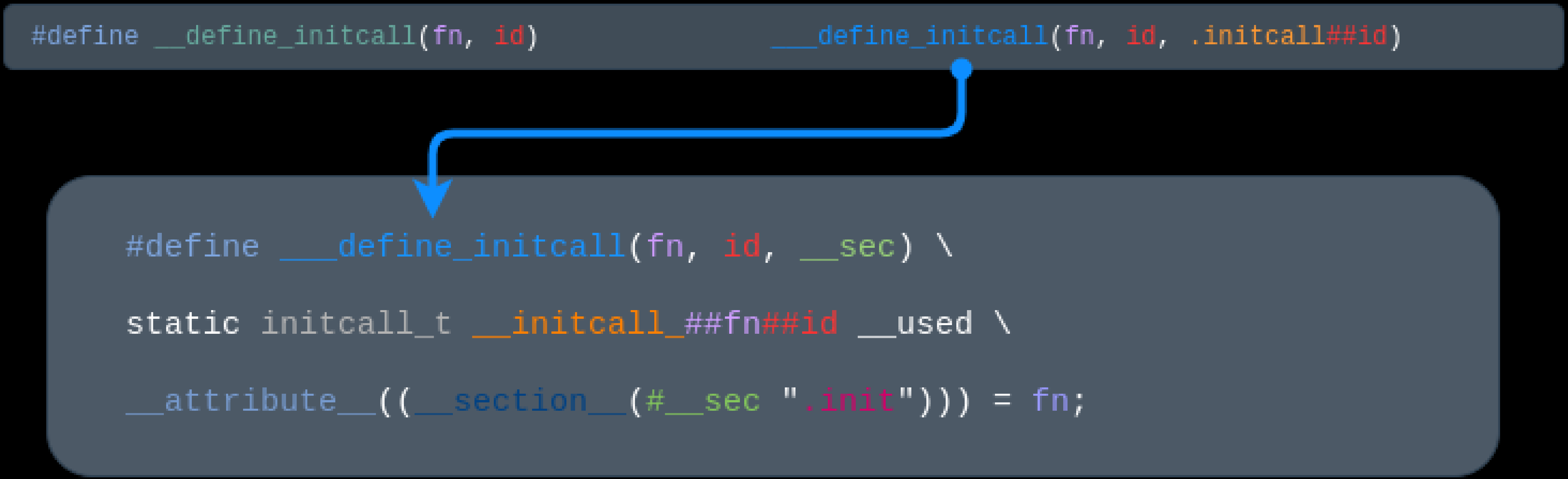
```
#define __define_initcall(fn, id)               ___define_initcall(fn, id, .initcall##id)
```

```
#define __define_initcall(foo_init, 2)          ___define_initcall(foo_init, 2, .initcall2)
```

```
#define ___define_initcall(fn, id, __sec) \
static initcall_t __initcall_##fn##id __used \
__attribute__((__section__(#__sec ".init"))) = fn;
```

2nd __define_initcall()

- Parameters:
  - fn: Initcall’s function name (foo_init)
  - id: initcall’s id (2 for postcore)
  - __sec: the section that will be used in the object file (.initcall2)



```
#define __define_initcall(fn, id)                ___define_initcall(fn, id, .initcall##id)
```

```
#define ___define_initcall(fn, id, __sec) \
static initcall_t __initcall_##fn##id __used \
__attribute__((__section__(#__sec ".init"))) = fn;
```

Expanded version

```c
static int __init foo_init(void)
{
    return 0;
}
postcore_initcall(foo_init);
```

```c
#define postcore_initcall(foo_init)          __define_initcall(foo_init, 2)
```

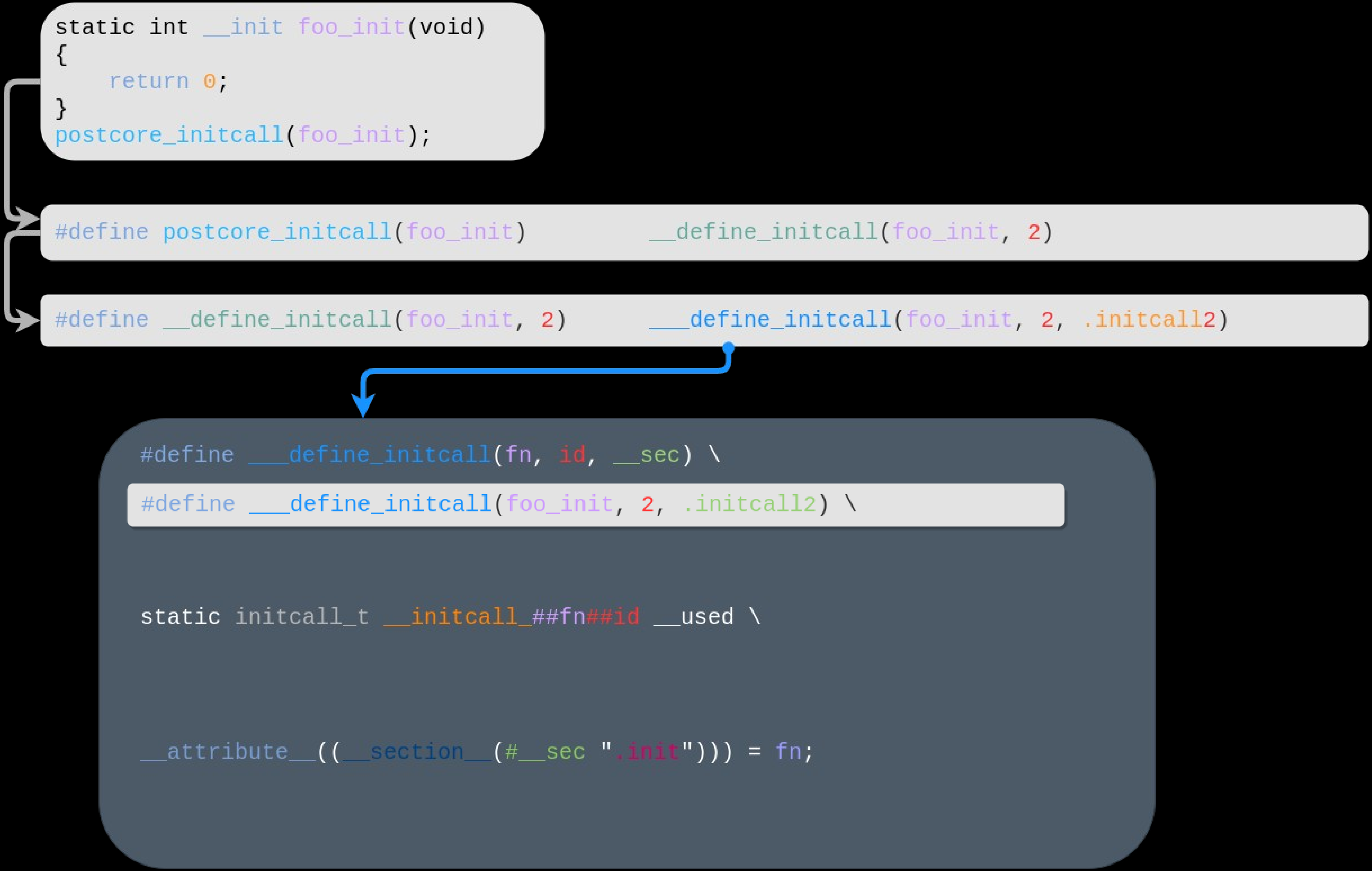
```c
#define __define_initcall(foo_init, 2)       ___define_initcall(foo_init, 2, .initcall2)
```

```c
#define ___define_initcall(fn, id, __sec) \

static initcall_t __initcall_##fn##id __used \

__attribute__((__section__(#__sec ".init"))) = fn;
```

Expanded version



```
static int __init foo_init(void)
{
    return 0;
}
postcore_initcall(foo_init);
```

```
#define postcore_initcall(foo_init)          __define_initcall(foo_init, 2)
```

```
#define __define_initcall(foo_init, 2)       ___define_initcall(foo_init, 2, .initcall2)
```

```
#define ___define_initcall(fn, id, __sec) \
#define ___define_initcall(foo_init, 2, .initcall2) \


static initcall_t __initcall_##fn##id __used \


__attribute__((__section__(#__sec ".init"))) = fn;
```

Expanded version

```
static int __init foo_init(void)
{
    return 0;
}
postcore_initcall(foo_init);
```

```
#define postcore_initcall(foo_init)          __define_initcall(foo_init, 2)
```

```
#define __define_initcall(foo_init, 2)       ___define_initcall(foo_init, 2, .initcall2)
```

```
#define ___define_initcall(fn, id, __sec) \
#define ___define_initcall(foo_init, 2, .initcall2) \


static initcall_t __initcall_##fn##id __used \
static initcall_t __initcall_foo_init2 __used \

__attribute__((__section__(#__sec ".init"))) = fn;
```

Expanded version

```
static int __init foo_init(void)
{
    return 0;
}
postcore_initcall(foo_init);
```

```
#define postcore_initcall(foo_init)         __define_initcall(foo_init, 2)
```

```
#define __define_initcall(foo_init, 2)      ___define_initcall(foo_init, 2, .initcall2)
```

```
#define ___define_initcall(fn, id, __sec) \
#define ___define_initcall(foo_init, 2, .initcall2) \


static initcall_t __initcall_##fn##id __used \
static initcall_t __initcall_foo_init2 __used \

__attribute__((__section__(#__sec ".init"))) = fn;
__attribute__((__section__(.initcall2 ".init"))) = foo_init;
```

Expanded version

```
#define ___define_initcall(fn, id, __sec) \
```

```
#define ___define_initcall(foo_init, 2, .initcall2) \
```

```
static initcall_t __initcall_##fn##id __used \
```

```
static initcall_t __initcall_foo_init2 __used \
```

```
__attribute__((__section__(#__sec ".init"))) = fn;
```

```
__attribute__((__section__(.initcall2 ".init"))) = foo_init;
```

Expanded version

```
#define ___define_initcall(fn, id, __sec) \
```

```
#define ___define_initcall(foo_init, 2, .initcall2) \
```

```
static initcall_t __initcall_##fn##id __used \
```

```
static initcall_t __initcall_foo_init2 __used \
```

```
__attribute__((__section__(#__sec ".init"))) = fn;
```

```
__attribute__((__section__(.initcall2 ".init"))) = foo_init;
```

- Create a `initcall_t` entry named `__initcall_foo_init2`

Expanded version

```
#define ___define_initcall(fn, id, __sec) \
#define ___define_initcall(foo_init, 2, .initcall2) \


static initcall_t __initcall_##fn##id __used \
static initcall_t __initcall_foo_init2 __used \

__attribute__((__section__(#__sec ".init"))) = fn;
__attribute__((__section__(.initcall2 ".init"))) = foo_init;
```

- Create a `initcall_t` entry named __initcall_foo_init2
- attribute/section → name an object-file section .initcall2.init

Expanded version

```
#define ___define_initcall(fn, id, __sec) \
#define ___define_initcall(foo_init, 2, .initcall2) \


static initcall_t __initcall_##fn##id __used \
static initcall_t __initcall_foo_init2 __used \

__attribute__((__section__(#__sec ".init"))) = fn;
__attribute__((__section__(.initcall2 ".init"))) = foo_init;
```

- Create a `initcall_t` entry named __initcall_foo_init2
- attribute/section → name an object-file section .initcall2.init

```
$ objdump -t vmlinux.o | grep foo
000007c l O   .initcall2.init     0000004      __initcall_foo_init2
```

All object-file sections

- `__define_initcall`:

  create an object-file section specific to the initcall used (thanks to its id) pointing to the entry created.

All object-file sections

- `__define_initcall`:

  create an object-file section specific to the initcall used (thanks to its id) pointing to the entry created.

```
$ objdump -t vmlinux.o | grep .initcall2.init
00000000 l     O .initcall2.init  00000004 __initcall_atomic_pool_init2
00000004 l     O .initcall2.init  00000004 __initcall_mvebu_soc_device2
00000008 l     O .initcall2.init  00000004 __initcall_coherency_late_init2
0000000c l     O .initcall2.init  00000004 __initcall_imx_mmdc_init2
00000010 l     O .initcall2.init  00000004 __initcall_omap_hwmod_setup_all2
[...]
0000007c l     O .initcall2.init  00000004 __initcall_foo_init2
00000080 l     O .initcall2.init  00000004 __initcall_rockchip_grf_init2
[...]
```

Demystifying Linux Kernel initcalls

Implementation

- ✓ General
- ☐ Ordering
  - ☐ For a particular level
  - ☐ Between all initcalls
- ☐ Execution

Level-initcalls ordering – Makefiles!

- examples

Level-initcalls ordering – Makefiles!

examples

drivers/rtc/mydriver.c

drivers/rtc/myotherdriver.c

Level-initcalls ordering – Makefiles!

examples

drivers/rtc/mydriver.c

```
#include <linux/init.h>

static int __init mydriver_func(void)
{
        return 0;
}
postcore_initcall(mydriver_func);
```

drivers/rtc/myotherdriver.c

Level-initcalls ordering – Makefiles!

examples

drivers/rtc/mydriver.c

```
#include <linux/init.h>

static int __init mydriver_func(void)
{
	return 0;
}
postcore_initcall(mydriver_func);
```

drivers/rtc/myotherdriver.c

```
#include <linux/init.h>

static int __init myotherdriver_func(void)
{
	return 0;
}
postcore_initcall(myotherdriver_func);
```

Level-initcalls ordering: 1st case

```
$ git diff drivers/rtc/Makefile
[...]
-rtc-core-y                               := class.o interface.o
+rtc-core-y                               := class.o interface.o mydriver.o myotherdriver.o
```

Level-initcalls ordering: 1st case

```
$ git diff drivers/rtc/Makefile
[...]
-rtc-core-y                      := class.o interface.o
+rtc-core-y                      := class.o interface.o mydriver.o myotherdriver.o
```

```
$ objdump -t vmlinux.o | grep "driver_func"
0008c3c8 l     F .init.text         00000008   mydriver_func
000000c8 l     O .initcall2.init    00000004   __initcall_mydriver_func2
0008c3d0 l     F .init.text         00000008   myotherdriver_func
000000cc l     O .initcall2.init    00000004   __initcall_myotherdriver_func2
```

Level-initcalls ordering: 1st case

```
$ git diff drivers/rtc/Makefile
[...]
-rtc-core-y                     := class.o interface.o
+rtc-core-y                     := class.o interface.o mydriver.o myotherdriver.o
```

```
$ objdump -t vmlinux.o | grep "driver_func"
0008c3c8 l     F .init.text        00000008   mydriver_func
000000c8 l     O .initcall2.init   00000004   __initcall_mydriver_func2
0008c3d0 l     F .init.text        00000008   myotherdriver_func
000000cc l     O .initcall2.init   00000004   __initcall_myotherdriver_func2
```

```
# cat /sys/kernel/debug/tracing/trace | grep driver_func
   swapper/0-1     [000] ....     0.059546: initcall_start:   func=mydriver_func+0x0/0x8
   swapper/0-1     [000] ....     0.059556: initcall_finish:  func=mydriver_func+0x0/0x8 ret=0
   swapper/0-1     [000] ....     0.059571: initcall_start:   func=myotherdriver_func+0x0/0x8
   swapper/0-1     [000] ....     0.059581: initcall_finish:  func=myotherdriver_func+0x0/0x8 ret=0
```

Level-initcalls ordering: 2nd case

```
$ git diff drivers/rtc/Makefile
[...]
-rtc-core-y                                       := class.o interface.o
+rtc-core-y                                       := class.o interface.o myotherdriver.o mydriver.o
```

Level-initcalls ordering: 2nd case

```
$ git diff drivers/rtc/Makefile
[...]
-rtc-core-y                     := class.o interface.o
+rtc-core-y                     := class.o interface.o myotherdriver.o mydriver.o
```

```
$ objdump -t vmlinux.o | grep "driver_func"
0008c3c8 l     F .init.text        00000008   myotherdriver_func
000000c8 l     O .initcall2.init   00000004   __initcall_myotherdriver_func2
0008c3d0 l     F .init.text        00000008   mydriver_func
000000cc l     O .initcall2.init   00000004   __initcall_mydriver_func2
```

Level-initcalls ordering: 2nd case

```
$ git diff drivers/rtc/Makefile
[...]
-rtc-core-y                        := class.o interface.o
+rtc-core-y                        := class.o interface.o myotherdriver.o mydriver.o
```

```
$ objdump -t vmlinux.o | grep "driver_func"
0008c3c8 l     F .init.text        00000008   myotherdriver_func
000000c8 l     O .initcall2.init   00000004   __initcall_myotherdriver_func2
0008c3d0 l     F .init.text        00000008   mydriver_func
000000cc l     O .initcall2.init   00000004   __initcall_mydriver_func2
```

```
# cat /sys/kernel/debug/tracing/trace | grep driver_func
   swapper/0-1     [000] ....     0.059520: initcall_start:   func=myotherdriver_func+0x0/0x8
   swapper/0-1     [000] ....     0.059530: initcall_finish:  func=myotherdriver_func+0x0/0x8 ret=0
   swapper/0-1     [000] ....     0.059545: initcall_start:   func=mydriver_func+0x0/0x8
   swapper/0-1     [000] ....     0.059555: initcall_finish:  func=mydriver_func+0x0/0x8 ret=0
```

Demystifying Linux Kernel initcalls

Implementation

✓ General

☐ Ordering

  ✓ For a particular level

  ☐ Between all initcalls

☐ Execution

☐ Modules

Initcall level ordering

- `init/main.c`

Initcall level ordering

- init/main.c

```
extern initcall_entry_t __initcall_start[];
extern initcall_entry_t __initcall0_start[];
extern initcall_entry_t __initcall1_start[];
extern initcall_entry_t __initcall2_start[];
extern initcall_entry_t __initcall3_start[];
extern initcall_entry_t __initcall4_start[];
extern initcall_entry_t __initcall5_start[];
extern initcall_entry_t __initcall6_start[];
extern initcall_entry_t __initcall7_start[];
extern initcall_entry_t __initcall_end[];

static initcall_entry_t *initcall_levels[] __initdata = {
        __initcall0_start,
        __initcall1_start,
        __initcall2_start,
        __initcall3_start,
        __initcall4_start,
        __initcall5_start,
        __initcall6_start,
        __initcall7_start,
        __initcall_end,
};
```

Initcall level ordering

- init/main.c
- initcall_levels: Array which each entry is a pointer for a particular level

```
extern initcall_entry_t __initcall_start[];
extern initcall_entry_t __initcall0_start[];
extern initcall_entry_t __initcall1_start[];
extern initcall_entry_t __initcall2_start[];
extern initcall_entry_t __initcall3_start[];
extern initcall_entry_t __initcall4_start[];
extern initcall_entry_t __initcall5_start[];
extern initcall_entry_t __initcall6_start[];
extern initcall_entry_t __initcall7_start[];
extern initcall_entry_t __initcall_end[];

static initcall_entry_t *initcall_levels[] __initdata = {
        __initcall0_start,
        __initcall1_start,
        __initcall2_start,
        __initcall3_start,
        __initcall4_start,
        __initcall5_start,
        __initcall6_start,
        __initcall7_start,
        __initcall_end,
};
```

Linker script

include/asm-generic/vmlinux.lds.h

```
#define INIT_CALLS_LEVEL(level)                    \
                __initcall##level##_start = .;     \
                KEEP(*(.initcall##level##.init))   \
                KEEP(*(.initcall##level##s.init))  \
```

Linker script

include/asm-generic/vmlinux.lds.h

```
#define INIT_CALLS_LEVEL(level)                    \
                __initcall##level##_start = .;     \
                KEEP(*(.initcall##level##.init))   \
                KEEP(*(.initcall##level##s.init)) \
```

arch/arm/kernel/vmlinux.lds

```
.init.data : AT(ADDR(.init.data) - 0)

__initcall_start = .;        KEEP(*(.initcallearly.init))
__initcall0_start = .;       KEEP(*(.initcall0.init))
__initcall1_start = .;       KEEP(*(.initcall1.init))
__initcall2_start = .;       KEEP(*(.initcall2.init))
__initcall3_start = .;       KEEP(*(.initcall3.init))
__initcall4_start = .;       KEEP(*(.initcall4.init))
__initcall5_start = .;       KEEP(*(.initcall5.init))
__initcallrootfs_start = .;  KEEP(*(.initcallrootfs.init))
__initcall6_start = .;       KEEP(*(.initcall6.init))
__initcall7_start = .;       KEEP(*(.initcall7.init))
__initcall_end = .
```

Linker script

include/asm-generic/vmlinux.lds.h

```
#define INIT_CALLS_LEVEL(level)                 \
                __initcall##level##_start = .;    \
                KEEP(*(.initcall##level##.init))  \
                KEEP(*(.initcall##level##s.init)) \
```

arch/arm/kernel/vmlinux.lds

```
.init.data : AT(ADDR(.init.data) - 0)

__initcall_start = .;       KEEP(*(.initcallearly.init))
__initcall0_start = .;      KEEP(*(.initcall0.init))
__initcall1_start = .;      KEEP(*(.initcall1.init))
__initcall2_start = .;      KEEP(*(.initcall2.init))
__initcall3_start = .;      KEEP(*(.initcall3.init))
__initcall4_start = .;      KEEP(*(.initcall4.init))
__initcall5_start = .;      KEEP(*(.initcall5.init))
__initcallrootfs_start = .; KEEP(*(.initcallrootfs.init))
__initcall6_start = .;      KEEP(*(.initcall6.init))
__initcall7_start = .;      KEEP(*(.initcall7.init))
__initcall_end = .
```

- `__initcall2_start`: points to the first address of .initcall2.init section in object-file

Demystifying Linux Kernel initcalls

Implementation

- ✓ General
- ✓ Ordering
  - ✓ For a particular level
  - ✓ Between all initcalls
- ☐ Execution
- ☐ Modules

do_initcalls function

init/main.c

```
static void __init do_basic_setup(void)
{
      [...]
      do_initcalls();
}

static void __init do_initcalls(void)
{
      int level;
      [...]

      for (level = 0; level < ARRAY_SIZE(initcall_levels)-1;level++) {
          [...]
          do_initcall_level(level, command_line);
      }
}
```

do_initcalls function

init/main.c

```
static void __init do_basic_setup(void)
{
      [...]
      do_initcalls();
}

static void __init do_initcalls(void)
{
      int level;
      [...]

      for (level = 0; level < ARRAY_SIZE(initcall_levels)—1;level++) {
          [...]
          do_initcall_level(level, command_line);
      }
}
```

- do_initcalls: A loop on all initcalls levels using initcall_levels array

do_initcall_level function

init/main.c

```
static void __init do_initcall_level(int level,char *command_line)
{
     initcall_entry_t *fn;
     [...]
     for (fn = initcall_levels[level]; fn < initcall_levels[level+1]; fn++)
        do_one_initcall(initcall_from_entry(fn));
}
```

do_initcall_level function

init/main.c

```
static void __init do_initcall_level(int level,char *command_line)
{
      initcall_entry_t *fn;
      [...]
      for (fn = initcall_levels[level]; fn < initcall_levels[level+1]; fn++)
          do_one_initcall(initcall_from_entry(fn));
}
```

- `do_initcall_level`: Calling all initcalls for a particular level

do_initcall_level function

init/main.c

```
static void __init do_initcall_level(int level,char *command_line)
{
      initcall_entry_t *fn;
      [...]
      for (fn = initcall_levels[level]; fn < initcall_levels[level+1]; fn++)
         do_one_initcall(initcall_from_entry(fn));
}
```

- `do_initcall_level`: Calling all initcalls for a particular level
- `initcall_entry_t`: Its first value is the address given by `__initcall2_start` (i.e. first `.initcall2.init` section)

do_initcall_level function

init/main.c

```
static void __init do_initcall_level(int level,char *command_line)
{
     initcall_entry_t *fn;
     [...]
     for (fn = initcall_levels[level]; fn < initcall_levels[level+1]; fn++)
        do_one_initcall(initcall_from_entry(fn));
}
```

- `do_initcall_level`: Calling all initcalls for a particular level
- initcall_entry_t: Its first value is the address given by `__initcall2_start` (i.e. first `.initcall2.init` section)
- Iteration on all the addresses of the section `.initcall2.init`

do_initcall_level example

```
static void __init do_initcall_level(int level,char *command_line)
{
   initcall_entry_t *fn;
   [...]
   for (fn = initcall_levels[level]; fn < initcall_levels[level+1]; fn++)
        do_one_initcall(initcall_from_entry(fn));
}
```

```
$ objdump -t vmlinux.o | grep .initcall2.init
00000000 l     O .initcall2.init  00000004 __initcall_atomic_pool_init2
00000004 l     O .initcall2.init  00000004 __initcall_mvebu_soc_device2
00000008 l     O .initcall2.init  00000004 __initcall_coherency_late_init2
[...]
```

do_initcall_level example

```
static void __init do_initcall_level(int level,char *command_line)
{
   initcall_entry_t *fn;
   [...]
   for (fn = initcall_levels[level]; fn < initcall_levels[level+1]; fn++)
        do_one_initcall(initcall_from_entry(fn));
}
```

```
$ objdump -t vmlinux.o | grep .initcall2.init
00000000 l     O .initcall2.init  00000004 __initcall_atomic_pool_init2
00000004 l     O .initcall2.init  00000004 __initcall_mvebu_soc_device2
00000008 l     O .initcall2.init  00000004 __initcall_coherency_late_init2
[...]
```

- Values of fn:

do_initcall_level example

```
static void __init do_initcall_level(int level,char *command_line)
{
   initcall_entry_t *fn;
   [...]
   for (fn = initcall_levels[level]; fn < initcall_levels[level+1]; fn++)
        do_one_initcall(initcall_from_entry(fn));
}
```

```
$ objdump -t vmlinux.o | grep .initcall2.init
00000000 l     O .initcall2.init  00000004 __initcall_atomic_pool_init2
00000004 l     O .initcall2.init  00000004 __initcall_mvebu_soc_device2
00000008 l     O .initcall2.init  00000004 __initcall_coherency_late_init2
[...]
```

- Values of `fn`:

1) address of 1st .initcall2.init

   = 00000000 → __initcall_atomic_pool_init2

do_initcall_level example

```
static void __init do_initcall_level(int level,char *command_line)
{
    initcall_entry_t *fn;
    [...]
    for (fn = initcall_levels[level]; fn < initcall_levels[level+1]; fn++)
         do_one_initcall(initcall_from_entry(fn));
}
```

```
$ objdump -t vmlinux.o | grep .initcall2.init
00000000 l     O .initcall2.init  00000004 __initcall_atomic_pool_init2
00000004 l     O .initcall2.init  00000004 __initcall_mvebu_soc_device2
00000008 l     O .initcall2.init  00000004 __initcall_coherency_late_init2
[...]
```

- Values of `fn`:

1) address of 1st .initcall2.init

   = 00000000 → __initcall_atomic_pool_init2

2) (`fn++`) next address:

   00000004 → __initcall_mvebu_soc_device2

do_initcall_level example

```
static void __init do_initcall_level(int level,char *command_line)
{
    initcall_entry_t *fn;
    [...]
    for (fn = initcall_levels[level]; fn < initcall_levels[level+1]; fn++)
        do_one_initcall(initcall_from_entry(fn));
}
```

```
$ objdump -t vmlinux.o | grep .initcall2.init
00000000 l     O .initcall2.init  00000004 __initcall_atomic_pool_init2
00000004 l     O .initcall2.init  00000004 __initcall_mvebu_soc_device2
00000008 l     O .initcall2.init  00000004 __initcall_coherency_late_init2
[...]
```

- Values of `fn`:

1) address of 1st .initcall2.init

   = 00000000 → __initcall_atomic_pool_init2

2) (fn++) next address:

   00000004 → __initcall_mvebu_soc_device2

3) (fn++) next address:

   00000008 → __initcall_coherency_late_init2

do_one_initcall function

```
int __init_or_module do_one_initcall(initcall_t fn) {
	int ret;
	[...]

	do_trace_initcall_start(fn);
	ret = fn();
	do_trace_initcall_finish(fn, ret);
	[...]

	return ret;
}
```

do_one_initcall function

```
int __init_or_module do_one_initcall(initcall_t fn) {
	int ret;
	[...]

	do_trace_initcall_start(fn);
	ret = fn();
	do_trace_initcall_finish(fn, ret);
	[...]

	return ret;
}
```

- start/finish trace functions

do_one_initcall function

```
int __init_or_module do_one_initcall(initcall_t fn) {
    int ret;
    [...]

    do_trace_initcall_start(fn);
    ret = fn();
    do_trace_initcall_finish(fn, ret);
    [...]

    return ret;
}
```

- start/finish trace functions
- Execute the initcall_t fn == function created

Summary

Summary

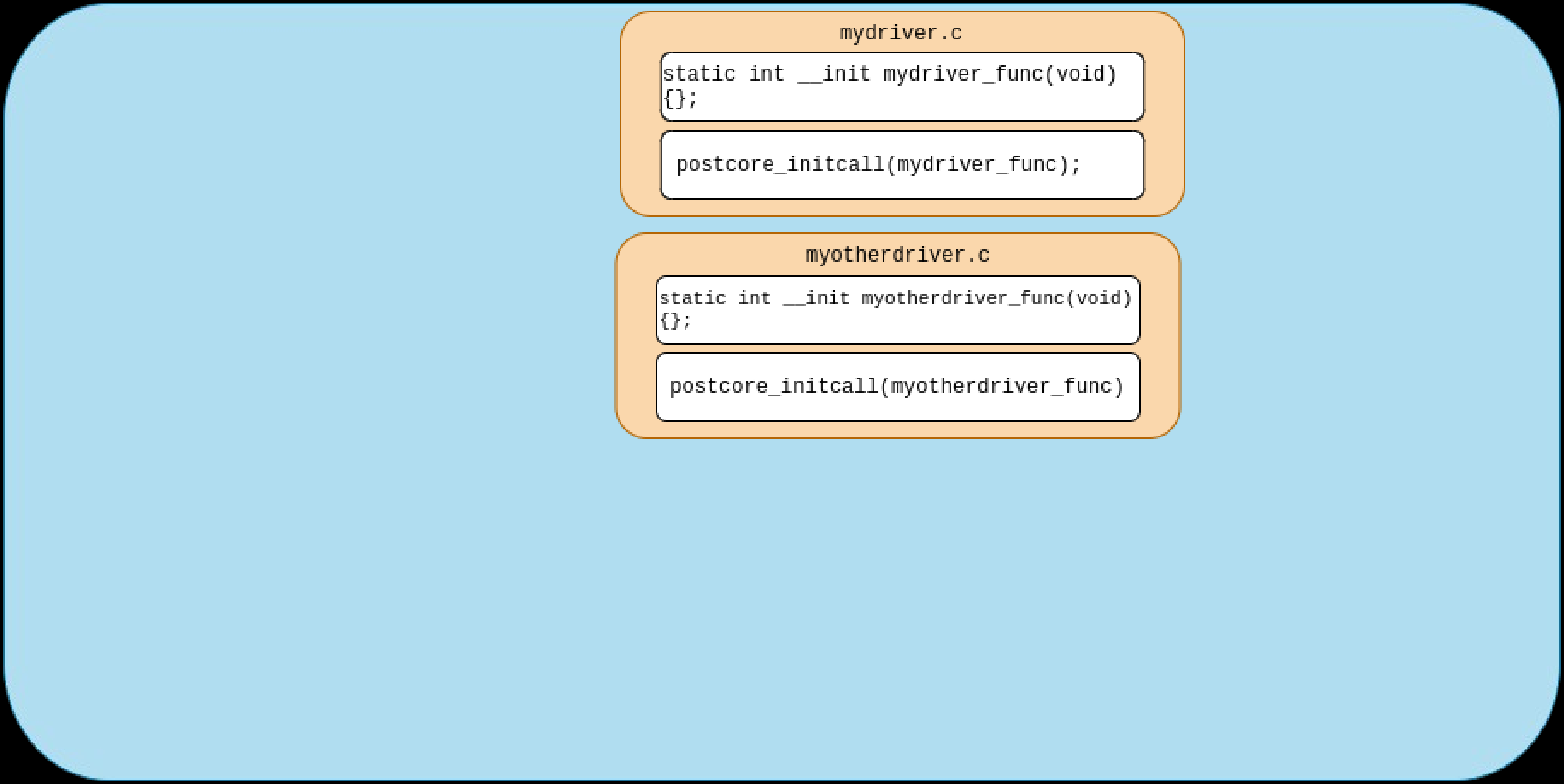
Legend

🟥 Kernel implementation

🟧 Developer's code

🟩 object file, after compilation

Summary

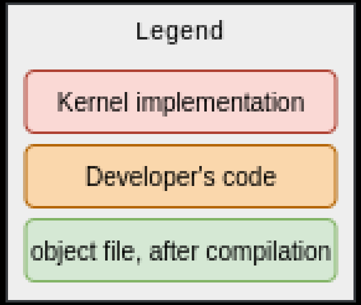


```
static int __init mydriver_func(void)
{};
```

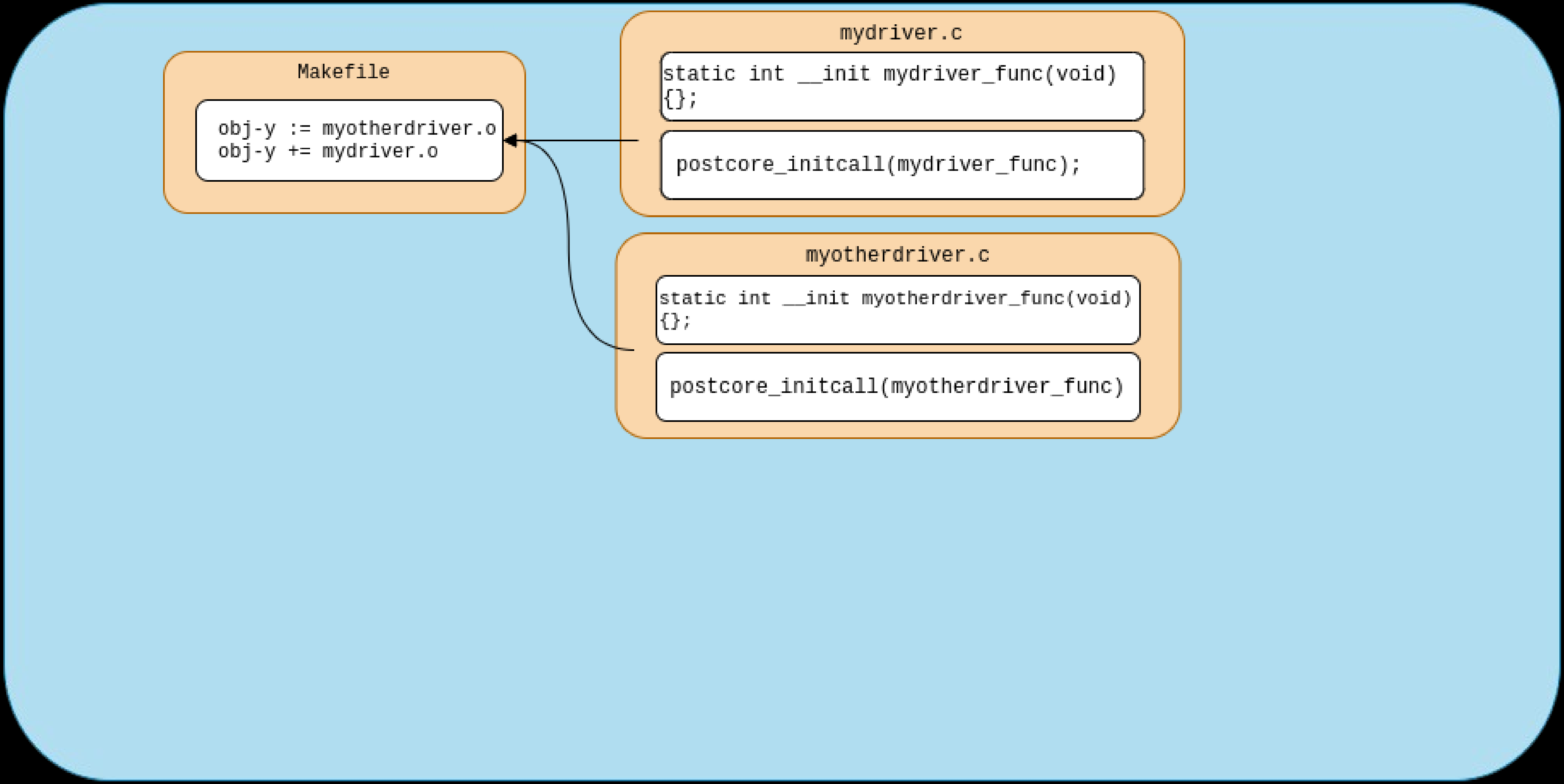
```
postcore_initcall(mydriver_func);
```

```
static int __init myotherdriver_func(void)
{};
```

```
postcore_initcall(myotherdriver_func)
```



Summary



Makefile

```
obj-y := myotherdriver.o
obj-y += mydriver.o
```

mydriver.c

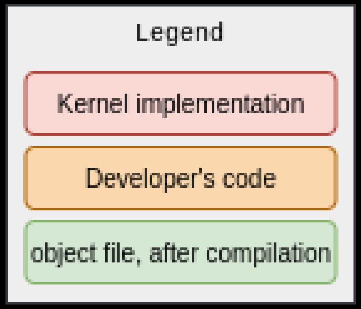
```
static int __init mydriver_func(void)
{};
```

```
postcore_initcall(mydriver_func);
```

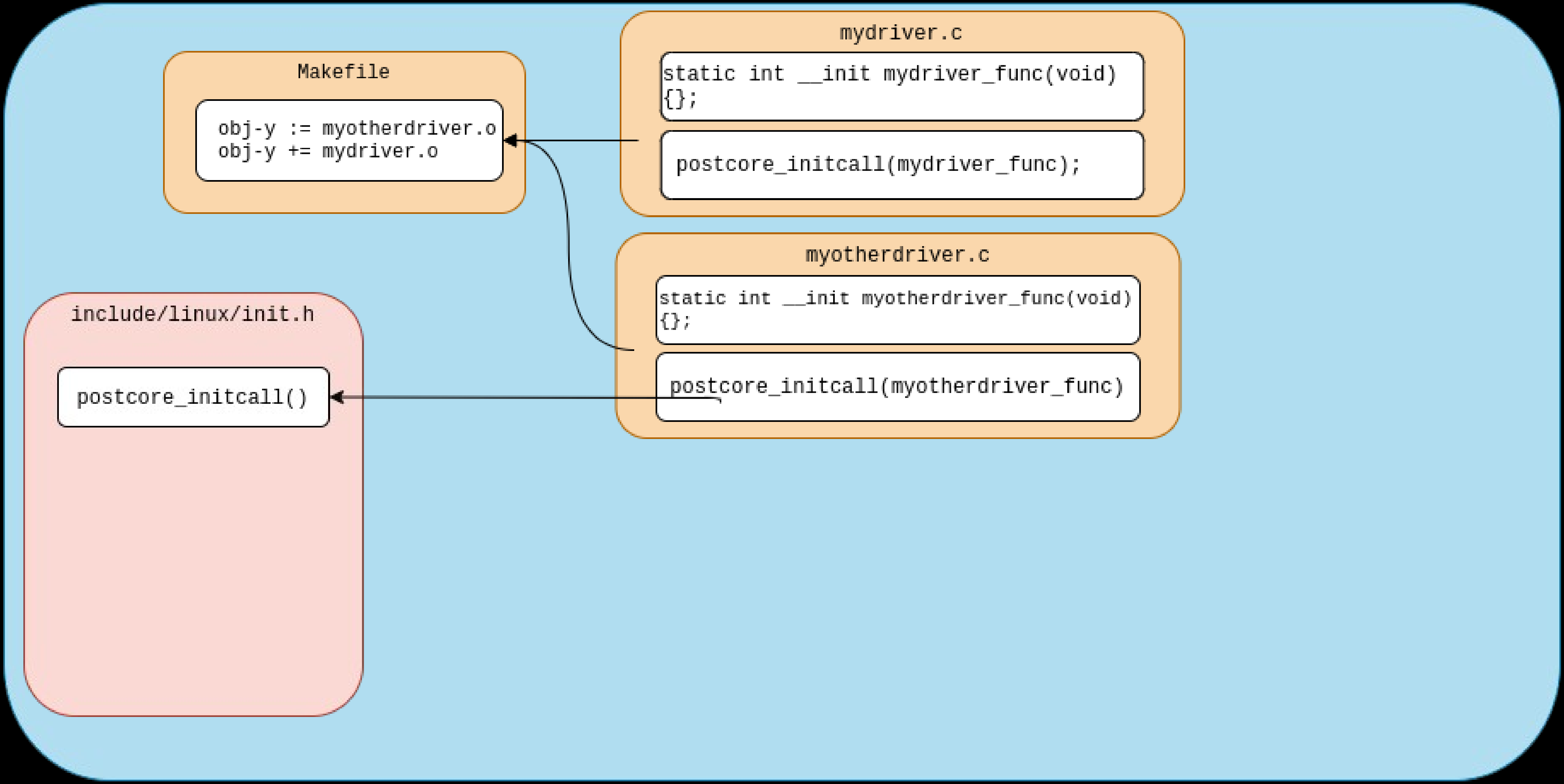
myotherdriver.c

```
static int __init myotherdriver_func(void)
{};
```

```
postcore_initcall(myotherdriver_func)
```



Summary



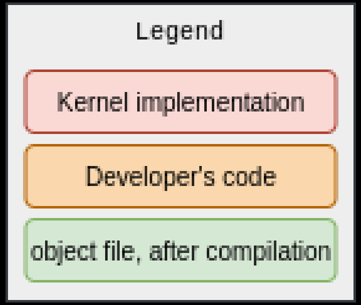
```
obj-y := myotherdriver.o
obj-y += mydriver.o
```

```
static int __init mydriver_func(void)
{};
```

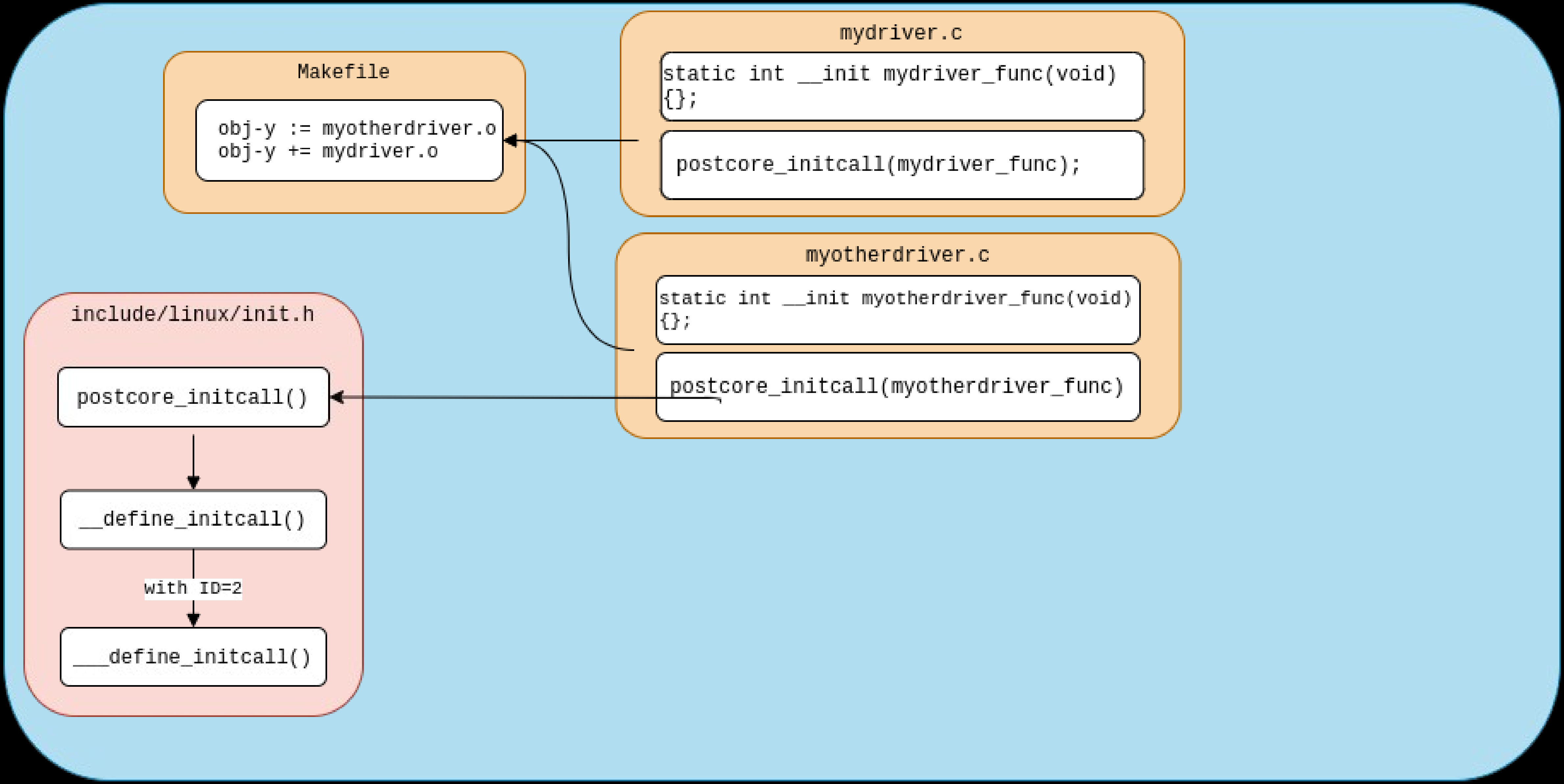
```
postcore_initcall(mydriver_func);
```

```
static int __init myotherdriver_func(void)
{};
```

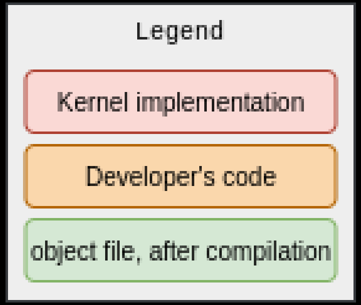
```
postcore_initcall(myotherdriver_func)
```



Summary

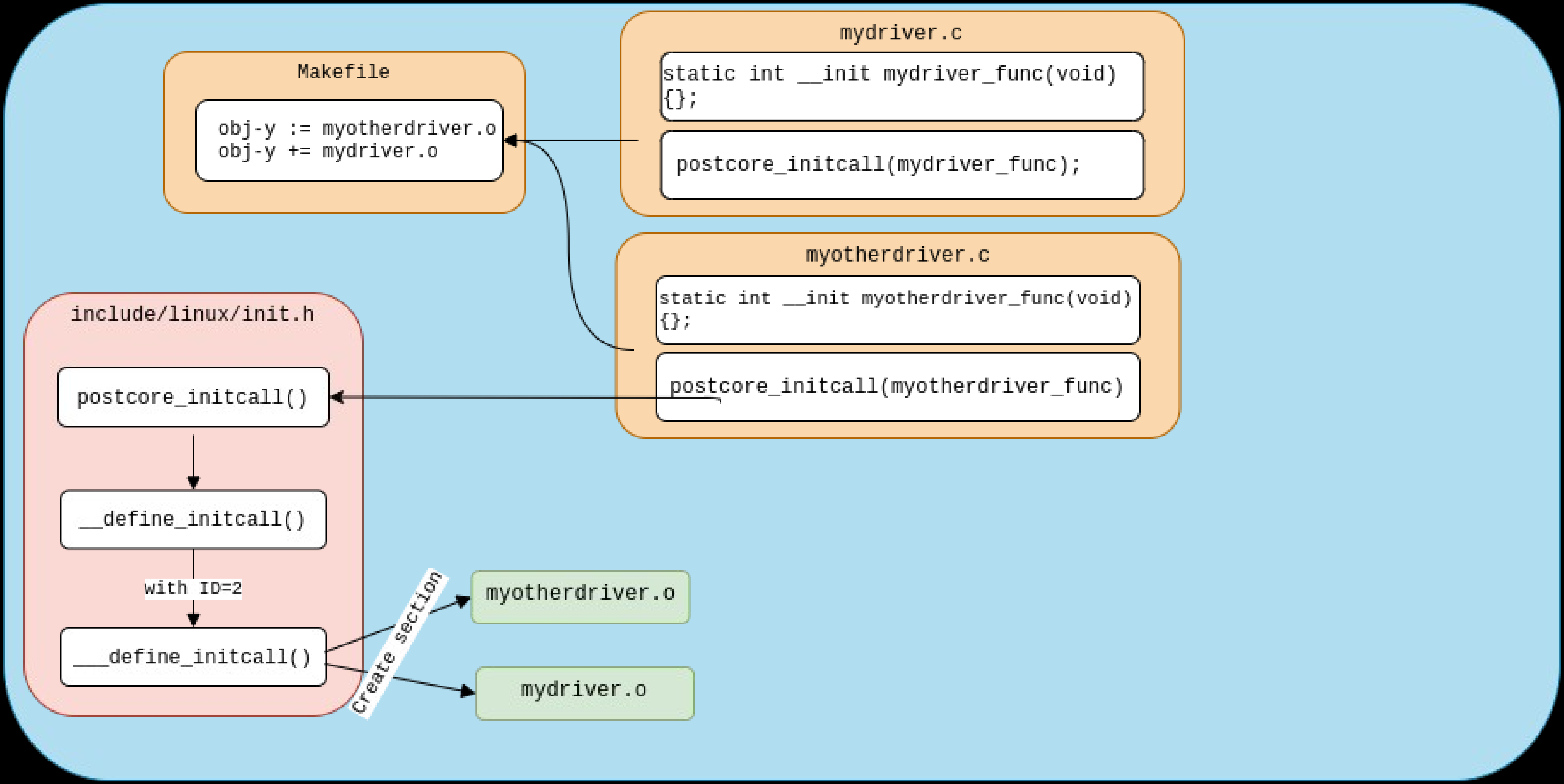


Makefile
obj-y := myotherdriver.o
obj-y += mydriver.o
mydriver.c
static int __init mydriver_func(void)
{};
postcore_initcall(mydriver_func);
myotherdriver.c
static int __init myotherdriver_func(void)
{};
postcore_initcall(myotherdriver_func)
include/linux/init.h
postcore_initcall()
__define_initcall()
with ID=2
___define_initcall()

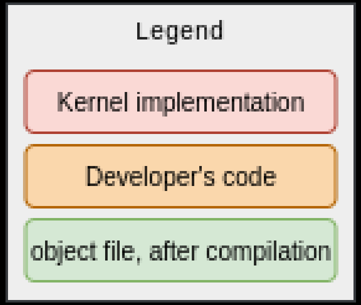


Legend
Kernel implementation
Developer's code
object file, after compilation

Summary

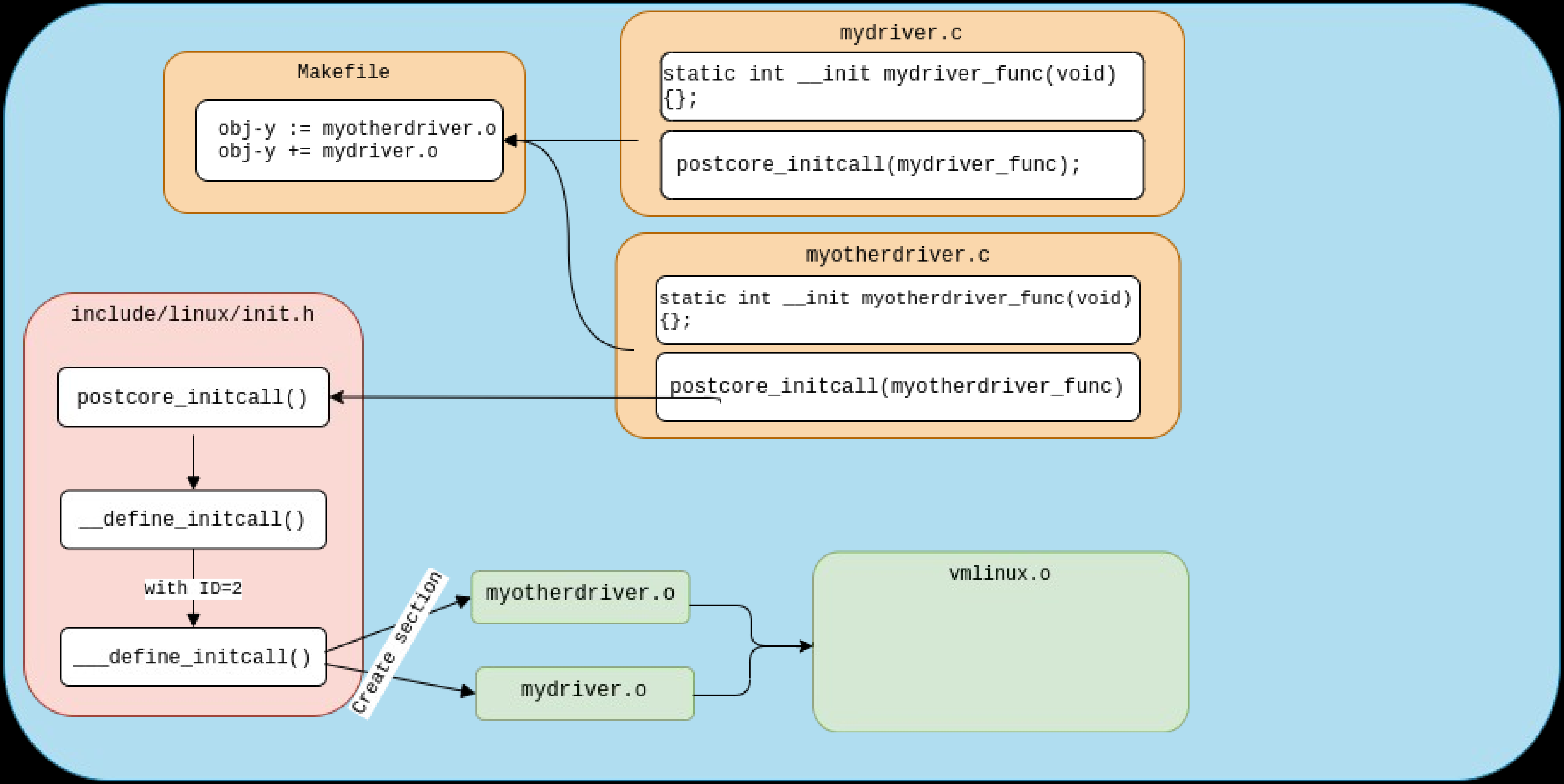


Makefile
obj-y := myotherdriver.o
obj-y += mydriver.o
mydriver.c
static int __init mydriver_func(void)
{};
postcore_initcall(mydriver_func);
myotherdriver.c
static int __init myotherdriver_func(void)
{};
postcore_initcall(myotherdriver_func)
include/linux/init.h
postcore_initcall()
__define_initcall()
with ID=2
___define_initcall()
create section
myotherdriver.o
mydriver.o

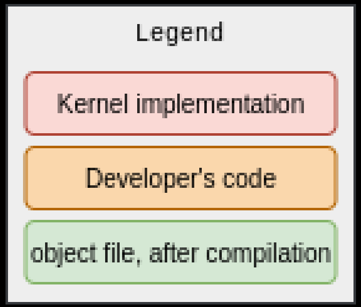


Legend
Kernel implementation
Developer's code
object file, after compilation

Summary

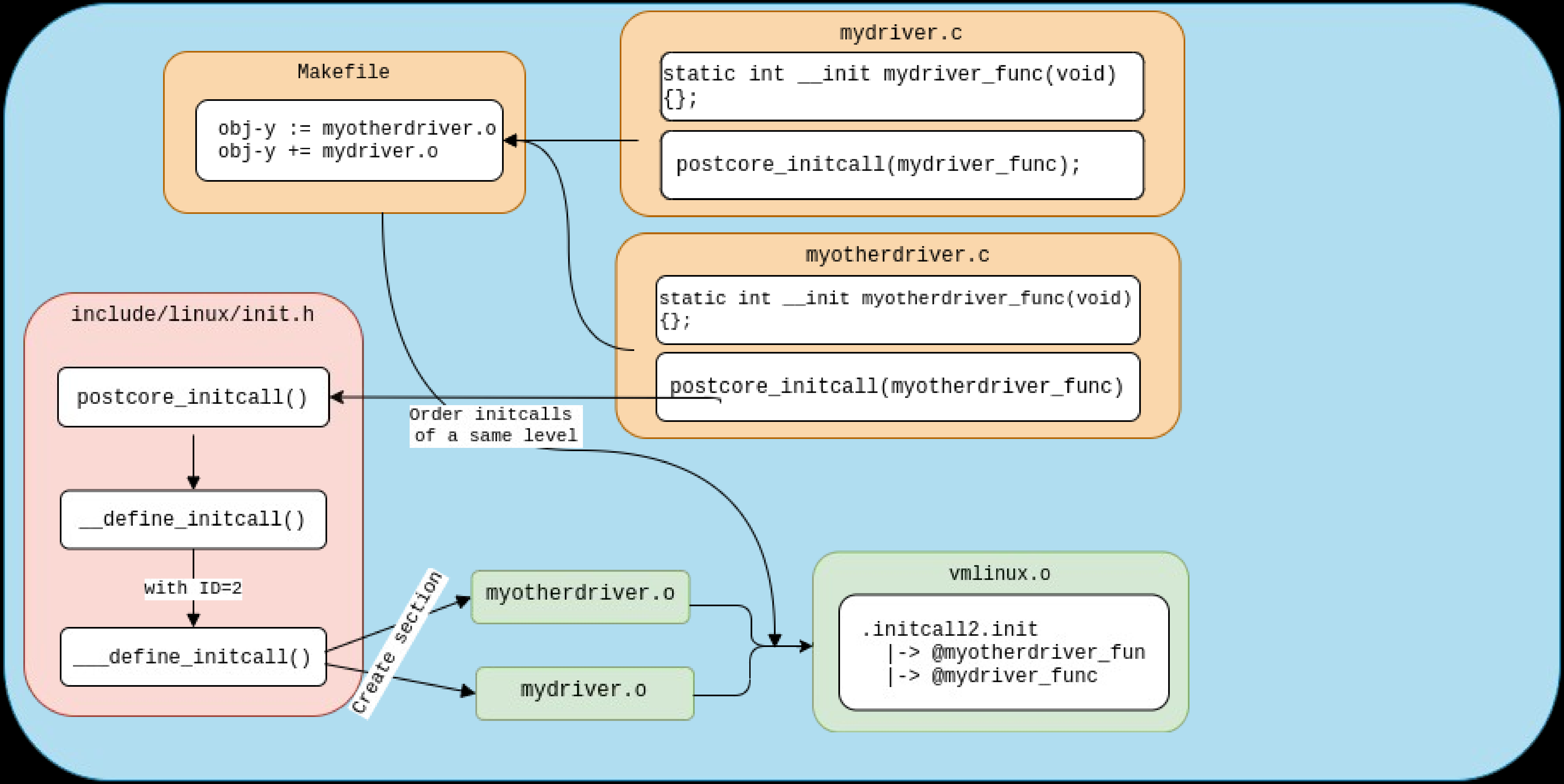


Makefile
obj-y := myotherdriver.o
obj-y += mydriver.o
mydriver.c
static int __init mydriver_func(void)
{};
postcore_initcall(mydriver_func);
myotherdriver.c
static int __init myotherdriver_func(void)
{};
postcore_initcall(myotherdriver_func)
include/linux/init.h
postcore_initcall()
__define_initcall()
with ID=2
___define_initcall()
create section
myotherdriver.o
mydriver.o
vmlinux.o

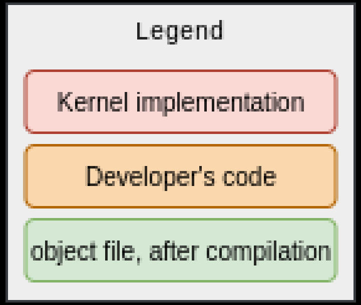


Legend
Kernel implementation
Developer's code
object file, after compilation

Summary

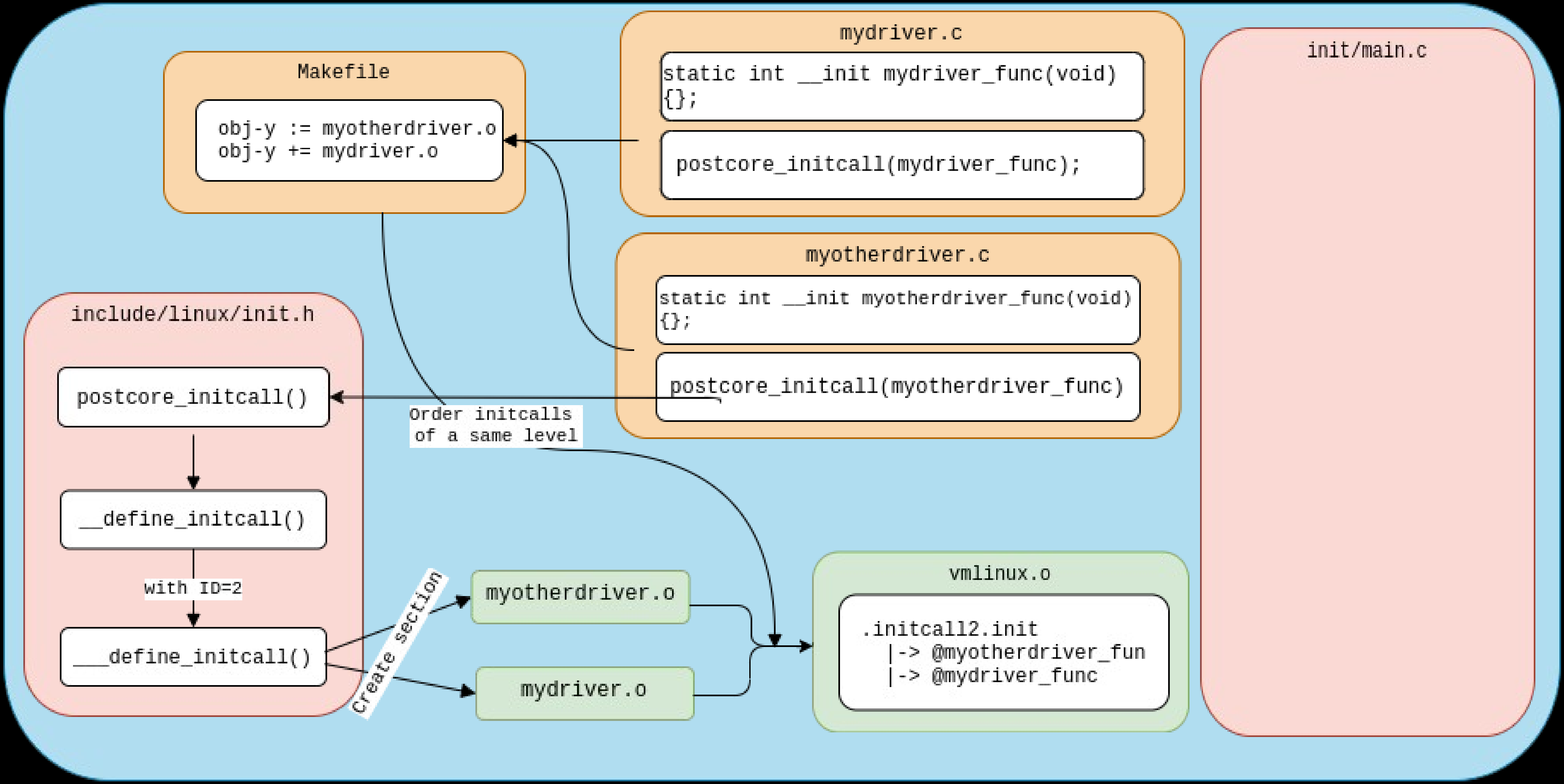


Makefile
obj-y := myotherdriver.o
obj-y += mydriver.o
mydriver.c
static int __init mydriver_func(void) {};
postcore_initcall(mydriver_func);
myotherdriver.c
static int __init myotherdriver_func(void) {};
postcore_initcall(myotherdriver_func)
include/linux/init.h
postcore_initcall()
__define_initcall()
with ID=2
___define_initcall()
Order initcalls of a same level
create section
myotherdriver.o
mydriver.o
vmlinux.o
.initcall2.init
|-> @myotherdriver_fun
|-> @mydriver_func

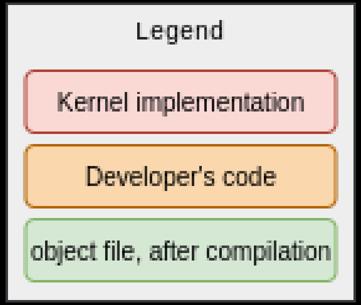


Legend
Kernel implementation
Developer's code
object file, after compilation

Summary

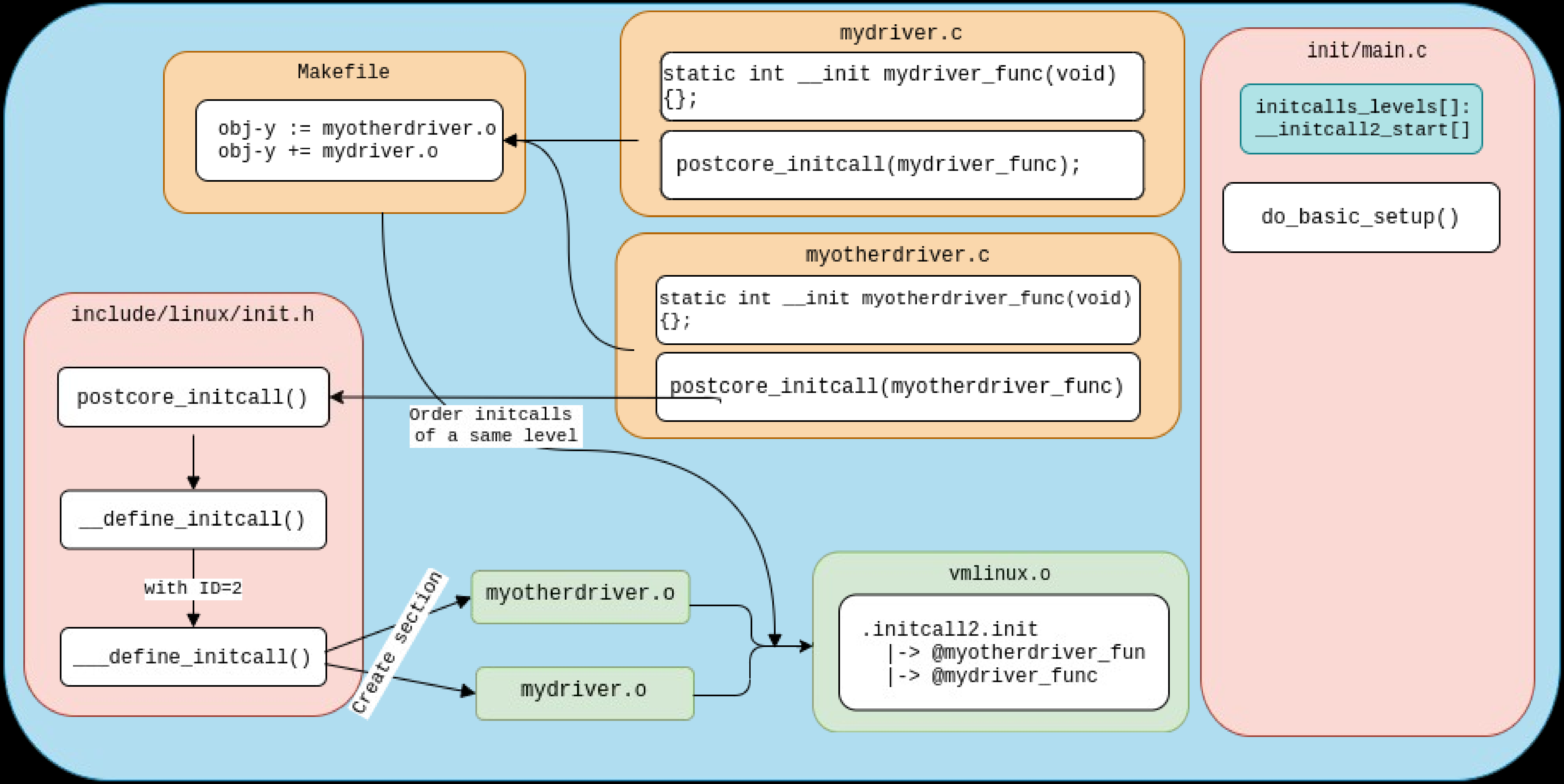


mydriver.c
static int __init mydriver_func(void) {};
postcore_initcall(mydriver_func);
Makefile
obj-y := myotherdriver.o
obj-y += mydriver.o
myotherdriver.c
static int __init myotherdriver_func(void) {};
postcore_initcall(myotherdriver_func)
init/main.c
include/linux/init.h
postcore_initcall()
__define_initcall()
with ID=2
___define_initcall()
Order initcalls of a same level
create section
myotherdriver.o
mydriver.o
vmlinux.o
.initcall2.init
|-> @myotherdriver_fun
|-> @mydriver_func

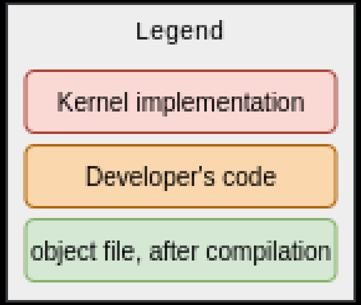


Legend
Kernel implementation
Developer's code
object file, after compilation

Summary

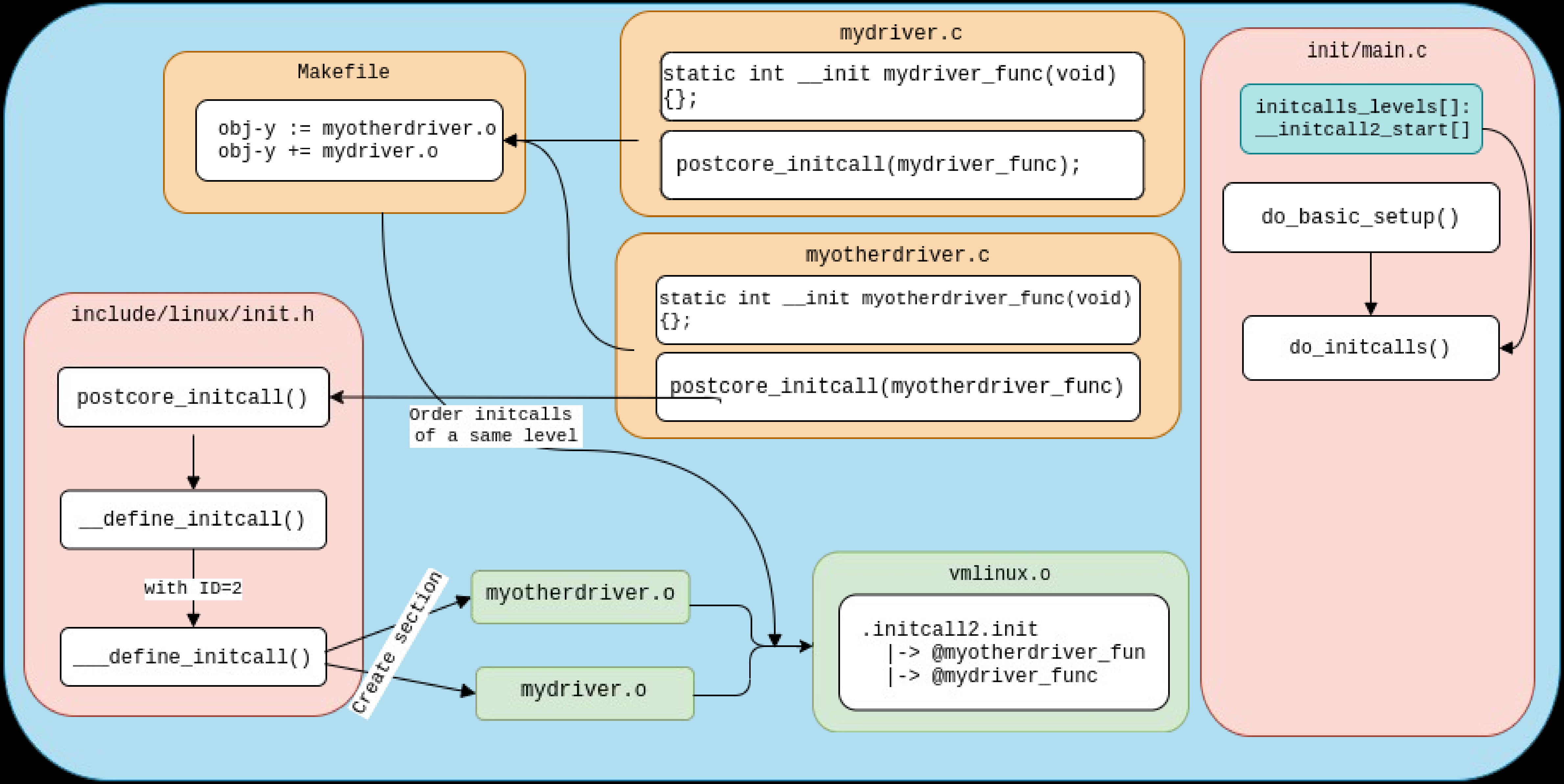


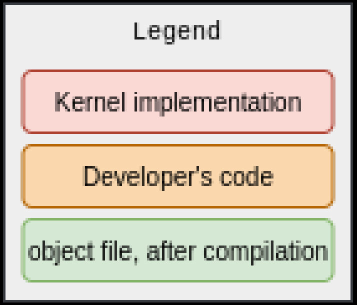
Makefile
obj-y := myotherdriver.o
obj-y += mydriver.o
mydriver.c
static int __init mydriver_func(void) {};
postcore_initcall(mydriver_func);
myotherdriver.c
static int __init myotherdriver_func(void) {};
postcore_initcall(myotherdriver_func)
init/main.c
initcalls_levels[]: __initcall2_start[]
do_basic_setup()
include/linux/init.h
postcore_initcall()
__define_initcall()
with ID=2
___define_initcall()
Order initcalls of a same level
create section
myotherdriver.o
mydriver.o
vmlinux.o
.initcall2.init
|-> @myotherdriver_fun
|-> @mydriver_func



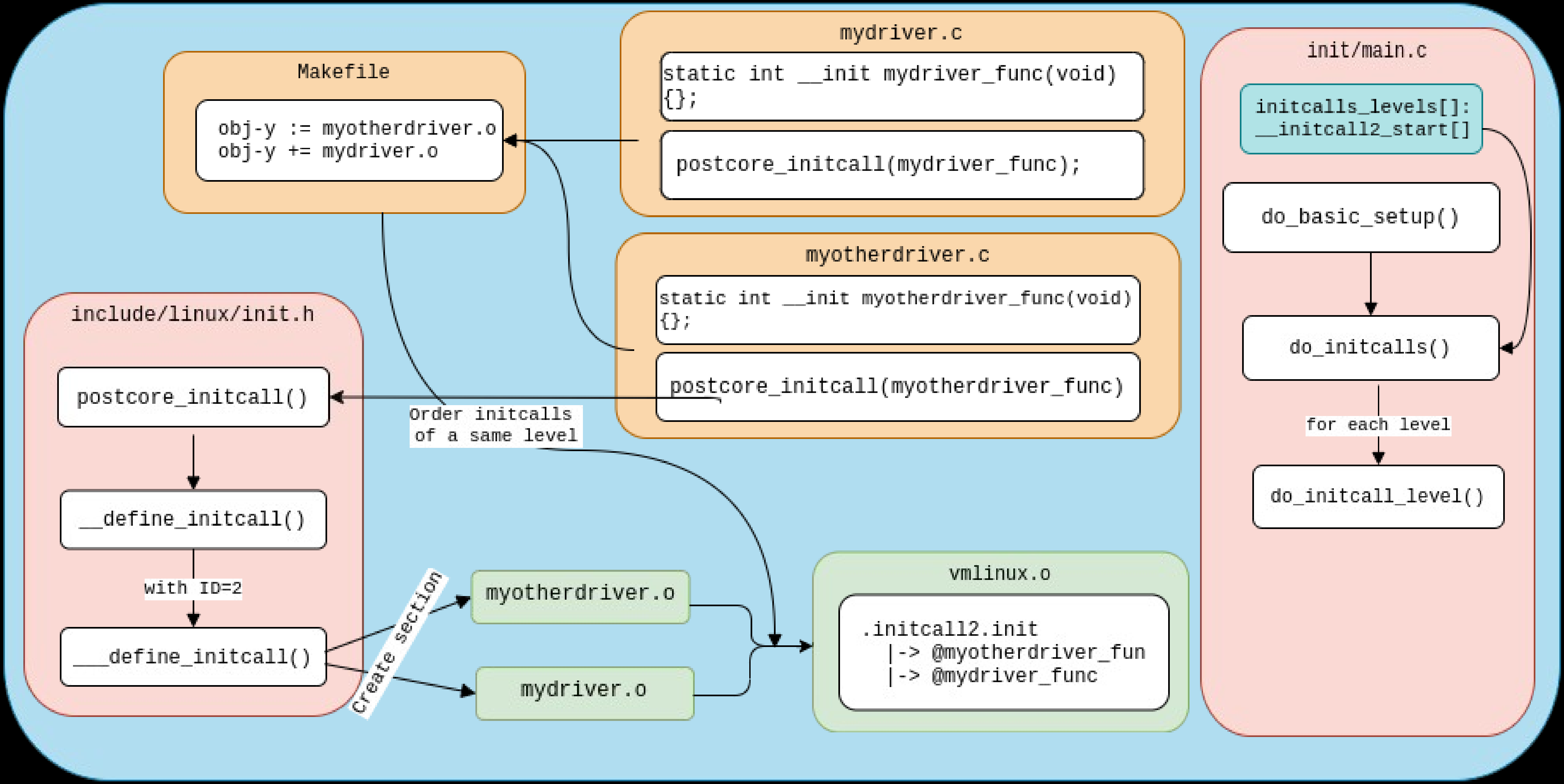
Legend
Kernel implementation
Developer's code
object file, after compilation

Summary

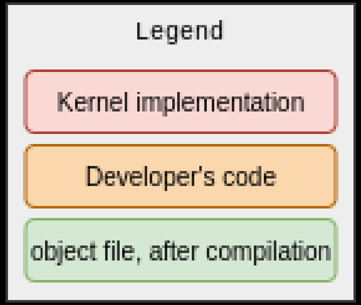

mydriver.c
static int __init mydriver_func(void)
{};
postcore_initcall(mydriver_func);
Makefile
obj-y := myotherdriver.o
obj-y += mydriver.o
init/main.c
initcalls_levels[]:
__initcall2_start[]
do_basic_setup()
do_initcalls()
myotherdriver.c
static int __init myotherdriver_func(void)
{};
postcore_initcall(myotherdriver_func)
include/linux/init.h
postcore_initcall()
Order initcalls of a same level
__define_initcall()
with ID=2
___define_initcall()
create section
myotherdriver.o
mydriver.o
vmlinux.o
.initcall2.init
|-> @myotherdriver_fun
|-> @mydriver_func


Legend
Kernel implementation
Developer's code
object file, after compilation

Summary

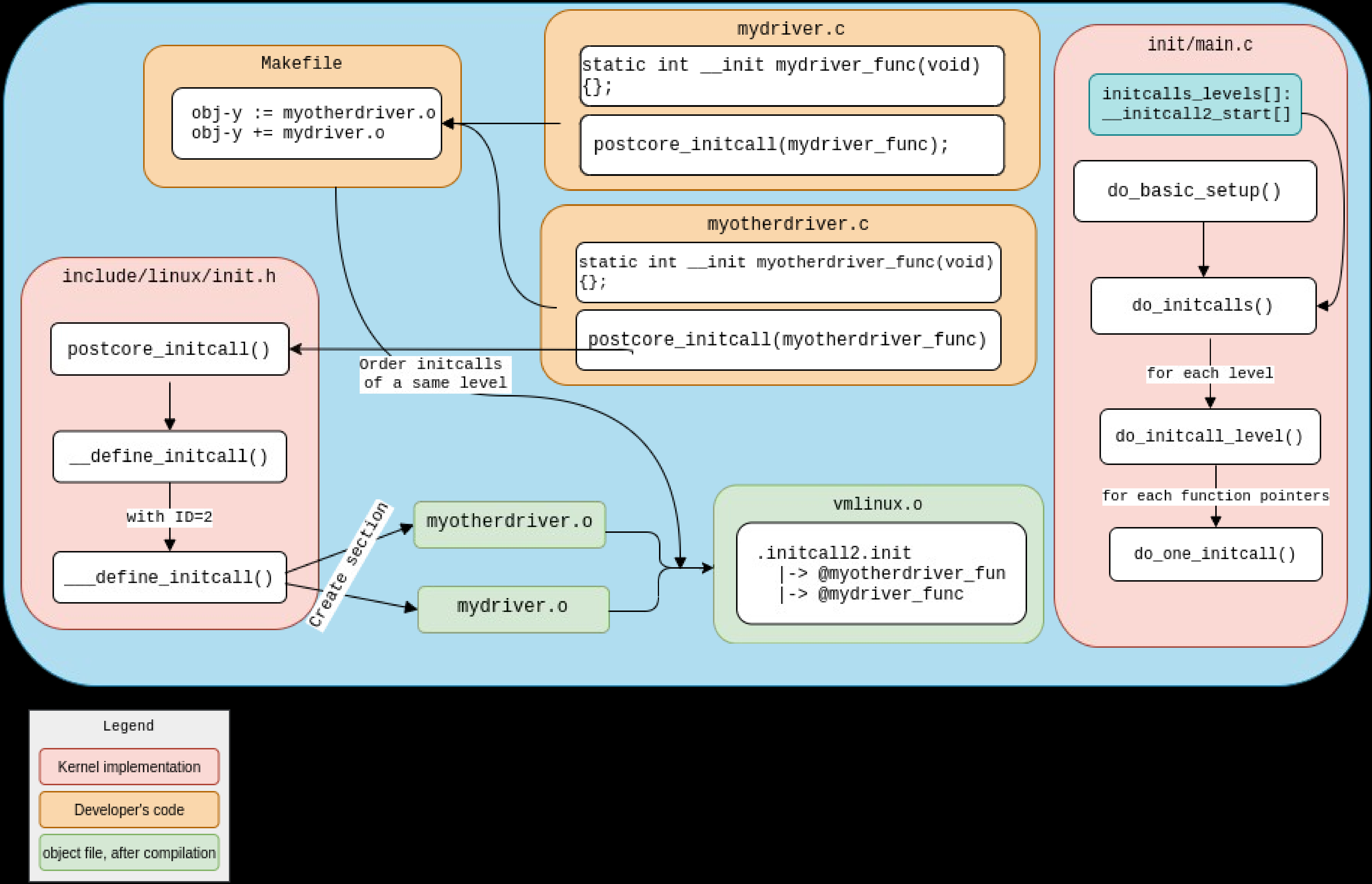


Makefile
obj-y := myotherdriver.o
obj-y += mydriver.o
mydriver.c
static int __init mydriver_func(void)
{};
postcore_initcall(mydriver_func);
myotherdriver.c
static int __init myotherdriver_func(void)
{};
postcore_initcall(myotherdriver_func)
init/main.c
initcalls_levels[]:
__initcall2_start[]
do_basic_setup()
do_initcalls()
for each level
do_initcall_level()
include/linux/init.h
postcore_initcall()
__define_initcall()
with ID=2
___define_initcall()
Order initcalls
of a same level
create section
myotherdriver.o
mydriver.o
vmlinux.o
.initcall2.init
|-> @myotherdriver_fun
|-> @mydriver_func

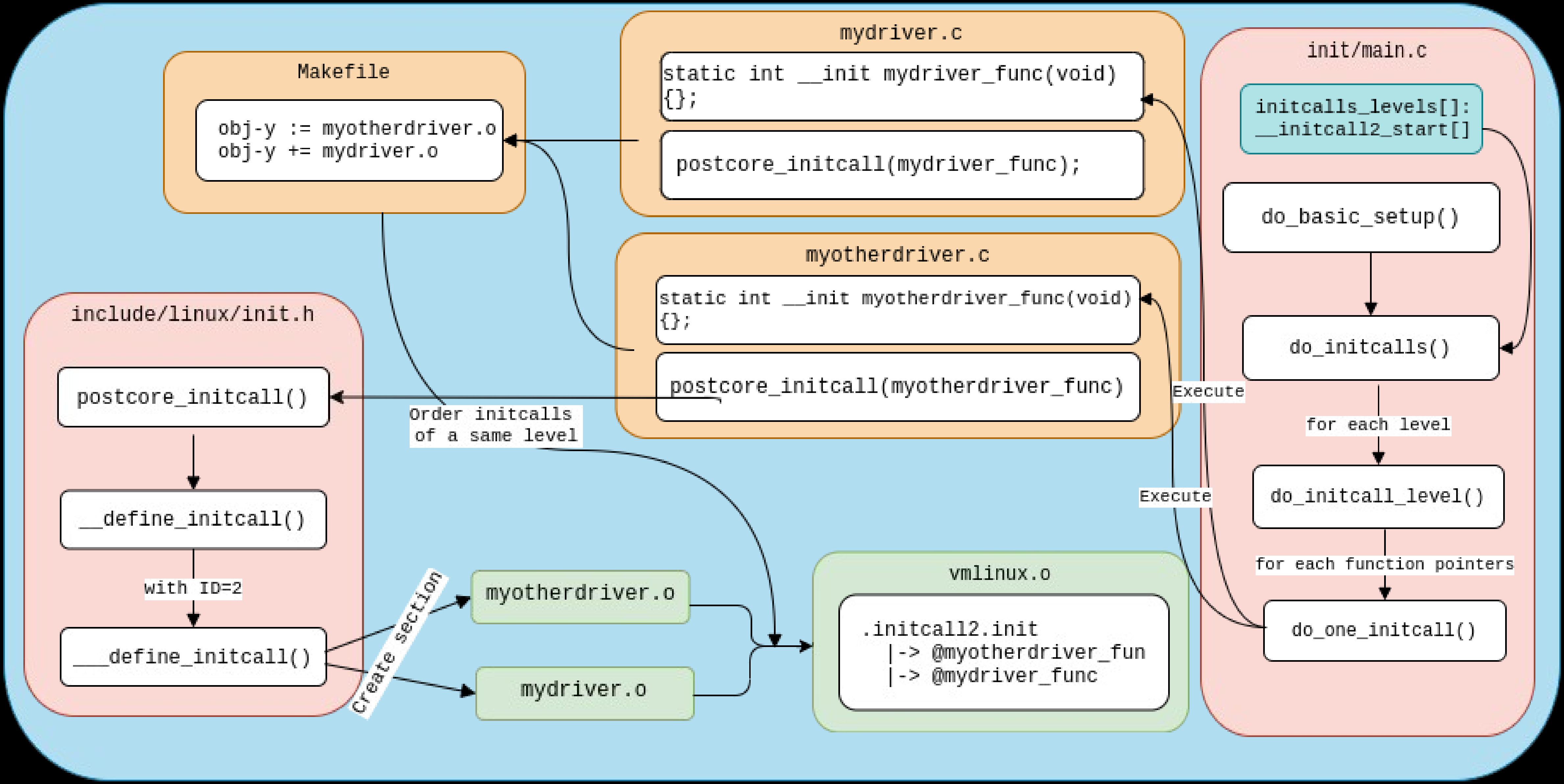


Legend
Kernel implementation
Developer's code
object file, after compilation

Summary


Makefile
obj-y := myotherdriver.o
obj-y += mydriver.o
mydriver.c
static int __init mydriver_func(void)
{};
postcore_initcall(mydriver_func);
myotherdriver.c
static int __init myotherdriver_func(void)
{};
postcore_initcall(myotherdriver_func)
include/linux/init.h
postcore_initcall()
__define_initcall()
with ID=2
___define_initcall()
Order initcalls
of a same level
create section
myotherdriver.o
mydriver.o
vmlinux.o
.initcall2.init
|-> @myotherdriver_fun
|-> @mydriver_func
init/main.c
initcalls_levels[]:
__initcall2_start[]
do_basic_setup()
do_initcalls()
for each level
do_initcall_level()
for each function pointers
do_one_initcall()
Legend
Kernel implementation
Developer's code
object file, after compilation

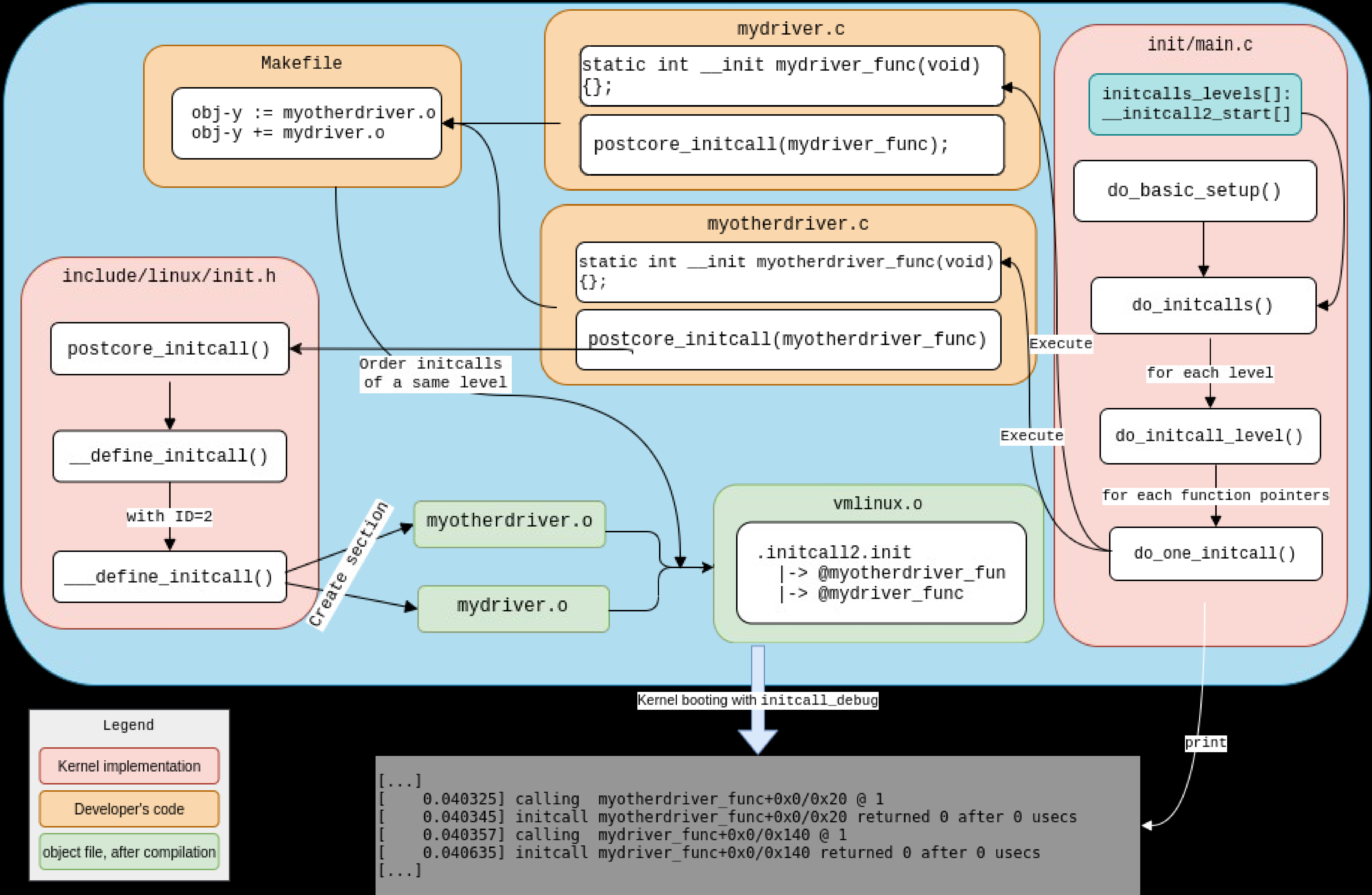
Summary



Legend

- 🟥 Kernel implementation
- 🟧 Developer's code
- 🟩 object file, after compilation

Summary



```
[...]
[    0.040325] calling  myotherdriver_func+0x0/0x20 @ 1
[    0.040345] initcall myotherdriver_func+0x0/0x20 returned 0 after 0 usecs
[    0.040357] calling  mydriver_func+0x0/0x140 @ 1
[    0.040635] initcall mydriver_func+0x0/0x140 returned 0 after 0 usecs
[...]
```

Demystifying Linux Kernel initcalls

Implementation

- ✓ General
- ✓ Ordering
  - ✓ For a particular level
  - ✓ Between all initcalls
- ✓ Execution
- ☐ Modules

Modules

- 2 different types of modules:
  - builtin modules *(‘y’ in Kconfig)*
  - Loadable modules *(‘m’ in Kconfig)*

Modules

- 2 different types of modules:
  - builtin modules *(‘y’ in Kconfig)*
  - Loadable modules *(‘m’ in Kconfig)*
- Not needed for a board to become usable

Modules

- 2 different types of modules:
  - builtin modules *('y' in Kconfig)*
  - Loadable modules *('m' in Kconfig)*
- Not needed for a board to become usable
- `module_init` may be enough → ~~early~~

Module_init - builtin

include/linux/module.h

```
#ifndef MODULE
#define module_init(x)    __initcall(x);
#define module_exit(x)    __exitcall(x);
```

Module_init - builtin

include/linux/module.h

```
#ifndef MODULE
#define module_init(x)    __initcall(x);
#define module_exit(x)    __exitcall(x);
```

- __initcall is in fact using device_initcall one

```
$ git grep __initcall include/linux/
[...]
include/linux/init.h:#define __initcall(fn)     device_initcall(fn)
```

Module_init - builtin

include/linux/module.h

```
#ifndef MODULE
#define module_init(x)    __initcall(x);
#define module_exit(x)    __exitcall(x);
```

- `__initcall` is in fact using `device_initcall` one

```
$ git grep __initcall include/linux/
[...]
include/linux/init.h:#define __initcall(fn)      device_initcall(fn)
```

- `device_initcall`: one of the last initcall executed

Module_init - loadable

include/linux/module.h

```
#else /* MODULE */
#define early_initcall(fn)      module_init(fn)
#define core_initcall(fn)       module_init(fn)
#define postcore_initcall(fn)   module_init(fn)
#define arch_initcall(fn)       module_init(fn)
#define subsys_initcall(fn)     module_init(fn)
#define fs_initcall(fn)         module_init(fn)
#define rootfs_initcall(fn)     module_init(fn)
#define device_initcall(fn)     module_init(fn)
#define late_initcall(fn)       module_init(fn)
[...]
```

Module_init - loadable

include/linux/module.h

```
#else /* MODULE */
#define early_initcall(fn)      module_init(fn)
#define core_initcall(fn)       module_init(fn)
#define postcore_initcall(fn)   module_init(fn)
#define arch_initcall(fn)       module_init(fn)
#define subsys_initcall(fn)     module_init(fn)
#define fs_initcall(fn)         module_init(fn)
#define rootfs_initcall(fn)     module_init(fn)
#define device_initcall(fn)     module_init(fn)
#define late_initcall(fn)       module_init(fn)
[...]


#define module_init(initfn)                                     \
     static inline initcall_t __maybe_unused __inittest(void)      \
     { return initfn; }                                         \
     int init_module(void) __copy(initfn) __attribute__((alias(#initfn)));
```

Module_init - loadable

include/linux/module.h

```
#else /* MODULE */
#define early_initcall(fn)      module_init(fn)
#define core_initcall(fn)       module_init(fn)
#define postcore_initcall(fn)   module_init(fn)
#define arch_initcall(fn)       module_init(fn)
#define subsys_initcall(fn)     module_init(fn)
#define fs_initcall(fn)         module_init(fn)
#define rootfs_initcall(fn)     module_init(fn)
#define device_initcall(fn)     module_init(fn)
#define late_initcall(fn)       module_init(fn)
[...]


#define module_init(initfn)                                     \
     static inline initcall_t __maybe_unused __inittest(void)      \
     { return initfn; }                                         \
     int init_module(void) __copy(initfn) __attribute__((alias(#initfn)));
```

- init_module: Creating an alias to our function

Module_init - loadable

- Additional code into a C module file

```
.init = init_module
```

Module_init - loadable

- Additional code into a C module file

```
.init = init_module
```

scripts/mod/modpost.c

```
static void add_header(struct buffer *b, struct module *mod)
{
      [...]
      buf_printf(b, "MODULE_INFO(name, KBUILD_MODNAME);\n");
      if (mod->has_init)
         buf_printf(b, "\t.init = init_module,\n");
      [...]
}
```

Module_init - loadable

- Additional code into a C module file

```
.init = init_module
```

scripts/mod/modpost.c

```
static void add_header(struct buffer *b, struct module *mod)
{
      [...]
      buf_printf(b, "MODULE_INFO(name, KBUILD_MODNAME);\n");
      if (mod->has_init)
         buf_printf(b, "\t.init = init_module,\n");
      [...]
}
```

kernel/module.c

```
static noinline int do_init_module(struct module *mod)
{
[...]
        /* Start the module */
        if (mod->init != NULL)
                ret = do_one_initcall(mod->init);
[...]
```

Module_init function

- Builtin: Execution at device level

Module_init function

- Builtin: Execution at device level
- Loadable: Execute at module’s insertion

Module_init function

- Builtin: Execution at device level
- Loadable: Execute at module’s insertion
- If no reason to execute a function at early stage of boot process => use module_init

Module_init function

- Builtin: Execution at device level
- Loadable: Execute at module's insertion
- If no reason to execute a function at early stage of boot process => use module_init
- Benefit: save time consumed at boot

Module_init function

- Builtin: Execution at device level
- Loadable: Execute at module's insertion
- If no reason to execute a function at early stage of boot process => use module_init
- Benefit: save time consumed at boot

Let most important functions being executed earlier

Thank you!

Questions?

Mylène Josserand

mylene.josserand@collabora.com