04-2 Adding to the Kernel, Kernel Initialization

Adding to the Kernel

- Makefile Targets
- Kernel Configuration
- Custom Configuration Options
- Kernel Makefiles
- Kernel Documentation

Composite Kernel Image

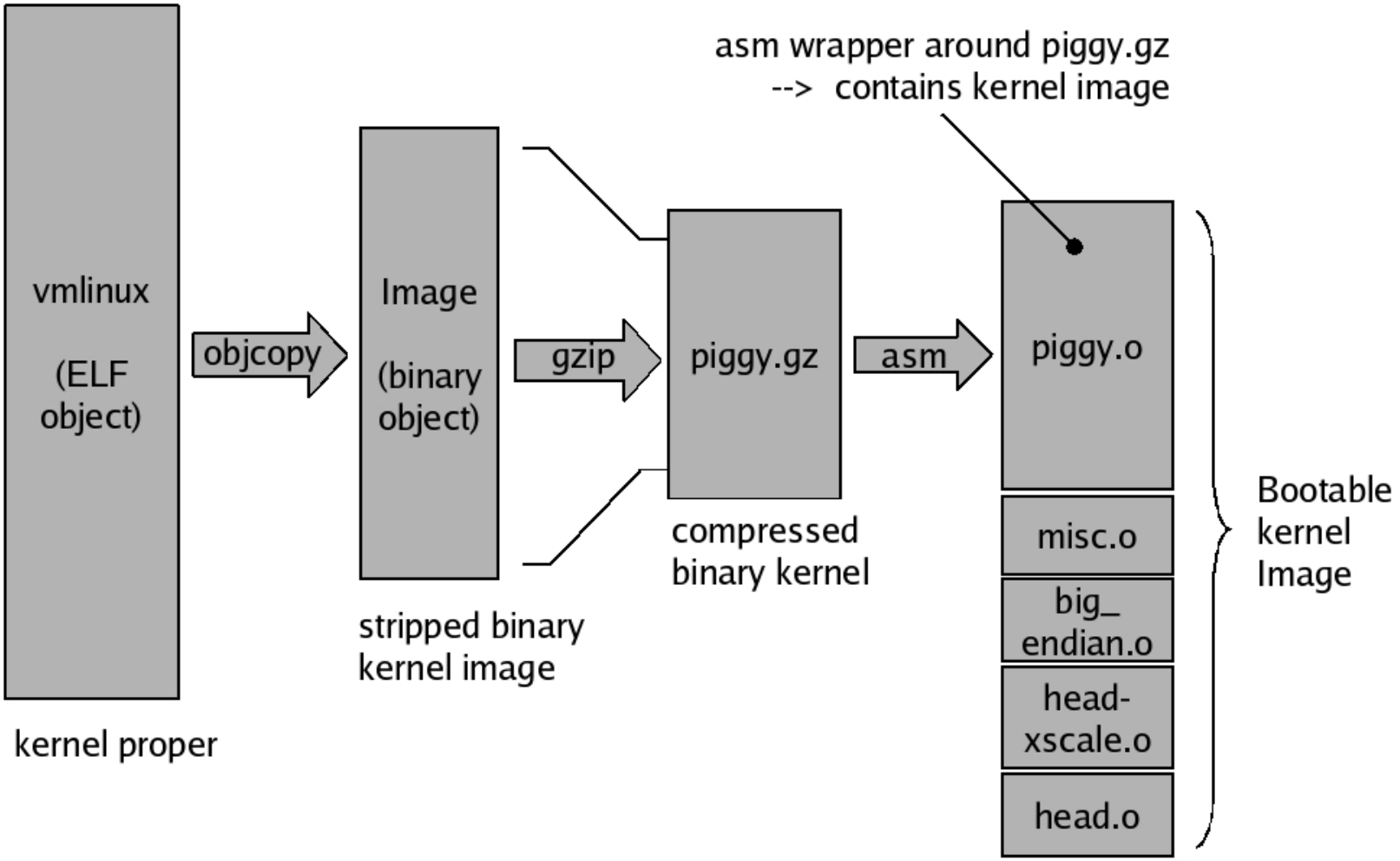


Figure 5.1 page 103

piggy.S

3.2 kernel

```
  .section .piggydata,#alloc
  .globl    input_data
input_data:
  .incbin   "arch/arm/boot/compressed/piggy.gz"
  .globl    input_data_end
input_data_end:
```

How do you find this file?

```
host$ cd bb-kernel/KERNEL
host$ find . –iname piggy.s
```

```
.incbin "arch/arm/boot/compressed/piggy_data"
```

Compiling Kernel

```
host$ source ~/crossCompileEnv.sh
host$ make -j3 uImage
... < many build steps omitted for clarity >
AS      arch/arm/boot/compressed/head.o
  XZKERN  arch/arm/boot/compressed/piggy.xzkern
…
  AS      arch/arm/boot/compressed/piggy.xzkern.o
  LD      arch/arm/boot/compressed/vmlinux
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
  UIMAGE  arch/arm/boot/uImage
Image Name:   Linux-3.8.13+
Created:      Thu Oct  3 17:13:18 2013
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    2898464 Bytes = 2830.53 kB = 2.76 MB
Load Address: 80008000
Entry Point:  80008000
  Image arch/arm/boot/uImage is ready
```

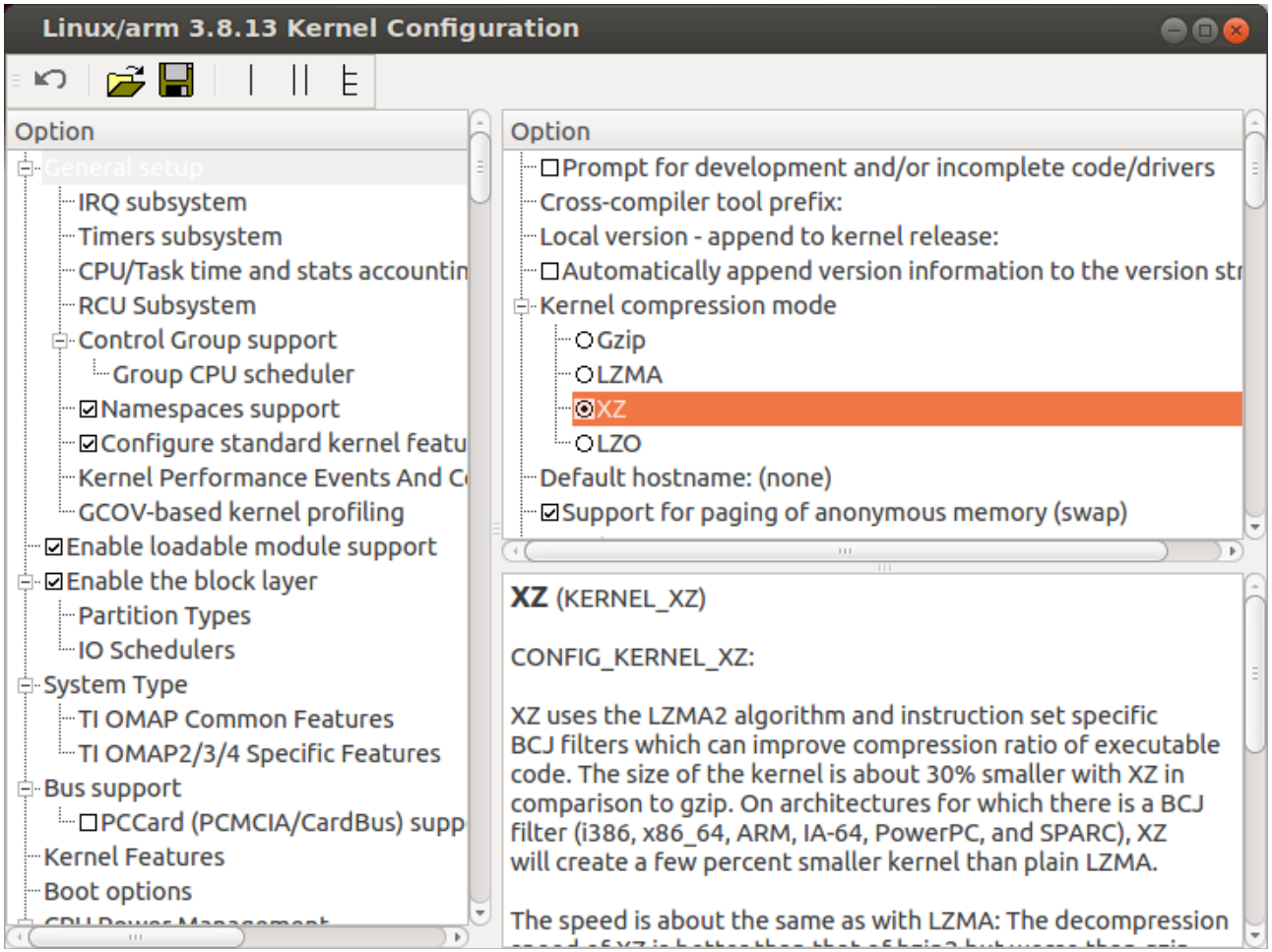
.../arch/arm/boot/compressed

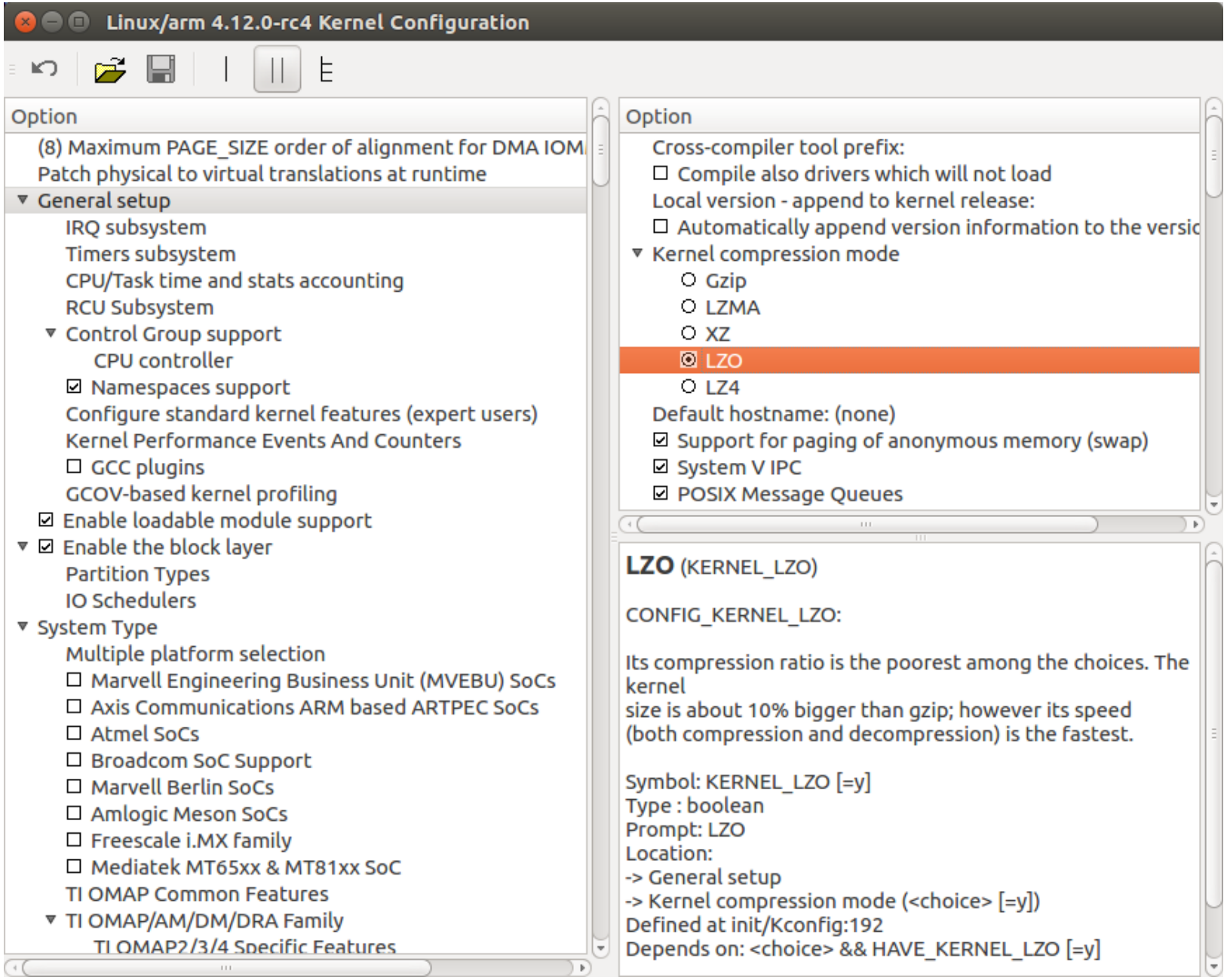
```
host$ ls
ashldi3.o          hyp-stub.o         piggy.lzo.S
ashldi3.S          hyp-stub.S         piggy.xzkern
atags_to_fdt.c     lib1funcs.o        piggy.xzkern.o
big-endian.S       lib1funcs.S        piggy.xzkern.S
decompress.c       libfdt_env.h       sdhi-sh7372.c
decompress.o       ll_char_wr.S       sdhi-shmobile.c
head.o             Makefile           sdhi-shmobile.h
head.S             misc.c             string.c
head-sa1100.S      misc.o             string.o
head-shark.S       mmcif-sh7372.c     vmlinux
head-sharpsl.S     ofw-shark.c        vmlinux.lds
head-shmobile.S    piggy.gzip.S       vmlinux.lds.in
head-xscale.S      piggy.lzma.S
```

piggy.xzkern.S

```
  .section .piggydata,#alloc
  .globl    input_data
input_data:
  .incbin
      "arch/arm/boot/compressed/piggy.xzkern"
  .globl    input_data_end
input_data_end:
```

How does it know to use kernxz?




Linux/arm 4.12.0-rc4 Kernel Configuration
Option
(8) Maximum PAGE_SIZE order of alignment for DMA IOM
Patch physical to virtual translations at runtime
General setup
IRQ subsystem
Timers subsystem
CPU/Task time and stats accounting
RCU Subsystem
Control Group support
CPU controller
Namespaces support
Configure standard kernel features (expert users)
Kernel Performance Events And Counters
GCC plugins
GCOV-based kernel profiling
Enable loadable module support
Enable the block layer
Partition Types
IO Schedulers
System Type
Multiple platform selection
Marvell Engineering Business Unit (MVEBU) SoCs
Axis Communications ARM based ARTPEC SoCs
Atmel SoCs
Broadcom SoC Support
Marvell Berlin SoCs
Amlogic Meson SoCs
Freescale i.MX family
Mediatek MT65xx & MT81xx SoC
TI OMAP Common Features
TI OMAP/AM/DM/DRA Family
TI OMAP2/3/4 Specific Features
Option
Cross-compiler tool prefix:
Compile also drivers which will not load
Local version - append to kernel release:
Automatically append version information to the versic
Kernel compression mode
Gzip
LZMA
XZ
LZO
LZ4
Default hostname: (none)
Support for paging of anonymous memory (swap)
System V IPC
POSIX Message Queues
LZO (KERNEL_LZO)
CONFIG_KERNEL_LZO:
Its compression ratio is the poorest among the choices. The kernel
size is about 10% bigger than gzip; however its speed
(both compression and decompression) is the fastest.
Symbol: KERNEL_LZO [=y]
Type : boolean
Prompt: LZO
Location:
-> General setup
-> Kernel compression mode (<choice> [=y])
Defined at init/Kconfig:192
Depends on: <choice> && HAVE_KERNEL_LZO [=y]

Bootstrap Loader (not bootloader)

- Provide context for kernel
  - Enable instruction set
  - Data caches
  - Disable interrupt
  - C runtime environment
- Decompress (misc.o)
- Relocate kernel image

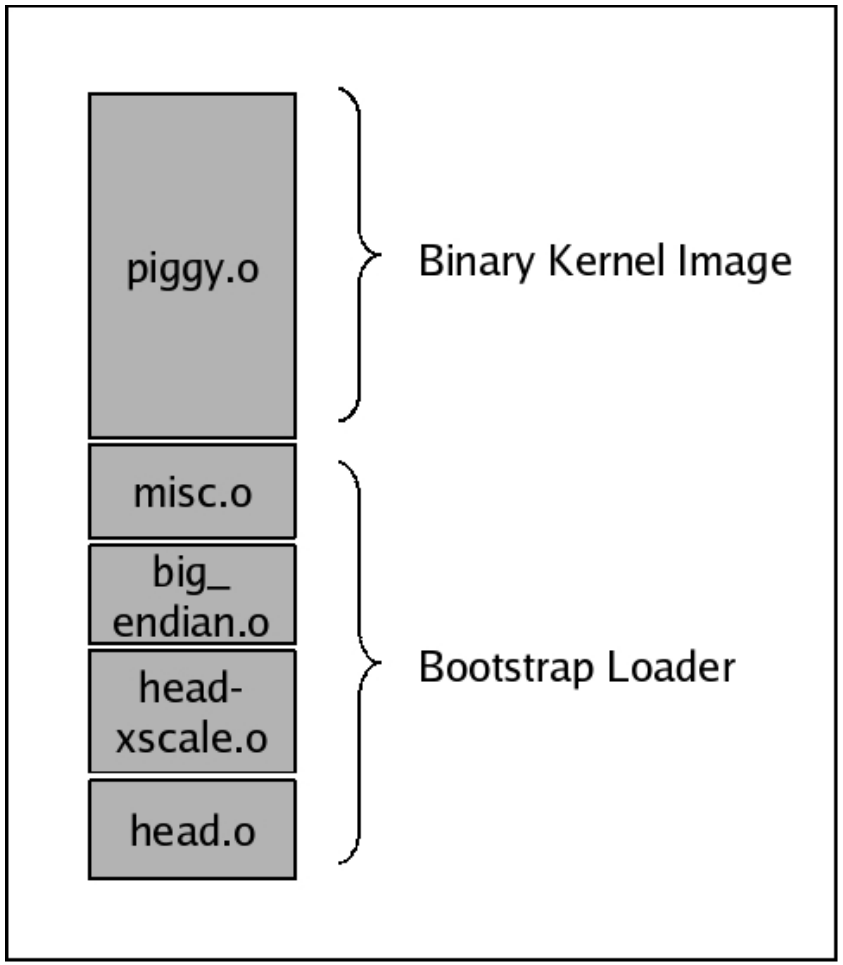


Figure 5-2 page 105

Bootstrap Loader (not bootloader)

```
 LD      vmlinux
  SORTEX  vmlinux
sort done marker at 81c420
  SYSMAP  System.map
  OBJCOPY arch/arm/boot/Image
  Kernel: arch/arm/boot/Image is ready
  AS      arch/arm/boot/compressed/head.o
  XZKERN  arch/arm/boot/compressed/piggy.xzkern
  CC      arch/arm/boot/compressed/misc.o
  CC      arch/arm/boot/compressed/decompress.o
  CC      arch/arm/boot/compressed/string.o
  SHIPPED arch/arm/boot/compressed/hyp-stub.S
  AS      arch/arm/boot/compressed/lib1funcs.o
  AS      arch/arm/boot/compressed/ashldi3.o
  AS      arch/arm/boot/compressed/hyp-stub.o
  AS      arch/arm/boot/compressed/piggy.xzkern.o
  LD      arch/arm/boot/compressed/vmlinux
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
  UIMAGE  arch/arm/boot/uImage
```

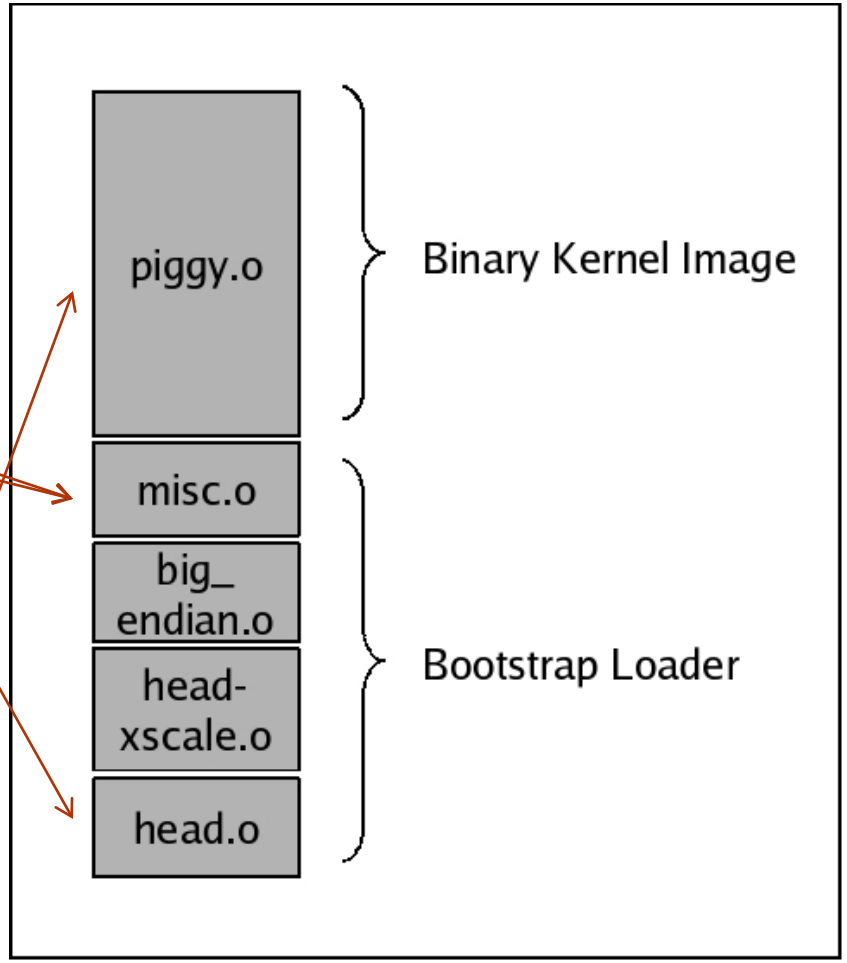
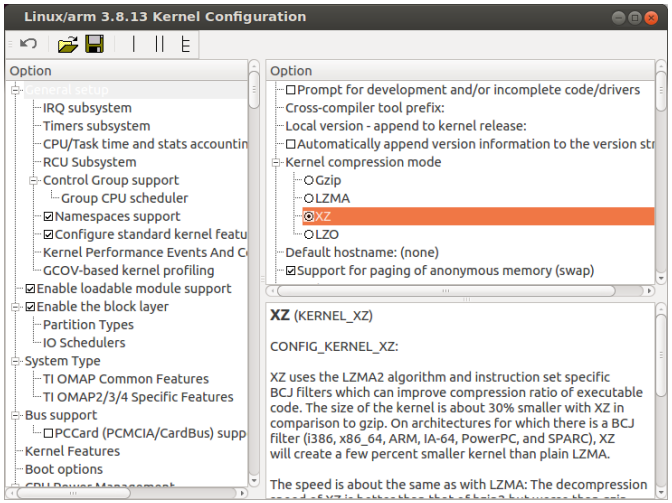


Figure 5-2 page 105

decompress.c

```
#ifdef CONFIG_KERNEL_GZIP
#include "../../../../lib/decompress_inflate
#endif

#ifdef CONFIG_KERNEL_LZO
#include "../../../../lib/decompress_unlzo.c"
#endif

#ifdef CONFIG_KERNEL_LZMA
#include "../../../../lib/decompress_unlzma.c"
#endif

#ifdef CONFIG_KERNEL_XZ
#define memmove memmove
#define memcpy memcpy
#include "../../../../lib/decompress_unxz.c"
#endif
```

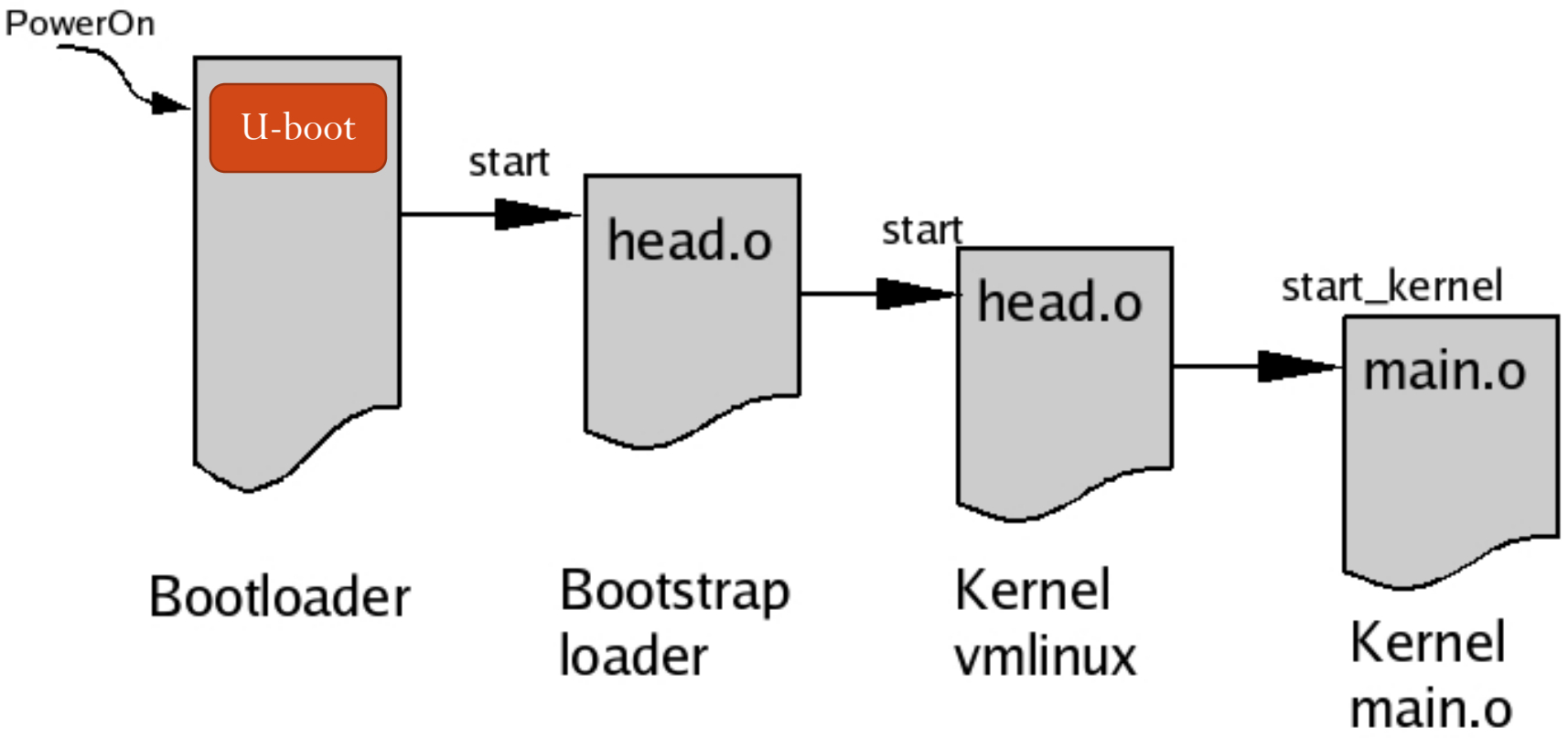


Boot Messages

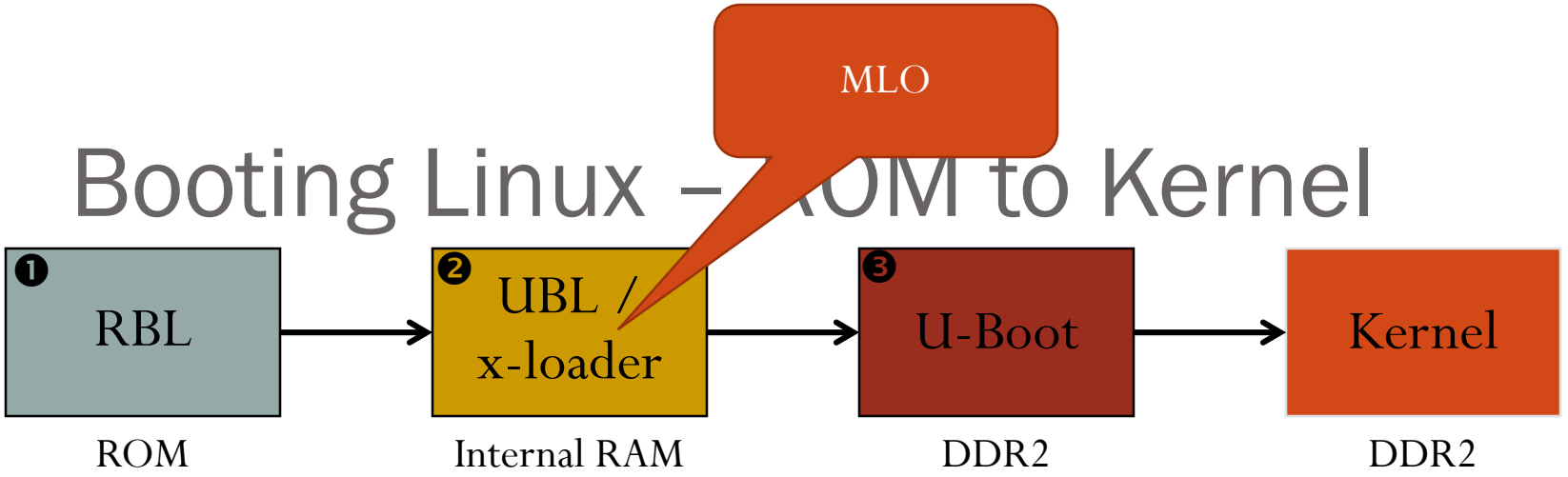
- See handout
- Note *kernel version string*
- Note *kernel command line*
- *EBC Boot Sequence* shows how to display the messages in the handout

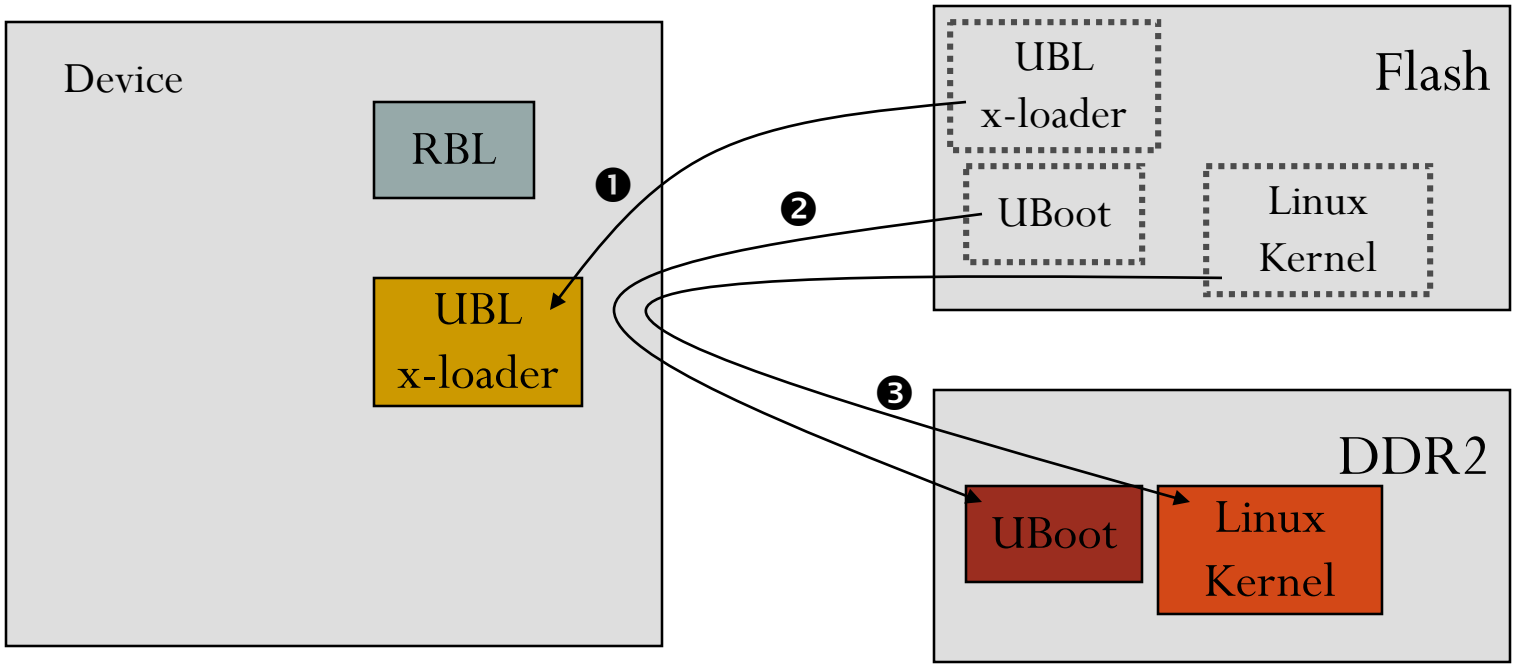
```
bone$ cd /boot
bone$ ls -F
config-4.4.15-bone11      initrd.img-4.4.21-ti-r47  vmlinuz-4.4.15-bone11*
config-4.4.19-ti-r41      SOC.sh                    vmlinuz-4.4.19-ti-r41*
config-4.4.21-ti-r47      System.map-4.4.19-ti-r41  vmlinuz-4.4.21-ti-r47*
config-4.4.22-bone13.1    System.map-4.4.21-ti-r47  vmlinuz-4.4.22-bone13.1*
dtbs/                     uboot/
initrd.img-4.4.19-ti-r41  uEnv.txt
bone$ cat uEnv.txt
#Docs: http://elinux.org/Beagleboard:U-boot_partitioning_layout_2.0
uname_r=4.4.22-bone13.1
…
cmdline=coherent_pool=1M quiet cape_universal=enable
```

5-3 ARM boot control flow



Booting Linux – ROM to Kernel



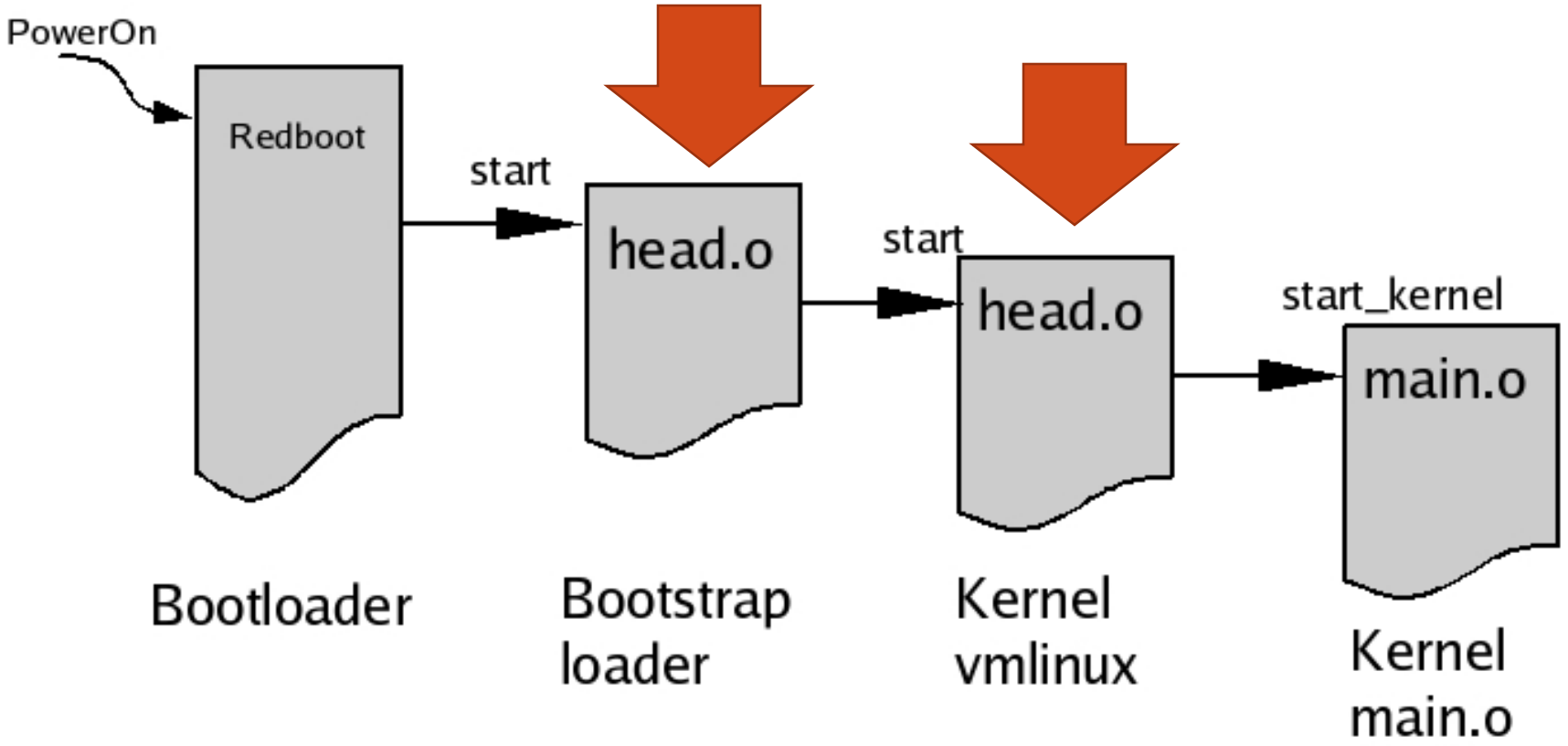


.../arch/arm/boot/compressed/head.S

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

#ifdef DEBUG

#if defined(CONFIG_DEBUG_ICEDCC)

#ifdef CONFIG_CPU_V7
        .macro  loadsp, rb
        .endm
        .macro  writeb, ch, rb
        mcr     p14, 0, \ch, c0, c5, 0
        .endm
#else
        .macro  loadsp, rb
        .endm
        .macro  writeb, ch, rb
        mcr     p14, 0, \ch, c1, c0, 0
        .endm
#endif
#
#else

#include <mach/debug-macro.S>

        .macro  writeb,  ch, rb
        senduart \ch, \rb
        .endm

#if defined(CONFIG_ARCH_SA1100)
        .macro  loadsp, rb
        mov     \rb, #0x80000000    @
    physical base address
```

How do you find the value?

2 head.o's


PowerOn
Redboot
start
head.o
start
head.o
start_kernel
main.o
Bootloader
Bootstrap loader
Kernel vmlinux
Kernel main.o

.../arch/arm/kernel/head.S

1. Checks for valid processor and architecture
2. Creates initial page table entries
3. Enables the processor's memory management unit (MMU)
4. Establishes limited error detection and reporting
5. Jumps to the start of the kernel proper, **`start_kernel()`** in **`main.c`**.

Find these on the handout

.../arch/arm/kernel/head.S

```
/*
 * Kernel startup entry point.
 * ---------------------------
 *
 * This is normally called from the decompressor code.  The requirements
 * are: MMU = off, D-cache = off, I-cache = dont care, r0 = 0,
 * r1 = machine nr, r2 = atags or dtb pointer.
 *
 * This code is mostly position independent, so if you link the kernel at
 * 0xc0008000, you call this at __pa(0xc0008000).
 *
 * See linux/arch/arm/tools/mach-types for the complete list of machine
 * numbers for r1.
 *
 * We're trying to keep crap to a minimum; DO NOT add any machine specific
 * crap here - that's what the boot loader (or in extreme, well justified
 * circumstances, zImage) is for.
 */
```

Kernel Startup

- **arch/arm/kernel/head.S**

```
b start_kernel
```

.../init/main.c

```
asmlinkage __visible void __init start_kernel(void)
{
        char *command_line;
        char *after_dashes;

        /*
         * Need to run as early as possible, to initialize the
         * lockdep hash:
         */
        lockdep_init();
        set_task_stack_end_magic(&init_task);
        smp_setup_processor_id();
        debug_objects_early_init();

        /*
         * Set up the the initial canary ASAP:
         */
        boot_init_stack_canary();

        cgroup_init_early();

        local_irq_disable();
        early_boot_irqs_disabled = true;
```

Kernel Command Line Processing

- Kernel Command-Line Processing
- The `__setup` macro

```
console=tty0 console=ttyO0,115200n8
root=/dev/mmcblk0p1 rootfstype=ext4
rootwait coherent_pool=1M
cape_universal=enable
```

Console Setup Code Snippet

```
/*
* Setup a list of consoles. Called from init/main.c
```

```
#ifdef CONFIG_SERIAL_OMAP
        if (!strncmp(str, "tty0", 4) && '0' <= str[4] && '9' >= str[4]) {
                str[3] = 'O';
                pr_warn("We are opening your eyes, assuming you want to
use an OMAP based serial driver and not a zeroMAP based one! ;)\n");
                pr_warn("Which means 'tty0%s' was changed to 'ttyO%s'
automagically for your pleasure.\n", str+4, str+4);
        }
#endif
```

```
...
return 1;
}
__setup("console=", console_setup);
```

Registration function

From …/kernel/printk/printk.c

Console Setup Code Snippet

```
/*
* Setup a list of consoles. Called from init/main.c
*/
static int __init console_setup(char *str)
{
        char buf[sizeof(console_cmdline[0].name) + 4]; /* 4 for "ttyS" */
        char *s, *options, *brl_options = NULL;
        int idx;
        if (_braille_console_setup(&str, &brl_options))
                return 1;

<body omitted for clarity...>
...
return 1;
}
__setup("console=", console_setup);
```

From …/kernel/printk/printk.c

.../include/linux/init.h

```
/*
 * Only for really core code.  See moduleparam.h for the normal way.
 *
 * Force the alignment so the compiler doesn't space elements of the
 * obs_kernel_param "array" too far apart in .init.setup.
 */
#define __setup_param(str, unique_id, fn, early)            \
   static char __setup_str_##unique_id[] __initdata __aligned(1) = str;
                                                            \
   static struct obs_kernel_param __setup_##unique_id       \
         __used __section(.init.setup)                      \
         __attribute__((aligned((sizeof(long)))))           \
         = { __setup_str_##unique_id, fn, early }

#define __setup(str, fn)                                    \
   __setup_param(str, fn, fn, 0)
```

__setup

```
__setup("console=", console_setup);
```

- Expands to

```
static const char __setup_str_console_setup[] __initconst \
__aligned(1) = "console=";
static struct obs_kernel_param __setup_console_setup __used \
__section(.init.setup) __attribute__
((aligned((sizeof(long))))) \
= { __setup_str_console_setup, console_setup, early};
```

- Which expands to

```
static struct obs_kernel_param __setup_console_setup \
__section(.init.setup) = { __setup_str_console_setup,
console_setup, early};
```

- This stores the code in a table in section **.init.setup.**

On initialization...

- The table in **`.init.setup`** has
  - Parameter string ("**`console=`**") and
  - Pointer to the function that processes it.
- This way the initialization code can process everything on the command line without knowing at compile time where all the code is.
- See section 5.3 of *Embedded Linux Primer* for more details.