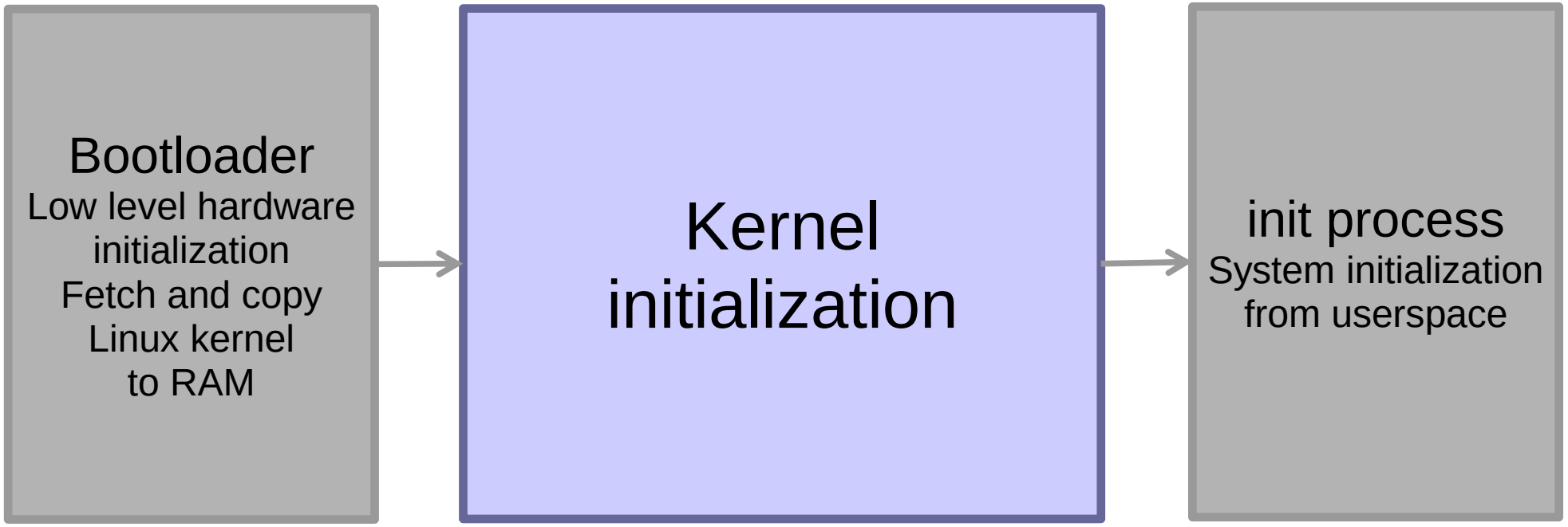
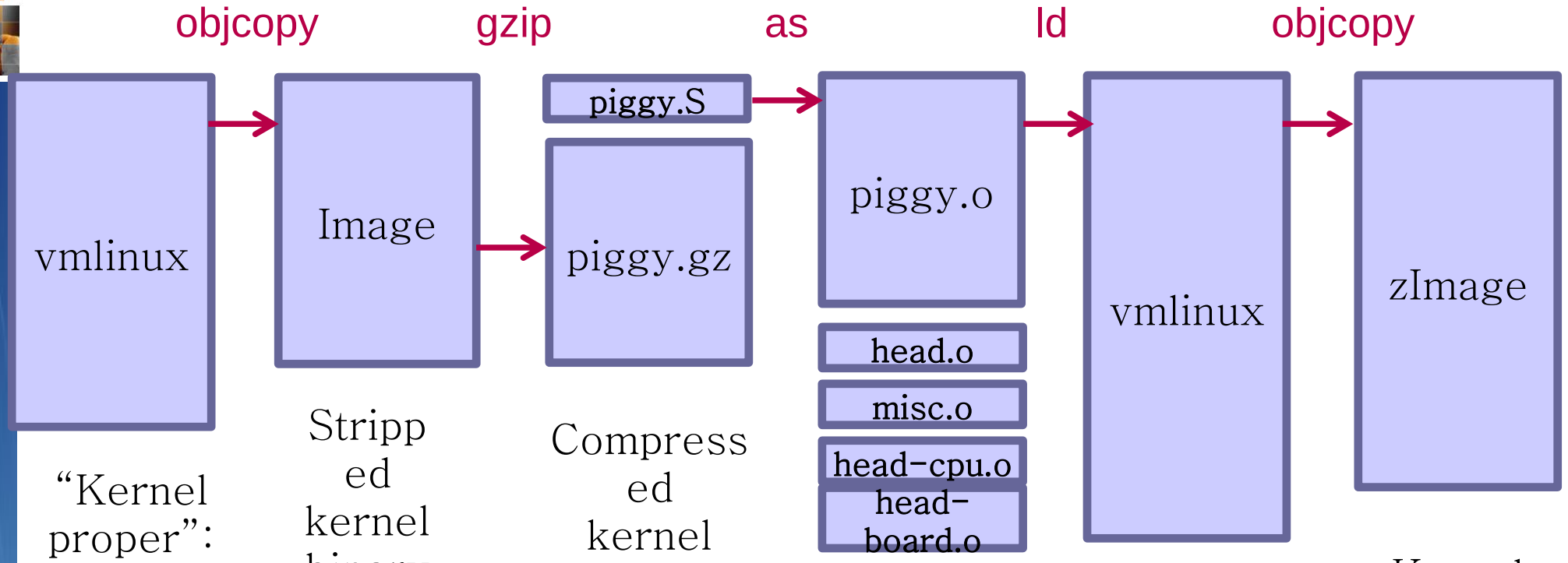
Kernel Initialization

VEDA Solutions
Inspired by open Source

From bootloader to userspace



Kernel bootstrap



"Kernel proper": Raw kernel executable (ELF object)

Stripped kernel binary (binary object)

Compressed kernel binary

asm wrapper around piggy.gz + bootstrap code

(in arch/<arch>boot/compressed)

Composite kernel image (ELF object)

Kernel image for bootloader (binary object)

Details found by compiling with make V=1

Bootstrap code

- head.o:

Architecture specific initialization code.
This is what is executed by the bootloader

- head-cpu.o

CPU specific initialization code

- head-board.o

Board specific initialization code

- misc.o:

Decompression routines

Bootstrap code tasks

Main work done by head.o:

- Check the architecture, processor and machine type.
- Configure the MMU, create page table entries and enable virtual memory.
- Calls the start_kernel function in init/main.c.

Same code for all architectures.
Anybody interesting in kernel startup should study this file!

start_kernel main actions

- Calls setup_arch(&command_line) (function defined in arch/<arch>/kernel/setup.c), copying the command line from where the bootloader left it.
- On arm, this function calls setup_processor (in which CPU information is displayed) and setup_machine(locating the machine in the list of supported machines).
- Initializes the console as early as possible (to get error messages)
- Initializes many subsystems (see the code)
- Eventually calls rest_init.

rest_init: starting the init process

Starting a new kernel thread which will later become the init process

```
static void noinline __init_refok rest_init(void)
        __releases(kernel_lock)
{
        int pid;

        kernel_thread(kernel_init, NULL, CLONE_FS | CLONE_SIGHAND);
        numa_default_policy();
        pid = kernel_thread(kthreadd, NULL, CLONE_FS | CLONE_FILES);
        kthreadd_task = find_task_by_pid(pid);
        unlock_kernel();

        /*
         * The boot idle thread must execute schedule()
         * at least one to get things moving:
         */
        preempt_enable_no_resched();
        schedule();
        preempt_disable();

        /* Call into cpu_idle with preempt disabled */
        cpu_idle();
}
```

Source: Linux 2.6.22

kernel_init

kernel_init does 2 main things:

- Call do_basic_setup

Now that kernel services are ready, start device initialization:

```
static void __init do_basic_setup(void)
{
        /* drivers will send hotplug events */
        init_workqueues();
        usermodehelper_init();
        driver_init();
        init_irq_proc();
        do_initcalls();
}
```

- Call init_post

do_initcalls

Calls pluggable hooks registered with the macros below.
Advantage: the generic code doesn't have to know about them.

```
/*
 * A "pure" initcall has no dependencies on anything else, and purely
 * initializes variables that couldn't be statically initialized.
 *
 * This only exists for built-in code, not for modules.
 */
#define pure_initcall(fn)               __define_initcall("0",fn,1)

#define core_initcall(fn)               __define_initcall("1",fn,1)
#define core_initcall_sync(fn)          __define_initcall("1s",fn,1s)
#define postcore_initcall(fn)           __define_initcall("2",fn,2)
#define postcore_initcall_sync(fn)      __define_initcall("2s",fn,2s)
#define arch_initcall(fn)               __define_initcall("3",fn,3)
#define arch_initcall_sync(fn)          __define_initcall("3s",fn,3s)
#define subsys_initcall(fn)             __define_initcall("4",fn,4)
#define subsys_initcall_sync(fn)        __define_initcall("4s",fn,4s)
#define fs_initcall(fn)                 __define_initcall("5",fn,5)
#define fs_initcall_sync(fn)            __define_initcall("5s",fn,5s)
#define rootfs_initcall(fn)             __define_initcall("rootfs",fn,rootfs)
#define device_initcall(fn)             __define_initcall("6",fn,6)
#define device_initcall_sync(fn)        __define_initcall("6s",fn,6s)
#define late_initcall(fn)               __define_initcall("7",fn,7)
#define late_initcall_sync(fn)          __define_initcall("7s",fn,7s)
```

initcall example

From arch/arm/mach-pxa/lpd270.c

```
static int __init lpd270_irq_device_init(void)
{
        int ret = sysdev_class_register(&lpd270_irq_sysclass);
        if (ret == 0)
                ret = sysdev_register(&lpd270_irq_device);
        return ret;
}

device_initcall(lpd270_irq_device_init);
```

init_post

The last step of Linux booting

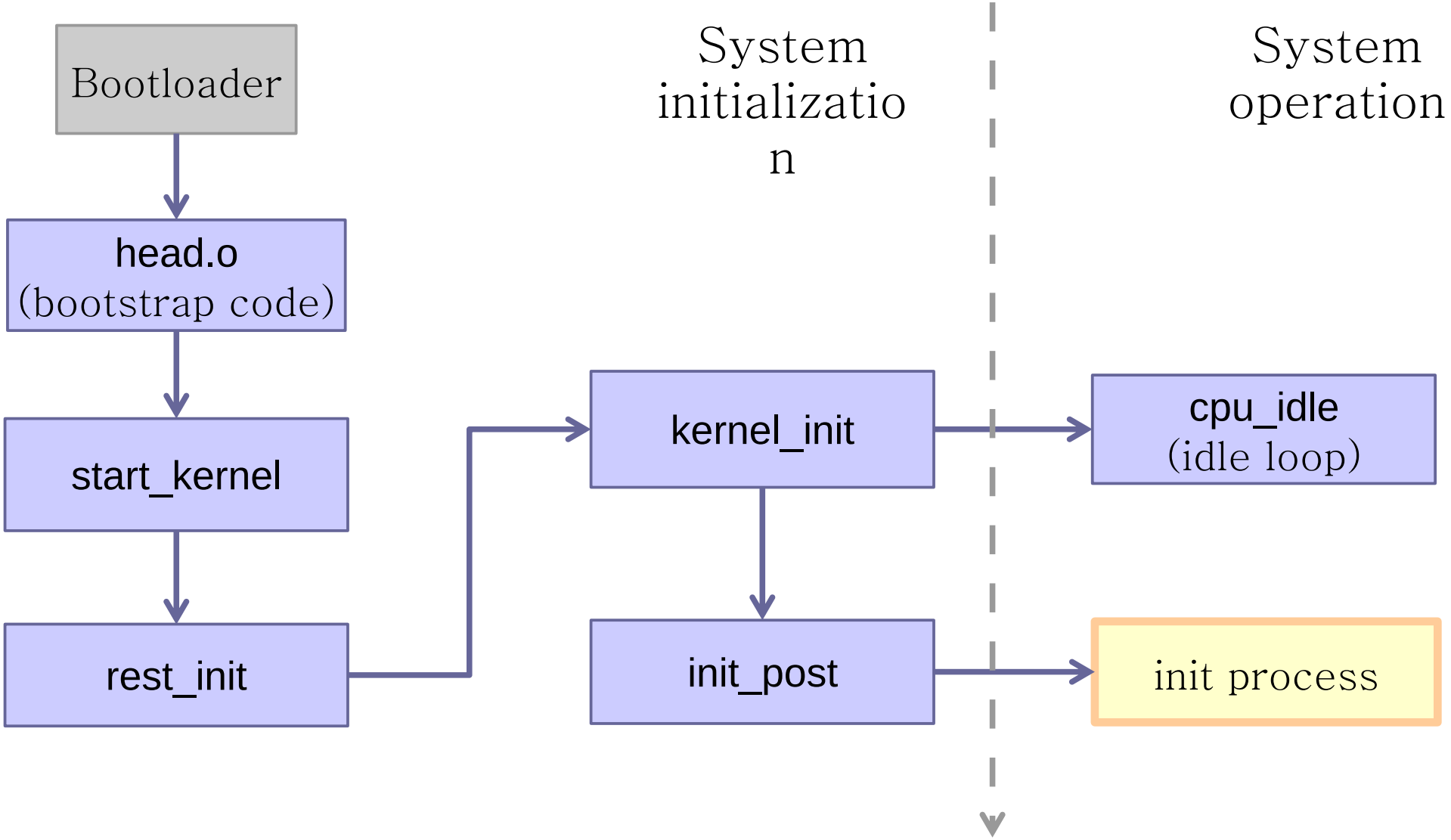
- First tries to open a console
- Then tries to run the init process, effectively turning the current kernel thread into the userspace init process.

init_post code

```
static int noinline init_post(void)
{
	free_initmem();
	unlock_kernel();
	mark_rodata_ro();
	system_state = SYSTEM_RUNNING;
	numa_default_policy();

	if (sys_open((const char __user *) "/dev/console", O_RDWR, 0) < 0)
		printk(KERN_WARNING "Warning: unable to open an initial console.\n");

	(void) sys_dup(0);
	(void) sys_dup(0);

	if (ramdisk_execute_command) {
		run_init_process(ramdisk_execute_command);
		printk(KERN_WARNING "Failed to execute %s\n",
				ramdisk_execute_command);
	}

	/*
	 * We try each of these until one succeeds.
	 *
	 * The Bourne shell can be used instead of init if we are
	 * trying to recover a really broken machine.
	 */
	if (execute_command) {
		run_init_process(execute_command);
		printk(KERN_WARNING "Failed to execute %s.  Attempting "
					"defaults...\n", execute_command);
	}
	run_init_process("/sbin/init");
	run_init_process("/etc/init");
	run_init_process("/bin/init");
	run_init_process("/bin/sh");

	panic("No init found.  Try passing init= option to kernel.");
}
```

Source:
init/main.c
in Linux 2.6.22

Kernel initialization graph



Kernel initialization - What to remember

- The bootloader executes bootstrap code.
- Bootstrap code initializes the processor and board, and uncompresses the kernel code to RAM, and calls the kernel's start_kernel function.
- Copies the command line from the bootloader.
- Identifies the processor and machine.
- Initializes the console.
- Initializes kernel services (memory allocation, scheduling, file cache...)
- Creates a new kernel thread (future init process) and continues in the idle loop.
- Initializes devices and execute initcalls.

Thank You

VEDA Solutions
Inspired by open Source