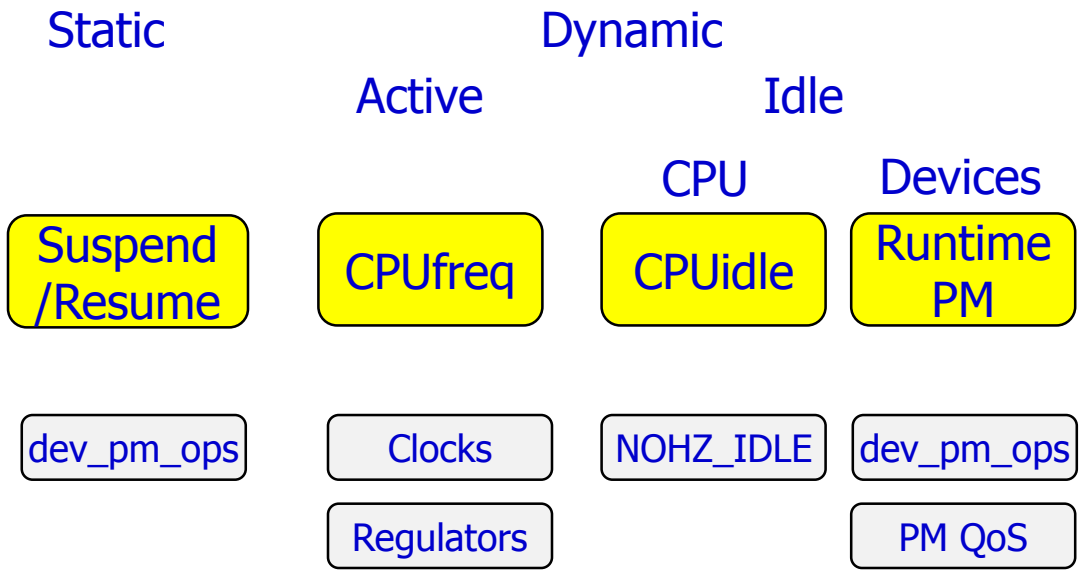
Rights to copy

Power Management

PM building blocks

- Several power management *building blocks*
  - Suspend and resume
  - CPUidle
  - Runtime power management
  - Frequency and voltage scaling
- Independent *building blocks* that can be improved gradually during development



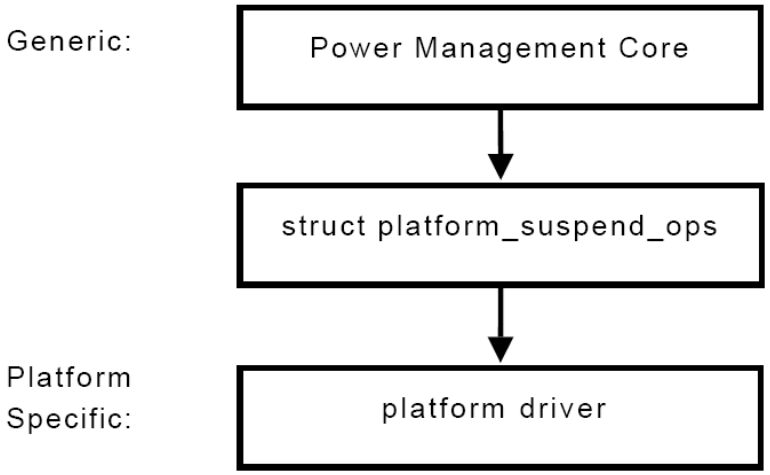

```
struct dev_pm_ops {
   int (*prepare)(struct device *dev);
   void (*complete)(struct device *dev);
   int (*suspend)(struct device *dev);
   int (*resume)(struct device *dev);
   int (*freeze)(struct device *dev);
   int (*thaw)(struct device *dev);
   int (*poweroff)(struct device *dev);
   int (*restore)(struct device *dev);
   int (*suspend_late)(struct device *dev);
   int (*resume_early)(struct device *dev);
   int (*freeze_late)(struct device *dev);
   int (*thaw_early)(struct device *dev);
   int (*poweroff_late)(struct device *dev);
   int (*restore_early)(struct device *dev);
   int (*suspend_noirq)(struct device *dev);
   int (*resume_noirq)(struct device *dev);
   int (*freeze_noirq)(struct device *dev);
   int (*thaw_noirq)(struct device *dev);
   int (*poweroff_noirq)(struct device *dev);
   int (*restore_noirq)(struct device *dev);
   int (*runtime_suspend)(struct device *dev);
   int (*runtime_resume)(struct device *dev);
   int (*runtime_idle)(struct device *dev);
};
```

Suspend-to-RAM

- kernel/power/suspend.c
- Freeze all tasks
- Suspend all devices
- (Usually) put the DRAM into a self-refresh mode
- Set the CPU into the deepest sleep state and wait for a wake-up event
- On wake-up, set the DRAM to normal refresh mode
- Resume all devices
- Thaw all tasks

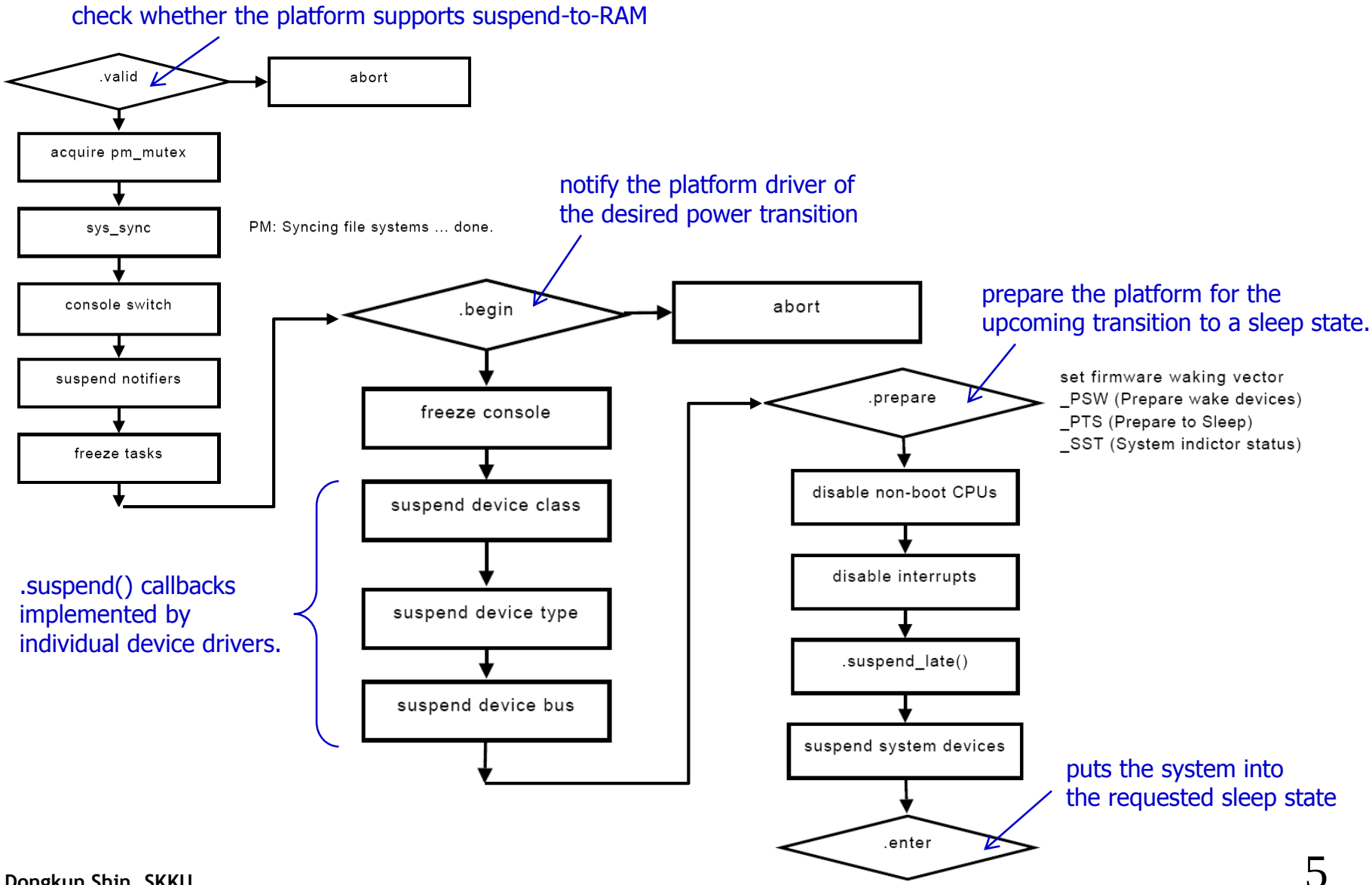
Suspend-to-RAM in Linux (OLS'08)
https://www.landley.net/kdocs/ols/2008/ols2008v1-pages-39-52.pdf

To suspend to RAM:
- echo mem > /sys/power/state

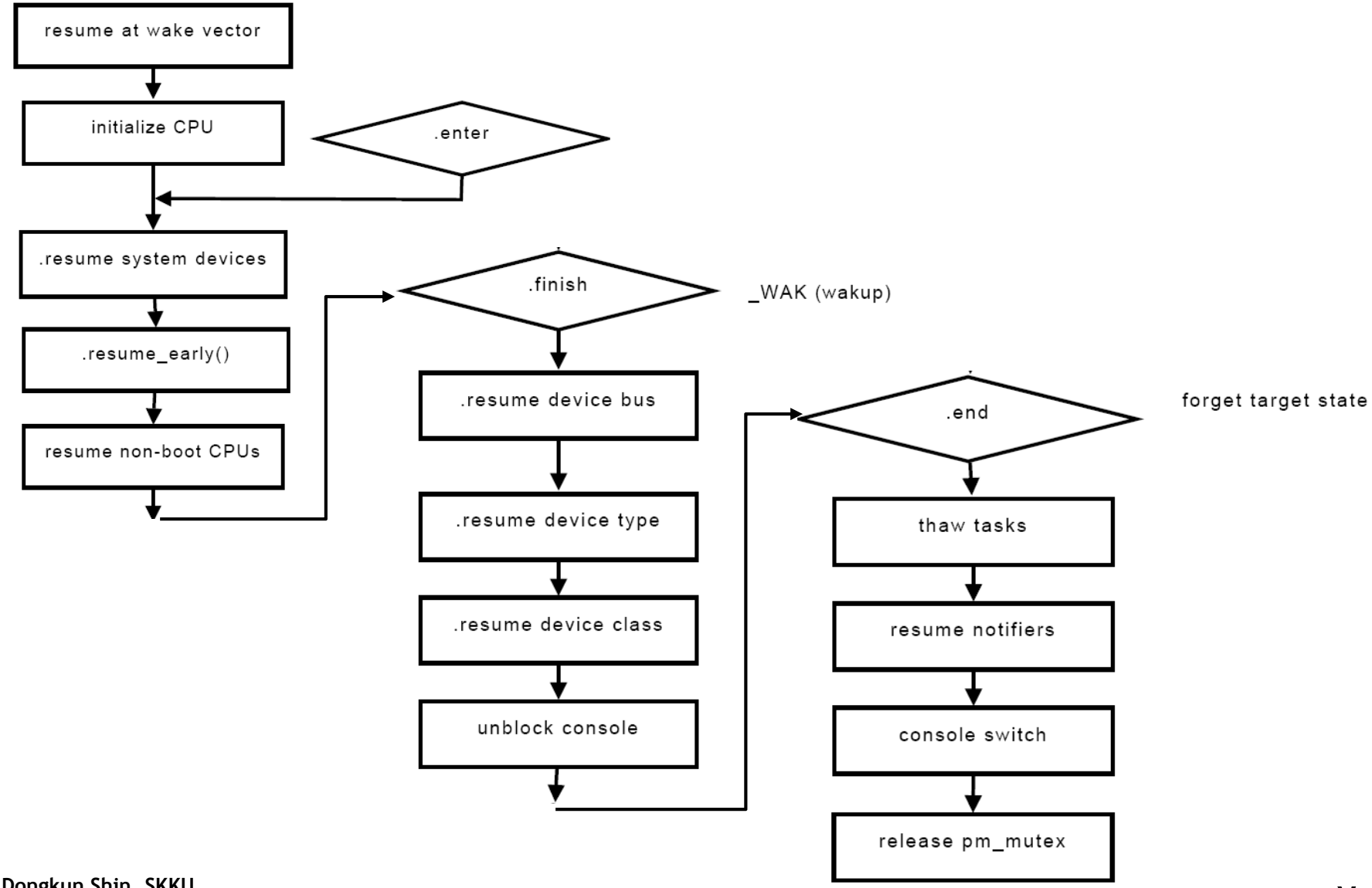



```
struct platform_suspend_ops {
  int (*valid)(suspend_state_t state);
  int (*begin)(suspend_state_t state);
  int (*prepare)(void);
  int (*enter)(suspend_state_t state);
  void (*finish)(void);
  void (*end)(void);
};
```

Suspend Sequence



Resume Sequence



Suspend and resume

- Infrastructure in the kernel to support suspend and resume
- Platform hooks
  - prepare(), enter(), finish(), valid() in struct platform_suspend_ops
  - Registered using the suspend_set_ops() function
  - See arch/arm/mach-at91/pm.c
- Device drivers
  - suspend() and resume() hooks in the *_driver structures (struct platform_driver, struct usb_driver, etc.)
  - See drivers/net/ethernet/cadence/macb_main.c

suspend_set_ops()

```
static void __init at91_pm_init(void (*pm_idle)(void))
{
        struct device_node *pmc_np;
        const struct of_device_id *of_id;
        const struct pmc_info *pmc;

        if (at91_cpuidle_device.dev.platform_data)
                platform_device_register(&at91_cpuidle_device);

        pmc_np = of_find_matching_node_and_match(NULL, atmel_pmc_ids, &of_id);
        soc_pm.data.pmc = of_iomap(pmc_np, 0);
        if (!soc_pm.data.pmc) {
                pr_err("AT91: PM not supported, PMC not found□n");
                return;
        }

        pmc = of_id->data;
        soc_pm.data.uhp_udp_mask = pmc->uhp_udp_mask;

        if (pm_idle)
                arm_pm_idle = pm_idle;

        at91_pm_sram_init();

        if (at91_suspend_sram_fn) {
                suspend_set_ops(&at91_pm_ops);
                pr_info("AT91: PM: standby: %s, suspend: %s□n",
                        pm_modes[soc_pm.data.standby_mode].pattern,
                        pm_modes[soc_pm.data.suspend_mode].pattern);
        } else {
                pr_info("AT91: PM not supported, due to no SRAM allocated□n");
        }
}
```

```
static const struct platform_suspend_ops at91_pm_ops = {
        .valid  = at91_pm_valid_state,
        .begin = at91_pm_begin,
        .enter = at91_pm_enter,
        .end    = at91_pm_end,
};
```

```
static int at91_pm_enter(suspend_state_t state)
{
        switch (state) {
        case PM_SUSPEND_MEM:
        case PM_SUSPEND_STANDBY:
                if (soc_pm.data.mode >= AT91_PM_ULP0 &&
                    !at91_pm_verify_clocks())
                        goto error;
                at91_pm_suspend(state);
                break;
        case PM_SUSPEND_ON:
                cpu_do_idle();
                break;
        default:
                pr_debug("AT91: PM - bogus suspend state
%d□n", state);
                goto error;
        }
}
```

arch/arm/mach-at91/pm.c

Triggering suspend

- struct suspend_ops functions are called by the enter_state() function.
- enter_state() also takes care of executing the suspend and resume functions for your devices.
- The execution of this function can be triggered from user space. To suspend to RAM:
  - echo mem > /sys/power/state
- Can also use the s2ram program from http://suspend.sourceforge.net/
- Read kernel/power/suspend.c

Triggering suspend kernel/power/suspend.c

```
/* Externally visible function for suspending the system.
@state: System sleep state to enter. */
int pm_suspend(suspend_state_t state)
{
        …
        error = enter_state(state);
        …
}
```

```
/*Do common work needed to enter system sleep state */
static int enter_state(suspend_state_t state)
{
        …
        error = suspend_prepare(state);
        …
        error = suspend_devices_and_enter(state);


 Finish:
        …
        suspend_finish();
        …
}
```

```
static void suspend_finish(void)
{
        suspend_thaw_processes();
        pm_notifier_call_chain(PM_POST_SUSPEND);
        pm_restore_console();
}
```

```
static int suspend_prepare(suspend_state_t state)
{
        …
        pm_prepare_console();

        error = __pm_notifier_call_chain(PM_SUSPEND_PREPARE, -1, &nr_calls);
        …
        error = suspend_freeze_processes();
        …

 Finish:
        __pm_notifier_call_chain(PM_POST_SUSPEND, nr_calls, NULL);
        pm_restore_console();
        return error;
}
```

```
int suspend_devices_and_enter(suspend_state_t state)
{
        …
        error = platform_suspend_begin(state);          call .begin

        suspend_console();
        error = dpm_suspend_start(PMSG_SUSPEND);

        do {
                error = suspend_enter(state, &wakeup);   call .prepare, .enter
        } while (!error && !wakeup && platform_suspend_again(state));

 Resume_devices:
        dpm_resume_end(PMSG_RESUME);
        resume_console();

 Close:
        platform_resume_end(state);
        pm_suspend_target_state = PM_SUSPEND_ON;
        return error;
 …
}
```

Saving power in the idle loop

- The idle loop is what you run when there's nothing left to run in the system.
- Implemented in all architectures in arch/<arch>/kernel/process.c
- Example to read: look for cpu_idle in arch/arm/kernel/process.c
- Each ARM cpu defines its own arch_idle function.
- The CPU can run power saving HLT instructions, enter NAP mode, and even disable the timers (tickless systems).

Managing idle

- Adding support for multiple idle levels
- Modern CPUs have several sleep states offering different power savings with associated wake up latencies
- The *dynamic tick* feature allows to remove the periodic tick to save power, and to know when the next event is scheduled, for smarter sleeps.
- CPUidle infrastructure to change sleep states
  - Platform-specific driver defining sleep states and transition operations
  - Platform-independent governors (ladder and menu)
  - Available for x86/ACPI, not supported yet by all ARM cpus.
    - look for cpuidle* files under arch/arm/
- See Documentation/cpuidle/ in kernel sources.

CPUIdle

- Put idle CPUs into low-power states (no code execution)
  - CPU scheduler knows when a CPU is idle.
  - Maximum acceptable latency from PM QoS.
    - PM QoS: a kernel infrastructure to facilitate the communication of latency and throughput needs among devices, system, and users.
  - CPU low-power states (C-states) characteristics are known.
  - Governors decide what state to go for.
  - CPU can run power saving HLT instructions, enter NAP mode, and even disable the timers (tickless systems).
    - HLT stops the CPU until an interrupt occurs
- Deeper sleep modes
  - stopping the clock to some parts of the core
  - powering down parts of the core, losing some state

	C1	C2	C3
Exit latency (μs)	**1**	**1**	**57**
Power consumption (mW)	**1000**	**500**	**100**

CPUIdle

- /sys/devices/system/cpu/cpu0/cpuidle
  - * desc : Small description about the idle state (string)
  - * disable : Option to disable this idle state (bool)
  - * latency : Latency to exit out of this idle state (in microseconds)
  - * name : Name of the idle state (string)
  - * power : Power consumed while in this idle state (in milliwatts)
  - * time : Total time spent in this idle state (in microseconds)
  - * usage : Number of times this state was entered (count)

```
# ls -lR /sys/devices/system/cpu/cpu0/cpuidle/
/sys/devices/system/cpu/cpu0/cpuidle/:
total 0
drwxr-xr-x 2 root root 0 Feb  8 10:42 state0
drwxr-xr-x 2 root root 0 Feb  8 10:42 state1
drwxr-xr-x 2 root root 0 Feb  8 10:42 state2
drwxr-xr-x 2 root root 0 Feb  8 10:42 state3

/sys/devices/system/cpu/cpu0/cpuidle/state0:
total 0
-r--r--r-- 1 root root 4096 Feb  8 10:42 desc
-rw-r--r-- 1 root root 4096 Feb  8 10:42 disable
-r--r--r-- 1 root root 4096 Feb  8 10:42 latency
-r--r--r-- 1 root root 4096 Feb  8 10:42 name
-r--r--r-- 1 root root 4096 Feb  8 10:42 power
-r--r--r-- 1 root root 4096 Feb  8 10:42 time
-r--r--r-- 1 root root 4096 Feb  8 10:42 usage
```

CPUIdle states

Documentation/devicetree/bindings/arm/idle-states.txt

State	Power (mW)	Latency (uS)
C0	-1	0
C1	1000	1
C2	500	1
C3	100	57

Intel CPU ACPI states

Mode	Core	Memory	Power	Characteristics	Exit to Run Mode
Run	Clocks On Power On	Clocks On Power On	Application Dependent	IEM Performance scaling	
Standby	Clocks Off Power On	Clocks Off Power On	Static Leakage	All state preserved. Enter by WFI instruction	Interrupt, Debug Request (<1us)
Dormant	Clocks Off Power Off	Clocks Off Power On	Static Lkg of Memory Arrays	Cache & TCM Contents Preserved	"Soft" Reset (100ms)
Shutdown	Clocks Off Power Off	Clocks Off Power Off	Zero	CPU & Cache State Lost	Reset (OS boot >500ms)

ARM 11 power modes Mode

CPUIdle - Dynamic tick

- **Historically Linux has a regular timer tick**
  - HZ from 100 to 1000
- **Can create unnecessary wakeups**
- **Dynamic tick mode means timer interrupts occur only when needed**
  - Since 2.6.21, the dynamic tick feature allows to remove the periodic tick to save power, and to know when the next event is scheduled, for smarter sleeps.
  - CONFIG_NO_HZ

- **PowerTOP (**https://01.org/powertop/)
  - parses the /proc/timer_stats output and gives a picture of what is causing wakeups (who is preventing sleep)
  - Now available on ARM cpus implementing CPUidle
  - Also gives you useful hints for reducing power.

I/O Runtime Power Management

- allows I/O devices to be put into energy-saving (low power) states at run time (or auto-suspended) after a specified period of inactivity and woken up in response to a hardware-generated wake-up event or a driver's request.
- per-device suspend/resume
  - Single device at a time
  - Controllable by driver
- devices are independent
  - one device cannot prevent other devices from PM
- New hooks must be added to the drivers:

```
struct dev_pm_ops {
    ...
    int (*runtime_suspend)(struct device *dev);
    int (*runtime_resume)(struct device *dev);
    int (*runtime_idle)(struct device *dev);
};
```

- API and details on Documentation/power/runtime_pm.txt

PM Quality Of Service interface

- Kernel and user mode interface for registering performance expectations by drivers, subsystems and user space applications.
- Two different PM QoS frameworks are available:
  - 1. PM QoS classes for cpu_dma_latency, network_latency, network_throughput, memory_bandwidth.
    - void pm_qos_add_request(handle, param_class, target_value)
    - void pm_qos_update_request(handle, new_target_value)
    - void pm_qos_remove_request(handle)
    - int pm_qos_request(param_class)
    - int pm_qos_request_active(handle)
    - int pm_qos_add_notifier(param_class, notifier)
    - int pm_qos_remove_notifier(int param_class, notifier)
    - From user mode: /dev/[cpu_dma_latency, network_latency, network_throughput]
  - 2. per-device PM QoS framework provides the API to manage the per-device latency constraints and PM QoS flags.
    - int dev_pm_qos_add_request(device, handle, type, value)
    - int dev_pm_qos_update_request(handle, new_value)
    - int dev_pm_qos_remove_request(handle)
    - s32 dev_pm_qos_read_value(device)
    - …
- According to these requirements, PM QoS allows kernel drivers to adjust their power management

  int max_latency = pm_qos_request (PM_QOS_CPU_DMA_LATENCY);
- See Documentation/power/pm_qos_interface.txt

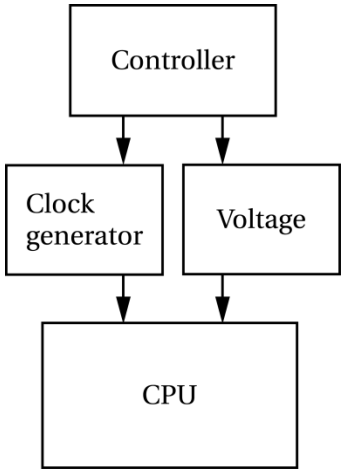
Dynamic Voltage & Frequency Scaling (DVFS)

- **Power scales with V^2 while performance scales roughly as V.**
- **Reduce operating voltage, add parallel operating units to make up for lower clock speed ➔ Multicore**

$$P \cong \alpha \cdot C_L \cdot f_{CLK} \cdot V_{DD}^2$$

$$f_{CLK} = k \frac{V_{DD}}{(V_{DD} - V_{TH})}$$

- **DVFS**
  - Scale both voltage and clock frequency.
  - Can use control algorithms to match performance to application, reduce power.
  - The Key issue is how to monitor the current workload and how to predict the future workload!



Frequency and voltage scaling (1)

- Frequency and voltage scaling possible through the cpufreq kernel infrastructure.
- Generic infrastructure: drivers/cpufreq/cpufreq.c and include/linux/cpufreq.h
- Generic governors, responsible for deciding frequency and voltage transitions
  - powersave - always select the lowest frequency
  - performance - always select the highest frequency
  - ondemand - change frequency based on utilization: if the CPU is idle < 20% of the time, set the frequency to the maximum; if idle >=30%, drop the frequency down in 5% decrements
  - conservative - as "ondemand", but switches to higher frequencies in 5% steps rather than going immediately to the maximum
  - userspace - frequency is set by a userspace application
- This infrastructure can be controlled from /sys/devices/system/cpu/cpu<n>/cpufreq/

Frequency and voltage scaling (2)

- CPU drivers in drivers/cpufreq/
  - Example: drivers/cpufreq/omap-cpufreq.c
- Must implement the operations of the cpufreq_driver structure and register them using cpufreq_register_driver()
  - init() for initialization
  - exit() for cleanup
  - verify() to verify the user-chosen policy
  - setpolicy() or target() to actually perform the frequency change
- See Documentation/cpu-freq/ for useful explanations

Frequency and voltage scaling (3)

```
static struct cpufreq_driver omap_driver = {
        .flags          = CPUFREQ_STICKY | CPUFREQ_NEED_INITIAL_FREQ_CHECK,
        .verify         = cpufreq_generic_frequency_table_verify,
        .target_index       = omap_target,
        .get            = cpufreq_generic_get,
        .init           = omap_cpu_init,
        .exit           = omap_cpu_exit,
        .name           = "omap",
        .attr           = cpufreq_generic_attr,
};

static int omap_cpufreq_probe(struct platform_device *pdev)
{
        mpu_dev = get_cpu_device(0);
        if (!mpu_dev) {
                pr_warn("%s: unable to get the MPU device□n", __func__);
                return -EINVAL;
        }

        mpu_reg = regulator_get(mpu_dev, "vcc");
        if (IS_ERR(mpu_reg)) {
                pr_warn("%s: unable to get MPU regulator□n", __func__);
                mpu_reg = NULL;
        } else {
                if (regulator_get_voltage(mpu_reg) < 0) {
                        pr_warn("%s: physical regulator not present for MPU□n",
                                __func__);
                        regulator_put(mpu_reg);
                        mpu_reg = NULL;
                }
        }

        return cpufreq_register_driver(&omap_driver);
}
```

drivers/cpufreq/omap-cpufreq.c

CPUFreq

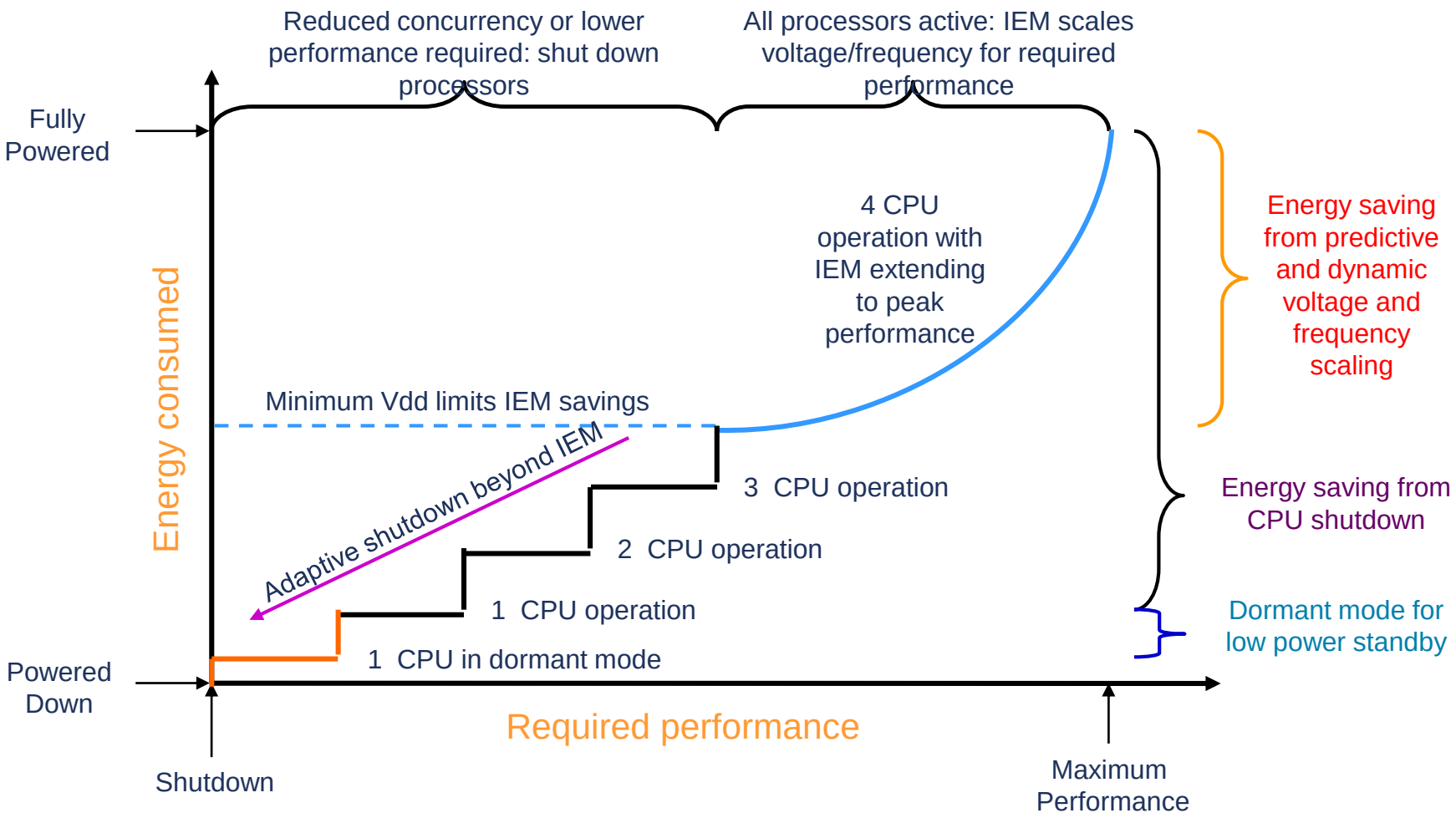
- can be controlled from **/sys/devices/system/cpu/cpu0/cpufreq/**
  - scaling_max_freq, scaling_min_freq, and scaling_available_frequencies
  - scaling_available_governors which lists the names of the built-in governors
  - scaling_governor which tells you the current governor and allows you to set a new one
  - scaling_setspeed allows you to set the speed if the governor is set to "userspace"

PowerTop

```
File  Edit  View  Terminal  Help
     PowerTOP version 1.11      (C) 2007 Intel Corporation

Cn                Avg residency       P-states (frequencies)
C0 (cpu running)        ( 2.8%)         1.60 Ghz     3.2%
polling           0.0ms ( 0.0%)         1333 Mhz     0.0%
C1 mwait          0.2ms ( 0.0%)         1067 Mhz     0.0%
C2 mwait          0.5ms ( 0.0%)          800 Mhz    96.8%
C4 mwait          0.2ms ( 0.0%)
C6 mwait          9.1ms (97.2%)
Wakeups-from-idle per second : 107.8    interval: 15.0s
no ACPI power usage estimate available

Top causes for wakeups:
  41.1% ( 62.1)       <interrupt> : eth0, psb@pci:0000:00:02.0
  21.4% ( 32.4)       <interrupt> : ehci_hcd:usb1, uhci_hcd:usb2, ra0
   8.9% ( 13.4)   USB device  2-2 : Microsoft Wireless Optical Mouse® 1.00 (Micr
   8.6% ( 12.9)     <kernel core> : hrtimer_start (tick_sched_timer)
   6.6% ( 10.0)     <kernel core> : add_timer (rtmp_timer_MlmePeriodicExec)
   5.4% (  8.1)     <kernel core> : usb_hcd_poll_rh_status (rh_timer_func)
   3.8% (  5.8)     <kernel core> : hrtimer_start_range_ns (tick_sched_timer)
   1.2% (  1.8)    gnome-terminal : hrtimer_start_range_ns (hrtimer_wakeup)
   0.6% (  0.9)              Xorg : psb_xhw_ioctl (process_timeout)
   0.4% (  0.6)              Xorg : queue_delayed_work (delayed_work_timer_fn)
   0.2% (  0.3)     <kernel core> : add_timer (rtmp_timer_AsicRxAntEvalTimeout)
   0.2% (  0.3)          nautilus : hrtimer_start_range_ns (hrtimer_wakeup)
   0.2% (  0.3)   update-notifier : hrtimer_start_range_ns (hrtimer_wakeup)
   0.2% (  0.3)   gnome-screensav : hrtimer_start_range_ns (hrtimer_wakeup)
   0.2% (  0.3)       gnome-panel : hrtimer_start_range_ns (hrtimer_wakeup)
   0.1% (  0.2)       <interrupt> : pata_sch
   0.1% (  0.2)   gnome-settings- : hrtimer_start_range_ns (hrtimer_wakeup)
   0.1% (  0.1)              Xorg : hrtimer_start (it_real_fn)
   0.1% (  0.1)         ssh-agent : hrtimer_start_range_ns (hrtimer_wakeup)
   0.1% (  0.1)   gnome-power-man : hrtimer_start_range_ns (hrtimer_wakeup)

Suggestion: increase the VM dirty writeback time from 5.00 to 15 seconds with:
  echo 1500 > /proc/sys/vm/dirty_writeback_centisecs
This wakes the disk up less frequently for background VM activity
 Q - Quit  R - Refresh  W - Increase Writeback time
```

MPCore extends beyond simply DVFS

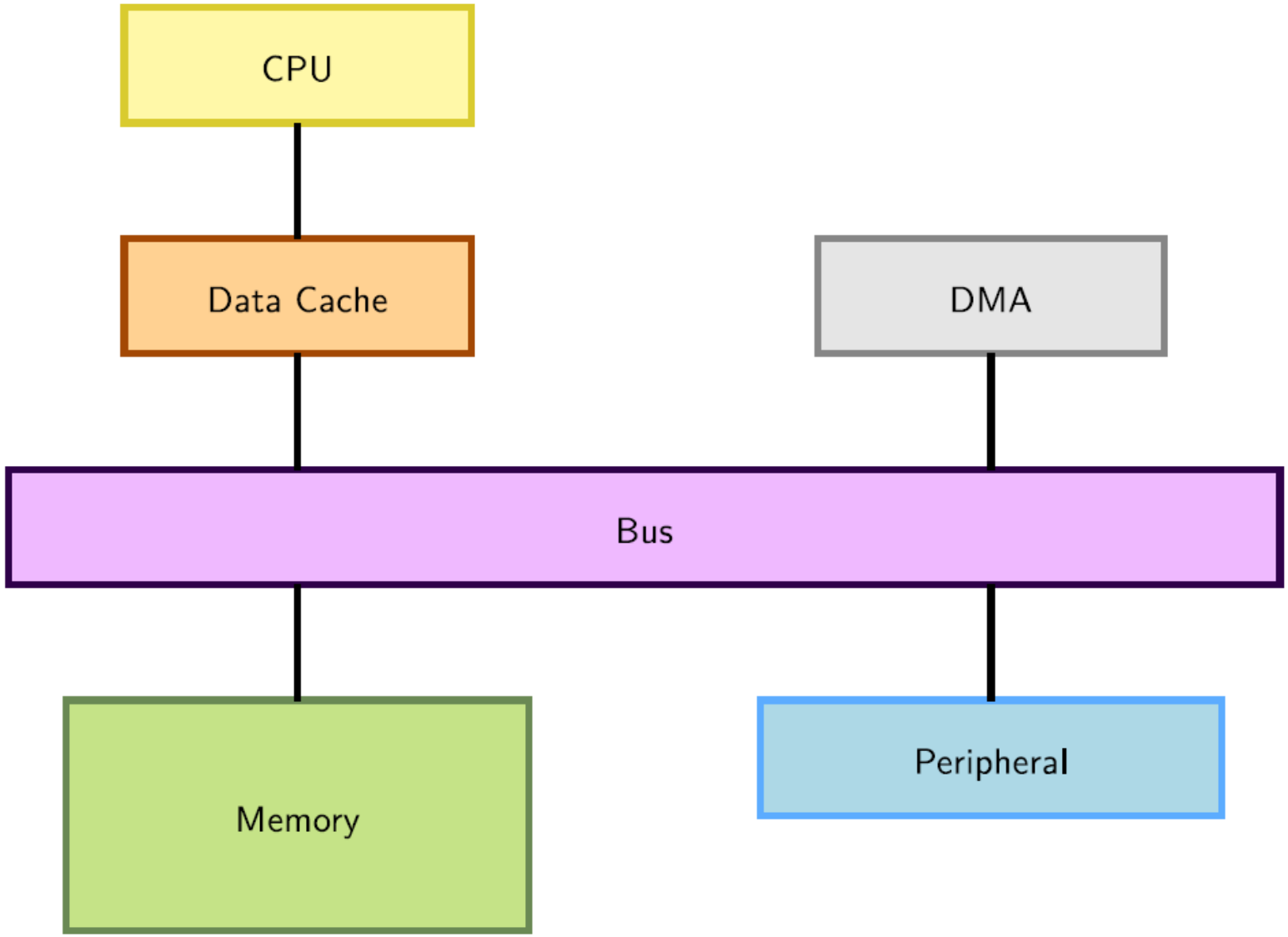
MPCore extends control over power usage by providing
both voltage and frequency scaling and turning off unused processors



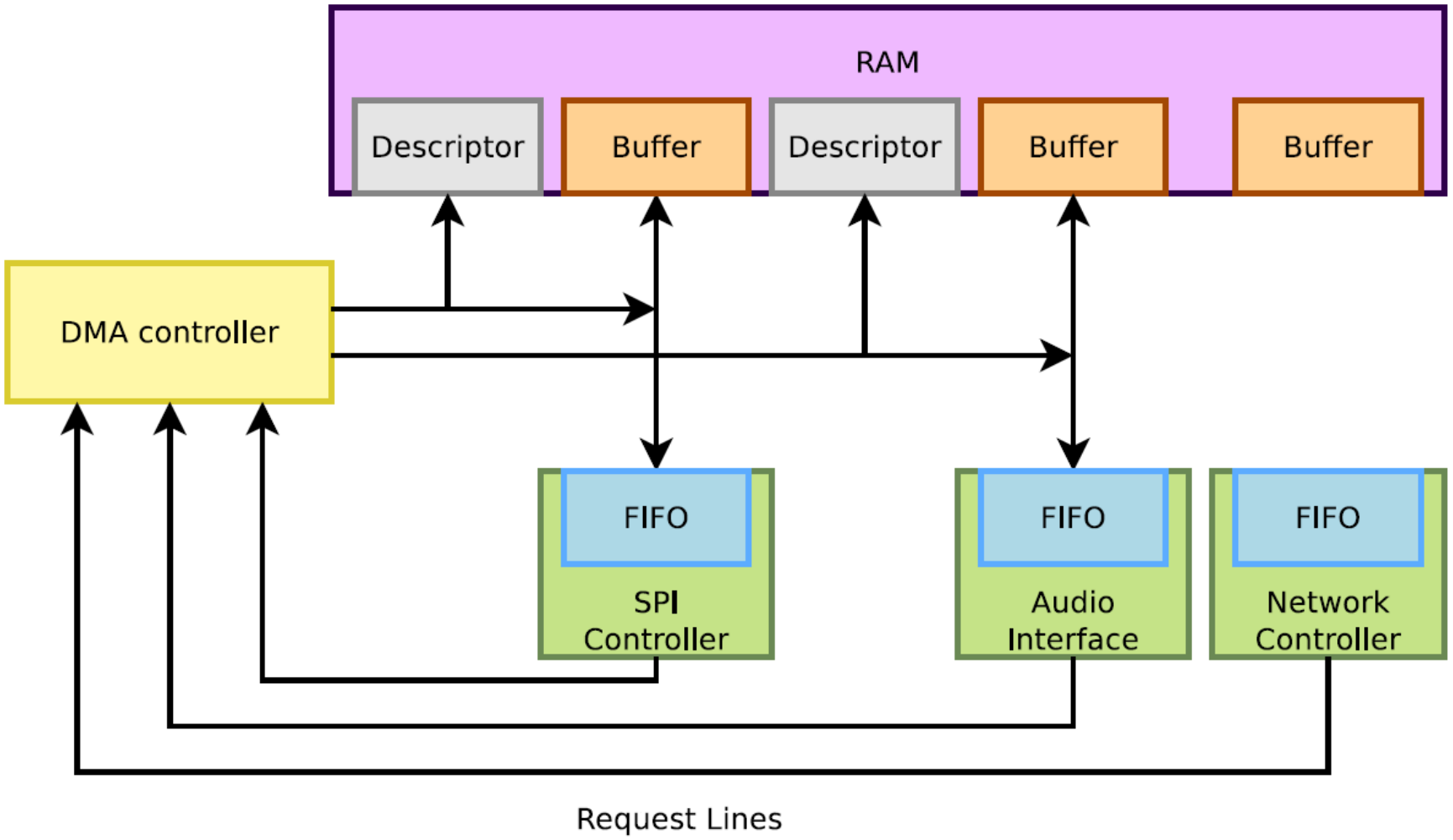
EAS (Energy-Aware Scheduler)

- https://elinux.org/images/6/69/Eas-unbiased1.pdf
- https://events.static.linuxfound.org/sites/events/files/slides/klf15_bpark.pdf

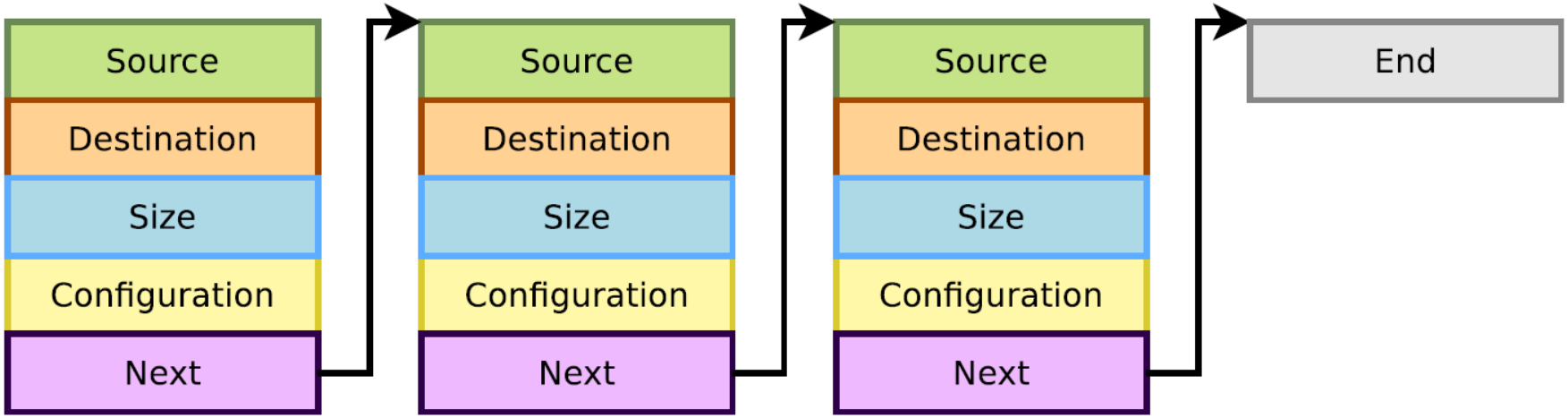
DMA integration



DMA controllers



DMA descriptors



Constraints with a DMA

- A DMA deals with physical addresses, so:
  - Programming a DMA requires retrieving a physical address at some point (virtual addresses are usually used)
  - The memory accessed by the DMA shall be physically contiguous
- The CPU can access memory through a data cache
  - Using the cache can be more efficient (faster accesses to the cache than the bus)
  - But the DMA does not access to the CPU cache, so one need to take care of cache coherency (cache content vs memory content)
  - Either flush or invalidate the cache lines corresponding to the buffer accessed by DMA and processor at strategic times

DMA memory constraints

- Need to use contiguous memory in physical space.
- Can use any memory allocated by kmalloc() (up to 128 KB) or __get_free_pages() (up to 8MB).
- Can use block I/O and networking buffers, designed to support DMA.
- Can not use vmalloc() memory (would have to setup DMA on each individual physical page).

Memory synchronization issues

- Memory caching could interfere with DMA
- Before DMA to device
  - Need to make sure that all writes to DMA buffer are committed.
- After DMA from device
  - Before drivers read from DMA buffer, need to make sure that memory caches are flushed.
- Bidirectional DMA
  - Need to flush caches before and after the DMA transfer.

Linux DMA API

- The kernel DMA utilities can take care of:
  - Either allocating a buffer in a cache coherent area,
  - Or making sure caches are flushed when required,
  - Managing the DMA mappings and IOMMU (if any).
  - See Documentation/DMA-API.txt for details about the Linux DMA generic API.
  - Most subsystems (such as PCI or USB) supply their own DMA API, derived from the generic one. May be sufficient for most needs.

Coherent or streaming DMA mappings

- Coherent mappings
  - The kernel allocates a suitable buffer and sets the mapping for the driver.
  - Can simultaneously be accessed by the CPU and device.
  - So, has to be in a cache coherent memory area.
  - Usually allocated for the whole time the module is loaded.
  - Can be expensive to setup and use on some platforms.
- Streaming mappings
  - The kernel just sets the mapping for a buffer provided by the driver.
  - Use a buffer already allocated by the driver.
  - Mapping set up for each transfer. Keeps DMA registers free on the hardware.
  - The recommended solution.

Allocating coherent mappings

The kernel takes care of both buffer allocation and mapping

```
#include <asm/dma-mapping.h>

void *                          /* Output: buffer address */
    dma_alloc_coherent(
          struct device *dev, /* device structure */
          size_t size,          /* Needed buffer size in bytes */
          dma_addr_t *handle, /* Output: DMA bus address */
          gfp_t gfp             /* Standard GFP flags */
);

void dma_free_coherent(struct device *dev,
    size_t size, void *cpu_addr, dma_addr_t handle);
```

Setting up streaming mappings

Works on buffers already allocated by the driver

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

dma_addr_t dma_map_single(
        struct device *,          /* device structure */
        void *,                   /* input: buffer to use */
        size_t,                   /* buffer size */
        enum dma_data_direction /* Either DMA_BIDIRECTIONAL,
                                 * DMA_TO_DEVICE or
                                 * DMA_FROM_DEVICE */
);

void dma_unmap_single(struct device *dev, dma_addr_t handdle,
    size_t size, enum dma_data_direction dir);
```

DMA streaming mapping notes

- When the mapping is active: only the device should access the buffer (potential cache issues otherwise).
- The CPU can access the buffer only after unmapping!
- Another reason: if required, this API can create an intermediate bounce buffer (used if the given buffer is not usable for DMA).
- The Linux API also supports scatter / gather DMA streaming mappings.

Starting DMA transfers

- If the device you're writing a driver for is doing peripheral DMA, no external API is involved.
- If it relies on an external DMA controller, you'll need to
  - Ask the hardware to use DMA, so that it will drive its request line
  - Use Linux DMAEngine framework, especially its slave API

DMAEngine slave API

- In order to start a DMA transfer, you need to call the following functions from your driver
  1. Request a channel for exclusive use with dma_request_channel(), or one of its variants
  2. Configure it for our use case, by filling a struct dma_slave_config structure with various parameters (source and destination adresses, accesses width, etc.) and passing it as an argument to dmaengine_slave_config()
  3. Start a new transaction with dmaengine_prep_slave_single() or dmaengine_prep_slave_sg()
  4. Put the transaction in the driver pending queue using dmaengine_submit()
  5. And finally ask the driver to process all pending transactions using dma_async_issue_pending()
- Of course, all this needs to be done in addition to the DMA mapping seen previously