Combining Runtime PM and suspend/resume

Kevin Hilman
khilman@linaro.org

For an Introduction to runtime PM
http://people.linaro.org/~khilman/runtime_PM.html

What ? !!!

Now I have to add Runtime PM to my driver?

I thought suspend/resume was enough !!


System PM
Runtime PM
Today: Separate worlds

Tomorrow

Happy
together

Runtime PM

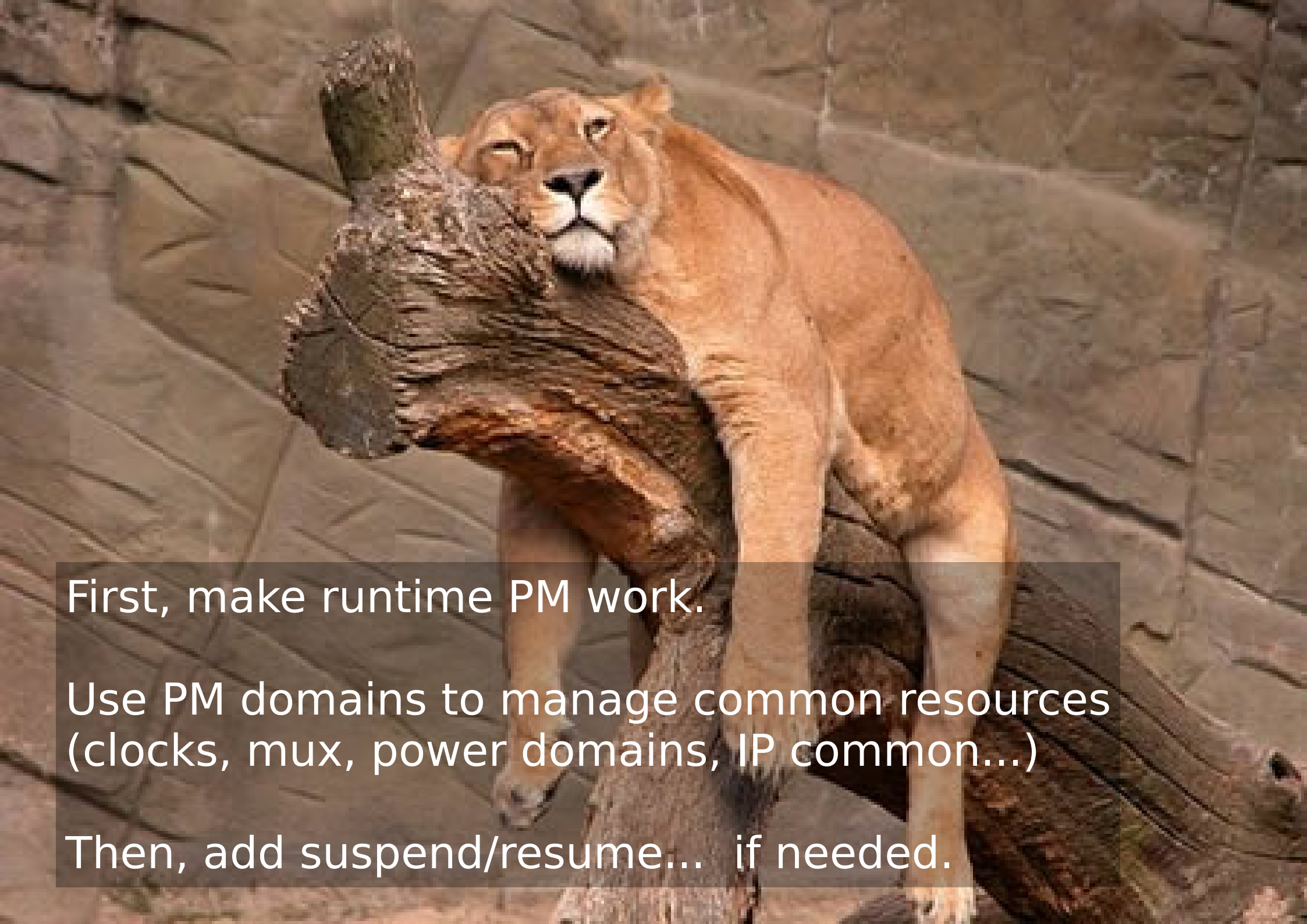
- suspend on inactivity
- resume when needed

System PM

- suspend when requested
- resume with system

Hey! what if I could implement runtime PM, and get suspend for free?

So... how to combine them?



First, make runtime PM work.

Use PM domains to manage common resources
(clocks, mux, power domains, IP common...)

Then, add suspend/resume... if needed.

Suspend triggers runtime PM

- Force inactivity in suspend
- device is now idle/inactive
- runtime PM takes over

Don't implement suspend

Necessary if device is used by other devices during suspend

Example: OMAP I2C driver

Needed during suspend by other drivers
· I2C-connected RTC
· regulators on I2C-connected PMIC

Solution:
· runtime PM only (no suspend/resume hooks)
· full (re)init per xfer; stateless

Quiz 1: What if I2C driver implemented suspend and was suspended before RTC?
Quiz 2: Who decides suspend ordering?

Great, so...

runtime PM, plus a little "enforcement" in `->suspend()` gets me both… right?

Yes...

Well...

Almost...

Except... there are several
ways for devices to be kept awake.

Tricky: async callbacks

PM workqueue frozen during suspend

- Asynchronous callbacks happen via PM core workqueue (`pm_wq`)
- workqueue frozen during suspend
- Result: no async callbacks

Solution: use `_sync()` methods

Tricky: Autosuspend timeouts

Driver does some work during `->suspend()`

- `_get_sync()`
- `_put_sync_autosuspend()`
- autosuspend callbacks are asynchronous
- Result: device left runtime enabled

Solution: PM domain magic

Tricky: PM core "protections"

PM core: `device_prepare()`
· increments usecount, decrements in `_complete()`
· effectively disables runtime suspend (but not resume)

Example: device runtime suspended: usecount = 0
· PM core prepare: `_get_noresume()`: usecount = 1
· driver `->suspend()`: `_get_sync()`: usecount = 2
· driver uses HW
· driver `->suspend()`: `_put_sync()`: usecount = 1

Result: stuck with usecount > 0, runtime enabled

Solution: PM domain magic

Tricky: userspace disable

runtime PM can be disabled from userspace

```
# echo on > /sys/devices/.../power/control
```

Oops, drivers using only runtime PM will
be left runtime enabled during suspend.

Solution: PM domain magic


PM domain magic

Use "late" callbacks

Ensure all devices are suspended

```
struct dev_pm_ops {
    [...]
    int (*suspend_late)(struct device *dev);
    int (*resume_early)(struct device *dev);
};
struct dev_pm_domain {
    struct dev_pm_ops    ops;
};
```

The magic trick

Runtime suspend "enforced" for stuck devices

```
static int my_domain_suspend_late(struct device *dev)>
{
        if (!pm_runtime_status_suspended(dev)) {
                if (pm_generic_runtime_suspend(dev) == 0) {
                        /* platform-specific PM */
                }
        }
        return 0;
}
```

Summary

- Runtime PM first
- use PM domains
- Keep drivers generic

What next?

- better understand PM core protections in `device_prepare()`
- Help PM domains proliferate
- What else?

¿ Questions ?

結束