Remote proc resource manager overview

Arnaud Pouliquen/Loic Pallardy

life.augmented

Proposed Terminology

To understand Resource Management mechanism, we need to be aligned on terminology. Here is a proposal:

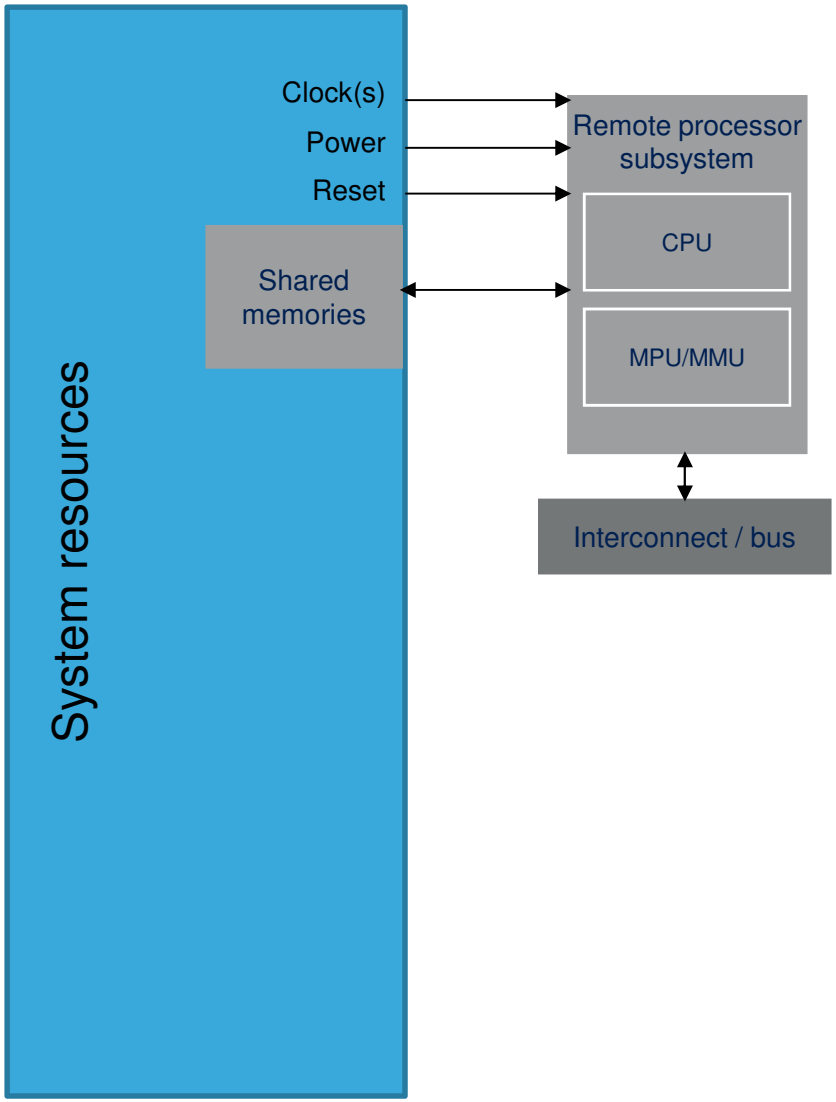
- **Peripheral resource**: A peripheral which can be assigned and controlled by a core without conflict with other cores:

  ⇒Peripheral can be isolated for a core (Hw semaphore, isolation, software resource manager…)

- **System resource**: central SoC resource required to operate the remote processor subsystem or a peripheral, shared by all cores and controlled by the master.
  - Resources which are commons : gpios, regulators, clocks, resets…
  - Resources which share common registers banks (platform dependent)

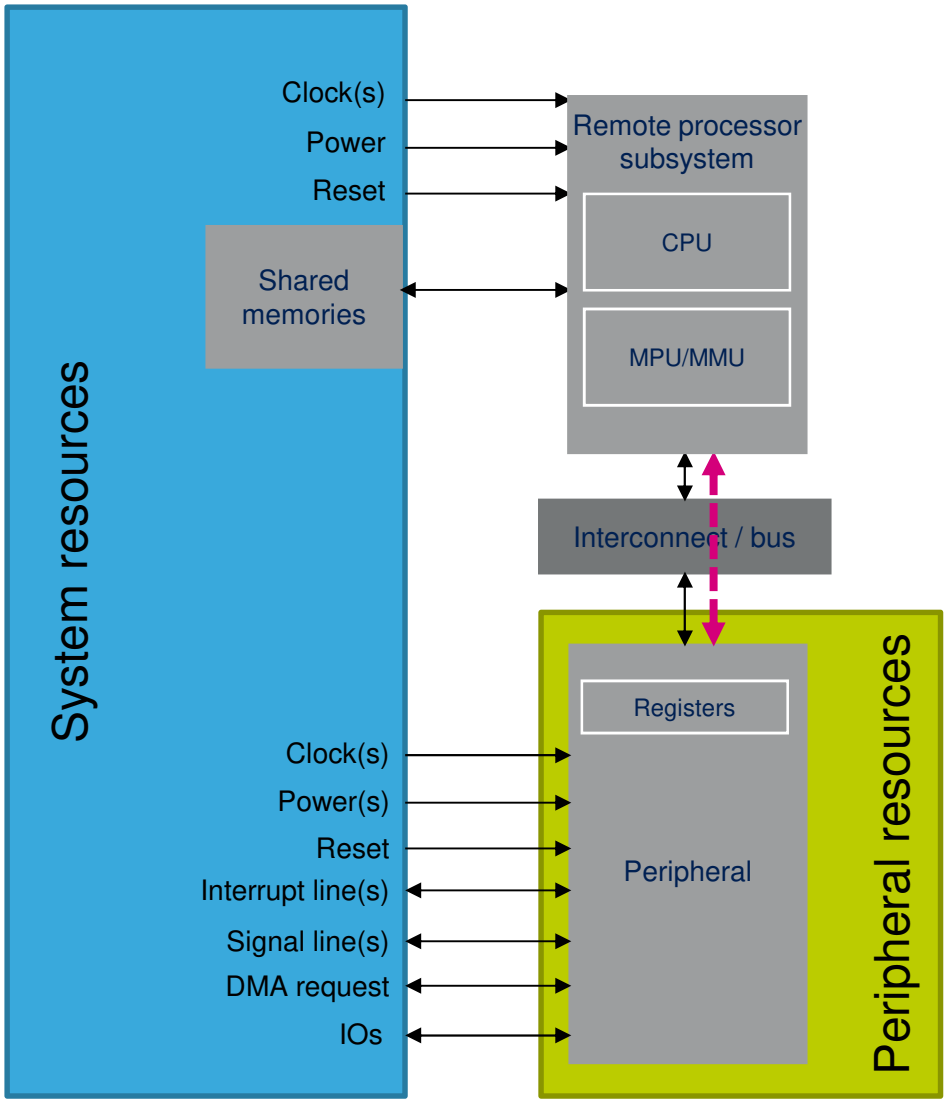
System resource for coprocessor management

- Remote proc platform driver is in charge of the system resources needed to operate the remote processor subsystem.
  - Clocks
  - Power
  - Reset
  - Memories access
- Rely on Linux frameworks that manages the system resources for Linux core.



System resource for Peripheral resources

- **System resources** also used to operate peripheral.
  - ⇒ If not configured, peripheral is not functional.
- **Peripheral resource** can be assigned:
  - To the master Linux core.
  - To the remote core.



Requirements Sum-up

- A **Peripheral resource** can be assigned to a master or slave core on remote processor firmware start.

  ⇒Static assignment.

- A **Peripheral resource** can be assigned or reassigned during remote processor runtime

  ⇒Dynamic assignment based on RPMsg.

- **System resources** must be handled by Master core as common for all cores.

  ⇒To manage concurrent access and global configuration (for instance clock tree)

- **System resources** associated to a peripheral can be updated (for instance clock rate update).

- Power management strategy can be implemented.

Why static assignment

⇒Solution proposed in current version of RPROC SRM.

- **Pro.**
  - Needed for subsystem without IPC (no shared memory, or remote processor with limited memory).
  - No Latency constraints induced by IPC messaging.
  - Remote processor has not to be aware how to configure the system resource (as a Linux driver).
  - Stop, crash and suspend management is simplified.

- **Cons.**
  - No check of the availability.
  - No reconfiguration possible during runtime.

⇒TI solution based on RPMsg is complementary and address these cons points. ST Plan it to implement services on top of rproc_srm.

ST solution: DT overview

- **One SRM core node** similar to a “device bus” for a remote processor:
  - List peripheral resources associated.
  - To be extended to add RPMsg channel for dynamic configuration.
- **One or several SRM devices** that represent(s) peripheral resource assigned to the remote processor.
  - Generic platform devices for basic system resources.
  - Specific platform devices for SoC specificities.
  - The peripheral is identified by the node name and/or physical address.
- Core assignment switching can be down by Bind/unbind (or overlay?).

```
soc {
        i2C1: i2c@F0010000 {
                compatible = "st,i2c";
                clocks = <&rcc_clk I2C1_K>;
                pinctrl-0 = <i2c1_pins_a>;
                status = “disabled";
        }
        slave_proc0@30000000 {
                compatible = “st, slave_rproc”;
                reg = <0x30000000 0x10000>,
                resets = <&rcc_rst>;
                reset-names = "slave_core0_rst";
                clocks = <&rcc_clk RPROC_K>;
                clock-names = “slave_core0_clk";
                system_resources {
                        compatible = "rproc-srm-core";
                        status = “okay";

                        I2C1: i2c@F0010000 {
                                compatible = "rproc-srm-dev";
                                clocks = <&rcc_clk I2C1_K>;
                                pinctrl-0 = <i2c1_pins_a>;

                                status = “okay";
                        };
                };
        };
};
```

life.augmented

Alternative 1: define all system resources in remoteproc node

- **Pro**
  - Simple to implement.
- **Cons**
  - No link between the system resources and a peripheral to facilitate reconfiguration.
    - Peripheral get /release.
    - Peripheral suspend/resume.

  ⇒Remote processor must know the system resources.
  - How to handle specific platform system resources?

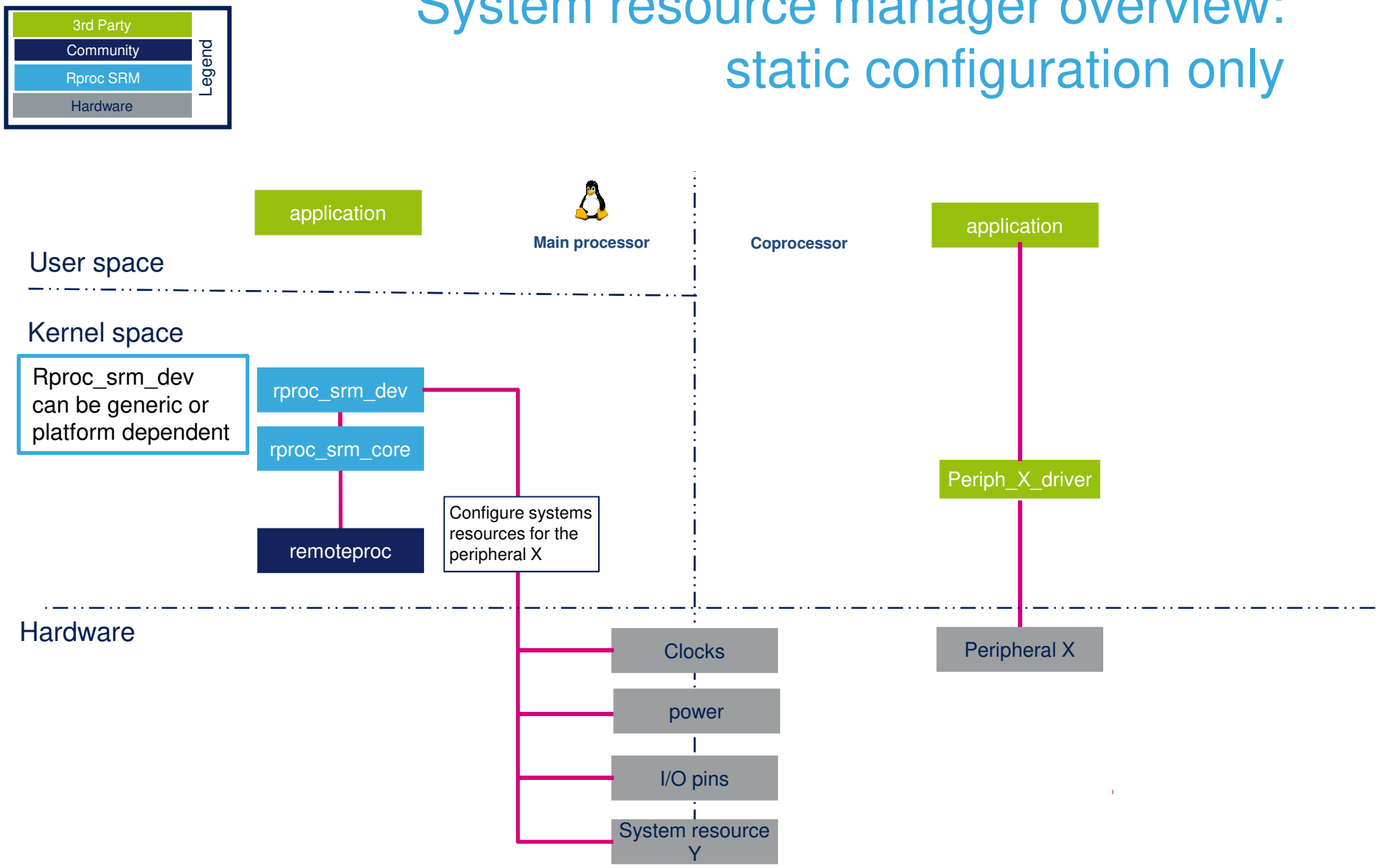
```
soc {
        i2C1: i2c@F0010000 {
                compatible = "st, i2c";
                clocks = <&rcc_clk I2C1_K>;
                pinctrl-0 = <i2c1_pins_a>;
                status = “disabled";
        }
        slave_proc0@30000000 {
                compatible = “st, slave_rproc”;
                reg = <0x30000000 0x10000>,
                resets = <&rcc_rst>;
                reset-names = "slave_core0_rst";
                clocks = <&rcc_clk I2C1_K>;
                pinctrl-0 = <i2c1_pins_a>;
        };
};
```

Alternative 2: phandle to peripheral node

- **Pro**
  - Only one node common for both core.
  - Reference to Linux core declaration to check availability for the remote processor.
- **Cons**
  - Design imposes that system resources are same for both cores to operate the peripheral.
  - Does it make sense to associate a driver to a device with « disabled » status?
  - How to handle specific platform system resources?

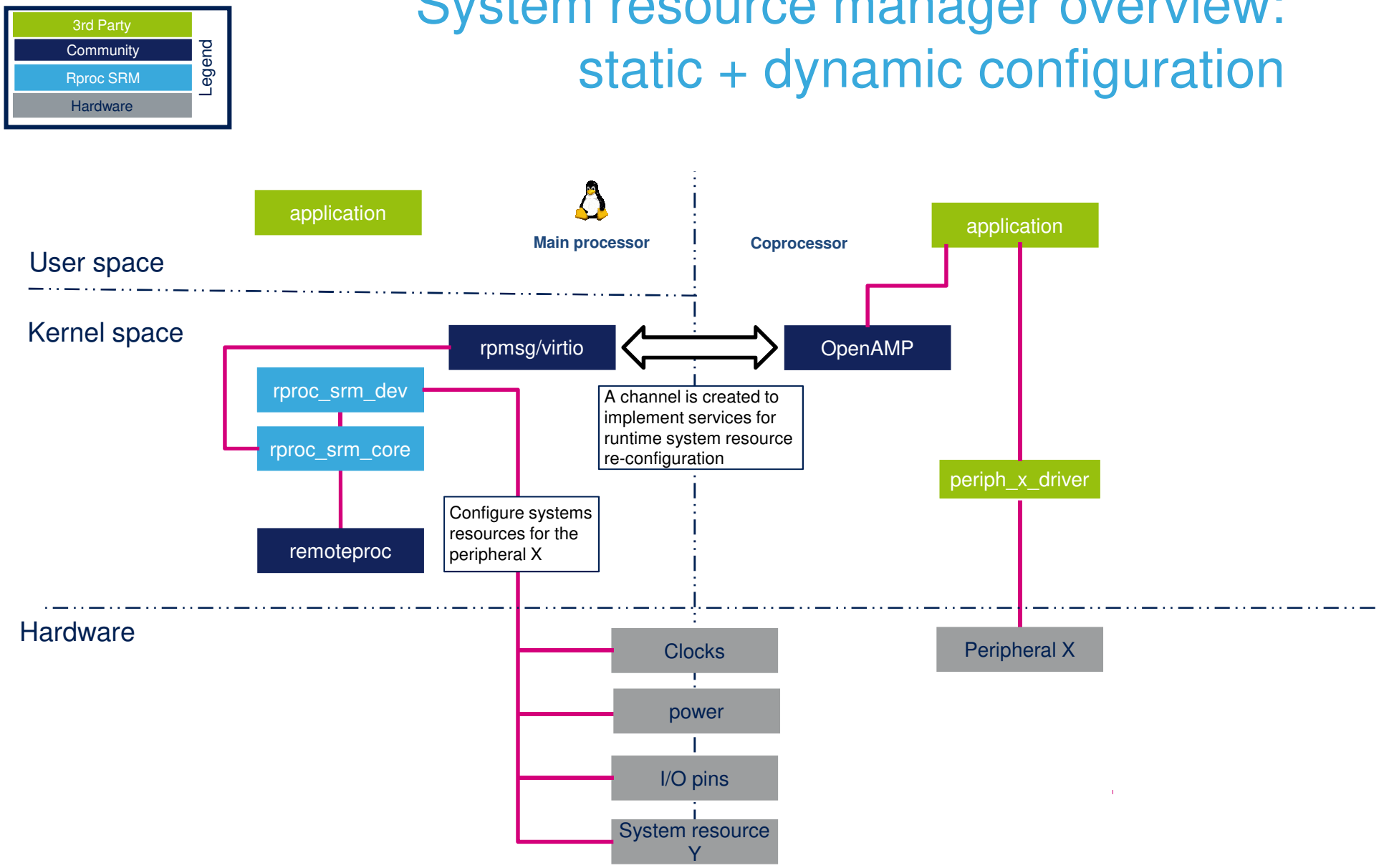
```
soc {
        i2C1: i2c@F0010000 {
                compatible = "st,i2c";
                clocks = <&rcc_clk I2C1_K>;
                pinctrl-0 = <i2c1_pins_a>;
                status = “disabled";
        }
        slave_proc0@30000000 {
                compatible = “st, slave_rproc”;
                …
                system_resources {
                        compatible = "rproc-srm-core";
                        res =<&i2C1>
                        res-name = « i2c1»
                };
        };
};
```

System resource manager overview: static configuration only



life.augmented

System resource manager overview: static + dynamic configuration



life.augmented