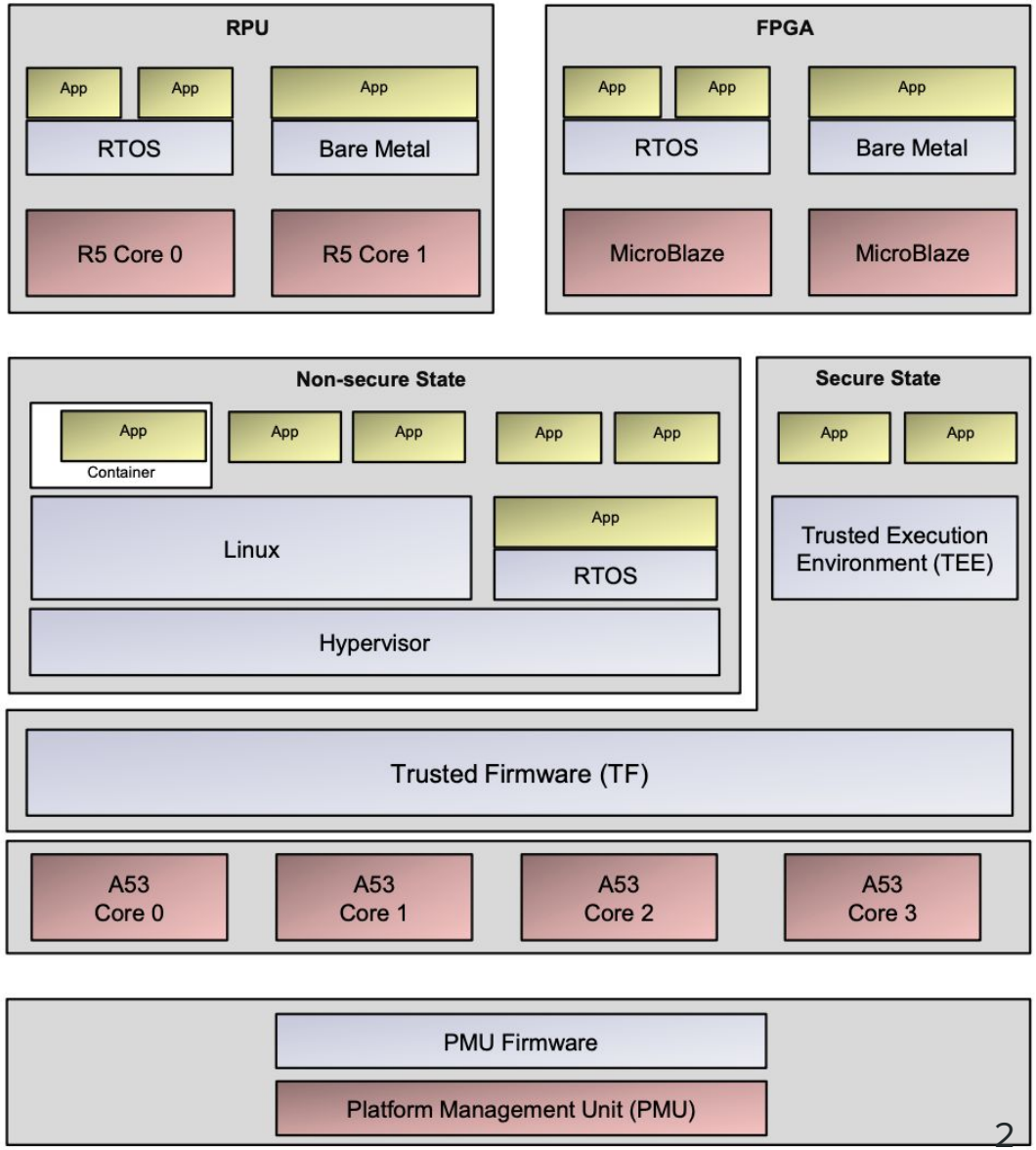
Heterogeneous Linux and RTOS Software Architecture for Low-Price RISC-V Cores

Ching-Chun Huang
Chi-Kuan Chiu
June 23, 2025
Open Source Summit North America

1931
國立成功大學
National Cheng Kung University

Modern Multicore Heterogeneity

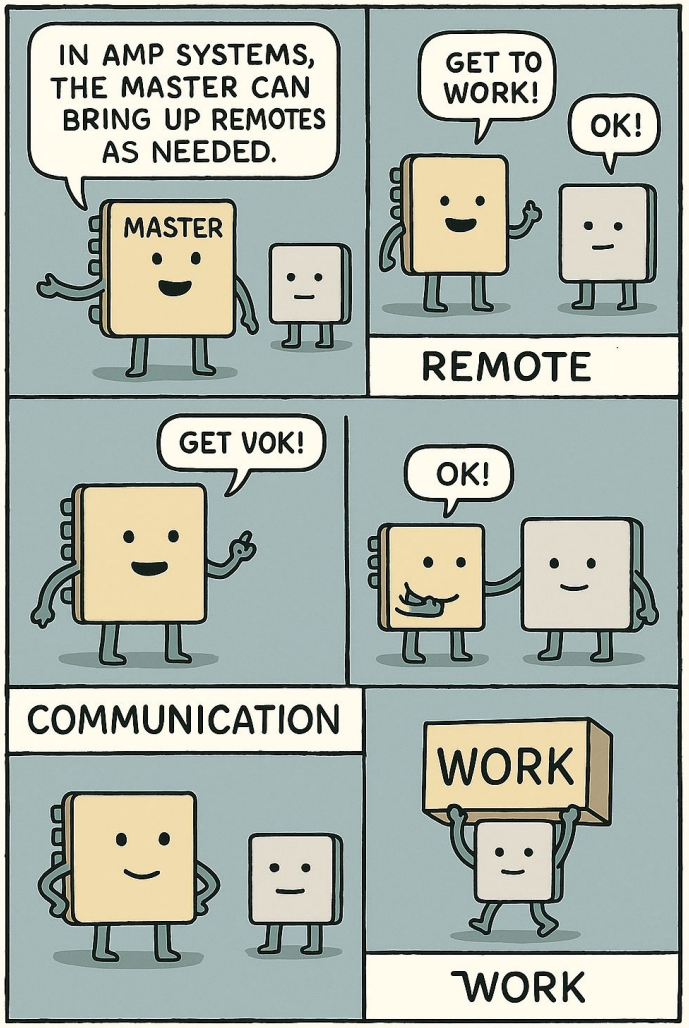
- Multiple core clusters:
  Cortex-A53, Cortex-R5, FPGA
- Multiple Execution Levels (EL)
  - EL0 – User space : Linux apps, Containers, RTOS apps
  - EL1 – OS space : Linux kernel, RTOS + RTOS apps
  - EL2 – Hypervisor : Xen, ..
  - EL3 – Firmware : Trusted Firmware



Source: The OpenAMP System Reference Project
https://www.openampproject.org/docs/presentations/OpenAMPSystemReferenceProject_2021Nov.pdf

Simplify S/W for heterogeneous environments via OpenAMP

- Key concepts
  - *Master*: CPU/software that boots first and manages other CPUs / software contexts
  - *Remote*: CPU/software context managed by the master software context
  - *Communication*: IPC mechanisms enable workload distribution and coordination
- OpenAMP is a community-driven initiative under Linaro, aimed at unifying the softwar stack needed to support AMP environments.



Reference: Linux and Zephyr “talking” to each other in the same SoC, Diego Sueiro, Sepura
Embedded Linux Conference Europe 2018

OpenAMP Framework

Challenges: inter-processor synchronization and communication, isolating and protecting shared resources, cache coherency, efficient power usage, etc.

Communications between APU and RPU

SoC

APU

Core1

IPC

Core2

B-M

Core3

Core4

RTOS

?

RPU

Core1

B-M

IPC

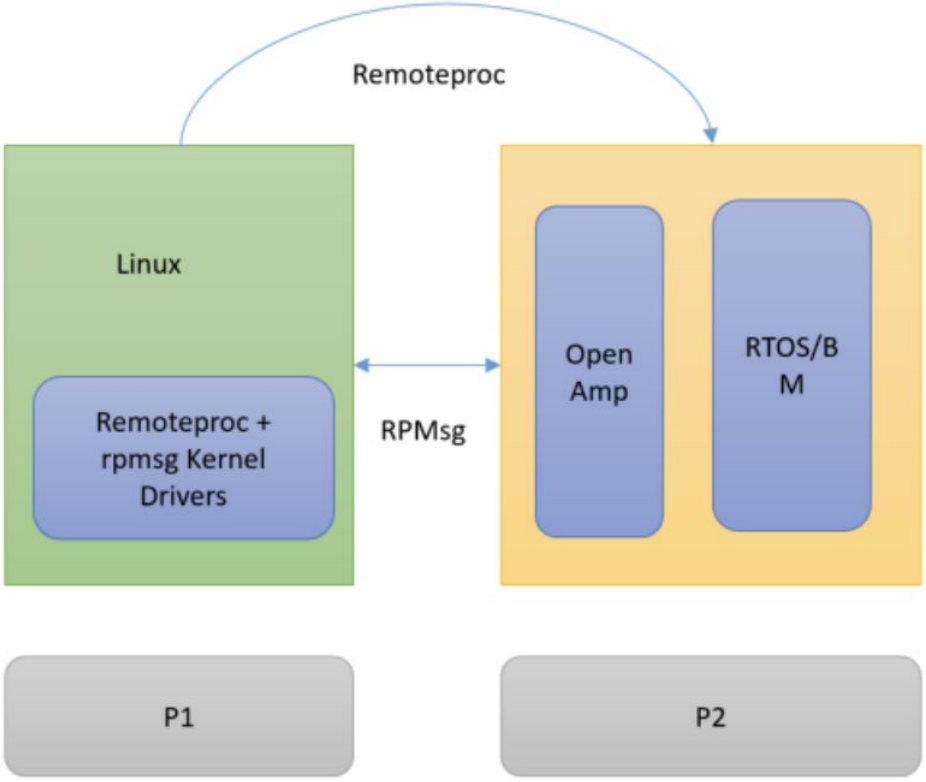
Core2

RTOS

Source: OpenAMP Framework
https://static.linaro.org/connect/bkk19/presentations/bkk19-204.pdf

Linux AMP (Asymmetric Multiprocessing)

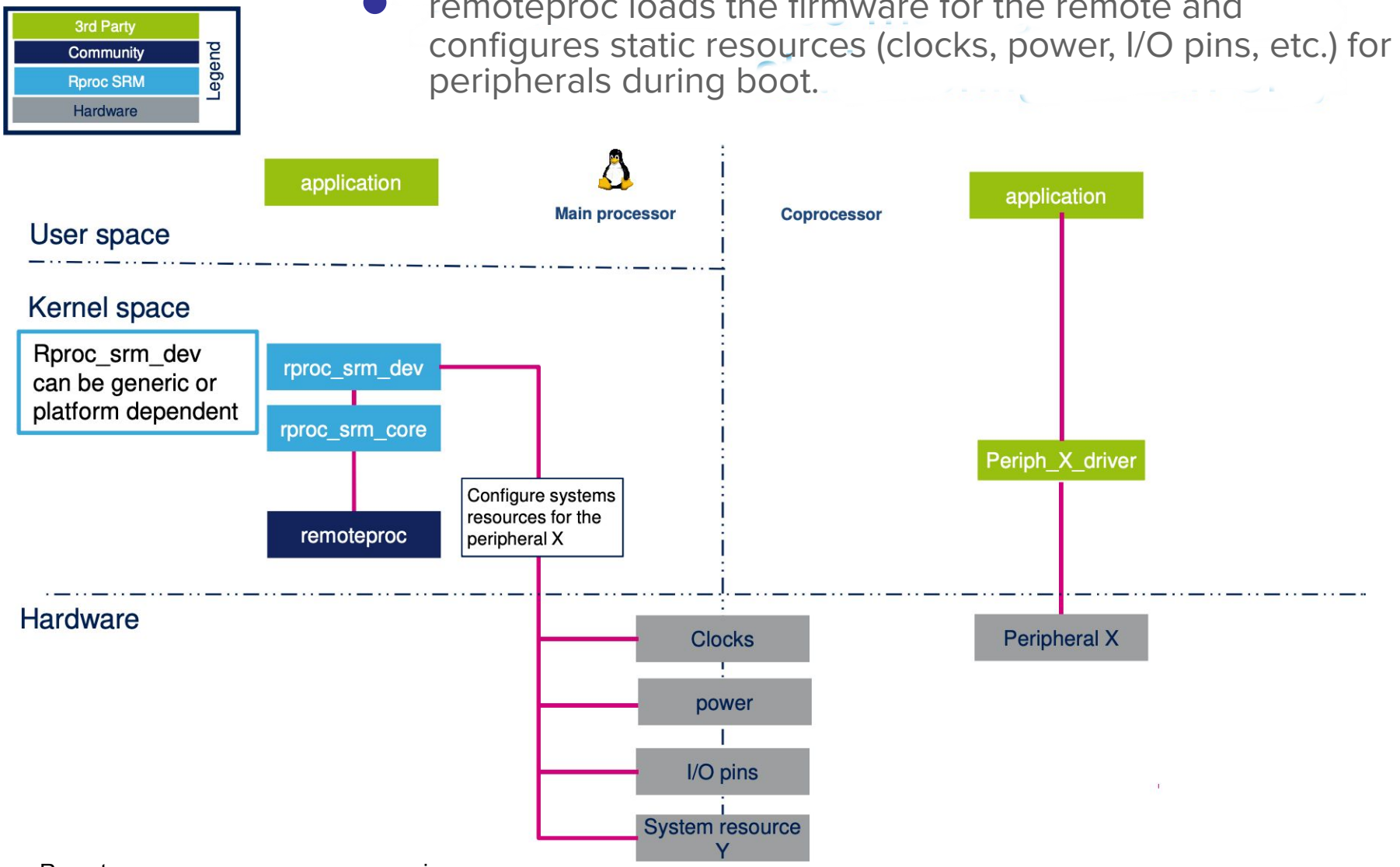
- **Remoteproc** (Remote Processor), allows a Linux-based master core to manage remote processors
- **RPMsg** (Remote Processor Messaging), provides a lightweight IPC mechanism between Linux kernel drivers and those remote processors.



Source: Introduction to OpenAMP Library
https://www.openampproject.org/docs/whitepapers/Introduction_to_OpenAMPlib_v1.1a.pdf

Remoteproc (static)

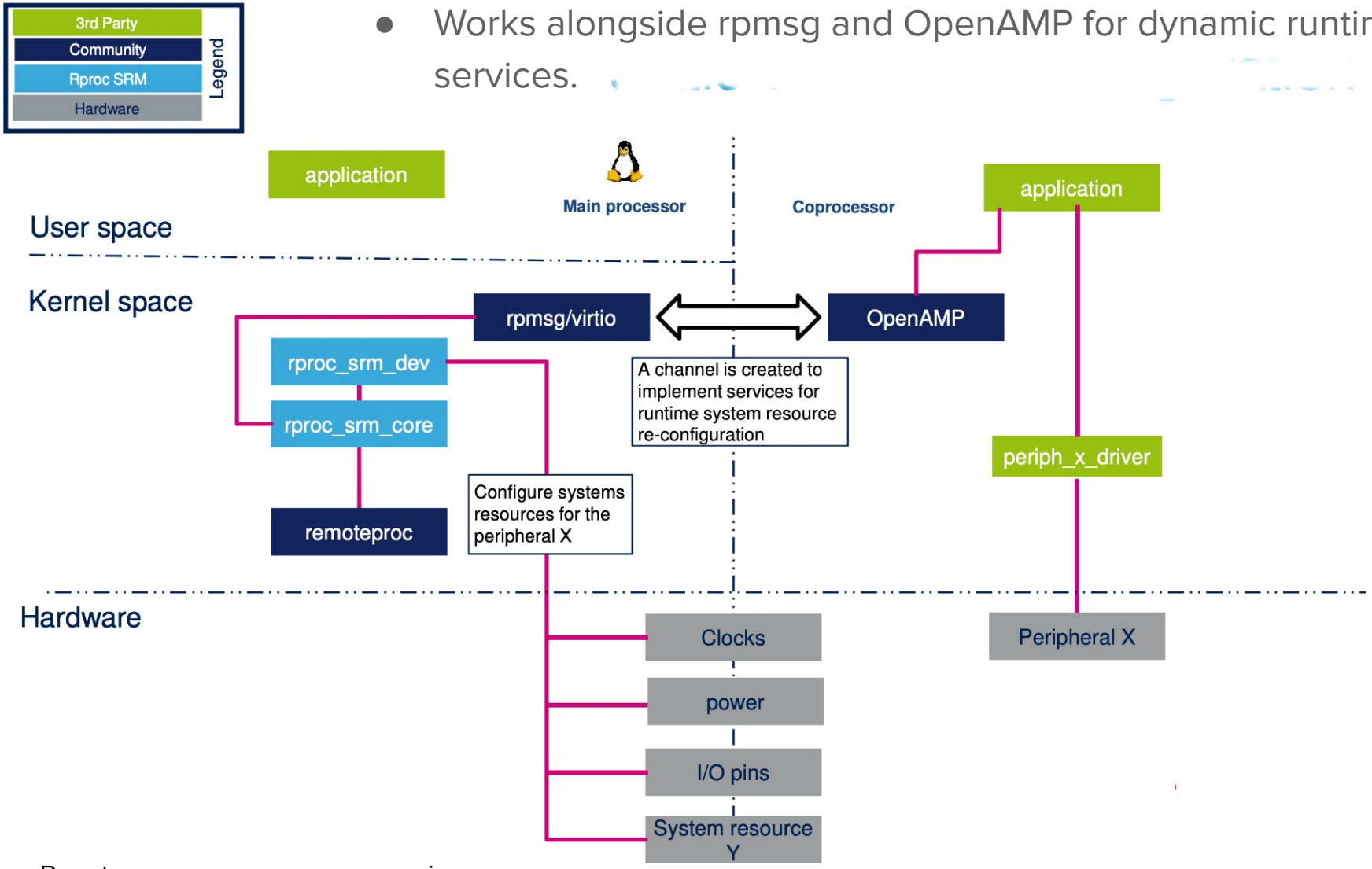
- In kernel space, the remoteproc framework is integrated with rproc_srm_core and rproc_srm_dev, which may be generic or platform-specific.
- remoteproc loads the firmware for the remote and configures static resources (clocks, power, I/O pins, etc.) for peripherals during boot.



Source: Remote proc resource manager overview
https://openamp.github.io/docs/mca/remoteproc-resource-manager-overview.pdf

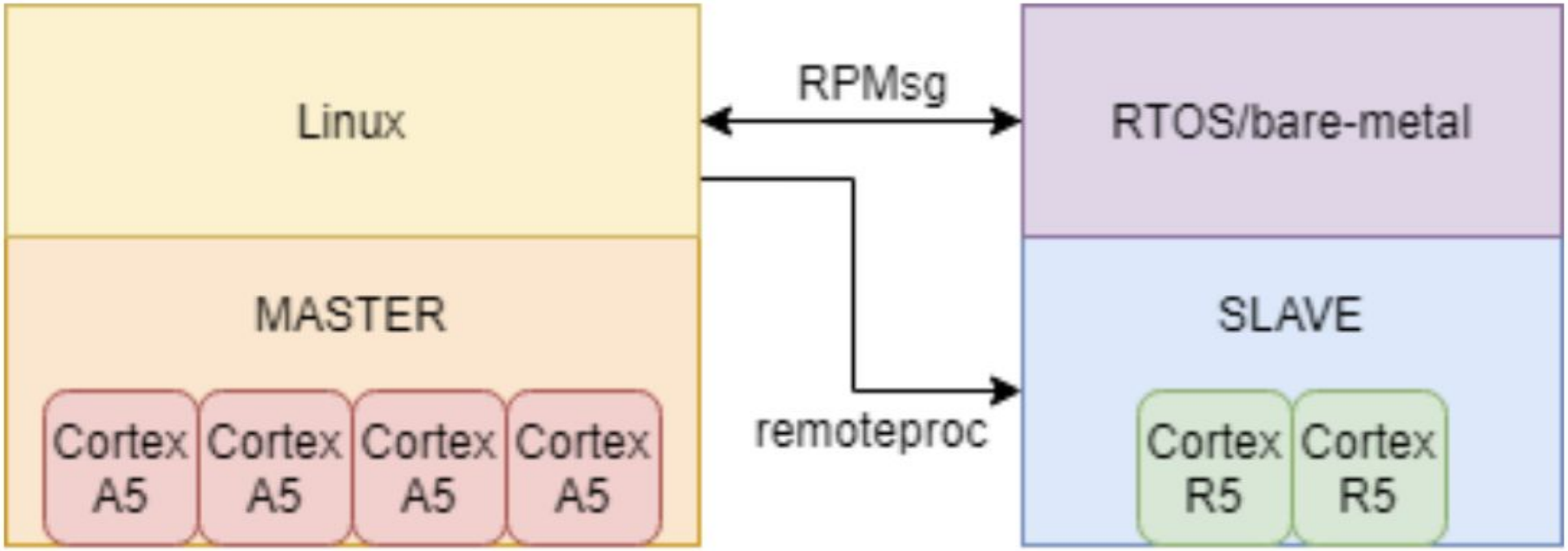
Remoteproc (static + dynamic)

- The coprocessor's application and peripheral driver are now aware of and responsive to runtime changes, interacting via OpenAMP services.
- Works alongside rpmsg and OpenAMP for dynamic runtime services.



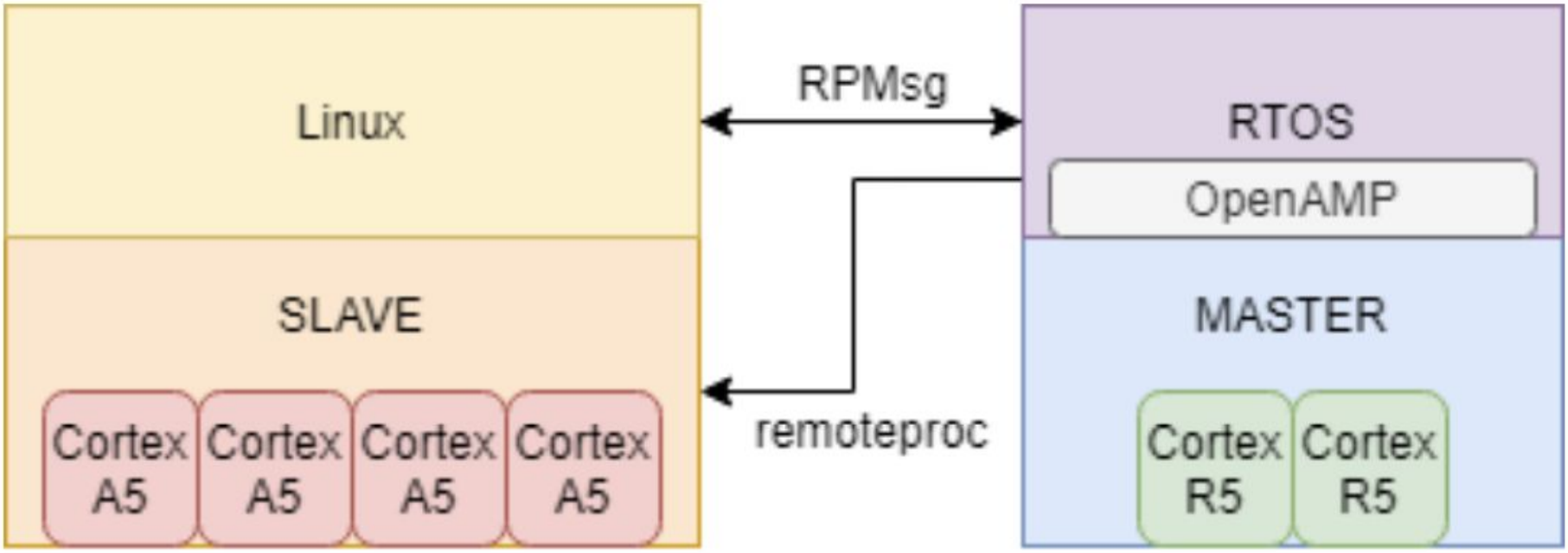
Source: Remote proc resource manager overview
https://openamp.github.io/docs/mca/remoteproc-resource-manager-overview.pdf

Remoteproc + RPMsg with RTOS/bare-metal



- libmetal is used for integration

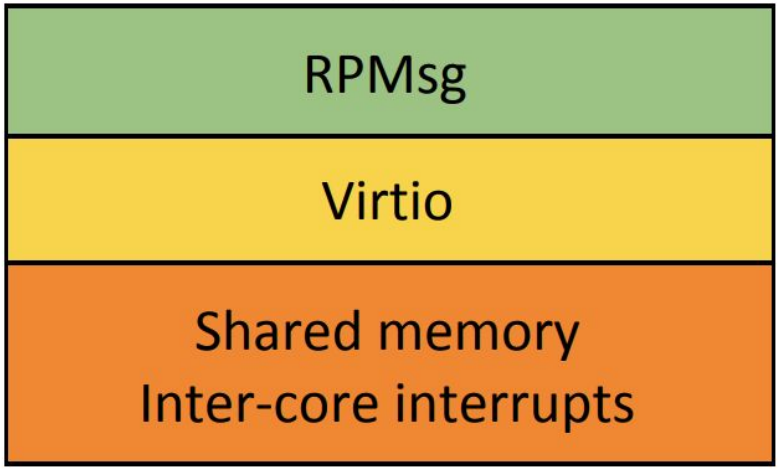
Remoteproc + RPMsg: Flexibility



- RTOS/OpenAMP stack handles both remoteproc and RPMsg functionality, managing Linux environment and reversing the traditional master-slave role

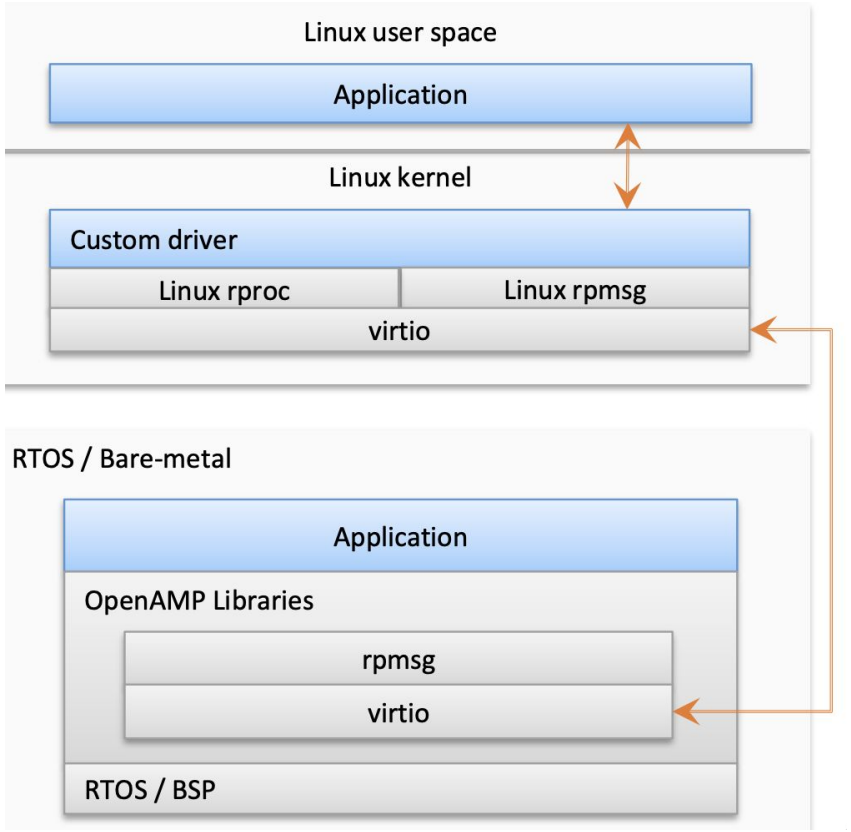
RPMsg Protocol

- Enables IPC between the master and remote processors in an AMP system.
  - Each RPMsg instance represents a dedicated communication channel
- RPMsg relies on 2 fundamental hardware features:
  - Shared memory, which serves as the transport buffer for message data and must be accessible by both processors
  - Inter-core interrupts, used to signal when new messages are available or actions are required



VirtIO

- Abstraction layer over devices in a para-virtualized hypervisor environment
- Implements OASIS VirtIO standard for shared memory management and device virtualization
- Directly applicable to OpenAMP configurations for AMP systems

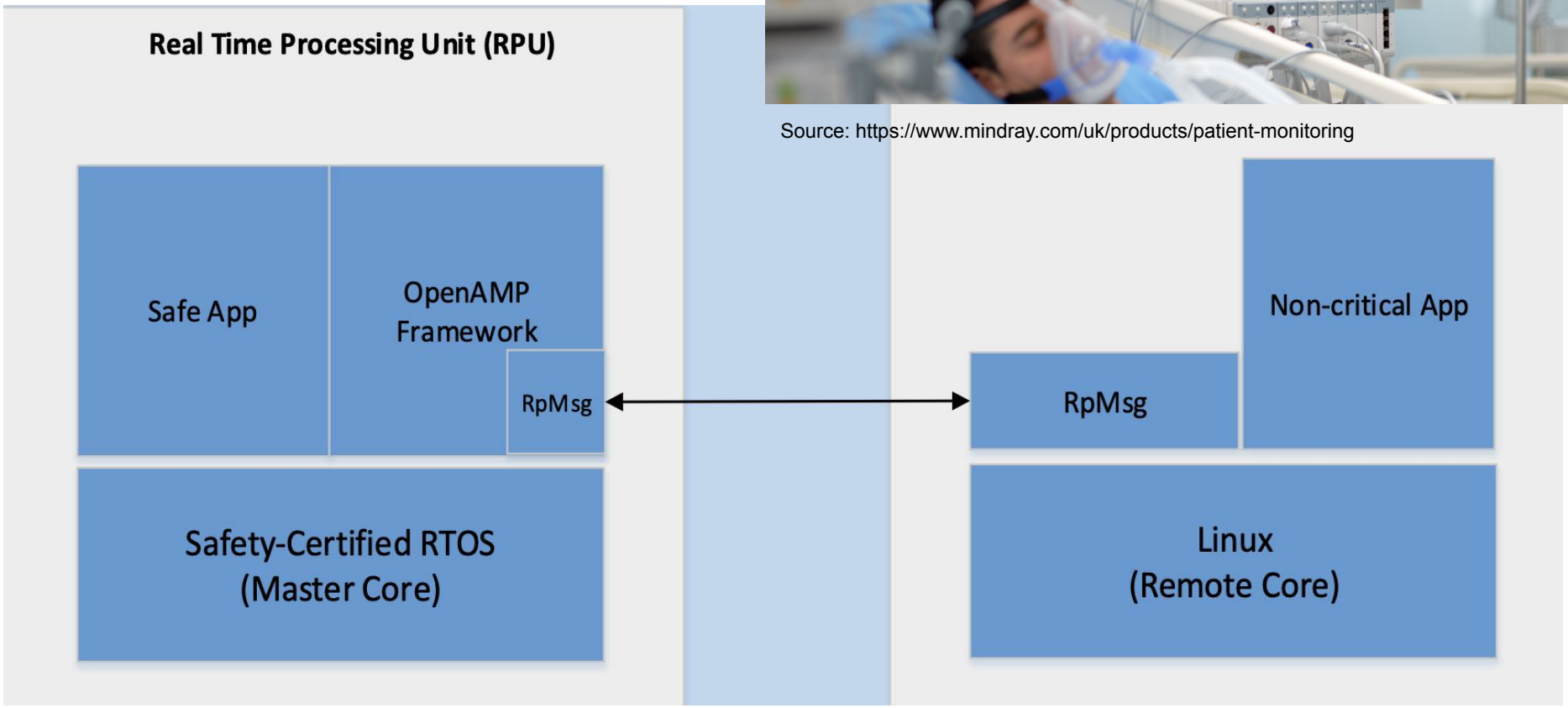


- RPMsg Applications
- RPMsg Protocol Layer:
- VirtIO Transport Layer: acts as the bridge between the RPMsg protocol and actual hardware, managing
  - Shared memory regions
  - Ring buffers (VirtIO queues)
  - Buffer allocation and synchronization
- Physical Layer (shared memory + interrupts)
  - Shared RAM used to exchange buffers
  - Mailbox or inter-processor interrupt mechanism used to notify each side when a message is ready

Use Cases

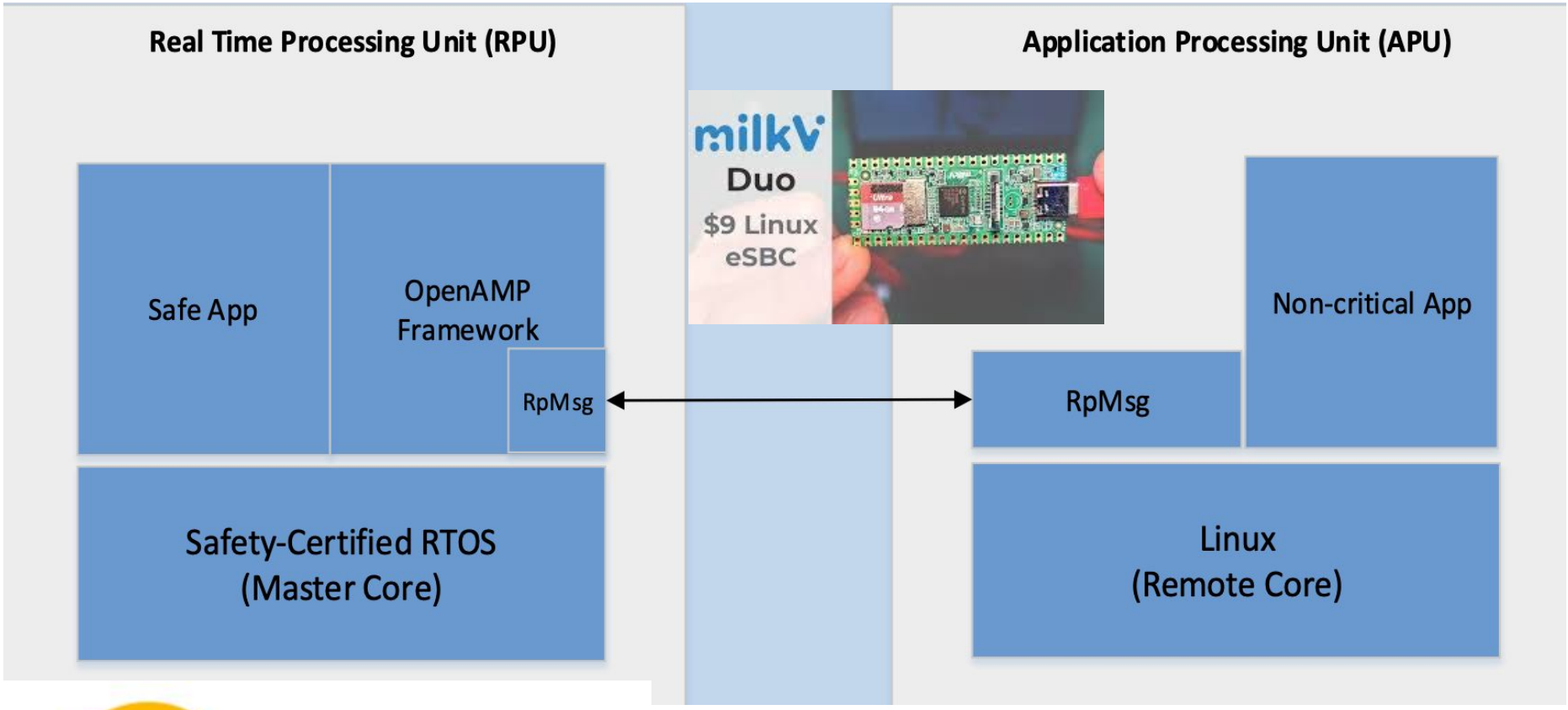
Mixed Safety-Critical System
e.g., Patient monitor system

Source: https://www.mindray.com/uk/products/patient-monitoring



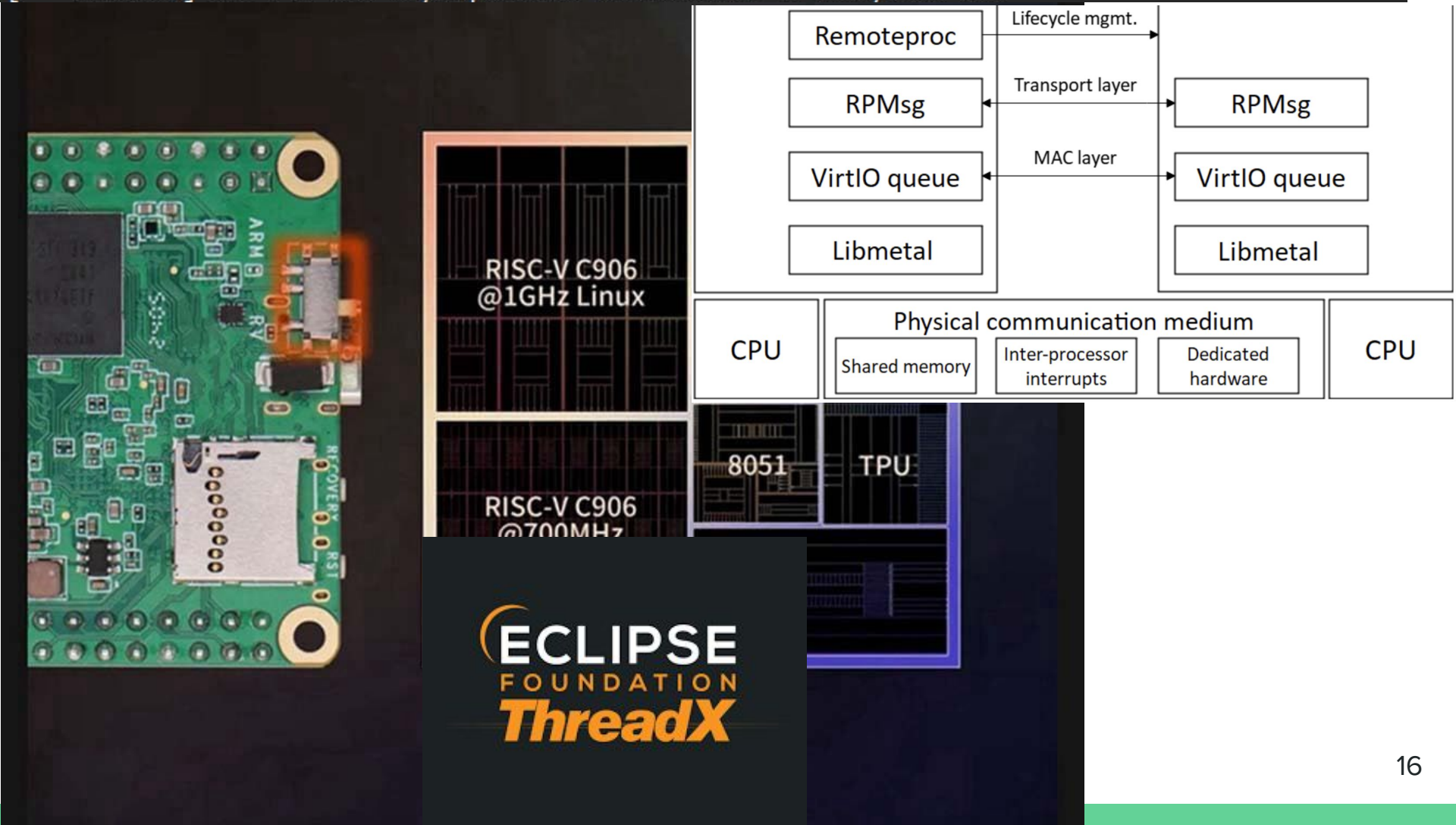
- RPU, running functional safety-certified software, is responsible for collecting patient vital signs from sensors and monitoring them in real time.
- APU runs a high-level OS that displays the data on an LCD and offers network connectivity, but it's not safety-critical.

Build fully open source use case
Using low-cost RISC-V hardware



ECLIPSE THREADX

```
[    1.432926] hub 2-0:1.0: 1 port detected
[    1.437399] imx-cpufreq-dt imx-cpufreq-dt: cpu speed grade 7 mkt segment 2 supported-hw 0x80 0x4
[    1.448704] sdhci-esdhc-imx 30b50000.mmc: Got CD GPIO
[    1.448985] remoteproc remoteproc0: imx-rproc is available
[    1.459340] remoteproc remoteproc0: attaching to imx-rproc
[    1.466314] virtio_rpmsg_bus virtio0: creating channel rpmsg-virtual-tty-channel-1 addr 0x1e
[    1.474865] virtio_rpmsg_bus virtio0: rpmsg host is online
[    1.480379]  remoteproc0#vdev0buffer: registered virtio0 (type 7)
[    1.486496] remoteproc remoteproc0: remote processor imx-rproc is now attached
[    1.490233] mmc1: SDHCI controller on 30b50000.mmc [30b50000.mmc] using ADMA
[    1.494316] On-module +V3.3_ETH: supplied by On-module +V3.3 (BUCK4)
[    1.659989] usb 1-1: new high-speed USB device number 2 using xhci-hcd
```

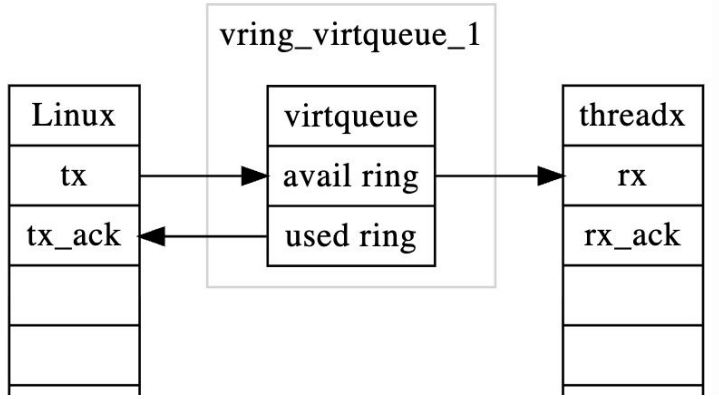


Send messages from Linux to ThreadX(1)

```
/* struct rpmsg_endpoint_info - endpoint info representation */
struct rpmsg_endpoint_info {
   char name[32];
   __u32 src, dst;
};
```

```
typedef struct the_message {
  unsigned int DATA;
} THE_MESSAGE, *THE_MESSAGE_PTR;
```

```
int main() {
  int status;
  struct rpmsg_endpoint_info ept_info =
{"rpmsg-openamp-demo-channel", 0x1e, 0x1e};
  int fd = open("/dev/rpmsg_ctrl0", O_RDWR);
  /* create endpoint interface */
  ioctl(fd, RPMSG_CREATE_EPT_IOCTL, &ept_info);
  /* create endpoint */
  int fd_ept = open("/dev/rpmsg0", O_RDWR);
```



Linux uses tx to send messages and tx_ack to handle acknowledgments.

ThreadX uses rx to receive and rx_ack to acknowledge receipt.

Source: https://github.com/HenryChaing/duo-examples/tree/amp/rpmsg_userspace

Send messages from Linux to ThreadX(2)

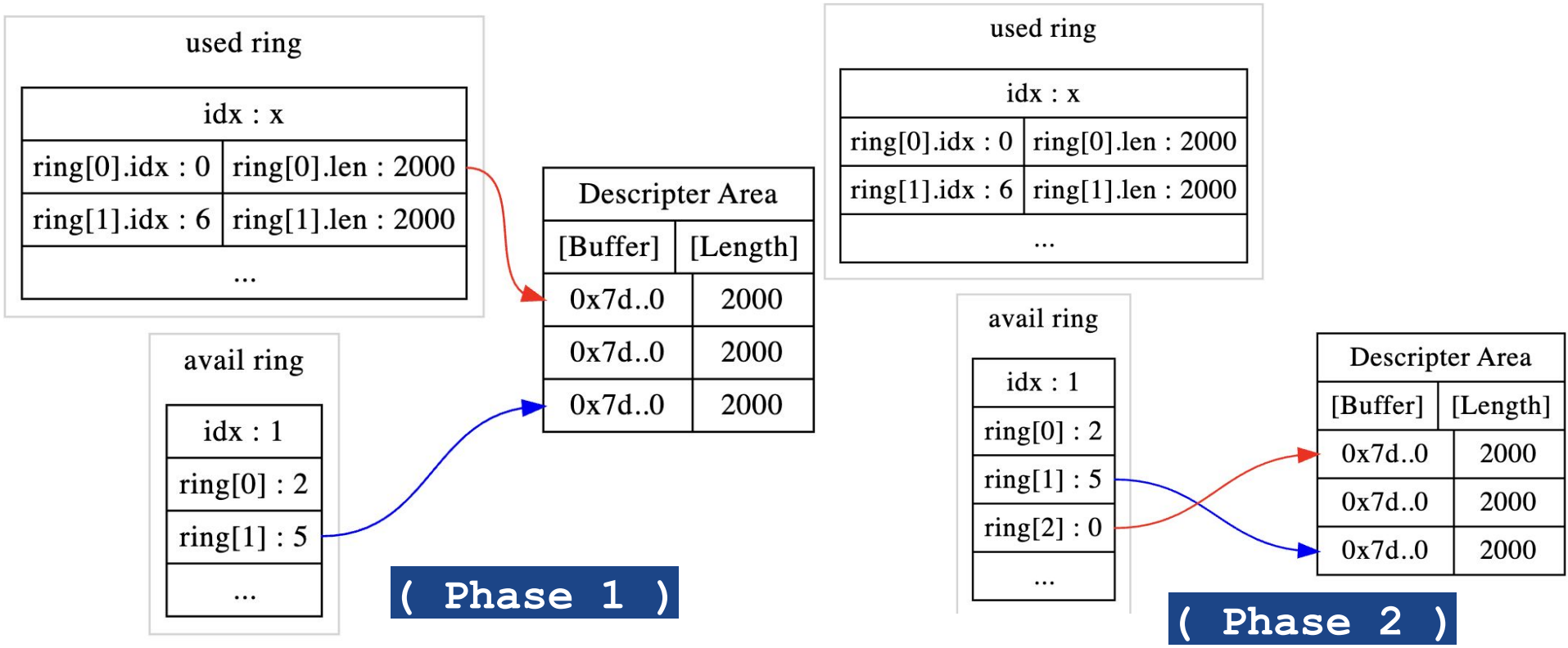
```
THE_MESSAGE r5_data = {.DATA = 0};

/* send data to remote device */
status = write(fd_ept, &r5_data, sizeof(THE_MESSAGE));

/* receive data from remote device */
for (int i = 0; i < TC_TRANSFER_COUNT; i++) {
  status = read(fd_ept, &r5_data, sizeof(THE_MESSAGE));
  printf("r5_data.DATA: %d\n",r5_data.DATA);
  status = write(fd_ept, &r5_data, sizeof(THE_MESSAGE));
}
/* destroy endpoint */
ioctl(fd_ept, RPMSG_DESTROY_EPT_IOCTL);
```

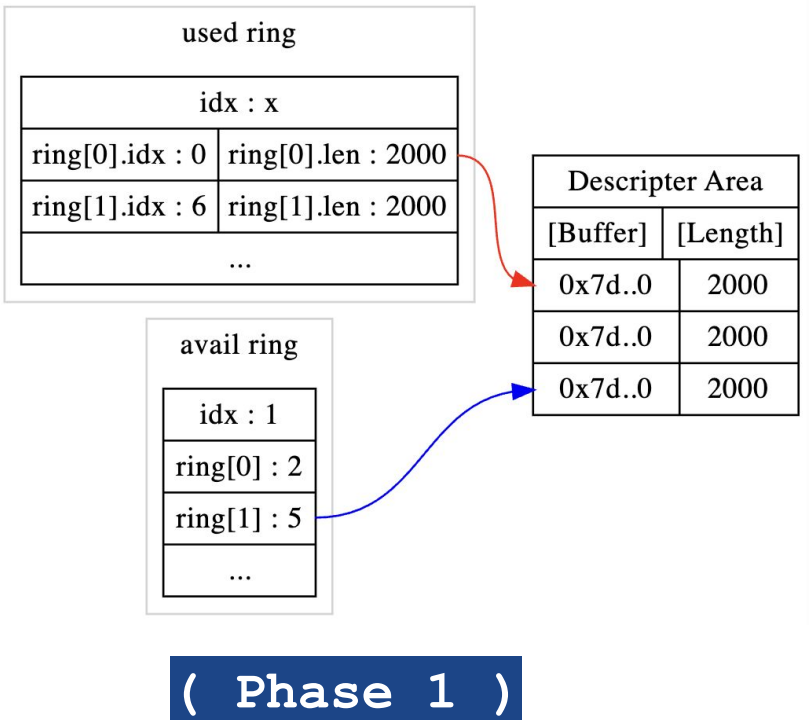
Source: https://github.com/HenryChaing/duo-examples/tree/amp/rpmsg_userspace

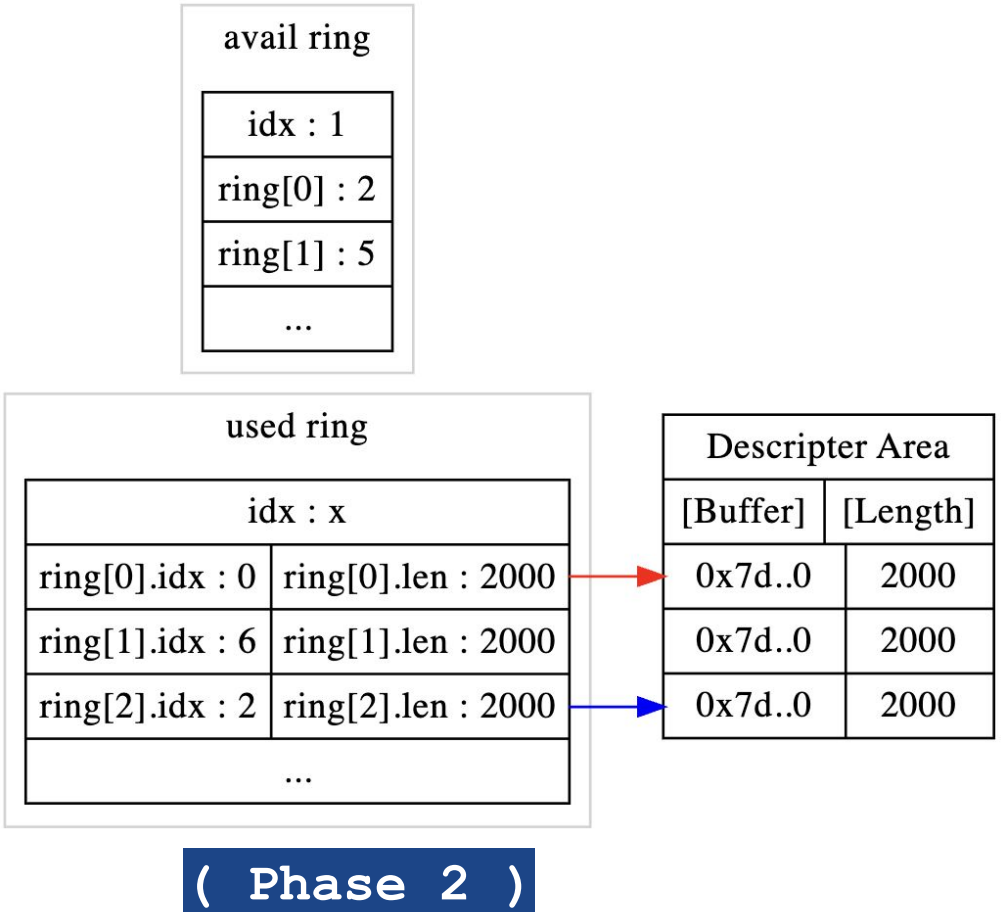
Send messages from Linux to ThreadX(3)




```
for (int i = 0; i < TC_TRANSFER_COUNT; i++) {
  status = read(fd_ept, &r5_data, sizeof(THE_MESSAGE));
  printf("r5_data.DATA: %d\n",r5_data.DATA);
  status = write(fd_ept, &r5_data, sizeof(THE_MESSAGE));
}
```

Send messages from ThreadX to Linux





Source: https://github.com/HenryChaing/duo-examples/tree/amp/rpmsg_userspace

Build everything from source(1) ⇒ repo1 / repo2

- Clone the Buildroot SDK Repository

```
$ git clone --recurse \
      https://github.com/HenryChaing/duo-buildroot-sdk-v2
```

- Start the Buildroot Compilation Process

```
$ ./build.sh lunch  # (Choose option 3.)
```

- Use Shell Script Functions to Build Individual Components

```
$ ./source step_by_step_compilation.sh
$ build_rtos && build_uboot && pack_sd_image
```

Build everything from source(2)

- Flash the Following Image Using a Tool Like balenaEtcher

  $ ./install/soc_sg2002_milkv_duo256m_musl_riscv64_sd/milkv-duo256m-musl-riscv64-sd.img

- Copy the Remote Processor Shell Script and RTOS Executable to the Root Filesystem

  $ **cp ./remoteproc.sh ./threadx/cvitek/install/bin/cvirtos.elf /media/$USER/rootfs/root/**

- Boot the Milk-V Board and Install the ADB Tool on the Host

  $ **sudo apt-get -y install android-tools-adb**

  $ **adb shell**

Build everything from source(3)

- Enable Remoteproc and Load the Executable to the Coprocessor

```
[Target]$ sh remoteproc.sh
```

- Clone the Userspace Application Development Repository

```
$ git clone https://github.com/HenryChaing/duo-examples
```

- Compile the Required Applications for the Target Device

```
$ cd rpmsg_userspace && make all
```

- Copy the Compiled Applications to the Device's Root Filesystem

```
$ cp mailbox_pingpong mailbox_led /media/$USER/rootfs/root/
```

- Boot the Milk-V Board and Run the Applications

```
[Target]$ sh remoteproc.sh
[Target]$ ./mailbox_led &
[Target]$ ./mailbox_pingpong
```

remoteproc.sh Content

```
# cvirtos.elf is the image name of ThreadX
cp cvirtos.elf /lib/firmware
# Setup firmware loading path
echo cvirtos.elf > /sys/class/remoteproc/remoteproc0/firmware
# Remoteproc restarting
echo start > /sys/class/remoteproc/remoteproc0/state
sleep 1
echo stop > /sys/class/remoteproc/remoteproc0/state
echo start > /sys/class/remoteproc/remoteproc0/state
```

Conclusion

Conclusion

- Systems developed using these heterogeneous SoCs, characterized by heterogeneity in both hardware and software, are generally termed as AMP systems.
  - In AMP systems, it is typical for software running on a master to bring up software/firmware contexts on a remote on a demand-driven basis and communicate with them using IPC mechanisms to offload work during runtime.
- Applications
  - Edge AI acceleration
  - Functional safety compliance
  - Security isolation requirements

Thanks For Listening !

contact: Ching-Chun (Jim) Huang
<jserv@ccns.ncku.edu.tw>