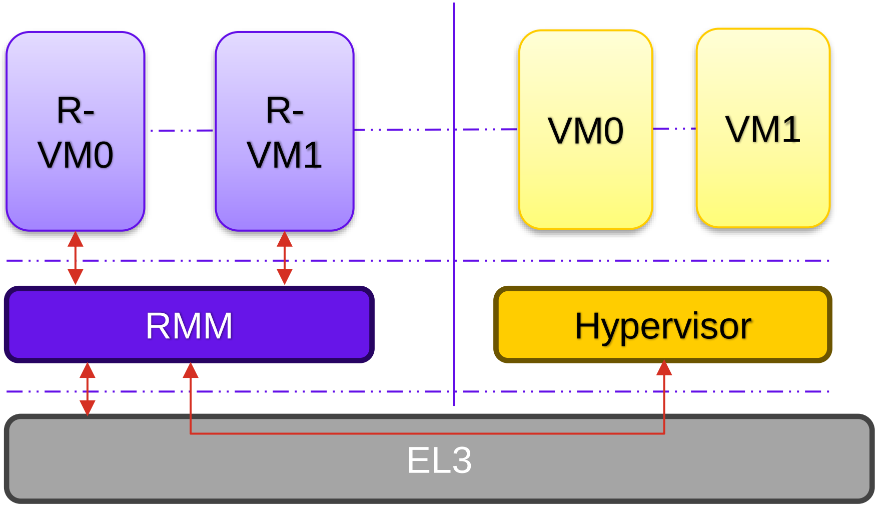
linaro Connect 2025

RMM Deprivileging using VHE

RMM EL0 app framework

Soby Mathew
TF-RMM Tech Lead

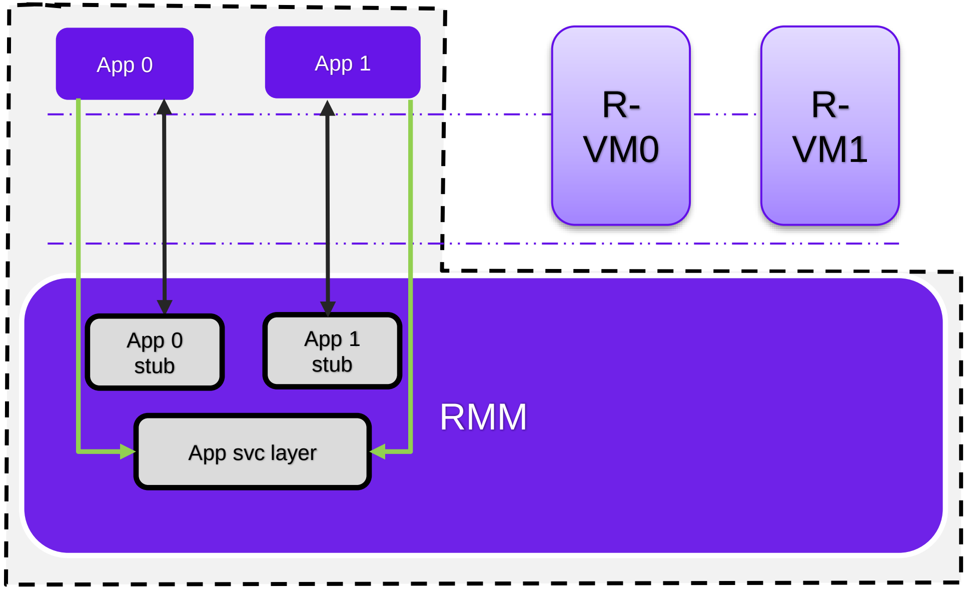
Introduction



RME CCA software flow

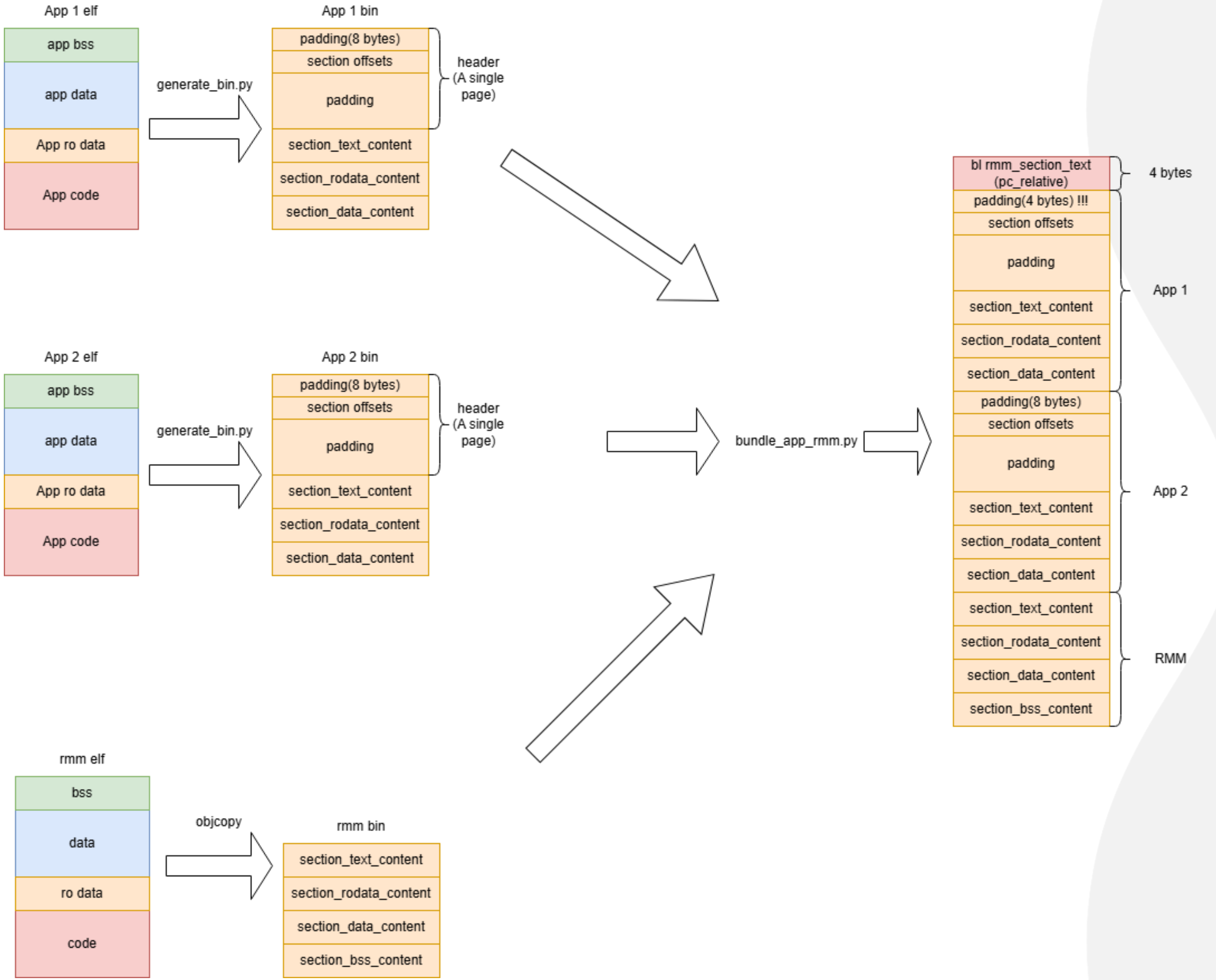
- EL0 app support is a mechanism to Deprivilege parts of RMM by running them at EL0 (using Virtual Host Extension)
  - RMM functionality sandboxed as a EL0 Application.
  - Isolated address space for EL0 execution
- Build and binary isolation with ability to build shared components specific to EL0 App needs.
  - Can share or build separate obj files for shared source files.
  - Separate Linker script and separate binary for apps.

Introduction



RMM with EL0 apps

- A given App can have several “instances”.
  - Conceptually like threads in a process
  - Every instance receives its own stack and heap which is private to the instance.
    - In addition, every instance has its own shared buffer, Xlat table etc.
  - The memory for instance can be allocated from per-cpu buffer in RMM, Auxiliary granule storage from REC (Realm Execution Context) or PDEV (Physical Device Object).



App 1 elf

app bss

app data

App ro data

App code

generate_bin.py

App 1 bin

padding(8 bytes)

section offsets

padding

section_text_content

section_rodata_content

section_data_content

header (A single page)

App 2 elf

app bss

app data

App ro data

App code

generate_bin.py

App 2 bin

padding(8 bytes)

section offsets

padding

section_text_content

section_rodata_content

section_data_content

header (A single page)

rmm elf

bss

data

ro data

code

objcopy

rmm bin

section_text_content

section_rodata_content

section_data_content

section_bss_content

bundle_app_rmm.py

bl rmm_section_text (pc_relative)

4 bytes

padding(4 bytes) !!!

section offsets

padding

section_text_content

section_rodata_content

section_data_content

App 1

padding(8 bytes)

section offsets

padding

section_text_content

section_rodata_content

section_data_content

App 2

section_text_content

section_rodata_content

section_data_content

section_bss_content

RMM

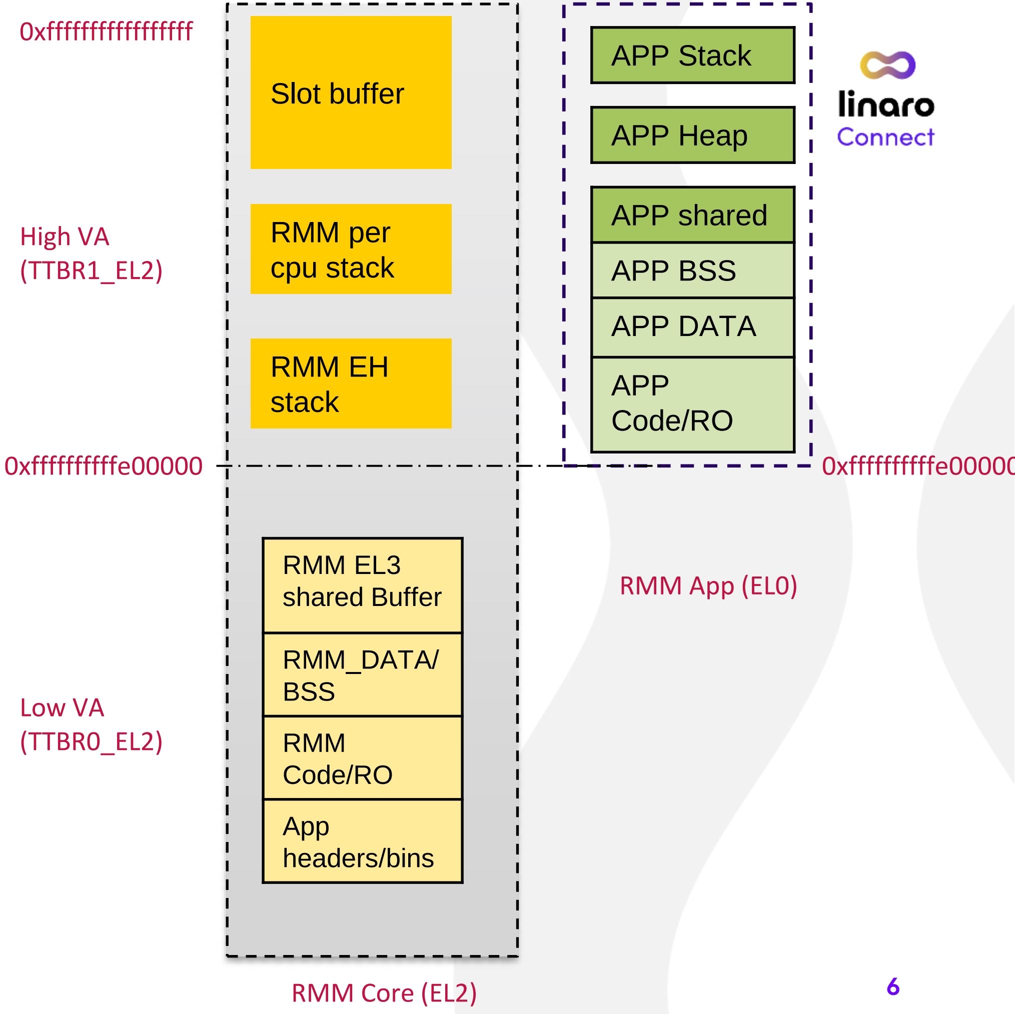
Build packaging

- Apps are built as separate elf files
- A python script is used to
  - Extract the binary content of the relevant sections
    - .text, .rodata, .data
  - Prepend a header to the extracted sections
- Elf sections must be page aligned, so that direct mapping .text, .rodata in the app memory is possible
- Header format is defined in RMM source as a C structure, the python script needs to be kept up-to-date on header format change
  - Header contains a header version, elf section offsets and lengths, stack/heap page count, app name and app id

Note : When executing the fake_host binary, the rmm_core and el0 app binaries must be specified.

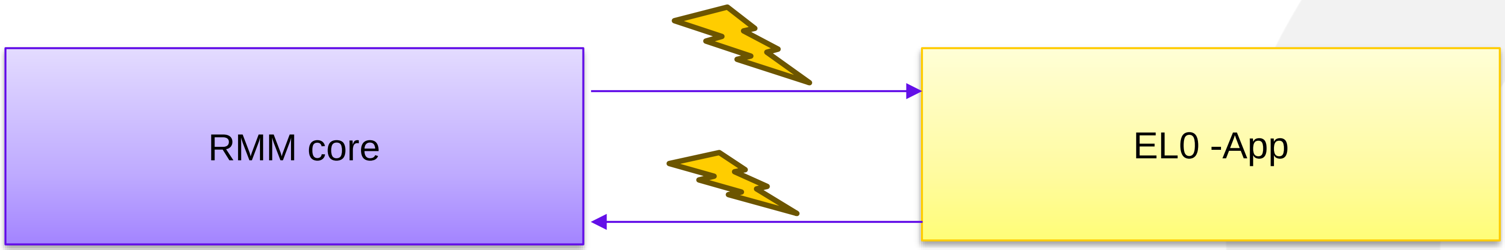
Memory layout

- The EL0 App runs in High VA space
  - The High VA is private to each CPU.
- Every app is assigned a unique ID which is used as its ASID
  - All instances of the app have the same ASID.
- Every app instance has its own Translation Table, Register context, Stack, Heap and a Shared buffer (4KB).
  - The shared buffer and Heap can be used for communicating between EL0 app and RMM core.



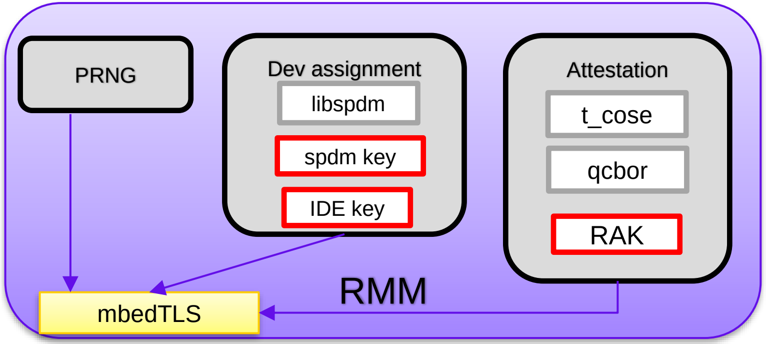
Security Model

- Focus on RMM security
  - designed to enhance the security of RMM by sandboxing sensitive data and complex functionality from the rest of the RMM address space.
  - By moving sensitive data to EL0 app framework can help reduce certain CPU vulnerabilities, such as those related to speculative execution, from leaking information.

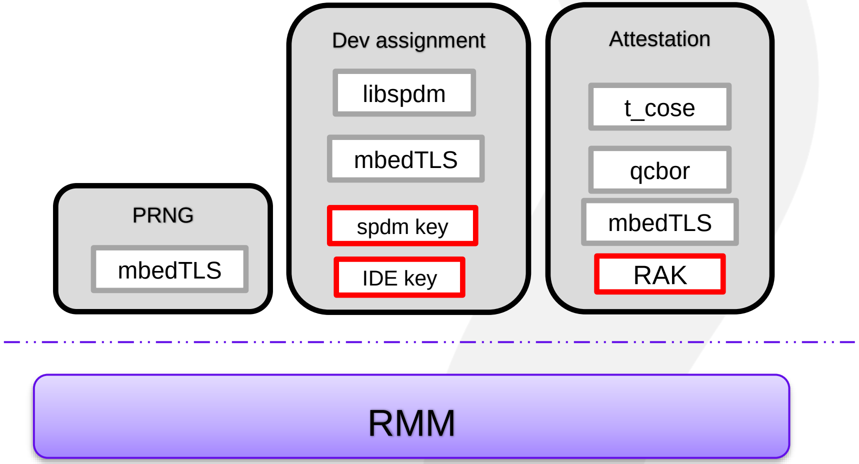


RMM with Apps

- Some problems solved with App framework
  - MbedTLS sharing issues
    - API/version mismatches and SIMD enablement problem.
  - The surface exposure of RMM Core to Non-Secure world is much reduced during SPDM interaction with Devices.
  - Allows NS yield for SPDM comms flow.



Before EL0 app framework



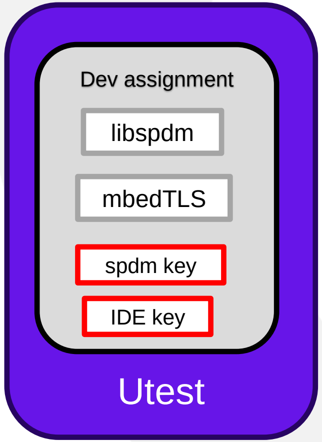
After EL0 app framework

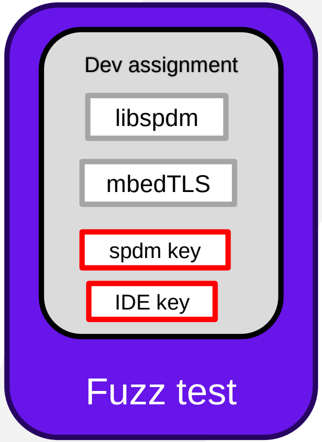
Drawbacks / known issues

- Debugging issues
    - RMM core offset need to taken into consideration when loading symbol info into Debugger.
    - Every App runs in the same High Address space
        - Breakpoints in the EL0 address space may match several apps.
    - Use ASID field in TTBR1_EL2 during Debugging to work out the current running App.
- Entering an EL0 app and exiting has a cost for performance
    - The usecases where this is enabled is not considered performance critical.
- The Auxiliary granules attached to REC and PDEV objects are mapped to RMM code at runtime currently
    - We can map and unmap on demand when APP is entered but since we are not able to measure impact on performance, we haven't done this yet.
- The BSS for apps are allocated from RMM BSS. This will be rectified once RMM can allocate memory from Realm carveout at boot time.

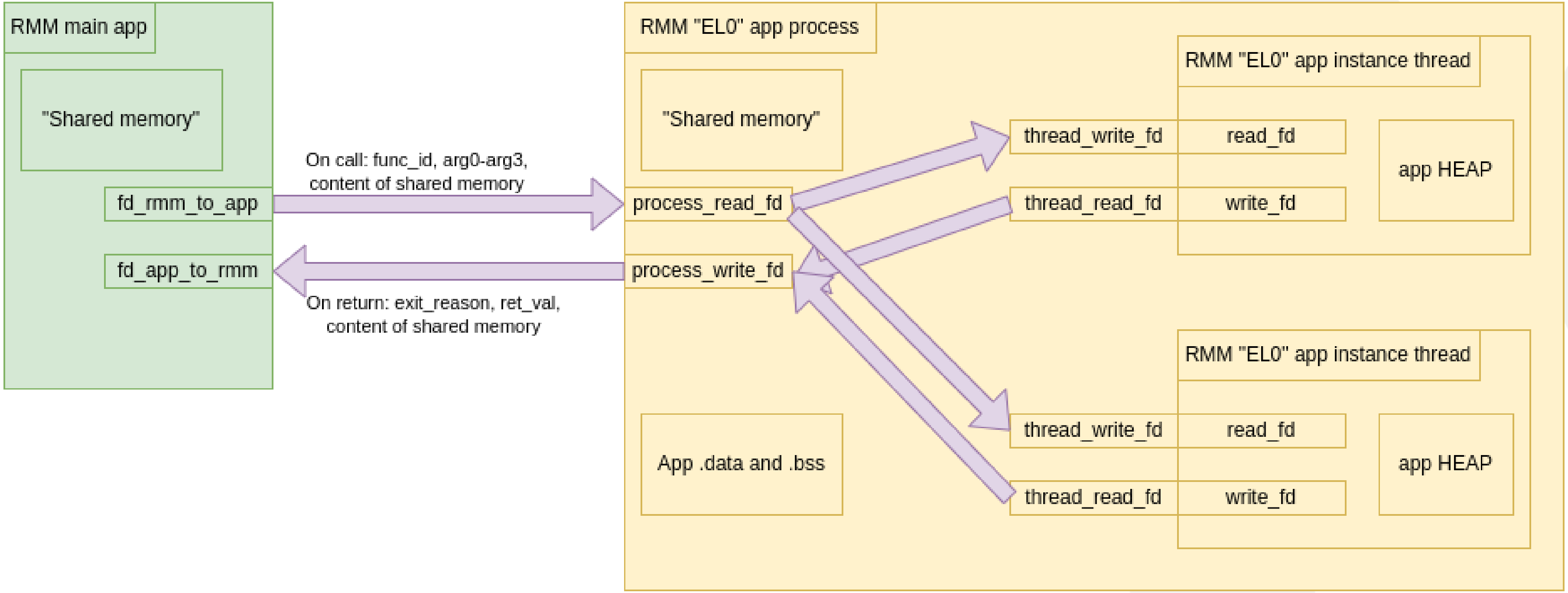
Future Direction

- Enable Unit testing / Fuzz testing of EL0 Apps
- Would also enable performance measurement at app level.
    - Decide on SIMD/SVE enablement in apps.
- Enable more optimizations and security hardening of App framework
    - Enable Arch security features (DIT, SSBS)
- Enable NS interrupt pre-emption for applicable use-cases.
- Enable crash recovery of apps
- Live firmware Activation (hitless update) state migration or State re-initialization as needed for the specific EL0 app.





Fake host EL0 app framework



linaro Connect 2025

Thank You!