TF-RMM Live Firmware Activation (design update)

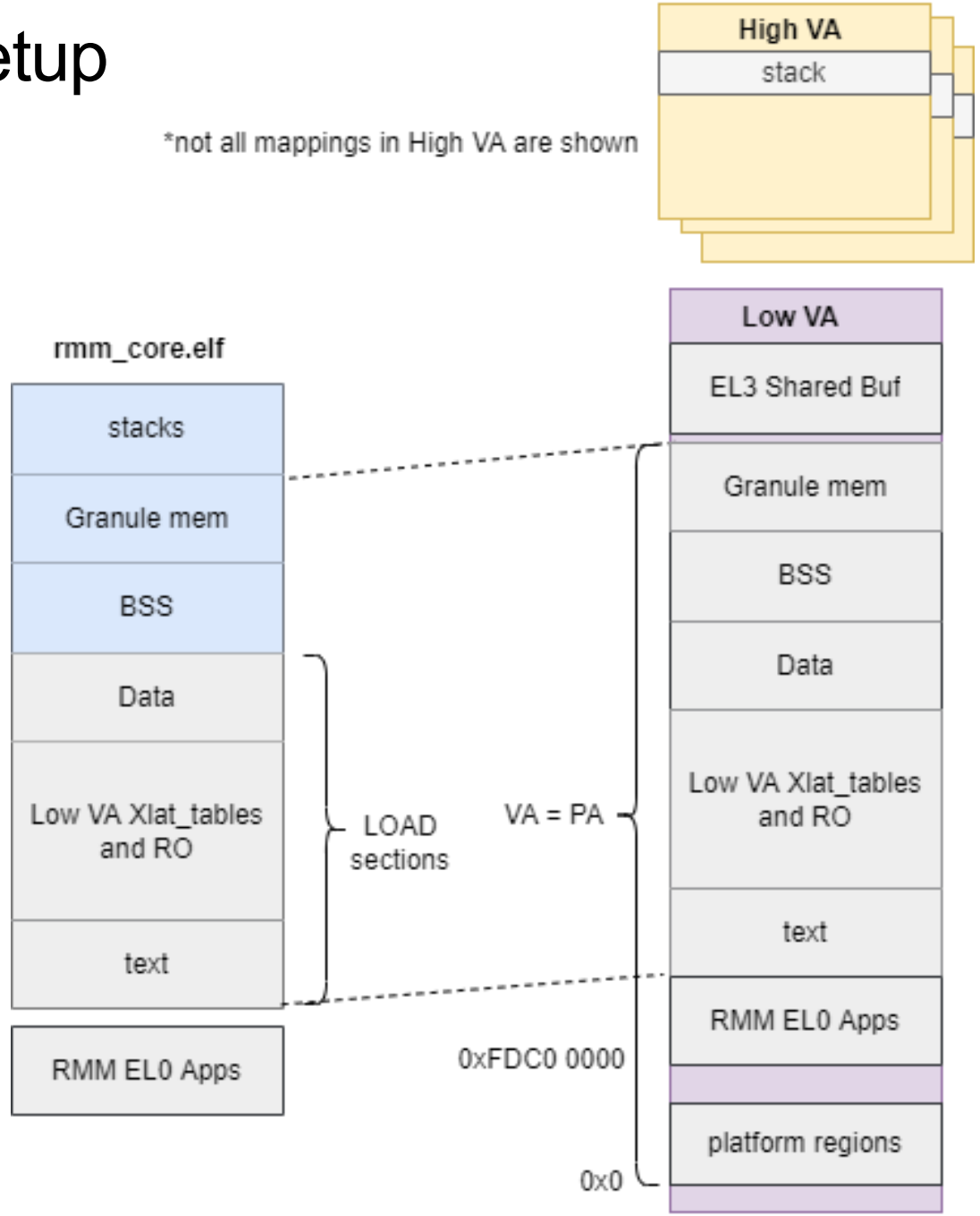
TF-A Tech Forum

Soby Mathew
11-12-2025

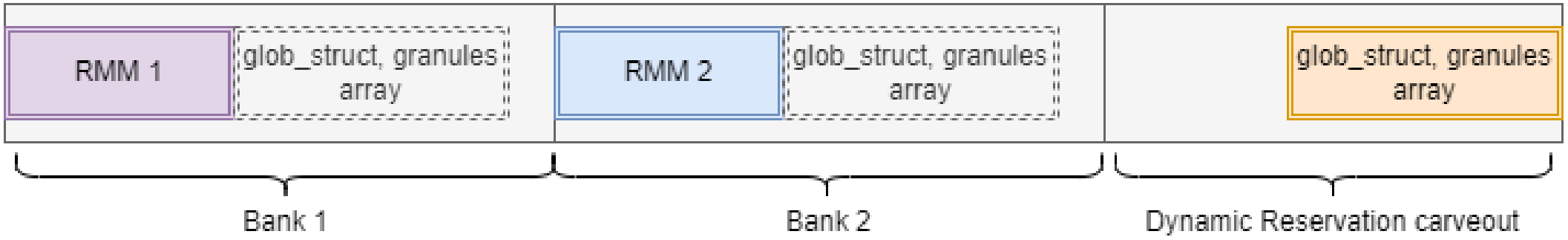
Introduction

- Live Firmware Activation (LFA) is an Arm spec (DEN0147)
  - Defines how updated firmware components are discovered and activated without rebooting the system.
- The exact orchestration (image load, LFA hand-off) is platform/EL3 specific.
- A previous design was presented in this forum on 12/Jun/2025:
  - https://github.com/TF-RMM/tf-rmm/wiki/TFA-Tech-Forum-Presentations
- The design assumes the availability of RESERVE_MEM capability in TF-A and ability to pass “activation token” between RMM images.
- Since that design, the RMM specification has added capability to dynamically map and unmap donated memory for RMM system objects (like struct granule)
  - This means RMM needs to transfer the mappings of the dynamic objects to LFA’d RMM.
- This presentation focuses on further design changes to TF-RMM Stage 1 required to support Live Firmware Activation.
- There are potential improvements that can be done in future which could simplify or remove some of the problems presented today.

RMM Stage 1 MMU setup



Previous LFA design



The dotted box shows the VA mapping of the objects reserved from carveout

Goals of this design

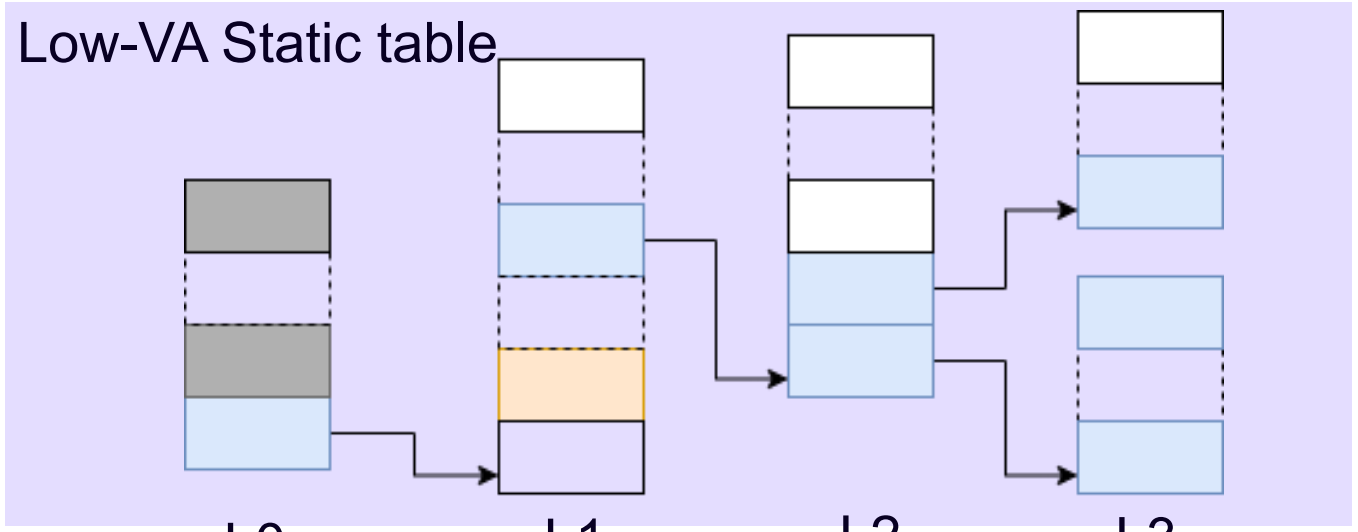
- Make the Low-VA mappings self-contained within `lib/xlat` (rather than platform code)
- Provide runtime mapping and unmapping capability at L3 granularity using a page table structure created at boot
  - Avoid structural modifications at runtime.
- Provide a simple VA allocator to assign virtual addresses for physical pages at runtime.
- Provide a framework for allocating and managing global runtime data.
  - These data structures must be version-controlled, migrate across LFA transitions, and reside outside the RMM image so that the updated RMM instance can reuse them.
- Ensure that allocations requiring EL3 reserved memory are completed at boot to prevent runtime nondeterminism, with a well-defined and deterministic allocation order.
- Split the Low VA into static and dynamic regions
  - Static region can be fully reconstructed in LFA'ed RMM
  - Dynamic VA region (and the VAs) need to be persistent across multiple LFA'ed RMMs.
  - Good to have - Static regions can be kept unmapped as a security hardening
- Current TF-RMM limitation : Identity VA to PA mapping for image sections
  - Good to remove in future.

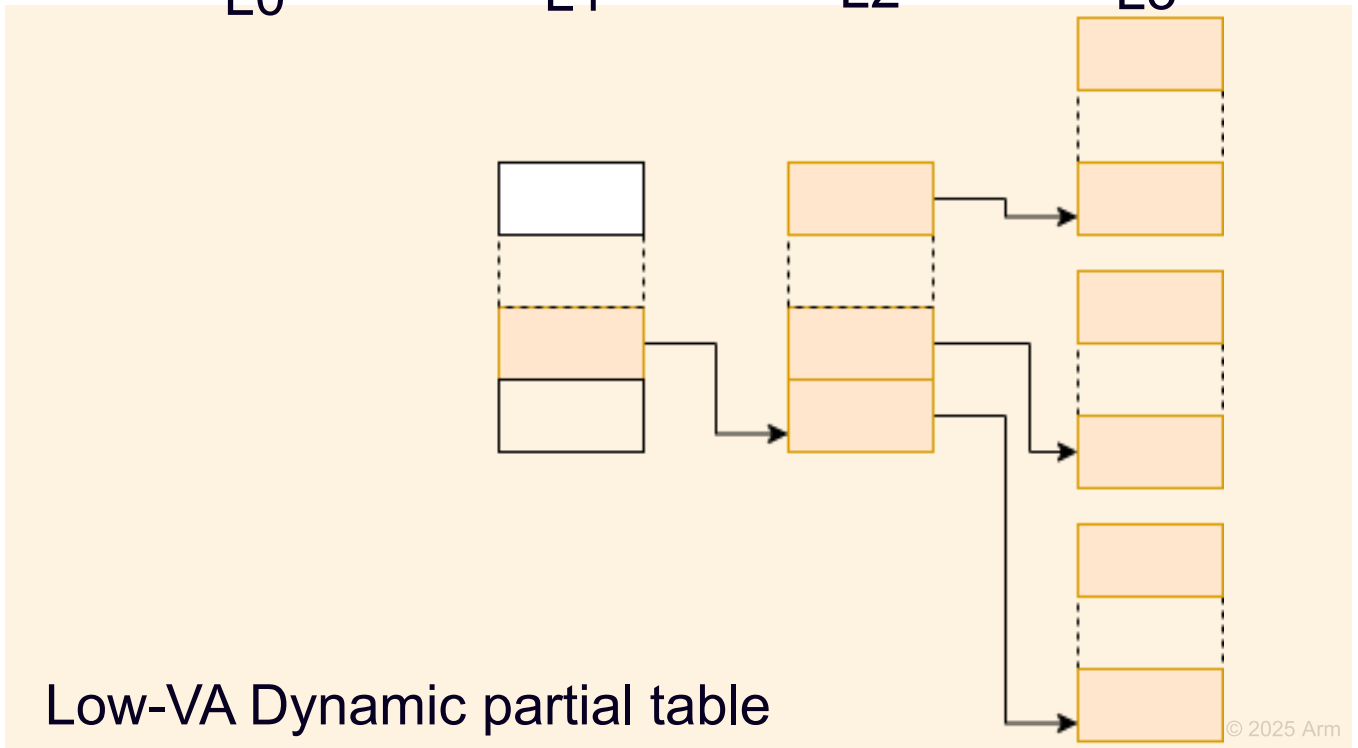
Low VA Dynamic – Static Split

Shared Buffer (RW)	0xFDF0_0000
RMM_RW (RW)	
RMM_RO (RO)	
RMM_CODE (X|R)	
RMM_APP (RO)	0xFDC0_0000

RMM_VA_POOL	1GB 0x4000_0000

RMM Low VA regions





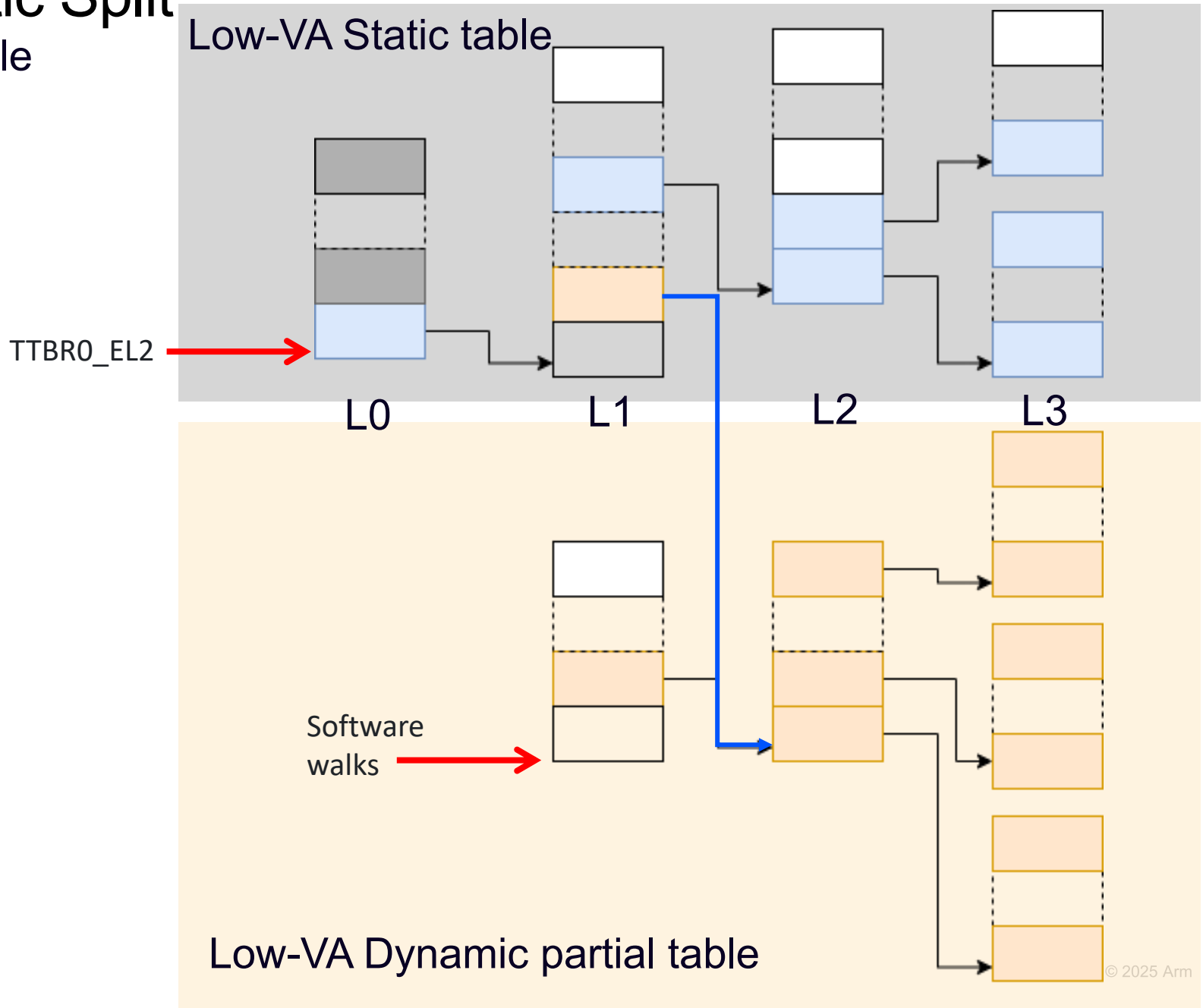
Low VA Dynamic – Static Split

After table stitch and MMU enable

Shared Buffer (RW)	0xFDF0_0000
RMM_RW (RW)	
RMM_RO (RO)	
RMM_CODE (X|R)	
RMM_APP (RO)	0xFDC0_0000

RMM_VA_POOL	1GB 0x4000_0000

RMM Low VA regions



Enhancement to xlat library

- Partial-table support (Low-VA only)
  - The xlat library logic now supports building only a sub-tree (e.g., an L1 tree covering the VA pool), instead of always building the entire Low-VA region.
- Table Stitching Support
  - New API: xlat_stitch_tables_l1(parent, child, base, size)
    - Connects dynamic subtree into static root
    - Must run before MMU enable
- Runtime Mapping API
  - xlat_map_l3_region() and xlat_unmap_l3_region()
    - L3-only modifications. This avoids creation of tables at runtime.
    - Require target descriptors to be TRANSIENT. The Dynamic VA range is initialized as TRANSIENT.
    - Perform local TLBI for modified VA range
    - No table allocations at runtime.
- Walking Tables when MMU is Enabled
  - Dynamic tables are remapped inside the dynamic VA pool. The walker now requires PA↔VA delta .
    - This feature can be removed if RMM maps the tables on a need basis (similar scheme to S2 walk).
- Table Count Estimation
  - The estimator computes required L2/L3 tables based on region layout.

Dynamic VA allocator

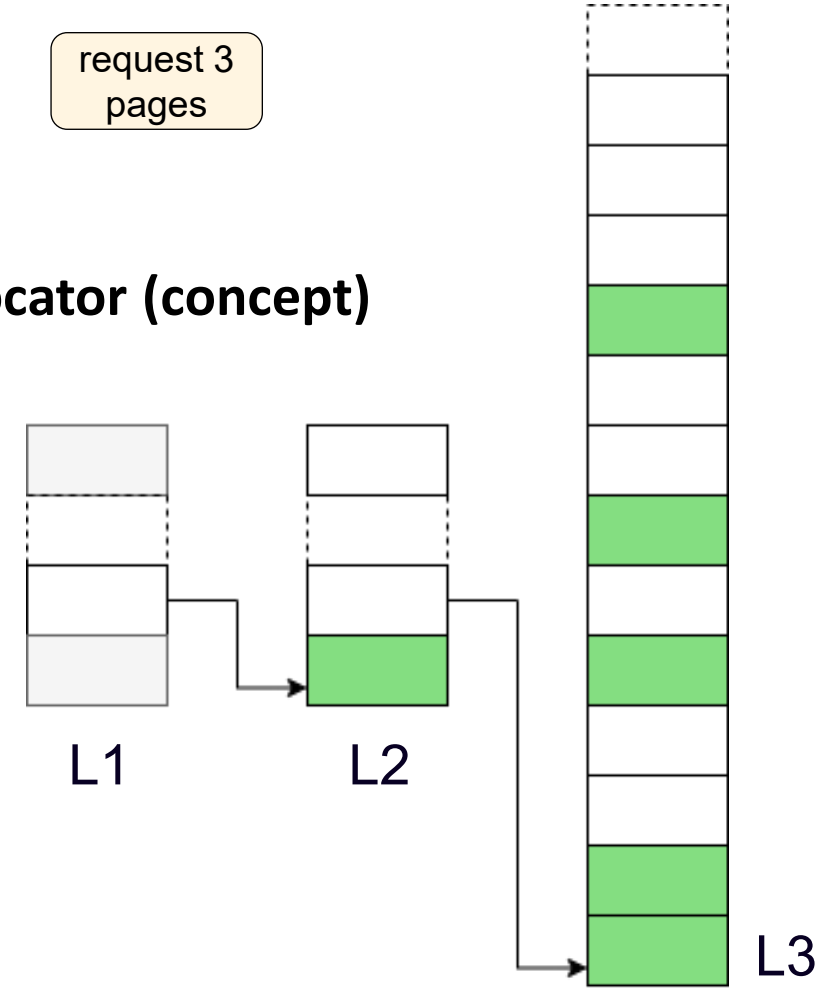
- **Purpose:** Allocate and free virtual addresses (VA) within the dynamic VA pool.
  - Note that this is only dishing out VA. Any VA allocated must be mapped to the respective PA in the Dynamic Tables before it can be accessed.
- How big should be VA pool be ?
  - Tracking **256 TB DDR** (48-bit address) → ~128 GB for `struct granule`.
  - Current assumption: Dynamic VA range will fit in a **Level 1 table** (assuming 4KB page)
  - Stitch logic can move to **L0 table** if required.
- **Design Considerations:**
  - Deterministic behavior.
  - Memory footprint of allocator structures should be independent of NS allocation strategy.
  - Data structure should be resilient even with fragmentation – fragmentation cannot be avoided.
- A **bitmap** based allocator seems to fit the bill.
  - 1 bit per 4 KB page. Need 8 Pages to track 1 GB of VA pool memory
- **Future Improvement:** Use Stage 1 software bits for tracking.
  - Eliminates extra memory for tracking.
  - Prototype planned later.

Bitmap-based VA Allocator (concept)

Legend
■ allocated (bit=1)
□ free (bit=0)

request 3 pages

PTE software bit-based VA allocator (concept)



`glob_data` struct

- Centralizes global runtime state that must survive LFA updates.
  - Owns the global data like Low-VA configuration, VA allocator config and its bitmap
  - Stores the info about backing store (eg: granules[], dev_granules[]) so that it can be transferred to the next RMM instance.
    - Physical, virtual address and size triples (PA/VA/size) are kept track
- The required EL3 memory reservation requests are done at init time
  - The xlat tables are reserved by the xlat library
- The address of this struct is passed as “activation_token” back to EL3
  - EL3 will send this token to the LFA RMM during boot.
- There are helper APIs for other components in RMM to discover their memory reservations at runtime.

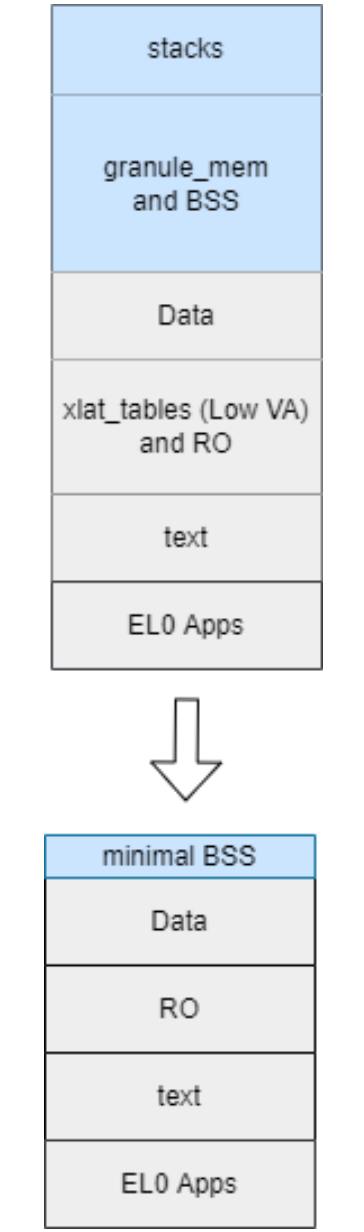
```
struct glob_data {
»       unsigned long version;

»       struct xlat_low_va_info low_va_info;

»       uintptr_t bitmap_pa;
»       uintptr_t bitmap_va;
»       size_t bitmap_size;

»       uintptr_t glob_data_pa;
»       uintptr_t glob_data_va;
»       size_t glob_data_size;

»       va_allocator_t va_alloc_info;

»       /* Memory for struct granule array */
»       uintptr_t granules_pa;
»       uintptr_t granules_va;
»       size_t granules_size;

»       /* Memory for struct dev_granule array */
»       uintptr_t dev_granules_pa;
»       uintptr_t dev_granules_va;
»       size_t dev_granules_size;
};
```

Compatibility mode & Platform considerations

- Some platforms may run EL3 firmware that does not yet implement SMC_RMM_RESERVE_MEMORY.
- To keep RMM functional on those systems, RMM includes a compile-time fallback
  - If RMM_EL3_COMPAT_RESERVE_MEM is defined, `rmm_el3_ifc_reserve_memory()` calls `compat_reserve_memory()` rather than making the SMC.
- **Impact:** Although RMM would still be functional, LFA will not
  - The memory reservations would be from the current RMM's BSS section.
- Deprecated platform config options :
  - RMM_MAX_GRANULES, RMM_MAX_COH_GRANULES, RMM_MAX_NCOH_GRANULES, PLAT_CMN_CTX_MAX_XLAT_TABLES
  - These controlled various object allocations in RMM, which are now no longer needed.
- New config option:
  - RMM_VA_POOL_SIZE to define the size of Dynamic VA range.
  - This can also be made dynamic and passed via Boot manifest.

Enhancements and Hardening

- Track VA allocations using Software bits in Xlat descriptor
  - Avoids need for separate bitmap memory
- Move per-cpu data (eg: stacks, internal objects) to EL3 reserved memory
  - Removes dependancy on MAX_CPUs
  - Enables reuse between different LFA instances.
  - Ties into NUMA strategy for local node allocation of per-cpu data.
- Dynamic table mapping on demand
  - Map Dynamic tables only when modification is needed; keep them unmapped otherwise.
  - Aligns with Stage 2 TT scheme in RMM.
- Security hardening for the VA allocations
  - Explore guard pages between allocations, Memory Tagging Extension (MTE), and other architectural features.
- Investigate removing identity mapping restriction for Stage 1
  - Enables advanced security measures like address space randomization.
- Increase unit testing and LFA related hardening.
  - LFA will require ongoing improvements and validation.



Vision for RMM binary

Conclusion

Live Firmware Activation (LFA) enables hitless firmware updates and improves uptime

Dynamic VA management and partial table stitching introduced

Security and determinism remain key priorities

Next steps are validation, integration, and optimization

A robust, scalable, and secure architecture for LFA in RMM

* Image created by copilot

arm

Merci
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Thank You
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה
ధన్యవాదములు
Köszönöm