Granule Protection Tables in TF-A

John Powell
October 21, 2021

What are Granule Protection Tables?

- ARMv9 introduces two additional security states for a total of four: root and realm, in addition to secure and non-secure. (FEAT_RME)
- These additional security states and their intended use cases require a new way to control memory access.
- Granule protection tables define the ranges of physical memory that each security state can access.

	Security State			
	Root	Realm	Secure	Non-secure
GPI_ROOT	yes	no	no	no
GPI_REALM	yes	yes	no	no
GPI_SECURE	yes	no	yes	no
GPI_NS	yes	no	no	yes
GPI_ANY	yes	yes	yes	yes
GPI_NO_ACCESS	no	no	no	no

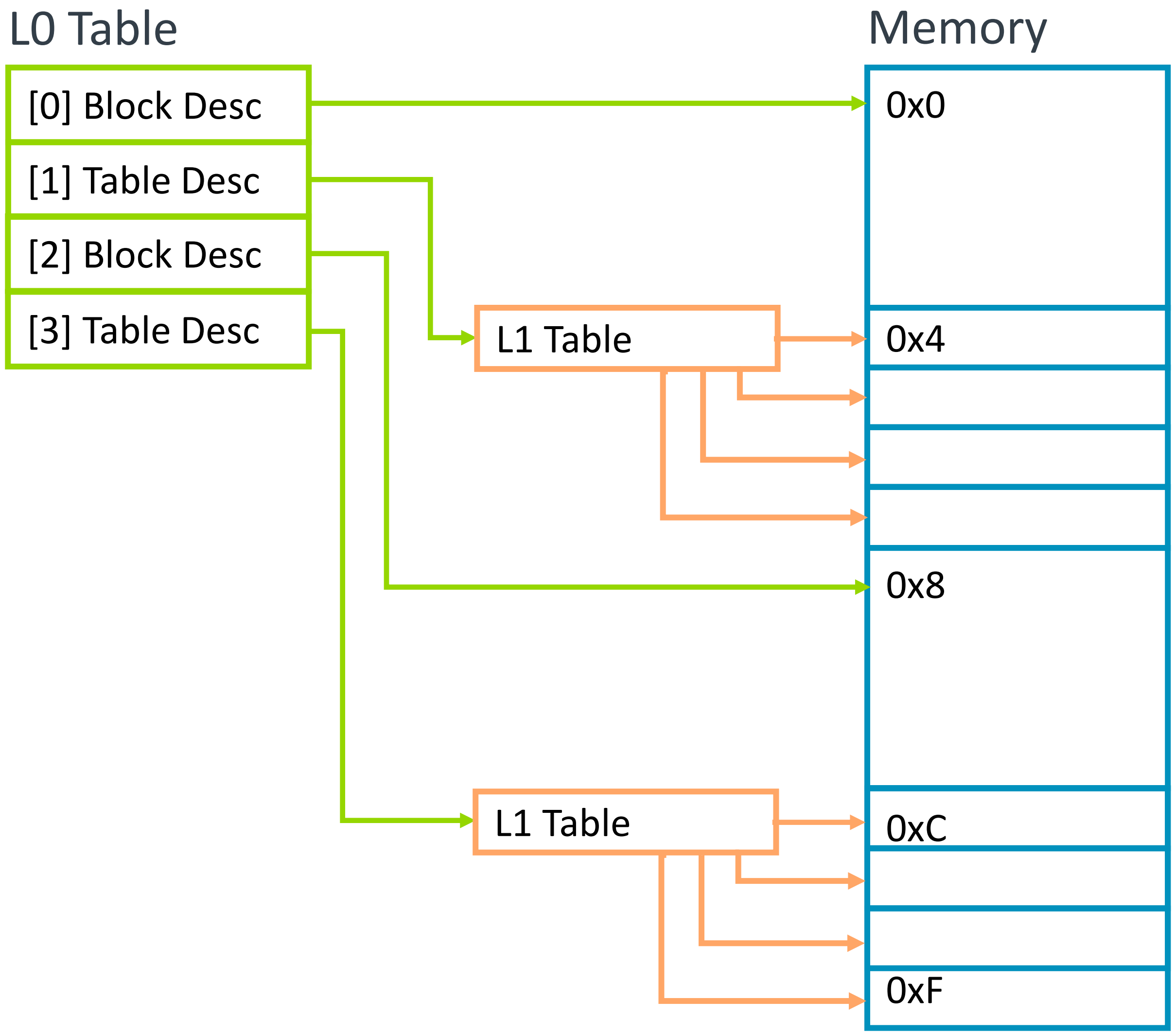
What are Granule Protection Tables?

- Memory regions are tagged with one of six GPI values. (granule protection information)
- The table to the right shows the four security states and what GPIs they have access to.
- Granule protection checks trigger an exception when a program attempts to access memory outside of what is permitted by its security state.

	Security State			
	Root	Realm	Secure	Non-secure
GPI_ROOT	yes	no	no	no
GPI_REALM	yes	yes	no	no
GPI_SECURE	yes	no	yes	no
GPI_NS	yes	no	no	yes
GPI_ANY	yes	yes	yes	yes
GPI_NO_ACCESS	no	no	no	no

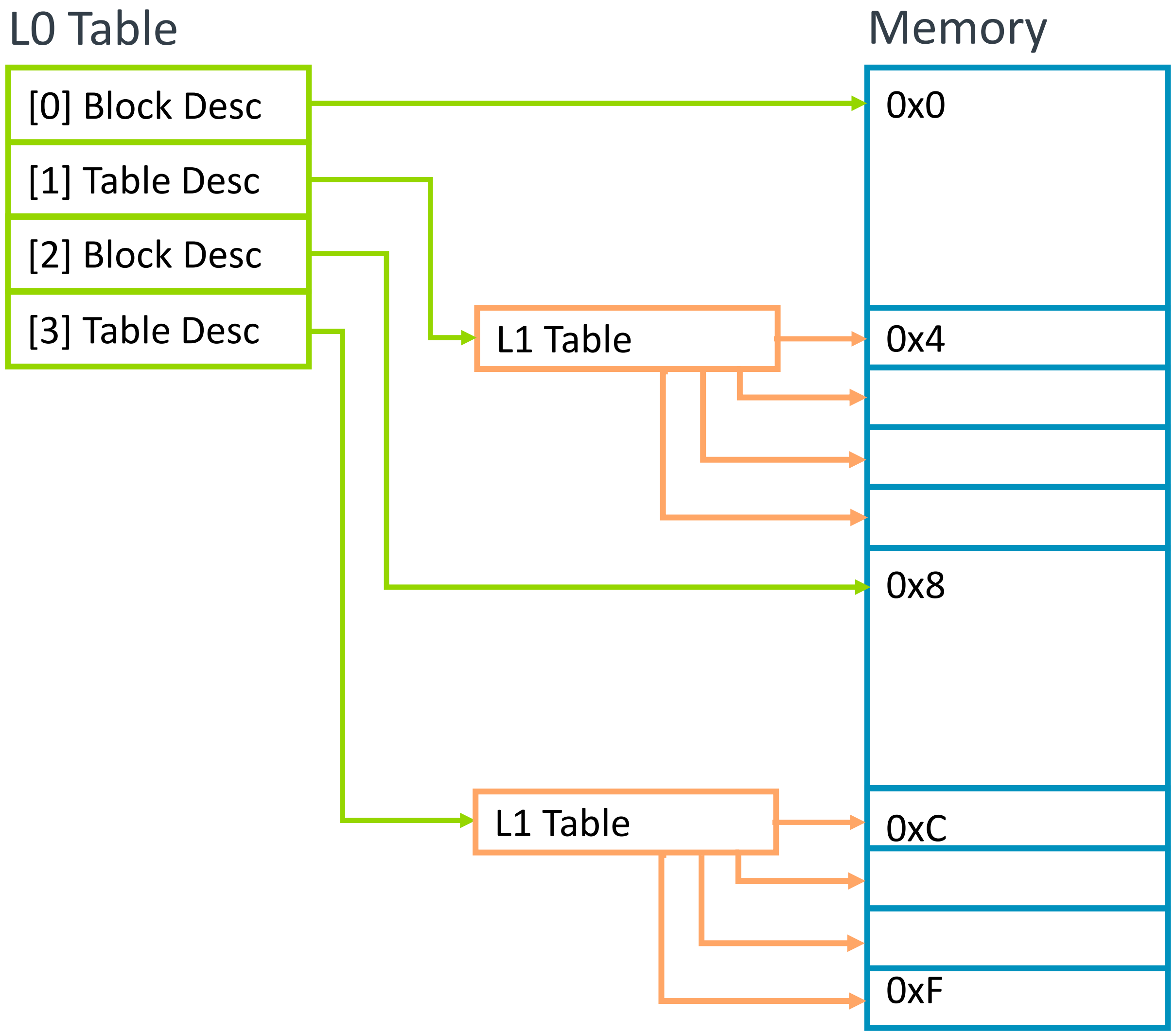
How do Granule Protection Tables work?

- Granule protection tables can use either a one or two stage lookup process using level 0 and level 1 tables.
- Each L0 table entry controls a large, fixed amount of memory.
- An L0 entry can either map its entire space with a single GPI (block descriptor) or point to an L1 table controlling individual granules (table descriptor).



How do Granule Protection Tables work?

- Block descriptors use a single stage lookup using only the L0 table entry.
  - GPI is fixed and cannot be changed after initialization.
- Table descriptors use a two-stage lookup using both the L0 table and an L1 table.
  - GPI can be changed at runtime using SMC calls.
- Granules are relatively small and allow for much finer control of memory.

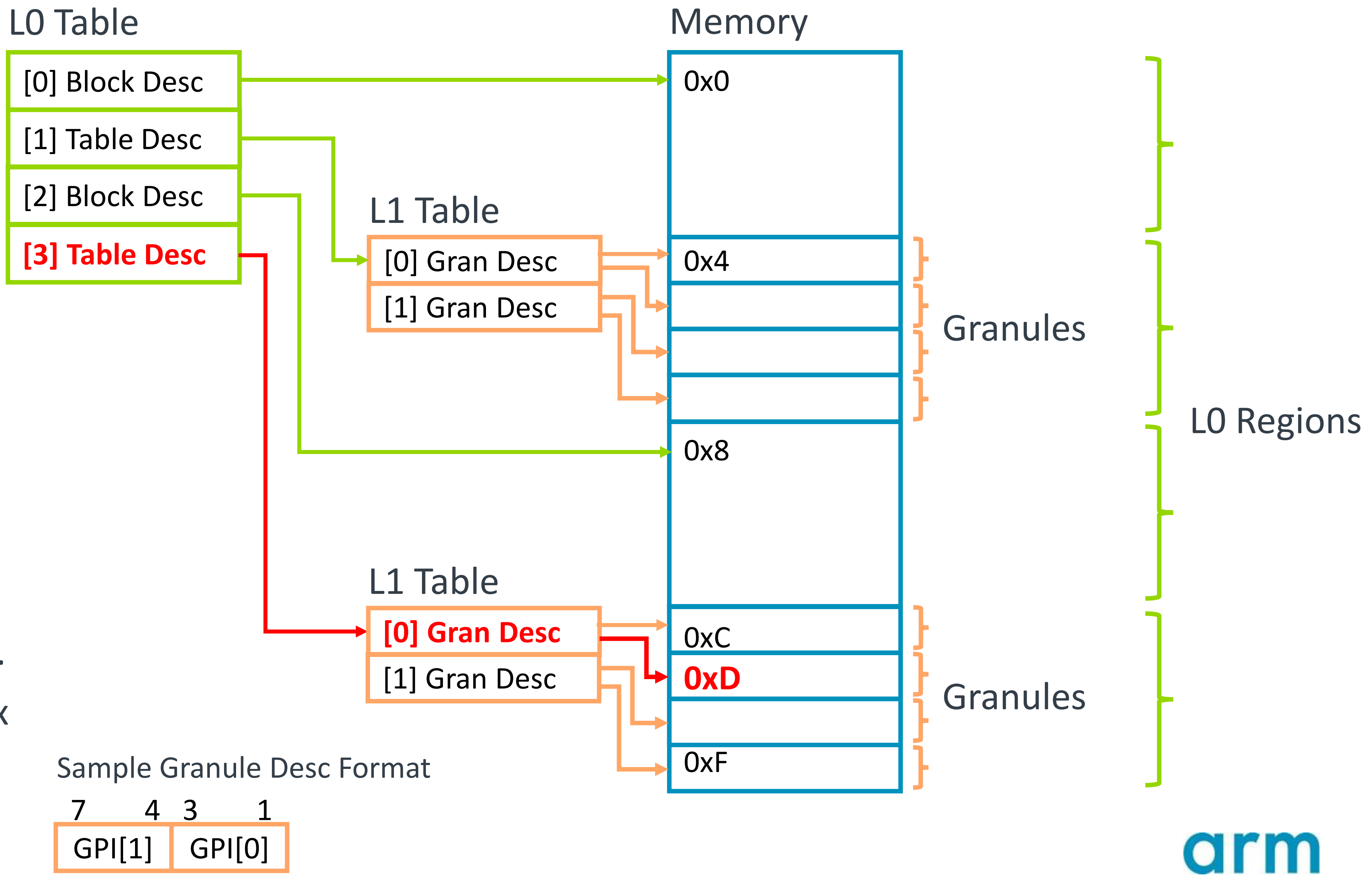


Granule Protection Table Configuration

- Three main parameters define how the tables and regions are organized.
- Protected Physical Address Size (PPS)
  - This parameter defines the size of the protected address space starting from 0x0.
  - Supported sizes are 4GB, 64GB, 1TB, 4TB, 16TB, 256TB, and 4PB.
- Physical Granule Size (PGS)
  - This defines the size of each granule.
  - Supported sizes are 4KB, 16KB, and 64KB.
- Level 0 GPT Size (L0GPTSZ)
  - This parameter determines how large each level 0 region is. This value is determined by hardware and is read from GPCCR_EL3 during table initialization.
  - Supported sizes are 1GB, 16GB, 64GB, and 512GB.

A Simple Example

- PPS = 16 bytes
- PGS = 1 byte
- L0GPTSZ = 4 bytes
- Let's access PA 0xD (0b1101)
- L0 table is indexed using bits [3,2] of the physical address, so index = 0b11.
- L0[3] is a table descriptor, so get the address of the L1 table from it then use bit[1] of the PA to get the index of the L1 descriptor.
- Use bit[0] to get the index of the GPI within the descriptor.

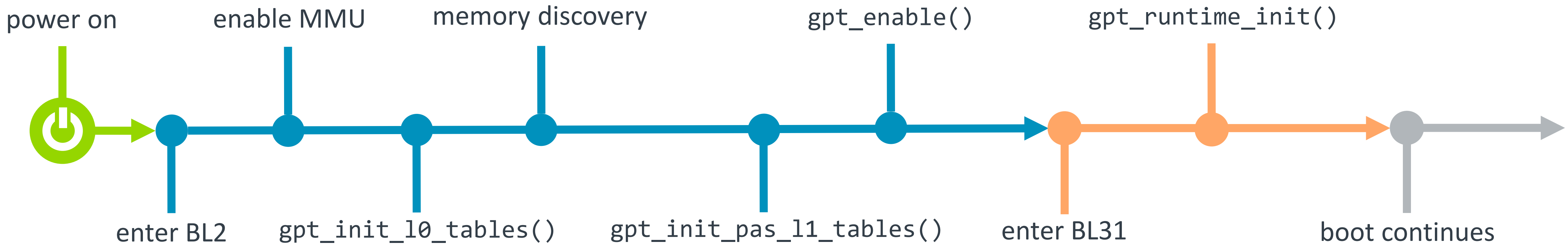


GPT Initialization in TF-A

- The MMU is enabled first, this simplifies cache management.
- All L0 entries are initialized as block descriptors allowing ANY access.
  - The L0 table is placed in SRAM to provide the best security.
  - `gpt_init_l0_tables()` is called in BL2 prior to system memory discovery.
  - PPS is set here, along with the level 0 table base address.
- Protected regions are then “carved out” of this space.
  - L1 tables are typically placed in DDR in a region with GPI_ROOT.
  - `gpt_init_pas_l1_tables()` is called in BL2 after system memory is discovered.
  - These regions can be either block or table (granule) descriptors.
  - PGS is set here, along with setting the base address for L1 tables.
  - This function can be called multiple times if placing the level 1 tables in different locations is desirable, such as separate banks of DDR having their own L1 tables.

GPT Initialization in TF-A

- Once the tables have been created, granule protection checks are enabled.
  - `gpt_enable()` is the final step of BL2 GPT initialization.
- Runtime firmware discovers the tables using register values programmed during initialization so the granule transition service knows where to look.
  - `gpt_runtime_init()` is called in BL31.
  - Level 0 tables are located along with the LOGPTSZ, PPS, and PGS parameters.
- For warm boots, BL31 simply calls `gpt_enable()` after enabling the MMU.



Granule Transition Service

- Realm and secure software can request that granules be transitioned between security states using SMC calls.
  - Secure software can request NS -> S transitions, and S -> NS transitions.
  - Realm software can request NS -> R, and R->NS transitions.
- When a transition request is received, runtime firmware walks the tables to find the requested granule, validates the request, then performs the transition.
- If the granule transition service is not needed, runtime firmware does not need to discover the tables
- Non-secure and root firmware cannot request granule transitions.

Future Enhancements

- Allow a range of granule-aligned memory to be transitioned at once instead of just single granules.
- The granule transition service currently relies on a single global lock to control access to the L1 table, performance could be improved by having multiple locks across separate L1 tables or even L1 descriptors.
- Add support for contiguous descriptors.

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