arm

Arm CCA – KVM Support

KVM Forum 2023

Suzuki K Poulose
14th June 2023

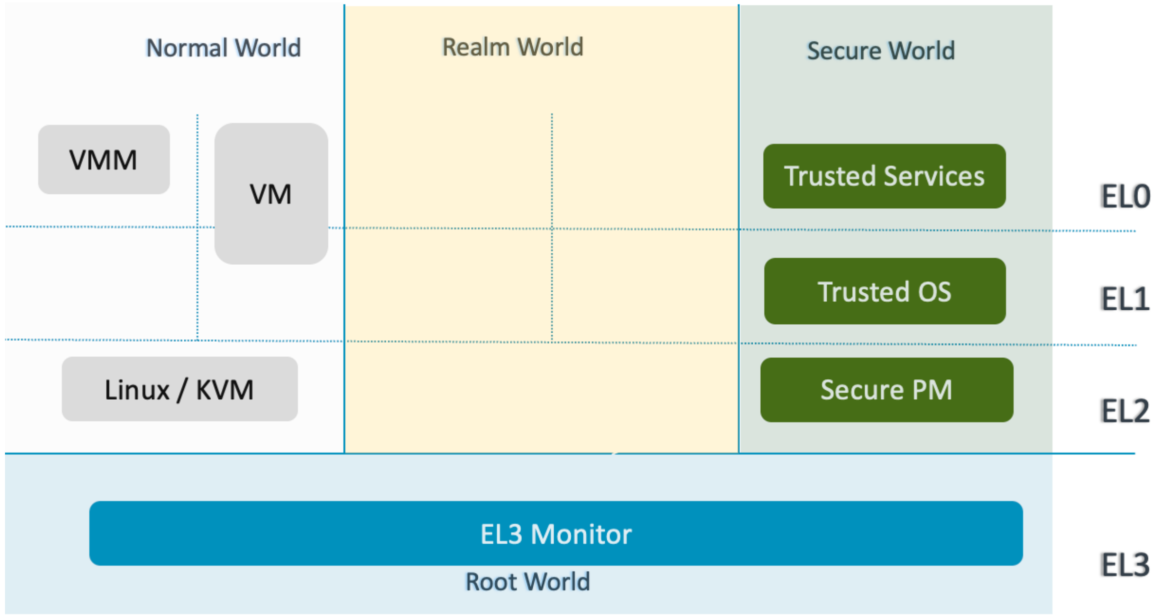
Agenda

- Introduction to Arm CCA
  - Arm v9 - Hardware Architecture - FEAT_RME
  - Arm CCA Software Architecture
    - RMM Host services
    - RMM Guest Services
    - Realm VM
    - Realm Life Cycle
    - REC Scheduling
- KVM Support
- Current Status
- Arm CCA – Future

arm

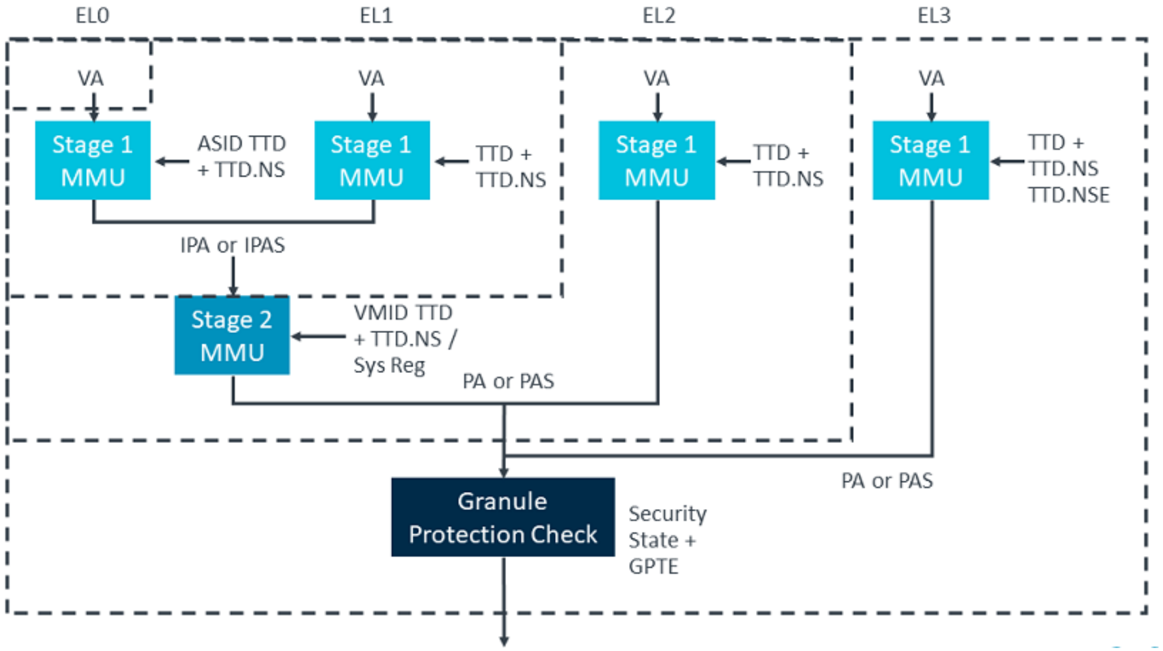
Arm v9 Realm Management Extensions – FEAT_RME

- Traditionally Arm has two security states
  - Secure and Non-Secure
- 4 Exception Levels (Privilege levels)
  - EL3 highest, EL0 lowest
- Introduces two new security state (Physical Address Space - PAS)
  - Secure – EL0, EL1, EL2, ~~EL3~~
  - Non-Secure – EL0, EL1, EL2
  - Root – Only EL3
  - Realm – EL0, EL1, EL2

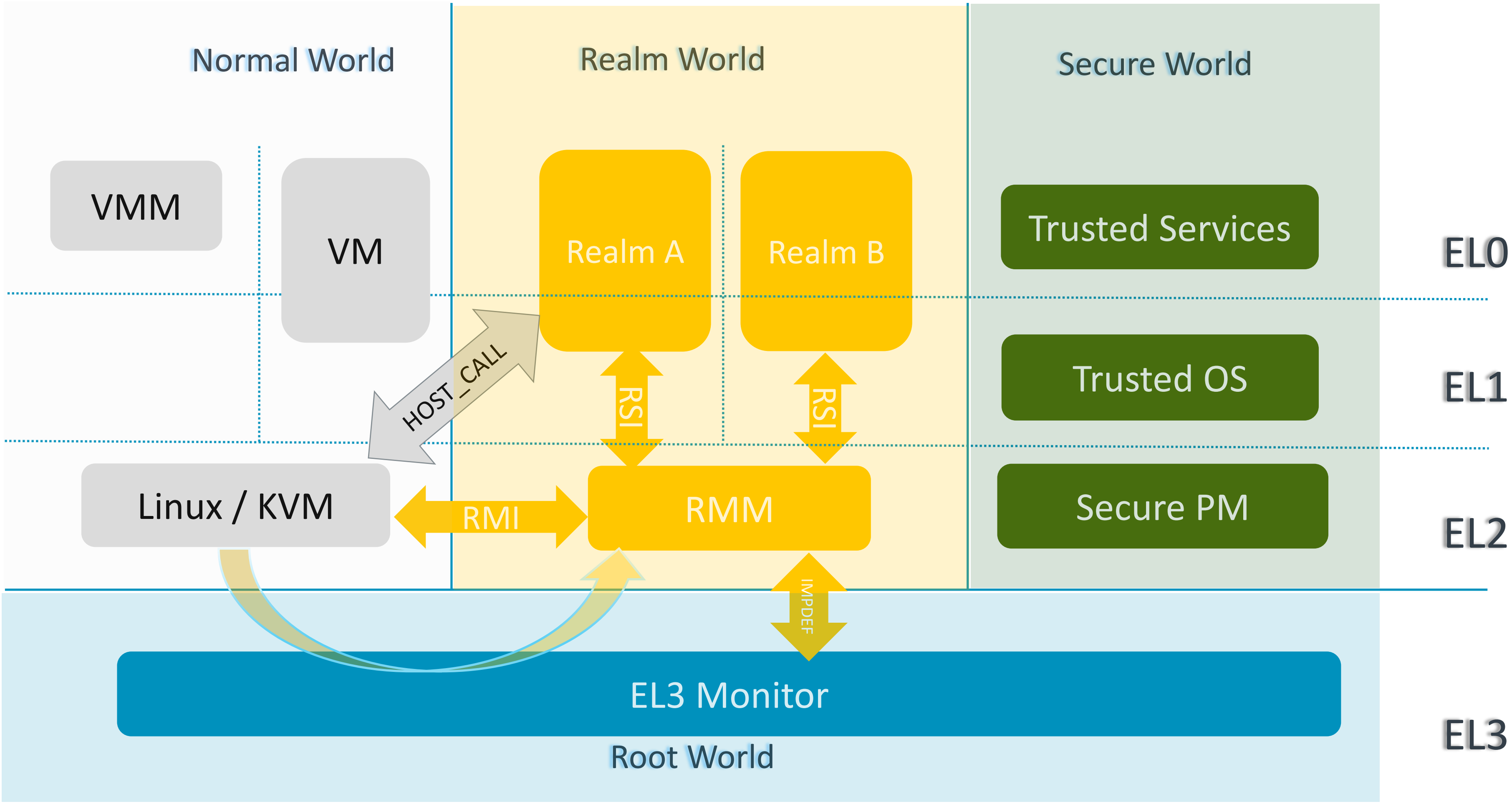


Arm v9 Realm Management Extensions – FEAT_RME

- Dynamic PAS check via Granule Protection Checks (GPC)
  - Stage 3, described by Granule Protection Table
  - GPT Keeps track of the PAS of each 4K Granule
  - Maintained by EL3 firmware, Root World
  - Access violations result in Granule Protection Fault
- EL3 can change the PAS of a granule by updating GPT



Arm CCA Software Architecture



Arm CCA - Software Architecture

- Reference Software Architecture using Arm v9 FEAT_RME
- Enables Confidential Computing VMs on Arm
  - Running VMs in the Realm World (R-EL1, R-EL0)
  - Removes access to the VM private data / state
  - NS-Host retains management of the VM
  - Protected from the Normal and Secure world
- Introduces a new firmware at Realm EL2 – Realm Management Monitor
  - Part of Confidential VM's TCB
  - Architected software component, RMM Specification v1.0
  - Developed in collaboration with Arm partners
  - Reference implementation from TrustedFirmware.org – TF-RMM
  - Loaded by at boot time by EL3 firmware
  - Part of the Platform Attestation report
- RMM Specification defines
  - Services to the Host for managing VMs aka Realms via RMI
  - Services to the Realm including Realm Service Interface (RSI)

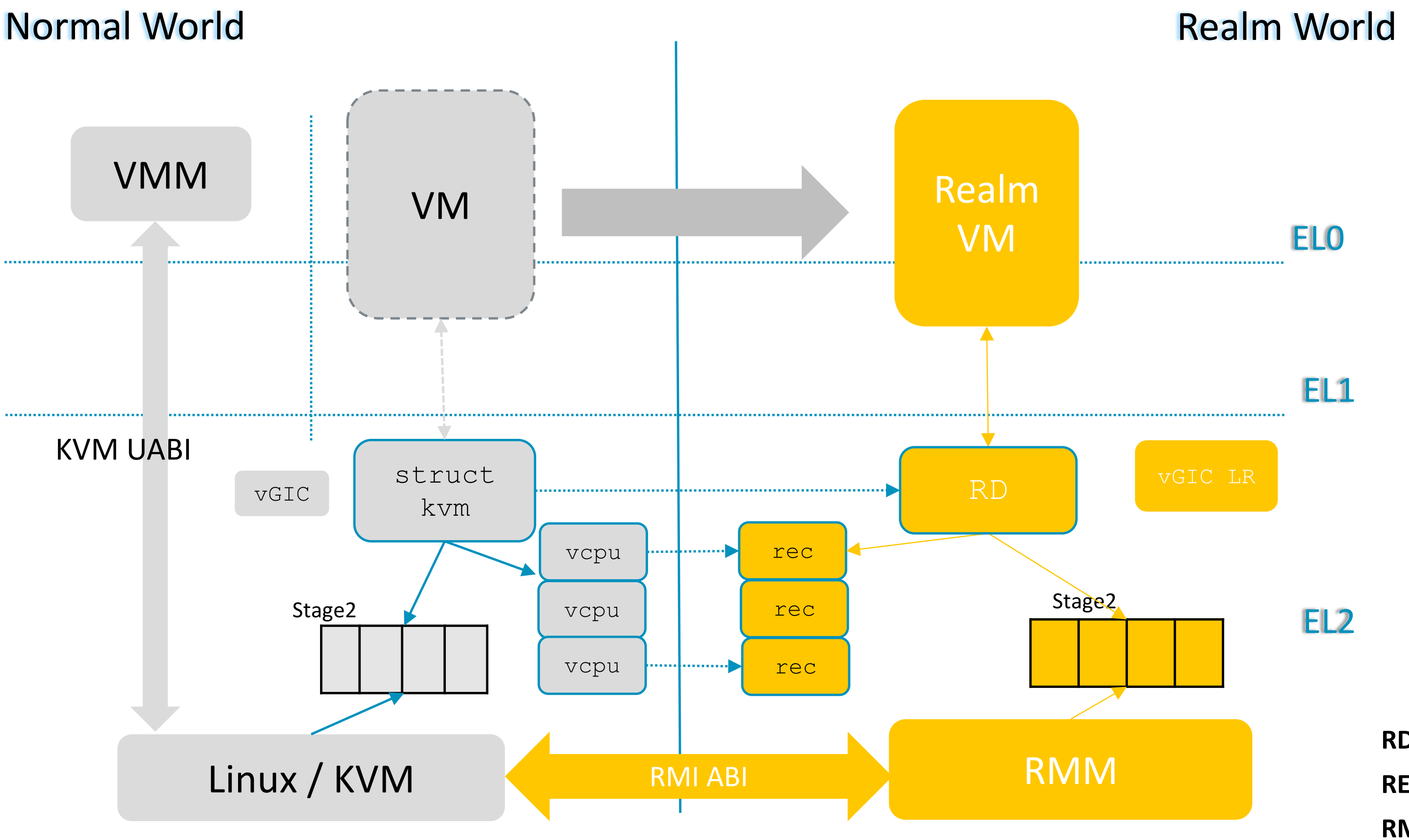
Arm CCA – Host Services - Realm Management Interface

- RMI Version
  - Major version and minor version (v1.0)
- RMM Feature discovery
  - SVE, PMU, LPA2, IPA Size etc.
- Move a Granule (4KB) between Non-Secure ⇔ Realm
- Life Cycle management of Realm VM
- Manage Realm Execution Context(REC) aka Virtual CPUs
  - Create with initial register state (measured)
  - Schedule RECs and handle exits
  - Inject virtual interrupts
- Manage memory for the Realms
  - Add / Remove memory
  - Manage Stage2 page table - monitored
    - Service stage2 faults

Arm CCA – Services to the Realm

- Ensure correctness of the Host actions
  - The IPA Space of the Realm is split to half
    - Protected (lower) IPA with RMM guarantees
    - Unprotected IPA (higher) with no security guarantees
  - Monitor RMI operations and provide security guarantees for Protected IPA
- Realm specific services via Realm Service Interface (RSI)
  - Query Realm configuration
  - Manage Realm IPA State(RIPAS) – see later
  - Communicate with the Host – RSI_HOST_CALL. (SMCCC compliant HVC)
- Attestation and Measurement
  - Platform Measurements – HW / Firmware including RMM
  - Realm Initial measurements – Host actions before activate – Data, CPUs etc
  - Realm Extendable Measurements
- Isolation from other Realms

Arm CCA – Realm VM (KVM view)



RD – Realm Descriptor

REC- Realm Execution Context

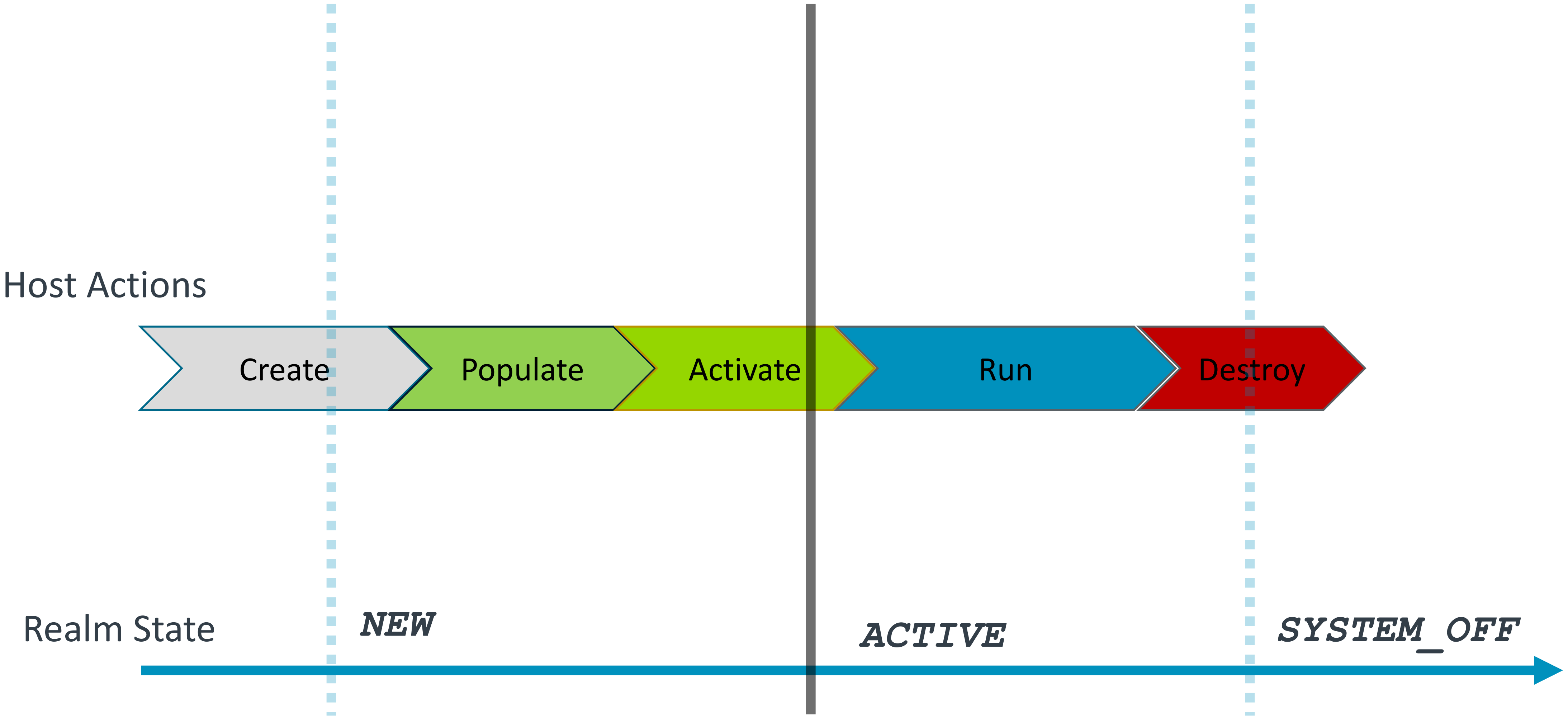
RMI – Realm Management Interface

Arm CCA – Realm VM

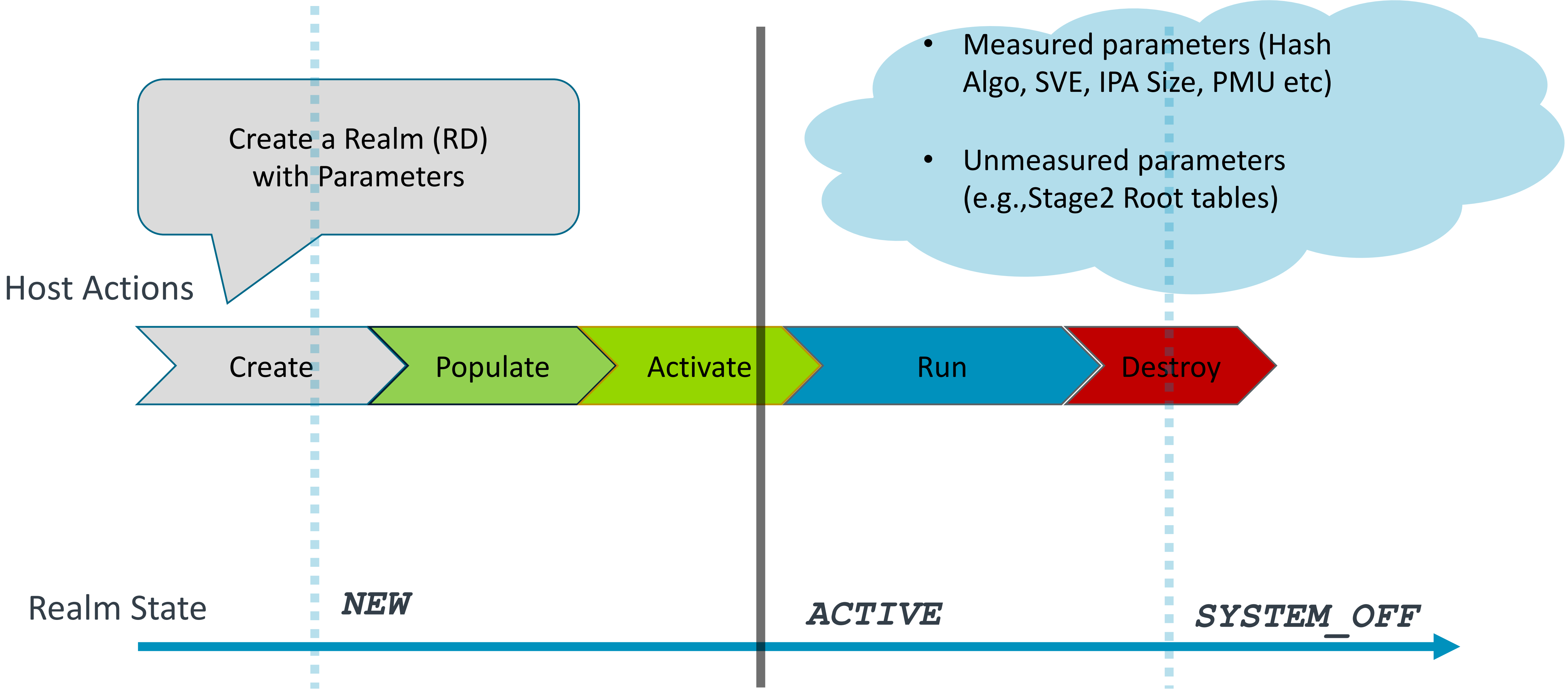
- RMM manages Realm via Objects in Realm PAS
  - Host donates memory via ***RMI_GRANULE_DELEGATE***
- VM described by Realm Descriptor (***RD***) - `struct kvm` equivalent
  - Created via *`RMI_REALM_CREATE(RmiRealmParameters)`*
    - Choose Hash Algorithm, IPA Size, SVE?, PMU? etc
    - Root Stage2 page table pages
  - Holds Realm Initial Measurements (RIM)
- VCPUs are described by Realm Execution Context (REC) objects
  - Created via *`RMI_REC_CREATE(RD, REC_Granule, REC_Params)`*
  - Saves vCPU context, Previous exit reason (Host Calls, MMIO etc), Outstanding requests
  - Variable storage via *`REC_Params.aux`*, depending on features (e.g., SVE_VL)
- Stage2 Page table pages - Realm Translation Table, RTT
  - Created via *`RMI_RTT_CREATE`*
  - Reference counted for each protected mapping
  - Holds additional metadata (when Inactive) – Host state (HIPAS) and Realm IPA State (RIPAS)
  - Host can read an RTT entry using *`RMI_RTT_READ_ENTRY`*

Arm CCA – Realm Life Cycle

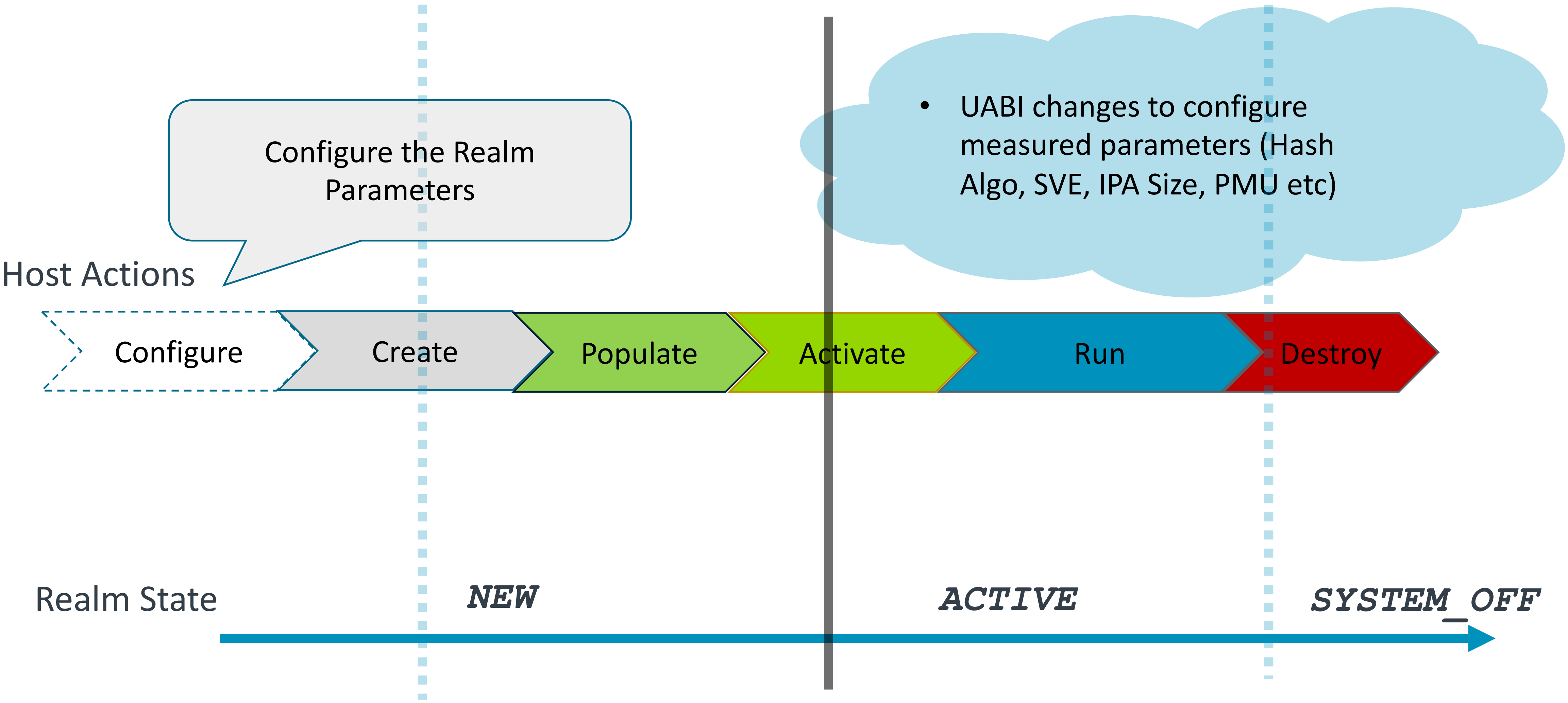
Realm Life Cycle



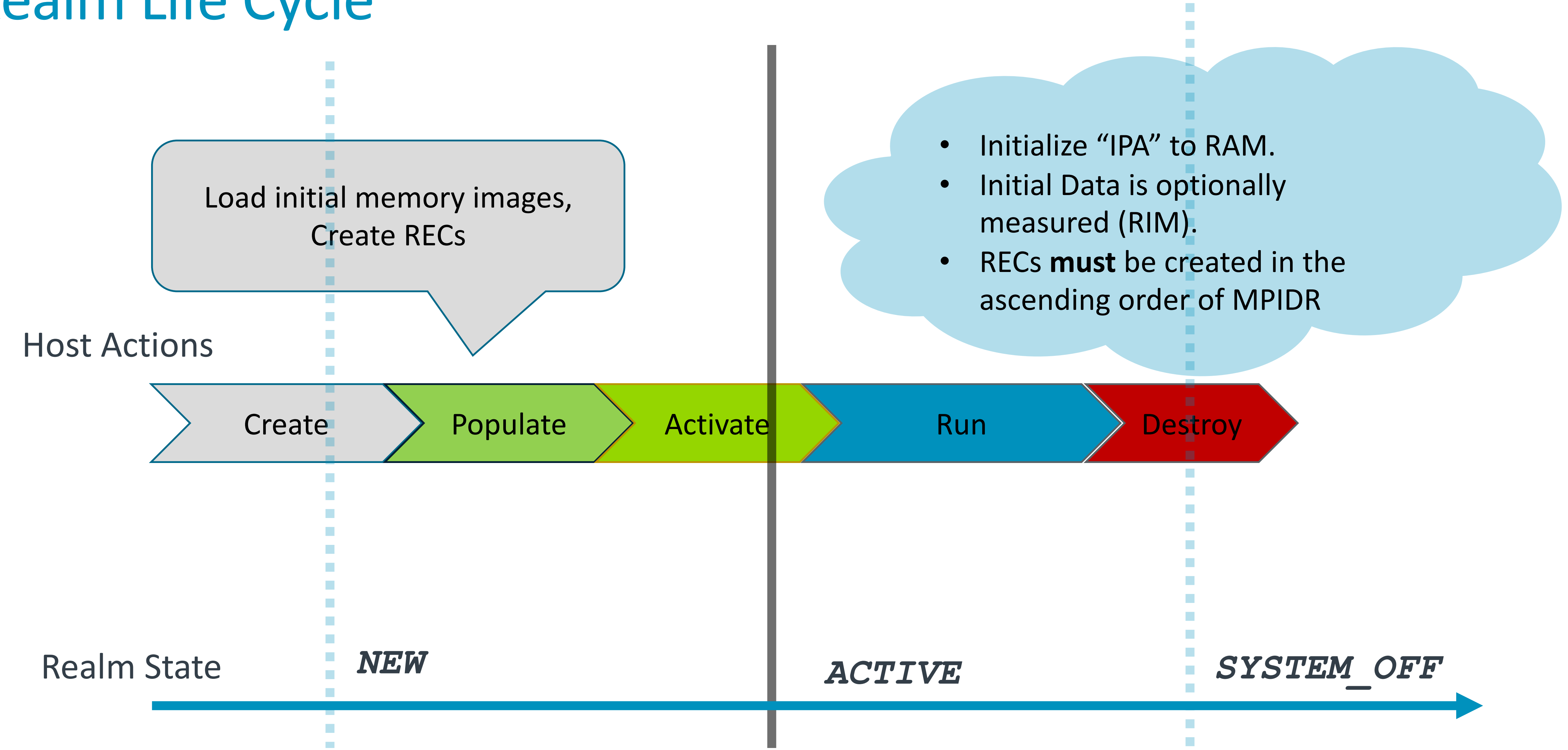
Realm Life Cycle



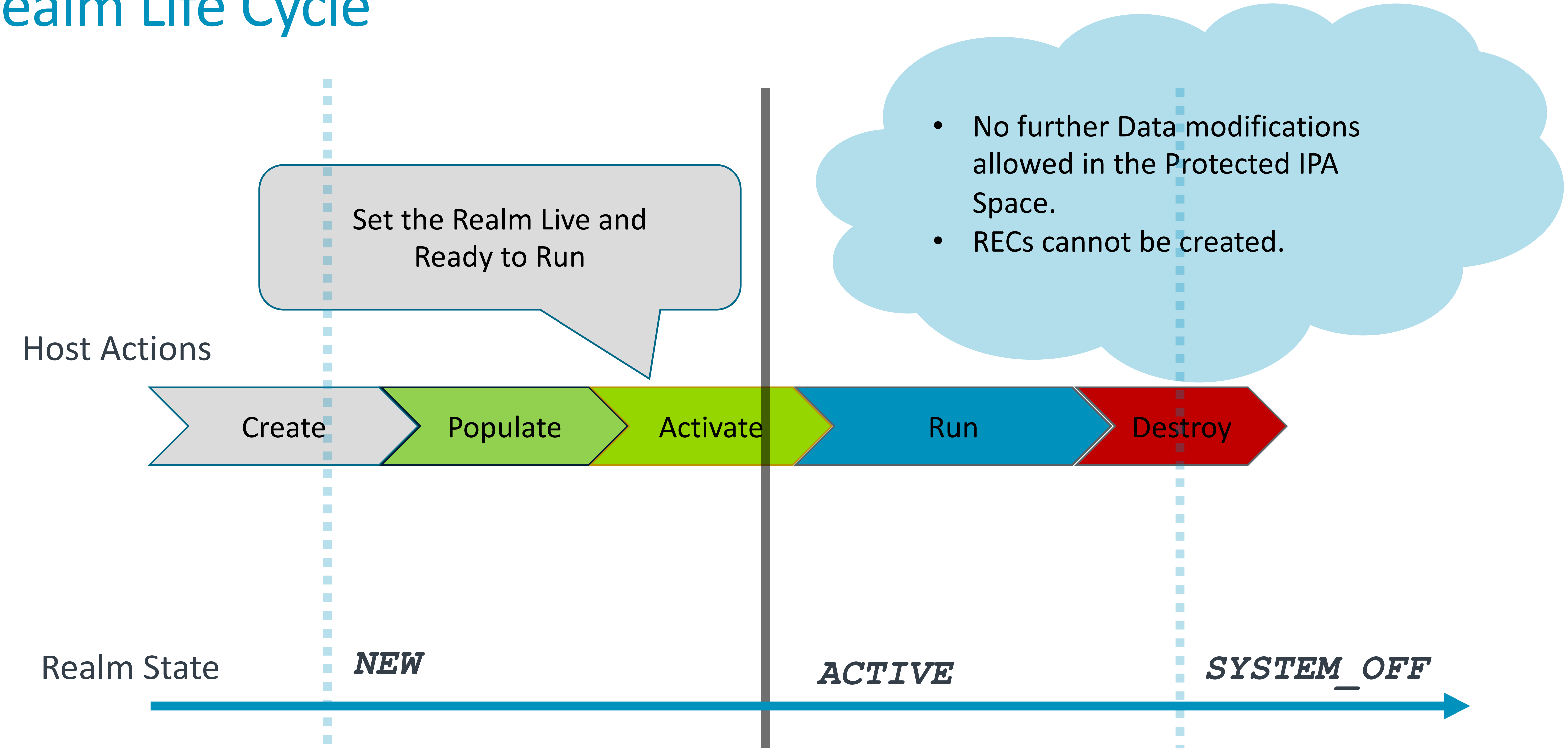
Realm Life Cycle



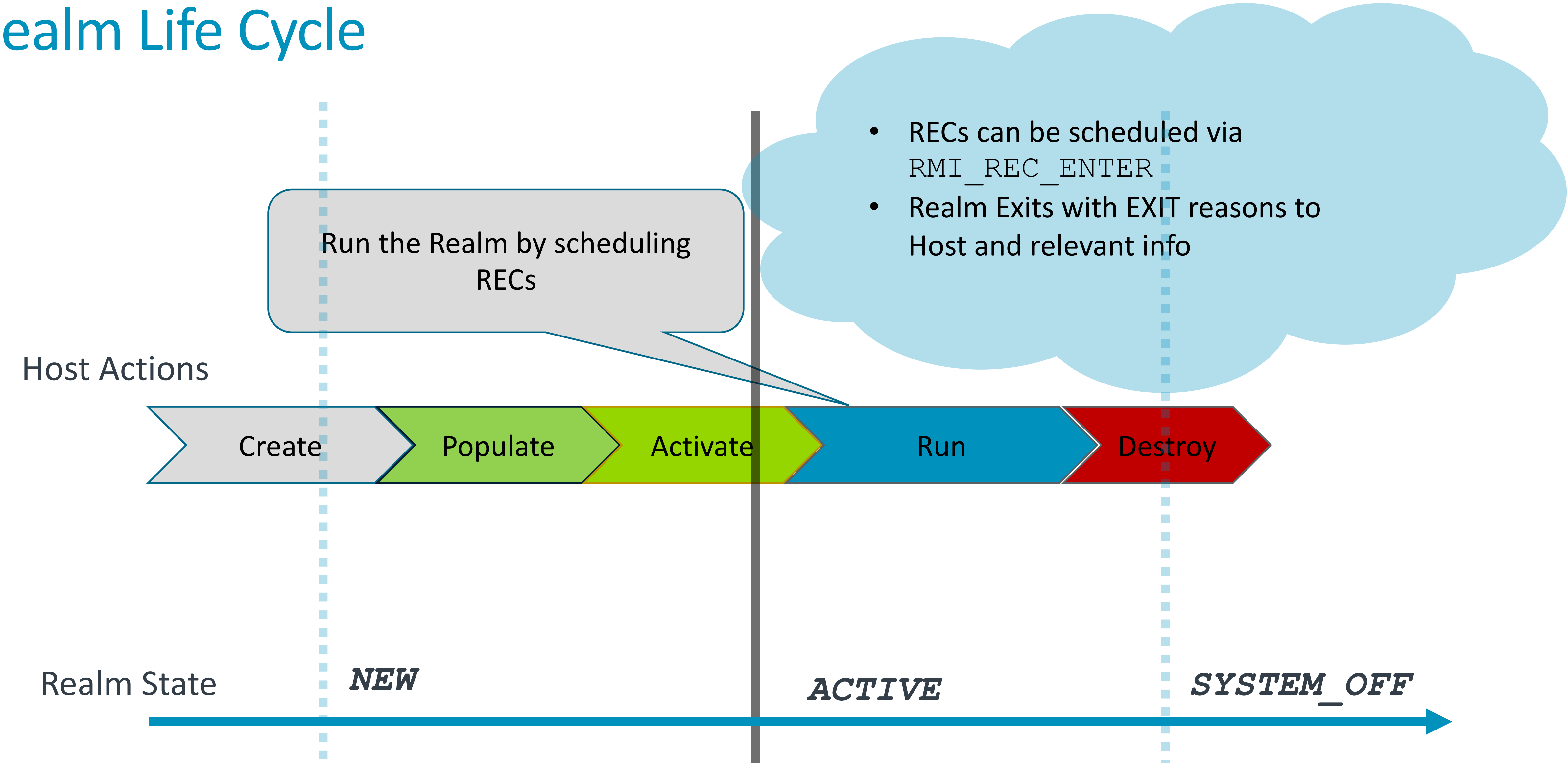
Realm Life Cycle



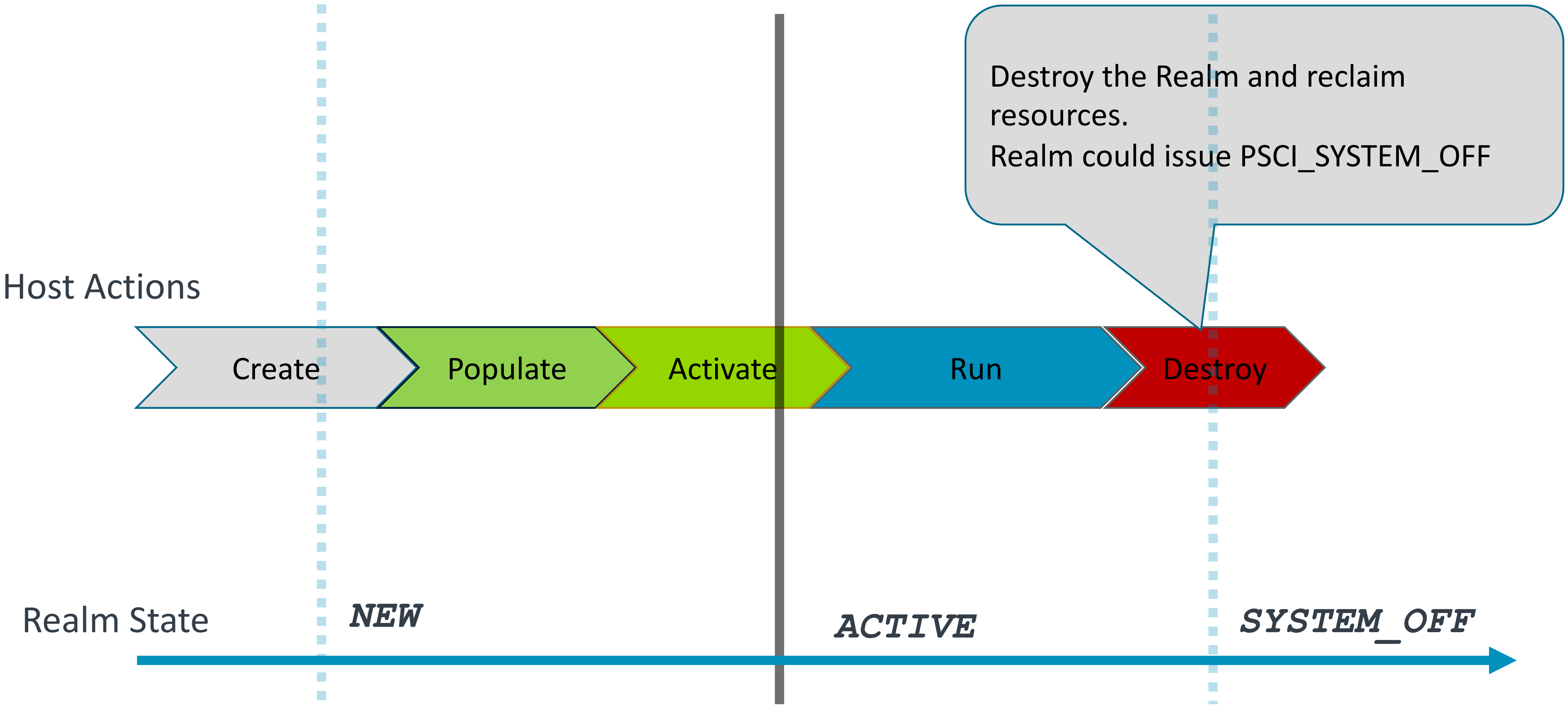
Realm Life Cycle



Realm Life Cycle



Realm Life Cycle



Arm CCA – REC Scheduling

- RECs once created, the context is invisible to the host
- RECs scheduled via `RMI_REC_ENTER(Rec, RmiRecRun)`
  - Inject Virtual Interrupts
  - Return MMIO reads to Unprotected IPA
    - Inject Sync. External Abort
  - Service Host Calls
  - Trap WFI/WFE
- Returns on Realm EXIT
  - Exit Reason with sufficient info
  - VGIC State
  - Timer State
  - PMU Overflow

```
struct rec_entry {

         /* MMIO, SEA, WFx */
        u64 flags;

        /* GPRS */

        /* VGIC Registers */
}

struct rec_exit {
        u64 exit_reason;

        /* Fault Info */
        /* GPRs */
        /* GICv3 Registers */
        /* Timer State */
        /* Set IPA State Request */
}
```

Arm CCA – KVM Support

Arm CCA – KVM Support

- Design principles
  - Reuse as much common code as possible
    - UABI, Guest Enter/Exit handling, VGIC
    - Add hooks for special handling in common code
      - `kvm_is_realm() / vcpu_is_rec()`
  - Hide and contain RMM interactions
- RMM Support advertised via new CAP: ***KVM_CAP_ARM_RME***
- Realm as new VM type at *KVM_CREATE_VM*
  - Bits[11:8] : 0 = Normal, x = Realm, y = ..
- Realm Configuration (for missing params) via ***KVM_CAP_ARM_RME***
  - *Hash Algorithim, SVE Vector Length, Realm Personalization Value*
  - Switch to VM attribute ?
  - Move some VCPU attributes to VM attributes ?
    - SVE vector length, PMU, Debug BP/WP registers

Arm CCA – KVM Support

- Realm Life Cycle managed via ***KVM_CAP_ARM_RME***
  - Initialize IPA State to RAM State for DRAM
  - Load initial image to Protected IPA
  - Set the Realm ACTIVE
- VCPUs set `KVM_ARM_VCPU_REC`
  - REC Creation via *`KVM_ARM_VCPU_FINALIZE`* `(KVM_ARM_VCPU_REC)`
  - May be do this from the kernel in the order of MPIDRs in one shot ?
- vCPU Scheduling follows common code, use RMM_REC_ENTER for the switch
  - Sync GPRs (HOST_CALL, MMIO read), vGIC state to `rec_entry`
  - Request WFx trap
- Realm Exit handled separately
  - Sync GPRs , vGIC state, Timer State from `rec_exit`
  - Handle any Realm world specific exits
  - Fallback to normal KVM handling for common exits (stage2 aborts)

Arm CCA – KVM Memory Management

- Stage2 controlled by RMM
  - Fixed 4K with variable IPA Size, L2 Block mapping (2M), LPA2
  - KVM support depends on `CONFIG_ARM64_4K_PAGES`
- KVM donates RTT pages
  - No shadow page tables
- No support for paging
  - Memory must be pinned by the VMM
- Restricted mem support – in progress/plan.
  - For RFC posts – Use normal anonymous memory, until it is merged
- Load the initial image for the Realm
  - Realm Initial State Measurement provides the guarantee for Realm
- Service "runtime" Stage2 aborts
- Stage2 tear down and reclaim memory
  - Move back to Normal world

Arm CCA - Support for RMM v1.0 - Status

- RMMv1.0-eac2 was publicly released on 7th June 2023
- RFC Series with RMM v1.0-Beta0 support (27th Jan 2023)
  - KVM, Linux Guest based on v6.0
  - kvm-unit-test and kvmtool
- Qemu support by Linaro
- RFC Series : Guest UEFI firmware Support
- Rebasing the work to RMM v1.0-eac2
  - Closely following the restrictedmem series
- Improve and generalize the UABI – Feedback please
- Add kselftests to stressing KVM driver/Linux
- Work in progress : Remote attestation flow Support
  - Boot Information Injection support
  - Guest Linux – Attestation/Measurement framework

Arm CCA – Future

- Devices Assignment for Realm
  - Allow (PCI)Devices access to/from Realm Memory – PCI TDISP
  - RMM to act as Trusted Security Manager (TSM)
  - Designing the low-level flow and RMM ABI
  - Keen to align with the “Generic” Linux/KVM story
- Partitioning of Realm Privilege Levels – Planes
  - Foundation for vTPM in Realm, with higher privilege than the OS
  - Design in progress
- Per Realm Memory Encryption Keys
- Paging
- Live Migration

Arm CCA - Realm IPA State Management

Arm CCA – Realm IPA State Management

- Realm's IPA space (controlled by ipa_size) is split into two halves.
  - protected - BIT(ipa_size – 1) == 0. RMM guarantees integrity
  - unprotected – BIT(ipa_size – 1) == 1 - No RMM guarantees
- Each "protected" IPA page has a state (Realm IPA State – RIPAS)
  - Controlled by Realm with the help of RMM, acknowledged by the Host
  - EMPTY - default state. Any access causes Synchronous External Abort to Realm
  - RAM – An area that is used as RAM memory by Realm, faults exits to Host
  - DESTROYED – An area that is untrusted due to Host operation (e.g. DATA destroyed)
- Dynamic memory sharing with fixed memory
  - A Guest – Host(KVM) agreement. Not mandated by RMM
  - All of guest RAM is protected
  - Realm sharing a page with IPA "x" follows
    - `RSI_IPA_STATE_SET(x, EMPTY) – Exits to Host, host requests RMM`
    - Realm access (x | BIT(ipa_size – 1)) . Update "stage1"
  - Keeps VMM memory layout unchanged (e.g., IO, PCI regions etc at lower end of IPA)

Arm CCA – Linux Guest Support

Arm CCA – Linux Guest Support

- Realm {I}PA State Management Support
  - Kernel image (image-header.size) and FDT must be marked as RAM (by host)
  - Scan FW description and initialize all of memory as RAM
    - UEFI behavior – TBD
- Detection of IPA size
  - Restrict PHYS_SHIFT_MASK to (ipa_size – 1)
  - BIT (ipa_size – 1) – Treated as a prot bit – "`PROT_NS_SHARED`"
  - Force SWIOTLB bounce buffering
- Host memory sharing
  - BIT(ipa_size – 1) is treated PROT_NS_SHARED
  - Force page level mapping for Linear map
  - Plugged into `set_memory_{en/de}crypted()`
- Virtio forced to use DMA API - (via `VIRTIO_F_ACCESS_PLATFORM`)
- GIC ITS Tables – Allocated as shared
- All I/O as non-secure by default - `ioremap()` adds `PROT_NS_SHARED`

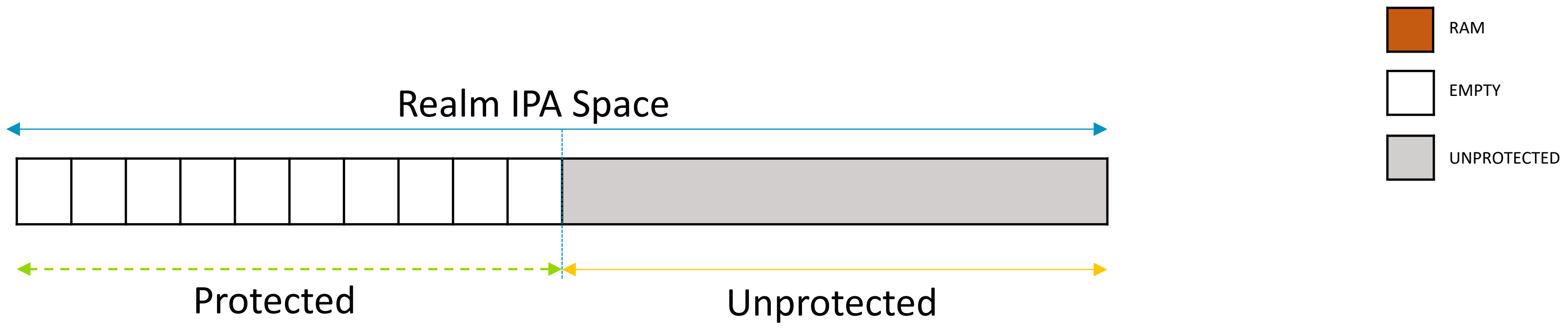
arm

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

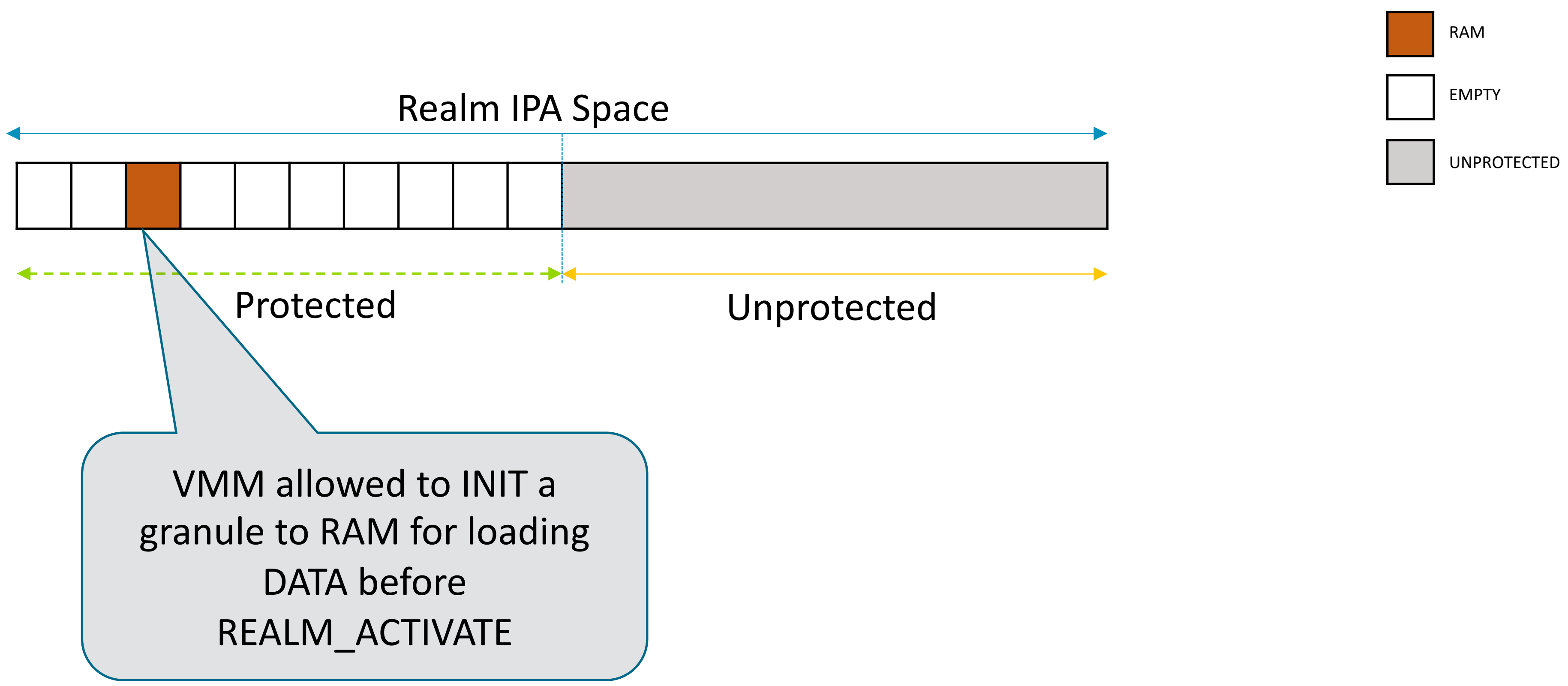
Backup

Arm CCA - Realm IPA State Management

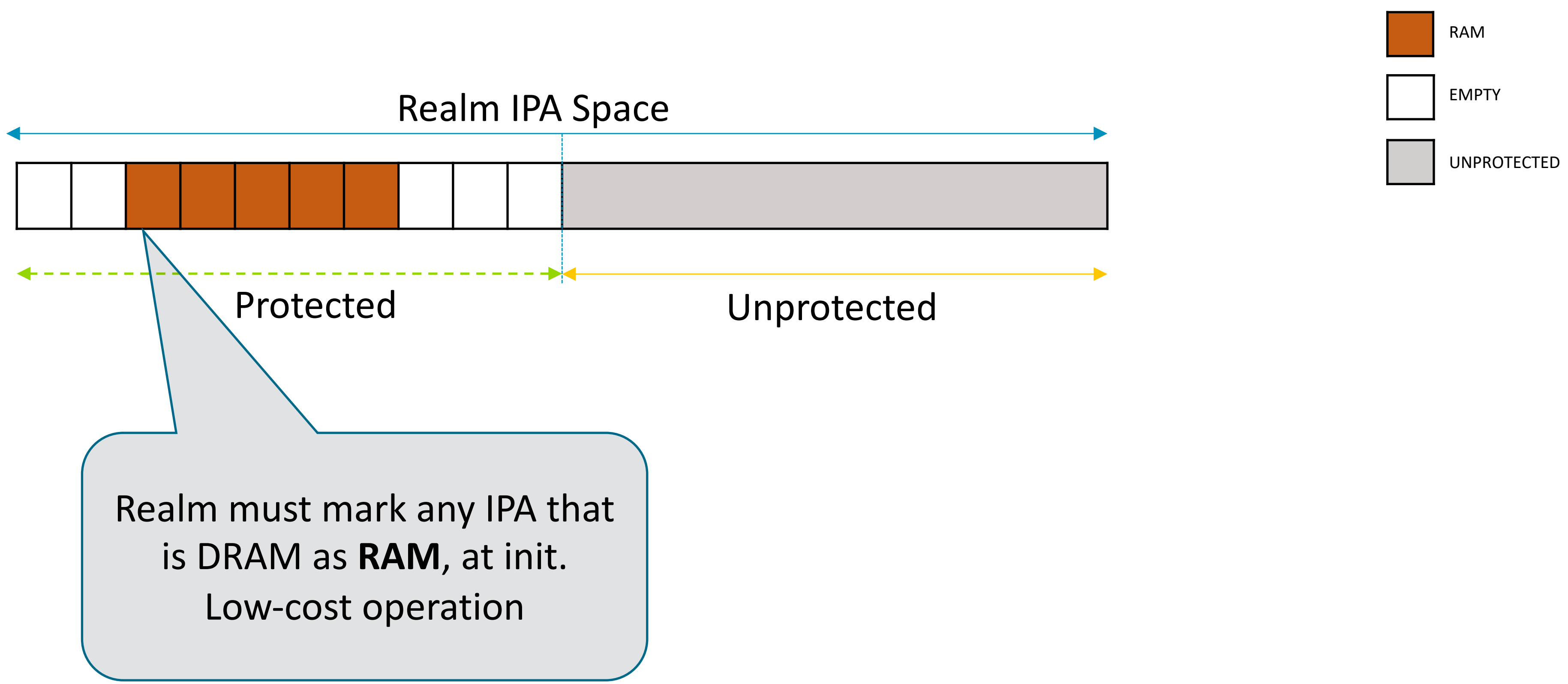
Arm CCA – Realm IPA Management



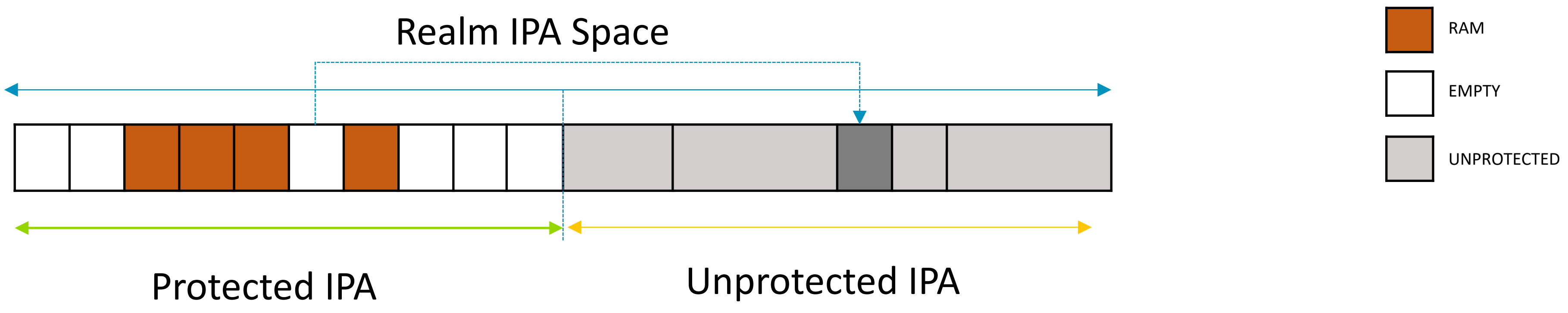
Arm CCA – Realm IPA Management

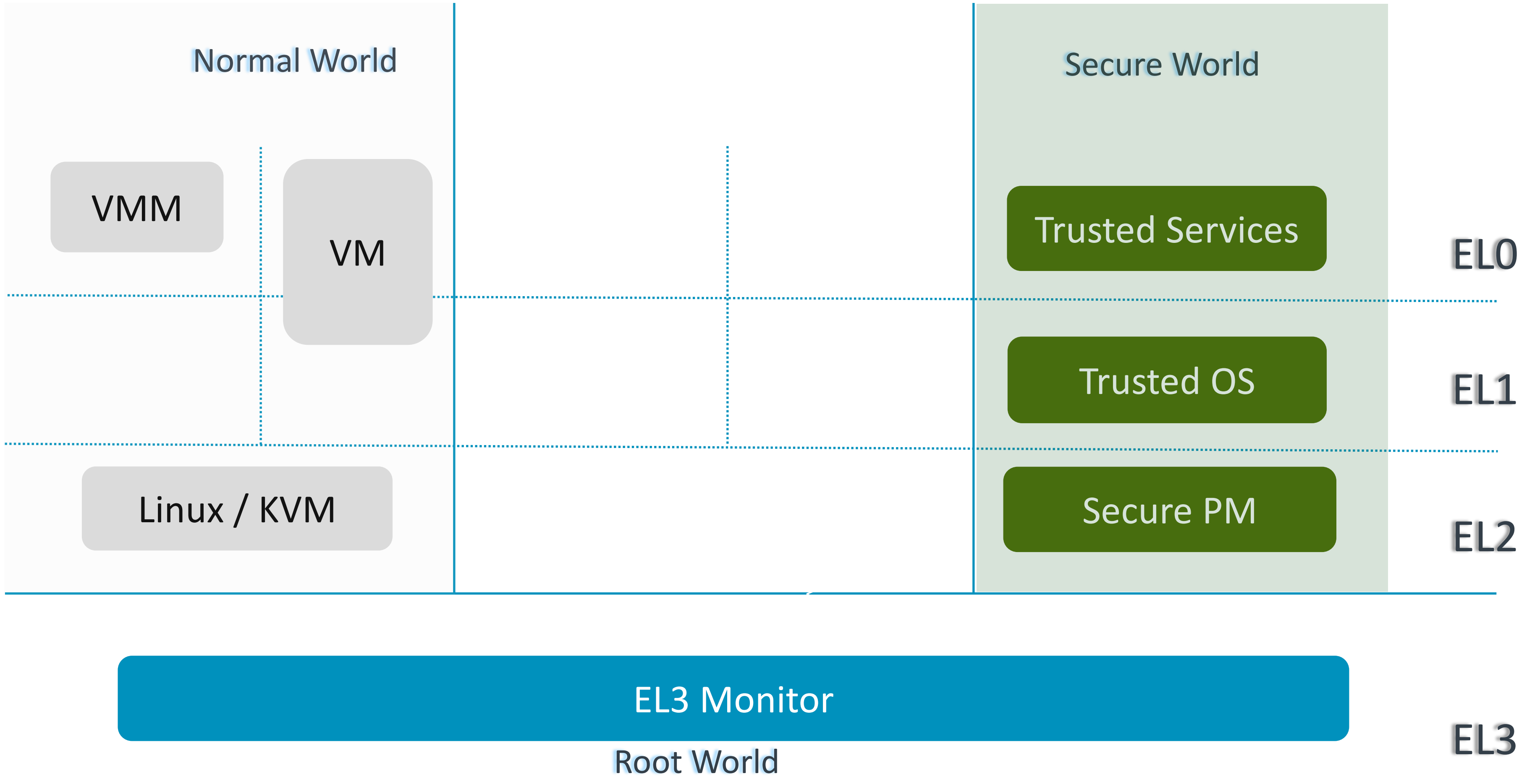


Arm CCA – Realm IPA Management



Arm CCA – Realm IPA Management




Normal World
Secure World
VMM
VM
Trusted Services
Trusted OS
Secure PM
Linux / KVM
EL0
EL1
EL2
EL3 Monitor
EL3
Root World

