

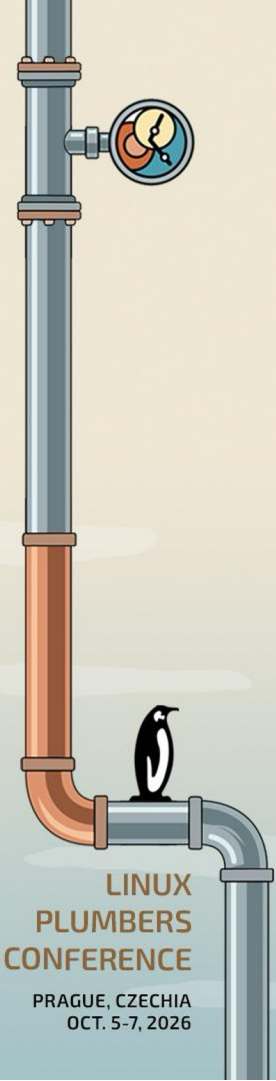
Arm CCA Realm Live Migration

Mathias Brossard, Arm



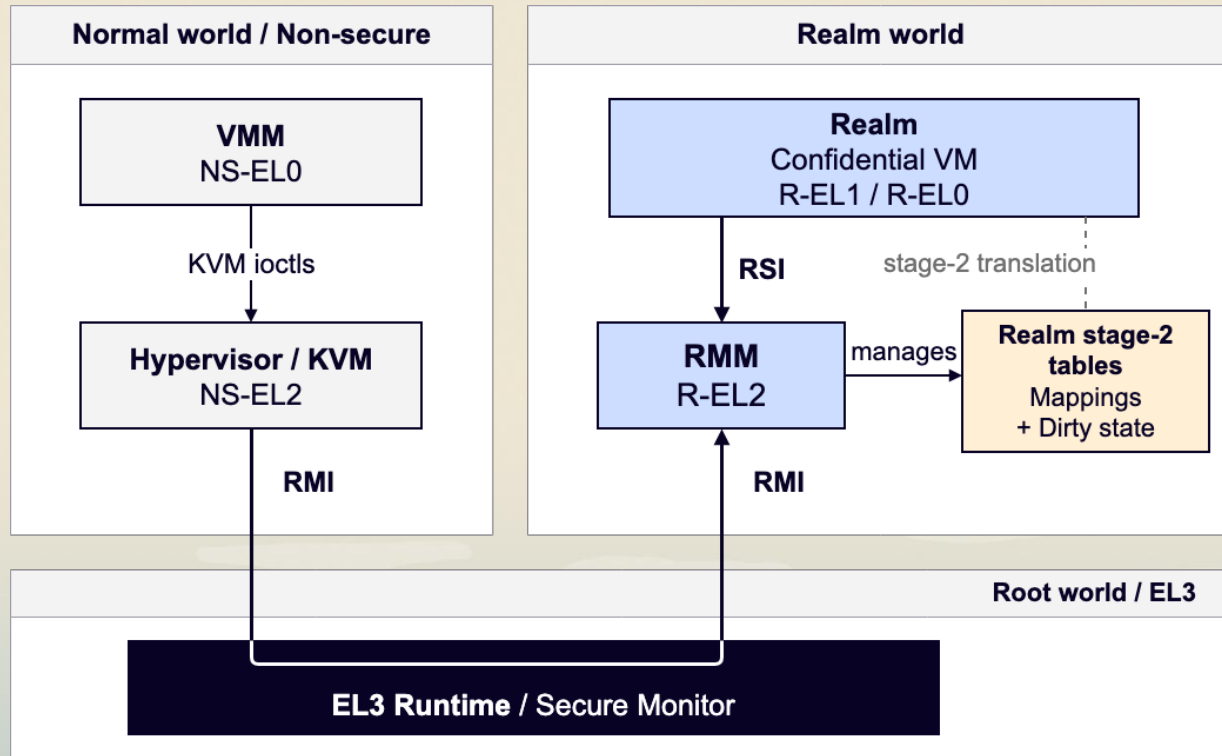
LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026



RMM: Realm Management Monitor
(Confidential Firmware)

EL: Exception Level
(Privilege Ring)

RMI: Realm Management Interface
(Host-to-Monitor Interface)

IPA: Intermediate Physical Address
(Guest Physical Address)

RSI: Realm Service Interface
(Guest-to-Monitor Interface)

SMC: Secure Monitor Call

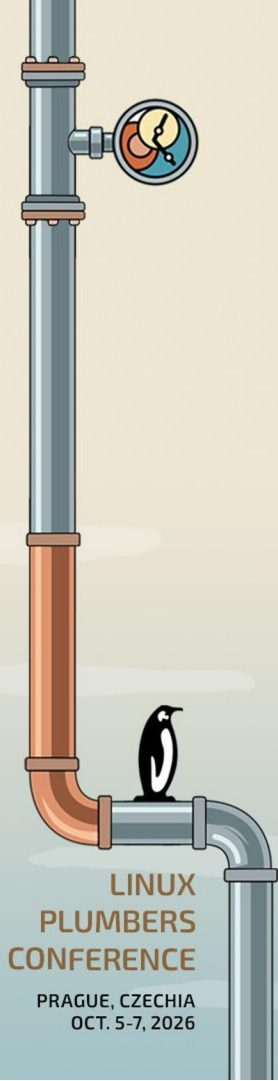
CCA Live Migration basic principles

CCA Live Migration design only aims to migrate "architectural state". This includes memory contents, CPU registers, contents of trusted page tables, and the state of architectural devices such as vSMMU. It excludes non-architectural devices such as PCIe functions.

Initial design will only support migration of page mappings; support for block mappings may be considered in the future.

Tracking of stale pages (pages export and then modified) can be done in two ways:

- Stage-2 page table dirty-bit **tracking**: CPU MMU and IO-MMU update table dirty bit; RMM scans the table to find dirty pages and returns the GPAs to KVM.
- Dirty-bit **logging** (future): CPU MMU and IOMMU append GPA to a buffer; RMM copies the log to a KVM-accessible location.



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

[RFC PATCH v2 0/4] Add KVM API for confidential guest live migration

Source host
=====

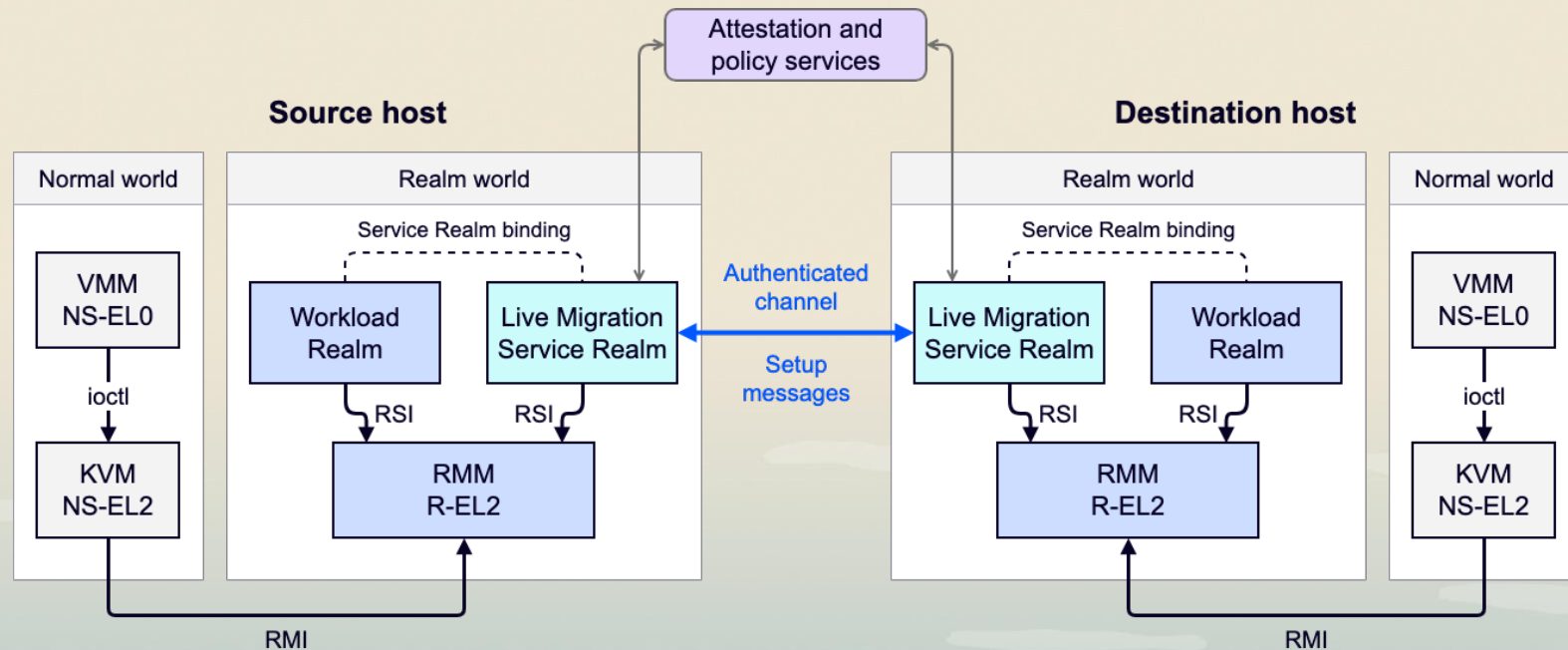
```
CMD(SETUP/SESSION)          <---- setup msgs ---->
      | (repeated)
CMD(SETUP/IMMUTABLE_STATE)  -- immutable state -->
      |
KVM_GET_DIRTY_LOG
KVM_EXPORT_MEMORY          ---- memory data ---->
CMD(ITERATION)             ---- epoch token ---->
      | (repeat until convergence)
CMD(STOP_AND_COPY/PAUSE)
CMD(STOP_AND_COPY/TD_STATE) ---- VM state ----->
KVM_EXPORT_VCPU             ---- vCPU state ----->
KVM_EXPORT_MEMORY          --- final memory --->
CMD(ITERATION/DONE)         ---- start token ---->
      |
CMD(END)
```

Destination host
=====

```
CMD(SETUP/SESSION)
      |
CMD(SETUP/IMMUTABLE_STATE)
      |
KVM_IMPORT_MEMORY
CMD(ITERATION)
      |
CMD(STOP_AND_COPY/TD_STATE)
KVM_IMPORT_VCPU
KVM_IMPORT_MEMORY
CMD(ITERATION)
      |
CMD(END)
```

(Source: Tony Lindgren and Artem Bitutskiy in <https://lore.kernel.org/all/20260831071304.762939-1-tony.lindgren@linux.intel.com/>)

Setup phase (context)



Setup phase (observations on KVM proposal)

Source host

=====

CMD(SETUP/SESSION)

| (repeated)

CMD(SETUP/IMMUTABLE_STATE)

<---- setup msgs ---->

-- immutable state -->

Destination host

=====

CMD(SETUP/SESSION)

|

CMD(SETUP/IMMUTABLE_STATE)

Concern: the proposal is a semantic mismatch for CCA, because session establishment does not flow through the host.

For CCA, the setup phase is broken down differently:

- The first part sets migration parameters, allocates tracking structures.
- [The session establishment happens separately through the Service Realms, in the middle of this setup phase]
- The second part transfers some immutable state and more importantly, enable dirty bit machinery to start migration.

Pre-Copy phase (context)

Pre-Copy or (Source) Brownout

Support for concurrent export operations based on:

- Exports are tied to a “stream” for export/import ordering and integrity.
- The mapping of pages to streams is tracked by RMM, and can change across generations (iterations/epochs).

Memory exports consist of:

- Metadata (GPA information, integrity protection values)
- Page content (optional)

Pre-Copy phase (observations on KVM proposal)

Source host

=====

[...]

KVM_GET_DIRTY_LOG

KVM_EXPORT_MEMORY

CMD(ITERATION)

| (repeat until convergence)

---- memory data ---->

---- epoch token ---->

Destination host

=====

|
KVM_IMPORT_MEMORY

CMD(ITERATION)

|

Concerns about KVM_GET_DIRTY_LOG:

- CCA does not maintain the "dirty log" but instead uses **concurrent** scanning with **caller-provided GPA ranges**.
- RMM returns a list of GFNs the memory export function can consume. Proposal round-trip through bitmap adds overhead.

Feedback about KVM_EXPORT_MEMORY / KVM_IMPORT_MEMORY :

- "transfer buffer size is vendor specific": the size depends on runtime values.
- Buffer layout: CCA uses separate content and metadata records. A single buffer cannot be treated as opaque from the point of view of the caller.
- Error handling should include failing GFN entry or method to determine number of successful exports.
- KVM_IMPORT_MEMORY gets the GFNs from the metadata rather than the input array.



LINUX
PLUMBERS
CONFERENCE

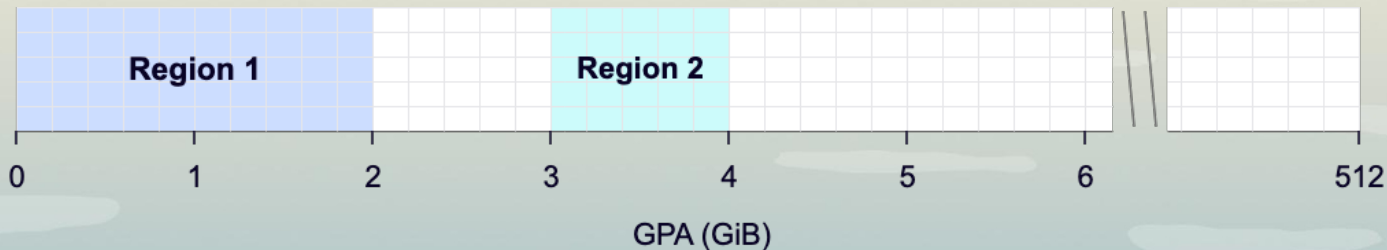
PRAGUE, CZECHIA
OCT. 5-7, 2026

Stop-and-Copy phase (context)

Stop-and-Copy or Cold-Copy or Downtime

Pages tables can contain information even if there is no content page.

We do metadata-only export for parts of the GPA range outside of KVM memory slots (the white parts).



Stop-and-Copy phase (observations on KVM proposal)

Source host

=====

[...]

CMD(STOP_AND_COPY/PAUSE)

CMD(STOP_AND_COPY/TD_STATE) ---- VM state ----->

KVM_EXPORT_VCPU ---- vCPU state ---->

KVM_EXPORT_MEMORY --- final memory --->

CMD(ITERATION/DONE) ---- start token ---->

|

CMD(END)

Destination host

=====

|

CMD(STOP_AND_COPY/TD_STATE)

KVM_IMPORT_VCPU

KVM_IMPORT_MEMORY

CMD(ITERATION)

|

CMD(END)

Note: The excerpt does not have KVM_GET_DIRTY_LOG. We assume it was omitted and should be placed before KVM_EXPORT_MEMORY.

Concern about limitations of current KVM_GET_DIRTY_LOG API:

- CCA needs to scan whole the GPA range (outside KVM memory slots) concurrently.

Feedback KVM_EXPORT_VCPU:

- Buffer size needed will vary depending on implementations, so size needs to be queried at runtime.

Thank you!

- Our main concern is about KVM_GET_DIRTY_LOG.
- Another class of questions revolves around how the caller can determine the size of the transfer buffer and then, after the call, which parts of it were consumed.

Questions?

