

Exploring KVM and SMMUv3 Stage-2 Page Table Sharing for arm64 pKVM

KVM MC - LPC 26
Mostafa Saleh (Google)



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

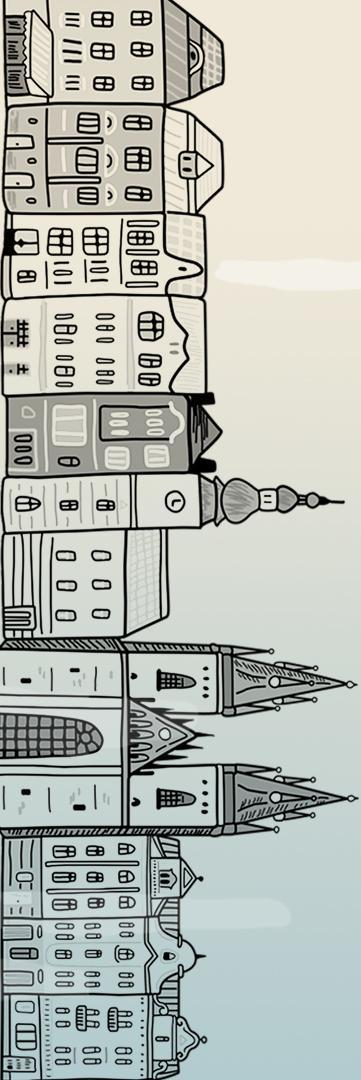


Presenter

- Software Engineer at Google
- Android Systems (KVM and IO virtualization)



Background & Related Work



LINUX PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Protected KVM (pKVM)

KVM protected mode (arm64 upstream[1], x86[2])

A tiny part of the kernel runs at a higher privilege level

- The host kernel (EL1) considered untrusted.
- pKVM uses the stage-2 MMU to isolate the host from guest/hypervisor.
- Other SW side channels (proxy TrustZone communication)



CoCo that doesn't need any fancy HW

[1] [Documentation/virt/kvm/arm/pkvm.rst](https://docs.kernel.org/virt/kvm/arm/pkvm.rst)

[2] https://www.youtube.com/watch?v=EP9ps_h-Wel

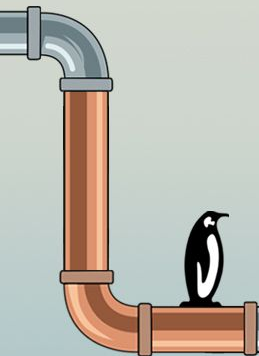
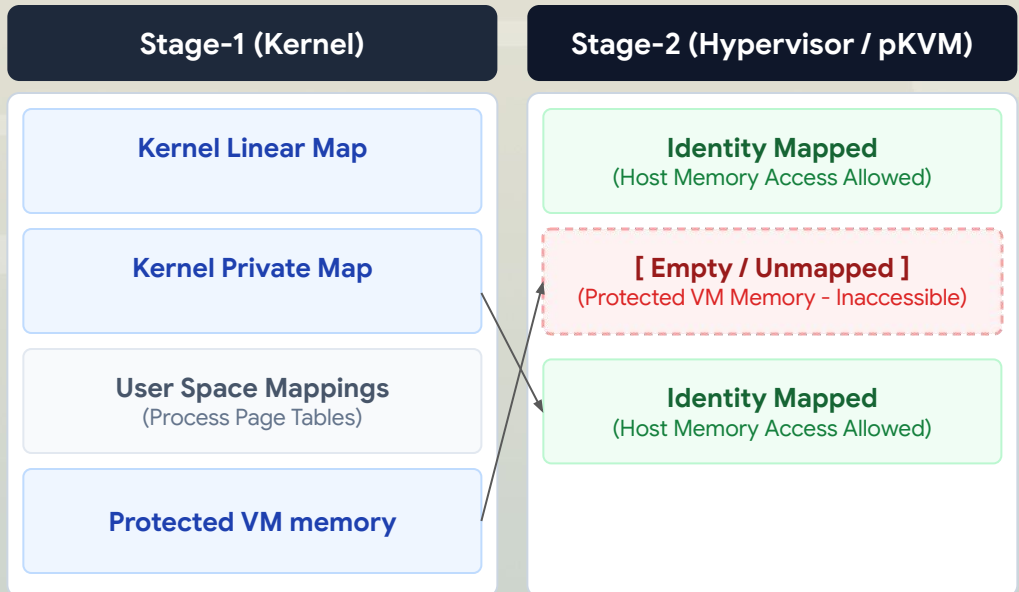


pKVM - CPU memory isolation (MMU)

With protected mode, the **host** runs with **dual stages of translation**

Stage-1: Managed by the kernel

Stage-2: Managed by the hypervisor, **identity mapped** with holes for memory owned by the hypervisor or protected virtual machines.



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

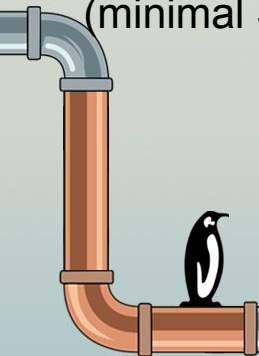
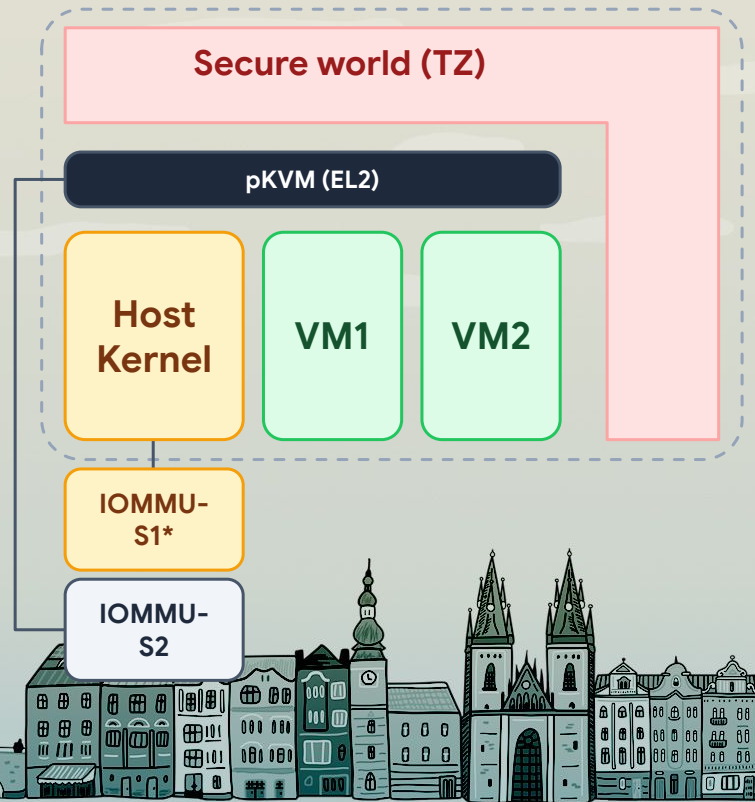


pKVM - DMA isolation

However, **DMA isolation** is missing, that means it is trivial to bypass the CPU MMU stage-2 using DMA capable devices to access protected memory (hypervisor or guest).

DMA isolation can be established by an **IOMMU**

IOMMUs **MUST** be managed by the hypervisor (minimal SW environment, no IOMMU API, no MM)



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

pKVM - DMA isolation

Pre-v4: Para-virtualization (Hypercalls)

- Map/unmap latencies.
- Complex memory management
- Still useful if DMA isolation is required with a single stage IOMMU or to reduce translation overhead.

V4-now (v8): Trap and Emulate

- Minimal change to the kernel driver (some code split for reuse).
- The hypervisor **traps and emulates** MMIO access from the SMMUv3 transparently.
- **Requires nested translation**



Related work

- [KVM forum 2021](#) Sharing IOMMU page tables with TDP in KVM
 - Discussion around pinning, uAPI and VFIO/KVM coordination
 - Guest CPU / IOMMU page table sharing to save memory



Kernel - SVA

Sharing of **stage-1 CPU** page table with the IOMMU for a **process** and device.

- SMMUv3 must satisfy some **features** (cache coh, PASIDs, page size, OAS, BBML2. HA/HD)
- Relies on the SMMUv3/device handling **page faults**:
 - SMMUv3 supporting STALLs (IDR0.STALL_MODEL) and the device too (**dma-can-stall**)
- **TLB invalidation** through **MMU notifiers**, **DVM** not used because
 - Complexity matching the ASID, and VMID (patches), **not used in practise?**

Typically niche cases (PCI accelerators)



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026



Other CoCo solutions

CCA: GPT: PA indexed table containing a Physical Address Space (PAS) owner (NS, S, Realm, Root ...)

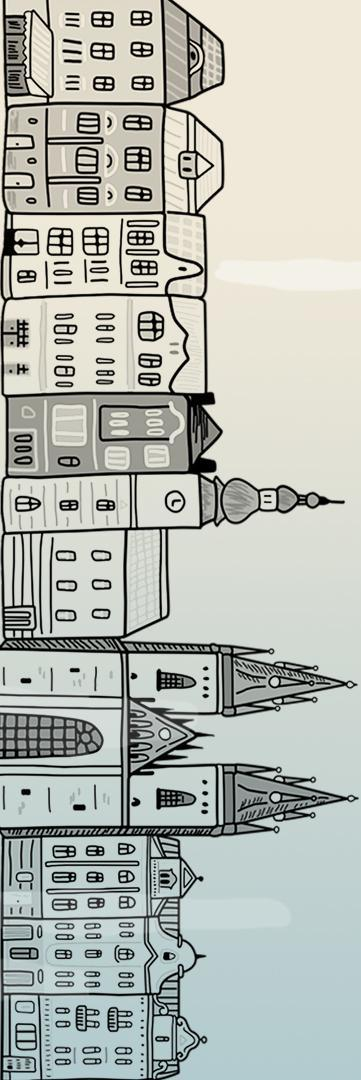
Owned by Root (the highest privilege), SMMUv3 extension.

TLB invalidation: Register based invalidation (or DVM)

Similarly, AMD SEV-SNP: RMP, INTEL TDX: PAMT



Problem & Proposal



LINUX PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



What is currently on the list (v8)

Page table shadowing

- Hooks for page table updates (on **caller** level)
- Reserving another carveout for SMMUv3 (~20-30MB)
- Borrow io-pgtable-arm from the kernel
 - Stubbing a bunch of kernel API.
 - Designed around the IOMMU API

file: mem_protect.c

```
ret = kvm_pgtable_stage2_map(&host_mmu.pgt, start,  
size, start, prot);
```

```
[...]
```

```
+ ret = pkvm_iommu_host_stage2_idmap(addr, addr +  
size, prot);
```

Alternative, **page table sharing** with the CPU?



Requirements (Subset of kernel SVA)

SMMUv3 must satisfy some features (**cache coh**, ~~PASIDs~~, page size, OAS, BBML2. ~~HA/HB~~)

Relies on the SMMUv3/device handling **page faults**

MMU notifiers for **TLB invalidation**



Page faults

We **can't rely** on all devices on the system and SMMUs to handle page faults and **stall**.
(unlike SVA)

But why do we have page faults in the first place? And can it be eliminated?



Page faults

Actually the CPU page table is not **fully** populated!

Some entries left unmapped and will be lazily faulted.

Some entries are transiently invalid. (LOCKED)

Page Table Entries (Stage-2 Translation)

0x00 [PTE 0] Mapped (Valid)

0x01 [PTE 1] Mapped (Valid)

0x02 [PTE 2] Mapped (Valid)

0x03 [PTE 3] Invalid (No access)

0x04 [PTE 4] Invalid (No access)

0x05 [PTE 5] Mapped (Valid)

0x06 [PTE 6] Mapped (Valid)

0x07 [PTE 7] Empty (lazily mapped)

0x08 [PTE 8] Invalid (No access)

0x09 [PTE 9] Empty (lazily mapped)

0x0A [PTE 10] Mapped (Valid)

0x0B [PTE 11] Invalid (LOCKED)



Page faults

Sources of the page fault (that are not fatal and should be recoverable)

Hypervisor logic

- 1) Empty page table after boot (lazy mapping)
- 2) MMIO unmapped under pressure because of carveout heuristics

Pgtable implementation

- 1) Lazy mapping when breaking blocks
- 2) BBM sequence (LOCKED)



Page faults

Hypervisor logic

- 1) Empty page table after boot (lazy mapping) => **Populate everything at boot**
- 2) MMIO unmapped under pressure because of carveout heuristics => **Populate all MMIO with block mapping (no device assignment)**

All address space, example: 48bits(256TB) (no LPA2)

- 4K => 2MB
- 16K => 64MB
- 64KB => 4MB

Alternative: set ceiling of MMIO from the cmdline, HVCs?



Page faults

Pgtable implementation

- 1) Lazy mapping when breaking blocks => Eager mapping [1]
- 2) BBM sequence (Splitting blocks) => BBML3 [2]

[1] <https://lore.kernel.org/all/20260911135053.146435-2-vdonnefort@google.com/>

[2] <https://lore.kernel.org/all/20260904132855.638117-1-smostafa@google.com/>



Page table coherency

Note: No HW (HTTU..) page table updates

io-pgtable-arm support non-coherent walkers by issuing CMO after updating the page table.

We can do the same in the KVM page table (in the presence of non-coherent walker)

There is period of time where there is an inconsistent view of memory between the CPU and the SMMUv3, but that shouldn't be a problem.

Otherwise, this is not a deal breaker and we can not support this case.



TLB invalidation

Hooks for TLB invalidation

SMMUv3 supports range invalidation, requiring:

- Address range, TG, addr range, **TTL**, **leaf**.

TTL and leaf only known to the page table walker:

1. Callbacks in pgtable.c (invalidate(), sync()) with **kvm_pgtable_force_pte_cb_t** as precedent.
2. Do it from caller + over invalidation (always non-leaf, no TTL hint)

```
struct kvm_pgtable_tlb_ops {  
    void (*inv_range)(u64 addr, u64 size,  
                      s8 level, bool leaf);  
    void (*sync)(void);  
};
```

```
struct kvm_pgtable {  
    [...]   
    enum kvm_pgtable_stage2_flags flags;  
    kvm_pgtable_force_pte_cb_t force_pte_cb;  
    + const struct kvm_pgtable_tlb_ops  
    *tlb_ops;  
};
```



Other nits

Mismatched CPU and SMMU features:

- LPA2 (arm64.nolva)
- FWB is now opportunistically used for the CPU.
- OAS doesn't usually match PAS

Would require command line overrides as they change the page table shape or encoding.

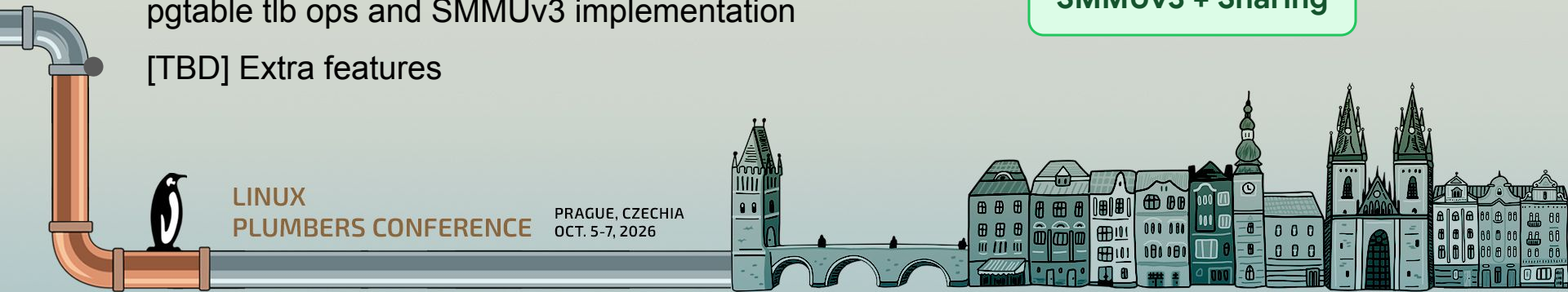
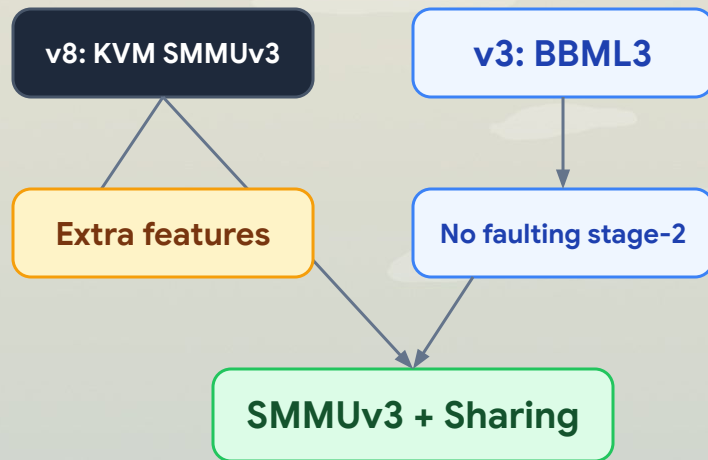
Other improvements:

- Page table coalescing support



Patches

- [v8] KVM SMMUv3 (25): Shadowing
- [v3] **BBML3** (2): useful beyond SMMUv3
- **No faulting stage-2** (4): Pre-populate, MMIO, pgtable split (useful beyond SMMUv3)
- [TBD] SMMUv3 page table sharing (4): Add pgtable tlb ops and SMMUv3 implementation
- [TBD] Extra features



Discussion

Sharing page tables should be possible for **some** systems, but it is not a replacement for shadowing.

Is it worth it to have it?

Design decisions:

- Page faults
- TLB
- Coherency

Not discussed here: DVM, BBML3/2, guest VM support



Backup

LINUX PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Aug 2022

TLB invalidation: DVM

Distributed Virtual Memory (DVM)/Broadcast TLB Maintenance (BTM):

- Require HW support (Interconnect...)

DVM is not really free

- Impacts system performance (interconnect traffic, DSB latency)

Still need a way to invalidate **ATC**



Page faults: BBML3

Patches for BBML3 in KVM are on the list, but what if BBML2 was supported?

SMMUv3 BBML2 doesn't generate conflict aborts.

Can we handle conflict aborts in EL2 on the CPU?

Issue TLBIALl or based on FAR_EL2.

Note: the abort is not in the same address space of the hypervisor (no double faults)

