Horizon Robotics

IOMMU solutions for seL4 (take ARM SMMUv3 for instance)

seL4 Summit 2023

KAN QIU kan.qiu@horizon.cc
LEI MAO lei.mao@horizon.cc

Horizon Robotics

Agenda

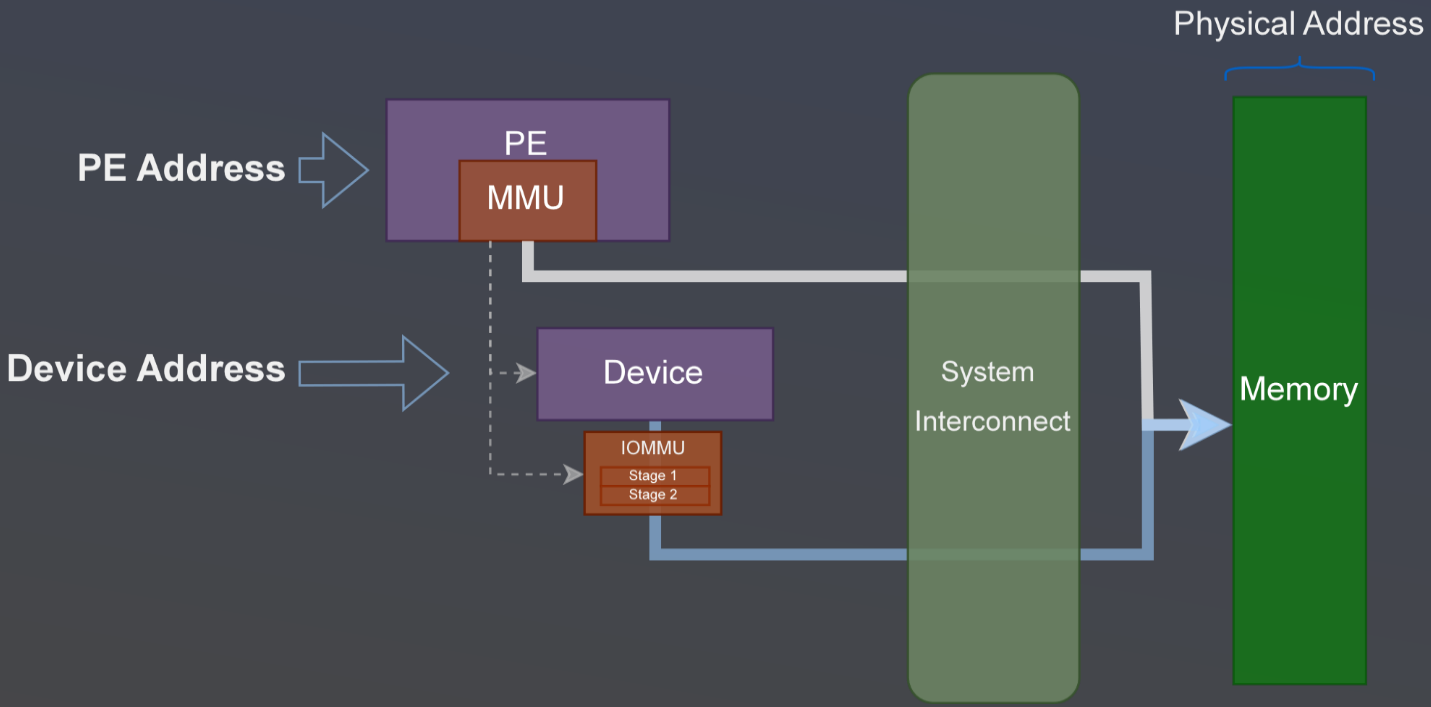
- Context
- Infrastructure
- Technical Challenges
- Summary
- Future Plans

Context

DMA device which is capable of overwriting any device-addressable memory on the whole system has potential security & safety problem.

IOMMU provides a virtual address space to DMA-capable devices, solving addressing issues and setting up transparent scatter/gather operations.

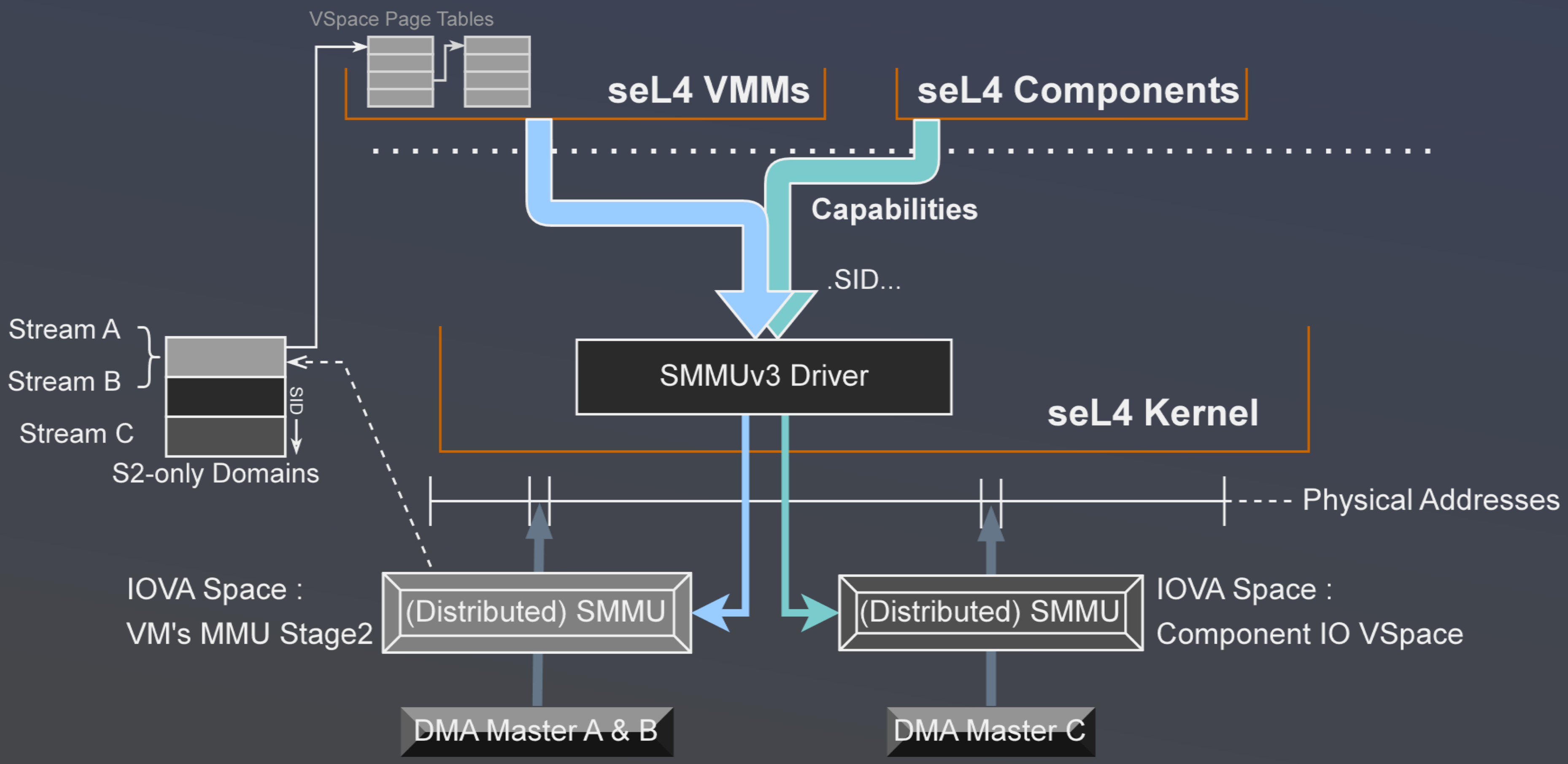
To prevent violation of the isolation model, the driving force is virtualization at this point, and the widely used IOMMU(in ARM case: SMMU) virtualization scheme in a Hypervisor is Stage-2 only or one-level stage.



Context

Horizon Robotics

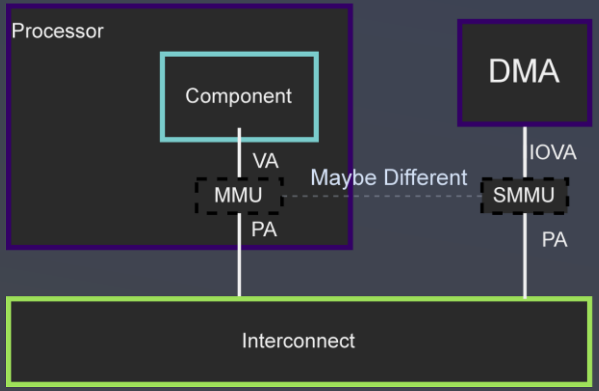
The VMM and other components maintain their own IO domains through the unified kernel interfaces that systematically manages all SMMU hardware abstracts, and has the proven reliability.



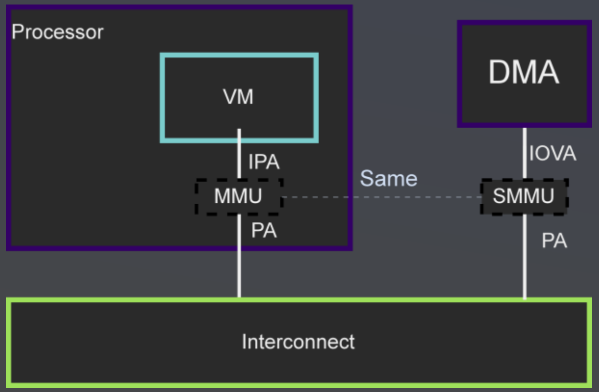
Infrastructure

Horizon Robotics

Exist SMMU Stage-2 Only scheme is simple enough:

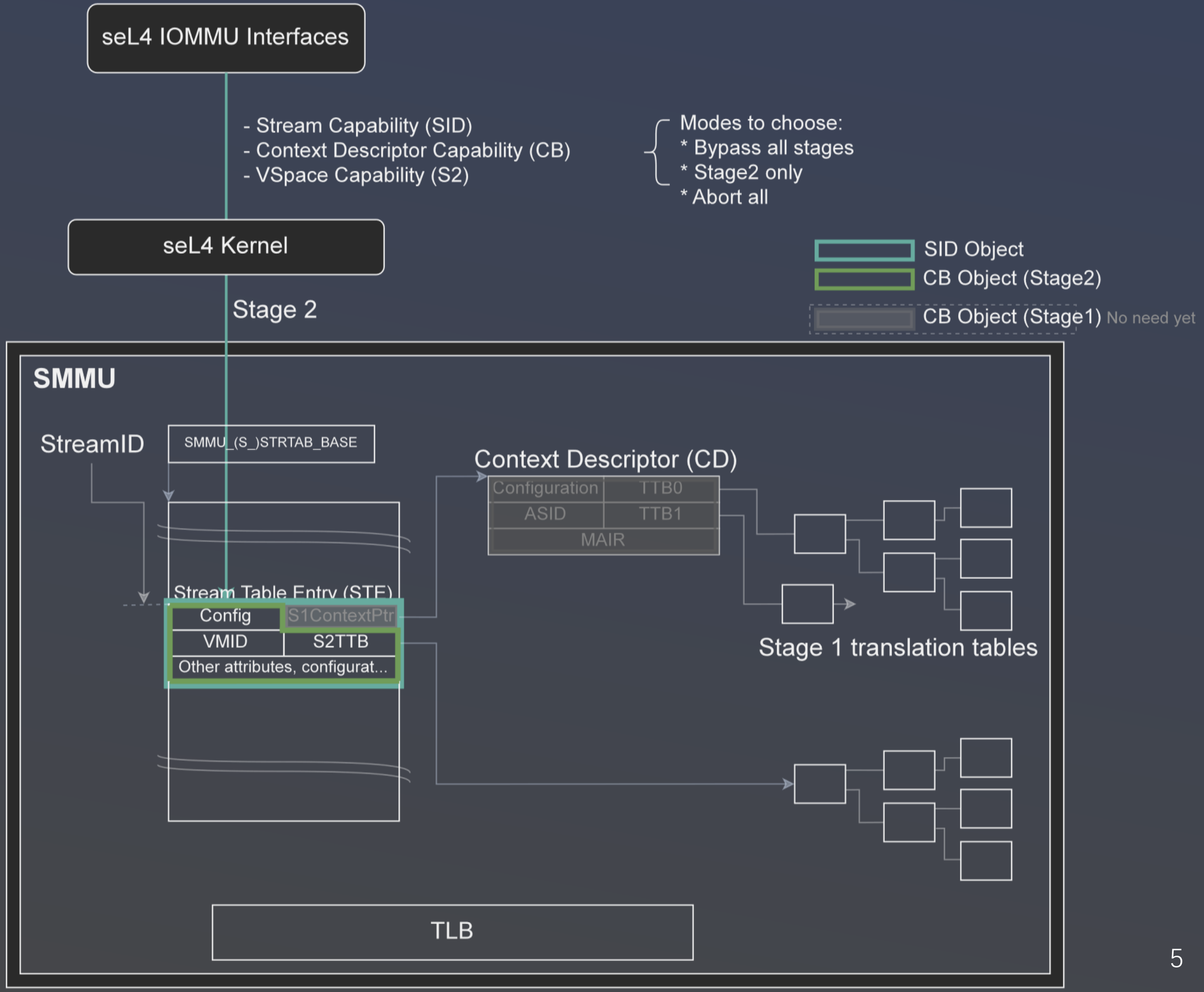


Use case for the native user-space components



Use case for virtualization components

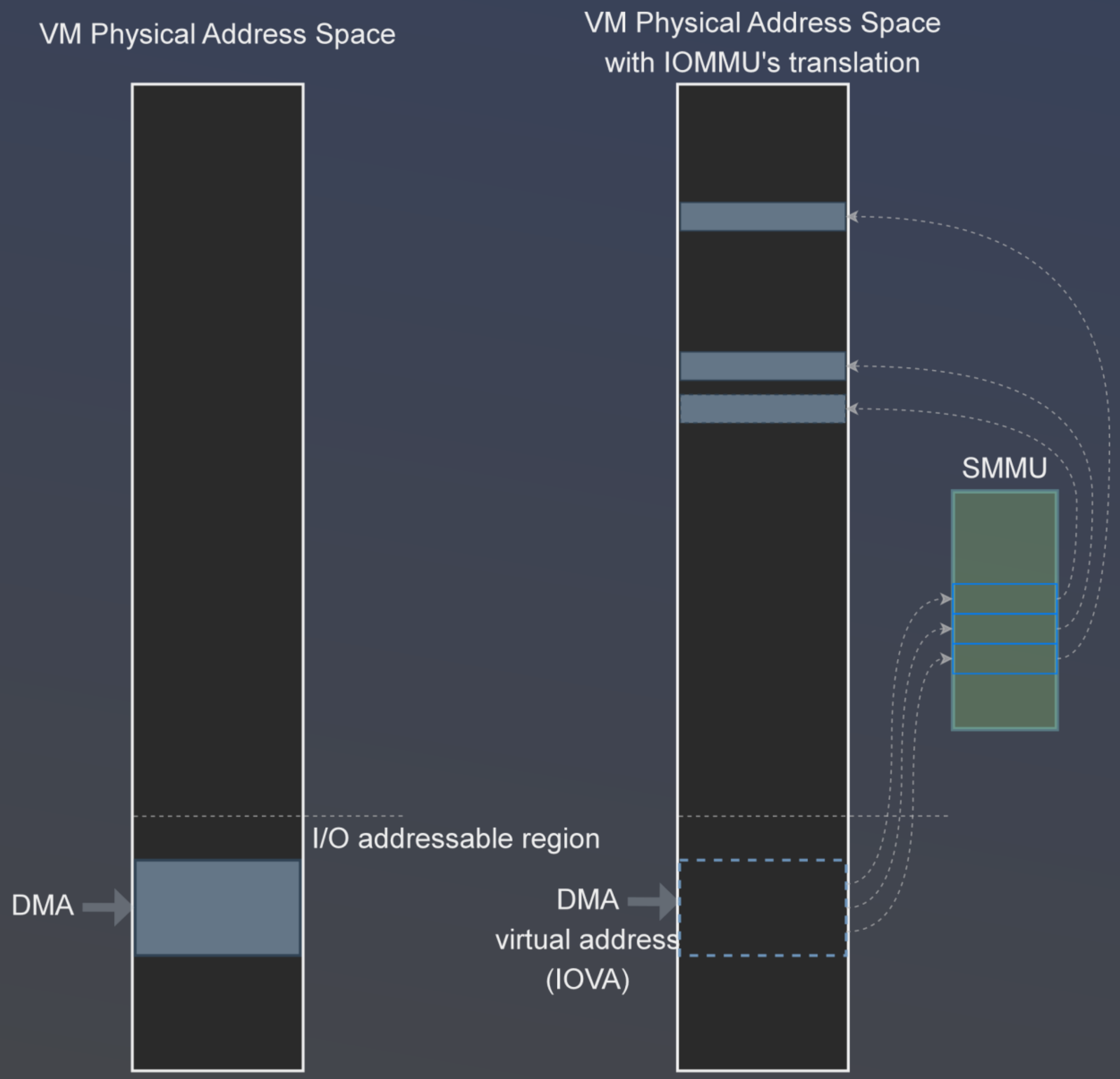
Component



Infrastructure

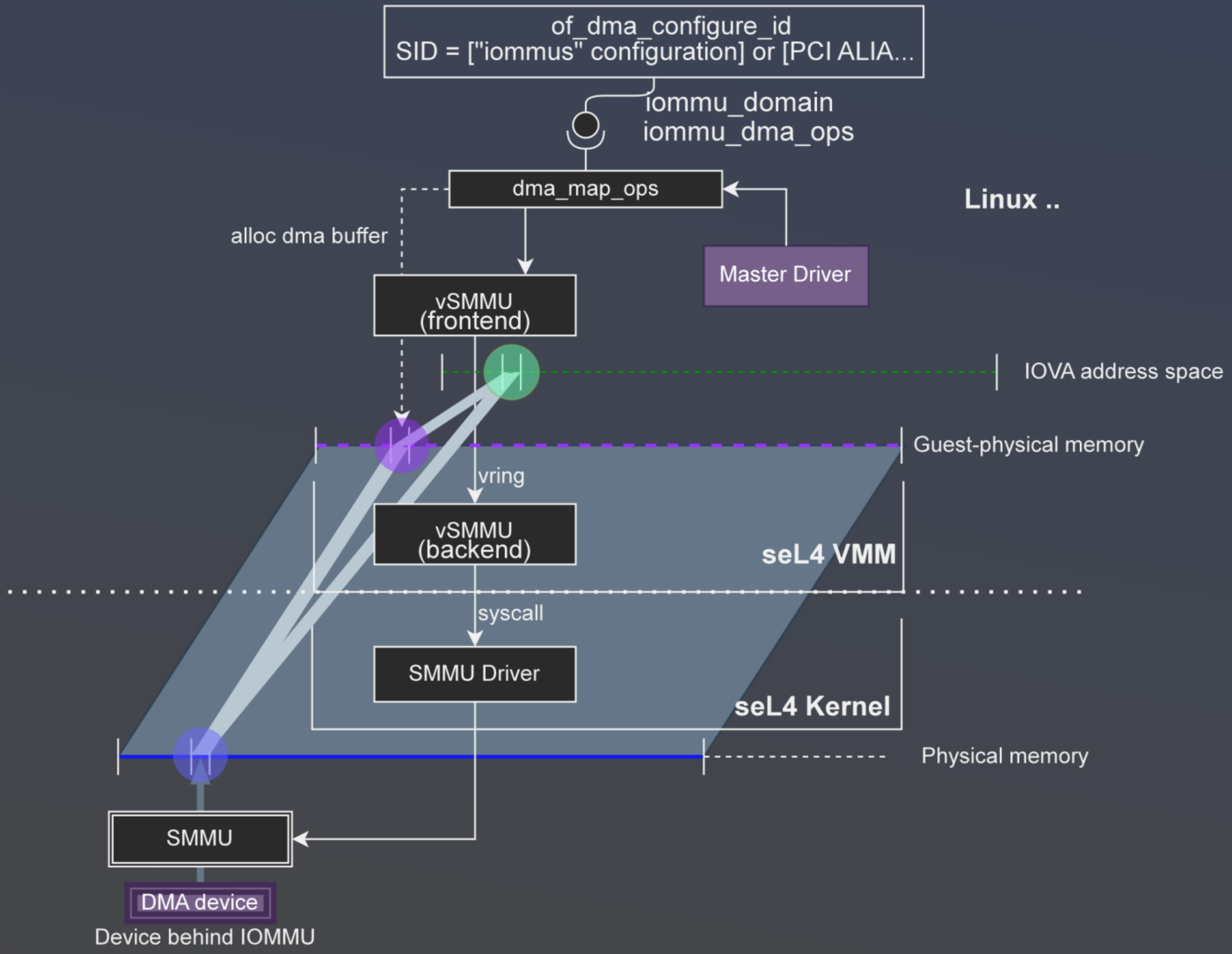
Although the IOMMU Stage-2 only scheme (the view of SMMU Stage-2 is as same as the MMU's) is simple, and seems enough at most of time for virtualization.

But in order to address physical memory fragmentation, and to avoid bounce buffers, the preferred approach seems to enable para-virtualization.
(There are legacy DMA IPs which can only access 24/32bit address)


VM Physical Address Space
VM Physical Address Space with IOMMU's translation
SMMU
I/O addressable region
DMA
DMA virtual address (IOVA)

Infrastructure

For a more flexible and finely controlled Stage-1, what firstly comes to mind is classic VIRTIO methodology.
The most important thing we need to do is interpreting requests from VIRTIO frontend into kernel system calls.



Infrastructure

We did following thing to enable SMMUv3 support in seL4

- ✓ first porting the SMMUv3 driver into the seL4 kernel
- ✓ modify Haskell scripts and RootServer to statically pre-generate and manage various SID/CB objects
- ✓ modify CAmkES scripts to configure and derive specified capabilities for components and virtual machines.

CAmkES Template：

```
/*# Generate SMMUv3 capabilities #*/
/*- set sid_cap = alloc(name='sid_%d_%d' % (id, sid), type=seL4_ARMSID, smmu_id=id, sid=sid) -*/
/*- set cb_cap = alloc(name='cb_%d_%d' % (id, sid), type=seL4_ARMCB, smmu_id=id, cb=sid) -*/
```

CAmkES Configuration：

```
vm0.smmu_sids = [
{"smmu_id":0, "sid":1},
{"smmu_id":0, "sid":2},
{"smmu_id":0, "sid":3},
];
```

Infrastructure

I/F - Function	Changes
decodeARMSIDControlInvocation	{ADD CMD} ARMSIDGetGlobalerror ARMSIDGetEventq ARMSIDSkipCmdqError ARMSIDClearGlobalerror ARMSIDDisableSMMU ARMSIDInitSMMU
decodeARMCBInvocation	{ADD CMD} **ARMCBAssignCDTable – stage1 CD Bind**

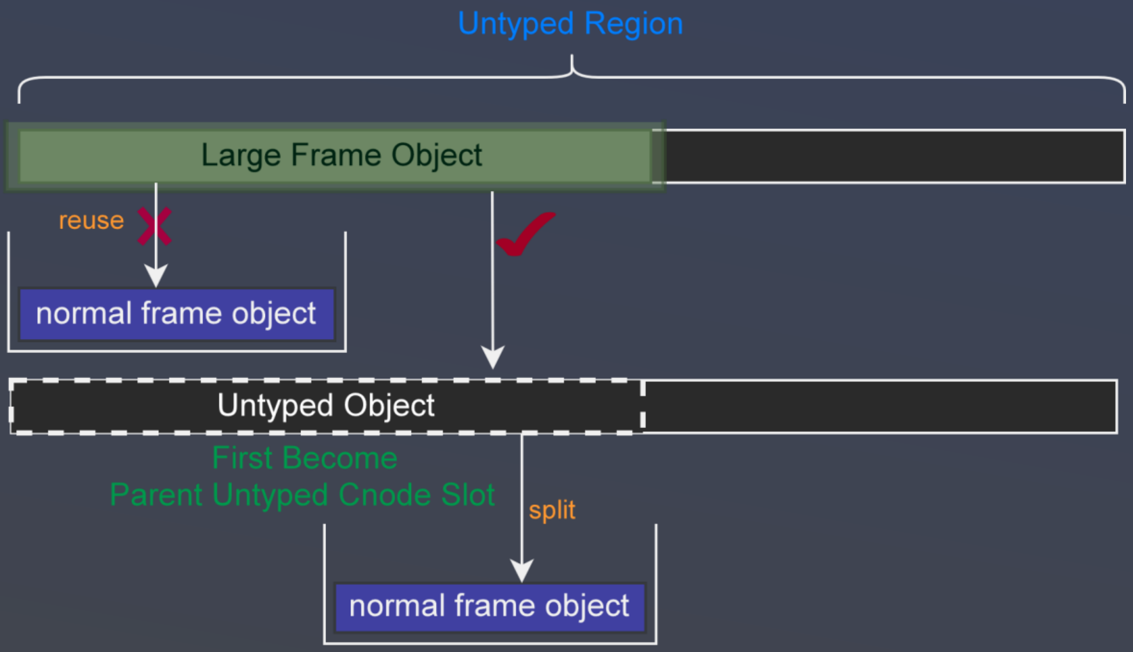
I/F - Typedef Structure	Changes
page_upper_directory_cap	{ADD MEMBER} field capPUDSMMUID 4
block sid_cap	{ADD MEMBER}; field capDevID 4 {MODIFY BIT.LEN} field capSID 12->32 padding 52->28
block cb_cap	{ADD MEMBER}; field capDevID 4 {MODIFY BIT.LEN} field capBindSID 12->32 padding 40->16

Technical Challenges

1. If we already distribute the region into one huge or several large frames, meanwhile, we can not reuse it as normal pages for the perspective of IO, and vice versa.

Q: To resolve the different view of granularity between MMU and SMMU, **how about creating a new type of IO-mapping dedicated capability?**

A: No, It looks heavy, complex and repetitive.



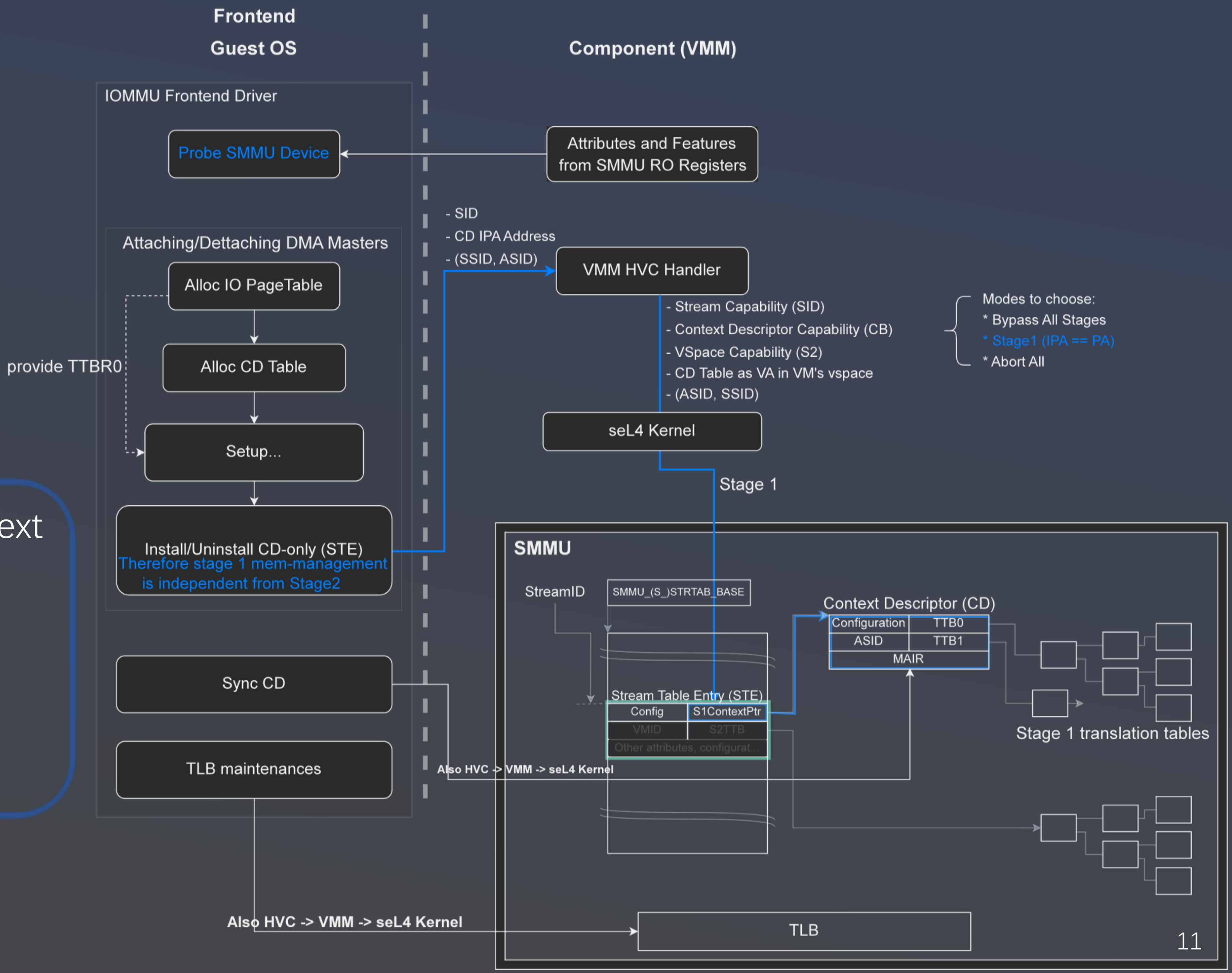
2. Do we need a complex VIRTIO-IOMMU backend for support different Guest OS usage?
A: No, we'd better just choose simple and universal way AARCH64 can help

Technical Challenges

To address the problem, we can use the scheme of Stage1 + Stage2, and the Stage1 construction for IO page tables has been realized within VM.

VMM don’t own/configure the stage 1 context descriptor.

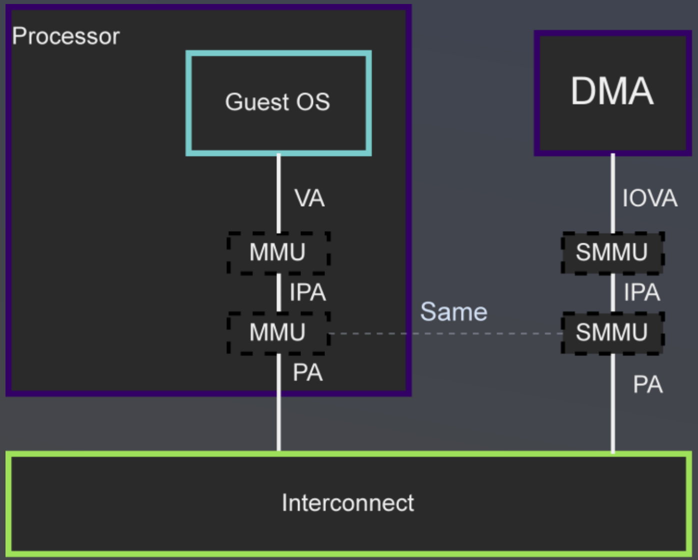
The IOMMU API is extended to convey the guest Stage1 configuration and the hook is implemented in the Guest SMMUv3 driver.

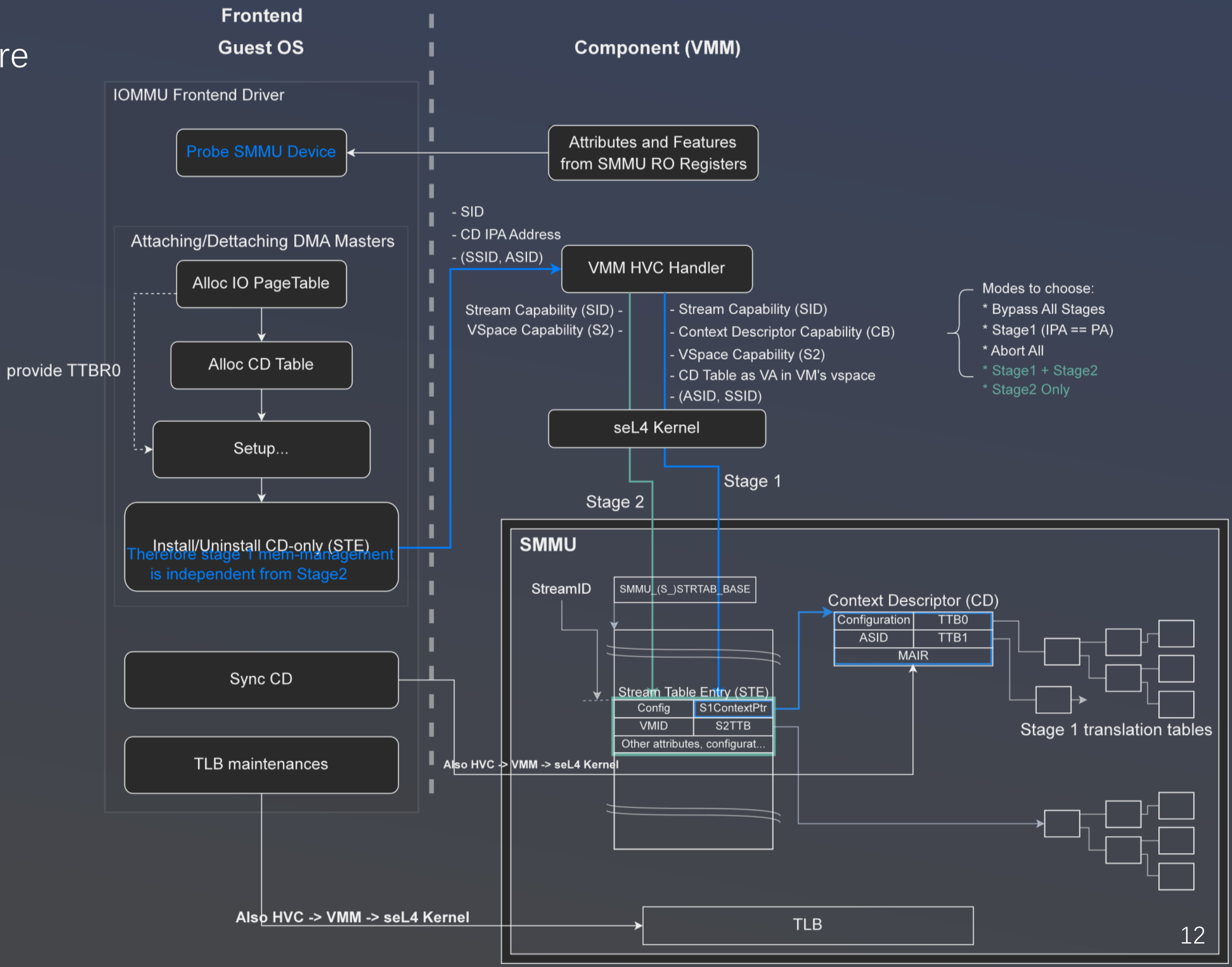


Technical Challenges

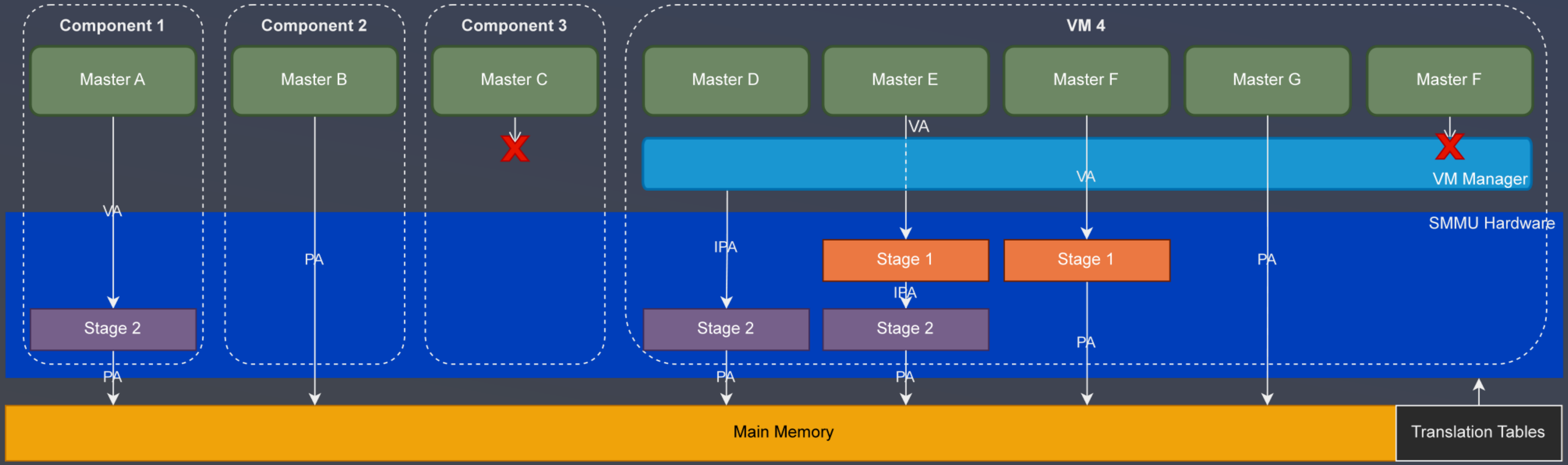
Because the internal paging in two stages are physical independent, this solution makes it easy to enable both S1 and S2 stages.

The addition of Stage2 ensures isolation of multiple virtual machines.





Use examples



Customers can choose above working modes for every master according to their own needs, and both in high-assurance native components or non-critical VMs.

Summary

I. SMMU S1+S2 can be supported without new kernel object and cap.

II. This design can be applied to use all modes(schema) SMMUv3 supported.

III. With S1 supported, Linux native SMMU driver keeps original operation logic, so no need to use heavy QEMU or VIRTIO-IOMMU backend driver.

Future Plans

- CAmkES
  - ➢ Parse device tree for SMMU device nodes and DMA-Master nodes, and auto-generate corresponding SID and CB capabilities, just like the generation of “IRQ Handler” capabilities.
- Do more verification in SMMUv3 builtin SoC Platform
  - ➢ Currently develop and verified in FVP platform. Plan to test real app in our next Journey SoC
  - ➢ Test PCIE devices which use SMMU

THANKS!