Security Level:

SVA：基于异构系统的内存管理技术

www.huawei.com

李泽帆 lizefan@Huawei.com
丁天虹 dingtianhong@Huawei.com

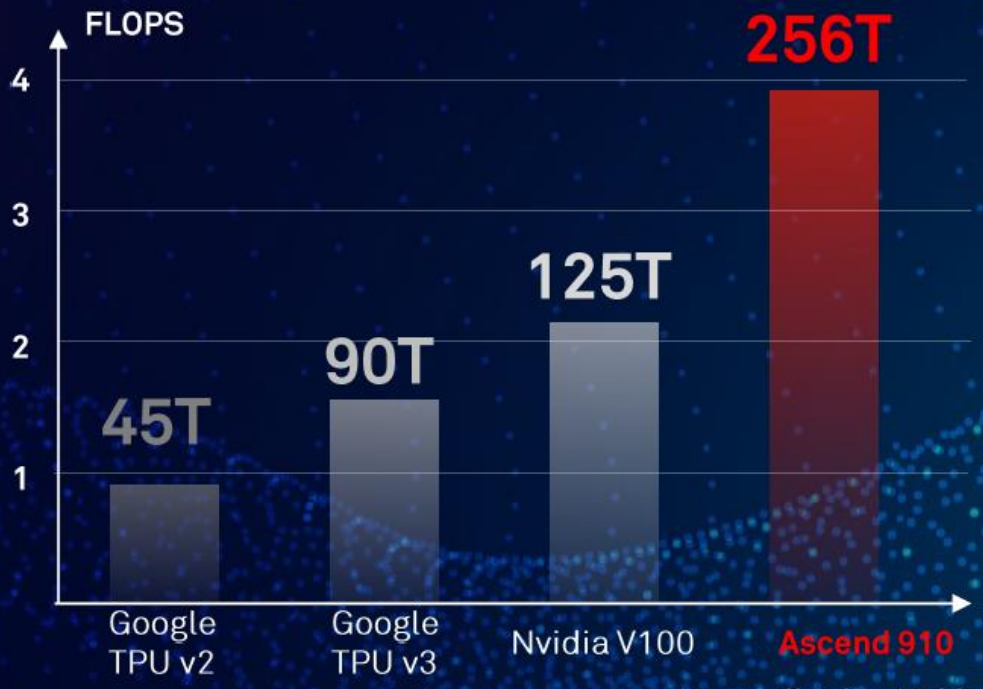
HUAWEI TECHNOLOGIES CO., LTD.

HUAWEI

Ascend 910: Greatest computing density in a single chip
华为昇腾910：单芯片计算密度最大

华为昇腾910

Ascend-Max	Ascend-Max
Architecture: Da Vinci	架构: 达芬奇
Half-Precision (FP16): 256 TeraFLOPS	半精度 (FP16): 256 TeraFLOPS
Integer-Precision (INT8) : 512 TeraOPS	整数精度 (INT8) : 512 TeraOPS
128 Channel FHD Video Decoder – H.264/265	128 通道 全高清 视频解码器 – H.264/265
Max Power: 350W	最大功耗: 350W
7nm	7nm
2019 Q2	2019 Q2

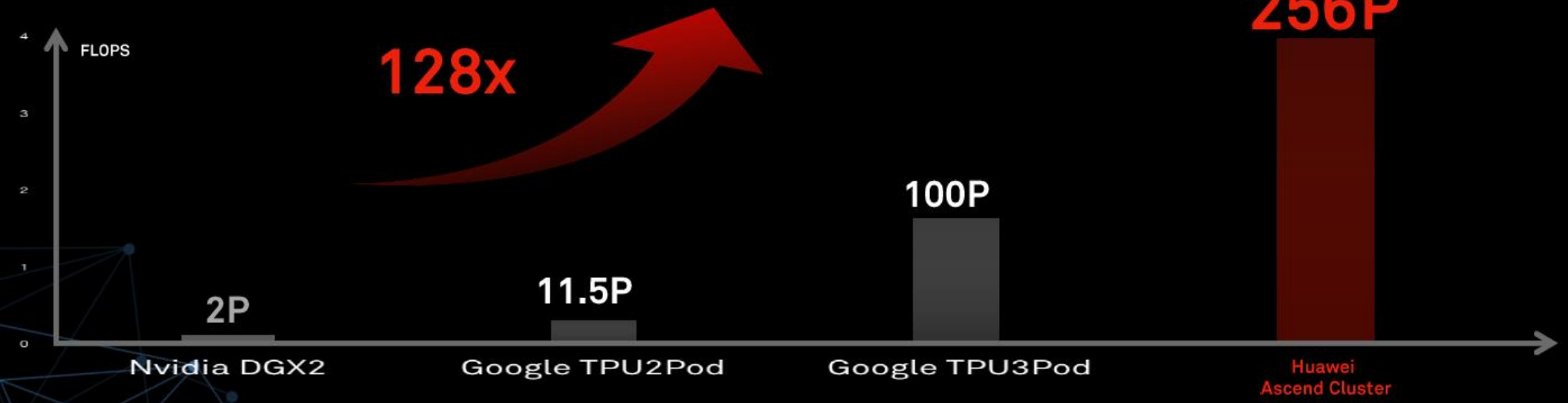


Large-scale distributed training system
大规模分布式训练系统

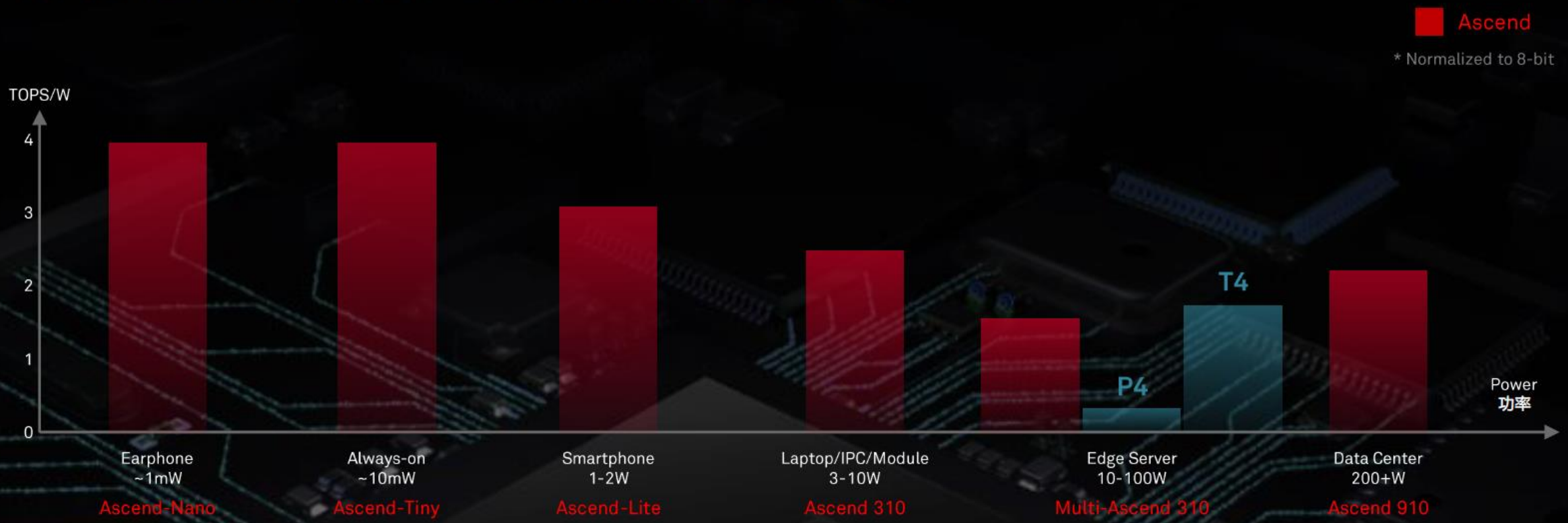
Ascend Cluster

256 PetaFLOPS
HBM: 32 TB
HBM BW: 8 Pbps
Network: 100 Tbps
Linearity:>90%
Parallelism: Data/Model/Hybrid
2019 Q2

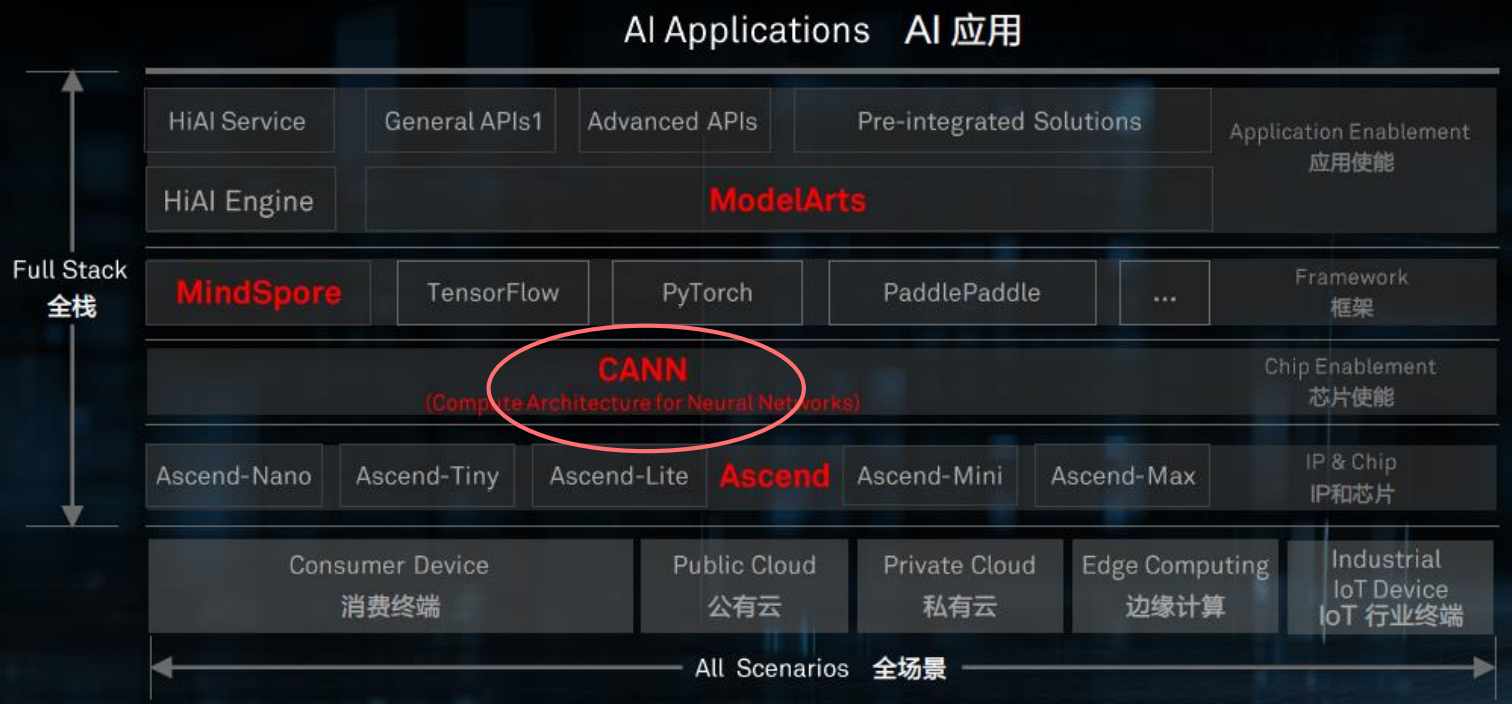
256 PetaFLOPS
HBM: 32 TB
HBM 带宽: 8 Pbps
内部互联网络: 100 Tbps
线性度:>90%
并行模式: 数据/模型/混合
2019 Q2



Ascend: Optimal TOPS/W across all scenarios
Ascend系列，横跨全场景的最优TOPS/W



Huawei's AI portfolio
华为AI解决方案



Application enablement
Full-pipeline services(ModelArts), hierarchical APIs, and pre-integrated solutions
应用使能：
提供全流程服务(ModelArts), 分层API和预集成方案

MindSpore：
Unified training and inference framework for device, edge, and cloud (both standalone and cooperative)
支持端、边、云独立的和协同的统一训练和推理框架

CANN：
Chip operators library and highly automated operators development toolkit
芯片算子库和高度自动化算子开发工具

Ascend：
AI IP and chip series based on unified scalable architecture
基于统一、可扩展架构的系列化AI IP 和 芯片

A unified architecture or not ?
是否统一架构?

One-time Ops development
一次性算子开发

Consistent development & debugging experience
一致的开发和调试体验

Smooth migration across device, edge, and cloud
开发一次，跨端、边和云的平滑迁移

Compute scalability　算力可扩展

- Scale out: Unacceptable power dissipation and area
- Scale in: Complicated scheduling and software
- Scale out: 难以承受的功耗和面积
- Scale in: 复杂的任务调度和软件

Memory wall　内存墙

- Ultra-high bandwidth　超高带宽
- Extremely low latency　极低延时

Interconnection　互联

- Power and area constraints　功率和面积约束

Benefits

Challenges

Agenda

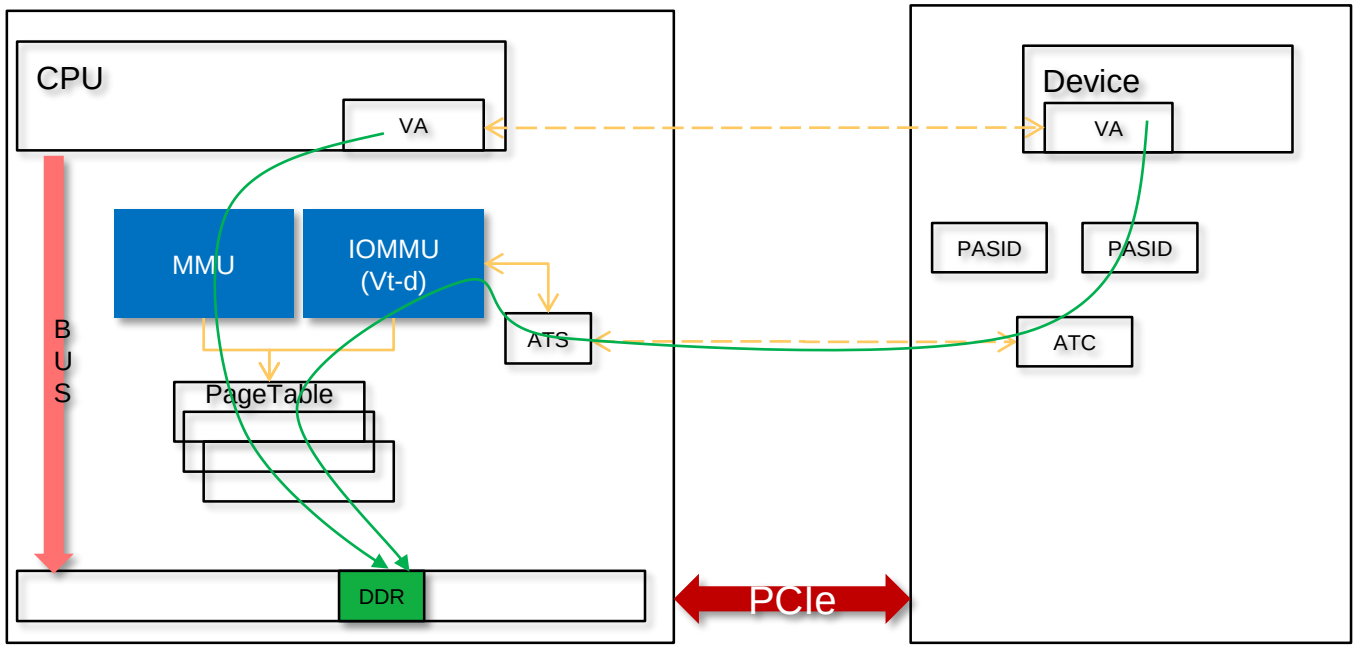
- What is SVA ?
- Why SVA?
- How SVA works?
- Our works
- Upstream status

What is SVA?

- SVA (Shared Virtual Address) means device use the same virtual address with CPU which gets the same thing. But there are various implementations in HW and SW.
  - DMAR (IOMMU, SMMU) or MMU
  - Use the same page table or not
  - Same physical address or not
  - Support zero copy or not
  - Support IO-Page Fault or not

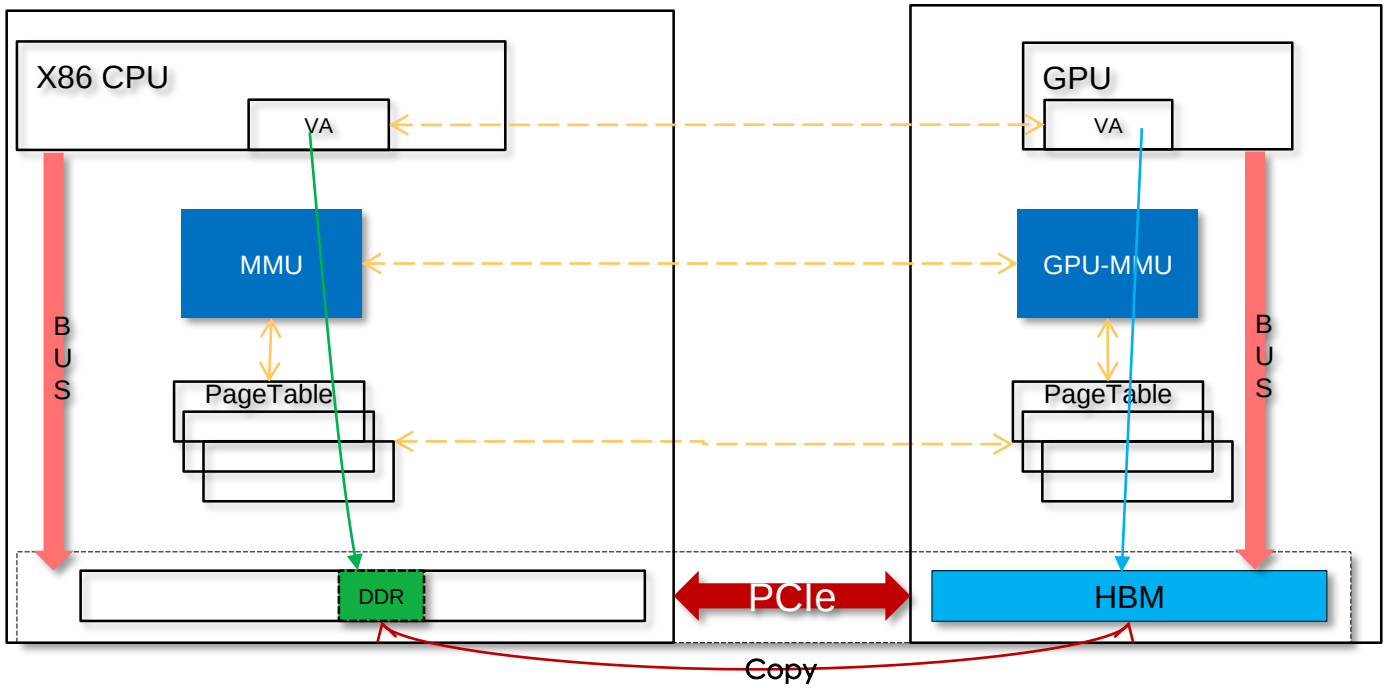
What is SVA?(cont.)

- Intel: (Share Virtual Memory)
  - CPU access DDR through MMU
  - Device access DDR through IOMMU
  - MMU share the same page table with IOMMU
  - (used by AMD’ s Secure Virtual Machine in Linux)



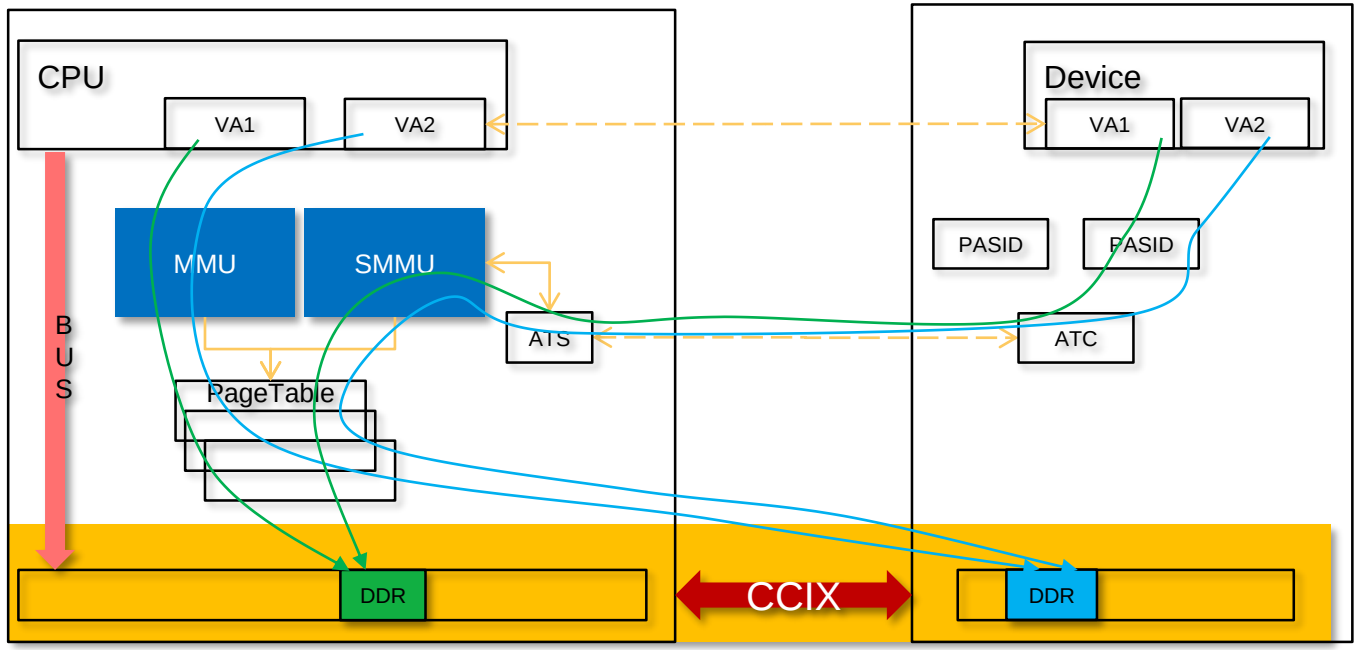
What is SVA?(cont.)

- Nvidia: (UVA: Unified Virtual addressing)
  - GPU has its own MMU
  - CPU can only access DDR while GPU can only access HBM
  - GPU MMU mirror the Page Table of CPU
  - When GPU access a VA not populated in HBM it will copy the data from DDR , and vice versa.



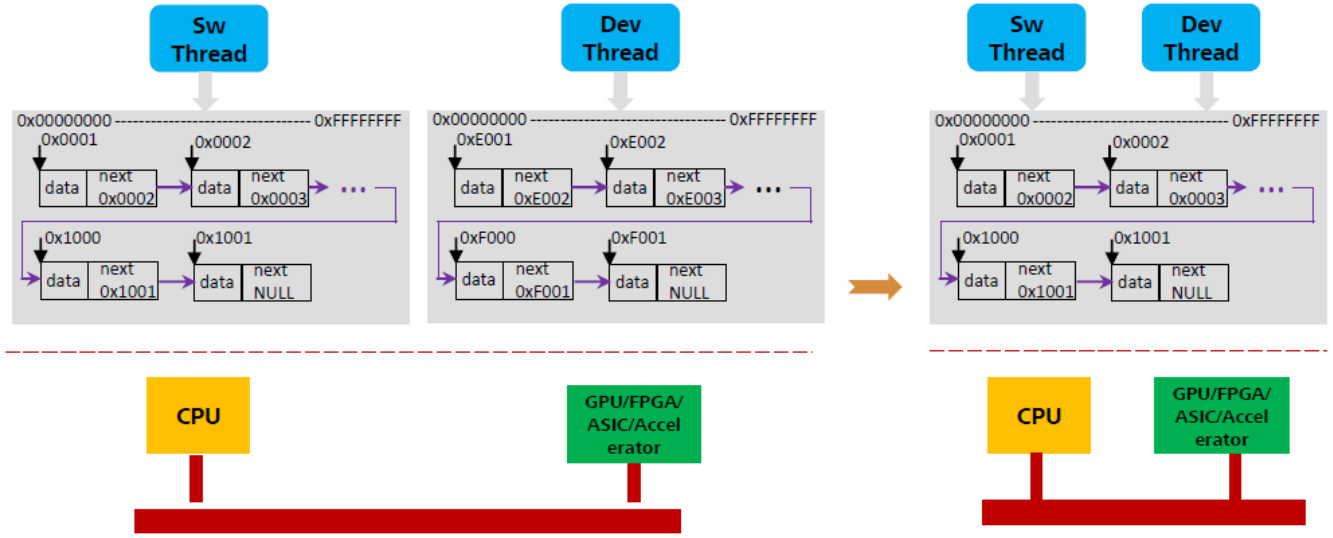
What is SVA?(cont.)

- ARM(SVA: shared virtual address space):
  - Use SMMU instead of IOMMU
  - Device may also have DDR memory
    - With the help of CCIX, both CPU and Device can access Sysmem and Devmem in Cache Coherent way
    - (Device memory management is another story)



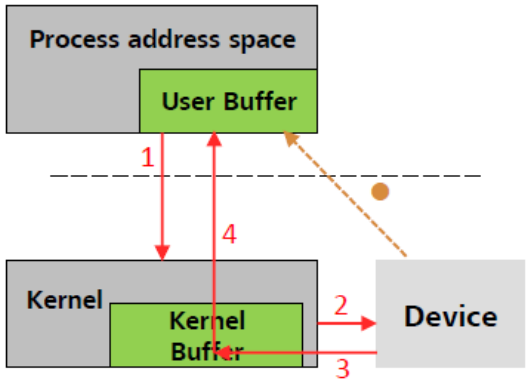
Why SVA?

- Simplify user program
  - Share objects is hard to achieve for complex data-set (list, tree, ...)
  - Zero-copy is easier to achieve (device can access DDR with cc)



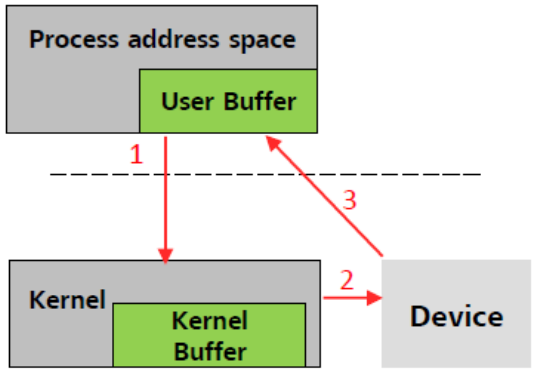
Why SVA?(cont)

- Device side on-demand paging(based on IO page fault)
  - Cons:
    - No need to pin memory – decrease pin memory overhead
    - Allow memory overcommit – usefully for low-end production
  - Cons:
    - Performance overhead of page fault for high speed device



1. Malloc() buffer, syscall or ioctl to kernel
2. Kmalloc() buffer request device initial dma
3. Device DMAs to/from kernel buffer
4. Kernel copies to user buffer

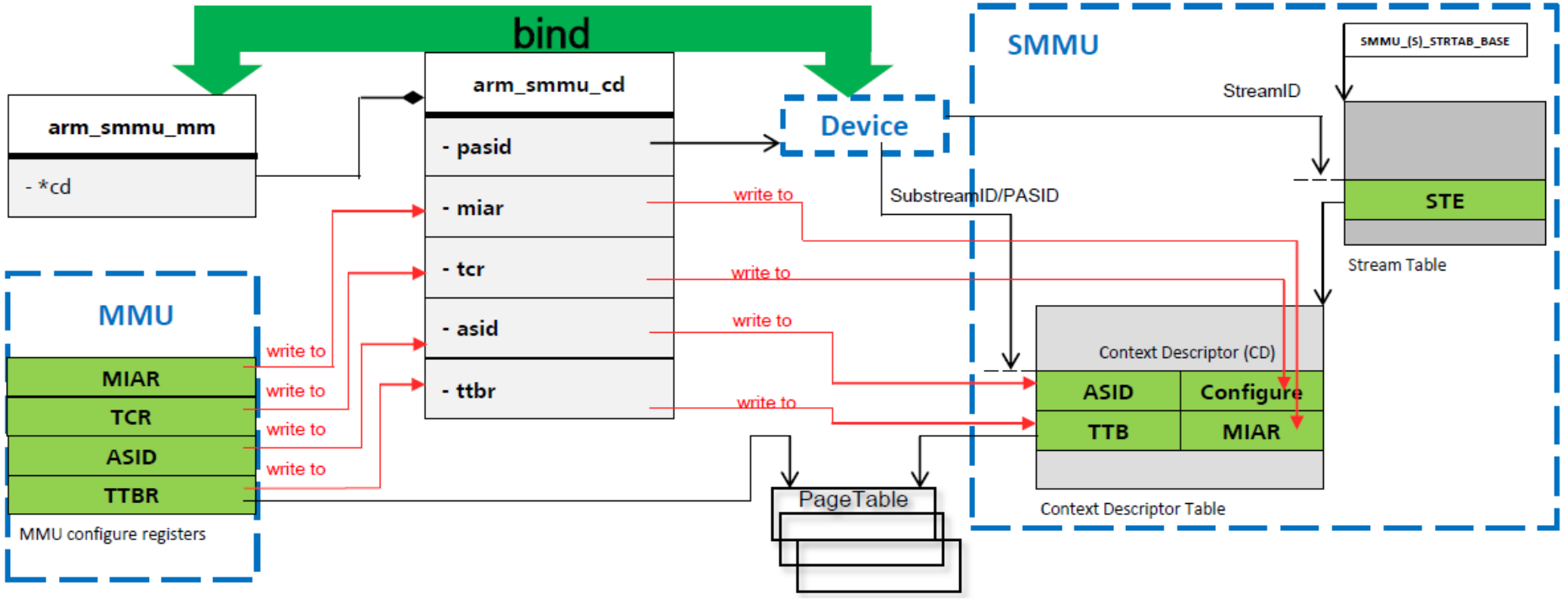
● Device DMA to user buffer(pin....)



1. Malloc() buffer, syscall or ioctl to kernel
2. Request device to initiate DMA
3. Device DMA access process address directly. It trigger interrupt to CPU when access non-populated Virtual Address.

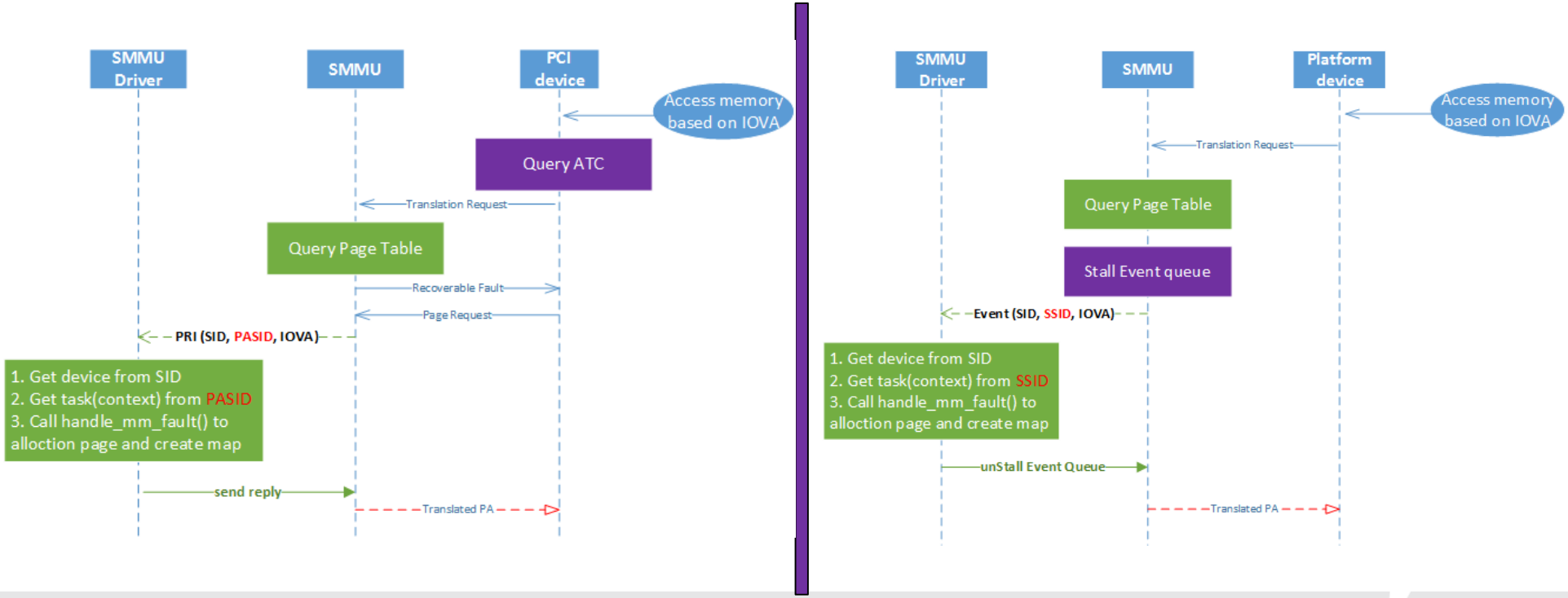
How SVA works?

- ARM- SMMUv3: Bind a task to device to share page table
  - arm_smmu_mm: address space abstraction in smmu
  - arm_smmu_cd: smmu context descriptor instant
    - When task is bound to device, its values are got from task



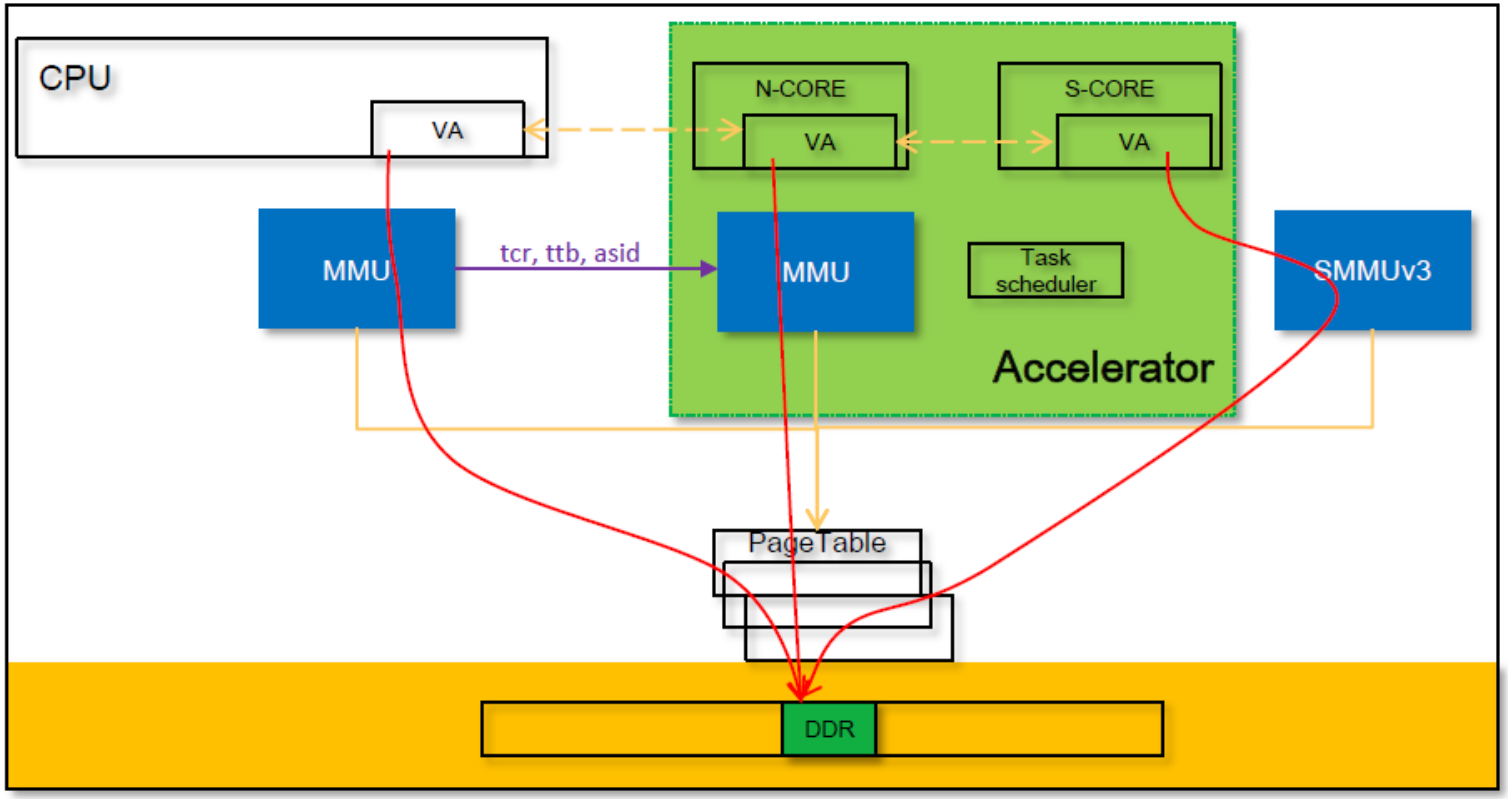
How SVA works ? (cont.)

- ARM- IO Pagefault: PRI Queue vs Event Queue
  - PCI devices use PRI Queue/PASID
  - Platform devices use Event Queue/SSID (Stall-mode)

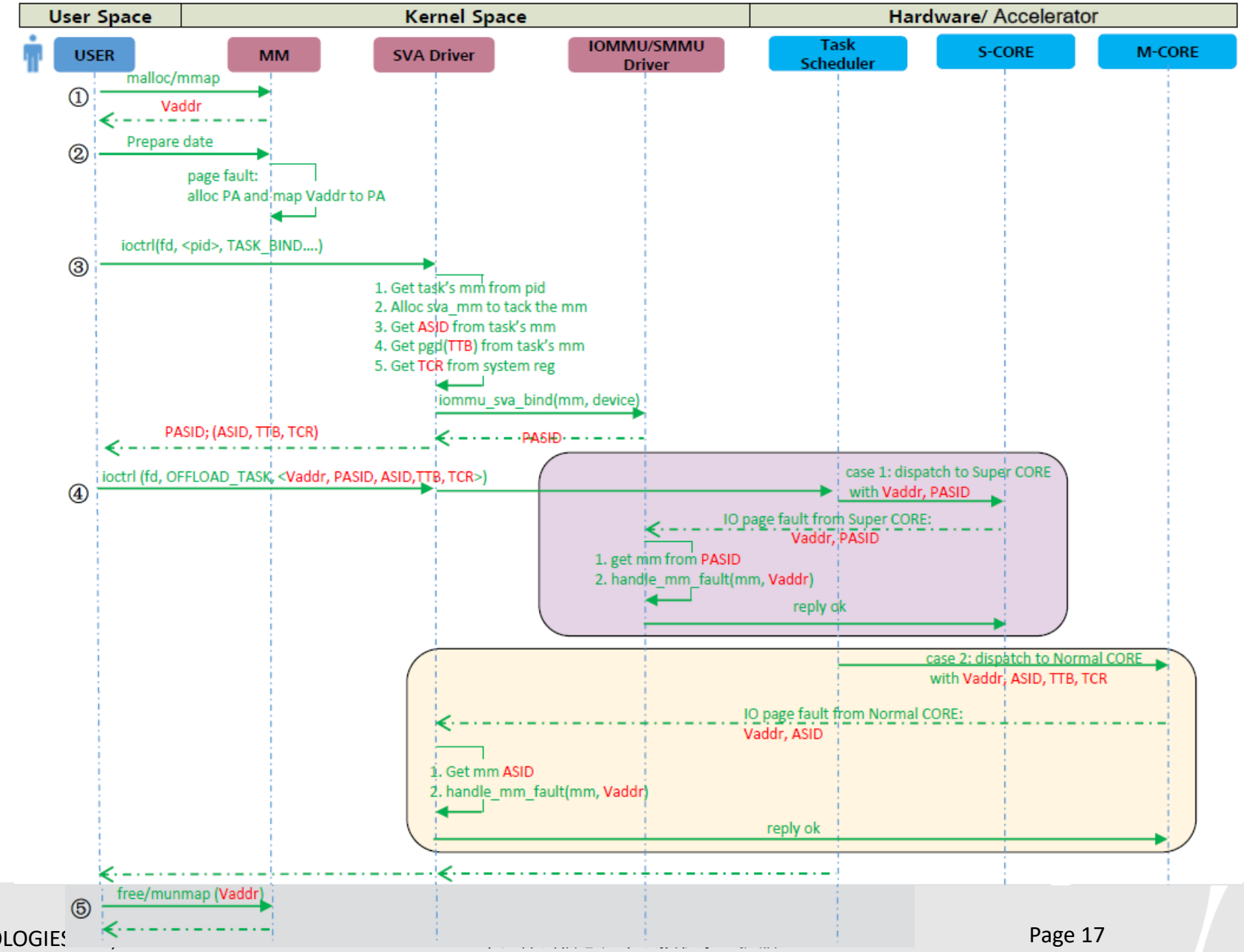


Our Works

- A SVA implement case:
  - M-COREs and Super COREs in on-chip device
  - S-COREs use SMMUv3 whichsworks in stall mode
  - M-COREs use MMU instead



Our Works (cont.)



Upstream status

- SVA for SMMUv3
  - RFC1: [RFC PATCH 00/30] Add PCIe SVM support to ARM SMMUv3
    - Only works for PCIE pri
  - RFC2: [RFCv2 PATCH 00/36] Process management for IOMMU + SVM for SMMUv3
    - Add support of stall mode for platform device
  - V1/V2: [PATCH v2 00/40] Shared Virtual Addressing for the IOMMU
    - Only support devices which have IO-Page Fault ability.

欢迎加入华为OS内核实验室

华为操作系统部：华为端、管、云核心的OS软件基础设施

OS Kernel Lab

- Linux内核（ARM/x86/异构等）的技术研发与创新
- 低时延、高安全、高可靠、高智能的下一代OS内核技术的研究和成果转化

招聘岗位

下一代操作系统研究员/高级工程师

形式化技术研究员/高级工程师

Linux内核架构师/高级工程师

工作地

杭州、北京、上海

简历投递

Tel：王先生/18658102676

Email：hr.kernel@huawei.com

Thank you

www.huawei.com