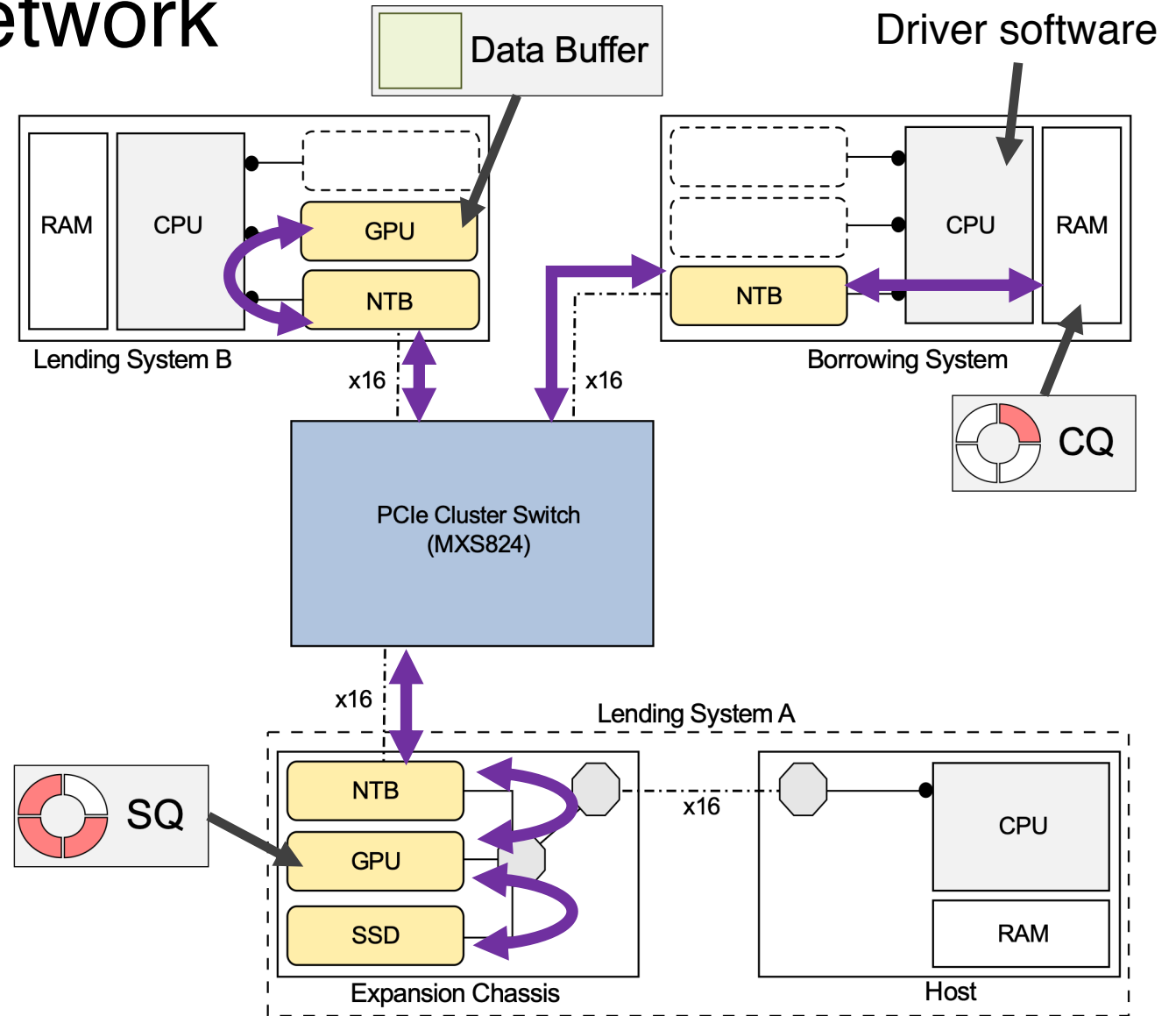


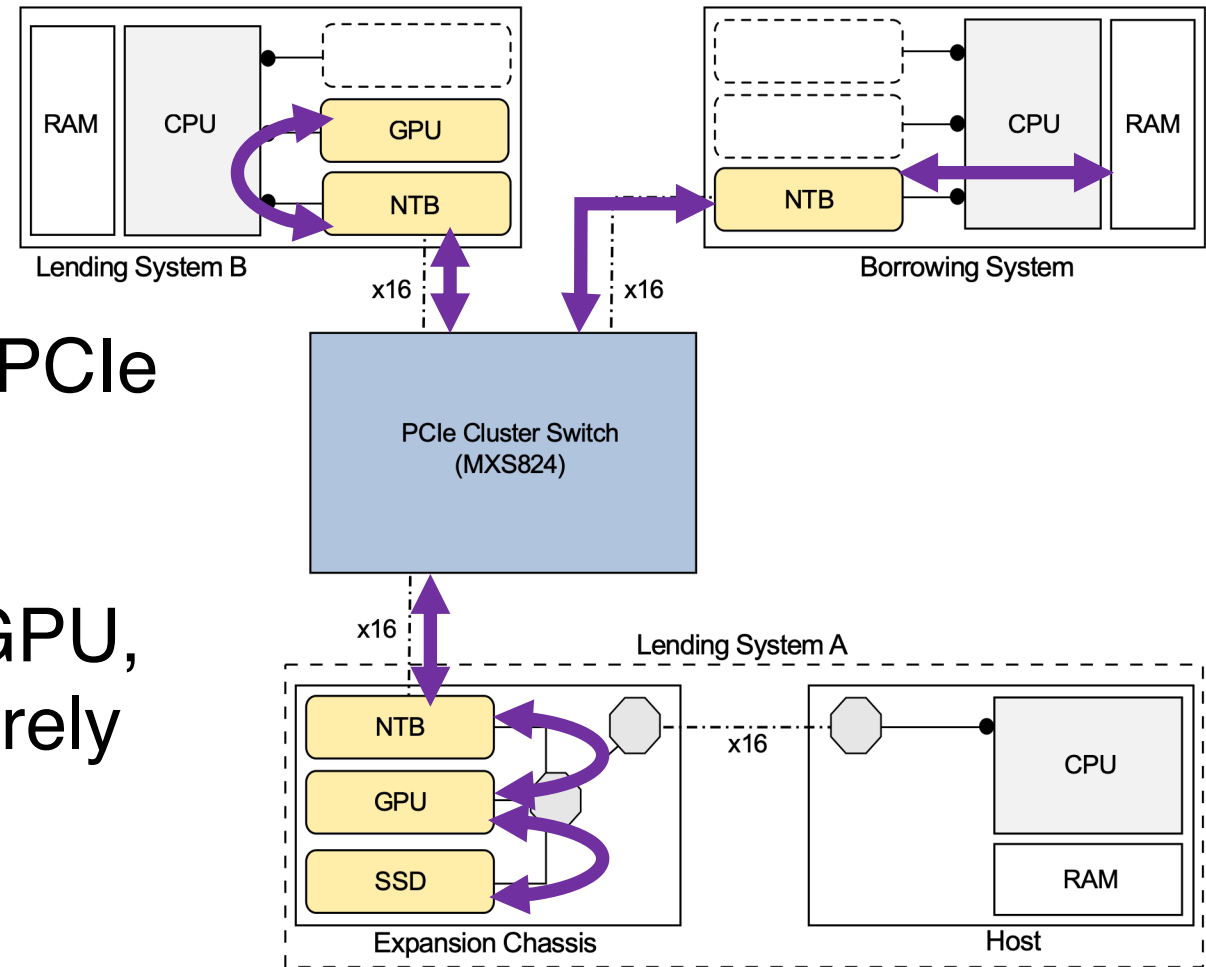
S9563: Efficient Distributed Storage I/O using NVMe and GPUDirect in a PCIe Network

Jonas Markussen
Software Architect and PhD Student
Dolphin Interconnect Solutions



Summary

- Optimized data transfer paths using PCIe peer-to-peer transactions
- Direct block-level disk access from GPU, eliminating CPU in I/O data path entirely
- Concurrently sharing NVMe drives between multiple hosts and GPUs
- PCIe non-transparent bridges offer great flexibility in dynamic device configurations

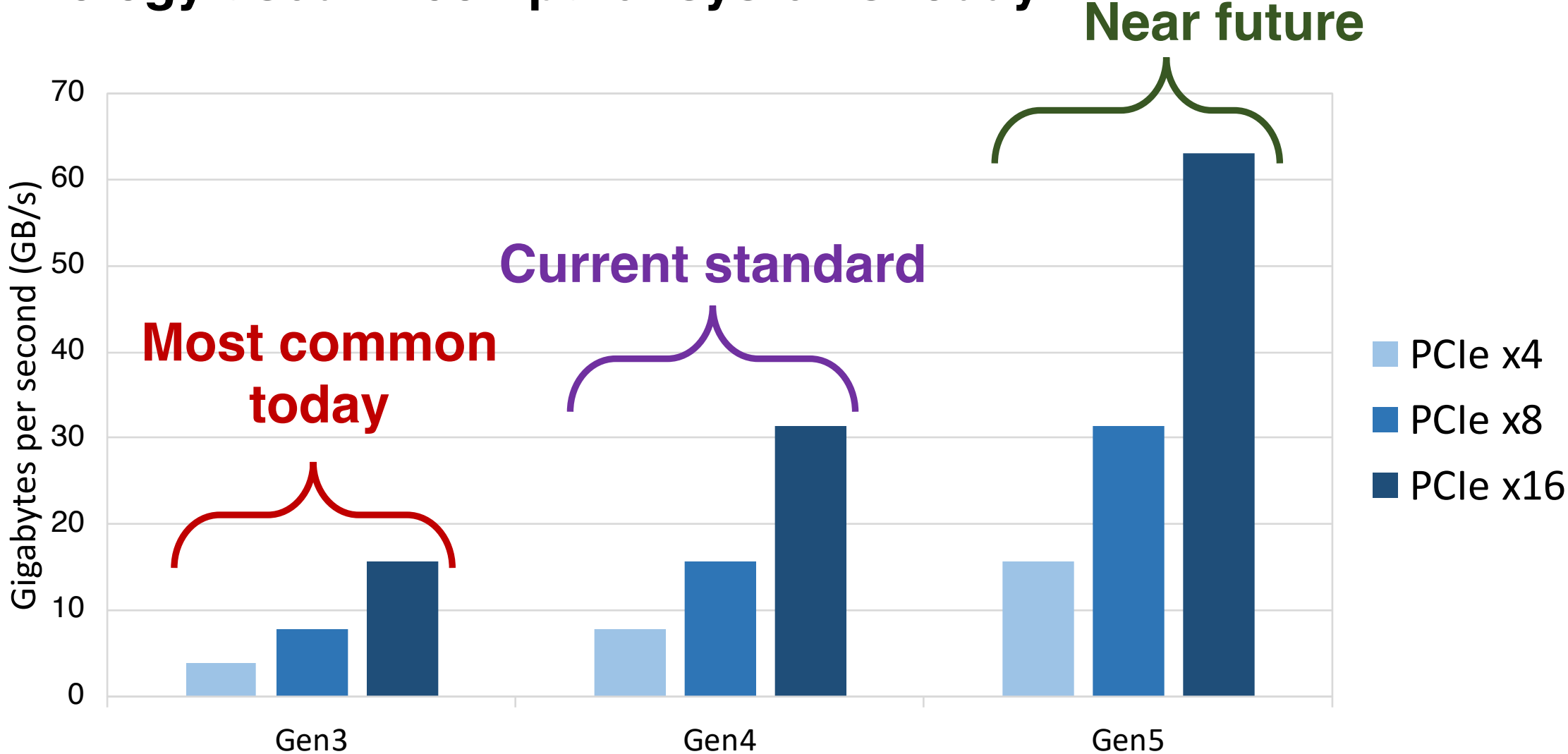


Outline

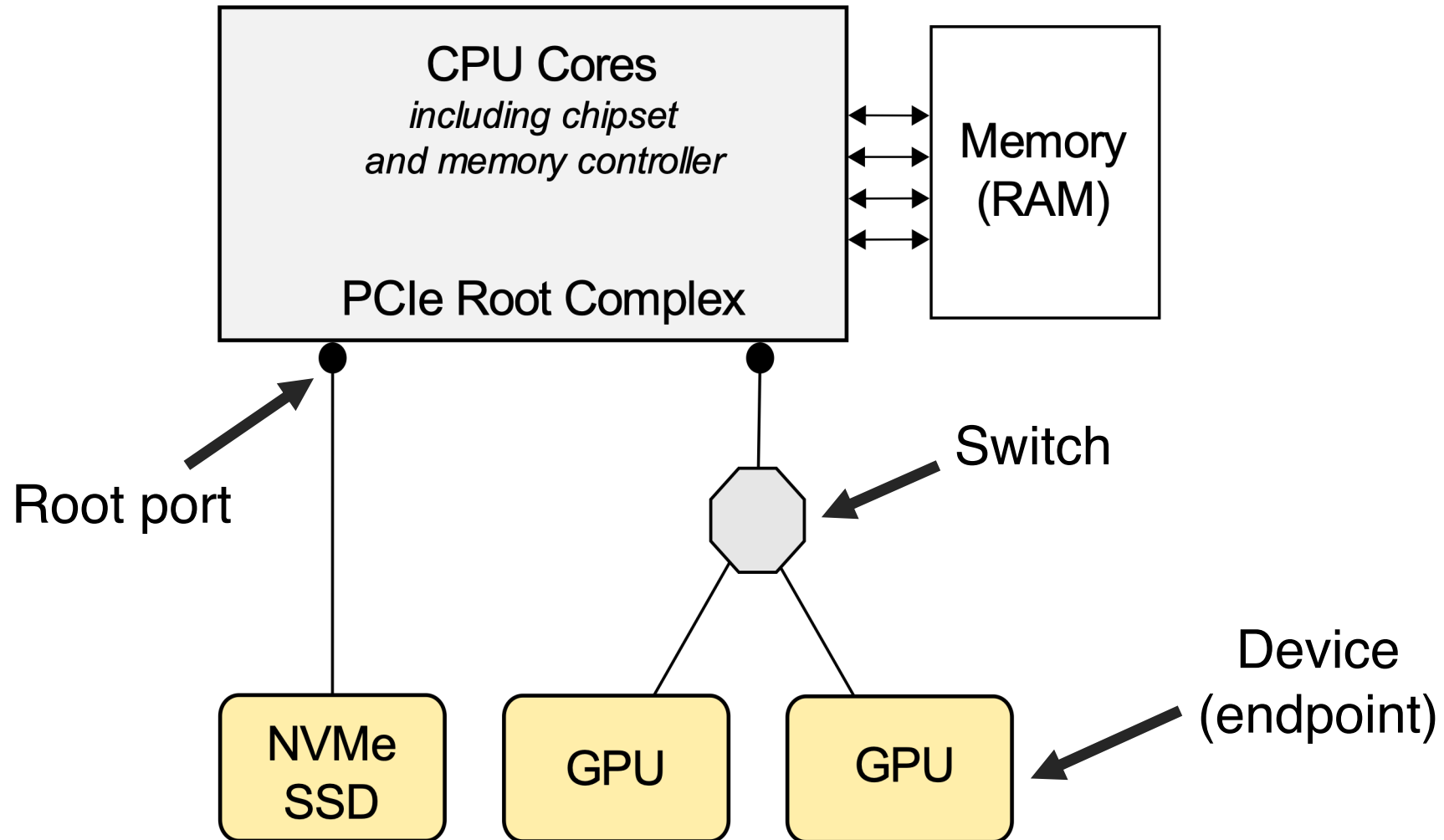
- PCIe and NVMe
- Non-Transparent Bridging
- GPUDirect RDMA & Async
- Device Lending and SmartIO

PCI Express (PCIe)

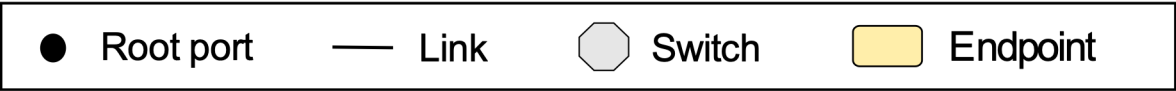
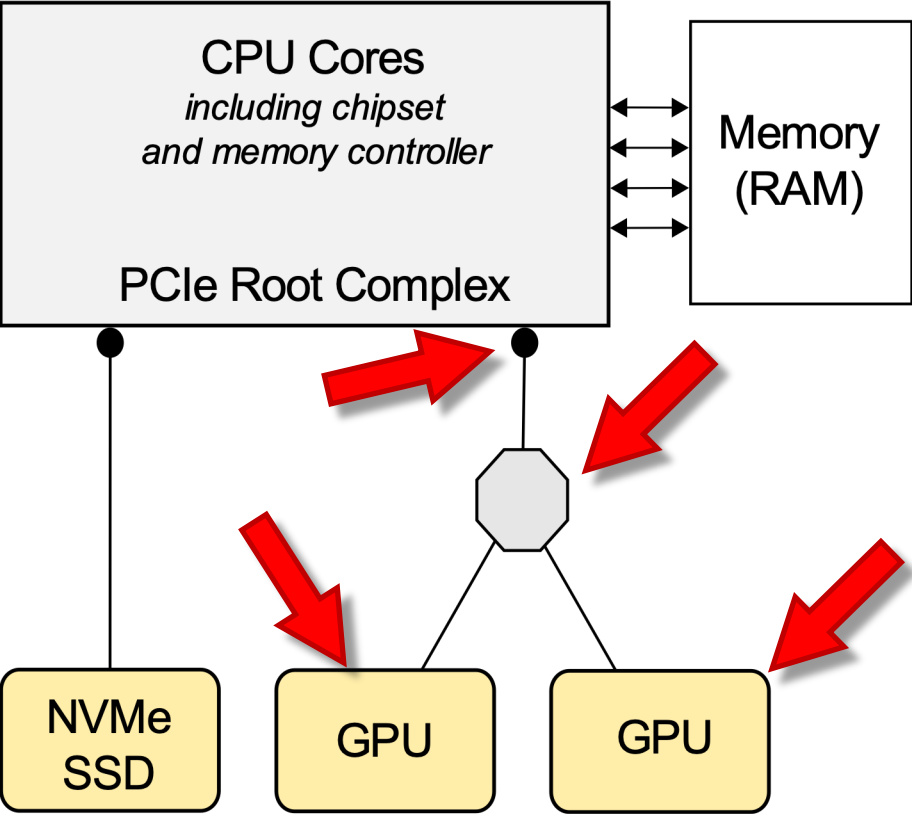
PCI Express (PCIe) is the most widely adopted I/O interconnection technology used in computer systems today



The PCIe fabric is structured as a tree, where devices form the leaf nodes (endpoints) and the CPU is on top of the root



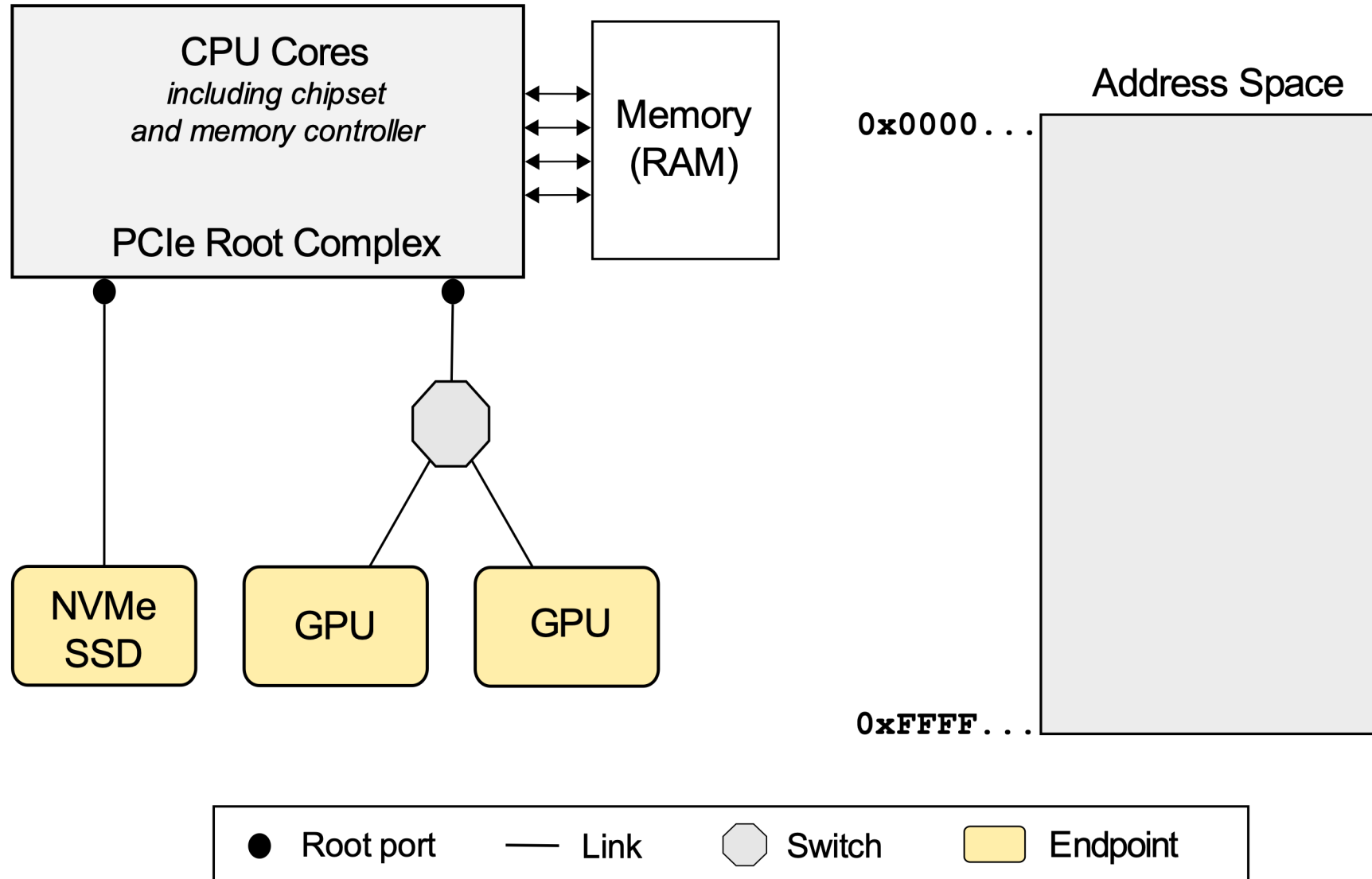
The PCIe fabric is structured as a tree, where devices form the leaf nodes (endpoints) and the CPU is on top of the root



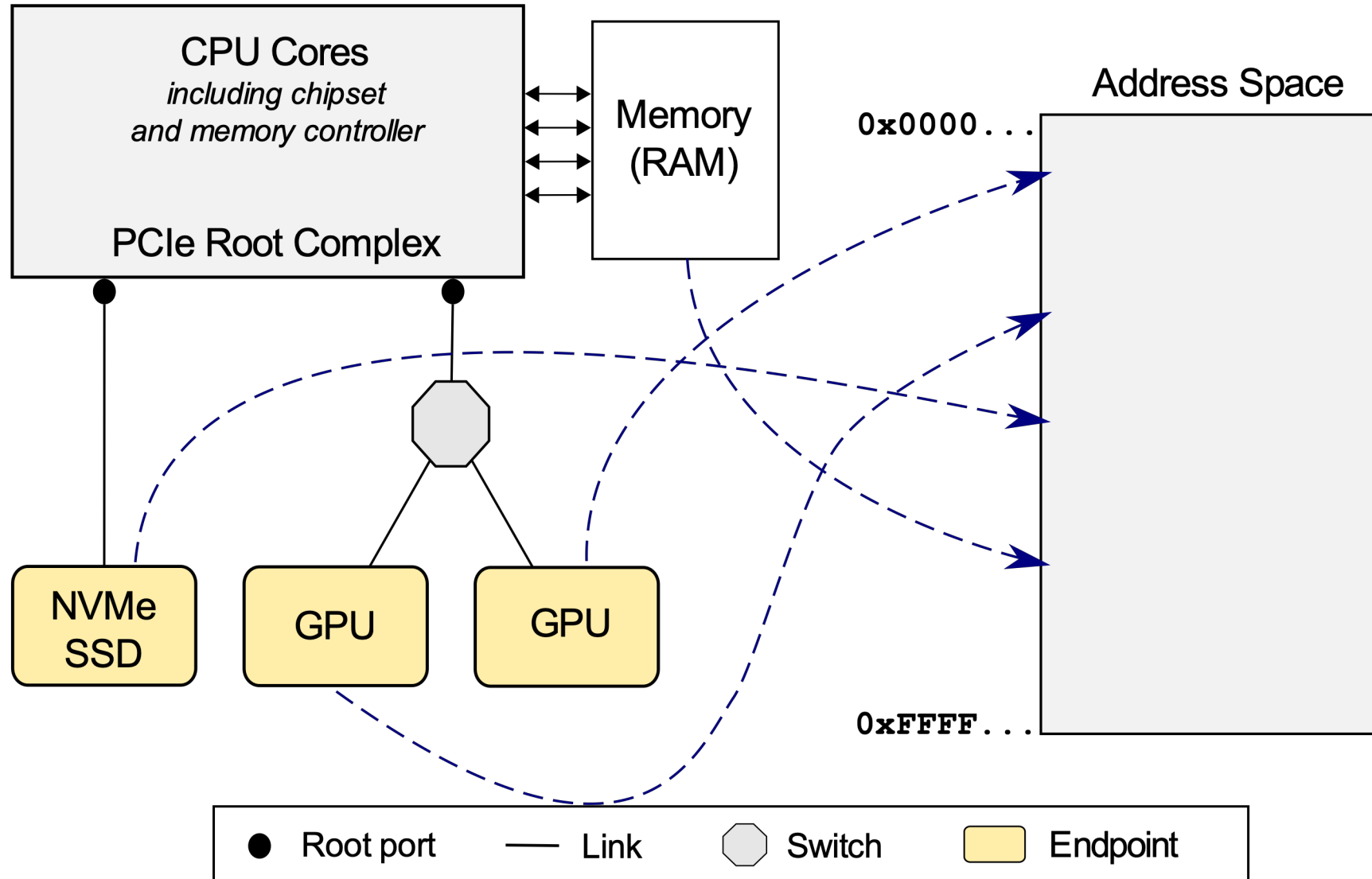
\$ lspci -tv

```
dolphin@xde: ~ (ssh)
|
|--1e.4 Intel Corporation Xeon E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
|--1f.0 Intel Corporation Xeon E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
|--1f.2 Intel Corporation Xeon E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
\-[0000:00]--+-00.0 Intel Corporation Xeon E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D DMI2
              +-01.0-[01]----00.0 Samsung Electronics Co Ltd NVMe SSD Controller SM961/PM961
              +-02.0-[02]---+00.0 Intel Corporation Xeon Processor D Family QuickData Technology Register DMA Channel 0
                  |
                  |   +-00.1 Intel Corporation Xeon Processor D Family QuickData Technology Register DMA Channel 1
                  |   +-00.2 Intel Corporation Xeon Processor D Family QuickData Technology Register DMA Channel 2
                  |   \-00.3 Intel Corporation Xeon Processor D Family QuickData Technology Register DMA Channel 3
              +-02.2-[03]---+00.0 Intel Corporation Ethernet Connection X552/X557-AT 10GBASE-T
                  |
                  |   \-00.1 Intel Corporation Ethernet Connection X552/X557-AT 10GBASE-T
              +-03.0-[04-0e]---+00.0-[05-0e]---00.0-[06-0e]---+00.0-[07-0e]---00.0-[08-0e]---00.0-[09-0e]---00.0-[0a]---
                  |
                  |   +-08.0-[0b]---
                  |   +-0c.0-[0c]---
                  |   +-00.0 NVIDIA Corporation GK107GL [Quadro]
                  |   \-00.1 NVIDIA Corporation GK107 HDMI Audio
                  |
                  |   +-10.0-[0d]---
                  |   \-14.0-[0e]---
                  |
                  |   \-00.1 PMC-Sierra Inc. Device 8532
              \-00.1 PMC-Sierra Inc. Device 8532
|--05.0 Intel Corporation Xeon E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D Map/VTd/Misc/System Management
|--05.1 Intel Corporation Xeon E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D IIO Hot Plug
|--05.2 Intel Corporation Xeon E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D IIO RAS/Control Status/Global Errors
```

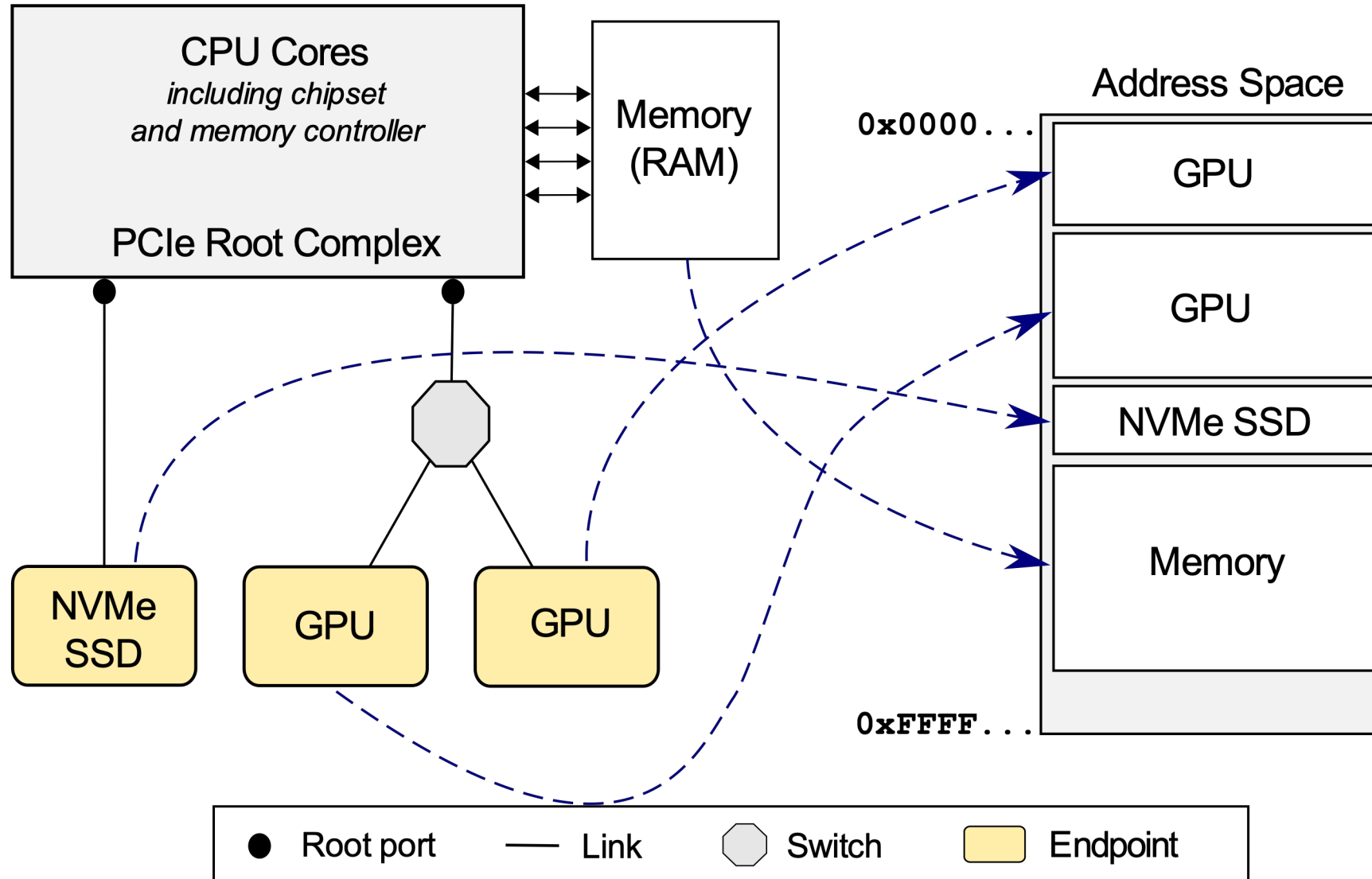
PCIe devices are mapped in to the same address space as CPUs, allowing devices to access system memory directly (DMA)



PCIe devices are mapped in to the same address space as CPUs, allowing devices to access system memory directly (DMA)



PCIe devices are mapped in to the same address space as CPUs, allowing devices to access system memory directly (DMA)

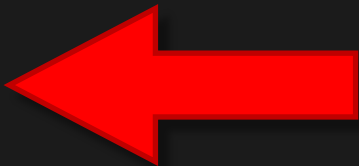


PCIe devices are mapped in to the same address space as CPUs, allowing devices to access system memory directly (DMA)

```
$ lspci -s XX:XX.X -v
```

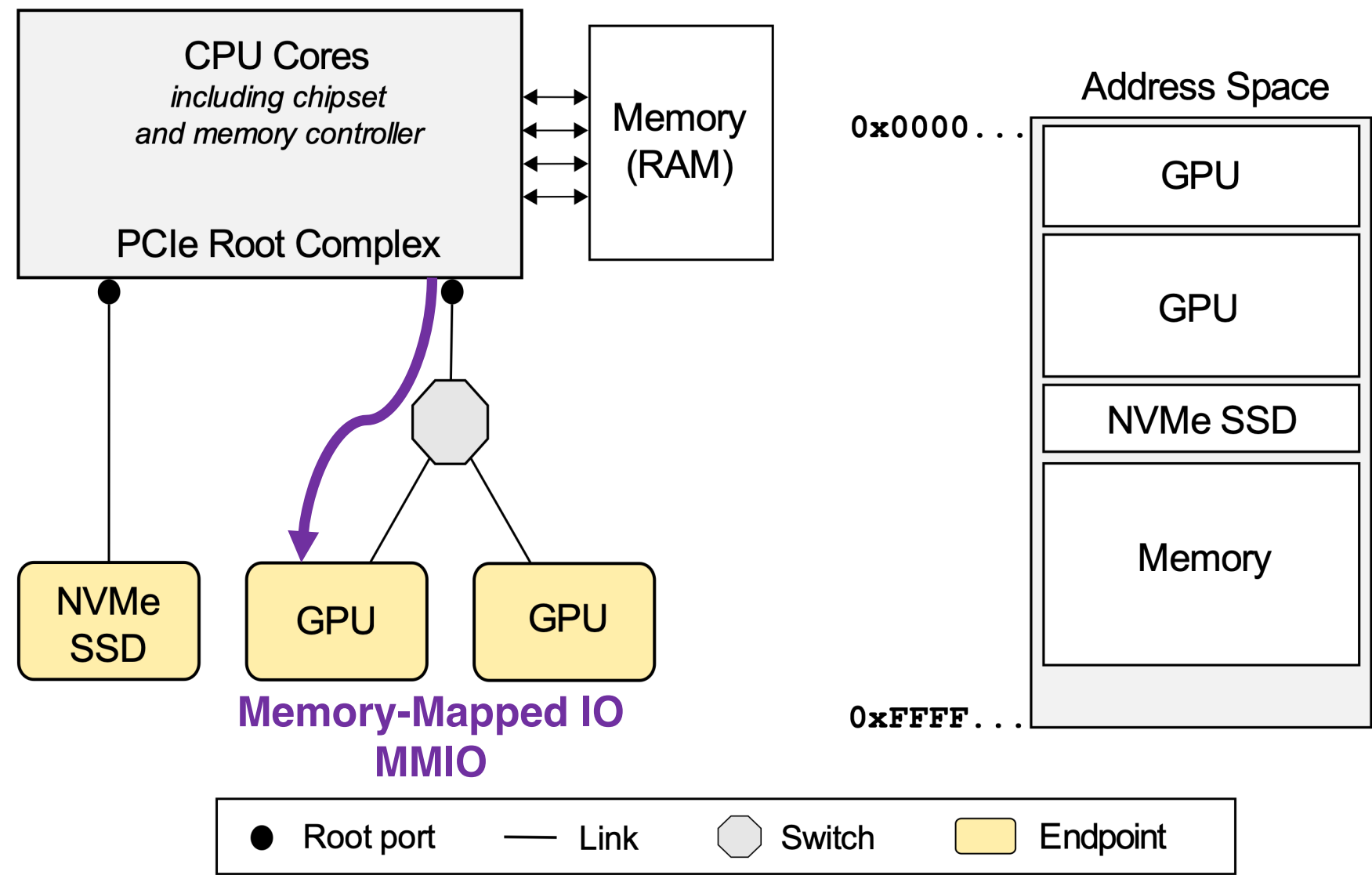
```
root@petty:~ (ssh)
[root@petty ~]#
[root@petty ~]# lspci -s 4:3.1 -v
04:03.1 VGA compatible controller: NVIDIA Corporation GP107GL [Quadro P620] (rev a1) (prog-if 00 [VGA controller])
    Subsystem: NVIDIA Corporation Device 1264
    Flags: bus master, fast devsel, latency 0, IRQ 71, NUMA node 0
    Memory at 3830e5000000 (32-bit, non-prefetchable) [size=16M]
    Memory at 3830f0000000 (32-bit, non-prefetchable) [size=256M]
    Memory at 383100000000 (32-bit, non-prefetchable) [size=32M]
    Expansion ROM at 3830e6000000 [disabled] [size=512K]
    Capabilities: [60] Power Management version 3
    Capabilities: [68] MSI: Enable- Count=1/1 Maskable- 64bit+
    Capabilities: [78] Express Legacy Endpoint, MSI 00
    Capabilities: [100] Virtual Channel
    Capabilities: [250] Latency Tolerance Reporting
    Capabilities: [128] Power Budgeting <?>
    Capabilities: [420] Advanced Error Reporting
    Capabilities: [600] Vendor Specific Information: ID=0001 Rev=1 Len=024 <?>
    Capabilities: [900] #19
    Kernel driver in use: nvidia
    Kernel modules: nouveau, nvidia_drm, nvidia

[root@petty ~]#
```

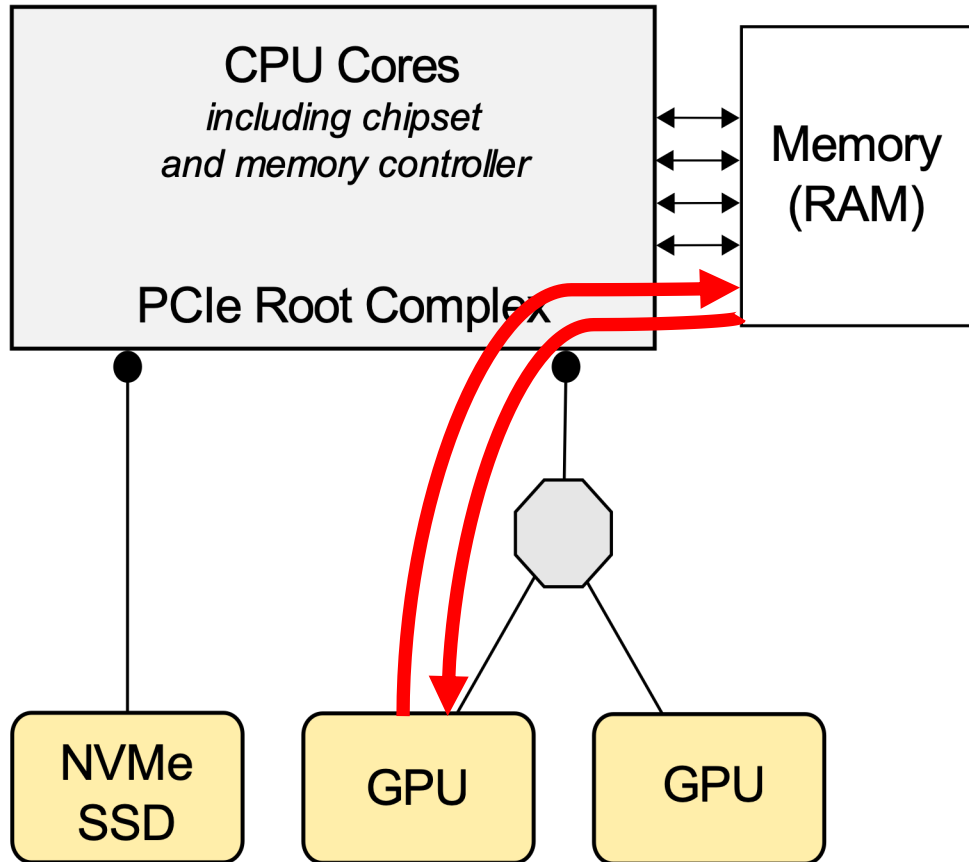


Base Address Regions
(BARs)

PCIe devices are mapped in to the same address space as CPUs, allowing devices to access system memory directly (DMA)



PCIe devices are mapped in to the same address space as CPUs, allowing devices to access system memory directly (DMA)



**DMA Read
(non-posted)**

\$./bandwidthTest

```
dolphin@xde: ~ (ssh)
dolphin@xde:~$ /usr/local/cuda/extras/demo_suite/bandwidthTest
[CUDA Bandwidth Test] - Starting...
Running on...

Device 0: Quadro K420
Quick Mode

Host to Device Bandwidth, 1 Device(s)
PINNED Memory Transfers
Transfer Size (Bytes)      Bandwidth(MB/s)
33554432                   6419.6

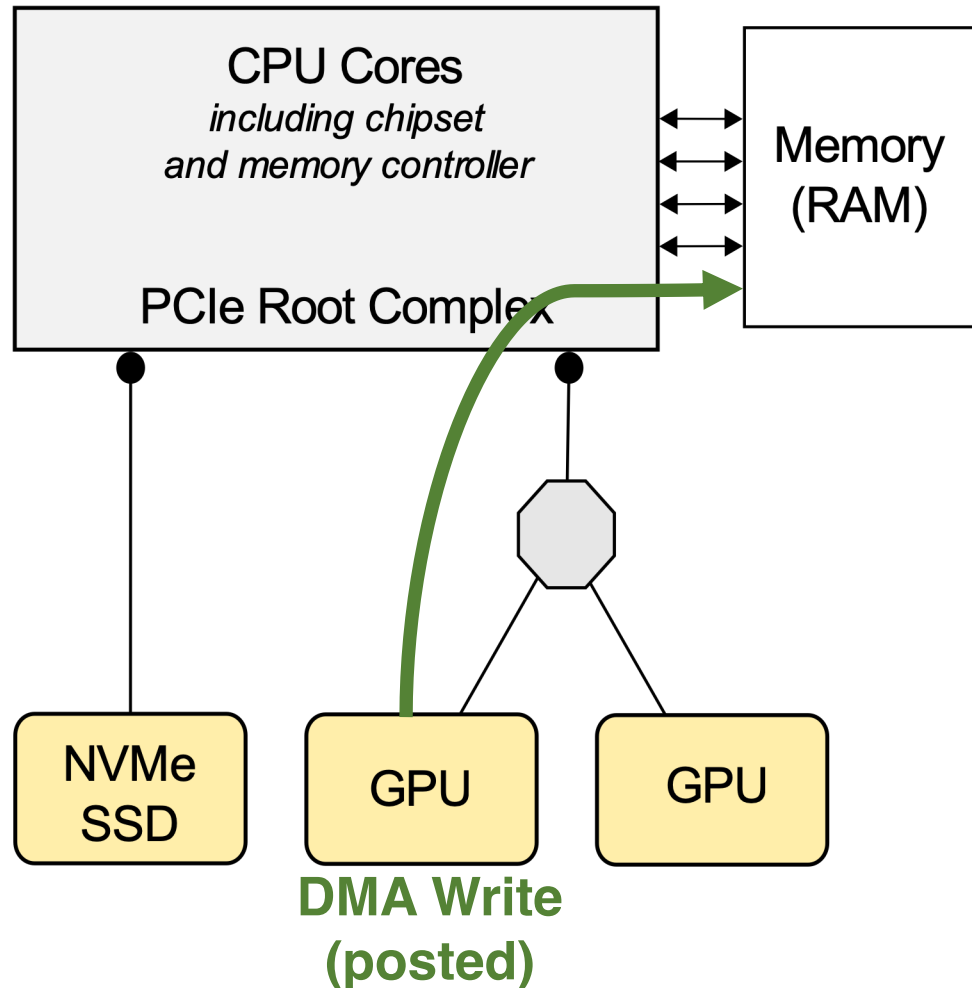
Device to Host Bandwidth, 1 Device(s)
PINNED Memory Transfers
Transfer Size (Bytes)      Bandwidth(MB/s)
33554432                   6553.5

Device to Device Bandwidth, 1 Device(s)
PINNED Memory Transfers
Transfer Size (Bytes)      Bandwidth(MB/s)
33554432                   20963.0

Result = PASS

NOTE: The CUDA Samples are not meant for performance measurements. Results
may vary when GPU Boost is enabled.
dolphin@xde:~$
```

PCIe devices are mapped in to the same address space as CPUs, allowing devices to access system memory directly (DMA)



\$./bandwidthTest

```
dolphin@xde: ~ (ssh)
dolphin@xde:~$ /usr/local/cuda/extras/demo_suite/bandwidthTest
[CUDA Bandwidth Test] - Starting...
Running on...

Device 0: Quadro K420
Quick Mode

Host to Device Bandwidth, 1 Device(s)
PINNED Memory Transfers
Transfer Size (Bytes)      Bandwidth(MB/s)
33554432                   6419.6

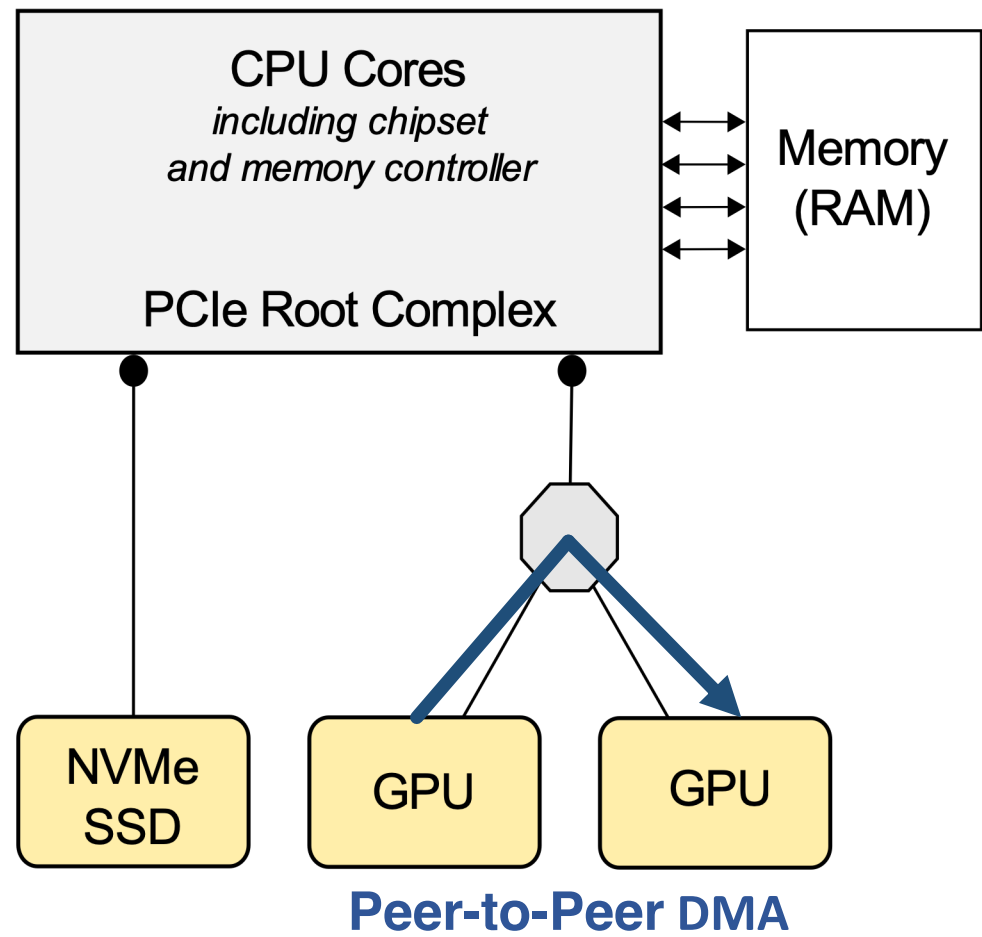
Device to Host Bandwidth, 1 Device(s)
PINNED Memory Transfers
Transfer Size (Bytes)      Bandwidth(MB/s)
33554432                   6553.5

Device to Device Bandwidth, 1 Device(s)
PINNED Memory Transfers
Transfer Size (Bytes)      Bandwidth(MB/s)
33554432                   20963.0

Result = PASS

NOTE: The CUDA Samples are not meant for performance measurements. Results
may vary when GPU Boost is enabled.
dolphin@xde:~$
```

As device memory is mapped in to the same address space by the system, devices can also access other devices' memory



```
$ ./p2pBandwidthLatencyTest
```

```
root@xde: /usr/local/cuda-10.1/samples/1_Uilities/p2pBandwidthLatencyTest (ssh)
```

```
P2P Connectivity Matrix
```

D\D	0	1
0	1	1
1	1	1

```
Unidirectional P2P=Disabled Bandwidth Matrix (GB/s)
```

D\D	0	1
0	21.44	5.76
1	5.78	21.45

```
Unidirectional P2P=Enabled Bandwidth (P2P Writes) Matrix (GB/s)
```

D\D	0	1
0	21.44	6.71
1	6.71	21.45

```
Unidirectional P2P=Disabled Bandwidth Matrix (GB/s)
```

D\D	0	1
0	21.47	5.77
1	5.78	21.48

```
Bidirectional P2P=Enabled Bandwidth Matrix (GB/s)
```

D\D	0	1
0	21.47	13.23
1	13.20	21.48

```
P2P=Disabled Latency Matrix (us)
```

GPU	0	1
0	6.35	24.79
1	26.33	6.34

```
CPU
```

	0	1
0	3.28	7.62
1	7.59	3.26

```
P2P=Enabled Latency (P2P Writes) Matrix (us)
```

GPU	0	1
0	6.39	2.05
1	2.00	6.42

```
CPU
```

	0	1
0	3.38	2.51
1	2.49	3.36

NOTE: The CUDA Samples are not meant for performance measurements. Results may vary when GPU Boost is enabled.

```
root@xde: /usr/local/cuda-10.1/samples/1_Uilities/p2pBandwidthLatencyTest#
```

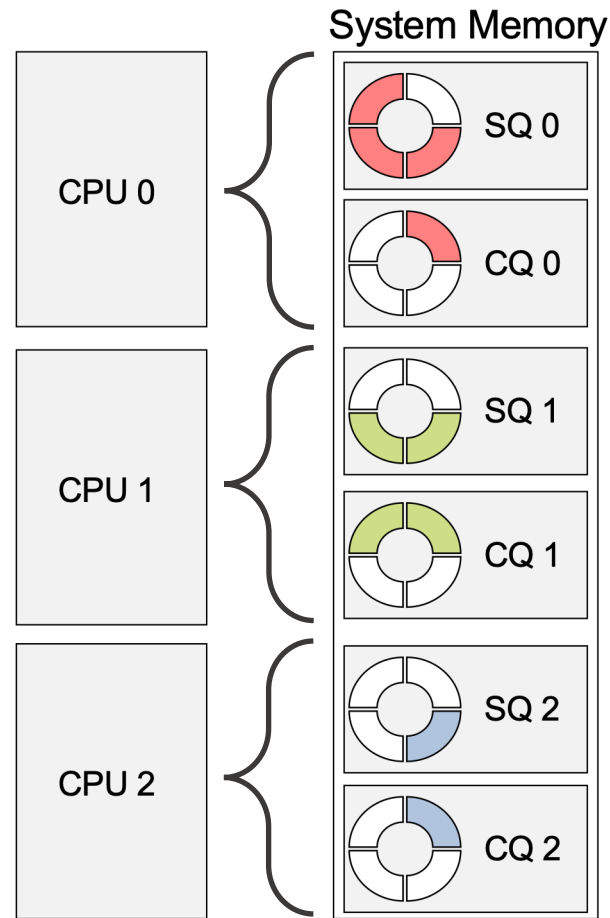
PCIe Summary

- Devices share address space with the CPU and are able to access memory = DMA
- Memory reads and writes are forwarded shortest path on the PCIe fabric
- Devices can access memory on other devices = peer-to-peer DMA

NVM Express (NVMe)

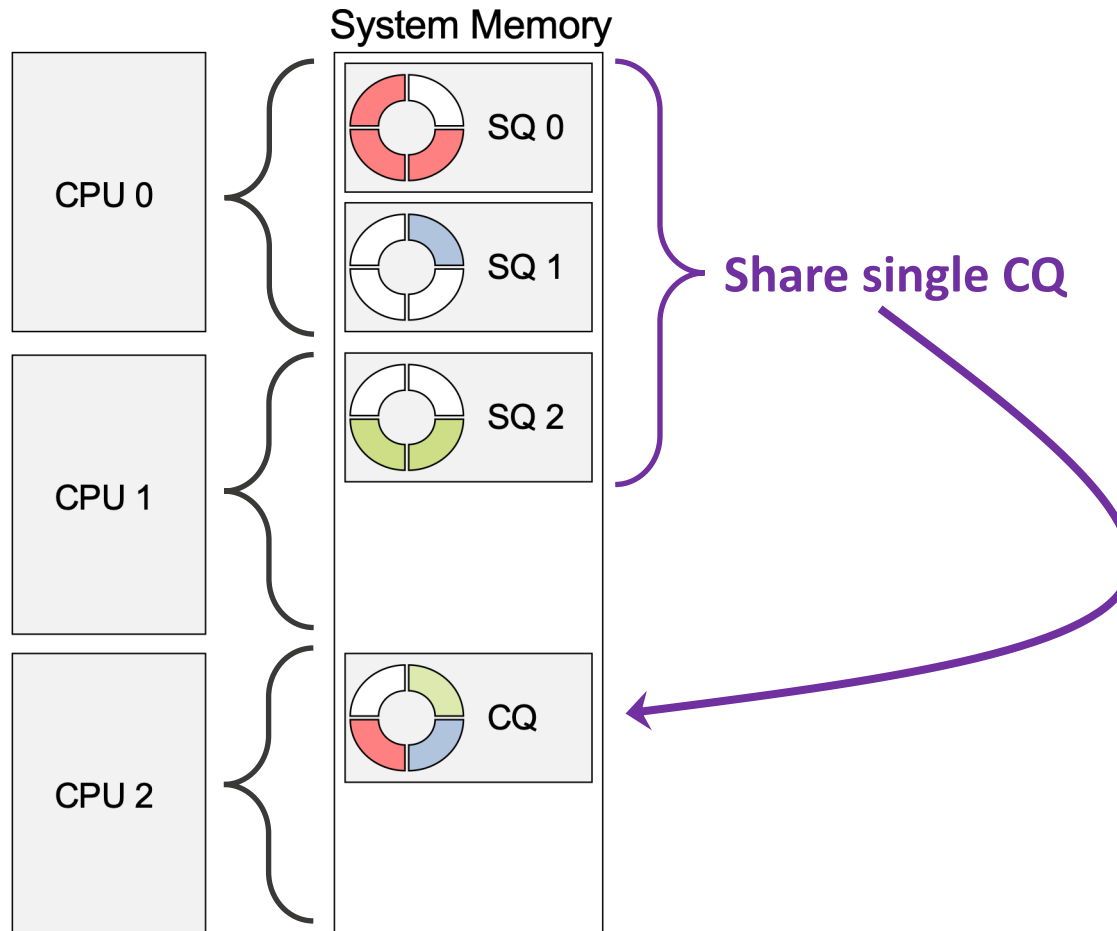


NVMe is designed around multiple parallel I/O command submission queues (SQs) and command completion queues (CQs)



One SQ and one CQ per CPU (1:1)

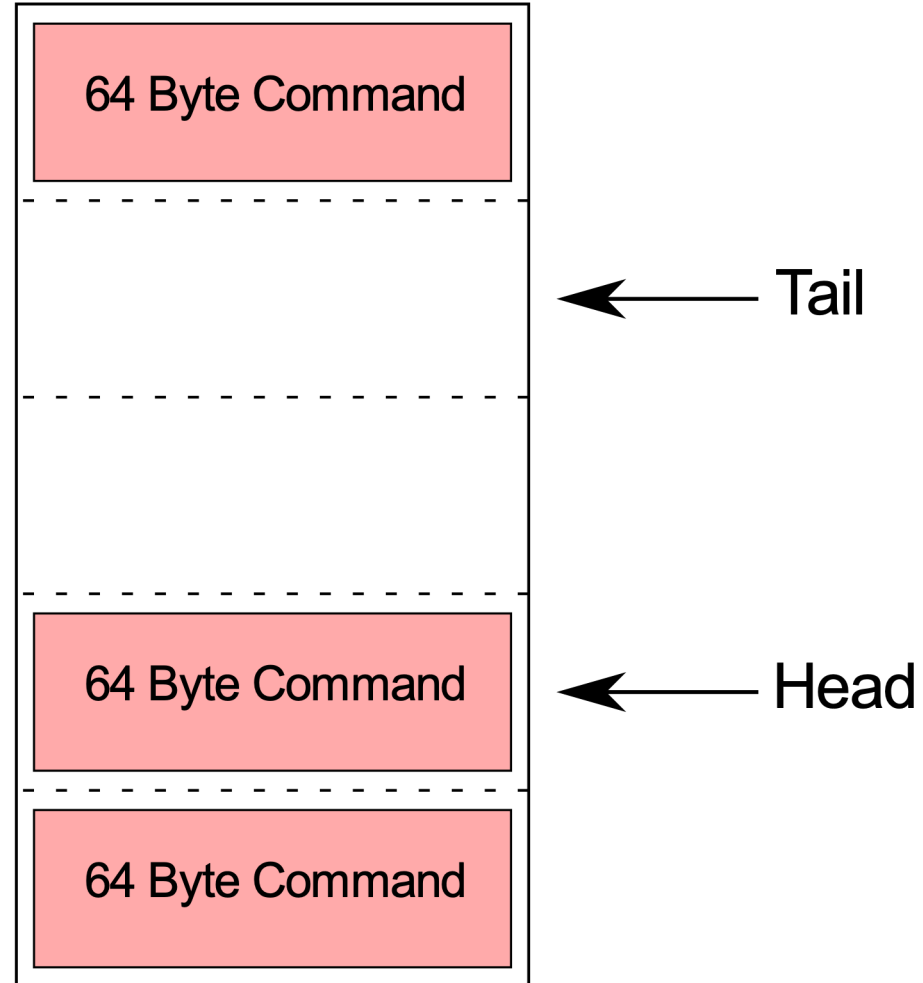
NVMe is designed around multiple parallel I/O command submission queues (SQs) and command completion queues (CQs)



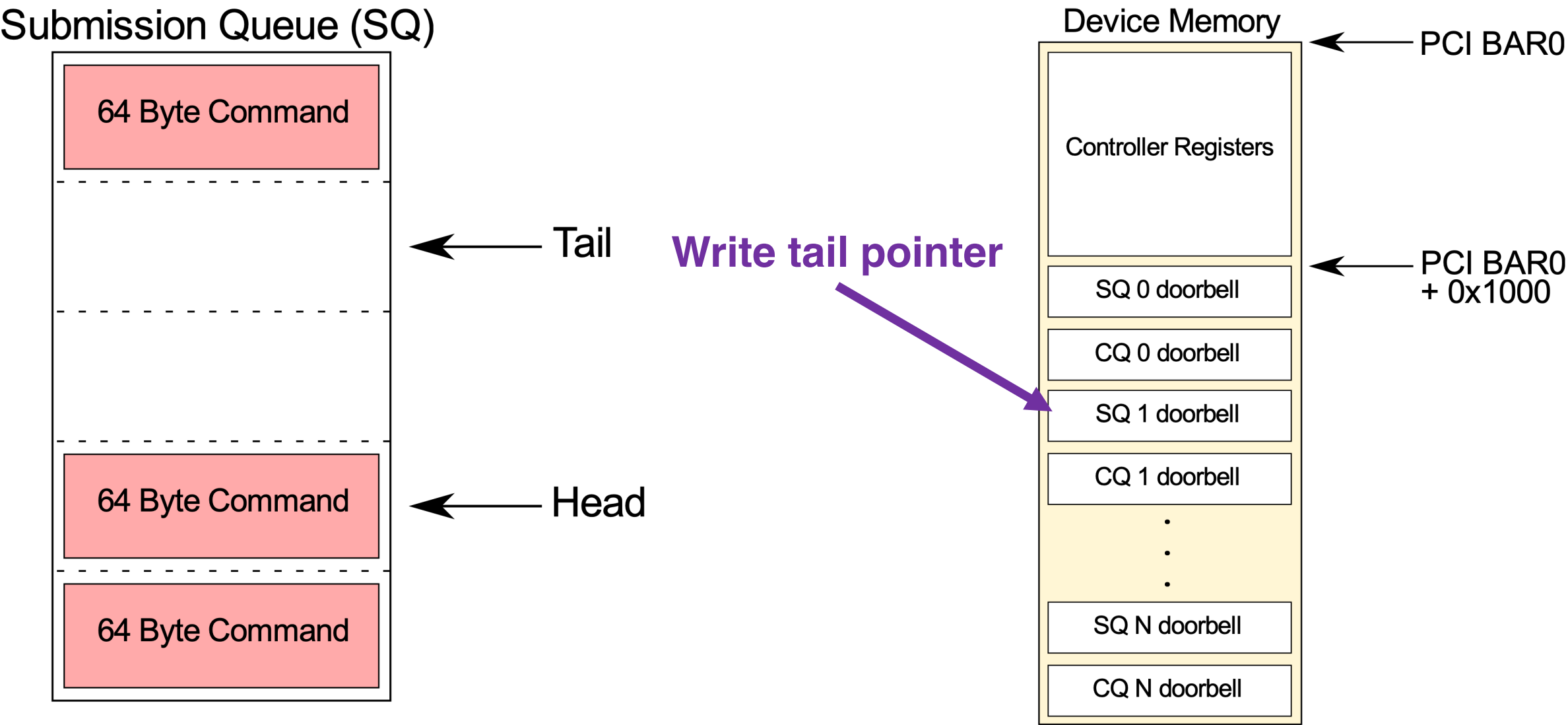
Multiple SQs sharing single CQ (N:M)

Command queues are implemented as ring-buffers where each individual queue has a dedicated doorbell register

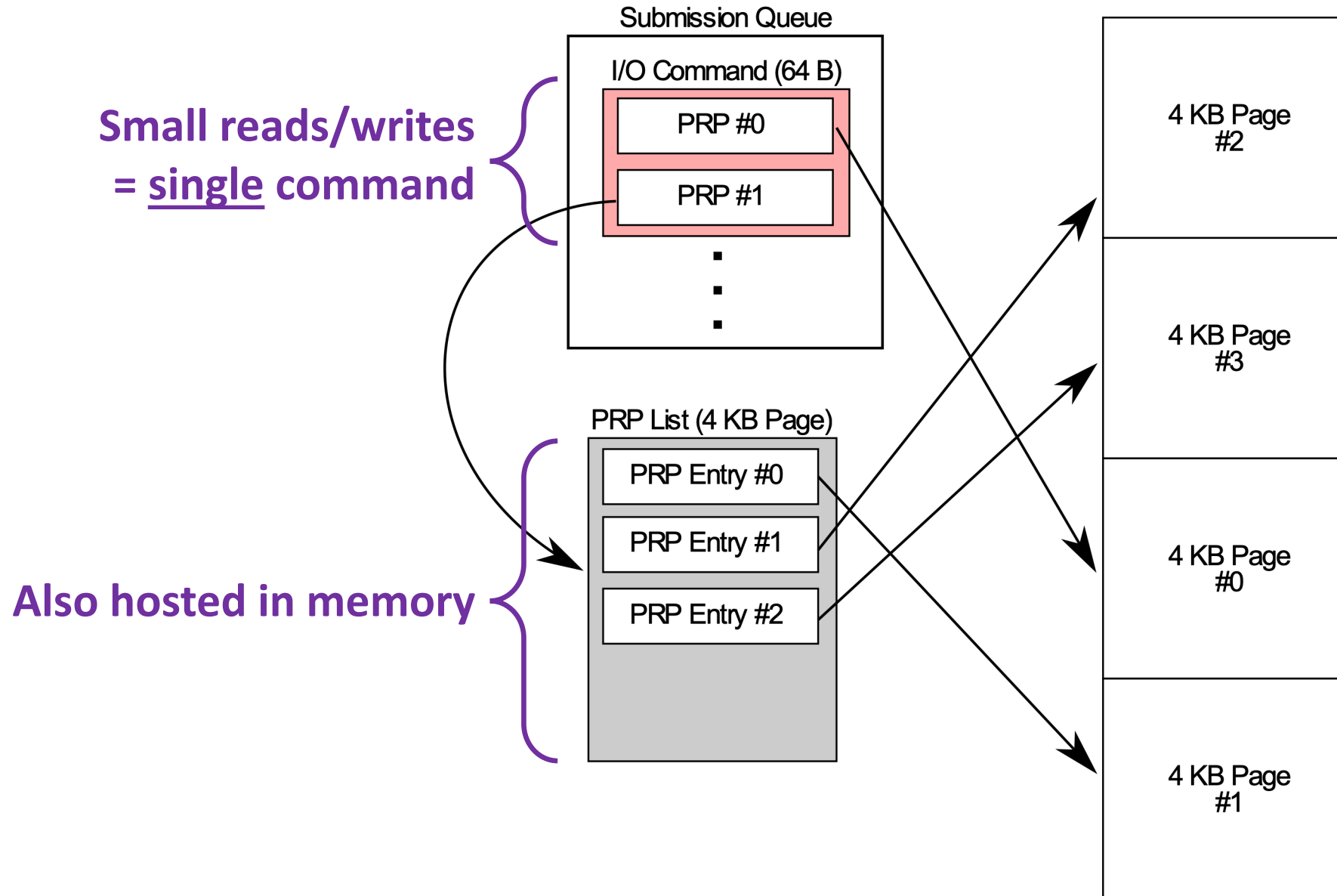
Submission Queue (SQ)



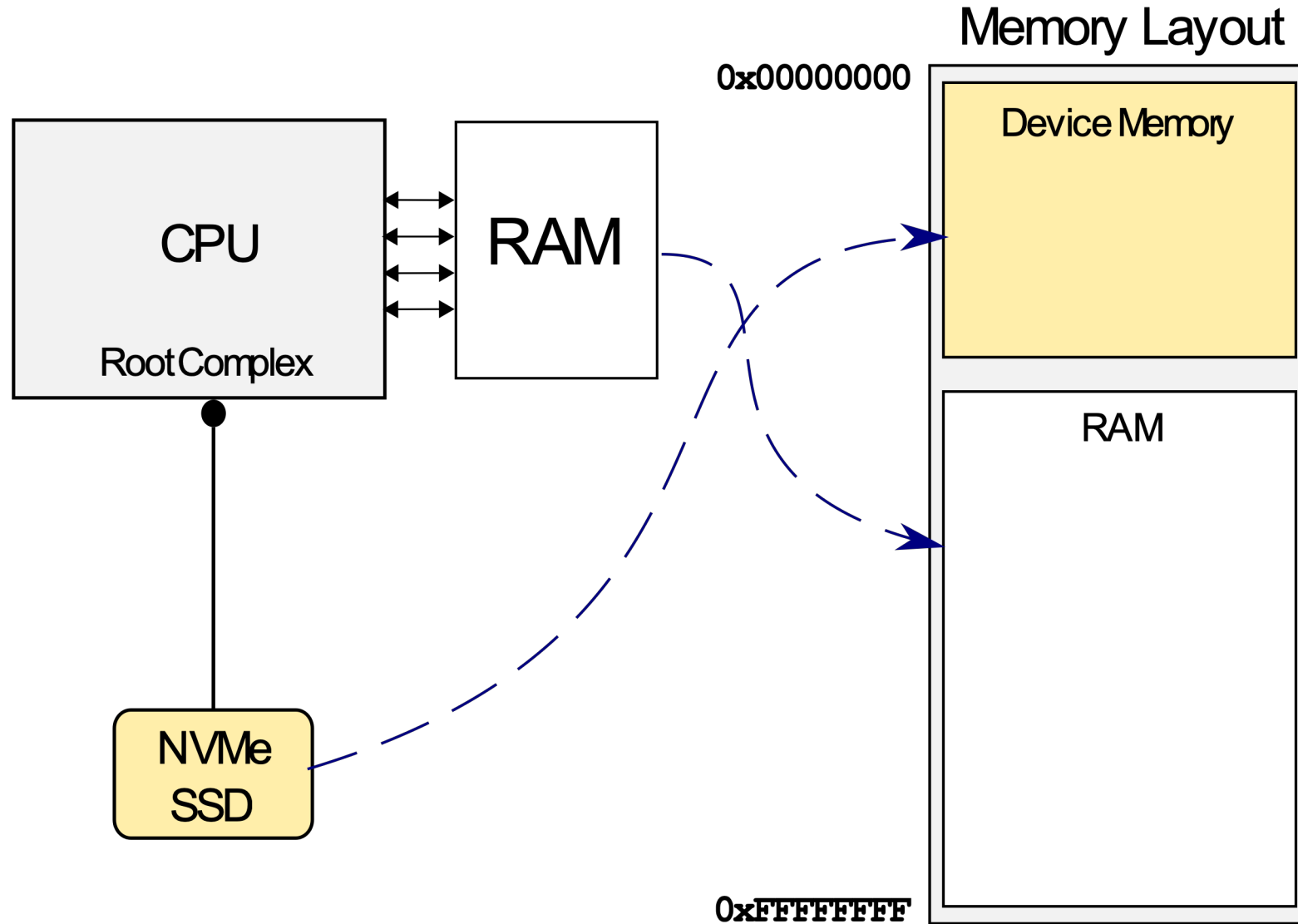
Command queues are implemented as ring-buffers where each individual queue has a dedicated doorbell register



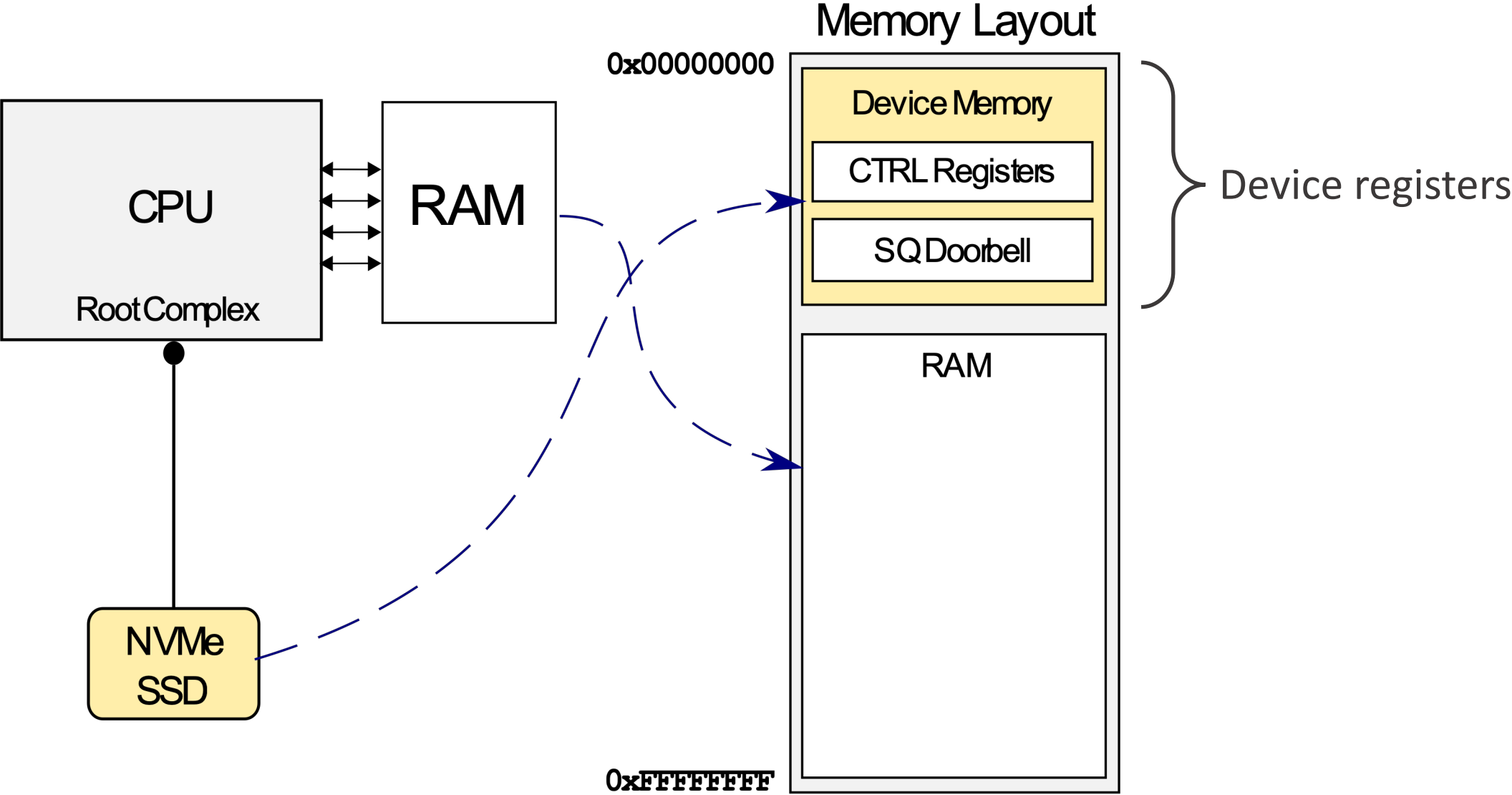
I/O commands use physical page region lists (PRP lists) to describe physical addresses of non-contiguous memory



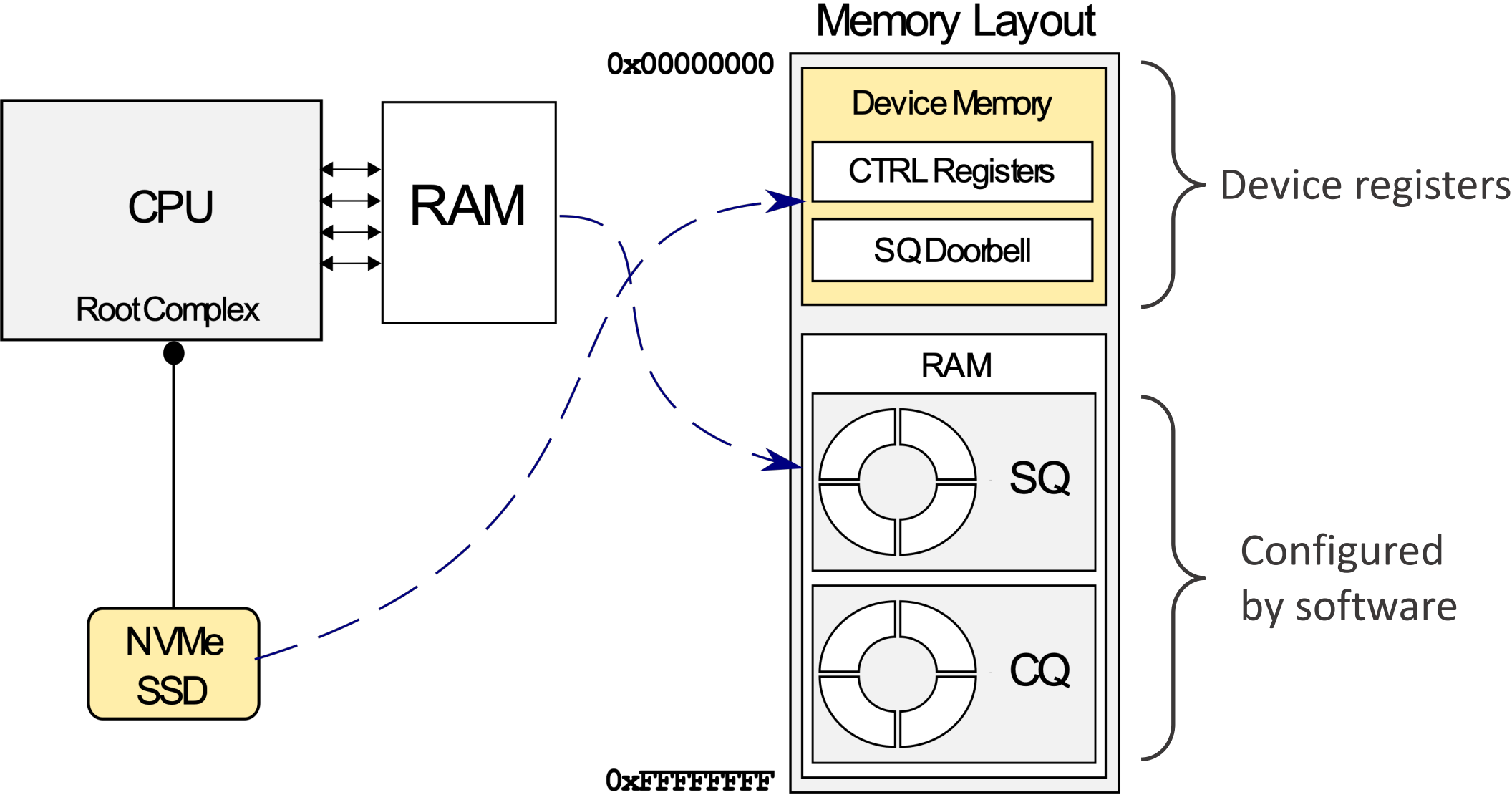
Software can set up I/O command queues anywhere in memory



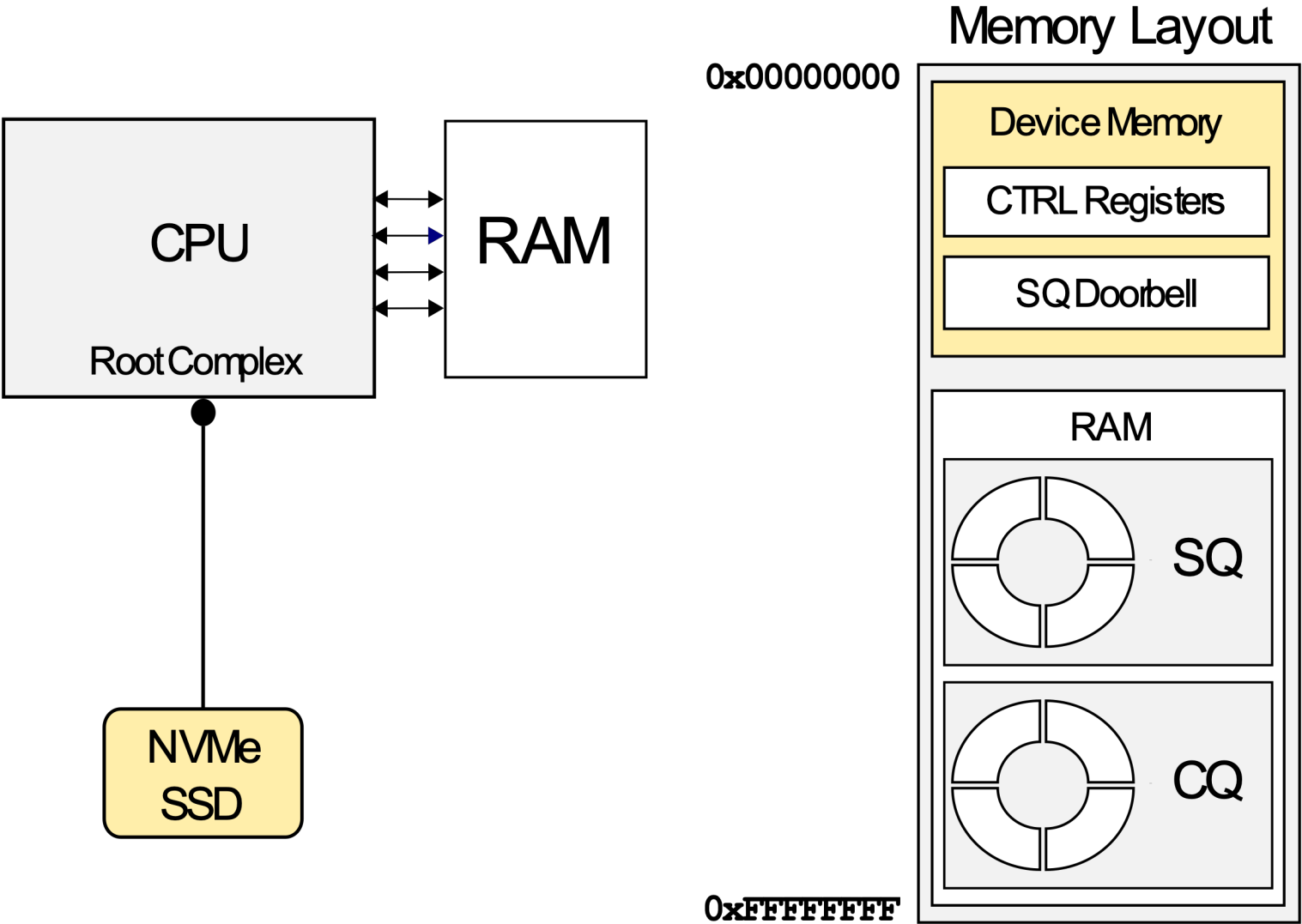
Software can set up up I/O command queues anywhere in memory



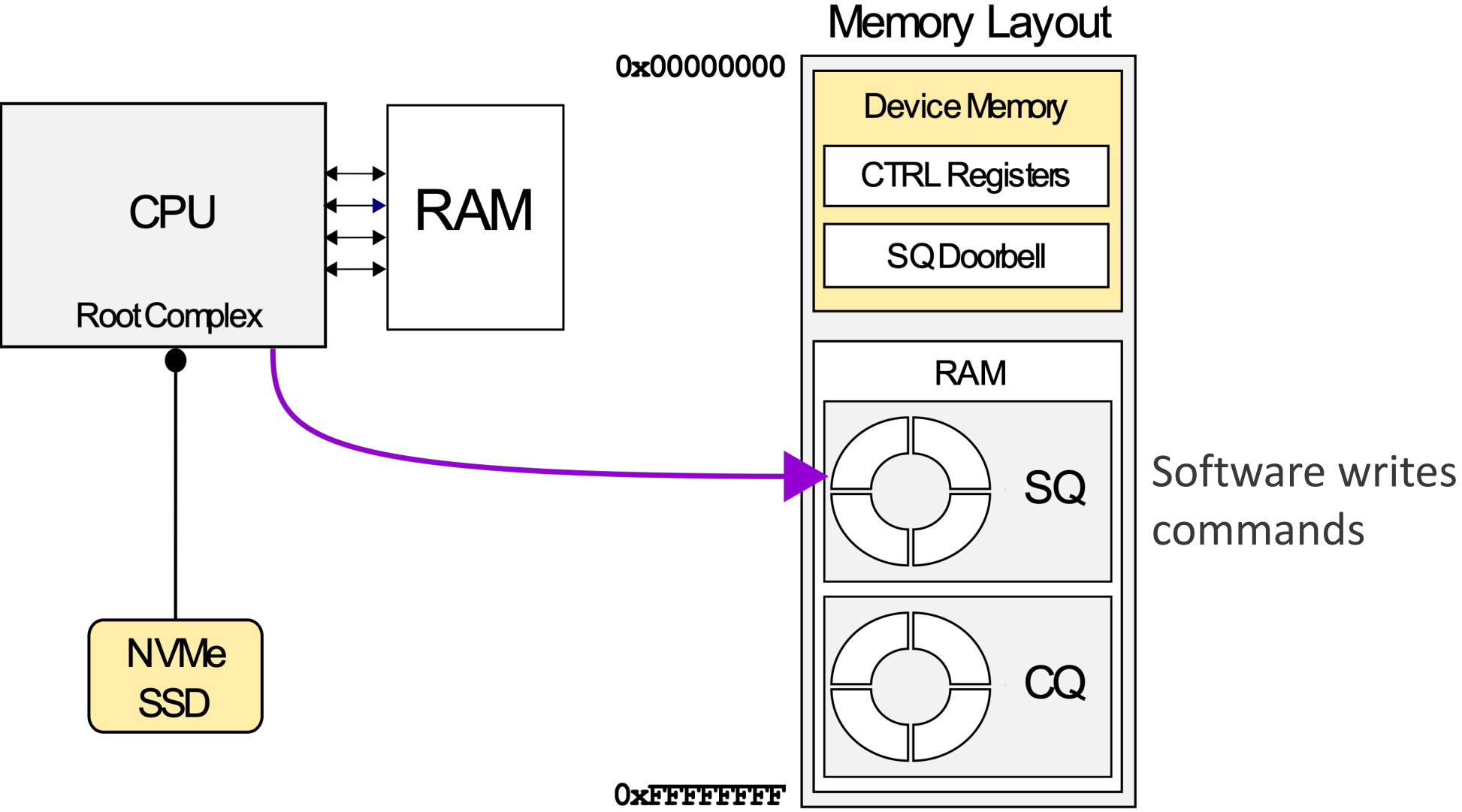
Software can set up up I/O command queues anywhere in memory



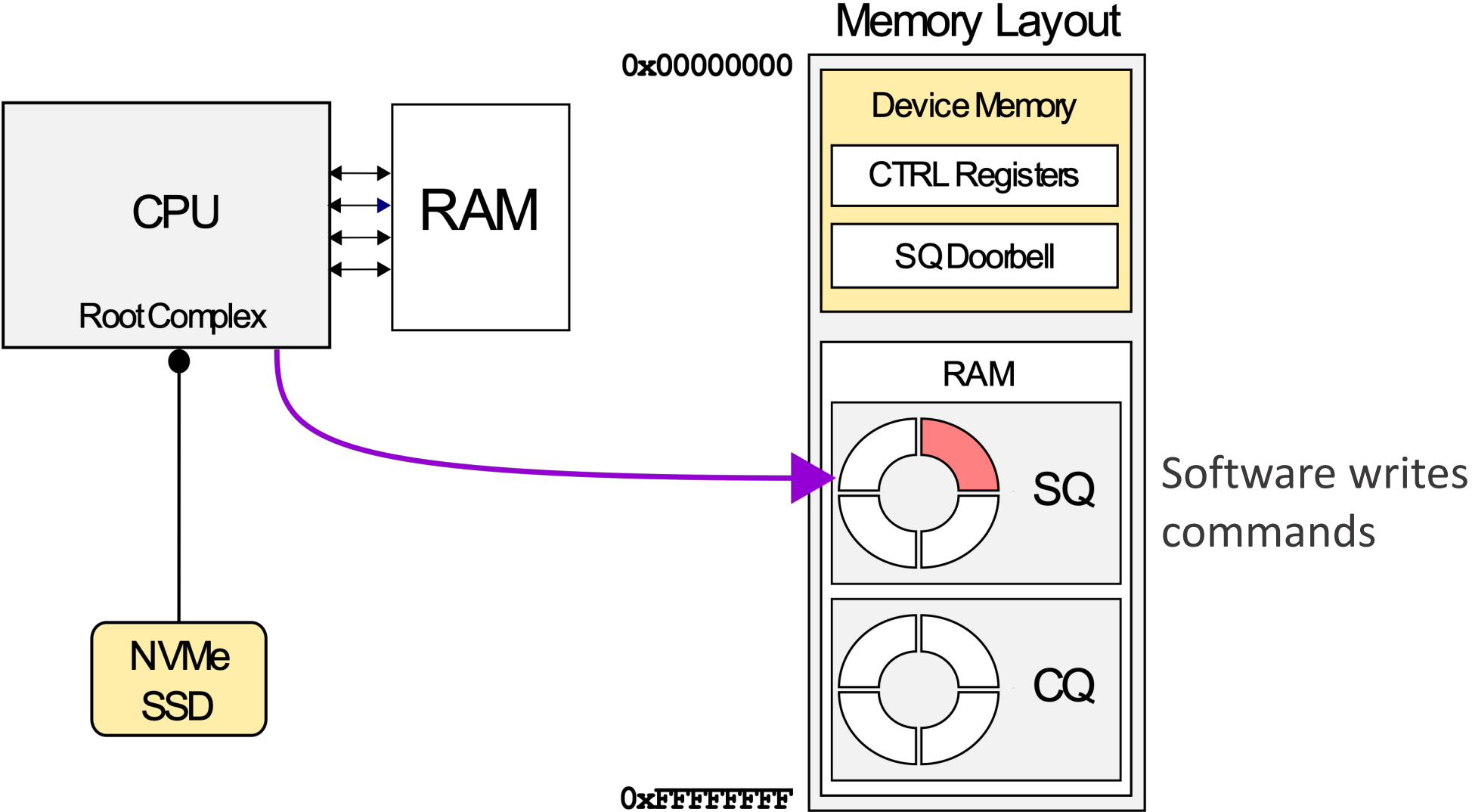
Software submits commands by writing to queue memory and updating the associated doorbell register



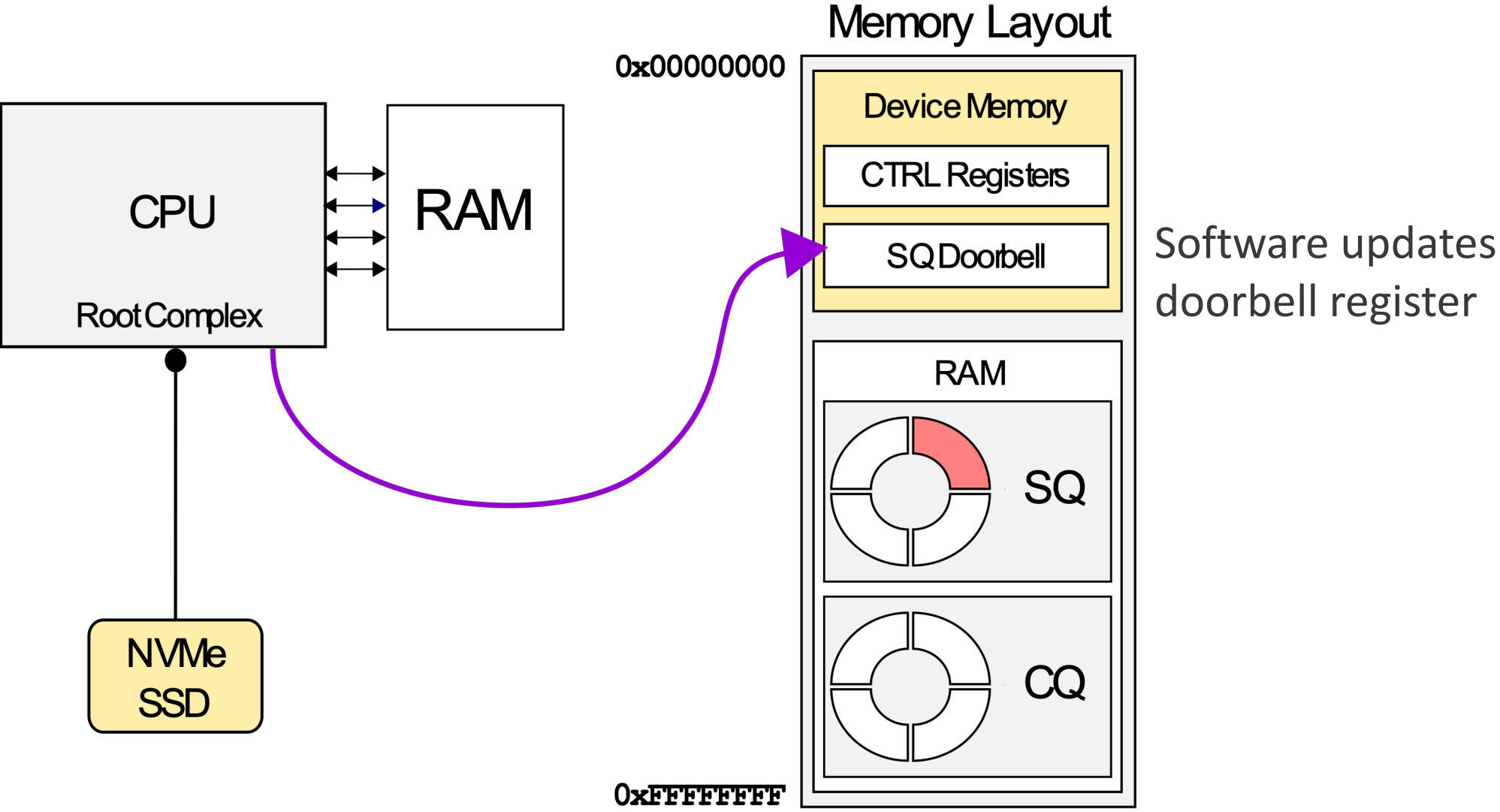
Software submits commands by writing to queue memory and updating the associated doorbell register



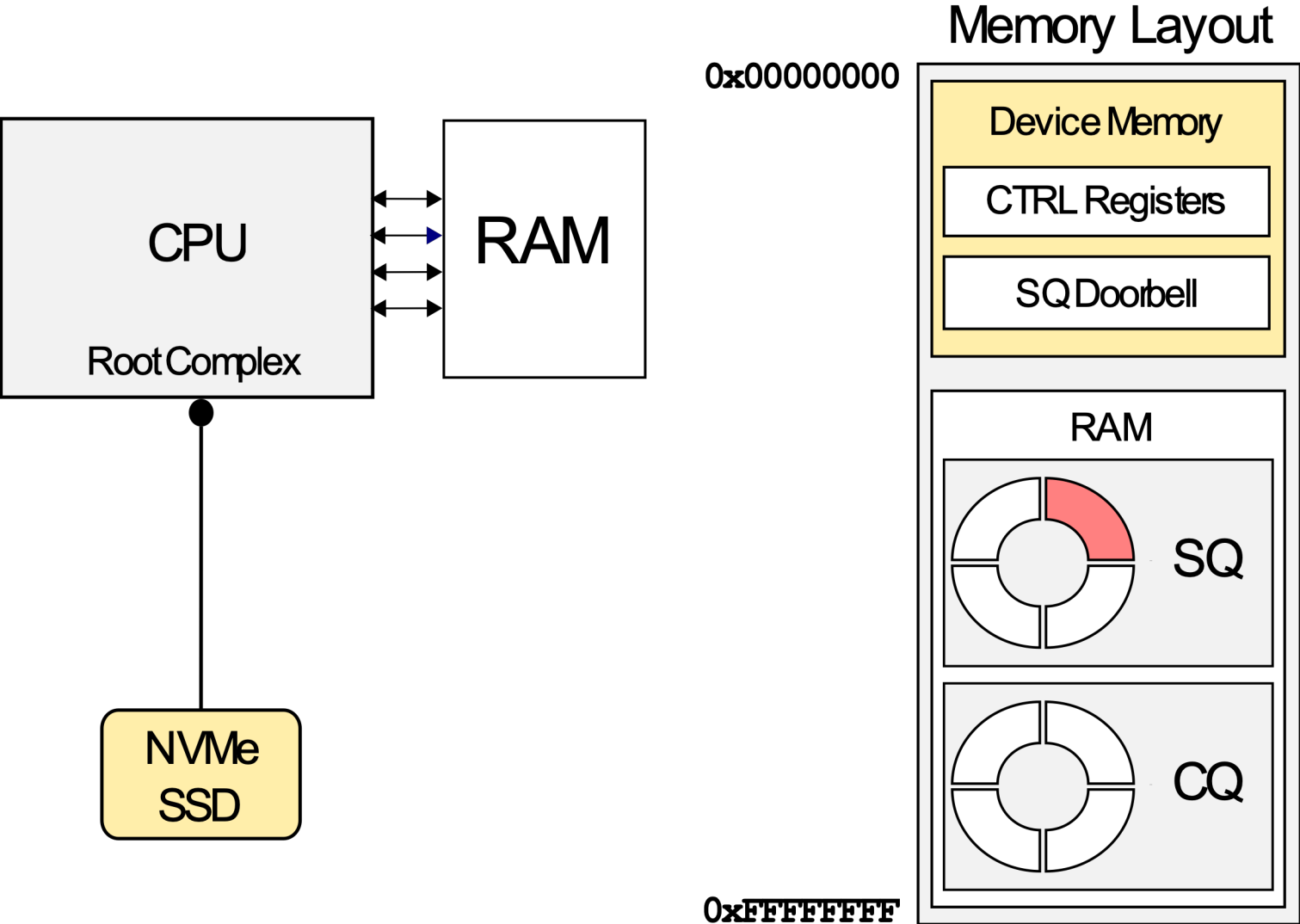
Software submits commands by writing to queue memory and updating the associated doorbell register



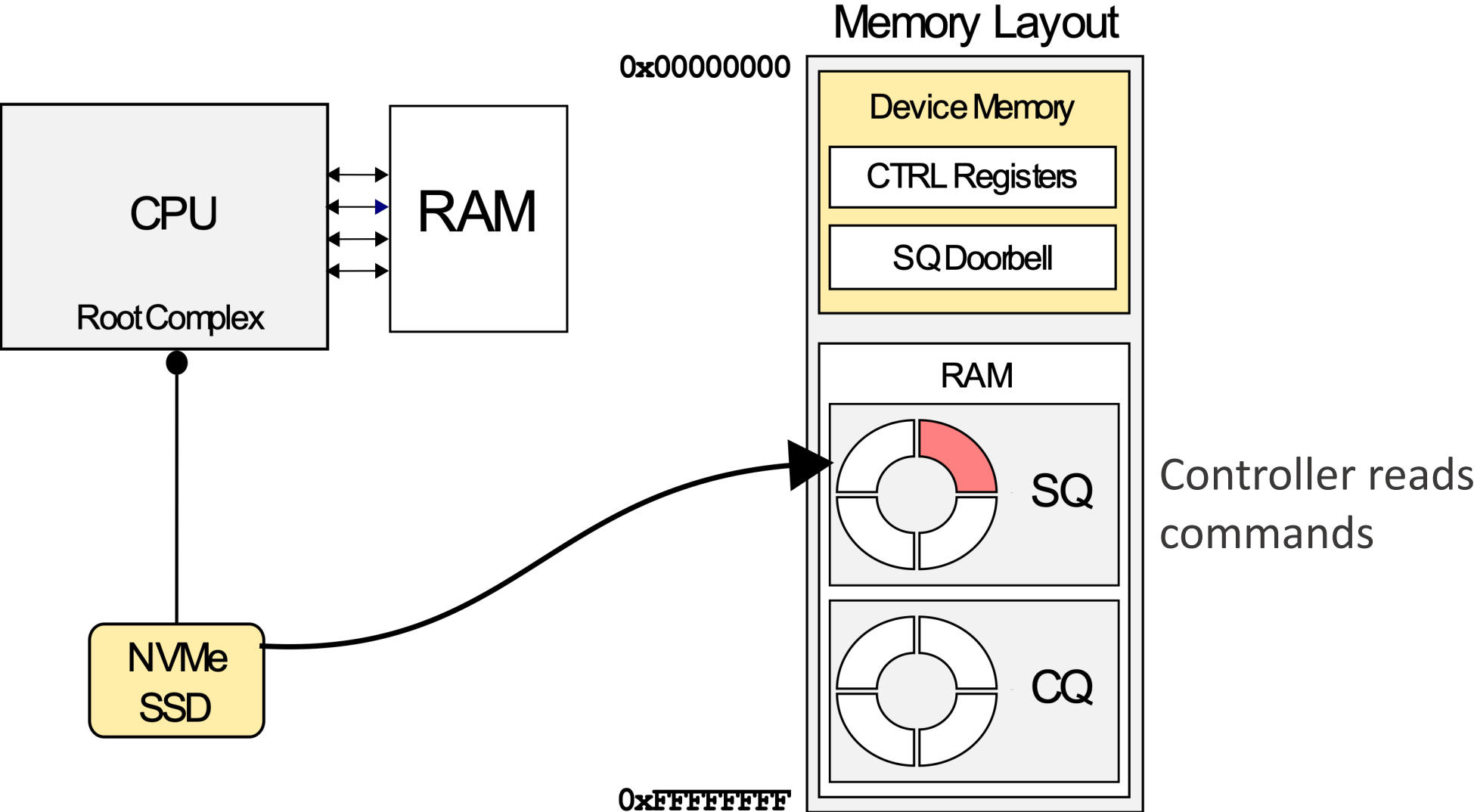
Software submits commands by writing to queue memory and updating the associated doorbell register



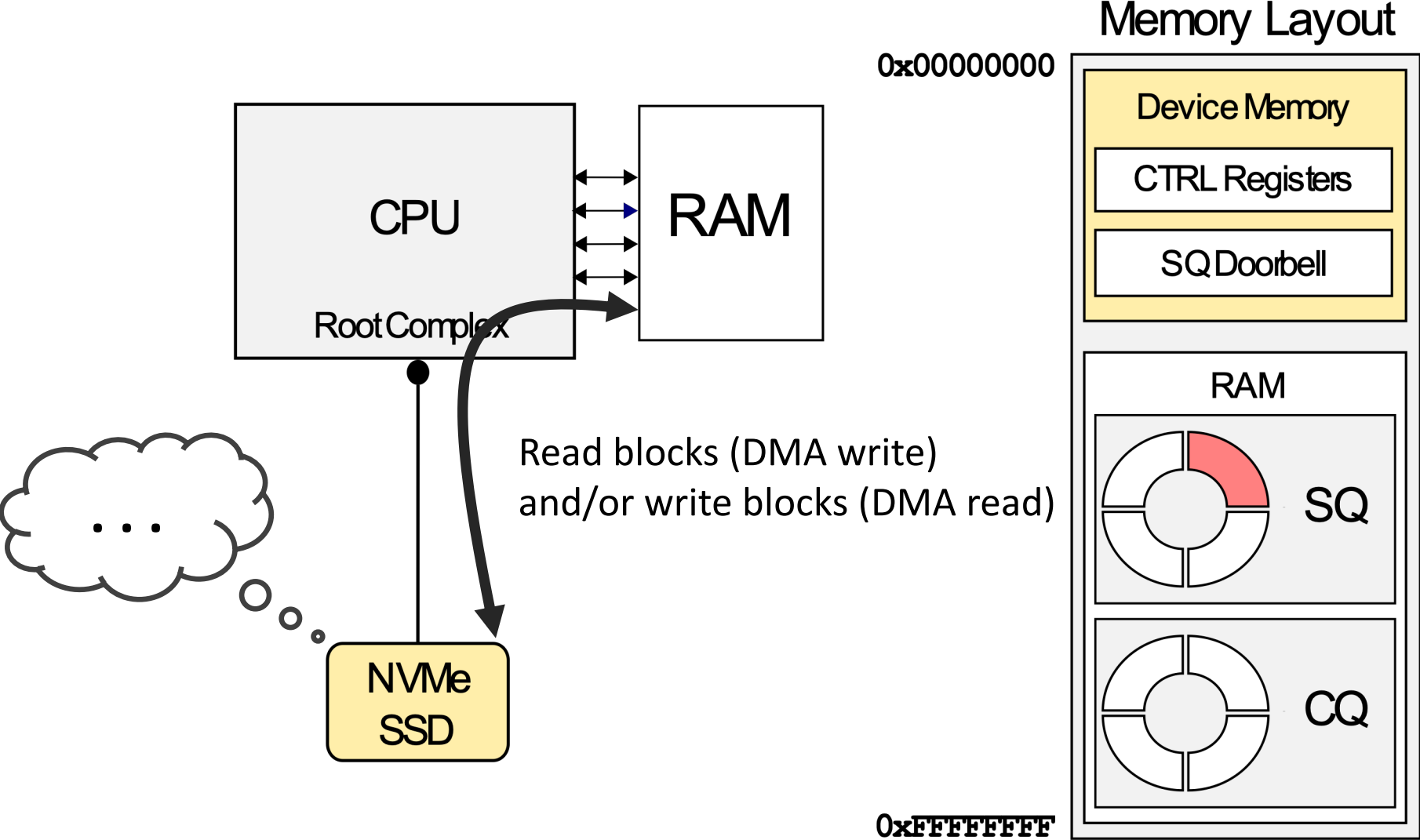
Controller reads commands from submission queue memory and writes completions to completion queue memory



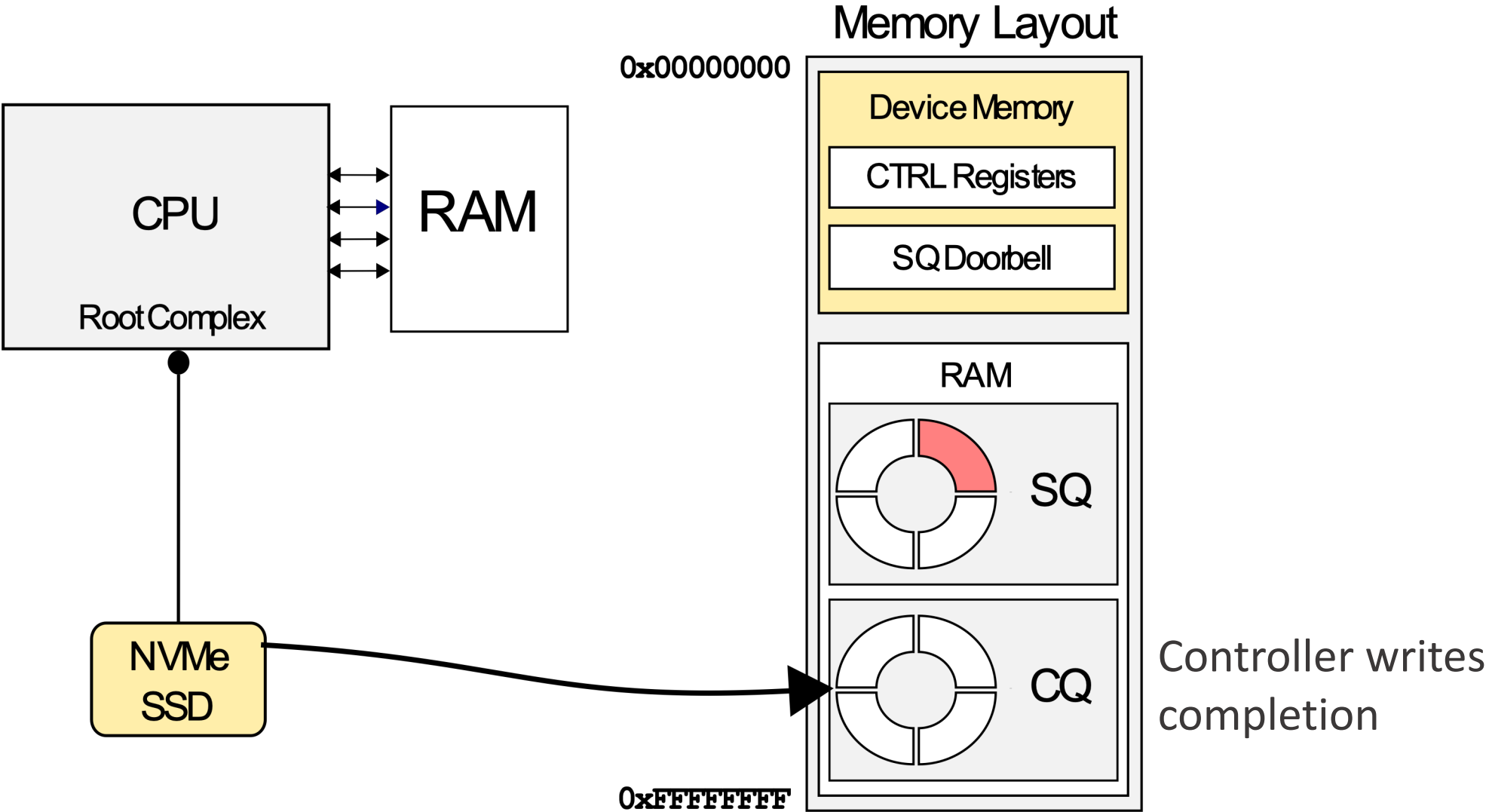
Controller reads commands from submission queue memory and writes completions to completion queue memory



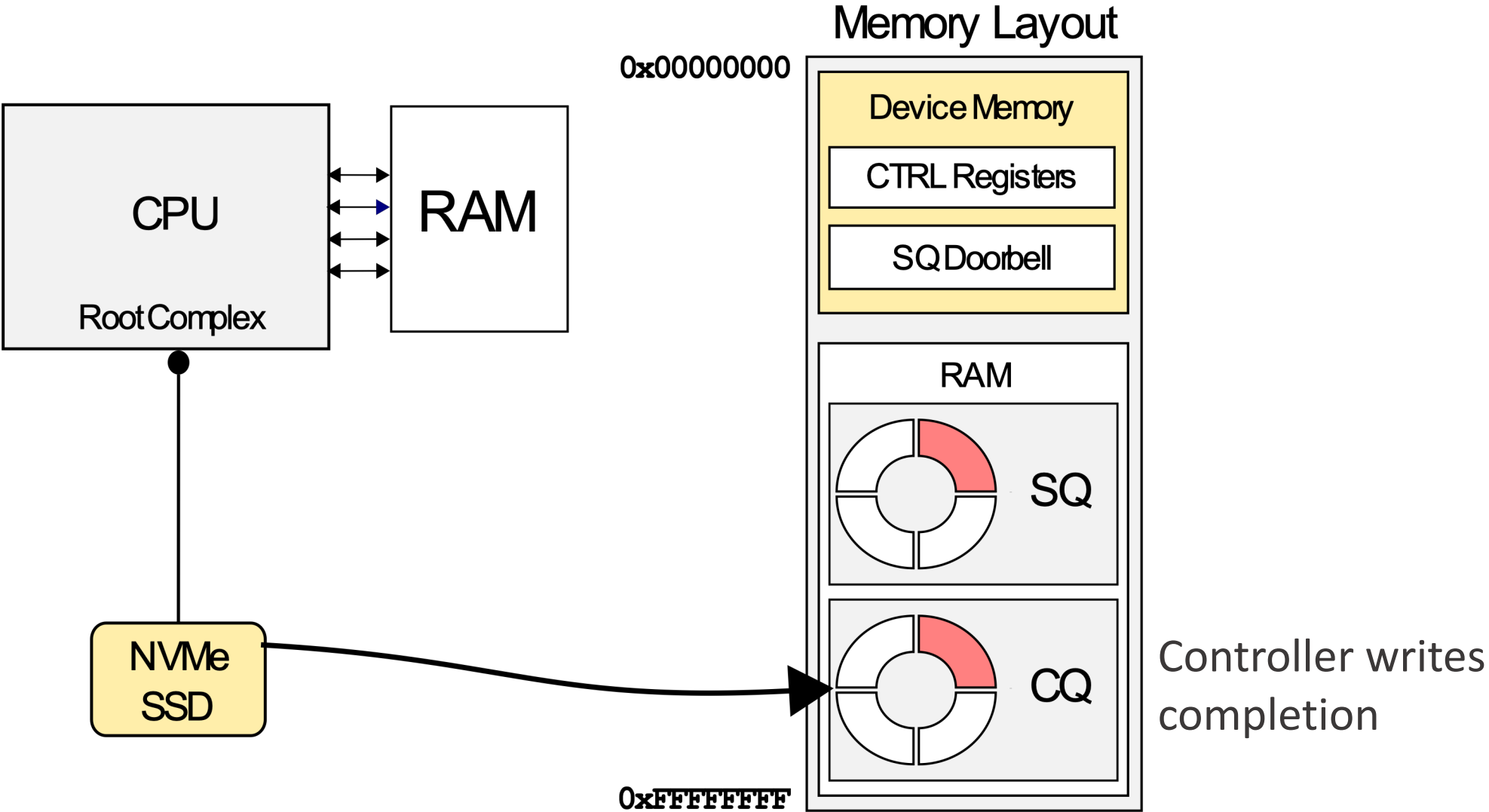
Controller reads commands from submission queue memory and writes completions to completion queue memory



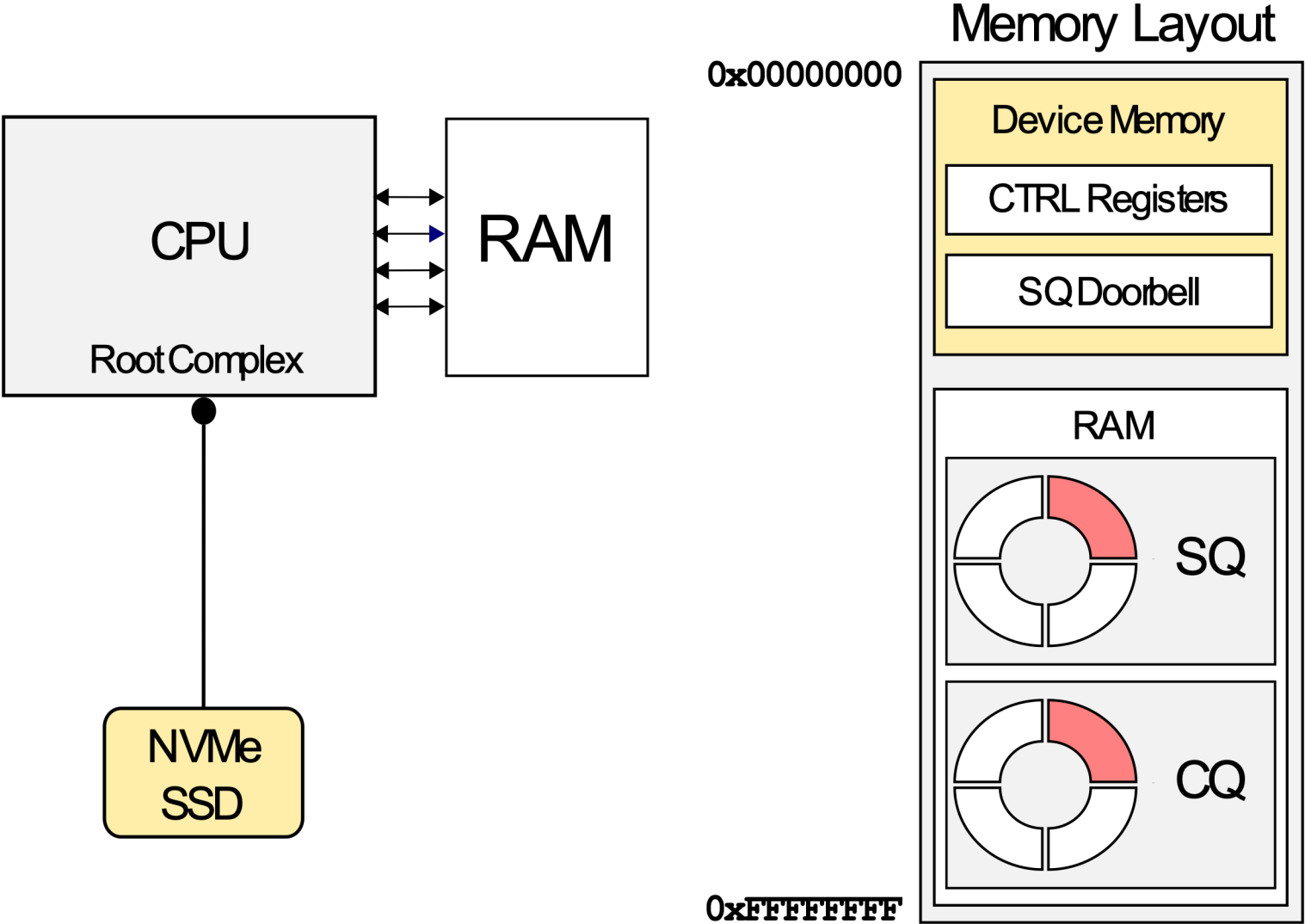
Controller reads commands from submission queue memory and writes completions to completion queue memory



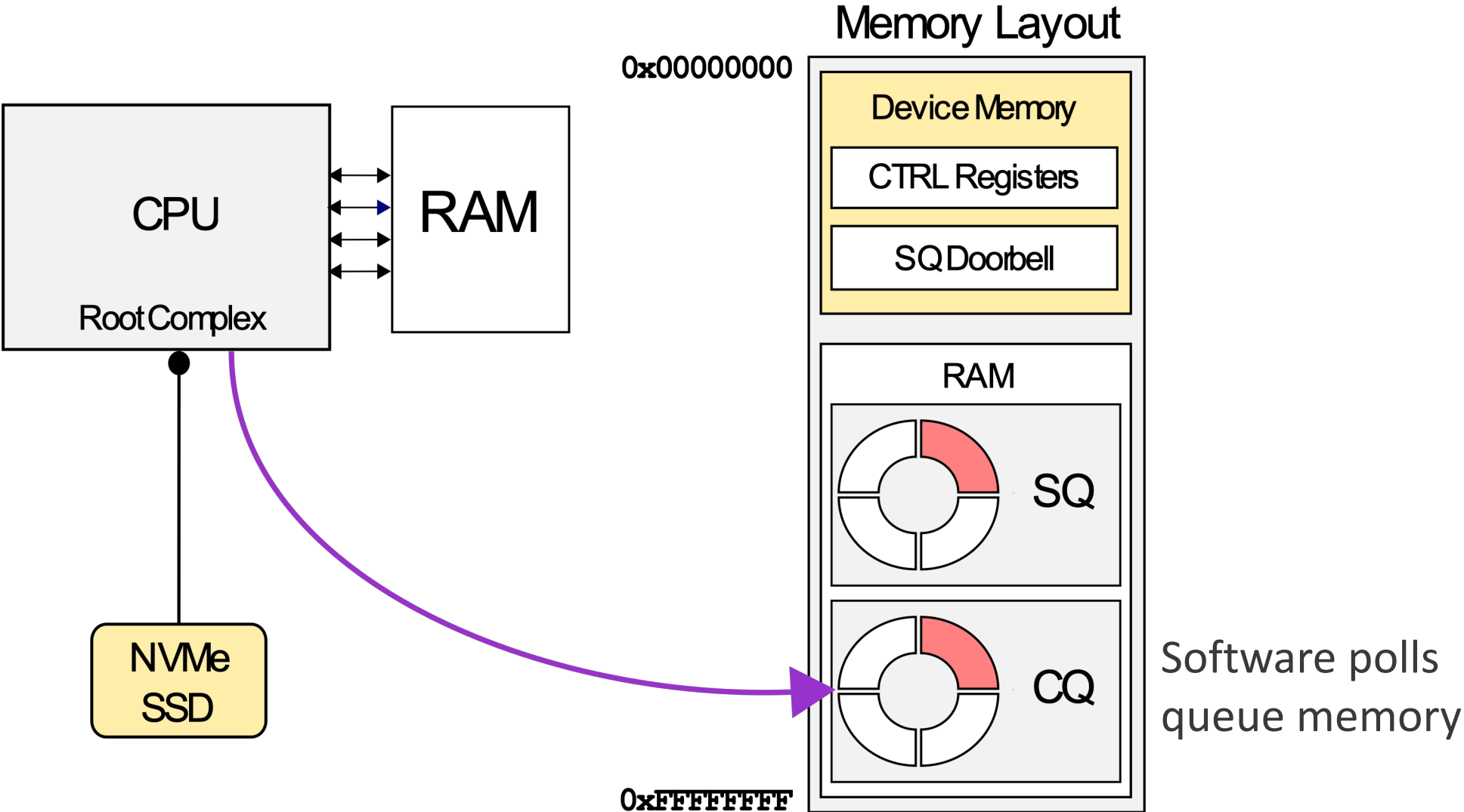
Controller reads commands from submission queue memory and writes completions to completion queue memory



Software detects completions either by waiting for hardware interrupts or by polling completion queue memory



Software detects completions either by waiting for hardware interrupts or by polling completion queue memory

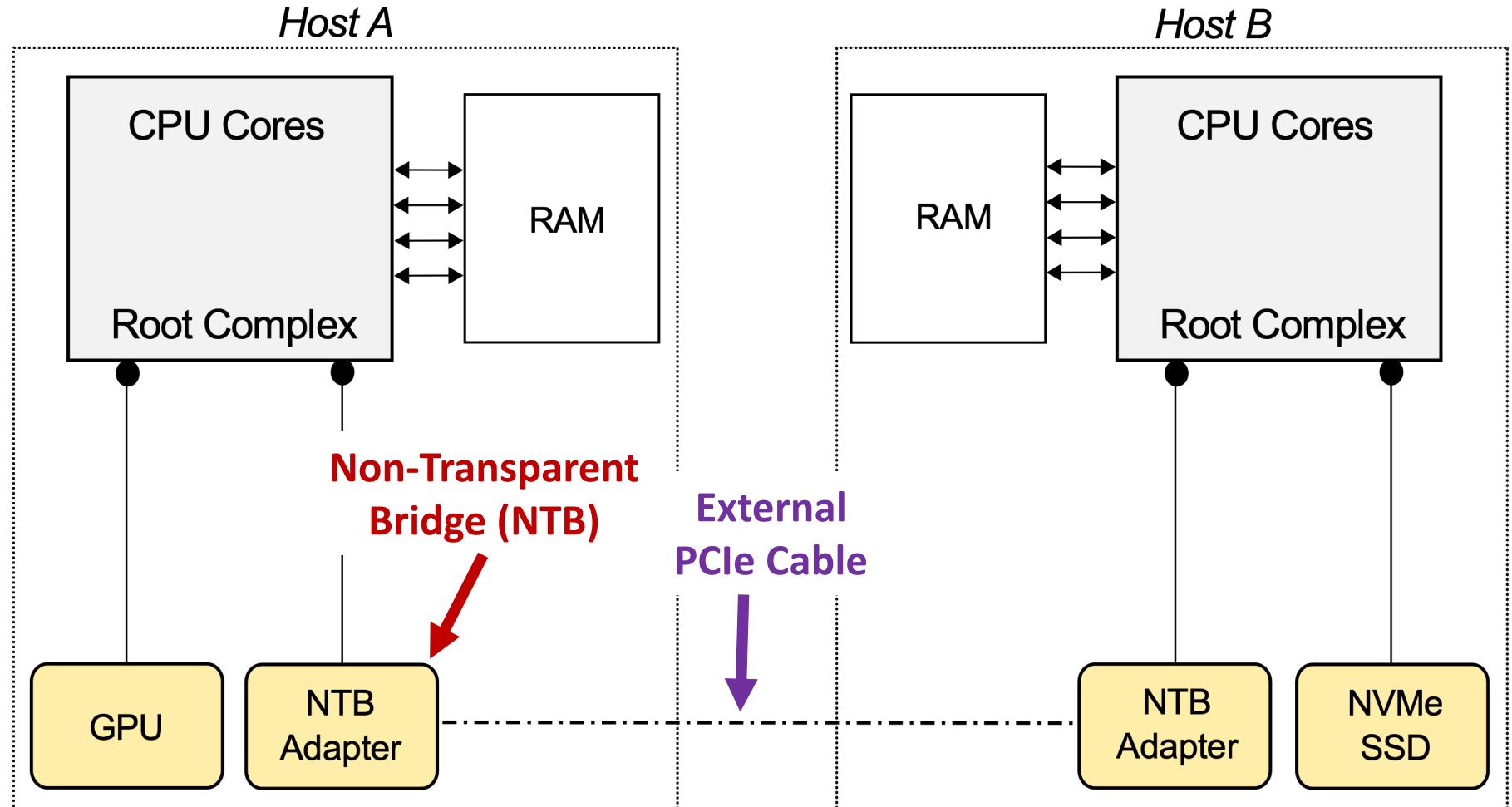


NVMe Summary

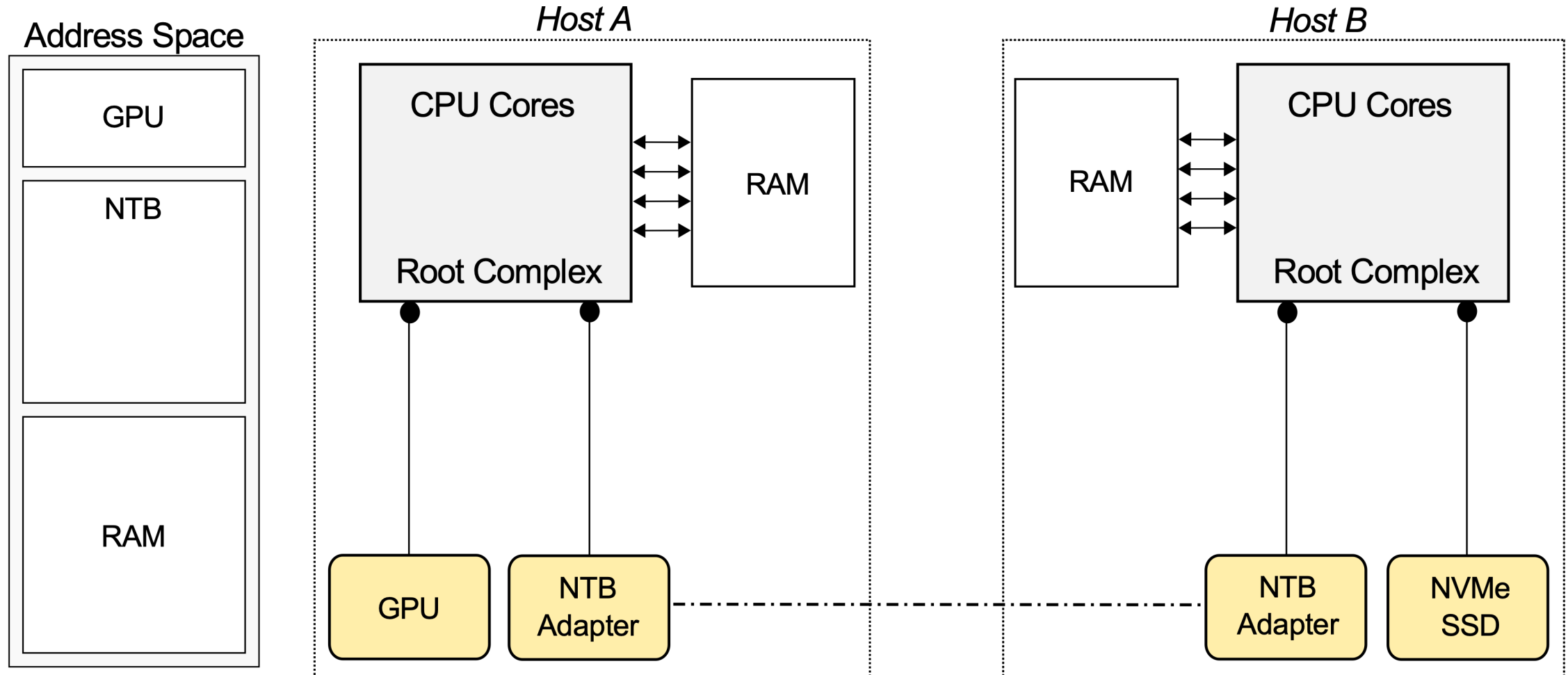
- Multiple I/O queues enables highly parallel design
- Controller uses DMA = queues and data buffers can be hosted anywhere in memory space
- Single doorbell register write in I/O path
- Software can poll for completions instead of waiting for hardware interrupt

Non-Transparent Bridging

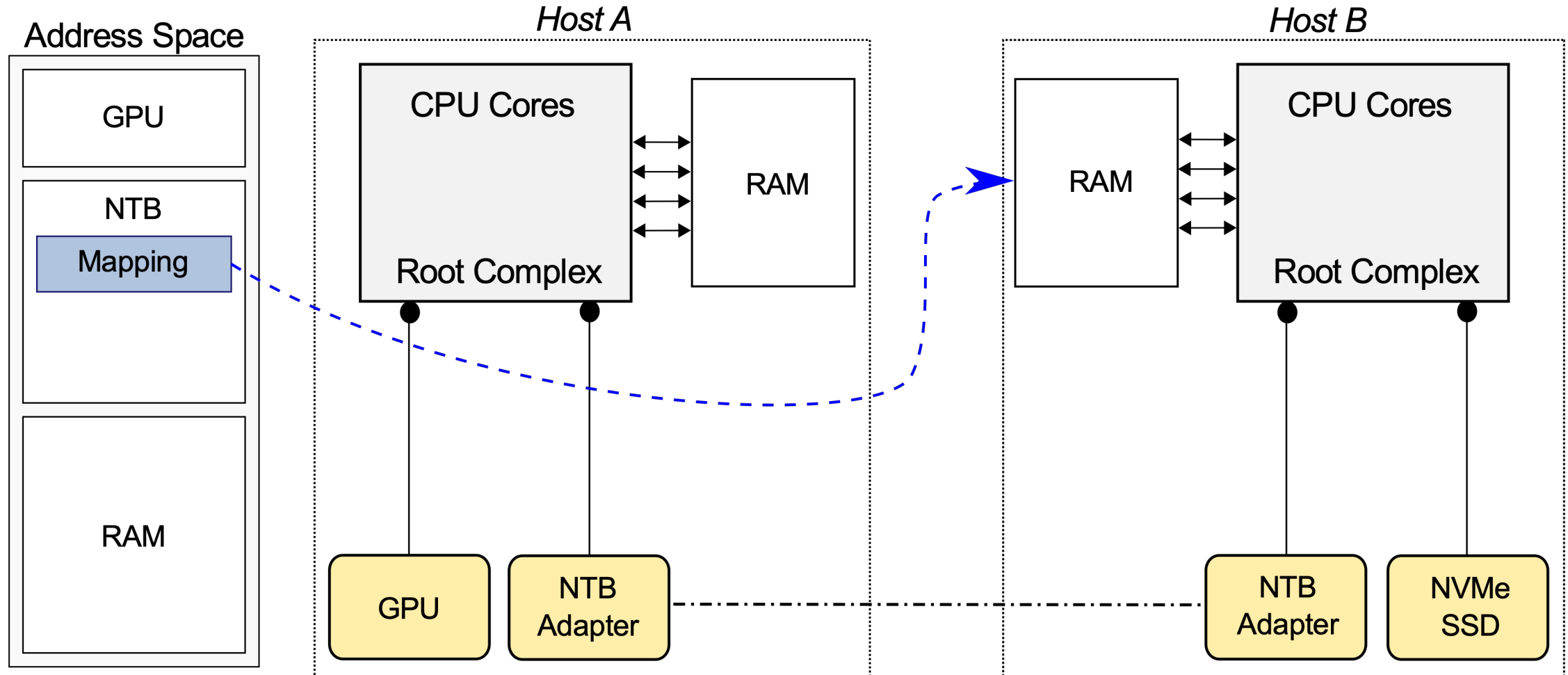
We can interconnect separate PCIe root complexes and translate addresses between them using a non-transparent bridge (NTB)



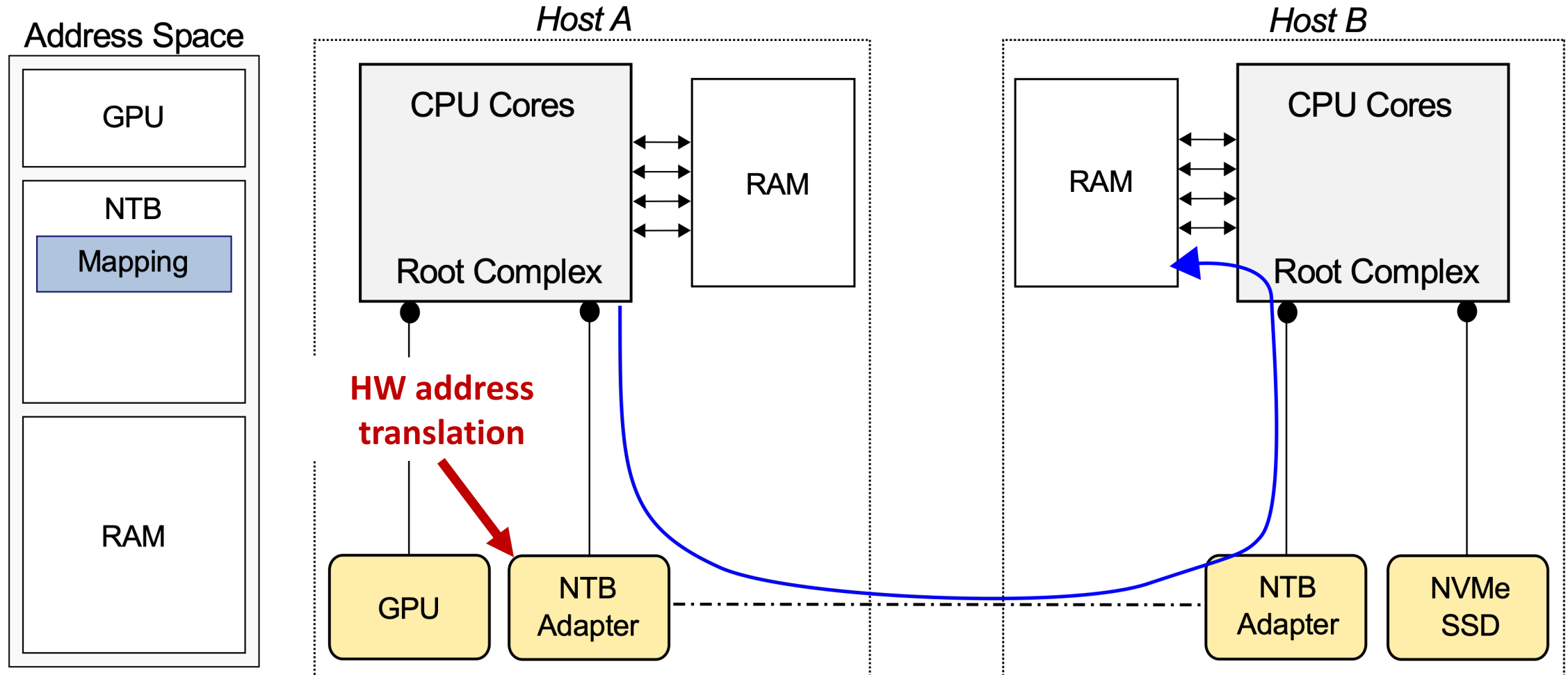
We can interconnect separate PCIe root complexes and translate addresses between them using a non-transparent bridge (NTB)



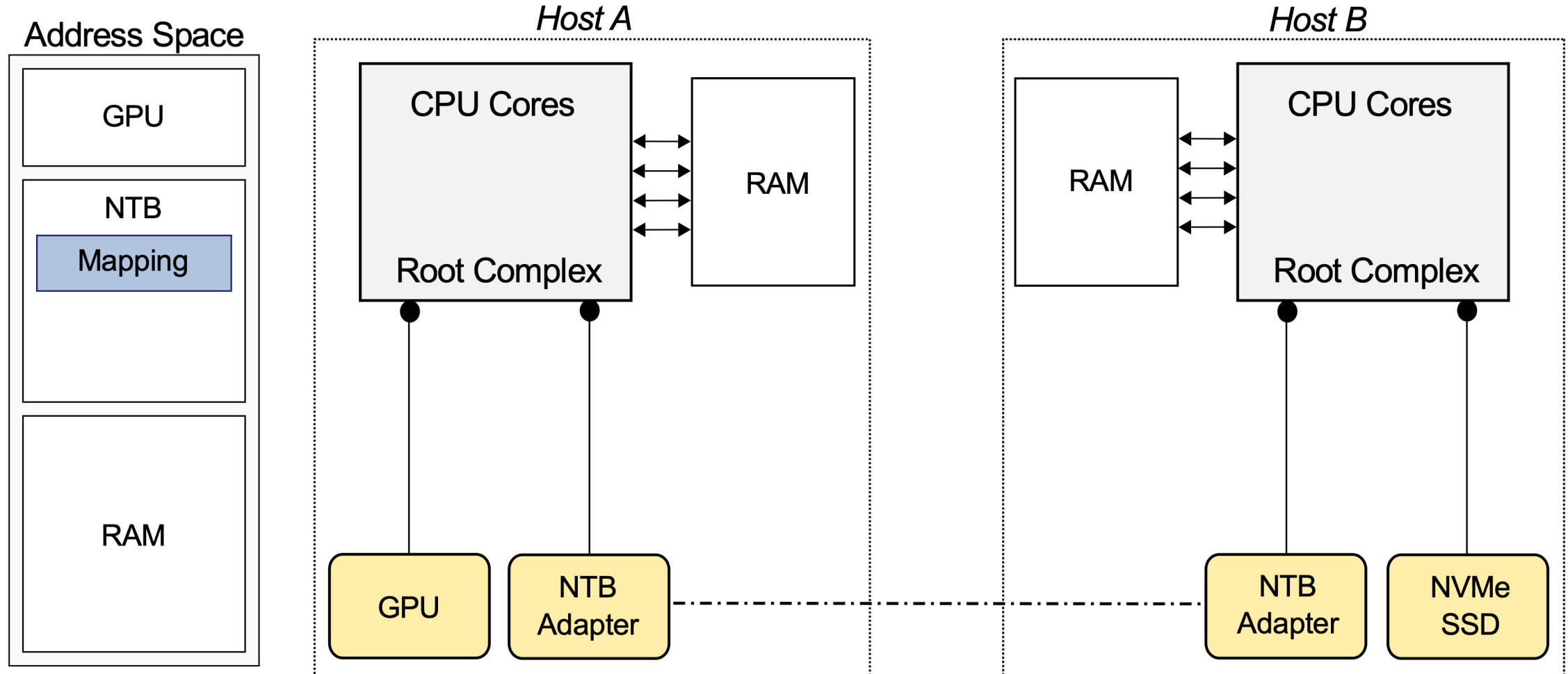
We can interconnect separate PCIe root complexes and translate addresses between them using a non-transparent bridge (NTB)



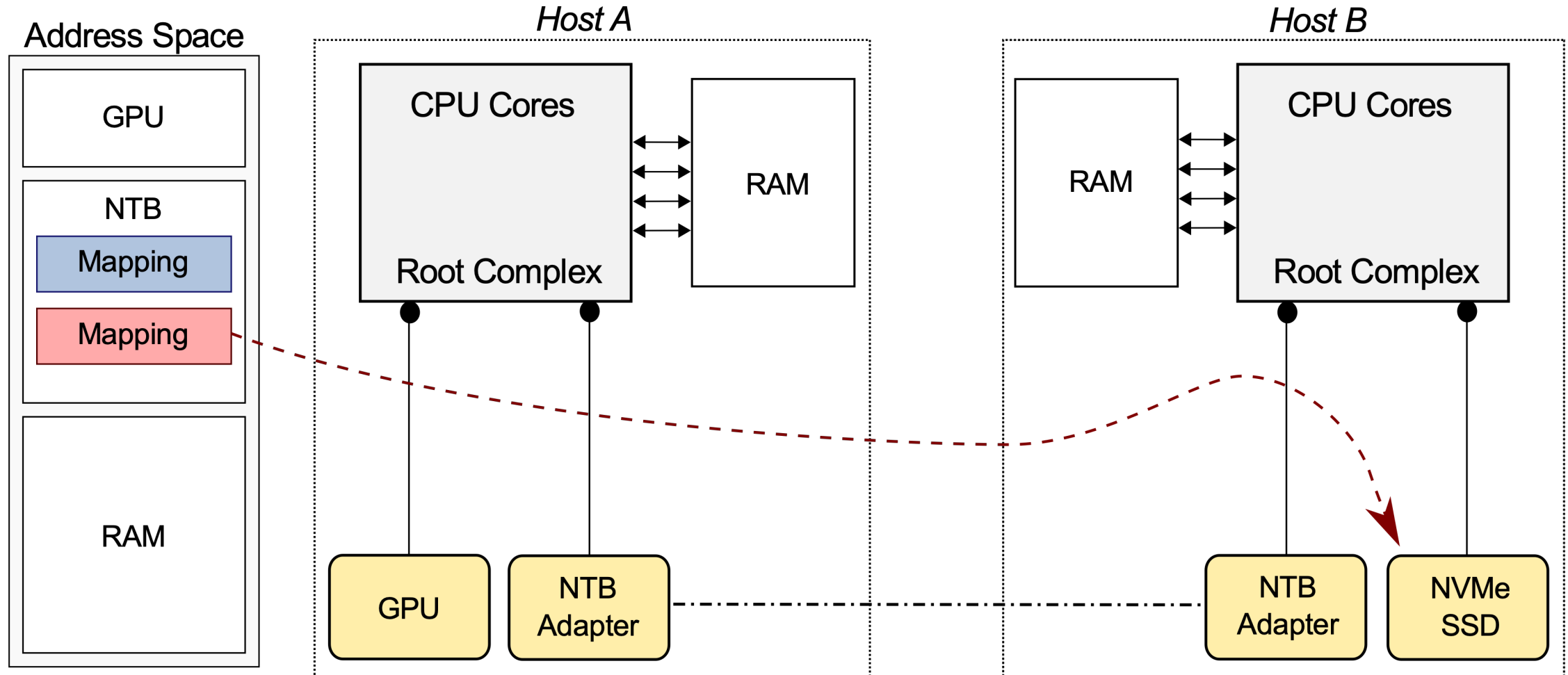
We can interconnect separate PCIe root complexes and translate addresses between them using a non-transparent bridge (NTB)



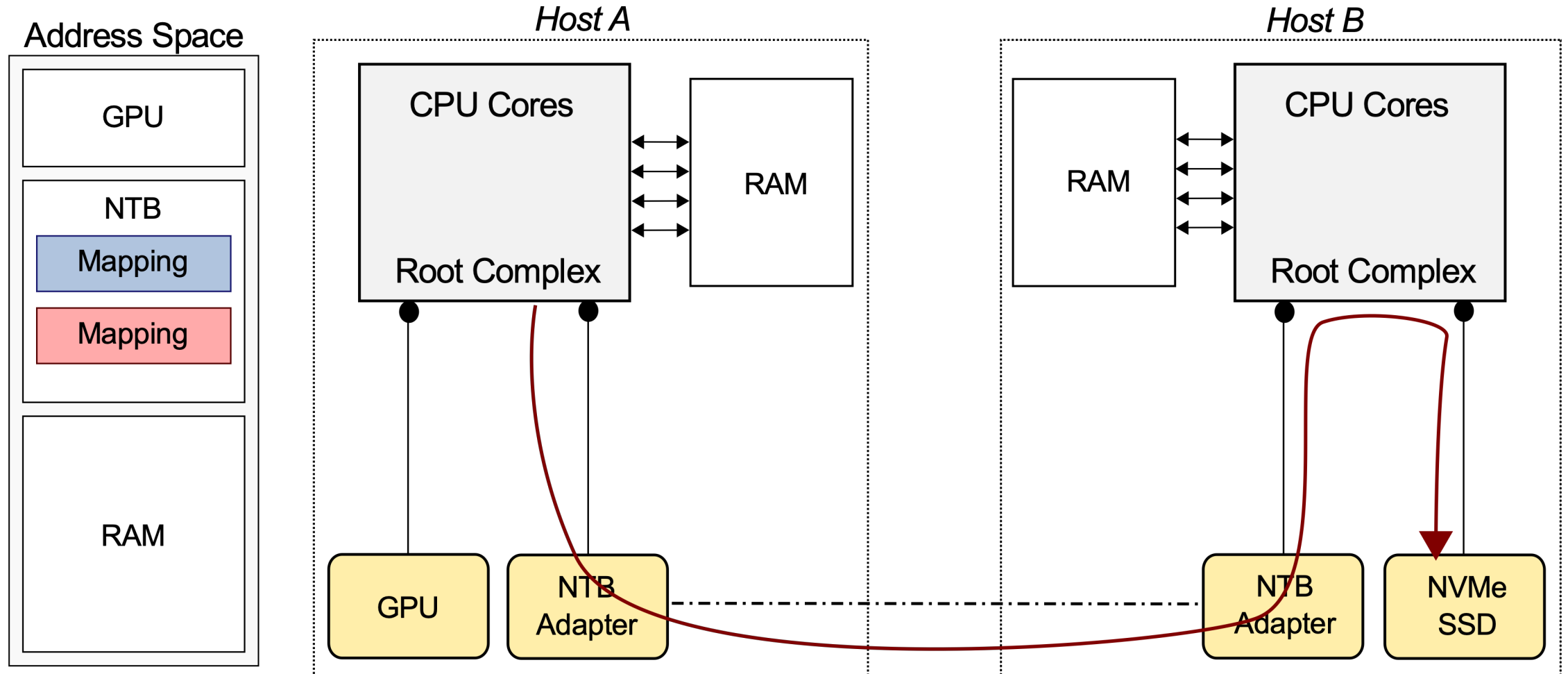
Since PCIe devices are also part of address space, it is also possible to map remote device resources



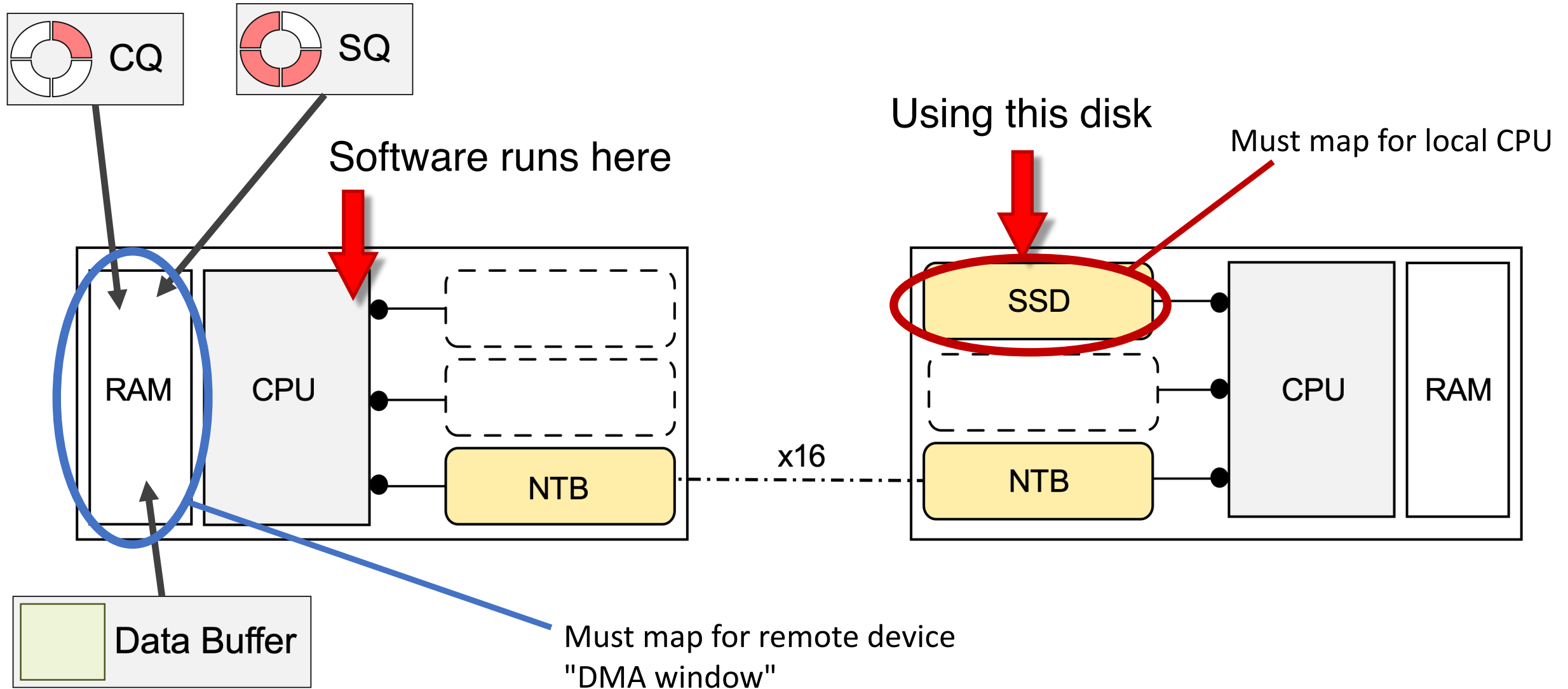
Since PCIe devices are also part of address space, it is also possible to map remote device resources



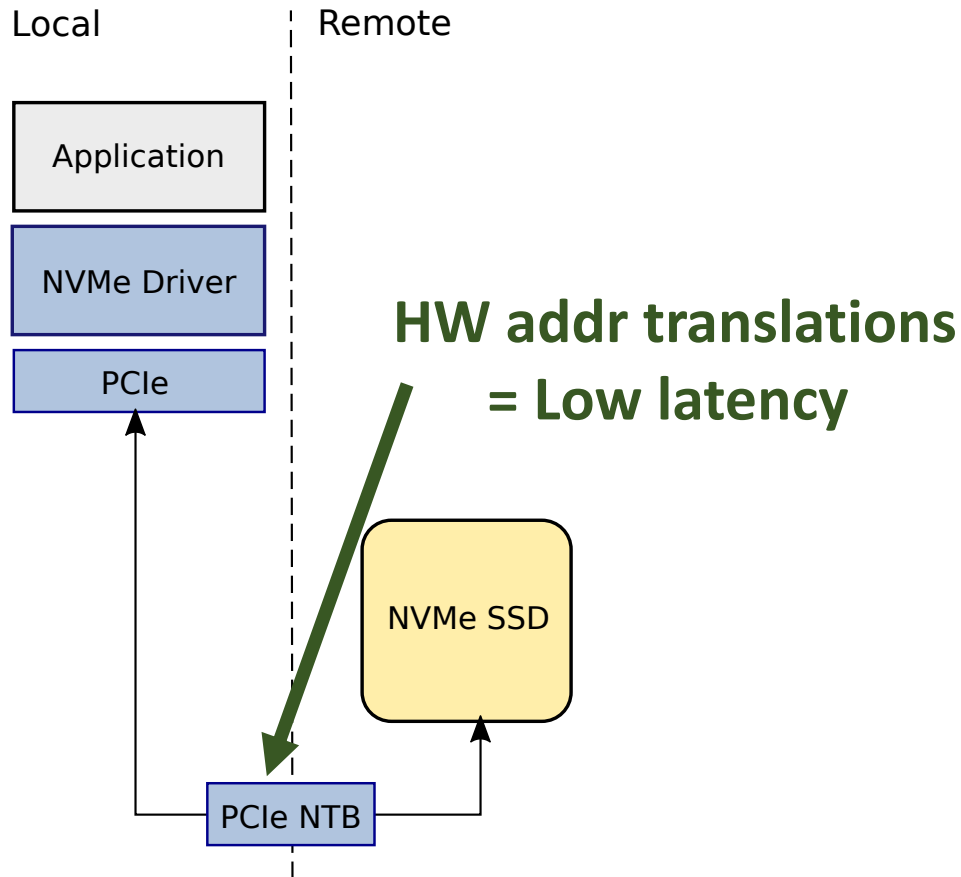
Since PCIe devices are also part of address space, it is also possible to map remote device resources



Using NTBs, it is possible for a local driver to use a remote device by setting up MMIO and DMA mappings



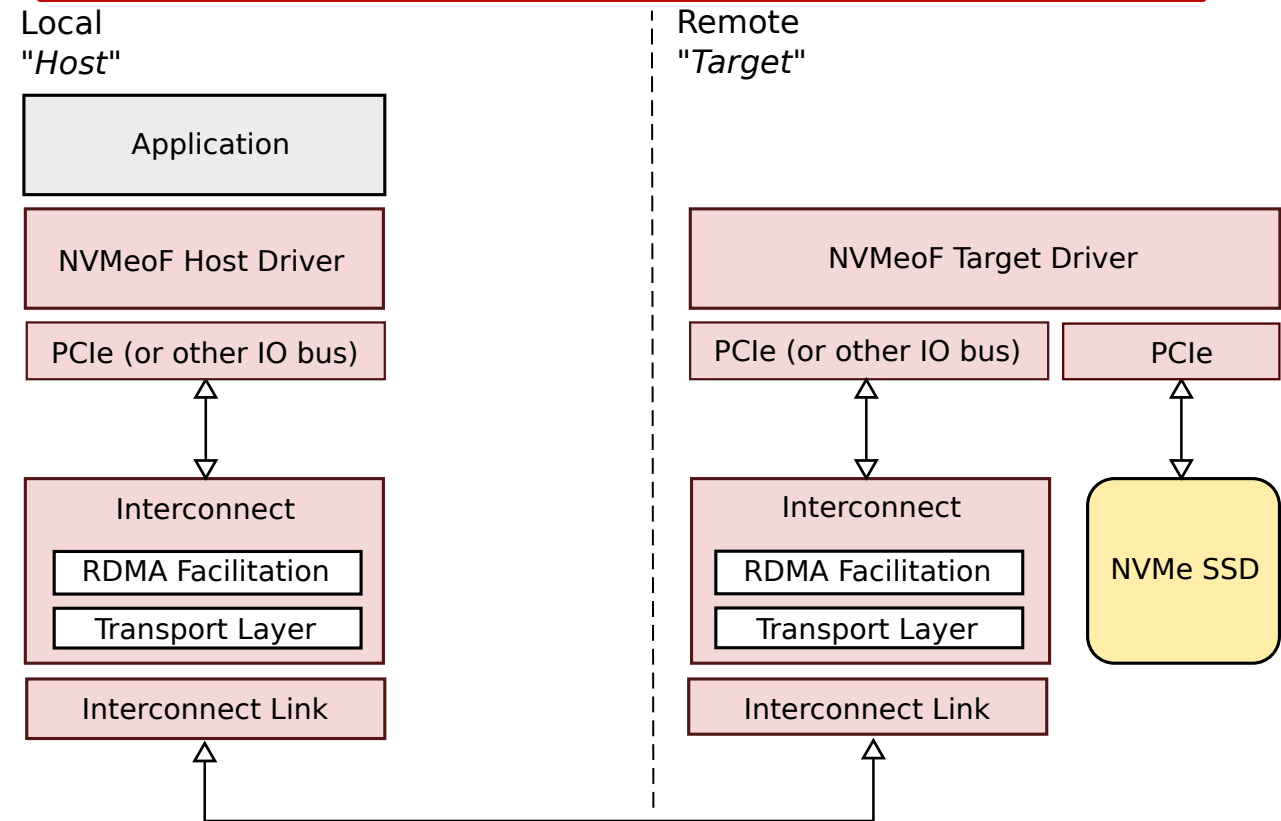
Using NTBs, it is possible for a local driver to use a remote device by setting up MMIO and DMA mappings



Native NVMe over PCIe NTB

4 kB read completion latency = $\sim 14.21 \mu\text{s}$

4 kB read completion latency = $\sim 17.91 \mu\text{s}$

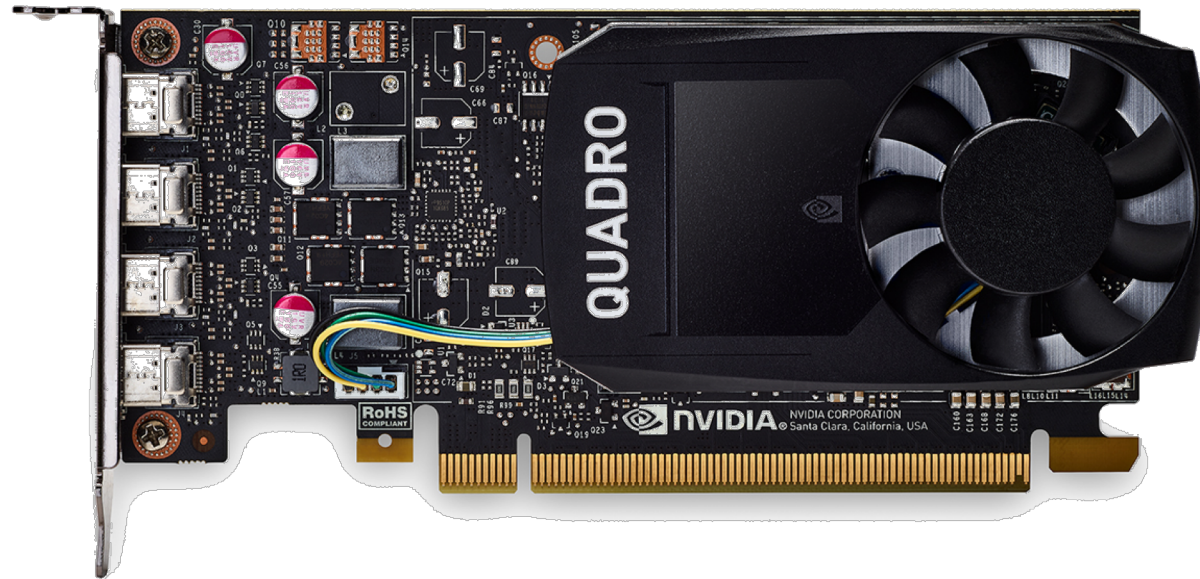


NVMe over Fabrics (NVMeoF)
using 100 GbE Ethernet RDMA
(SPDK target, kernel direct)

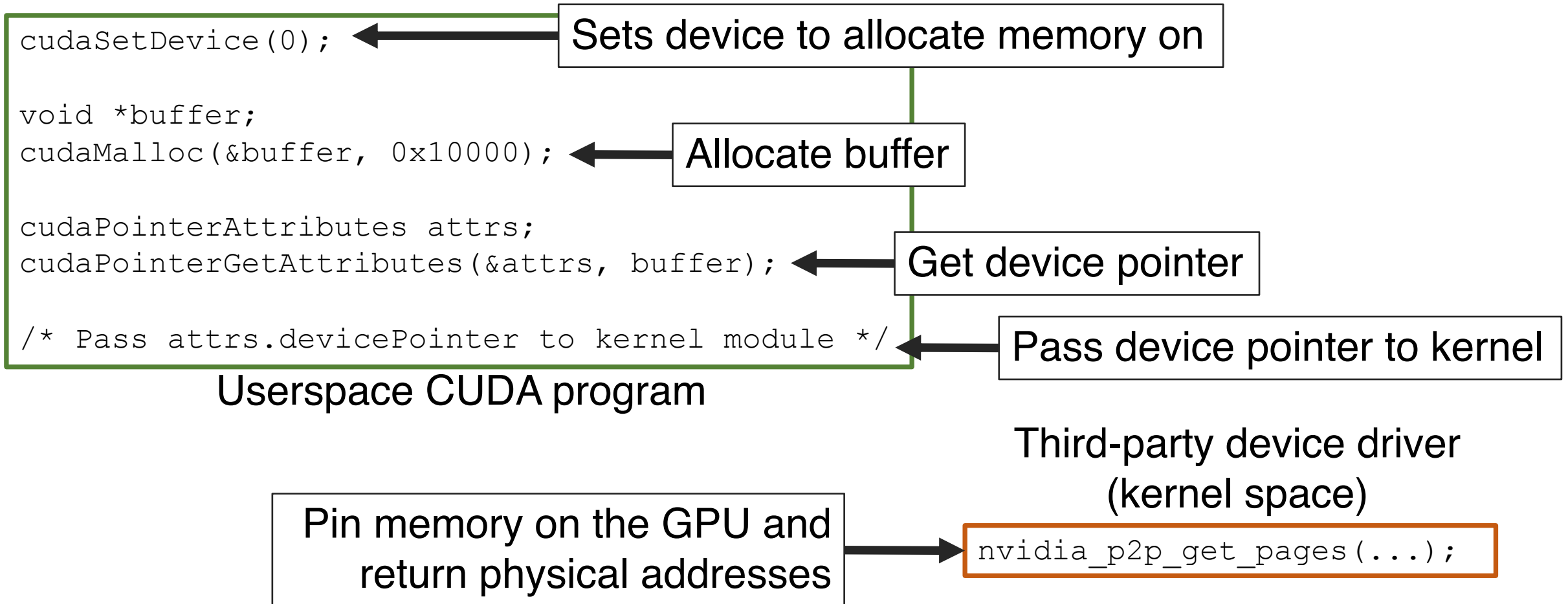
NTB Summary

- NTB connects separate independent root complexes and translating addrs between them
- Since device memory (BARs) are part of address space, we can map remote device resources for a local host

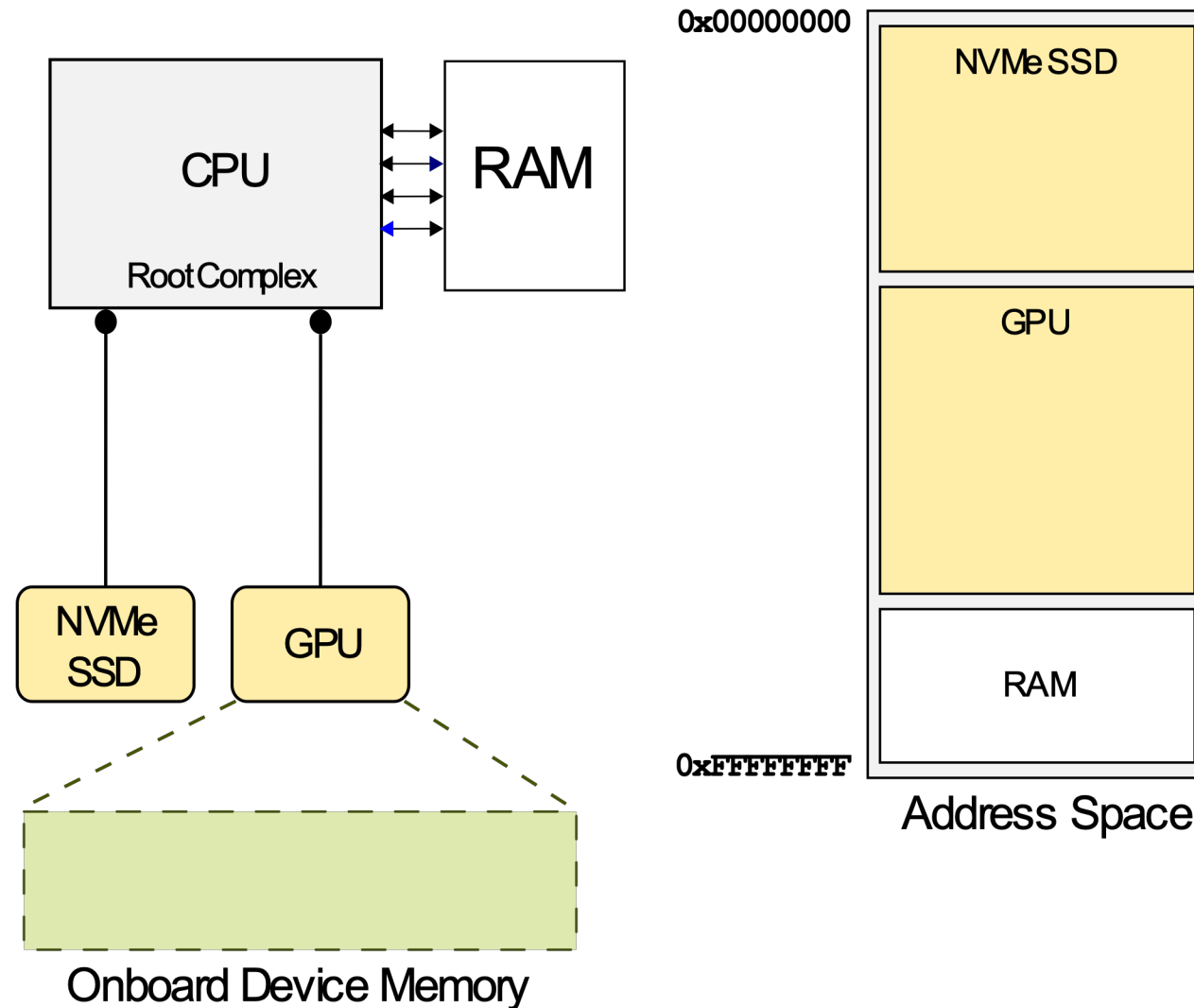
GPUDirect RDMA & Async



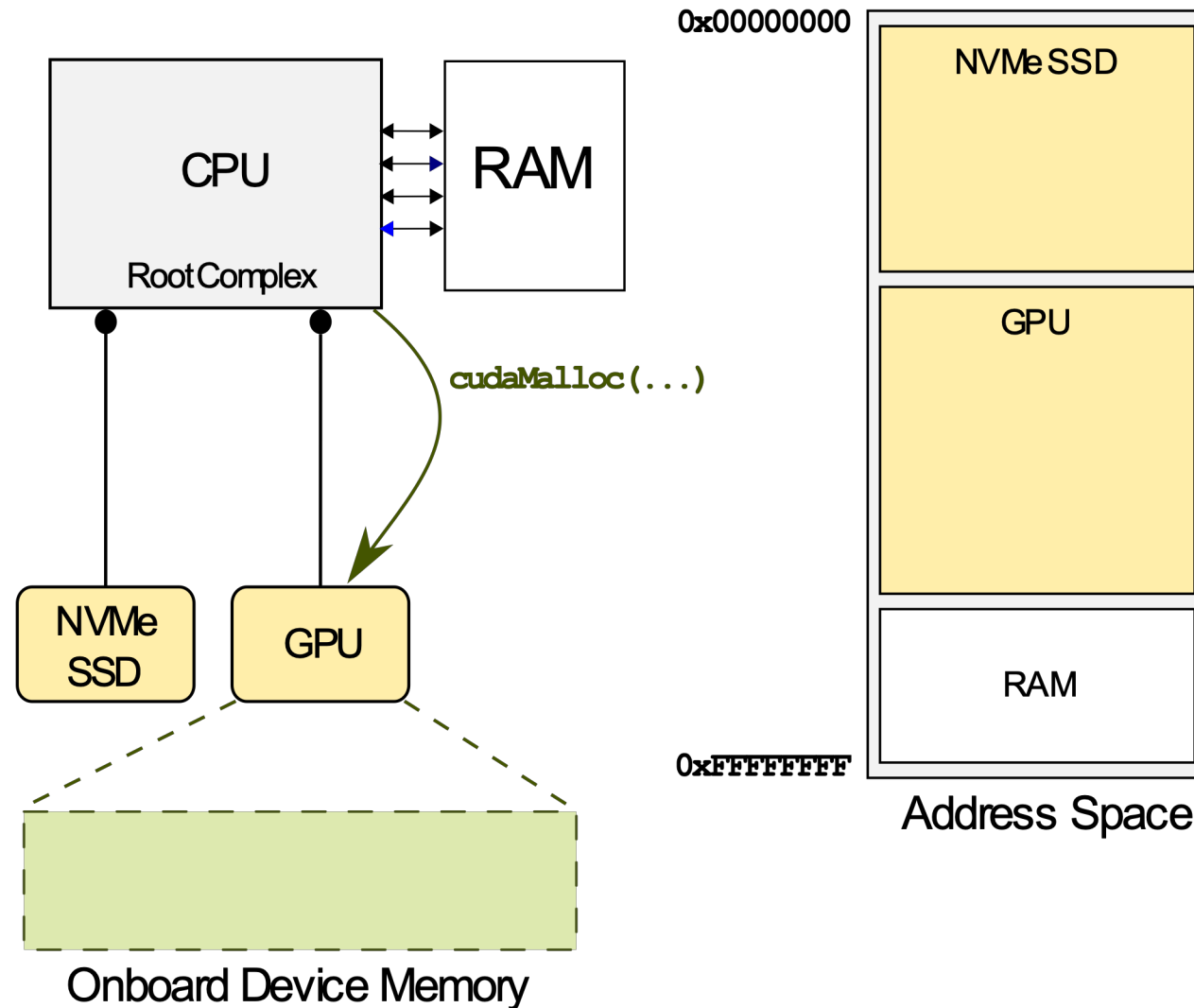
GPUDirect RDMA provides functionality to pin device memory on a GPU and exposing physical addresses of the pinned memory



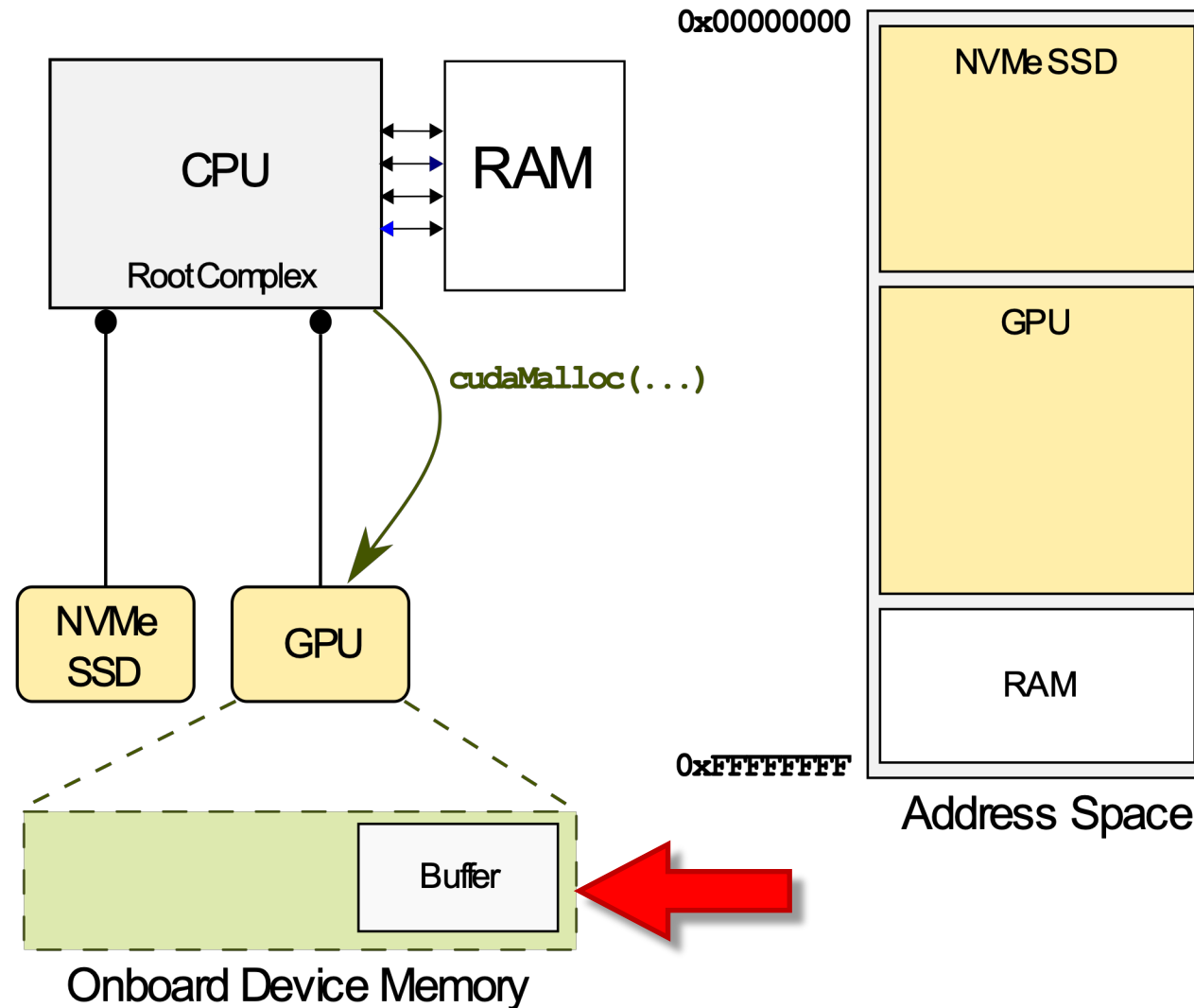
GPUDirect RDMA provides functionality to pin device memory on a GPU and exposing physical addresses of the pinned memory



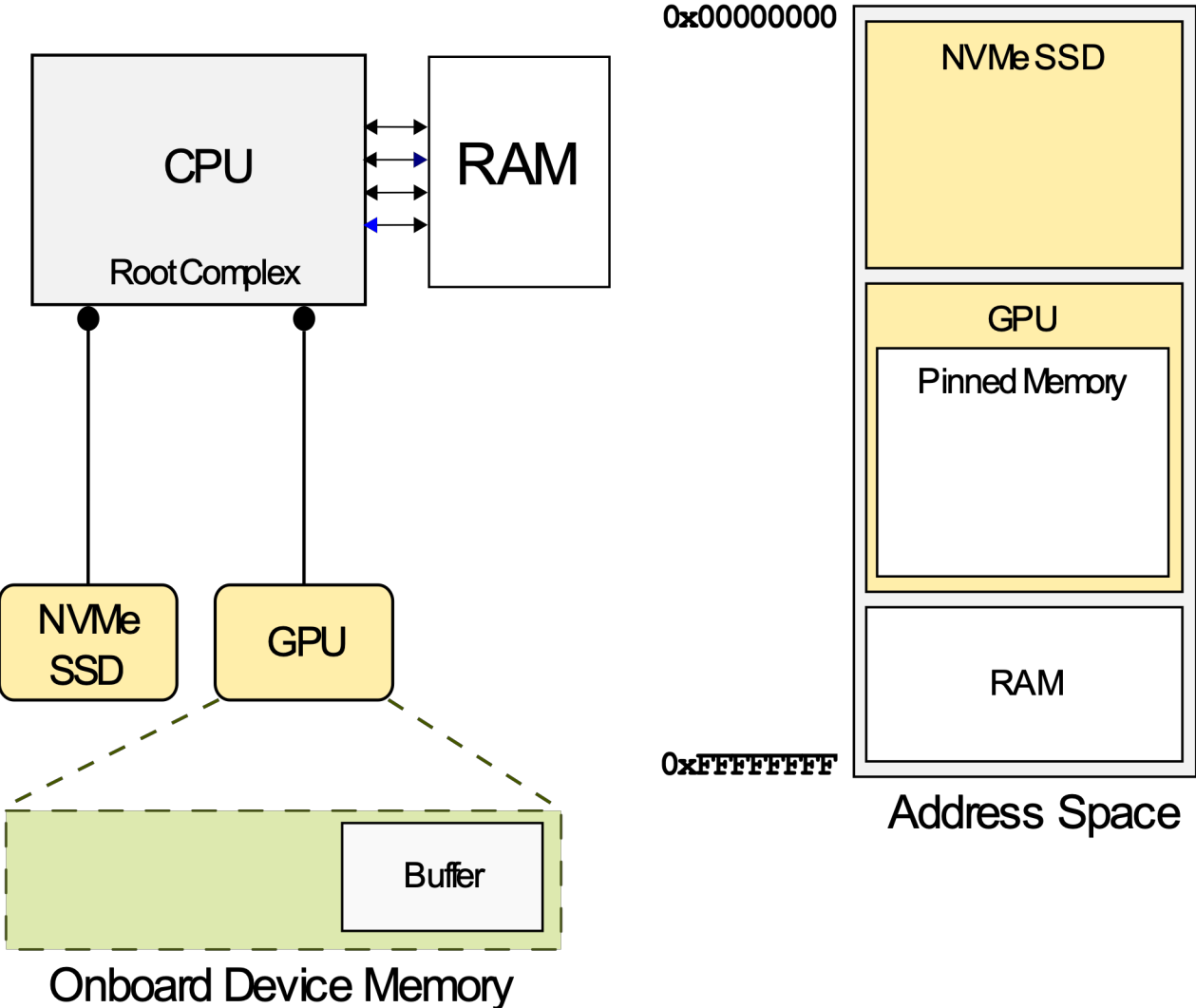
GPUDirect RDMA provides functionality to pin device memory on a GPU and exposing physical addresses of the pinned memory



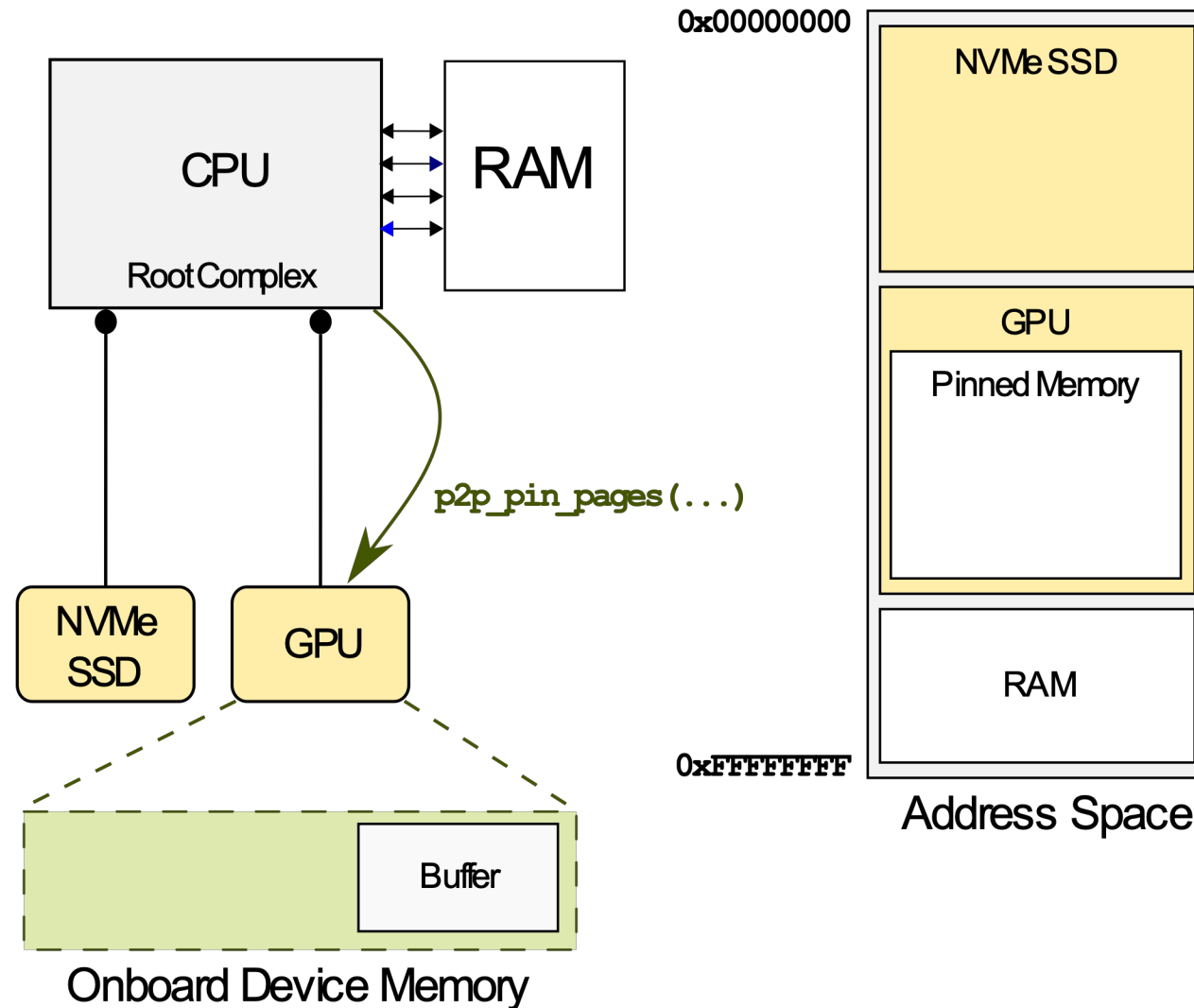
GPUDirect RDMA provides functionality to pin device memory on a GPU and exposing physical addresses of the pinned memory



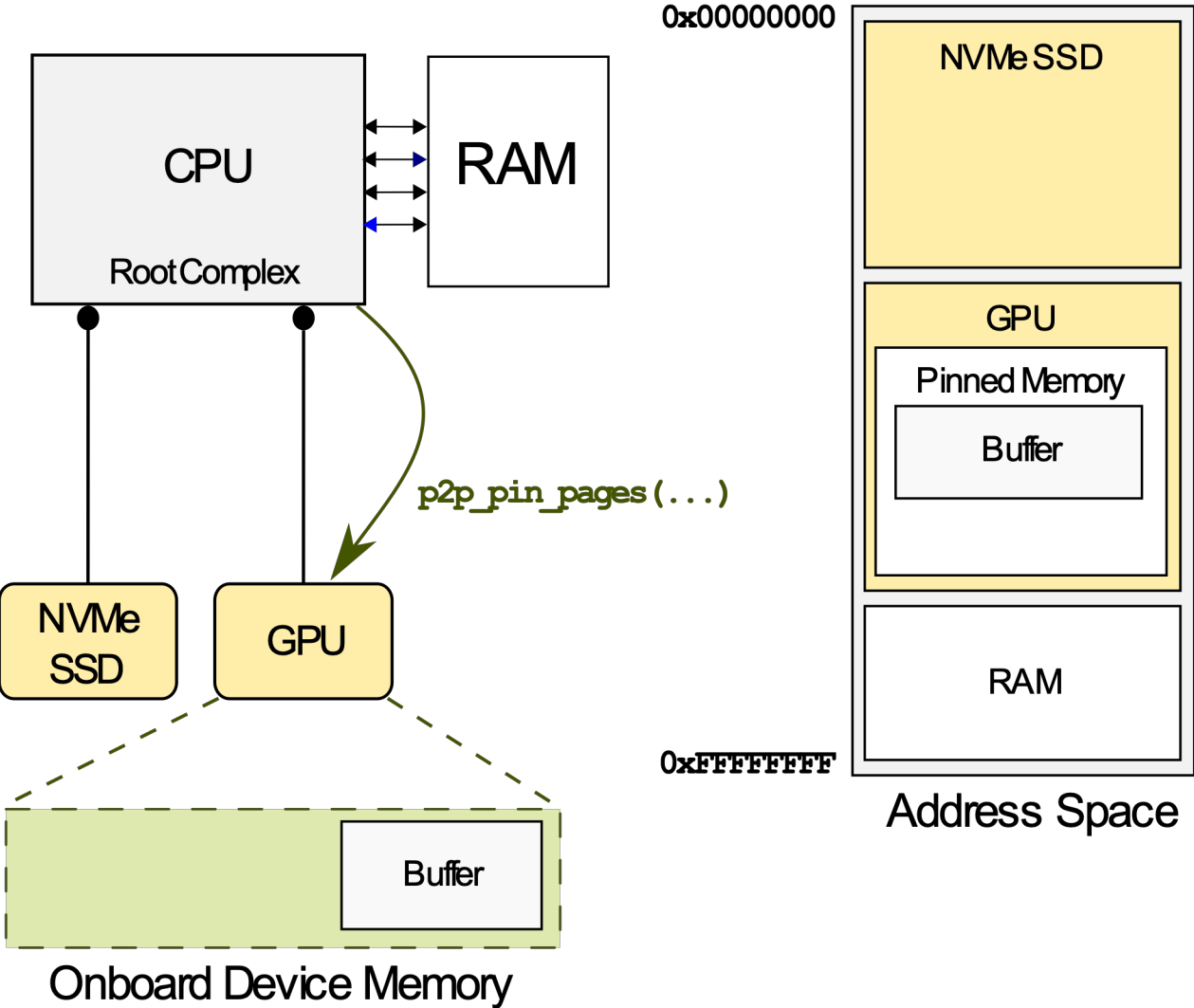
GPUDirect RDMA provides functionality to pin device memory on a GPU and exposing physical addresses of the pinned memory



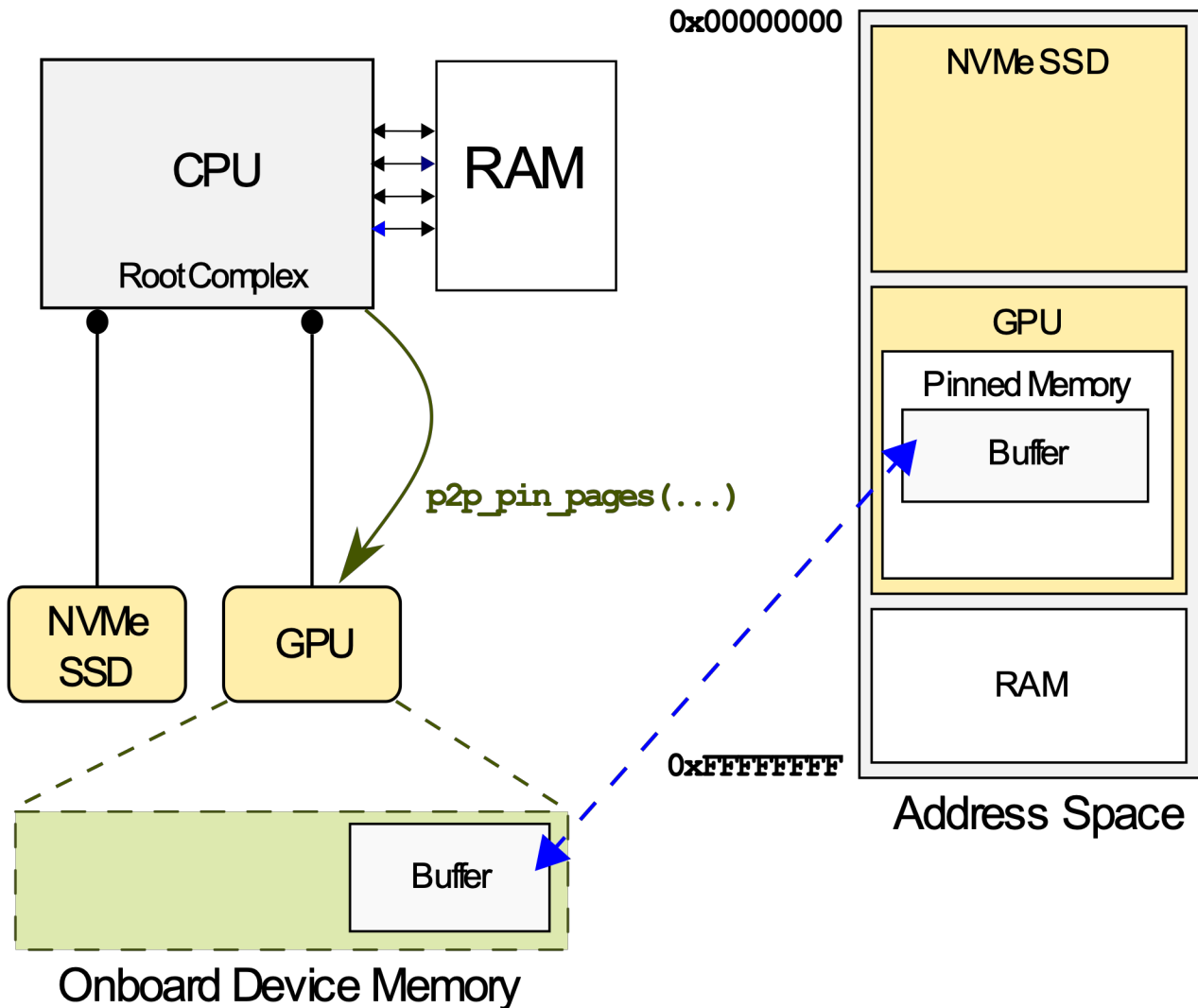
GPUDirect RDMA provides functionality to pin device memory on a GPU and exposing physical addresses of the pinned memory



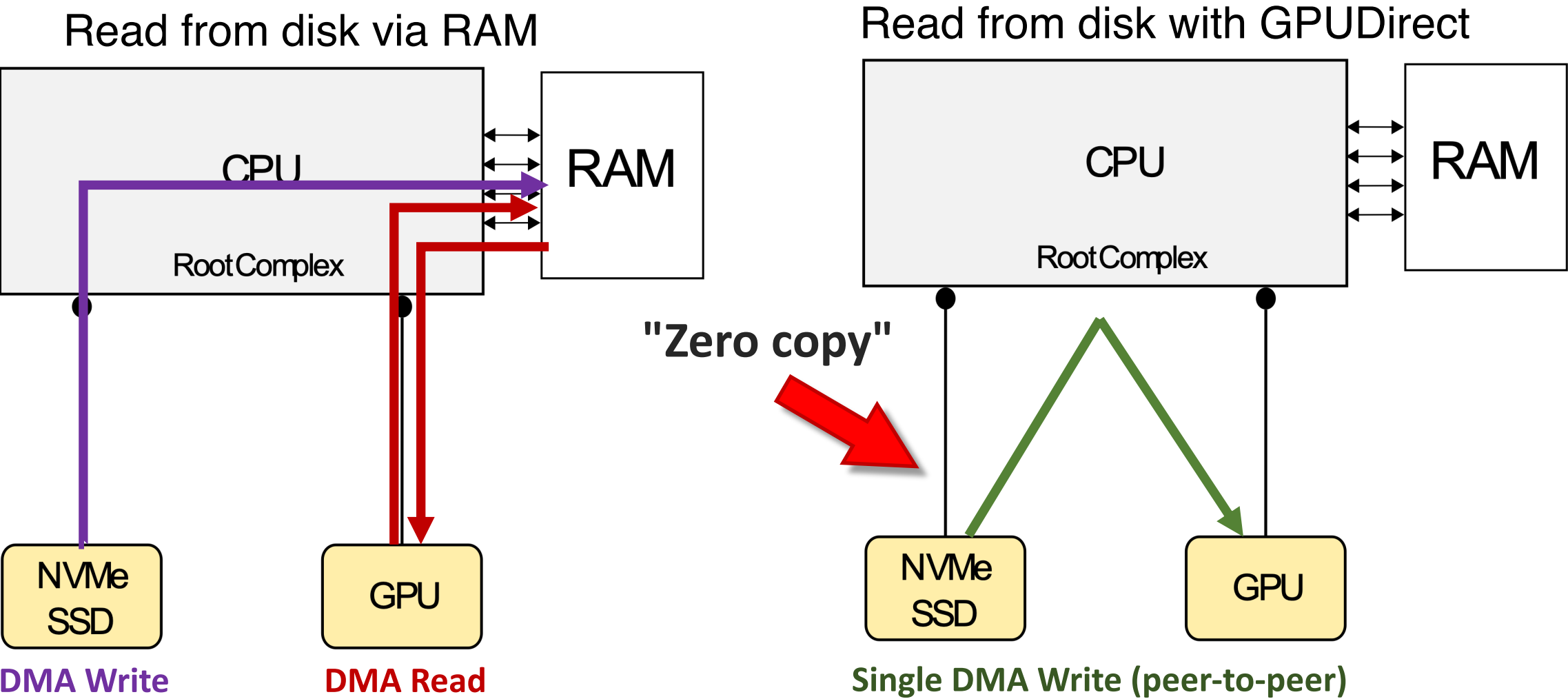
GPUDirect RDMA provides functionality to pin device memory on a GPU and exposing physical addresses of the pinned memory



GPUDirect RDMA provides functionality to pin device memory on a GPU and exposing physical addresses of the pinned memory



This allows a third-party device to read and write directly to GPU memory instead of copying to and from system memory



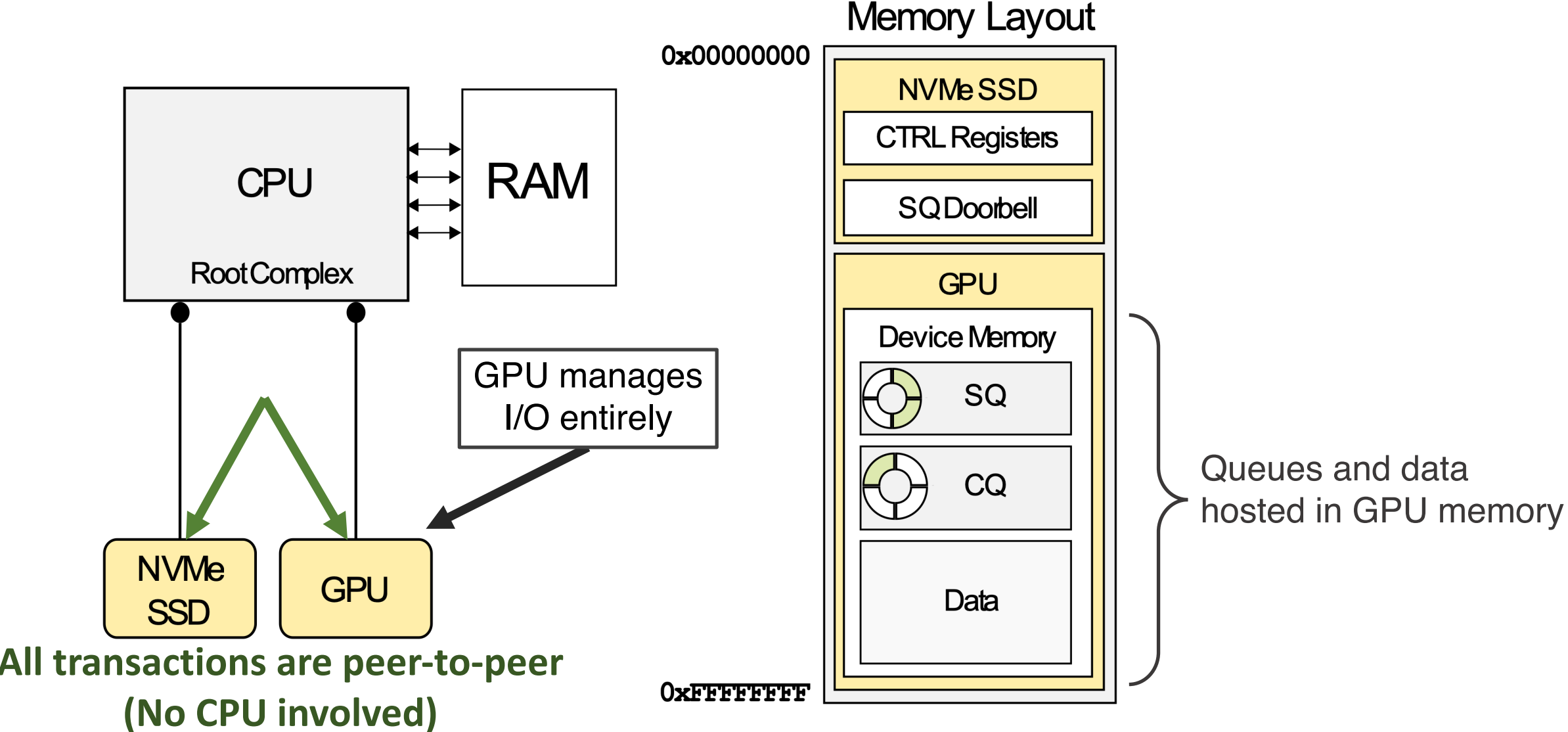
Unified Memory allows mapping controller registers and queue doorbells in to memory space managed by the CUDA driver

Maps registers in to virtual address space
(Memory mapped I/O)

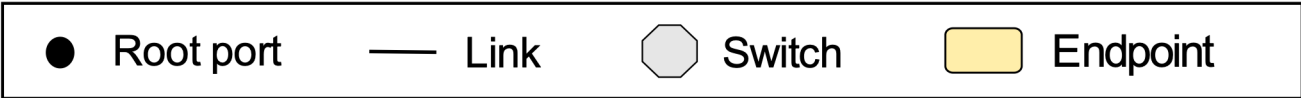
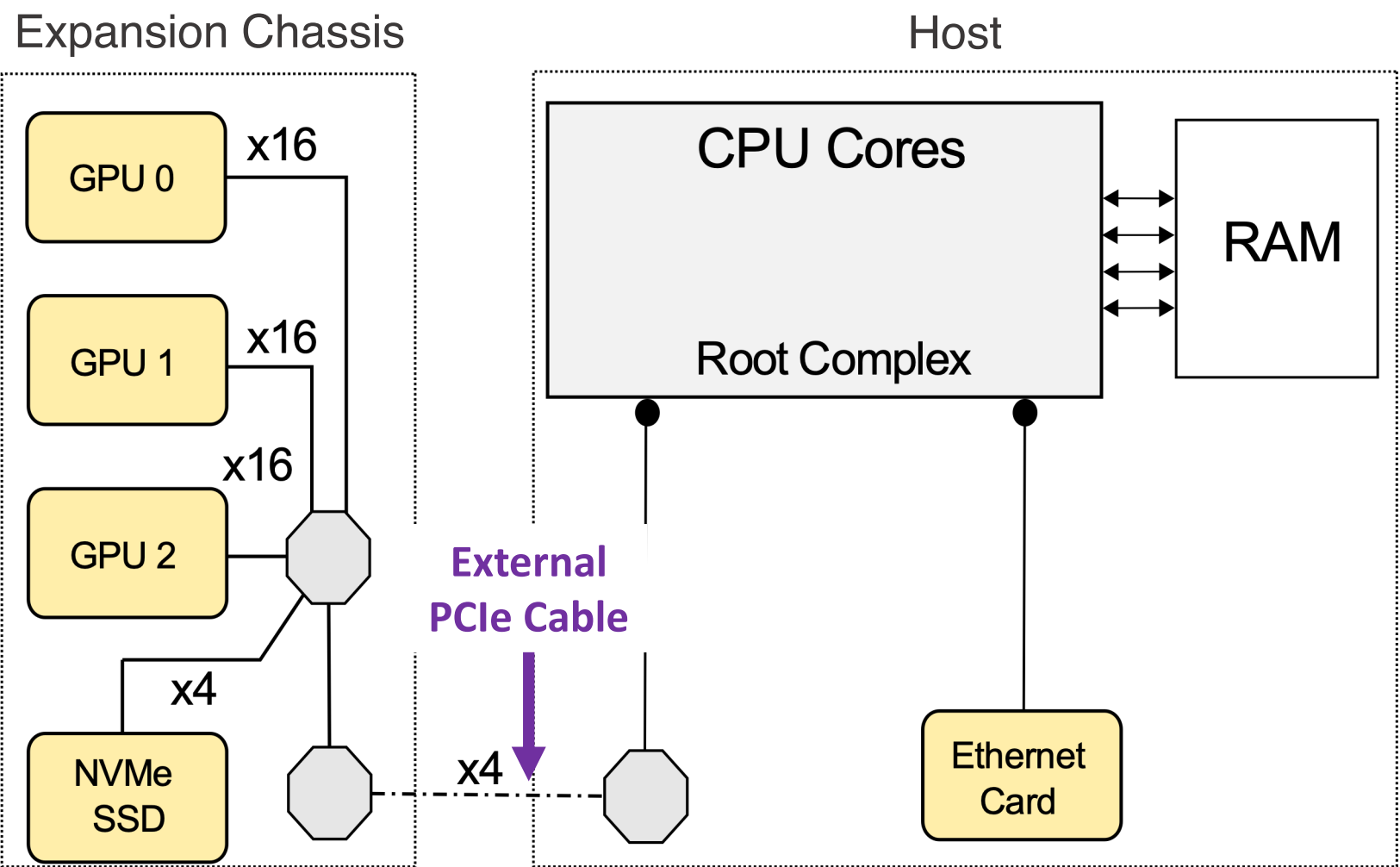
```
void *ptr = mmap(. . .);  
cudaHostRegister(ptr, 0x1000, cudaHostRegisterIoMemory);
```

Register mapped memory with CUDA

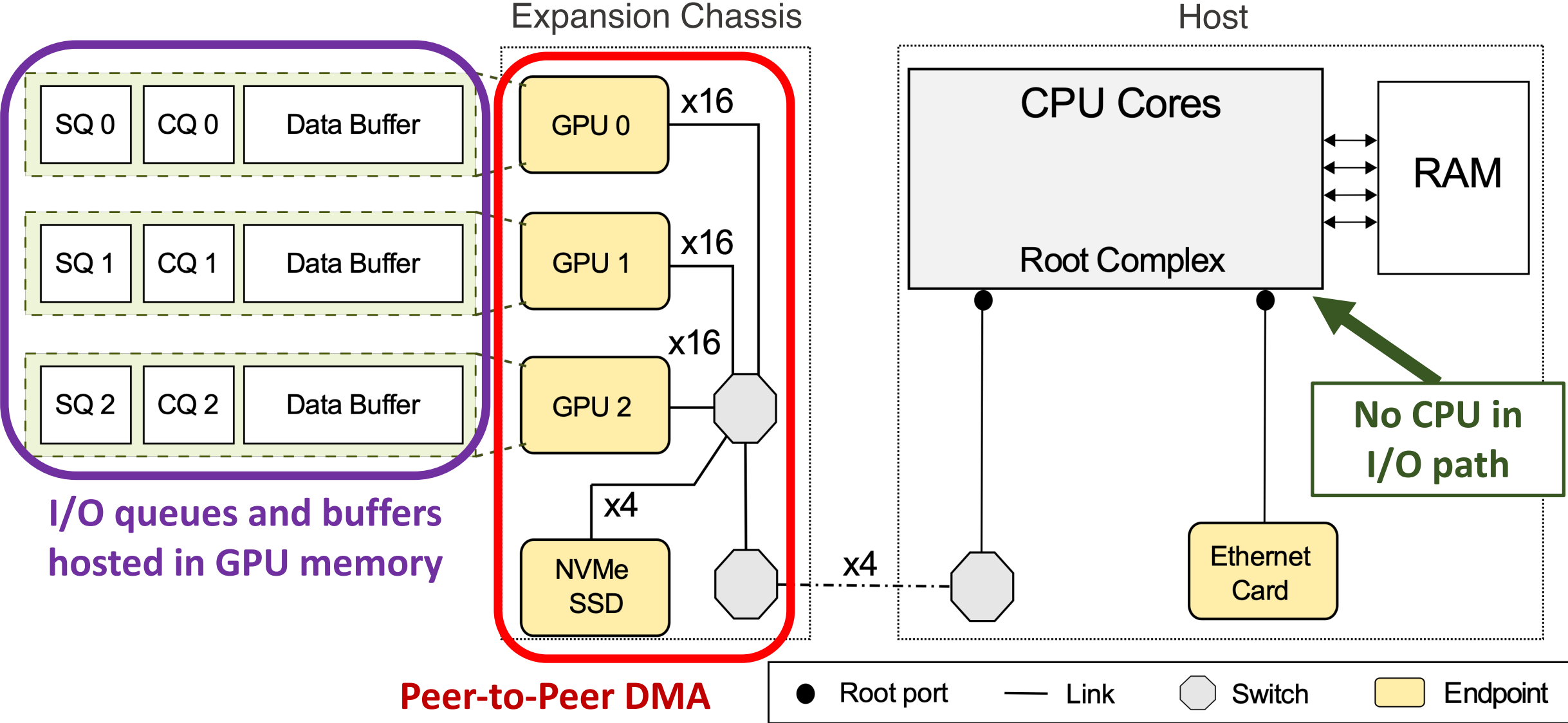
With doorbell registers mapped in to CUDA memory space, a GPU kernel can now trigger doorbell writes using GPUDirect Async



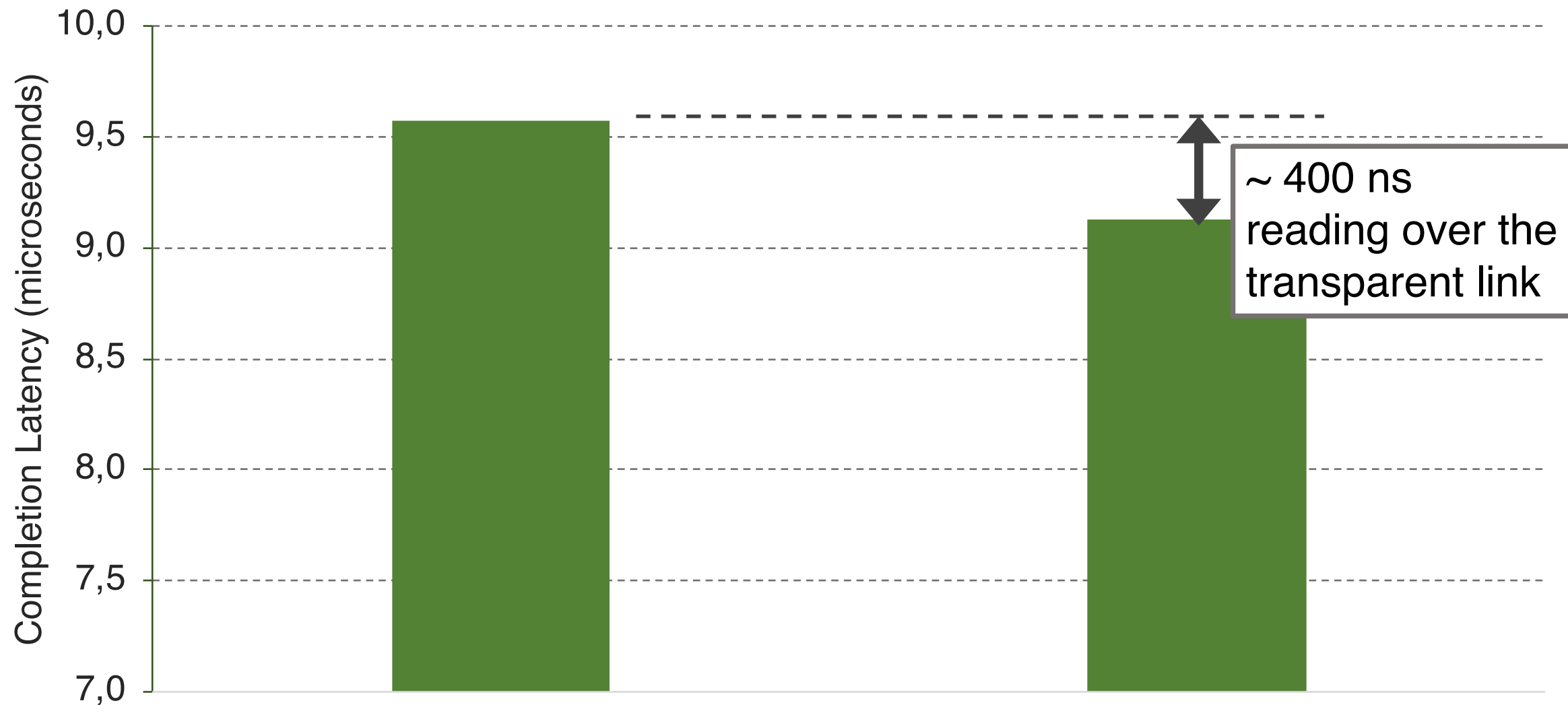
By assigning I/O queues to each individual GPU, multiple GPUs can share a single NVMe disk simultaneously



By assigning I/O queues to each individual GPU, multiple GPUs can share a single NVMe disk simultaneously



97th Percentile Latency (1 PRP = 4 kB, Random Read, 128 MB file)



Disk: Intel P4800X

GPU: Quadro P620

RAM: DDR4 2133 MHz

CPU: Intel Xeon E5-2603 v4

Submission Queue Memory Location

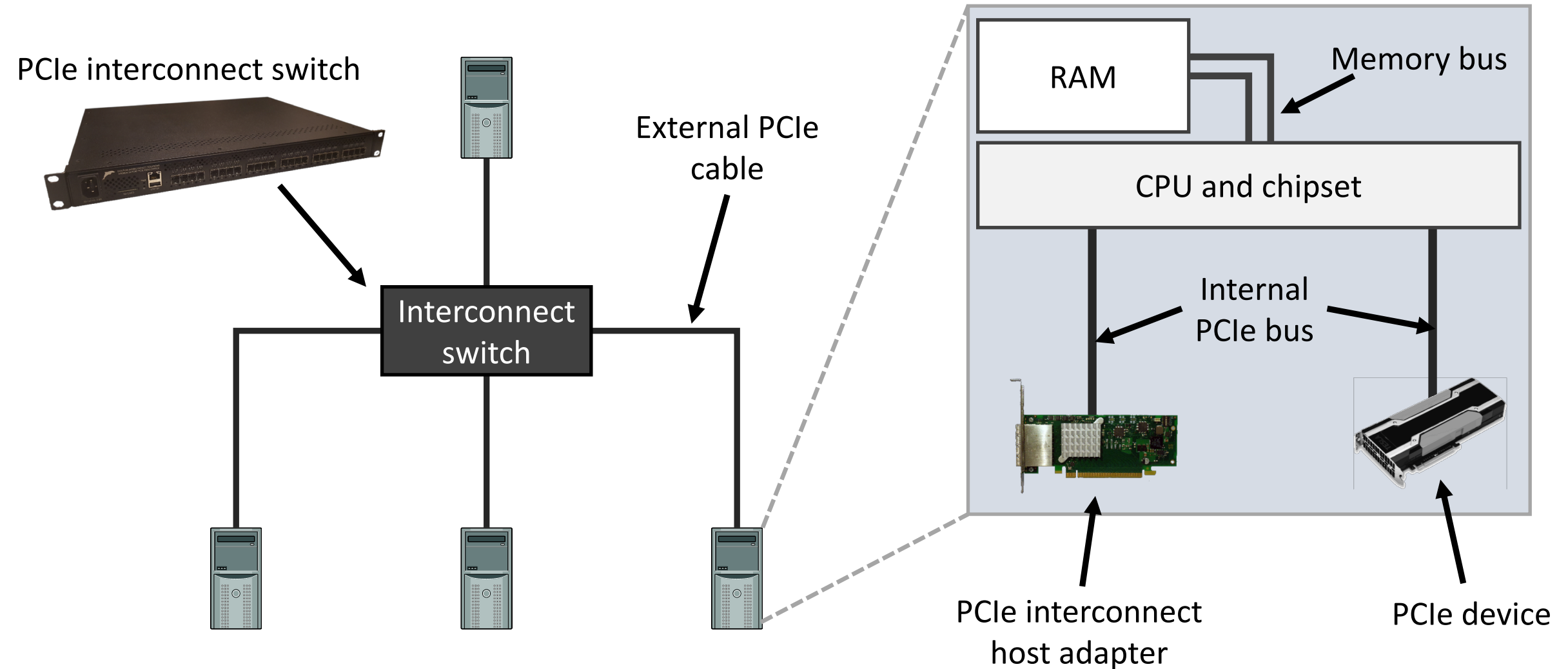
GPUDirect Summary

- GPUDirect DMA allows third-party devices, such as NVMe disks, to access GPU memory directly
- GPUDirect Async allows memory-mapped I/O from a CUDA kernel = eliminate CPU in I/O path entirely
- We have used these to make a distributed NVMe driver in CUDA kernel code

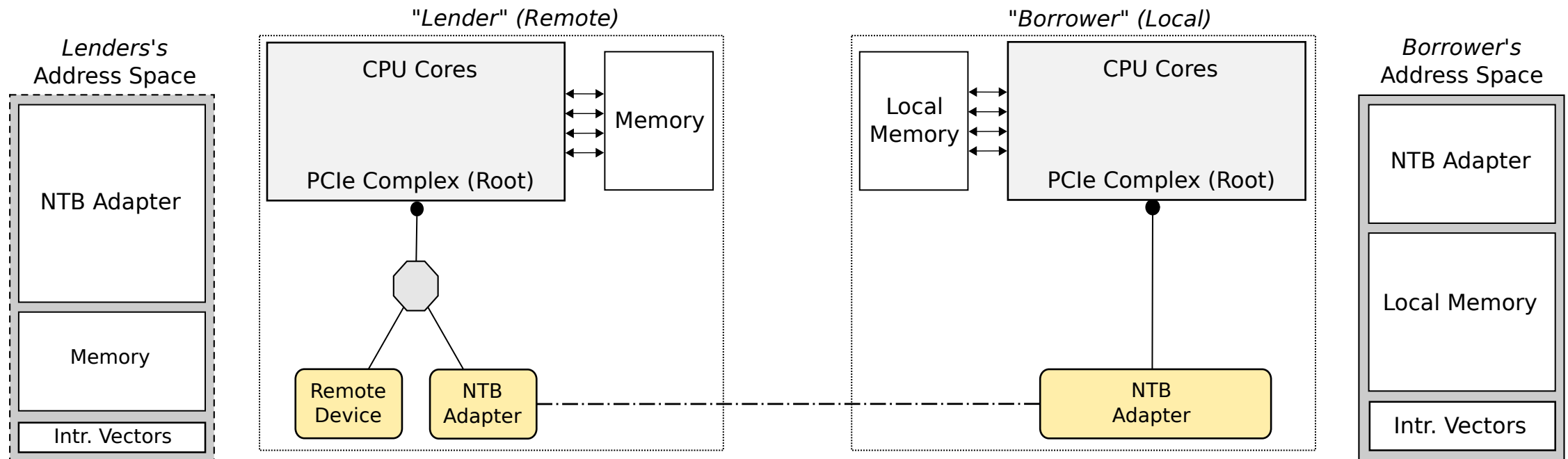
Device Lending and SmartIO



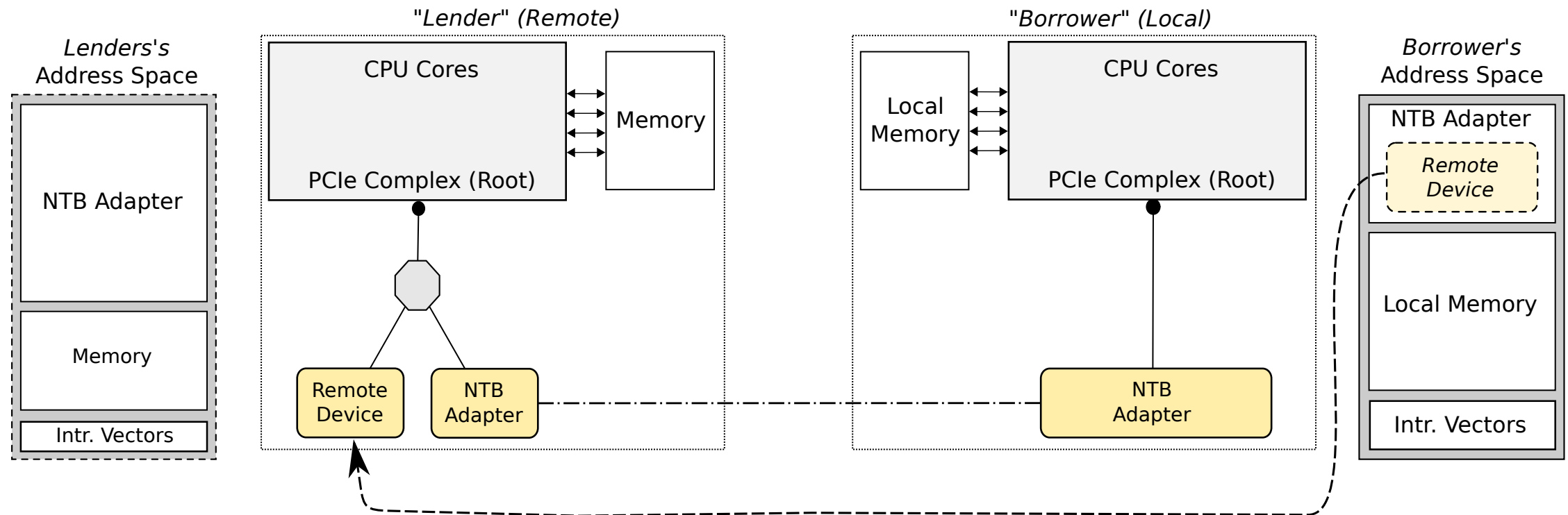
In PCIe clusters, the same fabric is used both for interconnecting hosts as well as the local I/O bus inside each host



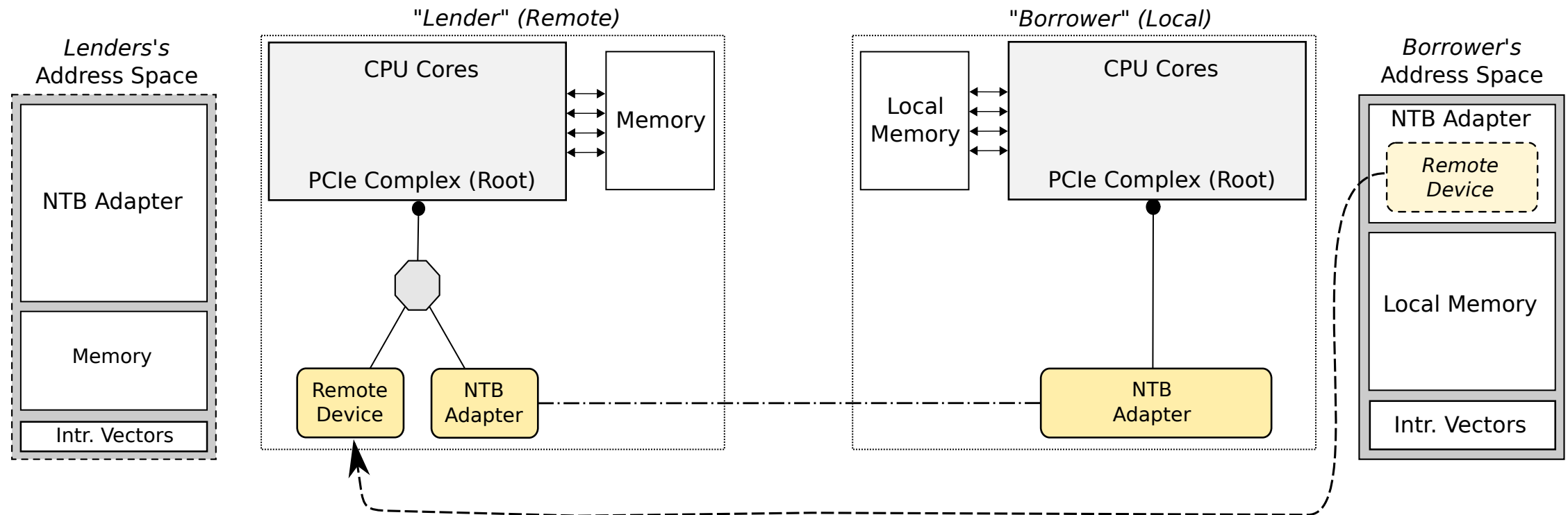
Using an NTB, it is possible to map remote device memory regions (BARs) for a local host



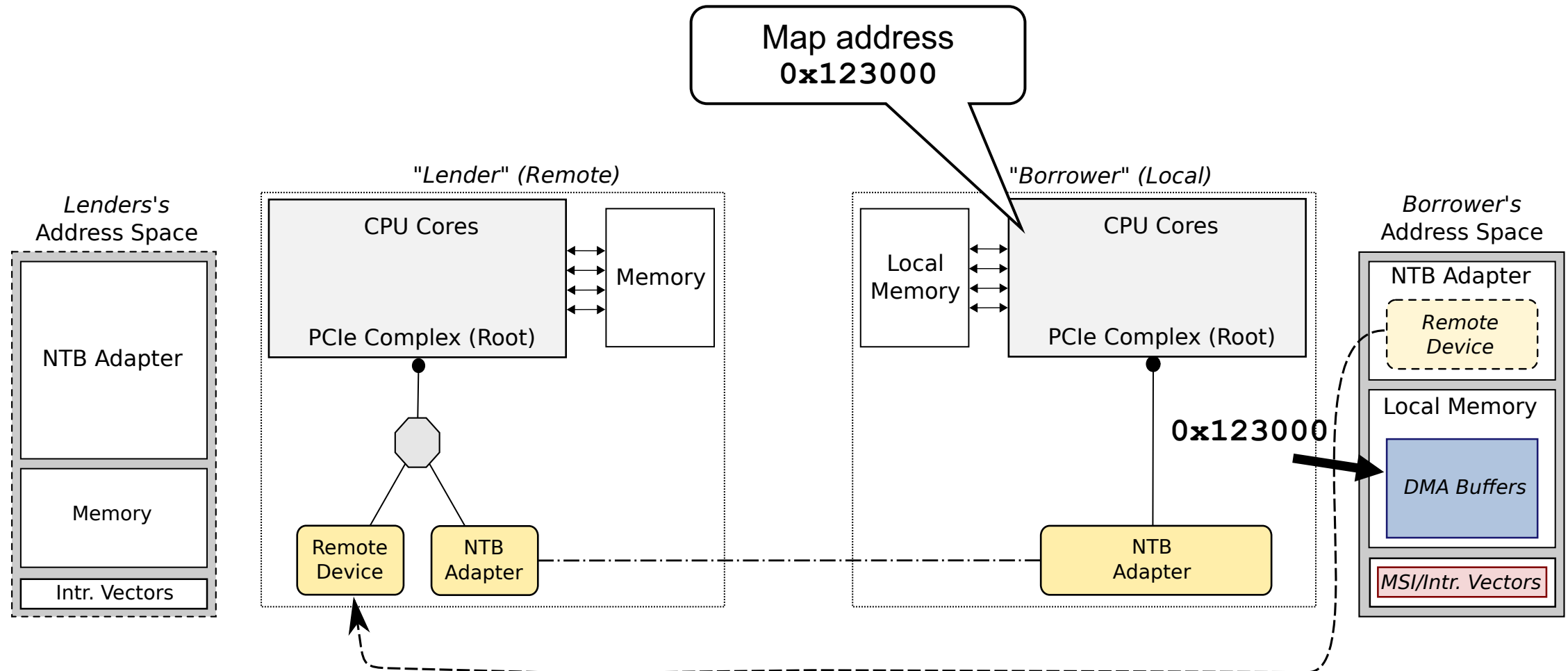
Using an NTB, it is possible to map remote device memory regions (BARs) for a local host



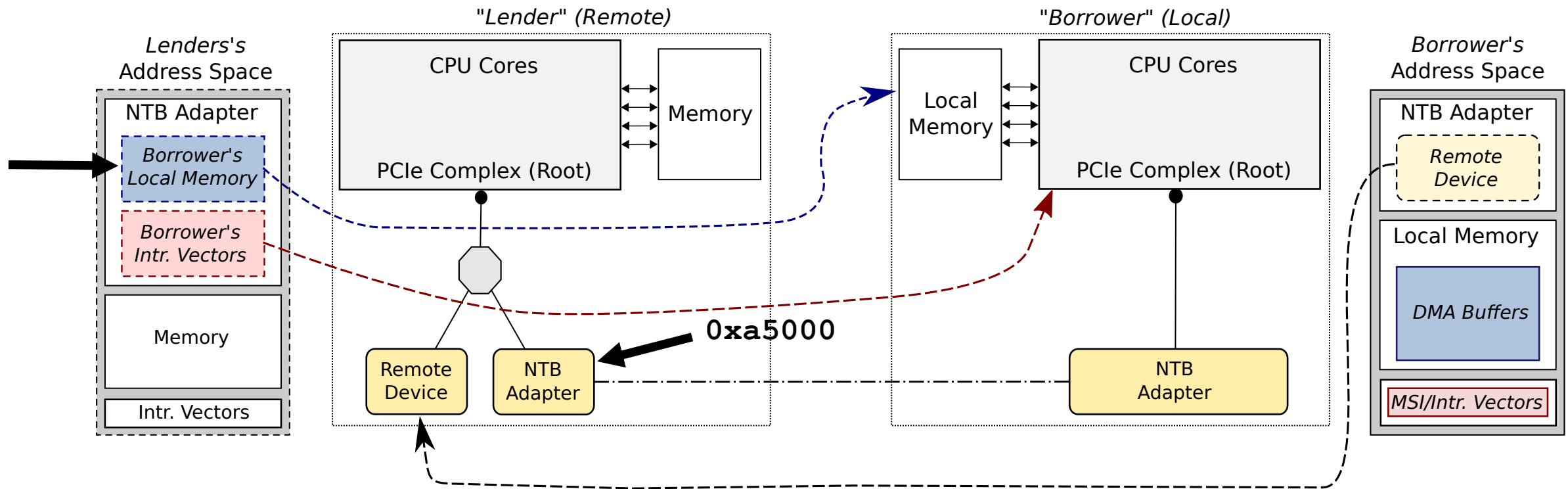
The remote system can in turn reverse-map the local system's memory and interrupt addresses for the device



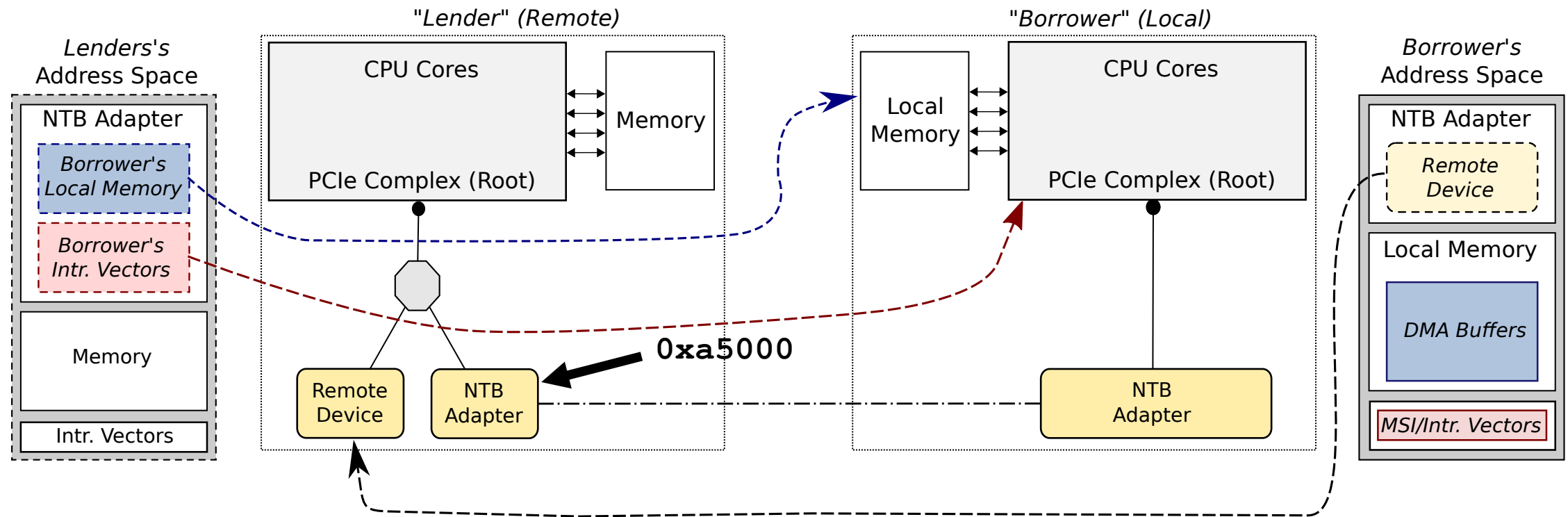
The remote system can in turn reverse-map the local system's memory and interrupt addresses for the device



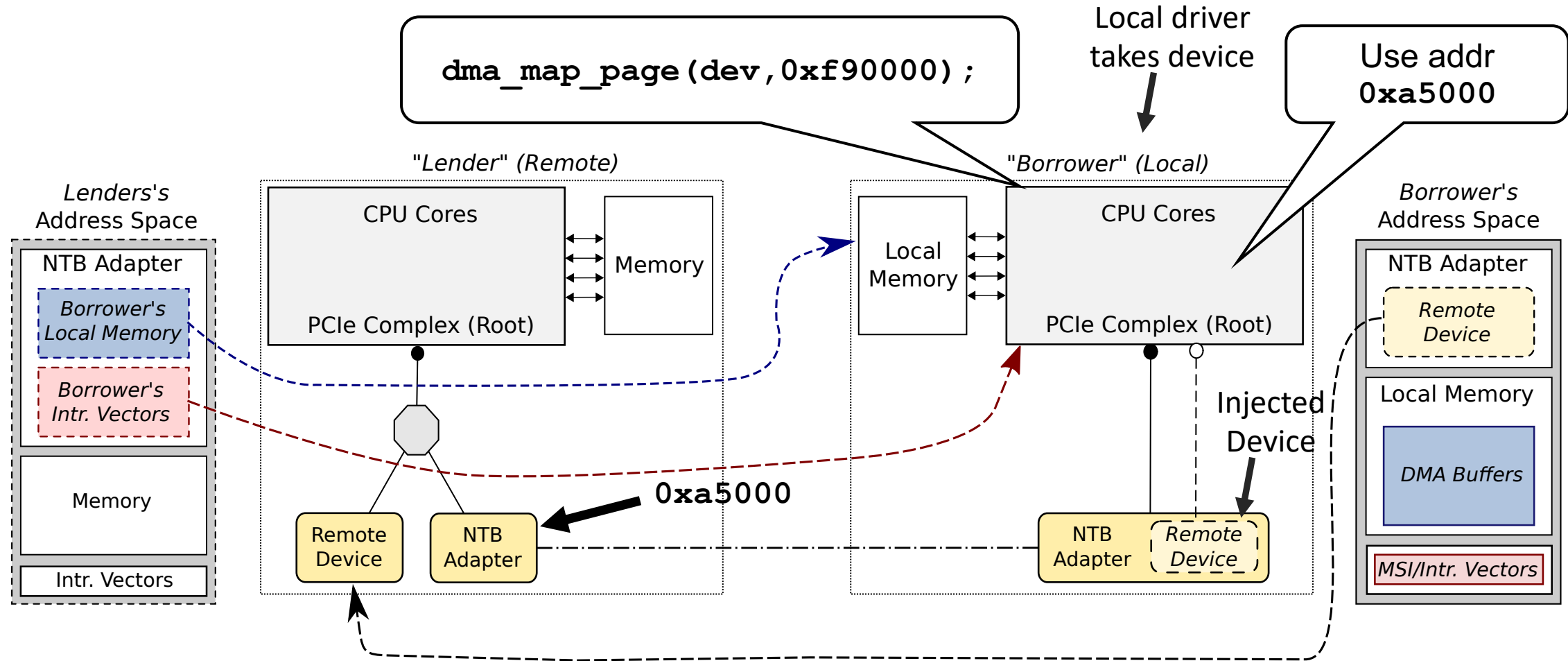
The remote system can in turn reverse-map the local system's memory and interrupt addresses for the device

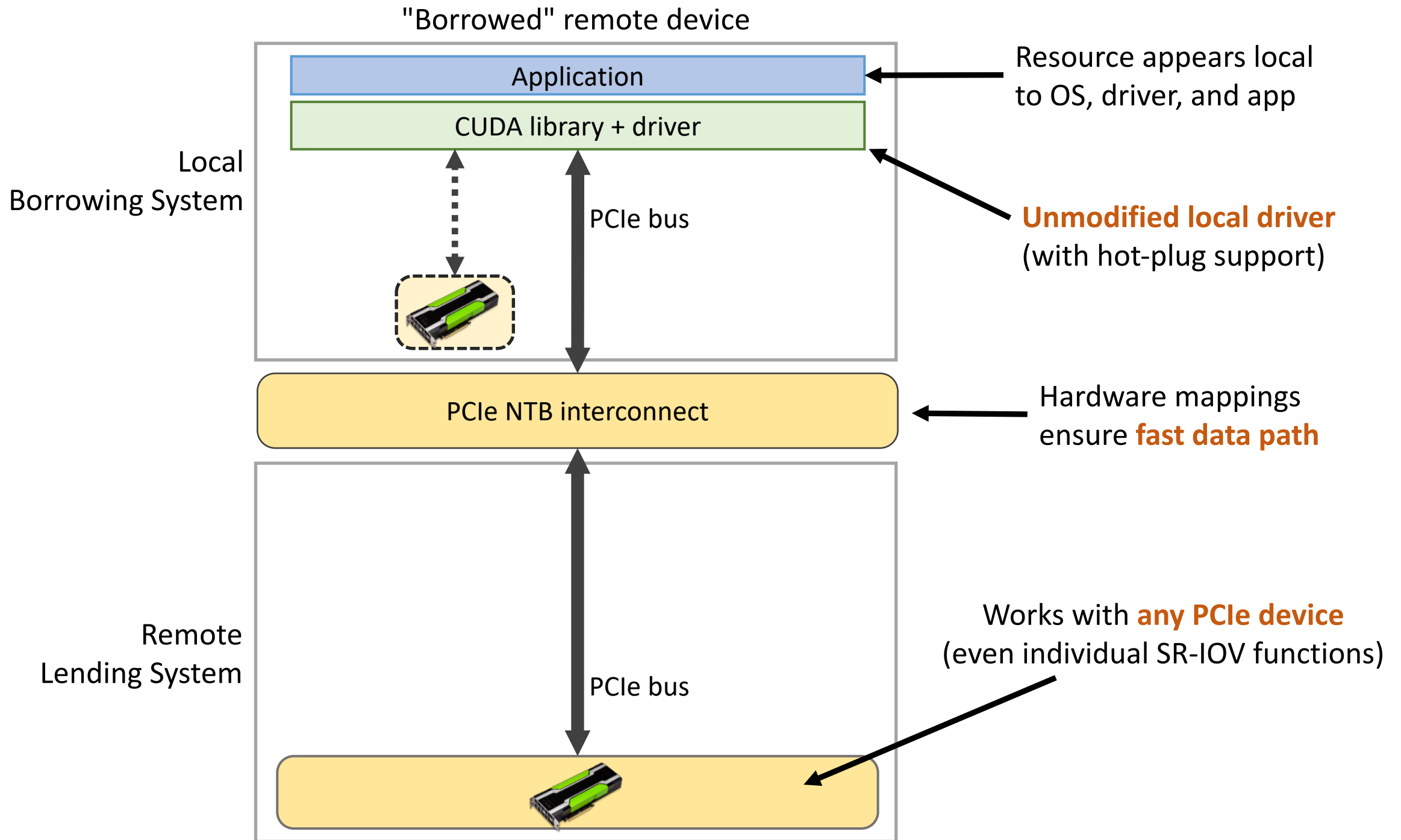


By emulating a PCIe hot-add event, the remote device is inserted into the kernel device tree, making it appear locally installed

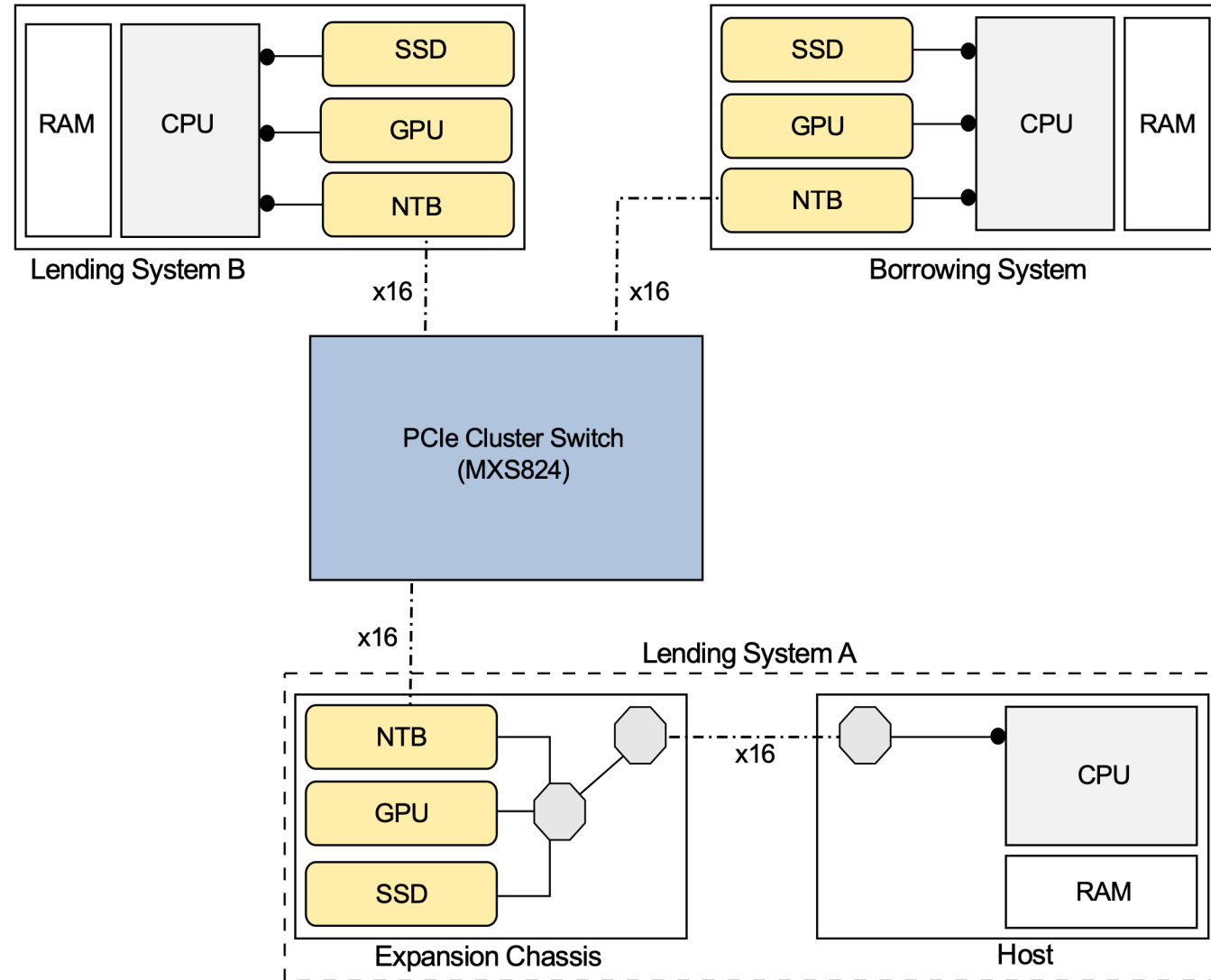


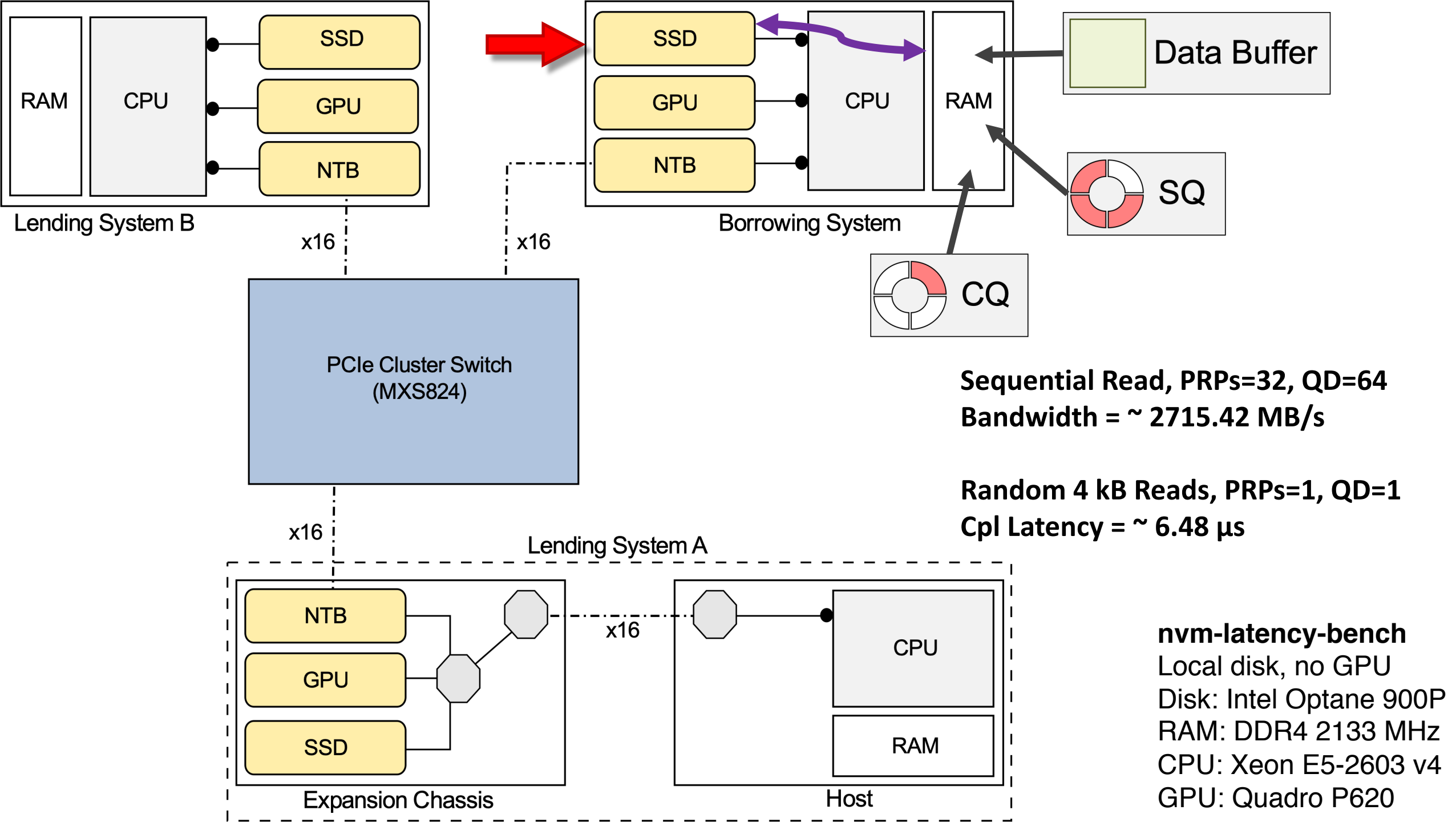
By emulating a PCIe hot-add event, the remote device is inserted into the kernel device tree, making it appear locally installed

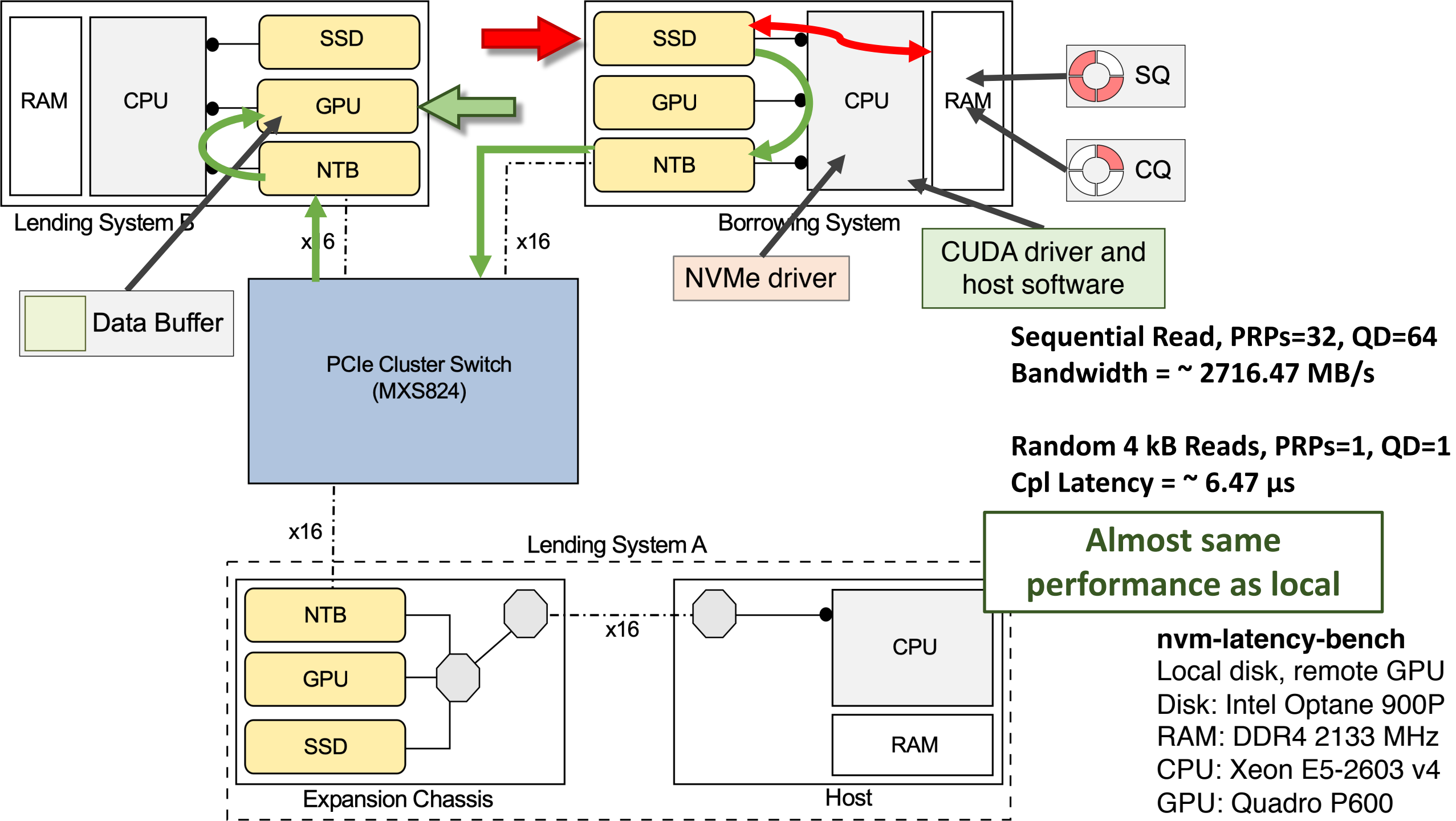


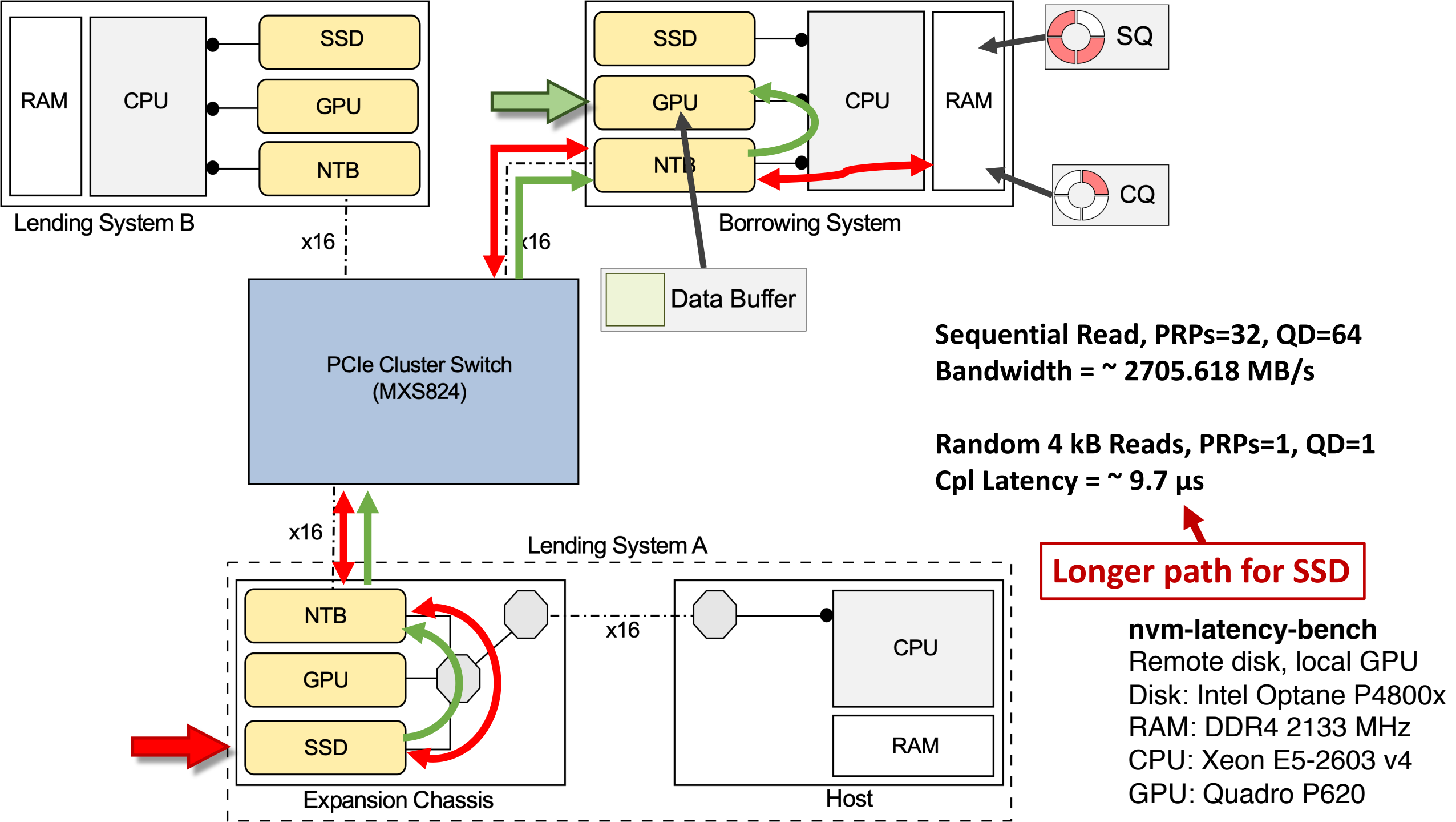


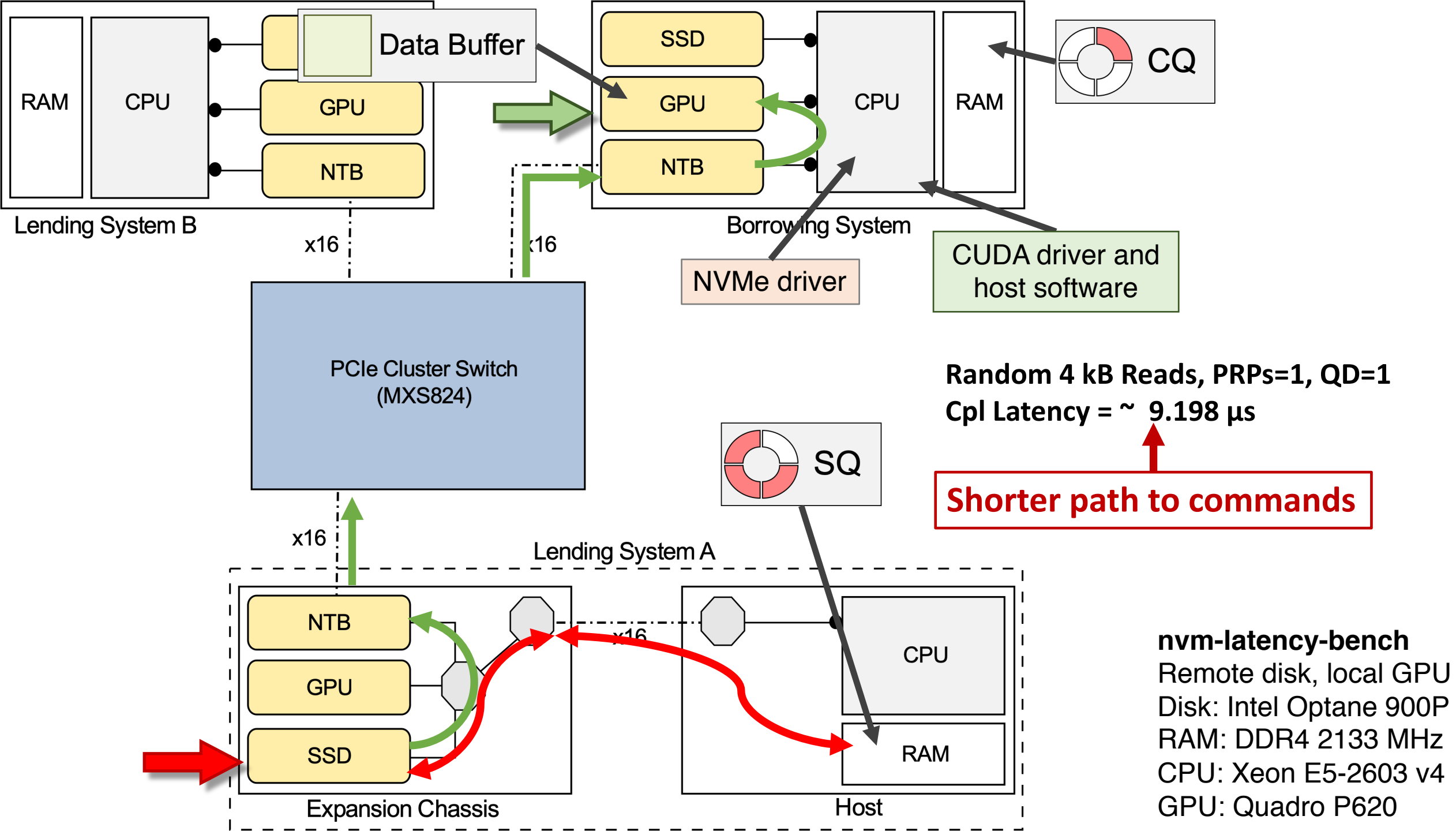
Using Device Lending and our CUDA NVMe driver, it is possible to use create highly flexible and distributed I/O workloads

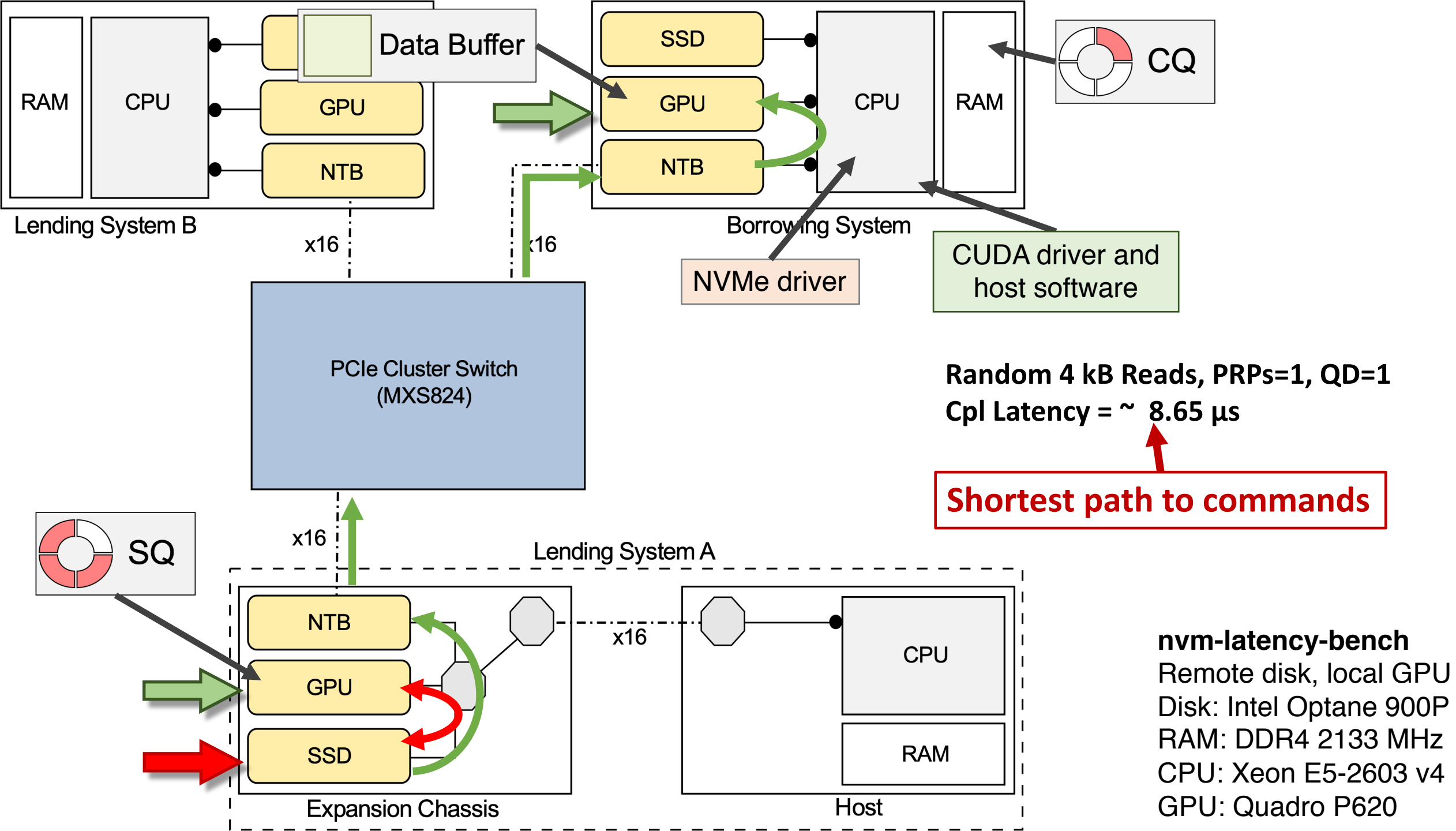




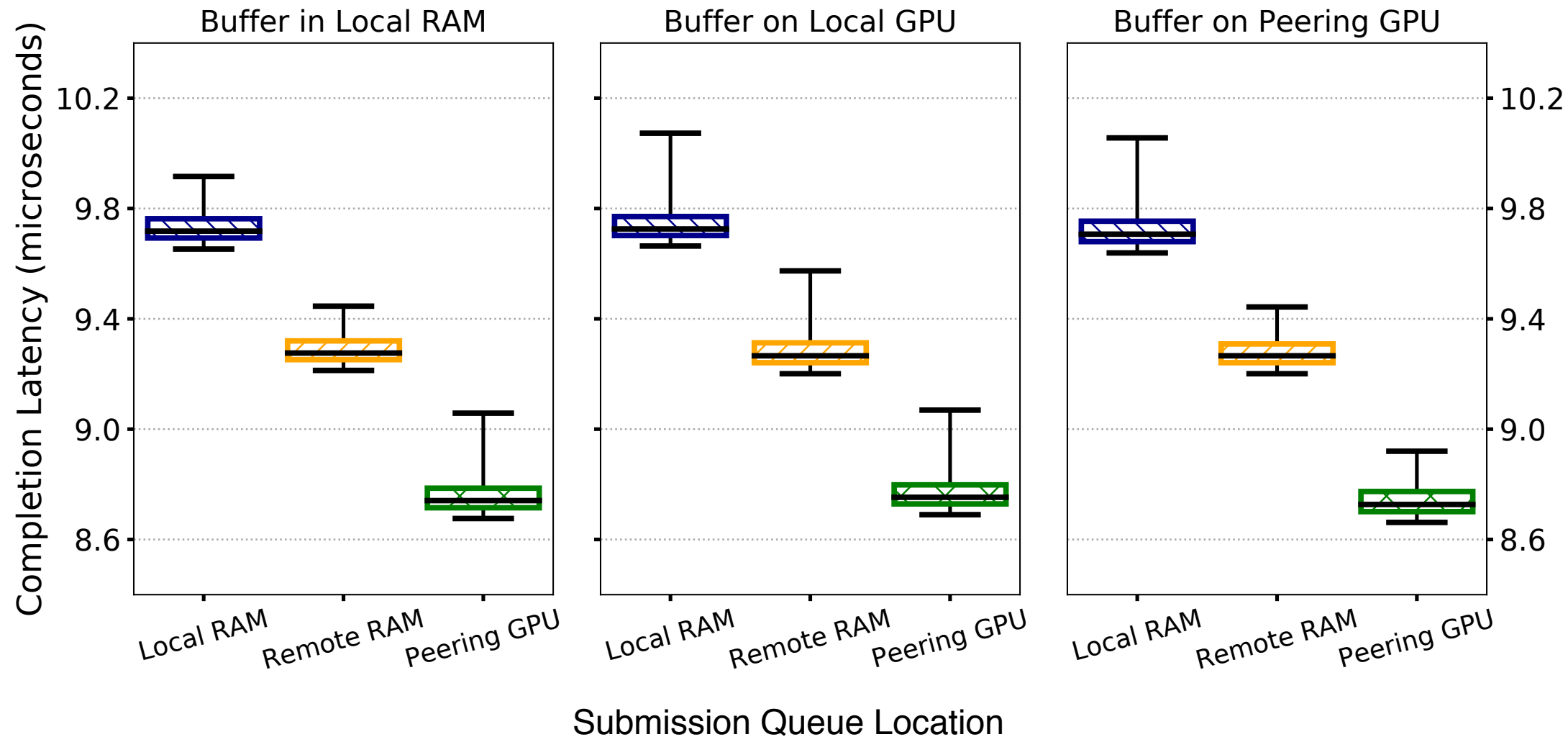


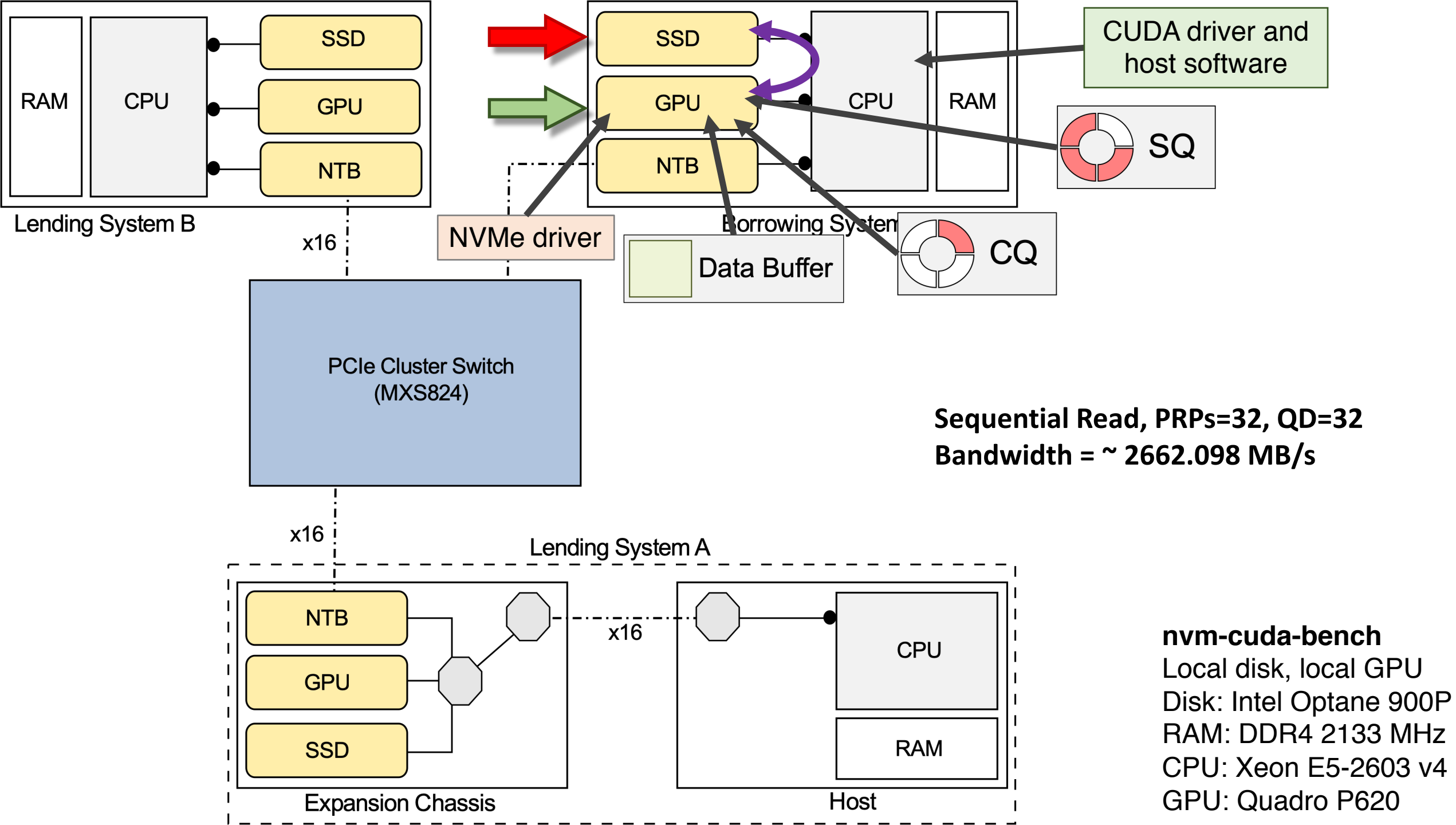


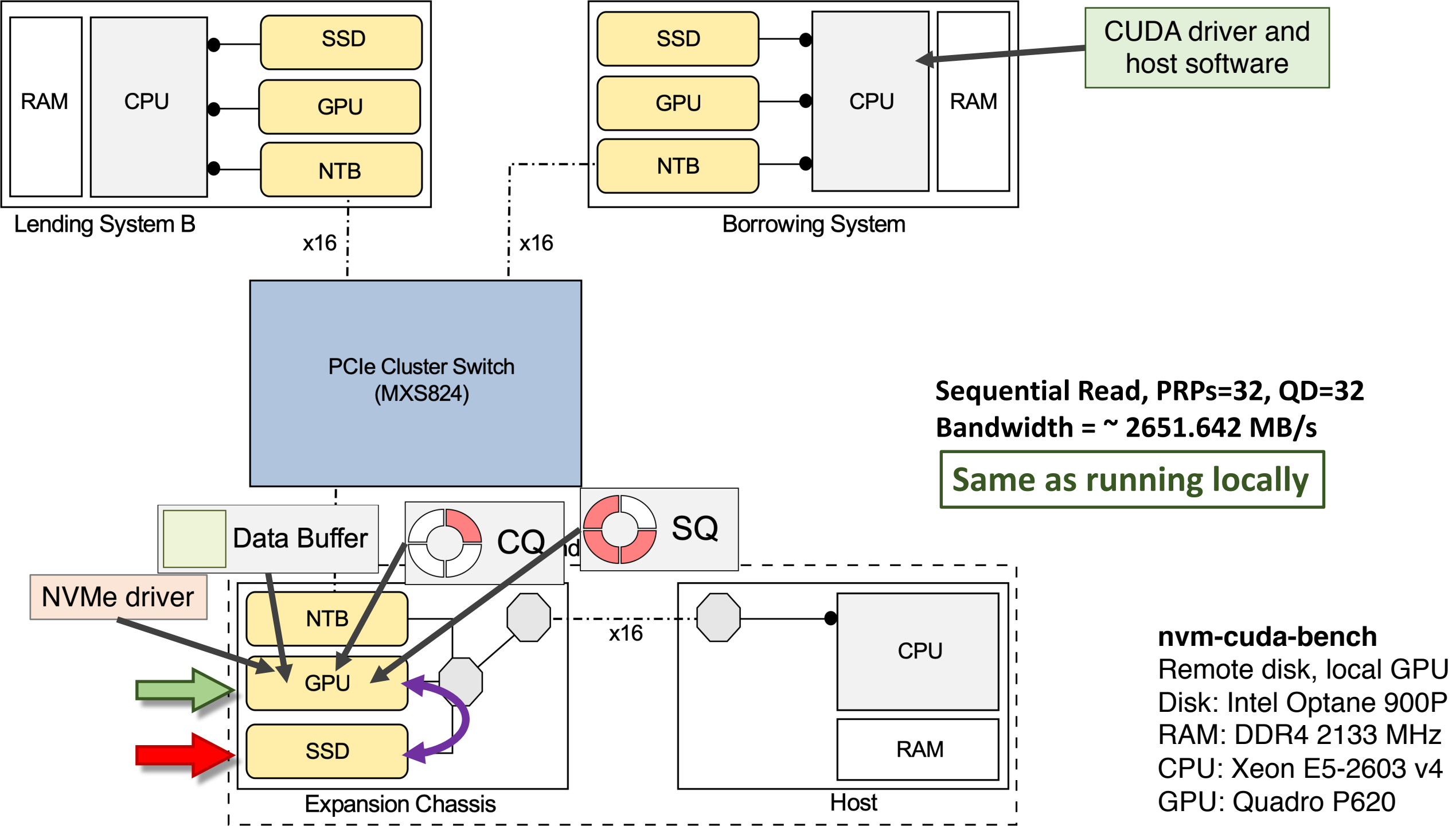




Command Submission Latency
Random Reads, Queue Depth=1, PRPs per Command=1

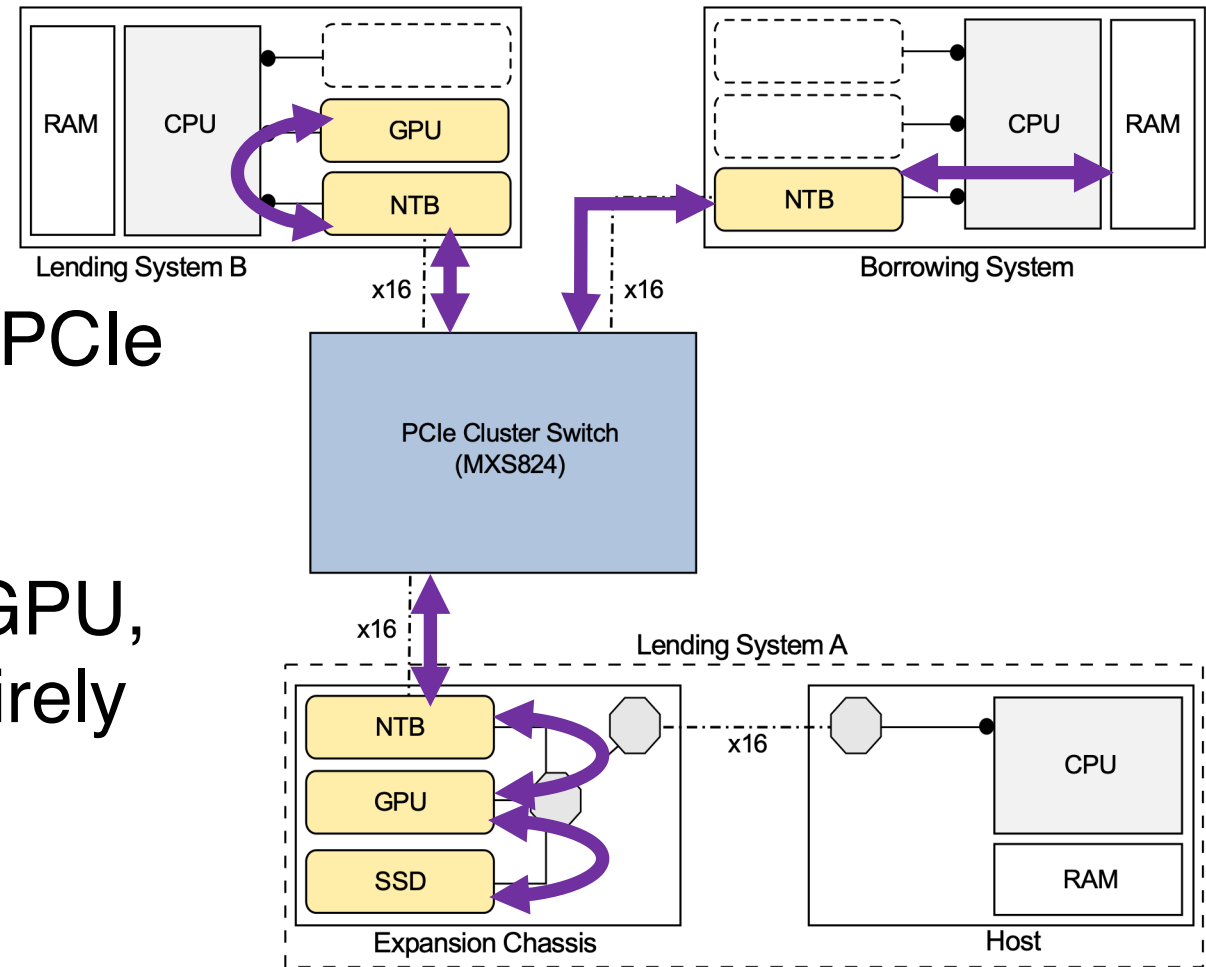






Summary

- Optimized data transfer paths using PCIe peer-to-peer transactions
- Direct block-level disk access from GPU, eliminating CPU in I/O data path entirely
- Concurrently sharing NVMe drives between multiple hosts and GPUs
- PCIe non-transparent bridges offer great flexibility in dynamic device configurations



Thank you!

jonassm@dolphinics.com

My e-mail address

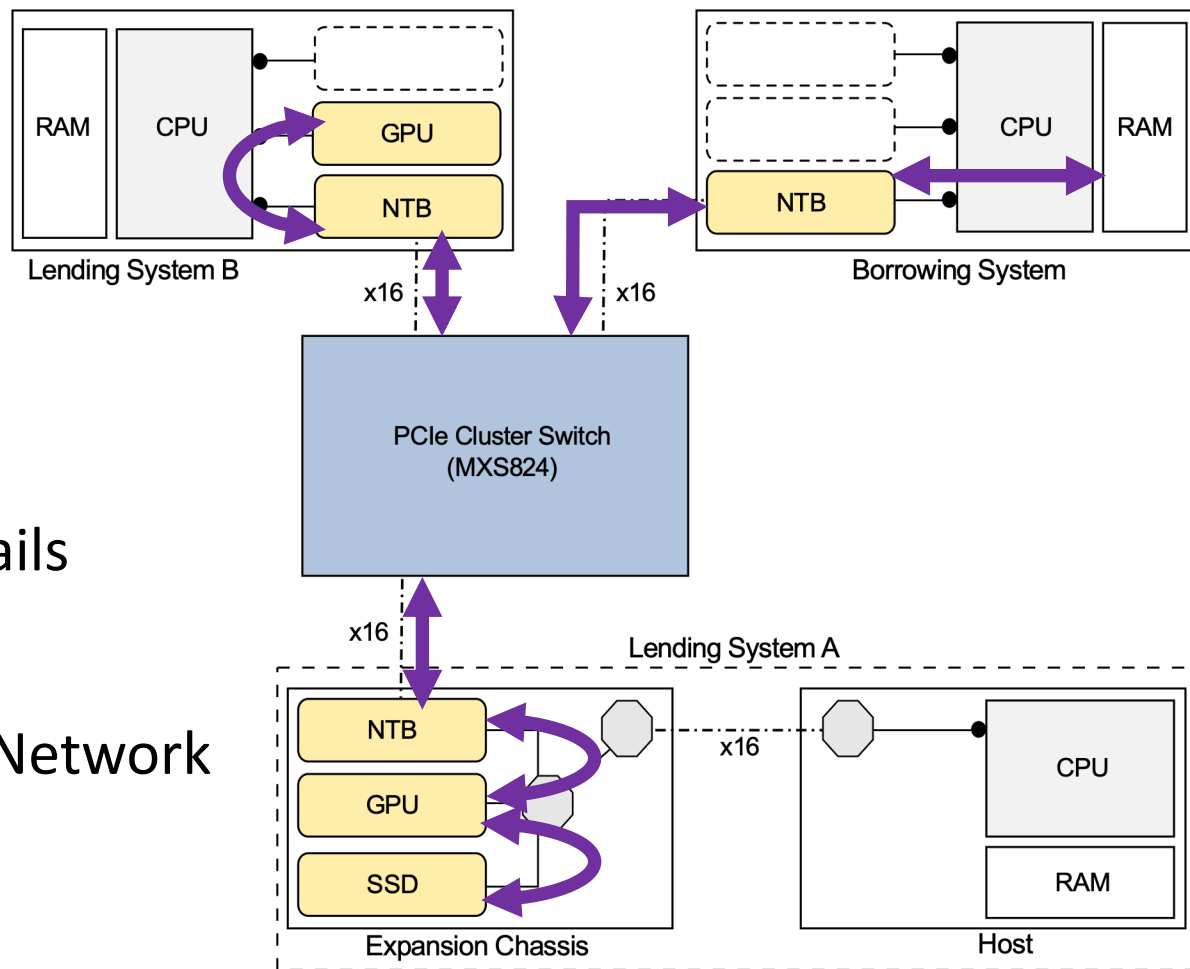
Device Lending details

S9709 Dynamic Sharing of GPUs and IO in a PCIe Network
Thursday March 21, Room 212B

Visit Dolphin at booth
#1520 for a live demo!

CUDA NVME driver + benchmarks

<http://github.com/enfiskutensykkel/ssd-gpu-dma>



Thank you!

jonassm@dolphinics.com

My e-mail address

Device Lending details

S9709 Dynamic Sharing of GPUs and IO in a PCIe Network
Thursday March 21, Room 212B

Visit Dolphin at booth
#1520 for a live demo!

CUDA NVME driver + benchmarks

<http://github.com/enfiskutensykkel/ssd-gpu-dma>

