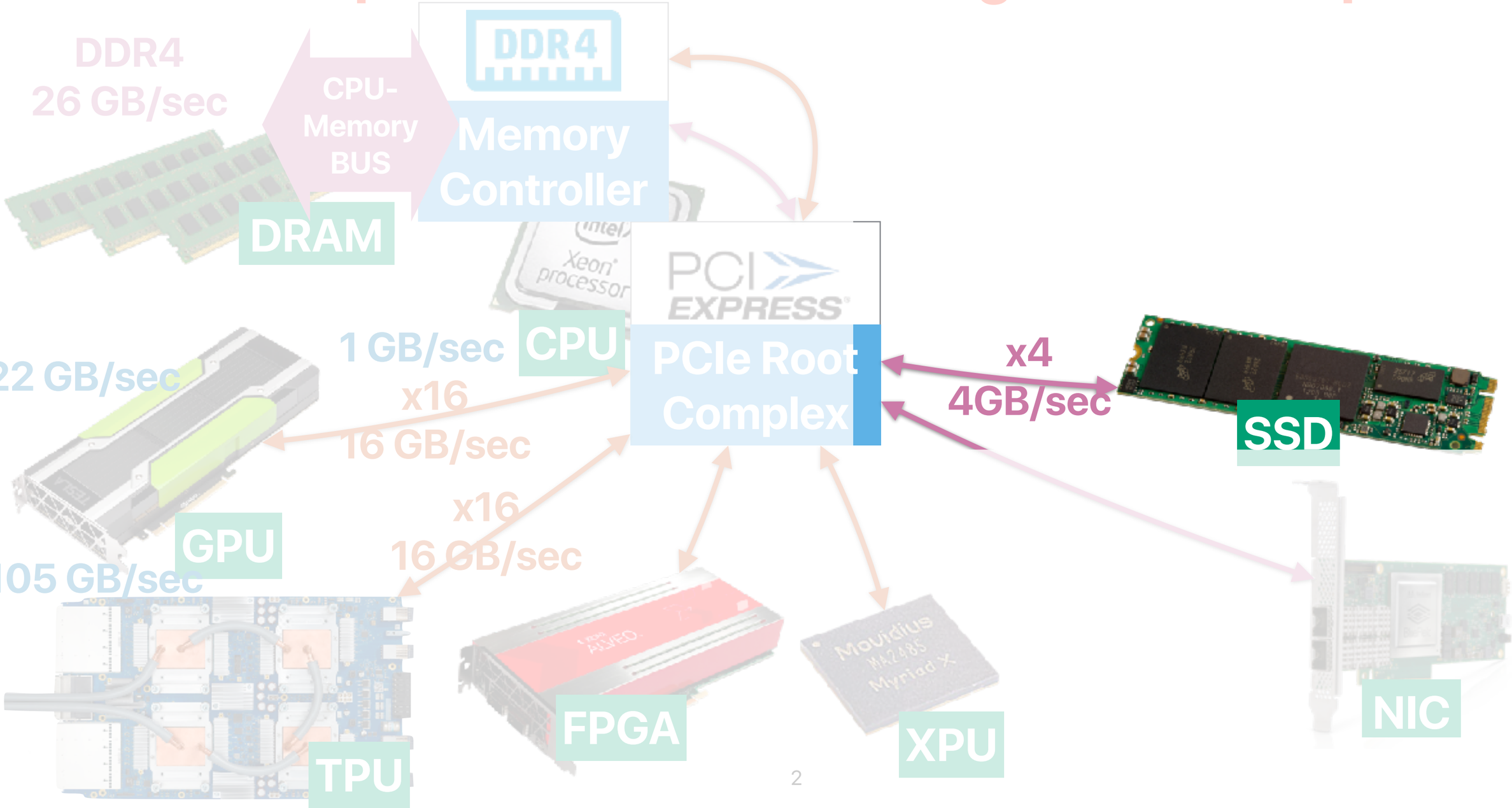
Modern Heterogeneous Computers: (7) Data Storage Systems and Interconnects

Hung-Wei Tseng

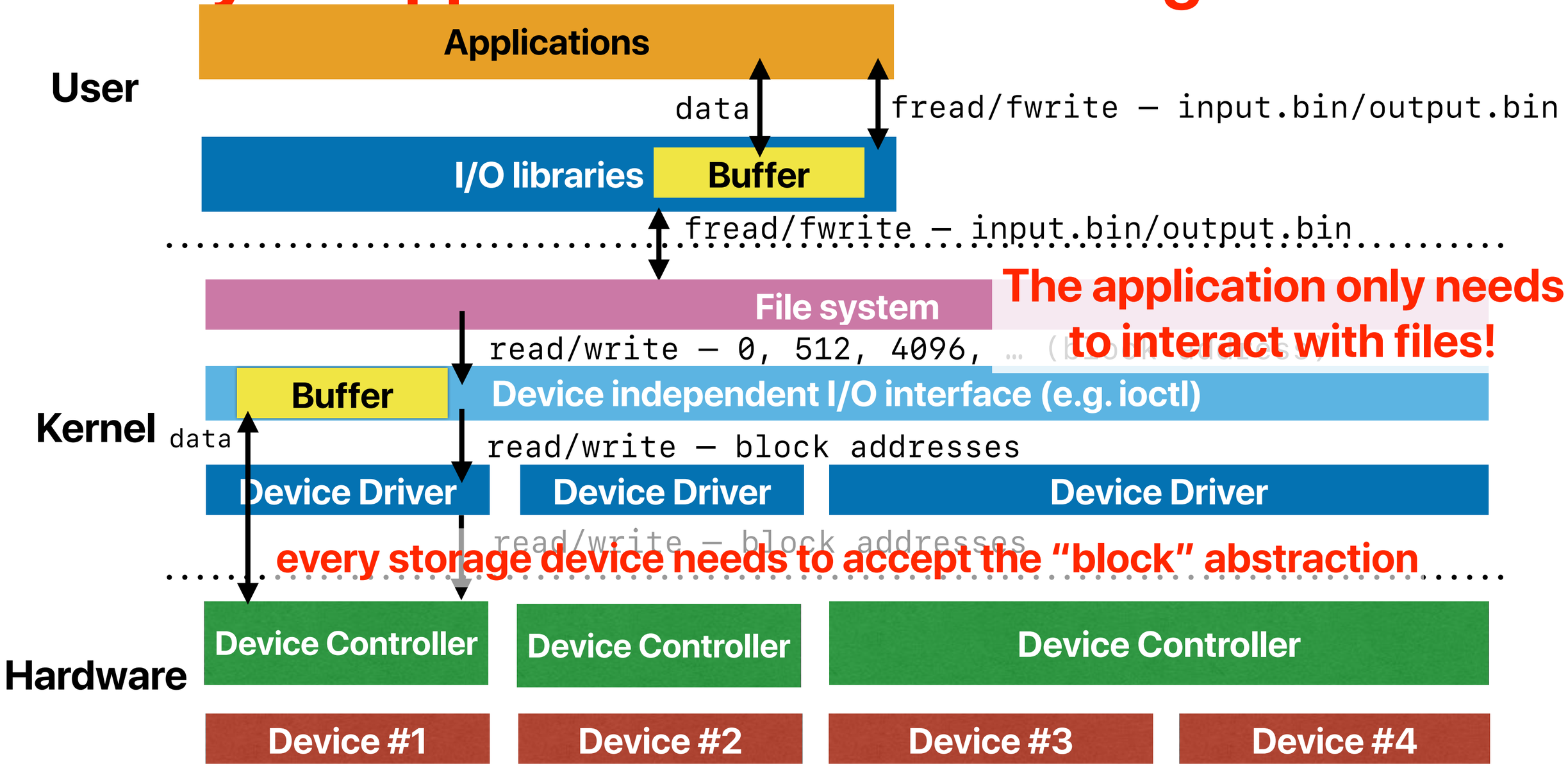
The “data path” in modern heterogeneous computers



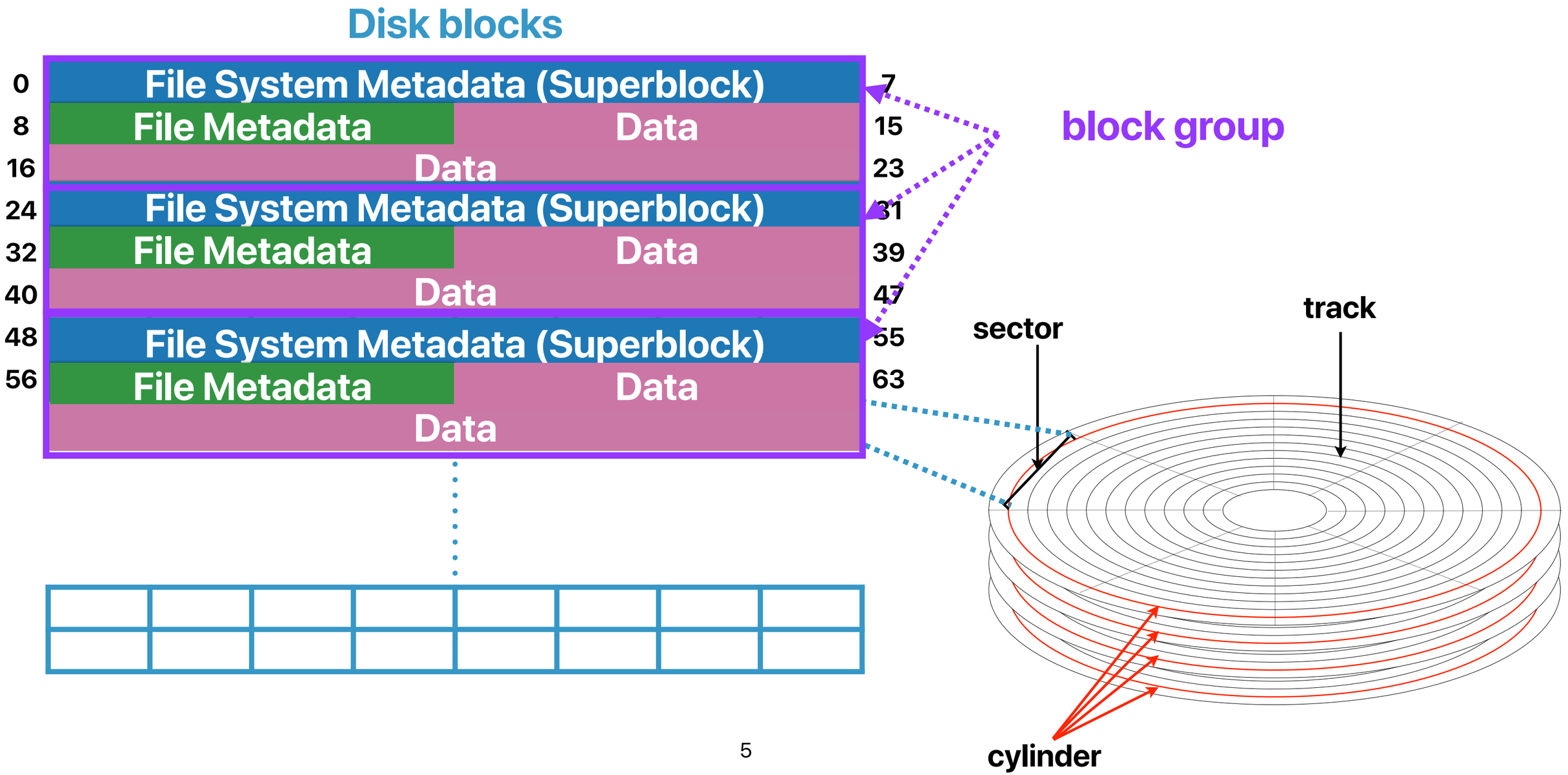
Storage Systems

- Persistent
- Lower $ per byte — using technologies with higher density
- Higher latency
  - Main memory is at 100 ns level
  - Storage is around us or ms level

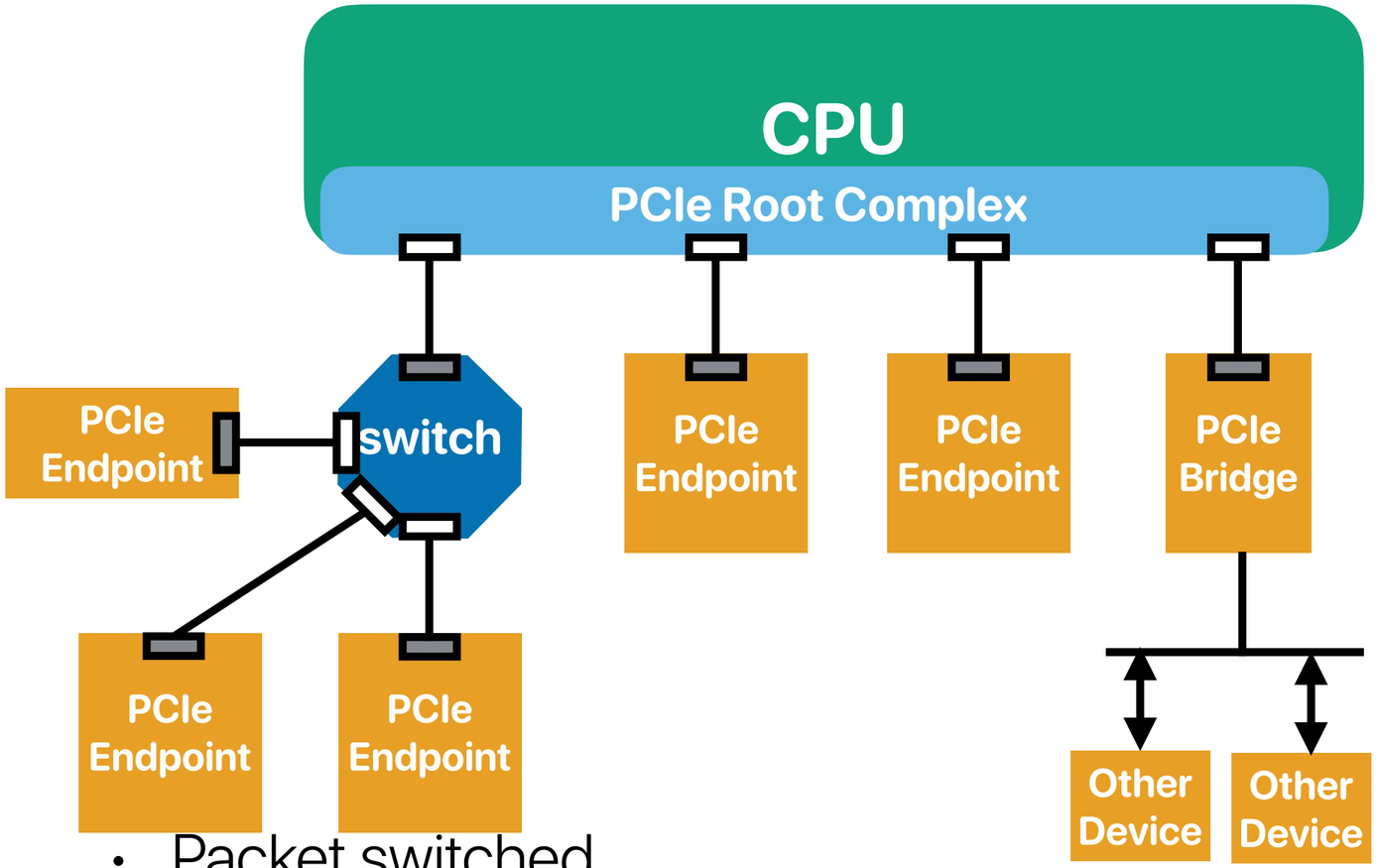
How your application reaches storage devices

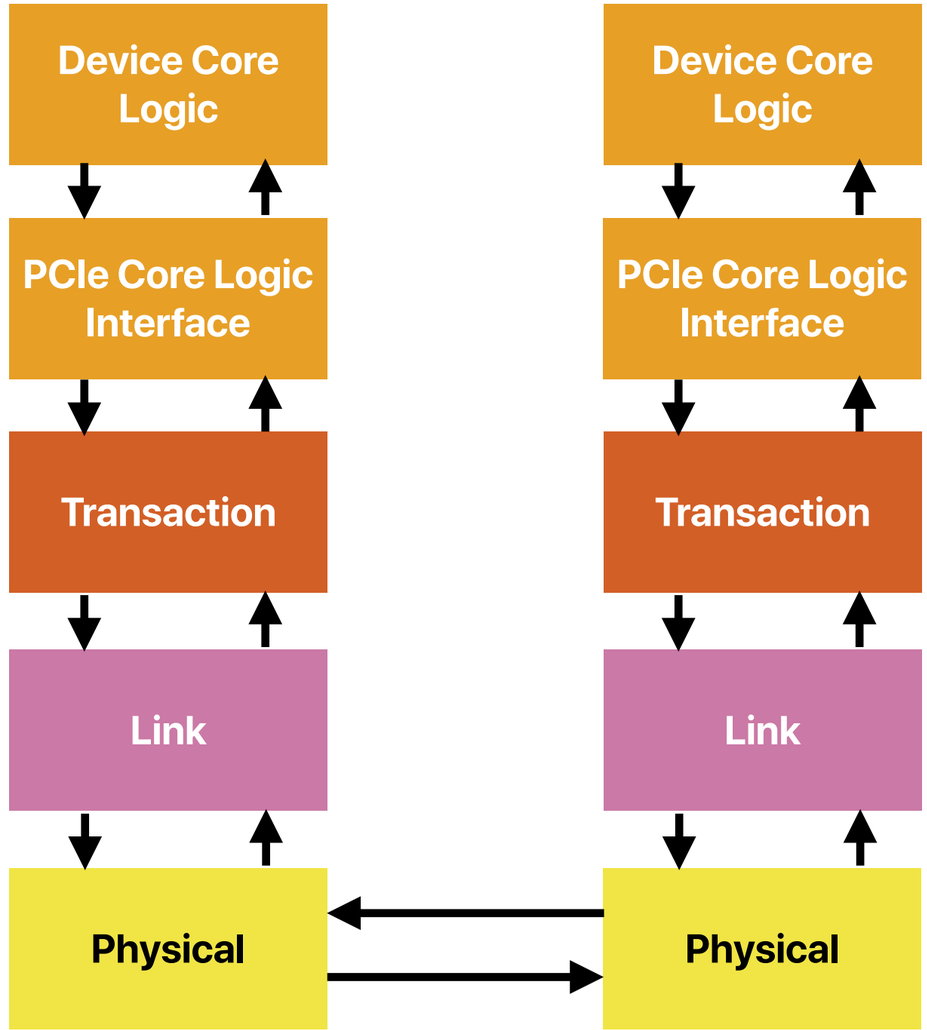


How ExtFS use disk blocks



PCIe "Interconnect"

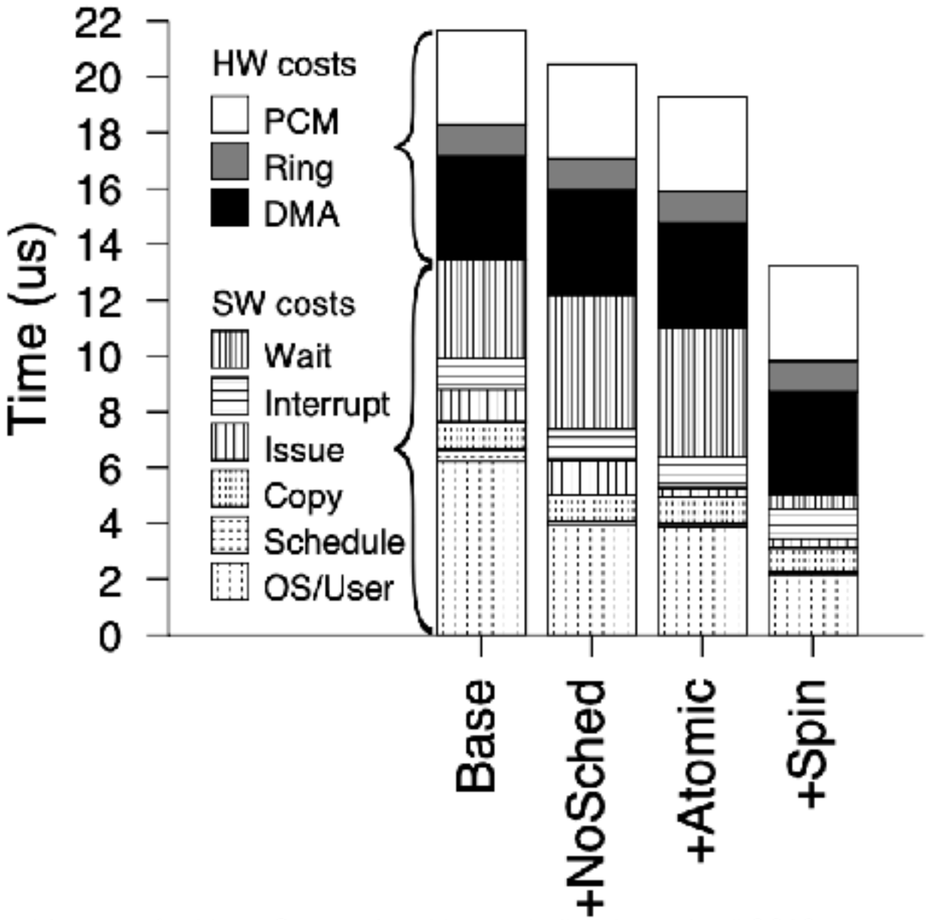




- Packet switched
- Use memory addresses to route packets
- Bandwidth is limited as the PCIe Root Complex has limited bandwidth

Software overhead

Label	Description	Baseline latency (μs) Write	Read
OS/User	OS and userspace overhead	1.98	1.95
OS/User	Linux block queue and no-op scheduler	2.51	3.74
Schedule	Get request from queue and assign tag	0.44	0.51
Copy	Data copy into DMA buffer	0.24/KB	-
Issue	PIO command writes to Moneta	1.18	1.15
DMA	DMA from host to Moneta buffer	0.93/KB	-
Ring	Data from Moneta buffer to mem ctrl	0.28/KB	-
PCM	4 KB PCM memory access	4.39	5.18
Ring	Data from mem ctrl to Moneta buffer	-	0.43/KB
DMA	DMA from Moneta buffer to host	-	0.65/KB
Wait	Thread sleep during hw	11.8	12.3
Interrupt	Driver interrupt handler	1.10	1.08
Copy	Data copy from DMA buffer	-	0.27/KB
OS/User	OS return and userspace overhead	1.98	1.95
Hardware total for 4 KB (accounting for overlap)		8.2	8.0
Software total for 4 KB (accounting for overlap)		13.3	12.2
File system additional overhead		5.8	4.2



A. M. Caulfield, A. De, J. Coburn, T. I. Mollow, R. K. Gupta and S. Swanson, "Moneta: A High-Performance Storage Array Architecture for Next-Generation, Non-volatile Memories," 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture, 2010, pp. 385-395, doi: 10.1109/MICRO.2010.33.

3:00

What's the "overall" process if we want compute datasets stored in a storage device using GPU?

The CUDA code

Moving data

```
   // allocate m
1: m = (float*) malloc(sizeof(float)*matrix_dim*matrix_dim);
   // open file
2: fp = open(filename, O_RDONLY|O_DIRECT);
   // read file
3: read(fp, m, sizeof(float)*matrix_dim*matrix_dim);

4: close(fp);
   // allocate GPU memory
5: cudaMalloc((void**)&d_m, matrix_dim*matrix_dim*sizeof(float));
   //copy from main memory to GPU memory
6: cudaMemcpy(d_m, m, matrix_dim*matrix_dim*sizeof(float),
              cudaMemcpyHostToDevice);
7: lud_diagonal<<<1,BLOCK_SIZE>>>(d_m, matrix_dim);
   //copy from GPU memory to main memory
8: cudaMemcpy(m, d_m, matrix_dim*matrix_dim*sizeof(float),
              cudaMemcpyDeviceToHost);
9: output_matrix(m, matrix_dim);
```

storing input in m

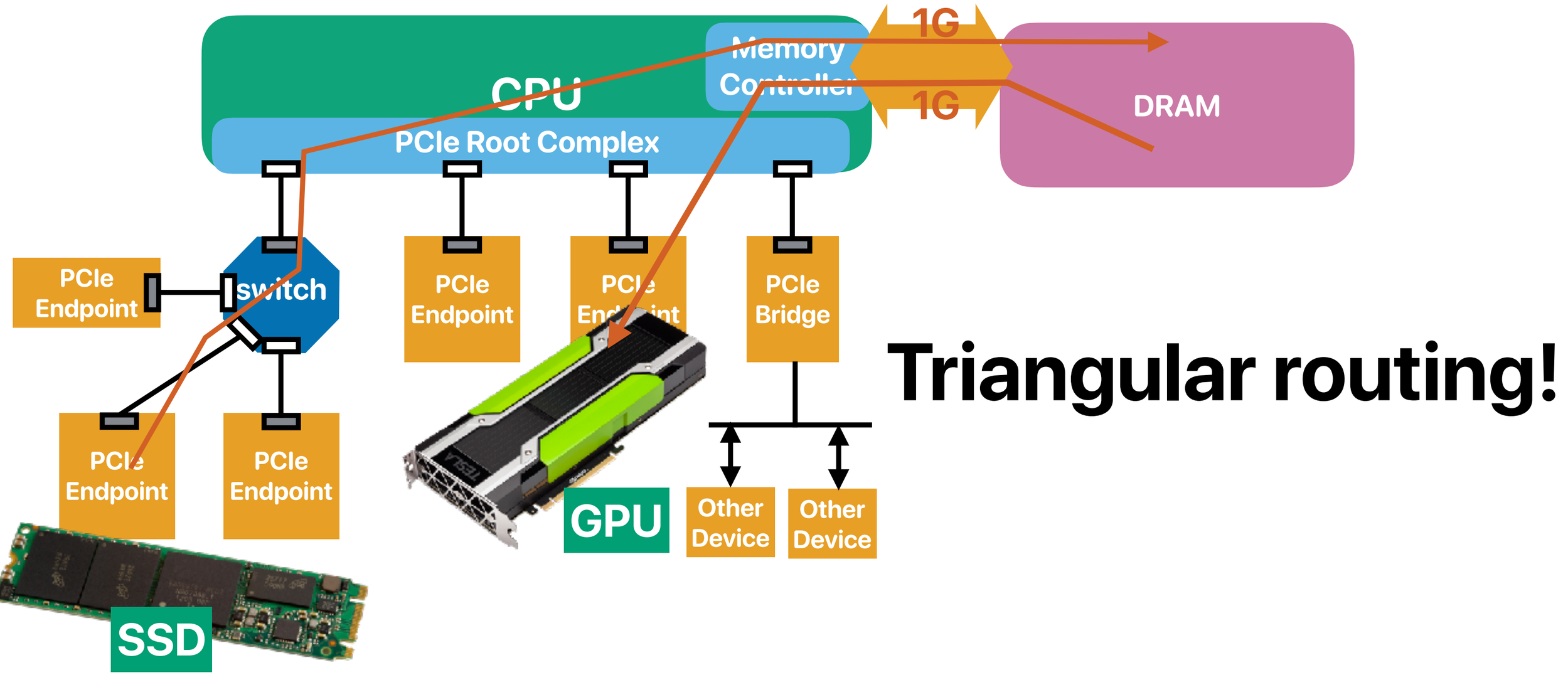
copying data in m to GPU memory

GPU computing

Moving data

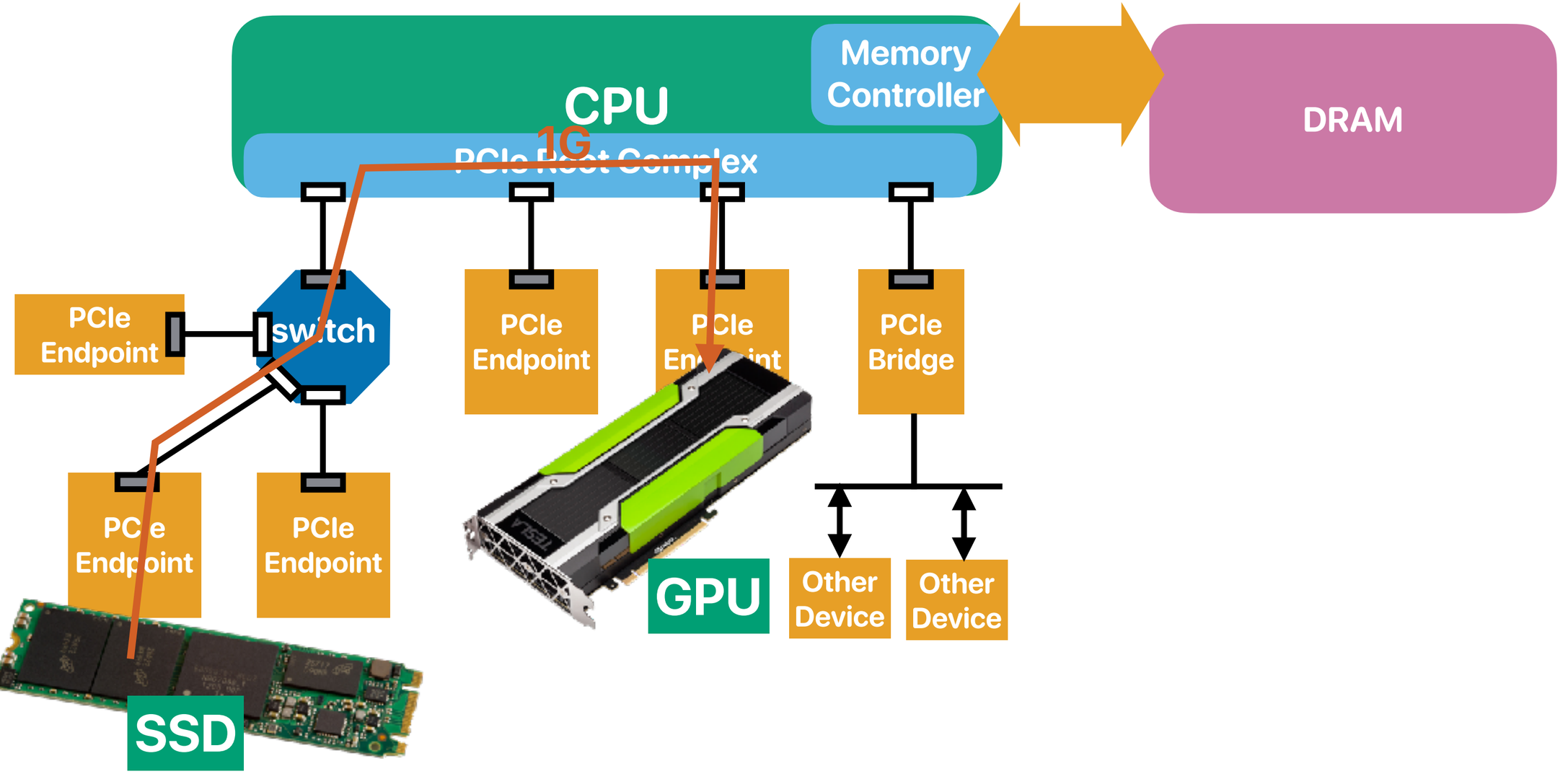
m is overwritten, previous m is only used as a “buffer”

PCIe "Interconnect"

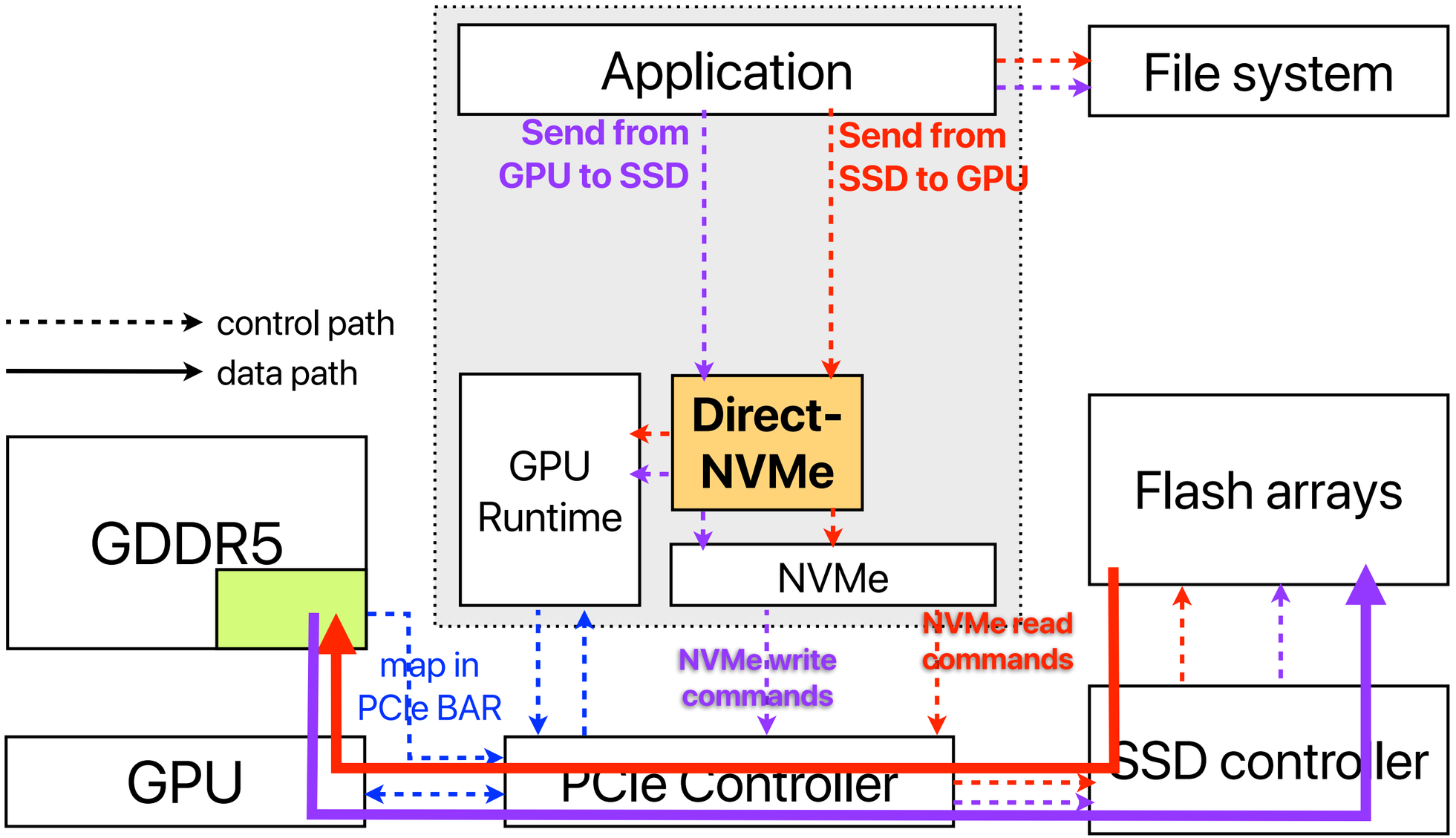


Triangular routing!

PCIe supports peer-to-peer communication!



NVMeDirect


Application
File system
Send from
GPU to SSD
Send from
SSD to GPU
control path
data path
GPU
Runtime
Direct-
NVMe
NVMe
GDDR5
Flash arrays
map in
PCIe BAR
NVMe write
commands
NVMe read
commands
GPU
PCIe Controller
SSD controller

```
cudaSetDevice(0)
ptr0 = cudaMalloc();
cuPointerGetAttribute(&return_data, CU_POINTER_ATTRIBUTE_P2P_TOKENS, ptr0);
// Returns [p2pToken = 0xabcd, vaSpaceToken = 0x1]
cudaFree(ptr0);
cudaSetDevice(1);
ptr1 = cudaMalloc();
assert(ptr0 == ptr1);
// The CUDA driver is free (although not guaranteed) to reuse the VA,
// even on a different GPU
cuPointerGetAttribute(&return_data, CU_POINTER_ATTRIBUTE_P2P_TOKENS, ptr0);
// Returns [p2pToken = 0x0123, vaSpaceToken = 0x2]
```

```
void pin_buffer(void *address, size_t size)
{
    unsigned int flag = 1;
    CUresult status = cuPointerSetAttribute(&flag,
CU_POINTER_ATTRIBUTE_SYNC_MEMOPS, address);
    if (CUDA_SUCCESS == status) {
        // GPU path
        pass_to_kernel_driver(address, size);
    } else {
        // CPU path
        // ...
    }
}
```

```
// for boundary alignment requirement
#define GPU_BOUND_SHIFT   16
#define GPU_BOUND_SIZE    ((u64)1 << GPU_BOUND_SHIFT)
#define GPU_BOUND_OFFSET  (GPU_BOUND_SIZE-1)
#define GPU_BOUND_MASK    (~GPU_BOUND_OFFSET)

struct kmd_state {
     nvidia_p2p_page_table_t *page_table;
    // ...
};

void kmd_pin_memory(struct kmd_state *my_state, void *address, size_t
size)
{
    // do proper alignment, as required by NVIDIA kernel driver
    u64 virt_start = address & GPU_BOUND_MASK;
    size_t pin_size = address + size - virt_start;
    if (!size)
     return -EINVAL;
    int ret = nvidia_p2p_get_pages(0, 0, virt_start, pin_size,
&my_state->page_table, free_callback, &my_state);
    if (ret == 0) {
        // Succesfully pinned, page_table can be accessed
    } else {
        // Pinning failed
    }
}
```

https://docs.nvidia.com/cuda/gpudirect-rdma/index.html

Demo

https://github.com/hungweitseng/EE277/tree/main/demo/NVMeDirect/

3:00

What idea can you build upon peer-to-peer PCIe communication?

References

- Direct data-path between SSD/GPU
  - Hung-Wei Tseng, Yang Liu, Mark Gahagan, Jing Li, Yanqin Jing, and Steven J. Swanson. Gullfoss: Accelerating and simplifying data movement among heterogeneous computing and storage resources. Department of Computer Science and Engineering, University of California, San Diego, 2015.
  - Shai Bergman, Tanya Brokhman, Tzachi Cohen, and Mark Silberstein. 2017. SPIN: seamless operating system integration of peer-to-peer DMA between SSDs and GPUs. In Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '17). USENIX Association, USA, 167–179
  - NVIDIA's GDS library — https://github.com/NVIDIA/gds-nvidia-fs
- Virtual memory for GPU
  - P. Markthub, M. E. Belviranli, S. Lee, J. S. Vetter and S. Matsuoka, "DRAGON: Breaking GPU Memory Capacity Limits with Direct NVM Access," SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, 2018, pp. 414-426, doi: 10.1109/SC.2018.00035.
- Database
  - Jing Li, Hung-Wei Tseng, Chunbin Lin, Yannis Papakonstantinou, and Steven Swanson. "Hippogriffdb: Balancing i/o and gpu bandwidth in big data analytics." Proceedings of the VLDB Endowment 9, no. 14 (2016): 1647-1658.
  - Min-Gyo Jung, Chang-Gyu Lee, Donggyu Park, Sungyong Park, Jungki Noh, Woosuk Chung, Kyoung Park, and Youngjae Kim. 2021. GPUKV: an integrated framework with KVSSD and GPU through P2P communication support. In Proceedings of the 36th Annual ACM Symposium on Applied Computing (SAC '21). Association for Computing Machinery, New York, NY, USA, 1156–1164. DOI:https://doi.org/10.1145/3412841.3441990
- GPU-FS
  -

3:00

Does having a P2P path address the performance issue?

References

- Page cache
  - Brokhman, Tanya, Pavel Lifshits, and Mark Silberstein. "{GAIA}: An {OS} page cache for heterogeneous systems." In 2019 USENIX Annual Technical Conference (USENIX ATC 19), pp. 661-674. 2019.
- Dynamic buffer on DRAM
  - Y. Liu, H. -W. Tseng, M. Gahagan, J. Li, Y. Jin and S. Swanson, "Hippogriff: Efficiently moving data in heterogeneous computing systems," 2016 IEEE 34th International Conference on Computer Design (ICCD), 2016, pp. 376-379, doi: 10.1109/ICCD.2016.7753307.

Project/paper presentation

- Starting from 5/24/2022 — check the schedule on the website
- Prepare an 18-minute talk (7 minutes for Q/A)
  - Why — the motivation (7 minutes)
    - What's the main problem you're addressing?
    - Why should we care about this?
    - Why is it the right time to do this?
  - What — the idea (6 minutes)
    - What's the proposed idea?
    - What's new about this idea?
  - How — the method/approach (5 minutes)
    - How can we implement the idea?
    - How does the idea perform?

Electrical
Computer Science
Engineering

277

つづく