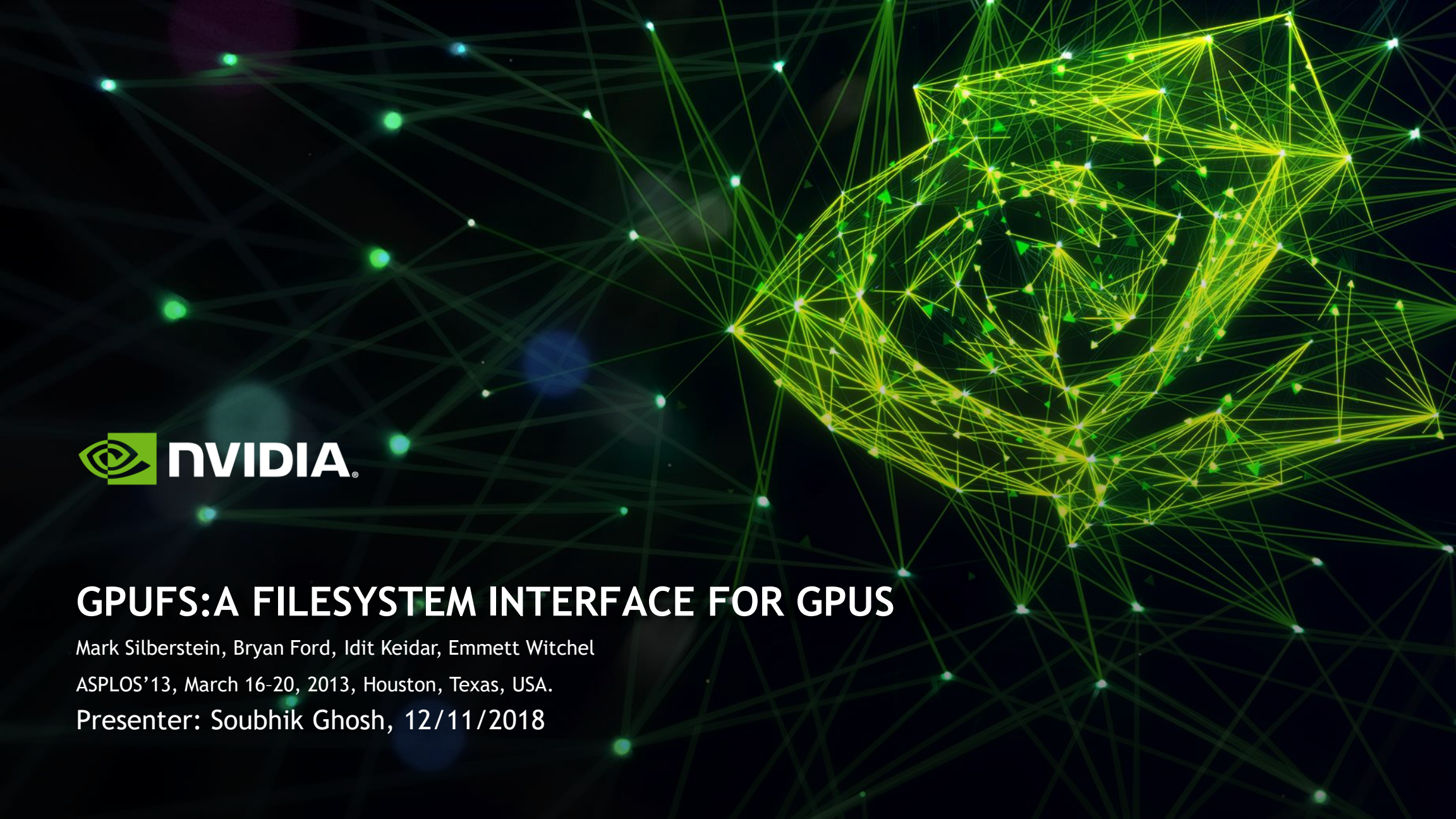


GPUFS:A FILESYSTEM INTERFACE FOR GPUS

Mark Silberstein, Bryan Ford, Idit Keidar, Emmett Witchel

ASPLOS'13, March 16–20, 2013, Houston, Texas, USA.

Presenter: Soubhik Ghosh, 12/11/2018

AGENDA

- Background of GPU Computing
- Common Issues: Lack of *OS abstractions*
- Solution - GPUfs
- Implementation of GPUfs

NVIDIA.

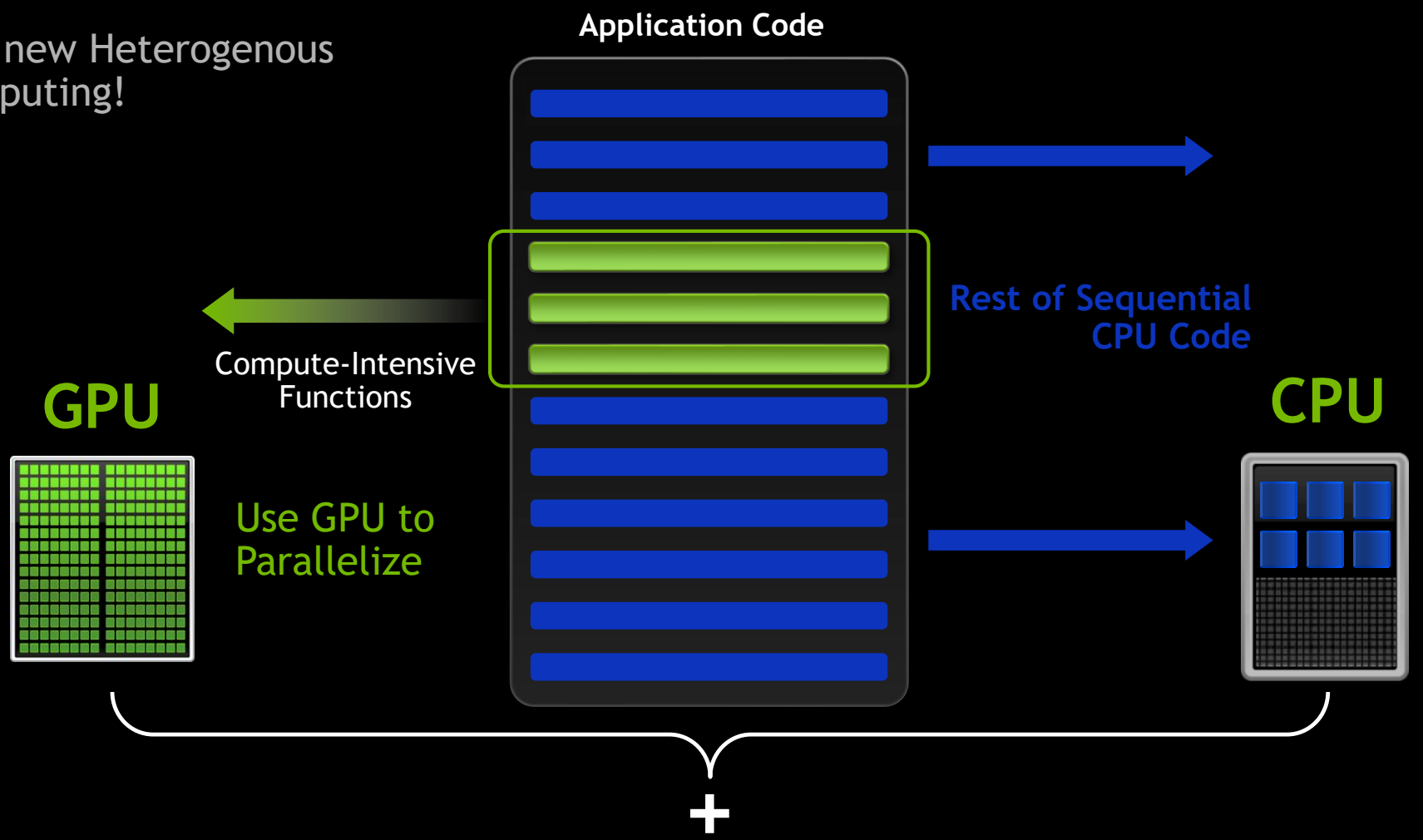
GPU INTRODUCTION

- GPU: Graphics processing unit
  - Thousands of Cores
  - Very high memory bandwidth
  - Evolved
    - From real-time, high-definition 3D graphics
    - To compute intensive general purpose processor
  - Available on most systems/can be easily installed
  - Following Moore's Law better than CPU!

NVIDIA.

GPU COMPUTING

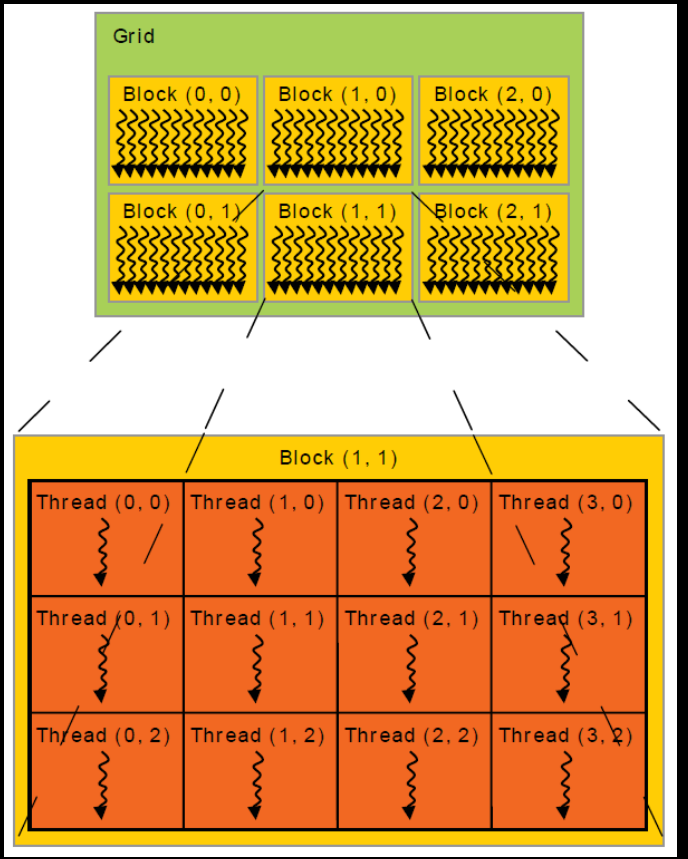
- The new Heterogenous computing!



NVIDIA

ARCHITECTURE

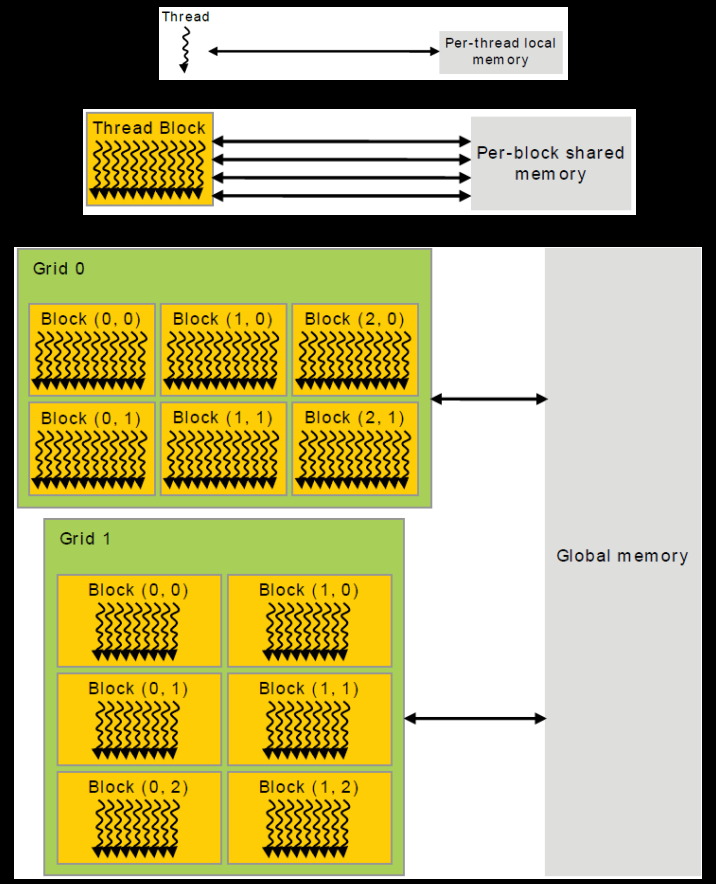
- Software layer
  - Each thread runs sequential code
  - Threads are organized into a block (Note: Each block is broken into warps of 32 threads)
  - Blocks are organized into a grid
- #(Blocks) >= #(Multiprocessor)
  - No preemption
- #(Threads) >= #(core)
  - Interleaved



NVIDIA.

ARCHITECTURE

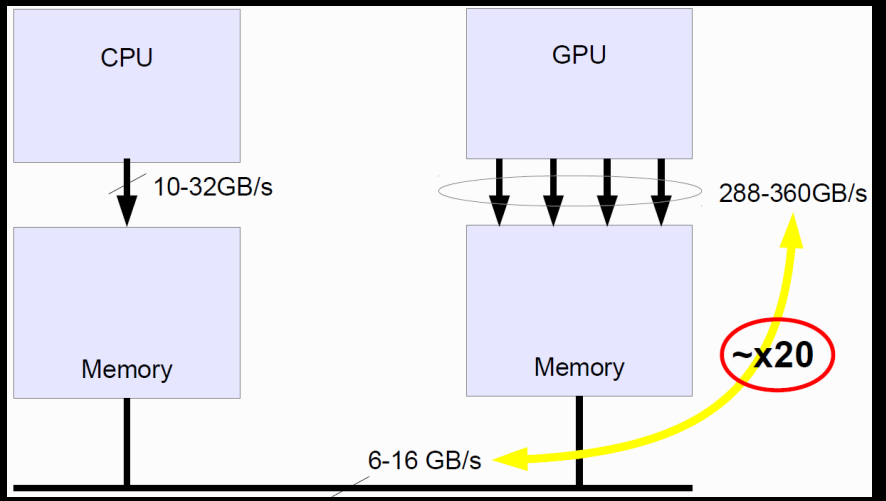
- Programming Model
  - CPU initializes the GPU
  - Tasks are sent to the GPU in the form of kernel functions.
  - Each kernel function is logically mapped to a grid instance.
  - CPU manages GPU memory allocation/deallocation
  - CPU decides block/grid size



NVIDIA

GPU COMPUTING CHALLENGES

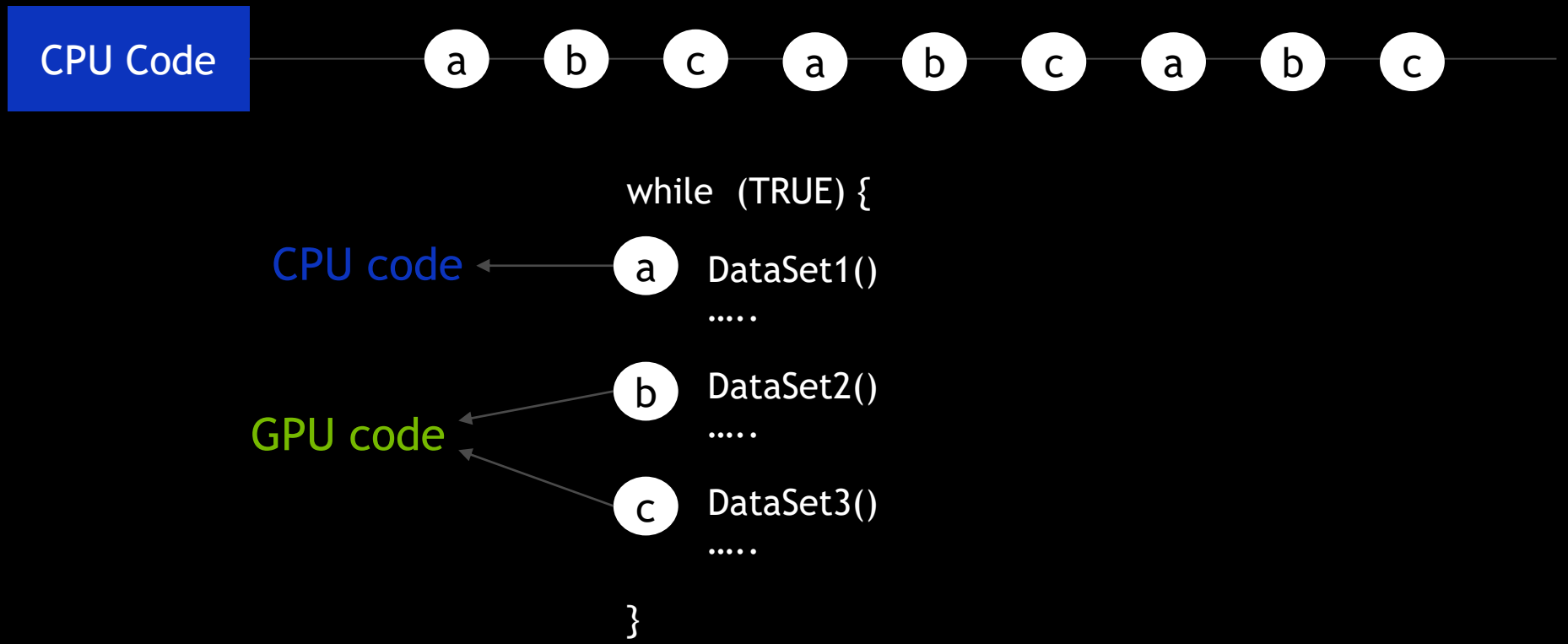
- Lacks core system abstractions
  - *Files, sockets*
- Incapable of initiating basic system interactions for itself
- Complex GPU-CPU data management
- No Memory coherence
  - Complex GPU-CPU interconnect topologies
    - PCIe, NVLink
  - Different CPU-GPU interconnect latencies/bandwidth



NVIDIA

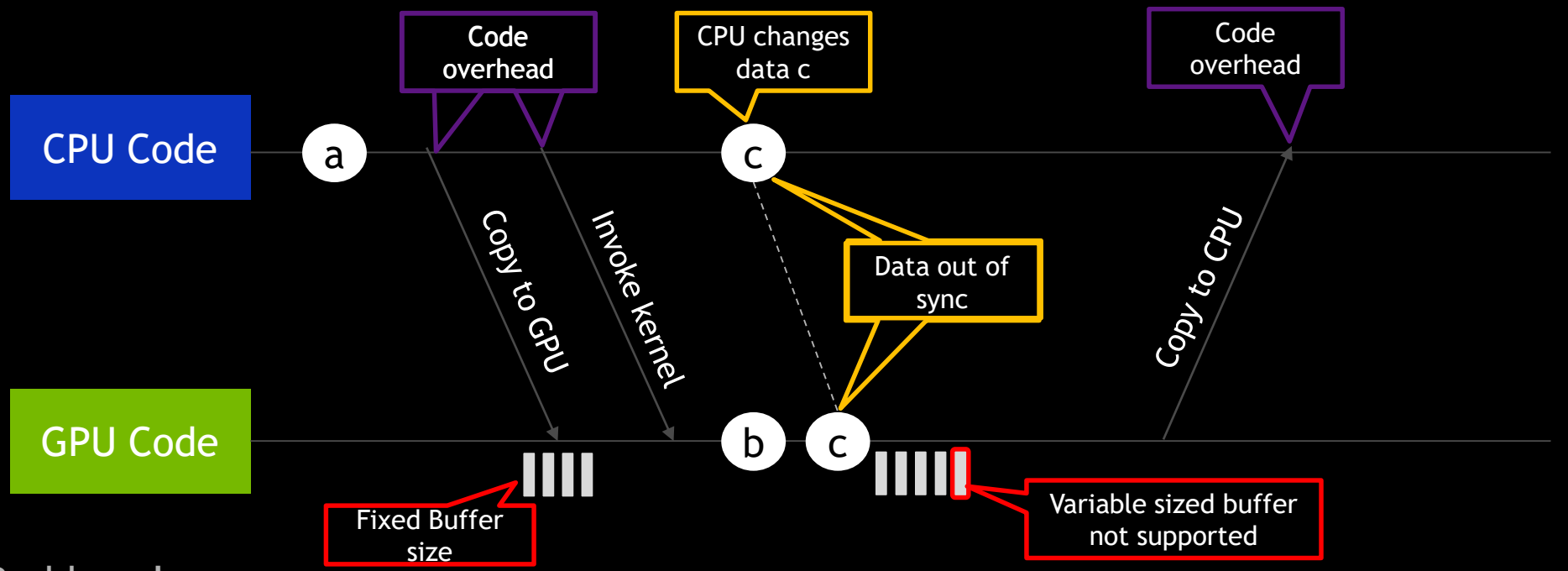
EXAMPLE OF CHALLENGES

Consider this application…



NVIDIA.

EXAMPLE OF CHALLENGES



- Problems!
  1. Complex low-level data management code
  2. Bounded buffer problem
  3. Data inconsistency problem (Double Buffering)

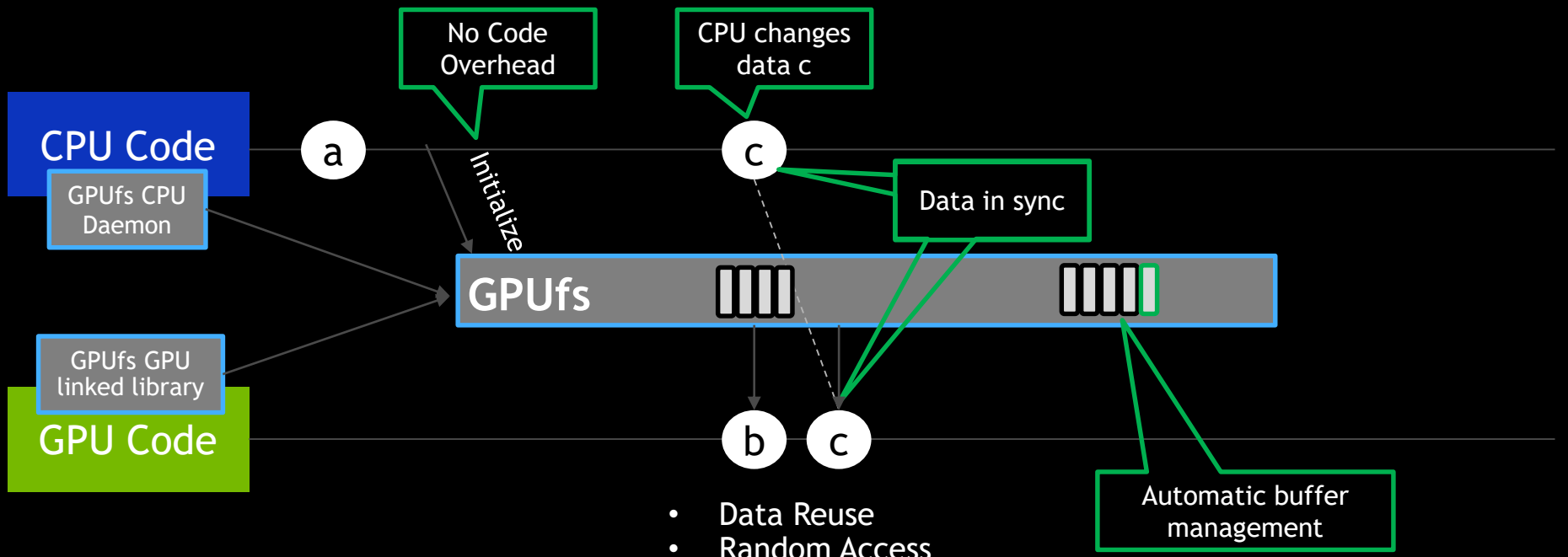
NVIDIA

MOTIVATION OF GPUFS

- GPUFS: GPU filesystem
  - File system abstraction for GPU applications
  - Hide GPU-CPU data management
    - cudaMalloc, cudaMemcpy, cudaFree
    - Oblivious to data catching - Device memory, system memory, Disk
    - Hide access locality optimizations and memory coherency
  - POSIX style API
    - *open, read, write, close, mmap, munmap*
  - GPU-CPU portable

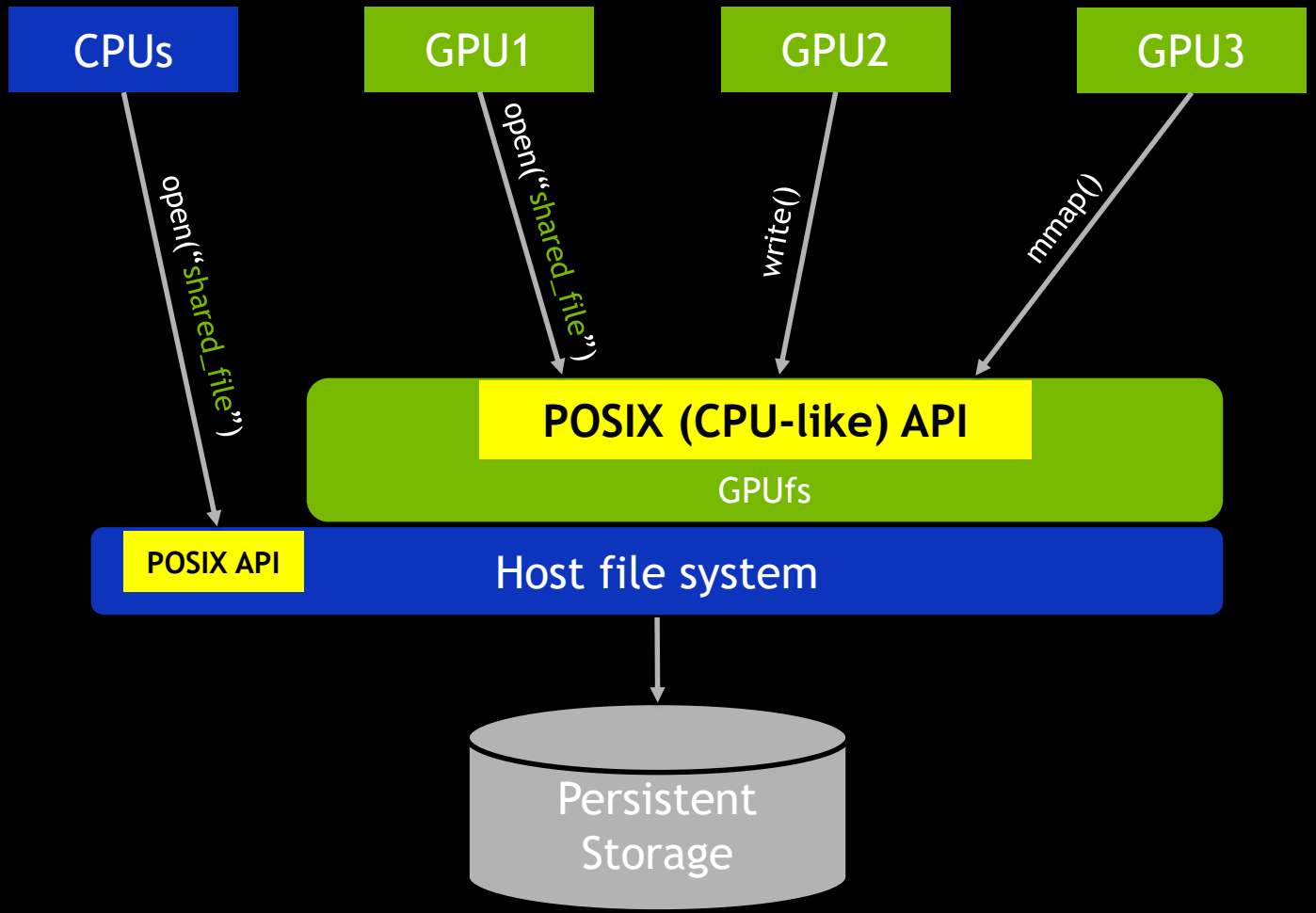
NVIDIA.

EXAMPLE OF SOLUTION



- Data Reuse
- Random Access

GPUFS: APPLICATION VIEW



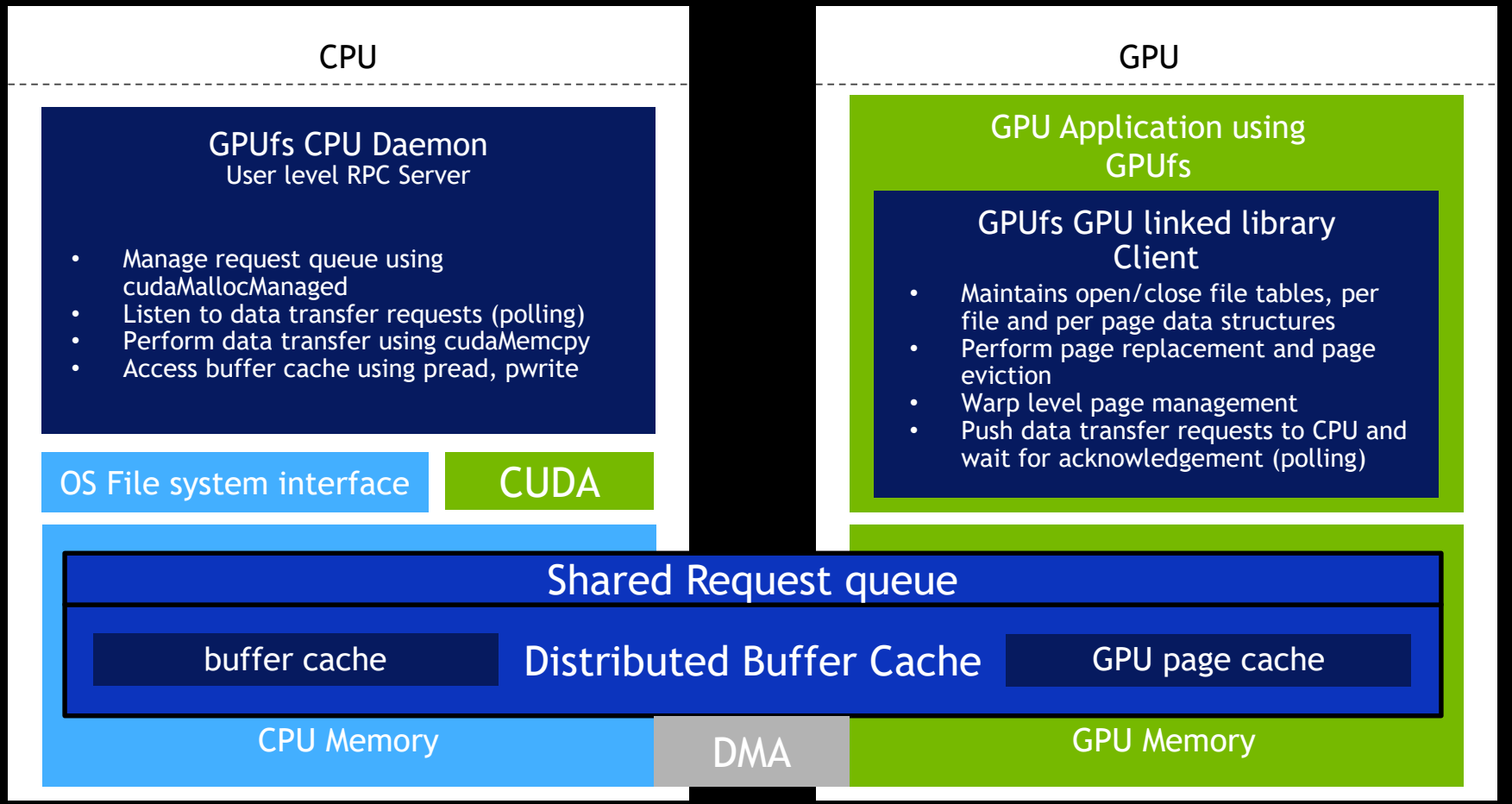
NVIDIA.

GPUFS DESIGN

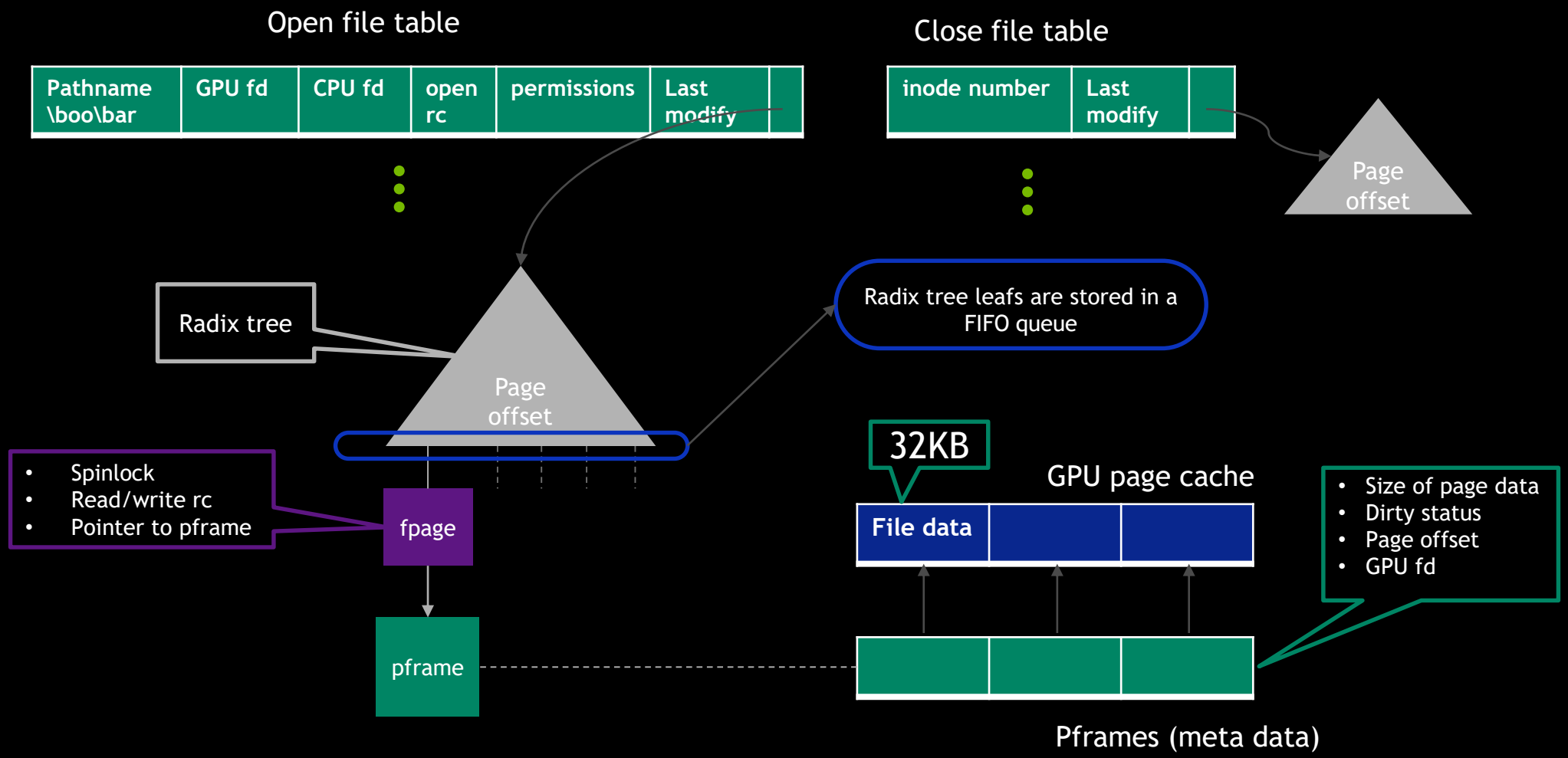
- Every GPU code links GPUfs library
  - GPUfs library and the host OS work
    - To coordinate the file-system namespace and data
    - Cache Filesystem data in CPU/GPU memory
- Recall that GPU has multiple granularities
  - Threads are grouped into warps of 32 threads
  - Warps are executed in lockstep
  - GPUfs API works at warp-level granularity
- Weak consistency model between GPU and CPU
- Strong consistency model inside a GPU

NVIDIA.

GPUFS DESIGN



GPUFS IMPLEMENTATION



GPU BUFFER CACHE

- Per-file buffer cache
  - Files consist of 32KB pages with internal fragmentation
  - Page is looked by through a radix tree of page offsets.
  - Radix tree lookup is done using unlocked reads and locked updates using seqlock
    - Reads are just traversing the tree
    - Updates consist of adding or removing pages from the tree
  - Page and its pframe are accessed for reading/writing/evicting actions using spinlock
- Buffer cache management (No free page)
  - Performed when reading/writing to a page not in memory
  - Culprit page(s) are chosen in the order of -> closed files -> read-only files -> writable files
    - Flushed accordingly when dirty in writable case

NVIDIA.

GPUFS API

- Open and file descriptors – gopen()
  - GPUfs maintains one global file descriptor(fd)
    - All threads share one fd per file (No seek pointer)
    - First gopen() calls POSIX open() and creates GPU fd
    - Subsequent gopen() increment open reference count
  - Reads ahead some file data and allocates radix tree for lookup
  - Checks the close file table first
    - If timestamp is old or no entry is present then request CPU to transfer data
    - Else copy the file buffer cache to the open file table
  - Adds O_NOSYNC

NVIDIA.

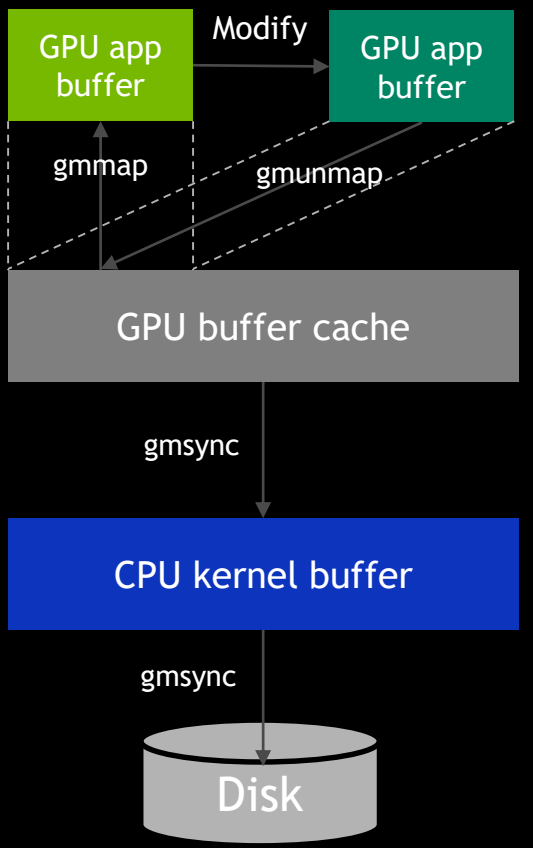
GPUFS API

- Read and write – gread()/gwrite()
  - Uses POSIX pread/pwrite semantics
  - Performs random access using offset
  - Lookup radix tree for relevant GPU page
  - If page not present or is dirty then request CPU to transfer the page
    - Acquire spinlock when reading/writing a particular page
- Close and synchronization – gclose()/gfsync()
  - Decrements open file reference count (orc)
  - gclose() moves the file buffer from open to the close file table when orc == 0
  - Both flush the file pages to the CPU buffer through a data transfer request

NVIDIA.

GPUFS API

- File mapping – gmmap()/gmunmap()/gmsync()
  - Map file directly to GPU memory
    - No separate buffer maintenance unlike gread()
  - GPUs lack virtual memory management
  - GPUfs maps a prefix of the file
    - *gmmap* returns the size of the prefixed matched
    - Direct read/write access to mapped memory
    - gmunmap flushes back to GPU file buffer



NVIDIA.

GPU-CPU REMOTE PROCEDURE CALL

- A single thread on CPU polls for requests for every GPU in a round robin fashion
- Coordinate data transfers between the CPU and GPU
- GPU issues requests to CPU in a shared request queue
  - Request queue is stored in CPU-GPU shared memory
  - Uses NVIDIA's unified memory model
- CPU initiates DMA memory transfers to/from GPU buffer cache pages
  - Uses cudaMemcpy
- CPU acknowledges completion of transfer by writing a done status in the request queue which is polled by GPU library

NVIDIA.

EVALUATION

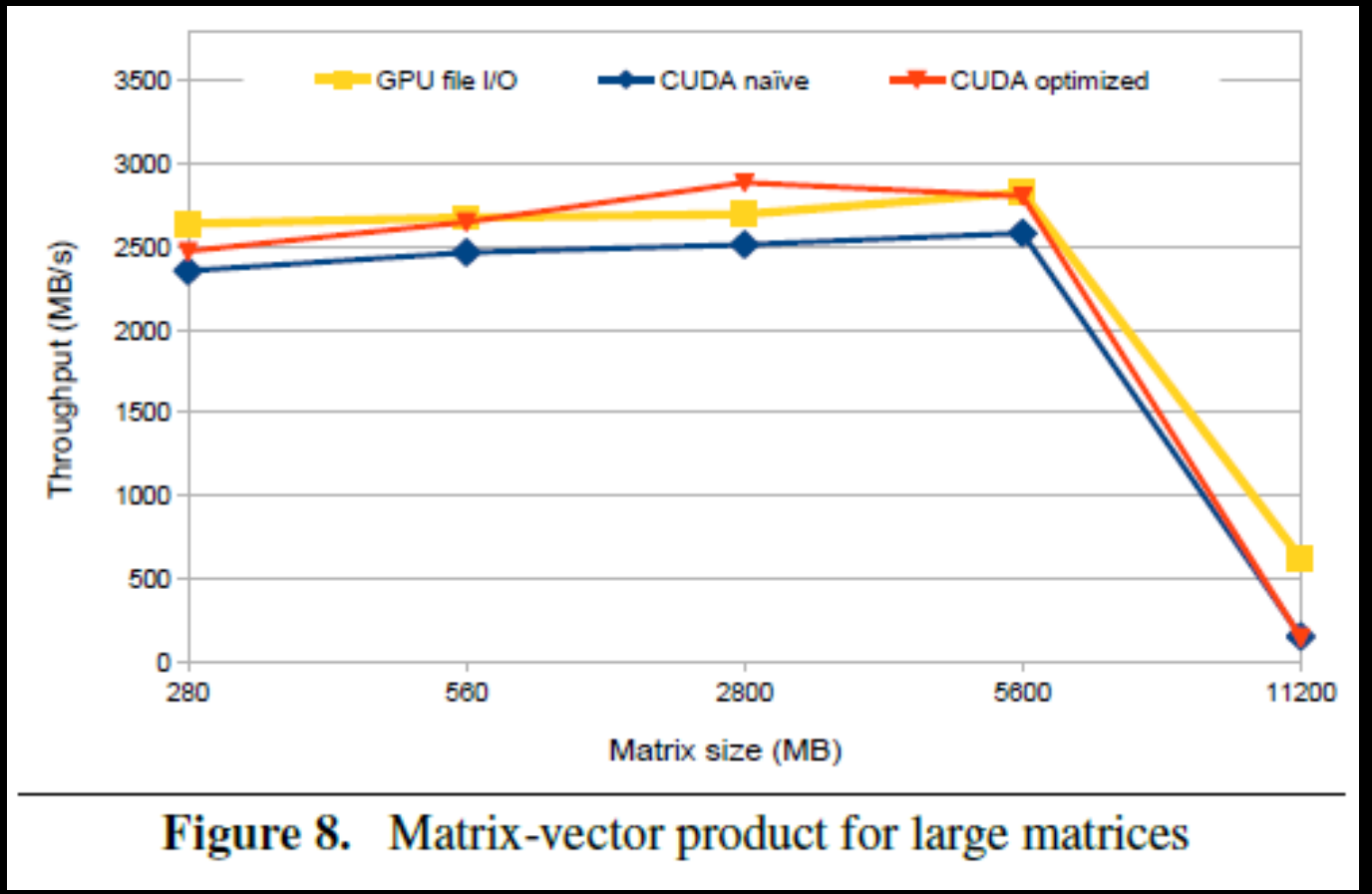


Figure 8. Matrix-vector product for large matrices

RELATED WORK

- Weak-consistency model (close-to-open consistency) similar to AFS (Andrew File System)
- Reading/writing of file data similar to GPFS (General Parallel File System)
  - GPUfs read/writes pages parallelly on a warp level granularity
  - GPFS read/writes blocks parallelly on multiple disks

NVIDIA.

REFERENCES

https://www.cs.utexas.edu/users/witchel/pubs/silberstein13asplos-gpufs.pdf

http://bford.info/pub/os/gpufs-cacm.pdf

https://github.com/gpufs/gpufs