Garena

FREE FIRE

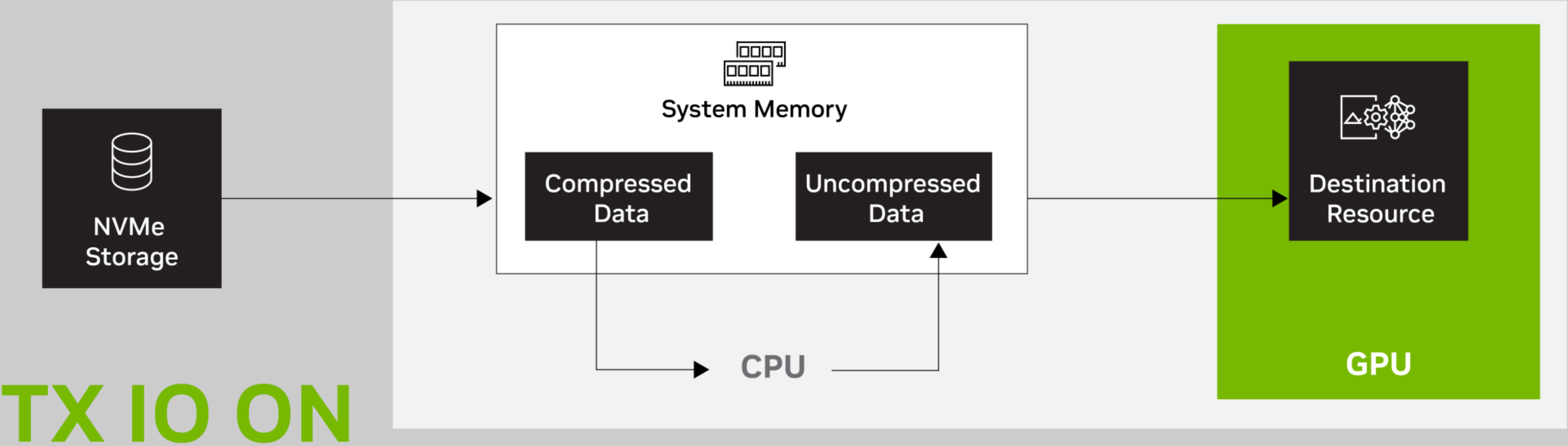
Direct Storage简介

2024.01

01

什么是Direct Storage

RTX IO OFF



NVMe Storage

System Memory

Compressed Data

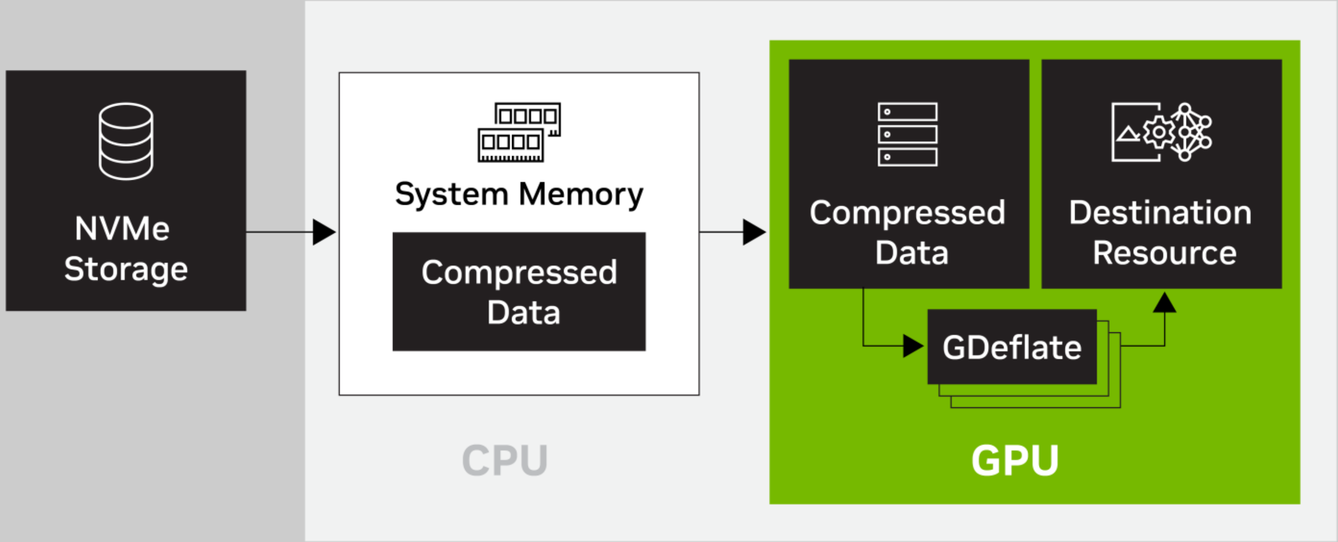
Uncompressed Data

CPU

Destination Resource

GPU

RTX IO ON



NVMe Storage

System Memory

Compressed Data

CPU

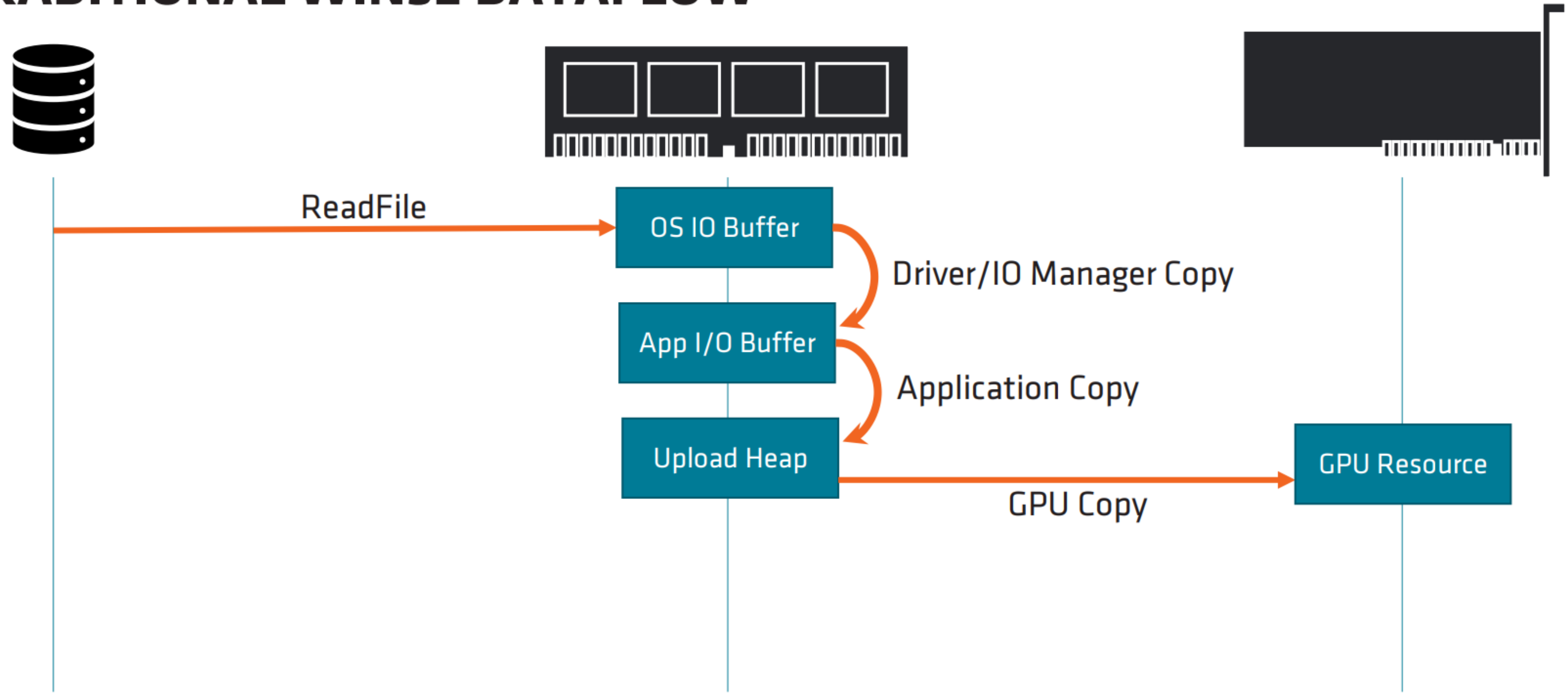
Compressed Data

Destination Resource

GDeflate

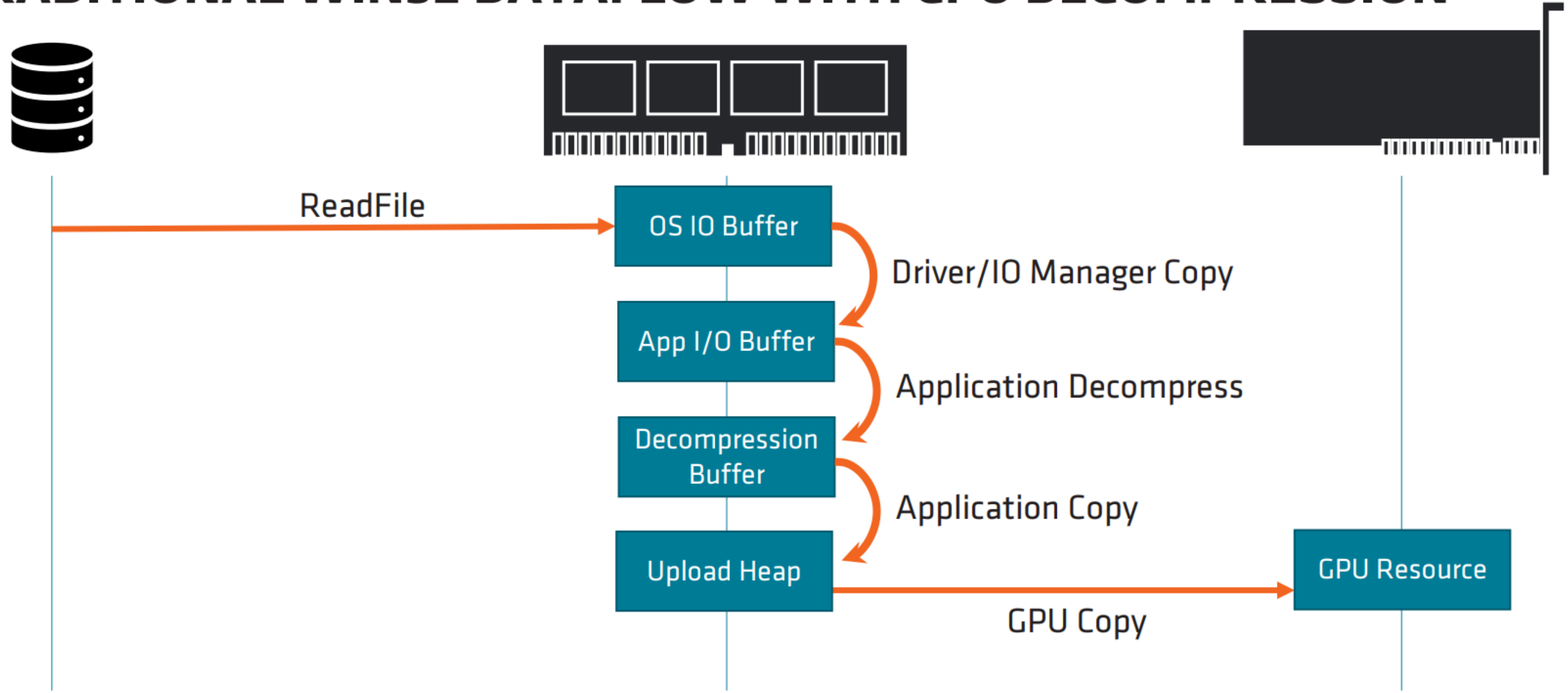
GPU

TRADITIONAL WIN32 DATAFLOW



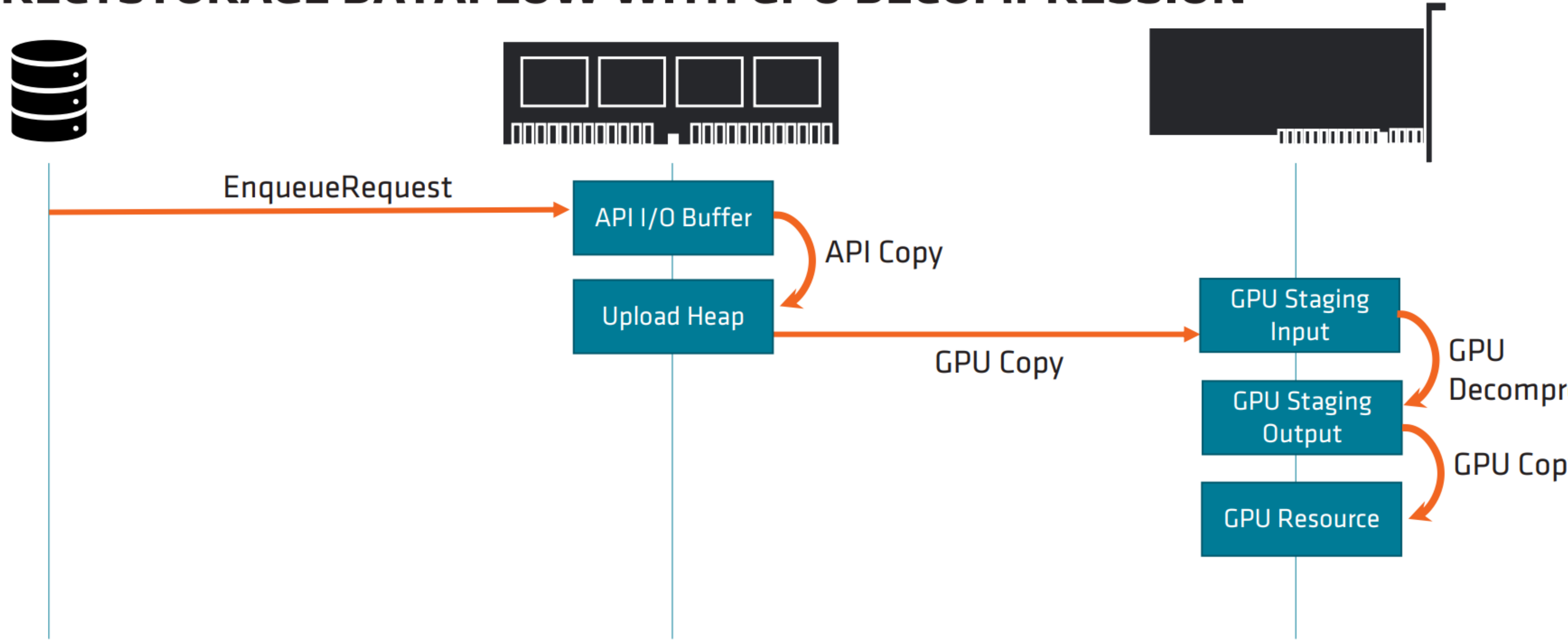
Up to 4x Copies

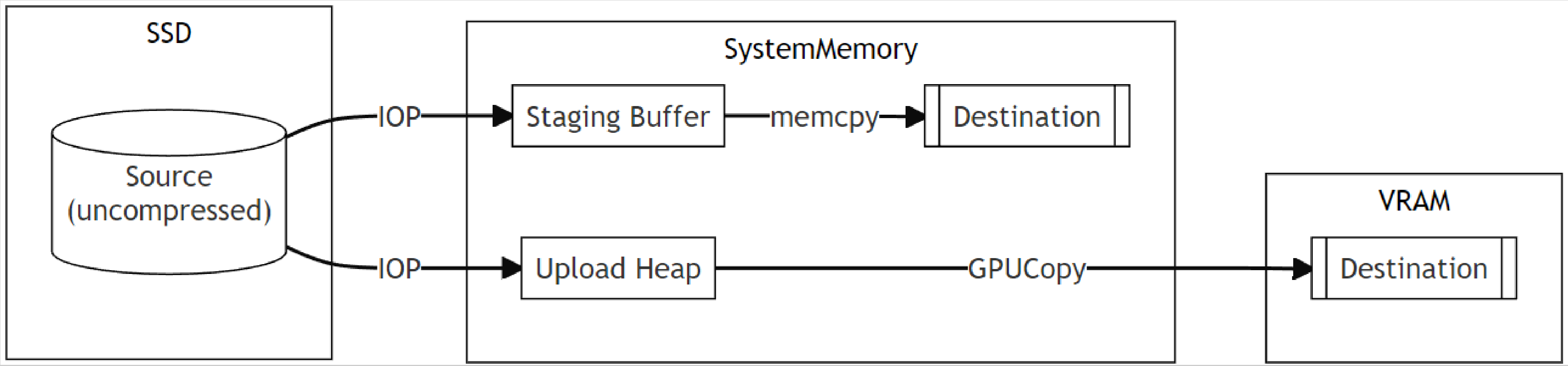
TRADITIONAL WIN32 DATAFLOW WITH CPU DECOMPRESSION

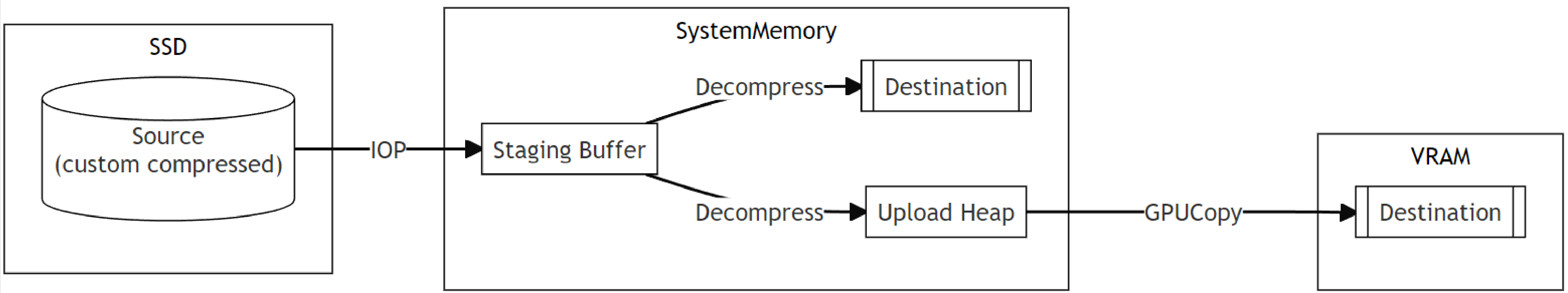


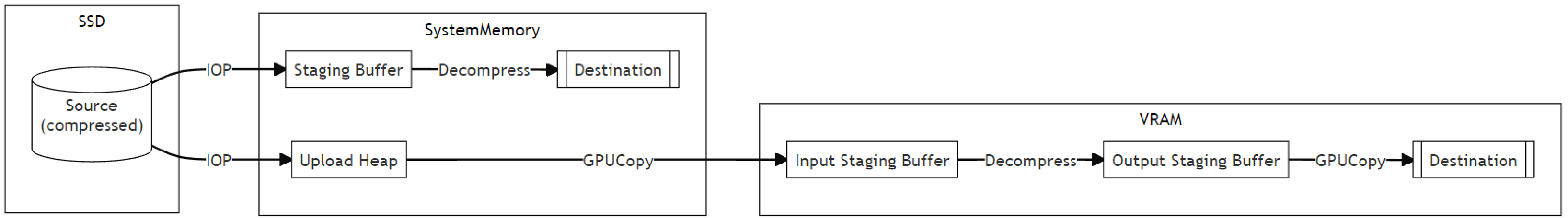
Up to 5x Copies

DIRECTSTORAGE DATAFLOW WITH GPU DECOMPRESSION




SSD
Source (uncompressed)
IOP
IOP
SystemMemory
Staging Buffer
memcpy
Destination
Upload Heap
GPUCopy
VRAM
Destination


SSD
Source (custom compressed)
IOP
SystemMemory
Staging Buffer
Decompress
Destination
Decompress
Upload Heap
GPUCopy
VRAM
Destination


SSD
Source (compressed)
IOP
IOP
SystemMemory
Staging Buffer
Decompress
Destination
Upload Heap
GPUCopy
VRAM
Input Staging Buffer
Decompress
Output Staging Buffer
GPUCopy
Destination

02

Why

Direct Storage

- Instead of loading large chunks at a time with very few IO requests, games now break assets like textures down into smaller pieces, only loading in the pieces that are needed for the current scene being rendered. **previous gen games had an asset streaming budget on the order of 50MB/s** which even at smaller 64k block sizes (ie. one texture tile) amounts to only hundreds of IO requests per second.

- Existing APIs require the application to manage and handle each of these requests one at a time first by submitting the request, waiting for it to complete, and then handling its completion. The overhead of each request is not very large and wasn't a choke point for older games running on slower hard drives, but multiplied tens of thousands of times per second, **IO overhead can quickly become too expensiv**e preventing games from being able to take advantage of the increased NVMe drive bandwidths. **Current storage APIs were not optimized for this high number of IO requests**, preventing them from scaling up to these higher NVMe bandwidths creating bottlenecks that limit what games can do.

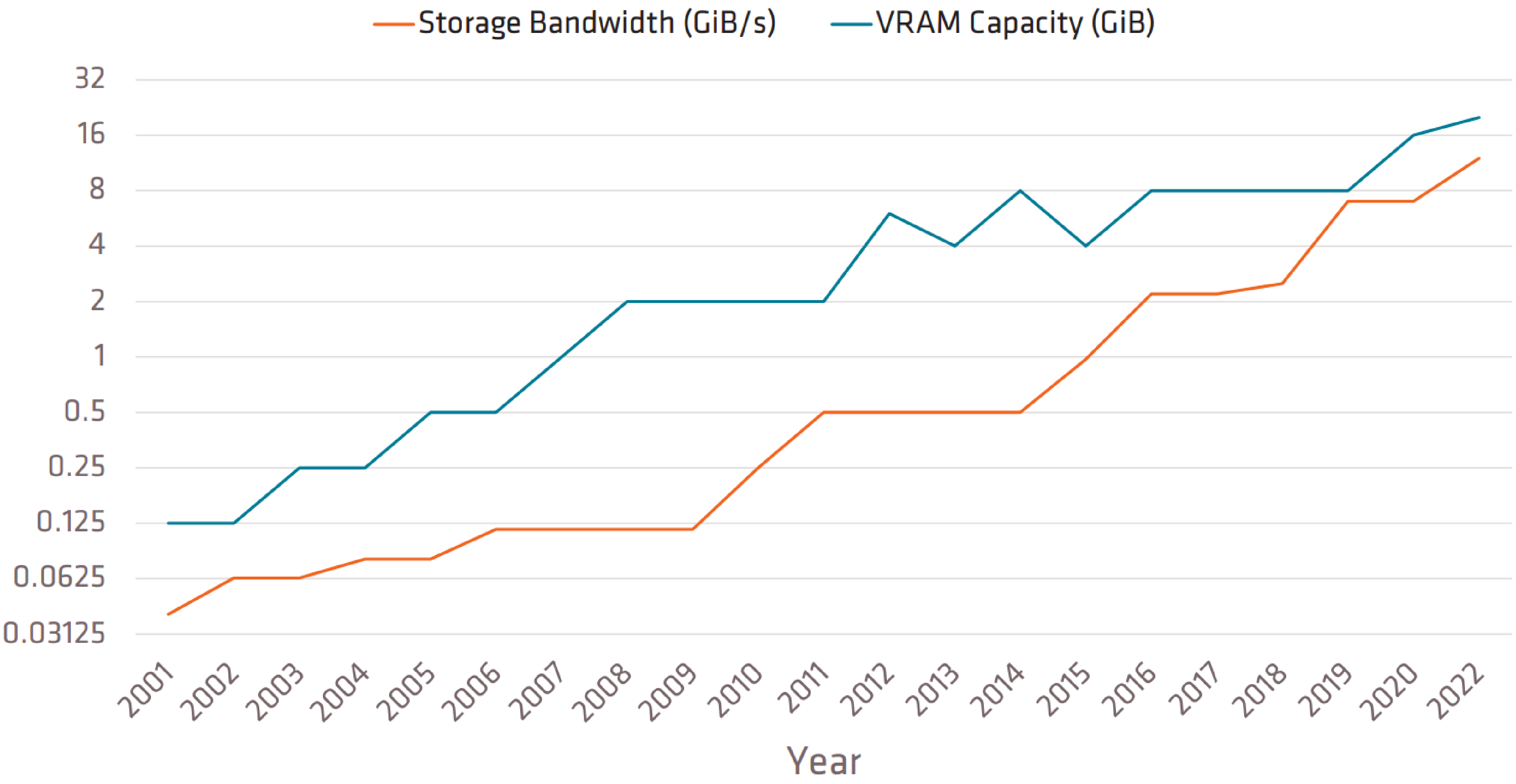
- Many of these assets are compressed. In order to be used by the CPU or GPU, they must first be decompressed. A game can pull as much data off the disk as it wants, but you still need an efficient way to decompress and get it to the GPU for rendering.

- Reducing per-request NVMe overhead, enabling batched many-at-a-time parallel IO requests which can be efficiently fed to the GPU, and giving games finer grain control over when they get notified of IO request completion instead of having to react to every tiny IO completion.

WHY?

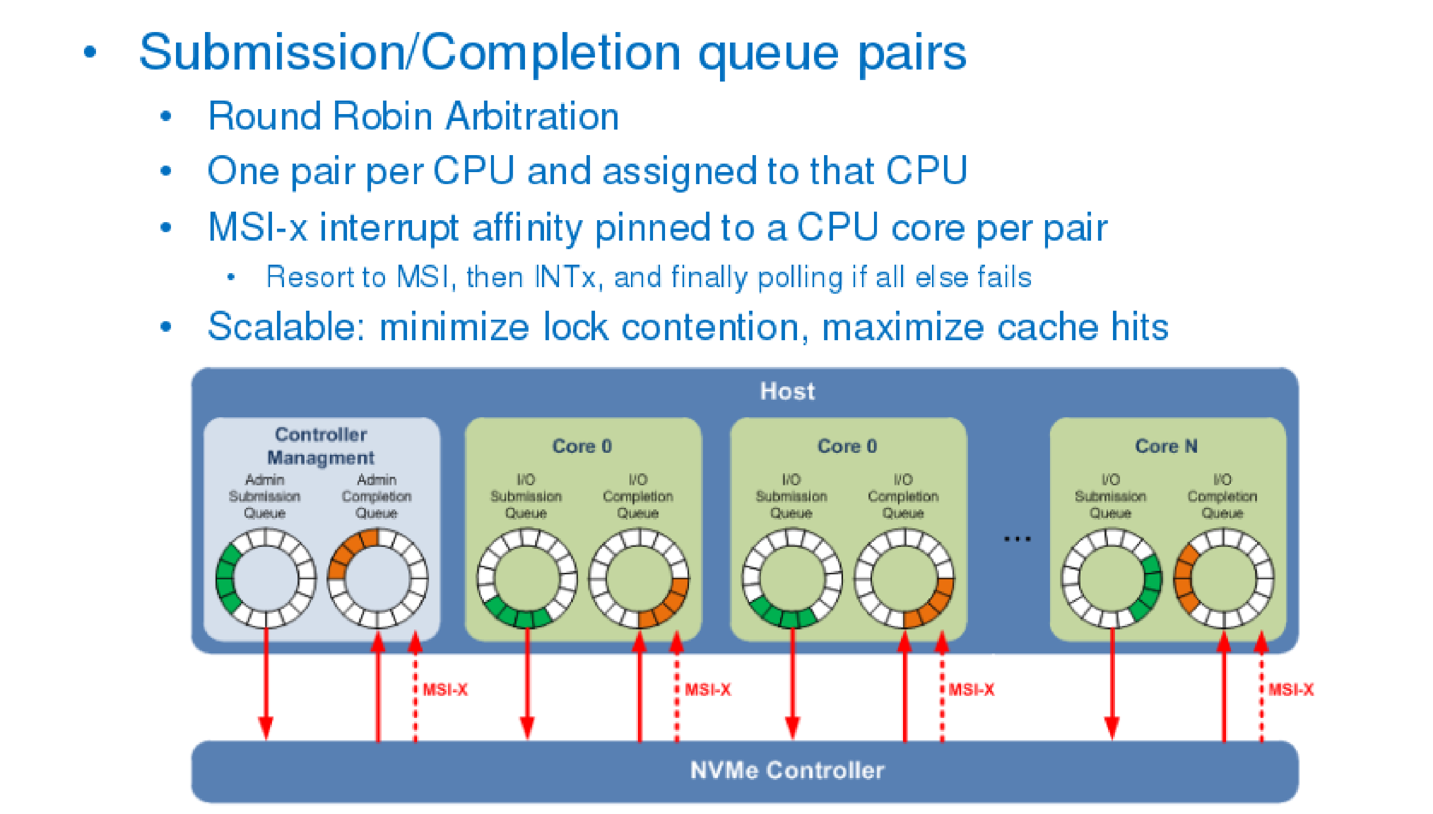
- Storage bandwidth has rapidly increased, and latency has decreased
- Standard Win32 I/O API doesn't encourage efficient usage
- More uniformity of API between console and PC
- A more "direct" route to placing assets in memory as resources… should be faster

STORAGE BANDWIDTH AND VRAM CAPACITY (LOG2 SCALE)



Direct Storage

- To get data off the drive, an OS submits a request to the drive and data is delivered to the app via these queues. **An NVMe device can have multiple queues and each queue can contain many requests at a time.** The DirectStorage programming model essentially gives developers direct control over that highly optimized hardware.
- The CPU shouldn’t become the bottleneck that holds back the I/O subsystem.



- Submission/Completion queue pairs
  - Round Robin Arbitration
  - One pair per CPU and assigned to that CPU
  - MSI-x interrupt affinity pinned to a CPU core per pair
    - Resort to MSI, then INTx, and finally polling if all else fails
  - Scalable: minimize lock contention, maximize cache hits

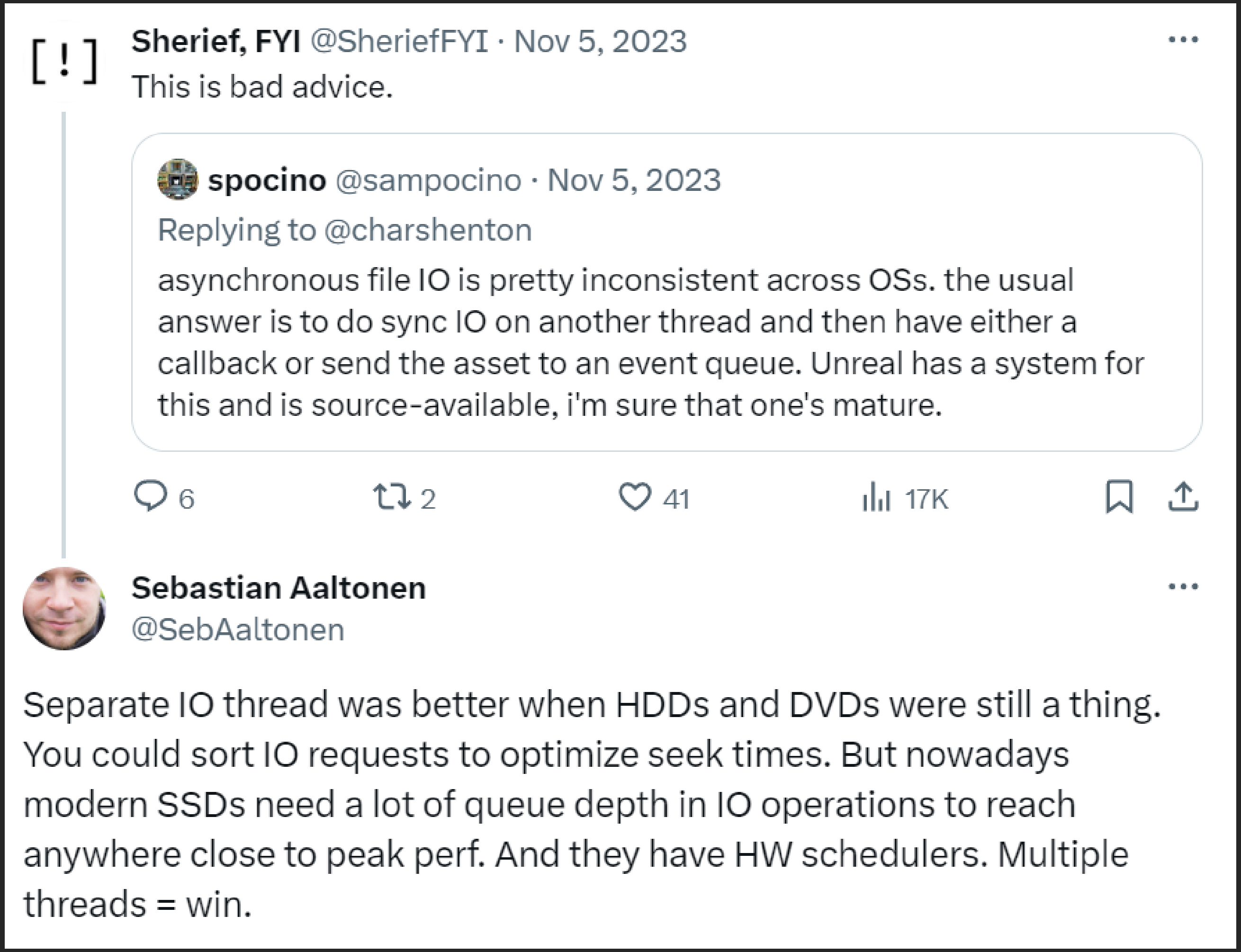
03

IO

Four types of IO

- 同步文件读取：读取请求直接陷入内核，读取块后立即返回线程现场和数据块。这样的接口包含 fread, ReadFile, fstream 等(稳妥的做法)
- 异步文件读取：读取请求从软件线程上解绑，在内核中异步。当请求完成时，内核激活信号量告知给应用程序。这样的接口包含 ReadFileEx 等，使用这种方式读取文件的典型库有 Boost.ASIO， IOCP等；
- 文件映射：不使用页面缓冲，将文件直接映射到虚拟地址空间的连续段上。随后用户可以访问这段虚拟内存来访问映射文件。这样的接口包含 mmap， CreateFileMapping 等; (PageFault)
- DirectStorage等：直接操作 NVMe 或是定制设备的并发队列。这样的接口包含 Microsoft DirectStorage，或者可以通过调用 linux NVMe 驱动实现; (真正发挥 NVMe 性能的 API)

IO Advice



Sherief, FYI @SheriefFYI · Nov 5, 2023

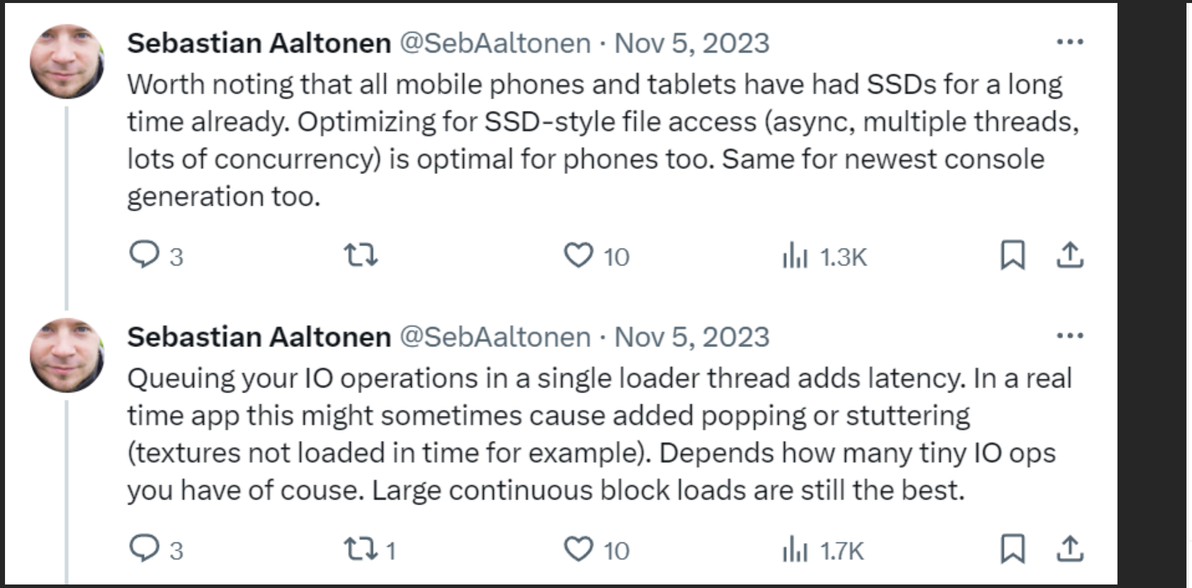
This is bad advice.

> spocino @sampocino · Nov 5, 2023
>
> Replying to @charshenton
>
> asynchronous file IO is pretty inconsistent across OSs. the usual answer is to do sync IO on another thread and then have either a callback or send the asset to an event queue. Unreal has a system for this and is source-available, i'm sure that one's mature.

Sebastian Aaltonen
@SebAaltonen

Separate IO thread was better when HDDs and DVDs were still a thing. You could sort IO requests to optimize seek times. But nowadays modern SSDs need a lot of queue depth in IO operations to reach anywhere close to peak perf. And they have HW schedulers. Multiple threads = win.

IO Advice

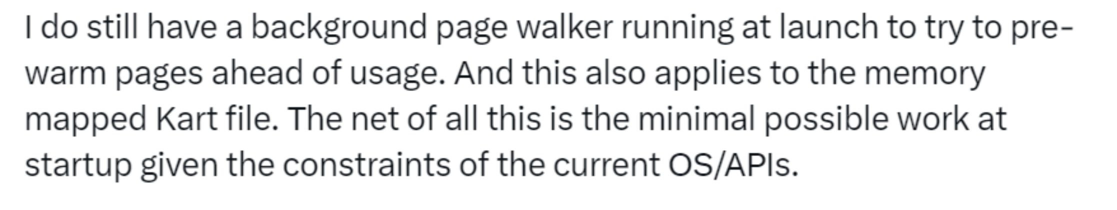


Sebastian Aaltonen @SebAaltonen · Nov 5, 2023

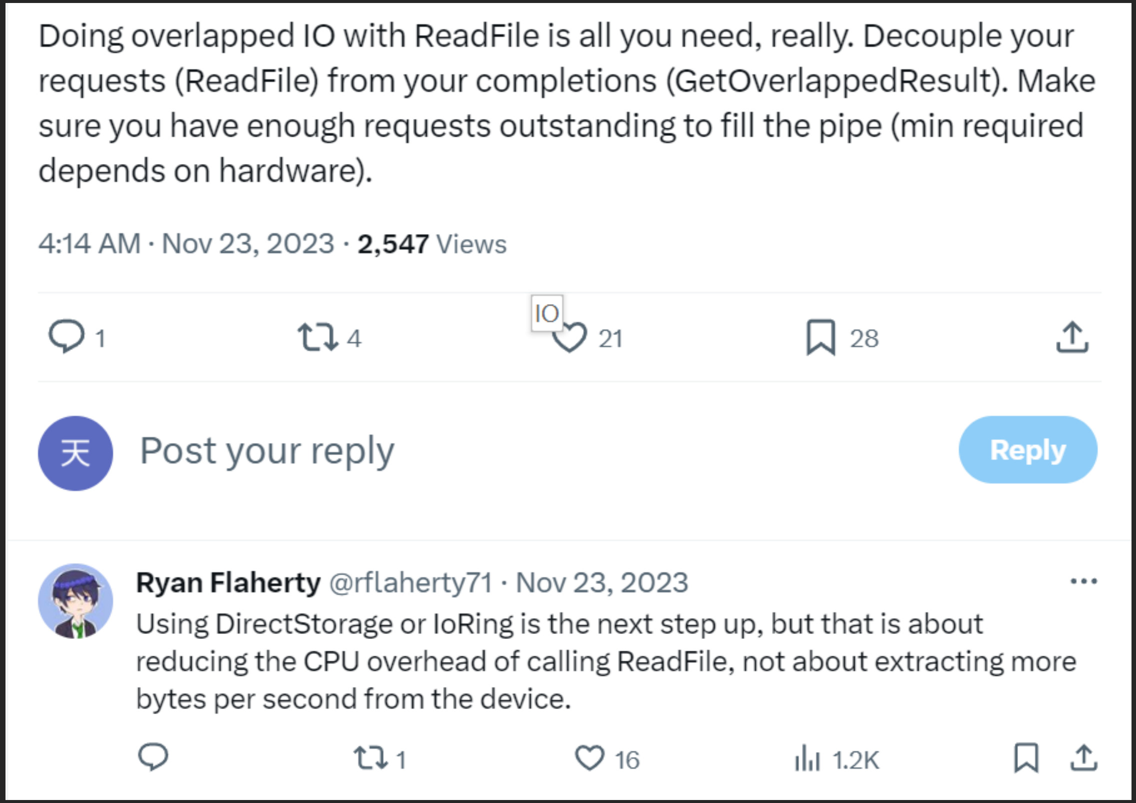
Worth noting that all mobile phones and tablets have had SSDs for a long time already. Optimizing for SSD-style file access (async, multiple threads, lots of concurrency) is optimal for phones too. Same for newest console generation too.

Sebastian Aaltonen @SebAaltonen · Nov 5, 2023

Queuing your IO operations in a single loader thread adds latency. In a real time app this might sometimes cause added popping or stuttering (textures not loaded in time for example). Depends how many tiny IO ops you have of couse. Large continuous block loads are still the best.



I do still have a background page walker running at launch to try to pre-warm pages ahead of usage. And this also applies to the memory mapped Kart file. The net of all this is the minimal possible work at startup given the constraints of the current OS/APIs.



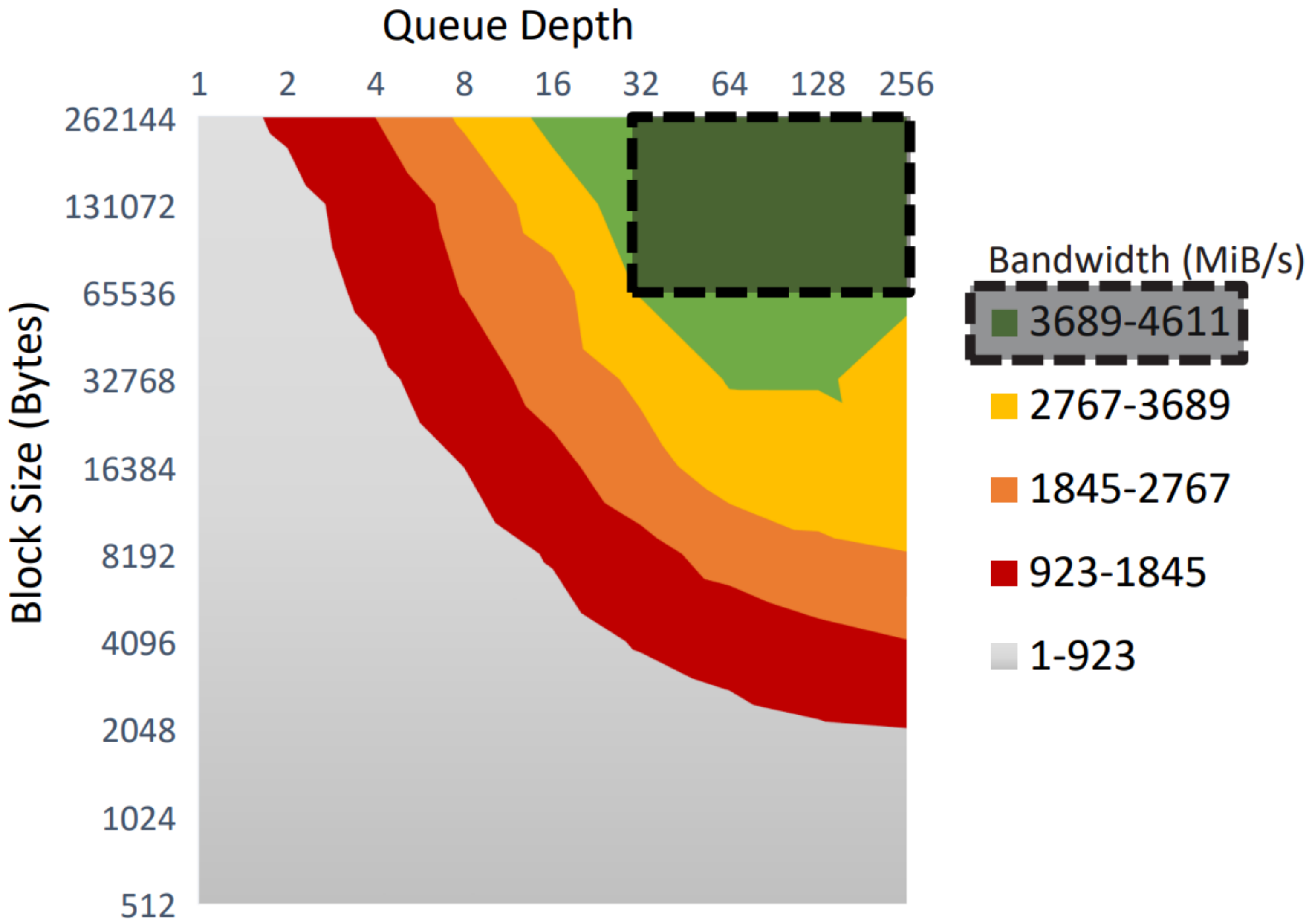
Doing overlapped IO with ReadFile is all you need, really. Decouple your requests (ReadFile) from your completions (GetOverlappedResult). Make sure you have enough requests outstanding to fill the pipe (min required depends on hardware).

4:14 AM · Nov 23, 2023 · **2,547** Views

Ryan Flaherty @rflaherty71 · Nov 23, 2023

Using DirectStorage or IoRing is the next step up, but that is about reducing the CPU overhead of calling ReadFile, not about extracting more bytes per second from the device.

BANDWIDTH HEATMAP – MAINTAING BEST 20% PERFORMANCE



ENABLING BATCHING: DECOUPLING DATA DEPENDENCIES

- Improve I/O performance regardless of the API choice

- Required to realize performance benefits of DirectStorage which is designed for batching and parallelism

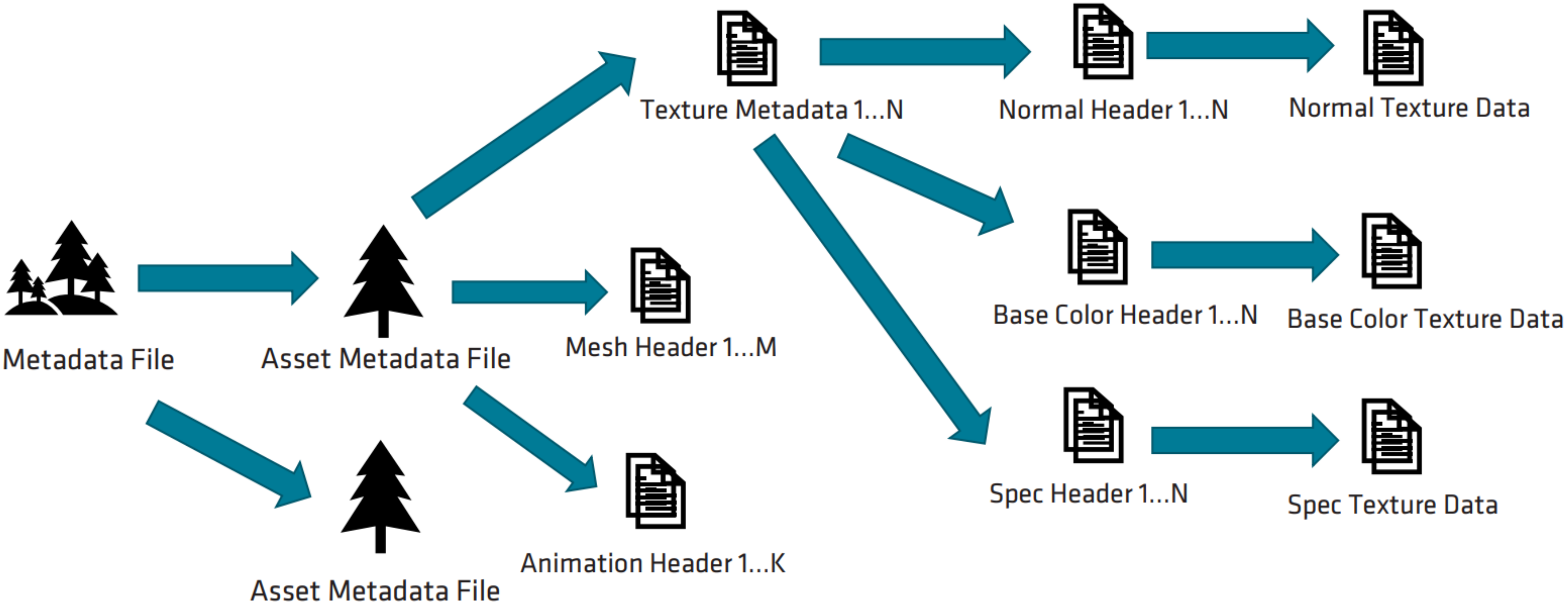
BEFORE WE CAN BATCH... WE MUST DECOUPLE DATA DEPENDENCIES

- What is tight data dependency coupling?
  - Read->Read dependencies create stalls

- What causes this?
  - Storing data necessary for loading assets in several places separately causing multiple reads
  - Often a consequence of the editor pipeline and workflow

- Simplified dependency example:
  - Read Texture Header->Read Texture Data->Read Each Mip Map...

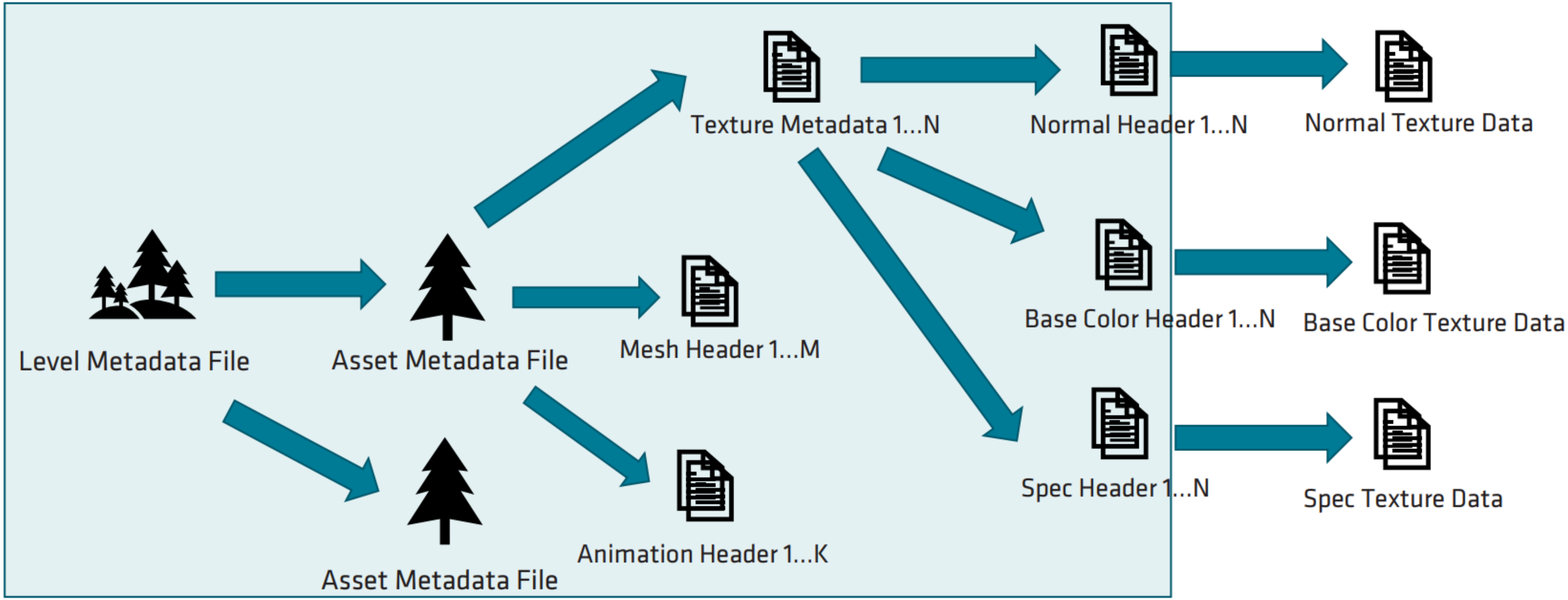
BUT GAME ENGINES ARE MORE COMPLEX...



Legend

➡ = Read

CONSIDER KEEPING THIS IN RAM IN A COLLAPSED FORMAT

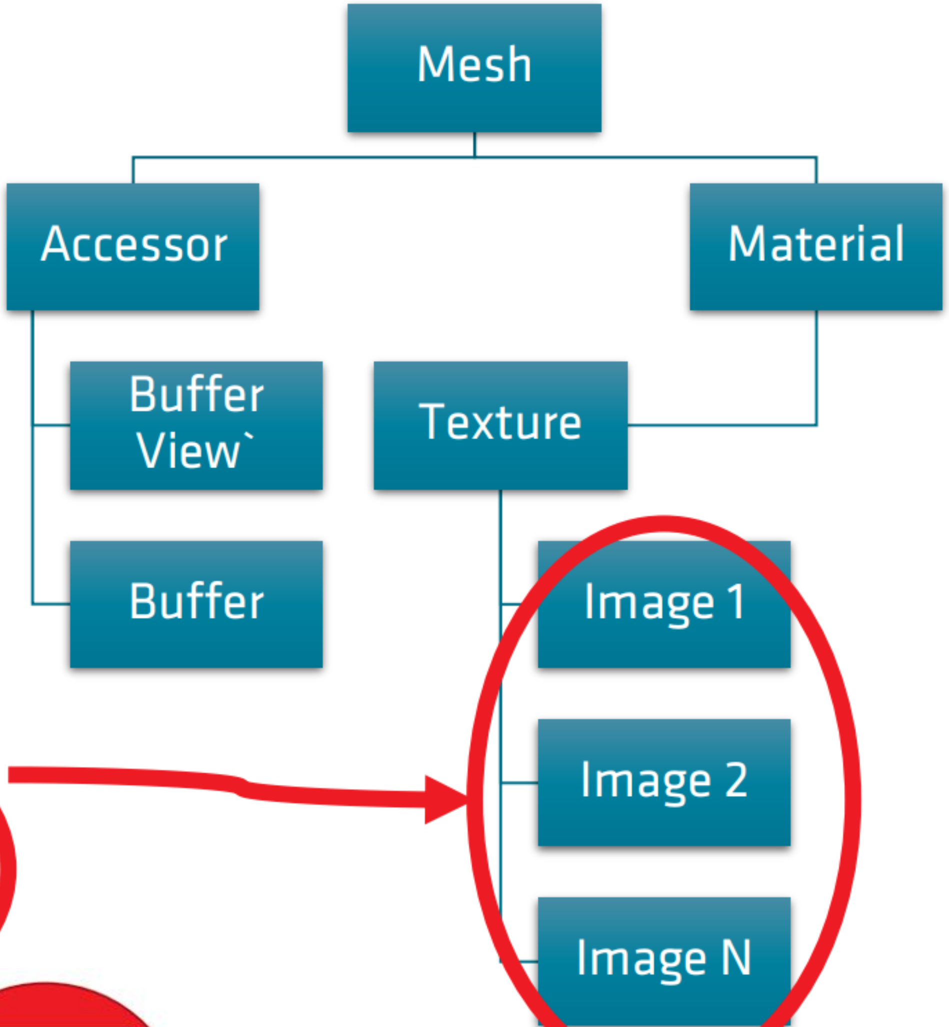


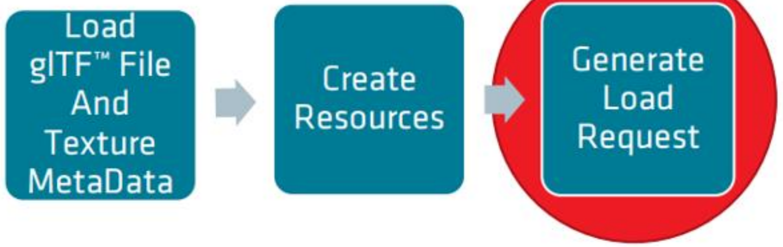
Legend

➡ = Read

GLTF™ EXAMPLE

- glTF™ is an API neutral 3D asset transmission format.
  - JSON
  - Images reference separate image files in PNG or JPG format as Uniform Resource Identifier (similar to URL).





WHAT OTHER DATAFLOW IMPROVEMENTS HAVE BEEN MADE?

- DirectStorage automatically uses the Copy Queue.

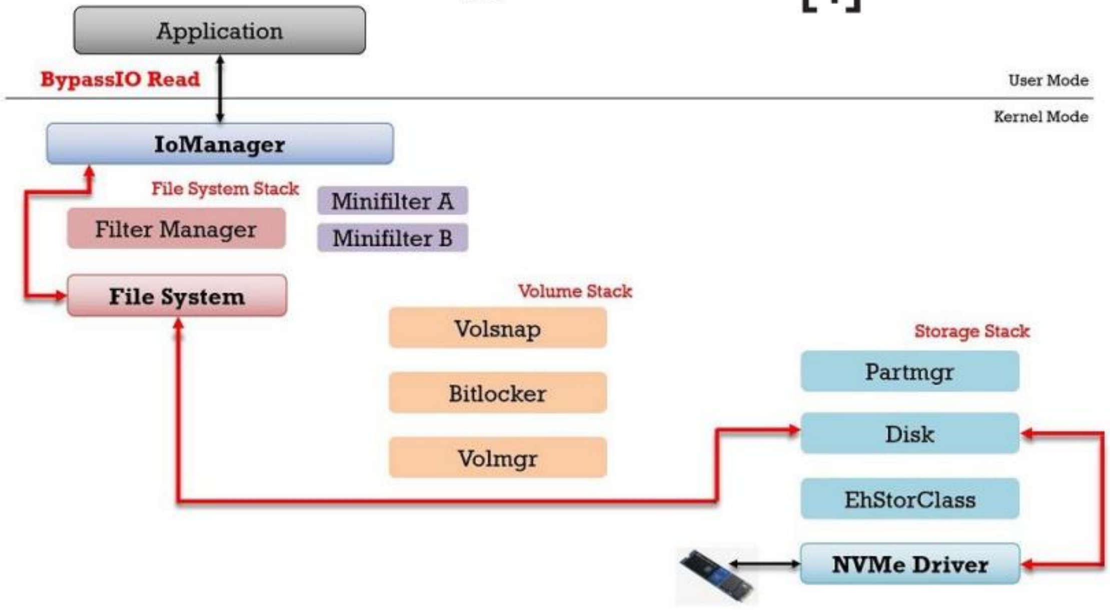
- Windows® 11's new I/O capabilities improve I/O throughput and CPU utilization, and DirectStorage takes advantage of these transparently.

  - Bypass I/O
    - Bypasses significant portions of the kernel I/O stack

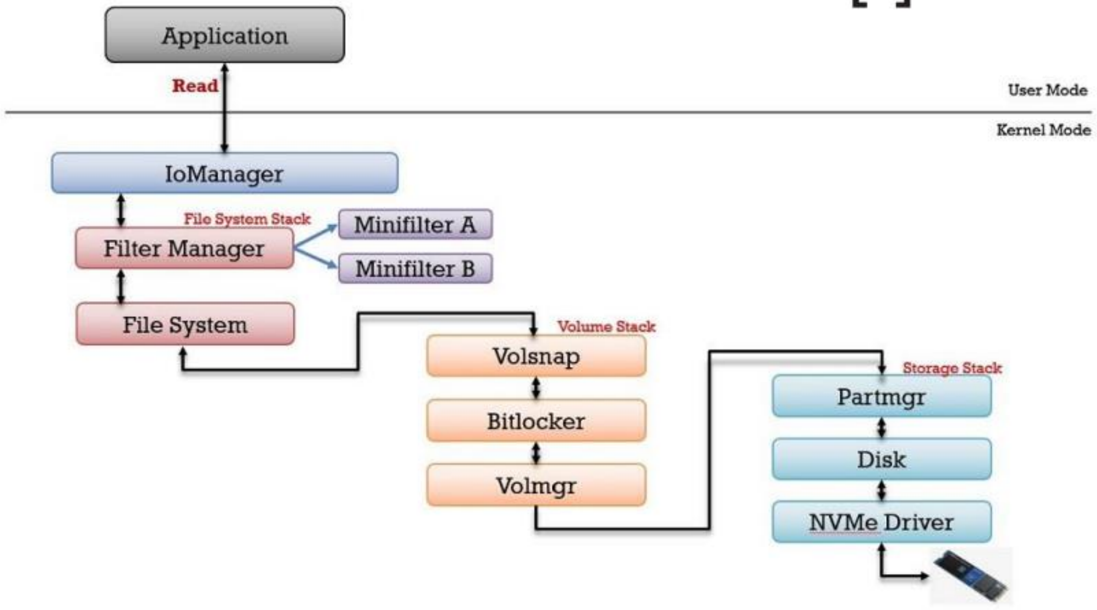
  - IORing File API
    - Batches I/O requests and completion notifications to reduce the CPU overhead of associated context switching

BYPASSIO

BypassIO Model[1]



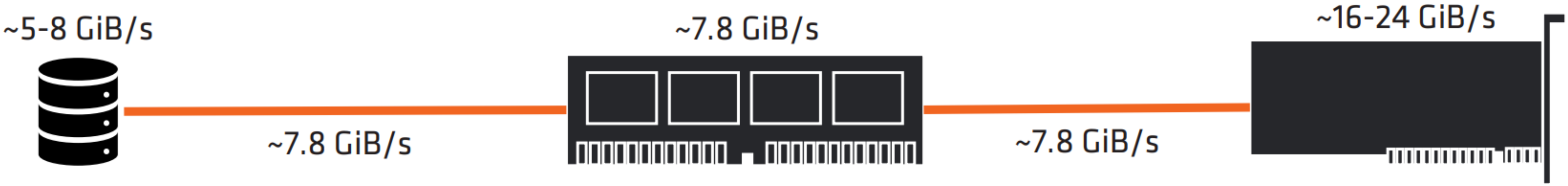
Traditional IO Model[1]



04

Decompression

WHY GPU DECOMPRESSION?



During load time, what are you doing with those GPU cycles anyway?

Most CPU cycles needed for:

- Compiling shaders
- Decompressing game data
- Initializing game objects

COMPRESSION OPTIONS

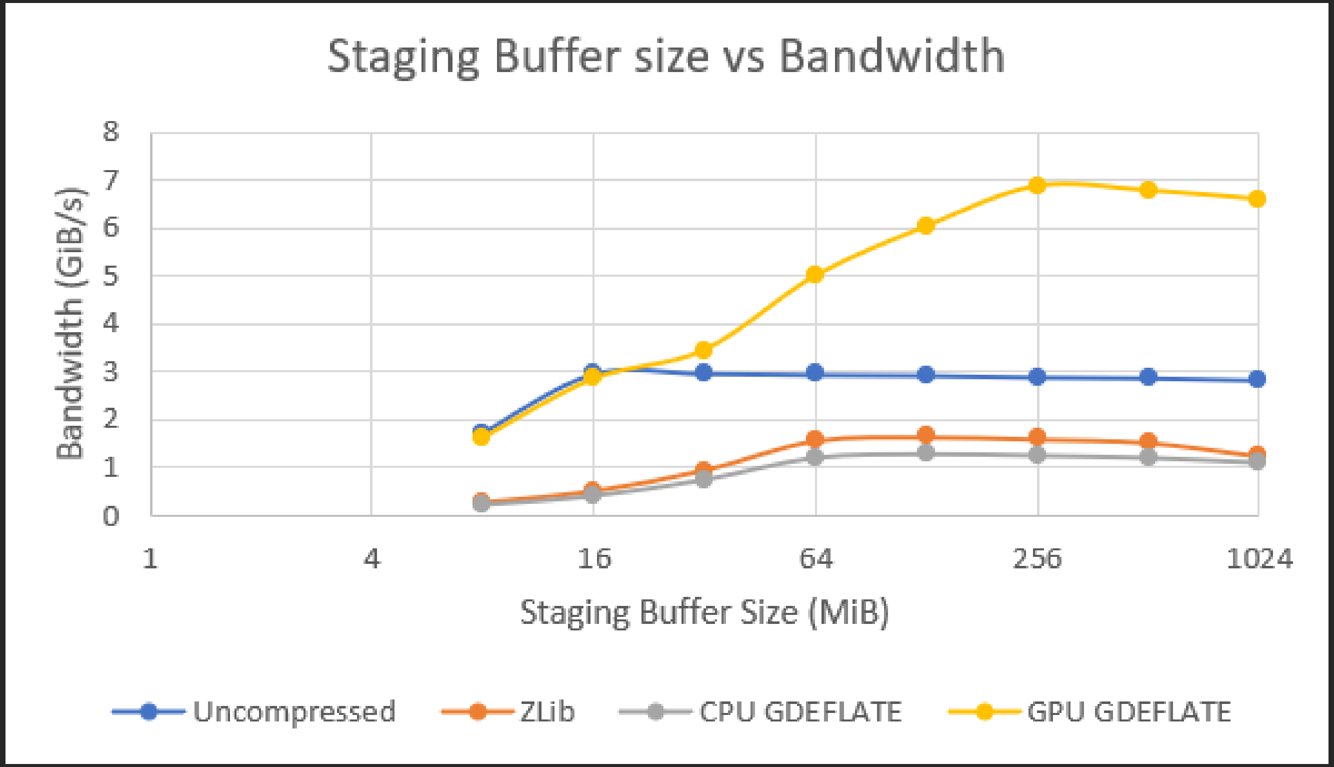
- None
  - This means no compression at all. This will be useful on data which isn’t very compressible to begin with such as already compressed data (compressed video, jpeg, etc.).

- Built-in
  - GDeflate – General purpose compression algorithm targeting the GPU. The tools for it are built into the API and it can be downloaded from GitHub for free to integrate into your own tools if necessary.

- Custom
  - Implement your own custom compression for CPU decompression. **Please do not try to implement your own GPU decompression. It most likely won’t perform as well as any built into the API.**

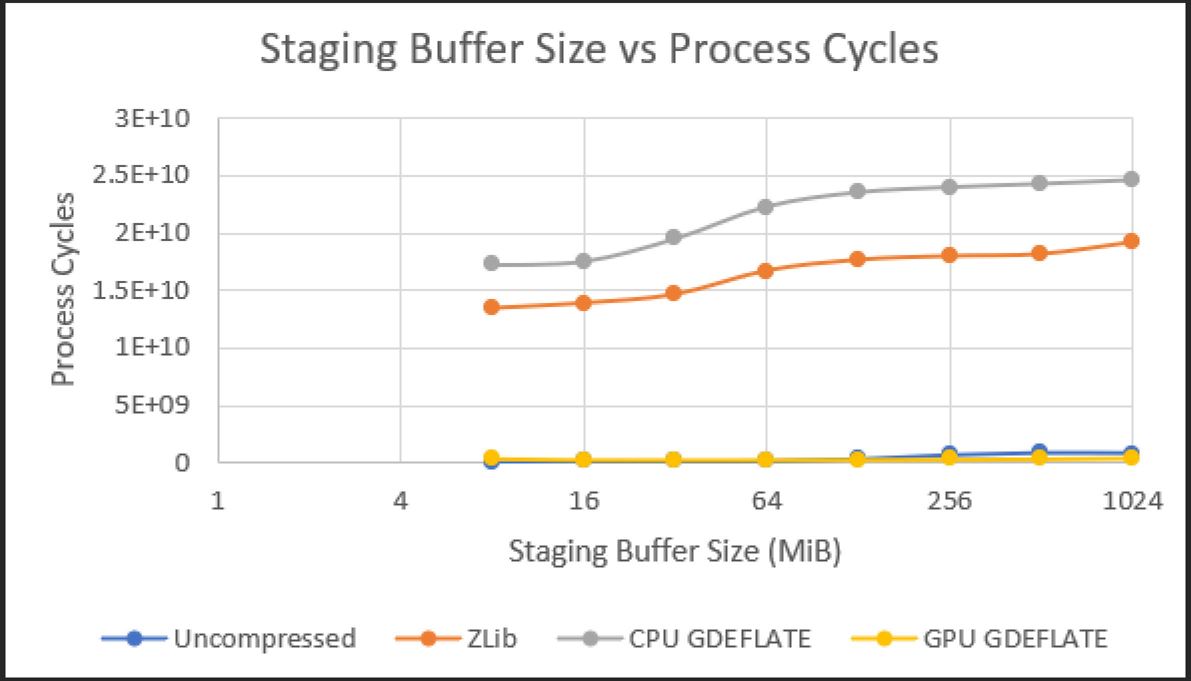
GDeflate

- DEFLATE is a compression format that uses a combination of LZ77 compression and Huffman coding. Literals and match copy lengths use a single Huffman tree and match copy distances use a separate Huffman tree.
- A many-core SIMD machine consumes the GDeflate bitstream by design, explicitly exposing parallelism at two levels.



GDeflate

- We designed GDeflate with the following goals:
  - GPU-optimized decompression to support the fastest NVMe devices
  - Offload the CPU to avoid making it the bottleneck during I/O operations
  - Portable to a variety of data-parallel architectures, including CPUs and GPUs
  - Can be implemented cheaply in fixed-function hardware, using existing IP
  - Establish as a data-parallel data compression standard



GDeflate

- Deciding the staging buffer size:
  - make sure to benchmark different platforms with varying NVMe, PCIe, and GPU capabilities when deciding on the staging buffer size. We found that a 128-MB staging buffer size is a reasonable default. Smaller GPUs may require less and larger GPUs may require more.
- Compression ratio considerations:
  - Make sure to measure the impact that different resource types have on compression savings and GPU decompression performance.
  - In general, various data types, such as texture and geometry, compress at different ratios. This can cause some variation in GPU decompression execution performance.
  - This won't have a significant effect on end-to-end throughput. However, it may result in variation in latency when delivering the resource contents to their final locations.
- Windows File System:
  - Try to keep disk files accessed by DirectStorage separate from files accessed by other I/O APIs. Shared file use across different I/O APIs may result in the loss of bypass I/O improvements.

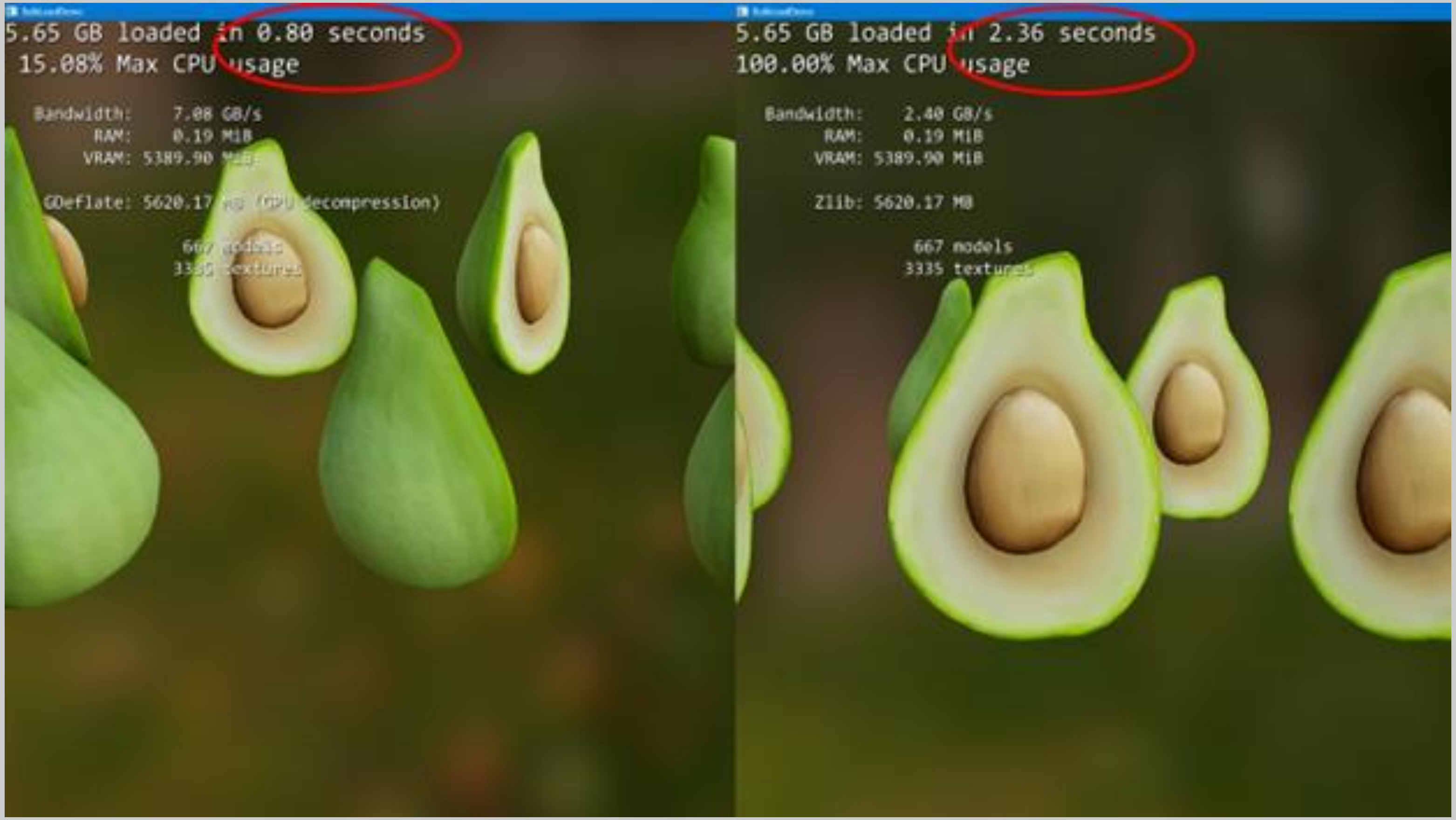
- it's necessary to rethink game engines' resource streaming architecture, and fully leverage improvements in I/O technology.

05

Benchmark

MS Demo性能对比

GPU with GDeflate (left) loading in 0.8 seconds vs CPU with Zlib (right) in 2.36 seconds.



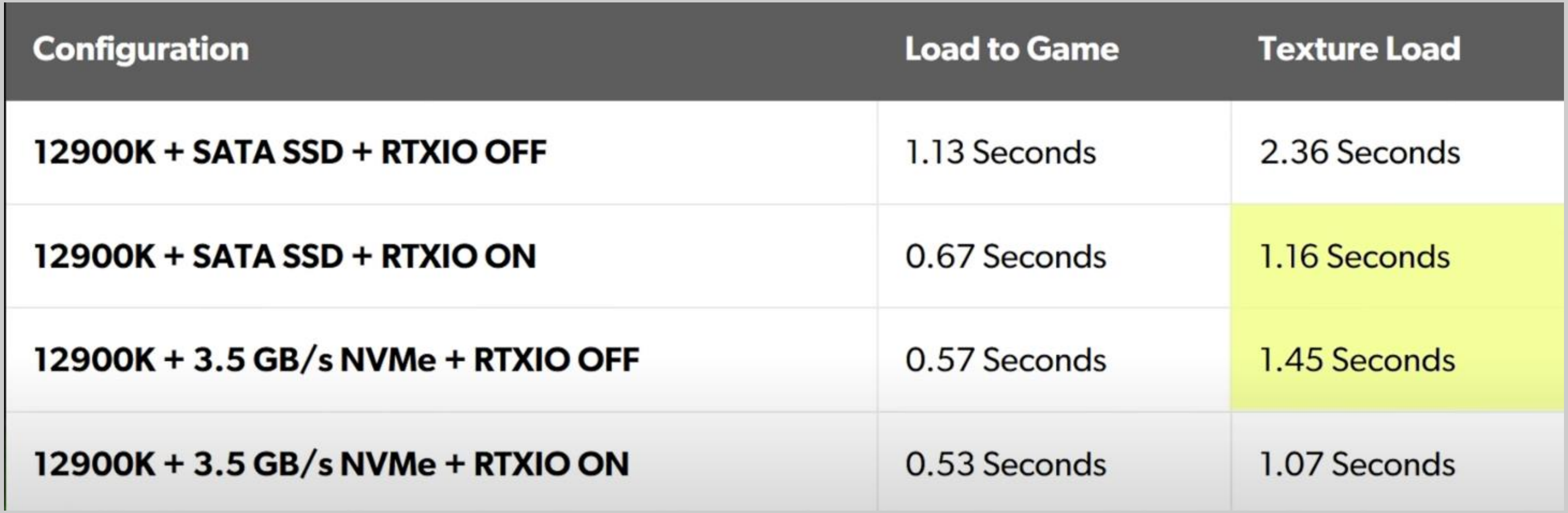
```
5.65 GB loaded in 0.80 seconds
 15.08% Max CPU usage

  Bandwidth:    7.08 GB/s
       RAM:    0.19 MiB
      VRAM: 5389.90 MiB

  GDeflate: 5620.17 MB (GPU decompression)

              667 models
             3335 textures
```

```
5.65 GB loaded in 2.36 seconds
100.00% Max CPU usage

  Bandwidth:    2.40 GB/s
       RAM:    0.19 MiB
      VRAM: 5389.90 MiB

      Zlib: 5620.17 MB

              667 models
             3335 textures
```

Amd Demo性能对比

mapName	mapSize Compressed(MiB)	mapSize Uncompressed (MiB)	DS On loadime	DS Off loadTime(ms)
CheckerPlane	0	0	48.31	138.12
CommandModule	915.3	2758.72	503.25	2343.35
BoomBox	10.86	85.37	61.12	229.83
SpaceShuttle	926.78	2475.79	517.91	2125.09
X1	29.37	170.69	142.85	668.82

Portal Prelude性能对比



Configuration	Load to Game	Texture Load
12900K + SATA SSD + RTXIO OFF	1.13 Seconds	2.36 Seconds
12900K + SATA SSD + RTXIO ON	0.67 Seconds	1.16 Seconds
12900K + 3.5 GB/s NVMe + RTXIO OFF	0.57 Seconds	1.45 Seconds
12900K + 3.5 GB/s NVMe + RTXIO ON	0.53 Seconds	1.07 Seconds

00

APPENDIX

参考资料

- https://www.youtube.com/watch?v=LvYUmVtOMRU
- https://zhuanlan.zhihu.com/p/613665728
- https://developer.nvidia.com/blog/accelerating-load-times-for-directx-games-and-apps-with-gdeflate-for-directstorage/
- https://hyunyoung2.github.io/2016/05/20/NVMe/
- https://gpuopen.com/gdc-presentations/2023/GDC-2023-DirectStorage-optimizing-load-time-and-streaming.pdf
- https://www.nvidia.com/en-us/geforce/news/rtx-io-for-geforce-gpus-available-now/
- https://www.youtube.com/watch?v=o9Sf4H2rYM8
- https://github.com/microsoft/DirectStorage