DIRECTSTORAGE

OPTIMIZING LOAD-TIME AND STREAMING

DAVID ZIMAN

together we advance_

Reticulating Splines…

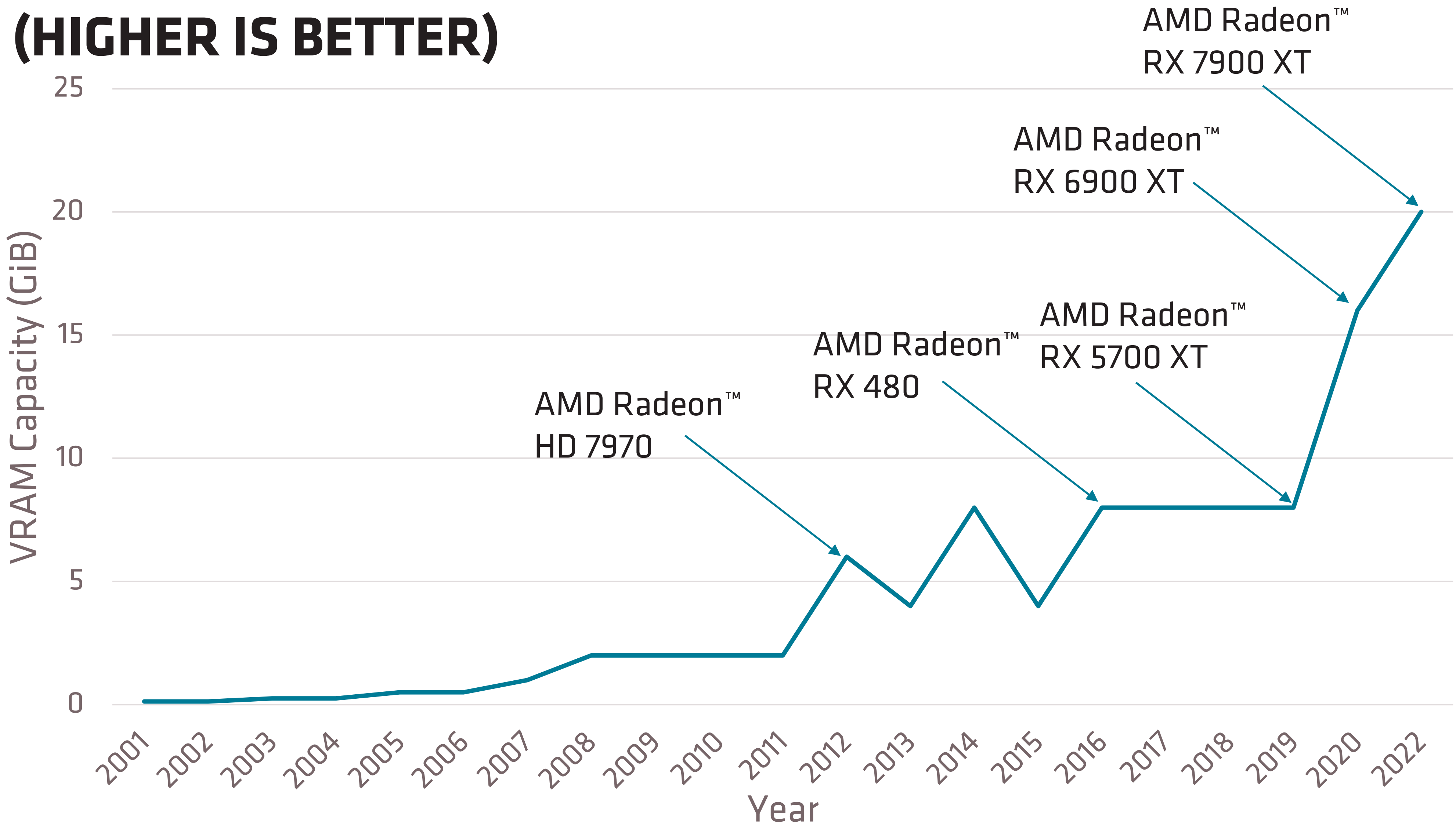
AGENDA

- Why do we need another Storage API?
- Enabling fast load times
- Dataflow Improvements
- Demo
- Profiling Suggestions
- Questions?

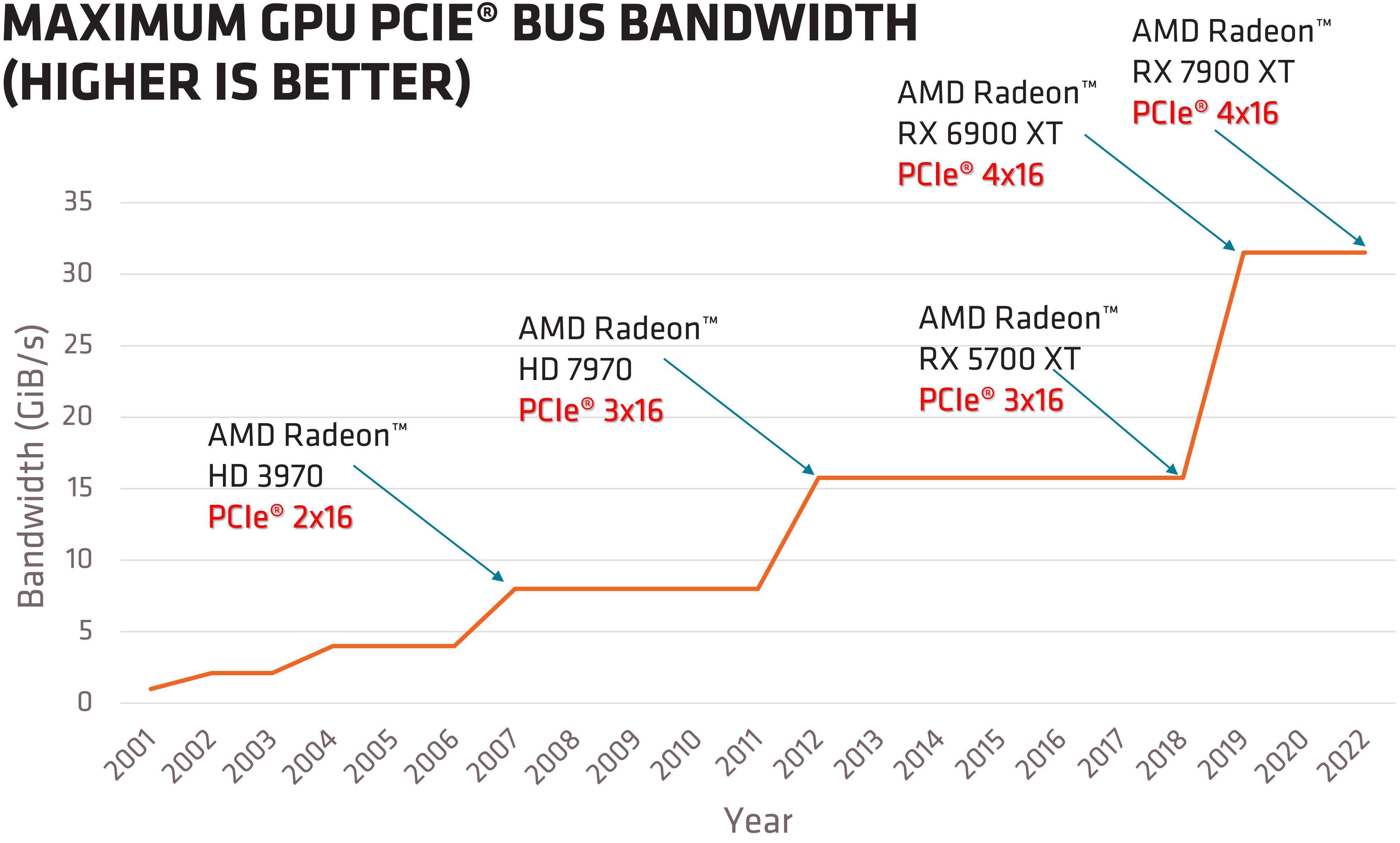
WHY?

- 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

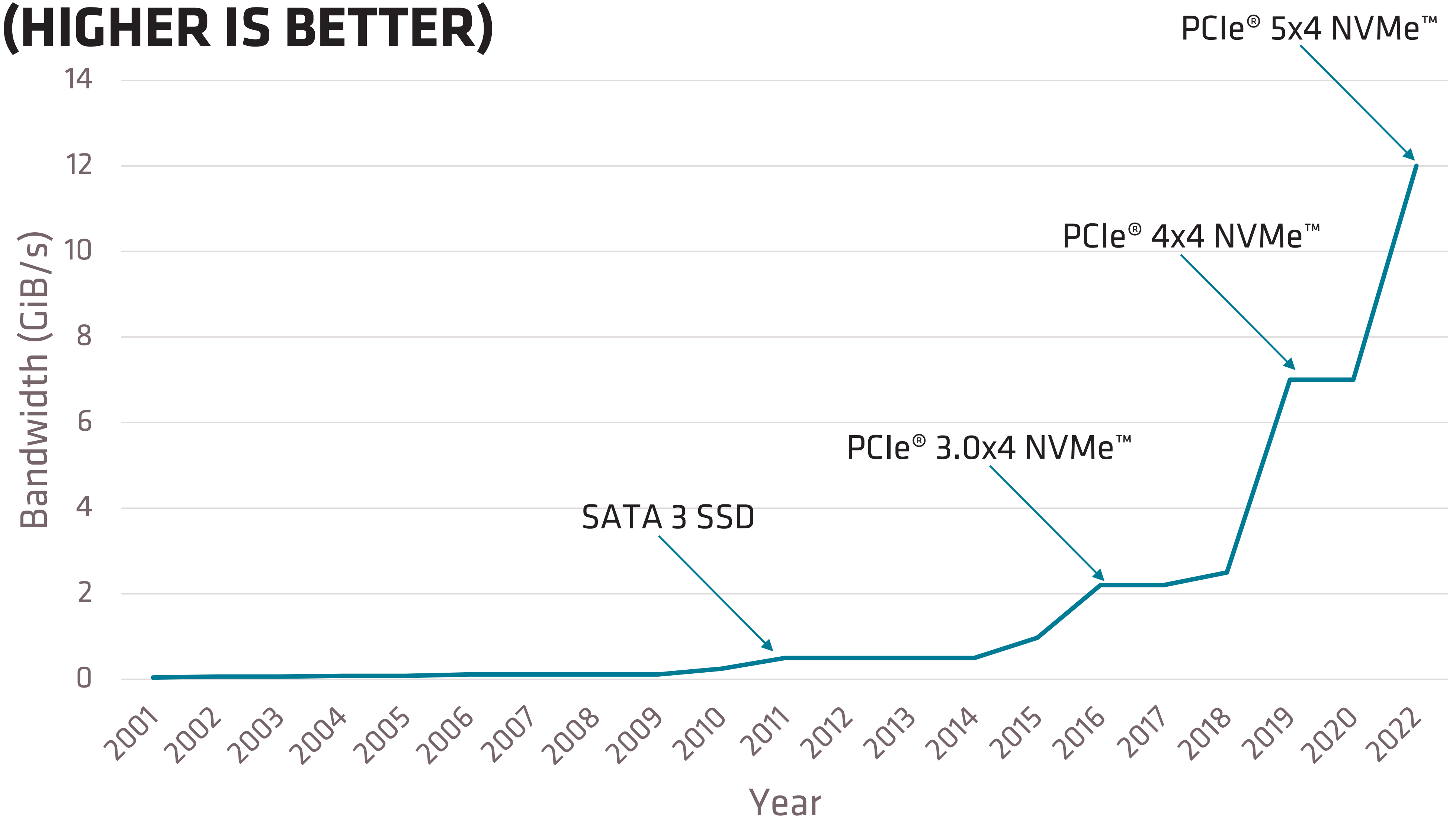
VRAM CAPACITY (HIGHER IS BETTER)



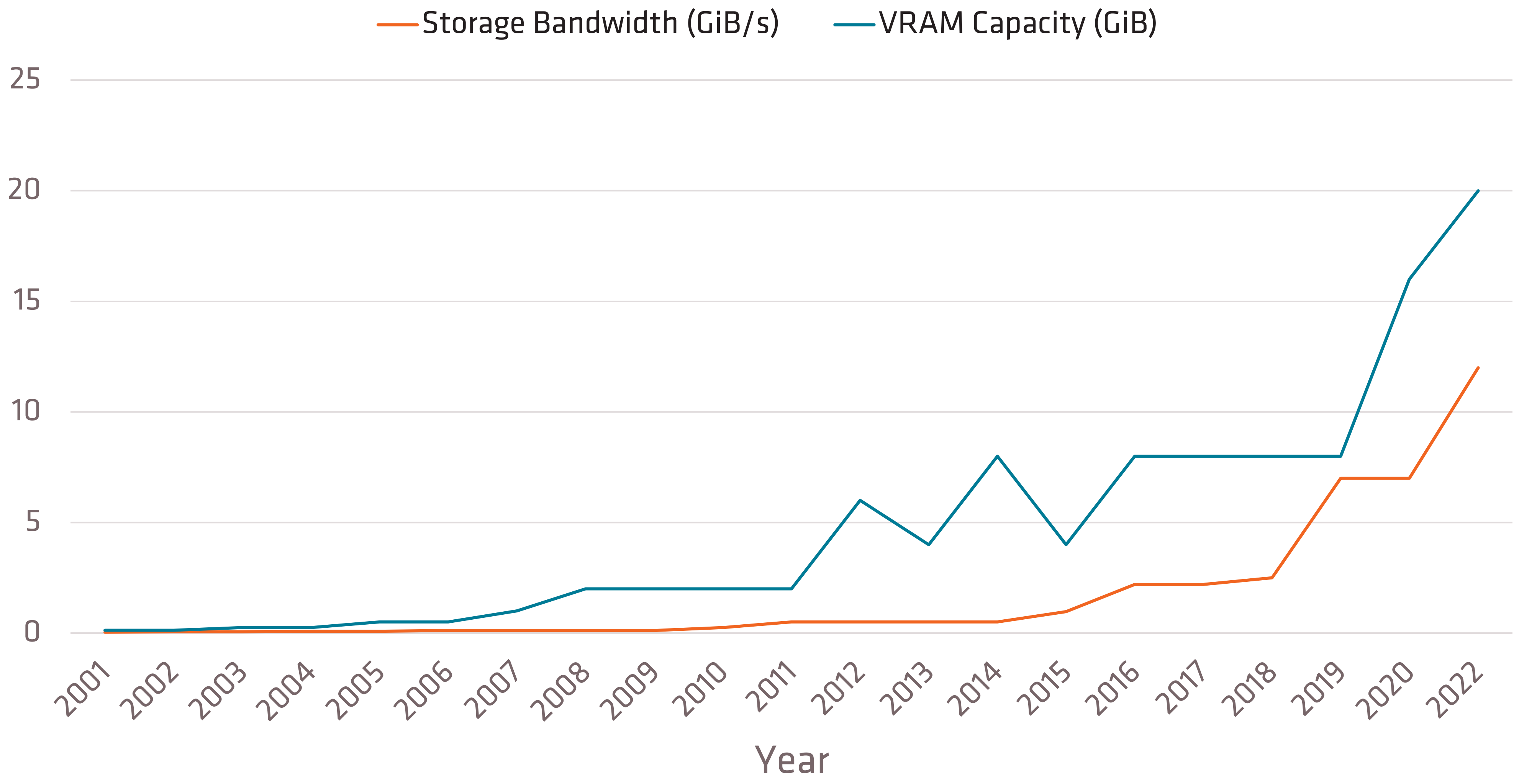
MAXIMUM GPU PCIE® BUS BANDWIDTH (HIGHER IS BETTER)



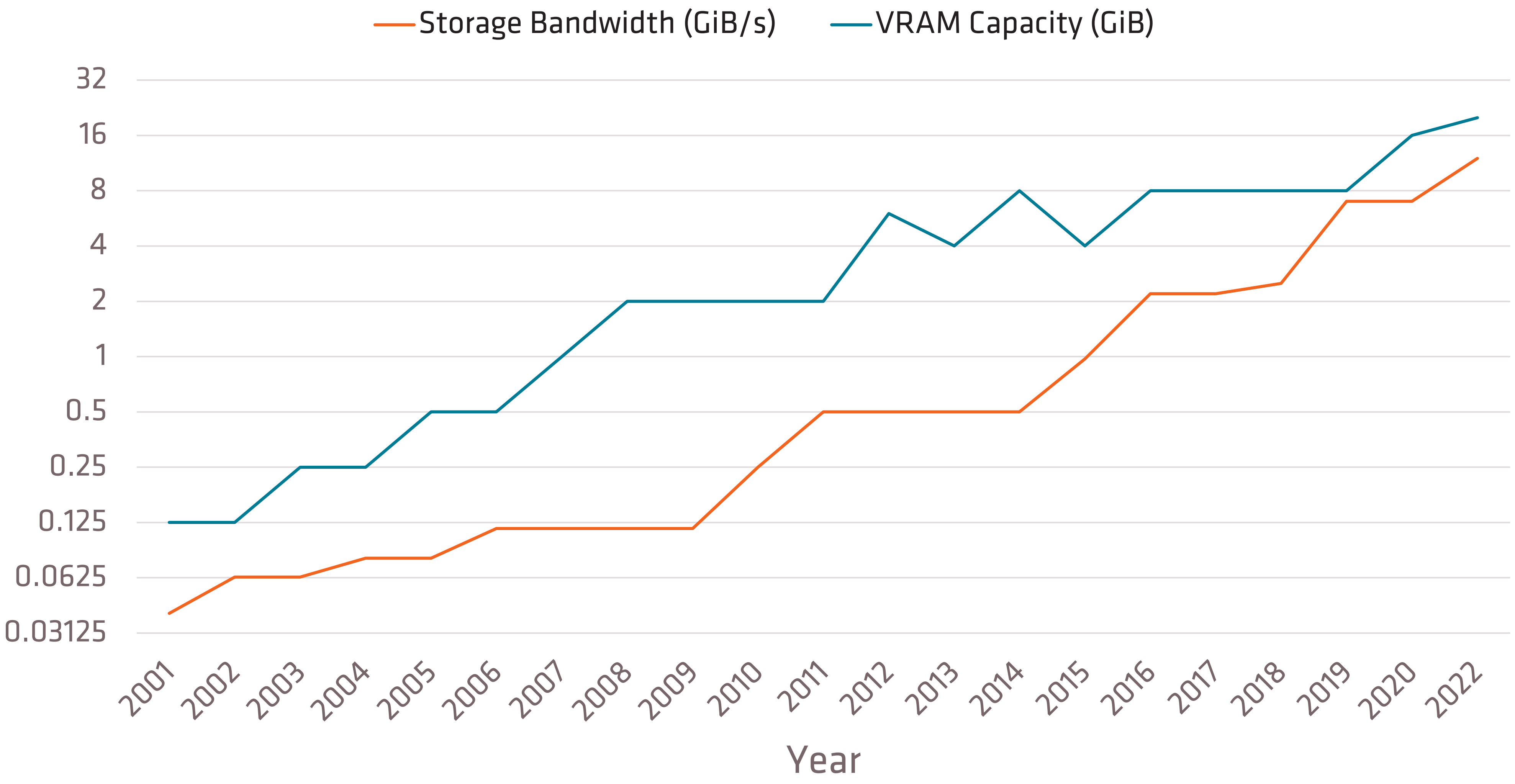
STORAGE BANDWIDTH - USUAL LIMITING FACTOR (HIGHER IS BETTER)



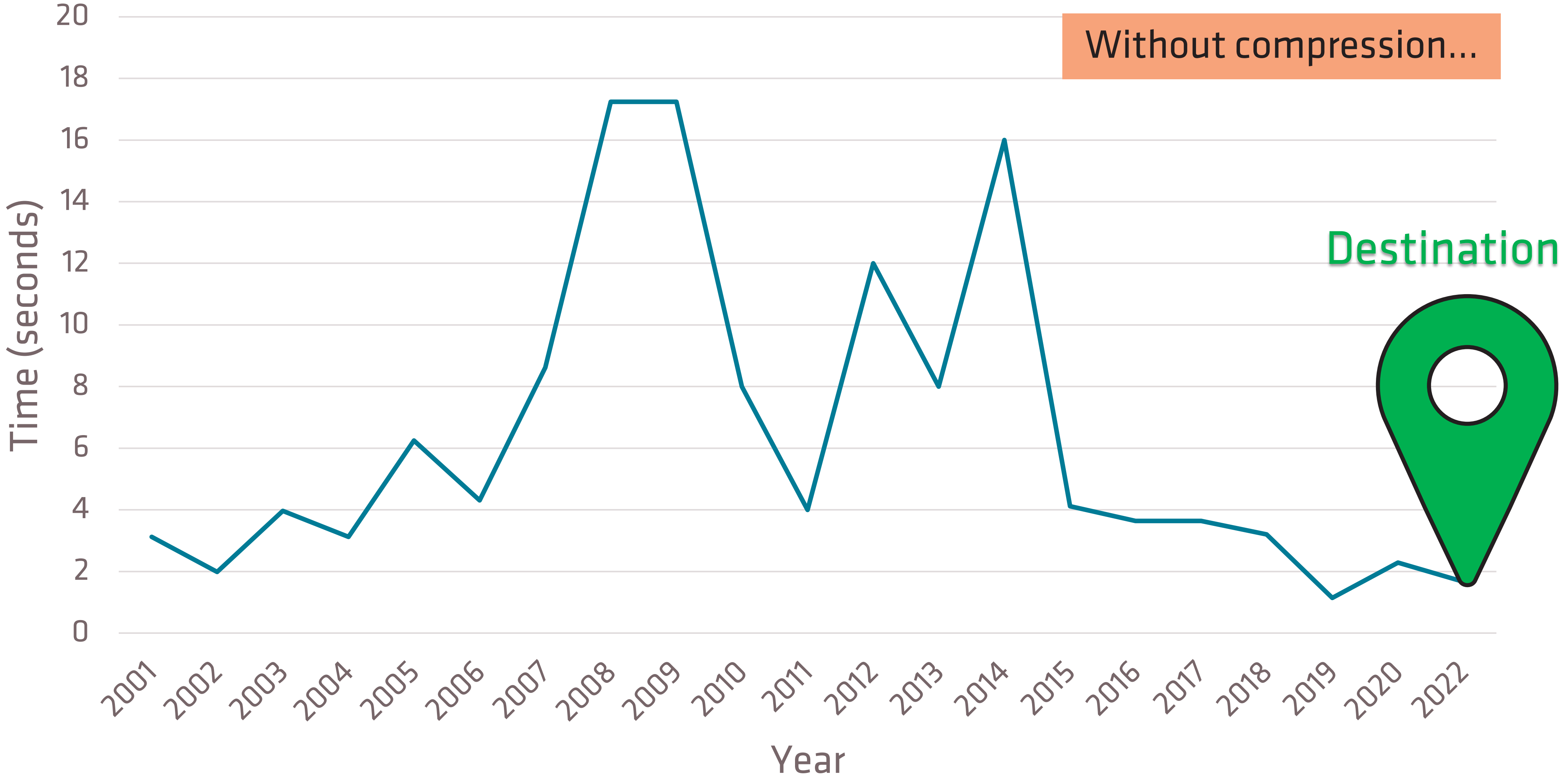
STORAGE BANDWIDTH AND VRAM CAPACITY (LINEAR SCALE)



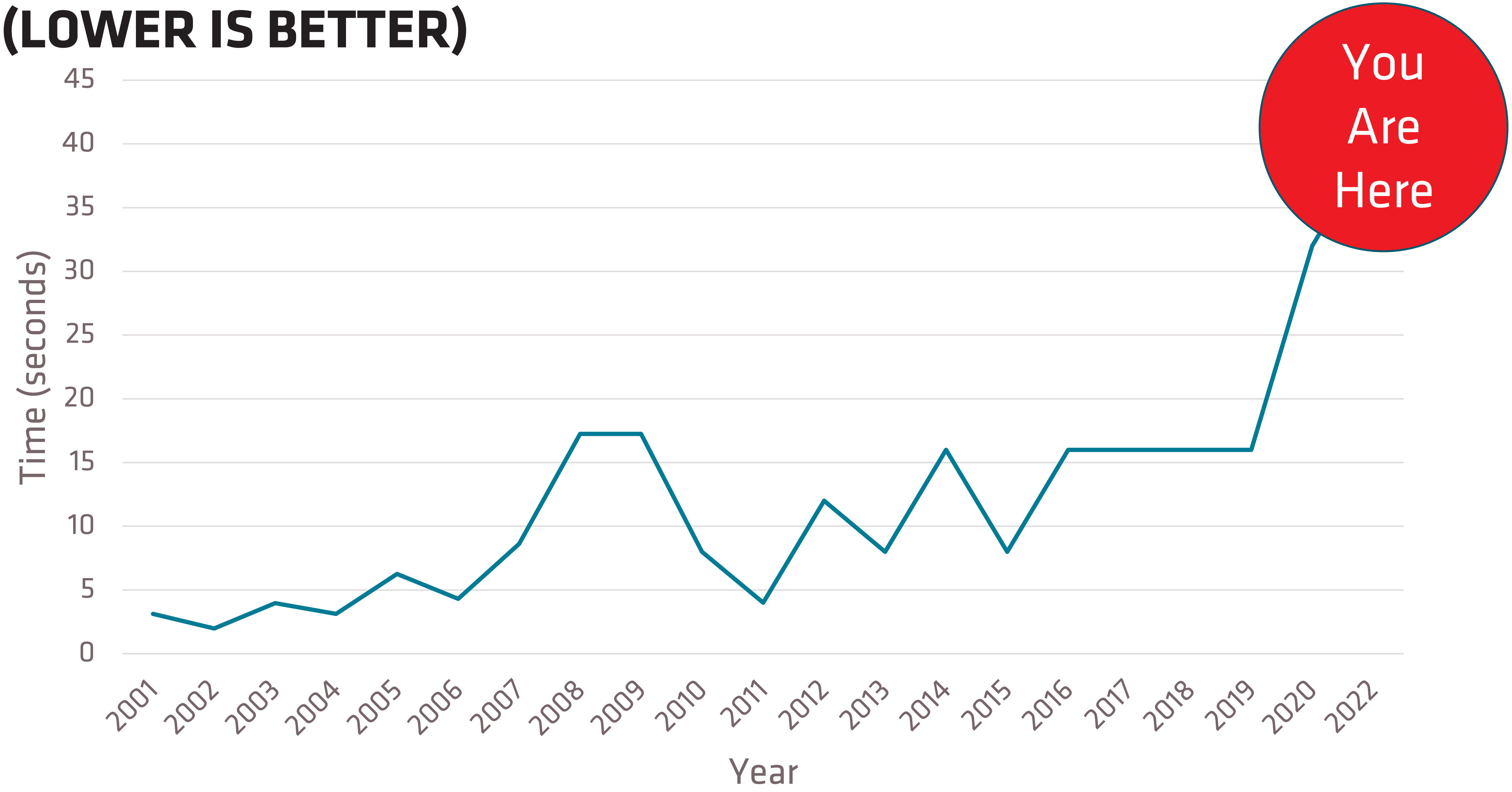
STORAGE BANDWIDTH AND VRAM CAPACITY (LOG2 SCALE)



BEST POSSIBLE LOAD TIME WITHOUT COMPRESSED DATA (LOWER IS BETTER)



BEST POSSIBLE LOAD TIME AT 500MIB/S (LOWER IS BETTER)



HOW DO WE GET THERE?

Batch

Encourage Decoupling Metadata from Resource Data

Encourage Coupling Resources to Storage Requests

Improve Dataflow

Compression

Prioritization

WHY BATCH?

- Lower CPU overhead with fewer context switches and system calls
- Keep the device busy…
- Schedule workloads for high throughput and parallelism with respect to priority

SYSTEM CONFIGURATION FOR FOLLOWING CHARTS

AMD Ryzen™ 9 3900X

MSI X570 GAMING PRO CARBON AC Motherboard

32GB (2 x 16GB DDR4-3200 at 22-22-22-52) memory

DeepCool Captain 240X Chiller

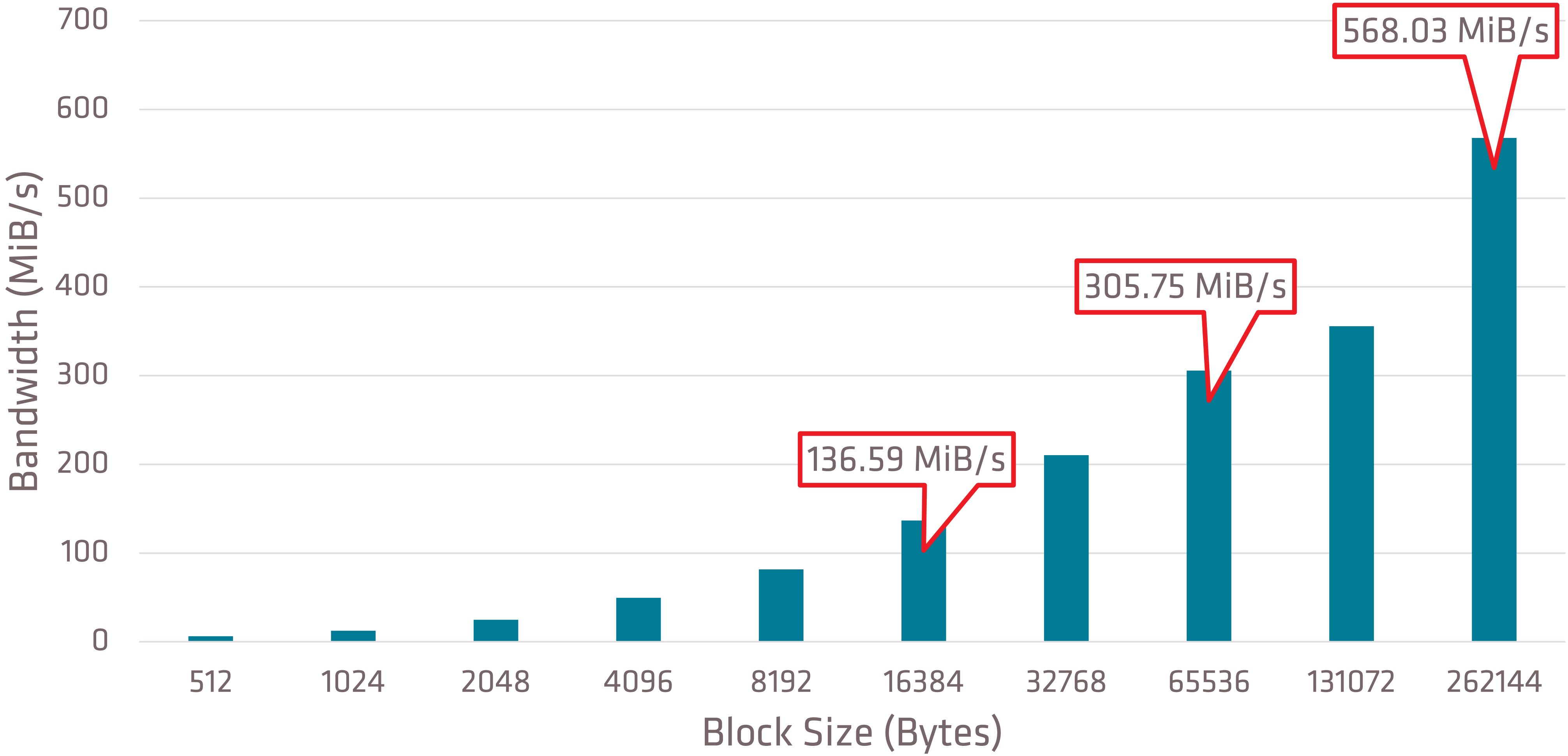
Viper VP4100 2TB M.2 NVME™ SSD with additional air cooling

Lubuntu Linux® 21.10, FIO (Flexible I/O Tester rev. 3.28)

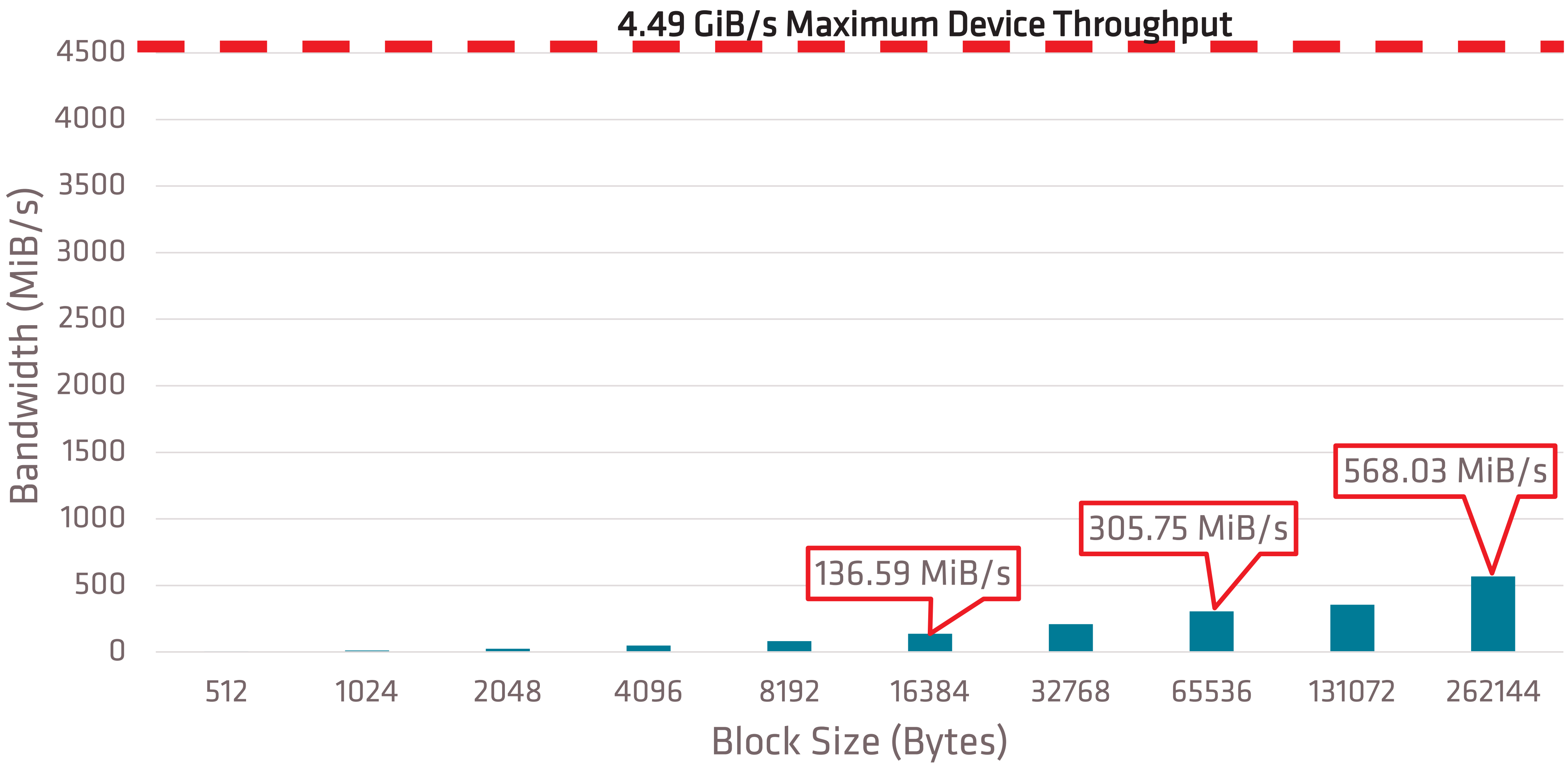
Actual results may vary.

Maximum measured bandwidth was 4.49 GiB/s!

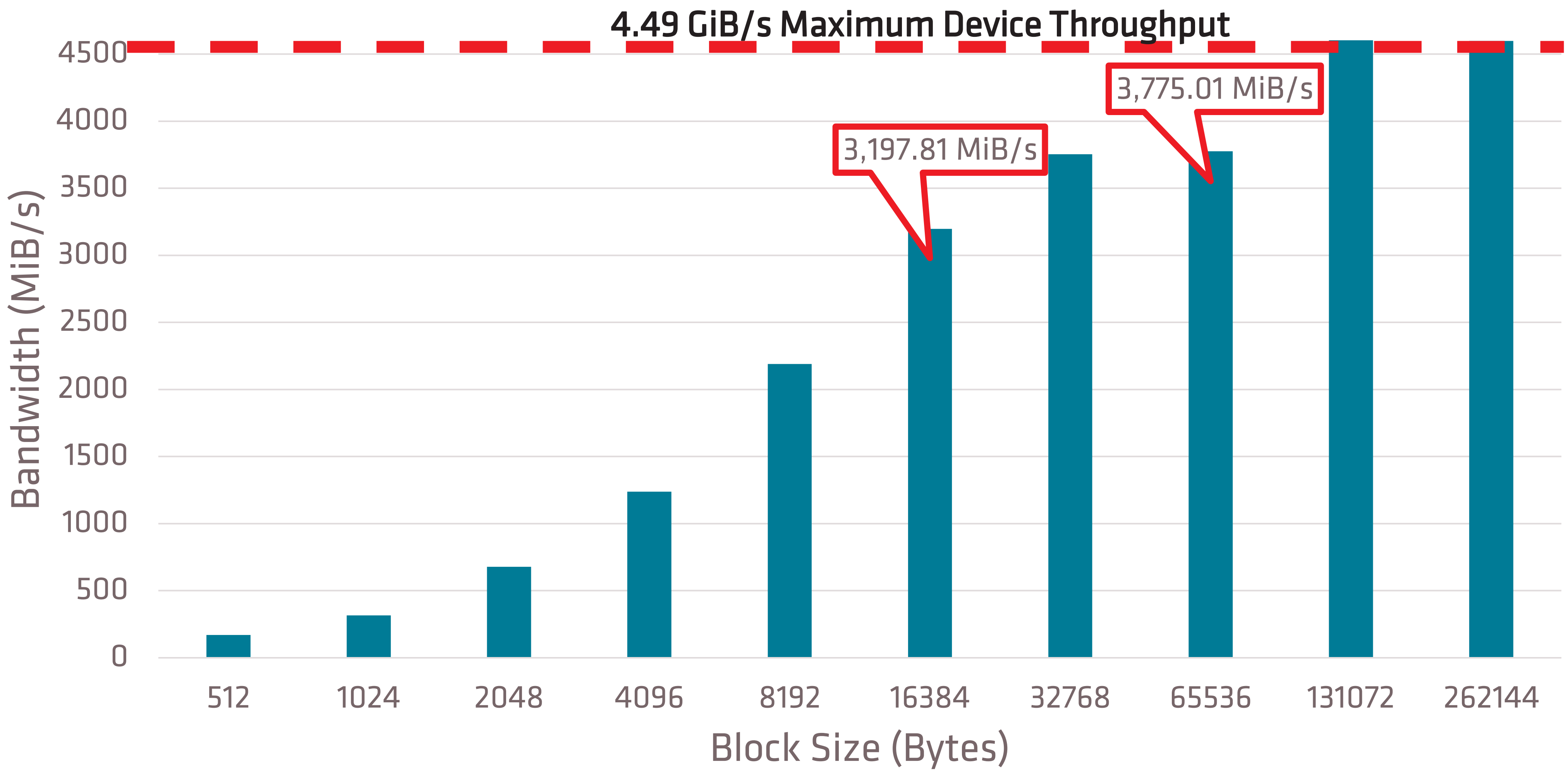
PERFORMANCE WHEN QUEUE DEPTH IS 1 (NO BATCHING)



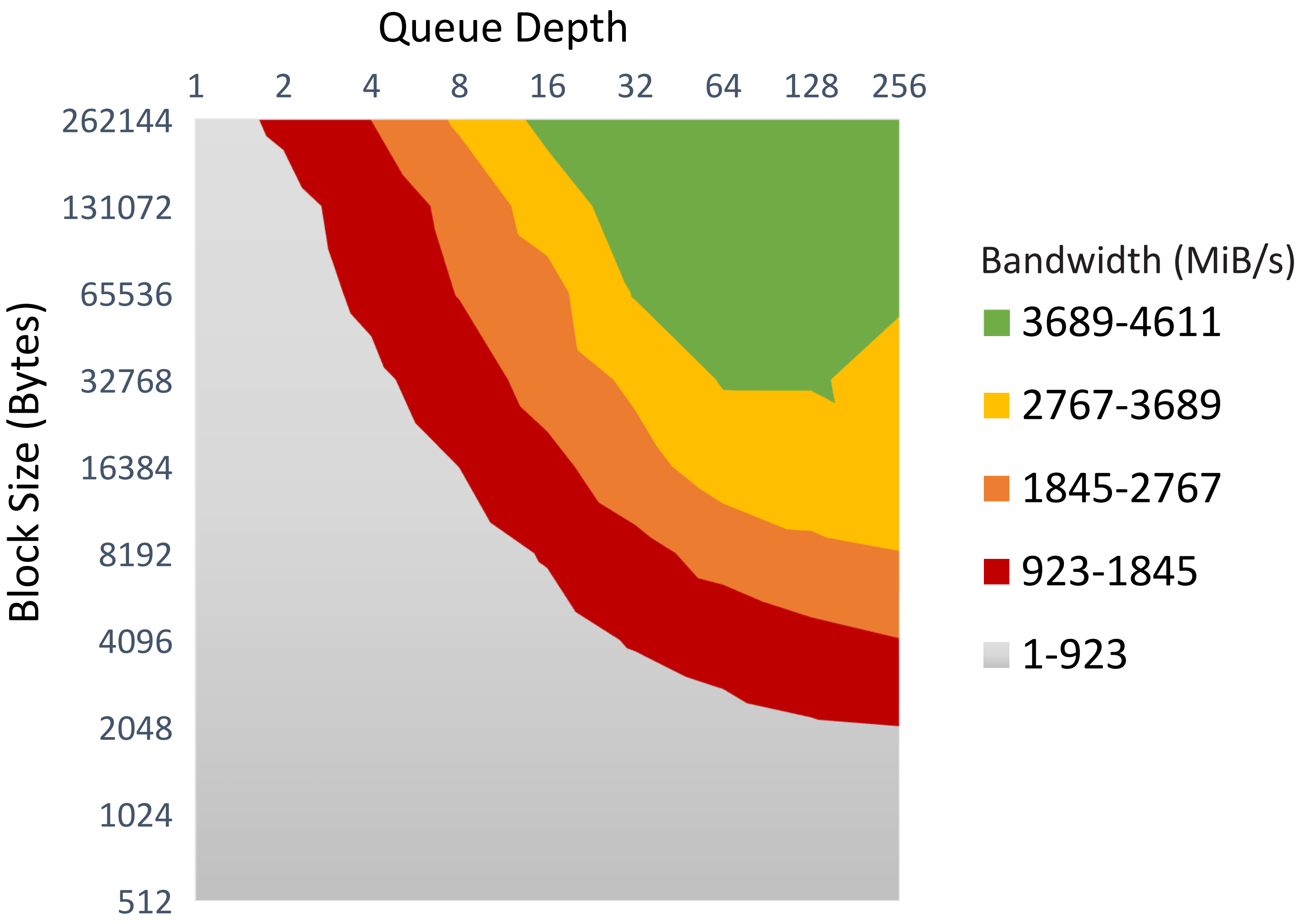
PERFORMANCE WHEN QUEUE DEPTH IS 1 (NO BATCHING)



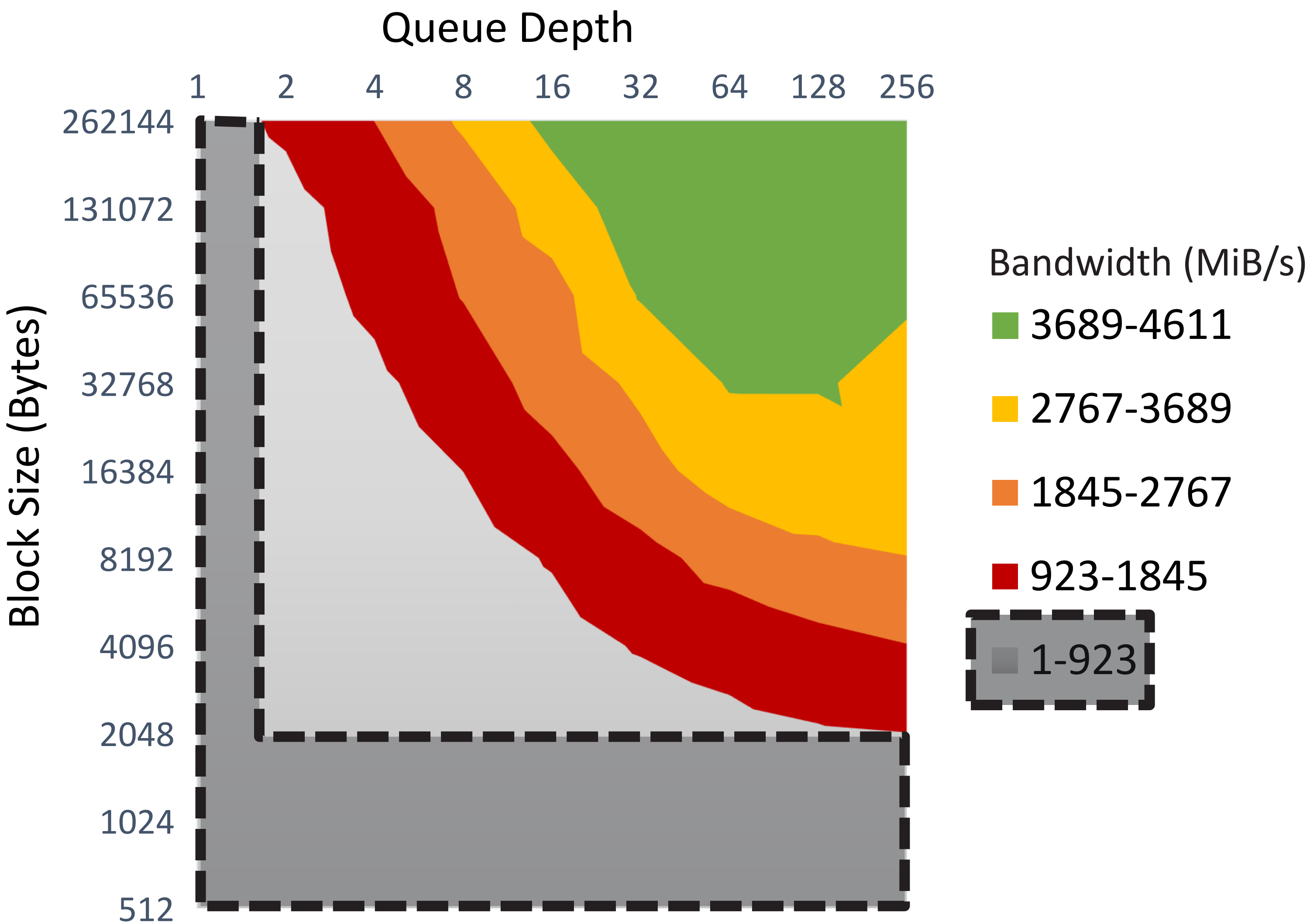
PERFORMANCE WHEN QUEUE DEPTH IS 64 (BATCHING)



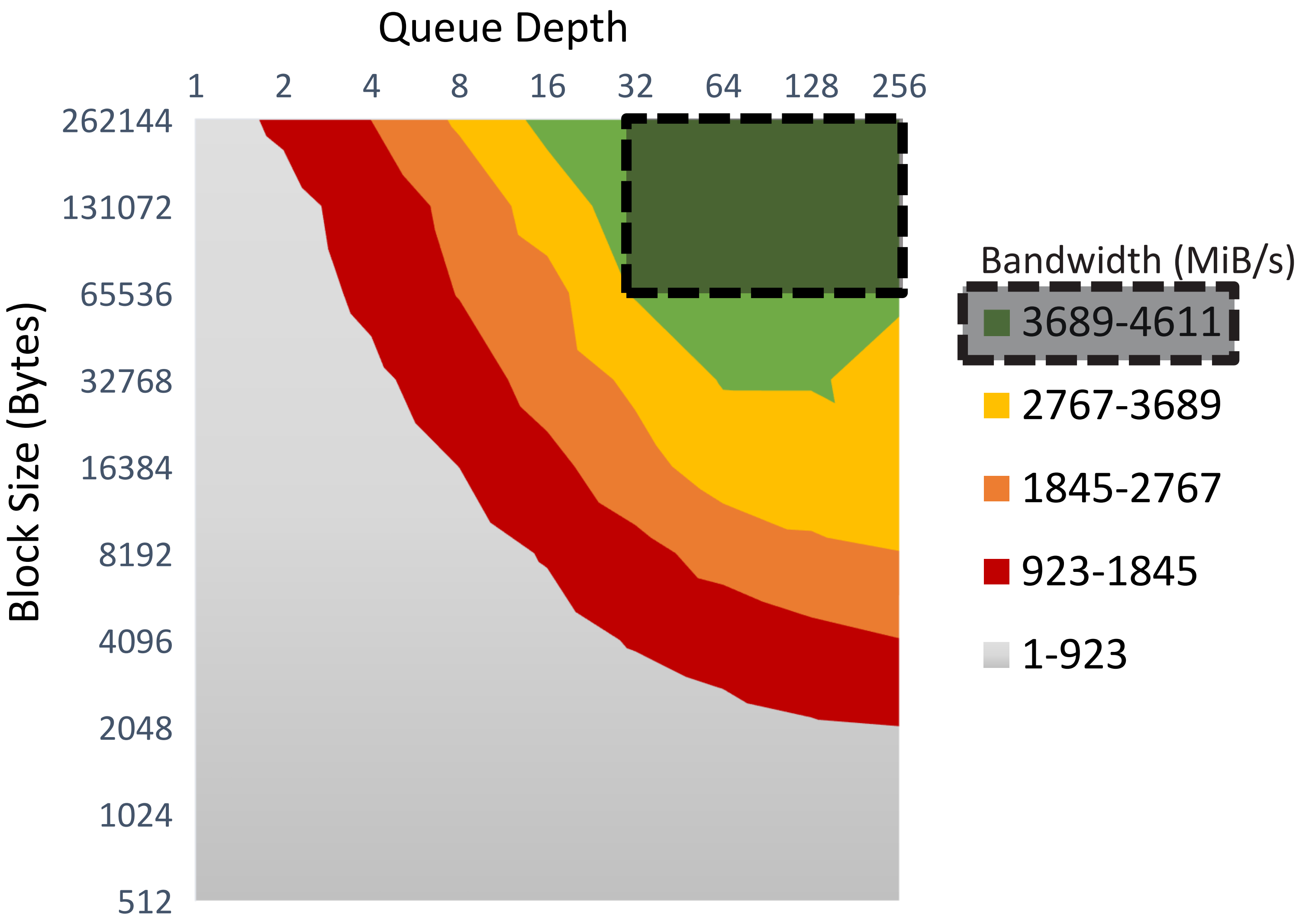
BANDWIDTH HEATMAP



BANDWIDTH HEATMAP – ~20% PERFORMANCE OR LESS



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

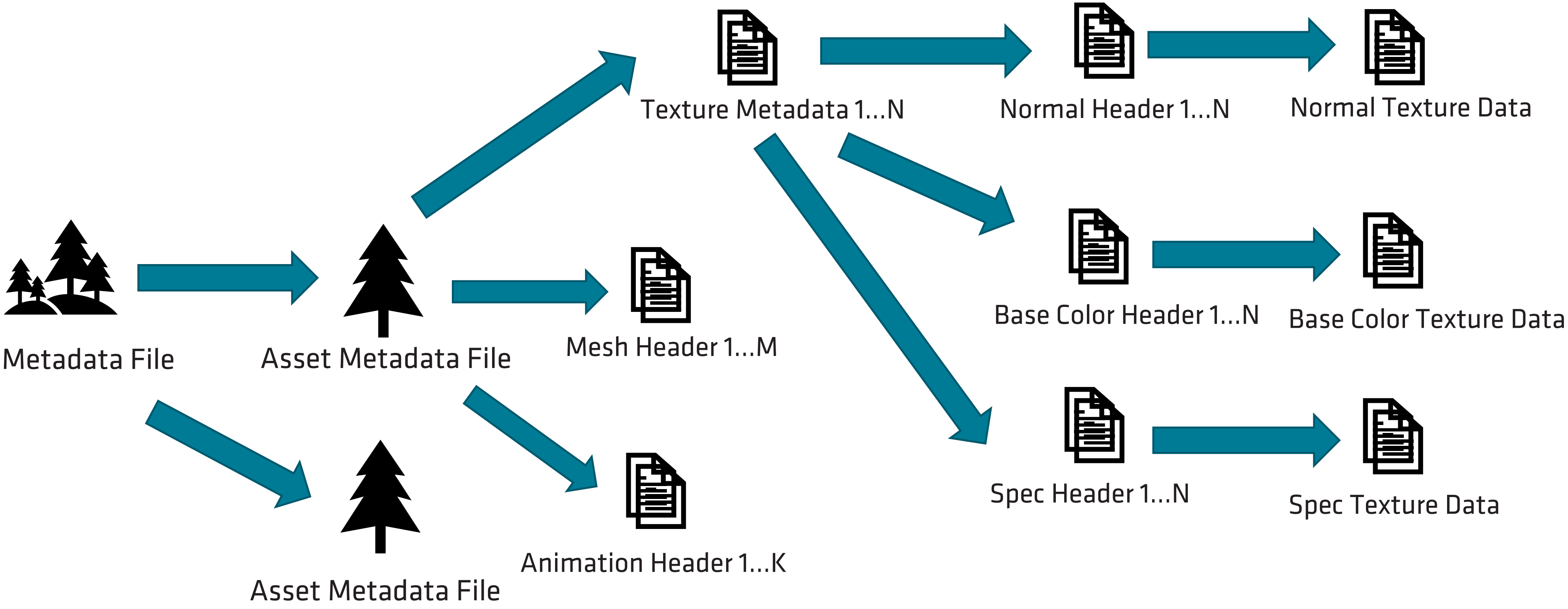
BEFORE WE CAN BATCH… WE MUST DECOUPLE DATA DEPENDENCIES

- 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…

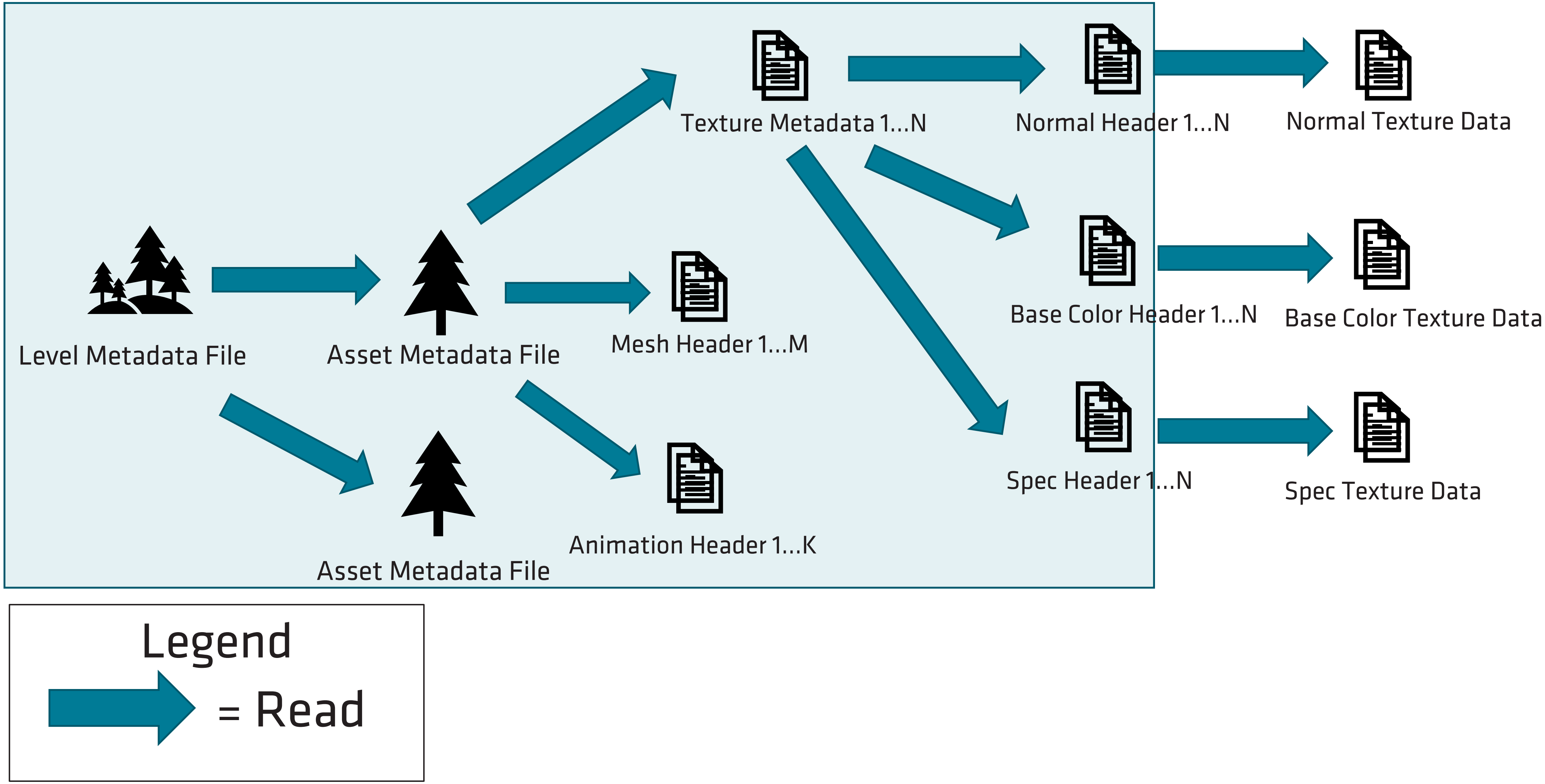
BUT GAME ENGINES ARE MORE COMPLEX...



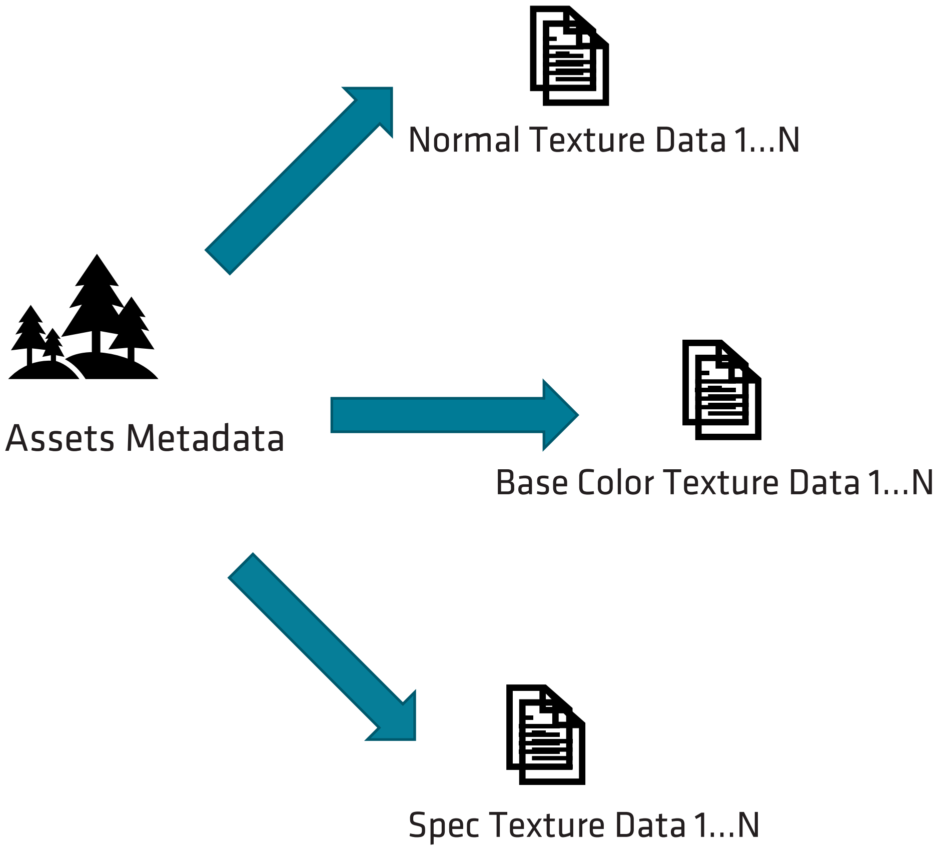
Legend

➡ = Read

CONSIDER KEEPING THIS IN RAM IN A COLLAPSED FORMAT



NOW ALL METADATA TO INITIATE READS IS LOADED IN ONE PLACE

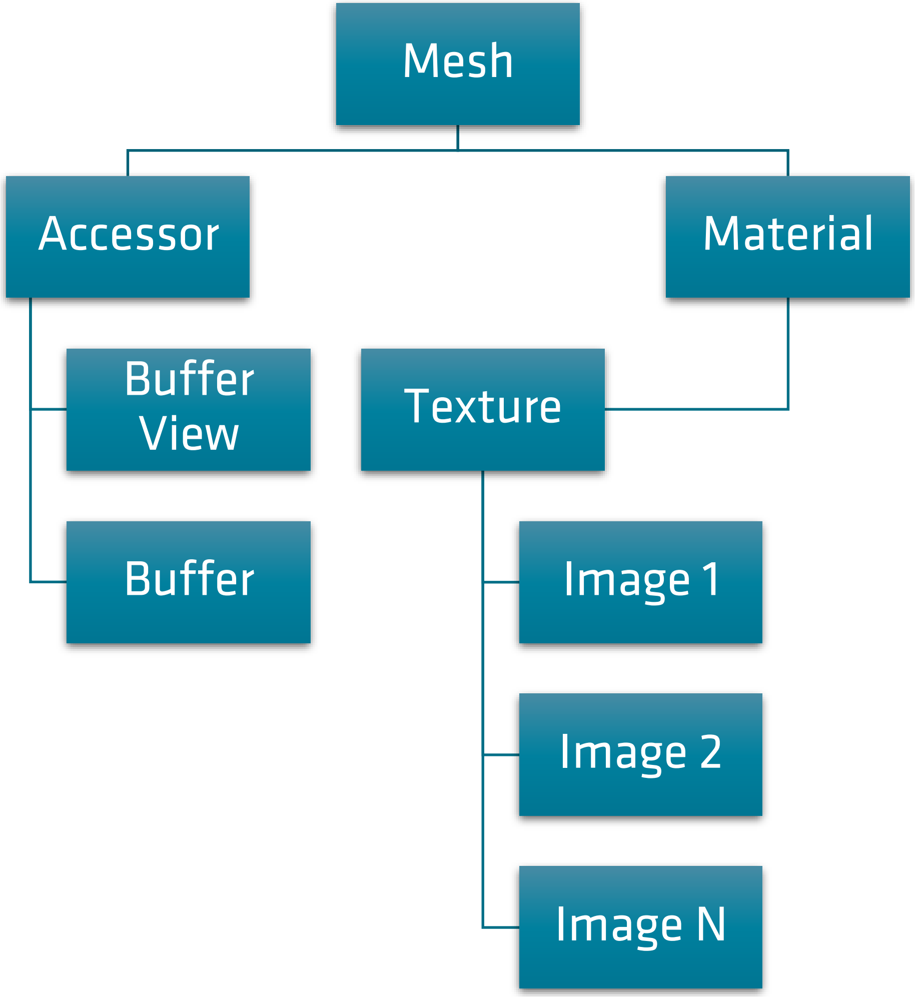

Normal Texture Data 1...N
Level AND Assets Metadata
Base Color Texture Data 1...N
Spec Texture Data 1...N

Legend

➡ = Read

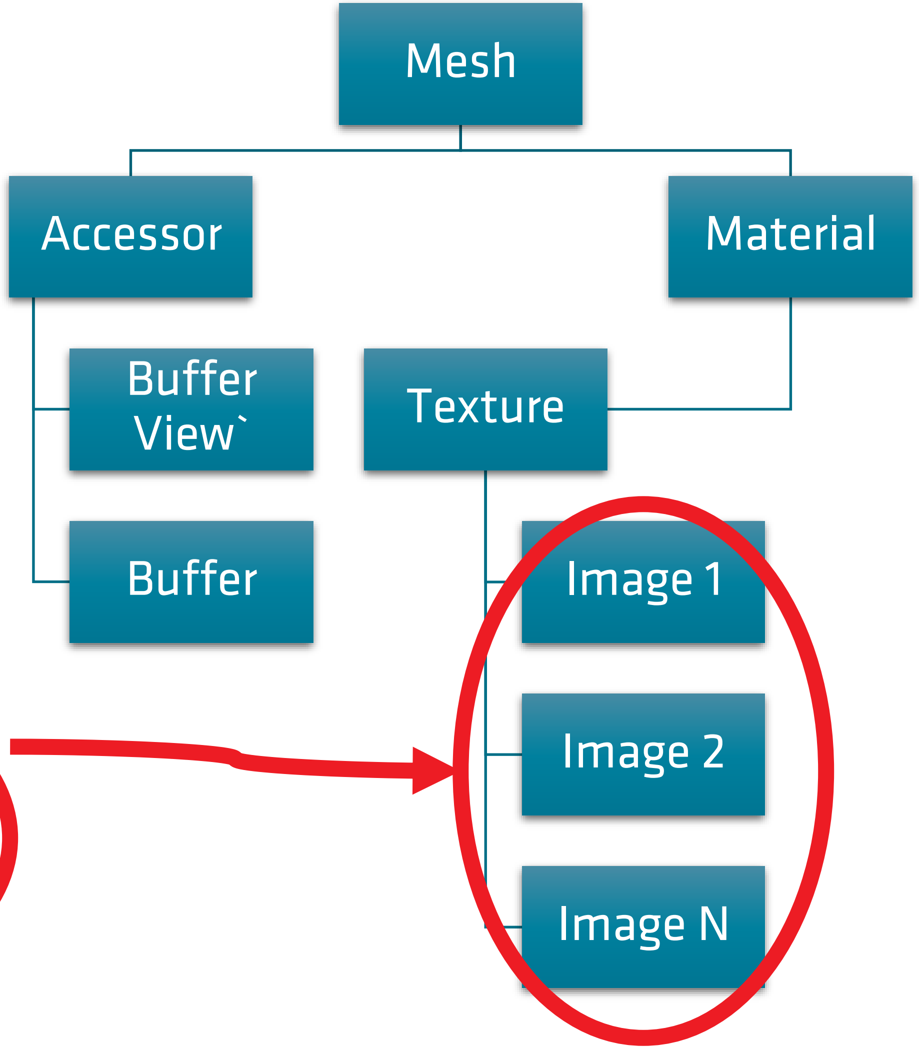
GLTF™ EXAMPLE

- glTF™ is an API neutral 3D asset transmission format.
  - JSON

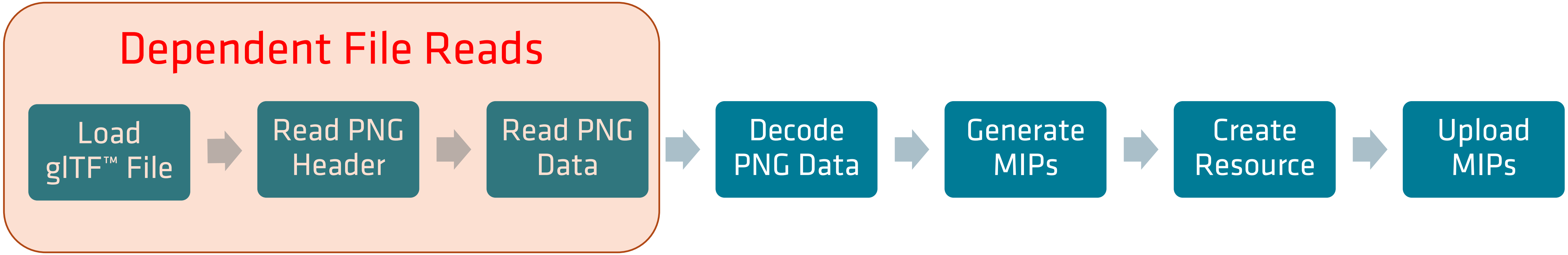


GLTF™ EXAMPLE

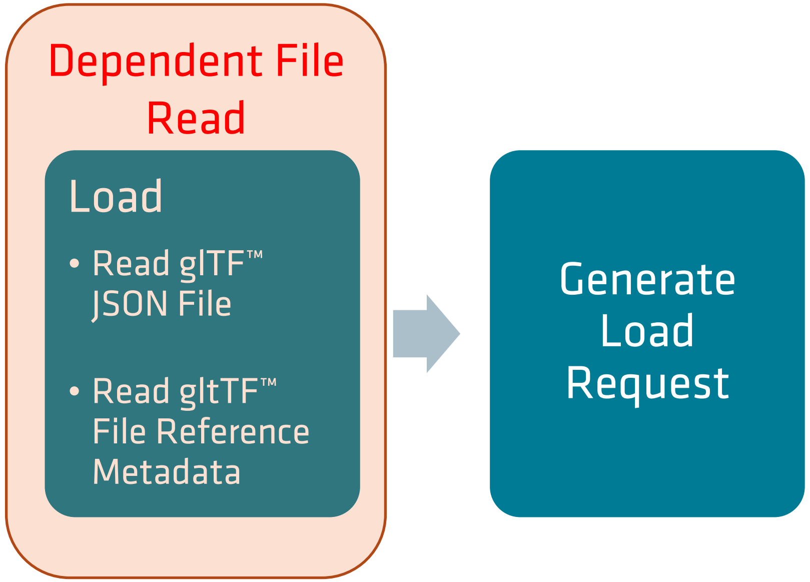
- glTF™ is an API neutral 3D asset transmission format.
  - JSON
  - Images reference separate image files in PNG or JPG format as Uniform Resource Identifier (similar to URL).



CURRENT GLTF™ LOADING PIPELINE



TARGET GLTF™ LOADING PIPELINE



Dependent File Read

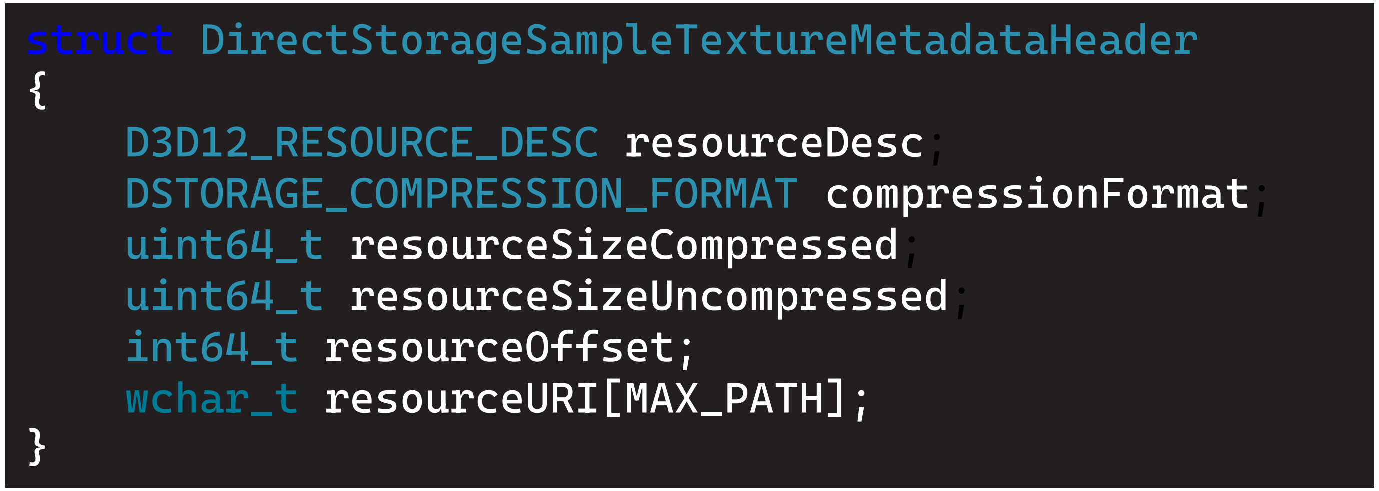
Load

- Read glTF™ JSON File
- Read gltTF™ File Reference Metadata

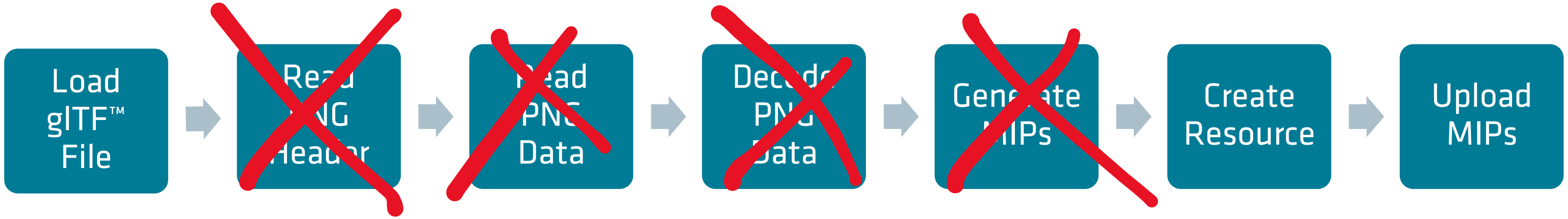
Generate Load Request

HOW DO WE GET THERE?

- Preprocess
  1. Load PNG
  2. Create Mip Chain Off-line and store it (More on this in a moment).
  3. Fill in Metadata structure to load when the JSON file loads or at startup.



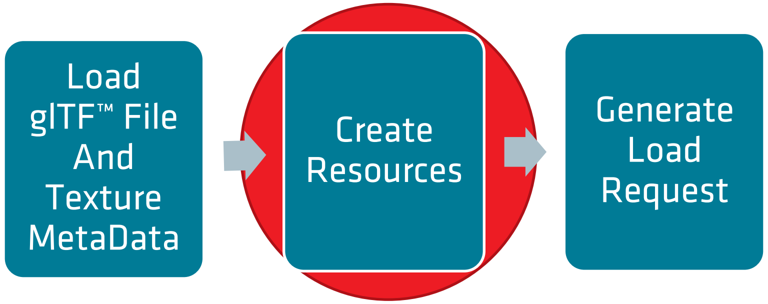
```
struct DirectStorageSampleTextureMetadataHeader
{
    D3D12_RESOURCE_DESC resourceDesc;
    DSTORAGE_COMPRESSION_FORMAT compressionFormat;
    uint64_t resourceSizeCompressed;
    uint64_t resourceSizeUncompressed;
    int64_t resourceOffset;
    wchar_t resourceURI[MAX_PATH];
}
```



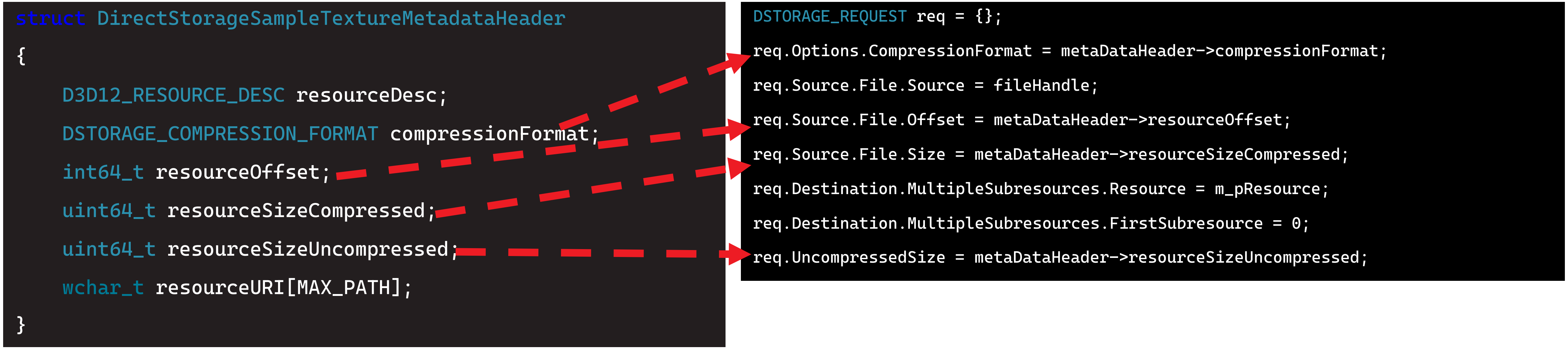
WHAT DATA IS REQUIRED TO FULFILL A REQUEST?

```
struct DirectStorageSampleTextureMetadataHeader
{
    D3D12_RESOURCE_DESC resourceDesc;
    DSTORAGE_COMPRESSION_FORMAT compressionFormat;
    int64_t resourceOffset;
    uint64_t resourceSizeCompressed;
    uint64_t resourceSizeUncompressed;
    wchar_t resourceURI[MAX_PATH];
}
```

```
HRESULT hr = pDevice->GetDevice()->CreatePlacedResource(pTextureHeap
                                    , resourceEntry.resourceHeapOffset
                                    , &metaDataHeader->resourceDesc
                                    , D3D12_RESOURCE_STATE_COMMON
                                    , nullptr
                                    , IID_PPV_ARGS(&m_pResource));
```

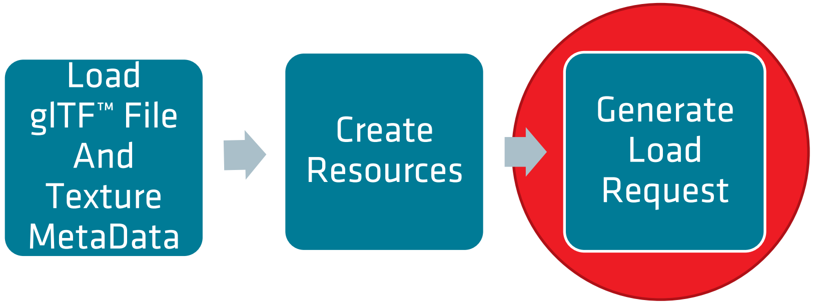


WHAT DATA IS REQUIRED TO FULFILL A REQUEST?

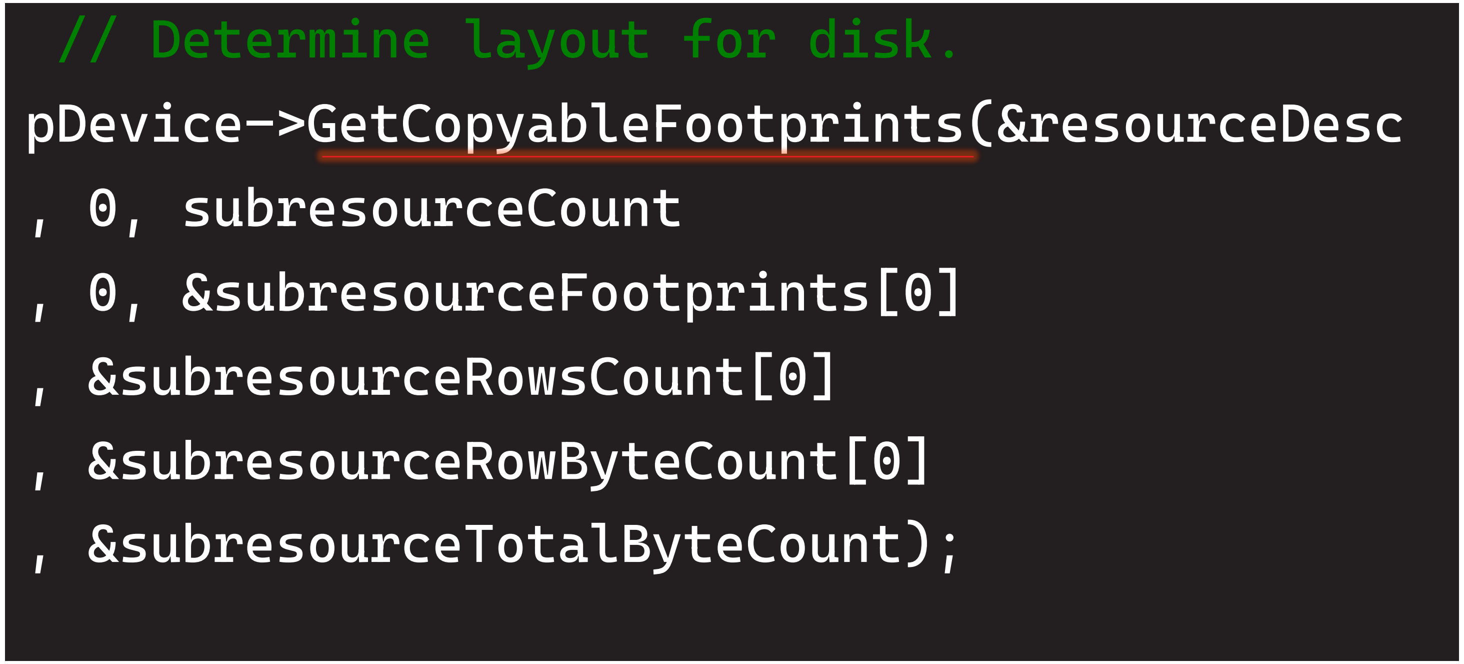


```
struct DirectStorageSampleTextureMetadataHeader
{
    D3D12_RESOURCE_DESC resourceDesc;
    DSTORAGE_COMPRESSION_FORMAT compressionFormat;
    int64_t resourceOffset;
    uint64_t resourceSizeCompressed;
    uint64_t resourceSizeUncompressed;
    wchar_t resourceURI[MAX_PATH];
}
```

```
DSTORAGE_REQUEST req = {};
req.Options.CompressionFormat = metaDataHeader->compressionFormat;
req.Source.File.Source = fileHandle;
req.Source.File.Offset = metaDataHeader->resourceOffset;
req.Source.File.Size = metaDataHeader->resourceSizeCompressed;
req.Destination.MultipleSubresources.Resource = m_pResource;
req.Destination.MultipleSubresources.FirstSubresource = 0;
req.UncompressedSize = metaDataHeader->resourceSizeUncompressed;
```



BUT WHAT ABOUT THE DATA?



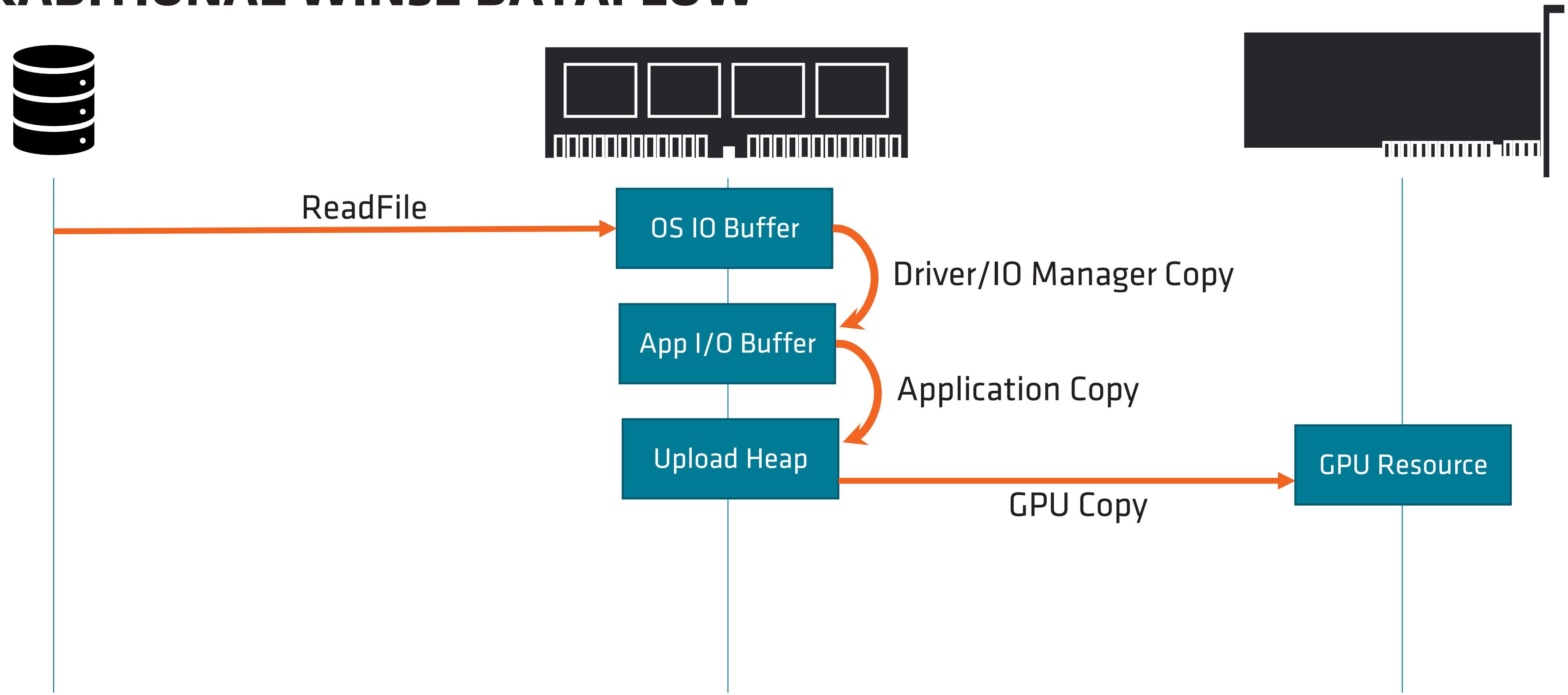
```
 // Determine layout for disk.
pDevice->GetCopyableFootprints(&resourceDesc
, 0, subresourceCount
, 0, &subresourceFootprints[0]
, &subresourceRowsCount[0]
, &subresourceRowByteCount[0]
, &subresourceTotalByteCount);
```

- Save the data out in the format specified by the values returned by this function
- Device independent format
- Compress afterwards if necessary
- Tips on compression later
- The offset in the file is located in the metadata structure!

The texture data is now prepared for DirectStorage consumption.

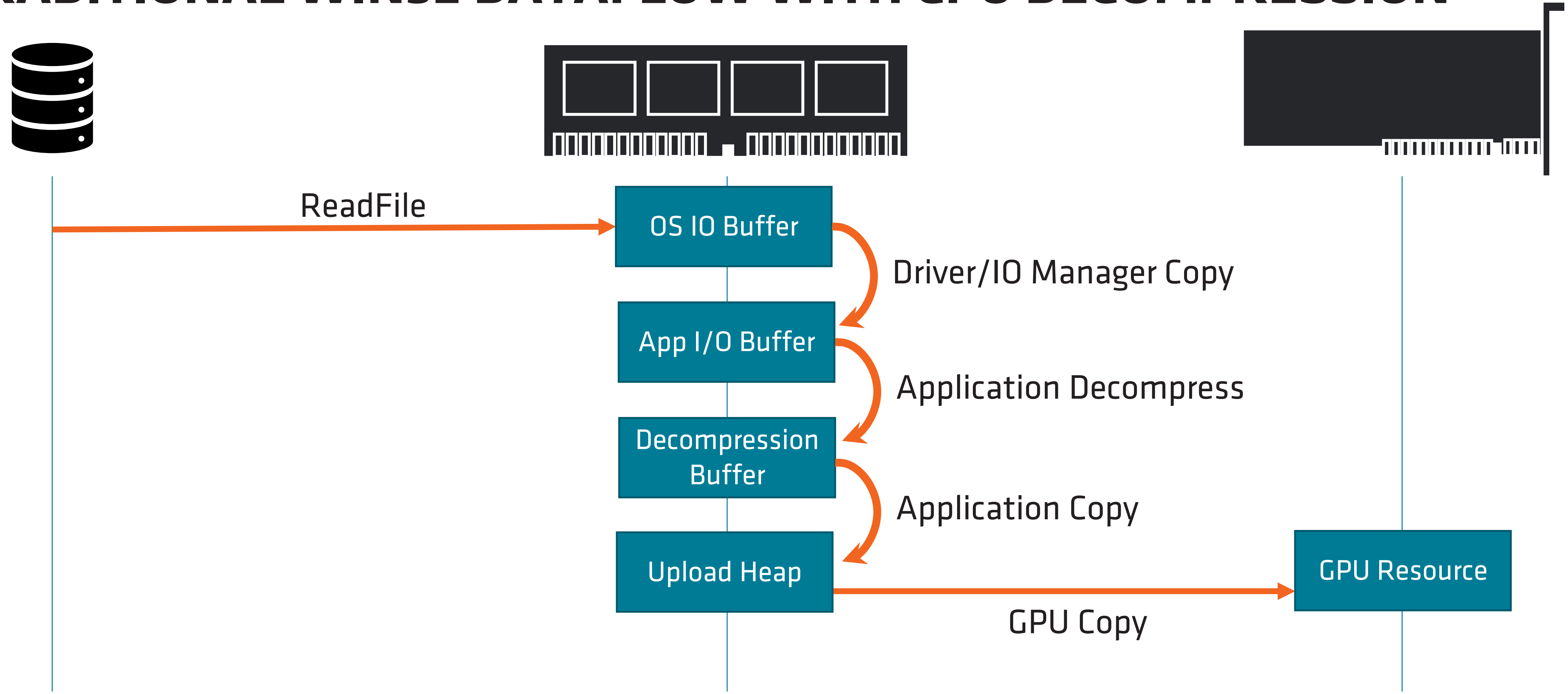
IMPROVED DATAFLOW

TRADITIONAL WIN32 DATAFLOW



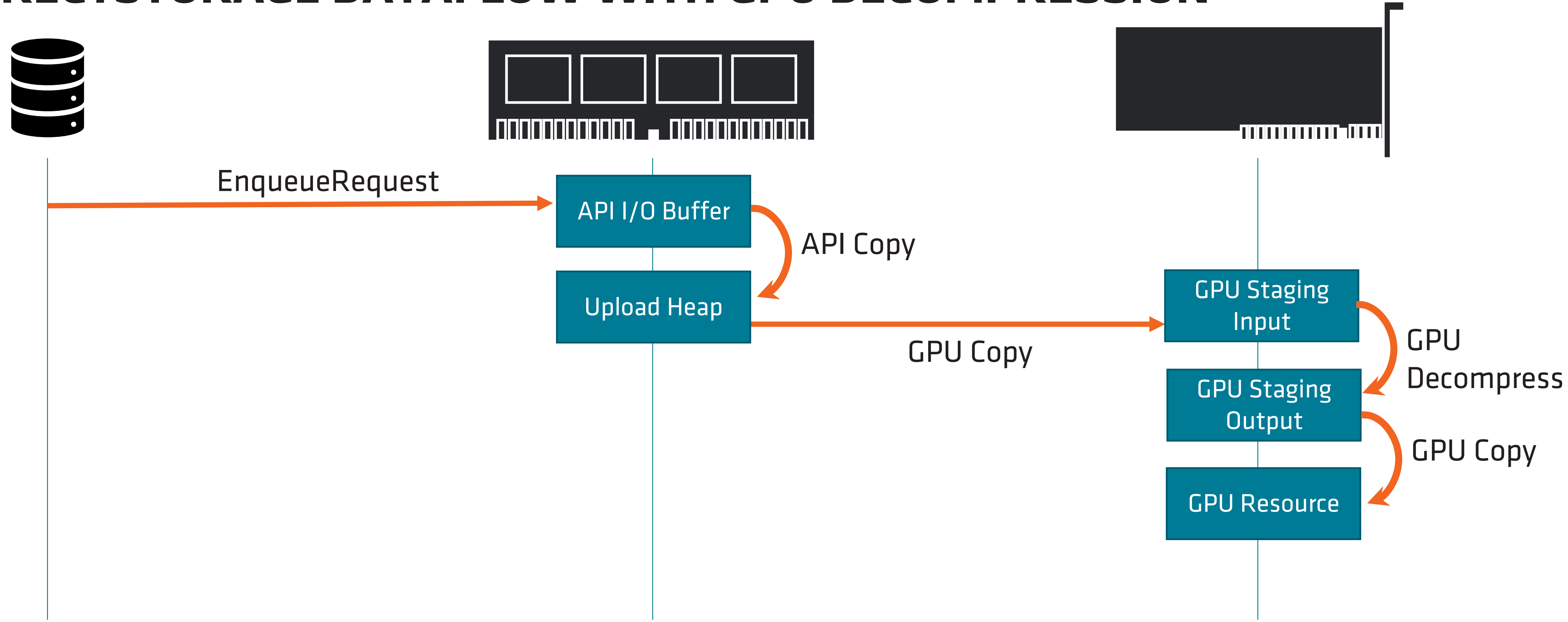
Up to 4x Copies

TRADITIONAL WIN32 DATAFLOW WITH CPU DECOMPRESSION



Up to 5x Copies

DIRECTSTORAGE DATAFLOW WITH GPU DECOMPRESSION



Up to 5x Copies

WHAT OTHER DATAFLOW IMPROVEMENTS HAVE BEEN MADE?

- DirectStorage automatically uses the Copy Queue.

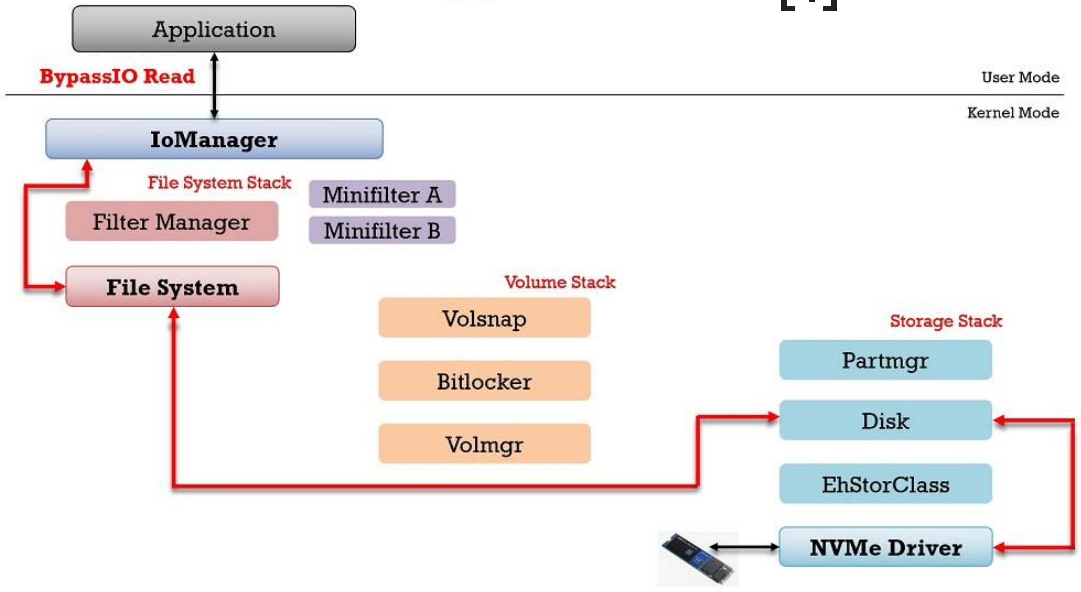
- Windows® 11's new I/O capabilities improve I/O throughput and CPU utilization, and DirectStorage takes advantage of these transparently.

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

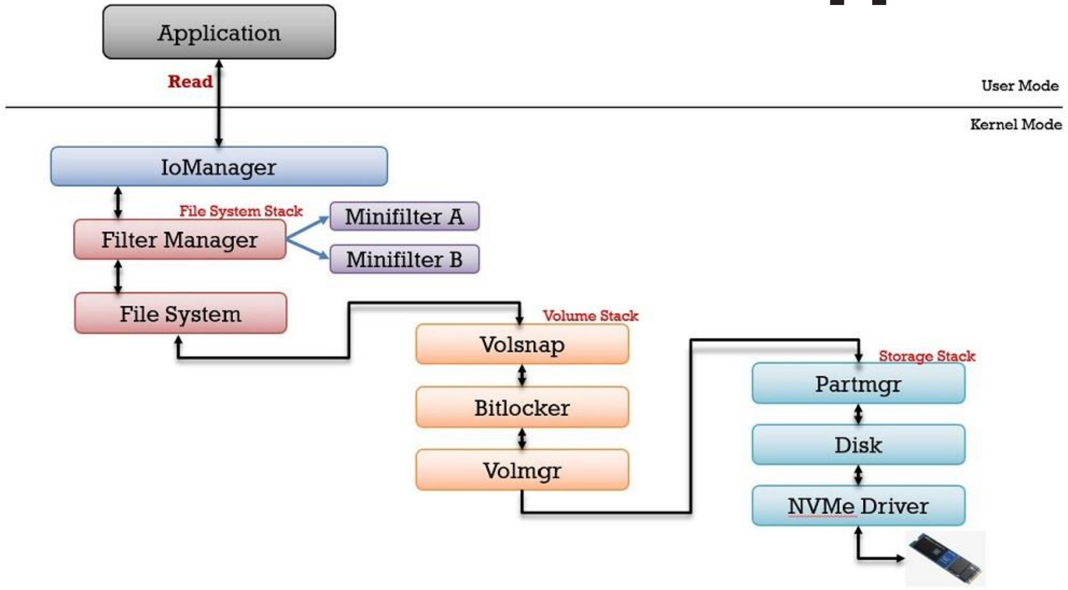
  - IORing File API
    - Batches I/O requests and completion notifications to reduce the CPU overhead of associated context switching

BYPASSIO

BypassIO Model [1]



Traditional IO Model [1]



BYPASSIO - CERTAIN CONDITIONS APPLY…

- Currently only works with NVMe™ devices
- Gets disabled under the following conditions…
  - Reading from a BitLocker encrypted volume will not bypass the Volume Stack
  - NTFS-encrypted files
  - NTFS-compressed files
  - Sparse files
  - During volume snapshots
  - Any filter driver can disable bypass I/O (they can veto it). Filter drivers must explicitly support it for it to work

- These conditions may impact profiling and testing!
  - Check to see if BypassIO is working on your system… Could change at run-time too
  - fsutil bypassIo state <path>

CANCELLATION

- This might be an unsung hero for low-end platforms and/or fast-moving titles with texture streaming.

- Saves bandwidth!

- Increases responsiveness!

- Consider when a player moves the camera quickly…
  - Texture requests for high level MIPs become irrelevant quickly
  - Overdue or irrelevant requests can now be cancelled, and new more relevant requests can progress sooner
  - It works. I’ve tried it, and you can too in our demo

COMPRESSION AND DECOMPRESSION

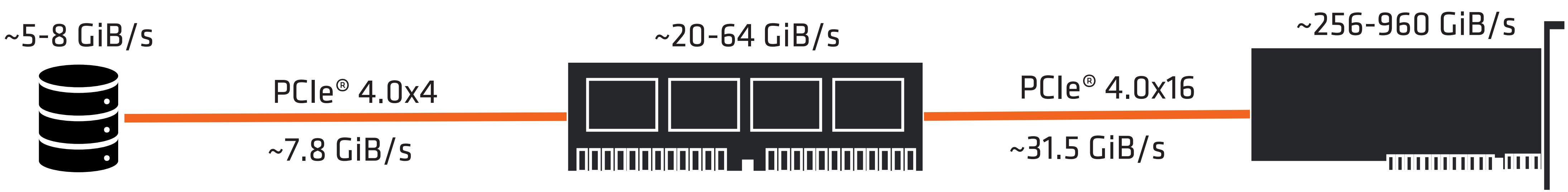
WHY COMPRESSION – THE OBVIOUS REASONS

- Less data to transfer across all interfaces
  - Save network transfer time and cost
  - Better utilization of PCIe® bandwidth

- Save disk space on end-user systems

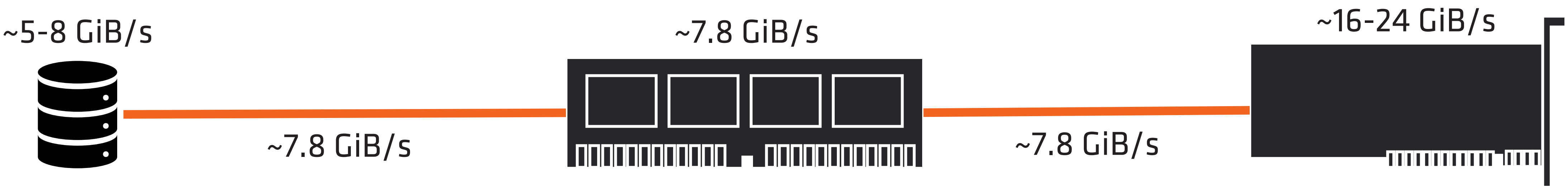
- Data obfuscation… NOT encryption… NOT a guarantee that data hasn’t been tampered with

WHY MOVE DECOMPRESSION TO THE GPU?



Only as fast as the slowest component

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.**

WHAT ABOUT NON-GPU RESOURCES?

- Built-in GDeflate decompression will be redirected to CPU.

- Advisable to compress with custom compressor with better performance more suitable to the data.
    - Examples: LZ4, LZO, ZSTD, etc. Pick an algorithm that meets your needs.

- For reference, it’s possible to achieve over ~7 GiB/s per core on modern CPUs using LZ4 with low compression settings.

HOW DOES IT IMPACT PERFORMANCE

- It depends…

- Factors include:
  - Storage device bandwidth
  - Staging buffer size
  - GPU performance

- We will explore this later in a discussion of our DirectStorage demo.

WHY USE DIRECTSTORAGE?

- Implementing and maintaining similar functionality in an engine is complex and requires maintenance
  - I/O API
  - Upload Heaps
  - Copy Queues and related resource upload functions
  - Optimal implementation regardless of Windows® version
- Encourages good file I/O optimization practices
- Reduces CPU load
- Similar functionality available on console

OUR DIRECTSTORAGE SAMPLE

- Simulation of streaming high detail assets while the camera moves through a scene with trigger volumes to determine whether to load or unload an assets

- Developed using our GPUOpen Cauldron Framework as a test bed to understand integration requirements, performance and provide an example

- By moving the camera at different rates, we control the demand on the streaming system and the hardware

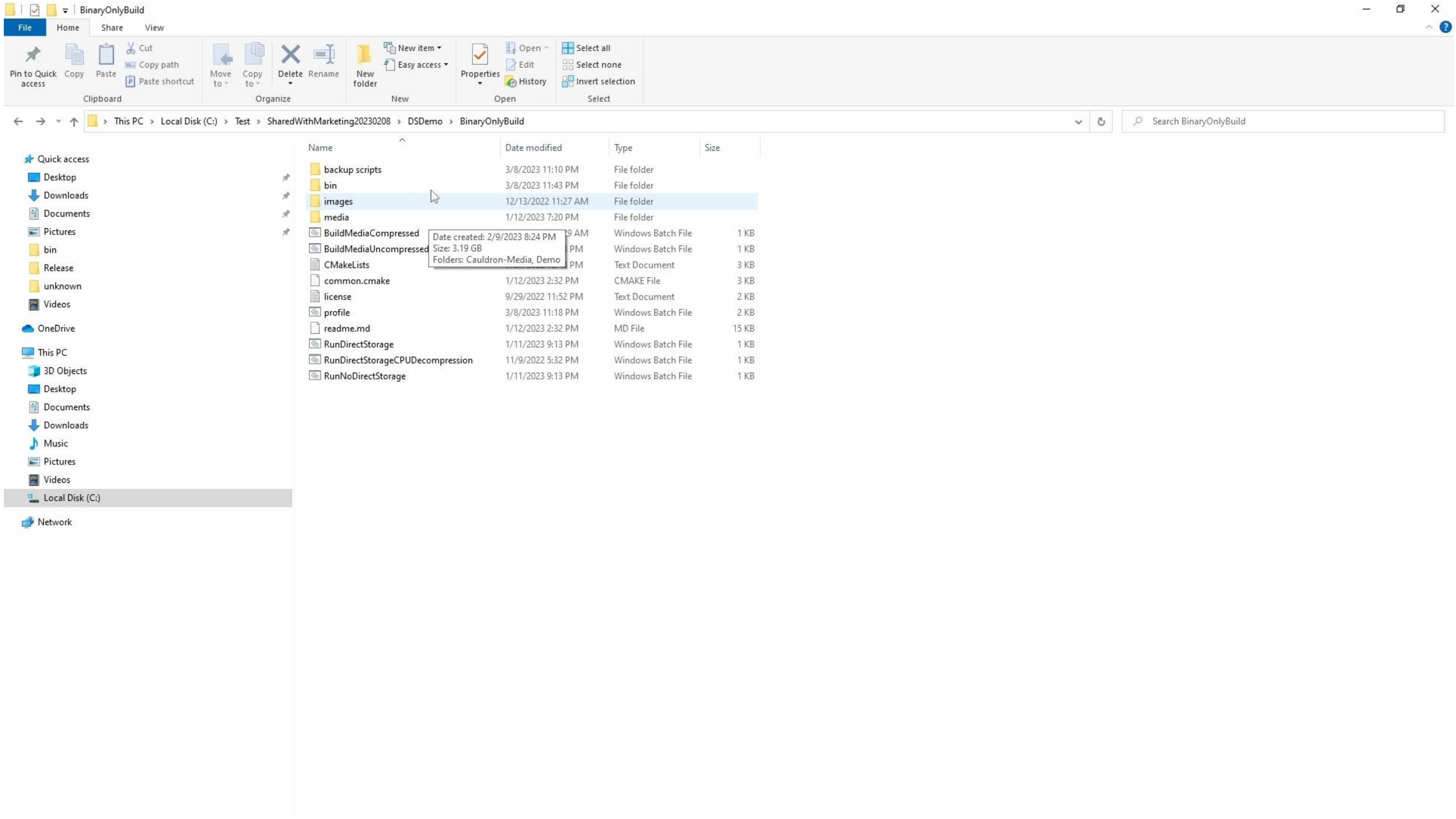
- The faster the load time the sooner an asset appears

SYSTEM CONFIGURATION FOR FOLLOWING CHARTS AND VIDEO

- AMD Ryzen™ 9 7900X
- AMD Radeon™ RX 7900 XTX
- Asrock X670E Steel Legend Motherboard
- 32GB (2x 16GB DDR5-6000 at 30-38-38-96) memory
- 500GB PCIe® 5 NVMe™ SSD with additional air cooling
- Windows® 10 Pro 64-bit (10.0, Build 19045)

- Actual results may vary

VIDEO OF OUR DEMO



This PC › Local Disk (C:) › Test › SharedWithMarketing20230208 › DSDemo › BinaryOnlyBuild

Name	Date modified	Type	Size
backup scripts	3/8/2023 11:10 PM	File folder	
bin	3/8/2023 11:43 PM	File folder	
images	12/13/2022 11:27 AM	File folder	
media	1/12/2023 7:20 PM	File folder	
BuildMediaCompressed	9 AM	Windows Batch File	1 KB
BuildMediaUncompressed	PM	Windows Batch File	1 KB
CMakeLists	PM	Text Document	3 KB
common.cmake	1/12/2023 2:32 PM	CMAKE File	3 KB
license	9/29/2022 11:52 PM	Text Document	2 KB
profile	3/8/2023 11:18 PM	Windows Batch File	2 KB
readme.md	1/12/2023 2:32 PM	MD File	15 KB
RunDirectStorage	1/11/2023 9:13 PM	Windows Batch File	1 KB
RunDirectStorageCPUDecompression	11/9/2022 5:32 PM	Windows Batch File	1 KB
RunNoDirectStorage	1/11/2023 9:13 PM	Windows Batch File	1 KB

Date created: 2/9/2023 8:24 PM
Size: 3.19 GB
Folders: Cauldron-Media, Demo

🟩 Loaded

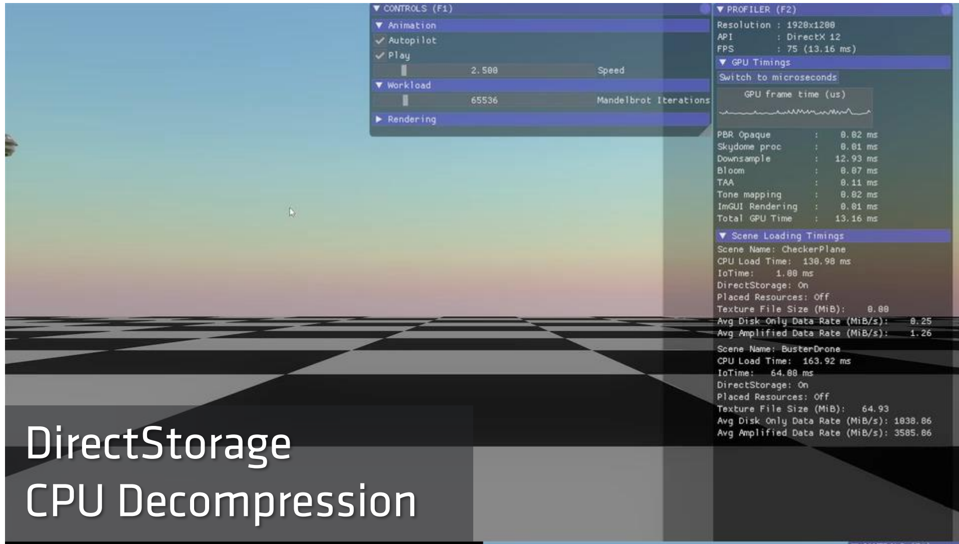
🟨 Loading

⬜ Not Loaded

SAMPLE FEATURES

- A way to pre-process glTF™ textures into a DirectStorage-compatible format with metadata
  - With Compression: Choose compression level or exhaustively try compression levels to find smallest one
  - Without compression: Disabling compression reduces GPU or workload decompressing data
  - glTF™ assets and new volumes can be added to scenes
- Adjustable staging buffer size via command-line parameters
- Adjustable artificial workload to study and understand how decompression interacts with other rendering and compute workloads
- PIX profiling markers on CPU and GPU to aid in understanding within timing captures
- Adjustable camera movement speed on screen or in command-line parameters
- Enable/disable DirectStorage and change the location of decompression (GPU or CPU) via command-line parameters
- Switch between using placed or committed resources via command-line parameters
- Enable/disable workload cancellation to test the ability to conserve bandwidth for assets that don't appear in time

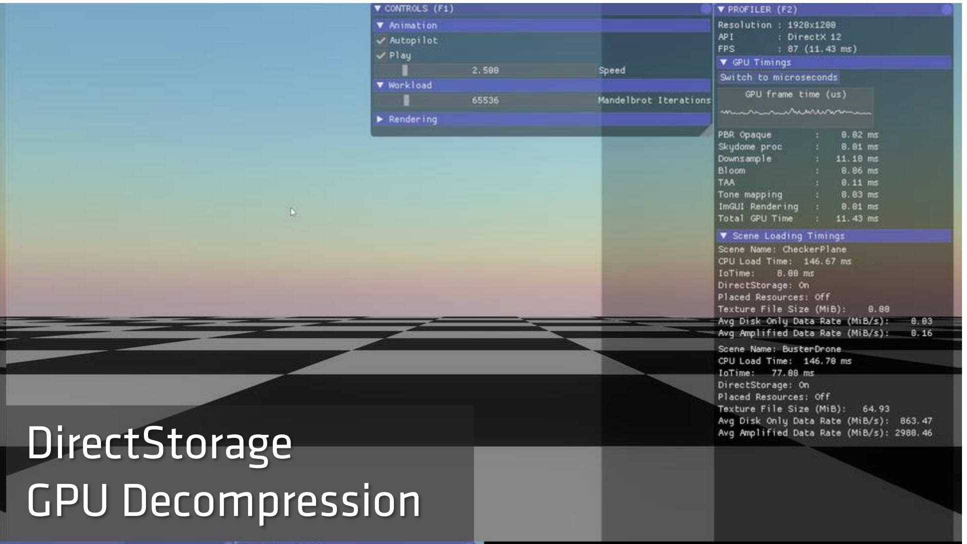
VIDEO COMPARISON



```
▼ CONTROLS (F1)
▼ Animation
✓ Autopilot
✓ Play
   2.500          Speed
▼ Workload
   65536          Mandelbrot Iterations
▶ Rendering
```

```
▼ PROFILER (F2)
Resolution : 1920x1200
API        : DirectX 12
FPS        : 75 (13.16 ms)
▼ GPU Timings
Switch to microseconds
   GPU frame time (us)
PBR Opaque        :   0.02 ms
Skydome proc      :   0.01 ms
Downsample        :  12.93 ms
Bloom             :   0.07 ms
TAA               :   0.11 ms
Tone mapping      :   0.02 ms
ImGUI Rendering   :   0.01 ms
Total GPU Time    :  13.16 ms
▼ Scene Loading Timings
Scene Name: CheckerPlane
CPU Load Time:  130.98 ms
IoTime:    1.00 ms
DirectStorage: On
Placed Resources: Off
Texture File Size (MiB):    0.00
Avg Disk Only Data Rate (MiB/s):    0.25
Avg Amplified Data Rate (MiB/s):    1.26
Scene Name: BusterDrone
CPU Load Time:  163.92 ms
IoTime:   64.00 ms
DirectStorage: On
Placed Resources: Off
Texture File Size (MiB):   64.93
Avg Disk Only Data Rate (MiB/s): 1038.86
Avg Amplified Data Rate (MiB/s): 3585.86
```

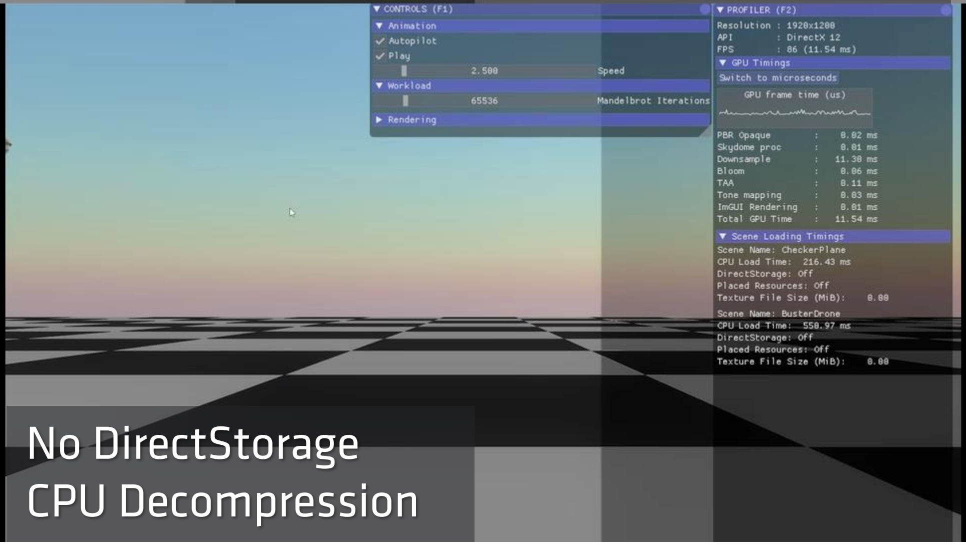
DirectStorage
CPU Decompression



```
▼ CONTROLS (F1)
▼ Animation
✓ Autopilot
✓ Play
   2.500          Speed
▼ Workload
   65536          Mandelbrot Iterations
▶ Rendering
```

```
▼ PROFILER (F2)
Resolution : 1920x1200
API        : DirectX 12
FPS        : 87 (11.43 ms)
▼ GPU Timings
Switch to microseconds
   GPU frame time (us)
PBR Opaque        :   0.02 ms
Skydome proc      :   0.01 ms
Downsample        :  11.18 ms
Bloom             :   0.06 ms
TAA               :   0.11 ms
Tone mapping      :   0.03 ms
ImGUI Rendering   :   0.01 ms
Total GPU Time    :  11.43 ms
▼ Scene Loading Timings
Scene Name: CheckerPlane
CPU Load Time:  146.67 ms
IoTime:    0.00 ms
DirectStorage: On
Placed Resources: Off
Texture File Size (MiB):    0.00
Avg Disk Only Data Rate (MiB/s):    0.03
Avg Amplified Data Rate (MiB/s):    0.16
Scene Name: BusterDrone
CPU Load Time:  146.70 ms
IoTime:   77.00 ms
DirectStorage: On
Placed Resources: Off
Texture File Size (MiB):   64.93
Avg Disk Only Data Rate (MiB/s):  863.47
Avg Amplified Data Rate (MiB/s): 2980.46
```

DirectStorage
GPU Decompression



```
▼ CONTROLS (F1)
▼ Animation
✓ Autopilot
✓ Play
   2.500          Speed
▼ Workload
   65536          Mandelbrot Iterations
▶ Rendering
```

```
▼ PROFILER (F2)
Resolution : 1920x1200
API        : DirectX 12
FPS        : 86 (11.54 ms)
▼ GPU Timings
Switch to microseconds
   GPU frame time (us)
PBR Opaque        :   0.02 ms
Skydome proc      :   0.01 ms
Downsample        :  11.30 ms
Bloom             :   0.06 ms
TAA               :   0.11 ms
Tone mapping      :   0.03 ms
ImGUI Rendering   :   0.01 ms
Total GPU Time    :  11.54 ms
▼ Scene Loading Timings
Scene Name: CheckerPlane
CPU Load Time:  216.43 ms
DirectStorage: Off
Placed Resources: Off
Texture File Size (MiB):    0.00
Scene Name: BusterDrone
CPU Load Time:  550.97 ms
DirectStorage: Off
Placed Resources: Off
Texture File Size (MiB):    0.00
```

No DirectStorage
CPU Decompression

🟩 Loaded

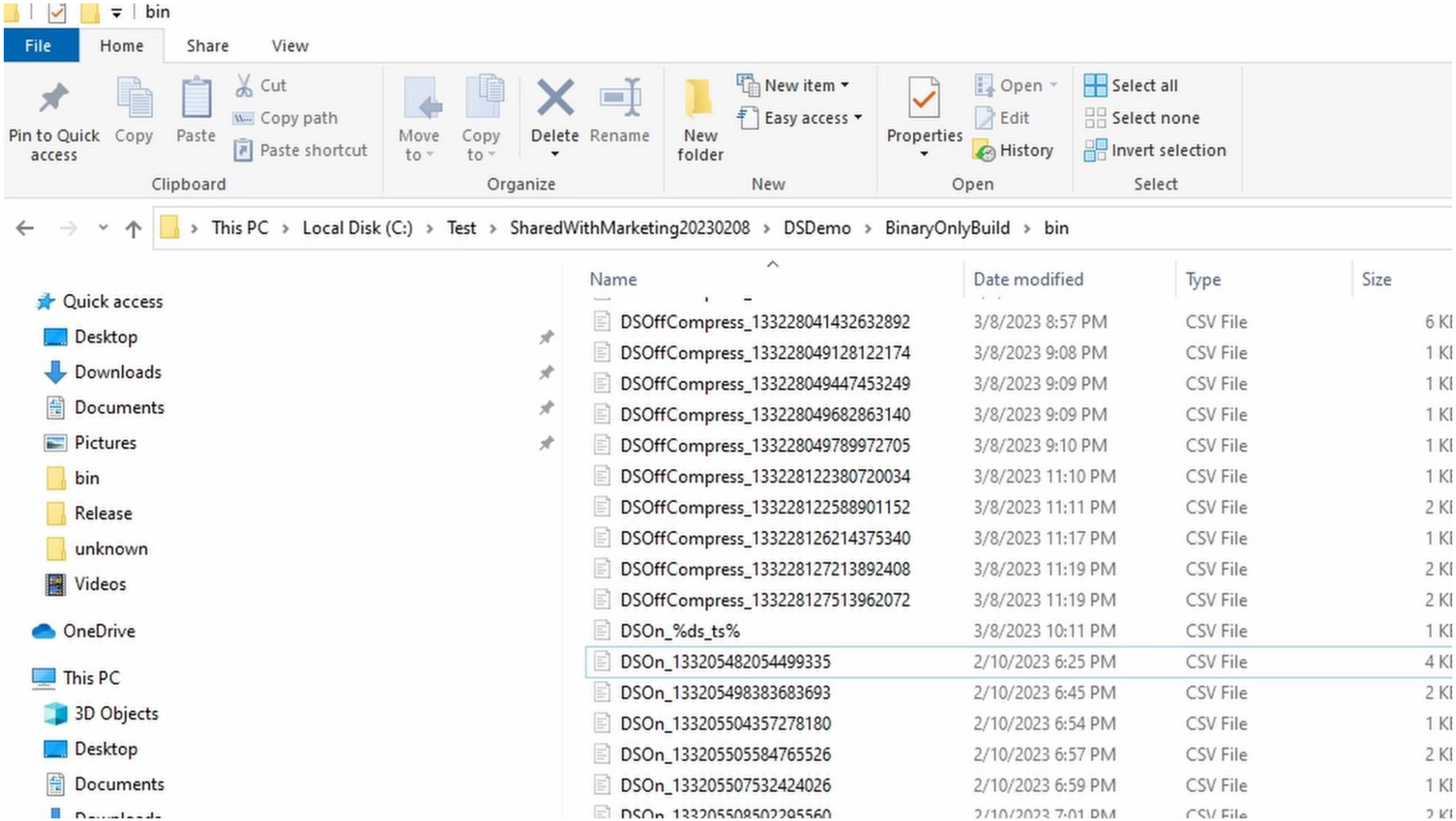
🟨 Loading

⬜ Not Loaded

SAMPLE MEASUREMENTS

- The sample measures the following statistics
  - Load time of an asset – From request to load asset, until asset is presentable on screen
  - ioTime – From first DirectStorage request for an asset until decompression completes on the GPU
  - Disk Only Data Rate – Throughput from the disk during a DirectStorage request.
  - Amplified Data Rate - When compression is enabled, this is the average throughput from first DirectStorage request until completion of the decompressed data size
  - Frame rate before and during an asset streaming in to understand the impact using DirectStorage has on frame rate
  - Number of frames it takes to load each asset to answer how many frames between the asset load request until rendering visibility
- Output is on-screen for interactive analysis and in CSV files for post-runtime analysis

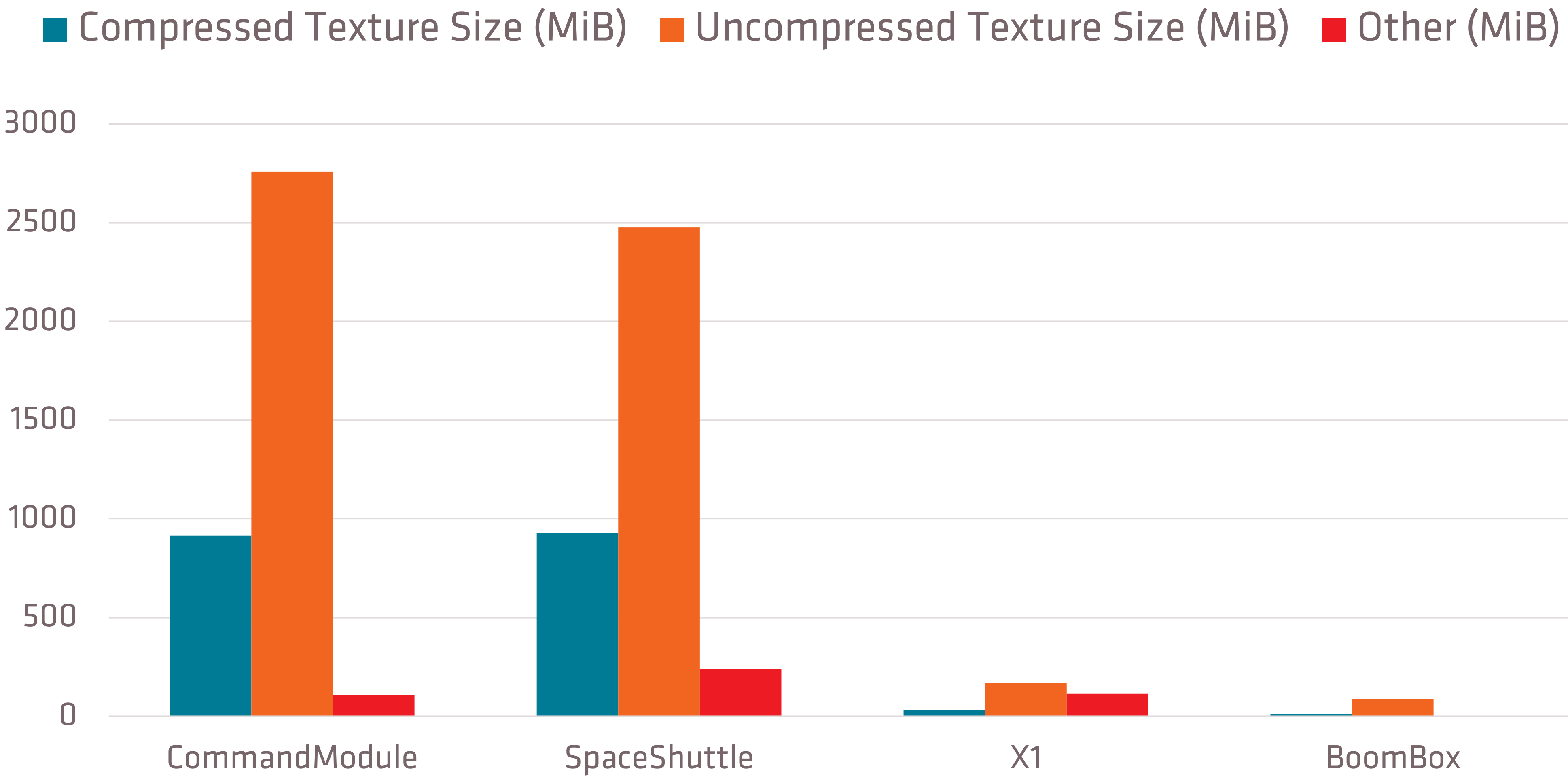
SAMPLE OUTPUT (CSV)



This PC › Local Disk (C:) › Test › SharedWithMarketing20230208 › DSDemo › BinaryOnlyBuild › bin

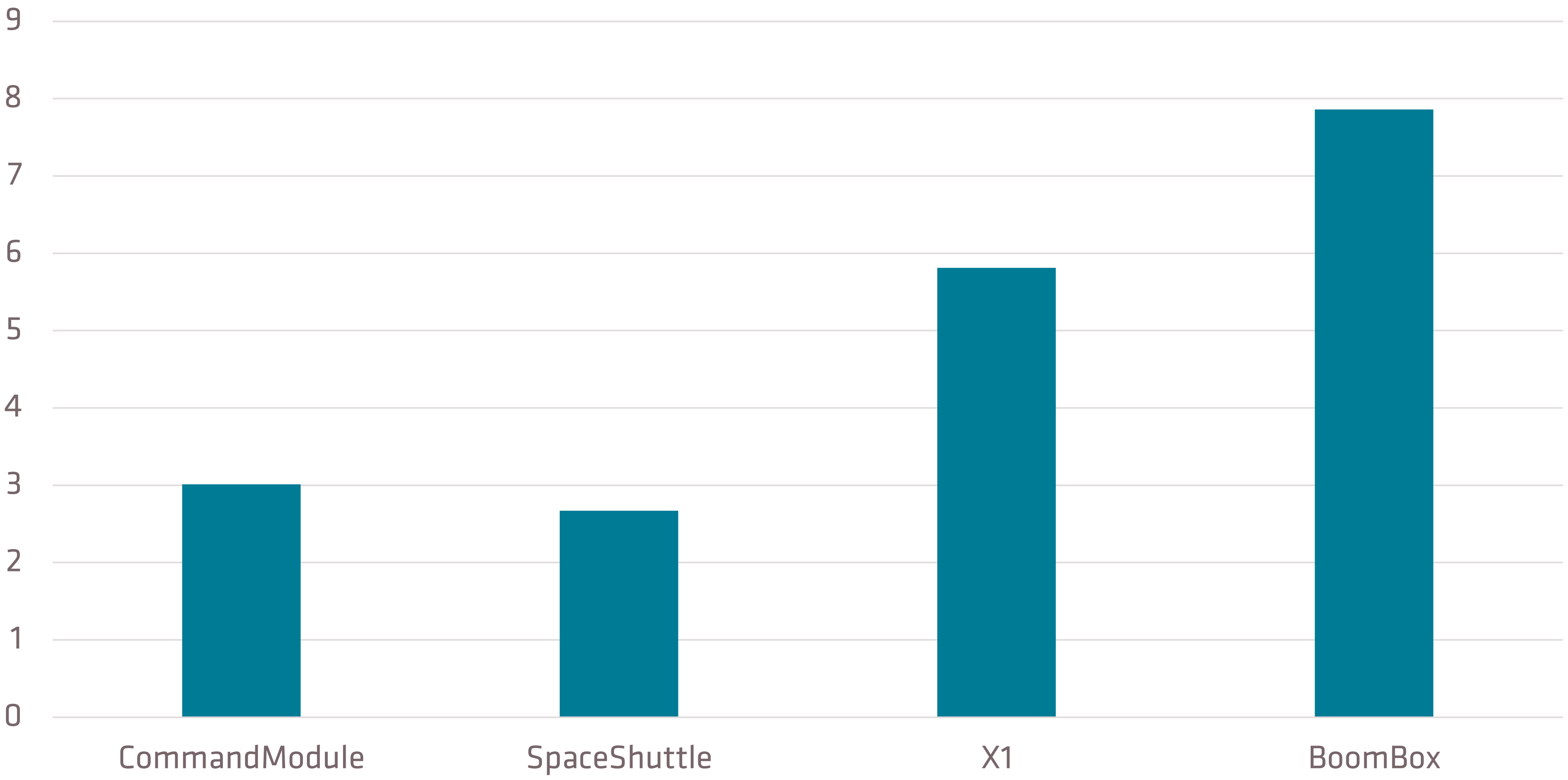
Name	Date modified	Type	Size
DSOffCompress_133228041432632892	3/8/2023 8:57 PM	CSV File	6 KI
DSOffCompress_133228049128122174	3/8/2023 9:08 PM	CSV File	1 KI
DSOffCompress_133228049447453249	3/8/2023 9:09 PM	CSV File	1 KI
DSOffCompress_133228049682863140	3/8/2023 9:09 PM	CSV File	1 KI
DSOffCompress_133228049789972705	3/8/2023 9:10 PM	CSV File	1 KI
DSOffCompress_133228122380720034	3/8/2023 11:10 PM	CSV File	1 KI
DSOffCompress_133228122588901152	3/8/2023 11:11 PM	CSV File	2 KI
DSOffCompress_133228126214375340	3/8/2023 11:17 PM	CSV File	1 KI
DSOffCompress_133228127213892408	3/8/2023 11:19 PM	CSV File	2 KI
DSOffCompress_133228127513962072	3/8/2023 11:19 PM	CSV File	2 KI
DSOn_%ds_ts%	3/8/2023 10:11 PM	CSV File	1 KI
DSOn_133205482054499335	2/10/2023 6:25 PM	CSV File	4 KI
DSOn_133205498383683693	2/10/2023 6:45 PM	CSV File	2 KI
DSOn_133205504357278180	2/10/2023 6:54 PM	CSV File	1 KI
DSOn_133205505584765526	2/10/2023 6:57 PM	CSV File	2 KI
DSOn_133205507532424026	2/10/2023 6:59 PM	CSV File	1 KI
DSOn_133205508502295560	2/10/2023 7:01 PM	CSV File	2 KI

ASSET SIZES



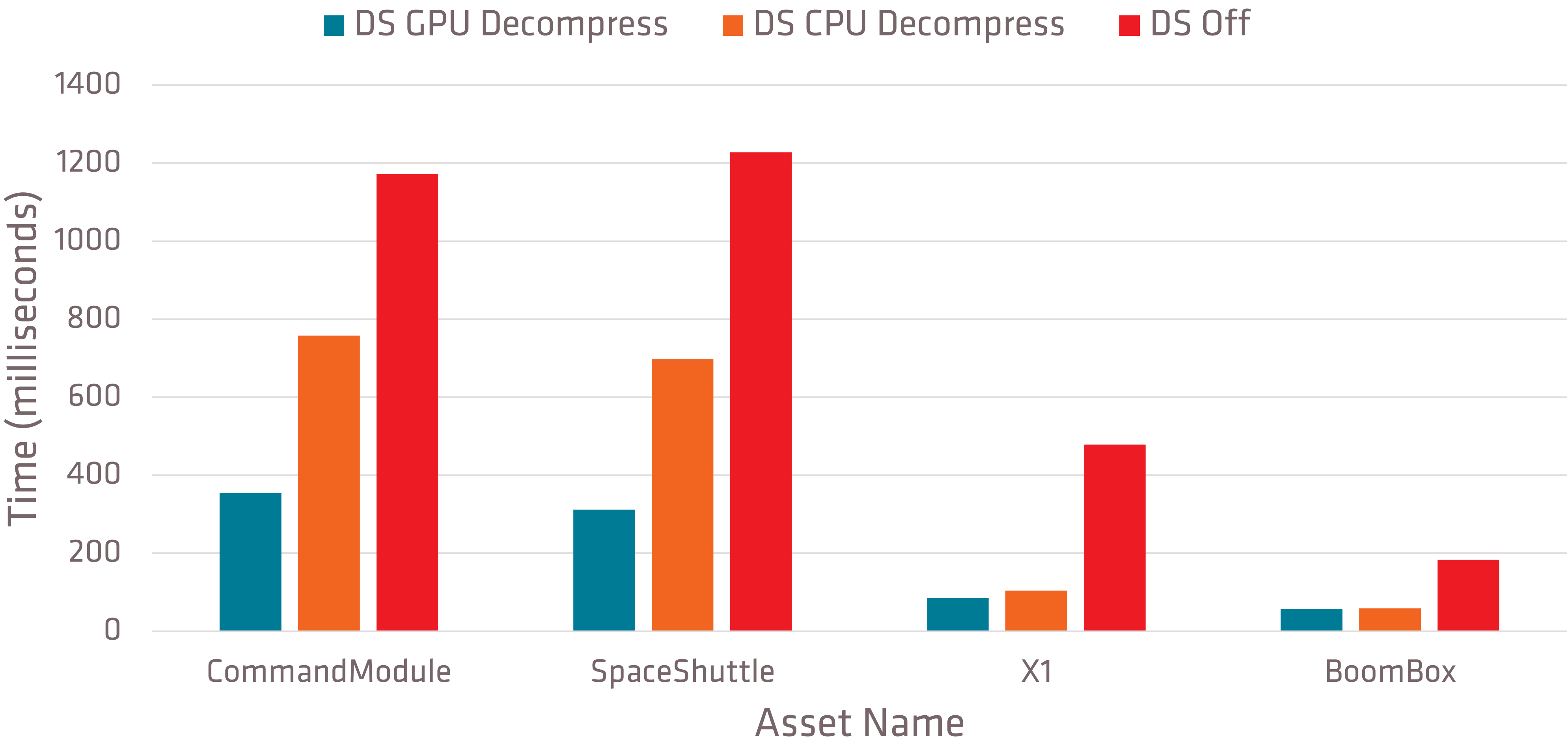
■ Compressed Texture Size (MiB)
■ Uncompressed Texture Size (MiB)
■ Other (MiB)
3000
2500
2000
1500
1000
500
0
CommandModule
SpaceShuttle
X1
BoomBox

COMPRESSION RATIO (GDEFLATE DEFAULT)

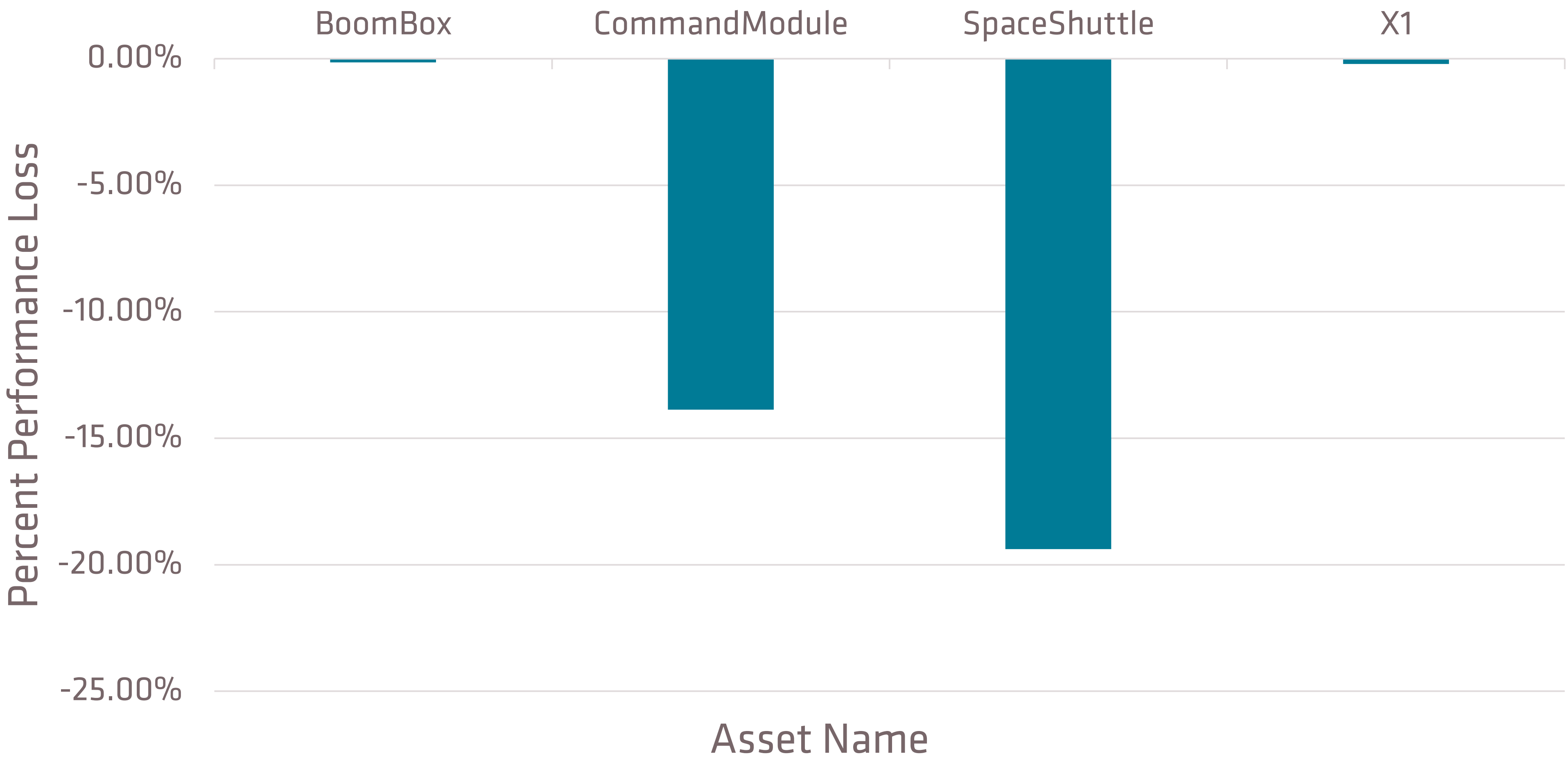


* Textures in non-BC formats!

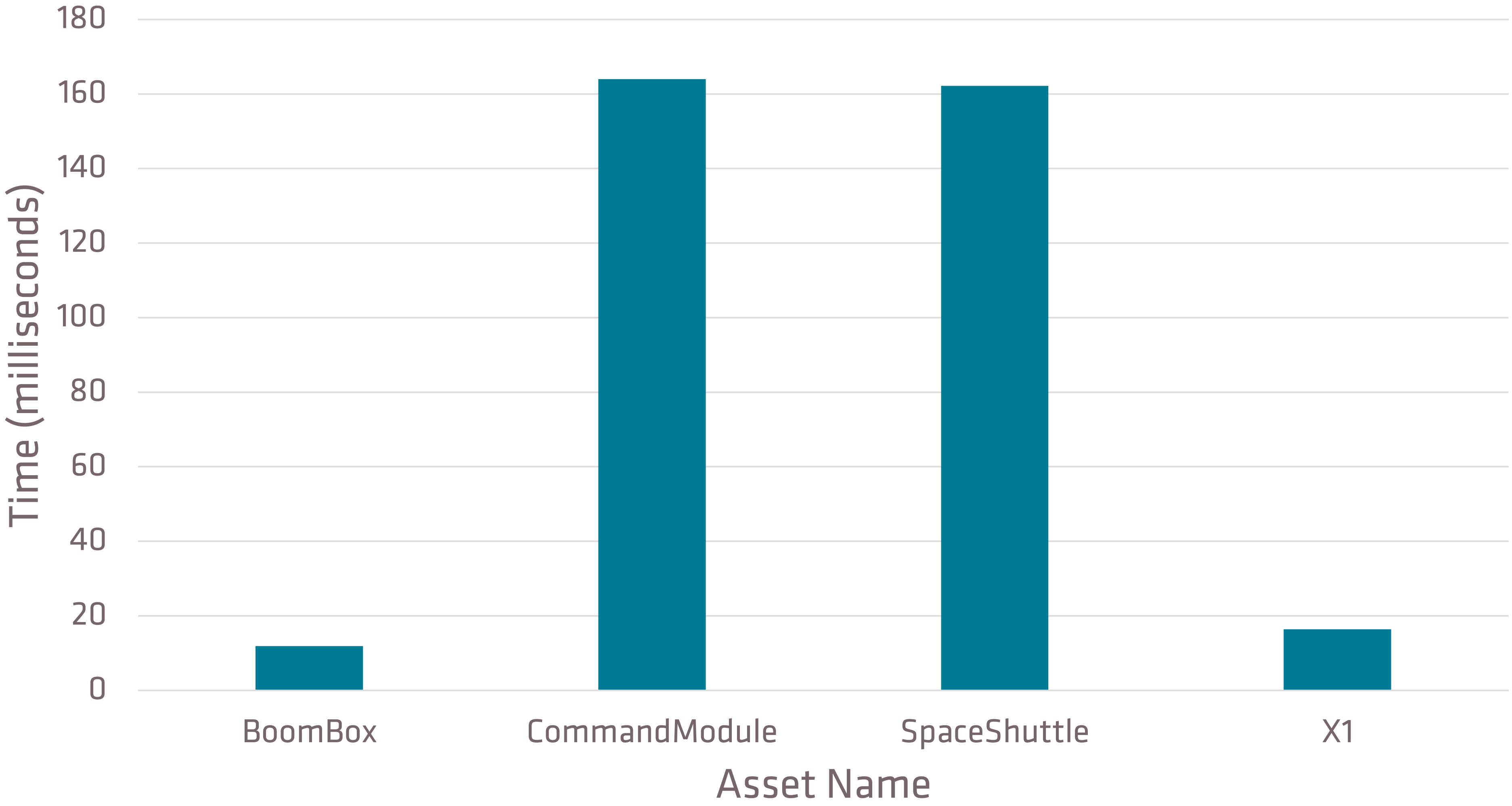
LOAD TIME DIRECTSTORAGE VS NO DIRECTSTORAGE (LOWER IS BETTER)



PERFORMANCE IMPACT OF DECOMPRESSION ON OTHER WORKLOADS (LOWER IS WORSE)



LENGTH OF PERFORMANCE IMPACT FROM DECOMPRESSION (HIGHER IS WORSE)



TIPS AND RECOMMENDATIONS

- For streaming in textures, placed resources are faster as long as the heap is created well ahead of time.
  - Our Demo does not create the heap well ahead of time, and you can see the consequences in PIX.
- Staging buffer size should be >= 256MiB for best performance.
- Use custom compression on CPU-only resources because GDeflate is slower than many decompression algorithms when it executes on the CPU.
- Fill in the debug field of each request such that it's easy to understand the real source and reason for an I/O request.
- For titles targeting extremely low-end hardware without dedicated GPU, for example laptops, consider adding a checkbox to force DS to switch to CPU decompression.
- DirectStorage supports unaligned reads, but for maximum performance, consider aligning data to page size.

PROFILING

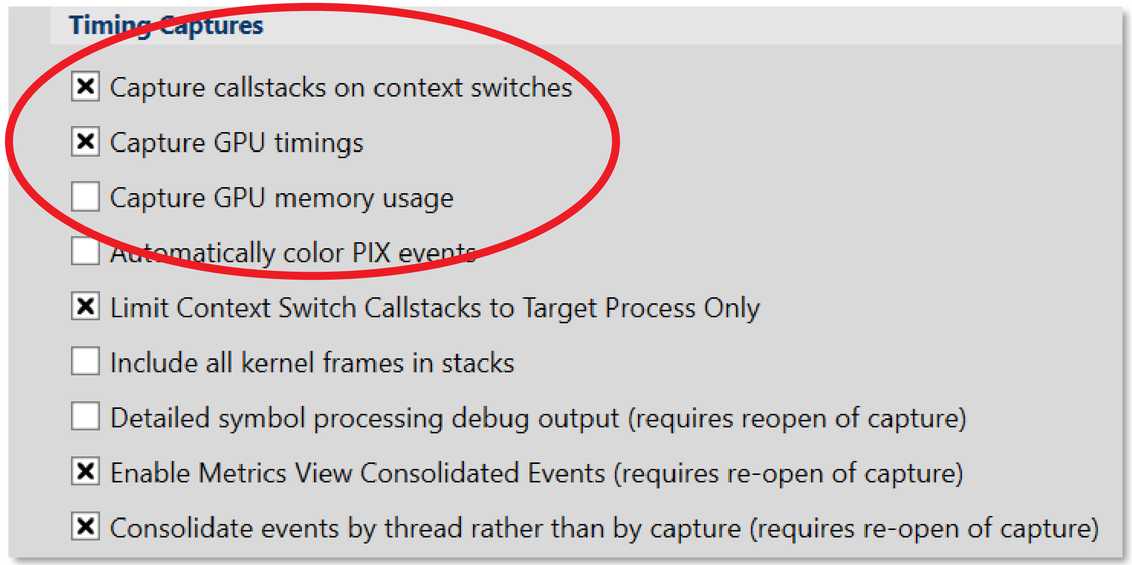
PIX Timing Capture

CONFIGURING PIX LAUNCH OPTIONS

Disable capture GPU memory usage

Enable GPU timings to see GPU Decompression

Callstacks on context switches are useful



Timing Captures

☒ Capture callstacks on context switches

☒ Capture GPU timings

☐ Capture GPU memory usage

☐ Automatically color PIX events

☒ Limit Context Switch Callstacks to Target Process Only

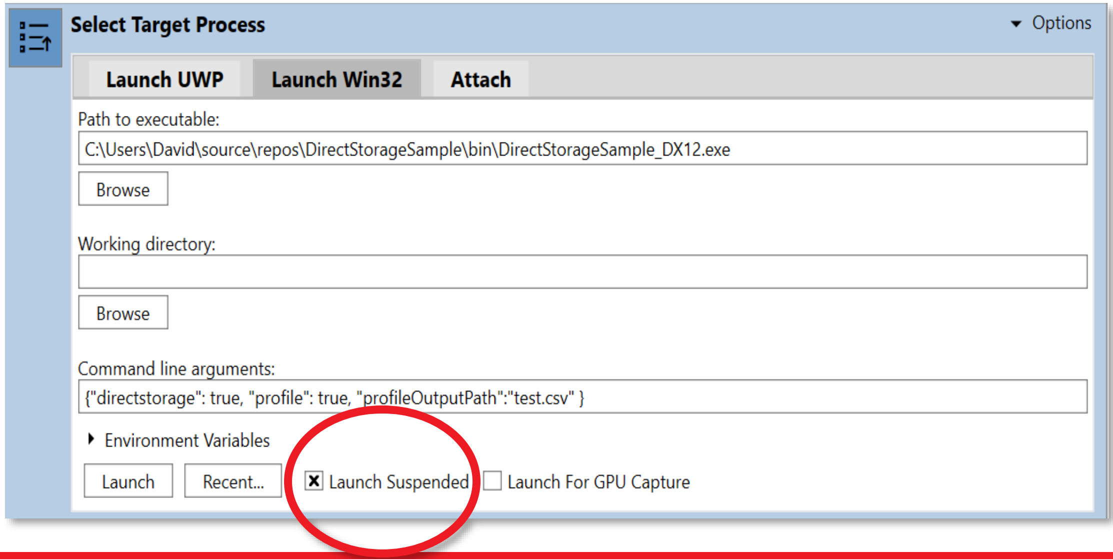
☐ Include all kernel frames in stacks

☐ Detailed symbol processing debug output (requires reopen of capture)

☒ Enable Metrics View Consolidated Events (requires re-open of capture)

☒ Consolidate events by thread rather than by capture (requires re-open of capture)

For load time profiling, launch suspended



Path to executable:

C:\Users\David\source\repos\DirectStorageSample\bin\DirectStorageSample_DX12.exe

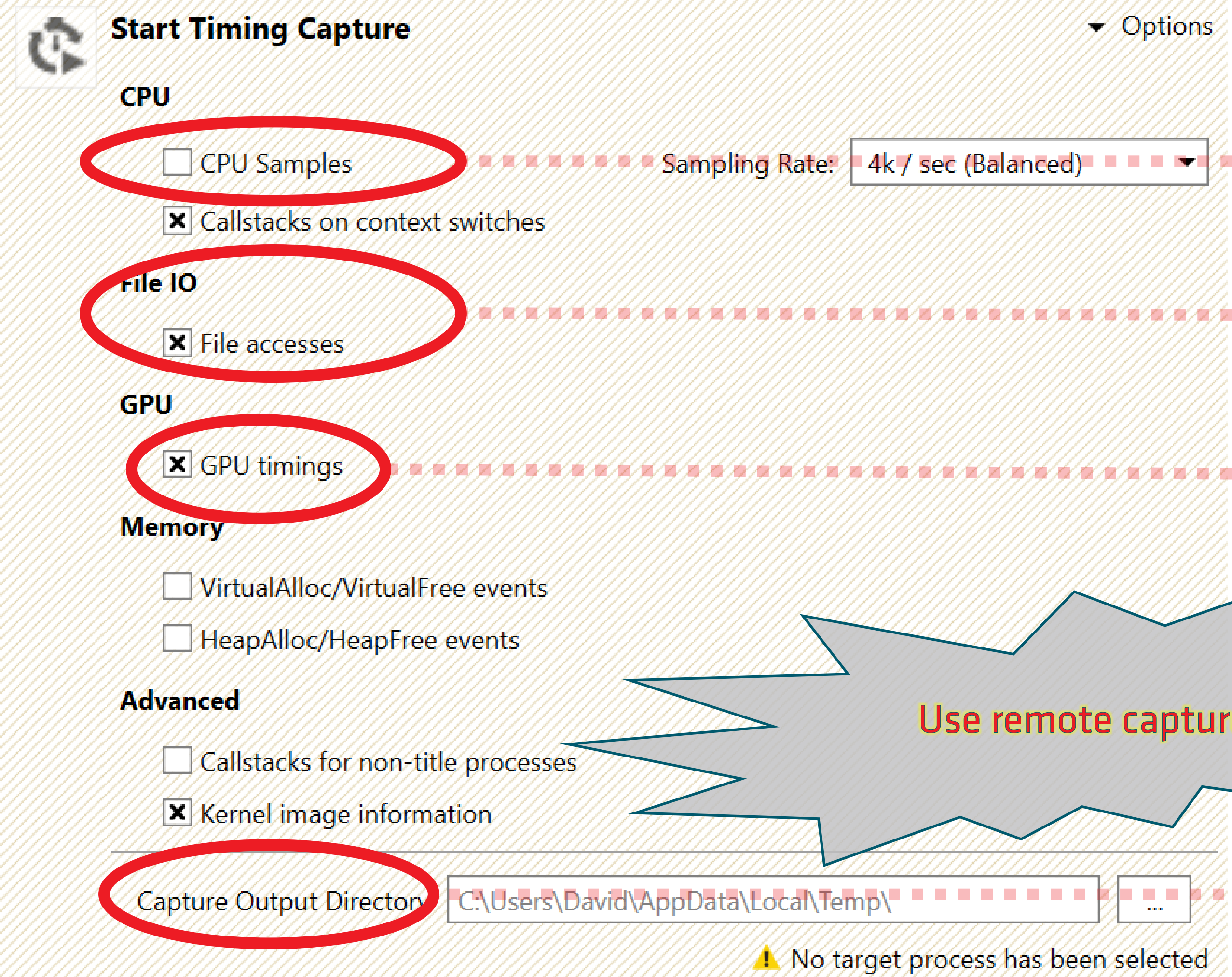
Working directory:

Command line arguments:

{"directstorage": true, "profile": true, "profileOutputPath":"test.csv" }

☒ Launch Suspended ☐ Launch For GPU Capture

TIMING CAPTURE OPTIONS



Start Timing Capture ▾ Options

CPU

□ CPU Samples Sampling Rate: 4k / sec (Balanced) ▾

☒ Callstacks on context switches

File IO

☒ File accesses

GPU

☒ GPU timings

Memory

□ VirtualAlloc/VirtualFree events

□ HeapAlloc/HeapFree events

Advanced

□ Callstacks for non-title processes

☒ Kernel image information

Capture Output Directory C:\Users\David\AppData\Local\Temp\ ...

⚠ No target process has been selected

Only enable if necessary

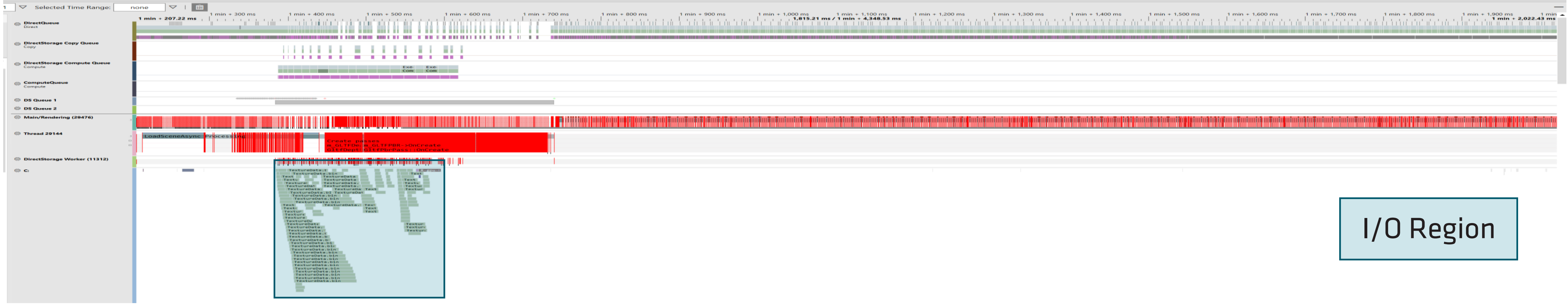
Queue depth, underutilization, gaps are hints. *BypassIO currently not shown

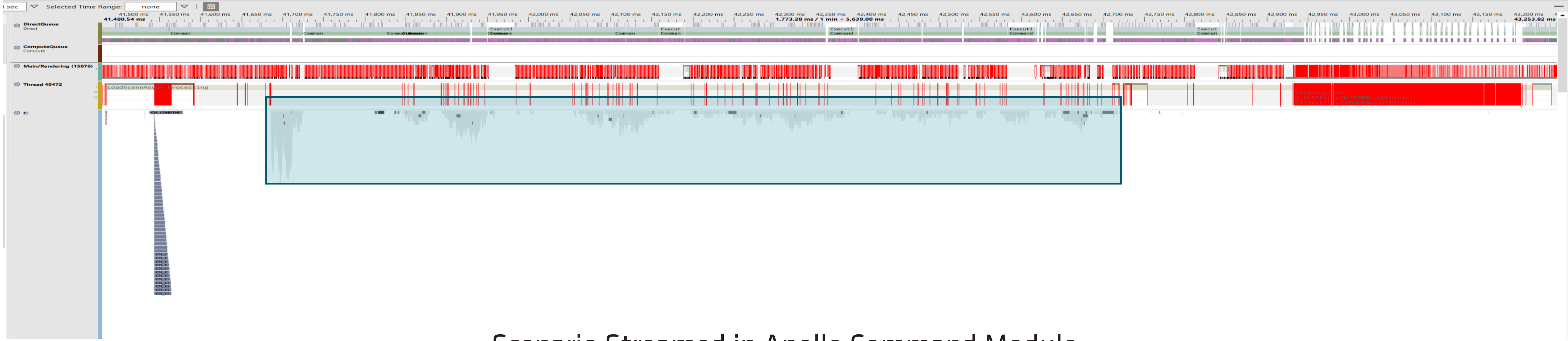
See DS controlled queues and workloads. Impact of decompression on rendering, pipelining, etc.

Use remote capture!

Select a device other than the one the application loads assets from

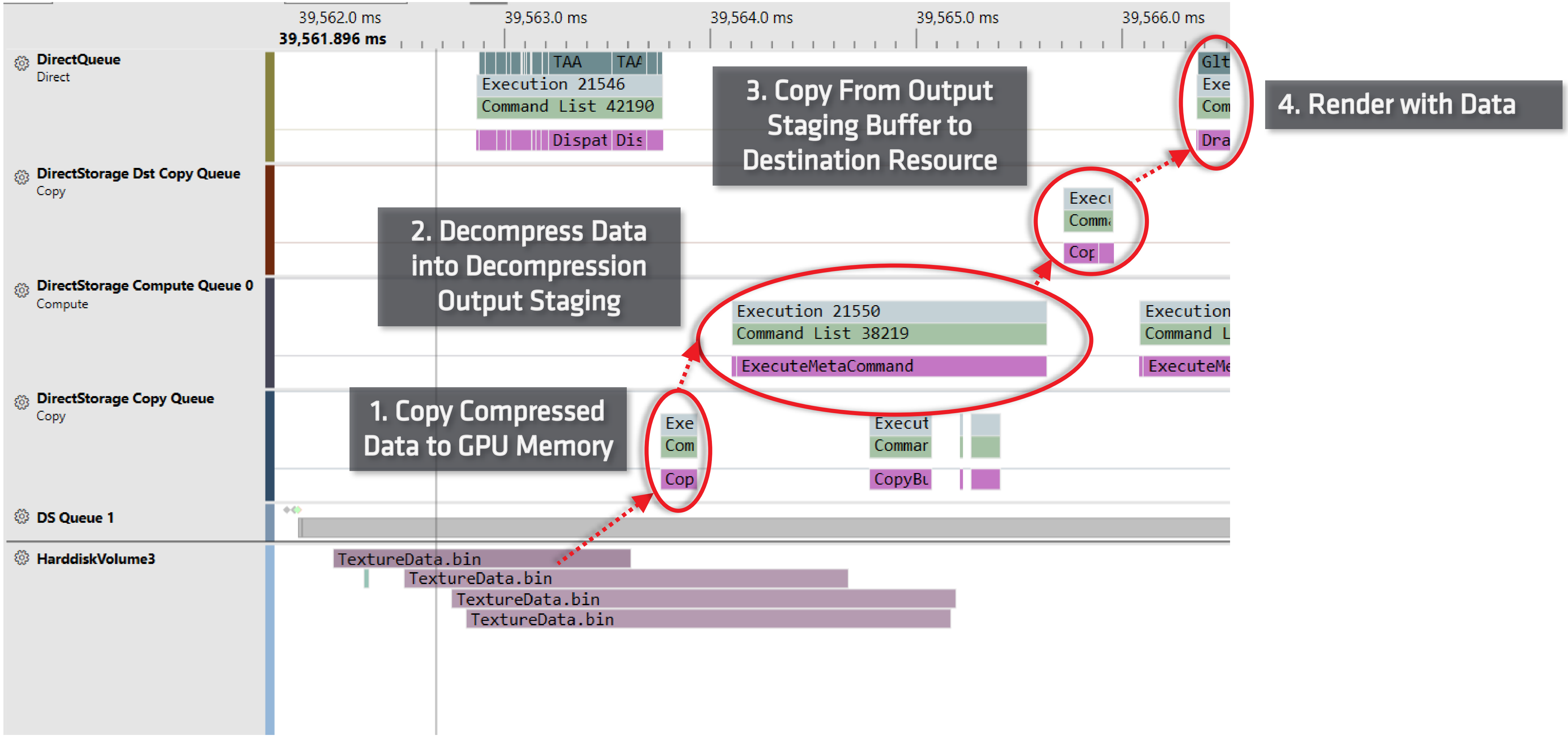
TIMING CAPTURE
BIRD'S EYE VIEW COMPARISON DS GPU DECOMPRESSION VS NO DS





Scenario Streamed in Apollo Command Module

TIMING CAPTURE - GPU DECOMPRESSION



1. Copy Compressed Data to GPU Memory

2. Decompress Data into Decompression Output Staging

3. Copy From Output Staging Buffer to Destination Resource

4. Render with Data

EFFECTIVE DIRECTSTORAGE INTEGRATIONS...

- Reduce read->read dependencies
- Batch
- Ensure transfer sizes are appropriate
- Use timing capture in PIX to find significant places of CPU Work

DIRECTSTORAGE DISTRIBUTION

Link to Microsoft® DirectStorage SDK and Samples: https://aka.ms/directstorage/

SYSTEM REQUIREMENTS

- Windows® 10 or Windows® 11
- Shader Model 6 compatible GPU to support GPU Decompression
- Does not require NVMe™

THANKS

Rosanna Ashworth-jones

Cassie Hoef

Damyan Pepper

Nicolas Thibieroz

Tom Lewis

QUESTIONS?

LEGAL DISCLAIMERS

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.
THIS INFORMATION IS PROVIDED 'AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
AMD, the AMD Arrow logo, AMD Radeon™, AMD Ryzen™, Radeon™ and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies. PCIe® is a registered trademark of PCI-SIG Corporation, Windows is a registered trademark of Microsoft Corporation in the US and/or other countries.
NVMe is a trademark of NVM Express, Inc

ADDITIONAL DISCLAIMERS AND ATTRIBUTIONS

[1] Microsoft, "Windows Driver Documentation," 3 August 2021. [Online]. Accessed 1 Februrary 2023. Available: https://github.com/MicrosoftDocs/windows-driver-docs/blob/11aa5c3e15a4a29158a9a67357c8308bb17315b1/windows-driver-docs-pr/ifs/images/bypass-io-path.jpg. CC BY-4.0.

Third-party content is licensed to you directly by the third party that owns the content and is not licensed to you by AMD. ALL LINKED THIRD-PARTY CONTENT IS PROVIDED “AS IS” WITHOUT A WARRANTY OF ANY KIND. USE OF SUCH THIRD-PARTY CONTENT IS DONE AT YOUR SOLE DISCRETION AND UNDER NO CIRCUMSTANCES WILL AMD BE LIABLE TO YOU FOR ANY THIRD-PARTY CONTENT. YOU ASSUME ALL RISK AND ARE SOLELY RESPONSIBLE FOR ANY DAMAGES THAT MAY ARISE FROM YOUR USE OF THIRD-PARTY CONTENT.