Flash Memory Summit

SOLIDIGM

(De)Compression using Computational Storage Drives (CSD)

Guangming Lu, Sr. Storage Architect

Vladimir Alves, Sr. Director Pathfinding

Disclaimers

All product plans, roadmaps, specifications, and product descriptions are subject to change without notice.

Nothing herein is intended to create any express or implied warranty, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, or any warranty arising from course of performance, course of dealing, or usage in trade.

Contact your Solidigm representative or your distributor to obtain the latest specifications before placing your product order.

For copies of this document, documents that are referenced within, or other Solidigm literature, please contact your Solidigm representative.

All products, computer systems, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.

Some results have been estimated or simulated using internal Solidigm analysis or architecture simulation or modeling, and provided to you for information purposes only. Any differences in your system hardware, software or configuration may affect your actual performance.

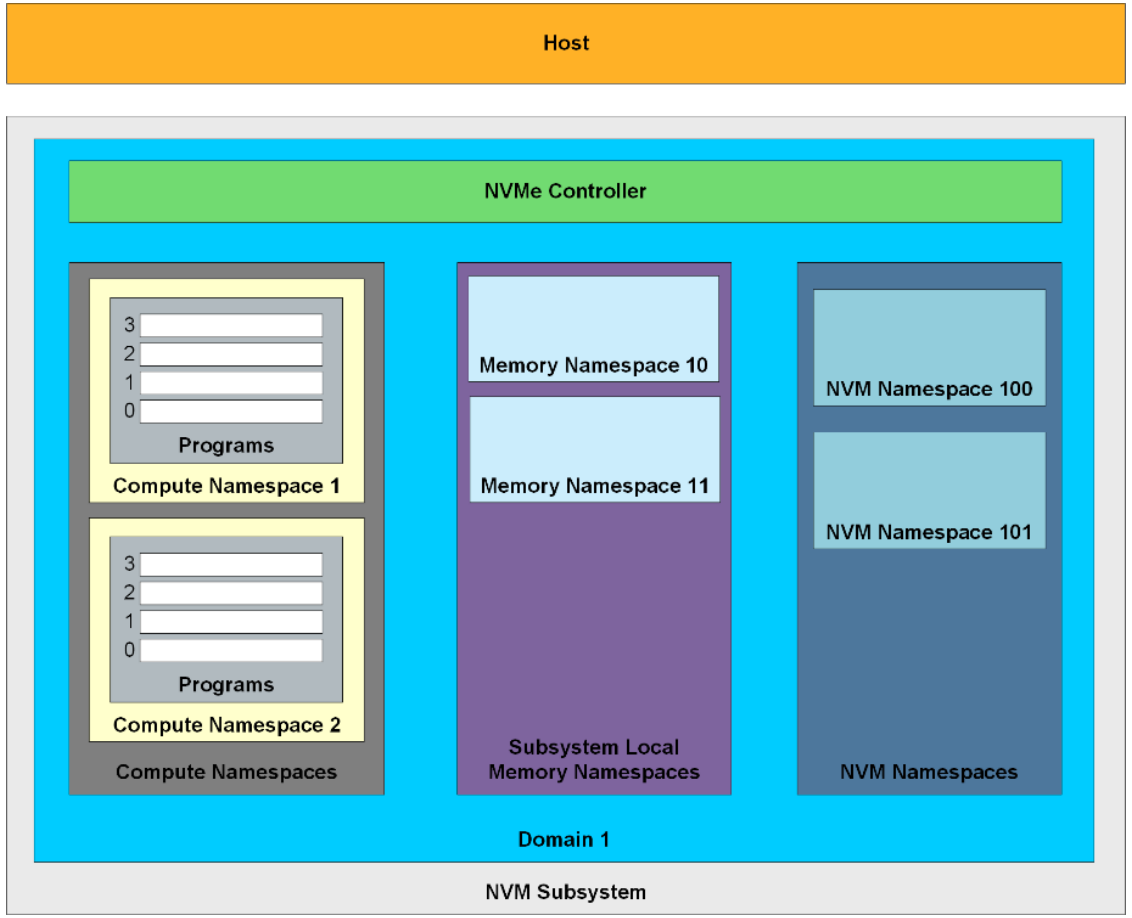
Outline

- Type of Compression Implementation
- Major CS Architectural Components
- in-line Compression Vs CSD Compression
- Comp/Decomp in CSD
- Companion User Library
- Performance and Scalability
- Use case of CSD (De)Compression
- Use case of CSD (De)Compression As Service
- Future work

Compression/Decompression Approaches

- Software: Flexible but CPU intensive, low throughput and lack scalability.
- CPU with built-in C/D engine, such as IBM POWER9 and z15.
  - More throughput requirement, more CPUs are needed.
- Accelerator add-in card. (GPU, ASIC or FPGA).
  - More data, more add-in cards.
  - Take some PCIe slots.
- NVMe SSD with built-in in-line C/D engine.
  - Highly scalable.
  - SSD FW must handle variable LBA size.
  - Typically, lower compression ratio.
  - Lacks the flexibility to choose the suitable compression algorithms.
- C/D engine in CSD.
  - Highly scalable.
  - Works with companion library.
  - SW compatible.
  - Flexibility to choose the suitable compression algorithms.

Major Architectural Components



The NVMe® computational storage architecture involves several types of namespaces:

- Compute namespaces (new – TP4091)
- Memory namespaces (new – TP4131)
- NVM namespaces
  - NVM, Zoned, and Key Value namespaces

Programs operate on data in Subsystem Local Memory

- Includes program input, output
- Data is copied between Subsystem Local Memory and host memory using new NVMe commands

NVMe Computational Storage

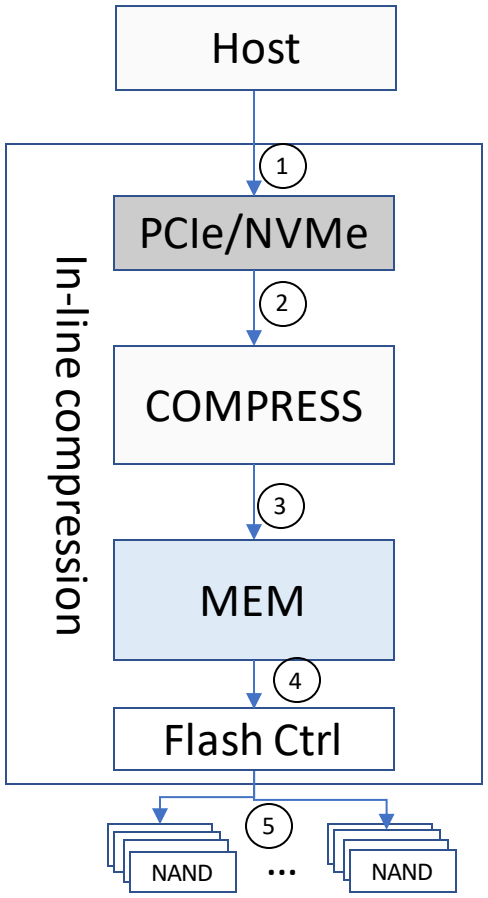
- **TP4091: Computational Programs**
- New I/O command set for computational namespaces
- Commands:
  - Load program
  - Activate program
  - Execute program
  - Create/Delete Memory Range Set
- Support for Identify Controller, Namespace

- **TP4131: Subsystem Local Memory**
- New I/O command set for memory namespaces
- Commands:
  - Memory Read
  - Memory Write
  - Memory Fill
  - Memory Copy
- Support for Identify Controller, Namespace

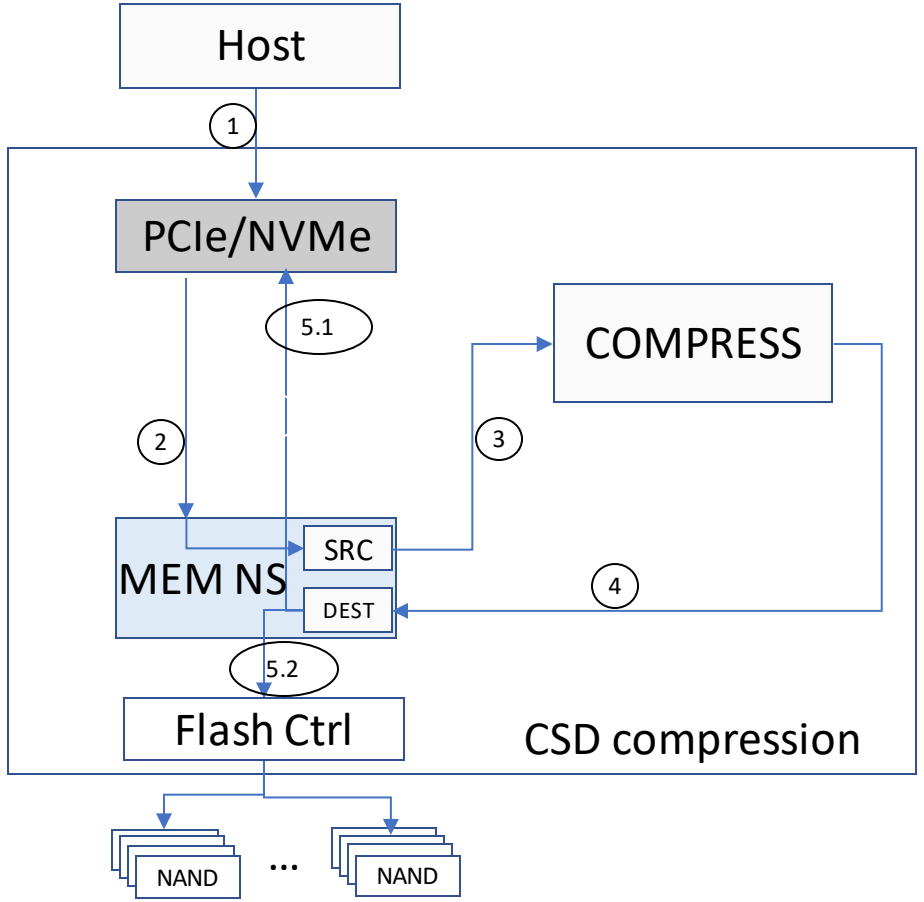
- NVM Copy: TP4130 Cross NameSpace Copy

In-line Compression vs. CSD Compression

1. In-line C/D is only used internally. Not visible to outside. Fixed or sub-set algorithm.
2. CSD C/D needs to be as versatile as possible. Support multiple algorithms.



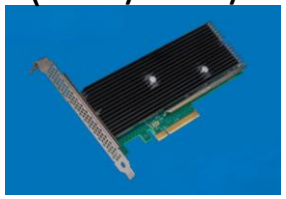
1. Neither FileSystem nor Driver need modification & transparent
2. Works on LBA granularity
3. Close system
4. But SSD FW becomes more complicated since it has to handle variable LBA size(after compression)
5. Larger DDR size to hold L2P due to more bits(size/offset) in each L2P entry.
6. C/D can be customized.



1. Need user library w/ FileSystem modification to log compressed file size, etc.
2. Works on file granularity
3. Open system
4. But SSD FW keeps same with additional CSD FW.
5. Send compressed data to host or NAND.
6. Choose most suitable algorithm for incoming file.
7. Higher Compression Ratio.
8. Expose Engines to outside of CSD. Compliant with Spec.

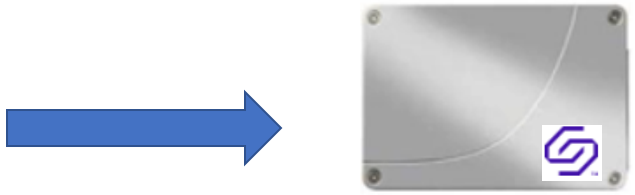
Comp/Decomp in CSD

Add-in (De)Compression Card(GPU/ASIC/FPGA)



+

Solidigm Traditional SSD



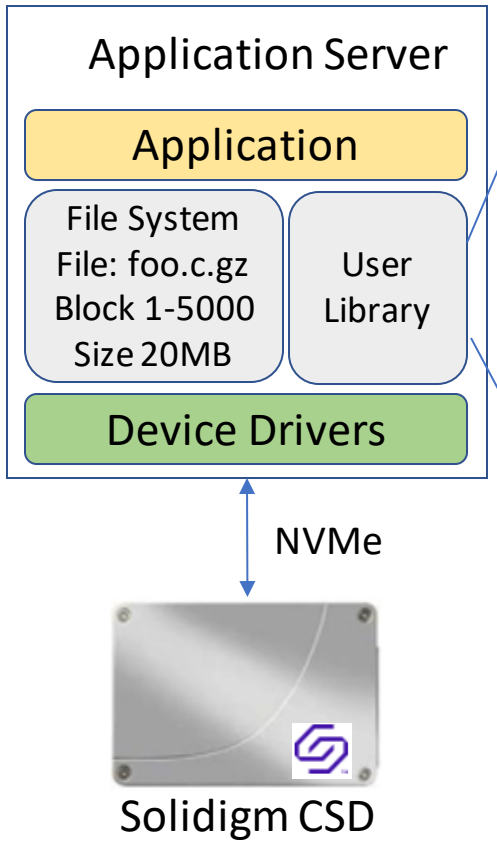
Solidigm Computational Storage Drive (CSD) w/ Compression & Decompression Functionalities

- (De)Compression Engines exposed to SW.
- Multiple (De)Compression Algorithm supported.
- Performance Scalable with # of CSD

Flexible combinations:

- SW compression for best CR; CSD decompression. Or vice versa
- Can configured as Accelerator (Both source/destination data in host/NAND) for (De)Compression as Service.

Companion User Library



Basic Library Functions

- Abstract device layer
- Handle cache coherency
- Build & Issue "EXEC" commands
- Prepare memory space for return data
- Harvests results
- Custom info. returns of public API
- Utilize SNIA CS API

Custom information example:

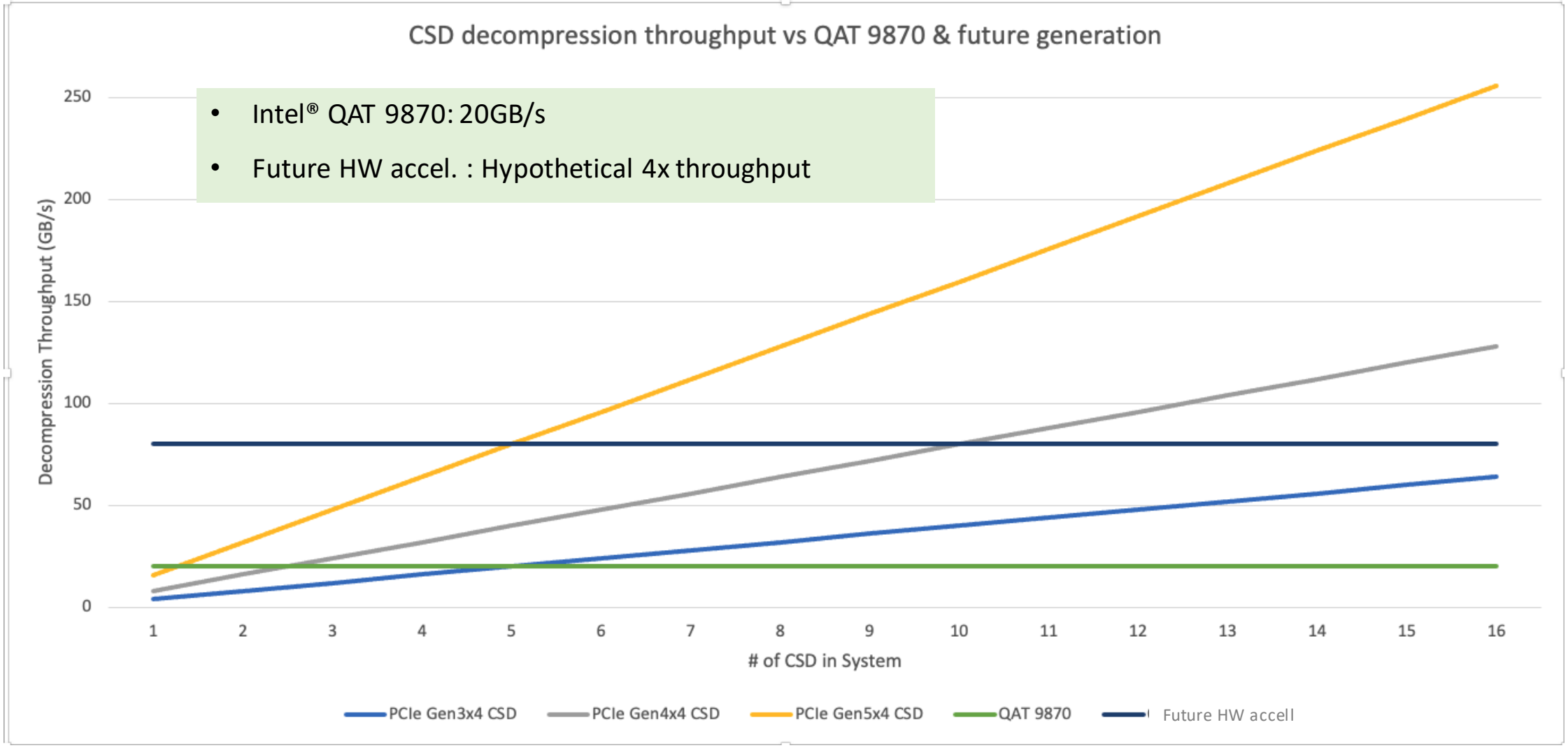
- EXEC Completion with some extra info:
  1. Mem ID# in NS# has #Dword data ready to pickup.
  2. CSD C/D Engine needs more data.

EXEC(ComputeNS#, op=#(Decompression),
SRC(MemoryNS#,Start addr=#, bufsz=#),
DEST(MemoryNS#,Start addr=#, bufsz=#),
NVMeNS# LBA1-LBA5000, SIZE=20MB)

TP4091 Cmd

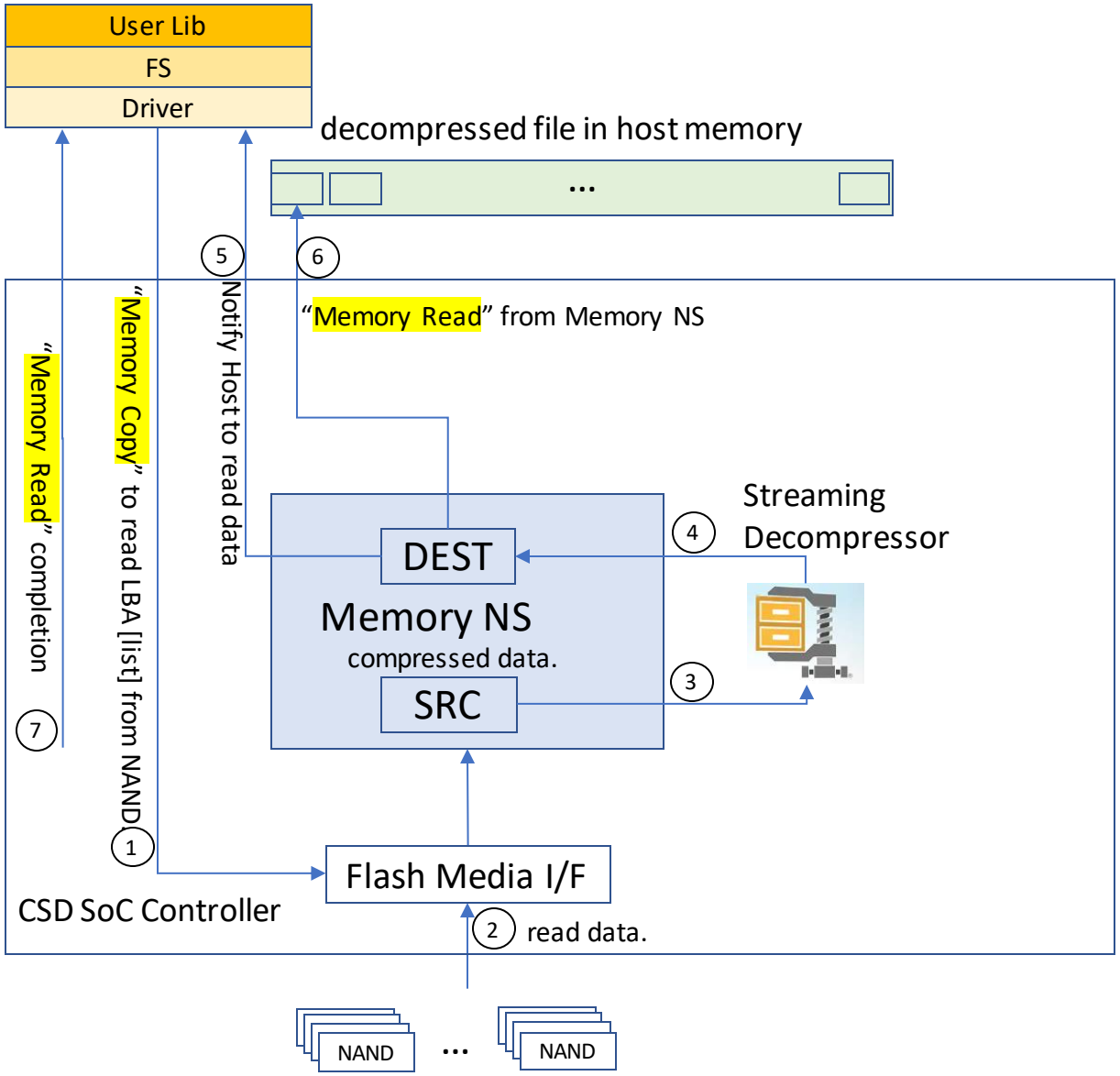
TP4131 Cmd

Performance and Scalability



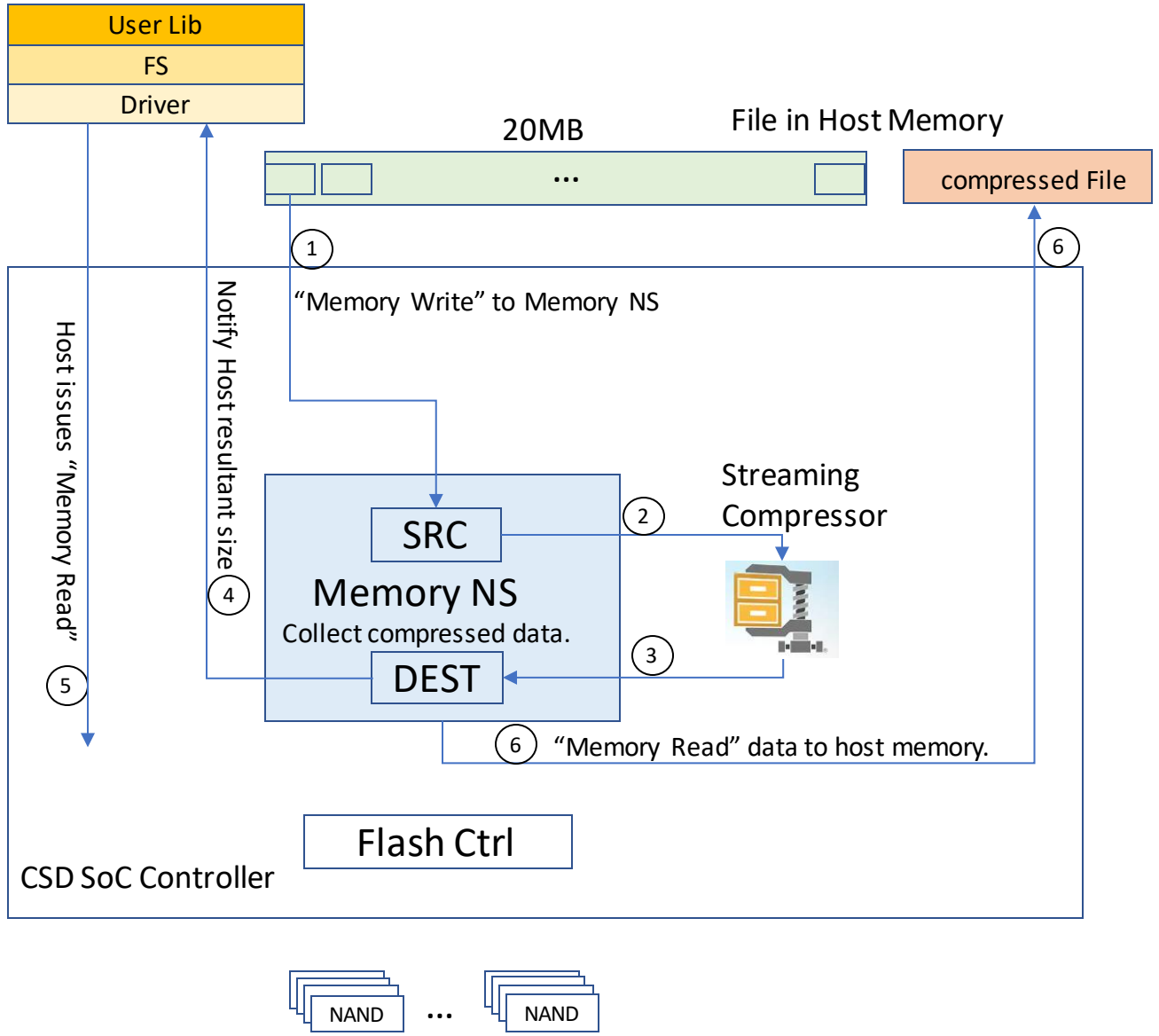
Modeling results assuming in-storage compression matches SSD max read bandwidth

NVMe Read + Decompression



1. TP4091 steps: load/activate/execute program not shown here.
2. User Lib orchestrates the data flow outside CSD.
3. CSD Decompression program orchestrates internal cmd & data flow.
4. Source data in NAND, destination data in host memory.
5. Source file may be compressed in SW.
6. Decompressor fully compatible with selected decompression algorithm.
7. "***Memory Read***", "***Memory Copy***" are TP4131 commands.

Compression as Service



1. TP4091 steps: load/activate/execute program, are already done, not shown here.
2. User Lib orchestrates data flow outside CSD.
3. CSD Compression program orchestrates internal cmd & data flow.
4. Both source and destination data in host memory.
5. Engine works as a compression accelerator.
6. "***Memory Read***", "***Memory write***" are TP4131 commands.

Future Pathfinding Work

- Using the ratified TPs in a future NVMe CSD solution once the updated NVM Express spec is released.
  - Align with NVMe TP4091
  - Align with NVMe TP4131

- Leverage the SNIA Architecture and Programming Model, and newly released Computational Storage API to delivery an industry standard solution to market.

Thank You