Microsoft

Using non-volatile memory (NVDIMM-N) as byte-addressable storage in Windows Server 2016

Tobias Klima
Program Manager

#Build2016

The Case for Byte-Addressability

Problem

Storage Class Memory (NVDIMM-N) performance is not optimal in block mode (software overhead)

Opportunity

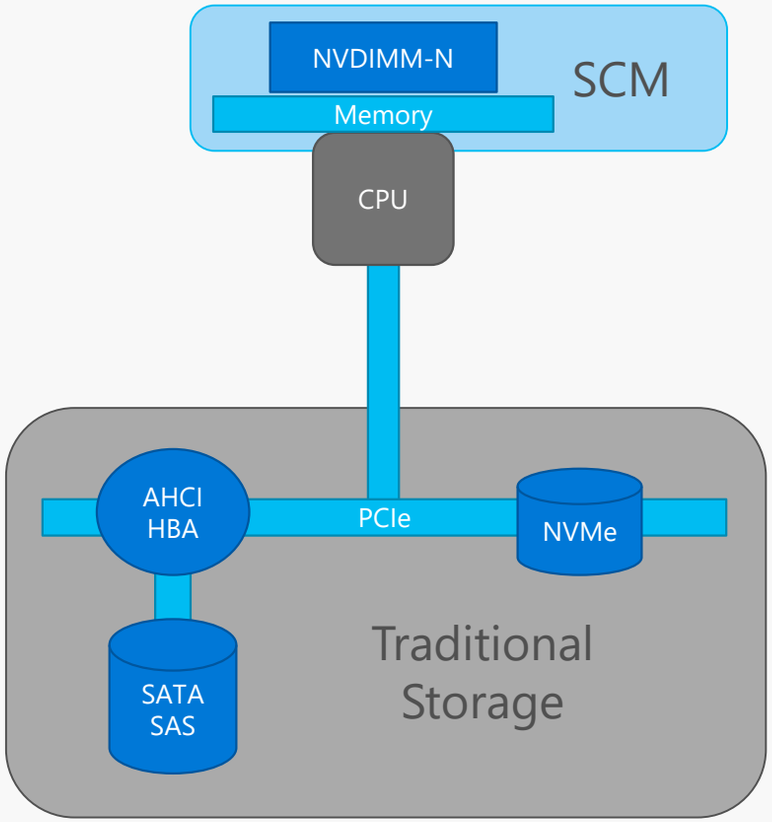
Utilize NVDIMM-N in byte-addressable (DAX) mode to achieve full performance potential

Approach

Memory-Map files on DirectAccess (DAX) volumes

NVDIMM-N supported in WS 2016

Exposes Block Interface (like a Disk), as well as byte-addressability option (DAX Volume)



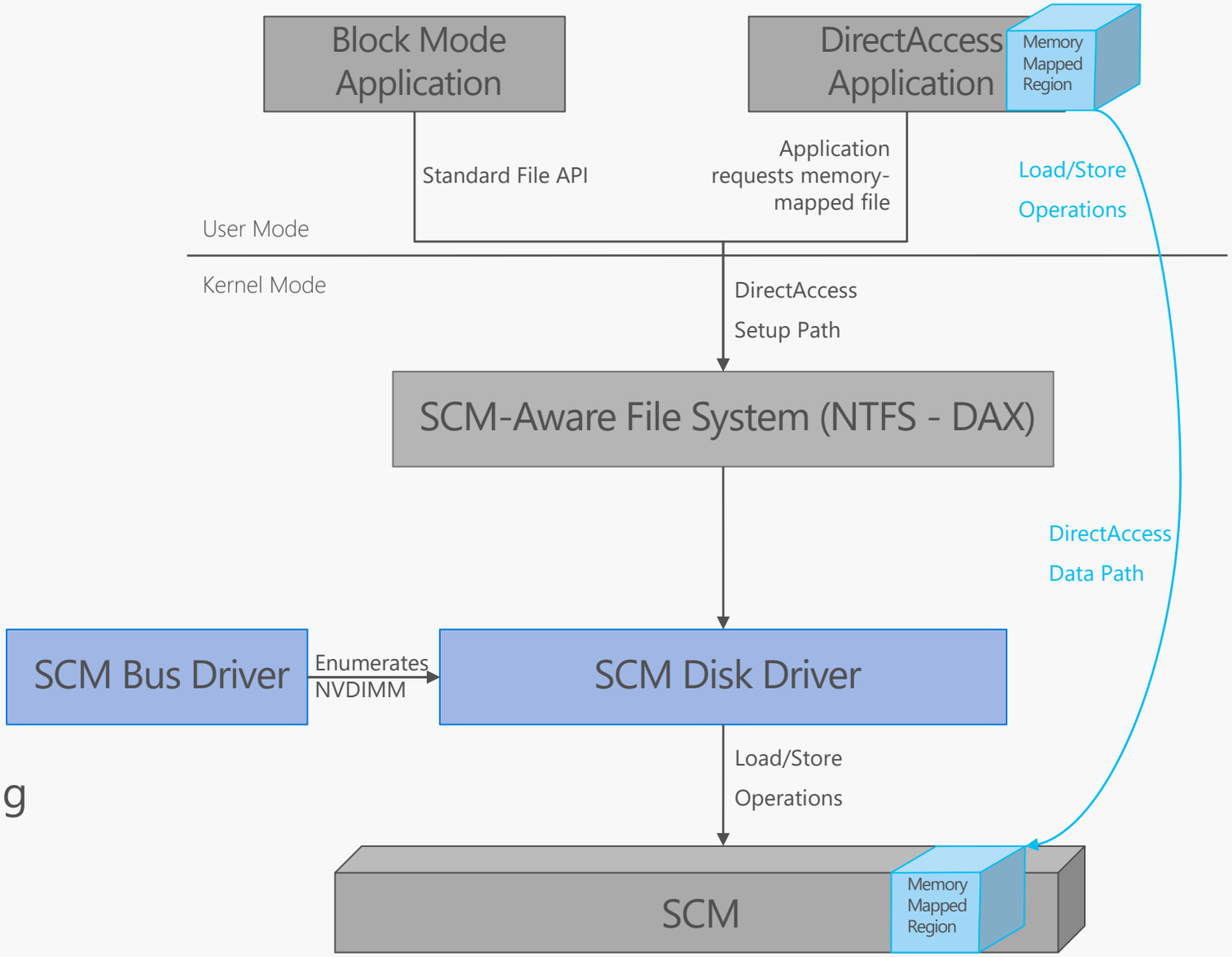
DirectAccess Architecture

Overview

- App has direct access to SCM via existing memory-mapping semantics
- Updates directly modify SCM, Storage Stack not involved
- DAX volumes identified through new flag

Characteristics

- True device performance (no software overhead)
- Byte-Addressable
- Filter Drivers relying on I/O may not work or attach – no I/O, new volume flag
- AV Filters can still operate (Windows Defender already updated)



Using DAX in WS 2016

DAX Volume Creation

→ `Format n: /dax /q`
→ `Format-Volume –DriveLetter n –IsDAX $true`

DAX Volume Identification

Is it a DAX volume?
→ call `GetVolumeInformation("C:\", ...)`
→ check `lpFileSystemFlags` for `FILE_DAX_VOLUME (0x20000000)`

Is the file on a DAX volume?
→ call `GetVolumeInformationByHandleW(hFile, ...)`
→ check `lpFileSystemFlags` for `FILE_DAX_VOLUME (0x20000000)`

Using DAX in WS 2016

Memory Mapping

```
1. HANDLE hMapping = CreateFileMapping(hFile, NULL, PAGE_READWRITE, 0, 0,
   NULL);
2. LPVOID baseAddress = MapViewOfFile(hMapping, FILE_MAP_WRITE, 0, 0, size);
3. memcpy(baseAddress + writeOffset, dataBuffer, ioSize);
4. FlushViewOfFile(baseAddress, 0);
```

OR … use non-temporal instructions for NVDIMM-N devices for better performance

```
1. HANDLE hMapping = CreateFileMapping(hFile, NULL, PAGE_READWRITE, 0, 0,
   NULL);
2. LPVOID baseAddress = MapViewOfFile(hMapping, FILE_MAP_WRITE, 0, 0, size);
3. RtlCopyMemoryNonTemporal(baseAddress + writeOffset, dataBuffer, ioSize );
```

Efficiency Gains with DAX Volumes

Outdated Assumptions

Storage is no longer “slow”
Creating, Tracking, Queueing, Completing I/O costs CPU cycles (context switches)
Past tricks to speed up storage (caching, async I/O, queues), now slow us down

Direct Access

NTFS Volume Mode (Regular vs. DAX) decided at Format Time
No I/O, no queueing, no async reads/writes – just load/stores
Flush user data via existing memory mapping APIs (FlushViewOfFile)
… Or – use non-temporal instructions on NVDIMM-N

Efficiency from DAX

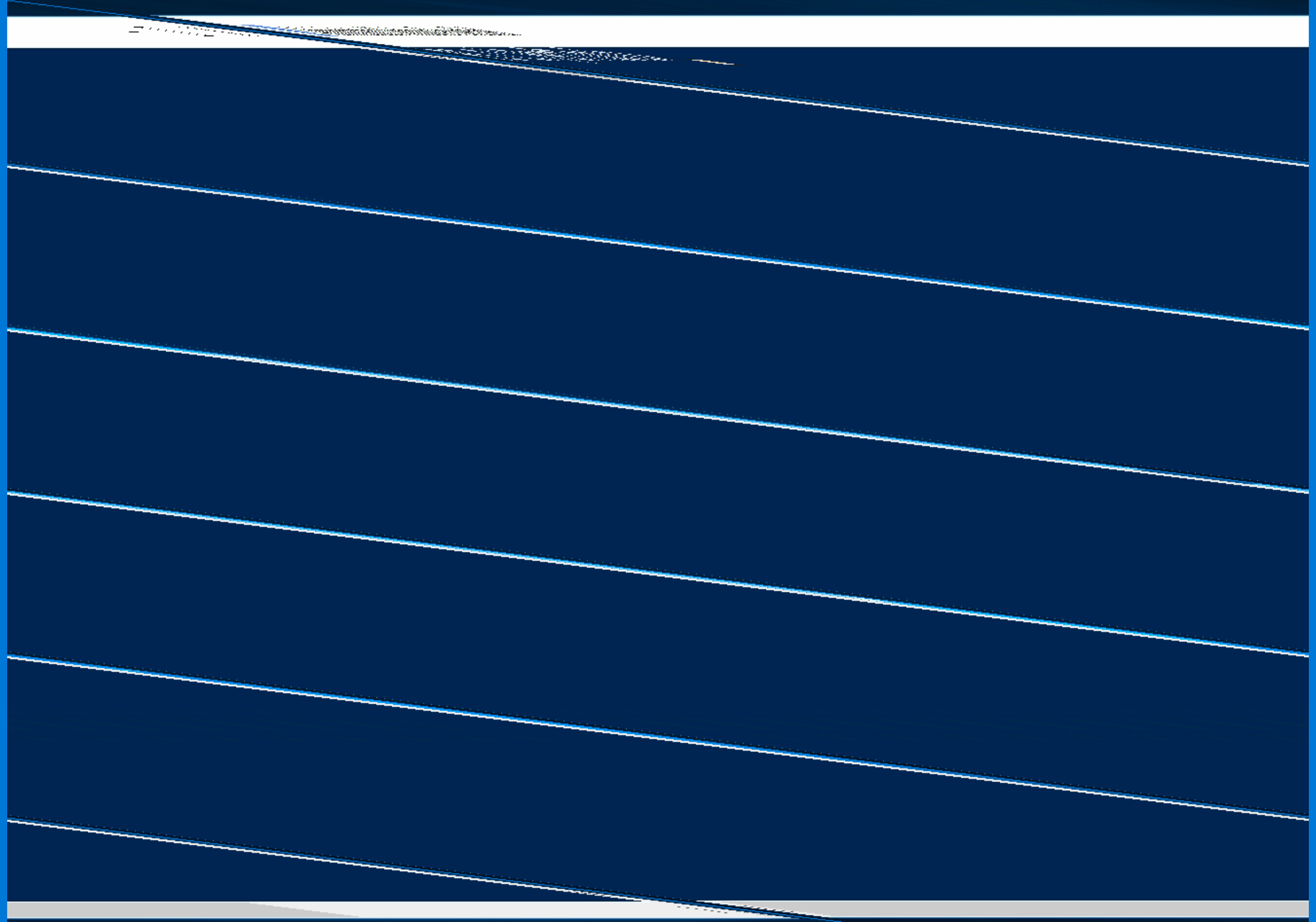
Less context switches on highly contested data structures
Additional CPU cycles for more useful work

NVDIMM-N Block vs. DAX Performance

Tobias Klima

Hardware Provided By:

Hewlett Packard Enterprise



Adapting Apps for DAX – NVML

Overview

- Helper Library making use of SCM easier
- Maintained on GitHub, http://pmem.io/
- Abstracts OS specific dependencies
- Provides: key/value stores, logs, flushing, persistent heap, …

Use Case

Alternative to handling flushing or creating SCM-aware data structures yourself

NVML on Windows

Work underway to port NVML to Windows

Call to Action

- Consider memory-mapped I/O on NVDIMM-N
- How would you change your app, if you could do fast, reliable, synchronous updates?
- What data(structures) do your apps use frequently? Which need the lowest possible access latencies?

Technical Resources

- Memory Mapping - https://msdn.microsoft.com/en-us/library/ms810613.aspx
- GetVolumeInformation - https://msdn.microsoft.com/en-us/library/windows/desktop/aa364993(v=vs.85).aspx
- GetVolumeInformationByHandleW - https://msdn.microsoft.com/en-us/library/windows/desktop/aa964920(v=vs.85).aspx

- Re-visit Build on Channel 9.
- Continue your education at Microsoft Virtual Academy online.

Microsoft