STORAGE INDUSTRY

SUMMIT

The Future of Computing:
The Convergence of Memory
and Storage through
Non-Volatile Memory (NVM)

JANUARY 28, 2014, SAN JOSE, CA

Bob “Beau” Beauchamp

EMC

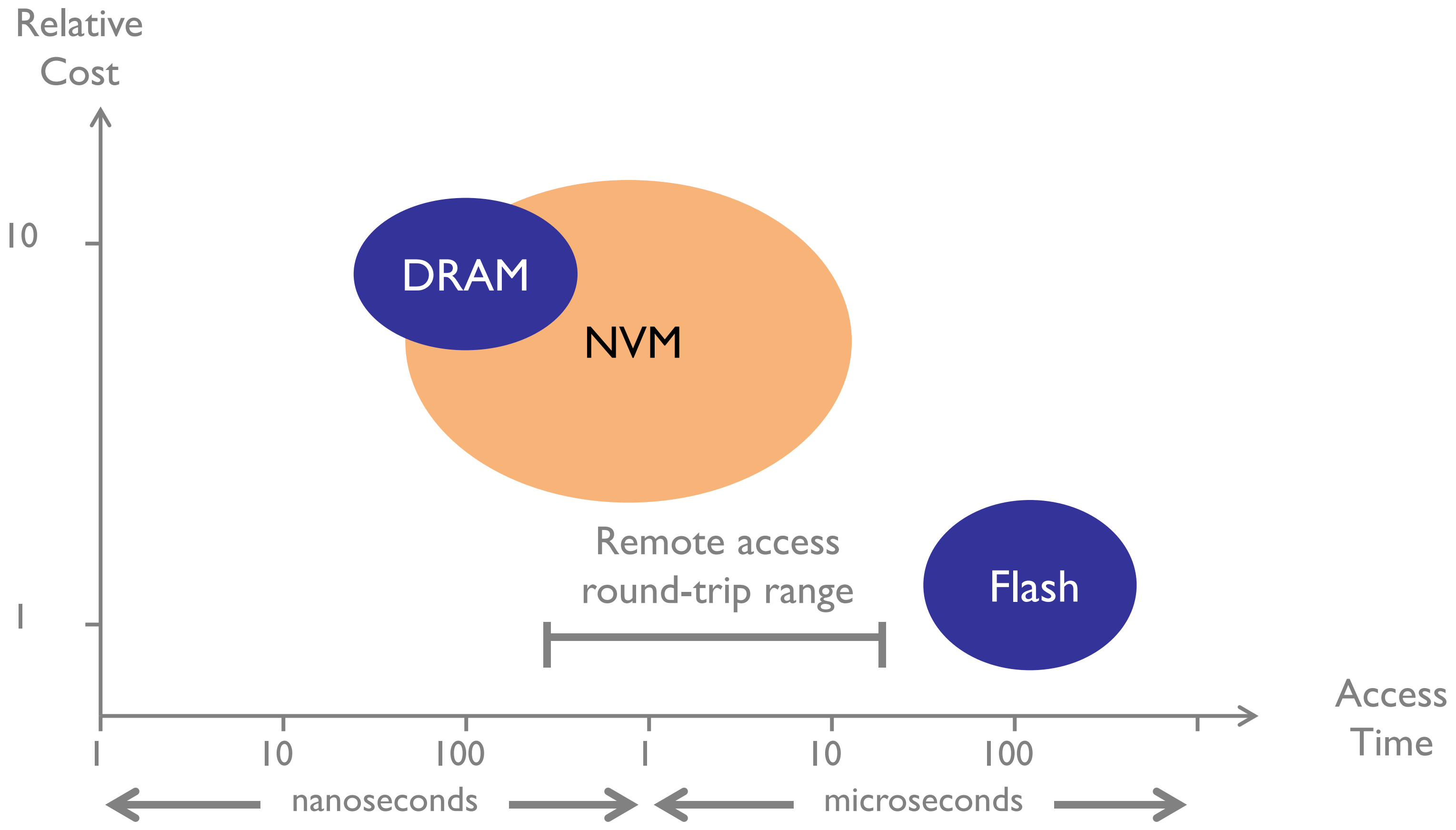
Distinguished Engineer

NVM as Storage

SNIA™

Solid State Storage Initiative

Assumed NVM characteristics



Possible Usage Goals

- Storage systems
  - Simplify implementation
    - Which can improve reliability
  - Improve performance
- Server systems
  - Improve performance and efficiency
    - Reduce access time
    - Scale memory capacity per core

Storage System NVM Usage

- The usual suspects:
  - Metadata space
  - Data Write-commit buffers
  - Data cache and/or storage tier
    - Diminishing return on latency relative to network
- Possible innovations
  - New access methods and/or protocols
    - Are there better ways to exploit the low latency and fine granularity of NVM ?

Need for more Metadata

- Increasing data capacity trend
- Increasing mapping information trend
  - Snap-shots
  - Thin provisioning
  - Fine-grained tiering
  - Additional map layer to manage sharing and allow easy physical mobility
    - Ex: LBA → deduplication-address → Physical address
- Data integrity and consistency checks
- Statistics about data usage

Server NVM Usage

SNIA
Solid State Storage Initiative

- IO Caching
- Complete “in memory” data set
- Larger capacity
  - Potentially higher density and lower $/GB and Watt/GB than DRAM
  - May not want/need non-volatility

NVM’s low latency value is best realized close to the CPU

Requirements for Storage

- Storage must provide that the data is accessible when needed; i.e., durable, available, recoverable, known integrity, etc.
- The degree needed for each dimension above is a per-application risk/reward decision, but it generally requires:
  - Error detection
  - A copy in an independent failure domain before the commit
  - Serially-consistent “control” state at all times

Choosing an NVM model

- What is the “best” model for an application?
  - Load/Store: direct memory mapped
  - Read/Write: Block, File, Object, ...
  - Can/should there be a hybrid of the two?
    - Say, read via Load, update via Write?

- Load/Store has Cache/Uncache options
  - Cache complicates consistent media updates

Intel Memory Types: Cacheable

- Sharing must be in the same coherency domain
  - Software can extend coherency beyond HW domain
- Reads: Low latency and speculative
- Write back
  - Low latency buffered writes
  - Unordered posted media updates
- Write through
  - Less efficient unbuffered writes
  - Ordered media updates

Intel Memory Types: Uncacheable

- No coherency domain that limits sharing
- Uncacheable
  - High read latency; no speculative reads
  - Less efficient unbuffered writes
  - Ordered media updates
- Write Combining
  - High read latency, but can do speculative reads
  - More efficient updates, but weakly ordered
    - No serial consistency guarantee for both CPU and Media
  - Easy to establish consistency points

How Strict for Consistency?

- There may be value in a “consistent” and “non-consistent” space concept, defined as:
  - Consistent “C-space” always transitions from one consistent state to another atomically
  - Non-consistent “NC-space” is allowed to transition non-atomically to eventually arrive at a consistent state
    - It is likely that NC-space variables are qualified by C-space variables; ie, whether the NC-space variable is currently consistent
- Is this a static declaration, or dynamic?
- If dynamic, can it be infered or does it need to be explicit?

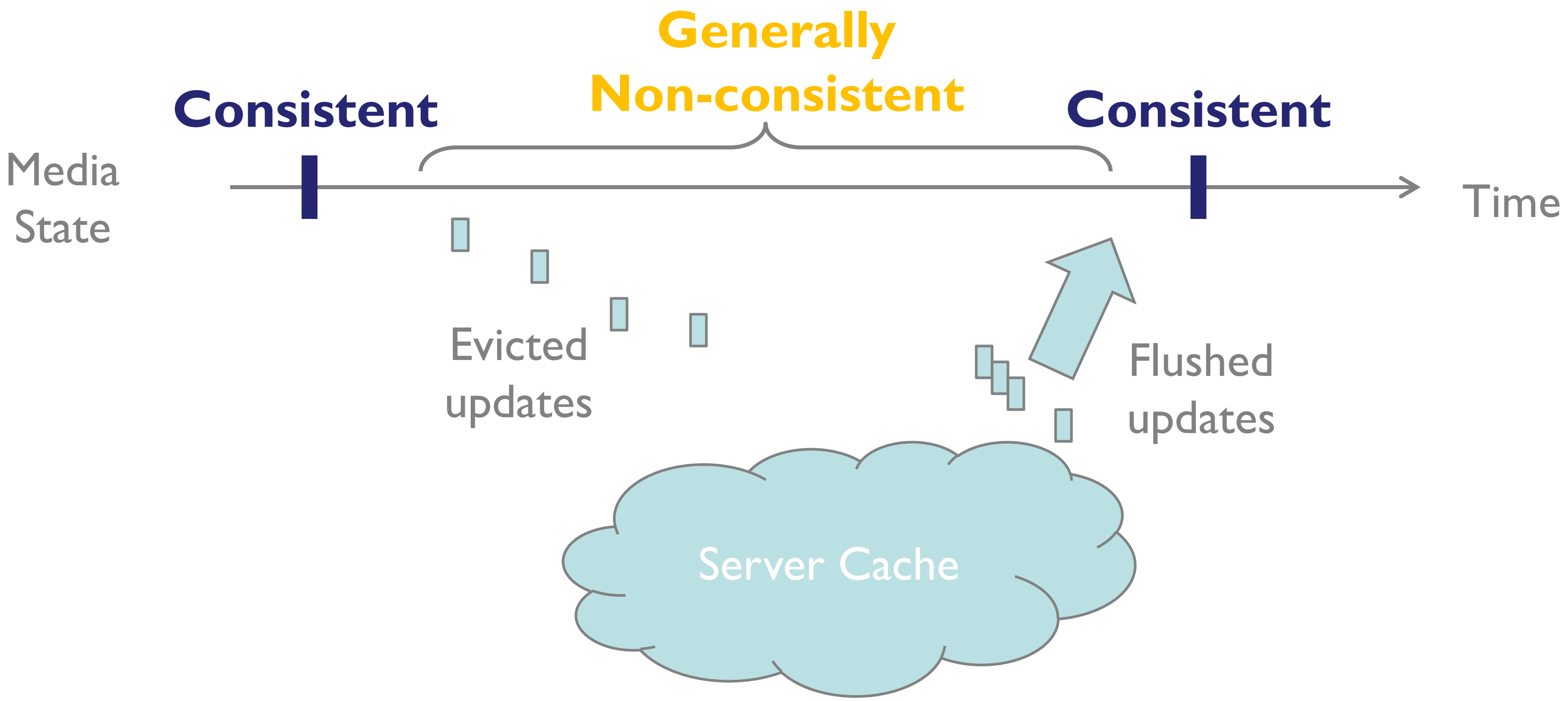
SNIA NVM Programming Model

- App memory-maps a named NVM object into its address space
- A memory-sync operation is used to guarantee updates are flushed to NVM
  - A list of memory ranges may be specified

NVM Programming Model Concerns

- If caching, the state of NVM can only be guaranteed serially consistent at exact completion of the sync
- Non-specific flush is a performance concern
- Specific-location flush is prone to latent bugs; eg, if the list is not complete
  - App may work correctly most of the time
  - Intermittent failures are harder to debug
- Should programmer manage caching type, too?

Caching Impact on Media Consistency



Sync-point consistency

- Database buffer managers have solved the posted-update consistency problem; choose one or both:
  1. Hold off updates in a re-do log to synchronously apply at completion of next checkpoint
  2. Capture before-image of updates into an undo log which can be discarded at completion of next checkpoint

- Performance concerns
  - Expensive to trap Store operations to implement #1
  - Copy-on-write can be used for #2, but likely 4KB images

The “Write” model

- Very familiar paradigm
- Hides the complexity of updating media
  - Can create both undo and redo logs if needed
  - Could provide multi-write-atomic transactions
- Analogous to “specific-range flush” from programmer’s perspective, but will fail consistently if a range is missed
- Provides a SW control point for extensibility
  - Efficiently create remote copies
  - Add integrity checks
  - Tighten write protection
  - Etc.

Futures

- Various research proposing compiler extensions to comprehend and manage NVM. Examples:
  - Mnemosyne: Lightweight Persistent Memory
    - http://pages.cs.wisc.edu/~swift/papers/asplos11_mnemosyne.pdf
  - Durability Semantics for Lock-based Multithreaded Programs
    - https://www.usenix.org/conference/hotpar13/workshop-program/presentation/chakrabarti
- Compiler can hide NVM media update complexity
  - Compiler solutions can use both Write or Store mode as appropriate for level of consistency needed

Summary

- NVM storage is not as simple as just having non-volatile memory
- There is a rich set of implementation trade-offs for the industry to evaluate
  - Load/Store versus Read/Write
  - Caching type attributes
- There is opportunity for HW and SW innovation
  - New memory systems, write stacks, compilers, ...
- It is fun to live in interesting times !!!