SNIA

Education

Storage tiering and the impact of flash on File Systems

Jonathan Goldick, LSI

SNIA Legal Notice

- The material contained in this tutorial is copyrighted by the SNIA.
- Member companies and individual members may use this material in presentations and literature under the following conditions:
  - Any slide or slides used must be reproduced in their entirety without modification
  - The SNIA must be acknowledged as the source of any material used in the body of any document containing material from these presentations.
- This presentation is a project of the SNIA Education Committee.
- Neither the author nor the presenter is an attorney and nothing in this presentation is intended to be, or should be construed as legal advice or an opinion of counsel. If you need legal advice or a legal opinion please contact your attorney.
- The information presented herein represents the author's personal opinion and current understanding of the relevant issues involved. The author, the presenter, and the SNIA do not assume any responsibility or liability for damages arising out of any reliance on or use of this information.

NO WARRANTIES, EXPRESS OR IMPLIED. USE AT YOUR OWN RISK.

Abstract

- Storage Tiering and the Impact of Flash on File Systems
  - With all of the talk about how storage systems will be impacted by large amounts of relatively inexpensive flash little has been said about how file systems will need to change to take advantage of it. This tutorial will cover how file systems are evolving tiered architectures to leverage flash. There will also be a discussion contrasting flash deployed as part of a storage array vs. a component of the file server.

Agenda

- Basic Characteristics of File Systems
- Basic Characteristics of Flash
- Storage Tiering with Flash
- Array-Based Tiering
- File System-Based Tiering
- Changing Our Assumptions
- File Systems With Dynamic Tiering
- Some Final Observations
- Questions

Basic Characteristics of File Systems

- User data can have any imaginable access pattern
  - The applications define the I/O pattern and file systems must cope as best they can to predict future I/O(s) from past behavior.
  - Most user data gets cold, i.e. read-only and read-rarely
  - User data hangs around indefinitely, who deletes files anymore?
  - Very large and growing
- Metadata tends to be dominated by very small, random I/O(s)
  - Inodes, directories, block allocation maps, etc.
  - High modification rate, at least for a while.
  - Most metadata gets cold rapidly and hangs around indefinitely.
  - Small percentage of storage used even in a small file environment.
- File system logs/journals are small sequential write dominated
  - Only read after a crash and then it's typically large sequential reads where time is of the essence.
  - Log write latency is critical to file system performance.
  - Blocks get recycled and rewritten rapidly.
  - Logs are relatively small, rarely more than a few gigabytes

Basic Characteristics of Flash

- Large advantage for random reads
  - IOPS are what matters here.
  - IOPS/$ for MLC flash is about 800 and for disk is around 1.
  - IOPS/Watt for MLC flash is about 40 and for disk is at or below 0.05.
- Large latency advantage
  - Time to read the first block is what matters here.
  - MLC flash is roughly 70usec while SATA disk is about 10000usec and 15K 3.5" disks are about 5000usec.
- Smaller advantage for sequential operations
  - Bandwidth is what matters here.
  - MB/sec/$ for MLC flash is about 3 and for SATA disk is about 1.
  - MB/sec/mWatt for MLC flash is about 0.1 and for SATA disk is about 0.01.

Basic Characteristics of Flash

- SATA disk is far superior for bulk storage
  - Cost for GB is what matters here.
  - GB/$ for MLC flash is about 1 and for SATA disk is about 10. Note that 15K RPC 3.5" drives are about the same as MLC.
  - MB/mWatt for MLC flash is about 50 and for SATA disk is about 100.
- Reliability differs significantly from disks.
  - Media rated for a maximum number of write-erase cycles. Addressed by write leveling to the extent possible but a high intensity write load will "use up" flash.
  - Shelf life built into reliability model. The media capacity shrinks as it sits on a shelf unused.
  - Data degrades over time even when inactive or read-only.
  - One wouldn't leave readonly data on flash indefinitely because it is subject to decay. Disk aerobics can keep rewriting decayed blocks but that also uses up lifetime.

Basic Characteristics of Flash

Check out SNIA Tutorial:

The Benefits of Solid State in Enterprise Storage Systems – Today and Tomorrow

Check out SNIA Tutorial:

Overview and Current Topics in Solid State Storage

Check out SNIA Tutorial:

Solid Business, Shifting Vision: What Users Can Realistically Expect from SSS in the Next 2 Years

Storage Tiering With Flash

- DRAM Tier
  - Fastest possible read cache but small, expensive and power hungry.
  - Write cache only when battery backed.
  - Flash tier allows for rapid copy during a power loss. Battery requirements per gigabyte of DRAM write cache directed related to copy speed.
- Flash Tier
  - Tier of choice for random access, high bandwidth requirements, and high IOPS data.
  - Often treated as a read cache with writes mirrored to a disk tier.
  - RAID-10 common when used as a read/write cache.
  - Other non-volatile storage can be substituted here, see the SSD tutorials.
- Widely-Striped Inexpensive Disk Tier
  - Large, sequentially read and written data.
  - Long term storage of cold metadata and data.
  - Slow spinning disk is fine if flash tier is in front. Less power.
  - Flash tier can even improve performance of disk tier by making it write-mostly and sequential access dominated.

Array-Based Tiering

- PROS
  - Arrays can readily take advantage of a very large read cache.
  - Logical extension of existing battery-backed DRAM tier to make a flash-based write cache.
    - Moves to a multi-level tiering model.
  - Transparent to hosts so broadly applicable.
  - Dynamic tiering avoids admins having to create pools of flash-based LUN(s) and create file systems for each class of storage.

Array-Based Tiering

- CONS
  - Arrays do not know what data blocks contain.
    - Initial placement is going to be on the highest performance tier with a waterfall over time.
    - Future I/O pattern less predictable at write time, which impacts optimal tier choice.
    - No standard for providing class of service hints when writing blocks.
  - Array doesn't know when a block is deallocated.
    - Unnecessary migration to disk for unused data.
    - Wasted wear leveling work.
  - Controller is the new bottleneck.
    - Controllers run out of CPU long before flash drives run out of IOPS.

File System-Based Tiering

- PROS
  - File system knows what blocks contain, metadata vs. data vs. log.
    - Can make informed decisions on initial placement.
  - Well established process for identifying cold data.
  - Knows when a block is deallocated.
    - Opens possibility to inform wear leveling facility to make it more efficient an extend flash drive lifetime.
    - Standards groups are working on it.
  - Potentially moves flash closer to application.
    - Less CPU overhead to access PCI-Express attached flash than through an array.
    - Removes array controller as the bottleneck on IOPS.

File System-Based Tiering

- CONS
  - File systems were not designed for large amounts of non-volatile cache. Think hundreds of Gigabytes to Terabytes.
  - Battery-backed DRAM has a different operational model from flash.
    - Generally implemented as a tier to be copied to disk ASAP because they are relatively small and battery constrained.
    - Treated more like a log for most recently written data rather than a large, high performance, random access read/write cache.
    - Must often be replicated to a failover partner for high availability architectures.
  - Few file systems have a dynamic tiering model for stable storage.
    - HSM models are common but are complex and lose much of the performance gains. They also often assume large files amortize their own overhead.
    - A transparent model requires an understanding that not every block has the same performance, not a typical file system capability.

Changing Our Assumptions

- Random I/O is no longer worth avoiding
  - File Systems try hard to avoid random access and use a lot of CPU, memory, and complex code to achieve it.
  - File System logs were created to maintain crash consistency when a single operation required multiple random write I/O(s). The bigger the log, the longer it takes for a file system to come online after a crash, the smaller it is the greater the random write I/O(s). If we are now free to immediately issue the random I/O(s) after the transaction has been logged, then we need at most a few kilobyte log file. We can even consider ordering the I/O(s) so that no log is needed at all and thereby eliminate a very complex system.
  - Inode and block allocators have a lot of logic to keep locality for objects likely to be accessed together in time.
  - Defragmenters exist for the sole purpose of keeping files contiguous on disk to avoid future seeks should they be read.
  - These algorithms are of little value if the underlying storage is flash.

Changing Assumptions

- Massive increase in available IOPS
  - File Systems work hard to minimize the number of disk I/O(s) for each operation.
  - File Systems, or their underlying storage layers, sort the pending reads and writes to coalesce them into the largest possible contiguous chunks. The goal is to reduce the IOPS required but this requires complex background writer logic that could be avoided.
  - While IOPS are not infinite with flash, this work costs a lot in terms of CPU and memory for what is no longer a scarce resource.

Changing Assumptions

- Read I/O latency has dropped dramatically
  - File Systems do a lot of read ahead work to hide I/O latency.
  - The hope is that future reads can be predicted accurately and occur before the DRAM is recycled. This breaks down in many workloads but has always been worth the cost for the times it does work because disk latency is about 50,000 to 100,000 times slower than DRAM depending on the drive speed.
  - Given that flash latency is only about 800 times worse than DRAM this may not be CPU and DRAM well spent. Remember that anything new brought into DRAM ejects something else that could been needed.

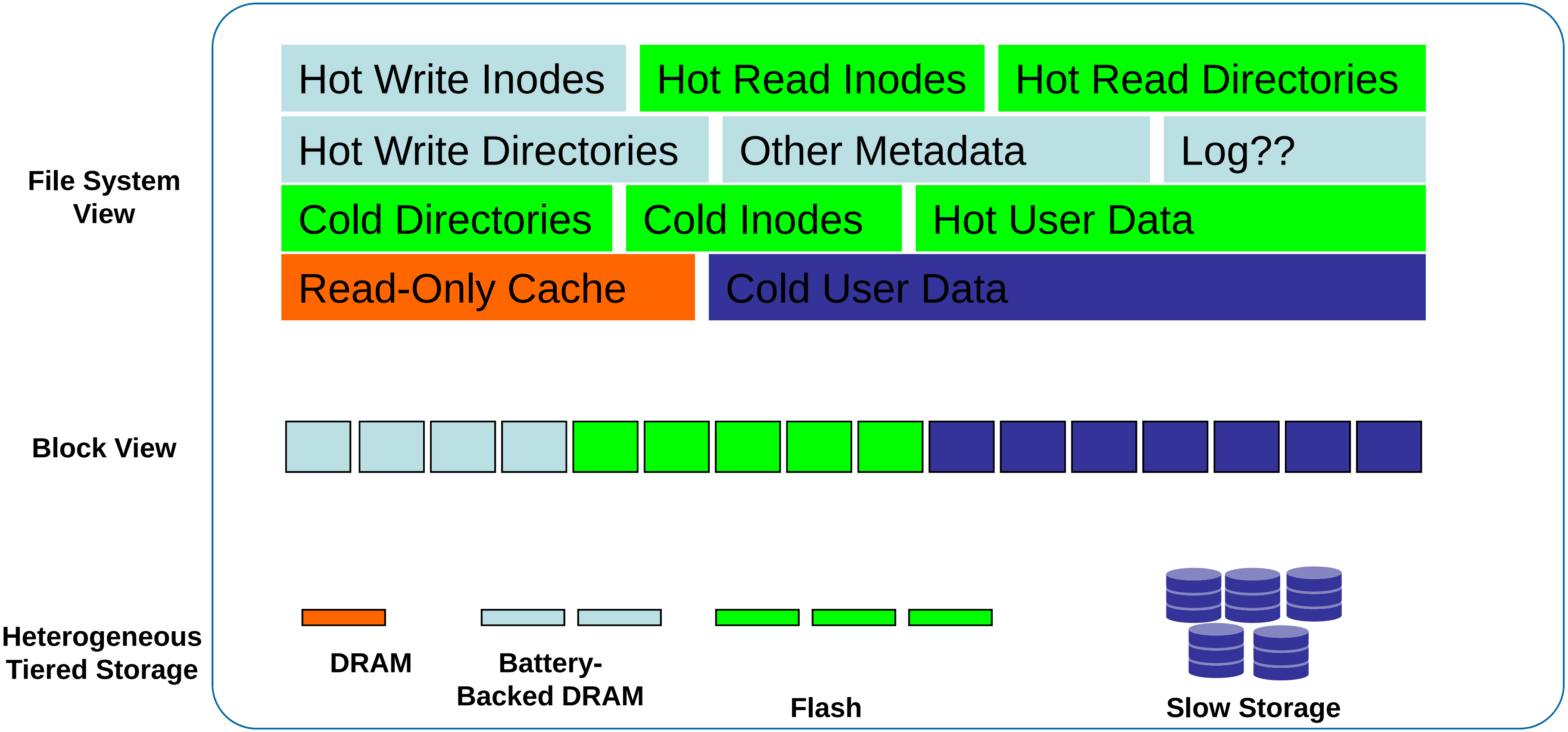
Changing Assumptions

- Not every block is the same.
  - File Systems have been built on the assumption that every block has the same performance and reliability characteristics.
  - File Systems need to be aware of the class of service offered by the available tiers of storage and direct I/O(s) to the best choice.
  - This information needs to be exposed to a much higher layer of the system than is typically done today.
  - The best block choice for a piece of data can change as it ages.

Changing Assumptions

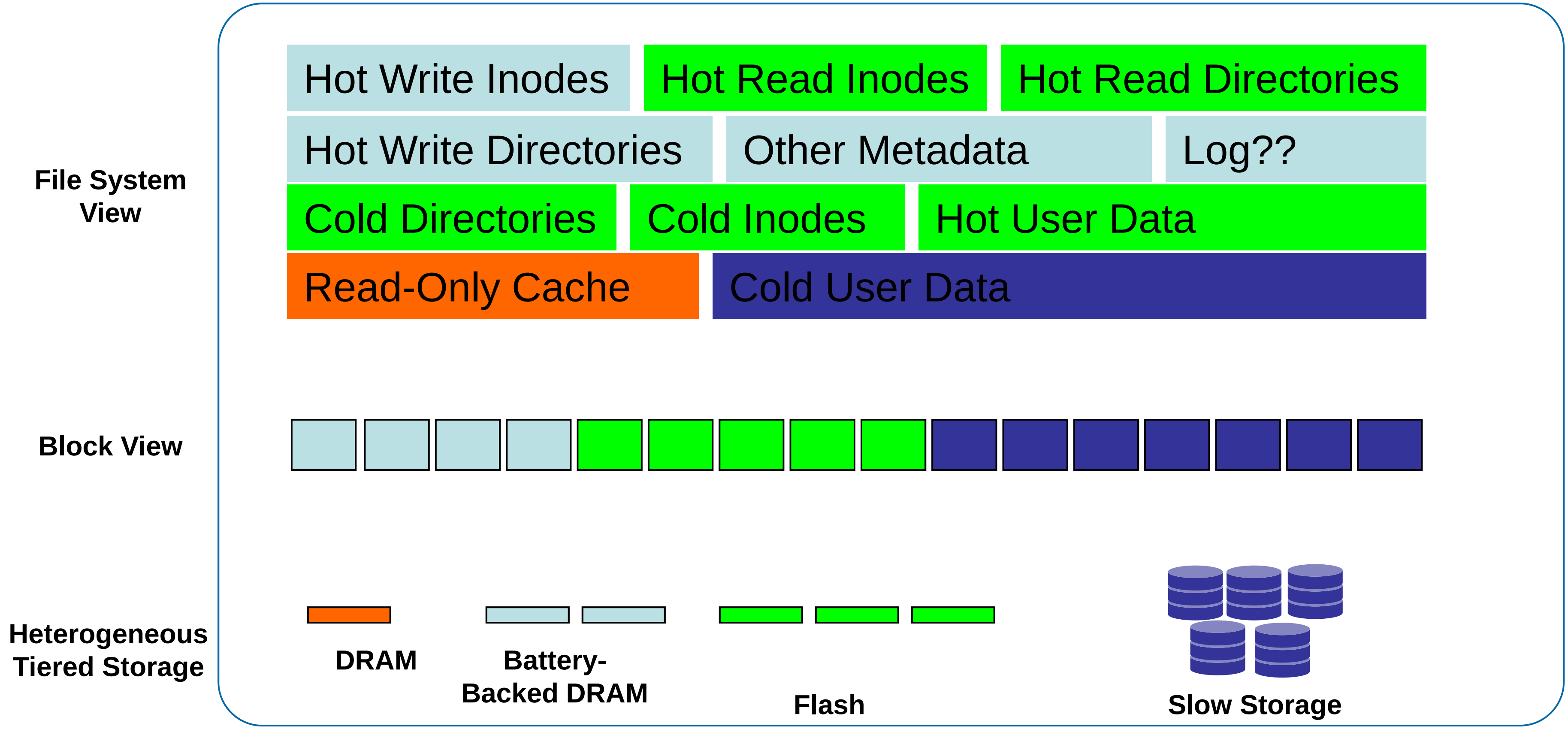
- File server CPU is the new scarce resource.
  - File Systems tradeoff CPU on the assumption of very long I/O times. If a read I/O takes roughly 5M processor cycles to complete it's worth a lot of logic and L2/DRAM cache access to optimize it away. When I/O latency is reduced to 20K processor cycles with flash this is no longer the clear win it was previously.
  - As array vendors adding flash shelves have discovered, we run out of CPU long before we run out of IOPS.
  - Fundamentally the software stacks need to be thinned out substantially.

File System With Dynamic Tiering



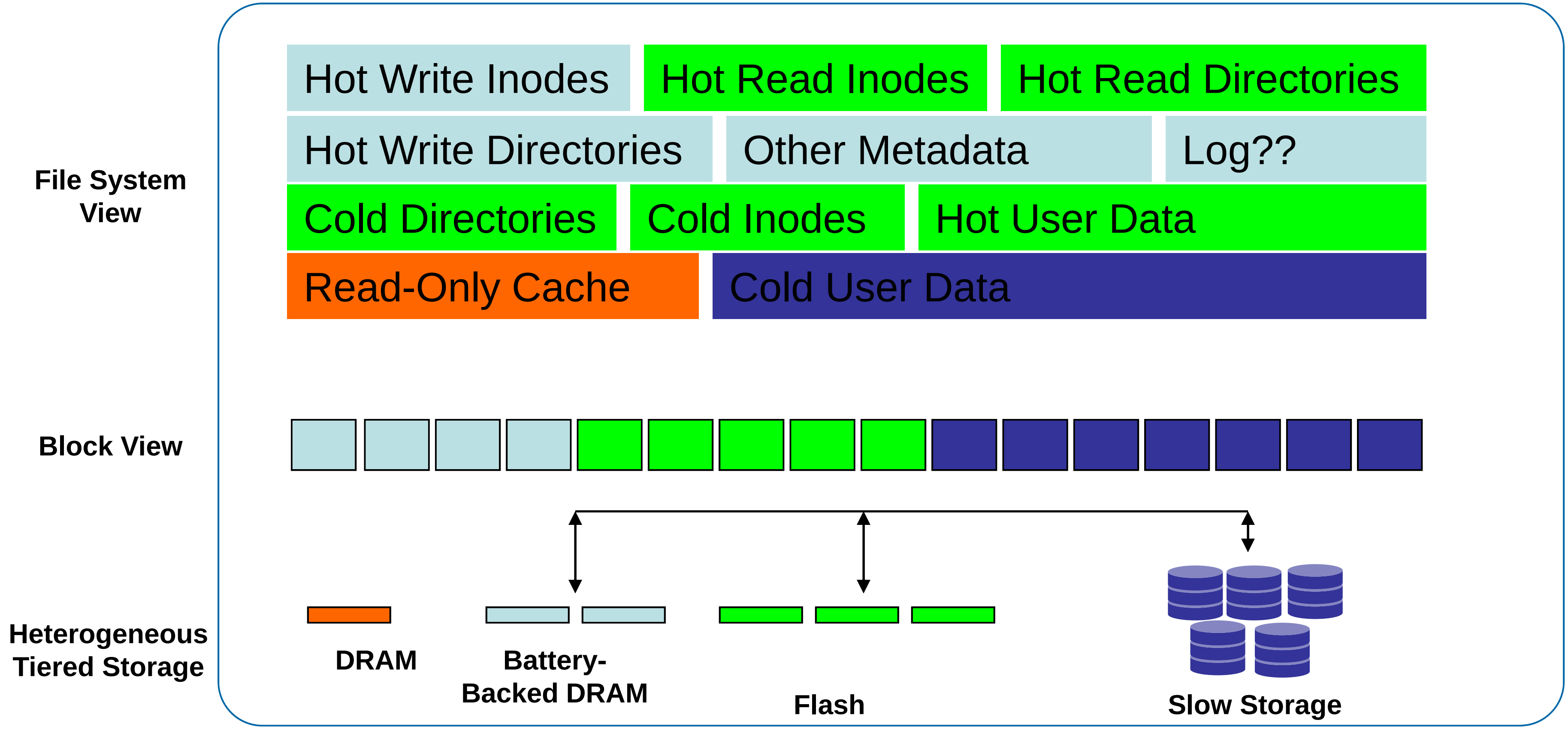
- A single file system can span many tiers of storage
- The slow storage tier is strictly about density and long term retention. Could even be powered up only on access (MAID). May not even be needed at all for some workloads.
- Read cache will duplicate blocks from flash and disk tiers but not battery-backed DRAM tier.

File System With Dynamic Tiering



- Place even cold metadata on flash to avoid worrying about latency hiding during file system checks, backup, file system scans, etc.
- As flash latency gets better over time or replaced by something like PCM we should avoid duplicating blocks between the read-only DRAM cache and the low latency tier.

File System With Dynamic Tiering



- Migration between tiers is a block relocation that updates the parent inode.
- Similar to what defragmenters do but with a different goal than compaction.
- Migration at this level doesn't affect file names and is snapshot-friendly.

Some Final Observations

- File systems can be made much less complex if they assume large amounts of storage with memory-like performance.

- A great deal of file system software technology becomes obsolete in the presence of flash.

- We are likely to see baby steps in the evolution of file systems as vendors are constrained to support configurations with little or no flash. More likely to see flash-oriented fast paths in software rather than dramatic new implementations until flash is universal.

- File systems will experience enormous increases in performance in the next couple of years, especially for random workloads.

Some Final Observations

- Array-based dynamic tiering makes it harder for file systems to leverage the promise of flash.
  - Read latency is unpredictable.
  - File systems cannot assume that random I/O will be predictably inexpensive.
  - No existing API for SCSI commands to provide arrays with information required to make intelligent placement decisions so they would be likely to use a standard waterfall approach.
  - No existing API for SCSI commands to move blocks across tiers based on file system knowledge. Arrays will use LRU rather than knowledge of what is and is not latency sensitive for future reads.

Q&A / Feedback

- Please send any questions or comments on this presentation to SNIA: trackfilemgmt@snia.org

Many thanks to the following individuals for their contributions to this tutorial.

- SNIA Education Committee

David Dale
Jonathan Goldick
Robert Ober
Bret Weber