Software-Defined Flash: TradeOffs of FTL, Open-Channel, and Cooperative Flash Management

Craig Robertson
Software Architect
Radian Memory Systems, Inc.

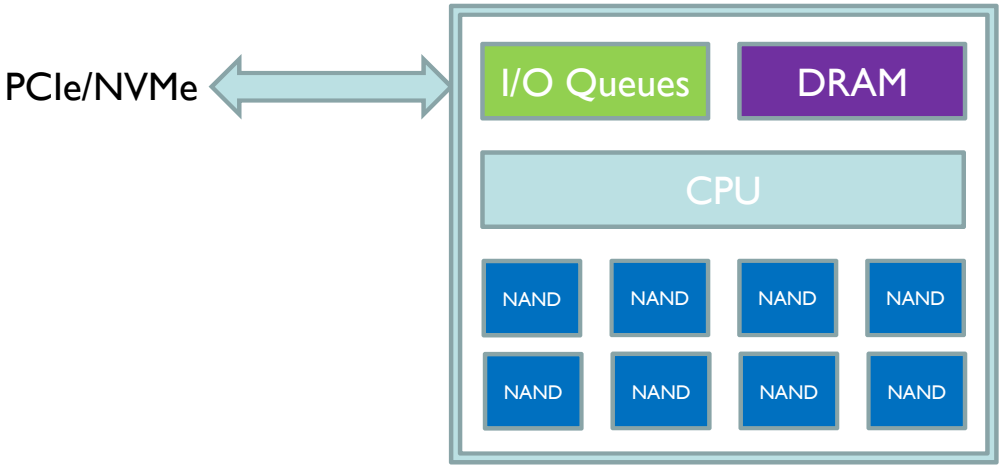
Intro

- Radian makes cutting edge:
  - NVMe Cooperative Flash Management SSDs
  - NVMe Open-Channel SSDs
  - NVMe NVRAM cards

Overview – what this talk will cover:

- FTL – review
  - NAND: how is it different than HDD?
- Current problems with SSDs
  - Latency
  - Endurance
- New SSD Architectures:
  - Open-Channel
  - Radian Symphonic Cooperative Flash Management (CFM)

Background

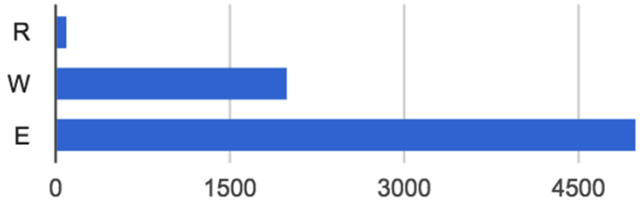


- What is an SSD?
  - NAND flash
  - A controller/CPU
  - Some DRAM
  - FTL: software on CPU
- Why (is it the way it is)?
  - Make NAND device fit HDD (block) API without change.
  - NAND properties

NAND – How is raw NAND not like HDD?

Radian MEMORY SYSTEMS

- PAGE: Must write in units of 16kB (or 32kB… or 64kB)…
  - Must write pages in a block sequentially…
- ERASE: Cannot overwrite a page (16kB) before erasing block it's part of:
  - Erase Blocks can be 4MB or larger!
  - Kernel wants to (over-) write in 4kB or 8kB blocks…
- NAND has non-uniform read/write/erase time
  - Read 100us
  - Write 1500us – 2500us
  - Erase 5000us – 15000us



NAND properties…

- Limited erase count: 10,000 or 3,000 or 1,000 cycles
  - Then data failure…
- Wear Leveling: maintain uniform use of erase count per block.
- Data Retention: must move/refresh data every 30 or 60 or 90 days (or fewer!)
- And many more-complicated details…
- SO: need a logical-to-physical (L2P) mapping table… need to be able to map/move pages around.
- Need a new process: “garbage collection” to reclaim space.

Write Amplification, Garbage Collection

- "Extra" writes not called for by the app.
- Measured as a multiplier: 1.5x, 2x, 5x, etc.
- Must move old/live data to a new block to reclaim stale pages.
- "Overwrites" cause old pages to be stale… dead space.
- This process is called "garbage-collection".
- Write pages 1, 2, 3, 4; then overwrite pages 2 and 4:

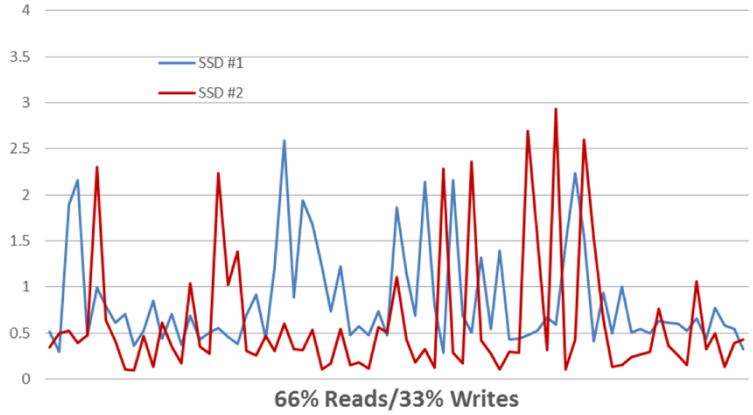
- Unless ALL pages in block are stale, creates write-amplification.

The FTL

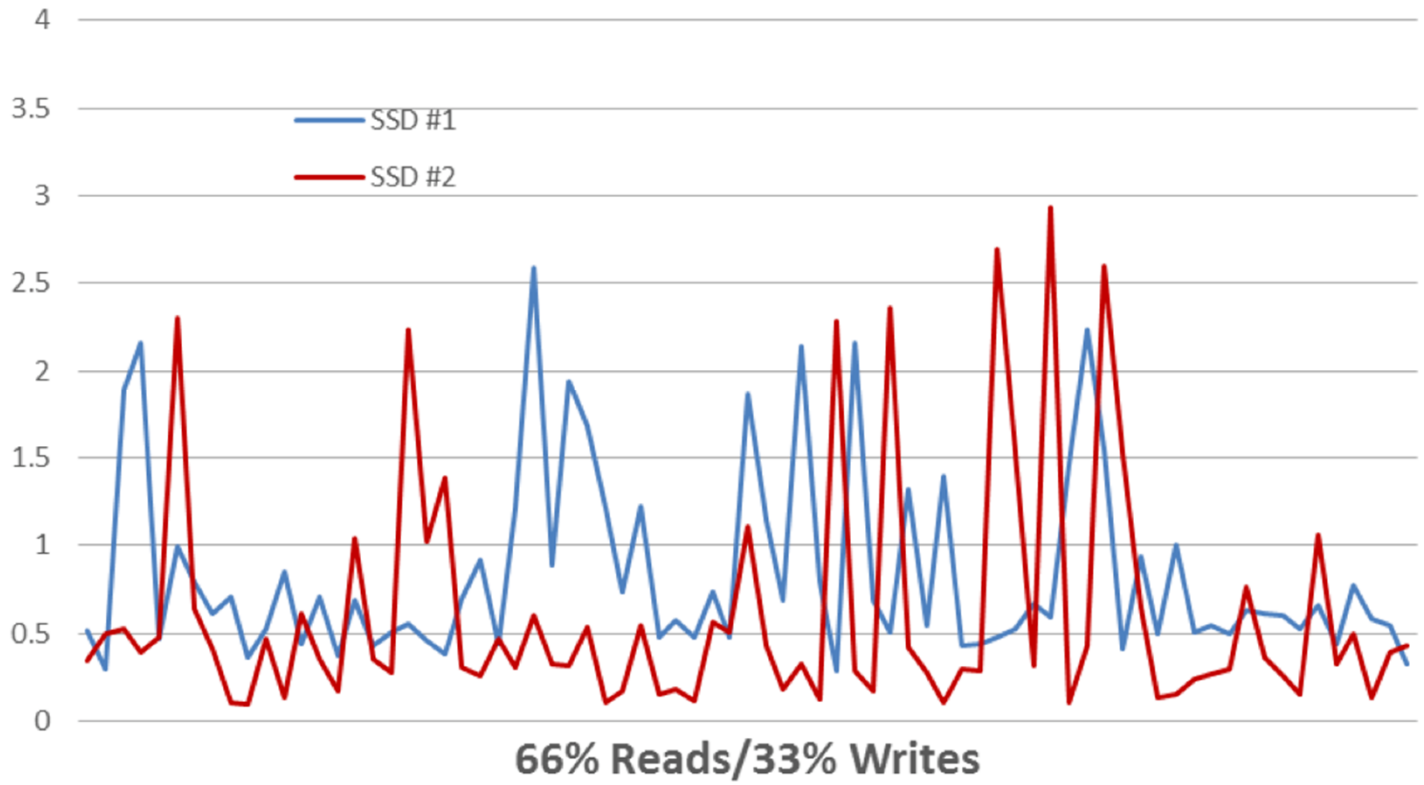
- "Flash Translation Layer"
  - Firmware in SSD to make NAND work with block device API
  - … without exposing explicit ERASE command
  - … takes care of "Garbage Collection"… moving live data so some blocks can be erased
  - … takes care of "Wear Leveling"
  - … causes "Write Amplification" which lowers endurance
  - … all WITHOUT CONSENT of operating system or user.

SSD Problems – read latency spikes

- Users discovered that their SSD's had latency spikes:
- So, they attempted to fix problem by reverse-engineering drive "logic":
  - IF app writes to drive
    - in LARGE, CONSECUTIVE stripes
    - Continuously
    - End-to-End
    - it appears that drive exhibits fewer latency spikes.
  - This is *guessing*…



Read Latency Spikes



Latency spike workaround?

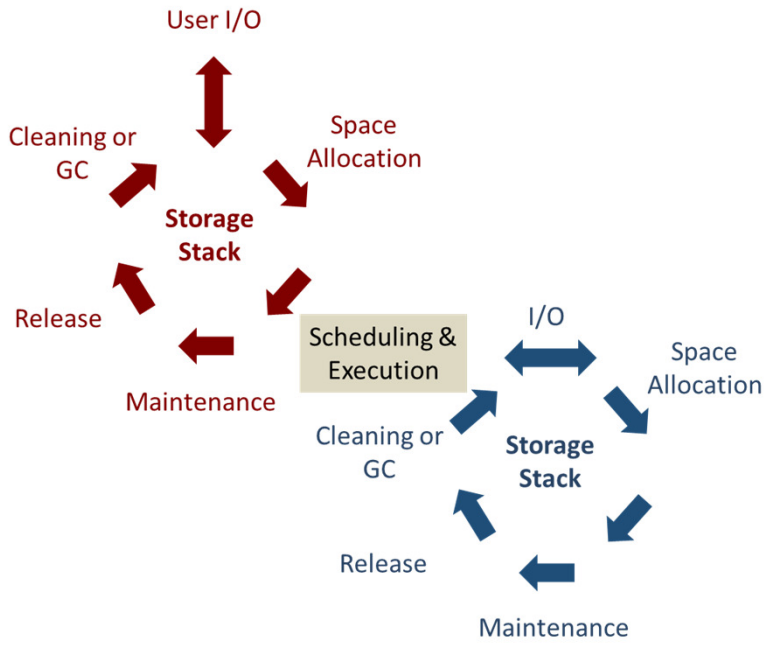
- Large write-stripe difficulties:
  - Only works for single-writer applications
  - Drive partitions probably don't work
  - Multi-threaded apps probably don't work
  - VM's probably don't work
  - No guarantee of behavior:
    - Across generations of device…
    - Or even across firmware updates!

What's wrong with 'Flash-as-Tape'

- 'Flash-as-Tape': writing large stripes, perhaps 1GB sequentially across drive
- No overwrites UNTIL each 1GB stripe can be fully rewritten
- Goal: induce erase-only GC in drive
- Large stripe cleaning causes excessive live-data copying
  - Must relocate even 100% full EUs to clean stripe
- Read latency is worse because write frontier is so large
- Keep more free space to accommodate data relocation

SSD Problems – Log on Log

- Both file systems and FTLs are maintaining their own:
  - L2P mapping tables
  - Allocation/reclamation algorithms
  - Locks
  - WITHOUT communicating
- Causes
  - Extra latency
  - Extra write-amplification



Storage Intelligence

- An attempt to fix the block API for SSD's
- SCSI T10 standard (and soon, part of NVMe)
  - Multistream writes…
    - Identify data with "similar lifetimes" by small handful of streamIDs
    - Intended to help reduce write-amplification
  - ABO: enable/disable "advanced background operations"
    - User can disable (!) GC, wear-leveling, data-retention…
    - Or run it in units of 100ms quanta

Storage Intelligence – Difficulties

- No agreement on how this should be integrated w/ kernel
- No agreement on how/if app participates in streamID choice
- No agreement on app API to indicate stream
- No agreement on who/which-app is auth'd to start/stop GC
- So: no way (yet) to be assured fine-grained benefit…

StreamID unknowns

- Bandwidth requirement of stream
- Size/quantity of objects in stream
- Hot write vs hot read not separated
- Helps at lowering WA, but doesn't solve it
- Drive has difficult time managing allocation of streams to underlying storage, once streams start filling up

Storage Intelligence - Observations

- The multistream feature is useful to enable physical grouping of "similar treatment" data
  - Can mitigate Write-Amplification
    - Data with similar lifetime stored in same blocks
    - So: blocks will be mostly empty when GC kicks in
- This does not help against latency spikes

New SSD Architectures – Alternatives

- Legacy file systems
  - Overwriting
- Newer file systems:
  - CoW (copy-on-write)
  - Are latency-sensitive
  - Are concerned about managing endurance
  - Want to be in total control…
  - -> aligned/compatible w/ Open-Channel and Cooperative Flash Management

Host Managed Flash

- First step at new SSD architecture:
- Move FTL operations to the host:
  - Overwrite management
  - GC – where, when
- Responsibility (and power) of using raw Flash is in host control

Host Managed Flash: Affinity

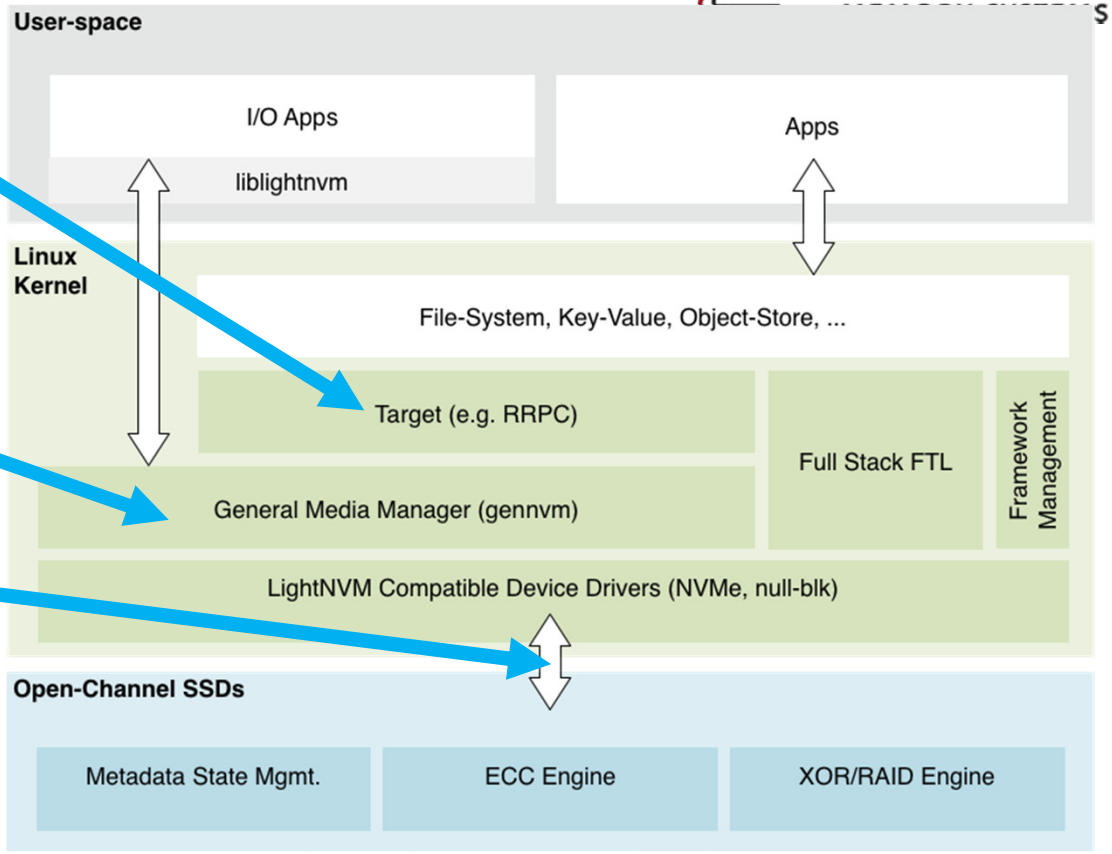
- Host Managed Flash works great for
  - Caches
    - GC can avoid copying “live” data - force cache miss instead
    - So, GC becomes “ERASE”
  - Pure CoW file systems
    - L2P table is already in the filesystem
    - GC still needs to move live data

Open-Channel

- Small API extension to NVMe (called “Open-Channel API”)
- Linux kernel components (“lightnvm”):
  - Media-manager: allocates/frees/erases blocks
  - Target: provides optional overwrite-in-place functionality (custom FTL)
- Targets can be tuned per app.
- Flash wear-leveling and DR may be done in media-mgr…
  - But this may involve proprietary information from NAND vendor which might prevent this code in the kernel
- Layers (mm, target) will likely need to change with each NAND product cycle…

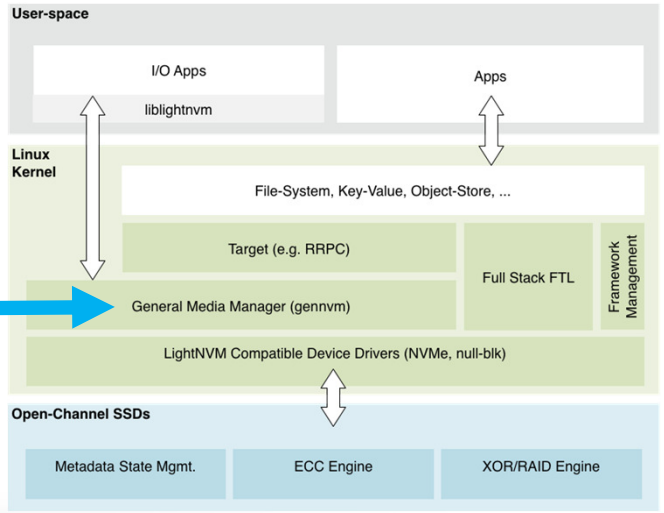
Open-Channel -- Components

- Target:
  - Host-side kernel FTL
  - Allows legacy kernel file system operation
- Media-Manager:
  - GET/PUT BLOCKS
  - (implied ERASE)
- Open-Channel API:
  - IDENTIFY
  - GET/SET BADBLOCK
  - PWRITE(PPALIST, DATA)
  - DATA = PREAD(PPALIST)
  - PERASE



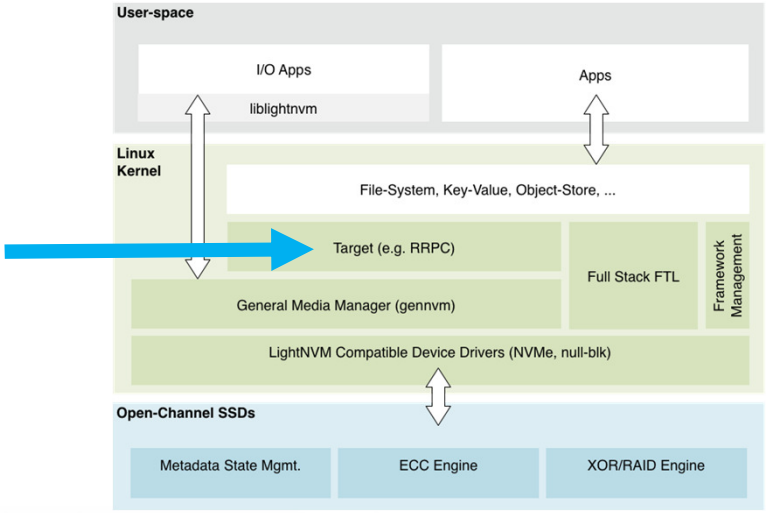
Open-Channel – media manager

- Allocate/free mechanism for accessing BLOCKS from underlying Open-Channel device
  - A BLOCK is a collection of physical pages and is likely 4MB or bigger
- Block return triggers erase
  - … scheduled by media-mgr
- Radian implements the NAND maintenance operations on the drive



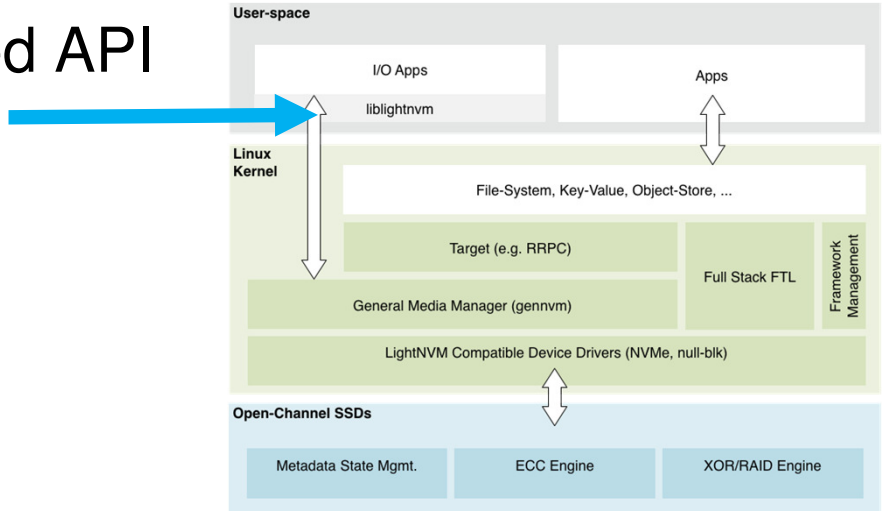
Open-Channel – pblk target

- Lightnvm kernel module provides host-side FTL
  - L2P mapping
  - GC as detected by LBA overwrite
- Exposes a block device that allows overwriting
- Status:
  - In development, not yet upstream
  - Core in 4.6



Open-Channel – liblightnvm

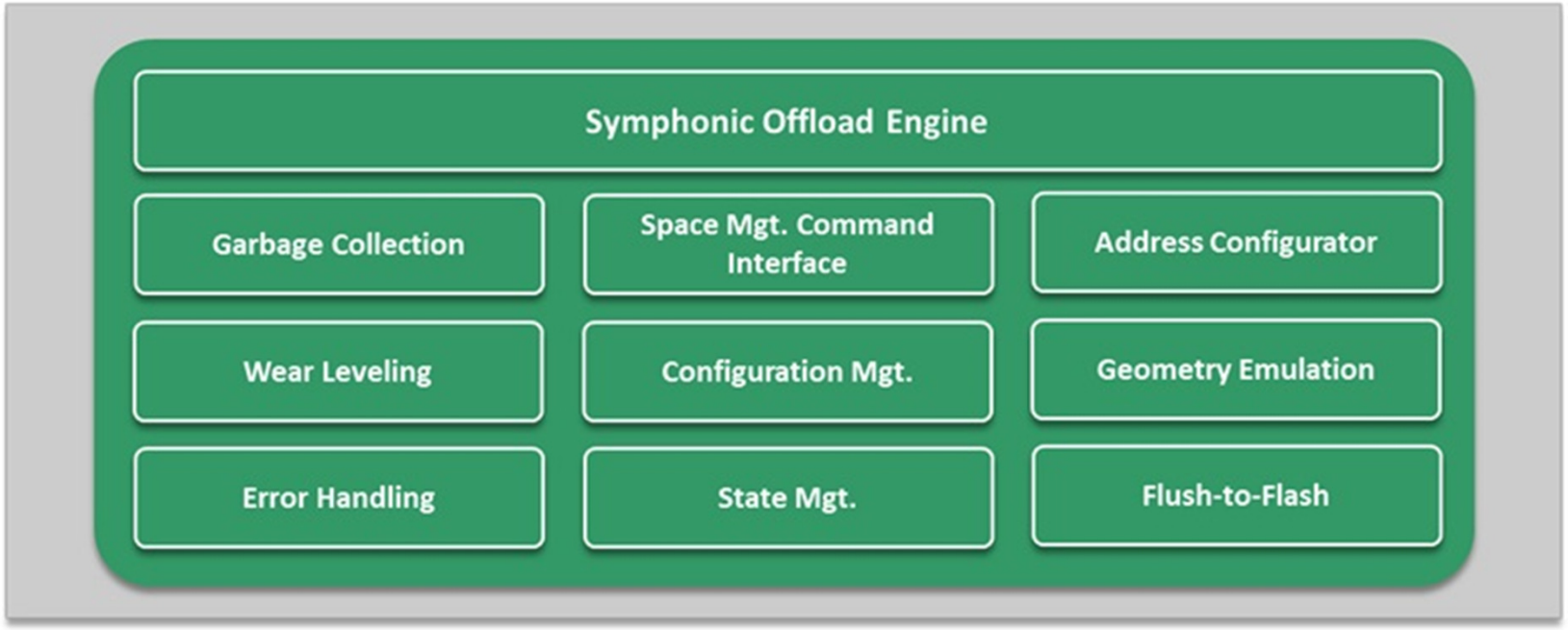
- "direct flash" – provides an ioctl-based API to
  - Read/write flash physical pages
  - App must avoid overwrites
  - App must write pages sequentially
  - App must reclaim live data before scheduling block erase.
  - Userspace liblightnvm library



Symphonic Cooperative Flash Management

- NVMe API conformant…
  - PLUS vendor-specific extensions
- Device exposes “idealized” flash:
  - No (apparent) bad blocks
  - All flash management (GC, WL, DR) activities identified by drive
  - All such activities can be scheduled by user
    - Default/appropriate actions are provided.
    - Knowledgeable apps can do something more fancy
  - If user doesn’t take action within timeout window, drive takes over
  - Allows for warranty model: “drive protects itself”

Symphonic Cooperative Flash Management

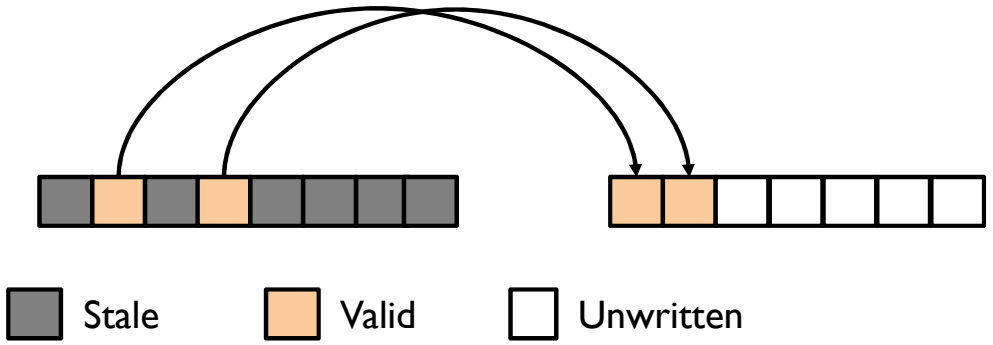


Cooperative Flash Management GC

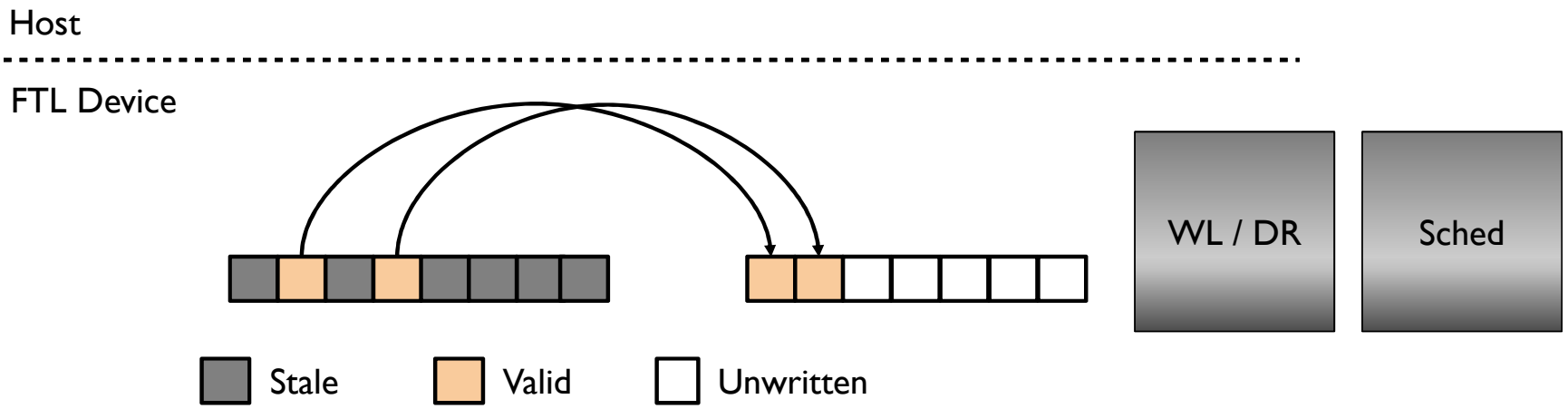
- Drive can optionally identify candidate blocks for garbage collection
- Host
  - determines what live data to relocate
  - determines relocation destination of live data
  - schedules garbage collection process
- Relocation process executed by SSD firmware on drive via Delegated copy/move feature

Garbage Collection - Ownership

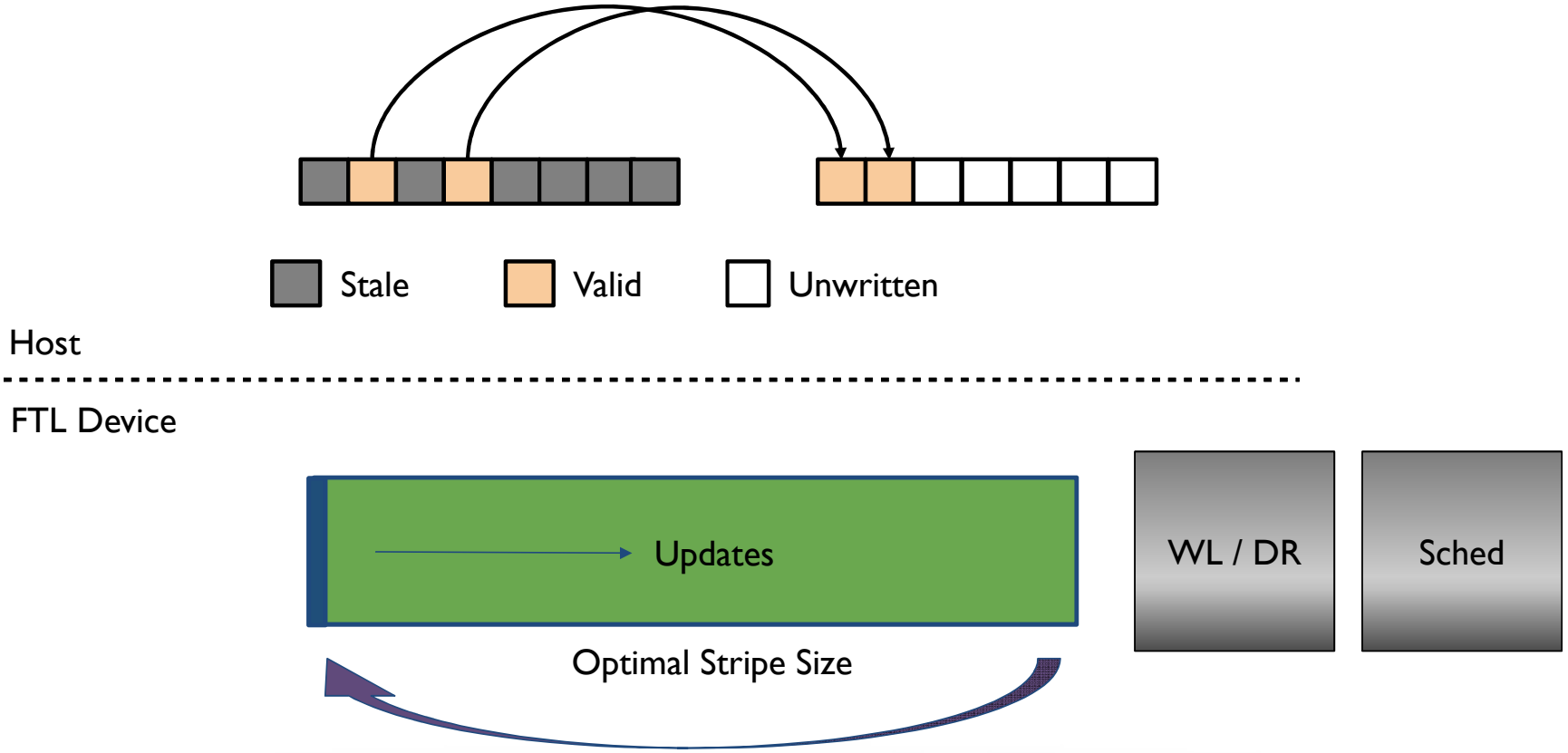
- Accounting, Garbage Collection candidate segment
- Execution of Copy/Relocate process
- Effect of Impact



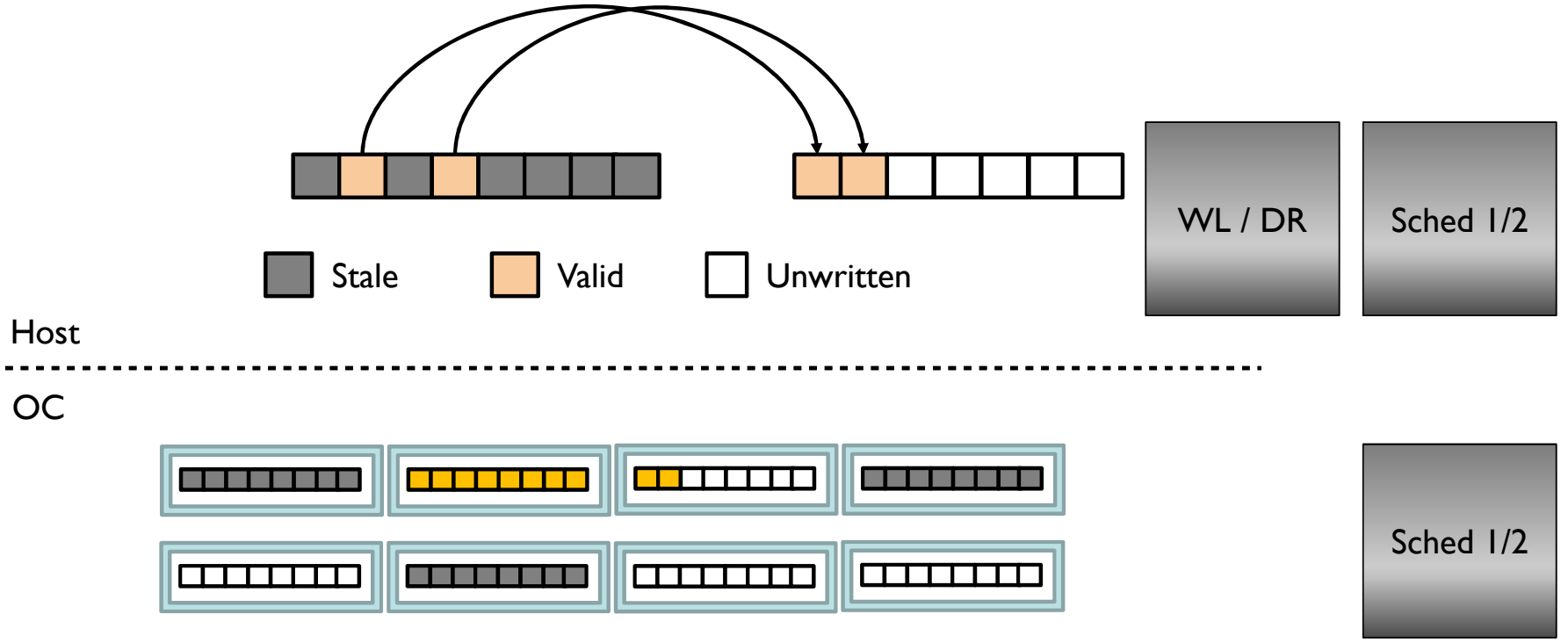
Garbage Collection - FTL



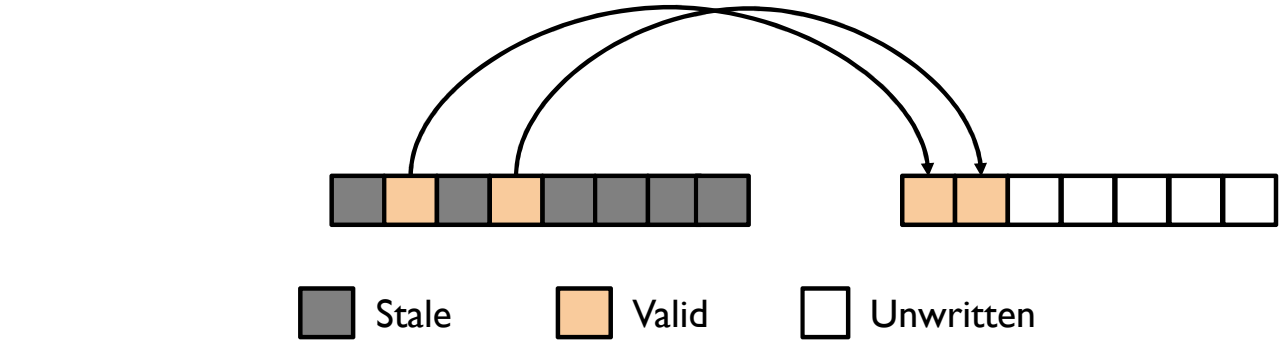
Garbage Collection – Erase Only GC



Garbage Collection - Open Channel



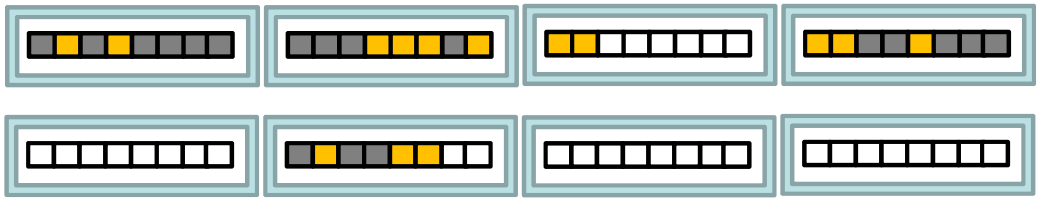
Garbage Collection - CFM



Sched

Host

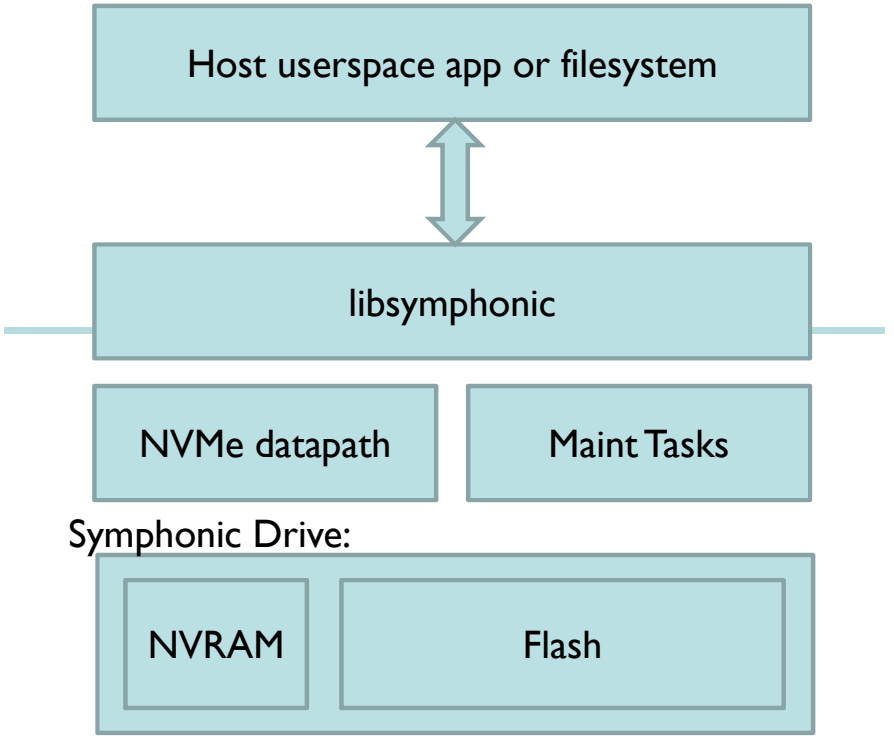
CFM



WL / DR

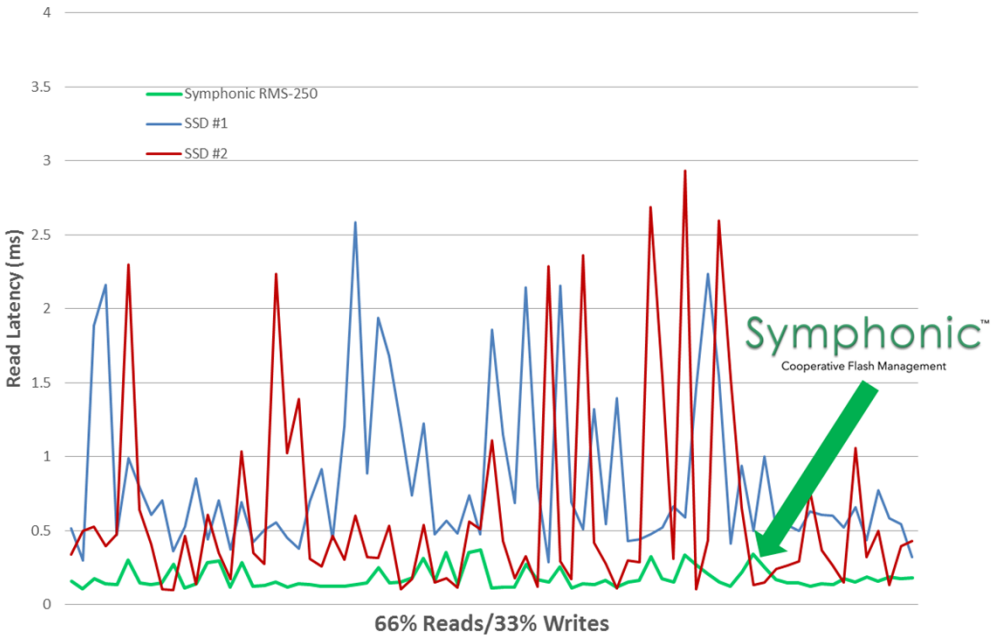
Symphonic API

- With libsymphonic,
  - Datapath to drive is untouched
  - I/O can be staged thru NVRAM
  - Symphonic directs flash Maint. Activities
  - User in control of all scheduling



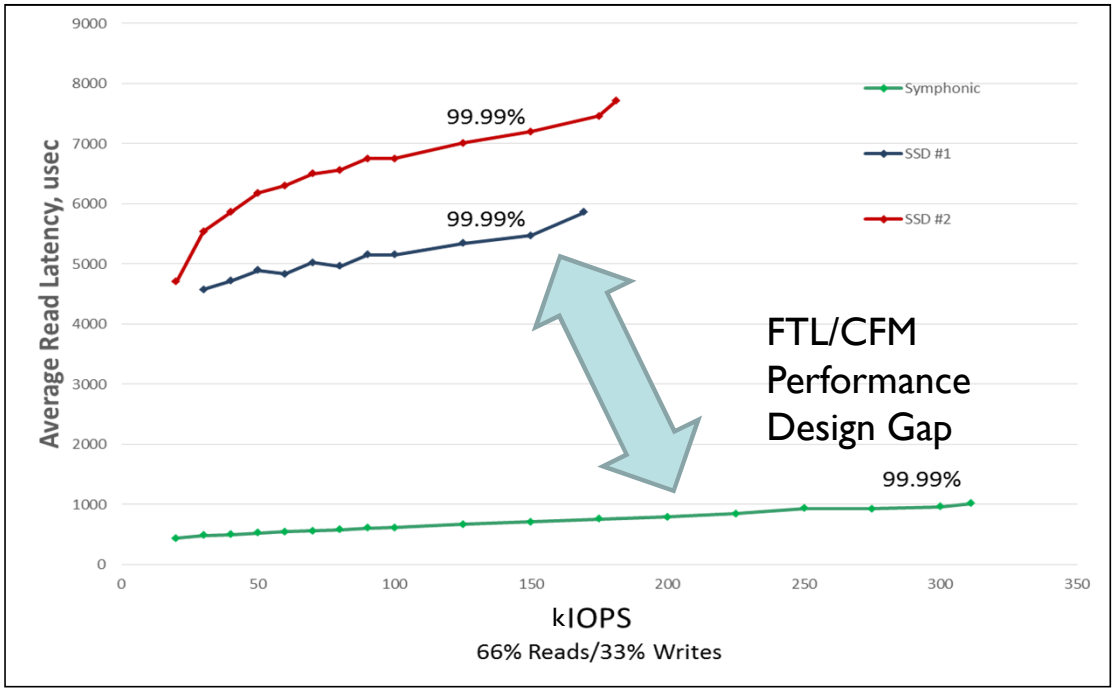
Cooperative Flash Benefits

- App architect has complete control over timing of operations on drive
  - No surprise latency spikes!
- NAND device details are hidden
- User is isolated from NAND detail changes from generation to generation

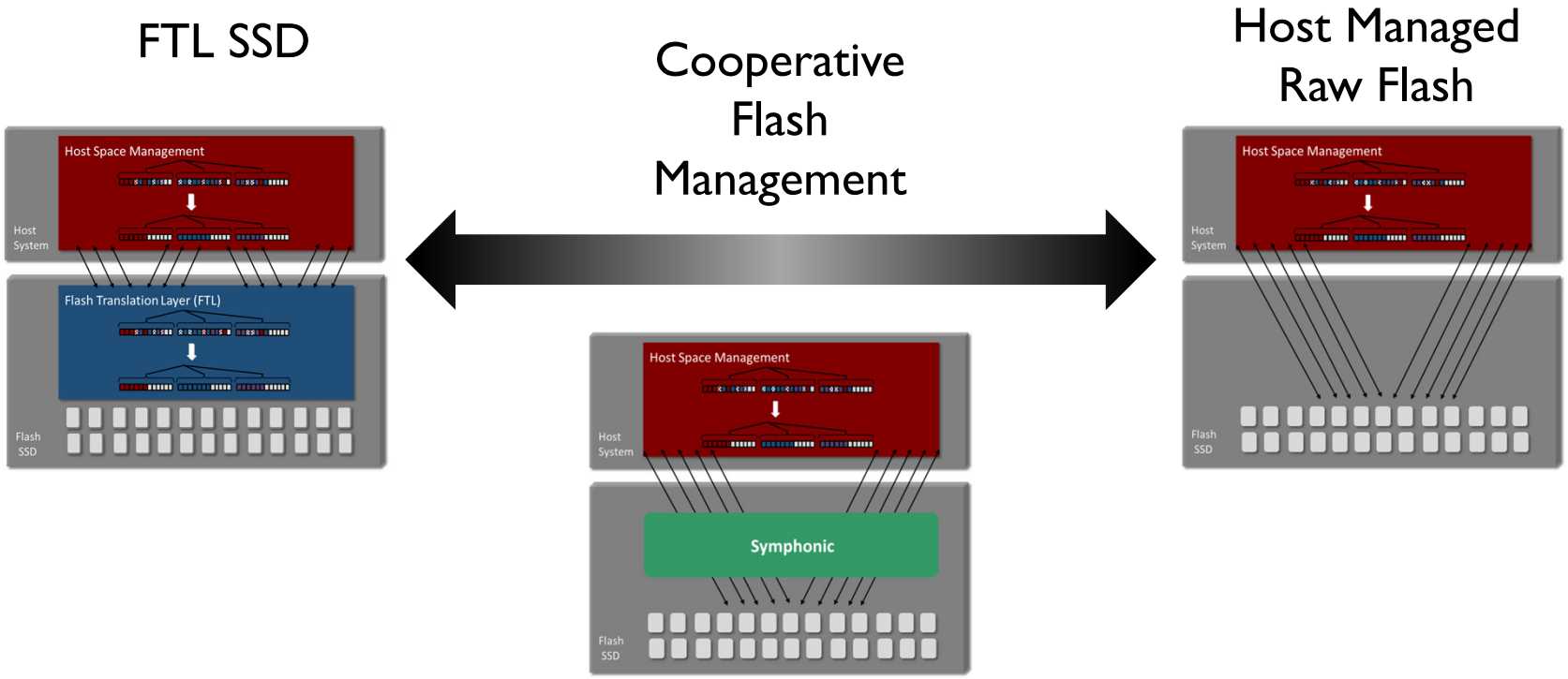


Latency spike reality

- Tail latencies at fixed IOPs load are bad:
- Application must be designed against this 99.99% performance
- Cooperative Flash Management enables achieving this benefit



Symphonic CFM benefits – "Goldilocks"



Comparison

Radian MEMORY SYSTEMS

Feature	FTL SSD	OC-pblk	Legacy FS need	Modern COW FS need	OC-dflash	CFM
Overwriting block API	Y	Y	Y	N		
Autonomous GC	Y	Y	Y	N		
User control of latency	N	N		Y	somewhat	TOTAL
Flash can be warranted	Y			Y		Y
Avoids Log-on-Log				Y	Y	Y

Questions, Notes, Thank you

- Questions?
- Visit our booth!
- Web: http://www.radianmemory.com
- Email: craigr@radianmemory.com
- Thank you!