SDC19 | September 23-26, 2019
Santa Clara, CA

Best Practices for OpenZFS L2ARC in the Era of NVMe

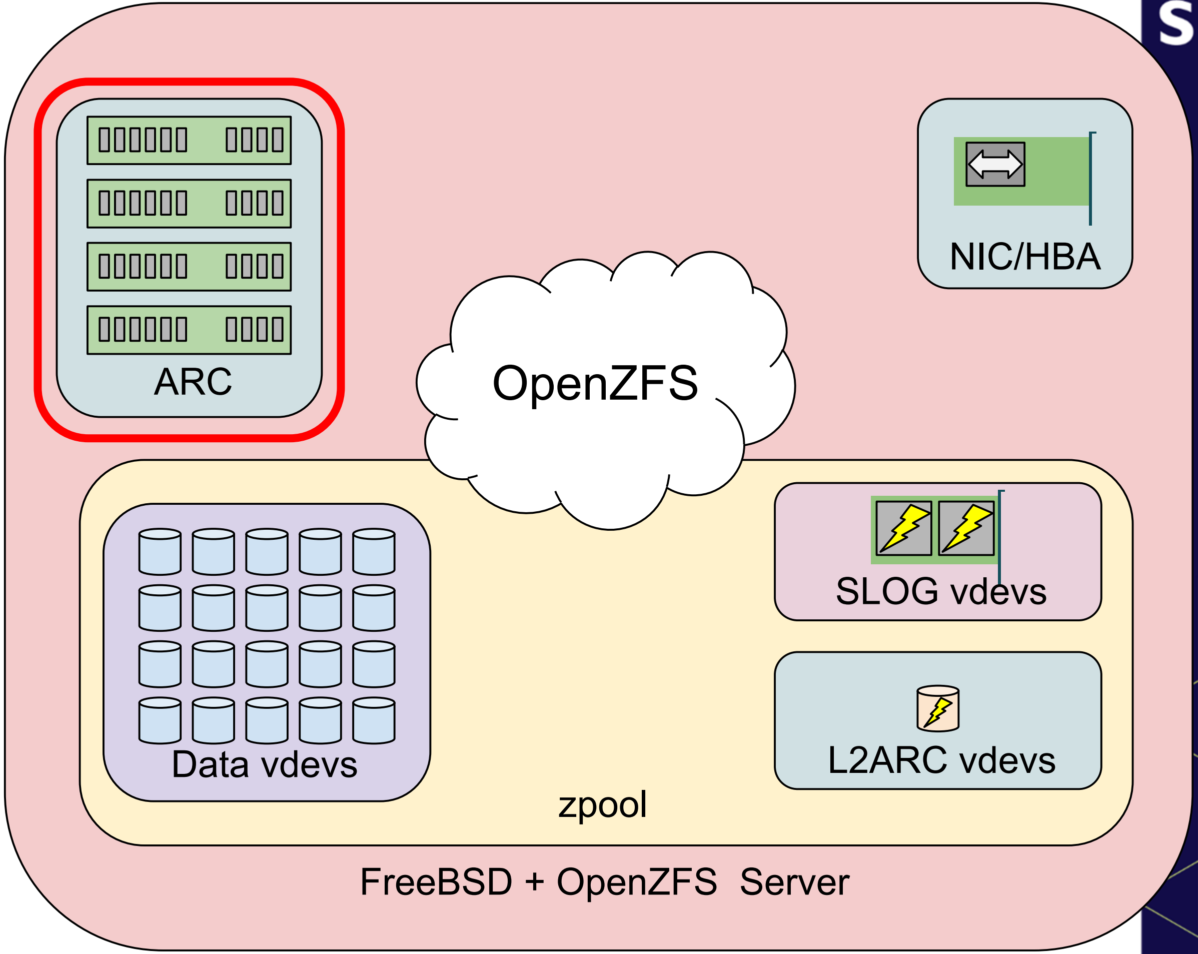
Ryan McKenzie
iXsystems

Agenda

- Brief Overview of OpenZFS ARC / L2ARC
- Key Performance Factors
- Existing “Best Practices” for L2ARC
    - Rules of Thumb, Tribal Knowledge, etc.
- NVMe as L2ARC
    - Testing and Results
- Revised “Best Practices”

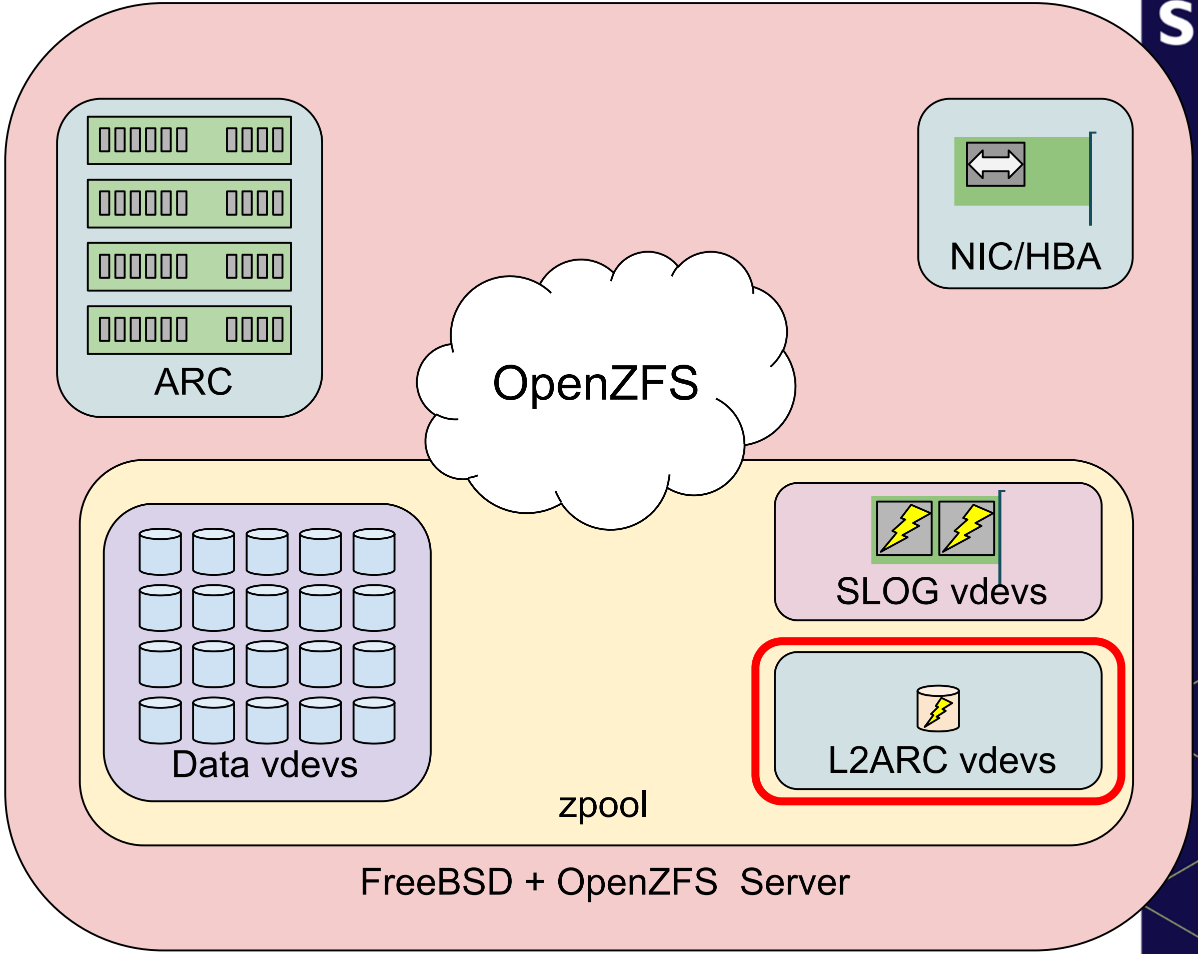
ARC Overview

- **A**daptive **R**eplacement **C**ache
- Resides in system memory
- Shared by all pools
- Used to store/cache:
  - All incoming data
  - “Hottest” data and Metadata (a tunable ratio)
- Balances between
  - Most **Frequently** Used (MFU)
  - Most **Recently** Used (MRU)



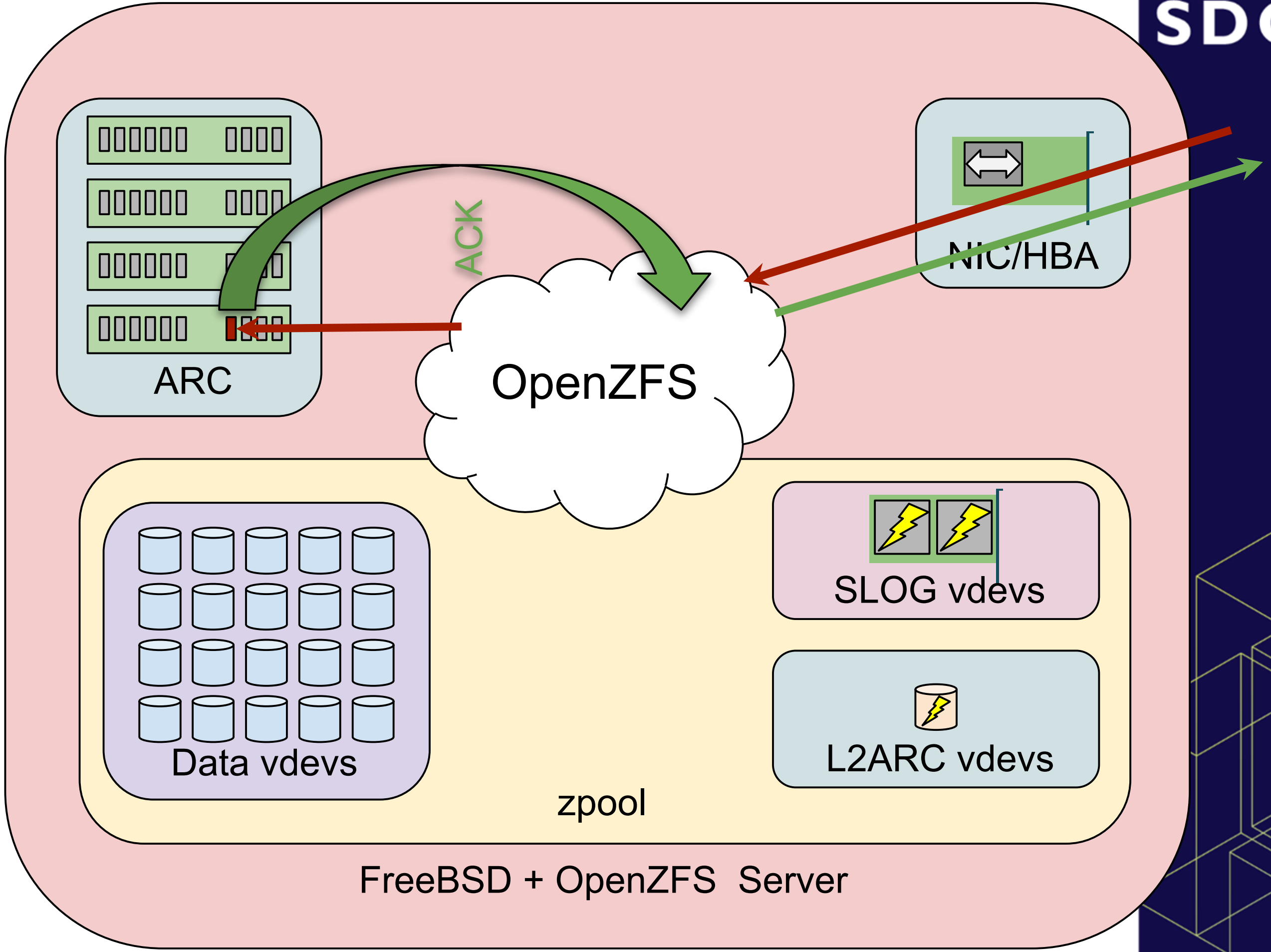
L2ARC Overview

- Level 2 Adaptive Replacement Cache
- Resides on one or more storage devices
  - Usually Flash
- Device(s) added to pool
  - Only services data held by that pool
- Used to store/cache:
  - “Warm” data and metadata (**about to be** evicted from ARC)
  - Indexes to L2ARC blocks stored in ARC headers



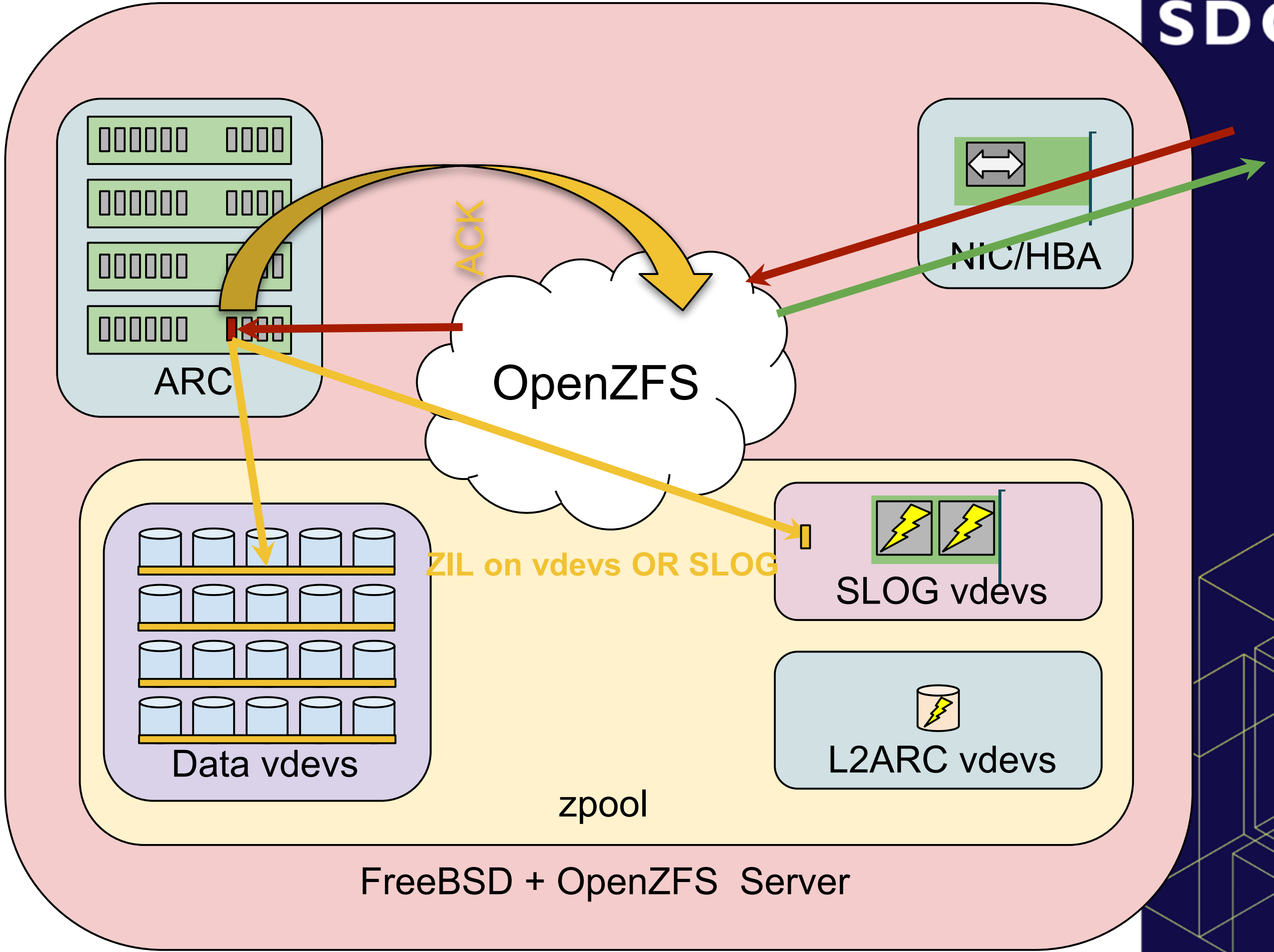
ZFS Writes

- All writes go through ARC, written blocks are “dirty” until on stable storage
  - async write ACKs immediately
  - sync write copied to ZIL/SLOG then ACKs
  - copied to data vdev in TXG
- When no longer dirty, written blocks stay in ARC and move through MRU/MFU lists normally



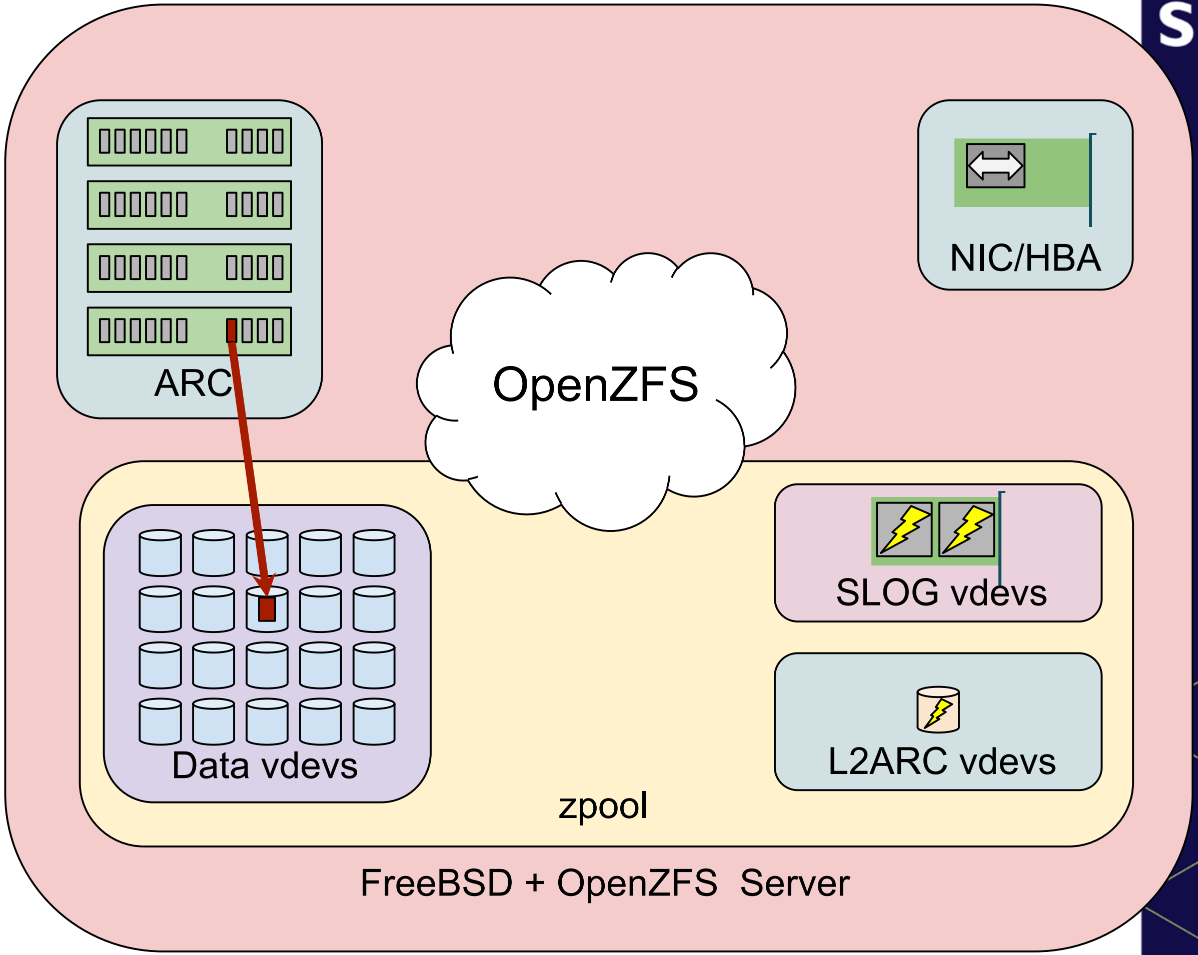
ZFS Writes

- All writes go through ARC, written blocks are “dirty” until on stable storage
  - async write ACKs immediately
  - sync write copied to ZIL/SLOG then ACKs
  - copied to data vdev in TXG
- When no longer dirty, written blocks stay in ARC and move through MRU/MFU lists normally



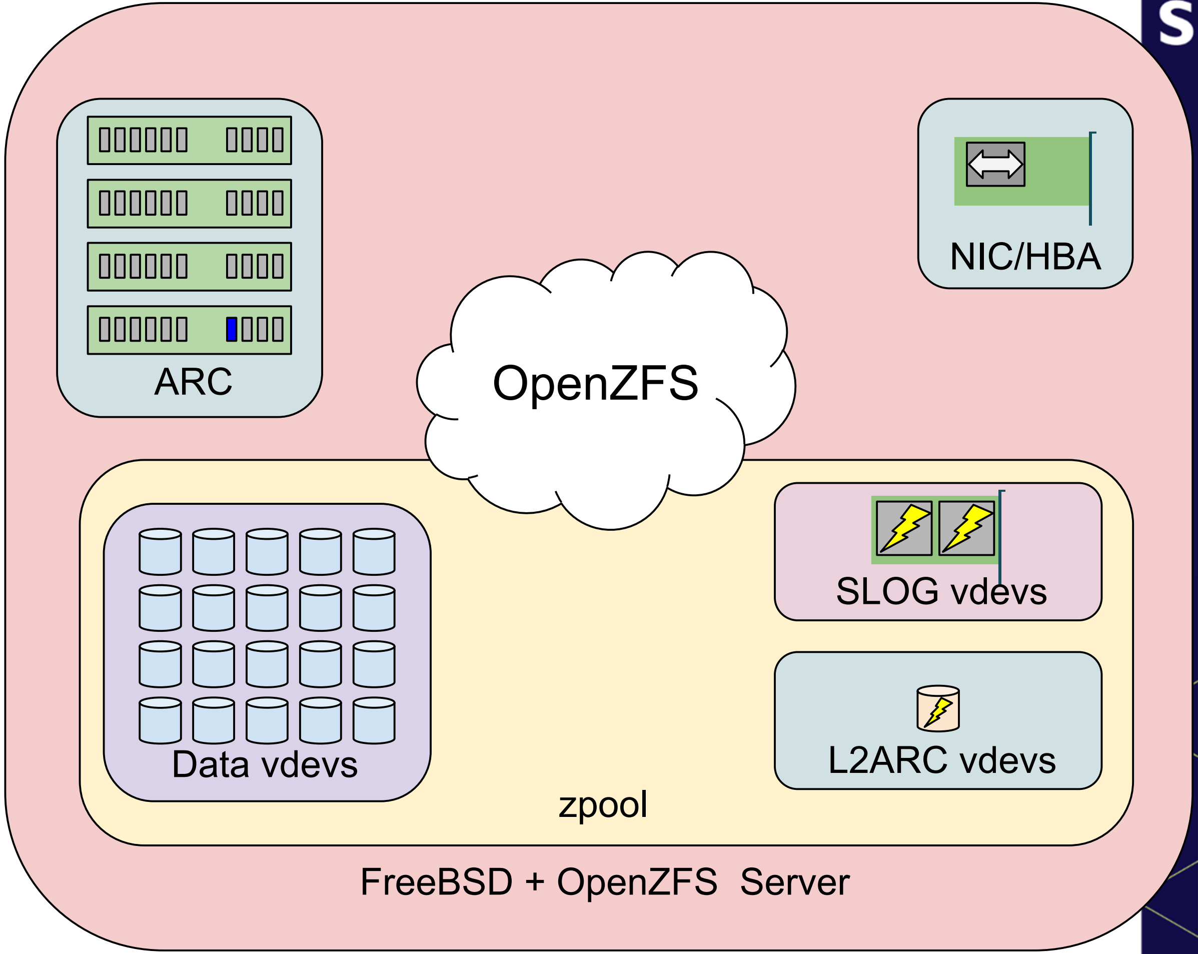
ZFS Writes

- All writes go through ARC, written blocks are “dirty” until on stable storage
  - async write ACKs immediately
  - sync write copied to ZIL/SLOG then ACKs
  - copied to data vdev in TXG
- When no longer dirty, written blocks stay in ARC and move through MRU/MFU lists normally



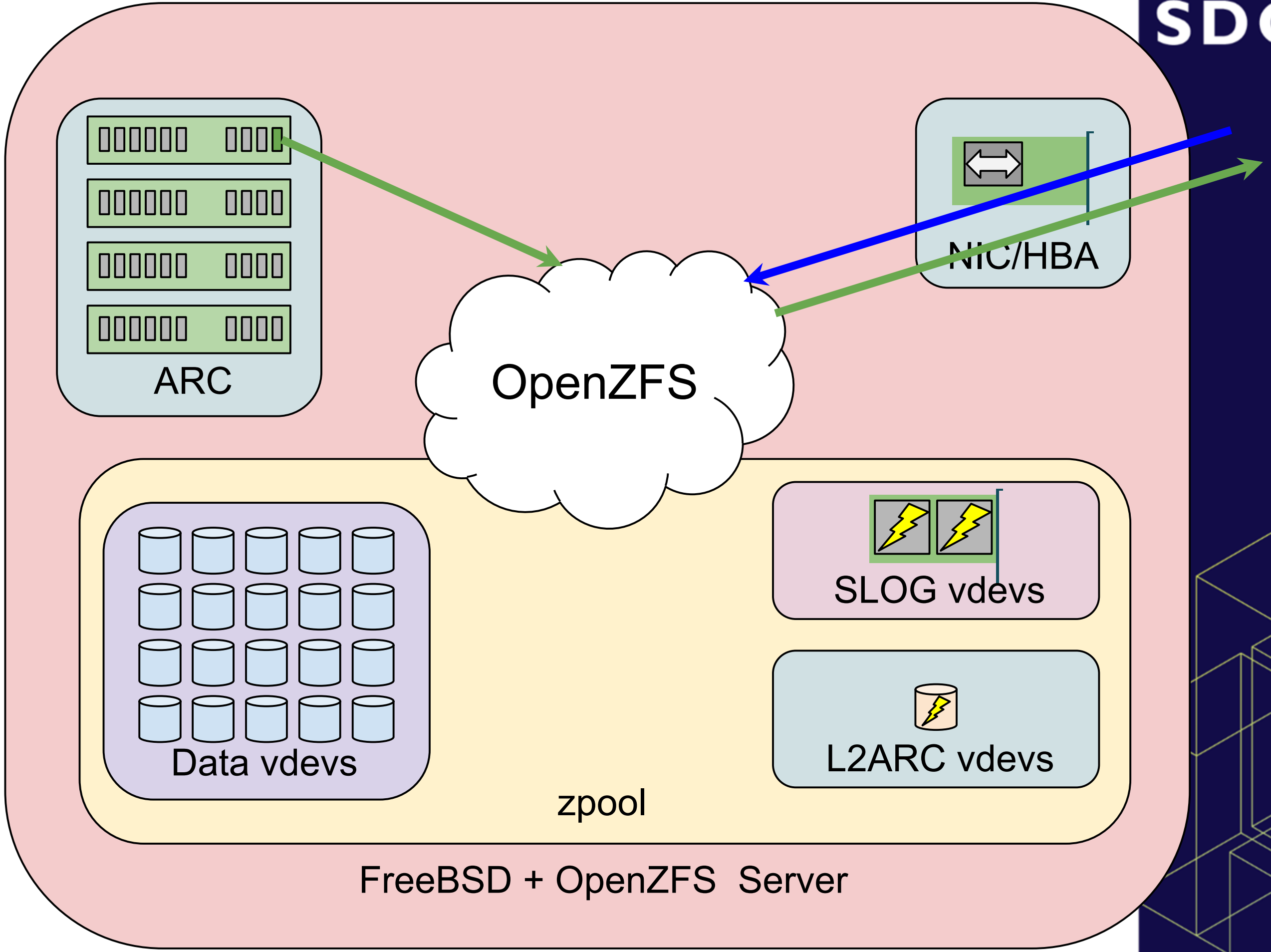
ZFS Writes

- All writes go through ARC, written blocks are “dirty” until on stable storage
  - async write ACKs immediately
  - sync write copied to ZIL/SLOG then ACKs
  - copied to data vdev in TXG
- When no longer dirty, written blocks stay in ARC and move through MRU/MFU lists normally



ZFS Reads

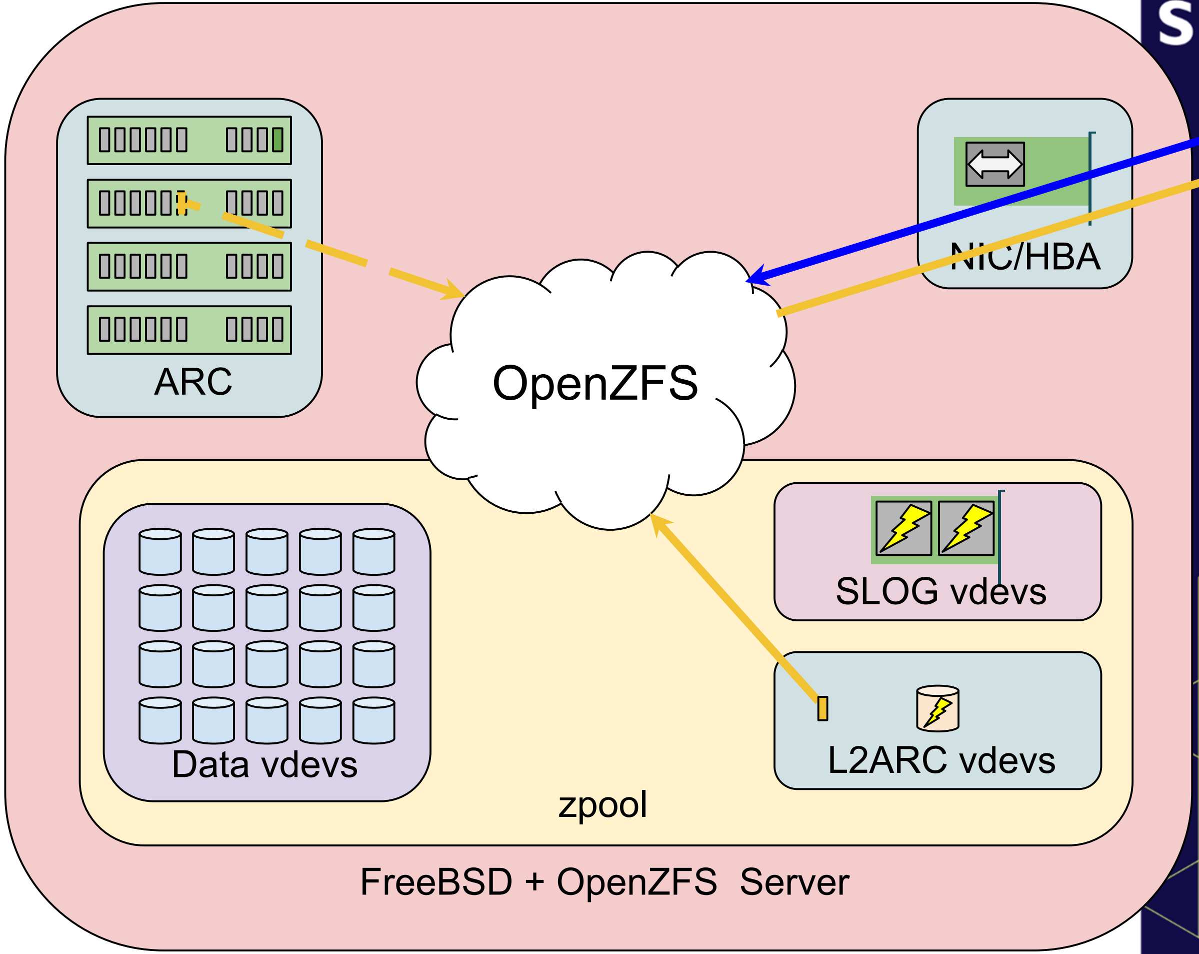
- Requested block in ARC
  - Respond with block from ARC
- Requested block in L2
  - Get index to L2ARC block (from ARC)
  - Respond with block from L2ARC
- Otherwise: read miss
  - Copy block from data vdev to ARC
  - Respond with block from ARC
  - "ARC PROMOTE": read block(s) stay in ARC and move through MRU/MFU lists normally



SDC 19

ZFS Reads

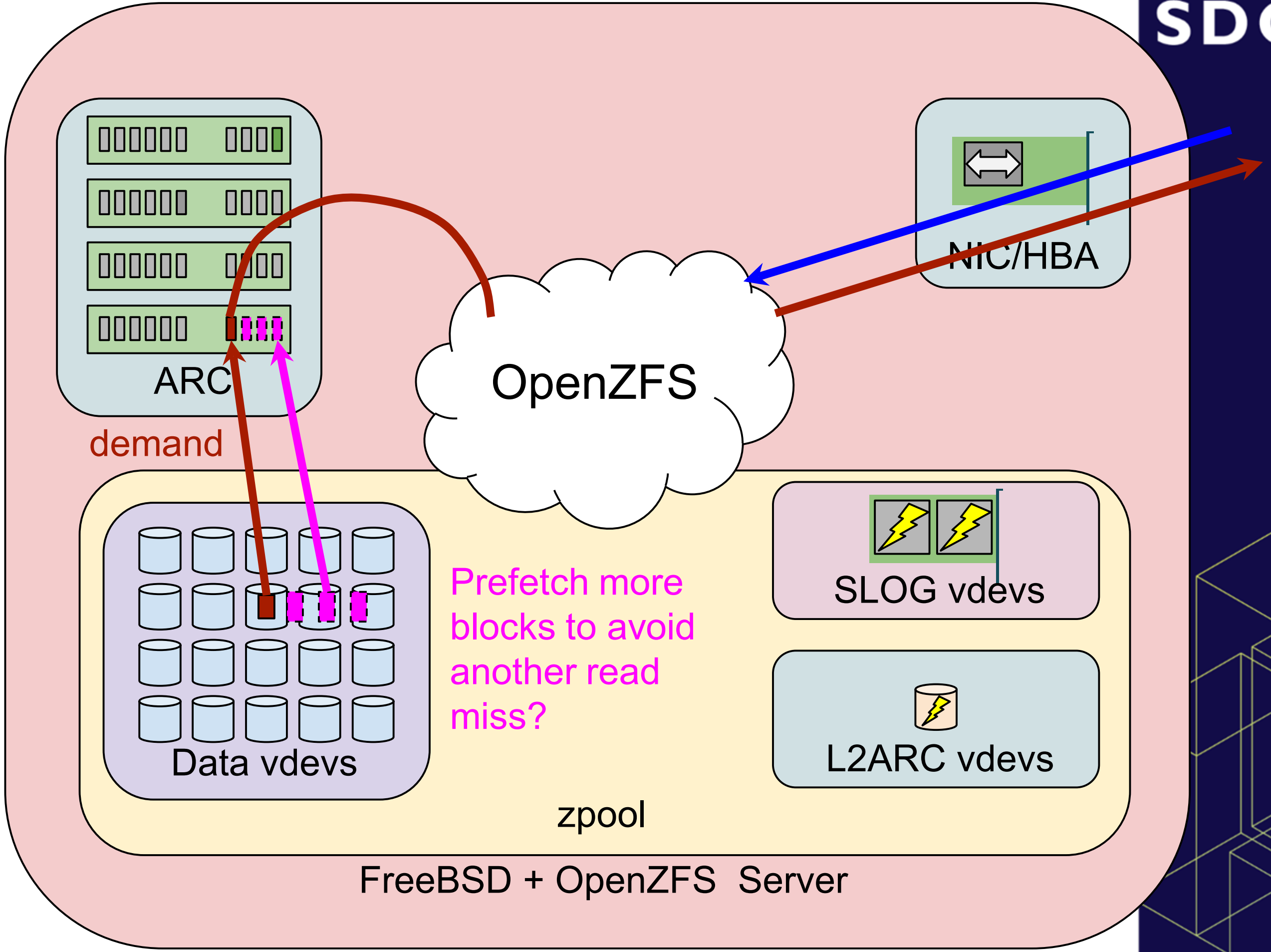
- Requested block in ARC
  - Respond with block from ARC
- Requested block in L2
  - Get index to L2ARC block (from ARC)
  - Respond with block from L2ARC
- Otherwise: read miss
  - Copy block from data vdev to ARC
  - Respond with block from ARC
  - “ARC PROMOTE”: read block(s) stay in ARC and move through MRU/MFU lists normally



SDC 19

ZFS Reads

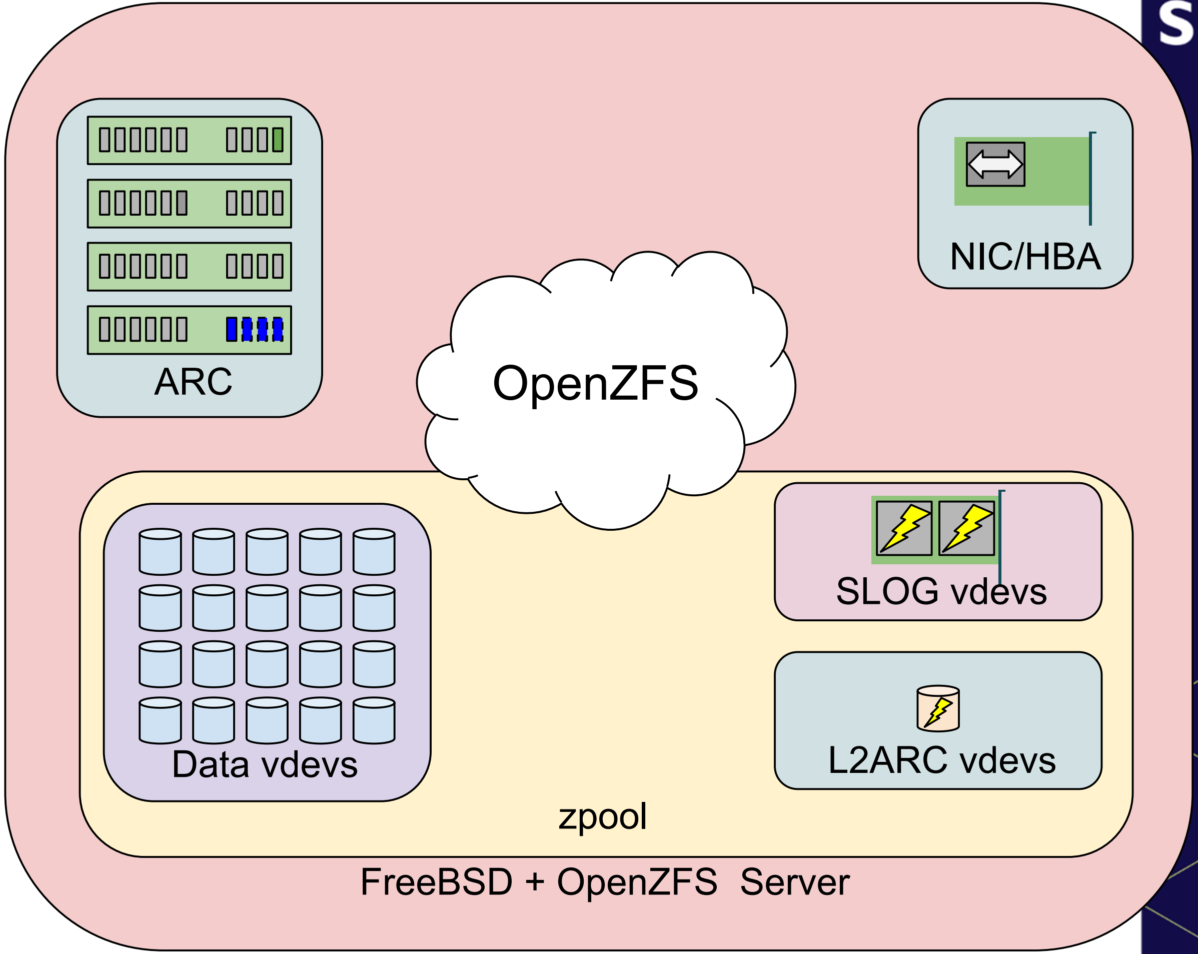
- Requested block in ARC
  - Respond with block from ARC
- Requested block in L2
  - Get index to L2ARC block (from ARC)
  - Respond with block from L2ARC
- Otherwise: read miss
  - Copy block from data vdev to ARC
  - Respond with block from ARC
  - “ARC PROMOTE”: read block(s) stay in ARC and move through MRU/MFU lists normally



SDC 19

ZFS Reads

- Requested block in ARC
  - Respond with block from ARC
- Requested block in L2
  - Get index to L2ARC block (from ARC)
  - Respond with block from L2ARC
- Otherwise: read miss
  - Copy block from data vdev to ARC
  - Respond with block from ARC
  - "ARC PROMOTE": read block(s) stay in ARC and move through MRU/MFU lists normally

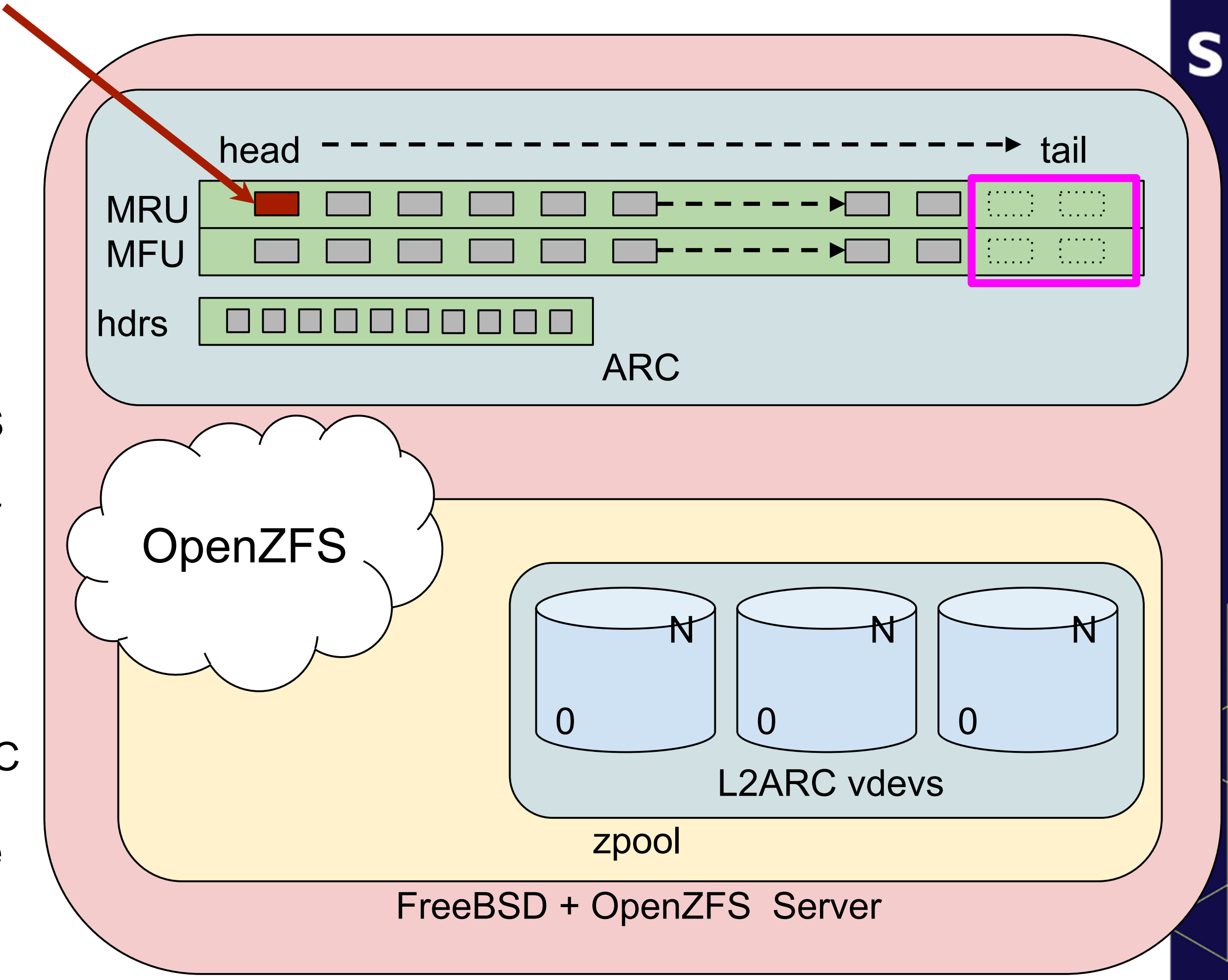


ARC Reclaim

- Called to maintain/make room in ARC for incoming writes

L2ARC Feed

- Asynchronous of ARC Reclaim to speed up ZFS write ACKs
- Periodically scans tails of MRU/MFU
  - Copies blocks from ARC to L2ARC
  - Index to L2ARC block resides in ARC headers
  - Feed rate is tunable
- Writes to L2 device(s) sequentially (rnd robin)



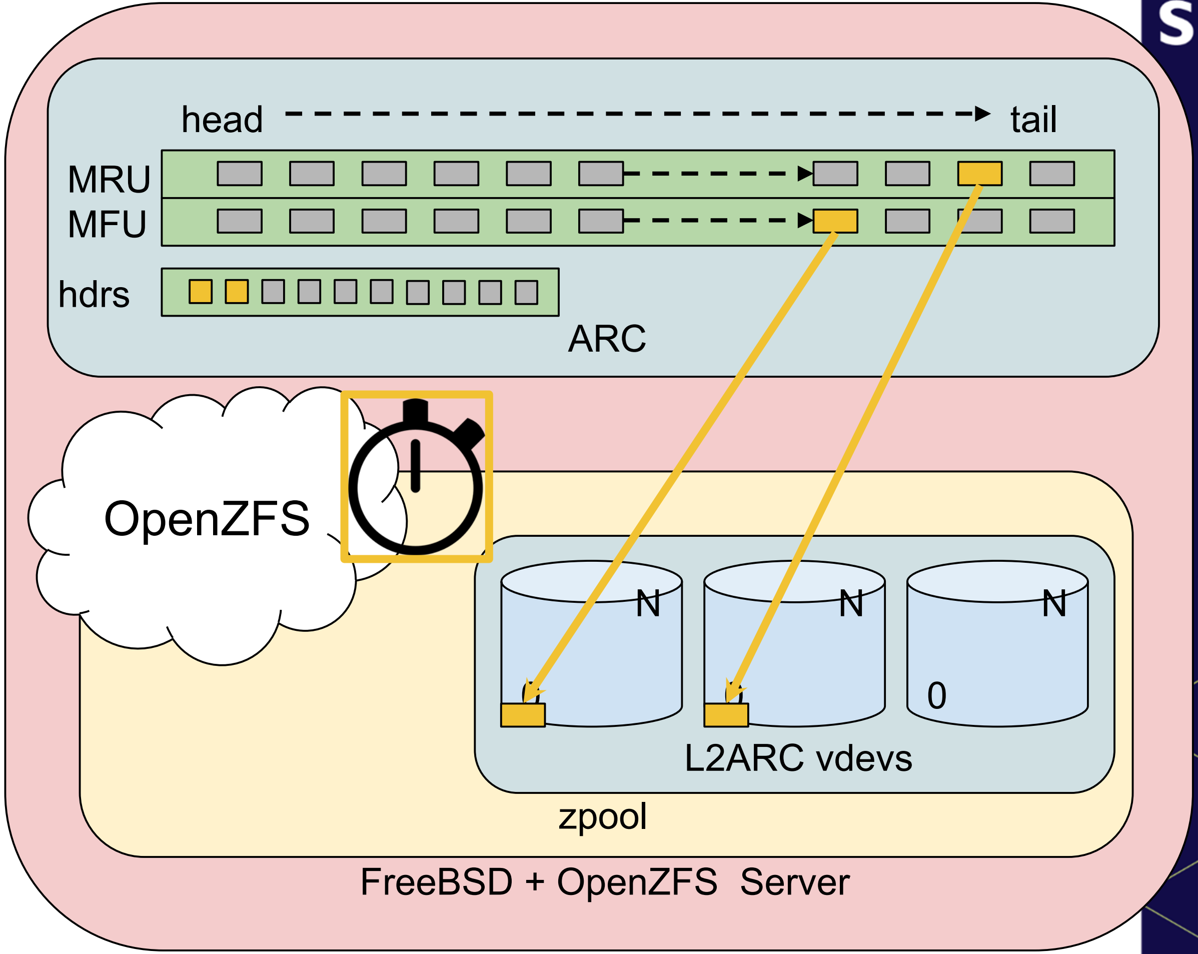
SDC 19

ARC Reclaim

- Called to maintain/make room in ARC for incoming writes

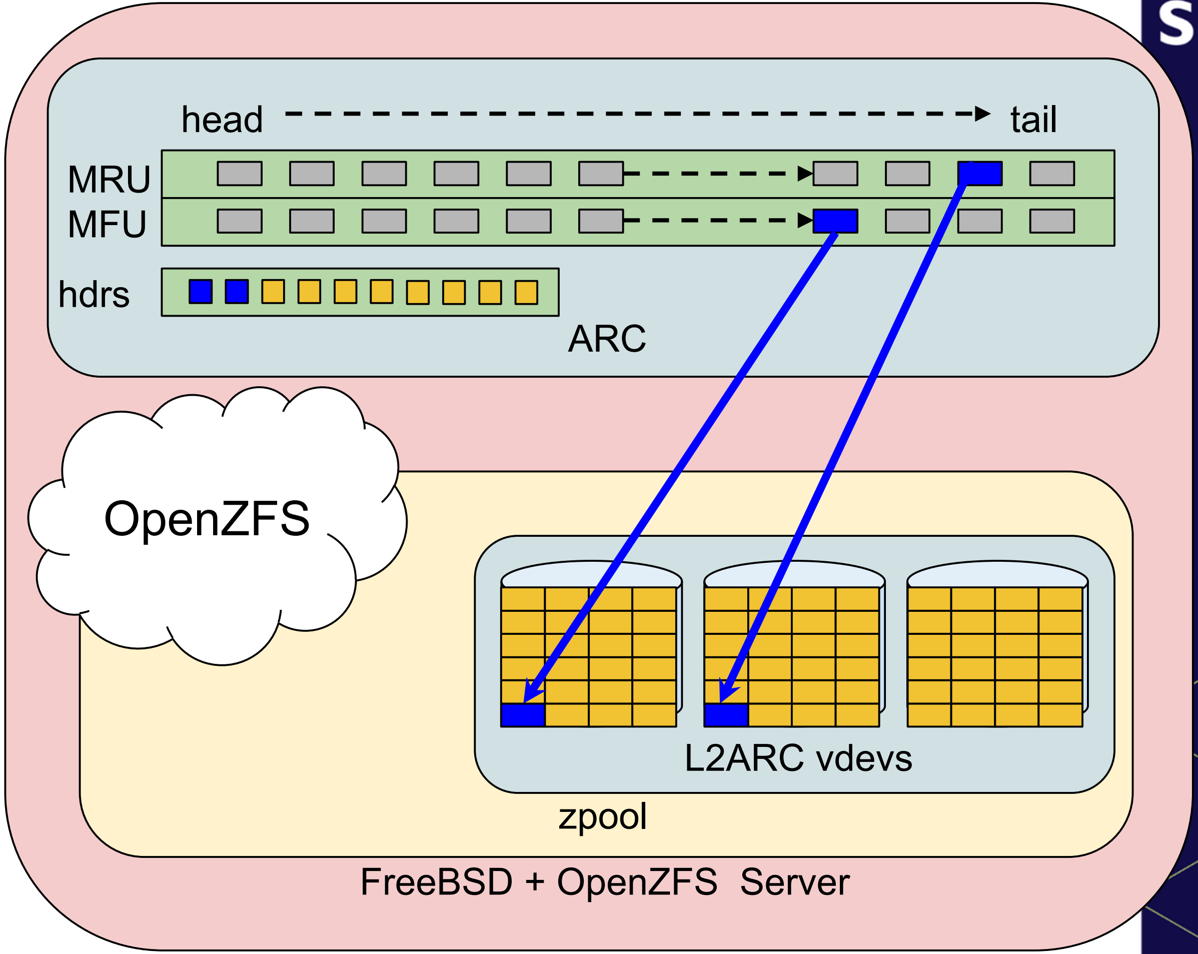
L2ARC Feed

- Asynchronous of ARC Reclaim to speed up ZFS write ACKs
- Periodically scans tails of MRU/MFU
  - Copies blocks from ARC to L2ARC
  - Index to L2ARC block resides in ARC headers
  - Feed rate is tunable
- Writes to L2 device(s) sequentially (rnd robin)



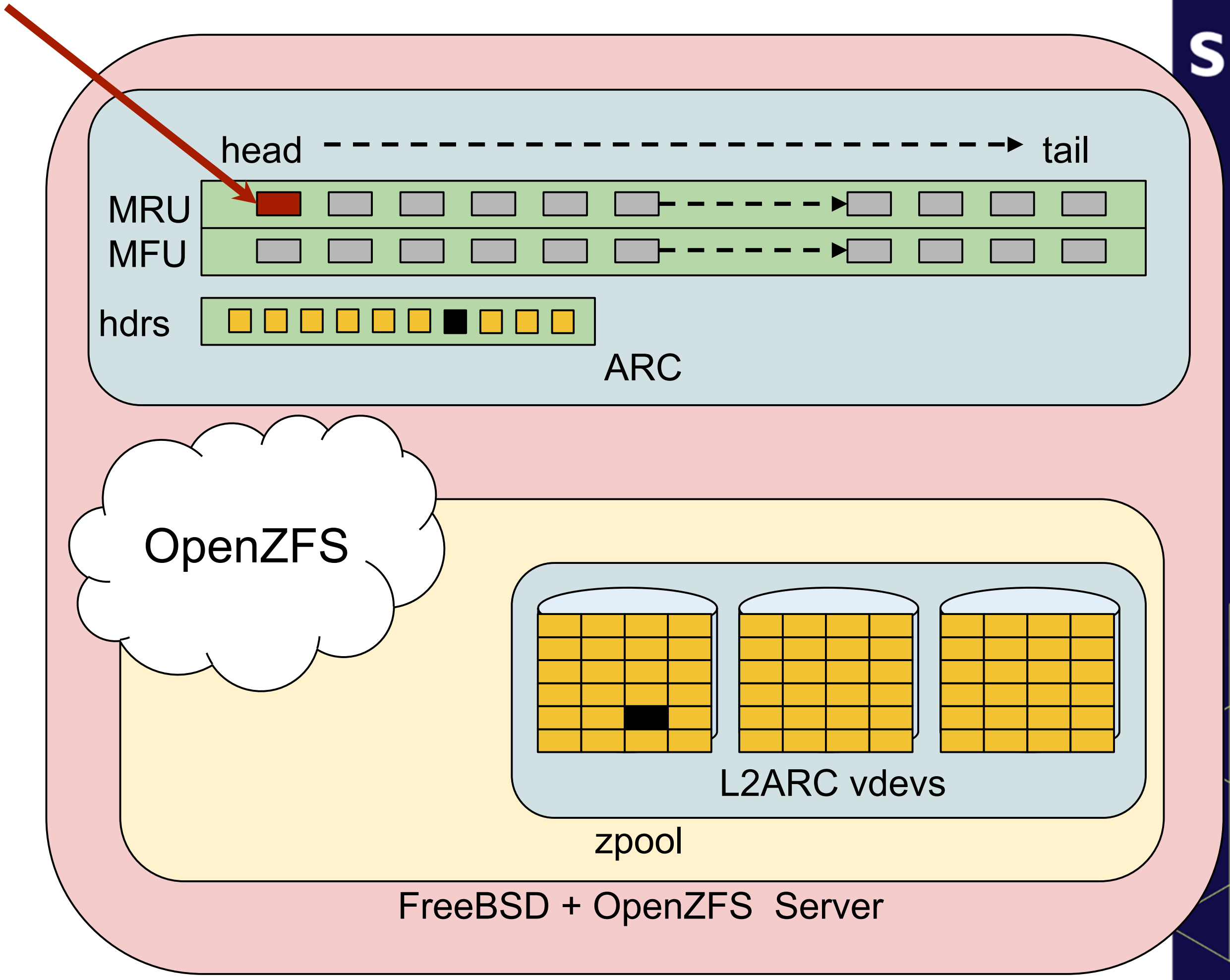
L2ARC “Reclaim”

- Not really a concept of reclaim
- When L2ARC device(s) get full (sequentially at the “end”), L2ARC Feed just starts over at the “beginning”
    - Overwrites whatever was there before and updates index in ARC header
- If an L2ARC block is written, ARC will handle the write of the dirty block, L2ARC block is stale and the index is dropped



L2ARC "Reclaim"

- Not really a concept of reclaim
- When L2ARC device(s) get full (sequentially at the "end"), L2ARC Feed just starts over at the "beginning"
  - Overwrites whatever was there before and updates index in ARC header
- If an L2ARC block is written , ARC will handle the write of the dirty block, **L2ARC block is stale and the index is dropped**



Some Notes:
ARC / L2ARC Architecture

- ARC/L2 “Blocks” are variable size:
  - =volblock size for zvol data blocks
  - =record size for dataset data blocks
  - =indirect block size for metadata blocks
- Smaller volblock/record sizes yield more metadata blocks (overhead) in the system
  - may need to tune metadata % of ARC

Some Notes:
ARC / L2ARC Architecture

- Blocks get into ARC via any ZFS write or by demand/prefetch on ZFS read miss
- Blocks cannot get into L2ARC unless they are in ARC first (primary/secondary cache settings)
  - CONFIGURATION PITFALL
- L2ARC is not persistent
  - Blocks are persistent on the L2ARC device(s) but the indexes to them are lost if the ARC headers are lost (main memory)

Some Notes:
ARC / L2ARC Architecture

SDC 19

- Write-heavy workloads can “churn” tail of the ARC MRU list quickly
- Prefetch-heavy workloads can “scan” tail of the ARC MRU list quickly
- In both cases…
  - ARC MFU bias maintains integrity of cache
  - L2ARC Feed “misses” many blocks
  - Blocks that L2ARC does feed may not be hit again - “Wasted L2ARC Feed”

Agenda

September 23-26, 2019
Santa Clara, CA

- ✓ Brief Overview of OpenZFS ARC / L2ARC
- ❑ **Key Performance Factors**
- ❑ Existing “Best Practices” for L2ARC
  - ❑ Rules of Thumb, Tribal Knowledge, etc.
- ❑ NVMe as L2ARC
  - ❑ Testing and Results
- ❑ Revised “Best Practices”

Key Performance Factors

SDC 19

- Random Read-Heavy Workload
  - L2ARC designed for this, expect very effective once warmed
- Sequential Read-Heavy Workload
  - L2ARC not originally designed for this, but specifically noted that it might work with “future SSDs or other storage tech.”
    - Let’s revisit this with NVMe!

Key Performance Factors

- Write-Heavy Workload (random or sequential)
  - ARC MRU churn, memory pressure, L2ARC never really warmed because many L2ARC blocks get invalidated/dirty
    - “Wasted” L2ARC Feeds
  - ARC and L2ARC are designed to primarily benefit reads
  - Design expectation is “do no harm”, but there may be a constant background performance impact of L2ARC feed

Key Performance Factors

- Small ADS (relative to ARC and L2ARC)
  - Higher possibility for L2ARC to fully warm and for L2ARC Feed to slow down
- Large ADS (relative to ARC and L2ARC)
  - Higher possibility for constant background L2ARC Feed

Agenda

September 23-26, 2019
Santa Clara, CA

- ✓ Brief Overview of OpenZFS ARC / L2ARC
- ✓ Key Performance Factors
- ❑ **Existing "Best Practices" for L2ARC**
  - ❑ **Rules of Thumb, Tribal Knowledge, etc.**
- ❑ NVMe as L2ARC
  - ❑ Testing and Results
- ❑ Revised "Best Practices"

Existing “Best Practices” for L2ARC

- Do not use for sequential workloads
  - Segregated pools and/or datasets
    - At config time
  - “l2arc noprefetch” setting
    - Per Pool, Runtime tunable
  - “secondarycache=metadata” setting
    - primarycache and secondarycache are per dataset, Runtime tunable
    - all, metadata, none

Existing “Best Practices” for L2ARC

- Do not use for sequential workloads
  - Segregated pools
    - Global, At config time
  - “l2arc noprefetch” setting
    - Per Pool, Runtime tunable
  - “secondarycache=metadata” setting
    - primarycache and secondarycache are per dataset, Runtime tunable
    - all, metadata, none

Existing “Best Practices” for L2ARC

- ❑ All are more or less plausible given our review of the design of L2ARC
- ❑ Time for some testing and validation!

Agenda

September 23-26, 2019
Santa Clara, CA

✓ Brief Overview of OpenZFS ARC / L2ARC
✓ Key Performance Factors
✓ Existing "Best Practices" for L2ARC
  ✓ Rules of Thumb, Tribal Knowledge, etc.
❑ **NVMe as L2ARC**
  ❑ **Testing and Validation**
❑ Revised "Best Practices"

Solution Under Test

SDC 19

- FreeBSD+OpenZFS based storage server
  - 512G main memory
  - 4x 1.6T NVMe drives installed
    - Enterprise-grade dual port PCIeG3
      - Just saying “NVMe” isn’t enough...
    - Tested in various L2ARC configurations
  - 142 HDDs in 71 mirror vdevs
  - 1x 100GbE NIC (optical)
  - Exporting 12x iSCSI LUNs (pre-filled)
  - 100GiB of active data per LUN (1.2TiB total ADS)

Solution Under Test

- ADS far too big for ARC
  - Will “fit into” any of our tested L2ARC configurations
- Preconditioned ARC/L2ARC to be “fully warm” before measurement (ARC size + L2 size >- 90% ADS)
  - Possibly some blocks aren’t yet in cache, esp. w/random
- System reboot between test runs (L2ARC configs)
  - Validated “balanced” read and write activity across the L2ARC devices

Solution Under Test

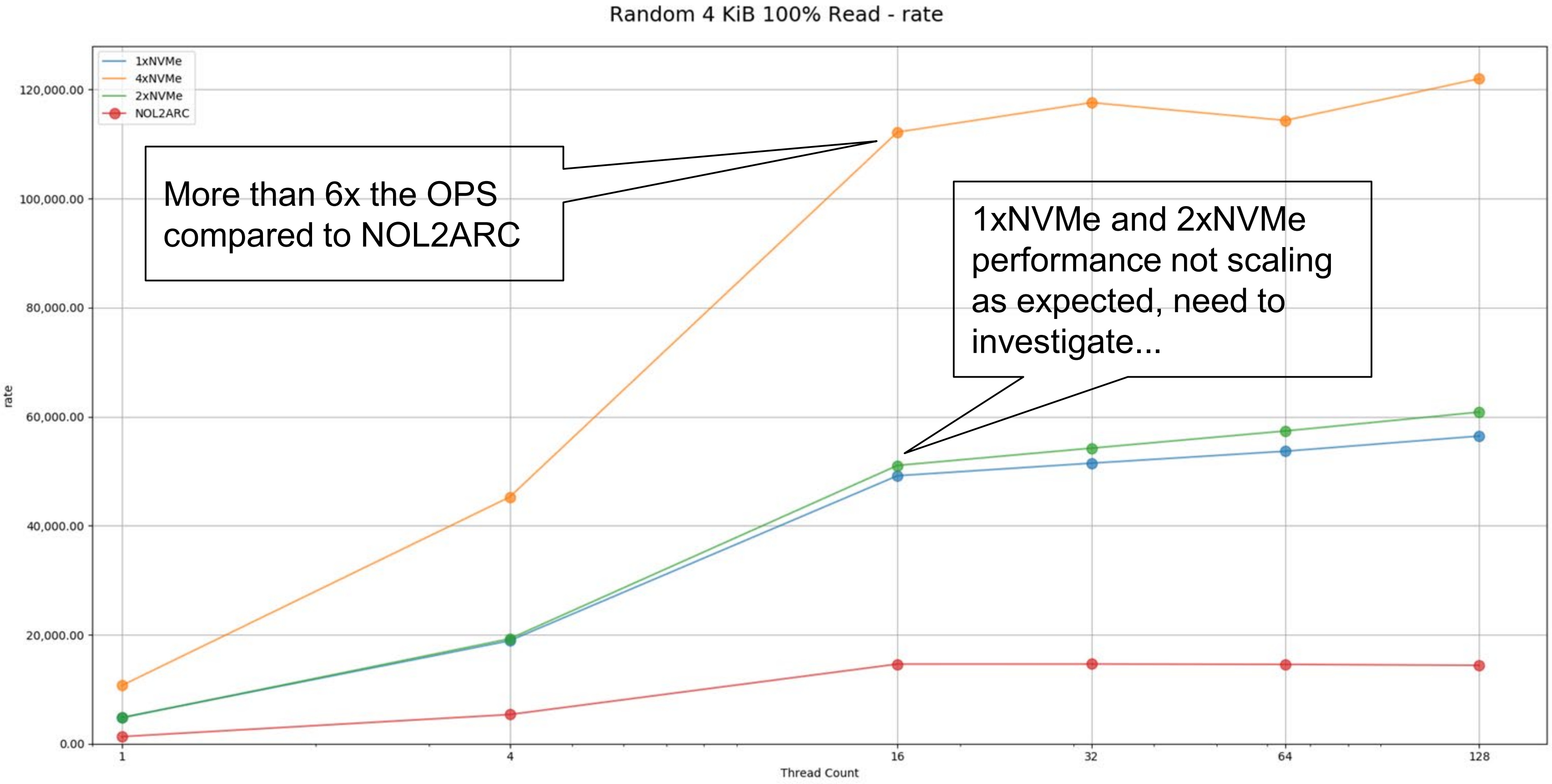
SDC 19

- Load Generation
  - 12x Physical CentOS 7.4 Clients (32G main memory)
  - 1x 10Gbe NIC (optical)
  - Map one LUN per client
  - VDBENCH to generate synthetic workloads
    - In-House “Active Benchmarking” Automation
  - Direct IO to avoid client cache
- Network (100GbE Juniper)
  - Server -> direct connect fiber via 100Gbe QSFP
  - Clients -> aggregated in groups of 4 via 40Gbe QSFP

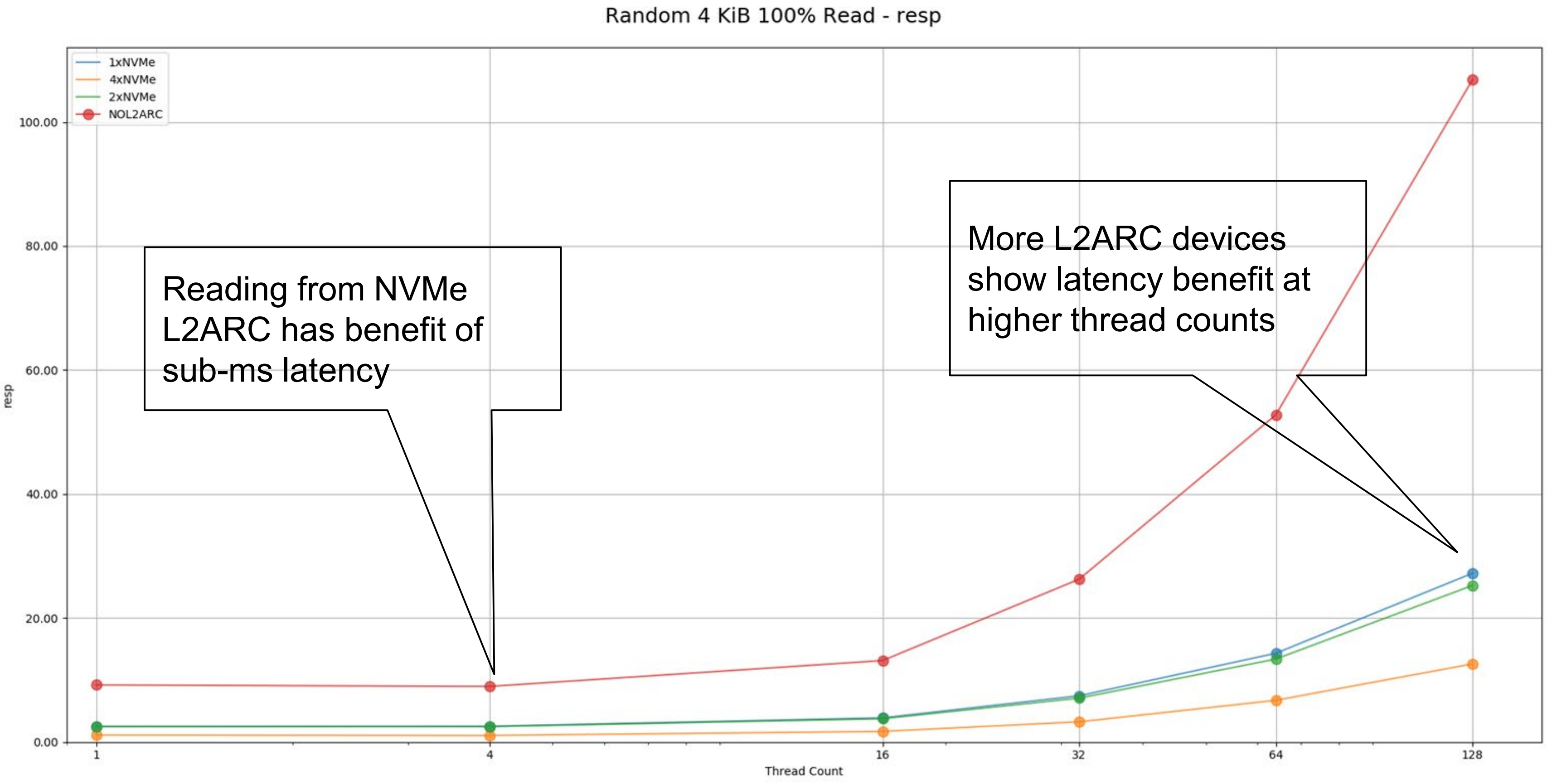
Random Read-Heavy

- 4K IO Size, Pure Random, 100% Read

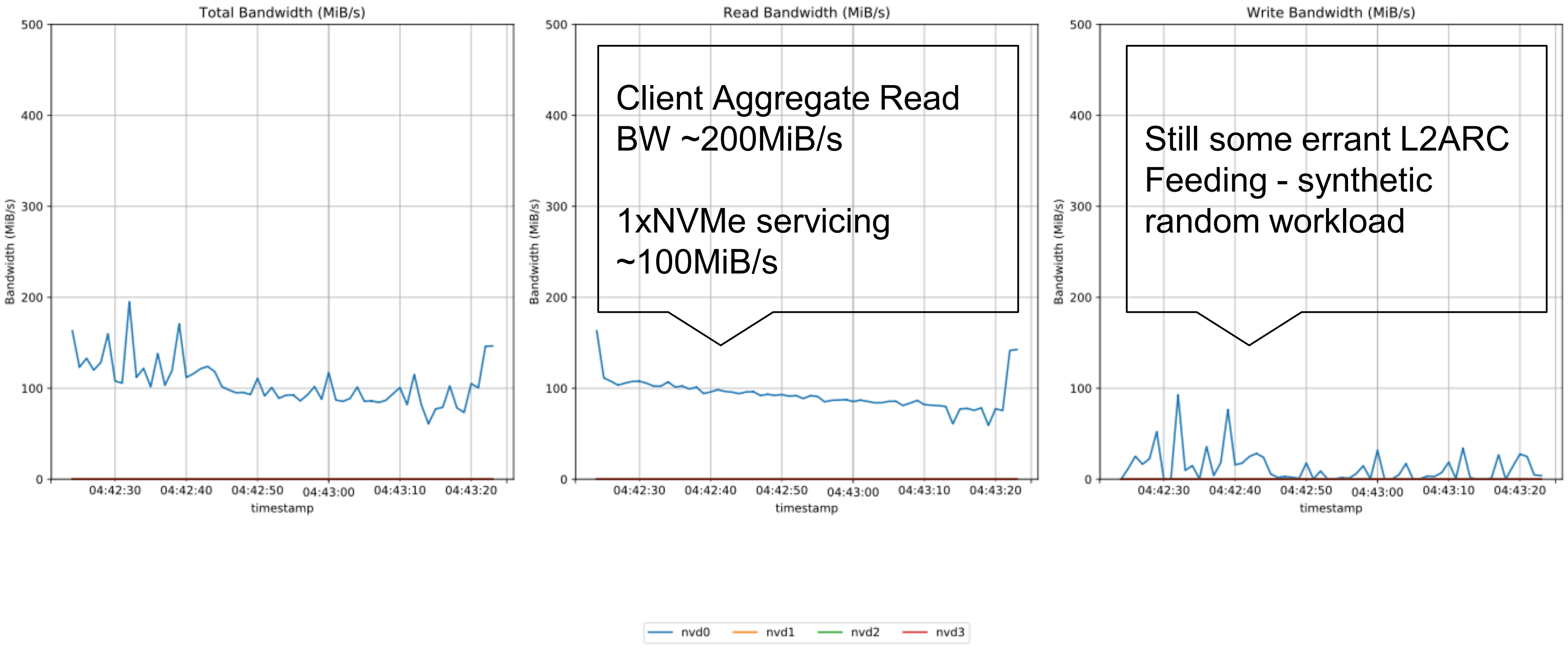
OPS Achieved - Thread Scaling



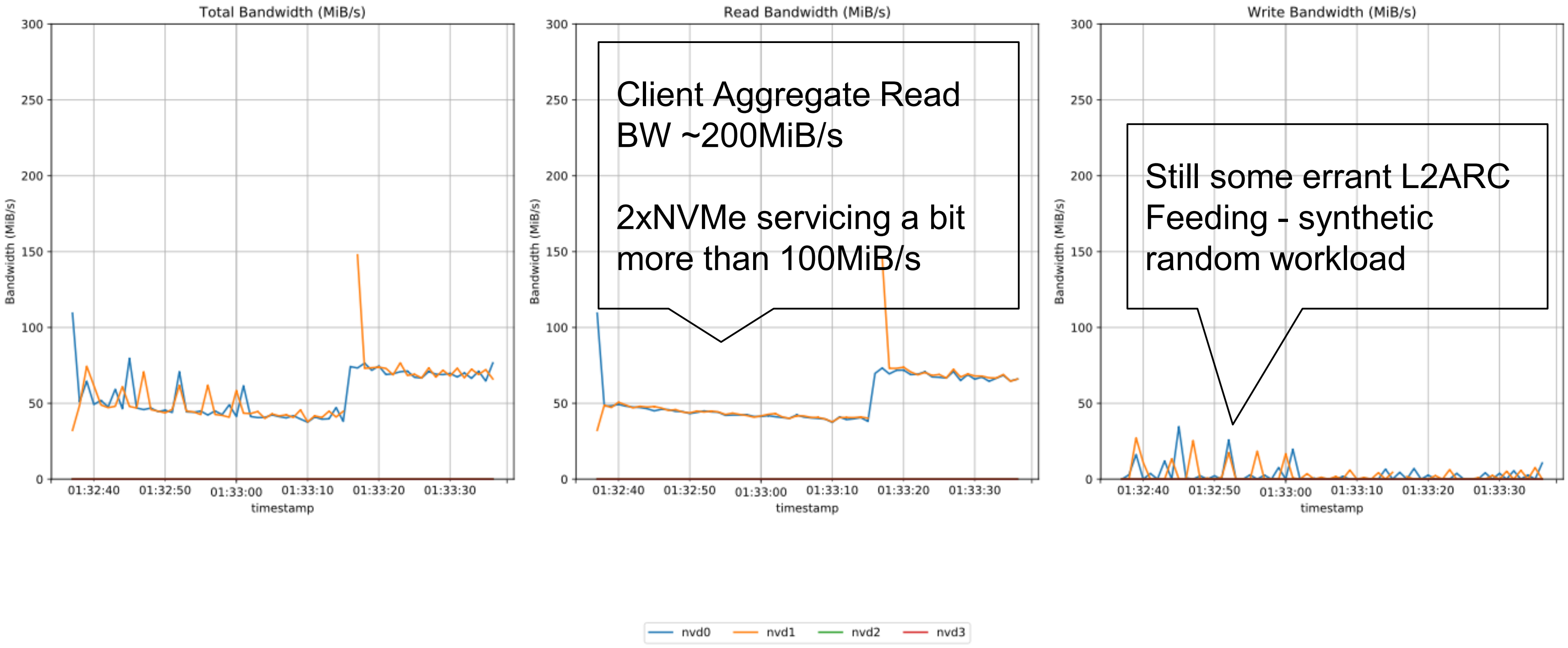
Avg. R/T (ms) - Thread Scaling



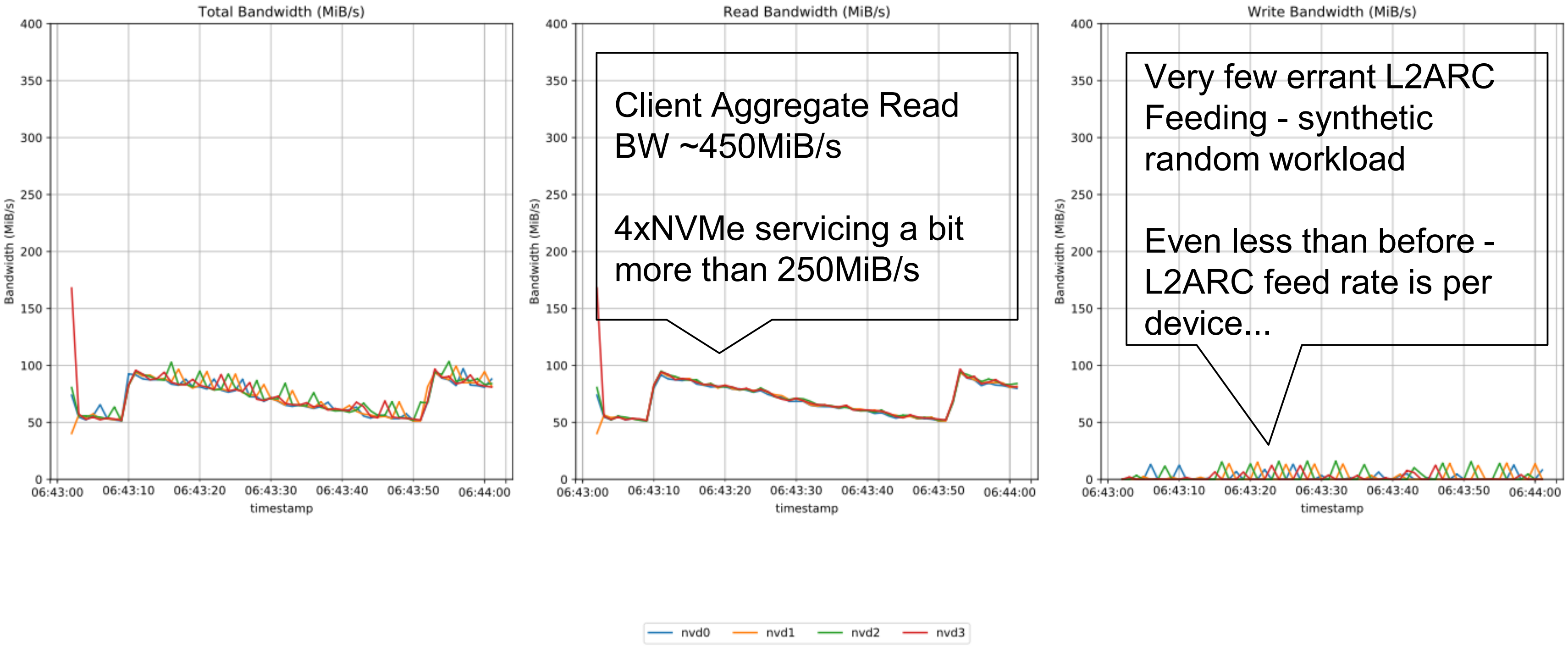
NVMe Activity - 1 in L2ARC
16 Thread Load Point ("best")



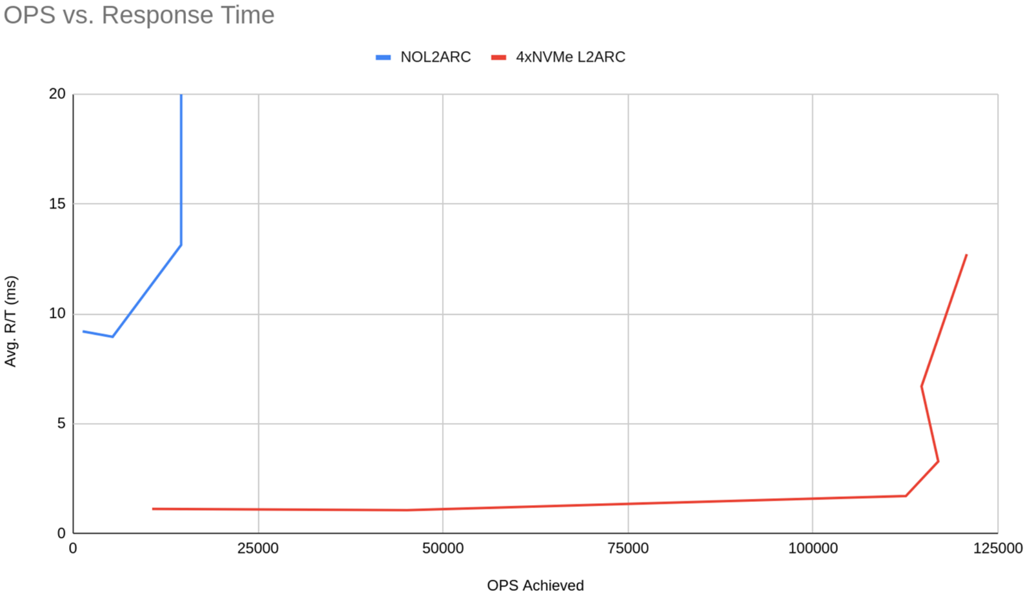
NVMe Activity - 2 in L2ARC
16 Thread Load Point ("best")



NVMe Activity - 4 in L2ARC
16 Thread Load Point ("best")



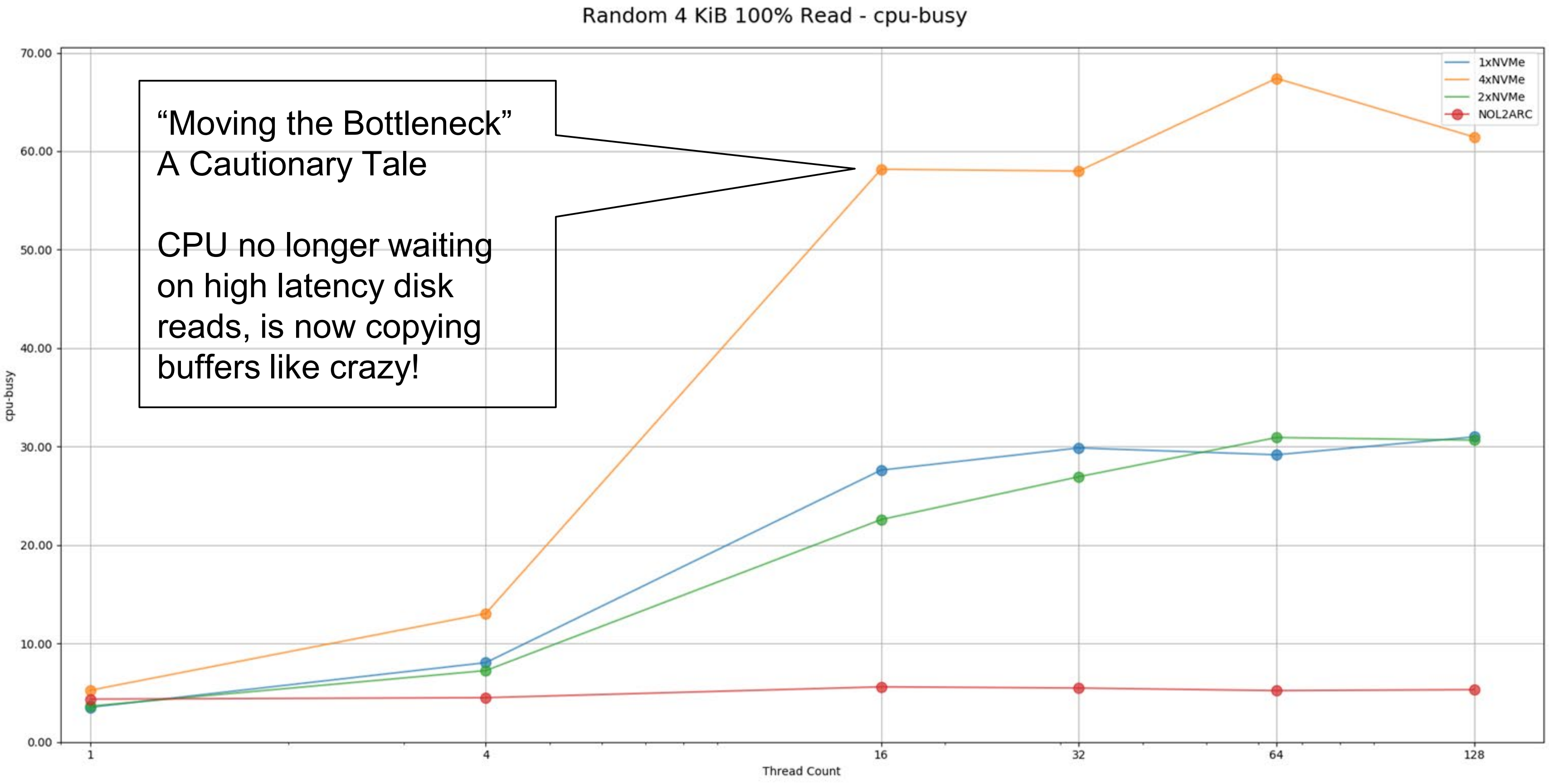
L2ARC Effectiveness



16 Threads Measurement Period ("best")

	NOL2ARC	4xNVMe
OPS Achieved	14610.2333	112526.2694
% increase OPS		670.19%
Avg. R/T (ms)	13.139	1.7057
% decrease R/T		-670.30%

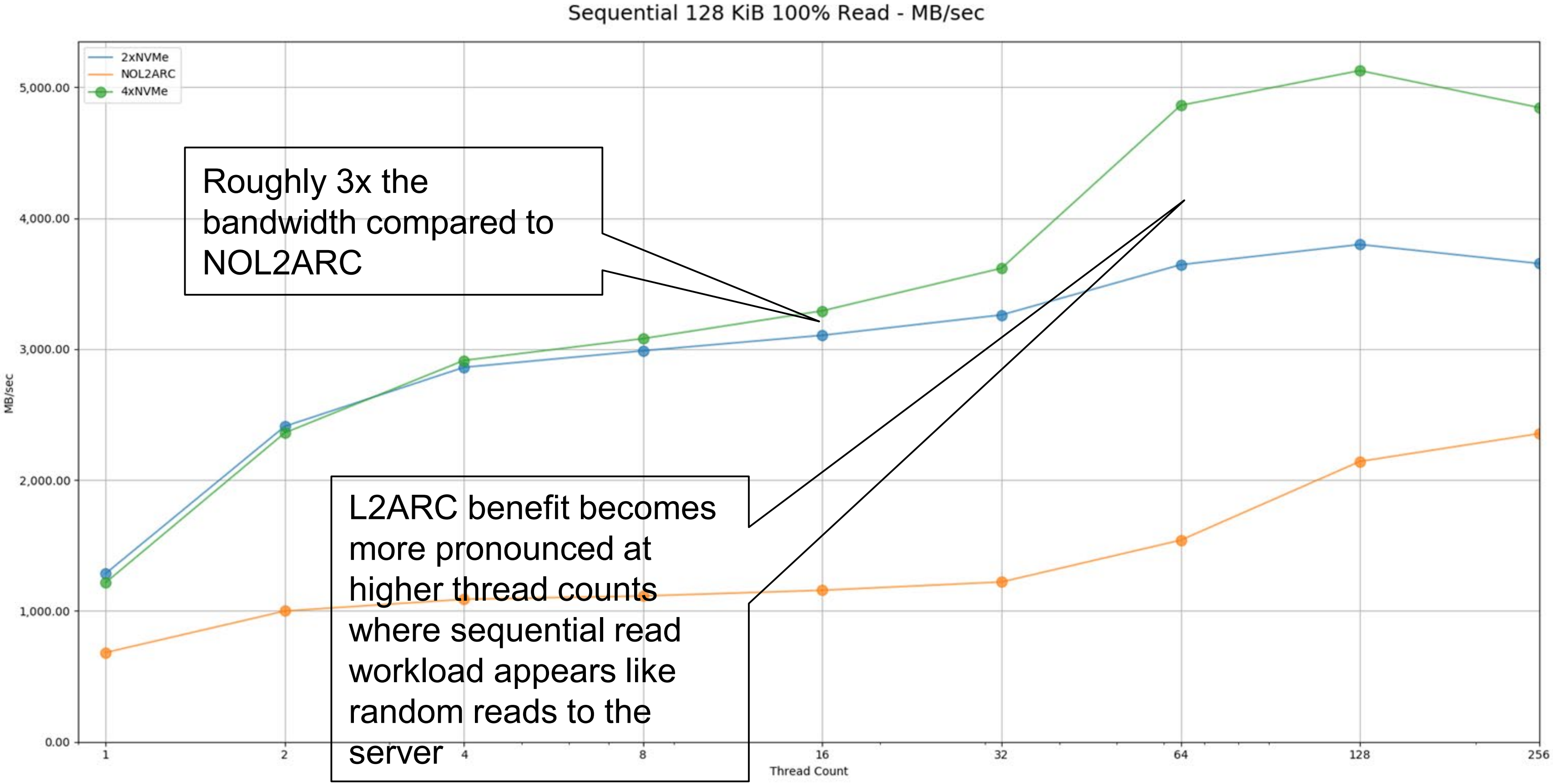
Server CPU %Busy - Avg. of All Cores



Sequential Read-Heavy

- 128K IO Size, Sequential, 100% Read
- 1xNVMe results omitted (invalid)
  - No system time available to re-run

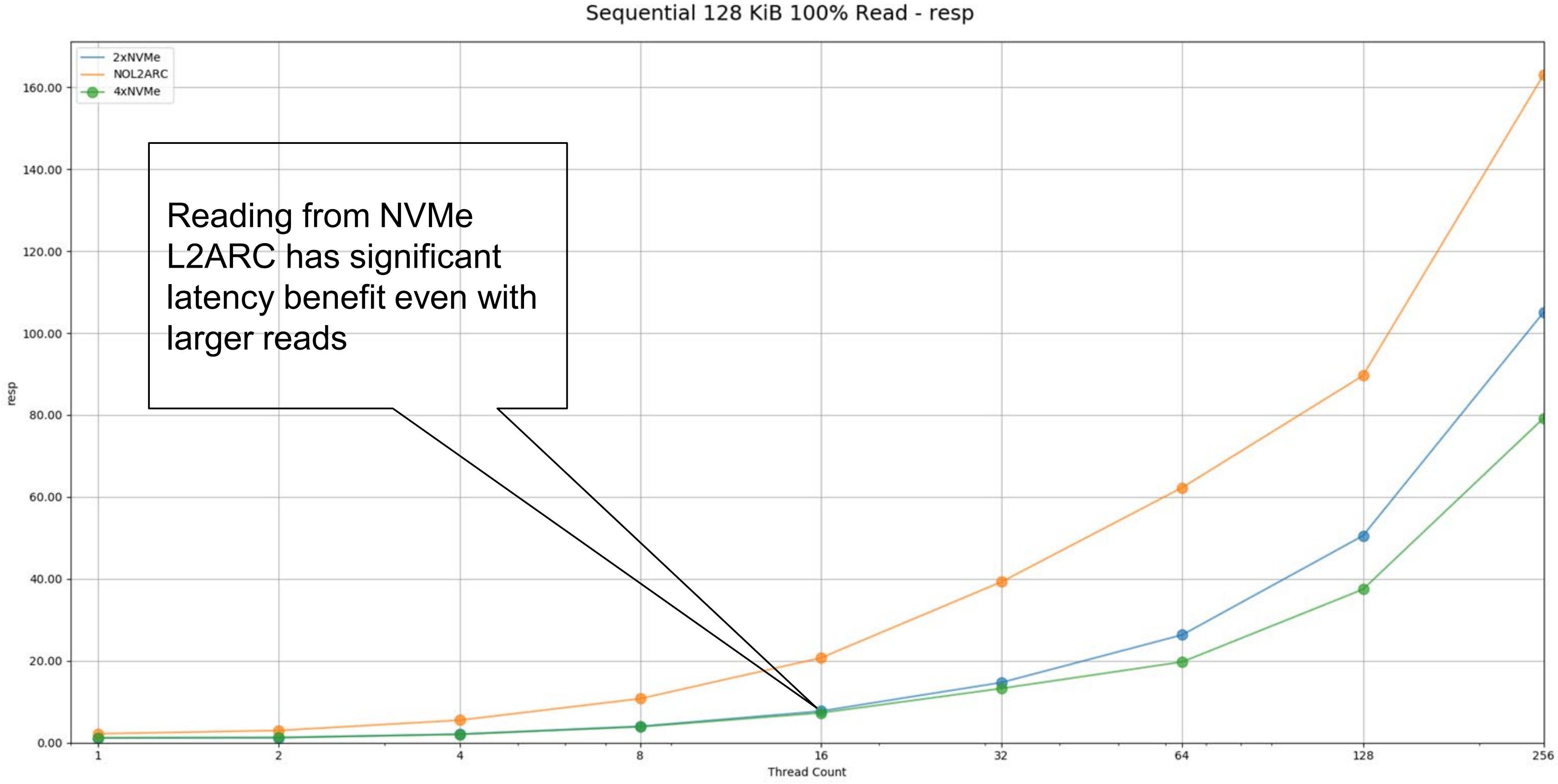
Bandwidth - Thread Scaling



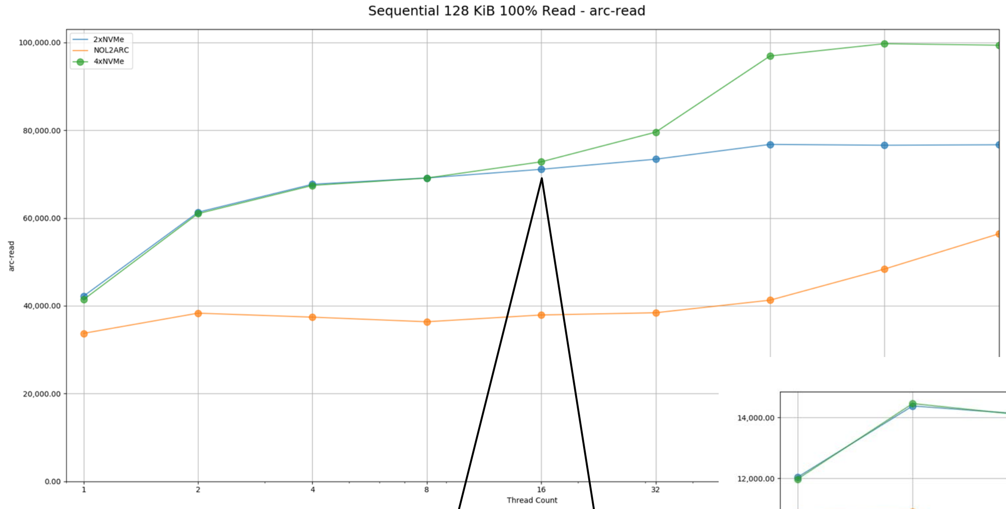
Roughly 3x the bandwidth compared to NOL2ARC

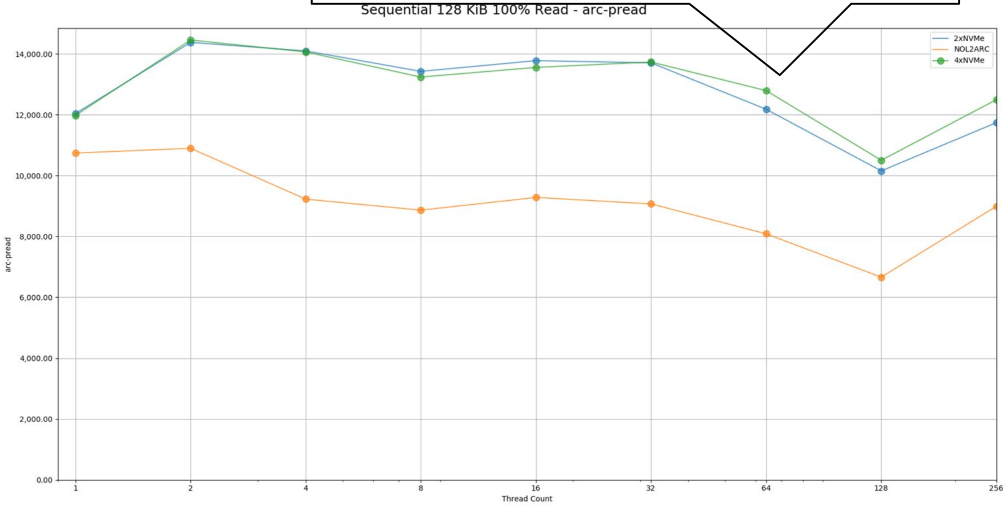
L2ARC benefit becomes more pronounced at higher thread counts where sequential read workload appears like random reads to the server

Avg. R/T (ms) - Thread Scaling



ARC Prefetching





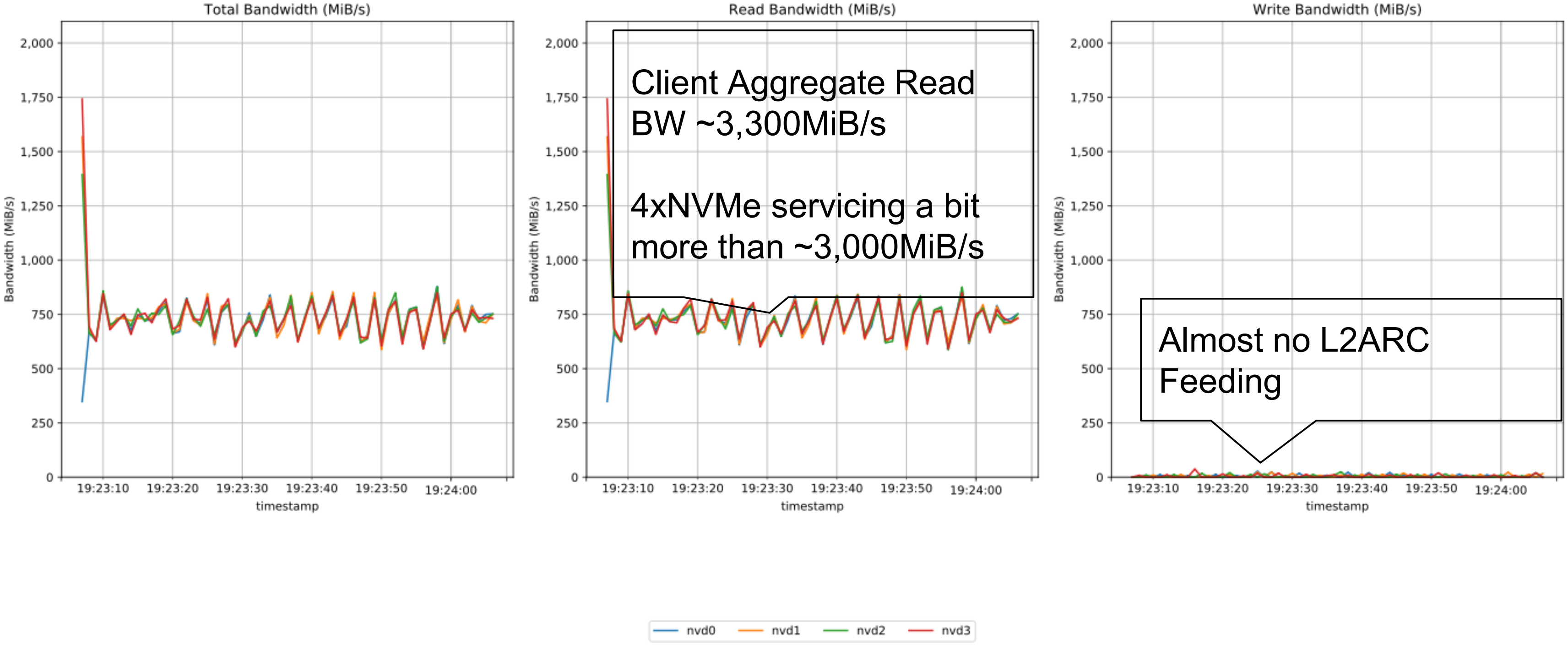
As we add more threads/clients the sequential read requests from them begin to arrive interleaved randomly at the server

Prefetch ARC reads consist of about 20% of Total ARC reads

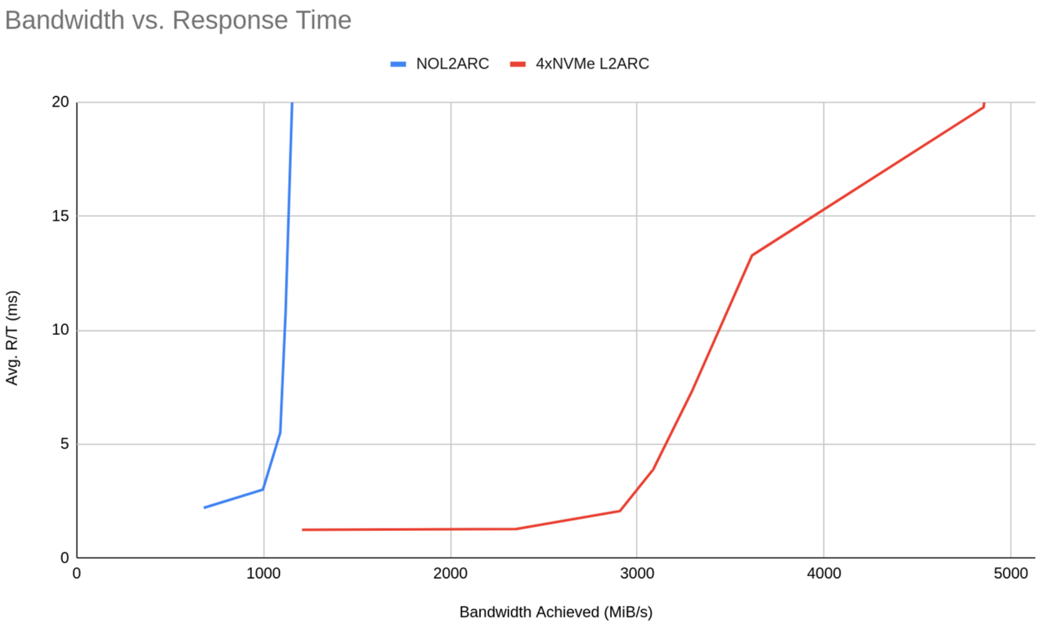
So ZFS only marks about 20% of our accesses as “streaming”

NVMe Activity - 2 in L2ARC 16 Thread Load Point ("best")

NVMe Activity - 4 in L2ARC 16 Thread Load Point ("best")



L2ARC Effectiveness



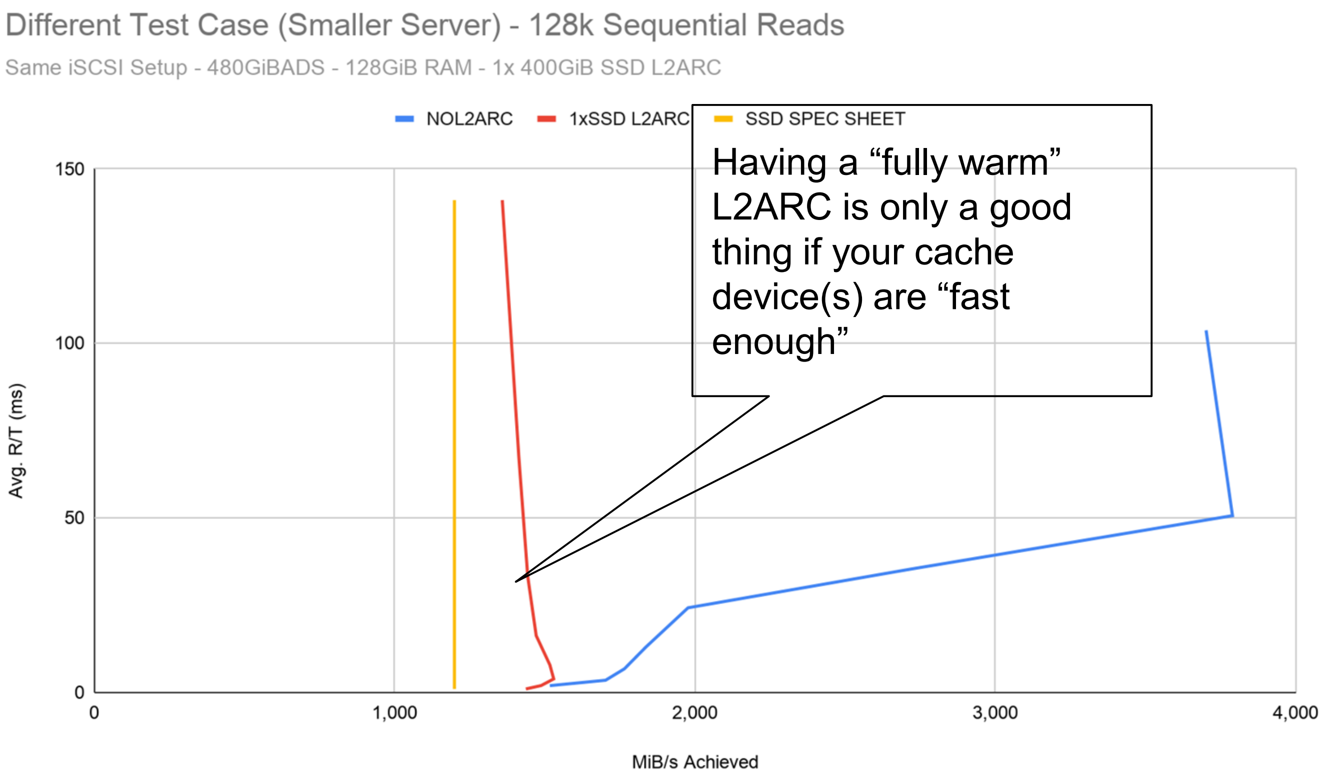
16 Threads Measurement Period ("best")

	NOL2ARC	4xNVMe
MiB/s Achieved	1155.3823	3290.7132
% increase MiB/s		184.82%
Avg. R/T (ms)	20.7702	7.2923
% decrease R/T		-64.89%

64 Threads Measurement Period ("max")

	NOL2ARC	4xNVMe
MiB/s Achieved	1545.3837	4852.7472
% increase MiB/s		214.02%
Avg. R/T (ms)	62.1173	19.7816
% decrease R/T		-68.15%

“All my data fits in L2ARC!” A Cautionary Tale



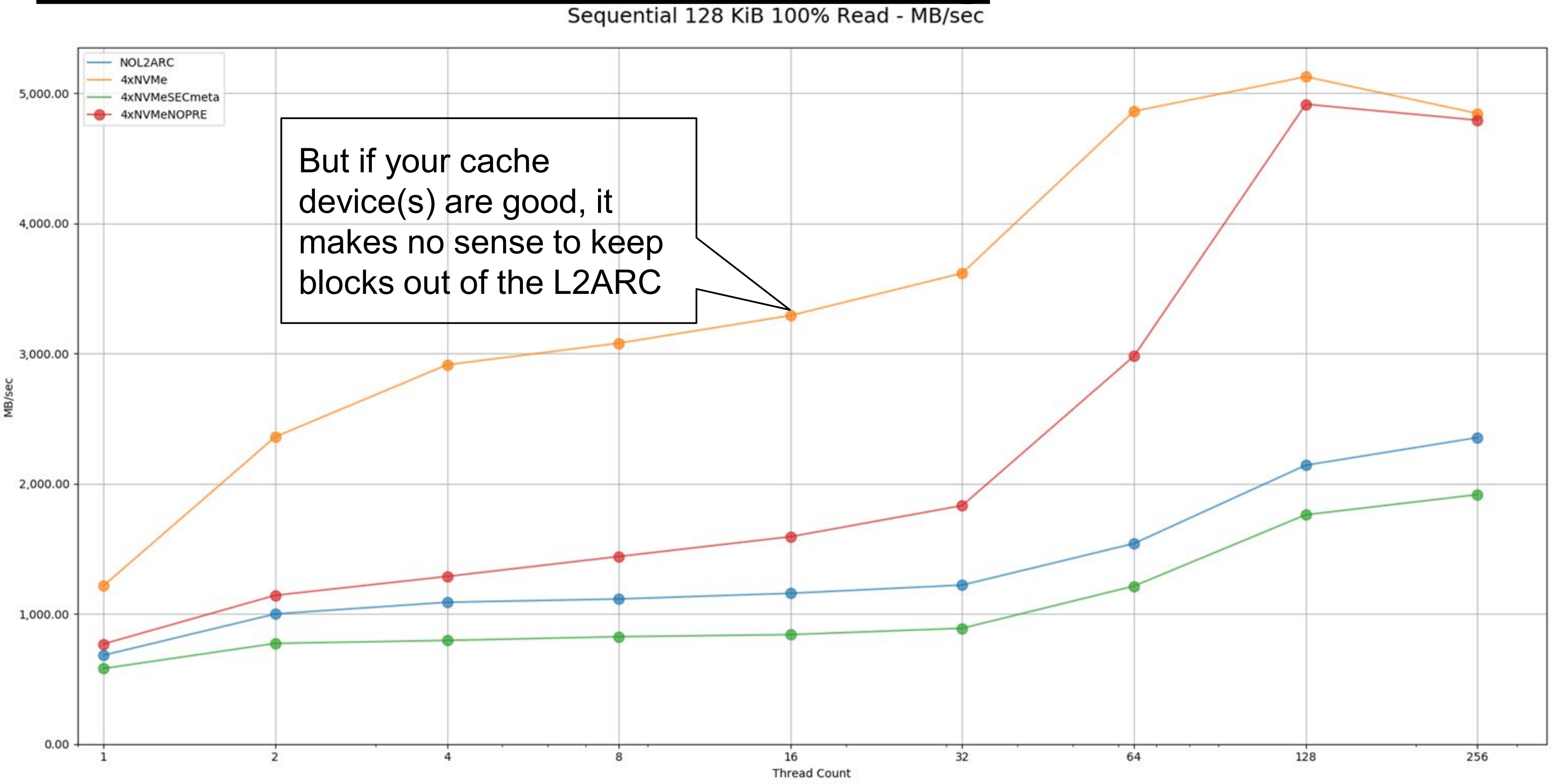
L2ARC with Sequential Reads

- ❑ vfs.zfs.l2arc_noprefetch=0
  - As tested on this server
  - ⇒ blocks that were put into ARC by prefetcher are eligible for L2ARC
- ❑ Let’s test vfs.zfs.l2arc_noprefetch=1
  - ⇒ blocks that were put into ARC by prefetcher are NOT L2ARC-eligible

L2ARC with Sequential Reads

- ❑ Let’s test secondarycache=metadata
  - ○ Only metadata blocks from ARC are are eligible for L2ARC
- ❑ Both of these are strategies to keep L2ARC from Feeding certain blocks
  - ○ With older/slower/fewer L2ARC devices this made sense...

Bandwidth - Thread Scaling



Summary for Write-Heavy Workloads

- 4K IO Size, Pure Random, 100% Write
  - L2ARC constantly feeding - writes causing memory pressure
  - Up to 20% reduction in OPS vs. NOL2ARC
  - Not worth tuning for mitigation
    - Pure writes are rare in practice
    - Trying a 50/50 read write mix (which is still more writes than typical random small block use cases) and all L2ARC configs beat NOL2ARC
- 128K IO Size, Sequential, 100% Write
  - Bandwidth differences of less than 10% +/- over NOL2ARC
    - “In the Noise” - meets design goal of “do no harm”
    - Trying a 50/50 mix again shows benefit on all L2 configs

Agenda

- ✓ Brief Overview of OpenZFS ARC / L2ARC
- ✓ Key Performance Factors
- ✓ Existing “Best Practices” for L2ARC
  - ✓ Rules of Thumb, Tribal Knowledge, etc.
- ✓ NVMe as L2ARC
  - ✓ Testing and Validation
- ❑ **Revised “Best Practices”**

Key Sizing/Config Metrics

- Key Ratio 1:
  ADS / (ARC Max + L2ARC Size)
  - <= 1 means your workload's active dataset will *likely* be persistently cached in ARC and/or L2ARC
- Key Ratio 2:
  (aggregate bandwidth of L2ARC devices) / (agg. bw. of data vdevs)
  - > 1 means your L2ARC can stream sequential data faster than all the data vdevs in your pool

L2ARC Device Selection

SDC 19

- Type ⇒ Bandwidth/OPS and Latency Capabilities
  - Consider devices that fit your budget and use case.
  - Random Read Heavy Workloads can benefit for almost any L2ARC device that is faster than the devices in your data vdevs
  - Sequential/Streaming Workloads will need very fast low latency devices
- Segregated pools vs. mixed use pool with segregated datasets

L2ARC Device Selection

SDC 19

- Capacity
  - Larger L2ARC devices can hold more of your active dataset, but will take longer to “fully warm”
    - For Random Read-Heavy Workloads, GO BIG
    - For Sequential Read-Heavy Workloads, only go big if your devices will have enough bandwidth to benefit vs. your data vdevs
  - Indexes to L2ARC data reside in ARC headers
    - 96 bytes per block (reclaimed from ARC)
    - 256 bytes per block (still in ARC also)
    - Run “top”, referred to as “headers”, small ARC and HUGE L2ARC is probably not a good idea…

L2ARC Device Selection

- Device Count
  - Our testing shows that multiple L2ARC devices will “share the load” of servicing reads, helping with latency
  - L2ARC Feeding rate is per device, so more devices can help warm faster (or have a higher performance impact if constantly feeding)

L2ARC Feeding

- What to Feed?
    - Random Read-Heavy Workload: feed everything
    - Sequential Read-Heavy Workloads
        - "Slow L2": Demand Feed - l2arc_noprefetch=1 (global)
        - "Fast L2": feed everything
    - Write-Heavy Workloads
        - "Slow L2": Feed Nothing - segregated pool
            - or secondarycache=none (per dataset)
        - "Fast L2": Feed Metadata - secondarycache=metadata (per dataset)
            - Avoids constant L2 writes but benefits metadata reads

The Best Practice

- Know your workload
  - For customers, know your top level application(s) and if possible, underlying factors such as access pattern, block size, and active dataset size
  - For vendors, get as much of the above information as you can
- Know your solution
  - For vendors, know how you solutions’s architecture and features interact with different customer workloads
    - (S31) So, You Want to Build a Storage Performance Testing Lab? - Nick Principe
  - For customers, familiarize yourself with this from the vendor’s sales engineers and support personnel that you work with

Big Picture

- With enterprise grade NVMe devices as L2ARC, we have reached the “future” the Brendan Gregg mentions in his blog and in the arc.c code:
  - Namely, L2ARC can now be an effective tool for improving the performance of streaming workloads

Agenda

- ✓ Brief Overview of OpenZFS ARC / L2ARC
- ✓ Key Performance Factors
- ✓ Existing “Best Practices” for L2ARC
  - ✓ Rules of Thumb, Tribal Knowledge, etc.
- ✓ NVMe as L2ARC
  - ✓ Testing and Validation
- ✓ Revised “Best Practices”

Thank You!

Questions and Discussion

Ryan McKenzie
iXsystems

References

- module/zfs/arc.c
  - https://github.com/zfsonlinux/zfs
- “ARC: A Self-Tuning, Low Overhead Replacement Cache”
  - Nimrod Megiddo and Dharmendra S. Modha
  - 2nd USENIX COnference on File Storage Technologies (2003)
  - https://www.usenix.org/conference/fast-03/arc-self-tuning-low-overhead-replacement-cache
- “Activity of the ZFS ARC”, Brendan Gregg’s Blog (January 9, 2012)
  - http://dtrace.org/blogs/brendan/2012/01/09/activity-of-the-zfs-arc/
- “ZFS L2ARC”, Brendan Gregg’s Blog (July 22, 2008)
  - http://www.brendangregg.com/blog/2008-07-22/zfs-l2arc.html
- “FreeBSD Mastery: Advanced ZFS”
  - Allan Jude and Michael W. Lucas
  - Tilted Windmill Press, 2016
  - ISBN-13: 978-1-64235-001-2
  - https://www.amazon.com/FreeBSD-Mastery-Advanced-ZFS/dp/164235001X