Elevating Commodity storage with the SALSA host translation layer

Nikolas Ioannou, Kornilios Kourtis, Ioannis Koltsidas

IBM Research – Zurich

MASCOTS 2018

Storage diversity

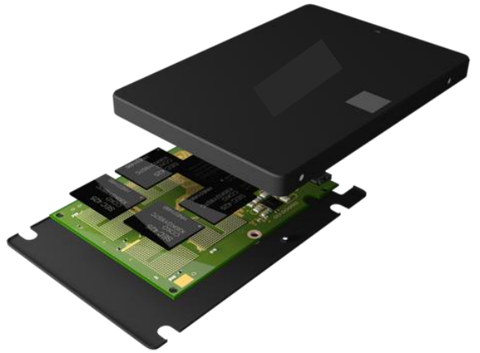
- Multiple technologies offering different levels of performance at different costs

- Technologies
  - SMRs
  - HDDs
  - NAND-Flash SSDs
  - NVM

- Systems might require a combination of different technologies to meet their requirements
- NAND-flash and SMRs are idiosyncratic

Idiosyncratic media

- No random updates
- NAND Flash
  - Pages need to be erased before programmed
  - Erase is performed in blocks of 100s or 1000s of pages

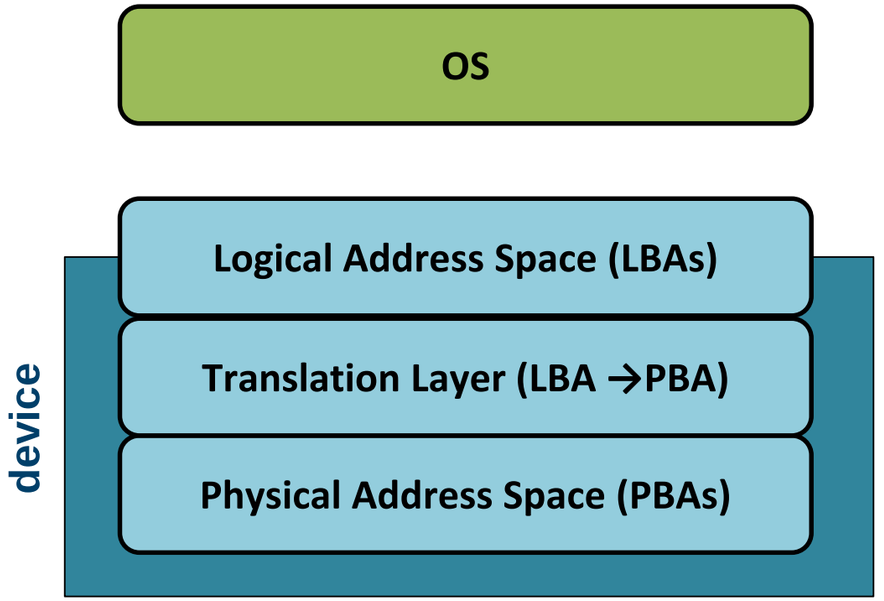


- SMR drives fit more tracks in the surface of a disk, but can only be written sequentially
  - Drives are split into zones

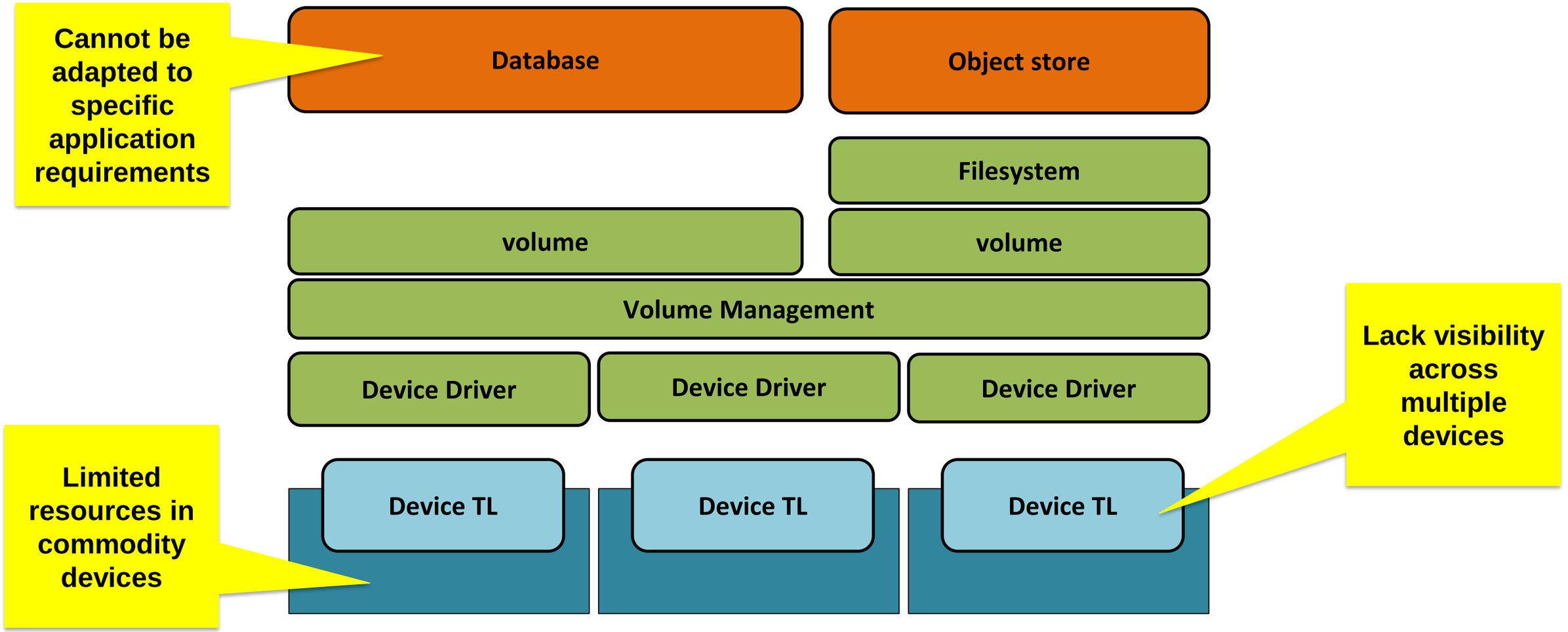
Device Translation Layer (TL)

- OS and applications are not (traditionally) built to handle idiosyncratic
- Device comes with a translation layer that present to software a device that can perform updates to random locations

- Translation layer
  - LBA to PBA mapping
  - Out-of-place sequential writes
  - Garbage Collection (GC)
  - IO amplification
    - A write to an LBA might cause additions Reads or Writes to the PBA



Device TLs are inefficient

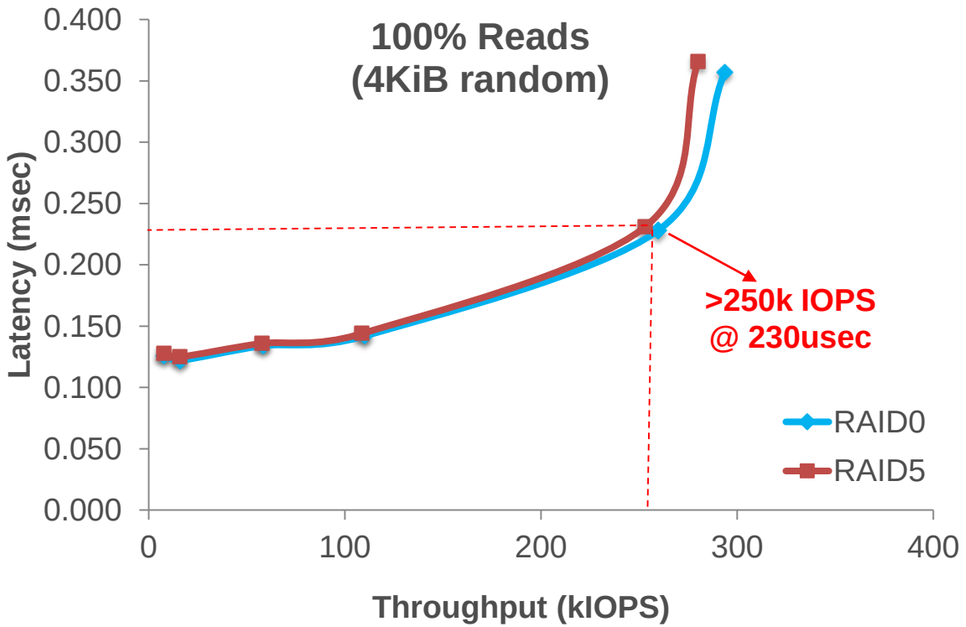


The trouble with low-cost Flash SSDs

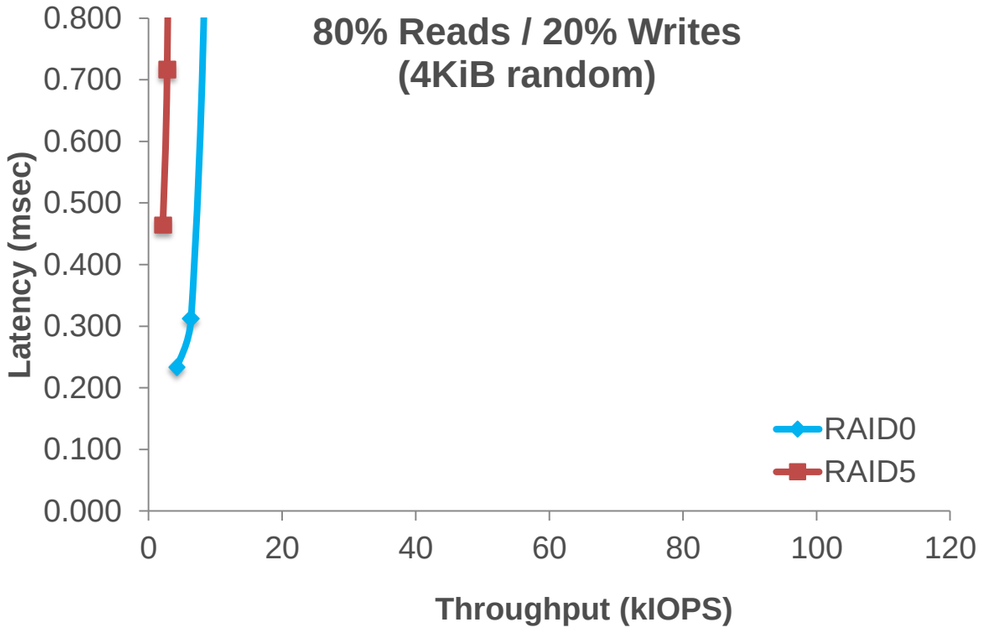
Can't we just use low-cost SSDs?

- Low-cost Flash suffers from high write latency, low endurance
- Limited resources, simple controllers to keep the cost as low as possible (~ $0.23 /GB!)
- Therefore, they only employ simple Flash management
  - Sufficiently good read performance
  - But, **limited write endurance**, **terrible write performance**

Raw low-cost SSDs are of limited use in a datacenter

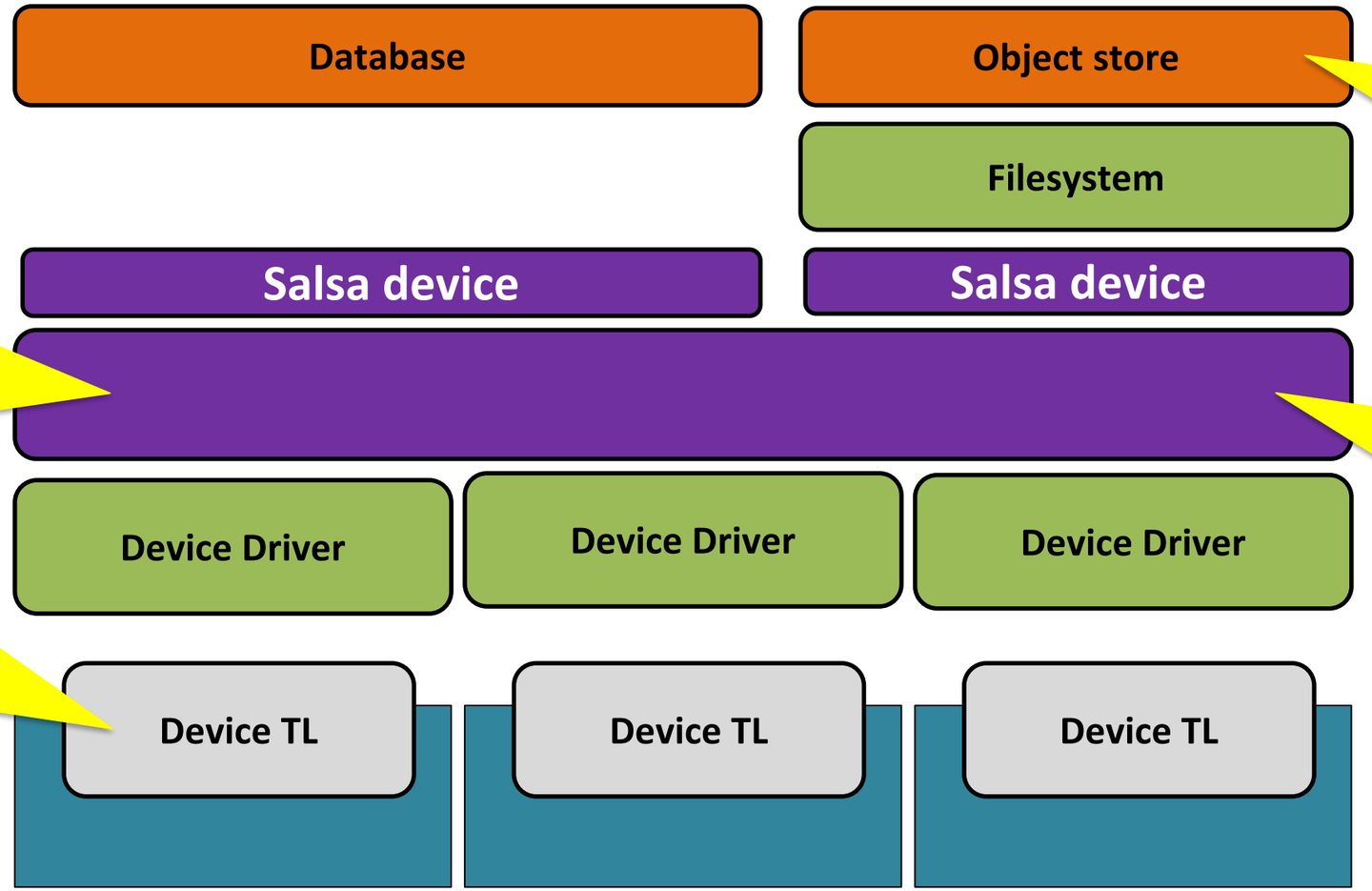


+ 20% writes



fio --filename=/dev/mapper/ssd --runtime=14400s --time_based --ioengine=psync --direct=1 --thread --norandommap --rw=randrw –rwmixread=80 --bs=4k --numjobs=32 --iodepth=1 **--sync=1**

SoftwAre Log Structured Array



Can be adapted and specialized to specific application requirements

SALSA implements its own GC and LBA to PBA mapping. Writes sequentially to devices

Has visibility to all devices and their properties

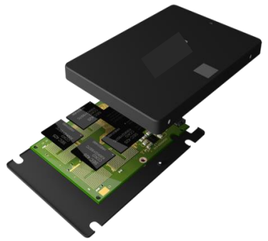
SALSA effectively disables Device TL GC by making its job trivial.

What SALSA can do

SSDs

- Improve performance
- Improve endurance

- Enable commodity SSDs to be used in the data-center.



SMRs

- Host-managed
- Host-aware
- Drive-managed

- Improved performance
- Specialized controller (dual-mapping) for object stores deployed on top of filesystem

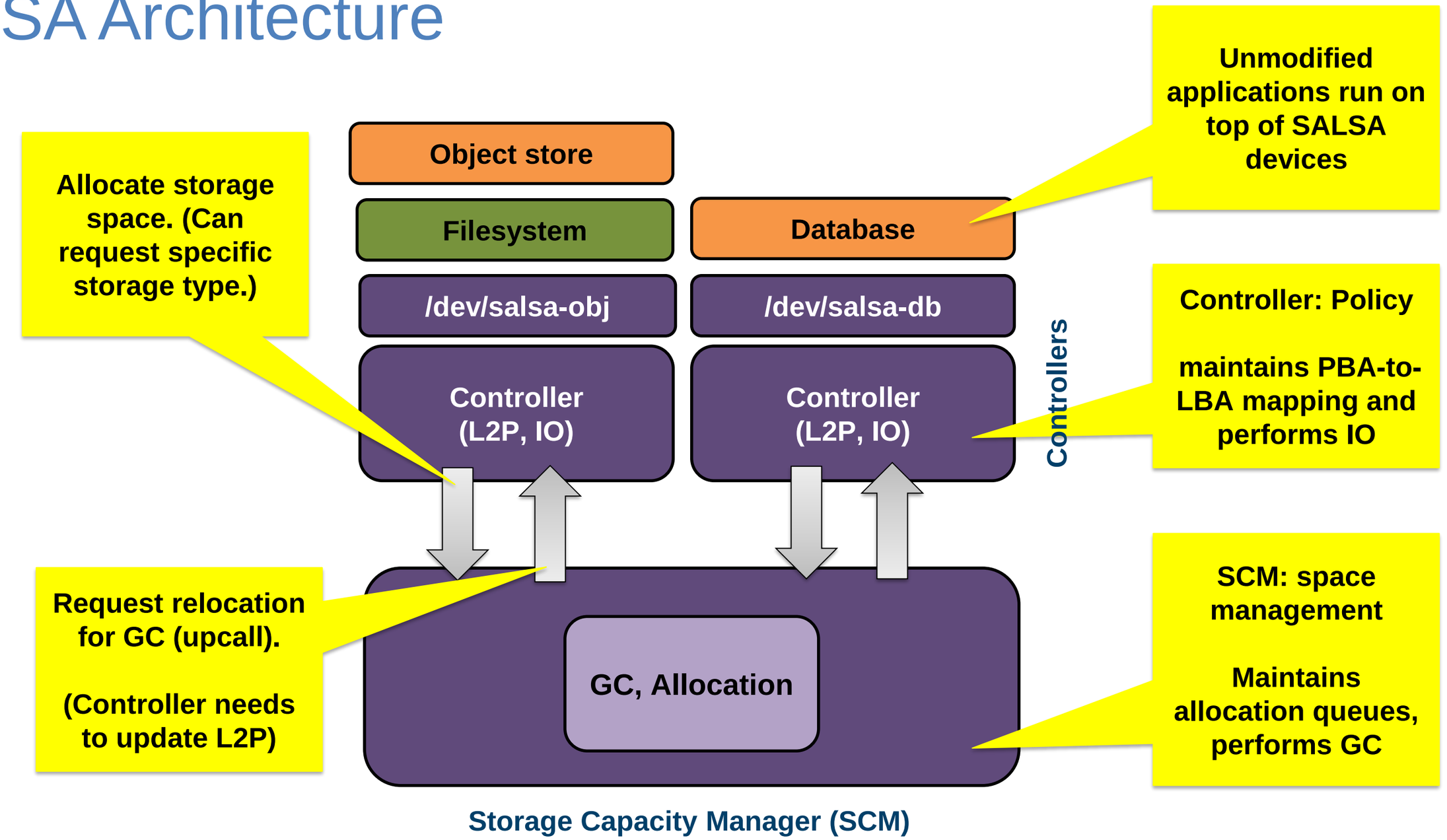
Mixed workloads

- Different application policies on top of a common pool of storage

Combine storage types

- Specialized controller that combines SSDs + SMRs
- Unmodified application (e.g., video server)

SALSA Architecture



SALSA technologies

- SALSA runs on:
  a) Host-managed SSDs/HDDs: SALSA has full control of the drive
  b) Regular SSDs: SALSA *implicitly* forces the SSD controller to not do Garbage Collection
- SALSA implements data placement and garbage collection above the drive

Data Placement

- **Log-Structured data layout**
- **Data Segregation based on write heat**
- **RAID5-equivalent protection without R-M-W**
- Small writes placed in conventional zones (SMR)
- Optimized placement for read-hot data
- Workload isolation & I/O stream separation
- Thin provisioning

Garbage Collection

- **State-of-the-art GC algorithms**
- Recurring pattern detection
- Trim support

Additional features

- Write throttling
- Optional in-memory caching
- Data reduction (experimental)
- RDMA interface

Containerized MySQL (1 device)

- 4 multithreaded MySQL containers deployed over 1 SSD device
- Sysbench to execute an OLTP workload

- partitions over raw device + ext4
- partitions over raw device + f2fs (a log-structured filesystem)
- Salsa devices + ext4

Containerized MySQL (1 device)

RAW			F2FS			SALSA		
tps	avg	95%	tps	avg	95%	tps	avg	95%
22.2	180ms	651ms	25.6	157ms	599ms	37.4	107ms	266ms
21.3	188ms	655ms	25.6	156ms	599ms	37.6	106ms	264ms
21.2	188ms	656ms	25.5	157ms	596ms	37.7	106ms	264ms
21.2	188ms	654ms	25.6	157ms	603ms	39.1	102ms	258ms

- SALSA improves throughput and average latency by **68%** vs raw
- SALSA improves throughput by **47%** and reduces tail latency by **145%** vs FS2FS

Containerized MySQL (RAID)

- Same experiment as before but using 4 SSDs in:

- Linux RAID-5 MD
- SALSA RAID-5 equivalent
  - SALSA can guarantee full-stripe writes with a small persistent buffer

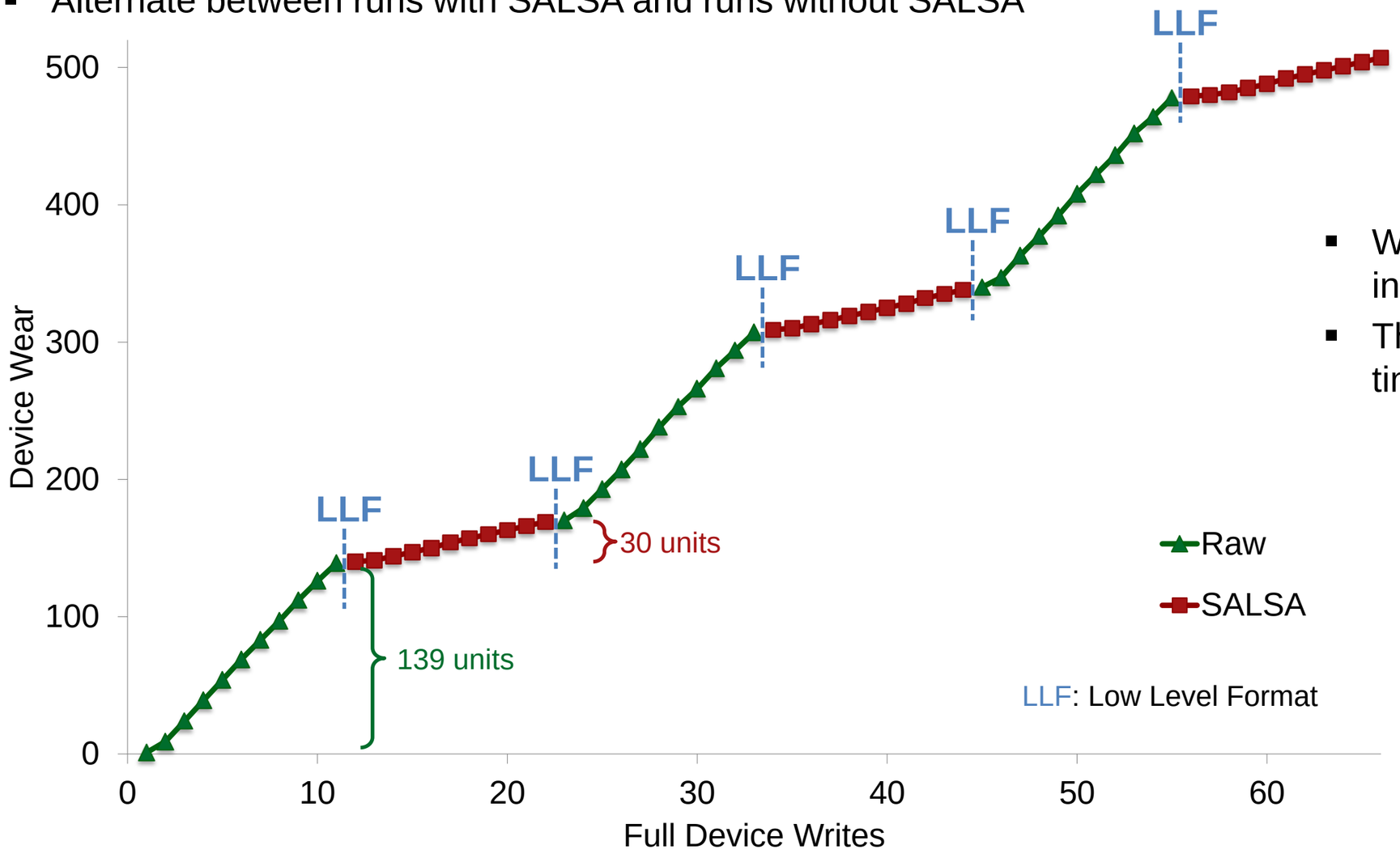
Containerized MySQL (RAID)

Linux MD			SALSA		
tps	avg	95\%	tps	avg	95%
8.1	2.0s	5.3s	287.2	55.7ms	99.5ms
8.1	2.0s	5.3s	290.5	55.1ms	98.4ms
8.3	1.9s	5.2s	286.5	55.9ms	99.9ms
7.8	2.1s	5.6s	291.1	55.0ms	98.2ms

- SALSA improves throughput and average latency by **x35.4** and **x36.8** vs Linux md
- MD has tail latency of seconds!

Endurance improvement

- Multiple runs of 4KB random writes on a Samsung 850 EVO SSD,
- At each iteration (= 10 FDWs) we do a low-level format (LLF) of the device.
- Alternate between runs with SALSA and runs without SALSA



- With SALSA, the same workload incurred **4.6x less device wear**
- The result was **repeatable** over time, and across devices

Write tput (MiB/s)	
Raw	15.9
SALSA	37.7

Object store

- Object store (swift) on a system with potentially 100s of SMR drives
  - Intended for large objects

- Salsa’s mapping is a flat table with 4 bytes per entry
  - 4KiB mappings: 1GiB RAM per 1TiB
  - 64KiB mappings:64MiB RAM per 1TiB
    - But: might cause Read-Modify-Write (RMW) updates

- Initial evaluation with 64KiB mappings showed reduced performance due to RMW operations

Object store: Dual Mapping controller

- Sources for RMW operations:
  - Unaligned data
  - Filesystem metadata
  - Small percentage, but did cause a performance hit

- Dual-mapping controller
  - Sparse mapping for 4KiB pages (hash table)
  - Dense mapping for 64KiB pages
  - Read operations check sparse mapping first
  - Write operations select mapping based on operation size

Object store: Dual Mapping controller

- Evaluate using the swift object store, and an a SMR drive
  - Emulating high load using high concurrency (32 green threads)

- Random PUTs, UPDATEs, and GETs
- 128KiB objects:
  - **x6.4** PUTs
  - **x4.1** UPDATEs

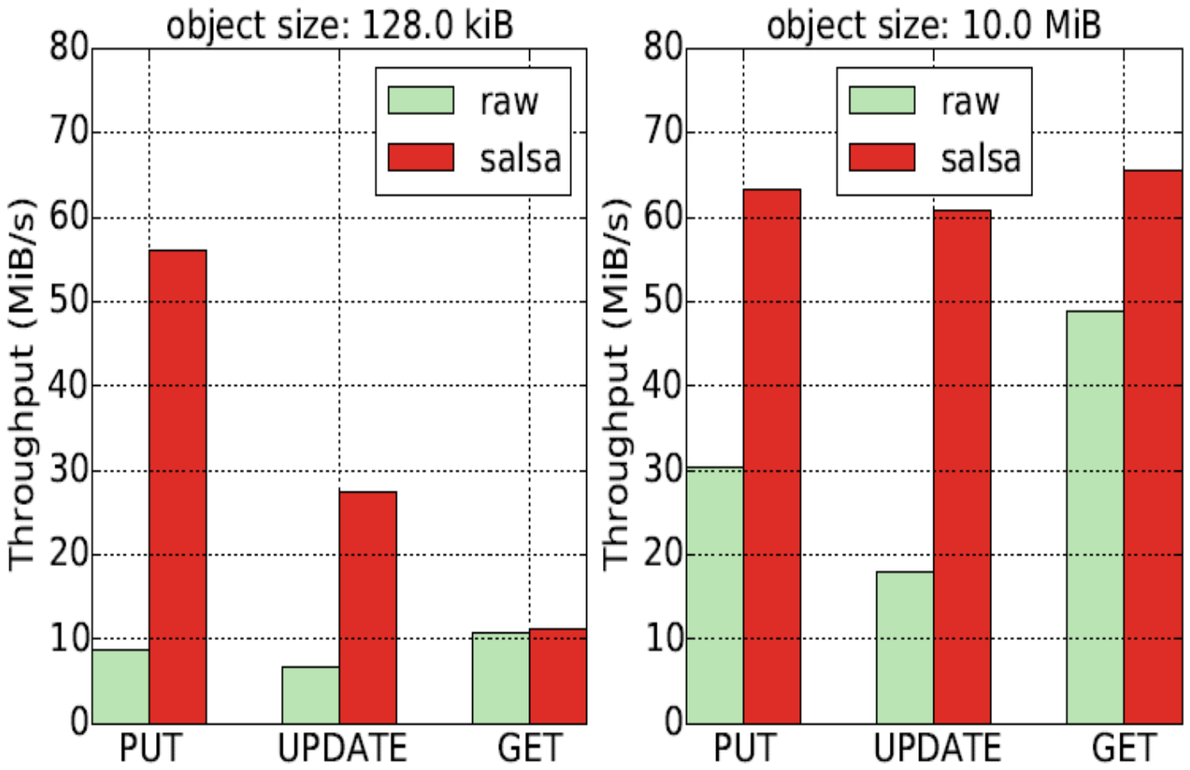


Figure 7: Swift storage server throughput for different operations, comparing the raw device and SALSA.

Hybrid (SMR+SSD) controller for video server

- Video service for user-generated content (e.g., YouTube)
  - Videos are typically short and disks cannot reach their full bandwidth
  - Huge number of videos uploaded daily: requires cost-effective storage (SMRs)
  - Small number of popular videos: can move popular videos into faster storage (SSDs)

- Hybrid Controller
  - SCM: two allocation streams one for Flash and one for SMR
  - Default: SMR
  - “Hot” pages are relocated to Flash
    - Temperature: 64-bit value for every 256 pages
    - Threshold, after which pages are relocated to Flash
    - re-use GC relocation mechanisms
  - No need to modify application

Hybrid (SMR+SSD) controller for video server

- Benchmark: `filebench` with video-server macro-workload
  - Active set: frequently read files
  - Passive set: new files are added

	RD (MiB/sec)	WR (MiB/sec)
raw	4.8	10.1
salsa	6.2	10.1
salsa-hybrid	118.5	10.0

Conclusion

SALSA is a host TL that:

- Improves performance and durability of SSDs
- Improves performance of (all types of) SMRs
- Allows application-specific IO policies
- Allows combining different storage types

Thank you!