About SSD

Dongjun Shin

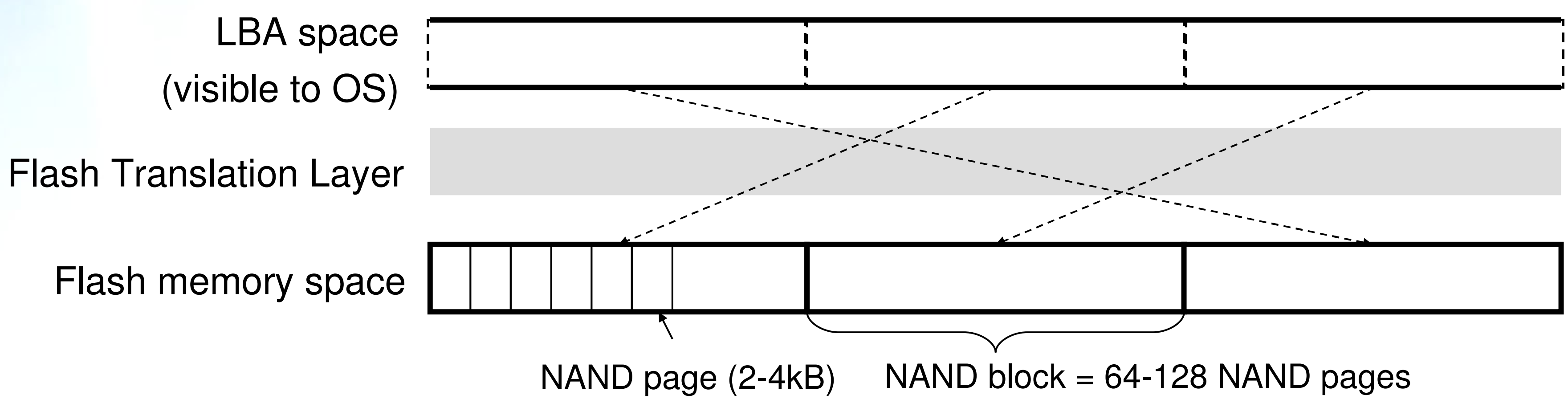
Samsung Electronics

Outline

- SSD primer
- Optimal I/O for SSD
- Benchmarking Linux FS on SSD
- Case study: ext4, btrfs, xfs
- Design consideration for SSD
- What's next?
    - New interfaces for SSD
    - Parallel processing of small I/O

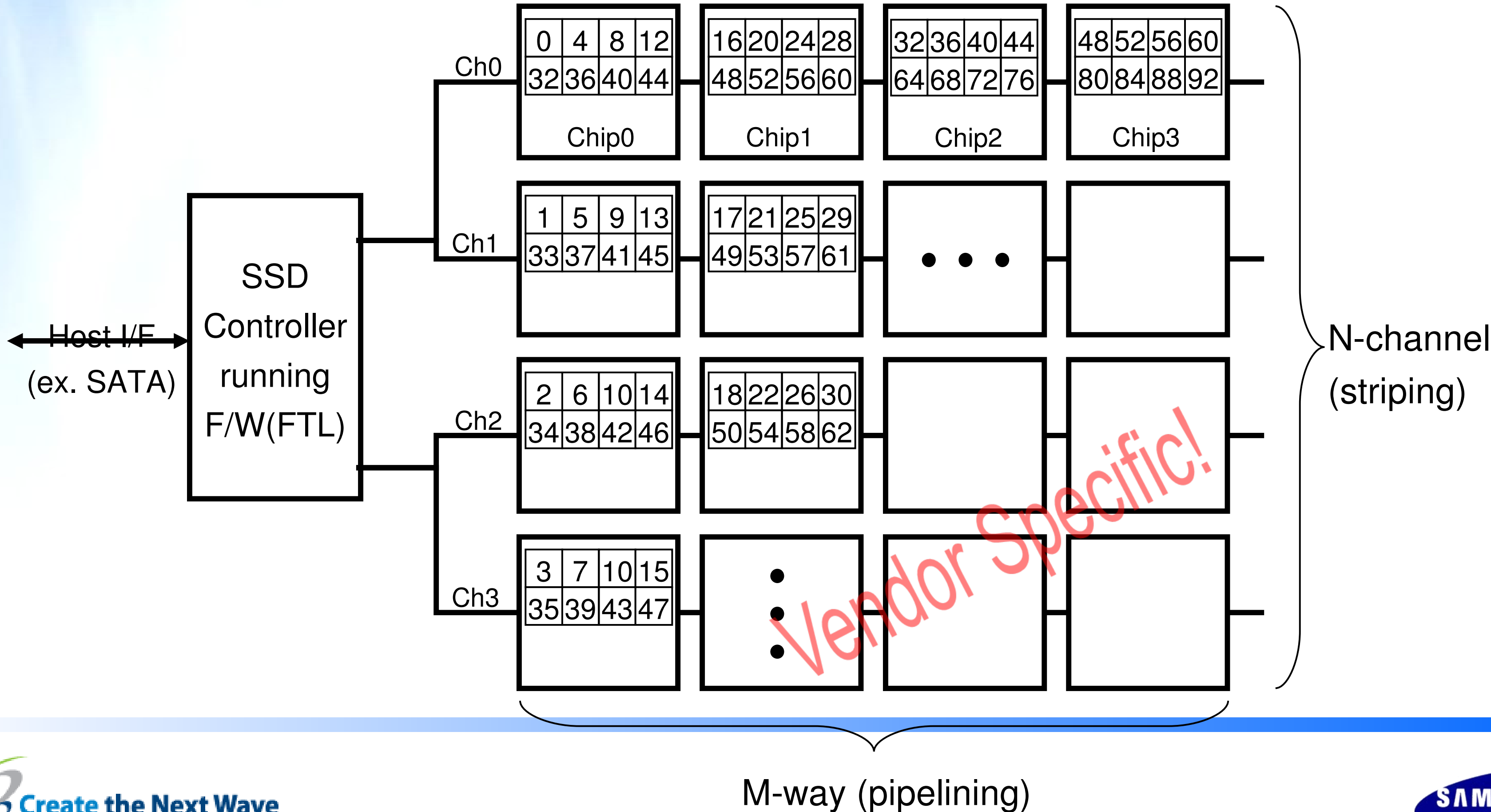
SSD Primer (1/2)

- Physical unit of flash memory
  - $Page_{NAND}$ – unit for read & write
  - $Block_{NAND}$ – unit for erase (a.k.a erasable block)
- Physical characteristics
  - Erase before re-write
  - Sequential write within an erasable block



SSD Primer (2/2)

- Internal organization: 2-dimensional (NxM parallelism)
  - Similar to RAID-0 (stripe size = sector or page$_{NAND}$)
  - Effective page & block size is multiplied by NxM (max)



Optimal I/O for SSD

- Key points
  - Parallelism
    - The larger the size of I/O request, the better
  - Match with physical characteristics
    - Alignment with page or block size of NAND*
    - Segmented sequential write (within an erasable block)

- What about Linux?
  - HDD also favors larger I/O → read-ahead, deferred aggregated write
  - Segmented FS layout → good if aligned with erasable block boundary
  - Write optimization → FS dependent (ex. allocation policy)

* Usually, partition layout is not aligned (1st partition at LBA 63)

Test environment (1/2)

- Hardware
  - Intel Core 2 Duo E6550@2.33GHz, 1GB RAM
- Software
  - Fedora 7 (Kernel 2.6.24)
  - Benchmark: postmark
- Filesystems
  - No journaling - ext2
  - Journaling - ext3, ext4, reiserfs, xfs
    - ext3, ext4: data=writeback,barrier=1[,extents]
    - xfs: logbsize=128k
  - COW, log-structured - btrfs (latest unstable, 4k block), nilfs (testing-8)
- SSD
  - Vendor M (32GB, SATA): read 100MB/s, write 80MB/s
  - Test partition starts at LBA 16384 (8MB, aligned)

Test environment (2/2)

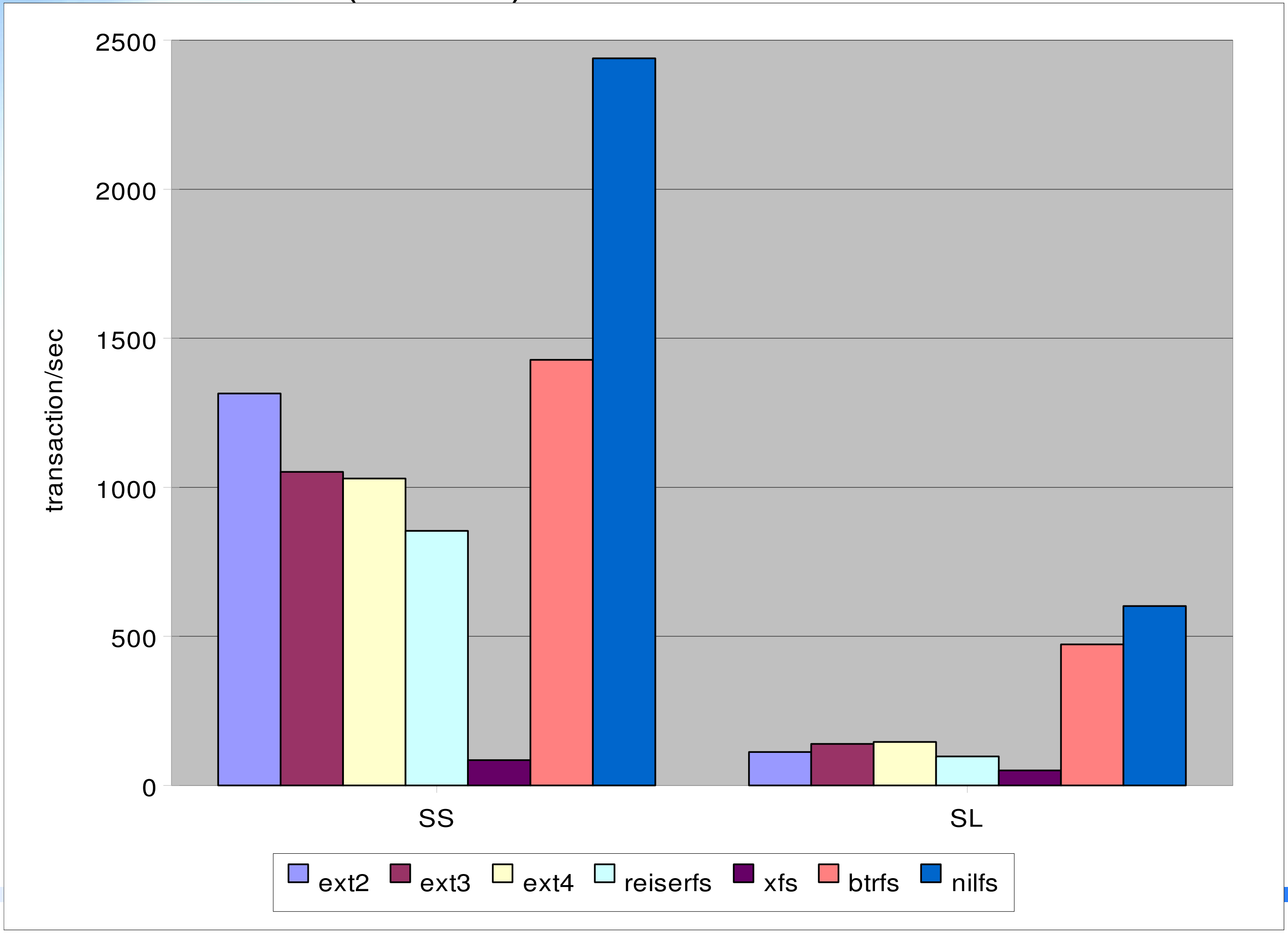
- Postmark workload
  - Ref: Evaluating Block-level Optimization through the IO Path (USENIX 2007)

Workload	File size	# of file (work-set)	# of transaction	Total app read/write
SS	9-15K	10,000	100,000	630M/755M*
SL	9-15K	100,000	100,000	600M/1.8G
LS	0.1-3M	1,000	10,000	9.7G/12G
LL	0.1-3M	4,250	10,000	9G/17G

* Mostly write-only

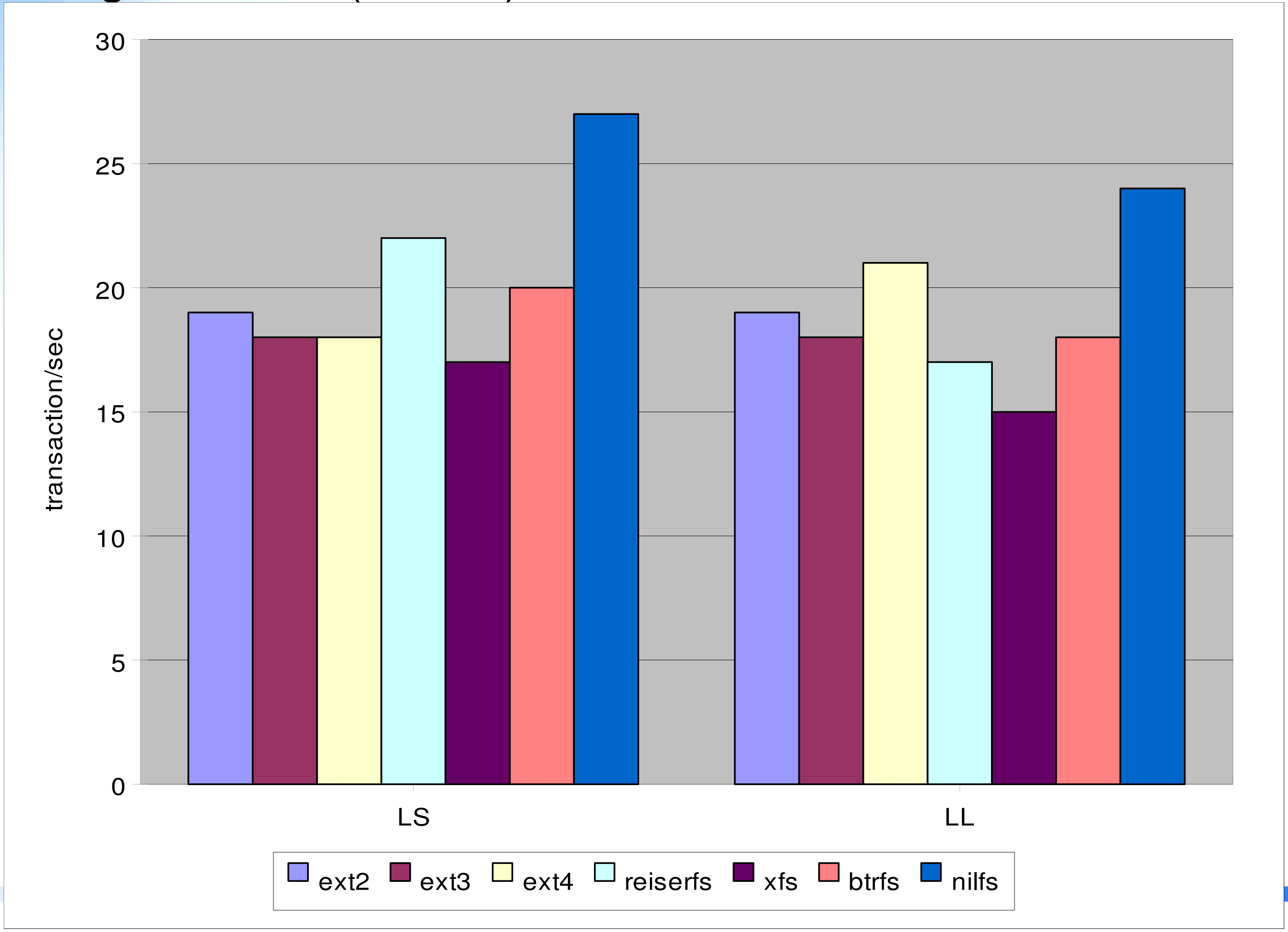
Benchmark results (1/2)

- Small file size (SS, SL)



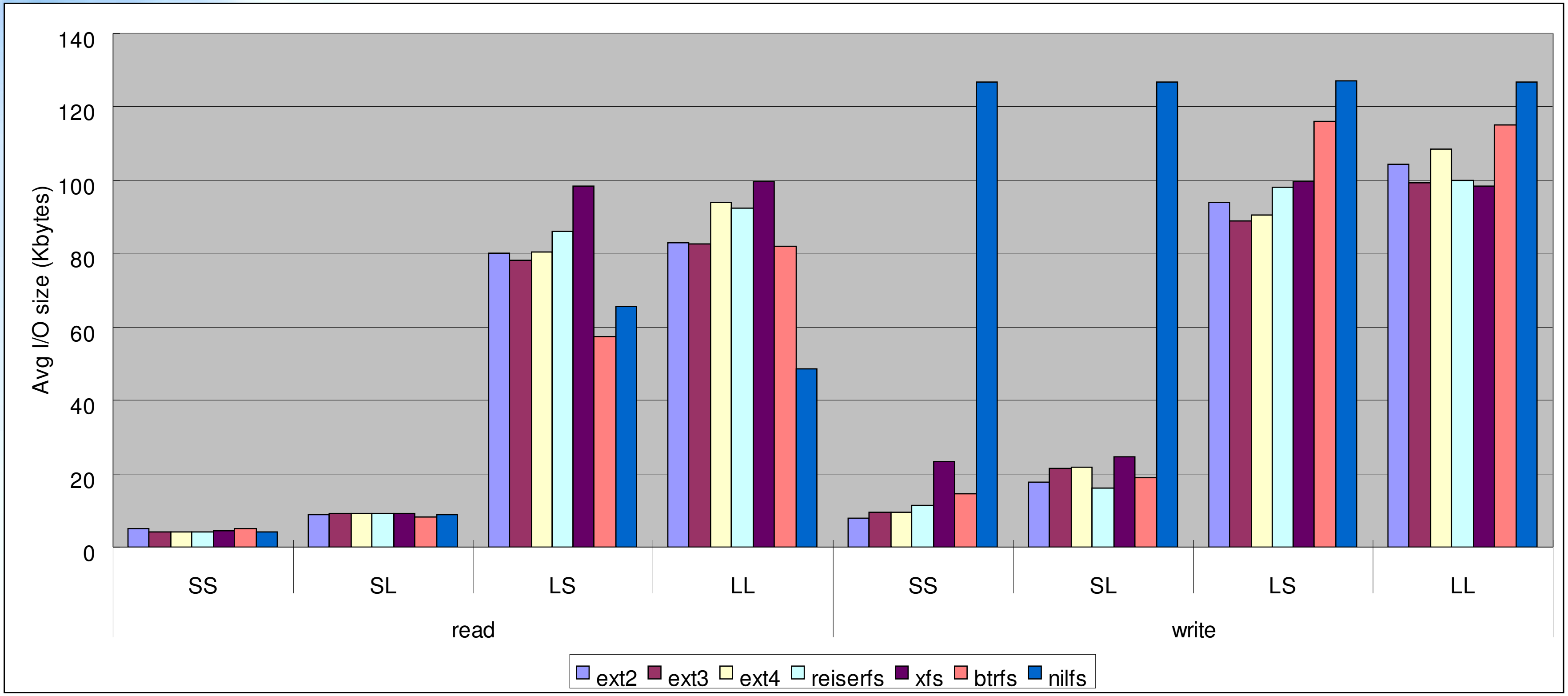
Benchmark results (2/2)

- Large file size (LS, LL)



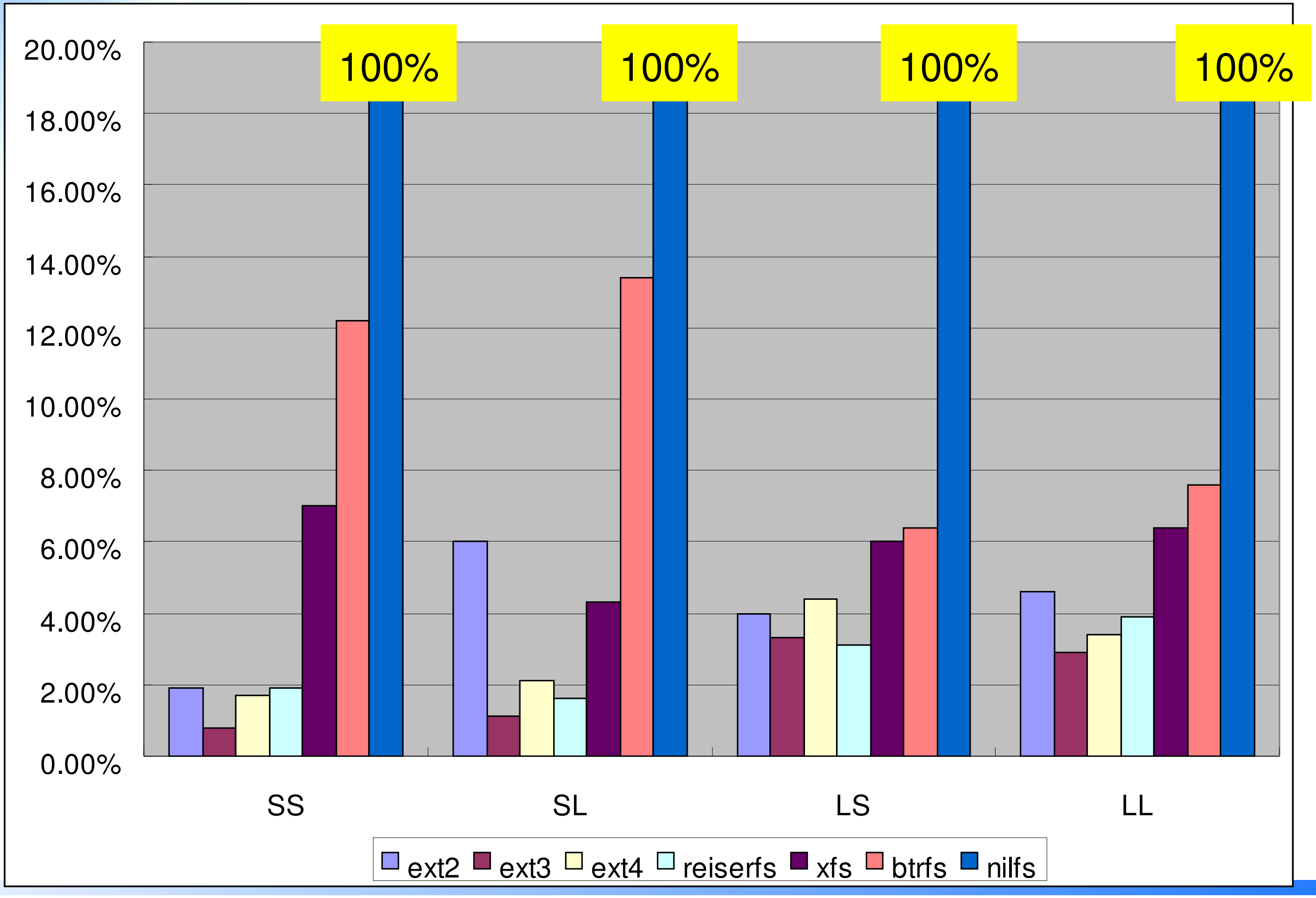
I/O statistics (1/2)

- Average size of I/O



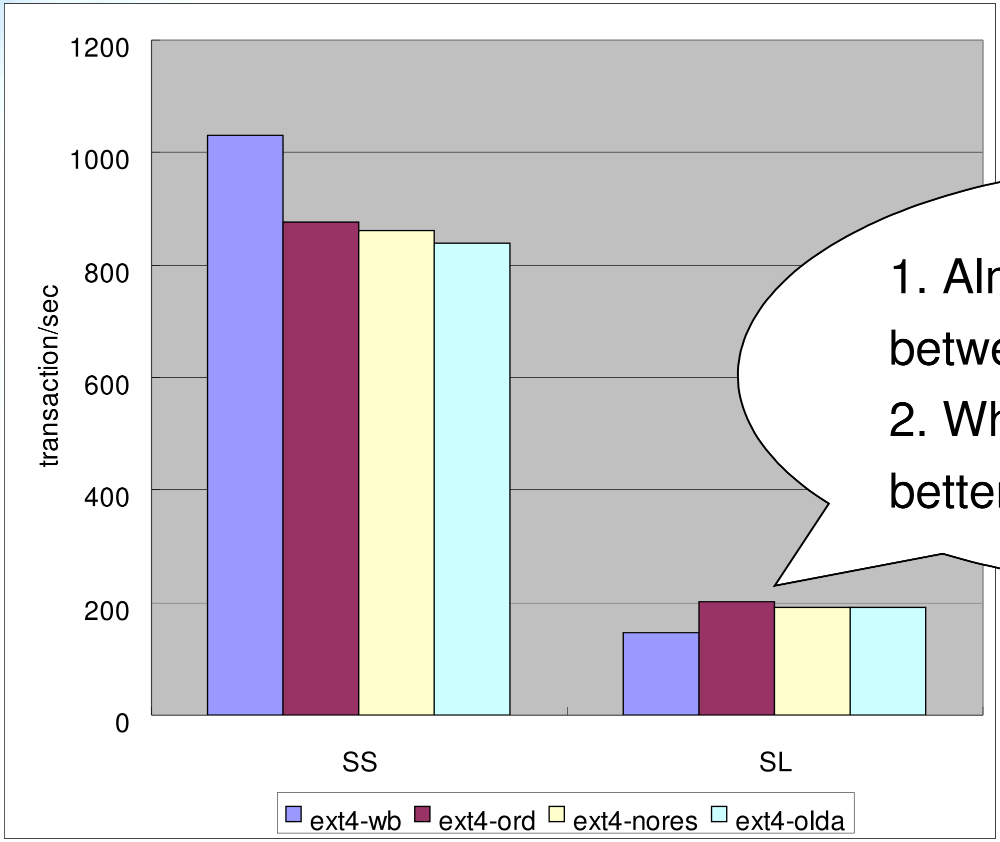
I/O statistics (2/2)

- Segmented sequentiality of write I/O (segment: 1MB)



Case study - ext4

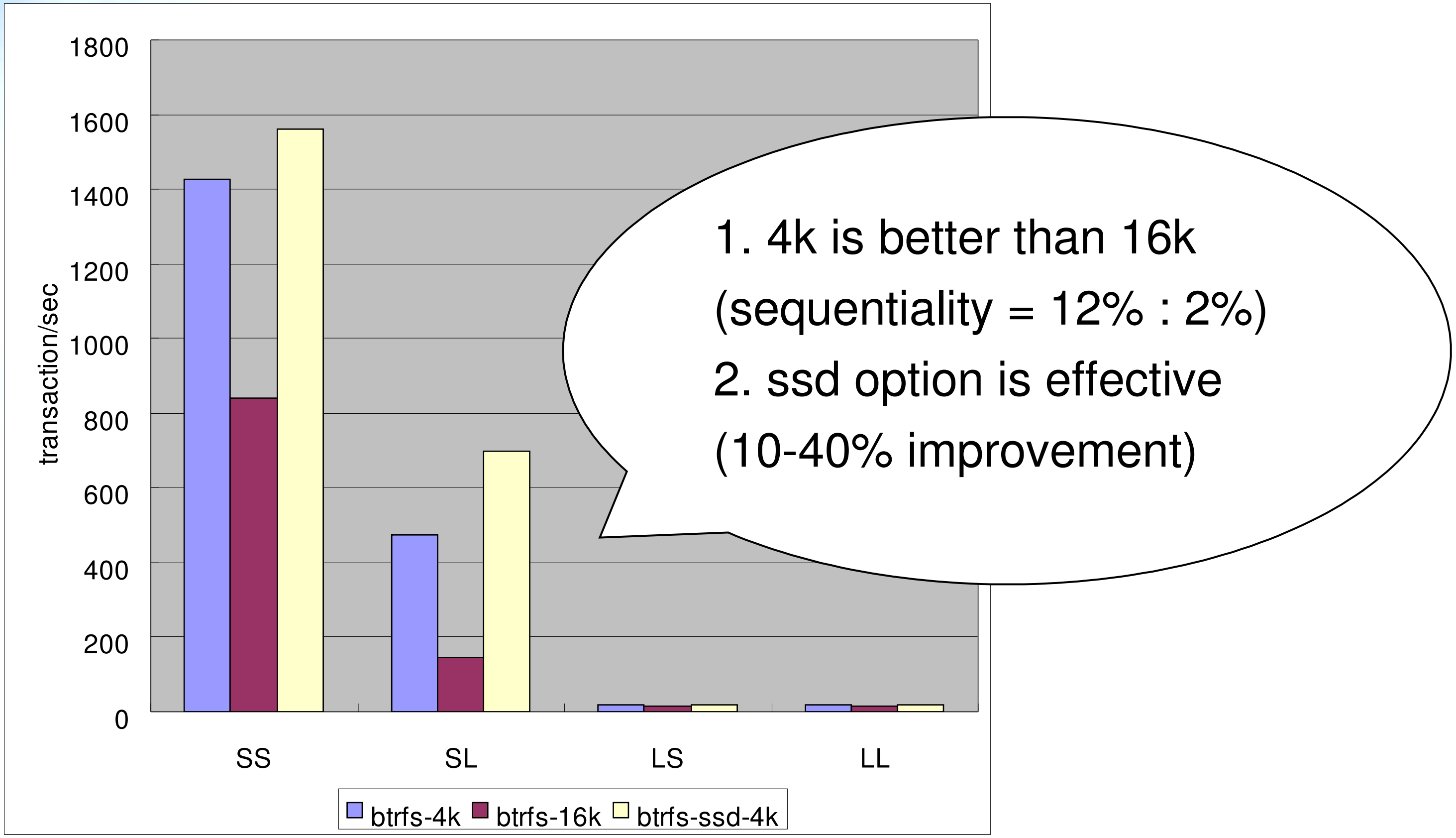
- Condition
  - data=ordered, allocation: default/noreservation/oldalloc



1. Almost no difference between allocation policies
2. Why data=ordered is better for SL?

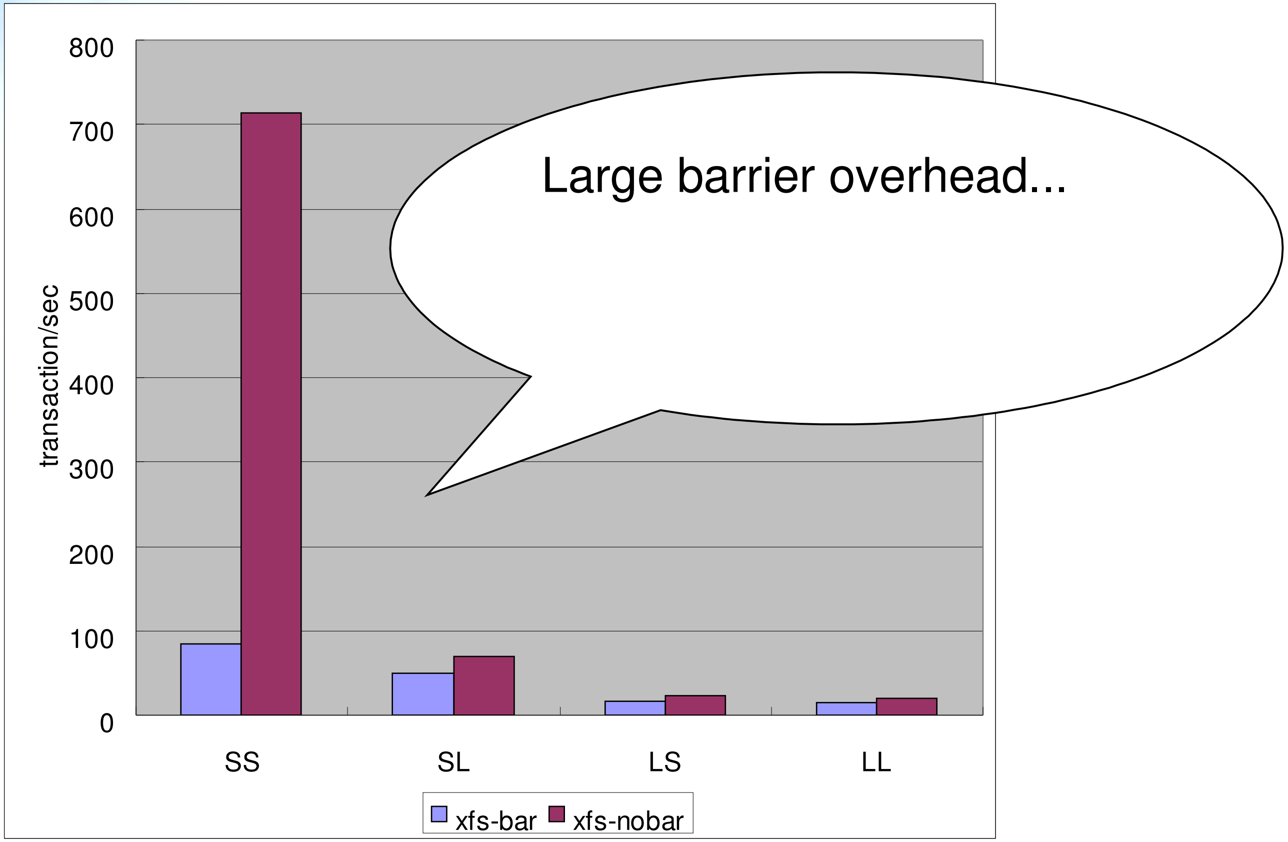
Case study - btrfs

- Condition
  - Block size: 4k/16k, allocation: ssd option on/off



Case study - xfs

- Condition
  - Mount with barrier on/off



Design consideration for SSD

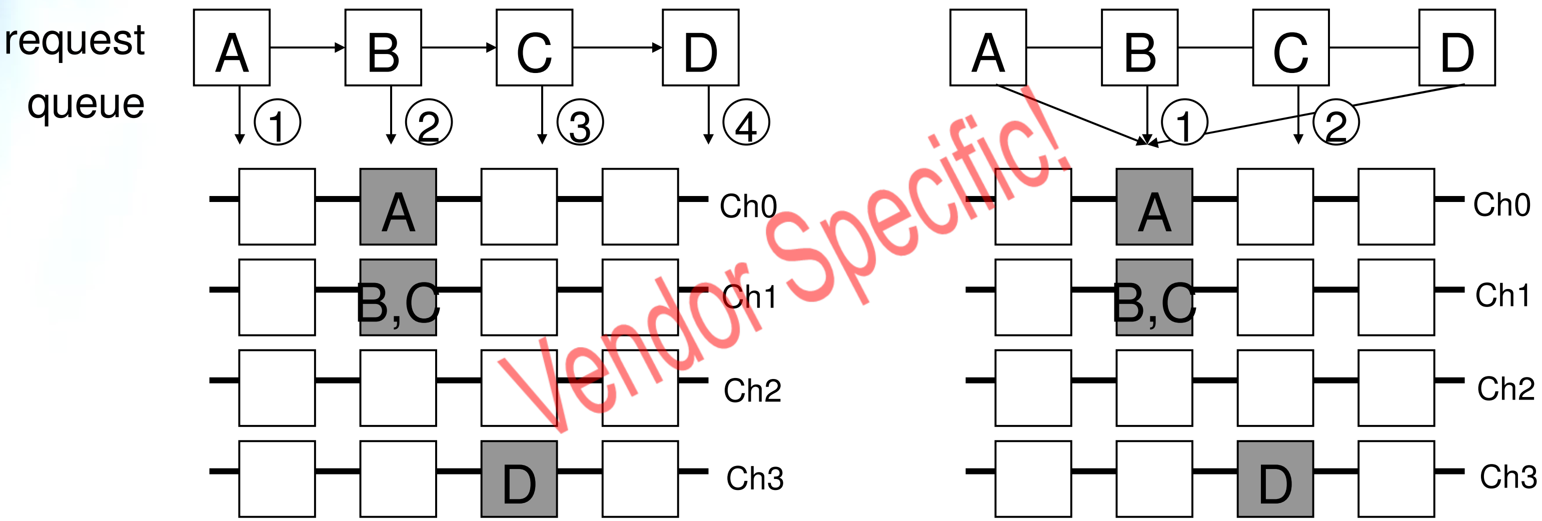
- Lessons from flash FS (ex. logfs)
  - Sequential writing at multiple logging points
  - Wandering tree
    - Trace-off between sequentiality vs. amount of write
    - Cf. space map (Sun ZFS)
  - Need to optimize garbage collection overhead
    - Either FS itself or FTL in SSD
- Next topic: End-to-end optimization
  - Exchange info with SSD (trim, SSD identification)
  - Make best use of parallelism

New interfaces for SSD (t13.org)

- Trim command
  - Let device know which LBA range is not used
    - This will be helpful for optimizing FTL
  - Should be passed through: FS → bio → scsi → libata
    - Passing bio with no data
    - What about I/O reordering & I/O queuing?
- SSD identification (added to “ATA identify”)
  - Report size of page and erasable block
    - Physical or effective?
  - Useful for FS and volume manager

Parallel processing of small I/O

- Make better use of I/O queuing (TCQ or NCQ)
  - Parallel processing of small I/O
  - Desktop environment? Barrier?



without I/O queuing, 4 steps

with I/O queuing, 2 steps

chip is busy

chip is idle

Summary

- Optimization for SSD
  - Alignment is important
  - Segmented sequentiality
  - Make better use of parallelism (either small or large)
    - I/O barrier may stall the pipelined processing

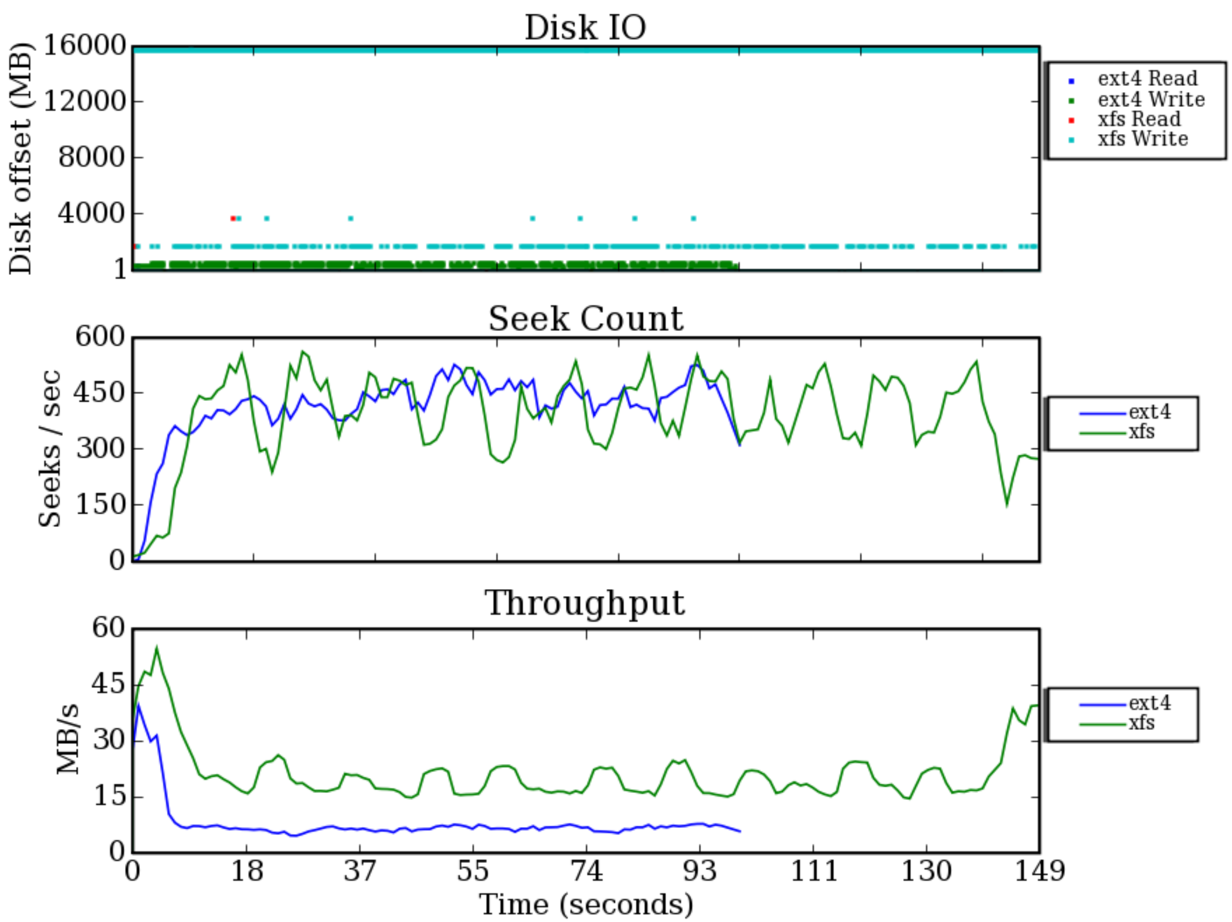
- What can you do?
  - File system: alignment, allocation policy, design (ex. COW)
  - Block layer: bio w/ hint, barrier, I/O queueing, scheduler(?)
  - Volume manager: alignment, allocation
  - Virtual memory: read-ahead

References

- T13 spec for SSD
  - http://www.t13.org/documents/UploadedDocuments/docs2007/e07153r(
  - http://www.t13.org/documents/UploadedDocuments/docs2007/e07154r(
- Introduction to SSD and flash memory
  - http://download.microsoft.com/download/a/f/d/afdfd50d-6eb9-425e-84e1
  - http://download.microsoft.com/download/d/f/6/df6accd5-4bf2-4984-8285
  - http://download.microsoft.com/download/a/f/d/afdfd50d-6eb9-425e-84e1
- FTL description & optimization
  - BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage (FAST '08)

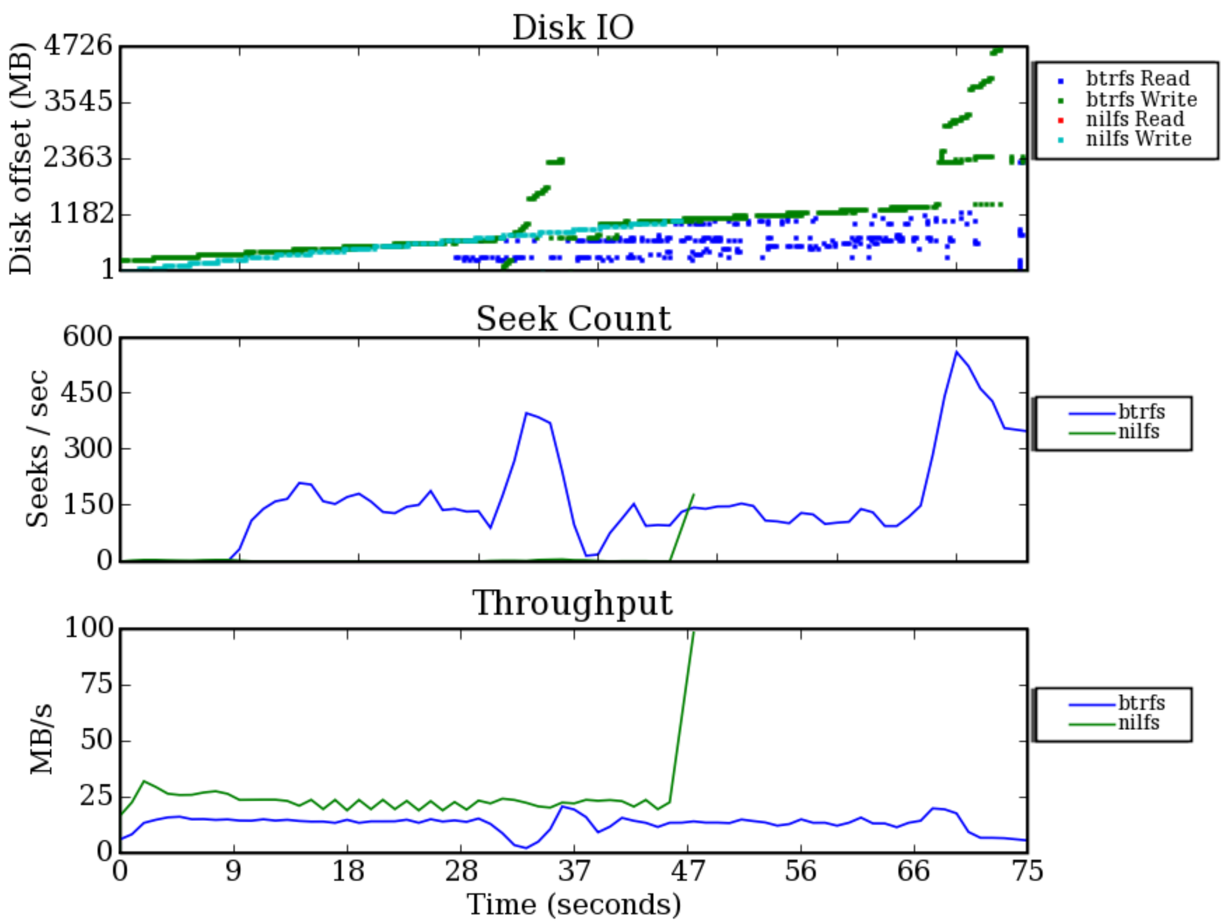
Appendix. I/O Pattern

- SS workload – ext4, xfs



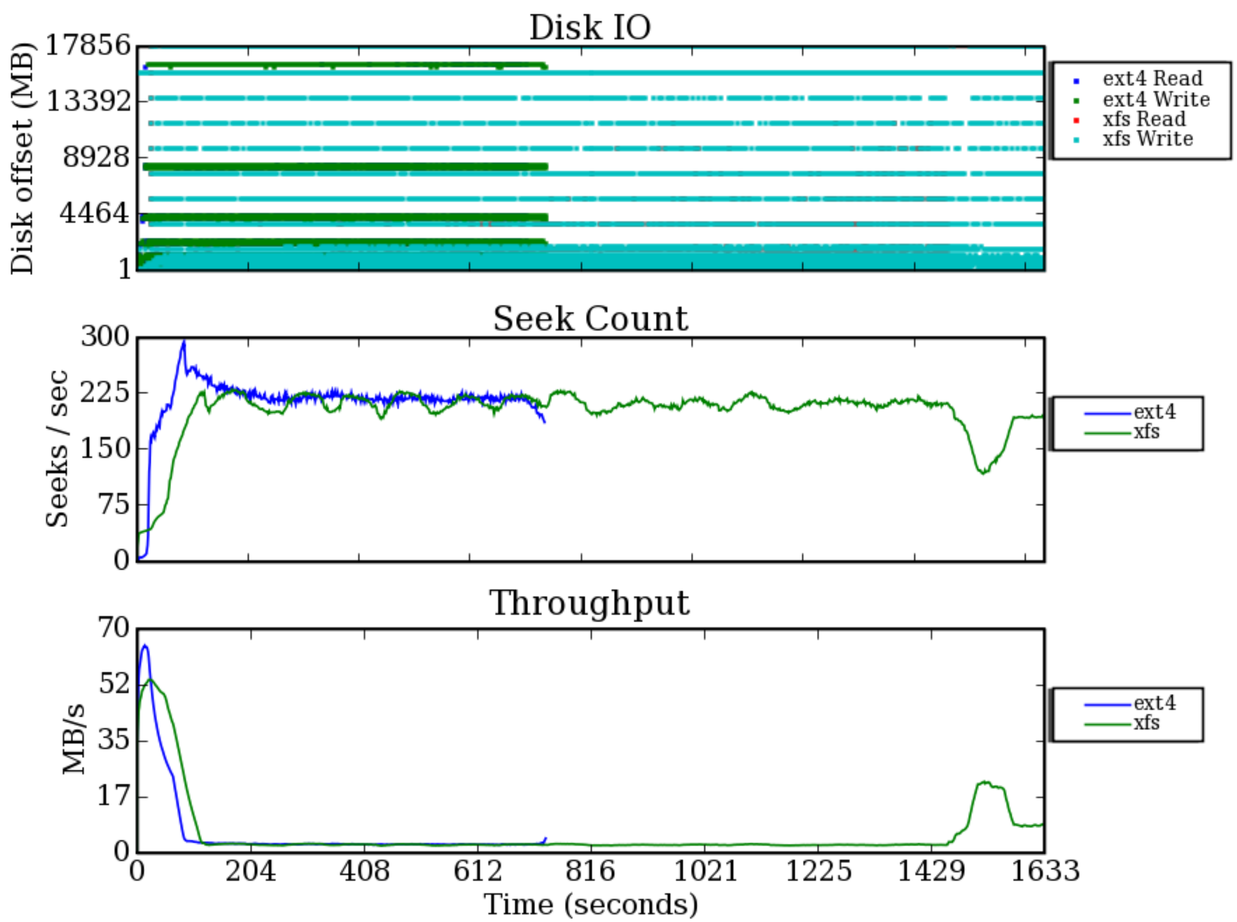
Appendix. I/O Pattern

- SS workload – btrfs, nilfs



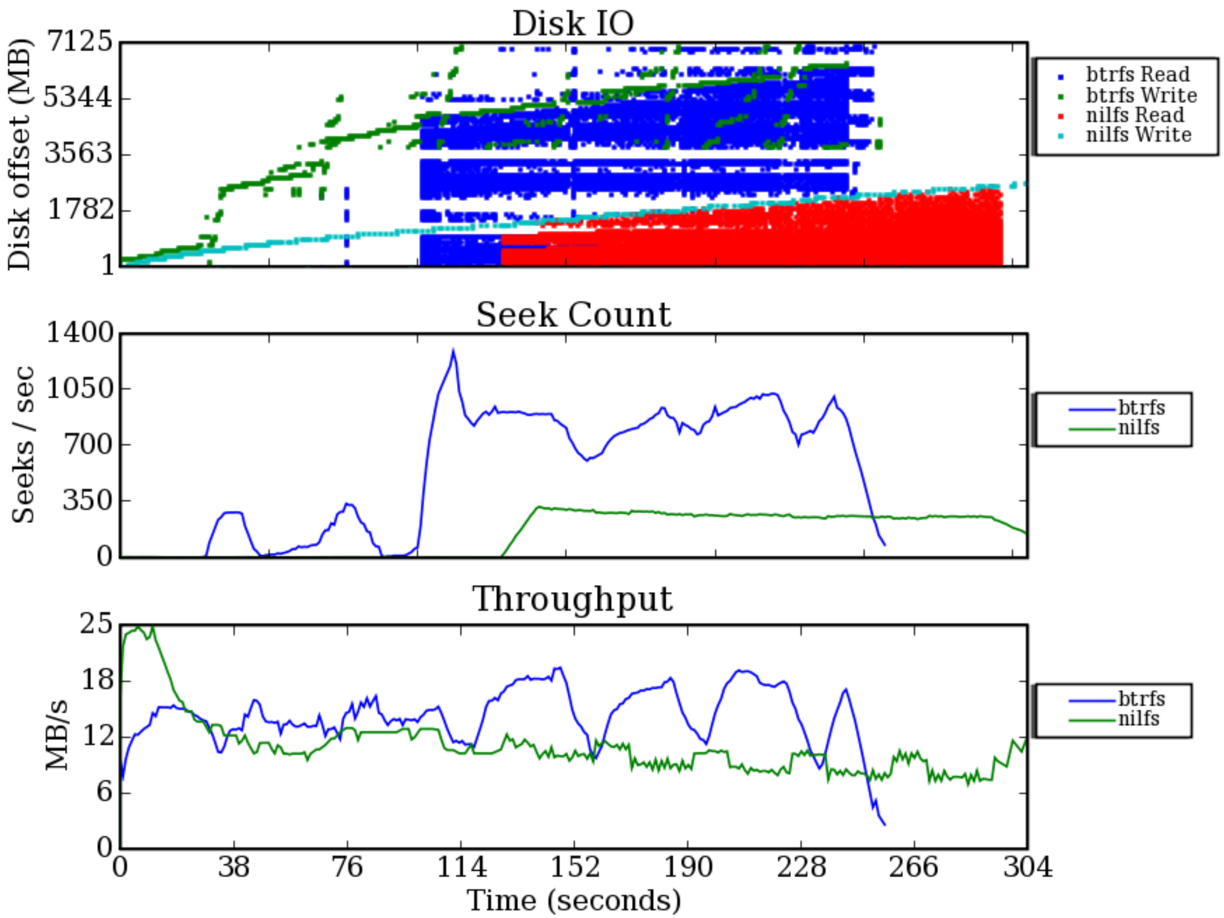
Appendix. I/O Pattern

- SL workload – ext4, xfs



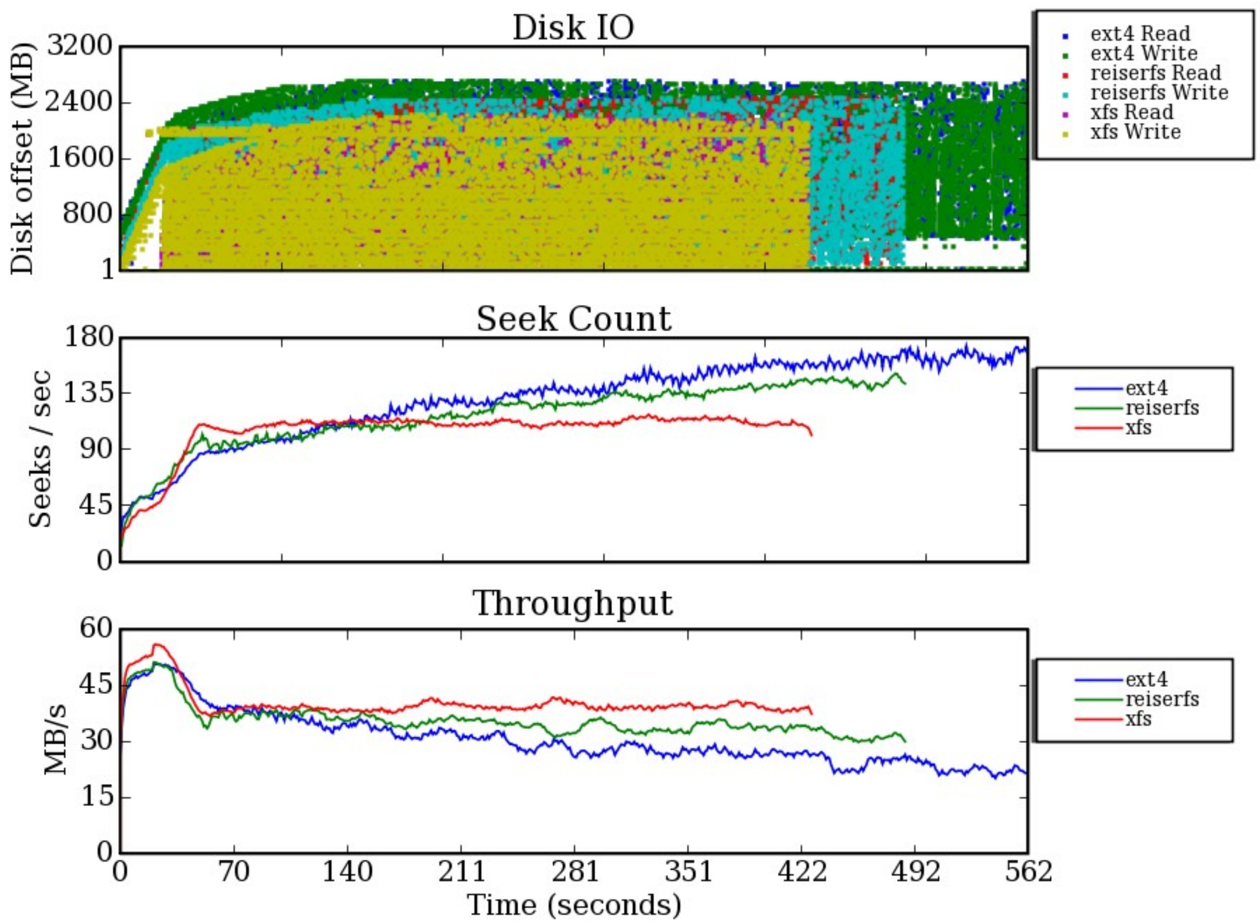
Appendix. I/O Pattern

- SL workload – btrfs, nilfs



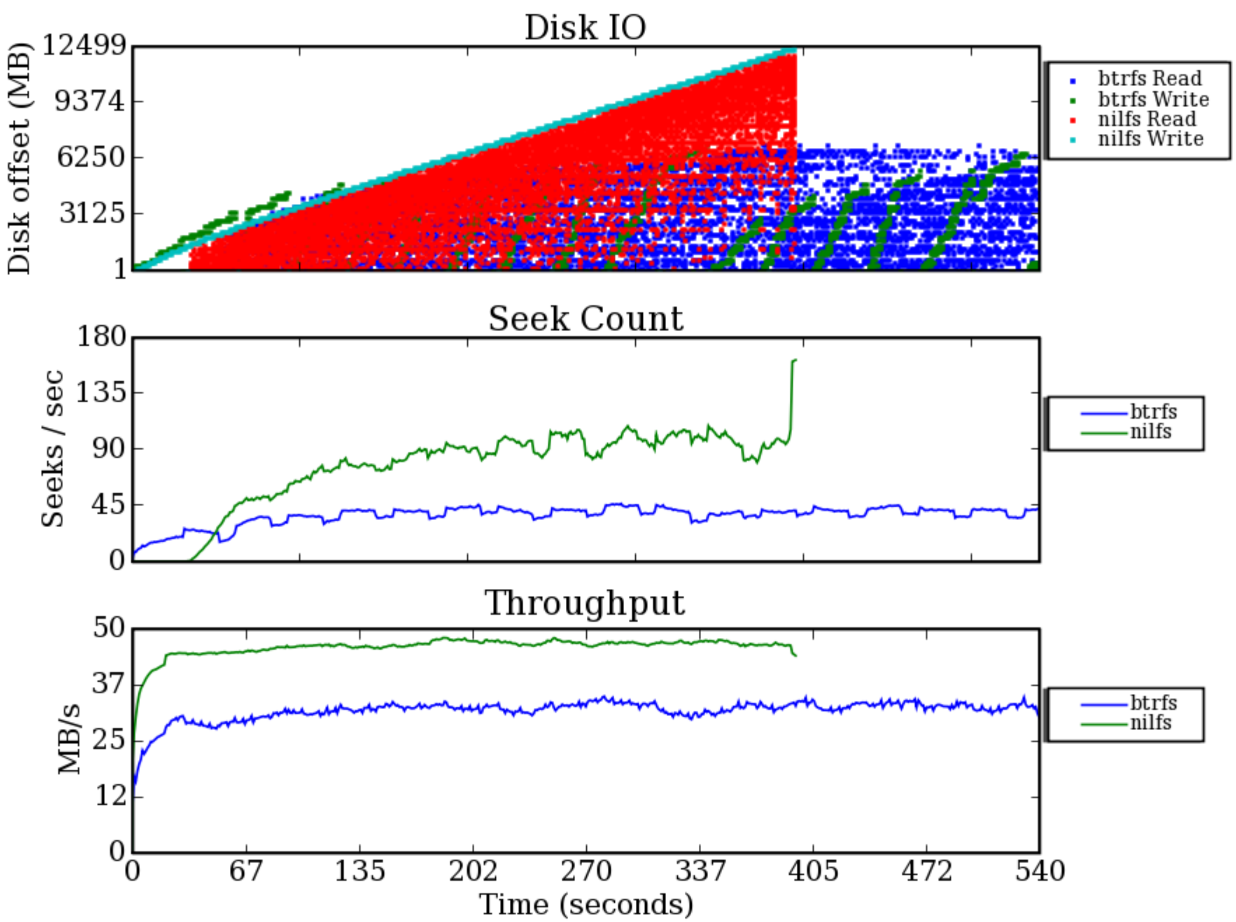
Appendix. I/O Pattern

- LS workload – ext4, reiserfs, xfs



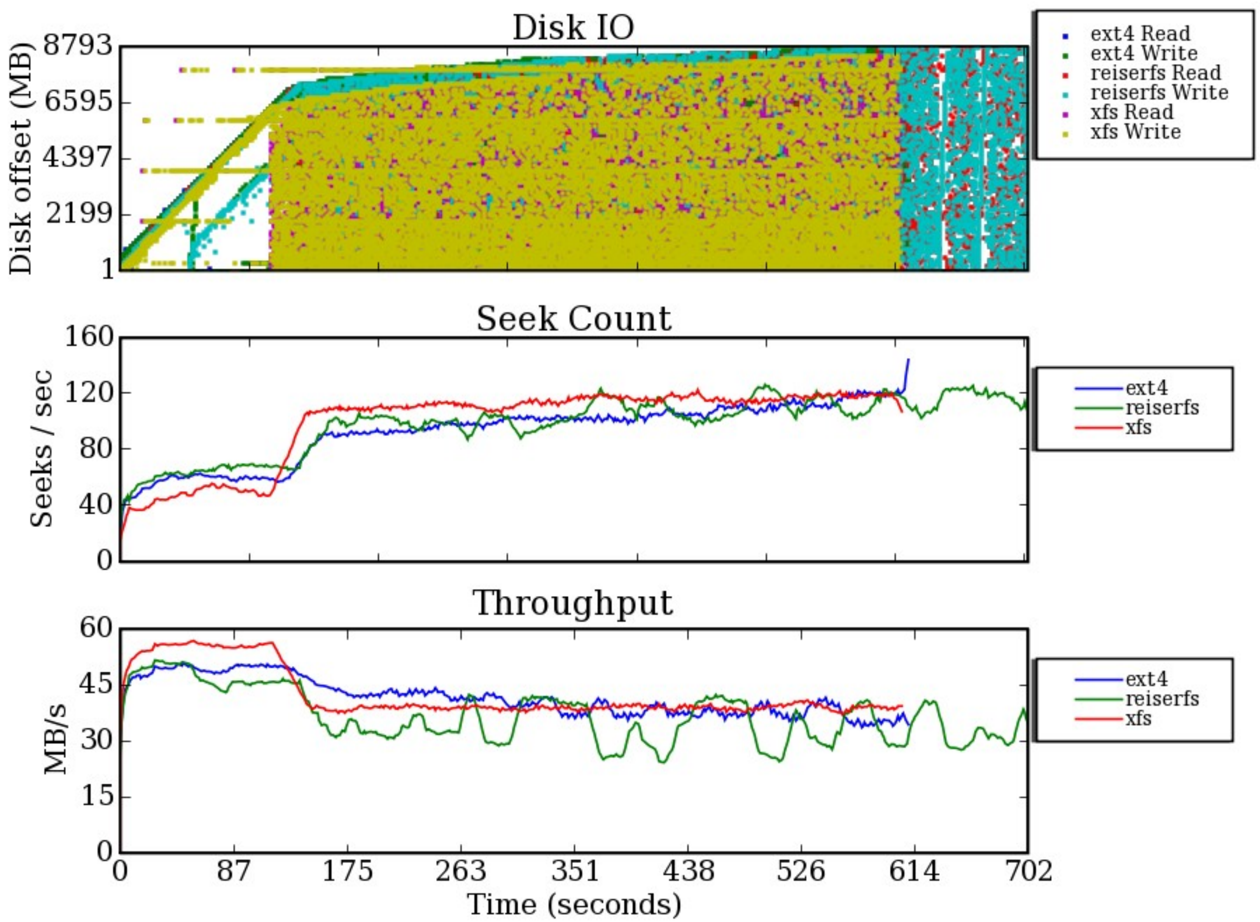
Appendix. I/O Pattern

- LS workload – btrfs, nilfs



Appendix. I/O Pattern

- LL workload – ext4, reiserfs, xfs



Appendix. I/O Pattern

- LL workload – btrfs, nilfs