SFS: Random Write Considered Harmful in Solid State Drives

Changwoo Min[1, 2], Kangnyeon Kim[1], Hyunjin Cho[2],

Sang-Won Lee[1], Young Ik Eom[1]

[1]Sungkyunkwan University, Korea
[2]Samsung Electronics, Korea

Outline

- Background
- Design Decisions
- Introduction
- Segment Writing
- Segment Cleaning
- Evaluation
- Conclusion

Flash-based Solid State Drives

- Solid State Drive (SSD)
  - A purely electronic device built on NAND flash memory
  - No mechanical parts
- Technical merits
  - Low access latency
  - Low power consumption
  - Shock resistance
  - Potentially uniform random access speed
- Remaining two problems limiting wider deployment of SSDs
  - Limited life span
  - Random write performance

Limited lifespan of SSDs

- Limited program/erase (P/E) cycles of NAND flash memory
  - Single-level Cell (SLC): 100K ~ 1M
  - Multi-level Cell (MLC): 5K ~ 10K
  - Triple-level Cell (TLC): 1K
- As bit density increases
  → cost decreases, lifespan decreases
- Starting to be used in laptops, desktops and data centers.
  - Contain write intensive workloads

Random Write Considered Harmful in SSDs

- Random write is slow.
  - Even in modern SSDs, the disparity with sequential write bandwidth is more than ten-fold.

- Random writes shortens the lifespan of SSDs.
  - Random write causes internal fragmentation of SSDs.
  - Internal fragmentation increases garbage collection cost inside SSDs.
  - Increased garbage collection overhead incurs more block erases per write and degrades performance.
  - Therefore, the lifespan of SSDs can be drastically reduced by random writes.

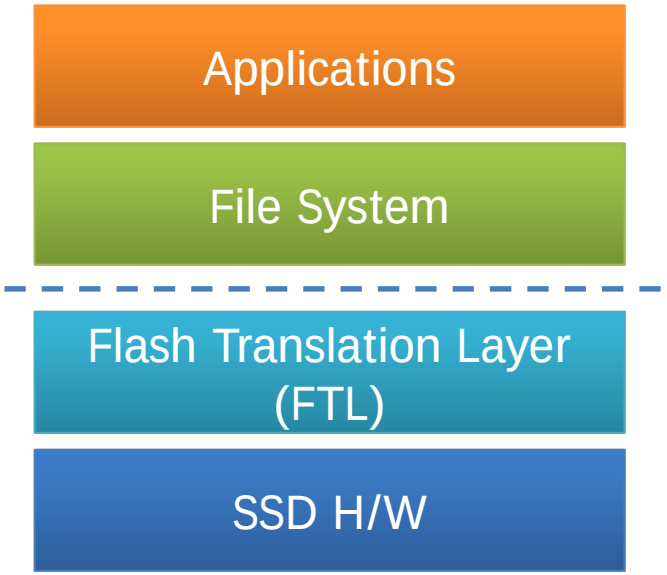
Optimization Factors

- SSD H/W
  - Larger over-provisioned space → lower garbage collection cost inside SSDs

  ➔ Higher cost
- Flash Translation Layer (FTL)
  - More efficient address mapping schemes
  - Purely based on LBA requested from file system
    - Less effective for the no-overwrite file systems

  ➔ Lack of information
- Applications
  - SSD-aware storage schemes (e.g. DBMS)
  - Quite effective for specific applications

  ➔ Lack of generality

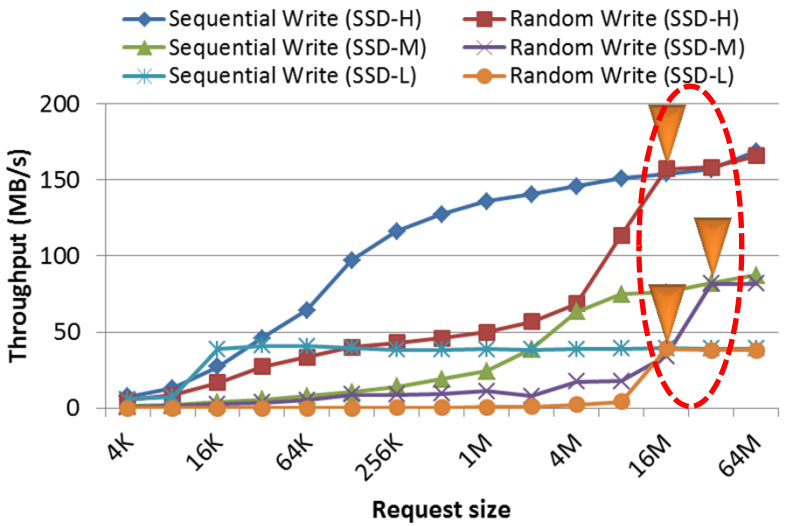


We took **a file system level approach** to directly exploit file block level statistics and provide our optimizations to general applications.

Outline

- Background
- Design Decisions
  - Log-structured File System
  - Eager *on writing* data grouping
- Introduction
- Segment Writing
- Segment Cleaning
- Evaluation
- Conclusion

Performance Characteristics of SSDs



- If the request size of the random write are same as erase block size, such write requests invalidate whole erase block inside SSDs.
- Since all pages in an erase block are invalidated together, there is no internal fragmentation.

The random write performance becomes same as sequential write performance when the request size is same as erase block size.

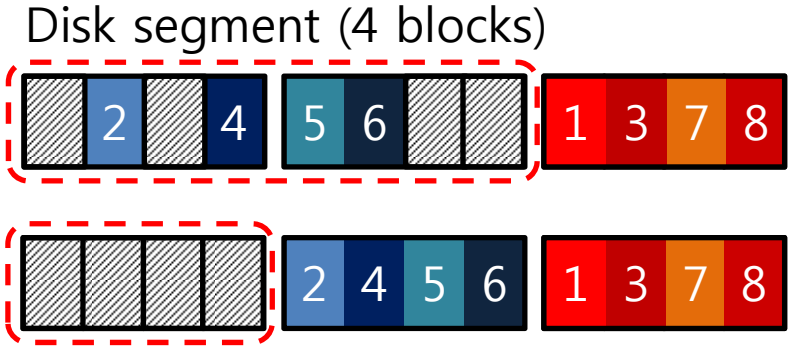
Log-structured File System

- *How can we utilize the performance characteristics of SSD in designing a file system?*

- Log-structured File System
  - It transforms the random writes at file system level into the sequential writes at SSD level.
  - If segment size is equal to the erase block size of a SSD, the file system will always send erase block sized write requests to the SSD.
  - So, write performance is mainly determined by sequential write performance of a SSD.

Eager *on writing* data grouping

- To secure large empty chunk for bulk sequential write, segment cleaning is needed.
  - Major source of overhead in any log-structured file system
  - When hot data is colocated with cold data in the same segment, cleaning overhead significantly increases.



Four live blocks should be moved to secure an empty segment.

No need to move blocks to secure an empty segment.

- Traditional LFS writes data regardless of hot/cold and then tries to separate data *lazily on segment cleaning*.
  - If we can categorize hot/cold data when it is first written, there is much room for improvement.

➔ *Eager on writing data grouping*

Outline

- Background
- Design Decisions
- Introduction
- Segment Writing
- Segment Cleaning
- Evaluation
- Conclusion

SFS in a nutshell

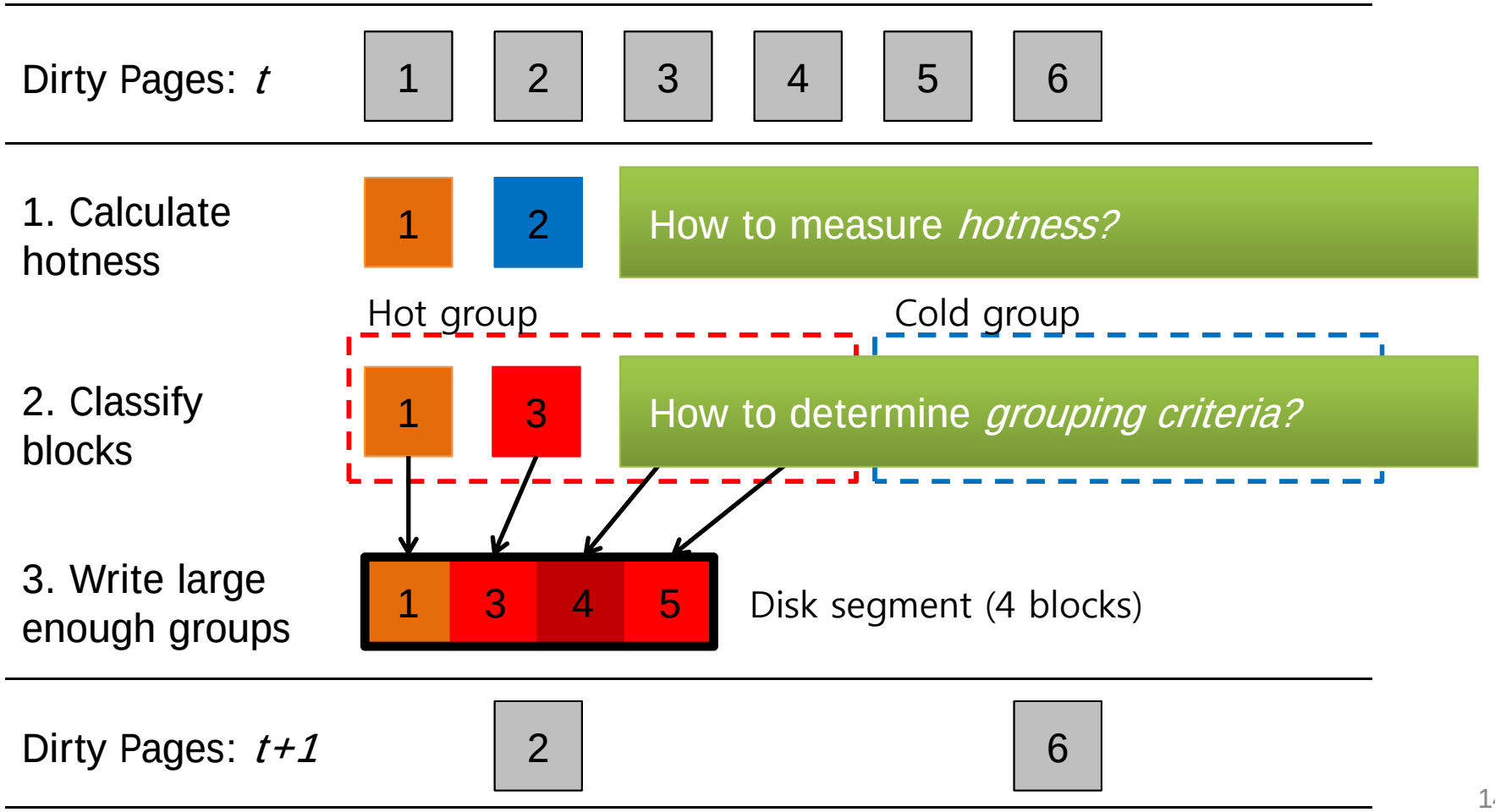
- A log-structured file system
- Segment size is multiple of erase block size
  - Random write bandwidth = Sequential write bandwidth
- Eager on writing data grouping
  - Colocate blocks with similar update likelihood, *hotness*, into the same segment when they are first written
  - To form bimodal distribution of segment utilization
  - Significantly reduces segment cleaning overhead
- Cost-hotness segment cleaning
  - Natural extension of cost-benefit policy
  - Better victim segment selection

Outline

- Background
- Design Decisions
- Introduction
- Segment Writing
- Segment Cleaning
- Evaluation
- Conclusion

On Writing Data Grouping

- Colocate blocks with similar update likelihood, *hotness*, into the same segment when they are first written.



Measuring Hotness

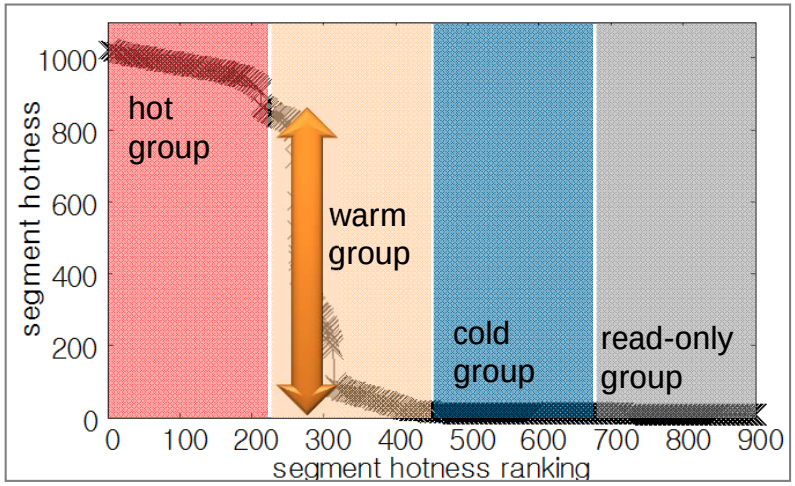
- Hotness: update likelihood
  - Frequently updated data ➔ hotness ↑
  - Recently updated data ➔ hotness ↑
  - $hotness = \frac{write\ count}{age}$

File block hotness H_b	Segment hotness H_s
$file\ block\ hotness$ $= \frac{write\ count\ of\ a\ file\ block}{age\ of\ a\ file\ block}$	$segment\ hotness$ $= average\ hotness\ of\ live\ blocks\ in\ a\ segment$ $\approx \frac{mean\ of\ write\ count\ of\ live\ blocks}{mean\ of\ age\ of\ live\ blocks}$

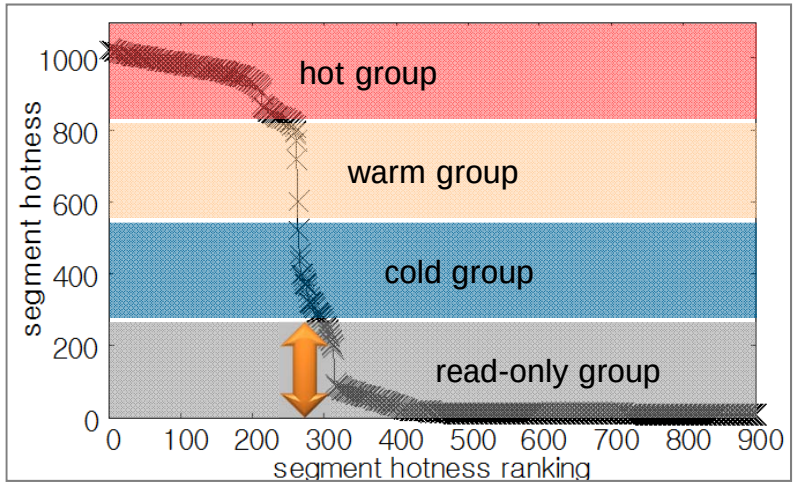
Determining Grouping Criteria : Segment Quantization

- The effectiveness of block grouping is determined by the grouping criteria.
  - Improper criteria may colocate blocks from different groups into the same segment, thus deteriorates the effectiveness of grouping.

- Naïve solution does not work.



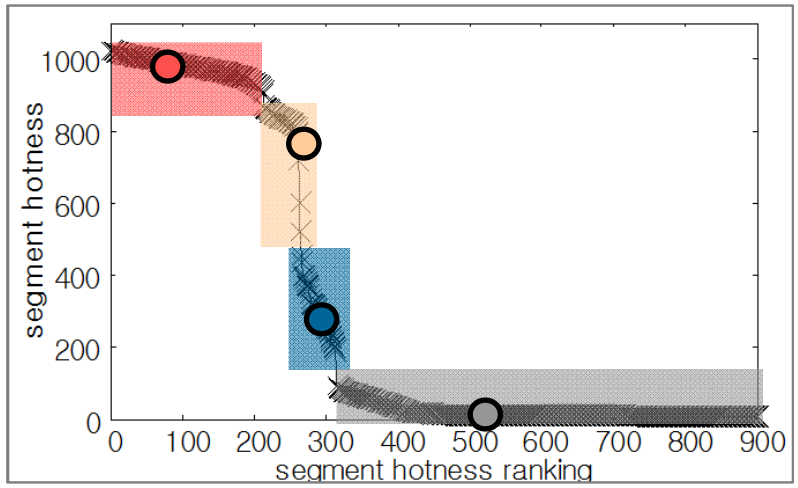
equi-width partitioning



equi-height partitioning

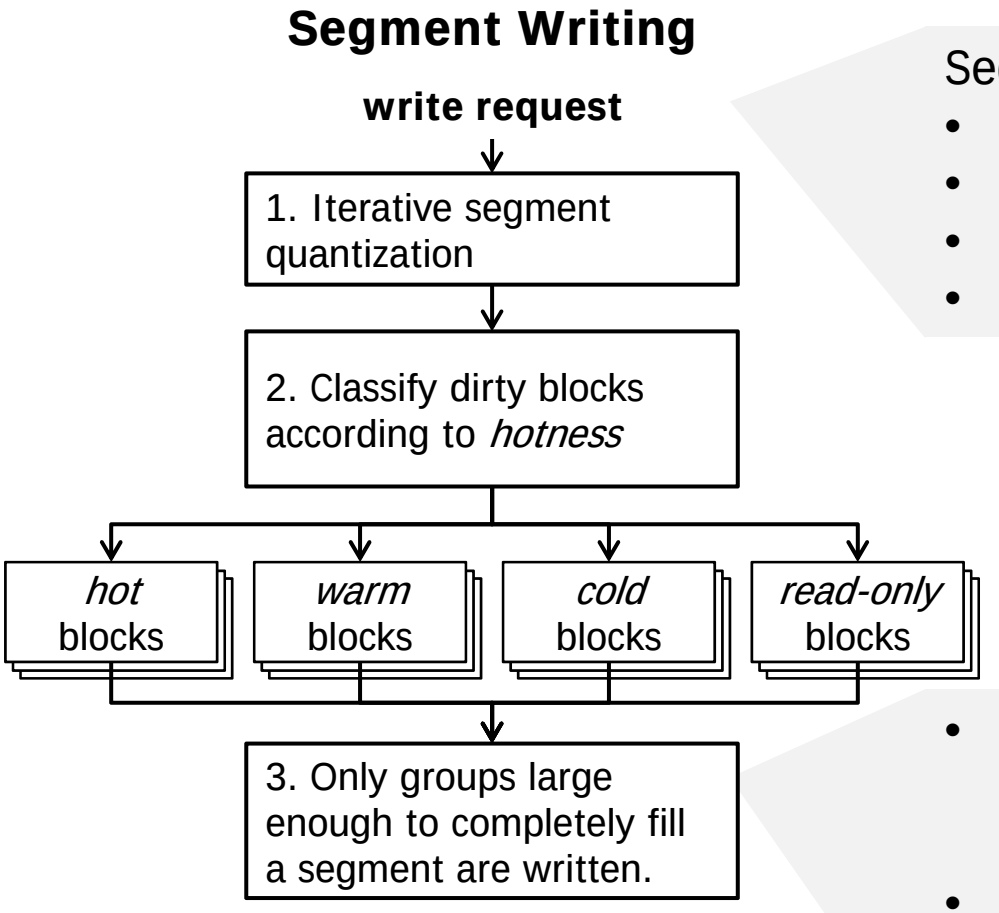
Iterative Segment Quantization

- Find *natural hotness groups* across segments in disk.
  - Mean of segment hotness in each group is used as grouping criterion.
  - Iterative refinement scheme inspired by k-means clustering algorithm
- Runtime overhead is reasonable.
  - 32MB segment → only 32 segments for 1GB disk space
  - For faster convergence, the calculated centers are stored in meta data and loaded at mounting a file system.



1. Randomly select initial center of groups
2. Assign each segment to the closest center.
3. Calculate a new center by averaging hotnesses in a group.
4. Repeat Step 2 and 3 until convergence has been reached or three times at most.

Process of Segment Writing



Segment writing is invoked in four case:

- every five seconds
- flush daemon to reduce dirty pages
- segment cleaning
- *sync* or *fsync*

- Writing of the small groups will be deferred until the size of the group grows to completely fill a segment.
- Eventually, the remaining small groups will be written at creating a check-point.

Outline

- Background
- Design Decisions
- Introduction
- Segment Writing
- Segment Cleaning
- Evaluation
- Conclusion

Cost-hotness Policy

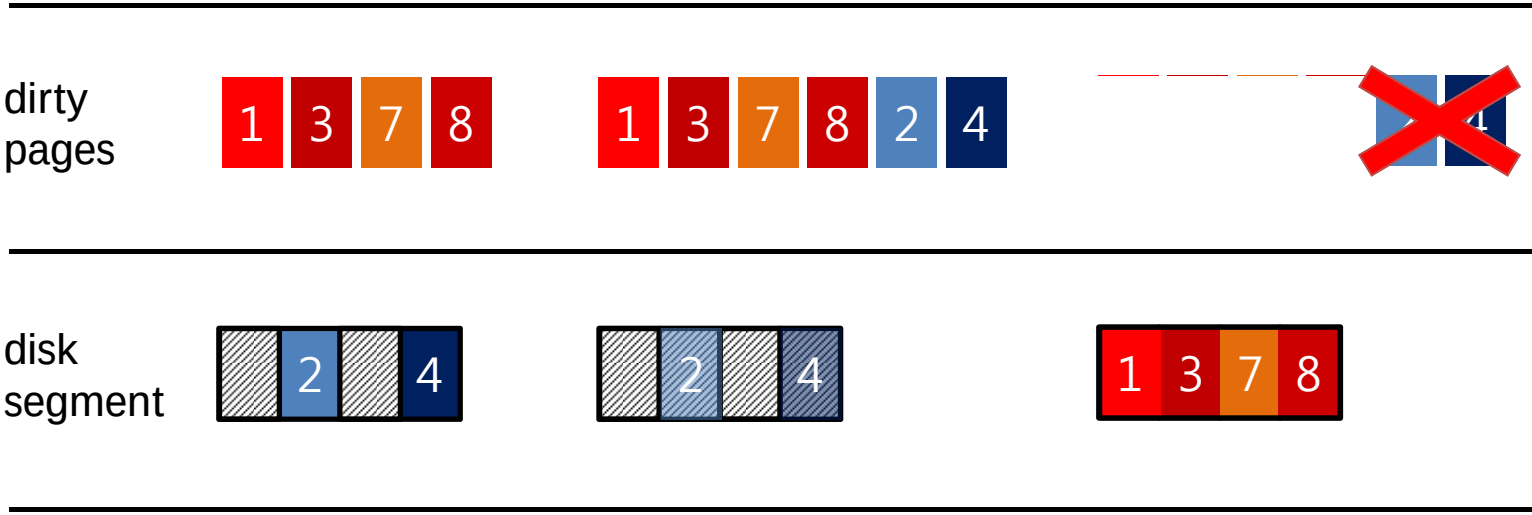
- Natural extension of cost-benefit policy
- In cost-benefit policy, the age of the youngest block in a segment is used to estimate update likelihood of the segment.
  - cost-benefit $= \frac{free\ space\ generated * \boldsymbol{age\ of\ data}}{cost}$
- In cost-hotness policy, we use segment hotness instead of the age, since segment hotness directly represents the update likelihood of segment.
  - cost-hotness $= \frac{free\ space\ generated}{cost * \boldsymbol{segment\ hotness}}$
  - Segment cleaner selects a victim segment with maximum cost-hotness value.

Writing Blocks under Segment Cleaning

- Live blocks under segment cleaning are handled similarly to typical writing scenario.
  - Their writing can also be deferred for continuous re-grouping
  - Continuous re-grouping to form bimodal segment distribution.

Scenario of Data Loss in System Crash

- There are possibility of data loss for the live blocks under segment cleaning in system crash or sudden power off.



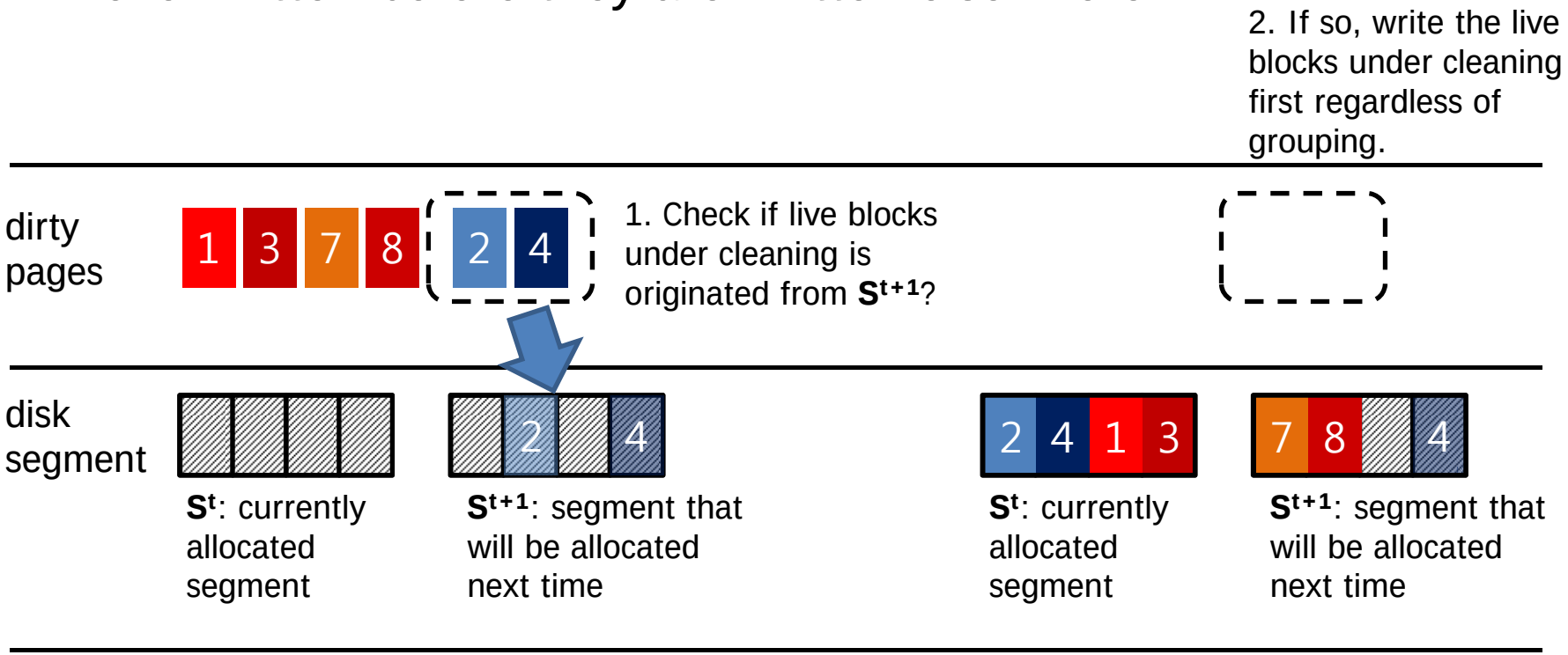
1. Segment cleaning. Live blocks are read into the page cache.

2. Hot blocks are written.

3. System Crash!! ➔ Block 2, 4 will be lost since they do not have on-disk copy.

How to Prevent Data Loss

- Segment Allocation Scheme
  - Allocate a segment in *Least Recently Freed* (LRF) order.
- Check if writing a normal block could cause data loss of blocks under cleaning.
- This guarantees that live blocks under cleaning are never overwritten before they are written elsewhere.



Outline

- Background
- Design Decisions
- Introduction
- Segment Writing
- Segment Cleaning
- Evaluation
- Conclusion

Evaluation

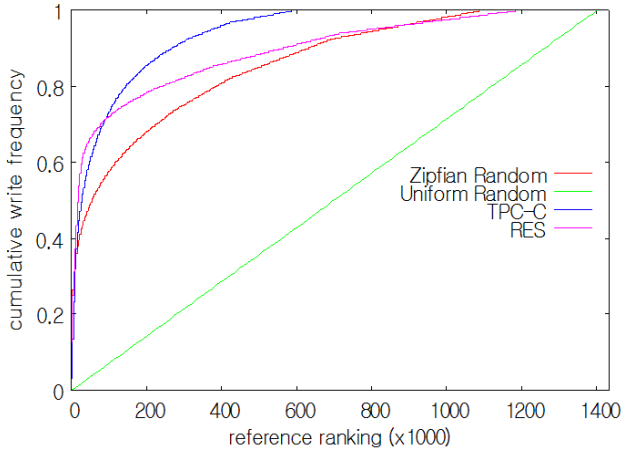
- Server
  - Intel i5 Quad Core, 4GB RAM
  - Linux Kernel 2.6.27
- SSD

	SSD-H	SSD-M	SSD-L
Interface	SATA	SATA	USB 3.0
Flash Memory	SLC	MLC	MLC
Max. Sequential Writes (MB/s)	170	87	38
Random 4KB Writes (MB/s)	5.3	0.6	0.002
Price ($/GB)	14	2.3	1.4

- Configuration
  - 4 data groups
  - Segment size: 32MB

Workload

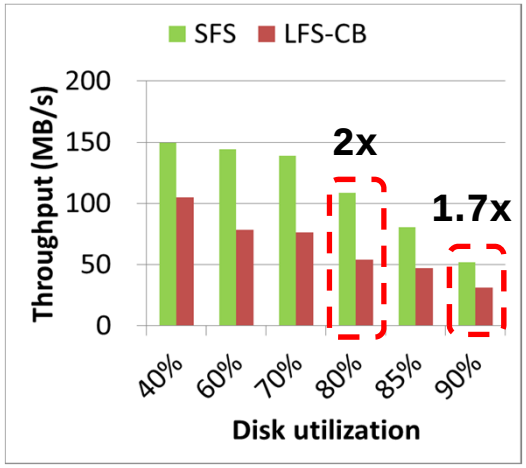
- Synthetic Workload
  - Zipfian Random Write
  - Uniform Random Write
    - No skewness → worst-case scenario of SFS



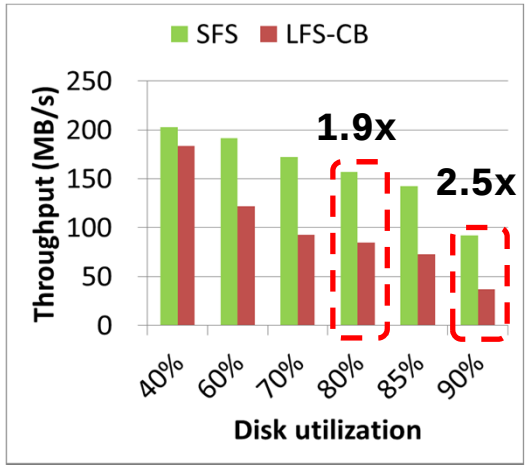
- Real-world Workload
  - TPC-C benchmark
  - Research Workload (RES) [Roseli2000]
    - Collected for 113 days on a system consisting of 13 desktop machines of research group.

- Replaying workload
  - To measure the maximum write performance, we replayed write requests in the workloads as fast as possible in a single thread and measured throughput at the application level.
  - Native Command Queuing (NCQ) is enabled.

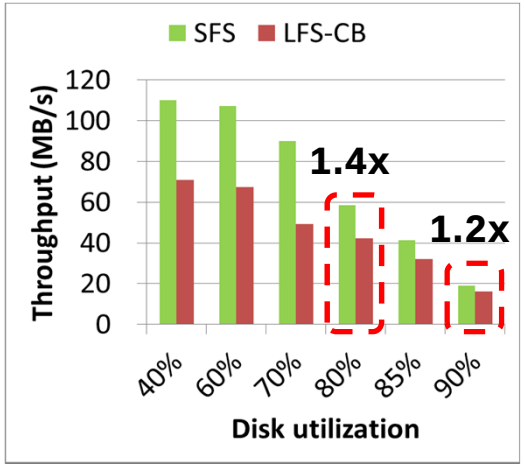
Throughput vs. Disk Utilization



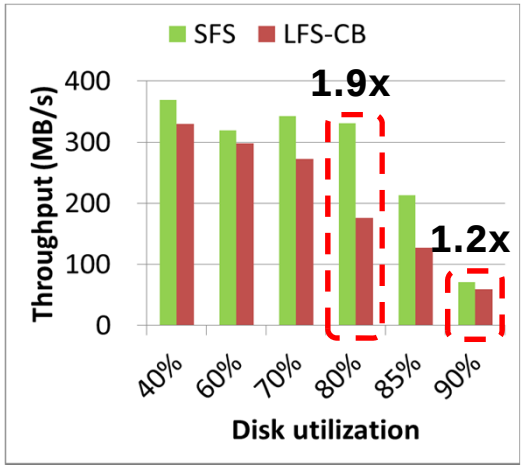
Zipfian Random Write



TPC-C



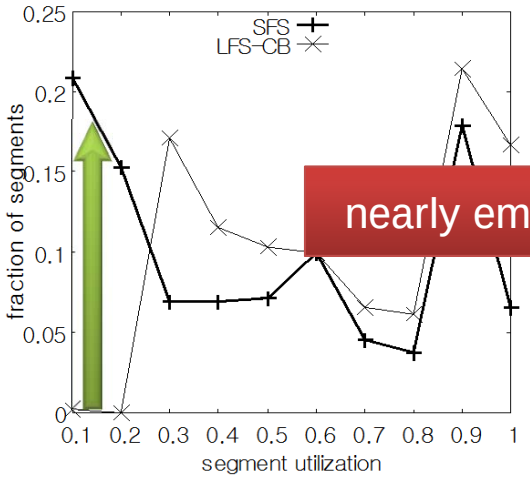
Uniform Random Write



RES

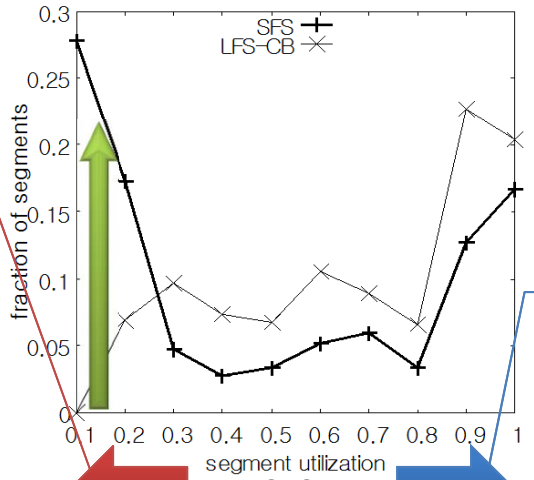
* SSD-M

Segment Utilization Distribution



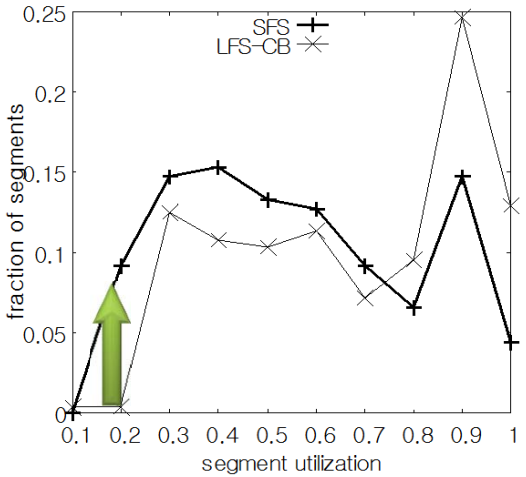
Zipfian Random Write

nearly empty

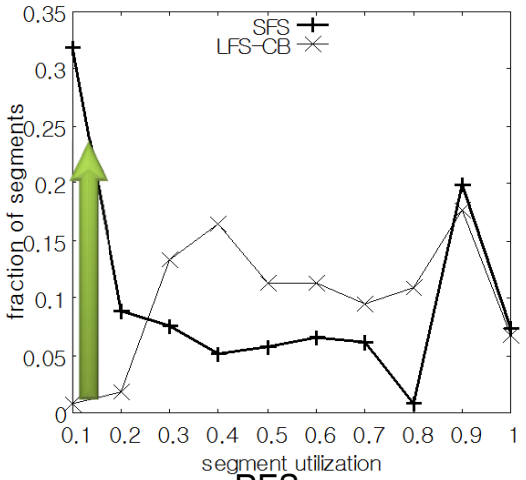


TPC-C

nearly full



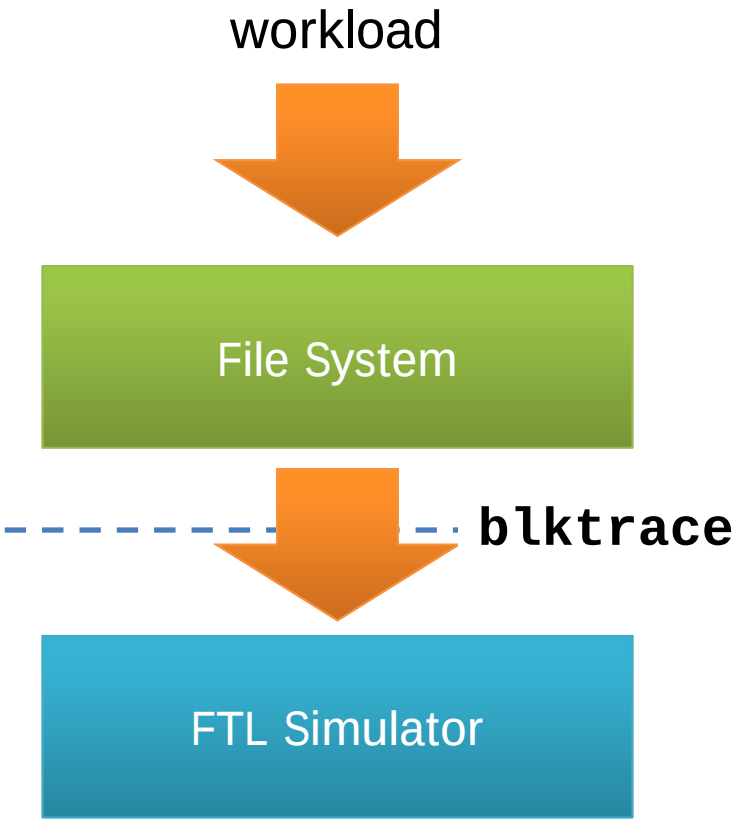
Uniform Random Write



RES

* Disk utilization is 70%.

Comparison with Other File Systems



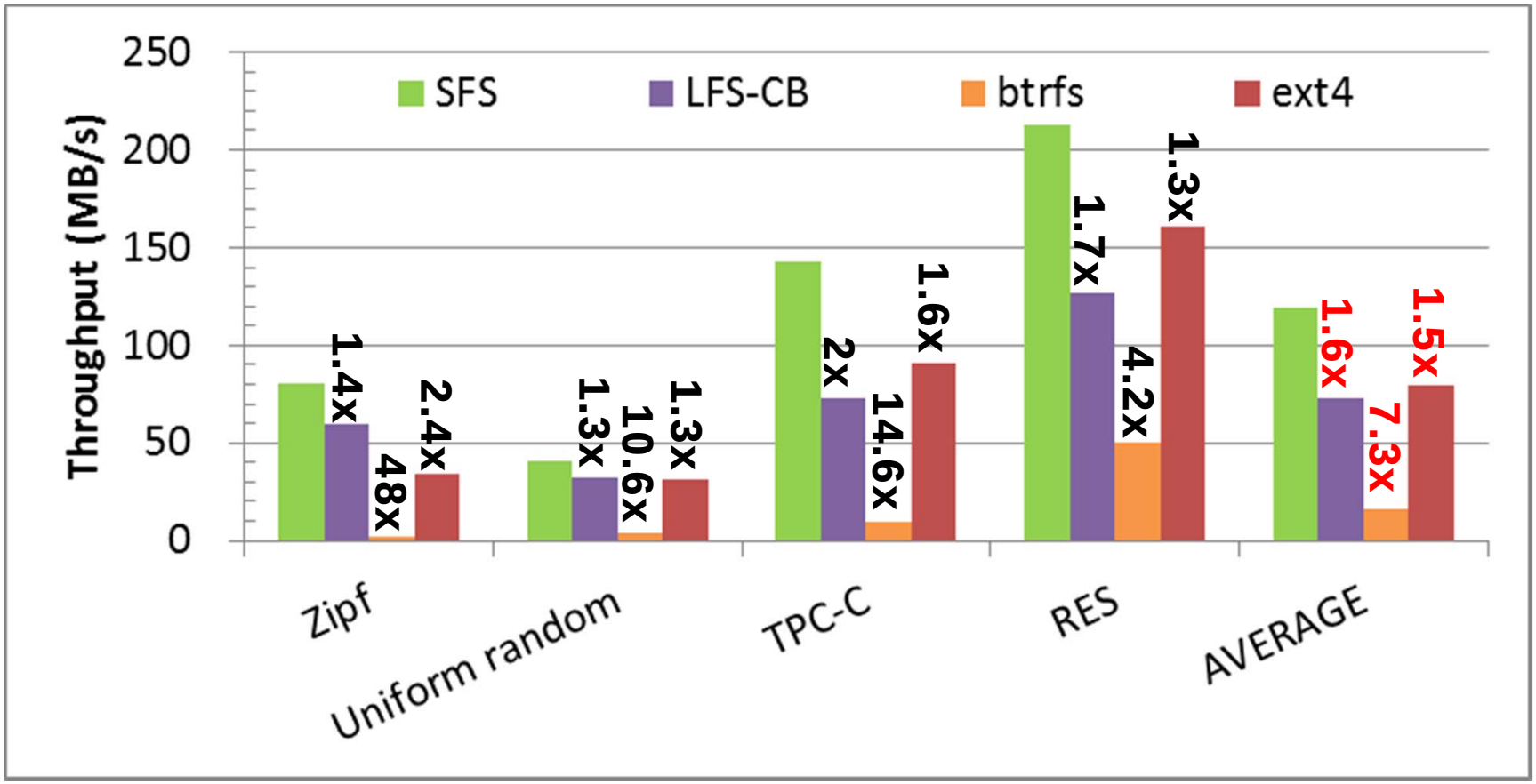
- Ext4
  - In-place-update file system
- Btrfs
  - No overwrite file system

➔ Measured Throughput

- Coarse grained hybrid mapping FTL
  - FAST FTL [Lee'07]
- Full page mapping FTL

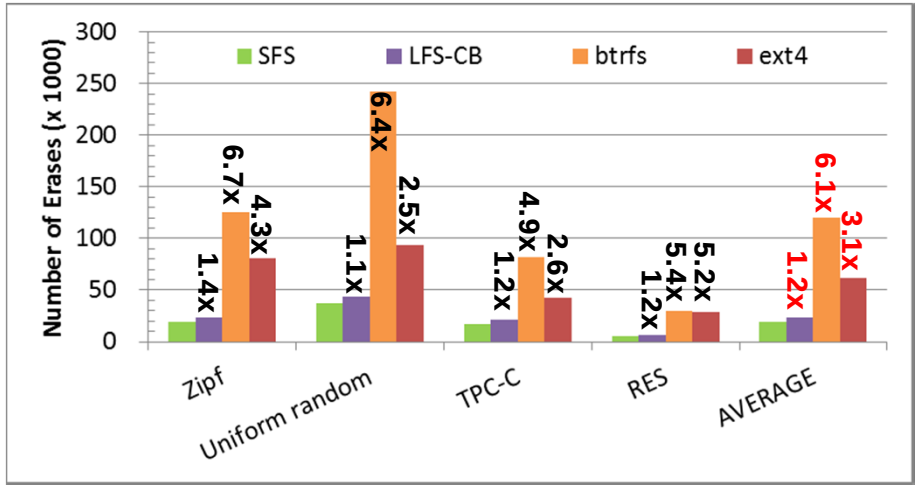
➔ Measured Write Amplification and Block Erase Count

Throughput under Different File Systems

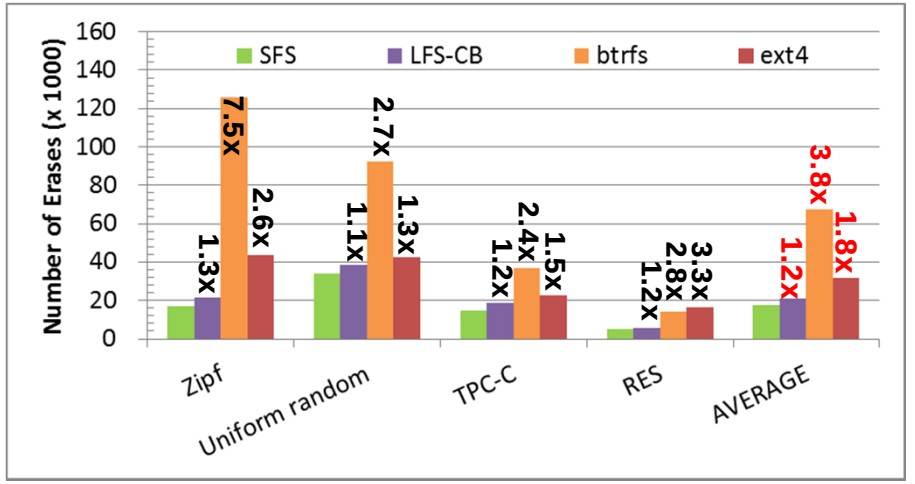


* Disk utilization is 85%. / SSD-M

Block Erase Count



Coarse Grained Hybrid Mapping FTL: FAST FTL



Full page mapping FTL

* Disk utilization is 85%.

Outline

- Background
- Design Decisions
- Introduction
- Segment Writing
- Segment Cleaning
- Evaluation
- Conclusion

Conclusion

- Random write on SSDs causes performance degradation and shortens the lifespan of SSDs.
- We present a new file system for SSD, SFS.
  - Log-structured file system
  - On writing data grouping
  - Cost-hotness policy
- We show that SFS considerably outperforms existing file systems and prolongs the lifespan of SSD by drastically reducing block erase count inside SSD.
- Is SFS also beneficial to HDDs?
  - Preliminary experiment results are available on our poster!

THANK YOU!

QUESTIONS?