ParaFS: A Log-Structured File System to Exploit the Internal Parallelism of Flash Devices

Jiacheng Zhang, Jiwu Shu, Youyou Lu

Tsinghua University

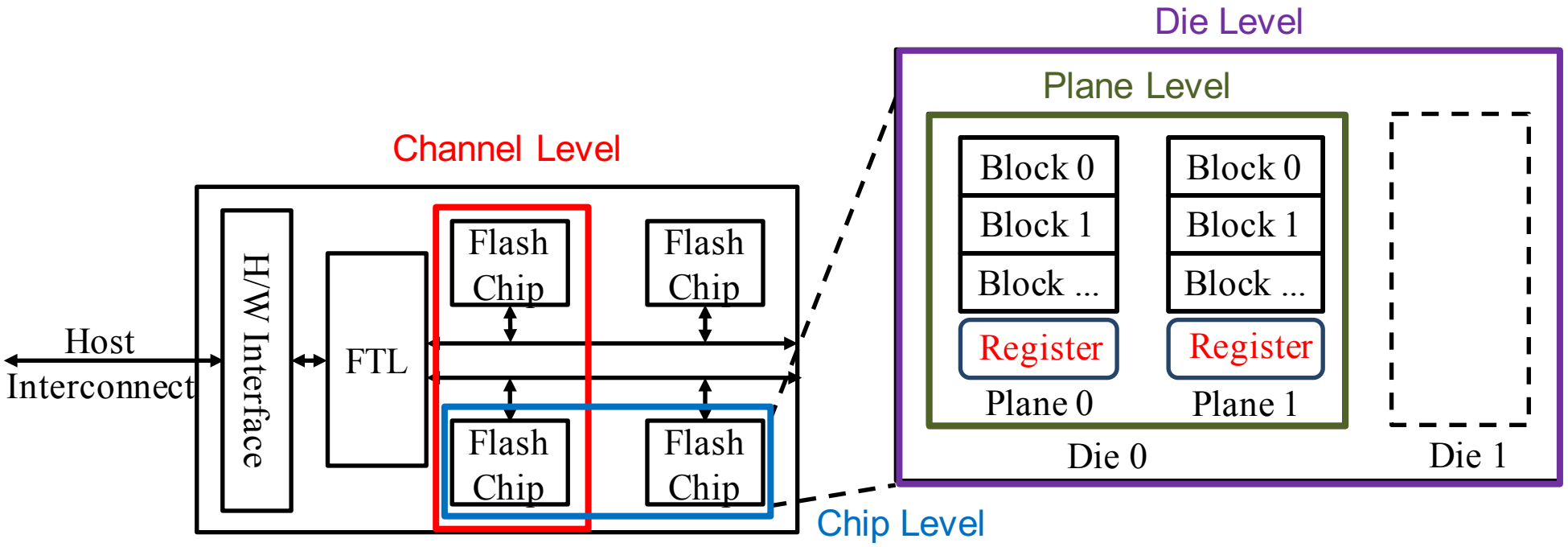
Outline

- Background and Motivation
- ParaFS Design
- Evaluation
- Conclusion

Solid State Drives – Internal Parallelism

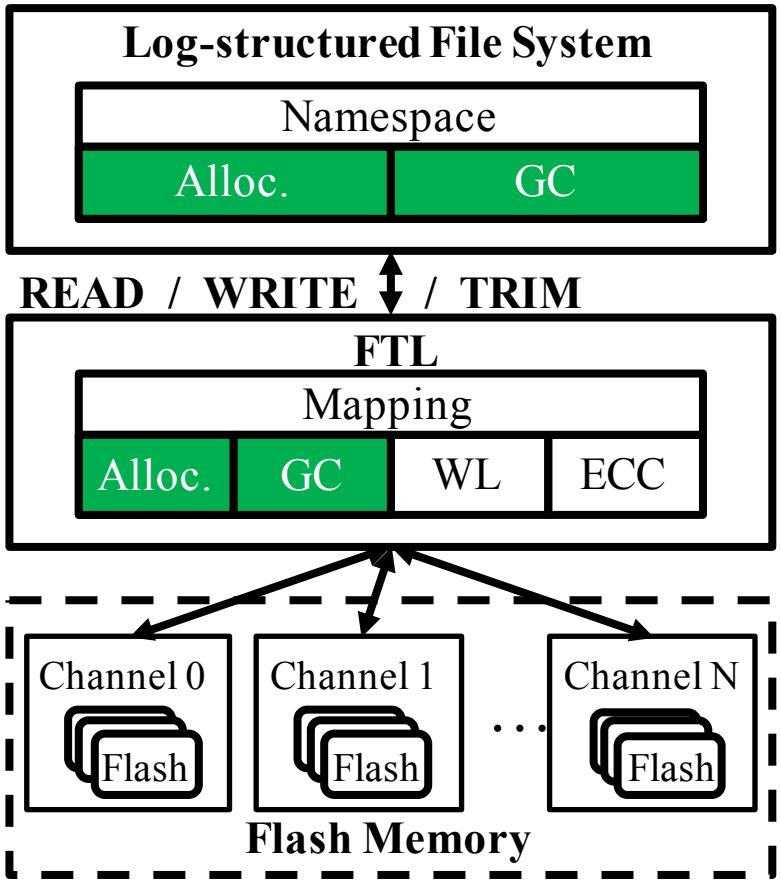
- Internal Parallelism
  - Channel Level, Chip Level, Die Level, Plane Level
  - Chips in one package share the same 8/16-bit-I/O bus, but have separated chip enable (CE) and ready/busy (R/B) control signals.
  - Each die has one internal R/B signal.
  - Each plane contains thousands of flash blocks and one data register.

✓ Internal Parallelism → High Bandwidth.



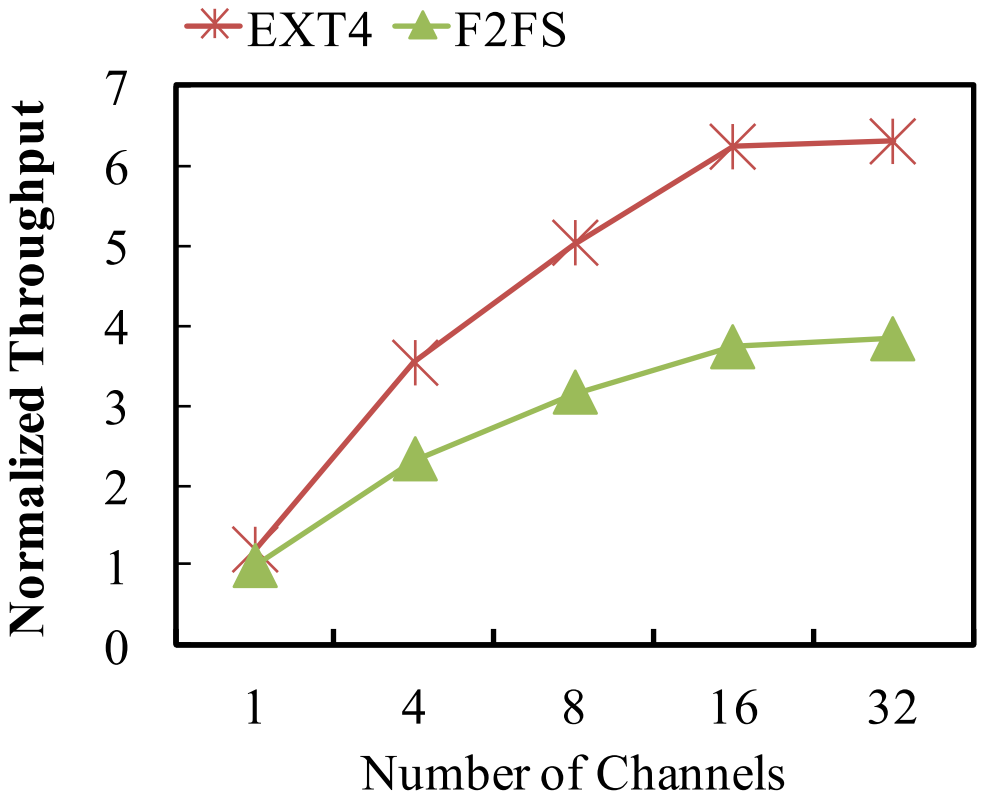
Flash File Systems

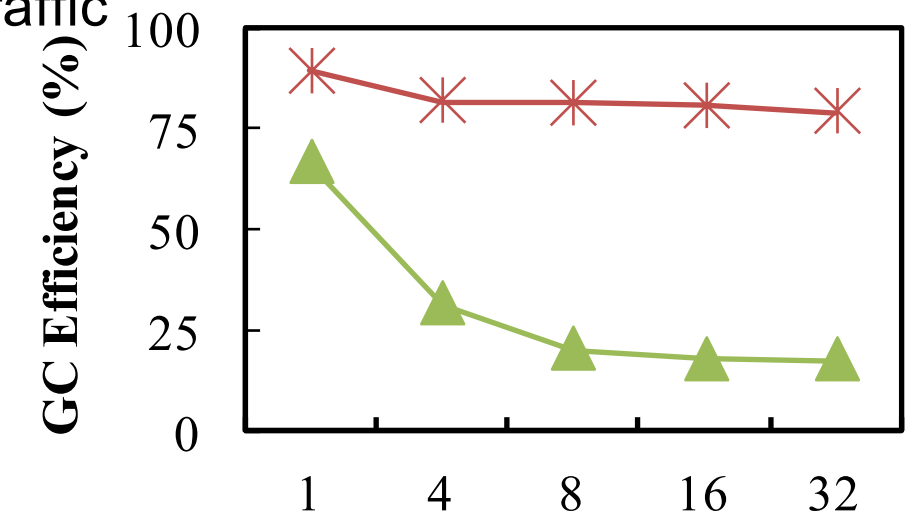
- Log-structured File System
  - Duplicate Functions: Space Allocation, Garbage Collection.
  - Semantic Isolation: FTL Abstraction, Block I/O Interface, Log on Log.

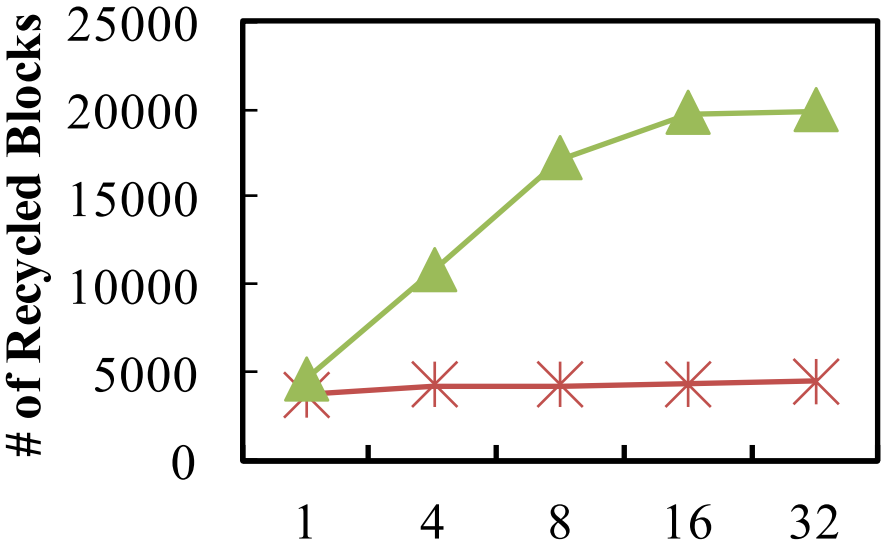


Observation

- F2FS vs. Ext4 **(under heavy write traffic)**
  - YCSB: 1000w random Read and Update operations
  - 16GB flash space + 24GB write traffic







F2FS has poorer performance than Ext4 on SSDs.

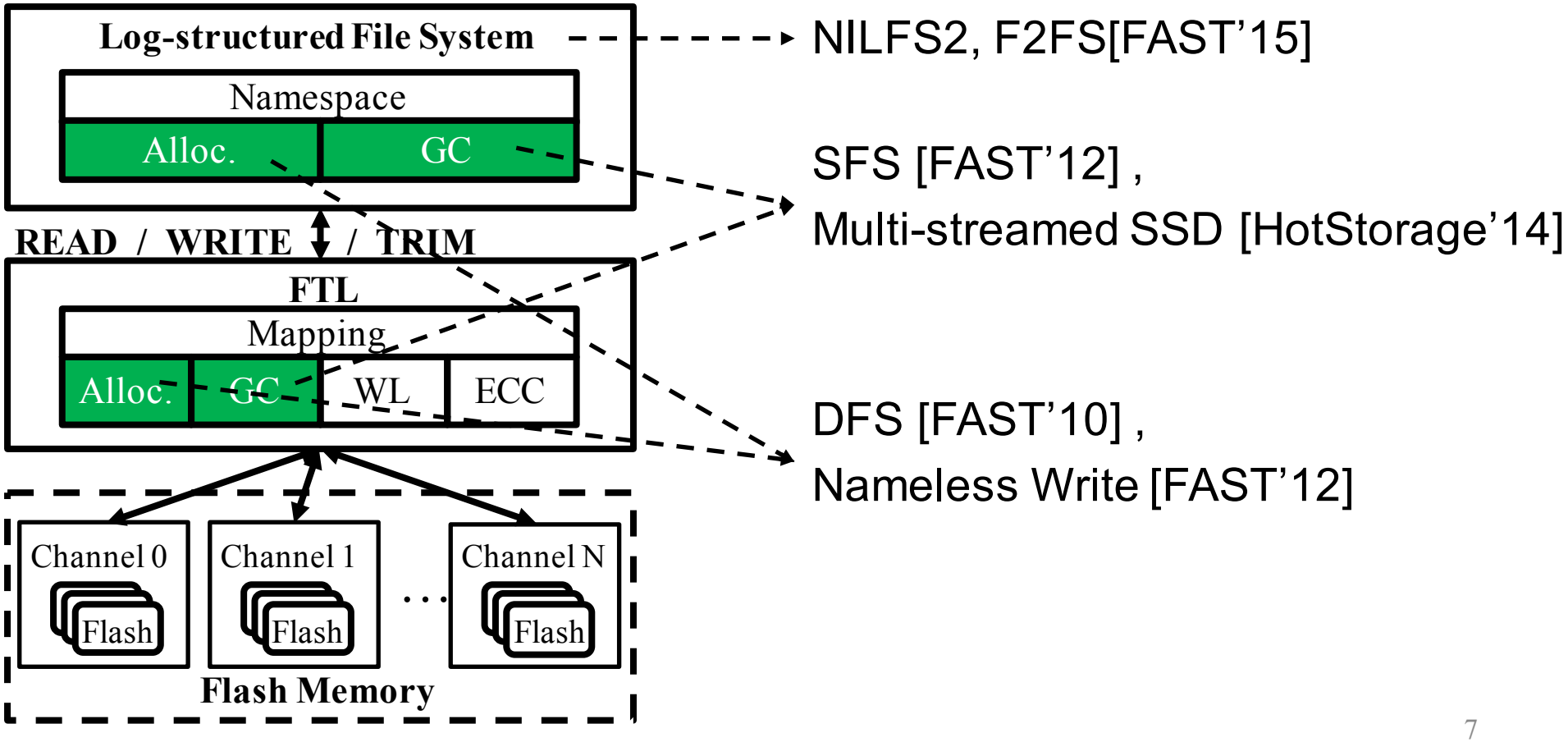
Problem

- Internal Parallelism Conflicts
  - Broken Data Grouping : Grouped data are broken and dispatched to different locations.
  - Uncoordinated GC Operations: GC processes in two levels are performed out-of-order.
  - Ineffective I/O Scheduling: erase operations always block the read/write operations, while the writes always delay the reads.

- The flash storage architecture block the optimizations.

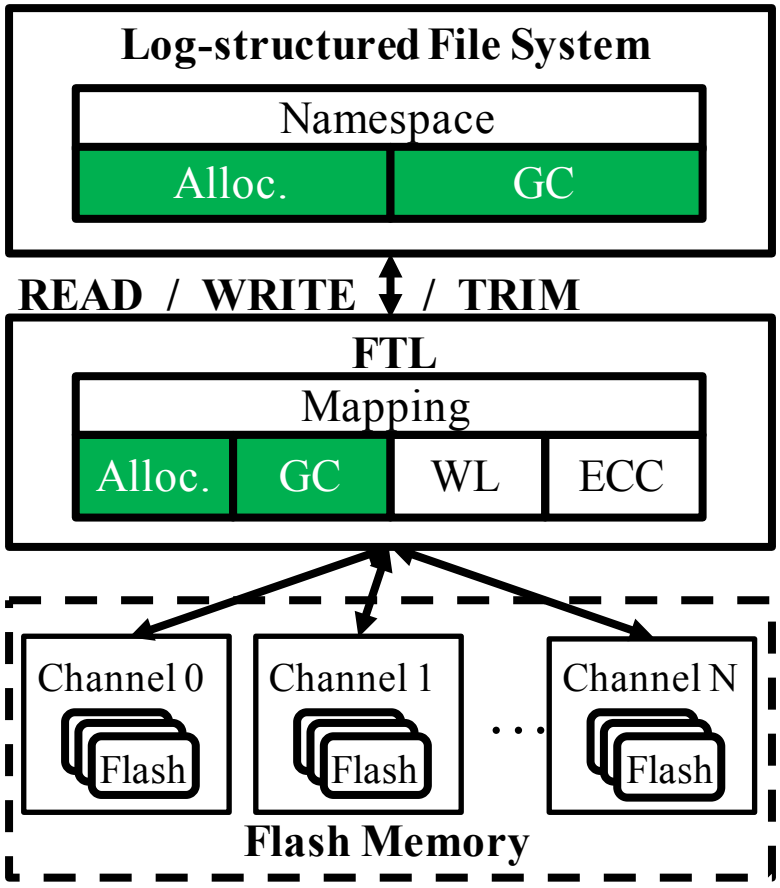
Current Approaches (1)

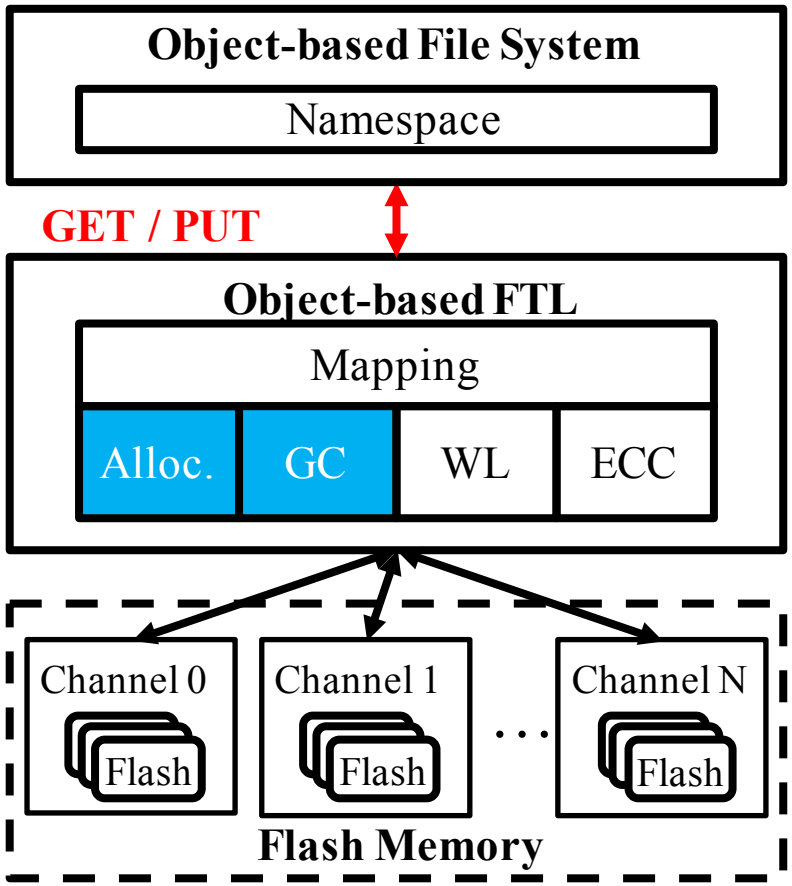
- Log-structured File System
  - Duplicate Functions: Space Allocation, Garbage Collection.
  - Semantics Isolation: FTL Abstraction, Block I/O Interface, Log on Log.



Current Approaches (2)

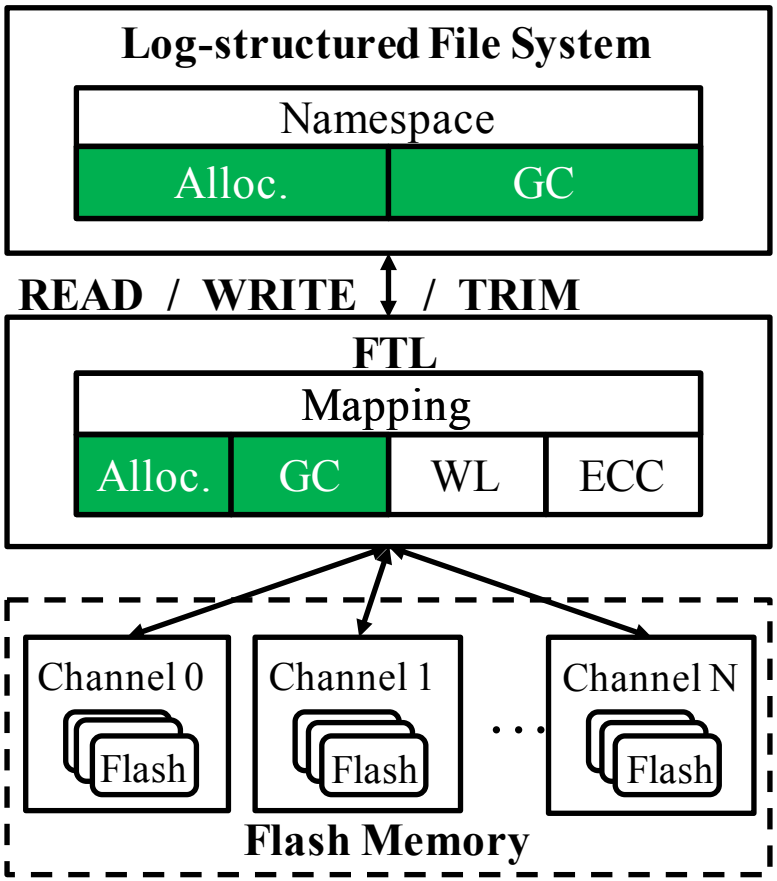
- Object-based File System [FAST'13]
  - Move the semantics from FS to FTL.
  - Difficult to be adopted due to dramatic changes
  - Internal parallelism under-explored

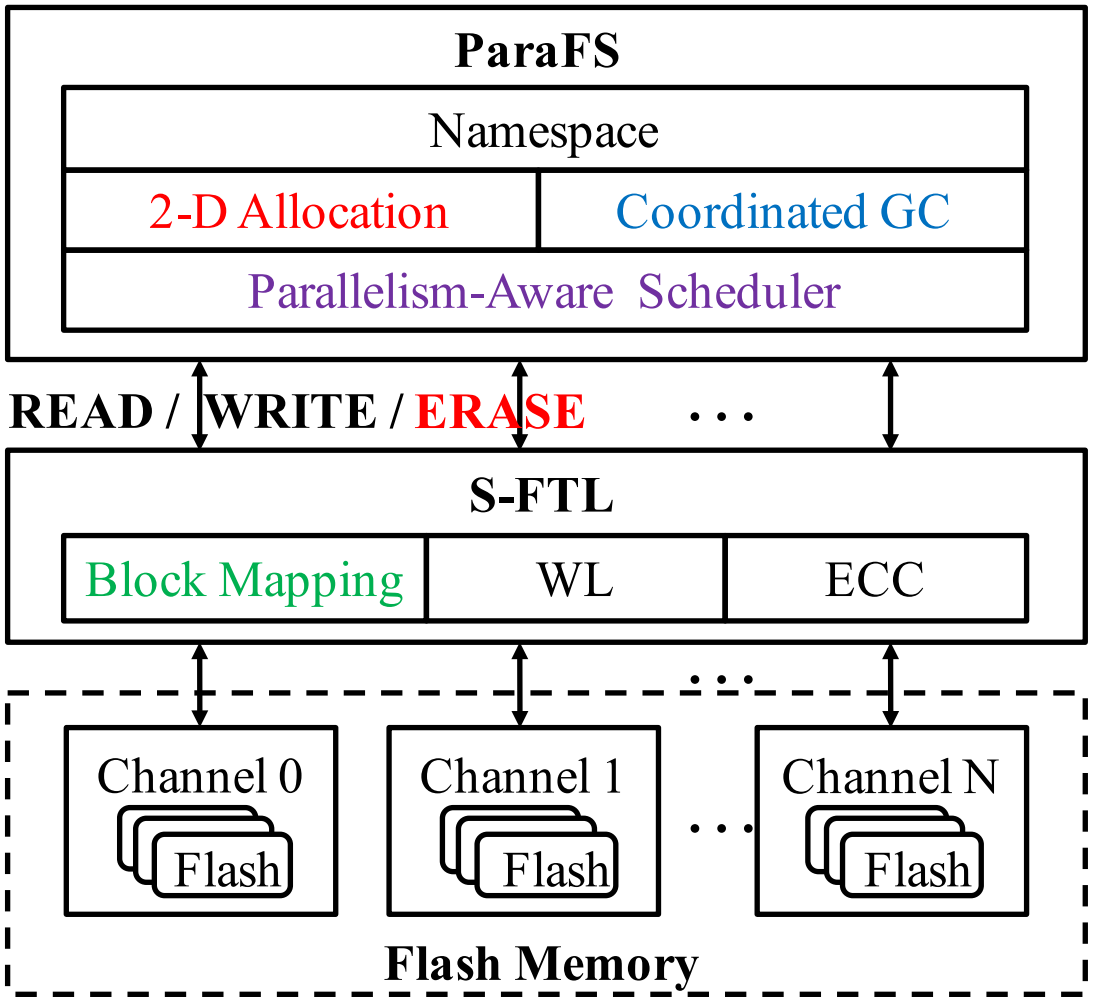




ParaFS Architecture

Goal: How to exploit the internal parallelism of the flash devices while ensuring effective data grouping, efficient garbage collection, and consistent performance?



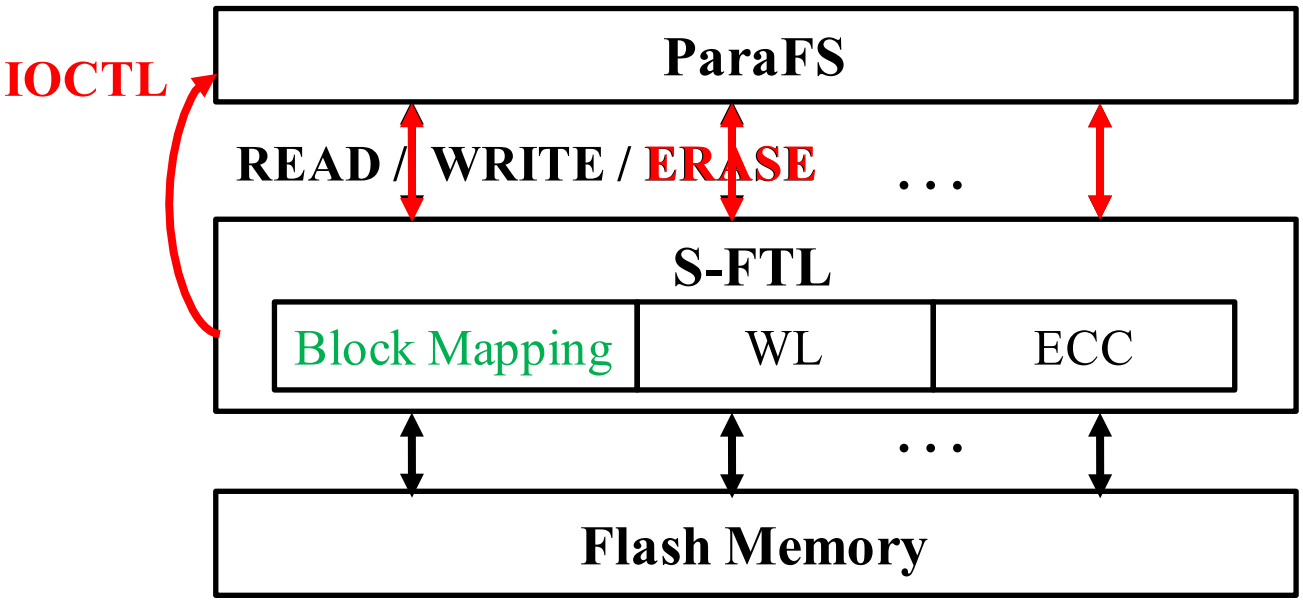


Outline

- Background and Motivation
- ParaFS Design
  - ParaFS Architecture
  - 2-D Data Allocation
  - Coordinated Garbage Collection
  - Parallelism-Aware Scheduling
- Evaluation
- Conclusion

ParaFS Architecture

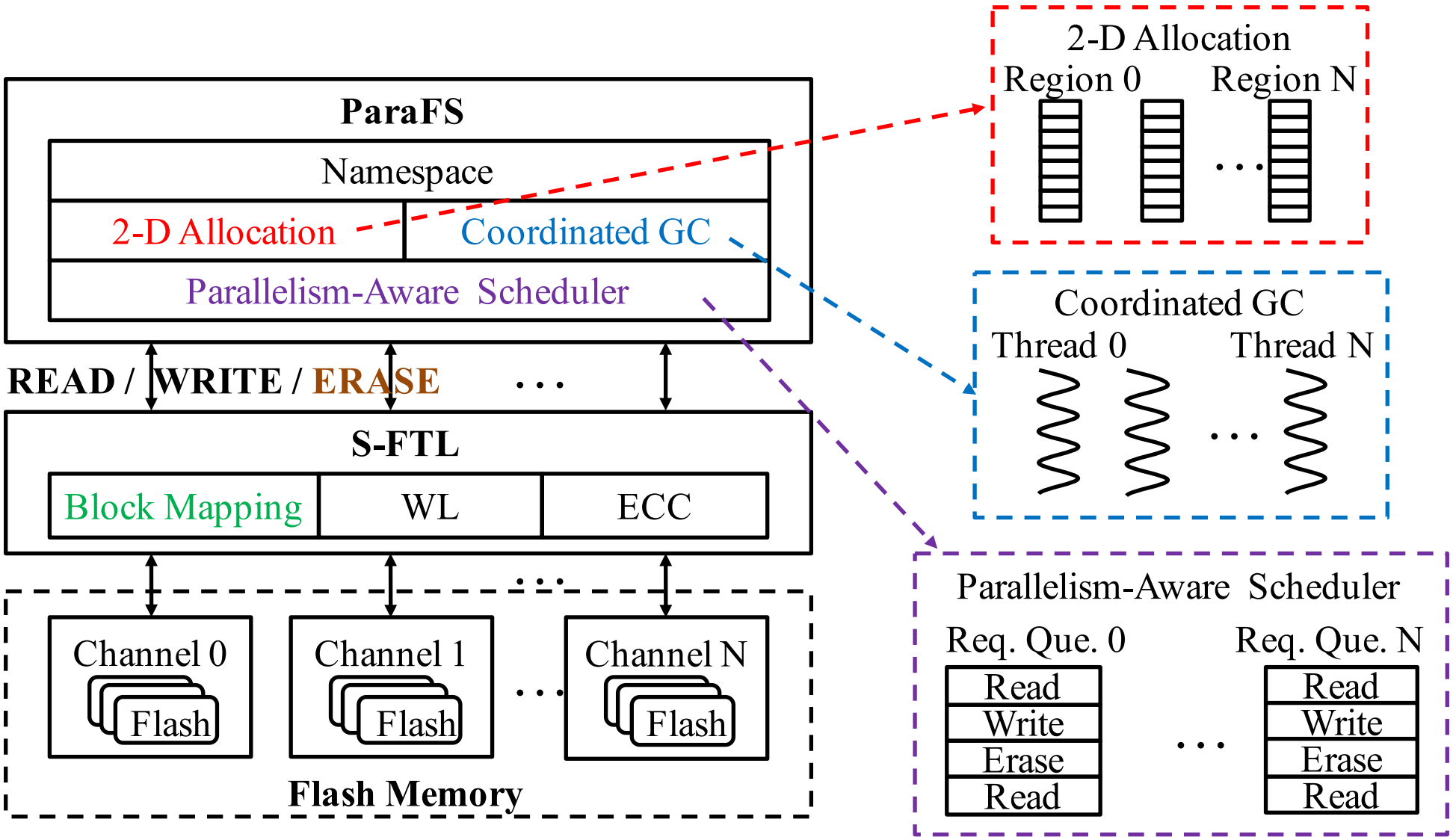
- Simplified FTL (S-FTL)
  - Exposing Physical Layout to FS: # of flash channels, Size of flash block, Size of flash page.
  - Static Block Mapping: Block-level, rarely modified.
  - Data Allocation Functionality is removed.
  - GC process is simplified to Erase process.
  - WL, ECC: functions which need hardware supports.



- Interface
  - ioctl
  - Erase → Trim

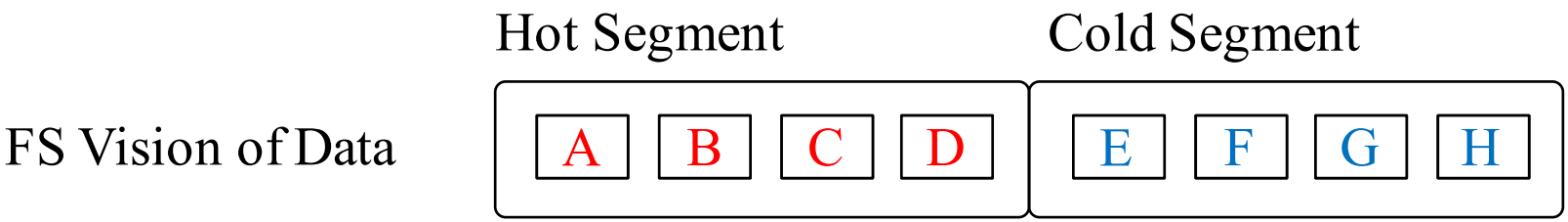
ParaFS Architecture

- ParaFS: Allocation, Garbage Collection, Scheduling

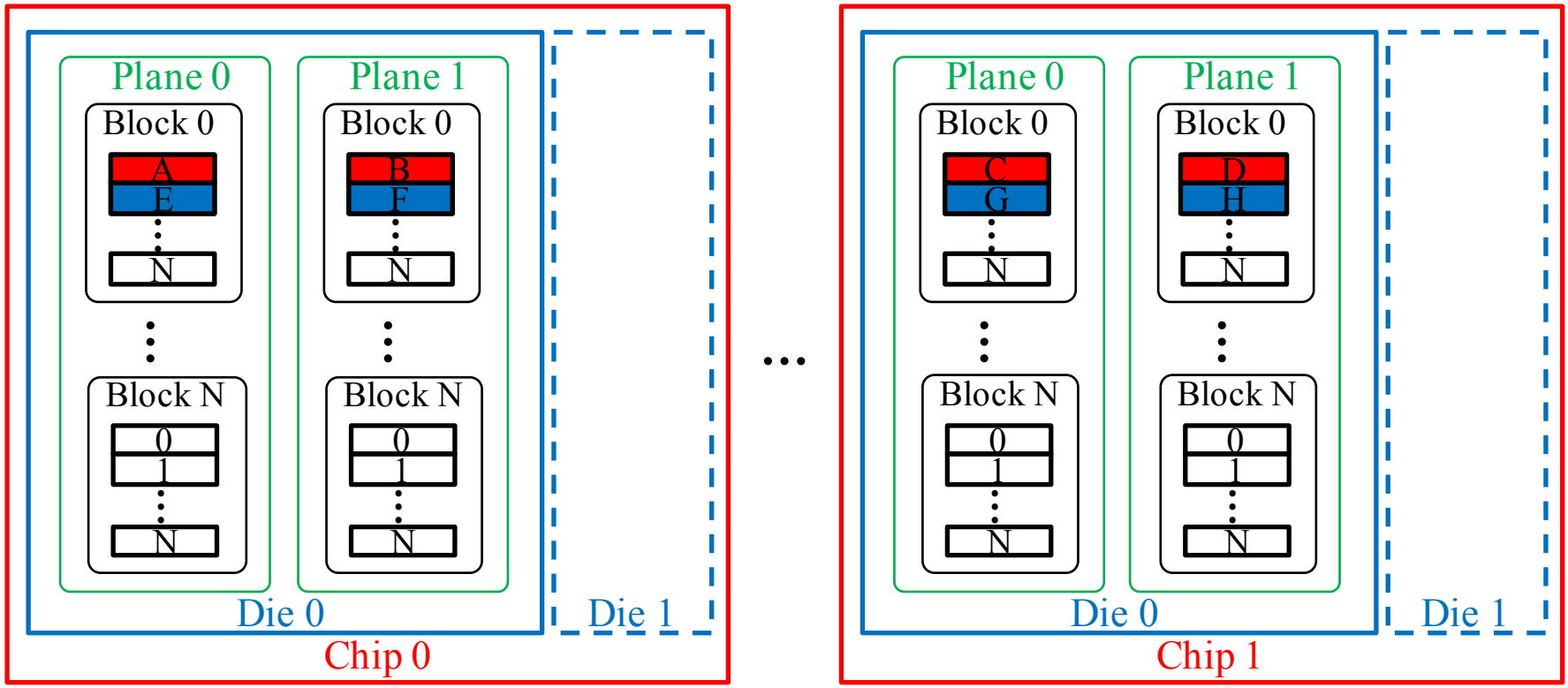


Problem #1: Grouping vs. Parallelism

- Hot/Cold Data Grouping

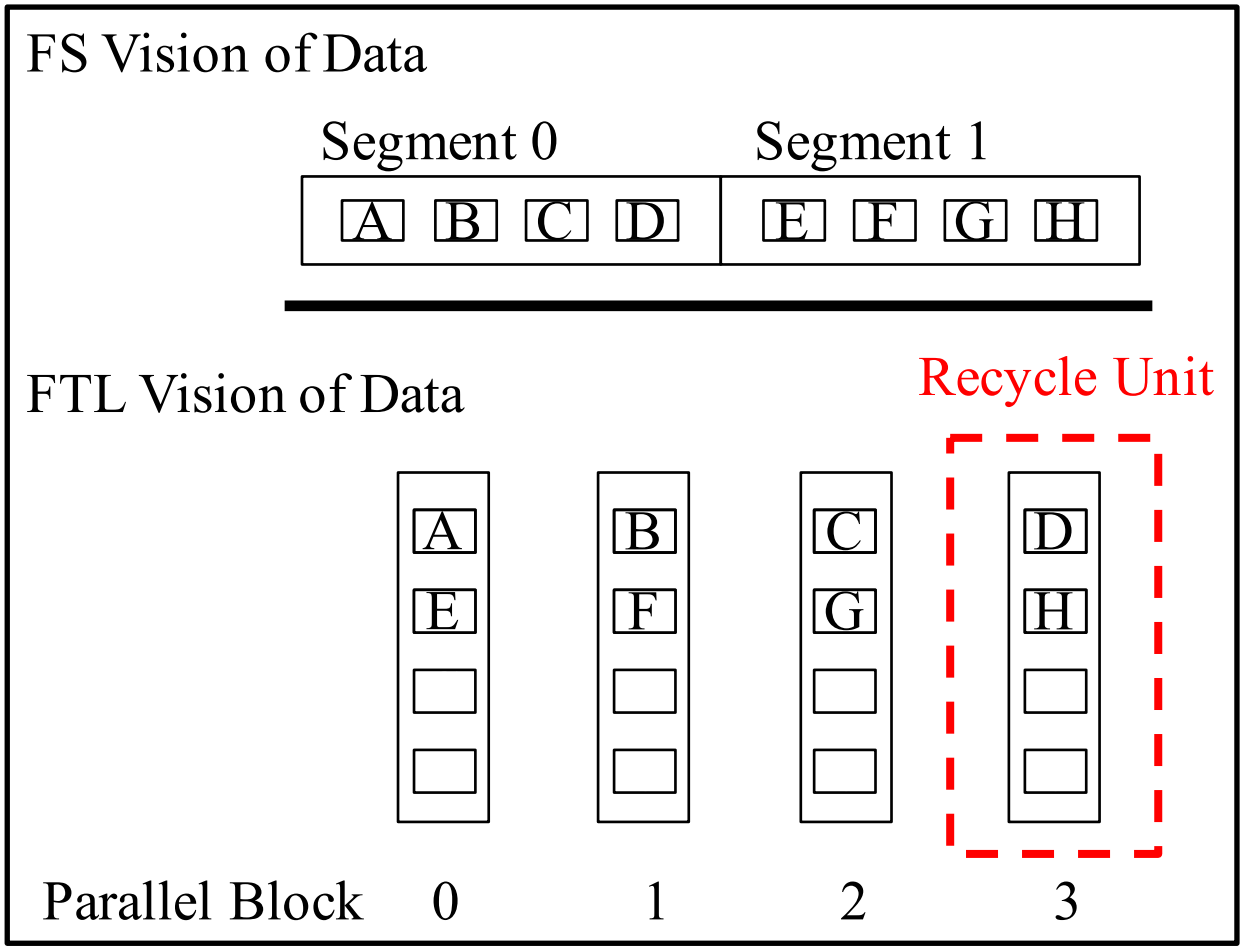


FTL Vision of Data



1. 2-D Data Allocation

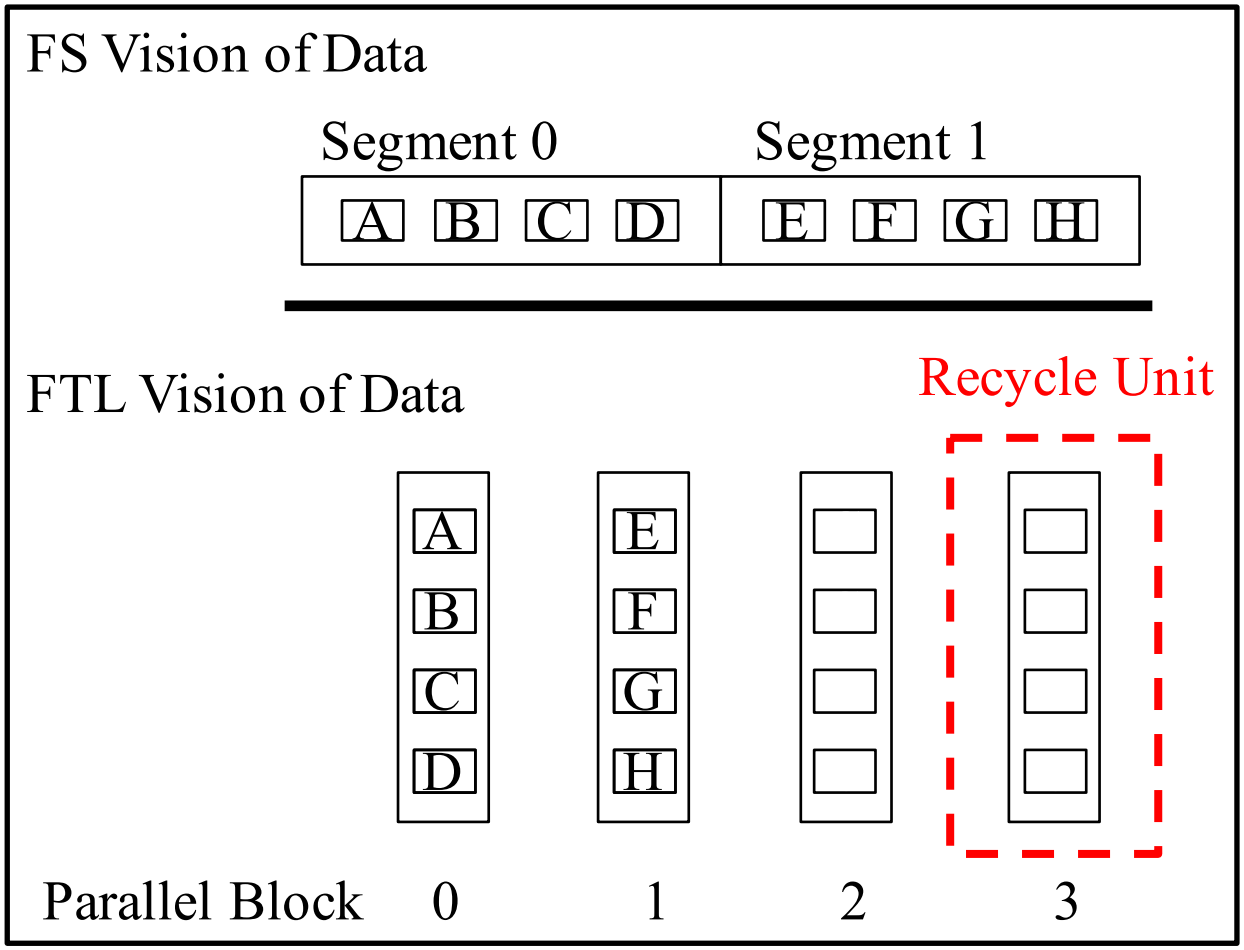
- Current Data Allocation Schemes
  - Page Stripe
  - Block Stripe
  - Super Block



- Exploiting internal parallelism with page unit striping causes heavy garbage collection overhead.

1. 2-D Data Allocation

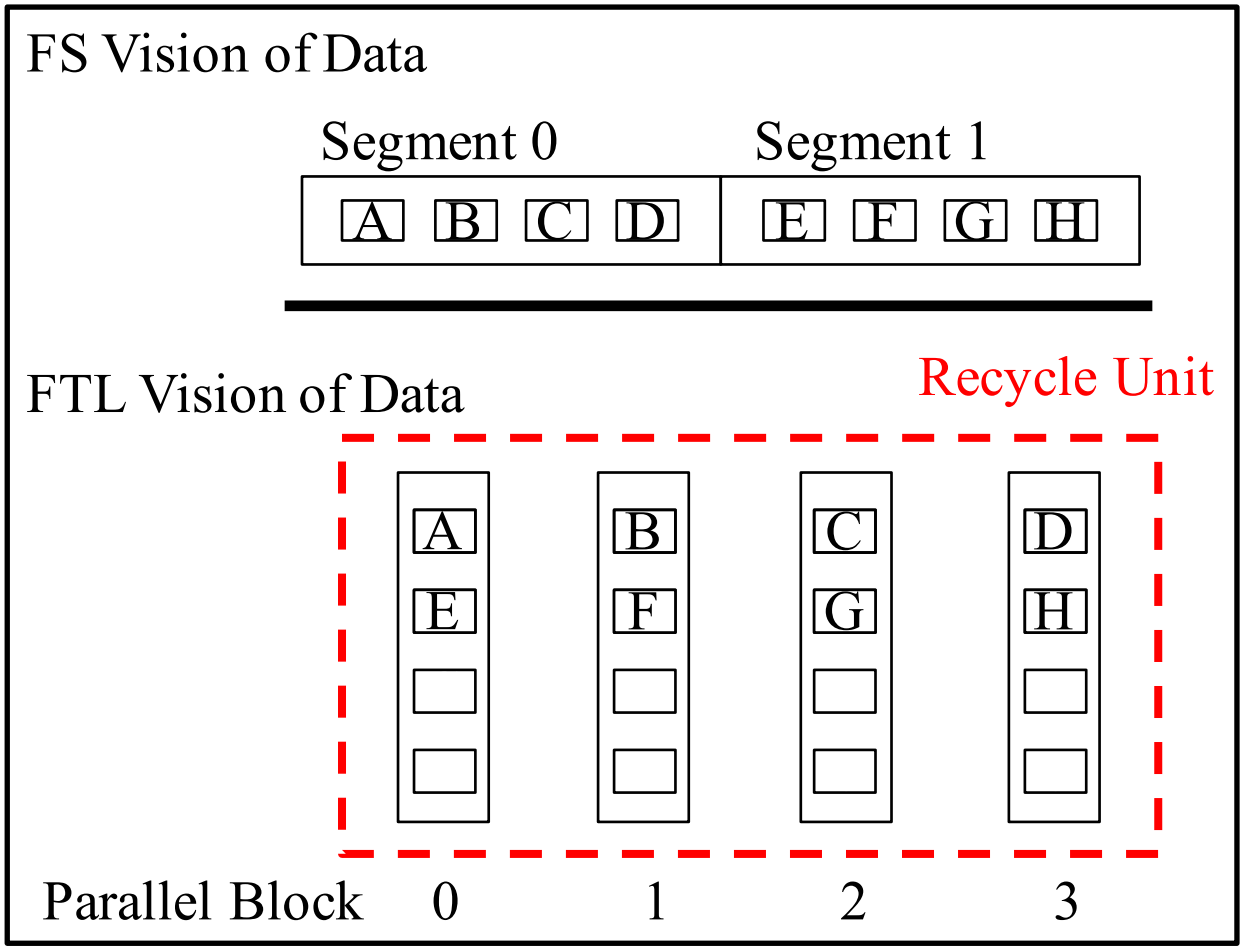
- Current Data Allocation Schemes
  - Page Stripe
  - Block Stripe
  - Super Block



- Not fully exploiting internal parallelism of the device, and performs badly in small sync write situation, mailserver.

1. 2-D Data Allocation

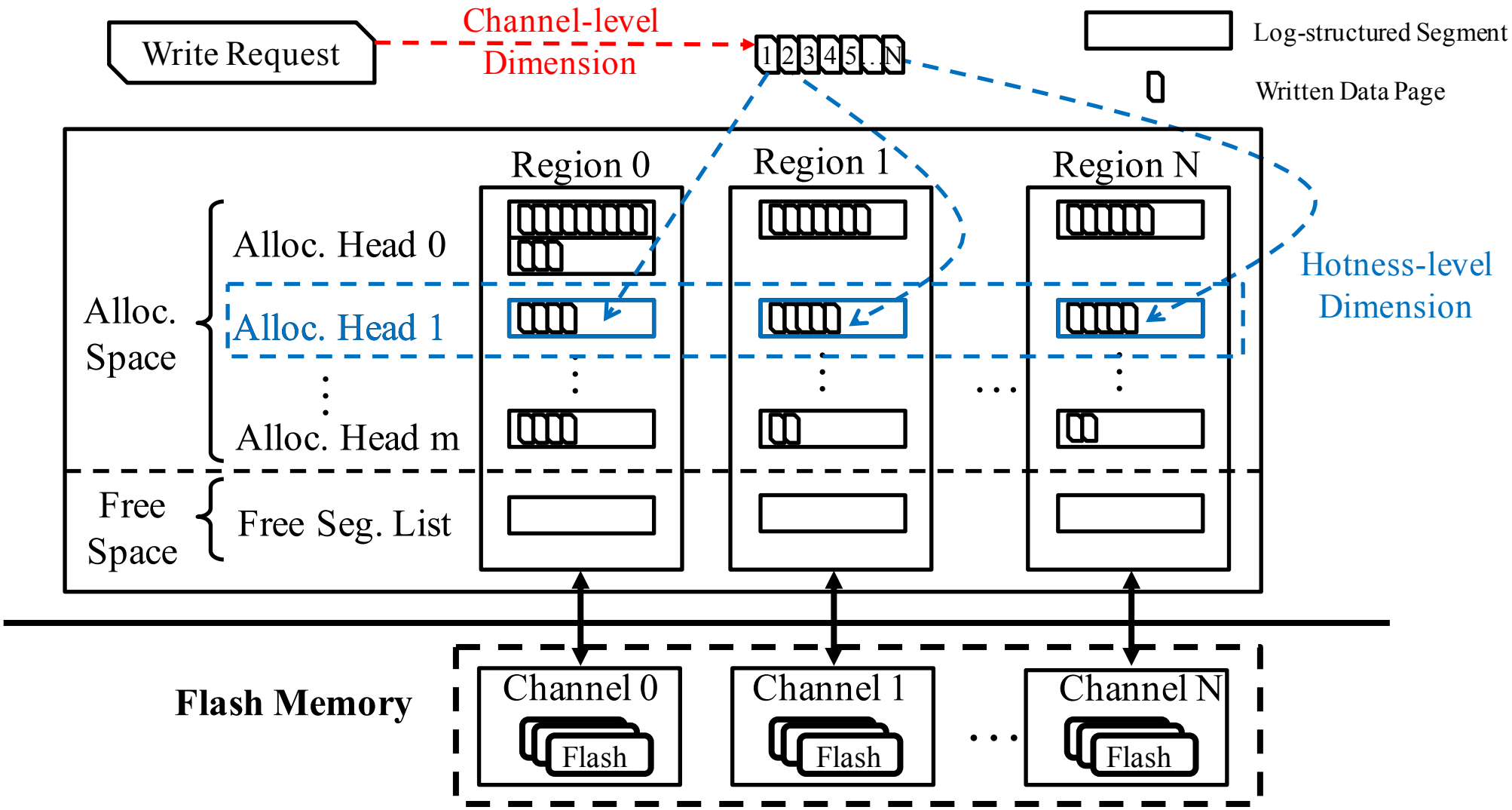
- Current Data Allocation Schemes
  - Page Stripe
  - Block Stripe
  - Super Block



- Larger GC unit causes extra garbage collection overhead.

1. 2-D Data Allocation

- Aligned Data Layout in ParaFS
  - Region → Flash Channel Segment → Flash Block Page → Flash Page



1. 2-D Data Allocation

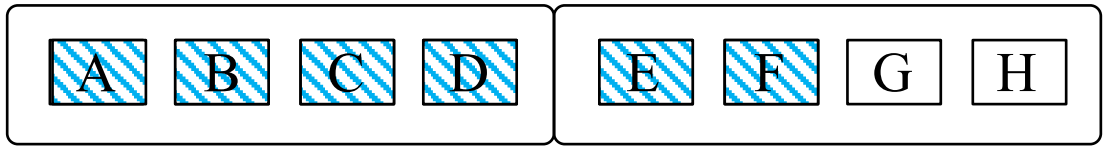
- Data Allocation Schemes Comparison
  - Page Stripe
  - Block Stripe
  - Super Block
  - 2-D Allocation

	Parallelism		Garbage Collection		
	Stripe Granularity	Parallelism Level	GC Granularity	Grouping Maintenance	GC Overhead
Page Stripe	**Page**	**High**	**Block**	**No**	**High**
Block Stripe	**Block**	**Low**	**Block**	**Yes**	**Low**
Super Block	**Page**	**High**	**Multiple Blocks**	**Yes**	**Medium**
2-D Allocation	**Page**	**High**	**Block**	**Yes**	**Low**

Problem #2: GC vs. Parallelism

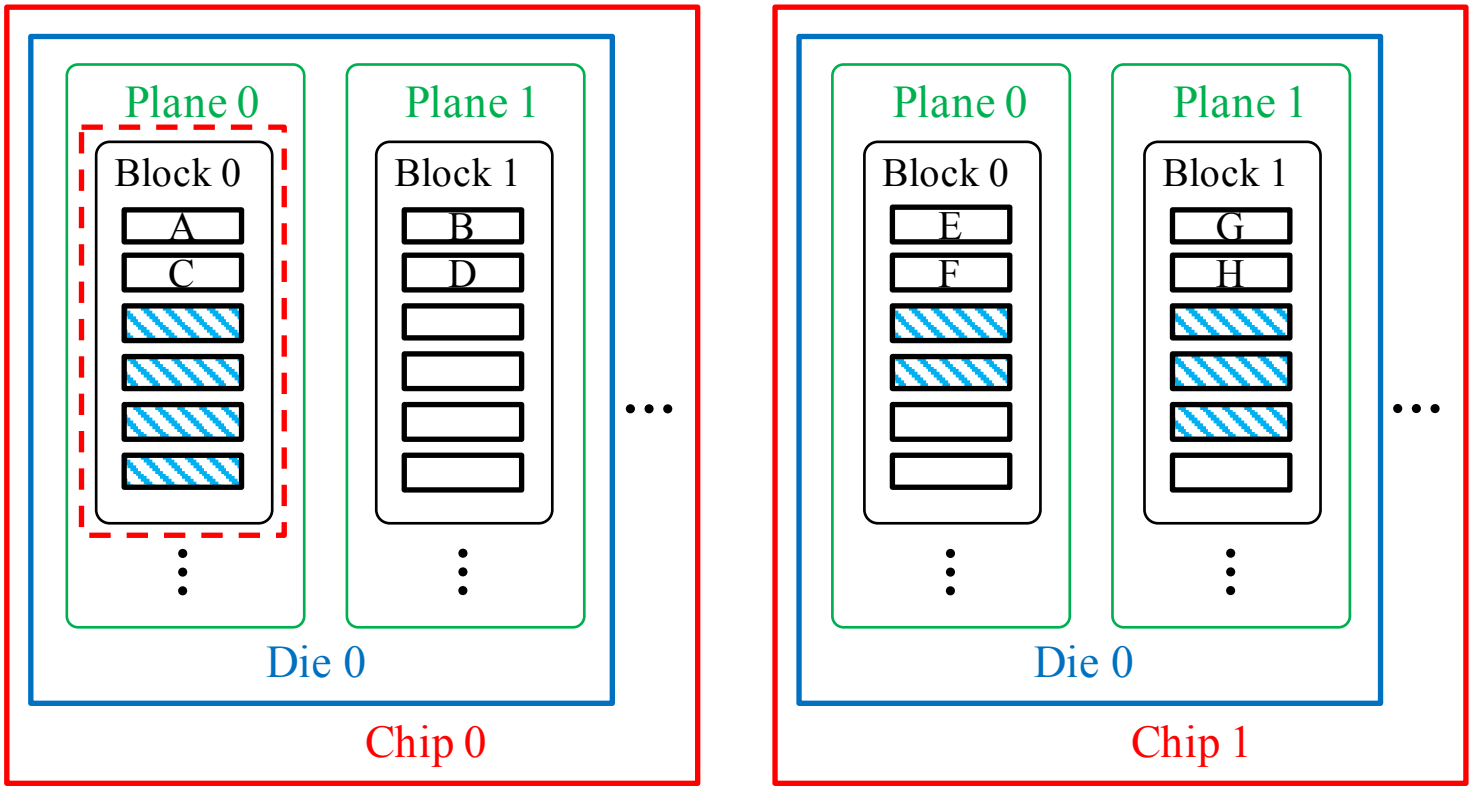
- Uncoordinated Garbage Collection

FS Vision of Data



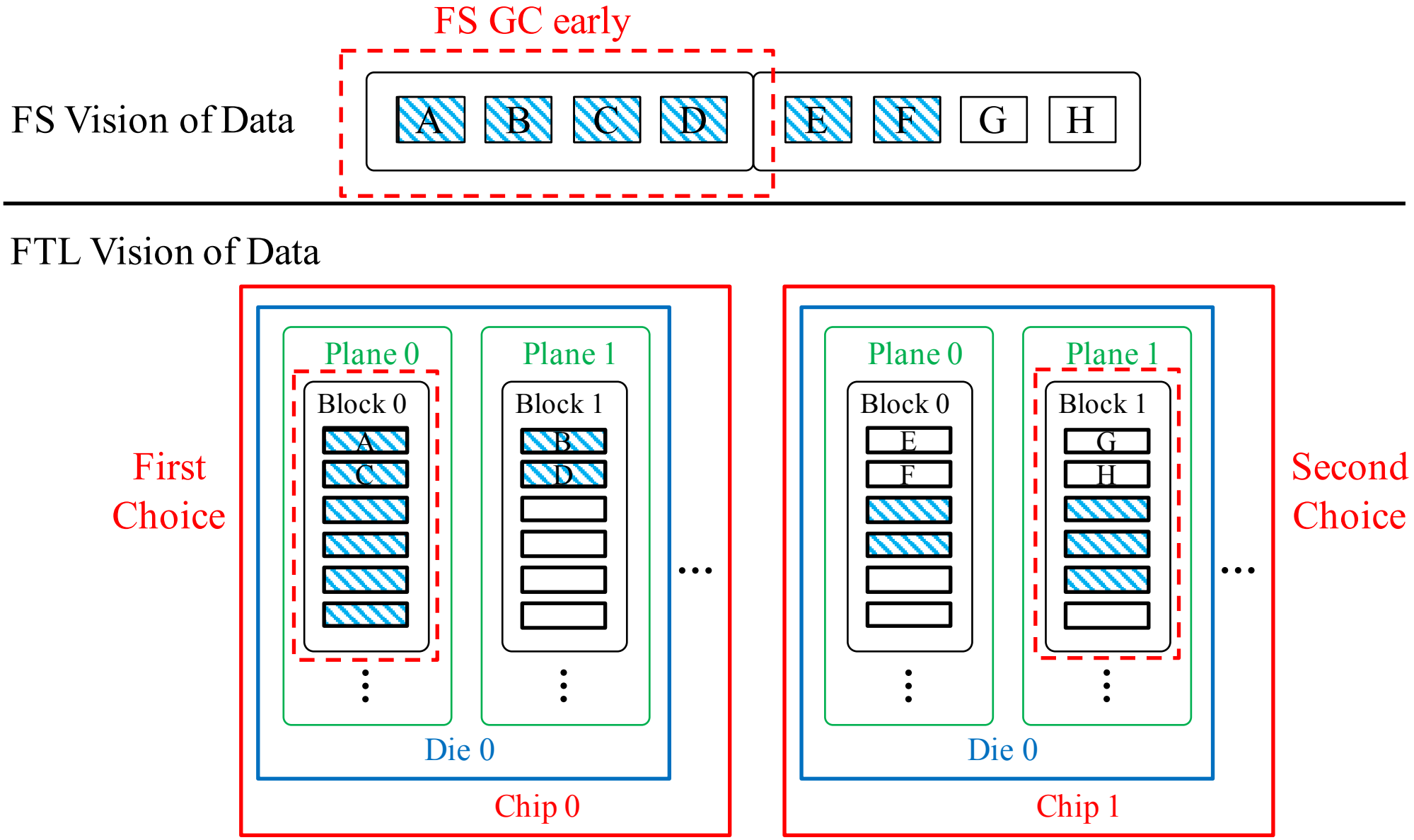
FTL Vision of Data

FTL GC early



Problem #2: GC vs. Parallelism

- Uncoordinated Garbage Collection



2. Coordinated Garbage Collection

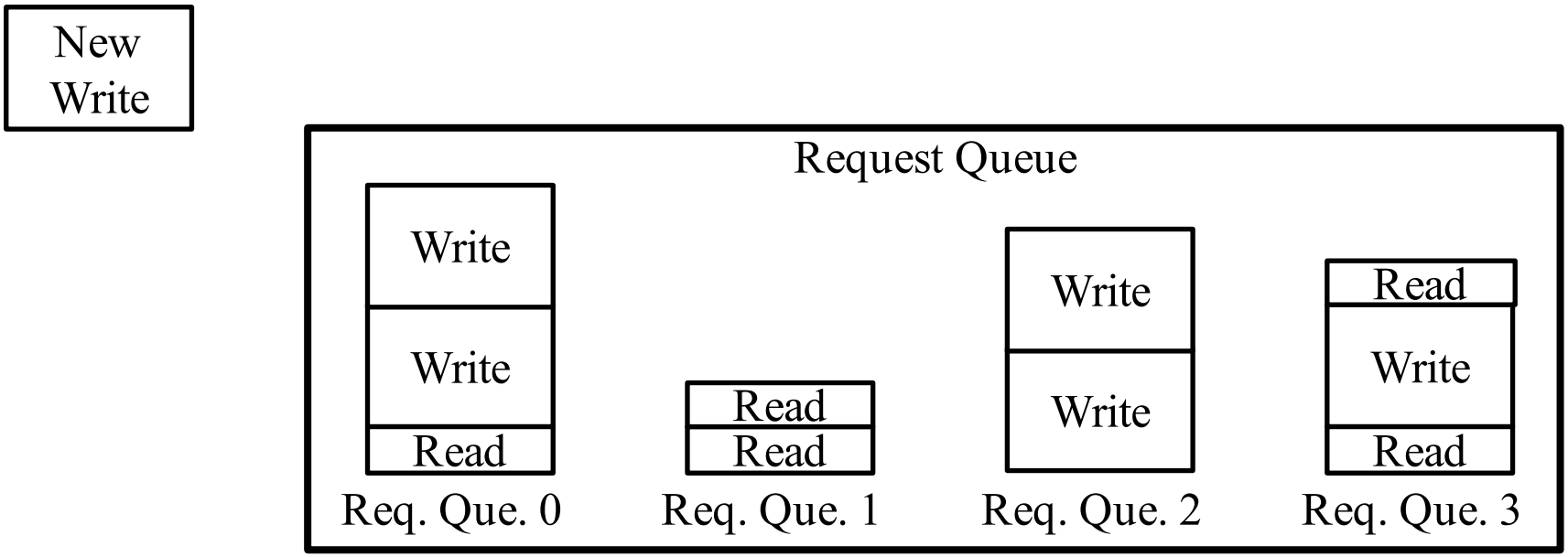
- Garbage Collection in Two levels brings overheads
  - FTL only gets page invalidation after Trim commands.
    - Due to the no-overwrite pattern.
    - Pages that are invalid in the FS, moved during FTL-level GC.
  - Unexpected timing of GC starts in two levels.
    - FTL GC starts before FS GC starts.
    - FTL GC blocks the FS I/O and causes unexpected I/O latency.
  - Waste Space.
    - Each level keeps over-provision space for GC efficiency.
    - Each level keeps meta data space, to record page and block(segment) utilization, to remapping.

2. Coordinated Garbage Collection

- Coordinated GC process in two levels
  - FS-level GC
    - Foreground GC or Background GC
      (trigger condition, selection policy)
    - Migration the valid pages from victim segments.
    - Do the checkpoint in case of crash.
    - Send the Erase Request to S-FTL.
  - FTL-level GC
    - Find the corresponding flash block by static block mapping table.
    - Erase the flash block directly.
- Multi-threaded Optimization
  - One GC process per Region (Flash Channel).
  - One Manager process to do the checkpoint after GC.

3. Parallelism-Aware Scheduling

- Request Dispatching Phase
  - Select the least busy channel to dispatch write request



$$W_{channel} = \sum (W_{read} \times Size_{read}, W_{write} \times Size_{write})$$

3. Parallelism-Aware Scheduling

- Request Dispatching Phase
  - Select the least busy channel to dispatch write request
- Request Scheduling Phase
  - Time Slice for Read Request Scheduling and Write/Erase Request Scheduling.
  - Schedule Write or Erase Request according to Space Utilization and Number of Concurrent Erasing Channels.

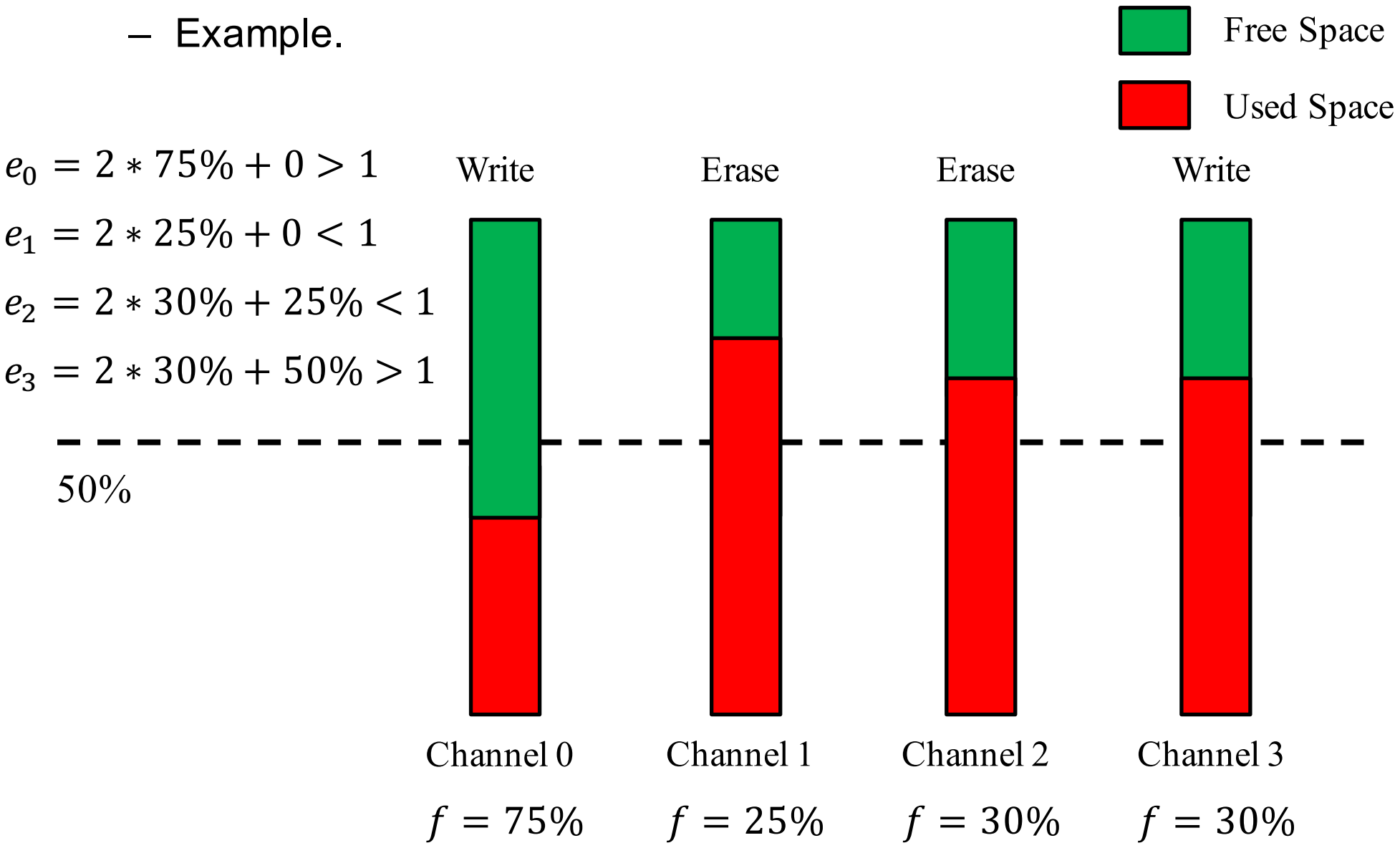
$$e = a \times f + b \times N_e$$

  - In implementation

$$1 \overset{?}{=} 2 \times f + 1 \times N_e$$

3. Parallelism-Aware Scheduling

- Request Scheduling Phase
  - Example.



Outline

- Background and Motivation
- ParaFS Design
- Evaluation
- Conclusion

Evaluation

- ParaFS implemented on Linux kernel 2.6.32 based on F2FS
  - Ext4 (In-place update), BtrFS (CoW), back-ported F2FS (Log-structured update)
  - F2FS_SB: back-ported F2FS with large segment configuration
- Testbed:
  - PFTL, S-FTL

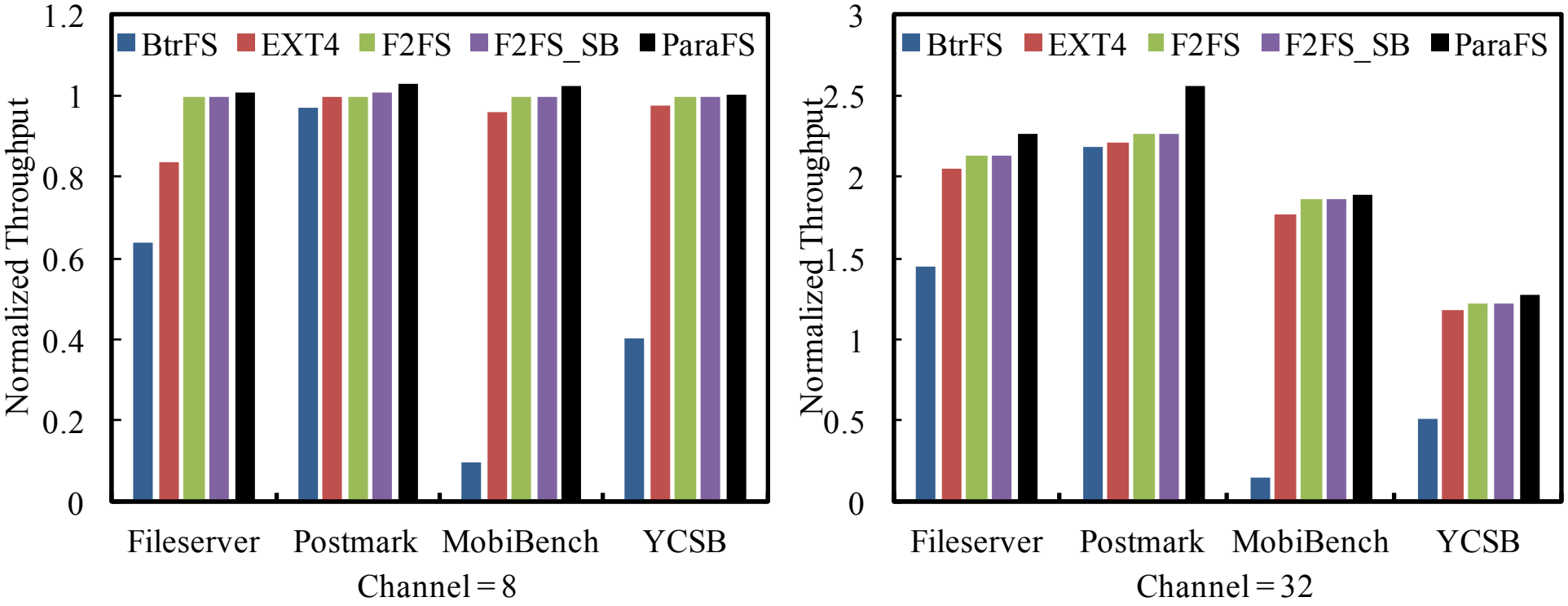
Host Interface	PCIe 2.0 x8
Number of Flash Channel	34
Capacity per Channel	32G
NAND Type	25nm MLC
Page Size	8KB
Block Size	2MB
Read Bandwidth per Channel	49.84 MB/s
Write Bandwidth per Channel	6.55 MB/s

- Workloads:

Workload	Pattern	R:W	FSYNC	I/O Size
Fileserver	random read and write files	33/66	N	1 MB
Postmark	create, delete, read and append files	20/80	Y	512 B
MobiBench	random update records in SQLite	1/99	Y	4 KB
YCSB	read and update records in MySQL	50/50	Y	1 KB

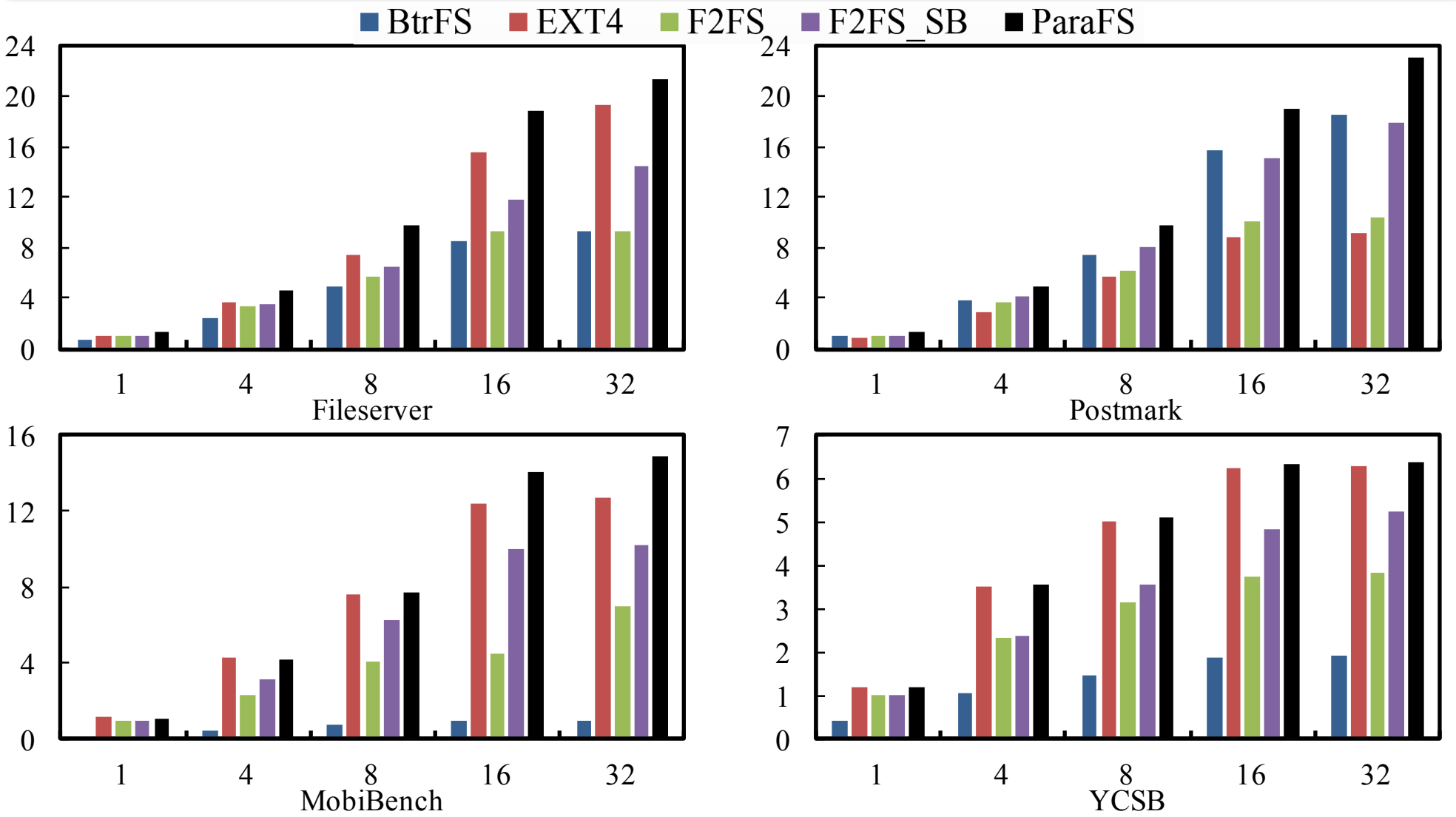
Evaluation – Light Write Traffic

- Normalized to F2FS in 8-Channel case



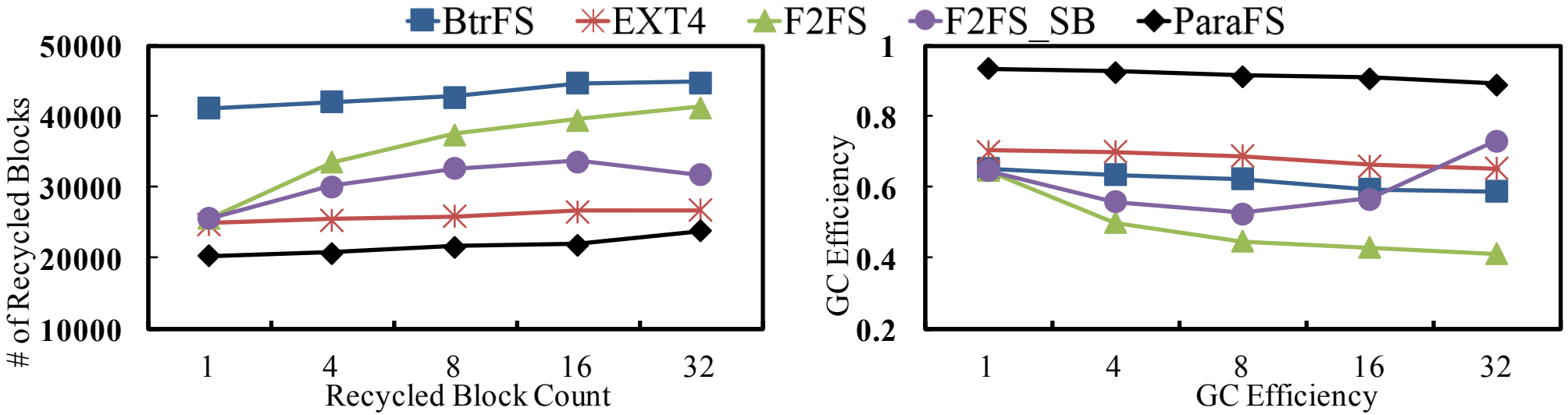
- Outperforms other file systems in all cases.
- Improves performance by up to 13% (Postmark 32-Channel)

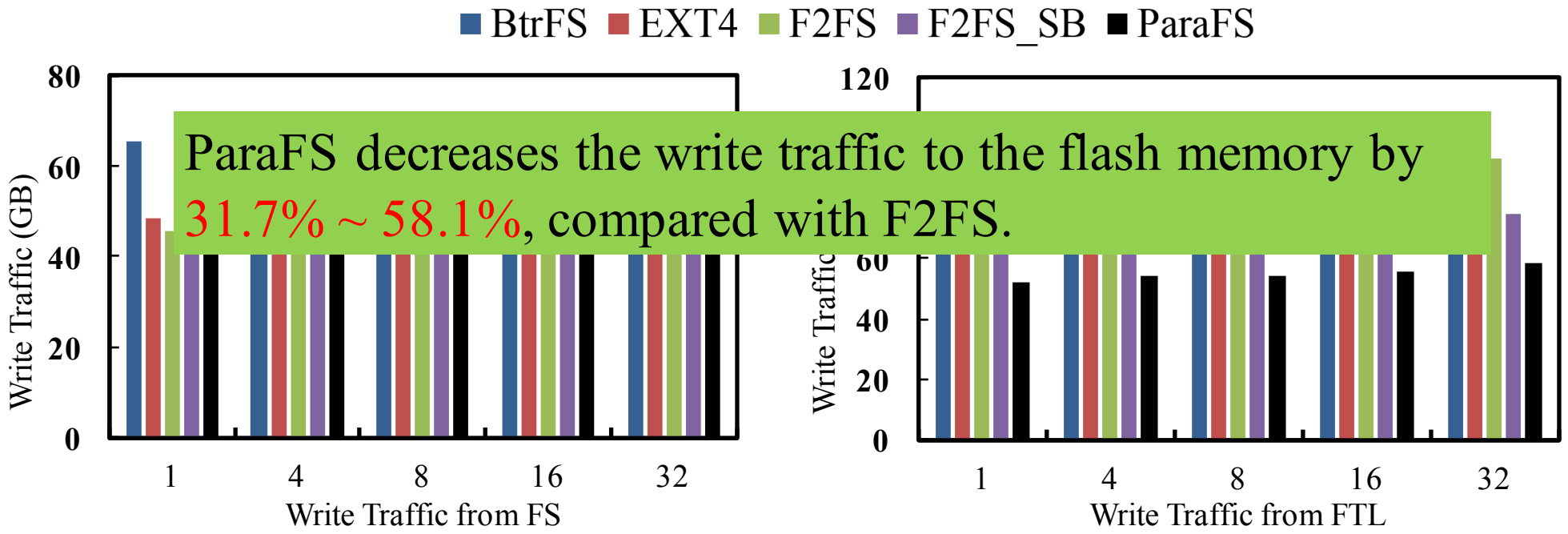
Evaluation – Heavy Write Traffic



- Achieves the best throughput among all evaluated file systems
- Outperforms: Ext4(1.0X ~ 2.5X) F2FS(1.6X ~ 3.1X) F2FS_SB(1.2X ~1.5X)

Evaluation - Endurance

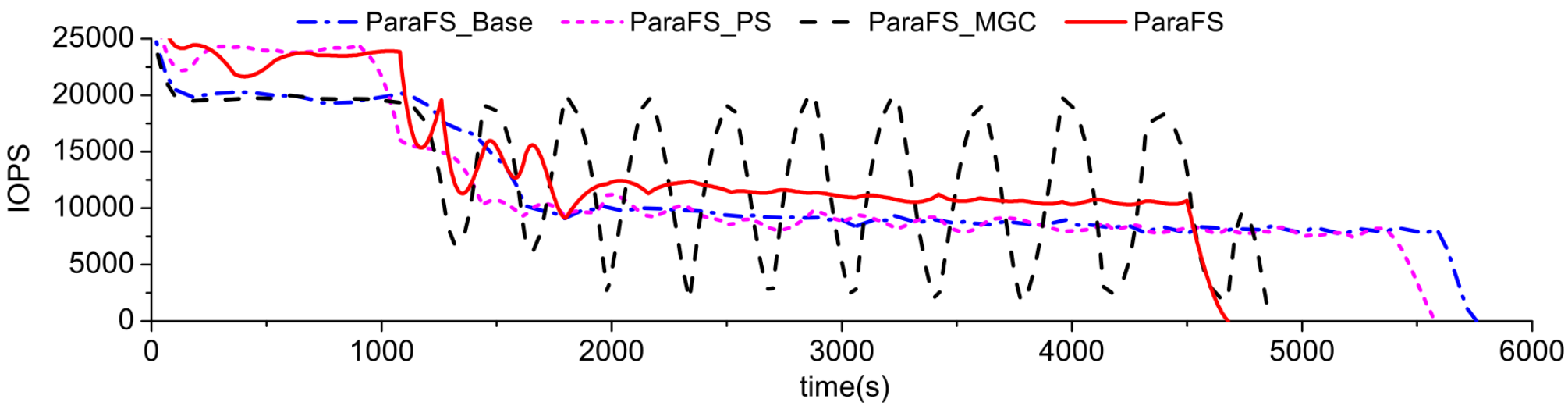




ParaFS decreases the write traffic to the flash memory by 31.7% ~ 58.1%, compared with F2FS.

Evaluation – Performance Consistency

- Two Optimizations
  - MGC: Multi-threaded GC process
  - PS: Parallelism-aware Scheduling



- Multi-threaded GC improves 18.5% (MGC vs. Base)
- Parallelism-aware Scheduling: 20% improvement in first stage. (PS vs. Base)
  much consistent performance in GC stage. (ParaFS vs. MGC)

Conclusion

- Internal parallelism has not been leveraged in the FS-level.
  - The mechanisms: Data Grouping, Garbage Collection, I/O Scheduling
  - The architecture: Semantics Isolation & Redundant Functions

- ParaFS bridges the semantic gap and exploits internal parallelism in the FS-level.
  - (1) 2-D Allocation: page unit striping and data grouping.
  - (2) Coordinated GC: improve the GC efficiency and speed.
  - (3) Parallelism-aware Scheduling: more consistent performance.

- ParaFS shows performance improvement by up to 3.1X and write reduction by 58.1% compared to F2FS.

Thanks

ParaFS: A Log-Structured File System to Exploit the Internal Parallelism of Flash Devices

Jiacheng Zhang, Jiwu Shu, Youyou Lu
luyouyou@tsinghua.edu.cn
http://storage.cs.tsinghua.edu.cn/~lu