Research paper presentation: "De-indirection for Flash-based SSDs with Nameless Writes"

Y. Zhang, L. P. Arulraj, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau

Federico Wasserman & Rodolphe Lepigre

MOSIG - Parallel, Distributed and Embedded Systems

December 19, 2012

Outline

1. Introduction
2. SSD principles
3. Indirection in SSDs
4. Nameless Writes
5. Evaluation
6. Conclusion

1 Introduction

2 SSD principles

3 Indirection in SSDs

4 Nameless Writes

5 Evaluation

6 Conclusion

What are Nameless Writes?

- New device interface for SSDs
- Indirection is used to improve reliability
- Remove the need for indirection
- Idea: the device chooses WHERE to write

How are Nameless Writes different?

Usual Writes:

- The FS requests the writing of data at some location
- The device performs the write

Nameless Writes:

- The FS requests the writing of data
- The device performs the write
- Address returned to the FS

1 Introduction

2 SSD principles

3 Indirection in SSDs

4 Nameless Writes

5 Evaluation

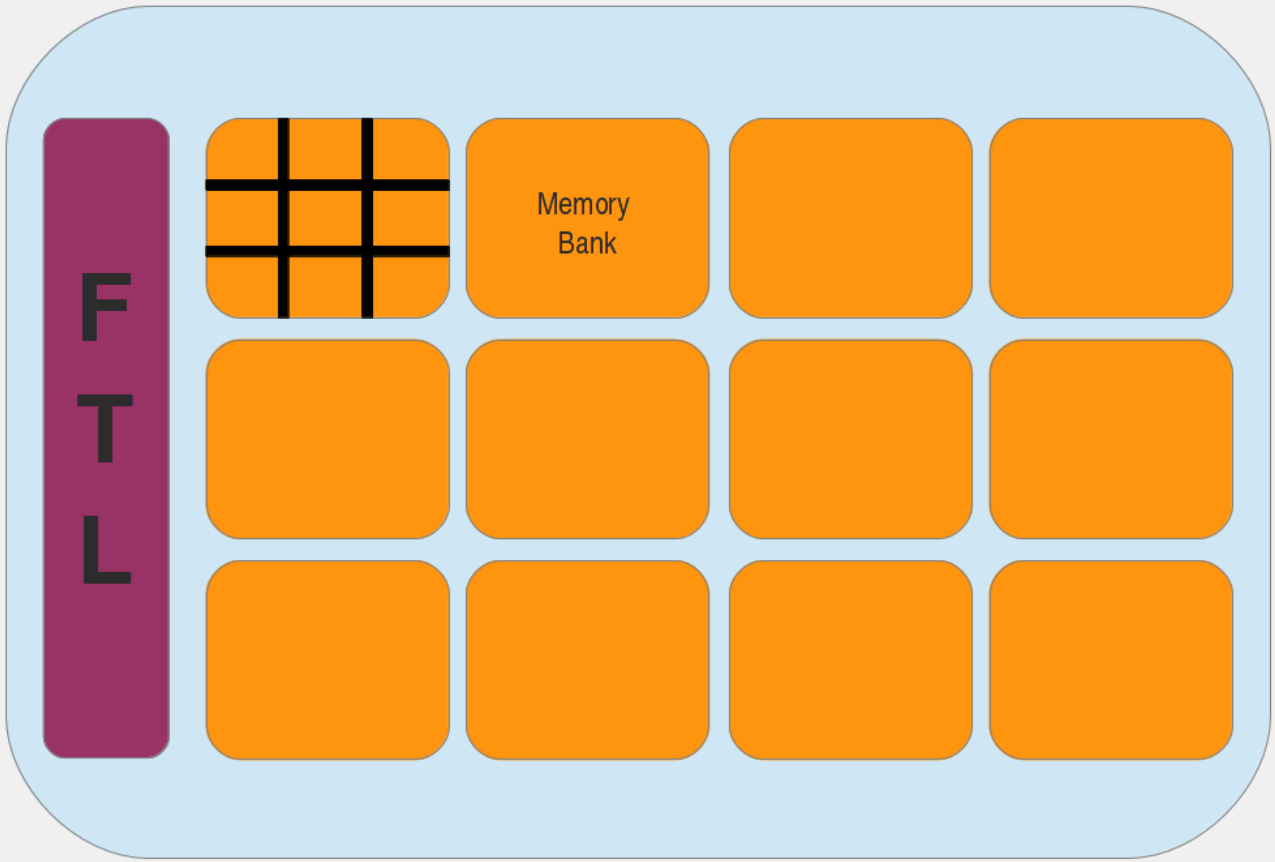
6 Conclusion

SSDs are not Hard Drives

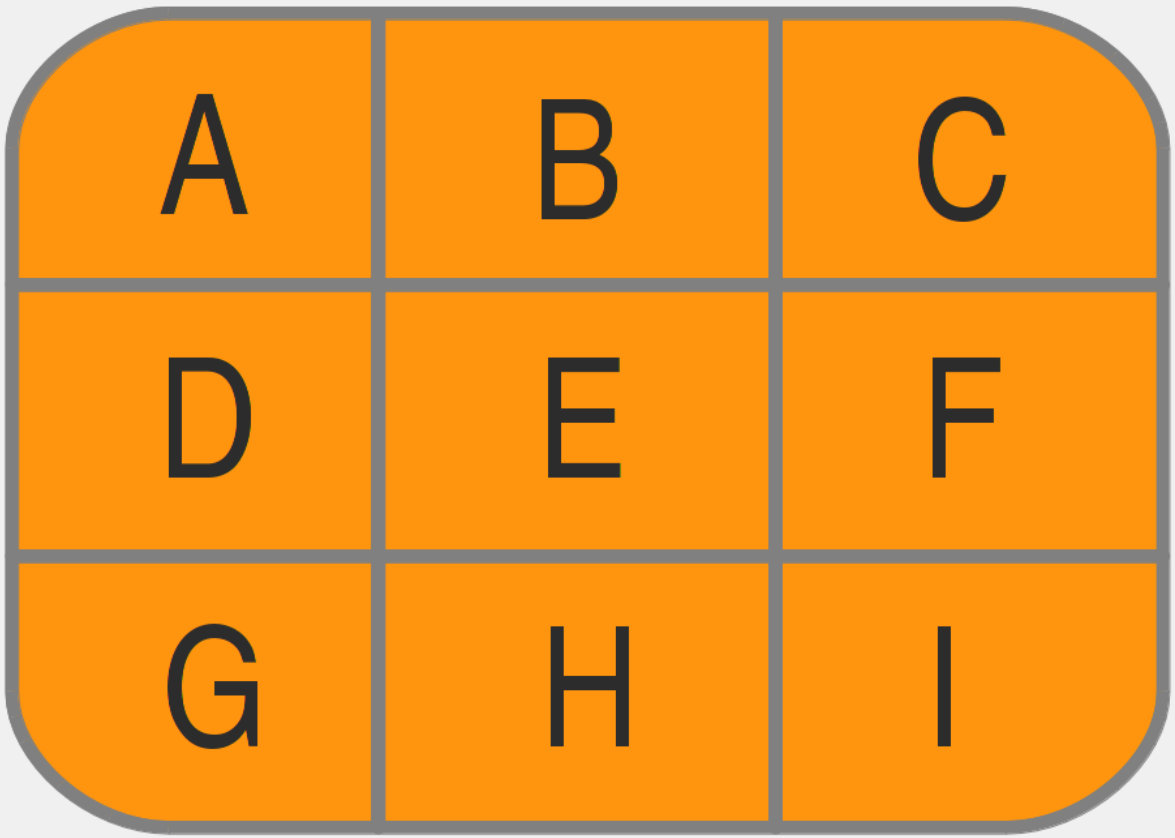
- Essentially different from Hard Drives
- Fast constant access to a random position
- Limited number of writes

SSDs Internally (1) - General View

SSDs Internally (2) - Blocks

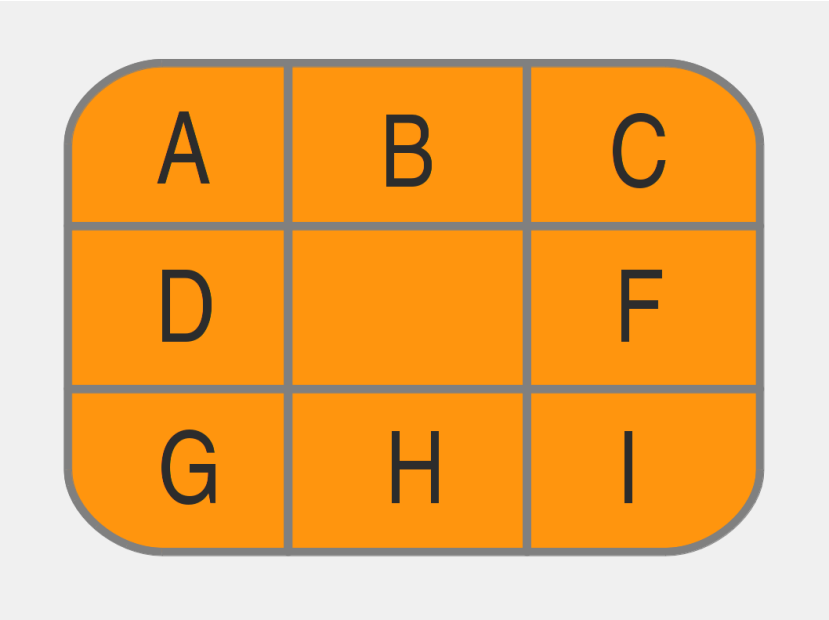


SSDs Internally (3) - Blocks and Pages

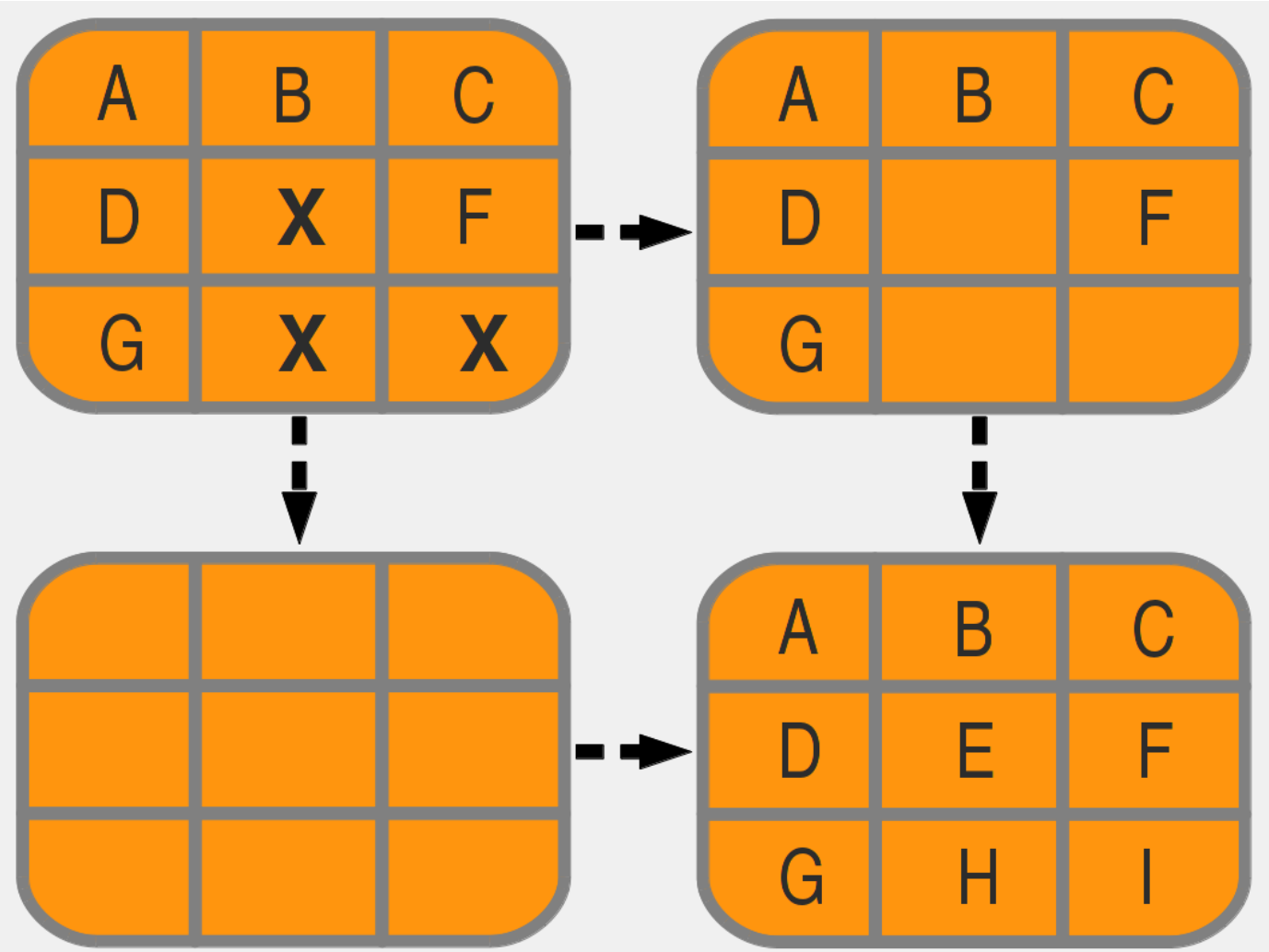


Writing to a block

- Page needs to be in a erased state
- Cannot overwrite data
- Erasing is done at block level



Overwrite - Erase/Program



Wear Leveling and Garbage Collection

- Reusing same block reduces its lifetime
- Erase/Program is expensive
- Move data to better use blocks
- Garbage Collection - Recover invalid pages

1 Introduction

2 SSD principles

3 Indirection in SSDs

4 Nameless Writes

5 Evaluation

6 Conclusion

SSDs need indirection

- Indirection is used to implement wear-leveling
- Absolutely necessary to ensure reasonable lifetime
- Problem: need to store indirection table
- 3 main techniques:
  - Full-page mapping
  - Block mapping
  - Hybrid mapping

Full-page mapping

- Each page can be mapped
- Consider 32-bit pointers per 2KB pages
- With 1TB SSD, 2GB indirection table
- Problem: Great space overhead, DRAM is expensive

Block mapping

- Mapping at block-level (128 pages)
- 32MB indirection table in the same settings
- Smaller memory overhead
- Problem: high garbage collection cost (Gupta et al.)

Hybrid mapping

- Map most data at block level
- Small page-mapped area
- Keeps space overhead low
- Avoids garbage collection overhead
- Problem: garbage collection can still hurt performances
- Problem: very complex FTL (Flash Translation Layer)
- Solution: Nameless Writes

1 Introduction

2 SSD principles

3 Indirection in SSDs

4 Nameless Writes

5 Evaluation

6 Conclusion

Reminder

Idea: the device chooses WHERE to write

- The FS requests the writing of data
- The device performs the write
- Address returned to the FS

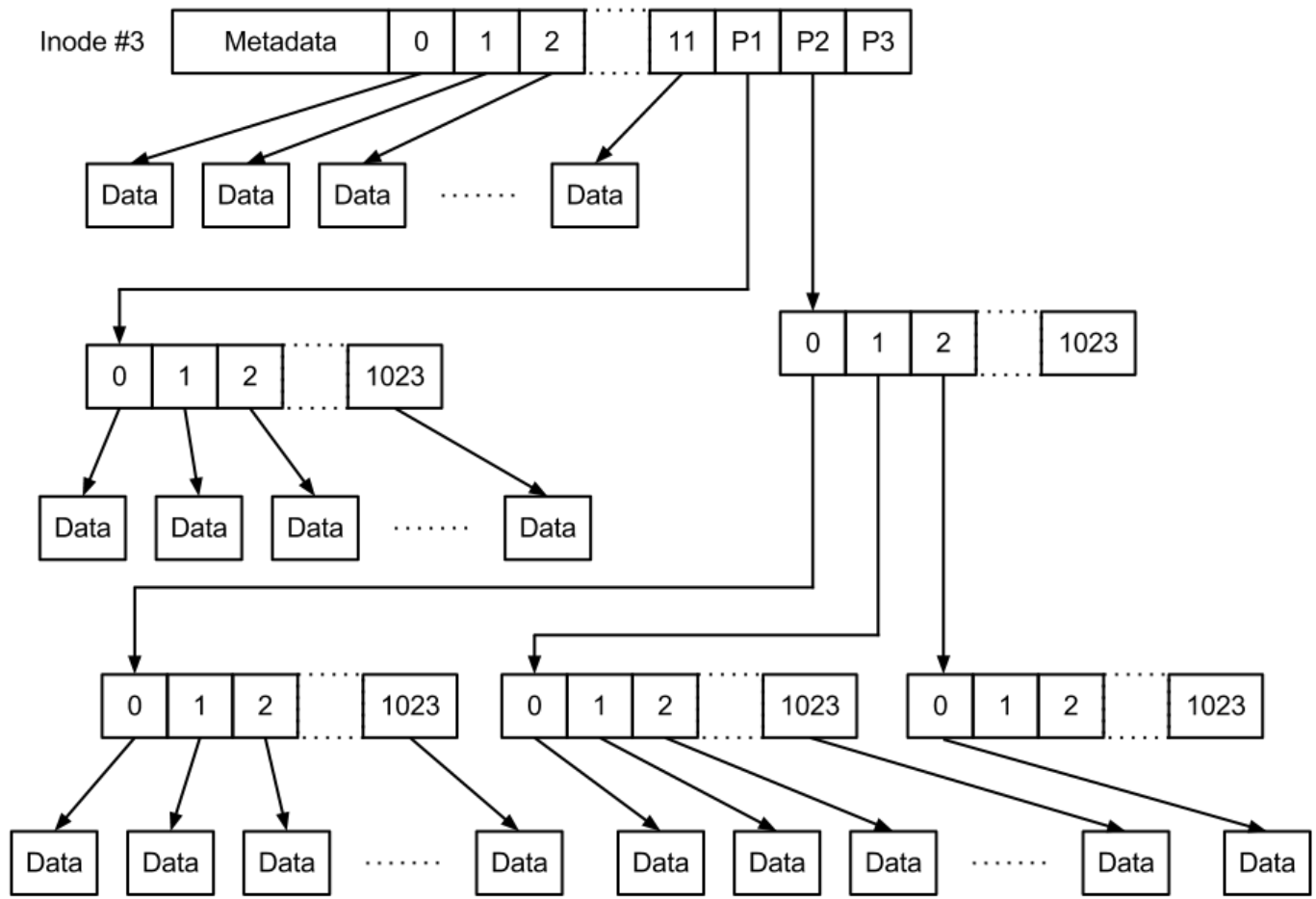
Main interface

```
Nameless_Write(data, len) : phys@
Nameless_Overwrite(phys@, data, len) : new@
Physical_Read(phys@, len) : data
Free(vitr/phys@, len)
```

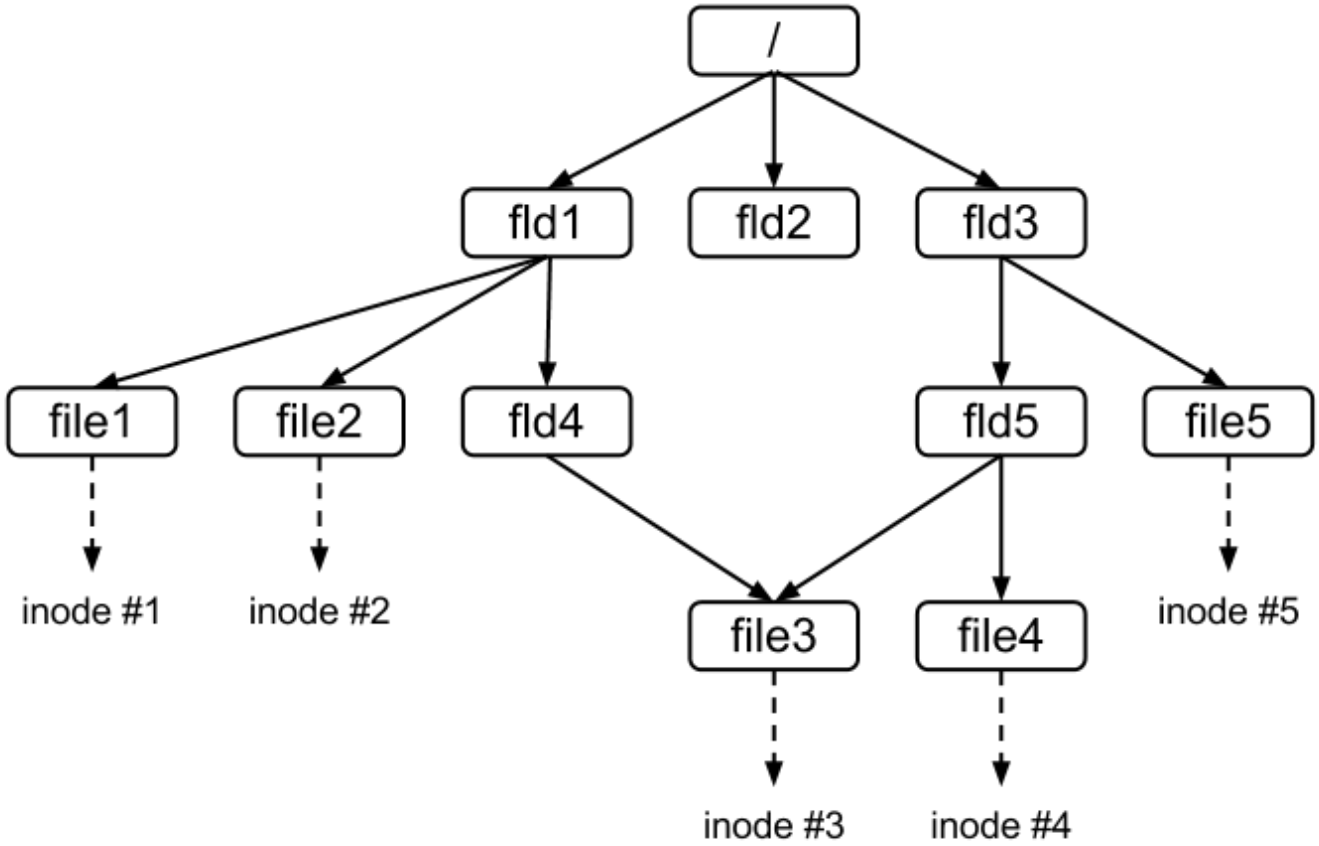
Recursive update problem

- Problem with this interface: recursive update
- File modification imply inode update
- The inode will move (Nameless overwrite)
- Every structure pointing to it will have to be updated
- ...

Inode, and file structure



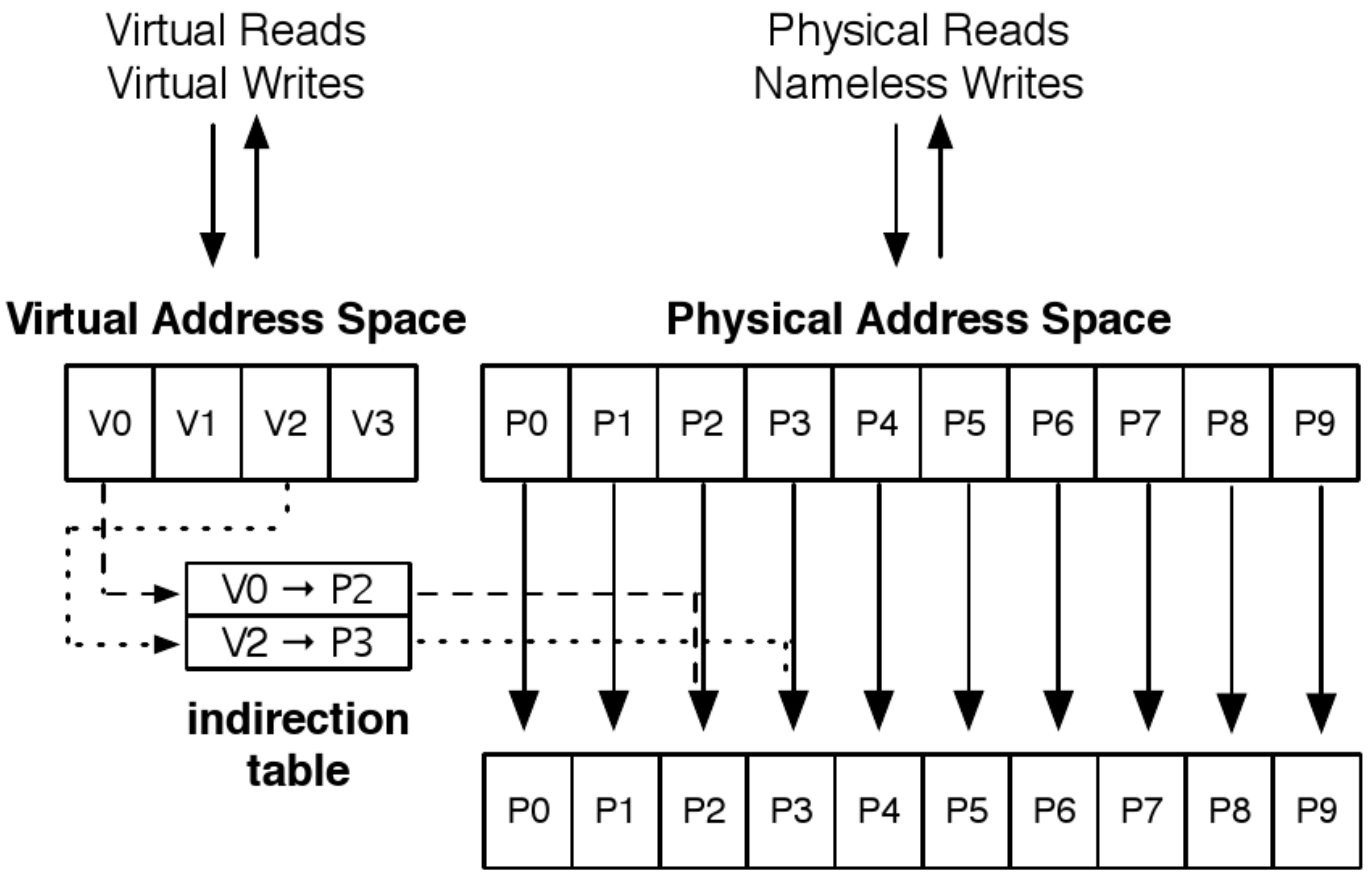
File tree


/
fld1
fld2
fld3
file1
file2
fld4
fld5
file5
inode #1
inode #2
file3
file4
inode #5
inode #3
inode #4

Solution: Segmented address space

- Large physical address space (for nameless writes)
- Small virtual address space (for traditional writes)
- Idea: keep pointer-based structures in virtual space

Segmented address space



Virtual read / write interface

```
Virtual_Write(virt@, data, len)
Virtual_Read(virt@, len) : data
```

Migration callback

- Callback provided for the SSD to notice data migration to the FS
- Useful for it to reclaim blocks (garbage collection)

```
Migration [Callback] (old_phys@, new_phys@)
```

1 Introduction

2 SSD principles

3 Indirection in SSDs

4 Nameless Writes

5 Evaluation

6 Conclusion

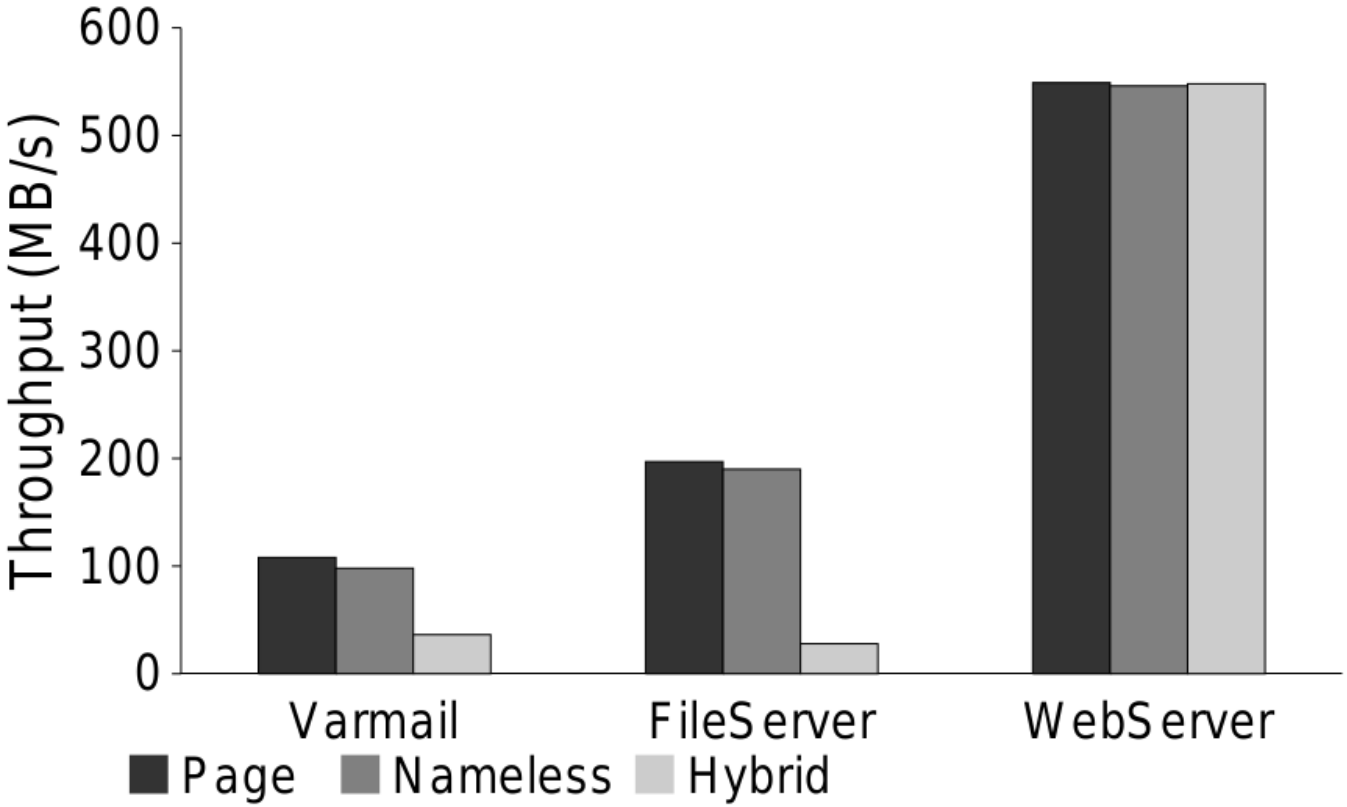
Setup

- Emulator with three different FTLs
  - Page-Level,
  - Hybrid 5% page-level mapping
  - Nameless Writes

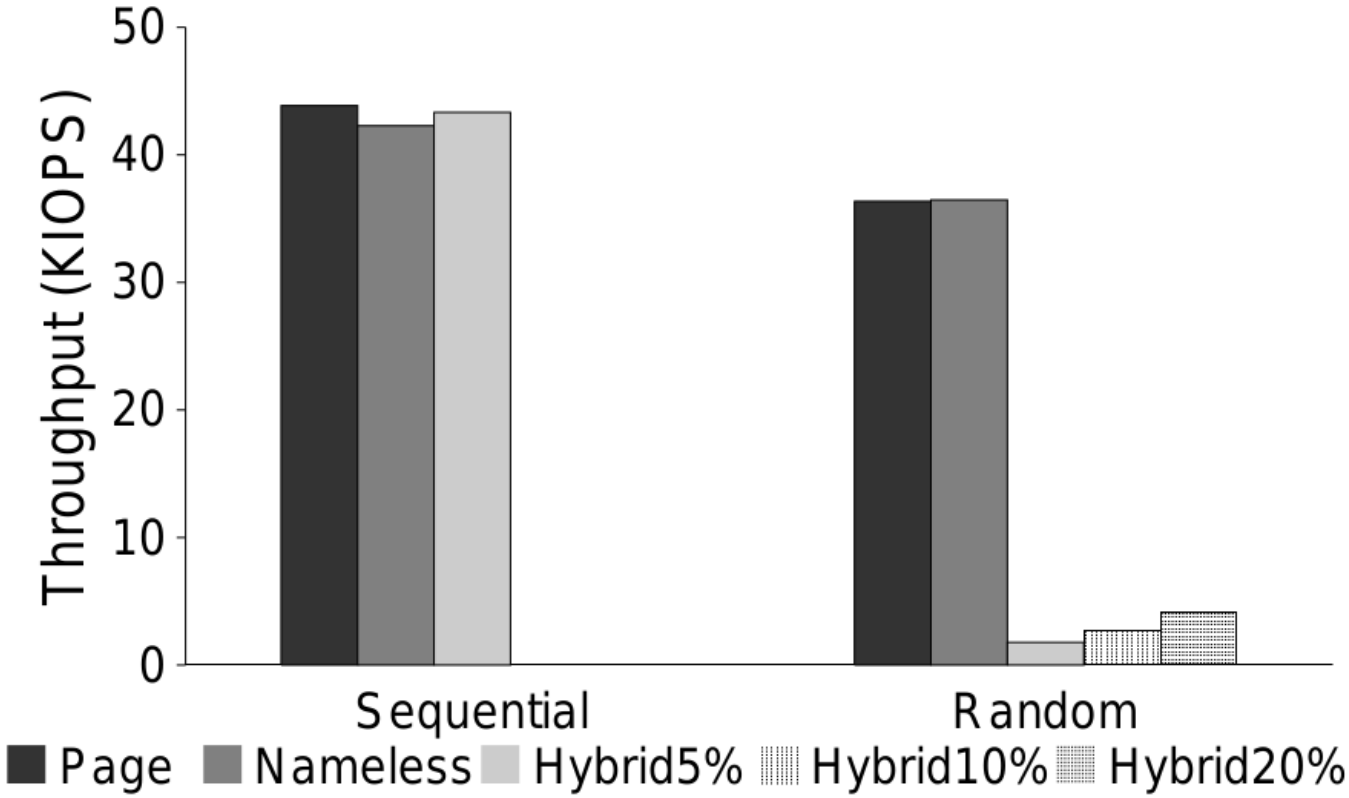
Memory Consumption

Image Size	Page	Hybrid	Nameless
328 MB	328 KB	38 KB	2.7 KB
2 GB	2 MB	235 KB	12 KB
10 GB	10 MB	1.1 MB	31 KB
100 GB	100 MB	11 MB	251 KB
400 GB	400 MB	46 MB	1 MB
1 TB	1 GB	118 MB	2.2 MB

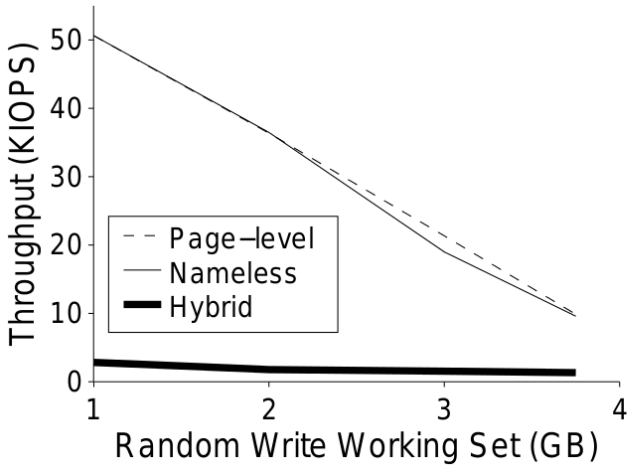
Performance

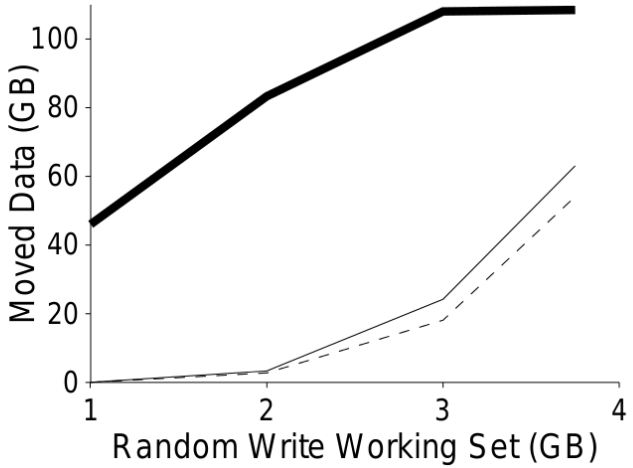


Performance



Performance





1 Introduction

2 SSD principles

3 Indirection in SSDs

4 Nameless Writes

5 Evaluation

6 Conclusion

Conclusion

- New write interface
- Improves random-write performance
- Reduce space costs
- Future work: Port to other file systems

Questions

- Questions?