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 18, 2012

Outline

What are Nameless Writes?

- New device interface for SSDs
- 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

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

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

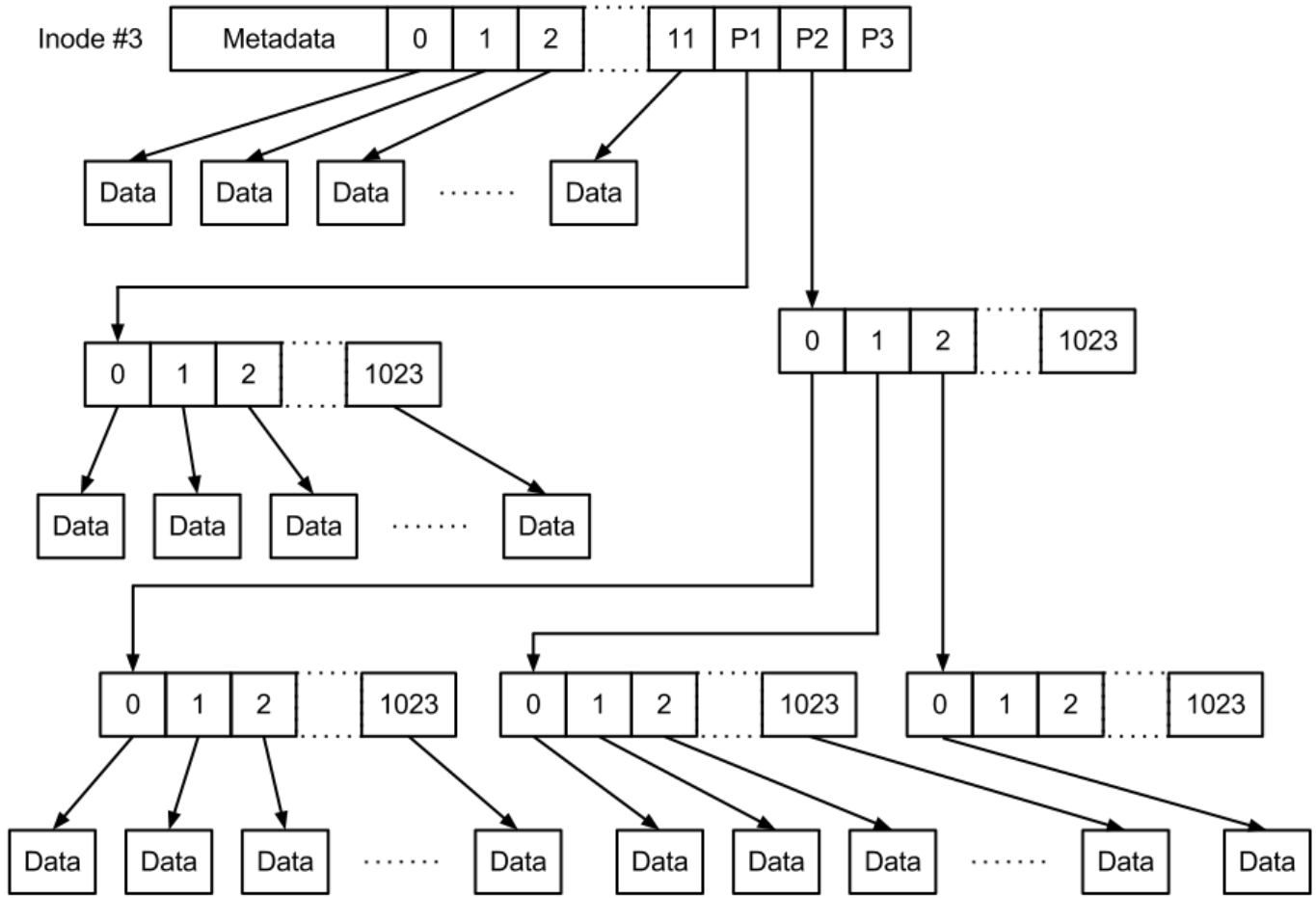
Main interface

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

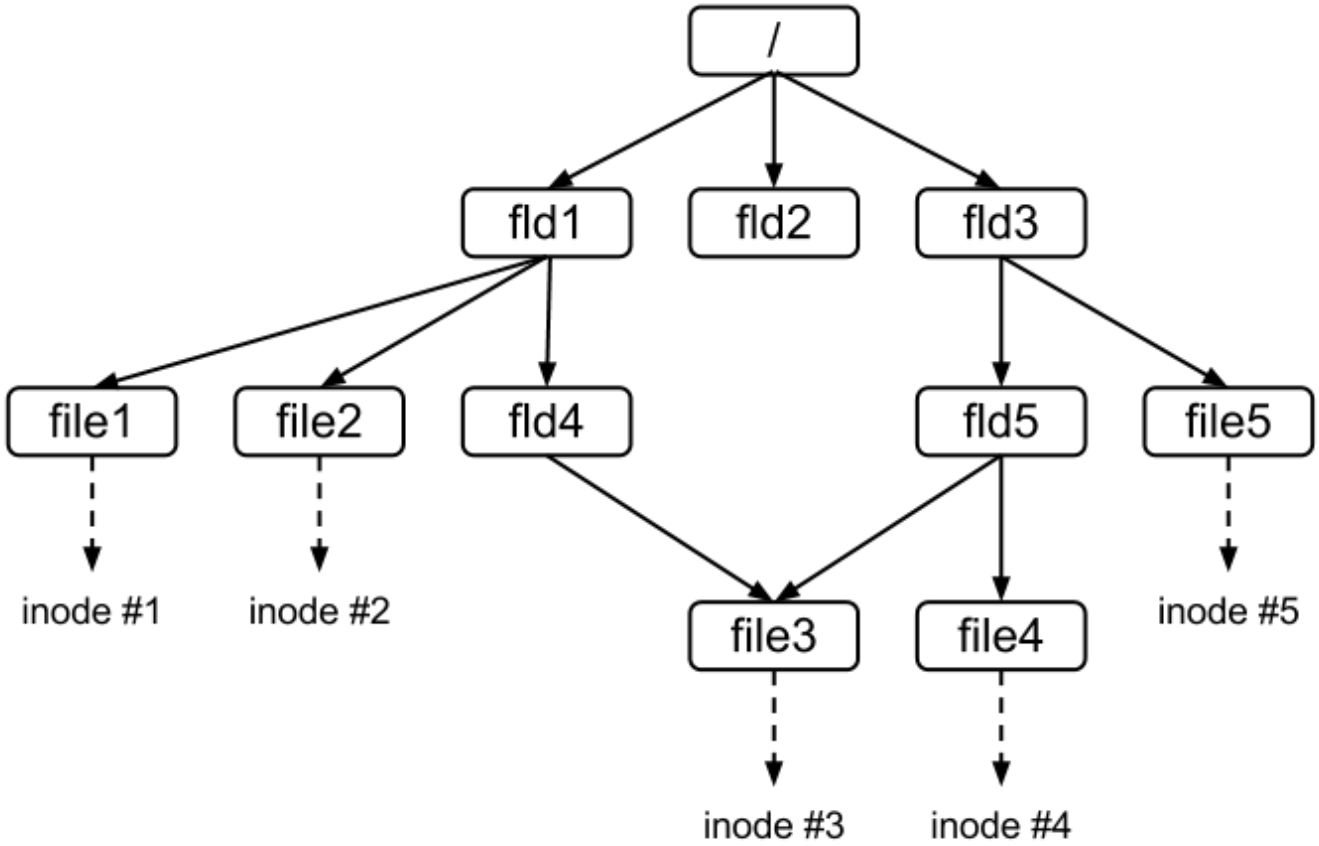
Recursive update problem

- 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



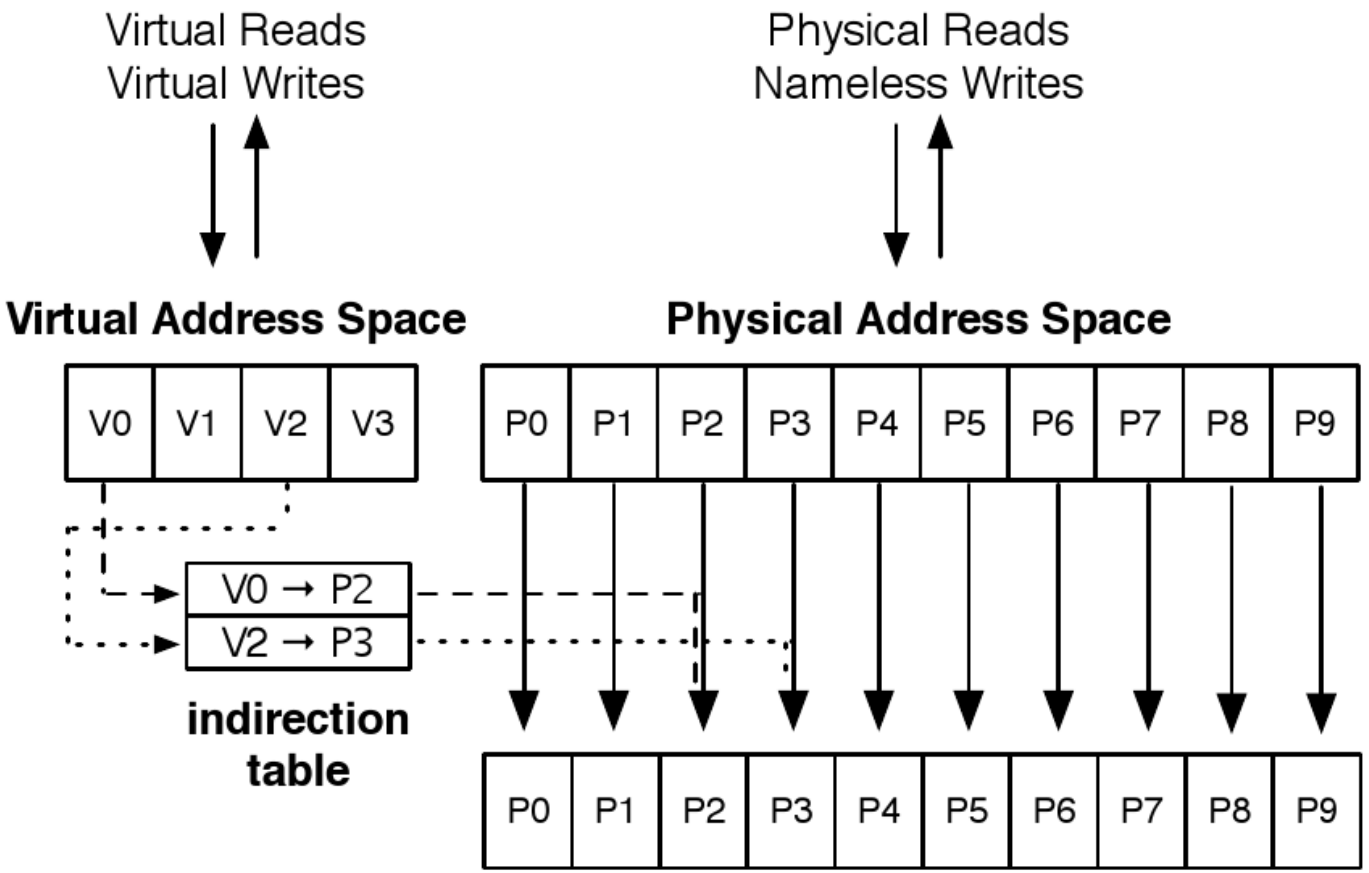
File tree


/
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@)
```