De-indirection for Flash-based SSDs with Nameless Writes

Yiying Zhang, Leo Prasath Arulraj,
Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau

University of Wisconsin - Madison

All problems in computer science can be solved by another level of indirection*

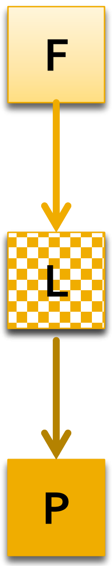
- Indirection
  - Reference an object with a different name
  - Flexible, simple, and modular

- Indirection in computer systems
  - Virtual memory: virtual to physical memory address
  - Hard disks: bad sectors to nearby locations
  - RAID arrays: logical to array physical address
  - SSDs: logical to SSD physical address

* Usually attributed to Butler Lampson

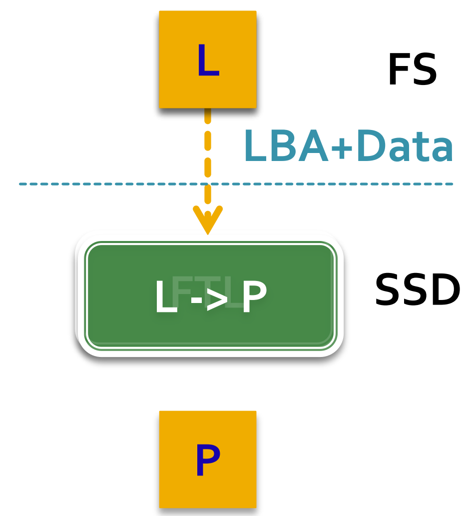
Indirection: Too Much of a Good Thing?

- Excess indirection
  - Redundant levels of indirection in a system
  - e.g. OS on top of hypervisor(s)
  - e.g. File system on top of RAID
- Are all indirections really necessary?
  - Some indirection can be removed
  - Space and performance cost
- What about flash-based SSDs?
  - File system: file offset to logical address (F -> L)
  - Device: logical address to physical address (L -> P)



Indirection in Flash-Based SSDs

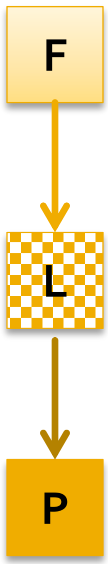
- Indirection in SSDs (L->P)
  - Mapping from logical to physical address
  - Hides erase-before-write and wear leveling
  - Implemented in Flash Translation Layer (FTL)



- Cost of indirection
  - RAM space to maintain indirection table
  - Hybrid: small page-mapped area + big block-mapped area
  - Performance cost of garbage collection
  - Performance impact on random writes [Kim '12]

De-indirection with Nameless Writes

- Solution: De-indirection
  - Remove indirection in SSDs (L->P)
  - Store physical addresses directly in file system (F->P)
- New interface: *Nameless Write*
  - Write without a name (logical address)
  - Device allocates and returns physical address
  - File system stores physical address
- Advantages
  - Reduces space and performance cost of indirection
  - Device maintains critical controls



Summary of Results

- Designed nameless writing interfaces

- Implemented a nameless-writing system
  - Built a nameless-writing SSD emulator
  - Ported ext3 to nameless writes

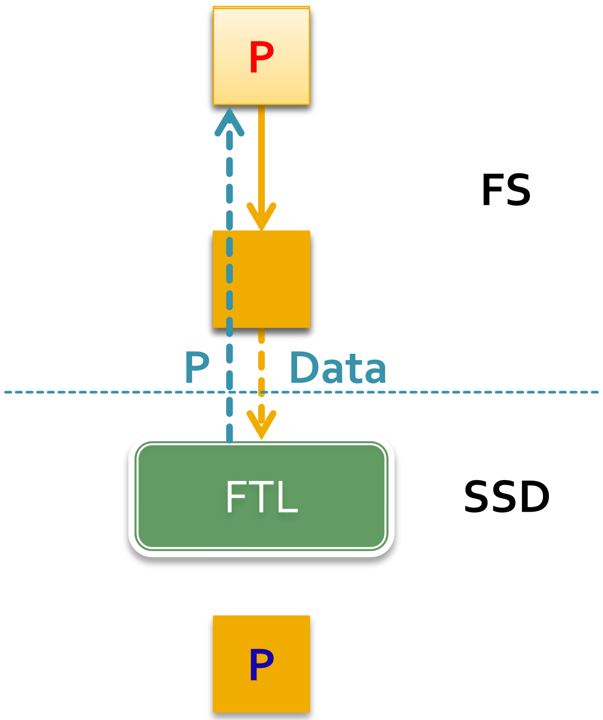
- Evaluation results
  - Evaluated against two other FTLs
  - Small indirection table, ~20x reduction over traditional SSDs
  - Better random write throughput, ~20x over traditional SSDs

Outline

- Introduction
- Nameless write interfaces
  - Basic interfaces
  - Problems of basic interfaces and solutions
- Nameless-writing device and ext3
- Results
- Conclusion

Basic Nameless Write Interfaces

- Nameless Write
  - Writes only data and no name
- Physical Read
  - Reads using physical address
- Free/Trim
  - Invalidates block at physical address



P: Addr the structure points to

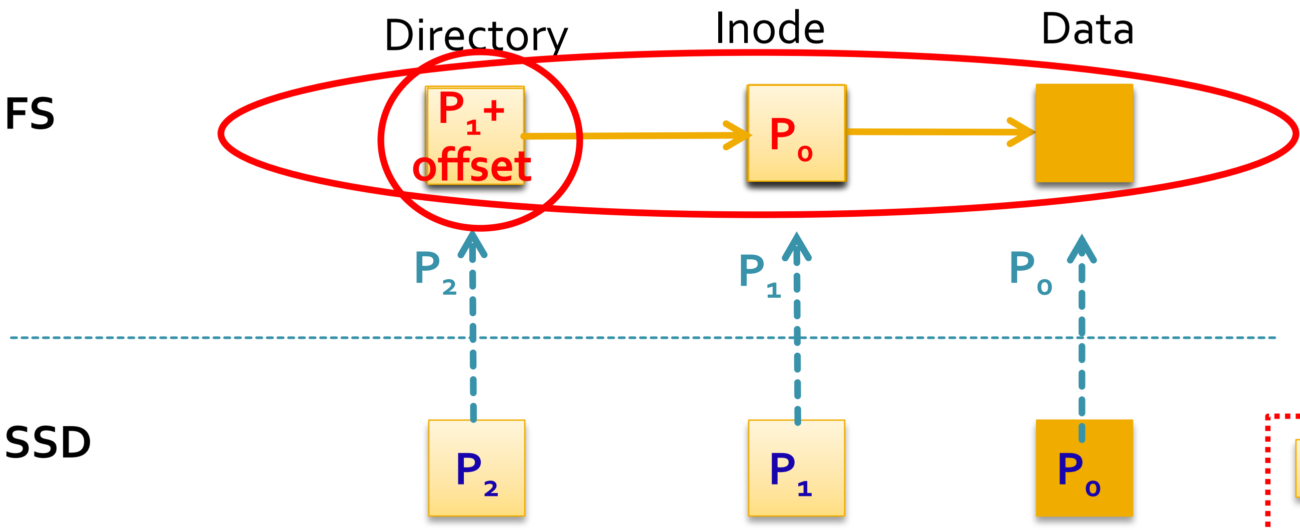
P: Addr of the block

Problems of Basic NW Interfaces

- P1: Cost of straw-man nameless-write approach
  - How to reduce the overheads of complete de-indirection?
- P2: Migration during wear leveling
  - How to reflect physical address change in the file system?
- P3: Locating metadata structures
  - How to find metadata structures efficiently?

P1: Nameless Write Straw-man

- Overwrite a data block in a file in ext3



Metadata

Data

Red: Addr the structure points to

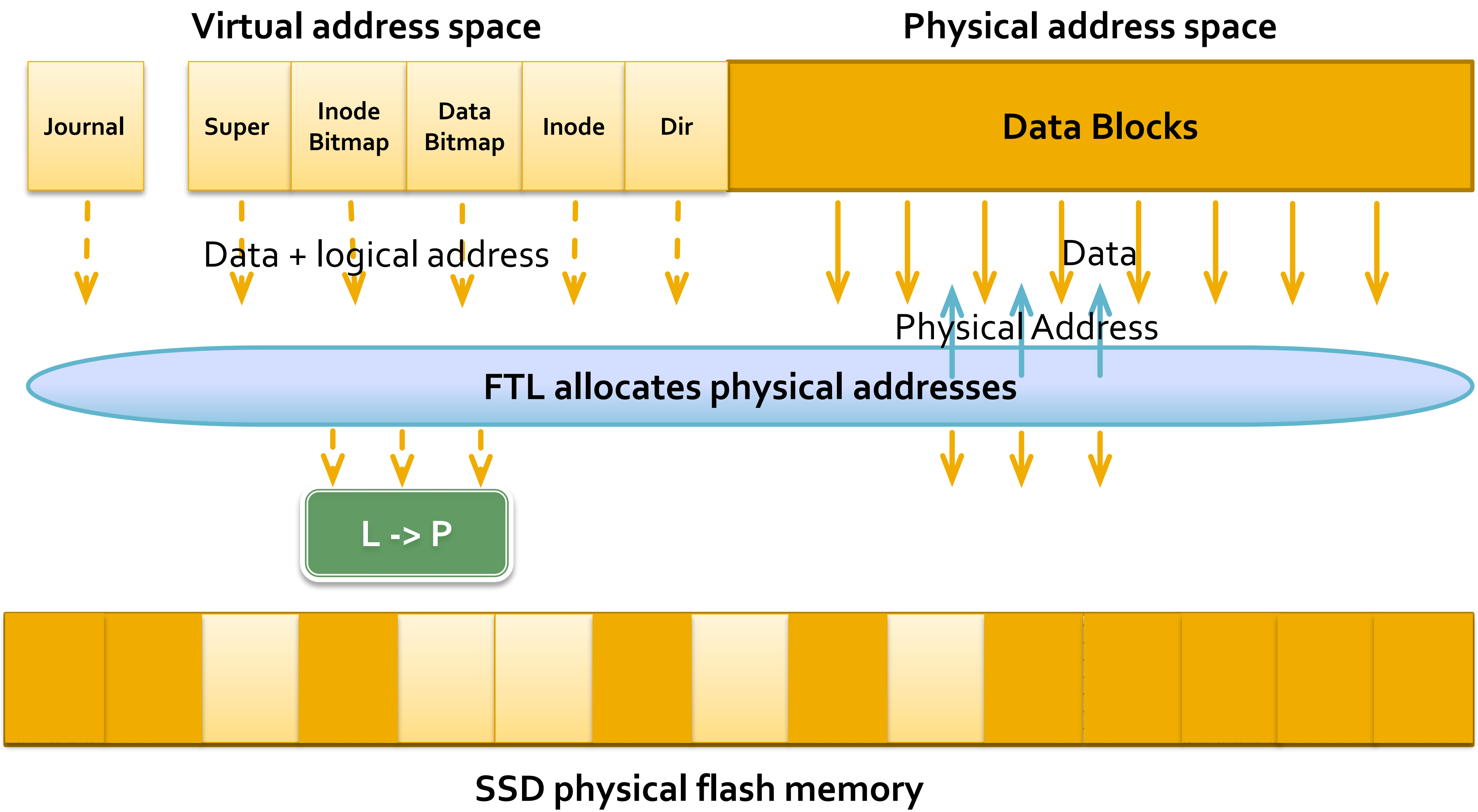
Blue: Addr of the block

- Problems
  - Overhead of updating along FS tree
  - FS more complex and less flexible

P1 Solution: Segmented Address Space

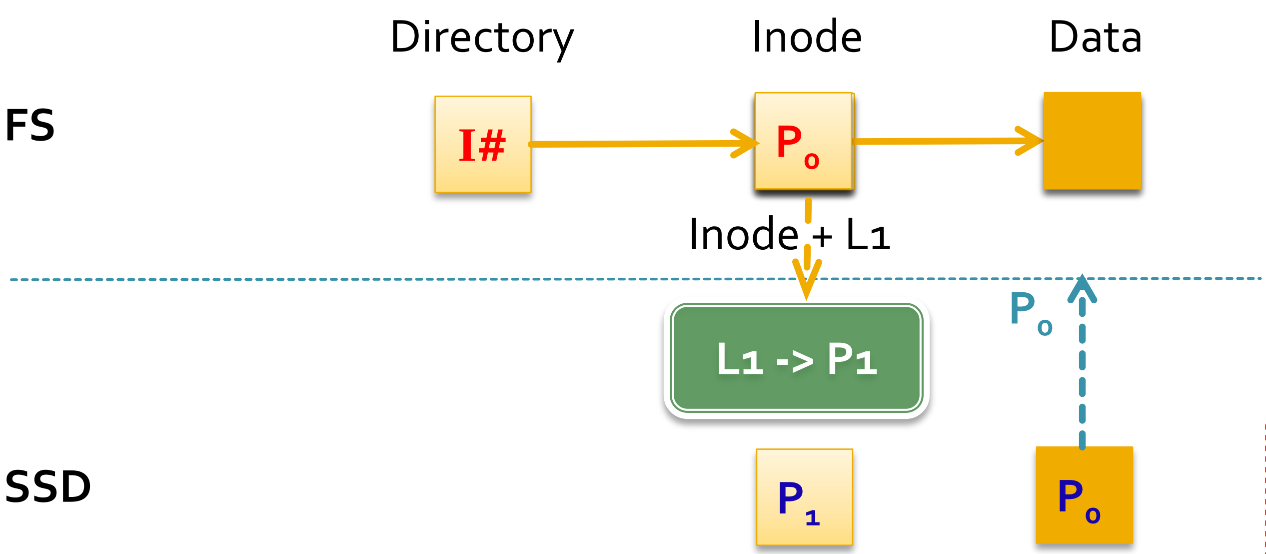
- Problem of recursive updates
  - Writes propagate to reflect physical addresses
- Solution: Two segments of address space
  - Stop recursive updates
- Physical address space
  - Nameless write, physical read
  - Contains data blocks
- Virtual address space
  - Traditional (virtual) read/write
  - Small indirection table in device
  - Contains metadata blocks (typically small metadata [Agrawal'07])

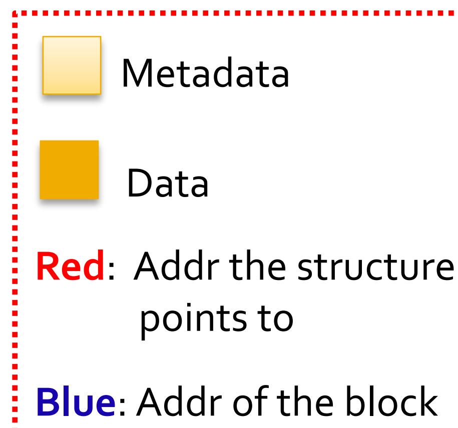
P1 Solution: Segmented Address Space Example



P1 Solution: Nameless Write with Segmented Address Space

- Overwrite a data block with segmented address space

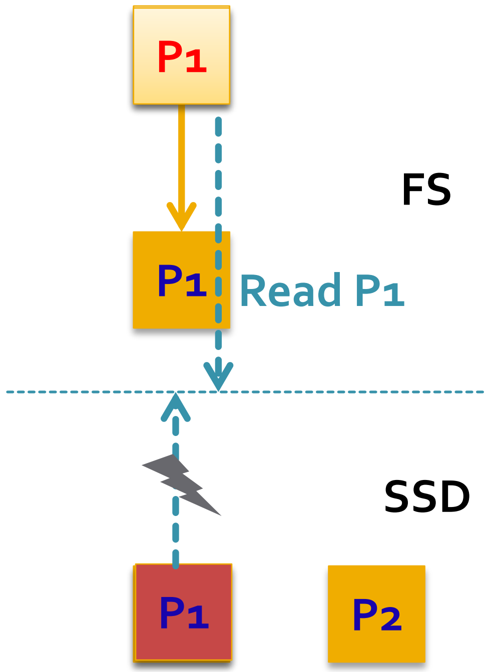




- Advantages
  - One level of update propagation
  - Simple implementation

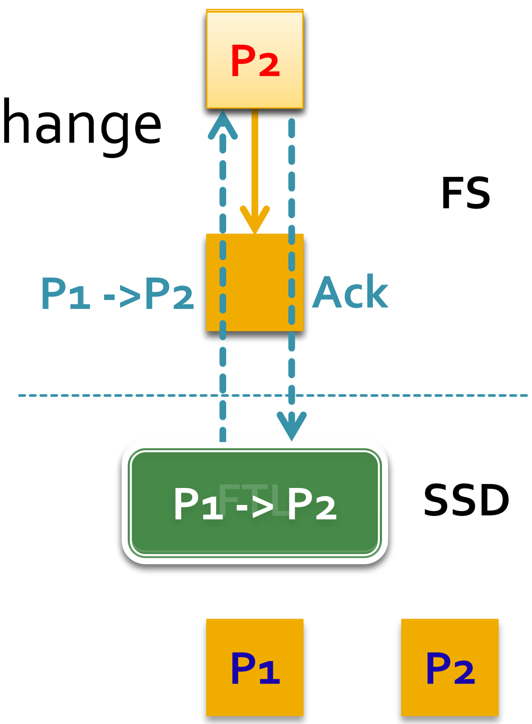
P2: Migration During Wear Leveling

- Block wear in SSDs
  - Uneven wear among blocks with data of different access frequency
- Wear leveling
  - SSD moves data to distribute block erases evenly
- Physical address change
  - File system needs to be informed
  - Only address change in the physical space



P2 Solution: Migration Callbacks

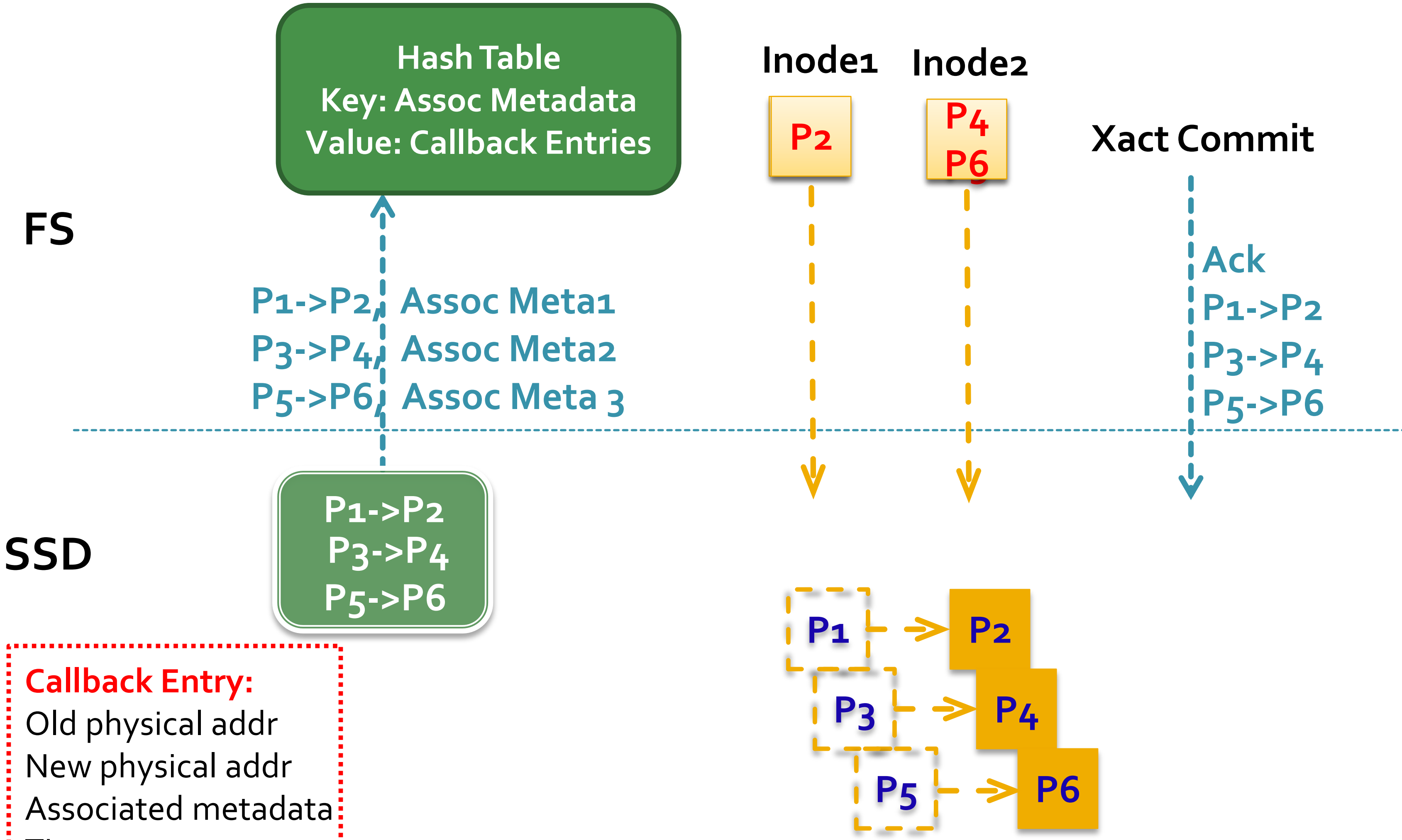
- New interface: *Migration Callbacks*
  - Device informs FS about physical address change
- Temporary remapping table
- Reads and overwrites to old address
  - Remapped to new address
- FS processes callbacks in background
  - Acknowledges device when metadata updated



P3: Associated Metadata

- *Problem*: Locating metadata structures
  - e.g. During callbacks
  - e.g. During recovery
  - Naive approach: traversing all metadata
- *Solution*: Associated Metadata
  - Small amount of metadata used to locate metadata
  - e.g. Inode number, inode generation number, block offset
  - Sent with nameless writes and migration callbacks
  - Stored adjacent to data pages on device, e.g. OOB area

P2 and P3 Implementation in Ext3



Callback Entry:
Old physical addr
New physical addr
Associated metadata
Timestamp

Outline

- Introduction
- Nameless write interfaces
- Nameless-writing device and ext3
- Results
- Conclusion

Nameless-Writing Device

- Supports nameless write interfaces
- Flexible device allocation
- Maintains small mapping table
  - Indirection of the virtual address space
  - Temporary remapping table for callbacks
- Control of garbage collection and wear leveling
  - Minimize physical address migration (In-place GC)

Porting Ext3 to Nameless Writes

- Ext3: Journaling file system extending ext2
- Ordered journal mode
  - Metadata always written after data
  - Fits well with nameless writes
- Interface support
  - Segmented address space
  - Nameless write
  - Physical read
  - Free/trim
  - Callback

Total Lines of Code

- Total: 4360
- Ext3: 1530
- JBD: 480
- Generic I/O: 2020
- Headers: 340

Outline

- Introduction
- Nameless write interfaces
- Nameless-writing device and ext3
- Results
- Conclusion

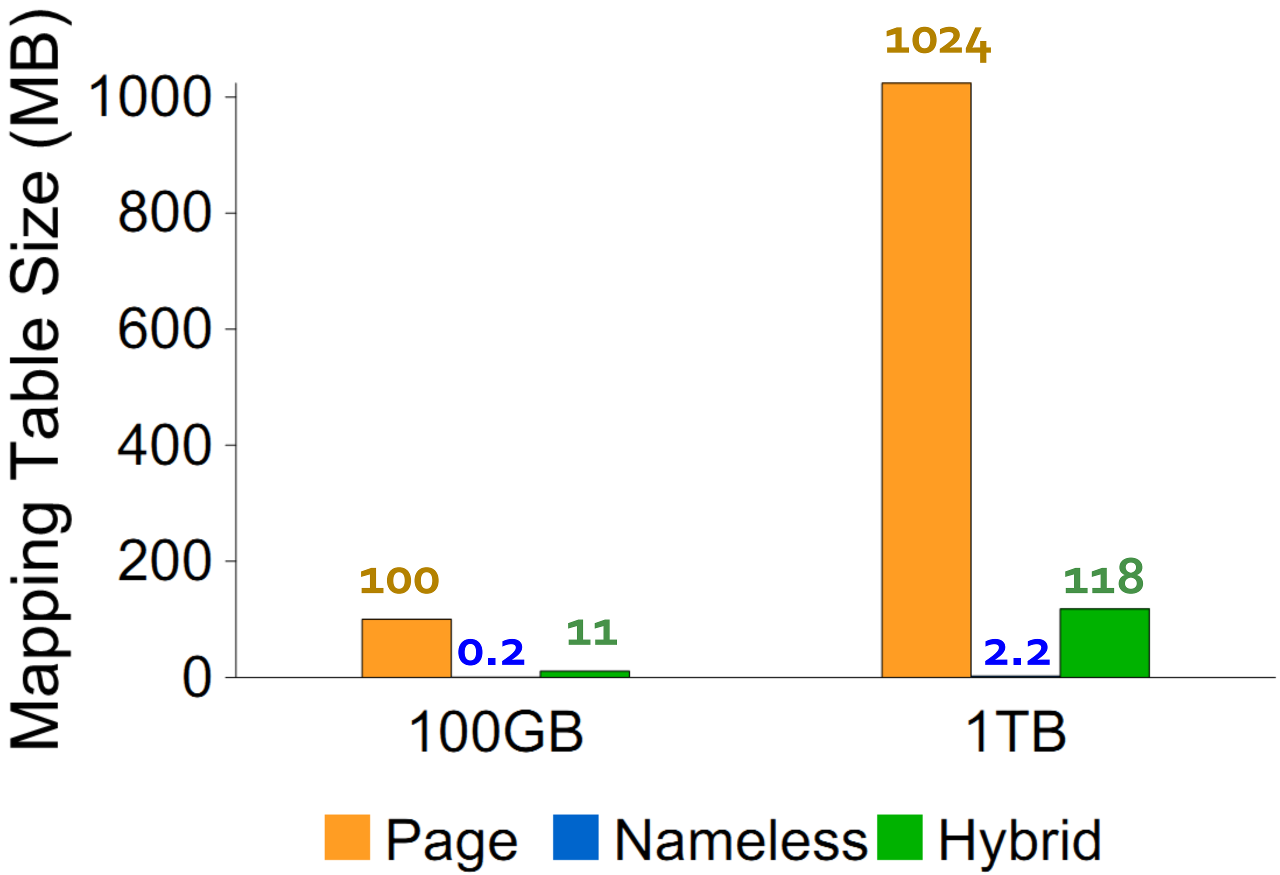
Evaluation Methodology

- SSD emulator
  - Linux pseudo block device
  - Data stored in memory

- FTLs studied
  - *Page* mapping: log-structured allocation
    ideal in performance, unrealistic in indirection space
  - *Hybrid* mapping: small page-mapped area + block-mapped area
    models real SSDs, realistic in indirection space
  - *Nameless-writing*

Indirection Table Space Cost

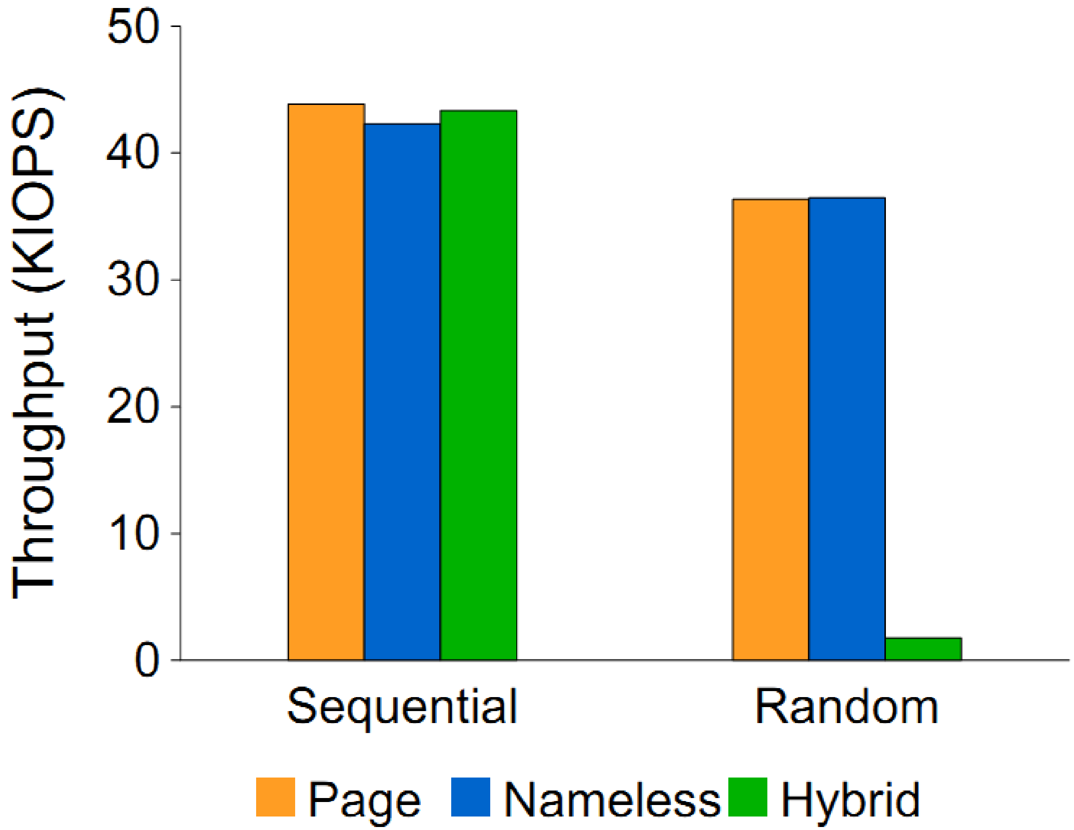
- Mapping table sizes for typical file system images [Agrawal'09]



Nameless writes use 2% - 7% mapping table space of traditional hybrid SSDs

Micro-benchmark Performance

- Sequential and sustained 4KB random write

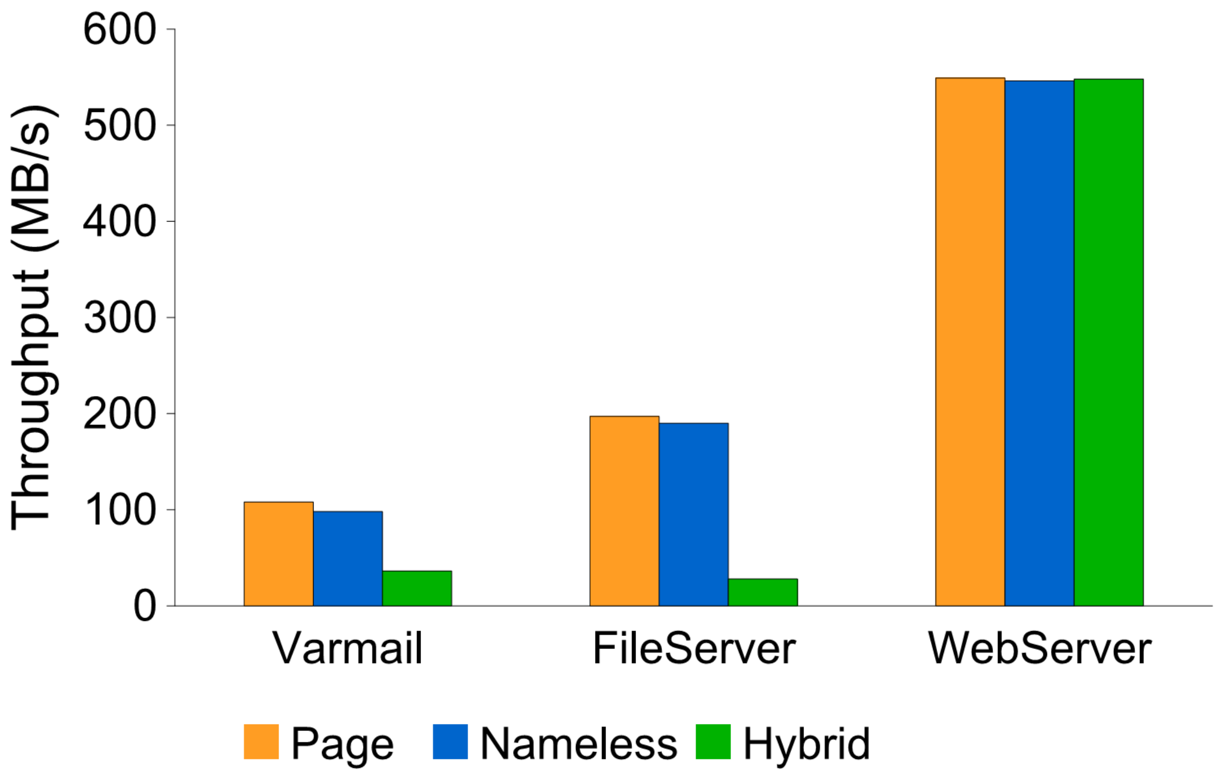


Nameless writes deliver 20x random write throughput over traditional hybrid SSDs

Performance of nameless writes is close to page FTL (upper-bound)

Macro-benchmark Performance

- Varmail, FileServer, and WebServer from Filebench



Similar performance when workload is read or sequential-write intensive

Performance of hybrid FTL is worse than the other two FTLs when workload has random writes

Outline

- Introduction
- Nameless write interfaces
- Nameless-writing device and ext3
- Results
- Conclusion

Summary

- *Problem*: Excess indirection in flash-based SSDs
- *Solution*: De-indirection with *Nameless Writes*
- *Implementation* of a nameless-writing system
  - Built an emulated nameless-writing SSD
  - Ported ext3 to nameless writes
- *Advantages* of nameless writes
  - Reduce the space cost of indirection over traditional SSDs
  - Improve random write performance over traditional SSDs
  - Reduce energy cost, simplify SSD firmware

Indirection: Reprise

- *"All problems in computer science can be solved by another level of indirection"*
  - *Usually attributed to Butler Lampson*
  - *Lampson attributes it to David Wheeler*

- *And Wheeler usually added:*

"but that usually will create another problem"

Indirection Conclusion

- Too much: Excess indirection
  - e.g. file offset => logical address => physical address
- Partial indirection
  - e.g. nameless writes with segmented address space
- Too little: Cost of (complete) de-indirection
  - e.g. overheads of recursive update

Thank you !

Questions ?

The ADvanced Systems Laboratory (ADSL)
http://www.cs.wisc.edu/adsl/