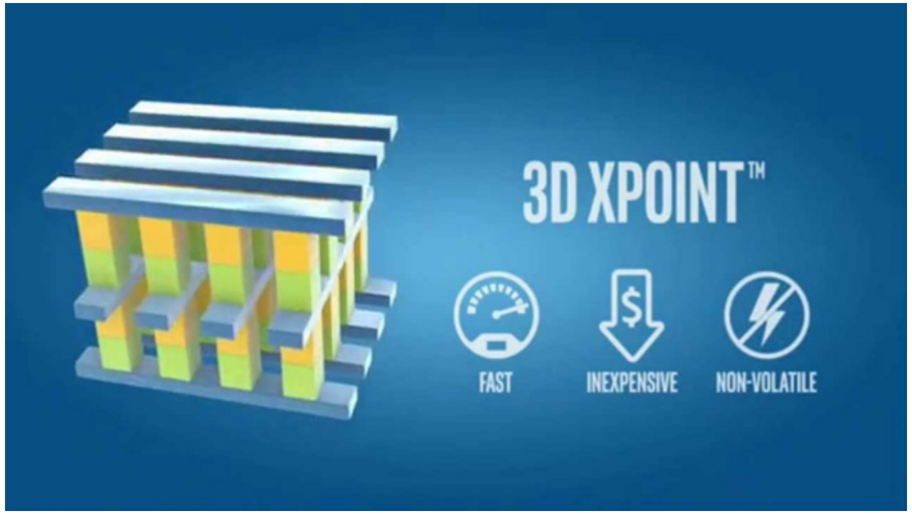
Soft Updates Made Simple and Fast on Non-volatile Memory

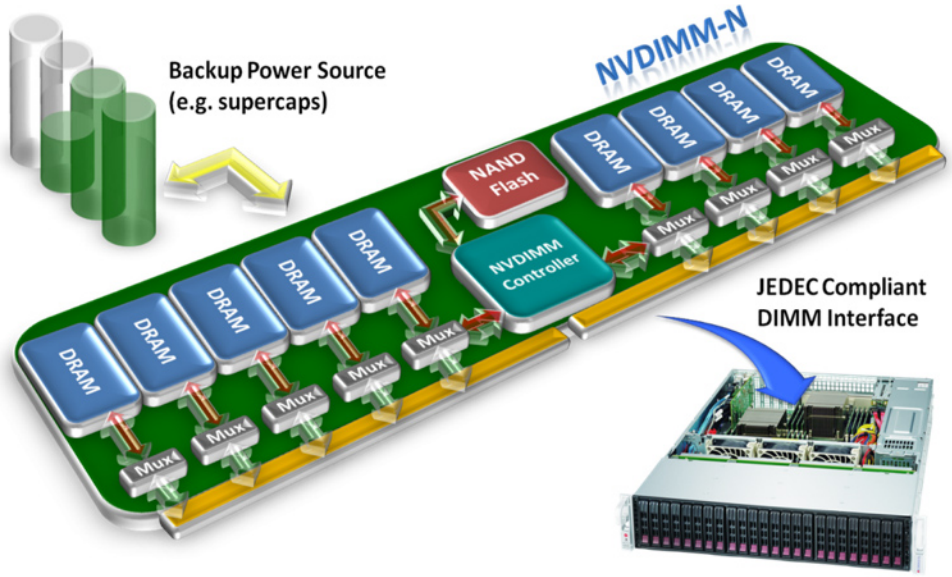
Mingkai Dong, Haibo Chen
Institute of Parallel and Distributed Systems,
Shanghai Jiao Tong University

@ ATC '17

Non-volatile Memory (NVM)

- ✓ Non-volatile
- ✓ Byte-addressable
- ✓ High throughput and low latency





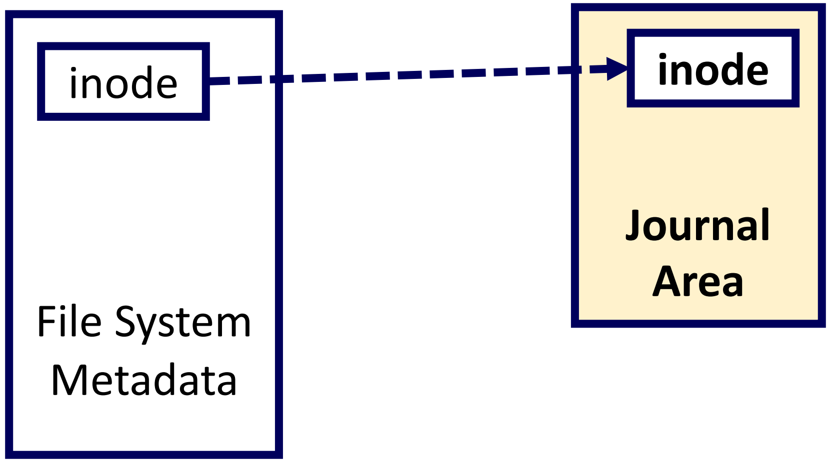
NVM File Systems (NVMFS)

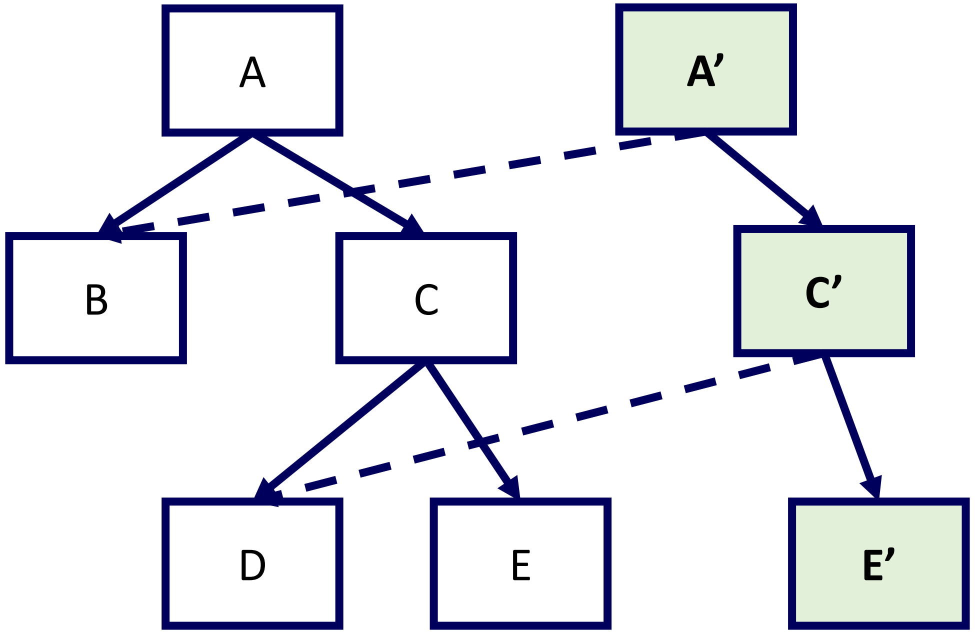
Existing NVMFS use **journaling** or **copy-on-write** for crash consistency

Synchronous cache flushes are necessary

Cache flushes are expensive!

Other options for crash consistency?



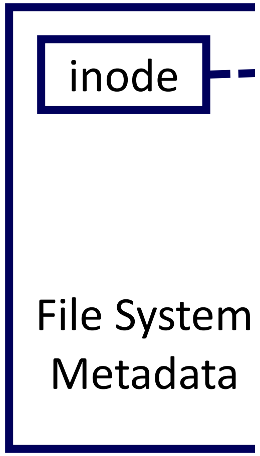


Existing NV ... onsistency

Synchrono

Cache flusl

Other opti



USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

The following paper was originally published in the

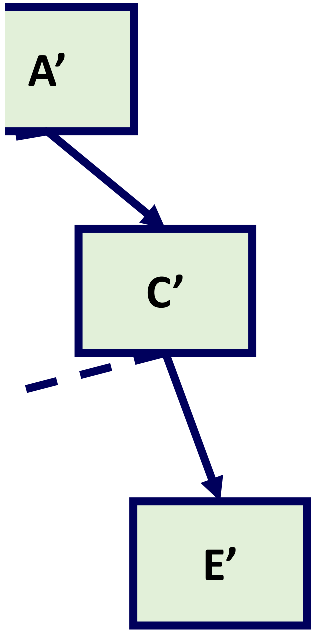
Proceedings of the FREENIX Track:
1999 USENIX Annual Technical Conference

Monterey, California, USA, June 6–11, 1999

Soft Updates: A Technique for Eliminating Most Synchronous Writes in the Fast Filesystem

Marshall Kirk McKusick
Author and Consultant

Gregory R. Ganger
Carnegie Mellon University



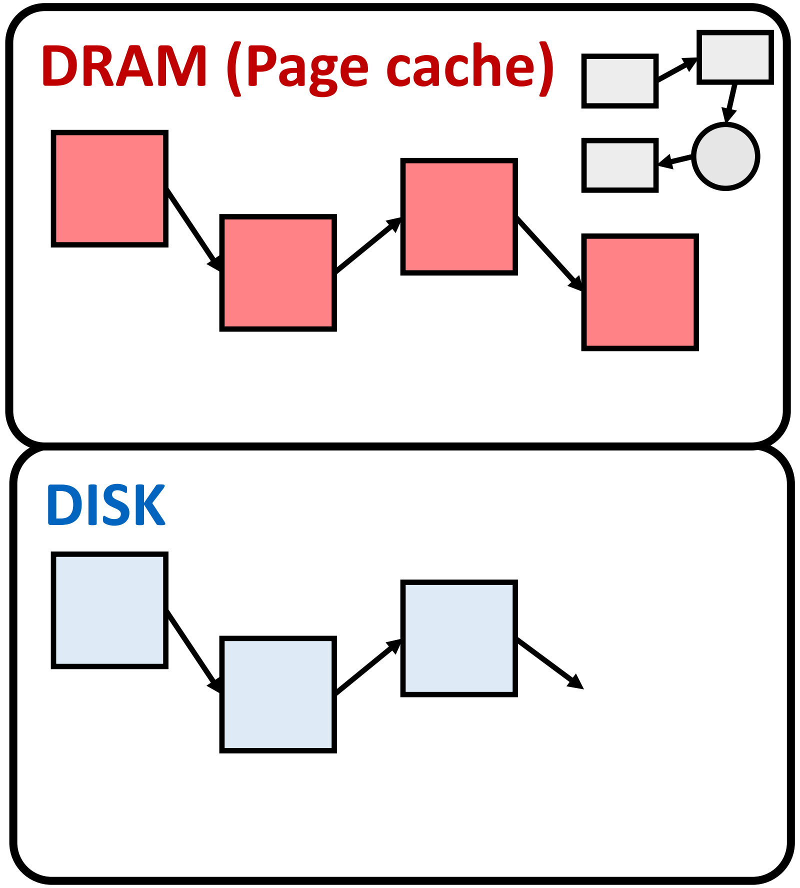
Soft Updates

Latest metadata in DRAM

- Updated in DRAM with dependency tracked
- ✓ DRAM performance
- ✓ No synchronous disk writes

Consistent metadata in disks

- Persisted to disks with dependency enforced
- ✓ Always consistent
- ✓ Immediately usable after crash

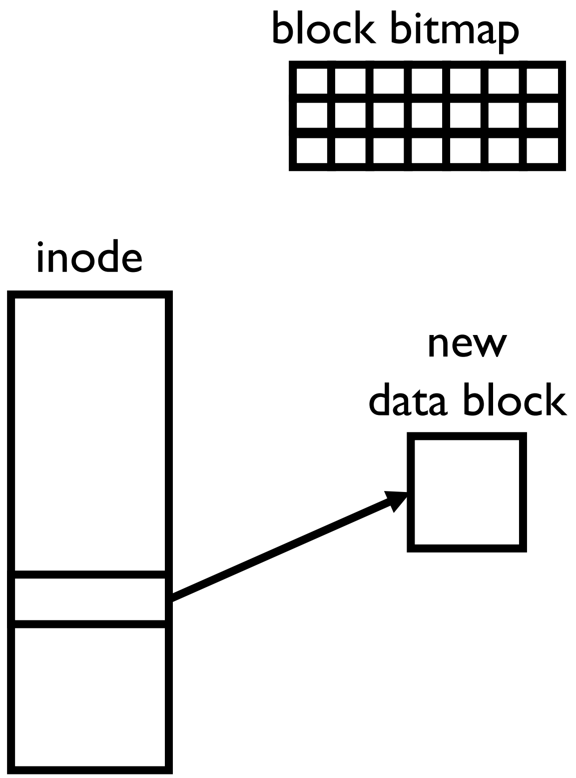


Traditional Soft Updates

Soft Updates

Update dependencies

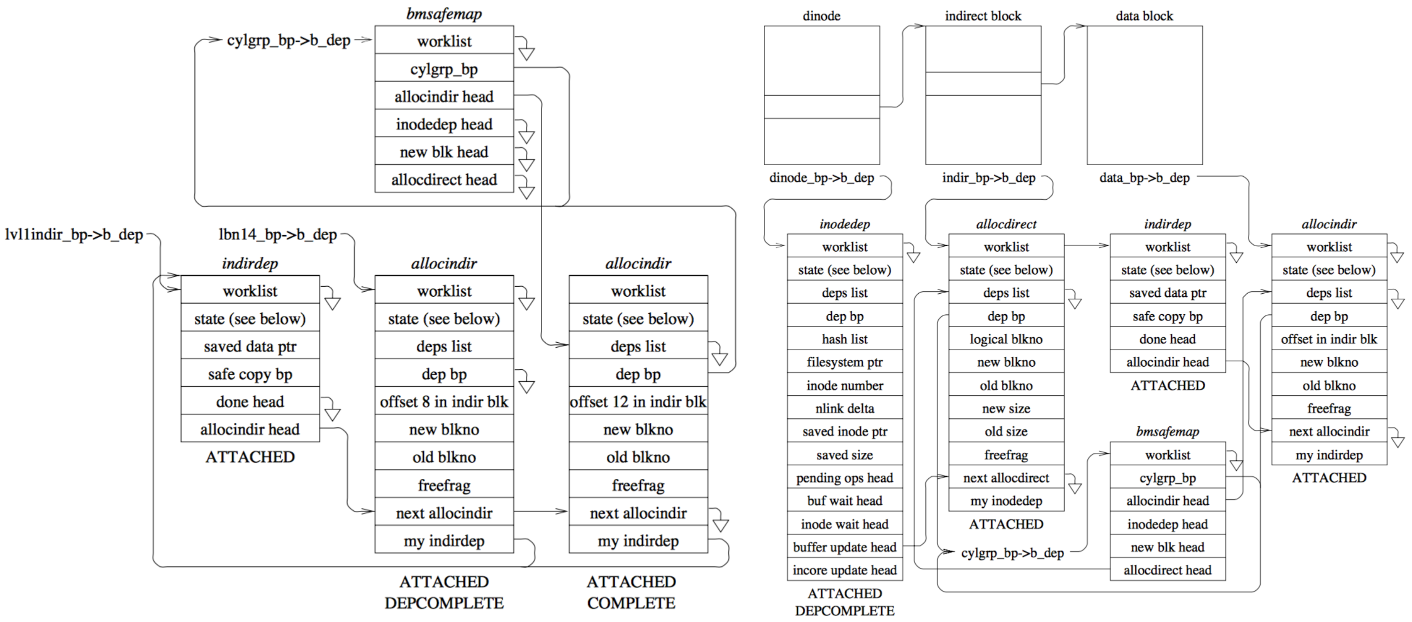
- E.g., allocating a new data block
  1. Allocate in bitmap
  2. Fill data in the block
  3. Update pointer to the block

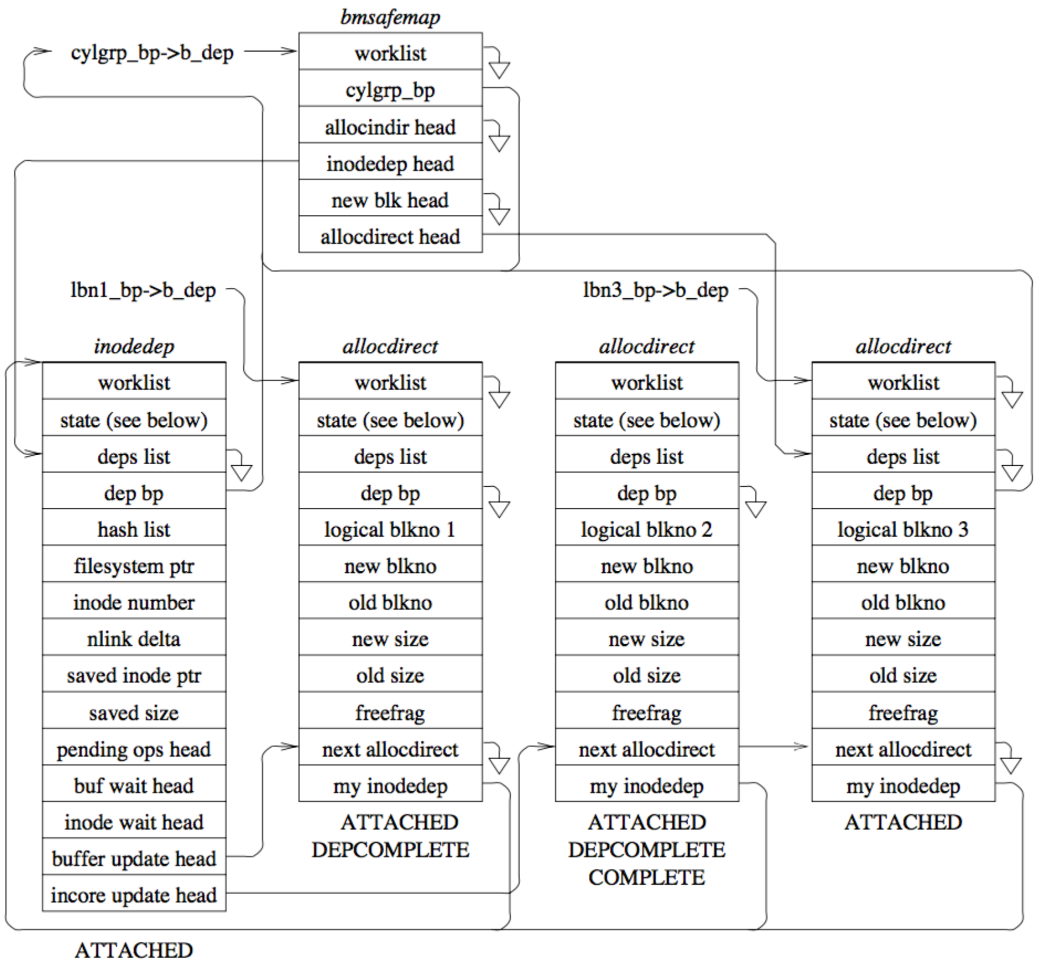


Soft Updates Is Complicated

Delayed disk writes

- Auxiliary structures for each update
- More complex dependencies





Figures from *Soft Updates: A Technique for Eliminating Most Synchronous Writes in the Fast Filesystem,* ATC '99

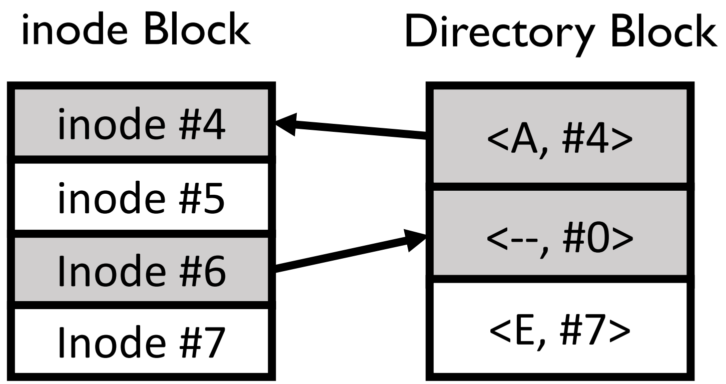
Soft Updates Is Complicated

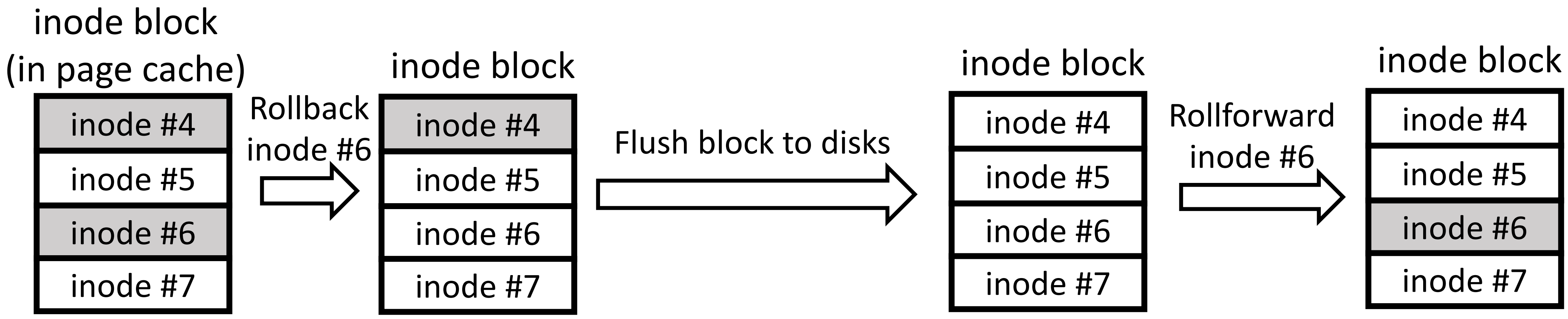
Delayed disk writes

- Auxiliary structures for each operation
- More complex dependencies

Cyclic dependencies

- Rolling back/forward





Soft Updates Is Complicated

Delayed disk writes

- Auxiliary structures for each operation
- More complex dependencies

Cyclic dependencies

- Rolling back/forward

The mismatch between **per-pointer-based dependency tracking** and **block-based interface of traditional disks**

Soft Updates Meets NVM

Soft Updates

- ✓ No synchronous cache flushes
- ✓ Immediately usable after crash

NVM: **byte-addressable** and **fast**

- ✓ Direct write to NVM without delays
- ✓ No false sharing => no rolling back/forward
- ✓ Simple dependency tracking/enforcement

SoupFS

A simple and fast NVMFS derived from soft updates

- Hashtable-based directories
  - No false sharing
- Pointer-based dual views
  - No synchronous cache flushes
- Semantic-aware dependency tracking/enforcement
  - Simple dependency tracking/enforcement

Get the best of both Soft Updates and NVM

Overview

Background

Design & Implementation

- Hashtable-based directories
- Pointer-based dual views
- Semantic-aware dependency tracking/enforcement

Evaluation

Conclusion

Overview

Background

Design & Implementation

- Hashtable-based directories
- Pointer-based dual views
- Semantic-aware dependency tracking/enforcement

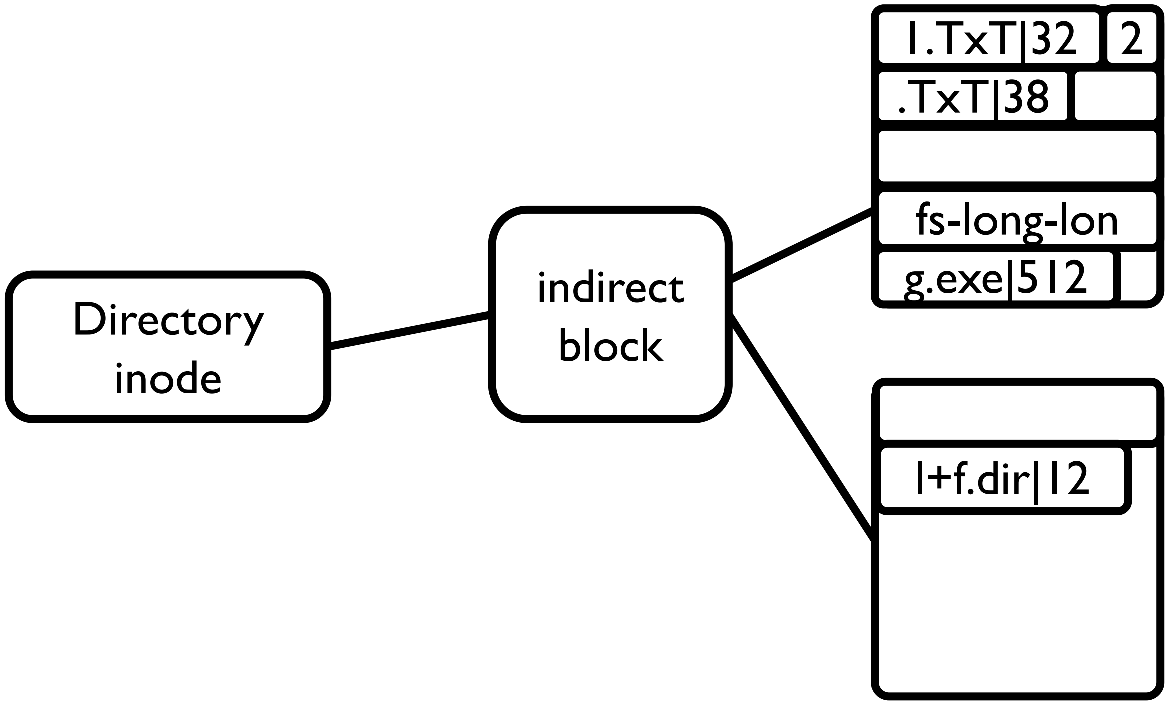
Evaluation

Conclusion

Block-based Directories

Block-based file systems usually use block-based directories

- False sharing
  - ✘ Cyclic dependency
  - ✘ Rolling back/forward
- Slow access
  - ✘ Linear scan



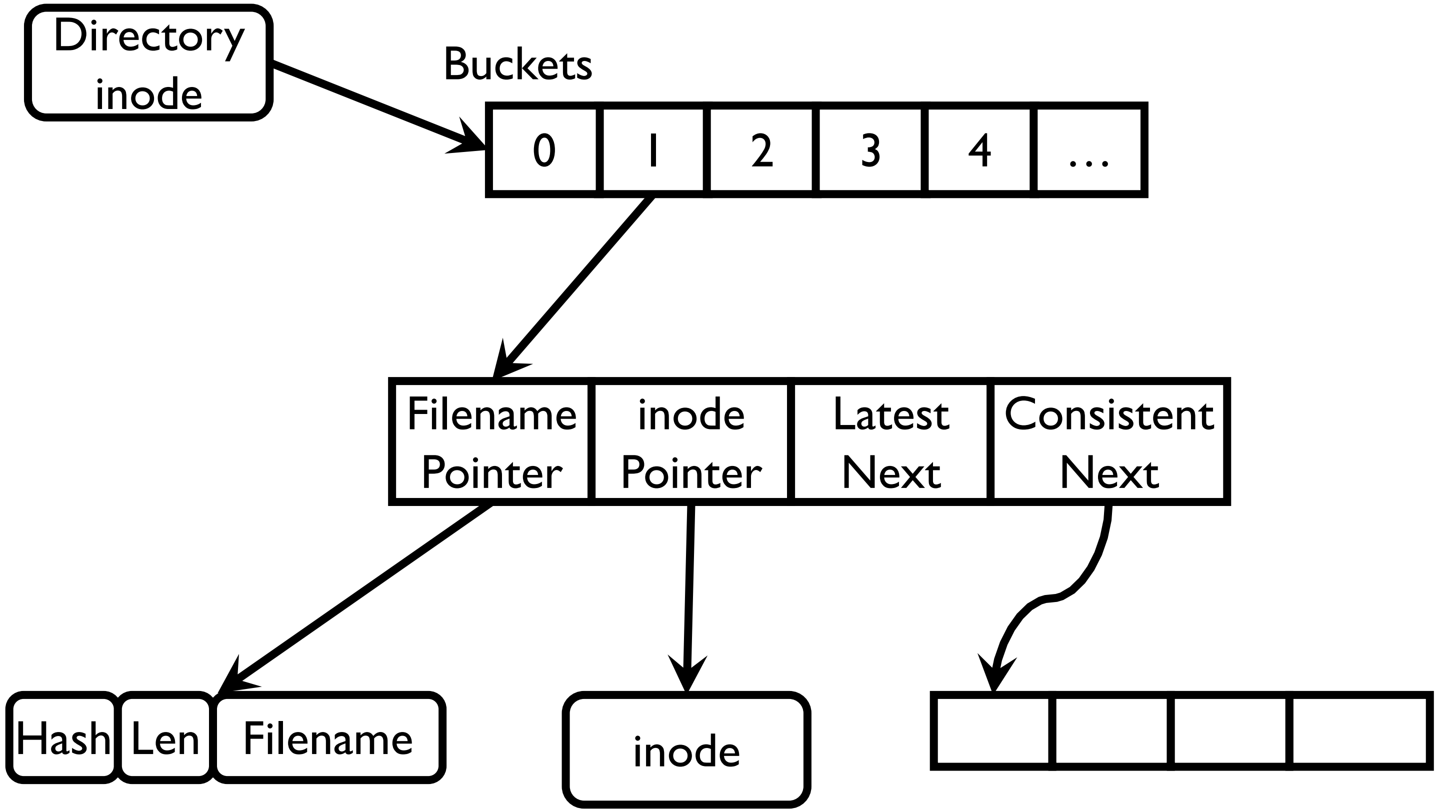
Hashtable-based Directories

Optimized for cache lines

- ✓ No false sharing
- ✓ No cyclic dependency

Efficient access

- ✓ No linear scan



Overview

Background

Design & Implementation

- ✓ Hashtable-based directories
- ▪ Pointer-based dual views
- ▪ Semantic-aware dependency tracking/enforcement

Evaluation

Conclusion

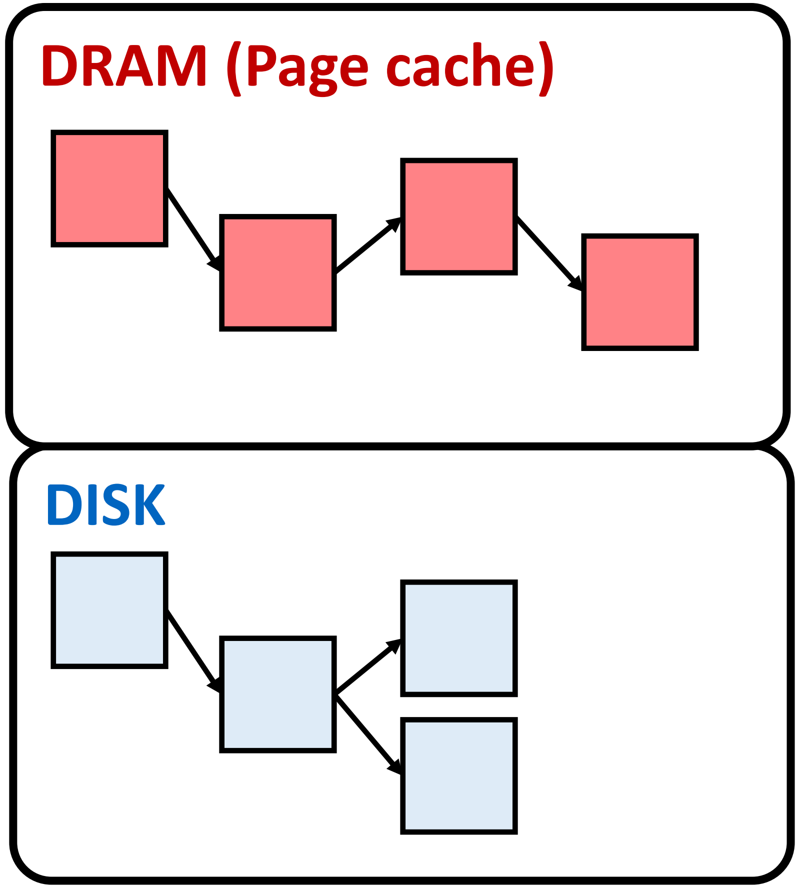
Dual Views

Latest view in page cache

Consistent view in disks

Dual views

- Eliminate synchronous writes
- Provide usability after crash



Traditional Soft Updates

Dual Views

~~Latest view in page cache~~

Consistent view in ~~disks~~ NVM

Latest view?

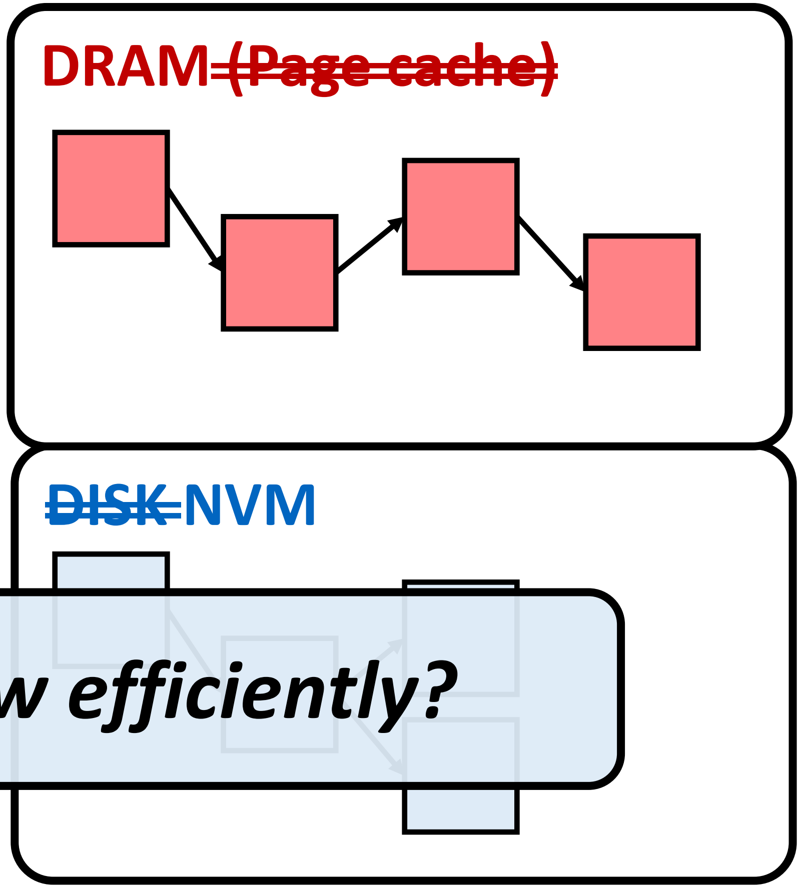
Another copy of metadata in DRAM

✗ Double writes

✗ Double storage overhead

✗ Unnecessary synchronizations

Challenge: How to present latest view efficiently?

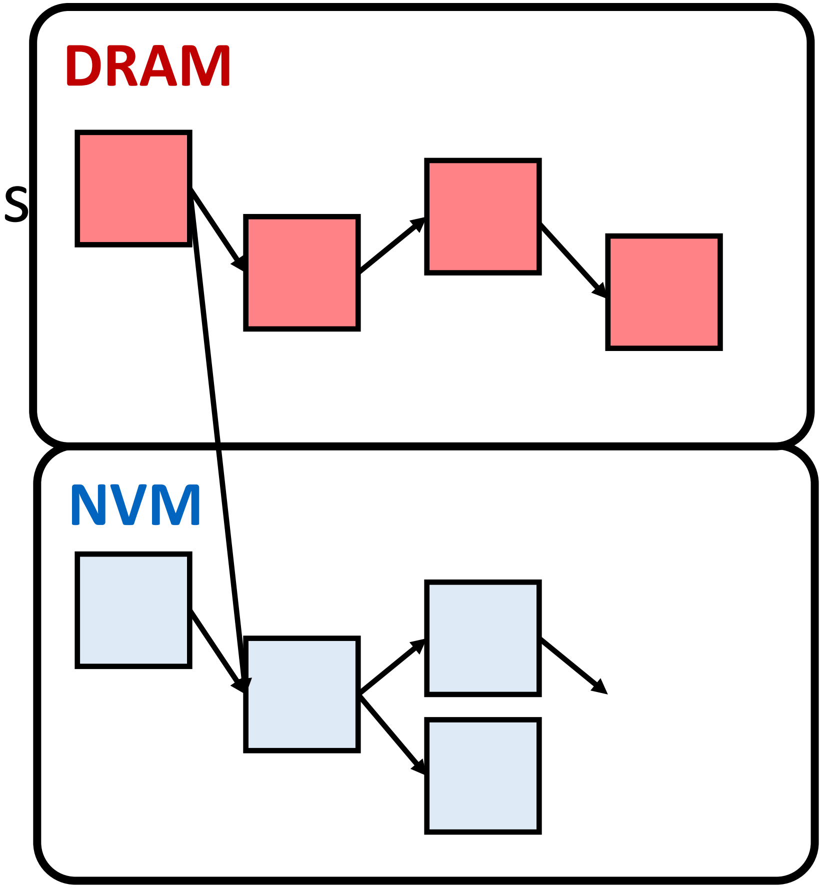


Soft Updates on NVM

Pointer-based Dual Views

Reuse data structures in both views

Distinguish views by different pointers/structures



Soft Updates on NVM

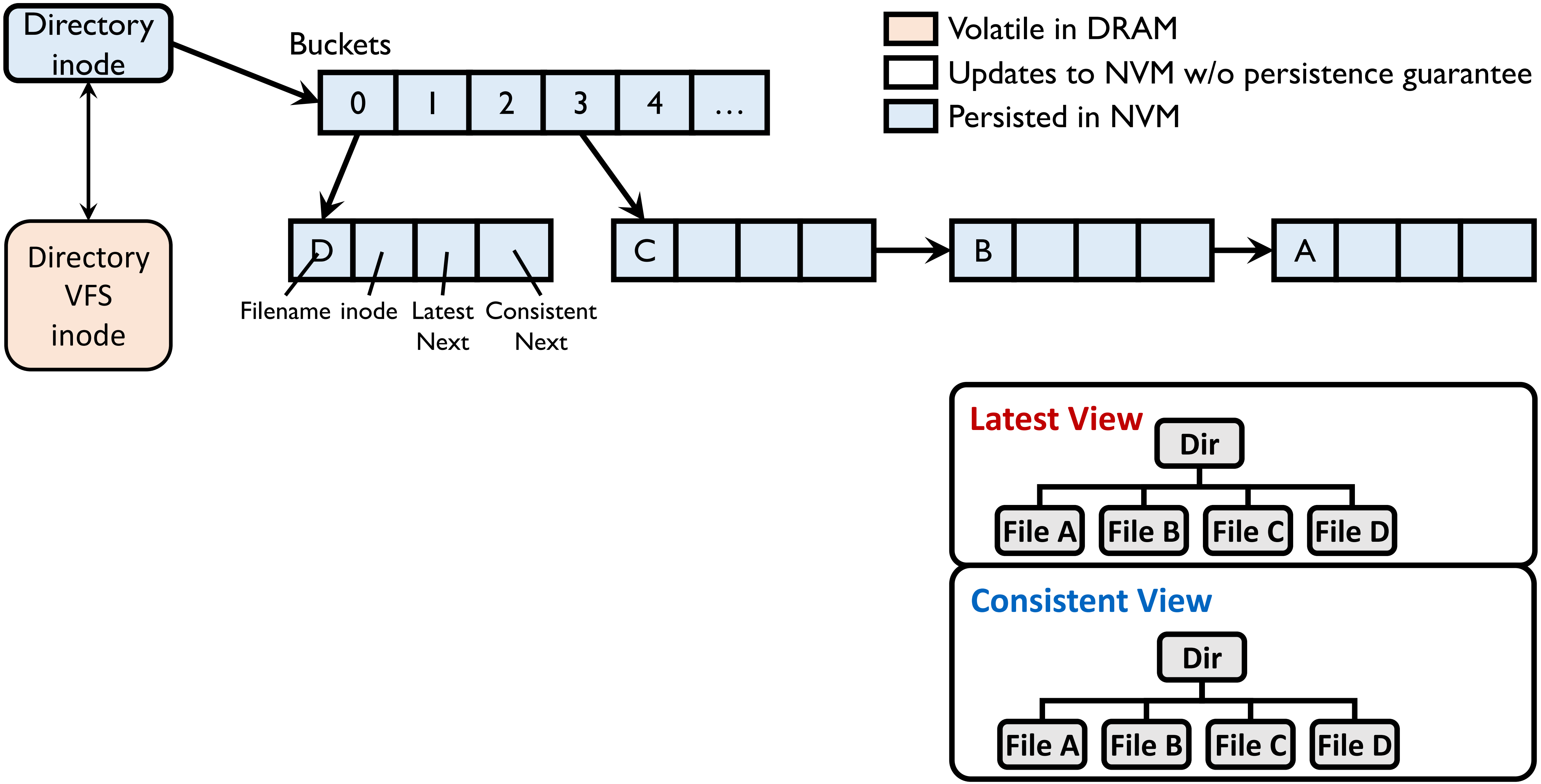
Pointer-based Dual Views

Reuse data structures in both views

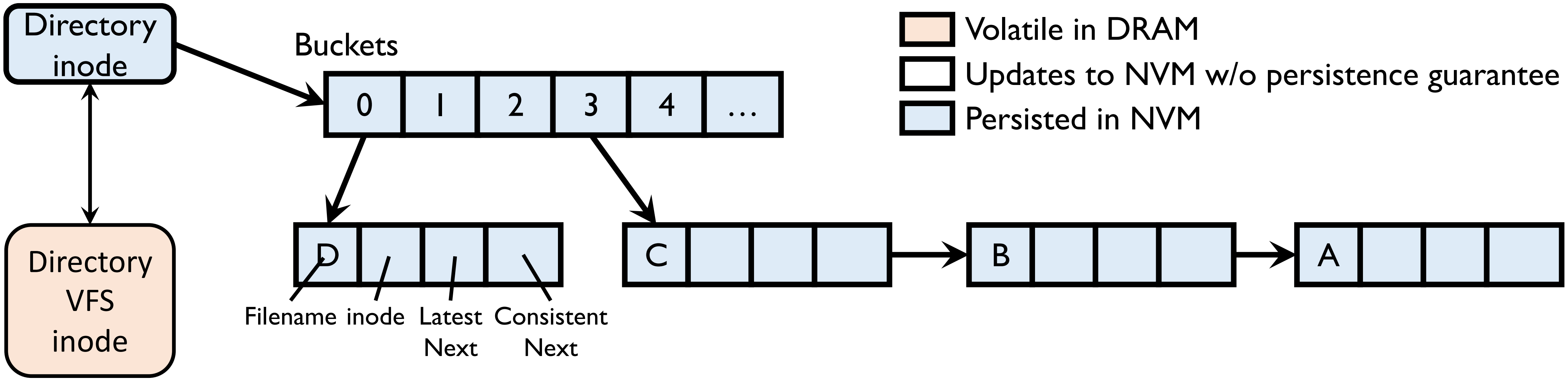
Distinguish views by different pointers/structures

Data Structures	*In Consistent View*	*In Latest View*
inode	SoupFS inode	VFS inode
dentry	consistent next pointer	latest next pointer
hash table	bucket	latest bucket if exists
B-tree	root/height in SoupFS inode	root/height in VFS inode

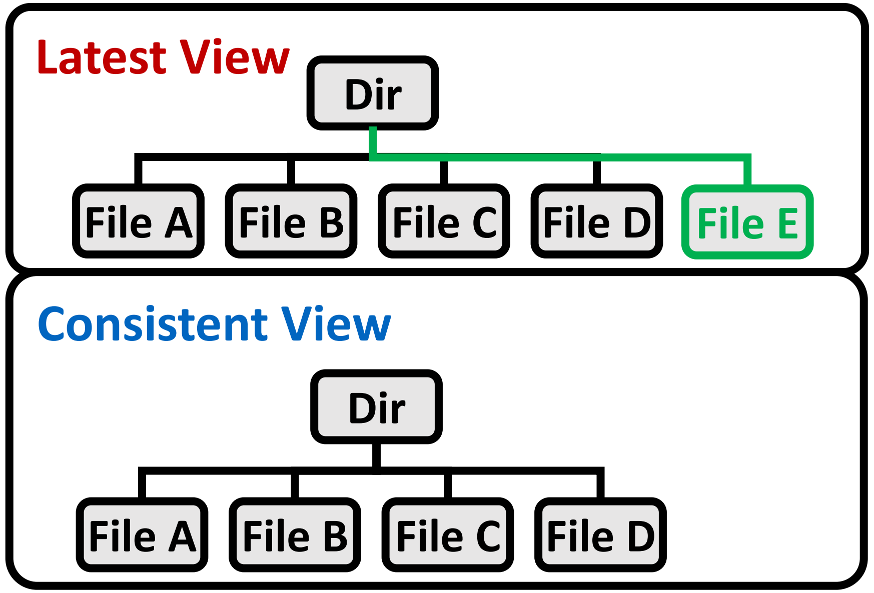
Pointer-based Dual Views



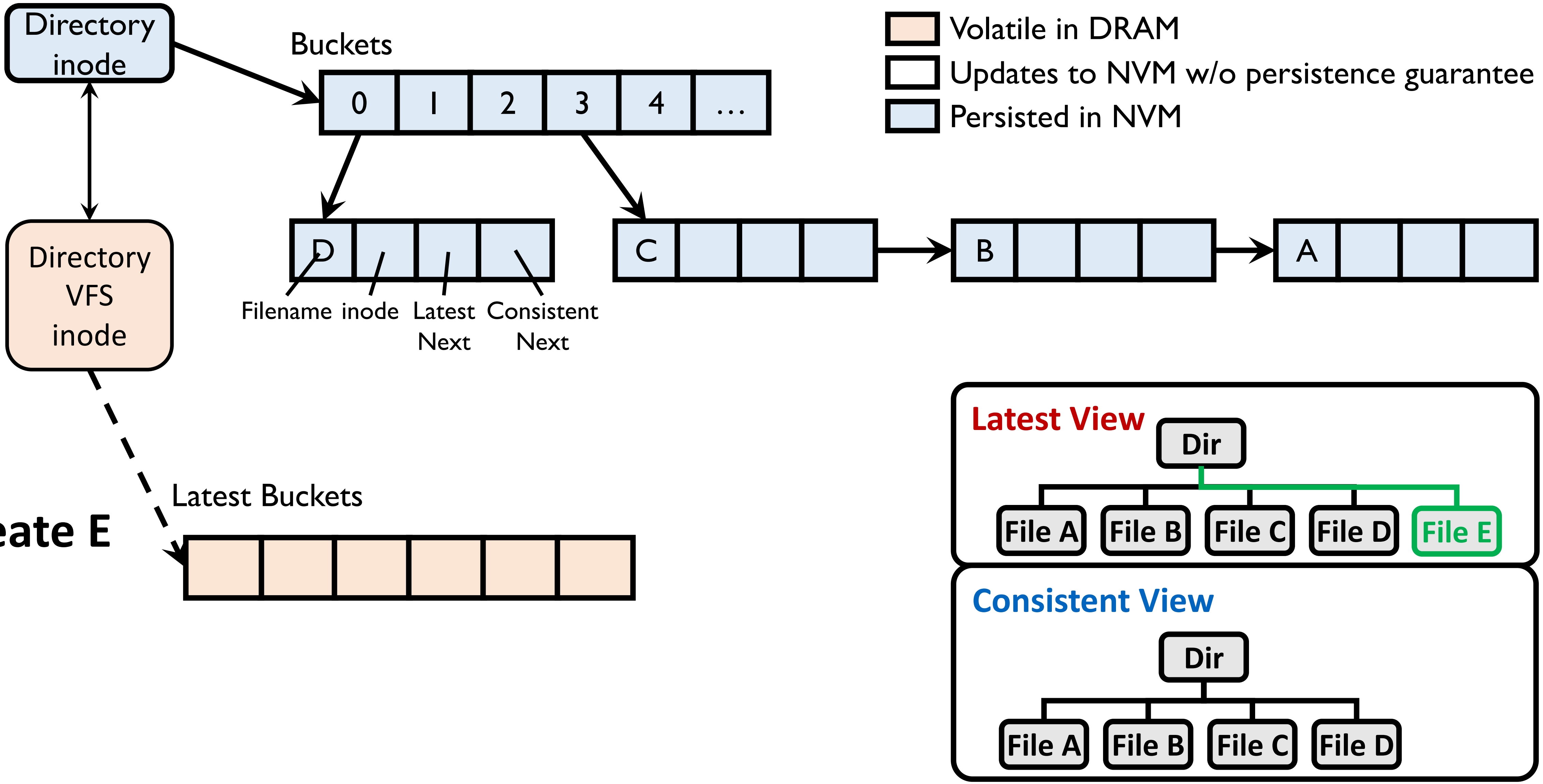
Pointer-based Dual Views



- create E

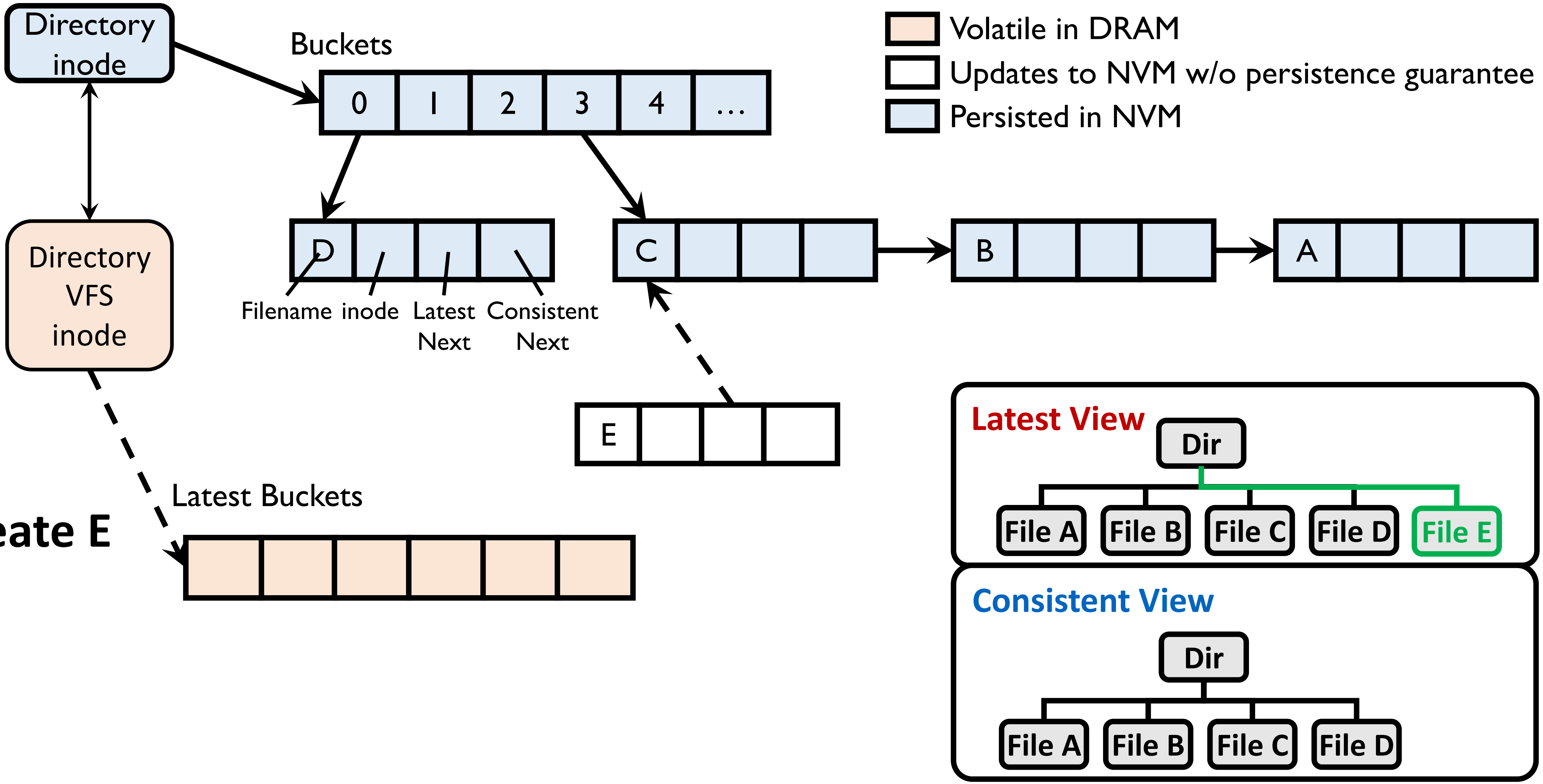


Pointer-based Dual Views



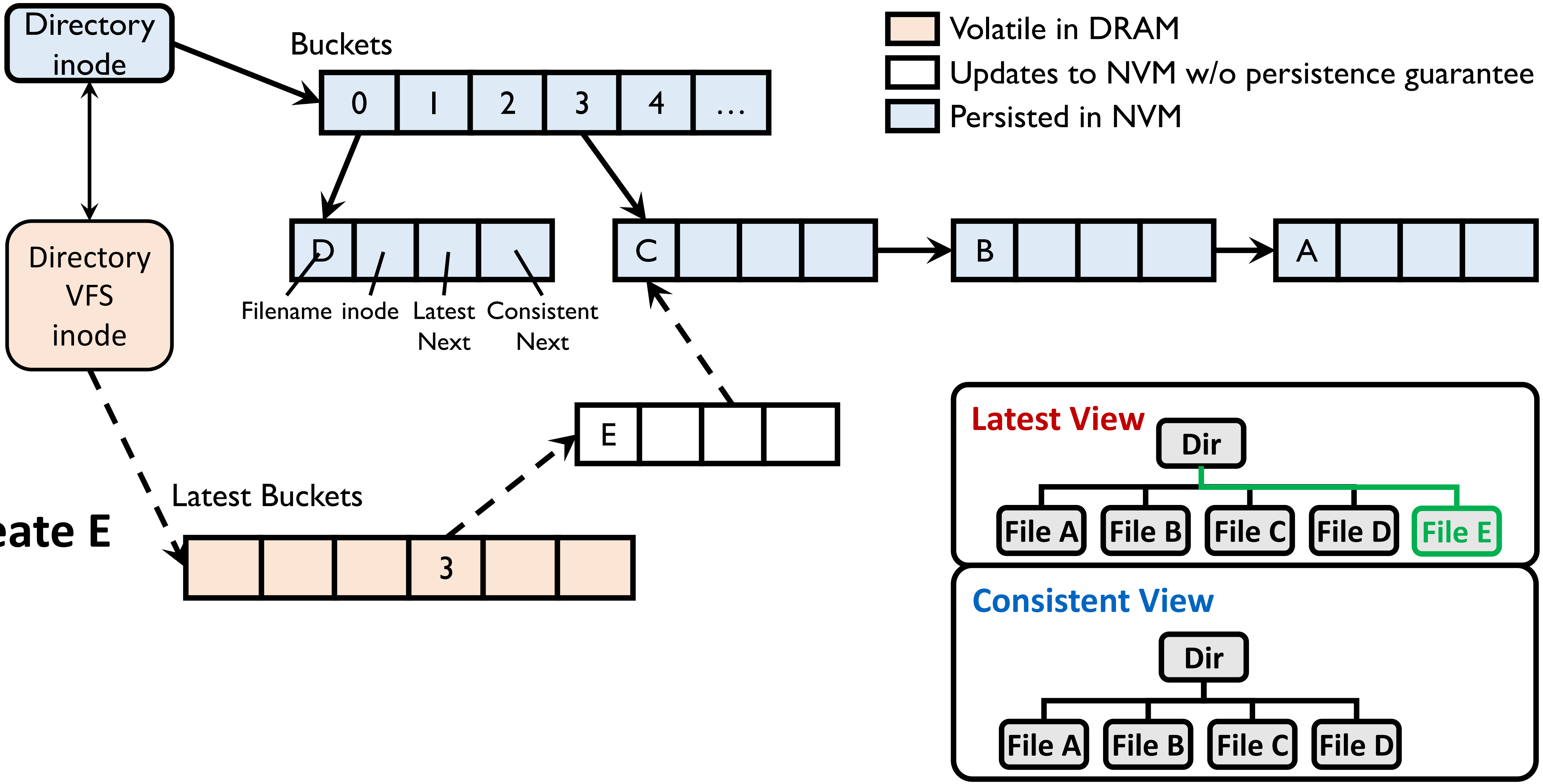
➢ **create E**

Pointer-based Dual Views



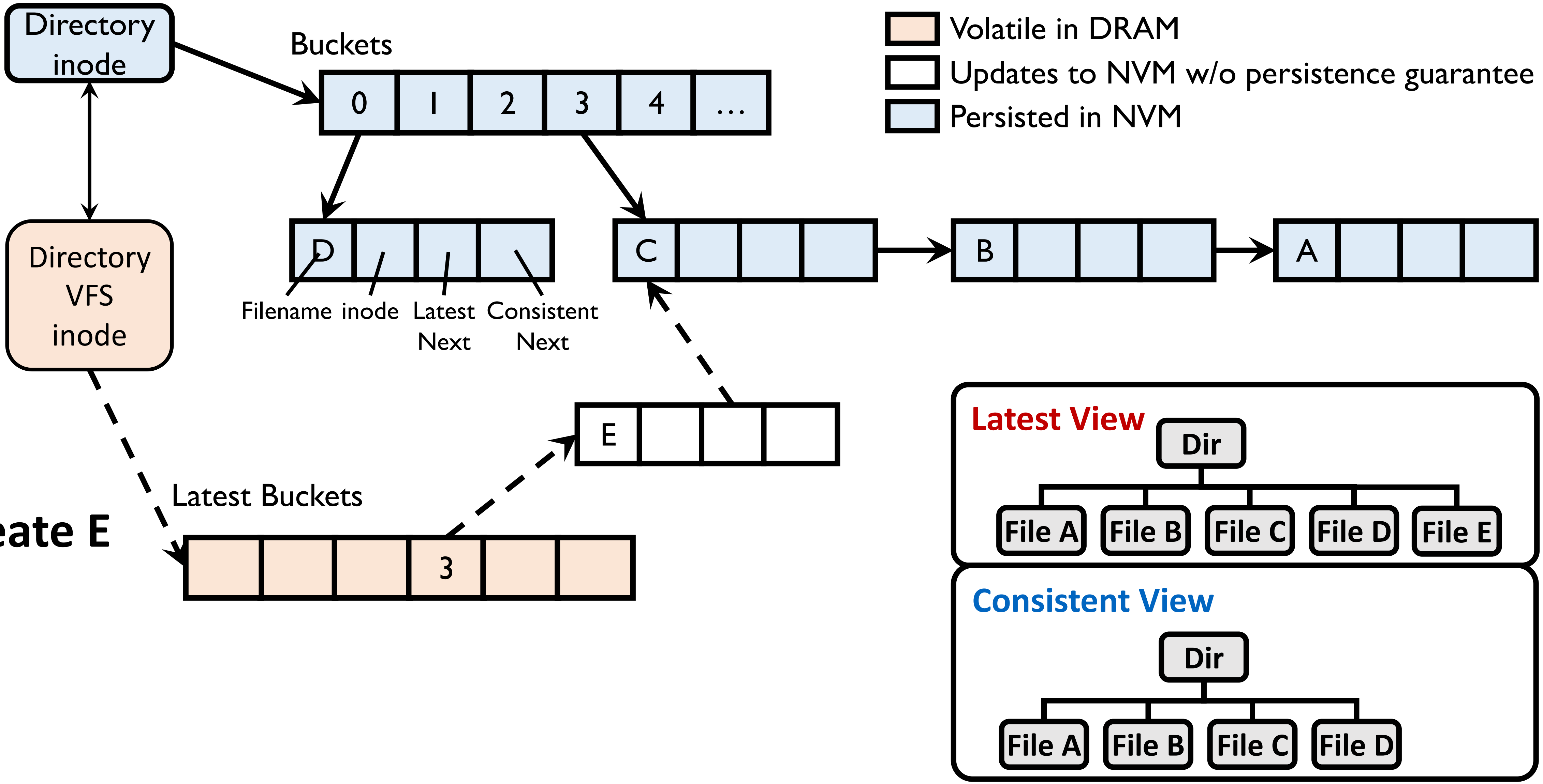
- ➢ **create E**

Pointer-based Dual Views



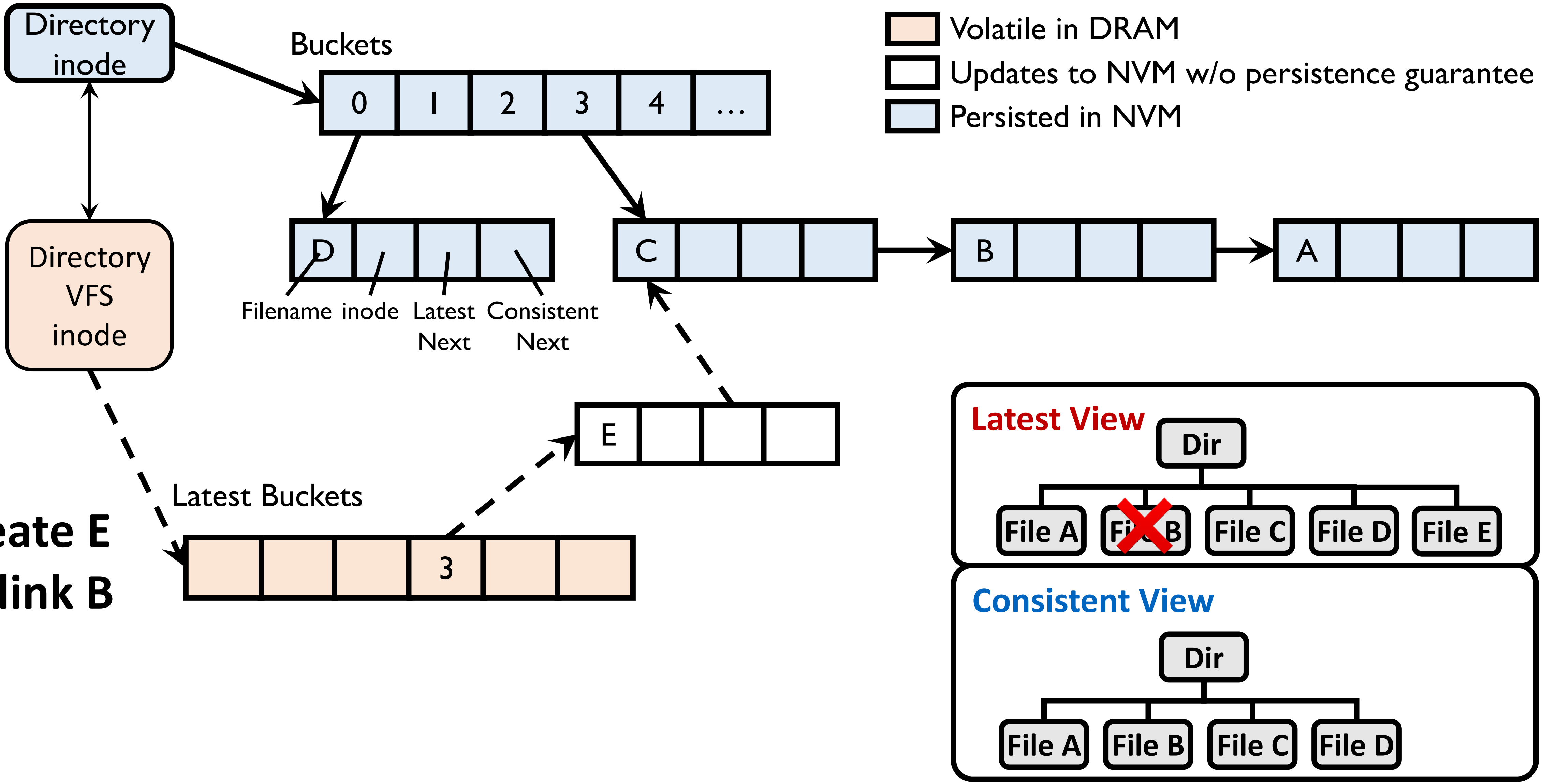
- ➢ **create E**

Pointer-based Dual Views



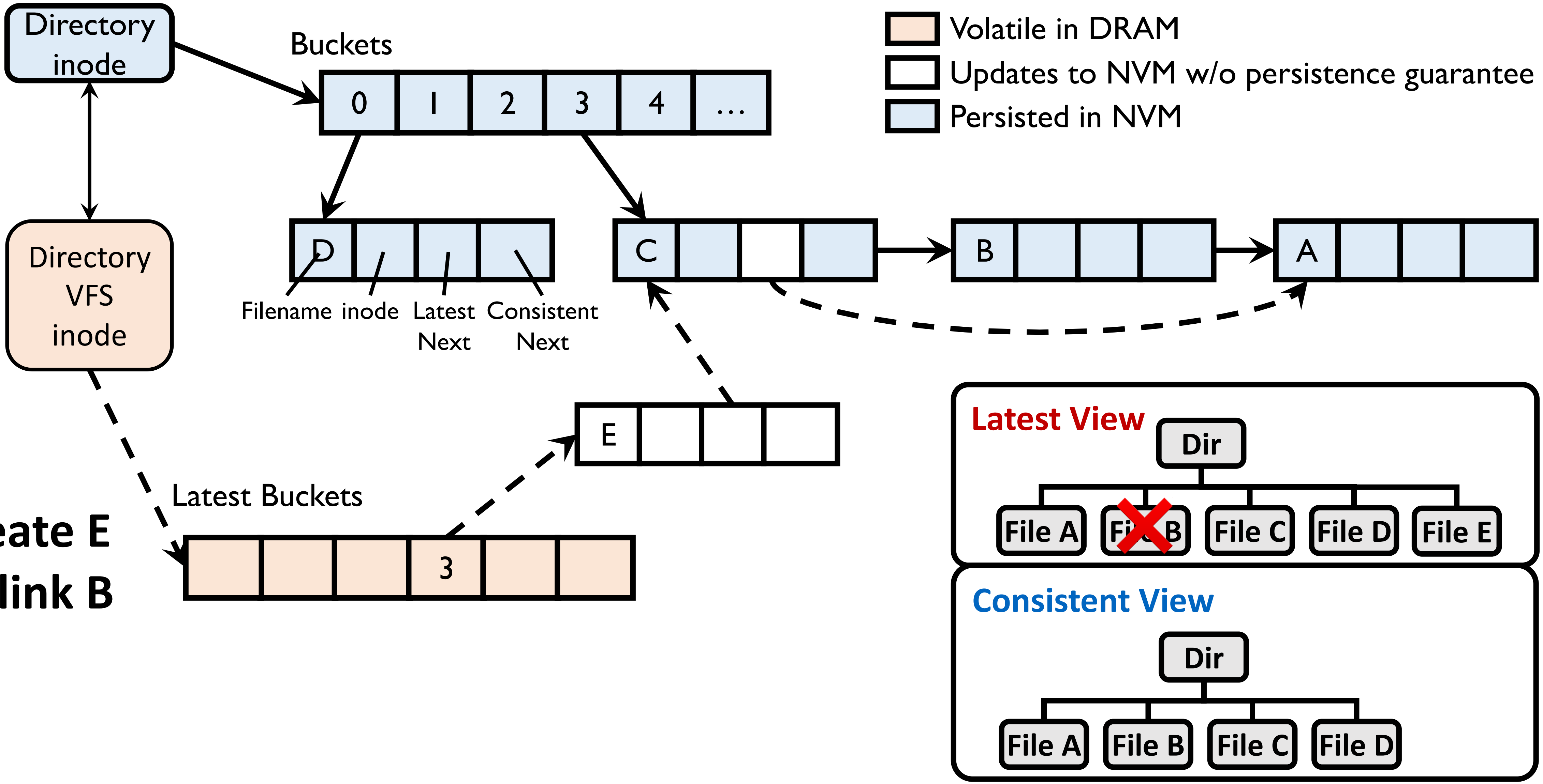
- **create E**

Pointer-based Dual Views



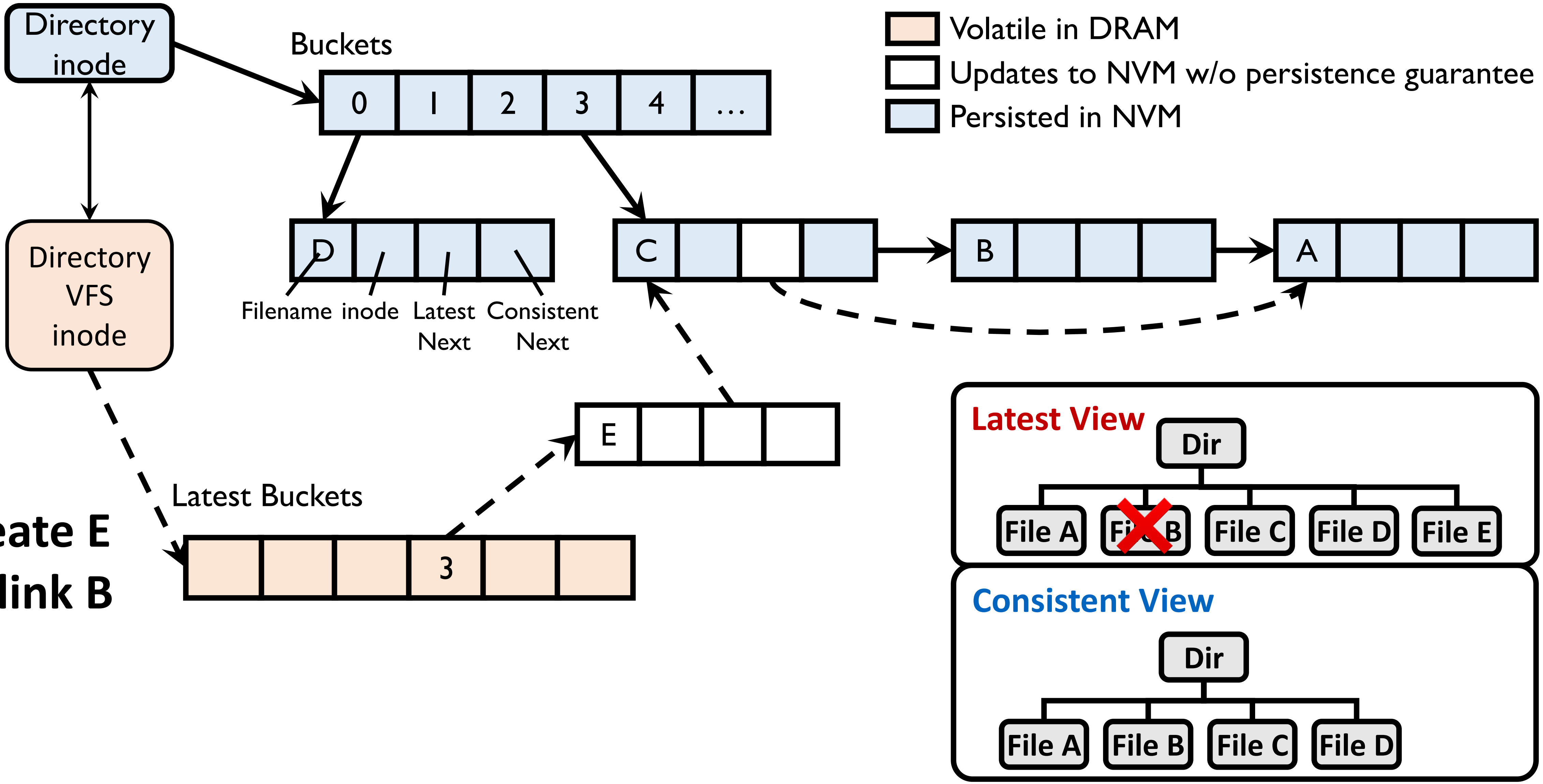
- create E
- ➢ unlink B

Pointer-based Dual Views



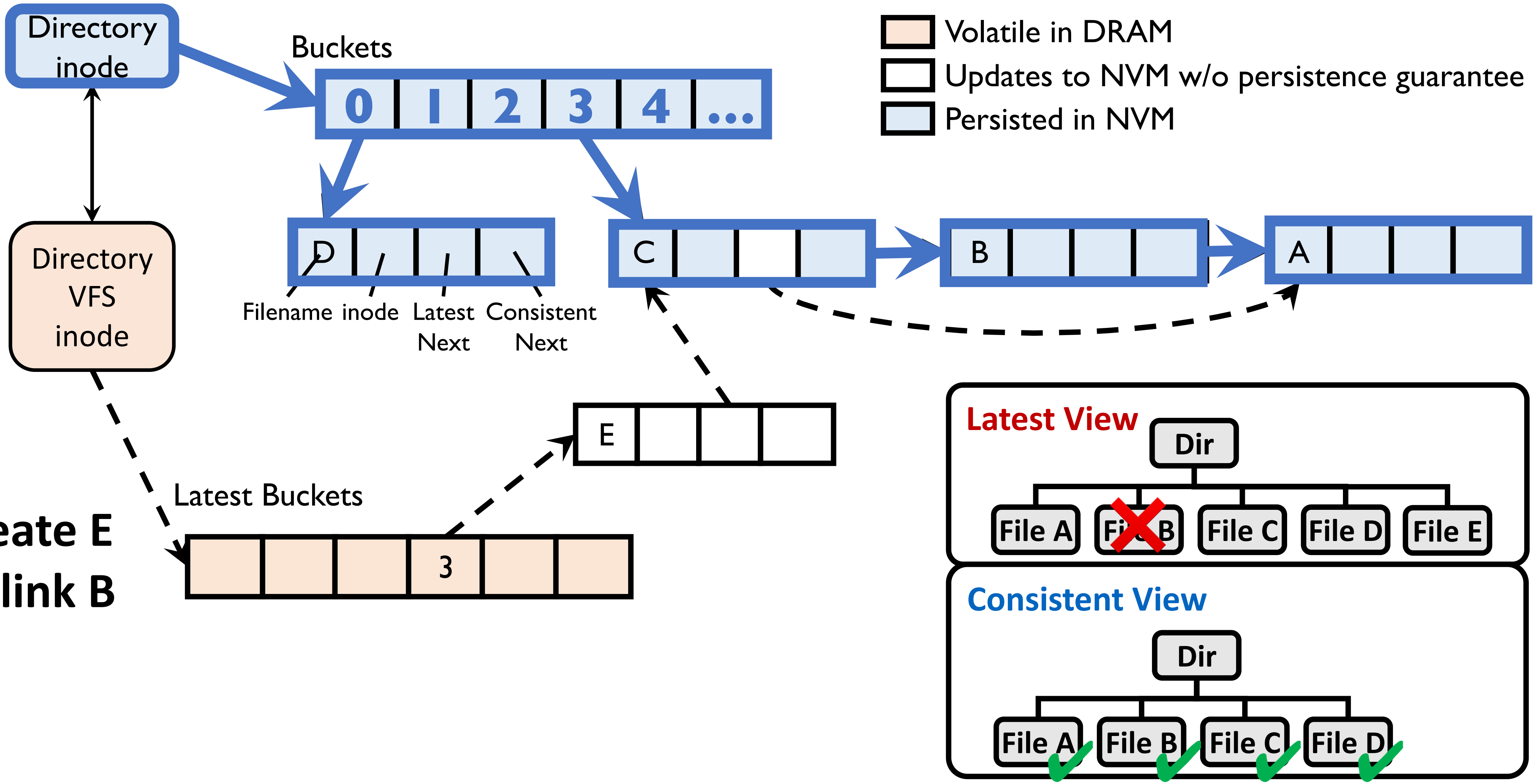
- create E
- ➢ unlink B

Pointer-based Dual Views



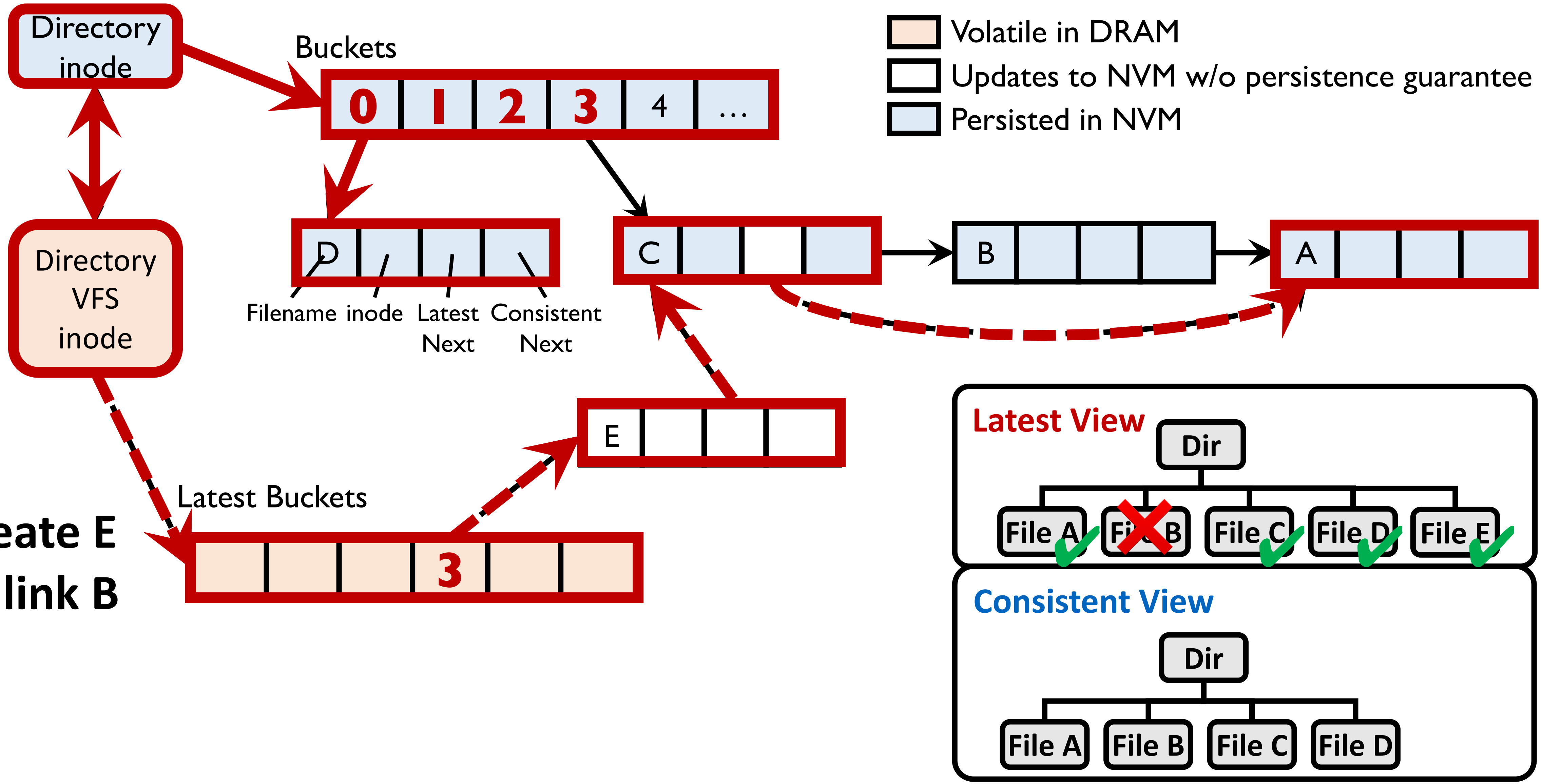
- **create E**
- **unlink B**

Pointer-based Dual Views



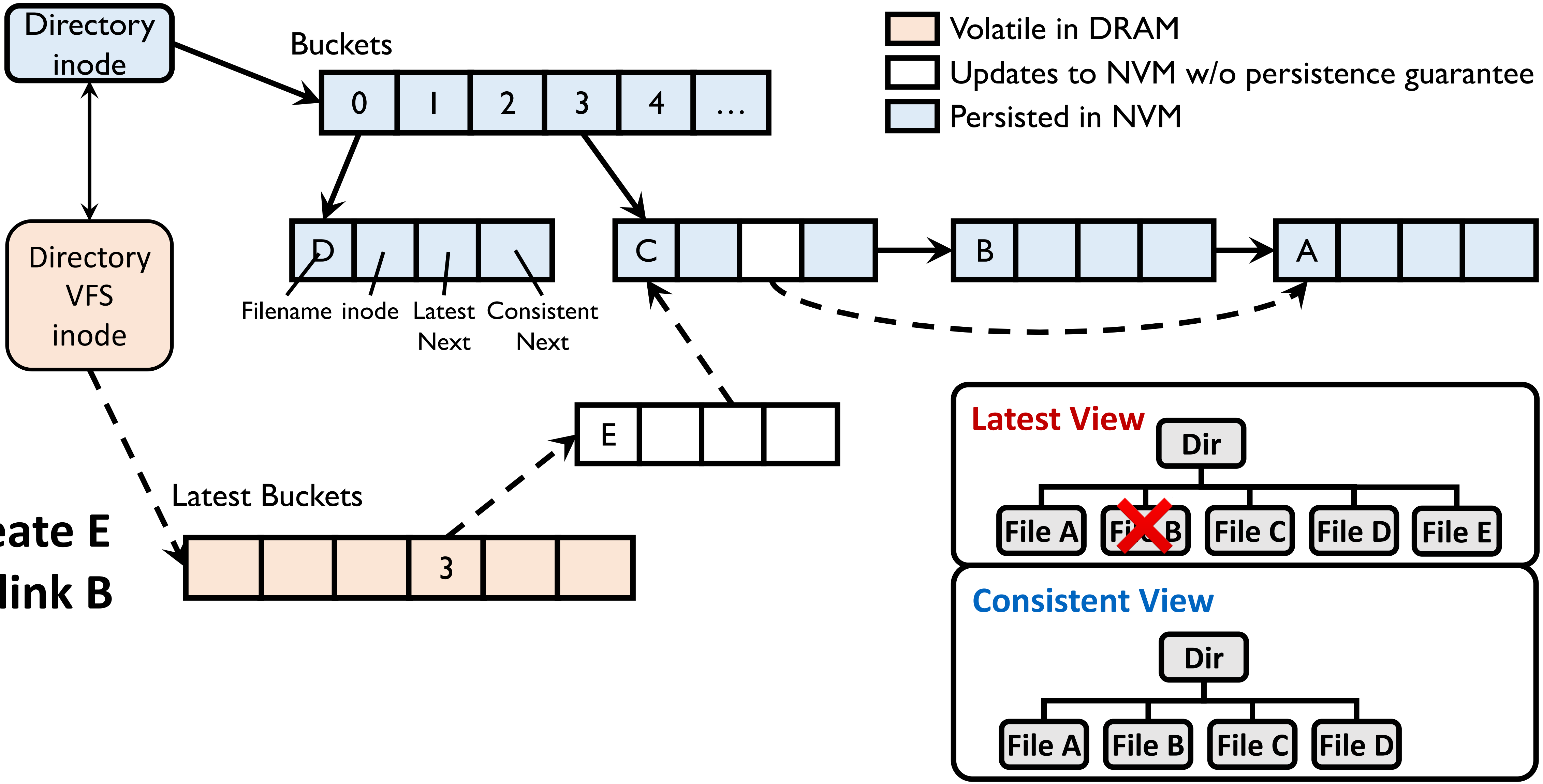
- **create E**
- **unlink B**

Pointer-based Dual Views



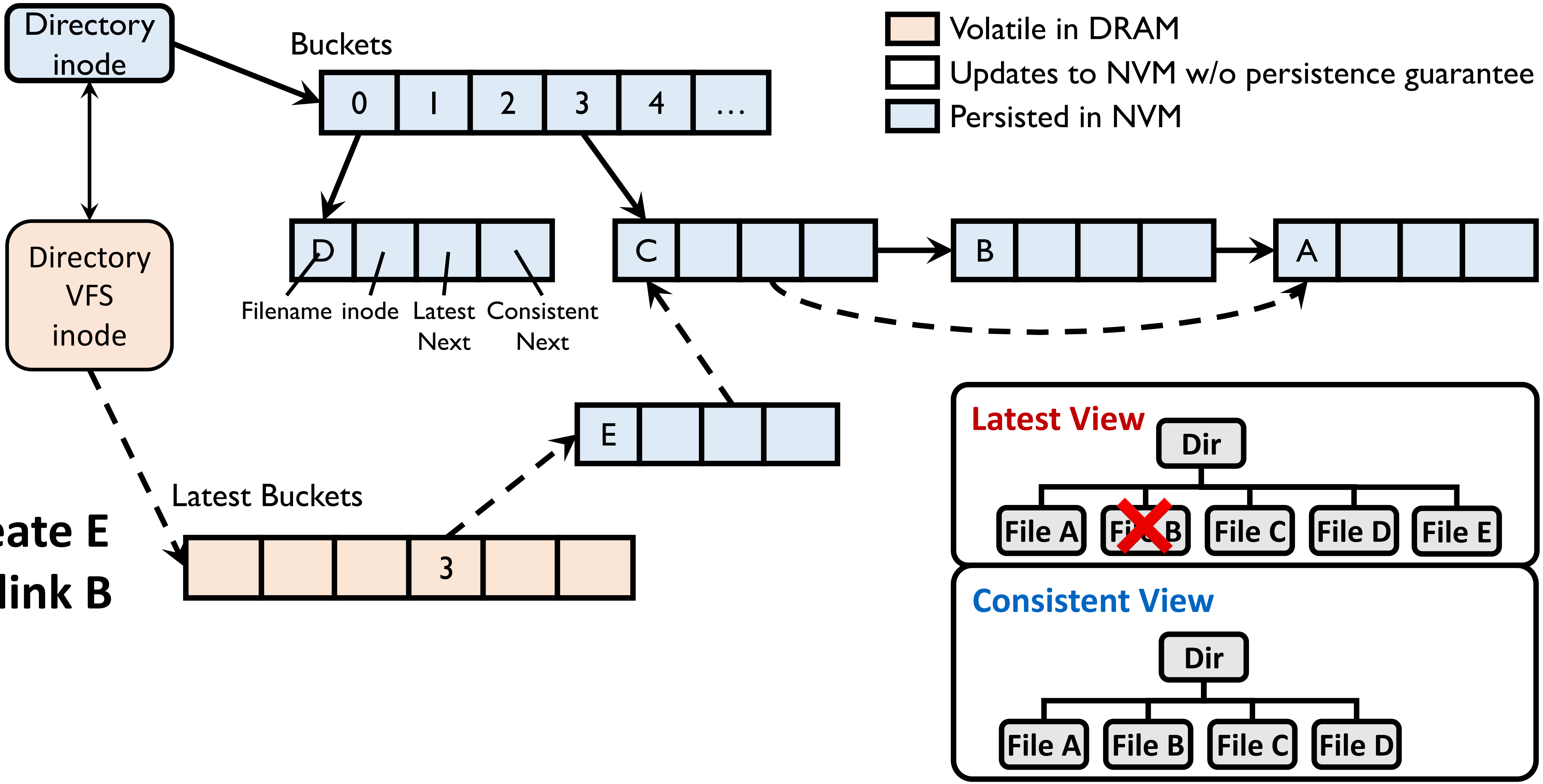
- **create E**
- **unlink B**

Pointer-based Dual Views



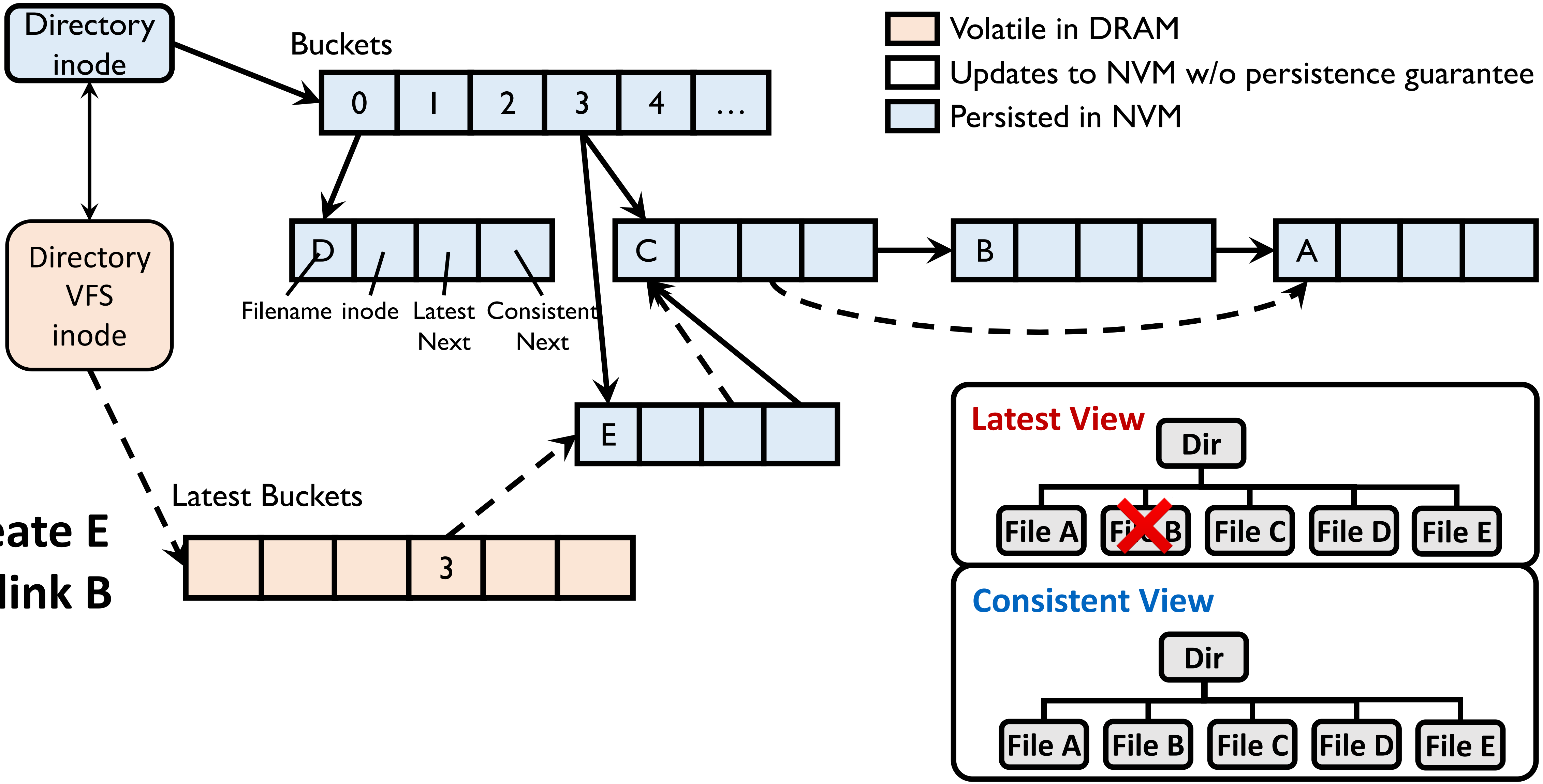
- **create E**
- **unlink B**

Pointer-based Dual Views



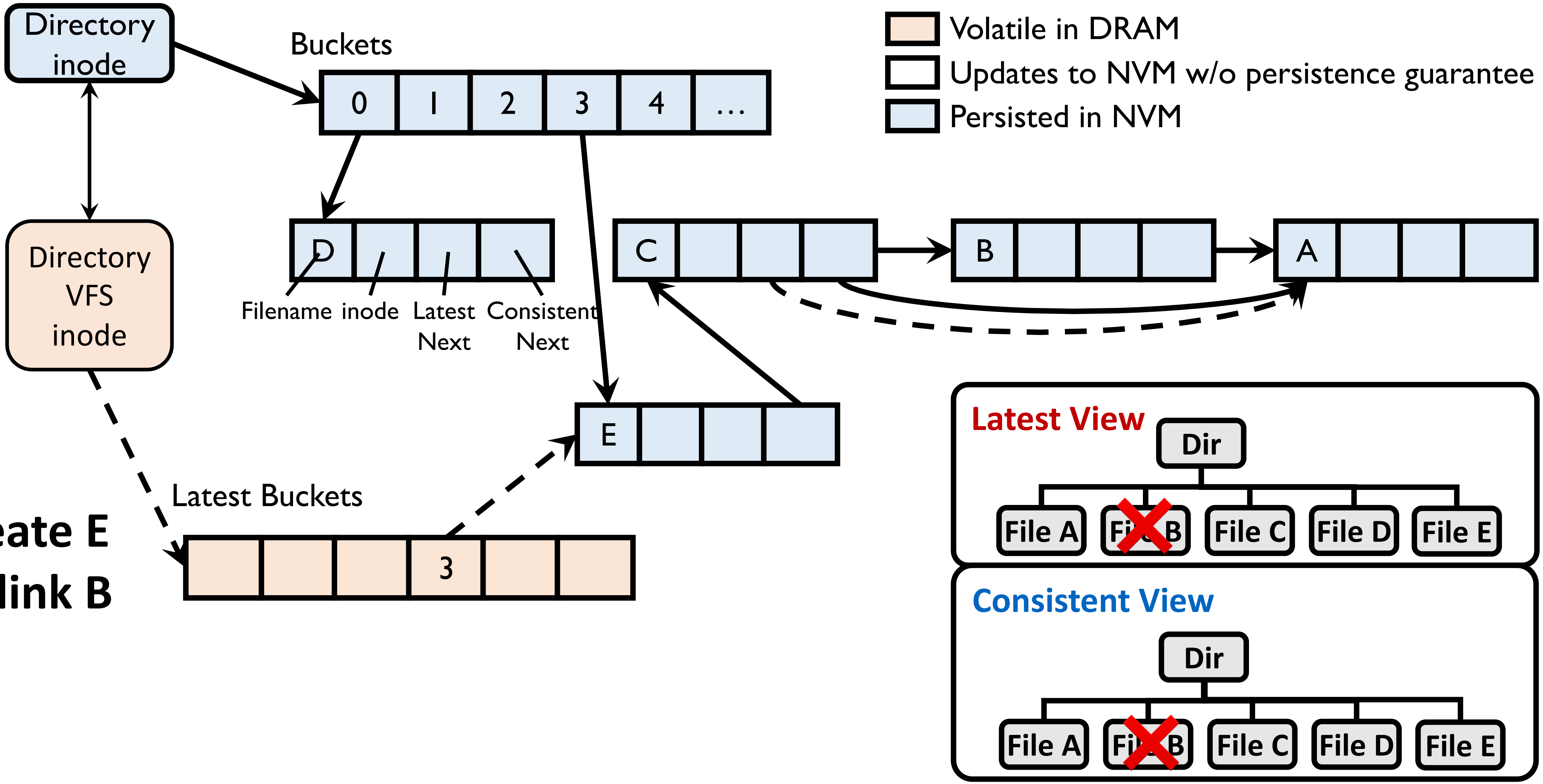
- **create E**
- **unlink B**

Pointer-based Dual Views



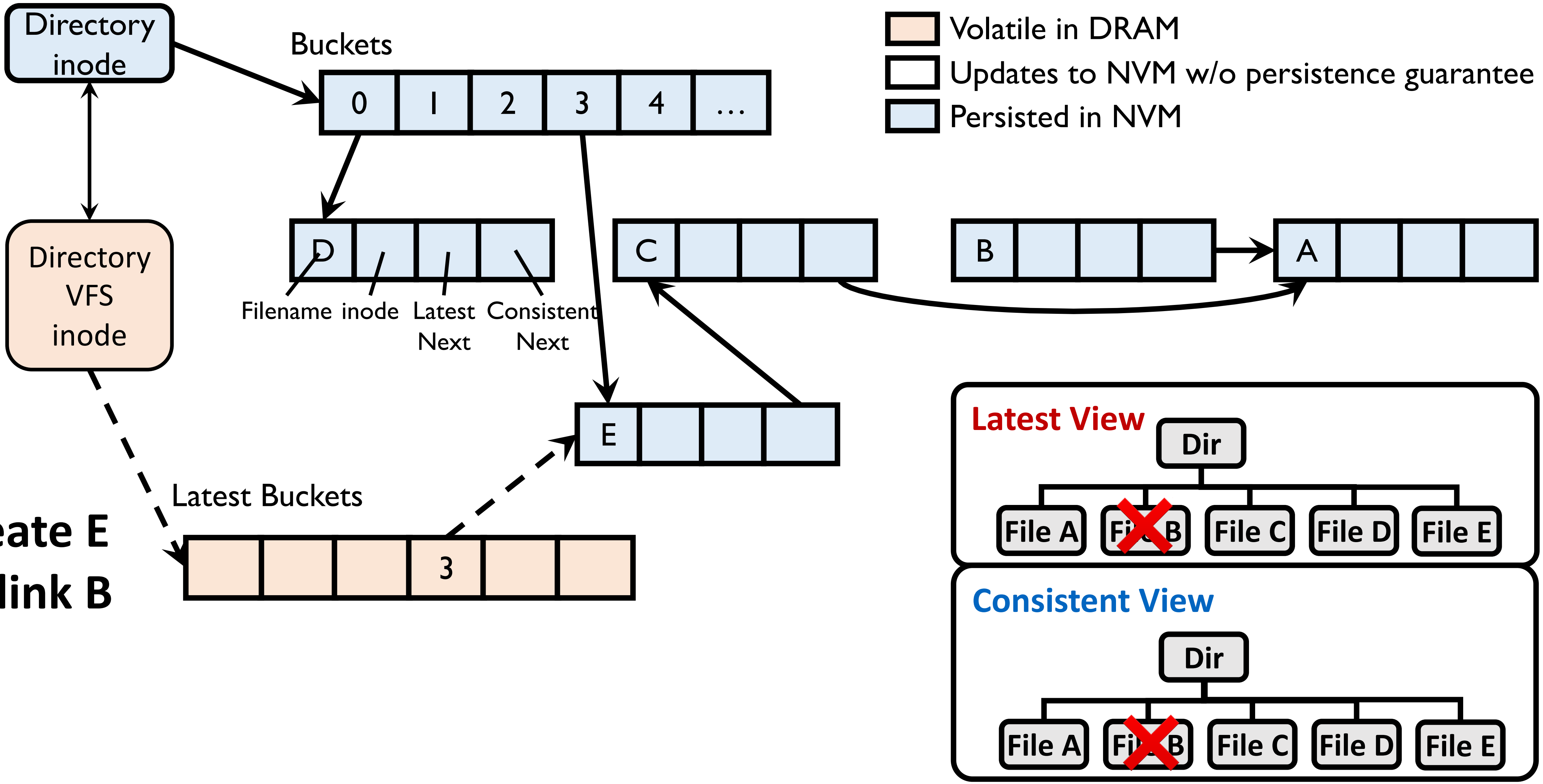
- **create E**
- **unlink B**

Pointer-based Dual Views



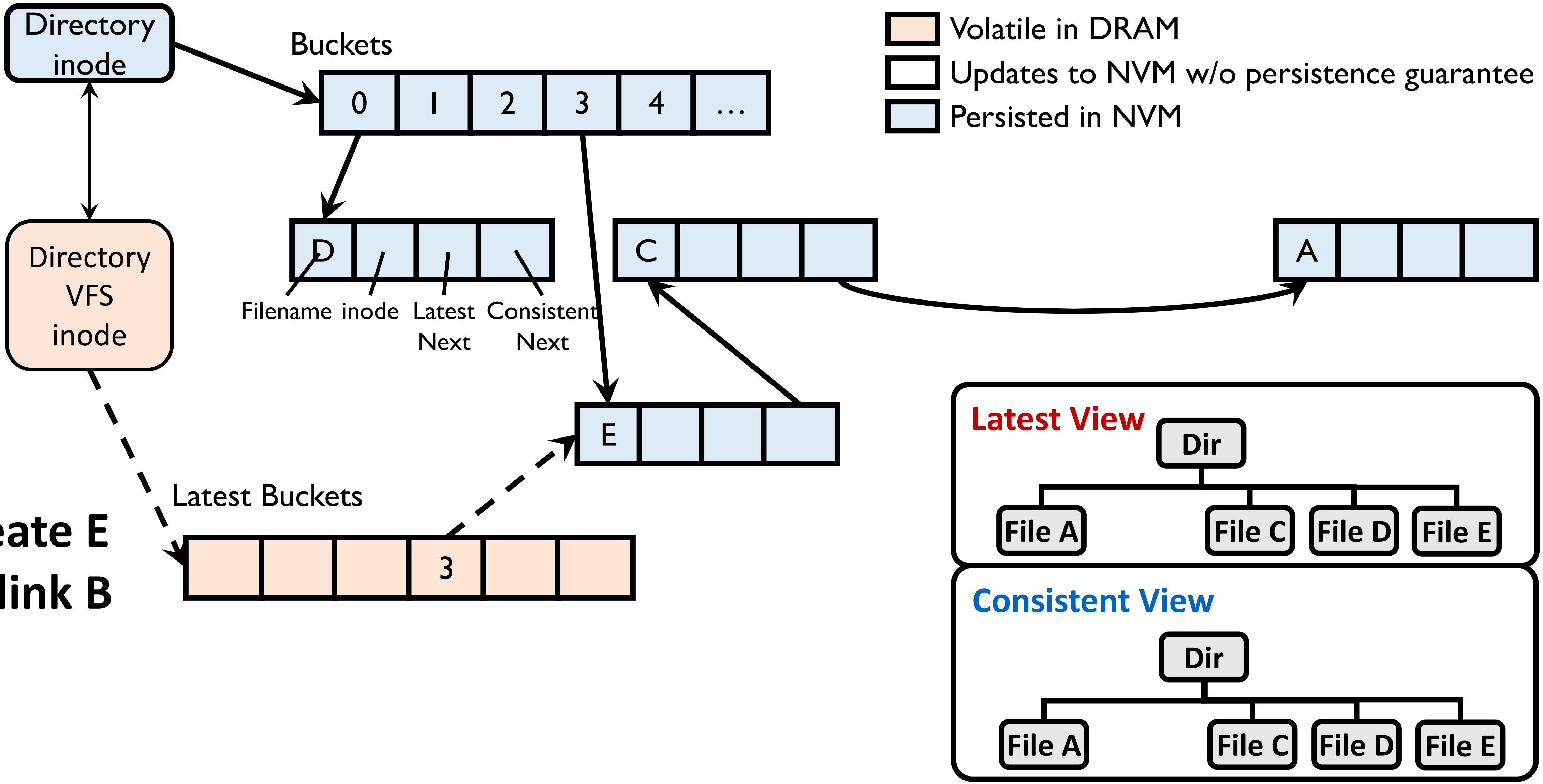
- **create E**
- **unlink B**

Pointer-based Dual Views



- create E
- unlink B

Pointer-based Dual Views

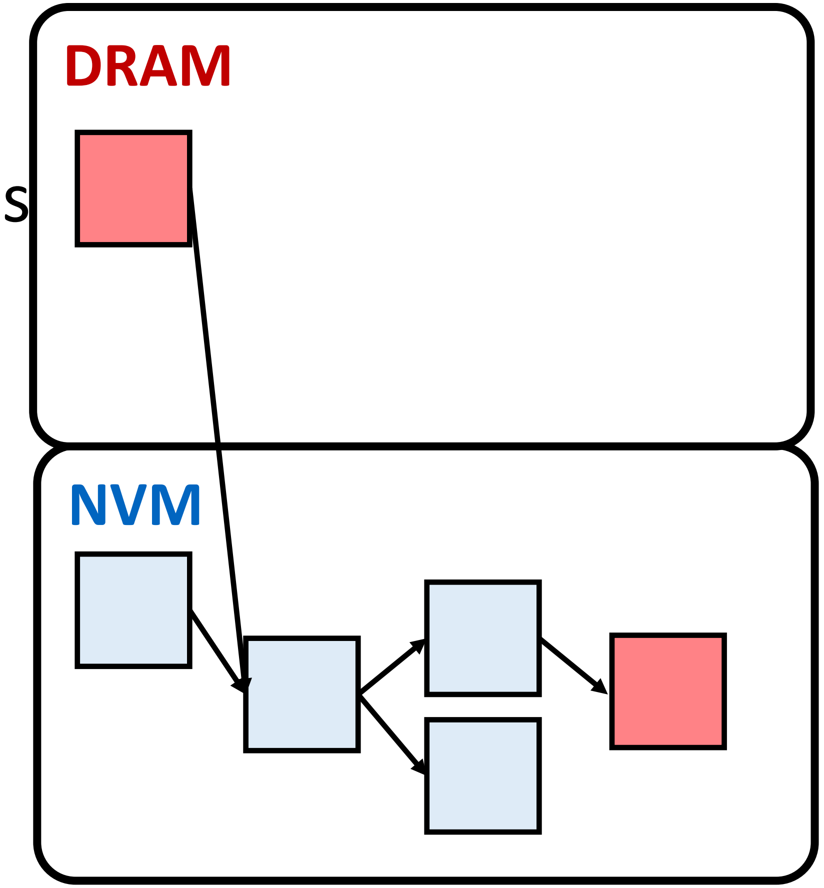


- **create E**
- **unlink B**

Pointer-based Dual Views

Reuse data structures in both views

Distinguish views by different pointers/structures



- ✓ Eliminate synchronous writes
- ✓ Provide usability after crash

- ✓ No double write
- ✓ Little space overhead

Soft Updates on NVM

Overview

Background

Design & Implementation

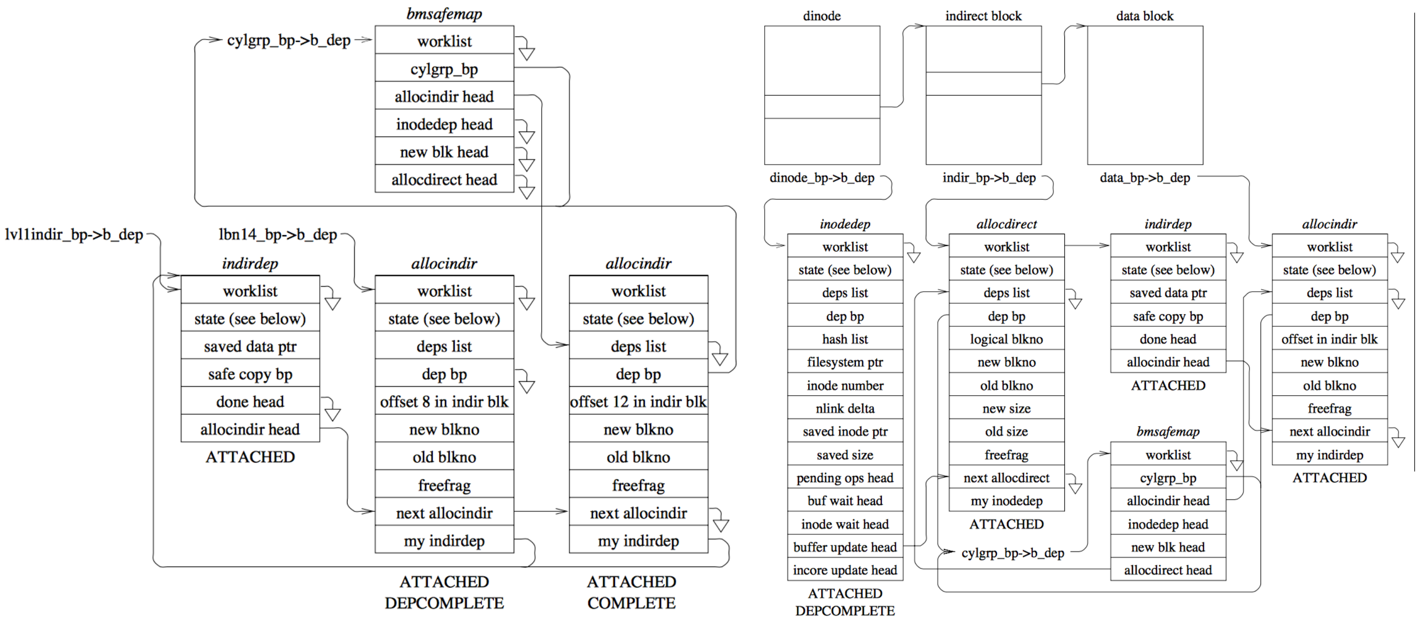
- ✓ Hashtable-based directories
- ✓ Pointer-based dual views
- ▪ Semantic-aware dependency tracking/enforcement

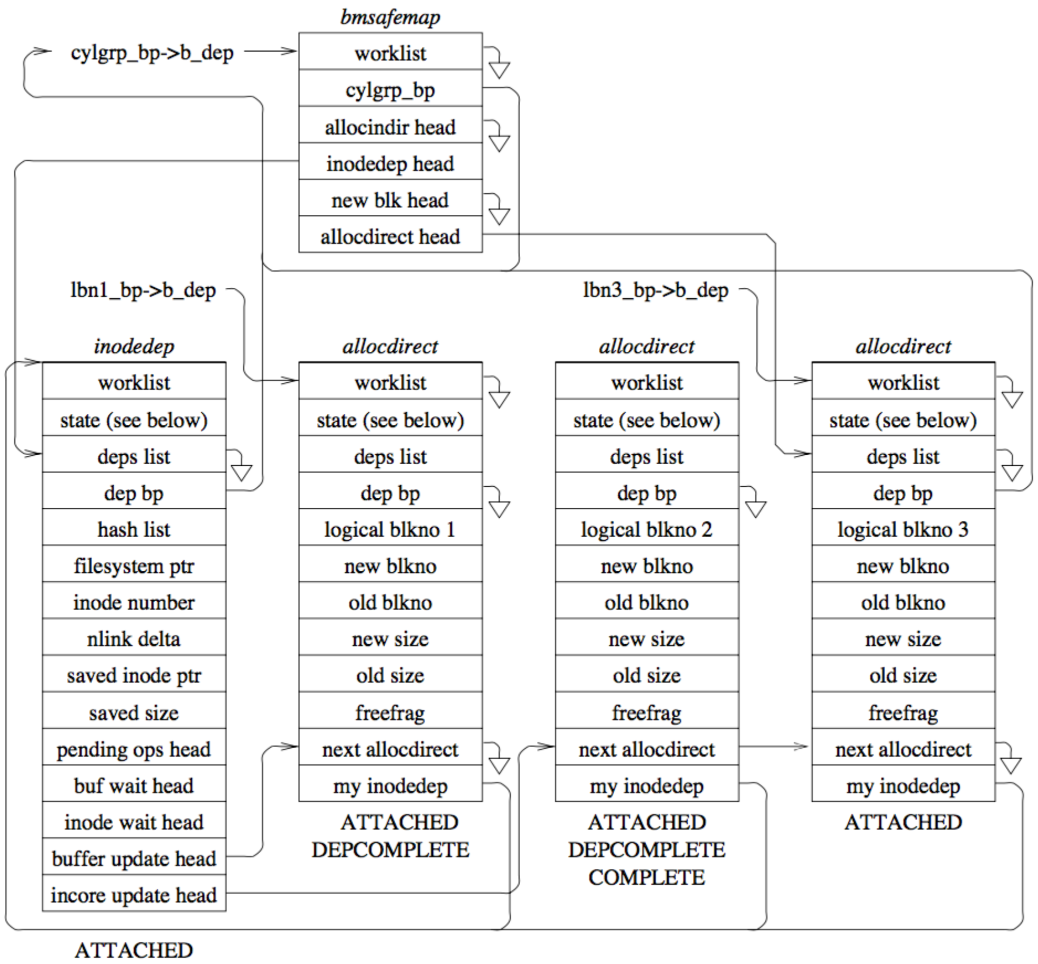
Evaluation

Conclusion

Dependency Tracking

Auxiliary structures for each updates





Dependency Tracking

Auxiliary structures for each updates

*The **semantic gap** between*

the page cache *(where enforcement happens)*

*and **the file system** (where tracking happens)*

After removing page cache, SoupFS involves **semantics** in dependency tracking/enforcement

Semantic-aware Dependency Tracking

Track semantic operations with complementary information

- Enough for dependency enforcement

Operation Type	*Complementary Information (pointers/integers)*
diradd	added dentry, source directory∗, overwritten inode∗
dirrem	removed dentry, destination directory∗
sizechg	the old and new file size
attrchg	nothing

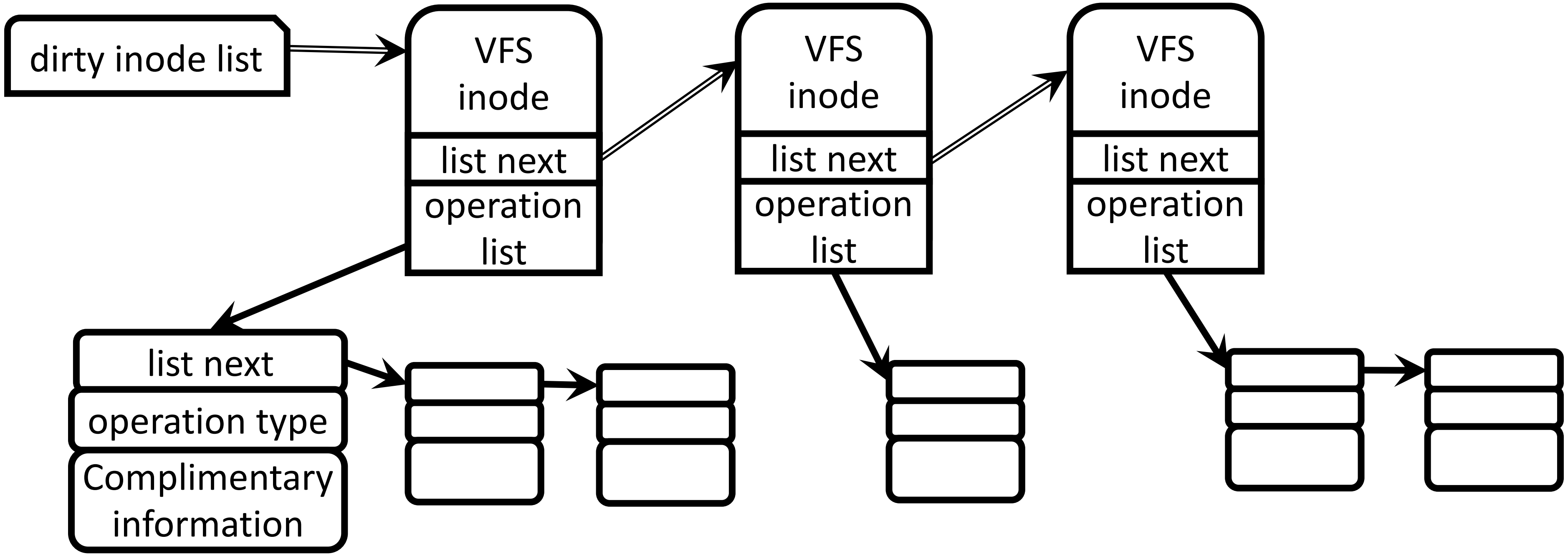
Information tagged with ∗ is for rename operation.

Semantic-aware Dependency Tracking

Track semantic operations with complementary information

- Enough for dependency enforcement

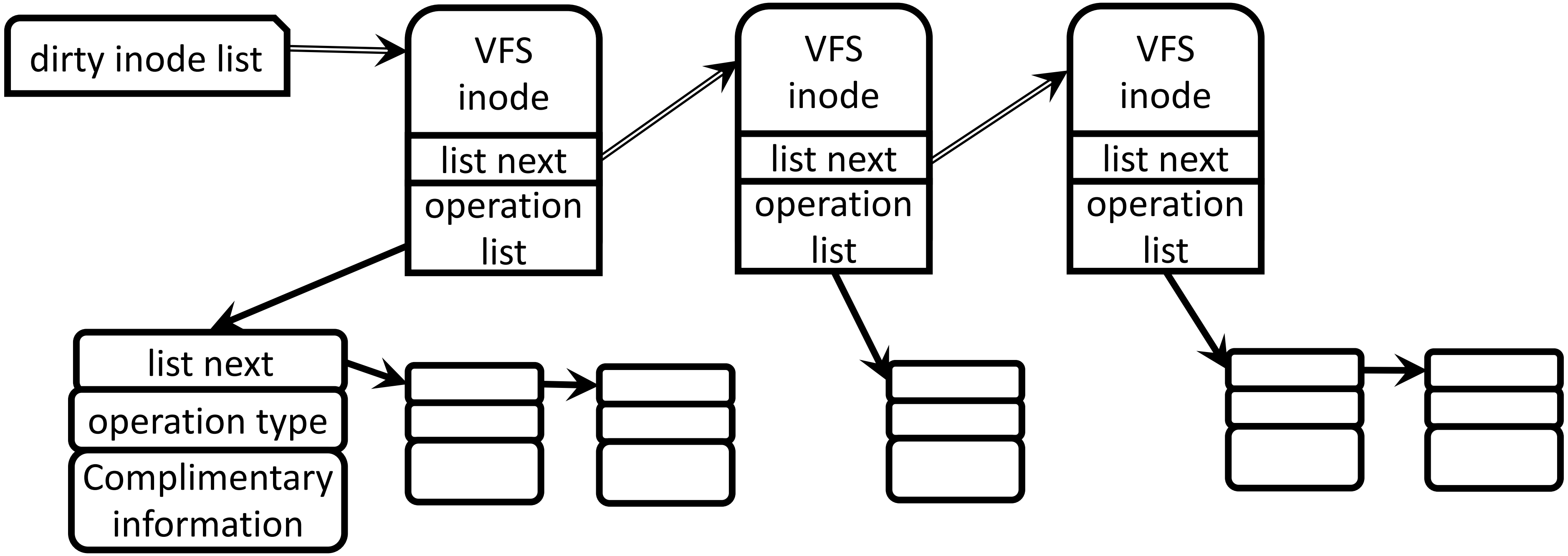
Operations are stored in operation list of each VFS inode



Semantic-aware Dependency Enforcement

Persister daemons traverse the dirty inode list in background

- persist each operation from the latest view to the consistent view with respect to update dependencies



Overview

Background

Design & Implementation

✓ Hashtable-based directories

✓ Pointer-based dual views

✓ Semantic-aware dependency tracking/enforcement

Evaluation

Conclusion

Evaluation Setup

Platform

- Intel Xeon E5 server with two 8-core processors
- 48 GB DRAM and 64 GB NVDIMM

File Systems

- SoupFS, PMFS, NOVA, Ext4-DAX, Ext4

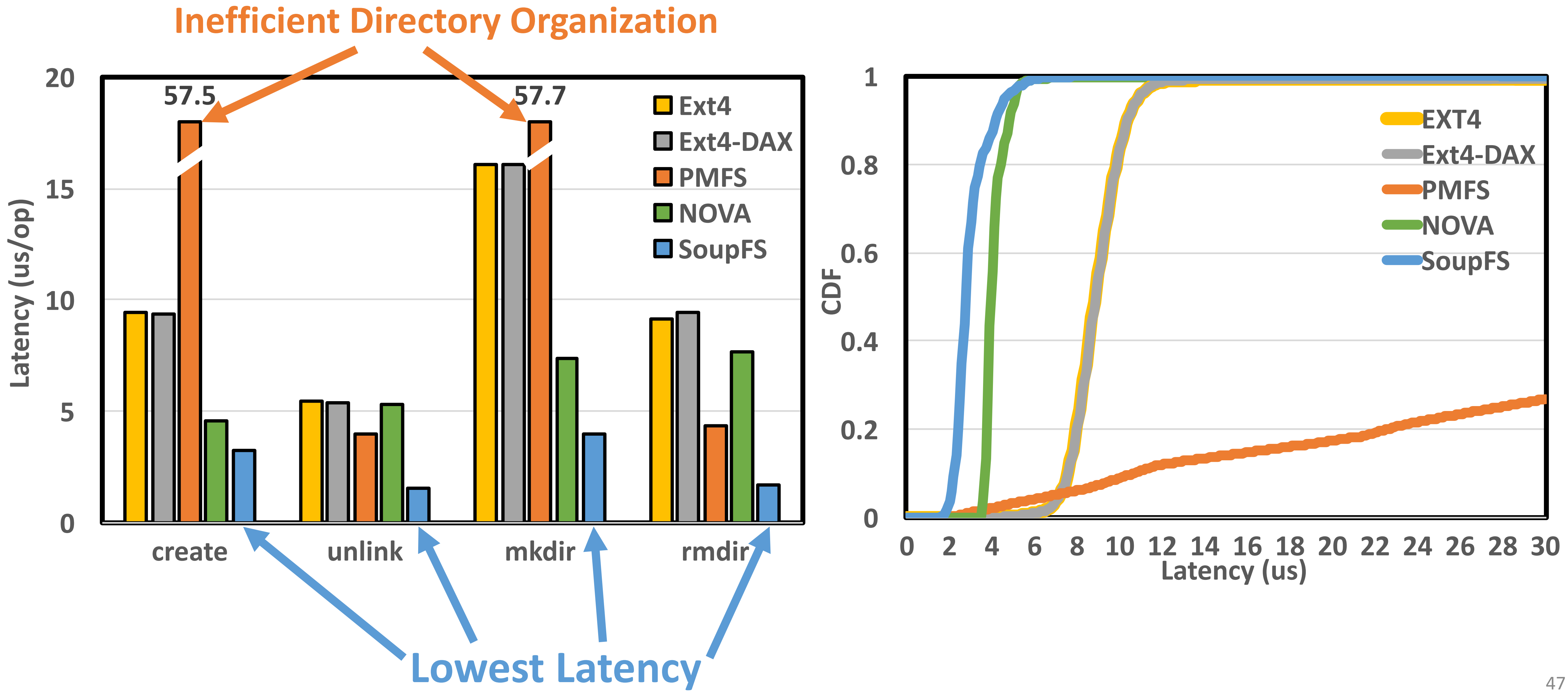
NVM Write Delay Simulation

- `ndelay()` after `clflush`

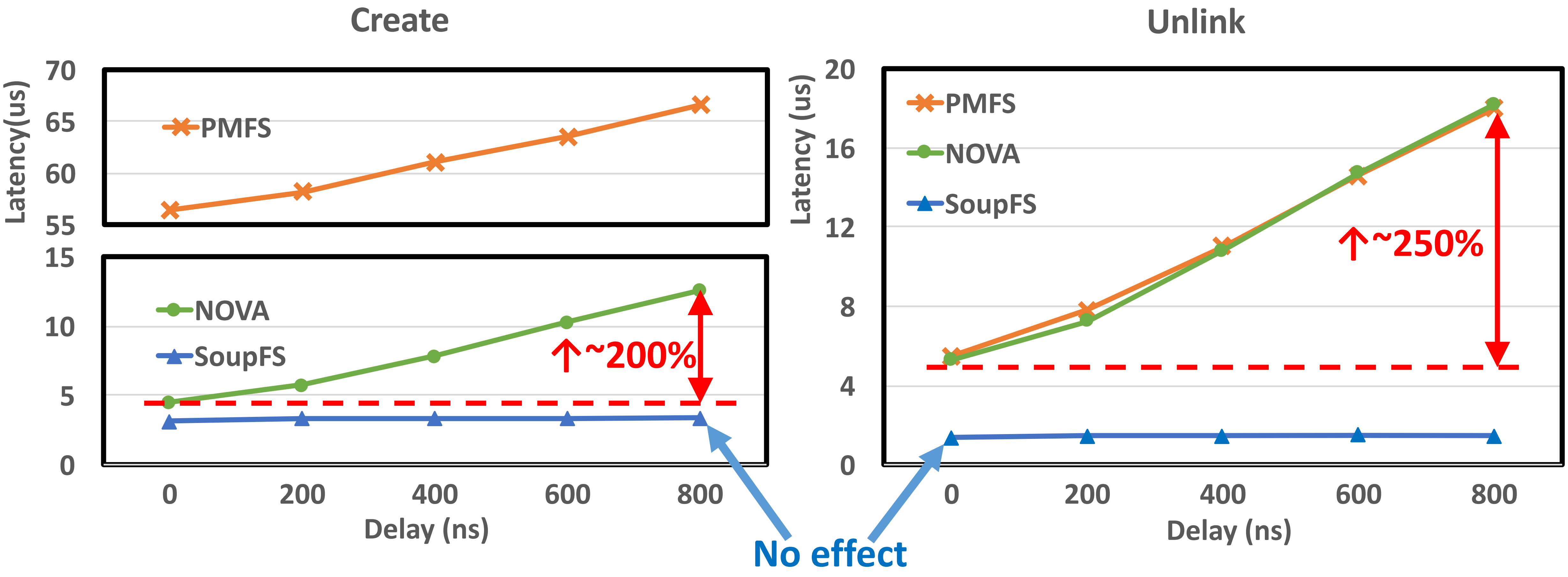
Benchmarks

- Micro-benchmarks: 100 iterations of 10^4 create/unlink/mkdir/rmdir
- Filebench and Postmark

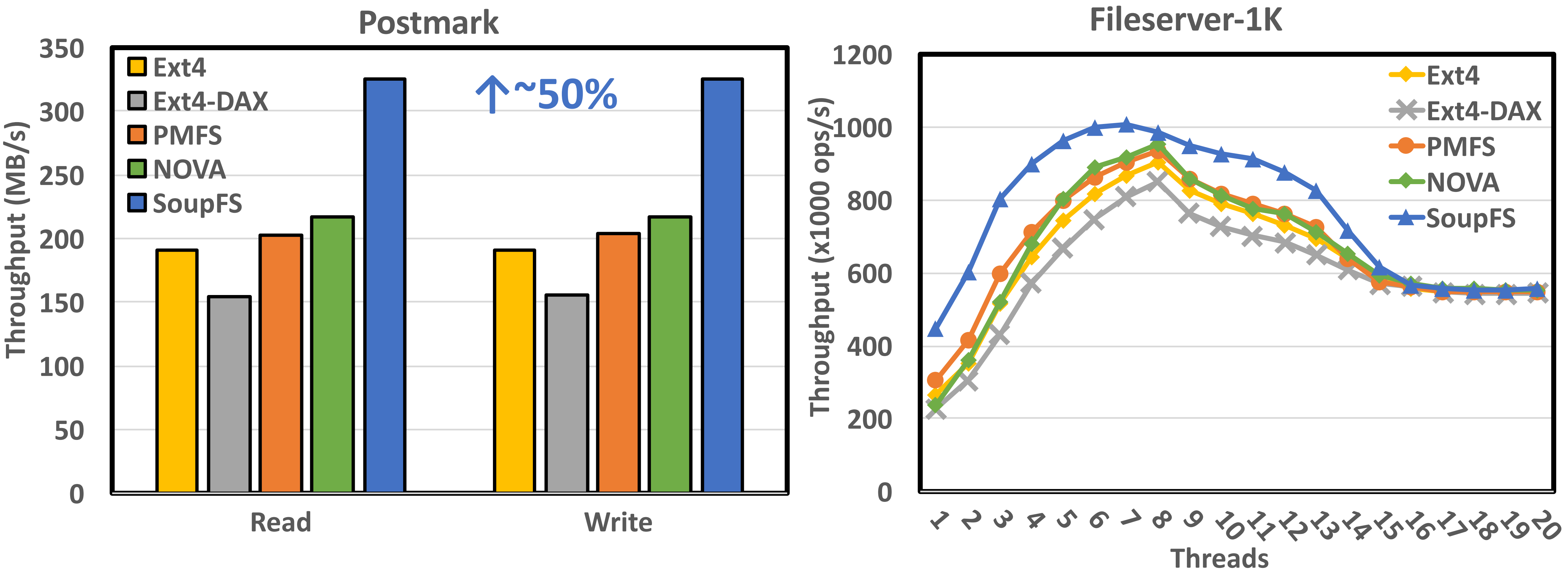
Micro-benchmark Latency



Sensitivity to NVM Write Delay



Postmark & Filebench



Overview

Background

Design & Implementation

✓ Hashtable-based directories

✓ Pointer-based dual views

✓ Semantic-aware dependency tracking/enforcement

Evaluation

Conclusion

Conclusion

- Soft updates is complicated due to the mismatch between *per-pointer-based dependency tracking* and *block-based interface of traditional disks*
- We design and implement SoupFS
  - ✓ Hashtable-based directories
  - ✓ Pointer-based dual views
  - ✓ Semantic-aware dependency tracking/enforcement
- Soft updates can be made simple and fast on NVM

Thanks & Questions? ;-)