Journaling versus Soft-Updates:
Asynchronous Meta-data Protection in File Systems

Margo I. Seltzer, Gregory R. Ganger, M. Kirk McKusick, Keith A. Smith,
Craig A. N. Soules, and Christopher A. Stein

USENIX Annual Technical Conference 2000

Overview

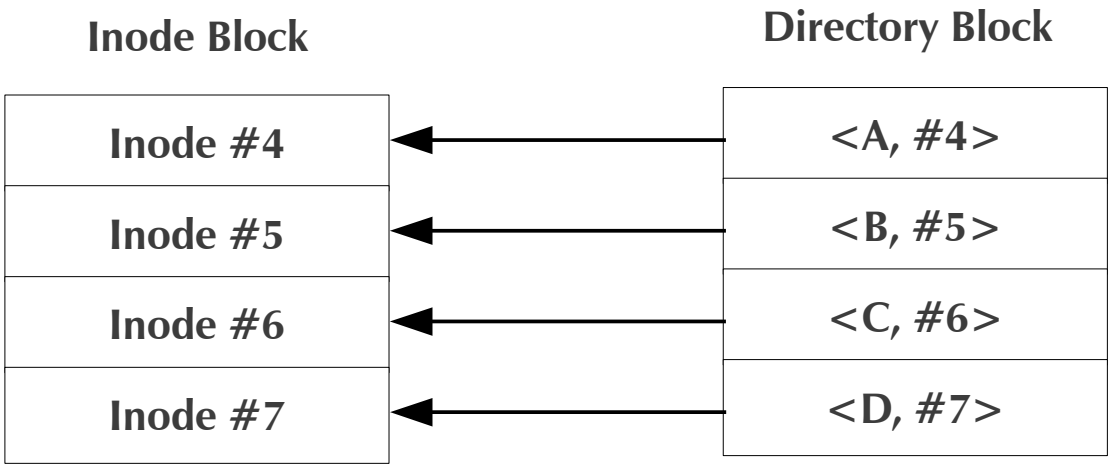
- **The problem**
  - *Filesystem integrity* and *metadata updates*
- **Synchronous metadata updates**
  - FFS: Works but poor run-time performance
- **Asynchronous approaches**
  - Journaling
  - Soft Updates
- **Compares two asynchronous approaches**

Metadata Updates Problem

- **Metadata operations**
  - Create, Delete, Rename
  - Modify the structure of the filesystem

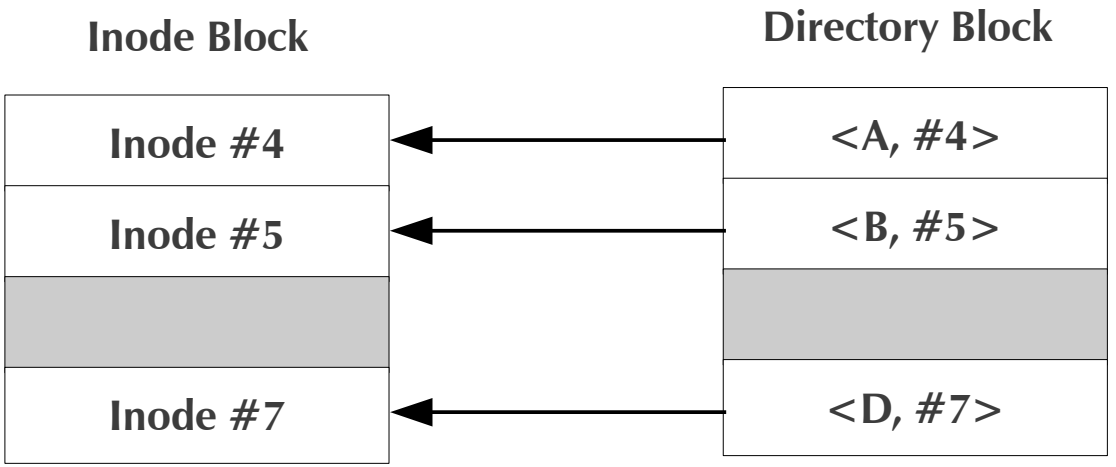
- **Filesystem integrity**
  - After a system crash, filesystem should be *recoverable to a consistent state* where it can continue to operate

Deleting a File



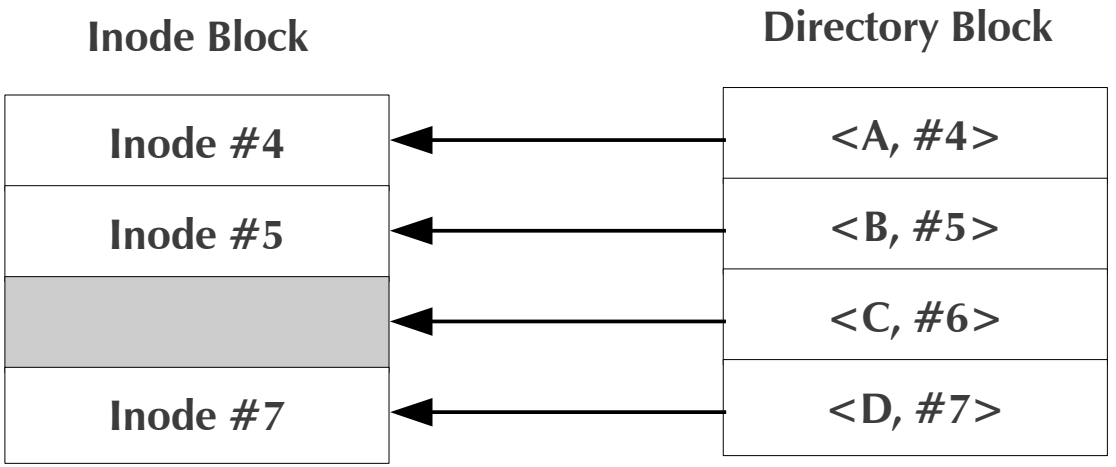
- Delete File C

Deleting a File



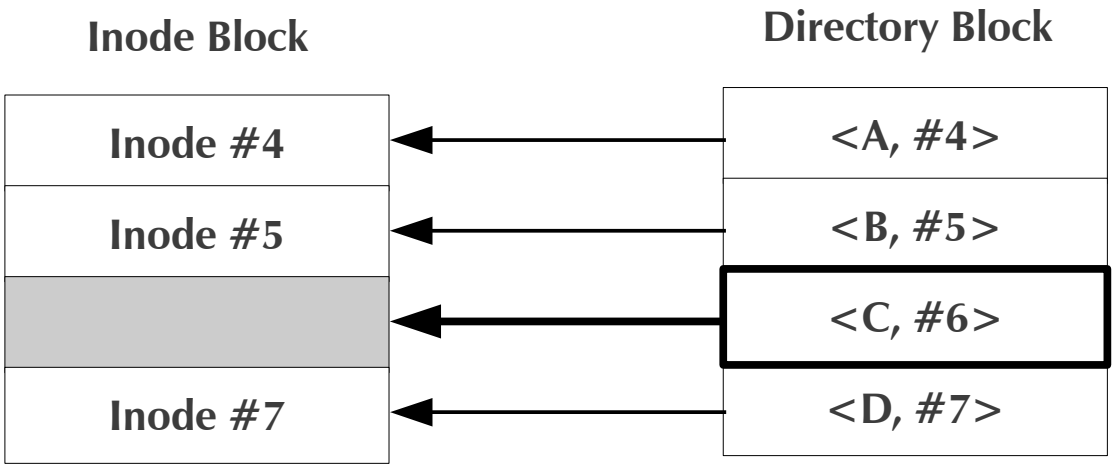
- **Delete File C**
  - Free inode entry and directory entry
  - 2 IOs involved
    - Modify inode block
    - Modify directory block
  - If we cannot do it atomically, ordering does matter!

Deleting a File



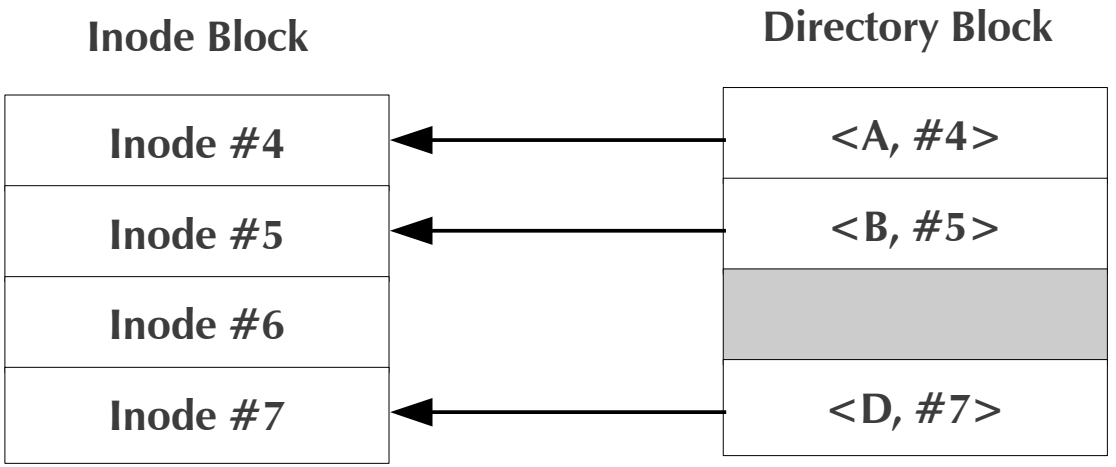
- **Delete File C**
  - Modify inode block first
  - System crash

Deleting a File



- **Delete File C**
  - Modify inode block first
  - System crash
  - Dentry <C, #6> points to a non-existing inode

Deleting a File



- **Delete File C**
  - Modify directory block first
  - After the crash, filesystem integrity is held
    - even though Inode #6 is lost (orphan resource)

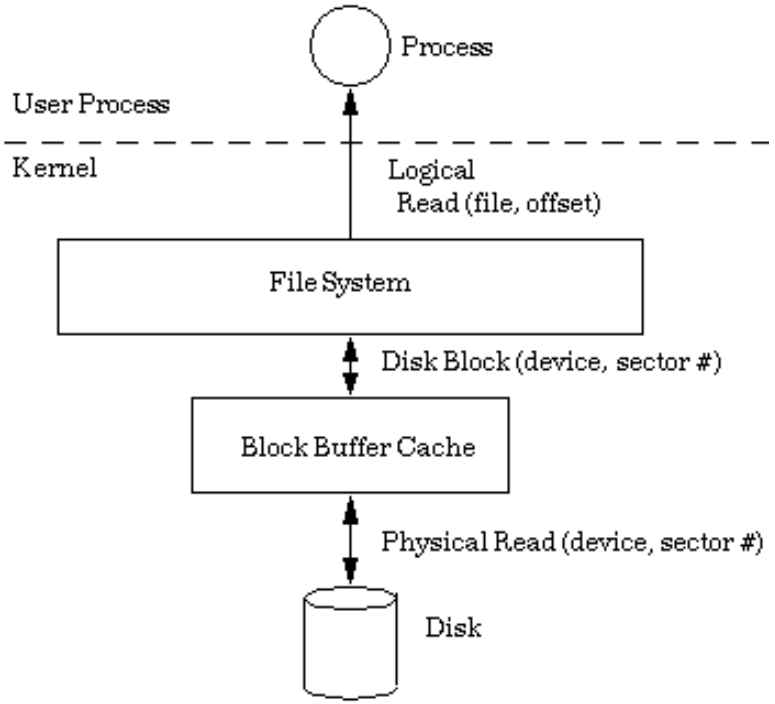
To keep the integrity

- **Metadata operations**
  - Do it atomically
  - Follow the ordering constraints

- **Ordering constraints**
  - Deleting a file
    - Delete the Directory entry
    - Deallocate the inode
    - Deallocate the data blocks
  - Creating a file
    - Deallocate the data blocks
    - Deallocate the inode
    - Delete the directory entry

What makes it complicated?

- **Multiple blocks are involved in a single logical operation**
  - And usually they are scattered on a disk
- **OS buffer cache**
  - Most update operations are asynchronous/delayed
  - Actual IO ordering is done by VM/Disk scheduler



Naive Approach - FFS

- **Enforce the ordering constraints, *synchronously***
  - Before the system call returns; the related metadata blocks are written synchronously in a correct order
  - Drawbacks
    - Poor runtime performance

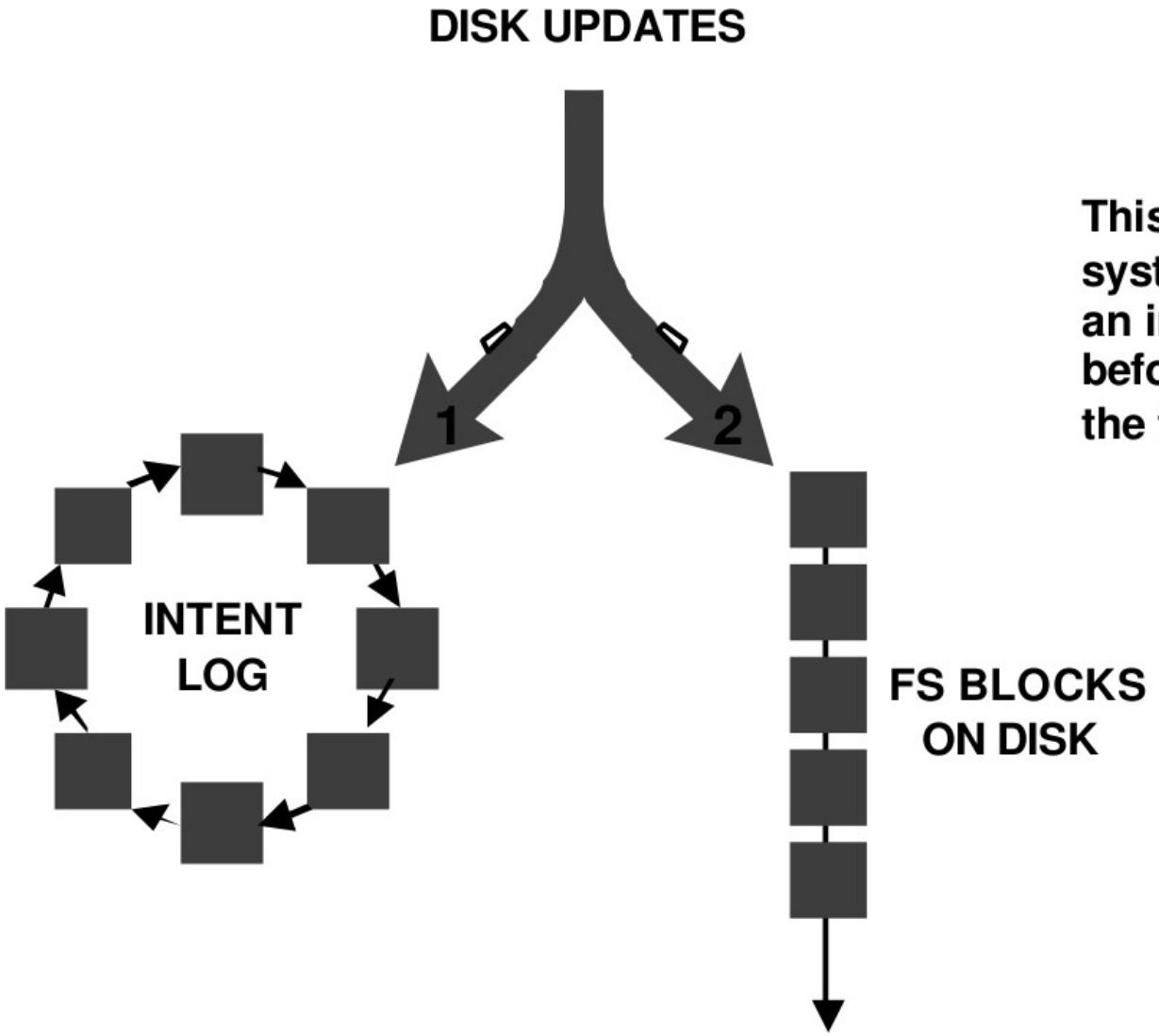
Asynchronous Approaches

- **Journaling**
  - Guarantees *atomicity* in metadata operations
- **Soft Updates**
  - Enforce the ordering constraints, in an *asynchronously* way

Asynchronous Solutions - Journaling

- **Write ahead logging**
  - Write changes to metadata in the journal
  - Blocks are written to disk *only after associated journal data has been committed*
  - On recovery, just replay(Roll Forward) for committed journal records
  - Atomic metadata operations

Asynchronous Solutions - Journaling



This journals file system updates to an intent log (1) before modifying the file itself (2).

Asynchronous Solutions - Journaling

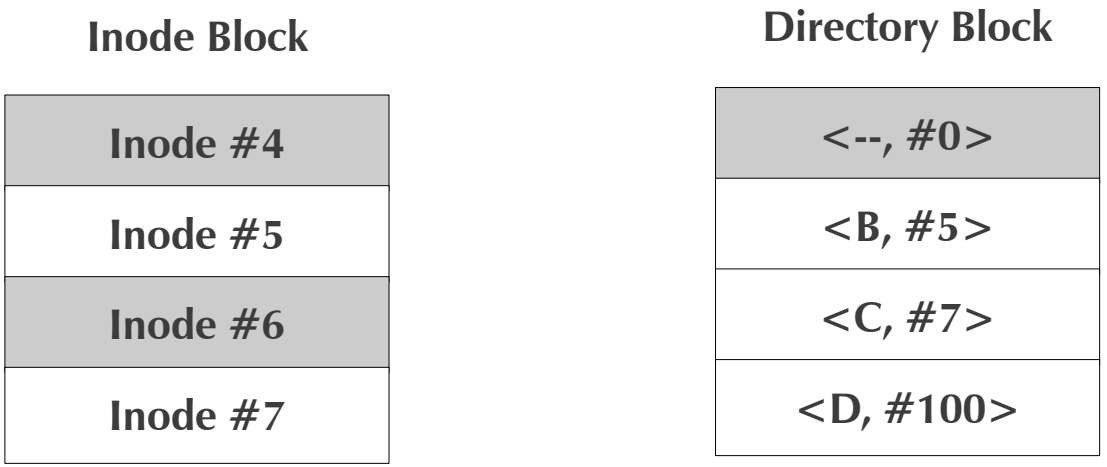
- **Benefits**
  - Quick recovery (`fsck`)
- **Drawbacks**
  - Extra IO generated

Asynchronous Solutions - Soft Updates

- **Enforce the ordering constraints, *asynchronously***
  - Maintain dirty blocks and relationships/dependencies to each other
  - Let VM sync any disk blocks
  - When a block is written by VM, soft update code can take care of the dependencies

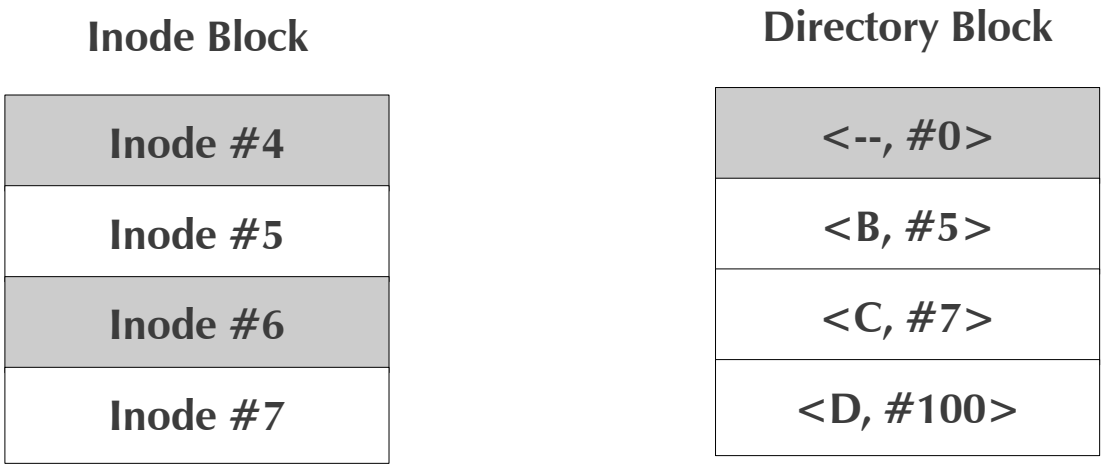
  - Dependency information
    - Block basis - cyclic dependencies problem
    - Pointer basis

Soft Updates - Cyclic Dependencies



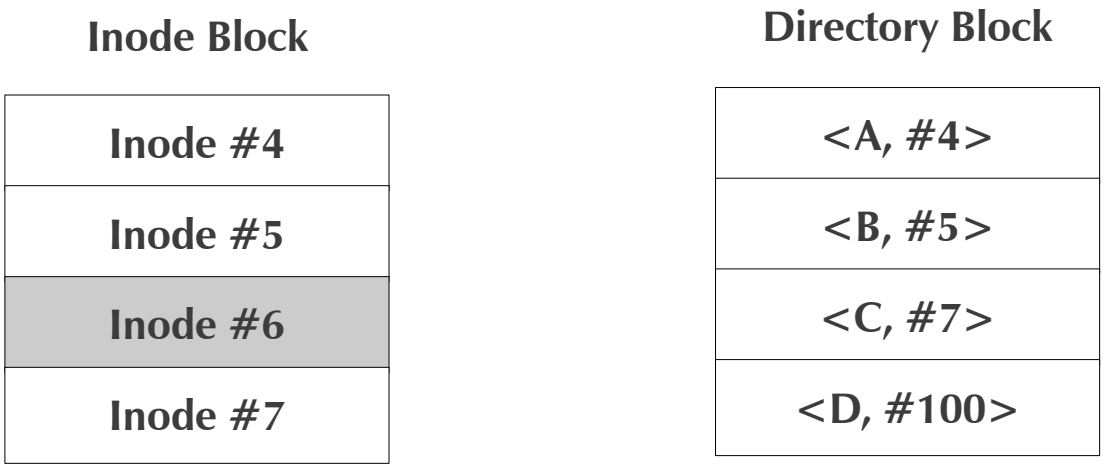
- Block-basis dependency

Soft Updates - Cyclic Dependencies



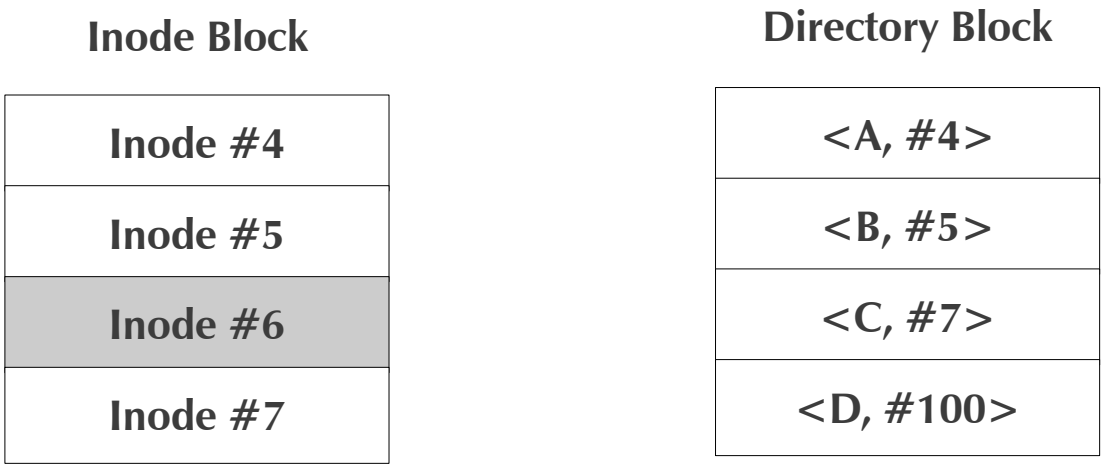
- **Block-basis dependency**
  - Create file <A, #4>

Soft Updates - Cyclic Dependencies



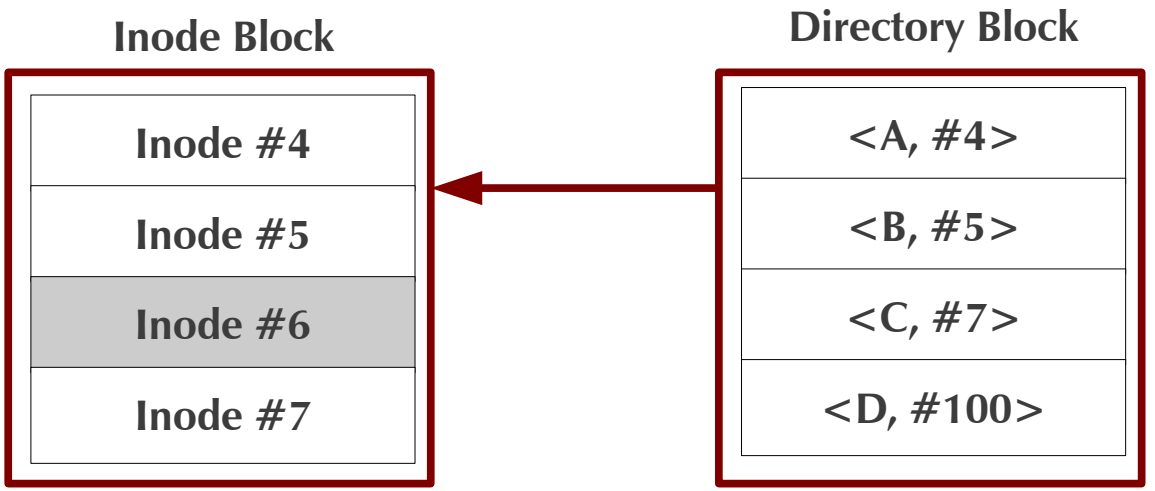
- **Block-basis dependency**
  - Create file <A, #4>

Soft Updates - Cyclic Dependencies



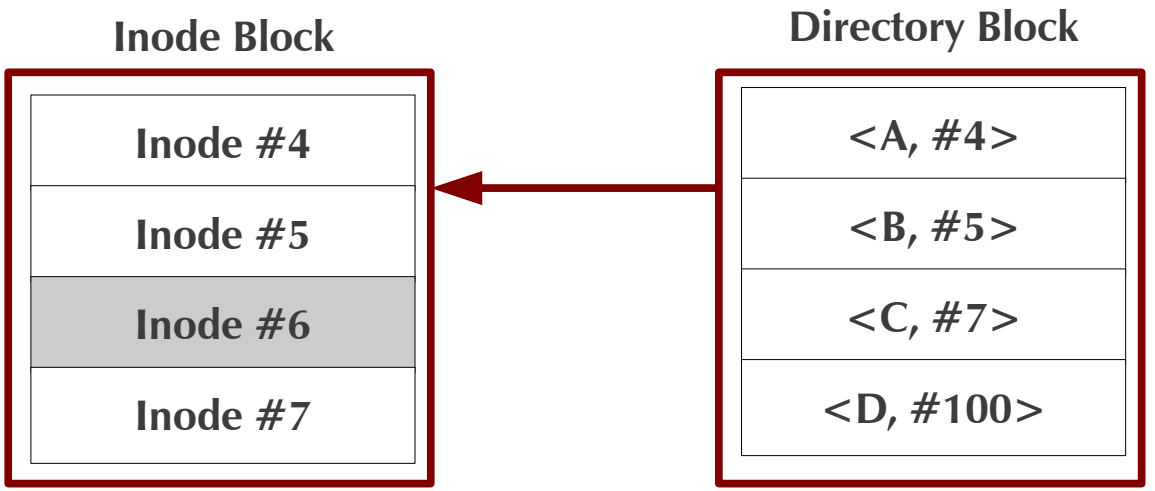
- **Block-basis dependency**
  - Create file <A, #4>
    - Correct order: Inode Block → Directory Block

Soft Updates - Cyclic Dependencies



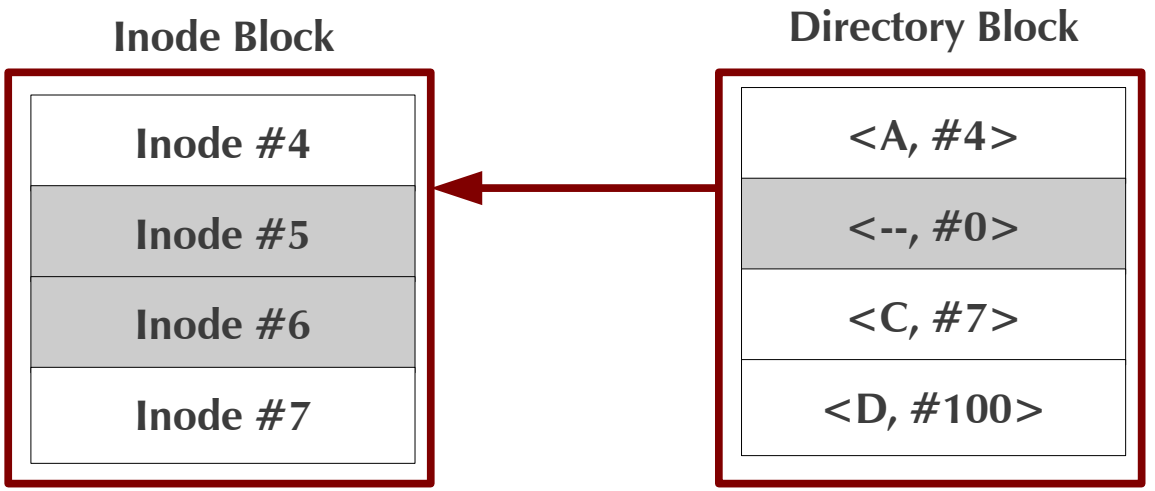
- **Block-basis dependency**
  - Create file <A, #4>
    - Correct order: Inode Block → Directory Block

Soft Updates - Cyclic Dependencies



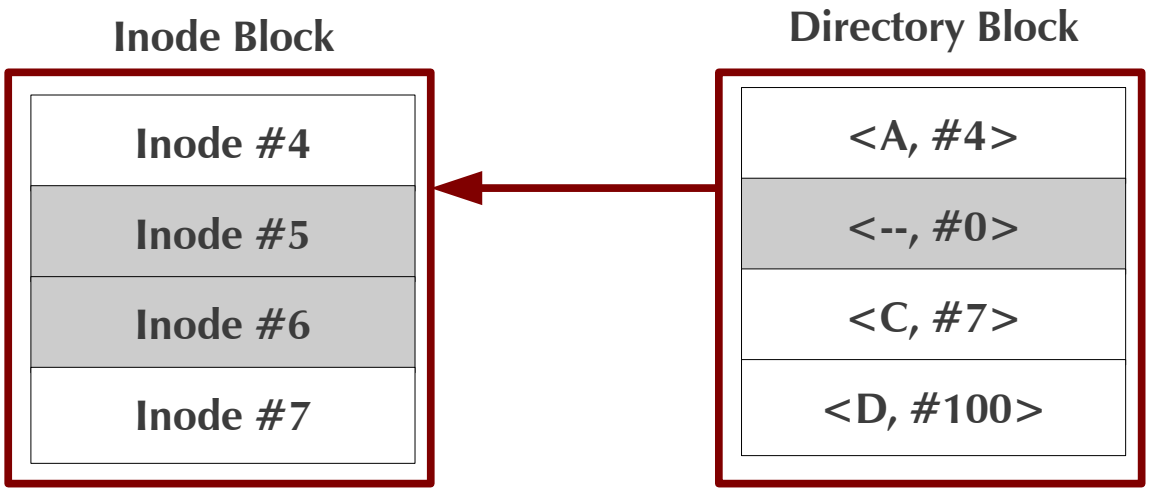
- **Block-basis dependency**
  - Create file <A, #4>
    - Correct order: Inode Block → Directory Block
  - Delete file <B, #5>

Soft Updates - Cyclic Dependencies



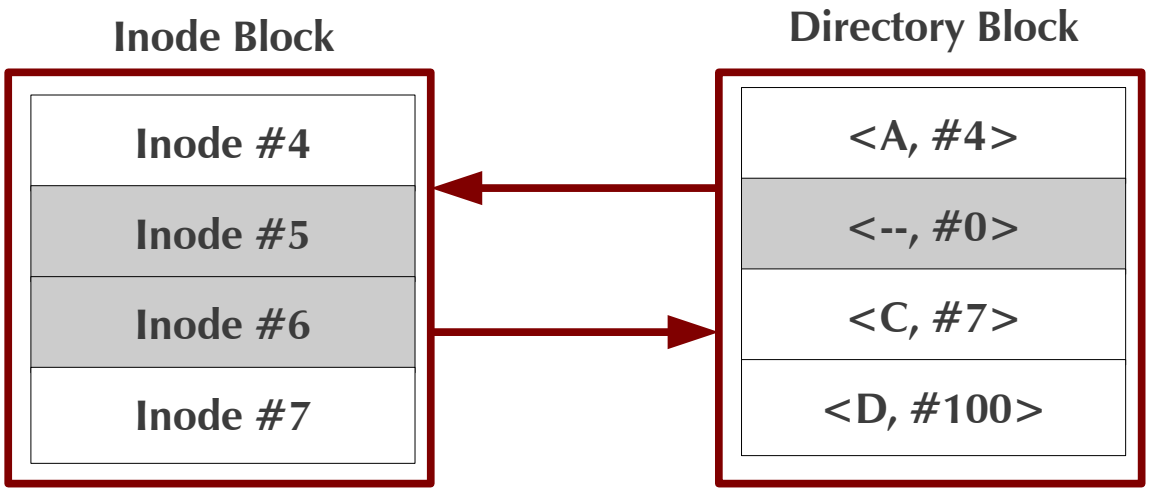
- **Block-basis dependency**
  - Create file <A, #4>
    - Correct order: Inode Block → Directory Block
  - Delete file <B, #5>

Soft Updates - Cyclic Dependencies



- **Block-basis dependency**
  - Create file <A, #4>
    - Correct order: Inode Block → Directory Block
  - Delete file <B, #5>
    - Correct order: Directory Block → Inode Block

Soft Updates - Cyclic Dependencies



Cyclic Dependencies

- **Block-basis dependency**
  - Create file <A, #4>
    - Correct order: Inode Block → Directory Block
  - Delete file <B, #5>
    - Correct order: Directory Block → Inode Block

Asynchronous Solutions - Soft Updates

Inode Block

Inode #4
Inode #5
Inode #6
Inode #7

Directory Block

<--, #0>
<B, #5>
<C, #7>
<D, #100>

Buffer Cache

Disk

Inode Block

Inode #4
Inode #5
Inode #6
Inode #7

Directory Block

<--, #0>
<B, #5>
<C, #7>
<D, #100>

Asynchronous Solutions - Soft Updates

Inode Block

Inode #4
Inode #5
Inode #6
Inode #7

Directory Block

<--, #0>
<B, #5>
<C, #7>
<D, #100>

1. Create File A

Buffer Cache

Disk

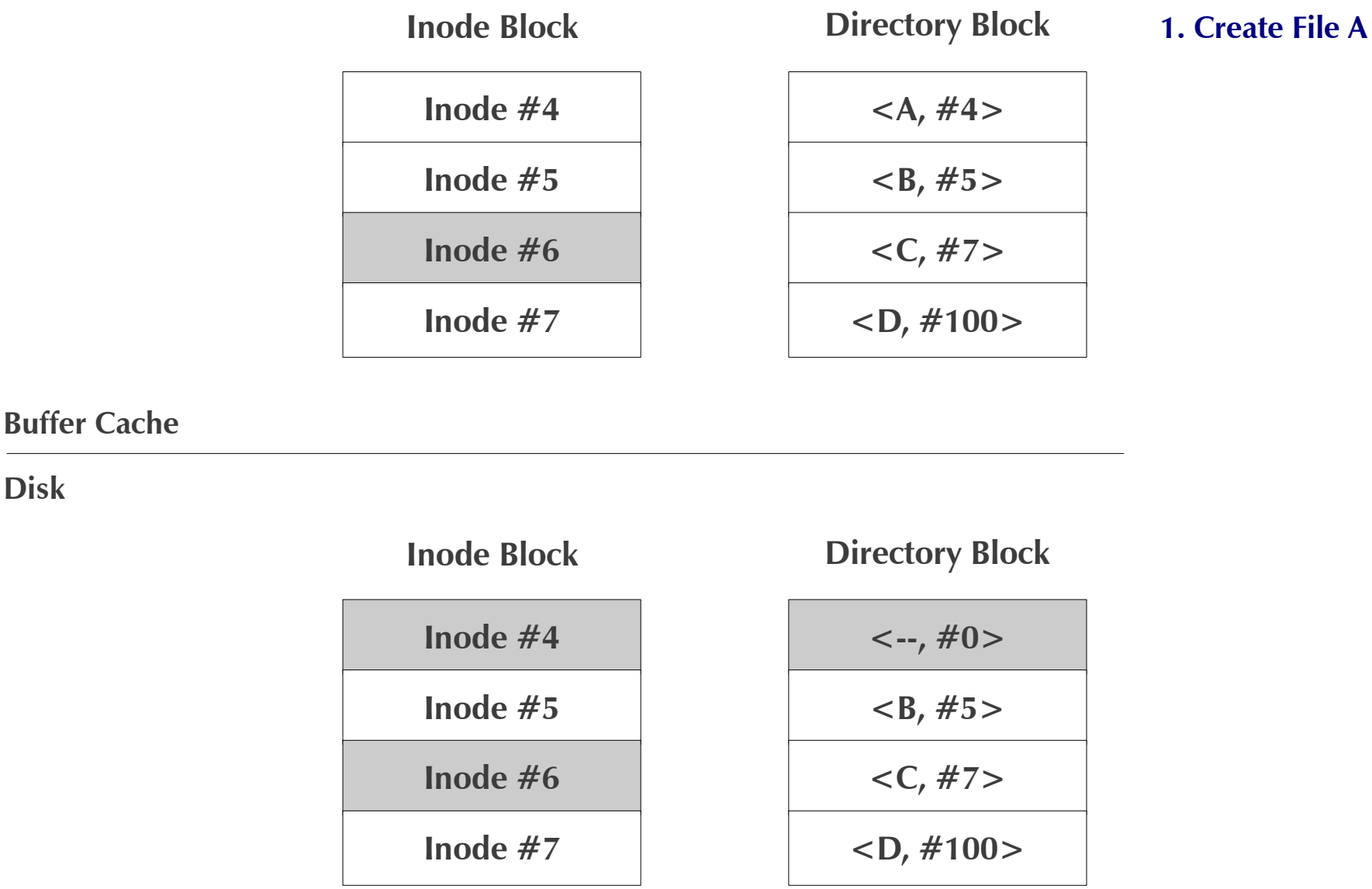
Inode Block

Inode #4
Inode #5
Inode #6
Inode #7

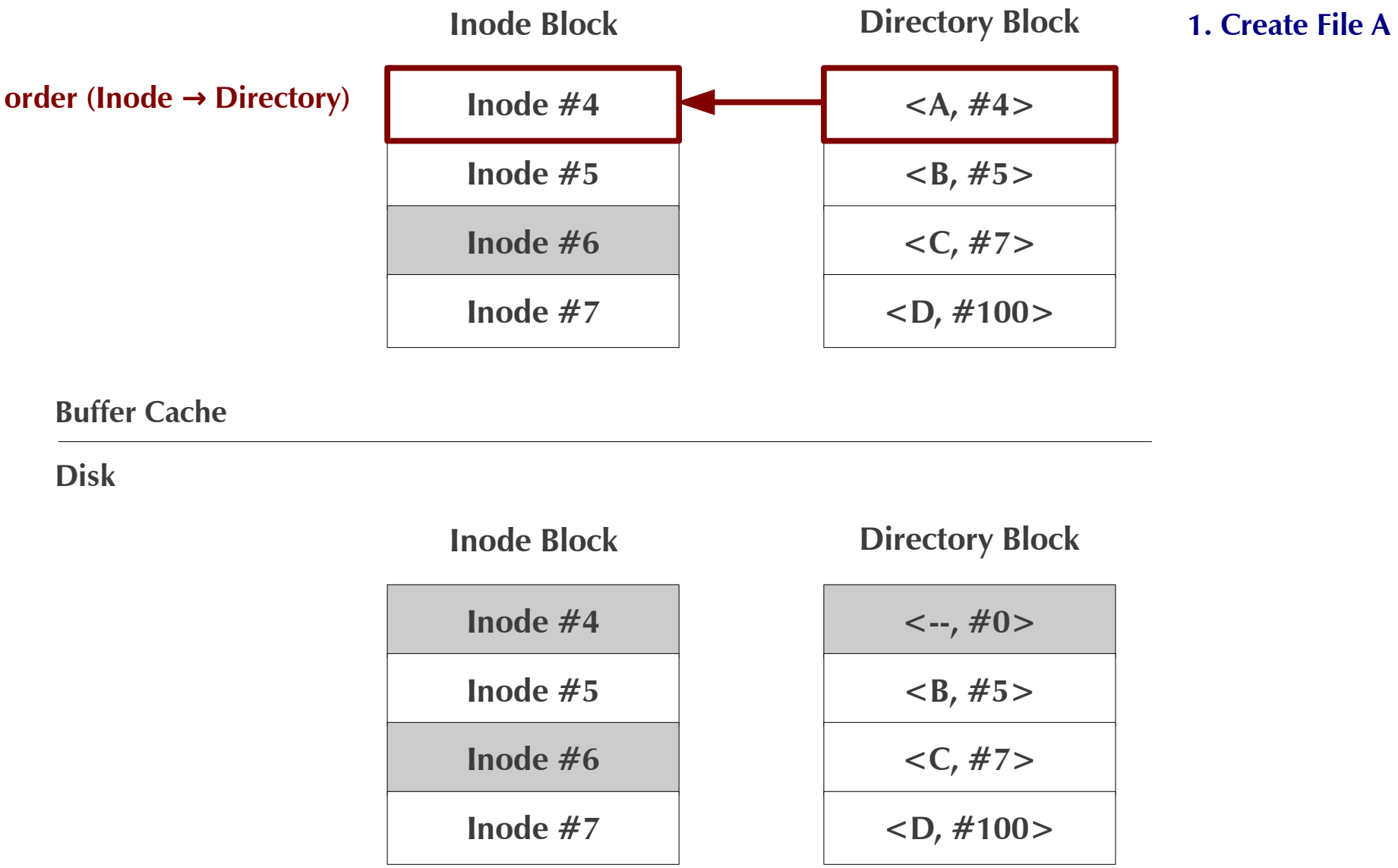
Directory Block

<--, #0>
<B, #5>
<C, #7>
<D, #100>

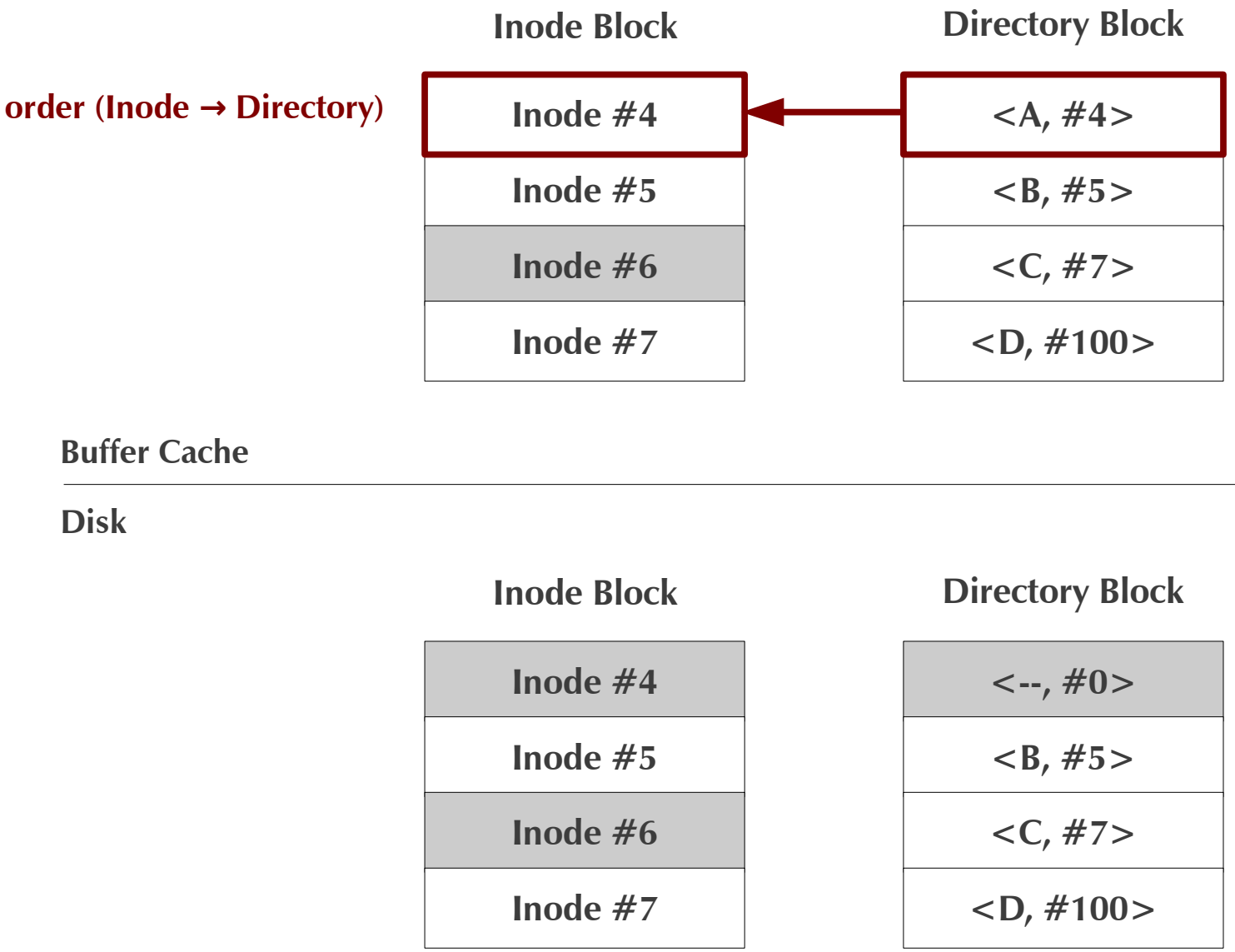
Asynchronous Solutions - Soft Updates



Asynchronous Solutions - Soft Updates

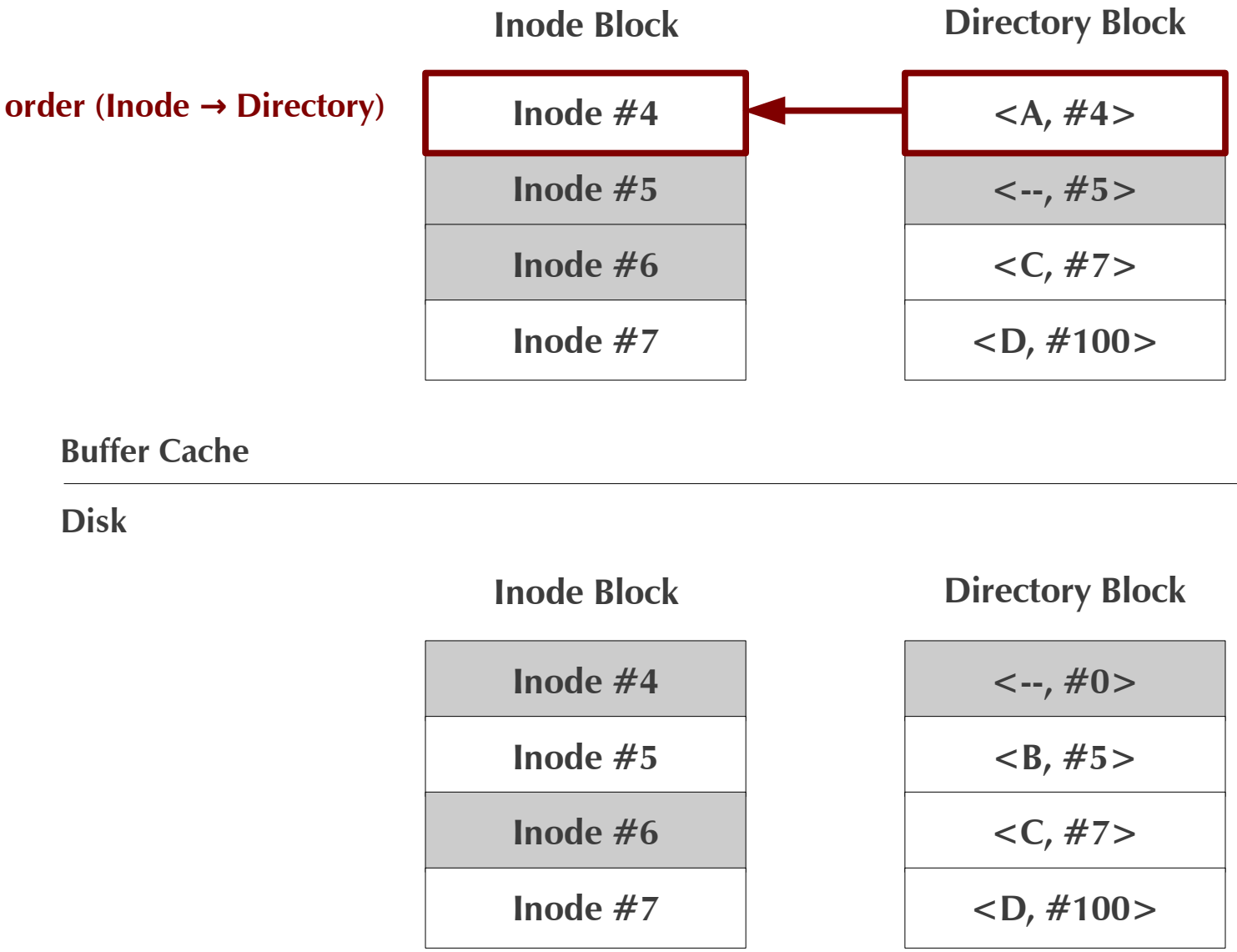


Asynchronous Solutions - Soft Updates



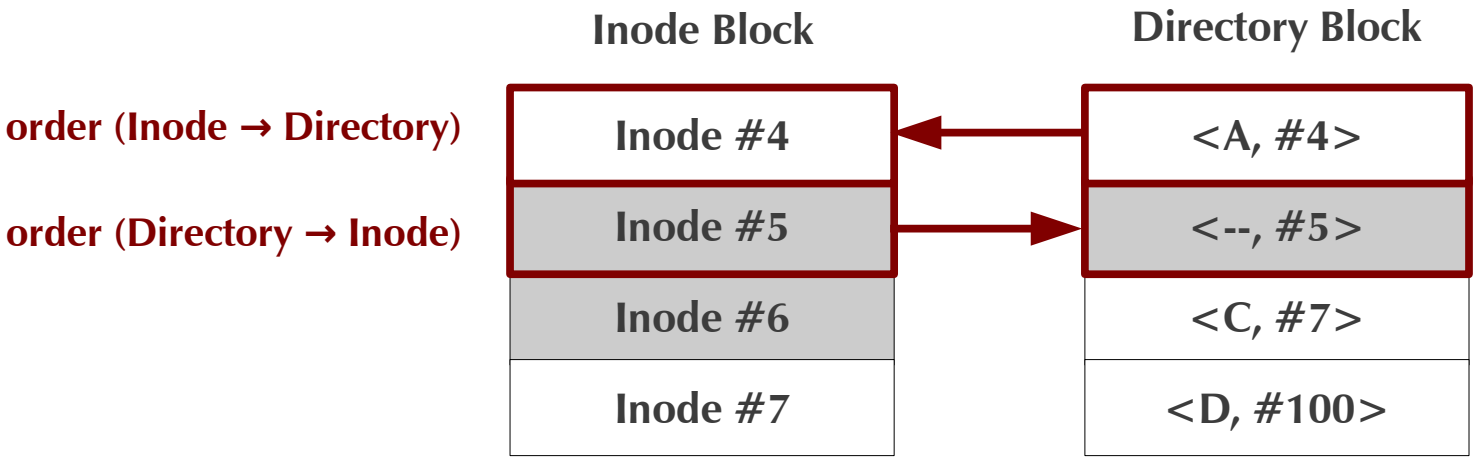
1. Create File A
2. Delete File B

Asynchronous Solutions - Soft Updates



1. Create File A
2. Delete File B

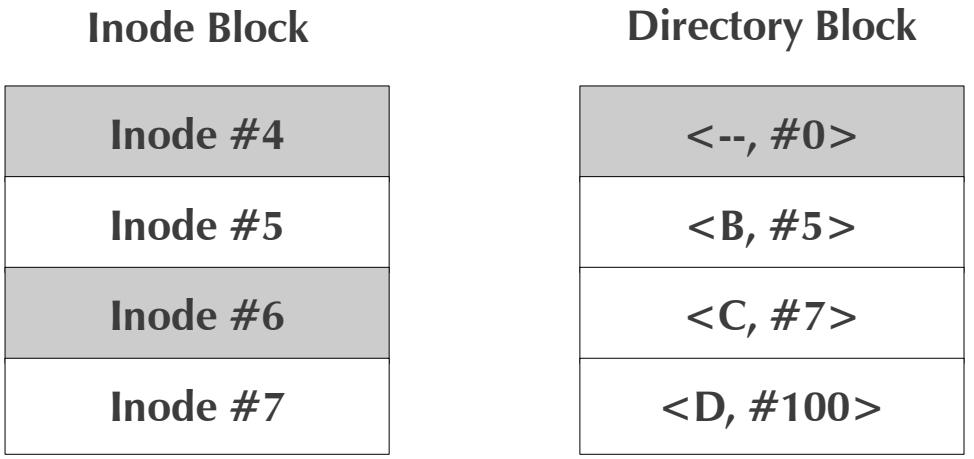
Asynchronous Solutions - Soft Updates



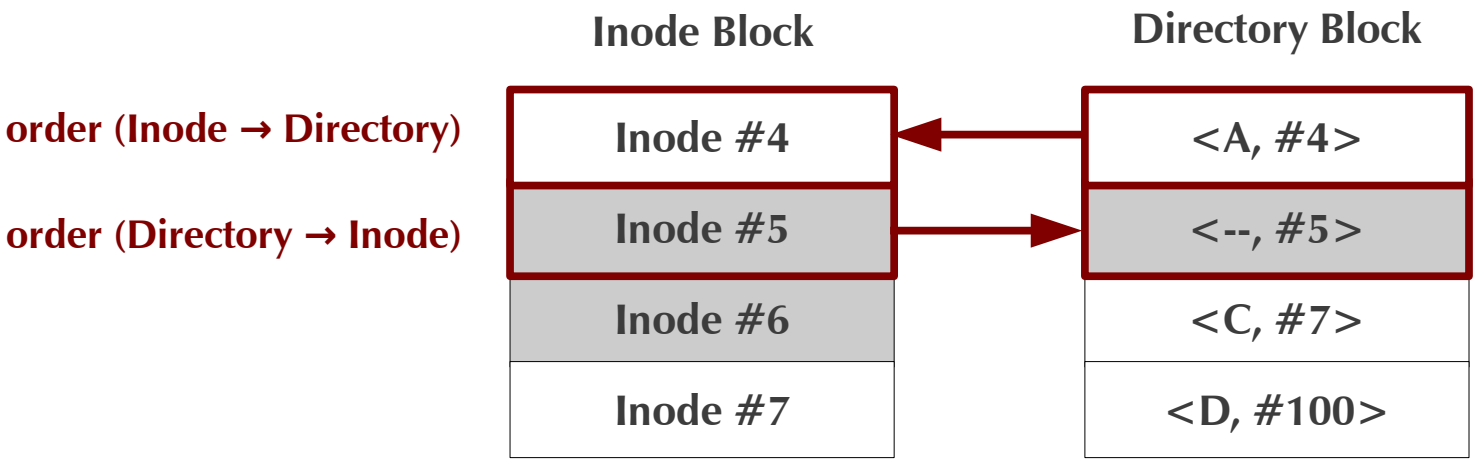
1. Create File A
2. Delete File B

Buffer Cache

Disk



Asynchronous Solutions - Soft Updates

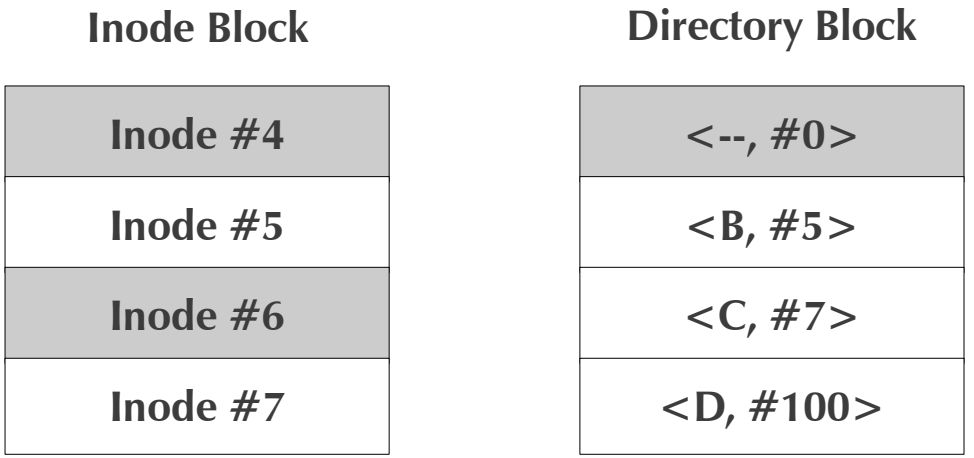


1. Create File A
2. Delete File B

3. VM - notify Soft Updates
   Sync(DirBlock)

Buffer Cache

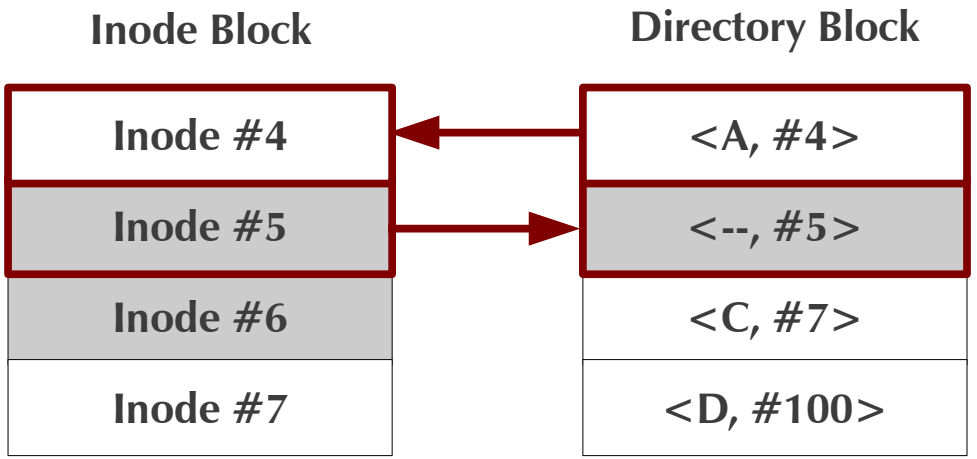
Disk



Asynchronous Solutions - Soft Updates

order (Inode → Directory)

order (Directory → Inode)

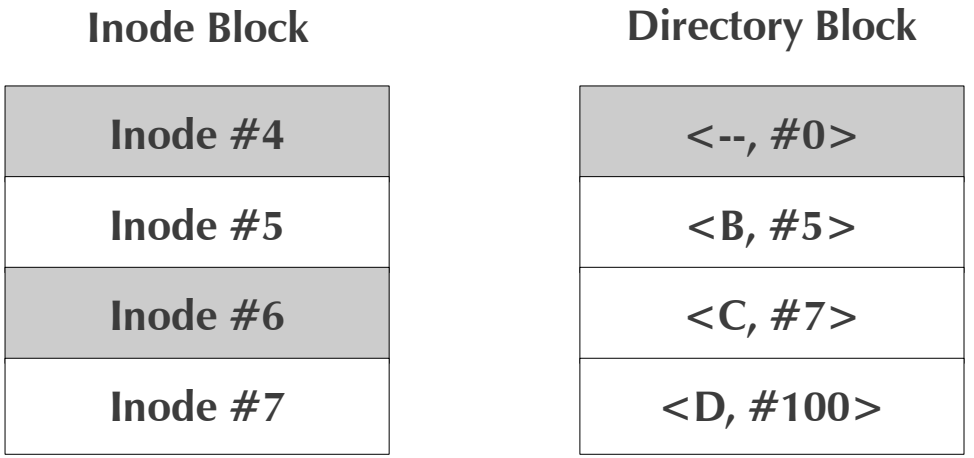


1. Create File A
2. Delete File B

3. VM - notify Soft Updates
   Sync(DirBlock)
4. **Soft Updates**
   CheckDep(DirBlock)
   → <A, #4> not ready

Buffer Cache

Disk

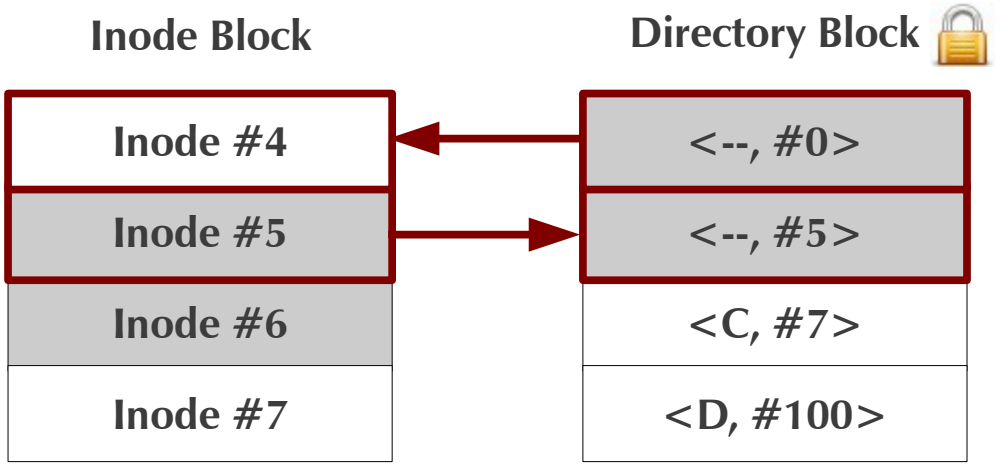


Asynchronous Solutions - Soft Updates

<A, #4>

order (Inode → Directory)

order (Directory → Inode)

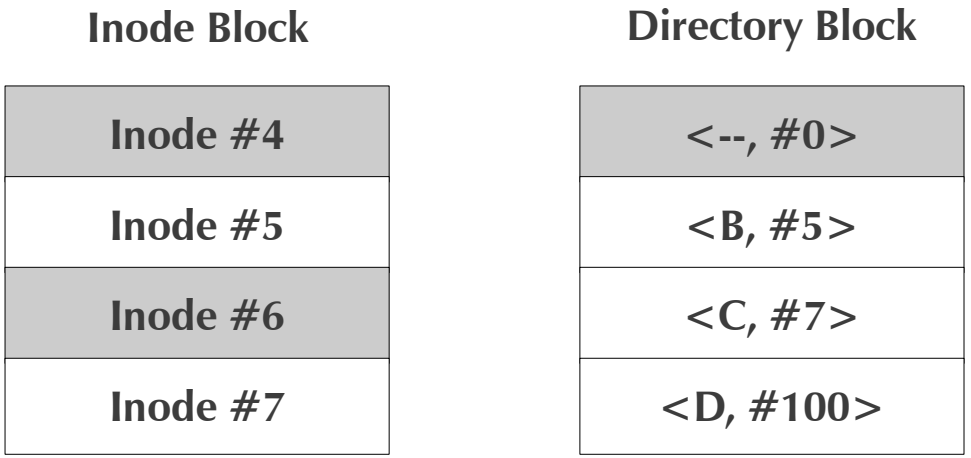


1. Create File A
2. Delete File B

3. VM - notify Soft Updates
   Sync(DirBlock)
4. Soft Updates
   CheckDep(DirBlock)
   → <A, #4> not ready
   Lock(DirBlock)
   Roll back <A, #4>

Buffer Cache

Disk

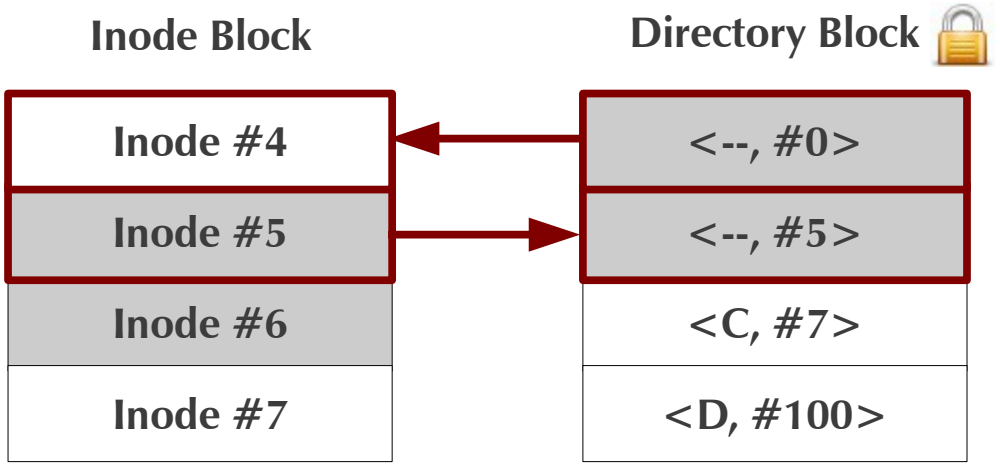


Asynchronous Solutions - Soft Updates

<A, #4>

order (Inode → Directory)

order (Directory → Inode)

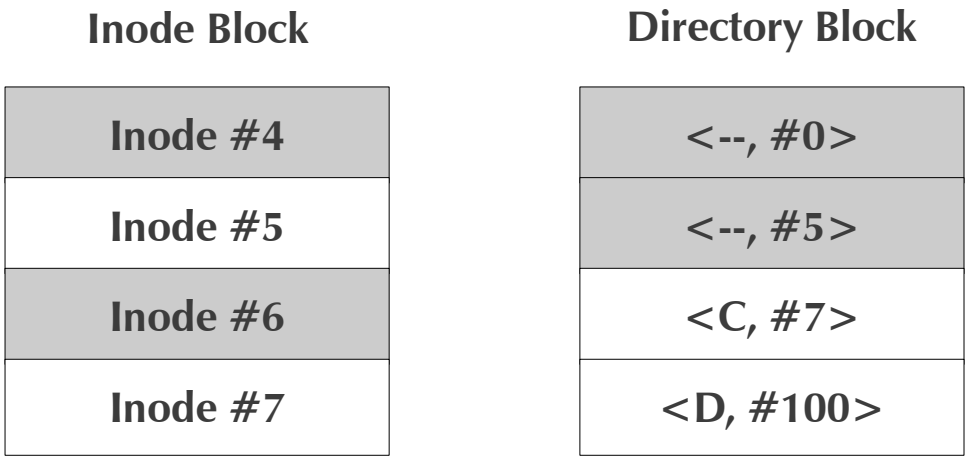


1. Create File A
2. Delete File B

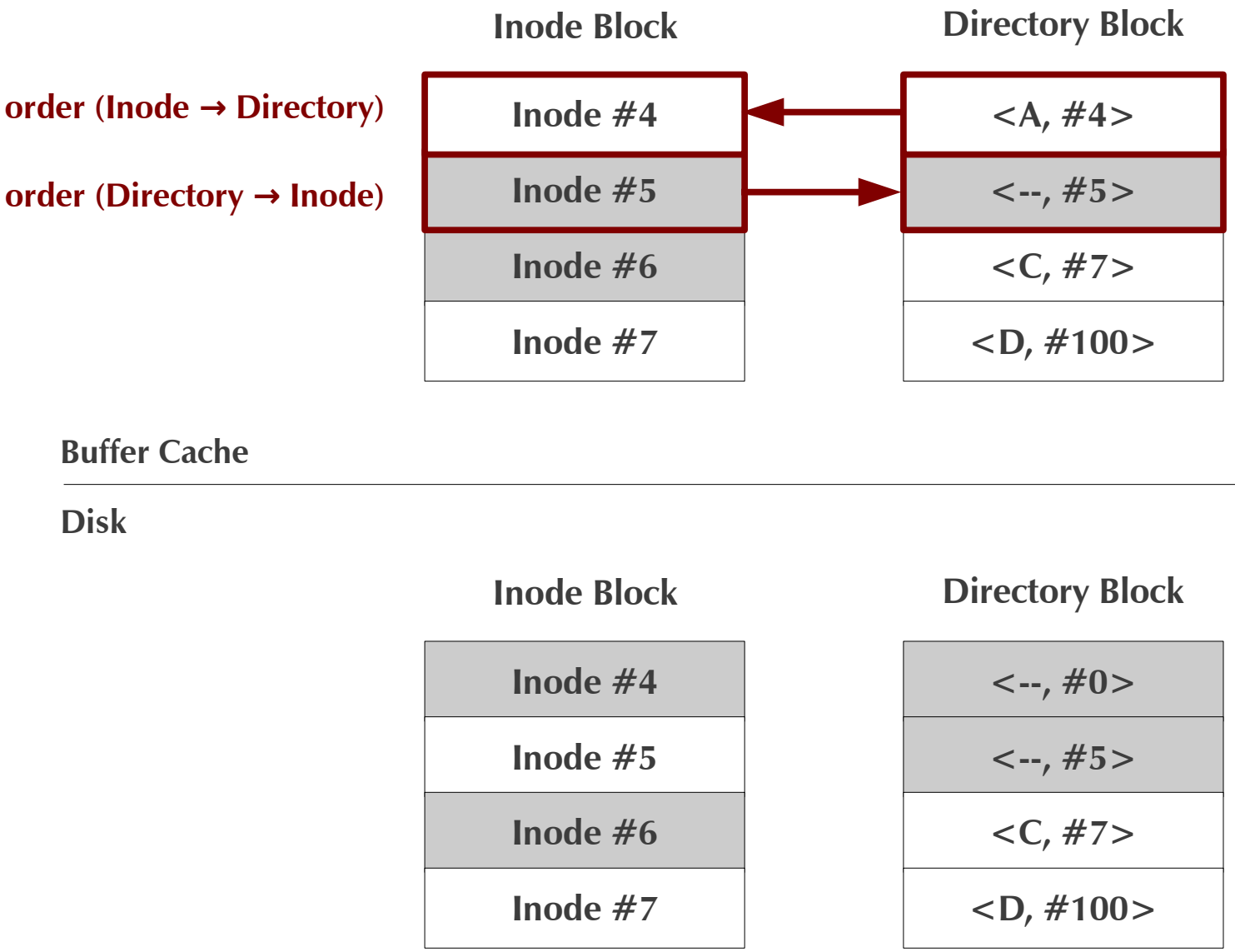
3. VM - notify Soft Updates
   Sync(DirBlock)
4. Soft Updates
   CheckDep(DirBlock)
   → <A, #4> not ready
   Lock(DirBlock)
   Roll back <A, #4>
5. VM - Sync(DirBlock)

Buffer Cache

Disk



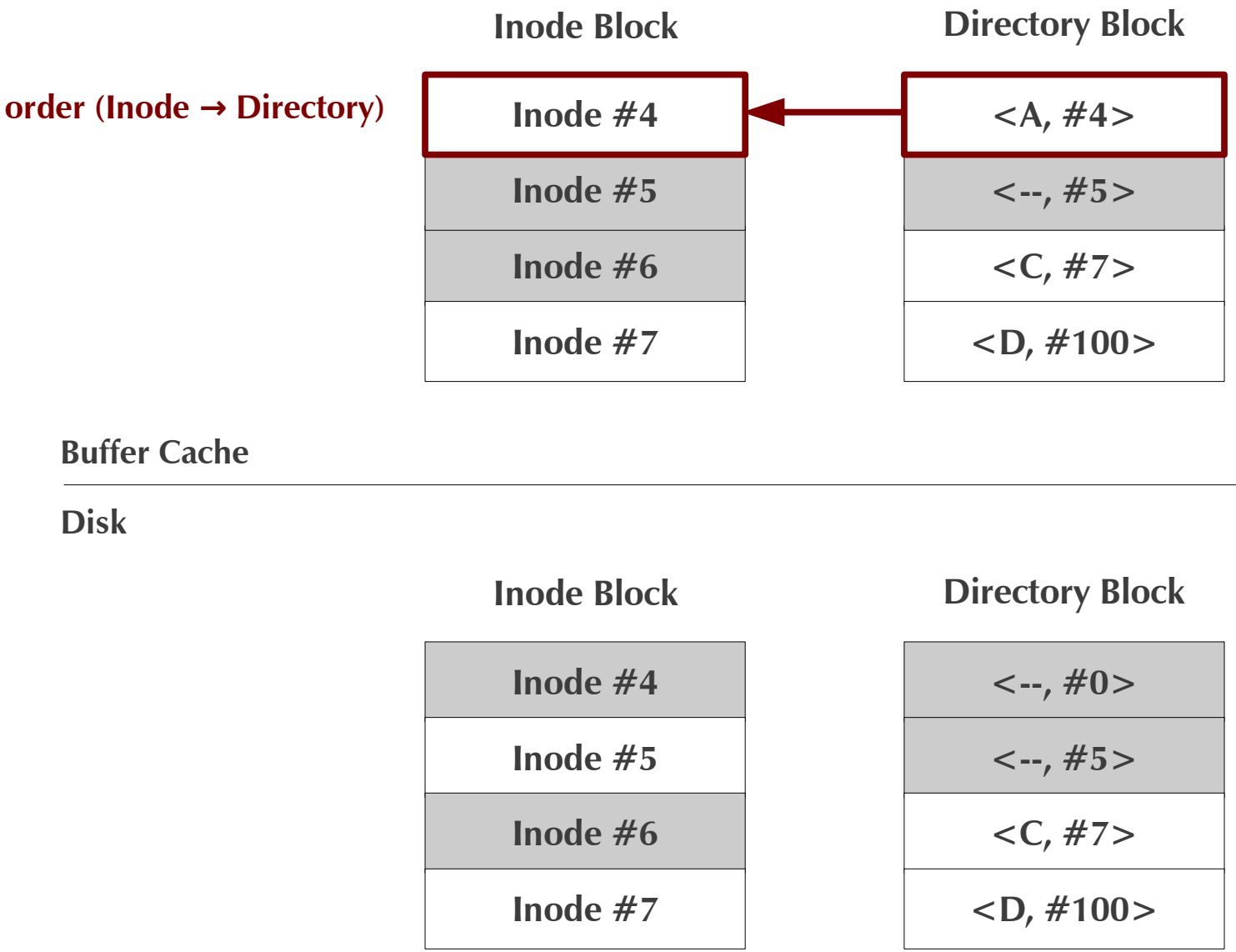
Asynchronous Solutions - Soft Updates



1. Create File A
2. Delete File B

3. VM - notify Soft Updates
   Sync(DirBlock)
4. Soft Updates
   CheckDep(DirBlock)
   → <A, #4> not ready
   Lock(DirBlock)
   Roll back <A, #4>
5. VM - Sync(DirBlock)
6. **Soft Updates**
   Roll forward <A, #4>
   UnLock(DirBlock)

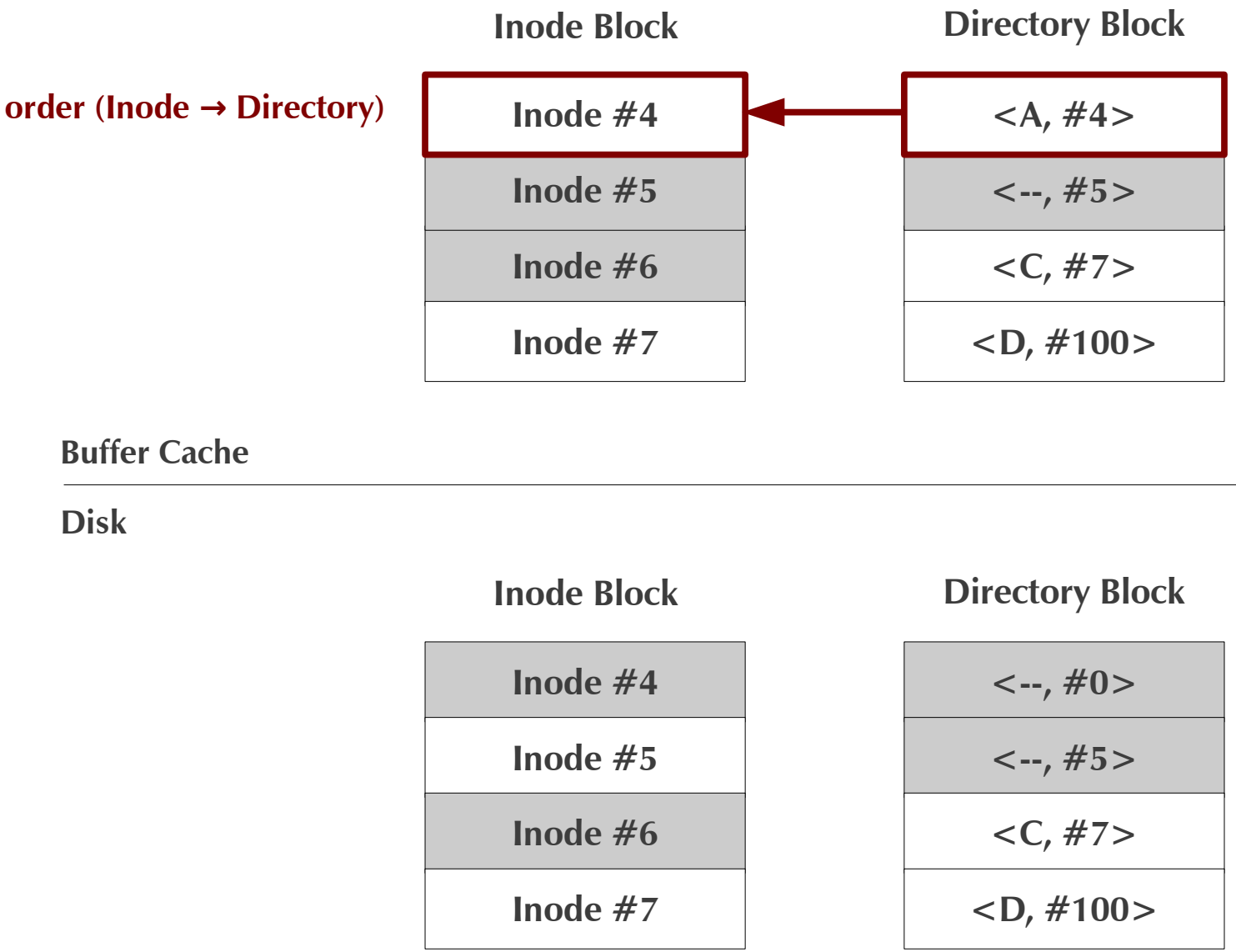
Asynchronous Solutions - Soft Updates



1. Create File A
2. Delete File B

3. VM - notify Soft Updates
   Sync(DirBlock)
4. Soft Updates
   CheckDep(DirBlock)
   → <A, #4> not ready
   Lock(DirBlock)
   Roll back <A, #4>
5. VM - Sync(DirBlock)
6. Soft Updates
   Roll forward <A, #4>
   UnLock(DirBlock)
   RemoveDep(DirBlock)

Asynchronous Solutions - Soft Updates

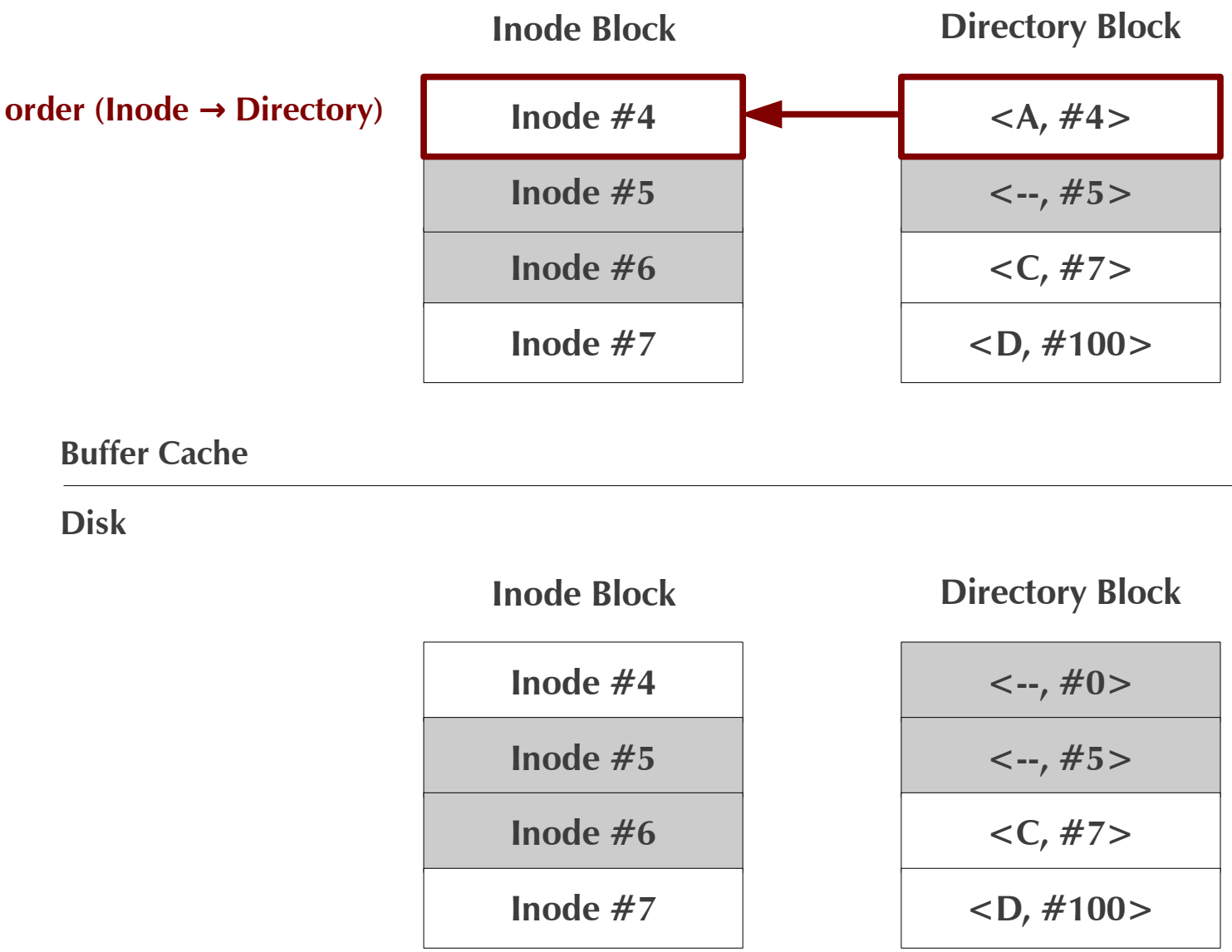


1. Create File A
2. Delete File B

3. VM - notify Soft Updates
   Sync(DirBlock)
4. Soft Updates
   CheckDep(DirBlock)
   → <A, #4> not ready
   Lock(DirBlock)
   Roll back <A, #4>
5. VM - Sync(DirBlock)
6. Soft Updates
   Roll forward <A, #4>
   UnLock(DirBlock)
   RemoveDep(DirBlock)

7. VM - notify SoftUpdates
   Sync(InodeBlock)

Asynchronous Solutions - Soft Updates

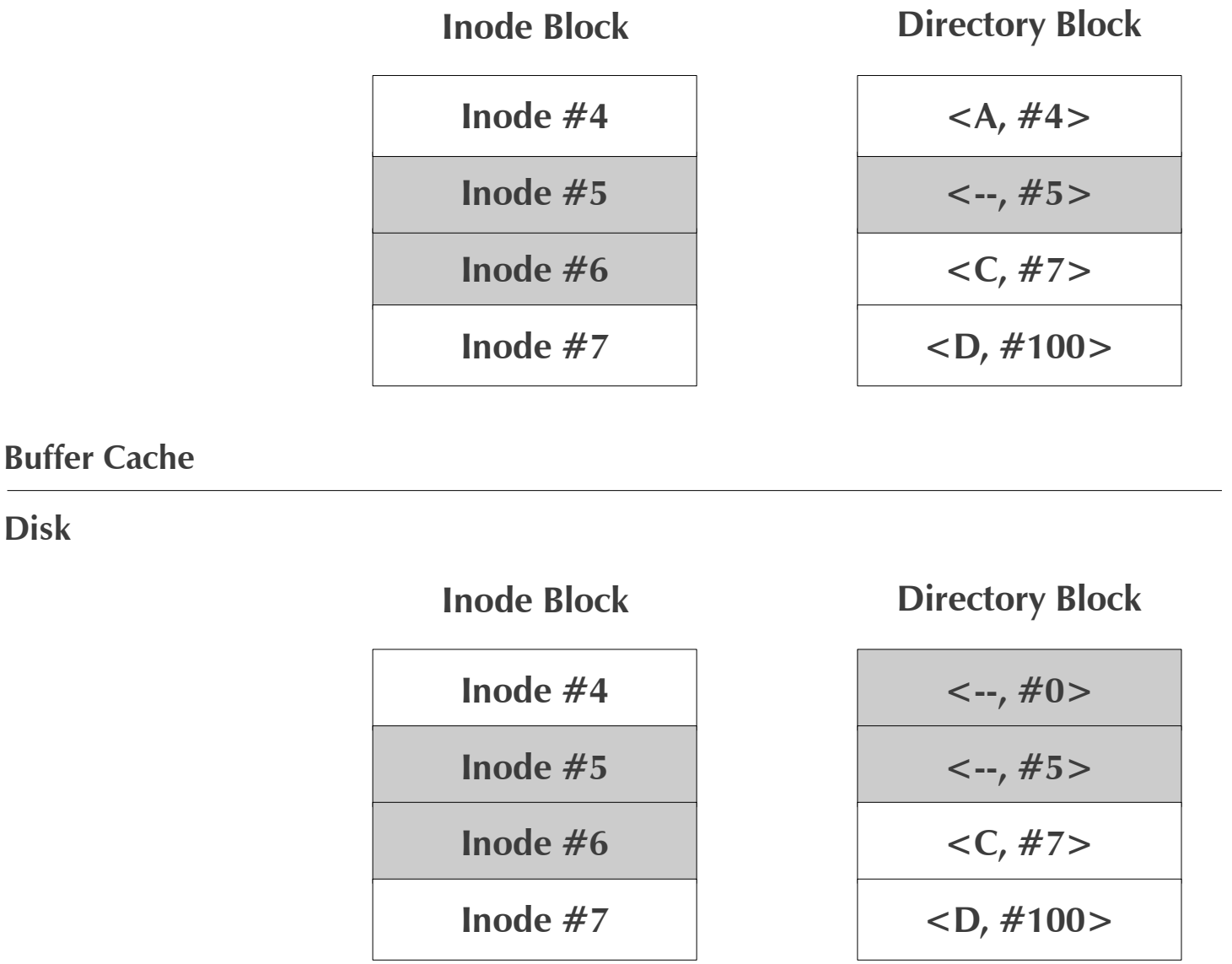


1. Create File A
2. Delete File B

3. VM - notify Soft Updates
   Sync(DirBlock)
4. Soft Updates
   CheckDep(DirBlock)
   → <A, #4> not ready
   Lock(DirBlock)
   Roll back <A, #4>
5. VM - Sync(DirBlock)
6. Soft Updates
   Roll forward <A, #4>
   UnLock(DirBlock)
   RemoveDep(DirBlock)

7. VM - notify SoftUpdates
   Sync(InodeBlock)
8. VM - Sync(InodeBlock)

Asynchronous Solutions - Soft Updates

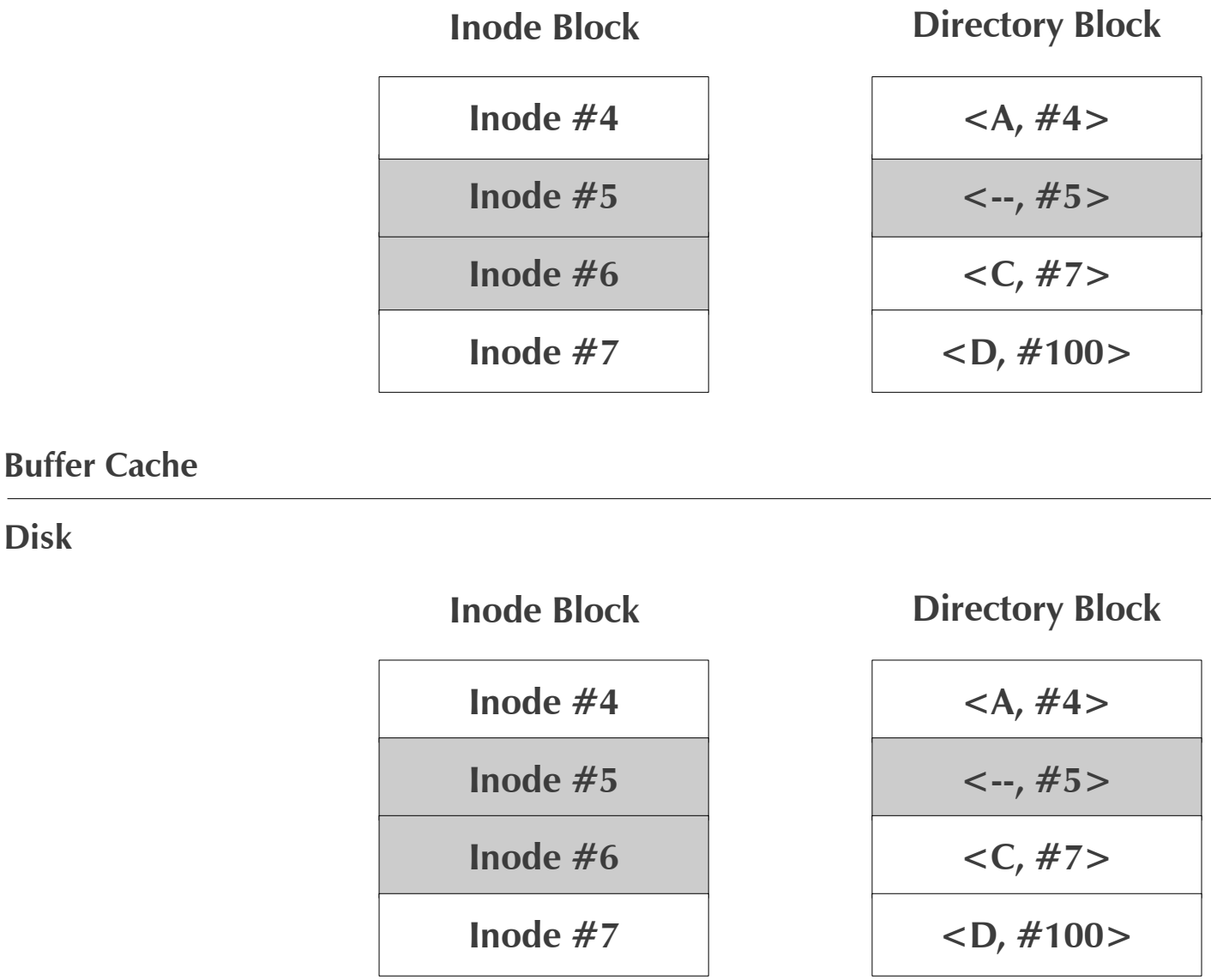


1. Create File A
2. Delete File B

3. VM - notify Soft Updates
   Sync(DirBlock)
4. Soft Updates
   CheckDep(DirBlock)
   → <A, #4> not ready
   Lock(DirBlock)
   Roll back <A, #4>
5. VM - Sync(DirBlock)
6. Soft Updates
   Roll forward <A, #4>
   UnLock(DirBlock)
   RemoveDep(DirBlock)

7. VM - notify SoftUpdates
   Sync(InodeBlock)
8. VM - Sync(InodeBlock)
9. **Soft Updates**
   RemoveDep(InodeBlock)

Asynchronous Solutions - Soft Updates



1. Create File A
2. Delete File B

3. VM - notify Soft Updates
   Sync(DirBlock)
4. Soft Updates
   CheckDep(DirBlock)
   → <A, #4> not ready
   Lock(DirBlock)
   Roll back <A, #4>
5. VM - Sync(DirBlock)
6. Soft Updates
   Roll forward <A, #4>
   UnLock(DirBlock)
   RemoveDep(DirBlock)

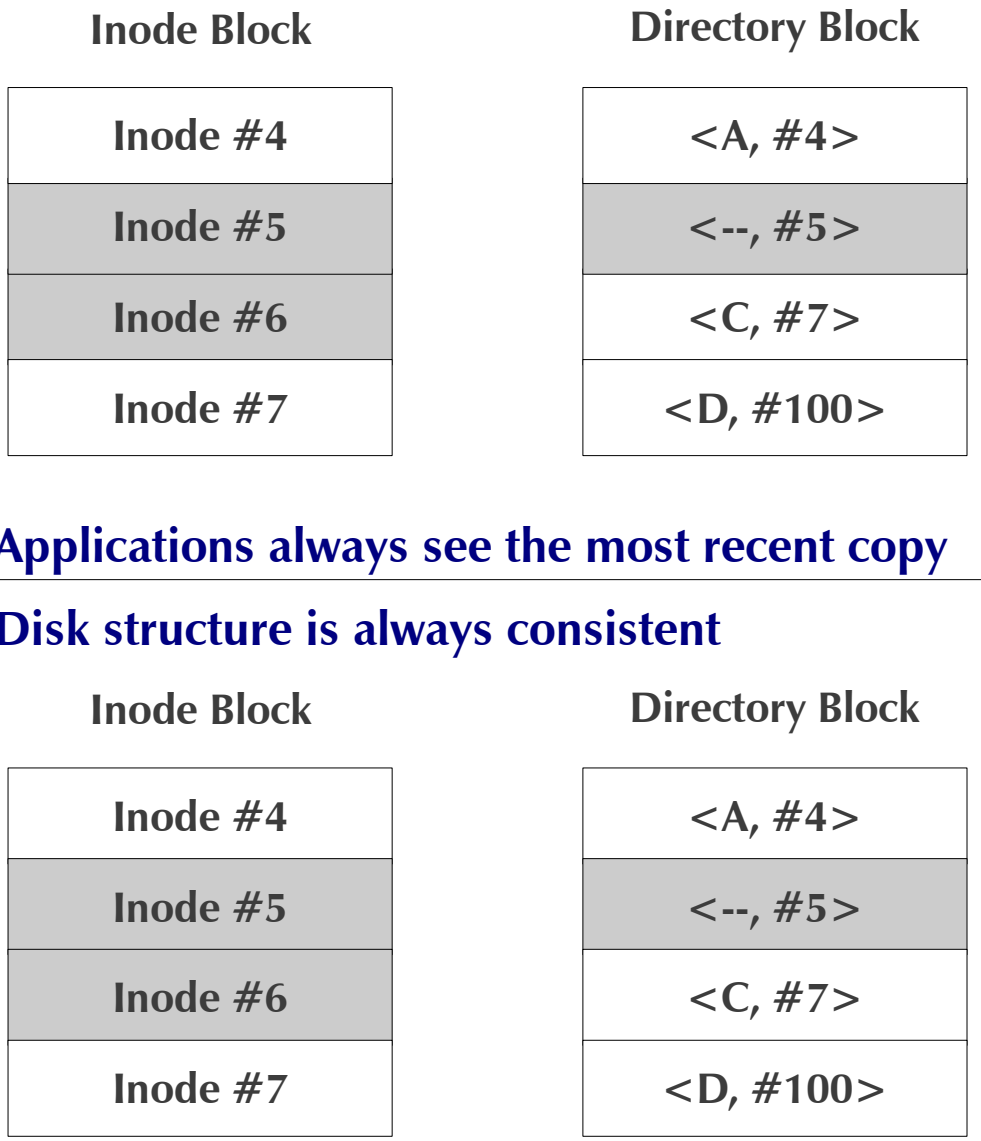
7. VM - notify SoftUpdates
   Sync(InodeBlock)
8. VM - Sync(InodeBlock)
9. Soft Updates
   RemoveDep(InodeBlock)

10. VM - Sync(DirBlock)

Asynchronous Solutions - Soft Updates

Buffer Cache

Disk



1. Create File A
2. Delete File B

3. VM - notify Soft Updates
   Sync(DirBlock)
4. Soft Updates
   CheckDep(DirBlock)
   → <A, #4> not ready
   Lock(DirBlock)
   Roll back <A, #4>
5. VM - Sync(DirBlock)
6. Soft Updates
   Roll forward <A, #4>
   UnLock(DirBlock)
   RemoveDep(DirBlock)

7. VM - notify SoftUpdates
   Sync(InodeBlock)
8. VM - Sync(InodeBlock)
9. Soft Updates
   RemoveDep(InodeBlock)

10. VM - Sync(DirBlock)

Asynchronous Solutions - Soft Updates

- **Benefits**
  - No recovery required (`fsck`)
  - Still enjoys delayed writes
- **Drawbacks**
  - Orphan resources (not atomic)
    - Integrity guaranteed, but still background `fsck` is required
  - Complex code

Which one is better?

Journaling VS Soft Updates

Implementations

- **FFS with journaling**
  - LFFS-file
  - LFFS-wafs
- **FFS with soft updates**

FFS with Journaling

- **LFFS-file**
  - Writes log records to a *file*
  - Writes log records *asynchronously*
  - 64KB cluster

- **LFFS-wafs**
  - Writes log records to a *separate filesystem*
    - Write Ahead File System
    - LFFS-wafs1(partition), LFFS-wafs2(disk)
  - Writes log records *synchronously/asynchronously*
  - 4KB log block

System Comparison

Feature Comparison

	Integrity	Durability	Atomicity	Recovery
FFS	X	X		fsck disk scan (minutes)
Sync Journaling	X	X	X	log-based (seconds)
Async Journaling	X		X	log-based (seconds)
Soft Updates	X			immediate
FFS-async				may be impossible

Performance Measurement

- **System configurations**

	FreeBSD Platform
Motherboard	Intel ASUS P38F, 440BX Chipset
Processor	500 Mhz Xeon Pentium III
Memory	512 MB, 10 ns
Disk	3 9 GB 10,000 RPM Seagate Cheetahs Disk 1: Operating system, /usr, and swap Disk 2: 9,088 MB test partition Disk 2: 128 MB log partition Disk 3: 128 MB log partition
I/O Adapter	Adaptec AHA-2940UW SCSI
OS	FreeBSD-current (as of 1/26/00 10:30 PM) config: GENERIC + SOFTUPDATES – bpfilter – unnecessary devices

Table 3. System Configuration.

- **Benchmarks**
  - Microbenchmark - only metadata operations (create/delete)
  - Macrobenchmarks - real workloads

Microbenchmark Result

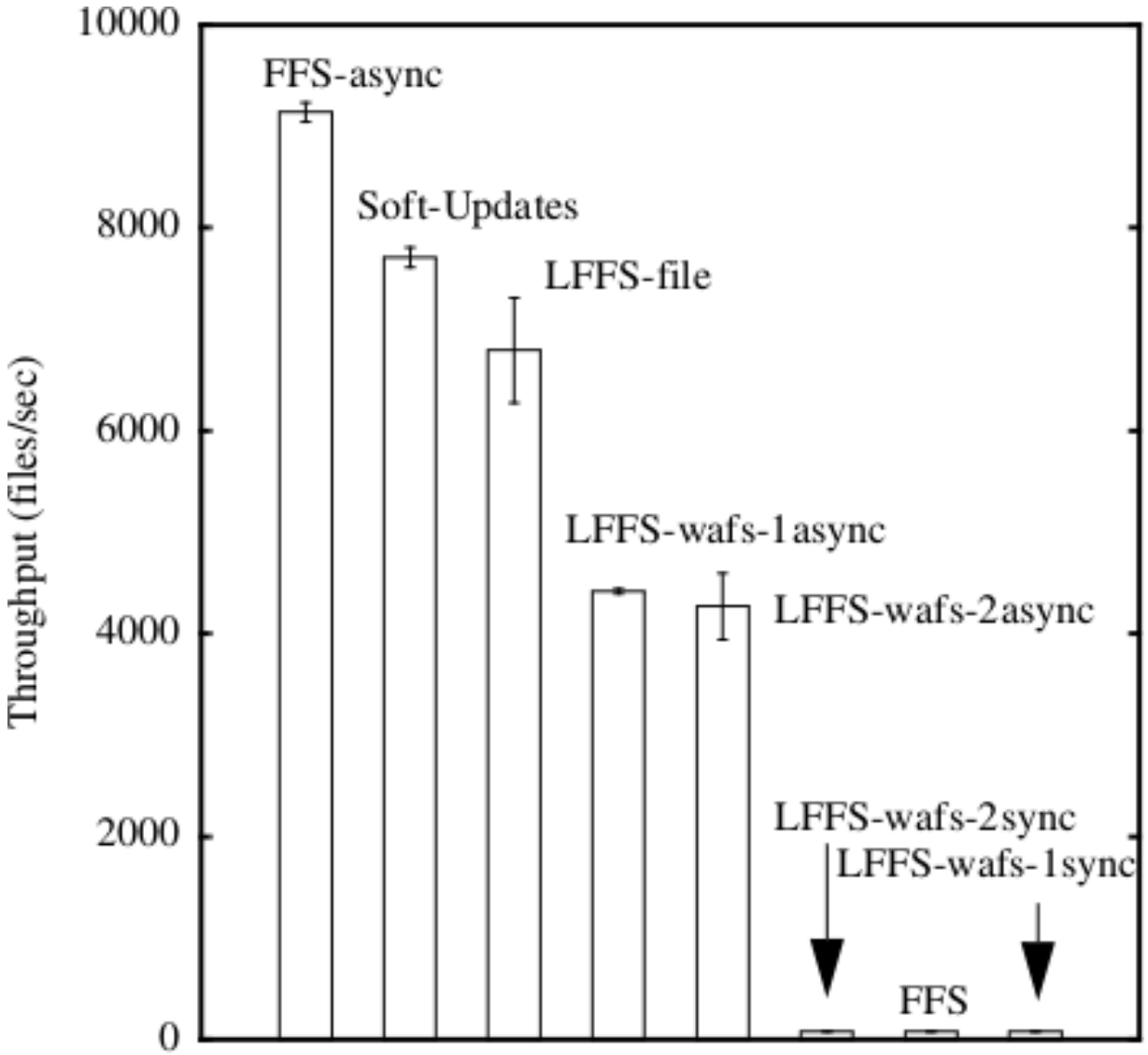


Figure 3. 0-length File Create/Delete Results in Files per Second.

Macrobenchmark Result - Postmark

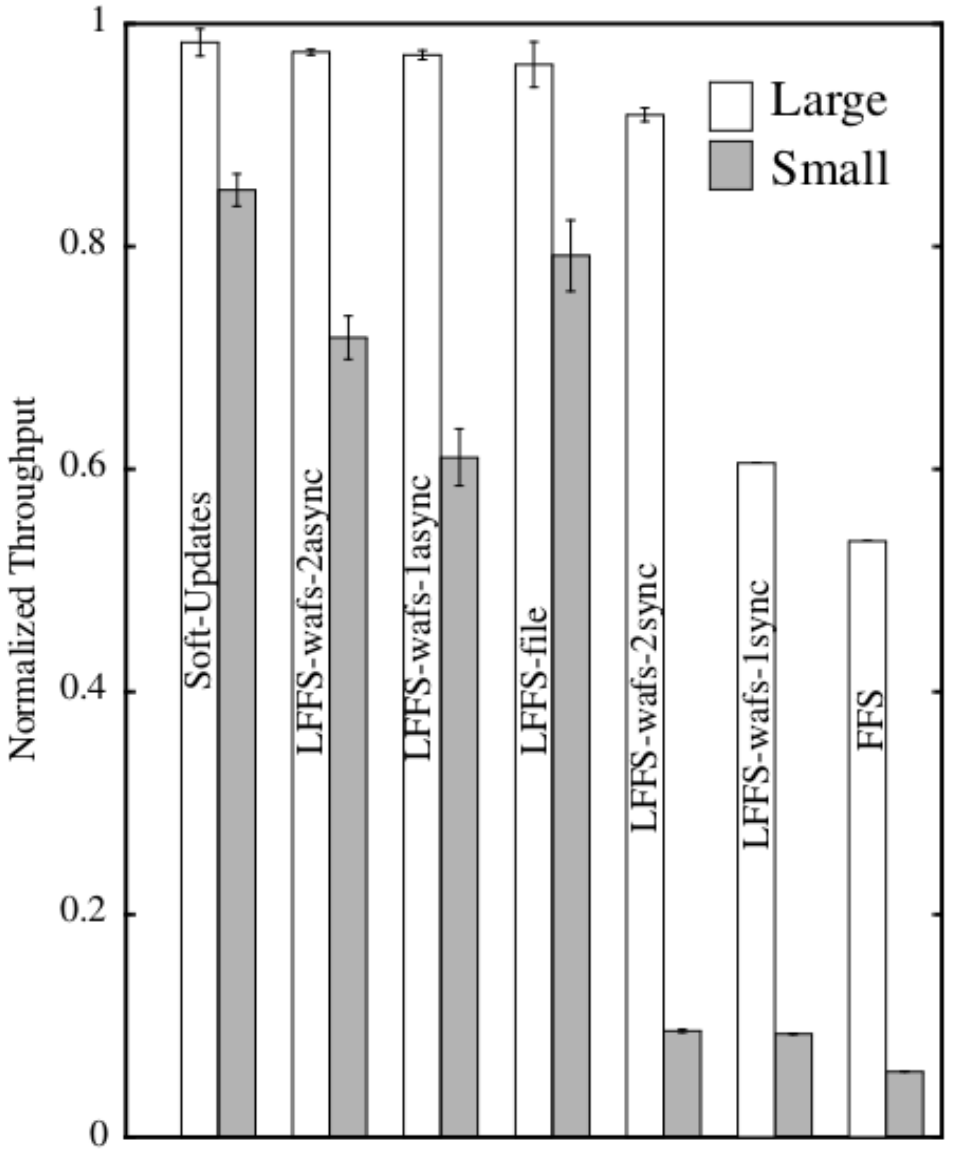


Figure 5. PostMark Results. The results are the averages of five runs; standard deviations are shown with error bars.

Conclusions

- **Metadata operations are significant**
- **Filesystem integrity**
  - Asynchronous approaches improves the performance of metadata operations
- **Journaling & Soft Updates**
  - Comparable performance

Arguments over Synchronous Update (FFS)

BSD "synchronous" filesystem updates are braindamaged.
BSD people touting it as a feature are WRONG. It's a bug.
Synchronous meta-data updates are STUPID:
(a) it's bad for performance
(b) it's bad for filesystem stability

… Linus Tovalds, 1995

- **The correct way should be:**
  - write out the data blocks first, _then_ write out the meta-data.
- **`fsck` can fix inconsistent metadata**
  - No reason to sacrifice the runtime performance
  - Do nothing about metadata operations - Ext2

Arguments over Soft Updates

One major downside with Soft Updates that you haven't mentioned in your note, is that the amount of complexity it adds to the filesystem is tremendous;
...
(about the instant mount after crash)
This is only true if you don't care about recovering lost data blocks. Fixing this requires that you run the equivalent of fsck on the filesystem.
… Theodore Ts'o, 2006

- **Implementation complexity**
  - Hard to extend a filesystem
- **Still `fsck` required**
  - Lost data blocks
- **Metadata oriented workload is unrealistic**

Journaling Soft Updates

- **Limitation of Soft Updates**
  - Lost data blocks
- **Journaling Soft Updates**
  - Kirk Mckusick, Jeffery Roberson
  - EuroBSD Conference, 2010
- **Use journaling to reclaim orphan resources**
  - Journal size can be much smaller than other journaling filesystems

Ext3 Journaling

- **Considers the *normal data blocks* as well**
- **Multiple journaling modes**
  - writeback
    - Only journals metadata changes
  - ordered
    - Only journals metadata changes
    - *Data updates are flushed to disk before journal records commit*
  - journal
    - Journals all data and metadata

Other Approaches to Metadata Update

- **NVRAM**
  - Quick recovery
  - Expensive/Unreliable hardware

- **Copy-on-write filesystems (LFS, ZFS, …)**
  - Never overwrites; always write the data in newly allocated blocks
  - Write throughput, cheap snapshots, always consistent
  - Disk fragmentation, memory overhead

Questions?