Consistency Without Ordering

Vijay Chidambaram, Tushar Sharma,
Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau

The Advanced Systems Laboratory
University of Wisconsin Madison

The problem: crash consistency

- Single operation updates multiple blocks
- System might crash in the middle of operation
  – Some blocks updated, some blocks not updated
- After crash, file system needs to be repaired
  – In order to restore consistency among blocks

Solution #1: Lazy, optimistic approach

- Write blocks to disk in any order
  - Fix inconsistencies upon reboot
- Advantage: Simple, High performance
- Disadvantage: Expensive recovery
- Example: ext2 with fsck [*Card94*]

Solution #2: Eager, pessimistic approach

- Carefully order writes to disk
- Advantage: Quick recovery
- Disadvantage: Perpetual performance penalty
- Examples
  - Soft updates (FFS) [*Ganger94*]
  - Journaling (CFS) [*Hangmann87*]
  - Copy-on-write (ZFS) [*Bonwick04*]

Ordering points considered harmful

- Reduce performance
  - Constrain scheduling of disk writes

- Increase complexity

- Require lower-level primitives
  - IDE/SATA Cache flush commands

Ordering points require trust

- File system runs on stack of virtual devices
  - Consistency fails if any device ignores commands to flush cache

F_FULLFSYNC *"…The operation may take quite a while to complete. Certain FireWire drives have also been known to ignore the request to flush their buffered data."*

VirtualBox *"If desired, the virtual disk images can be flushed when the guest issues the IDE FLUSH CACHE command. Normally these requests are ignored for improved performance"*

Is crash-consistency possible without ordering points?

- Middle ground between lazy and eager approaches
- Simplicity and high performance of lazy approach
- Strong consistency and availability of eager approach

Our solution:

No-Order File System (NoFS)

Order-less file system which uses mutual agreement between objects to obtain consistency

Results

- Designed a new crash-consistency technique
  - Backpointer-based consistency (BBC)
- Theoretically and experimentally verified that NoFS provides strong consistency
- Evaluated NoFS against ext2 and ext3
  - NoFS performance comparable to ext2
  - NoFS performance equal to or better than ext3

Outline

- Introduction
- Crash-consistency and Object identity
- The No-Order File System
- Results
- Conclusion

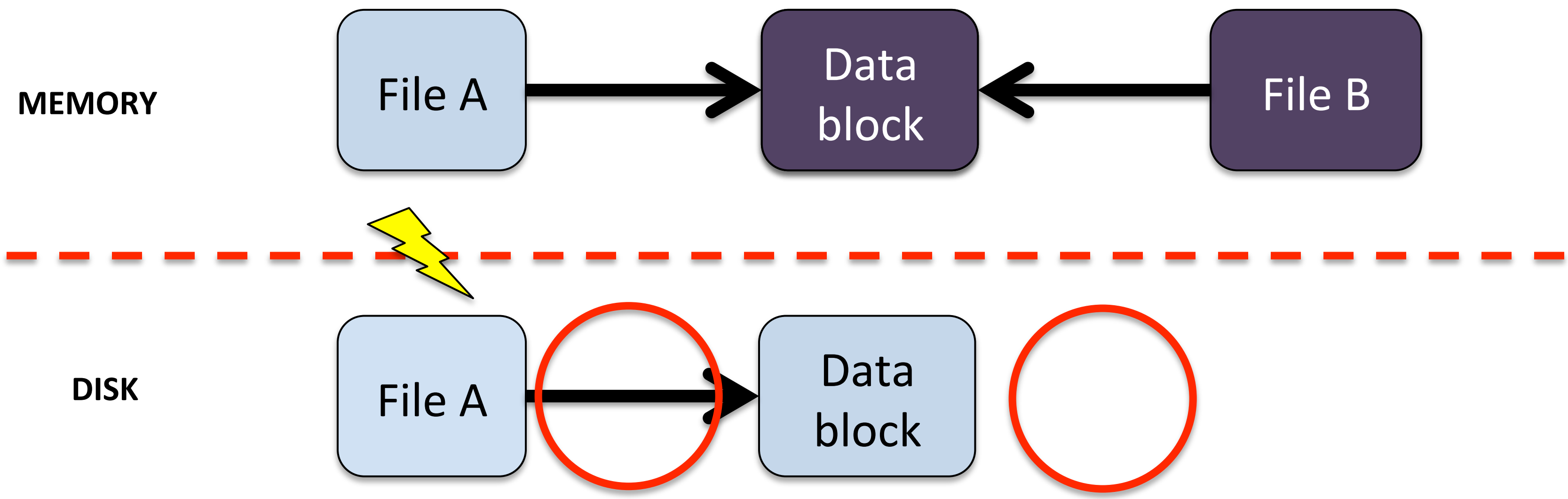
Crash consistency and object identity

All file system inconsistencies are due to ambiguity about the logical identity of an object

- Logical identity of an object
  - Data block: Owner file, offset
  - File: Parent directories
- Common inconsistencies
  - Two files claim the same data block
  - File points to garbage data

Crash Scenario

- Actions:
  - File A is truncated
  - The freed data block is allocated to File B
  - The updated data blocks are written to disk
- Problem: Due to a crash, File A is not updated on disk
- Result: On disk, both files claim the data block

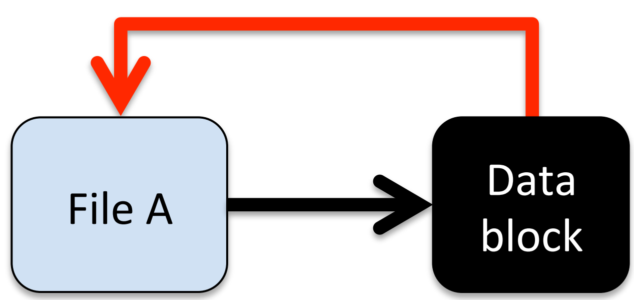


Outline

- Introduction
- Crash-consistency and Object identity
- The No-Order File System
  - Backpointer-based consistency (BBC)
  - Non-persistent allocation structures
- Results
- Conclusion

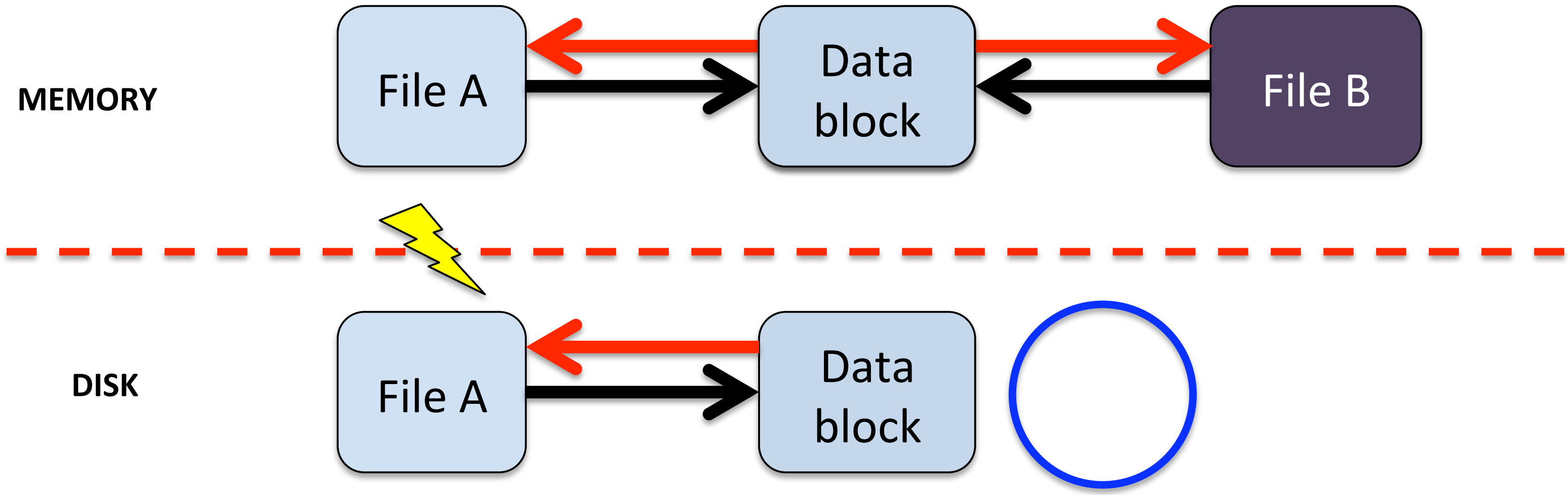
Backpointer-based consistency (BBC)

- Associate object with its logical identity
  - Embed *backpointer* into each object
  - Owner(s) of the object found through backpointer
- Consistency obtained through mutual agreement
- Key Assumption
  - Object and backpointer written atomically



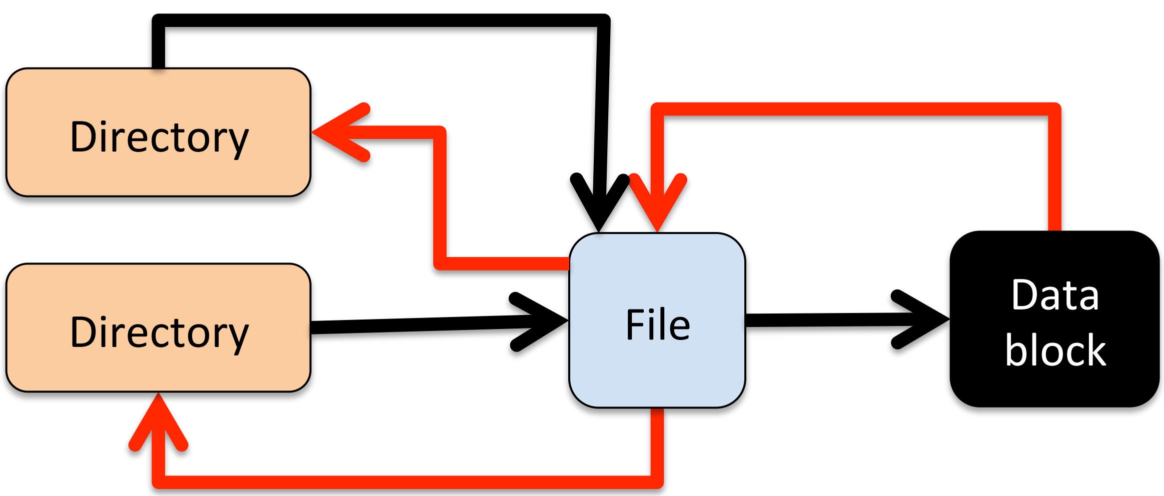
Using backpointers in a crash scenario

- Actions:
  - File A is truncated
  - The freed data block is allocated to File B
  - The updated data blocks are written to disk
- Problem: Due to a crash, File A is not updated on disk
- Result: Using the backpointer, the true owner is identified



Backpointers of different objects

- Data blocks have a single backpointer to file
- Files can have many backpointers
  - One for each parent directory
- Detection of inconsistencies
  - Each access of an object involves checking its backpointer



Formal Model of BBC

- Extended a formal model for file systems with backpointers [*Sivathanu05*]

- Defined the level of consistency provided by BBC
  – Data consistency

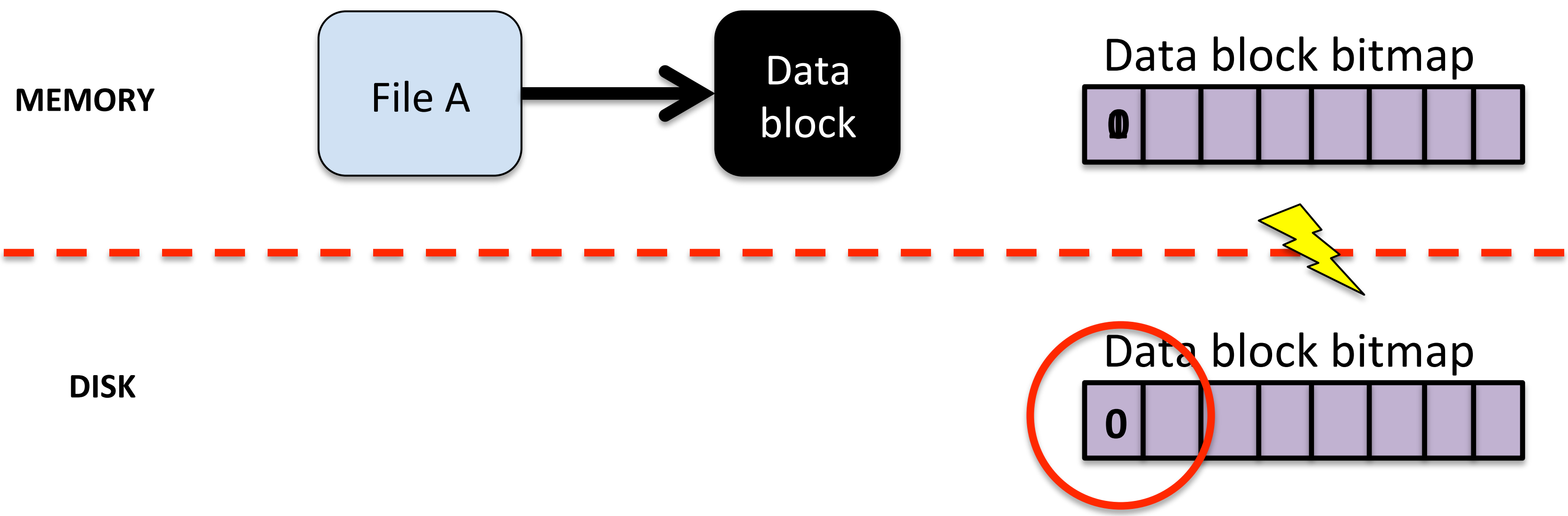
- Proved that a file system with backpointers provides data consistency

Outline

- Introduction
- Crash-consistency and Object identity
- The No-Order File System
  - Backpointer-based consistency
  - Non-persistent allocation structures
- Results
- Conclusion

Allocation structures

- File systems need to track allocation status
- Crash may leave such structures inconsistent
- True allocation status needs to be found



Allocation structures

- After a crash, true allocation status of all objects must be found
- Traditional file systems do this proactively
  - File-system check scans disk to get status
  - Journaling file systems write to a log to avoid scan

Non-persistent allocation structures

- NoFS does not persist allocation structures
- Why?
  - Cannot be trusted after crash, need to be verified
  - Complicate update protocol

Non-persistent allocation structures

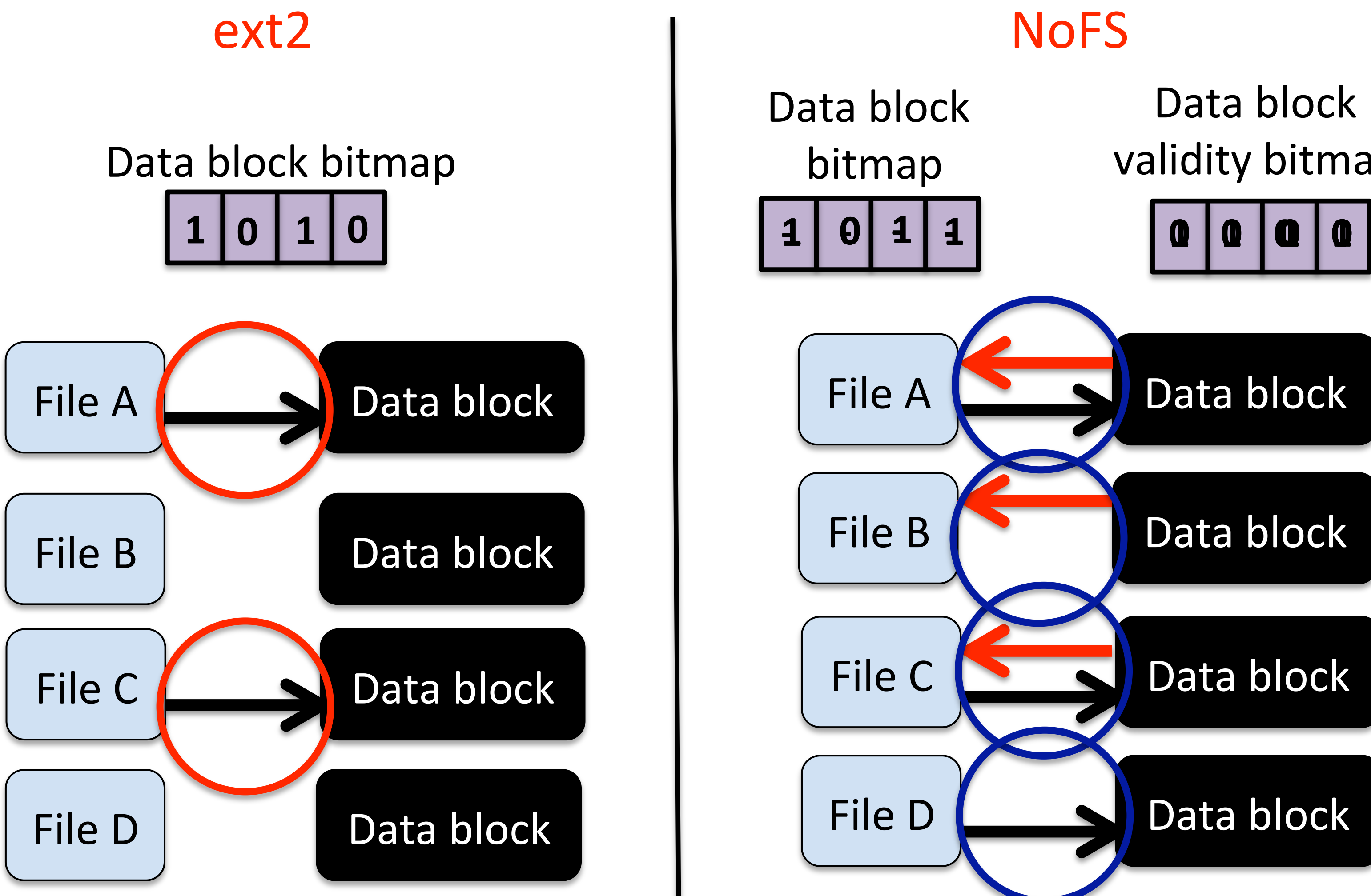
- How is allocation information tracked then?
  - Need to know which metadata/data blocks are free

- Move the work of finding allocation information to the background
  - Creation of new objects can proceed without complete allocation information

Non-persistent allocation structures

- Backpointers used to determine allocation
  - Object in use if pointers mutually agree
  - Check each object individually
  - Use *validity bitmaps* to track checked objects
- Allocation structures built up incrementally

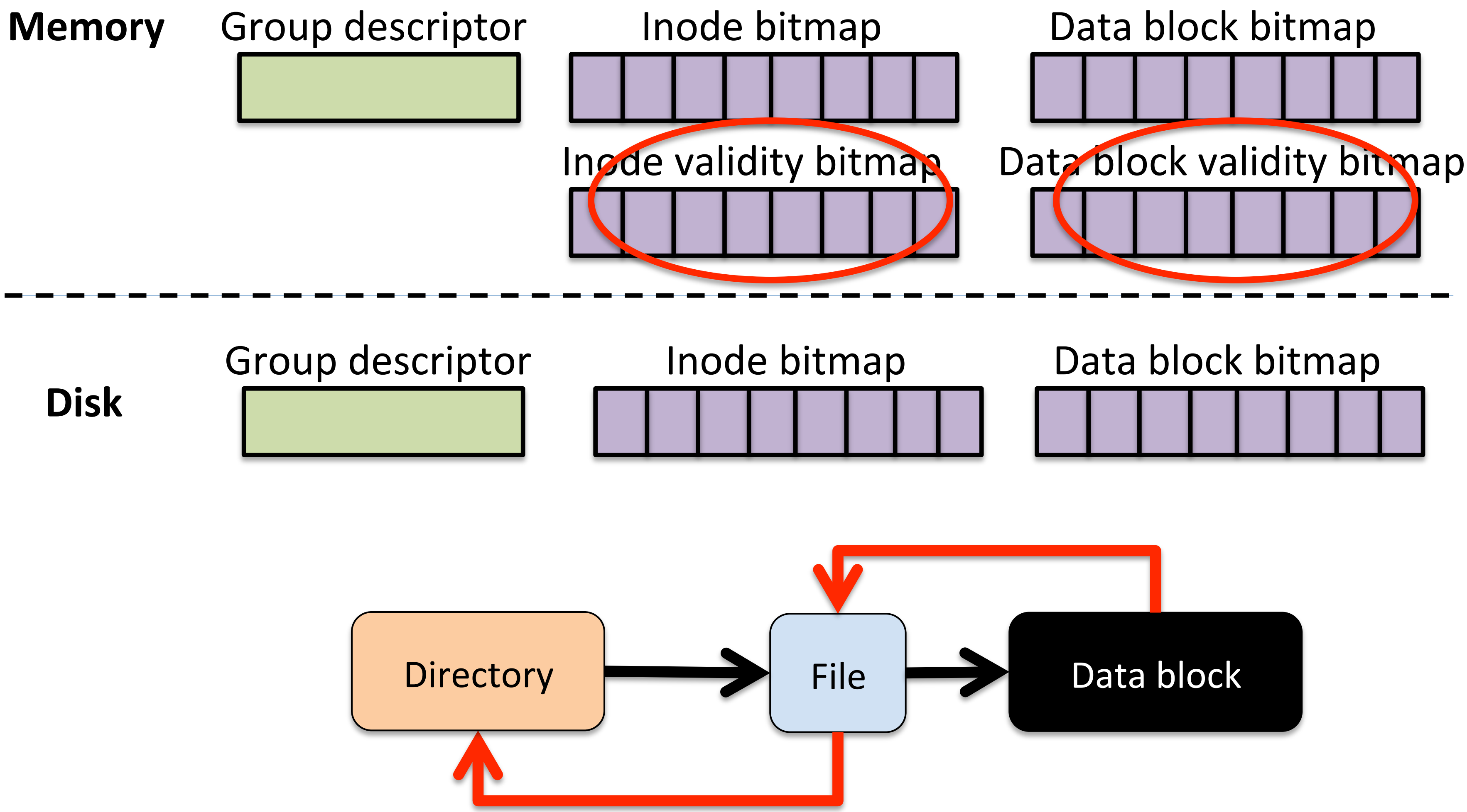
Determining allocation information



Background Scan

- Complete allocation information not needed
- Allocation information discovered using two background threads
  - One for metadata
  - One for data
- Scheduling of scan can be configured
  - Run when idle
  - Run periodically

Design


Memory
Group descriptor
Inode bitmap
Data block bitmap
Inode validity bitmap
Data block validity bitmap
Disk
Group descriptor
Inode bitmap
Data block bitmap
Directory
File
Data block

Implementation

- Based on ext2 codebase
- Three types of backpointers
  - Data block backpointers {inode num, offset}
  - Inode backlinks {inode num}
  - Directory block backpointers {dot directory entry}
- Inode size increased to support 32 backlinks
- Modified the linux page cache to add checks

Outline

- Introduction
- Crash-consistency and Object identity
- The No-Order File System
  - Backpointer-based consistency
  - Non-persistent allocation structures
- Results
- Conclusion

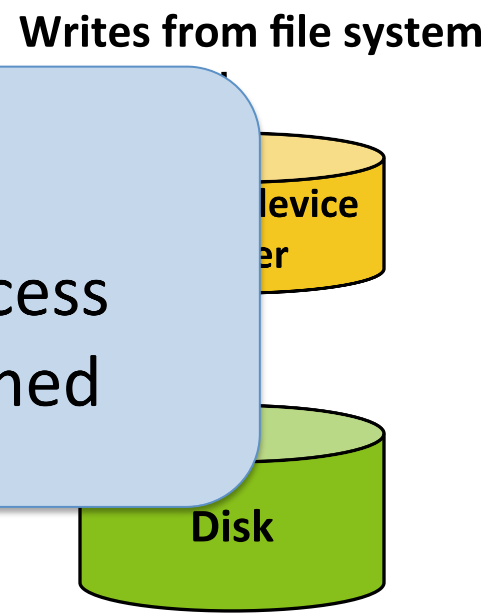
Evaluation

- Q: Is NoFS robust against crashes?
  - Fault injection testing
- Q: What is the overhead of NoFS?
  - Evaluated on micro and macro benchmarks
- Q: How does the background scan affect performance?
  - Measured write bandwidth, access latency during scan

Is NoFS robust against crashes?

Fault injection testing

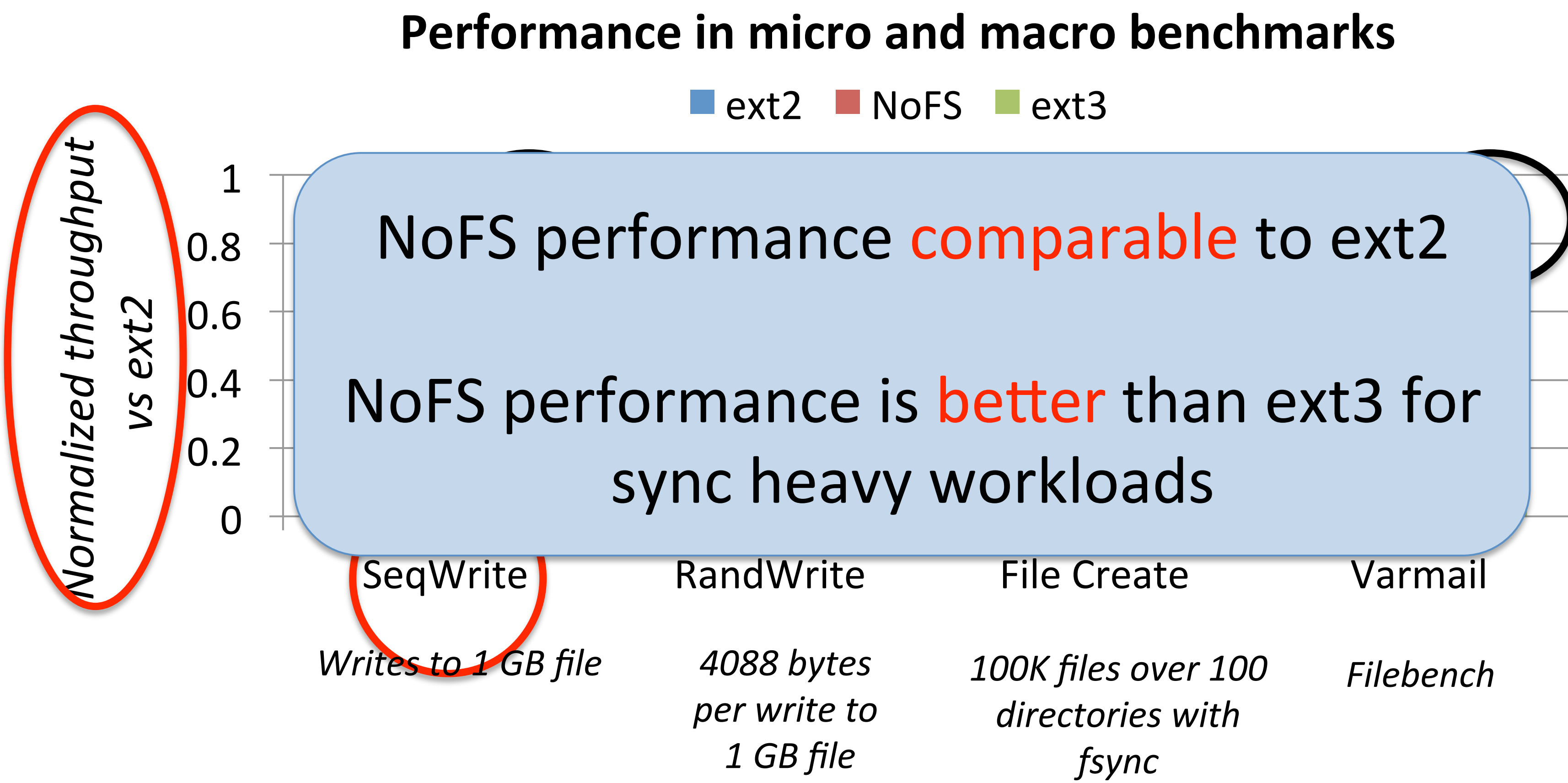
- Interpose pseudo-device driver bet
- Dis
- Em suc
- 20 different crash scenarios



NoFS detected all inconsistencies

- Errors returned on invalid access
- Orphan inodes/blocks reclaimed

What is the overhead of NoFS?



NoFS performance comparable to ext2

NoFS performance is better than ext3 for sync heavy workloads

How does the background scan affect performance?

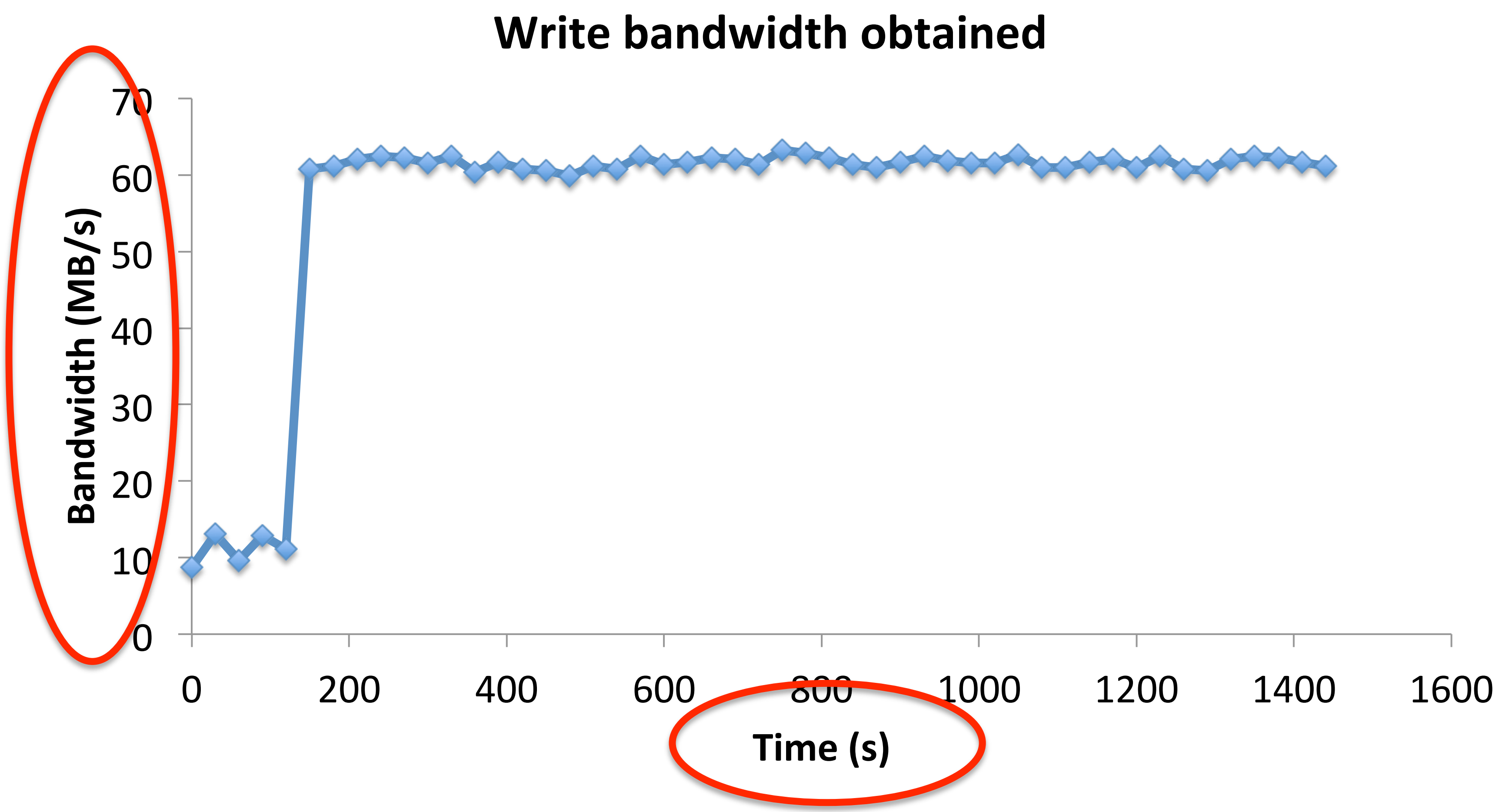
- Scan reads are interleaved with file system I/O
- Access to objects not verified by scan incurs a performance penalty

Scan reads are interleaved with file system I/O

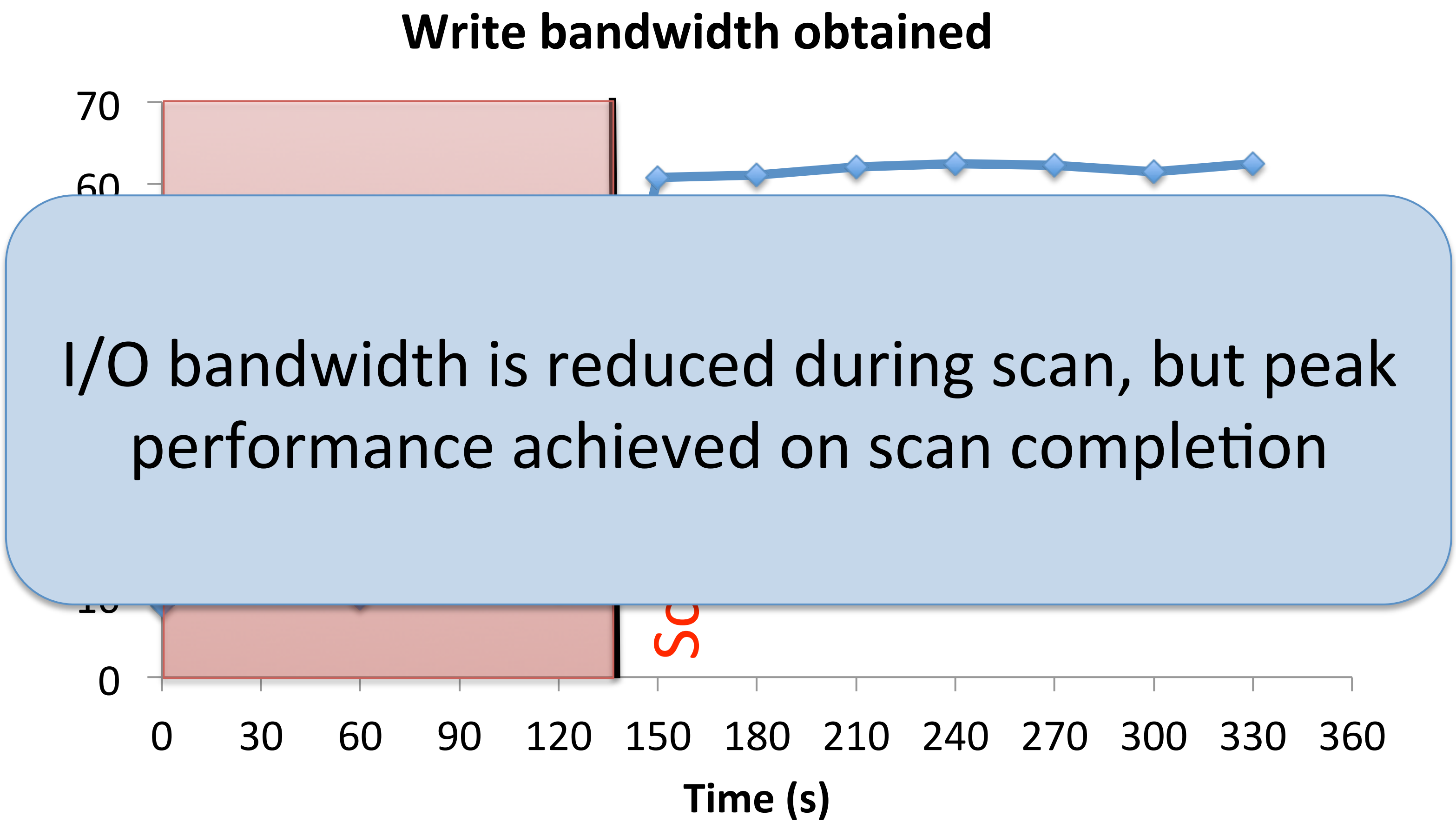
- Scan reads interfere with application reads and writes

- Experiment
  - Write a 200 MB file every 30 seconds
  - Measure bandwidth

Scan reads are interleaved with file system I/O



Scan reads are interleaved with file system I/O



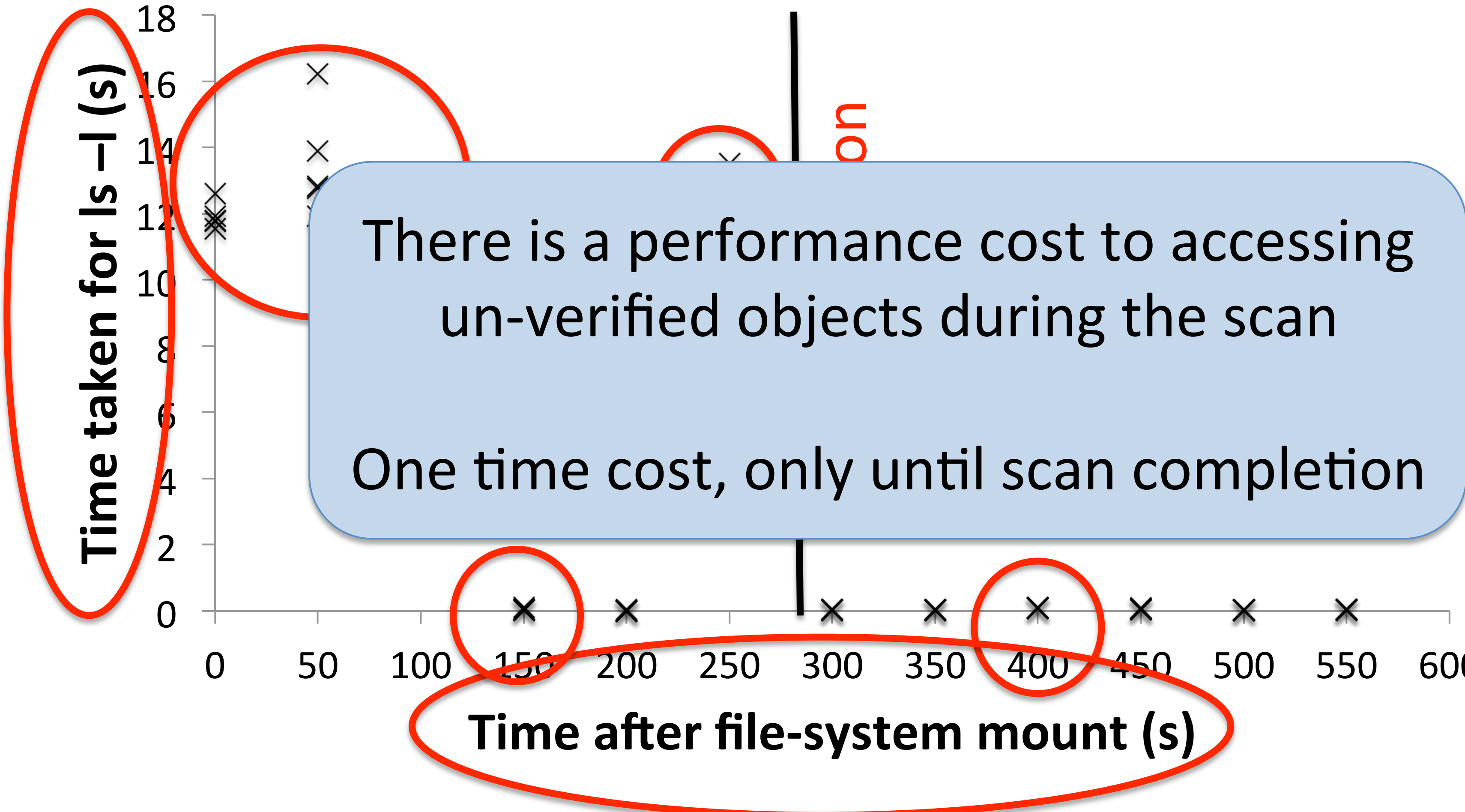
Access to objects not verified by scan costs more

- The stat problem
  - stat returns number of blocks allocated
  - This information might be stale for un-verified inode
  - NoFS verifies the inode upon stat
    - Involves checking each inode data block

Access to objects not verified by scan costs more

- Experiment
  - Create a number of directories with 128 files (each 1 MB)
  - At each 50 second interval, starting from file-system mount
    - Run ls –l on directory
    - This causes a stat call on every inode
    - stat on un-verified inodes requires reading all its data
  - Measure time taken

Access to objects not verified by scan costs more



Outline

- Introduction
- Crash-consistency and Object identity
- The No-Order File System
  - Backpointer-based consistency
  - Non-persistent allocation structures
- Results
- Conclusion

Summary

- Problem: Providing crash-consistency and high availability without ordering points
- Solution: NoFS with Backpointer-based consistency
  - Use mutual agreement to drive consistency
- Advantages:
  - Strong consistency guarantees
  - Performance similar to order-less file system

Conclusion

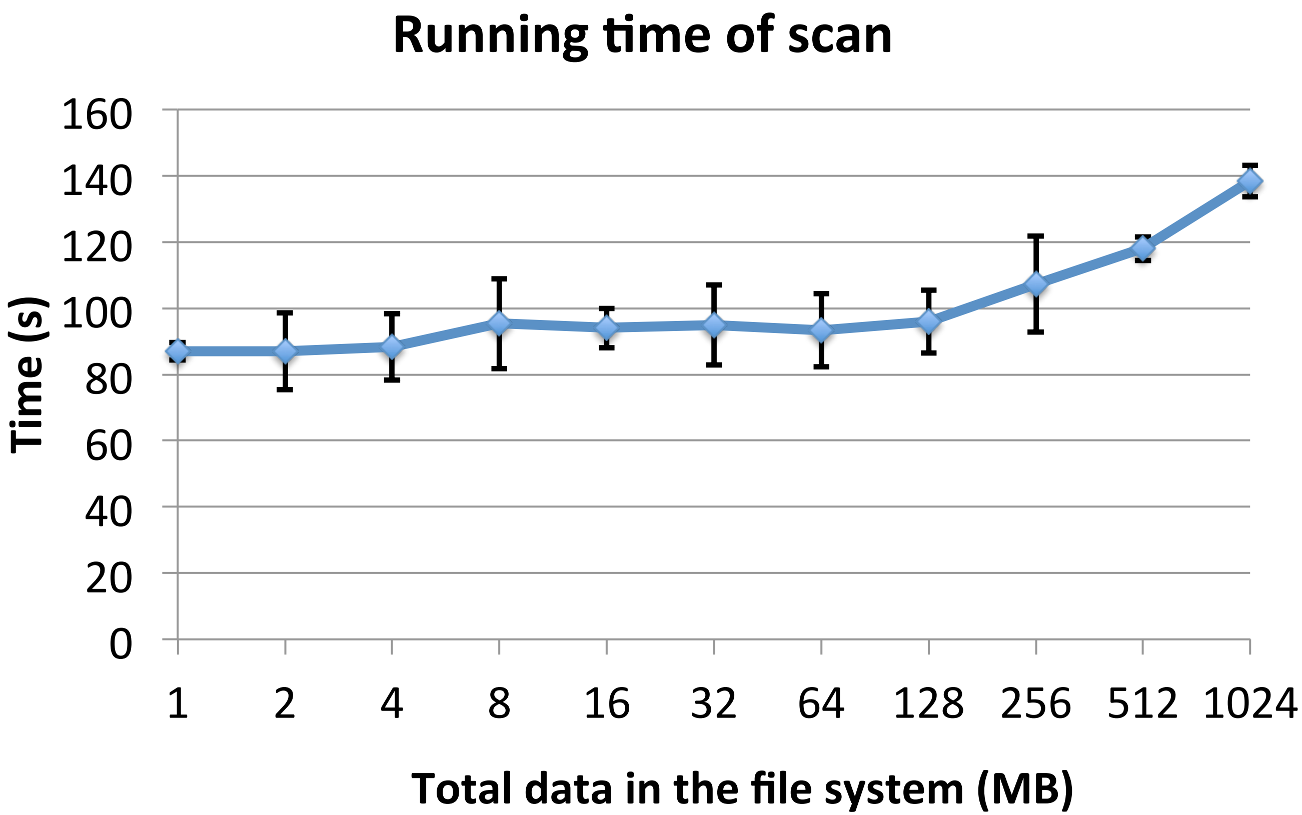
- Trust is implicit in many layers of storage systems
- Removing such trust is key to building robust, reliable storage systems

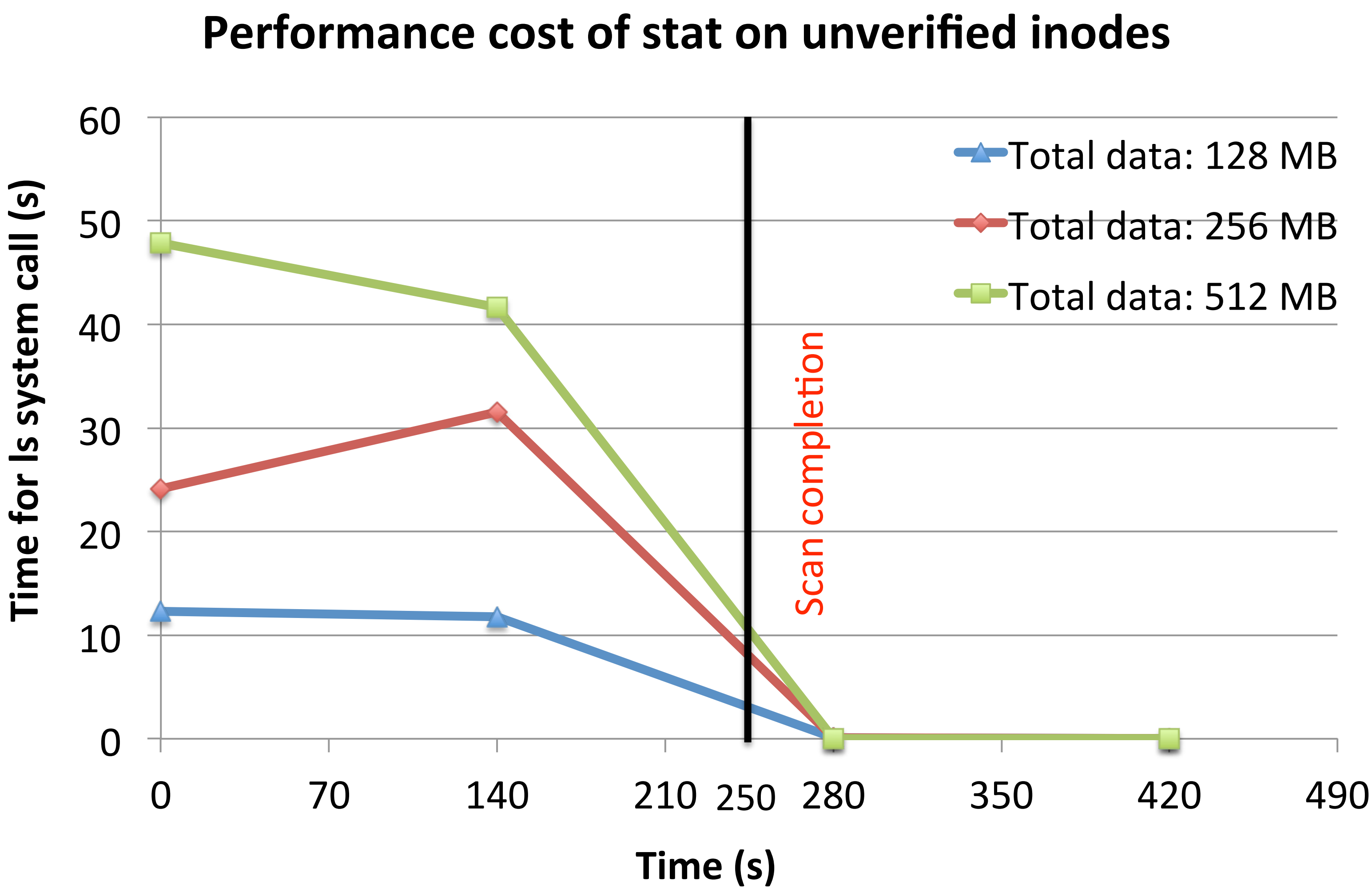
Thank you!

Questions?

Advanced Systems Lab (ADSL)
University of Wisconsin-Madison
http://www.cs.wisc.edu/adsl

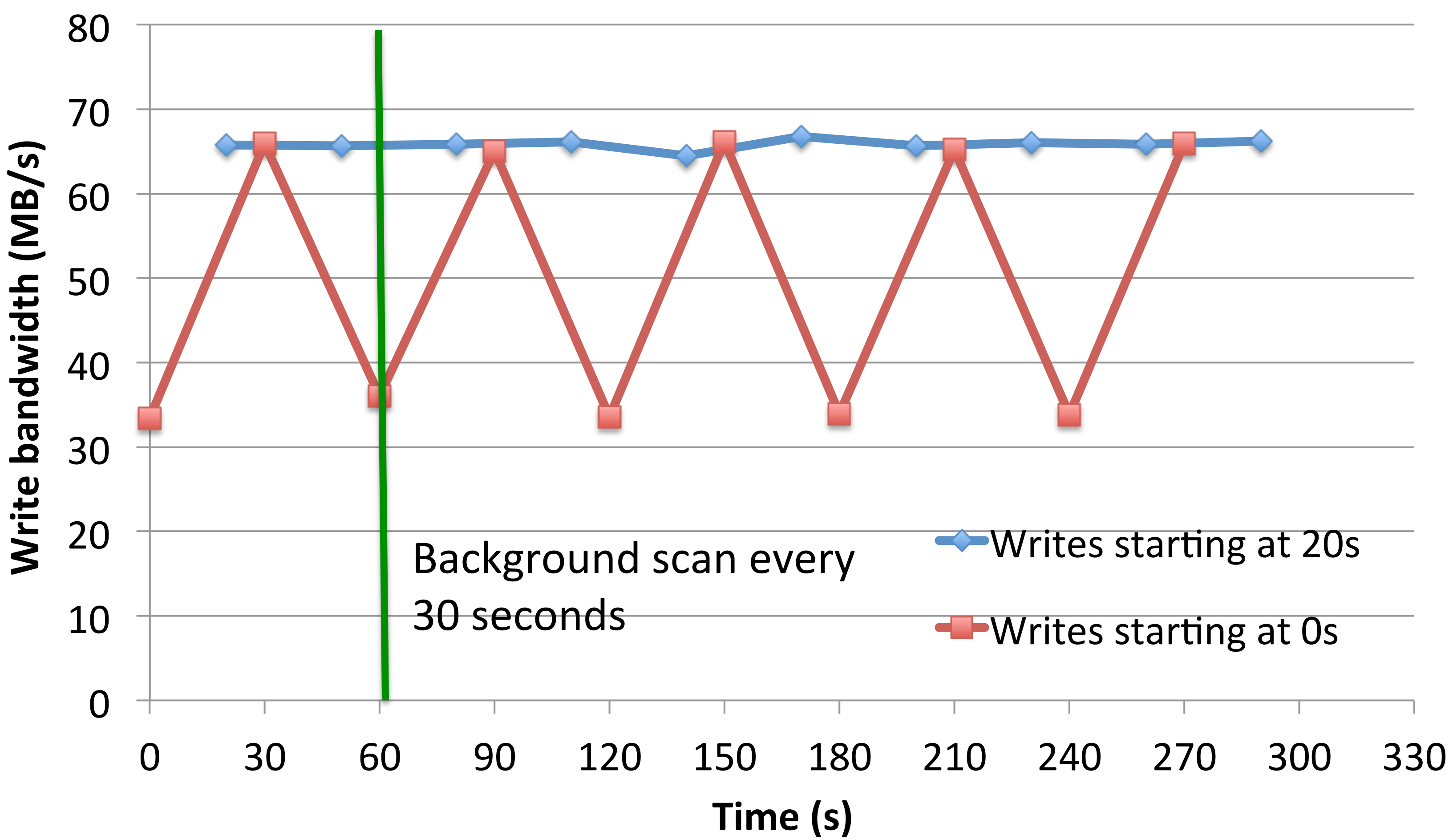
Backup Slides

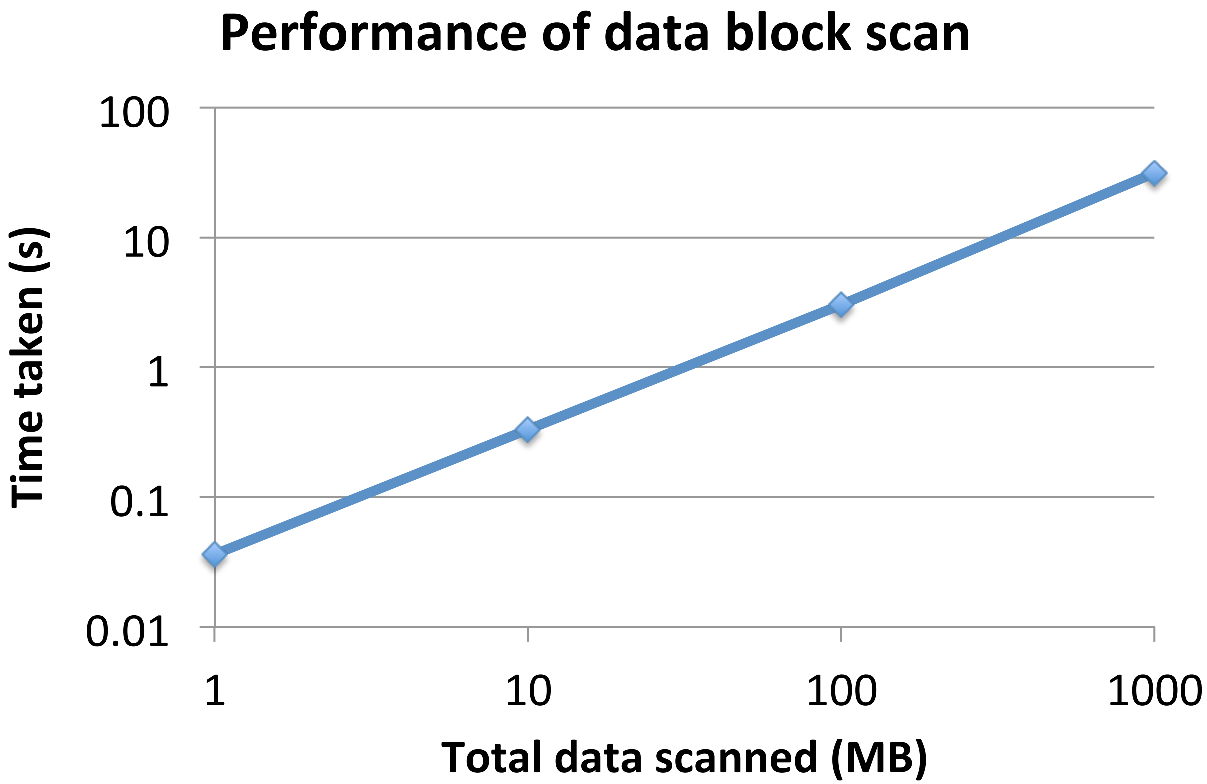

Running time of scan
Time (s)
160
140
120
100
80
60
40
20
0
1
2
4
8
16
32
64
128
256
512
1024
Total data in the file system (MB)



Performance cost of stat on unverified inodes
Time for ls system call (s)
60
50
40
30
20
10
0
0
70
140
210
250
280
350
420
490
Time (s)
Total data: 128 MB
Total data: 256 MB
Total data: 512 MB
Scan completion

Effect of background scan on write bandwidth




Performance of data block scan
100
10
1
0.1
0.01
Time taken (s)
1
10
100
1000
Total data scanned (MB)

Lines of code:	6765
Kernel:	2869
File system:	3869

Use cases

- NoFS provides crash-consistency without ordering
  - BBC can be used in conventional file systems to ensure runtime integrity
  - NoFs can be used as local file system in GFS, HDFS
- NoFS allows virtual machines to maintain consistency without trusting lower-layer primitives