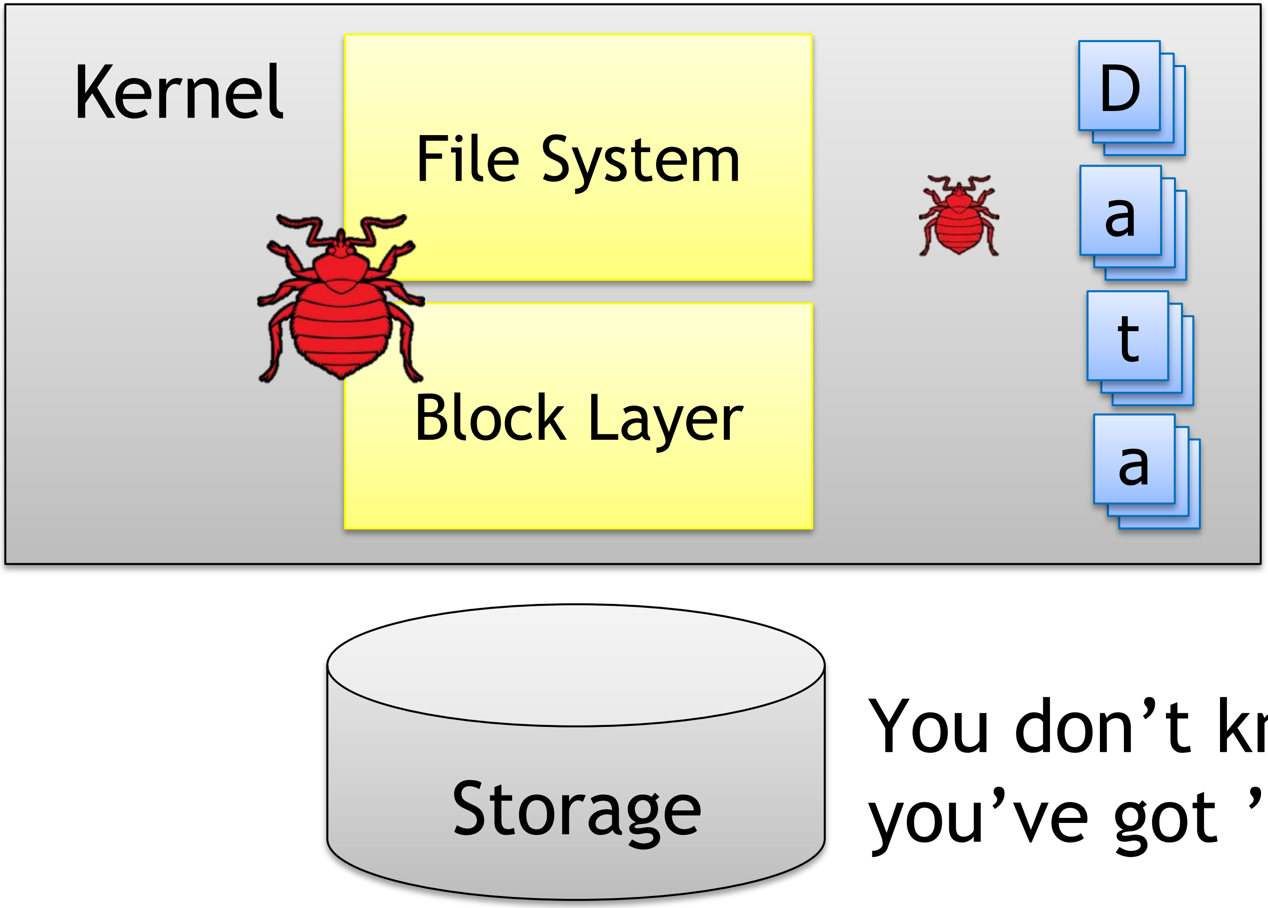
Recon: Verifying File System Consistency at Runtime

Daniel Fryer, Jack (Kuei) Sun,
Rahat Mahmood, TingHao Cheng, Shaun Benjamin,
Angela Demke Brown and Ashvin Goel

University of Toronto

Metadata Integrity is Crucial



You don't know what you've got 'til it's gone…

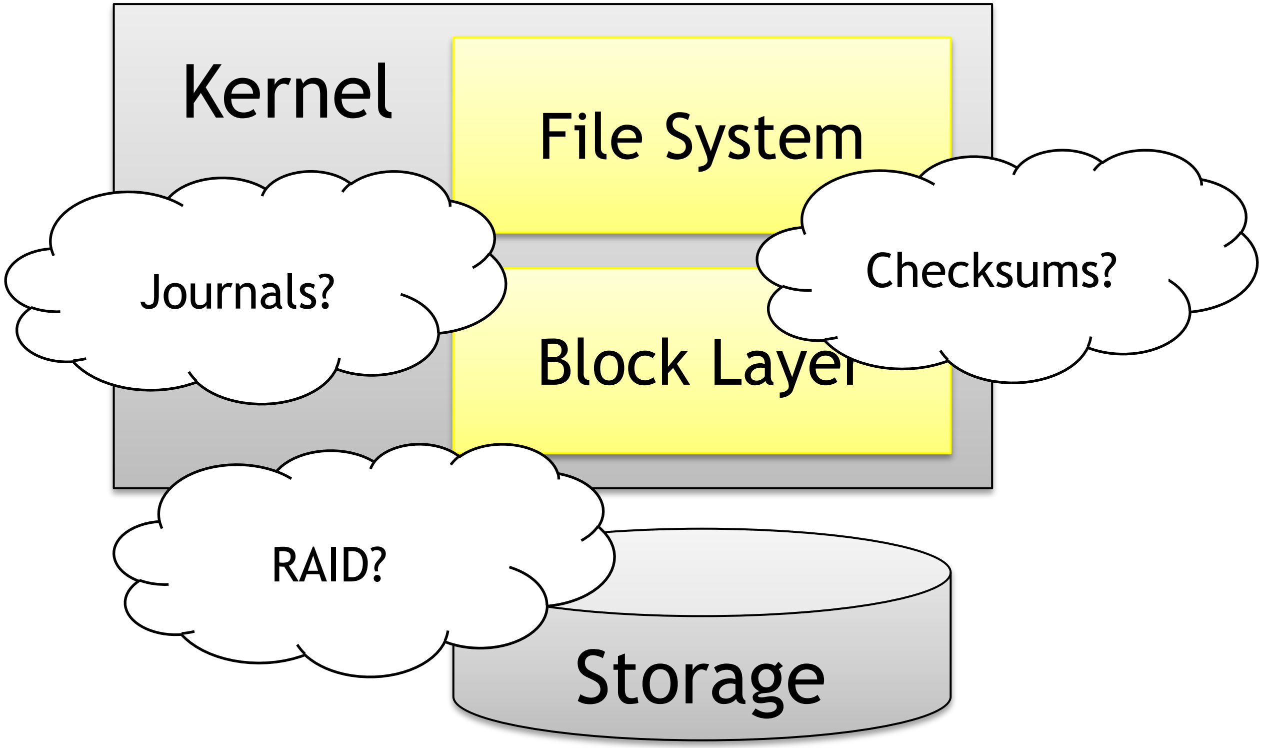
File Systems Have Bugs

Bugs in Linux Ext3 File System	Closed
panic/ext3 fs corruption with RHEL4-U6-re20070927.0	2007-11
Re: [2.6.27] filesystem (ext3) corruption (access beyond end)	2008-06
linux-2.6: ext3 filesystem corruption	2008-09
linux-image-2.6.29-2-amd64: occasional ext3 filesystem corruption	2009-06
ENOSPC during fsstress leads to filesystem corruption on ext2, ext3, and ext4	2010-03
ext3: Fix fs corruption when make_indexed_dir() fails	2011-06
Data corruption: resume from hibernate always ends up with EXT3 fs errors	Not yet

Why can’t existing solutions handle this problem?

“Solutions”

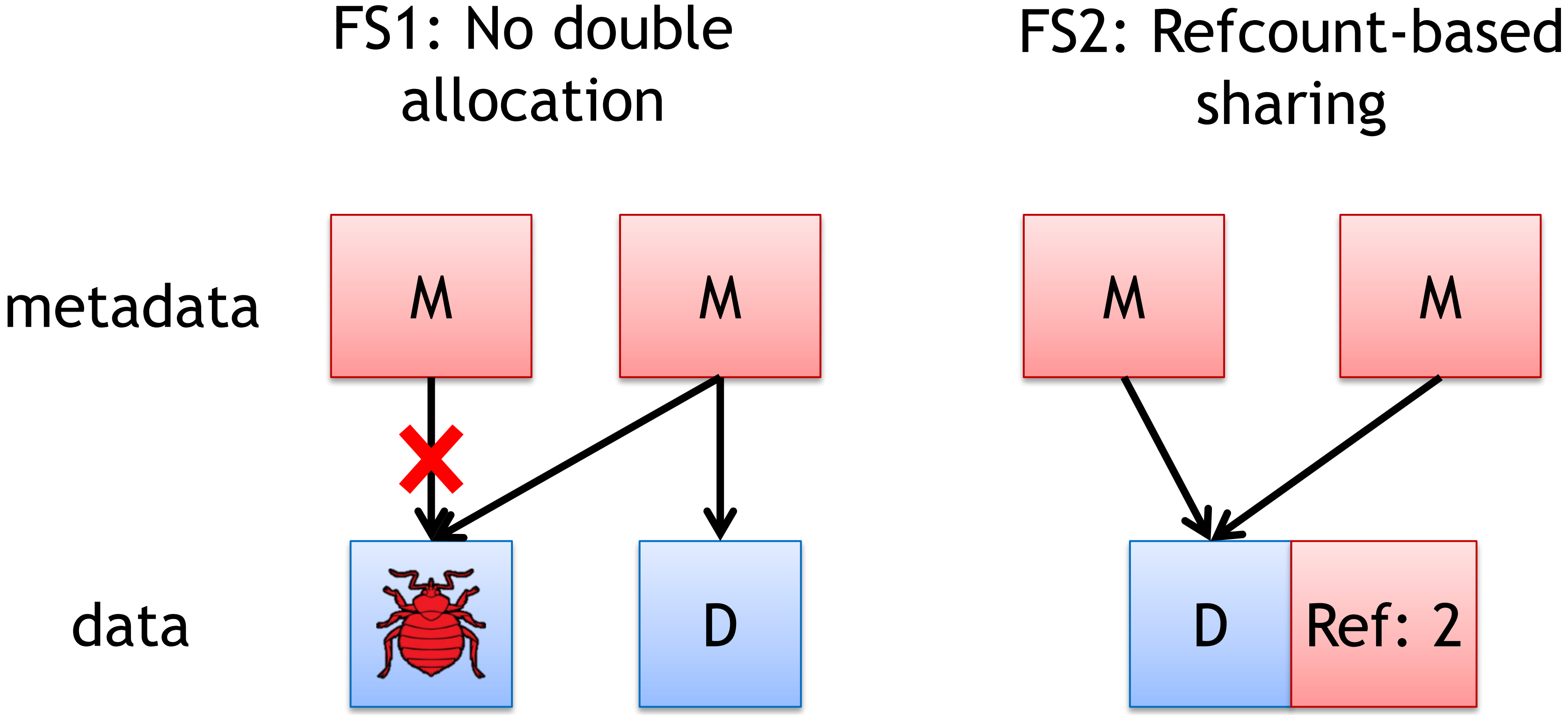
Existing approaches assume file systems are correct



None of these protect against bugs in file systems

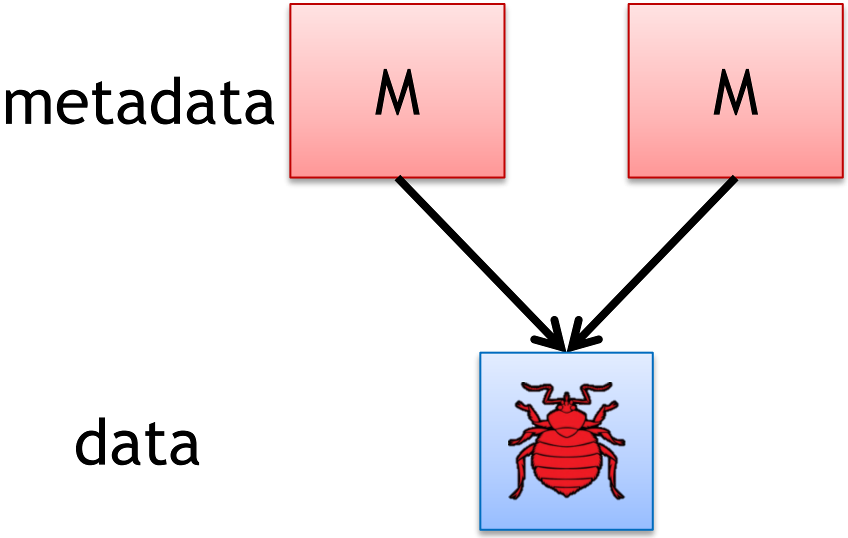
Offline Checking

- Check consistency offline, e.g., fsck
  - Consistency properties necessary for correctness



Problems with Offline Checking

- Slow, getting slower with larger disks
- Requires taking file system offline
- After the fact, repair is error prone



Outline

- Problem
  - Metadata can be corrupted by bugs and existing techniques are inadequate
- Our Solution: Recon
  - a system for protecting metadata from bugs
- Key idea
  - Runtime consistency checking
- Design
- Evaluation

Runtime Consistency Checking

- Ensure every update results in a consistent file system
- Makes repair unnecessary!
  - “What happens in DRAM stays in DRAM”

BUT

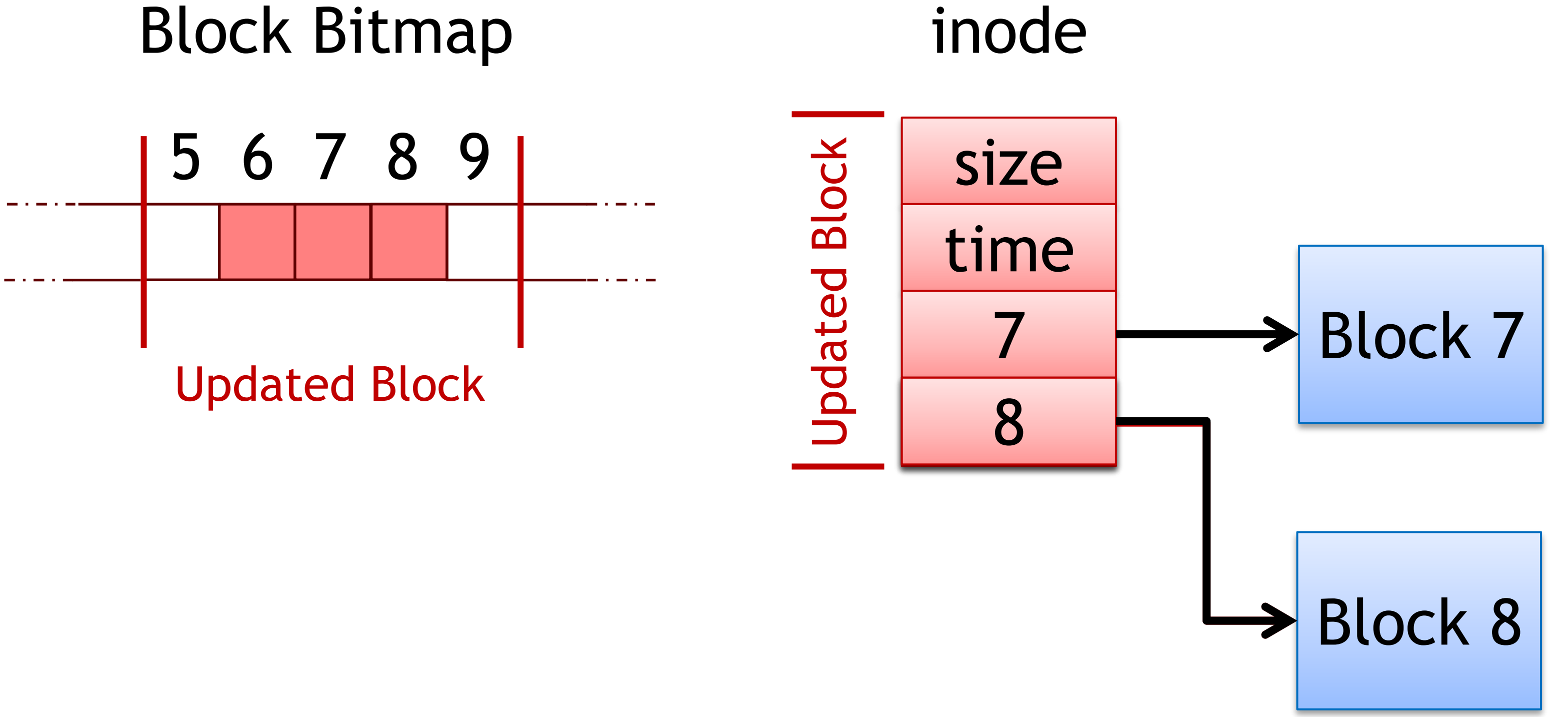
- Consistency properties are global
- Global properties require full scan
- We can’t run fsck at every write

Consistency Invariants

- We transform global consistency properties to fast, local consistency invariants
- Assume initial consistent state
  - New file system is clean
  - Use checksums/redundancy to handle errors below FS
- At runtime, check only what is changing
  - Do so before changes become persistent
- Resulting new state is consistent

Example: Block Allocation in Ext3

- Ext3 maintains a block bitmap – every allocated block is marked in the bitmap



Example: Block Allocation in Ext3

- Consistency Invariant

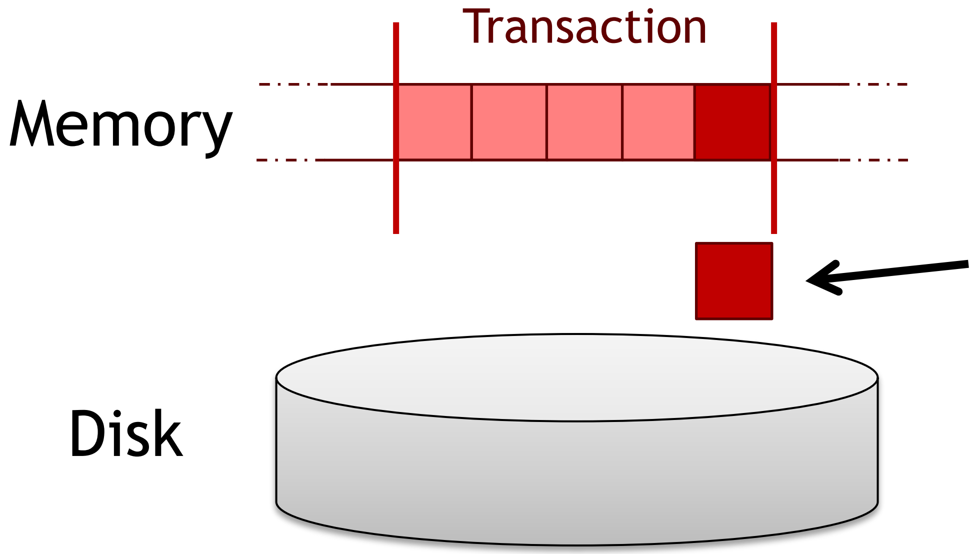
Bitmap bit X flip from “0” to “1” ↔ Block pointer set to X

- Invariant fails if either update is missing
  - Should not mark allocated without setting block pointer
  - Should not set block pointer without marking allocated

- Can any consistency property be transformed?
  - File systems should maintain consistency efficiently

When to Check Invariants

- Invariants involve changes to multiple blocks
  - When should they be consistent?
- Transactions are used for crash consistency
- Consistency can be checked at transaction boundaries

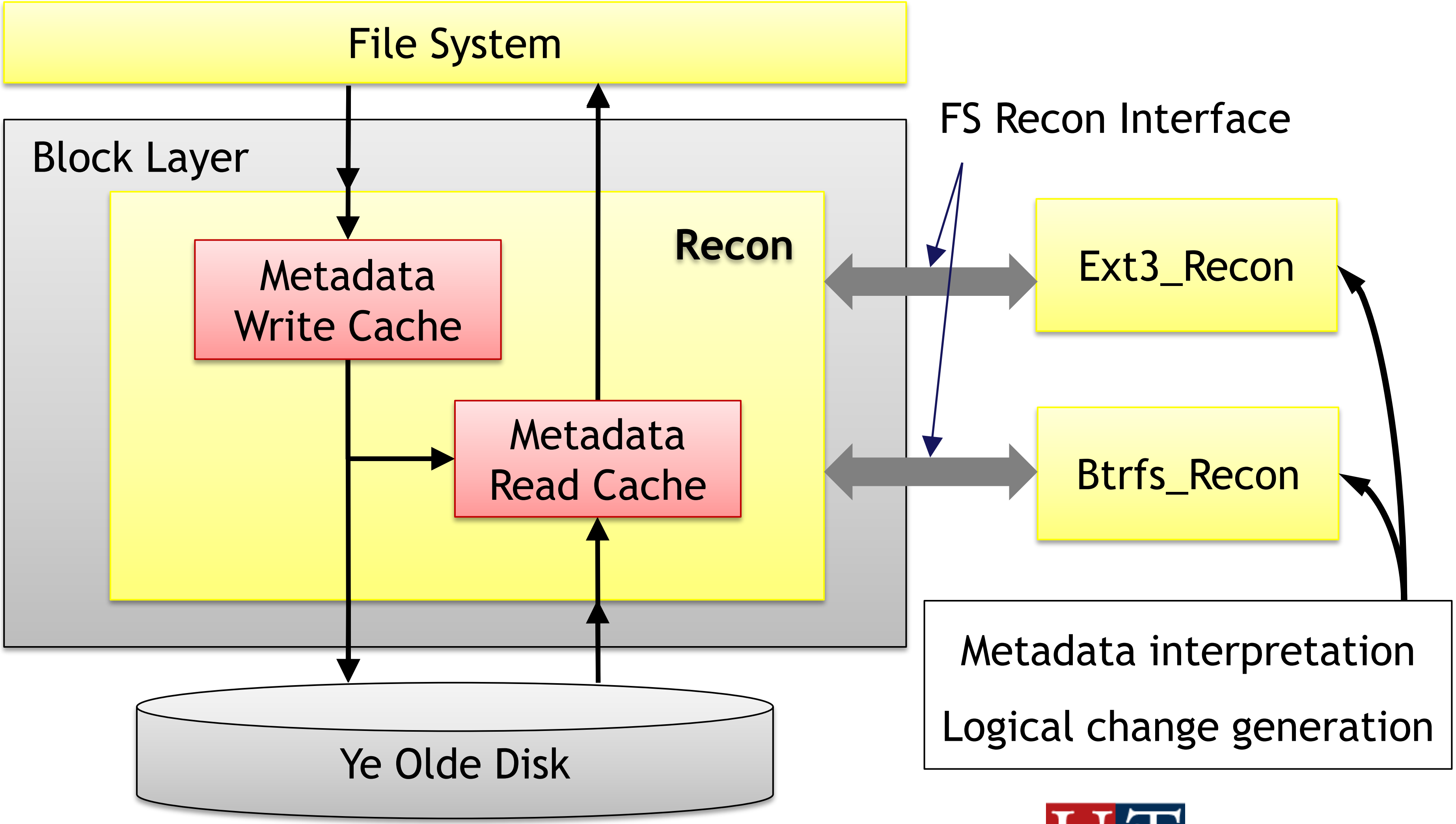


Must check transaction just before commit block reaches disk

Outline

- Problem
  - Metadata corruption cause by bugs
- Solution
  - Recon
- Key idea
  - Runtime checking
- Design
  - Metadata interpretation
  - Logical change generation
- Evaluation

The Recon Design



Metadata Interpretation

- To check invariants, we need to determine the type of a block on a read or write

- Take advantage of tree structure of metadata
- Superblock is the root of the tree
- Parents are read before children
  - For example, inode is read before indirect blocks
  - We see the pointer to the block before the block, and
  - The pointer within the parent determines the type of the child block

Logical Change Generation

- Invariants are expressed in terms of logical changes to structures, e.g., bitmaps, pointers

Bitmap bit X flip from “0” to “1” ↔ Block pointer set to X

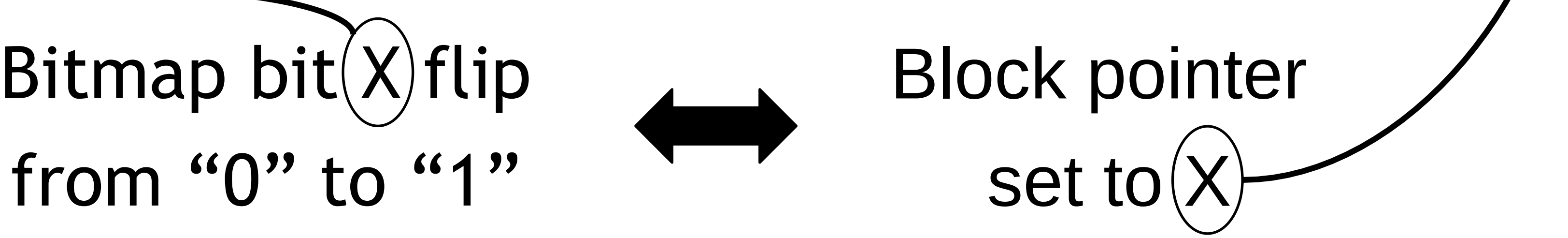
- Recon generates these changes based on
  - Block types
  - Comparing the blocks in the write and read cache
- Logical changes to metadata structures are represented as a set of change records:

[type, id, field, old, new]

Checking with Change Records

type	id	field	oldval	newval
inode	12	blockptr[1]	0	501
inode	12	i_size	4096	8192
inode	12	i_blocks	8	16
Bitmap	501	--	0	1
BGD	0	free_blocks	1500	1499

Transaction appends a new block to inode 12



Bitmap bit X flip from "0" to "1" ⟷ Block pointer set to X

Outline

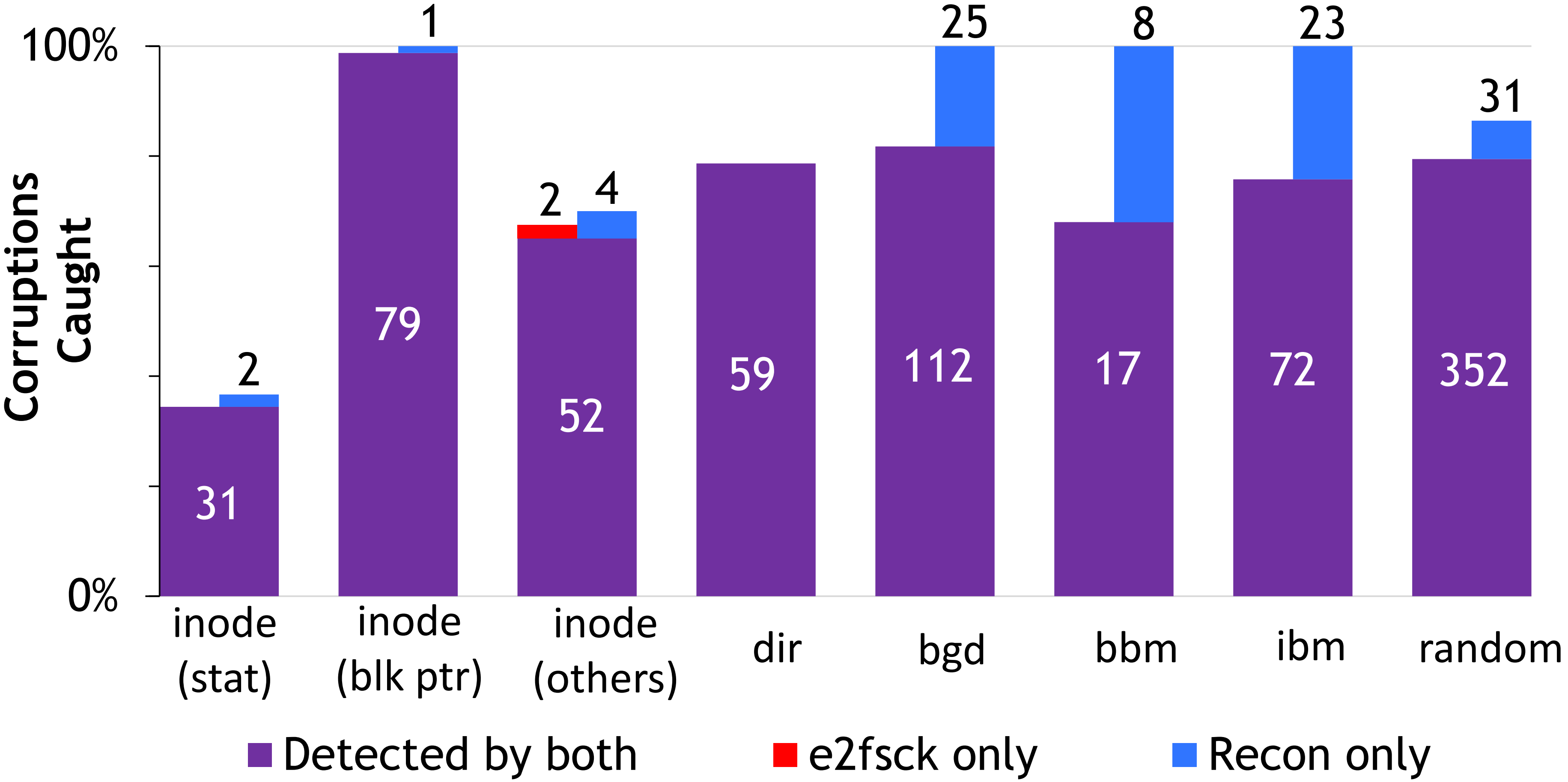
- Problem
  - Metadata corruption cause by bugs
- Solution
  - Recon
- Key idea
  - Runtime checking
- Design
- Evaluation
  - Complexity
  - Corruption detection
  - Performance overhead

Complexity

- Much simpler than FS code
  - Only need to verify result of file system operations
  - Each invariant can be checked independently

- Code divided into three sections
  - Generic Recon framework: 1.5 kLOC
  - Ext3 metadata interpretation: 1.5kLOC
  - 31 Ext3 invariants: 800 LOC

Corruption Detection



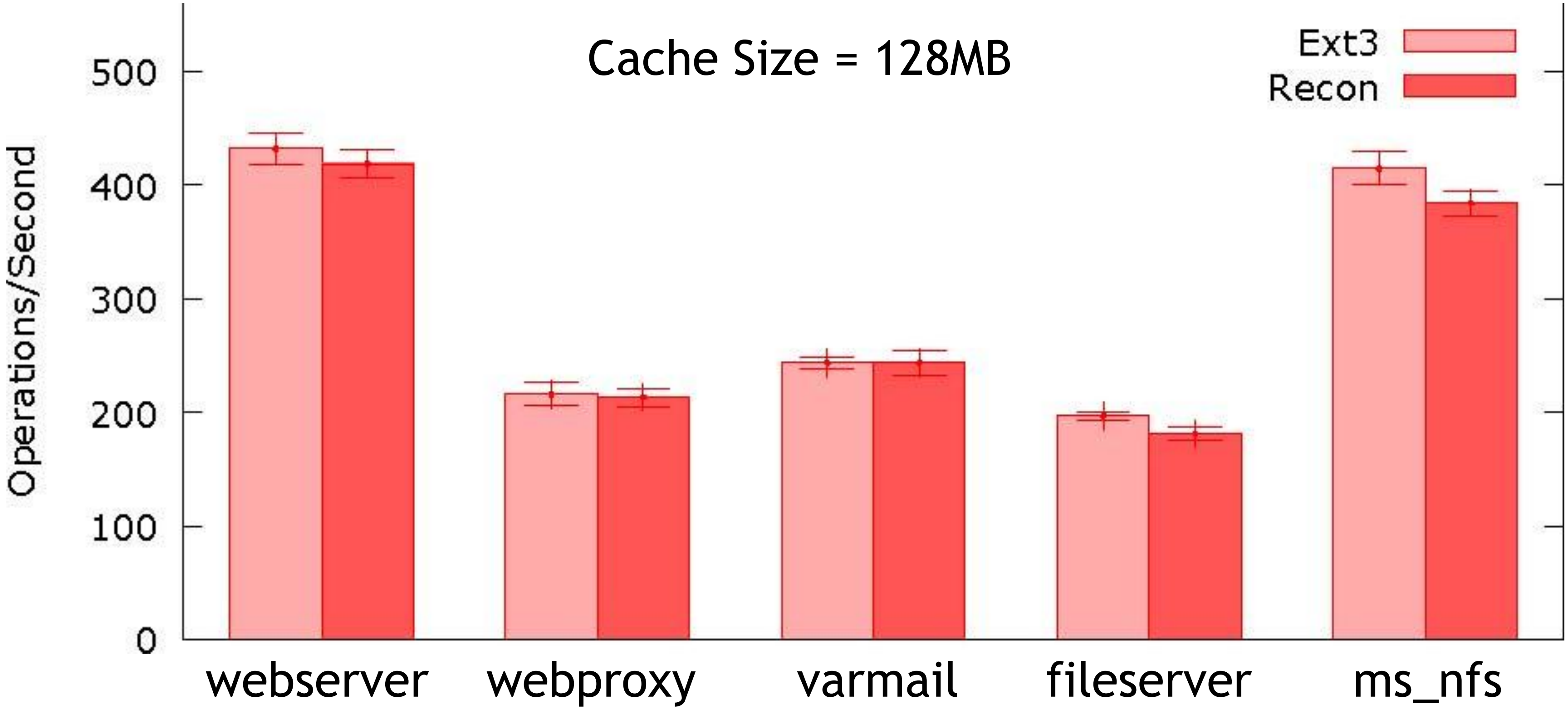
Recon matches e2fsck

Performance Evaluation

- Used Linux port of Sun's FileBench
  - Used 5 different emulated workloads
    - webserver, webproxy, varmail, fileserver, ms_nfs
    - ms_nfs configured to match metadata characteristics from Microsoft study (FAST'11)

- 3 GHz dual core Xeon CPUs, 2 GB RAM
- 1 TB ext3 file system

Performance Evaluation



For reasonable cache sizes, performance impact is modest

Handling Violations

Several options

- Prevent all writes, remount read-only
  - Preserves correctness
  - Reduces availability
- Take snapshot of filesystem and continue
  - Minimal availability impact, snapshot is correct
  - Requires repair afterwards
- Micro-reboot file system or kernel
  - Transparent to applications
  - Overcomes transient failures

Conclusion

- All consistency properties of fsck can be enforced on updates without full disk scan
  - Checking can be done outside the file system, entirely at the block layer

- Preventing corruption from being committed is a huge win over after-the-fact repair!

Thanks!

- To our anonymous reviewers
- To our shepherd, Junfeng Yang
- To the Systems Software Reading Group @ U of T

For their many insightful comments & suggestions!

- To Vivek Lakshmanan

For early insights that helped start the project!

This work was supported by NSERC through the Discovery Grants program