Five Years of Reliability Research

Remzi H. Arpaci-Dusseau
Professor @ Wisconsin-Madison

Why Storage Systems Are Broken (and What To Do About It)

Remzi H. Arpaci-Dusseau
Professor @ Wisconsin-Madison

What is a File System?

Persistent storage for data

Methods to name and organize data

Used in many settings

- Desktop
- Server

What is the state of the art?

- From FFS to modern systems

File System Innovations

Performance

- Caching, buffering, scaling, ...

Crash consistency

- Logging, copy-on-write, soft updates, ...

Functionality

- Search, ...

But what about **Reliability?**

This Work Began When...

We noticed the following:

- Disks seemed to be failing in new and interesting ways
- File systems seemed to be reacting to these failures in an odd manner

And thus...

An area of ignorance for us:

How do file systems react to disk failures?

Outline

Part I: How do disks actually fail?

Part II: How do systems react to failure?

Part III: Why is fault-handling so difficult?

Part IV: How can we do better?

Part I:
How Disks Fail

State of the art

Anecdotal

- Academics: Little information
- Web: Bad motherboard problem, etc.
- Industry: Depends whom you ask

Most sources agreed

- Disks failed in interesting ways
- But little hard data

Method

NetApp AutoSupport database

- Filers phone home periodically
- Huge amount of data on disk failures

Snapshot studied: [Bairavasundaram '07, '08]

- ~1.5 million disks in many environments
- 3 years of data

Types of Errors

Latent Sector Errors (LSEs)

- A single block read/verify/write returns a failure, whereas rest of disk is "working"
- Causes? Media scratch, bits flipped, etc.

Block corruption

- Disk returns wrong contents for block
- Causes? Faulty controller, bad bus, etc.

Result summary

Number of problems during period of study:

- Latent sector errors cheap: 9.4% - costly: 1.4%
- Block corruption cheap: 0.5% - costly: 0.05%

Also observed

- Spatial and temporal locality
- LSEs increase over time, with size
- Corruption not independent across disks in RAID

And some interesting other behaviors

- The block number problem, the cache-flush bug

Errors: Full Summary

LSEs

SCSI with >=1 error are as likely to develop additional errors as SATA

Most models: annual error rate increases in year 2 (for SATA, sharp increase)

LSEs increase with disk size

Most disks have <50 errors

Not independent: disk with errors more likely to develop additional errors

Significant amount of spatial and temporal locality

Disk scrubbing useful (60% LSEs discovered this way)

Enterprise: high correlation between recovered errors and LSEs

SATA: high correlation with not-ready errors

Corruption

Probability of checksum errors varies greatly across models within same disk class

Age affects are different (but fairly constant with age)

Disk size: little effect

Workload: little effect

Most with corruptions only have a few (a small # have many)

Corrupt SCSI develop many more corruptions than corrupt SATA

Not independent within disks

Not independent ACROSS disks in RAID

Spatial locality (but for consecutive blocks)

Some temporal locality

Weak correlation with LSEs, not-ready errors

Scrubbing detects most checksum mismatches

Conclusions

Partial failures are reality

- Not just whole-disk failure anymore

Fail-partial failure model [Prabhakaran ’05]

- Entire disk may fail
- Single block may fail
- Single block may become corrupt

Part II:
How File Systems React To Failure

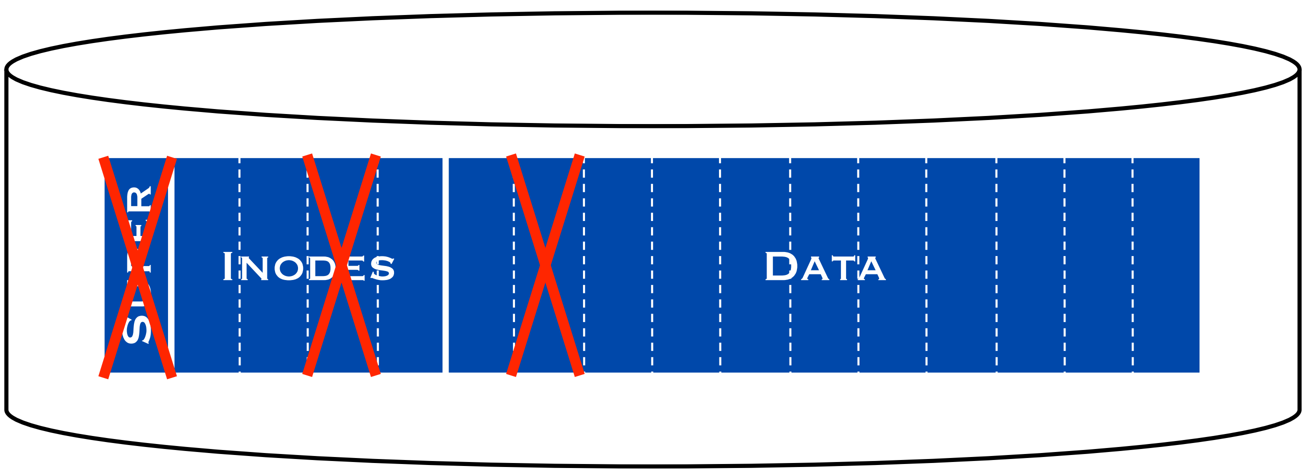
Type-Aware Fault Injection

Observation: File systems comprised of many different on-disk structures

- Superblocks, inodes, etc.

Idea: Make fault injecting layer aware of them

- Inject faults across all block types



The File Systems

Linux ext3

- Most popular, “FFS-like” + journaling

ReiserFS

- Entirely different lineage, lots of trees

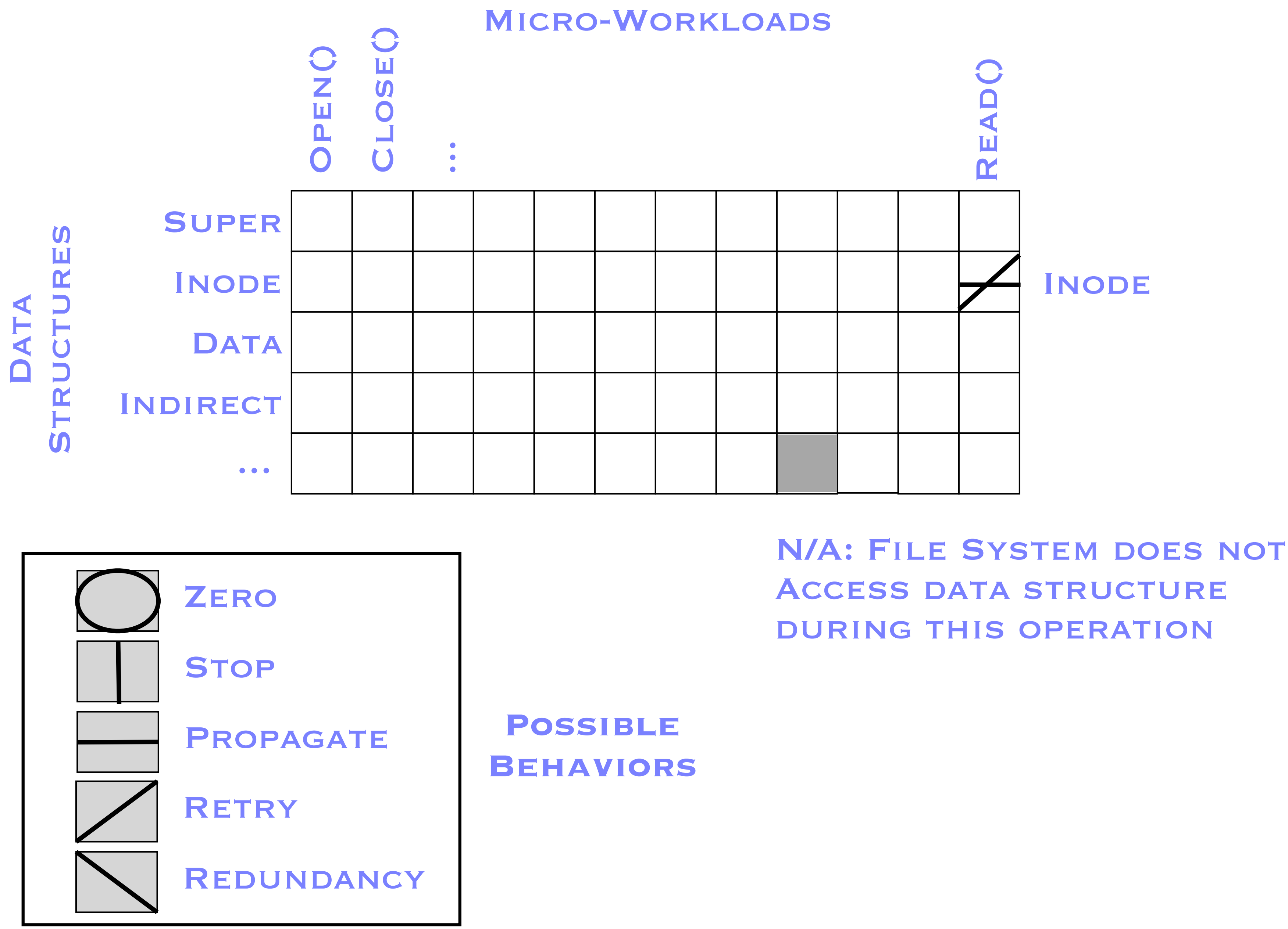
IBM JFS

- IBM’s journaling file system

Windows NTFS

- Commercial, not Linux

Result Matrix



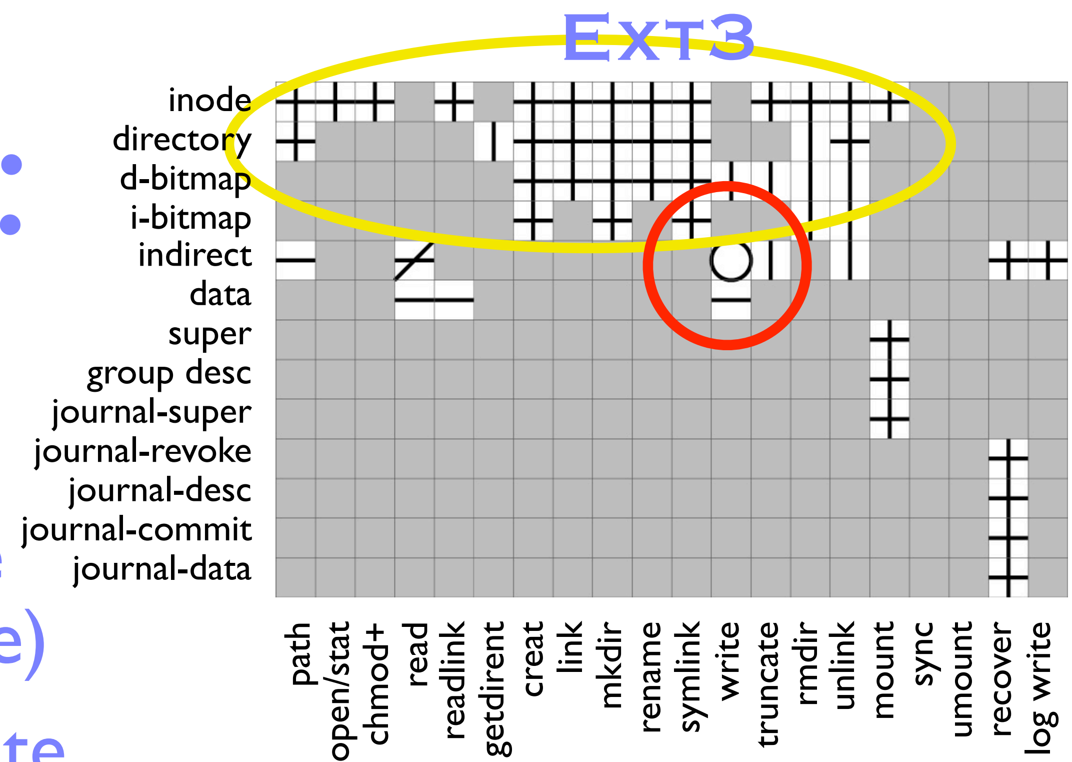
Read Errors: Recovery

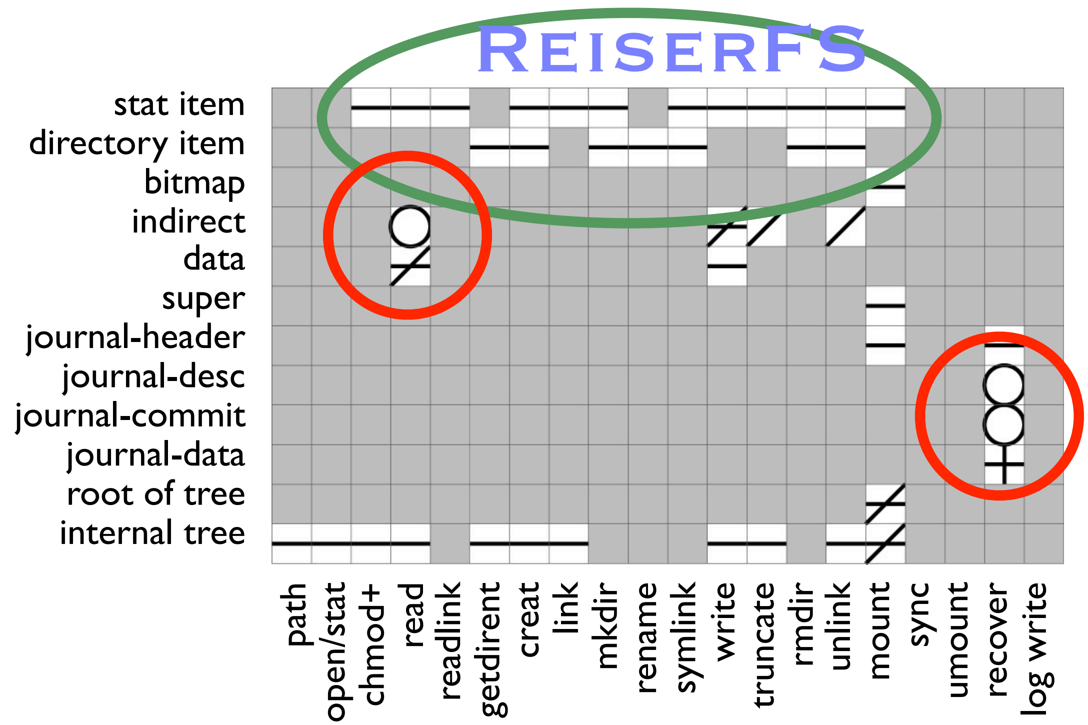
Ext3: Stop and propagate (don't tolerate transience)

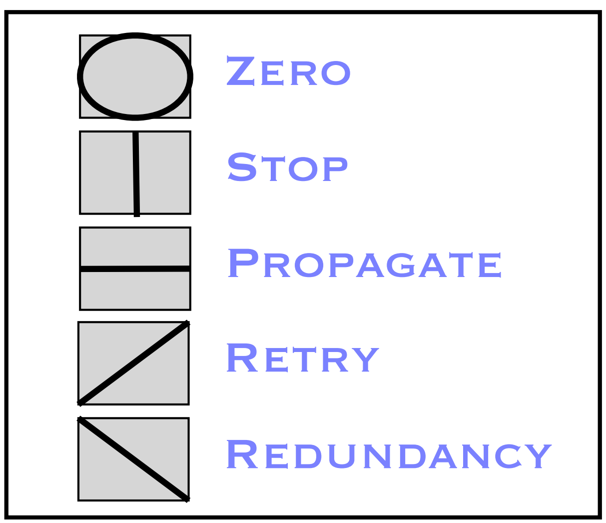
ReiserFS: Mostly propagate

JFS, NTFS (not shown)

All: Some cases missed







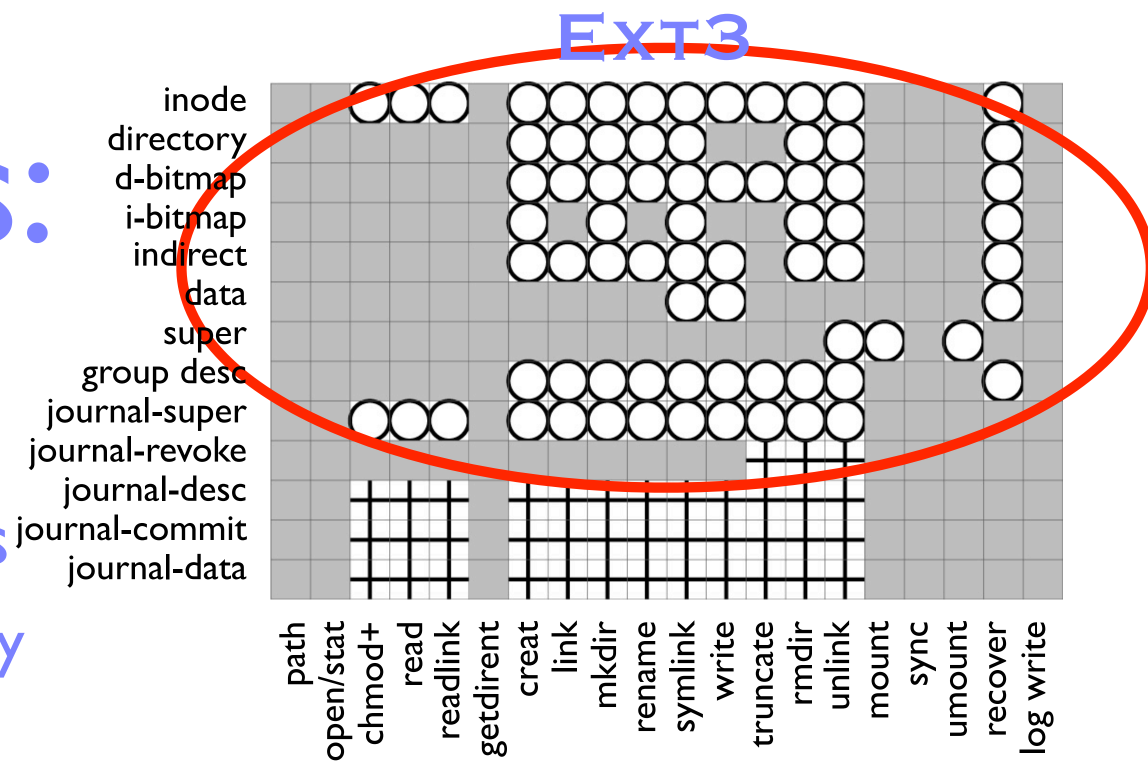
Write Errors: Recovery

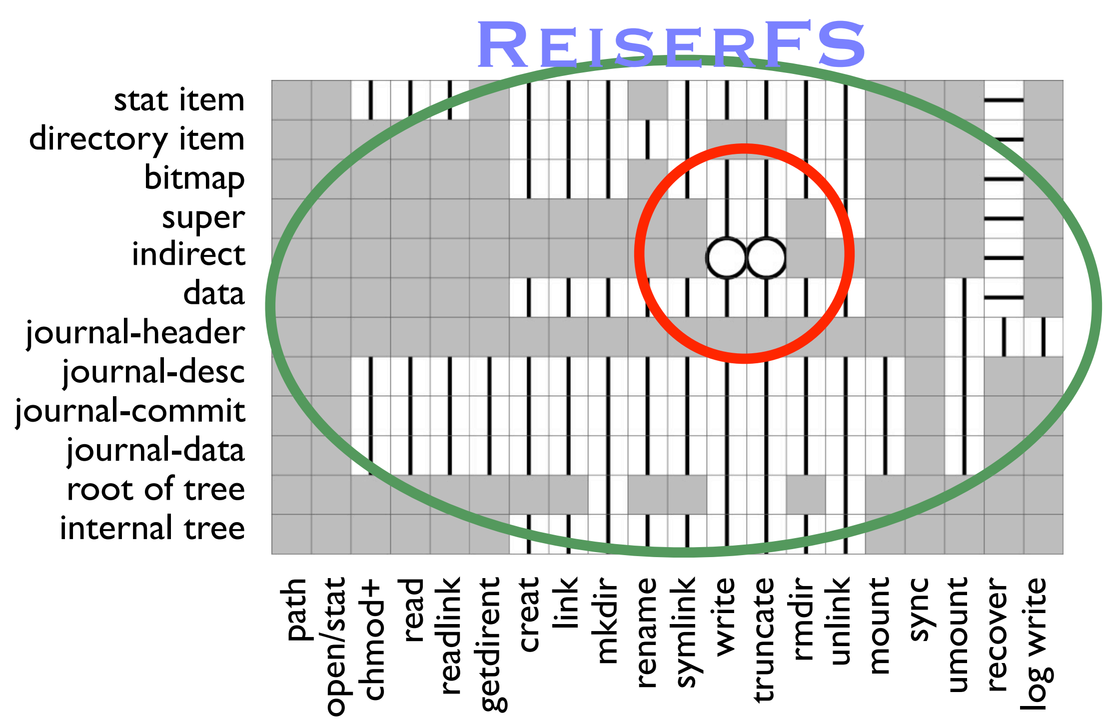
Ext3/JFS: **Ignore** write faults

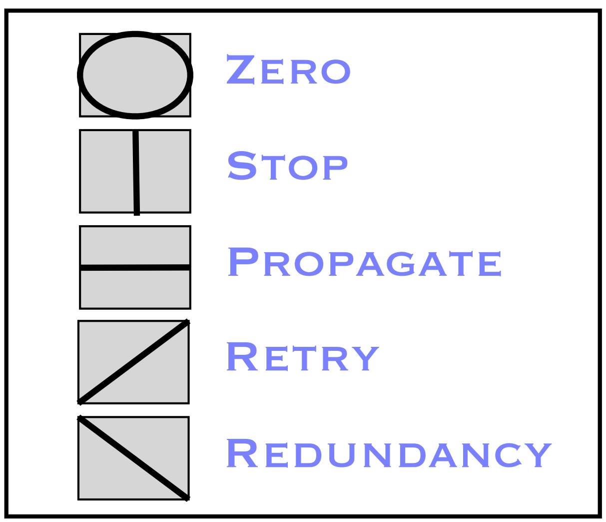
- No detection → no recovery
- Can corrupt entire volume

ReiserFS always calls **panic**

- Exception: indirect blocks







File System Results

Ext3: **Simple (but hypersensitive)**

- Overreacts on read faults (halt)
- Write faults: ignored

ReiserFS: **First do no harm**

- Write fault means panic()
- Integrity but at loss of availability

JFS: **The kitchen sink**

- If it can be done, JFS tries to do it

NTFS: **Try, try again**

- Liberal retry policy

More Generally

Illogical inconsistency

- Hard to make sense of policies
  (not easy to specify; scattered through code)

Bugs are common

- Lots of missed cases, code is rarely run
  (getting recovery right is hard)

It's the file system, not the disks

- Even though disks misbehave, the
  software in charge of them was worse

Part III:
Why Fault-handling is Challenging

Part III: Outline

Static analysis [FAST-08, PLDI-09]

- Linux file systems

Modeling failure [FAST-08]

- Commercial RAID designs

Error Propagation

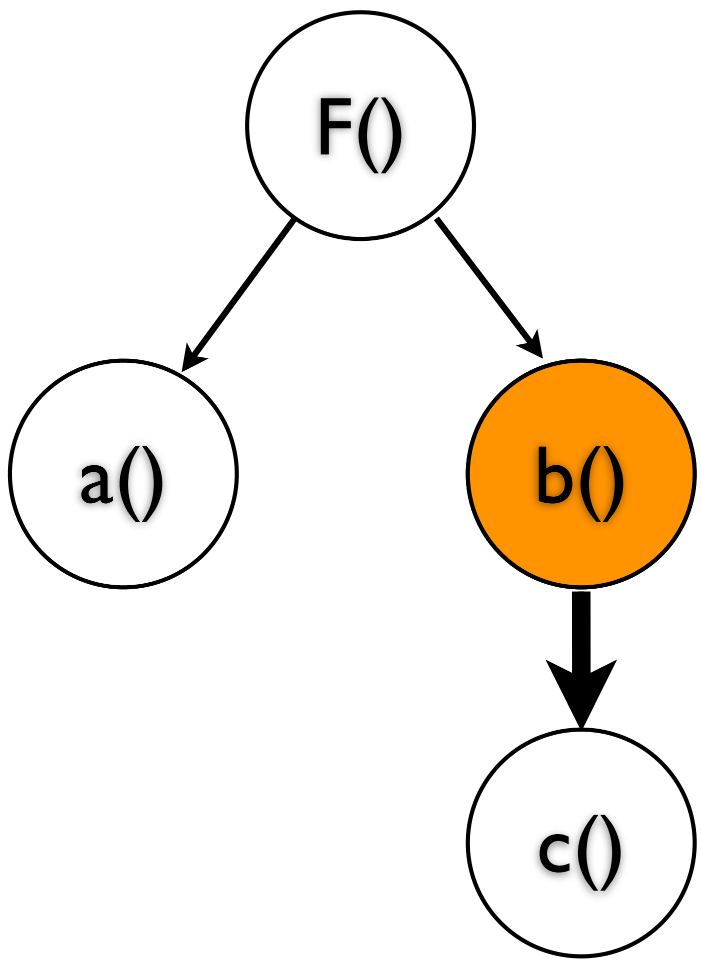
```
// fs/block.c
int sync_blockdev (block_device*) {
    int ret = 0, err;
    ret = filemap_fdatawrite ();
    err = filemap_fdatawait ();
    if (!ret)
        ret = err;
    return ret;   // PROPAGATON E.C.
}
// fs/jbd/recovery.c
int journal_recover (journal*) {
    int err;
    ...
    sync_blockdev (); // E.C. UNSAVED
    ...
    return err;
}
```

EDP: Tool To Analyze Error Propagation

Static analysis: Built using **CIL** [Necula '02]

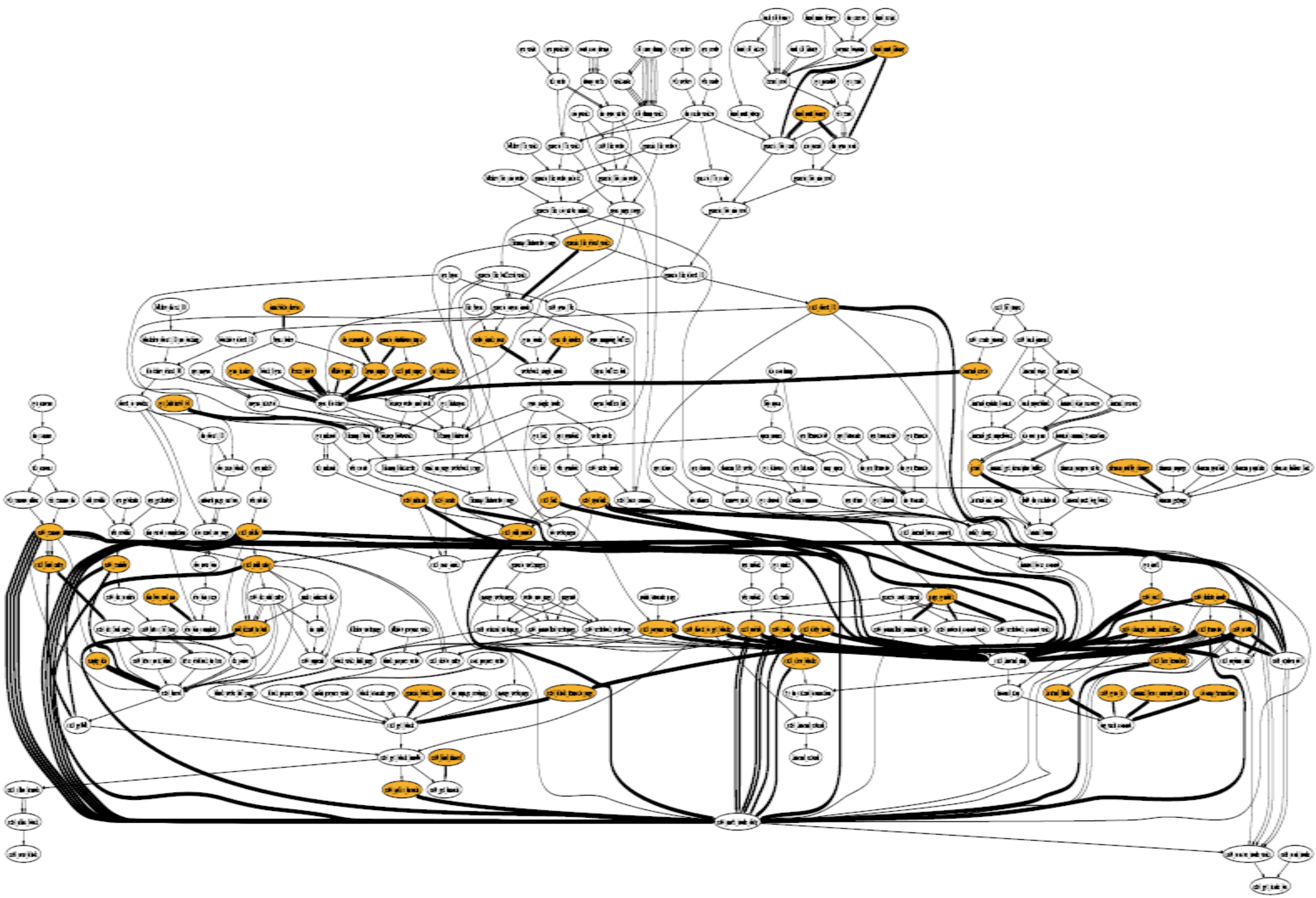
- Start with error codes
- Use dataflow analysis to trace where integer codes are "handled"
- Mark broken channels (where error is lost or overwritten)

Results: Annotated CFGs

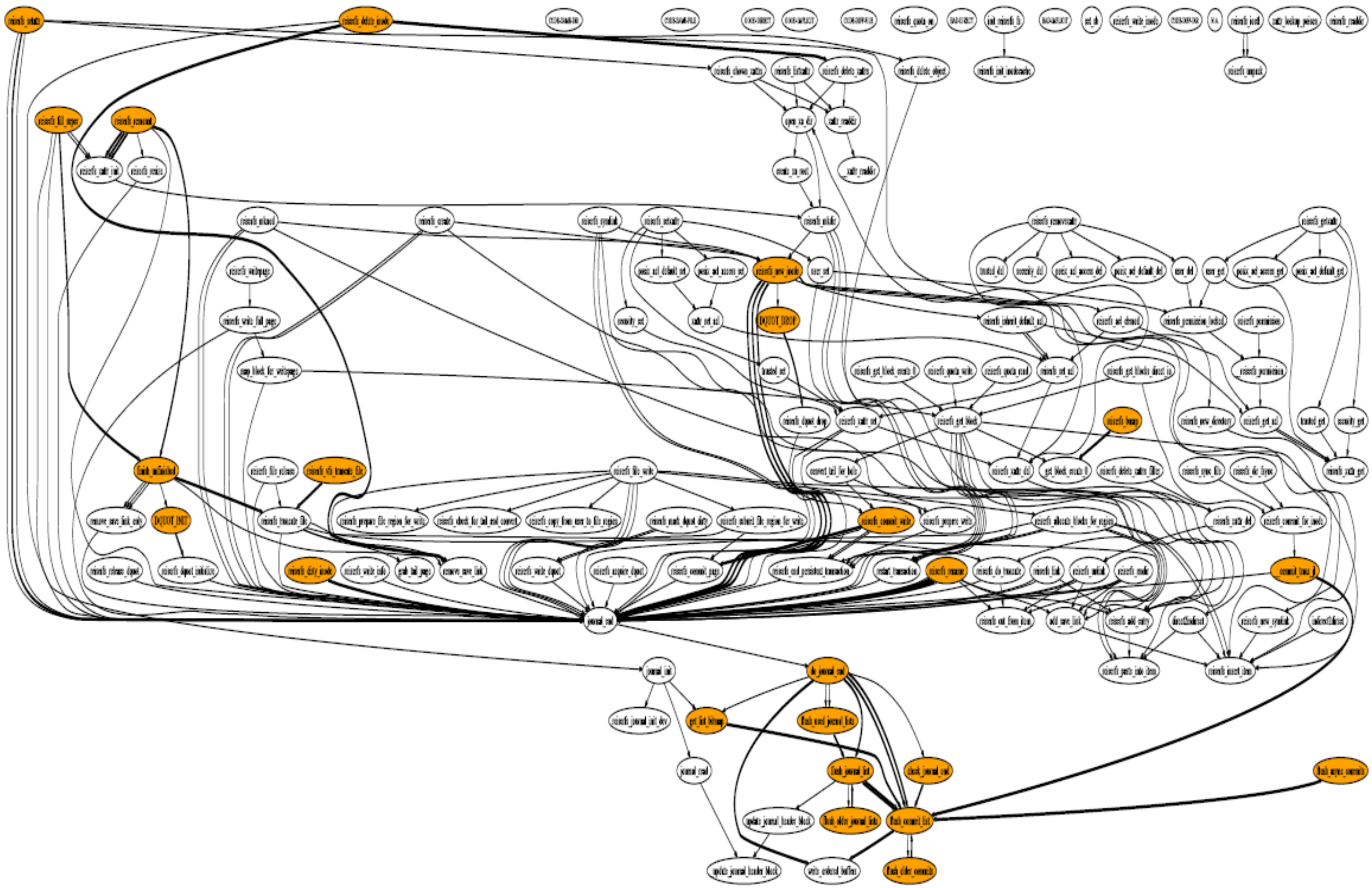


EIO: b() calls c(), but handles error code improperly

ext3



ReiserFS



SGI XFS

Anonymous reviewers said:

"What else to do but to stare in slack-jawed awe?"

"I asked a colleague in software engineering about his thoughts about the XFS graph, and he said you can't conclude much from it, except perhaps to say that XFS sucks."

EDP Summary

Our study

- Static analysis: Can find error-flow problems
- Ran tool on 51 Linux "file systems"

Sloppy error handling yields sloppy FS

- About 10% of calls drop errors

Part III: Outline

~~Static analysis~~ [FAST-08, PLDI-09]

- ~~Linux file systems~~

Modeling failure [FAST-08]

- Commercial storage designs

Data Integrity Techniques

Scrubbing scans data+parity in background

- To find and fix errors ASAP

Checksums for integrity

- Per sector, per block, in parent

Write verification

- After write, read back to ensure on disk

Extra ID

- Logical, physical

RAID Designs

System	RAID	scrubbing	sector csum	block csum	parent csum	write verify	physical ID	logical ID	other
Adaptec 2200S	x								
Linux Software	x	x							
Pilot								x	x
Tandem NonStop	x		x				x		
Dell Powervault	x	x	x						x
Hitachi Thunder 9500	x		x			x			
NetApp Data ONTAP	x	x		x		x	x	x	
ZFS + RAID-4	x	x			x				

Every design had corner cases where data was lost

Part IV: Towards Robust File and Storage Systems

Outline for Part IV

Approach #1: Higher-level Design

- SQCK [OSDI-08]
- I/O Shepherding [SOSP-07]

Approach #2: Assume Bugs Exist & Cope

- EnvyFS [USENIX-09]
- Membrane [FAST-10]

File System Checking

Check and repair (aka **fsck**)

- Turn corrupt image into consistent image
- Virtually all FS's (eventually) have one

Tough properties

- Rarely run
- Absolutely has to work correctly

Building fsck: State of the Art

Write lots of C code

Test it

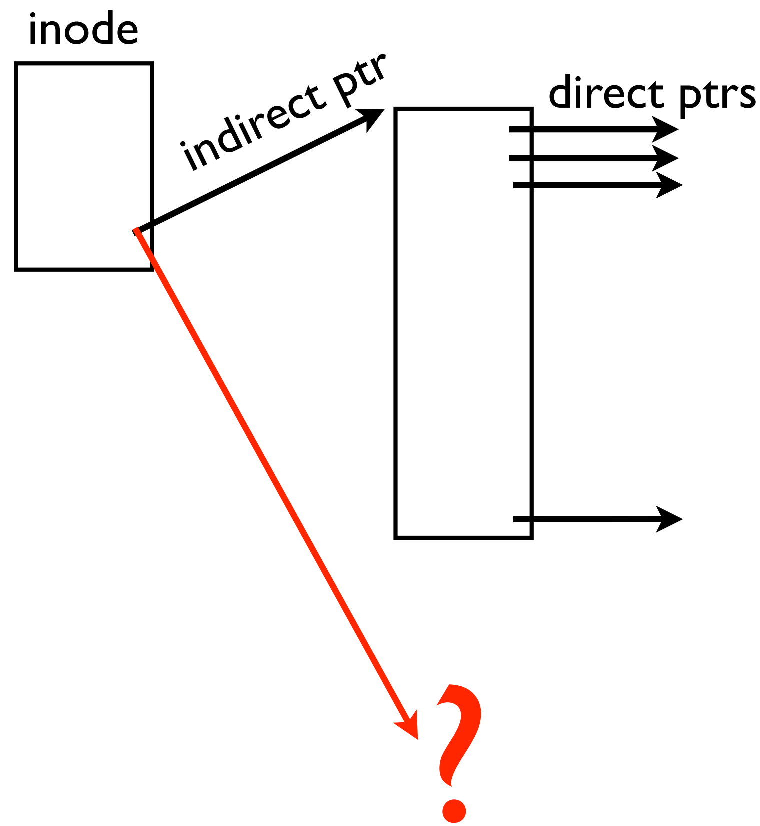
Tell customers to take frequent backups

First step: **Measure** existing ext* checker

Misordered Repair

Typical fix: Clear bad pointers

- "bad": outside of valid range



Problem: **Misordered repair**

- Trusts indirect pointer
- Clears pointed-to block

Kidnapping Problem

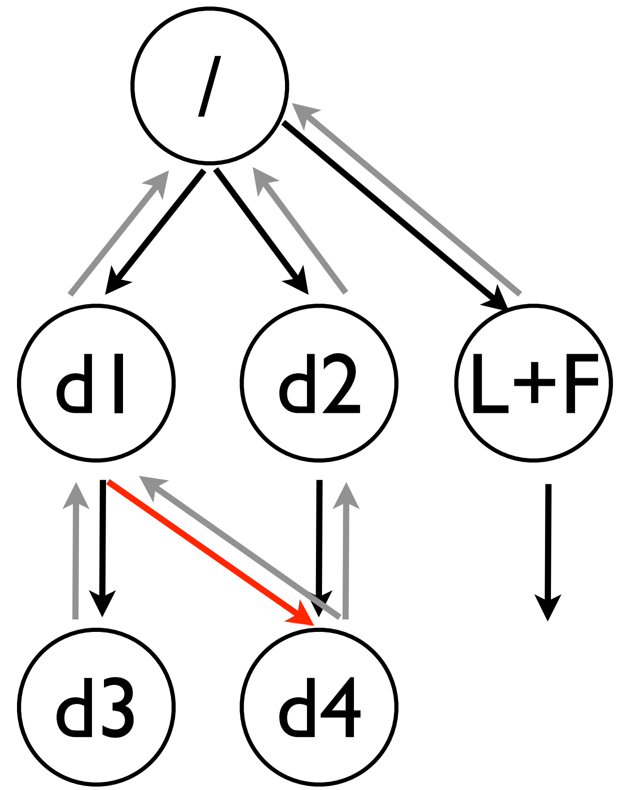
Corrupt single inode number (in d1)

But result is surprising

- Child d3 is lost, d4 kidnapped!

Information-incomplete repair

- Doesn't use all info to make best possible repair



SQCK (squeak)

Declarative checker [OSDI '08]

- 100s of SQL queries
  (not 1000s of lines of C)
- Simpler to understand,
  simpler to modify
- Not too slow (~same as fsck)

Simple

```
SELECT *
FROM   GroupDescTable G
WHERE  G.blkBitmap NOT BETWEEN
       G.start AND G.end
```

Finds block bitmap pointers that point outside the group (and are thus invalid)

Slightly Complex

```
SELECT *
FROM   ExtentTable X, SuperTable S
WHERE  S.copyNum = 1           AND
       X.type    = INDIRECT_PTR AND
      (X.start   < S.firstBlk   OR
       X.end    >= S.lastBlk)
```

Check for illegal indirect blocks

Is That My Child?

```
SELECT *
FROM   DirEntryTable P, DirEntryTable C
WHERE  P.entryIno  = C.ino    AND
       C.entryNum  = PARENT   AND
       C.entryIno <> P.ino
```

Check that parent/child agree on relationship
(P says C is its child, but C says otherwise)

Results

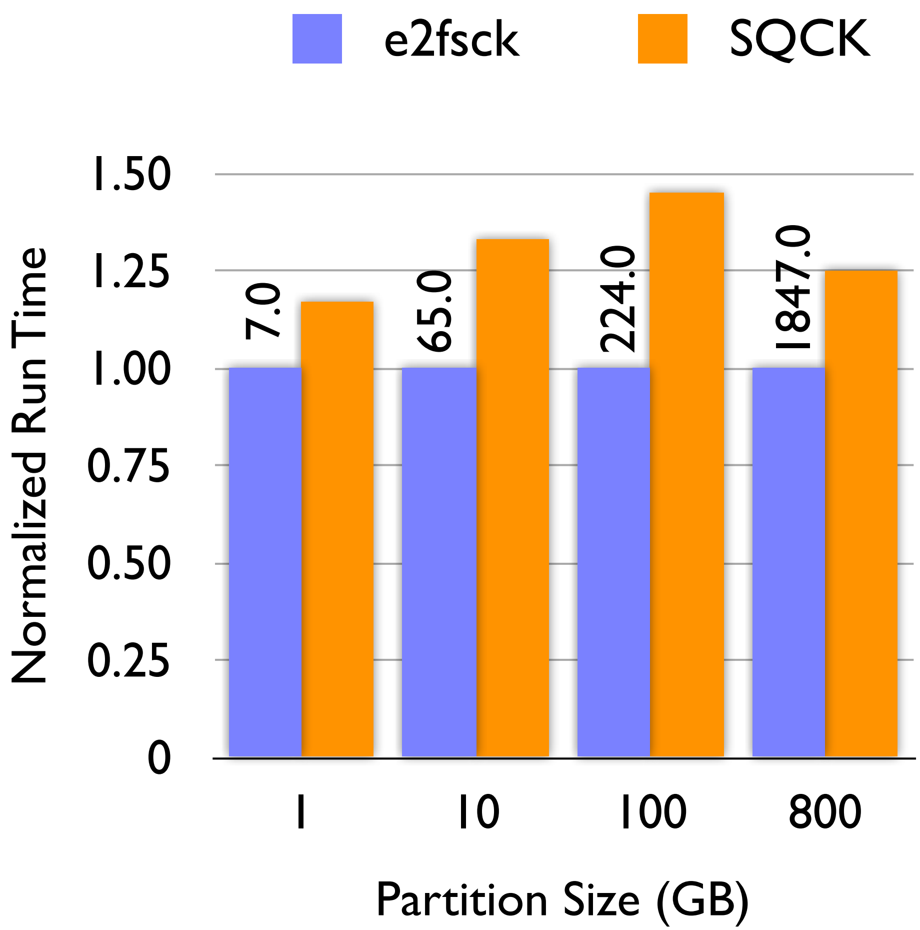
Complexity

Complexity: 121 repairs

- ~1100 lines of SQL code

Comparison: ~20-30K lines in e2fsck

Performance



Linux 2.6.12

MySQL 5.0.51a

2.2 GHz AMD Opteron

1 GB DRAM

1 TB WDC disk

Outline for Part IV

Approach #1: Higher-level Design

- SQCK [OSDI-08]
- I/O Shepherding [SOSP-07]

Approach #2: Assume Bugs Exist & Cope

- EnvyFS [USENIX-09]
- Membrane [FAST-10]

File system bugs: Here to stay

Could try to write a perfect file system

- Hard to do, even with modern tools

Likely reality: Imperfect file systems live on

Solution: N-versioning

Old idea [Avizienis '77]

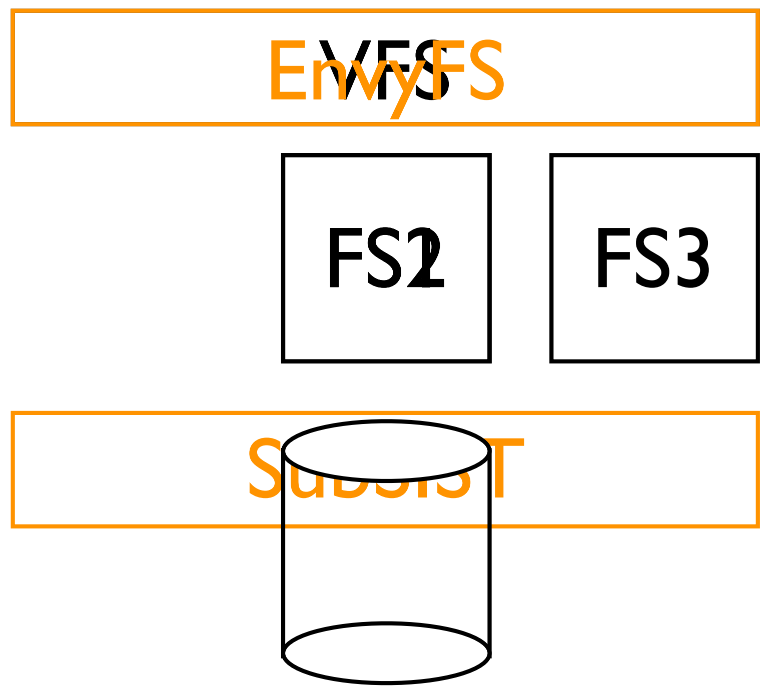
EnvyFS: For local FS

- Key: Can leverage Linux file systems

Problem: Overheads

- Time & Space

SubSIST: Single-instance store for EnvyFS



Technique: Comparator

Compare results from each FS operation

- Data struct comparison:
  inodes, superblocks, data, etc.

Special cases

- Directory - order not specified by VFS
  (thus read entire directory)
- Inode numbers - different across FSes
  (thus assign and map at EnvyFS level)

Optimizations

- Data blocks - only read and compare **two**

Hard Part (I): Crashes

Child file system crash may take down system

Full solution: Isolate each FS (not done here)

EnvyFS lightweight approach: Fail fast

- Redirect panic, BUG, BUG_ON to envyfs_child_panic()
- Simplest policy: Disable buggy child

Hard Part (II): Repair

Some simple repairs are automatic

- e.g., a child with one corrupt data block
- Solution: Overwrite bad block with correct value
- Result: Consistent file system

More complex repairs are challenging

- e.g., a branch of the FS tree is missing
- Current approach: Rebuild child from scratch

Hard Part (III): Overhead

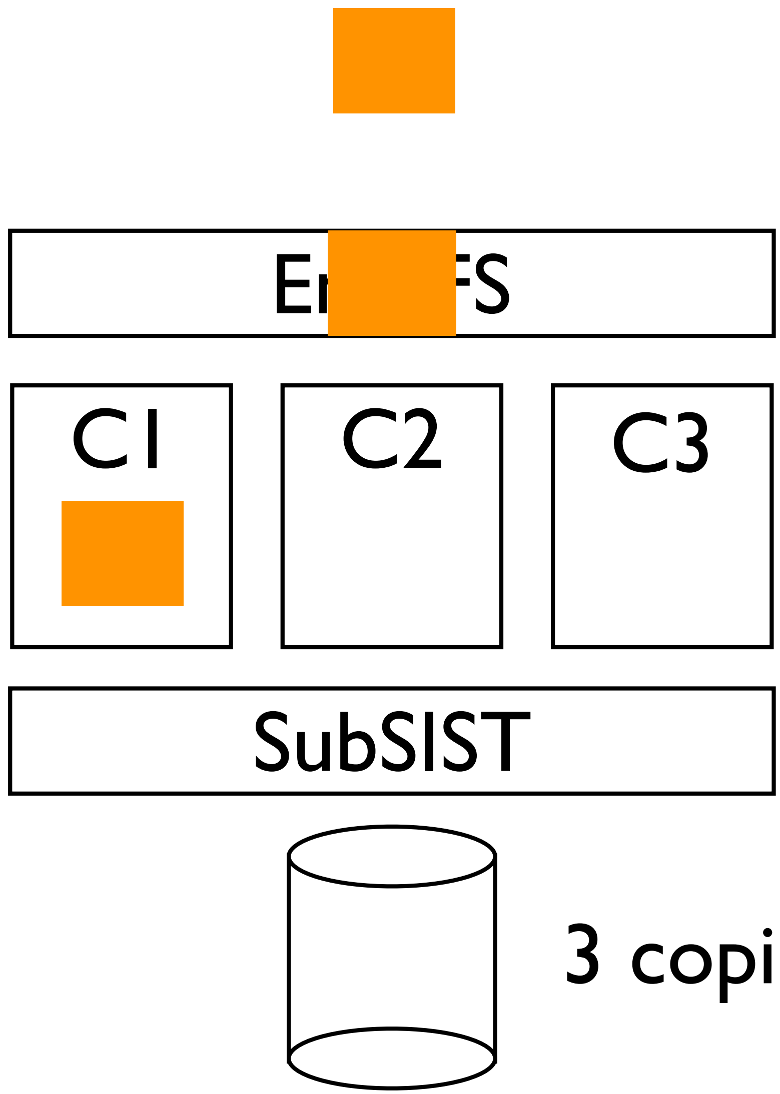
Three file systems means three data copies

- Time - have to access disk three times
- Space - have to keep three copies

Solution: SubSIST

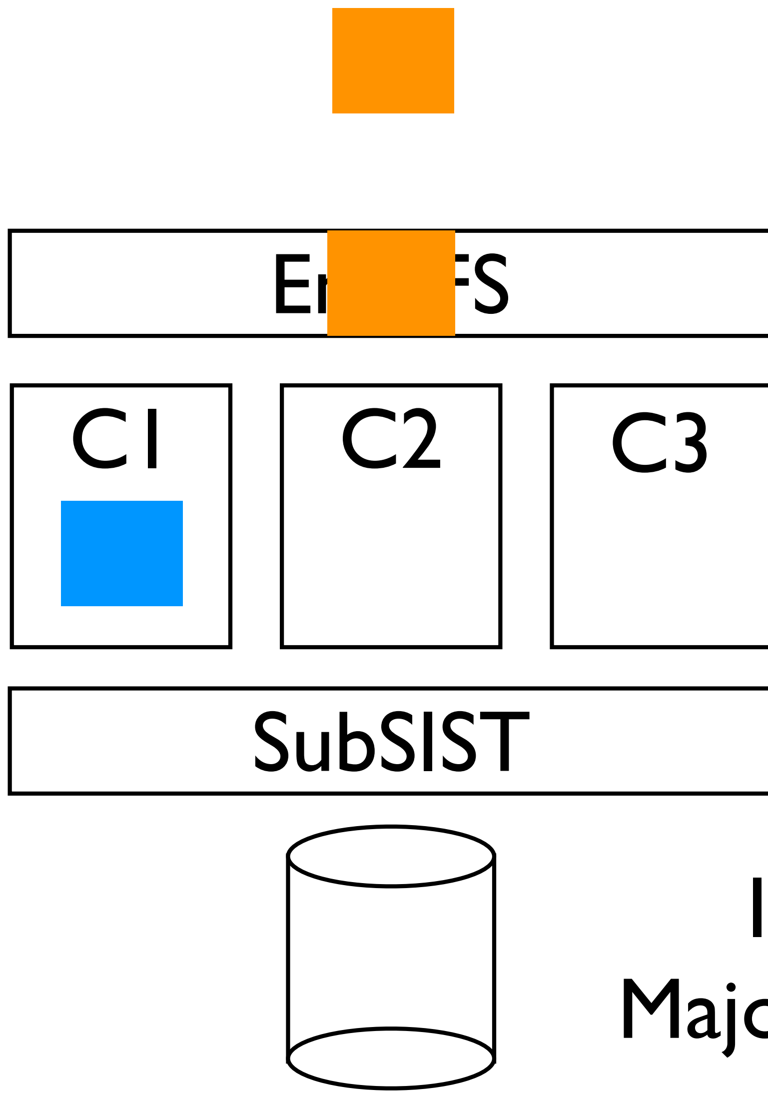
- Coalesce three copies into one transparently under each file system
- **Critical:** Can still detect/correct single faulty file system

SubSIST: Writing to Disk



3 copies coalesced to 1

SubSIST: Mistake Tolerance



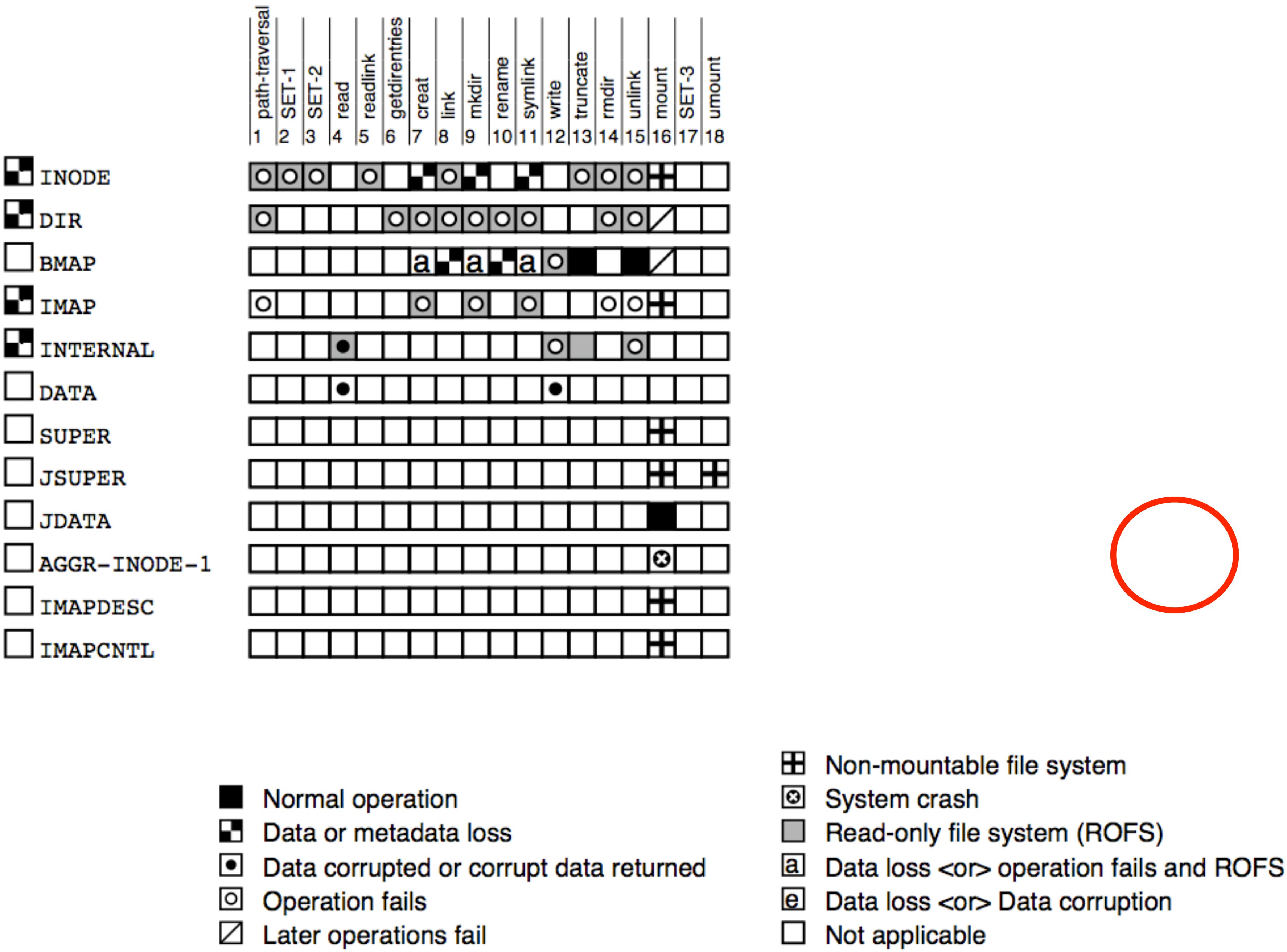
Corrupt!

Important:
Majority still rules

Evaluation

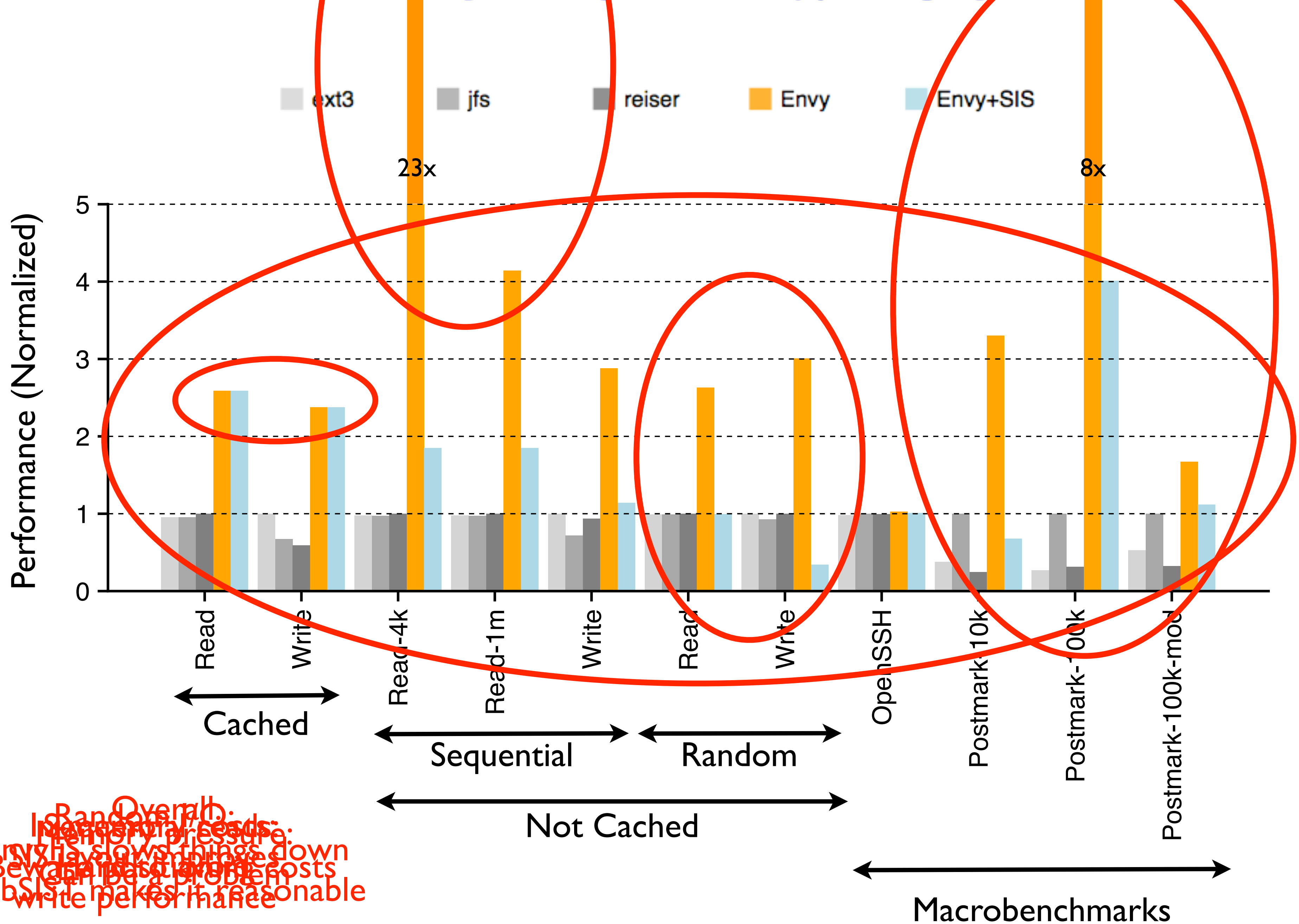
Robustness

(a) JFS



(need two more slides for full bar)

Performance



EnvyFS Conclusions

Old model

- Just fix bugs

New model

- Assume bugs exist,
  cope with their constant presence
- Not without cost (but slow > lost data?)

Final Thoughts

Research Lessons

Details matter: Small observation led to broad inquiry

Talk to industry: Source of "real" problems, source of data

Work to gain open-source "street cred": Linux ext4 story

Embrace ignorance: If you don't know it, maybe no one does

Listen/read broadly: Ideas are hard to come by; look around

Easiest interesting problem: Explore ideas w/o overcommitting

Right problem: Think hard about what problem you are solving

Measure then build: Don't solve before understanding it

Tell a story: And remember, story doesn't need to match reality

5 Years: A Summary

Problem we found: Reliability is 2nd-class citizen

- Disks fail in interesting ways...
- ... but **software is the main problem**
  - Design: Reliability added on, not built in
  - Implementation: Lots and lots of bugs

Need to rethink approach

- Higher-level systems design (SQCK, Shepherd)
- Assume bugs exist & cope (EnvyFS, Membrane)

Credits

Professors Andrea & Remzi Arpaci-Dusseau
(and Mike Swift and Ben Liblit)

Students (Past and Present)

- Lakshmi Bairavasundaram (PhD ’08, NetApp)
- Haryadi Gunawi (PhD ’09, Postdoc @ UCB)
- Vijayan Prabhakaran (PhD ’07, MSR SV)
- Nitin Agarwal (PhD ’09, NEC Research)
- Andrew Krioukov (BS ’08, Grad @ UCB)
- Swetha Krishnan (MS ’07, Cisco)
- Meenali Rungta (MS ’07, Google)
- Abhishek Rajimwale (M.S., ’10, DataDom.)
- Swami Sundararaman (PhD ‘??)
- Cindy Rubio-Gonzalez (PhD ’12)
- Sriram Subramanian (PhD ’11)

Want to learn more?

www.cs.wisc.edu/adsl