TECHNISCHE
UNIVERSITÄT
DRESDEN

Department of Computer Science Institute of System Architecture, Operating Systems Group

EIO: ERROR CHECKING IS OCCASIONALLY CORRECT

HARYADI S. GUNAWI, CINDY RUBIO-GONZÁLEZ, ANDREA C. ARPACI-DUSSEAU, REMZI H. ARPACI-DUSSEAU, BEN LIBLIT

CARSTEN WEINHOLD

MOTIVATION

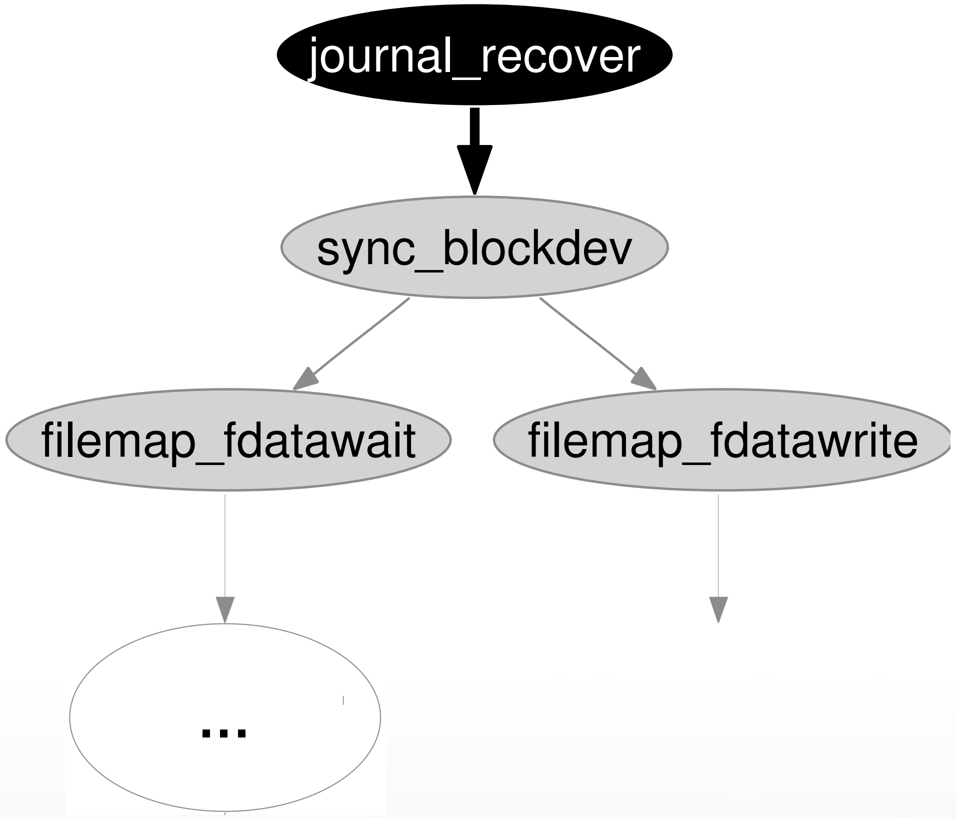
- File and storage systems must be robust
- **Previous research:** *„file systems are [...] unreliable when the underlying disk system does not behave as expected"*
- **Requirement:** comprehensive recovery policies need correct error reporting
- **Reality:** error propagation often incorrect
- Paper presents analysis error propagation in Linux code

EDP: APPROACH

- Error Detection and Propagation (EDP):
  - Static analysis of dataflow (error codes)
  - Uses source-to-source transformation
  - Tracks error propagation through call stacks
- Used to analyze Linux 2.6.15.4 source:
  - VFS, memory management
  - All file systems (ext3, XFS, NFS, VFAT, ...)
  - SCSI, IDE, soft RAID storage subsystems

EDP: CHANNELS

- Basic abstraction: channels
  - Set of function calls
  - **Generation endpoint:** error first exposed
  - **Termination endpoint:** end of error propagation
  - **Propagating functions** in between



EDP: TOOL

```
struct file_ops {
    int (*read)  ();
    int (*write) ();
};
```

```
switch (...) {
    case ext2:  ext2_read();  break;
    case ext3:  ext3_read();  break;
    case ntfs:  ntfs_read();  break;
    ...
}
```

```
struct file_ops ext2_f_ops {
    .read  = ext2_read;
    .write = ext2_write;
};
struct file_ops ext3_f_ops {
    .read  = ext3_read;
    .write = ext3_write;
};
```

$$\exists \ if\ (expr)\ \{\ ...\ \},\ \ where \\ errorCodeVariable \subseteq expr$$

TERMINOLOGY

- Error-complete channels:

```
void goodTerminationEndpoint() {
    int err = generationEndpoint();
    if (err)
        ...
}
int generationEndpoint() {
    return -EIO;
}
```

- Error-broken channels:

```
void badTerminationEndpoint() {
    int err = generationEndpoint();
    return;
}
```

Unchecked error

```
// hfs/bfind.c
int find_init(find_data *fd) {
    fd->search_key = kmalloc(.)
    if (!fd->search_key)
        return -ENOMEM;
    ...
}
// hfs/inode.c
int file_lookup() {
    find_init(fd); /* NOT-SAVED E.C */
    fd->search_key->cat = ...; /* BAD!! */
    ...
}
```

Unsaved error

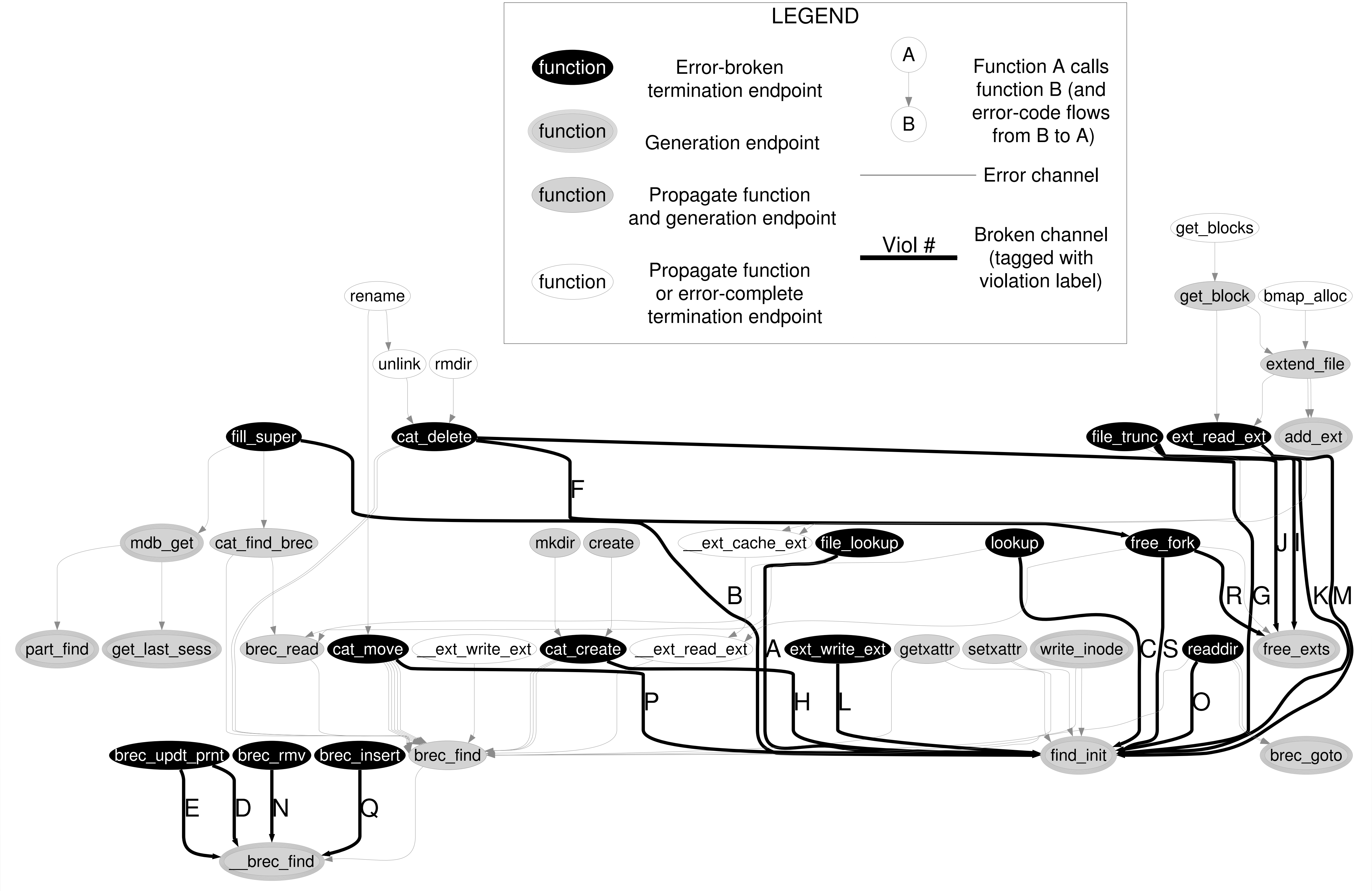
„Bad call“

FALSE POSITIVES

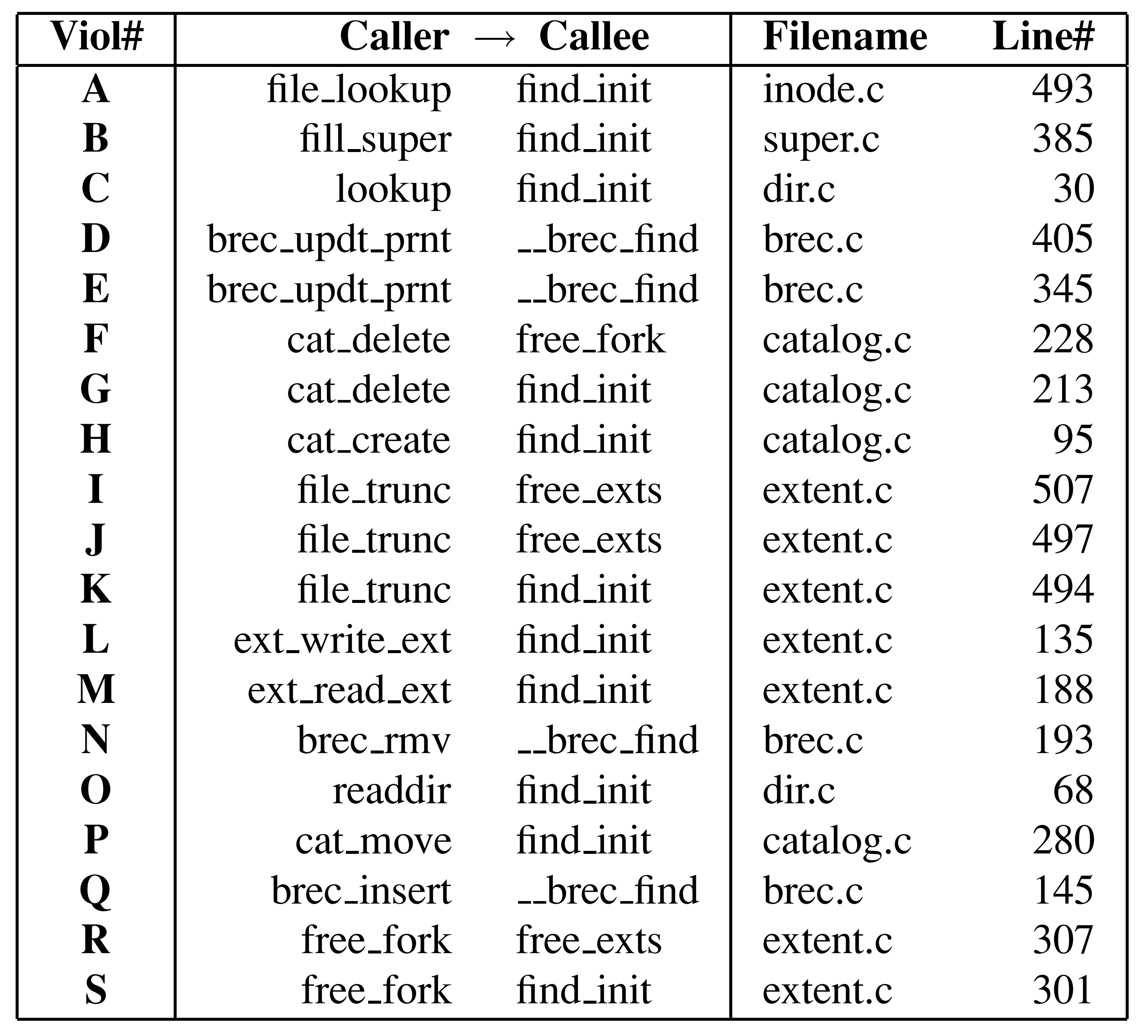
- Bad calls not always bad:
  - Multiple error returned, check only one
  - Rely on other callees to check errors

```
// fs/buffer.c
int sync_dirty_buffer (buffer_head* bh) {
    ...
    return ret; // RETURN ERROR CODE
}
// reiserfs/journal.c
int flush_commit_list() {
    sync_dirty_buffer(bh); // UNSAVED EC
    if (!buffer_uptodate(bh)) {
        return -EIO;
    }
}
```

EXAMPLE: HFS


LEGEND
function
Error-broken termination endpoint
function
Generation endpoint
function
Propagate function and generation endpoint
function
Propagate function or error-complete termination endpoint
A
B
Function A calls function B (and error-code flows from B to A)
Error channel
Viol #
Broken channel (tagged with violation label)
rename
unlink
rmdir
get_blocks
get_block
bmap_alloc
extend_file
fill_super
cat_delete
file_trunc
ext_read_ext
add_ext
F
mdb_get
cat_find_brec
mkdir
create
__ext_cache_ext
file_lookup
lookup
free_fork
J
I
B
R
G
K
M
part_find
get_last_sess
brec_read
cat_move
__ext_write_ext
cat_create
__ext_read_ext
A
ext_write_ext
getxattr
setxattr
write_inode
C
S
readdir
free_exts
P
H
L
O
brec_updt_prnt
brec_rmv
brec_insert
brec_find
find_init
brec_goto
E
D
N
Q
__brec_find

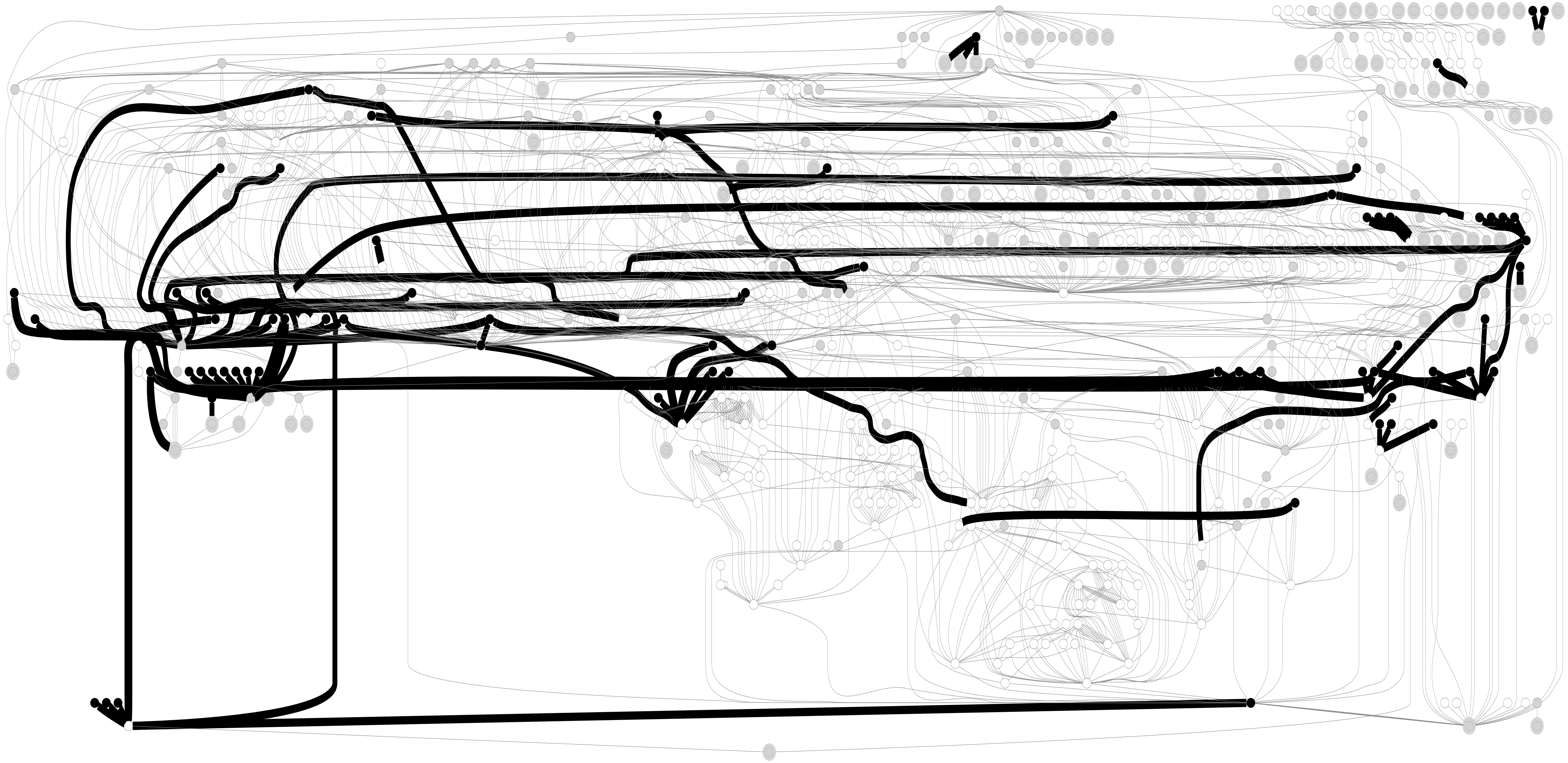
EXAMPLE: HFS



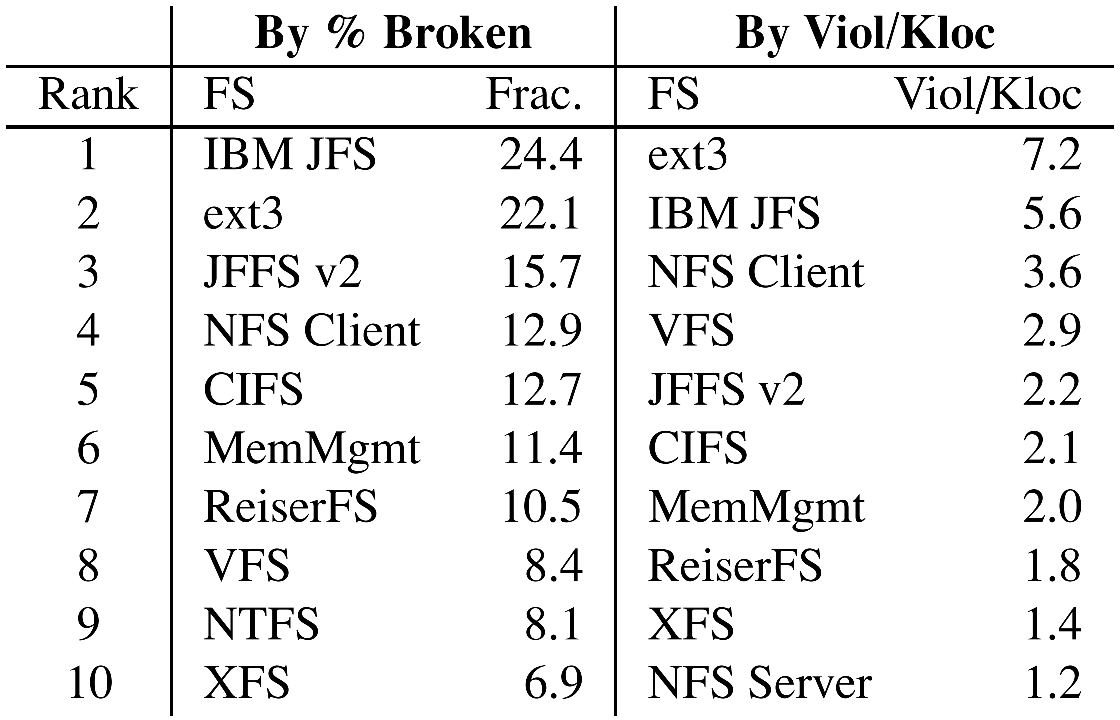
Viol#	Caller →	Callee	Filename	Line#
A	file_lookup	find_init	inode.c	493
B	fill_super	find_init	super.c	385
C	lookup	find_init	dir.c	30
D	brec_updt_prnt	__brec_find	brec.c	405
E	brec_updt_prnt	__brec_find	brec.c	345
F	cat_delete	free_fork	catalog.c	228
G	cat_delete	find_init	catalog.c	213
H	cat_create	find_init	catalog.c	95
I	file_trunc	free_exts	extent.c	507
J	file_trunc	free_exts	extent.c	497
K	file_trunc	find_init	extent.c	494
L	ext_write_ext	find_init	extent.c	135
M	ext_read_ext	find_init	extent.c	188
N	brec_rmv	__brec_find	brec.c	193
O	readdir	find_init	dir.c	68
P	cat_move	find_init	catalog.c	280
Q	brec_insert	__brec_find	brec.c	145
R	free_fork	free_exts	extent.c	307
S	free_fork	find_init	extent.c	301

COMPLEXITY

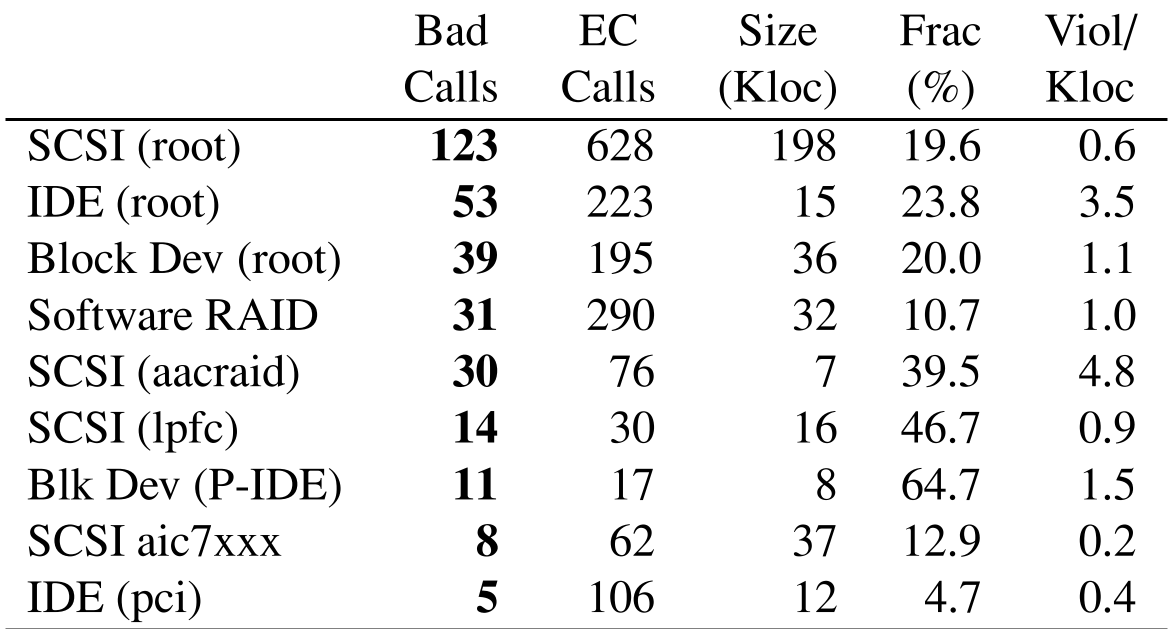
XFS [105 bad / 1453 calls, 7%]



ANALYSIS



	By % Broken		By Viol/Kloc	
Rank	FS	Frac.	FS	Viol/Kloc
1	IBM JFS	24.4	ext3	7.2
2	ext3	22.1	IBM JFS	5.6
3	JFFS v2	15.7	NFS Client	3.6
4	NFS Client	12.9	VFS	2.9
5	CIFS	12.7	JFFS v2	2.2
6	MemMgmt	11.4	CIFS	2.1
7	ReiserFS	10.5	MemMgmt	2.0
8	VFS	8.4	ReiserFS	1.8
9	NTFS	8.1	XFS	1.4
10	XFS	6.9	NFS Server	1.2



	Bad Calls	EC Calls	Size (Kloc)	Frac (%)	Viol/ Kloc
SCSI (root)	**123**	628	198	19.6	0.6
IDE (root)	**53**	223	15	23.8	3.5
Block Dev (root)	**39**	195	36	20.0	1.1
Software RAID	**31**	290	32	10.7	1.0
SCSI (aacraid)	**30**	76	7	39.5	4.8
SCSI (lpfc)	**14**	30	16	46.7	0.9
Blk Dev (P-IDE)	**11**	17	8	64.7	1.5
SCSI aic7xxx	**8**	62	37	12.9	0.2
IDE (pci)	**5**	106	12	4.7	0.4

- Only „complex“ file systems:
  10k+ SLOC, 50+ error related calls
- Ext3, JFS least robust, XFS most
- Storage: IDE has more violations than SCSI

WRITE ERRORS

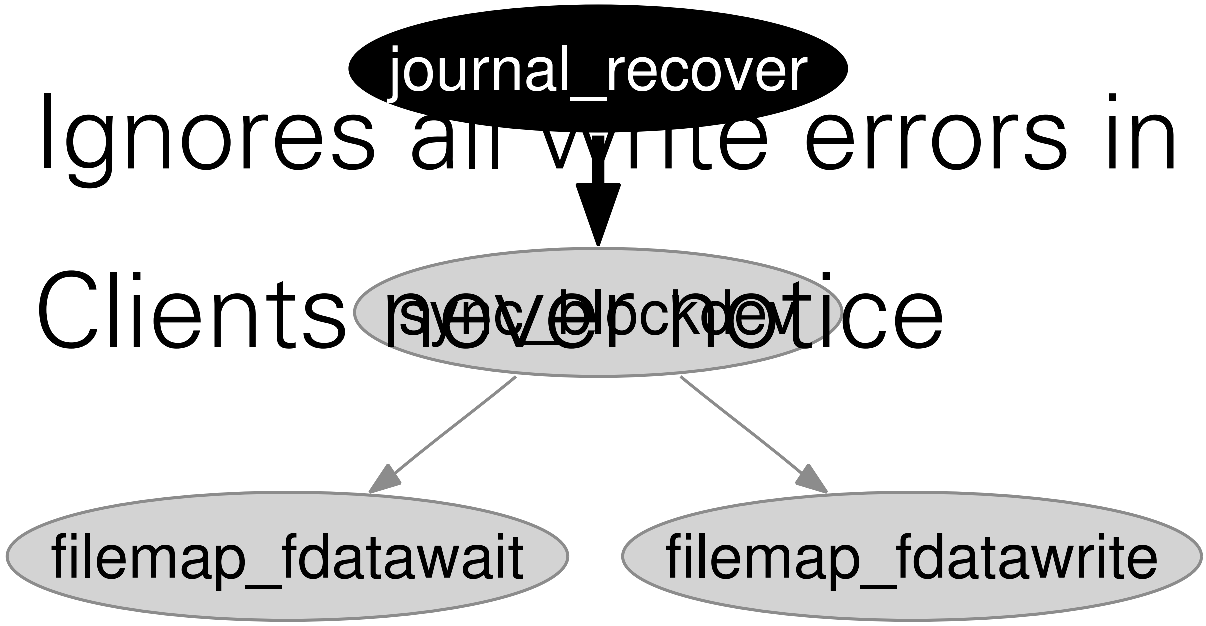
- More than 63% of write errors ignored
- Possible explanations:
  - No higher-level error handling
  - Errors neglected intentionally

Callee Type	Bad Calls	EC Calls	**Frac. (%)**
Read*	26	603	**4.3**
Sync	70	236	**29.7**
Wait	27	70	**38.6**
Write	80	598	**13.4**
Sync+Wait+Write	177	904	**19.6**
Specific Callee			
`filemap_fdatawait`	22	29	**75.9**
`filemap_fdatawrite`	30	47	**63.8**
`sync_blockdev`	15	21	**71.4**

SILENT FAILURE

- **Example 1:** Journaling Block Device (JBD)
  - JBD recovery code ignores all write errors
  - Error code dropped in middle of channel

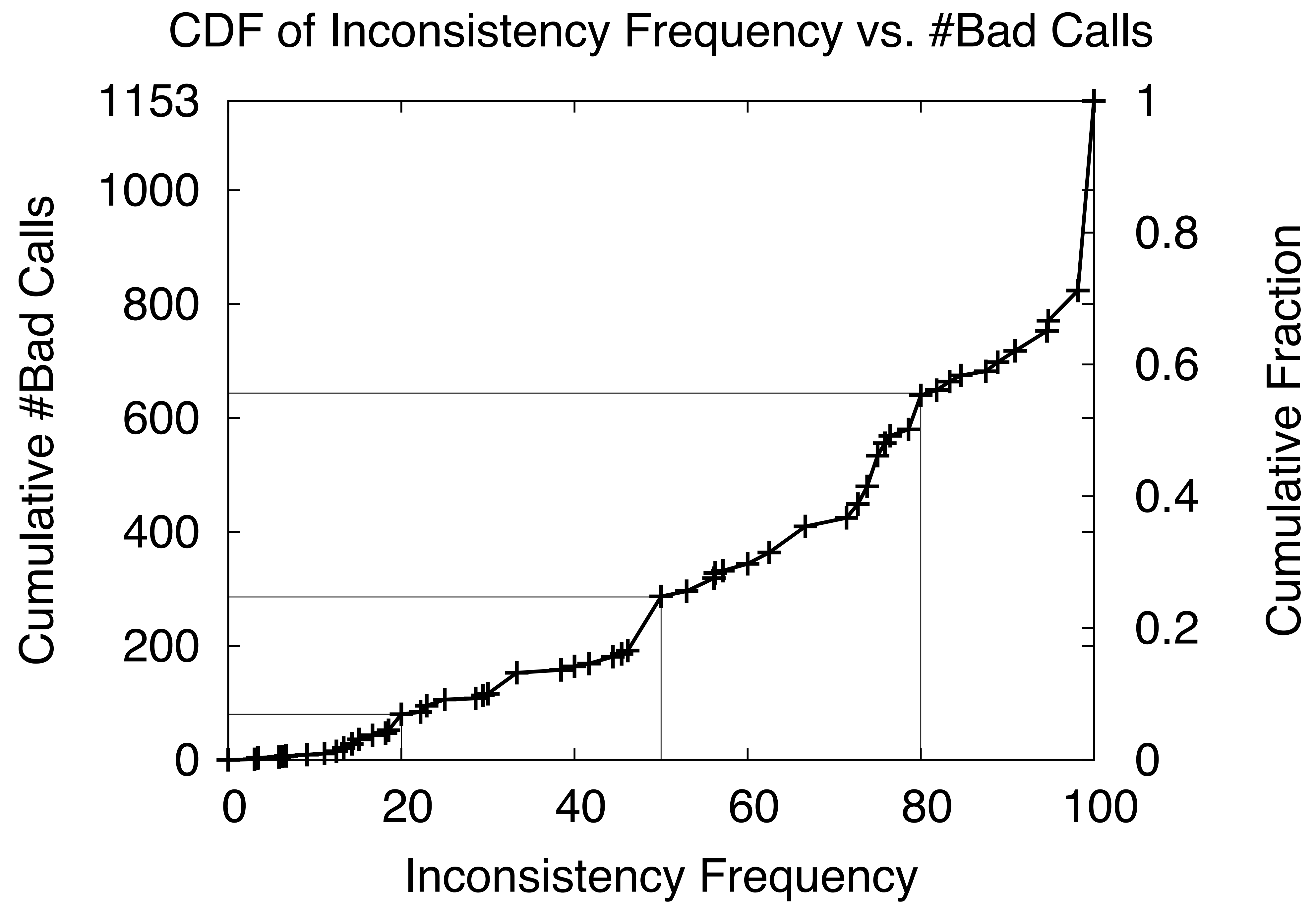
- **Example 2:** NFS server
- Ignores all write errors in sync writes
- Clients never notice



```
journal_recover()
  /* BROKEN CHANNEL */
  sync_blockdev();
```

```
sync_blockdev()
  ret = fm_fdatawrite();
  err = fm_fdatawait();
  if(!ret) ret = err;
  /* PROPAGATE EIO */
  return ret;
```

BUG FREQUENCY

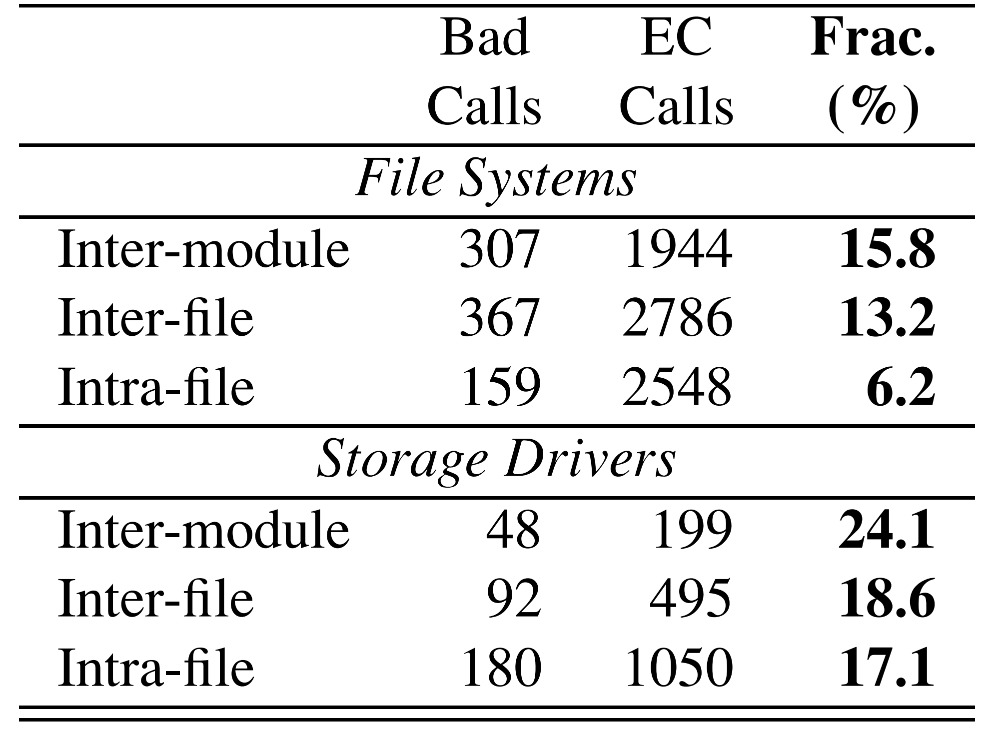


CHARACTERISTICS

- Where are error codes dropped?
- No clear pattern:
  - File systems:

    10% direct, 14% later
  - Storage drivers:

    20% direct, 15% later
- Call distance?



	Bad Calls	EC Calls	**Frac. (%)**
File Systems			
Inter-module	307	1944	**15.8**
Inter-file	367	2786	**13.2**
Intra-file	159	2548	**6.2**
Storage Drivers			
Inter-module	48	199	**24.1**
Inter-file	92	495	**18.6**
Intra-file	180	1050	**17.1**

SUMMARY

- Erros are not propagated correctly:

  Result: **1153** calls drop error (that's **13%**)
- Complex file systems are more likely to propagate errors incorrectly
- Popular file systems not the most robust
- Write errors consistently ignored:
  - May cause silent failure
  - Often no easy way to handle

DISCUSSION

- EDP catches only simple bugs, but reports many violations in all Linux file systems.
- Are the violations really that bad?
- Is OK to ignore write errors after all?
- Is ignoring write errors the *disease* or in fact a *symptom* of higher-level problems?
- Half the code is for error checking, is C the right language for that?