EIO: **E**rror-handling **i**s **O**ccasionally Correct

Haryadi S. Gunawi, Cindy Rubio-González,

Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, Ben Liblit

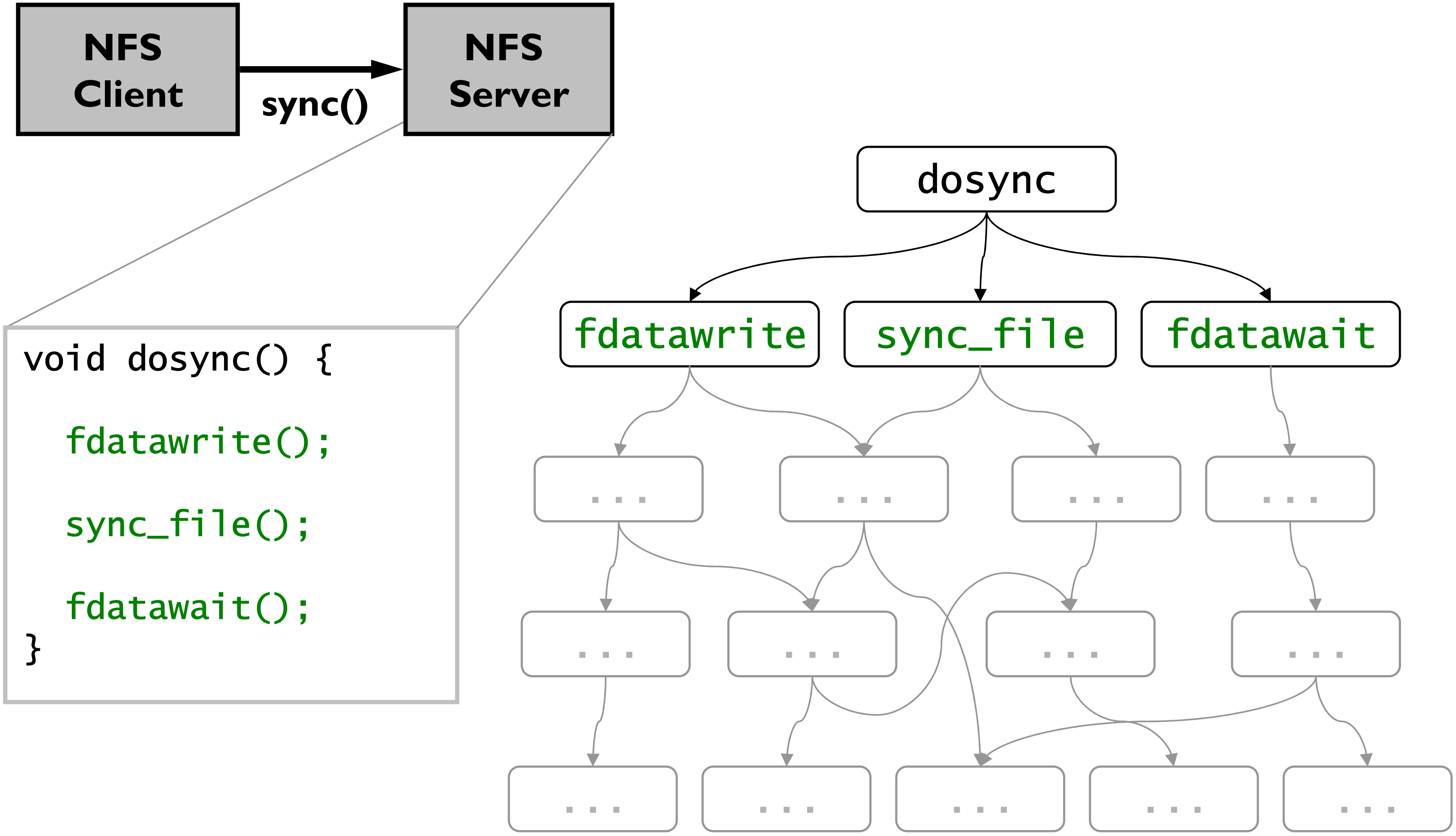
University of Wisconsin – Madison

FAST '08 – February 28, 2008

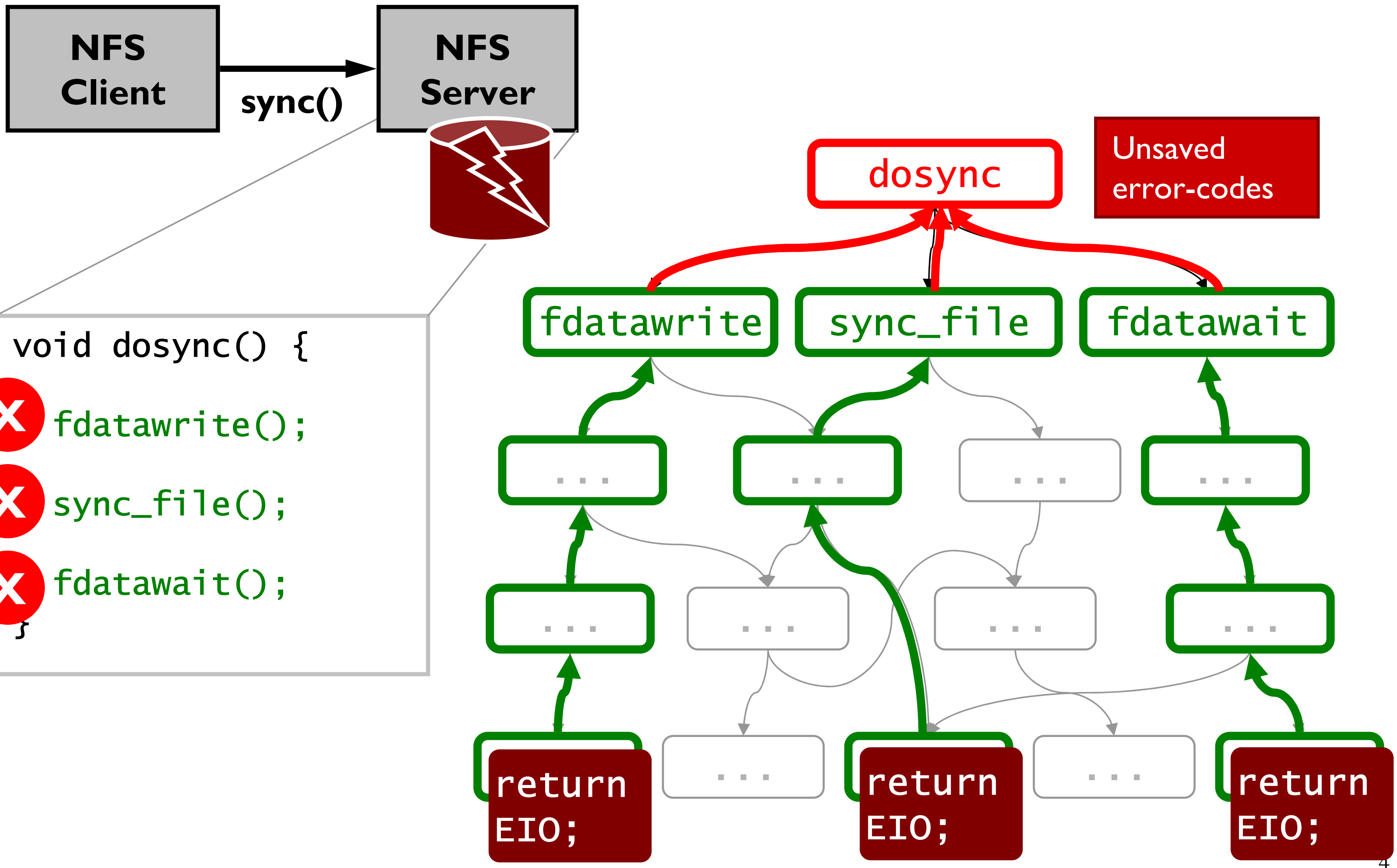
Robustness of File Systems

- Today's file systems have robustness issues
- Buggy implementation[FiSC-OSDI'04, EXPLODE-OSDI'06]
  - Unexpected behaviors in corner-case situations
- Deficient fault-handling[IRONFS-SOSP'05]
  - Inconsistent policies: propagate, retry, stop, ignore
- Prevalent ignorance
  - Ext3: Ignore write failures during checkpoint and journal replay
  - NFS: Sync-failure at the server is not propagated to client
  - What is the root cause?

Incorrect Error Code Propagation



Incorrect Error Code Propagation




```
void dosync() {
    fdatawrite();
    sync_file();
    fdatawait();
}
```

Implications

- Misleading error-codes in distributed systems
  - NFS client receives SUCCEED instead of ERROR
- Useless policies
  - Retry in NFS client is not invoked
- Silent failures
  - Much harder debugging process

EDP: Error Detection and Propagation Analysis

- Static analysis
  - Useful to show how error codes flow
  - Currently: 34 basic error codes (e.g. EIO, ENOMEM)
- Target systems
  - 51 file systems (all directories in **`linux/fs/*`**)
  - 3 storage drivers (SCSI, IDE, Software-RAID)

Results

- Number of violations
  - Error-codes flow through **9022** function calls
  - **1153** (**13%**) calls **do not save** the returned error-codes
- Analysis, a closer look
  - More complex file systems, more violations
  - Location distance affects error propagation correctness
  - Write errors are neglected more than read errors
  - Many violations are not corner-case bugs
    - Error-codes are consistently ignored

Outline

- Introduction
- ***Methodology***
  - *Challenges*
  - *EDP tool*
- Results
- Analysis
- Discussion and Conclusion

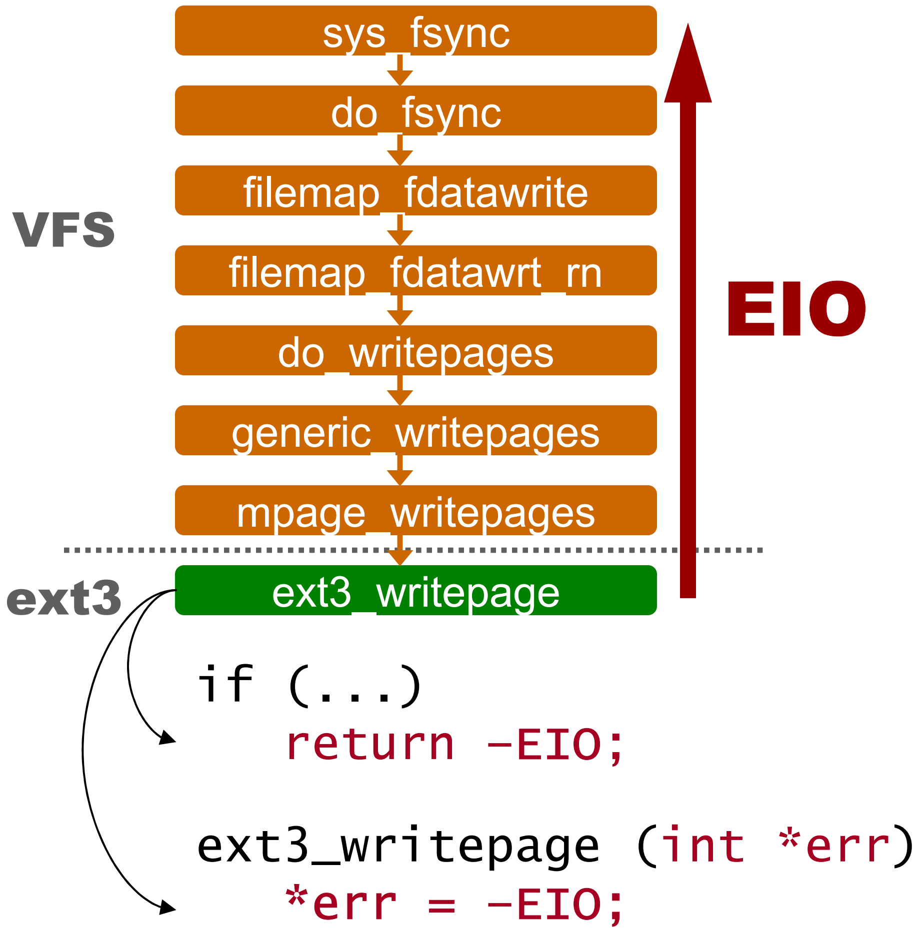
Challenges in Static Analysis

- File systems use many error codes
  - `buffer→state[Uptodate] = 0`
  - `journal→flags = ABORT`
  - `int err = -EIO; ... return err;`
- Error codes transform
  - Block I/O error becomes journal error
  - Journal error becomes generic error code
- Error codes propagate through:
  - Function call path
  - Asynchronous path (e.g. interrupt, network messages)

EDP

- State
  - Current State: Integer error-codes, function call path
  - Future: Error transformation, asynchronous path
- Implementation
  - Utilize CIL: Infrastructure for C program analysis [Necula-CC'02]
  - EDP: ~4000 LOC in Ocaml
- 3 components of EDP architecture
  - Specifying error-code information (e.g. EIO, ENOMEM)
  - Constructing error channels
  - Identifying violation points

Constructing Error Channels



- Propagate function
  - Dataflow analysis
  - Connect function pointers
- Generation endpoint
  - Generates error code
  - Example: return –EIO

Detecting Violations

Error-complete endpoint

```
func() {
    err = func_call();
    if (err)
      ...
}
```

Unchecked

```
func() {
    err = func_call();
}
```

Unsaved / Bad Call

```
func() {
    func_call();
}
```

Overwritten

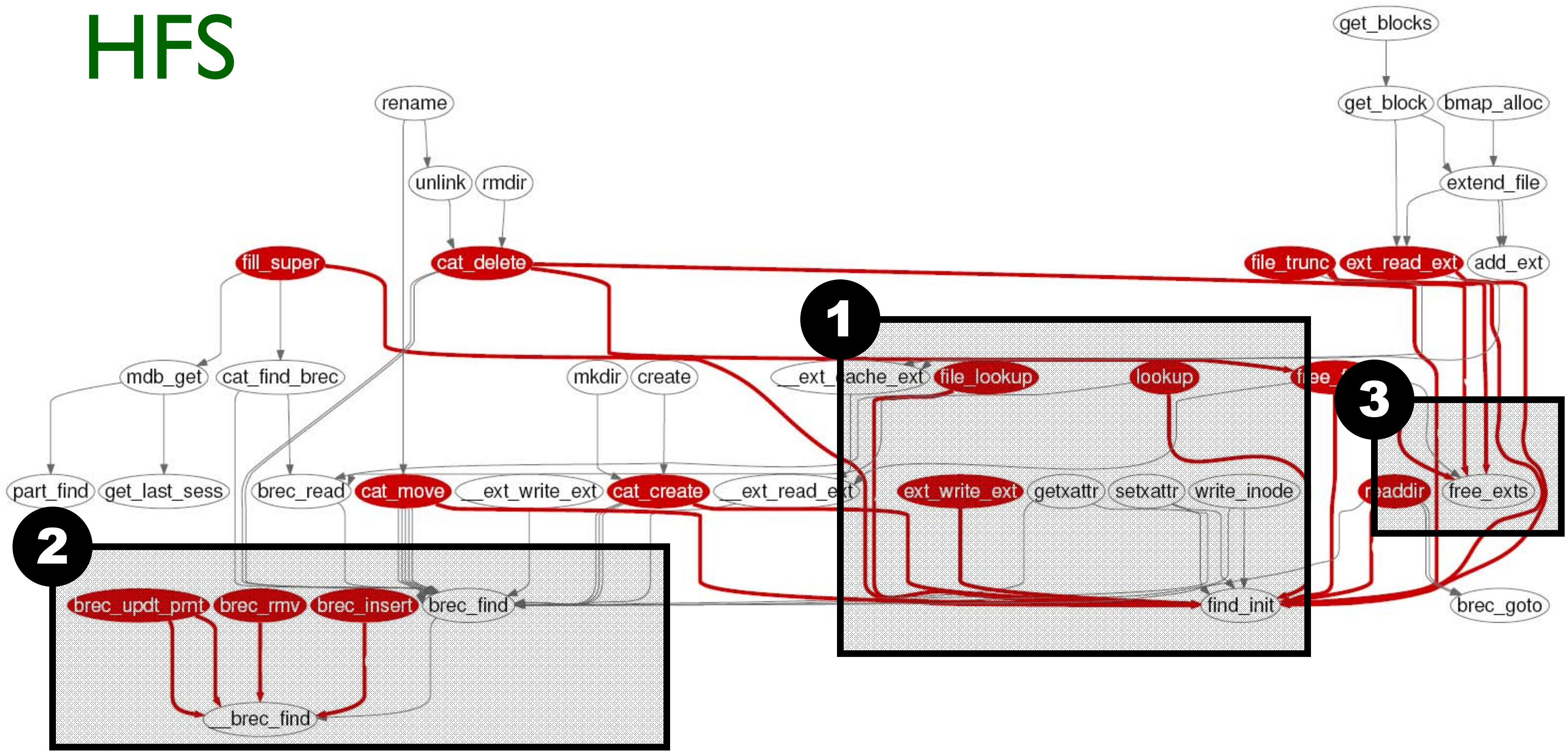
```
func() {
    err = func_call();
    err = func_call_2();
}
```

- Termination endpoint
  - Error code is no longer propagated
  - Two termination endpoints:
    - error-complete (minimally checks)
    - error-broken (unchecked, unsaved, overwritten)
- Goal:
  - Find error-broken endpoints

Outline

- Introduction
- Methodology
- ***Results*** *(unsaved error-codes / bad calls)*
  - *Graphical outputs*
  - *Complete results*
- Analysis of Results
- Discussion and Conclusion

HFS



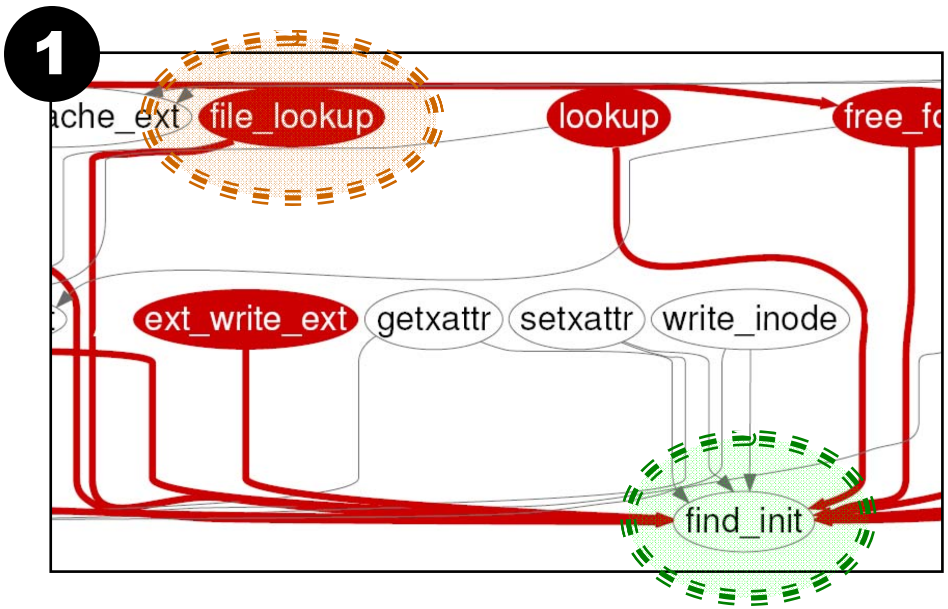
func Functions that generate/propagate error-codes

func Functions that make bad calls (do not save error-codes)

Good calls (calls that propagate error-codes)

Bad calls (calls that do not save error-codes)

HFS (Example 1)

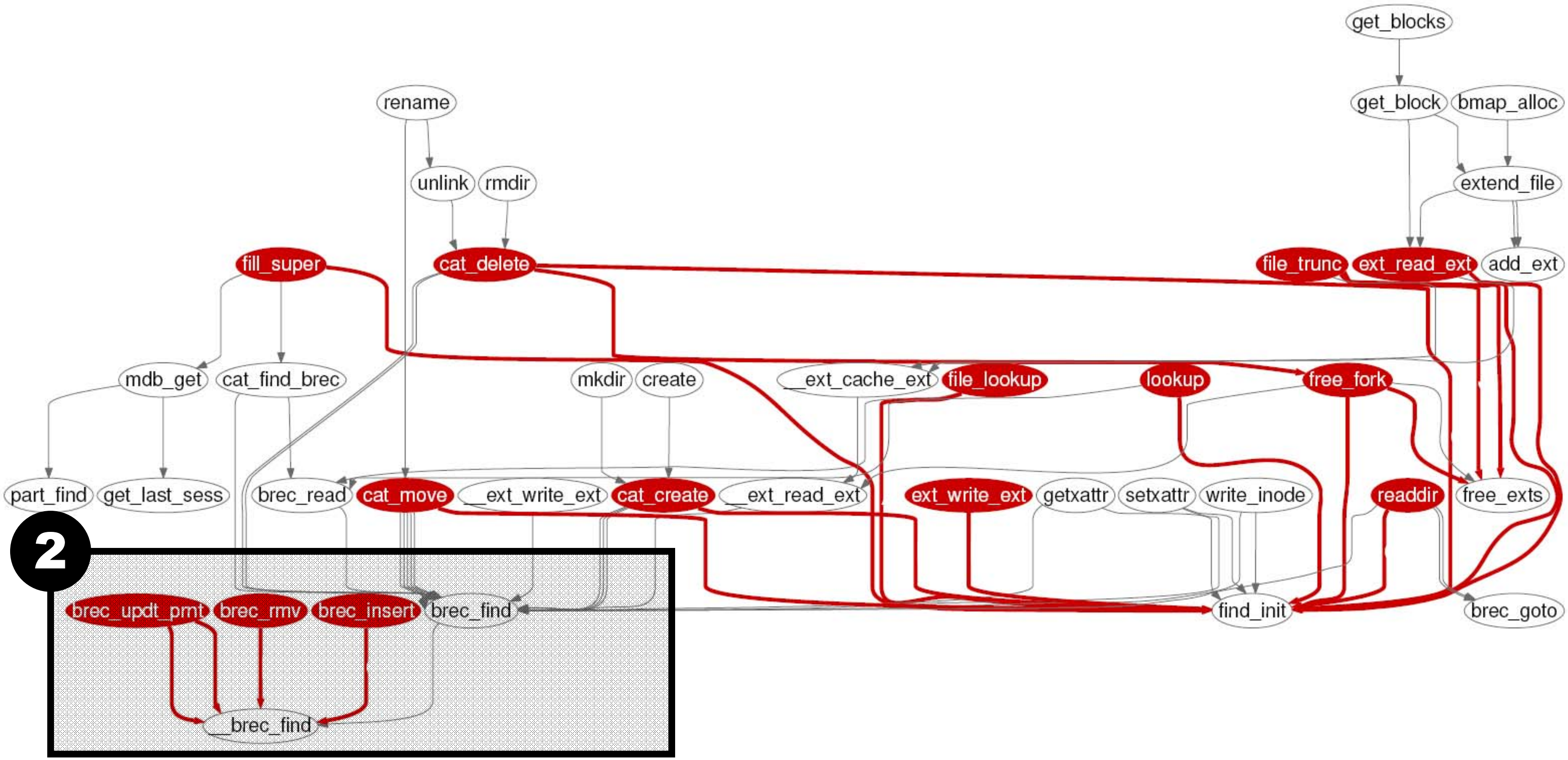



```
int find_init(find_data *fd) {
  …
  fd->search_key = kmalloc(…);
  if (!fd->search_key)
    return -ENOMEM;
  …
}
```

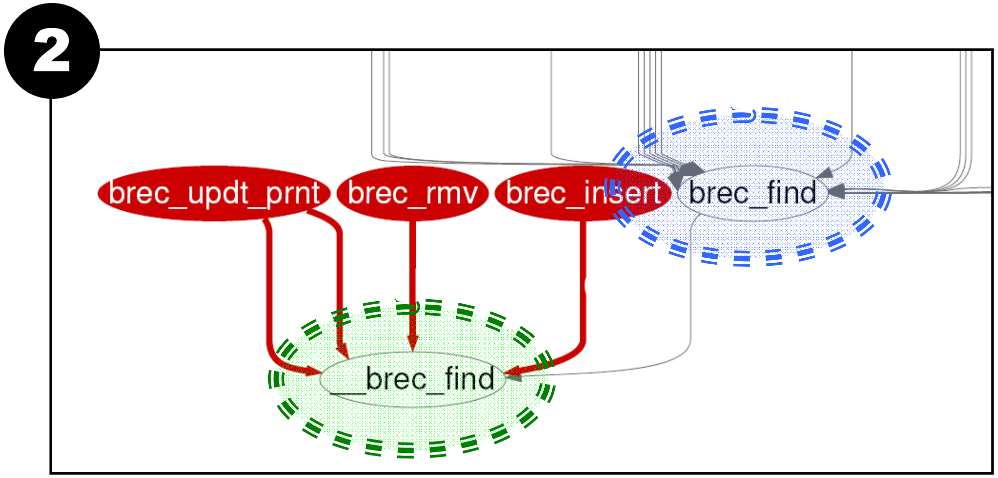
```
int file_lookup() {
  …
  find_init(fd);                 Bad call!
  fd->search_key->cat = …;
  …                              Null pointer dereference
}
```

Inconsistencies		
Callee	Good Calls	Bad Calls
find_init	3	11

HFS (Example 2)



HFS (Example 2)

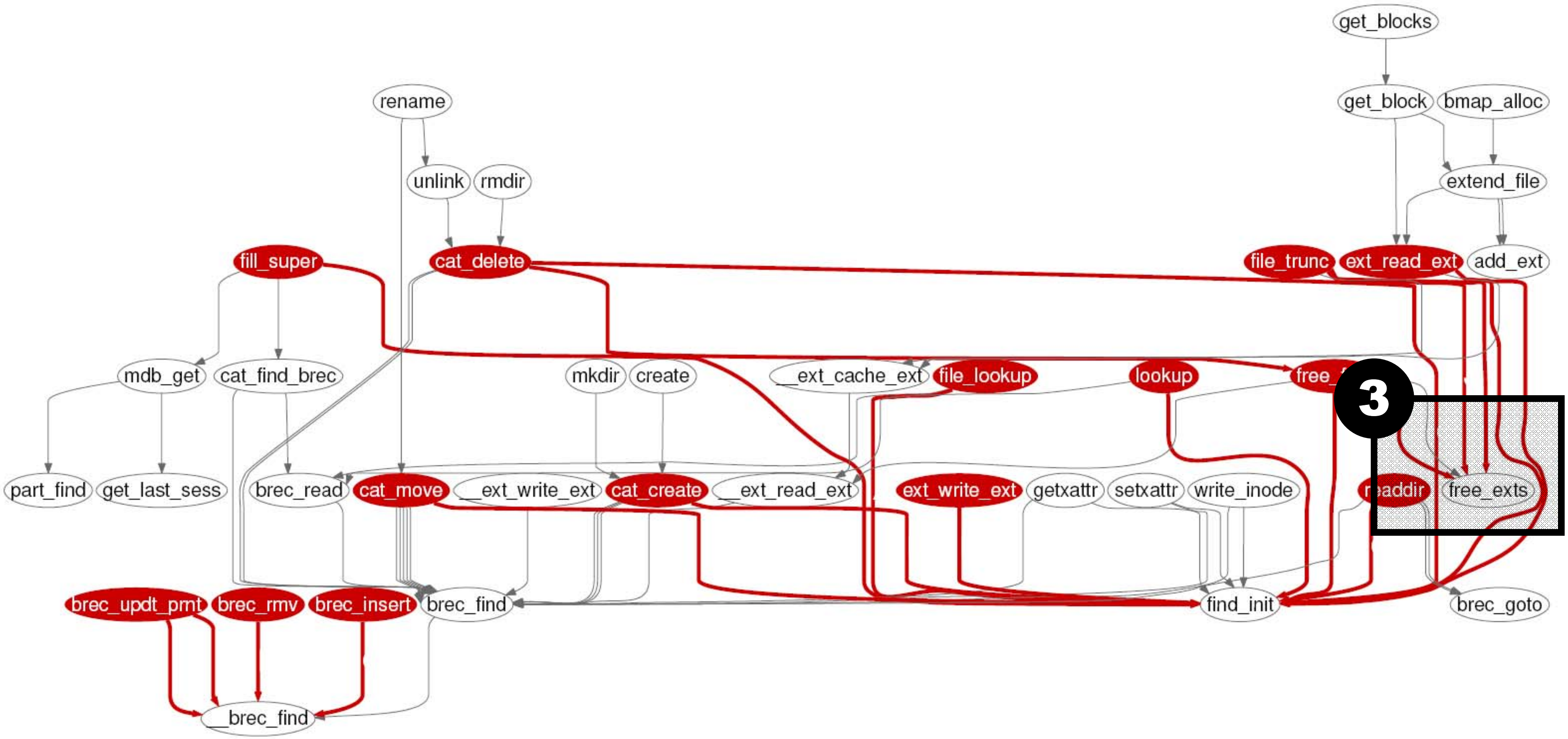



```
int __brec_find(key) {
    Finds a record in an HFS node
    that best matches the given key.
    Returns ENOENT if it fails.
}
```

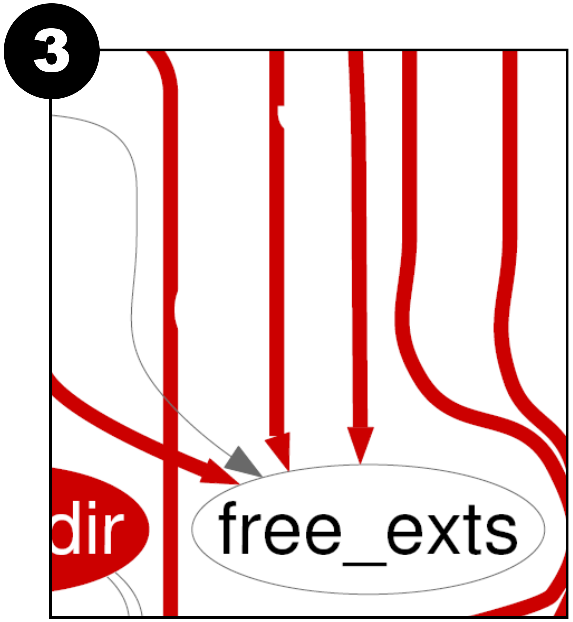
```
int brec_find(key) {
   …
   result = __brec_find(key);
   …
   return result;
}
```

Inconsistencies		
Callee	Good Calls	Bad Calls
find_init	3	11
__brec_find	1	4

HFS (Example 3)



HFS (Example 3)




```
int free_exts(…) {
    Traverses a list of extents and
    locate the extents to be freed.
    If not found, returns EIO.
    "panic?" is written before
     the return EIO statement.
}
```

Inconsistencies		
Callee	Good Calls	Bad Calls
find_init	3	11
__brec_find	1	4
brec_find	18	0
free_exts	1	3

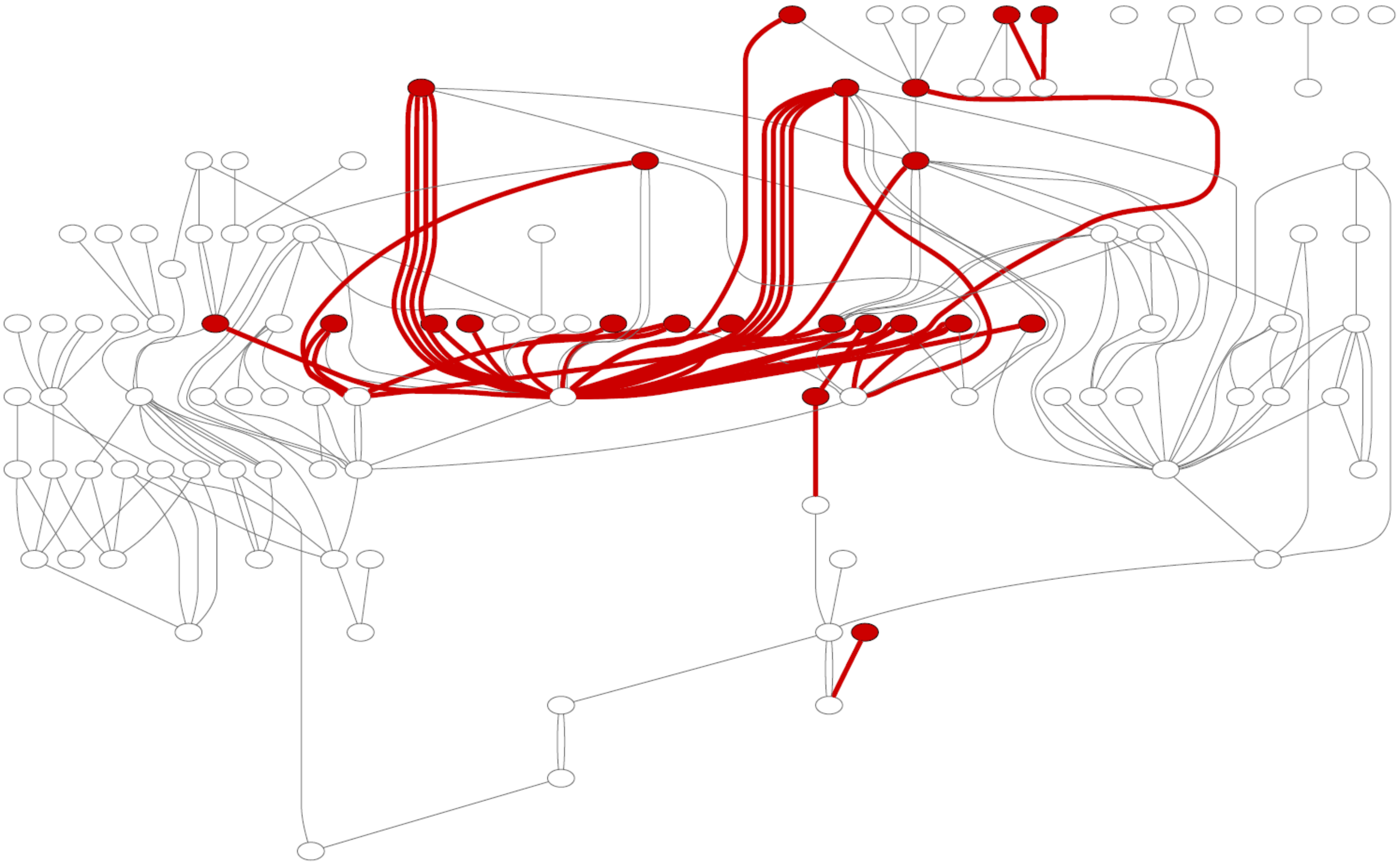
HFS (Summary)

Inconsistencies		
Callee	Good Calls	Bad Calls
find_init	3	11
__brec_find	1	4
brec_find	18	0
free_exts	1	3

- Not only in HFS
- Almost all file systems and storage systems have major inconsistencies

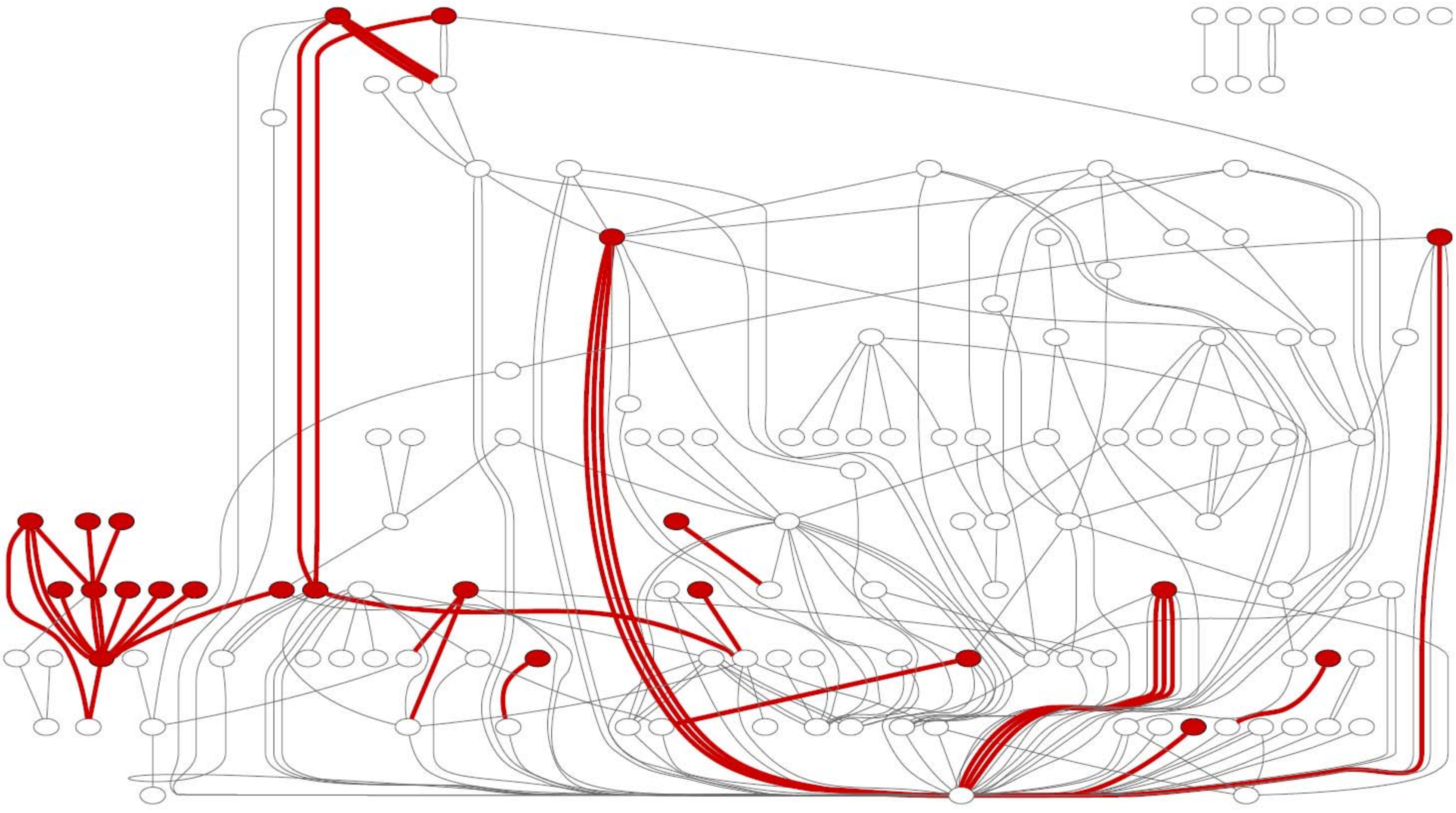
ext3

37 bad / 188 calls = 20%



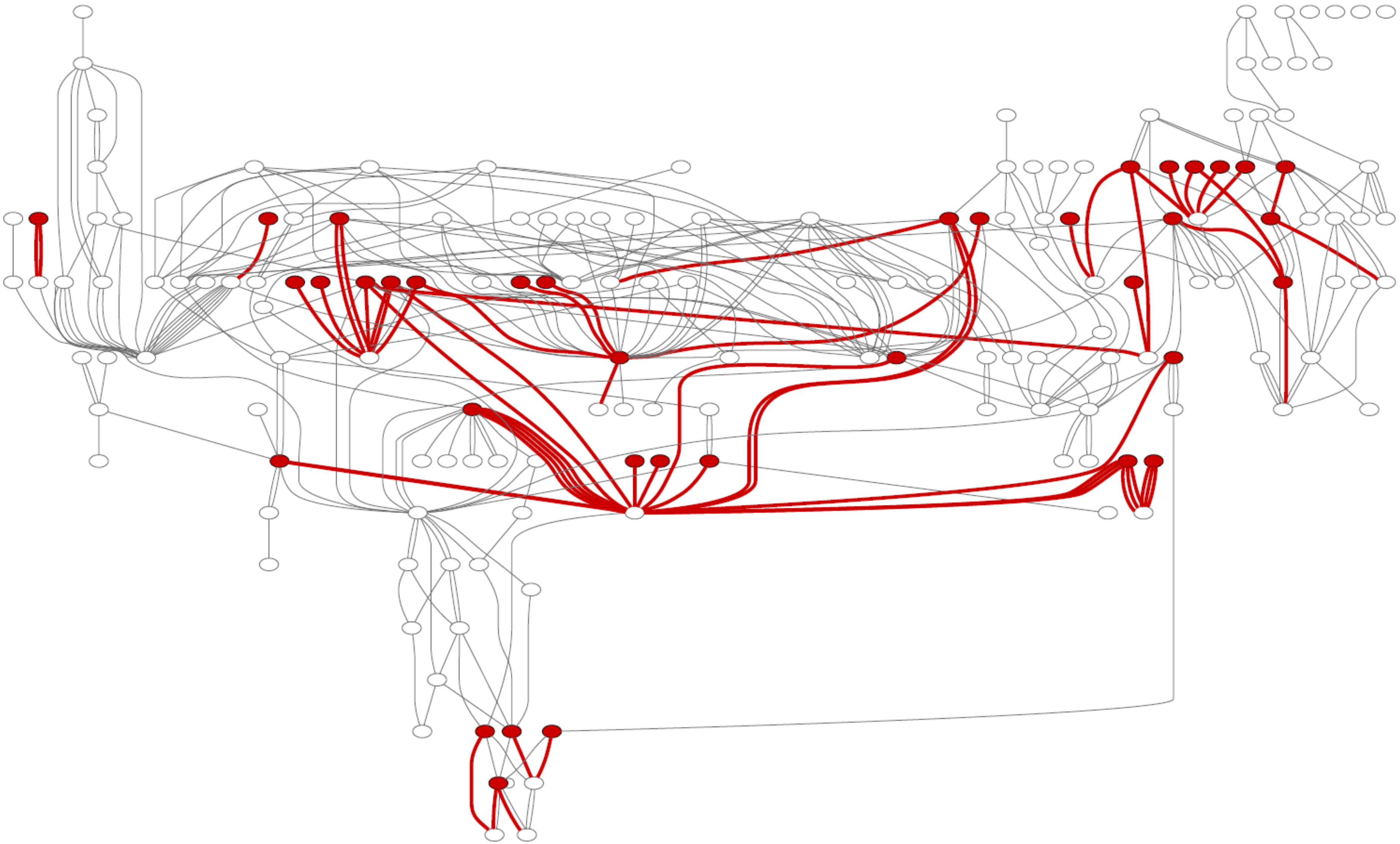
ReiserFS

35 bad / 218 calls = 16%



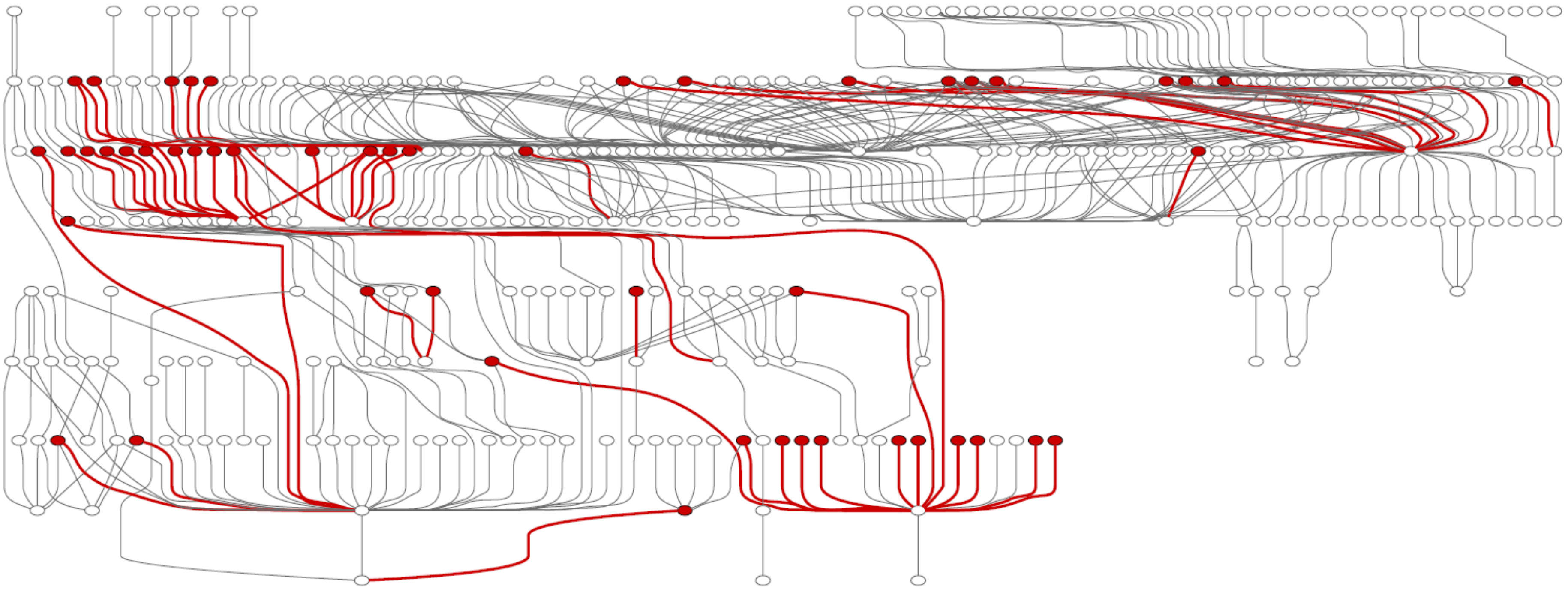
IBM JFS

61 bad / 340 calls = 18%



NFS Client

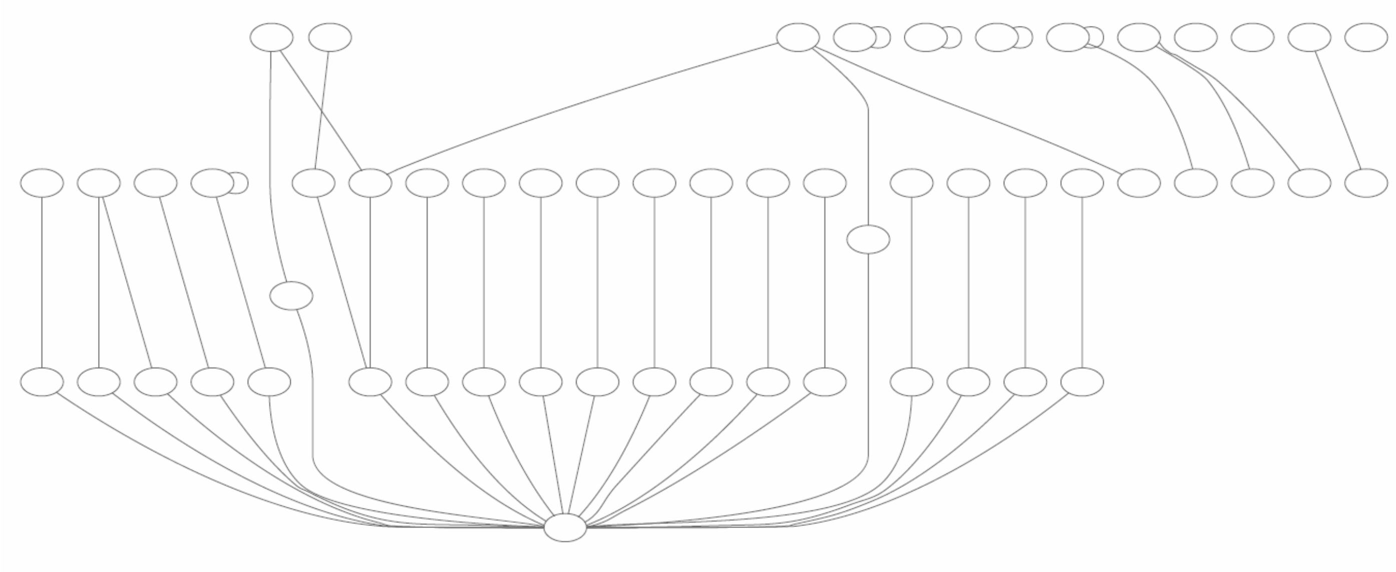
54 bad / 446 calls = 12%



Coda

0 bad / 54 calls = 0% (internal)

0 bad / 95 calls = 0% (external)



Summary

- Incorrect error propagation plagues almost all file systems and storage systems

	Bad Calls	EC Calls	Fraction
File systems	**914**	7400	**12%**
Storage drivers	**177**	904	**20%**

Outline

- Introduction
- Methodology
- Results
- ***Analysis of Results***
- ***Discussion and Conclude***

Analysis of Results

- Correlate robustness and complexity
  - Correlate file system size with **number** of violations
    - More complex file systems, more violations (Corr = 0.82)
  - Correlate file system size with **frequency** of violations
    - Small file systems make frequent violations (Corr = -0.20)
- Location distance of calls affects correct error propagation
  - Inter-module > inter-file > intra-file bad calls
- Read vs. Write failure-handling
- Corner-case or consistent mistakes

Read vs. Write Failure-Handling

- Filter read/write operations (string comparison)
  - Callee contains "**write**", or "**sync**", or "**wait**" → **Write ops**
  - Callee contains "**read**" → **Read ops**

Callee Type	Bad Calls	EC Calls	Fraction
Read	**26***	603	**4%**
Sync+Wait+Write	**177**	904	**20%**

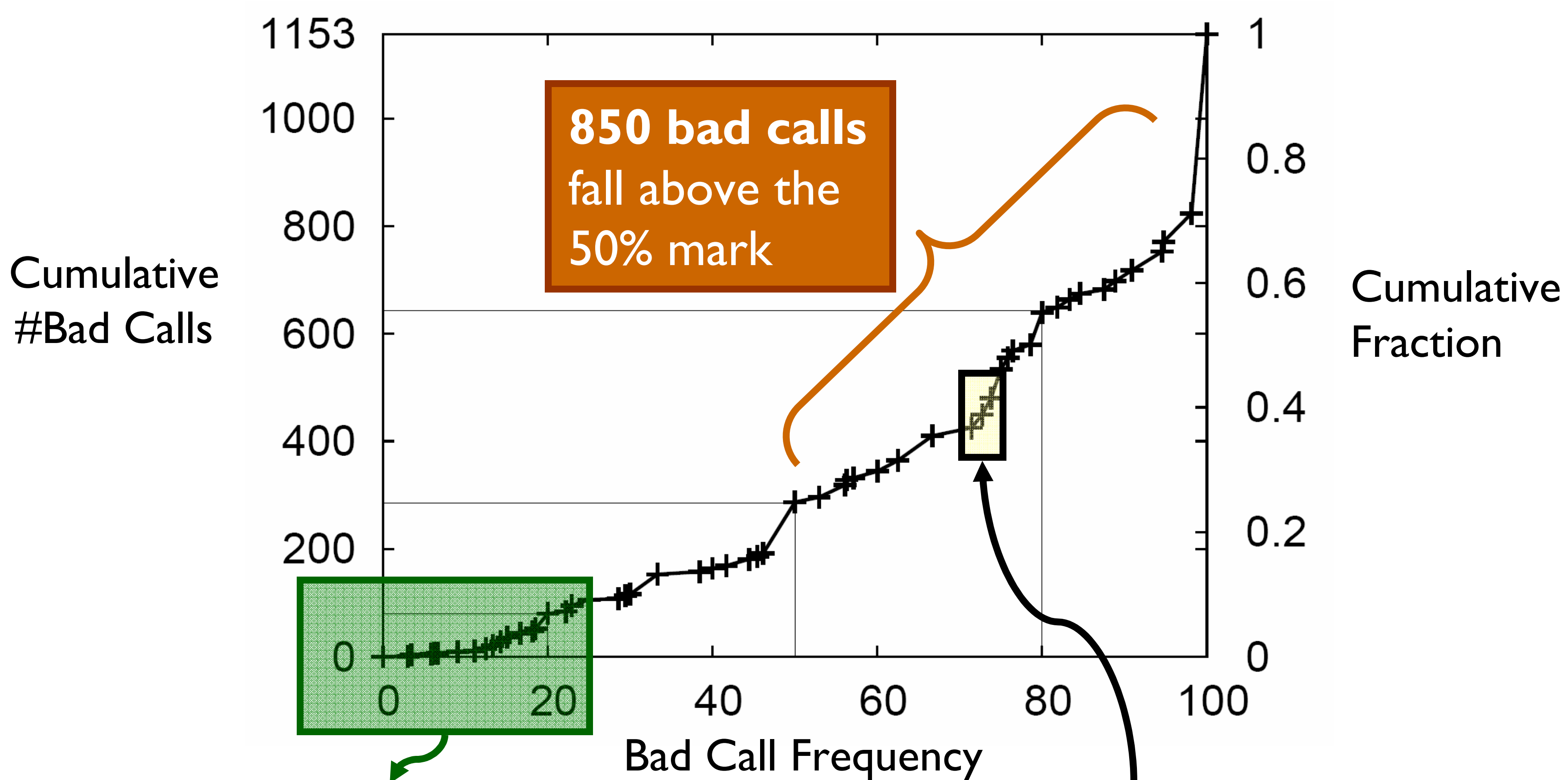
mm/readahead.c
Read prefetching in Memory Management

Lots of write failures are ignored!

Corner-Case or Consistent Mistakes?

- Define bad call frequency = $\frac{\text{\# Bad calls to f()}}{\text{\# All calls to f()}}$
  - Example: sync_blockdev, 15/21
  - Bad call frequency: **71%**
- Corner-case bugs
  - Bad call frequency < 20%
- Consistent bugs
  - Bad call frequency > 50%

CDF of Bad Call Frequency



Less than 100 violations are corner-case bugs

sync_blockdev
15 bad calls / 21 EC calls
Bad Call Freq: **71 %**
At **x = 71**, **y += 15**

What's going on?

- Not just bugs
- But more fundamental design issues
  - Checkpoint failures are ignored
    - Why? Maybe because of journaling flaw [IOShepherd-SOSP'07]
    - Cannot recover from checkpoint failures
    - Ex: A simple block remap could not result in a consistent state
  - Many write failures are ignored
    - Lack of recovery policies? Hard to recover?
  - Many failures are ignored in the middle of operations
    - Hard to rollback?

Conclusion (developer comments)

- **ext3** “there's no way of reporting error to userspace. So ignore it”
- **XFS** “Just ignore errors at this point. There is nothing we can do except to try to keep going”
- **ReiserFS** “we can't do anything about an error here”
- **IBM JFS** “note: todo: log error handler”
- **CIFS** “should we pass any errors back?”
- **SCSI** “Todo: handle failure”

Thank you!
Questions?

ADvanced **S**ystems **L**aboratory

www.cs.wisc.edu/adsl

Extra Slides