A Study of
Linux File System Evolution

Lanyue Lu
Andrea C. Arpaci-Dusseau
Remzi H. Arpaci-Dusseau
Shan Lu

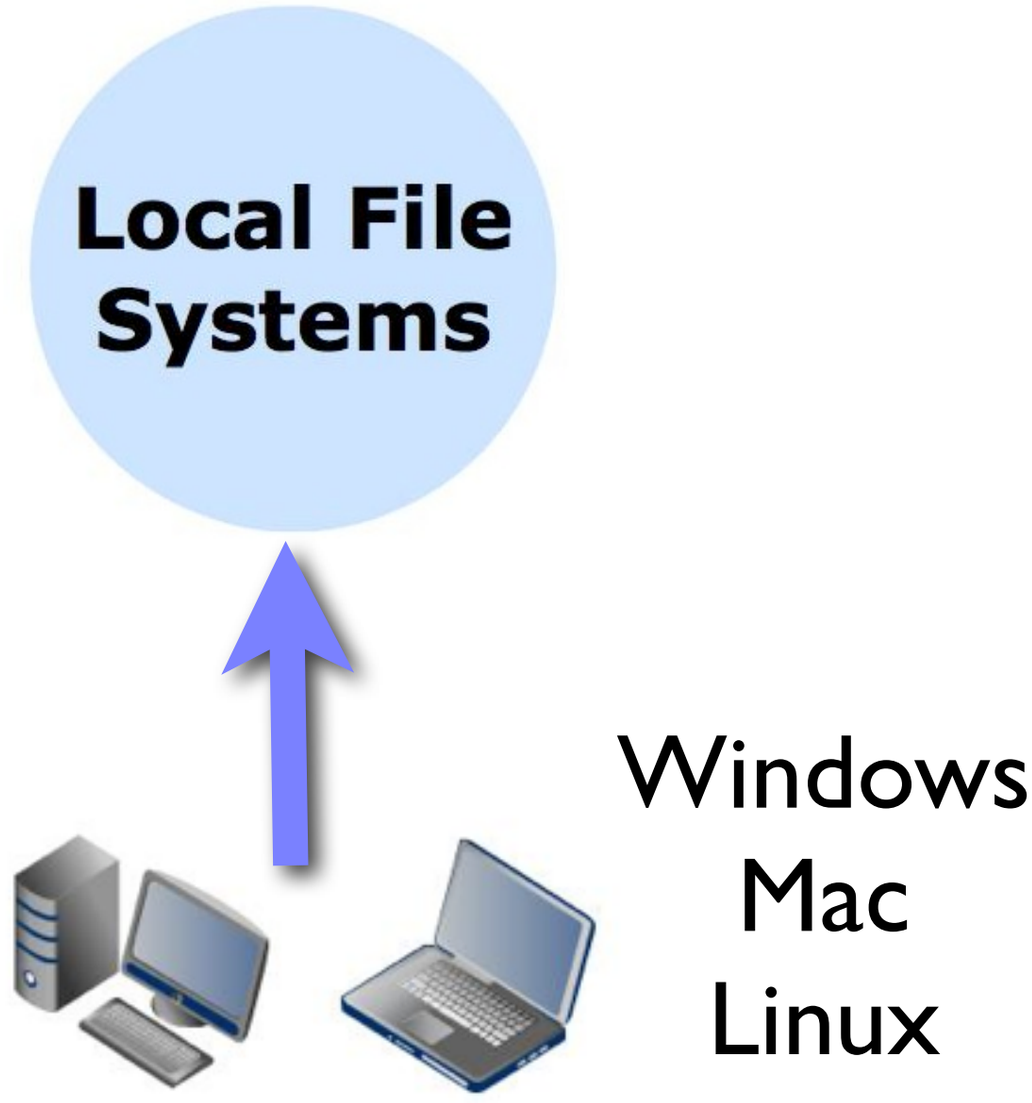
University of Wisconsin - Madison

Local File Systems Are Important

Local File Systems Are Important

Windows
Mac
Linux

Local File Systems Are Important



Local File Systems Are Important

Hadoop
DFS

Google
GFS

Local File Systems

Windows
Mac
Linux

Local File Systems Are Important

Hadoop
DFS

Google
GFS

Local File Systems

Windows
Mac
Linux

Local File Systems Are Important

Hadoop
DFS

Google
GFS

Android

iPhone

Local File Systems

Windows
Mac
Linux

Local File Systems Are Important

Hadoop
DFS

Google
GFS

Android

iPhone

Local File Systems

Windows
Mac
Linux

Why Study Is Useful ?

Why Study Is Useful ?

Study drives system designs

- previous work focuses on measurements
- little emphasis on system evolution

Why Study Is Useful ?

Study drives system designs

- previous work focuses on measurements
- little emphasis on system evolution

Answer important questions

- complexity of file systems
- dominant bug types
- performance optimizations
- reliability enhancements
- similarities across file systems

Who Is This Study Useful To ?

Who Is This Study Useful To ?

File system developers

- avoid same mistakes
- improve existing design and implementation

Who Is This Study Useful To ?

File system developers

- ➡ avoid same mistakes
- ➡ improve existing design and implementation

System researchers

- ➡ identify problems that plague existing systems
- ➡ match research to reality

Who Is This Study Useful To ?

File system developers

- avoid same mistakes
- improve existing design and implementation

System researchers

- identify problems that plague existing systems
- match research to reality

Tool builders

- large-scale statistical bug patterns
- effective bug-finding tools
- realistic fault injection

How We Studied ?

How We Studied ?

File systems are evolving

- code base is not static
- new features, bug-fixings
- performance and reliability improvement

How We Studied ?

File systems are evolving

- code base is not static
- new features, bug-fixings
- performance and reliability improvement

Patches describe evolution

- how one version transforms to the next
- every patch is available
- “system archeology”

How We Studied ?

File systems are evolving

- code base is not static
- new features, bug-fixings
- performance and reliability improvement

Patches describe evolution

- how one version transforms to the next
- every patch is available
- “system archeology”

Study with other rich information

- source code, design documents
- forum, mailing lists

What We Did ?

What We Did ?

Manual patch inspection

- XFS, Ext4, Btrfs, Ext3, Reiserfs, JFS
- Linux 2.6 series
- 5079 patches, multiple passes

What We Did ?

Manual patch inspection

- XFS, Ext4, Btrfs, Ext3, Reiserfs, JFS
- Linux 2.6 series
- 5079 patches, multiple passes

Quantitatively analyze in various aspects

- patch types, bug patterns and consequence
- performance and reliability techniques

What We Did ?

Manual patch inspection

- XFS, Ext4, Btrfs, Ext3, Reiserfs, JFS
- Linux 2.6 series
- 5079 patches, multiple passes

Quantitatively analyze in various aspects

- patch types, bug patterns and consequence
- performance and reliability techniques

Provide an annotated dataset

- rich data for further analysis

Major Results Preview

Major Results Preview

Bugs are prevalent

Major Results Preview

Bugs are prevalent

Semantic bugs dominate

Major Results Preview

Bugs are prevalent

Semantic bugs dominate

Bugs are constant

Major Results Preview

Bugs are prevalent

Semantic bugs dominate

Bugs are constant

Corruption and crash are most common

Major Results Preview

Bugs are prevalent

Semantic bugs dominate

Bugs are constant

Corruption and crash are most common

Metadata management has high bug density

Major Results Preview

Bugs are prevalent

Semantic bugs dominate

Bugs are constant

Corruption and crash are most common

Metadata management has high bug density

Failure paths are error-prone

Major Results Preview

Bugs are prevalent

Semantic bugs dominate

Bugs are constant

Corruption and crash are most common

Metadata management has high bug density

Failure paths are error-prone

Various performance techniques are used

Outline

Introduction

Methodology

Study Results

Methodology

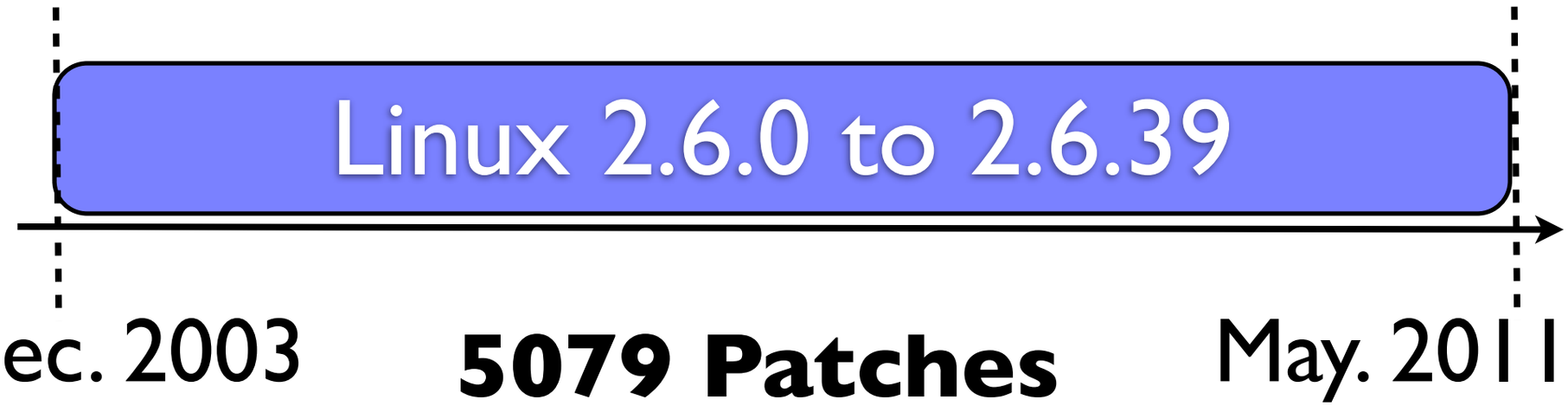
Methodology

Diverse: XFS | Ext4 | Btrfs | Ext3 | Reiser | JFS

Methodology

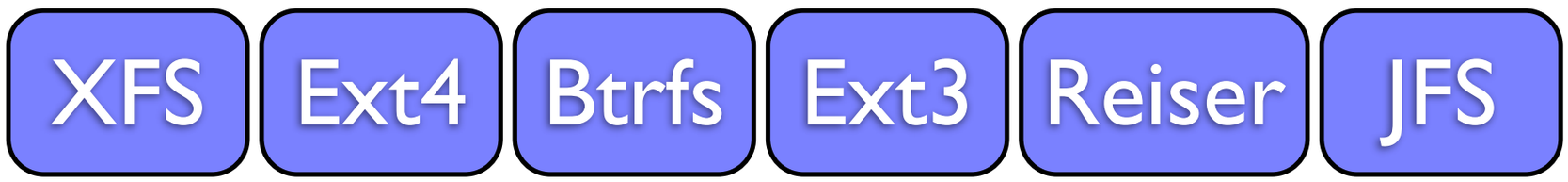
Diverse: XFS | Ext4 | Btrfs | Ext3 | Reiser | JFS

Complete:



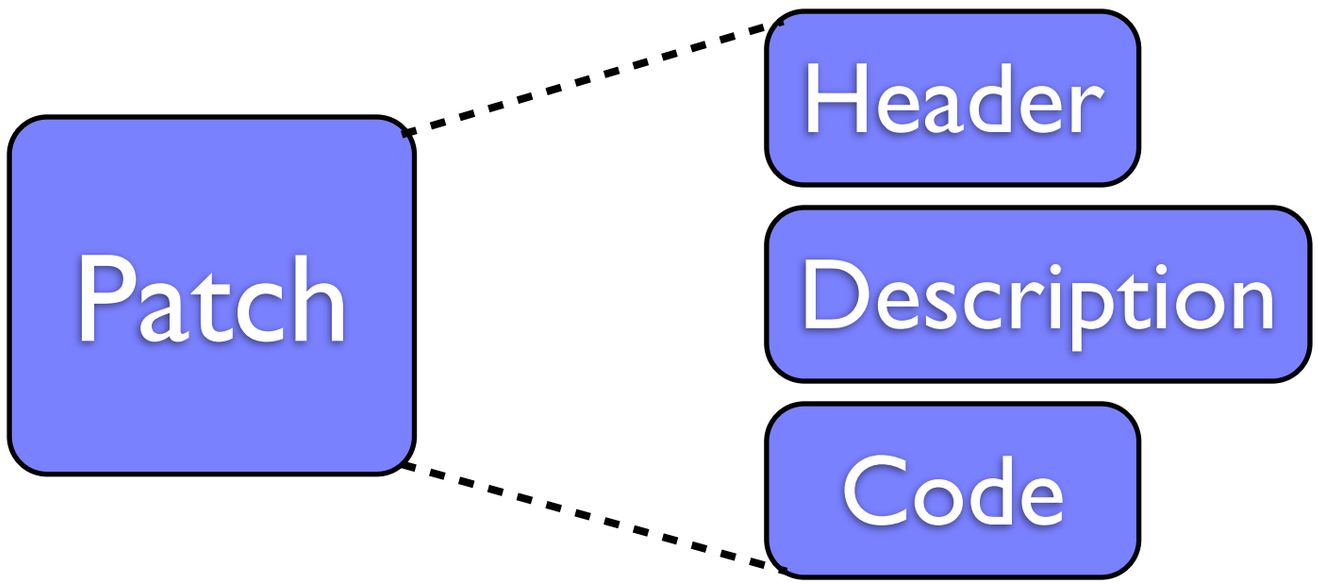
Methodology

Diverse:



Complete:

Comprehensive:



1

Patch Header

[PATCH] fix possible NULL pointer in ext3/super.c.

2 Patch Description

In fs/ext3/super.c::ext3_get_journal() at line 1675,
`journal' can be NULL, but it is not handled right
(detect by Coverity's checker).

3

Related Code

```
---    /fs/ext3/super.c
+++    /fs/ext3/super.c
@@  -1675,6 +1675,7
@@   journal_t *ext3_get_journal()

1    if (!journal) {
2       printk(KERN_ERR "EXT3: Could not load");
3       iput(journal_inode);

4    }
5    journal->j_private = sb;
```

3

Related Code

```
---    /fs/ext3/super.c
+++    /fs/ext3/super.c
@@  -1675,6 +1675,7
@@   journal_t *ext3_get_journal()

1     if (!journal) {
2         printk(KERN_ERR "EXT3: Could not load");
3         iput(journal_inode);

4     }
5     journal->j_private = sb;
```

3

Related Code

```
---    /fs/ext3/super.c
+++    /fs/ext3/super.c
@@  -1675,6 +1675,7
@@   journal_t *ext3_get_journal()

1    if (!journal) {
2       printk(KERN_ERR "EXT3: Could not load");
3       iput(journal_inode);
        return NULL;
4    }
5    journal->j_private = sb;
```

Classifications

Classifications

Patch overview

- type: bug
- size: l

Classifications

Patch overview

- type: bug
- size: l

Bug analysis

- pattern: memory (nullptr)
- consequence: crash
- data structure: super
- tool: Coverity

Classifications

Patch overview

- type: bug
- size: l

Bug analysis

- pattern: memory (nullptr)
- consequence: crash
- data structure: super
- tool: Coverity

Performance and reliability

- pattern
- location

Limitations

Limitations

Only six popular file systems

- ➡ many other file systems

Limitations

Only six popular file systems

- many other file systems

Only Linux 2.6 major versions

- omit earlier versions

Limitations

Only six popular file systems

- many other file systems

Only Linux 2.6 major versions

- omit earlier versions

Only reported bugs

- existing, but unknown bugs

Outline

Introduction

Methodology

Study Results

Questions to Answer

What do patches do ?

What do bugs look like ?

Do bugs diminish over time ?

What consequences do bugs have ?

Where does complexity of file systems lie ?

Do bugs occur on normal paths ?

What performance techniques are used ?

Q1:

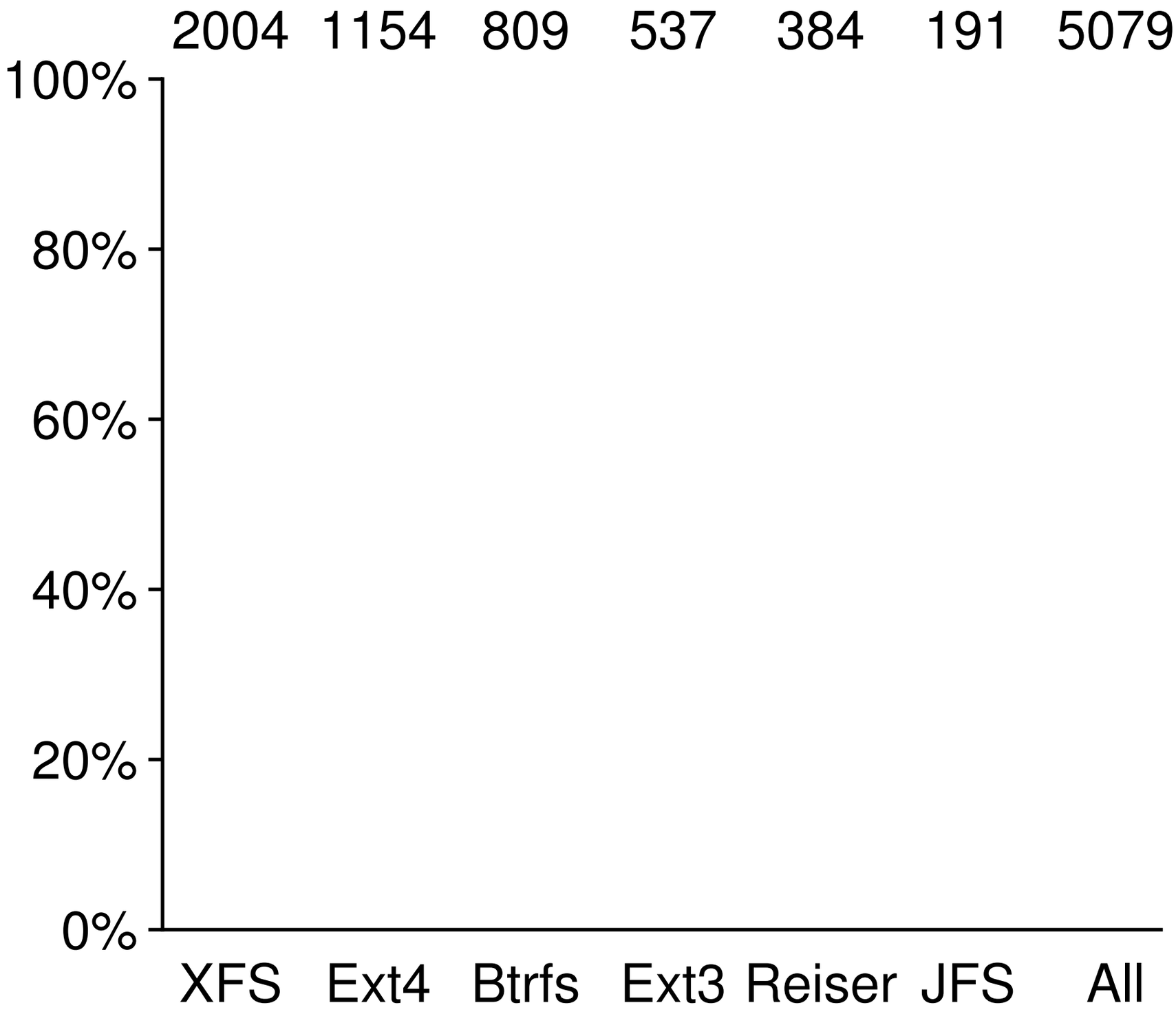
What do **patches** do ?

Patch Overview

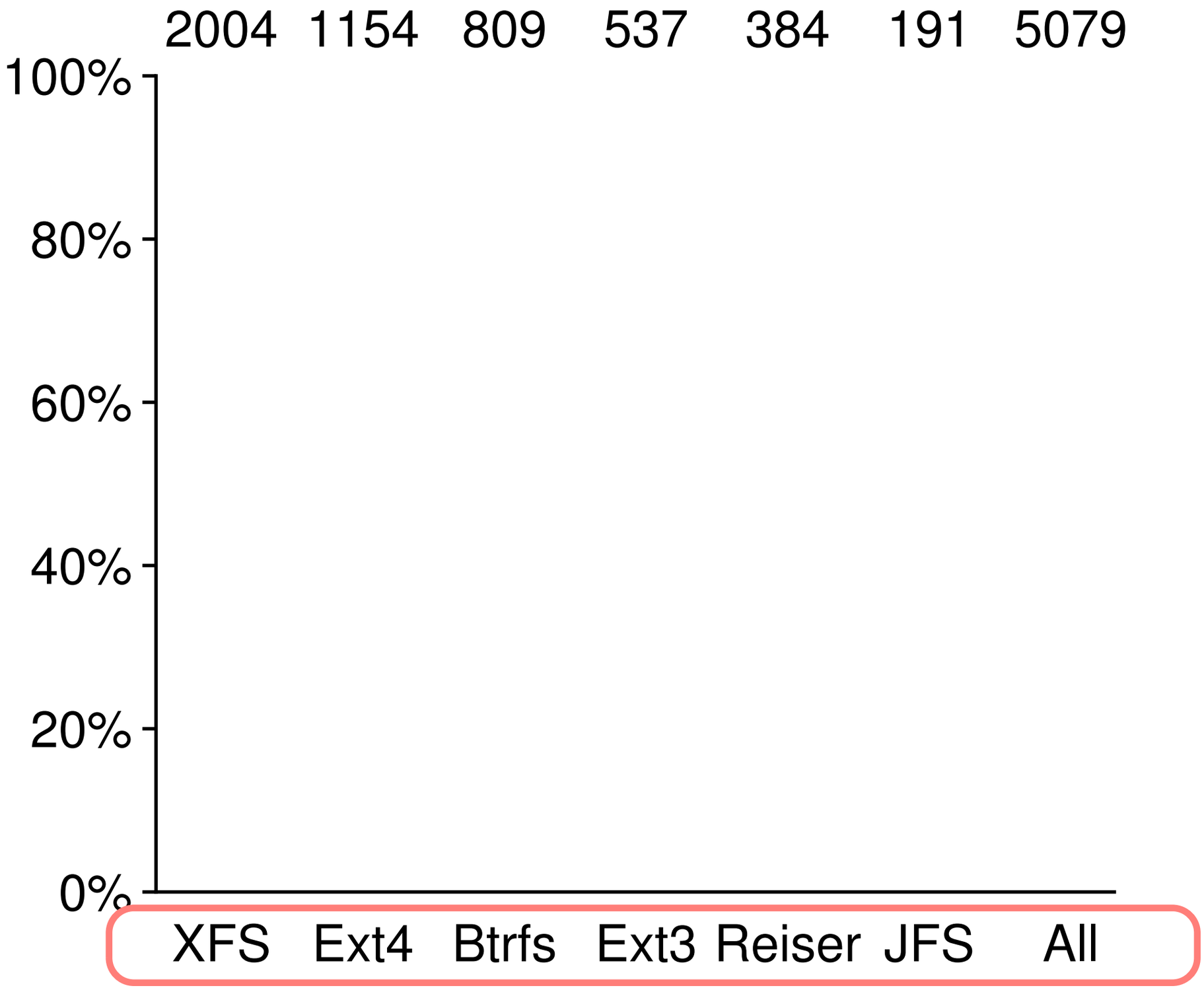
Patch Overview

Type	Description
Bug	Fix existing bugs
Performance	Propose efficient design or implementation
Reliability	Improve robustness
Feature	Add new functionality
Maintenance	Maintain the code and documentation

Patch Overview



Patch Overview

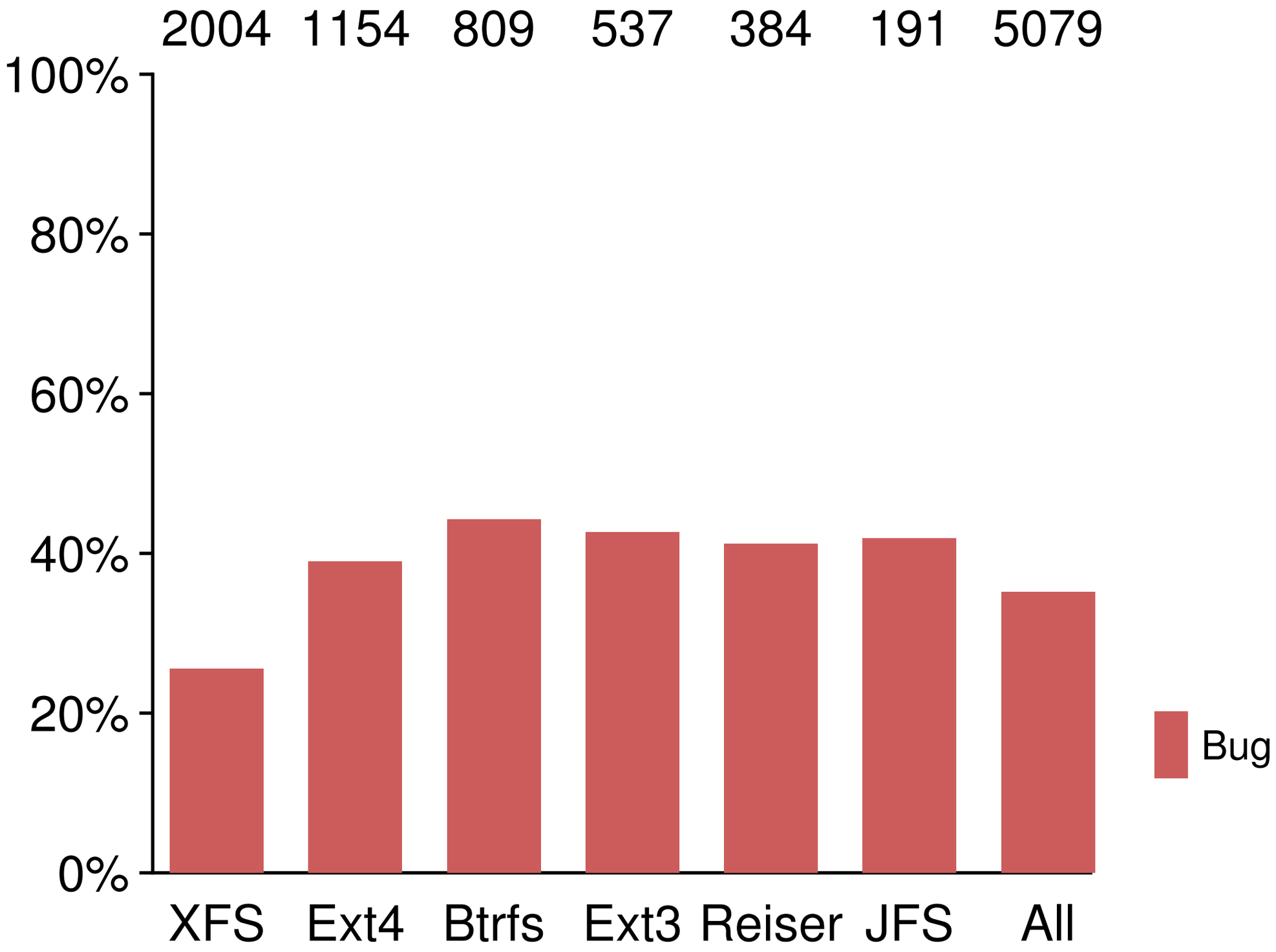


Patch Overview

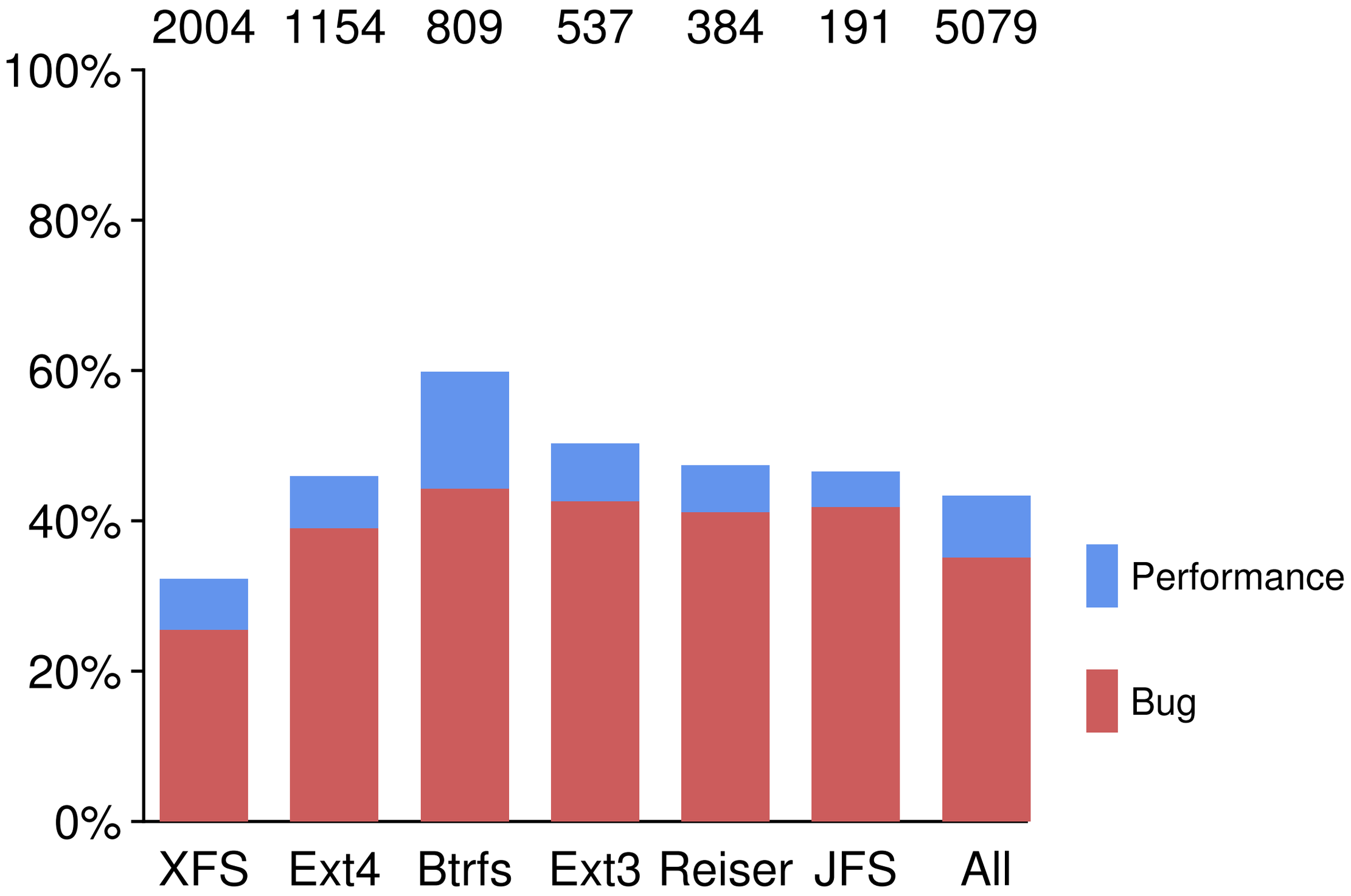
Patch Overview

Patch Overview

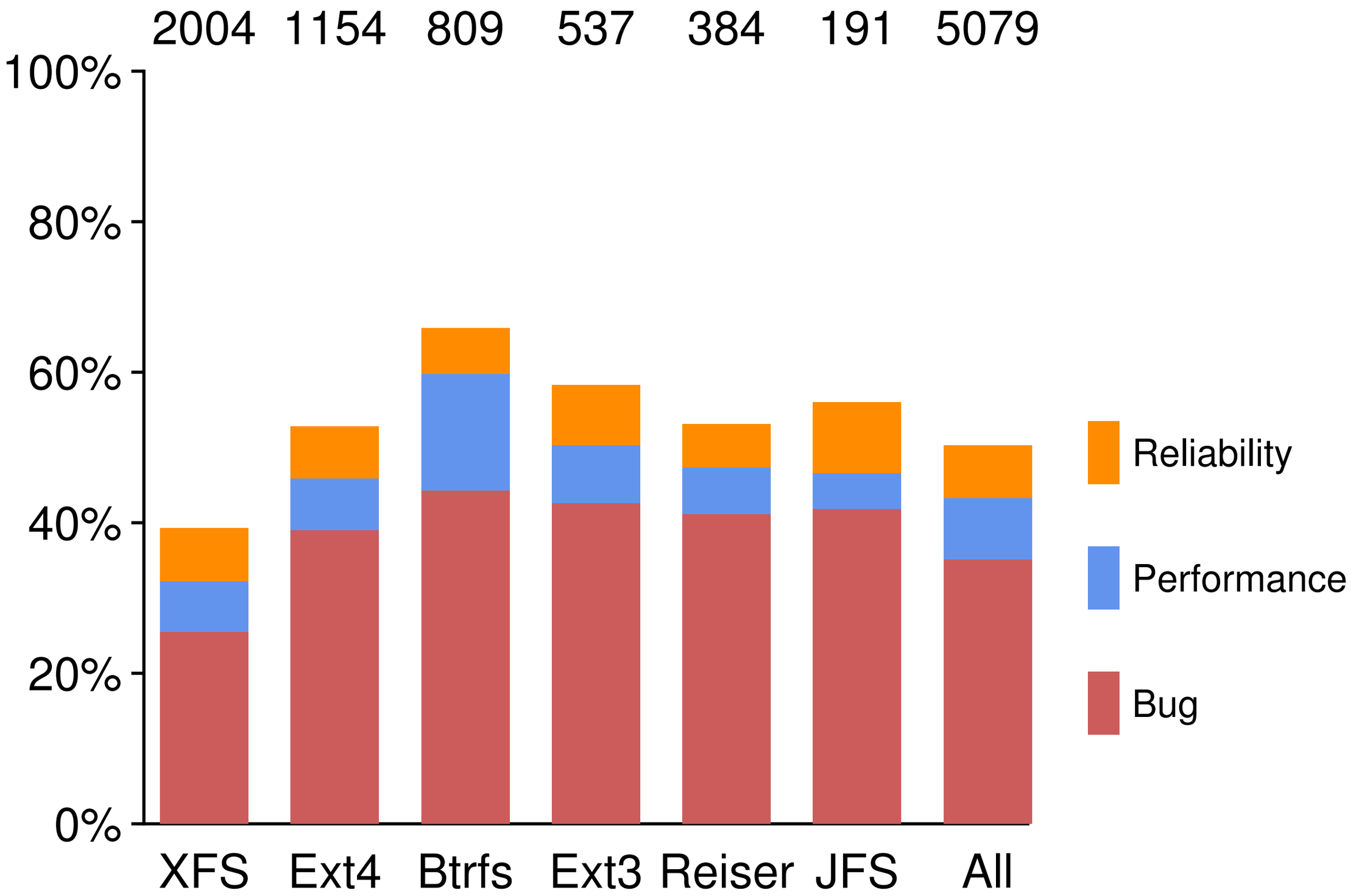
Patch Overview



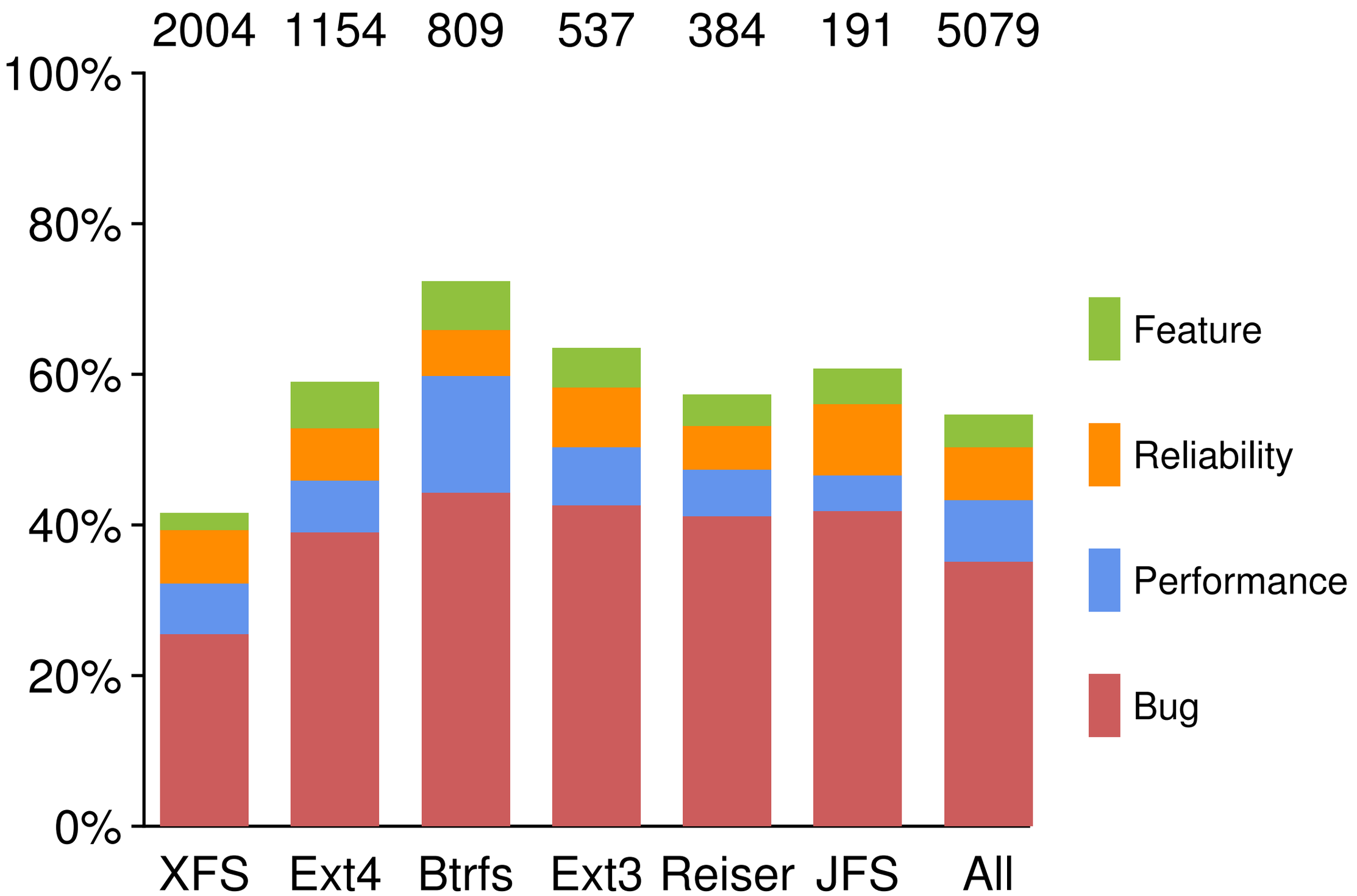
Patch Overview



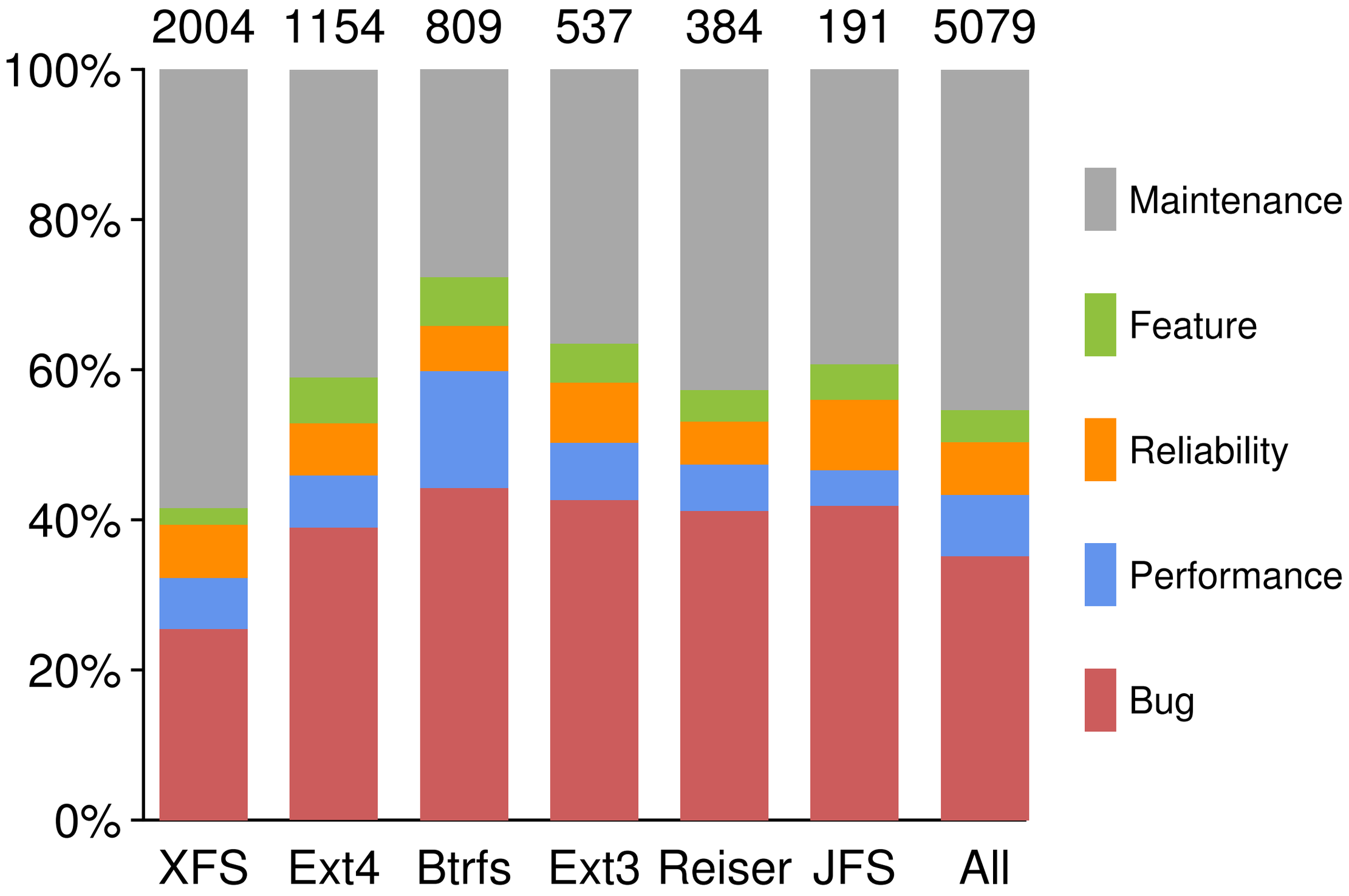
Patch Overview



Patch Overview



Patch Overview



45% of patches are for maintenance

45% of patches are for maintenance

35% of patches are bug fixing

45% of patches are for maintenance

35% of patches are bug fixing

Q2:

What do **bugs** look like ?

Bug Pattern

Bug Pattern

Type	Description
Semantic	Incorrect design or implementation (e.g. incorrect state update, wrong design)
Concurrency	Incorrect concurrent behavior (e.g. miss unlock, deadlock)
Memory	Incorrect handling of memory objects (e.g. resource leak, null dereference)
Error Code	Missing or wrong error code handling (e.g. return wrong error code)

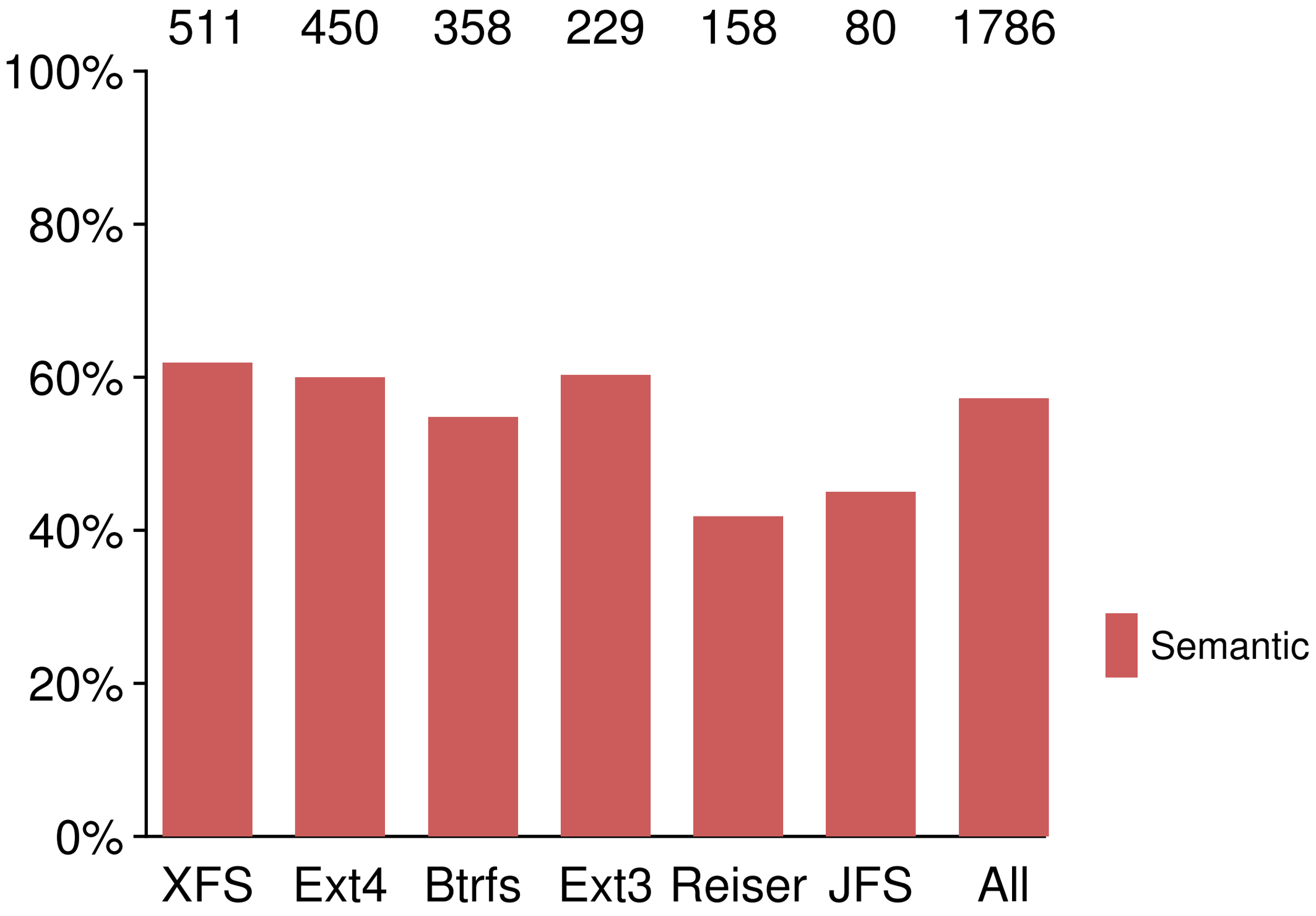
Semantic Bug Example

```
ext3/ialloc.c, 2.6.4

find_group_other(...){
        ... ...

1       group = parent_group + 1;
2       for (i = 2; i < ngroups; i++) { }



        ... ...
}
```

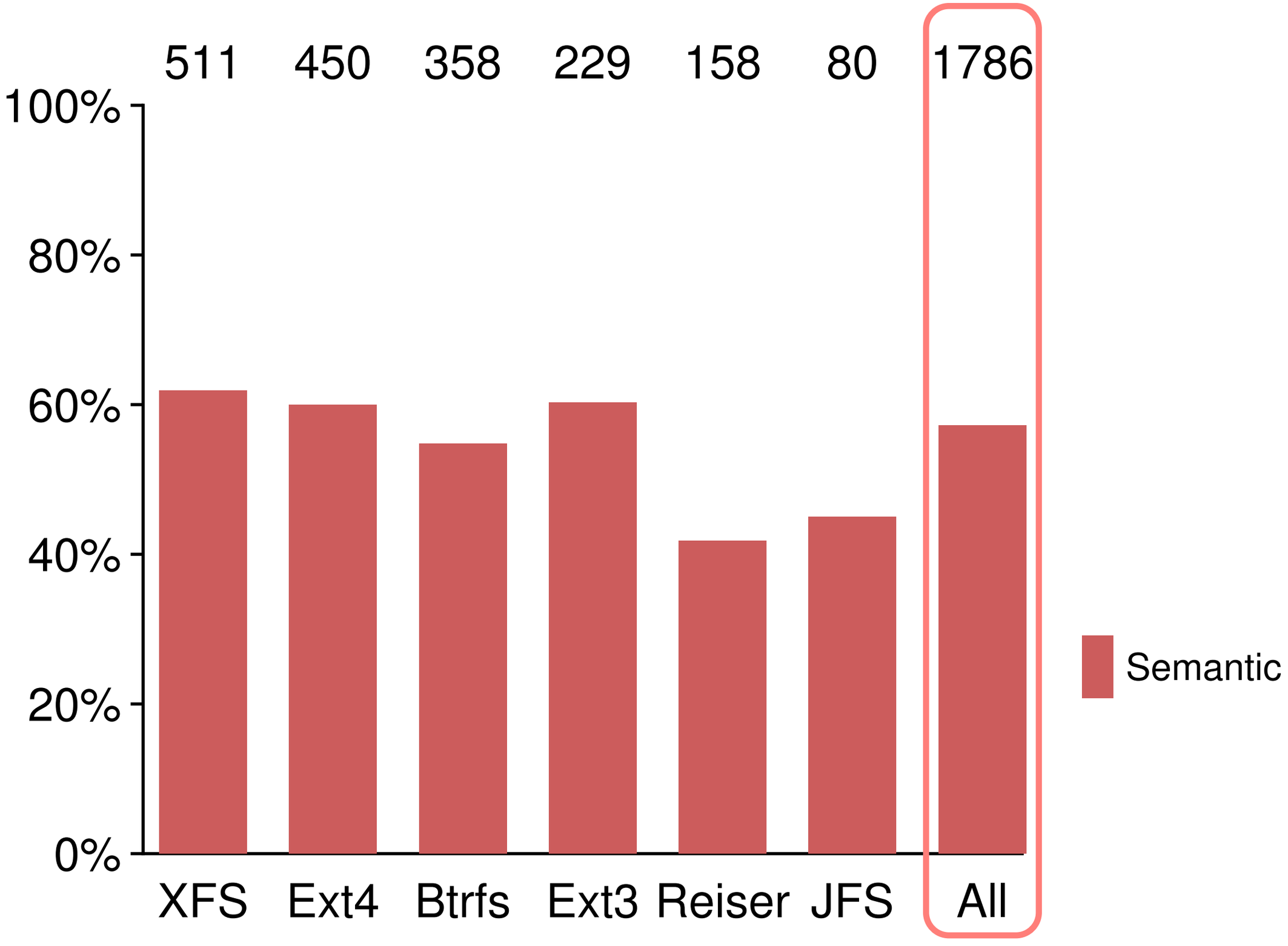
Semantic Bug Example

```
ext3/ialloc.c, 2.6.4

find_group_other(...){
        ... ...
1       group = parent_group;
2       for (i = 0; i < ngroups; i++) { }
        ... ...
}
```

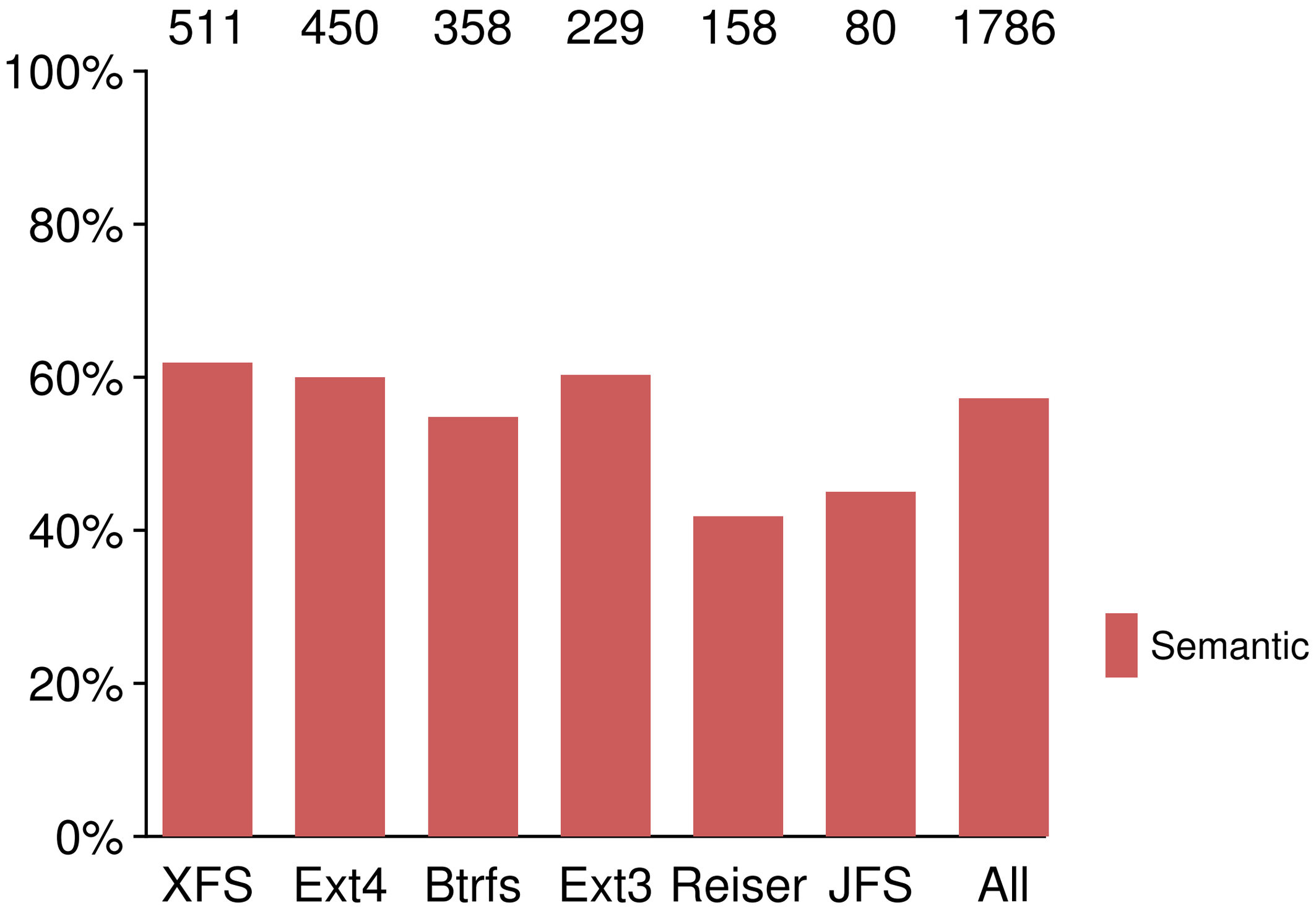
Bug Pattern


511
450
358
229
158
80
1786
100%
80%
60%
40%
20%
0%
XFS
Ext4
Btrfs
Ext3
Reiser
JFS
All
Semantic

Bug Pattern


511
450
358
229
158
80
1786
100%
80%
60%
40%
20%
0%
XFS
Ext4
Btrfs
Ext3
Reiser
JFS
All
Semantic

Bug Pattern


511
450
358
229
158
80
1786
100%
80%
60%
40%
20%
0%
XFS
Ext4
Btrfs
Ext3
Reiser
JFS
All
Semantic

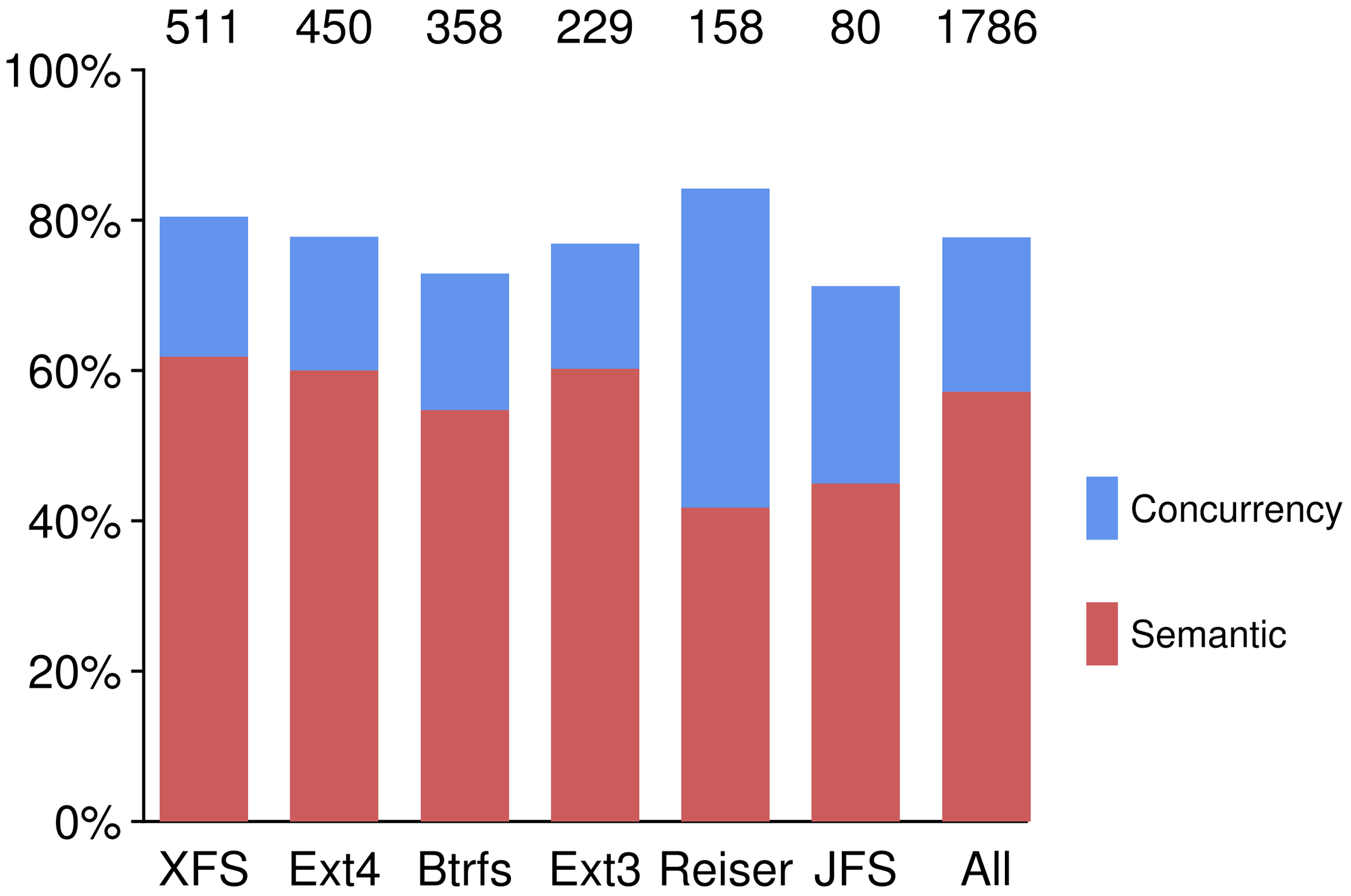
Concurrency Bug Example

```
ext4/extents.c, 2.6.30

ext4_ext_put_in_cache(...){
    ... ...

1       cex = &EXT4_I(inode)->i_cached_extent;
2       cex->ec_FOO = FOO;

}
```

Concurrency Bug Example

```
ext4/extents.c, 2.6.30

ext4_ext_put_in_cache(...){
    ... ...
    spin_lock(i_br_lock);
1     cex = &EXT4_I(inode)->i_cached_extent;
2     cex->ec_FOO = FOO;
    spin_unlock(i_br_lock);
}
```

Bug Pattern


511
450
358
229
158
80
1786
100%
80%
60%
40%
20%
0%
XFS
Ext4
Btrfs
Ext3
Reiser
JFS
All
Concurrency
Semantic

Memory Bug Example

```
  btrfs/inode, 2.6.30

  btrfs_new_inode(...){
1     inode = new_inode(...);
2     ret = btrfs_set_inode_index(...);
3     if (ret){

4         return ERR_PTY(ret);
      }
  }
```

Memory Bug Example

```
  btrfs/inode, 2.6.30

  btrfs_new_inode(...){
1     inode = new_inode(...);
2     ret = btrfs_set_inode_index(...);
3     if (ret){

4         return ERR_PTY(ret);
      }
  }
```

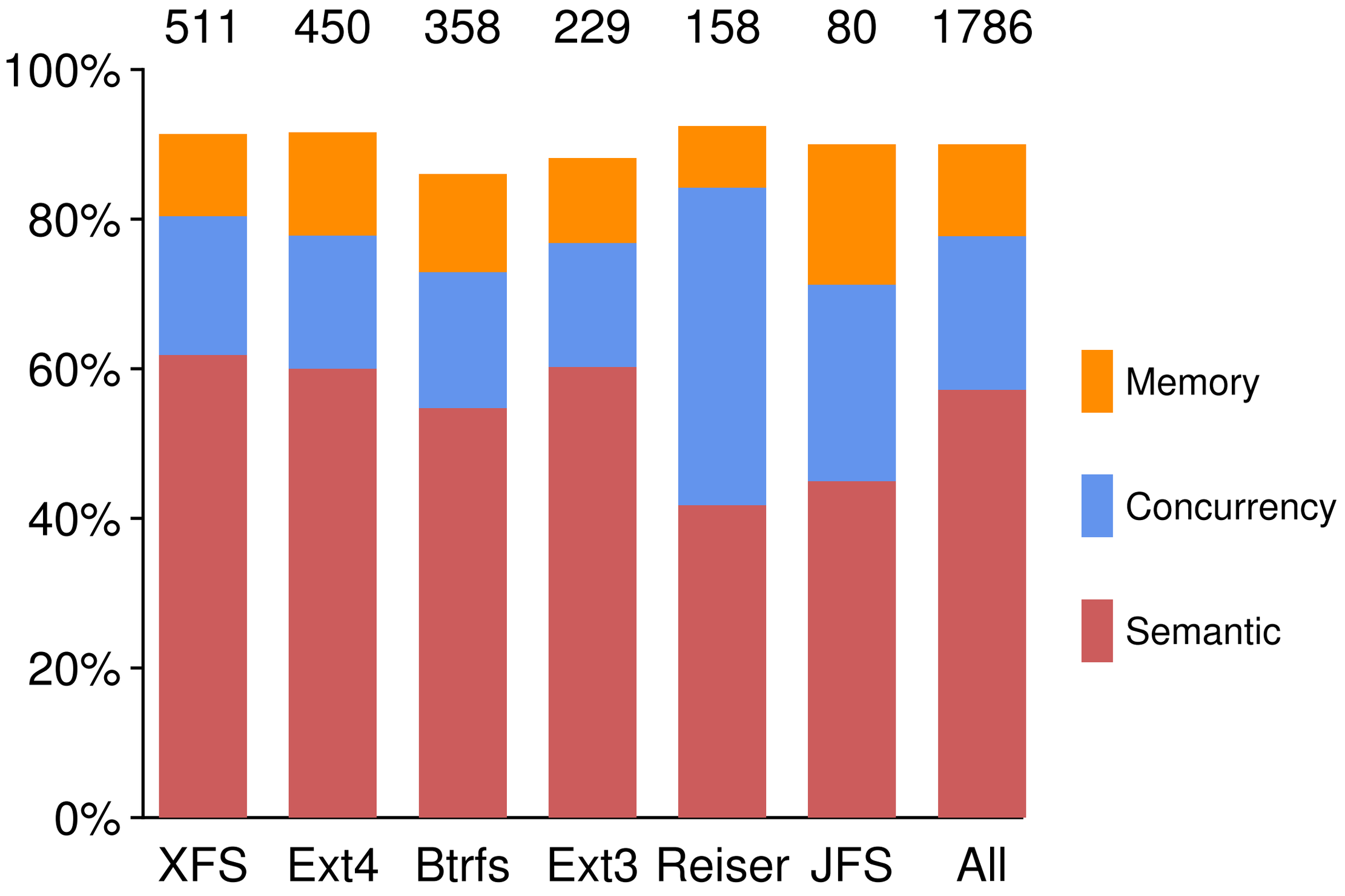
Memory Bug Example

```
  btrfs/inode, 2.6.30

  btrfs_new_inode(...){
1     inode = new_inode(...);
2     ret = btrfs_set_inode_index(...);
3     if (ret){

4         return ERR_PTY(ret);
      }
  }
```

Memory Bug Example

```
  btrfs/inode, 2.6.30

  btrfs_new_inode(...){

1     inode = new_inode(...);
2     ret = btrfs_set_inode_index(...);
3     if (ret){
          iput(inode);
4         return ERR_PTY(ret);
      }
  }
```

Bug Pattern


511
450
358
229
158
80
1786
100%
80%
60%
40%
20%
0%
XFS
Ext4
Btrfs
Ext3
Reiser
JFS
All
Memory
Concurrency
Semantic

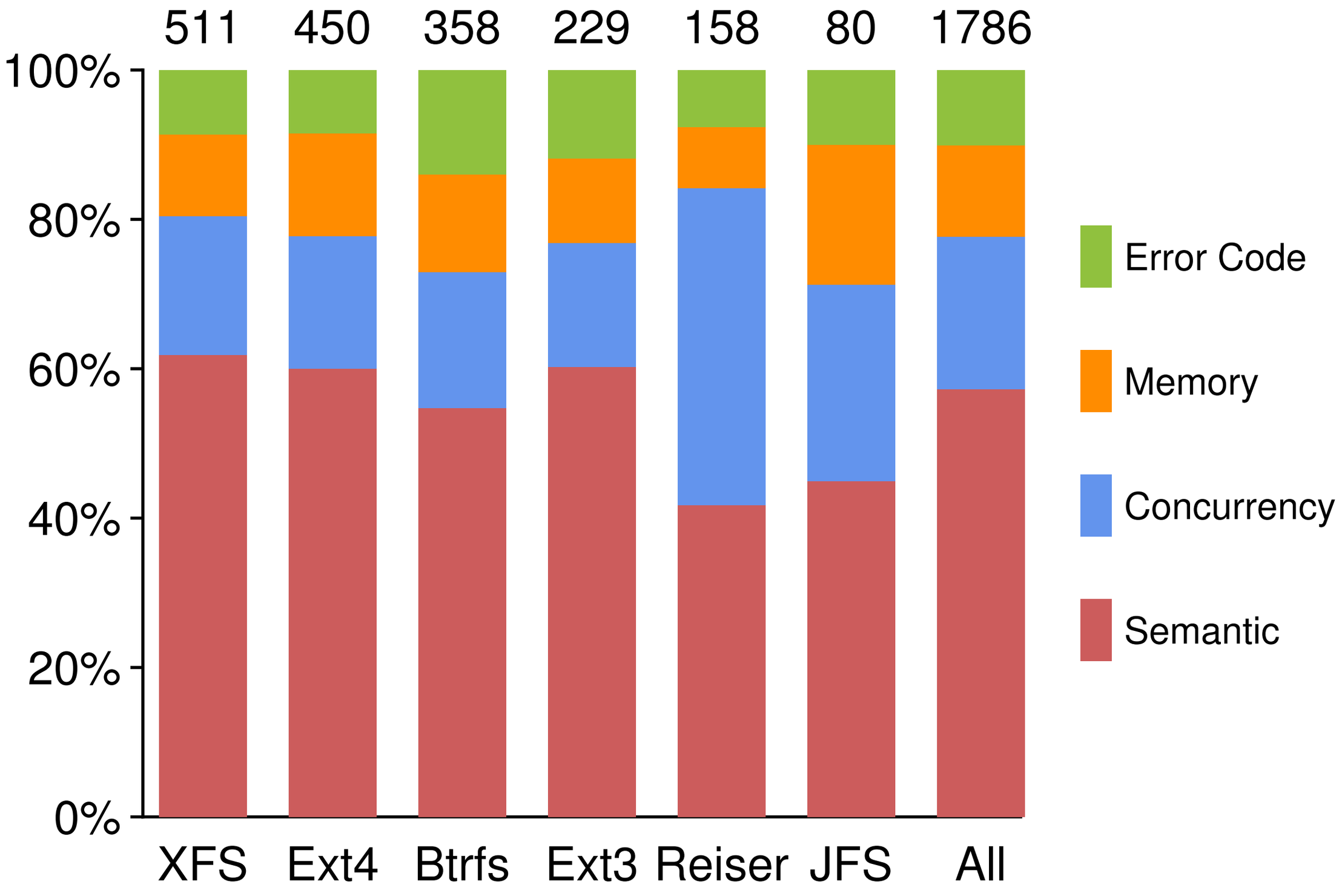
Error Code Example

```
reiserfs/xattr_acl.c, 2.6.16

reiserfs_get_acl(...)
{
        ... ...
1       acl = posix_acl_from_disk(...);

2       *p_acl = posix_acl_dup(acl);

}
```

Error Code Example

```
reiserfs/xattr_acl.c, 2.6.16

reiserfs_get_acl(...)
{
        ... ...
1       acl = posix_acl_from_disk(...);
        if (!IS_ERR(acl))
2           *p_acl = posix_acl_dup(acl);

}
```

Bug Pattern

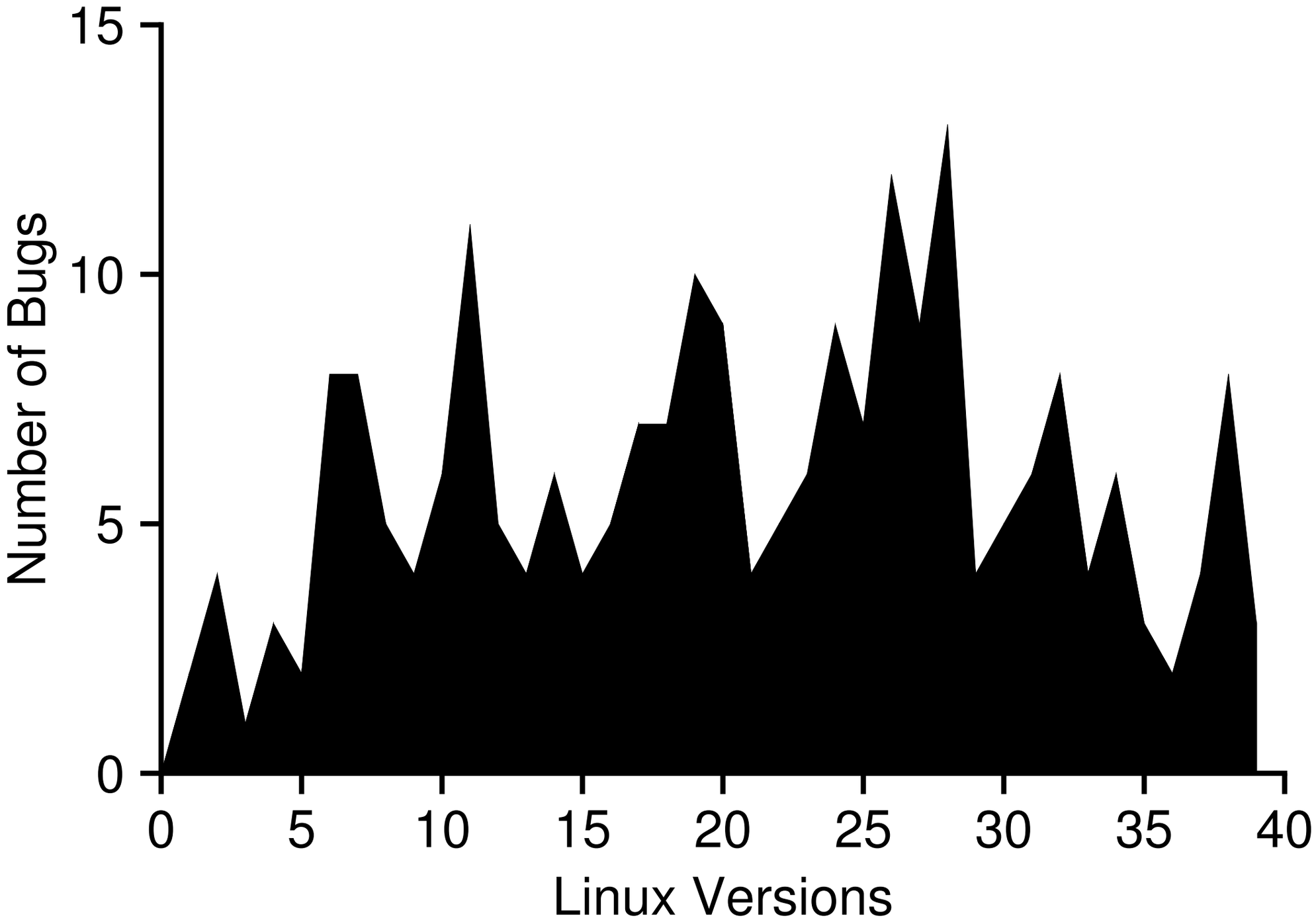

511
450
358
229
158
80
1786
100%
80%
60%
40%
20%
0%
XFS
Ext4
Btrfs
Ext3
Reiser
JFS
All
Error Code
Memory
Concurrency
Semantic

55% of file-system bugs are semantic bugs

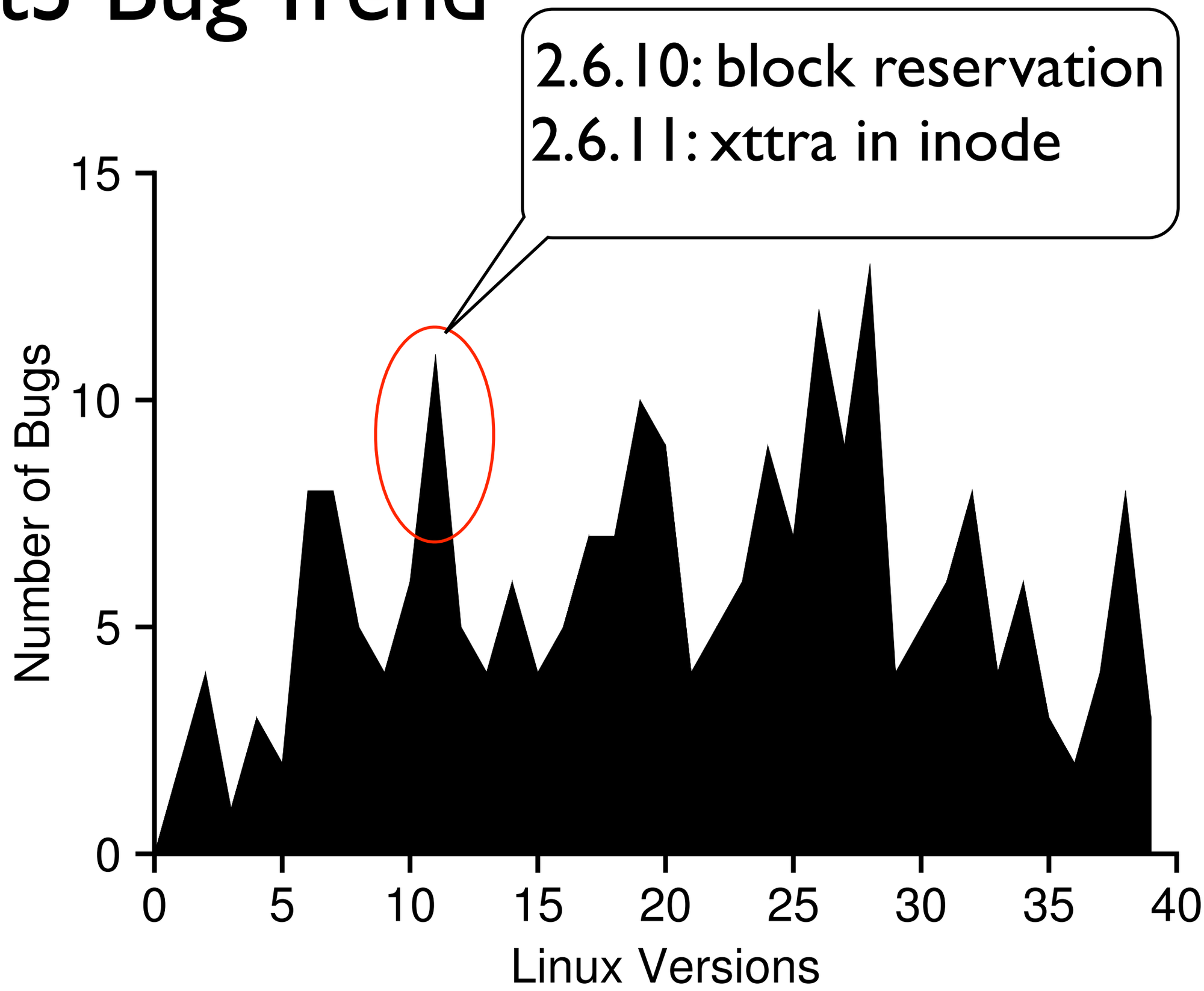
Q3:

Do bugs diminish over time ?

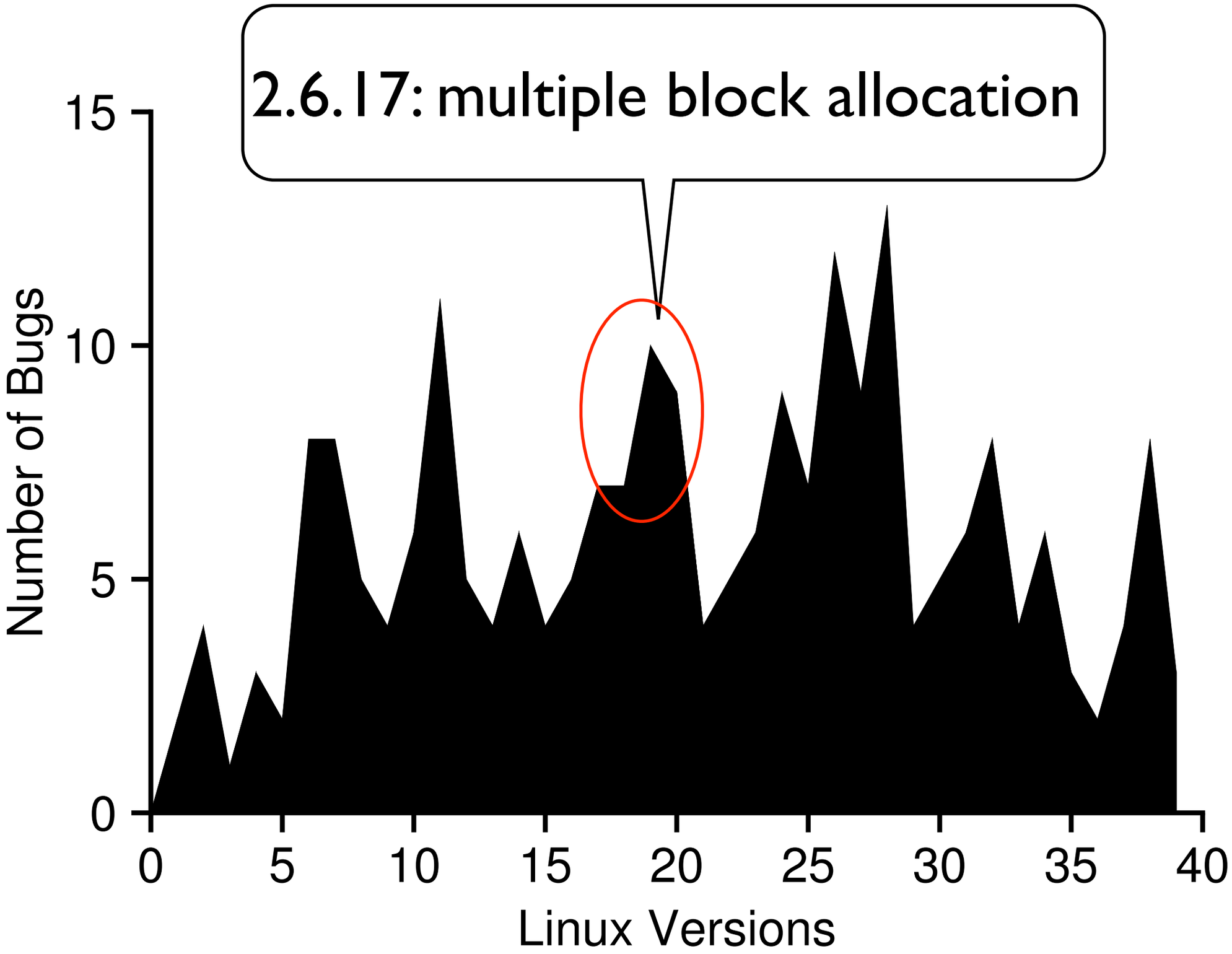
Ext3 Bug Trend



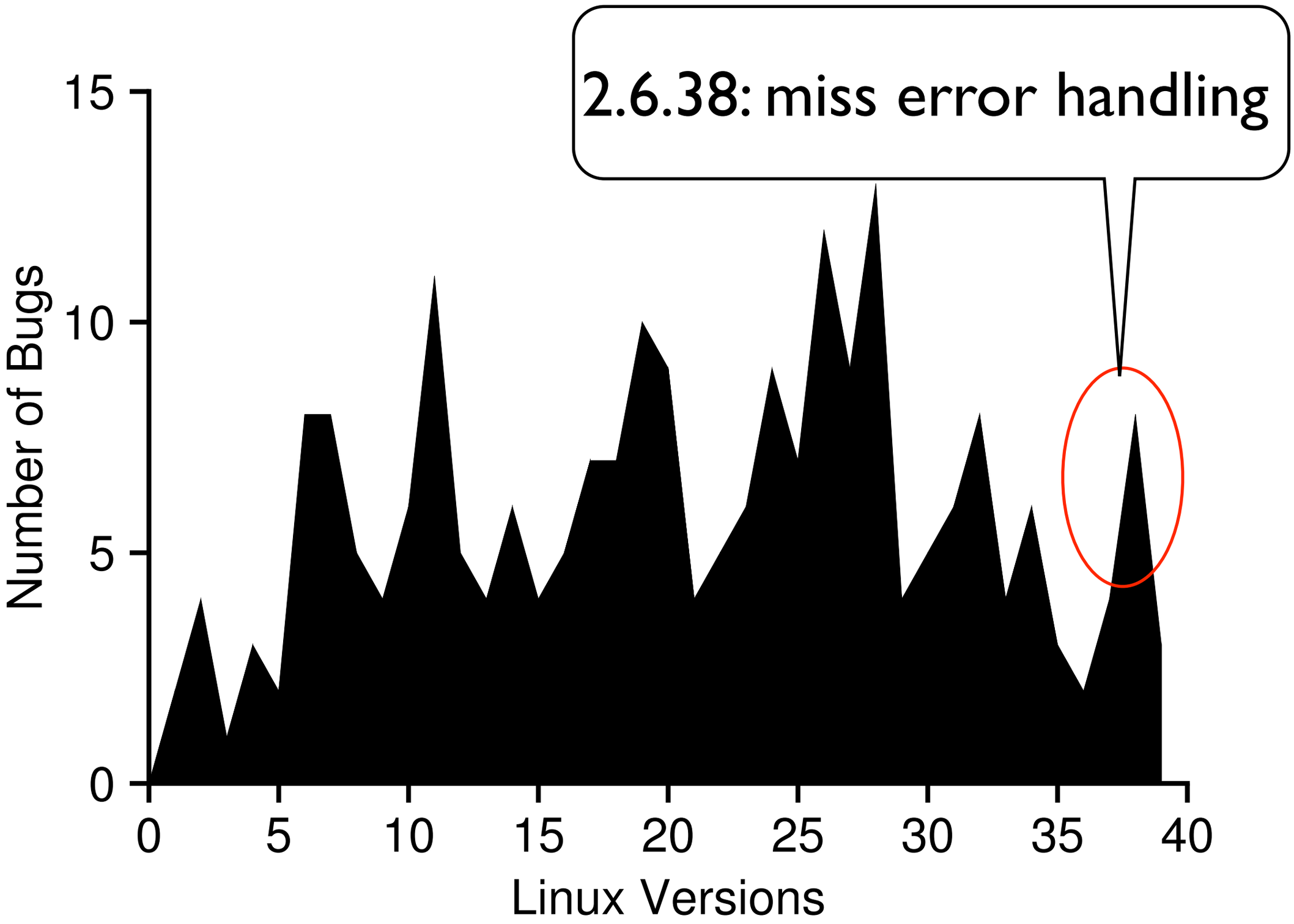
Ext3 Bug Trend



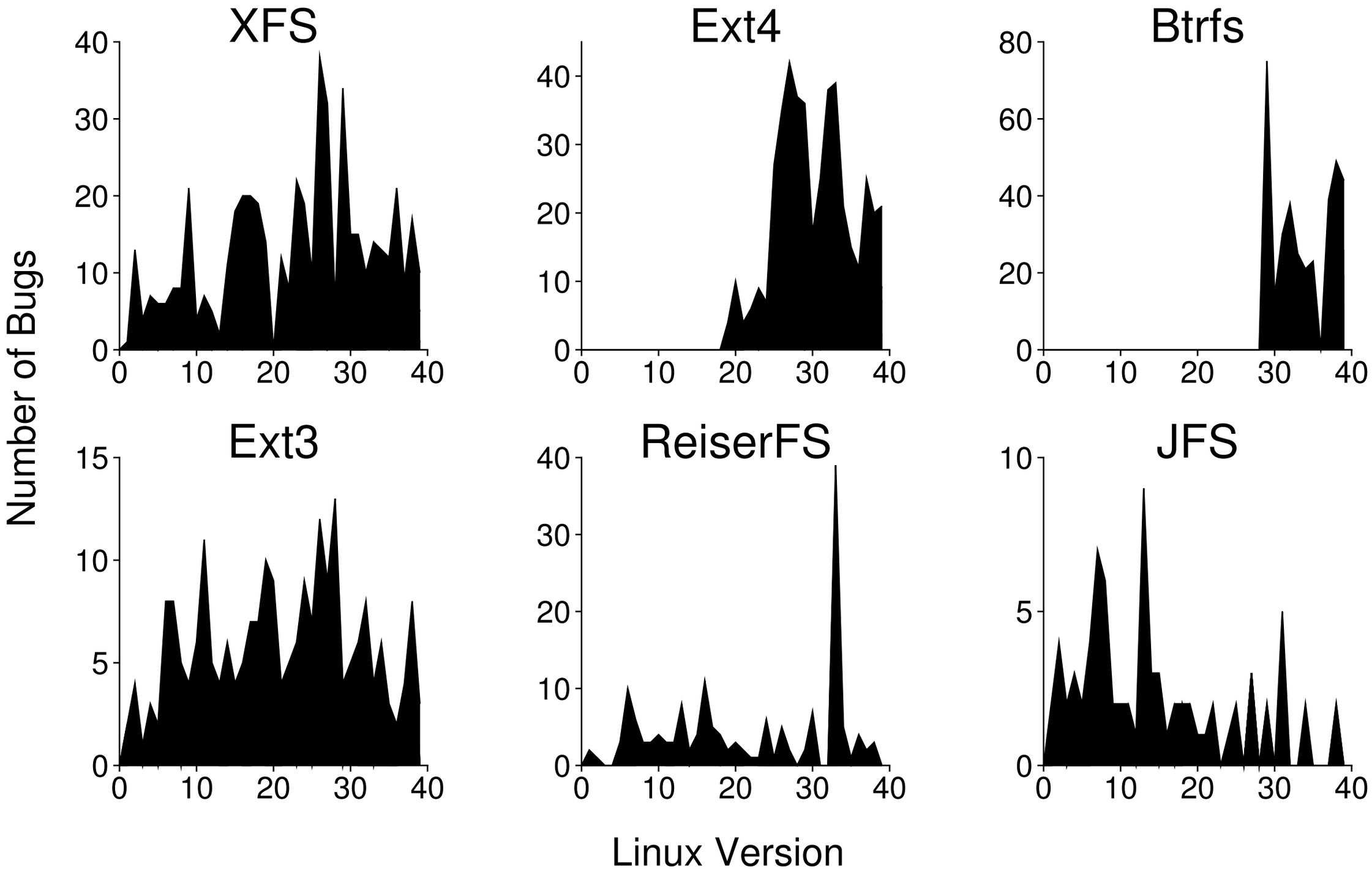
Ext3 Bug Trend



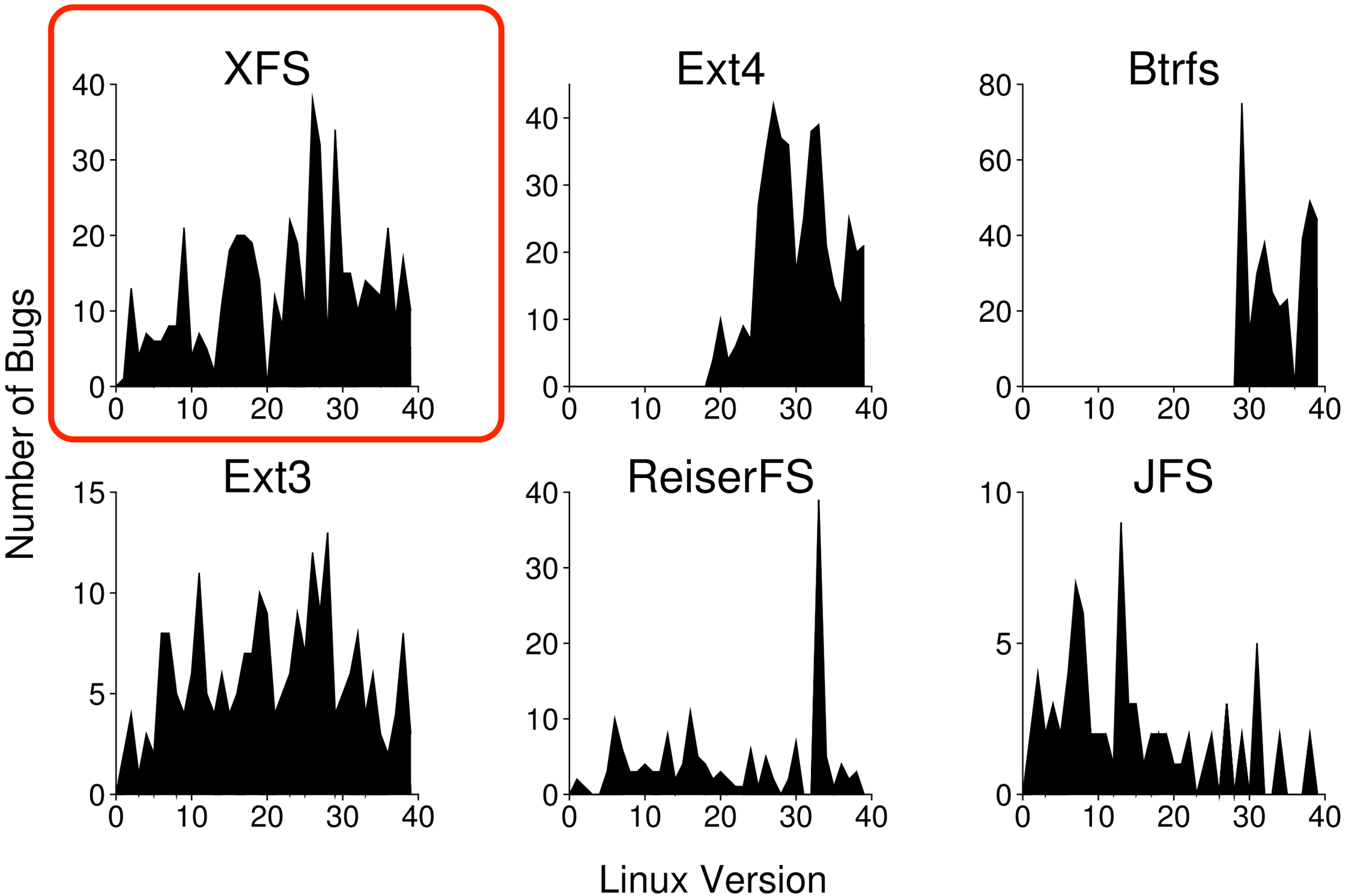
Ext3 Bug Trend



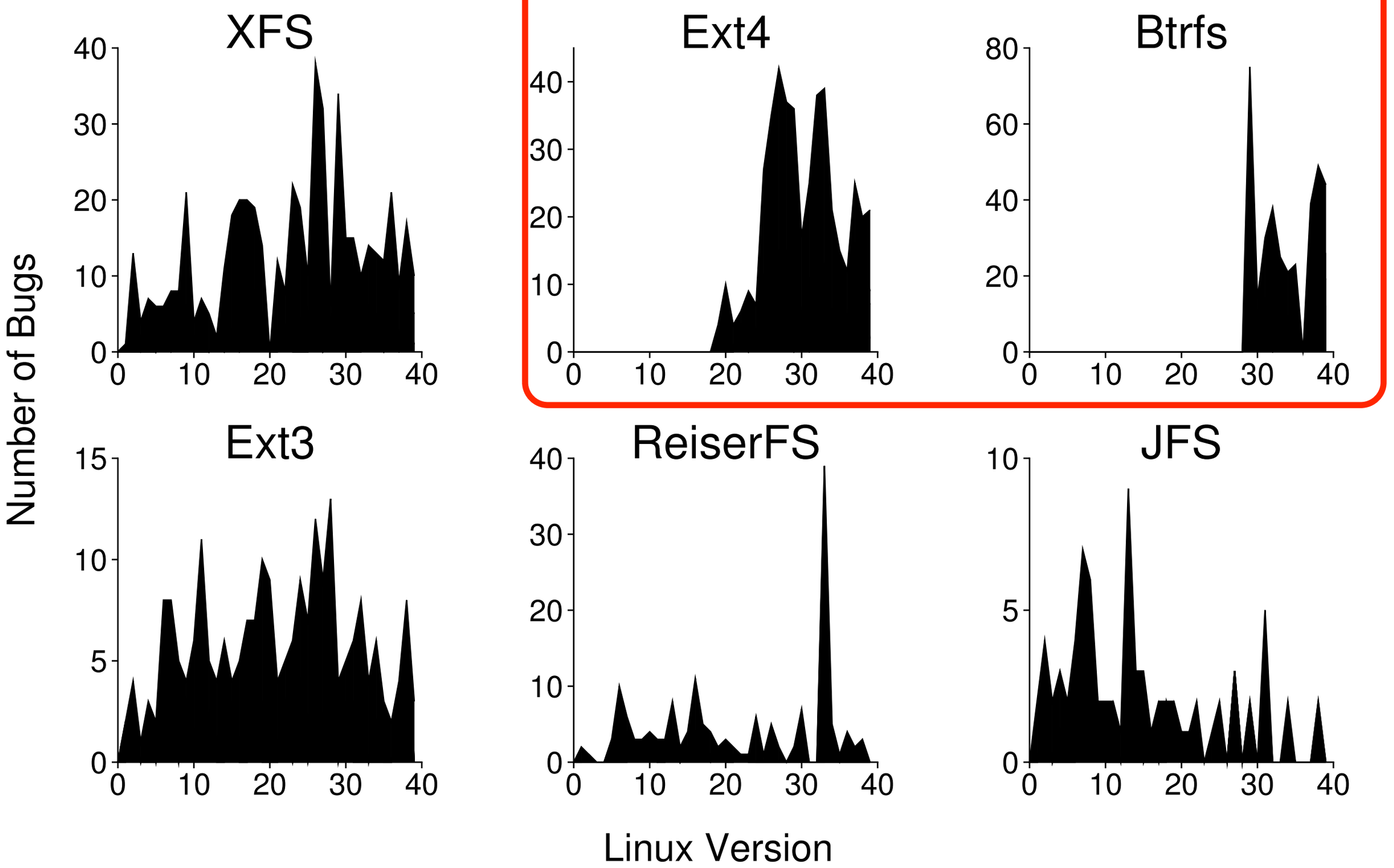
Bug Trend


XFS
Ext4
Btrfs
Ext3
ReiserFS
JFS
Number of Bugs
Linux Version

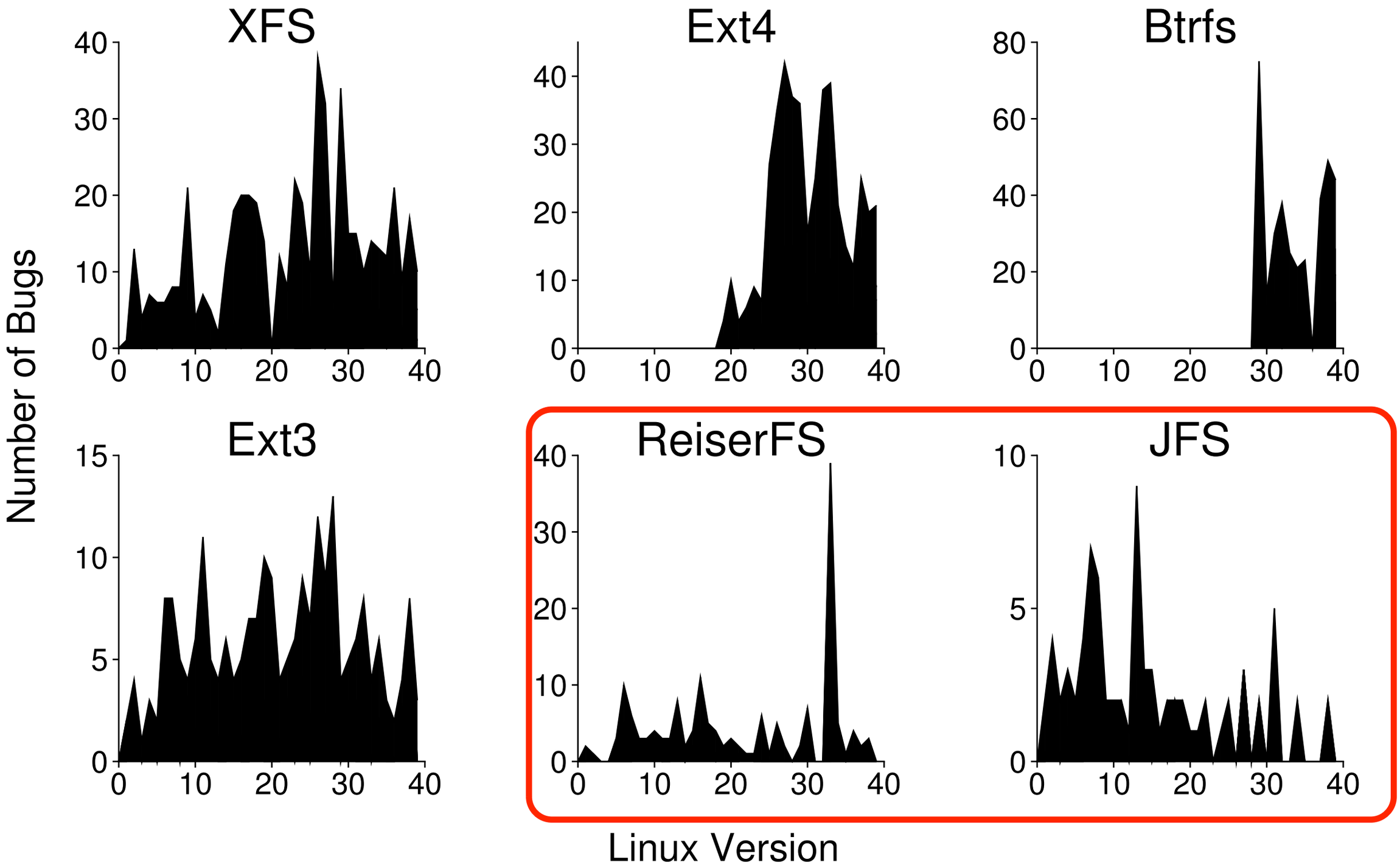
Bug Trend

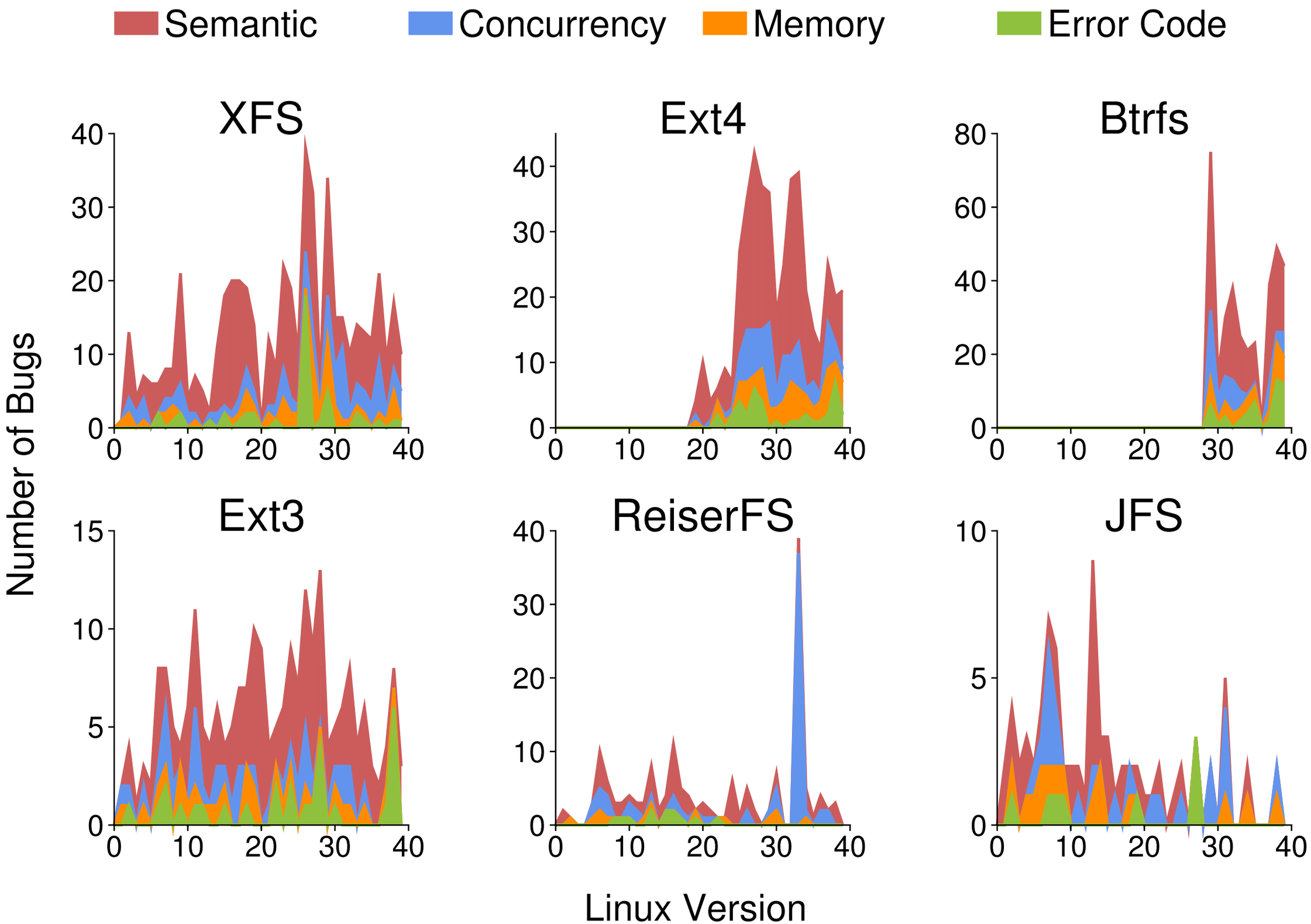

XFS
Ext4
Btrfs
Ext3
ReiserFS
JFS
Number of Bugs
Linux Version

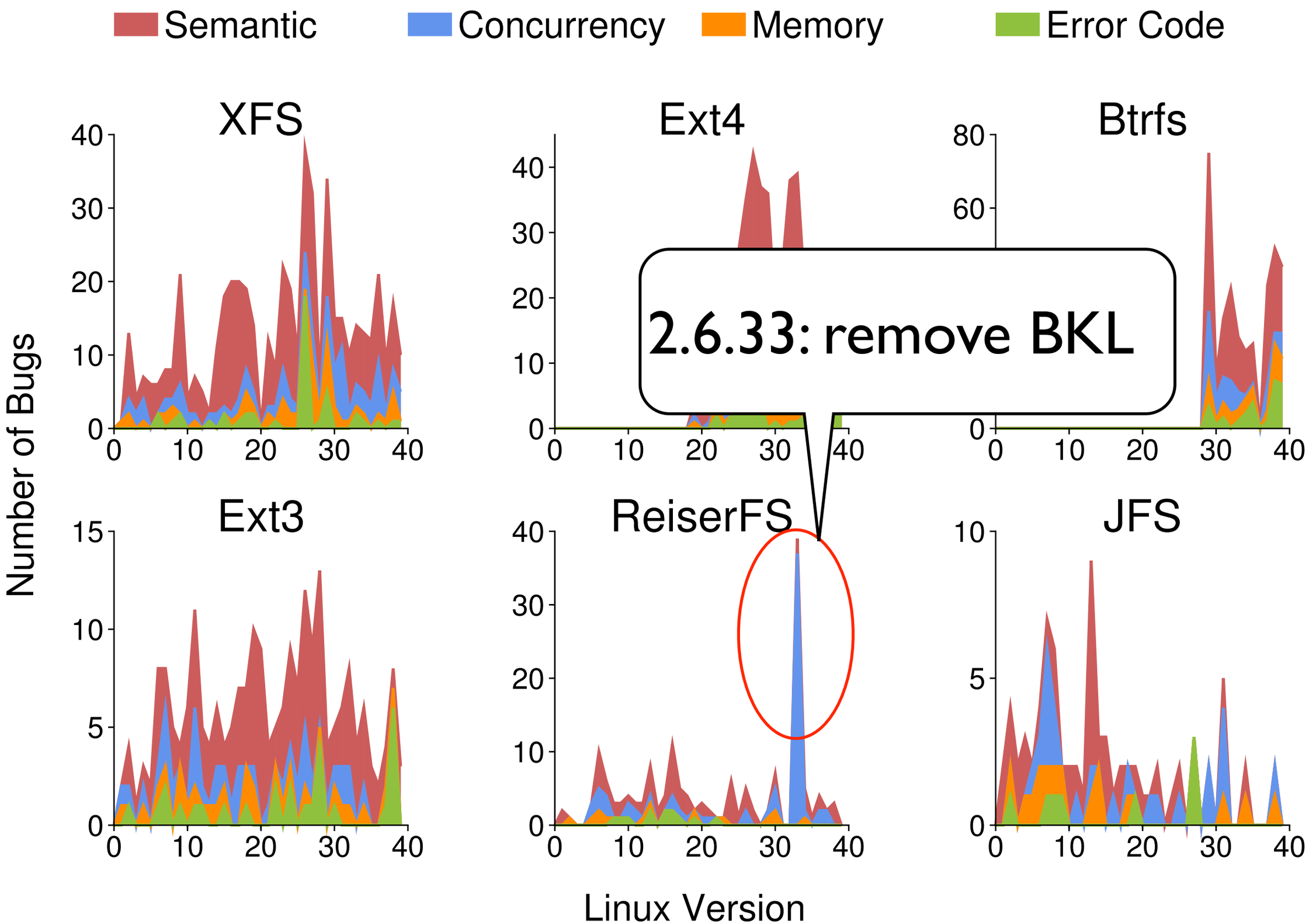
Bug Trend


XFS
Ext4
Btrfs
Ext3
ReiserFS
JFS
Number of Bugs
Linux Version

Bug Trend


XFS
Ext4
Btrfs
Ext3
ReiserFS
JFS
Number of Bugs
Linux Version


Semantic
Concurrency
Memory
Error Code
XFS
Ext4
Btrfs
Ext3
ReiserFS
JFS
Number of Bugs
Linux Version


Semantic
Concurrency
Memory
Error Code
XFS
Ext4
Btrfs
Ext3
ReiserFS
JFS
2.6.33: remove BKL
Number of Bugs
Linux Version

Bug-fixing is a Constant

in a file system's lifetime

Q4:

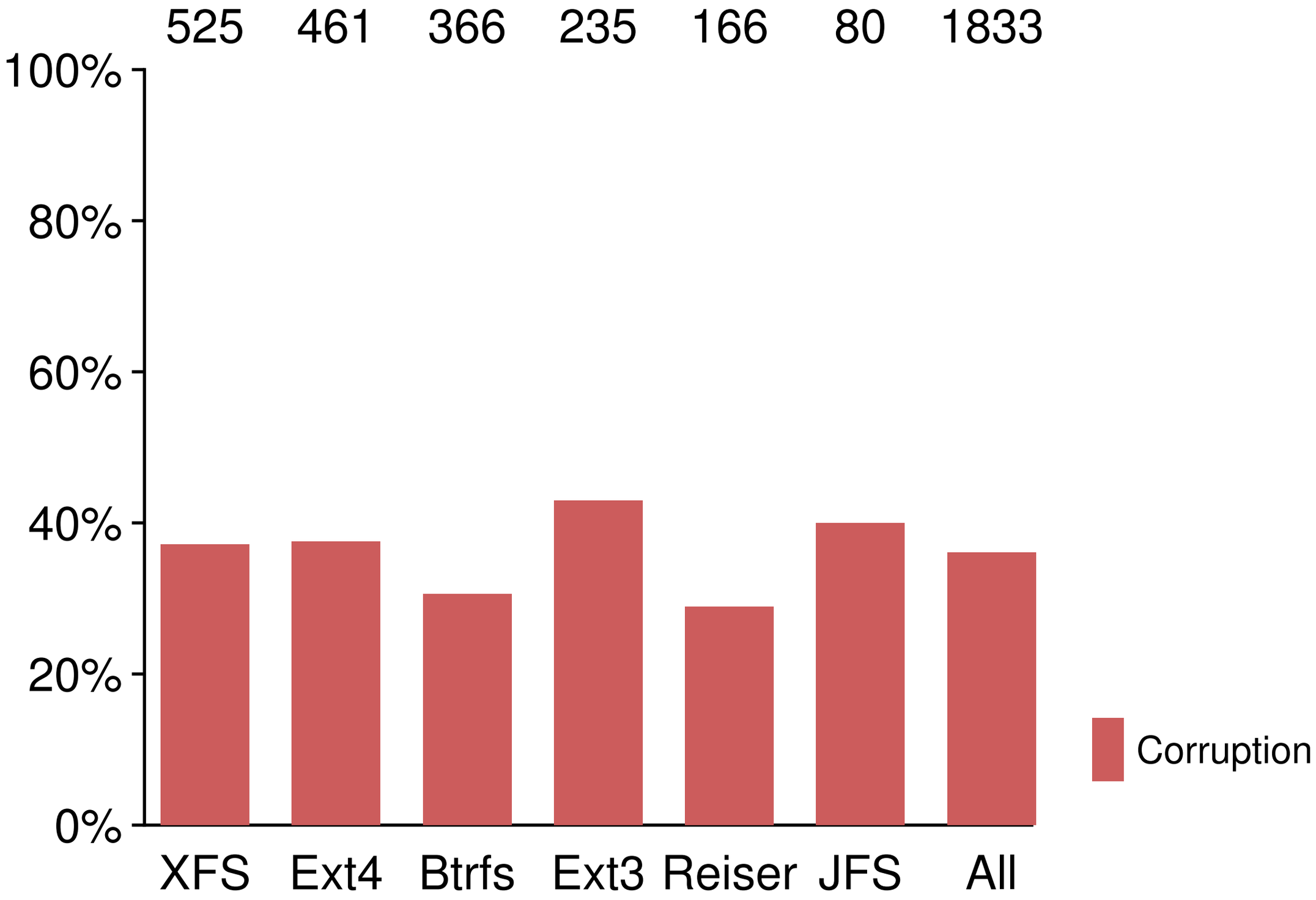
What consequences do file-system bugs have ?

Bug Consequence

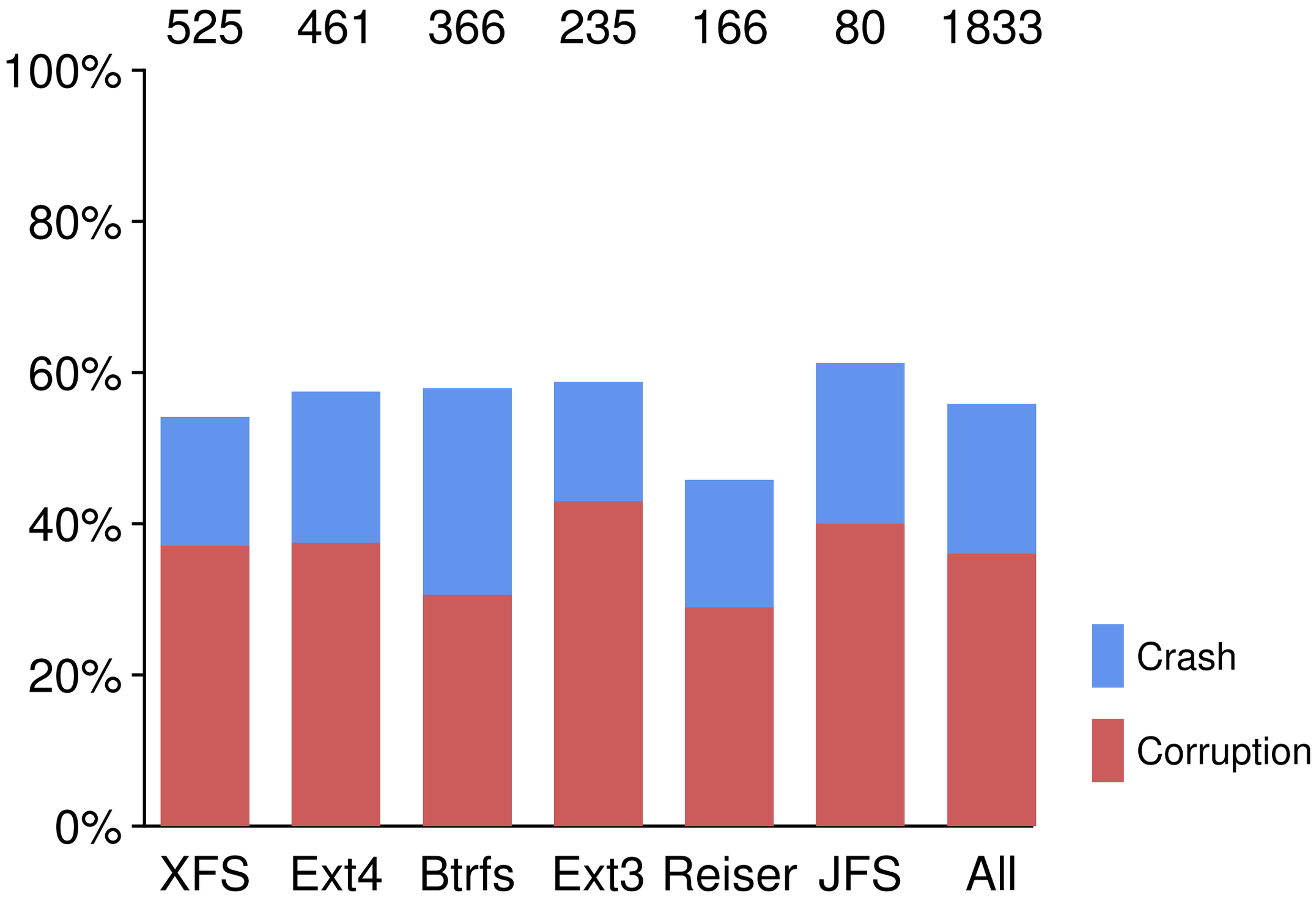
Bug Consequence

Type	Description
Corruption	On-disk or in-memory data is corrupted
Crash	File system becomes unusable
Error	Unexpected operation failure or error code
Deadlock	Wait for resources in circular chain
Hang	File system makes no progress
Leak	Resources are not freed properly
Wrong	Diverts from expectation (exclude above)

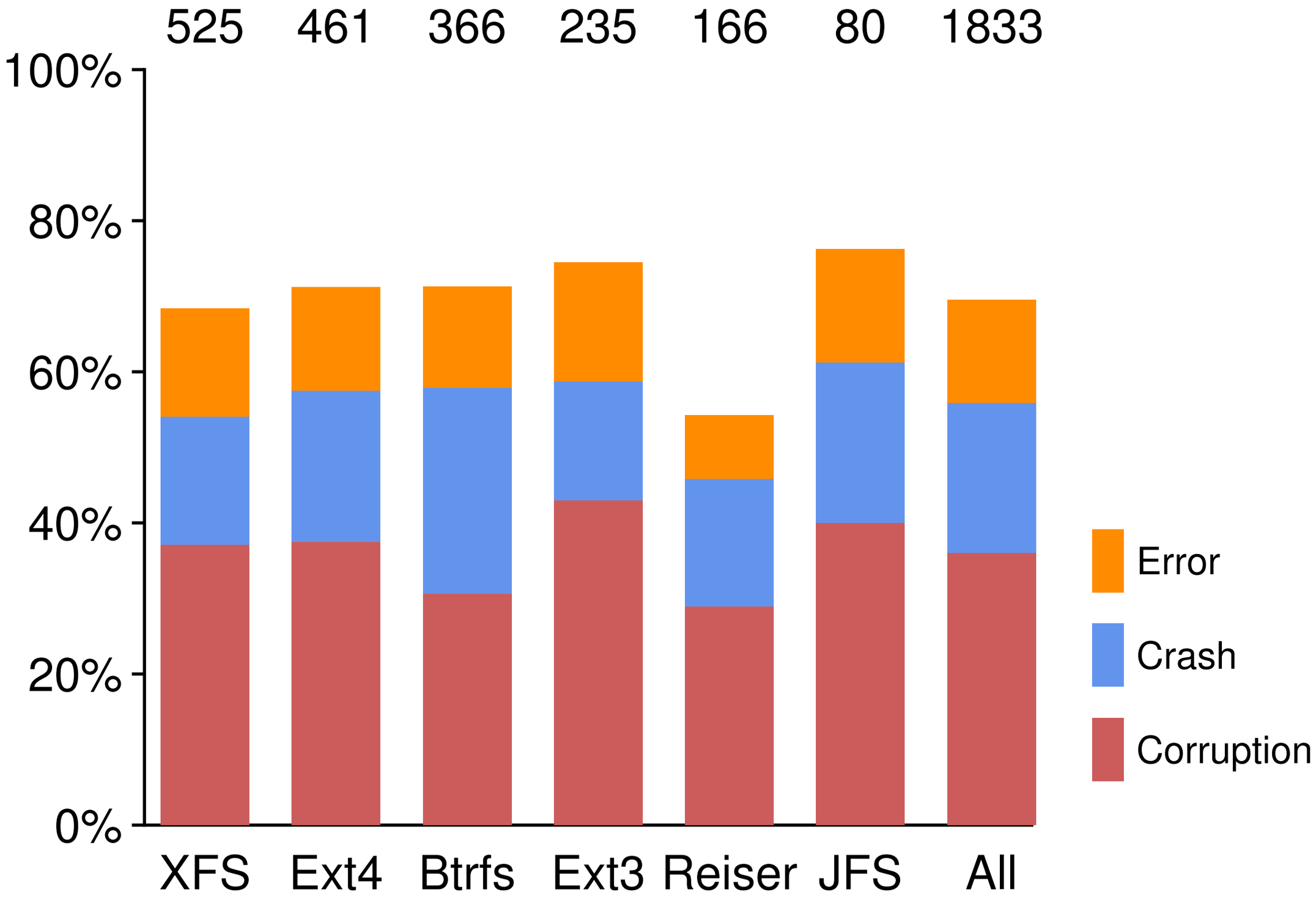
Bug Consequence



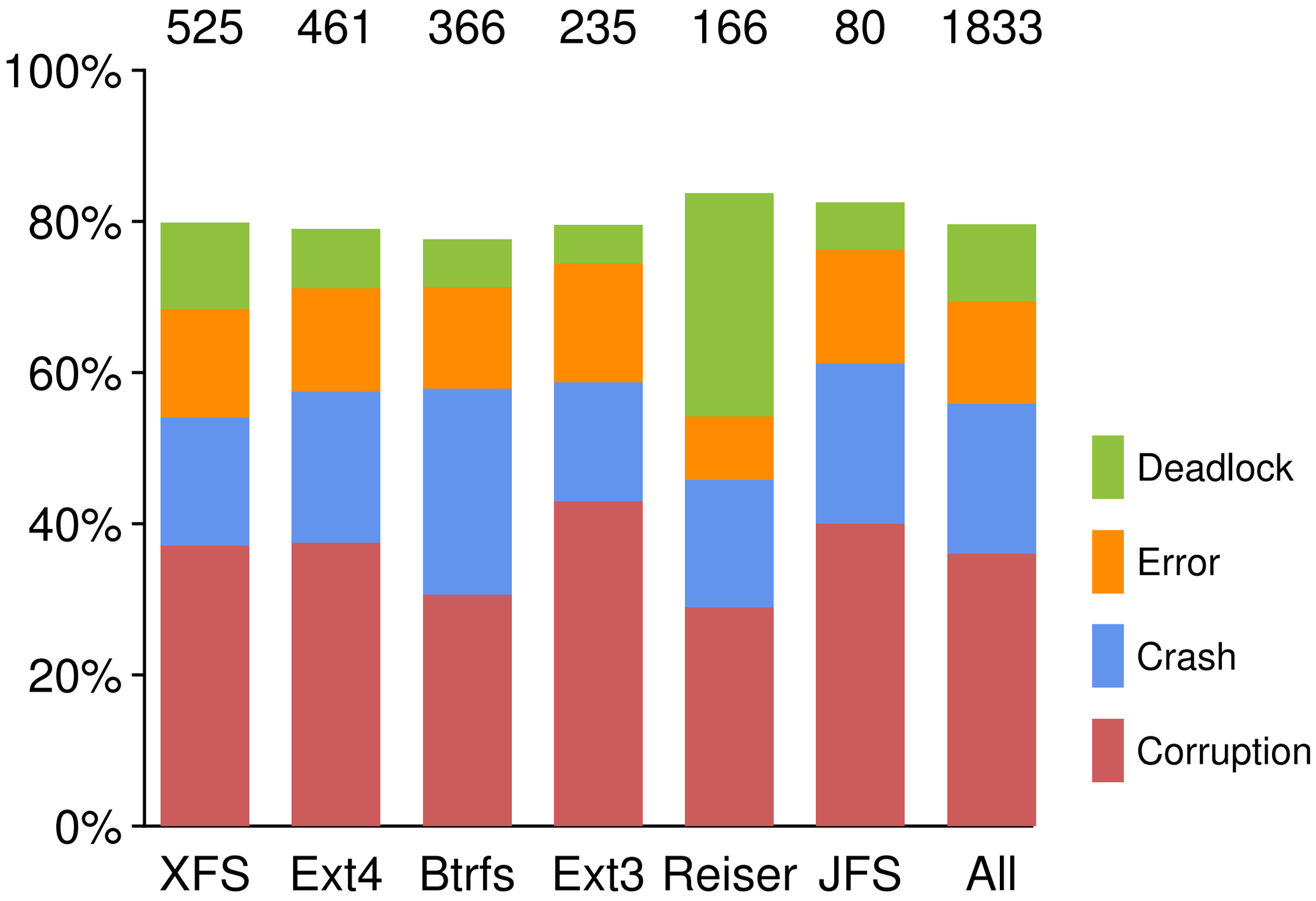
Bug Consequence



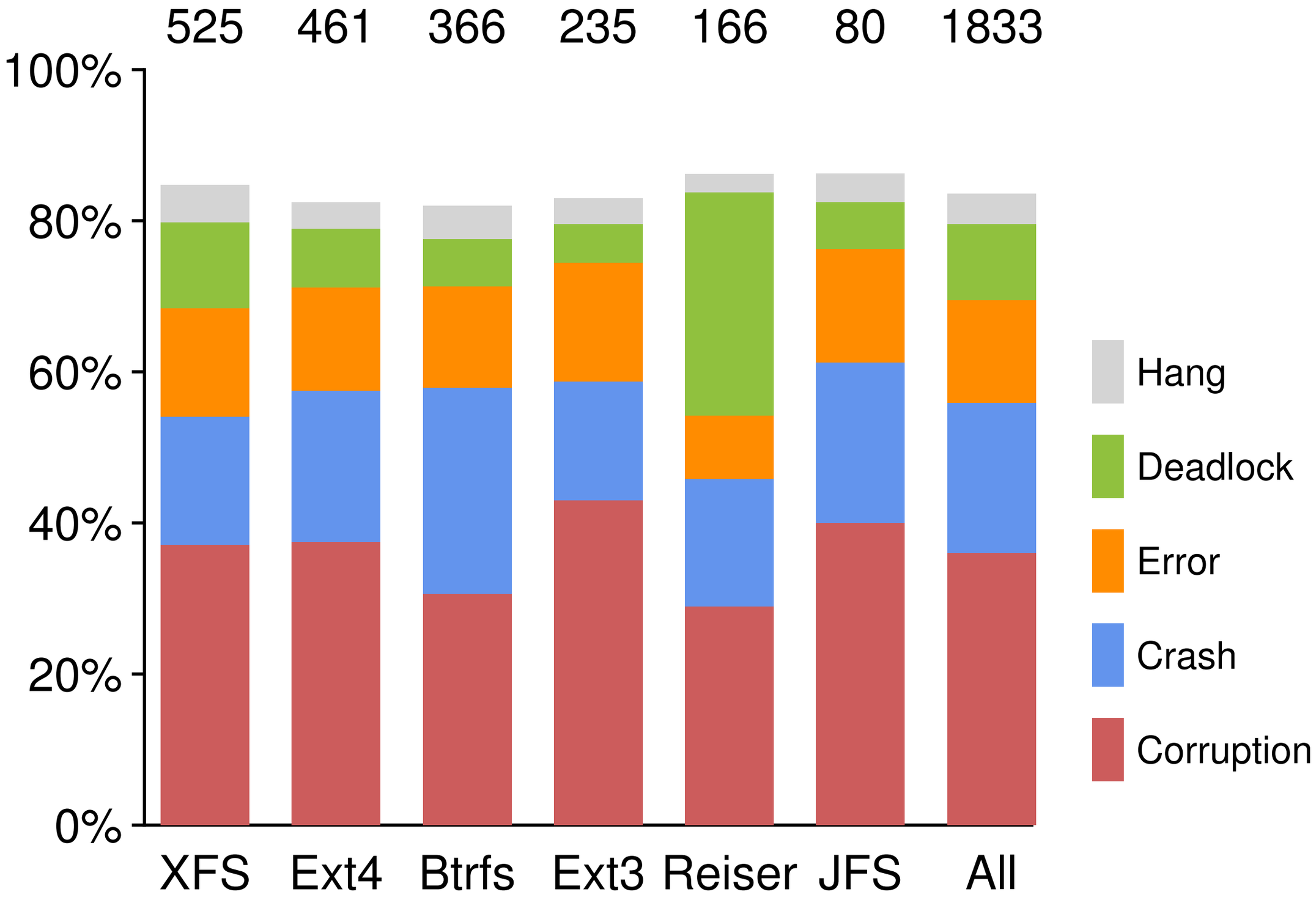
Bug Consequence



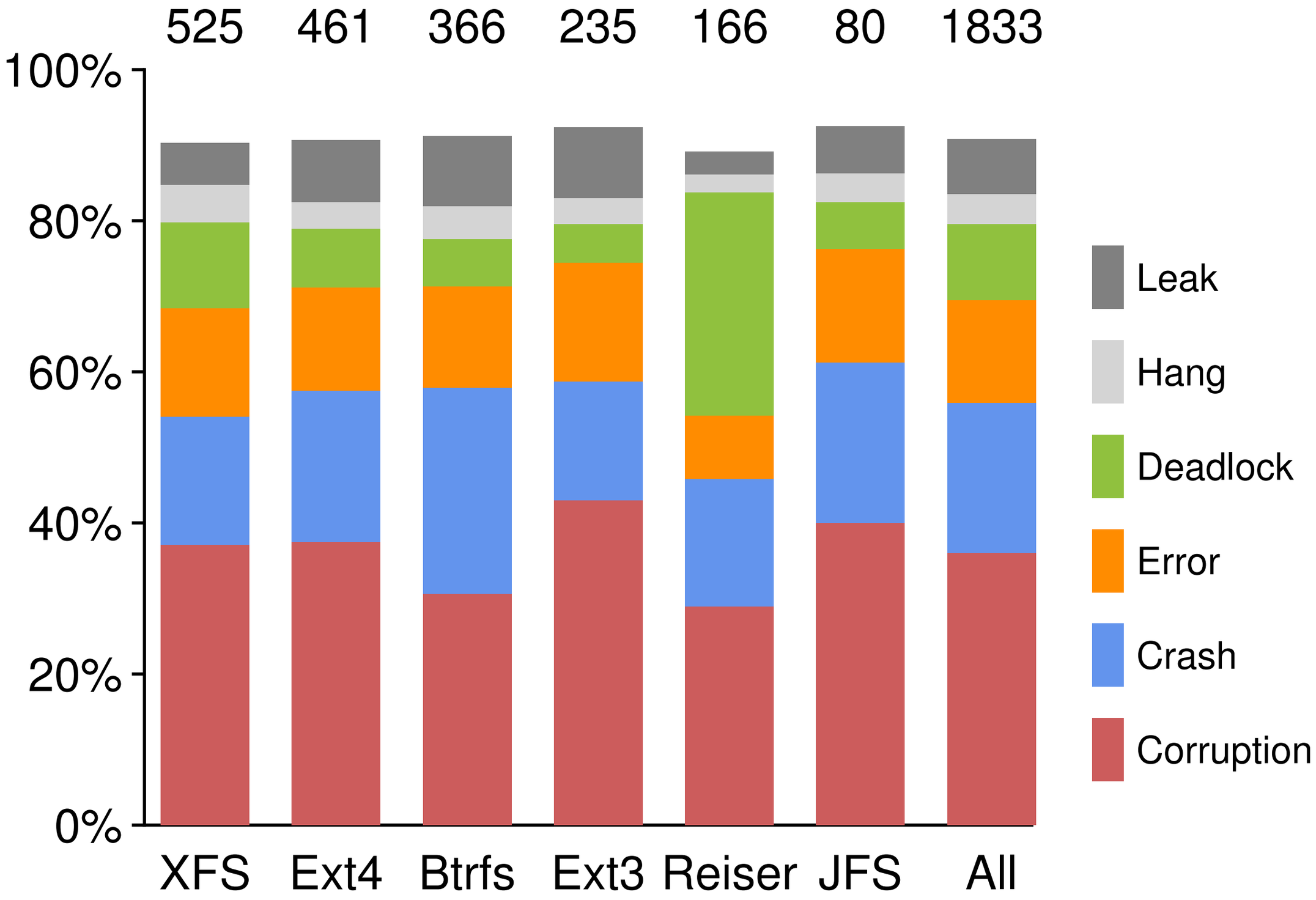
Bug Consequence



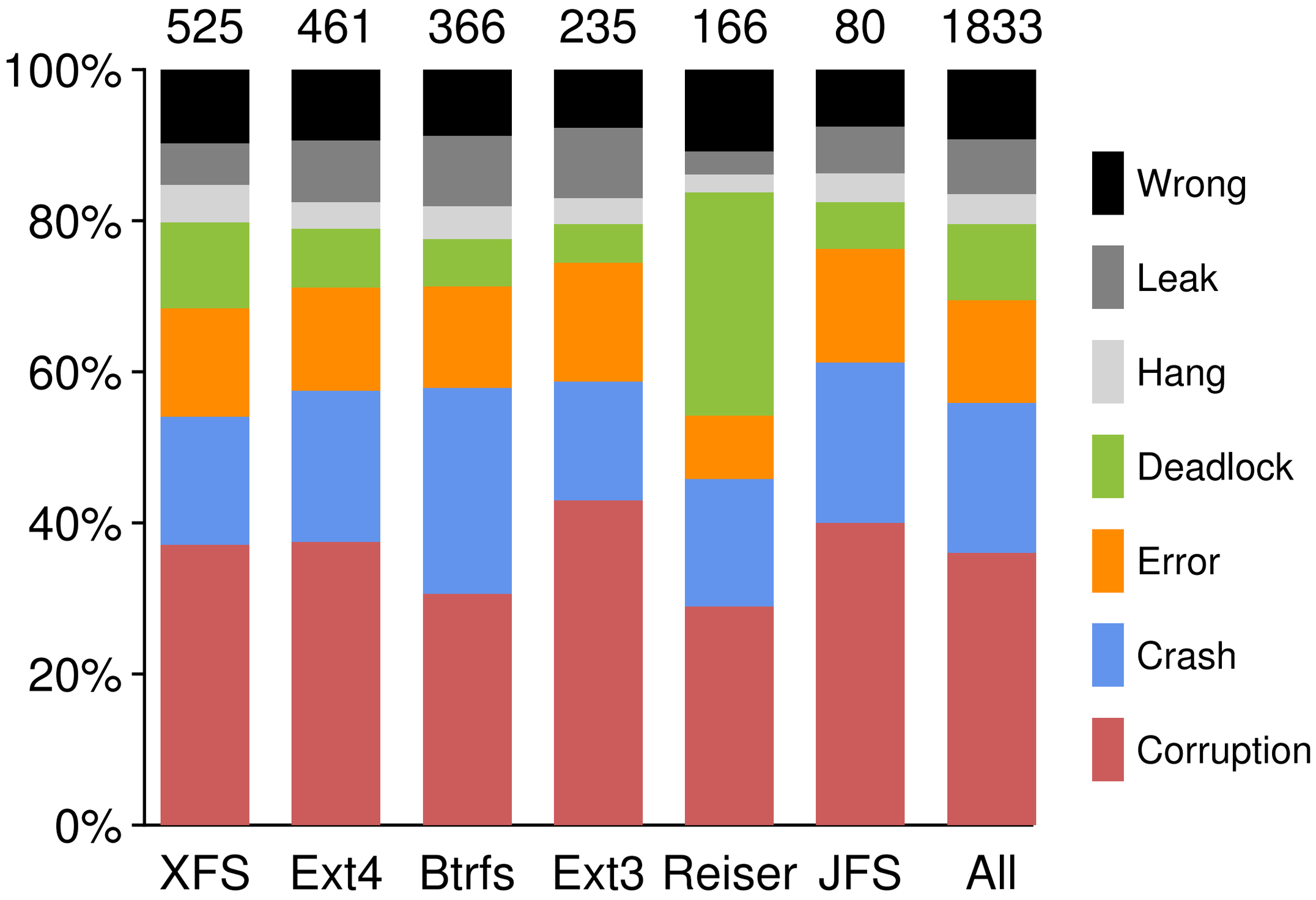
Bug Consequence



Bug Consequence



Bug Consequence



Corruption and Crash are most common

Q5:

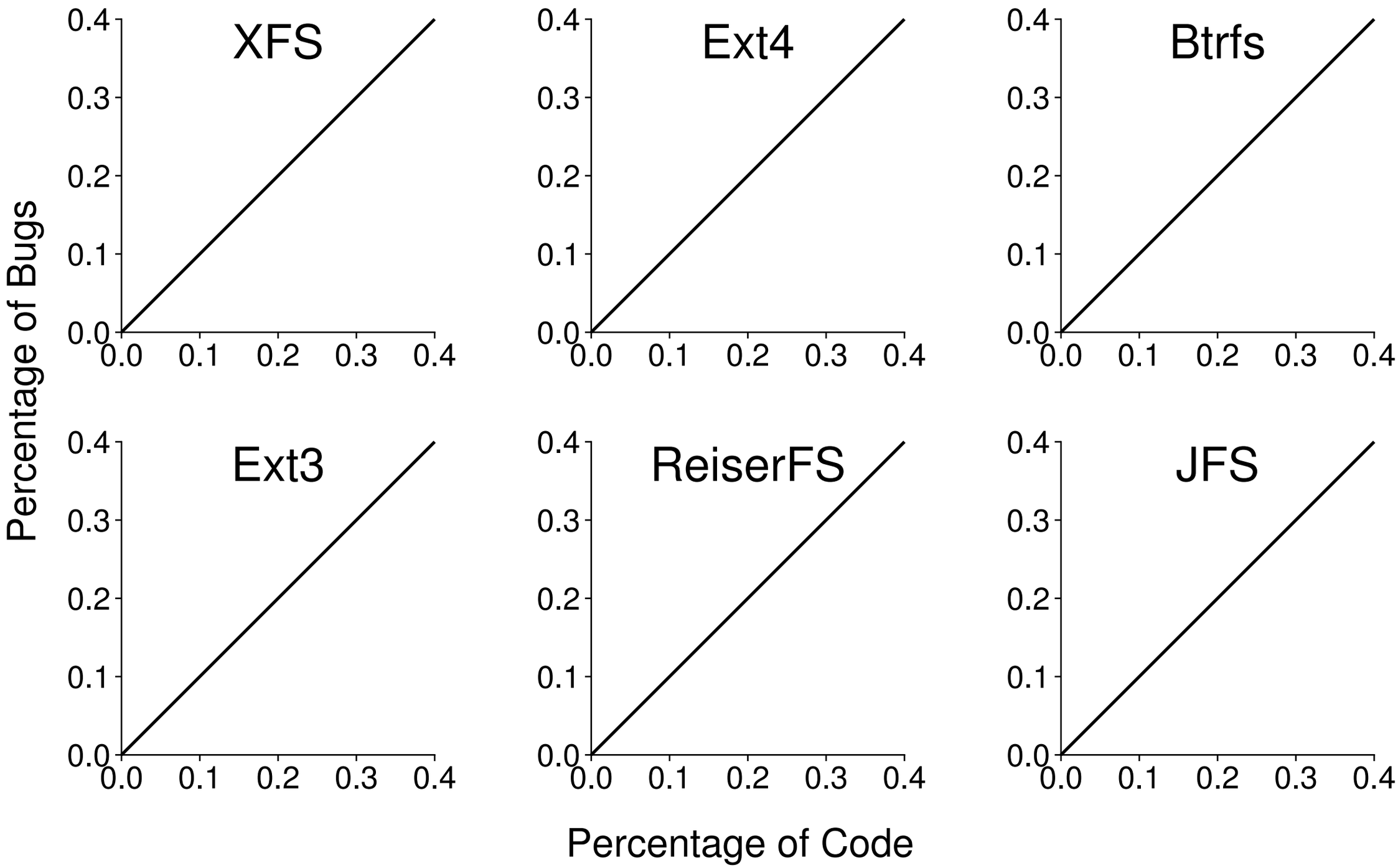
Does each logical component have an equal degree of **complexity** ?

Components

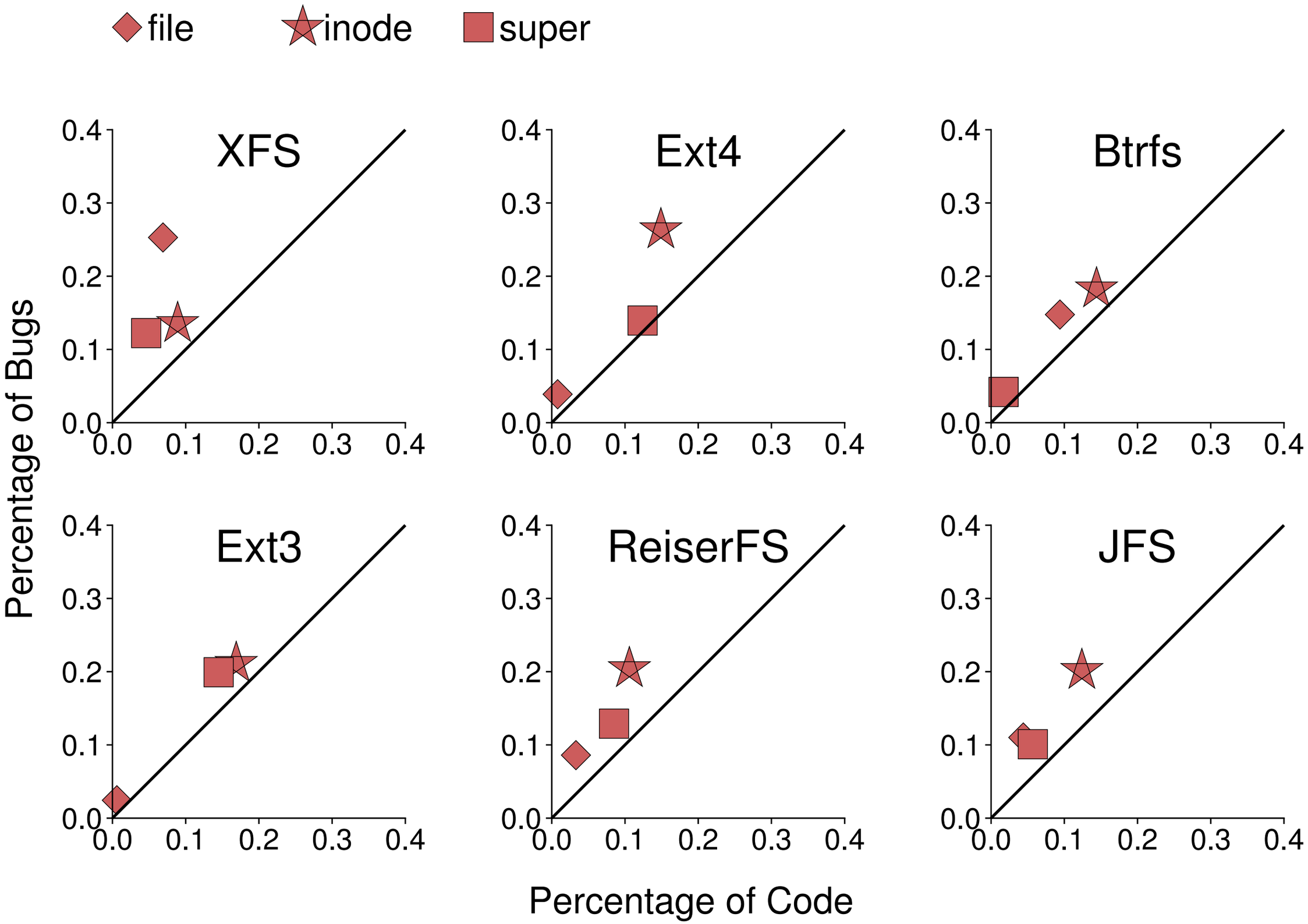
Components

Type	Description
balloc	Data block allocation and deallocation
dir	Directory management
extent	Contiguous physical blocks mapping
file	File read and write operations
inode	Inode-related metadata management
transaction	Journaling or other transactional support
super	Superblock-related metadata management
tree	Generic tree structure procedures
other	Other supporting components (e.g., xattr)

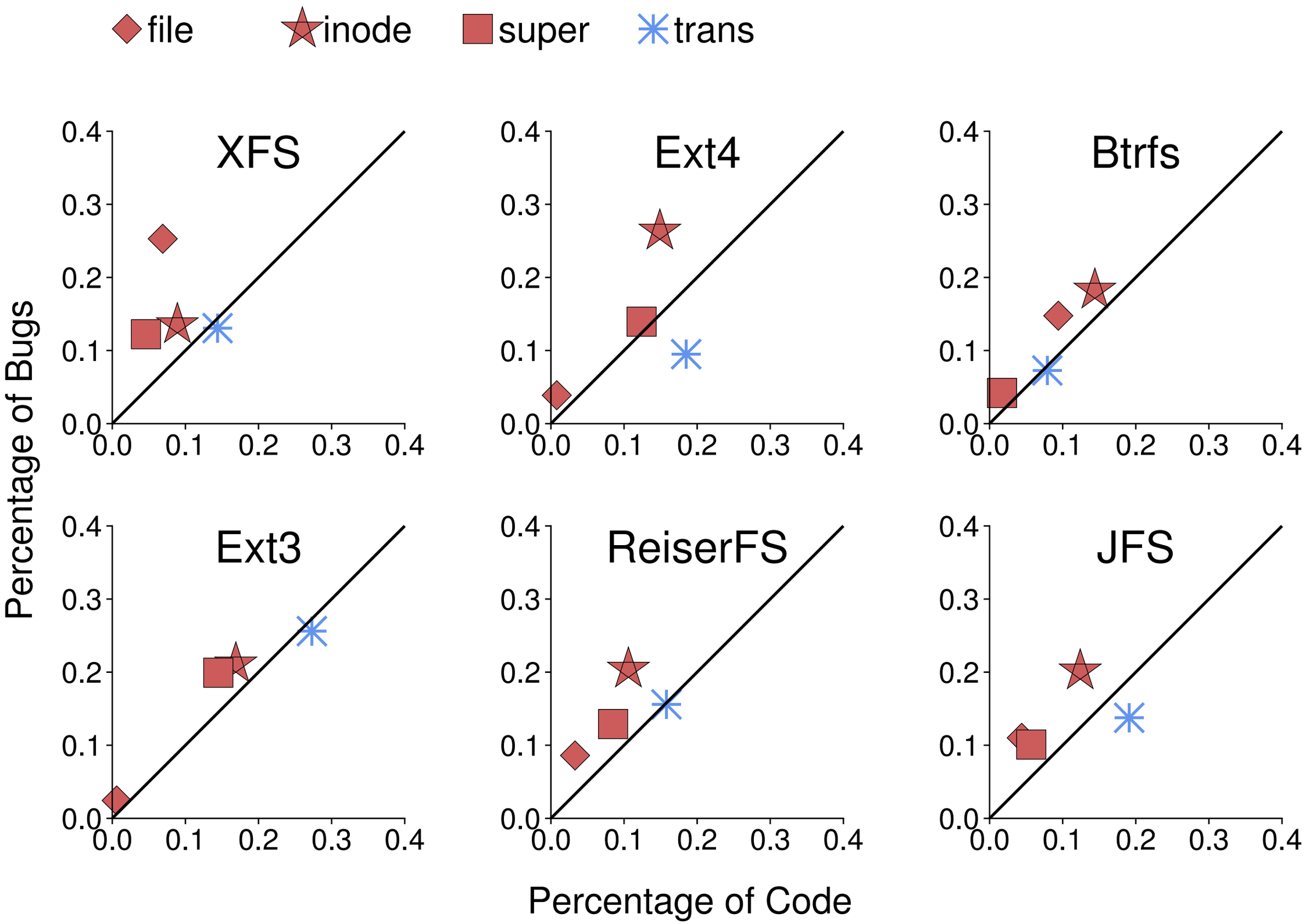
Correlation



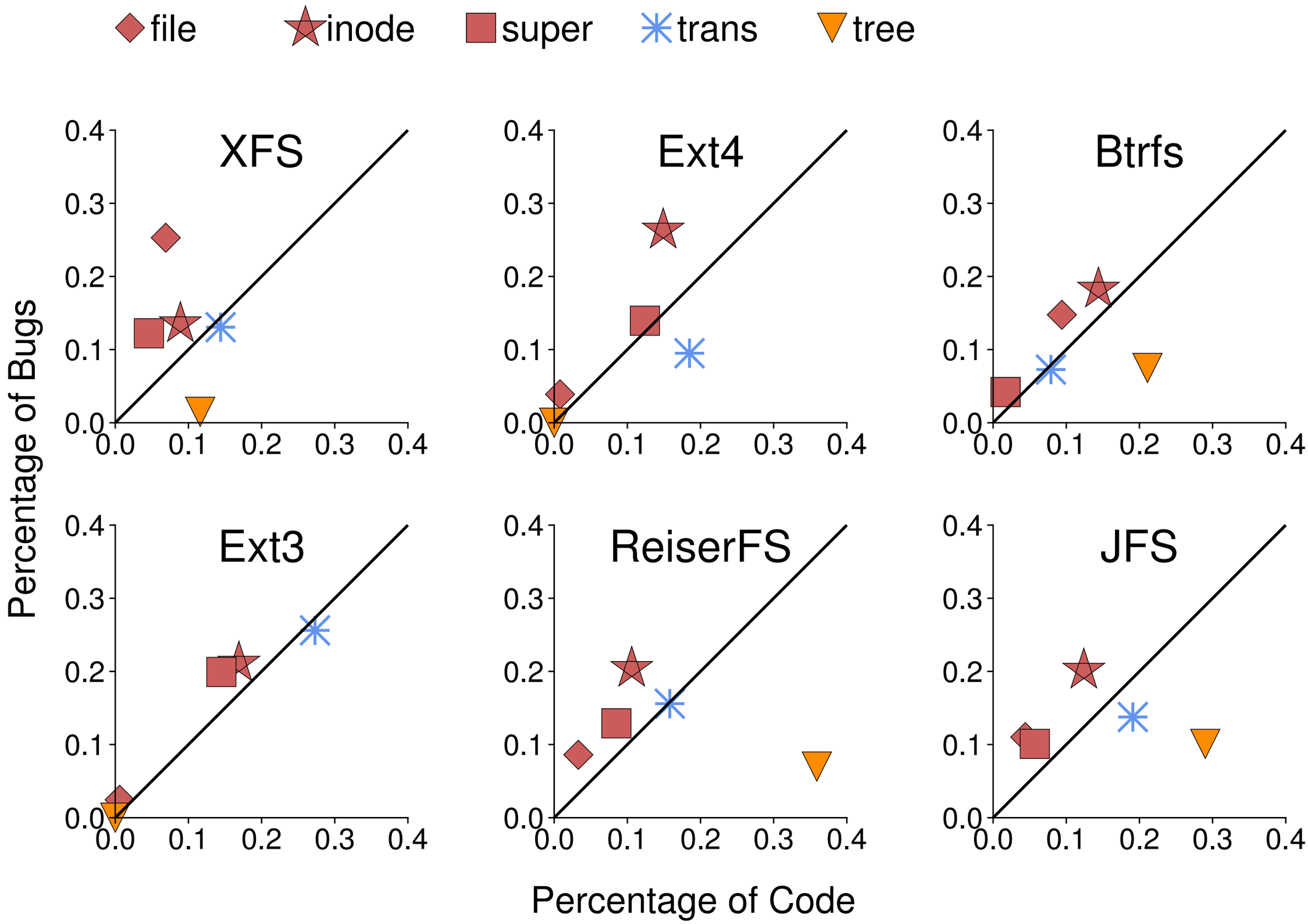
Correlation



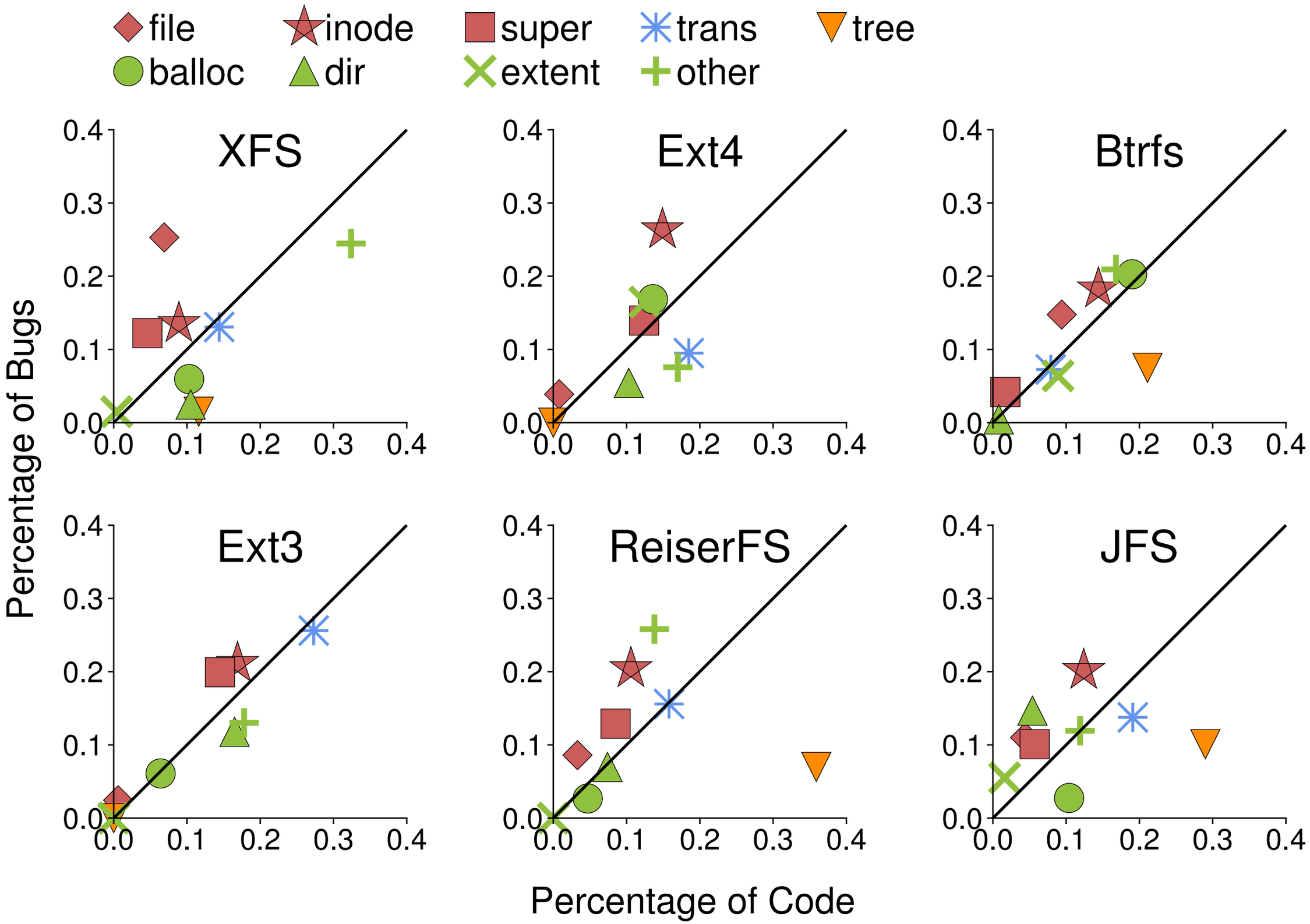
Correlation



Correlation



Correlation



Metadata management

has high bug density

Metadata management has high bug density

Tree related code is not particularly error-prone

Q6:

Do bugs occur at normal paths or failure paths?

Failure Path

Failure Path

A wide range of failures

- resource allocation failure
- I/O operation failure
- silent data corruption
- incorrect system states

Failure Path

A wide range of failures

- resource allocation failure
- I/O operation failure
- silent data corruption
- incorrect system states

Unique code style

- goto statement
- error code propagation

A Semantic Bug on Failure Path

ext4/resize.c, 2.6.25

```
ext4_group_extend( … ) {
        ... ...
1       if (count != ext4_blocks_count(es)){
2           ext4_warning("multiple resizers
                run on filesystem !");


3           err = -EBUSY;
4           goto exit_put;
        }
}
```

A Semantic Bug on Failure Path

```
  ext4/resize.c, 2.6.25

  ext4_group_extend( … ) {
        ... ...
1       if (count != ext4_blocks_count(es)){
2           ext4_warning("multiple resizers
                run on filesystem !");


3           err = -EBUSY;
4           goto exit_put;
        }
  }
```

A Semantic Bug on Failure Path

ext4/resize.c, 2.6.25

```
ext4_group_extend( … ) {
        ... ...
1       if (count != ext4_blocks_count(es)){
2           ext4_warning("multiple resizers
                run on filesystem !");

            ext4_journal_stop(handle);

3           err = -EBUSY;
4           goto exit_put;
        }
}
```

A Memory Bug on Failure Path

```
ext4/inode.c, 2.6.22

ext4_read_inode(struct inode * inode) {
        ... ...
1       if (inode_is_bad) {

2               goto bad_inode;
        }
}
```

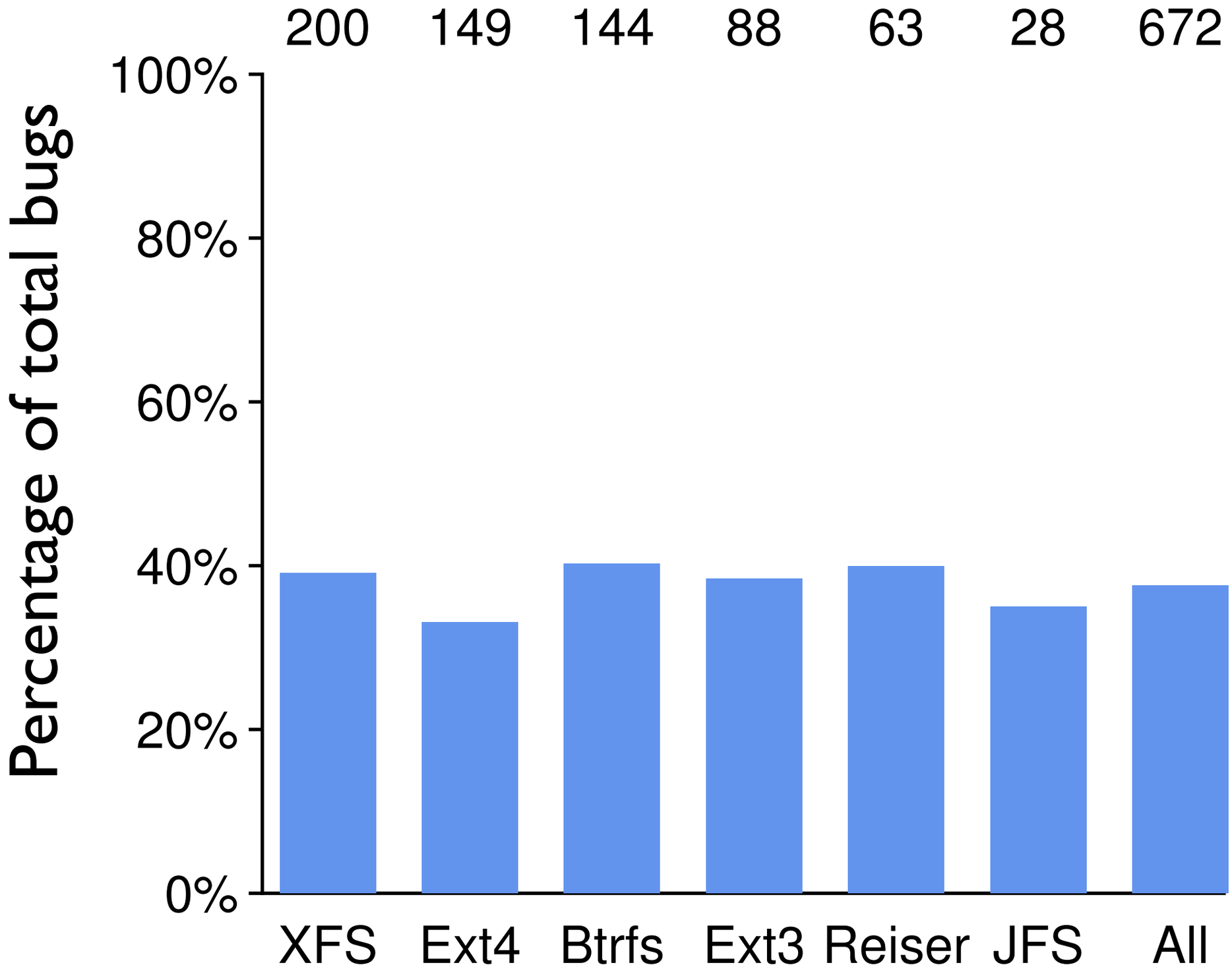
A Memory Bug on Failure Path

```
ext4/inode.c, 2.6.22

ext4_read_inode(struct inode * inode) {
        ... ...
1       if (inode_is_bad) {

2               goto bad_inode;
        }
}
```

A Memory Bug on Failure Path

```
ext4/inode.c, 2.6.22

ext4_read_inode(struct inode * inode) {
        ... ...
1       if (inode is bad) {
            brelse(bh);
2           goto bad_inode;
        }
}
```

Bugs on Failure Paths



38% of bugs are on failure paths

Q7:

What performance techniques are used by file systems ?

Performance

Performance

Type	**Description**
Synchronization	Improve synchronization efficiency
Access Optimization	Apply smarter access strategies
Scheduling	Improve I/O operations scheduling
Scalability	Scale on-disk and in-memory structures
Locality	Overcome sub-optimal block allocation
Other	Other performance improvement (e.g., reducing function stack usage)

Synchronization Example

```
ext4/extents.c, 2.6.31

ext4_fiemap(...){

1     down_write(&EXT4_I(inode)->sem);

2     error = ext4_ext_walk_space(...);

3     up_write(&EXT4_I(inode)->sem);
}
```

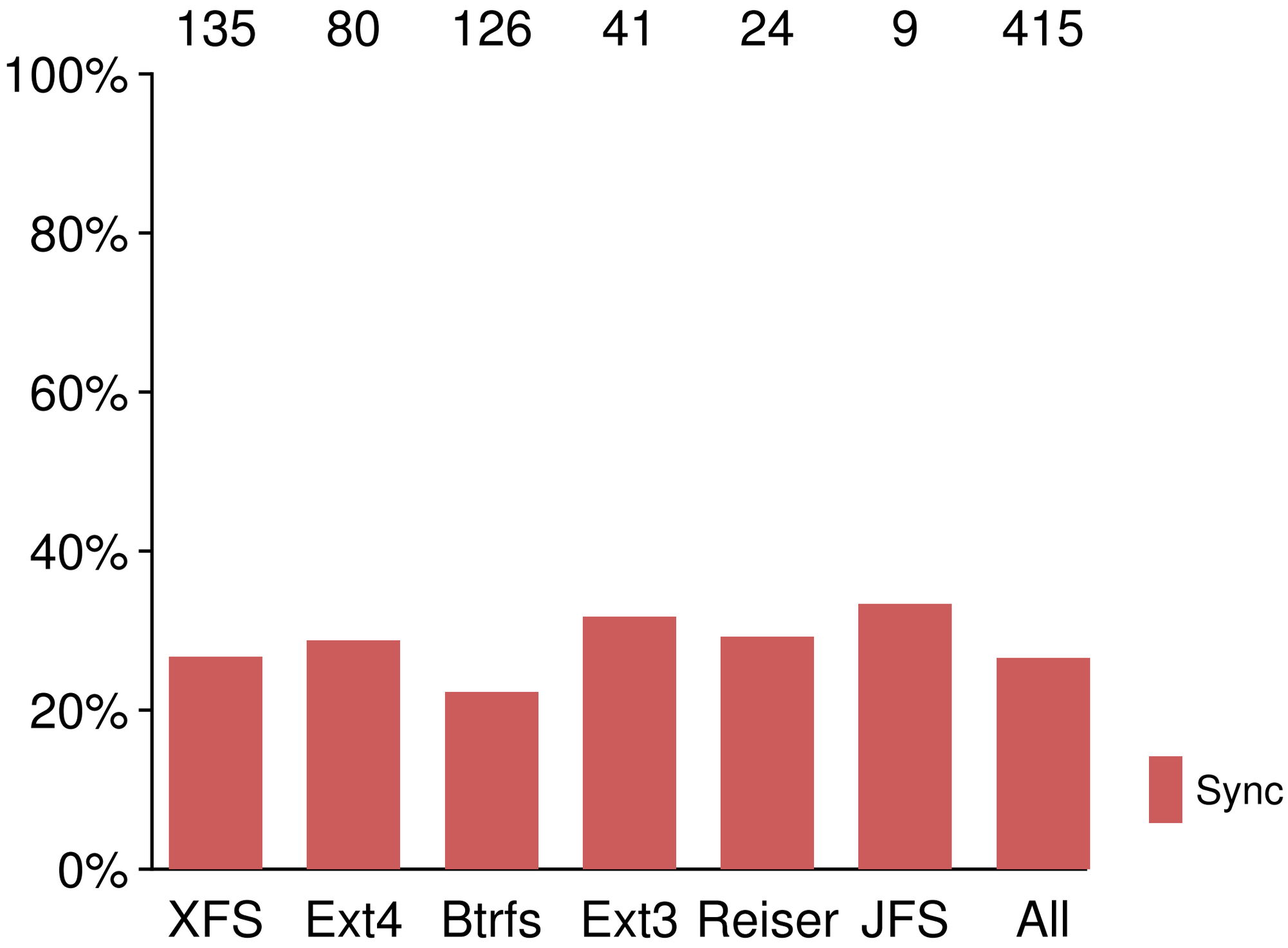
Synchronization Example

```
ext4/extents.c, 2.6.31

ext4_fiemap(...){

1       down_write(&EXT4_I(inode)->sem);

2       error = ext4_ext_walk_space(...);

3       up_write(&EXT4_I(inode)->sem);
}
```

Synchronization Example

```
ext4/extents.c, 2.6.31

ext4_fiemap(...){

1     down_read(&EXT4_I(inode)->sem);

2     error = ext4_ext_walk_space(...);

3     up_read(&EXT4_I(inode)->sem);
}
```

Performance



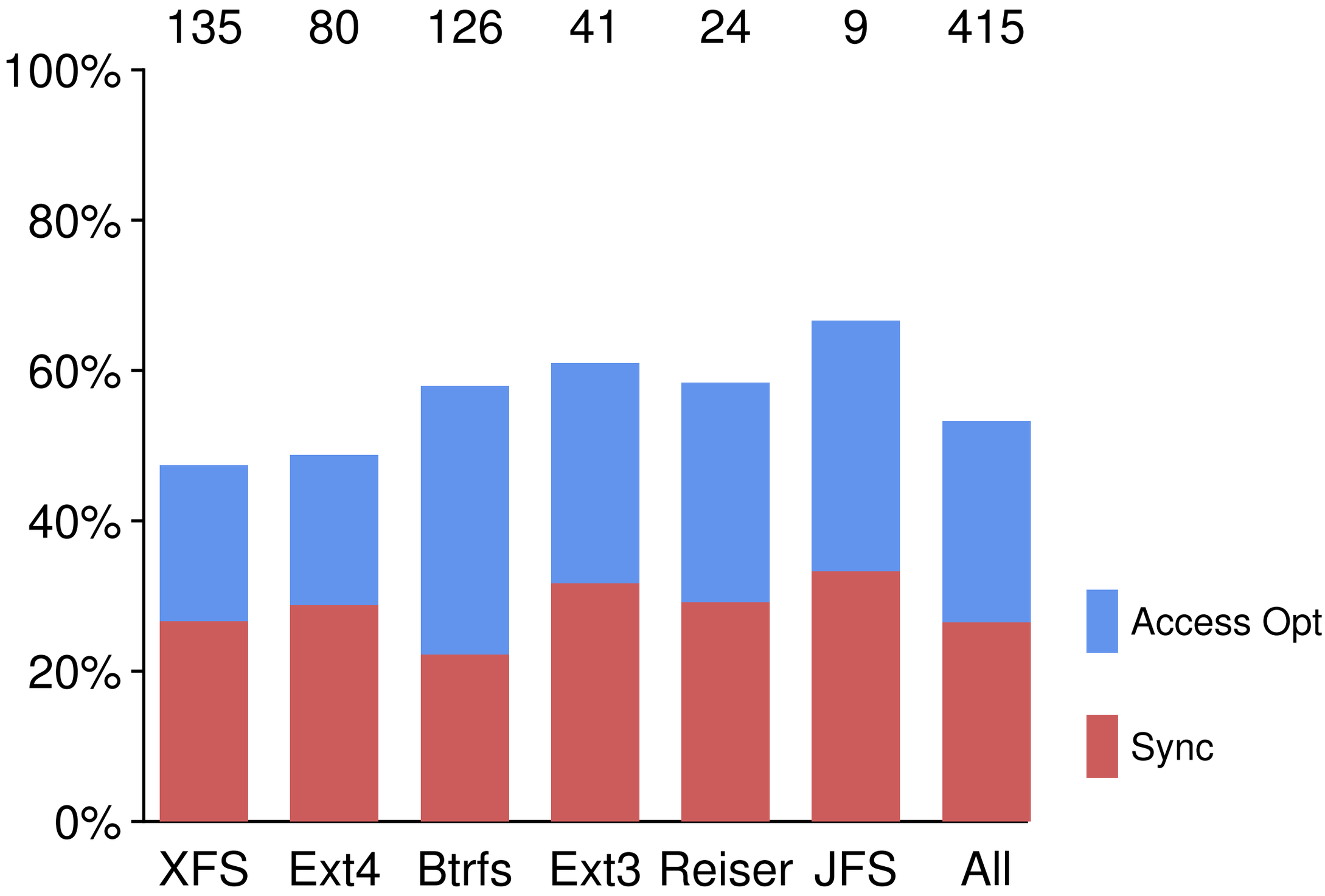
Access Optimization Example

```
btrfs/free-space-cache.c, 2.6.39

btrfs_find_space_cluster( … )

        /* start to search for blocks */
```

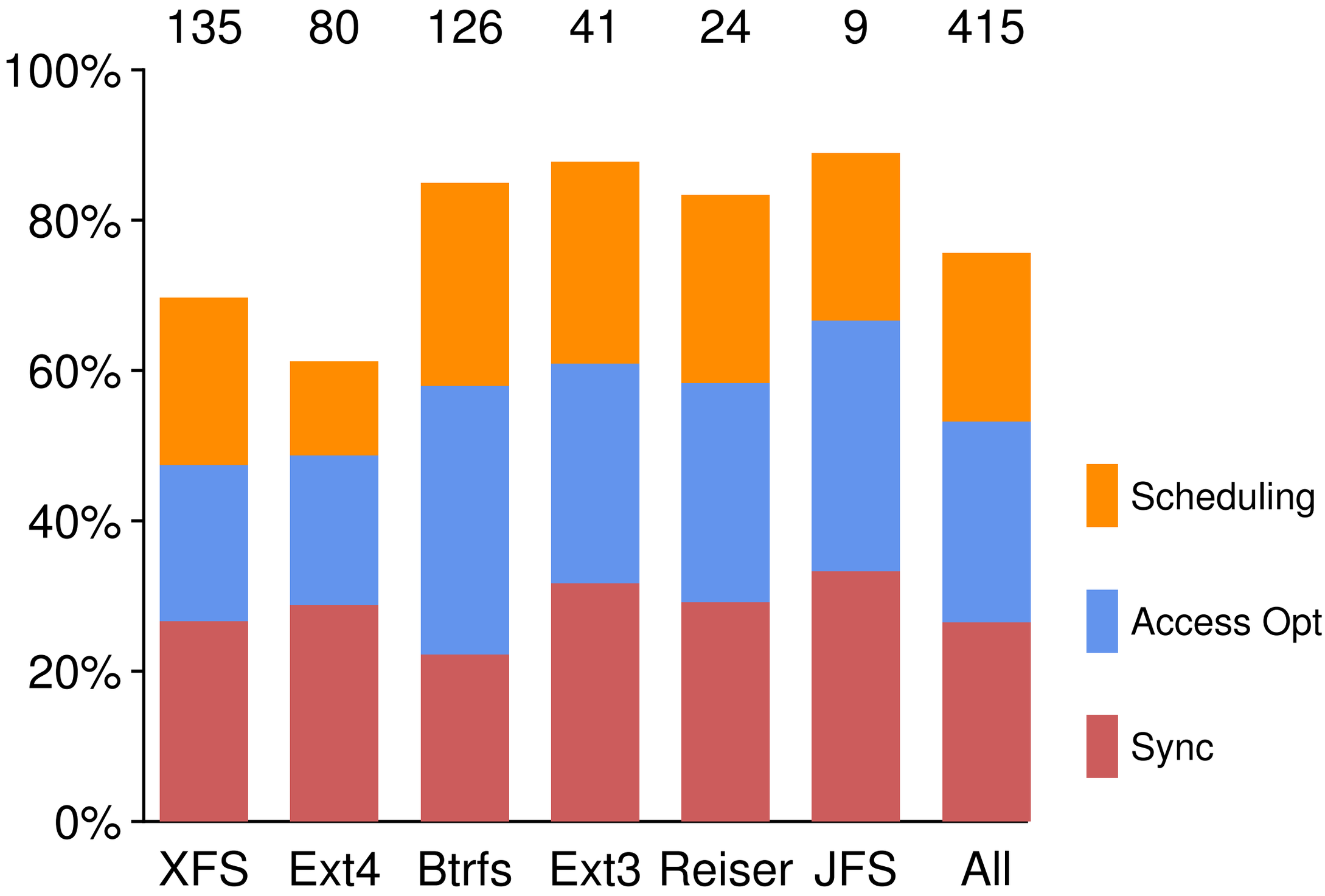
Access Optimization Example

```
btrfs/free-space-cache.c, 2.6.39

btrfs_find_space_cluster( … )

    if (free_space < min_bytes) {
        spin_unlock(&tree_lock);
        return -ENOSPC;
    }

    /* start to search for blocks */
```

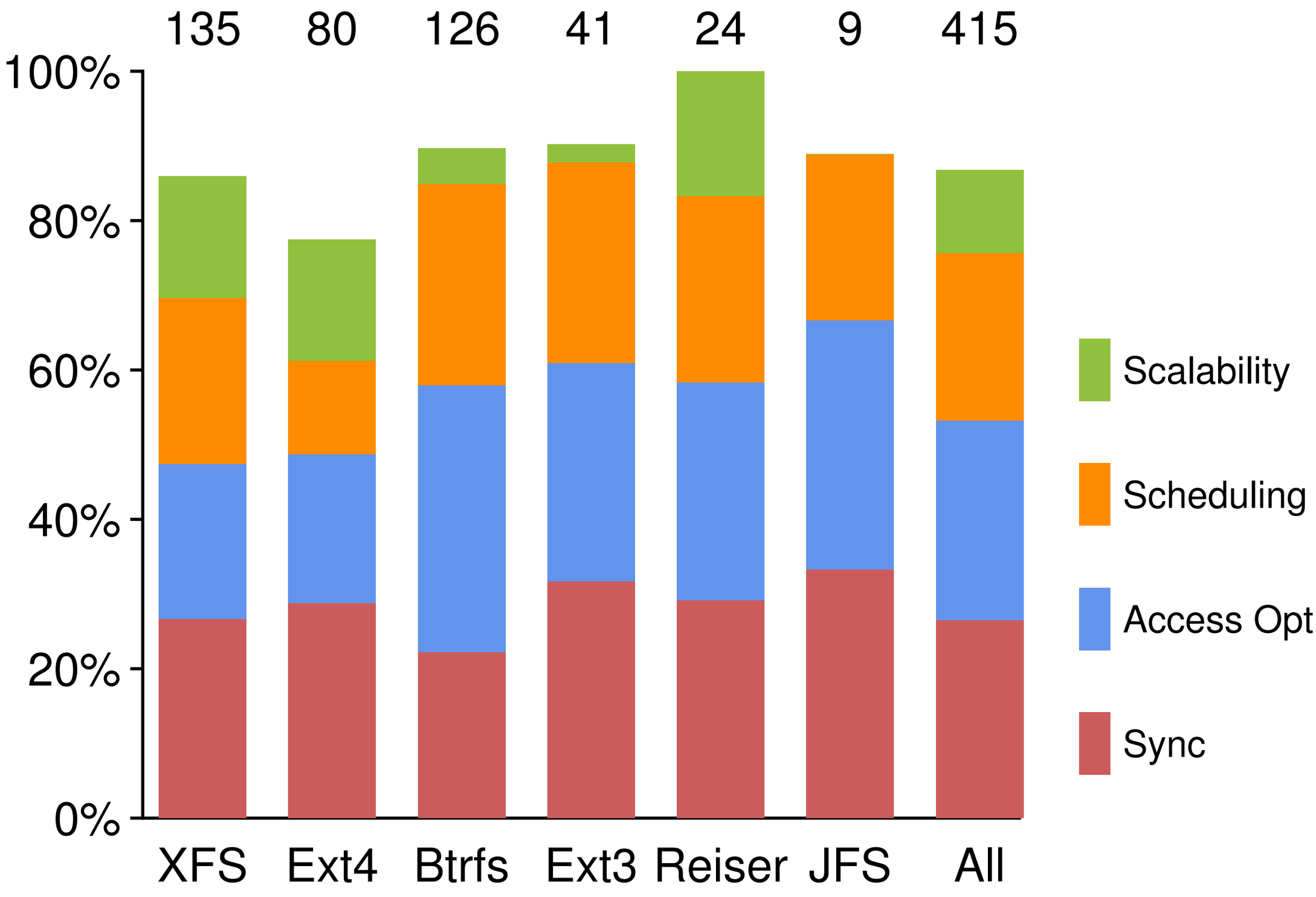
Performance



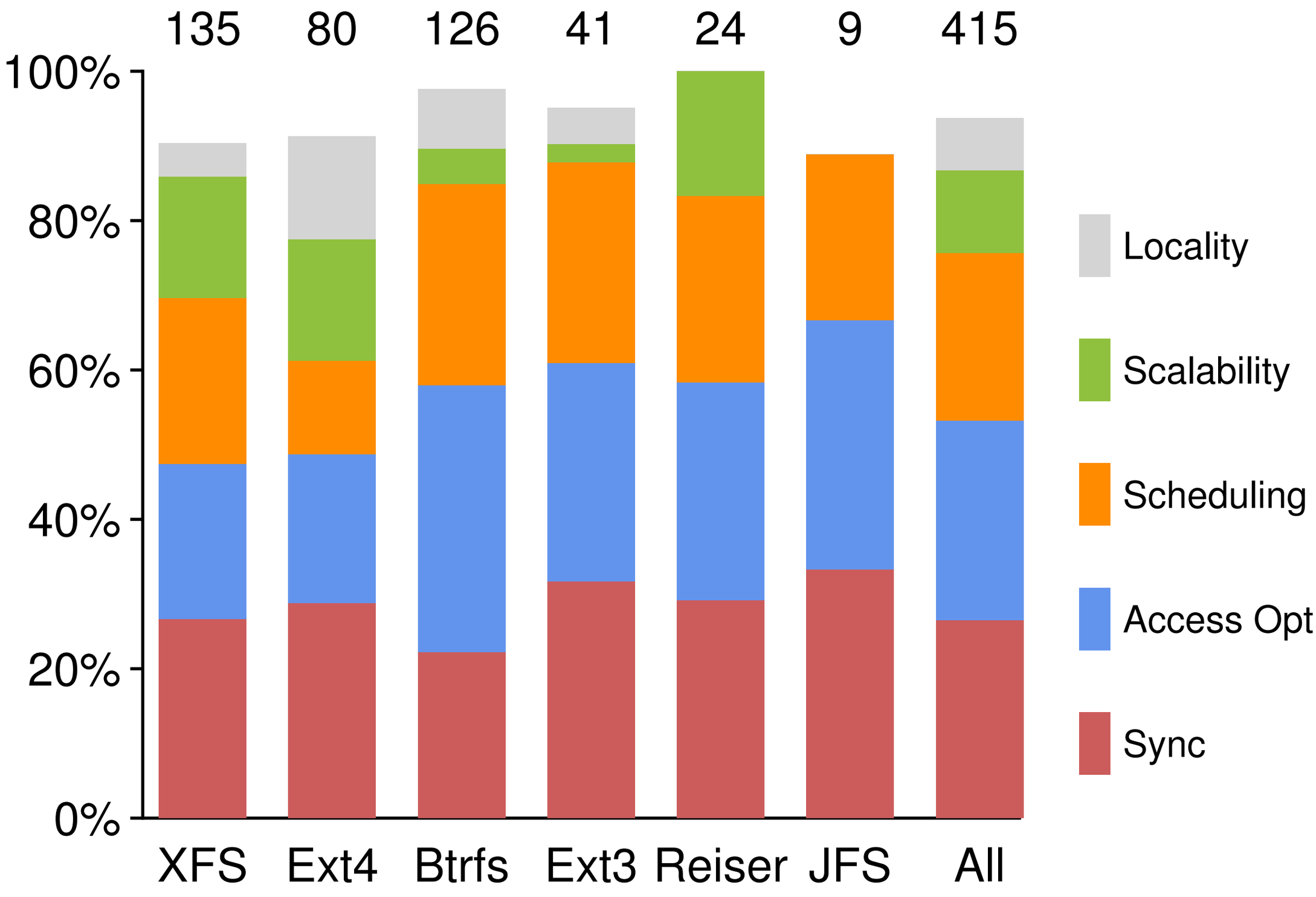
Performance



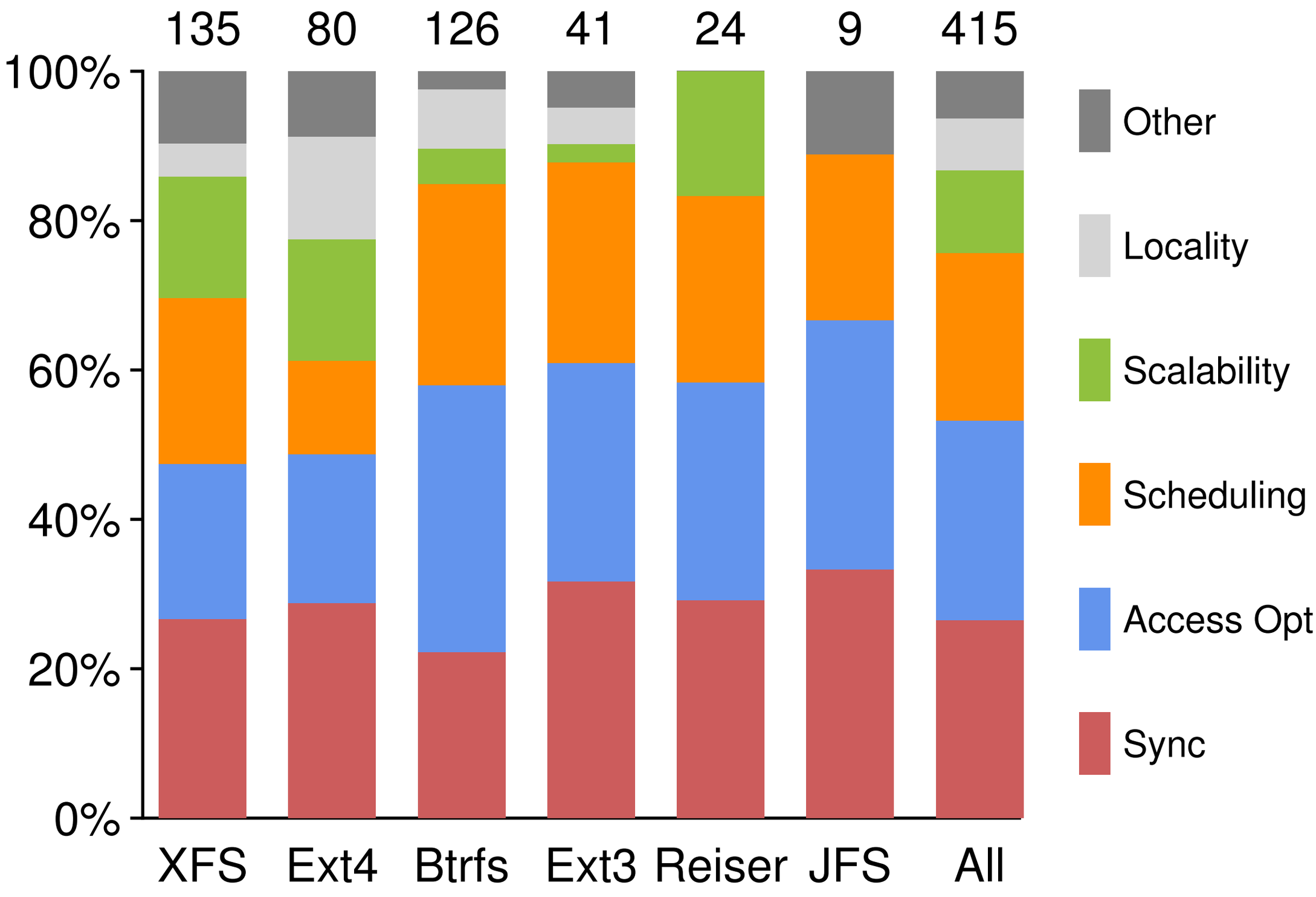
Performance



Performance



Performance



A **wide variety** of **same** performance techniques exist in all file systems

Results Summary

Results Summary

Bugs are prevalent

Semantic bugs dominate

Bugs are constant

Corruption and crash are common

Metadata management is complex

Failure paths are error-prone

Diverse performance techniques

Lessons Learned

Lessons Learned

A large-scale study is feasible and valuable

- time consuming, but still manageable
- similar study for other OS components
- new research opportunities

Lessons Learned

A large-scale study is feasible and valuable

- time consuming, but still manageable
- similar study for other OS components
- new research opportunities

Research should match reality

- new tools for semantic bugs
- more attention for failure paths

Lessons Learned

A large-scale study is feasible and valuable

- time consuming, but still manageable
- similar study for other OS components
- new research opportunities

Research should match reality

- new tools for semantic bugs
- more attention for failure paths

History repeats itself

- similar mistakes
- similar performance and reliability techniques
- learn from history for a better future

Resources

Resources

More information in paper

- detailed bug patterns
- reliability patches
- common patches across file systems

Resources

More information in paper

- detailed bug patterns
- reliability patches
- common patches across file systems

Even more information in dataset

- fix-on-fix
- detailed bug consequences
- tools used

Resources

More information in paper

- detailed bug patterns
- reliability patches
- common patches across file systems

Even more information in dataset

- fix-on-fix
- detailed bug consequences
- tools used

Our dataset is released

- http://research.cs.wisc.edu/wind/Traces/fs-patch/