Tolerating File-System Mistakes with EnvyFS

Lakshmi N. Bairavasundaram
NetApp, Inc.

Swaminathan Sundararaman
Andrea C. Arpaci-Dusseau
Remzi H. Arpaci-Dusseau
University of Wisconsin Madison

File Systems in Today's World

- Modern file systems are complex
  - Tens of thousands of lines of code (e.g., XFS 45K LOC)
- Storage stack is also getting deeper
  - Hypervisor, network, logical volume manager
- Need to handle a gamut of failures
  - Memory allocation, disk faults, bit flips, system crashes
- Preserve integrity of its meta-data and user data

File System Bugs

- Bug reports for Linux 2.6 series from Bugzilla
  - ext3: 64, JFS: 17, ReiserFS: 38
  - Some are FS corruption causing permanent data loss

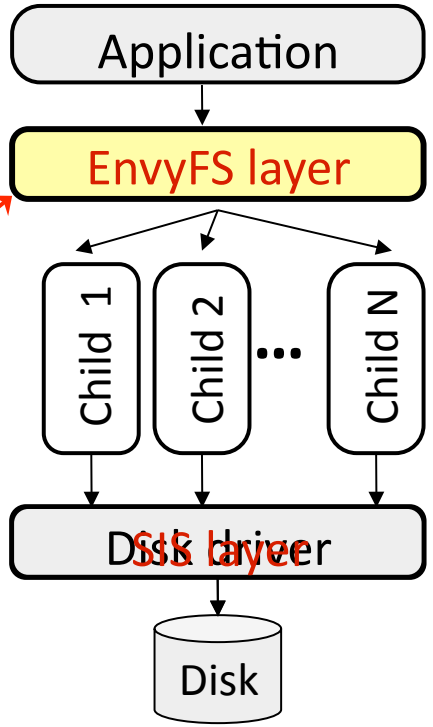
- FS bugs broadly classified into two categories
  - *"fail-stop"*: System immediately crashes
    - Solutions: Nooks [*Swift 04*], CuriOS [*David08*]
  - "*fail-silent*": Accidentally corrupt on-disk state
    - Many such bugs uncovered [*Prabhakaran05, Gunawi08, Yang04, Yang06b*]

Bugs are inevitable in file systems

Challenge: how to cope with them?

N-Version File Systems

- Based on N-version programming [*Avizienis77*]
  - NFS servers [*Rodrigues01*], databases [*Vandiver07*], security [*Cox06*]
- EnvyFS: Simple software layer
  - Store data in *N* child file systems
  - Operations performed on all children
- **Rely on a simple software layer**
- Challenge: reducing overheads while retaining reliability
  - SubSIST: Novel Single Instance Store



Results

- Robustness
  - Traditional file systems handle few corruptions (< 4%)
  - EnvyFS$_3$ tolerates 98.9% of single file system mistakes
- Performance
  - Desktop workloads: EnvyFS$_3$ has comparable performance
  - I/O intensive workloads:
    - Normal mode: EnvyFS$_3$ + SubSIST acceptable performance
    - Under memory pressure: EnvyFS$_3$ + SubSIST large overheads
- Potential as a debugging tool for FS developers
  - Pinpoint the source of "*fail-silent*" bug in ext3

Outline

- Introduction
- **Building reliable file systems**
- Reducing overheads with SubSIST
- Evaluation
- Conclusion

N-Version Systems

Development process:

1. Producing the specification of software
2. Implementing N versions of the software
3. Creating N-version layer
   - Executes different versions
   - Determines the consensus result

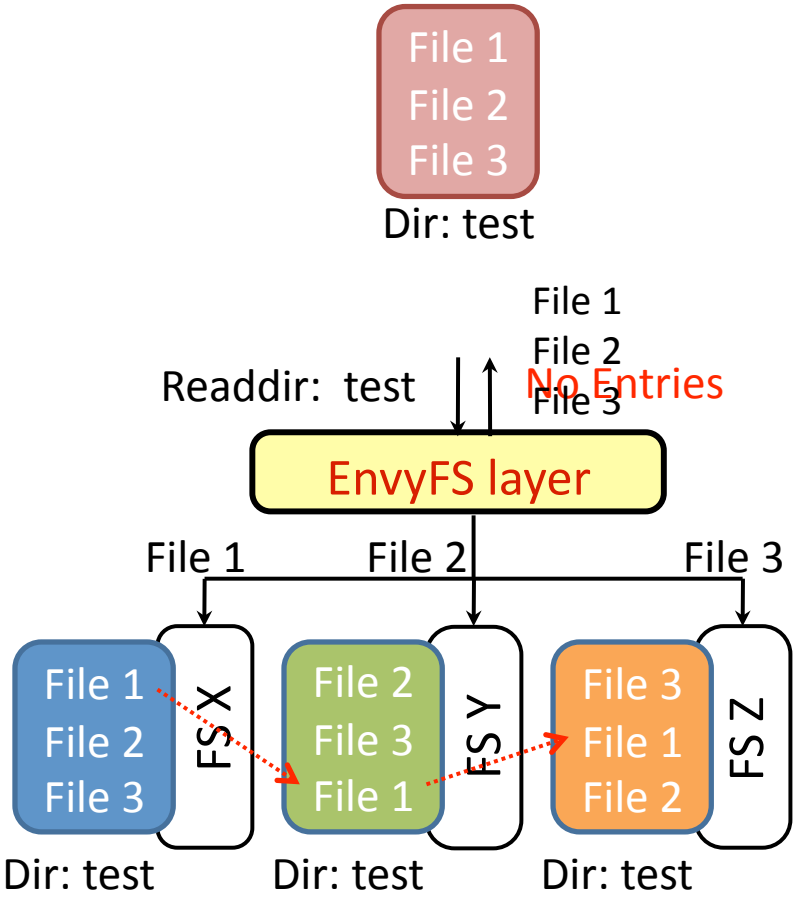
1. Producing Specification

- Our own specification ?
  - Impractical: Requires wide scale changes to file systems
  - Specifications take years to get accepted
- Can we leverage existing specification ?
  - Yes, can leverage VFS, but there are some issues
- VFS not precise for N-versioning purpose
  - Needs to handle cases where specification is not precise
  - e.g., Ordering directory entries, inode number allocation

Imprecise VFS Specification

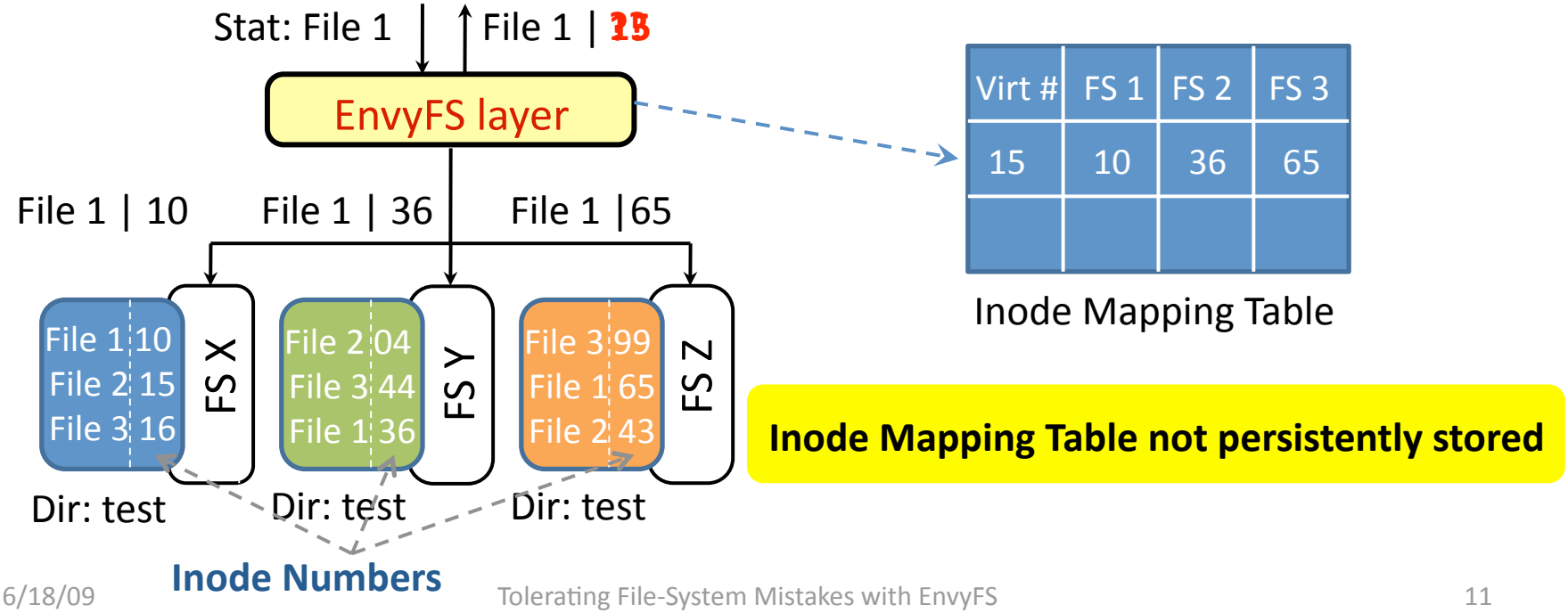
Ordering directory entries

- Issue:
  - No specified return order
  - Can't blindly compare entries
- Solution:
  - Read all entries from a directory (dir: test in our case) from all FSes
  - Match entries from FSes
  - Return majority results



Imprecise VFS Specification (cont)

- Inode number allocation
  - Inode numbers returned through system calls
  - Each child file system issues different inode numbers
  - Possible solution: Force file systems to use same algorithm?
  - Our solution: Issue inode numbers at EnvyFS layer

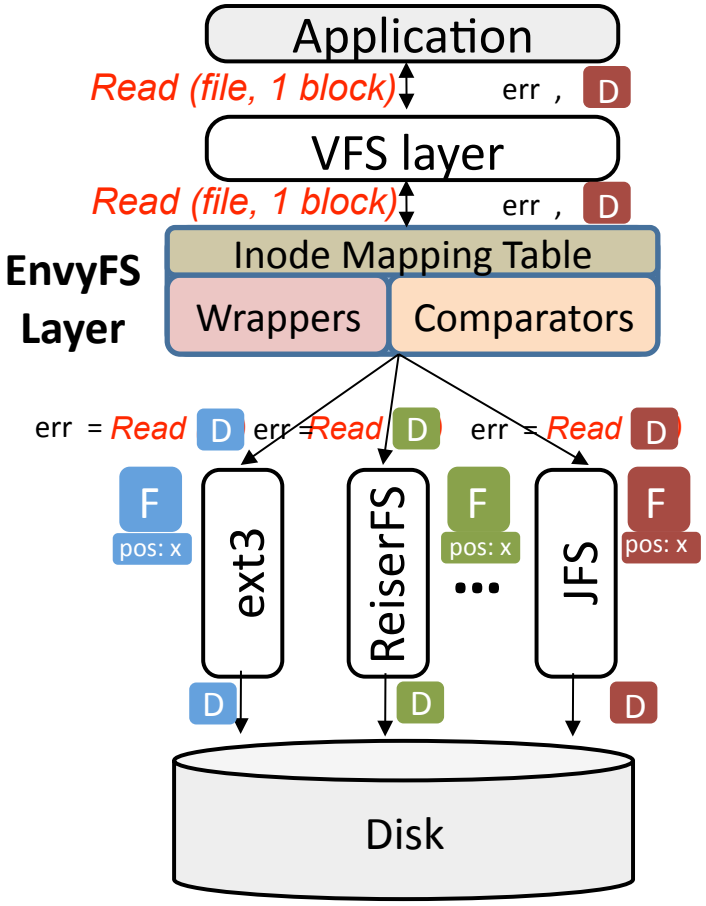


2. Implementing N versions of FS

- Painful process
  - High cost of development, long time delays
- Lucky! Hard work already done for us
  - 30 different disk based file systems in Linux 2.6
- Which file systems to use?
  - ext3, JFS, ReiserFS in a three-version FS
  - Others should work without modifications

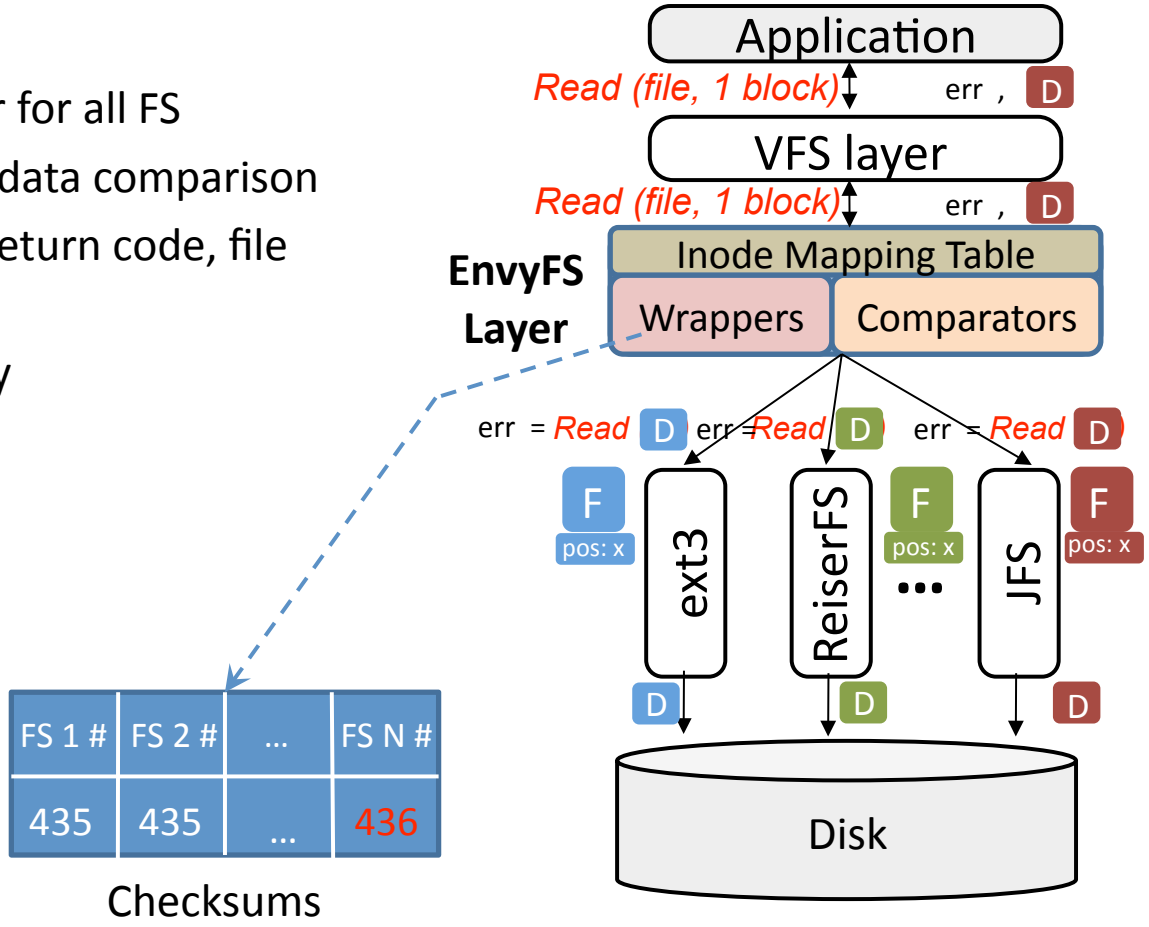
3. Creating N-Version Layer

- N-Version layer (*EnvyFS*)
  - Inserted beneath VFS
  - Simple design to avoid bugs
- Example: Reading a file
  - Allocate N data buffers
  - Read data block from the disk
  - Compare: data, return code, file position
  - Return: data, return code
- Issues:
  - Allocate memory for each read operation
  - Extra copy from allocated buffer to application
  - Comparison overheads



Reading a File in EnvyFS

- Solution:
  - Same application buffer for all FS
  - TCP-like checksums for data comparison
  - Compare: checksums, return code, file position
  - Read data until majority

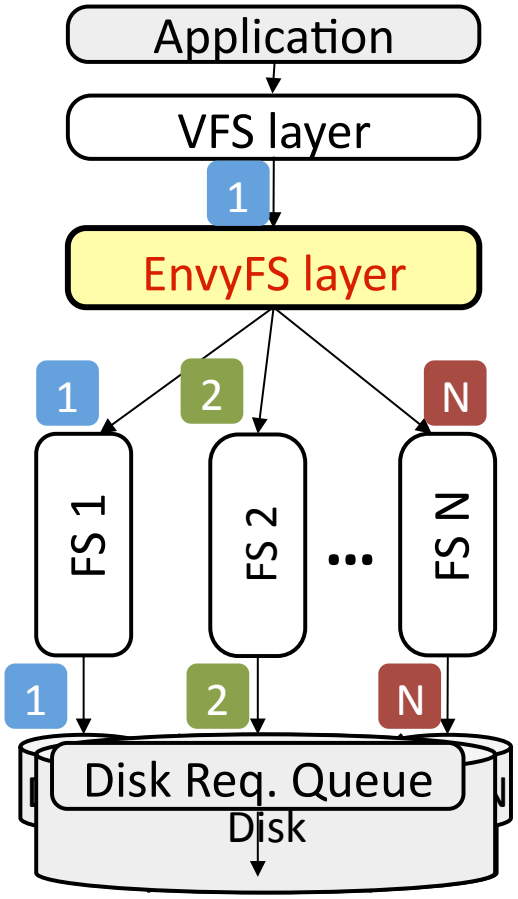


Outline

- Introduction
- **Building reliable file systems**
- **Reducing overheads with SubSIST**
- Evaluation
- Conclusion

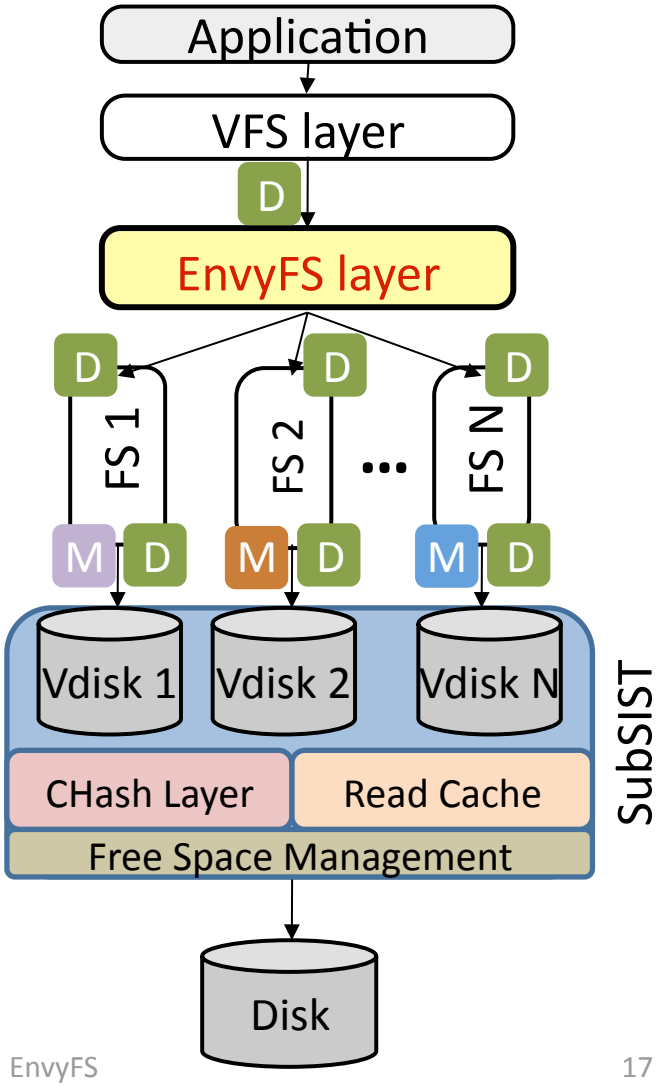
Case for Single Instance Storage (SIS)

- Ideal: One disk per FS
- Practical: One disk for all FS
- Overheads
  - Effective storage space: 1/N
  - N times more I/O (Read/write)
- **Challenge:** Maintain diversity while minimizing overheads



SubSIST: Single Instance Store

- Variant of an Single Instance Store
  - Selectively merges data blocks
- Block addressable SIS
  - Exports virtual disks to FSes
  - Manages mapping, free space info.
  - Not persistently stored on disk
- EnvyFS writes through N file systems
  - N data blocks **merged** to 1 data block
  - Content hashes **not stored persistently**
  - Meta-data blocks **not merged**
  - Inter FS blocks and **not intra FS**

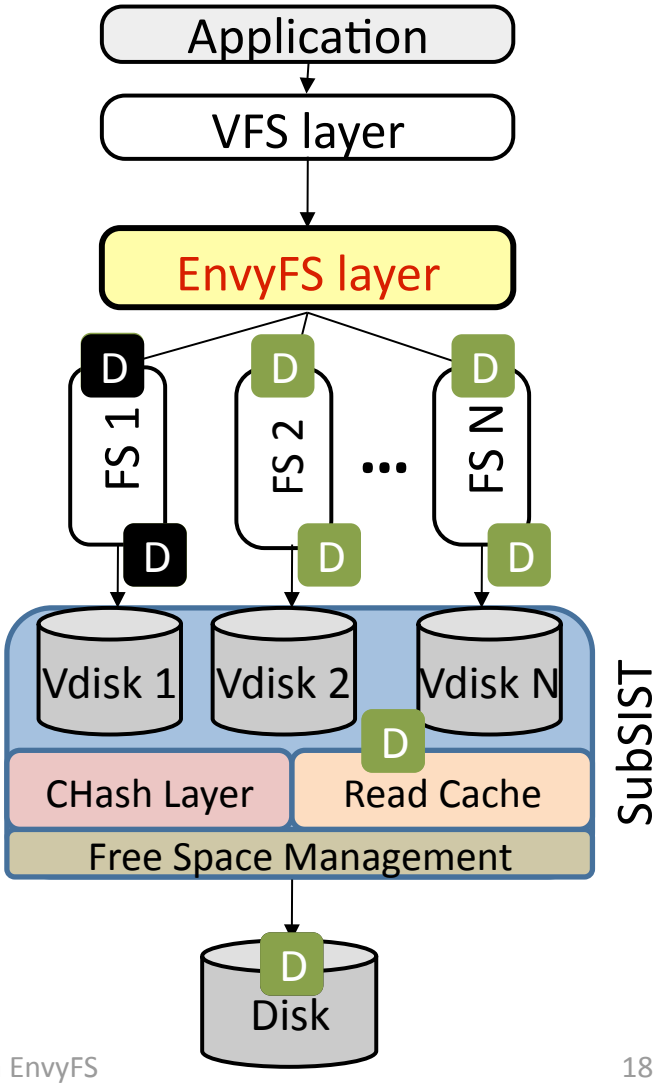


Handling Data Block Corruptions?

✓ Corruption to data in a single FS
- Due to bugs, bit flips, storage stack
- Corrupt data blocks not merged
- All other N-1 data blocks merged
- Corrupt data block fixed at next read

× Corruption to data block inside disk

• Single copy of data
- Different code paths
- Different on-disk structures



Outline

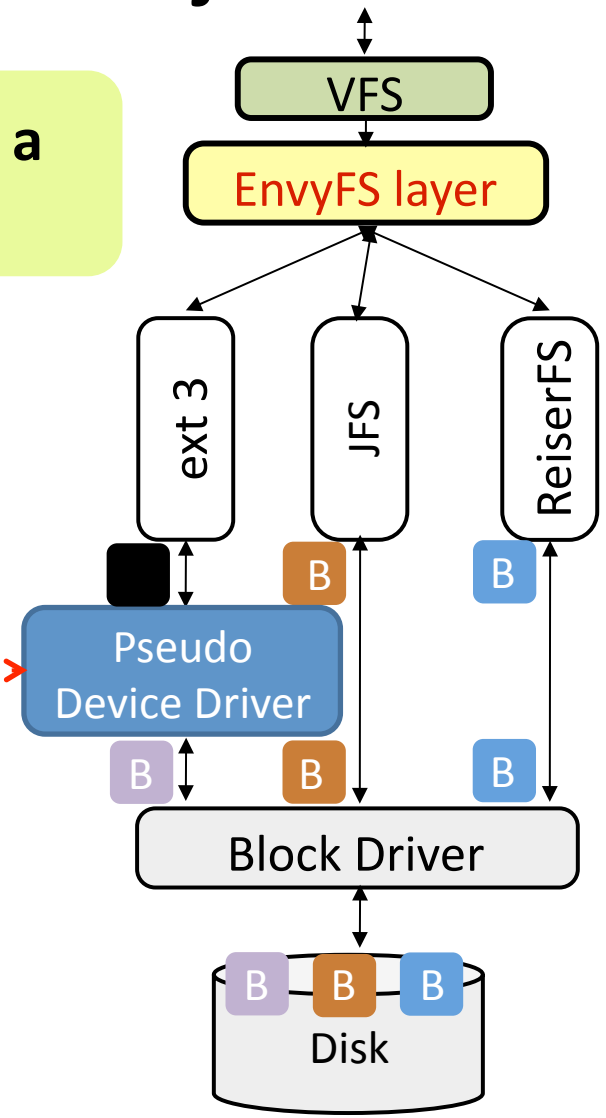
- Introduction
- Building reliable file systems
- Reducing overheads with SubSIST
- **Evaluation**
  - **Reliability**
  - Performance
- Conclusion

Reliability Evaluation: Fault Injection

Robustness of EnvyFS in recovering from a child file system's mistake?

- Corruption: bugs in FS / storage stack
- Types of disk blocks
  - superblock, inode, block bitmap, file data, …
- Perform different file ops
  - mount, stat, creat, unlink, read, …
- Report user visible results
- **All results are applicable with SubSIST except corruption to data blocks**

Type-aware fault injection [*Prabhakaran05*]



ext3

E

path traversal | SET-1 (stat, ...) | SET-2 (chmod) | read | readlink | getdirentries | creat | link | mkdir | rename | symlink | write | truncate | rmdir | unlink | mount | SET-3 (fsync) | umount

INODE

DIR

BMAP

IMAP

INDIRECT

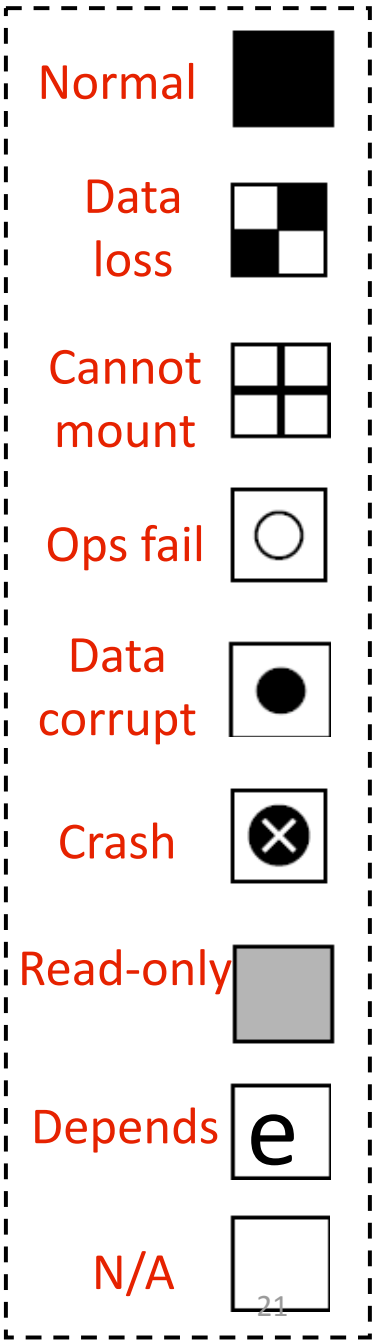
DATA

SUPER

JSUPER

GDESC

Result Matrix



ext3

E

path traversal | SET-1 (stat, ...) | SET-2 (chmod) | read | readlink | getdirentries | creat | link | mkdir | rename | symlink | write | truncate | rmdir | unlink | mount | SET-3 (fsync) | umount

INODE

DIR

BMAP

IMAP

INDIRECT

DATA

SUPER

JSUPER

GDESC

Ext3 stores many superblock copies;

but, does not handle superblock corruption

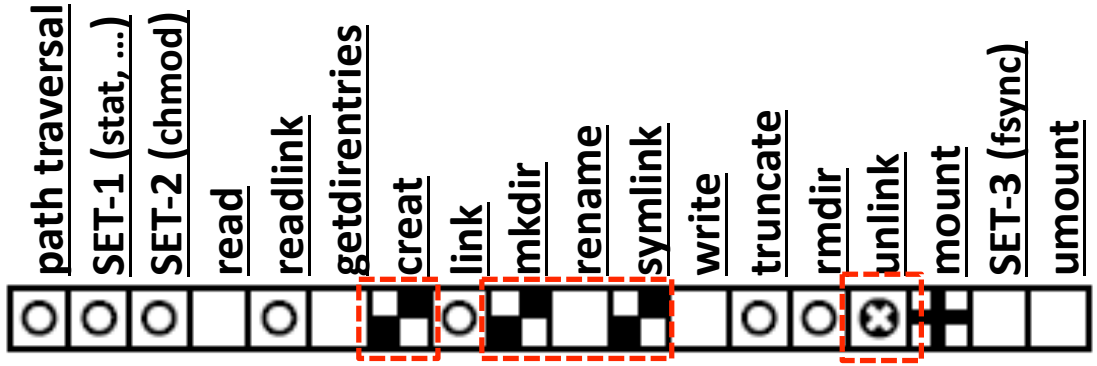
Data loss

Cannot mount

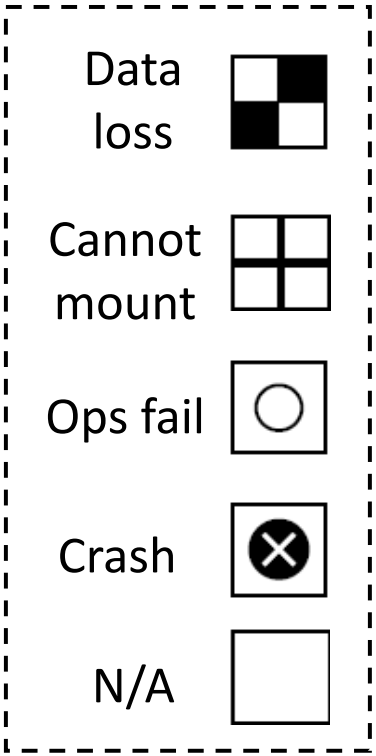
N/A

ext3

E

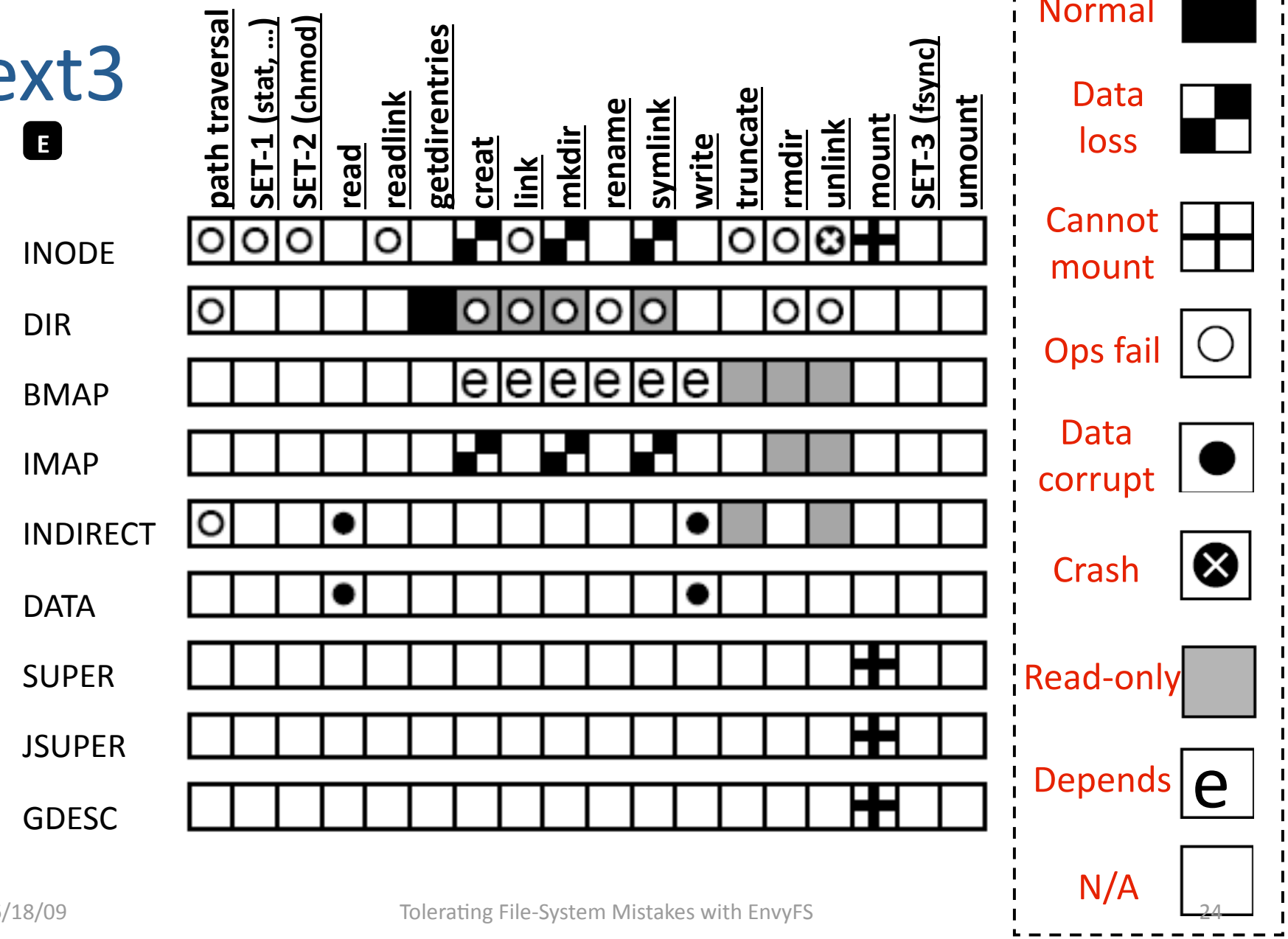


- In addition to operations failing, inode corruption leads to data loss
- Unlink: system crash during unmount

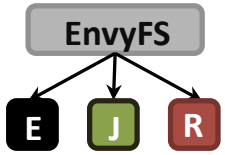


ext3

E

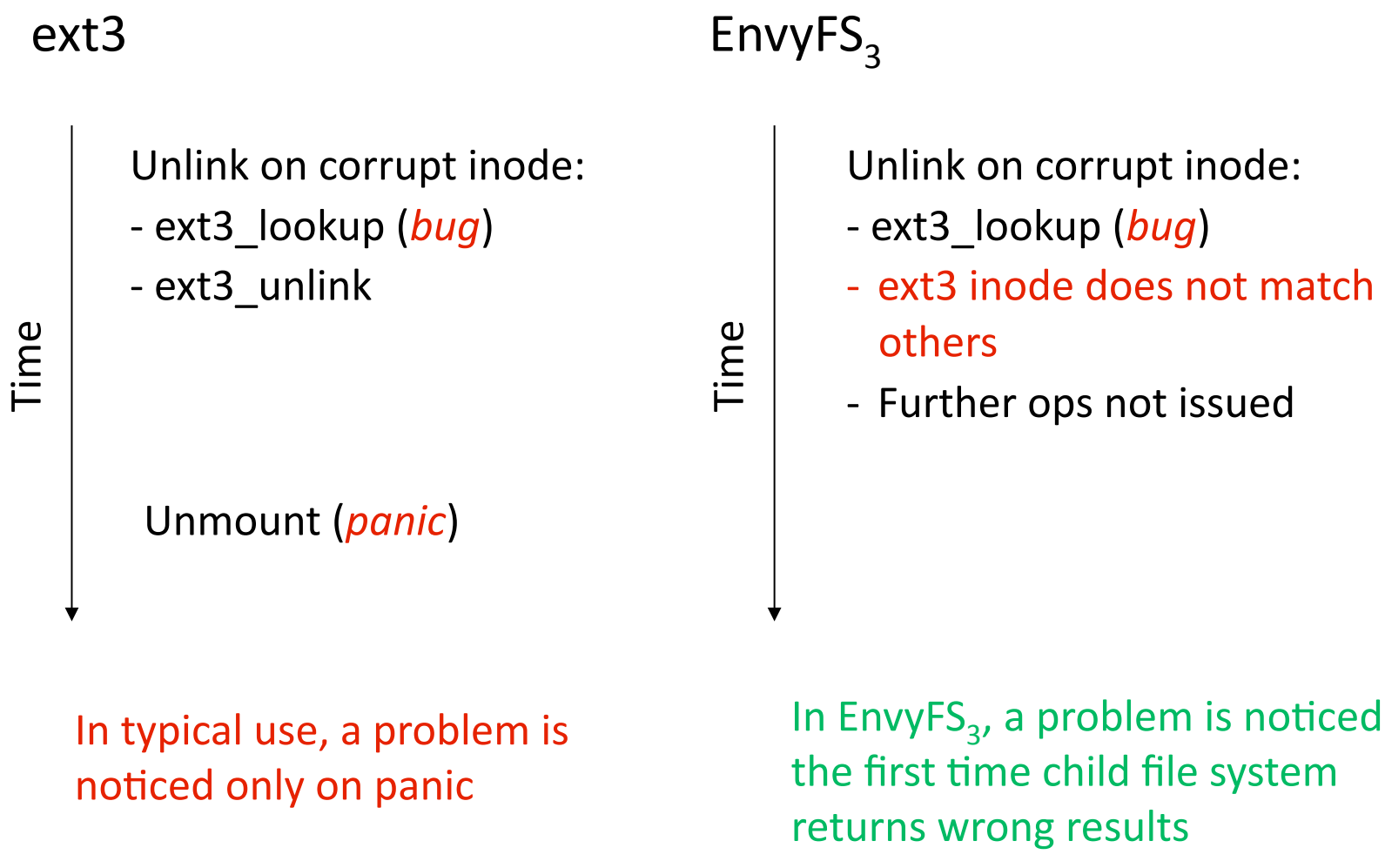

path traversal
SET-1 (stat, ...)
SET-2 (chmod)
read
readlink
getdirentries
creat
link
mkdir
rename
symlink
write
truncate
rmdir
unlink
mount
SET-3 (fsync)
umount
INODE
DIR
BMAP
IMAP
INDIRECT
DATA
SUPER
JSUPER
GDESC
Normal
Data loss
Cannot mount
Ops fail
Data corrupt
Crash
Read-only
Depends
e
N/A

EnvyFS$_3$



EnvyFS$_3$ works in every scenario

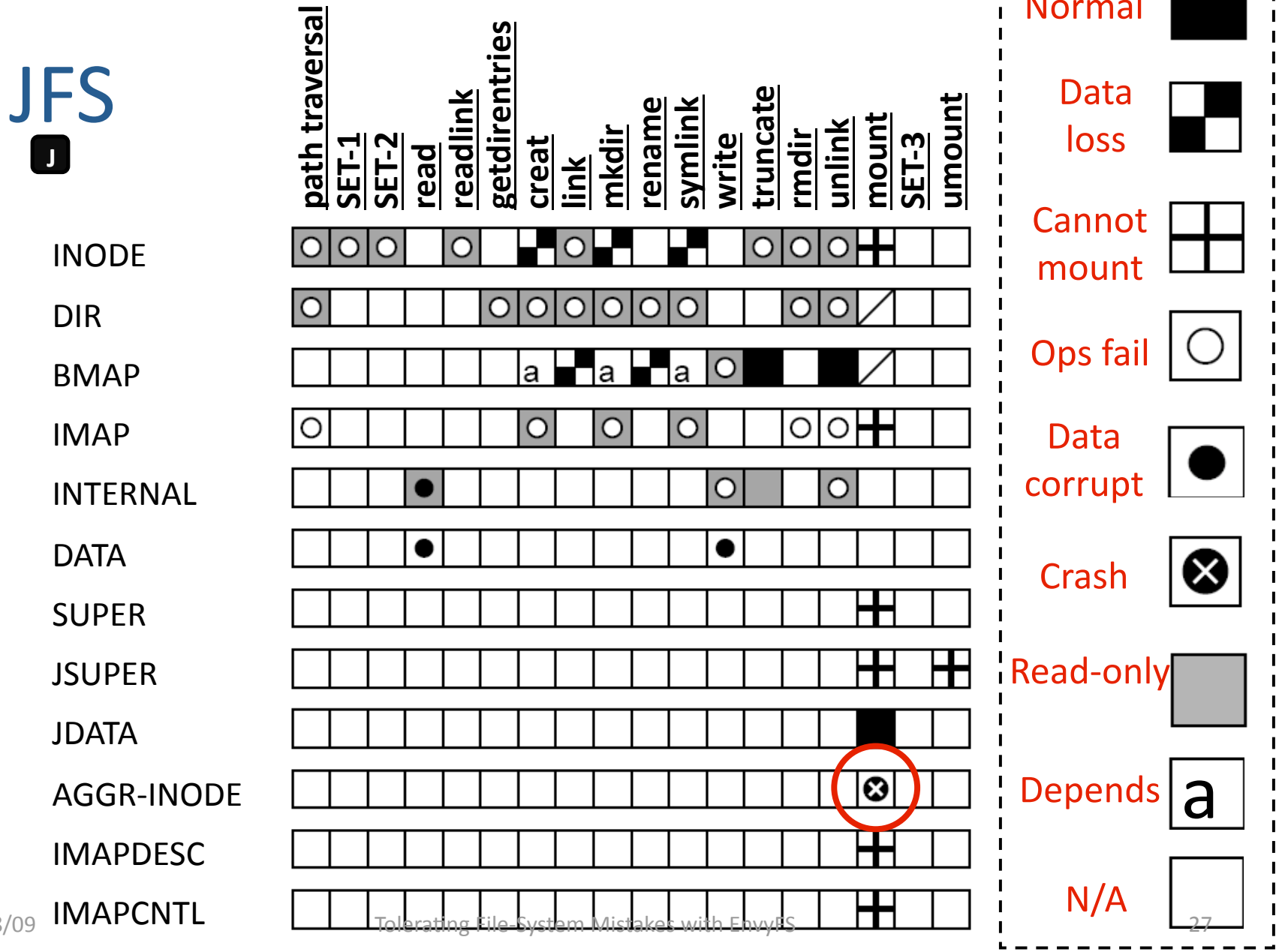
Potential for Bug Isolation



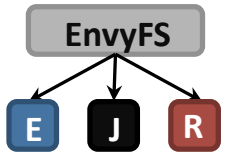
In typical use, a problem is noticed only on panic

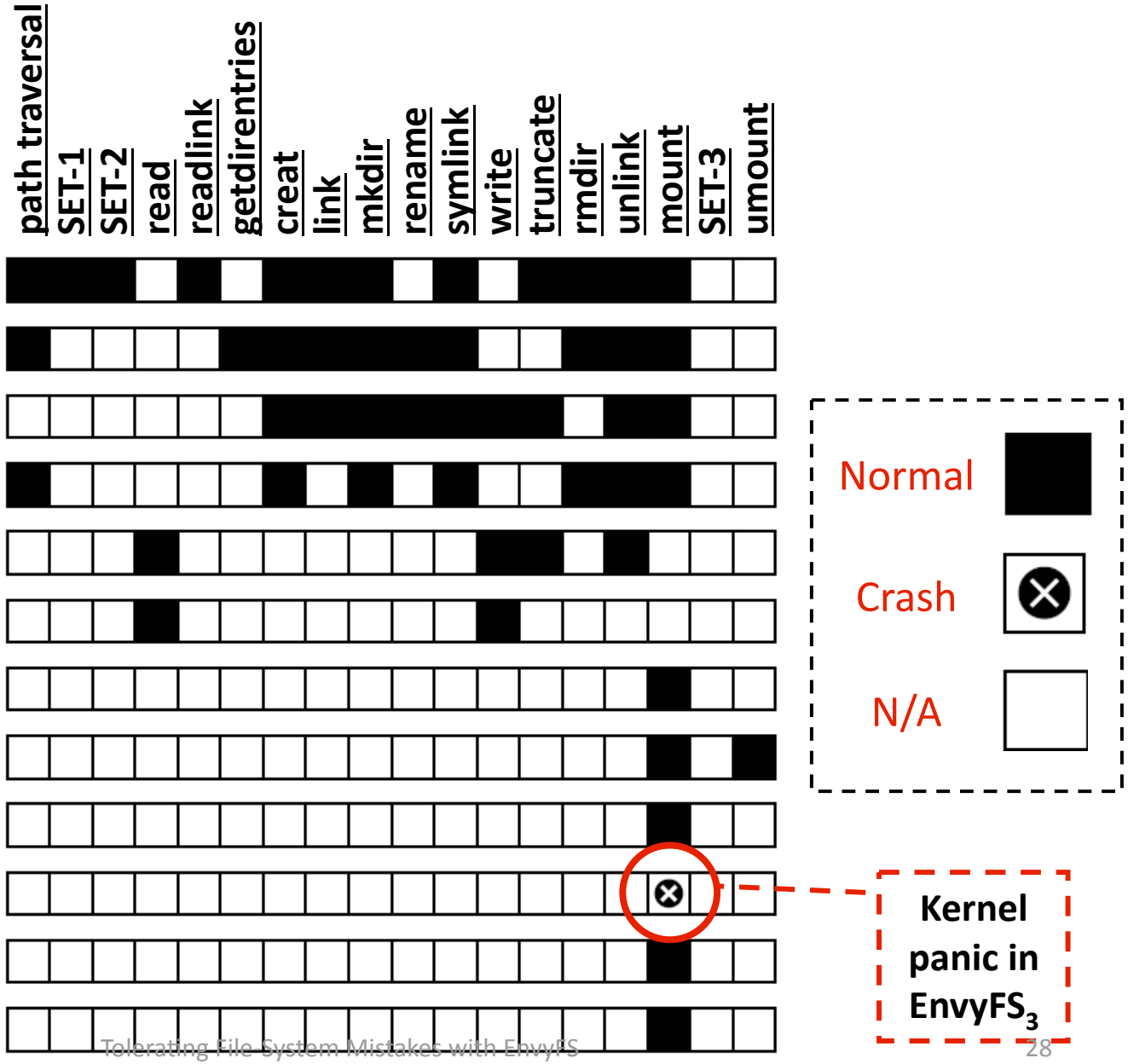
In $EnvyFS_3$, a problem is noticed the first time child file system returns wrong results

JFS

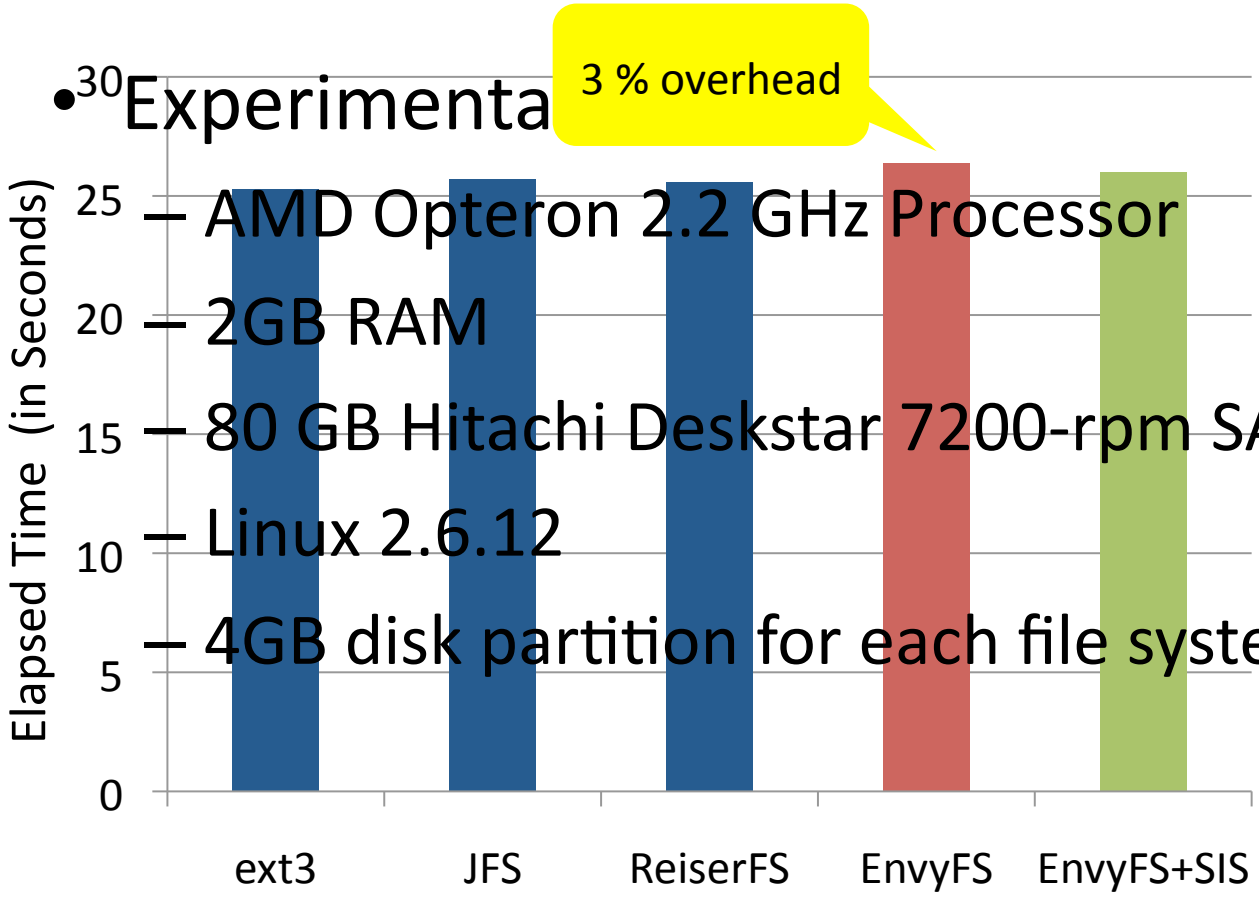

path traversal
SET-1
SET-2
read
readlink
getdirentries
creat
link
mkdir
rename
symlink
write
truncate
rmdir
unlink
mount
SET-3
umount
INODE
DIR
BMAP
IMAP
INTERNAL
DATA
SUPER
JSUPER
JDATA
AGGR-INODE
IMAPDESC
IMAPCNTL
a
Normal
Data loss
Cannot mount
Ops fail
Data corrupt
Crash
Read-only
Depends
N/A

EnvyFS$_3$


EnvyFS
E
J
R


path traversal
SET-1
SET-2
read
readlink
getdirentries
creat
link
mkdir
rename
symlink
write
truncate
rmdir
unlink
mount
SET-3
umount
INODE
DIR
BMAP
IMAP
INTERNAL
DATA
SUPER
JSUPER
JDATA
AGGR-INODE
IMAPDESC
IMAPCNTL
Normal
Crash
N/A
Kernel panic in EnvyFS$_3$

OpenSSH Benchmark



- Experimental Setup
  - AMD Opteron 2.2 GHz Processor
  - 2GB RAM
  - 80 GB Hitachi Deskstar 7200-rpm SATA disk
  - Linux 2.6.12
  - 4GB disk partition for each file system

- CPU Intensive
- OpenSSH 4.5

-- Copy, untar and make

Performance of EnvyFS$_3$ is comparable to a single file system

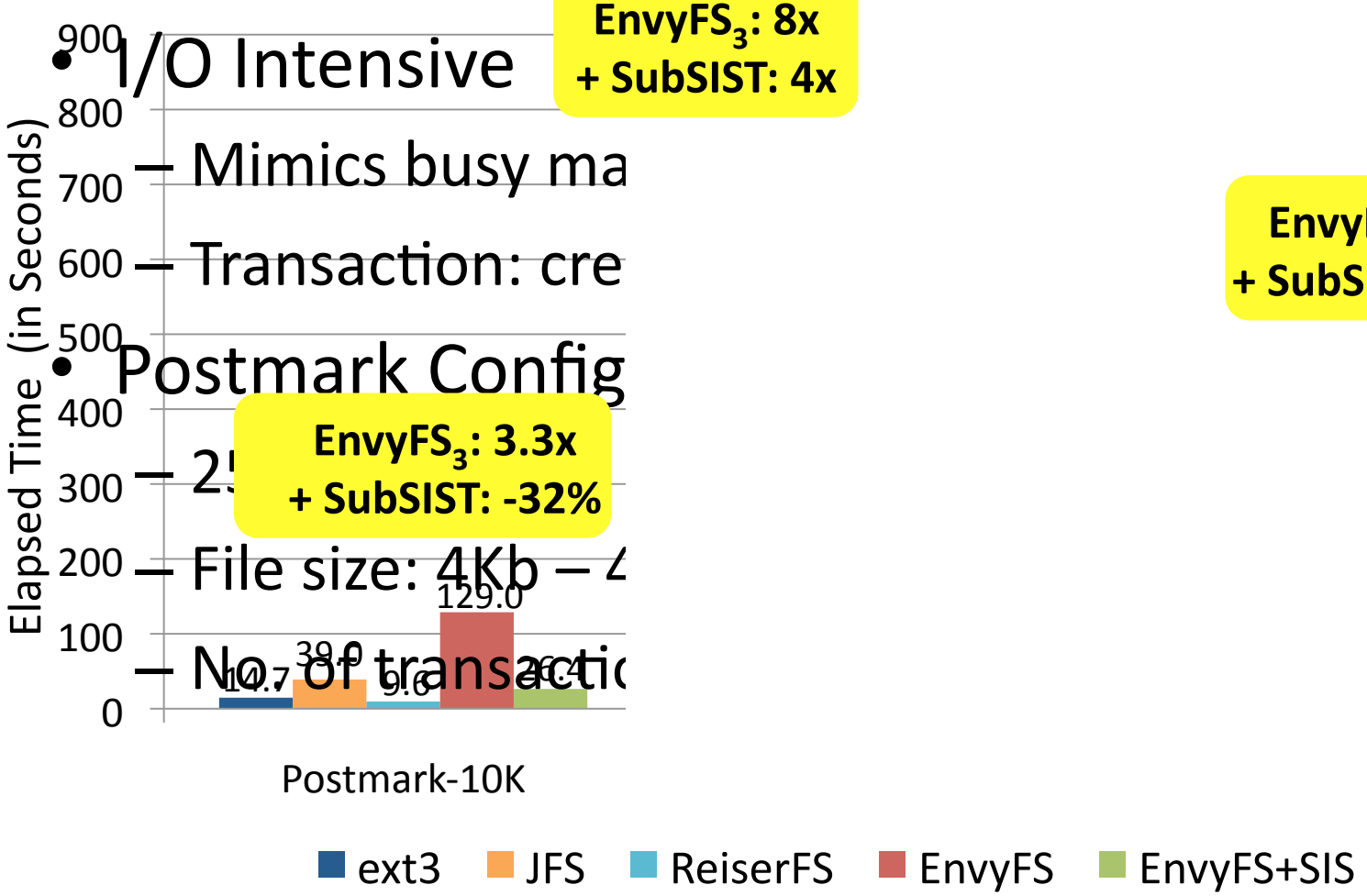
Postmark Benchmark

- I/O Intensive
  - Mimics busy ma
  - Transaction: cre
- Postmark Config
  - 2
  - File size: 4Kb – 4
  - No. of transactio

EnvyFS$_3$: 8x
+ SubSIST: 4x

EnvyFS$_3$: 3.3x
+ SubSIST: -32%

EnvyFS$_3$: 1.7x
+ SubSIST: 11.5%



Summary of Results

- Robustness
  - Traditional file systems vulnerable to corruptions
  - $EnvyFS_3$ tolerates almost all mistakes in one FS
- Performance
  - Desktop workloads: $EnvyFS_3$ has comparable performance
  - I/O intensive workloads:
    - Regular Operations: $EnvyFS_3$ + SubSIST acceptable performance
    - Memory pressure: $EnvyFS_3$ + SubSIST has large overhead

Outline

- Introduction
- Building reliable file systems
- Reducing overheads with SubSIST
- Evaluation
- **Conclusion**

Conclusion

- Bugs/mistakes are inevitable in any software
  - Must cope, not just hope to avoid
- EnvyFS: N-version approach to tolerating FS bugs
  - Built using existing specification and file systems
- SubSIST: single instance store
  - Decreases overheads while retaining reliability

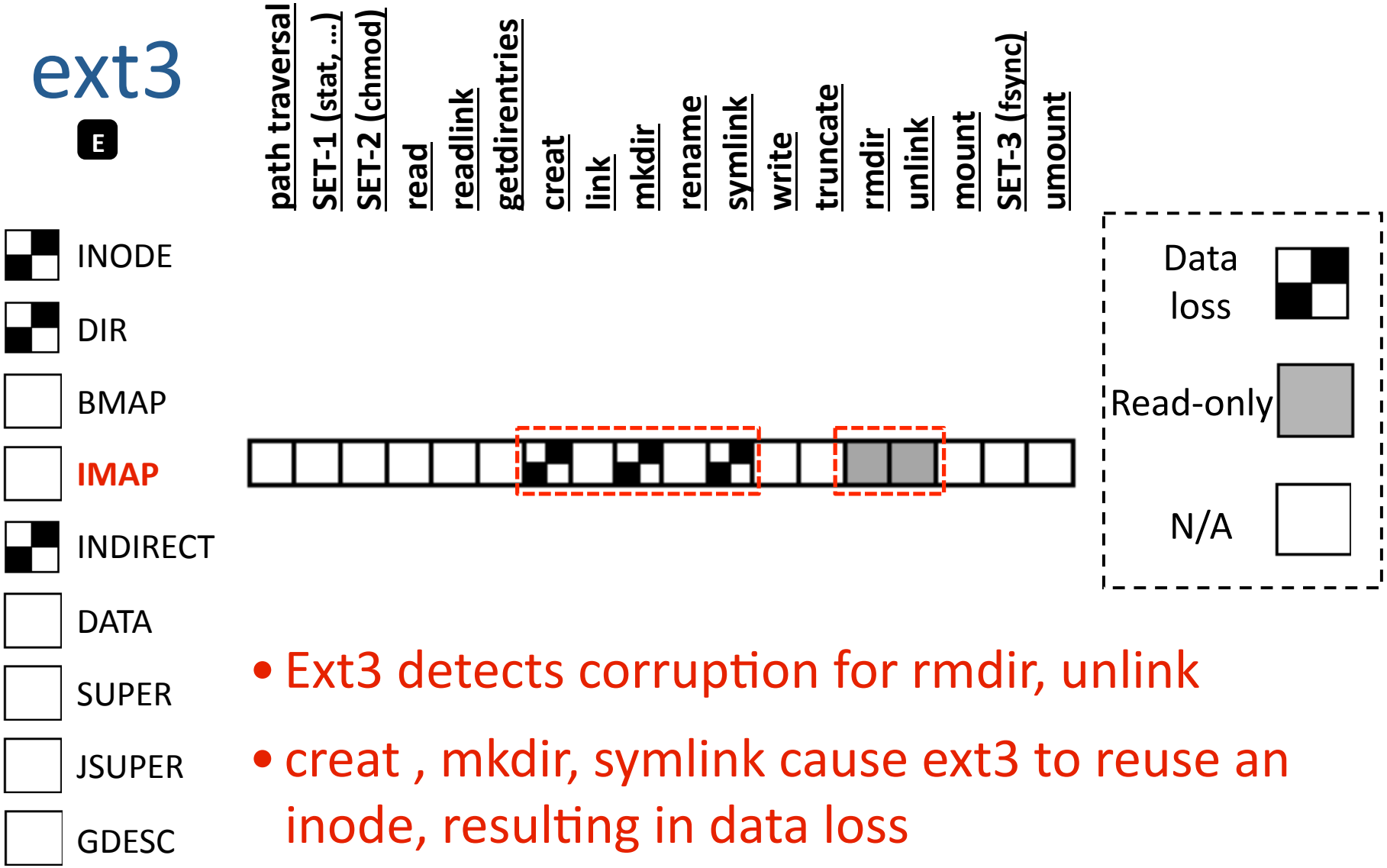
Thank You!

Advanced Systems Lab (ADSL)
University of Wisconsin-Madison
http://www.cs.wisc.edu/adsl

Future Work

- Debugging tool for developers
  - Run older and newer version of file systems
  - Compare results with older version
- File system repair
  - Simple repair: copy data from other file system
  - Complex repair: recreate entire file system tree
  - How to do micro repair ?

ext3



- Ext3 detects corruption for rmdir, unlink
- creat , mkdir, symlink cause ext3 to reuse an inode, resulting in data loss