Refuse to Crash with Re-FUSE

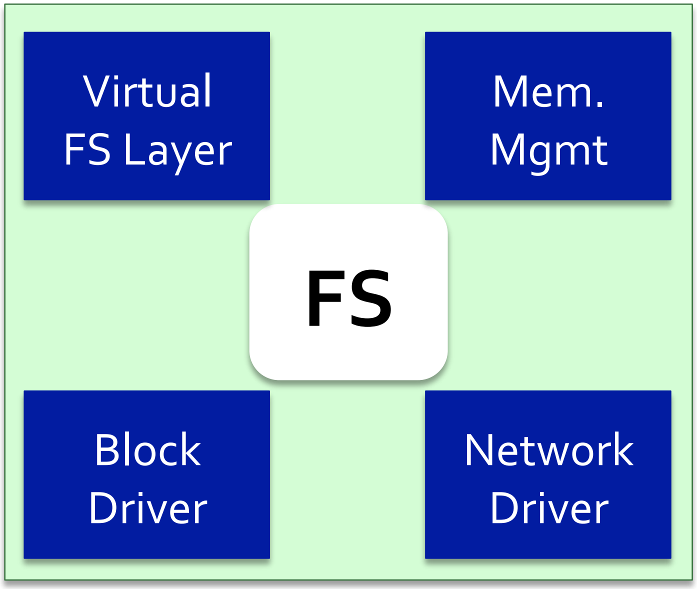
Swaminathan Sundararaman, Laxman Visampalli,
Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau

ADSL

THE UNIVERSITY of WISCONSIN MADISON

Kernel-level File Systems

- Hard to develop FSes
  - Interact with OS components
  - Support variety of features
  - Difficult to debug
- *Require:* skilled developers
- New file system in 5-10 years
  - Adding user-desired features is difficult



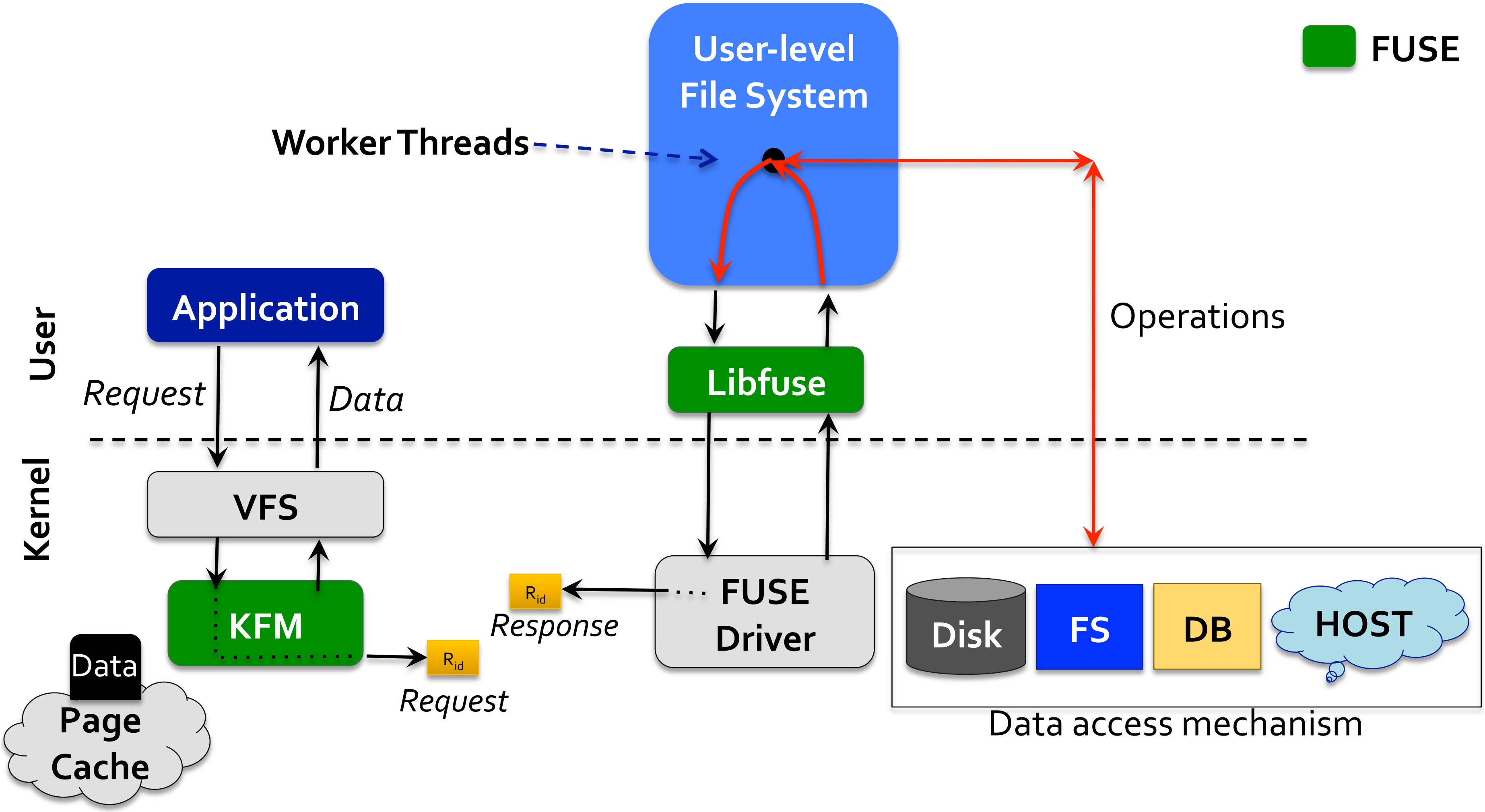
Operating System

FUSE: File system in USErspace

- Framework to run FS as a user-level process
  - Simplify development and deployment of FSes
  - FS interface to access underlying data
    - Database, email, ftp, http, ssh

- ~180 user-level file systems in few years
  - Can be customized to do just one thing
    - Compression, remote access (interface to cloud storage)
  - Raw device, pass through, and network-based

FUSE Architecture



User-level File System Issues

- Quick development
  - Not your typical file-system developer
  - No rigorous testing
  - No good documentation on FUSE
    - API, error scenarios
- Result in crashes
  - Require manual intervention to fix the problem
    - Repair & restart
  - Users feels that file system "*does not work*"
    - Decreases adoption chances

Re-FUSE: Restartable FUSE

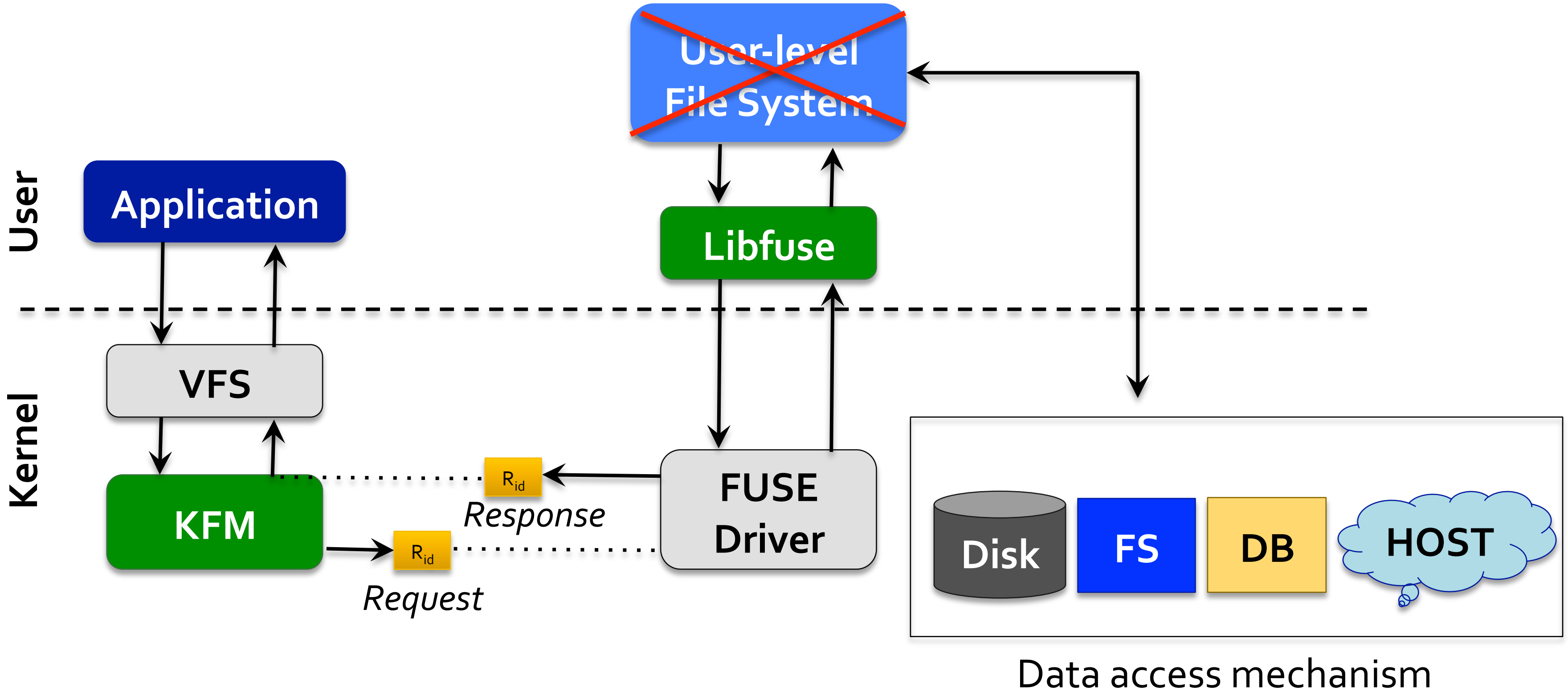
- Framework to restart FS on crashes
  - A generic mechanism to restart in transparent and stateful
    - Applications are *oblivious* to FS crashes
- Novel techniques
  - Request tagging, system call logging, non-interruptible system calls
  - Performance: page versioning and socket buffer caching
- Evaluation (Linux 2.6.18, FUSE 2.7.4, NTFS-3g, AVFS, and SSHFS)
  - Generality: < 10 lines of code for each FS
  - Robustness: 60 controlled & 300 random fault injection
  - Performance: < 13% for both micro & macro benchmarks
  - Recovery time: < 300 milliseconds to restart FS

Outline

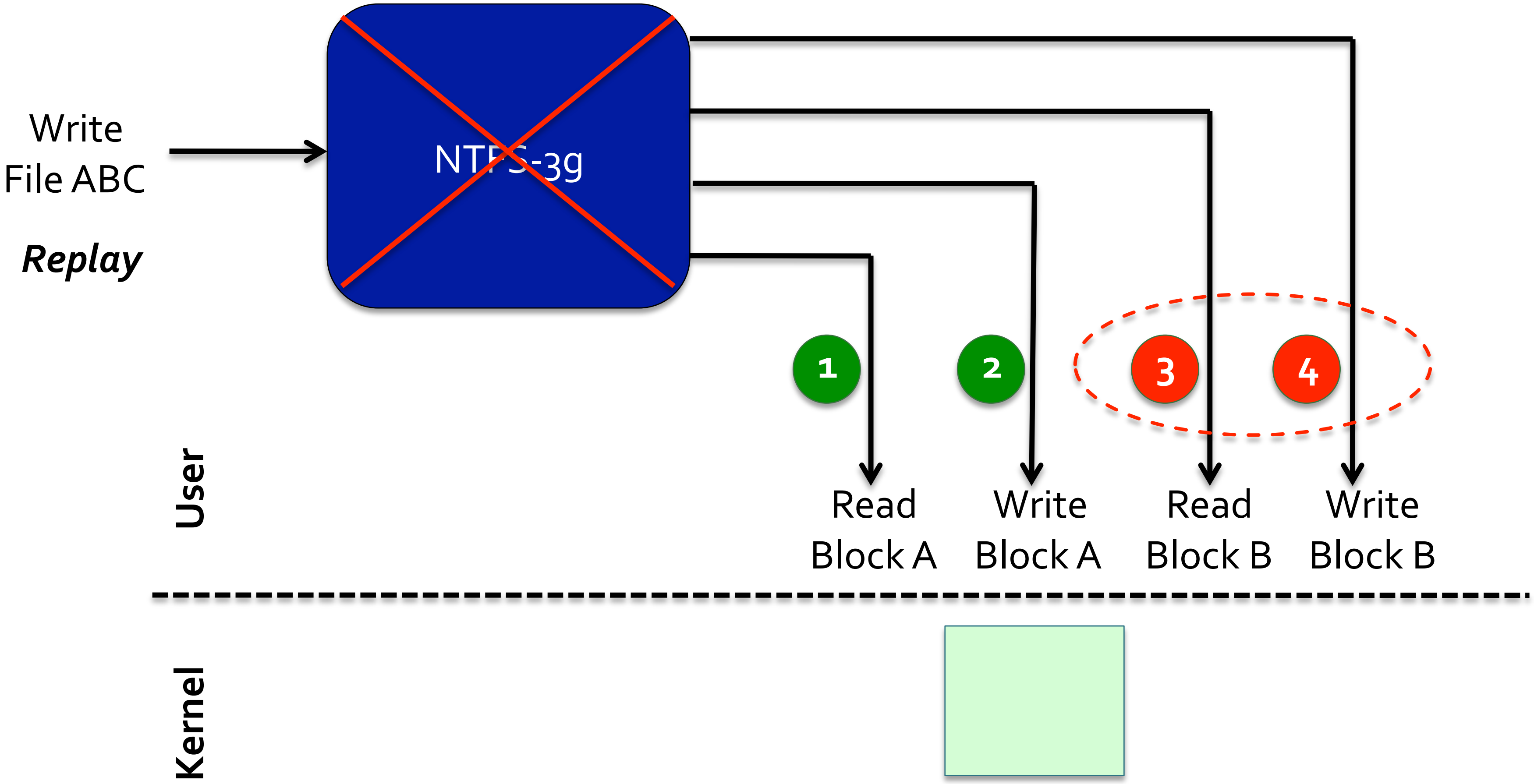
- Background
- Challenges
- Re-FUSE
- Evaluation
- Conclusions

Failure Model

- Only the user-level file system is unavailable

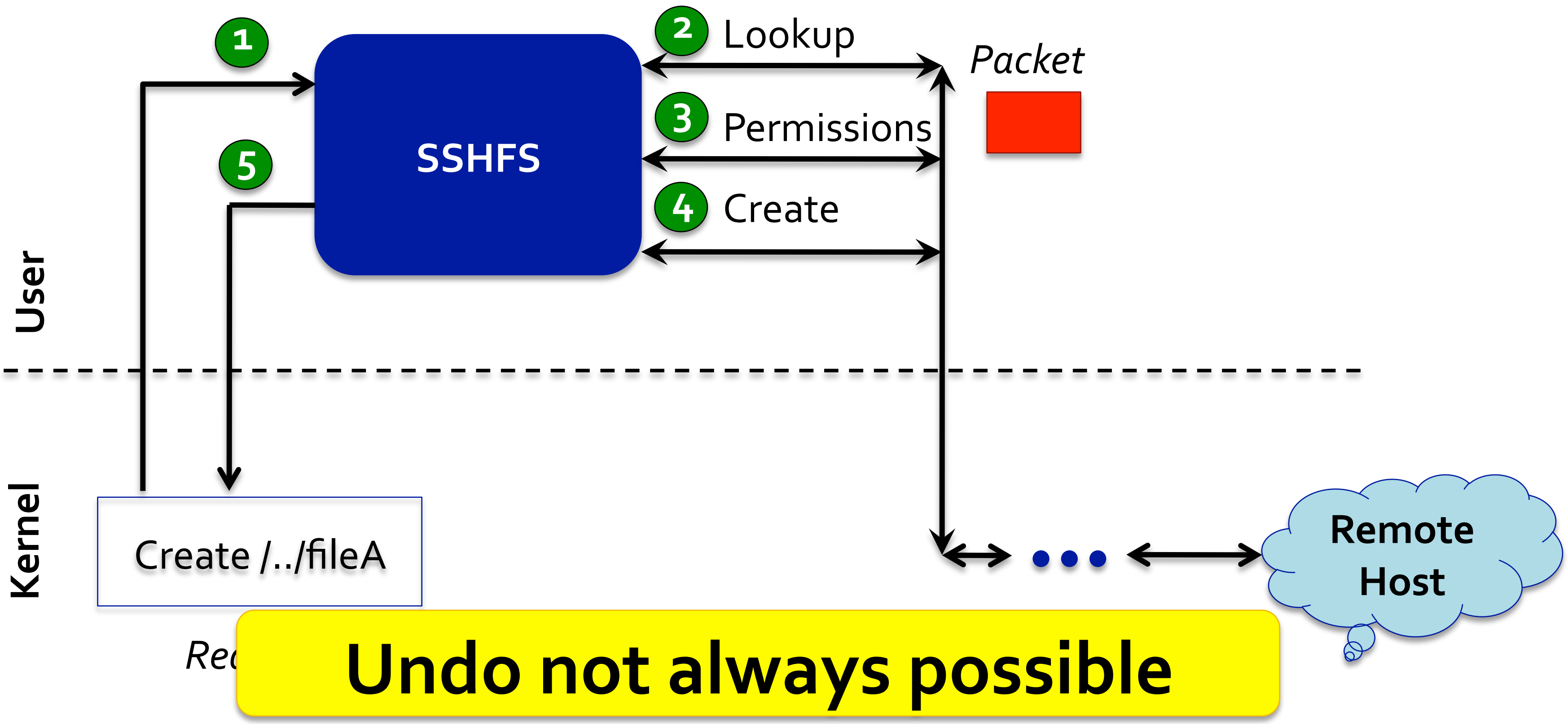


Can Simple Re-execution Work?



File System is left in an inconsistent state

How About Undoing Operations?



Undo not always possible

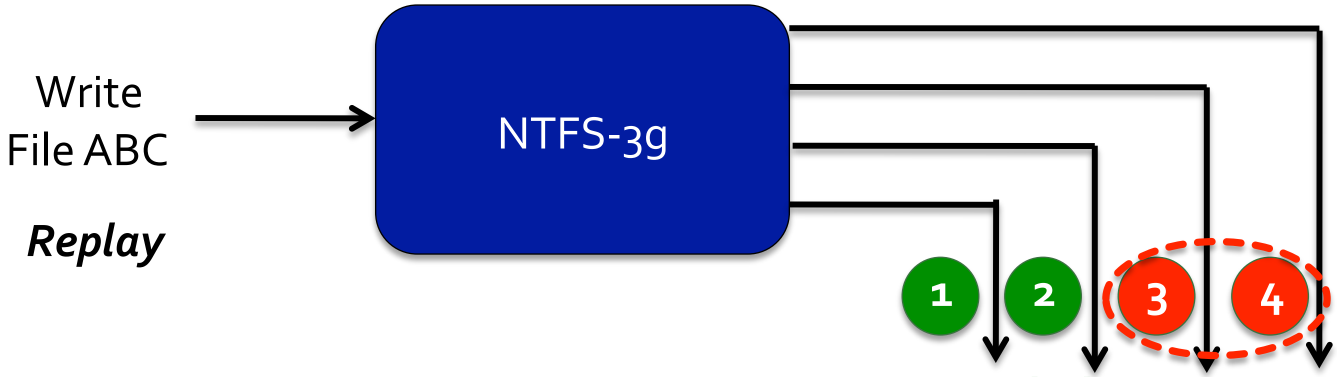
Need: alternate mechanism to restore FSes

Outline

- Background
- Challenges
- Re-FUSE
- Evaluation
- Conclusions

Re-FUSE

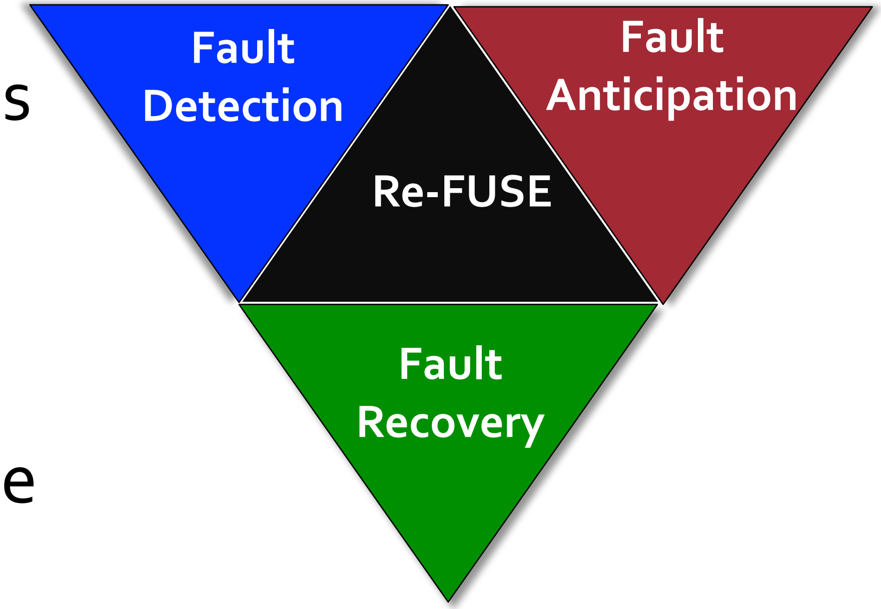
- Restartable FUSE
  - Designed to deal with file system crashes
  - Framework built inside FUSE and OS
- **Principle:** inconsistent ⟶ consistent state



Continue from the last completed operation

Components of Re-FUSE

- Fault Detection
  - Fail-stop, transient faults
  - Monitors FS crashes



- Fault Anticipation
  - Records file-system state
- Fault Recovery
  - Restart FS process and re-execute requests

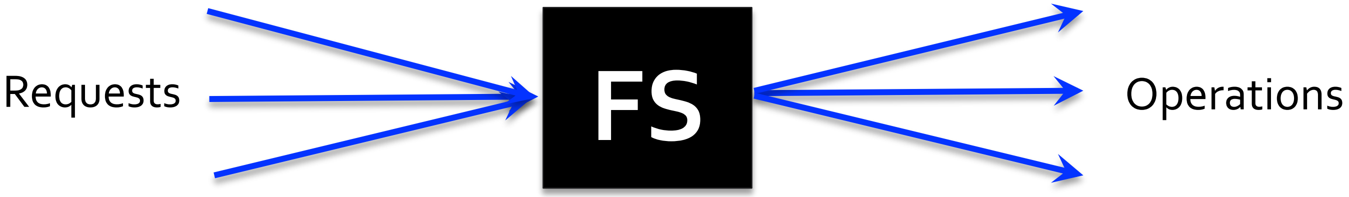
Fault Anticipation

Additional work done in preparation of failure

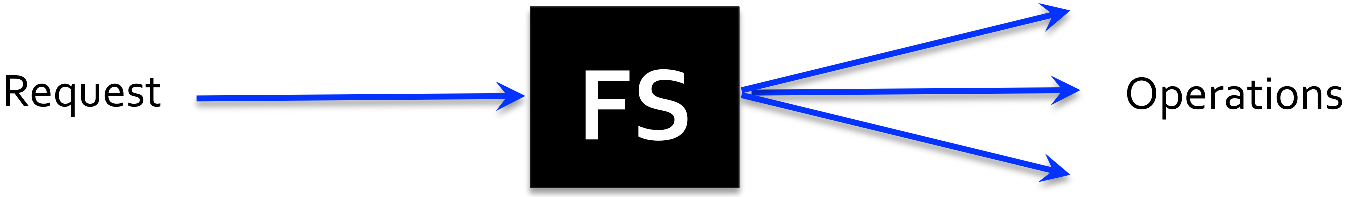
- Simplified due to FUSE architecture
  - Separate process executes FS requests
    - In-fight requests are preserved
  - Only the user-level file system crashes on failure
    - Applications state and updates (data) are preserved
- **Need:** generic mechanism to track progress
  - Mimic re-execution with sufficient recorded state

Challenges In Tracking Progress

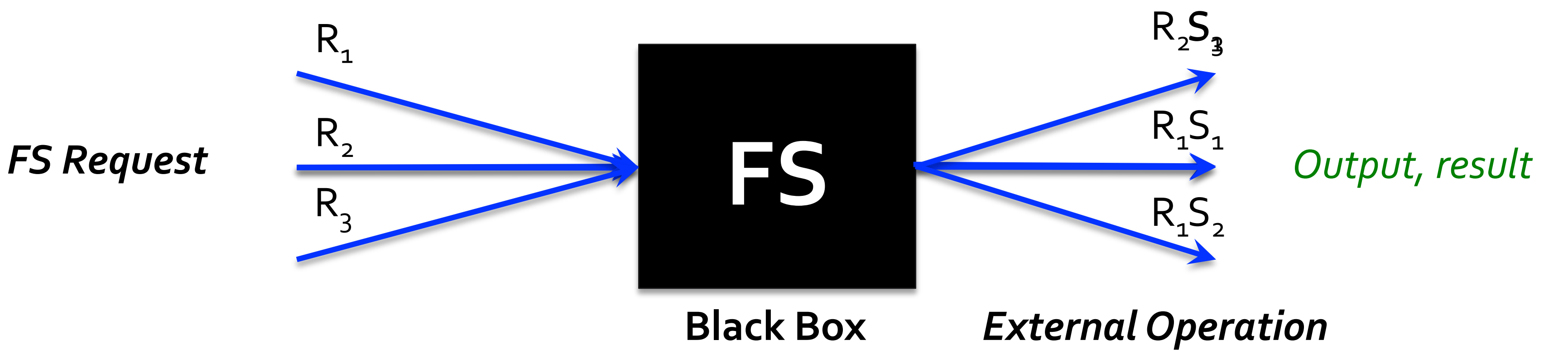
- Correlate FS actions with in-flight requests
  - Decoupled execution & multi threading



- Determine the state of individual sub tasks
  - Request splitting



Straw Man Approach

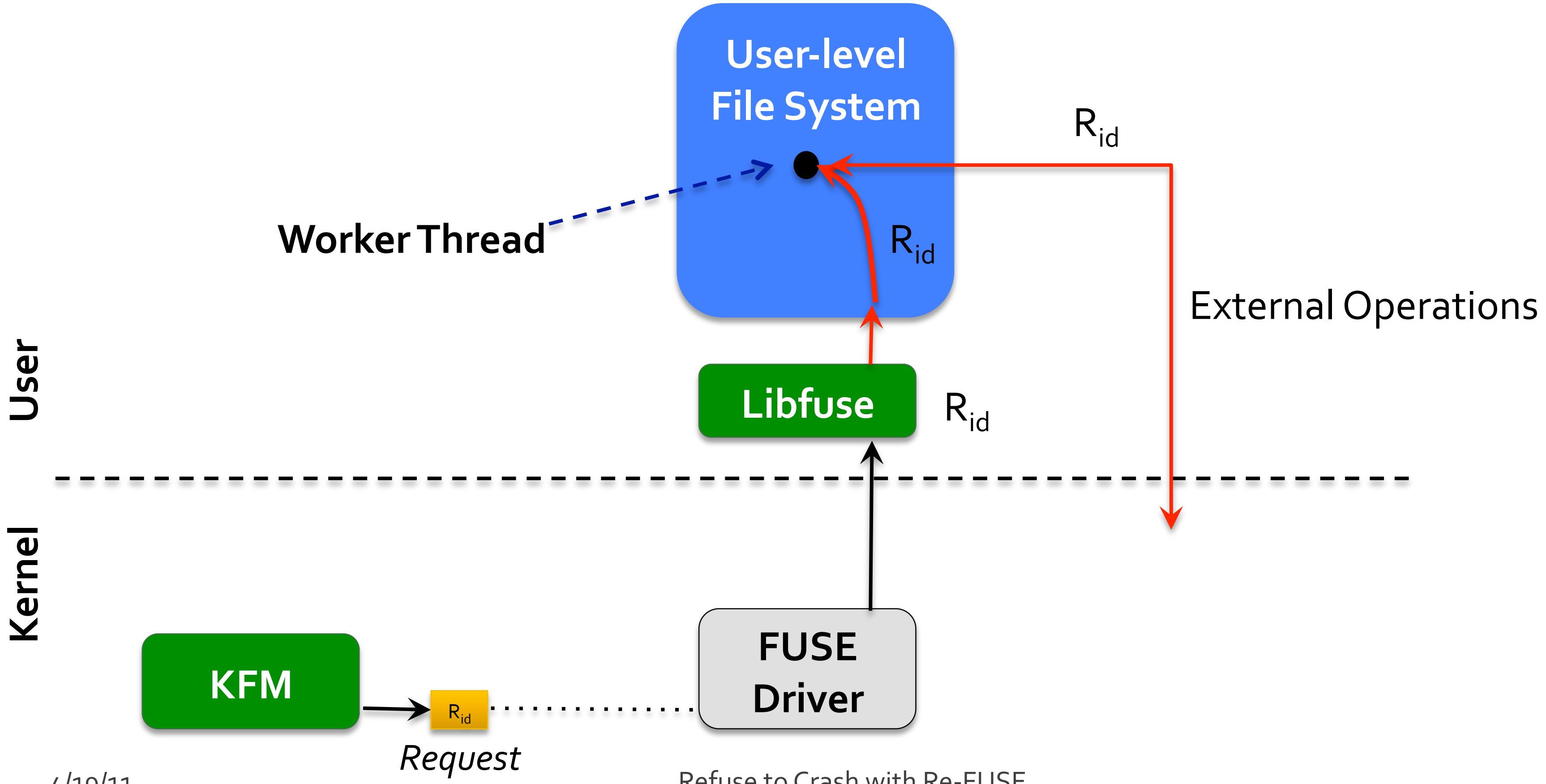


What if parameters are the same?

- Single request, single external operation
  - Require no additional support
- Single request, many external operations
  - Sequence number to correlate operations
- Many requests, many external operations
  - Request id and sequence number to correlate request and operation

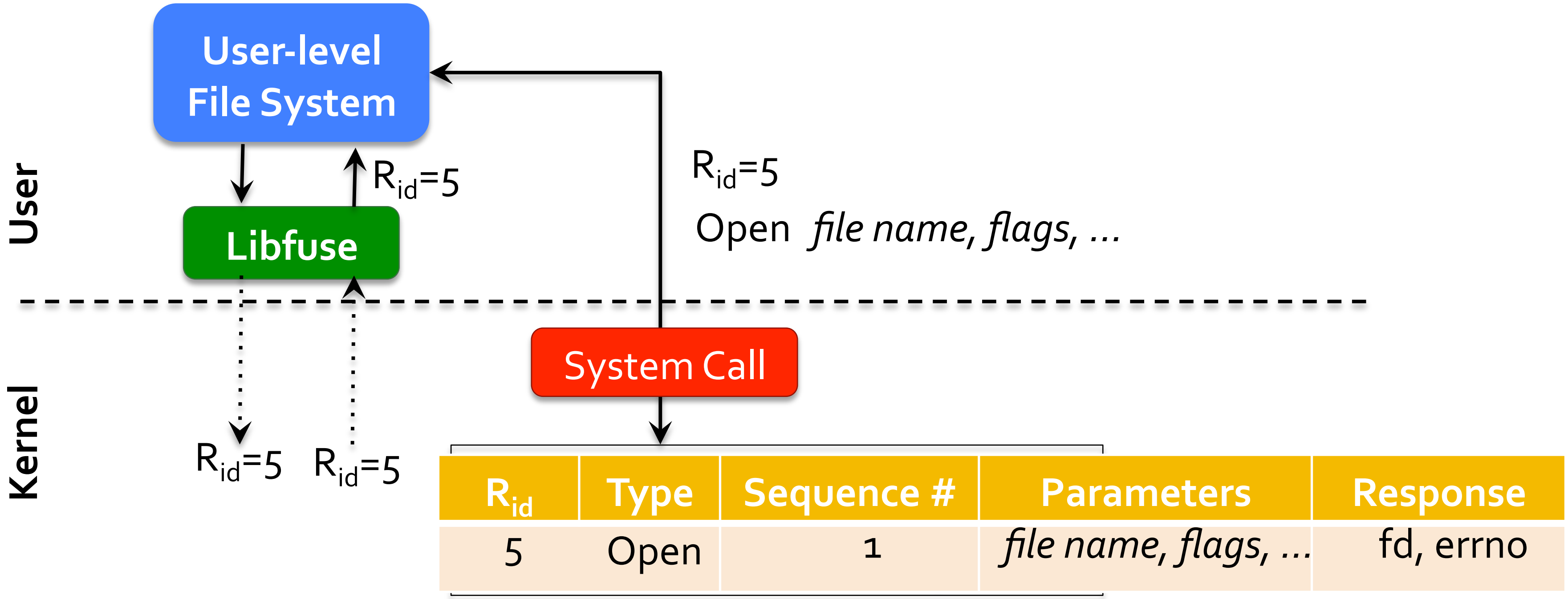
Request Tagging

- Correlates FS operation with external calls
    - Attach the fuse request id to the work thread



System Call Logging

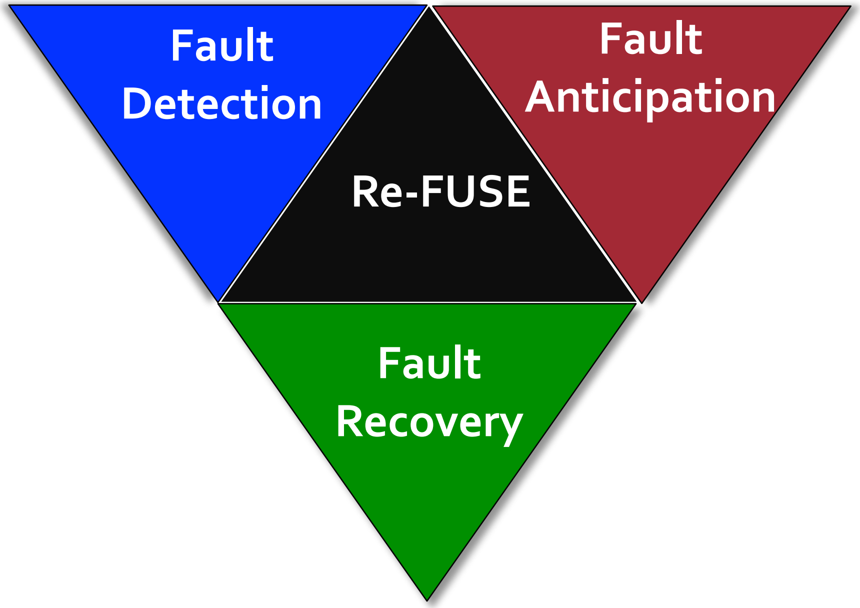
- Track progress of individual operations
  - On replay: helps mimic execution of completed requests



R_{id}	Type	Sequence #	Parameters	Response
5	Open	1	*file name, flags, …*	fd, errno

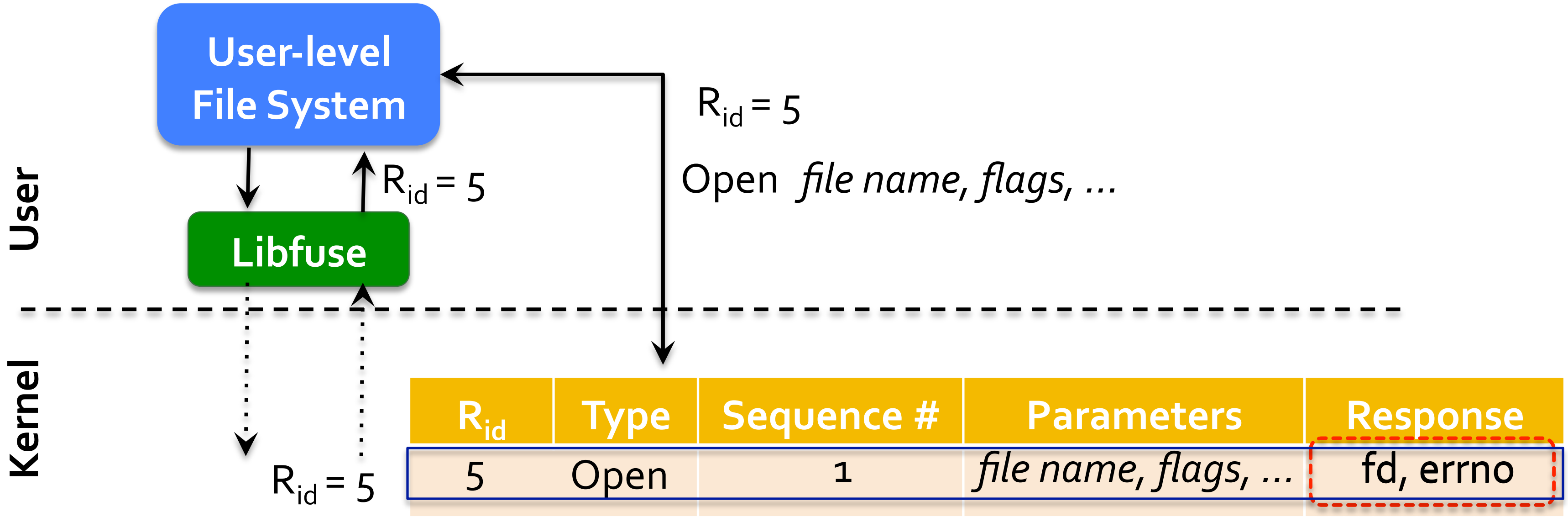
Syscall Request-Response Table

Components of Re-FUSE



Recovery

- Restore FS state needed to execute requests
  - Leverage cached state at the VFS layer
- Re-execute in-flight requests on restart
  - Leverage request queue at the KFM layer



R_{id}	Type	Sequence #	Parameters	Response
5	Open	1	*file name, flags, …*	fd, errno

Syscall Request-Response Table

Outline

- Background
- Challenges
- Re-FUSE
- Evaluation
- Conclusions

Evaluation

- Address the following questions
  - Implementation effort to work with Re-FUSE?
  - Can Re-FUSE hide failures from applications?
  - Performance overheads during user workloads?

- Experimental setup
  - 2.2 GHz Opteron, 2GB Ram, 2 80GB WDC disk
  - Implemented in Linux 2.6.18, FUSE (2.7.4)
  - Re-FUSE: 3300 loc in Linux kernel, 1000 loc in FUSE

Generality of Re-FUSE

	File System	Original	Added	Modified
Block-based interface	NTFS-3g	32K	10	1
Pass through interface	AVFS	39K	4	1
Network-based interface	SSHFS	4K	3	2

Code changes in individual FSes

- Additions
  - Daemonize user-level process
  - Notify local state (such as external file handles)
- Modifications
  - New mount interface that includes restart flag as parameter

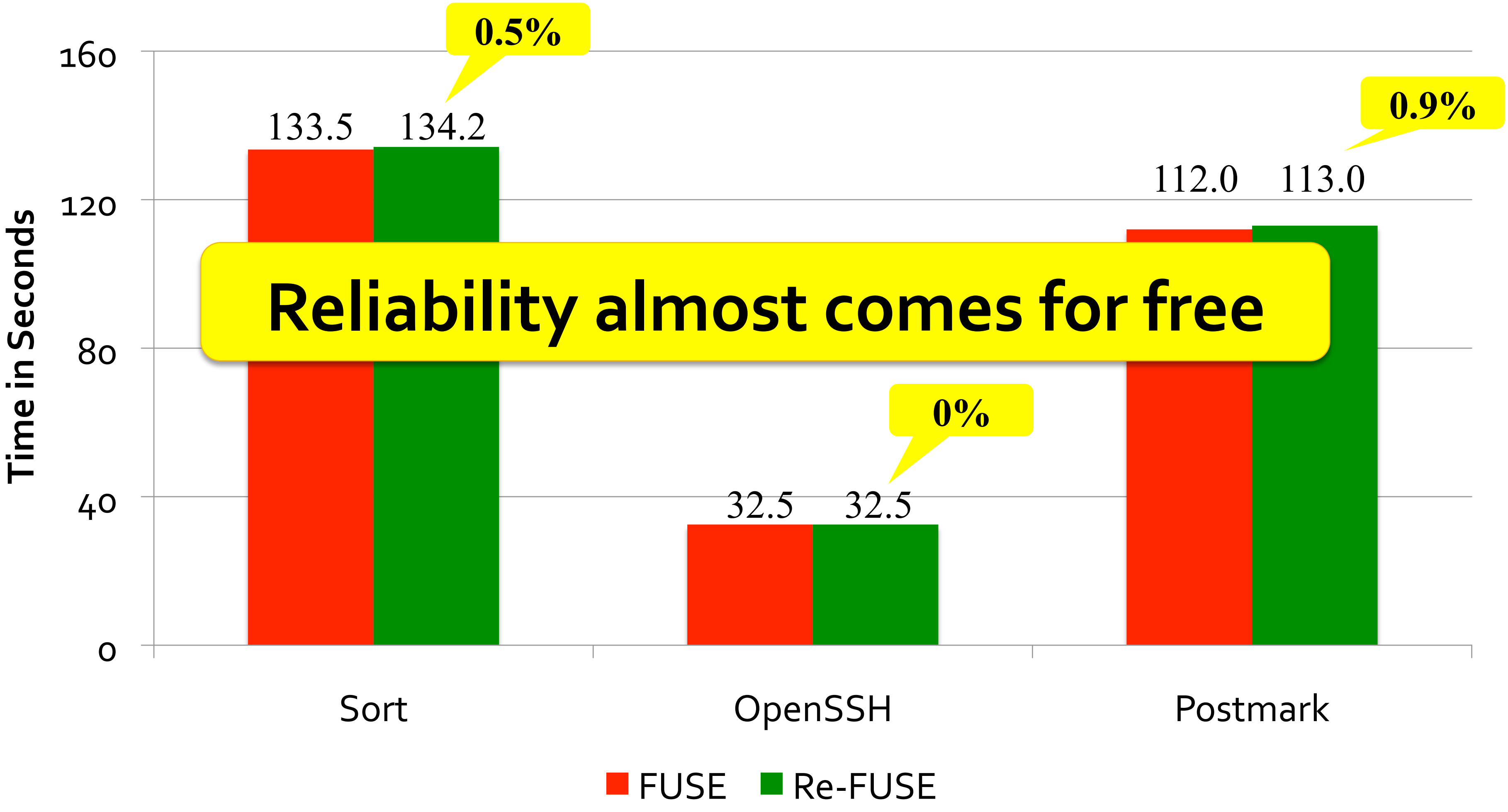
Robustness of Re-FUSE

- Inject transient faults
  - Crashes in FS
- Inspect appl., FS state
  - ✗ - bad ✓ - good
- NTFS-3g results
  - SSHFS, AVFS in paper
- Random fault injection
  - 100 faults for all 3 FSes

		FUSE NTFS-3g			Re-FUSE NTFS-3g		
Operation	NTFS-3g Function	Application?	FS Consistent?	FS Usable?	Application?	FS Consistent?	FS Usable?
create	fuse_create	✗	✗	✗	✓	✓	✓
mkdir	fuse_create	✗	✗	✗	✓	✓	✓
symlink	fuse_create	✗	✗	✗	✓	✓	✓
link	link	✗	✗	✗	✓	✓	✓
rename	link	✗	✗	✗	✓	✓	✓
open	fuse_open	✗	✓	✗	✓	✓	✓
						✓	✓
						✓	✓

Re-FUSE successfully hides failures

NTFS-3g Performance Overheads



0.5%

0.9%

0%

Reliability almost comes for free

Outline

- Overview
- Challenges
- Re-FUSE
- Evaluation
- Conclusions

Conclusions

"Failure is not falling down but refusing to get up."
- Chinese proverb

- Reliability through restartability
  - A generic mechanism to restart user-level file systems
- ***Principle***: inconsistent → consistent state
  - Inconsistency due to incomplete operations
  - Track progress of operations to continue from last execution
- Novel techniques
  - *Request tagging:* differentiate between serviced requests
  - *System call logging:* tracks sequence of operations
  - *Non-interruptible system call:* move threads to safe state
  - *Page versioning*: minimize logging overheads

Thank You!

Questions?

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