# MEMBRANE: OPERATING SYSTEM SUPPORT FOR RESTARTABLE FILE SYSTEMS

SWAMINATHAN SUNDARARAMAN, SRIRAM SUBRAMANIAN, ABHISHEK RAJIMWALE, ANDREA C. ARPACI-DUSSEAU, REMZI H. ARPACI-DUSSEAU, MICHAEL M. SWIFT

## CARSTEN WEINHOLD

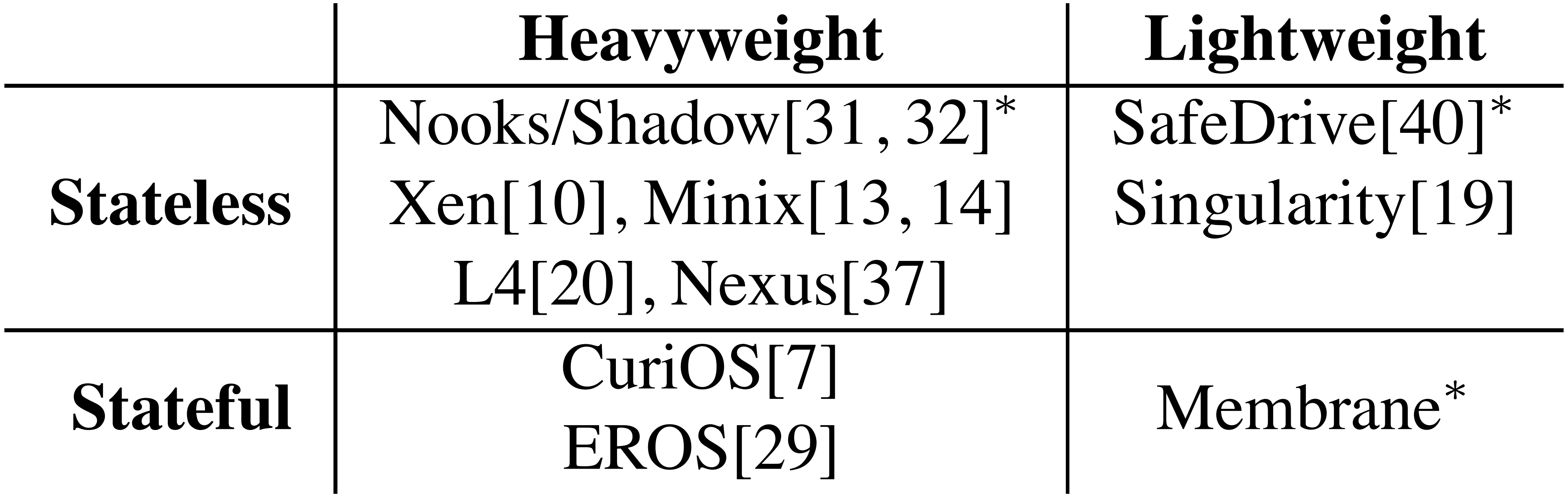
# MOTIVATION

- „Operating Systems crash.“
- „File systems fail.“
- Many bugs are in file systems
- Reasons:
  - Complex code bases
  - Under active development
  - Large number of file systems
- How to fix?

# HOW TO RECOVER?

- Research on OS subsystem recovery:
  - Isolation, micro-rebooting
  - Checkpoint / restart
- Problem: file systems are stateful
  - On-disk data
  - In-memory data
  - Spread across kernel / user memory

# SOLUTIONS

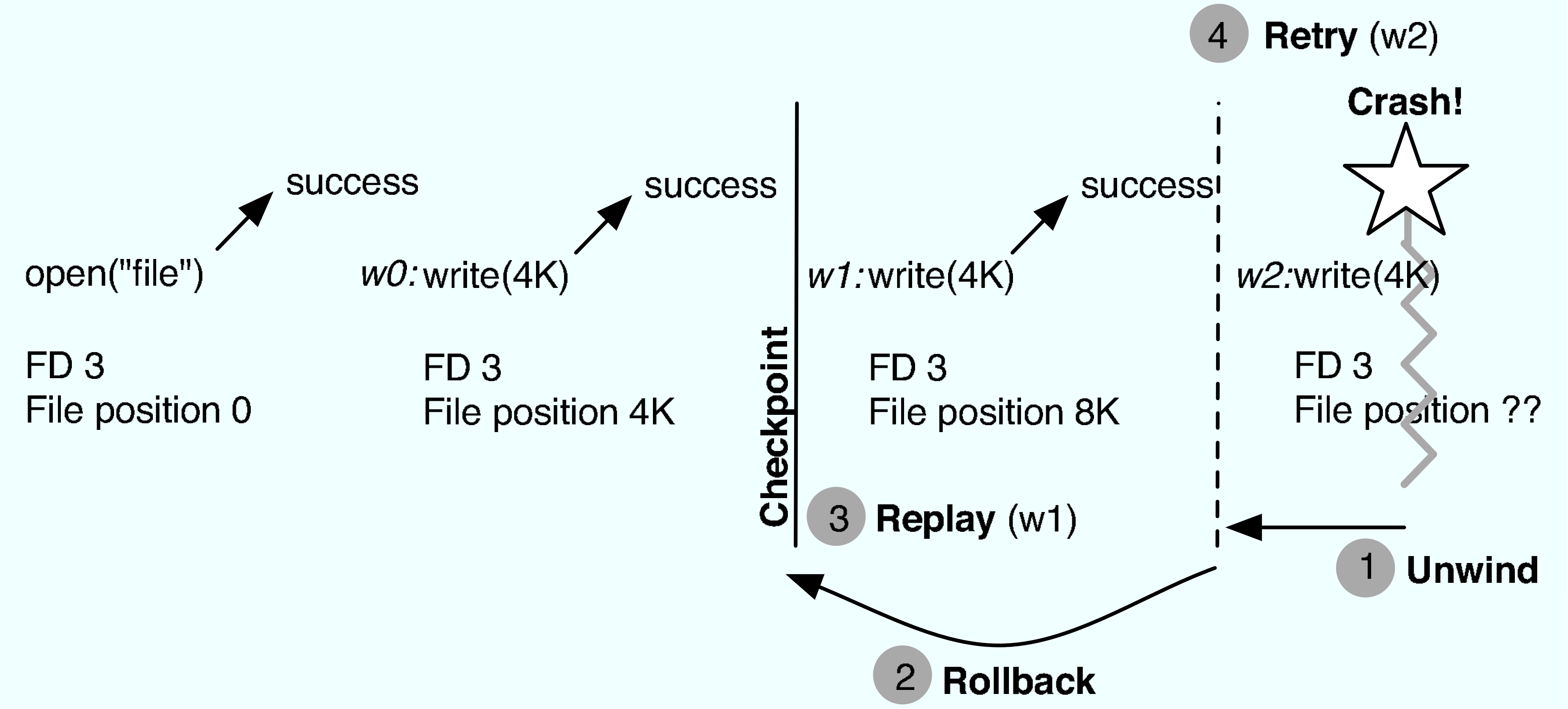


| | **Heavyweight** | **Lightweight** |
|---|---|---|
| **Stateless** | Nooks/Shadow[31, 32]*<br>Xen[10], Minix[13, 14]<br>L4[20], Nexus[37] | SafeDrive[40]*<br>Singularity[19] |
| **Stateful** | CuriOS[7]<br>EROS[29] | Membrane* |

# MEMBRANE

- Membrane is OS framework:
  - Light-weight stateful recovery
  - Checkpointing on-disk state
  - Logging of operations
  - In case of failure:
    - Park all file system operations
    - Cleanup state, reset file system
    - Replay logged operations from checkpoint
    - Continue

# OVERVIEW


4 Retry (w2)
Crash!
success
success
success
open("file")
*w0:* write(4K)
*w1:* write(4K)
*w2:* write(4K)
Checkpoint
FD 3
File position 0
FD 3
File position 4K
FD 3
File position 8K
FD 3
File position ??
3 Replay (w1)
1 Unwind
2 Rollback

# GOALS

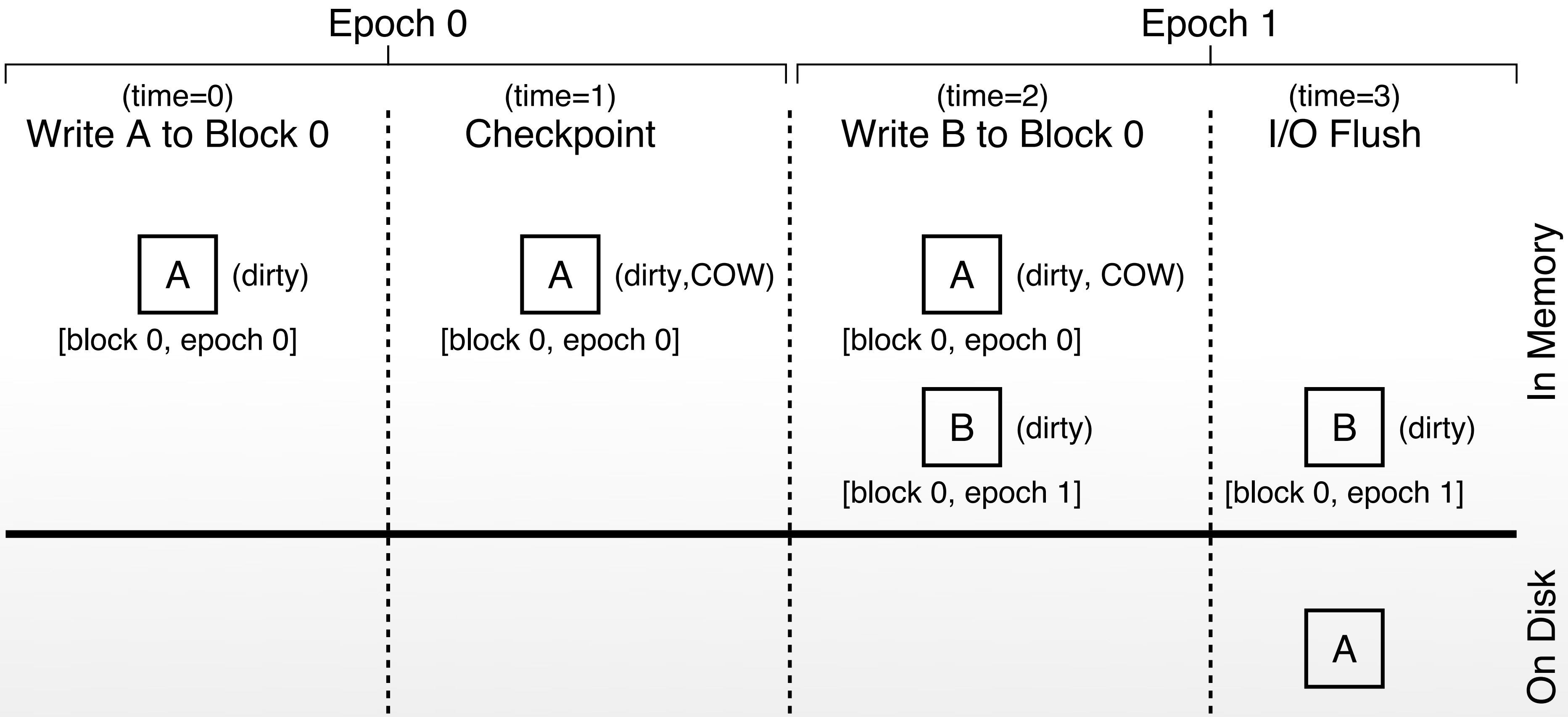
- Fault tolerant
- Lightweight
- Transparent
- Generic
- Maintain file-system consistency

# FAULT DETECTION

- Aims at *transient fail-stop* errors
- Light-weight detection:
  - Exceptions (divide-by-zero, page fault, ...)
  - assert(), panic(), BUG(), ...
  - Argument checks at kernel / file system boundaries
- No address-space isolation, etc.

# CLEAN STATE

- Recovery requires clean on-disk state
- Based on checkpointing
- Checkpoints mark begin / end of epochs



# CHECKPOINTING

- Reuse existing mechanisms:
  - Journaling support, snapshots, ...
  - Notify Membrane of begin / end of transaction
- Generic checkpointing at VFS level:
  - Park new file system operations
  - Wait for pending operations to complete
  - Copy dirty metadata back to buffers
  - Mark dirty buffers copy-on-write
  - Write back asynchronously

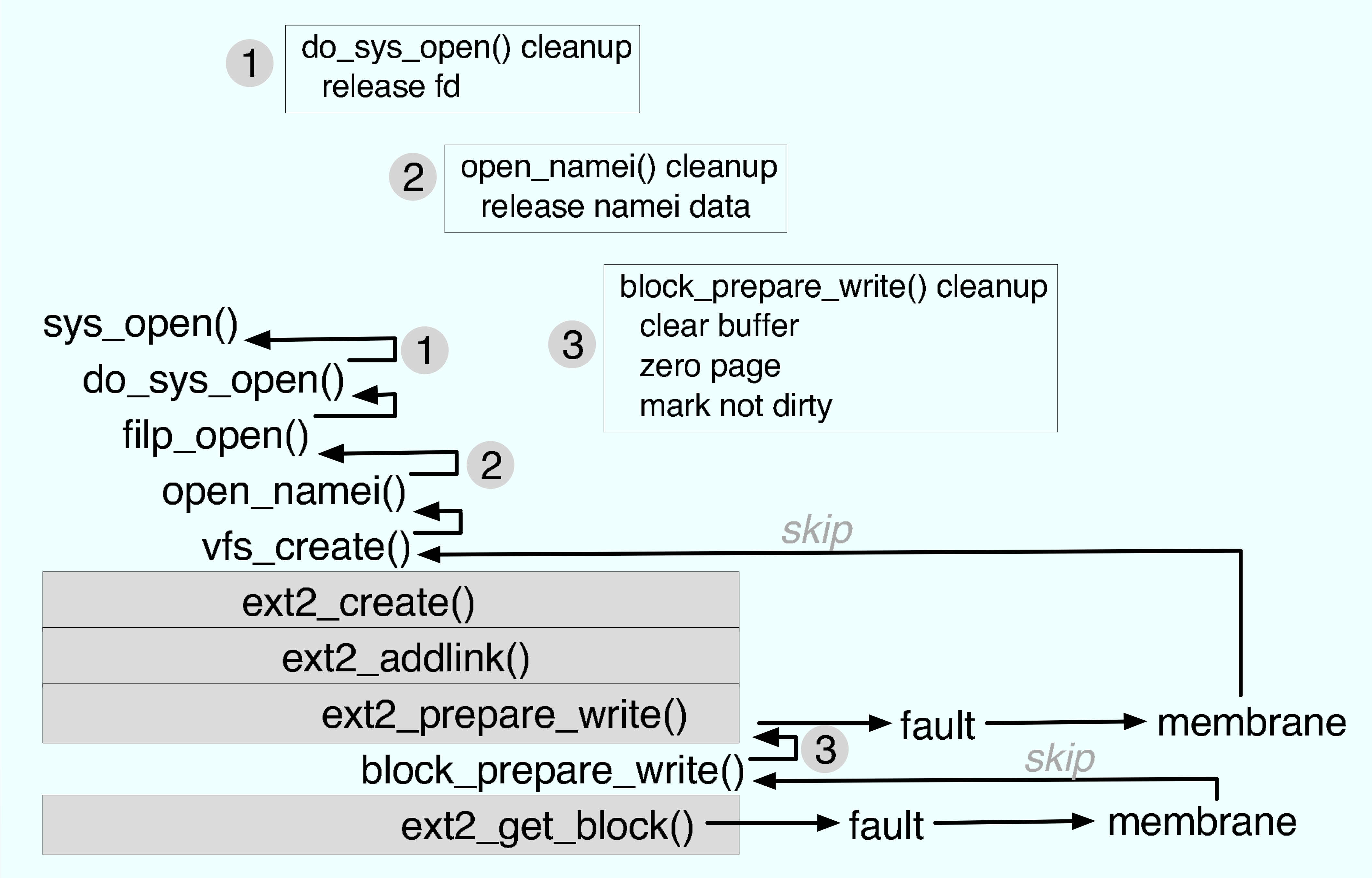
# LOGS

- **operation log:** operations and data
- **session log:** open files from previous epoch, file pointers, ...
- **malloc table:** all memory allocated by file system
- **lock stack:** all held global locks for LIFO releasing
- **unwind stack:** register state to support unwinding

# RECOVERY

- Halt execution and park threads
- Unwind in-flight threads
- Commit dirty pages from last epoch to stable storage
- Kill file system („unmount“)
- Restart file system („mount“)
- Roll forward logged operations / state
- Resume execution

# SKIP/TRUST

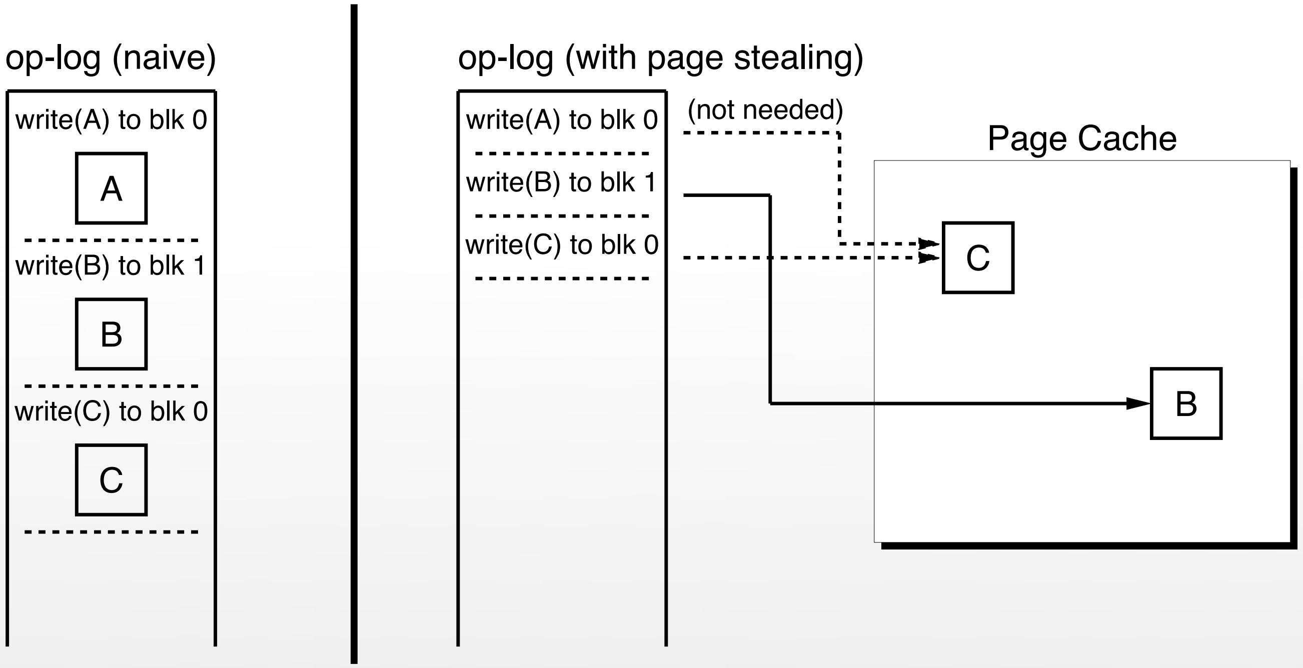

1
do_sys_open() cleanup
release fd
2
open_namei() cleanup
release namei data
3
block_prepare_write() cleanup
clear buffer
zero page
mark not dirty
sys_open()
do_sys_open()
filp_open()
open_namei()
vfs_create()
skip
ext2_create()
ext2_addlink()
ext2_prepare_write()
block_prepare_write()
ext2_get_block()
fault
membrane
skip
fault
membrane

# CLEANUP STATE

- Free all memory allocated by file system
- Release all global locks in LIFO order

# OPTIMIZATIONS

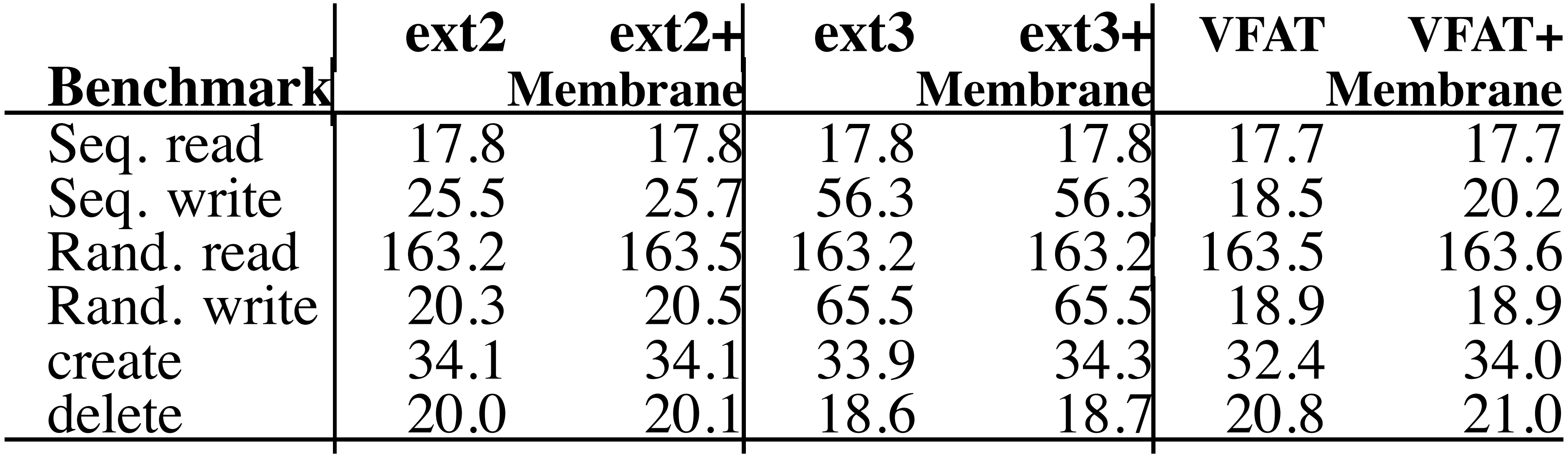
- Compressed operation log
- Uses „page stealing“
- Latest written data is in dirty pages
- „steal“ before recovery, write to disk



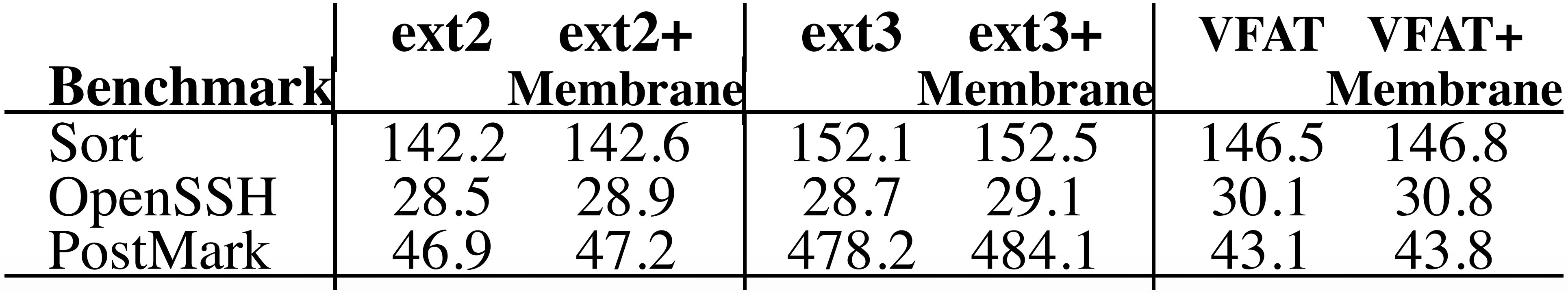
# FAULT STUDY

| | | ext2 | | | | ext2+ | | | | ext2+ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | boundary | | | | Membrane | | | |
| ext2_Function | Fault | How Detected? | Application? | FS:Consistent? | FS:Usable? | How Detected? | Application? | FS:Consistent? | FS:Usable? | How Detected? | Application? | FS:Consistent? | FS:Usable? |
| create | null-pointer | o | × | × | × | o | × | × | × | d | √ | √ | √ |
| create | mark_inode_dirty | o | × | × | × | o | × | × | × | d | √ | √ | √ |
| writepage | write_full_page | o | × | √ | √[a] | d | s | × | √[a] | d | √ | √ | √ |
| writepages | write_full_page | o | × | × | √[a] | d | s | × | √[a] | d | √ | √ | √ |
| free_inode | mark_buffer_dirty | o | × | × | × | o[b] | × | × | √[a] | d | √ | √ | √ |
| mkdir | d_instantiate | o | × | × | × | d | s | √ | √ | d | √ | √ | √ |
| get_block | map_bh | o | × | × | √[a] | o[b] | × | × | × | d | √ | √ | √ |
| readdir | page_address | G | × | × | × | G | × | × | × | d | √ | √ | √ |
| get_page | kmap | o | × | √ | × | o[b] | × | √ | × | d | √ | √ | √ |
| get_page | wait_page_locked | o | × | √ | × | o[b] | × | √ | × | d | √ | √ | √ |
| get_page | read_cache_page | o | × | √ | × | o | × | √ | × | d | √ | √ | √ |
| lookup | iget | o | × | √ | × | o[b] | × | √ | × | d | √ | √ | √ |
| add_nondir | d_instantiate | o | × | × | × | d | e | √ | √ | d | √ | √ | √ |
| find_entry | page_address | G | × | √ | × | G[b] | × | √ | × | d | √ | √ | √ |
| symlink | null-pointer | o | × | × | × | o | × | √ | × | d | √ | √ | √ |
| rmdir | null-pointer | o | × | √ | × | o | × | √ | × | d | √ | √ | √ |
| empty_dir | page_address | G | × | √ | × | G | × | √ | × | d | √ | √ | √ |
| make_empty | grab_cache_page | o | × | √ | × | o[b] | × | × | × | d | √ | √ | √ |
| commit_chunk | unlock_page | o | × | √ | × | d | e | × | × | d | √ | √ | √ |
| readpage | mpage_readpage | o | × | √ | √ | i | × | √ | √ | d | √ | √ | √ |

# PERFORMANCE



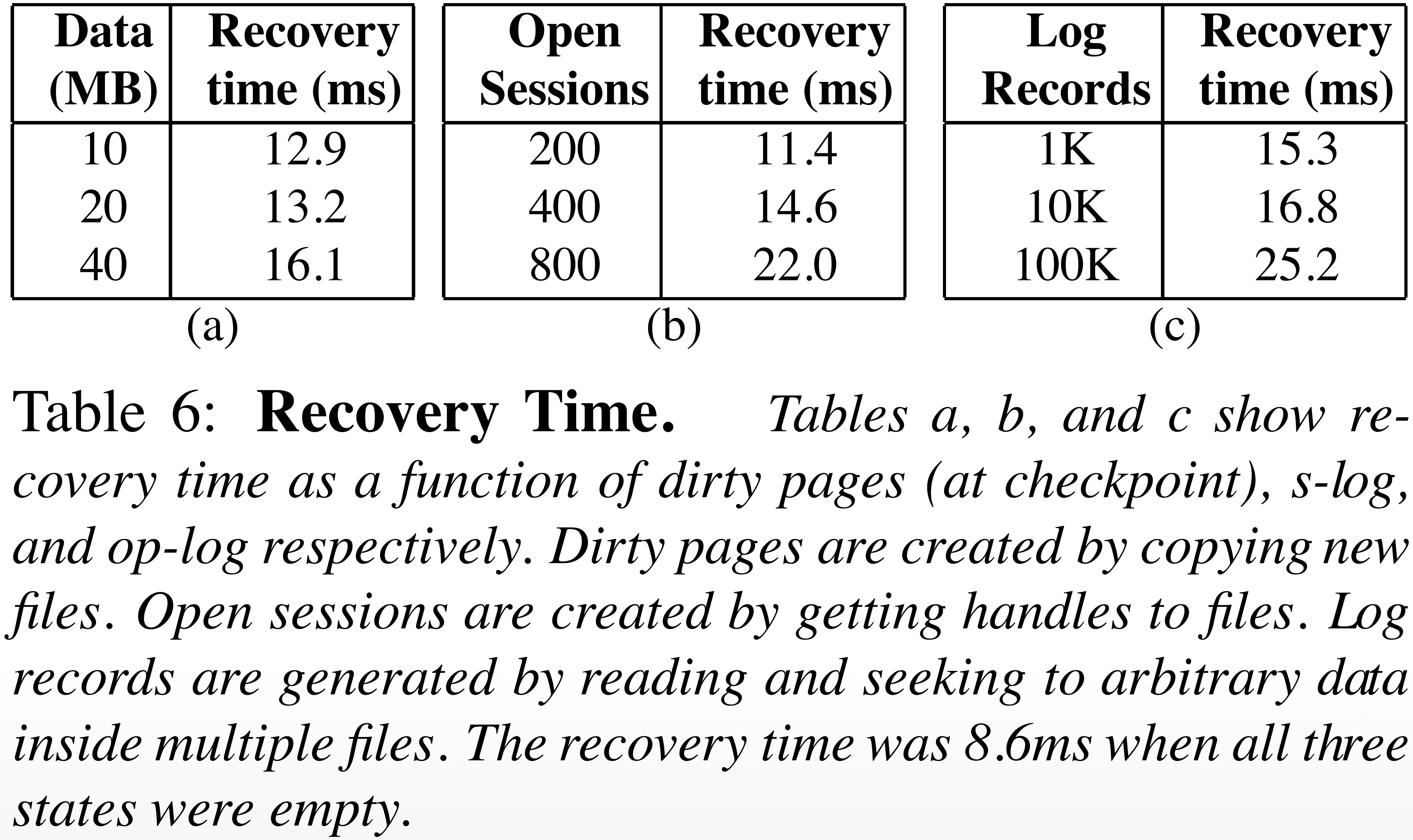
| | ext2 | ext2+ | ext3 | ext3+ | VFAT | VFAT+ |
|---|---|---|---|---|---|---|
| **Benchmark** | | **Membrane** | | **Membrane** | | **Membrane** |
| Seq. read | 17.8 | 17.8 | 17.8 | 17.8 | 17.7 | 17.7 |
| Seq. write | 25.5 | 25.7 | 56.3 | 56.3 | 18.5 | 20.2 |
| Rand. read | 163.2 | 163.5 | 163.2 | 163.2 | 163.5 | 163.6 |
| Rand. write | 20.3 | 20.5 | 65.5 | 65.5 | 18.9 | 18.9 |
| create | 34.1 | 34.1 | 33.9 | 34.3 | 32.4 | 34.0 |
| delete | 20.0 | 20.1 | 18.6 | 18.7 | 20.8 | 21.0 |



| | ext2 | ext2+ | ext3 | ext3+ | VFAT | VFAT+ |
|---|---|---|---|---|---|---|
| **Benchmark** | | **Membrane** | | **Membrane** | | **Membrane** |
| Sort | 142.2 | 142.6 | 152.1 | 152.5 | 146.5 | 146.8 |
| OpenSSH | 28.5 | 28.9 | 28.7 | 29.1 | 30.1 | 30.8 |
| PostMark | 46.9 | 47.2 | 478.2 | 484.1 | 43.1 | 43.8 |

- „[...] in all cases, the overheads were between 0% and 2%“

# RESTART



| Data (MB) | Recovery time (ms) |
|---|---|
| 10 | 12.9 |
| 20 | 13.2 |
| 40 | 16.1 |

(a)

| Open Sessions | Recovery time (ms) |
|---|---|
| 200 | 11.4 |
| 400 | 14.6 |
| 800 | 22.0 |

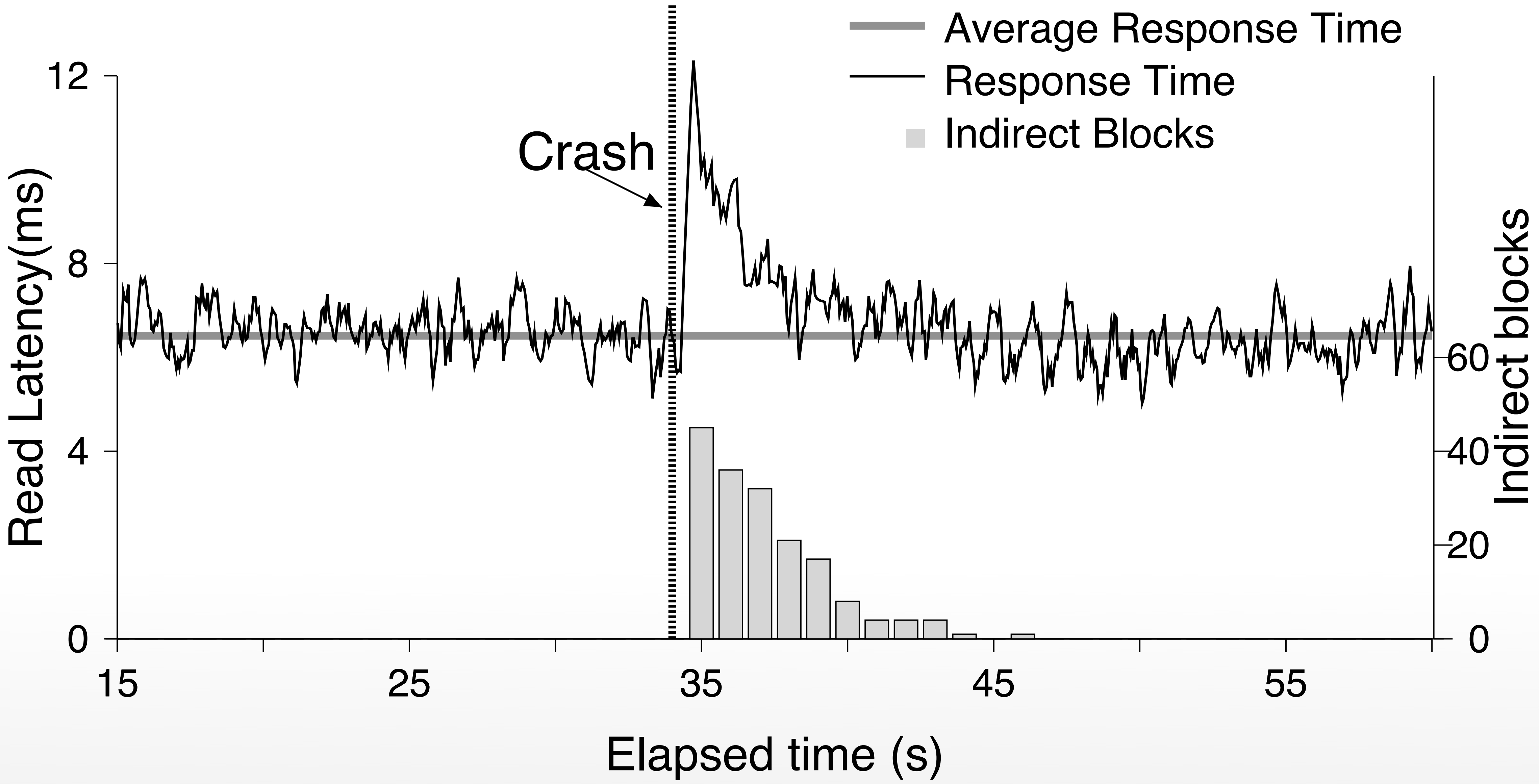
(b)

| Log Records | Recovery time (ms) |
|---|---|
| 1K | 15.3 |
| 10K | 16.8 |
| 100K | 25.2 |

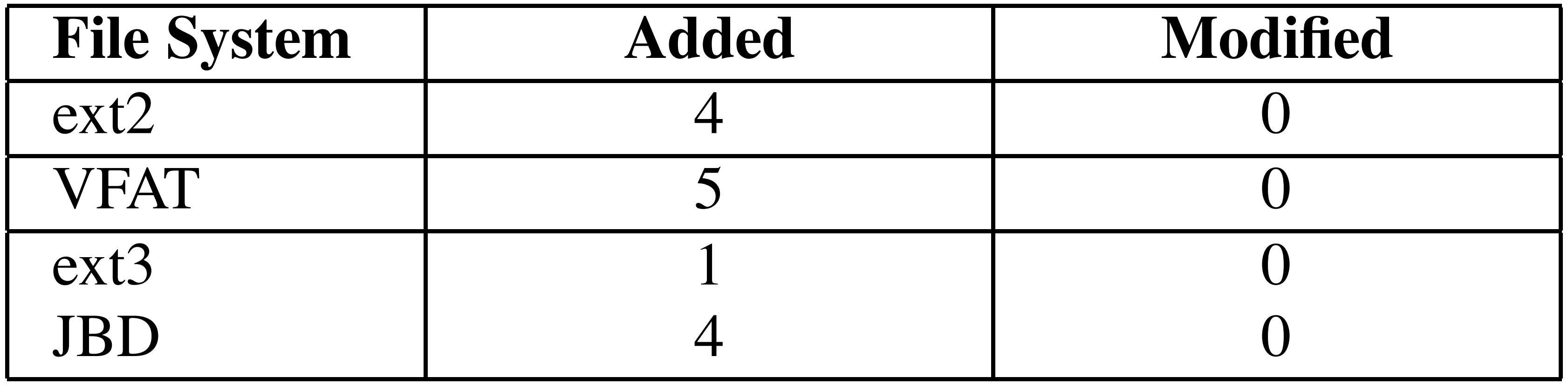
(c)

Table 6: **Recovery Time.** *Tables a, b, and c show recovery time as a function of dirty pages (at checkpoint), s-log, and op-log respectively. Dirty pages are created by copying new files. Open sessions are created by getting handles to files. Log records are generated by reading and seeking to arbitrary data inside multiple files. The recovery time was 8.6ms when all three states were empty.*

# RESTART IMPACT

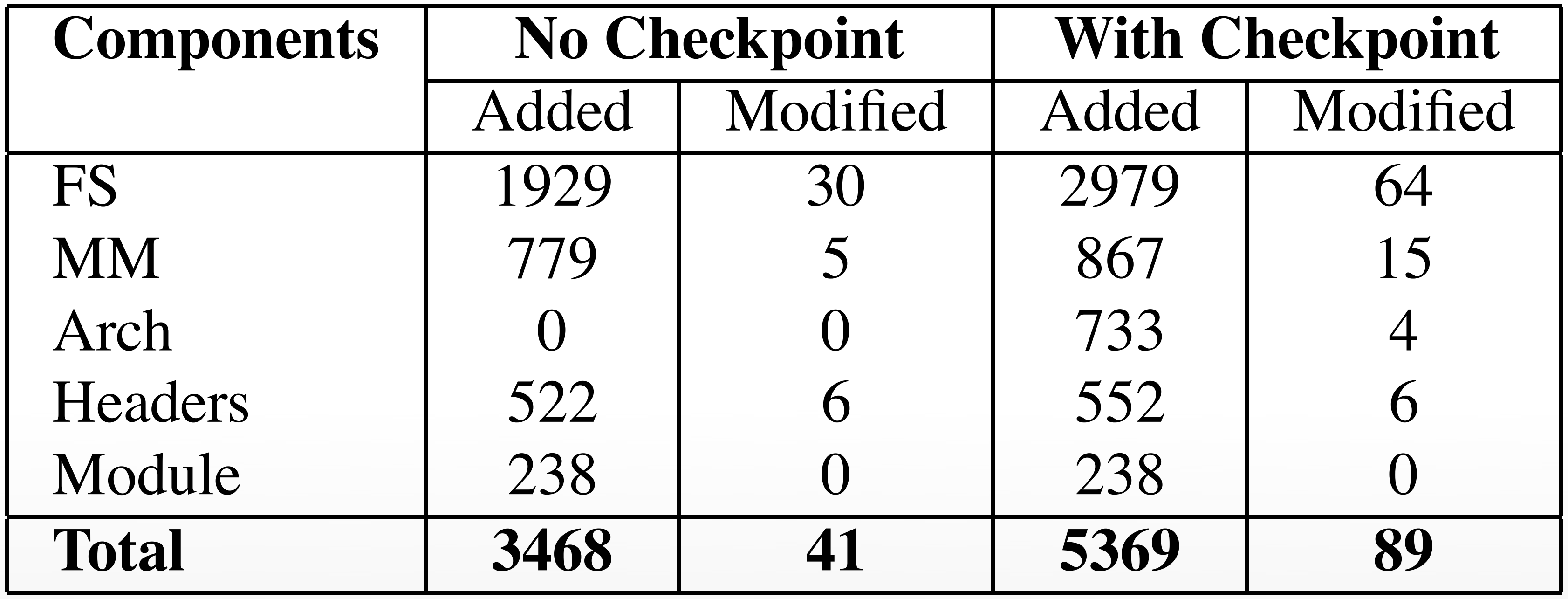


# COMPLEXITY



| File System | Added | Modified |
|---|---|---|
| ext2 | 4 | 0 |
| VFAT | 5 | 0 |
| ext3 | 1 | 0 |
| JBD | 4 | 0 |

Individual File-system Changes



| **Components** | **No Checkpoint** | | **With Checkpoint** | |
|---|---|---|---|---|
| | Added | Modified | Added | Modified |
| FS | 1929 | 30 | 2979 | 64 |
| MM | 779 | 5 | 867 | 15 |
| Arch | 0 | 0 | 733 | 4 |
| Headers | 522 | 6 | 552 | 6 |
| Module | 238 | 0 | 238 | 0 |
| **Total** | **3468** | **41** | **5369** | **89** |

Kernel Changes

# SUMMARY

- „File systems fail“
- Usually they case kernel / app crashes
- Membrane allows them to be restarted
  - Light-weight
  - Stateful
  - Generic
  - Transparent

# DISCUSSION

- Does the fault model actually cover most of the bugs?
- NFS?
- Why is it called „Membrane“?