Membrane: Operating System Support for Restartable File Systems

Swaminathan Sundararaman, Sriram Subramamanian, Abhishek Rajimwale, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, Michael M. Swift
University of Wisconsin, Madison

Slides by Alex Nelson
UCSC CMPS 229, 2010-05-13

File systems crash.

- (And they make a huge clatter.)
- Why they crash:
  - Large code base
  - Updated often

Related work
(Other things crash.)

- Driver fault isolation
  - Nooks: Memory isolation
  - SafeDrive: Inserted assertions
- File system fault isolation
  - CuriOS: Microkernel, protection domains

Weight and state

	Heavyweight	**Lightweight**
Stateless	Nooks	SafeDrive
Stateful	CuriOS	Membrane

Membrane: Lightweight, stateful

- Normal operation:
    - Log file system operations
    - Track file system objects
    - Checkpoint file system state
- On crash:
    - Pause operations, to carefully undo
    - Roll back, to redo

Goals

- For any file system restarter:
  - Fault tolerant
  - Lightweight
  - Transparent
  - Generic
  - Maintain file system consistency

Goals

- For Membrane:
  - Fault tolerant
  - Lightweight
  - Transparent
  - Generic
  - Maintain file system consistency

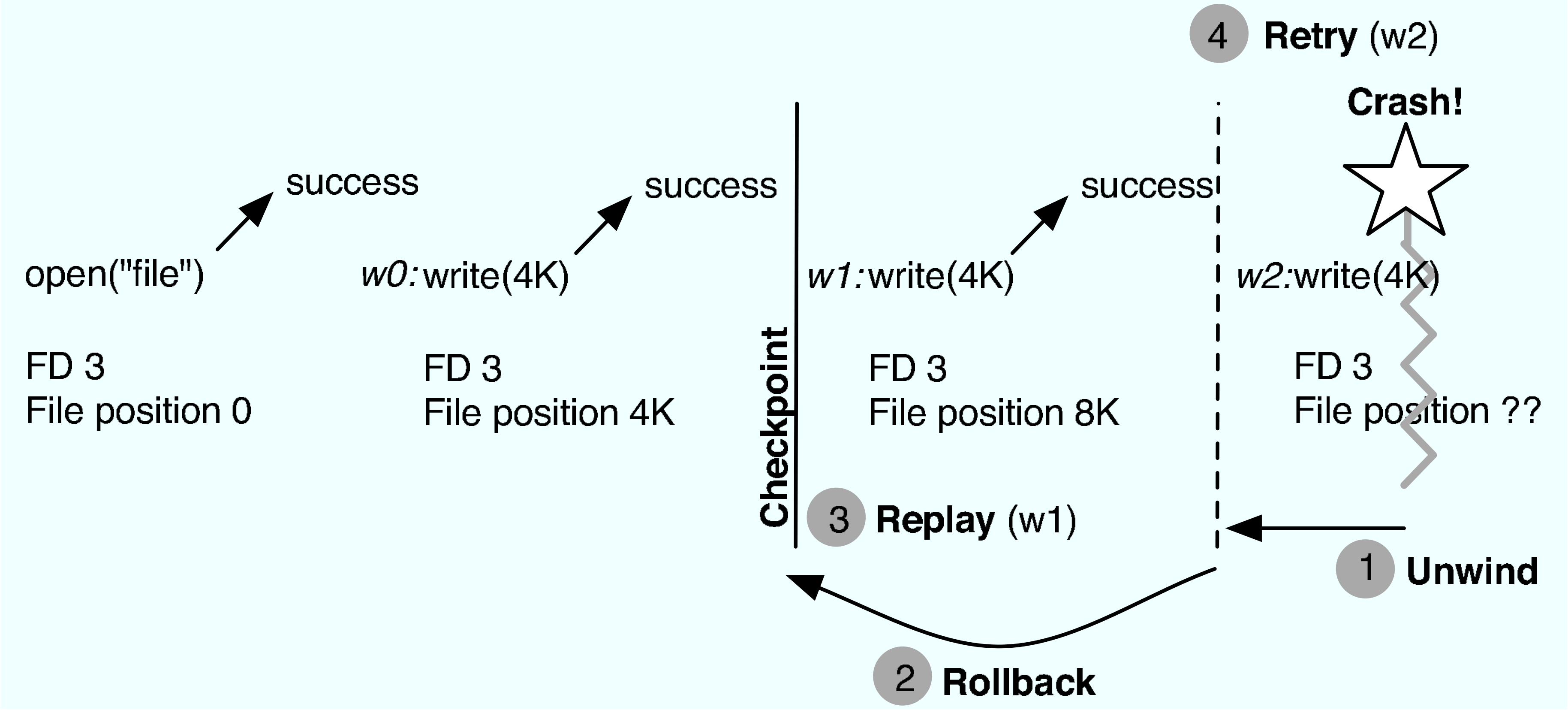
Fault model

- Transient and fail-stop faults targeted
    - Race conditions, environmental factors - recover and restart
- (Assume away other errors)
- Detecting fault revokes file system trust

Membrane overview

- Make checkpoints
- Detect faults
- Roll back and replay

Crash example



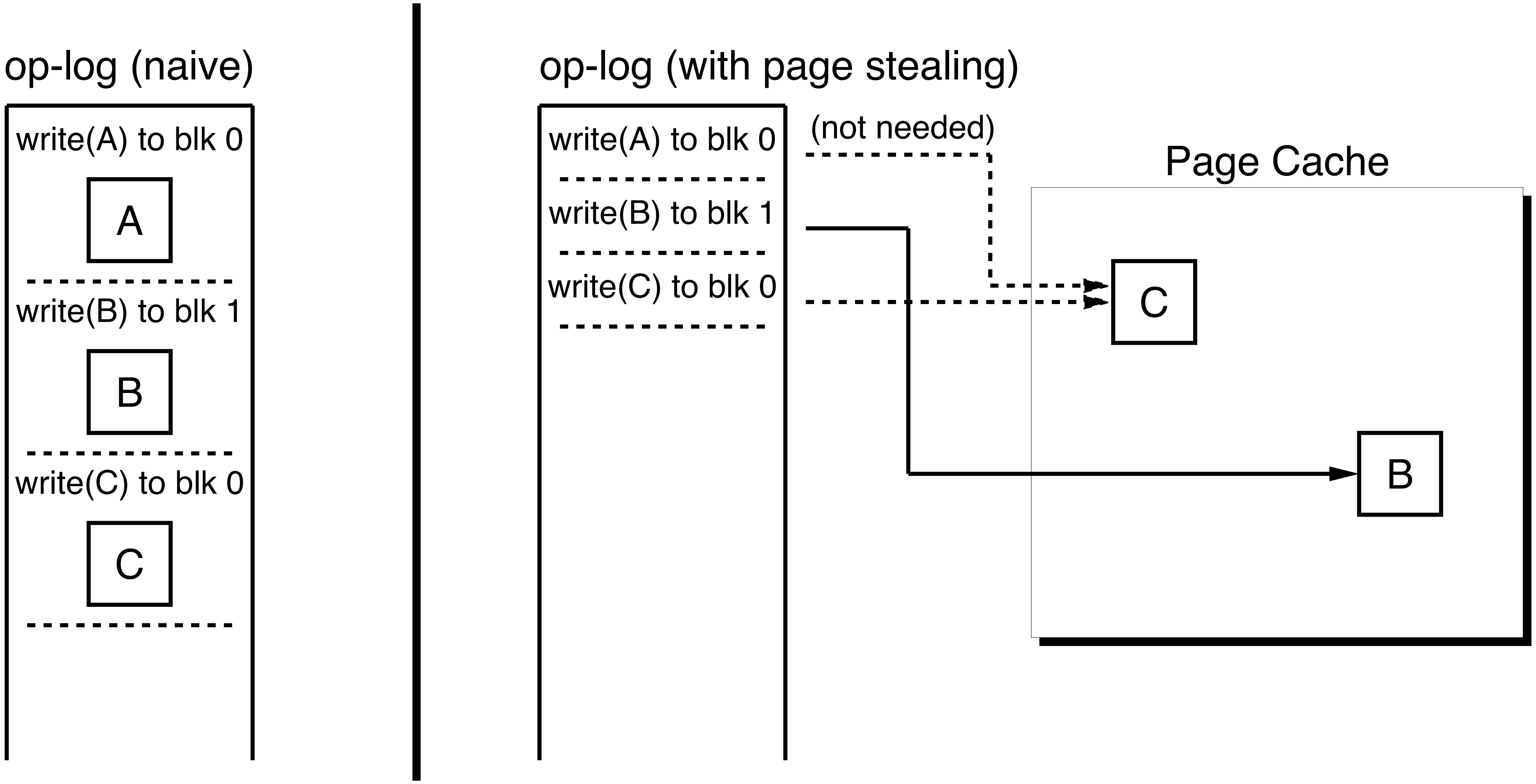
Fault anticipation: Checkpointing

- Inconsistent file system state handlers:
  - Journals, transactions (ext3)
  - Snapshots (WAFL)
  - None (ext2)
- Membrane either inherits, or checkpoints at VFS layer
  - Atomically commits batched operations

Fault anticipation: Tracking state

- Five logs and stacks, for:
  - File system operations: *operation log*
  - Application-visible sessions: *session log*
  - Mallocs: *malloc table*
  - Locks, per thread: *lock stack*
  - Execution state, per thread: *unwind stack*

Low-cost op-logging: Page stealing



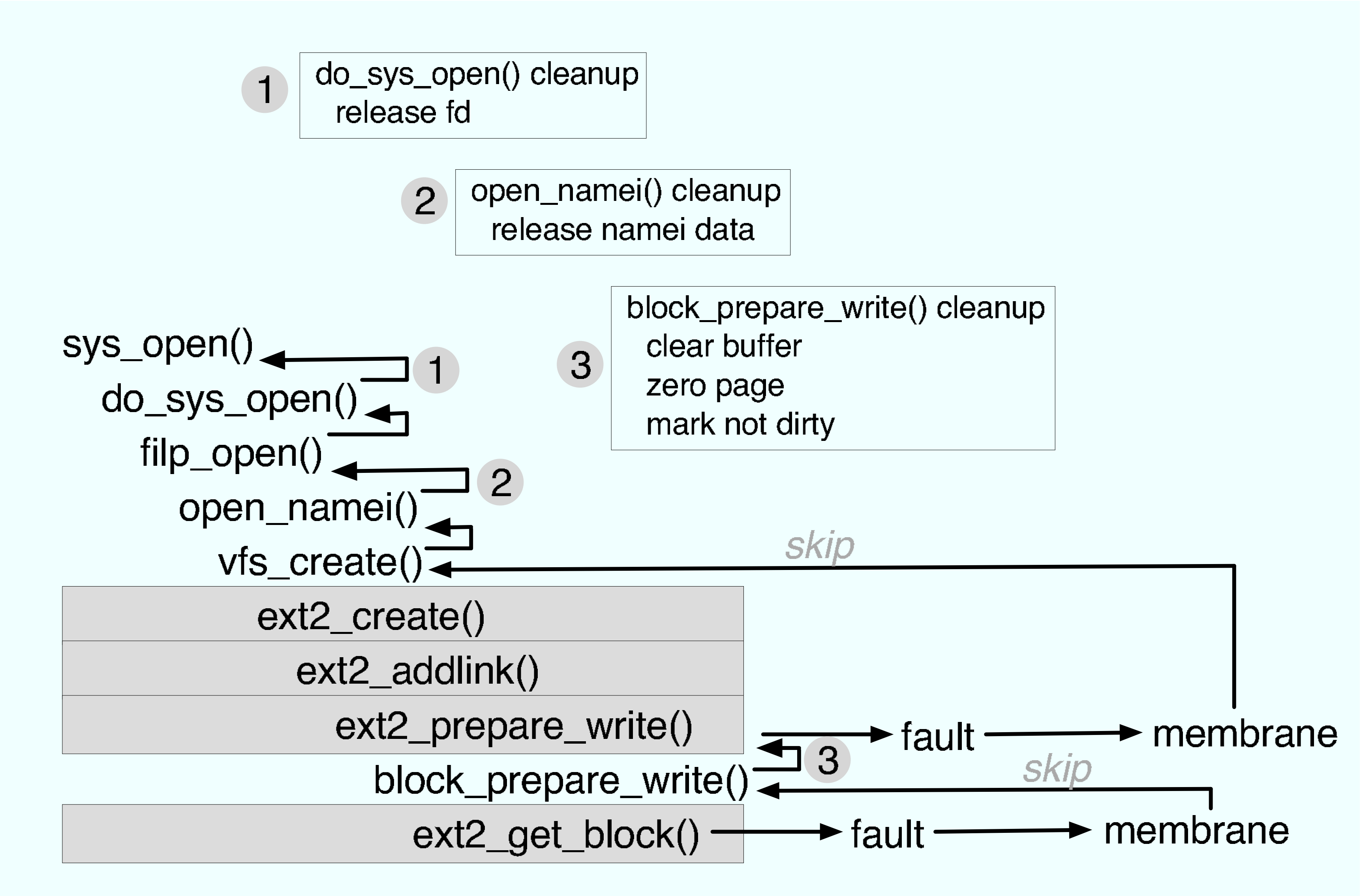
Fault detection

- Hardware: just simple exceptions
- Software: redefine macros (`BUG()`)
- Lightweight kernel/fs boundary wrappers

Fault recovery

- Steps to recover from a fault:
  - Halt executing threads
  - Unwind in-flight threads
  - Commit prior epoch’s dirty pages
  - Unmount file system
  - Remount file system
  - Replay from last consistent state
  - Resume execution

Unwinding:
The Skip/Trust Protocol



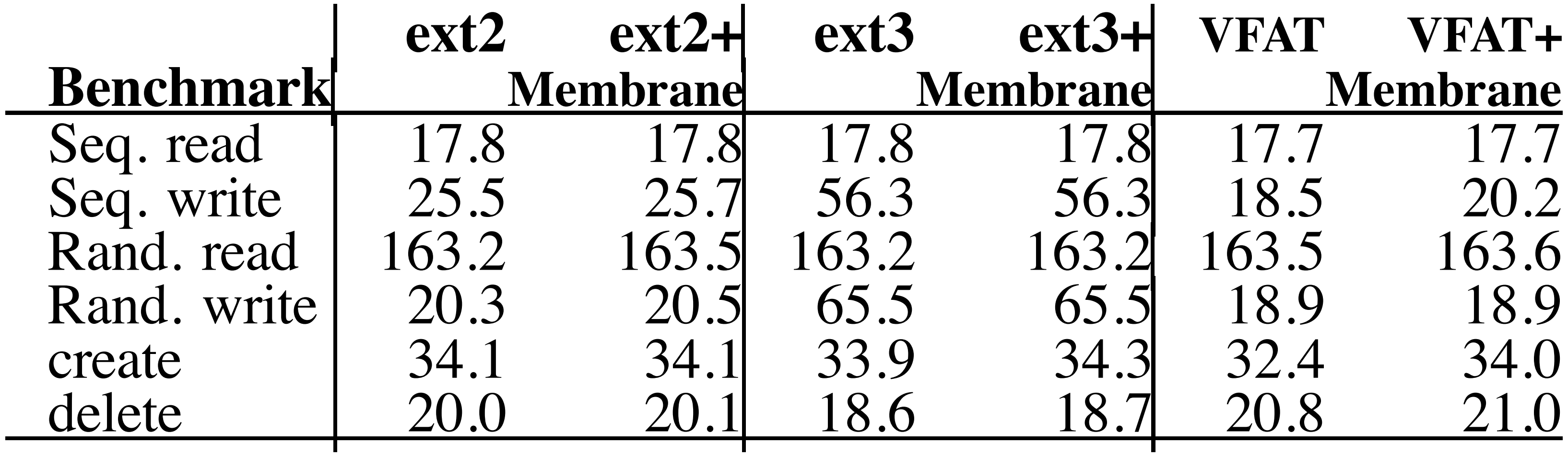
Evaluation

- Platform:
  - Linux 2.6.15, single-core 2.2 GHz
- Categories:
  - Transparency
  - Performance
  - Generality

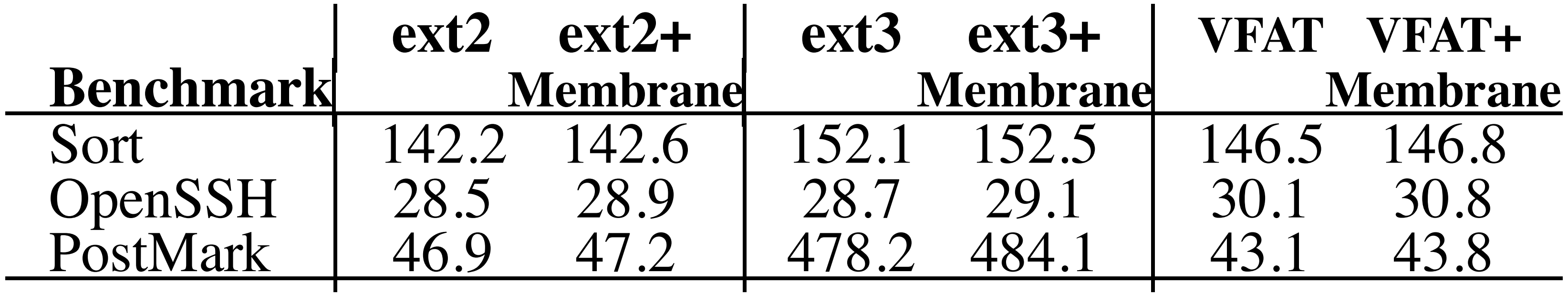
Evaluation: Transparency

- Inject a fault - does an application notice?
- Tested base file system; added boundary; added membrane
- Membrane had perfect performance
  - File system lived, usable
  - Applications didn’t notice
- OS was always usable, even with fs crashes

Evaluation: Performance



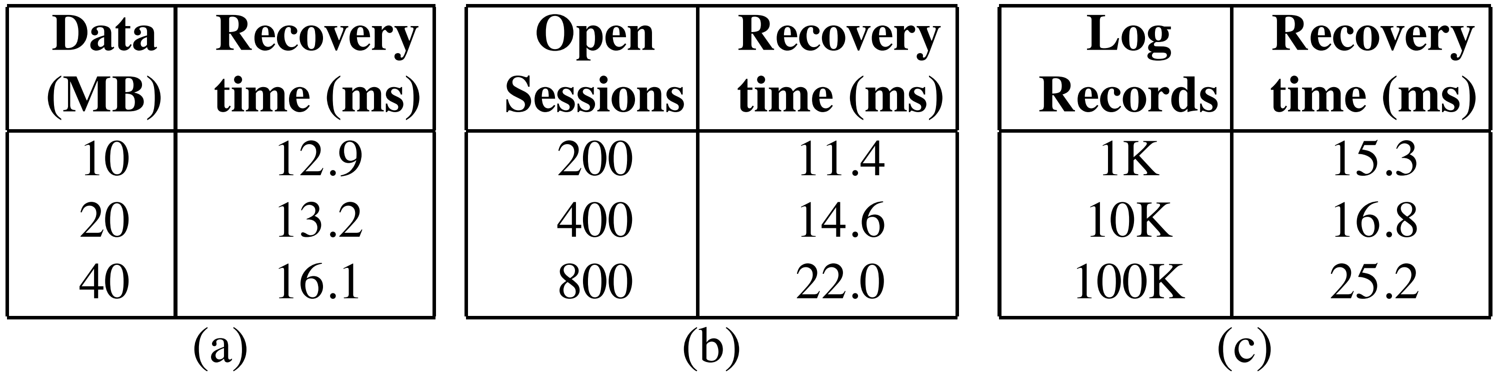
	ext2	ext2+	ext3	ext3+	VFAT	VFAT+
Benchmark		Membrane		Membrane		Membrane
Seq. read	17.8	17.8	17.8	17.8	17.7	17.7
Seq. write	25.5	25.7	56.3	56.3	18.5	20.2
Rand. read	163.2	163.5	163.2	163.2	163.5	163.6
Rand. write	20.3	20.5	65.5	65.5	18.9	18.9
create	34.1	34.1	33.9	34.3	32.4	34.0
delete	20.0	20.1	18.6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

- Micro/macro benchmark overhead: between 0 and 2%.

Evaluation: Performance



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

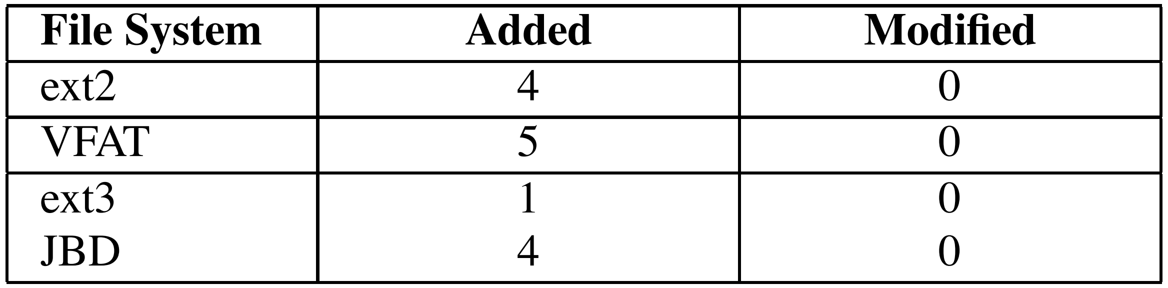
(b)

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

(c)

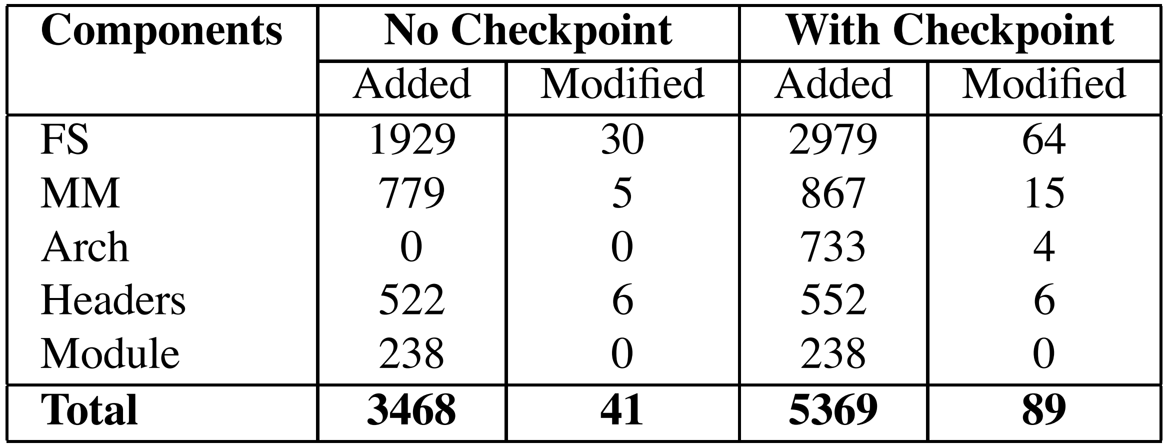
- Time to recover a crashed file system: Sub-linear growth w.r.t. state

Evaluation: Generality



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

- Minimal changes to file systems
- Most kernel additions were error checks or handlers

Conclusions

- File systems fail.
- Membrane: failures aren't even hiccups.
- "...Ship file systems sooner, as small bugs will not cause massive user headaches."

Questions?

- How can this apply to networked file systems?
  - Example: How do I recover from a nearly-completed append?
- Applicability: Does anything else touch as much of the kernel as file systems?