SDC
STORAGE DEVELOPER CONFERENCE
SNIA SANTA CLARA, 2014

Failure-atomic msync(): A Simple and Efficient Mechanism for Preserving the Integrity of Durable Data

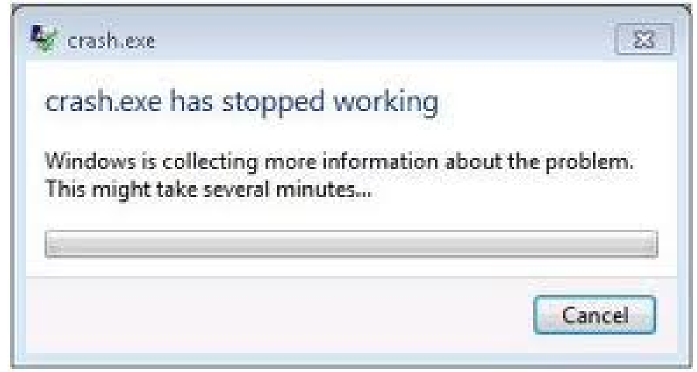
Stan Park

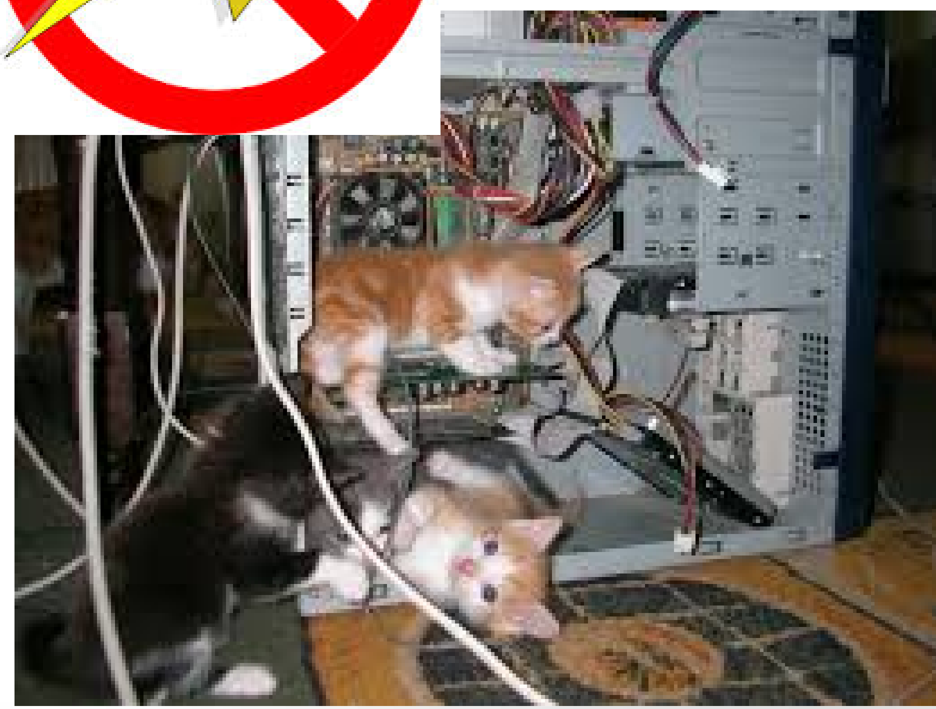
HP Labs

Joint work with:

- Terence Kelly, HP Labs
- Kai Shen, University of Rochester

Failures Happen





Solutions?

- Inadequate: FS journaling (self-centered, no user-accessible interfaces)
- Bloated or awkward, impractical: NoSQL, relational DBMS, atomic rename
- Homebrew: not reusable, potentially buggy

Failure-atomic msync() interface

- Allow the programmer to evolve durable state failure-atomically, all or nothing, always consistent
- Simple interface
  - mmap(MAP_ATOMIC)
  - msync(MS_SYNC)

Failure-atomic msync() interface

- More POSIX flags
  - MS_INVALIDATE: “Invalidate cached data”
  - MS_ASYNC: “Perform asynchronous writes”
- Implementation-specific semantics ⇨ ignored in Linux!

Failure-atomic msync() ⇨ Harmony with POSIX

- MS_INVALIDATE: Rollback functionality for failed transactions, programmer changes mind
- MS_ASYNC: Decouple blocking and atomicity; msync() is the interface for declaring intention

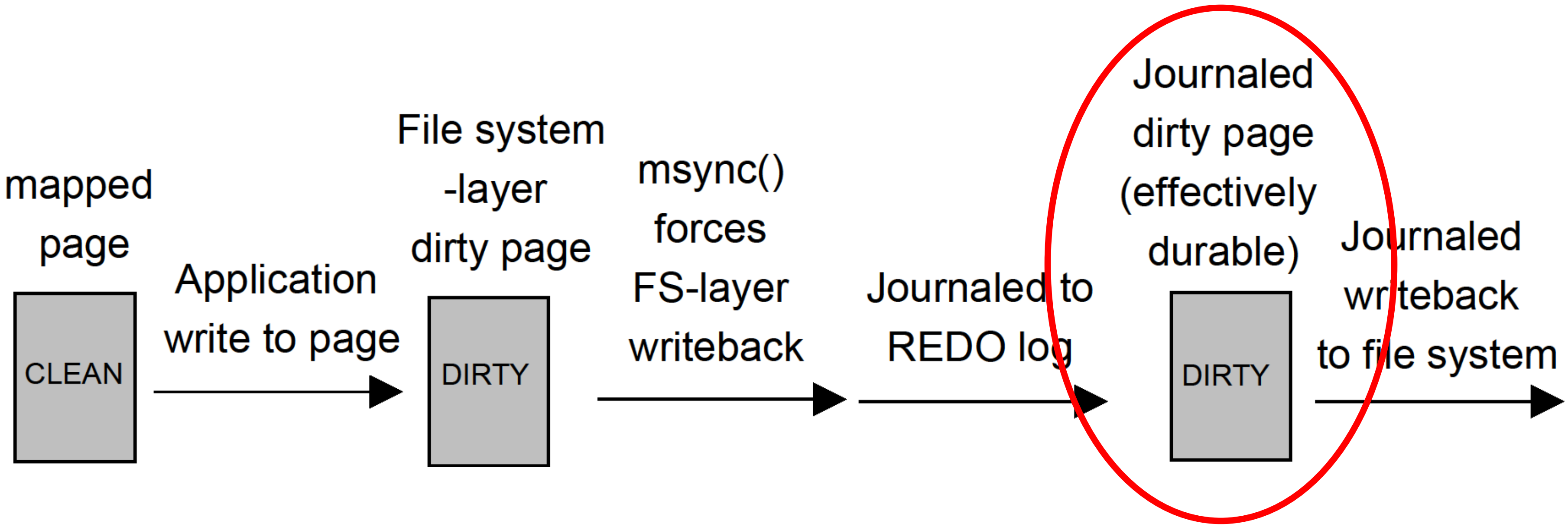
Failure-atomic msync()

- Two logical goals
  - Keep state consistent between msync()s
  - Keep state consistent during msync()s
- Implementation path
  - Prevent non-explicit writeback
  - REDO/UNDO Journaling, shadow copy

Failure-atomic msync() via journaling

- Journal is a redo log
- Well-defined, checksummed journal entries
- Write file updates to journal; out-of-place update keeps file consistent until full update transaction is durable
- Apply journal entries to FS: eager vs async

Eager vs Async Journaled Writeback



- Eager w/b flushes all FS-layer dirty pages
- Async w/b distinguishes between unjournaled and journaled dirty pages; defers non-critical work

Failure-atomic msync() implementation: ext4-JBD2

- Extend VFS interface
  - writepage: one page at a time
  - writepages: multiple contiguous pages
  - writepagesv: multiple noncontiguous pages in a range
- Support richer journaling in the FS
  - Failure-atomic: Encapsulate all work (multiple, non-contiguous block updates) in a single handle -> single JBD2 transaction

Failure-atomic msync() caveats

- msync() size: 2MB with default (128MB) journal, at least 16 MB with 3GB journal
- Isolation in multi-threaded code
- Memory pressure
  - Dirty pages may exceed physical memory, can't be journaled or written to FS until msync()
  - Use swap

Case Study: Persistent Heap and C++ STL

- Persistent heap based on failure-atomic msync(): < 200 LOC
- Persistent heap exports malloc()/free(); replace STL allocator: <20 LOC
- Programmer can utilize full power of STL in a familiar manner with persistent, failure-atomic properties

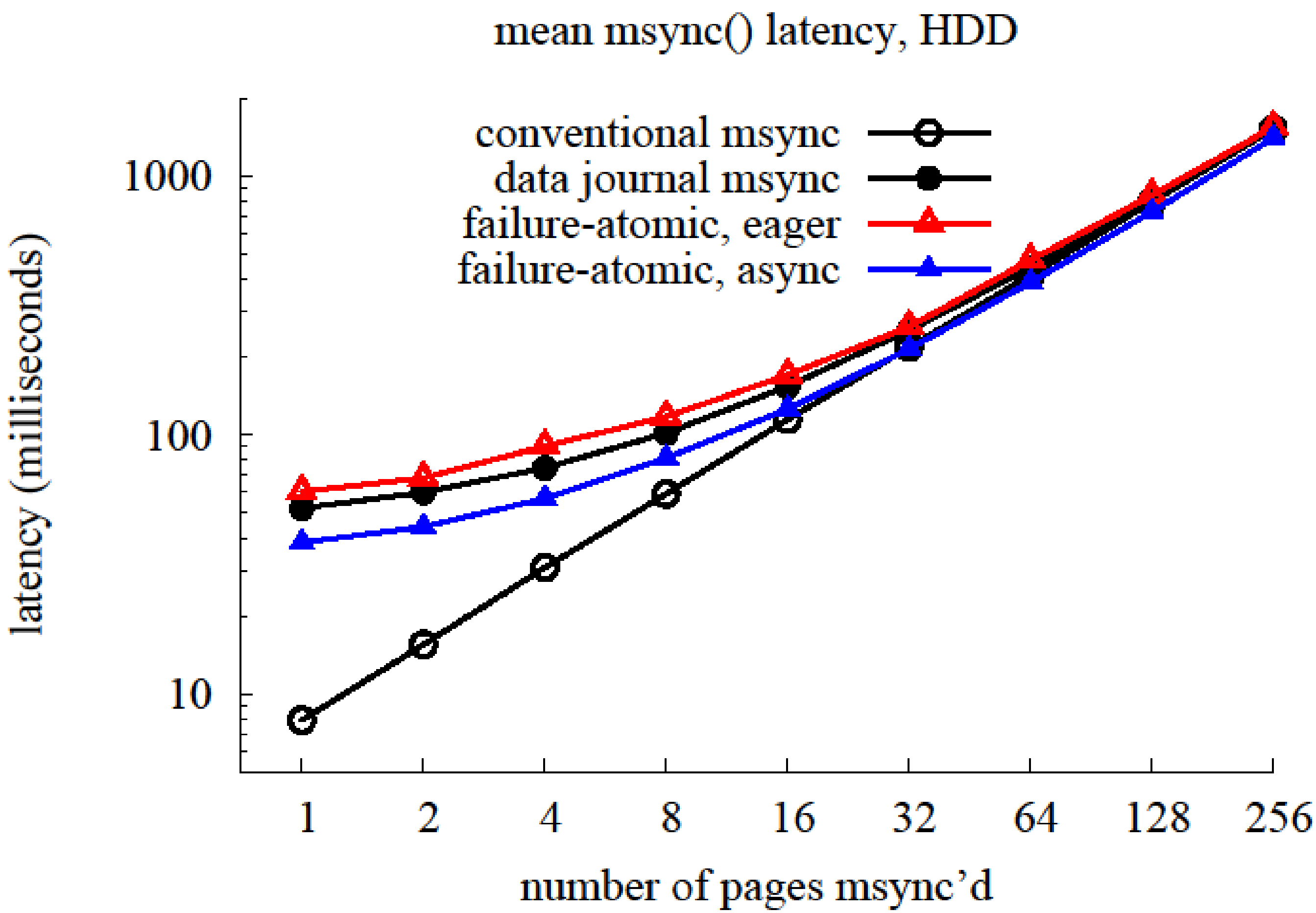
Case Study: Tycoon Key-Value Server

- Utilizes memory mapped region for data structures
- Two data integrity modes:
  - Synchronize: conventional msync() call; does not provide failure-atomicity
  - Transaction: utilizes undo logging; expensive, synchronous double write
- Retrofitting is simple: add MAP_ATOMIC flag to mmap() call; msync() is called as normal
- LOC changed: 1

Evaluation: Storage reliability

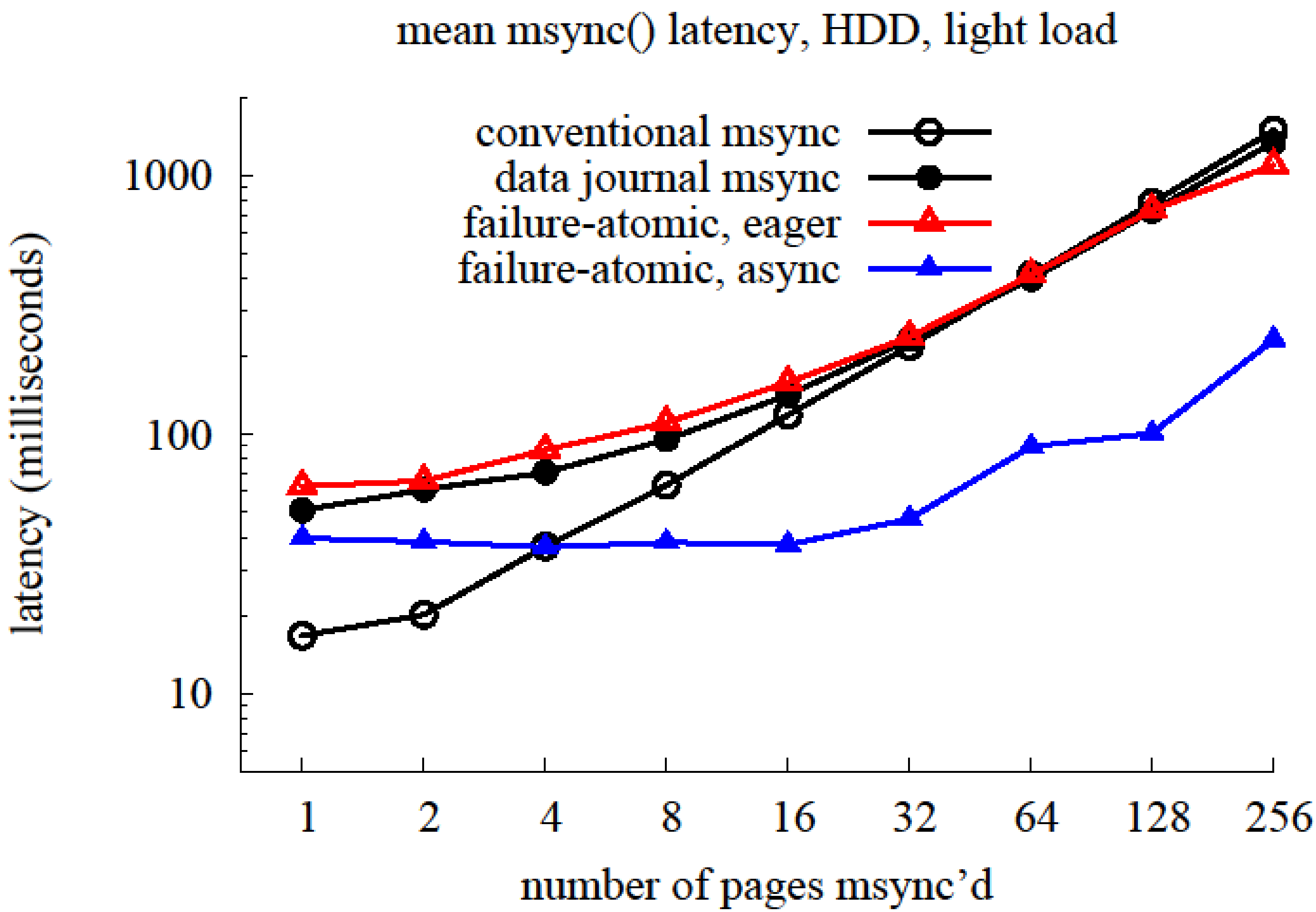
- 6 SSDs, one HDD
  - Known, checkable set of writes issued
  - Cut power to entire machine
  - Pick up the pieces and start over
- Hundreds of power faults later
  - Two SSDs, one HDD
  - Not all devices behave well under power loss (Zheng, et al., FAST '13)

Evaluation: Microbenchmarks



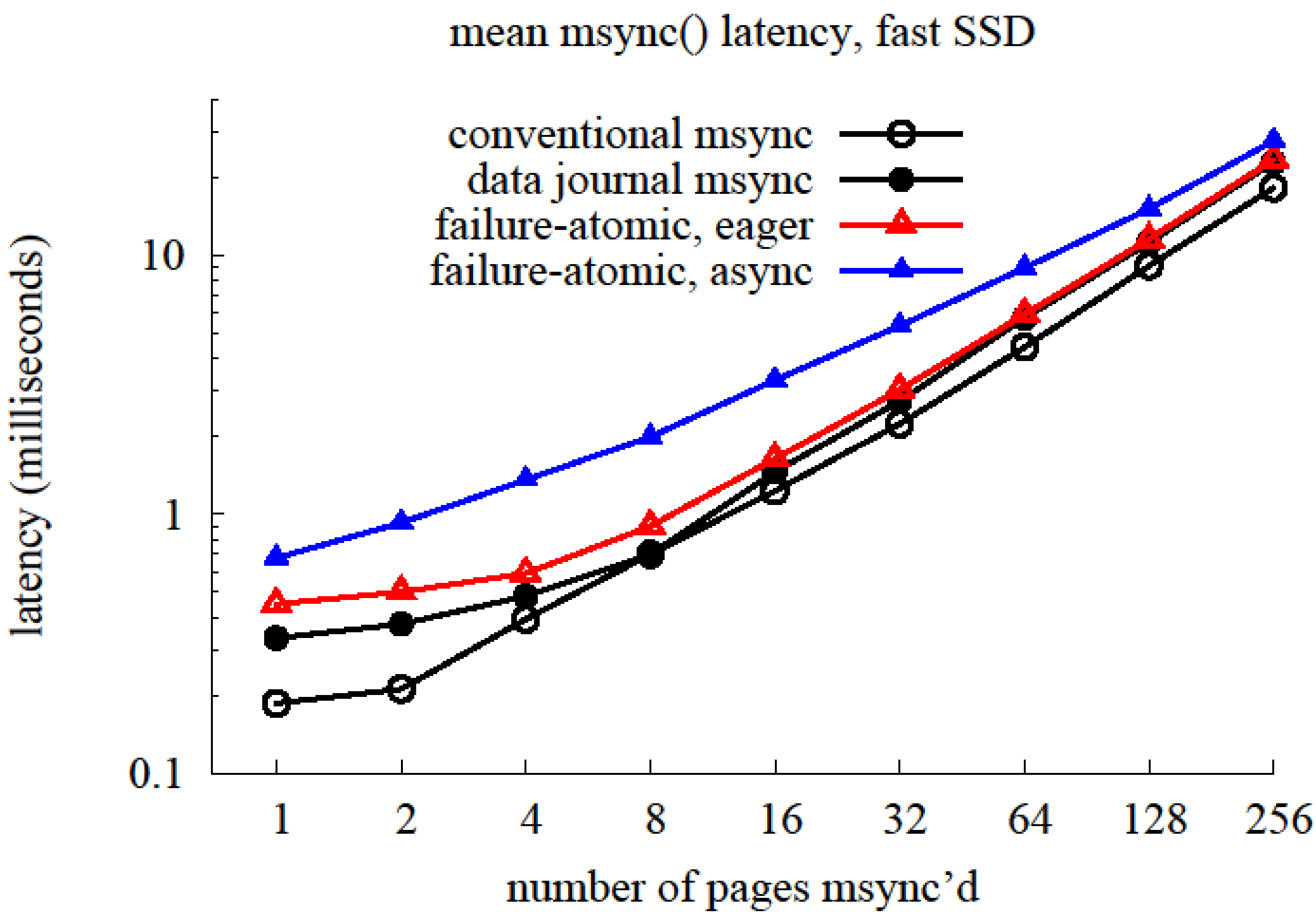
Overheads diminish as msync() size increases

Evaluation: Microbenchmarks



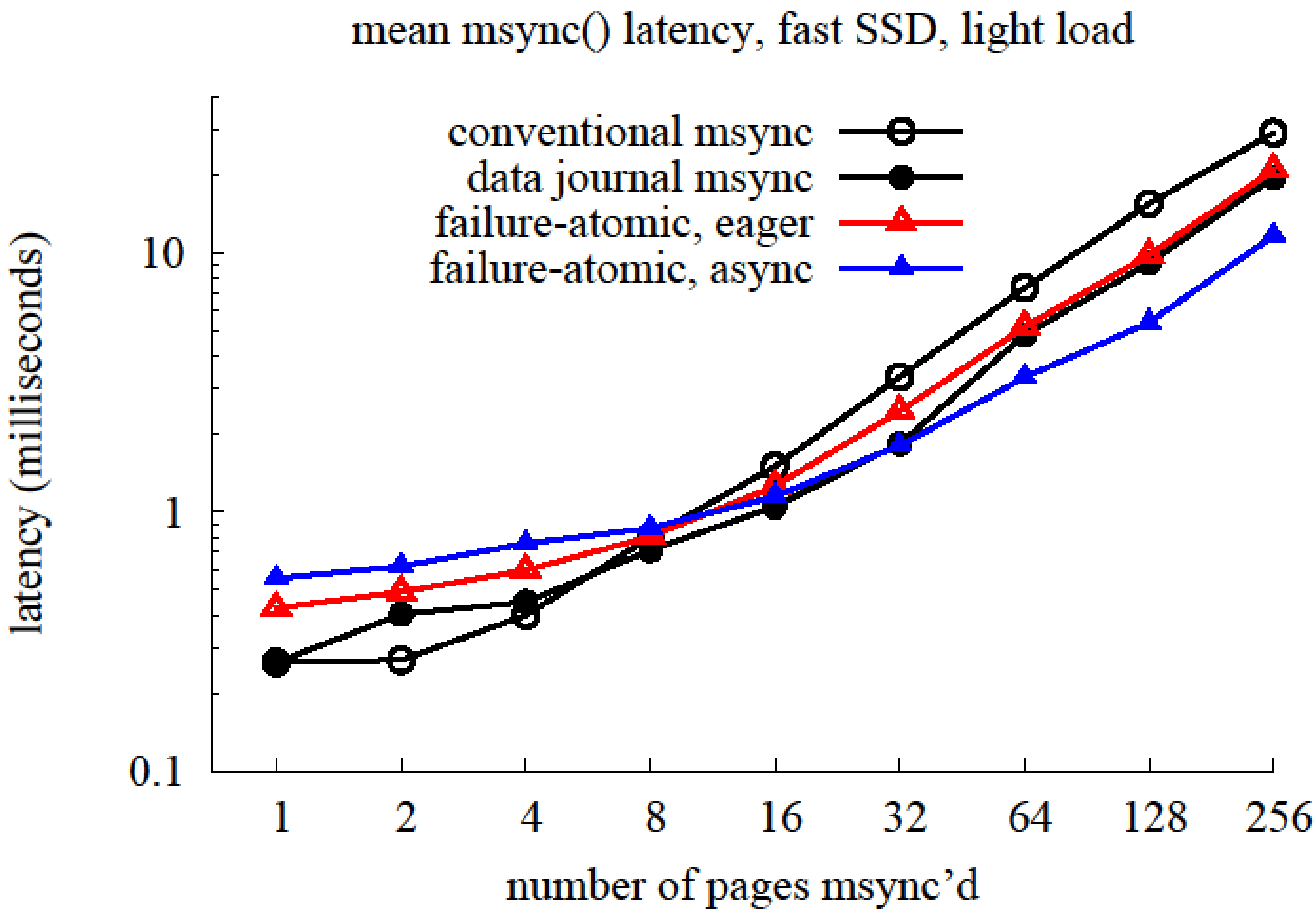
Under light load, async writeback makes failure-atomic msync() superior beyond 4 pages

Evaluation: Microbenchmarks



Again, overheads diminish as msync() size increases; on certain SSDs, eager writeback is better than async.

Evaluation: Microbenchmarks

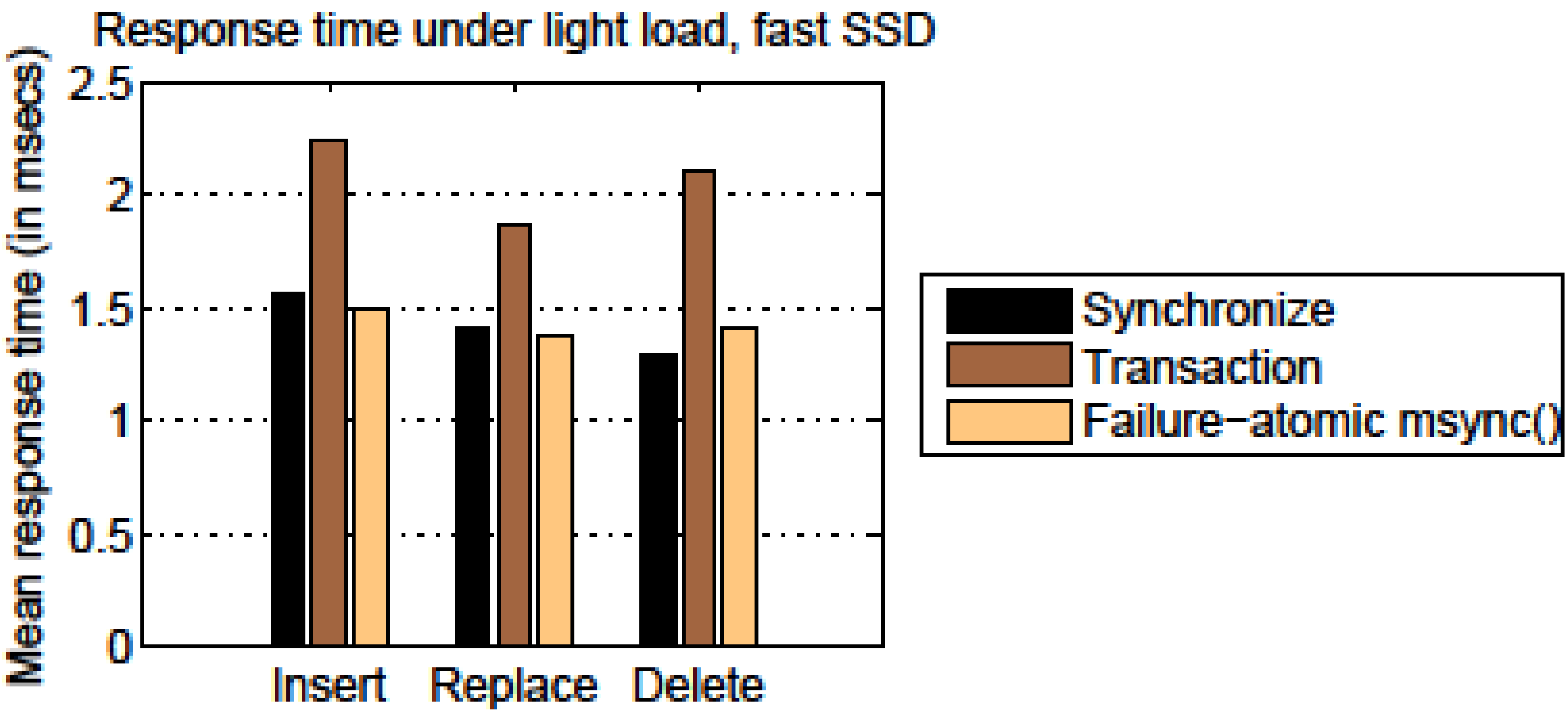


Under light load, async writeback makes failure-atomic msync() superior beyond 8 pages

Evaluation: Persistent Heap and C++ STL

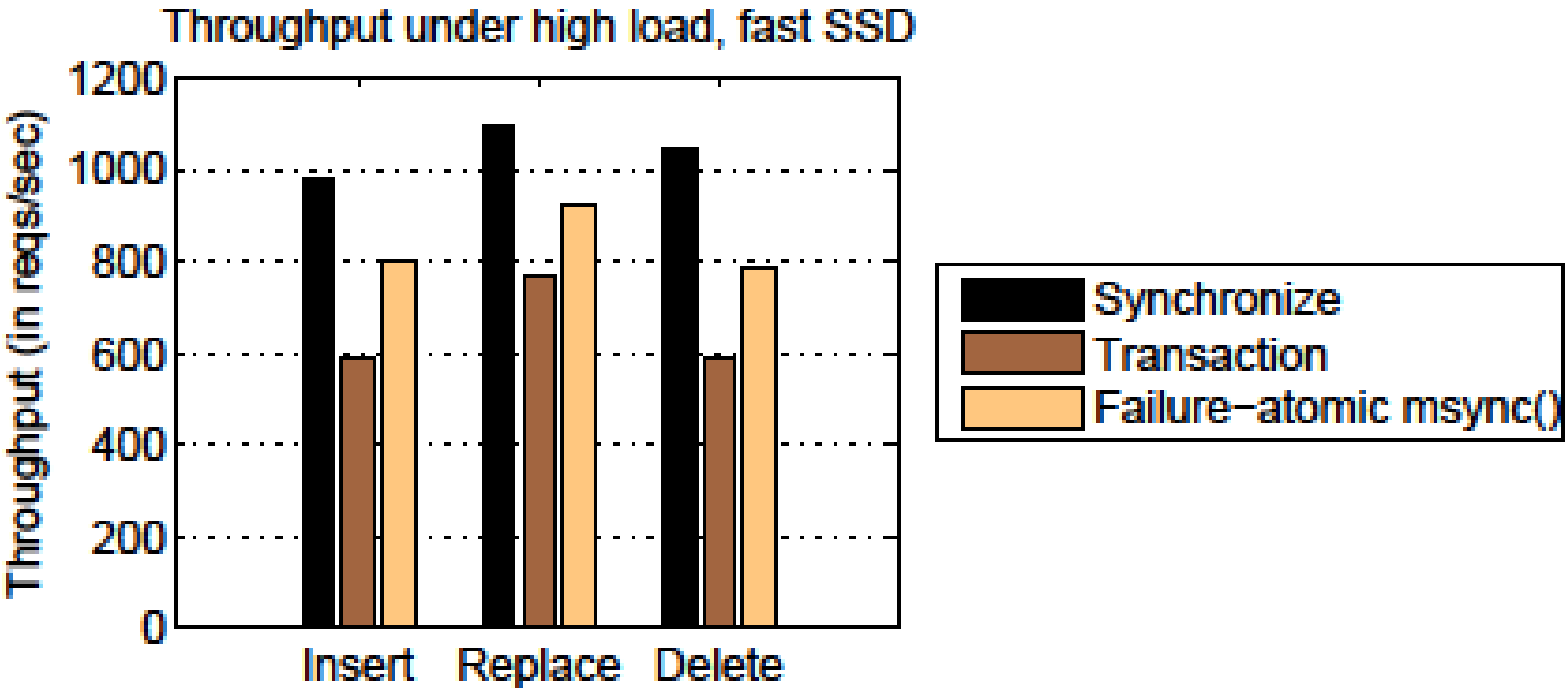
Response time (ms)	hard disk (HDD)			solid-state (fast SSD)		
	thinktime zero			thinktime zero		
	insert	replace	delete	insert	replace	delete
STL <map> + failure-atomic msync	36.538	37.372	45.017	0.586	0.581	0.690
Kyoto Cabinet	146.763	54.434	92.951	1.488	0.579	0.942
SQLite	117.067	100.089	84.817	1.229	1.128	1.047
LevelDB	19.385	19.669	8.645	0.212	0.220	0.116

Evaluation: Tycoon Key-Value Server



Easy to retrofit applications: Changed 1 LOC
Transaction reliability with Synchronize cost

Evaluation: Tycoon Key-Value Server



Easy to retrofit applications: Changed 1 LOC
Transaction reliability with Synchronize cost

Evaluation: Cost of Data Reliability

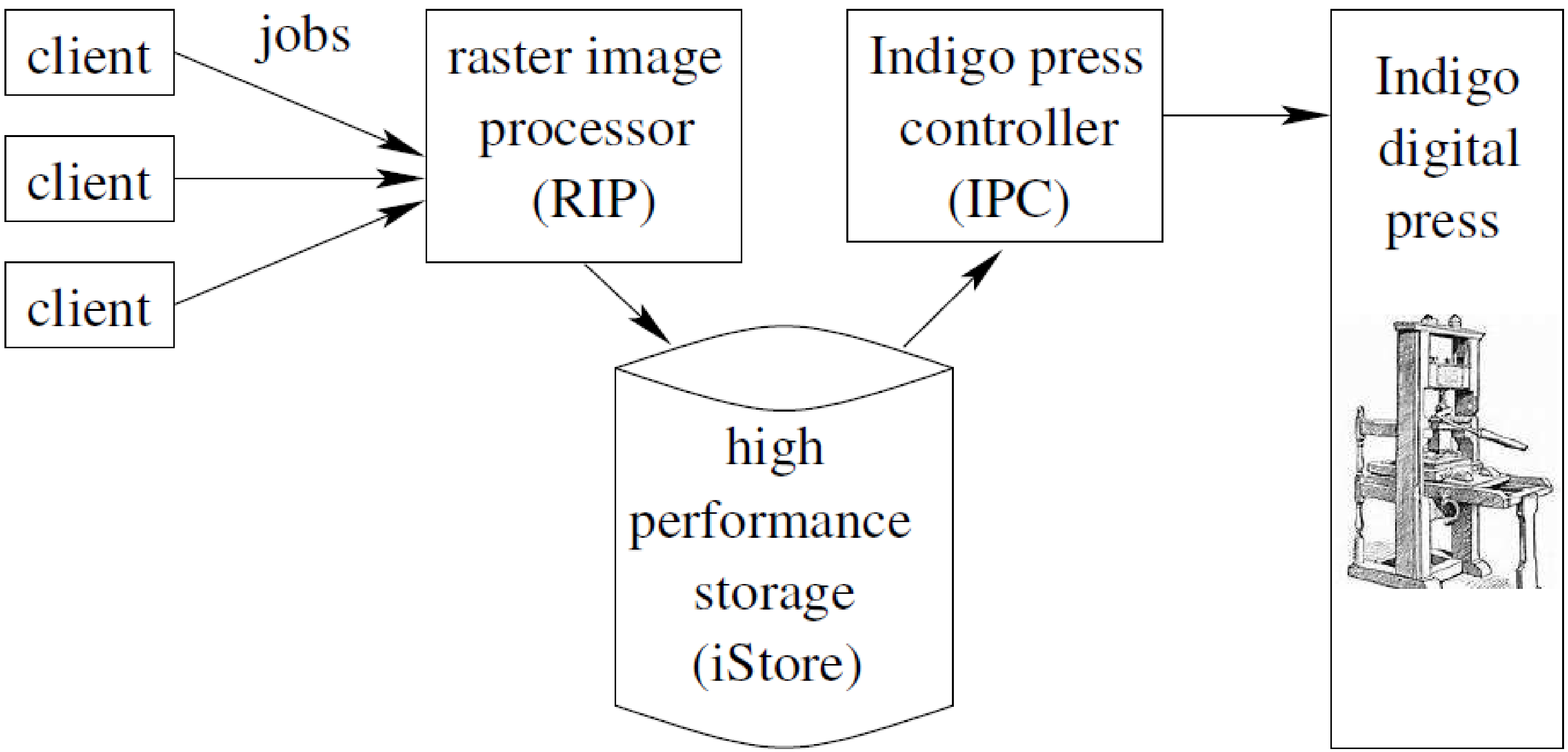
	Response time (ms)			Throughput (req/s)		
	insert	replace	delete	insert	replace	delete
no-sync	0.47	0.45	0.44	6646	6772	7406
failure-atomic msync()	1.49	1.38	1.41	805	919	784

Versus a no-sync Tycoon, adding reliable I/O incurs 3x response time increase, 9x throughput reduction

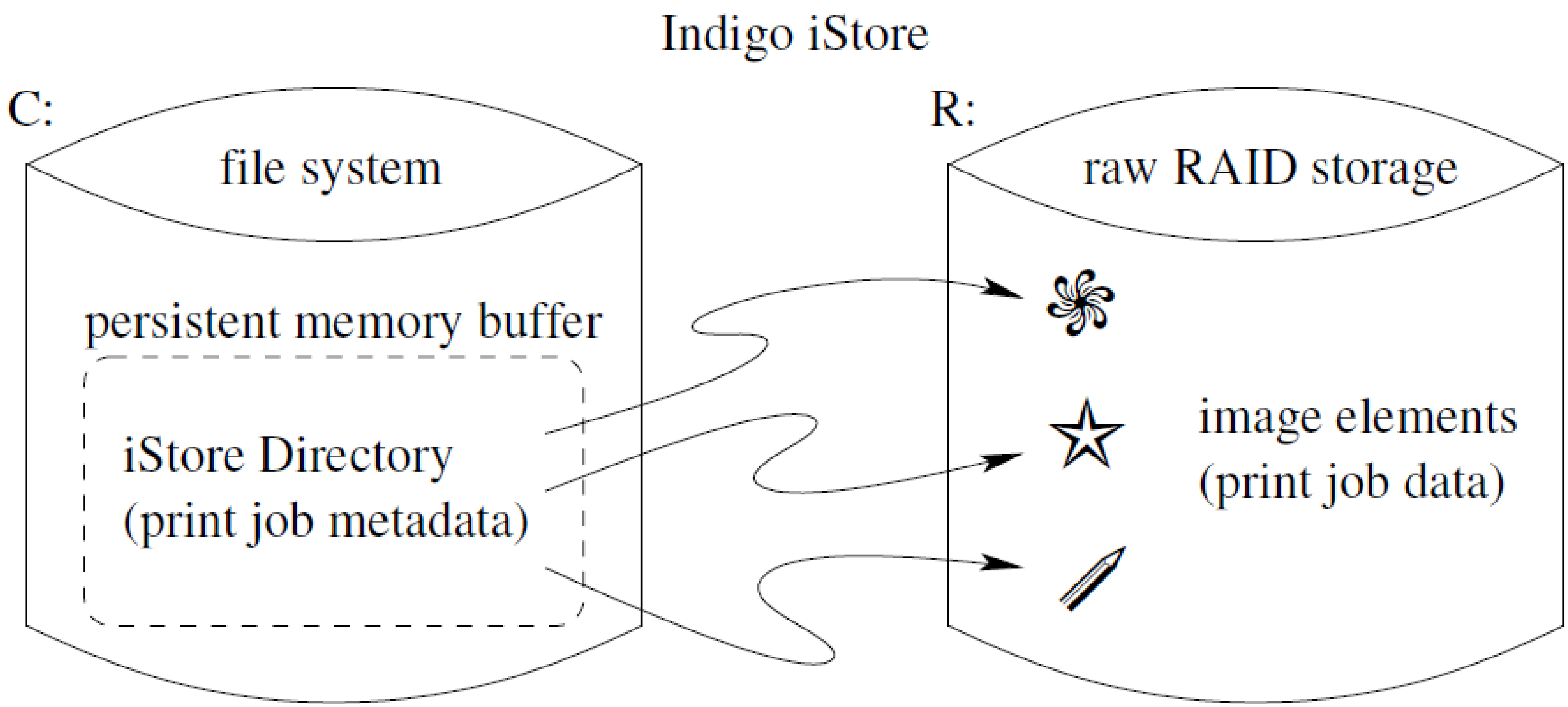
HP Indigo Printing Presses

- High-volume printing press, $500K+
- Job flow streamlined for failure-free operation
- Power outages, crashes corrupt in-progress job data
- Recovery can take days and technician support!

HP Indigo Printing Presses



HP Indigo Printing Presses



HP Indigo Printing Presses

- 425 crashes later, recovery succeeded every time
- Recovery time reduced from days to minutes
- Fortified iStore currently deployed in production presses

Related Work

- TxOS: doesn't support msync()
- MS Windows Vista: "extremely limited developer interest. . . due to its complexity and various nuances"
- Rio Vista: protect against power losses (via UPS) and software corruption
- RVM: similar in spirit, more complex interface
- Stasis: storage framework implementing general I/O transactions

Summary: Failure-atomic msync()

- A simple solution to an exact need
  - Easy for programmers to use
  - Natural foundational abstraction for building higher layers of abstraction
  - Retrofitting applications is simple
- Admits multiple implementations, flexibility
- Safe and efficient across disk and SSD
  - Comparable to or outperforms conventional, unsafe msync() by as few as 4-8 pages
  - Adding reliability can be affordable by leveraging newer SSDs and emerging storage