Enabling System Transactions

via Lightweight Kernel Extensions

R.P. Spillane, S. Gaikwad. M. Chinni, C.P. Wright, E. Zadok

Stony Brook University

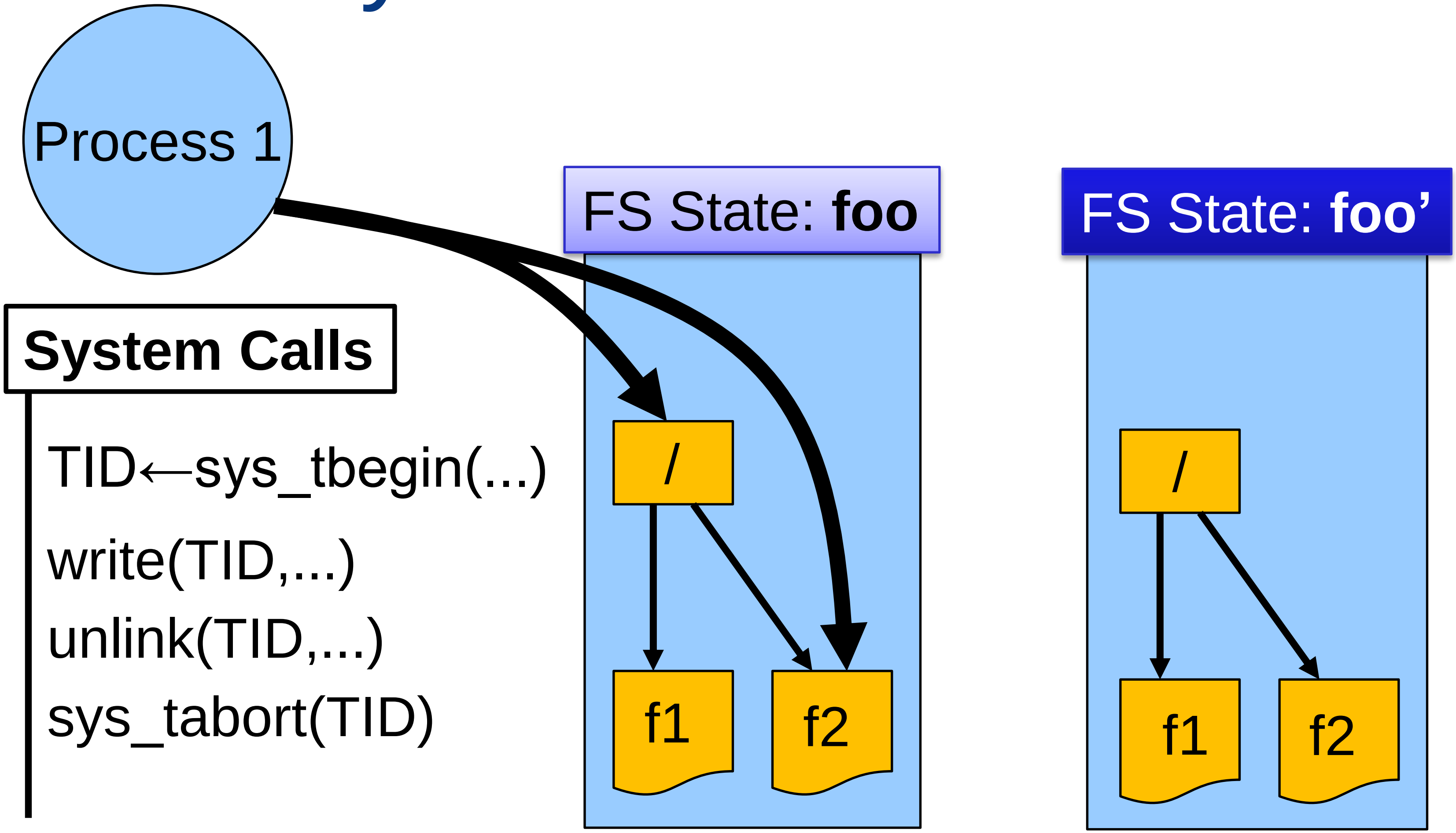
http://www.fsl.cs.sunysb.edu/

STONY BROOK
STATE UNIVERSITY OF NEW YORK

Summary

- What is the design complexity of system transactions implemented in the VFS?
  - Low
    - 100 lines of code added to page writeback
    - 4000 lines of module code (log implementation)
- What is the performance?
  - Valor: 35% overhead on top of theoretical best, compared to…
  - 104% overhead for an efficient user-level alternative

System Transaction



System Calls

TID←sys_tbegin(...)

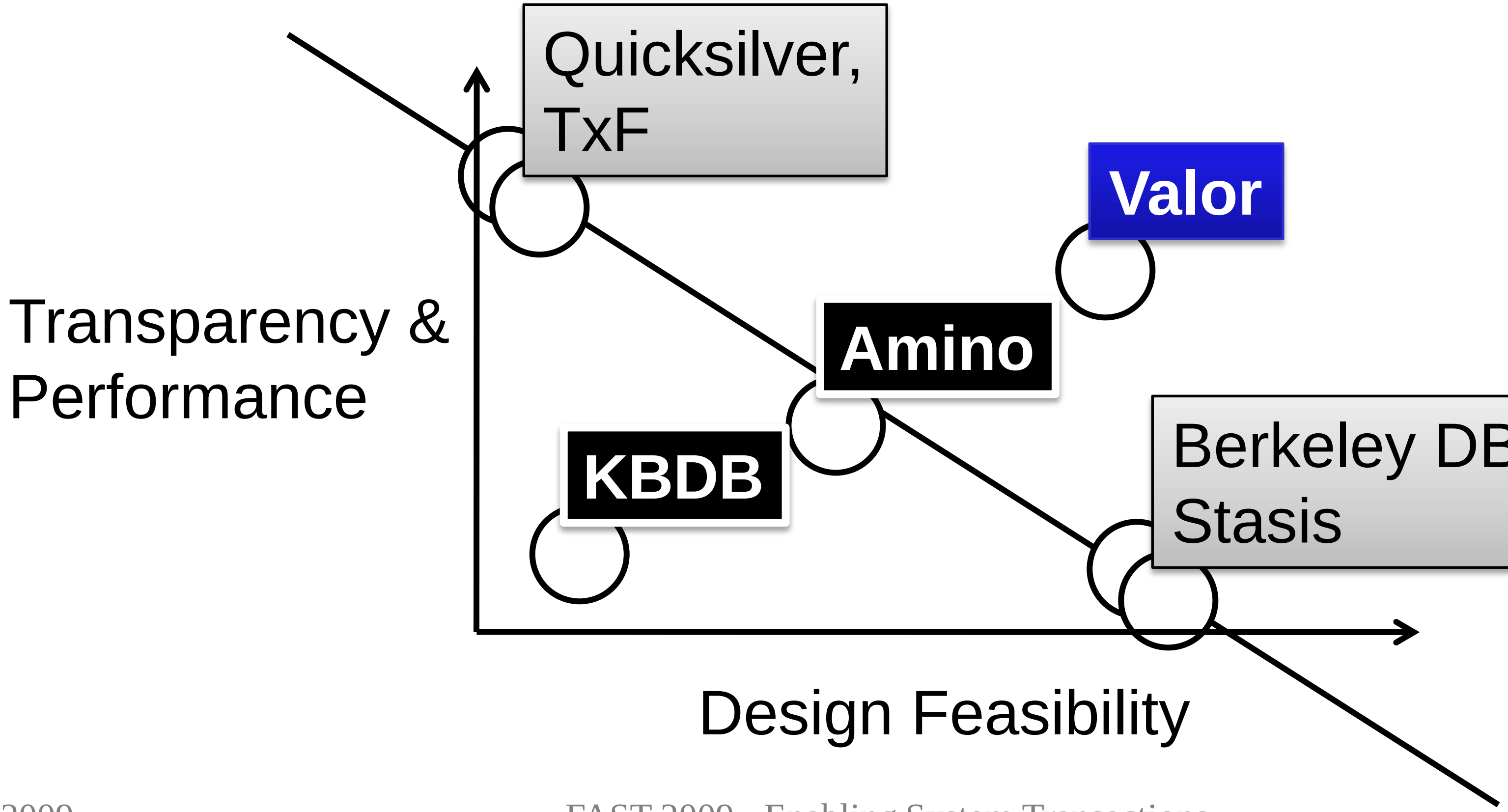
write(TID,...)

unlink(TID,...)

sys_tabort(TID)

The Design Spectrum

- Valor side-steps the traditional trade-off by working with the Kernel's page cache in a general way.



Valor’s Process Txn Model

- Transactional Model
  - Supported Operations:
    - dirtying a page
    - appending to a file, modifying an inode
    - modifying a directory
  - Locking:
    - directory locks, inode locks
    - page range locks for overwrites
    - intent locks for directory renames

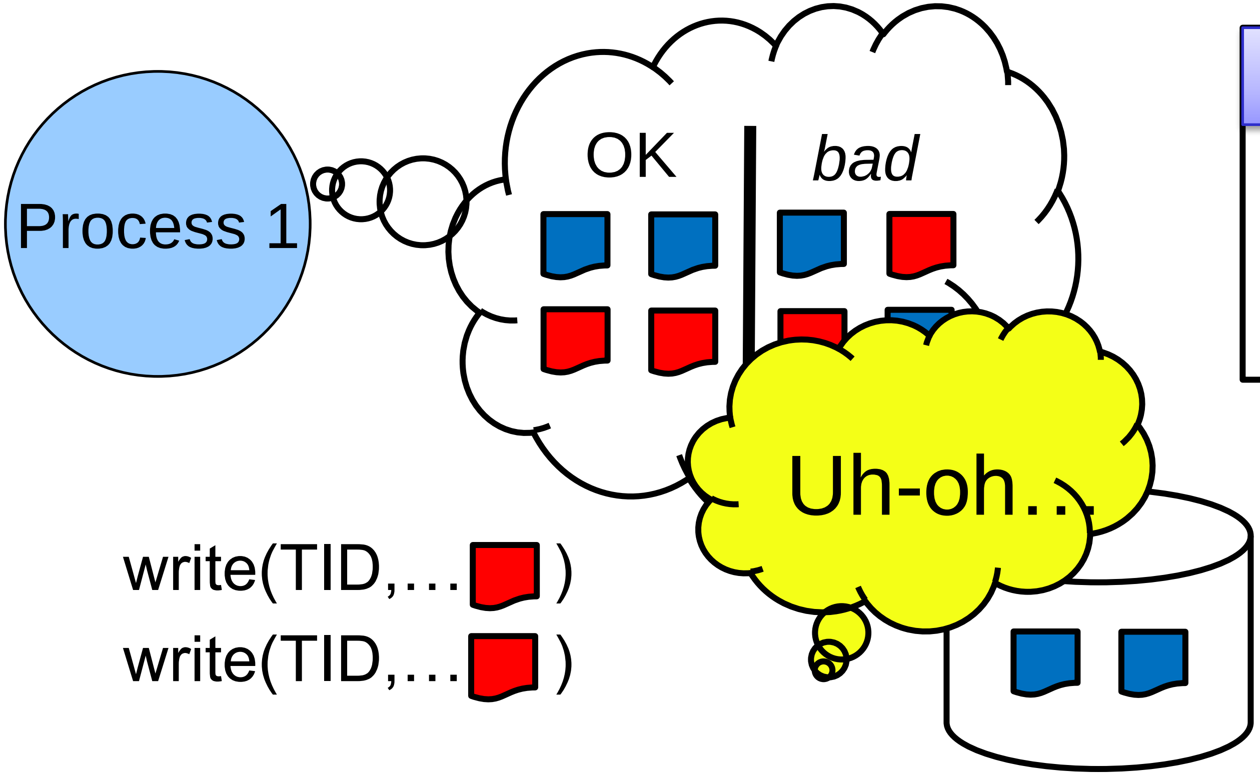
Asynchronous By Default

- ACI (no D w/o tsync)
- Similar to asynchronous write(2) with fsync(2)
- Same purpose (performance increase)
- Requires page cache for files updated transactionally

Valor Design

- Modify page writeback to support simple write ordering
- Implement an ARIES style undo/redo log module for FS-operations

Page Dirtying: No Txns



LEGEND:

Old Page 🟦

New Page 🟥

write(TID,…🟥)

write(TID,…🟥)

Page Dirtying: With Txns

LEGEND:

Old Page	🟦
U/R Page	🟨
New Page	🟥

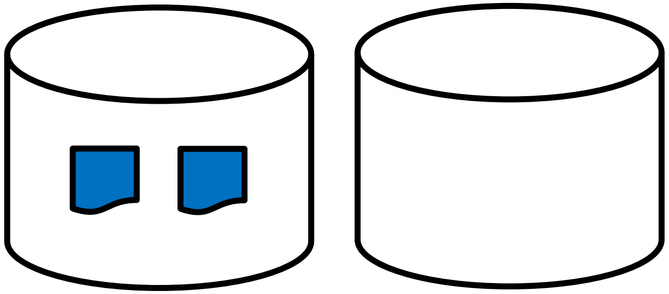
Process 1

log_append(TID,…🟨)

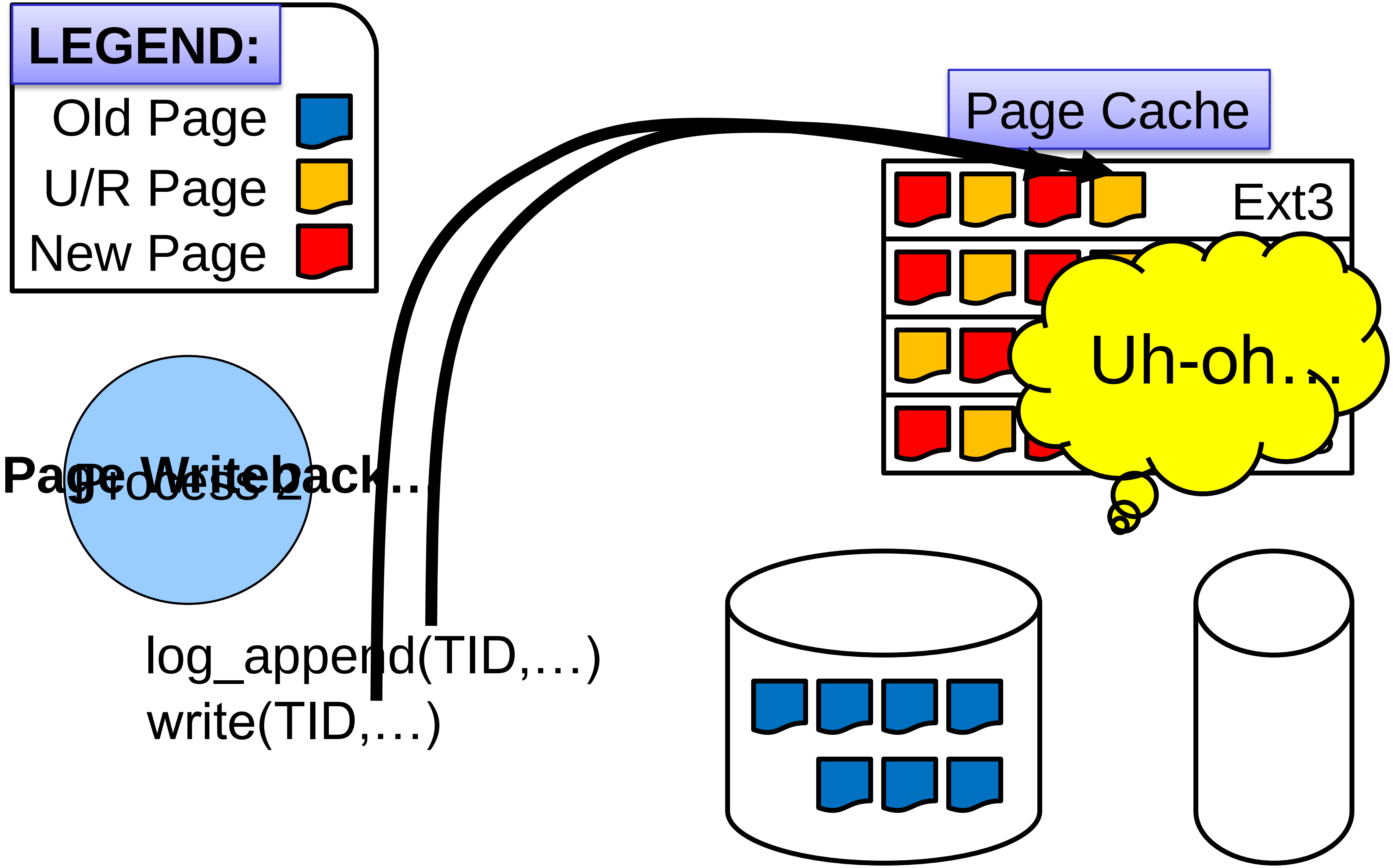
log_append(TID,…🟨)

write(TID,…🟥)

write(TID,…🟥)



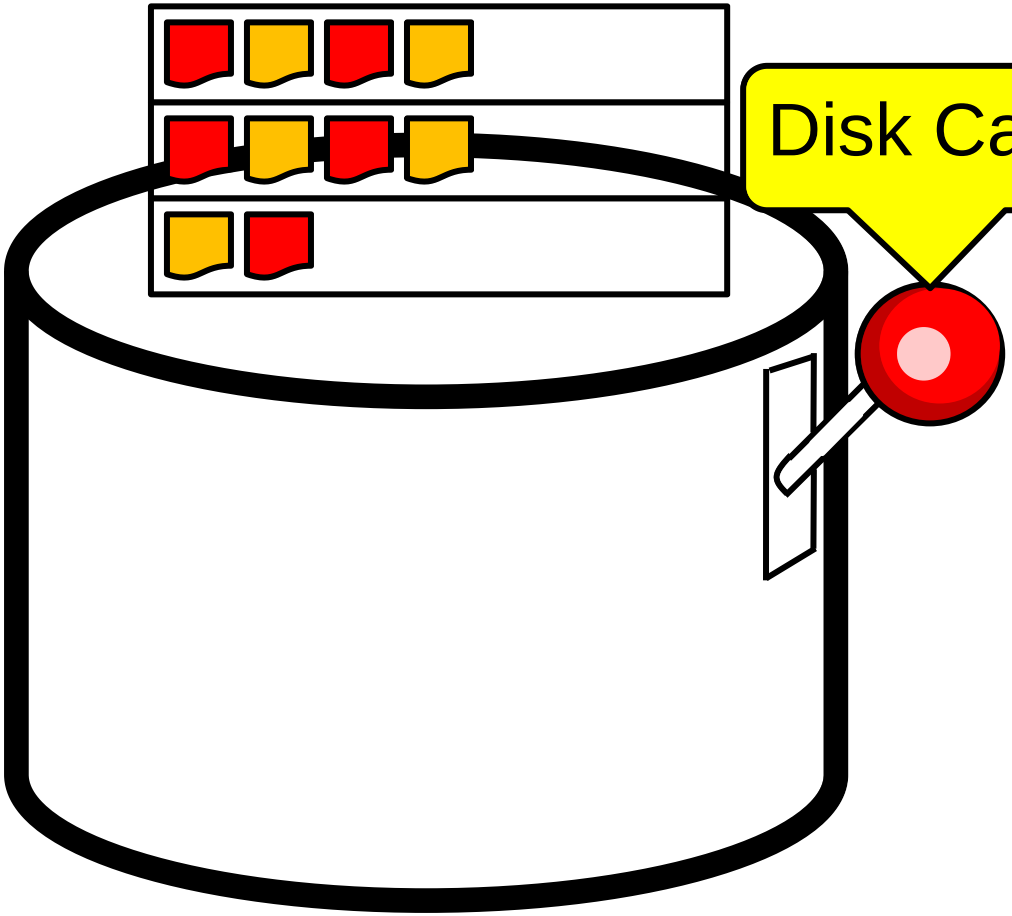
Current Kernel Design



log_append(TID,…)
write(TID,…)

What DBs Do

Page Cache II: The Wrath of Khan



Disk Cache Flush

(fsync)

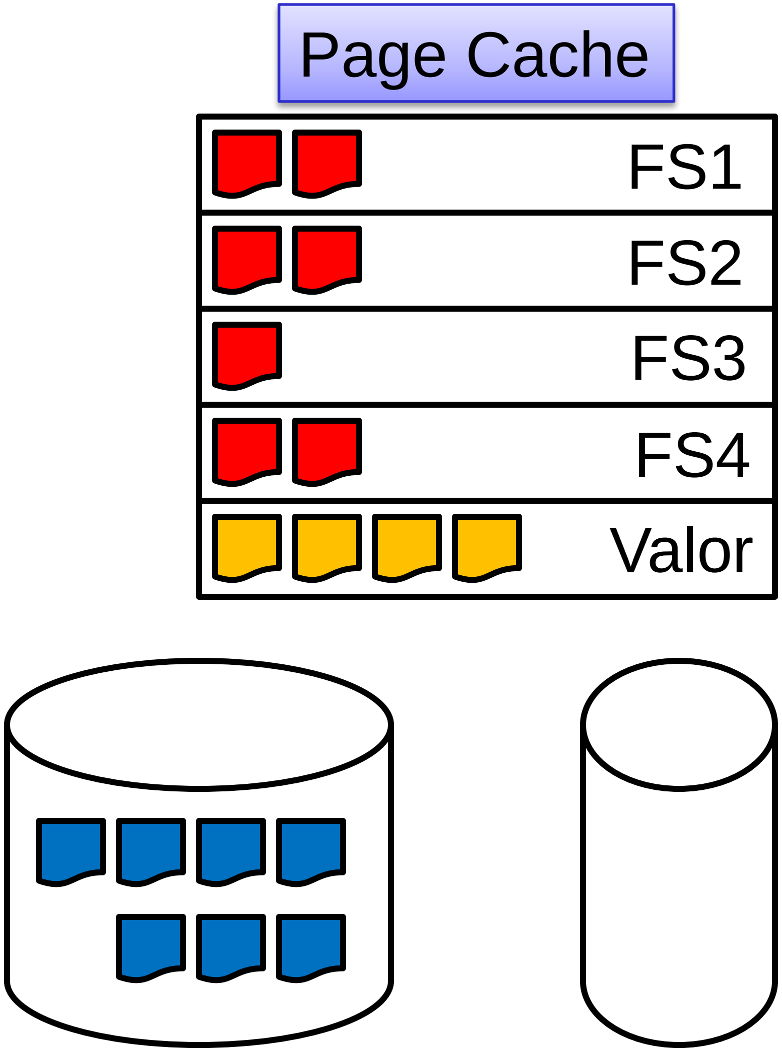
Simple Write Ordering

LEGEND:

Old Page 🟦

U/R Page 🟨

New Page 🟥



Page Cache

FS1

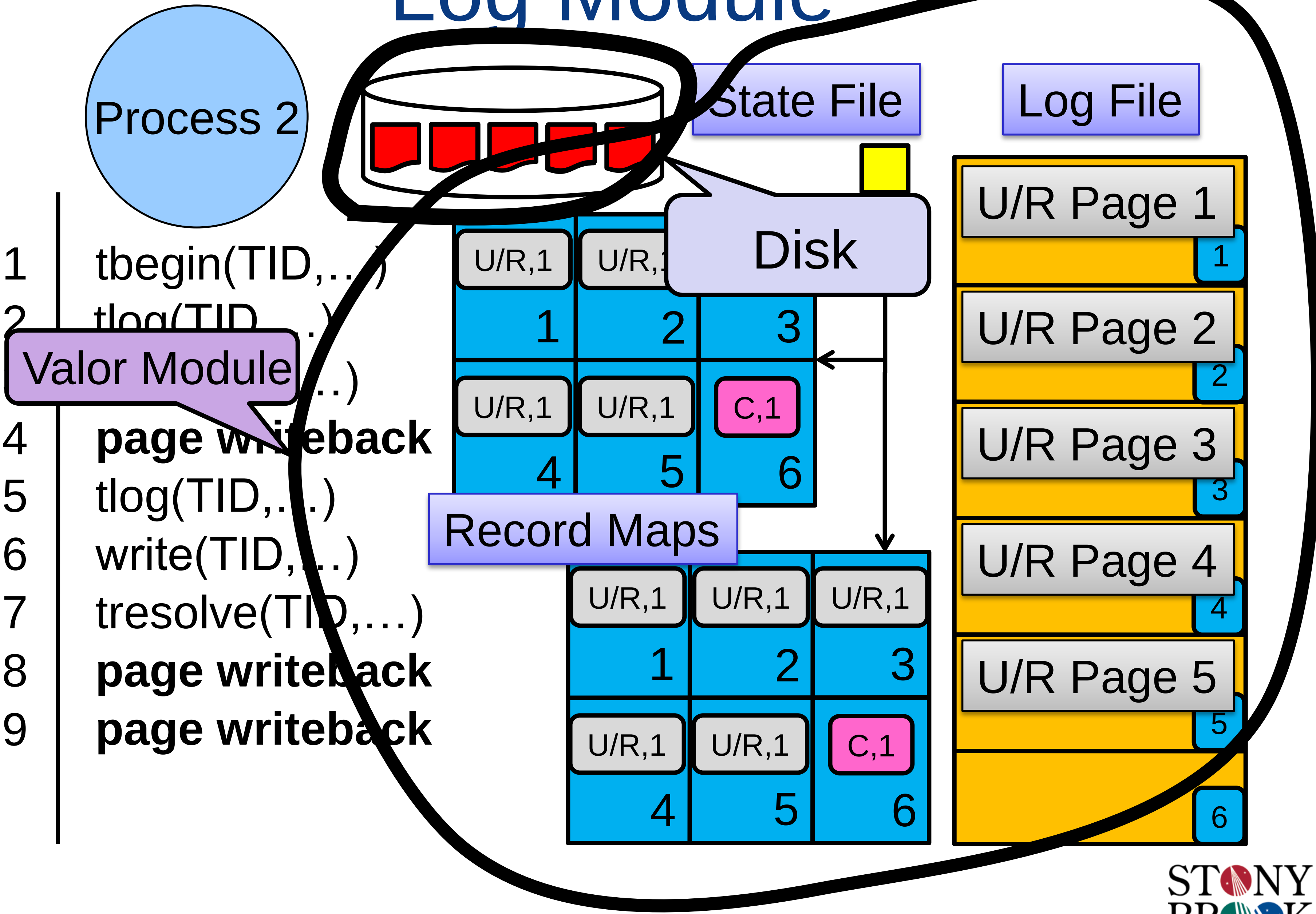
FS2

FS3

FS4

Valor

Log Module



1 tbegin(TID,…)
2 tlog(TID,…)
…)
4 **page writeback**
5 tlog(TID,…)
6 write(TID,…)
7 tresolve(TID,…)
8 **page writeback**
9 **page writeback**

Atomicity Argument

- Transition from pre-writeback to post-writeback disk state atomically ***iff***
  - All writes preceded by **sys_log_append**
  - Simple write ordering is implemented
  - writes to a single sector are atomic
- Valor satisfies the top 2 constraints
- A supported hard disk satisfies the third

Performing Recovery

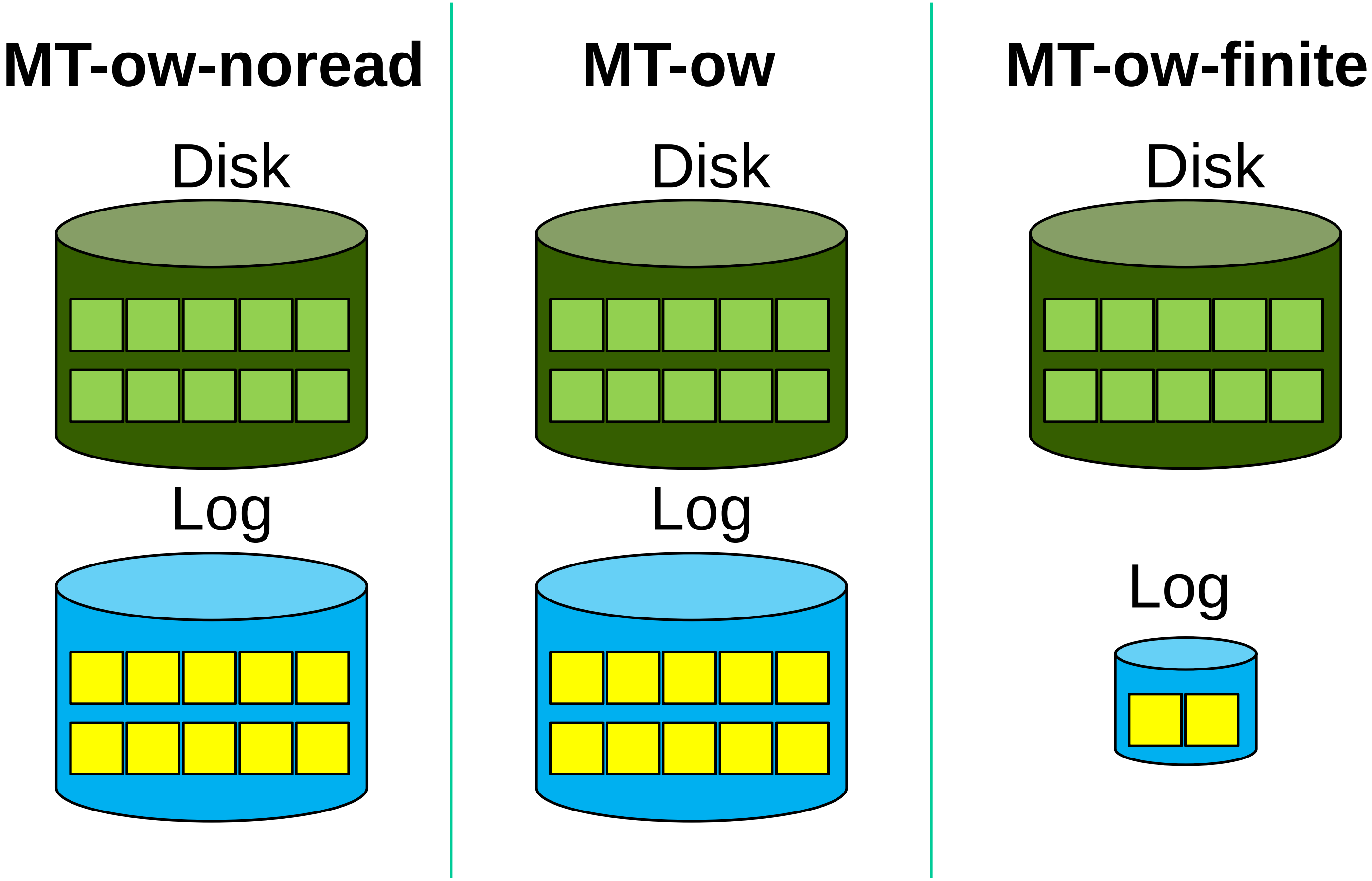
- Two kinds of recovery are supported:
  - System Recovery
  - Application Recovery (per-process abort)
- Standard recovery process:
  - Reconstruct RAM state from log
  - In reverse LSN order commit/abort landed transactions
  - Perform a page writeback

Evaluation

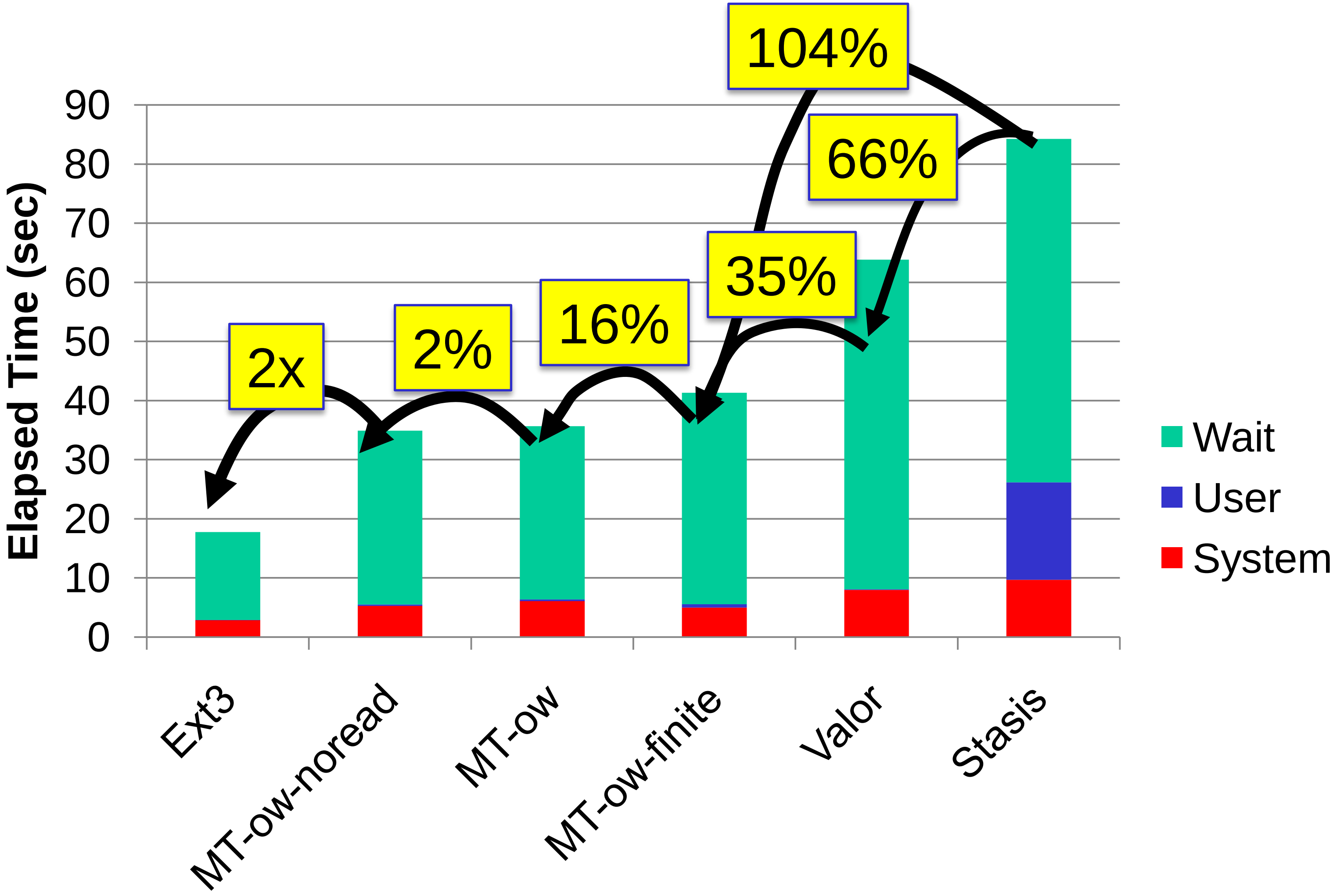
- We must compare against traditional asynchronous FSes
  - benchmark against asynchronous ext3
  - do serial transfer benchmarks for large files
- We turn *off* synchronous transactions for two other controls (for fairness)
  - FS built on top of Stasis
  - FS built on top of Berkeley DB

Mock ARIES Benchmark

- Important lower bound (not tight)

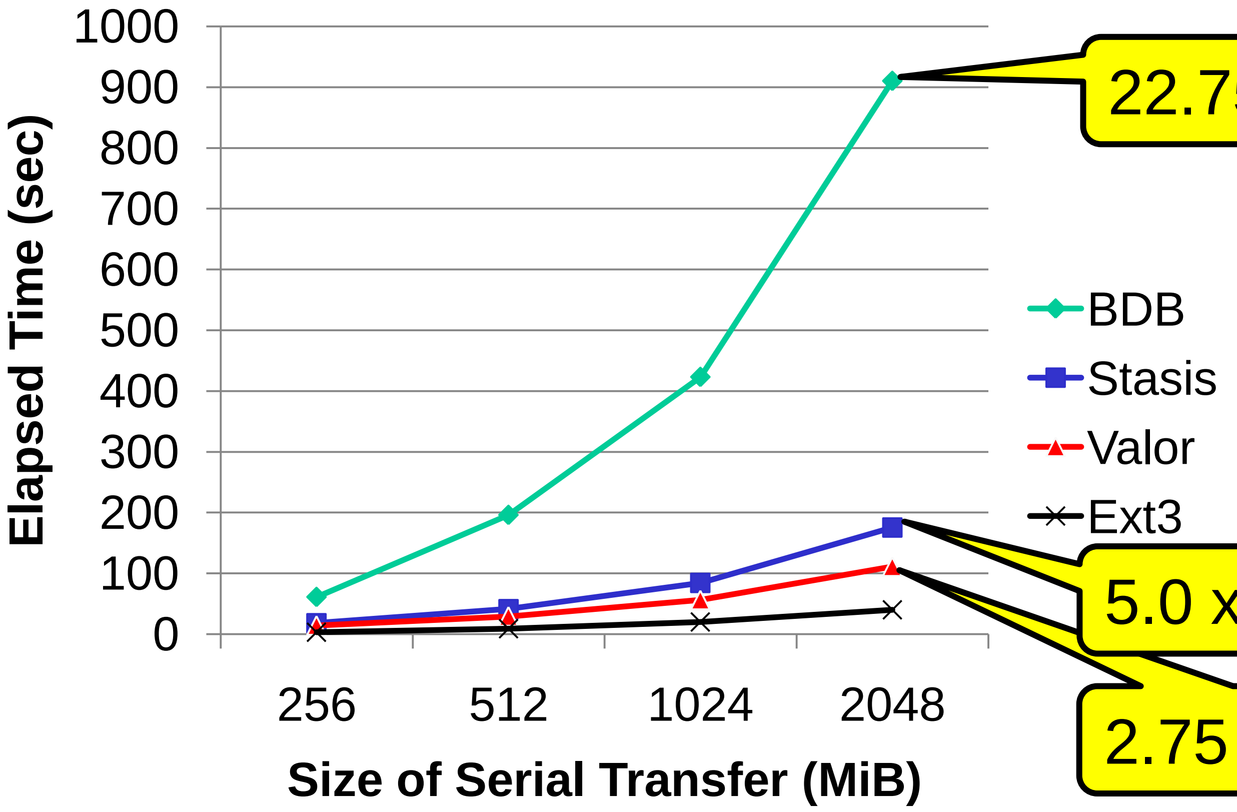


Mock ARIES Benchmark



Serial Overwrite

Transaction size: 16 pages

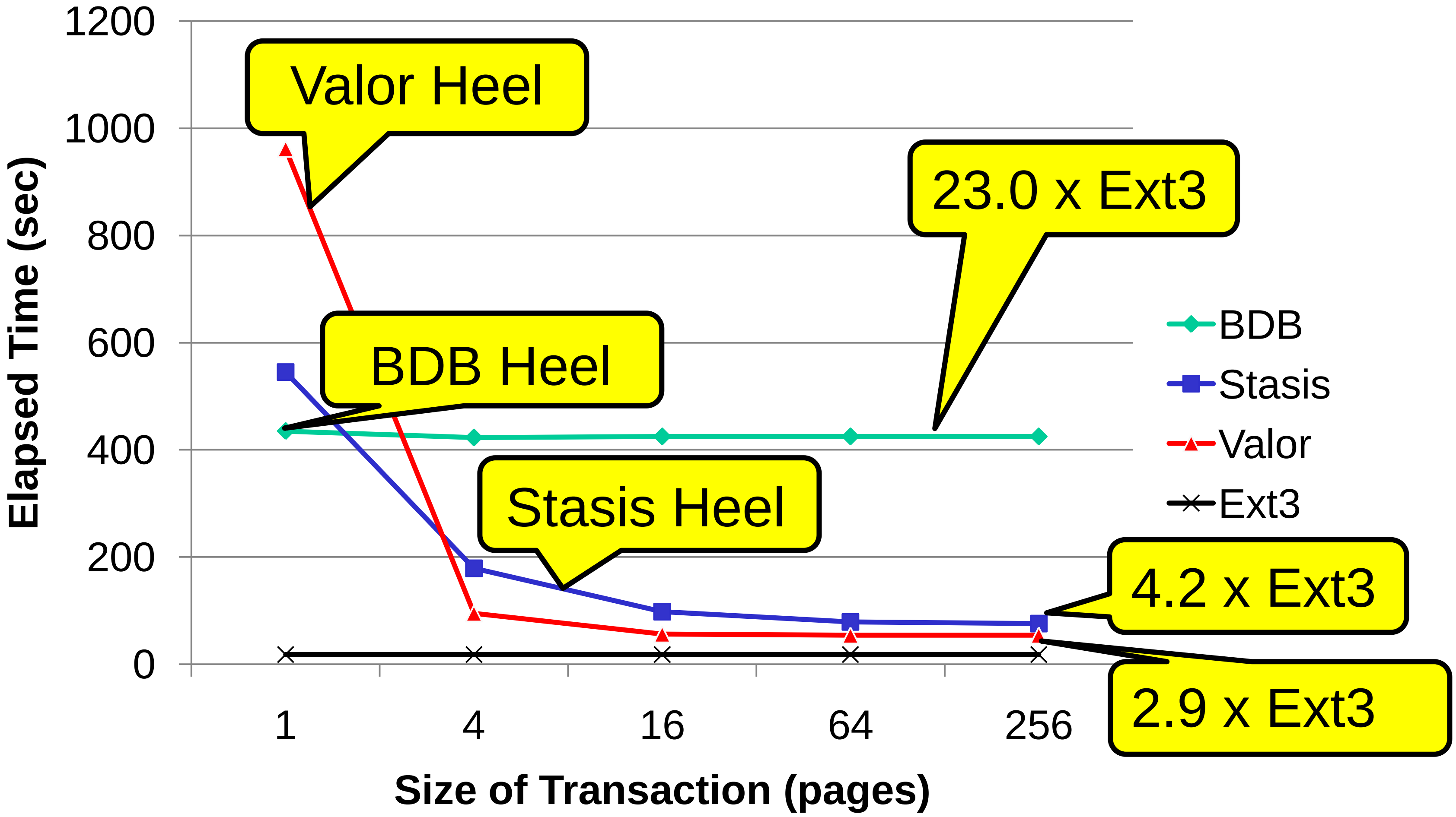


22.75 x Ext3

5.0 x Ext3

2.75 x Ext3

Transaction Throughput



Conclusions

- System transactions are feasible
- Valor achieves good overhead
- Minimal changes to existing kernels

Limitations/Future Work

- Limitations
  - Locking slows interleaved writes to the same page
  - Some FSes/Disks do not fsync() when asked to
- Future Work
  - Explore use of logging device as a coordinator in a transactional disk array

Q&A

Enabling System Transactions via Lightweight Kernel Extensions

R.P. Spillane, S. Gaikwad. M. Chinni, C.P. Wright, E. Zadok

Stony Brook University

http://www.fsl.cs.sunysb.edu/

TxF

- TxF is Microsoft’s transactional file system
  - Motivation: program installation, system updates, website updates
- Pros
  - Backed by Microsoft
- Cons
  - Specific to NTFS

Isolation

- Extended mandatory locking
  - Allows locking of directories
  - Do not have to set group exec/setgid bits
- Locking permissions
  - Let users decide if a file can be locked
- All processes acquire locks
  - Regular processes hold only for the syscall
- Lock inheritance
  - Allow multi-process transactions

Valor != Journaling

- Journaling FSes good at **fast recovery**
- …but are too **special-purpose:**
  - *No-Steal Caching*
    - all state modified by a txn. must remain in memory until commit/abort
  - *Non-Modular Design*
    - does not handle rollback of VFS and page caches, just disk-state on boot