OPERATING SYSTEM TRANSACTIONS

Donald E. Porter, Owen S. Hofmann,
Christopher J. Rossbach, Alexander Benn,
and Emmett Witchel

The University of Texas at Austin

OS APIs don’t handle concurrency

- OS is weak link in concurrent programming model
- Can’t make consistent updates to system resources across multiple system calls
  - Race conditions for resources such as the file system
  - No simple work-around
- Applications can’t express consistency requirements
- OS can’t infer requirements

System transactions

- System transactions ensure consistent updates by concurrent applications
  - Prototype called TxOS
- Solve problems
  - System level race conditions (TOCTTOU)
- Build better applications
  - LDAP directory server
  - Software installation

System-level races

(root)

foo == /etc/passwd

```
if(access(“foo”)) {

   fd = open(“foo”);
   write(fd,…);
   …
}
```

Time-of-check-to-time-of-use (TOCTTOU) race condition

TOCTTOU race eliminated

(root)

```
sys_xbegin();
if(access(“foo”)) {
   fd = open(“foo”);
   write(fd,…);
   …
}
sys_xend();
```

Example 1: better application design

□ How to make consistent updates to stable storage?

Application	Technique
Enterprise data storage	Database
User directory service (LDAP)	SysTx ????
Editor	rename()

Complex

Simple

Ex 2: transactional software install

```
sys_xbegin();
apt-get upgrade
sys_xend();
```

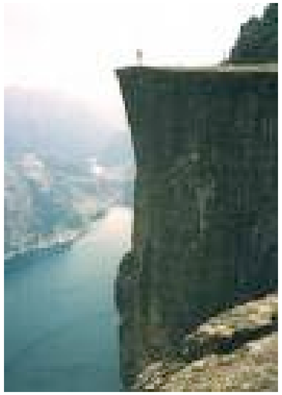
- A failed install is automatically rolled back
  - Concurrent, unrelated operations are unaffected
- System crash: reboot to entire upgrade or none

System transactions

- Simple API: `sys_xbegin, sys_xend, sys_xabort`
- Transaction wraps group of system calls
  - Results isolated from other threads until commit
- Transactions execute concurrently for performance
- Conflicting transactions must serialize for safety
  - Conflict most often read & write of same datum
  - Too much serialization hurts performance

Related work

- Developers changing syscall API for concurrency
  - Ad hoc, partial solutions: `openat()`, etc.
- System transactions have been proposed and built
  - QuickSilver [SOSP '91], LOCUS [SOSP '85]
- Key contribution: new design and implementation
  - Uphold strong guarantees and good performance
- System transactions != transactional memory
  - TxOS runs on commodity hardware



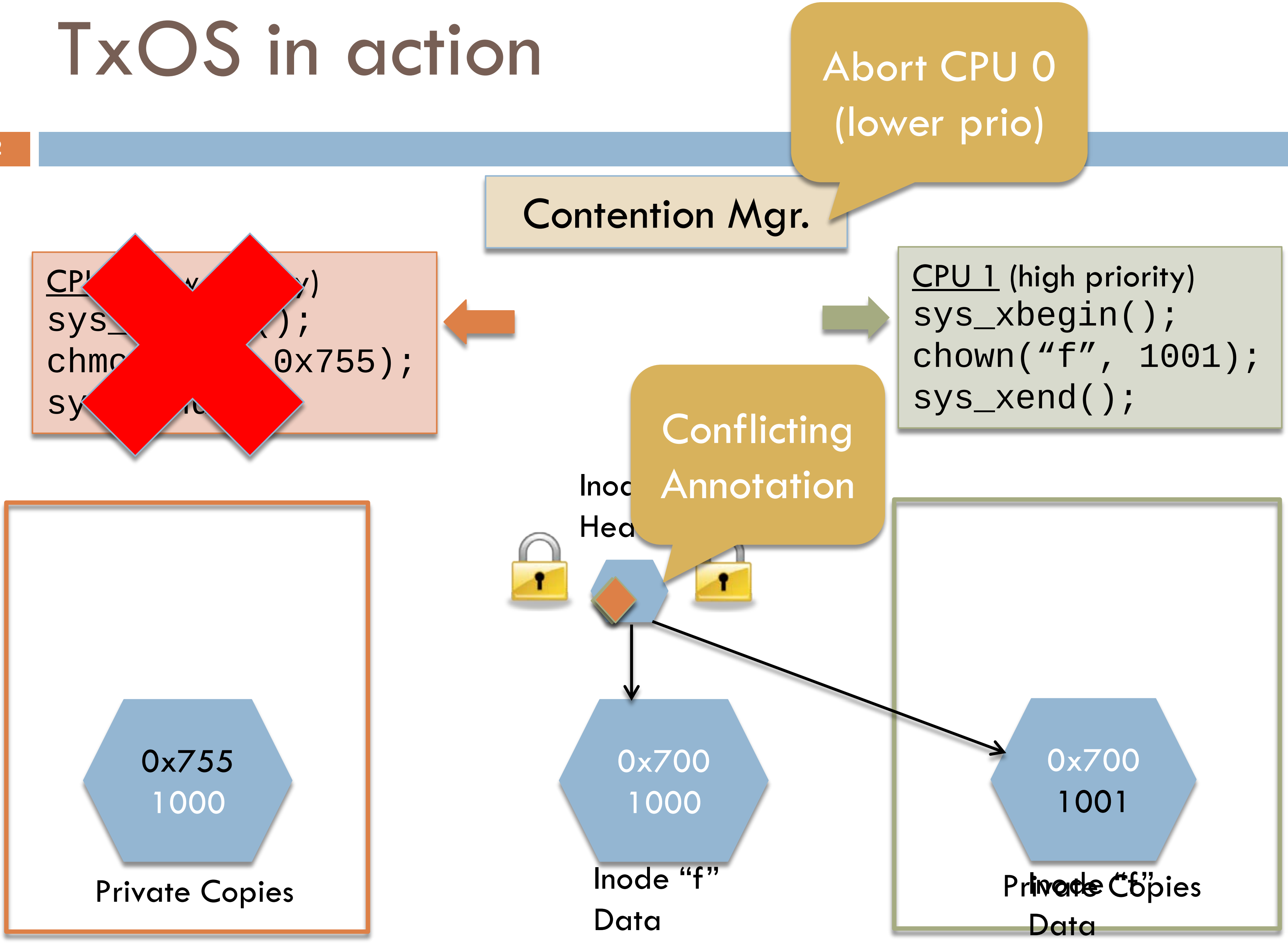
Outline

- Example uses of system transactions
- TxOS design and implementation
- Evaluation

Building a transactional system

- Version management
  - Private copies instead of undo log
- Detect conflicts
  - Minimize performance impact of true conflicts
  - Eliminate false conflicts
- Resolve conflicts
  - Non-transactional code must respect transactional code

TxOS in action



Abort CPU 0 (lower prio)

Contention Mgr.

```
CP      w        y)
sys_          ();
chmo          0x755);
sy      lc
```

CPU 1 (high priority)

```
sys_xbegin();
chown("f", 1001);
sys_xend();
```

Conflicting Annotation

System comparison

	Previous Systems	TxOS
Speculative write location	Shared data structures	Private copies of data structures
Isolation mechanism	Two-phase locking	Private copies + annotations
Rollback mechanism	Undo log	Discard private copies
Commit mechanism	Discard undo log, release locks	Publish private copy by ptr swap

Deadlock prone

Can cause priority inversion

Minimizing false conflicts

	R	Add/Del	
R	☺	✗	
Add/Del	✗	☺	✗
Add/Del+R	✗	✗	✗

OK if different files created, Dir not read

- Insight: object semantics allow more permissive conflict definition and therefore more concurrency
- TxOS supports precise conflict definitions per object type
- Increases concurrency without relaxing isolation

```
sys_xbegin();                sys_xbegin();
create("/tmp/foo");          create("/tmp/bar");
sys_xend();                  sys_xend();
```

Serializing transactions and non-transactions (strong isolation)

- TxOS mixes transactional and non-tx code
  - In database, everything is transaction
  - Semantically murky in historical systems
- Critical to correctness
  - Allows incremental adoption of transactions
  - TOCTTOU attacker will not use a transaction
- Problem: can't roll back non-transactional syscall
  - Always aborting transaction undermines fairness

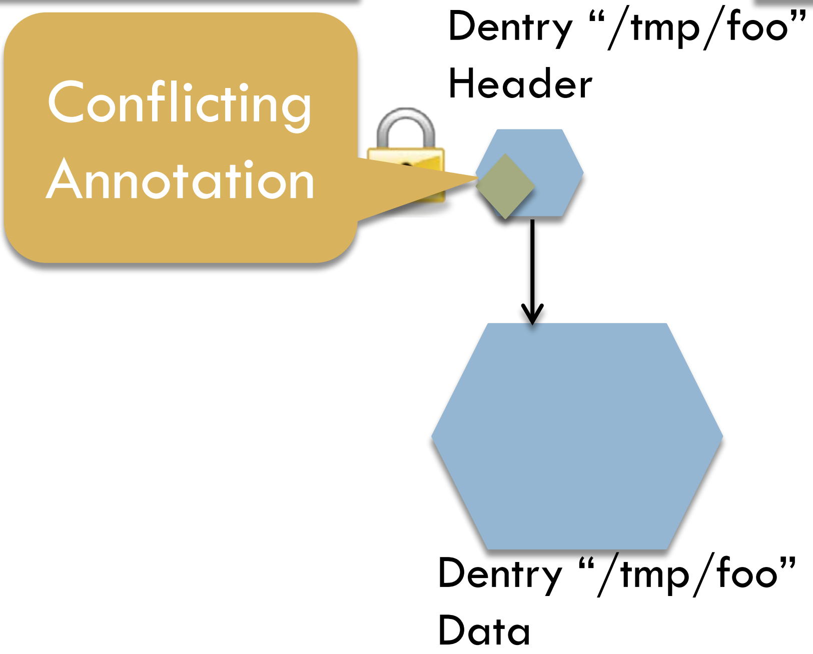
Strong isolation in TxOS

CPU 0

```
symlink("/etc/passwd",
        "/tmp/foo");
```

CPU 1

```
sys_xbegin();
if(access("/tmp/foo"))
  open("/tmp/foo");
sys_xend();
```



Contention Manager

- Options:
  - Abort CPU1
  - Deschedule CPU0

Transactions for application state

- System transactions only manage system state
- Applications can select their approach
  - Copy-on-write paging
  - Hardware or Software Transactional Memory (TM)
  - Application-specific compensation code

Transactions: a core OS abstraction

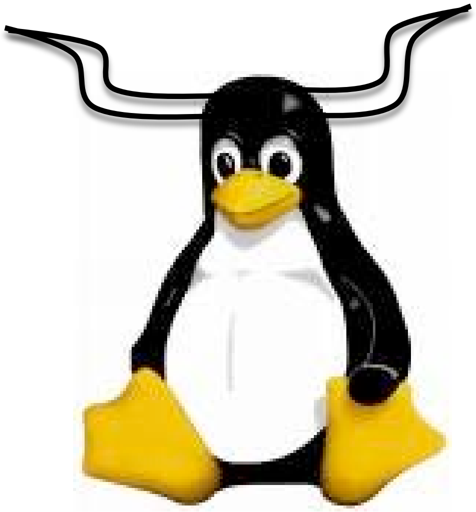
- Easy to make kernel subsystems transactional
- Transactional filesystems in TxOS
  - Transactions implemented in VFS or higher
  - FS responsible for atomic updates to stable store
- Journal + TxOS = Transactional Filesystem
  - 1 developer-month transactional ext3 prototype

Evaluation

- □ Example uses of system transactions
- □ TxOS design and implementation
- □ Evaluation
    - What is the cost of using transactions?
    - What overheads are imposed on non-transactional applications?

TxOS Prototype

- Extend Linux 2.6.22 to support system transactions
  - Add 8,600 LOC to Linux
  - Minor modifications to 14,000 LOC
- Runs on commodity hardware
- Transactional semantics for a range of resources:
  - File system, signals, processes, pipes



Hardware and benchmarks

- Quadcore 2.66 GHz Intel Core 2 CPU, 4 GB RAM

Benchmark	Description
install	install of svn 1.4.4
make	Compile nano 2.06 inside a tx
dpkg	dpkg install OpenSSH 4.6
LFS large/small	Wrap each phase in a tx
RAB	Reimplemeted Andrew Benchmark Each phase in a tx

Transactional software install

```
sys_xbegin();
dpkg –i openssh;
sys_xend();
```

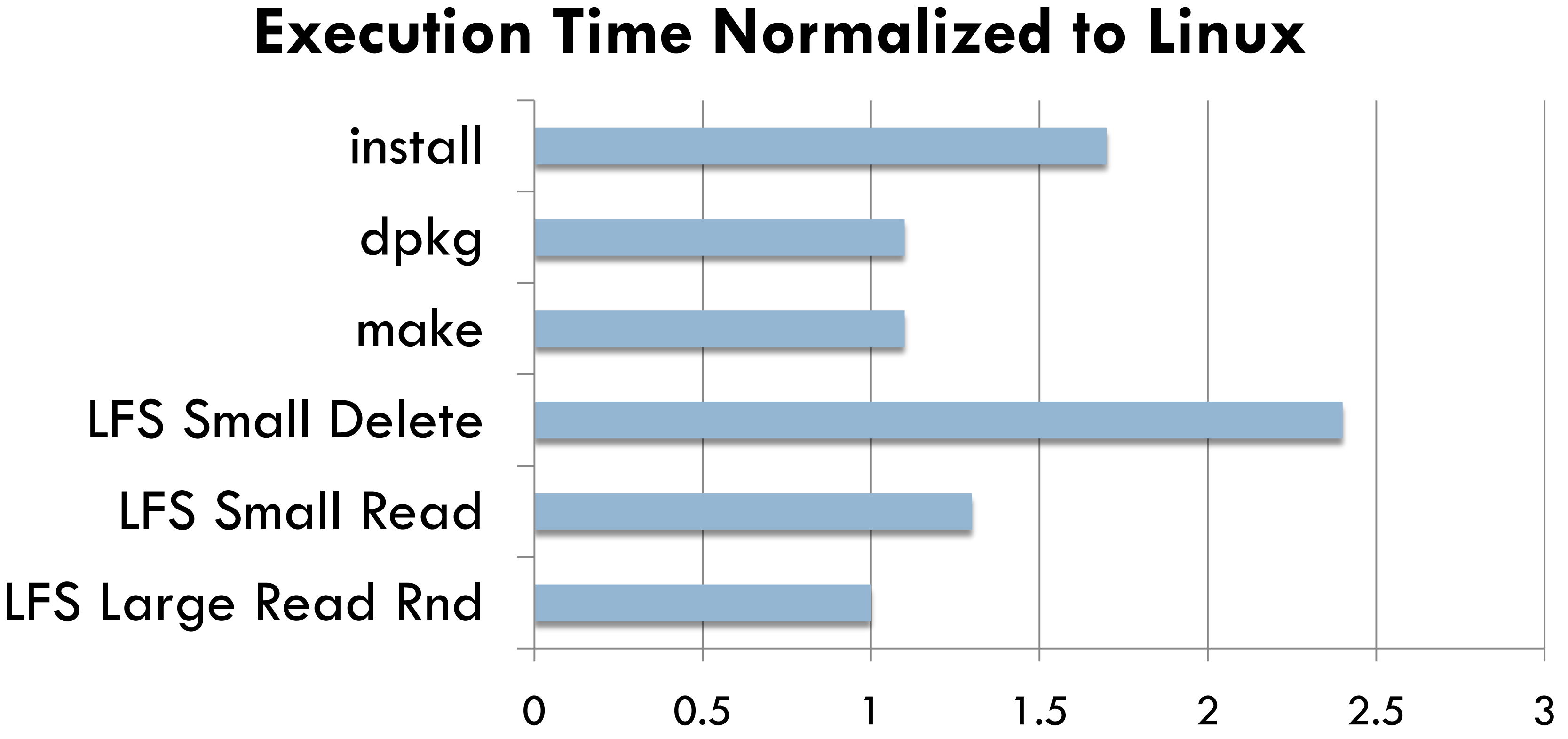
10% overhead

```
sys_xbegin();
install svn;
sys_xend();
```

70% overhead

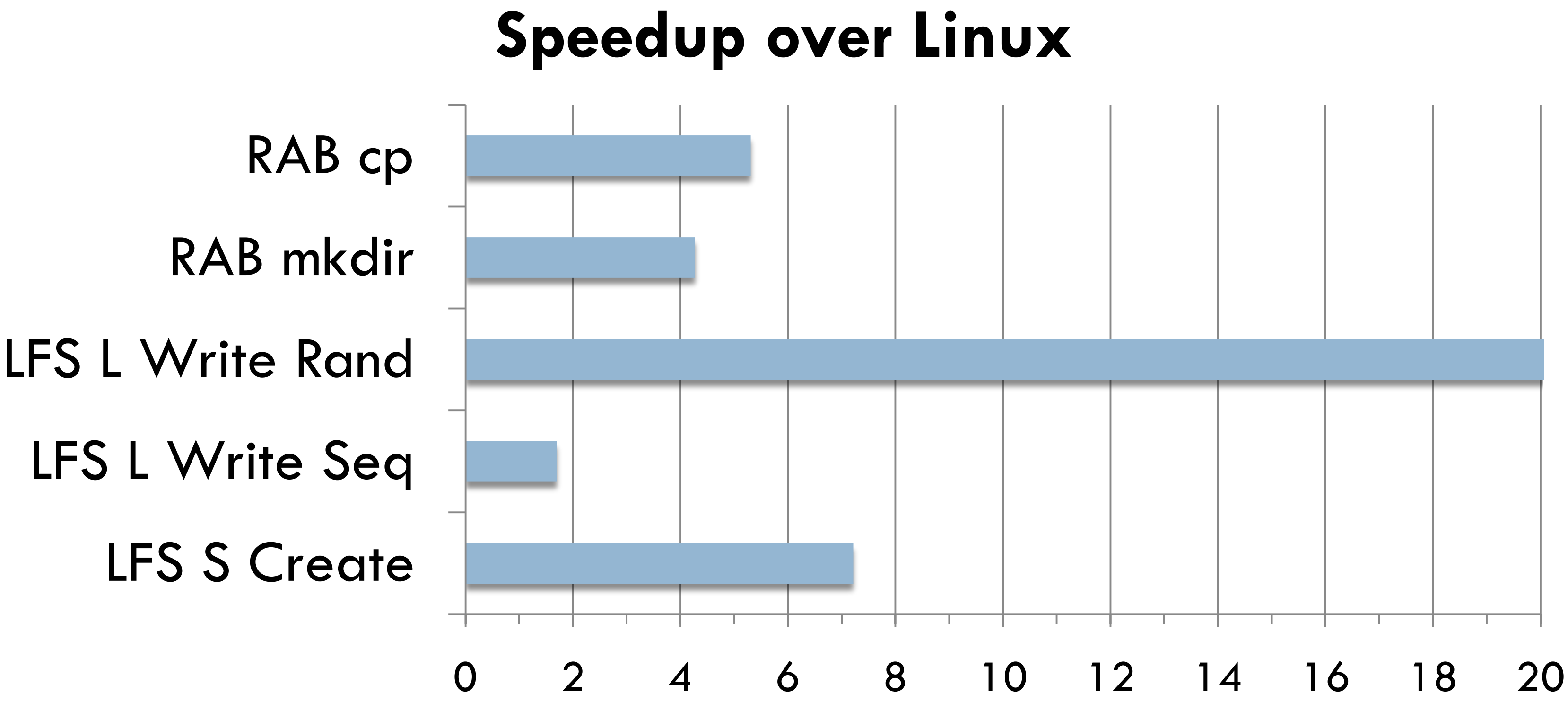
- A failed install is automatically rolled back
  - Concurrent, unrelated operations are unaffected
- System crash: reboot to entire upgrade or none

Transaction overheads



- Memory overheads on LFS large:
  - 13% high, 5% low (kernel)

Write speedups



- □ Better I/O scheduling – not luck
- □ Tx boundaries provide I/O scheduling hint to OS

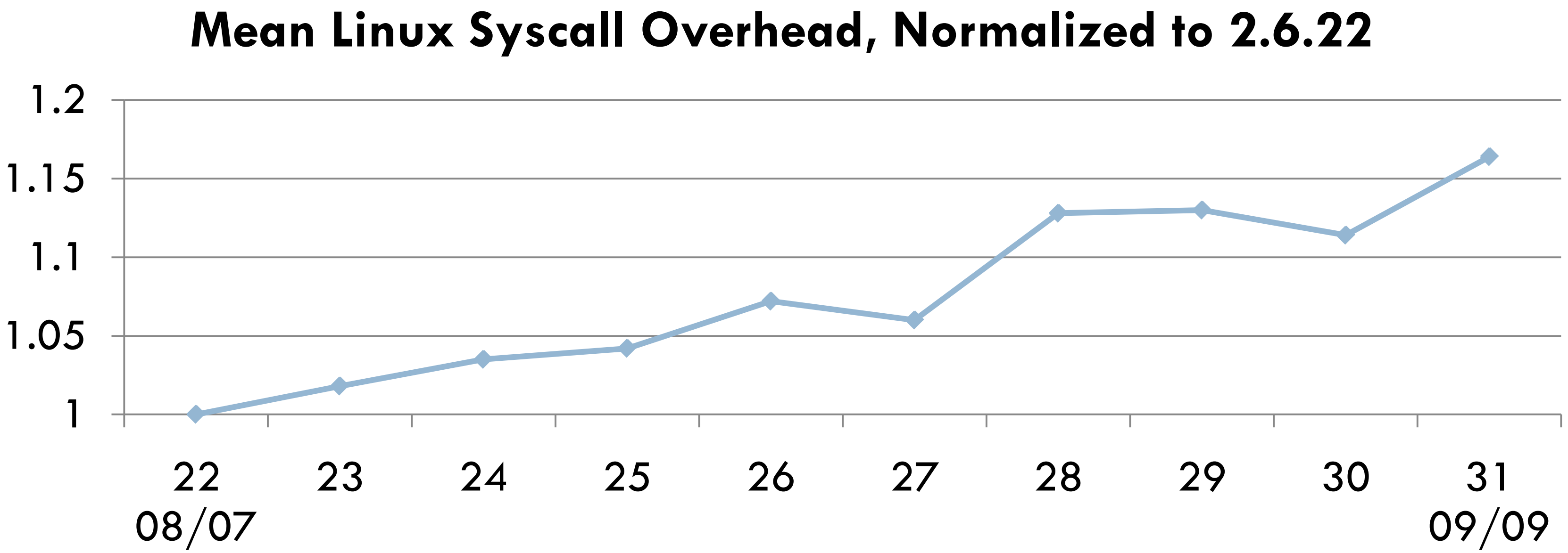
Lightweight DB alternative

- OpenLDAP directory server
  - Replace BDB backend with transactions + flat files
- 2-4.2x speedup on write-intensive workloads
- Comparable performance on read-only workloads
  - Primarily serviced from memory cache

Non-transactional overheads

- Non-transactional Linux compile: <2% on TxOS
  - Transactions are “pay-to-play”
- Single system call: 42% geometric mean
  - With additional optimizations: 14% geomean
  - Optimizations approximated by eliding checks

What is practical?



- Feature creep over 2 years costs 16%
- Developers are willing to give up performance for useful features
- Transactions are in same range (14%), more powerful

OSes should support transactions

- Practical implementation techniques for modern OS
- Transactions solve long-standing problems
  - Replace ad hoc solutions
- Transactions enable better concurrent programs

http://www.cs.utexas.edu/~porterde/txos

porterde@cs.utexas.edu

Backup Slides

Windows kernel transaction manager

- Framework for 2-Phase Commit
  - Coordinate transactional file system, registry
- Transactional FS and registry
  - Completely different implementation
  - FS updates in place, Registry uses private copies
  - Little opportunity for code reuse across subsystems
- Explicitly transacted code
  - More conservative, limited design choice
  - TxOS allows implicit transactions, application wrappers

Distributed transactions

- User/language-level transactions
  - Cannot isolate OS managed resources
- TABS [SOSP ‘85], Argus [SOSP ‘87], Sinfonia [SOSP ’07]
- TABS – transactional windows manager
  - Grayed out aborted dialog
- Argus – similar strategies for limiting false conflicts

Transactional file systems

- Good idea, difficult to implement
  - Challenging to implement below VFS layer
  - Valor [FAST '09] introduces OS support in page cache
- Lack simple abstractions
  - Users must understand implementation details
    - Deadlock detection (Transactional NTFS)
    - Logging and locking mechanism (Valor)
- Lack support for other OS resources in transactions
  - Windows KTM supports transactional registry

Speculator

- Goal: hide latency of operations
  - NFS client requests, synchronous writes, etc.
- Similar implementation at points
- Different goals, not sufficient to provide transactional semantics
  - Isolation vs. dependences

xCalls [EuroSys '09]

- User-level techniques for transactional system calls
  - Within a single application only
- Works for many common cases (buffering writes)
  - Edge cases difficult without system support
    - E.g., `close()` or `munmap()` can implicitly delete a file