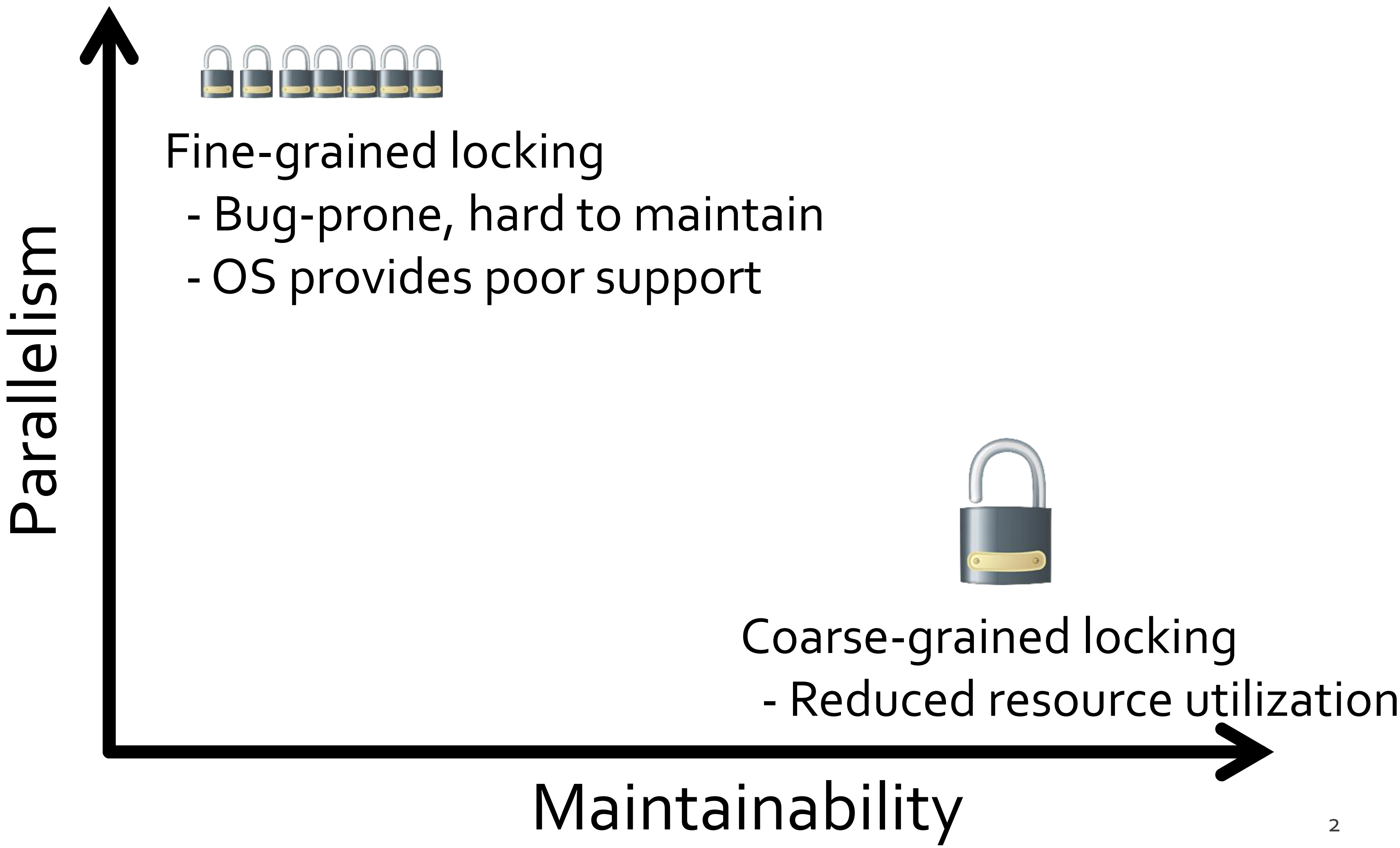
Improving Server Applications with System Transactions

Sangman Kim, Michael Z. Lee, Alan M. Dunn,
Owen S. Hofmann, Xuan Wang,
Emmett Witchel, Donald E. Porter

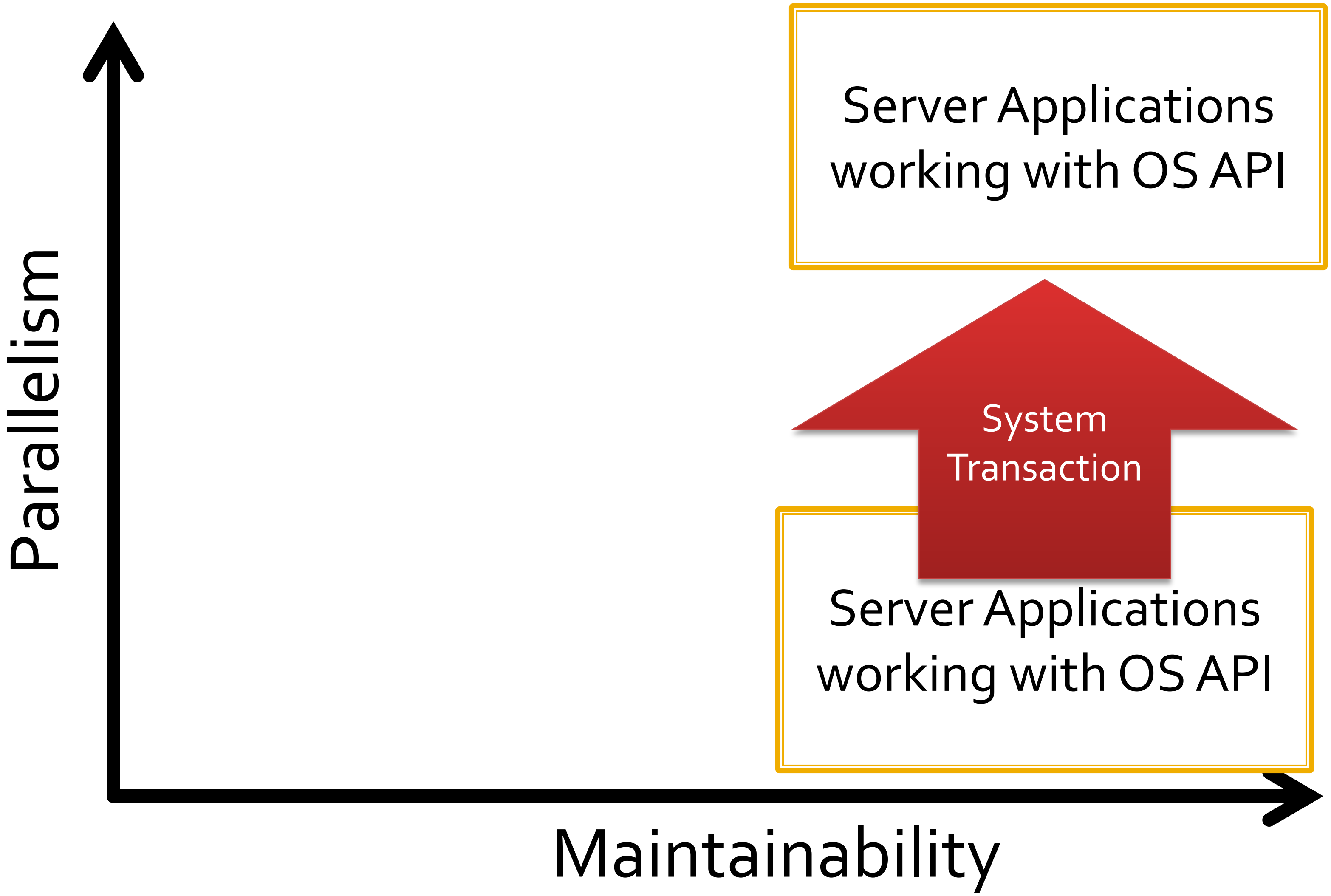
THE UNIVERSITY OF TEXAS AT AUSTIN

Stony Brook University

Poor OS API Support for Concurrency



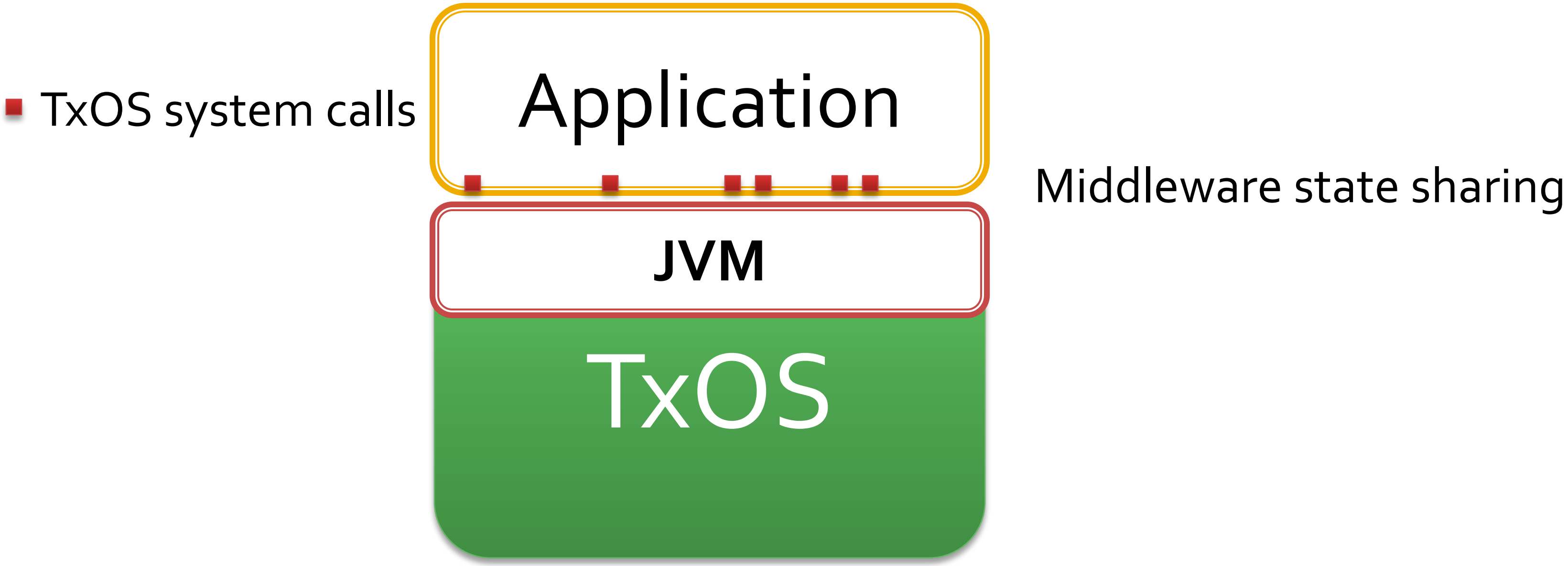
System Transaction Improves OS API Concurrency



Improving System Transactions

- TxOS provides operating system transaction
  [Porter et al., SOSP 2009]
  - Transaction for OS objects (e.g., files, pipes)

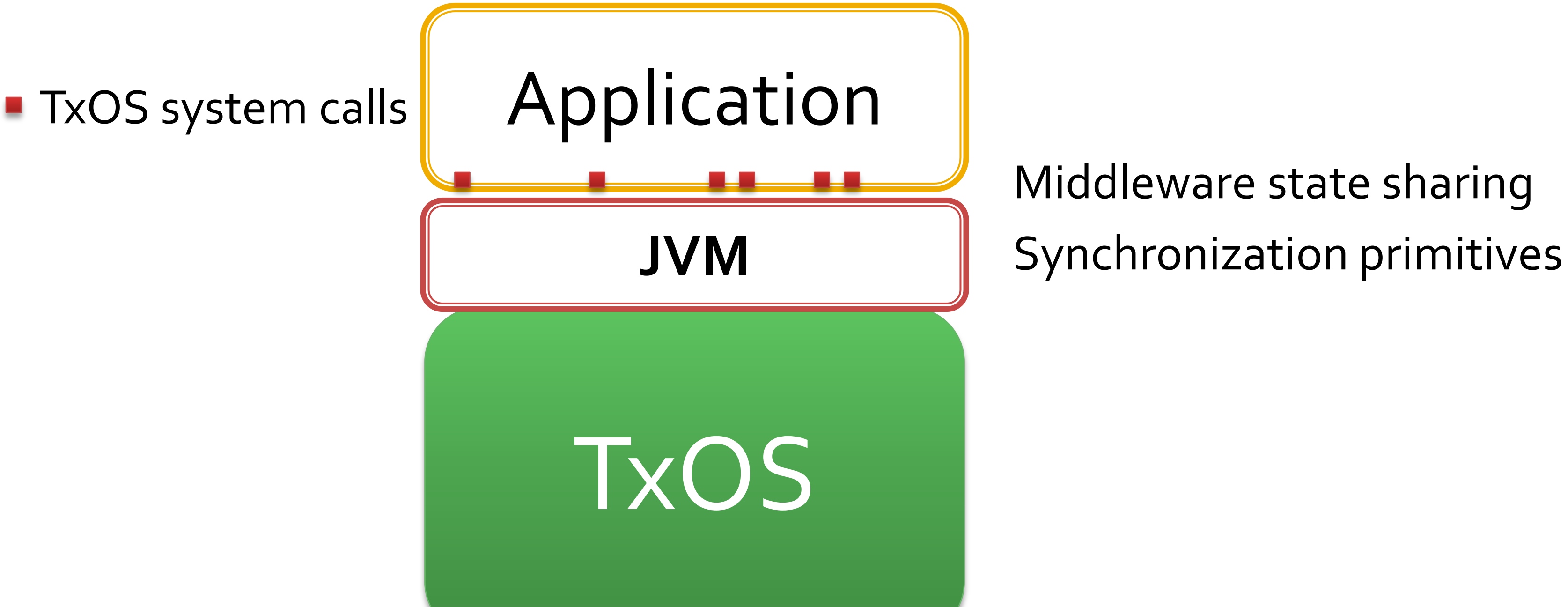
Middleware state sharing with multithreading



Improving System Transactions

- TxOS provides operating system transaction [Porter et al., SOSP 2009]
  - Transaction for OS objects (e.g., files , pipes)

Synchronization in legacy code



Improving System Transactions

- TxOS provides operating system transaction
  [Porter et al., SOSP 2009]

Up to 88% throughput improvement

At most 40 application line changes

- TxOS system calls

Application

JVM

TxOS+

Middleware state sharing

Synchronization primitives

TxOS+: pause/resume, commit ordering, and more

Outline

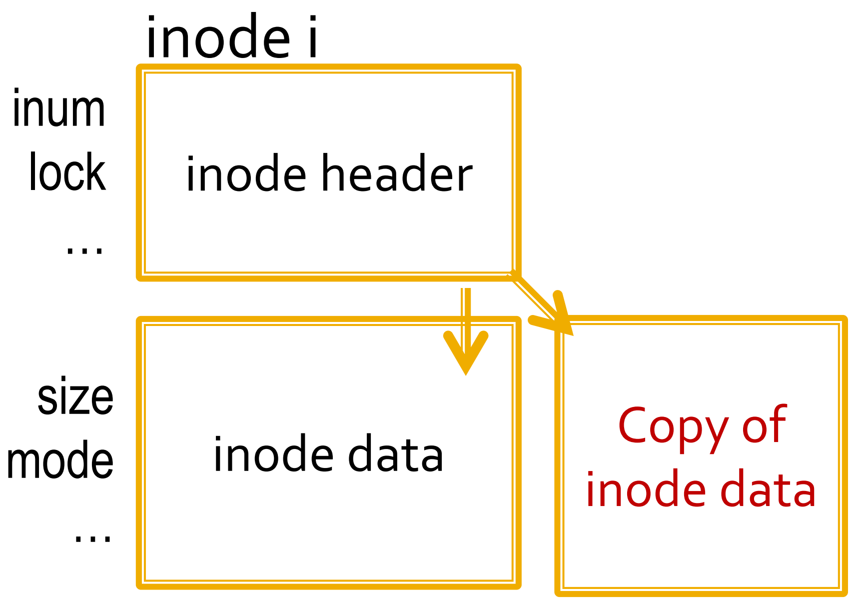
- Background: system transaction
- System transactions in action
- Challenges for rewriting applications
- Implementation and evaluation

Background: System Transaction

- Transaction Interface and semantics
  - System calls: xbegin(), xend(), xabort()
  - ACID semantics
    - Atomic – all or nothing
    - Consistent – one consistent state to another
    - Isolated – updates as if only one concurrent transaction
    - Durable – committed transactions on disk
  - Optimistic concurrency control
- Fix synchronization issues with OS APIs

Background: System Transaction

- Lazy versioning: speculative copy for data



xbegin();	←
write(f, buf);	← Conflict!
xend();	← Abort / Commit

- **TxOS requires no special hardware**

Outline

- Background: system transaction
- System transactions in action
- Challenges for rewriting applications
- Implementation and evaluation

Applications Parallelized with OS Transactions

- **Parallelizing applications that synchronize on OS state**
- Example 1: State-machine replication
  - Constraint: Deterministic state update
- Example 2: IMAP Email Server
  - Constraint: Consistent file system operations

Example 1: Parallelizing State-machine Replication

- Core component of fault tolerant services
  - e.g., Chubby, Zookeeper, Autopilot
- Replicas execute the same sequence of operations
  - Often **single-threaded** to avoid non-determinism
- **Ordered transaction**
  - Makes parallel OS state updates deterministic
  - Applications determine commit order of transactions

Example 2: Parallelizing IMAP Email Servers

- **Everyone has concurrent email clients**
  - Desktop, laptop, tablets, phones, ...
  - Need concurrent access to stored emails
- **Brief history of email storage formats**
  - mbox: single file, file locking
  - Lockless Maildir
  - Dovecot Maildir: return of file locking

mbox: Database Without Parallelism

- mbox
  - **Single file** mailbox of email messages

~/.mbox

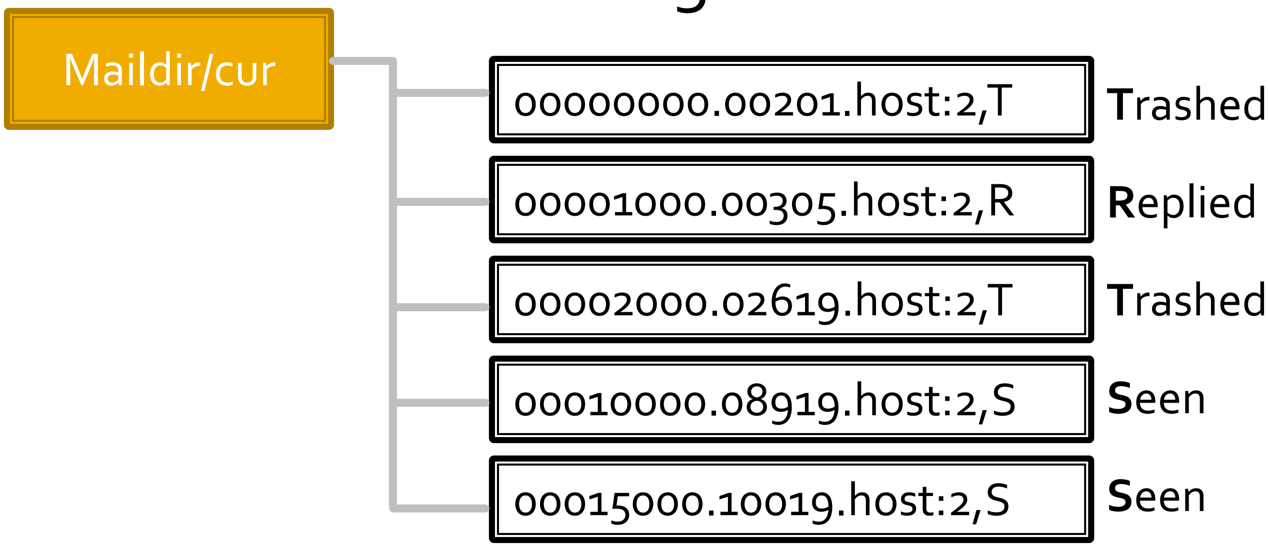
```
From MAILER-DAEMON Wed Apr 11 09:32:28 2012
From: Sangman Kim <sangmank@cs.utexas.edu>
To: EuroSys 2012 audience
Subject: mbox needs file lock. Maildir hides message.
.....
From MAILER-DAEMON Wed Apr 11 09:34:51 2012
From: Sangman Kim <sangmank@cs.utexas.edu>
To: EuroSys 2012 audience
Subject: System transactions good, file locks bad!
....
```

  - Synchronization with file-locking
    - One of fcntl(), flock(), lock file (.mbox.lock)
    - Very coarse-grained locking

Maildir: Parallelism Through Lockless Design

- Maildir: Lockless alternative to mbox
  - Directories of message files
  - Each file contains a message
  - Directory access with no synchronization (originally)

- Message filenames contain flags



Messages Hidden with Lockless Maildir

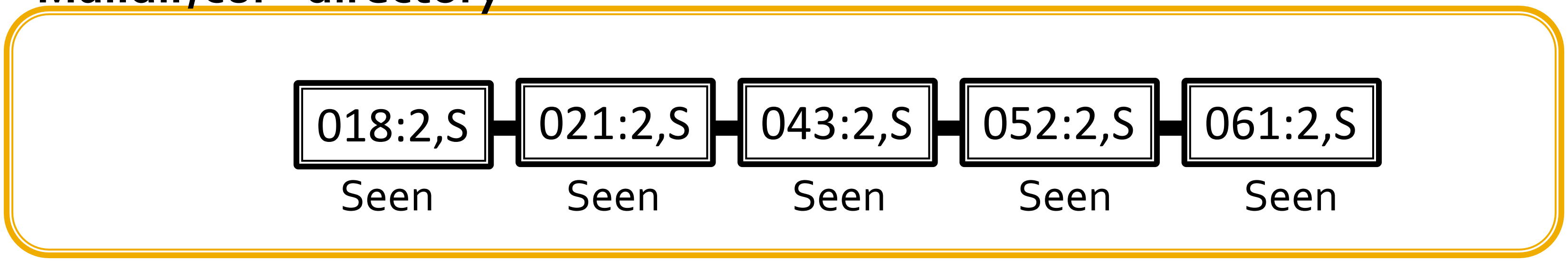
PROCESS 1 (LISTING)

```
while (f = readdir(“Maildir/cur”)):
    print f.name
```

PROCESS 2 (MARKING)

```
if (access(“043:2,S”)):
    rename(“043:2,S”, “043:2,R”)
```

“Maildir/cur” directory



Messages Hidden with Lockless Maildir

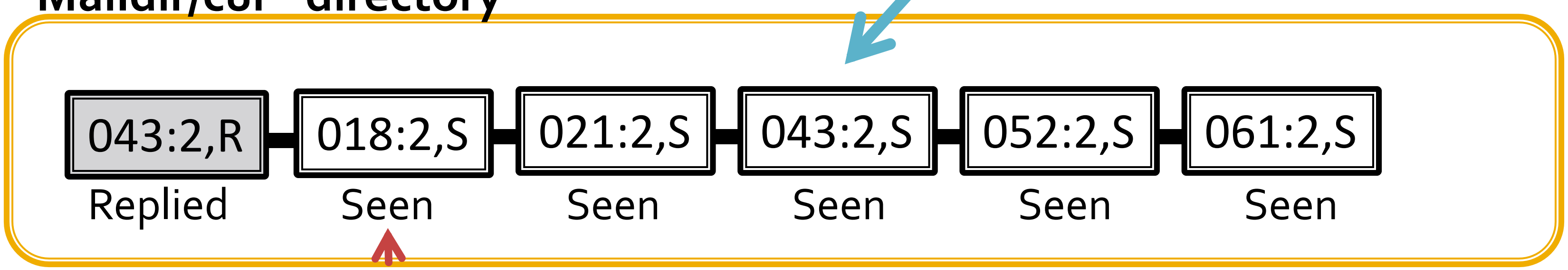
PROCESS 1 (LISTING)

```
while (f = readdir("Maildir/cur")):
    print f.name
```

PROCESS 2 (MARKING)

```
if (access("043:2,S")):
    rename("043:2,S", "043:2,R")
```

"Maildir/cur" directory



Process 1 Result

018:2,S
021:2,S
052:2,S
061:2,S

Message missing!

Return of The Coarse-grained File Locking

- Maildir synchronization
  - Lockless

    > *"certain anomalous situations may result"*
    >
    > – Courier IMAP manpage

  - File locks
    - Per-directory coarse-grained locking
    - Complexity of Maildir, performance of mbox
  - **System transactions**

Maildir Parallelized with System Transaction

PROCESS 1 (MARKING)

```
xbegin()
if (access("XXX:2,S")):
    rename("XXX:2,S",
   "XXX:2,R")
xend()
```

PROCESS 2 (MESSAGE LISTING)

```
xbegin()
while (f = readdir("Maildir/cur")):
    print f.name
xend()
```

Consistent directory accesses with better parallelism

Outline

- Background: system transaction
- System transactions in action
- Challenges for rewriting applications
- Implementation and evaluation

Challenges of Rewriting Applications

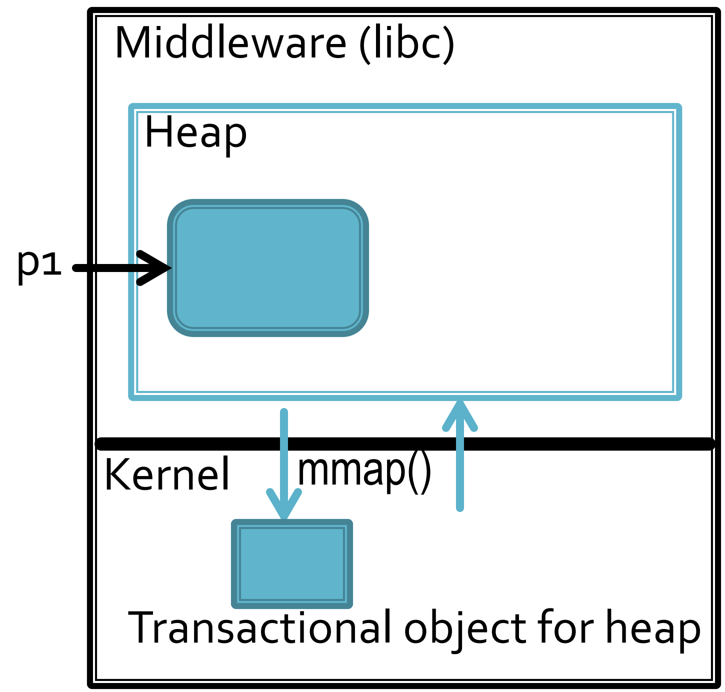
1. Middleware state sharing
2. Deterministic parallel update for system state
3. Composing with other synchronization primitives

Middleware and System Transaction

- Problem with memory management
  - Multiple threads share the same heap

In Transaction

Thread 1	Thread 2
→ xbegin();	
→ p1 = malloc();	
	p2 = malloc();
xabort();	
	*p2 = 1;

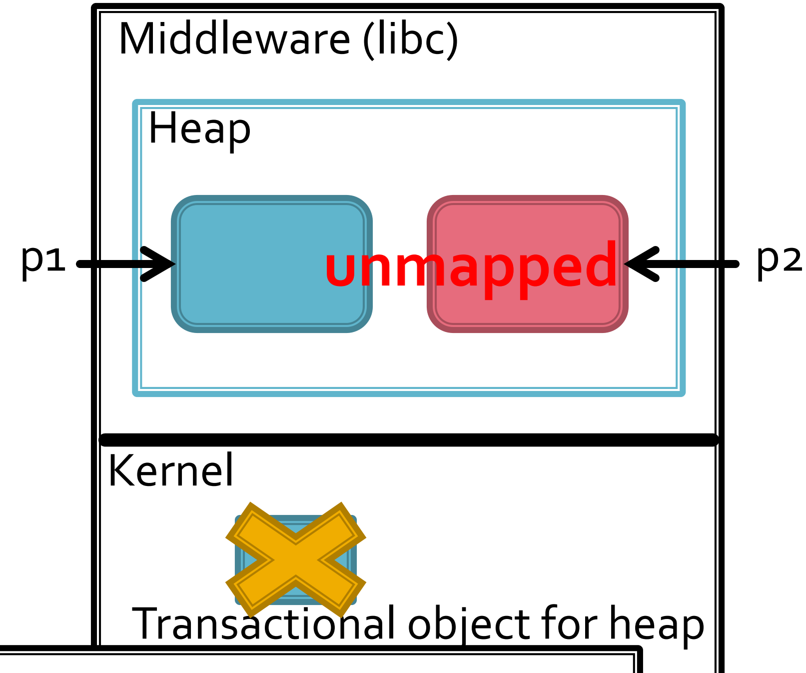


Middleware and System Transaction

- Problem with memory management
  - Multiple threads share the same heap

In Transaction

Thread 1	Thread 2
xbegin(); p1 = malloc();	
	→ p2 = malloc();
→ xabort();	
	→ *p2 = 1; **FAULT!**



Certain middleware actions should not roll back

Two Types of Actions on Middleware State

USER-INITIATED ACTION

- User changes system state
  - Most file accesses
  - Most synchronization

MIDDLEWARE-INITIATED

- System state changed as side effect of user action
  - malloc() memory mapping
  - Java garbage collection
  - Dynamic linking
- Middleware state shared among user threads
  - Can't just roll back!

Handling Middleware-Initiated Actions

- Transaction pause/resume
  - Expose state changes by **middleware-initiated actions** to other threads
  - Additional system calls
    - **xpause(), xresume()**
  - Limited complexity increase
    - We used pause/resume 8 times in glibc, 4 times in JVM
    - Only used in application for debugging

Pause/Resume In JVM Execution

Java code	JVM Execution
SysTransaction.begin();	xbegin();
files = dir.list();	files = dir.list();
	xpause()
	VM operations (garbage collection)
	xresume()
SysTransaction.end();	xend();

Other Challenges for Maturing TxOS

- 17,000 lines of kernel changes
  - Transactionalizing file descriptor table
  - Handling page lock for disk I/O
  - Memory protection
  - Optimization with directory caching
  - Reorganizing data structure
  - and more

- Details in the paper

Outline

- Background: system transaction
- System transactions in action
- Challenges for rewriting applications
- Implementation and evaluation

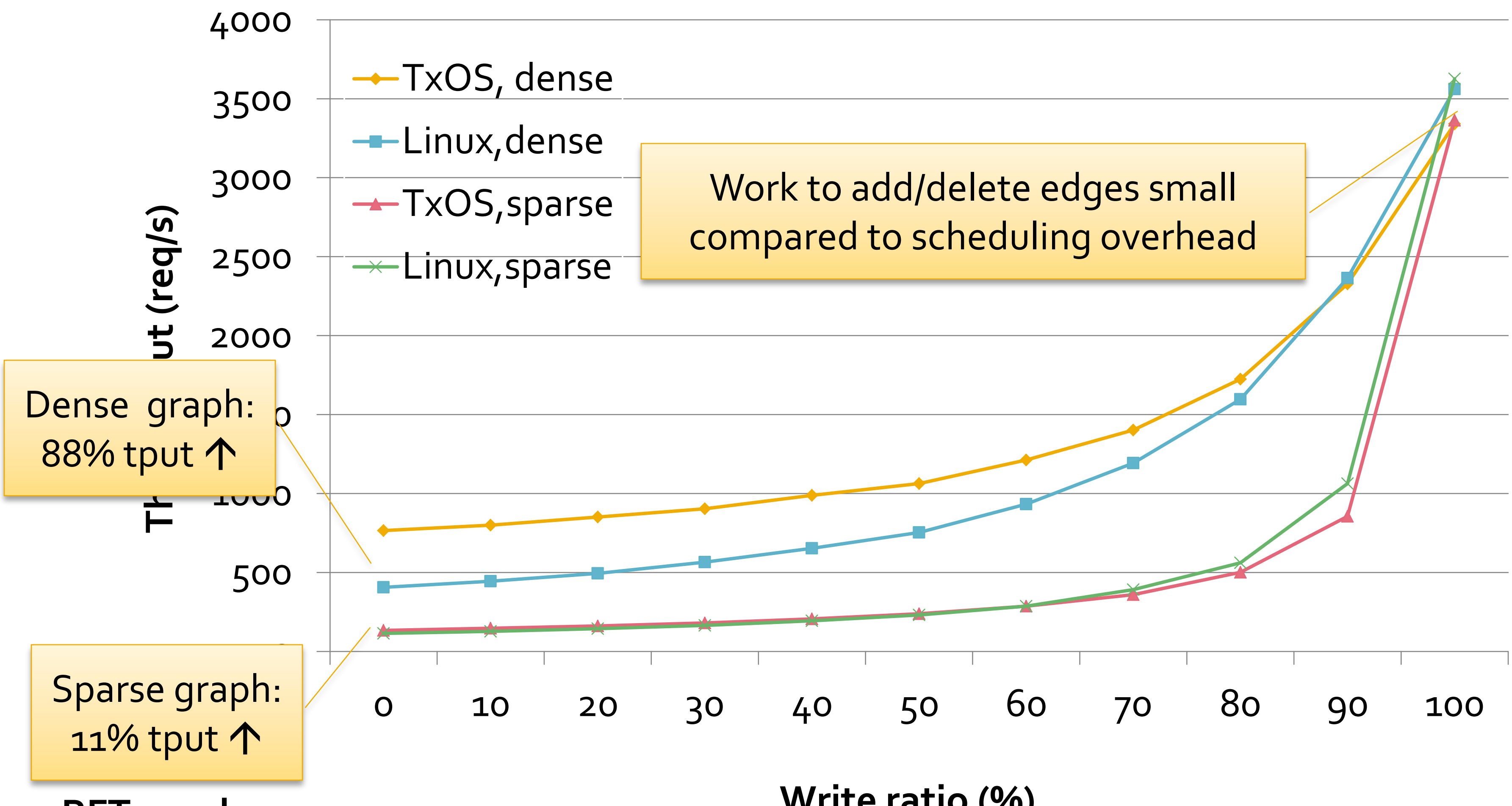
Application 1: Parallelized BFT Application

- Implemented in UpRight BFT library
- Fault tolerant routing backend
  - Graph stored in a file
  - Compute shortest path
  - Edge add/remove
- Ordered transactions for deterministic update

Minimal Application Code Change

Component	Total LOC	Changed LOC
Routing application	1,006	18 **(1.8%)**
Upright Library	22,767	174 **(0.7%)**
JVM	496,305	384 **(0.0008%)**
glibc	1,027,399	826 **(0.0008%)**

Deterministic State Update with Better Throughput



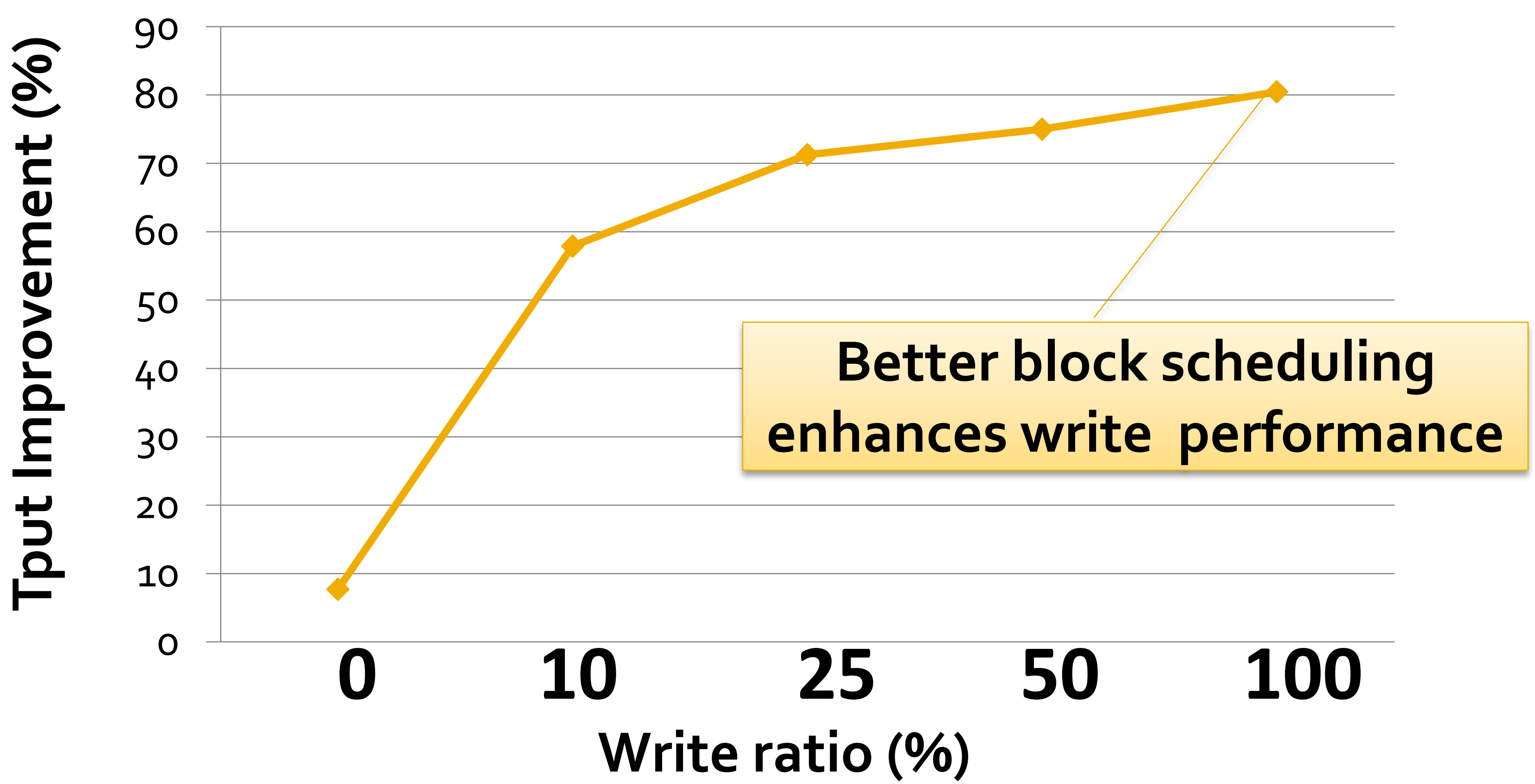
BFT graph server

Application 2: Dovecot Maildir access

- Dovecot mail server
  - Uses directory lock files for maildir accesses
- Locking is replaced with system transactions
  - Changed LoC: 40 out of 138,723
- Benchmark: Parallel IMAP clients
  - Each client executes operations on a random message
    - Read: message read
    - Write: message creation/deletion
    - 1500 messages total

Mailbox Consistency with Better Throughput

- Dovecot benchmark with 4 clients



Conclusion: OS Transactions Improve Server Performance

- System transactions parallelize tricky server applications
  - Parallel Dovecot maildir operations
  - Parallel BFT state update
- System transaction improves throughput with few application changes
  - Up to 88% throughput improvement
  - At most 40 changed lines of application code