Software Transactional Memory :

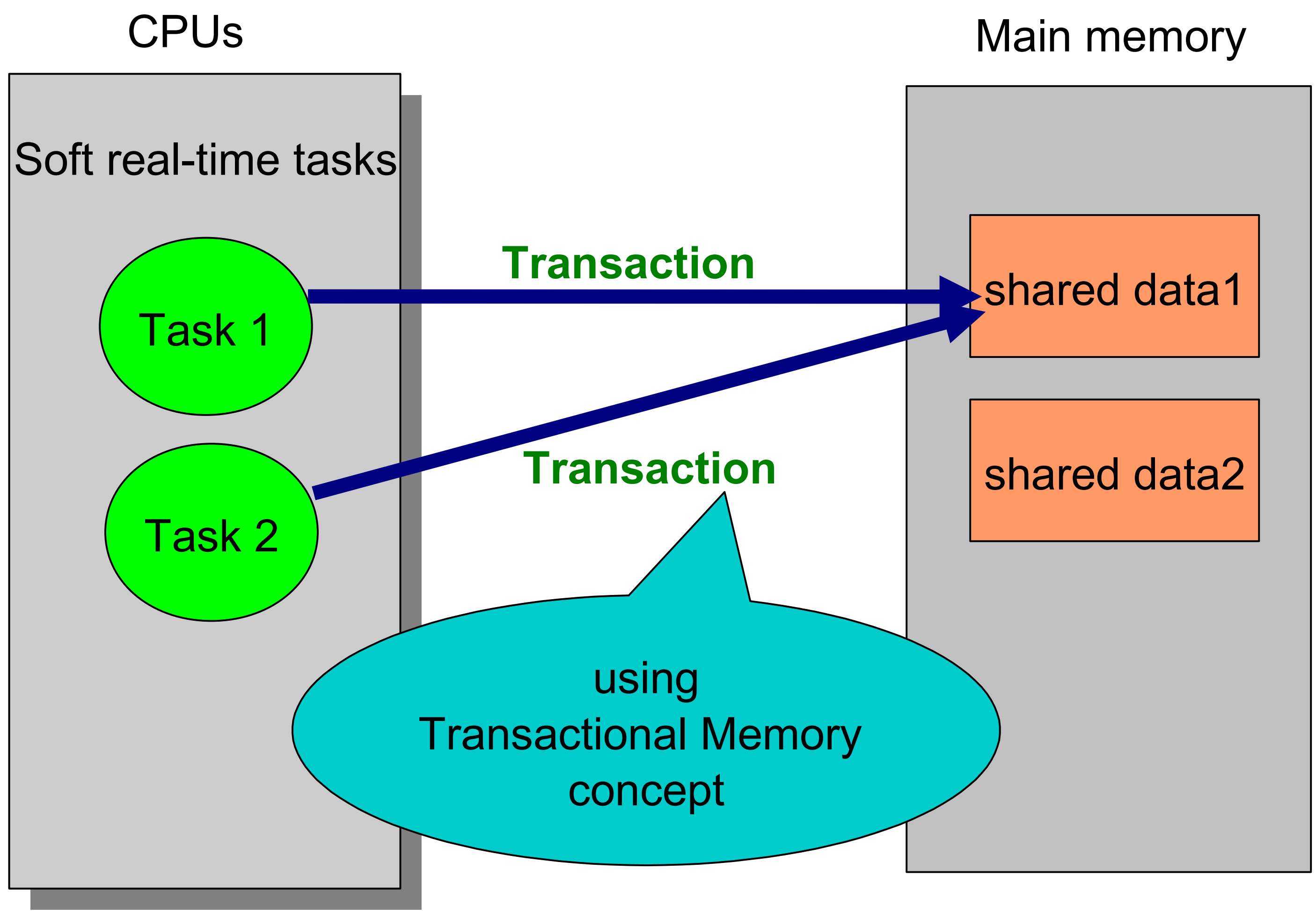
Worst Case Execution Time Analysis

T. Sarni, A. Queudet, P. Valduriez

Paris, October 2009

Outline of the talk

1. Introduction
2. Scheduling of transactions
3. Our contribution
4. Conclusion



Concurrent real-time tasks accessing multiple shared resources on multiprocessor systems

Framework Basis

Why transactions instead of locks ?

- Eases programming and avoids problems like deadlocks, priority inversion ...

```
Init Semaphores : S1, S2

P(S1)
 P(S2)

    <instructions>

 V(S1)
V(S2)
```

Lock-based method

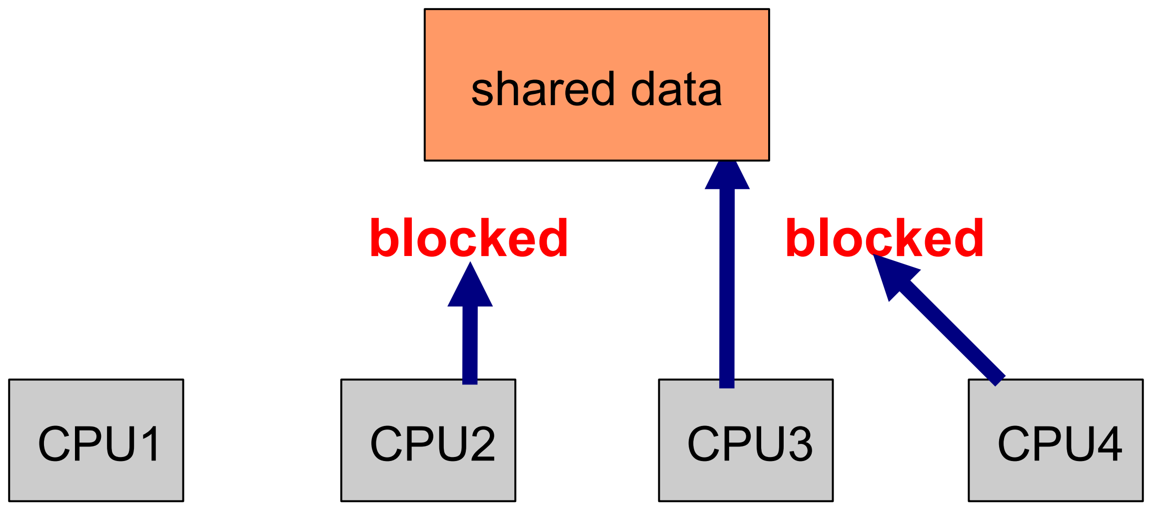
```
do{

 <instructions>

}until transaction_commit ()
```

Transaction

Framework Basis

Why transactions instead of locks ?

- Eases programming and avoids problems like deadlocks, priority inversion ...
- Locks reduce throughput in multiprocessor systems

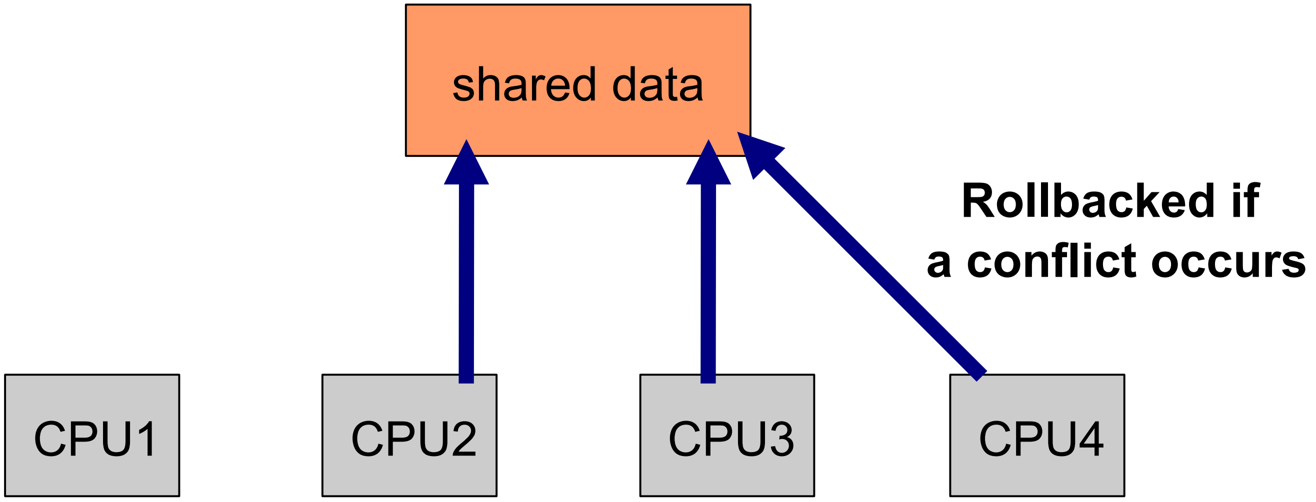


Lock-based access

Framework Basis

Why transactions instead of locks ?

- Eases programming and avoids problems like deadlocks, priority inversion ...
- Locks reduce throughput in multiprocessor systems



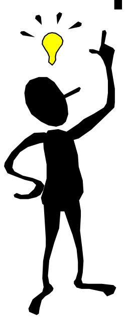
Transactions-based access

Framework Basis

Why transactions instead of locks ?

- Eases programming and avoids problems like deadlocks, priority inversion ...
- Locks reduce throughput in multiprocessor systems

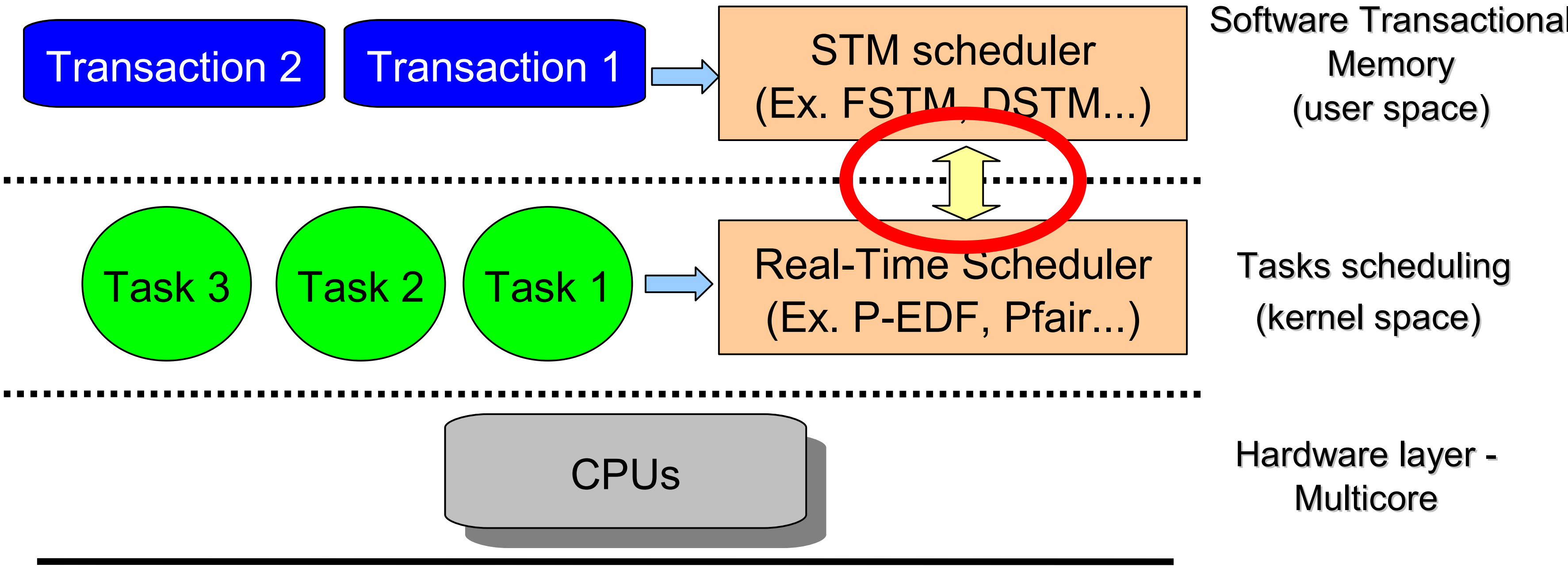
Transactional Memory (TM)



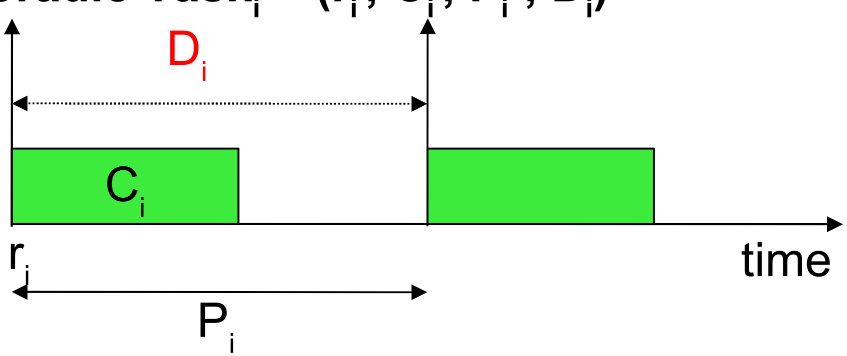
- The term TM is introduced by Herlihy *et al*. 93

- Interest renewed with advent of multicore systems
  - **Hardware-based (HTM)**
  - **Software-based (STM)**
  - **Hybrid-based (HyTM)**

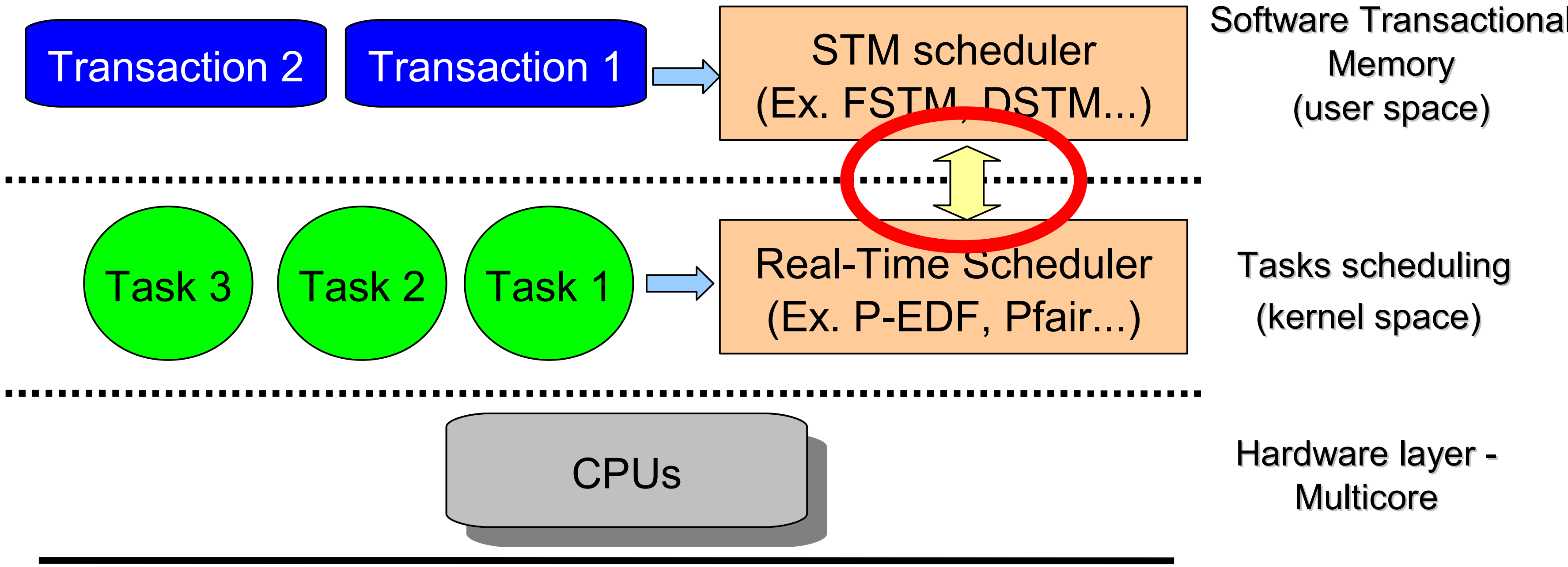
Issue

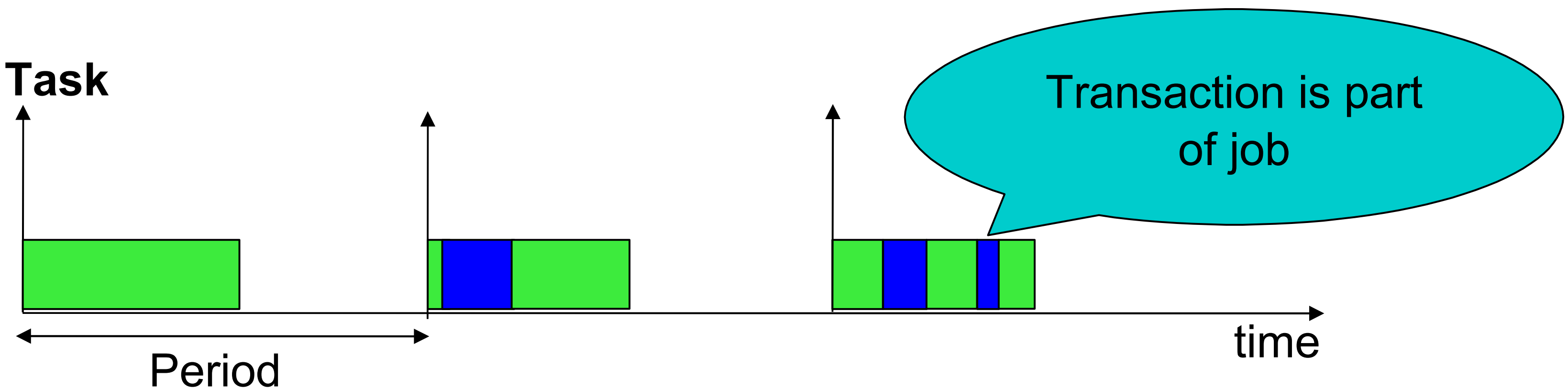


Sporadic Task$_i$ = (r_i , C_i , P_i , D_i)



Issue


Transaction 2
Transaction 1
STM scheduler
(Ex. FSTM, DSTM...)
Software Transactional Memory
(user space)
Task 3
Task 2
Task 1
Real-Time Scheduler
(Ex. P-EDF, Pfair...)
Tasks scheduling
(kernel space)
CPUs
Hardware layer - Multicore


Task
Transaction is part of job
time
Period

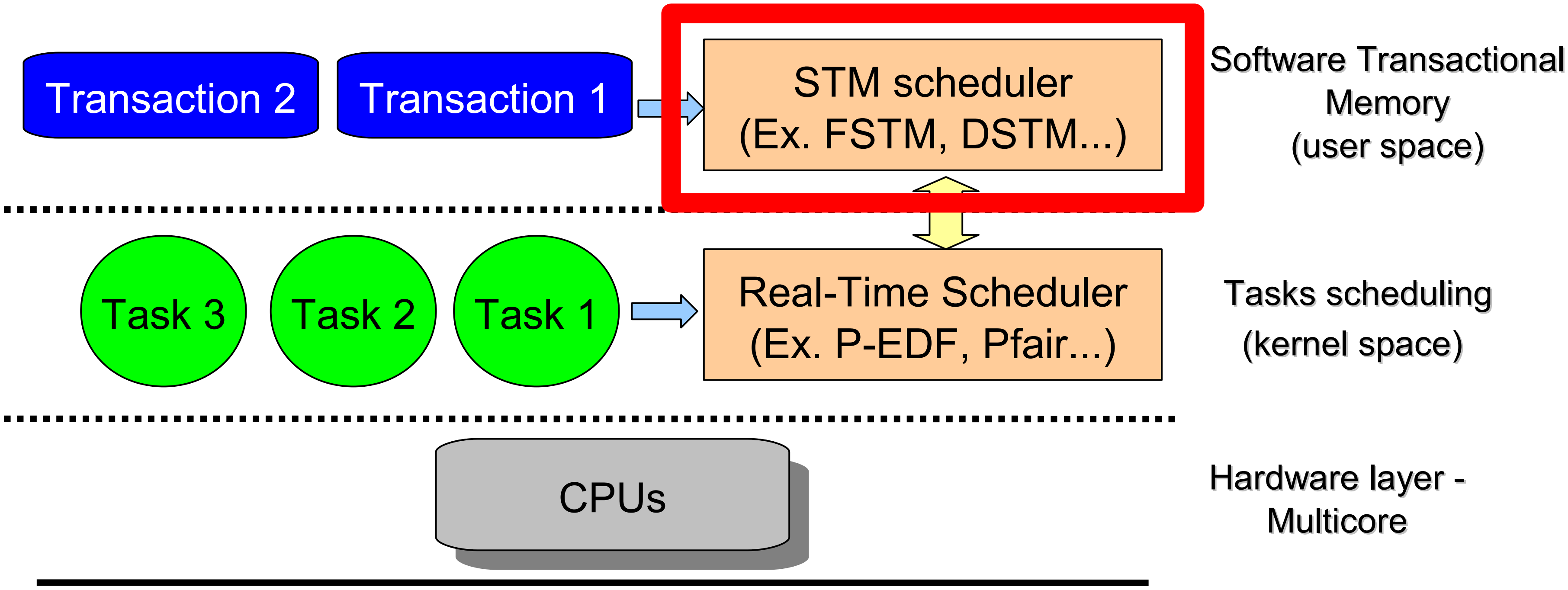
Outline of the talk

1. Introduction

2. Scheduling of transactions

3. Our contribution

4. Conclusion


Transaction 2
Transaction 1
STM scheduler
(Ex. FSTM, DSTM...)
Software Transactional Memory (user space)
Task 3
Task 2
Task 1
Real-Time Scheduler
(Ex. P-EDF, Pfair...)
Tasks scheduling (kernel space)
CPUs
Hardware layer - Multicore

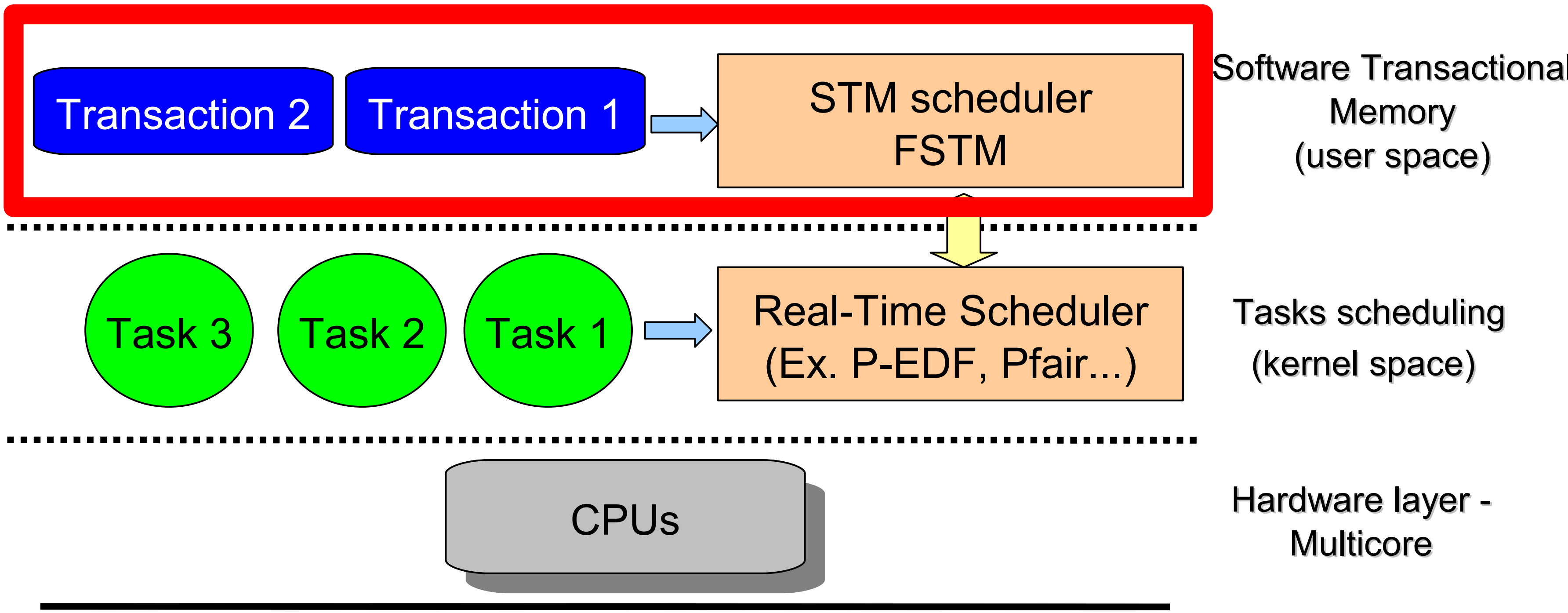
Classification

Transaction progress guarantuees

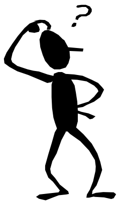
- **Wait-free**
  - Strongest guarantee for transactions to make progress
  - Hard real-time
- **Lock-free**
  - Ensures that at least one transaction is making progress
  - Soft real-time
- **Obstruction-free**
  - Weakest guarantee for transaction to make progress
  - Non real-time

Outline of the talk

1. Introduction
2. Scheduling of transactions
3. Our contribution
4. Conclusion


Transaction 2
Transaction 1
STM scheduler
FSTM
Software Transactional
Memory
(user space)
Task 3
Task 2
Task 1
Real-Time Scheduler
(Ex. P-EDF, Pfair...)
Tasks scheduling
(kernel space)
CPUs
Hardware layer -
Multicore

Studying STM



Which parts cause execution time variation ?

- Intuitively : The number of transaction retries (rollback times)

  But STMs are usually based upon a dynamic memory allocator ...

Impact of task scheduling on the execution time of transactions ?

- Evaluation of rollback and objects allocation times under : P-EDF, G-EDF, Pfair

Performance metrics

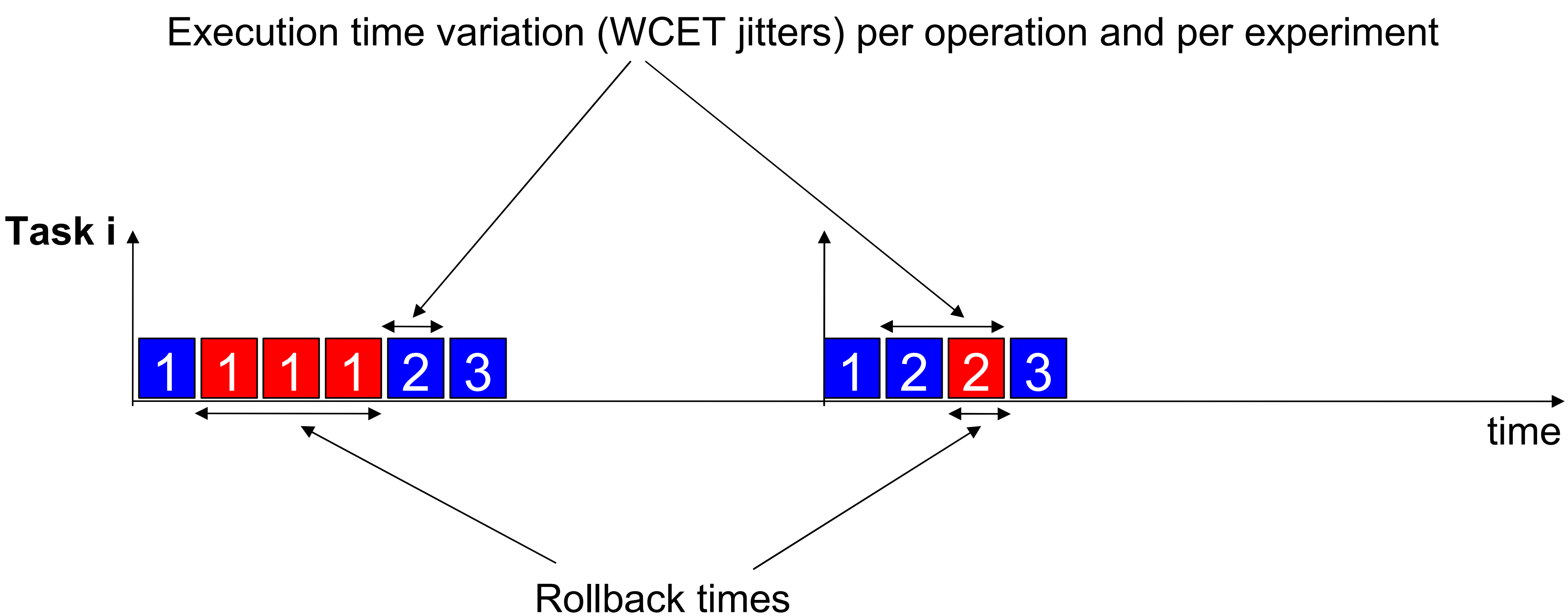
Rollback times

- Measuring the rollback time ratio for the whole set of tasks
- Global mean

WCET jitters

- **Observed** WCET jitter over 10 experiments

Performance metrics (example)



Legend

Red-black tree operations : 1 **Lookup** 2 **Update** 3 **Remove**

x Retry (=rollback)

Experimental context

Real-Time Operating System

- We choose LITMUS^{RT} (developed at University of North Carolina)
- LITMUS^{RT} is a Linux-based kernel
  - °Implements real-time multiprocessor scheduling policies

Software Transactional Memory

- Enhanced Fraser's STM for real-time [*Sarni et al*. RTCSA'09]
- Based on a lock-free algorithm

Experimental context

- Using a x86 Dual-core processor (m = 2)
- Under Ubuntu 8.04
- *N* tasks (C_i = ***20ms*** , $P_i = (N * C_i) / m$, $D_i = P_i$) => **heavy loads** (in order to obtain a high contention for shared resources accesses)

Experimental context

Dlmalloc (Linux allocator)

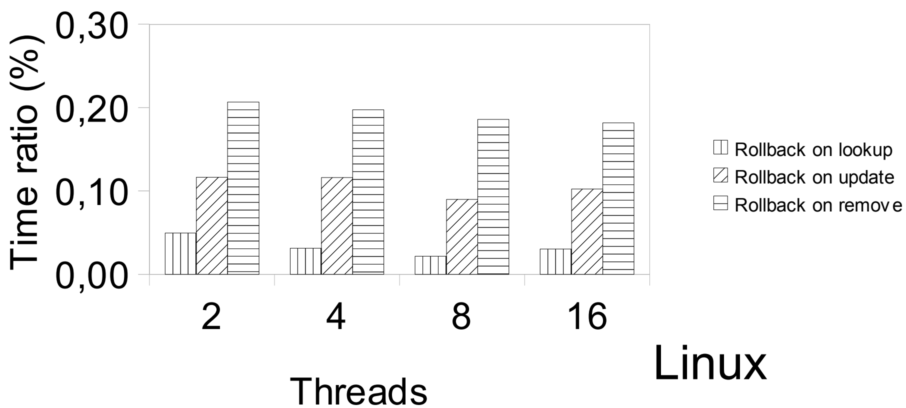
- Allocation $O(M/m)$ **M**: Maximum memory size, **m**: Largest allocated block
- Deallocation $O(1)$

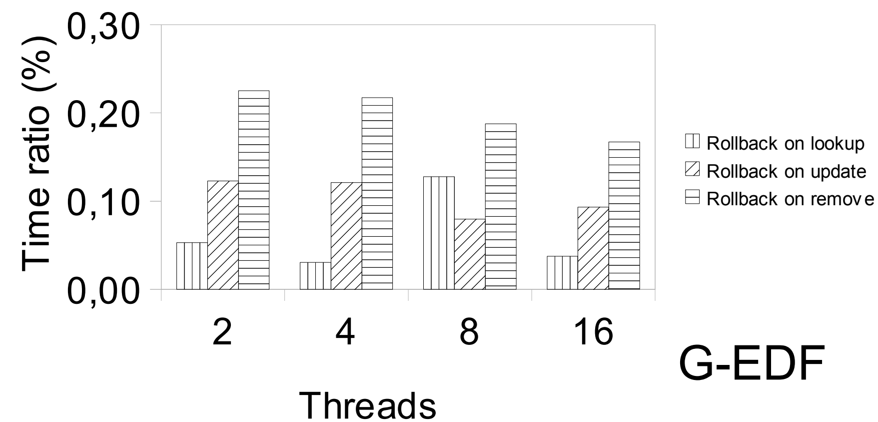
TLSF (Two-Level Segregate Fit) [Masmano *et al.* 04]

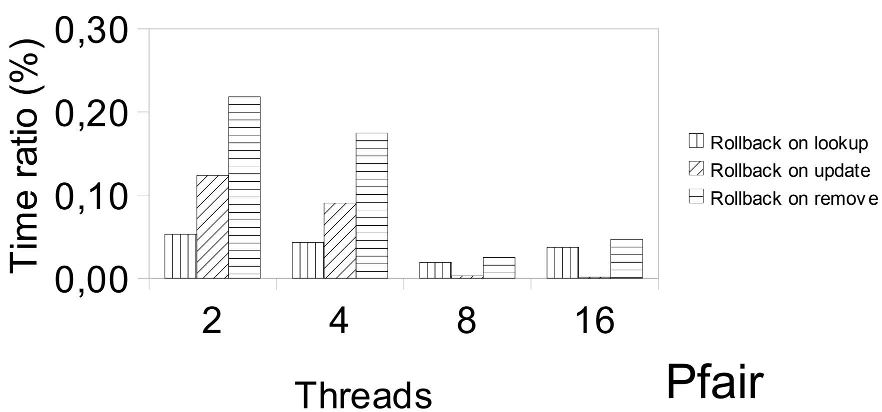
- Dynamic memory allocator
- Based on an algorithm with constant cost $\Theta(1)$
- Therefore, reasonable use in real-time context

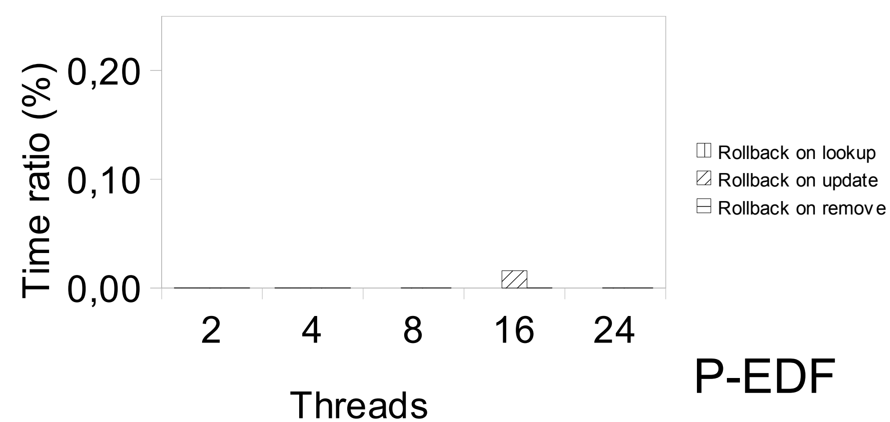
Experiment results

Rollback times



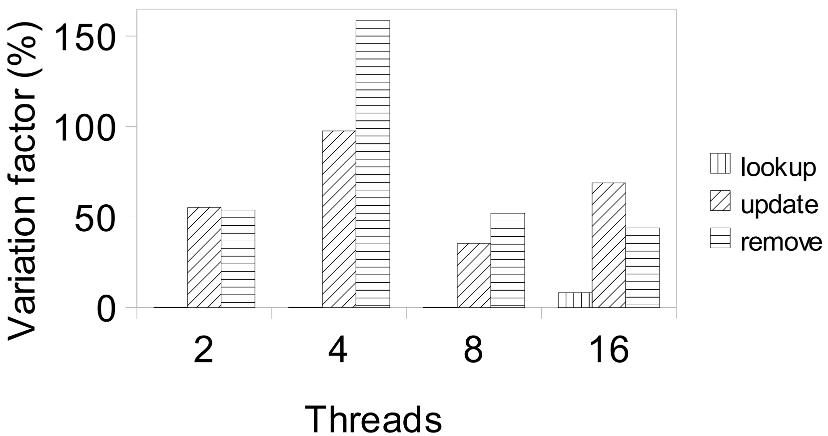




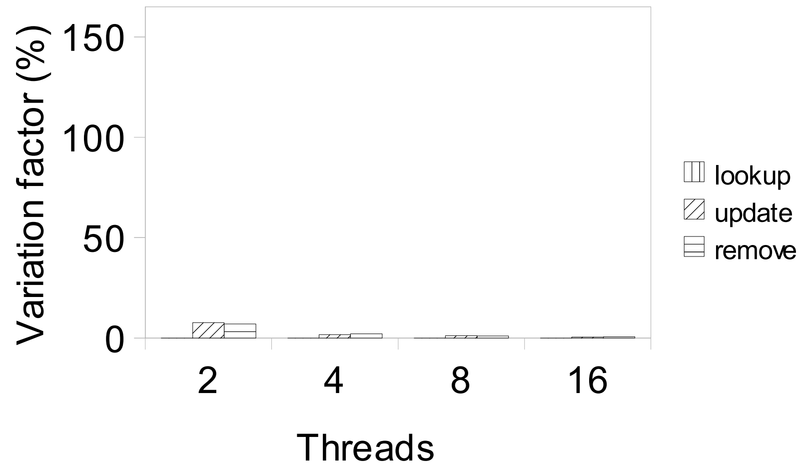


Experiment results

WCET Jitters

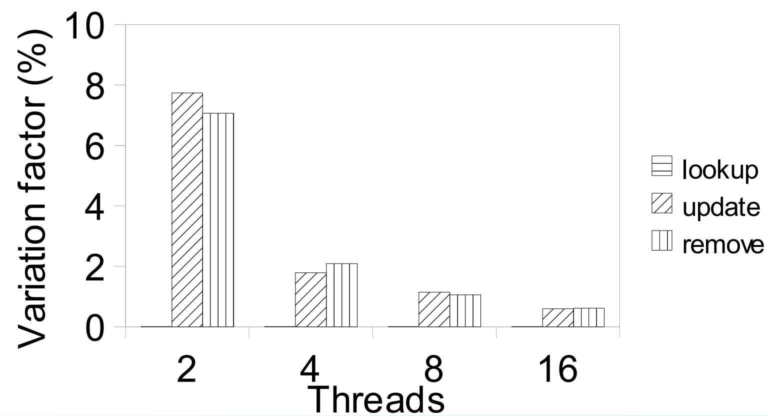


Using classical malloc (P-EDF)



Using TLSF (P-EDF)

Zoom (TLSF)



Outline of the talk

1. Introduction
2. Scheduling of transactions
3. Our contribution
4. Conclusion

Based on a <u>real platform</u> and for soft real-time constraints :

- In STM and **for real applications**, the rollback times are not the main cause of the execution time variation,
- A good memory allocator must be provided to bound the execution time of transactions

TO DO **Future work**

Study of the interaction between the real-time scheduler of transactions and that of tasks

Questions?