departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Concurrency and Parallelism
(Concorrência e Paralelismo – CP 11158)

Lecture 8
— Transactional Memory —

Slides based in material from:
Maurice Herlihy & Nir Shavit, and Guy Korland

Outline

- **Motivation**
- Deuce
- Implementation
- TL2
- LSA
- Benchmarks
- Summary
- References

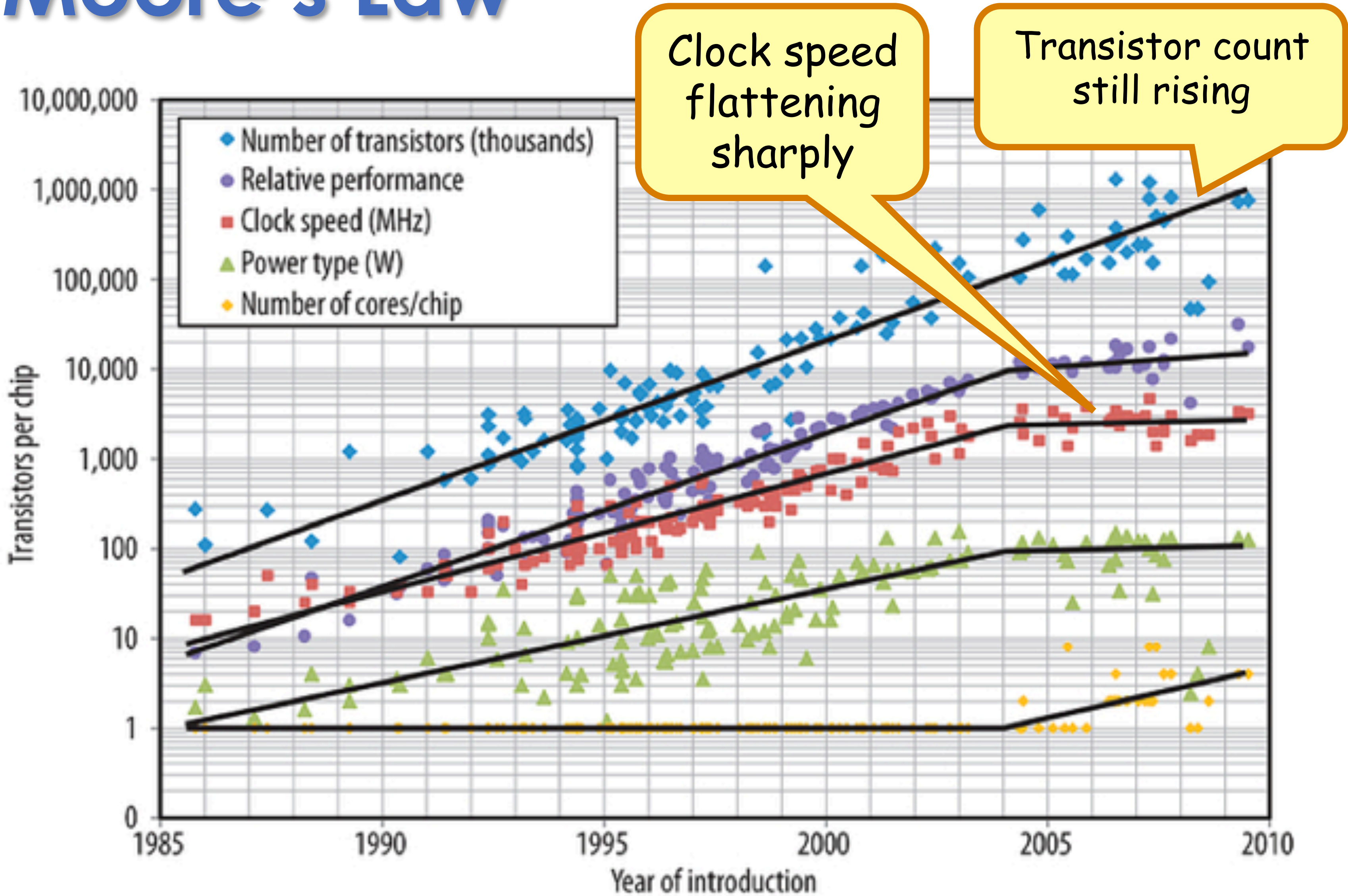
Motivation

From the New York Times …

May 7th, 2004 – "Intel said on Friday that it was **scrapping its development of a faster Pentium 4 to focus on "dual core" microprocessors**, a move that is a shift in the company's business strategy…."

May 8th,2004 – "Intel … [has] decided to **focus its development efforts on "dual core" processors** …. with two engines instead of one, allowing for greater efficiency because the processor workload is essentially shared."

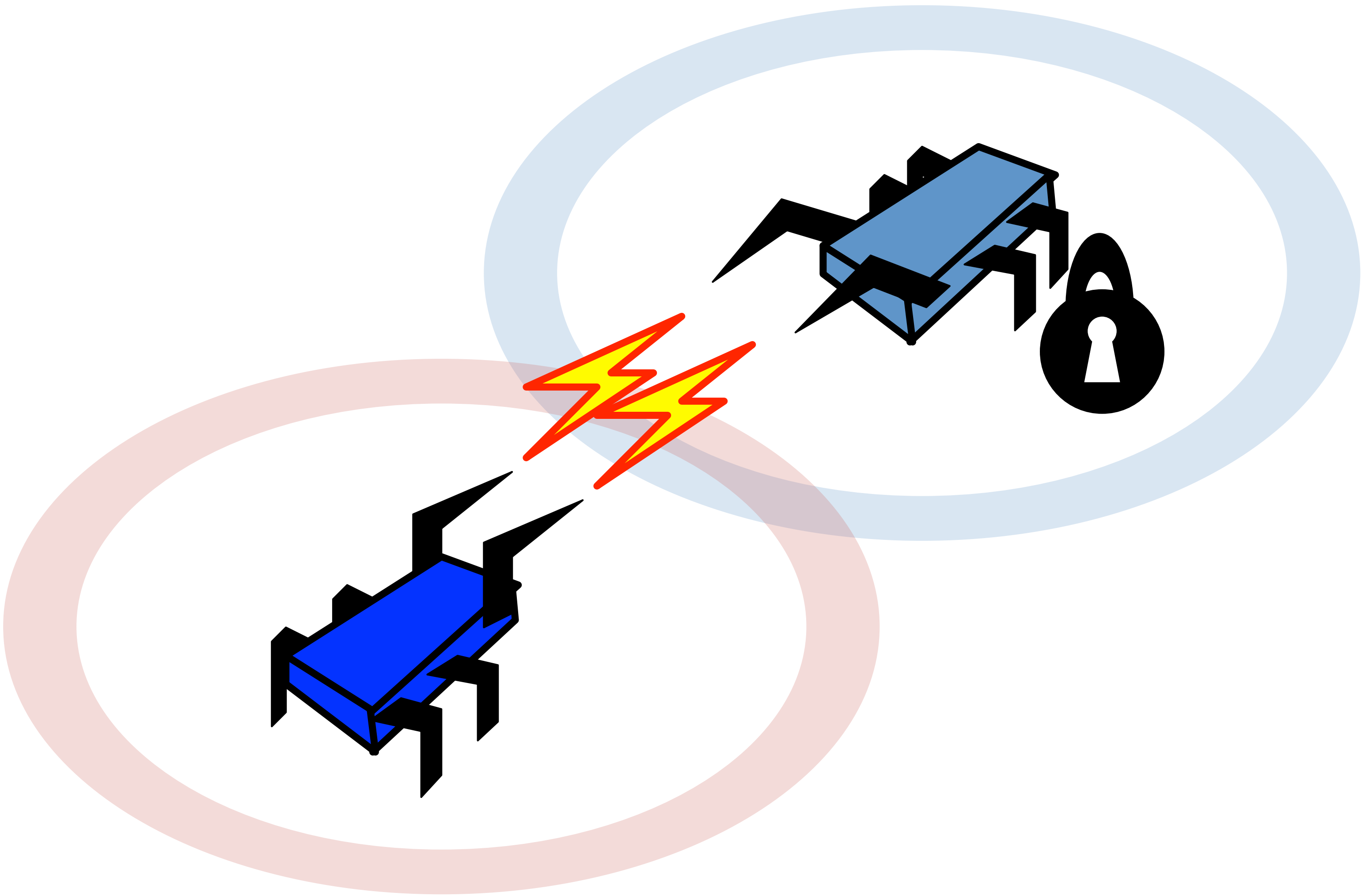
Moore's Law


Clock speed flattening sharply
Transistor count still rising
Number of transistors (thousands)
Relative performance
Clock speed (MHz)
Power type (W)
Number of cores/chip
10,000,000
1,000,000
100,000
10,000
1,000
100
10
1
0
Transistors per chip
1985
1990
1995
2000
2005
2010
Year of introduction

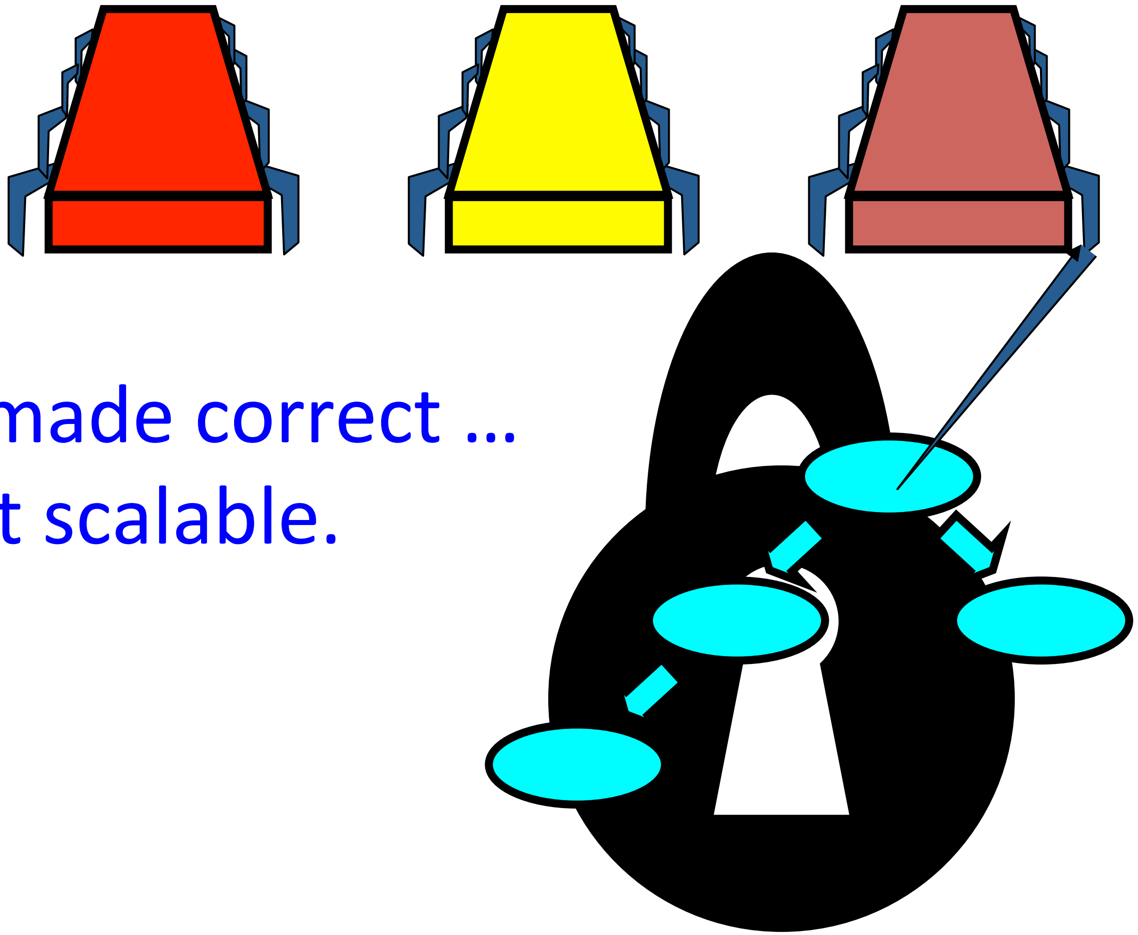
The Problem

- Cannot exploit cheap threads
- Today's Software
  - Non-scalable methodologies
- Today's Hardware
  - Poor support for scalable synchronization

Locking

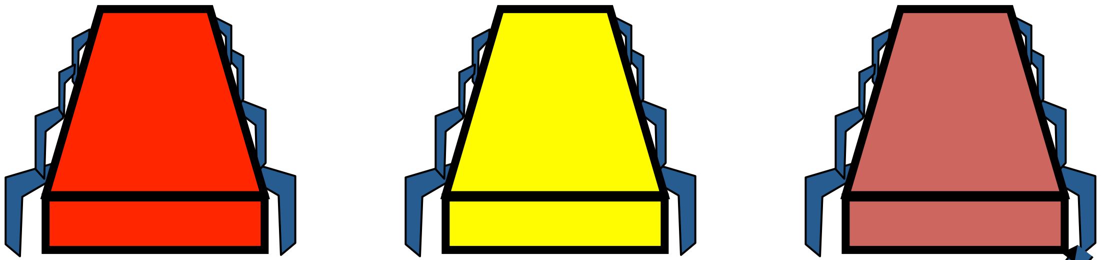


Coarse-Grained Locking

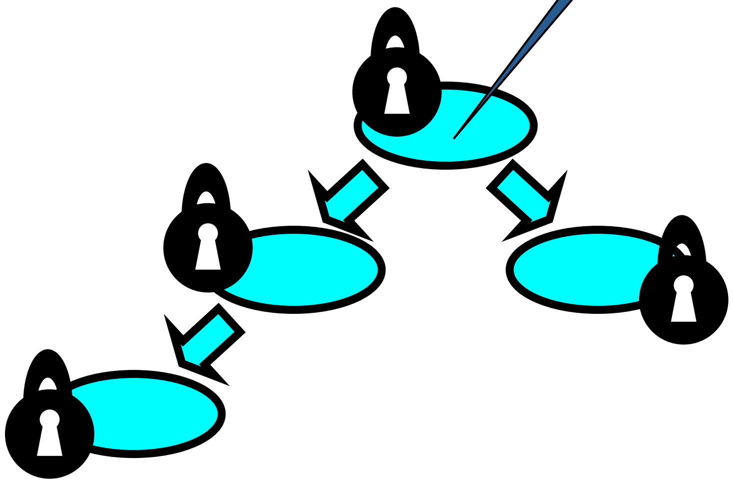


Easily made correct …
But not scalable.

Fine-Grained Locking



Here comes trouble …

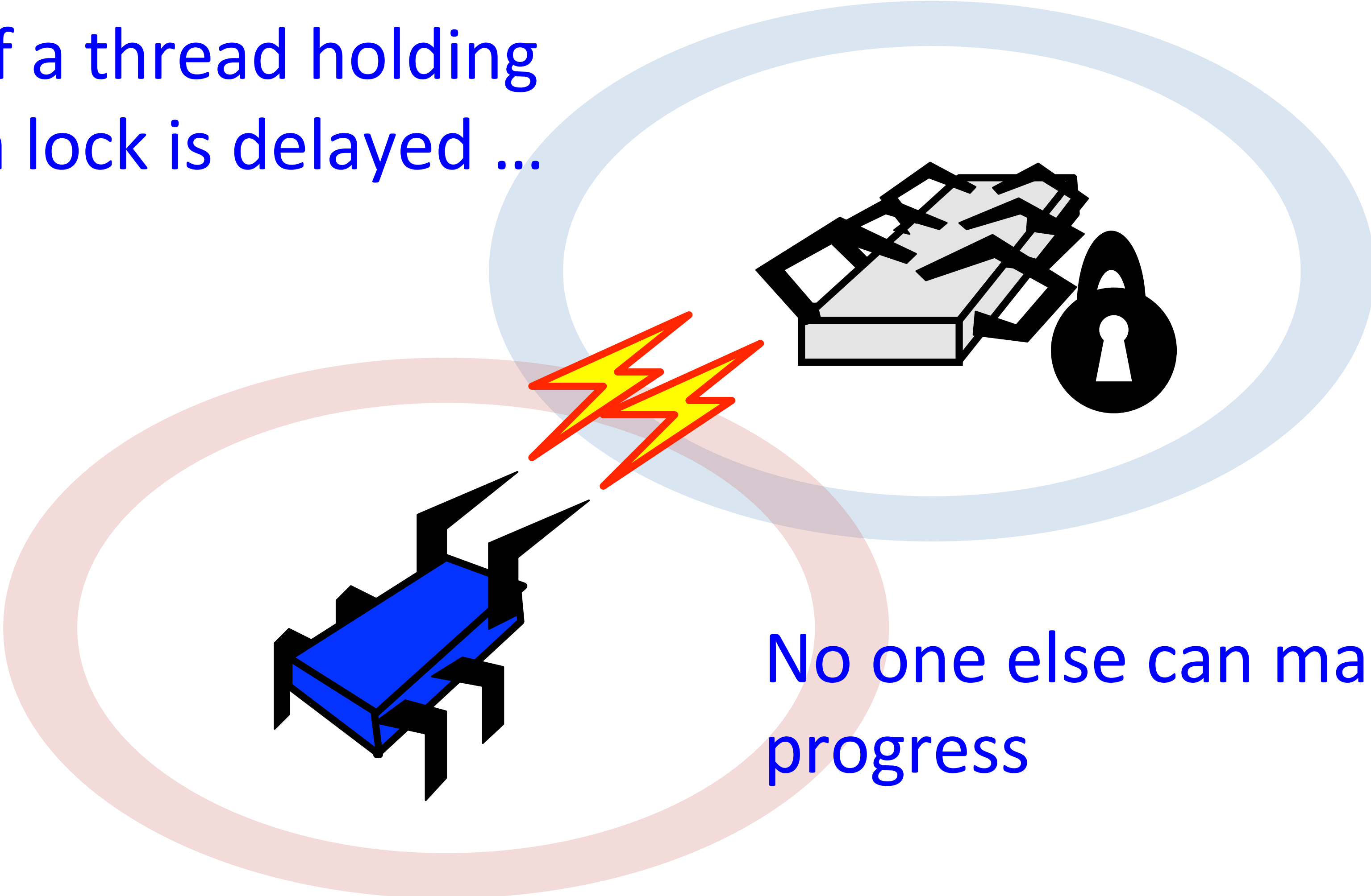


Why Locking Doesn't Scale?

- Not Robust
- Relies on conventions
- Hard to Use
  - Conservative
  - Deadlocks
  - Lost wake-ups
- Not Composable

Locks are not Robust

If a thread holding
a lock is delayed …



No one else can make
progress

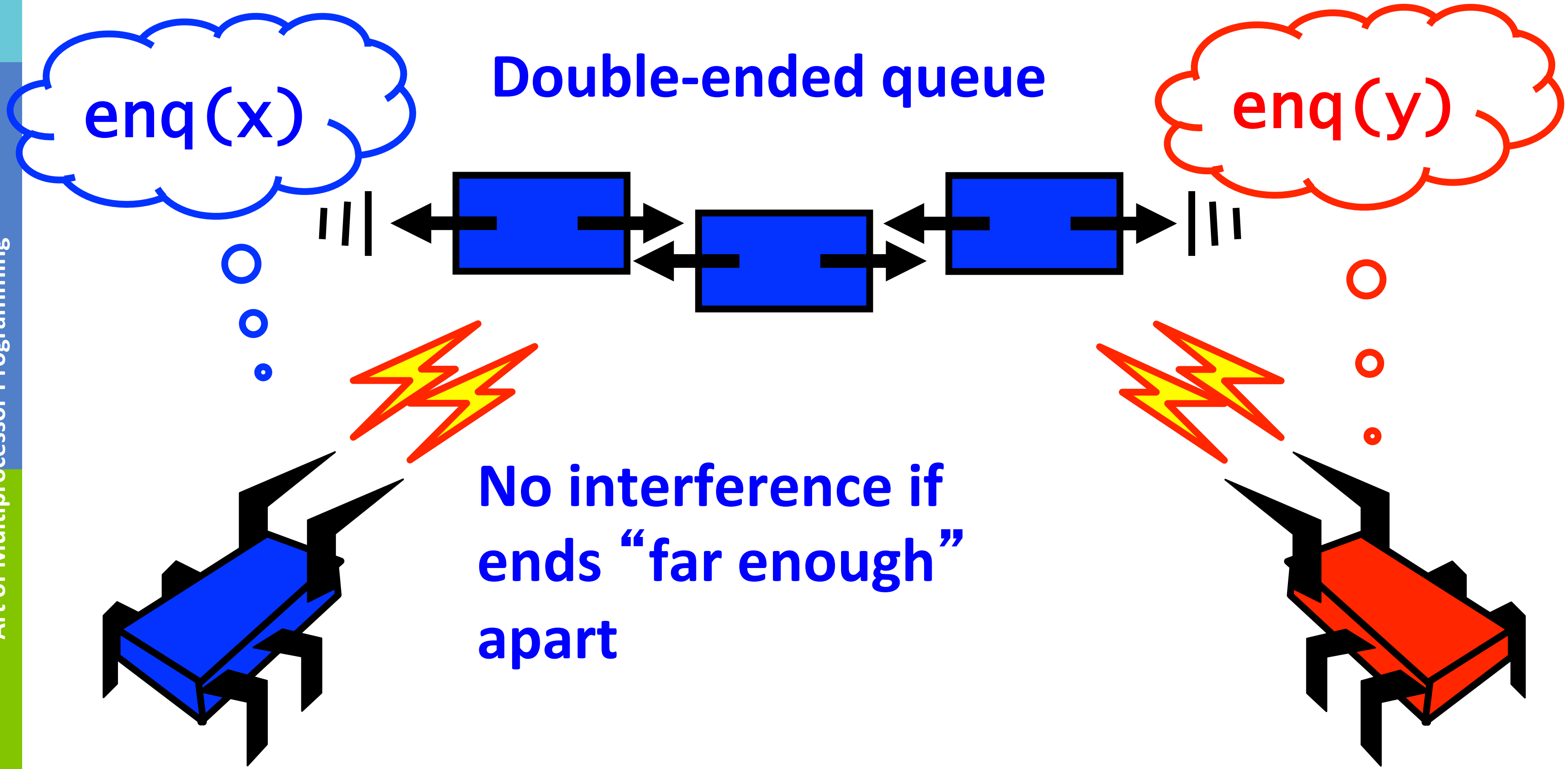
Locking Relies on Conventions

- Relation between
  - Lock bit and object bits
  - Exists only in programmer's mind

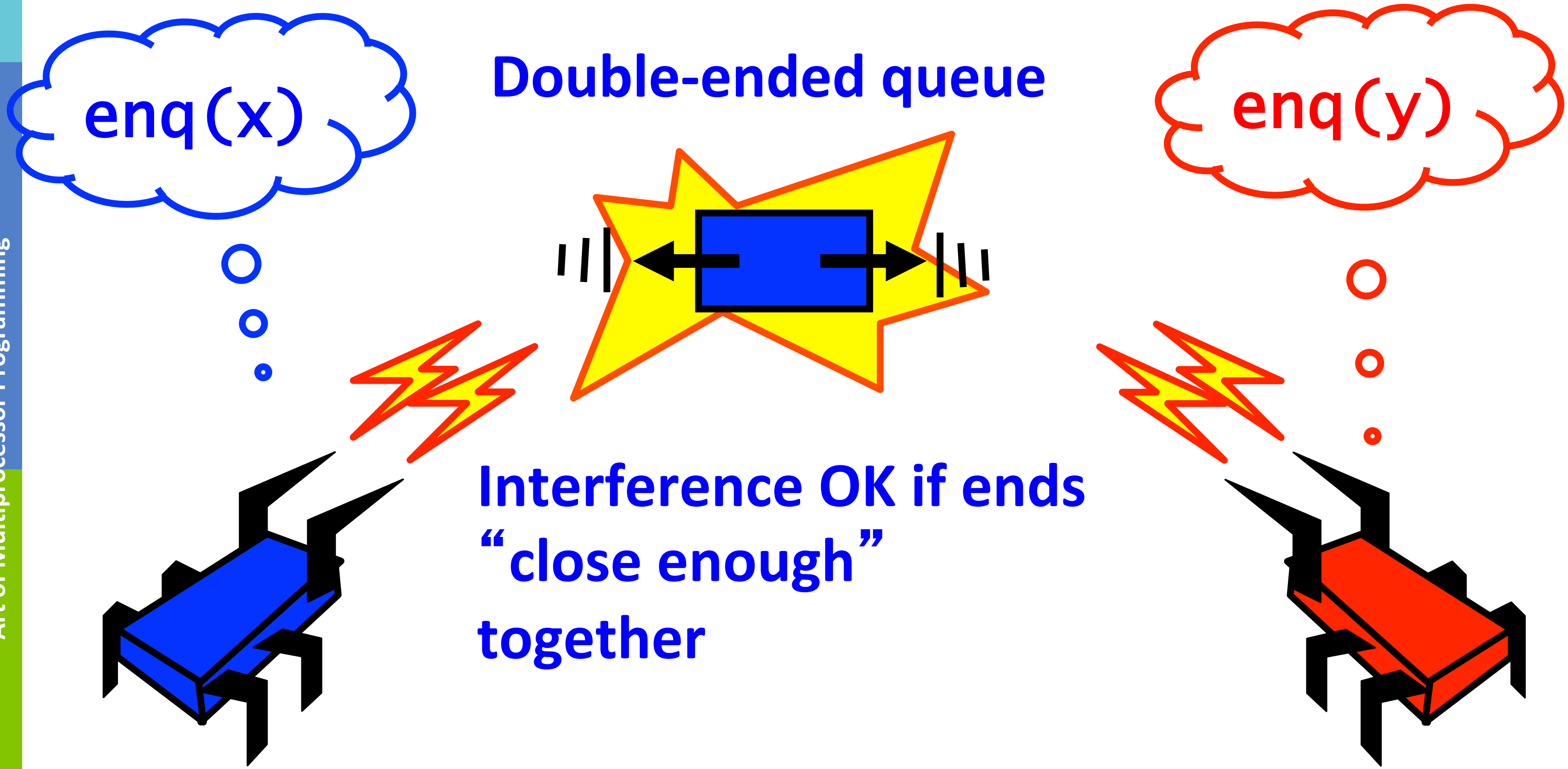
Actual comment from Linux Kernel
(hat tip: Bradley Kuszmaul)

```
/*
 * When a locked buffer is visible to the I/O layer
 * BH_Launder is set. This means before unlocking
 * we must clear BH_Launder,mb() on alpha and then
 * clear BH_Lock, so no reader can see BH_Launder set
 * on an unlocked buffer and then risk to deadlock.
 */
```

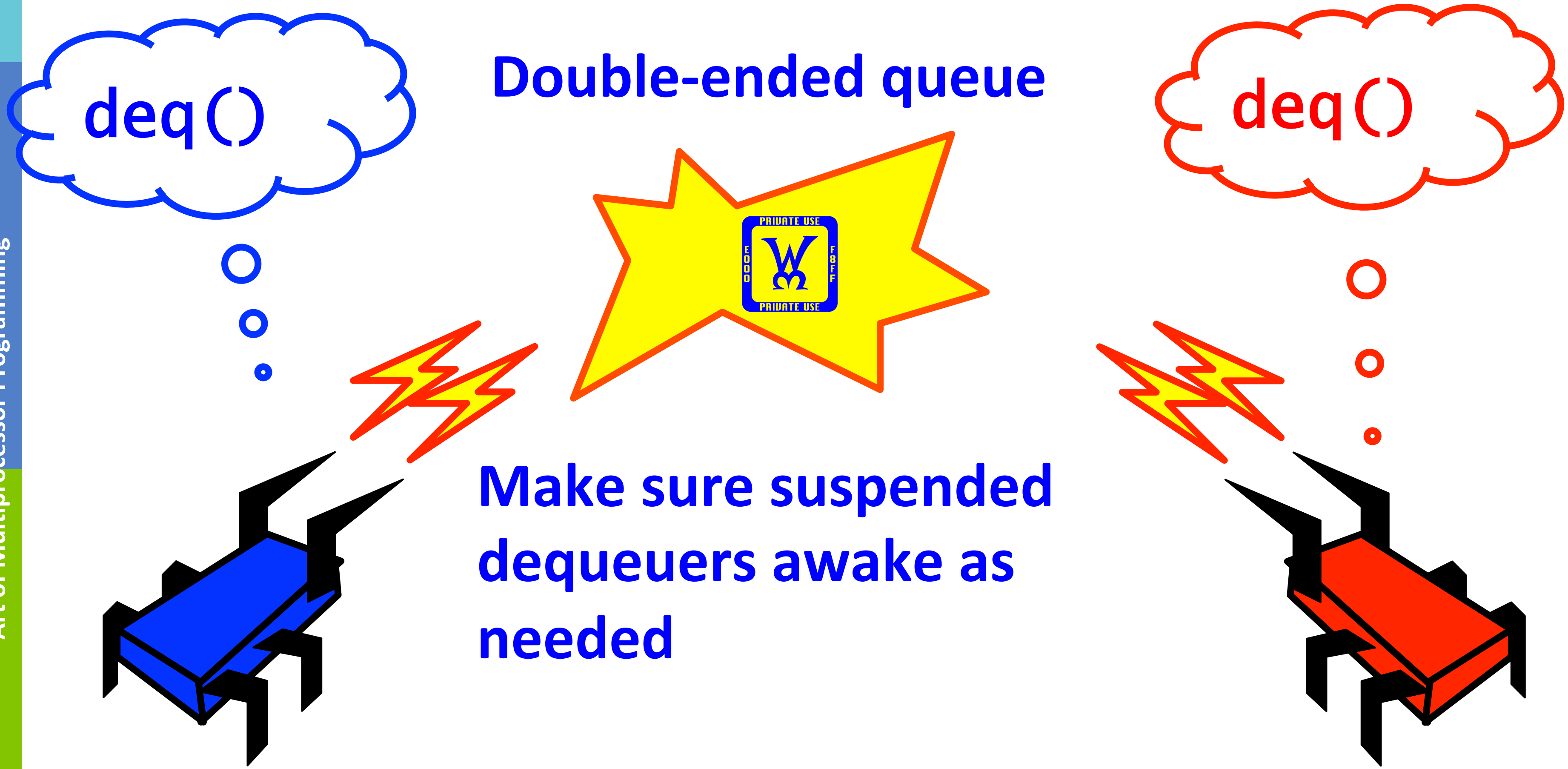
Locks are hard to use



Locks are hard to use



Locks are hard to use



departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

You Try It …

- One lock?
  - Too Conservative
- Locks at each end?
  - Deadlock, too complicated, etc
- Waking blocked dequeuers?
  - Harder that it looks

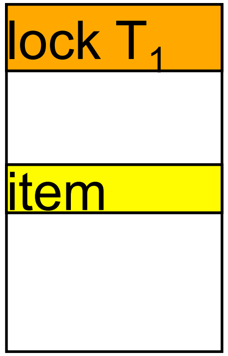
Actual Solution

- Clean solution would be a publishable result
- [Michael & Scott, PODC 96]
- What good is a methodology where solutions to such elementary problems are hard enough to be publishable?

Locks do not compose

Hash Table

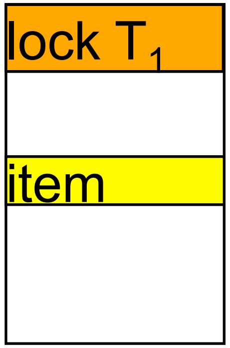
add(T_1, item)

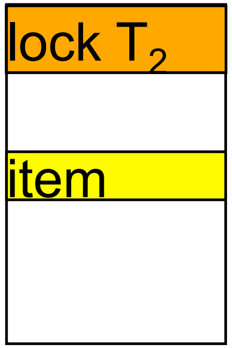


Must lock T_1 before adding item

Move from T_1 to T_2

delete(T_1, item)
add(T_2, item)





Must lock T_2 before deleting from T_1

Exposing lock internals breaks abstraction

The Transactional Manifesto

- What we do now is inadequate to meet the multicore challenge
- Research Agenda
  - Replace locking with a transactional API
  - Design languages to support this model
  - Implement the run-time to be fast enough

Transactions

- Atomic
  - Commit: takes effect
  - Abort: effects rolled back
    - Usually retried
- Linearizable
  - Appear to happen in one-at-a-time order

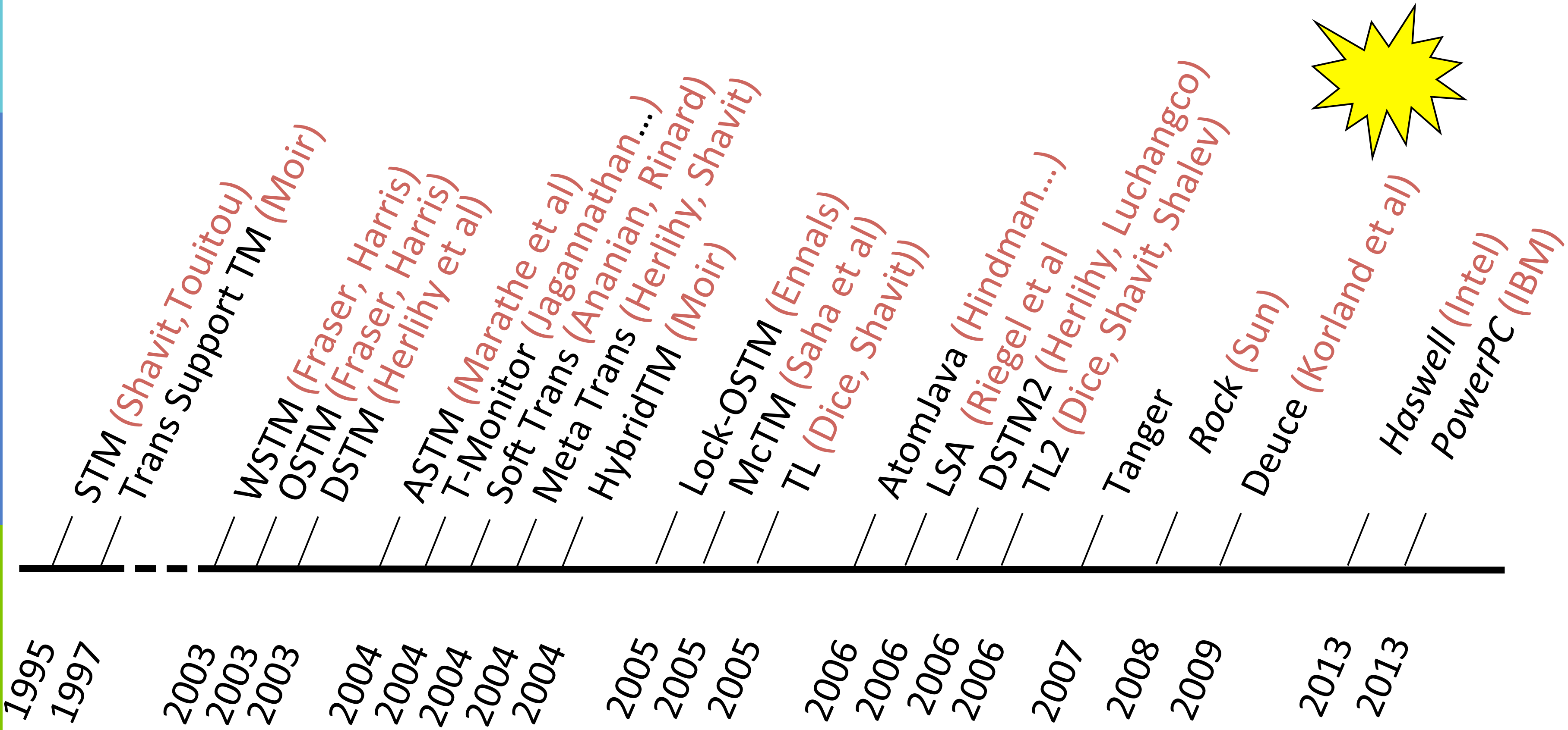
departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Transactional Memory History

- In 1977, Lomet proposed the idea to **map database transaction into programming language**, but no implementation
- In 1993, Herlihy and Moss proposed **hardware supported transactional memory**
- In 1993, Stone et al. proposed an **atomic multi-word operation** known as "Oklahoma Update"
- In recent years, a huge ground swell of **interest in both hardware and software systems** for implementing **transactional memory**.

departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

The Brief History of (S)TM



departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

TM: Overview

synchronized {

    <sequence of instructions>

}

atomic {

    <sequence of instructions>

}

Locks are hard to use – Revisited

```
Public void LeftEnq(item x) {
   Qnode q = new Qnode(x);
   q.left = this.left;
   this.left.right = q;
   this.left = q;
}
```

Write sequential Code

departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Locks are hard to use – Revisited

```
Public void LeftEnq(item x) {
 atomic {
   Qnode q = new Qnode(x);
   q.left = this.left;
   this.left.right = q;
   this.left = q;
 }
}
```

departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Locks are hard to use – Revisited

```
Public void LeftEnq(item x) {
 atomic {
  Qnode q = new Qnode(x);
  q.left = this.left;
  this.left.right = q;
  this.left = q;
 }
}
```

Enclose in atomic block

Warning

- Not always this simple
  - Conditional waits
  - Enhanced concurrency
  - Overlapping locks
- But often it is
  - Works for sadistic homework

departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Composition

```
Public void Transfer(Queue q1, q2)
{
 atomic {
  Item x = q1.deq();
  q2.enq(x);
 }
}
```

Trivial or what?

departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

What is a transaction?

- **A**tomicity – all or nothing
- **C**onsistency – consistent state (after & before)
- **I**solation – Other can't see intermediate.
- ~~Durability – persistent~~

Do Orphan (Zombie) Transactions Always See Consistent States?

- Yes!
  - Invariants observed (no surprises)
  - Expensive (maybe)
- No!
  - Who cares about surprises?
    - Divide by zero, infinite loops, et cetera …
    - Use exception/interrupt handlers?
  - More efficient (maybe)

Read Synchronization

- Visible (mark objects)
  - Consistent views
  - Strong contention management
  - Quick validation
  - Slower overall (maybe)

departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Read Synchronization

- Invisible (no footprint)
  - Inconsistent views
  - Weaker contention management
  - Slow validation
  - Faster overall (maybe)

departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Recovery

- Undo logs
  - Update in place
  - Reads are fast
  - Rolling back wedged transaction complex
- Redo logs
  - Apply changes on commit
  - Reads require look-aside
  - Rolling back wedged transaction easy

Other Differences

- Levels of indirection
- Compatibility with HTM
- Contention management policies
- There's lots more …

Outline

- Motivation
- **Deuce**
- Implementation
- TL2
- LSA
- Benchmarks
- Summary
- References

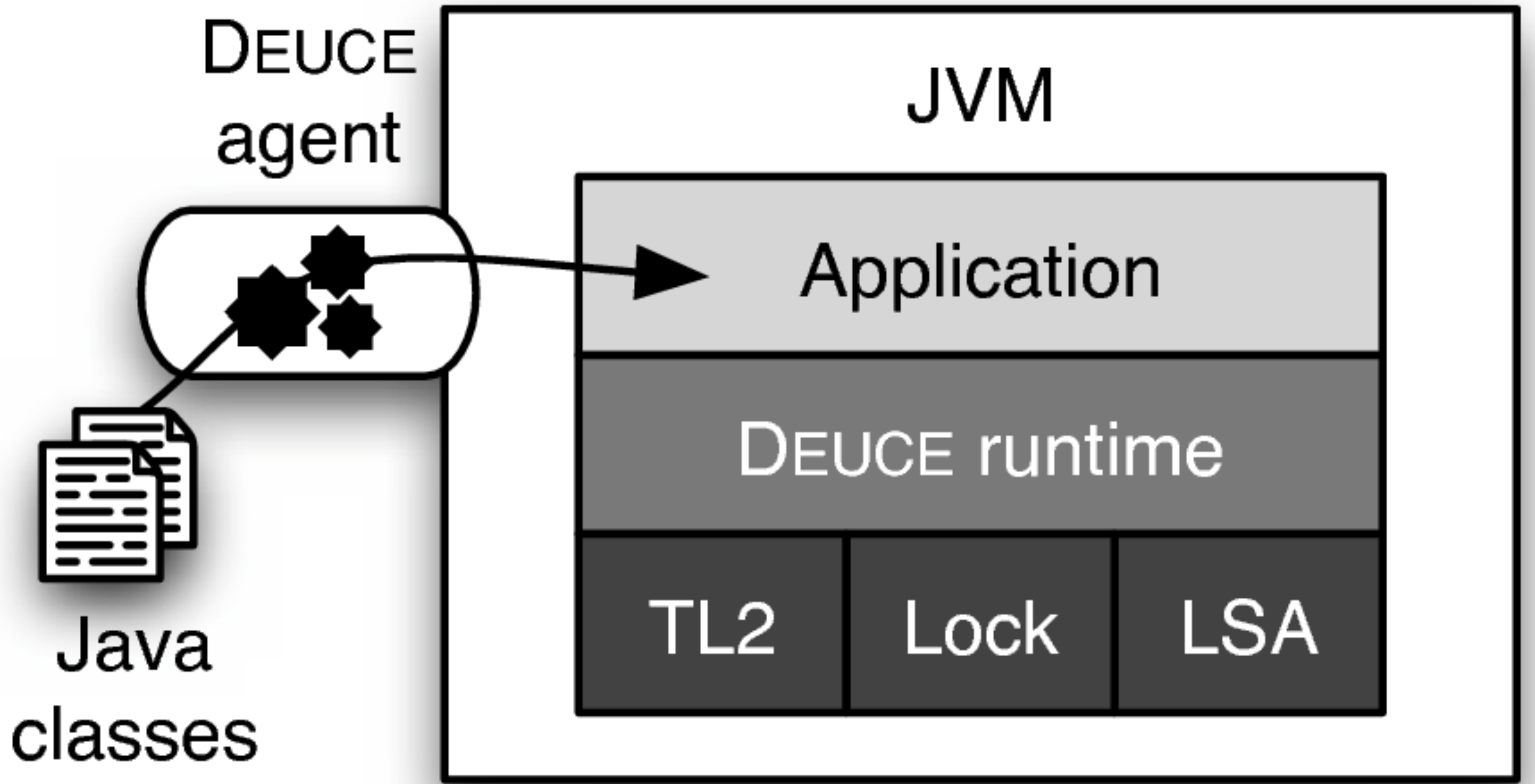
Deuce

- Java STM framework
  - @Atomic methods
  - Field based access
    - More scalable than Object bases.
    - More efficient than word based.
  - Support external libraries
    - Can be part of a transaction
  - No reserved words
    - No need for new compilers (Existing IDEs can be used)
- Research tool
  - API for developing and testing new algorithms.

Deuce - API

```
public class Bank{

    final private static double MAXIMUM_TRANSACTION = 1000;
    private double commission = 0;

    @Atomic(retries=64)
    public void transaction( Account ac1, Account ac2, double amount){
      if( amount < 0 || amount > MAXIMUM_TRANSACTION)
        throw new IllegalArgumentException("Illegal transaction");
      ac1.balance -= (amount + commission);
      ac2.balance += amount;
    }

    @Atomic
    public void update( double value){
      commission += value;
    }

}
```

Deuce - Overview



Deuce - Running

- –javaagent:deuceAgent.jar
  - Dynamic bytecode manipulation.
- -Xbootclasspath/p:rt.jar
  - Offline instrumentation to support boot classloader.
- -Dorg.deuce.exclude="org.junit.*,org.eclipse.*"
  - Exclude class from being instrumented.
  - @Exclude
- java –javaagent:deuceAgent.jar –cp "ex.jar;myjar.jar" MyMain

Outline

- Motivation
- Deuce
- **Implementation**
- TL2
- LSA
- Benchmarks
- Summary
- References

Implementation

- Method
  - @Atomic methods.
    - Replace the with a transaction retry loop.
    - Add another instrumented method.
  - Non-Atomic methods
    - Duplicate each with an instrumented version.

Implementation

```
public void update( double value){
  Context context = ContextDelegetor.getContext();
  for( int i = retries ; i > 0 ; --i){
    context.init();
    try{
      update( value, context);
      if( context.commit())
        return;
    }catch ( TransactionException e ){
      context.rollback();
      continue;
    }catch ( Throwable t ){
      if( context.commit())
        throw t;
    }
  }
  throw new TransactionException();
}
```

Implementation

```
public interface Context{

   void init (int atomicBlockId)
   boolean commit();
   void rollback ();

   void beforeReadAccess (Object obj , long field);
   Object onReadAccess (Object obj, Object value , long field);
   int onReadAccess (Object obj, int value , long field);
   long onReadAccess (Object obj, long value , long field);
   …
   void onWriteAccess (Object obj , Object value , long field);
   void onWriteAccess (Object obj , int value , long field);
   void onWriteAccess (Object obj , long value , long field);
   …
}
```

Outline

- Motivation
- Deuce
- Implementation
- **TL2**
- LSA
- Benchmarks
- Summary
- References

TL2 (Transaction Locking II)

Dave Dice, Ori Shalev and Nir Shavit [DISC 2006]

CTL - Commit-time locking

- Start
  - Sample global version-clock
- Run through a speculative execution
  - Collect write-set & read-set
- End
  - Lock the write-set
  - Increment global version-clock
  - Validate the read-set
  - Commit and release the locks

Outline

- Motivation
- Deuce
- Implementation
- TL2
- **LSA**
- Benchmarks
- Summary
- References

departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

LSA (Lazy Snapshot Algorithm)

Torvald Riegel, Pascal Felber and Christof Fetzer [DISC 2006]

ETL - Encounter-time locking

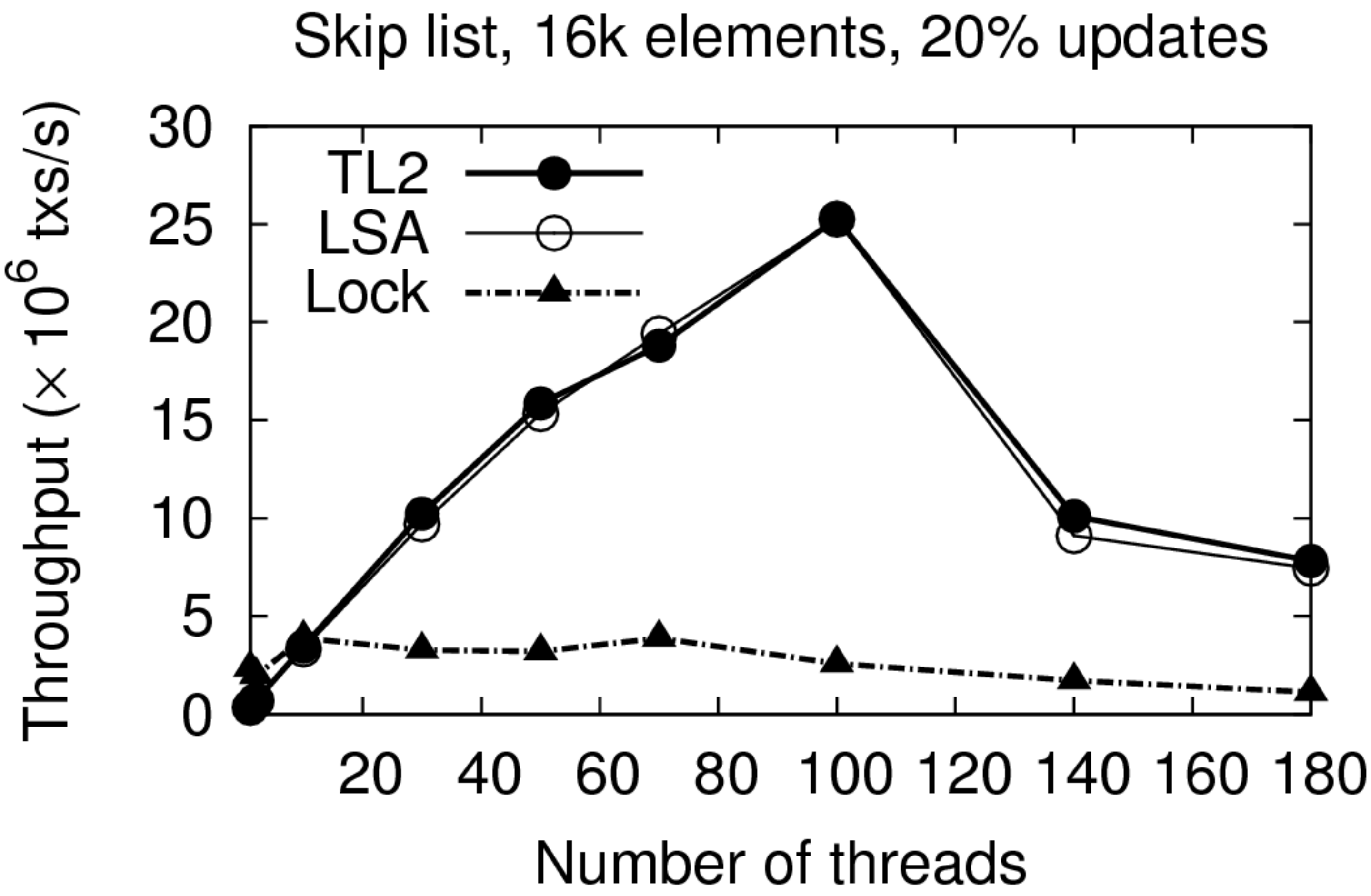
- Start
  - Sample global version-clock
- Run through a speculative execution
  - Lock on write access
  - Collect read-set & write-set
- End
  - Increment global version-clock
  - Validate the read-set
  - Commit and release the locks

Outline

- Motivation
- Deuce
- Implementation
- TL2
- LSA
- **Benchmarks**
- Summary
- References

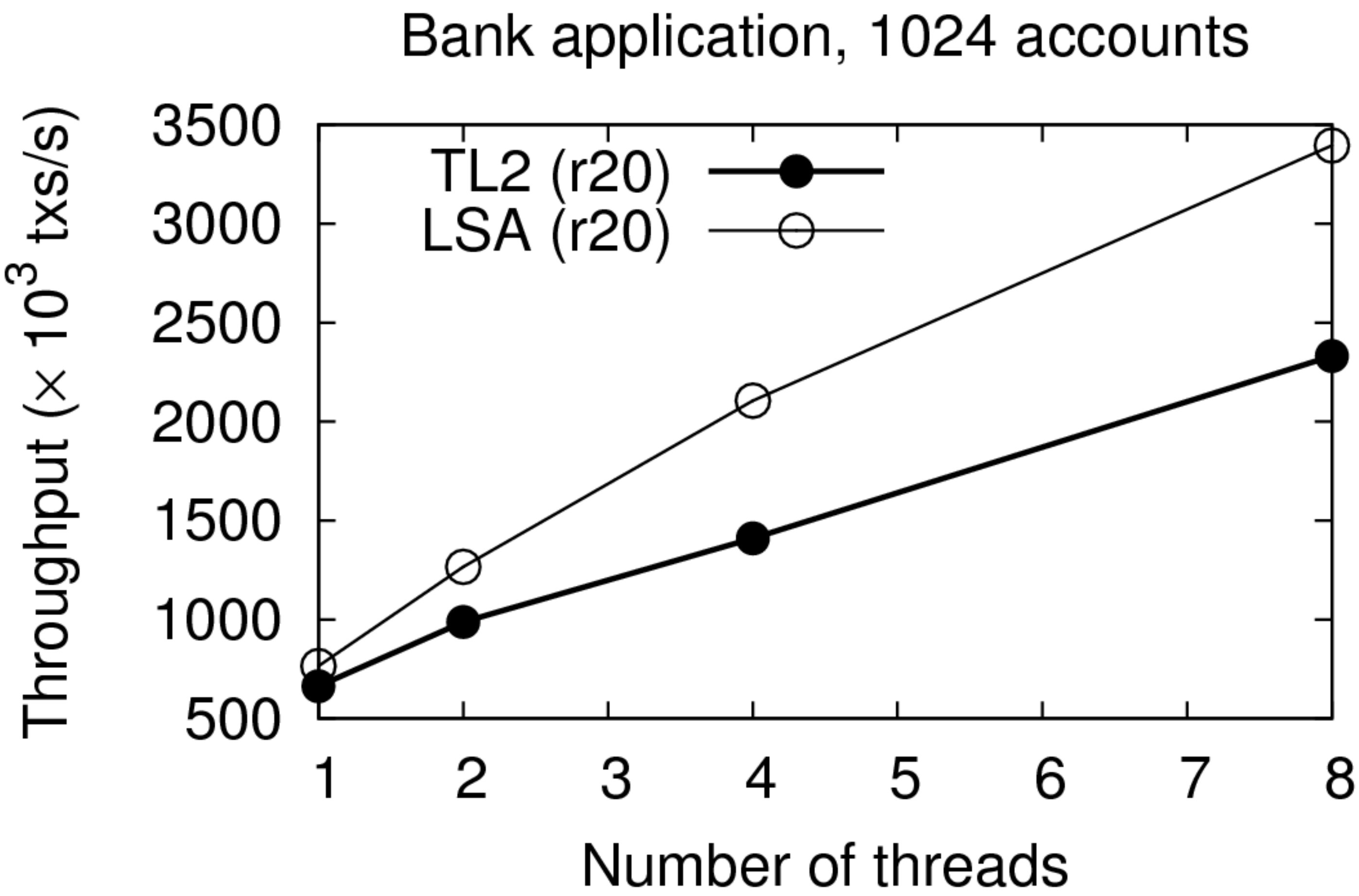
departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Benchmarks (Azul – Vega2 – 2 x 46)



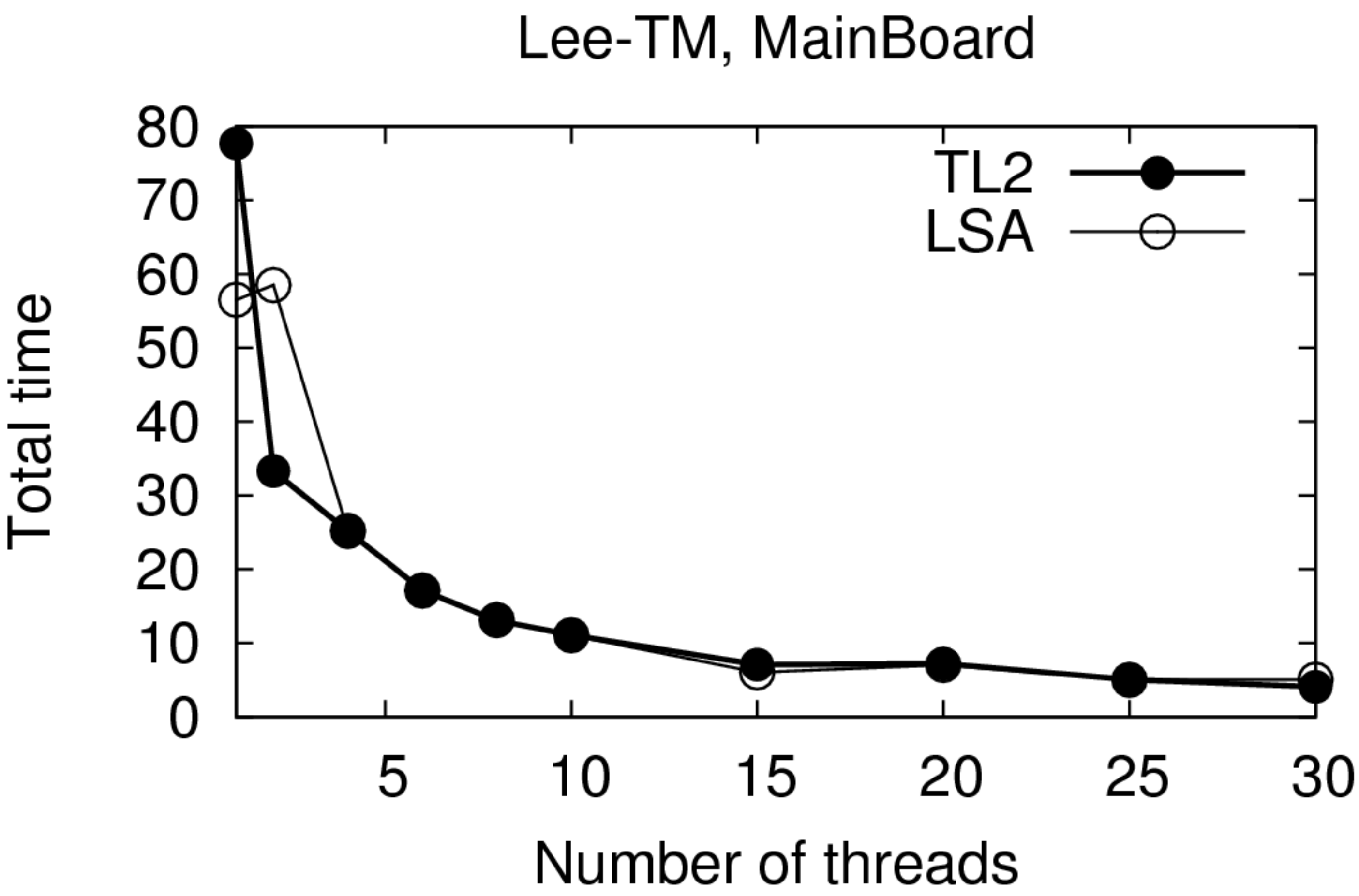
departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Benchmarks (SuperMicro – 2 x Quad Intel)



departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Benchmarks
(Sun UltraSPARC T2 Plus – 2 x Quad x 8HT)



Outline

- Motivation
- Deuce
- Implementation
- TL2
- LSA
- Benchmarks
- **Summary**
- References

Summary

- Simple API
  - @Atomic
- No change to Java
  - No reserved word
- OpenSource
  - On Google code
- Shows nice scalabilty
  - Field based

Outline

- Motivation
- Deuce
- Implementation
- TL2
- LSA
- Benchmarks
- Summary
- **References**

References

- Homepage - http://sites.google.com/site/deucestm/
- Project - http://code.google.com/p/deuce/
- Wikipedia - http://en.wikipedia.org/wiki/Software_transactional_memory
- TL2 – http://research.sun.com/scalable/
- LSA-STM - http://tmware.org/lsastm

Appendix

- Can you just replace synchronized blocks with @Atomic blocks?
  - Opacity
  - Privatization
  - I/O and other non-reversible operations

The End