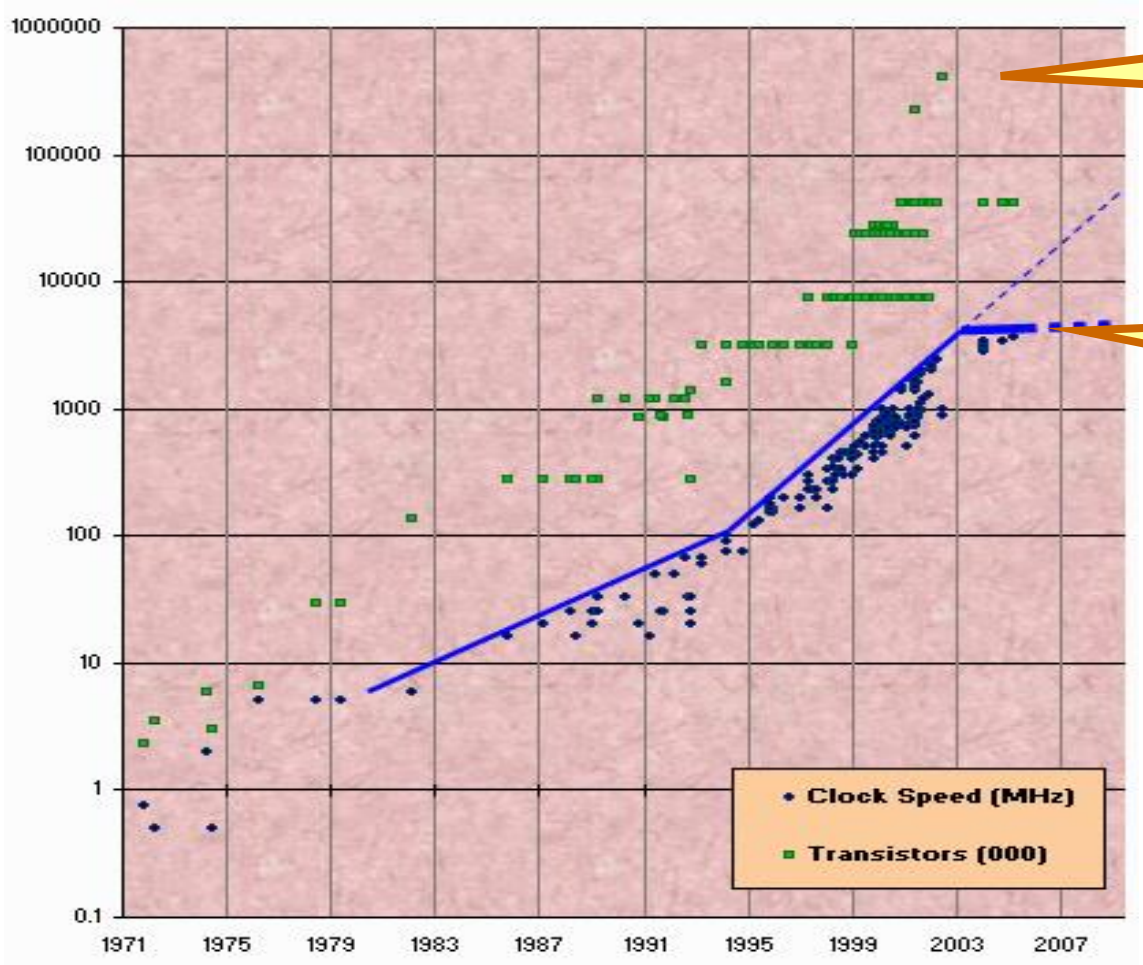
Transactional Memory

Companion slides for
The Art of Multiprocessor Programming
by Maurice Herlihy & Nir Shavit

From the New York Times …

SAN FRANCISCO, May 7. 2004 - Intel said on Friday that it was scrapping its development of two microprocessors, a move that is a shift in the company's business strategy….

Moore's Law



Transistor count still rising

Clock speed flattening sharply

(hat tip: Simon Peyton-Jones)

Multicore Architetures

- "Learn how the multi-core processor architecture plays a central role in Intel's platform approach. …."
- "AMD is leading the industry to multi-core technology for the x86 based computing market …"
- "**Sun's** multicore strategy centers around multi-threaded software. … "

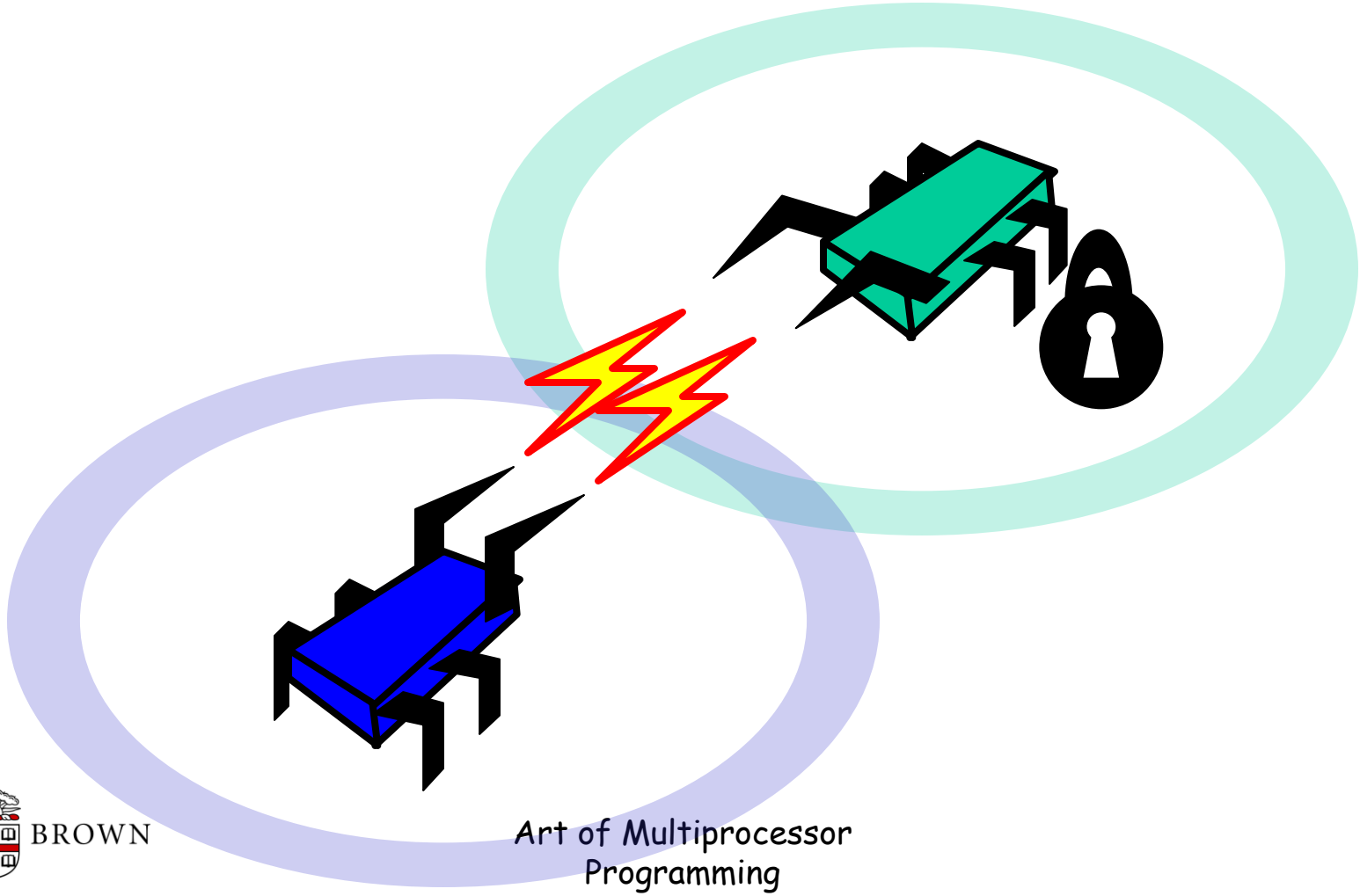
Why do we care?

- Time no longer cures software bloat
- When you double your path length
  - You can't just wait 6 months
  - Your software must somehow exploit twice as much concurrency

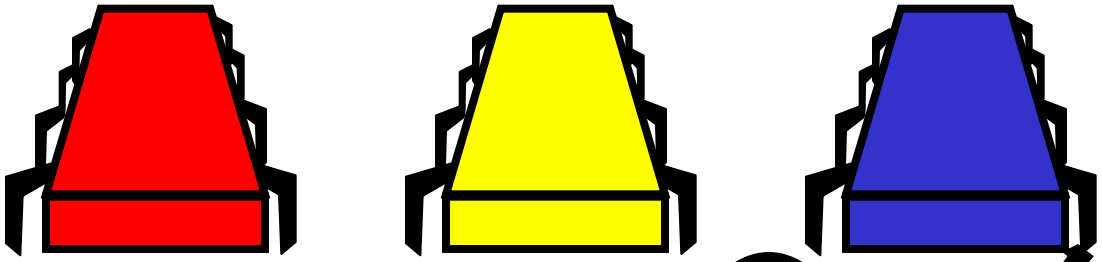
The Problem

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

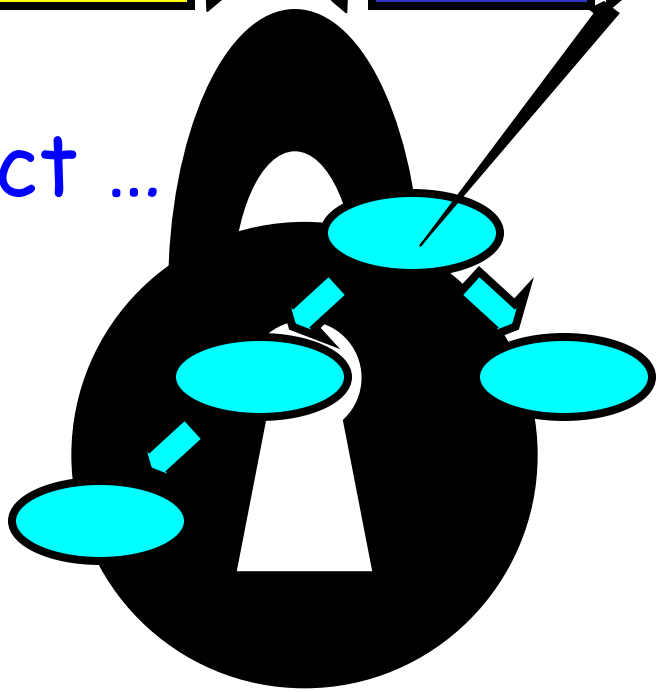
Locking



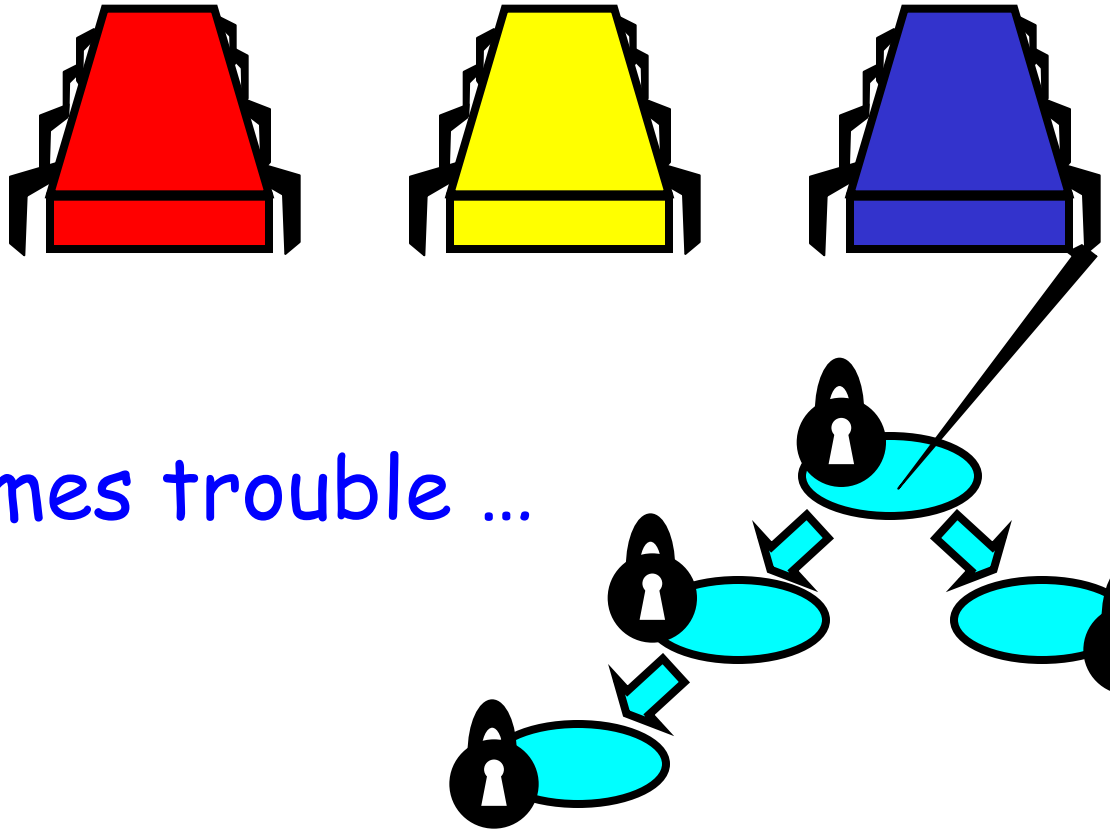
Coarse-Grained Locking



Easily made correct …
But not scalable.



Fine-Grained Locking



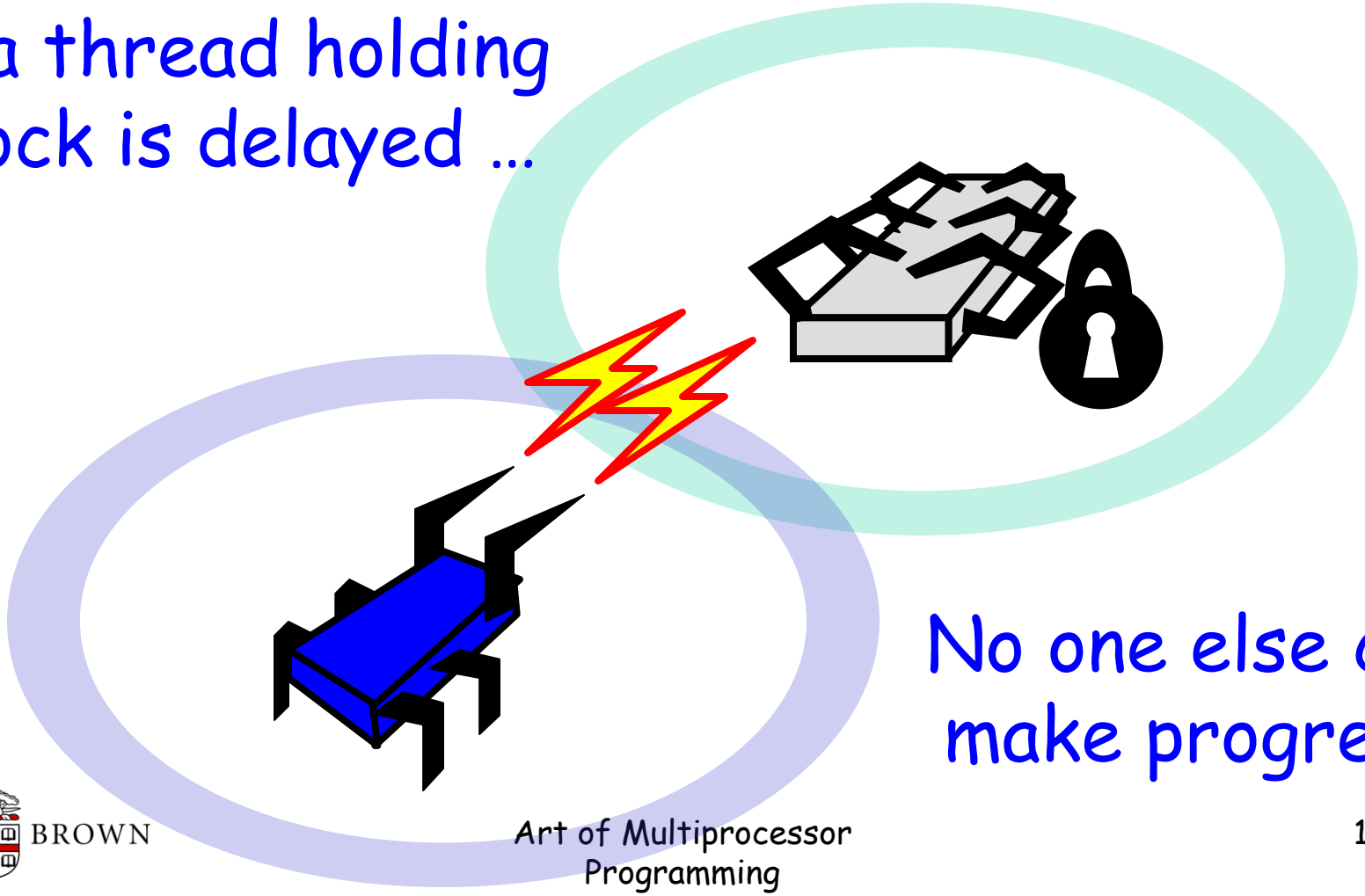
Here comes trouble …

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

Why Locking Doesn't Scale

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

Locking Relies on Conventions

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

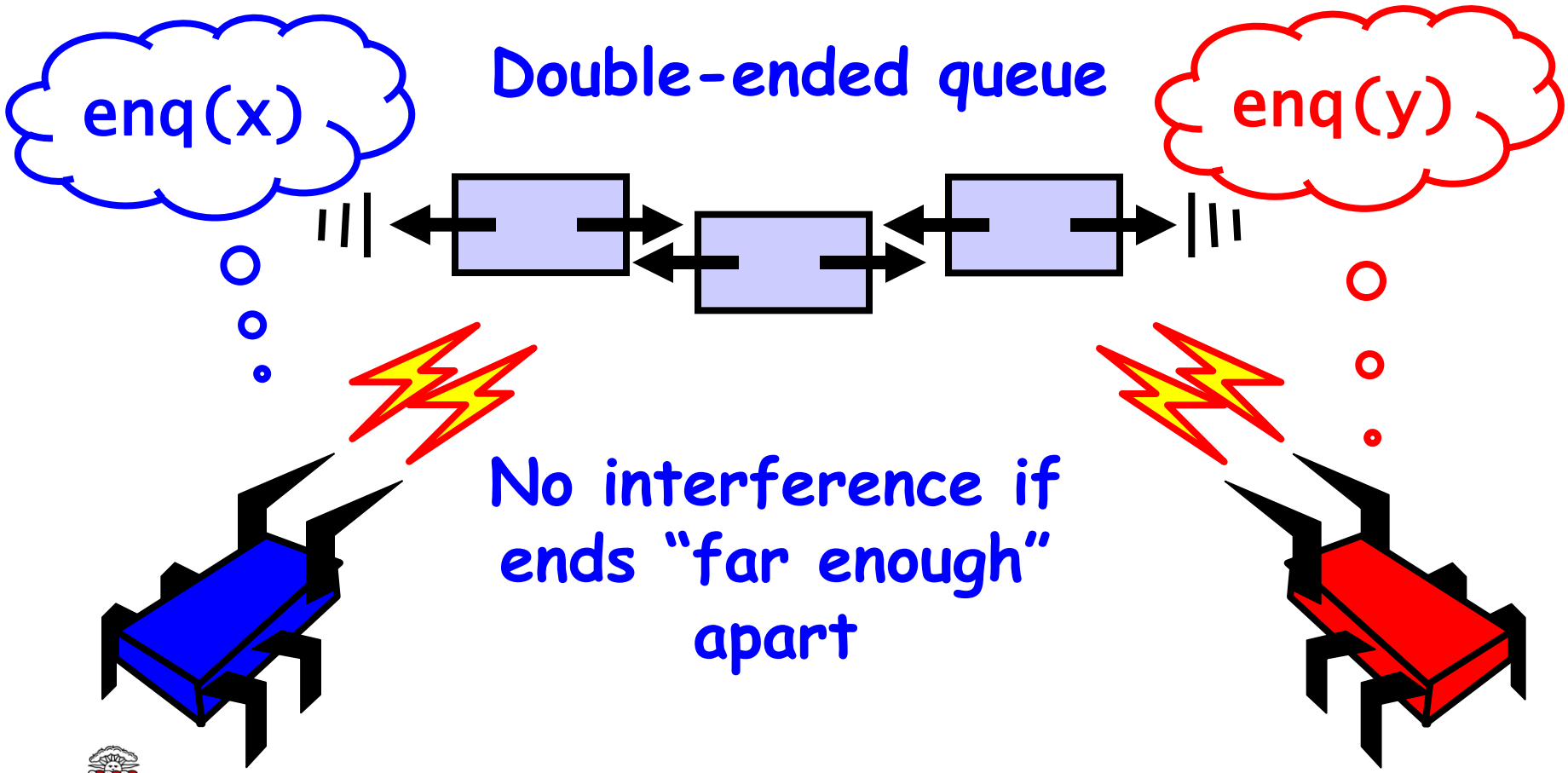
Actual comment from Linux Kernel
(hat tip: Bradley Kuszmaul)

```
/*
 * 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.
 */
```

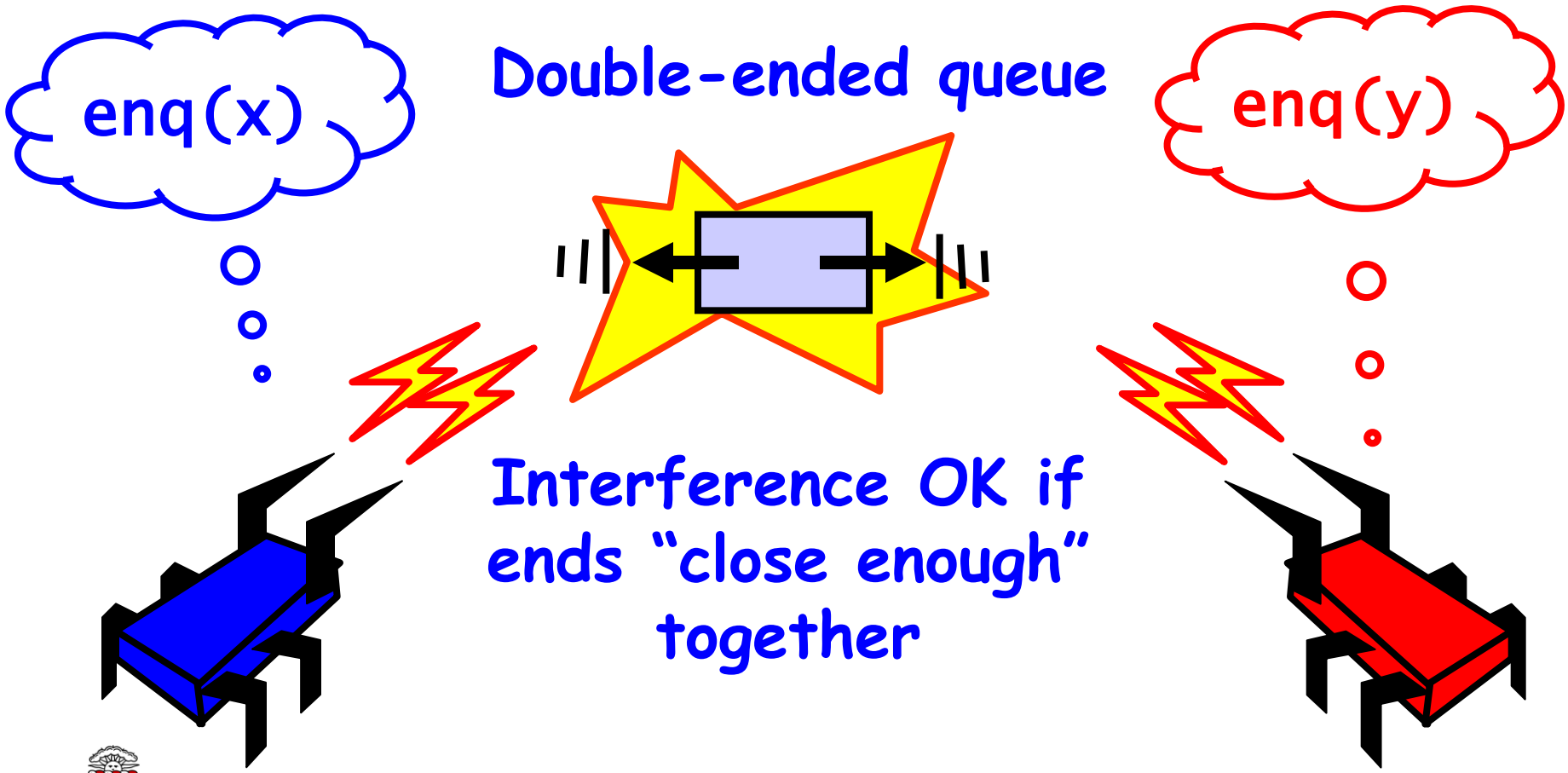
Why Locking Doesn't Scale

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

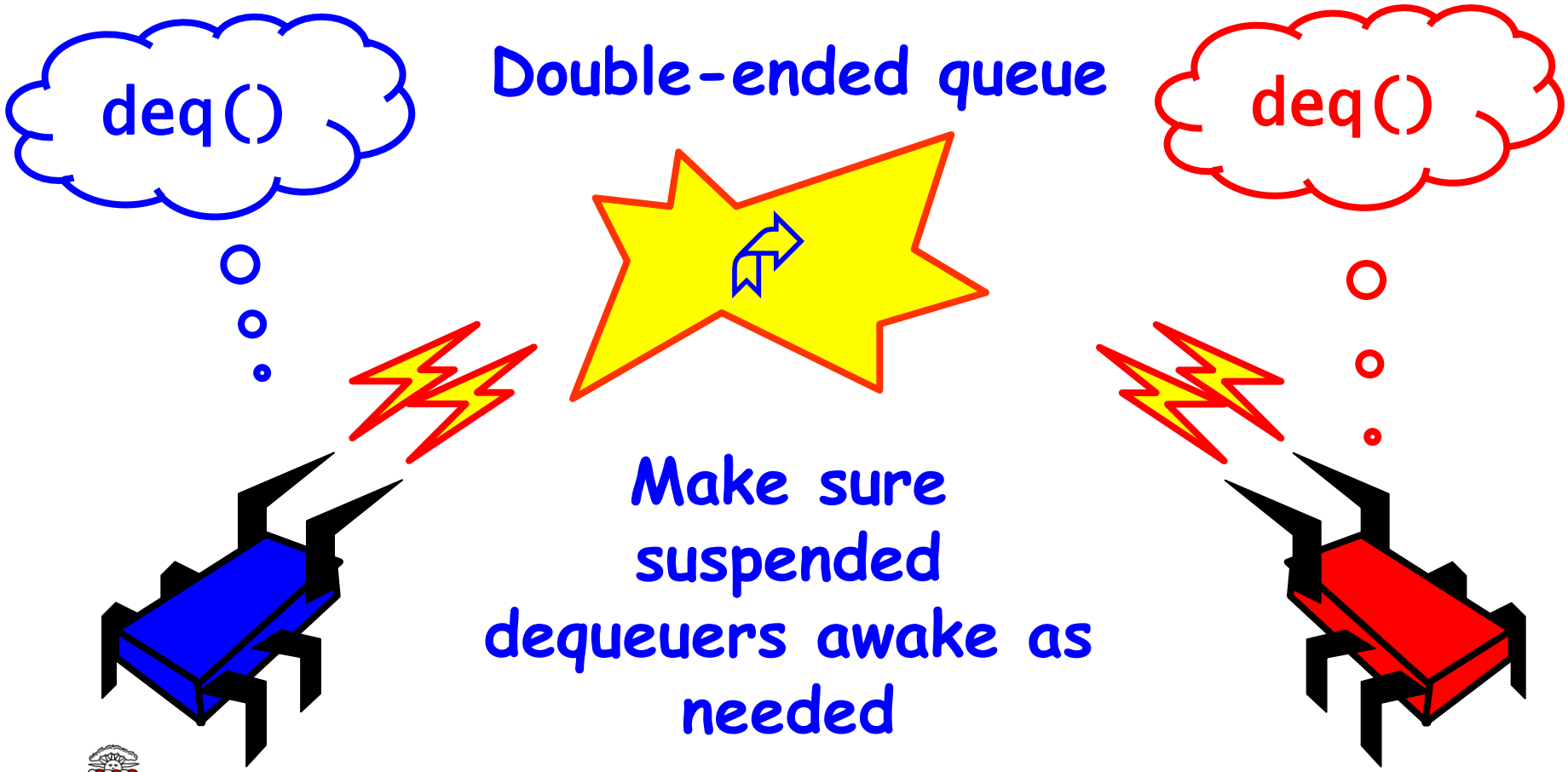
Sadistic Homework



Sadistic Homework



Sadistic Homework



You Try It …

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

In Search of the Lost Wake-Up

- Waiting thread doesn't realize when to wake up
- It's a real problem in big systems
  - "Calling pthread_cond_signal() or pthread_cond_broadcast() when the thread does not hold the mutex lock associated with the condition can lead to *lost wake-up* bugs."

from Google™ search for "lost wake-up"

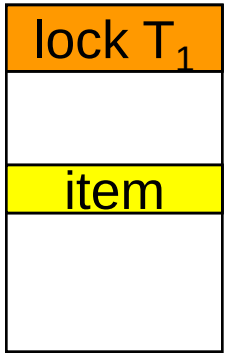
Why Locking Doesn't Scale

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

Locks do not compose

Hash Table

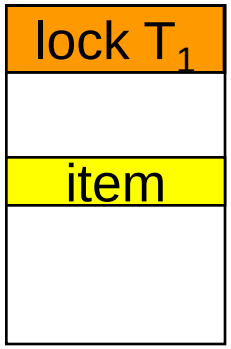
add(T_1, item)

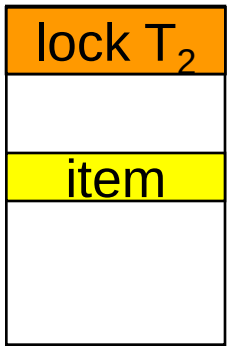


Must lock T_1 before adding item

Move from T_1 to T_2

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

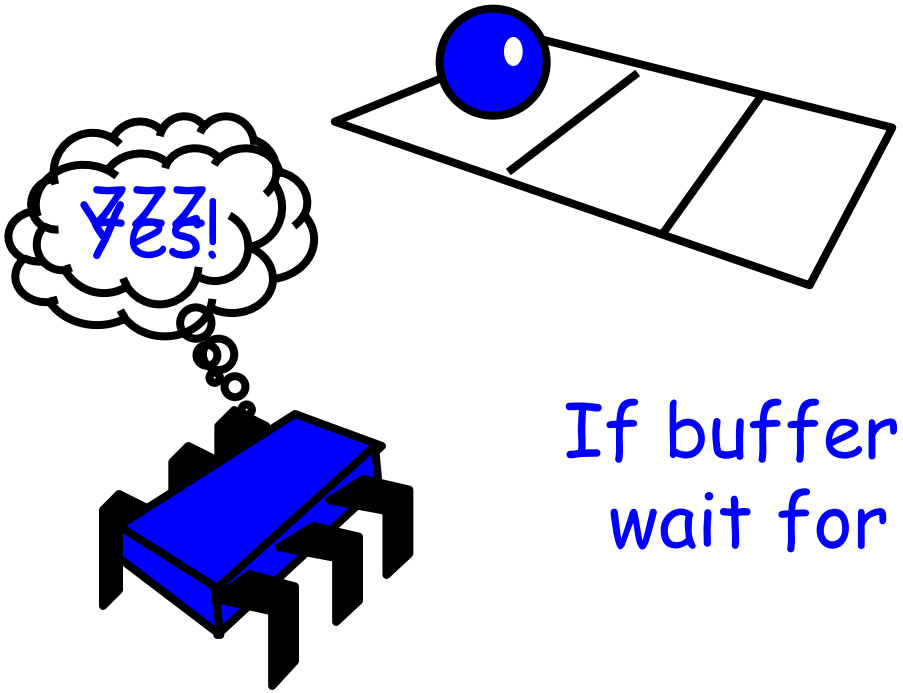




Must lock T_2 before deleting from T_1

Exposing lock internals breaks abstraction

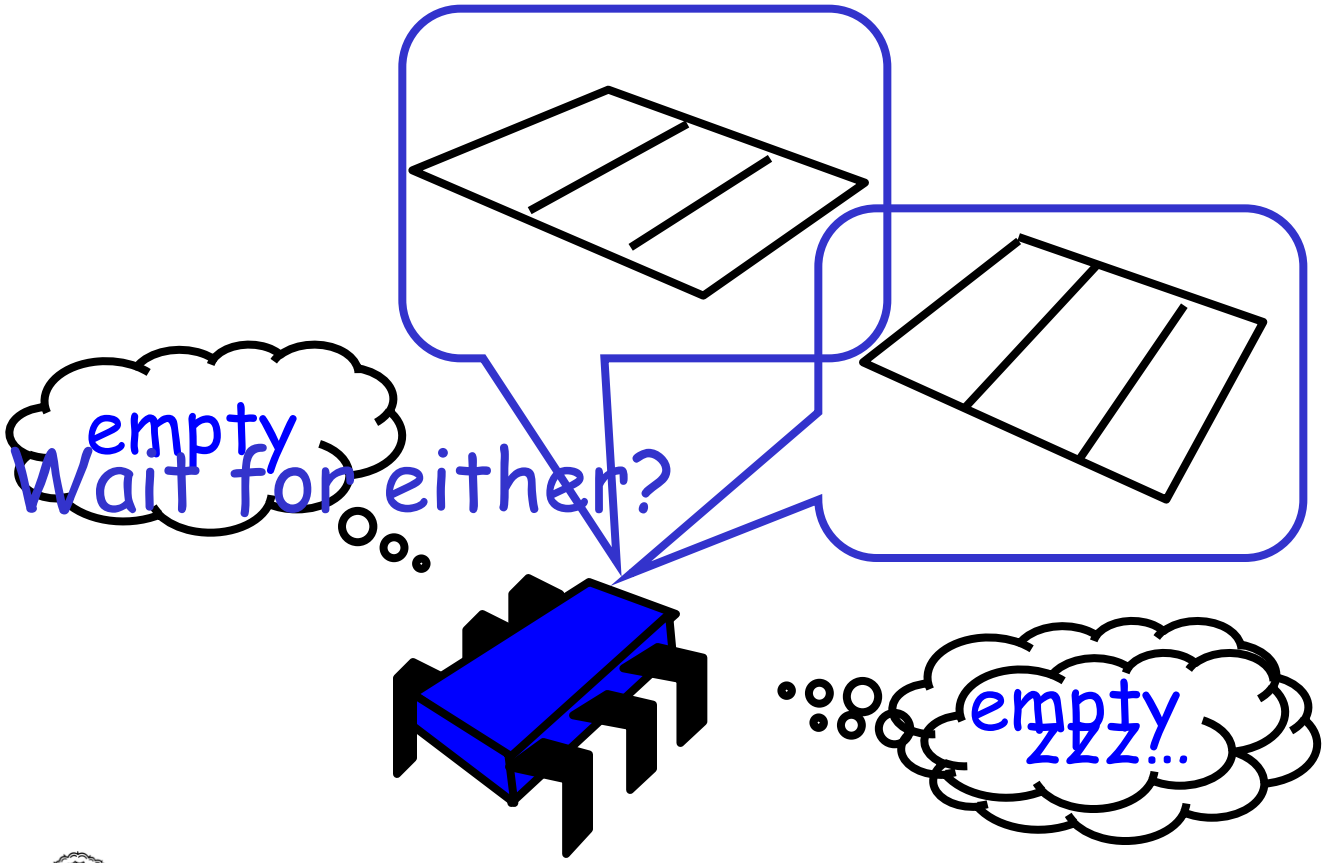
Monitor Wait and Signal



Empty buffer

If buffer is empty, wait for item to show up

Wait and Signal do not Compose



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

Sadistic Homework 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

Sadistic Homework Revisited

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

Sadistic Homework 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

Composition

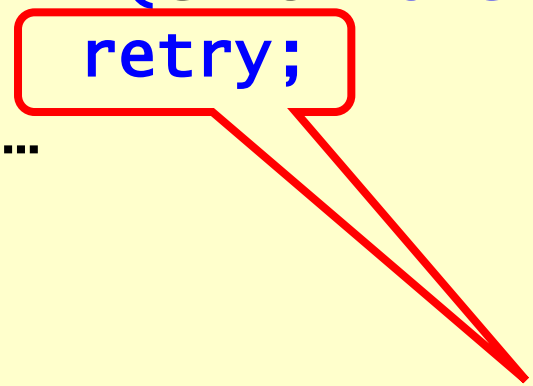
```
Public void Transfer(Queue q1, q2)
{
 atomic {
  Item x = q1.deq();
  q2.enq(x);
 }
}
```

Trivial or what?

Wake-ups: lost and found

```
Public item LeftDeq() {
 atomic {
  if (this.left == null)
    retry;
  …

 }
}
```



Roll back transaction and restart when something changes

OrElse Composition

```
atomic {
  x = q1.deq();
} orElse {
  x = q2.deq();
}
```

Run 1st method. If it retries …
Run 2nd method. If it retries …
Entire statement retries

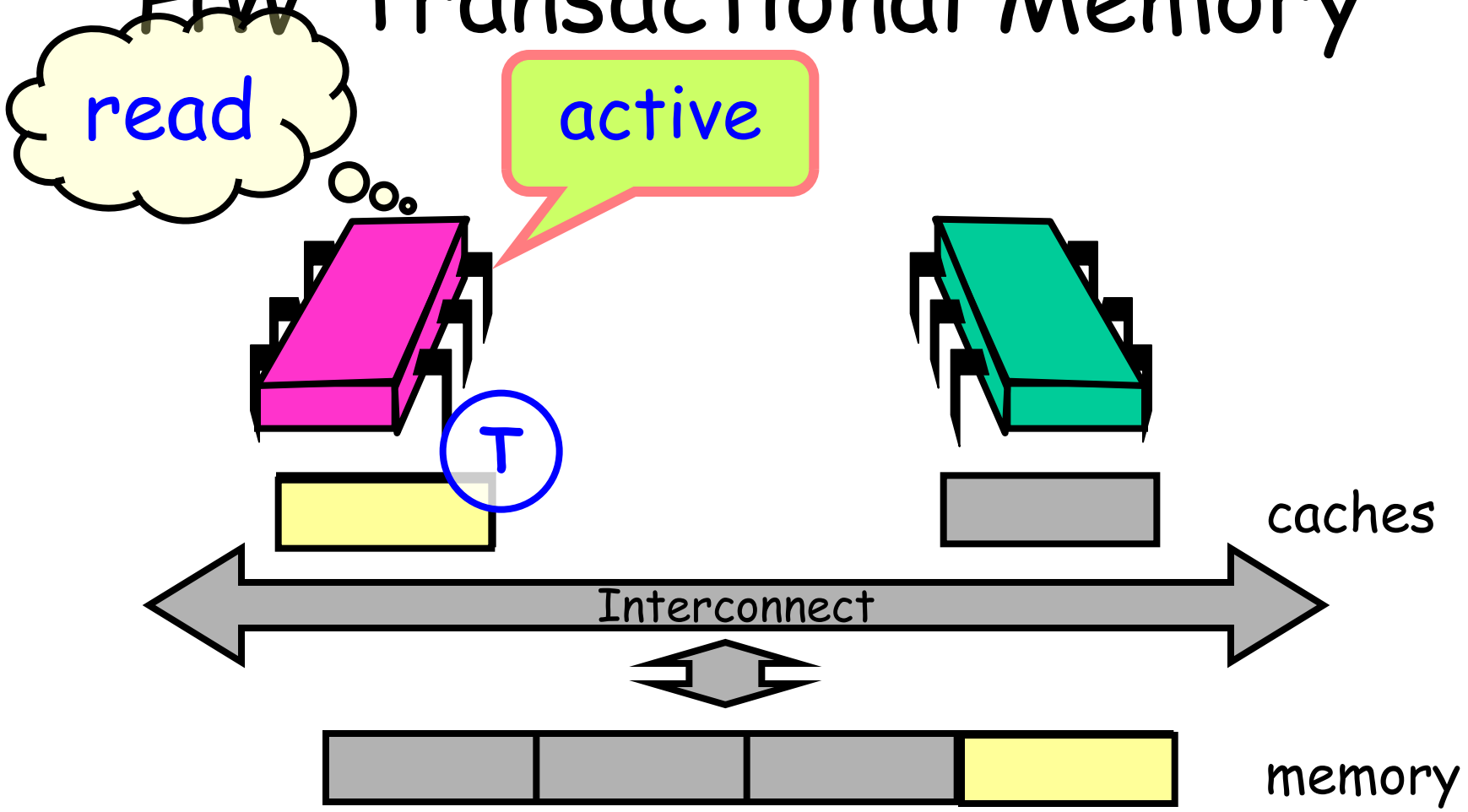
Related Work: Hardware

- First wave
  - Herlihy&Moss 93, Stone et al. 93
- Second wave
  - Rajwar&Goodman 02, Martinez&Torellas 02, Oplinger&Lam 02, TCC 04, VTM 05,
- Third wave
  - IBM's BlueGene/Q, Z-series
  - Intel's RTM(restricted transactional memory)

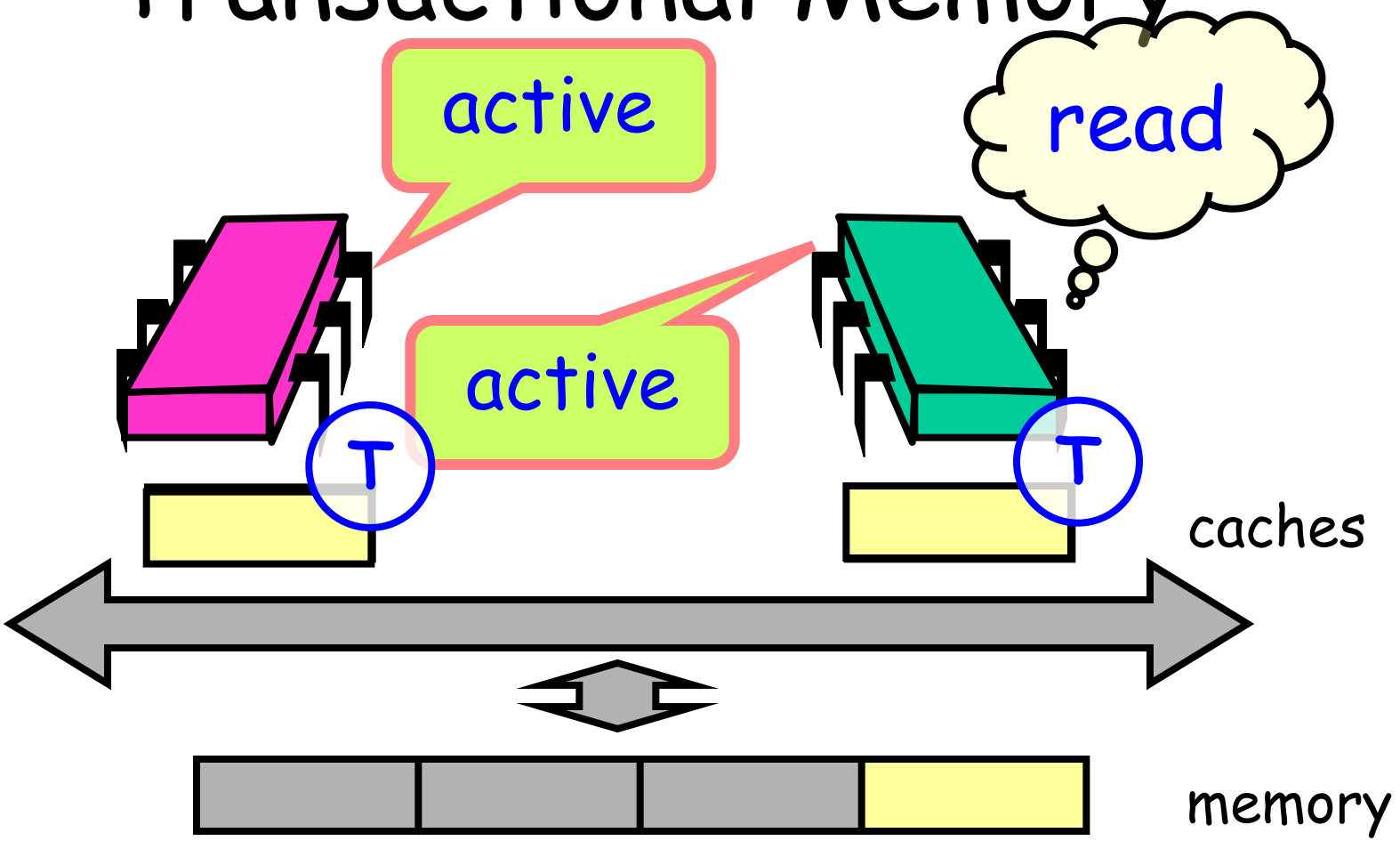
Hardware Overview

- Exploit Cache coherence protocols
- Already do almost what we need
  - Invalidation
  - Consistency checking
- Exploit Speculative execution
  - Branch prediction = optimistic synch!

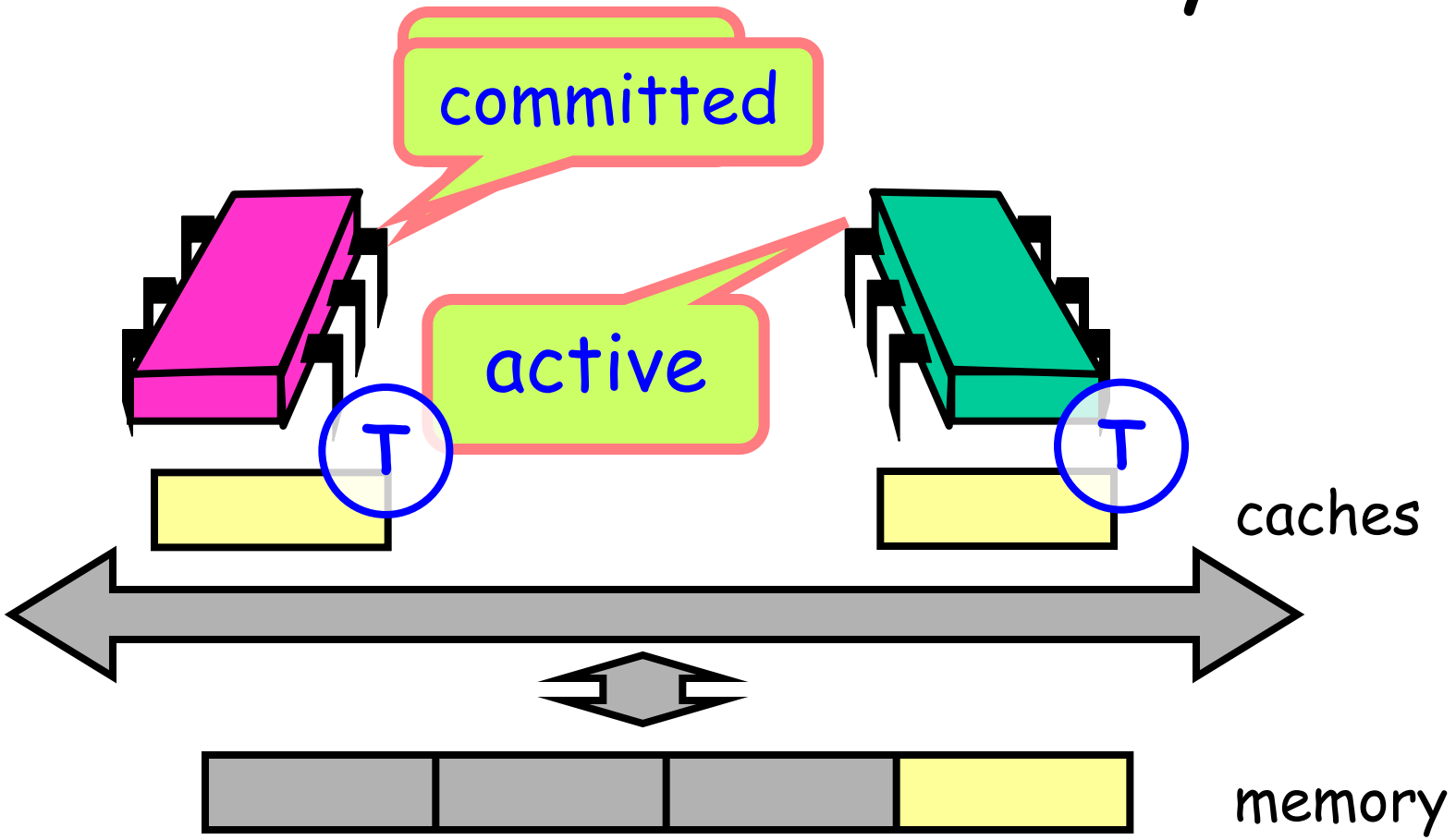
HW Transactional Memory



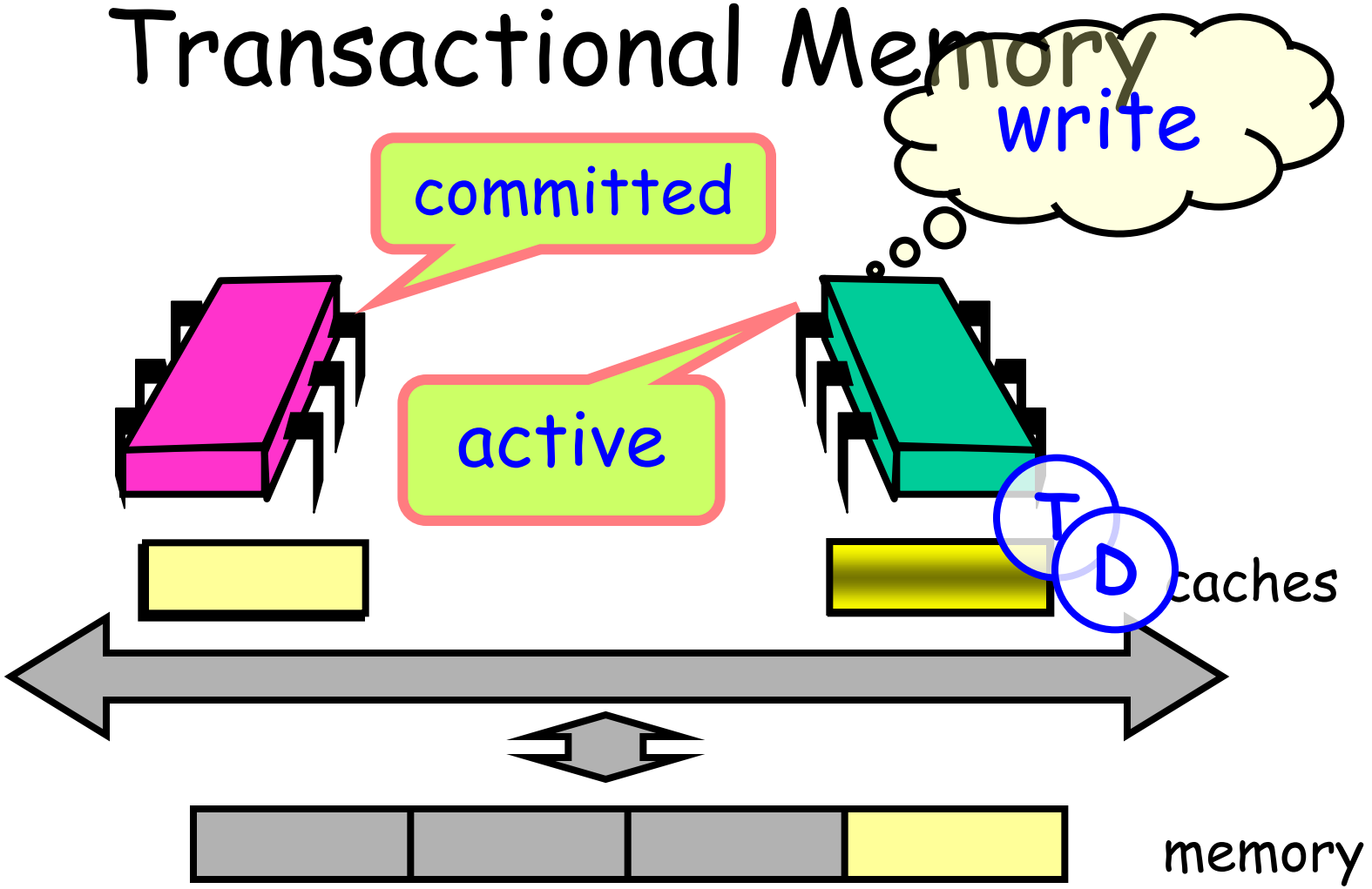
Transactional Memory

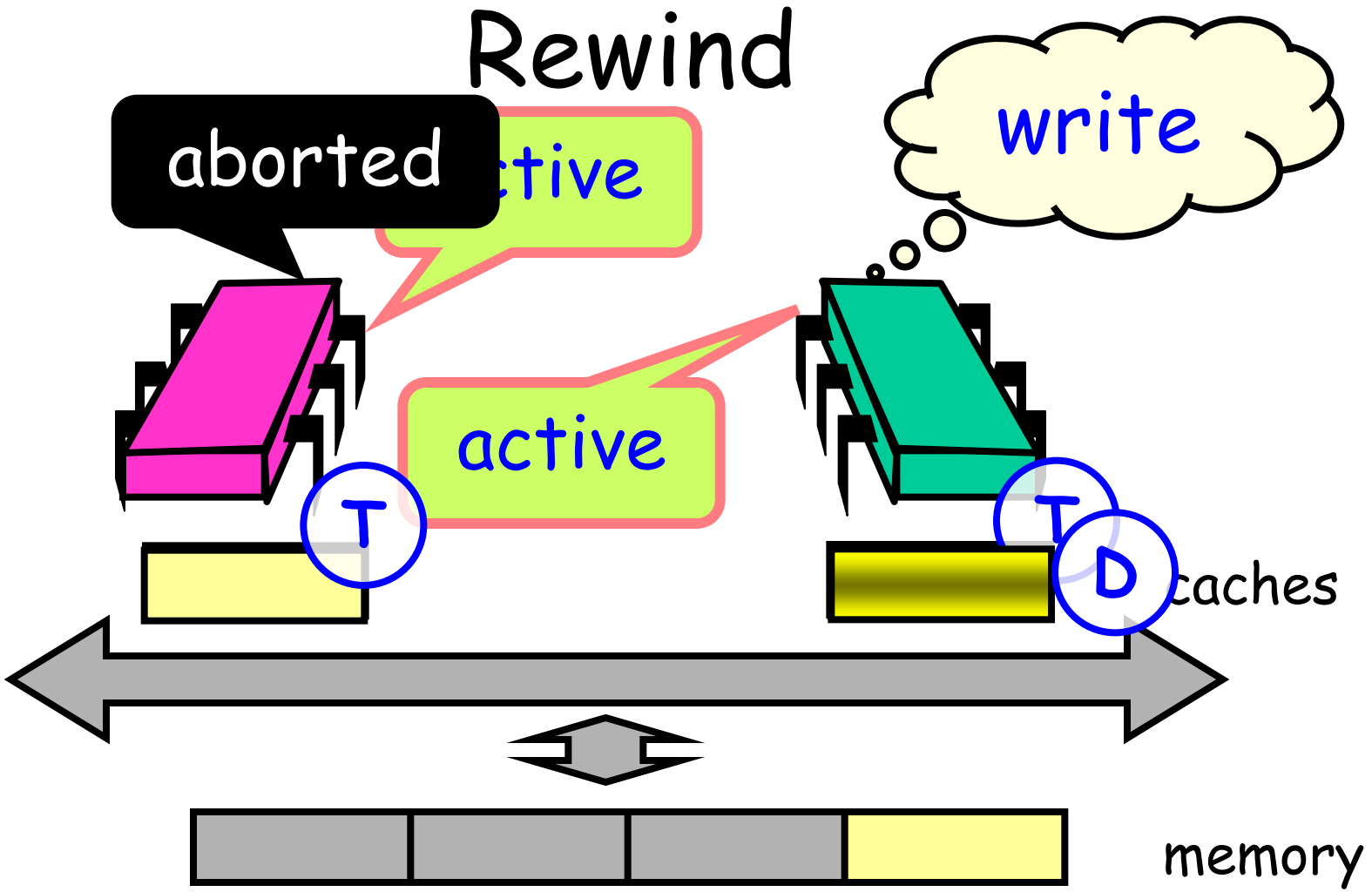


Transactional Memory



Transactional Memory




Rewind
aborted
write
active
T
T
D
caches
memory

Transaction Commit

- At commit point
  - If no cache conflicts, we win.
- Mark transactional entries
  - Read-only: valid
  - Modified: dirty (eventually written back)
- That's all, folks!
  - Except for a few details …

Not all Skittles and Beer

- Limits to
  - Transactional cache size
  - Scheduling quantum
- Transaction cannot commit if it is
  - Too big
  - Too slow
  - Actual limits platform-dependent

Some Approaches

- Trap to software if hardware fails
  - "Hybrid" approach
- Use locks if speculation fails
  - Lock elision
- "Virtualize" transactional memory
  - VTM, UTM, etc…

Related Work: Software

- DSTM [PODC 03]
  - Sun Microsystems, Java library
- FSTM, OSTM [OOPSLA 03]
  - Cambridge University, Java extension
- STM Haskell [PPoPP 05]
  - Microsoft Research
- SXM [TBA]
  - Microsoft Research, C# library

Hardware versus Software

- Do we need hardware at all?
  - Like virtual memory, probably need HW for performance
- Do we need software?
  - Policy issues don't make sense for hardware

We Don't have Language Support (Yet)

- Review a typical STM library

Goals of DSTM2 Project

- World Domination
  - Decent API
  - Encourage experimentation
    - No one agrees on anything yet
  - Capture mind-share
    - For example, ScM projects
    - Released under BSD-style license
    - See Sun download page

Why It's Hard

- Not just a collection of useful objects and methods
- Effect of transactional synchronization is pervasive
  - How classes are defined
  - Control flow: commit & abort
  - Exception handling, etc

Rules

- Provide Java Library
  - Usable by anyone
  - No language/compiler extensions
- Users don't need to master complicated 3rd-party tools
  - Weavers, etc.
  - OK to use such tools internally

This Talk

- How to Implement an STM
- Review: DSTM
- DSTM2 API
- DSTM2 engine room
- Reflections on life …

We Got Issues, Right Here in River City …

- Consistent versus inconsistent views
- Visible versus invisible reads
- Blocking versus non-blocking progress
- Engine-room issues …

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)

Read Synchronization

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

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 Sectarian Differences

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

What is to be done?

- Need an agnostic STM
- Allow users to install (almost) any STM algorithm or policy
- Contenders on common platform
- Low barrier-to-entry for people who want to do STM research

I, for one, Welcome our new Multicore Overlords …

- Multicore architectures force us to rethink how we do synchronization
- Standard locking model won't work
- Transactional model might
  - Software
  - Hardware
- A full-employment act!