IBM & Portland State University

Why The Grass May Not Be Greener On The Other Side: A Comparison of Locking vs. Transactional Memory

Paul E. McKenney, IBM Linux Technology Center
Maged M. Michael, IBM TJ Watson Research
Jonathan Walpole, Portland State University

PLOS 2007 (Revised)

© 2006, 2007 IBM Corporation

Overview, Rationale, and Methodology

- **Inexpensive multi-threaded/multi-core CPUs are here!**
- **Typical practitioner now must handle concurrency**
  - Exceptions include things like SQL
  - In addition, economic considerations may intervene
- **Transactional memory seen as one possible solution**
  - But need to compare fairly to existing mechanism: locking
  - Comparison must cover all relevant attributes
  - But balanced comparisons are difficult in “hot” fields like TM
- **Methodology for balanced comparison:**
  - Maged Michael: strong NBS background, working with STM
  - Paul McKenney: strong locking/RCU background
  - Jon Walpole: versatile, strong conflict-resolution skills
- **Any characterization of locking & TM that both Maged and Paul agree with is necessarily well-balanced**

© 2006, 2007 IBM Corporation

Background

- **How Paul ended up working on this stuff**
- **Context**
- **Background (Paul's view)**
  - Hardware Characteristics
  - Locking
  - Reader-Writer Locking
  - Non-Blocking Synchronization (NBS)
- **Transactional Memory (TM) – consensus view**
- **What Paul *really* thinks**

© 2006, 2007 IBM Corporation

How Paul Ended Up Working on This Stuff

- **Studied transactional memory in early 90s on own time**
  - But Sequent was not in a position to use this
- **Was therefore tapped to help IBM Research in 2002**
  - Collaboration with Josep Torrellas
- **Wrote RCU paper in 2006 on own time**
  - Rejected in late 2006 with particularly bizarre review:
    - "Might be interesting, suggest authors spend a couple of years gaining experience with RCU so that they will have something useful to report"
    - One wonders just how many decades of experience are required...
  - Thus answered a TM query more brusquely than normal
  - Which got me labeled a TM skeptic, and thus selected as an essential member of a within-IBM TM steering committee
  - Given that the work was done, why not publish?
- **So this work is in part the product of two of Paul's failed investments in himself!!!**

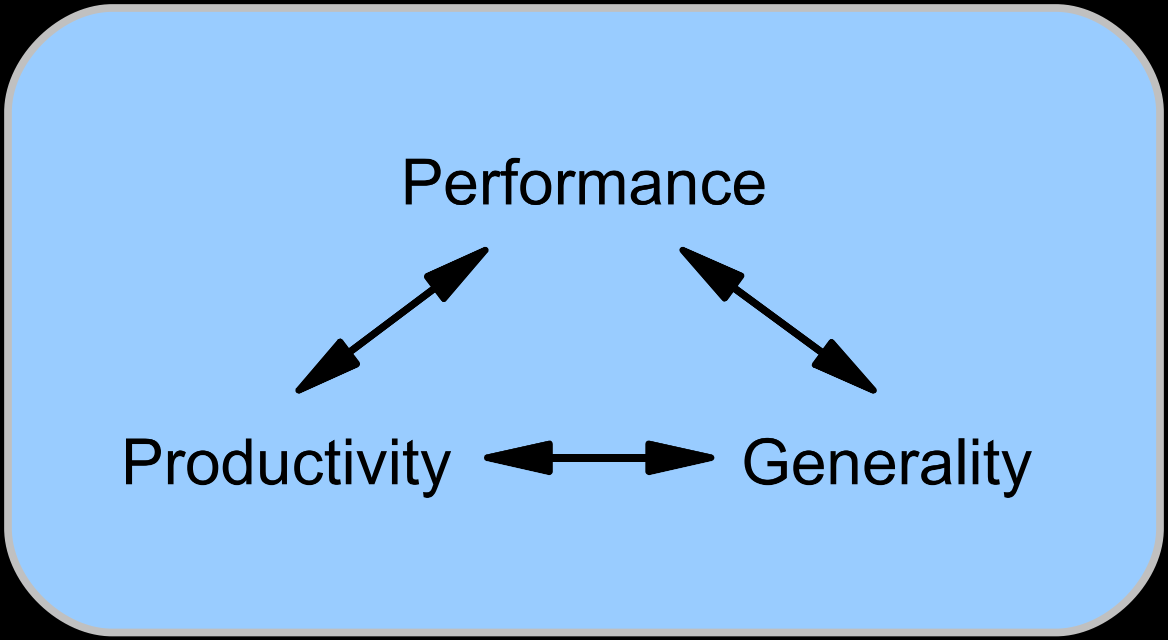
© 2006, 2007 IBM Corporation

Context

Joint work with Manish Gupta, Maged Michael,
Phil Howard, Joshua Triplett, and Jonathan walpole

© 2006, 2007 IBM Corporation

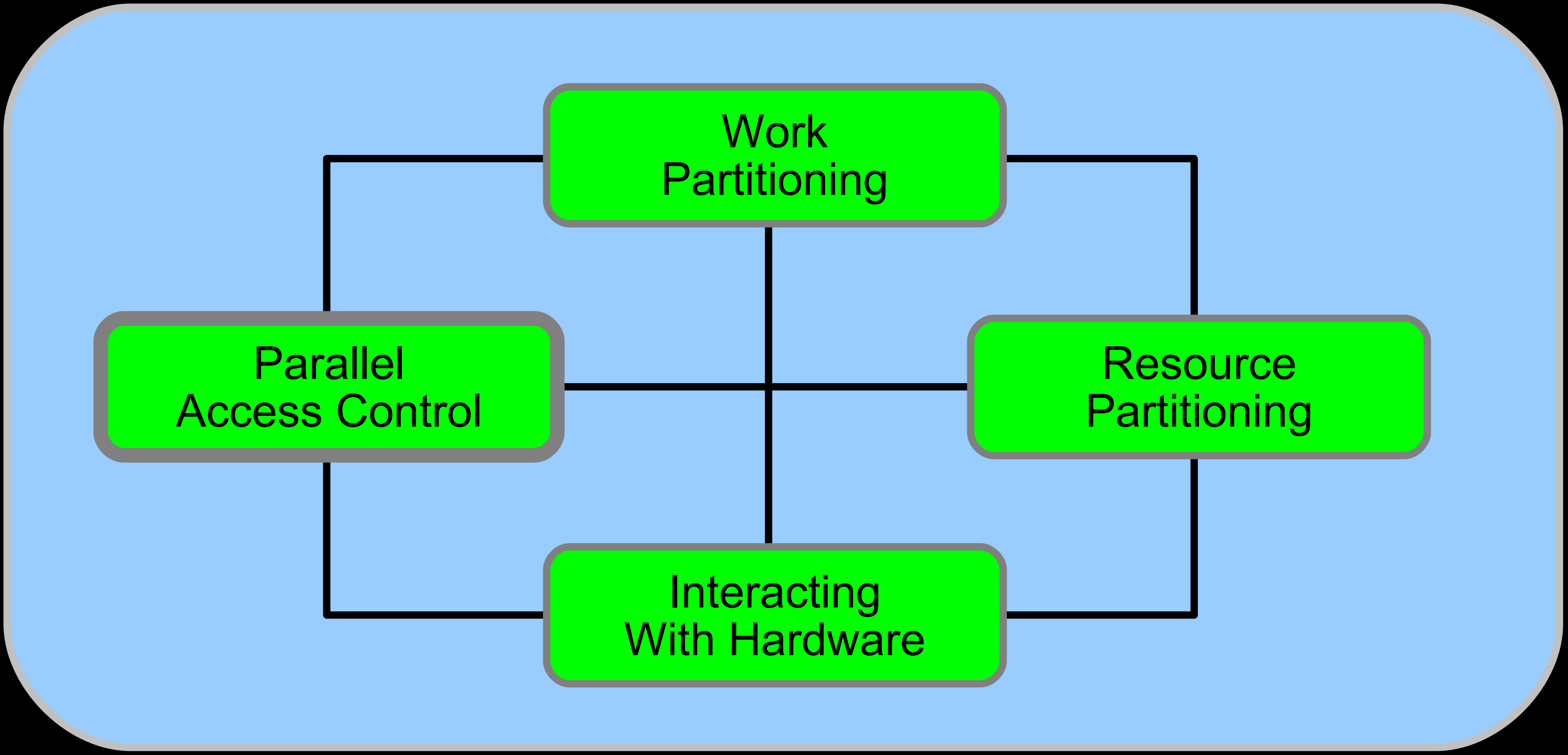
Context: Goals



At best, pick any two!!!

© 2006, 2007 IBM Corporation

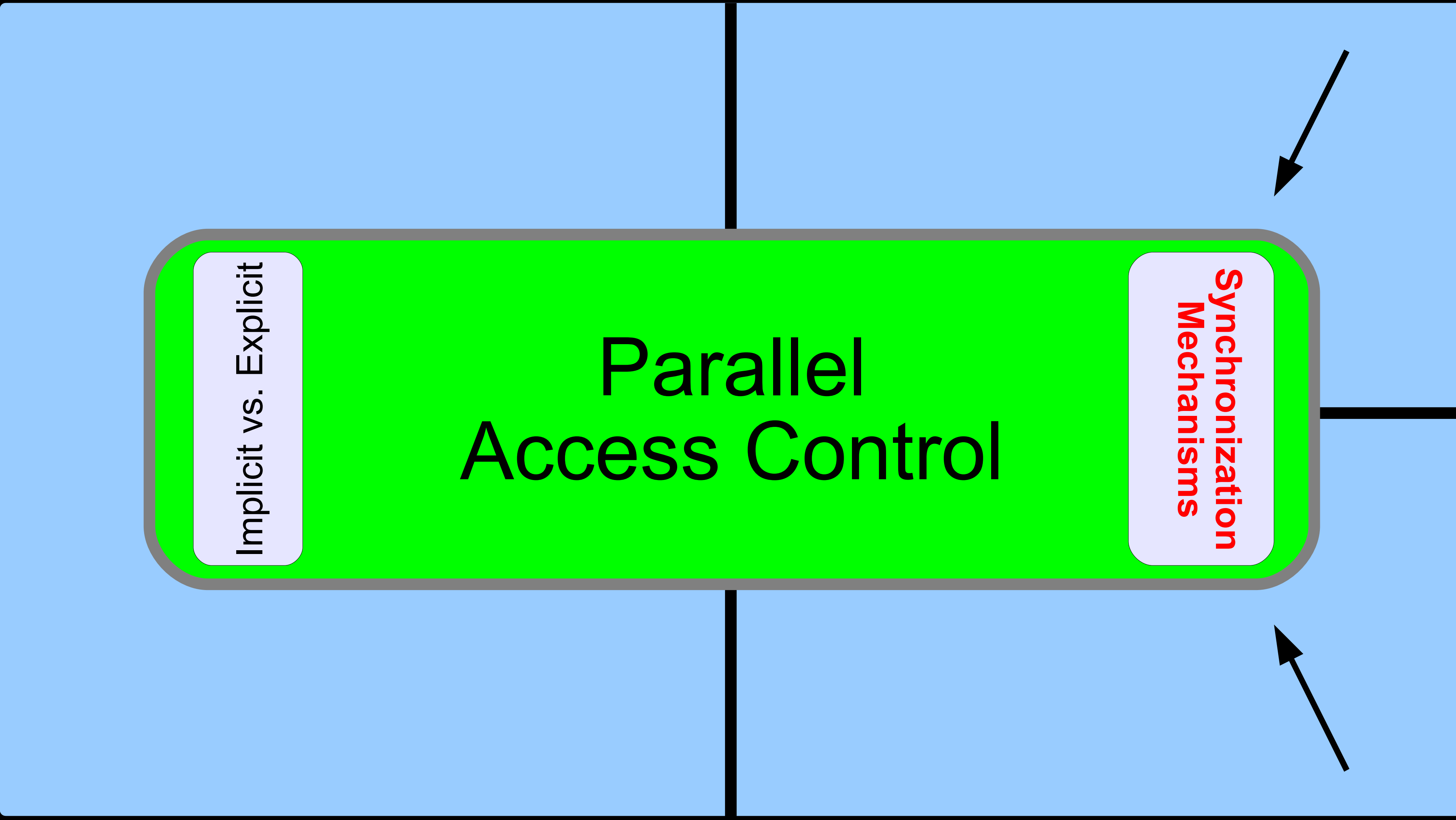
Context: Tasks



Data-parallel approach: first partition resources, then partition work, and only then worry about parallel access control. Lather, rinse, and repeat.

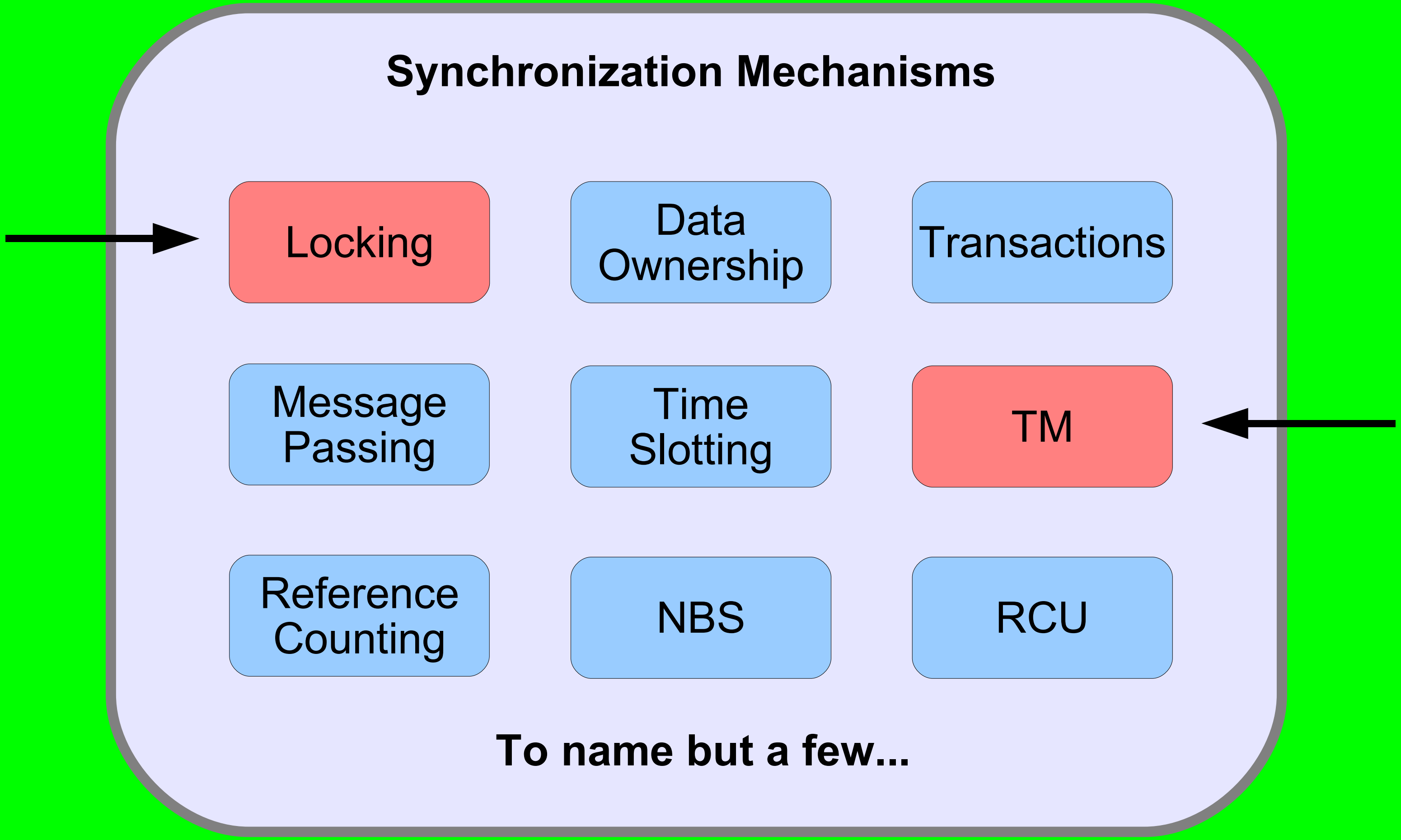
© 2006, 2007 IBM Corporation

Context: Tasks (Close-Up View)



© 2006, 2007 IBM Corporation

Context: Tasks (Even Closer View)



© 2006, 2007 IBM Corporation

Background (Paul's View)

© 2006, 2007 IBM Corporation

Not All Machine Instructions Are Created Equal

4-CPU 1.8GHz AMD Opteron 844 system

RCU

Operation	Cost (ns)	Ratio
Clock period	0.6	
Best-case CAS	37.9	63.2
Best-case lock	65.6	109.3
Single cache miss	139.5	232.5
CAS cache miss	306.0	510.0

Heavily optimized reader-writer lock might get here for readers (but too bad about those poor writers...)

Typical synchronization mechanisms do this a lot

Costs of atomic operations has improved, but how much more can we really get?
Remember: atomic operations normally cannot leverage CPU write buffer...

© 2006, 2007 IBM Corporation

Why Aren't All Instructions Created Equal?

	Store Buffer
a = 1;	a=1
b = 2;	a=1,b=2
c = 3;	a=1,b=2,c=3

	Store Buffer
a = 1;	a=1
b = 2;	a=1,b=2
t = CAS(&c, 0, 1);	a=1,b=2

Wait for cache line containing "c"!!!

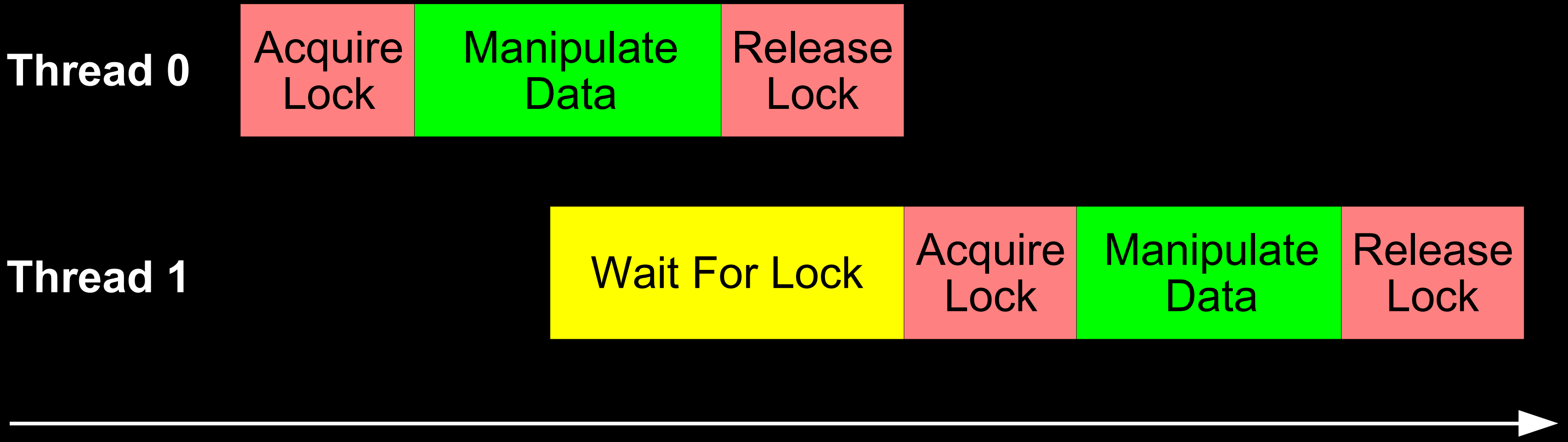
Cannot possibly know "t" till then!!!

There are tricks the HW guys play – otherwise the latencies would be much worse.

© 2006, 2007 IBM Corporation

Locking

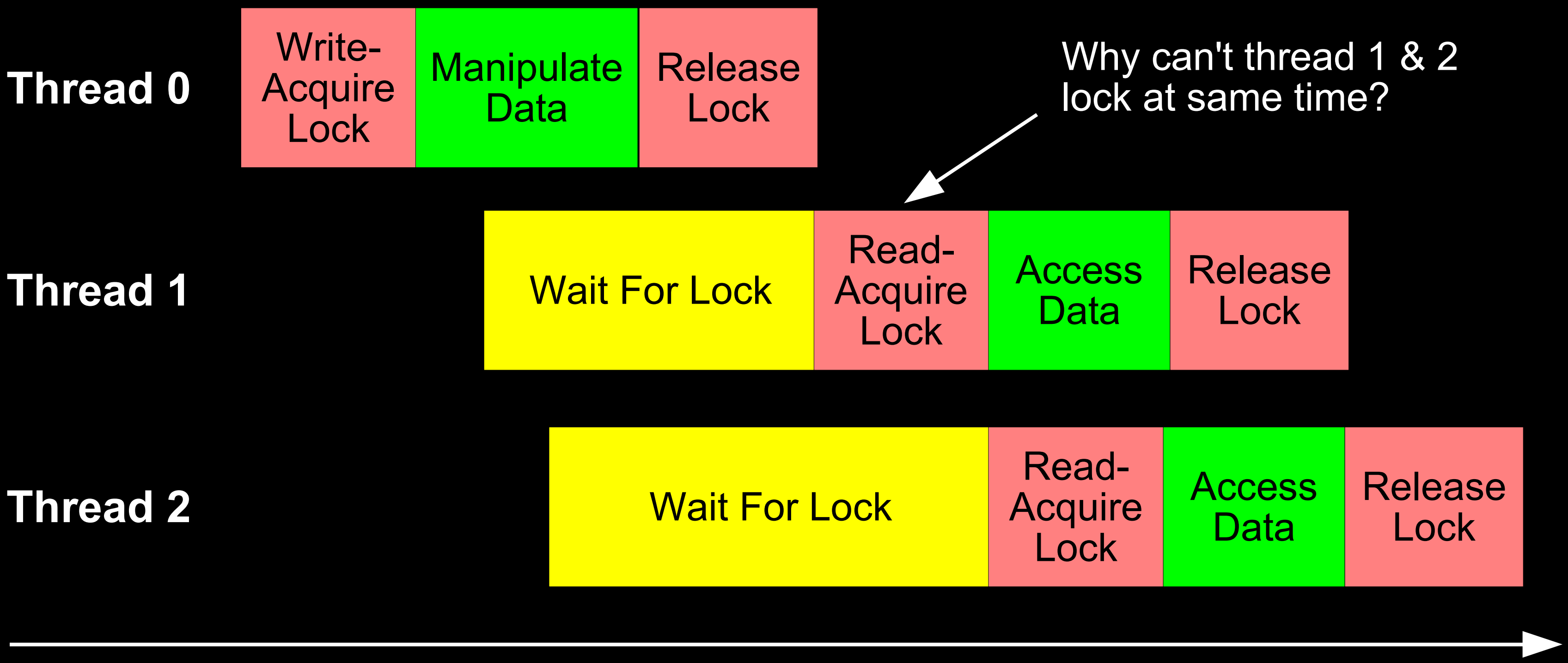
- **"Locks" associated with data**
- **To access a given piece of data, thread must hold the corresponding lock**
  - Despite rumors to the contrary, reasonably easy to use, given global visibility into and control of the code base (more on this later)



© 2006, 2007 IBM Corporation

Reader-Writer Locking

- **“Locks” again associated with data**
  - To read a given piece of data, thread read-holds corresponding lock
  - To modify a given piece of data, thread write-holds corresponding lock



© 2006, 2007 IBM Corporation

Non-Blocking Synchronization (NBS)

- **NBS can be thought of as “optimistic”**
  - Perform setup, then use atomic operations to do combination of verification and (if passes) finalization
    - If verification fails, rollback/retry or hand off, depending on type of NBS
    - **Note heavy use of atomic operations!!!**
  - Verification can be extremely complex
    - Even when assuming mythical sequentially consistent computer systems
  - Impact of contention can be quite severe
- **NBS favored in 1990s research**
  - Some production use: simple NBS and “semi-NBS” (weaker linearization and fault-tolerance properties)
  - Research focus shifting to TM (see next slide)

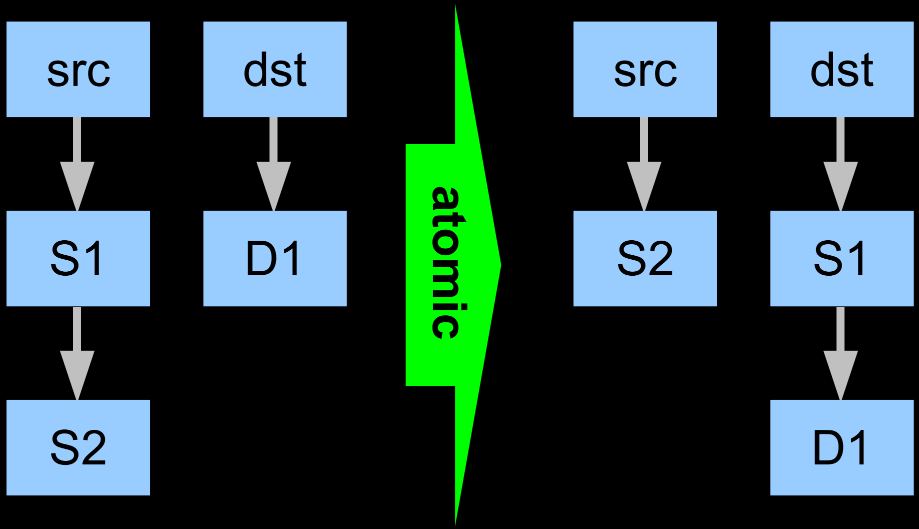
© 2006, 2007 IBM Corporation

Transactional Memory (TM)

- **Currently the focus of intense research effort**
  - So this slide is necessarily out of date
- **Can be constructed to be either optimistic or pessimistic**

```
struct foo *pop_push(struct foo_stack *src, struct foo_stack *dst)
{
        struct foo *q;

        begin_txn;
        q = src;
        src = q->next;
        q->next = dst;
        dst = q;
        end_txn;
}
```

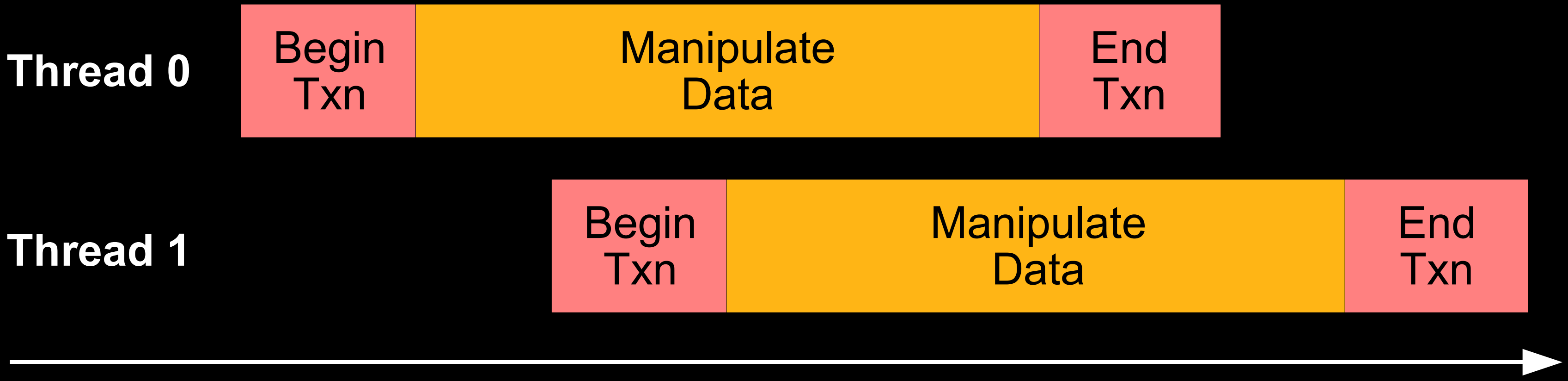


What is not to like?

© 2006, 2007 IBM Corporation

TM Does Not Suspend the Laws of Physics

- **Costs shown below can be moved around depending on TM implementation, but they are inherent (no CPU write buffer!!!)**
  - Beginning, ending, and aborting transactions
  - Adding a new object to a transaction
  - Handling conflicts among transactions
  - Or can accept transaction size limits with hardware implementation
- **Reducing these overheads is a critical research challenge**
- **Ratio of data and control operation overheads challenging for TM**
  - DBMS: data operation usually includes reads/writes to mass storage device
  - TM: data operations almost always includes only reads/writes to *memory...*



© 2006, 2007 IBM Corporation

Some TM Nomenclature

- **TM: Transactional Memory**
- **HTM: Hardware Transactional Memory**
  - Requires additional instructions, thus new hardware
- **STM: Software Transactional Memory**
- **UTM: Unbounded Transactional Memory**
  - Normally a hybrid using HTM for small transactions and STM for large transactions, but there are also hardware-only approaches
- **Log-based TM: create either an undo or redo log**
  - Undo log makes commit processing fast
  - Redo log makes abort processing fast
- **Inevitable transactions: designated transactions that are not permitted to abort**
  - Paul's view: “Locks in transactional clothing”

© 2006, 2007 IBM Corporation

Locking and TM: Comparison and Status

Consensus View

© 2006, 2007 IBM Corporation

Locking and TM: Basics

	Locking	Transactional Memory
Basic Idea	Allow only one thread at a time to access a given set of objects.	Cause a given operation over a set of objects to execute atomically.
Scope	Idempotent and non-idempotent operations.	Idempotent and non-concurrent non-idempotent operations.
		Concurrent non-idempotent operations require hacks.
Composability	Limited by deadlock.	Limited by non-idempotent operations and performance.
Scalability and Performance	Data must be partitionable to avoid lock contention.	Data must be partitionable to avoid conflicts.
	Partitioning typically must be fixed at design time.	Dynamic adjustment of partitioning carried out automatically.
	Contention effects can be focused on acquisition and release, so that critical section runs at full speed.	Contention effects can degrade performance of processing within the transaction.

© 2006, 2007 IBM Corporation

Locking and TM: Practical Applicability

	Locking	Transactional Memory
HW Support	Commodity hardware suffices.	New hardware required, else performance limited by STM.
	Performance insensitive to details of cache geometry.	HTM performance depends critically on cache geometry.
SW Support	APIs exist, large body of code and experience, debuggers operate naturally.	APIs emerging, little experience outside of DBMS, breakpoints mid-transaction can be problematic.
Practical applications exist	Yes.	Yes.
Wide applicability	Yes.	Jury still out.

© 2006, 2007 IBM Corporation

Status of STM and HTM

- **There are cases where STM works very well**
  - Scalability can overcome overhead penalty
  - In some special cases, with as few as 4 CPUs
    - In “managed languages” (e.g., Java), with as few as 2 CPUs
- **In other cases, STM is more painful**
  - 20x or, in rare cases, 100x overhead vs. uncontended locking
  - Some recent work makes more aggressive claims
- **Some indications that HTM falling back to STM incurs significantly greater overhead than pure STM**
  - Hardware acceleration for STM?
- **STM can be tailored for specific applications**

© 2006, 2007 IBM Corporation

Where Do Locking and TM Fit In?

- **Locking:**
  - Non-idempotent operations
  - Large critical sections
  - High performance on commodity hardware
  - Good scalability given good engineering (Linux on 1024 CPUs)
    - When data is statically partitionable
  - Large body of successful practice and experience
  - Excellent performance and scalability on read-mostly data
    - In conjunction with RCU or hazard pointers
- **TM:**
  - Large partitionable data structures without static partitionability
  - When no clear lock hierarchy exists (avoid deadlock)
  - Single-threaded software with embarrassingly parallel core
  - TM's applicability may increase if STM performance improves
    - Especially for “managed languages” such as Java

© 2006, 2007 IBM Corporation

Conclusion: Use the Right Tool For The Job!!!

- **There is no silver bullet: successful adoption of multi-threaded/multi-core CPUs will require combination of techniques**
  - But don't take our word for it, ask the TxLinux guys ☺
- **Analogy with engineering: How many types of fasteners are there? How many subtypes? Nail, screw, clip, bolt, glue, joint, magnet...**
- ***Neither locking nor TM solve the fundamental performance and scalability problems (later slides cover ease of use)***
  - STM struggling to achieve parity with uncontended locking, HTM performance benefits over uncontended locking appear to be quite limited
    - Which is a source of much amusement to those of us who have designed and implemented deadlock-immune mechanisms more than an order of magnitude faster than uncontended locking (RCU and Hazard Pointers)
- **Future work: Relativistic Programming**
  - Formalize and generalize existing techniques such as RCU
  - Integrate with other techniques: “use the right tool for the job”
  - Combine performance, scalability, *and* ease of use
  - Account for common hardware properties
    - Allow hardware designers freedom to improve performance

© 2006, 2007 IBM Corporation

Corroboration From SOSP 2007 TxLinux Paper

- **Tried transactions: 6-person-year effort, difficult change**
  - Brings doubts to TM ease-of-use claims...
- **Used locking/transaction hybrid approach: 1 month**
  - Modest performance gains of ~2%
    - Even with favorable-to-TxLinux single-cycle-per-instruction assumption
    - Contrast with tens-of-percent and order-of-magnitude gains from other changes
  - Locking required for I/O and runqueue locks
  - Encountered priority inversion, requiring scheduler support
  - Because TxLinux falls back to locking, deadlock can still arise
    - "While this is unfortunate, deadlock is also a possibility for advanced transaction models that allow open nesting."
    - Suggested solution: use single global lock for transactions that are unlikely to fail
    - However, additional deadlock scenarios are generated by hybrid approach!!!
    - Question: has TxLinux really delivered on the ease-of-programming TM promise?
- **In short, TM is not immune to vicissitudes of large and complex real-world software artifacts**
  - Question: suppose TxLinux team had instead applied HTM to a few key areas in the Linux kernel where deadlock avoidance results in complex code?
    - Might doing so result in a large removed-lines-of-code metric?

© 2006, 2007 IBM Corporation

Recent Work on TM

- **“Inevitable Transactions”: special transactions containing non-idempotent operations (I/O)**
  - Such transactions unconditionally abort any conflicting transactions, thus non-idempotence is OK
  - Allowing more than one concurrent inevitable transaction is necessary to achieve reasonable I/O performance, but feasibility is an open question
    - Compiler might prove that given groups of inevitable transactions cannot conflict (see Maged's recent work)
- **Might use inevitable transactions for real-time**
  - But many applications require large numbers of real-time threads, and performance and scalability are critical

© 2006, 2007 IBM Corporation

Future Work

- **Expand the comparison to include other synchronization mechanisms (message passing, deferred reclamation, RCU)**
- **Investigate combining different mechanisms:**
  - TM and locking (much work in this area)
  - RCU and locking (typical use of RCU)
  - TM and RCU (very little work done here)
- **There might still be hope for a "silver bullet"**
  - But until then, it would be quite foolish to ignore combinations of existing mechanisms

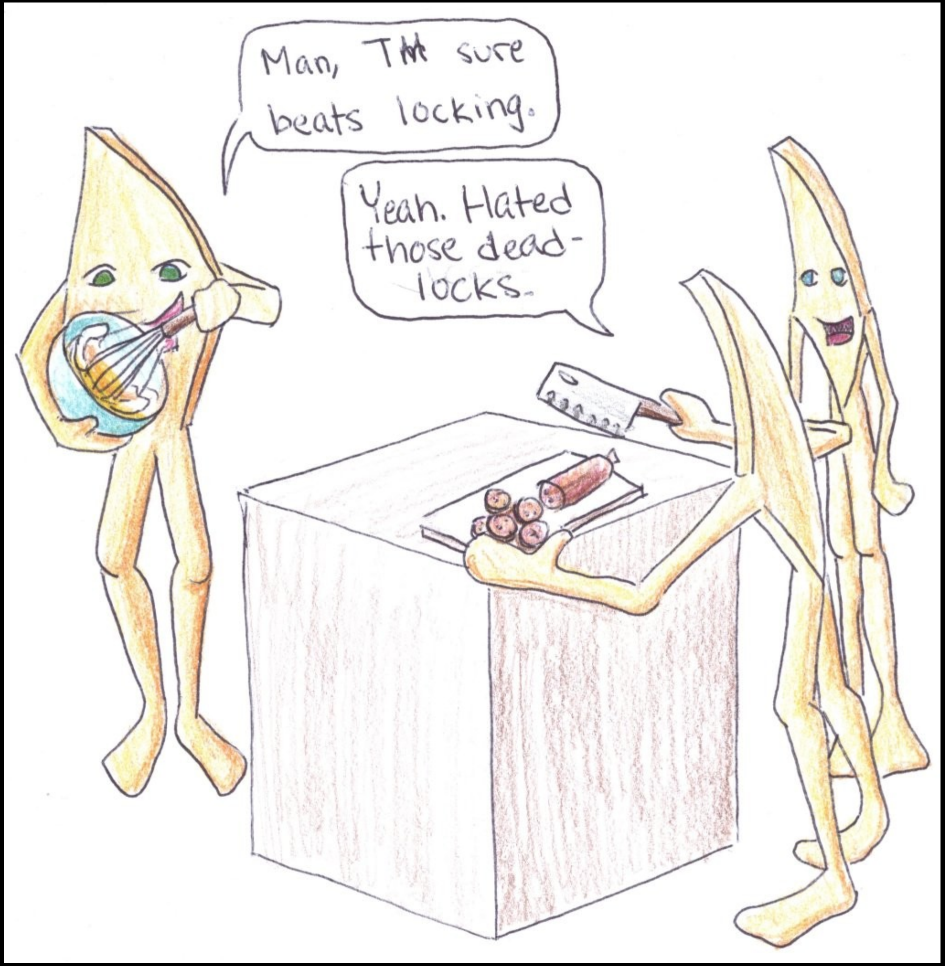
© 2006, 2007 IBM Corporation

End of Balanced Presentation

What Paul Really Thinks

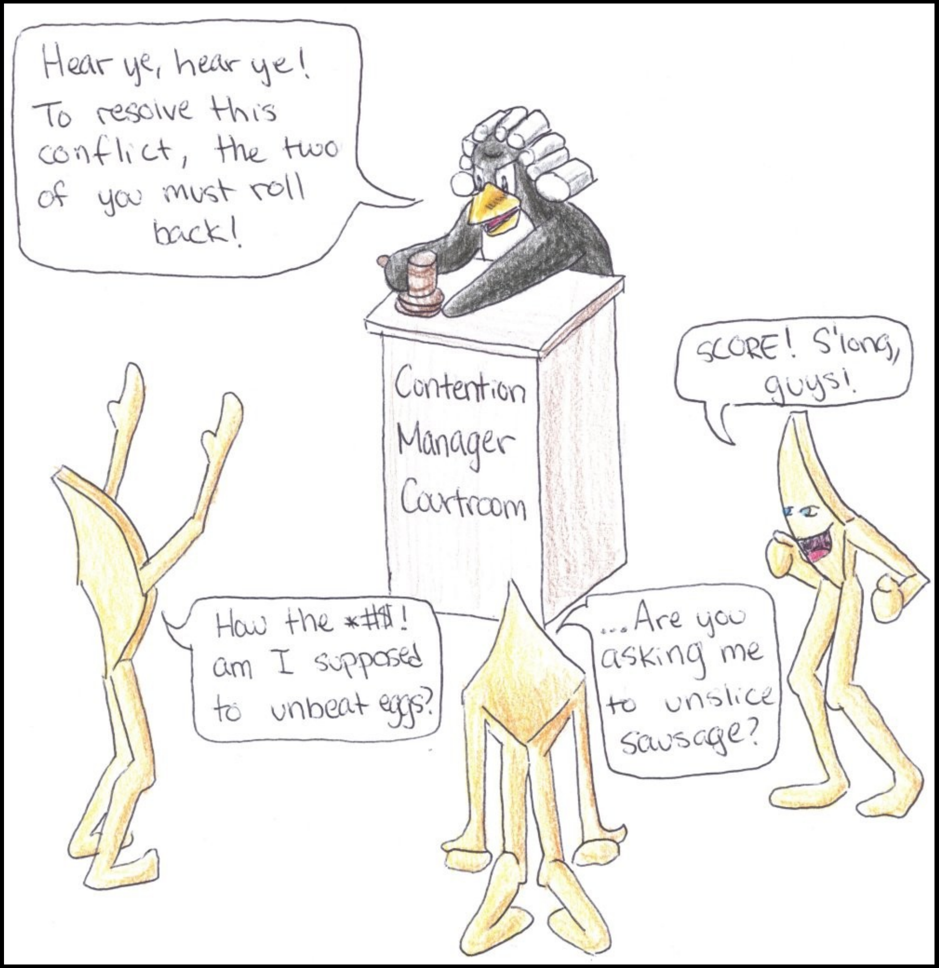
© 2006, 2007 IBM Corporation

TM the Vision



© 2006, 2007 IBM Corporation

TM the Reality: Non-Idempotent Operations


Hear ye, hear ye! To resolve this conflict, the two of you must roll back!
Contention Manager Courtroom
SCORE! S'long, guys!
How the *#$! am I supposed to unbeat eggs?
...Are you asking me to unslice sausage?

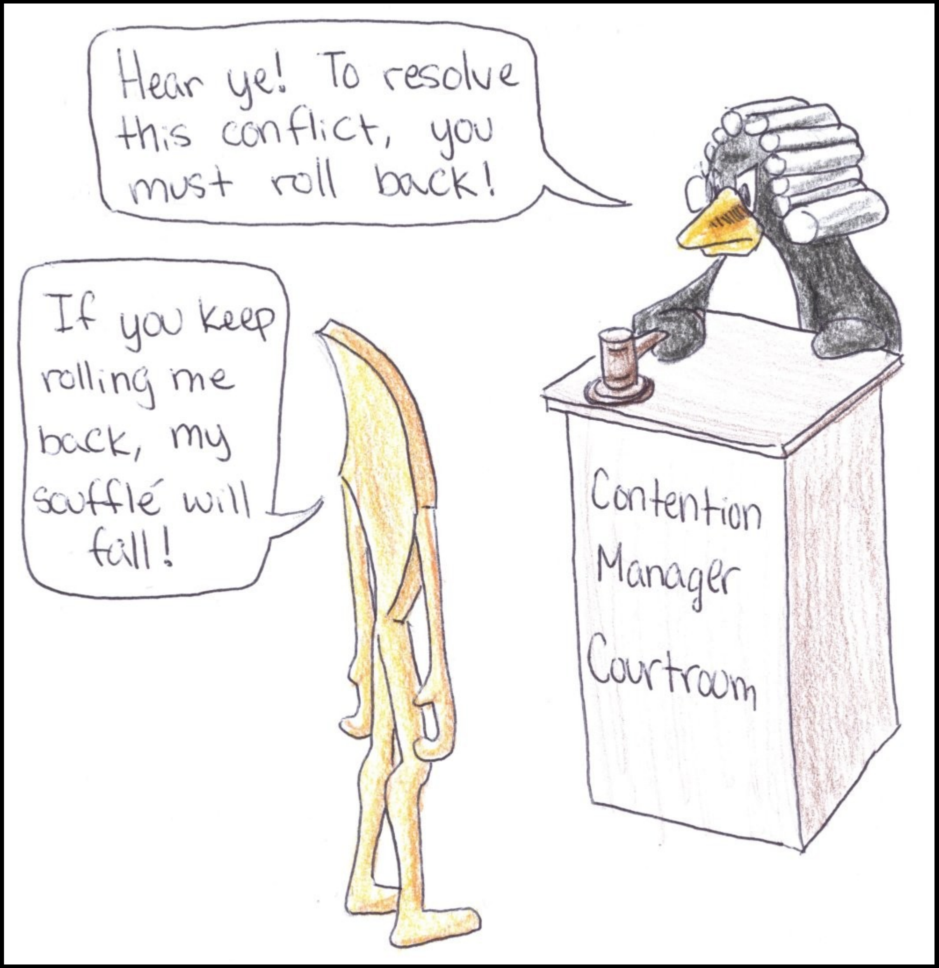
© 2006, 2007 IBM Corporation

TM the Reality: Conflict-Prone Variables

Isn't this the third time today?
Contention Manager Courtroom
But, your honor, lots of recipes need eggs!

© 2006, 2007 IBM Corporation

TM the Reality: Real-Time Response


Hear ye! To resolve this conflict, you must roll back!
If you keep rolling me back, my soufflé will fall!
Contention Manager Courtroom

© 2006, 2007 IBM Corporation

Maged's and Paul's Summary

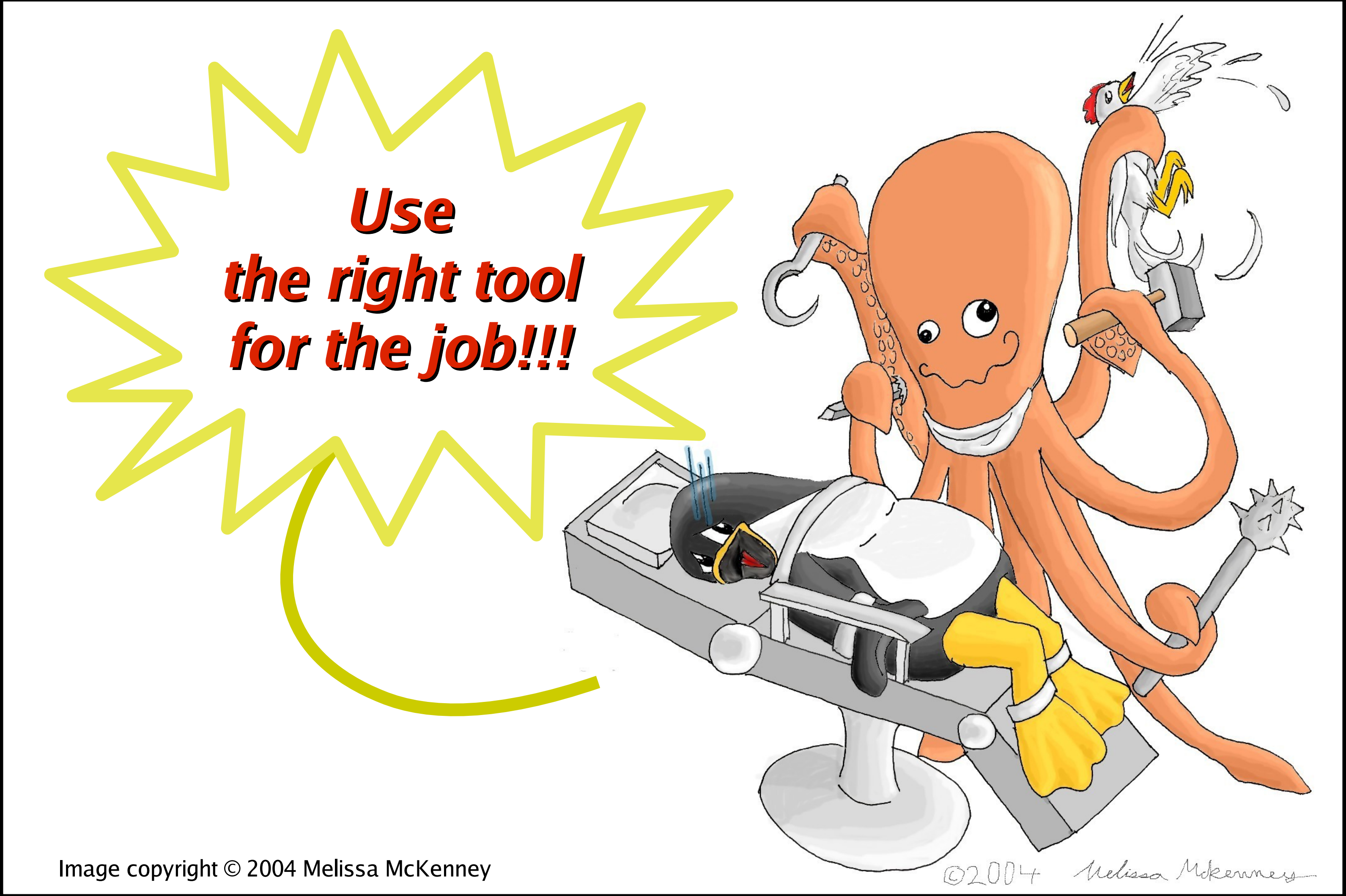


Image copyright © 2004 Melissa McKenney

© 2006, 2007 IBM Corporation

Legal Statement

- This work represents the view of the author and does not necessarily represent the view of IBM.
- IBM, IBM (logo), e-business (logo), pSeries, e (logo) server, and xSeries are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries.
- Linux is a registered trademark of Linus Torvalds.
- Other company, product, and service names may be trademarks or service marks of others.
- This material is based upon work supported by the National Science Foundation under Grant No. CNS-0719851.

© 2006, 2007 IBM Corporation

Questions and Discussion

© 2006, 2007 IBM Corporation