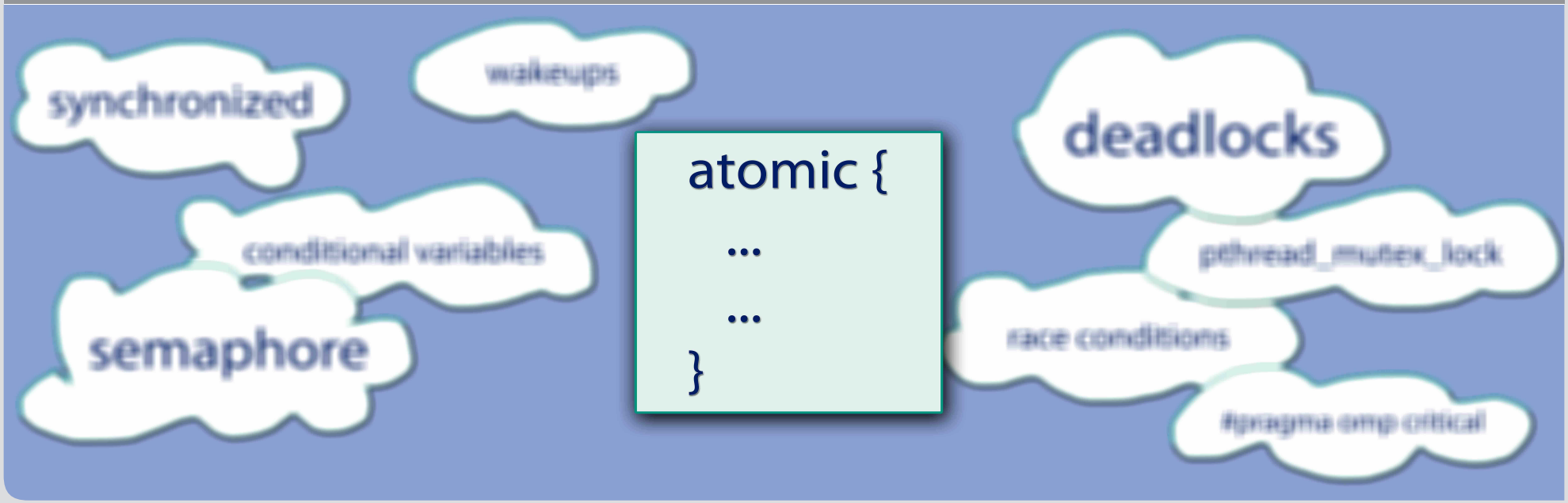
Karlsruher Institut für Technologie

Transactional Memory

Seminar: Sprachen für Parallelprogrammierung

Sven Janko

IPD Snelting, Lehrstuhl Programmierparadigmen



Übersicht

Karlsruher Institut für Technologie

1. Motivation

2. Transactional Memory
   - Transaktion
   - Konflikterkennung
   - Abschluss / Abbruch einer Transaktion

3. Fazit

Motivation

- Nebenläufige Programmierung ist schwierig.
- Forderung nach einem einfachen Modell zur Entwicklung paralleler Programme besteht.

Aktuelle Mechanismen zur Parallelisierung

Karlsruher Institut für Technologie

Threads und Locks

- shared Memory
- mehrere Threads
- Locks, um die Zugriffe zwischen den Threads zu koordinieren

Probleme

- Synchronisation und Koordination ist schwierig
- schlechte Skalierbarkeit
- Entscheidung über Granularität der verwendeten Sperren
- Deadlocks, Livelocks
- Prioritätsumkehr
- Prozessabbruch

Transactional Memory

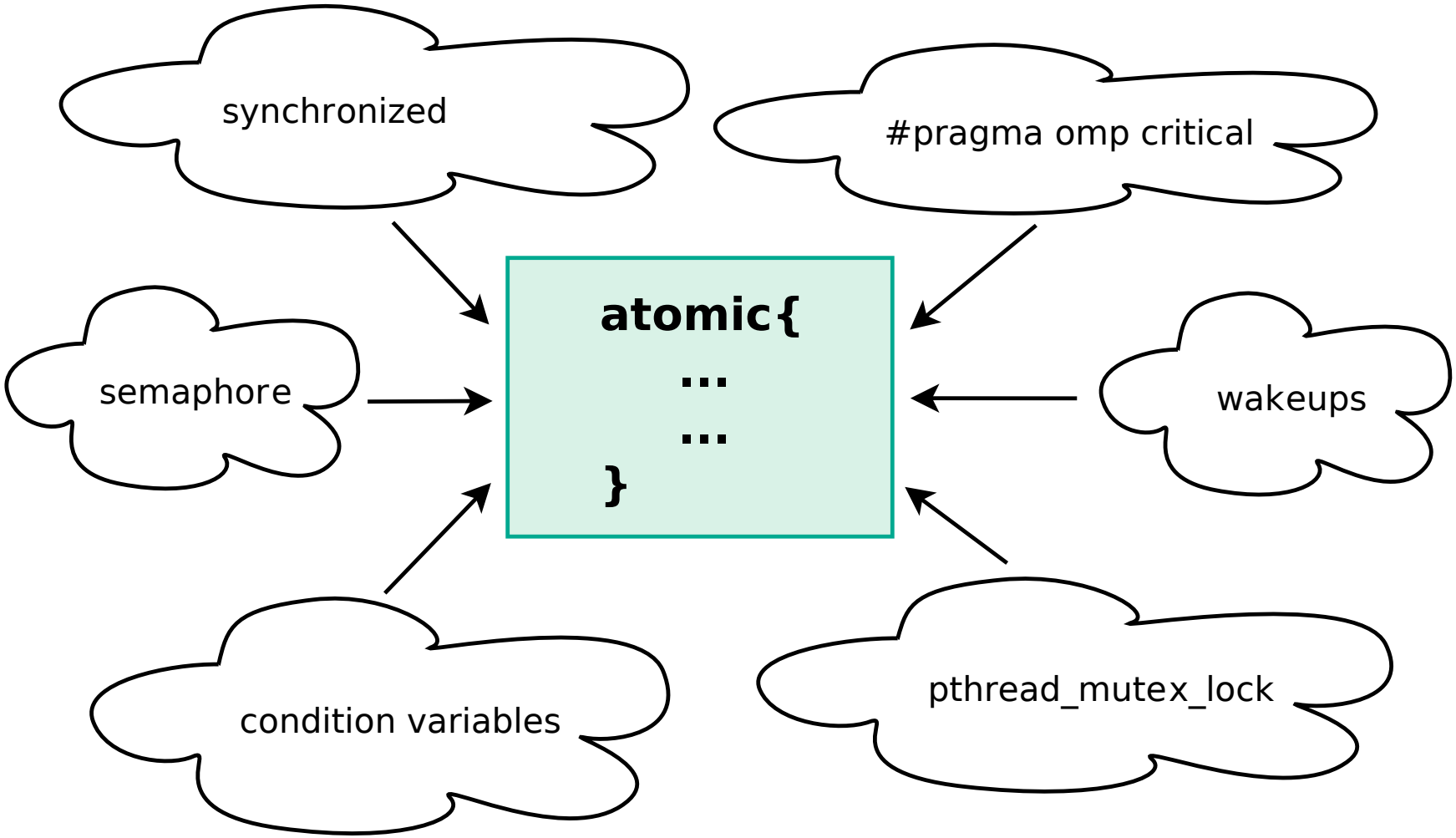
Karlsruher Institut für Technologie

- Hauptspeicherkonzept für Mehrprozessorsysteme
- Idee aus Datenbankkonzept, umgesetzt für Random Access Memory
- stellt Konstrukte für atomare Transaktionen bereit

Grundsätzlich gibt es drei verschiedene Implementierungsmöglichkeiten:

- Software Transactional Memory (STM)
- Hardware Transactional Memory (HTM)
- Hybrid Transactional Memory (HyTM)

atomic-Konstrukt



Transaktion

Eine Transaktion fasst mehrere Aktionen/Befehle zu einer Sequenz zusammen. Es werden drei der vier Eigenschaften des AKID-Prinzips (engl.: ACID) gefordert:

- **Atomarität**: „Alles oder nichts.“
- **Kontinuität**: Das System befindet sich zu jeder Zeit in einem konsistenten Zustand.
- **Isolation**: Jede Transaktion läuft korrekt ab, unabhängig davon, ob noch andere Transaktionen parallel ausgeführt werden.
- ~~Dauerhaftigkeit~~: RAM ist flüchtig.

Synchronisationsmechanismen

Karlsruher Institut für Technologie

Die Synchronisationsmechanismen sorgen dafür, dass der parallele Zugriff auf Datenstrukturen nicht mehr blockiert. Algorithmen werden hier folgendermaßen klassifiziert:

- **obstruction-free**: Ein Thread macht nur Fortschritte, falls keine Konflikte auftreten.
- **lock-free**: Das System als Ganzes macht Fortschritte.
- **wait-free**: Jeder Thread macht Fortschritte.

Beispiel: Geldauszahlung

Karlsruher Institut für Technologie

Mit Locks

- Sperre auf das Konto wird angefordert.
- Falls erhalten: Zugriff auf dieses Konto ist für andere gesperrt.
  1. Lese Kontostand und prüfe Bonität.
  2. Ziehe den auszuzahlenden Betrag ab.
  3. Schreibe neuen Kontostand.
  4. Gebe Sperre wieder frei.

Mit Transaktionen

1. Lese Kontostand und prüfe Bonität.
2. Ziehe den auszuzahlenden Betrag ab.
3. Schreibe neuen Kontostand.
4. Schließe Transaktion ab. Falls das fehlschlägt: Starte erneut bei 1.

Konflikterkennung

Karlsruher Institut für Technologie

Grundsätzlich verfolgt man ein optimistisches Grundprinzip (optimistic concurrency) um Konflikte zu erkennen, weshalb innerhalb einer Transaktion alle Lese- und Schreibzugriffe auf gemeinsam genutzten Speicher in einem Log festgehalten werden müssen.

- version number
- read-sets
- write-sets

Es gibt zwei Vorgehensweisen zur Konflikterkennung:

- **eager**: Konflikte werden früh erkannt.
- **lazy**: Konflikte werden spät erkannt.

Konflikte

Es wird zwischen drei unterschiedlichen Konflikten unterschieden, die in zwei Klassen eingeordnet werden können:

true conflicts

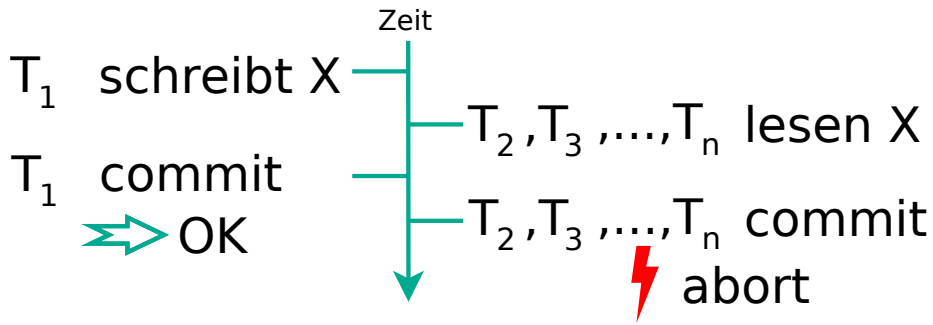
- **W-W**: Zwei Transaktionen greifen schreibend auf dieselbe Speicherzelle zu.

false conflicts

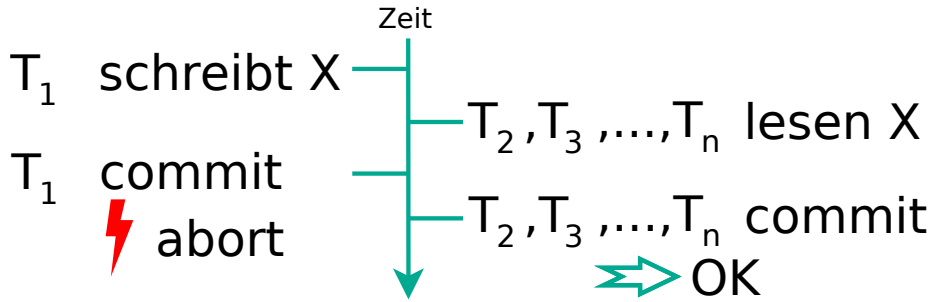
- **W-R**: Eine Transaktion greift schreibend, eine andere lesend auf dieselbe Speicherzelle zu.
- **R-W**: Eine Transaktion greift lesend, eine andere schreibend auf dieselbe Speicherzelle zu.

Commit-Time Validation / Commit-Time Invalidation

- Validation



- Invalidation



Abschluss einer Transaktion

- **commit**: Alle lokalen Änderungen werden in den Speicher geschrieben
- **abort**: Änderungen werden nicht übernommen.

Beispiel STM:

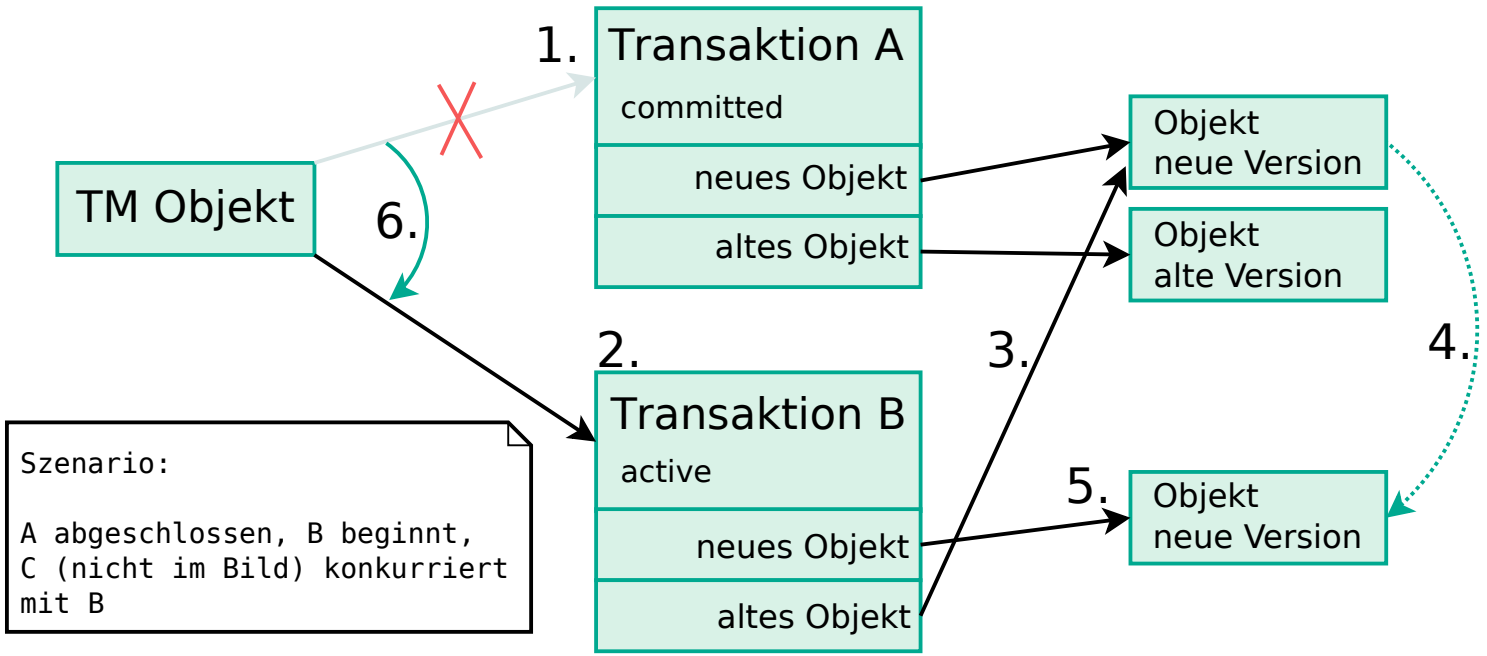
- Verwendung von Compare and Swap (CAS)
  - Atomare Instruktion, die von der Hardware angeboten wird.
  - Drei Parameter: Speicheradresse, Vergleichswert, neuer Wert

Abbruch einer Transaktion (abort)

- Der Zustand vor dem Start der Transaktion muss komplett wiederhergestellt werden.
- **Alle** Operationen einer Transaktion müssen rückgängig gemacht werden können.
- STM: Typischerweise wird auf einer Kopie des Speicherbereichs gearbeitet.

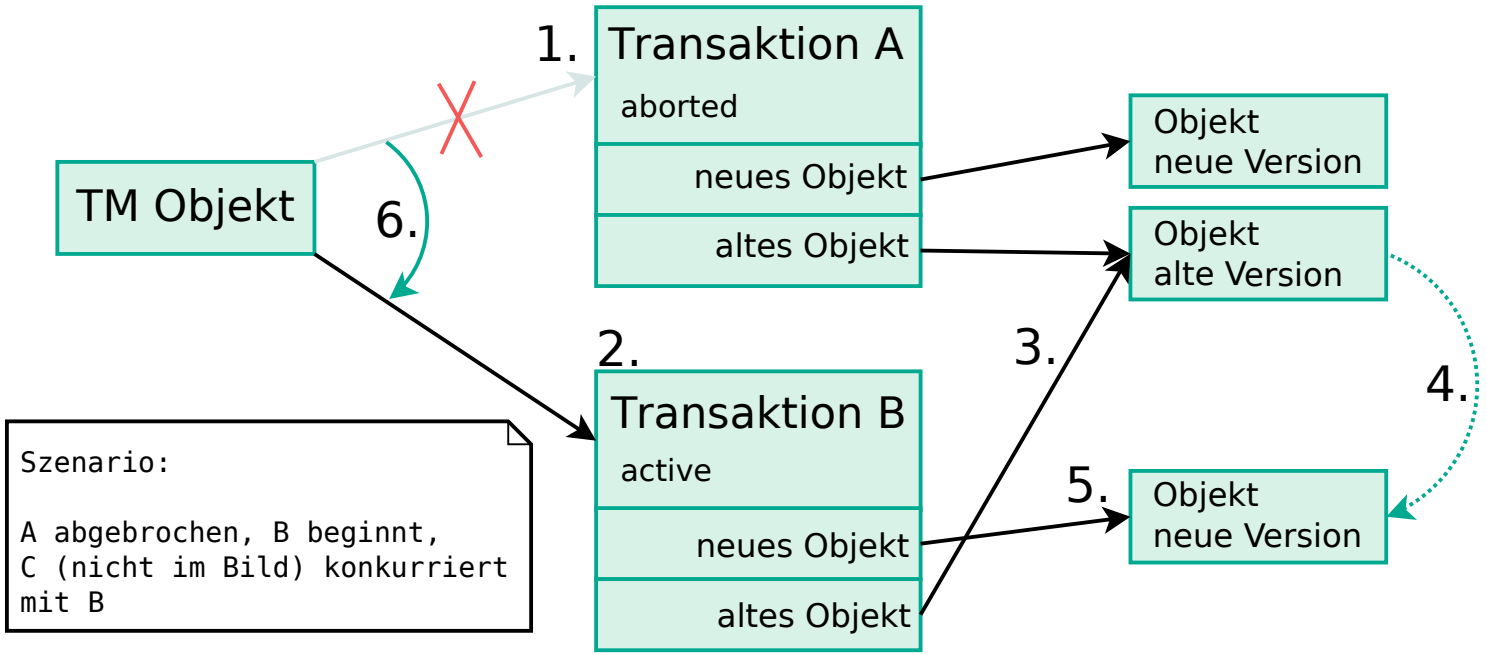
STM Beispiel (1)

1. Zeiger auf den alten Lokator A
2. Neuer Lokator für B
3. Zeiger "altes Objekt" von B zeigt auf die aktuellen Daten von A
4. Kopie der aktuellen Daten von A wird erzeugt
5. Zeiger "neues Objekt" von B zeigt auf diese Kopie.
6. Zeiger des TM Objekts wird mittels CAS umgebogen



STM Beispiel (2)

1. Zeiger auf den alten Lokator A
2. Neuer Lokator für B
3. Zeiger "altes Objekt" von B zeigt auf die alten Daten von A
4. Kopie der alten Daten von A wird erzeugt
5. Zeiger "neues Objekt" von B zeigt auf diese Kopie.
6. Zeiger des TM Objekts wird mittels CAS umgebogen



TM in der Praxis

Karlsruher Institut für Technologie

HTM

- Rock Prozessor von Sun.
  - Zwei neue Instruktionen und ein neues Register.

STM

- Es existiert eine Vielzahl von Ansätzen und Implementierungen.
- Performanz: Nur die wenigsten Implementierungen erreichen die Performanz von fein-granularen Sperren.
- Sept ’08: „STM: Why Is It Only a Research Toy?“

Fazit

- Attraktive Alternative zu den bisherigen Sperrkonstrukten.
- Leichteres Programmiermodell mit mehr Garantien.
- Verspricht Korrektheit, Skalierbarkeit und Effizienz der parallelen Programme.

Aber

- Semantikprobleme:
  - Was ist mit E/A-Operationen?
  - Schwache oder starke Atomarität? (weak vs. strong atomicity)
  - Ausnahmebehandlung?
- Nichtdeterminismus erschwert Fehlerfindung.
- Kompatibilität mit bisherigen Konstrukten zur Parallelisierung.
- Lange Transaktionen können verhungern.

Links

- http://haskell.org/ghc/
- http://en.wikipedia.org/wiki/Software_transactional_memory
- http://labs.oracle.com/spotlight/2007/2007-08-13_transactional_memory.html
- http://software.intel.com/en-us/articles/intel-c-stm-compiler-prototype-edition-20/
- http://en.wikipedia.org/wiki/Rock_(processor)

Ende

Karlsruher Institut für Technologie

Vielen Dank für eure Aufmerksamkeit!

Gibt es Fragen?

Quellenangaben

- Transactional Memory: Architectural Support for Lock-Free Data Structures, *Maurice Herlihy, J. Eliot B. Moss*, May 1993
  http://portal.acm.org/citation.cfm?id=165164
- Concurrent Programming Without Locks, *Keir Fraser, Tim Harris*, 2007
  http://portal.acm.org/citation.cfm?id=1233307.1233309&coll=GUIDE&dl=GUIDE&CFID=92043844&CFTOKEN=83703377
- Flexible Decoupled Transactional Memory Support, *Arrvindh Shriraman, Sandhya Dwarkadas, Michael L. Scott*,
  http://portal.acm.org/citation.cfm?id=1381306.1382134&coll=GUIDE&dl=GUIDE&CFID=92043844&CFTOKEN=83703377
- Software Transactional Memory: Why Is It Only a Research Toy?, *Cascaval, Blundell, Michael, Cain, Wu, Chiras, Chatterjee*, Sept 08
  http://portal.acm.org/citation.cfm?id=1454456.1454466&coll=GUIDE&dl=GUIDE&CFID=92043844&CFTOKEN=83703377
- Composable Memory Transactions, *Tim Harris, Simon Marlow, Simon Peyton Jones, Maurice Herlihy*, 2005
  http://portal.acm.org/citation.cfm?id=1065952
- An Efficient Software Transactional Memory Using Commit-Time Invalidation, *Justin E. Gottschlich, Manish Vachharajani, Jeremy G. Siek*, 2010
  http://portal.acm.org/citation.cfm?id=1772954.1772970&coll=GUIDE&dl=GUIDE&CFID=92043844&CFTOKEN=83703377
- Hybrid Transactional Memory, *Peter Damron, Alexandra Fedorova, Yossi Lev, Victor Luchangco, Mark Moir, Daniel Nussbaum*, 2006
  http://portal.acm.org/citation.cfm?id=1168857.1168900&coll=GUIDE&dl=GUIDE&CFID=92043844&CFTOKEN=83703377
- Vorlesung: Multikern-Rechner und Rechnerbündel, *Victor Pankratius*, KIT WS 2009
  http://wwwipd.ira.uka.de/Tichy/uploads/folien/166/Cluster05STM.pdf