Multicore Programming: Transactional Memory

Vaitl Christian

07 Mai 2009

4 Software Transactional Memory(STM)

- Funktionsweise
- Beispielimplementation
- Probleme
- Vor- und Nachteile

5 Hardware Transactional Memory(HTM)

- Funktionsweise
- Beispiel: ATLAS
- Vor- und Nachteile

6 Hybrid Transactional Memory(HyTM)

- Funktionsweise

Problem

Bei bisherigen Speicherkonzepten hat ein Prozessor oder ein Programm seine Speicherbereiche stets exklusiv für sich beansprucht. Andere parallel laufende Software konnten darauf nicht zugreifen.

Lösung

Der traditionelle Ansatz zur Synchronisation nebenläufiger Programme ist die Verwendung von Locks.
Möchte ein Prozess exklusiven Zugriff auf eine Ressource, muss er eine Sperre bei einem Verwaltungsprozess anfordern.

Probleme

- Deadlocks
- Livelocks
- Problem-Potenziale beim Prozessabbruch
- Prioritätsumkehr
- Warten trotz höherer Priorität

Transaktion

- **Atomarität**: Aktion wird komplett ausgeführt oder hinterlässt keine Spuren bei Abbruch
- **Kontinuität**: System muss immer in einem Konsistenten Zustand sein
- **Isolation**: Ergebnis der Transaktion muss korrekt sein

Verschachtelte Transaktionen

- **flach**: Abbruch der inneren Transaktion bricht auch die äußere ab
- **geschlossen**: Abruch der inneren Transaktion bricht äußere nicht ab. Änderungen erst nach korrektem Abschluss der äußeren Transaktion sichtbar
- **offenen**: Abruch der inneren Transaktion bricht äußere nicht ab. Änderungen der inneren Transaktion nach korrektem Abschluss der inneren Transaktion sichtbar

Synchronisationsmechanismen

- **Lock-free**: Das Gesamtsystem kann sich nicht in einen Deadlock geraten
- **Wait-free**: Jeder Faden macht Fortschritte, sogar wenn andere Fäden ihre Ausführung verzögern oder abbrechen

Compare and Swap (CAS)

- **v**: Einen Vergleichswert, der den alten Wert einer Operation repräsentiert
- **n**: Den neuen Wert der Operation
- **M**: Adresse der Hauptspeicherstelle aus der v am Anfang der Operation gelesen wurde

Nach der Operation wird der Wert an M mit v verglichen:

- Stimmen sie überein, wird der neue Wert n auf M geschrieben
- Stimmen sie nicht überein, wird n verworfen und die Operation noch einmal mit dem neuen v ausgeführt

Grundprinzip

- optimistisches Verfahren im Vergleich zu Locks
- Aufzeichnung aller read/write Aktivitäten
- Überprüfung nach Abschluss der Transaktion
  - Bei Konflikt werden die Daten verworfen und die Transaktion neu gestartet
  - Bei Erfolg werden die neuen Werte in den Haupspeicher geschrieben

Eager Versioned

- Direkte Aktualisierung der Speicherinhalte
- Auffangen der überschriebenen Werte in Undo-Log
- Rückspielen des Logs bei Konflikt
- Schnelle Commits
- Langsame Aborts
- Keine Fehlertoleranz
- Schwache Atomizität

Lazy Versioned

- Pufferung der Schreibzugriffe
- Aktualisierung der Speicherinhalte aus Puffer bei Commit
- Verwerfen des Puffers bei Konflikt
- Langsame Commits
- Schnelle Aborts
- Fehlertolerant
- Starke Atomizität

Read/Write Sets

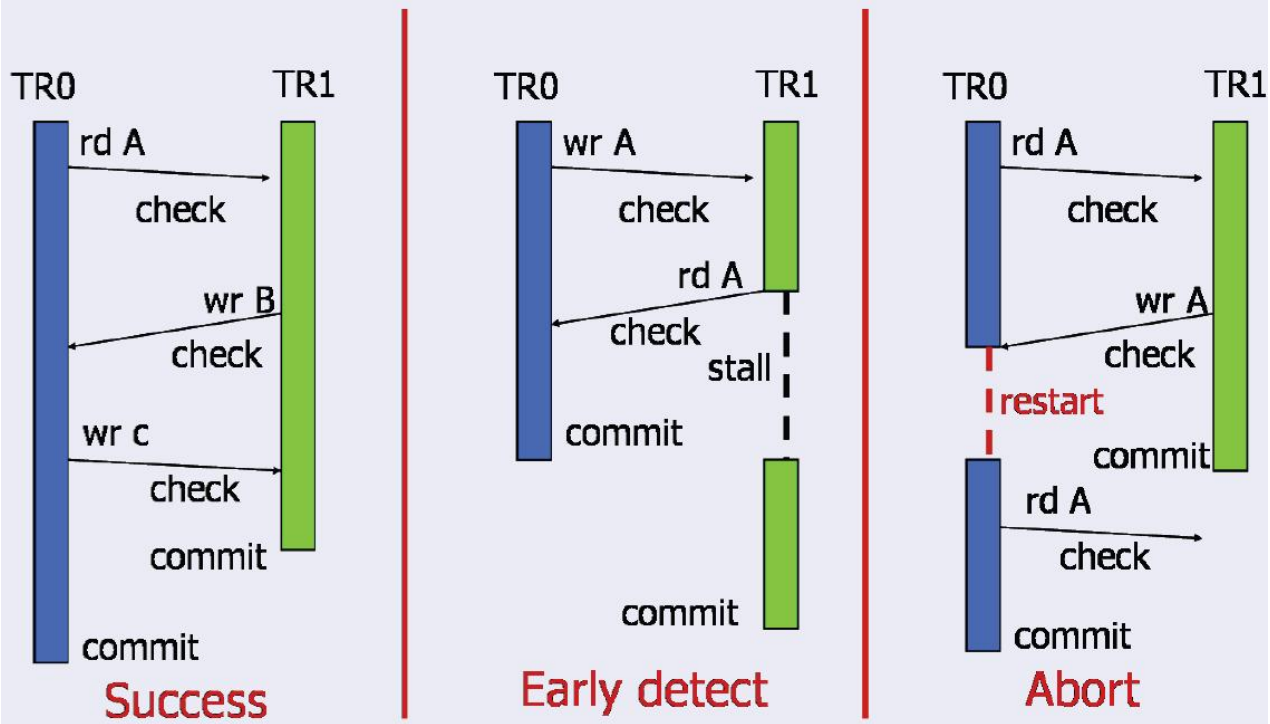
Um Konflikte zu entdecken und zu behandeln, hat ist eine Transaktion normalerweise mit zwei Datensätzen verbunden.

- Read Set: Hier wird die Speicheradresse jedes 'read'-Befehls der Transaktion gespeichert
- Write Set: Hier wird die Speicheradresse, und der Wert jedes 'write'-Befehls der Transaktion gespeichert

Eager Checking

- Überprüfung während Lade- und Speicheroperationen
- Pessimistisches Modell
- Verwendung des Contention Managers zur Entscheidung
- Read und Write Sets müssen auf dem ganzen System sichtbar sein

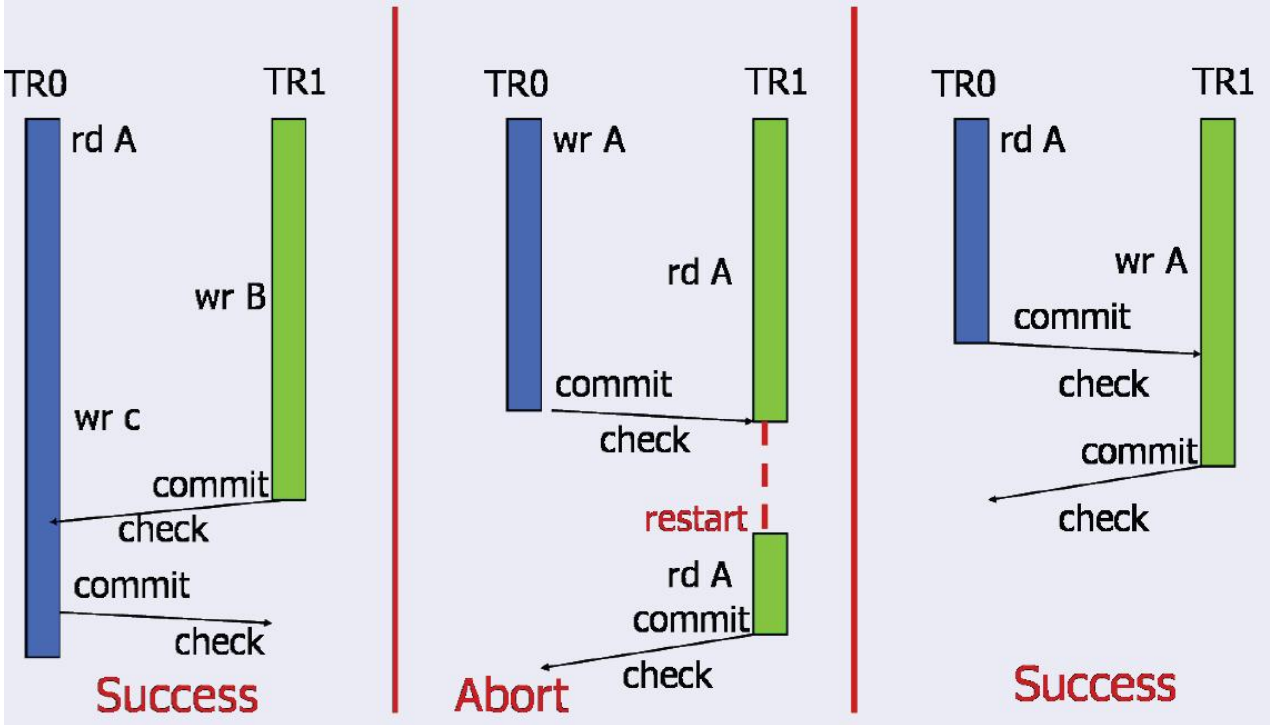
Beispiel


TR0
TR1
rd A
check
wr B
check
wr c
check
commit
commit
Success
TR0
TR1
wr A
check
rd A
check
stall
commit
commit
Early detect
TR0
TR1
rd A
check
wr A
check
restart
commit
rd A
check
commit
Abort

Lazy Checking

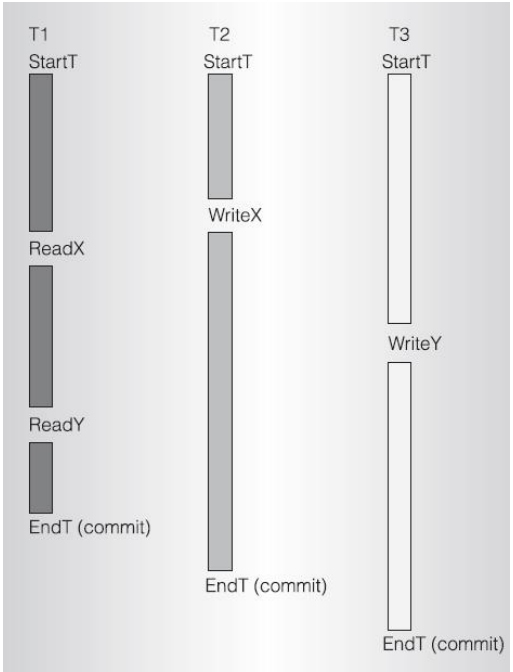
- Überprüfung auf Konflikt erst mit Commit
- Optimistisches Modell
- Mischen unterschiedlicher Mechanismen für Lade- und Speicheroperationen möglich

Beispiel


TR0
TR1
rd A
wr B
wr c
commit
check
commit
check
Success
TR0
TR1
wr A
rd A
commit
check
restart
rd A
commit
check
Abort
TR0
TR1
rd A
wr A
commit
check
commit
check
Success

Problem

Die Entscheidung welche der Transaktionen abgebrochen wird, ist im Allgemeinen aber sehr komplex.



Wortgranularität

- Minimierung von False-Sharing
- Steigender Aufwand

Objektgranularität

- Geringer Aufwand
- Entspricht Programmierer-Denkweise
- Große Objekte bedingen False-Sharing und damit unnötige Aborts
- Nur SW-TM bzw. Hybridmodell

Cachezeilengranuarität

- Kompromiß zwischen Objekt- und Wortgranularität
- Tauglich für HTM und STM

Mix and Match

- Ausrichtung an TM-Modell und Gegebenheiten

Vor- und Nachteile

Vorteile

- Kohärenz-Kontrolle einfacher
- Kommunikation zwischen Prozessen seltener
- Weniger Synchronisationspunkte nötig
- Illusion von Einprozessorsystemen

Nachteile

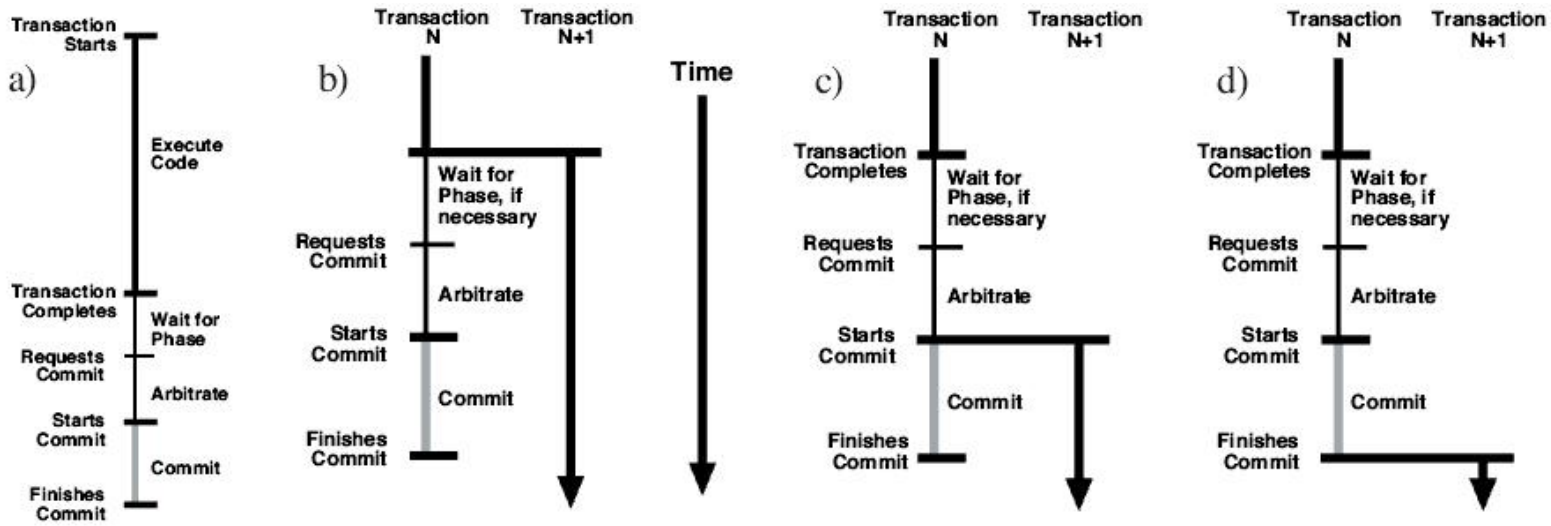
- Interprozessor-Bus muss hohe Bandbreite haben
- Falls Commit zu groß für den Cache, muss der Bus gesperrt werden
- Algorithmus in Hardware naher Sprache kann schneller sein

Optimierungsmöglichkeiten

- Double Buffering
- Hardware-Controlled Transactions
- Localisation of Memory References

Double Buffering

- Zusätzliche Write-Buffers und read- bzw. write-Bits in der Cacheline



Folge: CPU kann die nächste Transaktion starten bevor die Aktuelle committet.

Hardware-Controlled Transactions

- Teilt das Programm selbstständig in Transactions
- Fasst kleinere Transactions zu einer großen zusammen
- Gelegentlich Barrier setzen, um tote Threads zum Leben erwecken

Folge: Höherer Speedup.

Localisation of Memory References

- Load & Store, die nur lokal benötigt werden, nicht broadcasten(z.B. Stack Referenzen)
- Realisierbar, indem man entweder Seiten als local-only markiert oder mit Befehlen wie local-load und local-store arbeitet

Folge: Bandbreite wird gespart

Funktionsweise eines typischen STM

- Deklarierung gemeinsamer Speicherbereiche durch '**open**'
- Festlegung ob nur lesend oder auch schreibend zugegriffen wird
- Lazy Versioning
- Objektgranualität
- Locator wird erstellt
- Inbesitznahme eines Blocks für Schreibzugriffe durch '**aquire**'

Konflikterkennung bei aquire()

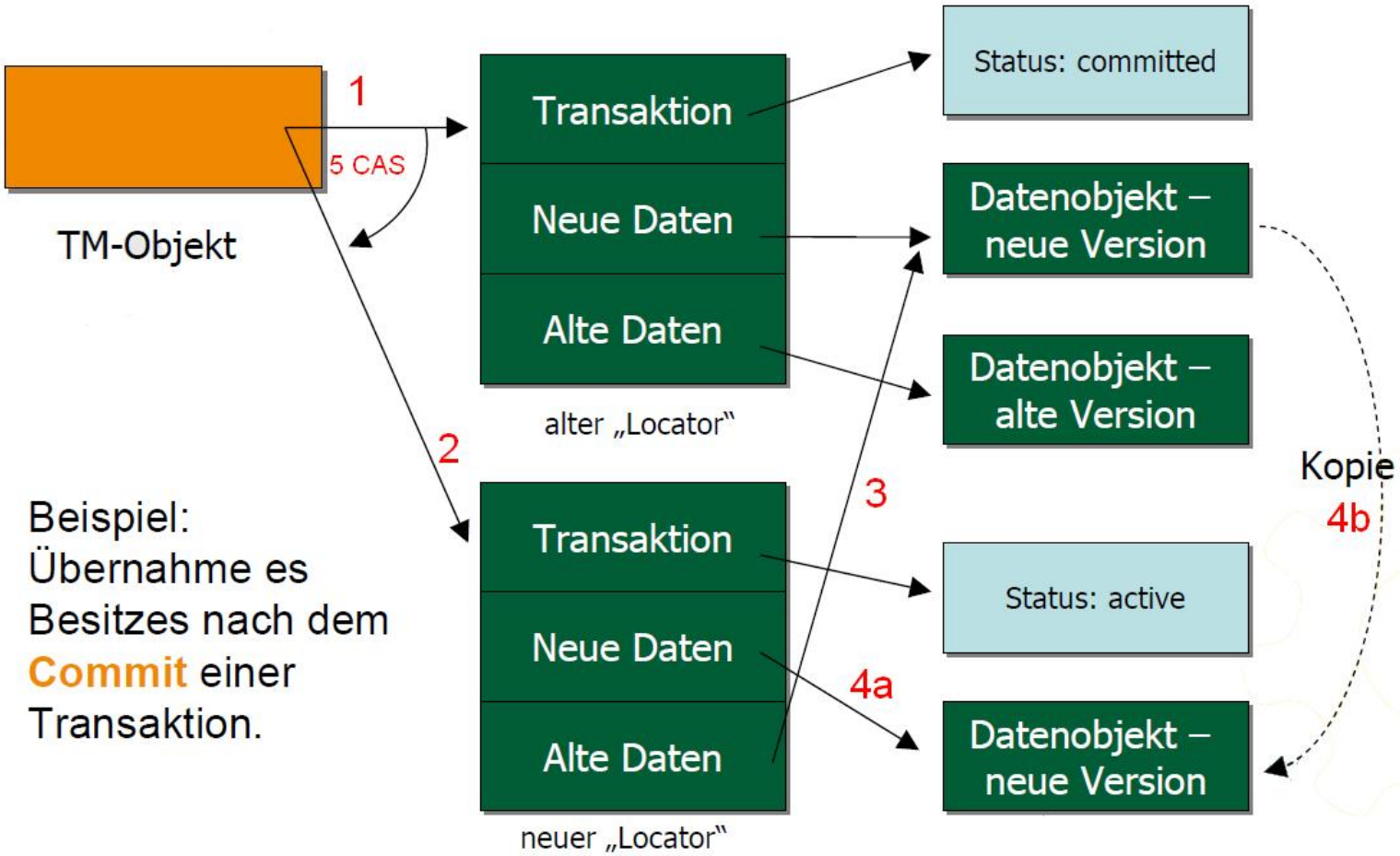
- Wenn eine andere Transaktion den fraglichen Bereich in Besitz hält, kann eine der beiden Transaktionen abgebrochen werden.
- Andere Transaktionen können abgebrochen werden, wenn sie die Daten gelesen haben, für die hier grade der Besitz angefordert wird.
- Die Transaktion kann abgebrochen werden, wenn Daten ihrer vorhergehenden Lesezugriffe inzwischen überschrieben wurden.

Implementation

Beispiel atomarer Zähler:

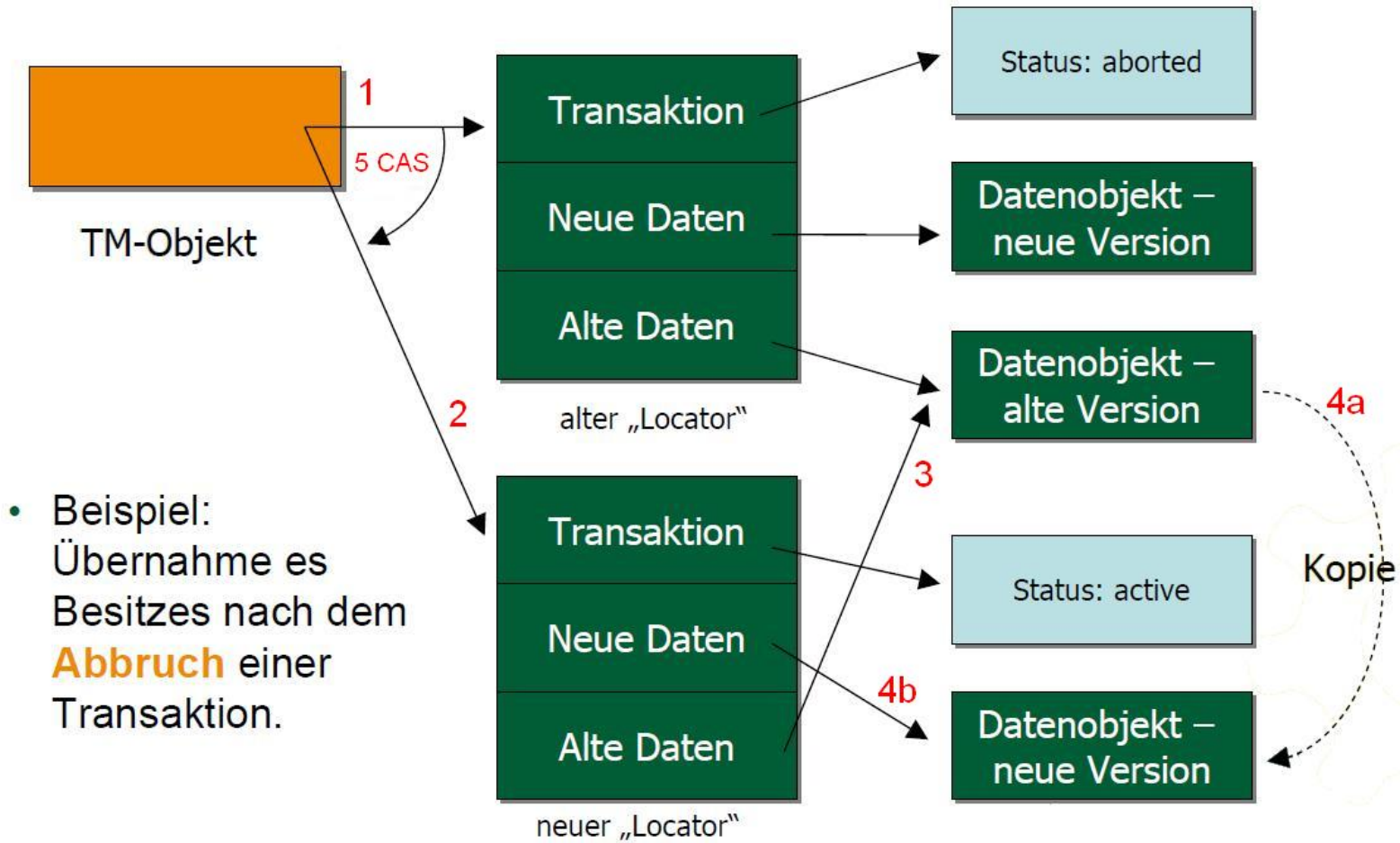
```
//Initialisierung
Counter   counter = new Counter(0);
TMObject tmObject = new TMObject(counter);

//Verwendung innerhalb einer Transaktion
//Faden ruft beginTransaction auf
Counter counter=(Counter)tmObject.open(WRITE);
counter.inc();
...
```

Besitzübernahme bei Commit



- Beispiel: Übernahme es Besitzes nach dem **Commit** einer Transaktion.

Besitzübernahme bei Abort



- Beispiel: Übernahme es Besitzes nach dem **Abbruch** einer Transaktion.

Probleme

- Deadlocks durch Endlosschleifen
- Null-Referenzen

Diese Probleme müssen in Abhängigkeit von OS, Sprache und STM gelöst werden.

Beispiel Deadlock

```
atomic {
  if (x != y)
    while (true) {}
}
```

Transaktion A

```
atomic {
  x++;
  y++
}
```

Transaktion B

Vor- und Nachteile

Vorteile

- Bereits auf bestehenden Systemen einsetzbar

Nachteile

- Schlechte Skalierbarkeit

Funktionsweise

- einige neue Befehle in der ISA
- zwischenspeicherung der neuen Daten in abgegrenzten Buffern(oder Caches)

Neue Befehle

- **STR**: Starten einer Transaktion
- **ETR**: Beenden einer Transaktion
- **TLD**: load Operation
- **TST**: write (store) Operation

Optional:

- **ABR**: Abbruch einer Transaktion
- **VAL**: Validierung einer Transaktion

Cachelines

- **Read Bit**: Cacheline wurde während einer Transaktion gelesen
- **Modified Bit**: Falls Cachline (teilweise) geändert wurde

Zusätzlich möglich:

- **Renamed Bit**: Zeigt an, welcher Teil einer Cacheline geändert wurde

ATLAS ein HTM-Prototype

- Erstes CMP-System mit Shared Memory und TM-Support
- 8 PowerPC 405 + 1 für OS
- Auf BEE2-Multi-FPGA Board gemappt
- Läuft mit 100MHz
- Verwendet kein MESI-Protokoll

ATLAS ein HTM-Prototype

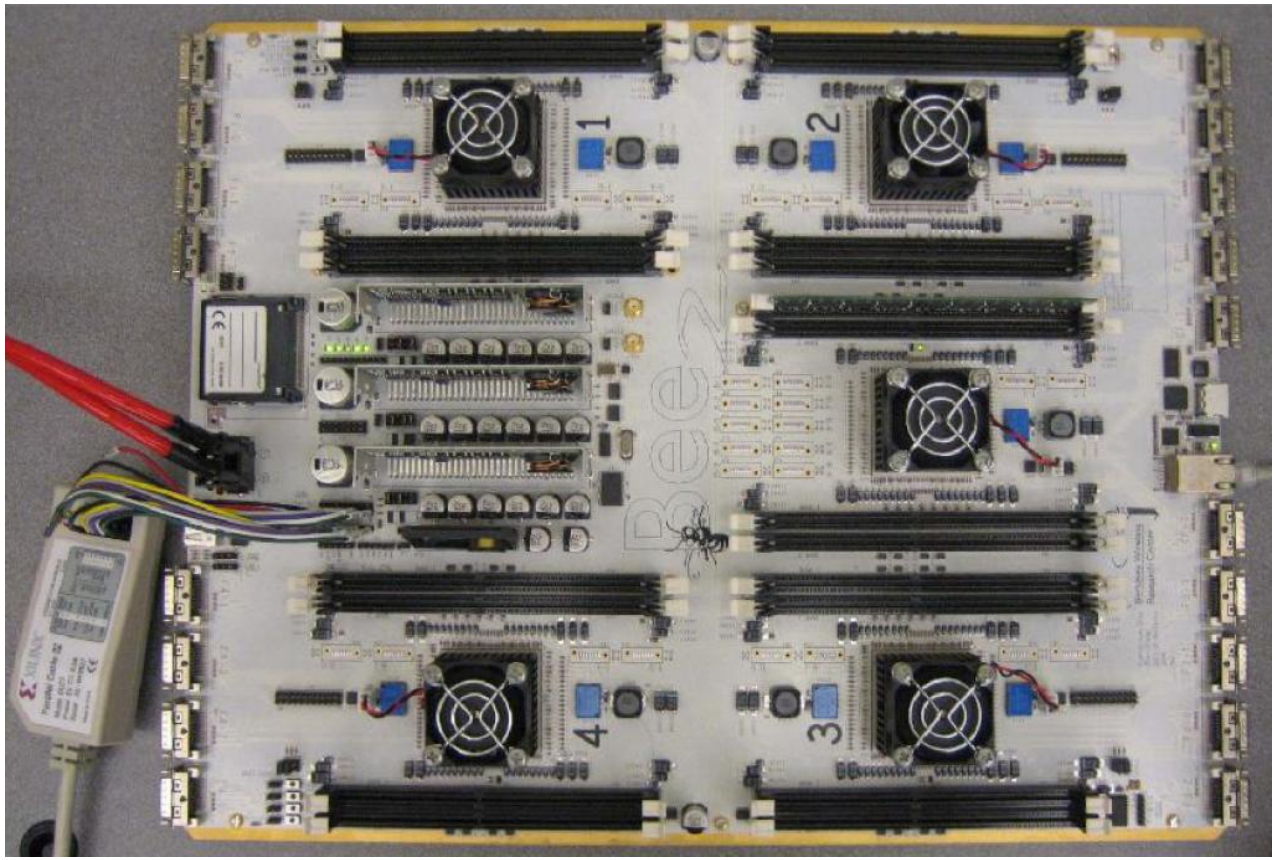


Abbildung: Aufbau des ATLAS

ATLAS ein HTM-Prototype

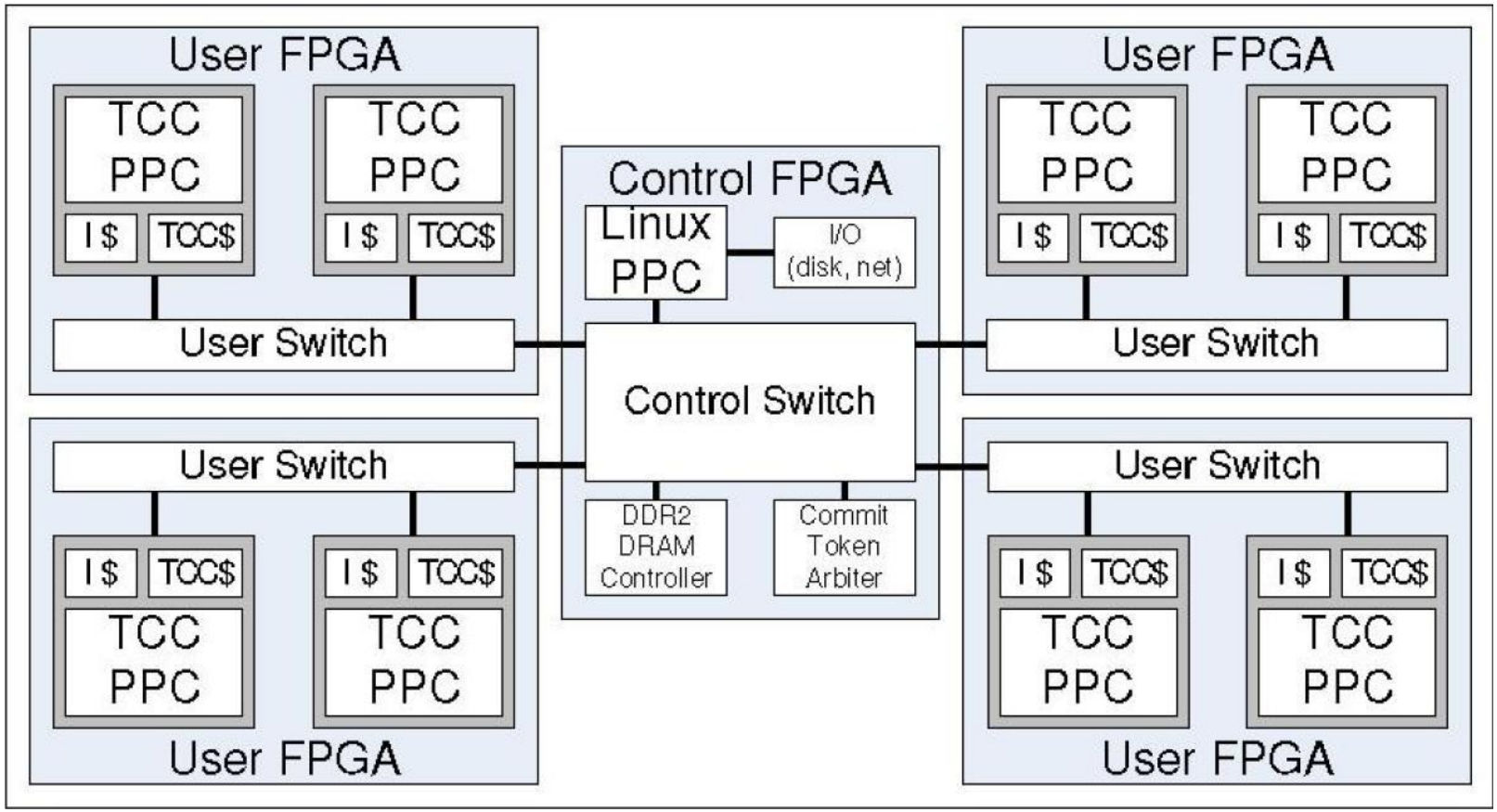


Abbildung: Schaltbild des ATLAS

ATLAS ein HTM-Prototype

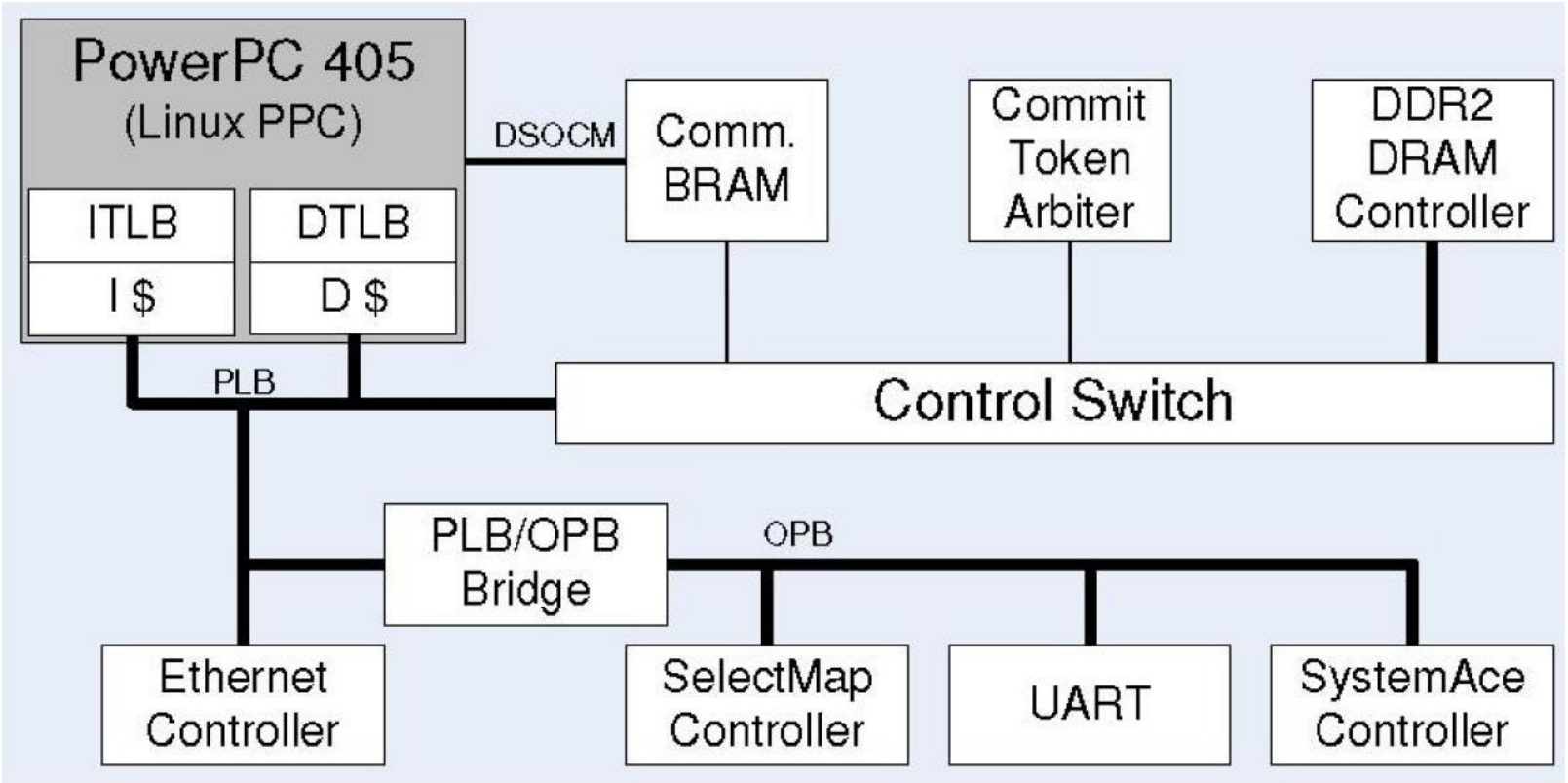


Abbildung: Control FPGA

ATLAS ein HTM-Prototype

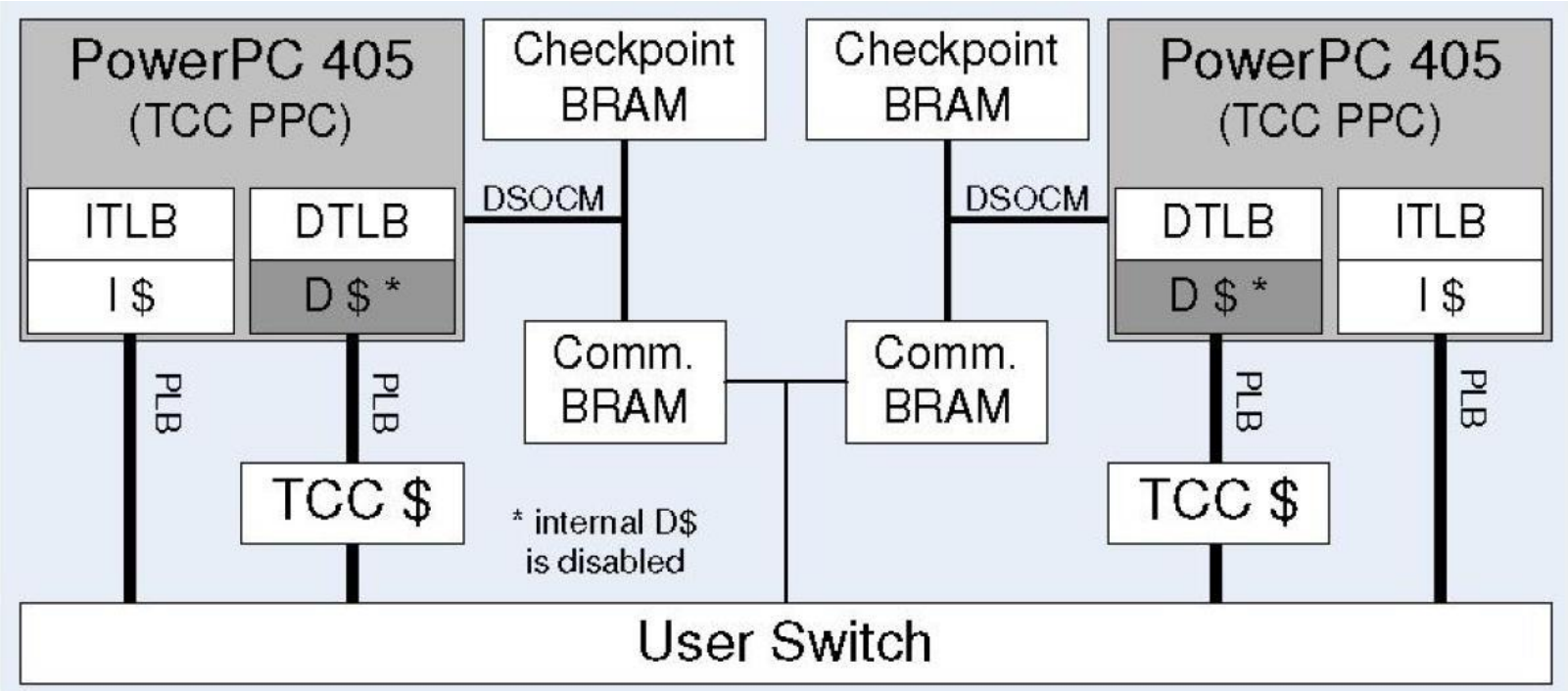
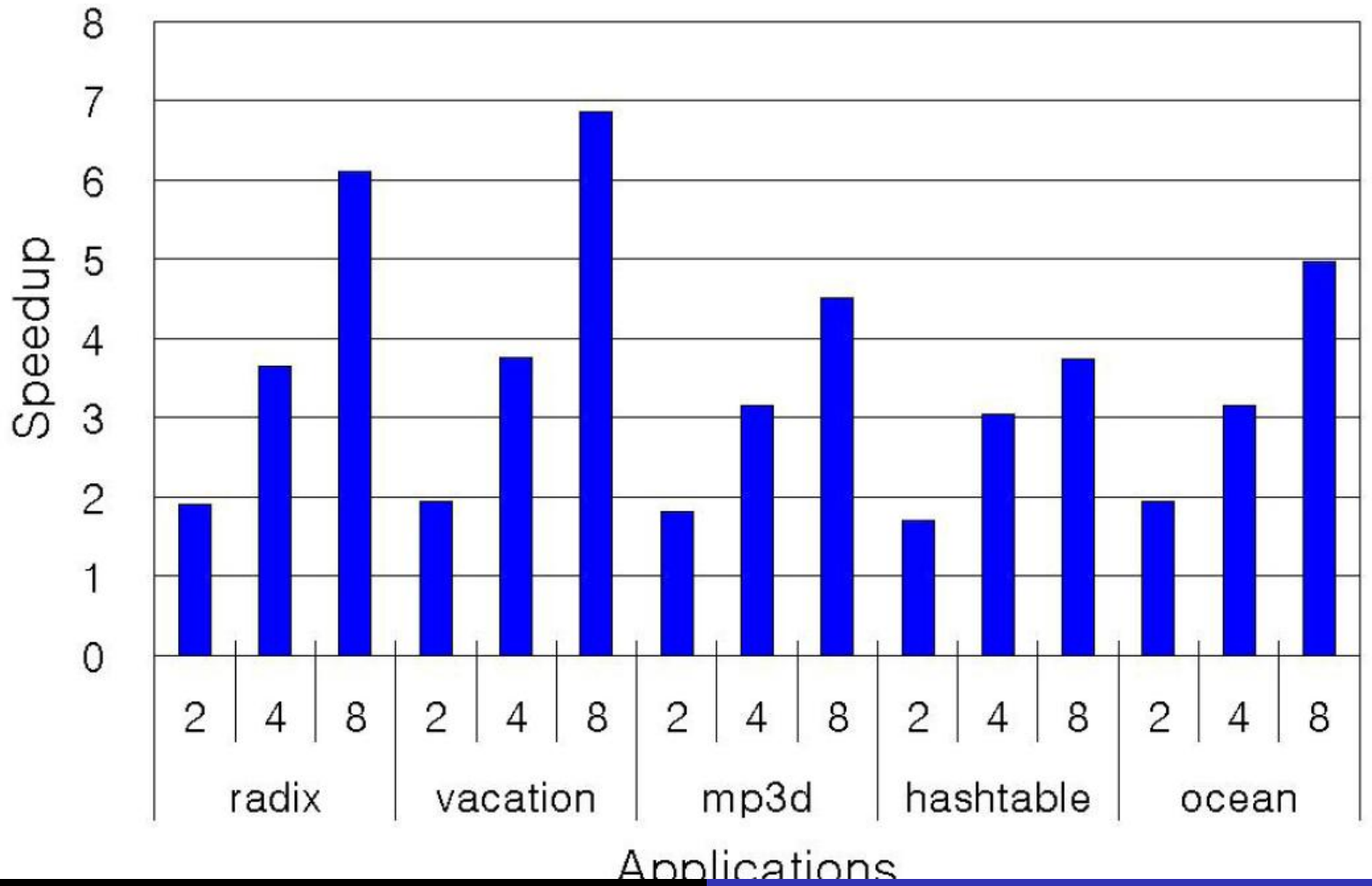


Abbildung: User FPGA

ATLAS ein HTM-Prototype



Vor- und Nachteile

Vorteile

- Skaliert sehr gut
- Atomizität transparent für alle Anwendungen

Nachteile

- Erfordert neue Hardware
- Nicht einfach zu Programmieren
- Nicht so flexibel wie STM

Funktionsweise

- Transaktion zunächst per HTM
- Falls gescheitert, übernahme durch STM (**fall-back**)

Konflikterkennung und -lösung

- STM/STM-Konflikte durch STM
- HTM/HTM-Konflikte durch HTM
- HTM/STM-Konflikte durch zusätzlichen Code in HTM-Codepfad

Interoperabilität

- Zusammenspiel zwischen HTM und STM
- Überprüfung des Transaction Records
- Sicherstellung, daß Zugriffsadresse nicht durch STM-Transaktion blockiert ist
- Zwei Basisfunktionen
  - HTMReadBarrier(*addr*)
    - Lesen, falls *addr* nicht durch STM-Transaktion blockiert
    - Abort bei Konflikt
  - HTMWriteBarrier(*addr*)
    - Schreiben, falls addr nicht durch STM-Transaktion blockiert
    - Abort bei Konflikt

Ende

VIELEN DANK FÜR IHRE AUFMERKSAMKEIT!