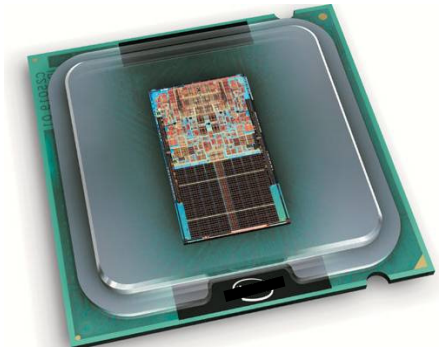
JAVAFORUM 2008 stuttgart



Can multi-core do more?

Andreas Salzborn, Marco Scheuermann

„Ich kann freilich nicht sagen, ob es besser werden wird, wenn es anders wird; aber soviel kann ich sagen: es muß anders werden, wenn es gut werden soll."

[Georg Christoph Lichtenberg, 1742-1799, Schriftsteller und Physiker]

„Wir können freilich nicht sagen, ob es schneller werden wird, wenn es anders wird; aber soviel können wir sagen: es muß anders werden, wenn es schneller werden soll."

[was Lichtenberg über die Notwendigkeit der Umstellung von Einzelprozessor auf Multiprozessor Software Architekturen sagen würde]

Willkommen zum 11. Java Forum Stuttgart 2008 und zur Beantwortung der Frage

„Can multi-core do more?“

„Das ist doch aber alles nichts Neues, das haben wir früher auch schon so gemacht“

„Ich weiß gar nicht was Ihr da 45 Minuten lang erzählen wollt“

Das ist neu


HDD-Kapazität: 500 GB
Prozessorgeschw.: 2,4 GHz
Prozessortyp: Core2 Quad Q6600
699,-
Arbeitsspeicher: 2048 MB; Grafikkarte: GF8600 GS; Netzw.karte: LAN; optische Laufwerke: BlueRay-Combo-Laufwerk (BlueRay Leselaufwerk, DVD +/- Brenner); Ausgänge: 5 x USB 2.0, D-Sub, DVI, ...

Topics of the day

- Prozessortrends
- Paralleles Programmieren
- Ein Pattern für den Multicore
- Checkliste für das Design paralleler Programme
- Eine sehr kritische Betrachtung des „MultiCore Hypes“
- Fragen und Diskussion

Topics of the day

Prozessortrends

Parallele Programmierung

Ein Pattern für den Multicore

Checkliste für das Design paralleler Programme

Eine sehr kritische Betrachtung des „MultiCore Hypes“

Fragen und Diskussion

Core Roadmap (Desktop)

Architektur	**Single Core SMT***
Kerne	1
Simultane Threads	2
Erscheinungs-jahr	2002

*SMT = Simultanes Multithreading

Core Roadmap (Desktop)

Architektur	Single Core SMT*	Dual Core
Kerne	1	2
Simultane Threads	2	2
Erscheinungs-jahr	2002	2005

*SMT = Simultanes Multithreading

Core Roadmap (Desktop)

Architektur	Single Core SMT*	Dual Core	Triple Core Quad Core
Kerne	1	2	3/4
Simultane Threads	2	2	3/4
Erscheinungs-jahr	2002	2005	2007

*SMT = Simultanes Multithreading

Core Roadmap (Desktop)

Architektur	Single Core SMT*	Dual Core	Triple Core Quad Core	Dual Core SMT Quad Core SMT
Kerne	1	2	3/4	2/4
Simultane Threads	2	2	3/4	4/8
Erscheinungs-jahr	2002	2005	2007	2008

*SMT = Simultanes Multithreading

Core Roadmap (Desktop)

	Simultaneous Multithreading (SMT)	MultiCore		MultiCore & SMT	
Architektur	**Single Core SMT***	**Dual Core**	**Triple Core Quad Core**	**Dual Core SMT Quad Core SMT**	**Octo Core SMT**
Kerne	1	2	3/4	2/4	8
Simultane Threads	2	2	3/4	4/8	16
Erscheinungs-jahr	2002	2005	2007	2008	2009

Simultaneous Multithreading (SMT) (1/2)

- **Ziel**: Erhöhung der Instruktionen pro Takt auf **einem** physikalischen Prozessor unter optimaler Ausnutzung der Pipeline Architektur
- bekanntester Vertreter: „**Hyperthreading**“
- während ein Thread z.B. nach einem **Cache Miss** (Daten sind bei Anfrage nicht im Cache vorhanden) auf die Daten aus dem Hauptspeicher warten muss, kann ein zweiter Thread parallel ausgeführt werden

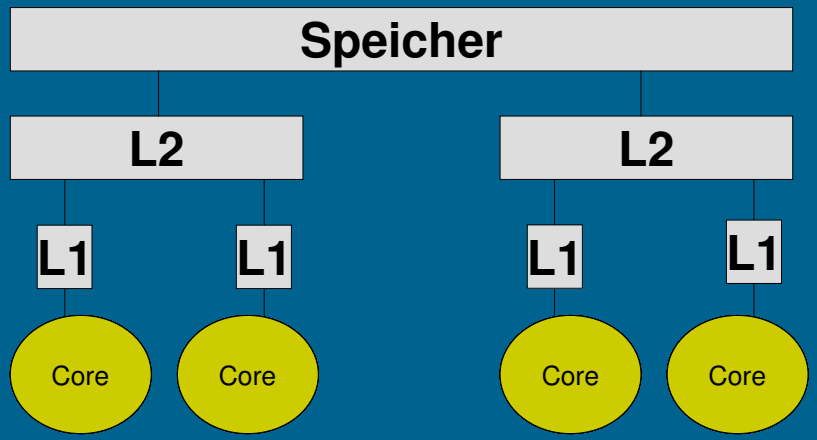
Simultaneous Multithreading (SMT) (2/2)

- aus Sicht des Betriebssystems ergeben sich zwei logische Prozessoren
- jeder logische Prozessor verfügt über einen eigenen Interrupt-Controller sowie Zustands- und Datenregister
- jeder logische Prozessor legt seinen Zustand in einem separaten Bereich ab

ABER: ALU und FPU unterliegen dem gemeinsamen Zugriff
Performanzgewinn: bis ca. 30 % (Quelle **intel**)

Multicore

- mehrere Prozessorkerne auf einem Chip
- jeder Kern verfügt über eigene ALU, FPU und Register
- **es kann wirklich parallel gerechnet werden !**
- hierarchisches Design:



- z.B. AMD X2, X3, X4, Intel Core2Duo, Intel Core2Quad

Prozessortrends - *Fazit*

Sowohl für SMT– als auch für MultiCore Prozessoren gilt:

Es müssen die Techniken des parallelen Programmierens angewandt werden, um die zugrunde liegende Hardware nutzen und daraus resultierend eine mögliche Steigerung der Performance erreichen zu können.

Topics of the day

Prozessortrends

Parallele Programmierung

Ein Pattern für den Multicore

Checkliste für das Design paralleler Programme

Eine sehr kritische Betrachtung des „MultiCore Hypes“

Fragen und Diskussion

Parallele Programmierung

- Konzepte
- Package java.util.concurrent
- Synchronisation
- Beispiel

Parallele Programmierung

Arten von Parallelität

- implizite Parallelität:
  - vom konkreten Compiler und Hardware abhängig (z.B. superskalare Prozessoren)
  - kein direkter Einflussbereich des Software-Entwicklers
  - auch funktionale Programmiersprachen
- explizite Parallelität:
  - explizit vom Entwickler als „parallel“ gekennzeichneter Code
    (z.B.: Compiler Direktiven / OpenMP)
  - über Concurrent APIs / Job APIs parallel implementierte Tasks

Parallele Programmierung

Ebenen der Parallelität

- Instruktionsebene:
  - Compiler; falls keine Datenabhängigkeit zwischen n-Anweisungen; superskalare Prozessoren Ziel „Schallgrenze“ von 1 Anweisung pro Takt durchbrechen.
- Anweisungsebene:
  - mehrere Anweisungen auf denselben oder verschiedenen Daten
- Schleifenebene:
  - unterschiedliche Iterationen zueinander parallel (It. unabhängig)
  - Anweisungen im Schleifenrumpf in Form von Vektoranweisungen
- Funktionsebene:
  - ganze Funktionen / Algorithmen eines Programms zueinander parallel

Parallele Programmierung

Entwurf paralleler Programme

- Fragestellung:
  - Welche Teilaufgaben eignen sich parallel zueinander ausgeführt zu werden?
- verschiedene Berechnungsströme:
  - Zerlegen der Gesamtaufgabe in Teilaufgaben Tasks
- Tasks:
  - Größe der Tasks Granularität
  - Erkennen / Definieren von Abhängigkeiten / Gesetzmäßigkeiten
  - Zuweisung von Ressourcen und Prioritäten
  - Definieren von Lebensläufen / Lebenszyklen (für zyklische P.)
  - Synchronisation der von einander abhängigen Tasks

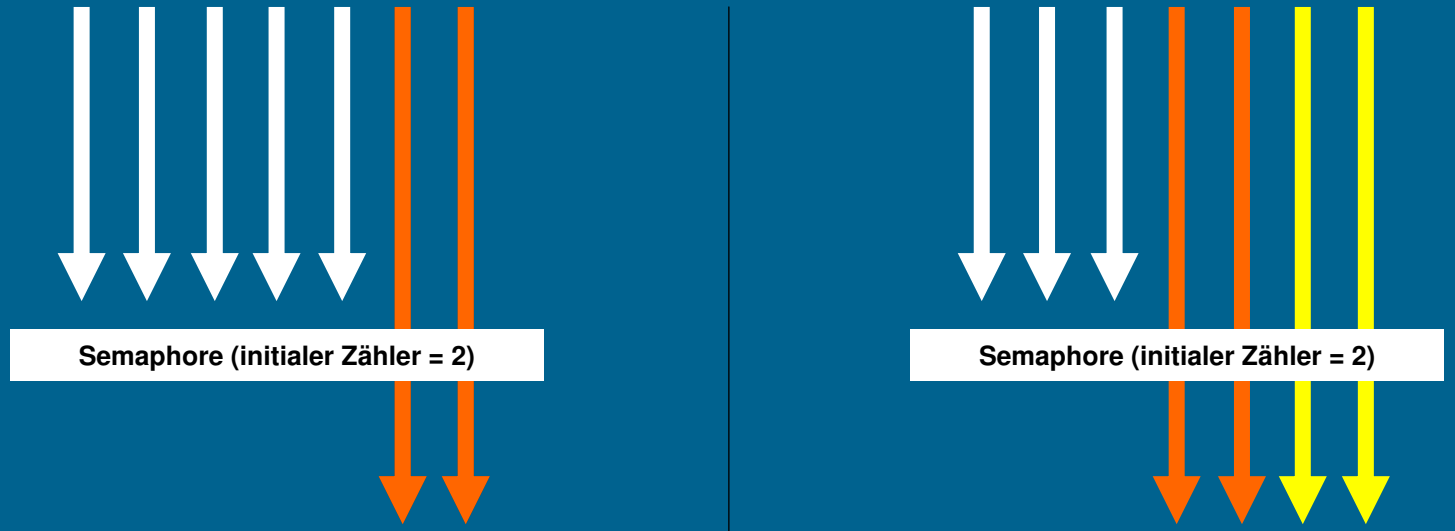
Parallele Programmierung

Package java.util.concurrent

- Synchronisation
  - Semaphore
  - CountDownLatch / CyclicBarrier
  - Rendezvous Punkte (Exchanger)
  - Locks
- atomare Datentypen (AtomicBoolean, AtomicInteger, AtomicReference, …)
- parallele Datenstrukturen (mehr Leser als Schreiber)
  - ConcurrentMap
  - CopyOnWriteArrayList / CopyOnWriteArraySet
- Queues (BlockingQueue – Familie)
- Threadpools (ExecutorService - Familie)

Synchronisation (1/4)

Semaphore („ähnlich einer Warteschlange an der Kasse im Supermarkt“)

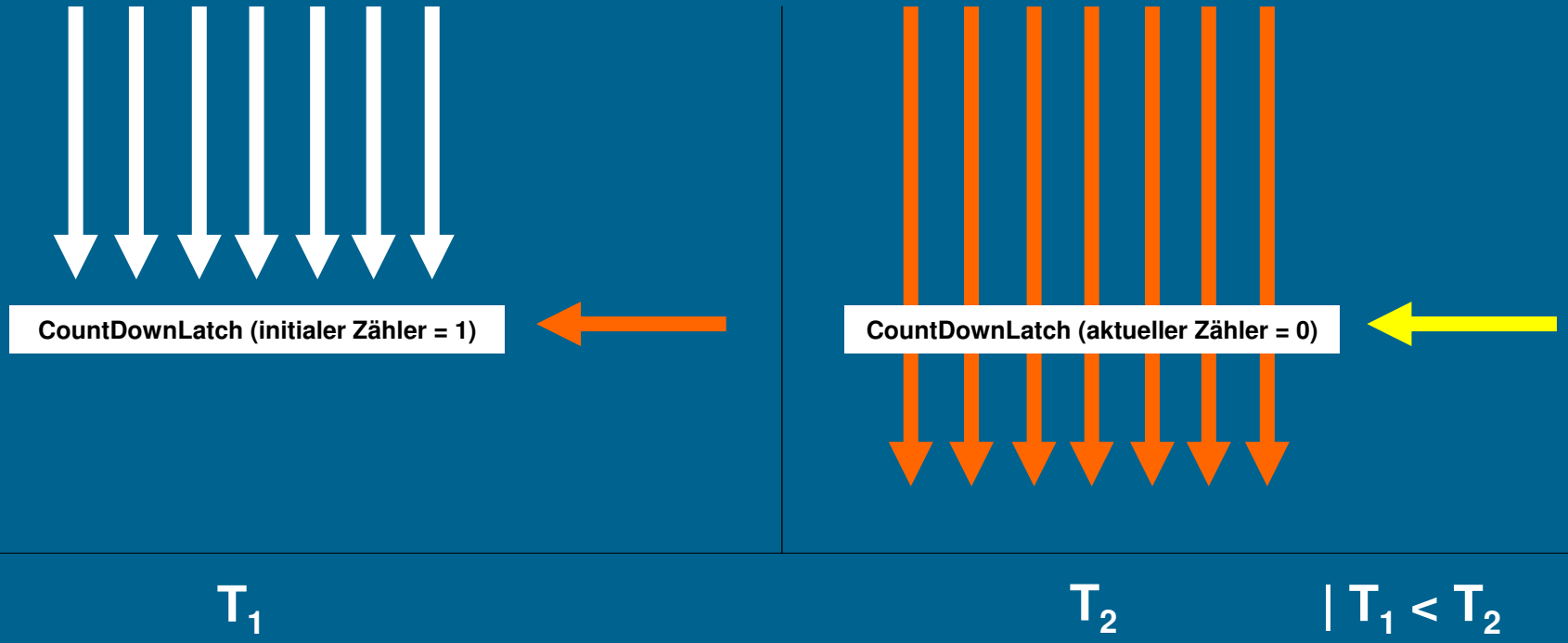


T_1 T_2 $| T_1 < T_2$

wartende Threads aktive Threads abgearbeitete Threads

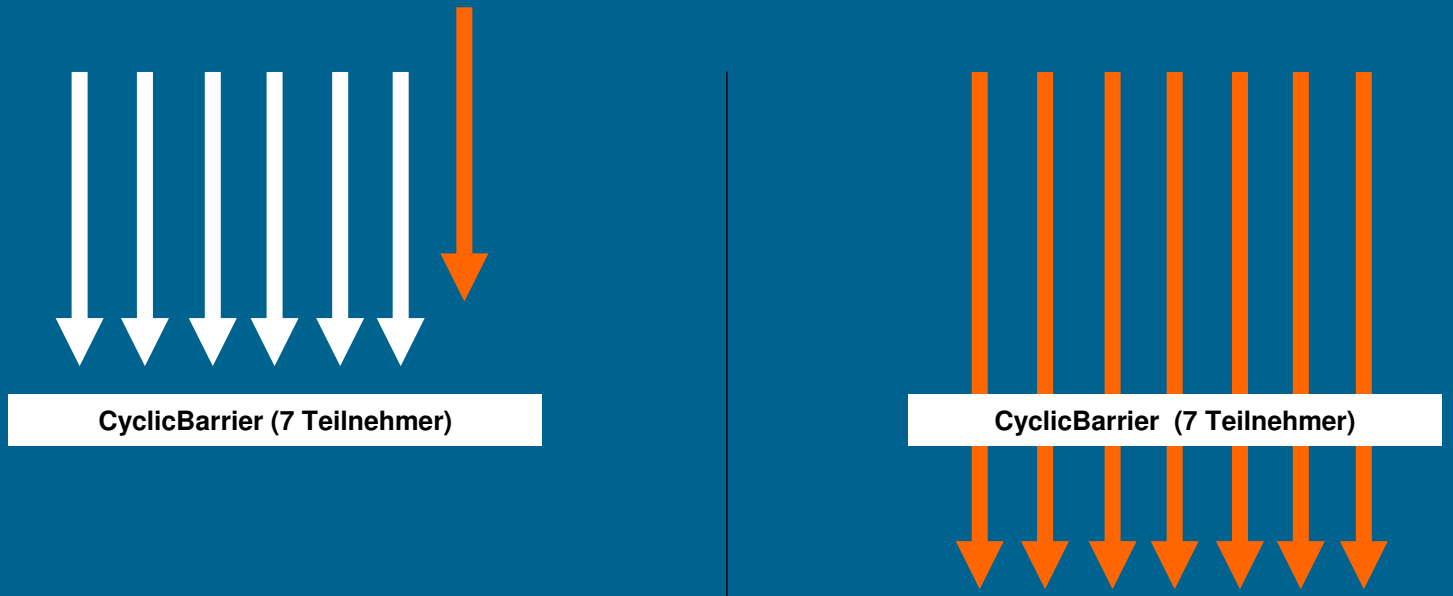
Synchronisation (2/4)

CountDownLatch („ähnlich einem Countdown und Startschuss beim Staffellauf“)



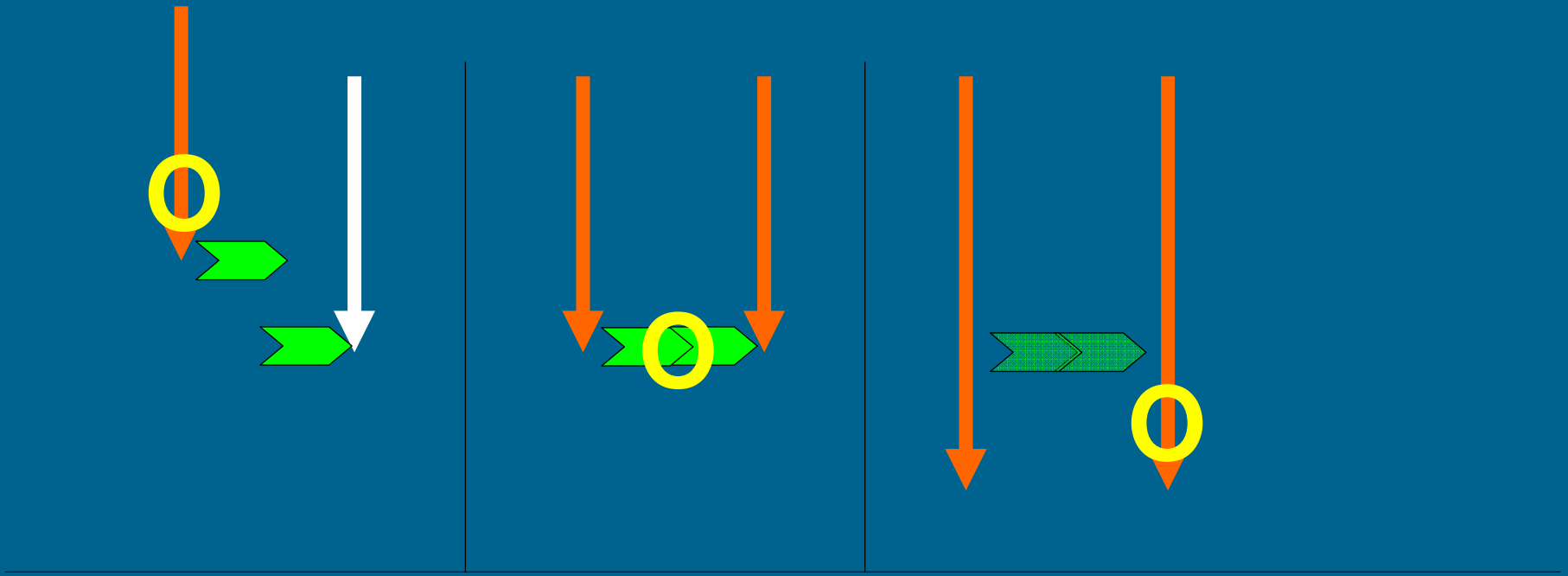
Synchronisation (3/4)

CyclicBarrier („ähnlich dem Treffen am Bahnsteig und gemeinsamen Einsteigen“)



Synchronisation (4/4)

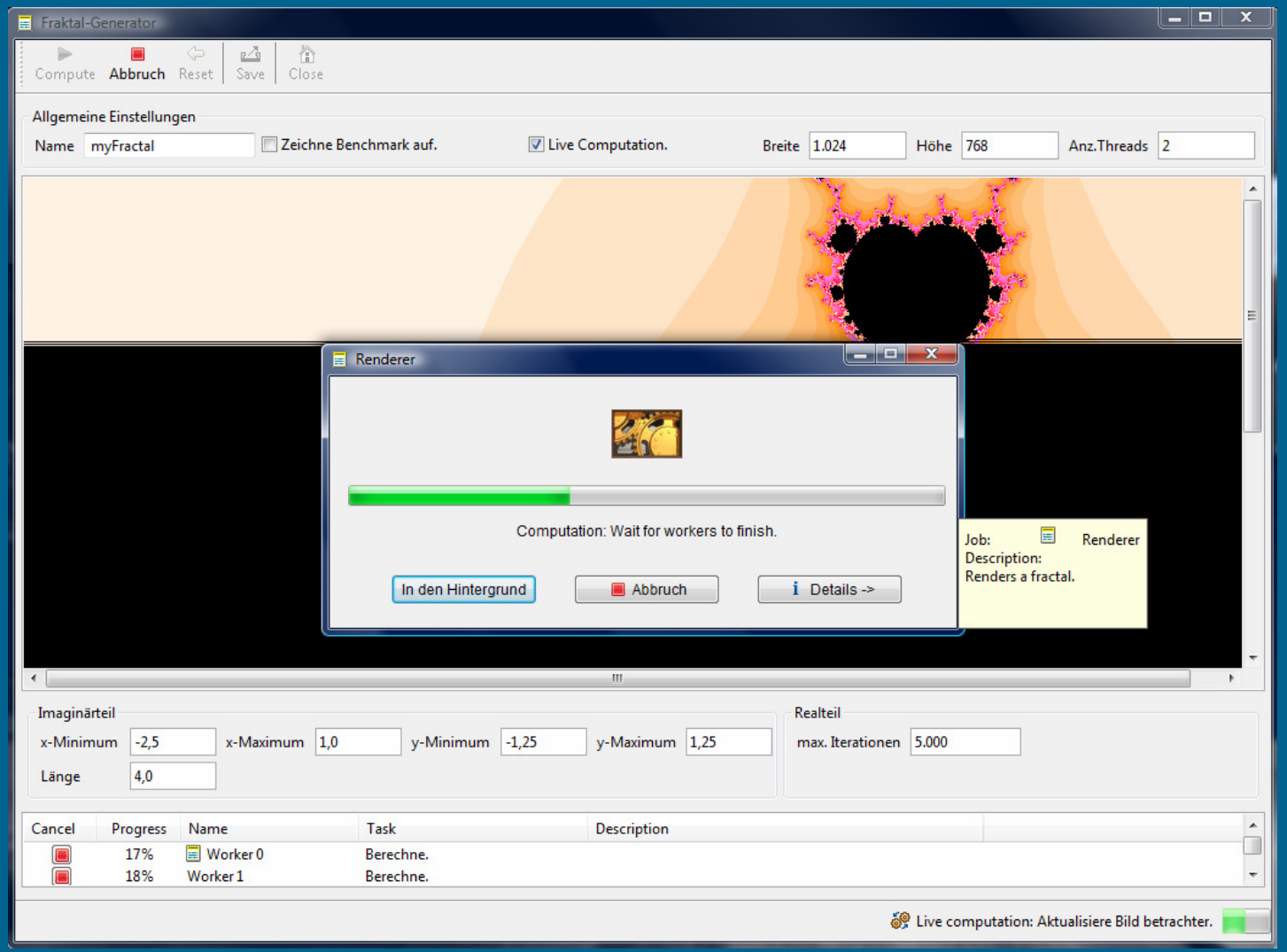
Exchanger („ähnlich der Übergabe des olympischen Feuers zwischen Läufern“)



T_1 T_2 T_3 $| T_1 < T_2 < T_3$

wartende Threads aktive Threads Rendezvous Punkt

Akademisches Beispiel (Grüne Wiese)



Akademisches Beispiel (Grüne Wiese)

Parallele Applikation auf 2 Prozessorkernen:

- Task 1: „UI-Thread“ (P5 - Normal)
- Task 2: „Renderer“ (P3 - Niedrig)
  - verteilt gesamt Problem auf Worker
  - stellt Rendezvous Punkt für Worker bereit
  - vereint und präsentiert Teilergebnisse, nach erreichen des Rendezvous Punktes
- Task 3: „Worker 1“ (P4 - Weniger als Normal)
  - berechnet Teilproblem, **erzeugt CPU-Last**
- Task 4: „Worker 2“ (P4 - Weniger als Normal)
  - berechnet Teilproblem, **erzeugt CPU-Last**
- Task 5: „Graphics Updater“ (P2 – Idle)
  - aktualisiert Hintergrundgrafik alle 5s
- Task 6: „Progress Updater“ (P2 – Idle)
  - aktualisiert Fortschrittsanzeigen aller Tasks (spezifisch für verwendetes Job API)

Topics of the day

Prozessortrends

Parallele Programmierung

Ein Pattern für den Multicore

Checkliste für das Design paralleler Programme

Eine sehr kritische Betrachtung des „MultiCore Hypes“

Fragen und Diskussion

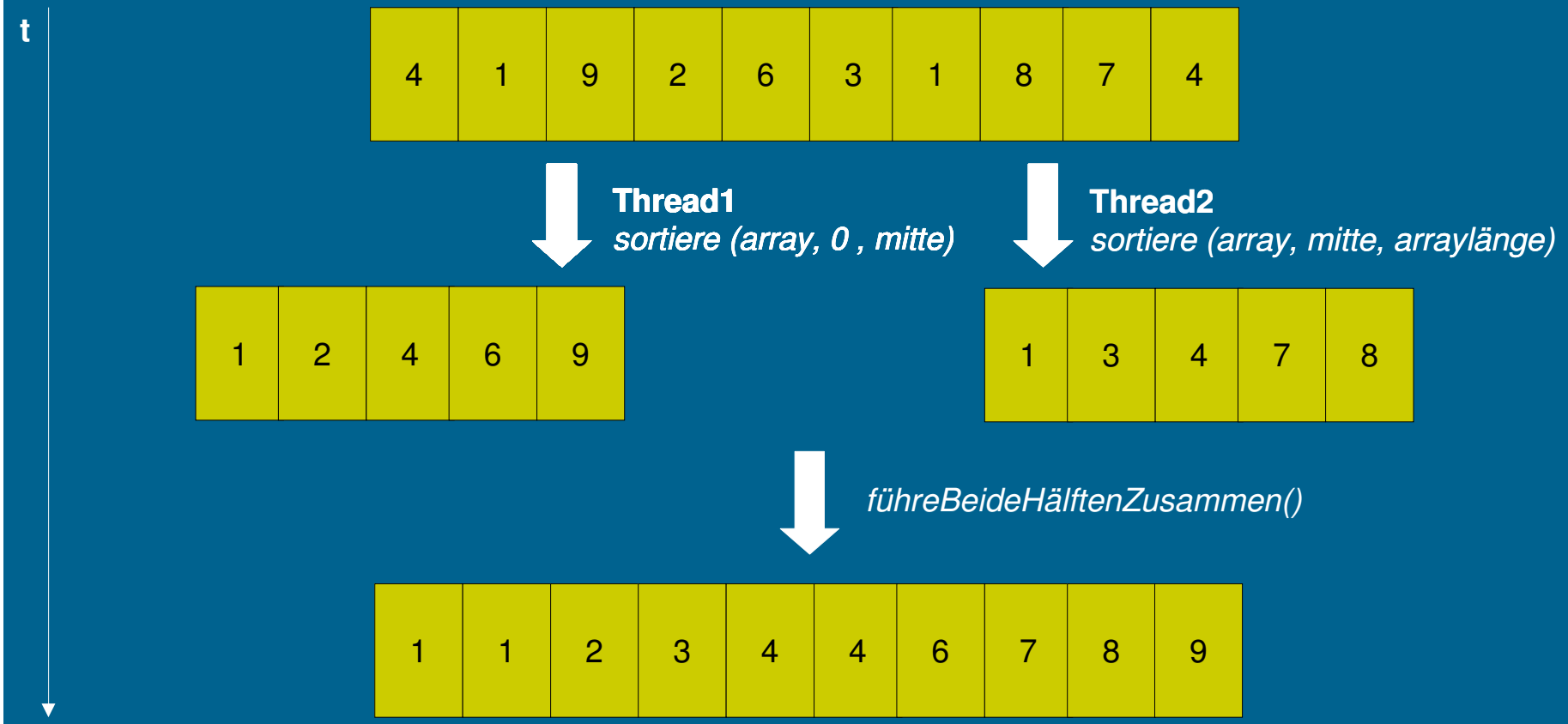
Ein Pattern für den MultiCore - *Theorie*

- hier findet ein Grundprinzip der Informatik seine Anwendung: **divide and conquer**
- durch Zerlegung des Gesamtproblems in kleinere Teilprobleme sinkt der Aufwand zur Lösung des Gesamtproblems
- angewandt auf die **Multicore Welt**: rechenlastige Abschnitte des Programms sind in mehrere rechenlastige **voneinander unabhänigige** Teilabschnitte zu unterteilen. Die Ergebnisse aus den Ausführungen der Teilabschnitte werden dann wieder zu einem Gesamtergebnis zusammengefügt.
- dabei ist jede Berechnung eines Teilabschnitts einer physikalischen oder logischen Prozessoreinheit zuzuteilen

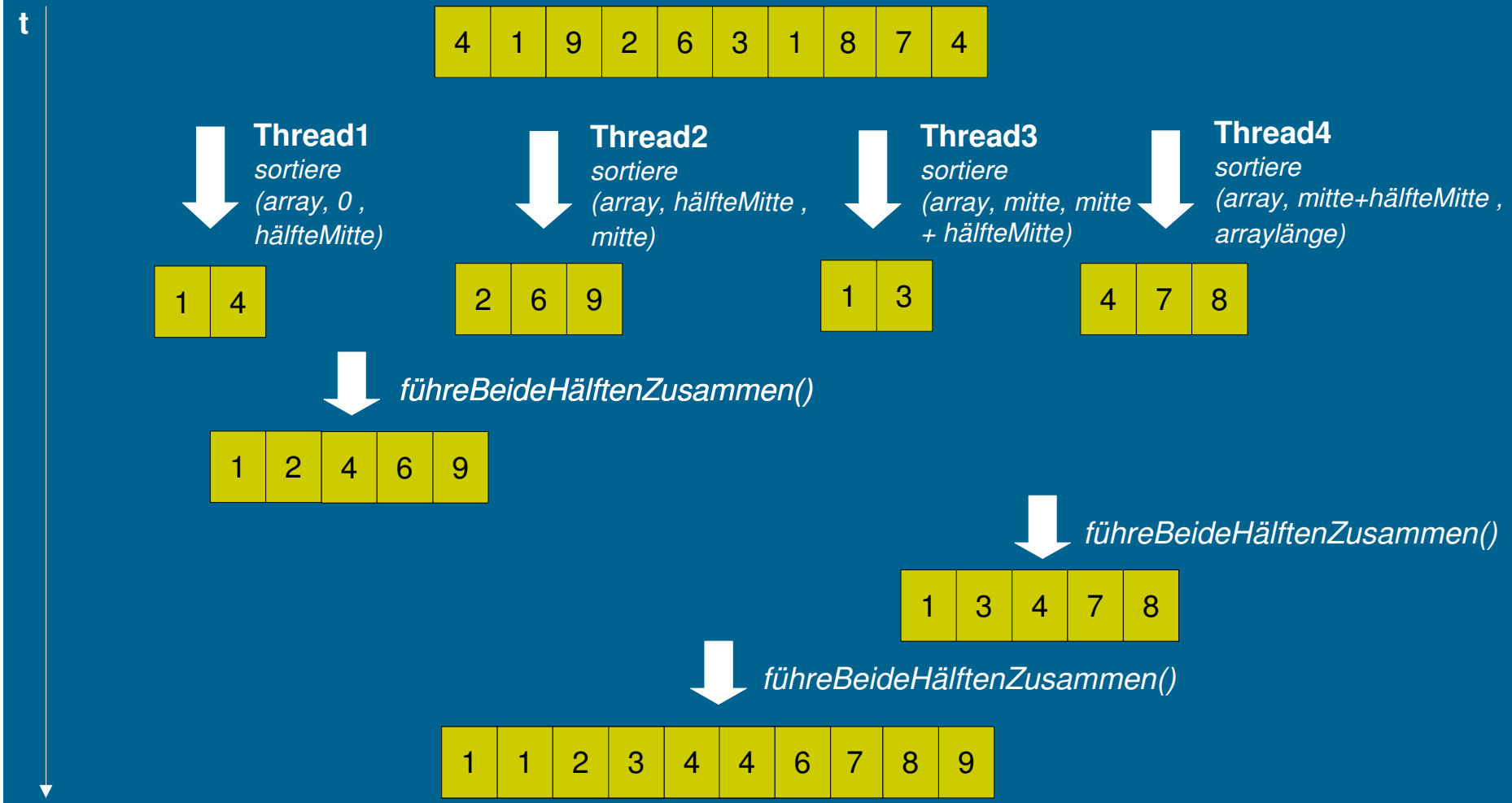
Ein Pattern für den MultiCore - *Anwendung*

- *java.util.Arrays.sort(…)* bietet eine Funktion zum Sortieren von Arrays verschiedener Datentypen
- als Sortieralgorithmus wird der MergeSort verwendet
- die Implementierung wurde von Sun **nicht** für mehrere Prozessorkerne optimiert!
- Aufgabe: Sortierung von n Elementen, die via "*1 + (int) (1000000 * Math.random())* " erzeugt wurden
- Sortierung erfolgt mit der Standardimplementierung als auch mit einer für den DualCore und einer für den QuadCore optimierten Version

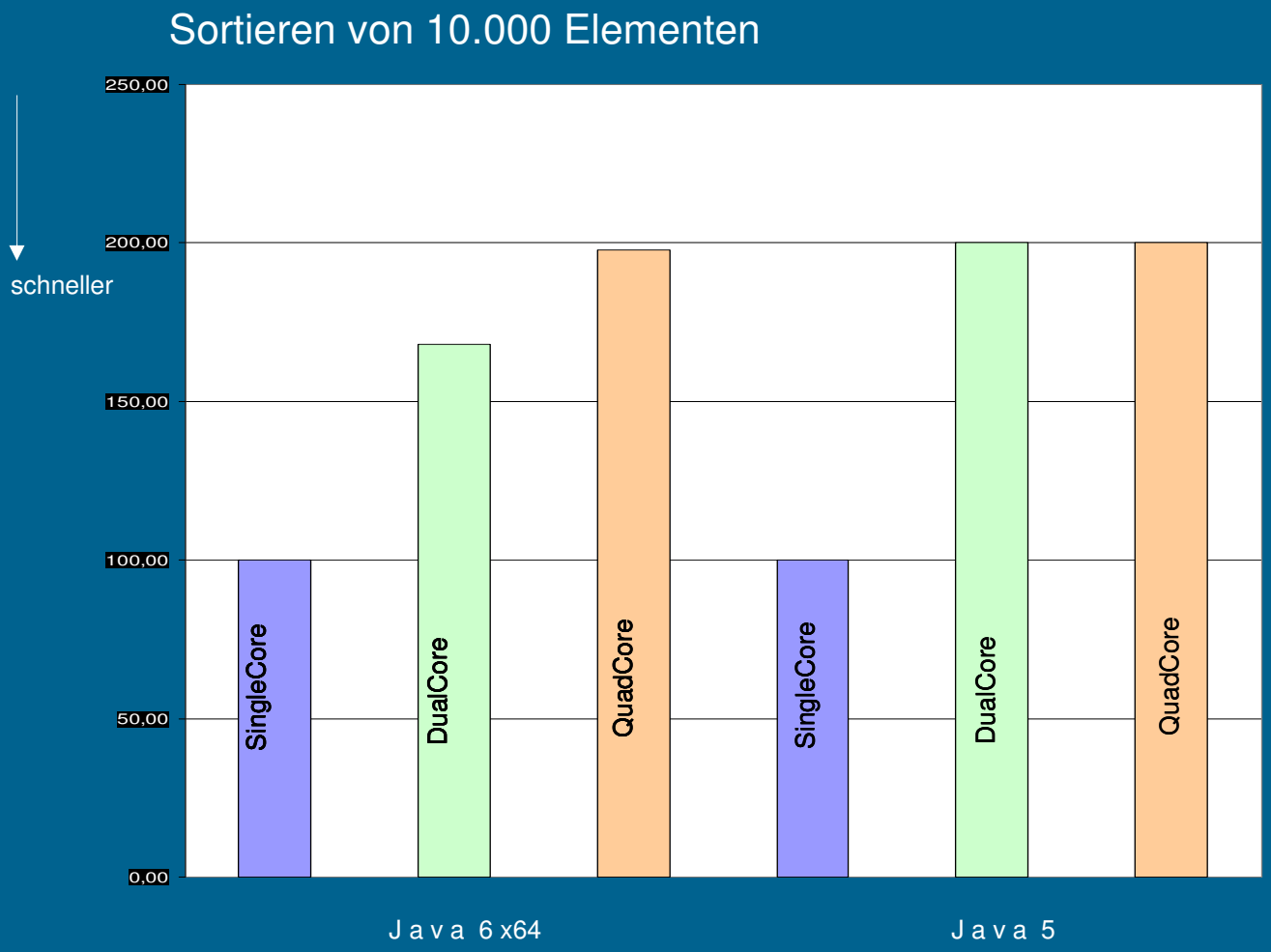
Ein Pattern für den MultiCore – *Anwendung DualCore*



Ein Pattern für den MultiCore – *Anwendung QuadCore*



Ein Pattern für den MultiCore – *Messergebnisse (1/5)*



Test-Setup:

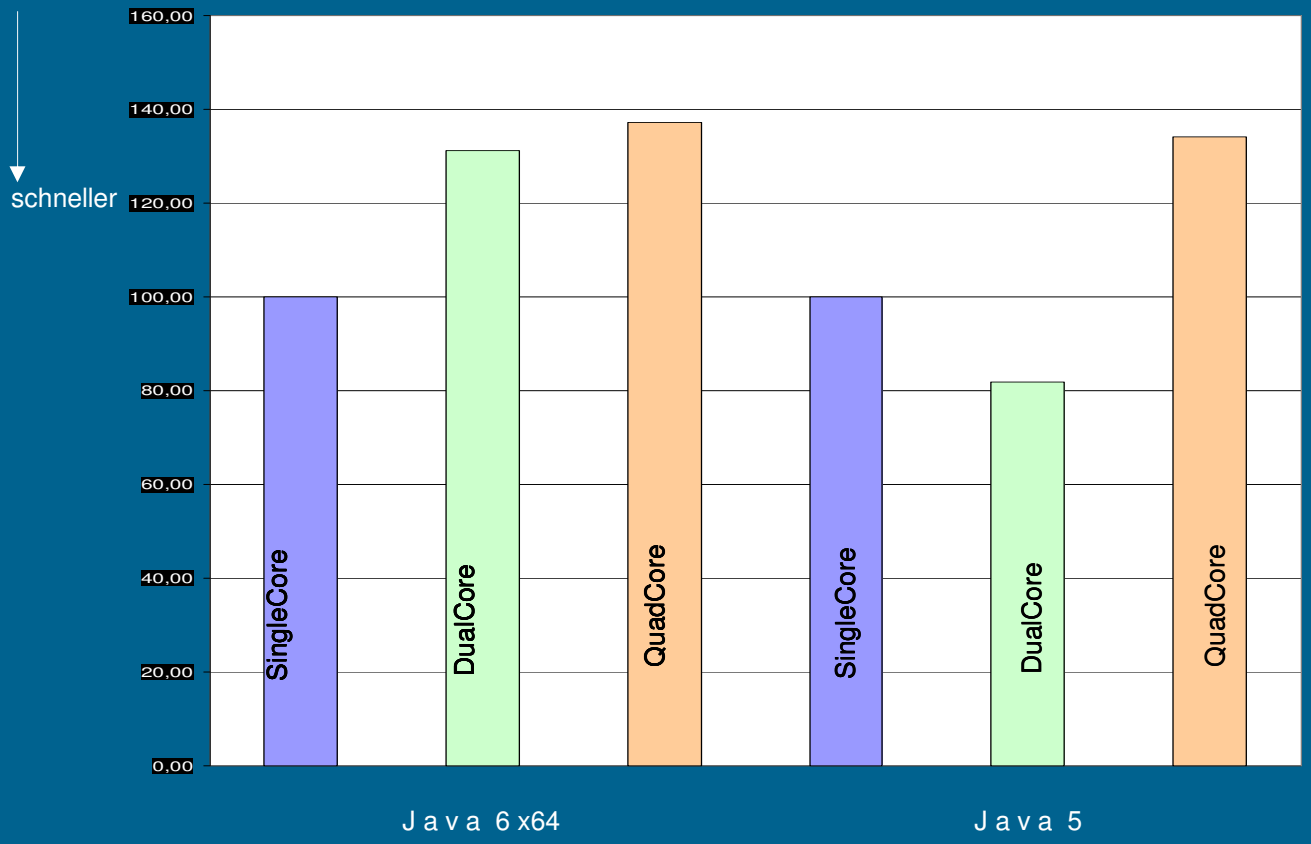
Intel Core2Quad 4 * 2,4 GHz, Windows Vista x64

SUN JDK 1.5.0.15

SUN JDK 1.6.05 x64

Ein Pattern für den MultiCore – *Messergebnisse (2/5)*

Sortieren von 100.000 Elementen



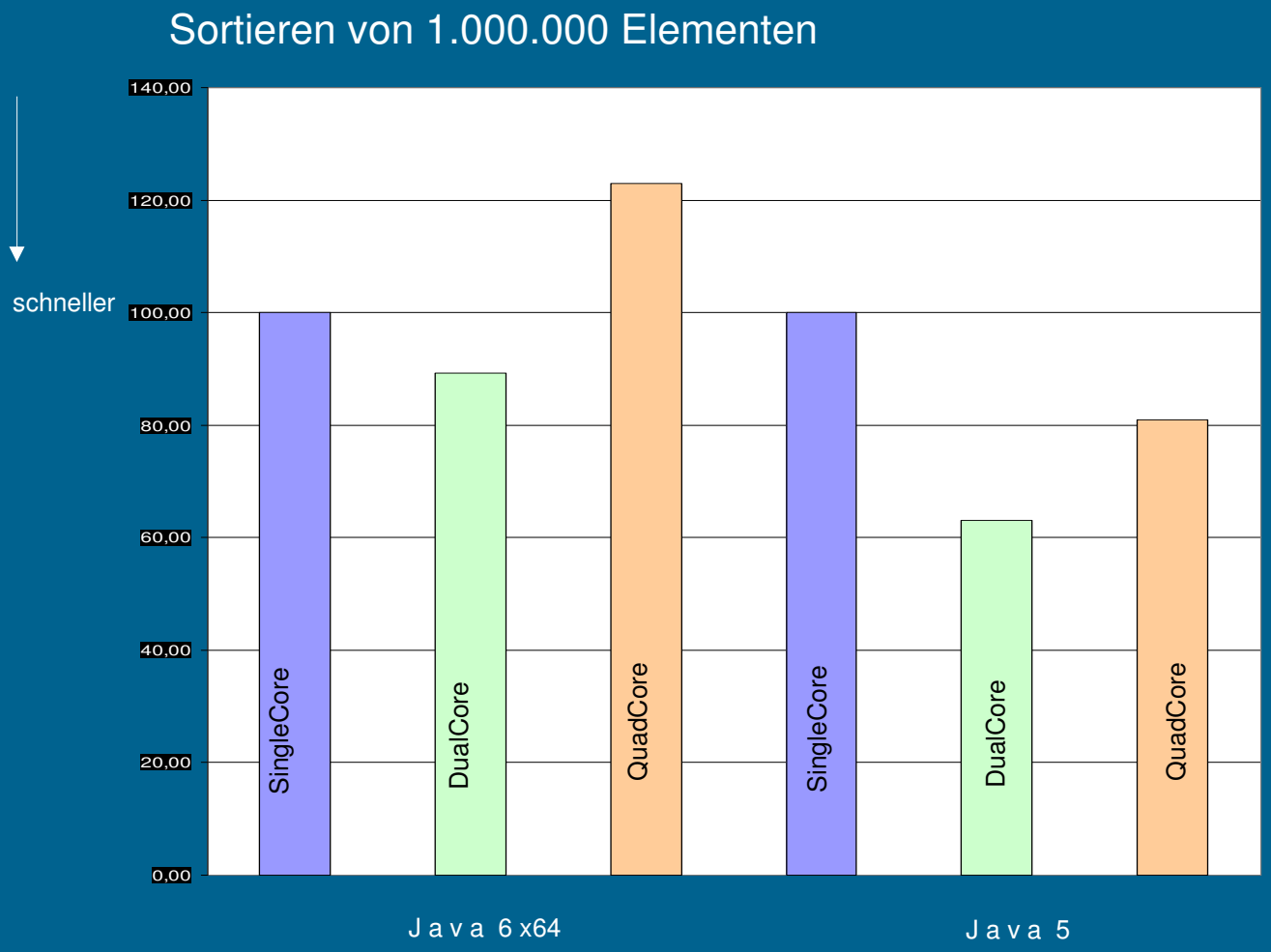
Test-Setup:

Intel Core2Quad 4 * 2,4 GHz, Windows Vista x64

SUN JDK 1.5.0.15

SUN JDK 1.6.05 x64

Ein Pattern für den MultiCore – *Messergebnisse (3/5)*



Test-Setup:

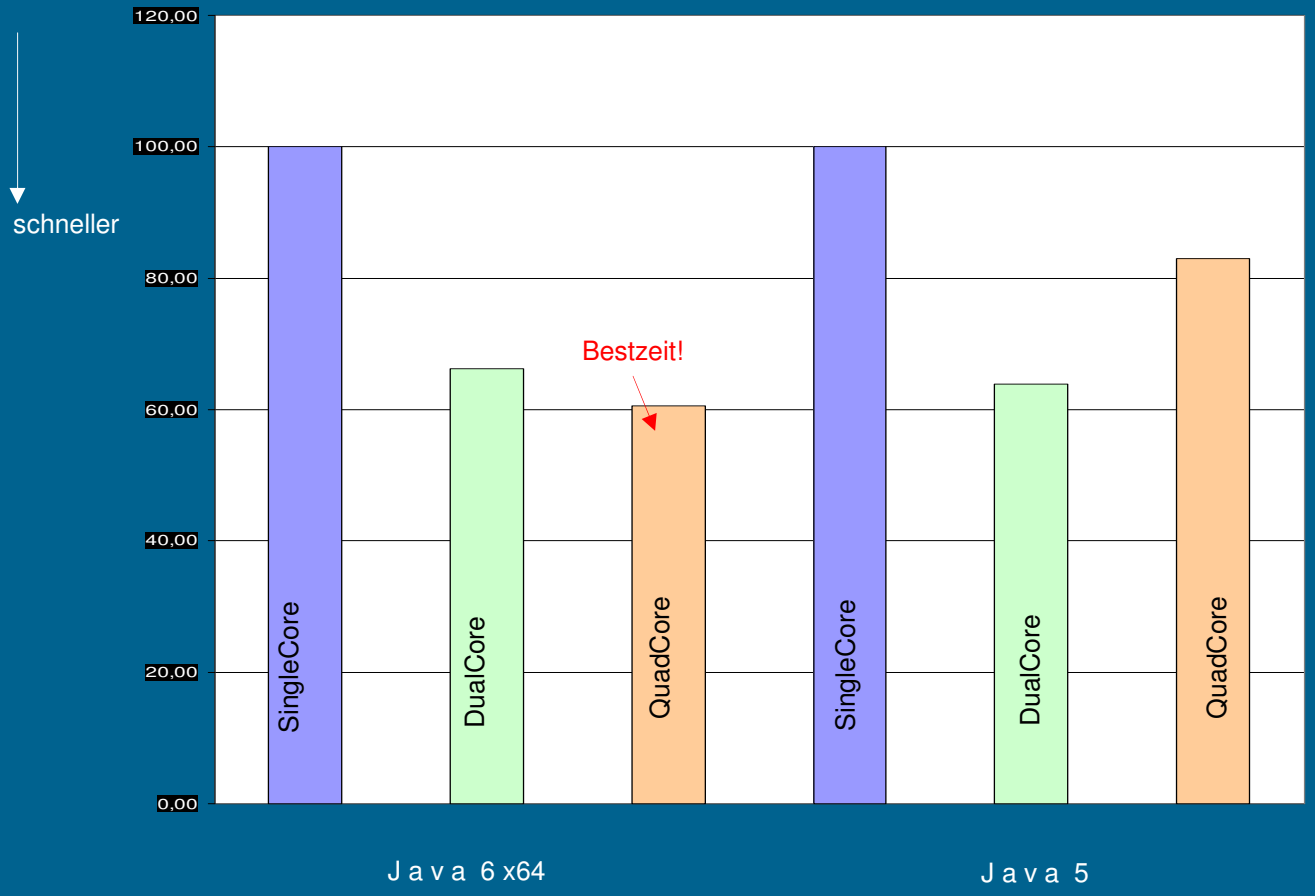
Intel Core2Quad 4 * 2,4 GHz, Windows Vista x64

SUN JDK 1.5.0.15

SUN JDK 1.6.05 x64

Ein Pattern für den MultiCore – *Messergebnisse (4/5)*

Sortieren von 10.000.000 Elementen



Test-Setup:

Intel Core2Quad 4 * 2,4 GHz, Windows Vista x64

SUN JDK 1.5.0.15

SUN JDK 1.6.05 x64

Ein Pattern für den MultiCore – *Messergebnisse (5/5)*

Sortieren von 15.000.000 Elementen

```
Running on Dual Core
Dual Core: Merged sorted parts in 1.139 seconds.
Array sorted in 7.177 seconds.
Verfiying Sorting result...
Everything is sorted properly!
```

Test-Setup:

Intel Core2Quad 4 * 2,4 GHz, Windows Vista x64

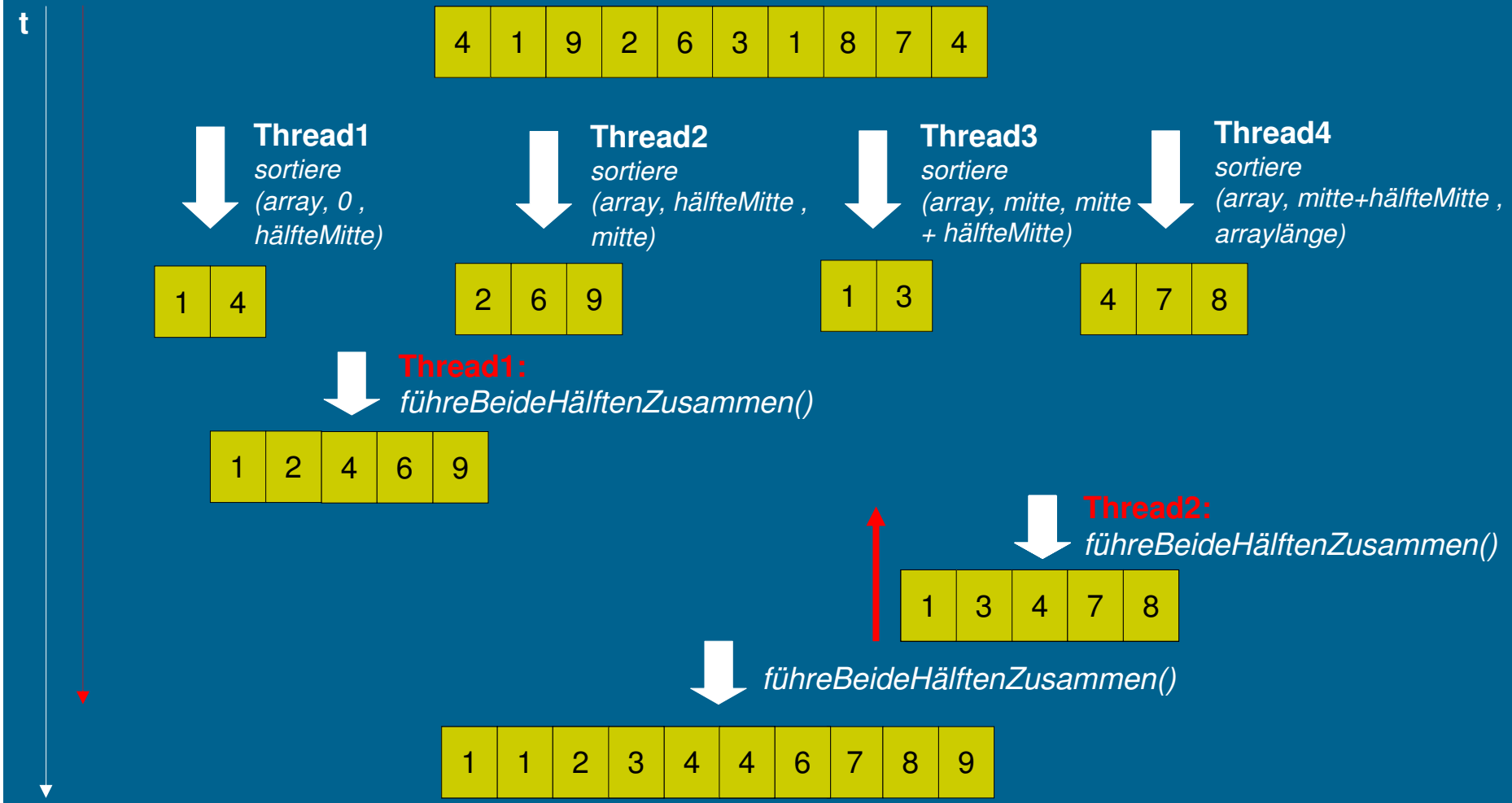
SUN JDK 1.6.05 x64

-Xmx2048m

Aufwand für die Zusammenführung der sortierten Teilarrays (kann noch optimiert werden).

```
Problems | Javadoc | Declaration | Search | Call Hierarchy
<terminated> ArrayExtensionTest [Java Application] D:\development\java\jdk
Running on Quad Core
Quad Core: Merged sorted parts in 2.855 seconds.
Array sorted in 6.444 seconds.
Verfiying Sorting result...
Everything is sorted properly!
```

Ein Pattern für den MultiCore – *Optimierung QuadCore*



Ein Pattern für den MultiCore – *Optimierung QuadCore*

Sortieren von 15.000.000 Elementen

*Optimierung*sansatz:
Teilmerges werden auf dem QuadCore auf zwei Threads aufgeteilt und können so parallel ausgeführt werden.

Test-Setup:

Intel Core2Quad 4 * 2,4 GHz, Windows Vista x64

SUN JDK 1.6.05 x64

-Xmx2048m

× 6 %

Vor Optimierung:

```
Problems | Javadoc | Declaration | Search | Call Hierarchy
<terminated> ArrayExtensionTest [Java Application] D:\development\java\jdk
Running on Quad Core
Quad Core: Merged sorted parts in 2.855 seconds
Array sorted in 6.444 seconds.
Verfiying Sorting result...
Everything is sorted properly!
```

Nach Optimierung:

```
Problems | Javadoc | Declaration | Search | Call Hierarchy
<terminated> ArrayExtensionTest [Java Application] D:\development\java\j
Running on Quad Core
Quad Core: Merged sorted parts in 2.293 seconds.
Array sorted in 6.054 seconds.
Verfiying Sorting result...
Everything is sorted properly!
```

Ein Pattern für den MultiCore – *Fazit* (1/3)

- mit der Ausführung des Algorithmus auf zwei Prozessoren lässt sich die Ausführungszeit um das bis zu 1,5 fache steigern
- die Ausführungszeit variiert je nach verwendeter Java Runtime environment unter Umständen erheblich (ab 100.000 Elementen)
- der QuadCore kann seinen Vorteil erst bei 10.000.000 Elementen unter Java 6 64 Bit ausschöpfen
- hier ist der Vierkern ca. 10% schneller als sein Zweikern Pendant

Ein Pattern für den MultiCore – *Fazit* (2/3)

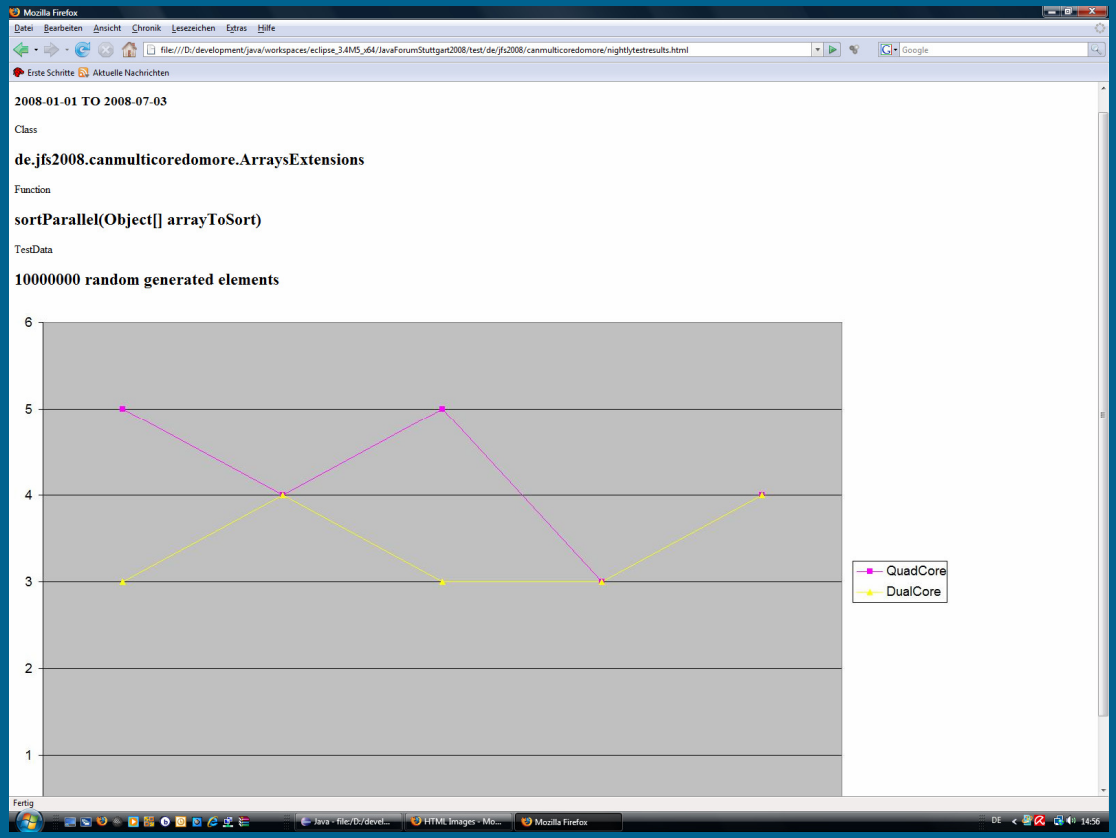
• für sämtliche Source Code Teile, die Sie (z.B. via **Divide and Conquer**) parallelisieren wollen gilt:
Testen, Messen, Testen, Messen … !

• Schreiben Sie Tests für alle unterstützten Prozessorarchitekturen, führen Sie diese im Nightly Test aus und persistieren Sie die Messergebnisse

• Werten Sie die Messergebnisse regelmäßig aus (vielleicht sogar automatisiert) !

Ein Pattern für den MultiCore – *Fazit* (3/3)

Beispielhafte Auswertung von Nightly Tests



Topics of the day

Prozessortrends

Parallele Programmierung

Ein Pattern für den Multicore

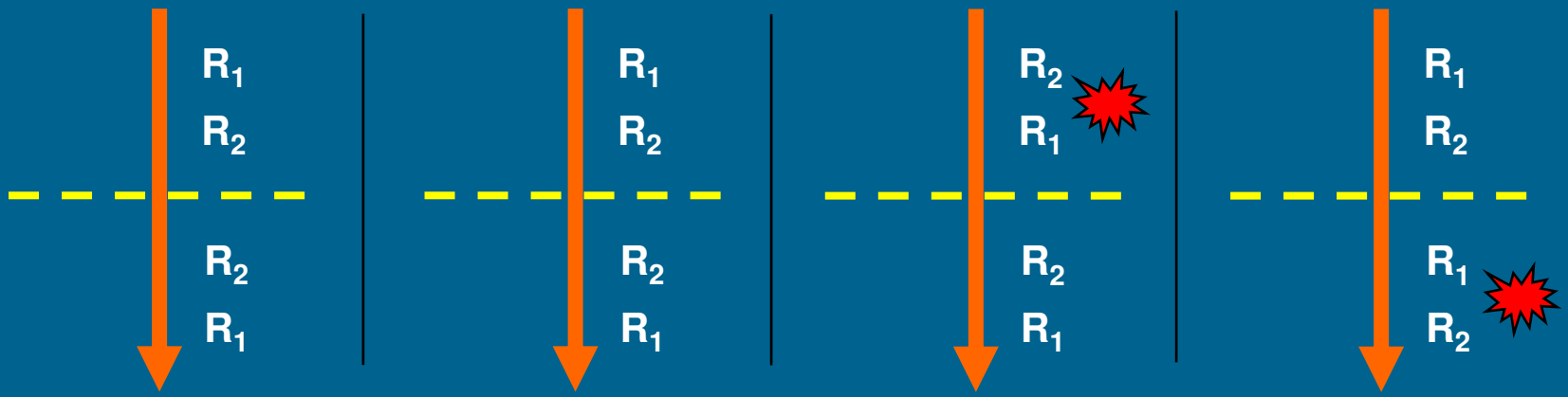
Checkliste für das Design paralleler Programme

Eine sehr kritische Betrachtung des „MultiCore Hypes“

Fragen und Diskussion

Checkliste (1/2)

- kritische Abschnitte (wenige, kurze Laufzeit)
- kurzlebige Prozesse (Sind Threadpools passend konfiguriert?)
  - corePoolSize / maximumPoolSize
  - keepAliveTime
- unveränderliche Objekte („immutable“ statt exklusiven Zugriff)
  - falls Zustand des unv. Obj. geändert werden muss neues Exemplar
- Stimmt Reihenfolge von exklusiven Ressourcen?



Checkliste (2/2)

- Ist der Code auch sequentiell ausführbar? (für Debugging)
  Stellen Sie sicher, dass das Programm auch sequentiell ausgeführt werden kann!
- Könnte es zu Verklemmungen kommen? (z.B.: nach Refaktorierung)
  - vor allem nach manueller Umgestaltung
  - nicht naiv glauben, dass automatische Umgestaltung keine Seiteneffekte hat
- Sind Prioritäten für Prozesse geeignet gewählt?
  „Verhungert“ ein Prozess falls System unter Last steht?
- Bestehen unnötige Abhängigkeiten (Synch.Punkte) zwischen Prozessen?
  z.B.: In IDE alle aufgerufenen Methoden prüfen (Caller / Callee Hierarchie).

Topics of the day

Prozessortrends

Parallele Programmierung

Ein Pattern für den Multicore

Checkliste für das Design paralleler Programme

Eine sehr kritische Betrachtung des „MultiCore Hypes“

Fragen und Diskussion

Paralleles Denken – ein BIOS Update für Ihr Gehirn

Quelle: www.unipublic.unizh.ch

Eine sehr kritische Betrachtung des „MultiCore Hypes“

objectit

- „Although threads seem to be a small step from sequential computation, in fact, they represent a huge step. The discard the most essential and appealing properties of sequential computation: understandability, predictability, and determinism“

[Prof. Edward A.Lee, University of California at Berkeley]

- Können wir wirklich parallel denken ?

- Es mag für das menschliche Gehirn noch machbar sein, sich parallele Ausführungen von Sortieralgorithmen vorzustellen, das parallele Durchdenken der Ausführung eines komplexen Programms (Parallelität über Objektgrenzen hinweg, parallele Rekursionen etc.) fällt ihm allerdings sehr schwer

Eine sehr kritische Betrachtung des „MultiCore Hypes“

- „[…] But the mechanisms (such as semaphores) still require considerable sophistication to use, and very likely will still result in incomprehensible programs with subtle lurking bugs“
[Prof. Edward A.Lee, University of California at Berkeley]

- „ […] suffered only by expert technology providers“

- die Java concurrent API mag sehr umfangreich sein, allerdings ist hier das Problem in ihrer möglichen Kombinatorik selbst begründet

- um sie fehlerfrei Anwendung zu können, sollten wir sehr genau wissen, was wir tun

- oder wir verwenden Alternativen von Leuten, die sich schon sehr lange mit Parallelisierung beschäftigt haben (z.B. *openMP*)

Eine sehr kritische Betrachtung des „MultiCore Hypes“


```
//omp parallel
{
   doWorkParallel();
}
```

Fazit

Can multi-core do more ?

JEIN

Topics of the day

- Prozessortrends
- Parallele Programmierung
- Ein Pattern für den Multicore
- Checkliste für das Design paralleler Programme
- Eine sehr kritische Betrachtung des „MultiCore Hypes“
- **Fragen und Diskussion**

Quellen

Multicore: Parallele Programmierung. Prof. Dr. Thomas Rauber, Prof. Dr. Gudula Rünger. Springer Verlag. 2008. ISBN 978-3-540-73113-9

The Problem with Threads. Prof. Dr. Edward A. Lee. Electrical Engineering and Computer Sciences. University of California at Berkeley

High Performance Java Technology in a Multi-Core World. Java One Conference. 2007
http://developers.sun.com/learning/javaoneonline/2007/pdf/TS-2885.pdf

Multicore processing for client-side applications. Kirill Grouchnikov. JavaWorld.com. 2007
http://www.javaworld.com/javaworld/jw-09-2007/jw-09-multicoreprocessing.html

Programmieren für Multi-Core Prozessoren. Rami Radi. Intel. tecchannel. 2008
http://www.tecchannel.de/webtechnik/entwicklung/1755202/

Die drei wichtigsten Aspekte der Multi-Core-Optimierung (Parallelismus). James Reinders. Chief Evangelist Software Products Division, Intel. ZDnet.
http://www.zdnet.de/specials/whiteboard-series/0,39038336,39157991,00.html

Class Dependency Analyzer. Manfred Duchrow. Caprica Limited.
http://www.dependency-analyzer.org/