Scala

Westfälische
Wilhelms-Universität
Münster

Thomas Kruse

Seminar „Parallele Programmiersprachen“

Logo: © EPFL http://www.scala-lang.org/

Ziele

- Gefühl für Scala vermitteln
- (Einige) wesentliche Merkmale vorstellen
- Beispiele und Syntax
- Bewertung für die Praxis

Foto: Miles Sabin
http://www.scala-lang.org/node/3486

Motivation Parallelisierung

- Geschwindigkeit von CPUs stagniert
- Stattdessen: Parallelität für mehr Performance
  - Mehr Threads pro Core
  - More Cores pro Socket
  - GPU „Cluster als Steckkarte“
- Erreichbar über Threads
  - In fast allen Programmiersprachen verfügbar
  - Abstraktionsniveau vergleichbar mit Assembler
  - Skalierungsprobleme bei vielen Threads

„Programmierung ist schwer....

Aber parallele Programmierung ist tausend mal schwerer...“

Simon Peyton Jones

(1999)

- Wunsch: Auf hohem Niveau Abläufe beschreiben und von automatischer Parallelisierung profitieren
- Map-Reduce
  - Patentiert von Google
  - Apache Hadoop
  - Java: JDK 7 Fork-Join
- Actors
  - Erlang, **Scala**
  - Java: Kilim, Jetlang, ...

Bilder: Projekt Homepages (JDK7, Hadoop, Google), www.oscar.org

- Funktionale Programmiersprachen
  - In der Regel: Ohne Nebenwirkungen
  - Kein globaler bzw. geteilter Zustand
  - Erleichtert Parallelisierung
- Scala ist...
  - Funktional
  - Objektorientiert
  - Statisch typisiert (Typinferenz)
  - Prägnant und leicht erweiterbar
  - Eng mit Java integriert

Bild: http://www.cs.chalmers.se/Cs/Research/Functional/

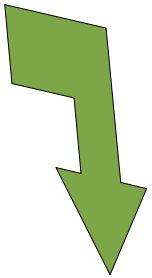
„If I were to pick a language to use today other than Java, it would be Scala.“

James Gosling

(JavaOne 2008, Community Booth)

Weniger ist mehr


```
public class Person {
  private int age; private String name;

  public Person(int age, String name) {
    this.age = age; this.name = name;  }

  public int getAge() { return this.age;   }

  public void setAge(int age) { this.age = age;   }

  public String getName() { return this.name;   }

  public void setName(String name) { this.name = name;  } }
```



```
class Person(var age: Int, var name: String)
```

Scala vs. Java

Hinzugefügt	Entfernt
+ Operator Überladung	- Statische Felder
+ Objektsystem	- primitive Typen
+ Closures	- break, continue
+ Mixin Konzept mit Traits	- Interface Typ
+ Pattern Matching	- Wildcards
+ Abtrakte Typen	- Raw Types
+ Operator Überladung	- Enums

- Bibliothek: assert, enums, Properties, Actors...

- Typdeklaration

```
i: int

s: String
```

- Konstanten und Variablen

```
val c = „Hello World“

var i = 0
```

- Methoden

```
def add(x: Int, y: Int): Int = { x + y }

override def toString = ...
```

- Konstruktor: Klassen Rumpf

```
class Person(val name: String) { … }
```

- Singleton

```
object App{ def main(args: Array[String]){...} }
```

- Methoden Aufruf

```
myObject.doSomething(567)

myObject doSomething(567)

myObject doSomething 567
```

- Rückgabewert: Wert des letzten Ausdrucks

First-Class-Functions


```
// Funktion an die Variable even zugewiesen
val even = (i: Int) => i % 2 == 0
// einfache Liste
val numbers = List(1,2,3,4,5,6)
// filter  (p : (A) => Boolean) : List[A]
numbers.filter(even) // 2,4,6
// _ ist Wildcard: hier anonyme implizite Funktion
numbers.filter(_ > 2) // 3,4,5,6
```

- For Schleife

```
for(s <- args) println(s)

for(i <- 0 to 3) println(i)
```

- if, else, while
- Exceptions

```
try{} catch {
    case e: OilSpillException => …
    case e: StockTooLowException => … }
finally {}
```

Case Classes & Actors


```
abstract case class Command{
   def execute;
}

object DemoActor extends Actor {
    def act() = while(true) { receive {
         case c: Command => c.execute
         case "exit" => println("Exiting..."); exit
         case x: Any => println("Unkn message:" + x) } } }

    def main(args: Array[String]): Unit = {
      val c = new Command{ def execute = {println("Hello
World")}}

     DemoActor.start()
     DemoActor ! new Command{ def execute = { println ("Hi")}}
     DemoActor ! c
     DemoActor ! "exit" } }
```

- Funktionale Programmierung
  - Komposition komplexer Systeme aus einfachen Bausteinen
  - Funktionen höherer Ordnung, Pattern Matching
- Objektorientierte Programmierung
  - Vorhandene Systeme können integriert und erweitert werden
  - Vererbung, Klassen als Abstraktion
- Enge Integration
  - Objektfunktionale Programmierung, Java
- Transparente Verteilung

Erweiterbarkeit

- Scala ist auf Erweiterbarkeit ausgelegt
- Traits (Mixins)
- „Domain specific Languages“ (DSL)
- Operatoren Überladung
  - Gilt für jeden Operator (auch „+“, „:“, „-“)
- Alles wird über Funktionsaufrufe abgebildet
- Beispiele
  - Actors, Scala Test

```
map should contain key ('a')
```

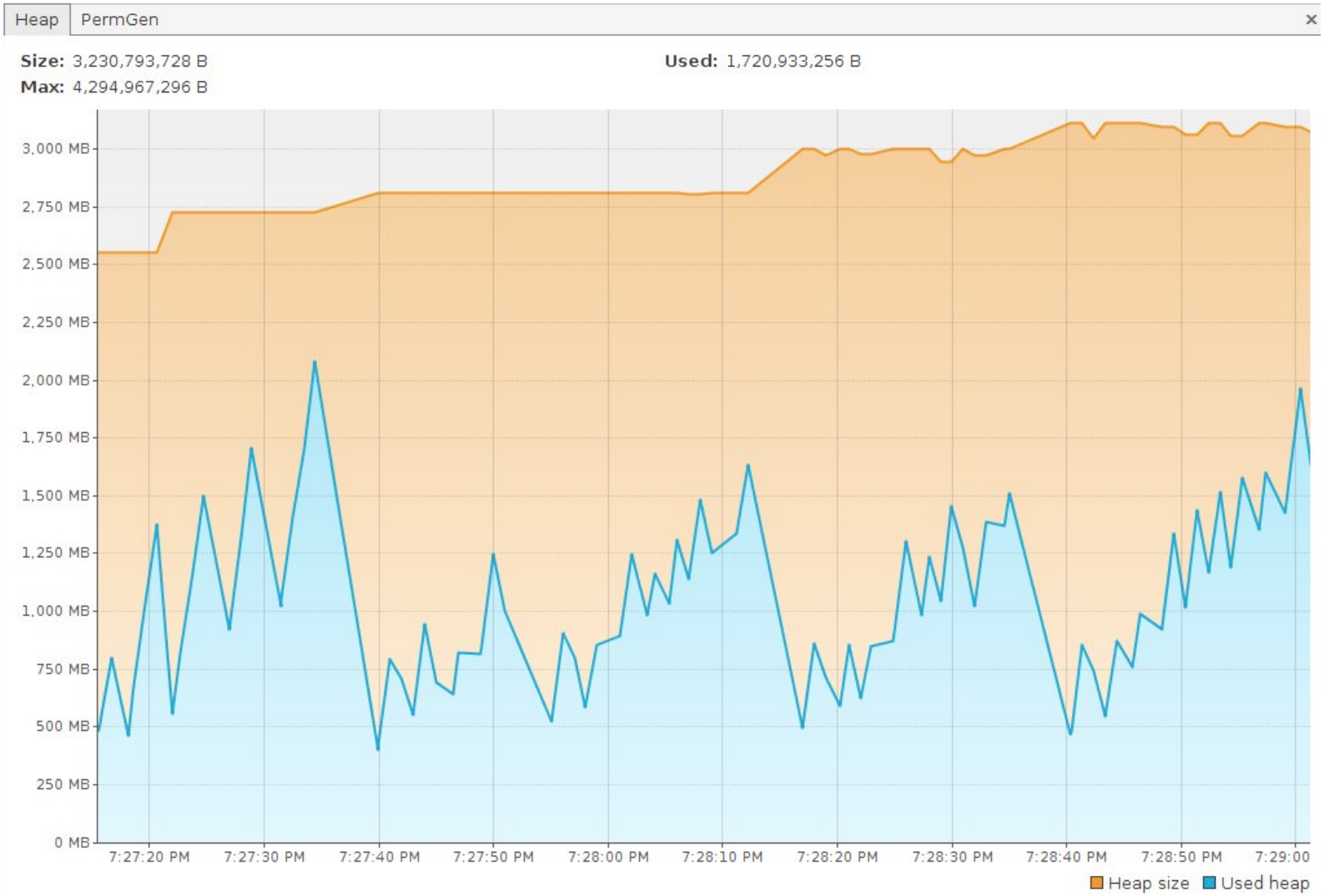
Praxis: Paralleler Quicksort

- Sortiert mittels Actors (ereignisorientierte Implementierung)
  - Trivial zu parallelisieren
  - „shared nothing“
- 1 Mio Zufallszahlen sortieren auf zwei-Kern AMD
  - Laufzeit parallel: **59s** (53,2s – 64,5s) (~10s/500k, ~4.5s/250k)
  - Laufzeit sequentiell: **0.1s** (0.08s – 0.273s)
- Unerwartetes Ergebnis!

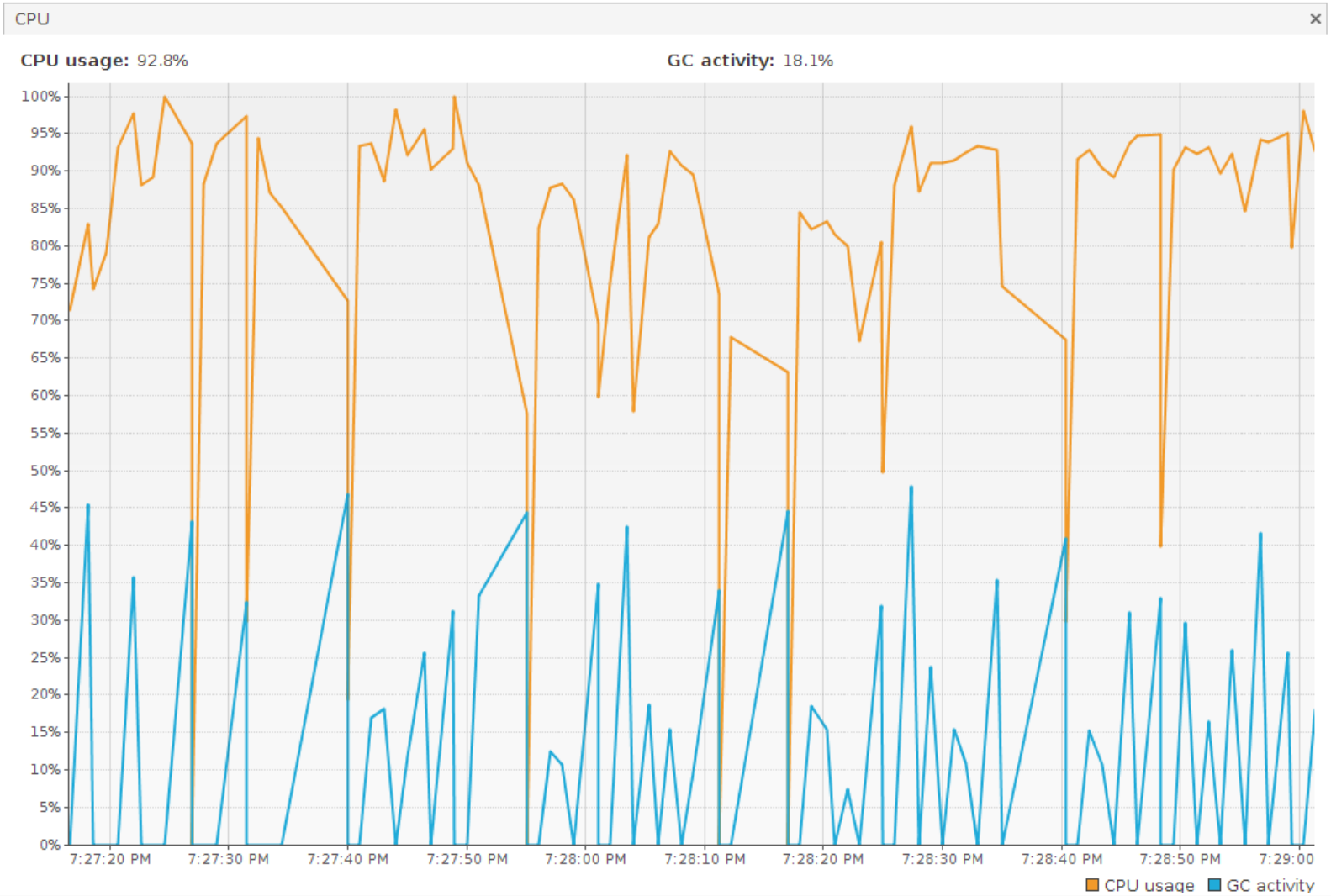
Umsetzung: Quicksort mit Scala Actors


```
case class ComputeQuicksort(a: List[Int], replyTo: Actor)
// genauso: LeftComputerQuicksort, RightComputeQuicksort
// und Result, RightResult, LeftResult

// ComputerQuicksort wird ebenfalls von Actor aufgerufen:
 case ComputeQuicksort(n, replyTo) => {
 if (n.length < 2) replyTo ! Result(n)
 else {
  val pivot = n(n.length / 2)
  (new Psort) ! LeftComputeQuicksort(n.filter(_ < pivot),this)
  (new Psort) ! RightComputeQuicksort(n.filter(_ > pivot),this)
   react {
     case LeftResult(r1) => react {
       case RightResult(r2) =>
         replyTo ! Result(r1 ::: n.filter(_ == pivot) ::: r2)
     }
   }
 }
```

Speicher Nutzung



CPU Nutzung



- Optimierung der Implementierung
  - Algorithmus Umsetzung
  - Statt „shared nothing“ auf gemeinsamem Array arbeiten
  - Durch disjunkte Intervalle kein Konkurrenz-Problem
  - Aber: Nicht mehr transparent verteilbar (Remote-Actor)
- Plattform und Laufzeitumgebung / GC
  - Java SE 6 Virtual Machine Garbage Collection Tuning

  http://java.sun.com/javase/technologies/hotspot/gc/gc_tuning_6.html
- **No free lunch.**

Diskussion

- Quicksort
  - Synthetischer Micro-Benchmark, nicht unbedingt repräsentativ
- Skalierung anhand anderer Beispiele
  - Chat-System (Ausarbeitung), Twitter, Foursquare, …
  - Daten sind oft bereits partitioniert! (Anwender Sitzungen)
- Performance Verständnis
  - Vergleichsweise geringe Performance kann durch gute Verteilung kompensiert werden
- Entwickler-Zeit ist wertvoller als CPU Zeit

- Sehr kurzer Einblick in Scala
- Theorie und Praxis
- … Scala bietet vieles mehr!
  - Traits
  - XML Unterstützung
  - Collections
  - Pattern-Matching
  - Frameworks (Lift)
- … und noch mehr: Zugang zu Java Bibliotheken

Fragen?

Danke für Ihre Aufmerksamkeit

Folien:

Auf Seminar-Homepage und http://everflux.de/

(Ausarbeitung auf Anfrage als PDF)

Kontakt für weitergehende Fragen:

Thomas Kruse – tkruse#sforce.org

Interesse an Scala/Java? http://www.jug-muenster.de/ – Java Usergroup Muenster

Links

- MapReduce Patent http://www.google.com/patents?vid=USPAT7650331
- Scala Homepage http://www.scala-lang.org/
- Lift: http://www.liftweb.net/
- VisualVM https://visualvm.dev.java.net/
- Bildnachweis: Wenn nicht anders angegeben: Eigene Bilder

Reserve Folien

Backup Slides

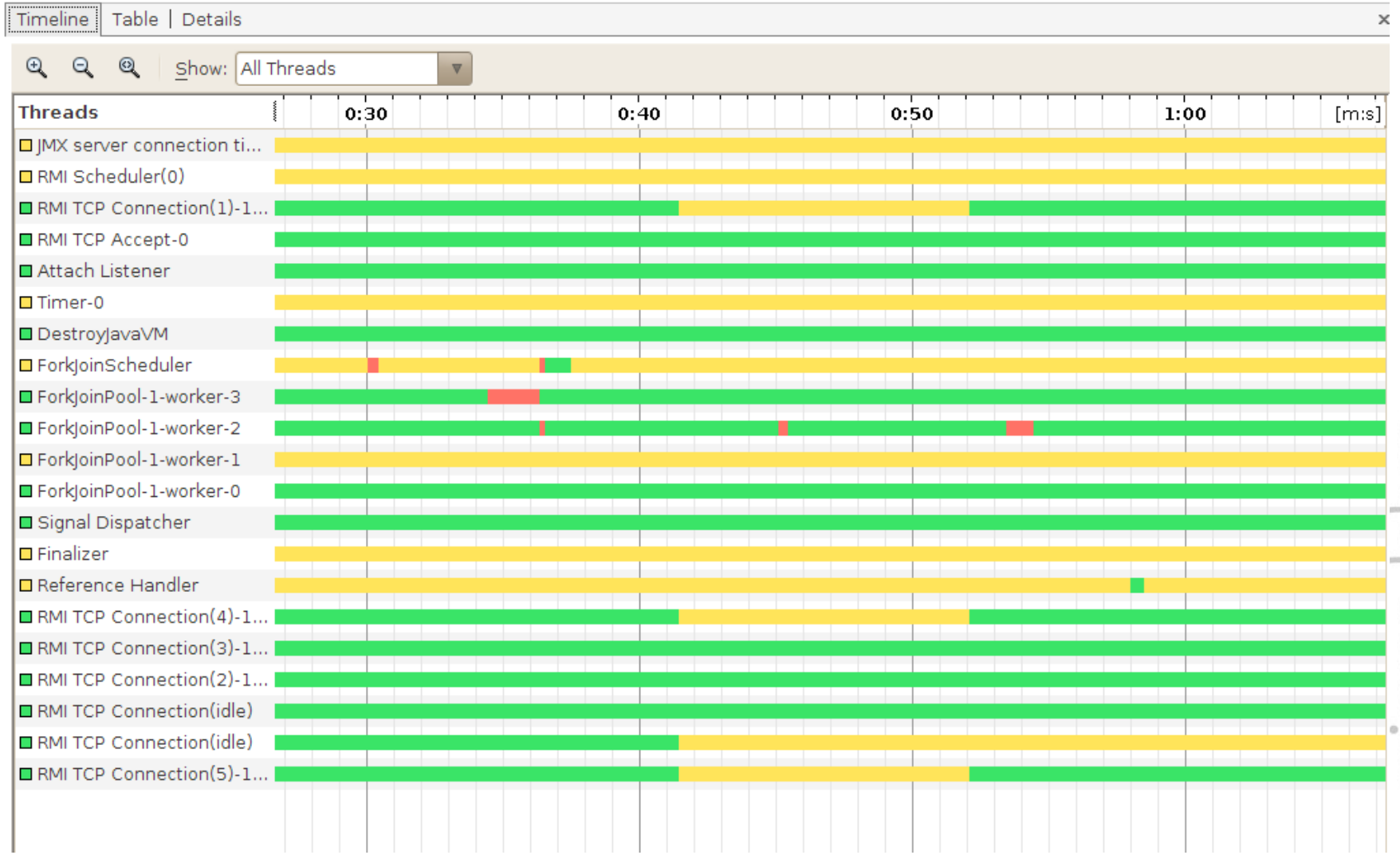
Operator-Überladung

- Jeder Operator kann überladen werden

```
val numbers = 1 :: 2 :: 3 :: 5 :: 8 :: Nil

class Complex(val a: Double, val b: Double){

def +(that: Complex) = new

Complex(this.a+that.a,this.b+that.b); // c = a + b !
```

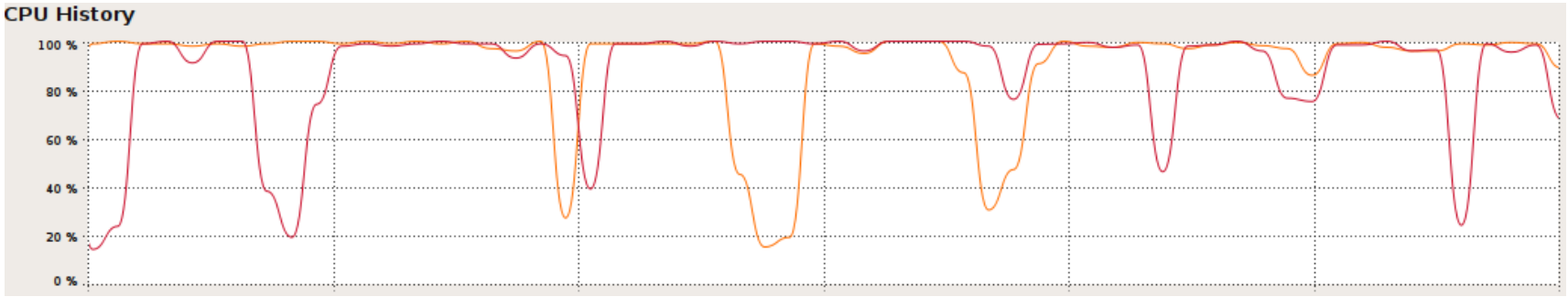
- Eine Grundlage für Erweiterbarkeit von Scala
  - → Domain specific languages (DSL)

Monitor Belegung



Auslastung CPU Kerne

- Garbage-Collector nicht parallel
- → Amdahl's Gesetz
- Parallelisierung trivial – Umsetzung nicht
- Anwendungsspezifisch



Sichtbarkeit

- „public“ ist Standard
- „protected“ erlaubt nur Unterklassen im selben Paket Zugriff
- „private“ wie gewohnt
- Qualifizierte Sichtbarkeit möglich
  - private[com.sample] {...} – bis zum Paket „com.sample“ ist der Bereich sichtbar
  - [_root_] bezeichnet dabei oberste Wurzel
- Mehrere Pakete innerhalb einer Datei zulässig

- Interface / Mixin: Verhalten

```
class Person(age: Int) {
  override def toString = age + „ years old“ }
trait AgeTrait {
  override def toString = "I am really " +
    super.toString }

val person = new Person(16) with AgeTrait
println(person) // => I am really 16 years old
```

- Import von Namensräumen auch in Blöcken möglich
- Import von Typen und Feldern
  - `import BigInteger.ZERO`
- Wildcard Import
  - `import BigInteger._`

- Scala 2.8:
- Continuations (vgl. Coroutinen in Simula)
- Default Parameter
- Benannte Parameter
- Bessere Unterstützung für Werkzeuge
  - Eclipse / IntelliJ / Netbeans Plugin

Parallelisierung vs. Verteilung

- Parallelisierung
  - Lokal
  - Multi Core/CPU
- Verteilung
  - „scale out“
  - Message Passing
  - Clustering
  - Cloud Computing

There is no problem in computer science that cannot be solved by an extra level of indirection.

(unbekannt)

Any performance problem
can be solved by removing a
level of indirection

David Conrad

(ICANN, 2006)

http://www.iab.org/about/workshops/routingandaddressing/indirection.pdf