DSL Embedding in Scala

Tiark Rompf

PURDUE UNIVERSITY

DSL Summer School, July 2015

Scala

IBM

twitter™

LinkedIn

SONY
make.believe

SONY PICTURES imageworks

SIEMENS

xerox

Novell.

outside.in

eDF

NASA

foursquare™

TOMTOM

amazon

UBS

Office DEPOT
Taking Care of Business

guardian.co.uk

OPOWER
ENERGY EFFICIENCY. DELIVERED

Bank of America

yammer

Thatcham

Autodesk

CREDIT SUISSE

Scala = scalable language

Small Scripts -> Large Systems

gradual, evolutionary

FP + OOP

```
class Matrix[T:Numeric](val rows: Int, val cols: Int) {
  private val arr: Array[T] = new Array[T](rows * cols)

  def apply(i: Int, j: Int): T = arr(i*cols + j)

  def *(that: Matrix[T]) = { val res = new Matrix[T]; ...; res }
  def +(that: Matrix[T]) = ...
}
```

```
// a,b,c,d: Matrix[Double]
val x = a*b + a*c
val y = a*c + a*d
println(x+y)
```

Operator overloading, higher-order functions, by-name parameters, implicits, traits, …

What about performance?

"The compiler / JVM will make it run fast"

(wishful thinking)

10x – 1000x slowdown

(hard reality)

InfoQ

En | 中文 | 日本語 | Br

562,907 Mar unique visitors

Development | Architecture & Design | Process & Practices | Operations & Infrastructure

Mobile | HTML5 | JavaScript | Agile Techniques | Cloud Security | SOA | Agile | ALM

News | My Bookmarks | Contribute an Article | print

Yammer Moving from Scala to Java

Posted by Alex Blewitt on Nov 30, 2011

Sections Development Topics

Programming , Social Networki

Share

An e-mail, sent from Yan
Typesafe, ended up being
that Yammer is moving it
with complexity and perf

Yammer PR Shelley Risk c
of Coda Hale, rather than
original author has been
Coda clarified that the m
(CEO of Typesafe) followi

Update: Code has published Yammer's official position on the subject, which confirms the

> "Via profiling and examining the bytecode
> we managed to get a 100x improvement
> by adopting some simple rules:
>
> Don't ever use a for-loop
> Don't ever use scala.collection.mutable
> Don't ever use scala.collection.immutable
> Always use private[this]
> Avoid closures"

Slow Program -> Fast Program?

Much harder!

"Any problem in computer science can be solved by another level of indirection"

-- David Wheeler

"Any problem in computer science can be solved by another level of indirection -- except problems caused by too many levels of indirection"

Abstraction Penalty

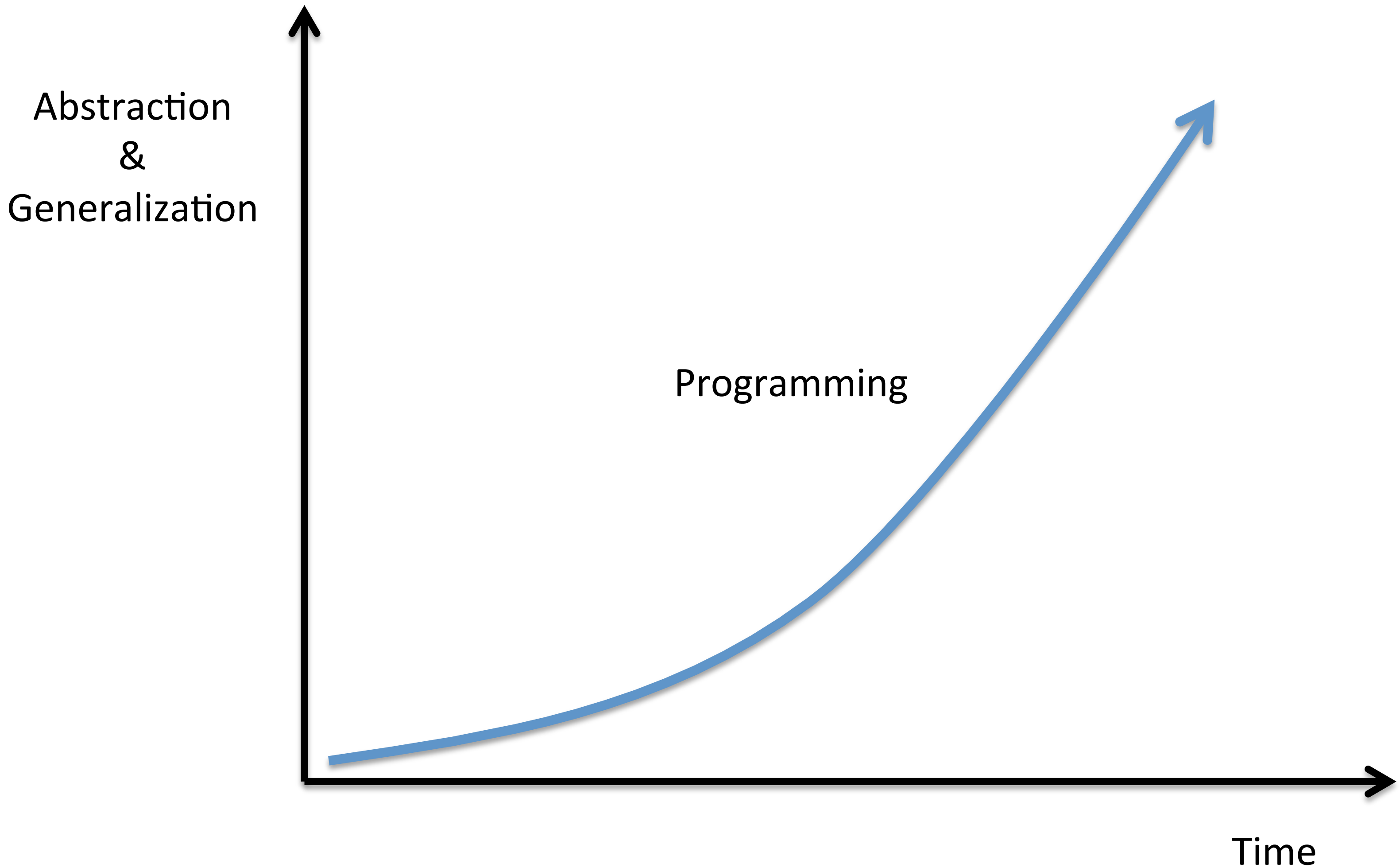
```
class Matrix[T:Numeric:Manifest](                , val cols: Int) {
  private val arr: Array[T] = new Array[T](rows * cols)
  private val num = implicitly[Numeric[T]]; import num._

  def apply(i: Int, j: I
    arr(i*cols + j)

  def update(i: Int, j: Int, e: A): Unit =
    arr(i*cols + j) = e

  def *(that: Matrix[T]) = {
    val res = new Matrix[T](this.r

    for (i <- 0 until this.r
      for (j <- 0 until that.cols) {
        for (k <- 0 until this.rows)
          res(i, j) += this(i, k) * that(k, j)
      }
    }
    res
  }
}
```

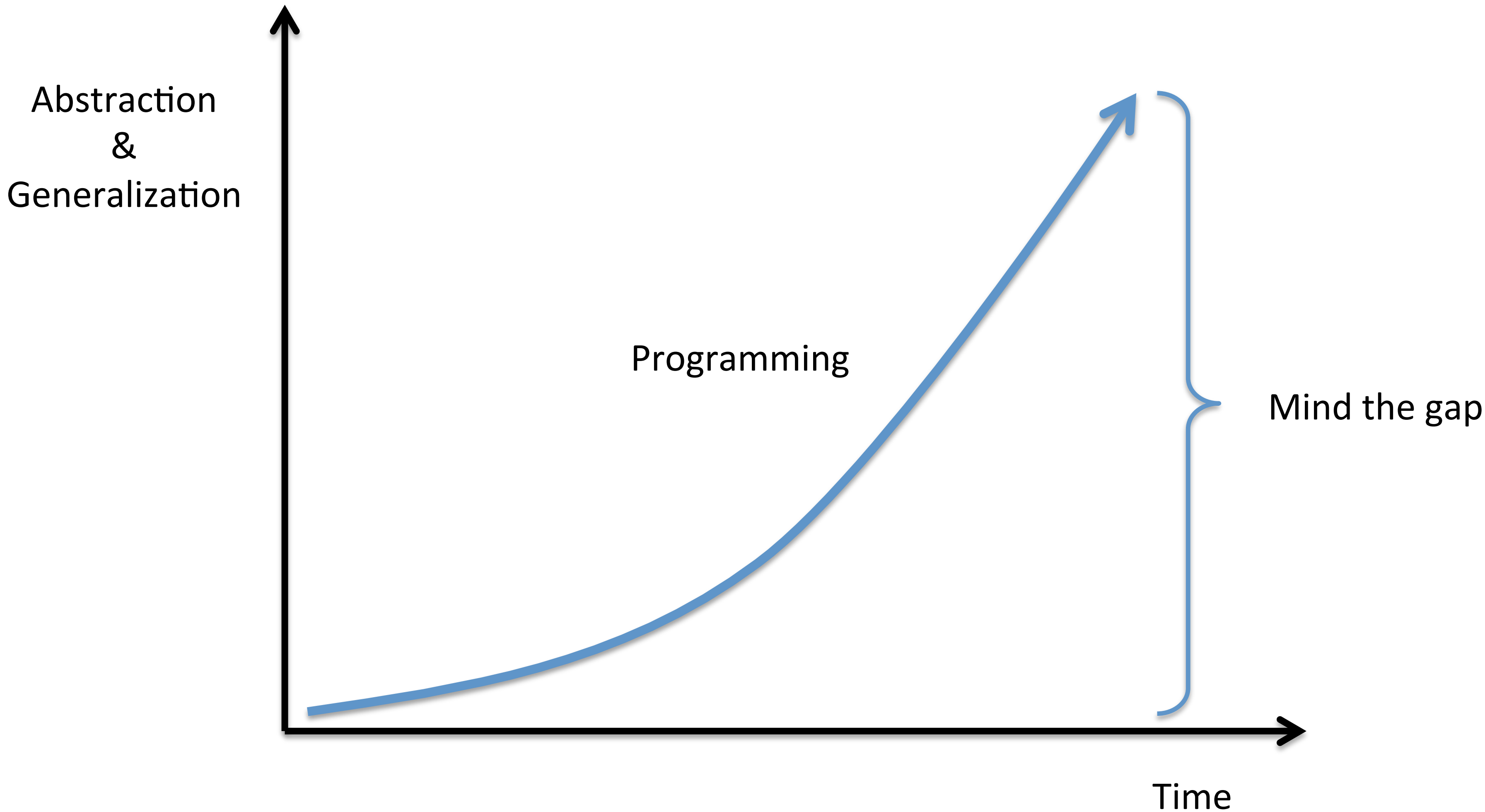
Type Classes

Indirection

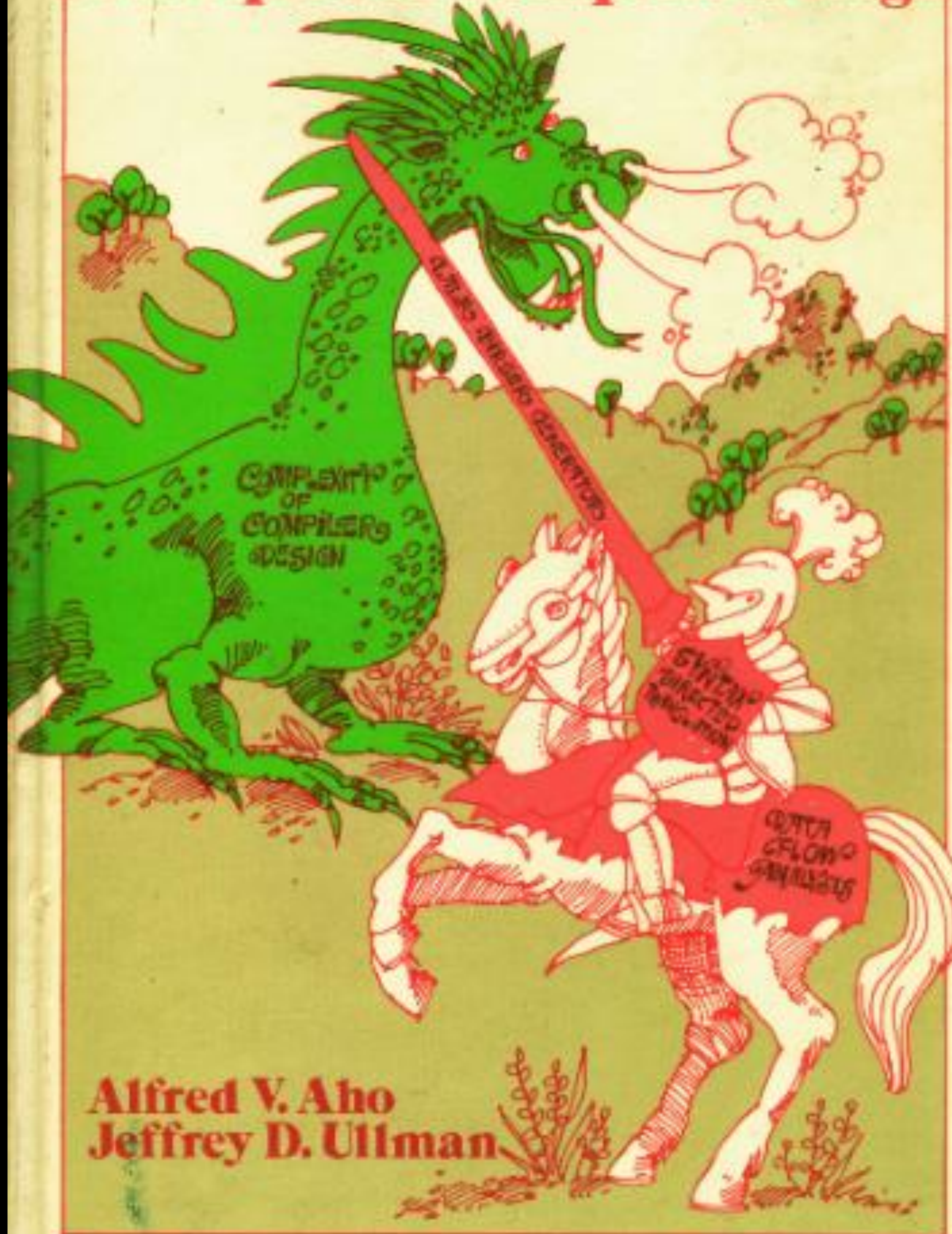
Closures (and megamorphic call sites)

Method Calls

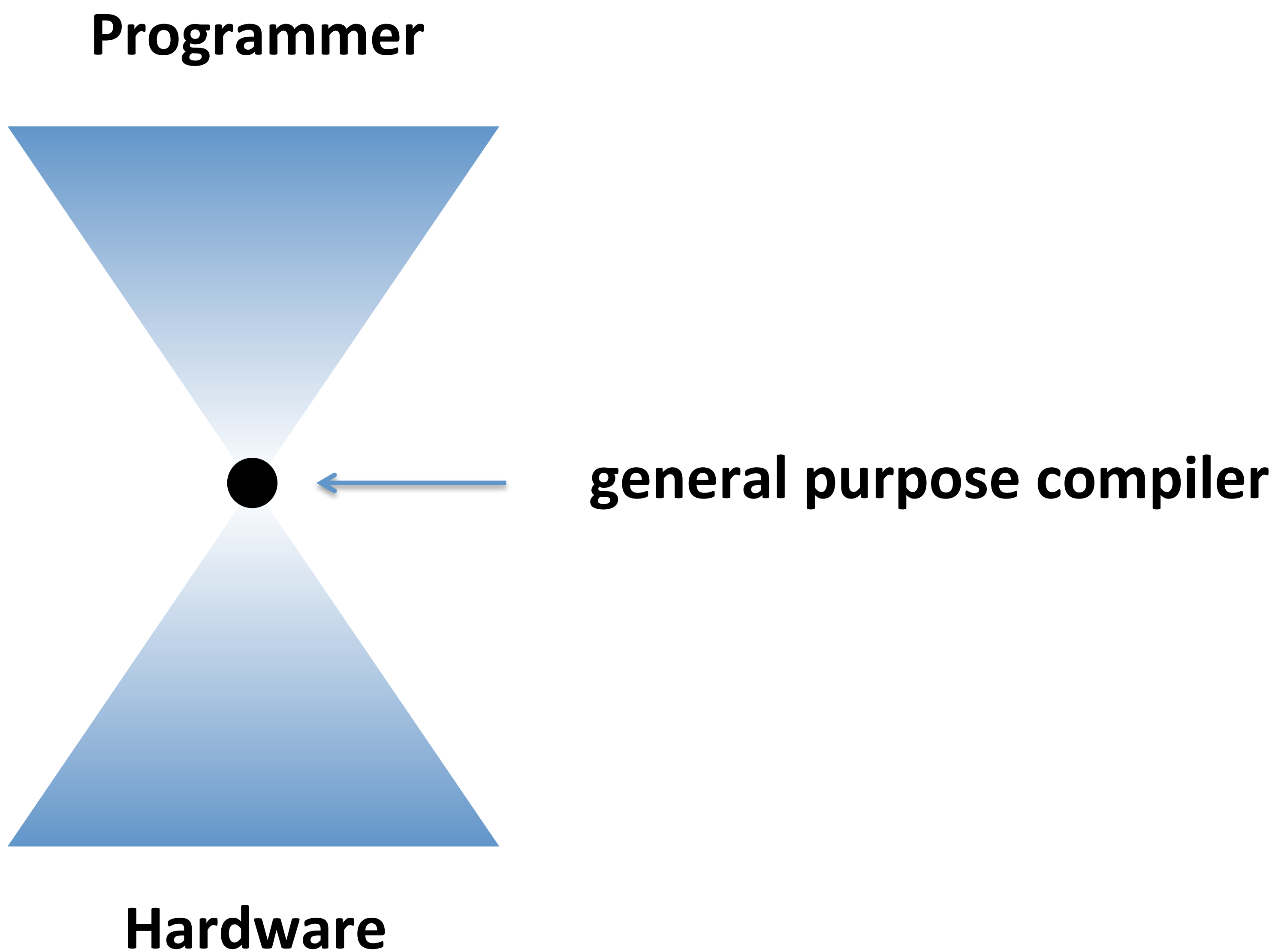

Abstraction
&
Generalization
Programming
Time


Abstraction
&
Generalization
Programming
Mind the gap!
Time

Principles of Compiler Design

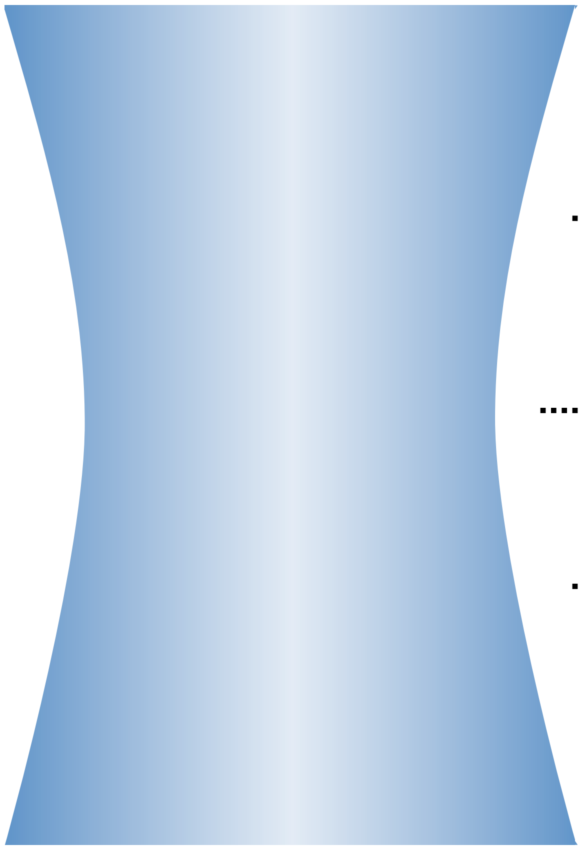


Alfred V. Aho
Jeffrey D. Ullman



(illustration: Markus Püschel)

Programmer



Hardware

Matrix, Graph, ...

Array, Struct, Loop, ...

SIMD, GPU, cluster, ...

- **horizontal and vertical** extensibility
- **generic optimizations** at each level (cse, dce, ...)

Secret Weapon: LMS

LMS = Lightweight Modular Staging

- An extensible compiler framework
- Implemented as a Scala library
- Execute 'now' vs 'compile and exec later'
- Specialize and compile program pieces at runtime

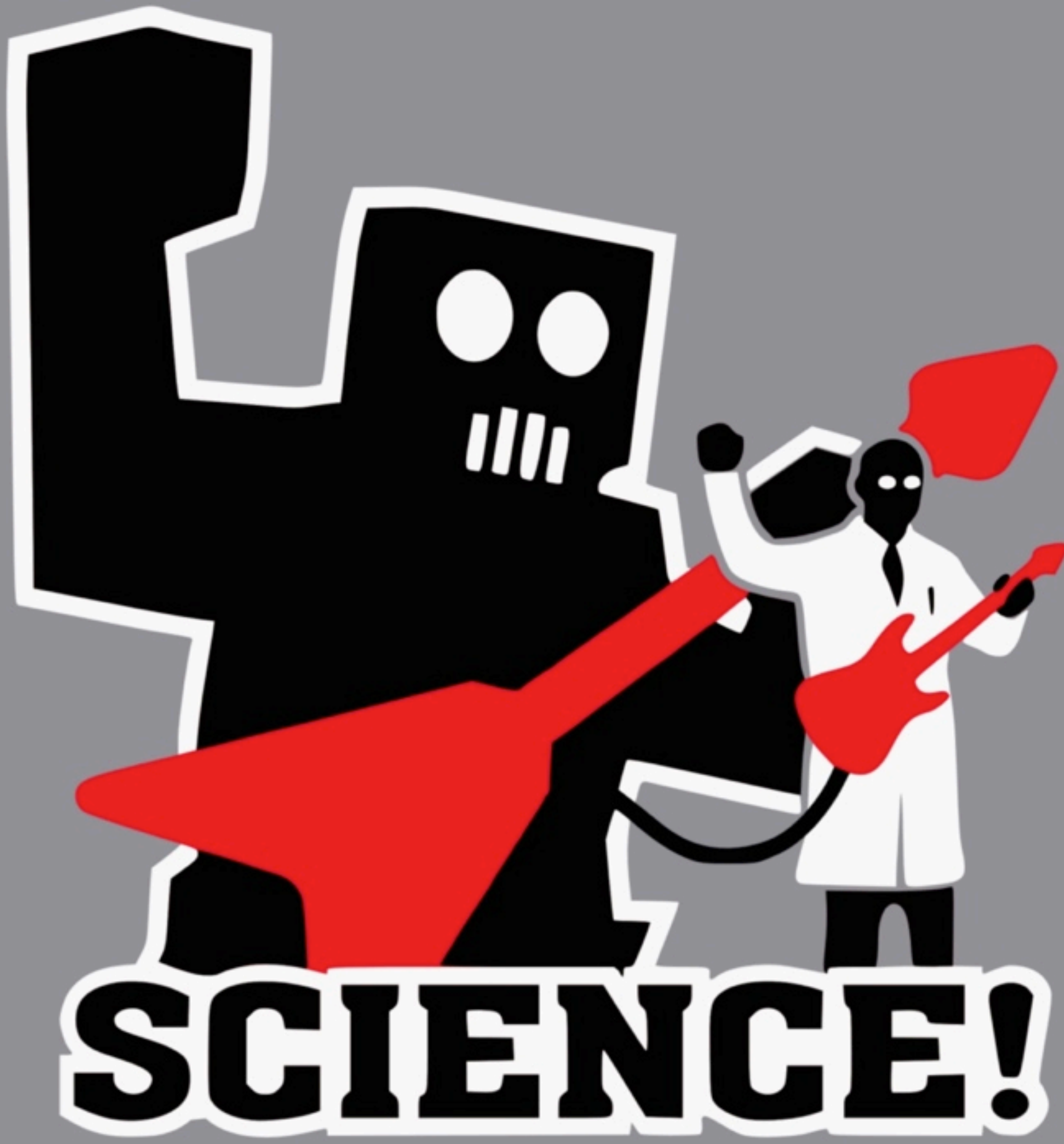
Staging = Multi-Stage Programming

- Computations can generally be separated into stages (Jørring, Scherlis 1986), distinguished by:
  - frequency of execution
  - availability of data

- Multi-Stage Programming (Taha, Sheard 1997): make stages explicit in a program:
  - "delay" expressions to a generated stage
  - "run" delayed expressions
  - staged program fragments as first class values

Generative

METAPROGRAMMING



(by Amorphia Apparrel)

program generically ...

... and run specialized !

Projects / Collaborations

- Delite (Stanford)
  Onward!'10, PACT'11,DSL'11,IEEE Micro 10/11, ECOOP'13 ,GPCE'13, TECS 4/14
  – DSLs and Big Data on heterogeneous devices
- Spiral (ETH) GPCE'13, ARRAY'14
  – Fast numeric libraries
- LegoBase (EPFL DATA) DCP'14, VLDB'14
  – Databases and query processing
- Lancet (Oracle Labs) PLDI'14
  – Integrate LMS with JVM / JIT compilation
- Hardware (EPFL PAL) FPT'13, FPL'14
  – Domain specific HW synthesis
- Parser Combinators (EPFL LAMP) OOPSLA'14
  – Protocols and dynamic programming
- JavaScript (EPFL, INRIA Rennes) ECOOP'12, GPCE'13
  – LMS for the web

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

LMS = Lightweight Modular Staging

- Int, String, T
  - "execute now"
- Rep[Int], Rep[String], Rep[T]
  - "generate code to execute later"
- if (c) a else b   ->   __ifThenElse(c,a,b)
  - "language virtualization"
- Extensible IR, transformers, loop fusion, ...
- "Batteries included"

Example: Matrix

```
class Matrix[T:Numeric:Manifest](val rows: Rep[Int], val cols: Rep[Int]) {
  private val arr: Rep[Array[T]] = ArrayNew[T](rows * cols)
  private val num = implicitly[Numeric[T]]; import num._

  def apply(i: Rep[Int], j: Rep[Int]): A =
    arr(i*cols + j)

  def update(i: Rep[Int], j: Rep[Int], e: Rep[A]): Unit =
    arr(i*cols + j) = e

  def *(that: Matrix[T]) = {
    val res = new Matrix[T](this.rows, that.cols)

    for (i <- 0 until this.rows) {
      for (j <- 0 until that.cols) {
        for (k <- 0 until this.rows)
          res(i, j) += this(i, k) * that(k, j)
      }
    }
    res
  }
}
```

Matrices are “now” objects, their data arrays are “later” objects

Generate Low-Level Code

```
var x27 = 500 * 500
var x28 = new Array[Double](x27)
var x29: Int = 0
while (x29 < 500) {
  var x30: Int = 0
  while (x30 < 500) {
    var x31: Int = 0
    while (x31 < 100) {
      ...
      x31 += 1
    }
    var x46 = ()
    x46
    x30 += 1
  }
  var x47 = ()
  x47
  x29 += 1
}
```

```
val m = randomMatrix(500, 100
val n = randomMatrix(100, 500)

val p = m * n
```

--- generic took 2.691s
--- generic took 1.4s
--- generic took 1.464s
--- generic took 1.359s
--- generic took 1.244s

--- double took 1.062s
--- double took 1.228s
--- double took 1.076s
--- double took 1.03s
--- double took 1.076s

--- staged took 0.088s
--- staged took 0.058s
--- staged took 0.055s
--- staged took 0.054s
--- staged took 0.056s

20x!

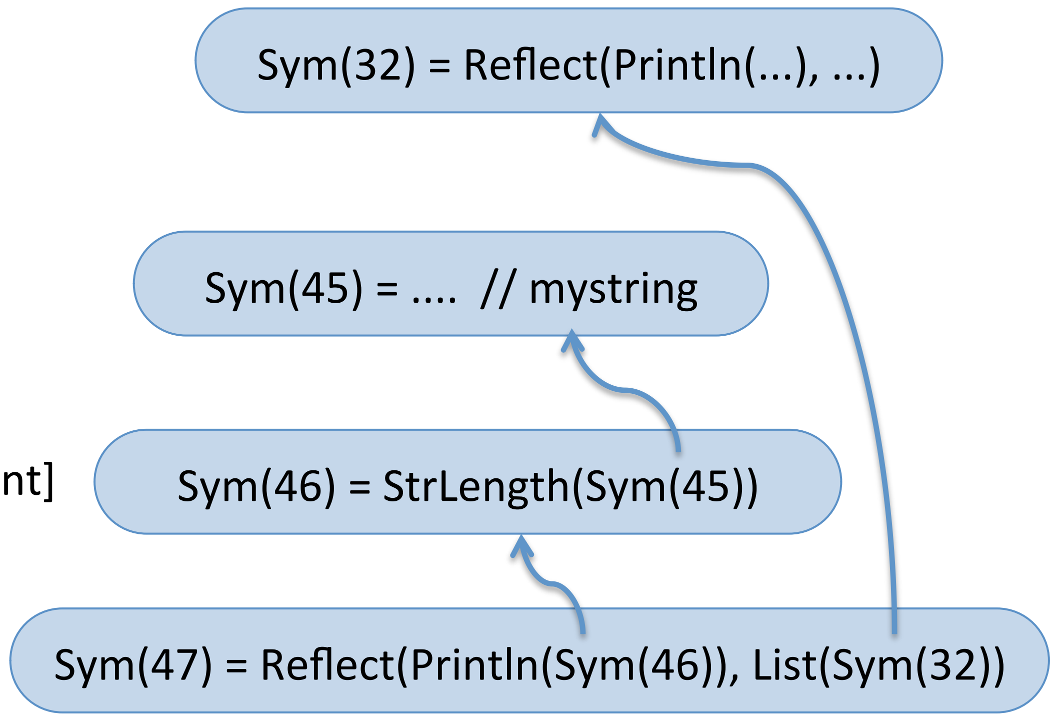
(still far from optimal:
should block loops for locality)

User code

```
println(...)
val mystring = ... // Rep[String]
println(mystring.length)
```

DSL interface

```
type Rep[T]
def infix_length(s: Rep[String]): Rep[Int]
def println(x: Rep[Any]): Rep[Unit]
```



DSL Implementation

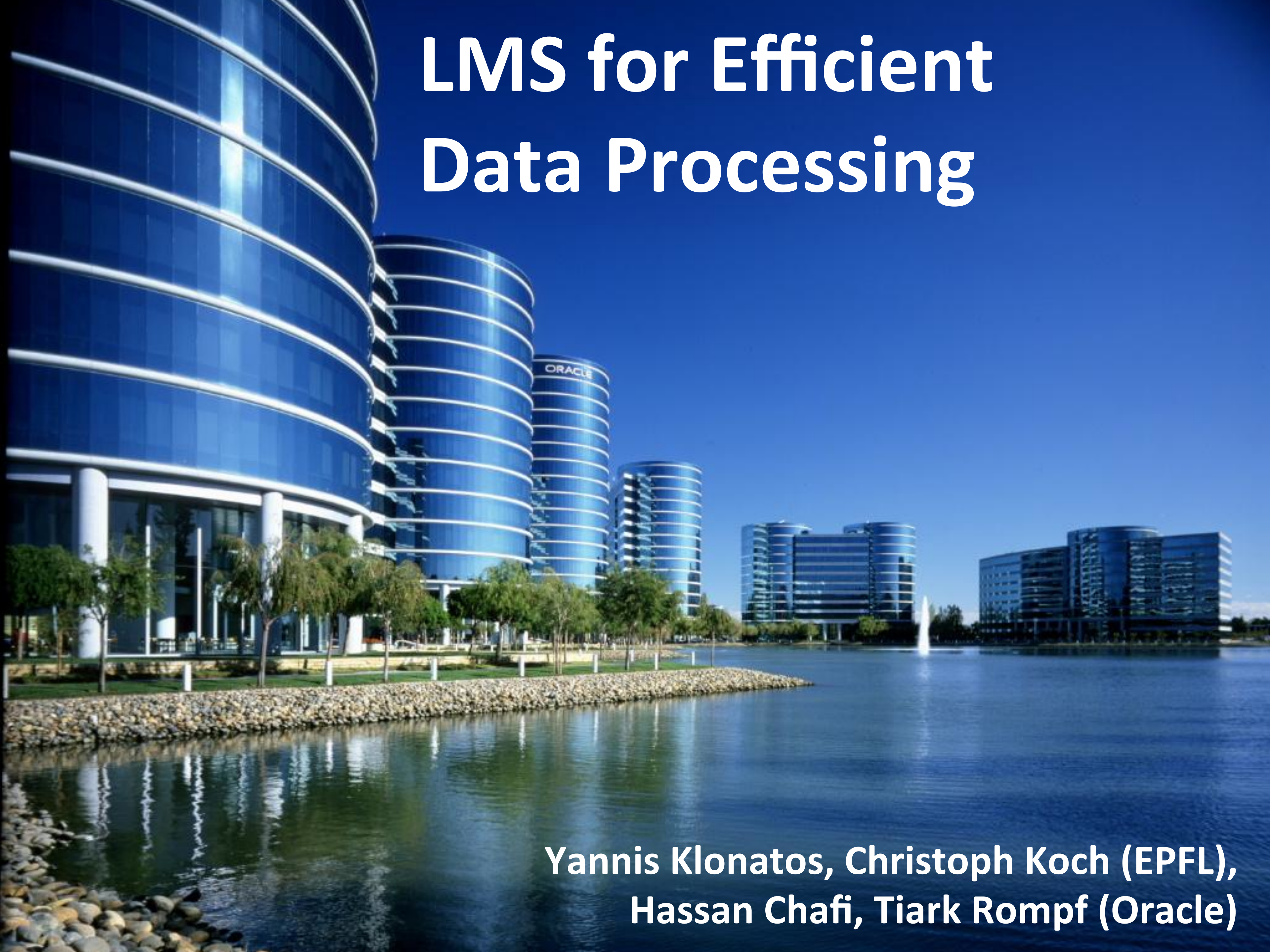
```
type Rep[T] = Exp[T]  // Sym[T] | Const[T]

case class StrLength(s: Exp[String]) extends Def[Int]
case class Println(s: Exp[Any]) extends Def[Unit]

def infix_length(s: Exp[String]): Exp[Int] = s match {  case Const(s) => Const(s.length)
                                                         case _ => reflectPure(StrLength(s))  }
def println(x: Exp[Any]): Exp[Unit]         = reflectEffect(Println(x))
```

Demo Time

https://scala-lms.github.io/tutorials/shonan.html

http://scala-lms.github.io/tutorials/linq.html

LMS for Efficient Data Processing



Yannis Klonatos, Christoph Koch (EPFL),
Hassan Chafi, Tiark Rompf (Oracle)

Databases: State of the Art

- Popular generic DBMSs consist of > 1M lines of optimized C code
- Manual specialization for performance – e.g. PostgreSQL:
  - 20 implementations of memory page abstraction
  - 7 implementations of B-trees
- Difficult to adapt
  - e.g. disk based → in memory
- Still 10 – 100x slower than hand-written queries
  (Stonebraker: time for a complete rewrite, Zukowski: Monetdb/x100)
- Commercial DBMS interpret query execution plans
  - some research on query compilation using LLVM (e.g. HyPer)

LegoBase

- New in-memory DB query engine, written in Scala
- Staged query interpreter
  - Compiles query execution plans (from Oracle DB) to C code
  - Supports all 22 TPCH queries
  - ~3000 lines of Scala code
- Use LMS for additional optimizations
  - Operator inlining
  - Optimizing data structures
  - Optimizing control flow (push vs pull)

It is indeed possible to write high performance systems in a high level language

Klonatos et al. VLDB '14


VLDB
2014
Very Large Data Base Endowment Inc.
(VLDB Endowment)
VLDB Best Paper Award
presented to
Tiark Rompf
for the paper entitled
Building Efficient Query Engines in
a High-Level Language
40th International Conference on Very Large Data Bases
September 1st-5th, Hangzhou, China

A SQL engine in 500 LOC

https://scala-lms.github.io/tutorials/query.html

Rompf and Amin, ICFP'15

Data is not only stored but also transferred

Efficient, hand-optimized HTTP parser

```
switch (s) {
   case s_req_spaces_before_url:
     if (ch == '/' || ch == '*') {
       return s_req_path;
     }
     if (IS_ALPHA(ch)) {
       return s_req_schema;
     }
     break;

   case s_req_schema:
     if (IS_ALPHA(ch)) {
       return s;
     }

     if (ch == ':') {
       return s_req_schema_slash;
     }
     break;
```

- Originally part of Nginx, later Node.js
- 2000+ lines of code
- Callbacks for header names/values triggered
- State-machine like code
- “Flat” code, loops/conditions

Jonnalagedda et al. OOPSLA'14

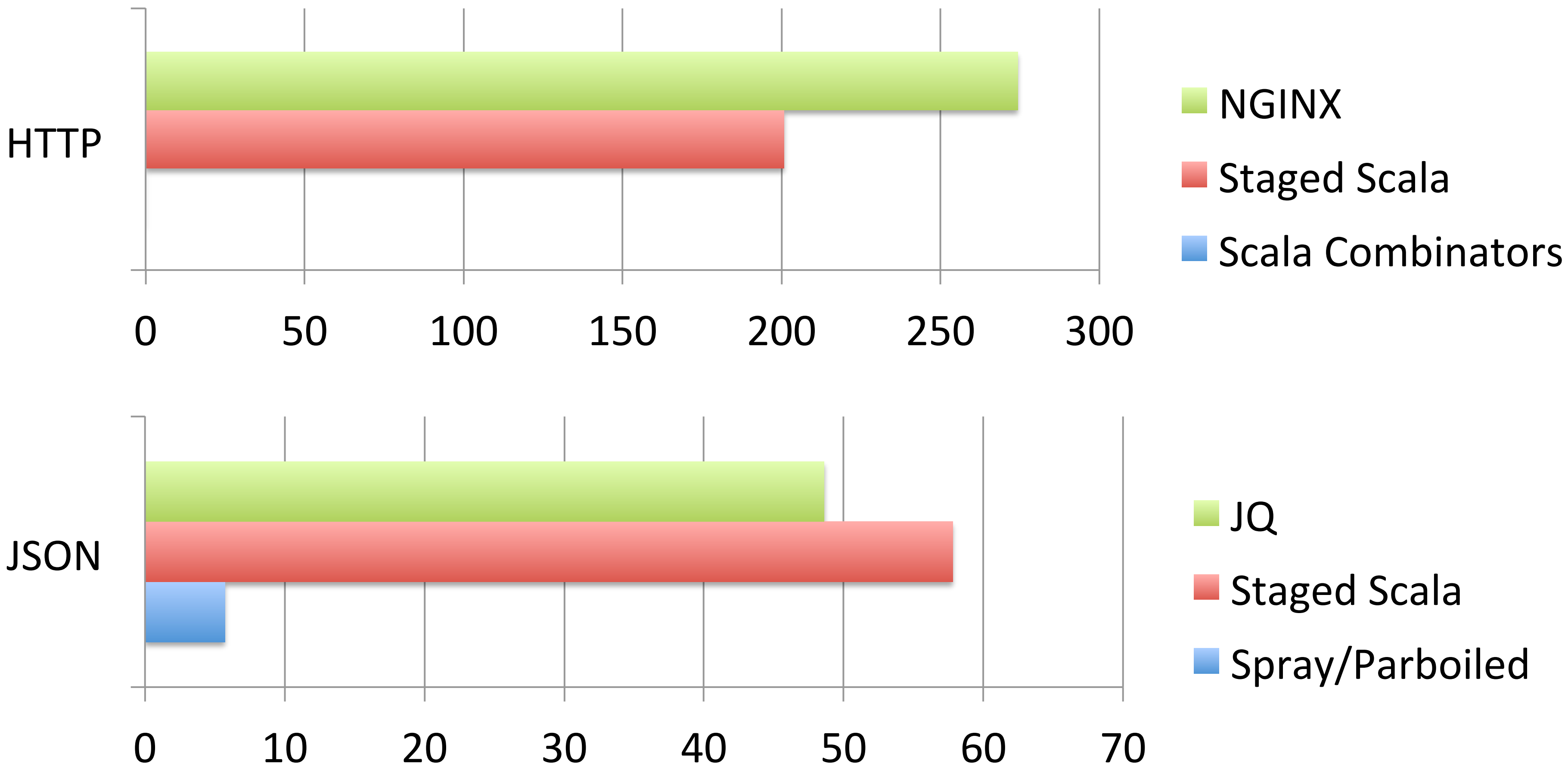
Staged Parser Combinators

```
def status: Parser[Int] =
    ("HTTP/"~decimalNumber)~>wholeNumber<~(wildRegex~crlf) ^^ (_.toInt)

def header: Parser[Option[(String,Any)]] =
    (headerName<~":")~(wildRegex<~crlf) ^^ {
        case hName~prop => collect(hName.toLowerCase, prop)
    }

def headers = rep(header)

def response = status ~ headers

...
```

- 200+ lines of code
- Fairly easy to change behaviour of a parser
  - Ex: ~ vs <~ vs ~>

Jonnalagedda et al. OOPSLA'14

Staged Parser Combinators



Jonnalagedda et al. OOPSLA'14