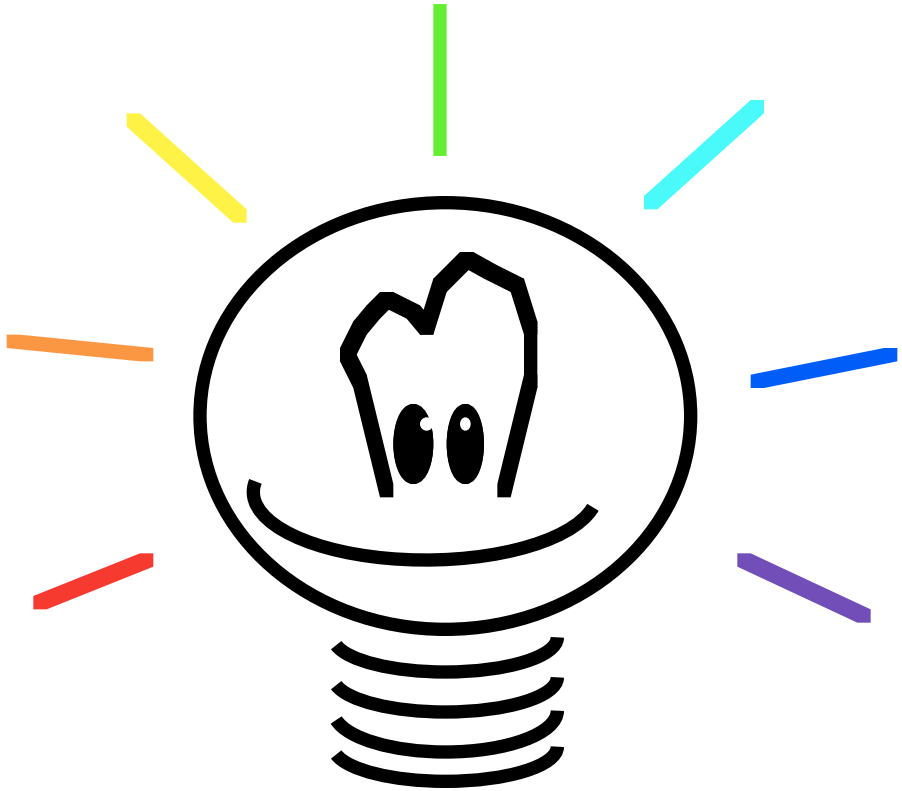
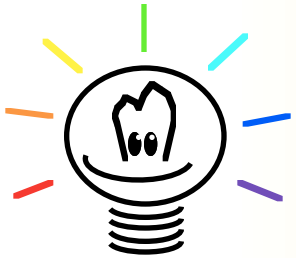
Xroma - Two Years Later

Progress Report

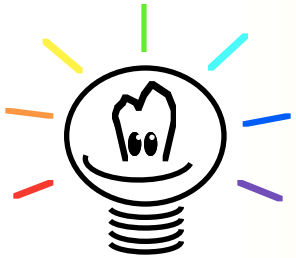




Key Points

Just Remember This

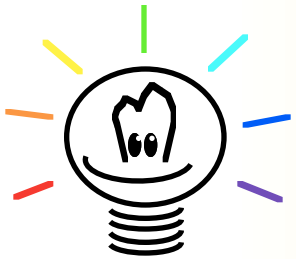
- Software Development: Fighting Moore's Law
  - Complexity Grows by Moore's Law, Tools Don't
  - This Impacts Our Daily Lives: Bugs, Costs, Complexity
- Proposal: "Concept Programming"
  - WYSIWYG Philosophy of Programming
  - Need a *Revolution* in Development Tools
- Continuous Effort, Many Results
  - Xroma, LX: Cool Ideas
  - Mozart, Moka: Infrastructure, First Usable Tools
  - XL: State of the Art



Problem Statement

Why Waste My Time?

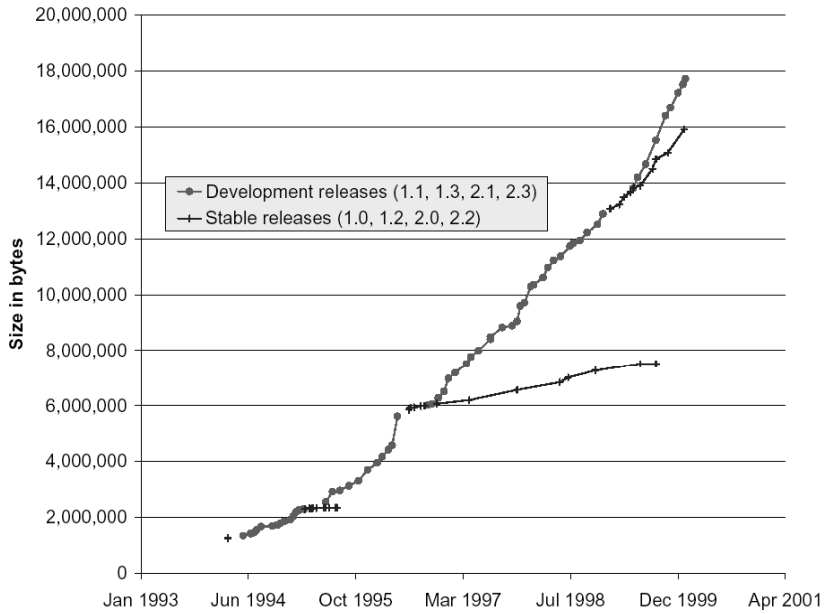
- Software Grows Too Fast
  - Complexity Follows Moore's Law
  - Increased Business Pressure - "Time To Market"
- No Incremental Growth of Development Tools
  - Tools Grow Discontinuously - "Paradigm Shifts"
  - Last Big Two: Java (Internet) and C++ (GUI)
- Direct Impact on Our Lives
  - Software is Always Late
  - Software is Always Buggy



Explosive Growth

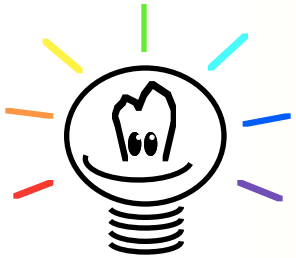
The Initial Stage

- Linux Kernel Growth



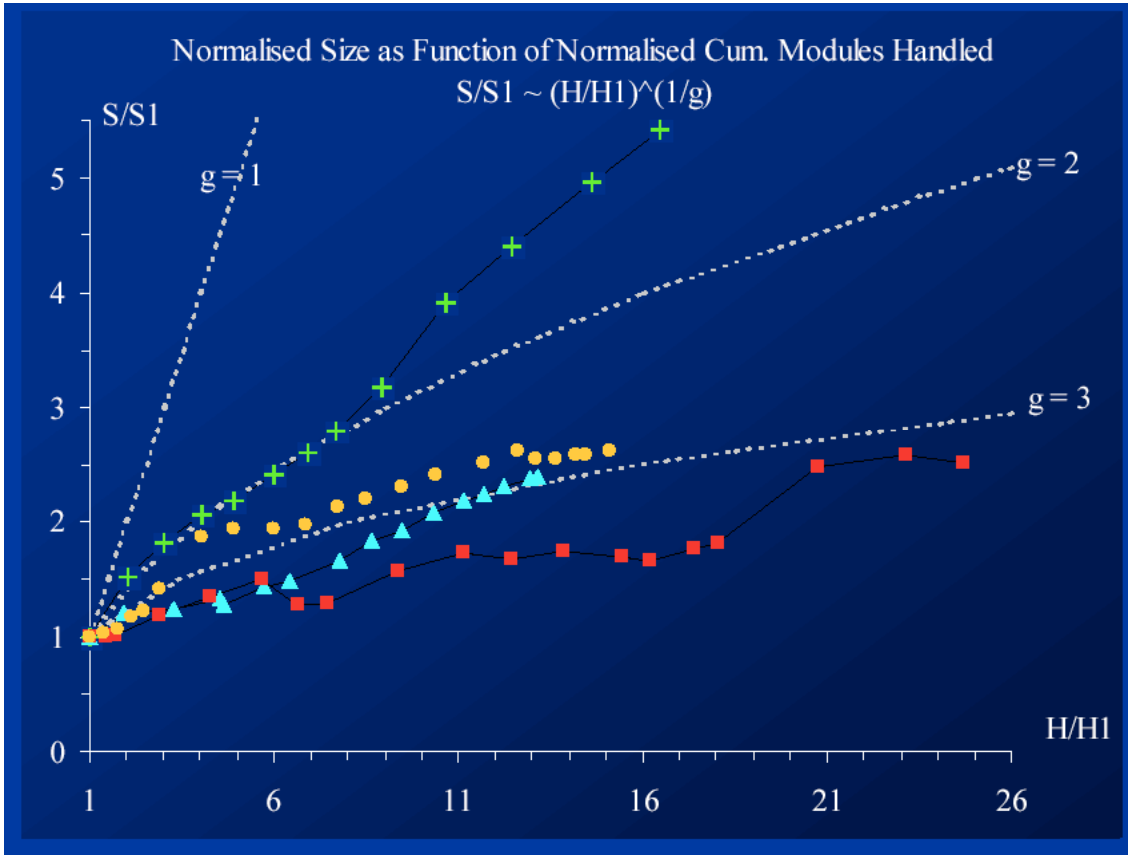
1. Growth of the compressed `tar` file for the full Linux kernel source release.

Source: http://plg.uwaterloo.ca/~migod/papers/icsm00.pdf

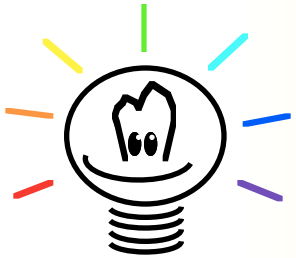


Reaching Saturation

Diminishing Returns



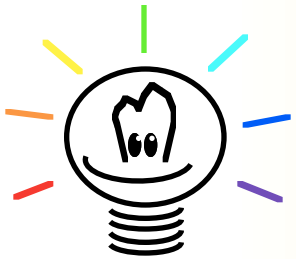
Source: http://www.doc.ic.ac.uk/~mml/feast2/papers/pdf/jfr103c.pdf



Concept Programming

Old Ideas, New Spin

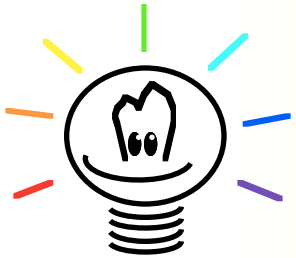
- Code Should Represent Application Concepts
  - Map Domain Space to Code Space
  - Necessary and Sufficient Level of Abstractions
  - Minimize "Artificial Complexity"
- Enable Ecosystems of Concepts
  - Transposition of Application Domain Ecosystem
- Example: OO Methodology
  - Objects Represent "Names"
  - Methods Represent "Verbs"



Simple Concept

A General "Max" Function in XL

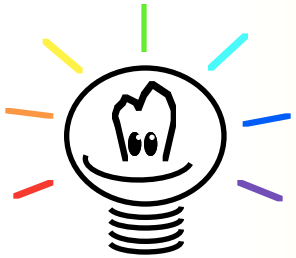
```
generic type ordered if
   with ordered A, B
   with boolean Test := A < B

function Max(ordered X) return ordered is
   return X
function Max(ordered X; other) return ordered is
   result := Max(other)
   if result < X then result := X

procedure Test() is
  with real R := Max(1.0, 3.0, 5.0)
  with integer I := Min(1, 3, 4, 5, 6, -1)
```



Concepts ≠ Objects

Object-Oriented "Max"

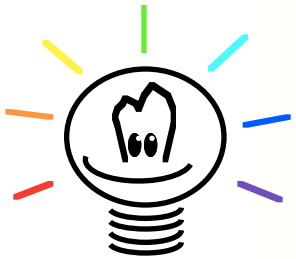
```
#include <iostream>

using namespace std;

template<class T>
class Maximum {
private:
   T max;
public:
   Maximum(T X) { max = X; }
   Maximum &operator,(T X) {
      if (X > max)
         max = X;
      return *this;
   }
   operator T() {
      return max;
   }
};

template <class T>
Maximum<T> Max(T t) {
   return Maximum<T>(t);
}

int main() {
   cout << "Max(2,8,5,7)="
        << (Max(3),8,5,7)
        << endl;
   cout << "Max(2.5,8.2,5.1,7.3)="
        << (Max(2.5),8.2,5.1,7.3)
        << endl;
}
```



What Have We Gained

Not Just Incremental Benefits

- Less "Artificial Complexity"
  - No Need for Intermediate "Maximum" Manager Object
  - No Comma Operator Overloading
  - Less Punctuation
- More Safety
  - "Ordered" Validates its Arguments
  - Type-Safe Variable Argument Lists - WriteLn
- More Expressive Power
- The Right Level of Abstraction



Expression Reduction

Operator Overloading++

- Reduction of Function Calls

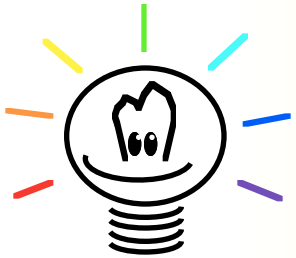
```
function MultiplyAndAdd(matrix A, B, C) return matrix
      written A * B + C
matrix M := M1 * M2 + M3 * M4 + M5
```

- Reduction of Generic Types

```
generic [type item] type pointer written pointer to item
pointer P to item
```

- Reduction of Constructors

```
generic [type item] function vector(integer Size)
      return vector of item
      written vector[Size] of integer
vector V[3] of integer
```



True Generics
For a Better STL

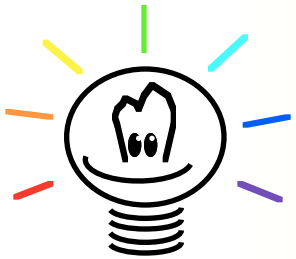
- Standalone (non parameter) Generic Types

```
generic type ordered
function Max(ordered A) return ordered
```

  - *Declare* a Real Concept
  - C++: Convention on Template Argument Names
- Validation Clauses

```
generic type ordered if
      with ordered A, B
      with boolean C := A<B
```

  - *Model* a Real Concept
  - Enhance Robustness, Enable Diagnostics



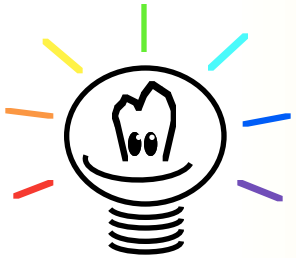
Improving Performance

Abstraction ≠ !Efficiency

- Giving Information to the Compiler
  - Avoiding "Noisy" Semantics, such as Implicit Pointers
  - Keeping Freedom of Implementation

```
void DrawRect(Rect *r);
```

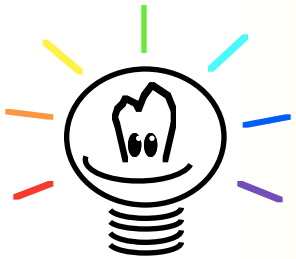
- "Complex Numbers" Core Code (Julia Sets)
  - 70% faster than C++ on Itanium
  - Major Benefit: Everything in Registers
  - XL is 7x Faster if C++ uses <iostream>



Improving Performance

No Loss in Abstraction

```
type complex is record with
      real Re, Im

function Complex(real Re, Im := 0.0) return complex is
      result.Re := Re
      result.Im := Im

function Add(complex X, Y) return complex written X+Y is
      result.Re := X.Re + Y.Re
      result.Im := X.Im + Y.Im
```

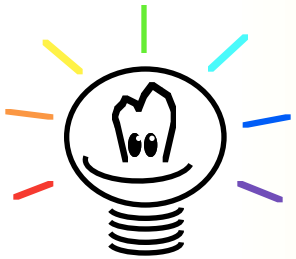


Higher-Order Concepts

Example: Symbolic Derivative

- Here is What You'd Like to Write

```
class Test
{
   public static final double omega = 3.276;
   public static final double theta = 0.227;
   public static final double decay = 1.447E-3;

   public static int main(String args[]) {

      // Tabulate the following expression
      for (double t = 0.0; t < 50.0; t += 0.01) {
        double y = d(Math.sin(2 * omega * t + theta) * Math.exp(-decay * t))/dt;
         System.out.println("t=" + t + ", y=" + y);
      }
      return 0;
   }
}
```

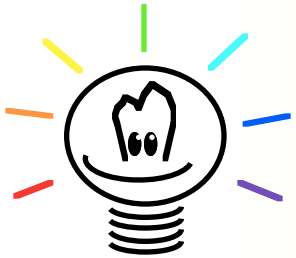


Language Extensions

Moka External Plug-ins

```
ddd% ./moka tests/derivation.java +derivation +constantfold -out
/* Generated by Moka using moka.stylesheet */
// This example demonstrates the symbolic derivation "plugin"
class Test
{
  public static final double omega = 3.276;
  public static final double theta = 0.227;
  public static final double decay = 0.001447;

  public static int main(String[] args)
  {

    // Tabulate the following expression
    for(double t = 0; t < 50; t += 0.01)
    {
      double y = Math.cos (2 * omega * t + theta) * (2 * omega) * Math.exp
               (-(decay * t)) + Math.sin (2 * omega * t + theta)
               * -(Math.exp (-(decay * t)) * decay);
       System.out.println ("t=" + t + ", y=" + y);
    }
    return 0;
  }
```

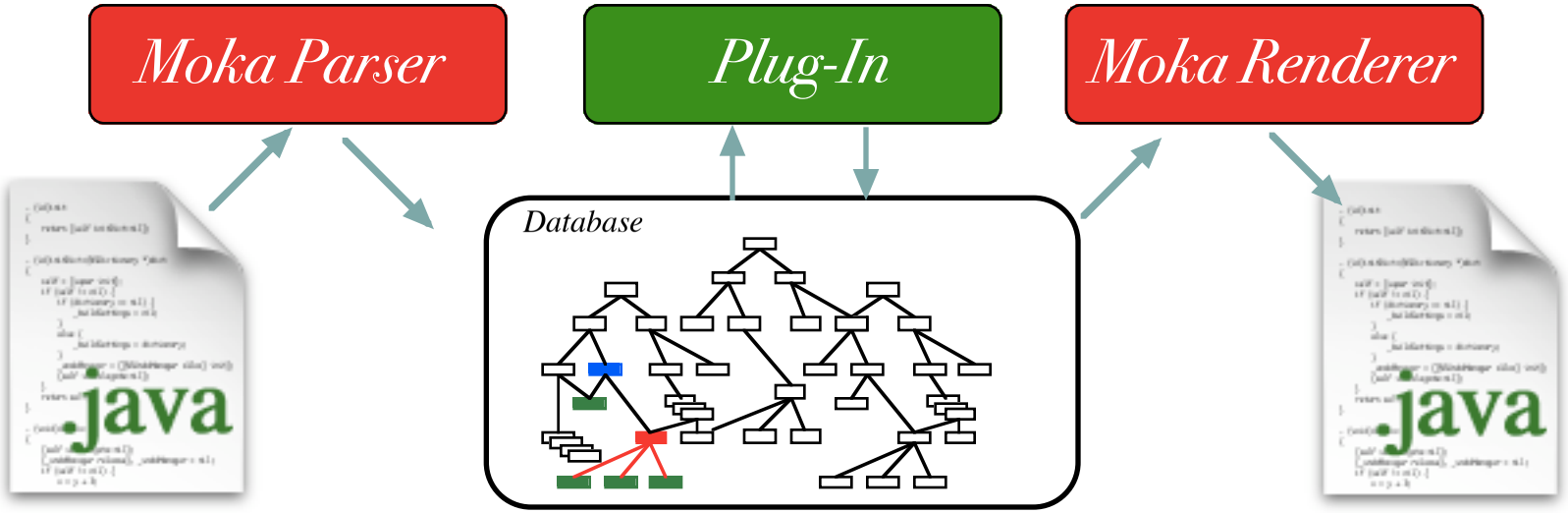
- A Plug-In With 464 Lines of Code (Commented)

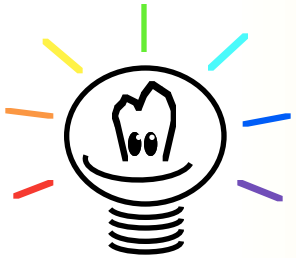


How Moka Works

Mozart API - Persistent DB

- Separation of Concerns
  - Moka: Parsing and Unparsing Java Code
  - Mozart: Program Representation and Persistence
  - Plug-In: Derivatives

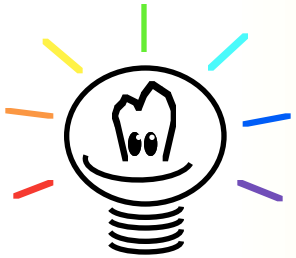




Other Applications

A few existing Moka Plug-Ins

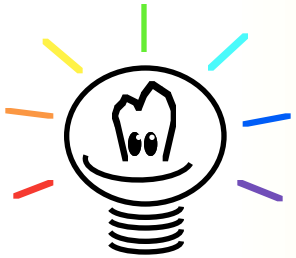
- Symbolic Derivation
- Constant Folding and Simplification
- Programming by Contract
- Stripping Selected Code (debug code, tracing)
- Generating Execution Traces ("profiling")



Extensions = Dialects

Philosophy Conflict

- Concept Abstractions That You can't even *Read*
  - One Dialect Per Developer - Best Case
  - Non Obvious Boundaries
  - Hidden External Dependencies (found in Command Line)
- Still Worth Fixing
  - More than Incremental Increase in Abstraction
  - General Manipulation of Tree

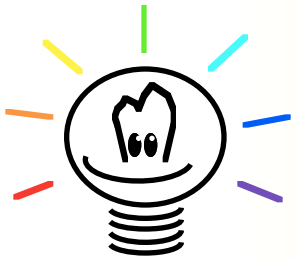


XL Pragmas
Escape Codes

- Non-Invasive Pragma Notation

```
for t in 0.0..50.0 step 0.01 loop
      {derivation} real Y := d(sin(2*omega*t+theta) *
                                    exp(-decay*t))/dt
      WriteLn "t=", t, " y=", y
```

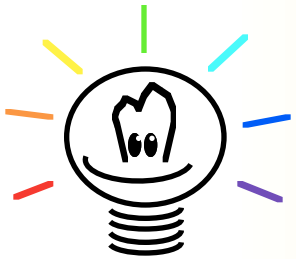
- Fix Dialect Issues
  - Make Use of Extensions Visible
  - Pragma Name Indicates Dependency
  - Ensure Locality of Pragma Effects



Unlimited Capabilities

"Revolution"

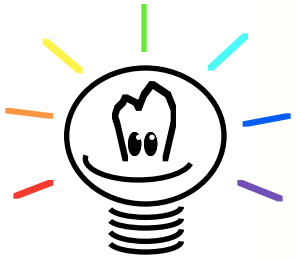
- Implementation Details
  - {by_value}, {bit_size}, {address}, {volatile}, {debug}
- Optimizations
  - {inline}, {fast}, {commutative}
- Object Models
  - {C "bcopy"}, {dynamic}, {persistent}, {clonable}
- Foreign Paradigms
  - {task}, {prolog}
- Custom-Defined
  - {derivation}, {warn "Obsolete!"}, {doc "This is a function"}



Modelling Tasking

Ada-Like Syntax... Library-Made

```
{protected} record Buffer with
   {entry} function Count() return unsigned
   {entry} function Available() return unsigned
   when Count() > 0:
      {entry} procedure Read(out character C)
   when Available() > 0:
      {entry} procedure Write(in character C)

// Producing task
task Producer is
  with character C
  loop
    C := ProduceCharacter()
    Buffer.Write C
    exit if C = ASCII.EOT
```

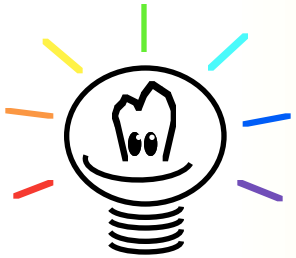


Progress Report

Summary of Results

- Mozart - The Foundations
  - Language-Independant Intermediate Language
  - Extensible, Reversible, Persistent
- Moka - Java to Java Compiler
  - Parser and Unparser
  - Java Extensions using Compiler Plug-Ins
- XL Compiler
  - Supports "Concept Programming"
  - More Efficient than C / C++ on Modern Processors

http://mozart-dev.sf.net



Key Points

Just Remember This

- Software Development: Fighting Moore's Law
  - Complexity Grows by Moore's Law, Tools Don't
  - This Impacts Our Daily Lives: Bugs, Costs, Complexity
- Proposal: "Concept Programming"
  - WYSIWYG Philosophy of Programming
  - Need a *Revolution* in Development Tools
- Continuous Effort, Many Results
  - Xroma, LX: Cool Ideas
  - Mozart, Moka: Infrastructure, First Usable Tools
  - XL: State of the Art