ORACLE®

Truffle: A Self-Optimizing Runtime System

Thomas Wuerthinger
Oracle Labs

JVM Language Summit, July 31, 2012

Java, Python, Ruby,
JavaScript, Groovy,
Clojure, Scala, ...

Generality
+
Performance

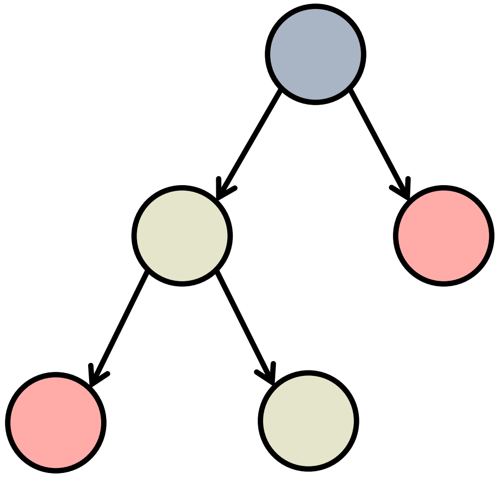
```
function f(a, n) {
   var x = 0;
   while (n-- > 0) {
      x = x + a[n];
   }
   return x;
}
```

$\Rightarrow$

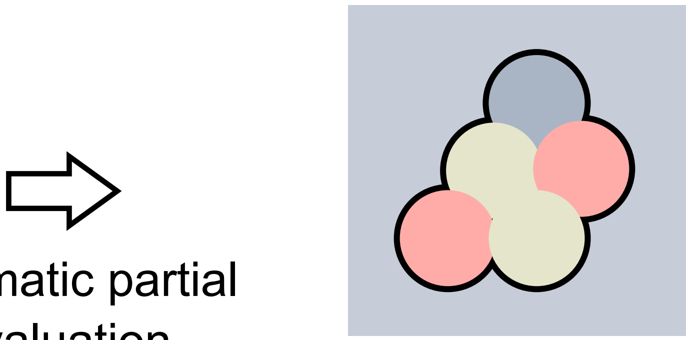
```
L1: decl rax
jz L2
movl rcx, rdx[16+4*rax]
cvtsi2sd xmm1, rcx
addsd xmm0, xmm1
jmp L1
L2:
```

AST Interpreter

Compiled Code



automatic partial evaluation



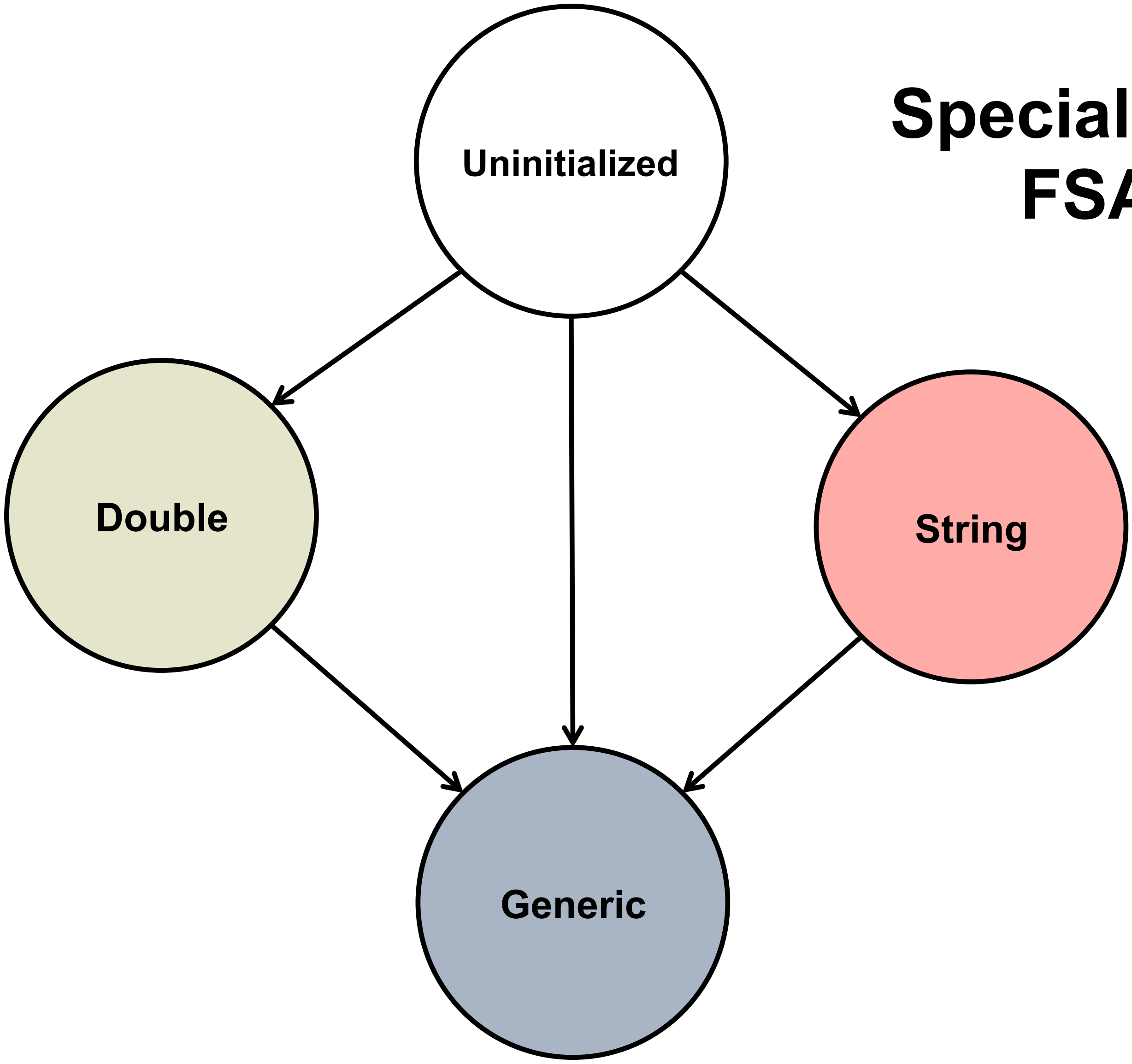
```
Object add(Object a, Object b) {
   if(a instanceof Double && b instanceof Double) {
      return (double)a + (double)b;
   } else if (a instanceof String && b instanceof String) {
      return (String)a + (String)b;
   } else {
      return genericAdd(a, b);
   }
}
```

```
double add(double a,
           double b) {
   return a + b;
}
```

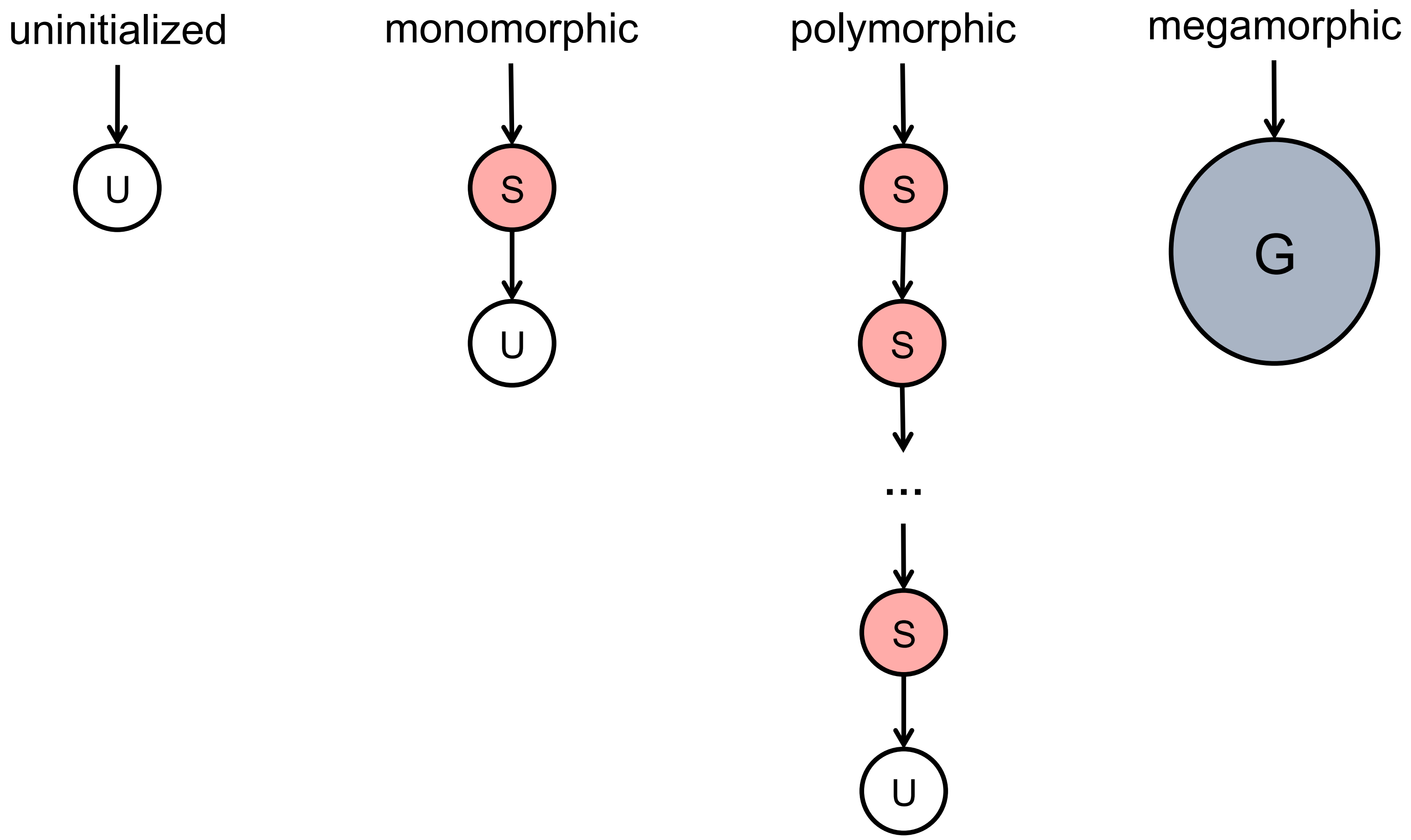
```
String add(String a,
           String b) {
   return a + b;
}
```

```
Object add(Object a,
           Object b) {
   return genericAdd(a, b);
}
```

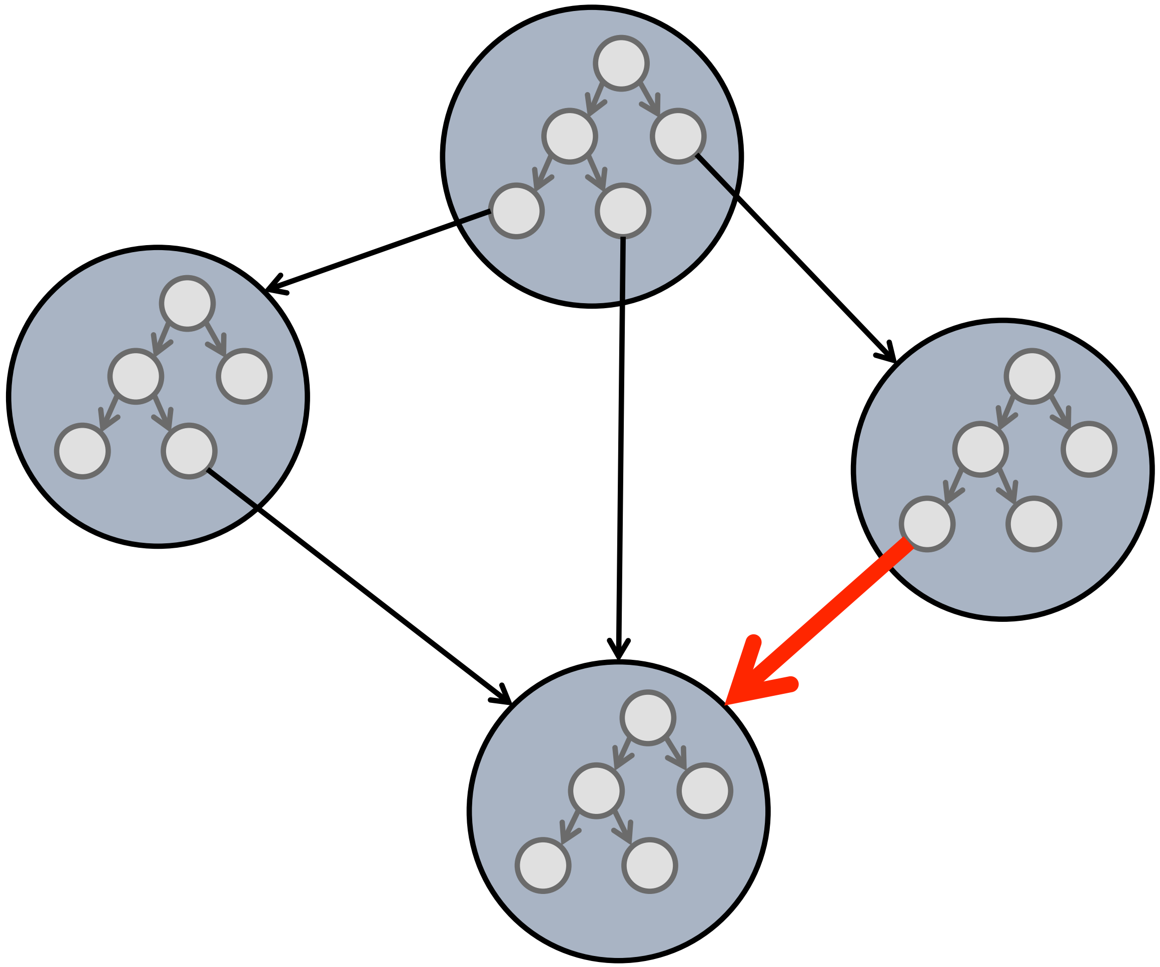
Specializing FSA



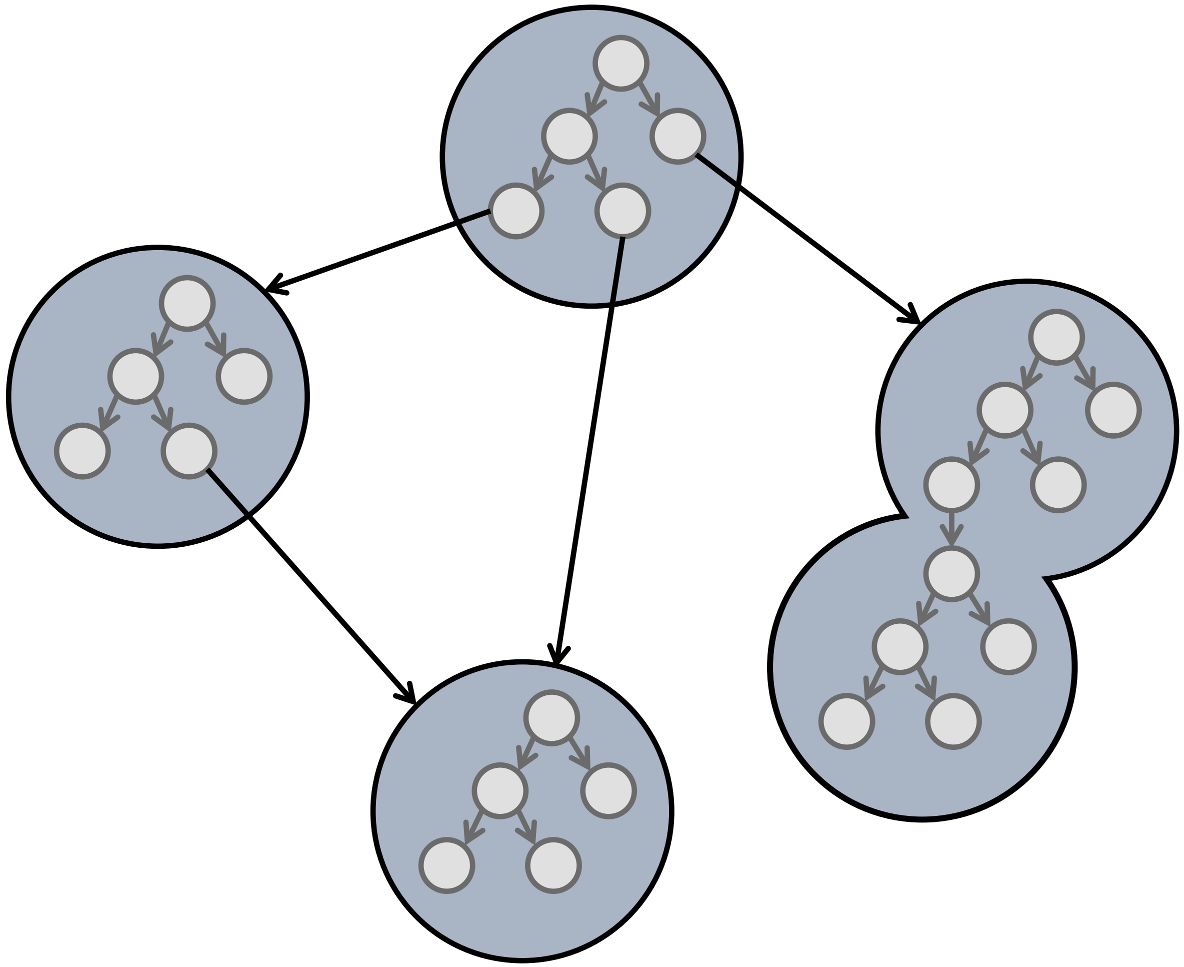
Inline Caching



Hot Call Site Detection



AST Level Inlining



```
function f(a, n) {
   var x = 0;
   while (n-- > 0) {
      x = x + a[n];
   }
   return x;
}
```

JavaScript AST

⇨

automatic partial evaluation

Java IR

```
Object f(Object[] args) {
   var a = args[0];
   var n = args[1];
   var x = 0;
   while (n-- > 0) {
      x = x + a[n];
   }
   return x;
}
```

parameters

```
Object f(Object[] args) {
    Object a = args[0];
    if (!(args[1] instanceof Integer)) deoptimize;
    int n = (int)args[1];
    double x = 0;
    while (n-- > 0) {
        x = x + a[n];
    }
    return x;
}
```

type specialized local variables

```
Object f(Object[] args) {
   Object a = args[0];
   if (!(args[1] instanceof Integer)) deoptimize;
   int n = (int)args[1];
   double x = 0;
   while (n-- > 0) {
      x = x + a[n];
   }
   return x;
}
```

control structures

```
Object f(Object[] args) {
   Object a = args[0];
   if (!(args[1] instanceof Integer)) deoptimize;
   int n = (int)args[1];
   double x = 0;
   while (n = safeDecrement(n) > 0) {
      x = x + a[n];
   }
   return x;
}
```

specialized operations

```
Object f(Object[] args) {
   Object a = args[0];
   if (!(args[1] instanceof Integer)) deoptimize;
   int n = (int)args[1];
   double x = 0;
   while (n-- > 0) {
      x = x + a[n];
   }
   return x;
}
```

overflow check elimination

```
Object f(Object[] args) {
   Object a = args[0];
   if (!args[1] instanceof Integer) deoptimize;
   int n = (int)args[1];
   double x = 0;
   while (n-- > 0) {
      if (!(a instanceof IntArray)) deoptimize;
      IntArray intArray = (IntArray)a;
      int[] content = a.content;
      if (n < a.lower || n > a.upper) deoptimize;
      x = x + content[n];
   }
   return x;
}
```

specialized array access

```
Object f(Object[] args) {
   Object a = args[0];
   if (!args[1] instanceof Integer) deoptimize;
   int n = (int)args[1];
   double x = 0;
   if (!(a instanceof IntArray)) deoptimize;
   IntArray intArray = (IntArray)a;
   while (n-- > 0) {
      int[] content = a.content;
      if (n < a.lower || n > a.upper) deoptimize;
      x = x + content[n];
   }
   return x;
}
```

loop invariant code motion

```
Object f(Object[] args) {
   Object a = args[0];
   if (!args[1] instanceof Integer) deoptimize;
   int n = (int)args[1];
   double x = 0;
   if (!(a instanceof IntArray)) deoptimize;
   IntArray intArray = (IntArray)a;
   int[] content = a.content;
   while (n-- > 0) {
      if (n < a.lower || n > a.upper) deoptimize;
      x = x + content[n];
   }
   return x;
}
```

loop invariant code motion

```
Object f(Object[] args) {
   Object a = args[0];
   if (!args[1] instanceof Integer) deoptimize;
   int n = (int)args[1];
   double x = 0;
   if (!(a instanceof IntArray)) deoptimize;
   IntArray intArray = (IntArray)a;
   int[] content = a.content;
   if (0 < a.lower || n > a.upper) deoptimize;
   while (n-- > 0) {
      x = x + content[n];
   }
   return x;
}
```

optimistic loop invariant code motion

ORACLE

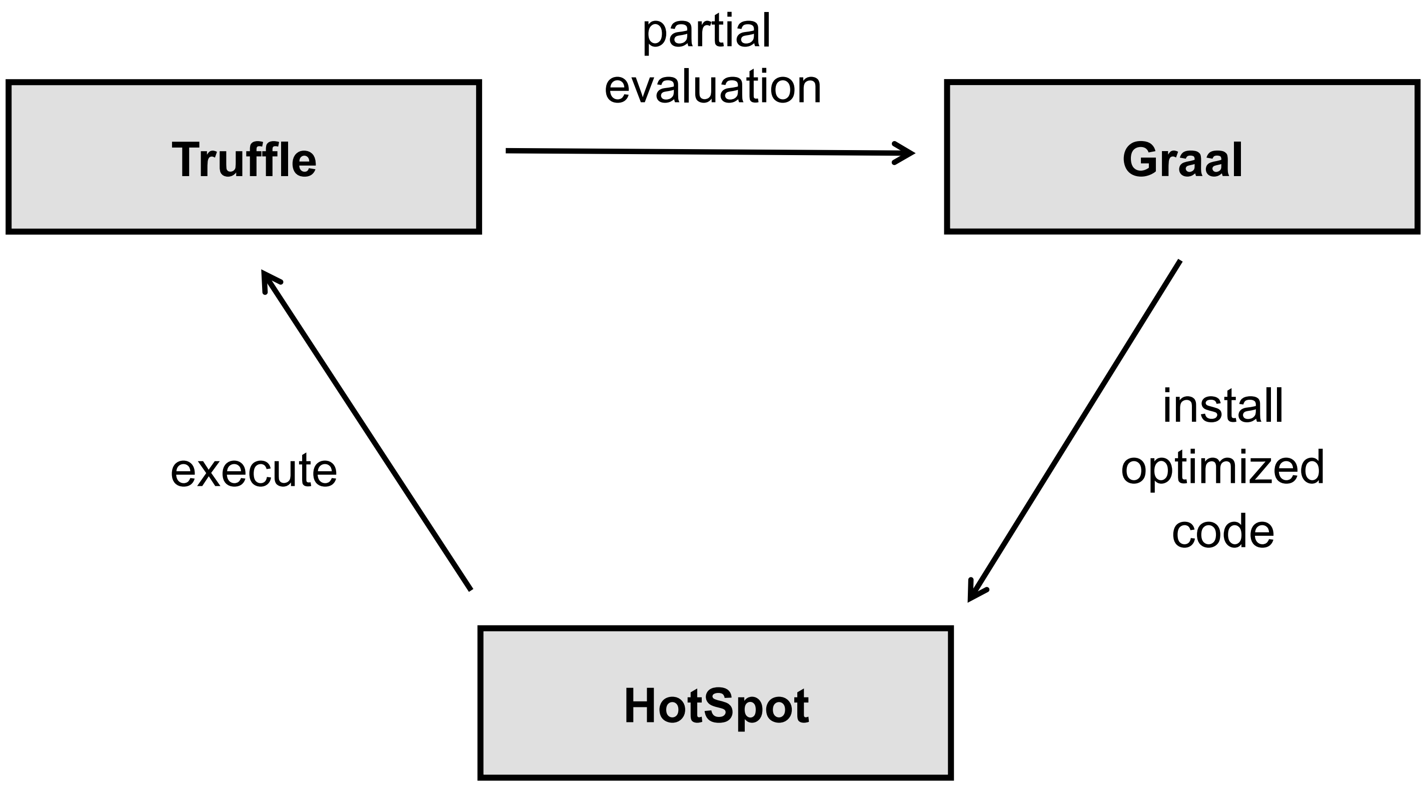
```
while (n-- > 0) {
   x = x + content[n];
}
```

```
L1: decl rax
jz L2
movl rcx, rdx[16+4*rax]
cvtsi2sd xmm1, rcx
addsd xmm0, xmm1
jmp L1
L2:
```

Java
IR

⇨
Graal compiler

Optimized Assembly


Truffle
partial evaluation
Graal
install optimized code
HotSpot
execute

Acknowledgements

Oracle Labs
Michael Haupt
Shams Imam (Intern)
Peter Kessler
Helena Kotthaus (Intern)
David Leibs
Doug Simon
Michael Van De Vanter
Christian Wimmer
Mario Wolczko
Thomas Wuerthinger

Purdue University
Floreal Morandat
Jan Vitek

JKU Linz
Gilles Duboscq
Matthias Grimmer
Christian Haeubl
Christian Humer
Christian Huber
Manuel Rigger
Lukas Stadler
Andreas Woess

Java, Python, Ruby, JavaScript, Groovy, Closure, Scala, ...

Q/A

Generality + Performance

```
function f(a, n) {
   var x = 0;
   while (n-- > 0) {
      x = x + a[n];
   }
   return x;
}
```

⇨

```
L1: decl rax
jz L2
movl rcx, rdx[16+4*rax]
cvtsi2sd xmm1, rcx
addsd xmm0, xmm1
jmp L1
L2:
```

Hardware and Software

ORACLE®

Engineered to Work Together

ORACLE®