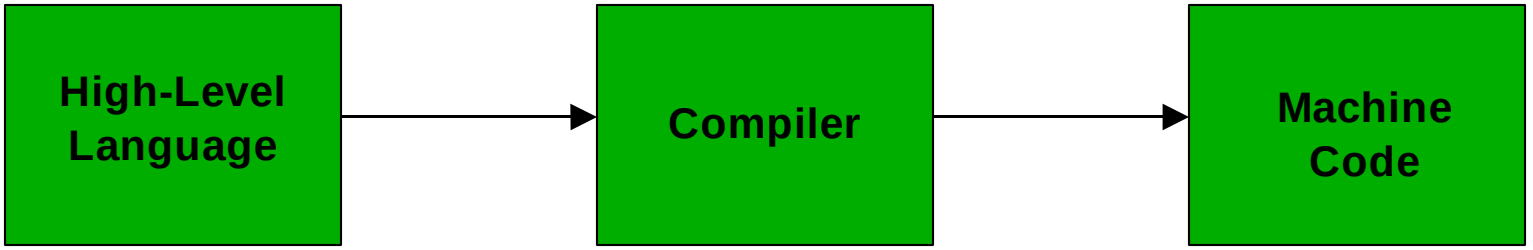
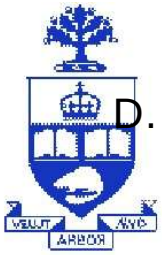
XJit: A Framework for Self-Optimizing Libraries (A Work in Progress)

Hamza Karamali
Derek Woo
Michael Voss

Motivation #1

Leveraging Programmer Abstractions

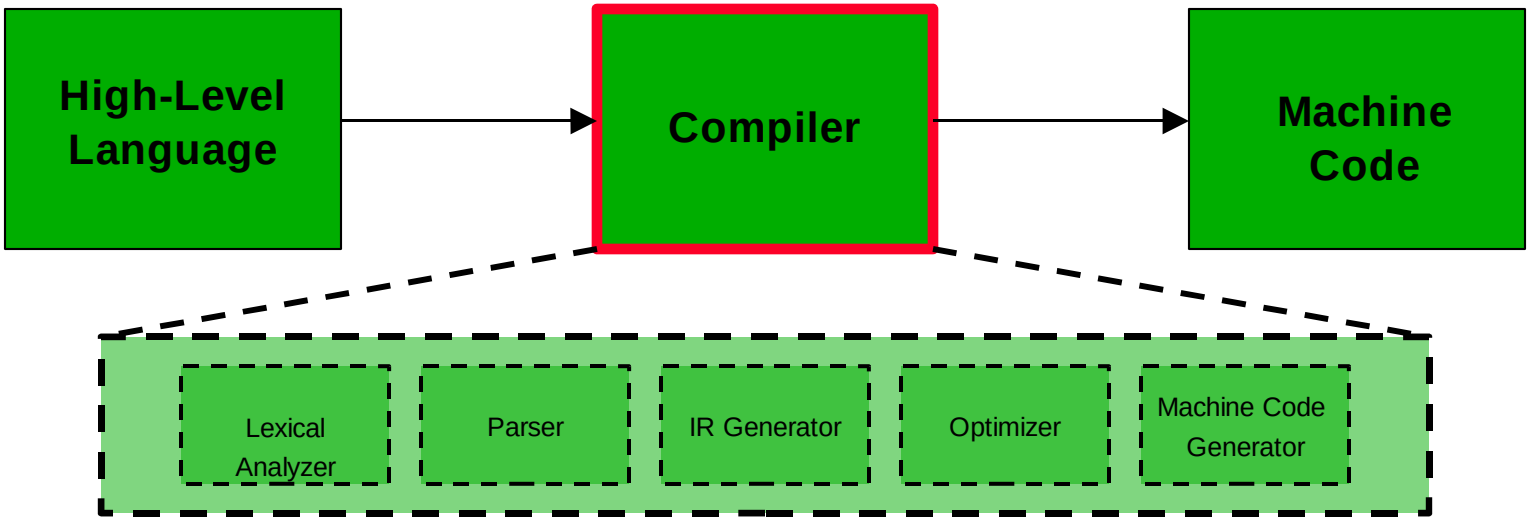




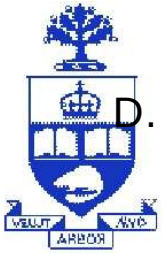
D. Engler. *Interface Compilation: Steps toward Compiling Interfaces as Languages.*

Motivation #1

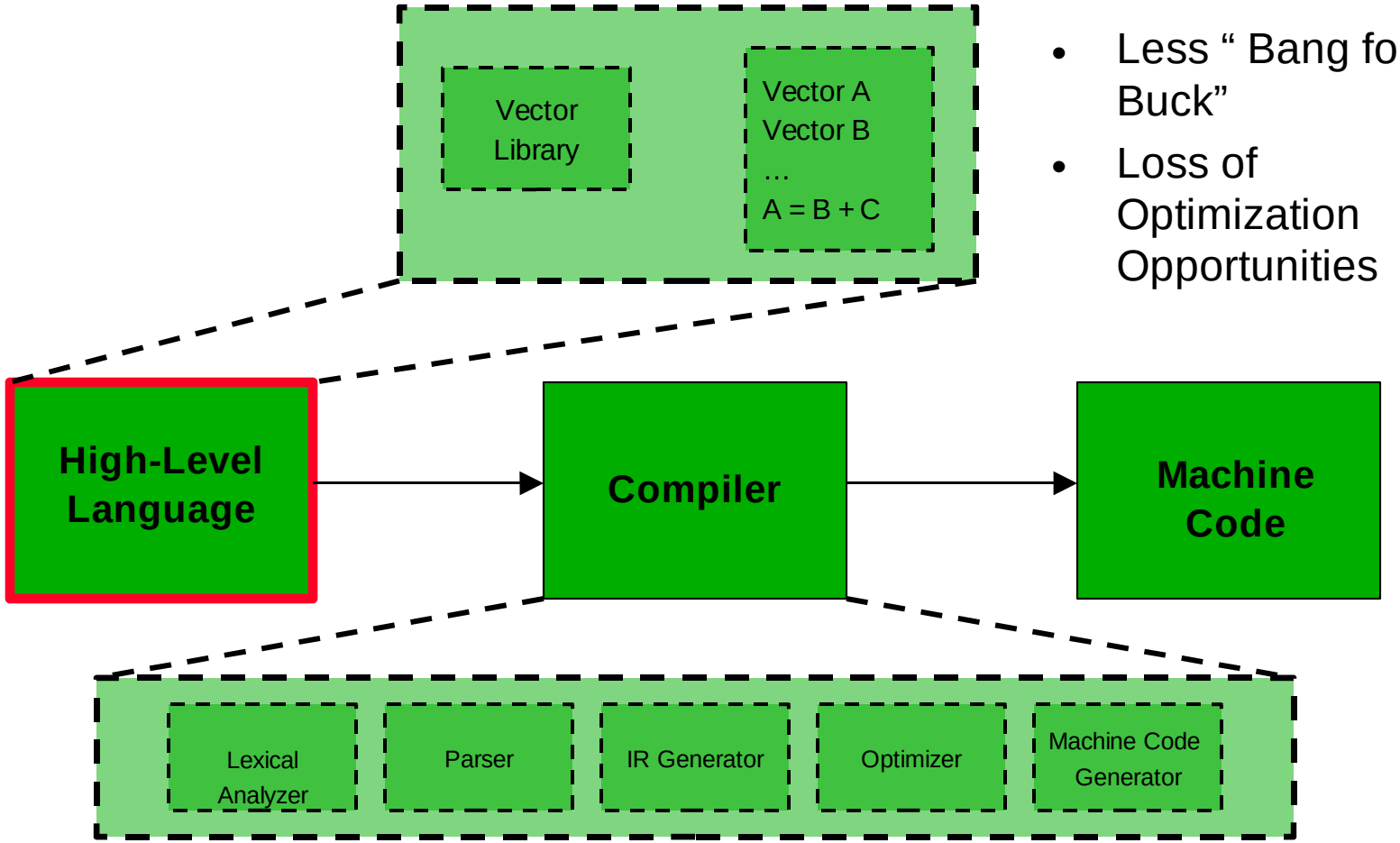
Leveraging Programmer Abstractions



D. Engler. *Interface Compilation: Steps toward Compiling Interfaces as Languages.*



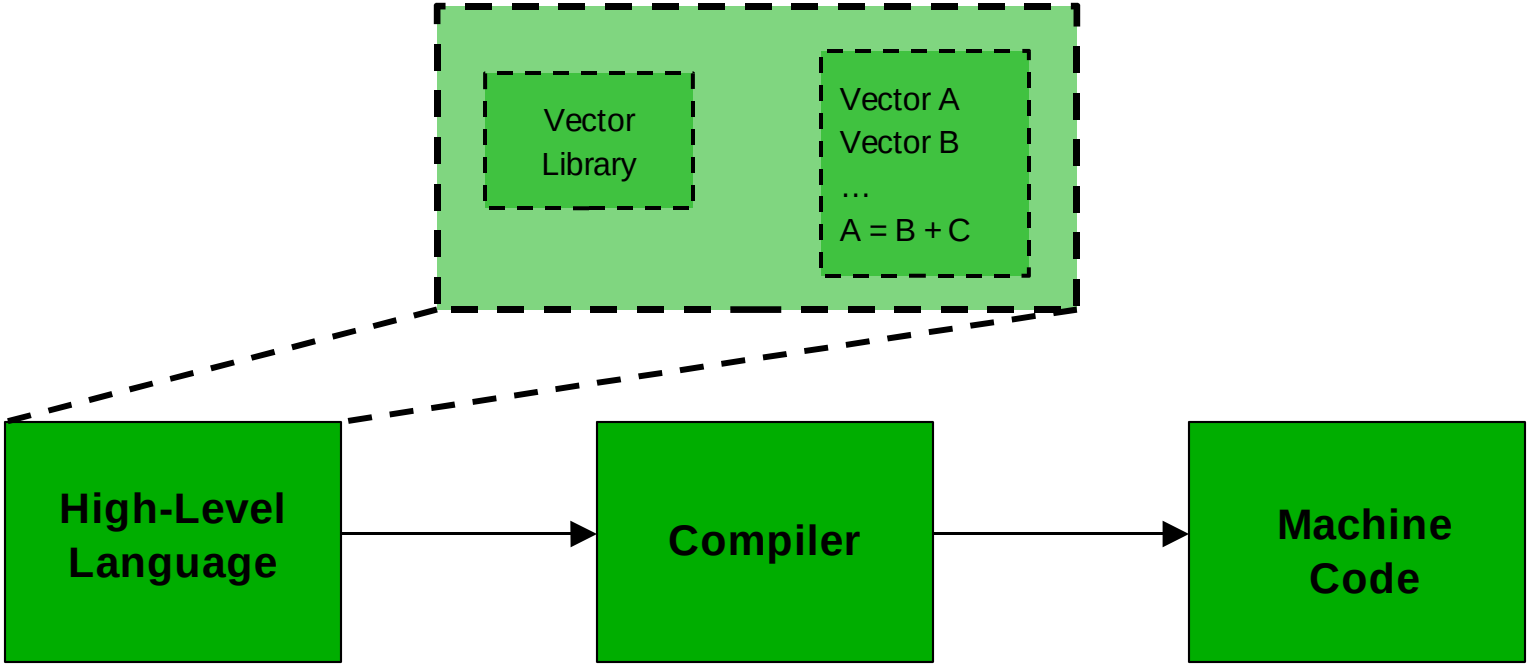
Motivation #1



- Less “ Bang for Buck”
- Loss of Optimization Opportunities

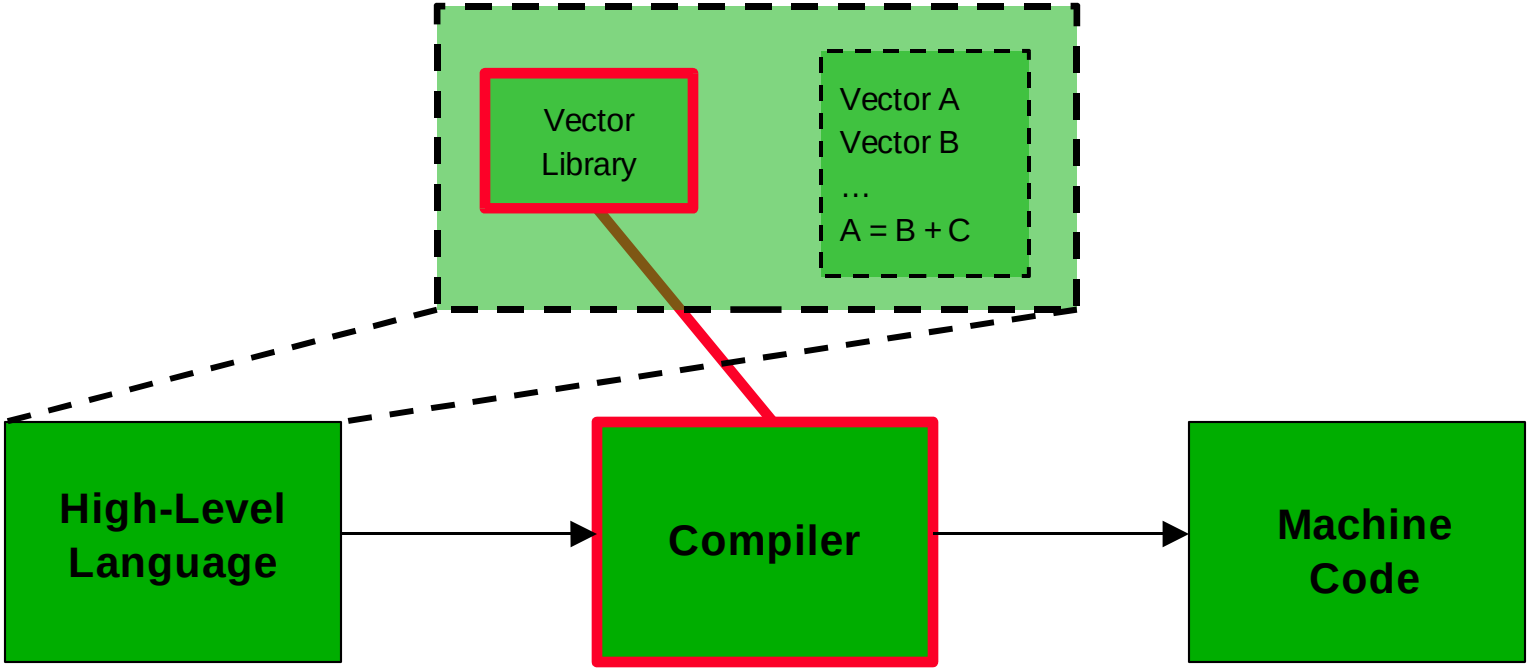
D. Engler. *Interface Compilation: Steps toward Compiling Interfaces as Languages.*

Motivation #1



P. Wu. *Efficient Support for Complex Numbers in Java.*

Motivation #1



P. Wu. *Efficient Support for Complex Numbers in Java.*

Motivation #2

Niche Optimizations and Market Forces

Inlining

Constant Propagation

Loop Unrolling

Register Allocation

Software Pipelining

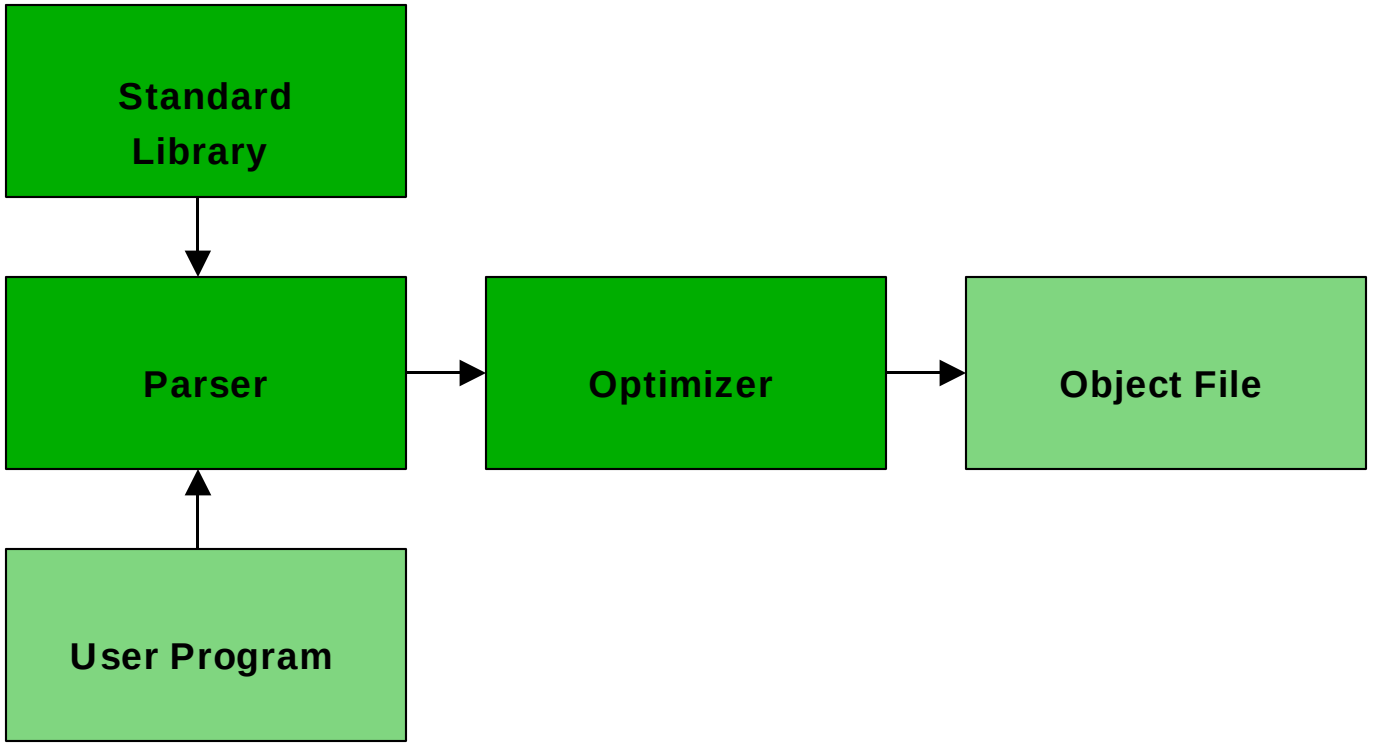
Instruction Scheduling

...

A. Robinson. *Impact of Economics on Compiler Optimization.*

Motivation #2

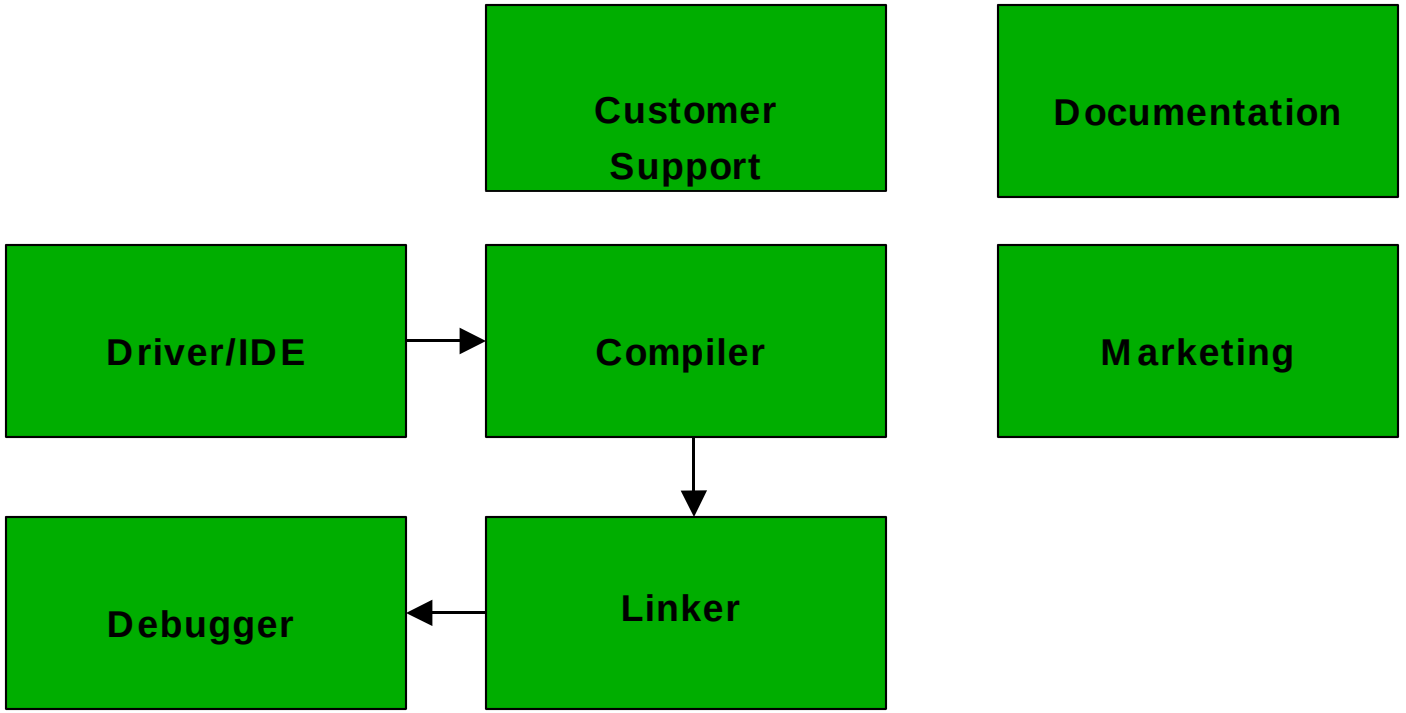
Niche Optimizations and Market Forces



A. Robinson. *Impact of Economics on Compiler Optimization.*

Motivation #2

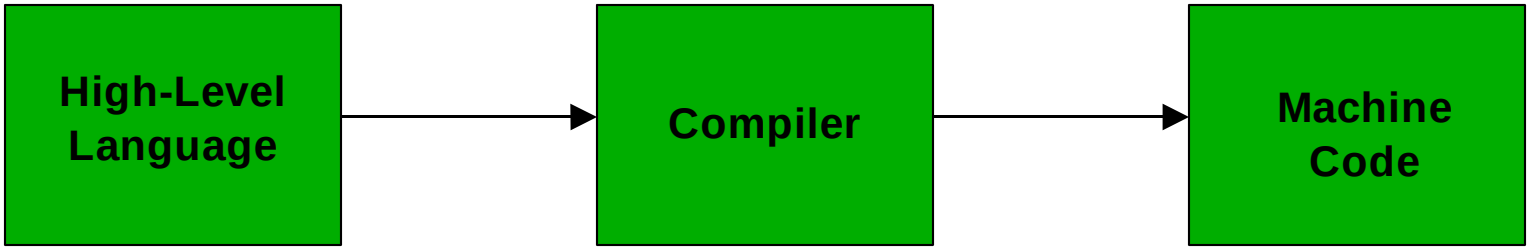
Niche Optimizations and Market Forces

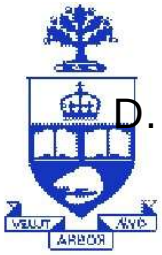


A. Robinson. *Impact of Economics on Compiler Optimization.*

Motivation #2

Niche Optimizations and Market Forces

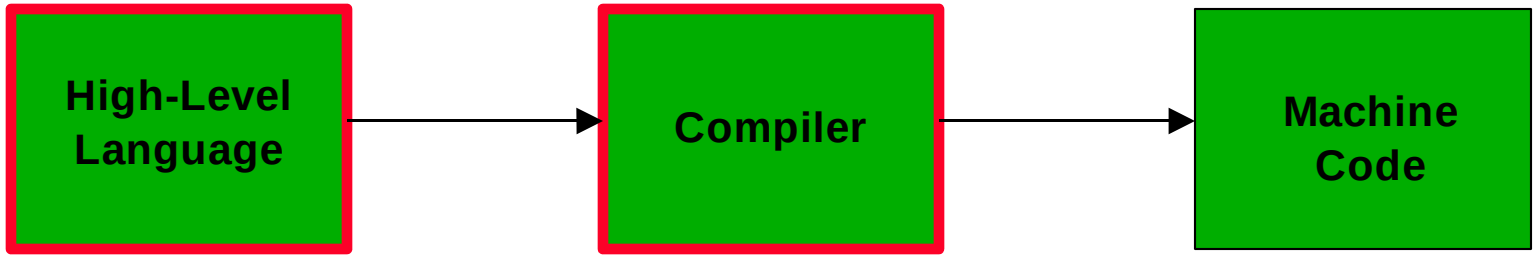


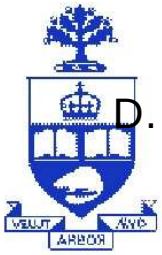


D. Engler. *Interface Compilation: Steps toward Compiling Interfaces as Languages.*

Motivation #2

Niche Optimizations and Market Forces





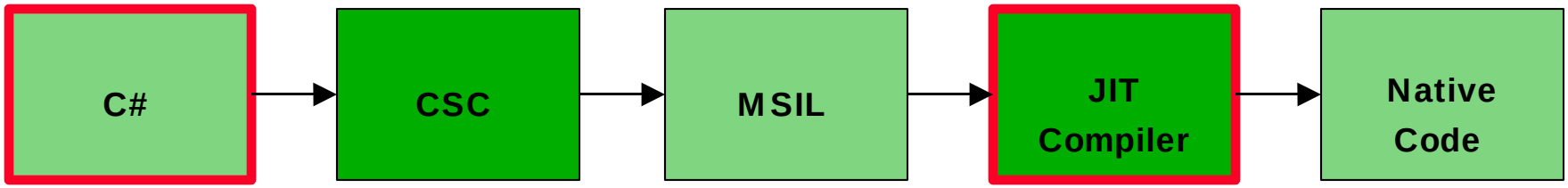
D. Engler. *Interface Compilation: Steps toward Compiling Interfaces as Languages.*

Motivation #3

Extensible JIT Compilation allows Self-Optimizing Libraries

Motivation #3

Extensible JIT Compilation allows Self-Optimizing Libraries



Library programmer can define optimizations in C# that plug into the JIT compiler

Presentation Outline

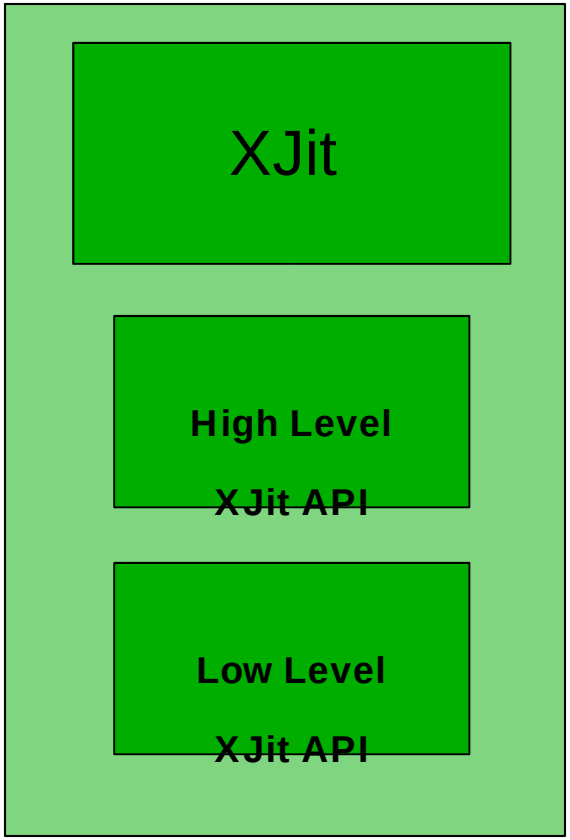
- XJit System Overview
- Example
- Results
- Conclusion

XJit System Overview

- .NET supports " attributes"
- Attributes can specify optimizer and trigger event
- Optimizer is written in C# using XJit API
- XJit recognizes attribute, detects, compiles, and invokes optimizer upon event

C# Program

```
...
[XJitOptimizeMe
(function, event)]
SomeLibraryFunc(..) {
...
}
...
```



(Mono open-source .NET framework)

Example

```
public static void Main()
{

   IntVector A, B, C, D, E;
   ...
   ...

   for( int i = 0; i < 10000; ++i ) {
       E = (A + B + C)/(D - A + B);
   }

   ...
   ...
}
```

Example

E = (A + B + C)/(D - A + B);

=

```
temp1 = A + B
temp2 = temp1 + C
temp3 = D – A
temp4 = temp3 / B
E = temp2 + temp4
```

Example

temp = A + B

=

```
public IntVector op_Addition(IntVector A, IntVectorB) {
      IntVector temp = new IntVector(A.length());
      for( i = 0; i < A.length(); i++ ) {
               temp(i) = A(i) + B(i);
      }
      return temp;
}
```

Example

IntVector Library Programmer detects simple optimization opportunity

Function Inlining + Loop Fusion + Use Integer Temps

```
Int temp1, temp2, temp3;
for( i = 0; i < A.length(); i++ ) {
        temp1 = A(i) + B(i);
        temp2 = temp1 + C(i);
        temp1 = D(i) – A(i);
        temp3 = temp1 + B(i);
        E(i) = temp3 / temp2;
}
```

Self Optimizing Libraries

```
[XJitOptimizeMeAttribute(specializer)]

static void MyLibraryMethod(int RunTimeConstant) {
        ...
        if( RunTimeConstant == ... ) {
                ...
        } else if( RunTimeConstant == ... ) {
                 ...
        } else if( RunTimeConstant == ... ) {
                 ...
         ...
        }
         ...
}
```

Self Optimizing Libraries

```
[XJitOptimizeMeAttribute(specializer)]
static void MyLibraryMethod(int RunTimeConstant) {
            ...
            if( RunTimeConstant == ... ) {
                        ...
            } else if( RunTimeConstant == ... ) {
                         ...
            } else if( RunTimeConstant == ... ) {
                         ...
             ...
            }
             ...
}
```

```
public void specializer(cfg) {
        ...
        XJitVar a = XJitGetVar(RunTimeConstant);

        /* modify cfg so it is specialized
         * for RunTimeConstant */
        ...
}
```

Self Optimizing Libraries

```
...
mini_method_compile(... , method,
        ... ) {
  ...
  if(HasOptimizeMeAttribute(method) {
    OptimizationPass
      = GetOptimizationPass(method);
    AddToPassList(OptimizationPass);
  }
  ...
  ...
  foreach(Pass in PassList ) {
    /* execute Optimization Pass */
  }
  ...
}
```

```
[XJitOptimizeMeAttribute(specializer)]
static void MyLibraryMethod(int RunTimeConstant) {
            ...
            if( RunTimeConstant == ... ) {
                        ...
            } else if( RunTimeConstant == ... ) {
                         ...
            } else if( RunTimeConstant == ... ) {
                         ...
             ...
            }
             ...
}


public void specializer(cfg) {
            ...
            XJitVar a = XJitGetVar(RunTimeConstant);

            /* modify cfg so it is specialized
             * for RunTimeConstant */
            ...
}
```

Self Optimizing Libraries

```
public static void Main()
 {

    IntVector A, B, C, D, E;
    ...
    ...

    for( int i = 0; i < 10000; ++i ) {
        E = (A + B + C)/(D - A + B);
    }

    ...
    ...
 }
```

Self Optimizing Libraries

```
public static void Main(){
   ...
   for( int i = 0; i < 10000; ++i ) {
        push A
        push B
        call op_Addition
        push C
        call op_Addition
        push A
        push D
        call op_Subtraction
        push B
        call op_Addition
        call op_Division
    }
    ...
}
```

Self Optimizing Libraries

```
[XJitOptimizeMyCallerAttribute(optimizer)]
public IntVector op_Addition(IntVector A,
   IntVectorB) {
   IntVector temp = new IntVector(A.length());
   for( i = 0; i < A.length(); i++ ) {
        temp(i) = A(i) + B(i);
   }
   return temp;
}
```

```
public static void Main(){
   ...
   for( int i = 0; i < 10000; ++i ) {
            push A
            push B
            call op_Addition
            push C
            call op_Addition
            push A
            push D
            call op_Subtraction
            push B
            call op_Addition
            call op_Division
    }
    ...
}
```

Self Optimizing Libraries

```
...
mini_method_compile(... , method,
        ... ) {
  ...
  if(HasOptimizeMeAttribute(method) {
    OptimizationPass
      = GetOptimizationPass(method);
    AddToPassList(OptimizationPass);
  }
  ...
  method_to_ir( ..., method, cfg, ... );
  ...
  foreach(Pass in PassList ) {
    /* execute Optimization Pass */
  }
  ...
}
```

```
public static void Main(){
   ...
   for( int i = 0; i < 10000; ++i ) {
            push A
            push B
            call op_Addition
            push C
            call op_Addition
            push A
            push D
            call op_Subtraction
            push B
            call op_Addition
            call op_Division
    }
    ...
}
[XJitOptimizeMyCallerAttribute(optimizer)]
public IntVector op_Addition(IntVector A,
     IntVectorB) {
     IntVector temp = new IntVector(A.length());
     for( i = 0; i < A.length(); i++ ) {
            temp(i) = A(i) + B(i);
     }
     return temp;
}
```

Self Optimizing Libraries

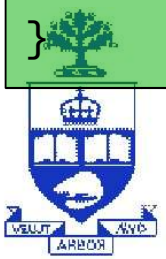
```
...
method_to_ir( ..., method, cfg, ... ) {
  ...
  if( inst == CALL ) {
    /* add to cfg */
    ...
    if(HasOptimizeMyCallerAttribute(
        GetMethod(inst)) {
        OptimizationPass =
          GetOptimizationPass(inst);
        AddToPassList(OptimizationPass);
  }

  ...
}
```

```
public static void Main(){
   ...
   for( int i = 0; i < 10000; ++i ) {
            push A
            push B
            call op_Addition
            push C
            call op_Addition
            push A
            push D
            call op_Subtraction
            push B
            call op_Addition
            call op_Division
    }
    ...
}
[XJitOptimizeMyCallerAttribute(optimizer)]
public IntVector op_Addition(IntVector A,
    IntVectorB) {
    IntVector temp = new IntVector(A.length());
    for( i = 0; i < A.length(); i++ ) {
           temp(i) = A(i) + B(i);
    }
    return temp;
}
```

Self Optimizing Libraries

```
...
mini_method_compile(... , method,
        ... ) {
  ...
  if(HasOptimizeMeAttribute(method) {
    OptimizationPass
      = GetOptimizationPass(method);
    AddToPassList(OptimizationPass);
  }
  ...
  method_to_ir( ..., method, cfg, ... );
  ...
  foreach(Pass in PassList ) {
    /* execute Optimization Pass */
  }
  ...
}
```



```
public static void Main(){
   ...
   for( int i = 0; i < 10000; ++i ) {
            push A
            push B
            call op_Addition
            push C
            call op_Addition
            push A
            push D
            call op_Subtraction
            push B
            call op_Addition
            call op_Division
    }
    ...
}
[XJitOptimizeMyCallerAttribute(optimizer)]
public IntVector op_Addition(IntVector A,
     IntVectorB) {
     IntVector temp = new IntVector(A.length());
     for( i = 0; i < A.length(); i++ ) {
            temp(i) = A(i) + B(i);
     }
     return temp;
}
```

Status (A Work in Progress)

- Detects OptimizeMe and OptimizeCaller
- Identifies plug-in optimizations
- Dynamically constructs pass list
- Invokes pass list
- Optimization API is in flux
    - High-level API (MSIL-like)
    - Low-level API (MONO-like)
- Small toy optimizer has been built
- Detection overheads studied

Results

- IntVector Example
  - Original Execution Time: 17 sec
  - New Execution Time: 6.7 sec
- Detection overhead “ C#Grande” virtually zero
  - No custom attributes
- Optimization overhead
  - Small and amortizable

Conclusion

- XJit: A Framework for Self-Optimizing Libraries
- Allows library writers to define plug-in optimizations
- User code is transparently optimized using plug-in optimization