# Recent Developments in Unicon

**Clint Jeffery**

**University of Idaho**

**jeffery@uidaho.edu**

joint work with

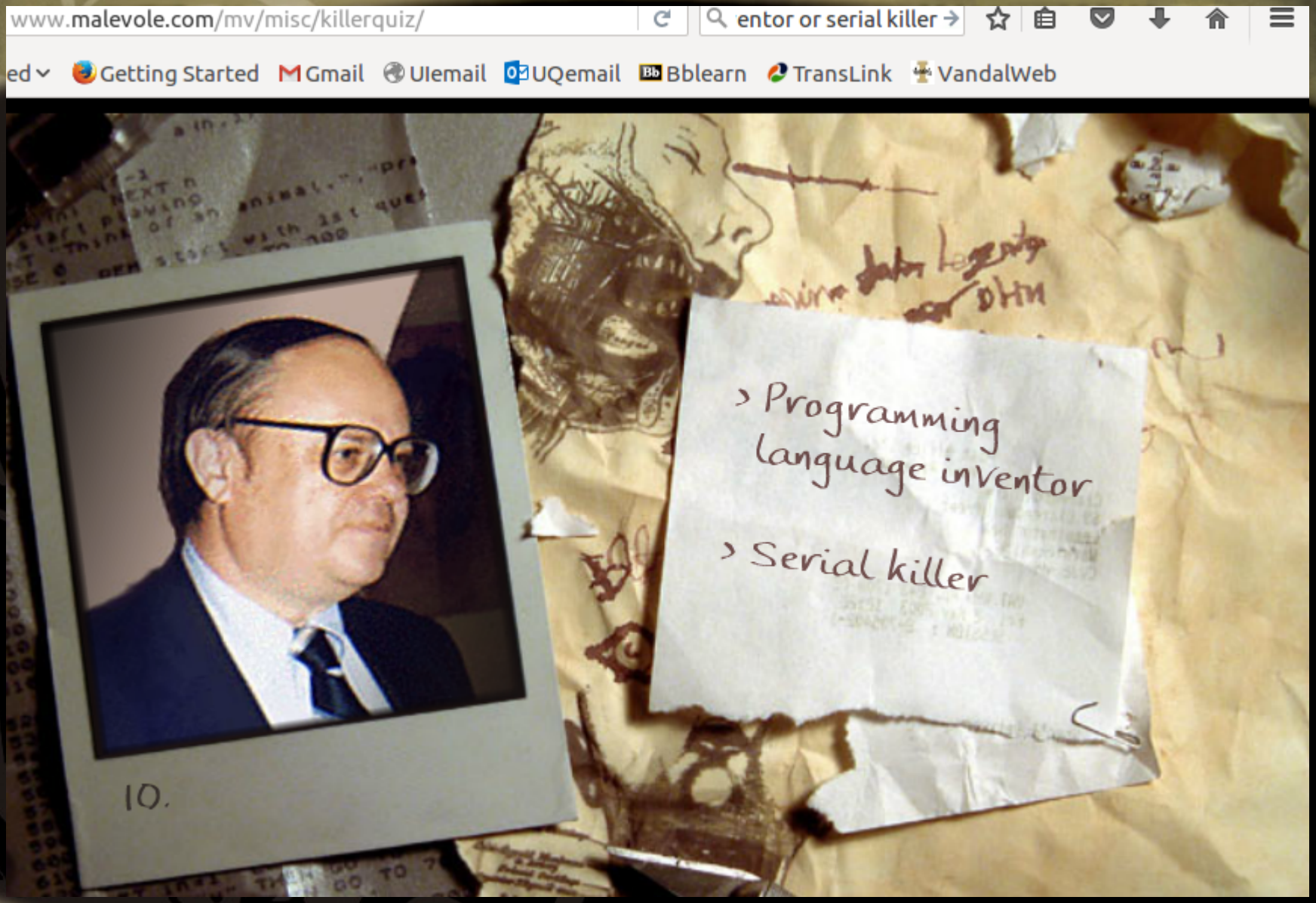
Peter Mills, John Goettsche, U Idaho

Phillip Thomas, U.S. NLM, NIH

Sudarshan Gaikaiwari, Yelp

# In This Talk

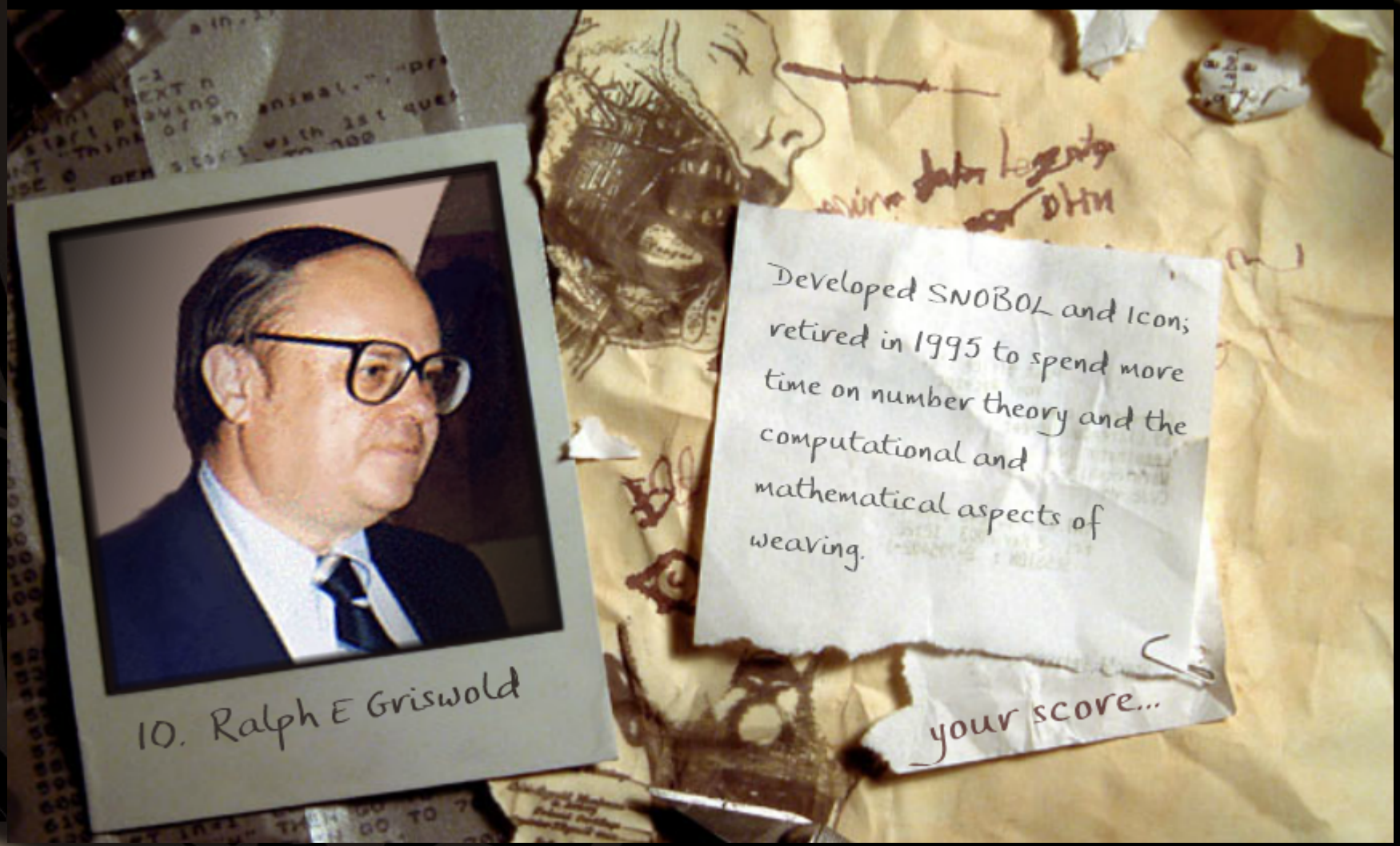
- Ramblings on mortality
- Seamless Language Embedding
- ~~Integrating SNOBOL patterns, regexes and string scanning~~


www.malevole.com/mv/misc/killerquiz/
entor or serial killer
ed
Getting Started
Gmail
UIemail
UQemail
Bblearn
TransLink
VandalWeb
10.
› Programming language inventor
› Serial killer

www.malevole.com/mv/misc/killerquiz/

entor or serial killer

ced Getting Started Gmail UIemail UQemail Bblearn TransLink VandalWeb



# Context

- SNOBOL (AT&T 1960's)
- ...evolved into Icon (Arizona 1980's)
- ...and now Unicon (unicon.org, 2000's)

- Main customers: NLM and AT&T

# Goal-directed Evaluation

- Success and failure
- Generators with backtracking search
- Expression evaluation allows anything to be a generator (that lazily produces a sequence of values until it fails)
- Compose  cross-product of operands

  (1 to 2) * (3 to 5)

  1*3 | 1*4 | 1*5 | 2*3 | 2*4 | 2*5
- Filter for successful results
  - (1 to 2) * isprime(3 to 5)

# Desire

- Preserve Griswold's ideas
- If not immortality, then at least longevity
- Not just the language, but the programs written in the language
- Running on the platforms users use

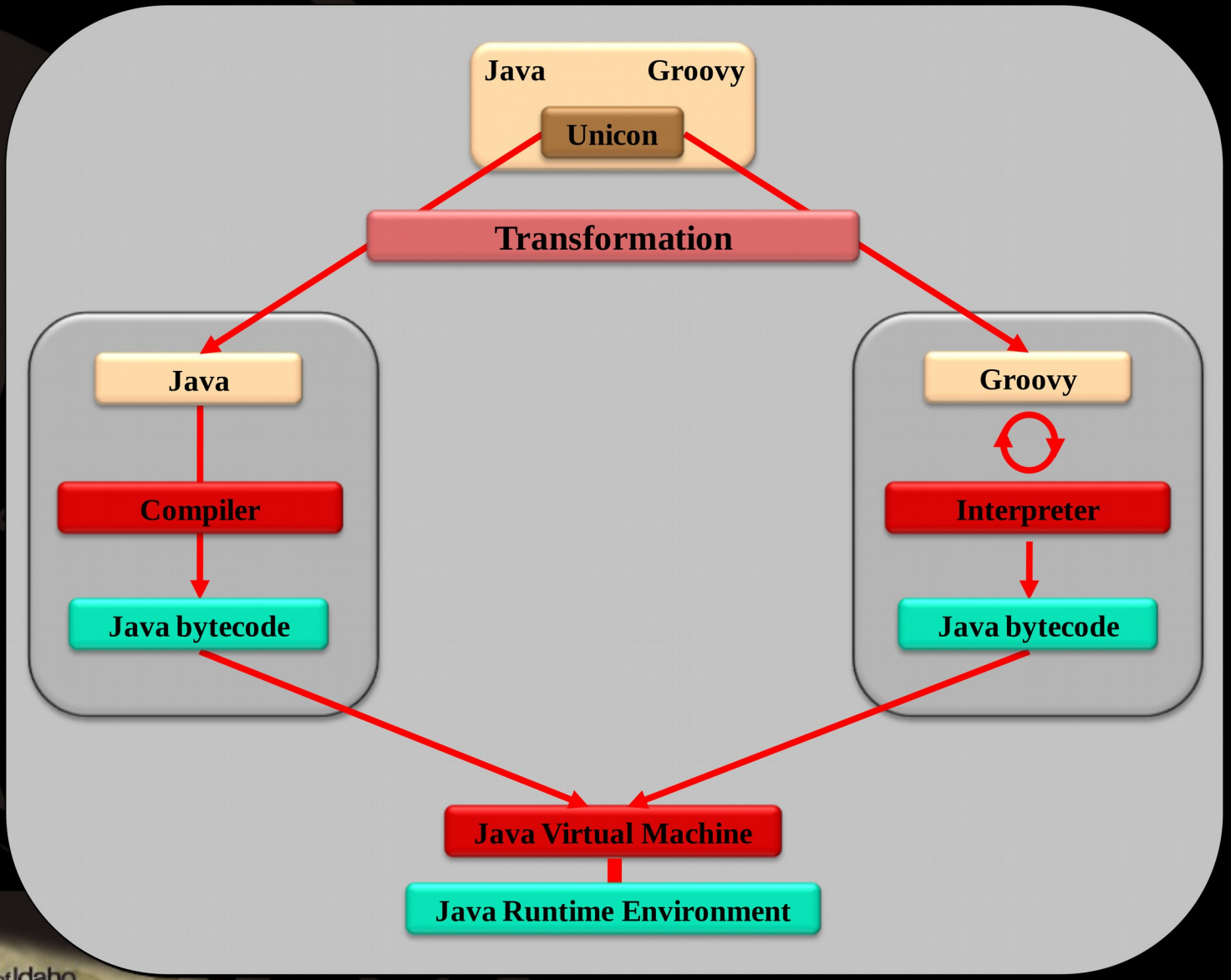
# Semi-Seamless Interoperation

- Use Unicon in Java and Java in Unicon
- Leverage Java portability, concurrency, graphics
- Avoid/solve problems found in other language embedding efforts

# Approach

- Embedding through transformation
- Unicon into Java, but fully nested
- Annotations delimit regions to be transformed
- Expression, method or class granularity
- Transforms unravel syntax to conventional Java form
- Generators and goal-direction become explicit iteration
- Targets: Java and Groovy

# Embedding through Transformation



# Mixed-Language Embedding

```
@<script lang="java">
public class PrimeMultiples {
    public void printAll (int lower, int upper) {
        for (Object i : primeMultiples(lower, upper)) {
            System.out.println(i.toString());
        }
    }

    @<script lang="junicon">
        def  primeMultiples(lower, upper) {
            suspend (lower to upper) * this::isprime(lower to upper);
        }
    @</script>

    public Object isprime (Object num) {
        VariadicFunction<Number,Iterator> generator =
        (VariadicFunction<Number,Iterator>)
        @<script lang="junicon">
            { (num) -> {  local i;
         return if not((i = 2 to Math::sqrt((Double) num))  & (num % i == 0)) then num else fail }
            };
        @</script>
        return generator.apply(((Number) num).doubleValue()).next();
    }
}
@</script>
```

**Scoped annotations**

- Transform and inject into surrounding context
- From innermost outwards

**Exact transforms depend on language types**

**Transforms leave foreign code unchanged**

# Transformation

- Unravel Unicon “normal-looking” syntax to its actual semantics
- What does f(e) really mean?

  (x in e) & (y in f(x))

  = for(x in e) { for(y in f(x)) { y } }
- Formalized as reduction semantics [Felleisen94]

# Transformation

- Translation of constructs and operators
- Maps constructs onto stream-like iterator calculus
- Implemented as small kernel for composing suspendable iterators
- Translation of classes generates duals for variables and methods: the plain Java variable/method and an updateable reference form of it.

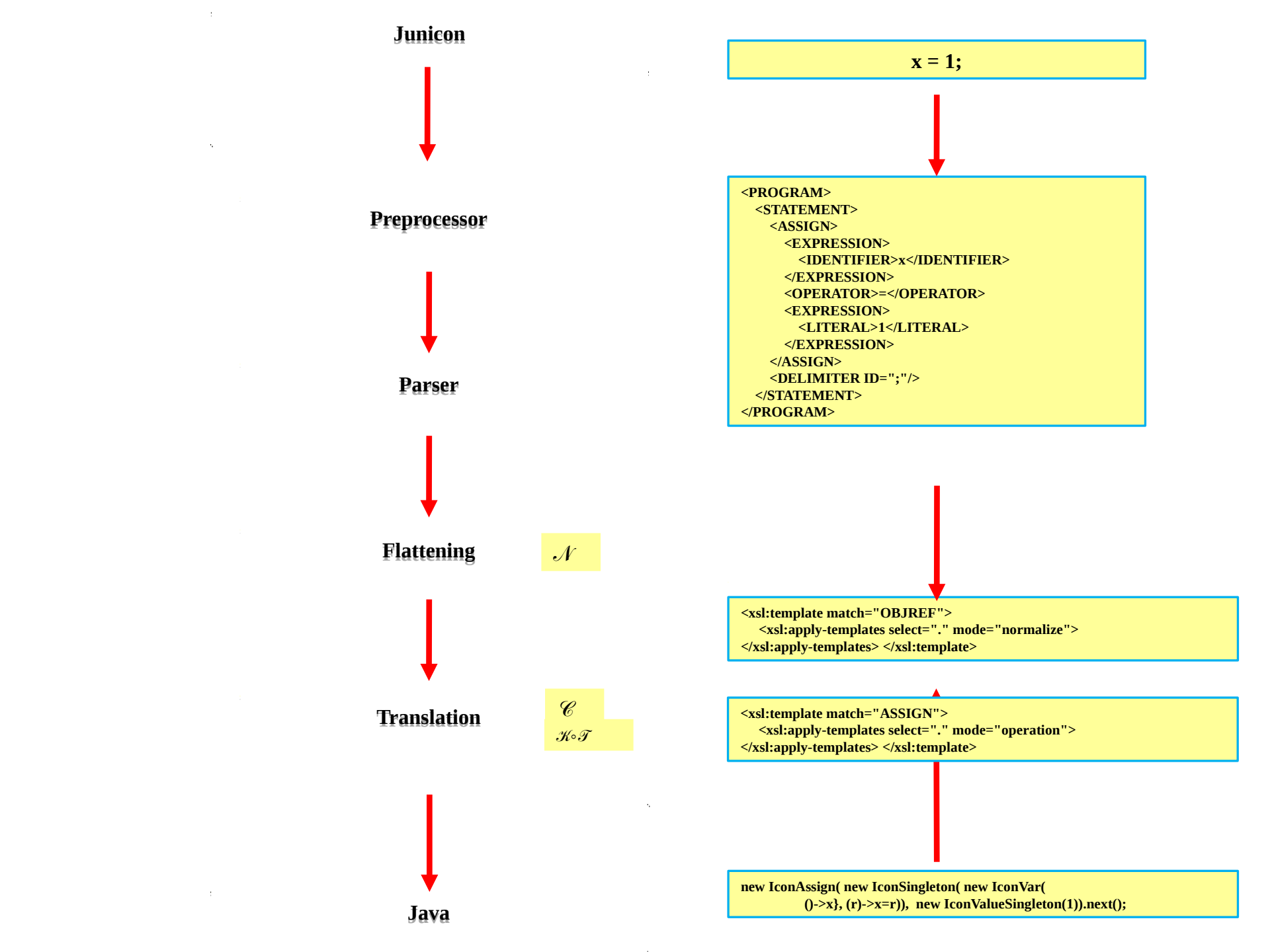
# Transformed Method

```
// Expose as method reference
public Object primeMultiples = (VariadicFunction) this::primeMultiples;

// Define variadic method that returns an iterator
public Iterator primeMultiples (Object... args) {
        // Reuse method body
        IconIterator body = methodCache.getFree("primeMultiples_m");
        if (body != null) { return body.reset().unpackArgs(args); };
        // Reified parameters
        IconVar lower_r = new IconVar().local();
        IconVar upper_r = new IconVar().local();
        // Temporaries
        IconTmp x_0_r = new IconTmp();
        // Unpack parameters
        VariadicFunction unpack = (Object... params) -> {
                if (params ==  null) { params = EmptyArray; };
                lower_r.set((params.length > 0) ? params[0] : null);
                upper_r.set((params.length > 1) ? params[1] : null);
                return null;
        };
        // Method body
        body = new IconSequence(new IconSuspend(new IconOperation(IconOperators.times).over((new IconToIterator(lower_r, uppe
        new IconProduct(new IconIn(x_0_r, new IconToIterator(lower_r, upper_r)), new IconInvokeIterator(()-> this.isprime(x_0_r.deref(
        new IconNullIterator(), new IconFail());
        // Return body after unpacking arguments
        body.setCache(methodCache, "primeMultiples_m");
        body.setUnpackClosure(unpack).unpackArgs(args);
        return body;
}
```

# Junicon Implementation

- XSLT
- Spring dependency injection
- Full interactive interpreter (Groovy) as well as compiled output (Java)
- Windows or Linux standalone executables


Junicon
x = 1;
Preprocessor
<PROGRAM>
<STATEMENT>
<ASSIGN>
<EXPRESSION>
<IDENTIFIER>x</IDENTIFIER>
</EXPRESSION>
<OPERATOR>=</OPERATOR>
<EXPRESSION>
<LITERAL>1</LITERAL>
</EXPRESSION>
</ASSIGN>
<DELIMITER ID=";"/>
</STATEMENT>
</PROGRAM>
Parser
Flattening
<xsl:template match="OBJREF">
<xsl:apply-templates select="." mode="normalize">
</xsl:apply-templates> </xsl:template>
Translation
<xsl:template match="ASSIGN">
<xsl:apply-templates select="." mode="operation">
</xsl:apply-templates> </xsl:template>
Java
new IconAssign( new IconSingleton( new IconVar(
()->x}, (r)->x=r)), new IconValueSingleton(1)).next();

# Junicon Status

- Real, on sourceforge
- Execution from 3x to 0.3x of Unicon's normal speed, on average maybe 0.6x
- Implementation of transformation-based compiler/interpreter is very small, but dense/deep
- Runtime system addition relatively simpler than in the C implementation

# Junicon Future Work

- Current work is to complete runtime system, particularly graphics
- LibGDX target, will be first phone/tablet implementation of Unicon
- Can we achieve this (or better) level of embedding for the C implementation?
- What would “fully seamless” language interoperation look like?