Do you speak-a my Language?



I HATE CTT

I HATE CTT

HATE

I HATE CTT

HATE

HATE

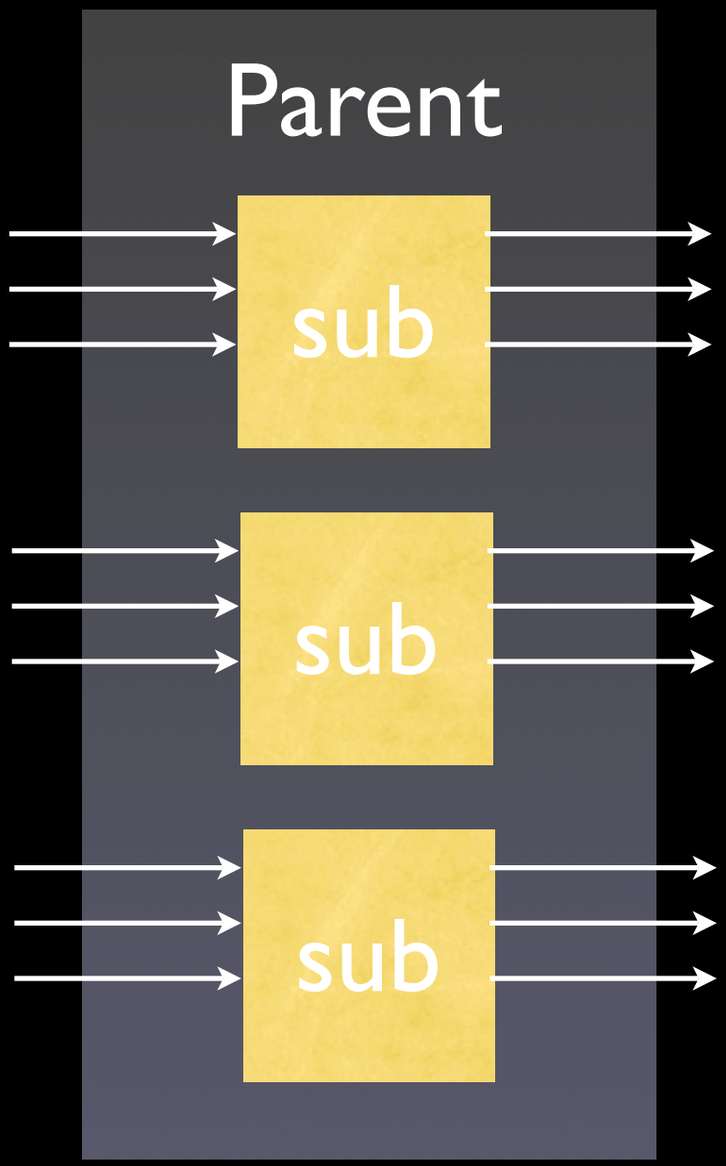
I HATE CTT

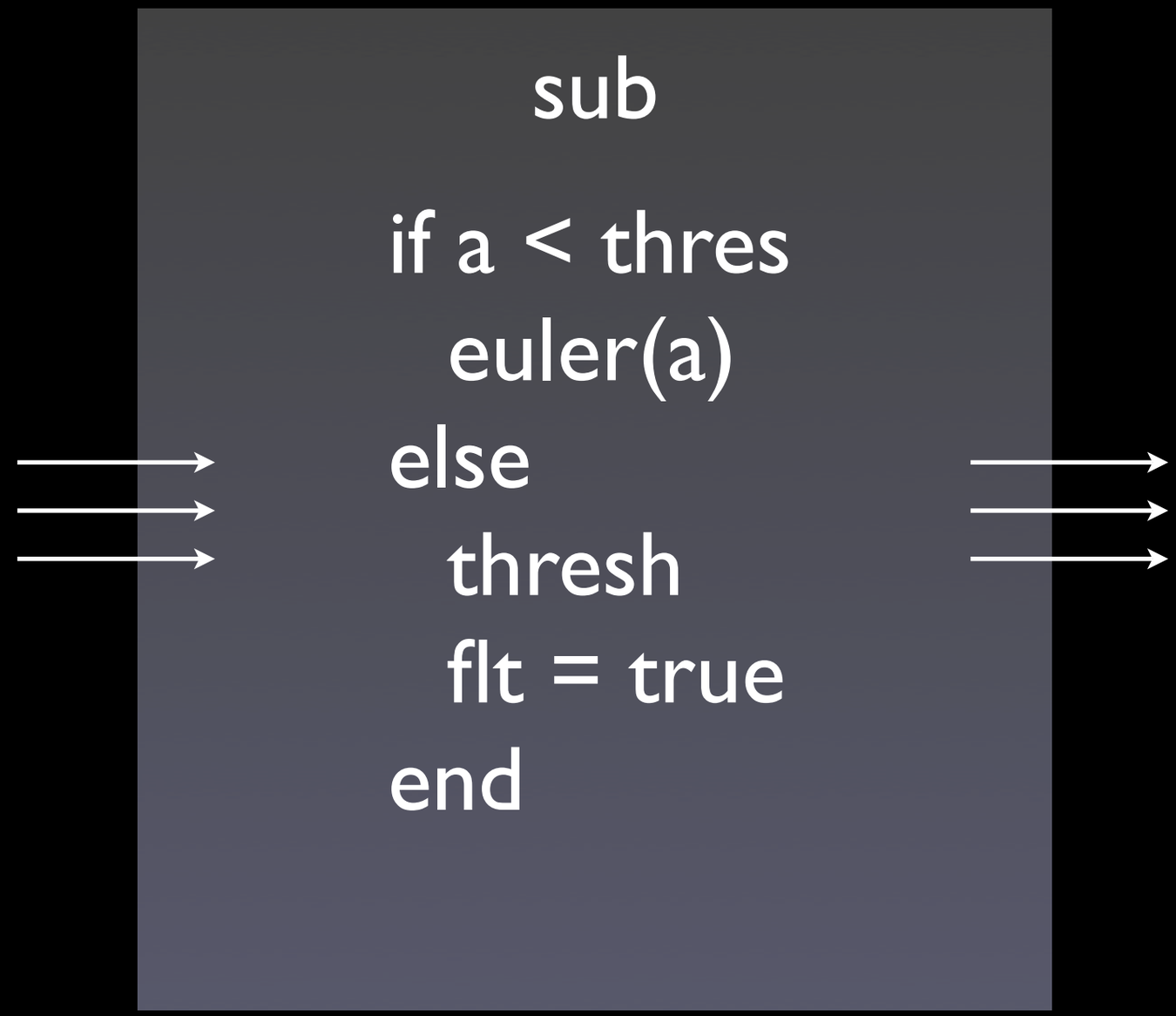
HATE

HATE

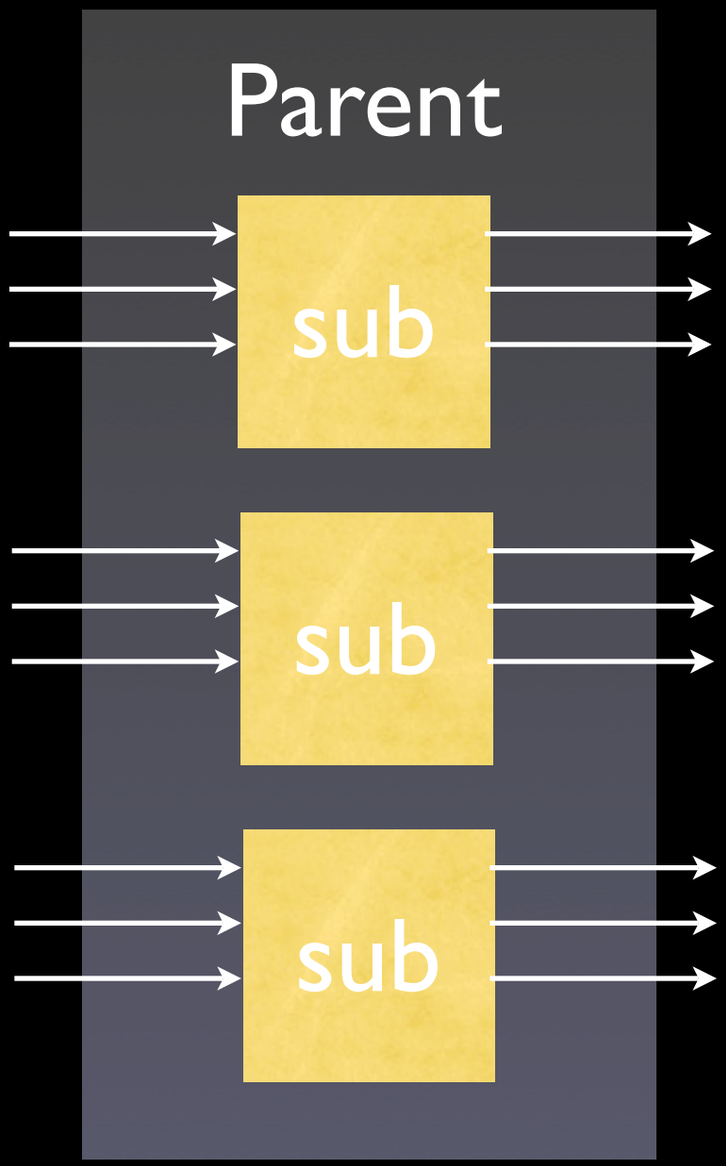
HATE

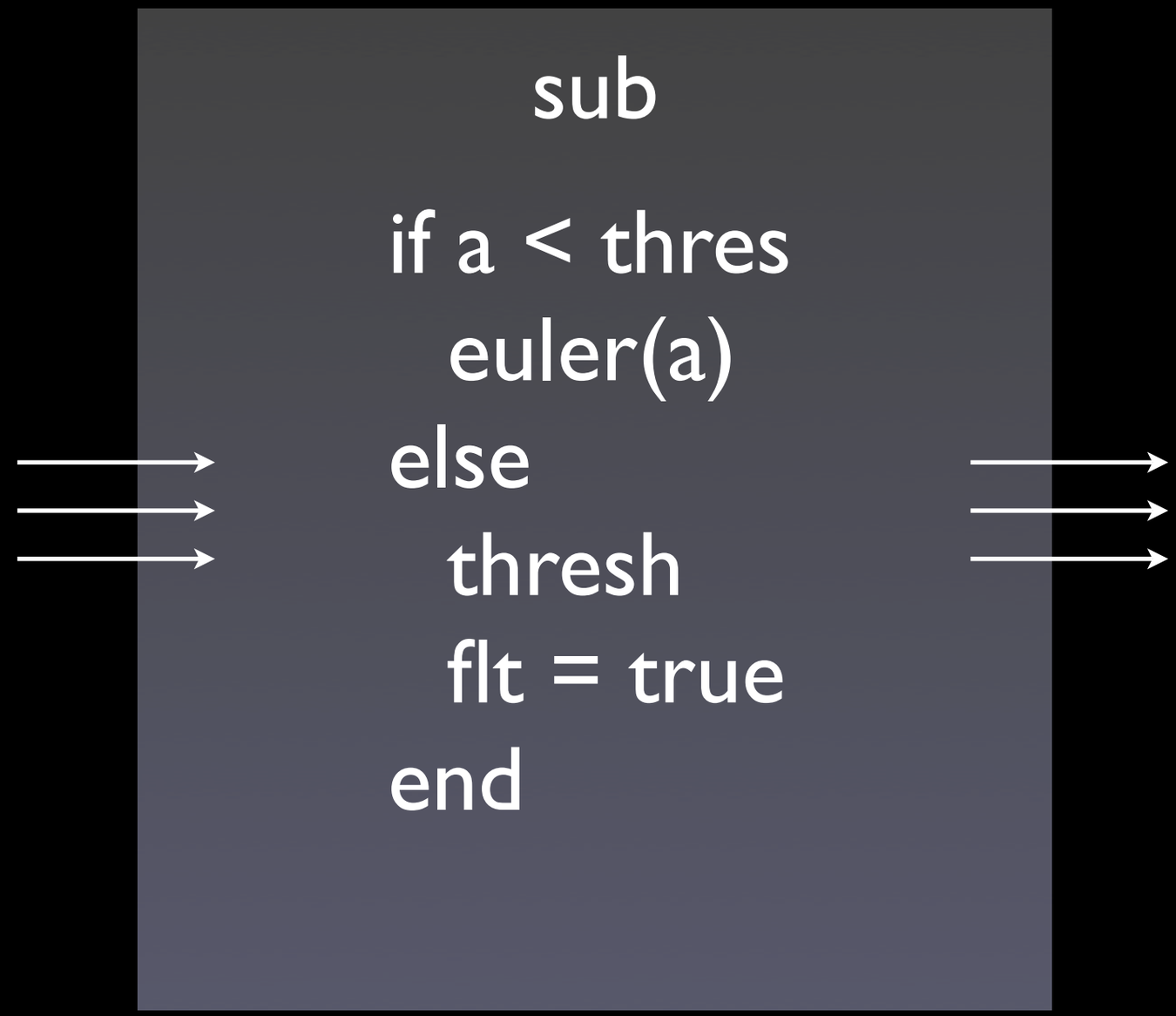

HATE
LOVE
ECSTASY
FREEDOM


Parent
sub
sub
sub


sub
if a < thres
euler(a)
else
thresh
flt = true
end

```
SYMBOL flt_in COMPUTED BOOLEAN := flt'IN;
SYMBOL flt_out COMPUTED BOOLEAN :=
  if a < thresh then
    flt_in
  else
    true
end;
SYMBOL pass COMPUTED BOOLEAN :=
  flt'OUT == flt_out;
```

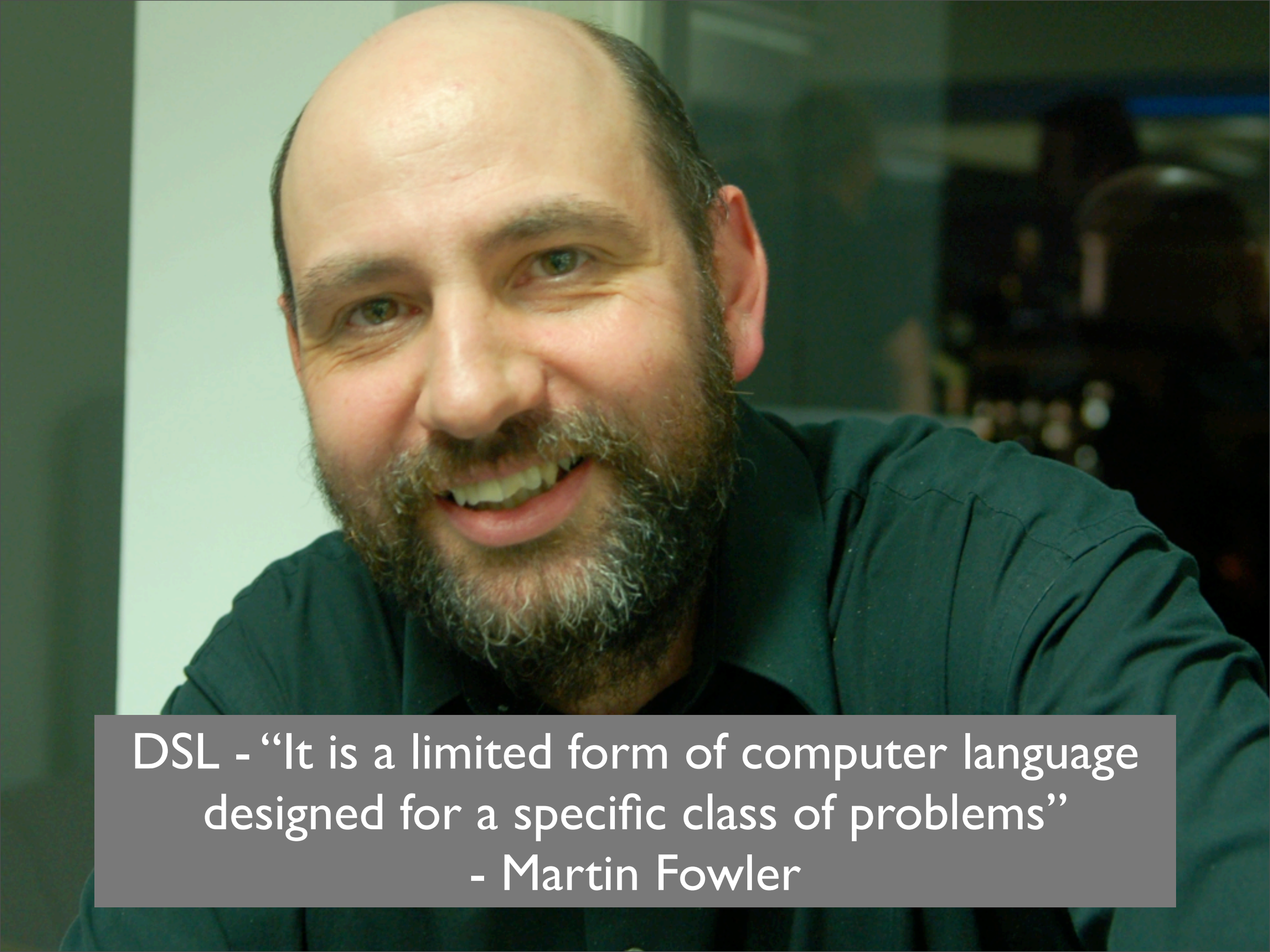

Parent
sub
sub
sub


sub
if a < thres
euler(a)
else
thresh
flt = true
end


WIN

DSL - noun, Geeks geeking out and making it harder than it needs to be.

- your manager



DSL - “It is a limited form of computer language designed for a specific class of problems”
- Martin Fowler



External vs Internal

External

```
clean:
    rm -r *.o
    rm -r my_bin
```

```
clean:
    rm -r *.o
    rm -r my_bin
```

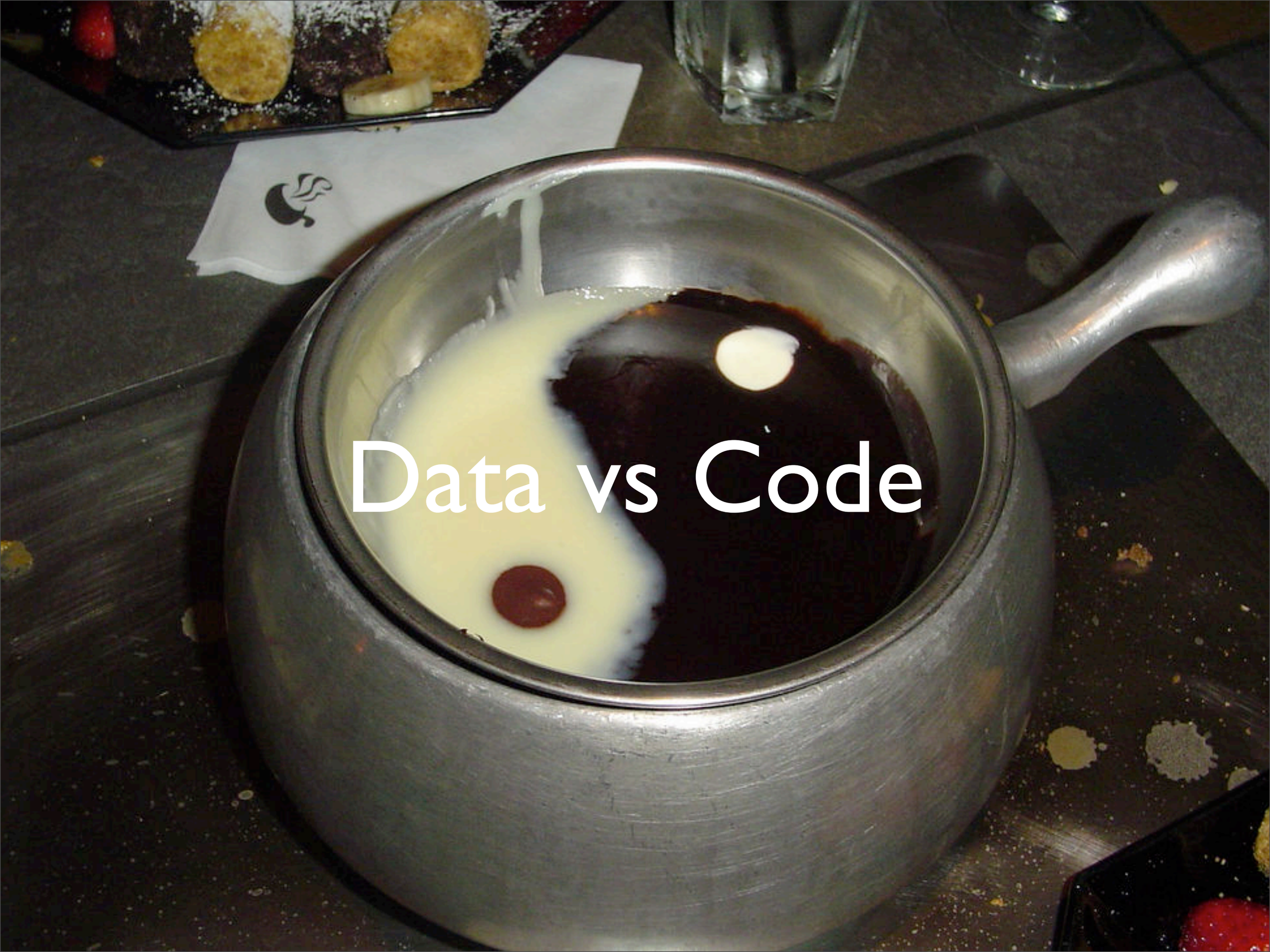
```
app: *.c
    gcc -o my_bin app.c lib.c
```

```
employee John Doe
compensate 500 dollars for each deal closed in
  the past 30 days
compensate 100 dollars for each active deal that closed
  more than 365 days ago
compensate 5 percent of gross profits if gross profits
  are greater than 1,000,000 dollars
compensate 3 percent of gross profits if gross profits
  are greater than 2,000,000 dollars
compensate 1 percent of gross profits if gross profits
  are greater than 3,000,000 dollars
```

Internal

```
Assert.That( value, Is.Not.Null );
```

```
[<Scenario>]
let When_calculating_fac_5_it_should_equal_120() =
  Given 5
  |> When calculating factorial
  |> It should equal 120
  |> Verify
```


Data vs Code

Code

```
grammar Expr;
prog:   stat+
stat:     expr NEWLINE
      |     ID '=' expr NEWLINE
      |     NEWLINE;
expr:   multExpr (('+' | '-') multExpr*;
multExpr:   atom ('*' atom)*;
atom:   INT
          |  ID
          |  '(' expr ')';

ID: ('a'..'z'|'A'..'Z')+;
INT: '0'..'9'+;
NEWLINE: '\r'? '\n';
WS: (' '|'\t'|'\n'|'\r')+ { skip(); };
```

Data

```
application: comic_site
version: 1
runtime: python
api_version: 1

handlers:
  - url: /
    script: welcome.py
  - url: /admin/.*
    script: admin.py
    login: admin
```

Terse vs Verbose

Terse

```
map.connect 'articles/:year/:month/:day',
  :controller => 'articles',
  :action     => 'find_by_date',
  :year       => /\d{4}/,
  :month      => /\d{1,2}/,
  :day        => /\d{1,2}/
```

Verbose

```
describe Bowling, "When scoring gutters" do
  it "should score 0 for gutter game" do
    bowling = Bowling.new
    20.times { bowling.hit(0) }

    bowling.score.should == 0
  end
end
```

"All problems in computer science can be solved by another level of indirection"

- David Wheeler

"...except for the problem of too many layers of indirection."

- Kelvin Henney


power
performance
in reserve
Sounds good.
Where do I sign up?

Tools



Your Language

C#

```
Id( x => x.Id );
Map( x => x.Name);
Map( x => x.Price);
HasManyToMany( x => x.StoresStockedIn)
  .Cascade.All()
  .Inverse()
  .WithTableName("StoreProduct");
```

Ruby

```
plot = Plot.new
plot.add Circle.new [2,1], 5
plot.add Polygon.new [[2,1], [2,3], [1,1]]
```

```
Plot.new do
  @regions << Circle.new [2,1], 5
  @regions << Polygon.new [[2,1], [2,3], [1,1]]
end
```

```
Plot.new do
  circle :center => [2, 1], :radius => 5
  polygon [[1,1], [2,1], [2,3]]
end
```

Treetop

```
circle [1,2] 5
polygon [[1,2], [3,4], [5,4]]
rectangle [3,3] width=5 height=1
```

```
grammar Shapes
  rule shapes
    shape+
  end
  rule shape
    circle / rectangle / polygon
  end
  rule ws
    (' ')+
  end
  rule circle
    'circle' ws center:point ws 'radius=' radius:[0-9]+
  end
  rule rectangle
    'rectangle' ws start:point ws 'width=' width:[0-9]+ ws 'height=' height:[0-9]+
  end
  rule point
    '[' x:[0-9]+ ',' y:[0-9]+ ']'
  end
  rule polygon
    'polygon' ws '[' first_point:point other_points:(', ' a_point:point)* ']' {
      def points
        [first_point] + rest_points
      end
      def rest_points
        other_points.elements.map {|comma_and_point| comma_and_point.a_point }
      end
    }
  end
end
```

```
require 'rubygems'
require 'treetop'
require 'shapes'

parser = ShapesParser.new
thing = parser.parse("circle [1,2] radius=5").elements.first
puts thing
puts thing.center.x.text_value
puts thing.center.y.text_value
puts thing.radius.text_value

thing = parser.parse("rectangle [5,4] width=2 height=3").elements.first
puts thing
puts thing.start.x.text_value
puts thing.start.y.text_value
puts thing.width.text_value
puts thing.height.text_value

thing = parser.parse("polygon [[4,3], [2,3], [1,5]]").elements.first
puts thing
puts thing.points.map{ |e| e.text_value }
```

Irony

```
public ShapesGrammar()
{
    // Terminals
    var number = new NumberLiteral("number");

    // Non-Terminals
    var point           = new NonTerminal("point", CreatePointNode);
    var pointList       = new NonTerminal("pointList");
    var pointCollection = new NonTerminal("pointCollection", CreatePointCollection);
    var circle          = new NonTerminal("circle");
    var polygon         = new NonTerminal("polygon", CreatePolygon);
    var rectangle       = new NonTerminal("rectangle");
    var shape           = new NonTerminal("shape");
    var shapes          = new NonTerminal("shapes");

    // BNF Rules
    point.Rule           = "[" + number + "," + number + "]";
    pointList.Rule       = MakePlusRule(pointList, Symbol(",") , point);
    pointCollection.Rule = "[" + pointList + "]";
    polygon.Rule         = "polygon" + pointCollection;
    circle.Rule          = "circle" + point + "radius" + "=" + number;
    rectangle.Rule       = "rectangle" + point + "height" + "=" + number + "width" + "=" + number;
    shape.Rule           = rectangle | circle | polygon;
    shapes.Rule          = MakePlusRule(shapes, NewLine, shape);

    this.Root = shapes;

    RegisterPunctuation("[", "]", "=");
}
```

Antlr

```
age = 4
myvar = 8 - 4 * (age + 3)
```

```
grammar Expr;
prog:   stat+
stat:     expr NEWLINE
      |     ID '=' expr NEWLINE
      |     NEWLINE;
expr:   multExpr (('+' | '-') multExpr*;
multExpr:   atom ('*' atom)*;
atom:   INT
          |  ID
          |  '(' expr ')';

ID: ('a'..'z'|'A'..'Z')+;
INT: '0'..'9'+;
NEWLINE: '\r'? '\n';
WS: (' '|'\t'|'\n'|'\r')+ { skip(); };
```

MGrammar

Chris is 24 years old.
Pat is 32 years old.
Billy is 3 years old.
Granny is 98 years old.

```
module LangNet {
  language Contacts {
    syntax Main = p:Person*
          => People { valuesof(p) };
    syntax Person =
          n:Name "is" a:Age "years" "old"
          => { Name = n, Age = a };
    token Age = '0'..'9'+;
    token Name = ('A'..'Z' | 'a'..'z')+;
    interleave Whitespace = ' ' | '\r' | '\n' | '\n\r';
  }
}
```

Meta Programming System

Lex & Yacc

Roll Your Own

Technique

Define Your Purpose

Define Your Structure

Define Your Language

Implement Your Language

Implement Your Transformation

Rinse and Repeat


ROYAL OAK
RITUAL PRESENTS
FLOGGING MOLLY
TONITE 7 PM
TICKETS COM

Why?

Model

Schedule

Diagrams

Tests

Initial Conditions

Diagram Instances

Other Symbols

Explicit Assertions

Schedule

```
@sched = Schedule.new do
    project :EngineProgram
    netlist :Module, :Function
    ...
end

puts @sched
```

```
@sched = Schedule.new do |s|
  s.project :EngineProgram
  s.netlist :Module, :Function
  ...
end

puts @sched
```

```
class Schedule
  attr_writer :project
  def initialize(&block)
    instance_eval( &block )
  end
  def netlist(netlist, diagram)
    @netlist = netlist
    @diagram = diagram
  end
  def test(number, &block)
    @tests ||= []
    @tests << Test.new(number,&block)
  end
  def namespaces
    [@netlist, "Globals"].join(", ")
  end
  def to_s
"
Project #{@project};
Diagram #{@netlist}.#{@diagram};

Use #{namespaces};

#{@tests.join("\n")}

-- End Schedule
"
  end
end
```

Test

```
test 10, "describe why this test exists" do
    inputs :var_a => 0,
           :var_b => [true, false]
    outputs :some_bit => true
    inouts :xVar_aState => [0,2]
    ...
end
```

```
float :max, "IF (a > b) THEN a ELSE b END"
```

```
component :switch do
  input :selector, :BOOLEAN
  input :if_t, :FLOAT
  input :if_f, :FLOAT
  output :out, :FLOAT,
     "IF (selector) THEN if_t ELSE if_f END"
end
```

```
vs = instance :switch, :valid_signal,
  :selector => :sig_fault,
  :if_t => :signal_default,
  :if_f => :signal_raw
```

```
assert :sig_out, vs.d
assert :sig_out, vs.d, "0.004"
```

But I left

Wishful Thinking