Learning to see the tiger

(Insert scary nonstandard disclaimer of your choice here)

Many ruby programmers use ruby as if it were a fancy BASIC; they don't realize they are riding a tiger.

But they are.

Consider this simple patch...

Completing the square

We can write things like:

```
9.divmod 2
```

and even

```
class Fixnum
    def divmod(other)
        Pair.new(super other)
        end
    end
```

Or even

```
class Fixnum
    def apples
        [:mac]*self
        end
    end
```

And then use it:

```
9.apples
```

And we can write things like:

```
9 / 2
```

and even:

```
class Fixnum
    def /(other)
        Ratio.new(divmod.other)
        end
    end
```

But we can't write things like:

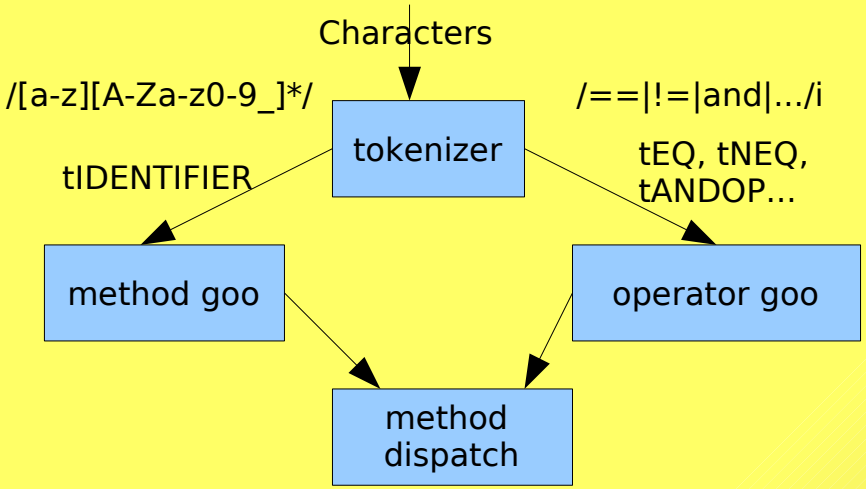
```
class Fixnum
    def |+>(other)
        Ket_sum.new(self,other)
        end
    end
```

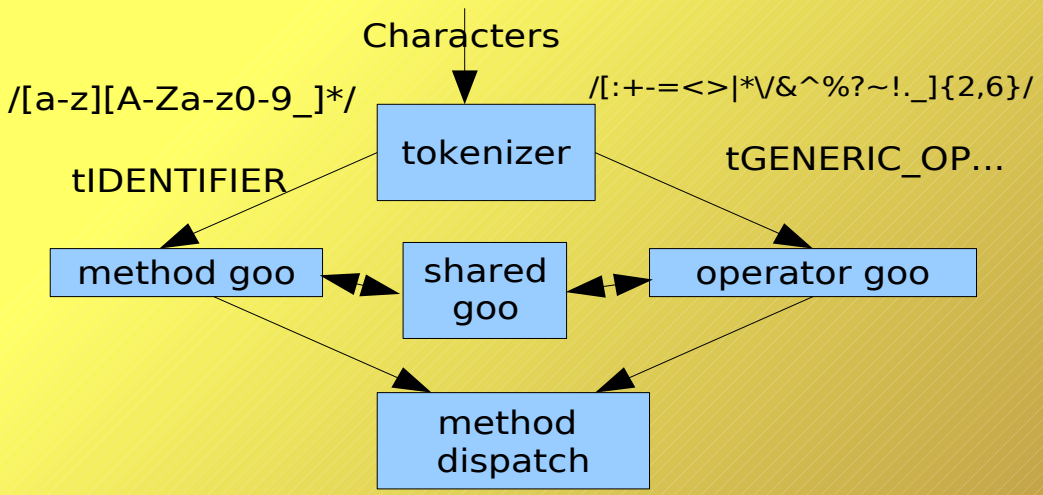
And then use:

```
9 |+> 2
```

A grossly oversimplified picture

that may still be too detailed





The real ruby compiler treats all identifiers uniformly, but special cases the operators.

Instead we could treat operators more analogously to identifiers.

This actually simplifies some parts of the compiler, in a sort of DRYer than it maybe should be way.

Consider this patch...
(2nd Disclamer)

The patch lets you write things like:

```
def ++/--?(foo)
    ...
    end

print “oopsy!” unless a ++/--? b
```

Matz considered it, and said “thanks, but no thanks”

Quite a few people tried it, but no one (to my knowledge) ever used it in production.

And frankly, I don't blame them.

What could we do with this?

Clever little IDLs / SPILs:

A whistle & siren specifier:

```
(C _# --> G)/2.seconds & ( G --> C)/2.seconds
```

Range and Hash analogs:

OrderedHashes

```
:key1=>>“value1” || :key2=>>“val2” || :k3=>>“foo”
```

Fancy ranges:

0 <=..< 7

x ..<+ 20

Executable serialization formats for something YAML-oid

Asynchronous method calls al la Dramatis

How we could do this sort of thing

```
class Object
    def =>>(val)
        result = OrderedHash.new
        result[self] = val
        result
        end
    end

class OrderedHash
    def ||(other)
        update(other)
        end
    # Ordered Hash goodness goes here
    :
```

What trouble could we get into?

Built-ins are optimized—and we might be less inclined to use them, making ruby appear slower.

Ambiguous decomposition. When someone writes "`x &&! y`" do they mean to use some new `&&!` operator or do they mean "`x && !y`"?

Precedence (and associativity) would have to be global, but it's not clear how these should be assigned. Oh my dear aunt Sally!

Ruby might become a line noise language.

Some people fear that conflicting uses would arise. This problem isn't really any worse than we already face with method names.

Pushing these ideas a little further
(i.e. too far)

Pushing the ideas behind the patch further we could...

...eliminate the lexically based distinction between operators and methods and simplify the compiler even further

...allow unicode characters in operators (dot and cross product, anyone?)

...syntactically unify blocks and either Procs or lambdas with method/operators as well (so that the same “{|...| ...}” syntax produced a block or either a Proc or a lambda depending on context

...and then it's just a short step to having (gack!) anonymous operators too:

```
:fry {|a,b| do_it_to_it(a,b) } :fish
```

This sort of unification is like eating peanuts (candied peanuts, with lots of syntactic sugar on them).

What, that didn't take well over five minutes?

The interesting thing is that free form operators are just syntactic sugar. The don't change the semantics of ruby one wit.

If you find any aspect of them disturbing, just remember *you're already riding that tiger* even if you don't normally look at it.