hello

@tjjjwxzq

All I'd Wanted to Know About

`Ruby's Object Model`

Starting Out

...And

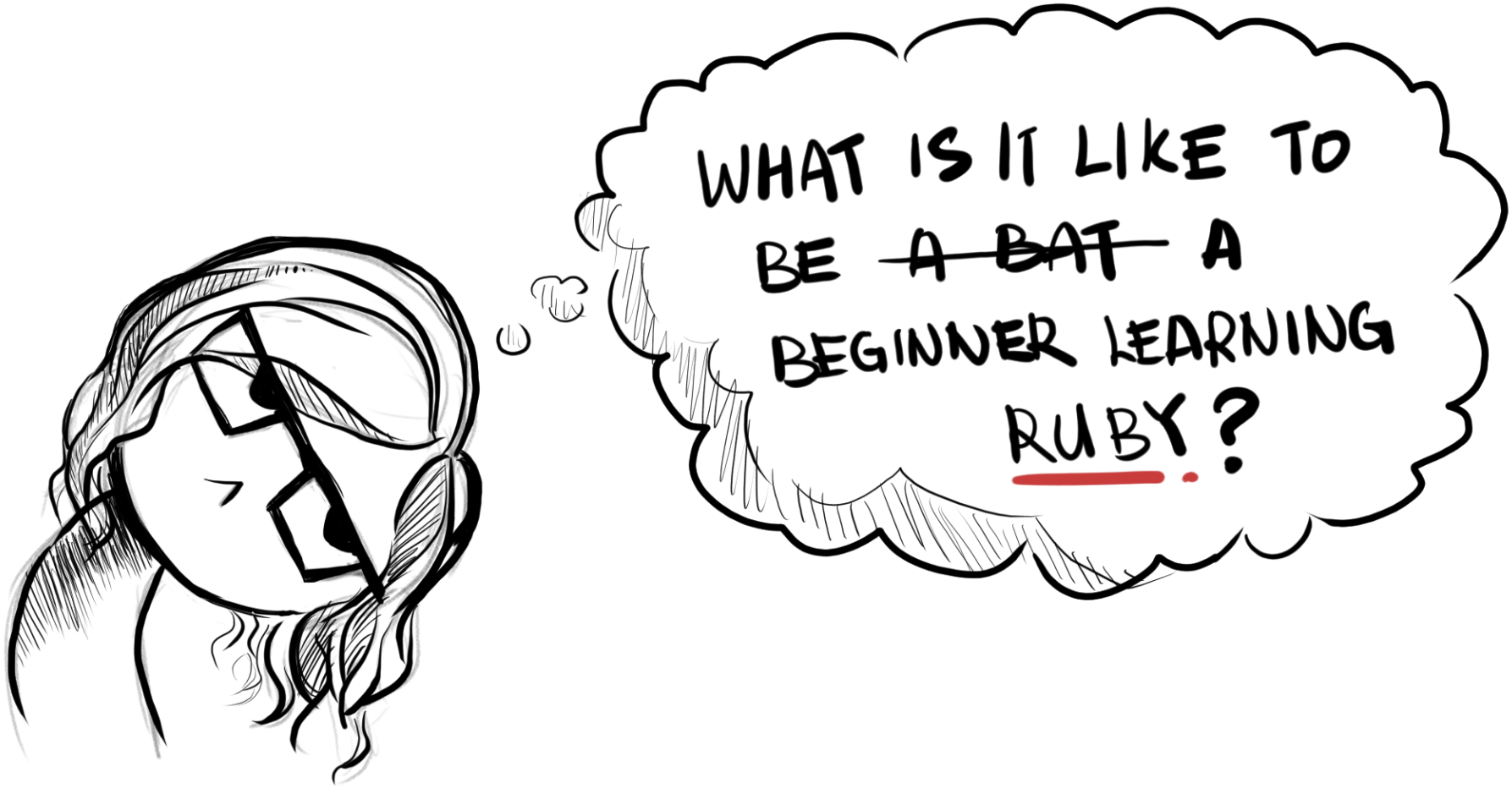
Moooarr!!!


WAIII?

YAEYYY

FREE RDRC TICKETS!!!!!!!


I AM BUT A YOUNG PADAWAN

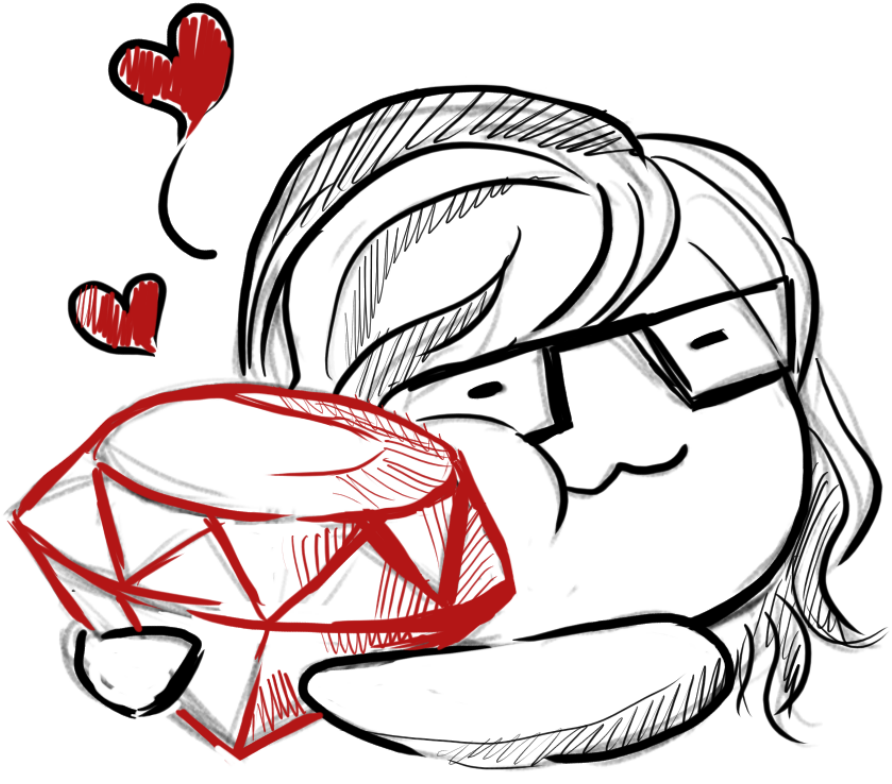

WHAT IS IT LIKE TO BE ~~A BAT~~ A BEGINNER LEARNING RUBY?


SUCH RUBY
SO BEAUTY

```
stuff = %w(
  breathe read wonder introspect
  learn code write improve marvel
  laugh cry connect love sing draw
  teach create receive give rejoice
  struggle remember forget suffer
  paint wander ramble believe live
)

while 'life' do
  stuff.sample
end
```

```ruby
def turn_your_block_into_a_shiny_proc(&block)
    puts "Did I mention that procs are kinda tricksy?"
    block.call
end

turn_your_block_into_a_shiny_proc do |tricksy|
    puts "I #{tricksy ? "am" : "am not"} tricksy"
end
```

```ruby
class DangerousThing
    def consume
        [:increase_insurance_premium]
    end
end

class Cupcake < DangerousThing
    include Icing

    def consume
        super + (add_icing << :lots_of_calories)
    end
end

module Icing
    def add_icing
        [:moar_calories]
    end
end
```




class_eval

default definee

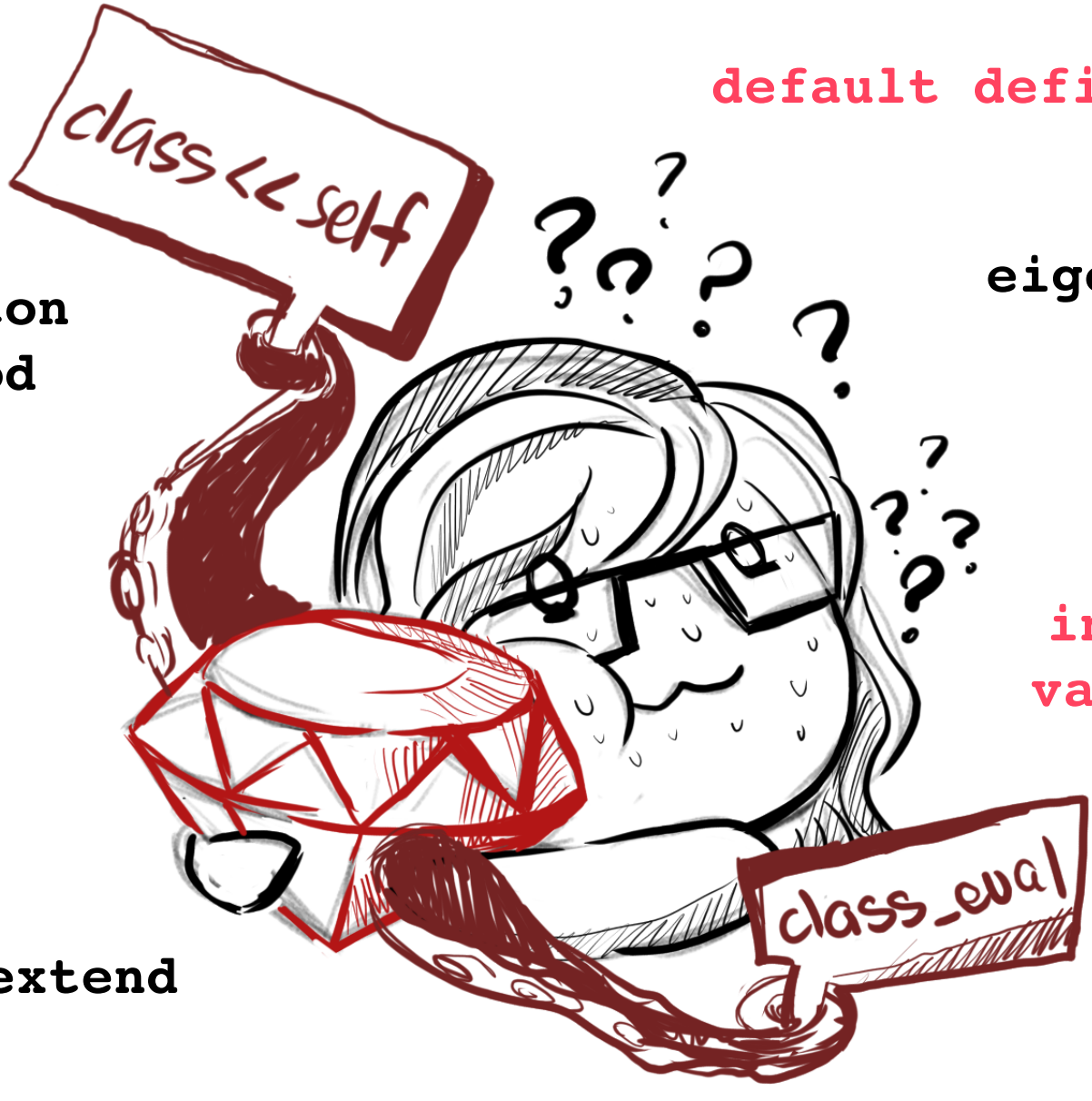
eigenclass

singleton
method

class
instance
variables

metaclass

extend



default definee

singleton method

eigenclass

class instance variables

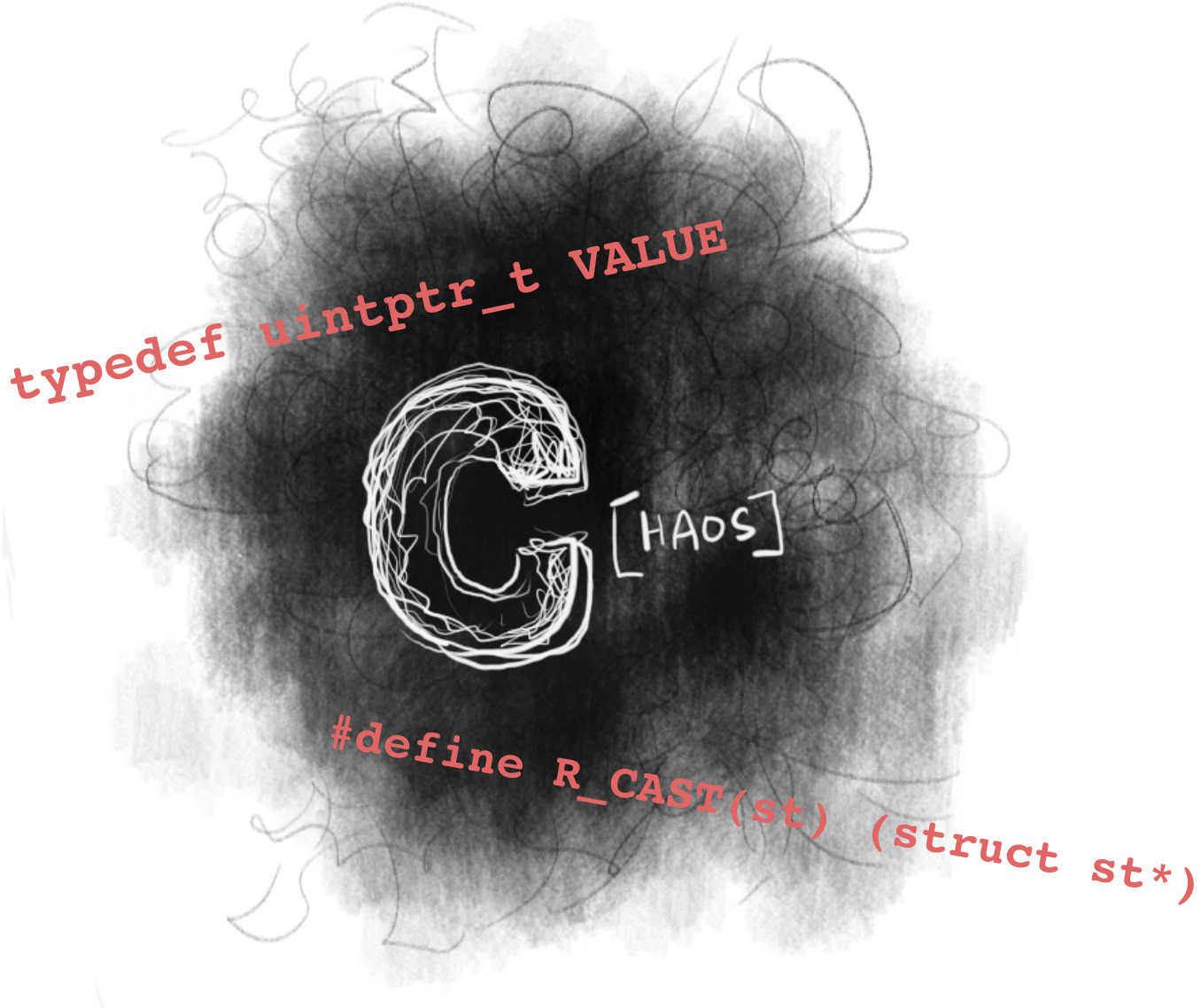
metaclass

extend

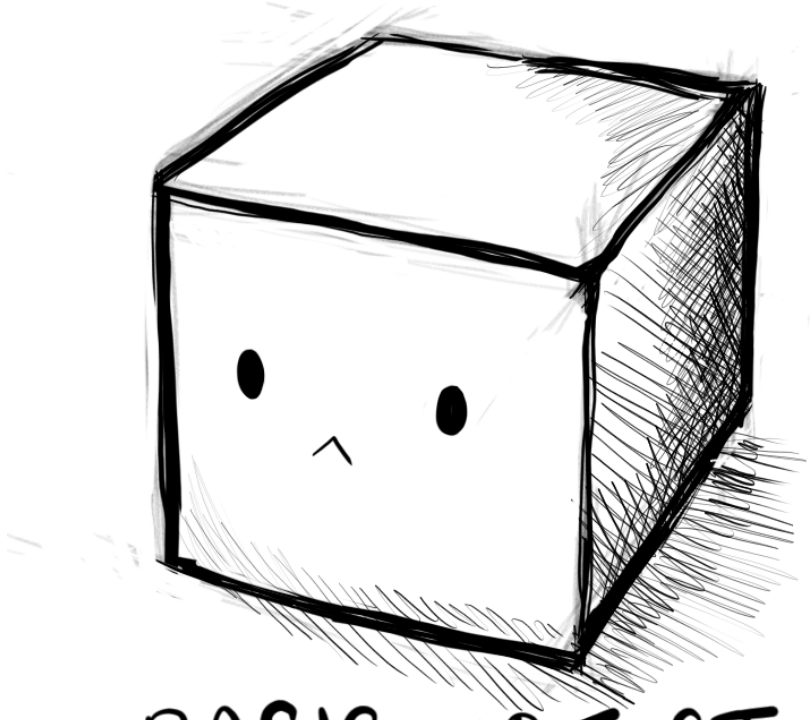

ALL I'D WANTED TO
KNOW ABOUT RUBY'S
OBJECT MODEL

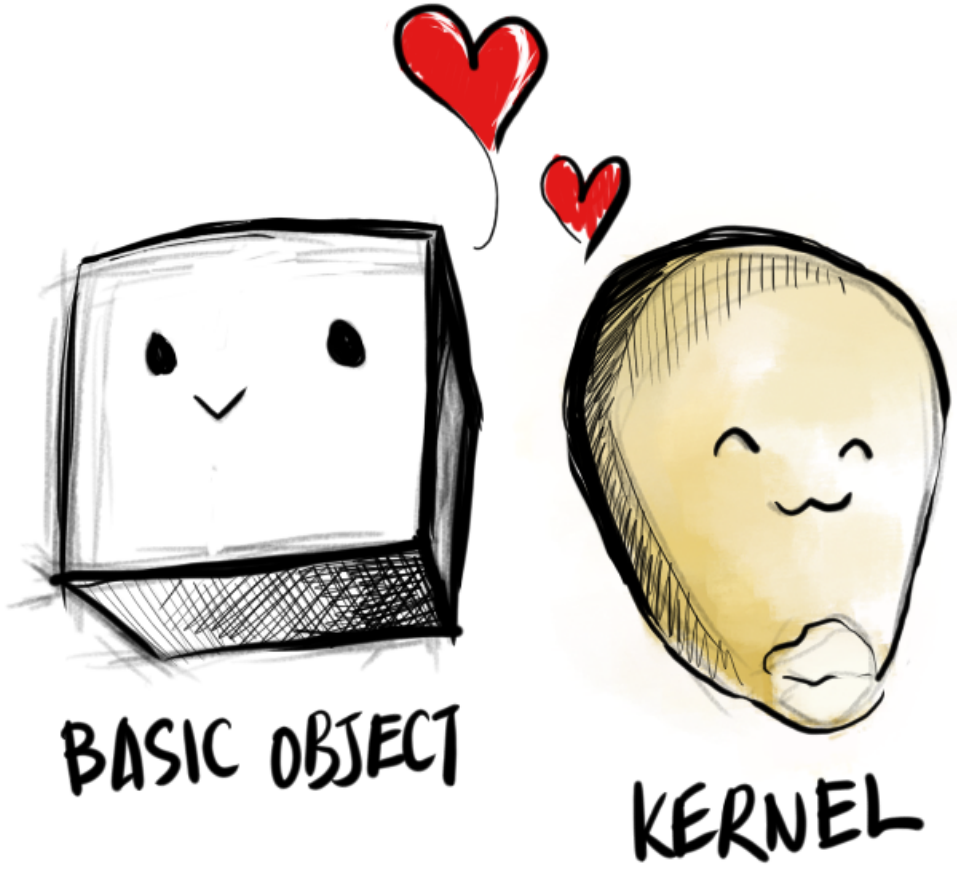

COMPLETIONIST

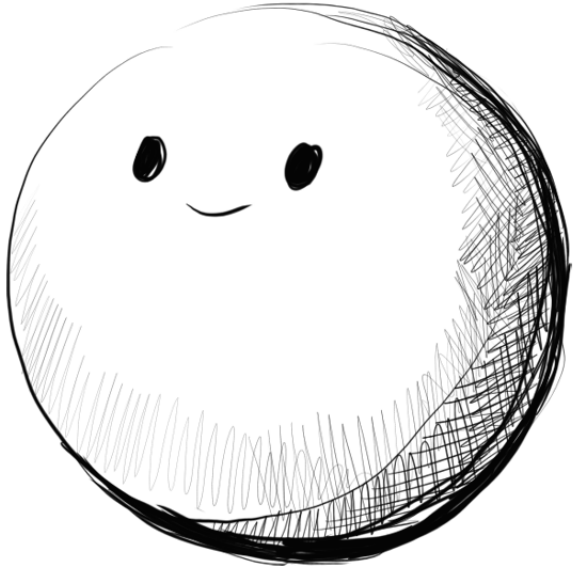

MOAR
!!!!

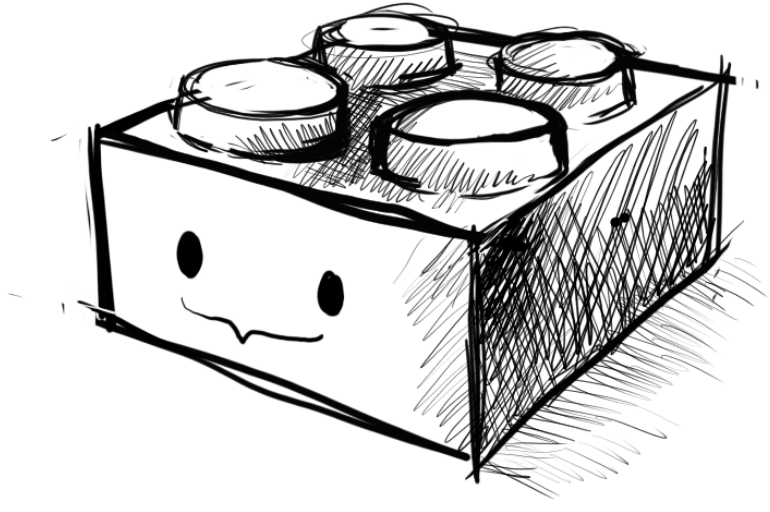

typedef uintptr_t VALUE
C [HAOS]
#define R_CAST(st) (struct st*)


LET US
MAKE OBJECTS!


BASIC OBJECT

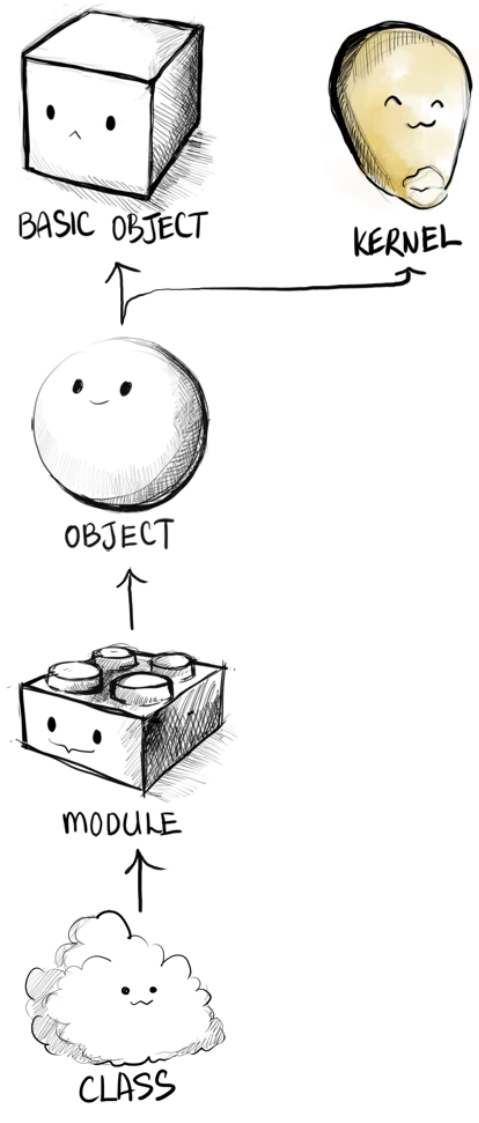

BASIC OBJECT
KERNEL


OBJECT

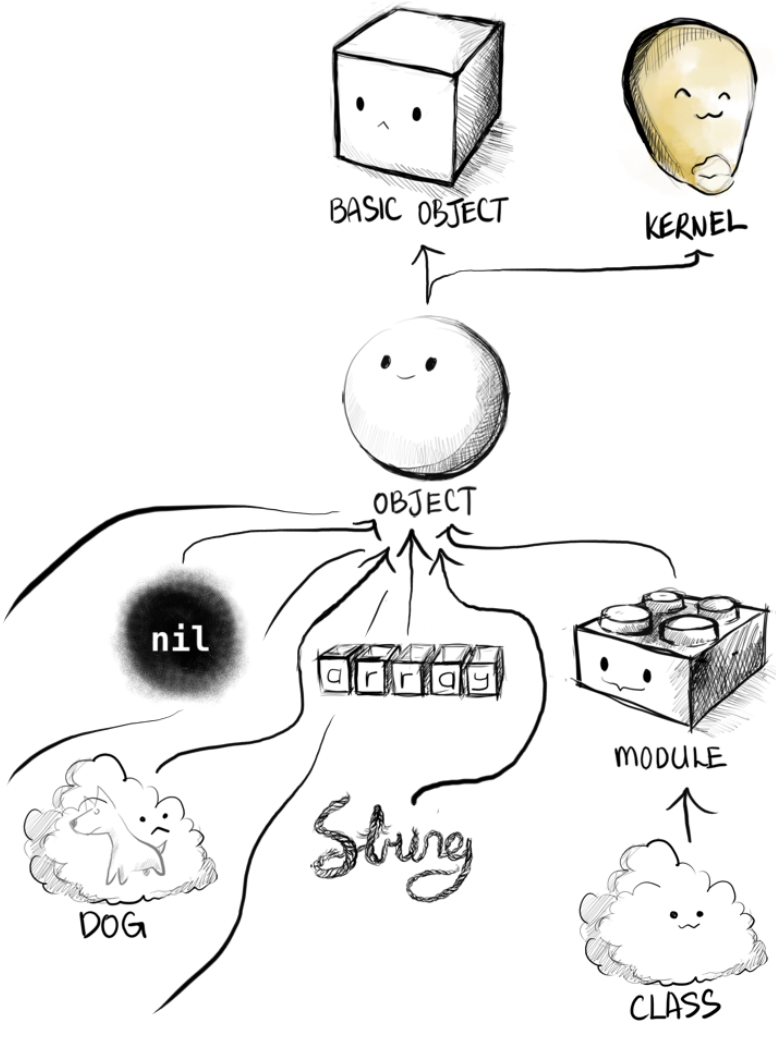


MODULE


CLASS


BASIC OBJECT
KERNEL
OBJECT
MODULE
CLASS


DOG
nil
array
String
OBJECT

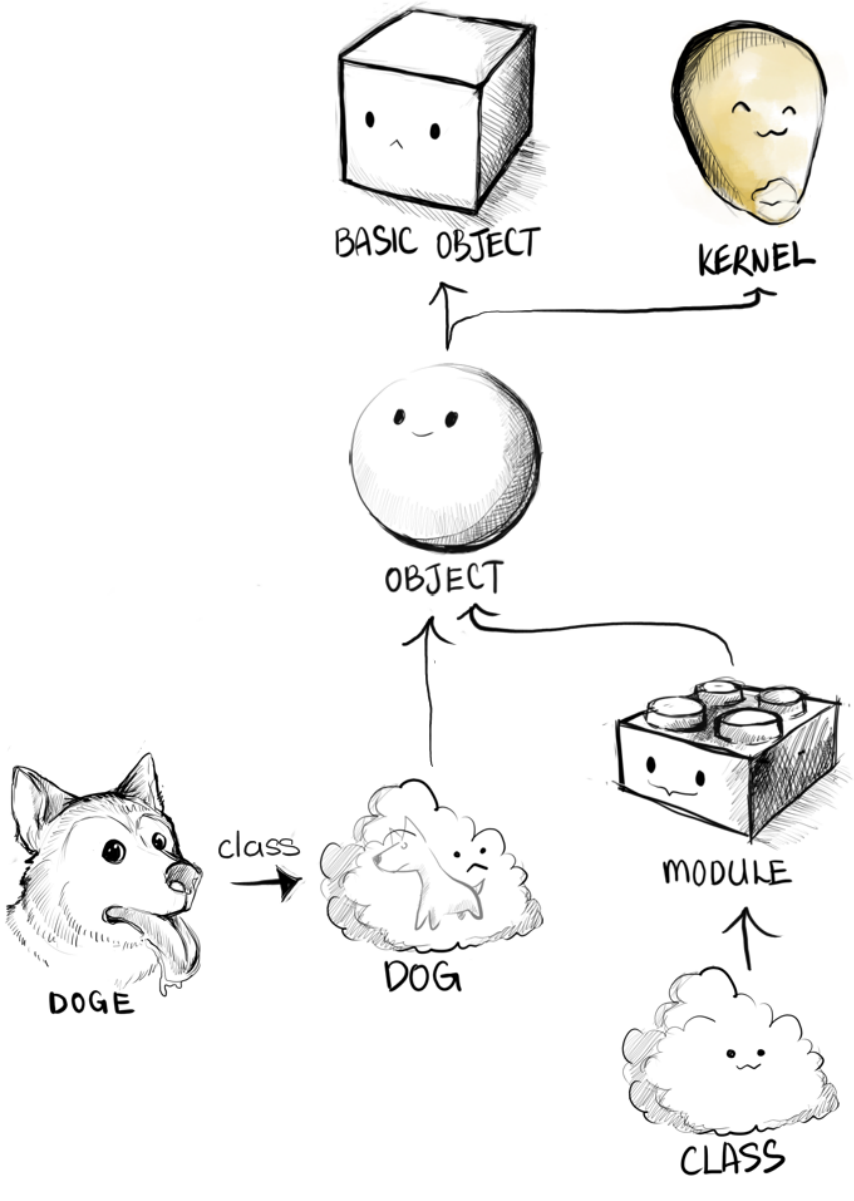

BASIC OBJECT
KERNEL
OBJECT
nil
array
MODULE
String
DOG
CLASS

INTROSPECTION


```ruby
class Dog
# ...
end

doge = Dog.new
```

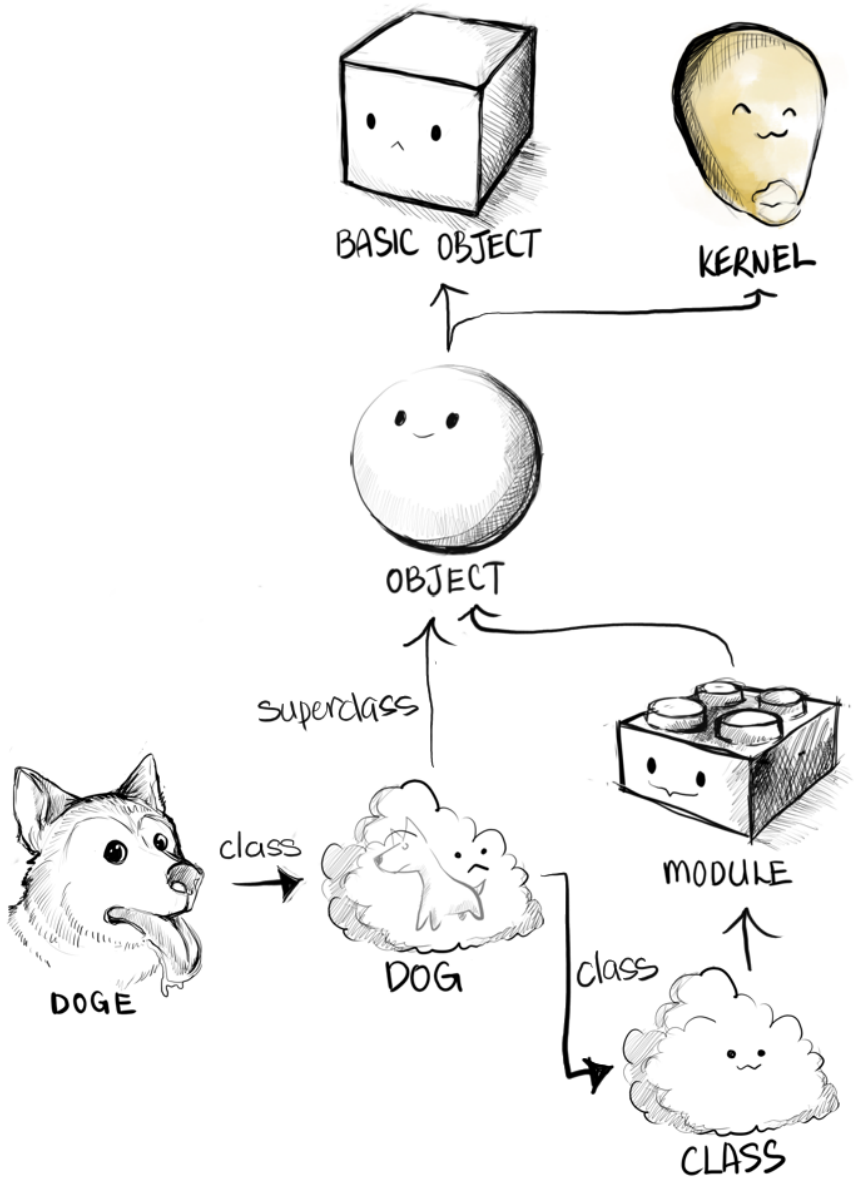

```
doge.class
```

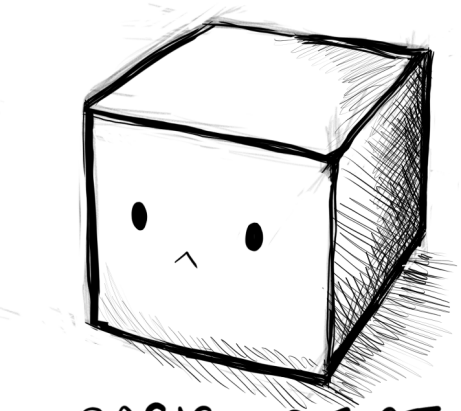

```
doge.class # => Dog
```

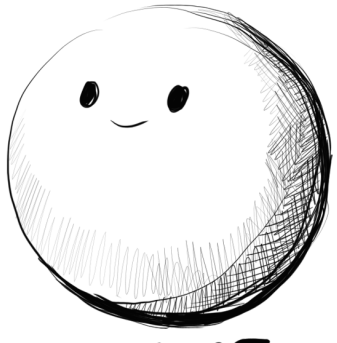

BASIC OBJECT
KERNEL
OBJECT
MODULE
class
DOGE
DOG
CLASS

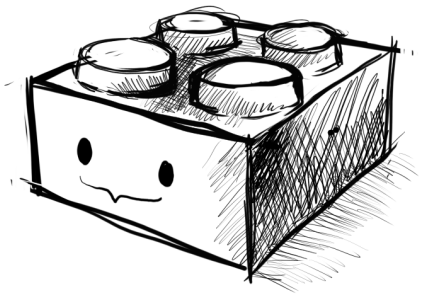

```
Dog.class # => Class
```


```
Dog.superclass # => Object
```


BASIC OBJECT
KERNEL
OBJECT
superclass
MODULE
class
DOG
class
DOGE
CLASS

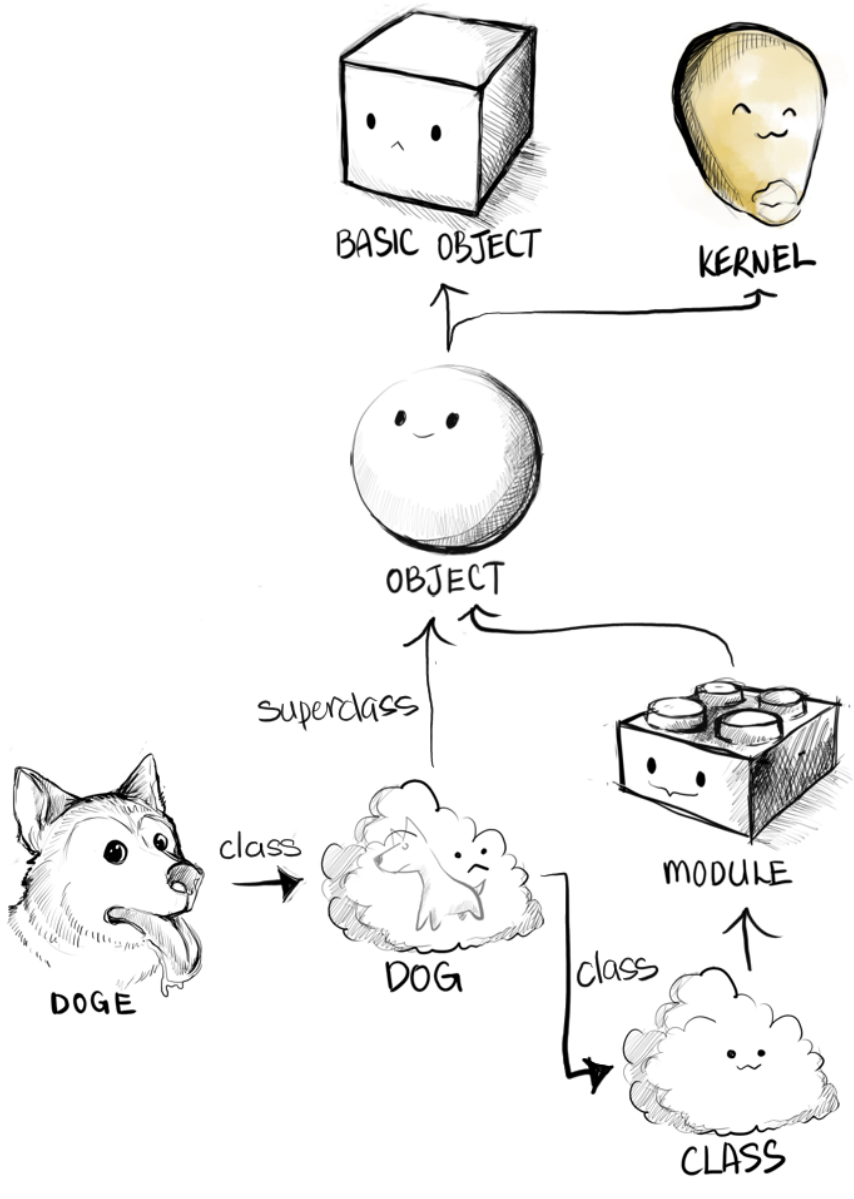

BASIC OBJECT


OBJECT

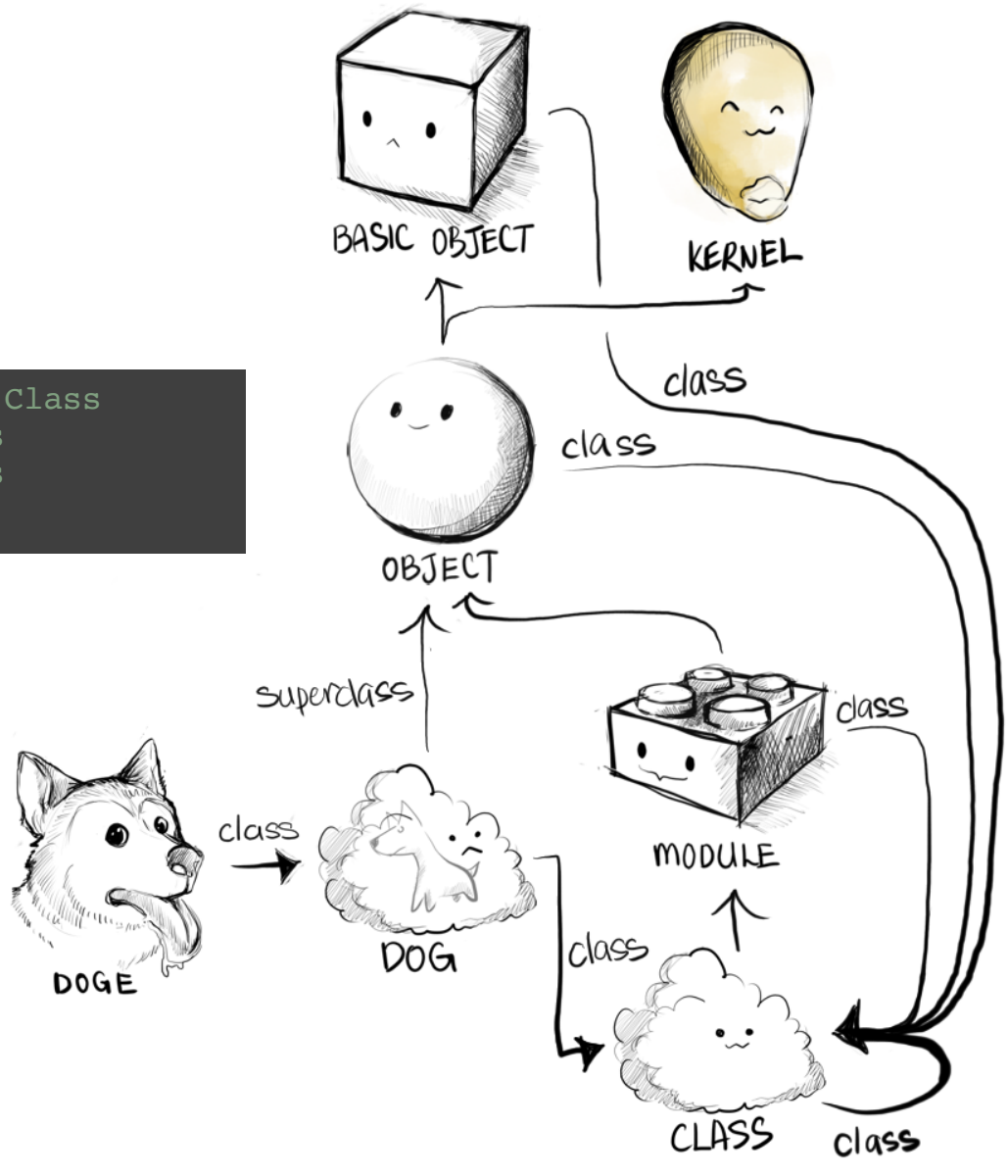

MODULE

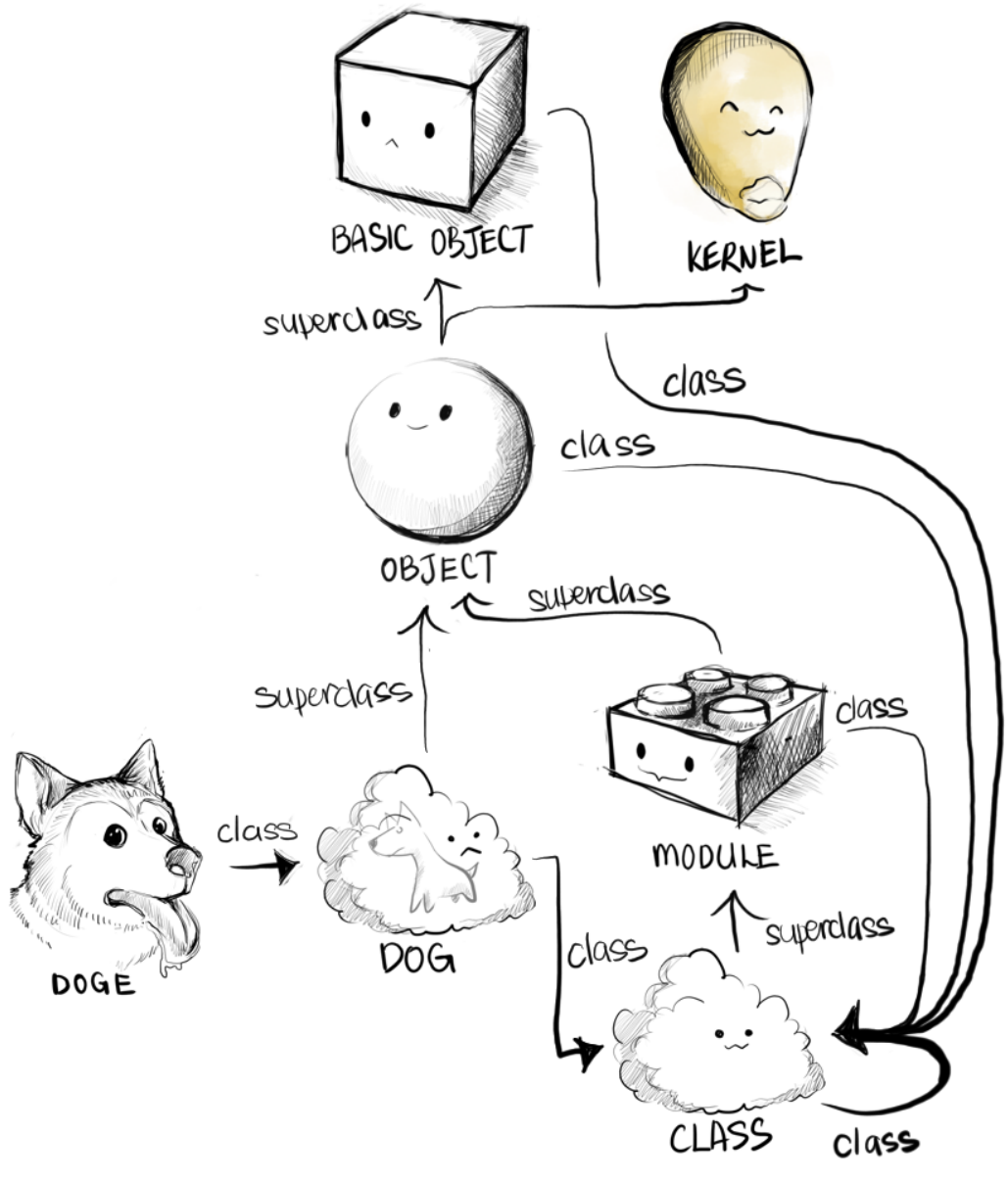

CLASS

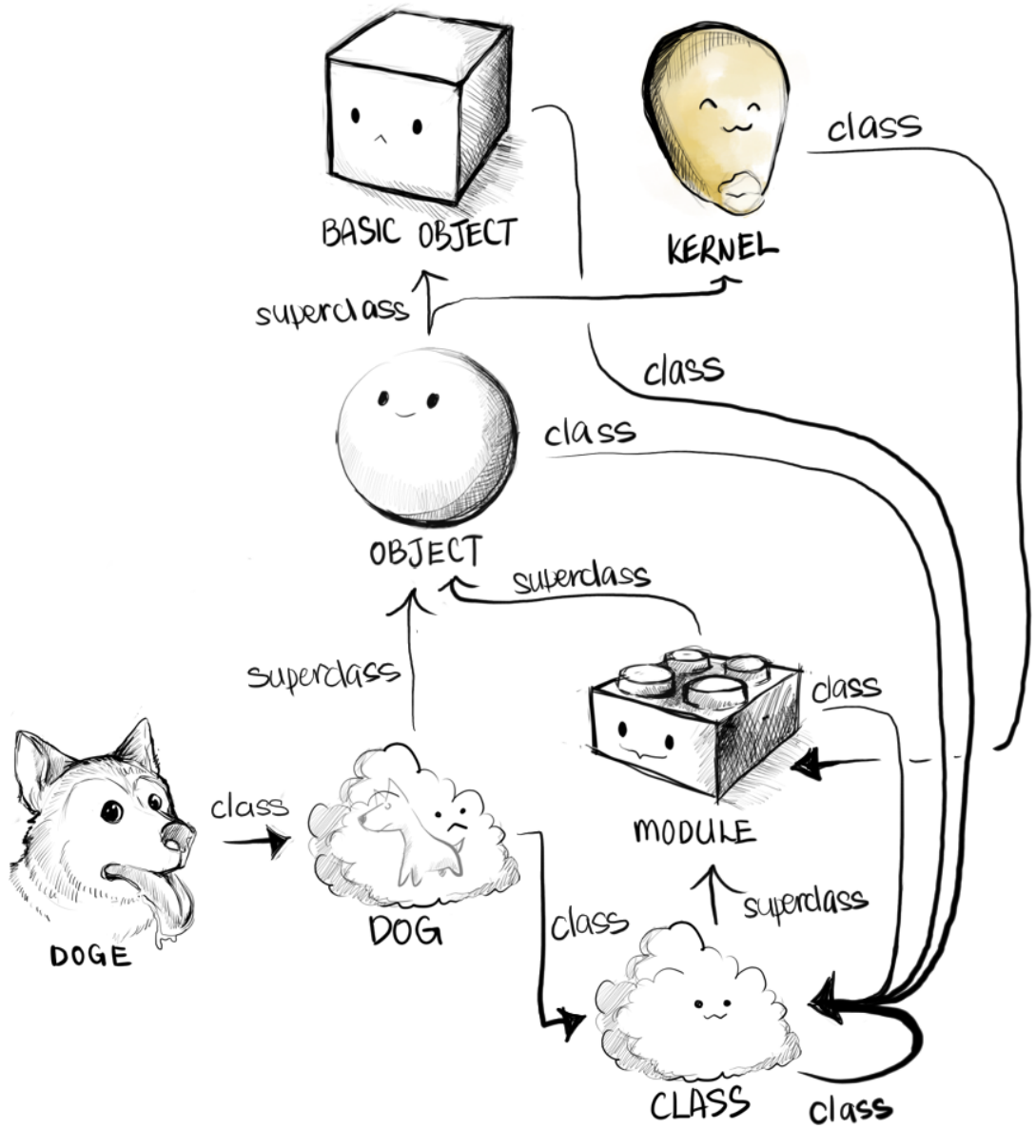

WHAT
AM I?


BASIC OBJECT
KERNEL
OBJECT
superclass
MODULE
class
DOG
class
DOGE
CLASS

```
BasicObject.class # => Class
Object.class # => Class
Module.class # => Class
Class.class # => Class
```




BASIC OBJECT
KERNEL
superclass
class
class
OBJECT
superclass
superclass
class
MODULE
DOGE
class
DOG
class
superclass
CLASS
class

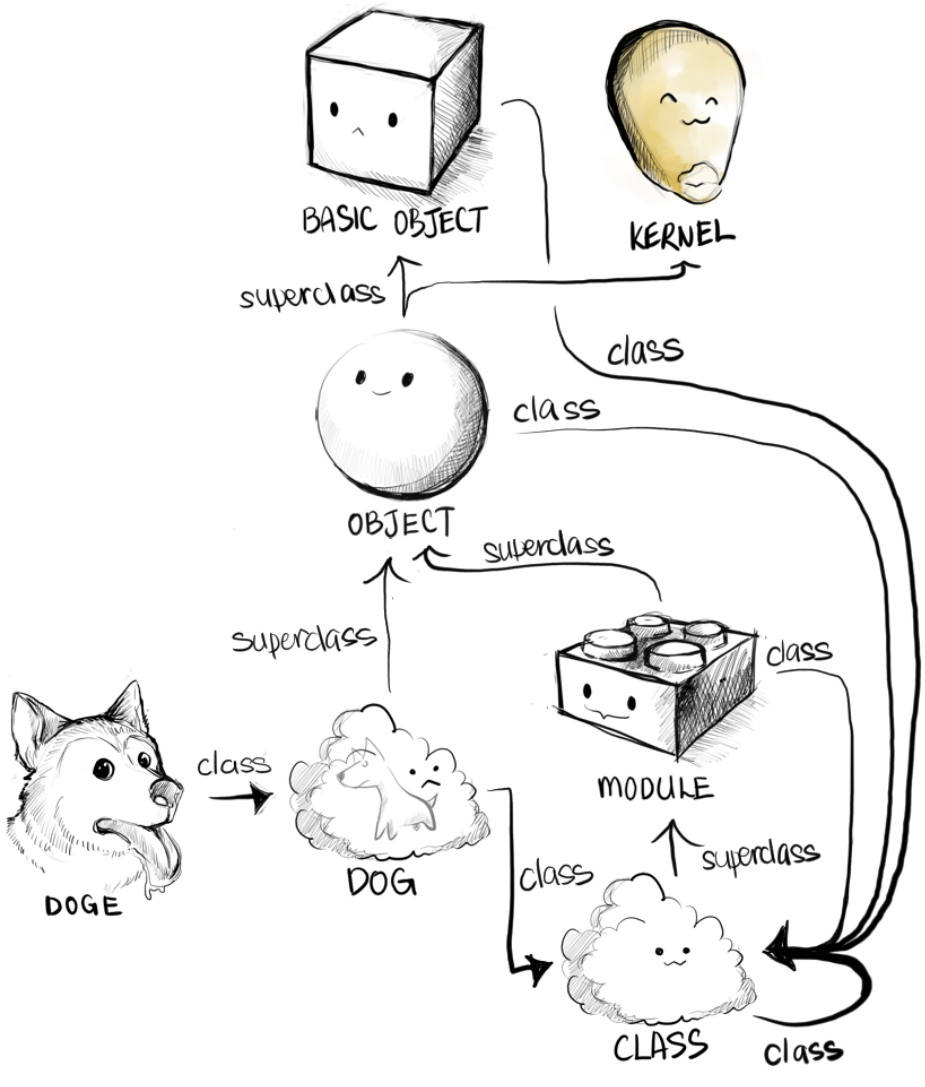

BASIC OBJECT
KERNEL
class
superclass
class
class
OBJECT
superclass
superclass
class
MODULE
DOGE
class
DOG
class
superclass
CLASS
class

* dogespeak: make me a unique snowflake

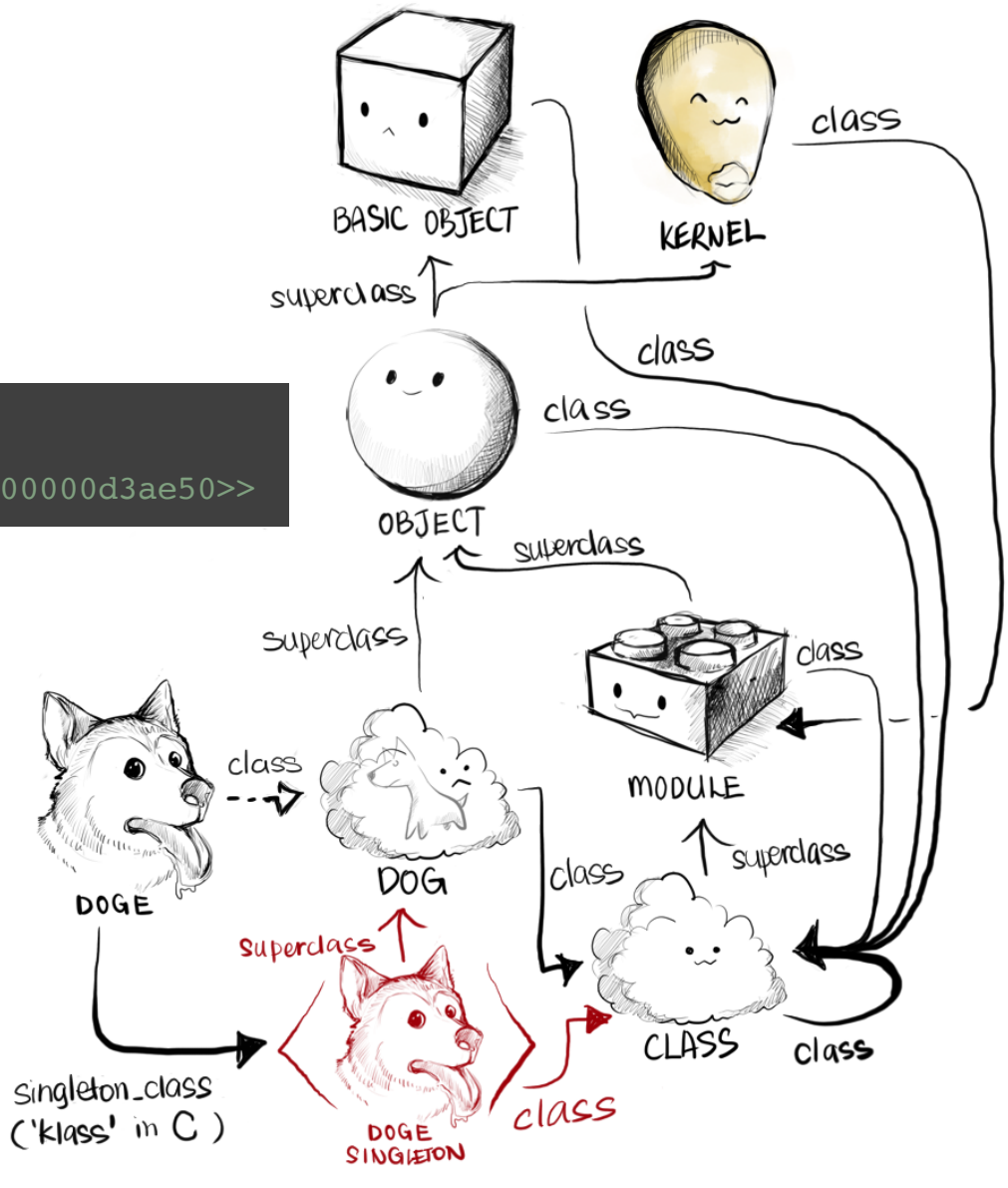

```
def doge.speak(word)
    @modifiers |= %w(so very many much such)
    "#{@modifiers.sample} #{word}"
end
```


```
def doge.speak(word)
    @modifiers |= %w(so very many much such)
    "#{@modifiers.sample} #{word}"
end
```

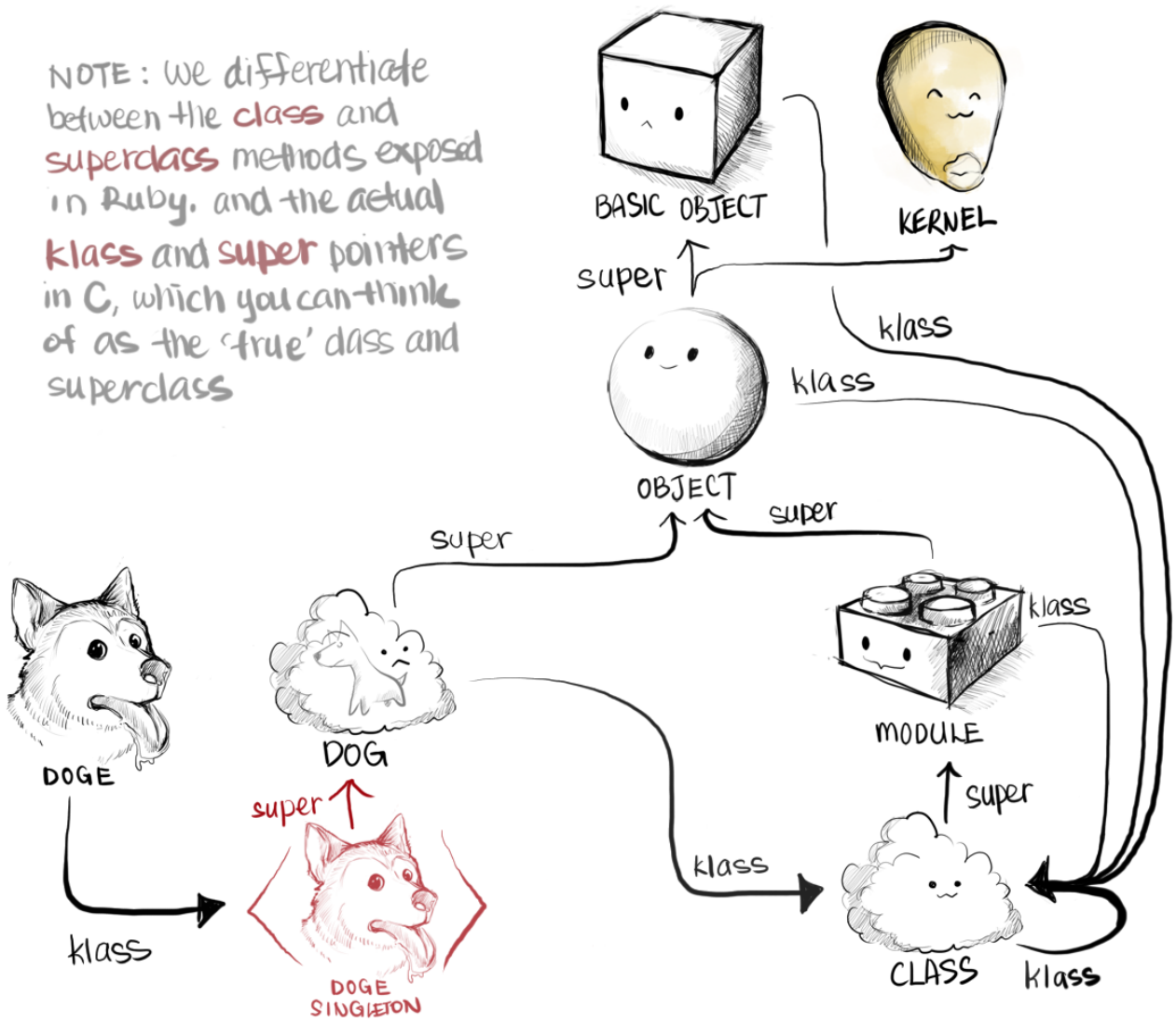

```
def doge.speak(word)
    @modifiers |= %w(so very many much such)
    "#{@modifiers.sample} #{word}"
end
```

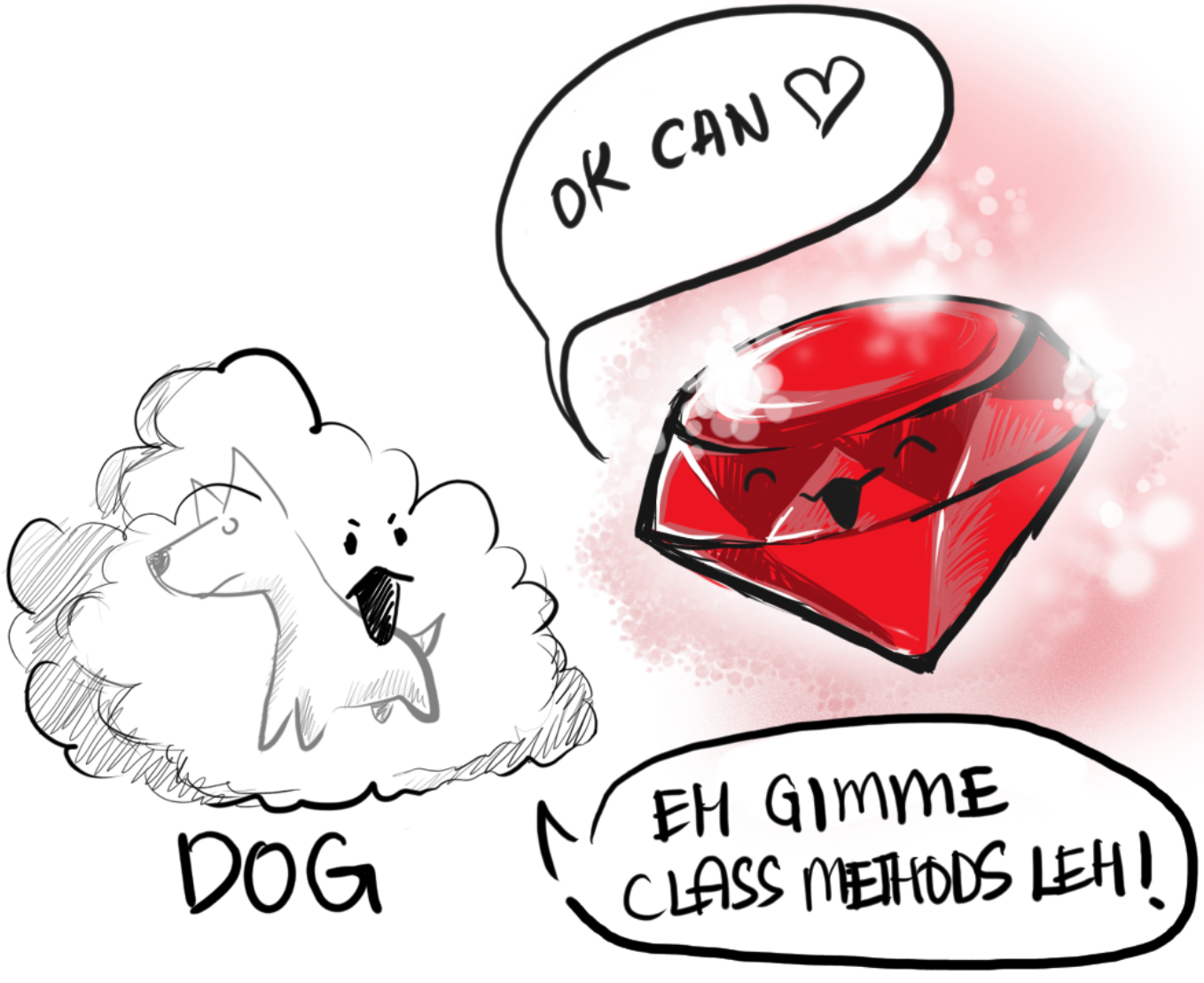

BASIC OBJECT
KERNEL
superclass
class
class
OBJECT
superclass
superclass
class
MODULE
class
DOGE
class
DOG
class
superclass
CLASS
class

```
doge.class # => Dog
doge.singleton_class
# => #<Class:#<Dog:0x00000000d3ae50>>
```

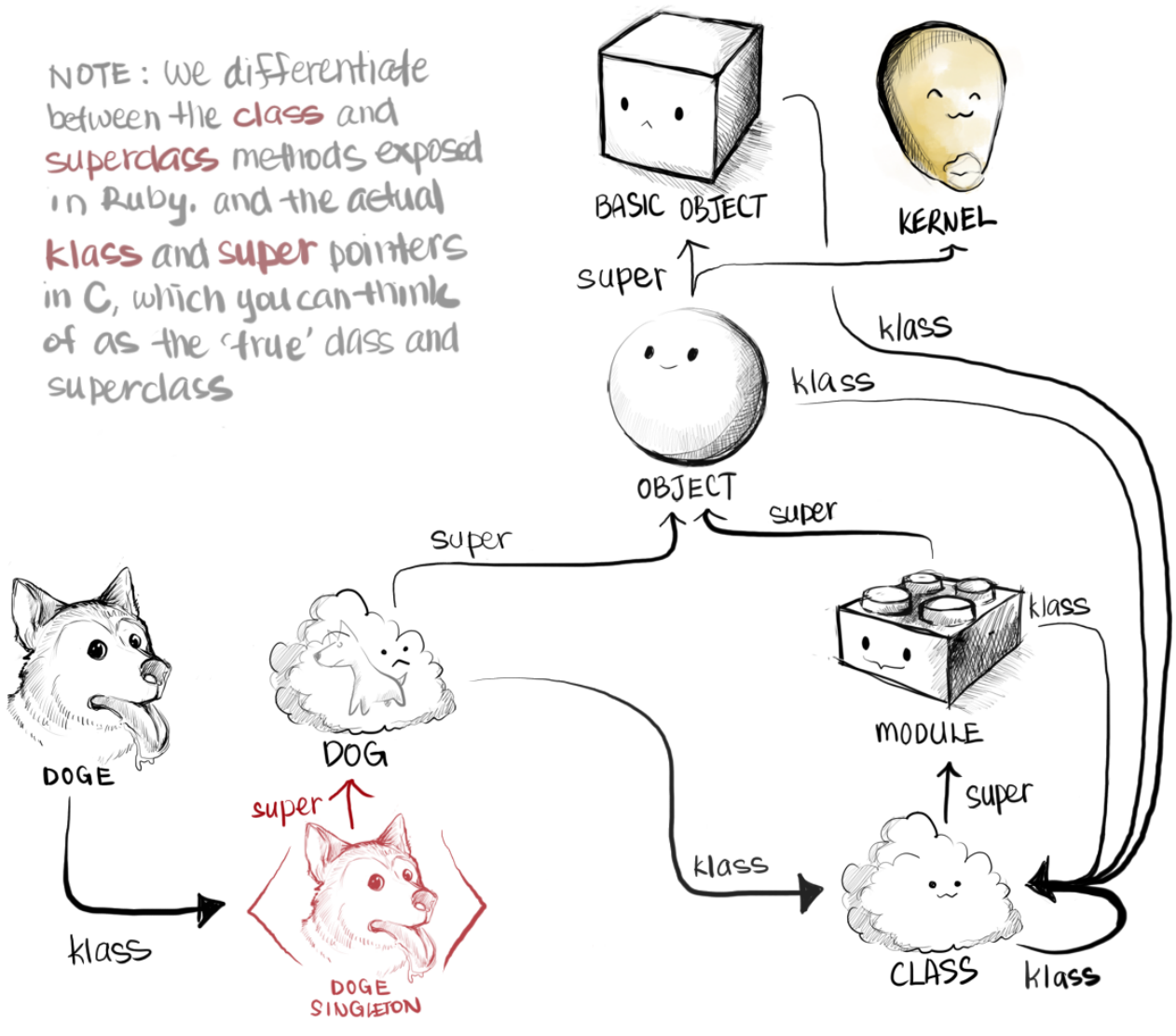


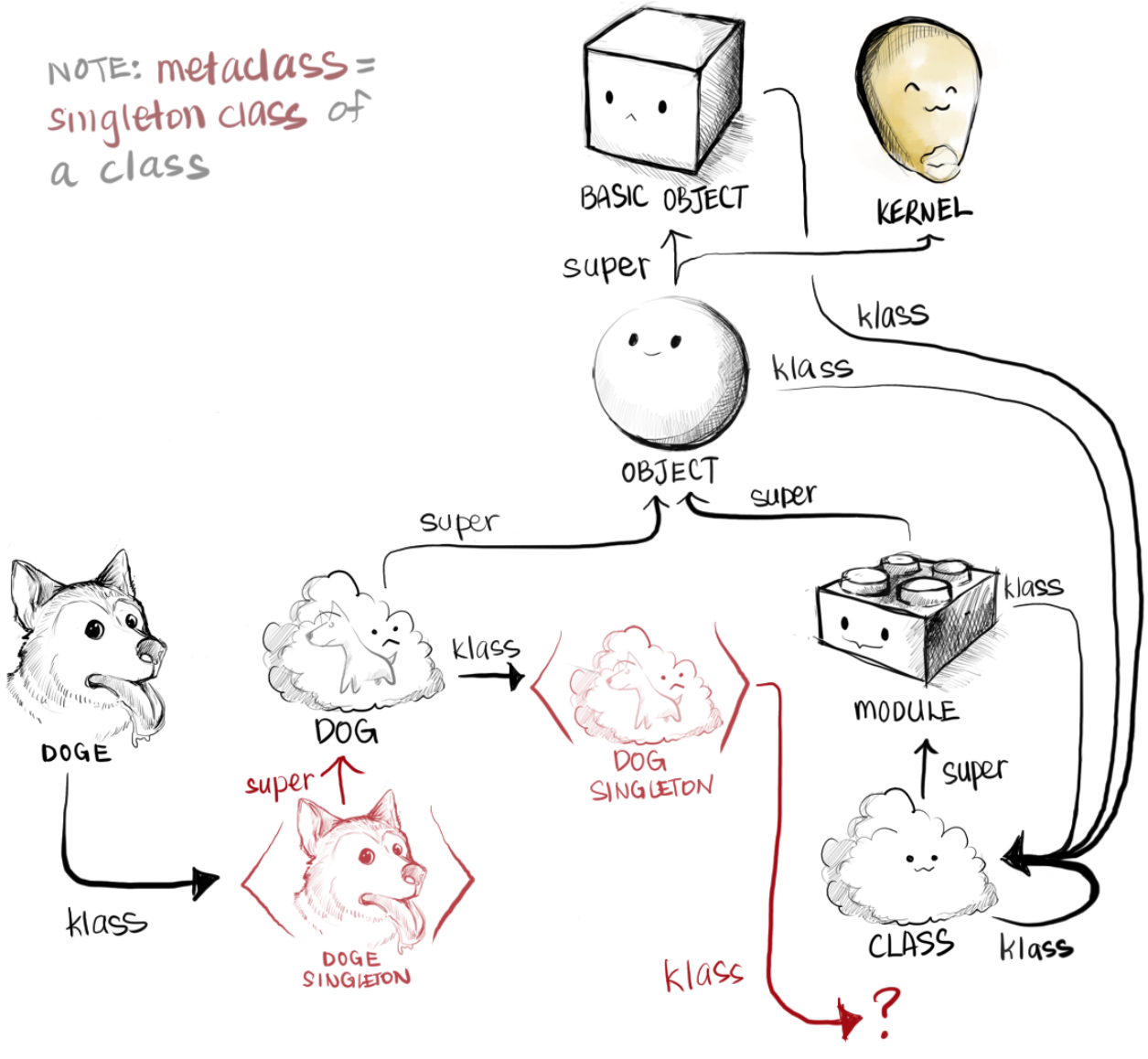
NOTE: We differentiate between the **class** and **superclass** methods exposed in Ruby, and the actual **klass** and **super** pointers in C, which you can think of as the 'true' class and superclass

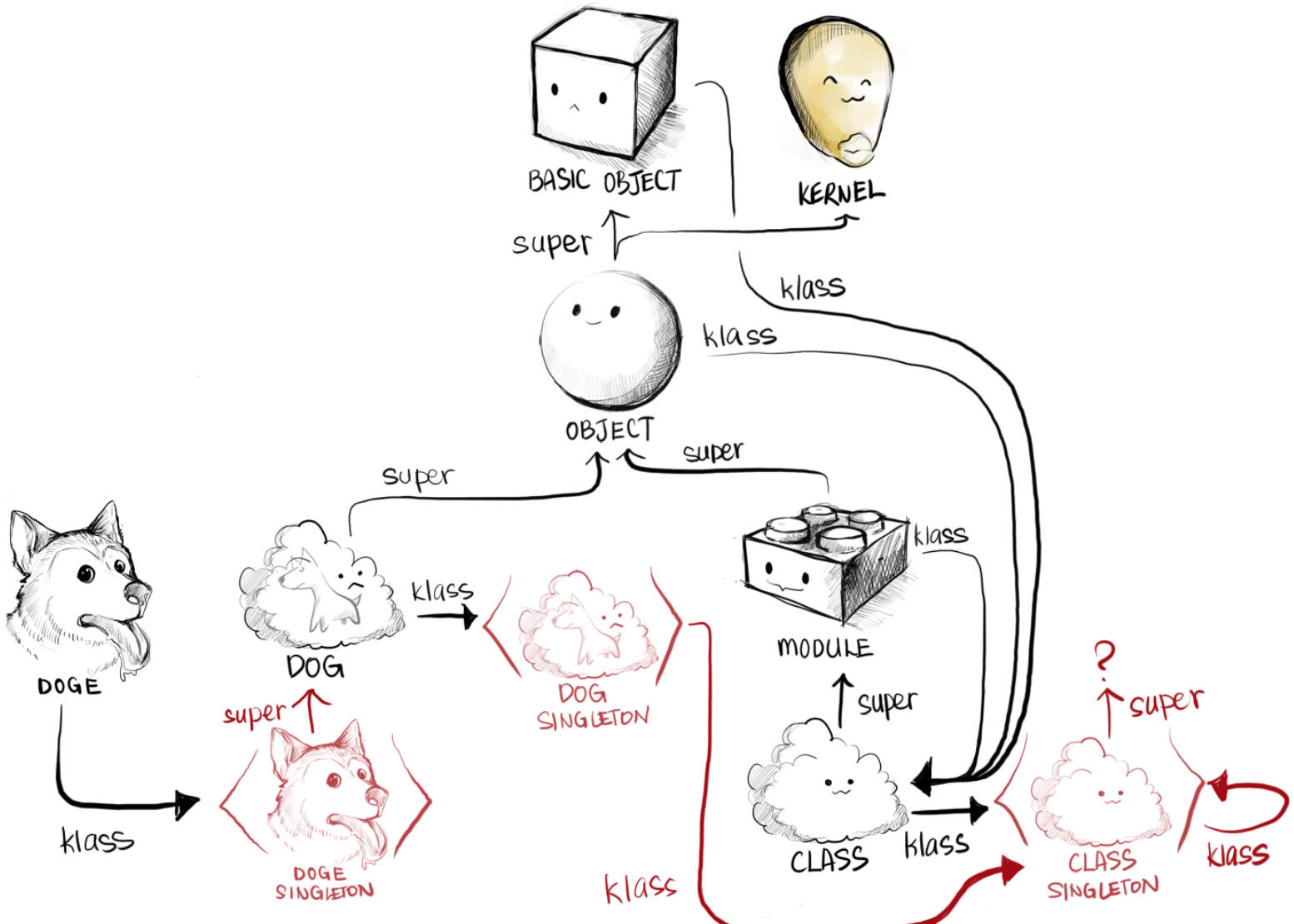


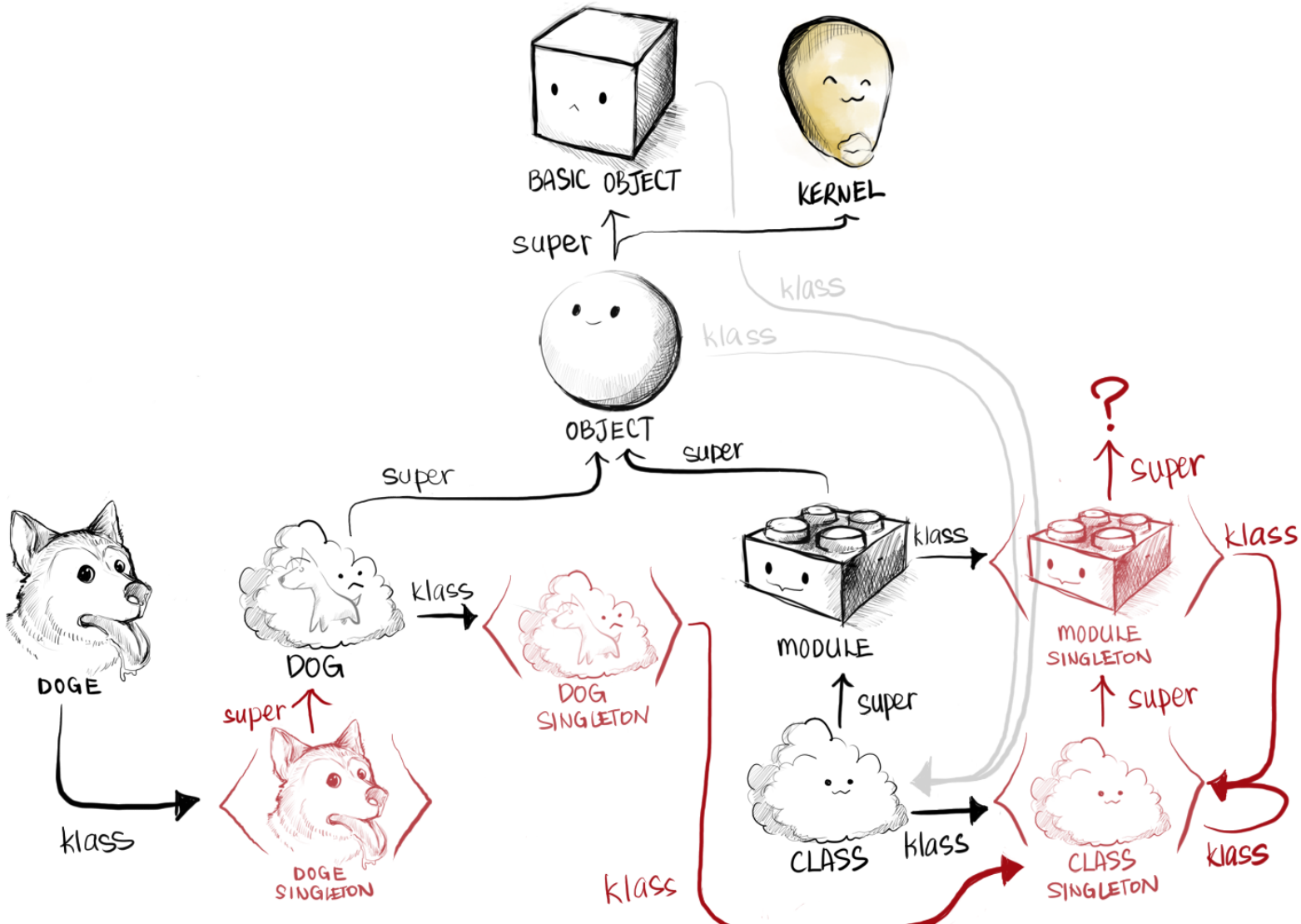

OK CAN ♡
DOG
EH GIMME CLASS METHODS LEH!

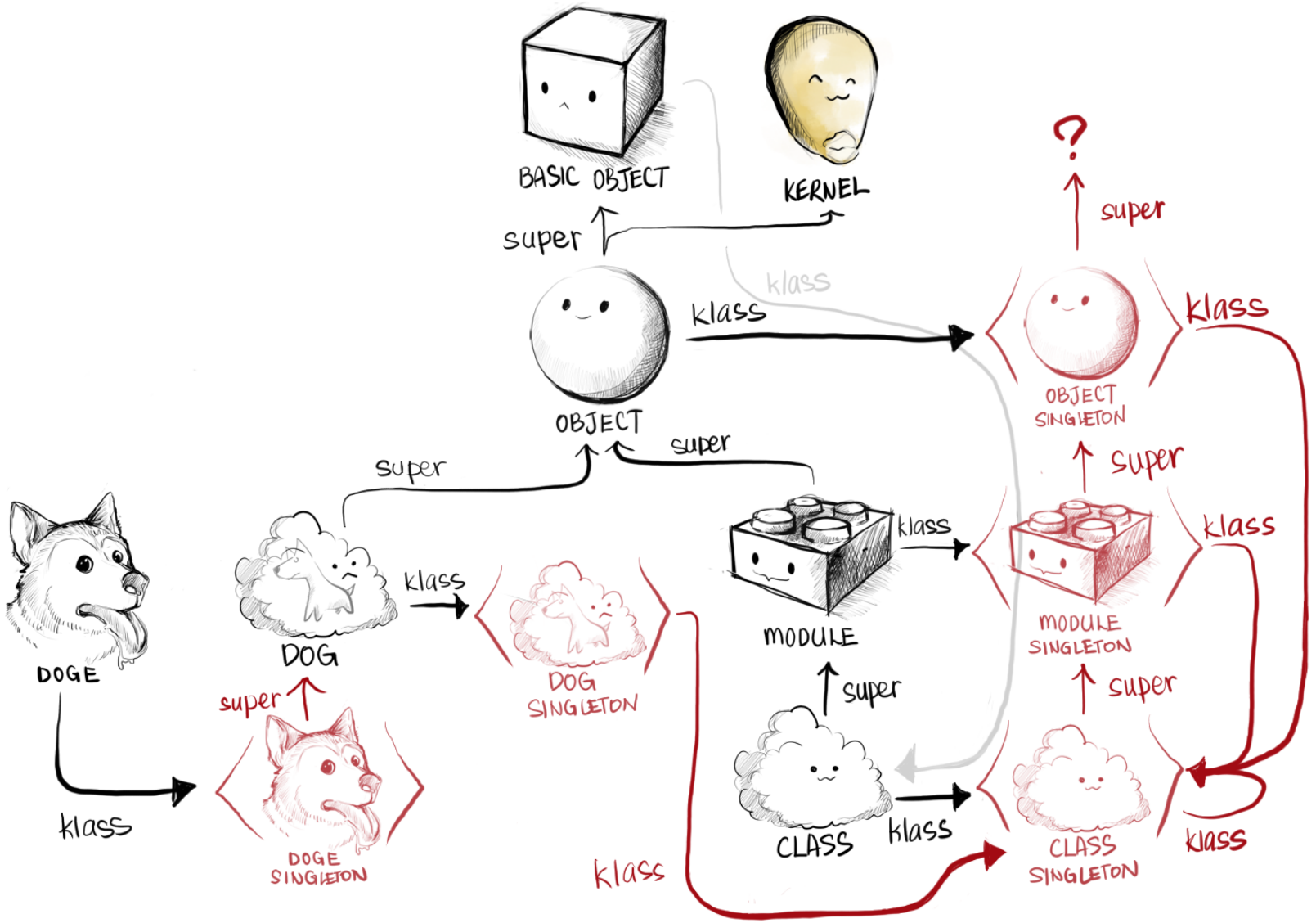
NOTE: We differentiate between the **class** and **superclass** methods exposed in Ruby, and the actual **klass** and **super** pointers in C, which you can think of as the 'true' class and superclass

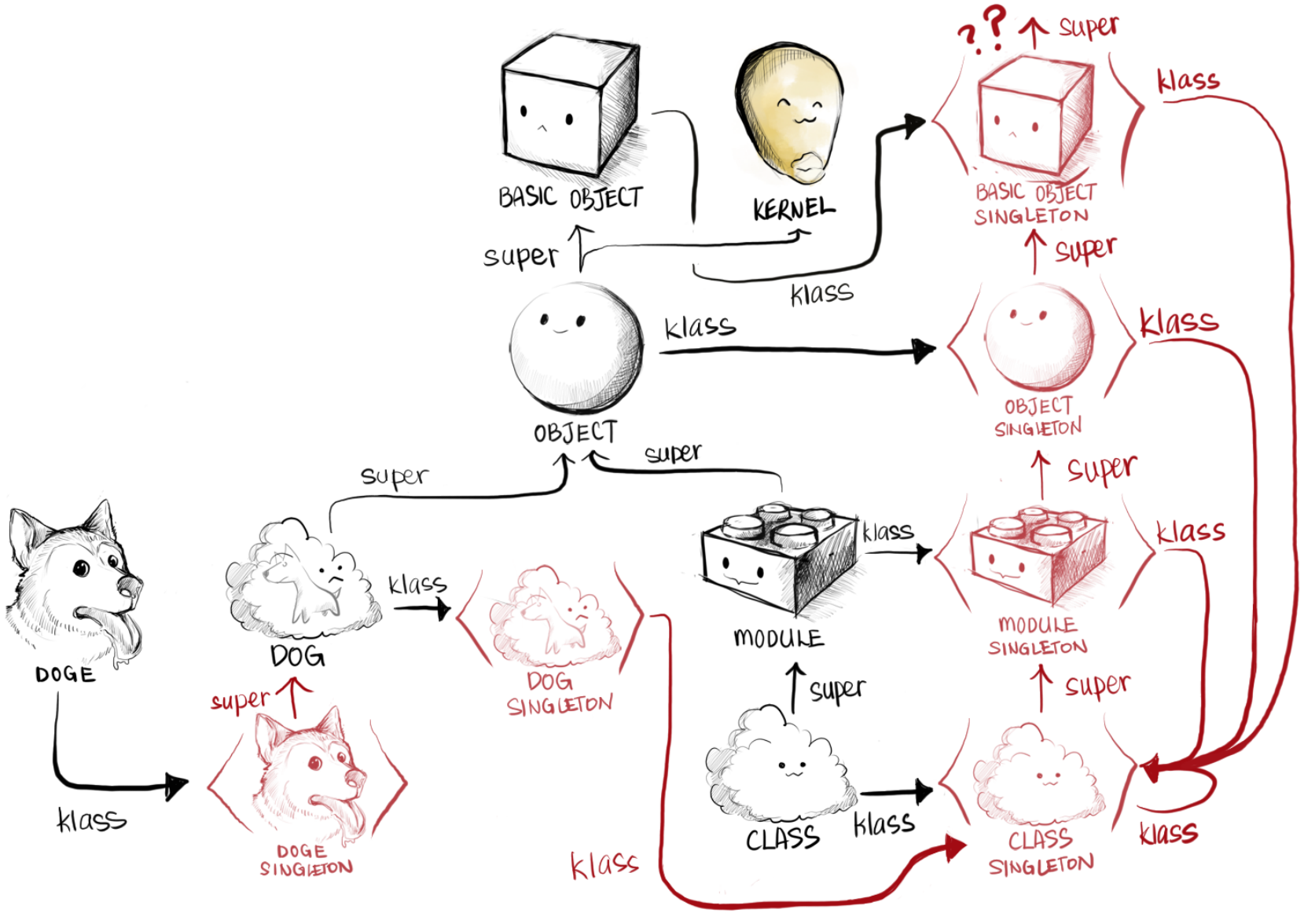


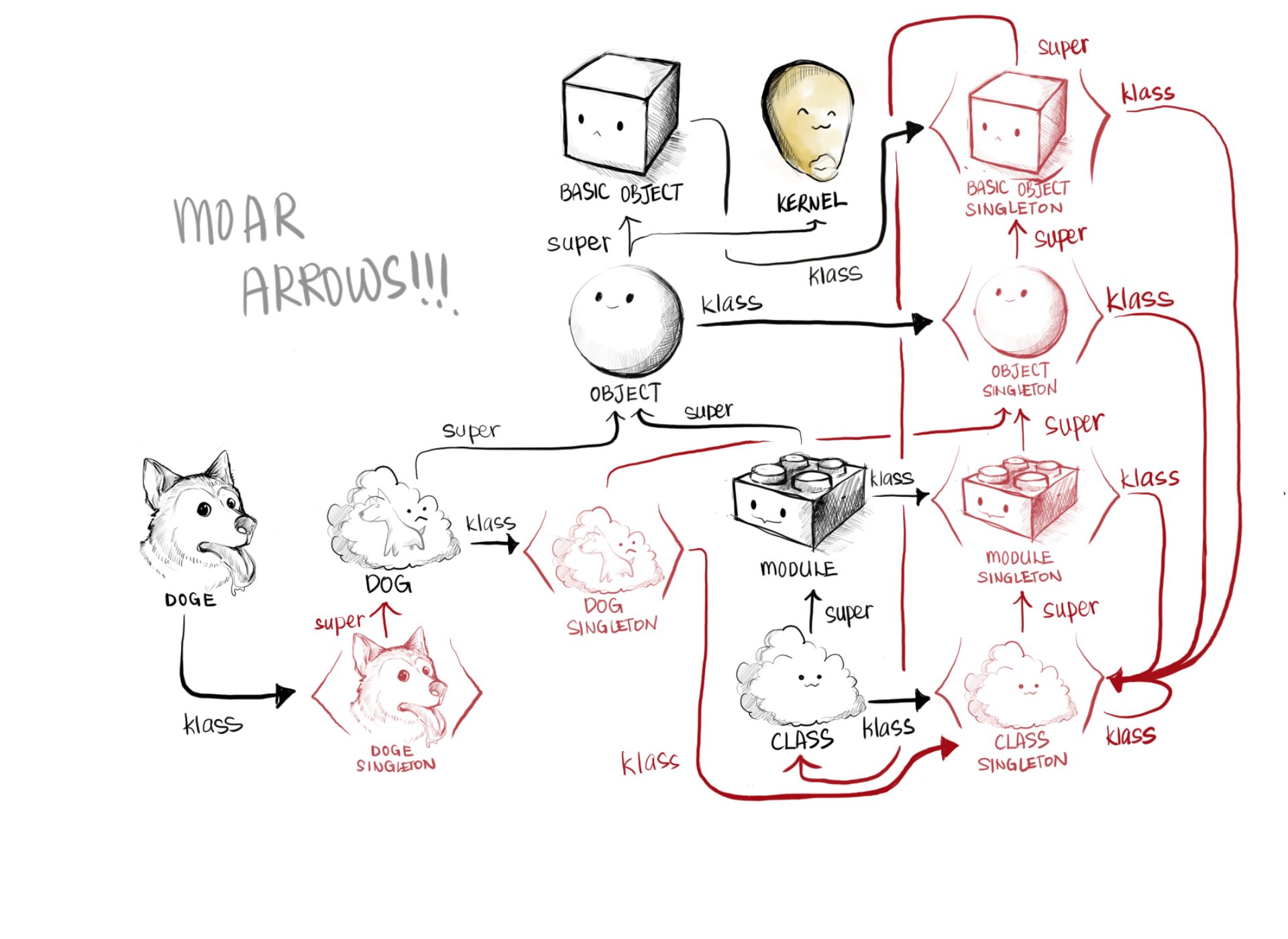

NOTE: metaclass = singleton class of a class
BASIC OBJECT
KERNEL
super
klass
klass
OBJECT
super
super
klass
DOG
klass
DOG SINGLETON
DOGE
super
DOGE SINGLETON
klass
MODULE
klass
super
CLASS
klass
klass
?

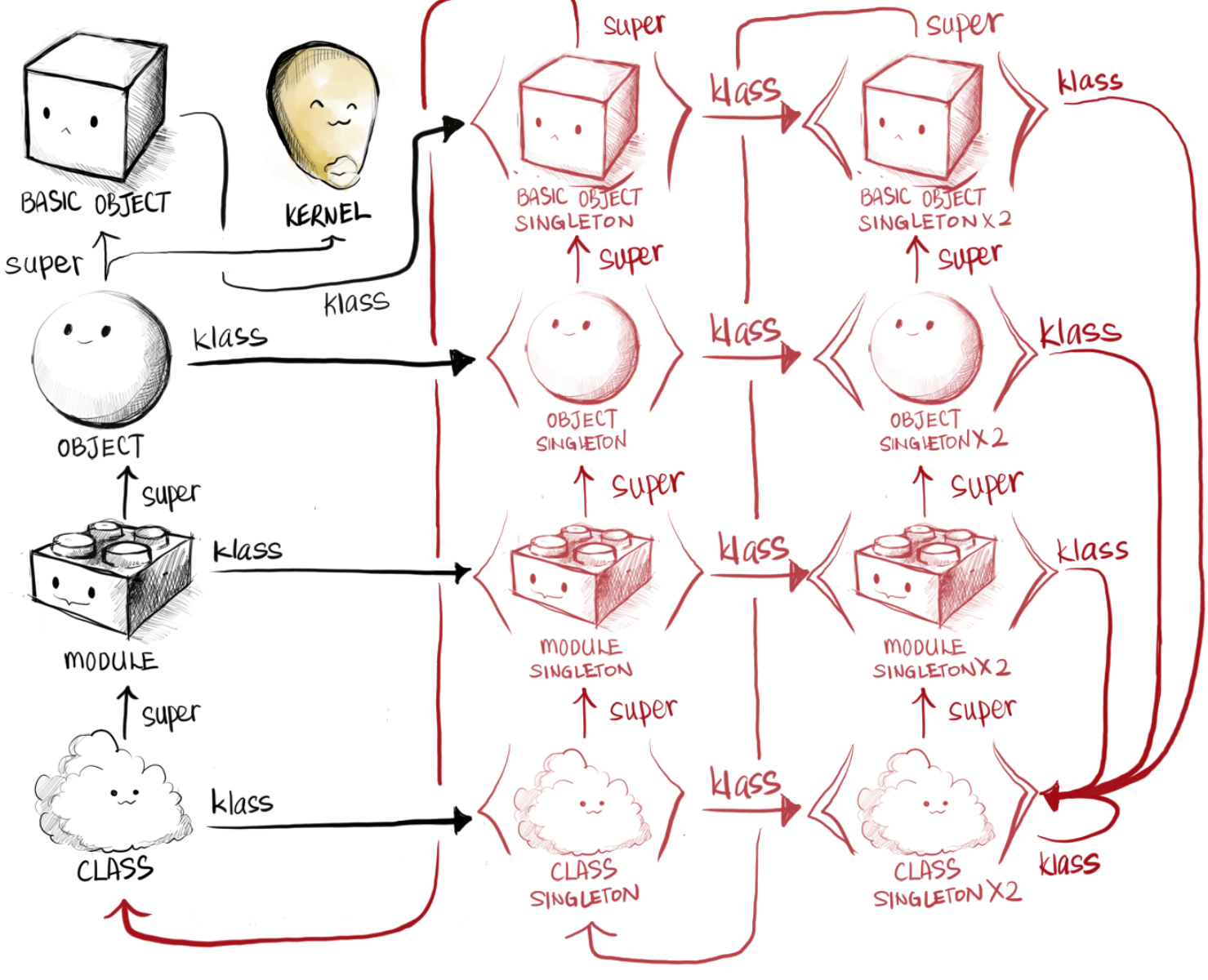

BASIC OBJECT
KERNEL
super
klass
klass
OBJECT
super
super
klass
DOG
klass
DOG SINGLETON
MODULE
klass
DOGE
super
super
klass
DOGE SINGLETON
CLASS
klass
?
super
CLASS SINGLETON
klass
klass

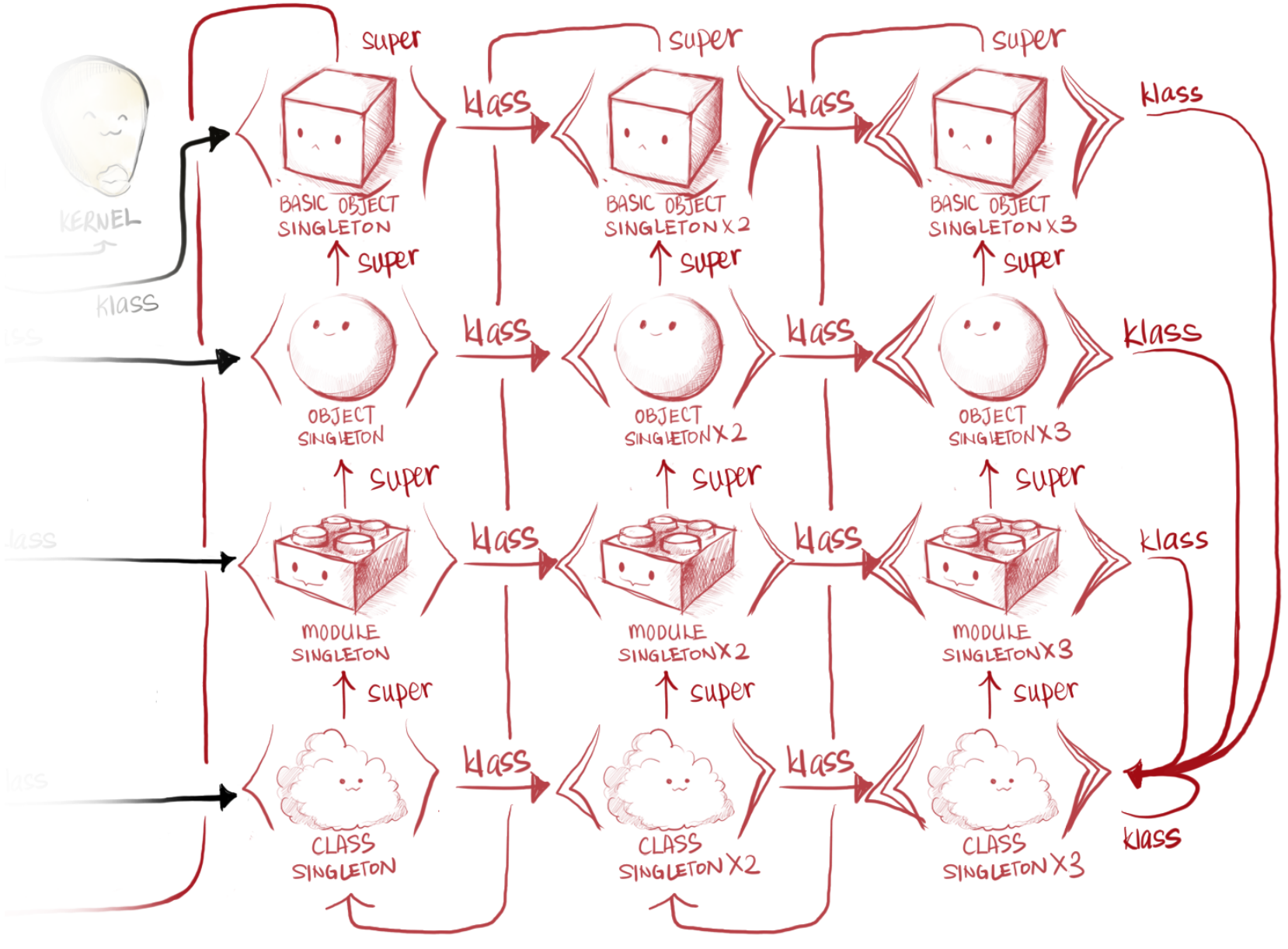

BASIC OBJECT
KERNEL
super
klass
klass
OBJECT
super
super
?
super
klass
klass
DOGE
DOG
klass
DOG SINGLETON
MODULE
MODULE SINGLETON
super
super
super
DOGE SINGLETON
klass
CLASS
klass
CLASS SINGLETON
klass
klass

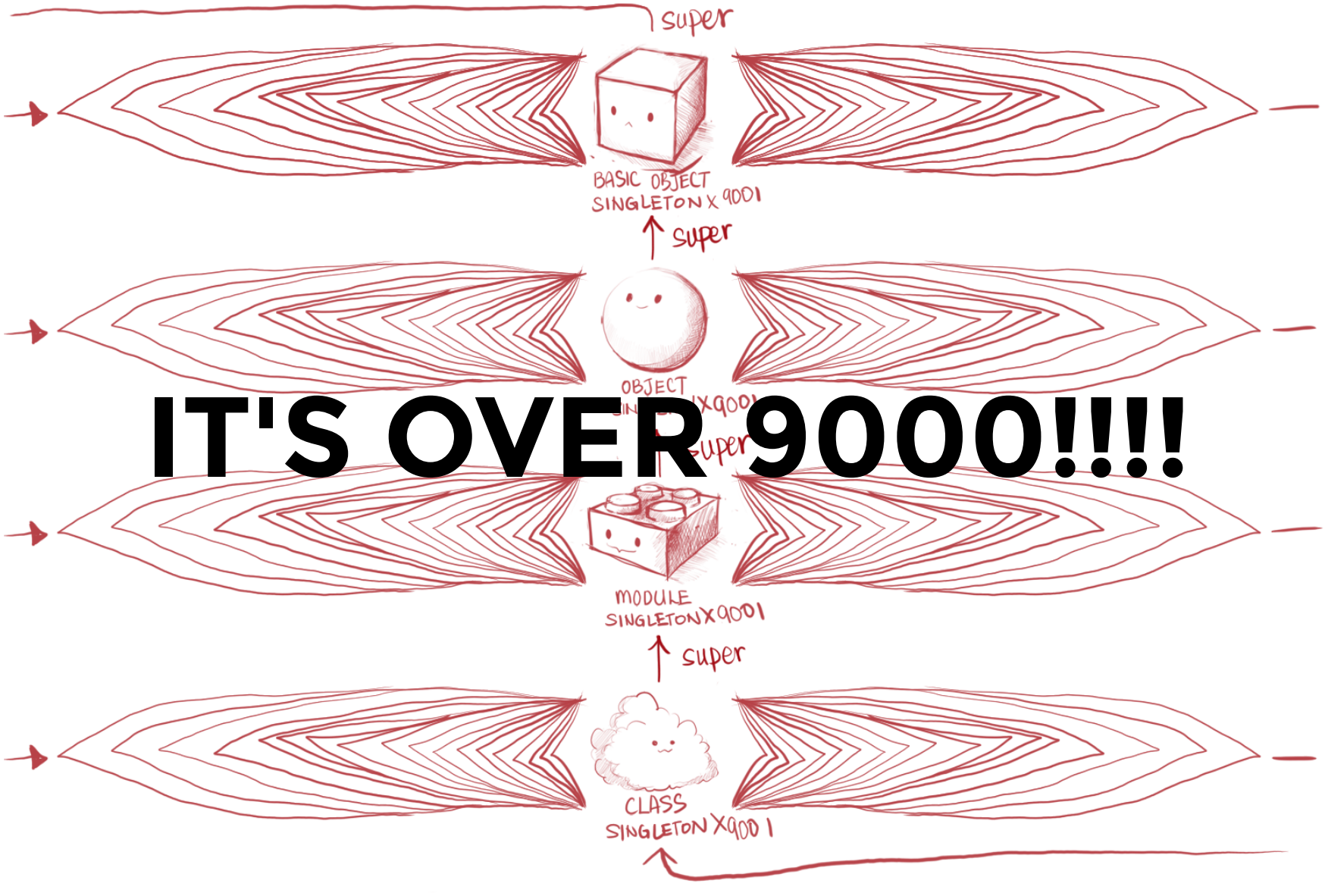

BASIC OBJECT
KERNEL
?
super
super
klass
klass
OBJECT
OBJECT SINGLETON
klass
super
super
super
DOGE
DOG
klass
DOG SINGLETON
MODULE
klass
MODULE SINGLETON
klass
super
super
super
klass
DOGE SINGLETON
CLASS
klass
CLASS SINGLETON
klass
klass


BASIC OBJECT
KERNEL
super
klass
??
super
klass
BASIC OBJECT SINGLETON
super
klass
OBJECT
klass
OBJECT SINGLETON
super
super
super
klass
DOGE
DOG
klass
DOG SINGLETON
MODULE
klass
MODULE SINGLETON
super
super
super
DOGE SINGLETON
klass
CLASS
super
klass
CLASS SINGLETON
klass
klass


MOAR ARROWS!!!
BASIC OBJECT
KERNEL
super
klass
BASIC OBJECT SINGLETON
OBJECT
OBJECT SINGLETON
MODULE
MODULE SINGLETON
DOGE
DOG
DOG SINGLETON
DOGE SINGLETON
CLASS
CLASS SINGLETON


BASIC OBJECT
KERNEL
super
klass
OBJECT
MODULE
CLASS
BASIC OBJECT SINGLETON
BASIC OBJECT SINGLETON X 2
OBJECT SINGLETON
OBJECT SINGLETON X 2
MODULE SINGLETON
MODULE SINGLETON X 2
CLASS SINGLETON
CLASS SINGLETON X 2


KERNEL
klass
super
BASIC OBJECT
SINGLETON
BASIC OBJECT
SINGLETON X2
BASIC OBJECT
SINGLETON X3
OBJECT
SINGLETON
OBJECT
SINGLETON X2
OBJECT
SINGLETON X3
MODULE
SINGLETON
MODULE
SINGLETON X2
MODULE
SINGLETON X3
CLASS
SINGLETON
CLASS
SINGLETON X2
CLASS
SINGLETON X3

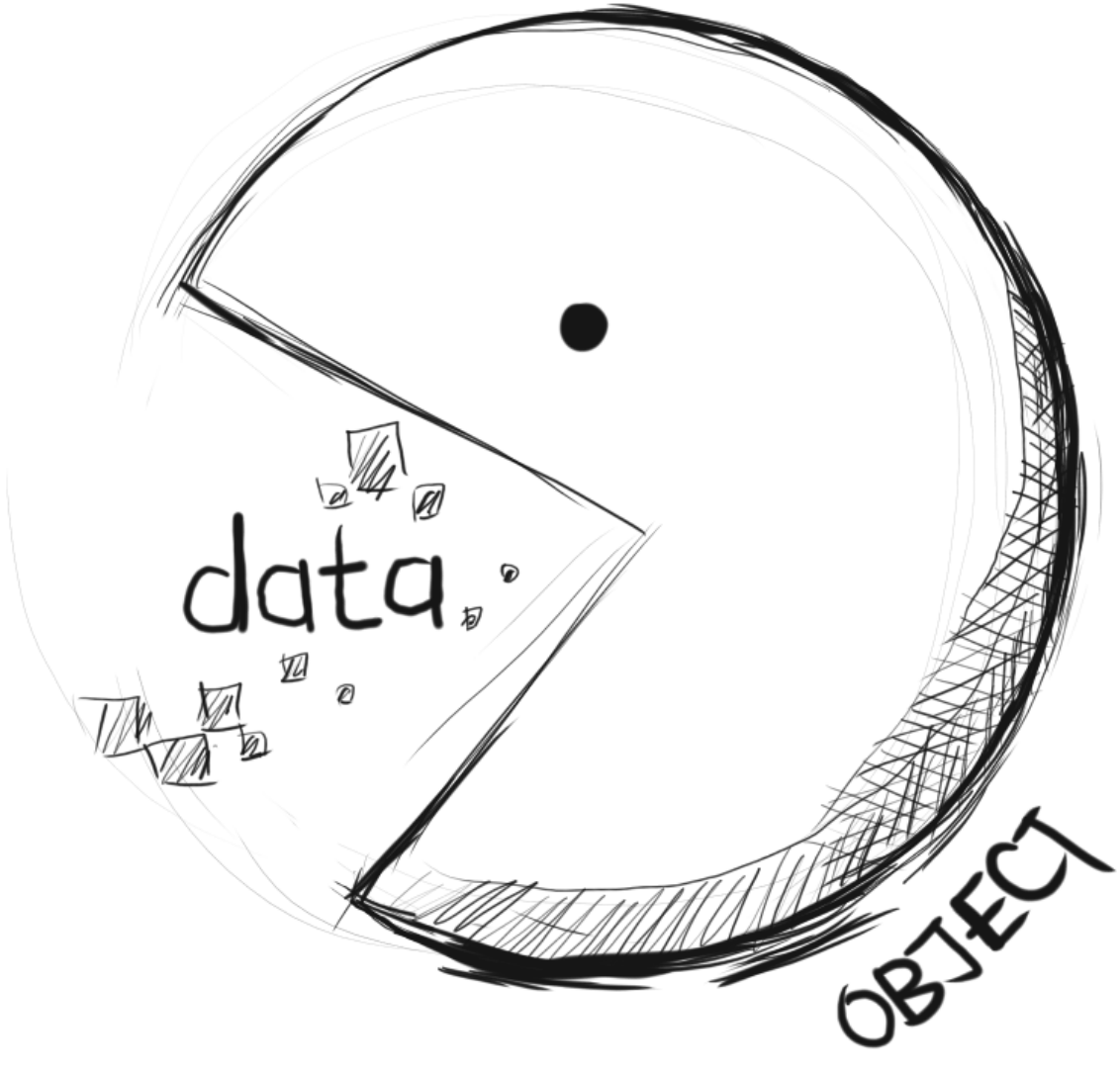


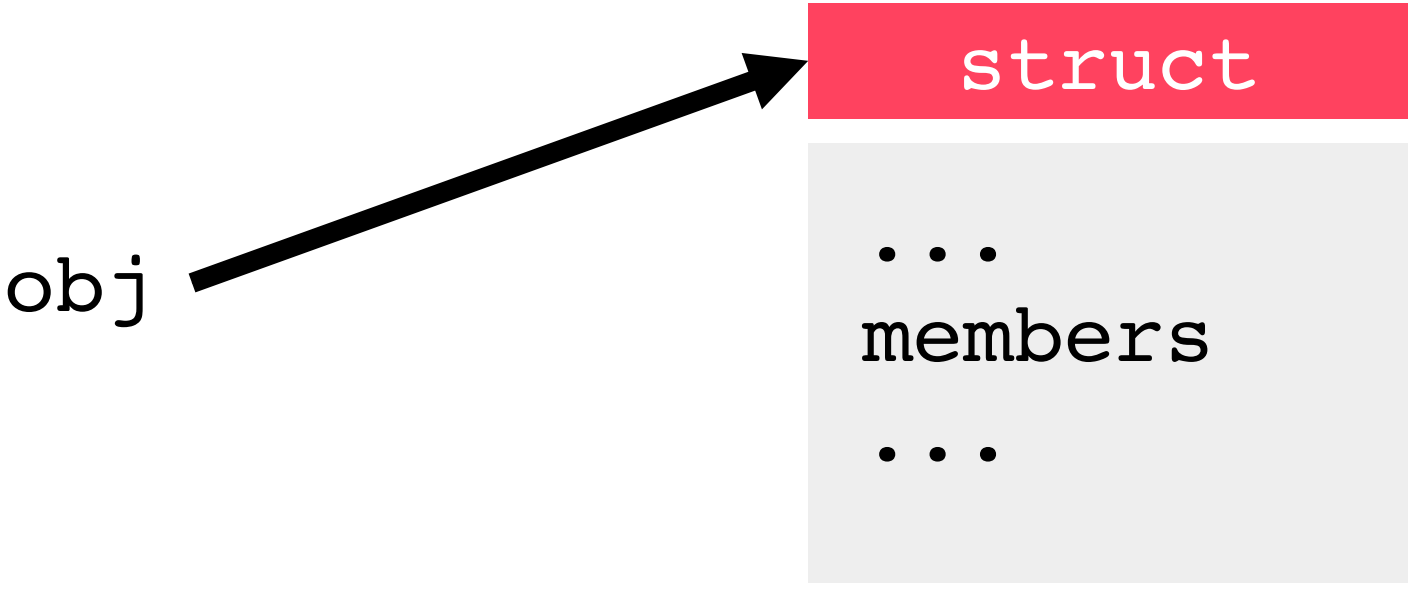
IT'S OVER 9000!!!!

The End


moar
!!!!!


BRACE YOURSELVES
C IS COMING


data
OBJECT


obj
struct
...
members
...

3 structs we care about

```
struct RObject {
  struct RBasic basic;
  union {
    struct {
      uint32_t numiv;
      VALUE *ivptr;
      void *iv_index_tbl;
    } heap;
    VALUE ary[ROBJECT_EMBED_LEN_MAX];
  } as;
};
```

RObject

(include/ruby/ruby.h)

```
struct RClass {
    struct RBasic basic;
    VALUE super;
    rb_classext_t *ptr;
    struct rb_id_table *m_tbl;
};
```

RClass

(internal.h)

```
struct RBasic {
  VALUE flags;
  const VALUE klass;
}
```

RBasic

(include/ruby/ruby.h)

RBasic

```
struct RBasic {
  VALUE flags;
  const VALUE klass;
}
```

Metadata

- *what type of object?* (T_OBJECT, T_CLASS, T_MODULE etc.)
- *singleton class?* (FL_SINGLETON)

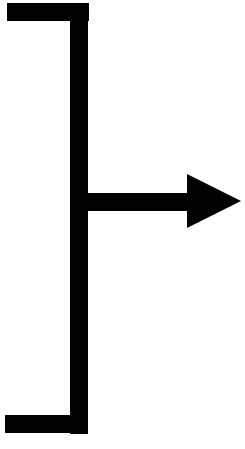
'True' class

- *pointer to RClass struct*

RObject

RBasic plus stuff

```
struct RObject {
  struct RBasic basic;
  union {
    struct {
      uint32_t numiv;
      VALUE *ivptr;
      void *iv_index_tbl;
    } heap;
    VALUE ary[ROBJECT_EMBED_LEN_MAX];
  } as;
};
```



Instance variables

- *pointer to the heap*

OR

- *embedded directly*

RClass

Normal objects are stored as RObject, class objects as RClass

```
struct RClass {
    struct RBasic basic;
    VALUE super;
    rb_classext_t *ptr;
    struct rb_id_table *m_tbl;
};
```

'True' superclass

- *pointer to RClass struct*

Class Extension Struct

- class instance variables
- class constants...

Method table

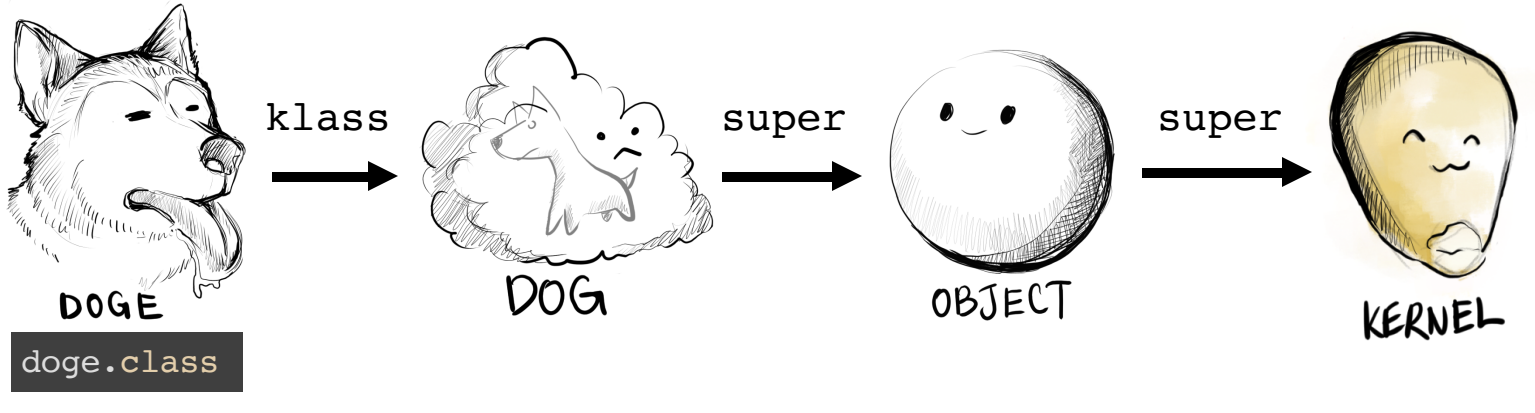
- *store instance methods*

Method Dispatch

```c
static inline rb_method_entry_t*
search_method(VALUE klass, ID id, VALUE *defined_class_ptr)
{
    rb_method_entry_t *me;

    for (me = 0; klass; klass = RCLASS_SUPER(klass)) {
        if ((me = lookup_method_table(klass, id)) != 0) break;
    }

    if (defined_class_ptr)
        *defined_class_ptr = klass;
    return me;
}
```

(vm_method.c)

Keep searching method tables of **supers** till we find something



doge.class

Where does it begin?

Bootstrap

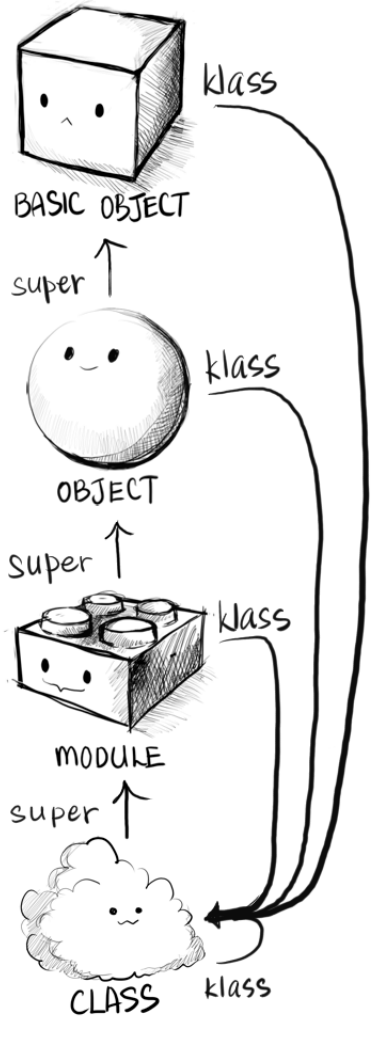
```
void
InitVM_Object(void)
{
    Init_class_hierarchy();

#undef rb_intern
#define rb_intern(str) rb_intern_const(str)

    rb_define_private_method(rb_cBasicObject, "initialize", rb_obj_dummy, 0);
    rb_define_alloc_func(rb_cBasicObject, rb_class_allocate_instance);
    rb_define_method(rb_cBasicObject, "==", rb_obj_equal, 1);
    rb_define_method(rb_cBasicObject, "equal?", rb_obj_equal, 1);
    rb_define_method(rb_cBasicObject, "!", rb_obj_not, 0);
    rb_define_method(rb_cBasicObject, "!=", rb_obj_not_equal, 1);

    rb_define_private_method(rb_cBasicObject, "singleton_method_added", rb_obj_dummy, 1);
    rb_define_private_method(rb_cBasicObject, "singleton_method_removed", rb_obj_dummy, 1);
    rb_define_private_method(rb_cBasicObject, "singleton_method_undefined", rb_obj_dummy, 1);

    // ... really long function ...
```

(object.c)

Bootstrap

```
void
Init_class_hierarchy(void)
{
  rb_cBasicObject = boot_defclass("BasicObject", 0);
  rb_cObject = boot_defclass("Object", rb_cBasicObject);
  rb_gc_register_mark_object(rb_cObject);

  /* resolve class name ASAP for order-independence */
  rb_class_name(rb_cObject);

  rb_cModule = boot_defclass("Module", rb_cObject);
  rb_cClass =  boot_defclass("Class",  rb_cModule);

  rb_const_set(rb_cObject,
  rb_intern_const("BasicObject"), rb_cBasicObject);

  RBASIC_SET_CLASS(rb_cClass, rb_cClass);
  RBASIC_SET_CLASS(rb_cModule, rb_cClass);
  RBASIC_SET_CLASS(rb_cObject, rb_cClass);
  RBASIC_SET_CLASS(rb_cBasicObject, rb_cClass);
}
```

(class.c)

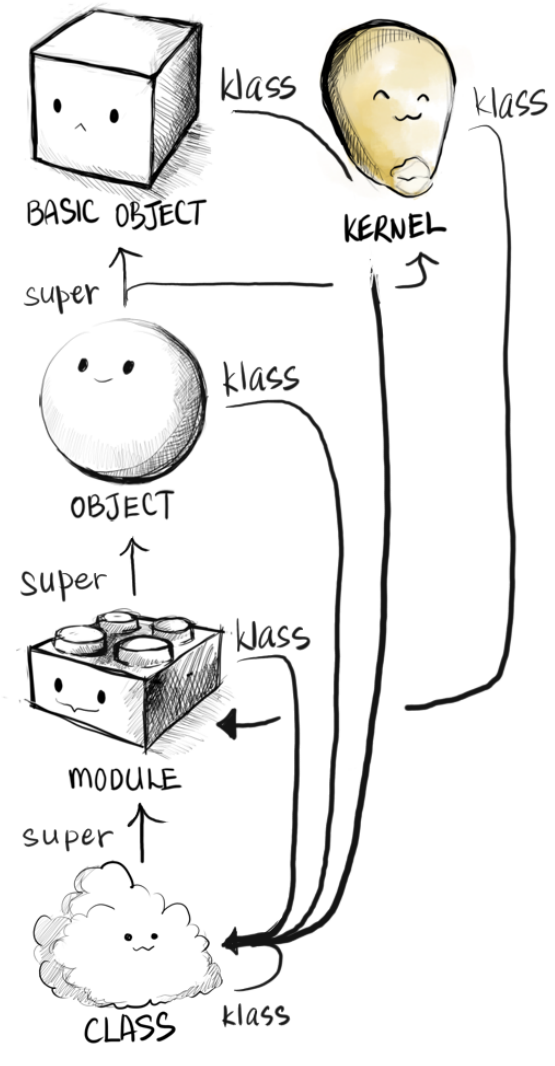


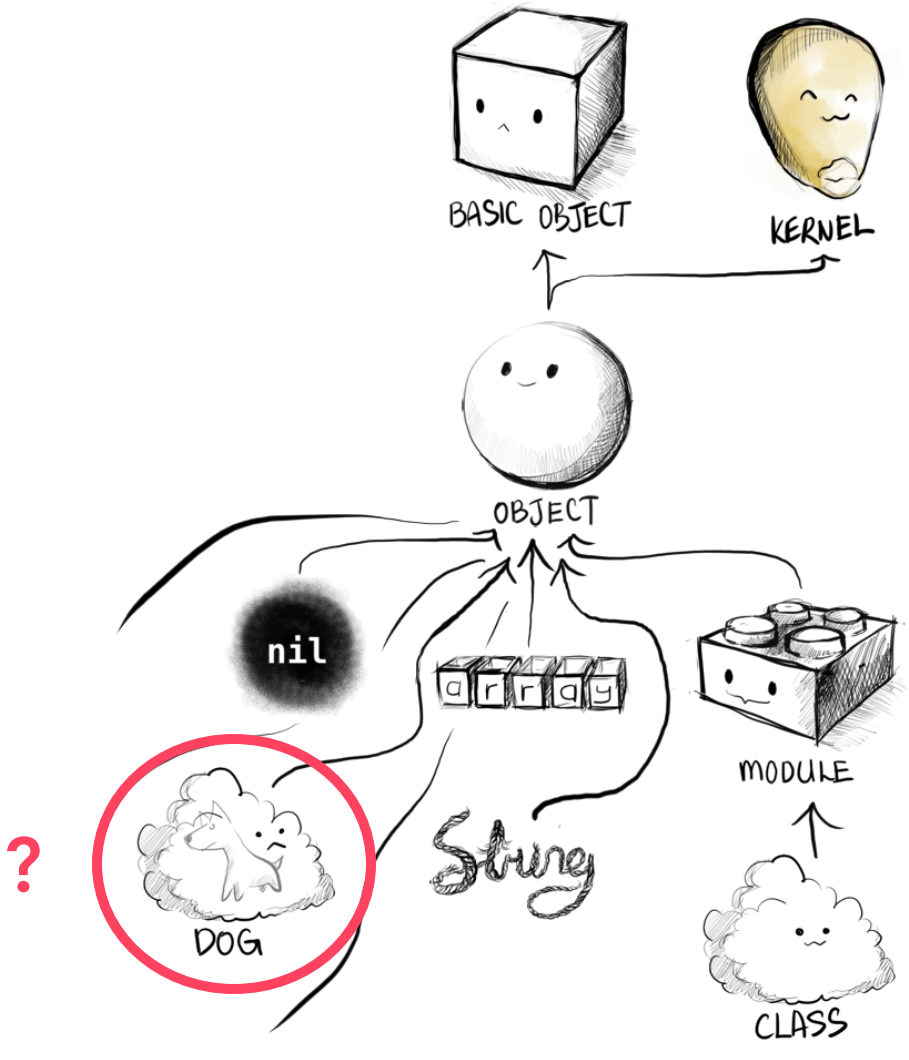
Where's Kernel?

Bootstrap

```
void
InitVM_Object(void)
{
    Init_class_hierarchy();

    // ... bunch of stuff ...

    rb_mKernel = rb_define_module("Kernel");
    rb_include_module(rb_cObject, rb_mKernel);

    // ... really long function ...
```

(object.c)




BASIC OBJECT
KERNEL
OBJECT
nil
array
MODULE
?
DOG
String
CLASS

Make me a Dog

```
VALUE
rb_define_class_id(ID id, VALUE super)
{
    VALUE klass;

    if (!super) super = rb_cObject;
    klass = rb_class_new(super);
    rb_make_metaclass(klass, RBASIC(super)->klass);

    return klass;
}
```

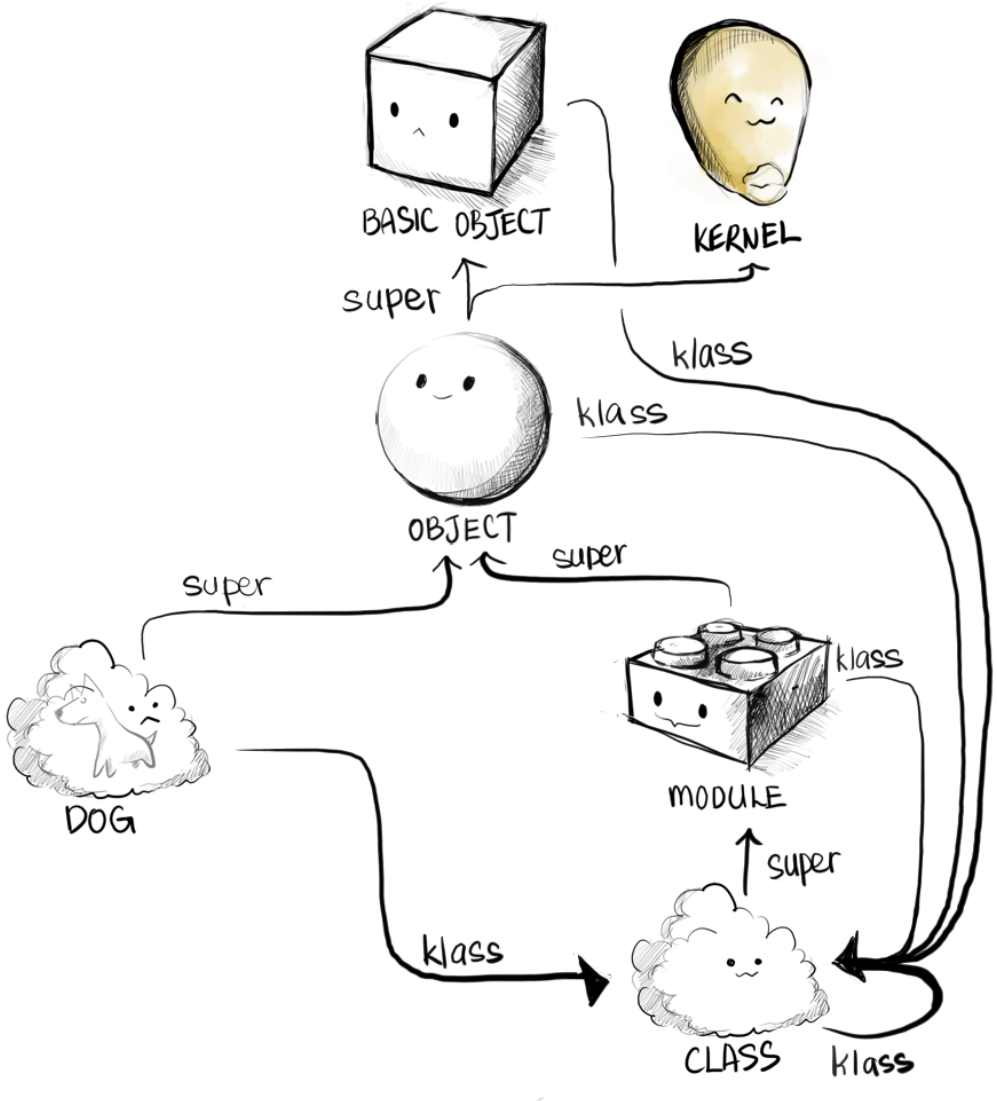
(class.c)

Default super is **Object**

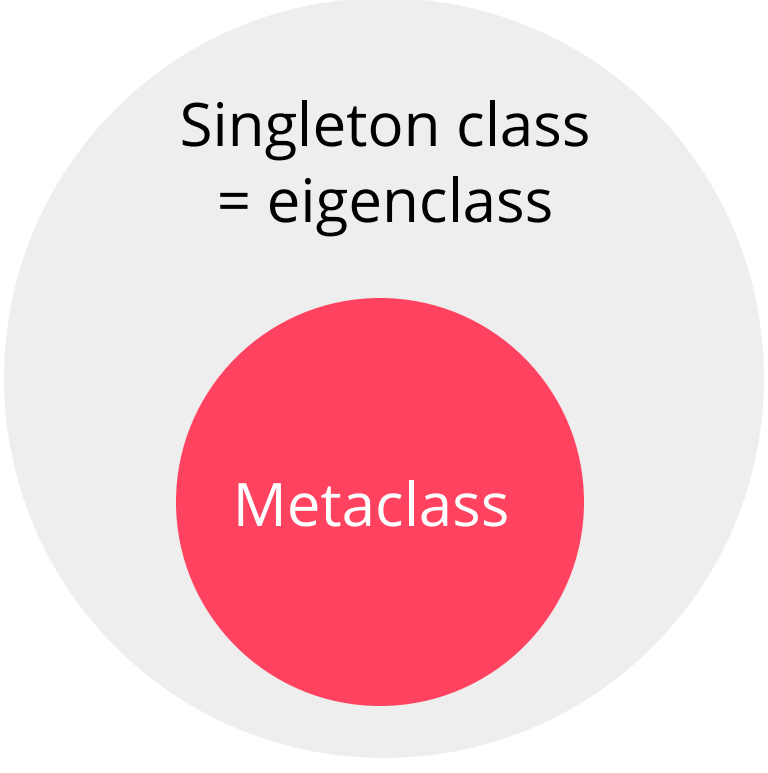
Create the new RClass struct with given **super**

Make and set Dog's **metaclass** straightaway

(ignore the second parameter, it is unused)


BASIC OBJECT
KERNEL
super
klass
klass
OBJECT
super
super
klass
DOG
MODULE
super
klass
CLASS
klass

Terminology...



Terminology...

Metaclasses are also classes, so...

Metametaclasses

Metametametaclasses

Metametametametaclasses

Meta^(n)classes


ONE DOES NOT SIMPLY
MAKE A METACLASS

Make me a <Dog>

(class.c)

```
static inline VALUE make_metaclass(VALUE klass)
{
  VALUE super;
  VALUE metaclass = rb_class_boot(Qundef);

  FL_SET(metaclass, FL_SINGLETON);
  rb_singleton_class_attached(metaclass, klass);

  if (META_CLASS_OF_CLASS_CLASS_P(klass)) {
    SET_METACLASS_OF(klass, metaclass);
    SET_METACLASS_OF(metaclass, metaclass);
  }
  else {
    VALUE tmp = METACLASS_OF(klass);
    SET_METACLASS_OF(klass, metaclass);
    SET_METACLASS_OF(metaclass, ENSURE_EIGENCLASS(tmp));
  }

  super = RCLASS_SUPER(klass);
  while (RB_TYPE_P(super, T_ICLASS)) super = RCLASS_SUPER(super);
  RCLASS_SET_SUPER(metaclass, super ? ENSURE_EIGENCLASS(super) : rb_cClass);

  OBJ_INFECT(metaclass, RCLASS_SUPER(metaclass));

  return metaclass;
}
```

1

Initialize metaclass as new RClass struct; set singleton flag; (attach Dog class to metaclass)

2

Set Dog's **klass** to point to the new metaclass;

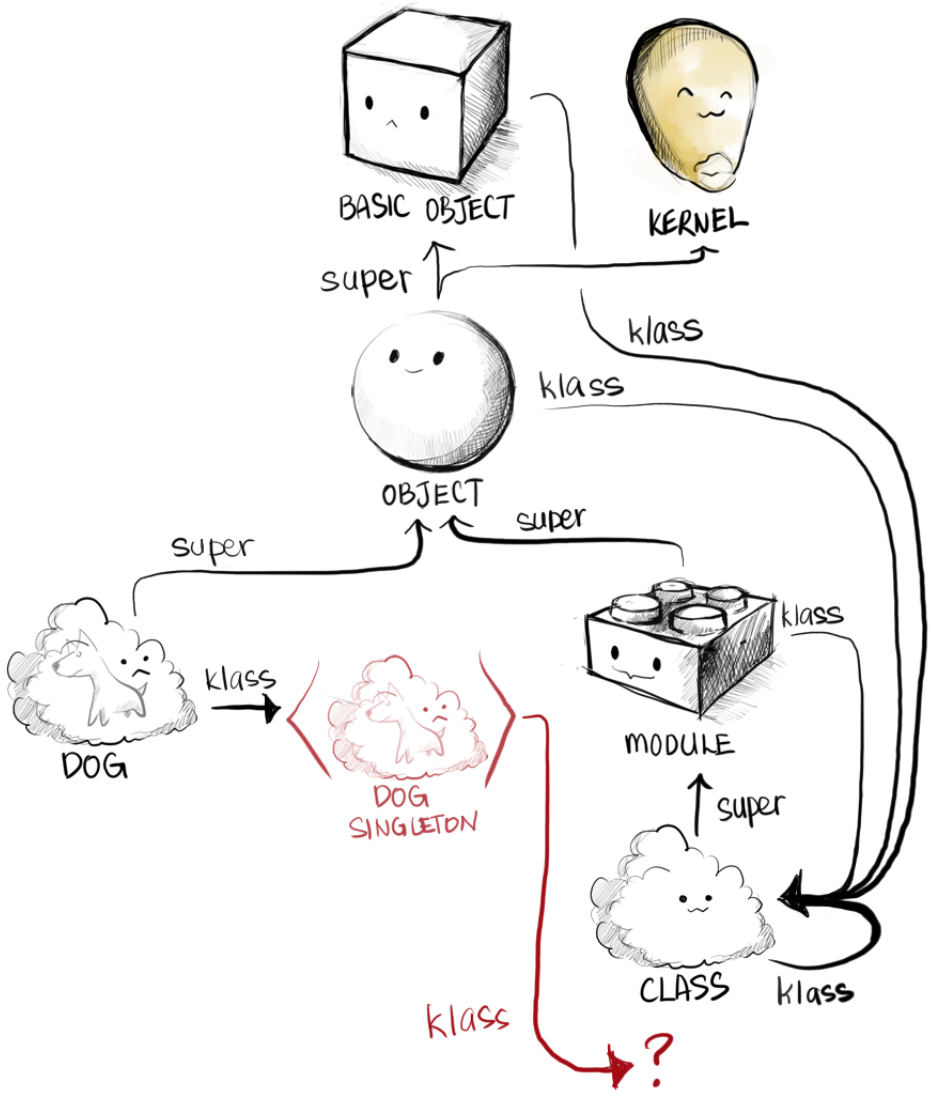
Set metclass's **klass** to point to...what?

3

Set metaclass's **super** to point to...what?

Make me a <Dog>

```
static inline VALUE make_metaclass(VALUE klass)
{
  VALUE super;
  VALUE metaclass = rb_class_boot(Qundef);

  FL_SET(metaclass, FL_SINGLETON);
  rb_singleton_class_attached(metaclass, klass);

  if (META_CLASS_OF_CLASS_CLASS_P(klass)) {
    SET_METACLASS_OF(klass, metaclass);
    SET_METACLASS_OF(metaclass, metaclass);
  }
  else {
    VALUE tmp = METACLASS_OF(klass);
    SET_METACLASS_OF(klass, metaclass);
    /* SET_METACLASS_OF(metaclass, ENSURE_EIGENCLASS(tmp)); */
  }

  super = RCLASS_SUPER(klass);
  while (RB_TYPE_P(super, T_ICLASS)) super = RCLASS_SUPER(super);
  RCLASS_SET_SUPER(metaclass, super ? ENSURE_EIGENCLASS(super) : rb_cClass);

  OBJ_INFECT(metaclass, RCLASS_SUPER(metaclass));

  return metaclass;*/
}
```


BASIC OBJECT
KERNEL
super
klass
klass
OBJECT
super
super
klass
DOG
klass
DOG
SINGLETON
MODULE
super
CLASS
klass
klass
?

Make me a <Dog>

```
static inline VALUE make_metaclass(VALUE klass)
{
  VALUE super;
  VALUE metaclass = rb_class_boot(Qundef);

  FL_SET(metaclass, FL_SINGLETON);
  rb_singleton_class_attached(metaclass, klass);

  if (META_CLASS_OF_CLASS_CLASS_P(klass)) {
    SET_METACLASS_OF(klass, metaclass);
    SET_METACLASS_OF(metaclass, metaclass);
  }
  else {
    VALUE tmp = METACLASS_OF(klass);
    SET_METACLASS_OF(klass, metaclass);
    SET_METACLASS_OF(metaclass, ENSURE_EIGENCLASS(tmp));
  }

  super = RCLASS_SUPER(klass);
  while (RB_TYPE_P(super, T_ICLASS)) super = RCLASS_SUPER(super);
  RCLASS_SET_SUPER(metaclass, super ? ENSURE_EIGENCLASS(super) : rb_cClass);

  OBJ_INFECT(metaclass, RCLASS_SUPER(metaclass));

  return metaclass;
}
```

2

Set Dog's **klass** to point to the new metaclass;

Set metclass's **klass** to point to...what?

Make me a <Dog>

```
static inline VALUE make_metaclass(VALUE klass)
{
  VALUE super;
  VALUE metaclass = rb_class_boot(Qundef);

  FL_SET(metaclass, FL_SINGLETON);
  rb_singleton_class_attached(metaclass, klass);

  if (META_CLASS_OF_CLASS_CLASS_P(klass)) {
    SET_METACLASS_OF(klass, metaclass);
    SET_METACLASS_OF(metaclass, metaclass);
  }
  else {
    VALUE tmp = METACLASS_OF(klass);
    SET_METACLASS_OF(klass, metaclass);
    SET_METACLASS_OF(metaclass, ENSURE_EIGENCLASS(tmp));
  }

  super = RCLASS_SUPER(klass);
  while (RB_TYPE_P(super, T_ICLASS)) super = RCLASS_SUPER(super);
  RCLASS_SET_SUPER(metaclass, super ? ENSURE_EIGENCLASS(super) :

  OBJ_INFECT(metaclass, RCLASS_SUPER(metaclass));

  return metaclass;
}
```

2

tmp is whatever Dog's **klass** points to before we set it to the new metaclass.

`tmp = Class`

This macro calls **`make_metaclass(Class)`**

The Dog metaclass's **klass** points to the *Class metaclass!*

Make me a <Class>

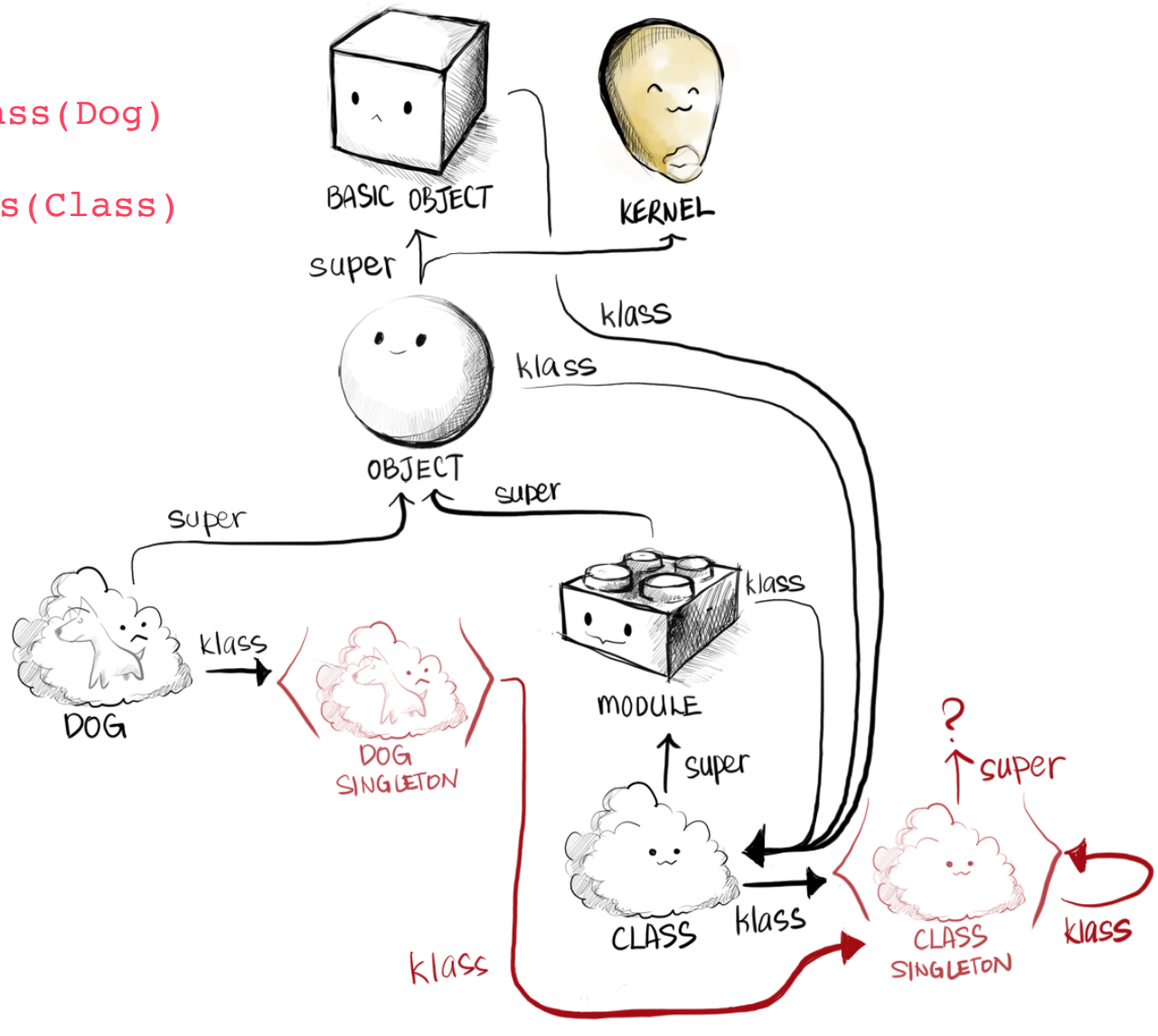
```
static inline VALUE make_metaclass(VALUE klass)
{
  VALUE super;
  VALUE metaclass = rb_class_boot(Qundef);

  FL_SET(metaclass, FL_SINGLETON);
  rb_singleton_class_attached(metaclass, klass);

  if (META_CLASS_OF_CLASS_CLASS_P(klass)) {
    SET_METACLASS_OF(klass, metaclass);
    SET_METACLASS_OF(metaclass, metaclass);
  }
  else {
    VALUE tmp = METACLASS_OF(klass);
    SET_METACLASS_OF(klass, metaclass);
    SET_METACLASS_OF(metaclass, ENSURE_EIGENCLASS(tmp));
  }

  super = RCLASS_SUPER(klass);
  while (RB_TYPE_P(super, T_ICLASS)) super = RCLASS_SUPER(super);
  RCLASS_SET_SUPER(metaclass, super ? ENSURE_EIGENCLASS(super) : rb_cClass);

  OBJ_INFECT(metaclass, RCLASS_SUPER(metaclass));

  return metaclass;
}
```

2

The Class metaclass's **klass** points to itself!

(just like how Class's **klass** used to point to itself)

```
make_metaclass(Dog)
        ↓
make_metaclass(Class)
```



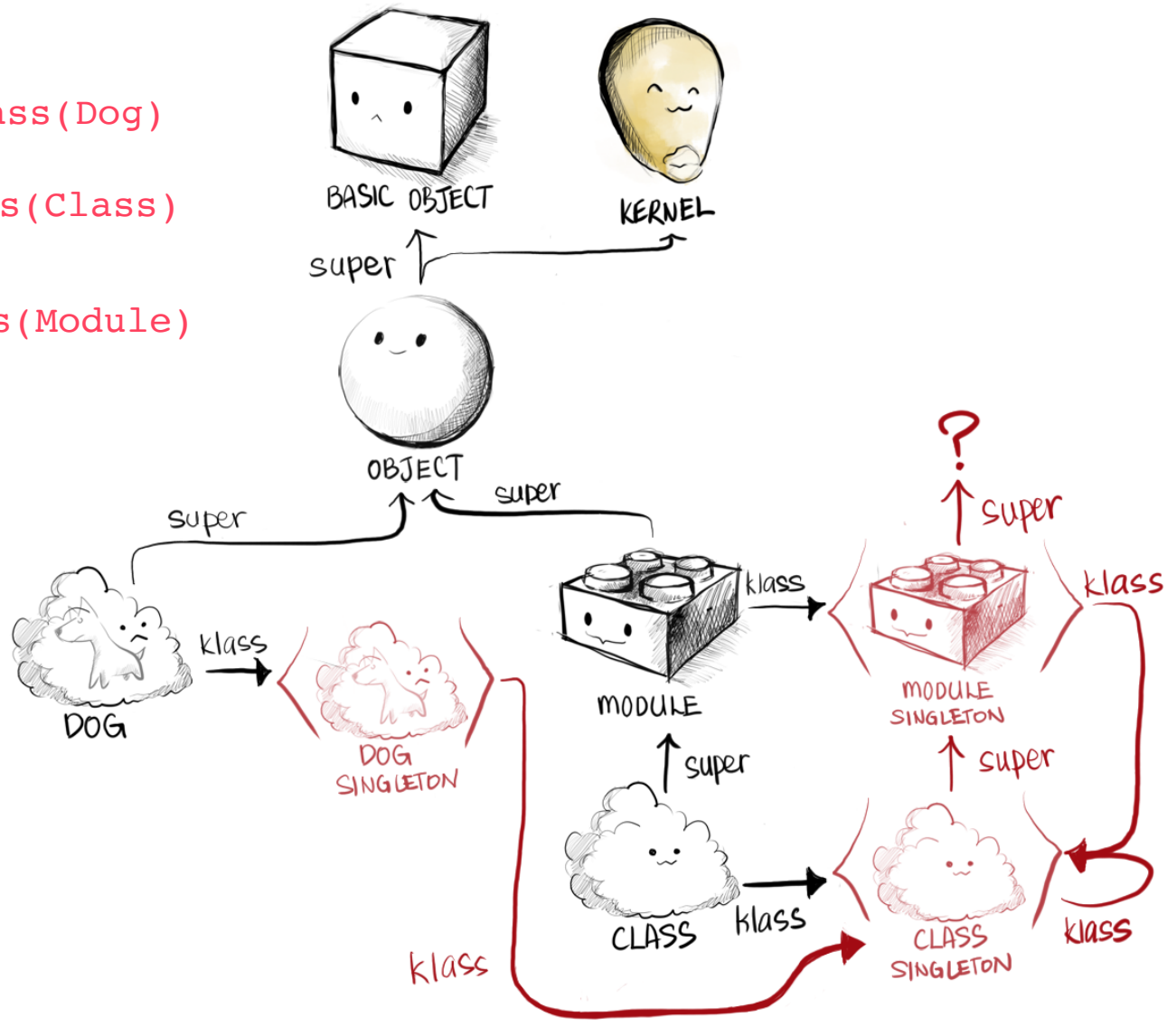
Make me a <Class>

```
static inline VALUE make_metaclass(VALUE klass)
{
  VALUE super;
  VALUE metaclass = rb_class_boot(Qundef);

  FL_SET(metaclass, FL_SINGLETON);
  rb_singleton_class_attached(metaclass, klass);

  if (META_CLASS_OF_CLASS_CLASS_P(klass)) {
    SET_METACLASS_OF(klass, metaclass);
    SET_METACLASS_OF(metaclass, metaclass);
  }
  else {
    VALUE tmp = METACLASS_OF(klass);
    SET_METACLASS_OF(klass, metaclass);
    SET_METACLASS_OF(metaclass, ENSURE_EIGENCLASS(tmp));
  }

  super = RCLASS_SUPER(klass);
  while (RB_TYPE_P(super, T_ICLASS)) super = RCLASS_SUPER(super);
  RCLASS_SET_SUPER(metaclass, super ? ENSURE_EIGENCLASS(super) : rb_cClass);

  OBJ_INFECT(metaclass, RCLASS_SUPER(metaclass));

  return metaclass;
}
```

The **superclass** of the Class metaclass is the ***metaclass*** *of Class's superclass*

3

```
make_metaclass(Dog)
        ↓
make_metaclass(Class)
        ↓
make_metaclass(Module)
```



make_metaclass(Dog)

↓

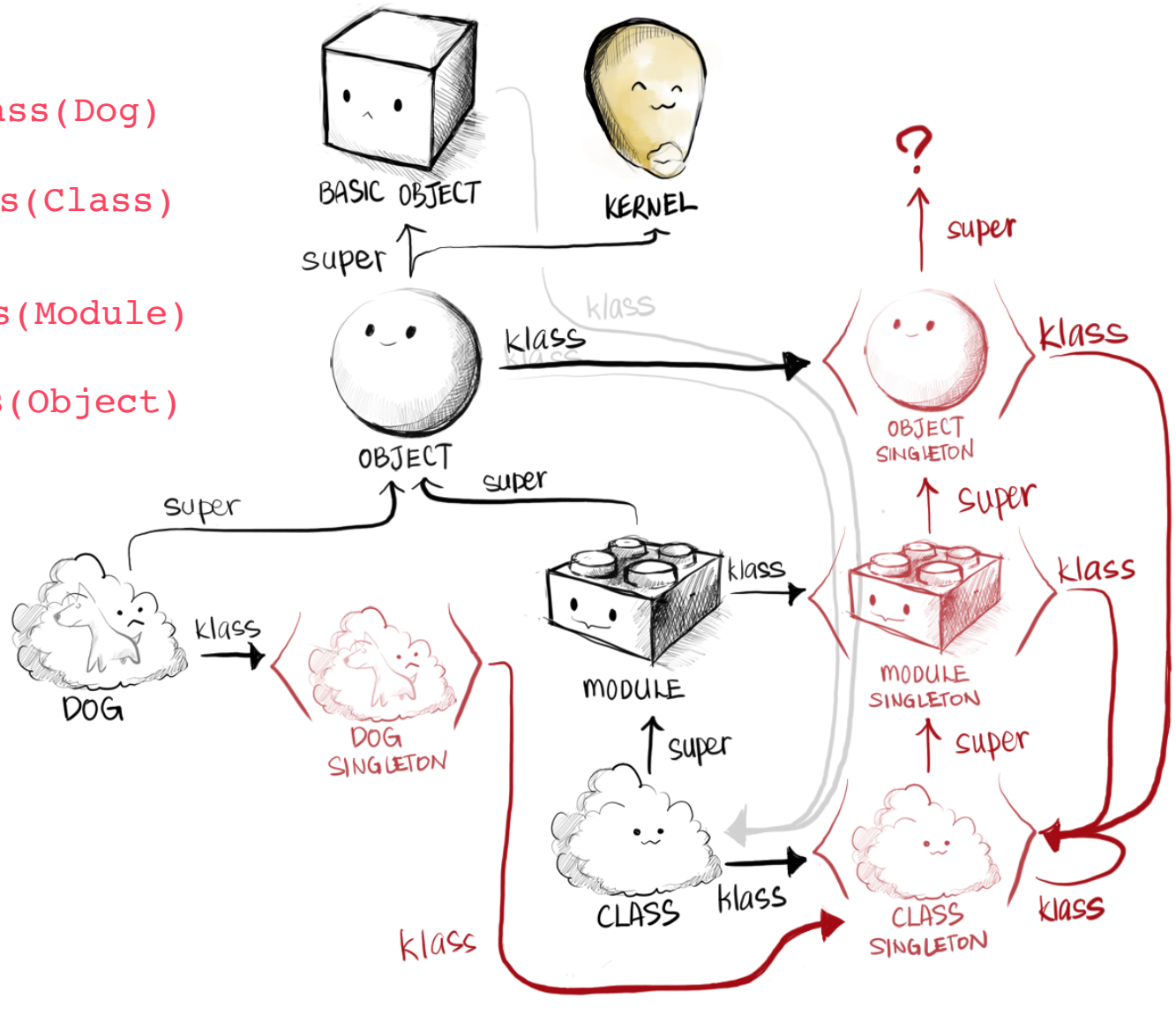
make_metaclass(Class)

↓

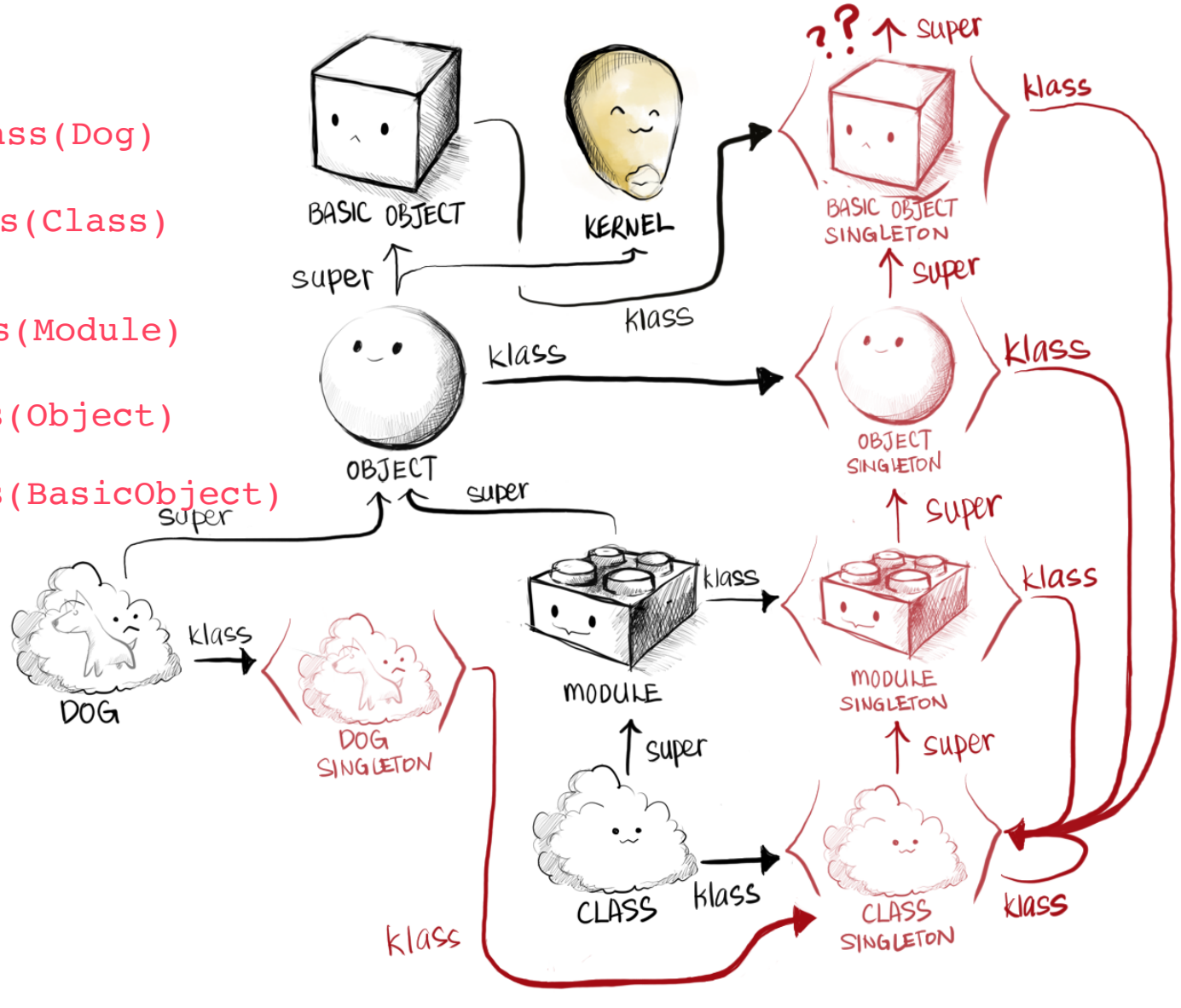
make_metaclass(Module)

↓

make_metaclass(Object)



```
make_metaclass(Dog)
        ↓
make_metaclass(Class)
        ↓
make_metaclass(Module)
        ↓
make_metaclass(Object)
        ↓
make_metaclass(BasicObject)
```



super of <BasicObject>?

```
static inline VALUE make_metaclass(VALUE klass)
{
  VALUE super;
  VALUE metaclass = rb_class_boot(Qundef);

  FL_SET(metaclass, FL_SINGLETON);
  rb_singleton_class_attached(metaclass, klass);

  if (META_CLASS_OF_CLASS_CLASS_P(klass)) {
    SET_METACLASS_OF(klass, metaclass);
    SET_METACLASS_OF(metaclass, metaclass);
  }
  else {
    VALUE tmp = METACLASS_OF(klass);
    SET_METACLASS_OF(klass, metaclass);
    SET_METACLASS_OF(metaclass, ENSURE_EIGENCLASS(tmp));
  }

  super = RCLASS_SUPER(klass);
  while (RB_TYPE_P(super, T_ICLASS)) super = RCLASS_SUPER(super);
  RCLASS_SET_SUPER(metaclass, super ? ENSURE_EIGENCLASS(super) : rb_cClass);

  OBJ_INFECT(metaclass, RCLASS_SUPER(metaclass));

  return metaclass;
}
```

The **superclass** of the BasicObject metaclass is *Class*

3

super of <Dog>?

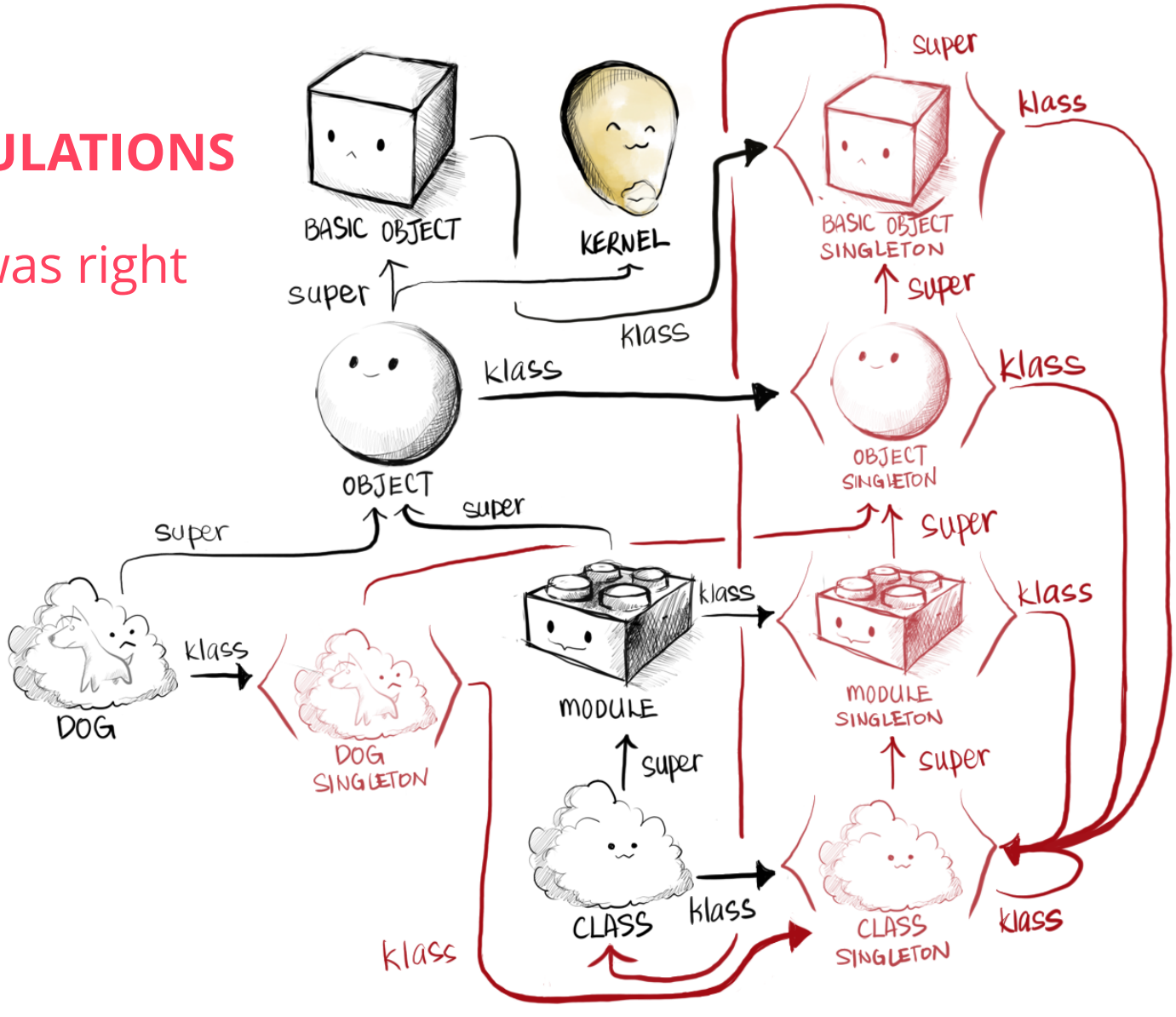
```
static inline VALUE make_metaclass(VALUE klass)
{
  VALUE super;
  VALUE metaclass = rb_class_boot(Qundef);

  FL_SET(metaclass, FL_SINGLETON);
  rb_singleton_class_attached(metaclass, klass);

  if (META_CLASS_OF_CLASS_CLASS_P(klass)) {
    SET_METACLASS_OF(klass, metaclass);
    SET_METACLASS_OF(metaclass, metaclass);
  }
  else {
    VALUE tmp = METACLASS_OF(klass);
    SET_METACLASS_OF(klass, metaclass);
    SET_METACLASS_OF(metaclass, ENSURE_EIGENCLASS(tmp));
  }

  super = RCLASS_SUPER(klass);
  while (RB_TYPE_P(super, T_ICLASS)) super = RCLASS_SUPER(super);
  RCLASS_SET_SUPER(metaclass, super ? ENSURE_EIGENCLASS(super) : rb_cClass);

  OBJ_INFECT(metaclass, RCLASS_SUPER(metaclass));

  return metaclass;
}
```

The **superclass** of the Dog metaclass is the ***metaclass*** *of Dog's superclass*

3

CONGRATULATIONS

Boromir was right




BUT YES, WE CAN SIMPLY
MAKE A SINGLETON CLASS

Make me a <doge>

```
static inline VALUE make_singleton_class(VALUE obj)
{
    VALUE orig_class = RBASIC(obj)->klass;
    VALUE klass = rb_class_boot(orig_class);
    FL_SET(klass, FL_SINGLETON);

    RBASIC_SET_CLASS(obj, klass);
    rb_singleton_class_attached(klass, obj);

    SET_METACLASS_OF(klass, METACLASS_OF(rb_class_real(orig_class)));
    return klass;
}
```

1

Initialize singleton class as new RClass struct whose super is the original class; set singleton flag

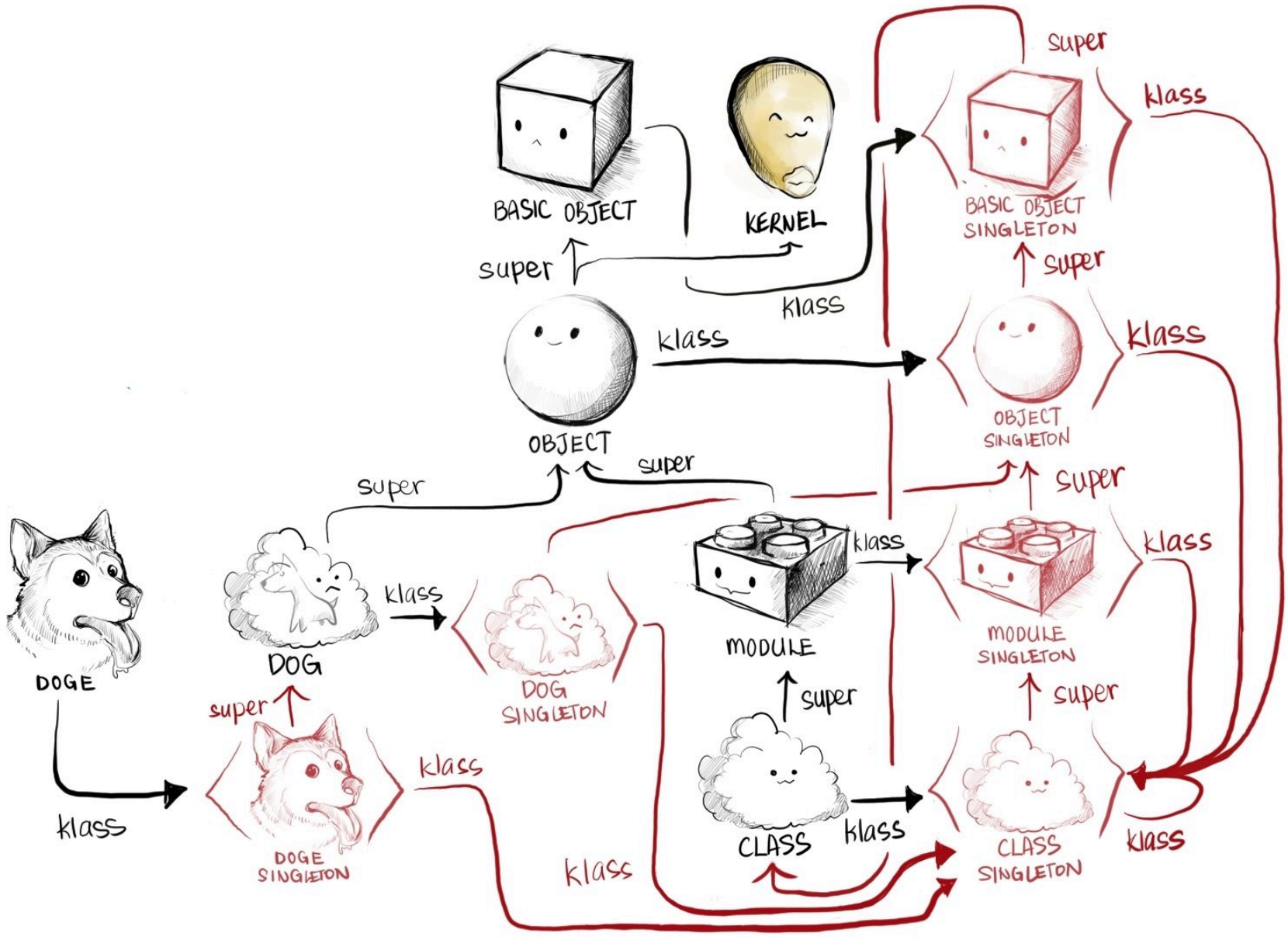
2

Set doge's **klass** to point to the new singleton class

3

Set the doge singleton class's **klass** to point to the *Class metaclass*

(class.c)

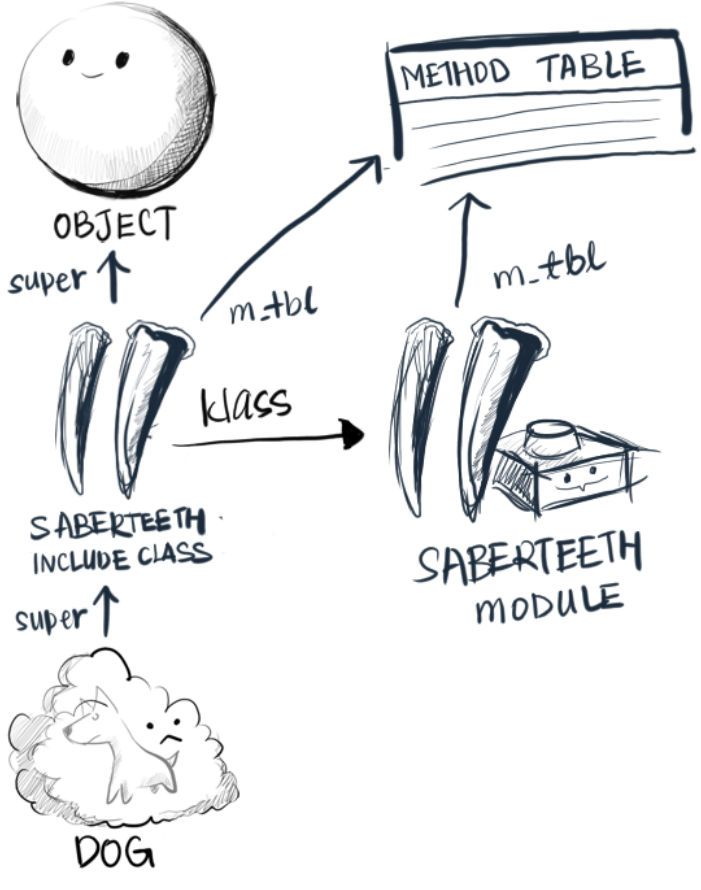

BASIC OBJECT
KERNEL
super
klass
BASIC OBJECT SINGLETON
OBJECT
OBJECT SINGLETON
MODULE
MODULE SINGLETON
DOG
DOG SINGLETON
DOGE
DOGE SINGLETON
CLASS
CLASS SINGLETON

Wait, what about **modules**?

Include Classes

Saberdoge

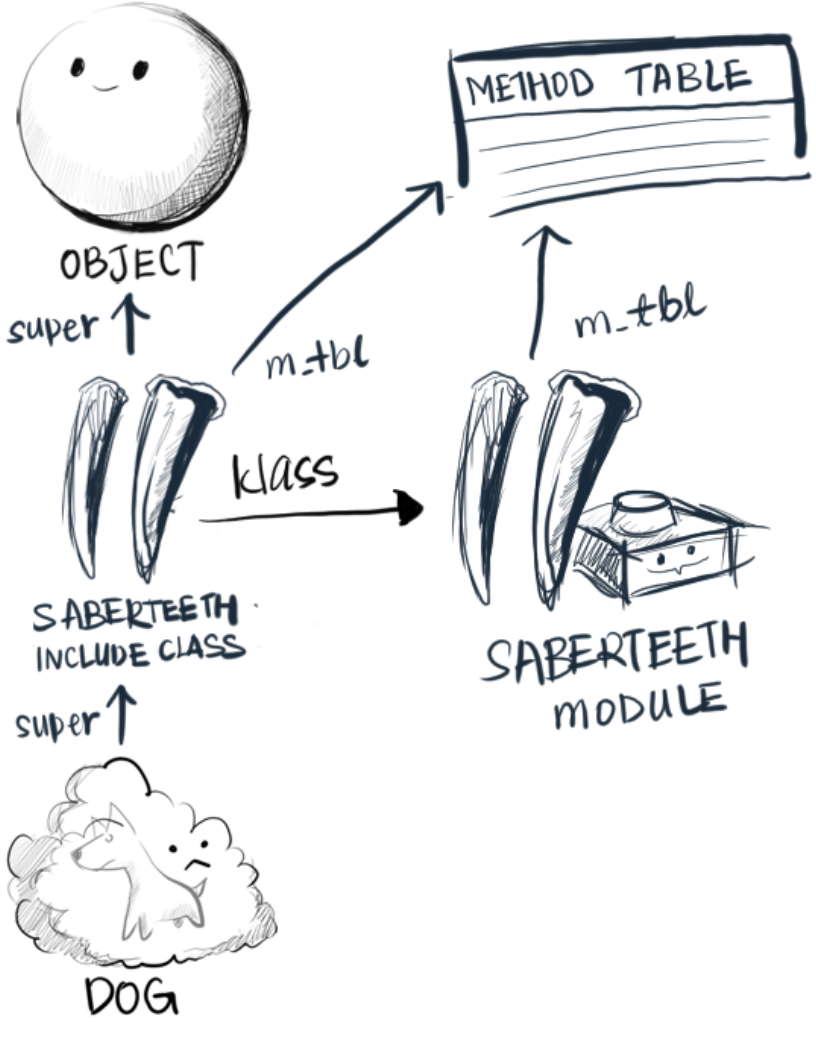
```
class Dog
  include Saberteeth
  # ...
end
```



Make me an include class

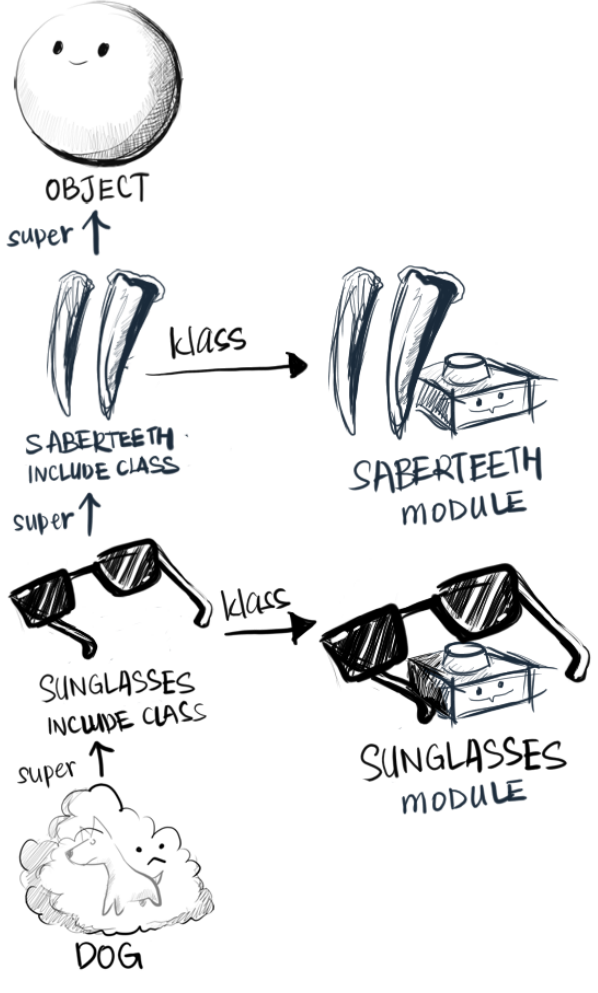
```
static int
include_modules_at(const VALUE klass, VALUE c, VALUE module, int search_super)
{
    // check for duplicate includes, cyclic includes,
    // modules included in modules, modules included in refinements...

    iclass = rb_include_class_new(module, RCLASS_SUPER(c));
    c = RCLASS_SET_SUPER(c, iclass);

    // ...
}
```

(class.c)


METHOD TABLE
OBJECT
super
m_tbl
m_tbl
klass
SABERTEETH
INCLUDE CLASS
SABERTEETH
MODULE
super
DOG

Multiple Include Classes

```
class Dog
  include Saberteeth
  include Sunglasses
  # ...
end
```



Make me an include class

```
static int
include_modules_at(const VALUE klass, VALUE c, VALUE module, int search_super)
{
    // check for duplicate includes, cyclic includes,
    // modules included in modules  modules included in refinements...

    iclass = rb_include_class_new(module, RCLASS_SUPER(c));
    c = RCLASS_SET_SUPER(c, iclass);

    // ...
}
```

(class.c)

Including modules in modules?

Modules and Include Classes are also ***RClass structs***

```
struct RClass {
    struct RBasic basic;
    VALUE super;
    rb_classext_t *ptr;
    struct rb_id_table *m_tbl;
};
```

Can modules have a **super**?

Yes, but not by default

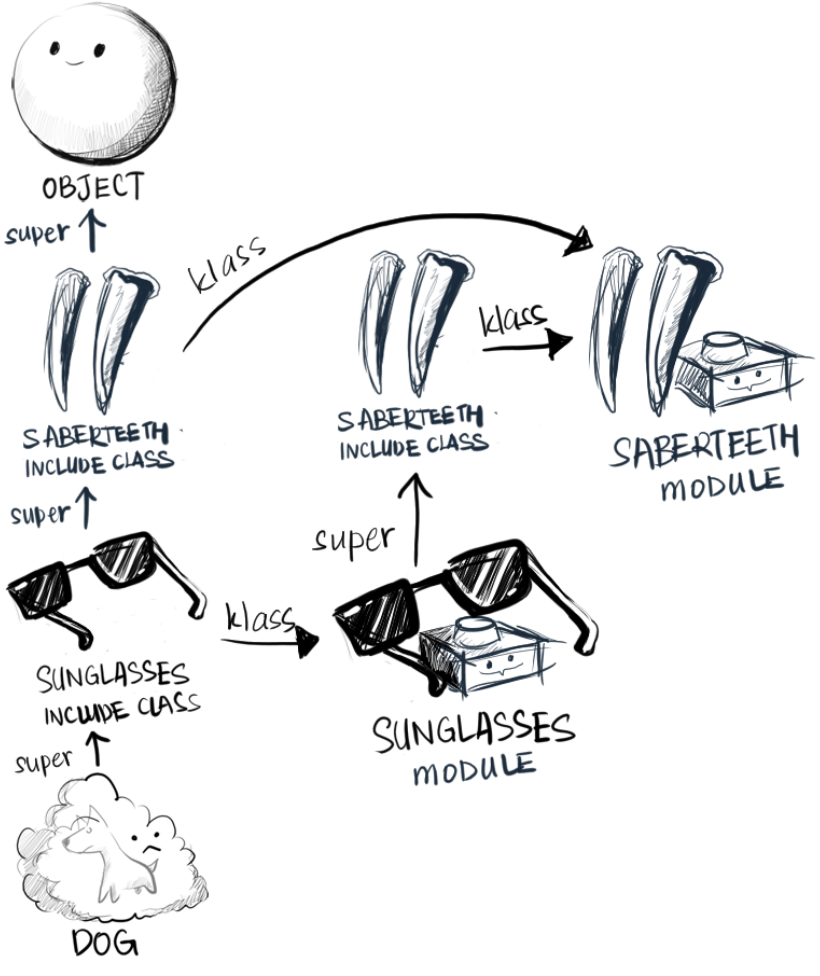
super comes in handy when we include modules in modules

Sabersunglass Doge

```
module Sunglasses
  include Saberteeth
end

class Dog
  include Sunglasses
end
```

Module in a module

```
module Sunglasses
  include Saberteeth
end
```



how objects and classes
are represented

singleton class and
metaclass creation

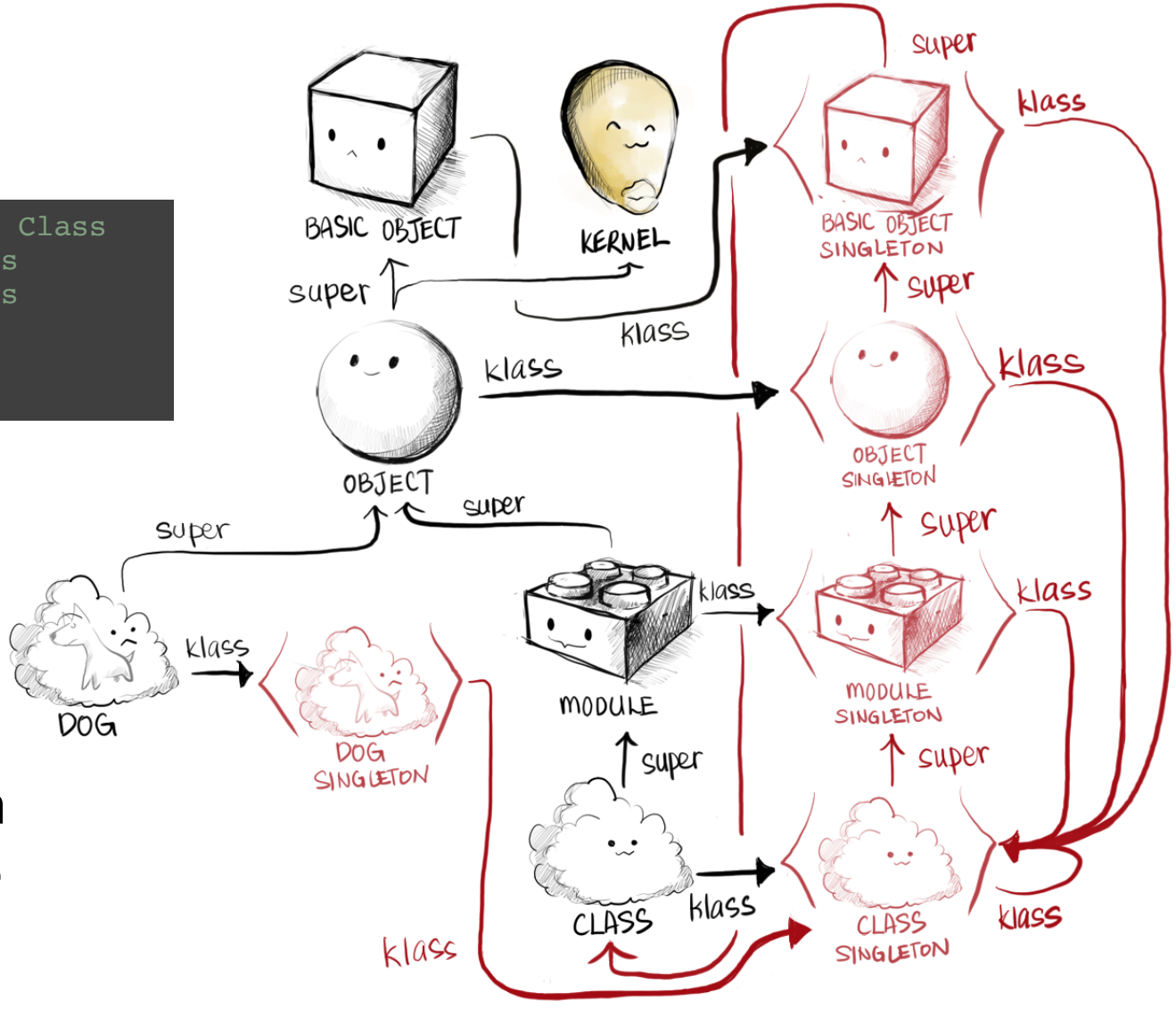
how modules work

Just to clarify...

C	**Ruby**
klass	class
super	superclass

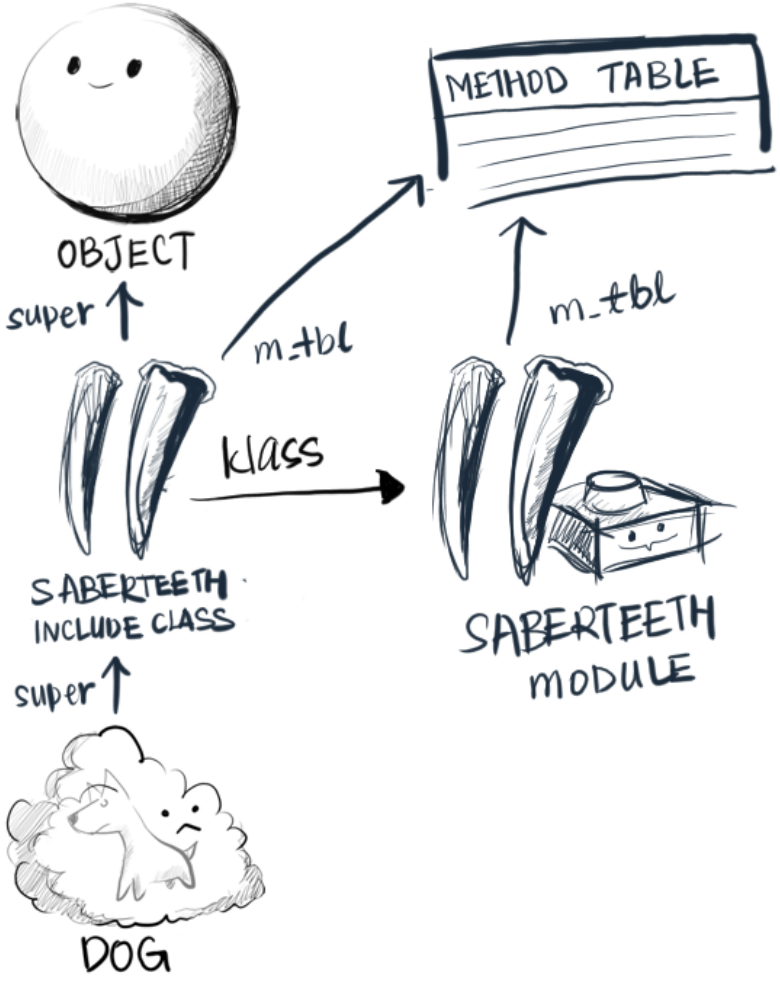
```
BasicObject.class # => Class
Object.class # => Class
Module.class # => Class
Class.class # => Class
Dog.class # => Class
```

Ruby's `class` ignores singleton classes & include classes



```ruby
class Dog
  include Saberteeth
end

Dog.superclass # => Object
```

Ruby's `superclass` ignores include classes




WAIII?

class_eval vs instance_eval

include vs extend (vs prepend)

class << self; end

```
module Sunglasses
  include Saberteeth
end

class Dog
  include Sunglasses
end
```

vs

```
class Dog
  include Sunglasses
end

module Sunglasses
  include Saberteeth
end
```

class variables vs class instance variables

```
module A
  def a_method; end
end

module B
  extend A
end

module C
  extend B
end
```

```
# should I expect this to work??
C.a_method
```


ALL I'D WANTED TO
KNOW ABOUT RUBY'S
OBJECT MODEL


MOAR
!!!!


nil
THANK YOU
arrray