Changing the Rules of the Game

Objective-C Runtime Manipulation

Whitney Young
FadingRed

@wbyoung
wbyoung.github.com

Runtime API

86 *public, open source* C functions

Runtime API

- Classes
- Methods
- Objects
- Properties & instance variables
- Selectors
- Protocols

Example

Simplifying Core Data Query Syntax

Ruby on Rails

```ruby
@articles = Article.
  where(:created_at => start_date..end_date).
  order('created_at')
```

Django

```
self.articles = Article.objects.
  filter(created__gte=start_date,
         created__lte=end_date).
  order_by('created')
```

Core Data

```
NSEntityDescription *entity =
    [NSEntityDescription entityForName:@"Article" inManagedObjectContext:context];
NSFetchRequest *request = [[NSFetchRequest alloc] init];
[request setEntity:entity];
[request setPredicate:
 [NSPredicate predicateWithFormat:@"created >= %@ && created <= %@",
  startDate, endDate]];
[request setSortDescriptors:
 @[[NSSortDescriptor sortDescriptorWithKey:@"created" ascending:YES]]];
NSError *error = nil;
self.articles = [context executeFetchRequest:request error:&error];
```

Core Data + Runtime

```
self.articles = context.query.articles[@{
  @"created__gte": startDate,
  @"created__lte": endDate }][@"^created"];
```

Implementation Goals

- Easy to read and use
- Little to no code overhead for use
- Delayed evaluation
- Cheap copies of query objects

- Good runtime examples

Implementation Plan

- Query operations always return a copy
- Query objects are array proxies
  - Evaluate on first use only
- Queries can be bound (or unbound)
  - Knows the entity it’s trying to fetch

Core Data + Runtime

```
self.articles = context.query.articles[@{
  @"created__gte": startDate,
  @"created__lte": endDate }][@"^created"];
```

Implementation I

- Make query work like an array

```
[query objectAtIndex:0];
```

What we need

- Message forwarding
- Proxy objects

Message Sending

```
[array objectAtIndex:0];
```

Message Sending

```
[array objectAtIndex:0];

objc_msgSend(
  array,
  @selector(objectAtIndex:),
  0);
```

Message Sending

- Search the class's method cache for the `IMP`
- If not found, search the class hierarchy for the `IMP`
- Jump to `IMP` if found (`jmp` assembly instruction)
- If not found, jump to `_objc_msgforward`

- All objects can forward messages they don't respond to

Message Forwarding

```
- (void)forwardInvocation:(NSInvocation *)invocation {
   [invocation setTarget:_realArray];
   [invocation invoke];
}

- (NSMethodSignature *)methodSignatureForSelector:(SEL)sel {
   return [_realArray methodSignatureForSelector:sel];
}
```

Message Forwarding

- Other Options

```
- (id)forwardingTargetForSelector:(SEL)name
+ (BOOL)resolveClassMethod:(SEL)name
+ (BOOL)resolveInstanceMethod:(SEL)name
```

NSProxy

- Designed for use with message forwarding

- Root class
- Implements minimal number of methods
- Used frequently throughout Cocoa

Implementation I

- Make query work like an array

```
[query objectAtIndex:0];
```

Demo

Implementation II

- Return unbound query from managed object context

```
[context query]
```

What we need

- Associated objects

Associated Objects

- Arbitrary key/value storage for any object

```
static void *kSomeKey = &(void *){0};
objc_setAssociatedObject(obj, kSomeKey, val,
                                 OBJC_ASSOCIATION_RETAIN);
objc_getAssociatedObject(obj, kSomeKey);
```

Implementation II

- Return unbound query from managed object context

```
[context query]
```

Demo

Implementation III

- Resolve binding methods dynamically

```
[context.query people]
```

What we need

- Message forwarding
- Adding methods to classes

Adding Methods

```
BOOL class_addMethod(Class class, SEL name,
                     IMP imp, const char *types)
```

- `IMP` is simply a C function
- Type encoding
    - Defines the return & argument types
    - Best retrieved from another method

Adding Methods

```
void MyRuntimeMethod(id self, SEL _cmd, NSString *arg) {
    // implementation
}


Method prototype = class_getInstanceMethod([NSString class],
                                           @selector(appendString:));
char *types = method_getTypeEncoding(prototype);

class_addMethod([MyClass class], @selector(myNewMethod:),
                (IMP)MyRuntimeMethod, types);
```

Implementation III

- Resolve binding methods dynamically

```
[context.query people]
```

Demo

Implementation IV

- Implementing binding method

```
[context.query people]
```

What we need

- Nothing new :)

Demo

Implementation IV

- Pluralization

```
[context.query people]
```

Goal

- Pluralize simple words automatically
- Allow simple override by each model class

Override Example

```
@implementation FRPerson

+ (NSString *)pluralizePerson {
    return @"people";
}

@end
```

Implementation IV

- Pluralization

```
[context.query people]
```

What we need

- Selector names from strings
- Performing selectors

Selectors

- Really just unique C strings

```
SEL sel_getUid(const char *str)
SEL NSSelectorFromString(NSString *string)

const char *sel_getName(SEL selector)
NSString *NSStringFromSelector(SEL selector)
```

Performing Selectors

```
[object performSelector:selector];

[object performSelector:selector
             withObject:arg1
             withObject:arg2];
```

Performing Selectors ARC

```
IMP imp = [object methodForSelector:sel];
void (*func)(id,SEL,CGRect,CGFloat) = (void *)imp;

func(object, sel, rect, float);
```

Performing Selectors ARC

```
Class class = [object class];
IMP imp = class_getMethodImplementation(class, sel);
void (*func)(id,SEL,CGRect,CGFloat) = (void *)imp;

func(object, sel, rect, float);
```

Implementation IV

- Pluralization

```
[context.query people]
```

Demo

Implementation V

- Keyed subscript & other query operations

```
context.query.people[@"name = 'Whitney'"]
```

What we need

- No runtime needed

Explore the code

Example

Mixins

Ruby: Modules

```ruby
class Article < Object
  attr_accessor :created_at
end
module TimeAgo
  def time_ago # more code would go in here
    seconds = Time.now - self.created_at
    "#{seconds.to_i / 3600} hours ago"
  end
end
```

Ruby: Class Mixin

```ruby
class Article
  include TimeAgo
end

a = Article.new
a.created_at = Time.now - 3600 * 3
a.time_ago # "3 hours ago"
```

Ruby: Instance Mixin

```ruby
a = Article.new
a.created_at = Time.now - 3600 * 3
a.extend TimeAgo
a.time_ago # "3 hours ago"
```

Objective-C: Modules

```objc
@interface FRTimeAgo : FRModule
@end

@implementation FRTimeAgo

- (NSString *)timeAgo { // more code would go in here
  NSTimeInterval seconds = -[self.creationDate timeIntervalSinceNow];
  return [NSString stringWithFormat:@"%i hours ago", seconds / 3600];
}

@end
```

Objective-C: Class Mixins

```
[FRTimeAgo extendClass:[FRArticle class]];

FRArticle *a = [[FRArticle alloc] init];
a.creationDate =
  [NSDate dateWithTimeIntervalSinceNow:-3600*3];
[(id)a timeAgo]; // @"3 hours ago"
```

Objective-C: Instance Mixins

```
FRArticle *a = [[FRArticle alloc] init];
a.creationDate =
  [NSDate dateWithTimeIntervalSinceNow:-3600*3];
[FRTimeAgo extendInstance:a];
[(id)a timeAgo]; // @"3 hours ago"
```

Implementation I

- Class Mixins

```
[FRTimeAgo extendClass:[FRArticle class]];
```

What we need

- Method enumeration
- Replacing methods on classes

Method Enumeration

- Copy methods from module to destination class

```
Method *
class_copyMethodList(Class cls,
                     unsigned int *outCount)
```

- Returns methods from just that class
- Allocated memory must be freed

Replacing Methods

```
IMP class_replaceMethod(Class class, SEL name,
                        IMP imp, const char *types)
```

- Similar to `class_addMethod`
- Adds or replaces method
- Returns `IMP` if method was replaced

Implementation I

- Class Mixins

```
[FRTimeAgo extendClass:[FRArticle class]];
```

Demo

Implementation II

- Instance Mixins

```
[FRTimeAgo extendInstance:a];
```

What we need

- Dynamic subclassing
- Changing object class

Dynamic Subclassing

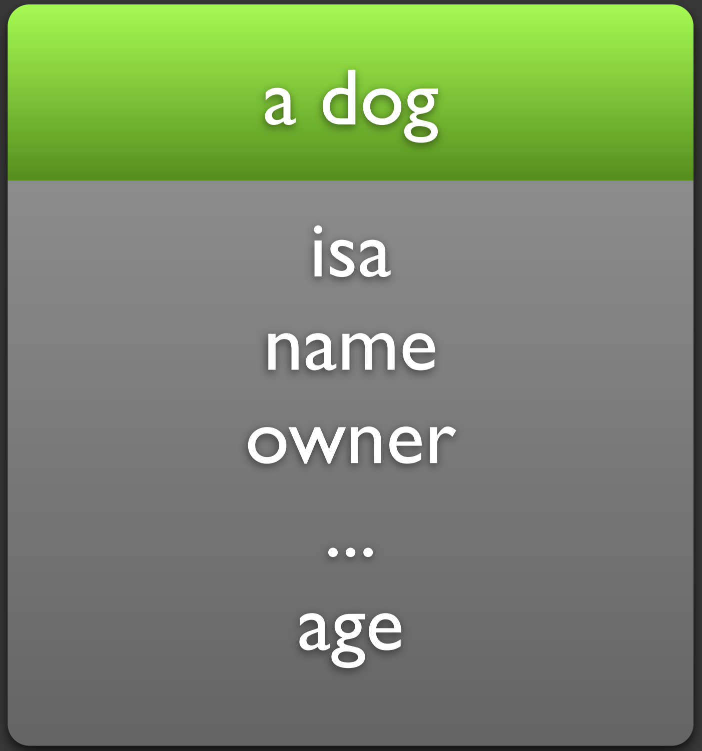
- Why create a dynamic subclass?

- Subclass won't affect instances of main class
- We can safely change the object's class to this class
  - Apple does this with KVO

Let’s step back

Objects & Classes

What are Objects?



What are Objects?

instance

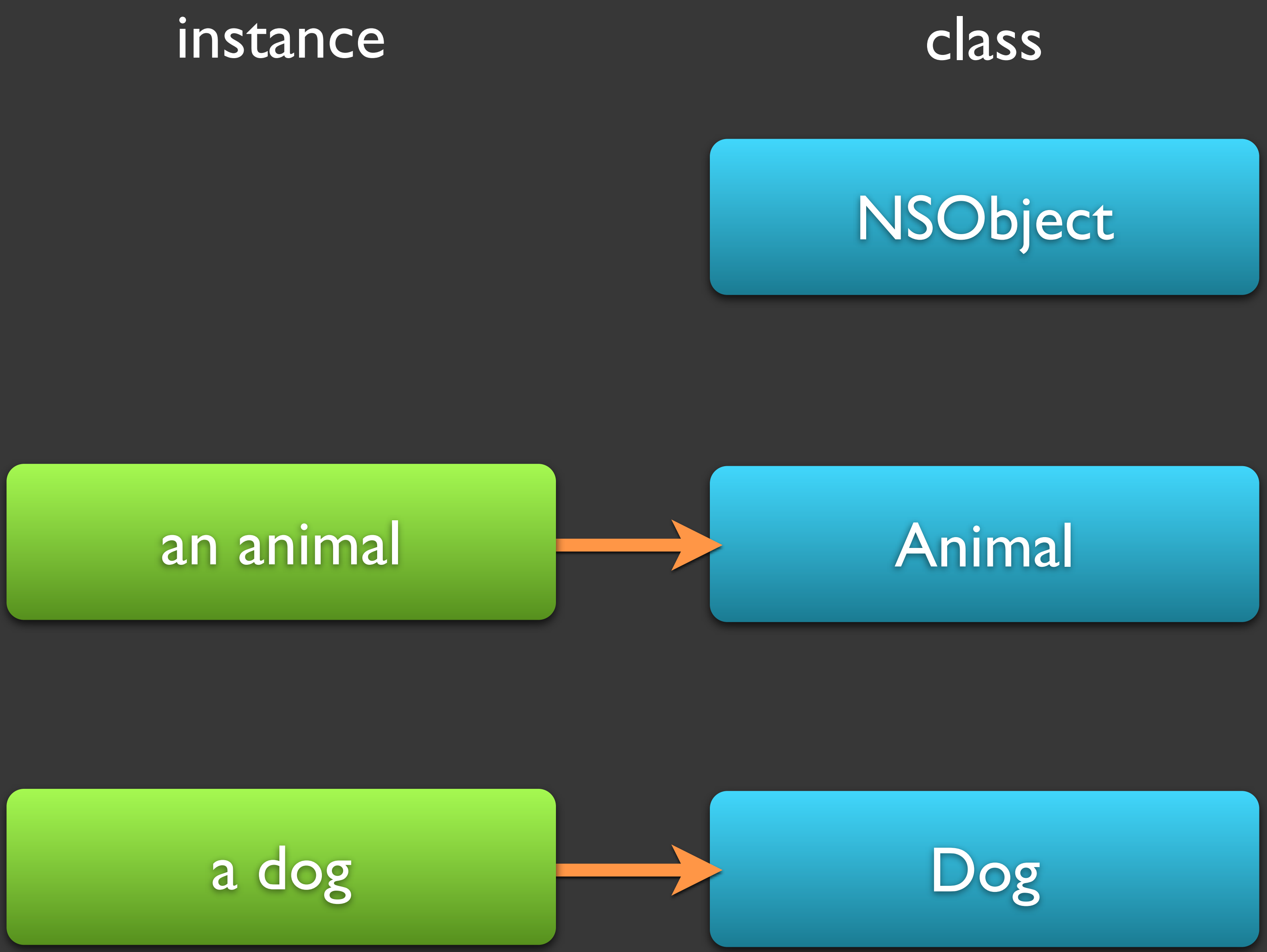
an animal

a dog

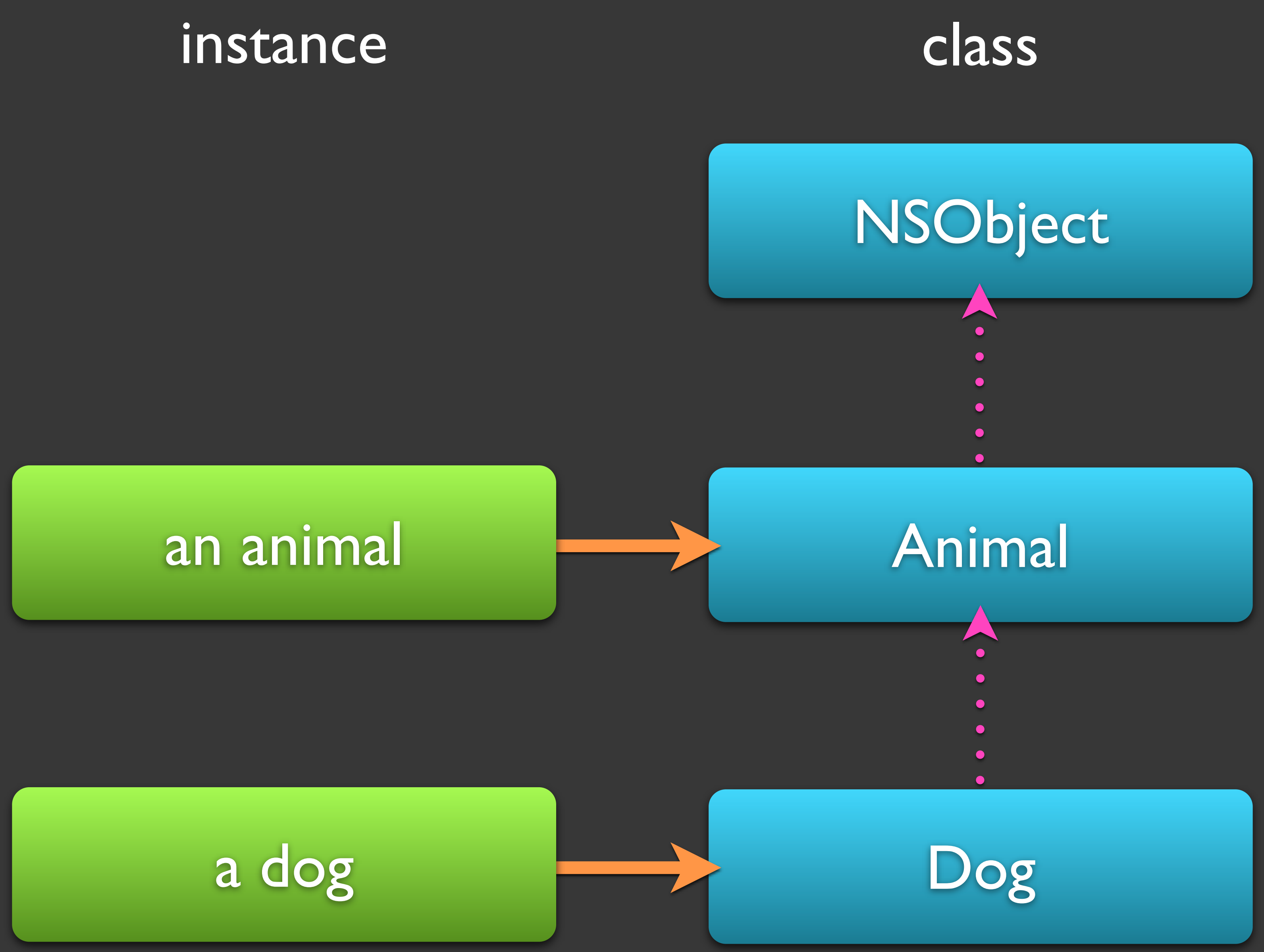
What are Objects?

instance	class
	NSObject
an animal	Animal
a dog	Dog

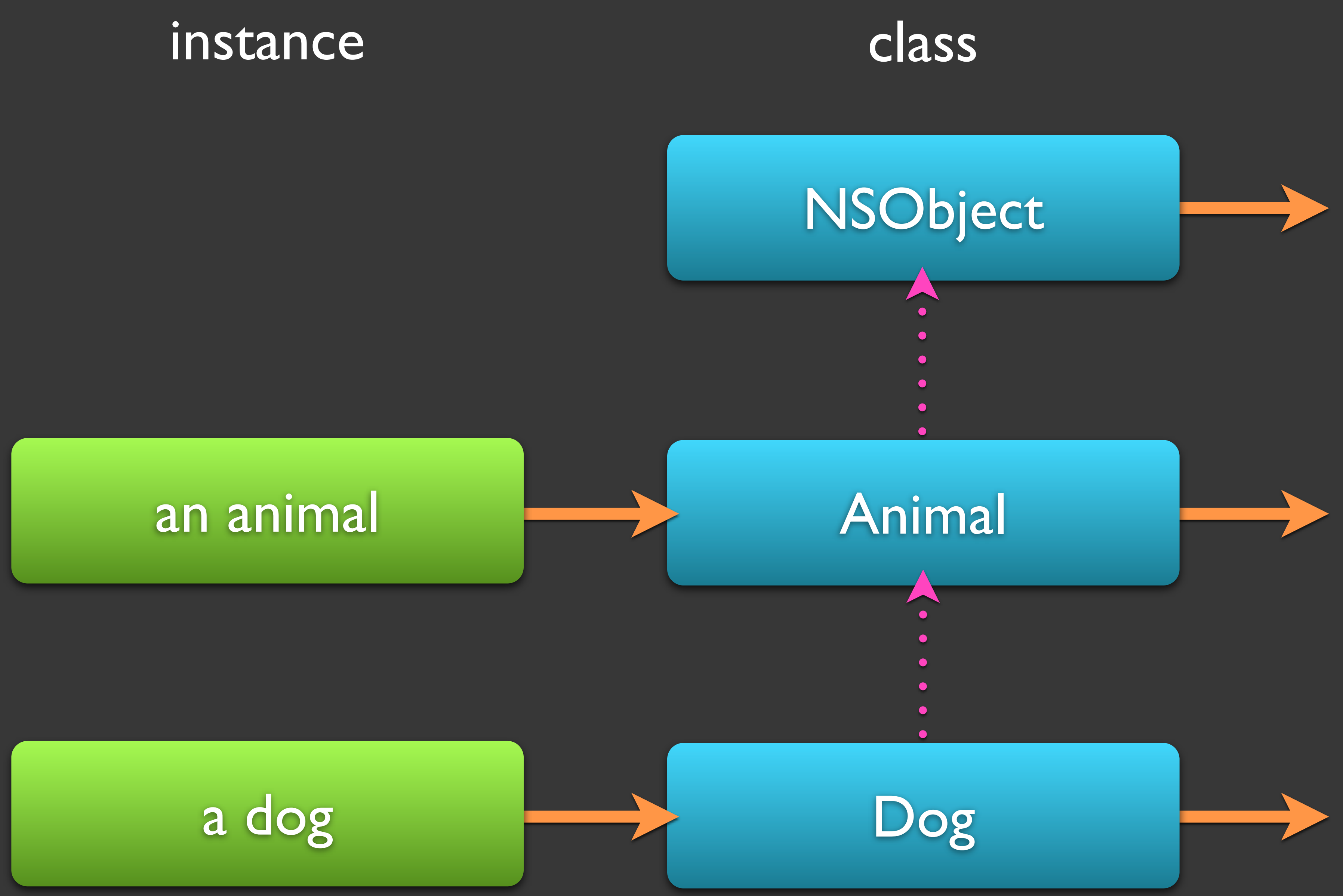
What are Objects?



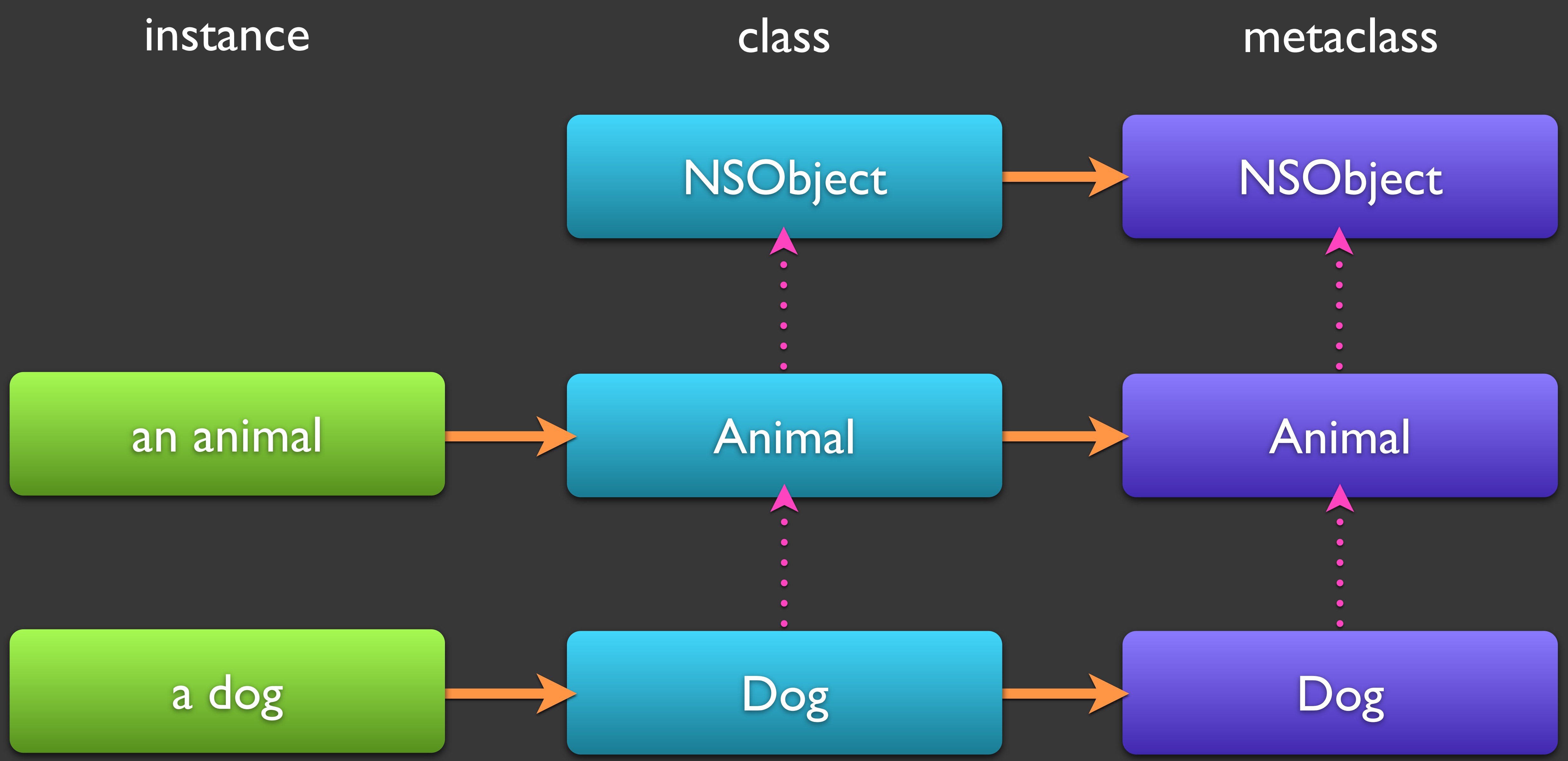
What are Objects?



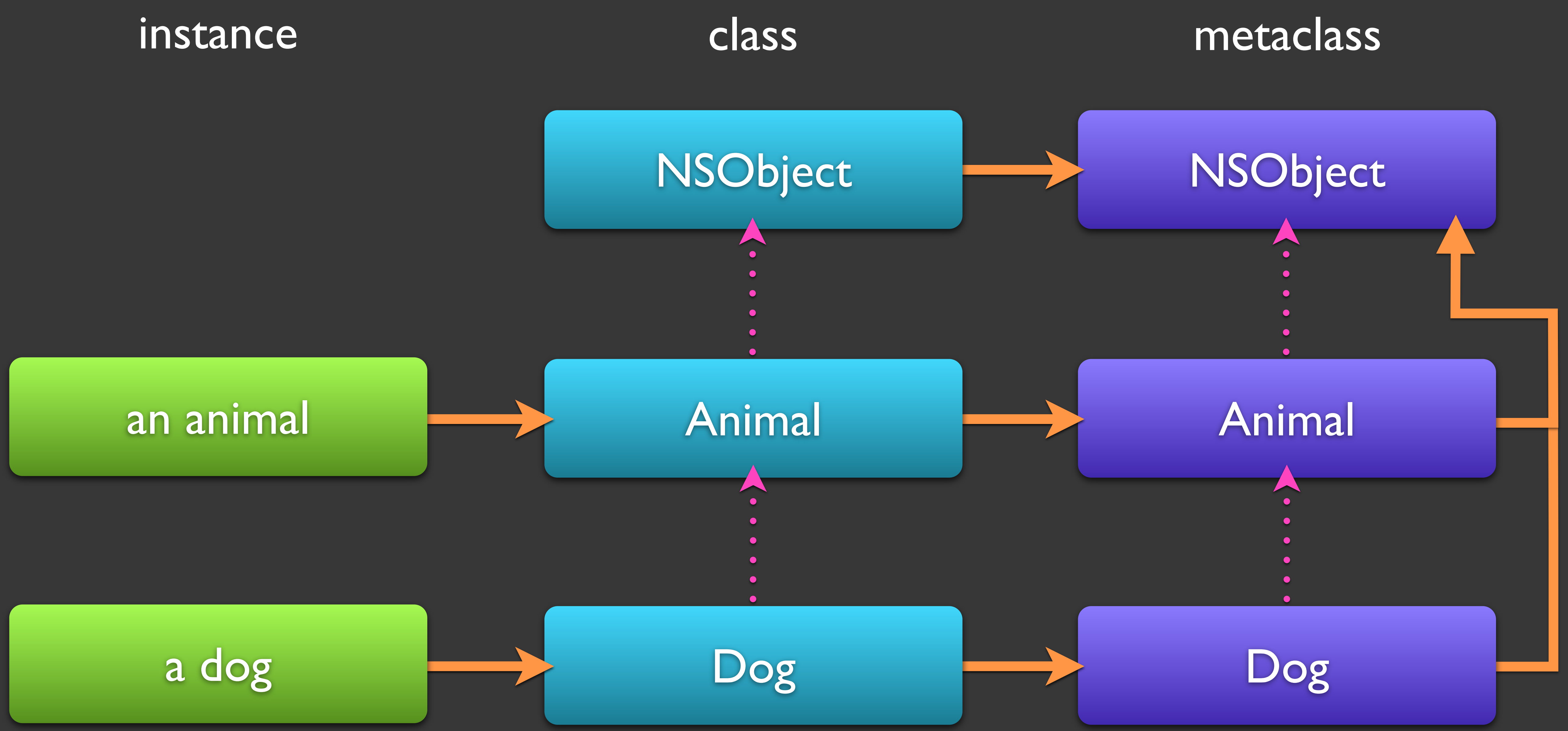
What are Objects?



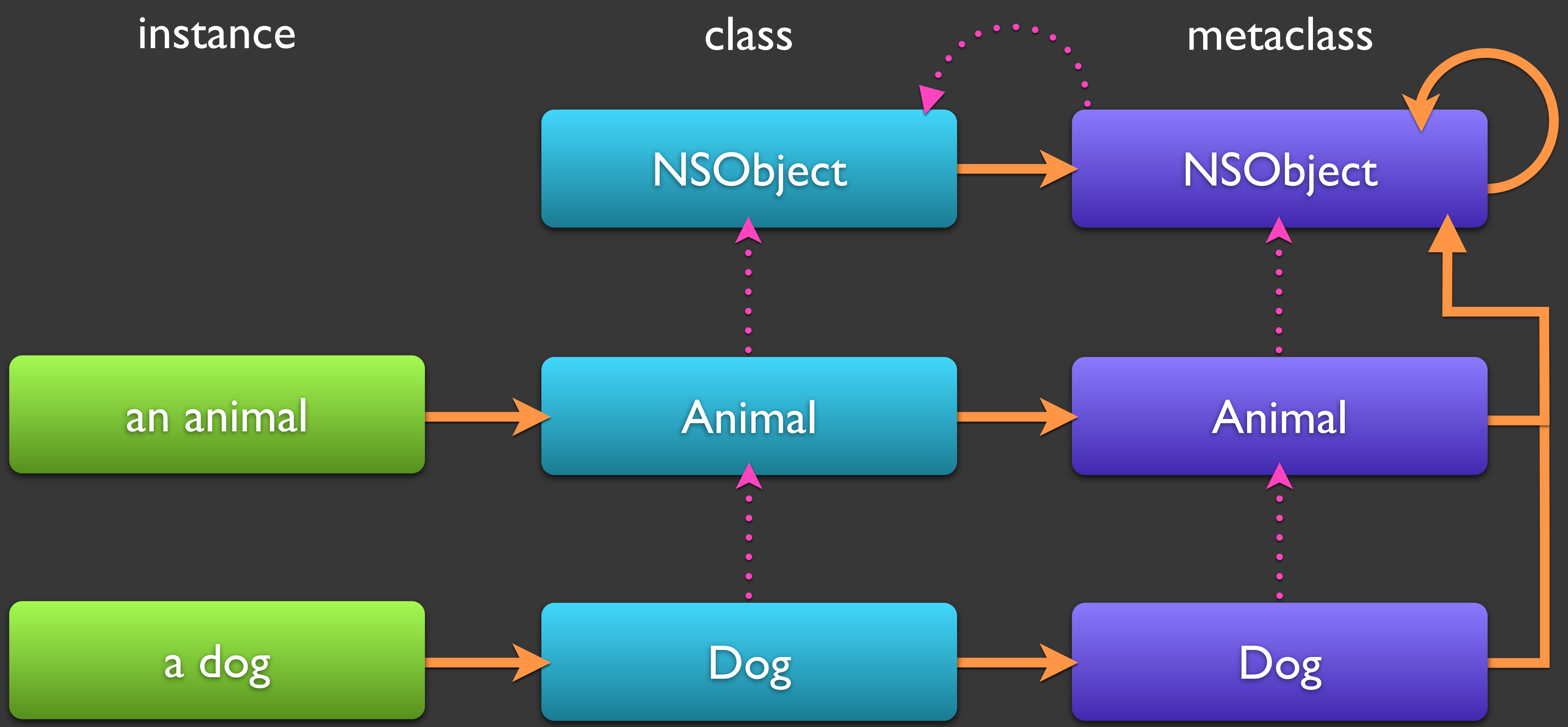
What are Objects?



What are Objects?



What are Objects?



Dynamic Subclassing

- Create dynamic subclass to handle instance mixins

```
Class objc_allocateClassPair(Class superclass,
                             const char *name,
                             size_t extraBytes)
void objc_registerClassPair(Class class)
```

- Changing an object’s class

```
object_setClass(id obj, Class class)
```

Implementation II

- Instance Mixins

```
[FRTimeAgo extendInstance:a];
```

Demo

Implementation III

- Calling original implementation
  - Like calling super, but from the module

What we need

- Nothing new :)

Thought exercise

Other Techniques

Standard Runtime Uses

Replacing Methods

a.k.a. Method Swizzling

Goals

- Replace method
- Be able to call original method

Replacing Methods

- Store the original `IMP` to call later

  ```
  class_getInstanceMethod
  method_getImplementation
  ```

- Replacing the method is simple

  ```
  class_replaceMethod
  ```

Replacing Methods

```
void (*OrigDrawRect)(id, SEL, NSRect);
void MyDrawRect(id self, SEL cmd, NSRect rect) {
  if ([[self title] isEqualToString:@"OK"]) {
    // draw custom okay buttons
  }
  else { OrigDrawRect(self, _cmd, rect); }
}
```

Replacing Methods

```
@implementation NSButton (ButtonDrawing)
+ (void)load {
  [self swizzle:@selector(drawRect:)
             with:(IMP)MyDrawRect
            store:(IMPPointer)&OrigDrawRect];
}
@end
```

Replacing Methods

```
typedef IMP *IMPPointer;

BOOL class_swizzleMethodAndStore(Class class, SEL original, IMP replacement, IMPPointer store) {
    IMP imp = NULL;
    Method method = class_getInstanceMethod(class, original);
    if (method) {
        const char *type = method_getTypeEncoding(method);
        imp = class_replaceMethod(class, original, replacement, type);
        if (!imp) {
            imp = method_getImplementation(method);
        }
    }
    if (imp && store) { *store = imp; }
    return (imp != NULL);
}

@implementation NSObject (FRRuntimeAdditions)
+ (BOOL)swizzle:(SEL)original with:(IMP)replacement store:(IMPPointer)store {
    return class_swizzleMethodAndStore(self, original, replacement, store);
}
@end
```

Replacing Methods

- StackOverflow: What are the Dangers of Method Swizzling in Objective C?
- Mike Ash: Method Replacement for Fun and Profit
- CocoaDev

Bypassing Message Sending

```
SEL selector = @selector(setFilled:);
void (*setter)(id, SEL, BOOL) =
   (void *)[target methodForSelector:selector];
for (int i = 0; i < 1000; i++)
  setter(targetList[i], selector, YES);
```

You'll probably never need to do this!

Be Creative

The sky’s the limit!

Thank you!

Questions?

Whitney Young
FadingRed

@wbyoung
wbyoung.github.com