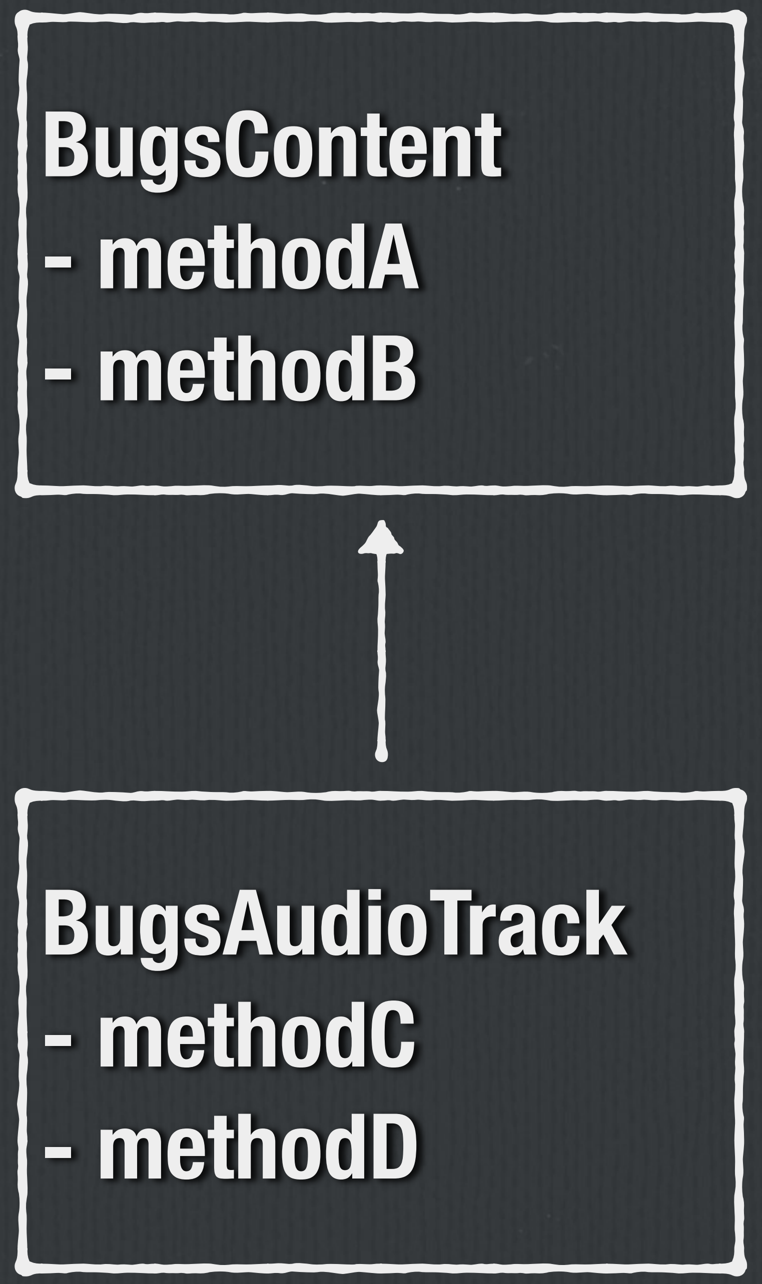
Method swizzling in Bugs iOS

Contents

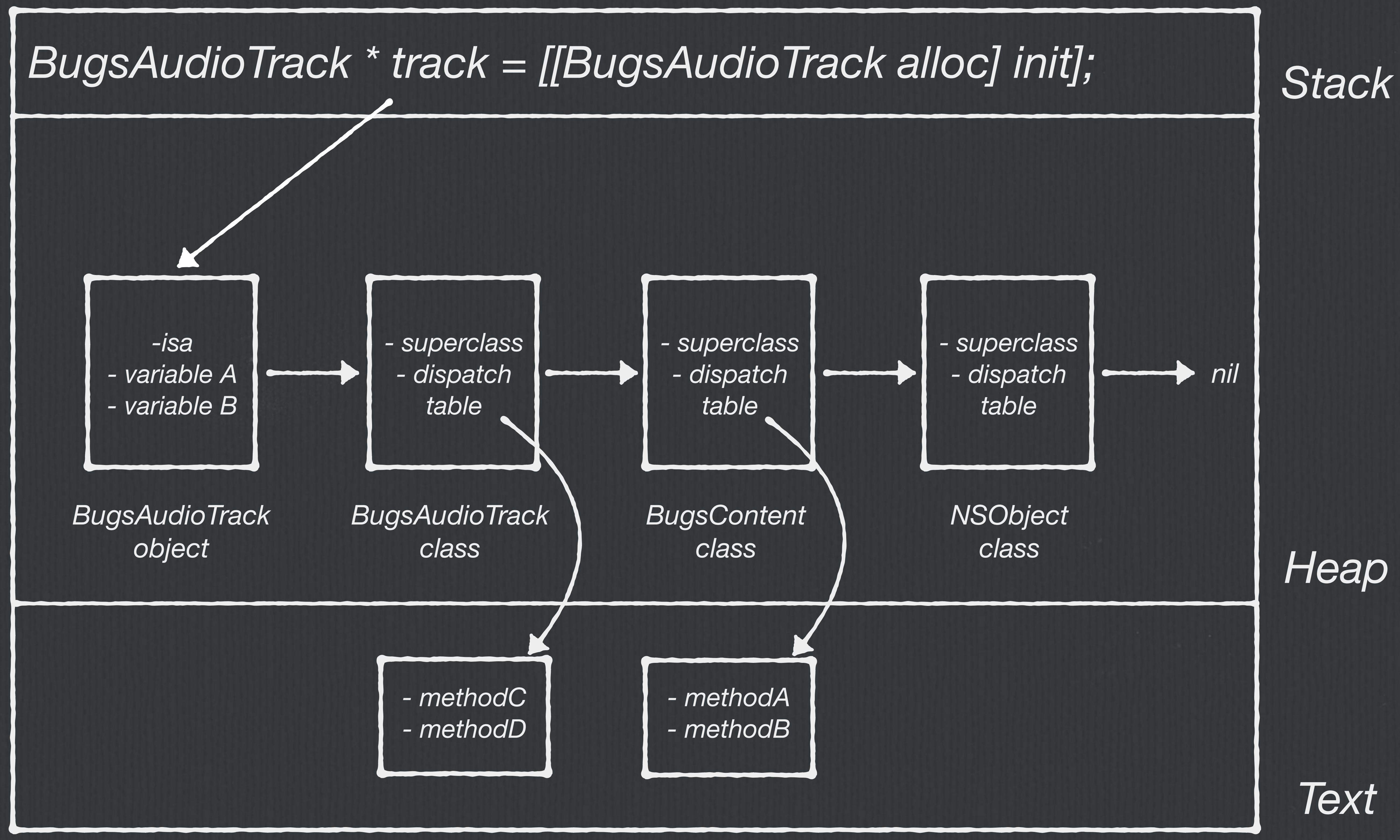
Understanding Objective-C Runtime

The Objective-C language defers as many decisions as it can from compile time and link time to runtime. Whenever possible, it does things dynamically.

Understanding Objective-C Runtime



Inheritance UML



Memory layout

* ‘superclass’ is unavailable in Objective C 2.0 Runtime

Understanding Objective-C Runtime

Compiler

The compiler just converts a message into a call on a messaging function *objc_msgSend*

```
BugsAudio Track * track = [BugsAudioTrack new];

[track title];

objc_msgSend(track, @selector(title));

[track belongToArtist: param1];

objc_msgSend(track, @selector(belongToArtist), param1);
```

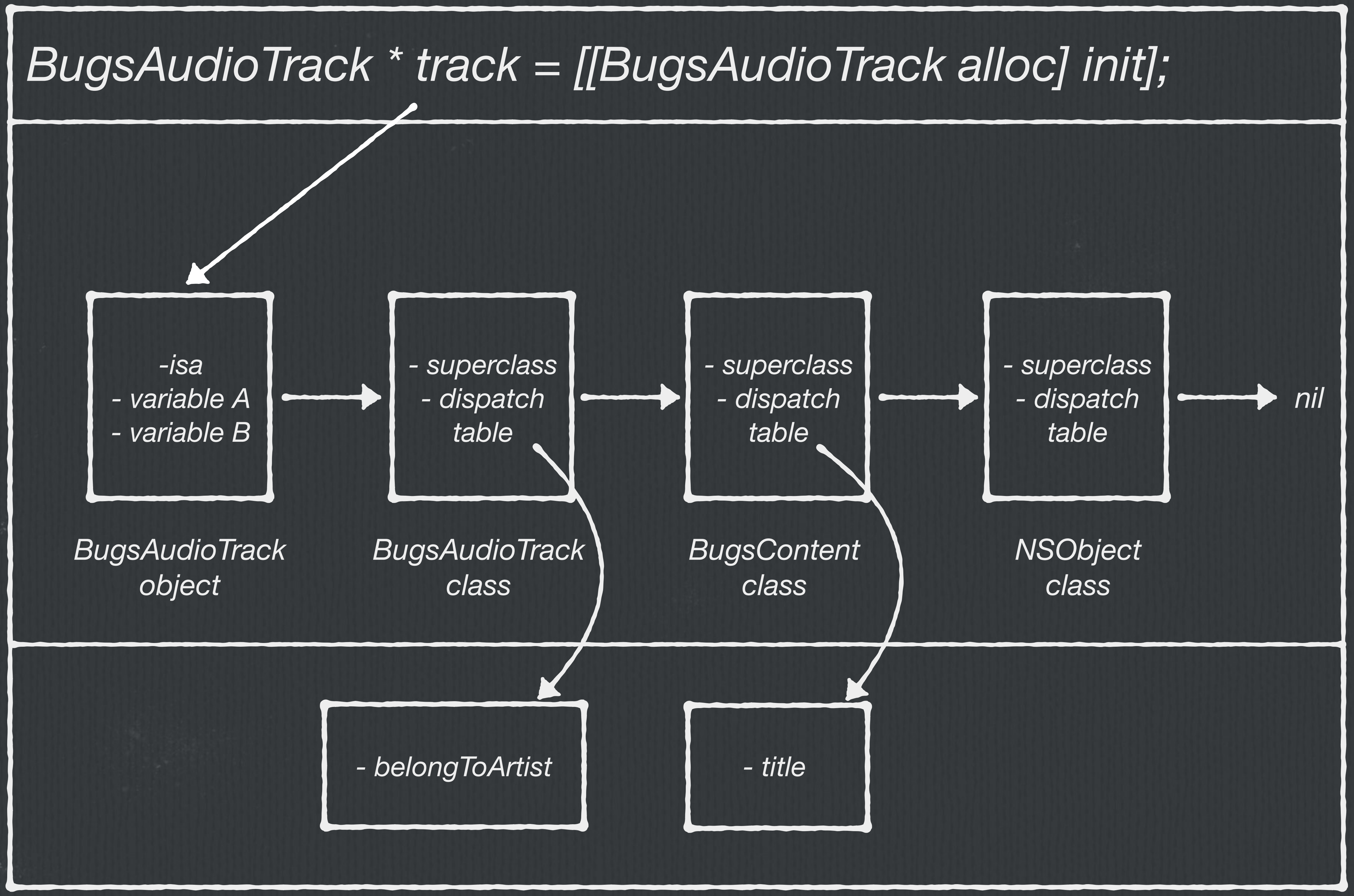
Understanding Objective-C Runtime

Runtime

1. Looks up the method selector in the dispatch table
2. If it can’t find the selector there, objc_msgSend follows the pointer to the superclass and tries to find the selector in its dispatch table.
3. Successive failures cause objc_msgSend to climb the class hierarchy until it reaches the NSObject class

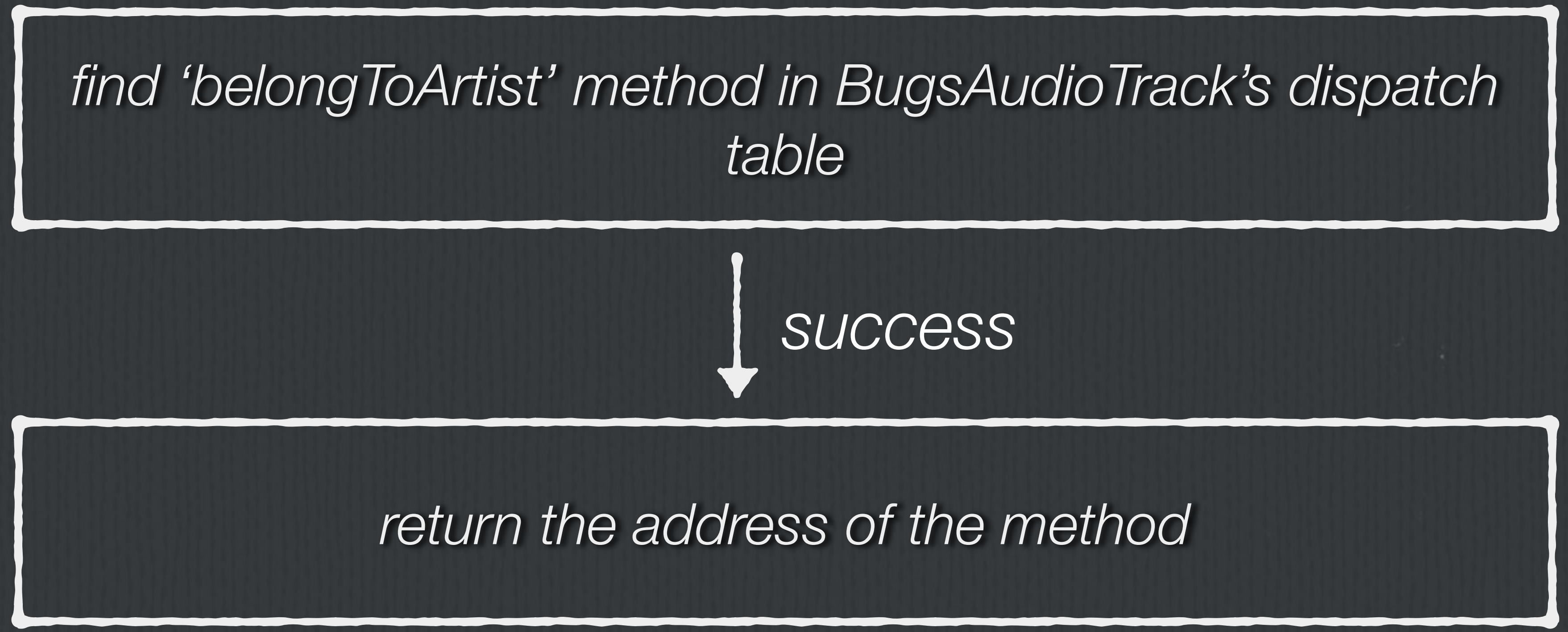
Understanding Objective-C Runtime

Runtime



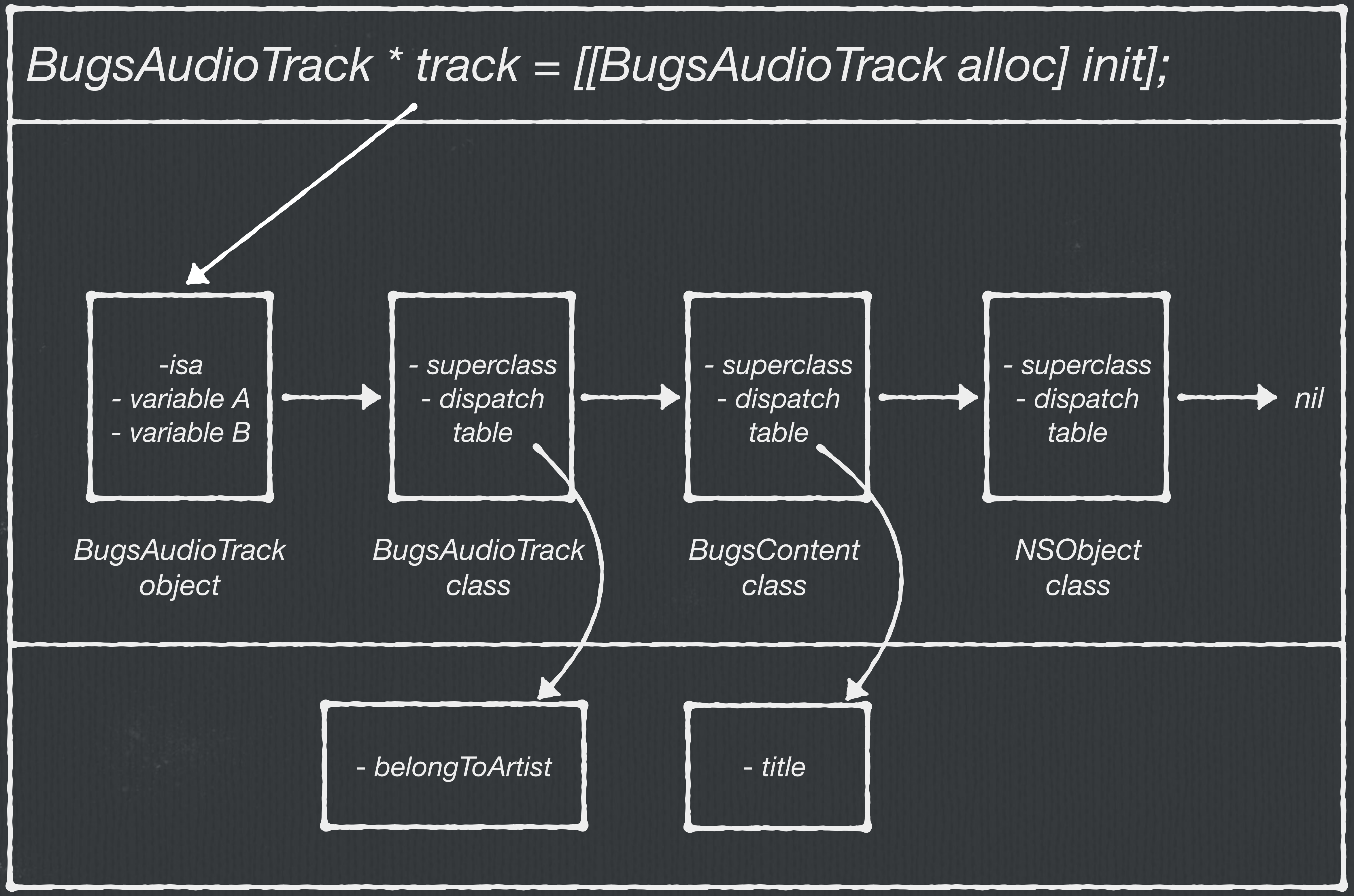
Memory layout

objc_msgSend(track, @selector(belongToArtist), param1)


find 'belongToArtist' method in BugsAudioTrack's dispatch table
success
return the address of the method

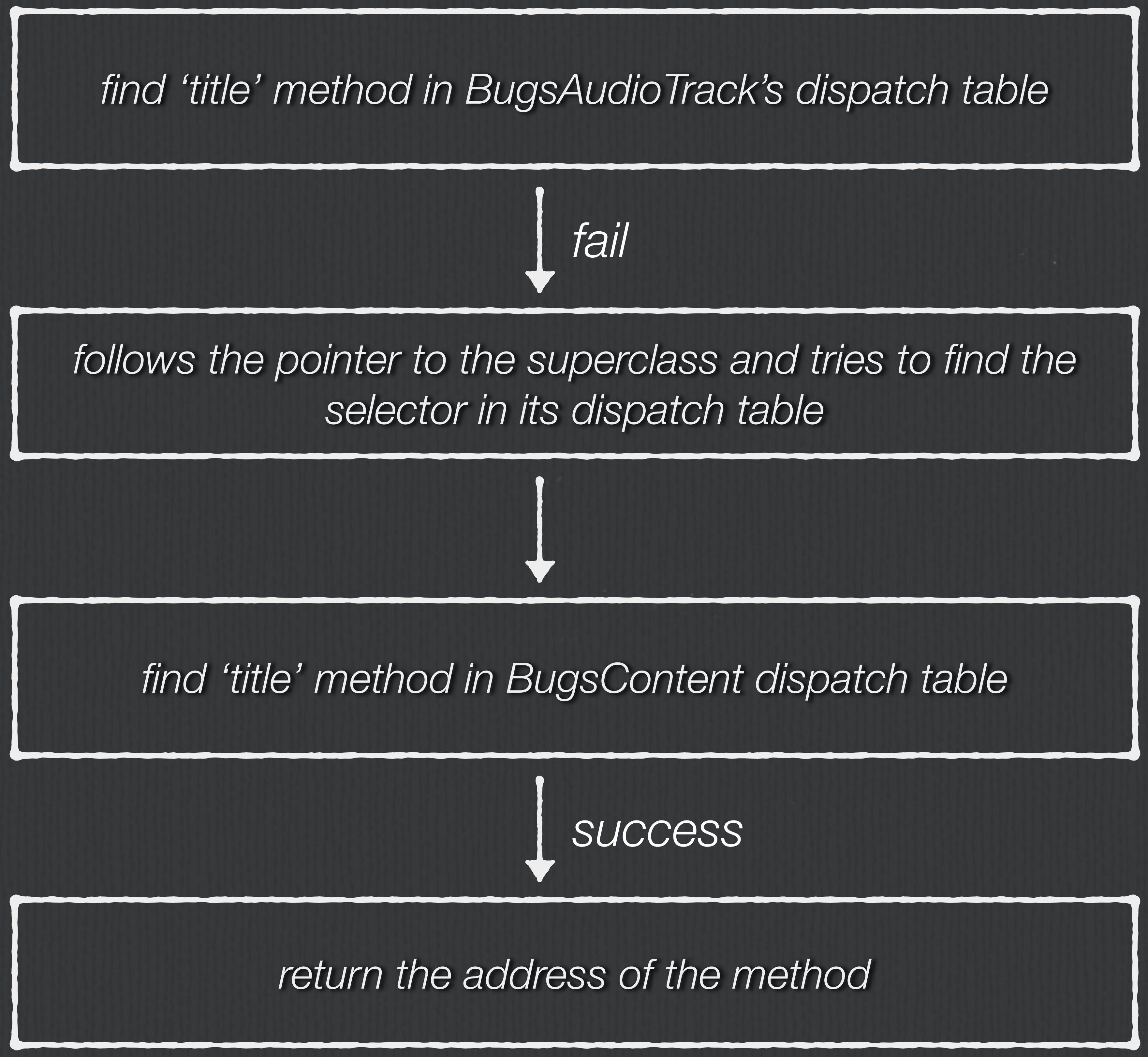
Understanding Objective-C Runtime

Runtime



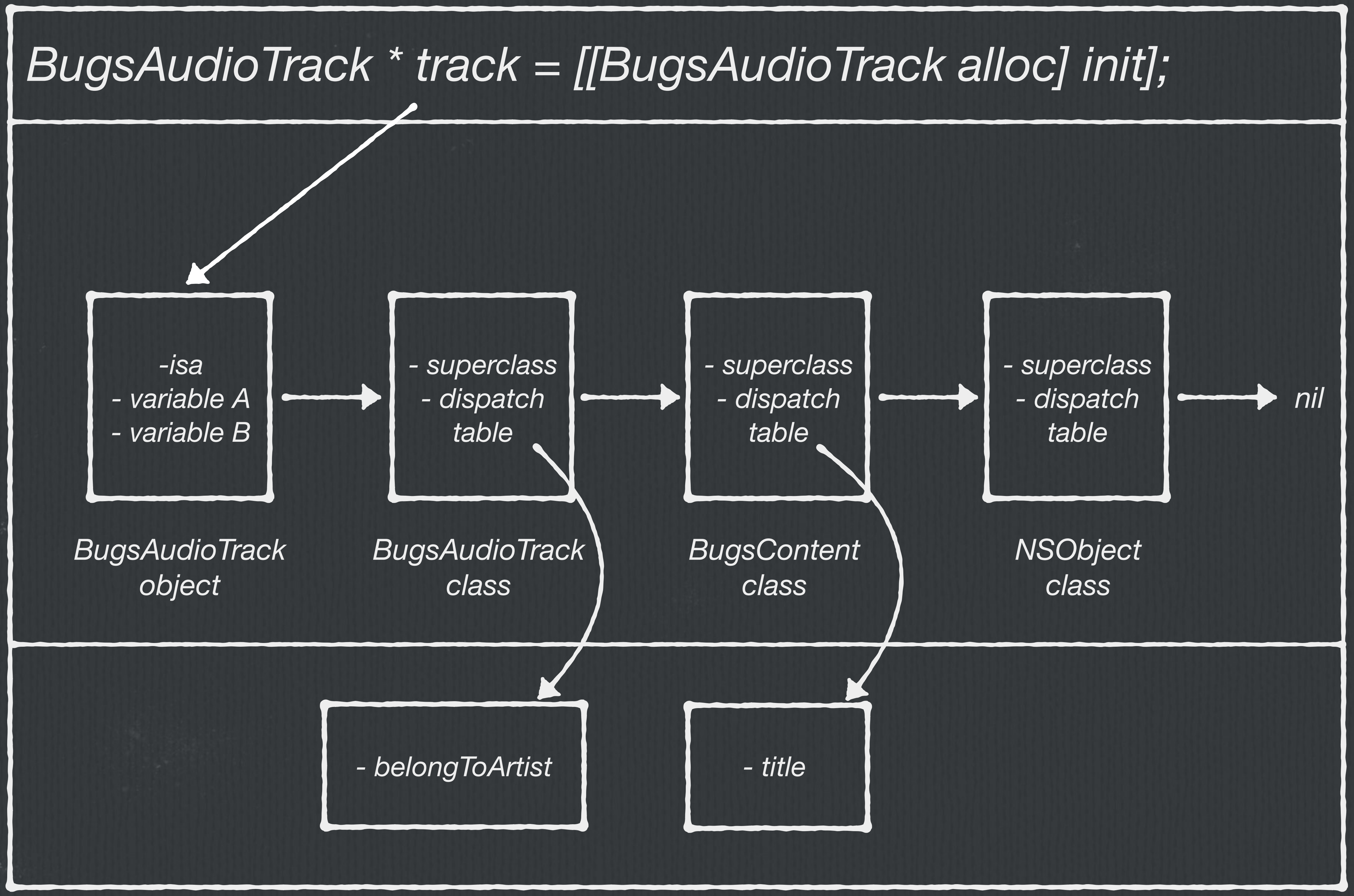
Memory layout

objc_msgSend(track, @selector(title))


find 'title' method in BugsAudioTrack's dispatch table
fail
follows the pointer to the superclass and tries to find the selector in its dispatch table
find 'title' method in BugsContent dispatch table
success
return the address of the method

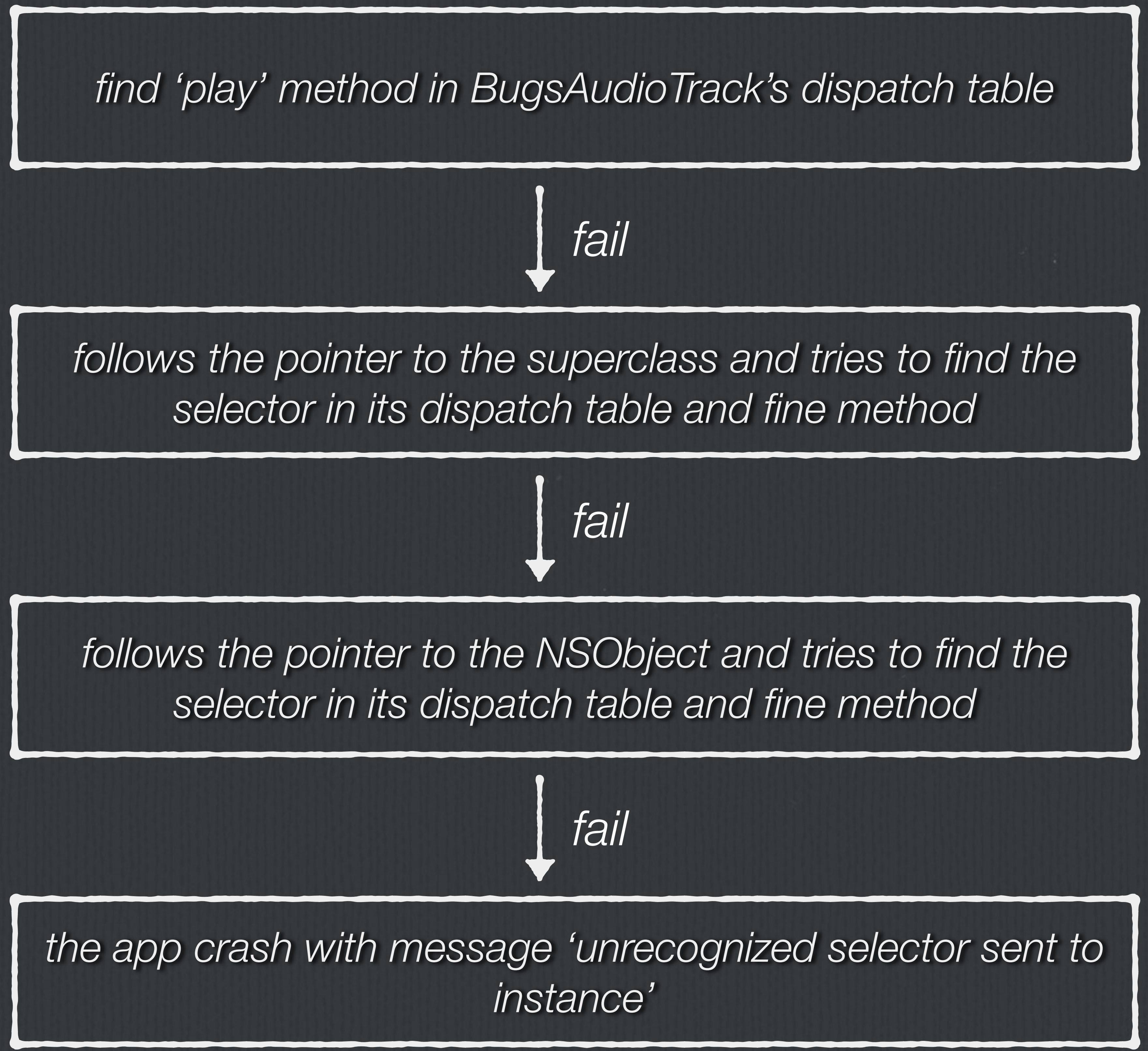
Understanding Objective-C Runtime

Runtime Exception case



Memory layout

objc_msgSend(track, @selector(play))



find 'play' method in BugsAudioTrack's dispatch table
fail
follows the pointer to the superclass and tries to find the selector in its dispatch table and fine method
fail
follows the pointer to the NSObject and tries to find the selector in its dispatch table and fine method
fail
the app crash with message 'unrecognized selector sent to instance'

* Not consideration 'Message forwarding'

Anatomy of Swizzling

Runtime functions

Many of these functions allow you to use plain C to replicate what the compiler does when you write Objective-C code

Examples

class_addMethod
class_replaceMethod
class_getInstanceMethod
method_getImplementation

Anatomy of Swizzling

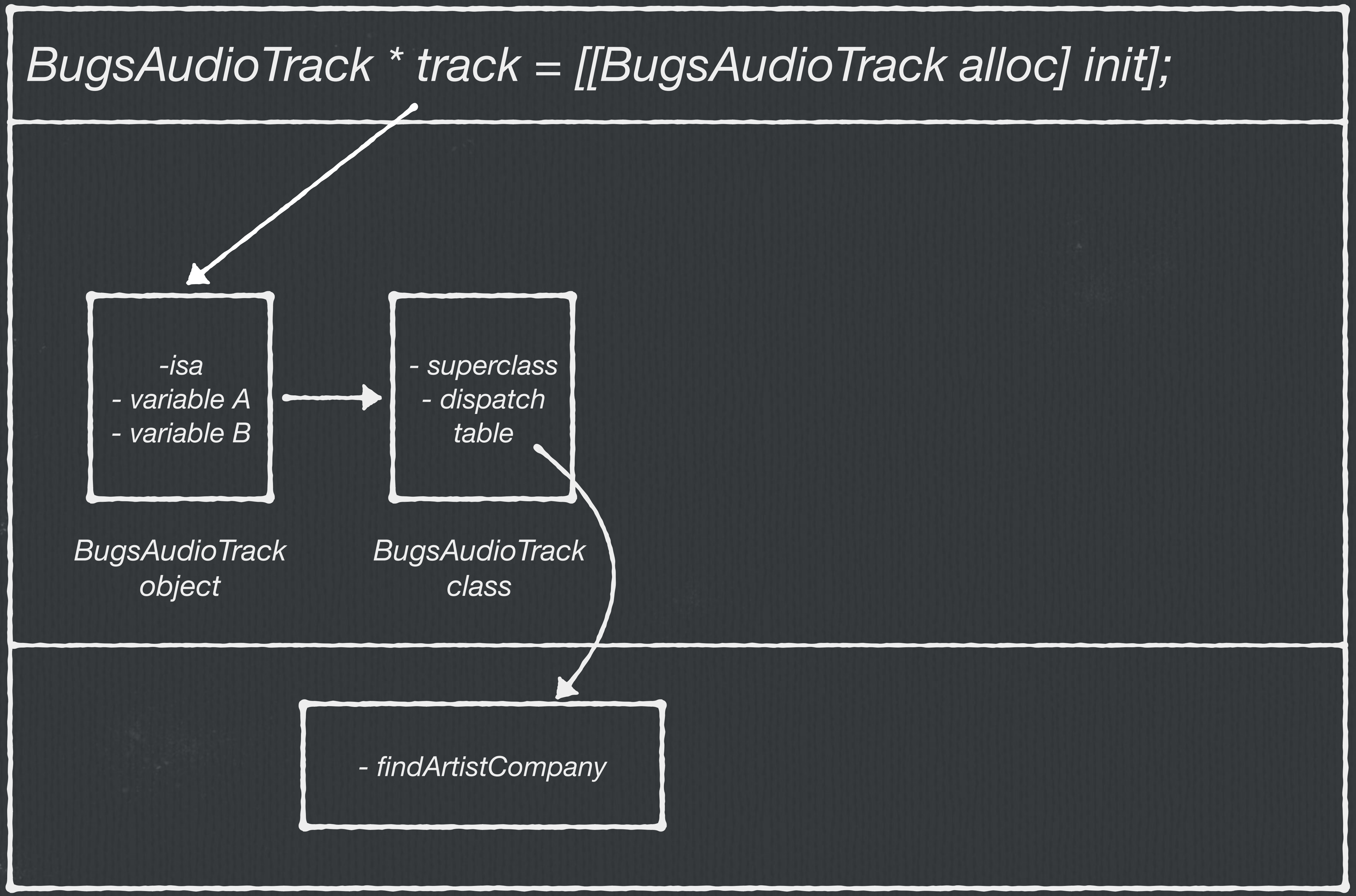
Runtime functions (class_addMethod)

class_addMethod(Class cls, SEL name, IMP imp, const char *types);

```
{
    // …
    class_addMethod(BugsAudioTrack.class, @selector(artistCompany), (IMP)findArtistCompany, nil);
    // …
}

void findArtistCompany(id self, SEL _cmd) {
   // New business logic.
}

[(id)track artistCompany]
```

Anatomy of Swizzling

Runtime functions (class_addMethod)



Memory layout

Anatomy of Swizzling

Method swizzling is the process of changing the implementation of an existing selector at runtime.

Anatomy of Swizzling

1. Swizzling implementation with class_replaceMethod

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


{
    // …
    class_replaceMethod(BugsAudioTrack.class, @selector(artistName), (IMP)newGetArtistName, nil);
    // …
}

void newGetArtistName(id self, SEL _cmd) {
   // New business logic.
}

[track artistName]
```

Anatomy of Swizzling

2. ***Swizzling implementation with method_setImplemention***

*class_replaceMethod(Class cls, SEL name, IMP imp, const char *types);*

```
{
    // …
    Method oldName = class_getInstaceMethod(BugsAudioTrack.class, @selector(artistName));
    IMP old_implementation = method_setImplementation(oldName, (IMP)newSwizzlingMethod);
    // set ‘oldImplementation’ as internal variable
}

void newSwizzlingMethod(id self, SEL _cmd) {

   // Do ‘New business logic’
  old_implementation(self, _cmd);
}
```

** method_setImplementation return the previous implementation of method.*

Anatomy of Swizzling

Bugs Practice 1

We have a unique presenting rule. For example When the system invoke to present Player UI, we have to dismiss all other UI like UIViewController.

```
{
    // …
    Method oldName = class_getInstaceMethod(UIViewController.class, @selector(presentViewController…));
    IMP originOfMethod = method_setImplementation(oldName, (IMP)newPresentMethod);
    // set ‘originOfMethod’ as internal variable
}

void newPresentMethod(id self, SEL _cmd, UIViewController * toPresent, Bool animated) {

    if ([toPresent conformToProtocol:@protocol(PresentingContext)] == YES)
    {
        // do something with method which conform to specific protocol. For example dismiss UIViewController.
    }

    originOfMethod(self, _cmd, toPresent, animated …);
}
```

Anatomy of Swizzling

Bugs Practice 2

Let’s affect the Darkmode. To do easily, we have to use ColorAsset with UIColor(name:) method.
But UIColor(name:) method will be crashed in iOS10 below

```
{
    // …
    Method oldName = class_getInstaceMethod(UIViewController.class, @selector(loadNibMethod…));
    IMP originOfMethod = method_setImplementation(oldName, (IMP)newloadNibMethod);
    // set ‘originOfMethod’ as internal variable
}

void newloadNibMethod(id self, SEL _cmd, NSString * identifier, …) {

    if available(iOS11) {
        originOfMethod(self, _cmd, identifier, …); // Use Color(name:) method without crash.
    }
    else  {
        originOfMethod(self, _cmd, idenfier_for_iOS10,…); // Don’t use Color(name:) method.
    }
}
```

Advanced Topics

1. *Getting a Method Address*

The only way to circumvent dynamic binding is to get the address of a method and call it directly as if it were a function.

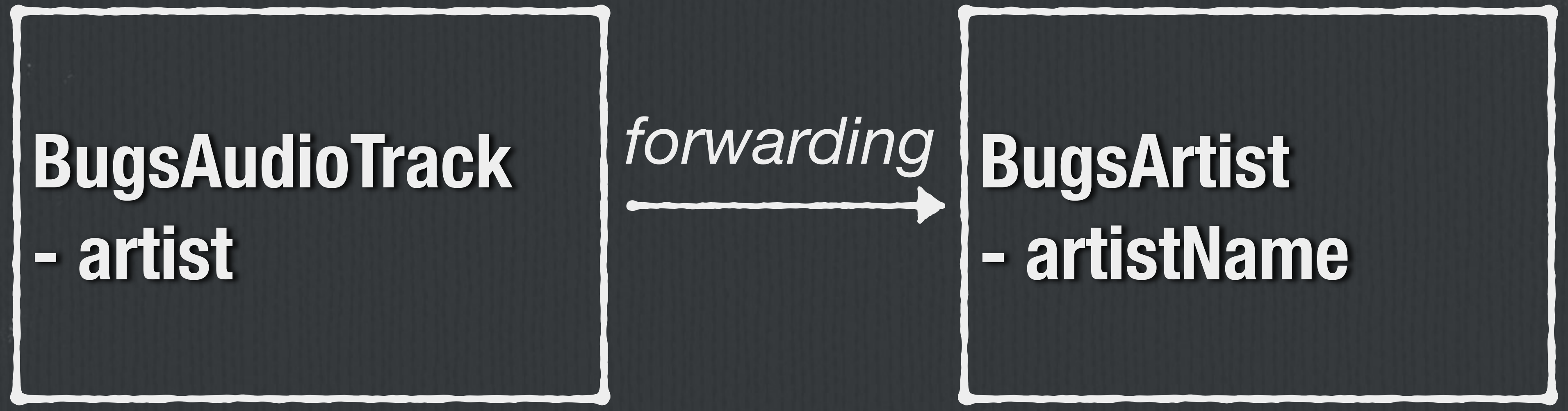
```
void (*setter)(id, SEL, BOOL);
int i;

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

Using methodForSelector: to circumvent dynamic binding saves most of the time required by messaging. However, the savings will be significant only where a particular message is repeated many times, as in the for loop shown above.

Advanced Topics

2. ***Message Forwarding***

Sending a message to an object that does not handle that message is an error. However, before announcing the error, the runtime system gives the receiving object a second chance to handle the message.



- *Heavily used in BugsAppkit/BugsUIKit Framework.*
- *Let's see detail in next presentation.*