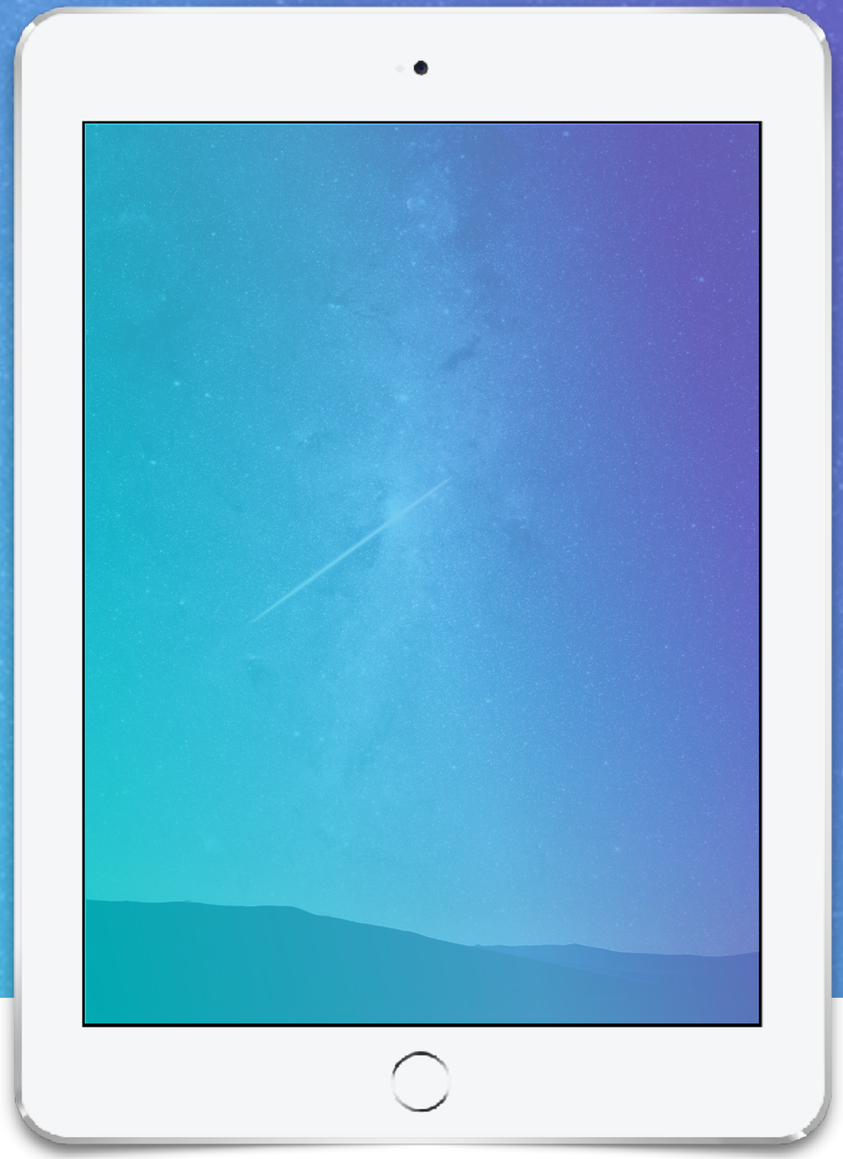
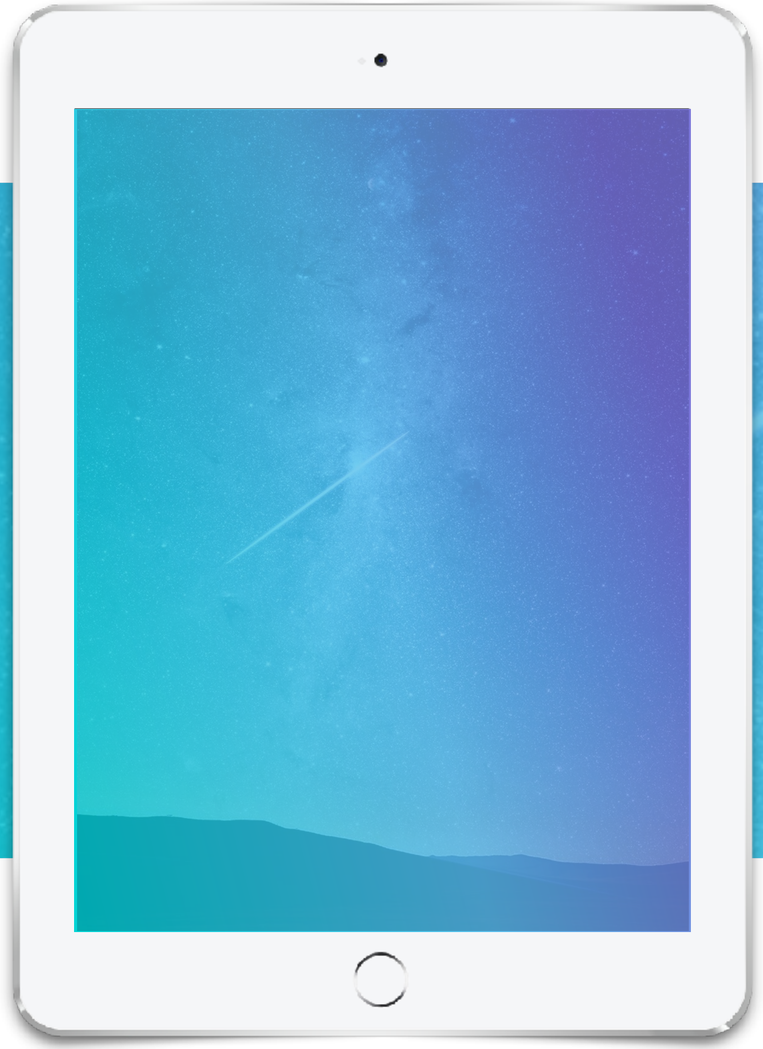
iOS 黑魔法课程

第三课
OC语言魔法（上）



StuQueue

一个新的学习方式



本课内容

- OC 黑魔法与 Runtime
- OC Runtime 基本概念

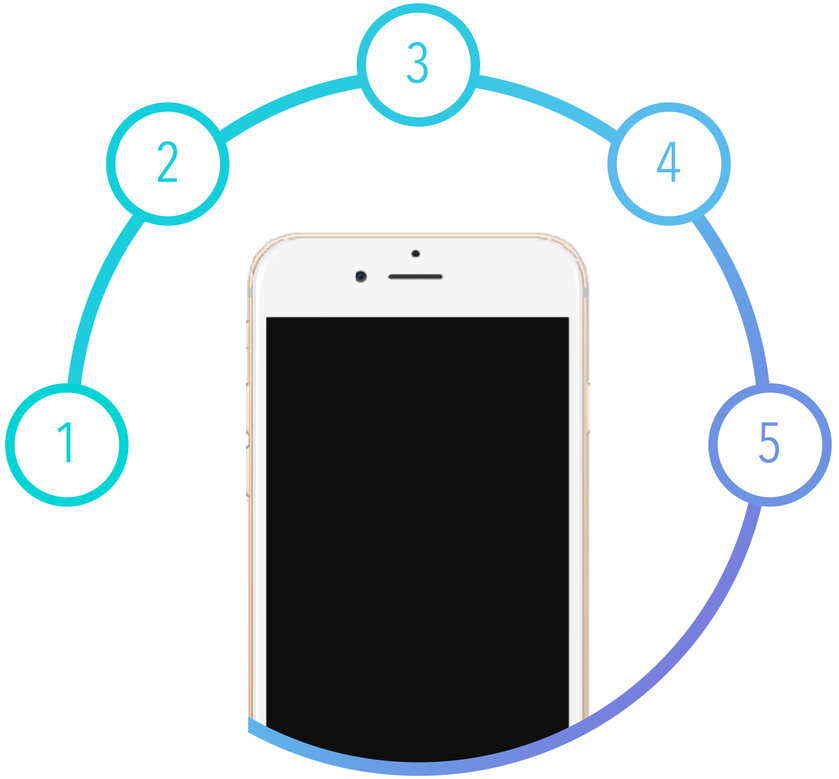
OC 黑魔法与 Runtime

StuQ

OC 黑魔法与 RUNTIME

说说 OC 语言

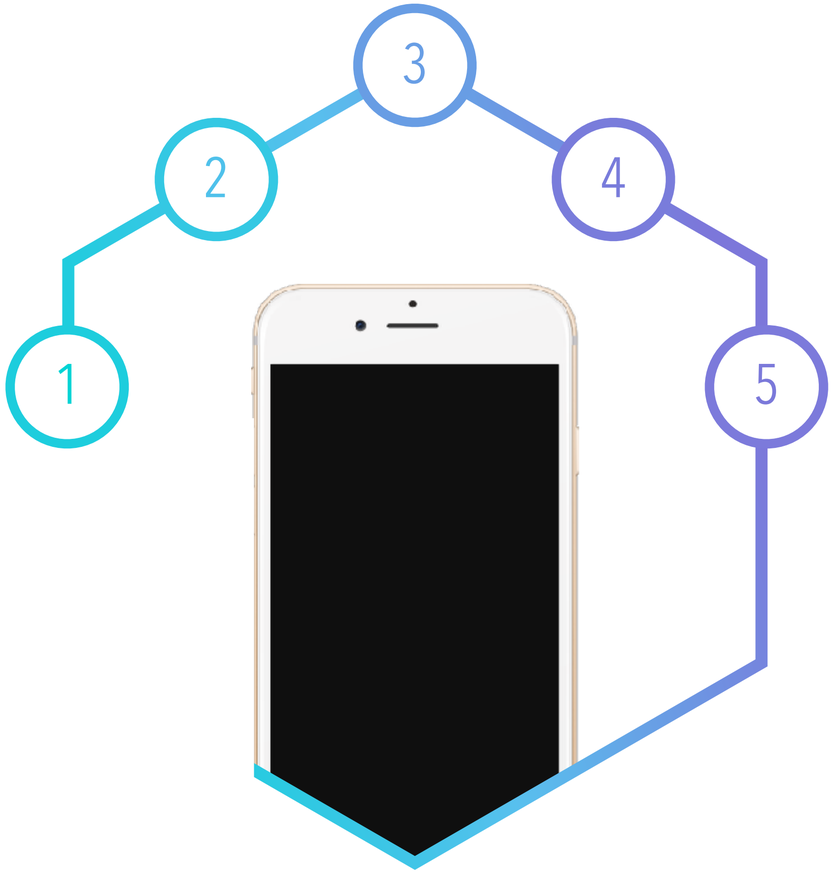
- C 的超集
- OOP 语言
- 动态语言



OC 黑魔法与 RUNTIME

何为 Runtime?

- 直译为“运行时”，表示运行的时候
- 为 OOP 提供的运行环境
- 与 Build Time 对应
- 内存布局 + 执行逻辑



OC 黑魔法与 RUNTIME

Runtime 与 Runtime API

- Runtime API 是 Runtime 的接口

```
@interface TestClass : NSObject

- (NSInteger)testMethod;
@end
```

```
@implementation TestClass

- (NSInteger)testMethod
{
    return 15;
}
@end
```

```
int testMethod2(void *_, void *__) {
    return 16;
}

void exchange_method_no_api(void *obj) {

    void *p1 = *(void **)obj;
    void *p2 = (void *)(*(long *)(p1 + 0x20) & (unsigned long)(-7));
    void *p3 = (void *)*(long *)(p2 + 0x10) + 0x8;
    typedef int (*M)(void *, void *);
    M *m = (M *)(p3 + 0x10);
    *m = &testMethod2;

}
```

OC 黑魔法与 RUNTIME

Runtime 与 Runtime API

- Runtime API 是 Runtime 的接口

```
int testMethod3(id self, SEL cmd) {
    return 17;
}


void exchange_method_with_api(id obj) {
    Class class = object_getClass(obj);
    unsigned int count = 0;
    Method *methods = class_copyMethodList(class, &count);
    if (count != 1) return;
    Method method = methods[0];
    method_setImplementation(method, (IMP)testMethod3);
}
```

```
int testMethod2(void *_, void *__) {
    return 16;
}

void exchange_method_no_api(void *obj) {

    void *p1 = *(void **)obj;
    void *p2 = (void *)(*(long *)(p1 + 0x20) & (unsigned long)(-7));
    void *p3 = (void *)*(long *)(p2 + 0x10) + 0x8;
    typedef int (*M)(void *, void *);
    M *m = (M *)(p3 + 0x10);
    *m = &testMethod2;

}
```

OC 黑魔法与 RUNTIME

OC 黑魔法 ≠ OC Runtime

基于 OC 的程序设计

C 语言黑魔法

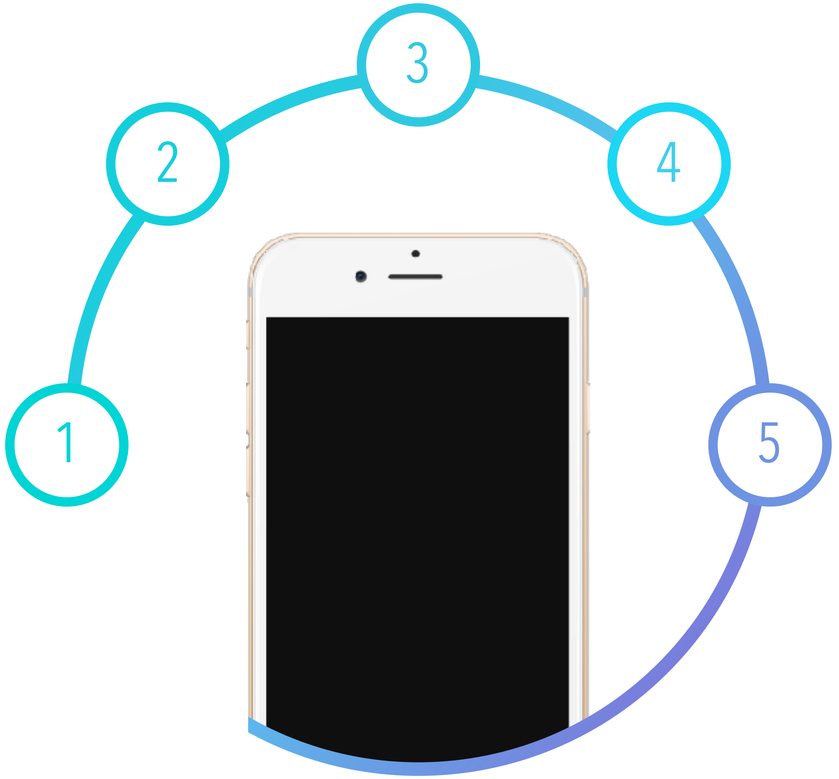
OC Runtime

OC Runtime 基本概念

OC RUNTIME的基本概念

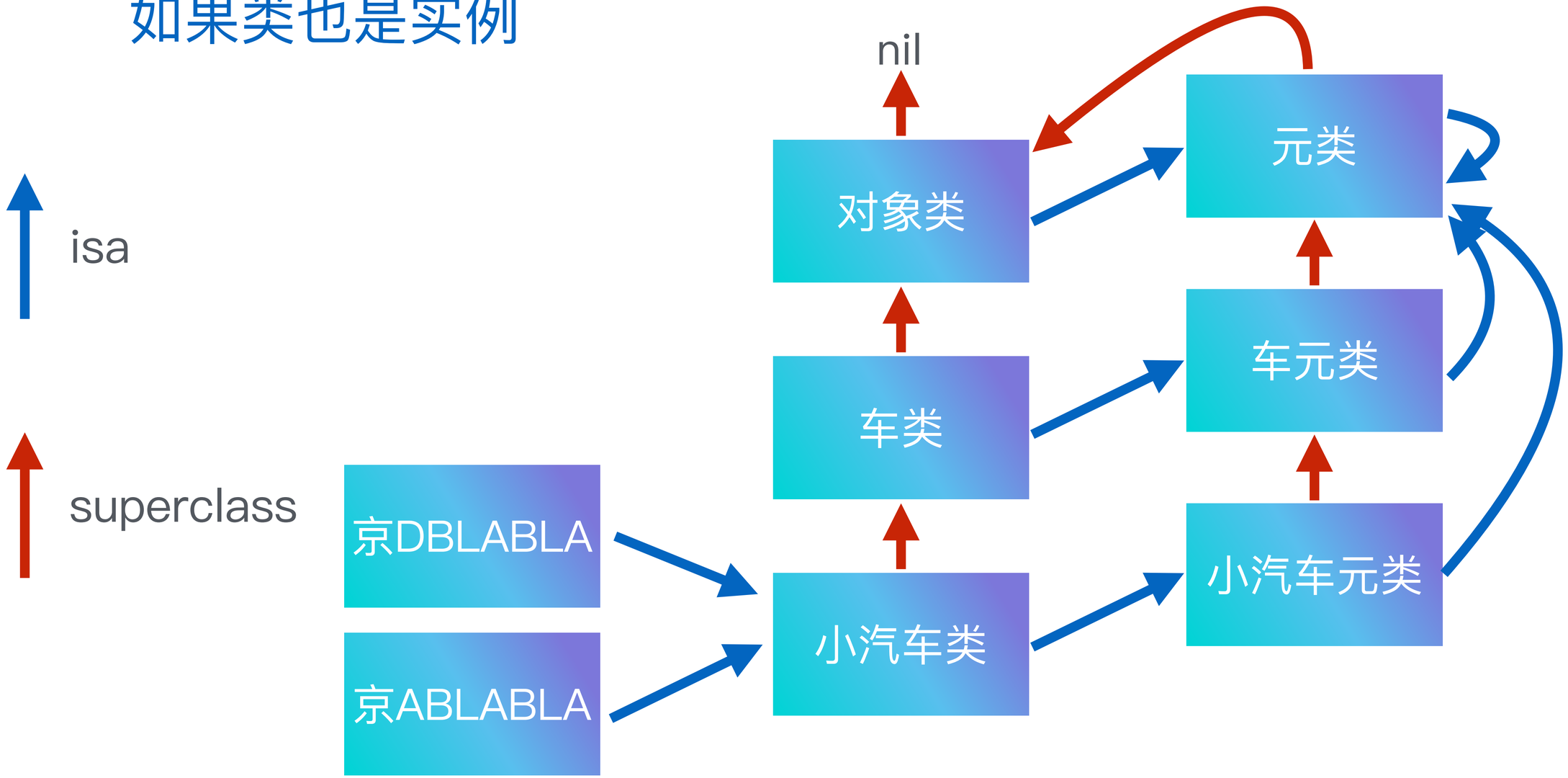
说说 OC 的 OOP 选型

- 类也是实例
- 类型运行时确定
- 行为储存于类
- 状态存储于实例



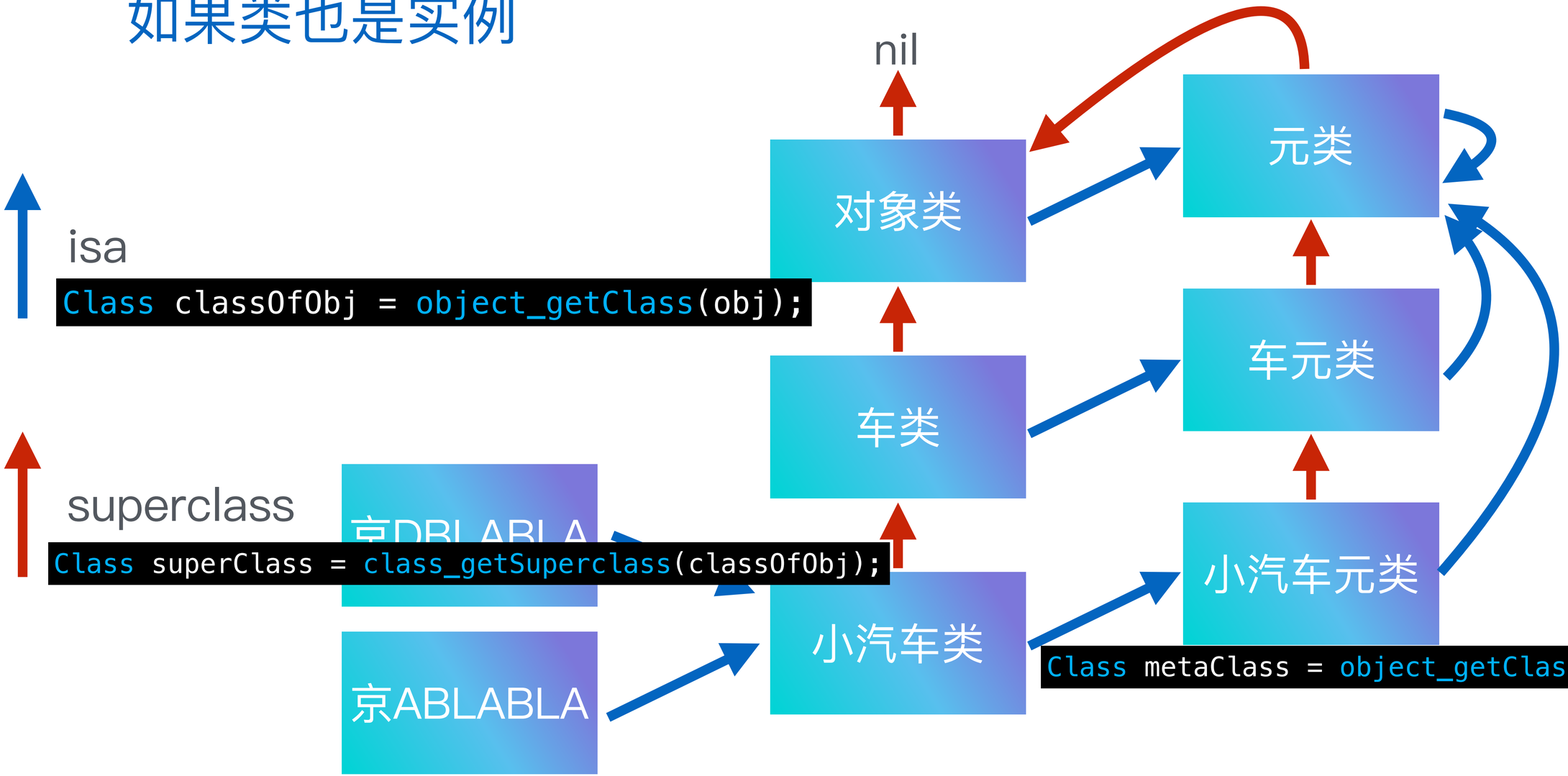
OC RUNTIME的基本概念

如果类也是实例



OC RUNTIME的基本概念

如果类也是实例



OC RUNTIME的基本概念

类型的确定

```
class SomeClassA {
    int var1;
public:
    void func1();
};

class SomeClassB {
    int var2;
public:
    void func2();
};
```

```
void SomeClassA::func1() {
    std::cout << __FUNCTION__ << std::endl;
}

void SomeClassB::func2() {
    std::cout << __FUNCTION__ << std::endl;
}
```

编译时确定类型

```
void testCpp1() {
    SomeClassA a = SomeClassA();
    a.func1();
}
```

```
void testCpp2() {
    SomeClassA a = SomeClassA();
    SomeClassB *b = reinterpret_cast<SomeClassB *>(&a);
    b->func2();
}
```

OC RUNTIME的基本概念

类型的确定

```
@interface OCSomeClassA : NSObject
- (void)func1;
@end

@interface OCSomeClassB : NSObject
- (void)func2;
@end
```

```
@implementation OCSomeClassA
- (void)func1
{
    NSString *string = [NSString stringWithUTF8String:__FUNCTION__];
    NSLog(@"%@", string);
}
@end

@implementation OCSomeClassB
- (void)func2
{
    NSString *string = [NSString stringWithUTF8String:__FUNCTION__];
    NSLog(@"%@", string);
}
@end
```

```
void testOC1()
{
    OCSomeClassA *a = [OCSomeClassA new];
    [a func1];
}
```

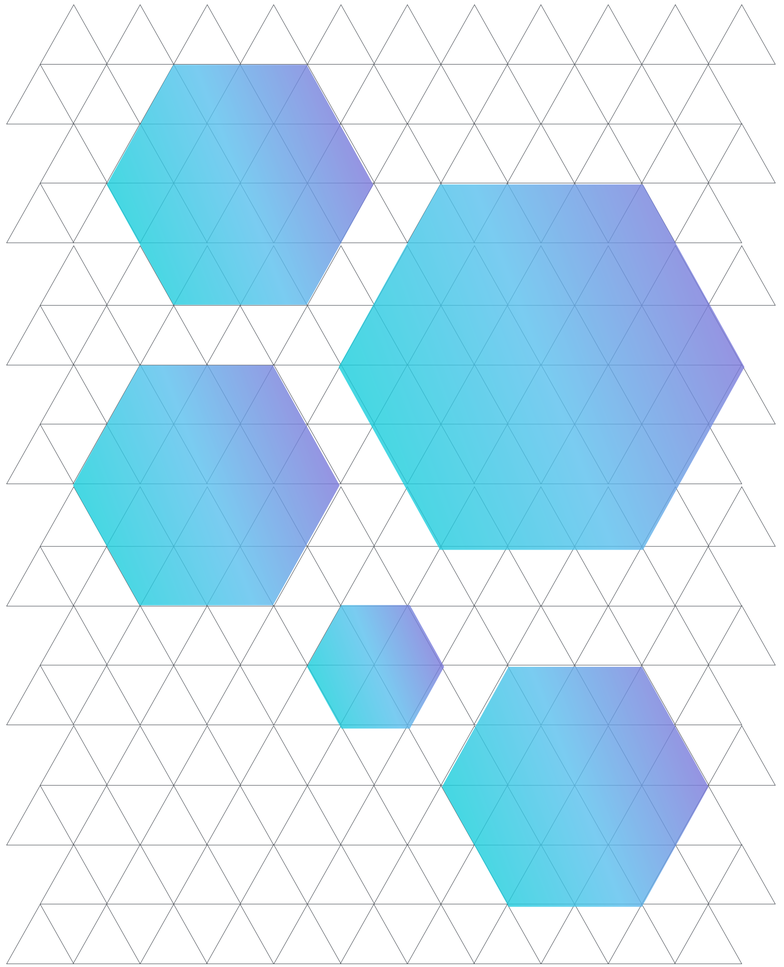
```
void testOC2()
{
    OCSomeClassA *a = [OCSomeClassA new];
    OCSomeClassB *b = (OCSomeClassB *)a;

    [b func2];

}
```

运行时确定类型

OC RUNTIME的基本概念

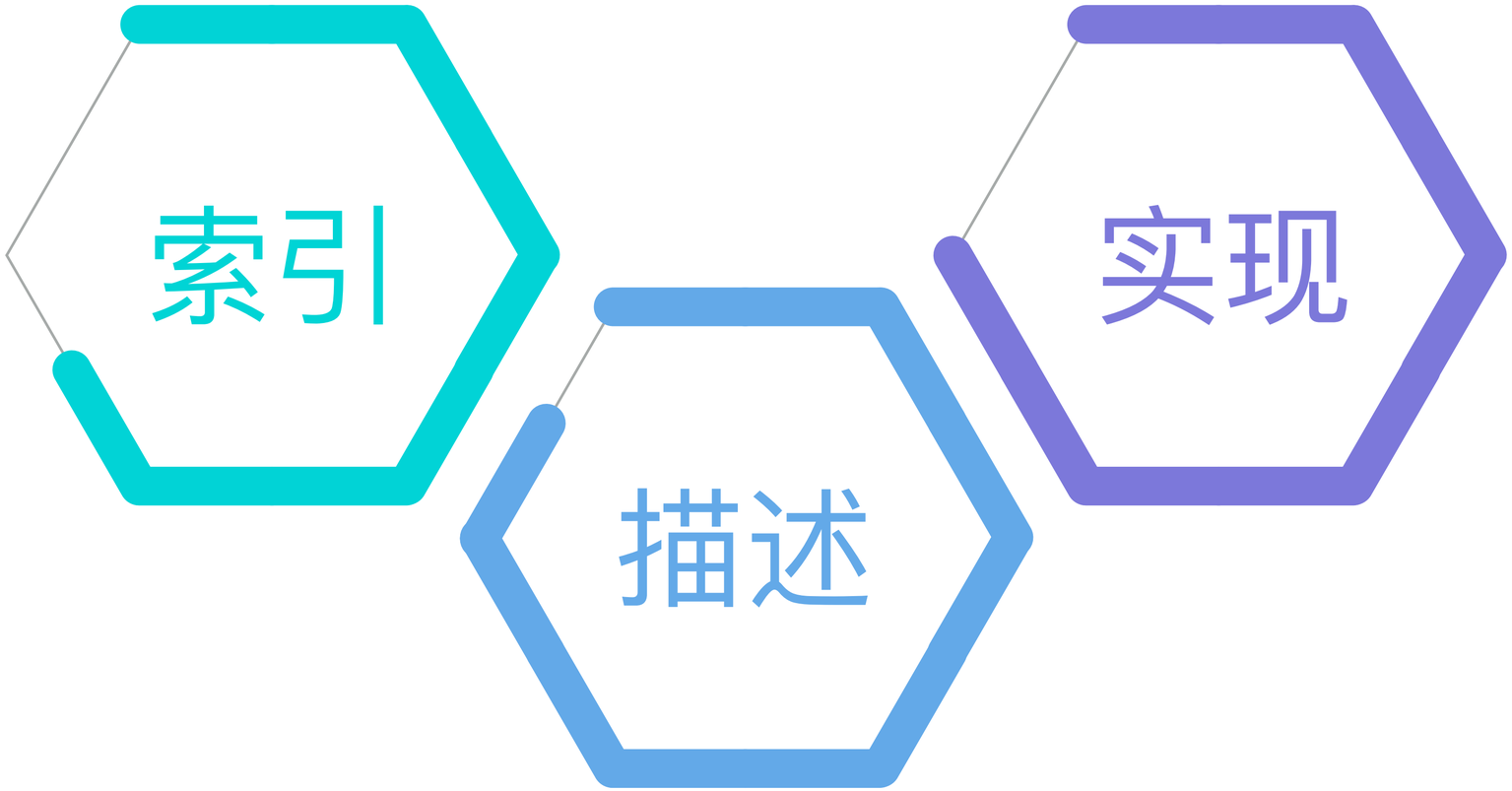
行为存储于类

- 方法的要素
- 方法的存储
- 实例如何找到方法



OC RUNTIME的基本概念

方法的基本要素



OC RUNTIME的基本概念

方法的索引——SEL

```
[a addObserver:b forKeyPath:@"c" options:NSKeyValueObservingOptionNew context:NULL];
[a removeObserver:b forKeyPath:@"c"];
SEL sel = @selector(addObserver:forKeyPath:options:context:);
const char *cstring = (const char *)sel;
NSString *string = [NSString stringWithUTF8String:cstring];
```

```
OCSomeClassB *b = [OCSomeClassB new];
[b performSelector:(SEL)"func2"];
```

- SEL是一个字符串
- 但字符串不是SEL
- SEL要保证唯一性
- SEL相关API

```
const char *s = sel_getName(@selector(func2));
SEL func2 = sel_getUid("func2");
SEL newSel = sel_registerName("newSelecotor:withBla:");
```

OC RUNTIME的基本概念

方法的描述——签名

- 什么是签名?

OC RUNTIME的基本概念

方法的描述——签名

- 什么是签名?
- 如何描述一个类型?

Table 6-1 Objective-C type encodings

Code	Meaning
`c`	A `char`
`i`	An `int`
`s`	A `short`
`l`	A `long` `l` is treated as a 32-bit quantity on 64-bit programs.
`q`	A `long long`
`C`	An `unsigned char`
`I`	An `unsigned int`
`S`	An `unsigned short`
`L`	An `unsigned long`
`Q`	An `unsigned long long`
`f`	A `float`
`d`	A `double`
`B`	A C++ `bool` or a C99 `_Bool`
`v`	A `void`
`*`	A character string (`char *`)
`@`	An object (whether statically typed or typed `id`)
`#`	A class object (`Class`)
`:`	A method selector (`SEL`)
[*array type*]	An array
{*name=type...*}	A structure
(*name=type...*)	A union
`b`*num*	A bit field of *num* bits
^*type*	A pointer to *type*
`?`	An unknown type (among other things, this code is used for function pointers)

OC RUNTIME的基本概念

方法的描述——签名

- 什么是签名?
- 如何描述一个类型?
- 如何描述一个方法?

```
Method method = class_getInstanceMethod([NSObject class],
                                        @selector(addObserver:
                                                  forKeyPath:
                                                     options:
                                                     context:));
const char *type = method_getTypeEncoding(method);
        NSString *s = [NSString stringWithUTF8String:type];
expect(s).to.equal(@"v48@0:8@16@24Q32^v40");
```

```
v48@0:8@16@24Q32^v40
```

```
@48@0:8{CGRect={CGPoint=dd}{CGSize=dd}}16
```

OC RUNTIME的基本概念

方法的实现

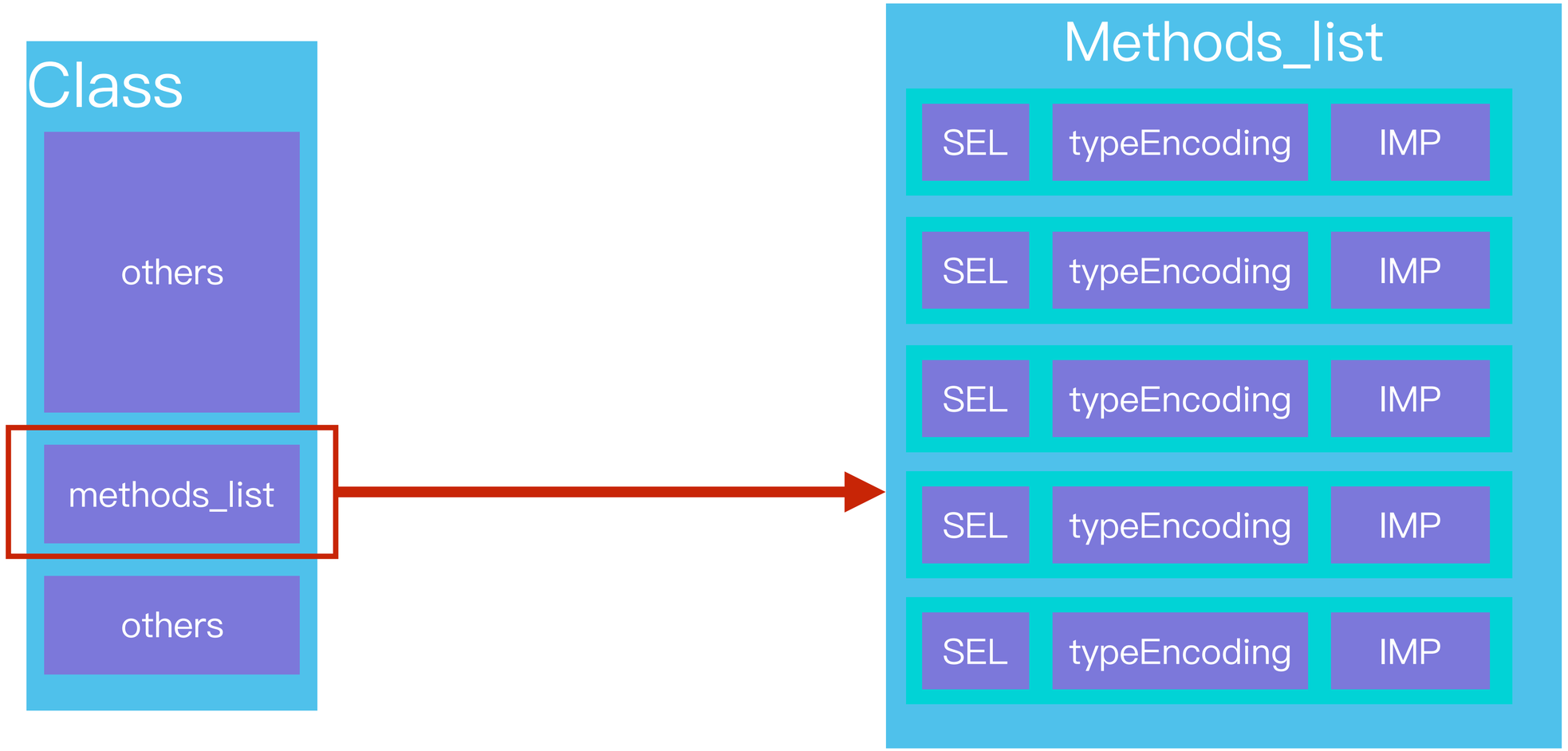
```
- (CGRect)makeCGRectWithOther:(CGRect)newCGRect
{
    return CGRectMake(self.myRect.origin.x + newCGRect.origin.x,
                      self.myRect.origin.y + newCGRect.origin.y,
                      self.myRect.size.width + newCGRect.size.width,
                      self.myRect.size.height + newCGRect.size.height);
}
```

```
CGRect realMakeCGRectWithOther(TestClass *self, SEL cmd, CGRect newCGRect) {
    return CGRectMake(self.myRect.origin.x + newCGRect.origin.x,
                      self.myRect.origin.y + newCGRect.origin.y,
                      self.myRect.size.width + newCGRect.size.width,
                      self.myRect.size.height + newCGRect.size.height);
}
```

IMP

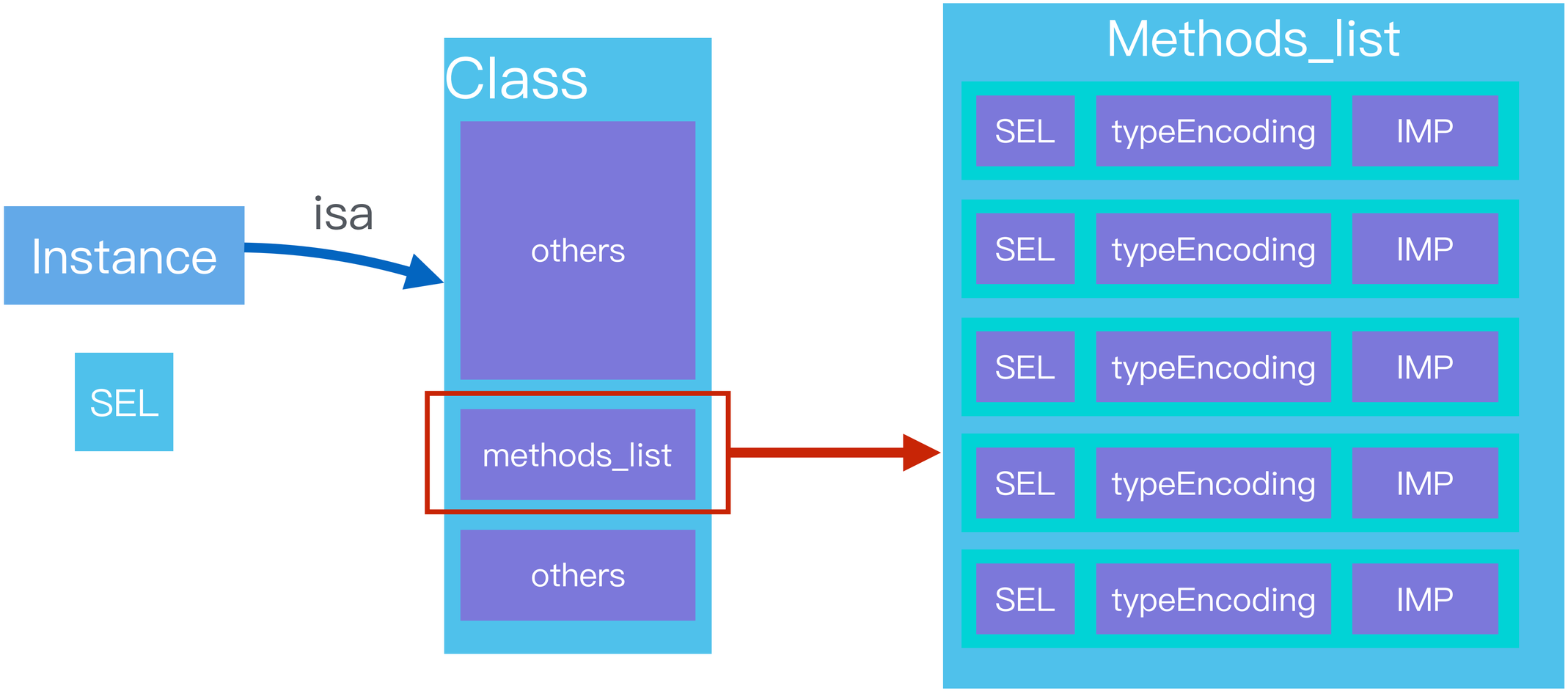
OC RUNTIME的基本概念

方法的存放



OC RUNTIME的基本概念

实例如何找到方法



OC RUNTIME的基本概念

方法相关API

```
•Method class_getInstanceMethod(Class cls, SEL name);
•Method class_getClassMethod(Class cls, SEL name);
•Method *class_copyMethodList(Class cls, unsigned int *outCount);
•BOOL class_addMethod(Class cls, SEL name, IMP imp, const char *types);
•IMP class_replaceMethod(Class cls, SEL name, IMP imp, const char *types);
•SEL method_getName(Method m);
•IMP method_getImplementation(Method m);
•const char *method_getTypeEncoding(Method m);
•IMP method_setImplementation(Method m, IMP imp);
•void method_exchangeImplementations(Method m1, Method m2);
```

OC RUNTIME的基本概念

状态存储于实例

- 成员变量与属性
- 成员变量定义与成员变量存储
- 实例的创建
- 成员的提取/修改

OC RUNTIME的基本概念

成员变量与属性

- 成员变量是状态，存于实例
- 属性是行为，存于类
- 属性是成员变量的外部表现形式
- 成员变量是属性的实现方式之一

```
@interface TestVarOrProperty : UIView {
    @public
    int var1;
    @private
    float var2;
    NSDictionary *var3;
    __weak UIView *var4;
}

@property (nonatomic, strong) UITableView *prop1;
@property (nonatomic, readonly, weak) UIView *prop2;

@end
```

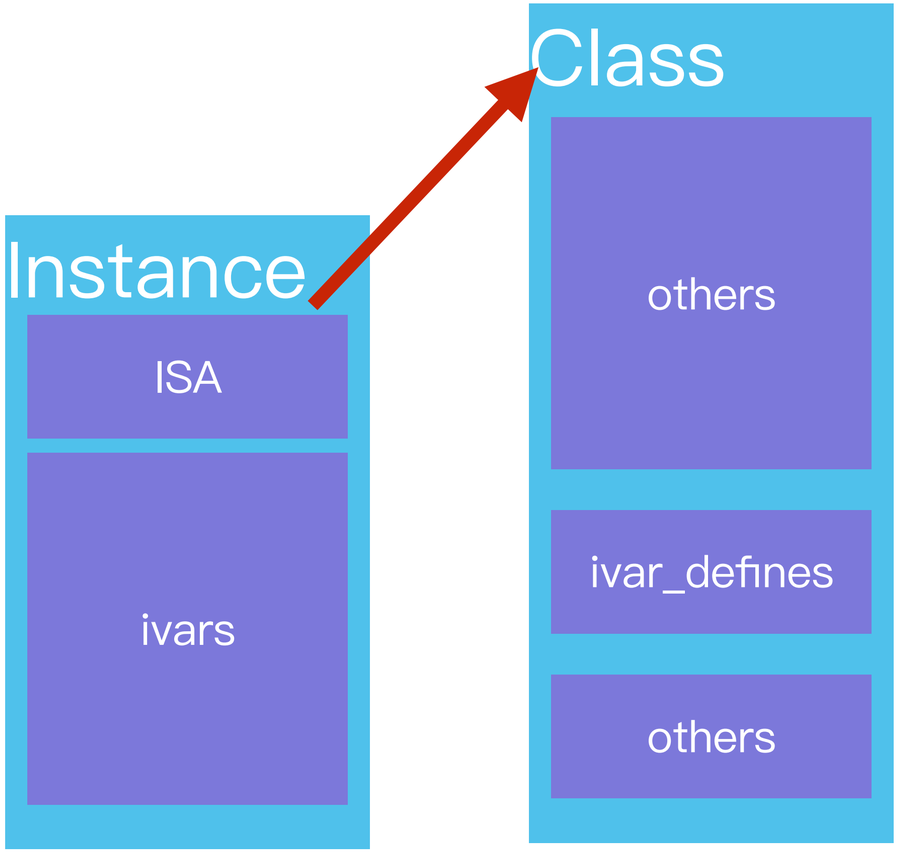
```
@implementation TestVarOrProperty
@synthesize prop2=var4;
@end
```

```
TestVarOrProperty *obj = [TestVarOrProperty new];
int var1 = obj->var1;
id obj2 = obj.prop1;
id obj3 = [obj prop1];
[obj setProp1:nil];
```

OC RUNTIME的基本概念

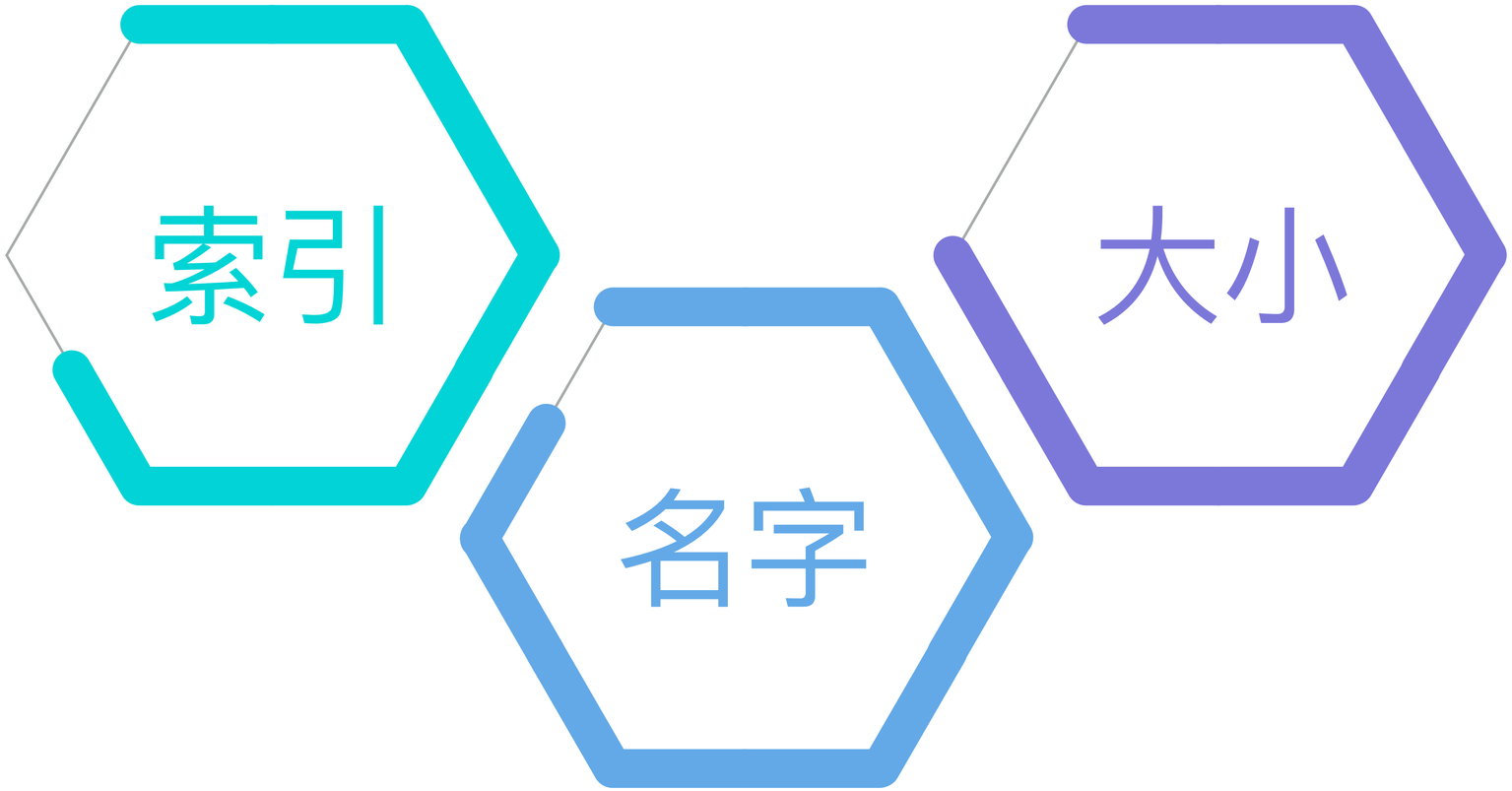
变量定义与变量存储

- 同一类的实例
  - 变量的定义相同
  - 变量的值不同
- 变量的定义存于类
- 变量存与实例



OC RUNTIME的基本概念

状态需要定义的基本要素

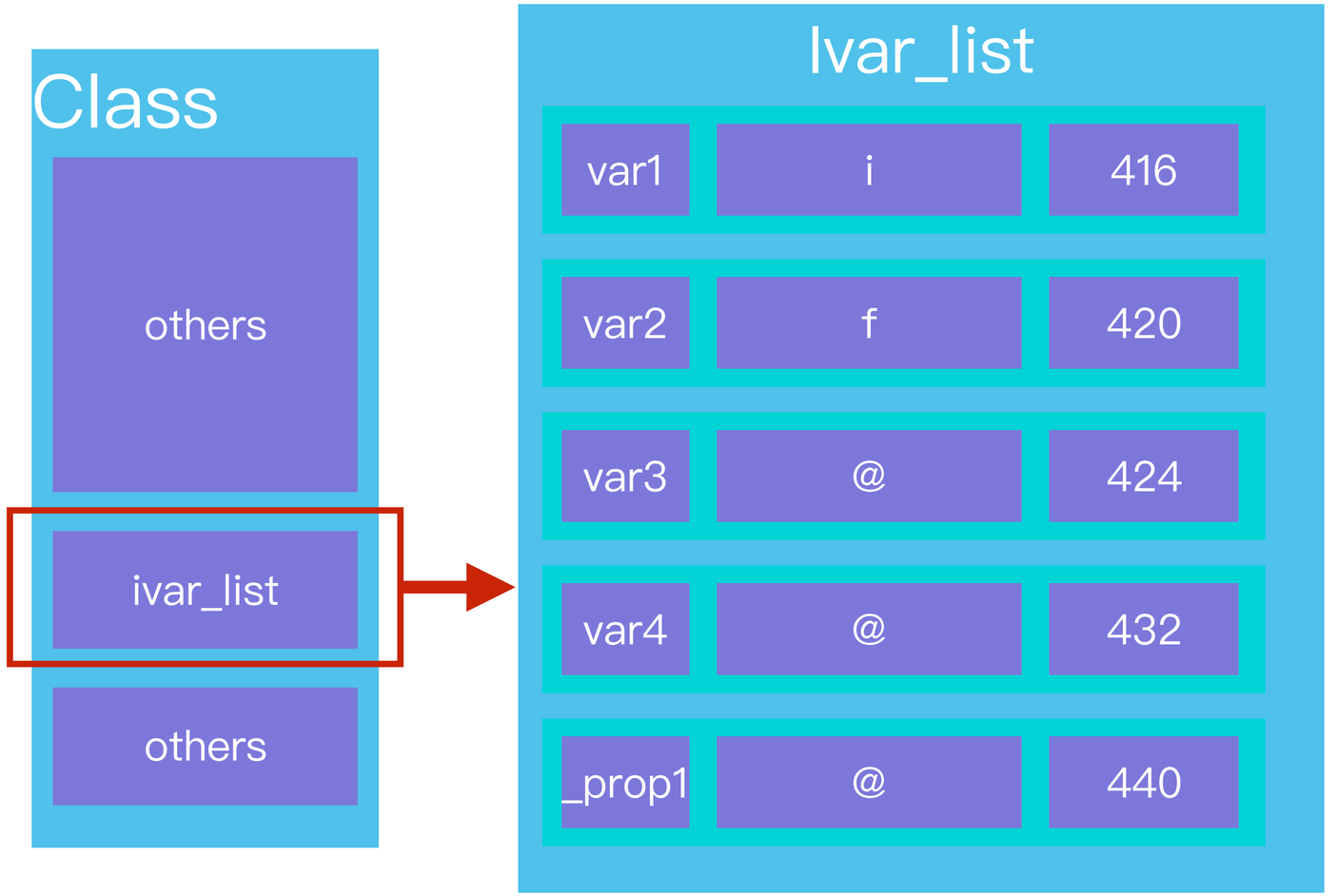


OC RUNTIME的基本概念

成员变量描述

```
@interface TestVarOrProperty : UIView {
    @public
    int var1;
    @private
    float var2;
    NSDictionary *var3;
    __weak UIView *var4;
}

@property (nonatomic, strong) UITableView *prop1;
@property (nonatomic, readonly, weak) UIView *prop2;

@end
```

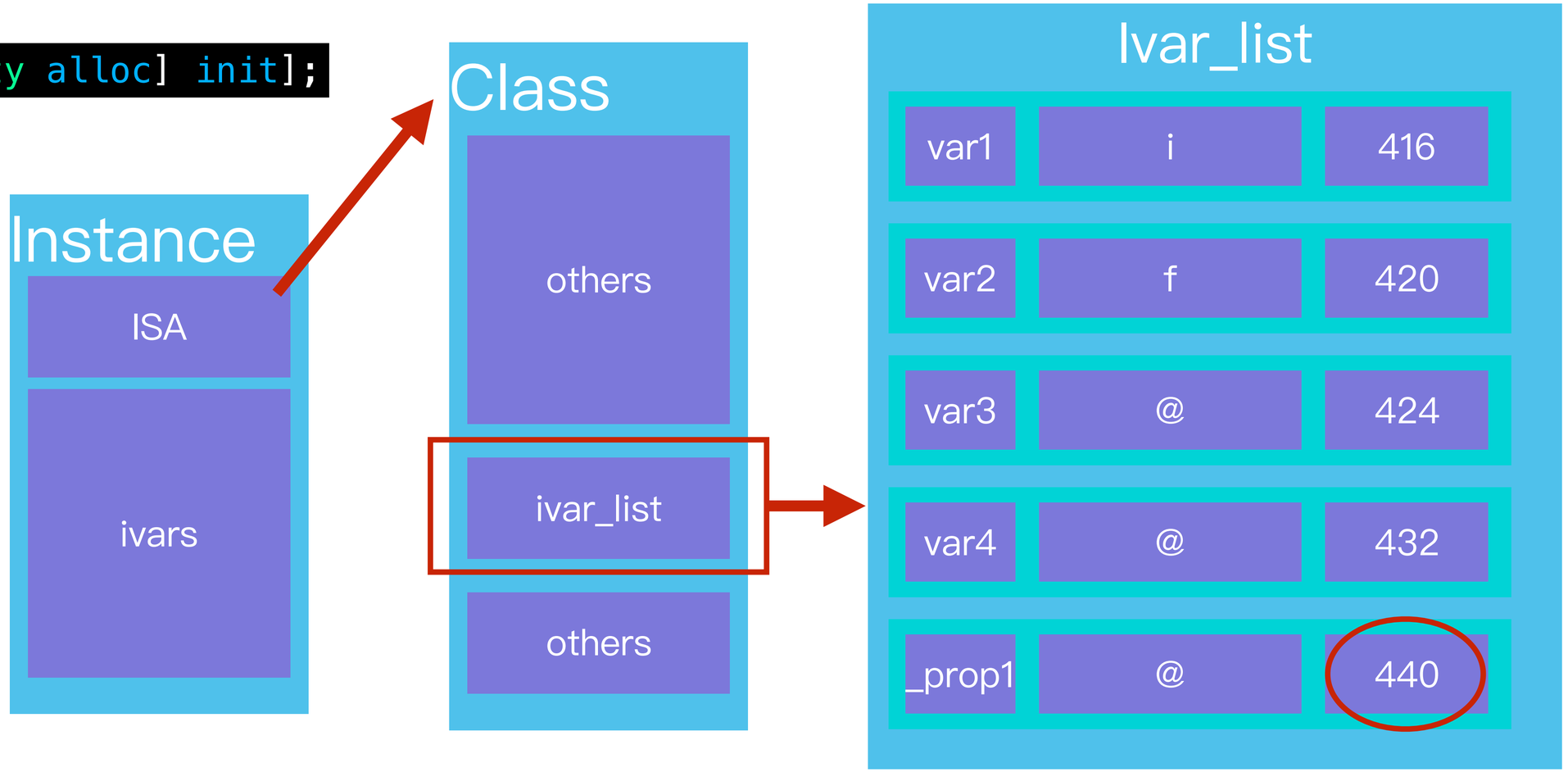
```
@implementation TestVarOrProperty
@synthesize prop2=var4;
@end
```



OC RUNTIME的基本概念

实例如何创建

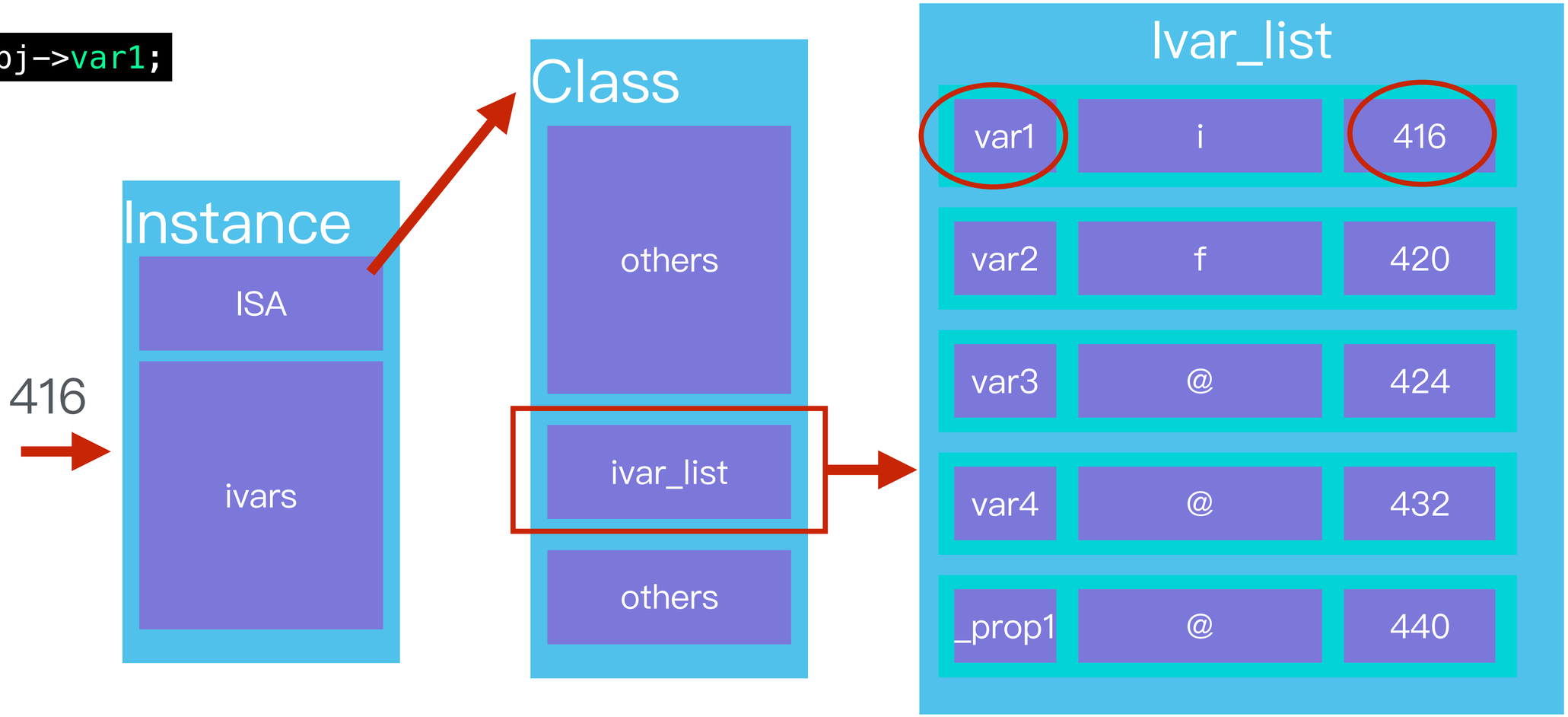
```
[[TestVarOrProperty alloc] init];
```



OC RUNTIME的基本概念

实例访问成员

```
int var1 = obj->var1;
```



OC RUNTIME的基本概念

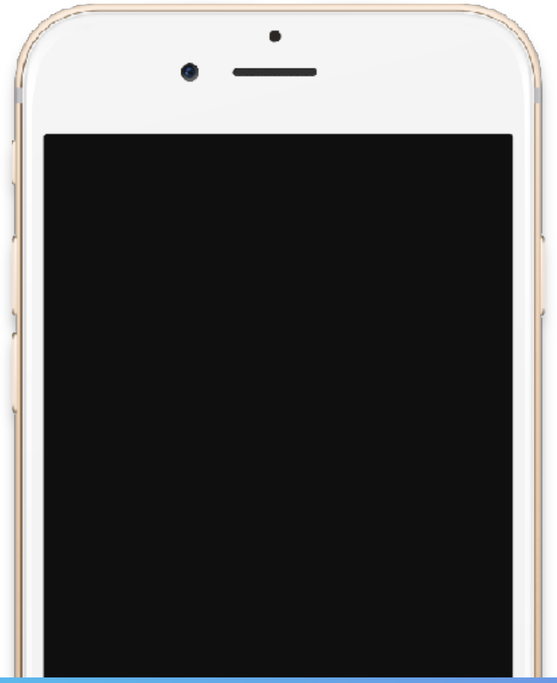
成员变量相关API

```
• id object_getIvar(id obj, Ivar ivar);
• void object_setIvar(id obj, Ivar ivar, id value);
• Ivar class_getInstanceVariable(Class cls, const char *name);
• Ivar *class_copyIvarList(Class cls, unsigned int *outCount);
• BOOL class_addIvar(Class cls, const char *name, size_t size, uint8_t alignment, const char *types);
• const char *ivar_getName(Ivar v);
• const char *ivar_getTypeEncoding(Ivar v);
• ptrdiff_t ivar_getOffset(Ivar v);
```

OC RUNTIME的基本概念

万物皆对象?

对象



非对象

- 实例
- 类

- Method
- iVar

Q&A

感谢聆听