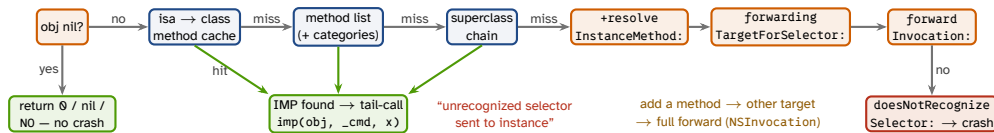
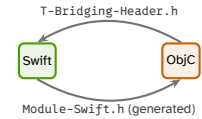


In one line: An ObjC call is a **message**: [obj sel] becomes objc_msgSend(obj, @selector(sel)), resolved **at runtime** in the receiver's class — hence nil messaging, categories, KVO, swizzling. Swift enters that world via @objc (*visible* to the runtime) and dynamic (*dispatched* through it), on NSObject-rooted classes.

[obj tap:x] → objc_msgSend(obj, @selector(tap:), x)



two headers, two directions



How it works — ObjC

- SEL = interned name; IMP = C fn (id self, SEL _cmd, ...). Cached lookup, never inlined. **nil** receiver returns zero — a silent bug, not a crash. id = any object, unchecked (Swift: Any).
- Properties:** default atomic (accessor lock only, not thread safety). strong own · weak zeroing · copy for NSString/blocks (caller may pass a mutable one) · assign scalars only (an object dangles; delegates are weak) · unsafe_unretained not zeroed.
- Blocks** capture locals **by value** at creation and **retain** captured objects; __block = by reference, mutable. Born on the stack, ARC copies to heap when stored. self → block → self cycle: weak/strong dance.
- Categories** add methods (not ivars — use associated objects) to any class. Same selector twice = undefined winner ⇒ prefix. @interface C () = private extension in .m.
- KVC** valueForKeyPath:@"a.b", @sum.x; bad key → NSUnknownKeyException. **KVO** isa-swizzles the object to hidden NSKVONotifying_C whose setter wraps will/didChangeValueForKey:: fires on the *mutating* thread.
- Swizzle** with method_exchangeImplementations in +load + dispatch_once (+load: pre-main, per class and category; +initialize: lazy, may run twice).

Example

```

@property (nonatomic, copy) NSString *title;
@property (nonatomic, weak, nullable) id<PlayerDelegate> delegate;
@property (nonatomic, copy) void (^onEnd)(void);
__weak typeof(self) weakSelf = self;
self.onEnd = ^{
    __strong typeof(weakSelf) s = weakSelf; // alive for the body
    if (!s) return;
    [s.delegate playerDidEnd:s]; // nil delegate: no-op
}

```

```

final class Model: NSObject {
    @objc dynamic var progress = 0.0 // KVO works
    @objc var name = "" // visible, NOT KVO-safe
    @objc(loadWithID:) func load(id: Int) {}
    let token = model.observe(\.progress, options: [.new]) { m, c in
        print(c.newValue as Any) // hold token; its deinit removes
    }
}

```

ObjC → Swift import

id	Any	Class	AnyClass
instancetype	Self	SEL	Selector
BOOL	Bool	block	@convention(block) closure
NSInteger	Int	category	extension

How it works — interop

- @objc exposes (selector, -Swift.h) but may still be vtable-called; dynamic forces objc_msgSend ⇒ KVO, swizzling. @objcMembers = all members. Swift 4 (SE-0160) stopped inferring @objc.
- @objc class must inherit NSObject; @objc protocol is class-only, allows optional.
- Bridging header: app/test targets (frameworks use umbrella header). -Swift.h: internal+ in an app, public in a framework; #import in .m, @class in .h.
- nonnull→T · nullable→T? · unannotated→T!; use NS_ASSUME_NONNULL_BEGIN.
- NS_SWIFT_NAME renames · NS_REFINED_FOR_SWIFT → __name · NS_ENUM→enum (@unknown default) · NS_OPTIONS→OptionSet · NSArray<NSString*>→[String] (else [Any]) · NSError **↔throws.
- Can't cross:** structs (bar bridged String/Data/URL...), enums with payloads (only @objc enum E: Int), generics, tuples, Int? (use NSNumber?), PATs / Swift-only protocols. Default args cross *without* defaults.

Interview traps

- ! @objc alone: Swift callers use the vtable and bypass KVO's setter or a swizzled IMP — observation silently never fires.
- ! Swizzling: global, order-dependent, _cmd changes; on an *inherited* method it swaps the superclass's (class_addMethod first); breaks across iOS versions.
- ! [Int]→NSArray boxes each element in NSNumber; String↔NSString may transcode — costly in a loop.
- ! Overloads collide as one selector — @objc(name:). No @objc on an IB action → “unrecognized selector”.
- ! Header says nonnull, ObjC returns nil → Swift crashes on use. @Published is Combine, not KVO.

Remember

“@objc shows, dynamic sends.” nil eats messages; categories add methods not storage; KVO = a secret subclass.

Likely questions

1. Message to nil? — zero value back; no crash.
2. copy for NSString? — snapshot a possibly mutable arg.
3. __block? — capture by reference so the block can mutate.
4. How does KVO work? — isa-swizzled subclass overrides the setter.
5. Unknown selector? — resolve → forwardingTarget → forwardInvocation → crash.