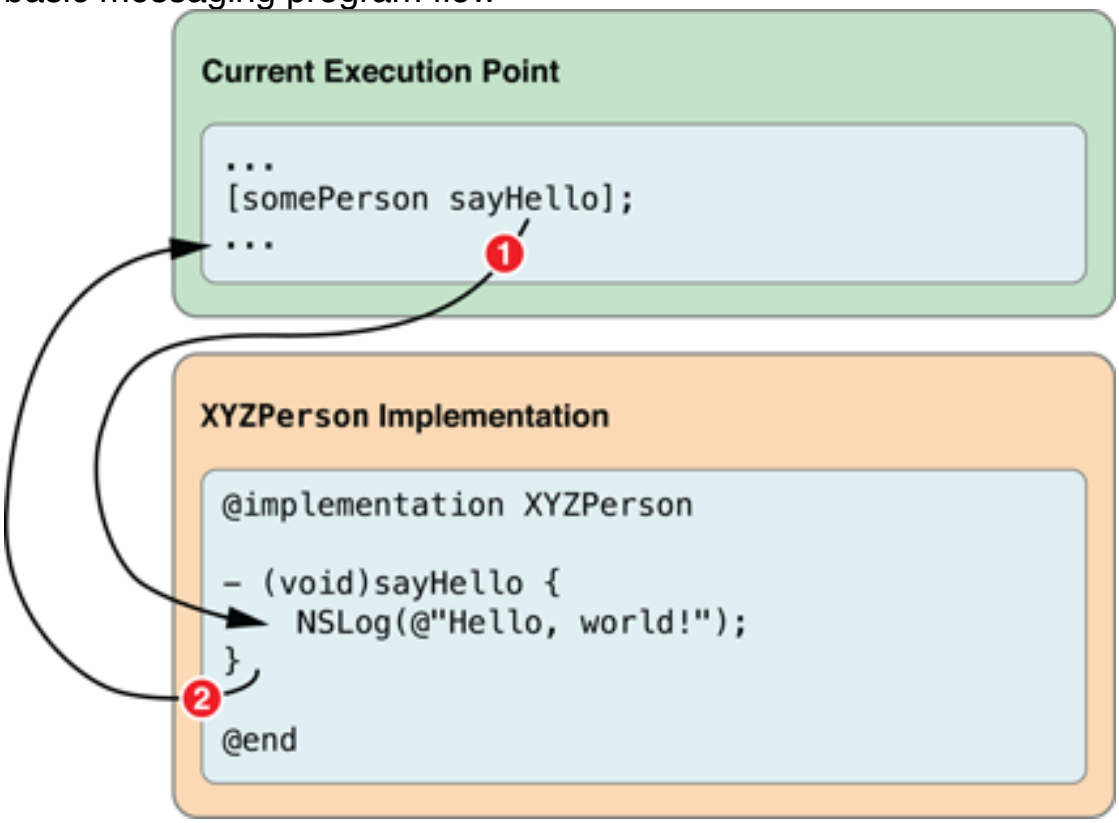
Working with Objects

Objects Send and Receive Messages
basic messaging program flow

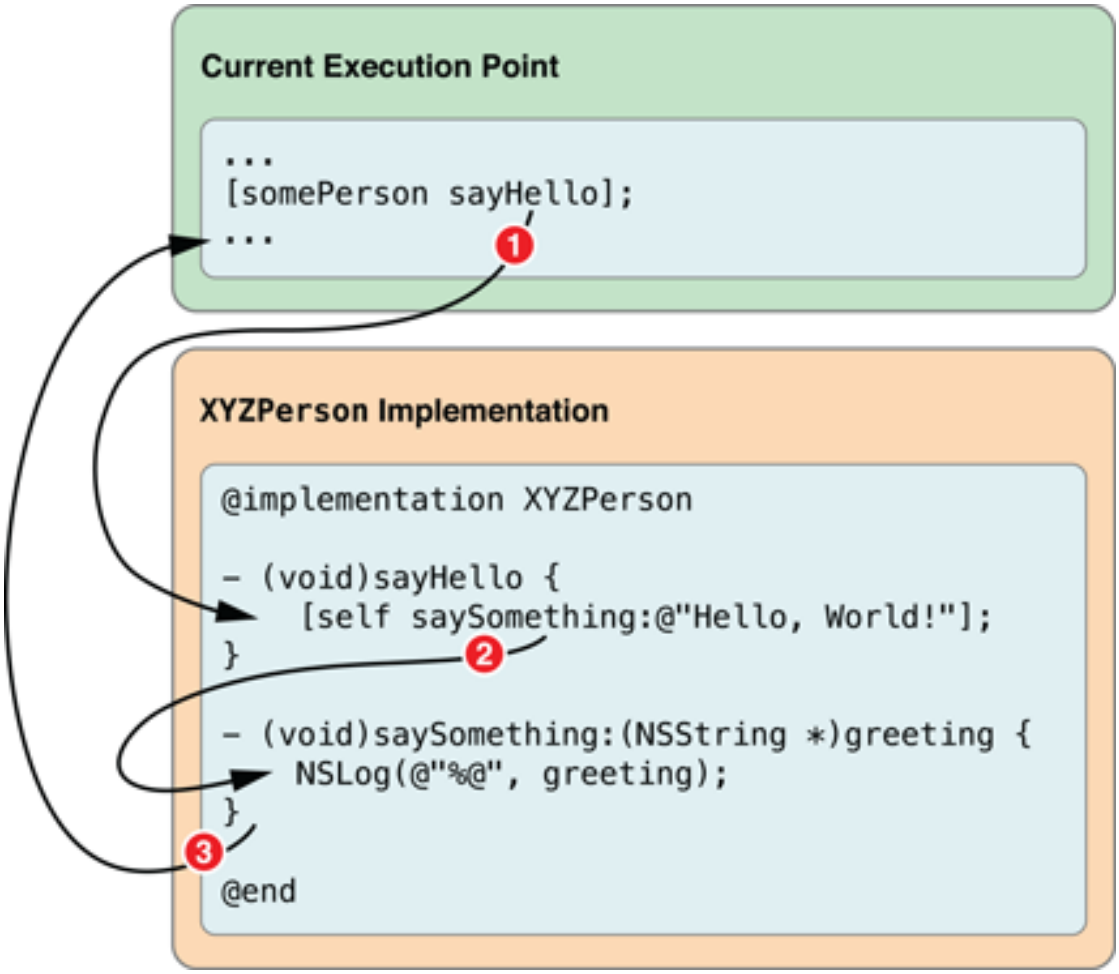


> Use Pointers to Keep Track of Objects

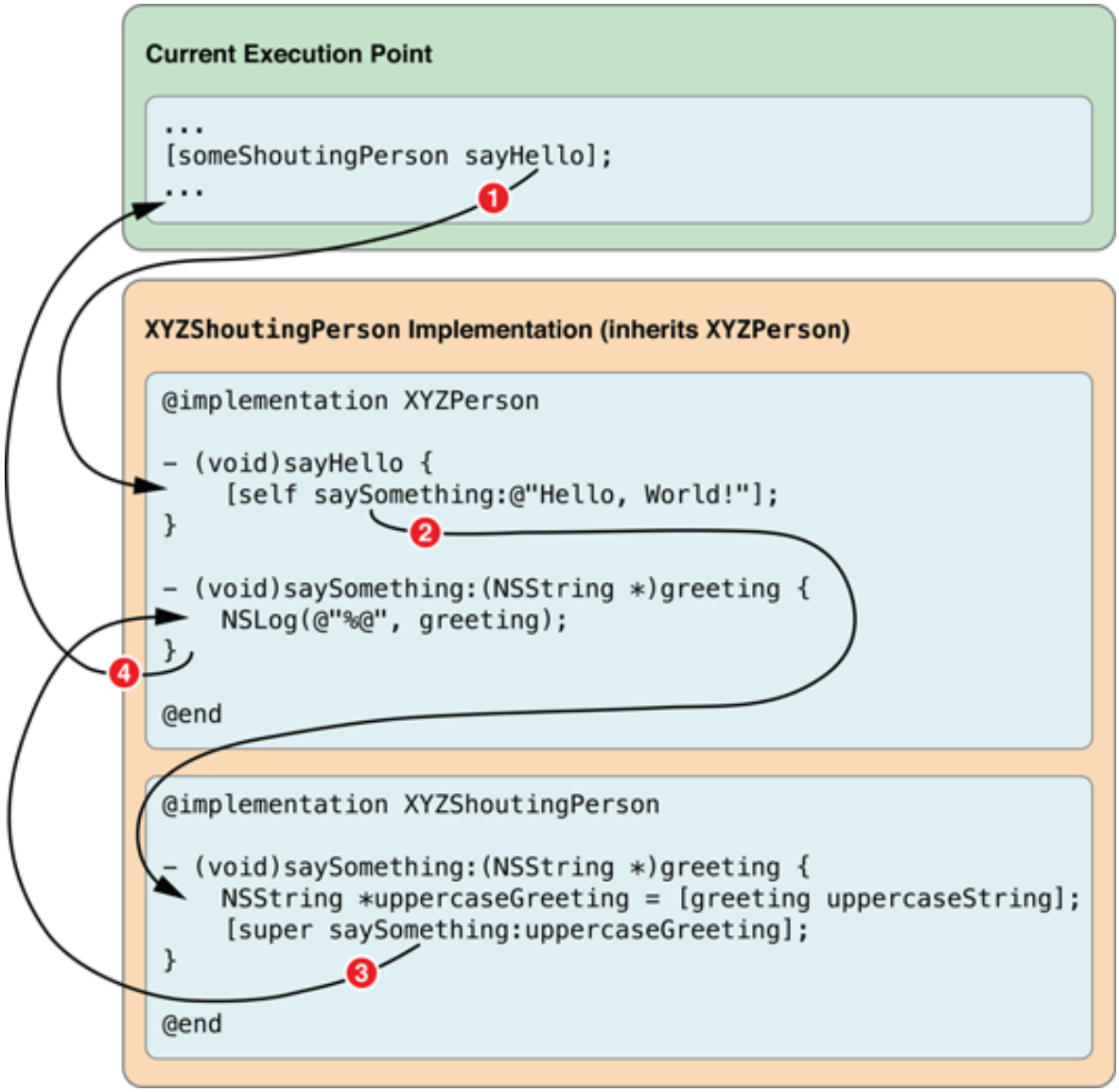
> You Can Pass Objects for Method Parameters

> Methods Can Return Values

> Objects Can Send Messages to Themselves
program flow when messaging self



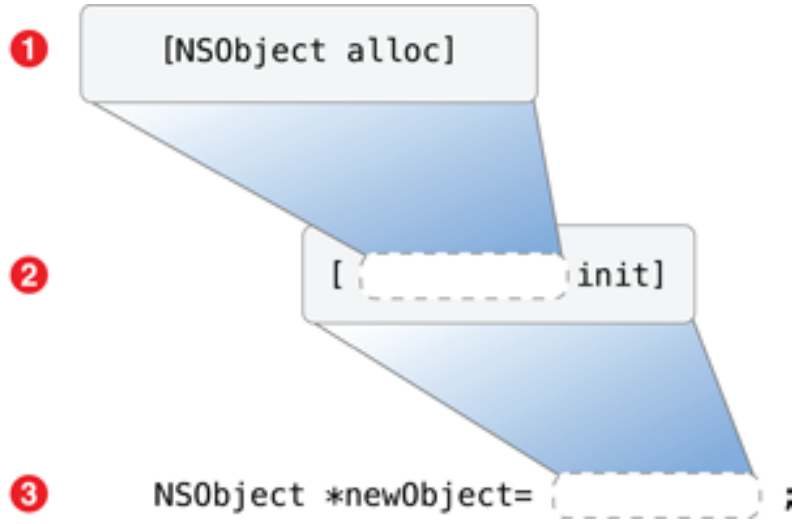
> Objects Can Call Methods Implemented by Their Superclasses
Program flow when messaging super



Objects Are Created Dynamically

alloc
+ (id)alloc;
1. Make sure enough memory is allocated not only for the properties defined by an object's class, but also the properties defined on each of the superclasses in its inheritance chain.
2. Clear out the memory allocated for the object's properties by setting them to zero.

init
- (id)init;
1. Make sure its properties have suitable initial values at creation.
NSObject *newObject = [[NSObject alloc] init];



> Initializer Methods Can Take Arguments

> Class Factory Methods Are an Alternative to Allocation and Initialization

> Use new to Create an Object If No Arguments Are Needed for Initialization

```
XYZObject *object = [XYZObject new];
// is effectively the same as:
XYZObject *object = [[XYZObject alloc] init];
```

> Literals Offer a Concise Object-Creation Syntax

Objective-C Is a Dynamic Language

> Determining Equality of Objects

1. When dealing with objects, the == operator is used to test whether two separate pointers are pointing to the same object:
2. If you need to test whether two objects represent the same data, you need to call a method like isEqual:

> Working with nil