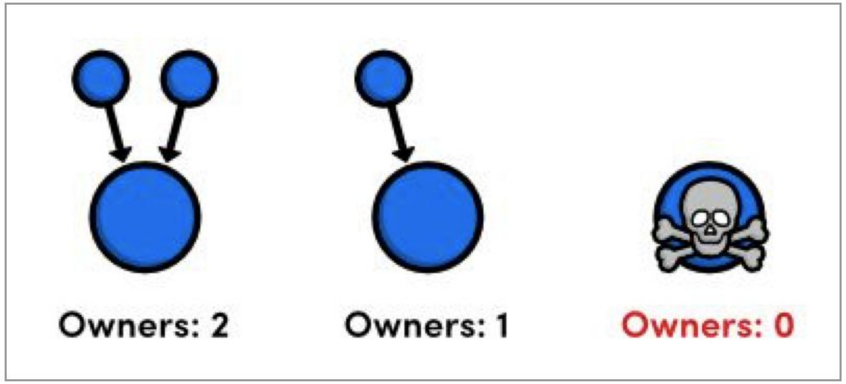
Memory Management

Objective-C
MADP - 401

Memory Management

With the advent of **Automatic Reference Counting (ARC)**, the compiler manages all of your object ownership automatically. This means that you'll never to worry about how the memory management system actually works. But, you do have to understand the **strong**, **weak** and **copy** attributes of **@property**, since they tell the compiler what kind of relationship objects should have.



Destroying an object with no owners

Managing Objects

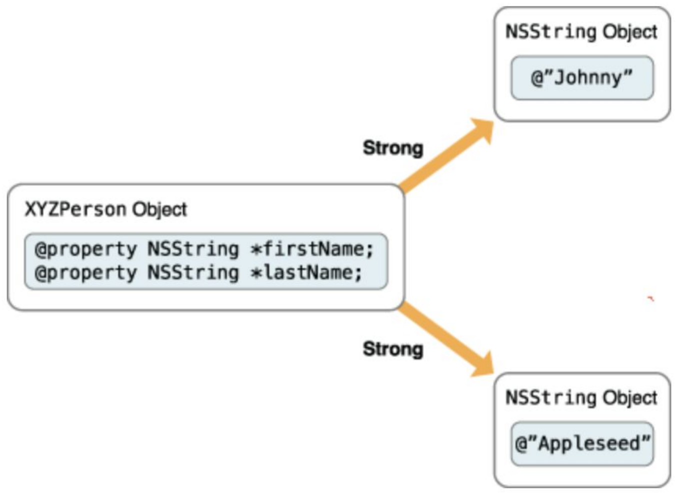
Memory for Objective-C objects is allocated dynamically (on the heap), which means you need to use pointers to keep track of an object’s address.

Unlike scalar values, it’s not always possible to determine an object’s lifetime by the scope of one pointer variable.

Instead, an object must be kept active in memory for as long as it is needed by other objects.

Strong relationship

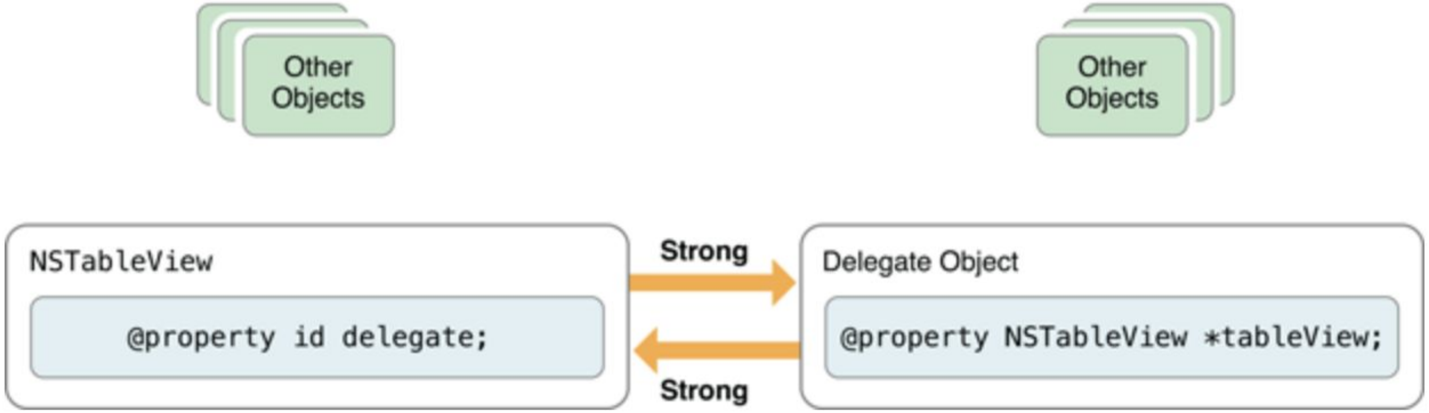
Figure 3-2 Strong Relationships



When an XYZPerson object is deallocated from memory, the two string objects will also be deallocated, assuming there aren't any other strong references left to them.

Avoid Strong Reference Cycles!

Figure 3-8 A strong reference cycle



```
@property id delegate;
```

```
@property NSTableView *tableView;
```

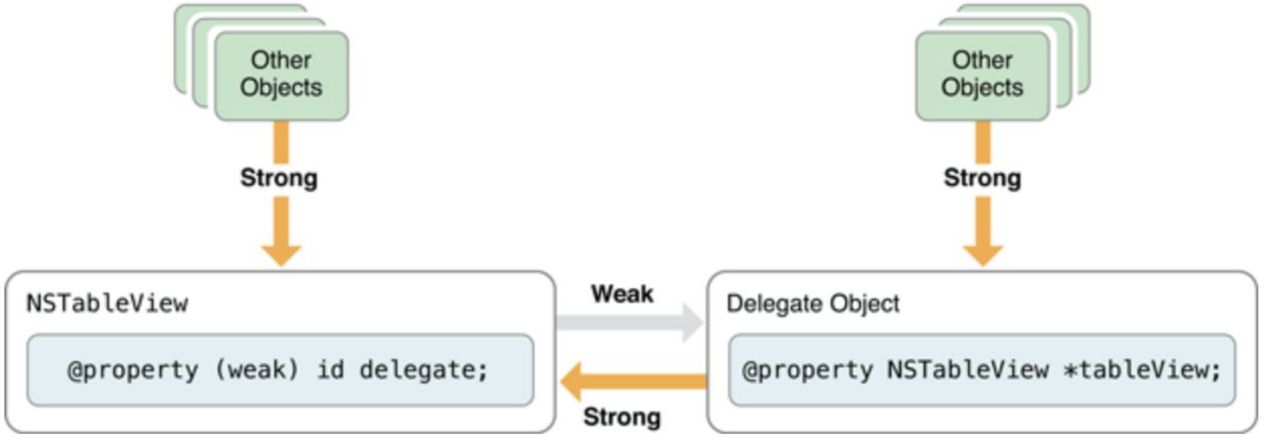
Retain Cycle

Although strong references work well for one-way relationships between objects, you need to be careful when working with groups of interconnected objects. If a group of objects is connected by a circle of strong relationships, they keep each other alive even if there are no strong references from outside the group.

This is called ***Retain Cycle***

Right way!

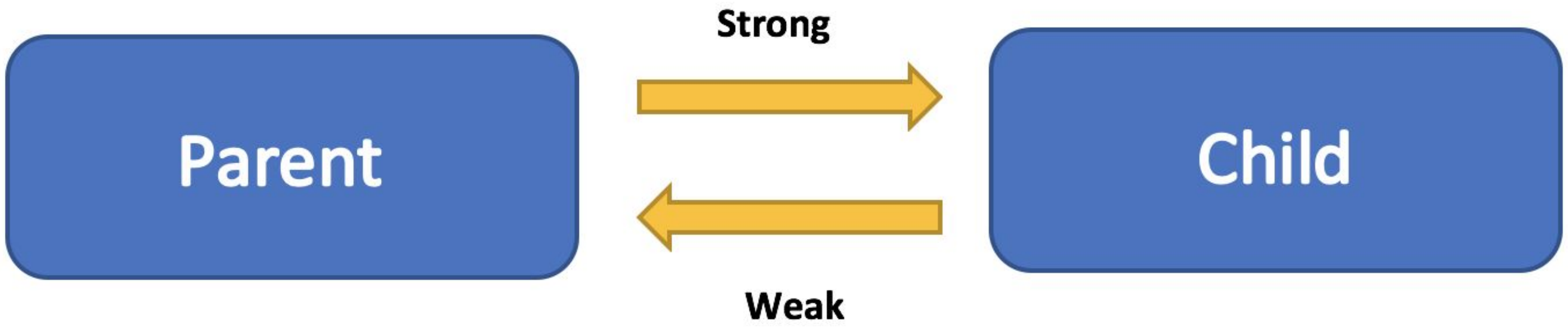
Figure 3-9 The correct relationship between a table view and its delegate



```
@property (weak) id delegate;
```

```
@property NSTableView *tableView;
```

To Remember!



strong attribute

There’s no need to specify the strong attribute explicitly, because it is the default.

A variable maintains a strong reference to an object only as long as that variable is in scope, or until it is reassigned to another object or **nil**.

Examples:

```
@property id delegate;
@property (strong) id delegate;
```

weak attribute

If you don’t want a variable to maintain a strong reference, you can declare it as weak.

Because a weak reference doesn’t keep an object alive, it’s possible for the referenced object to be deallocated while the reference is still in use. To avoid a dangerous dangling pointer to the memory originally occupied by the now deallocated object, a weak reference is automatically set to nil when its object is deallocated.

Example: `@property (weak) id delegate;`

copy attribute

The copy attribute is an alternative to strong.
 Keep its own copy of any objects that are set for its properties.

Any object that you wish to set for a copy property must support **NSCopying**, which means that it should conform to the **NSCopying** protocol.

Let's see a demo!