Swift Reference Types

reference cycles: strong, weak, and unowned

Glenn Cole, April 2020

Strong reference

released when set to nil or owner deallocated

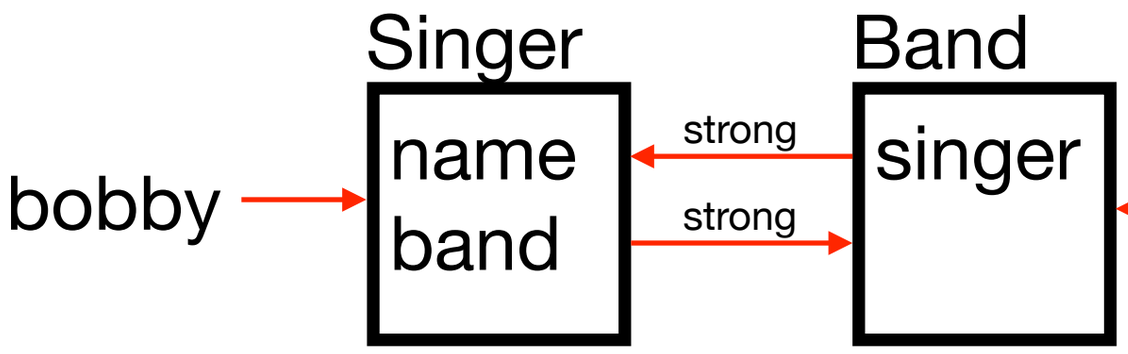
```
class Band {
    var singer: Singer?

    init(singer: Singer? = nil) {
        self.singer = singer
    }

    deinit {
        print("\(Self.self)", #function)
    }
}
```

```
class Singer {
    let name: String
    let band: Band

    init(name: String, band: Band) {
        self.name = name
        self.band = band
    }

    deinit {
        print("\(Self.self)", #function)
    }
}
```

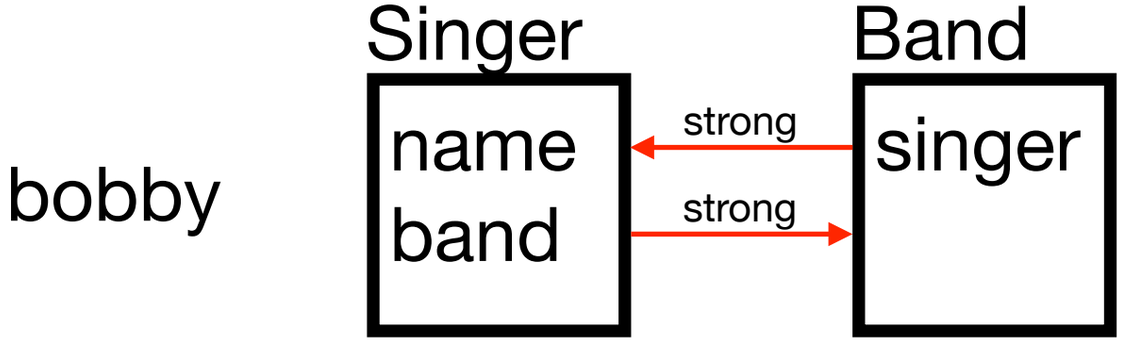
```
// following slide assumes:
//
var dragon: Band! = Band()
var bobby: Singer!
bobby = Singer(name:"Bobby", band:dragon)
dragon.singer = bobby

// vars are Optionals so we can assign
// nil to them to deallocate,
// and force-unwrapped for ease of use
```

Strong reference

released when set to nil or owner deallocated




```
dragon = Band()
bobby = Singer(..., dragon)
dragon.singer = bobby
```




```
bobby = nil
dragon = nil
// Singer and Band still
// have a strong reference
// to each, so they will
// NOT be deallocated.
// Instant memory leak!
```

Weak reference

released when no more strong references to target

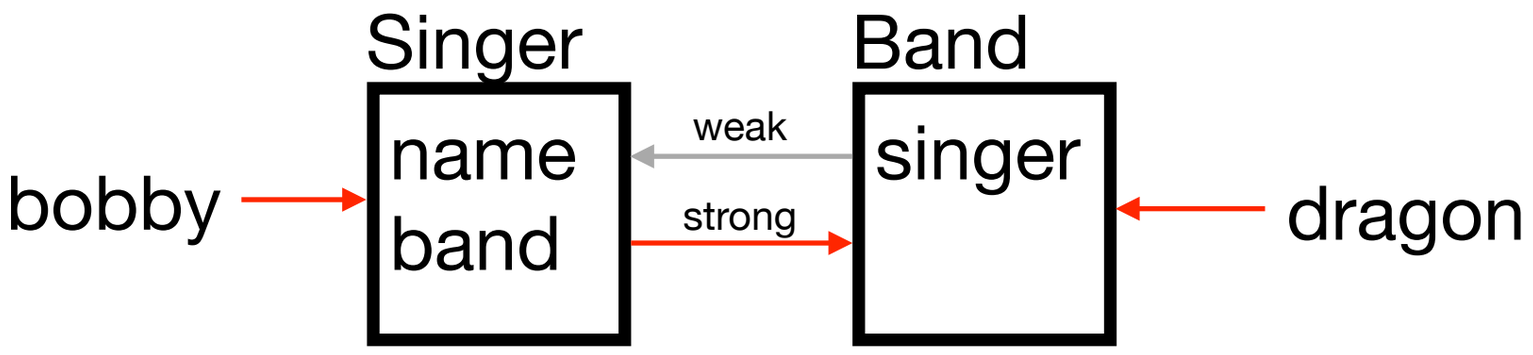
```
class Band {
    weak var singer: Singer?

    init(singer: Singer? = nil) {
        self.singer = singer
    }

    deinit {
        print("\(Self.self)", #function)
    }
}
```

```
class Singer {
    let name: String
    let band: Band

    init(name: String, band: Band) {
        self.name = name
        self.band = band
    }

    deinit {
        print("\(Self.self)", #function)
    }
}
```

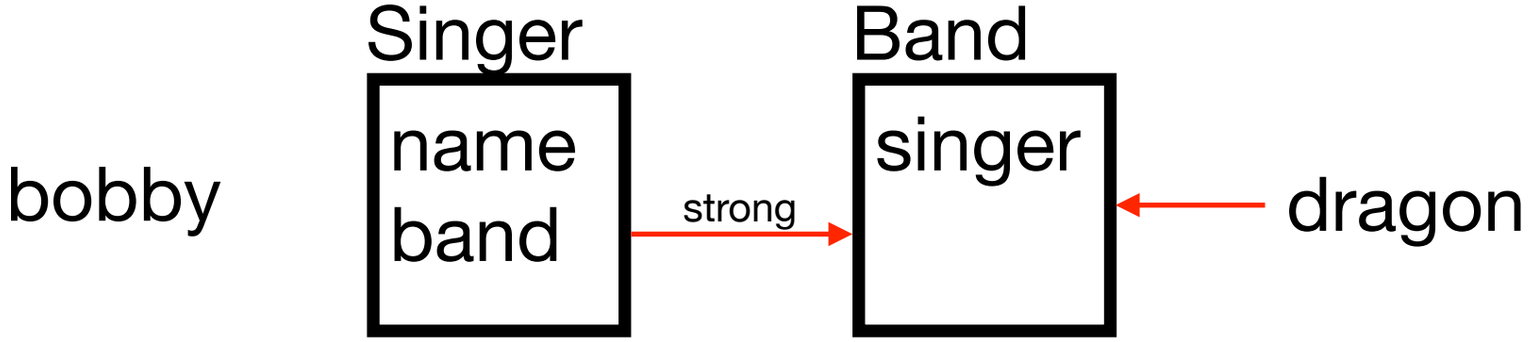
```
// following slides assume:
//
var dragon: Band! = Band()
var bobby: Singer!
bobby = Singer(name:"Bobby", band:dragon)
dragon.singer = bobby

// vars are Optionals so we can assign
// nil to them to deallocate,
// and force-unwrapped for ease of use
```

Weak reference

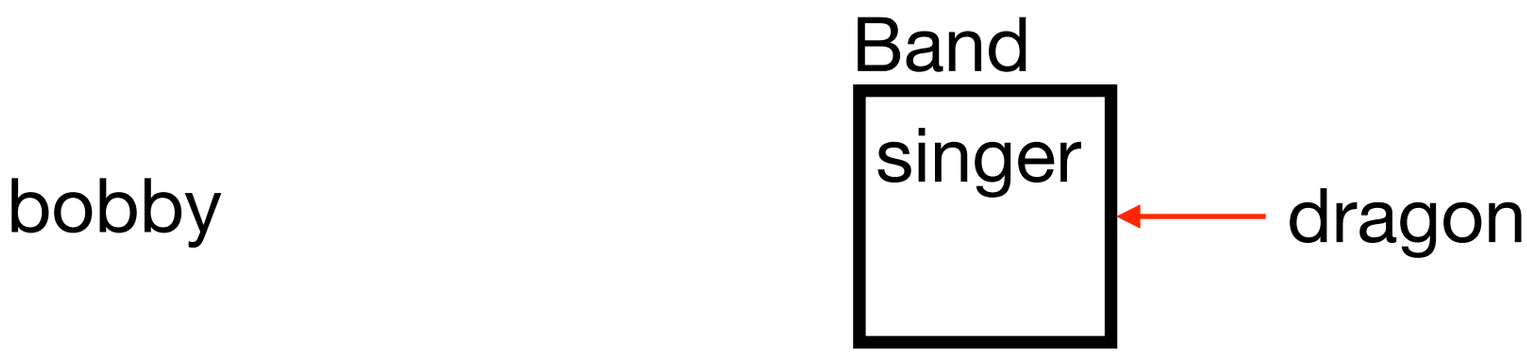
use when instance lifetime > target lifetime




```
dragon = Band()
bobby = Singer(..., dragon)
dragon.singer = bobby
```



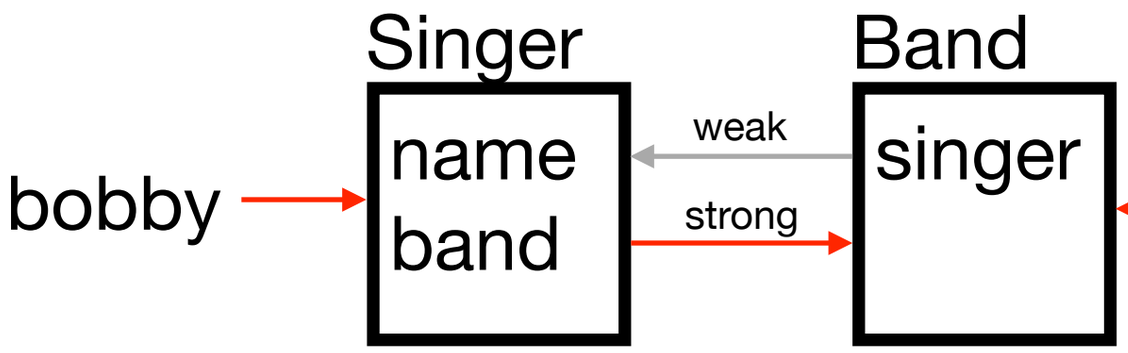

```
bobby = nil
// no more strong refs
// to Singer instance,
// so weak reference
// dragon.singer is
// automatically set to nil
```



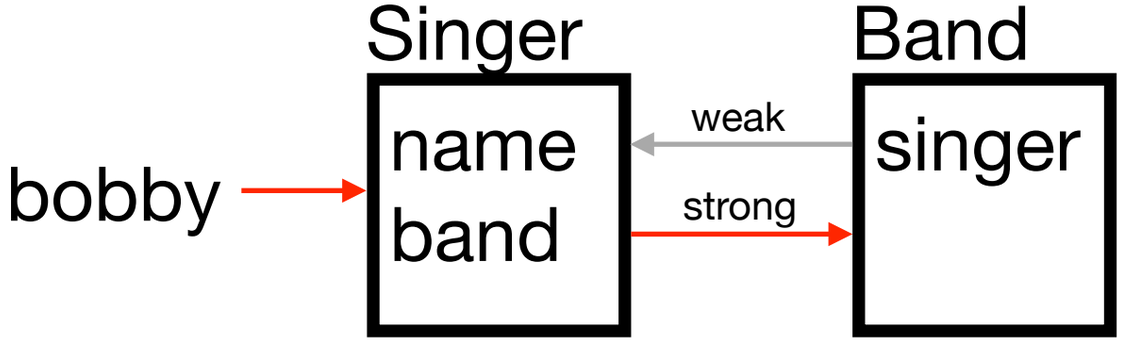

```
// ...so now no more ref to
// Singer, so deallocated
//
// when dragon set to nil,
// Band will be deallocated
```

Weak reference

what if we release the other one first?




```
dragon = Band()
bobby = Singer(..., dragon)
dragon.singer = bobby
```



dragon

```
dragon = nil
// Singer and Band instances
// still have strong refs
// to each (but COULD assign
// bobby.band = nil to
// release Band instance)
```

bobby

dragon

```
bobby = nil
// no more strong ref to
// Singer, so deallocated,
// so no more strong ref to
// Band, so deallocated
```

Unowned reference

expects target to outlive instance variable

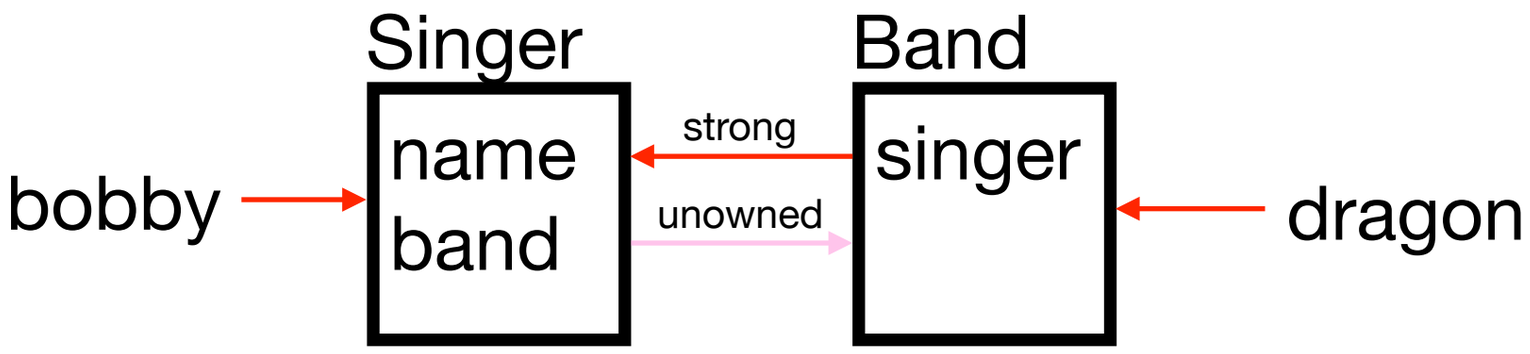
```
class Band {
    var singer: Singer?

    init(singer: Singer? = nil) {
        self.singer = singer
    }

    deinit {
        print("\(Self.self)", #function)
    }
}
```

```
class Singer {
    let name: String
    unowned let band: Band

    init(name: String, band: Band) {
        self.name = name
        self.band = band
    }

    deinit {
        print("\(Self.self)", #function)
    }
}
```

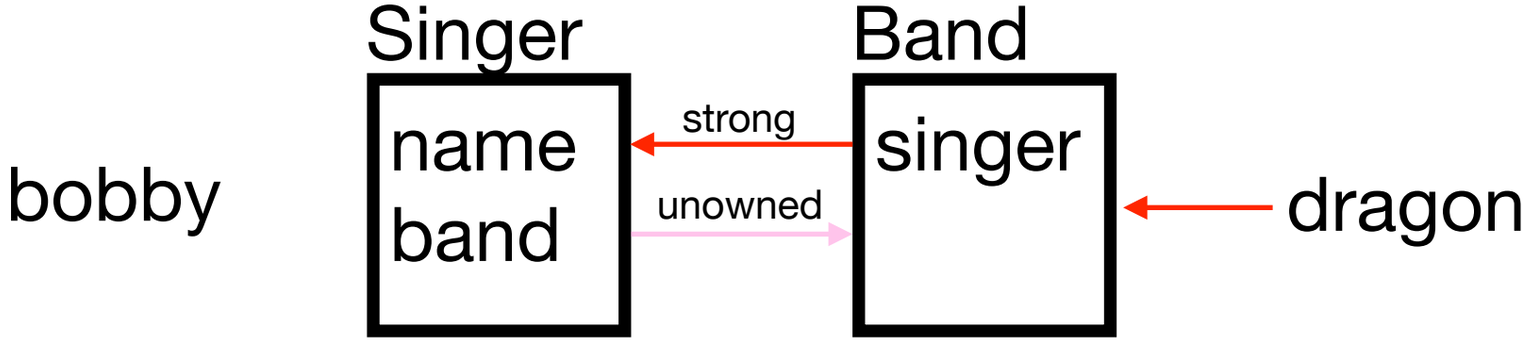
```
// following slides assume:
//
var dragon: Band! = Band()
var bobby: Singer!
bobby = Singer(name:"Bobby", band:dragon)
dragon.singer = bobby

// vars are Optionals so we can assign
// nil to them to deallocate,
// and force-unwrapped for ease of use
```

Unowned reference

use when instance lifetime ≤ target lifetime




```
dragon = Band()
bobby = Singer(..., dragon)
dragon.singer = bobby
```




```
bobby = nil
// Band and Singer instances
// still have strong refs
// to each (but COULD assign
// dragon.singer = nil to
// release Singer instance)
```

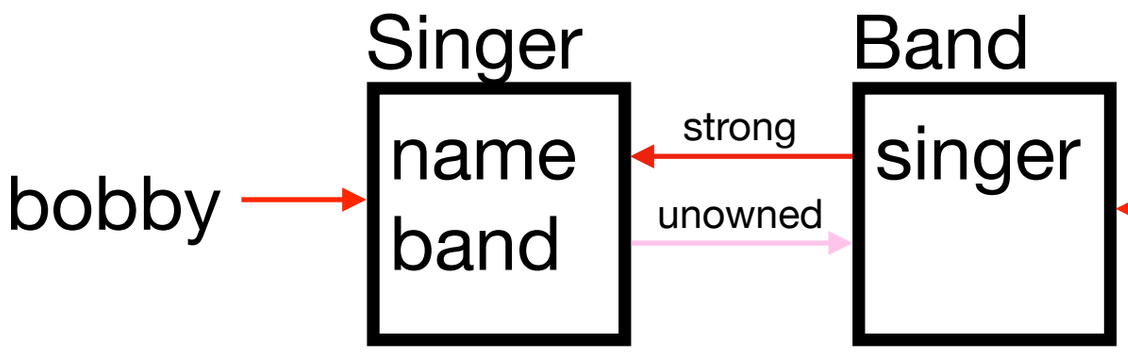
bobby

dragon

```
dragon = nil
// no more strong ref to
// Band, so deallocated,
// so no more strong ref to
// Singer, so deallocated
```

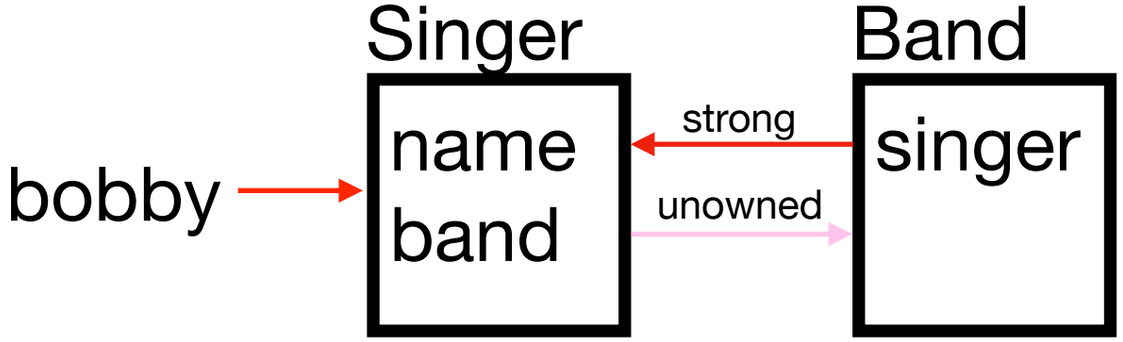
Unowned reference

what if we release the target first?



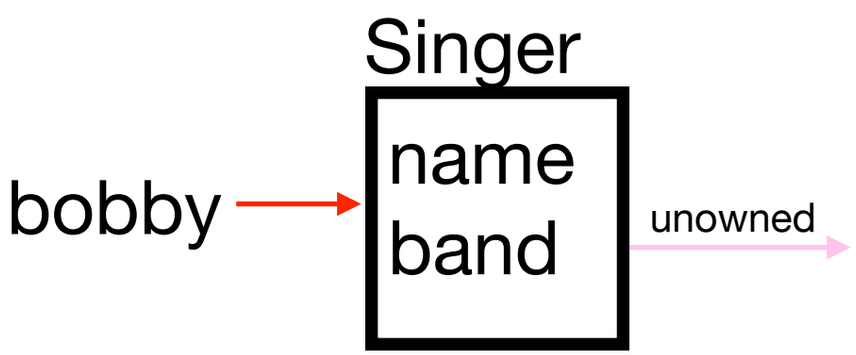
dragon

```
dragon = Band()
bobby = Singer(..., dragon)
dragon.singer = bobby
```



dragon

```
dragon = nil
// no more strong ref
// to Band instance,
// so Band is deallocated
```



dragon

```
// careful! bobby.band now
// has dangling pointer;
// dereferencing would throw
// run-time error.
// Cannot check for (or
// assign!) nil because
// unowned cannot be Optional
bobby = nil
```

References

```
// Examples based on Scott Gardner's article,
// "Conquering Capture Lists:
//
https://scotteg.github.io/Conquering-Capture-Lists

// Additional guidance from the official documentation
// "Automatic Reference Counting"
//
https://docs.swift.org/swift-book/LanguageGuide/
AutomaticReferenceCounting.html
```