

Swift Reference Types

reference cycles: strong, weak, and unowned

Strong reference

released when last owner set to nil or 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
  }

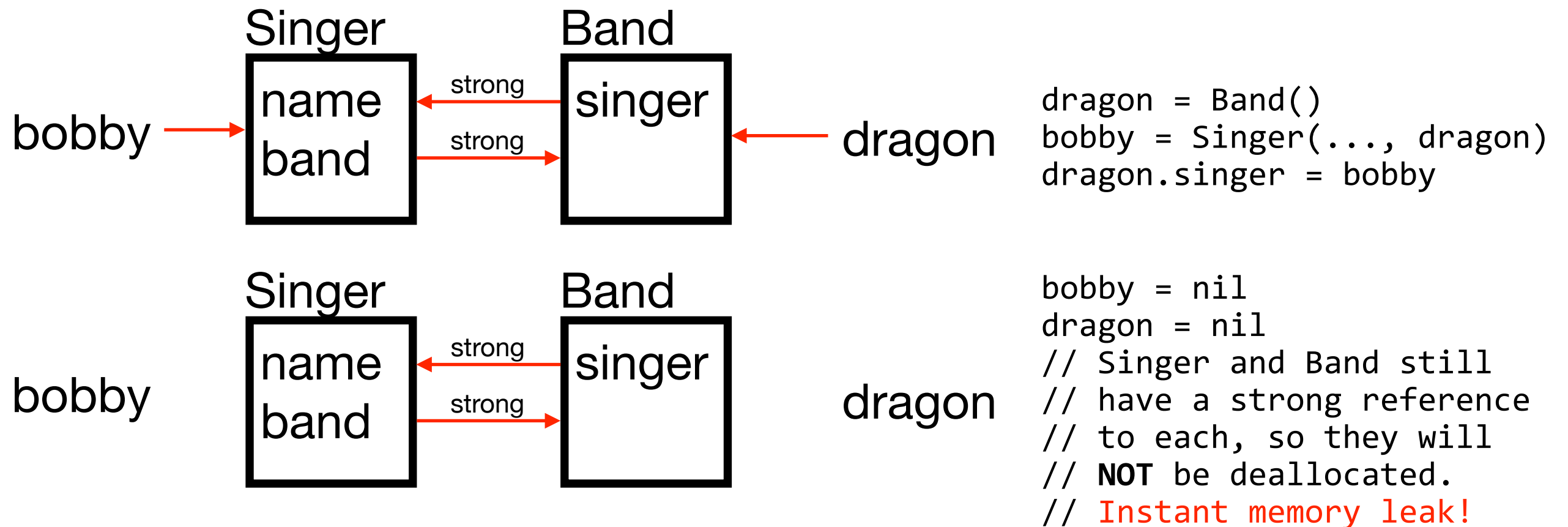
  deinit {
    print("\(Self.self)", #function)
  }
}
```

```
// 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 last owner set to nil or deallocated



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
  }

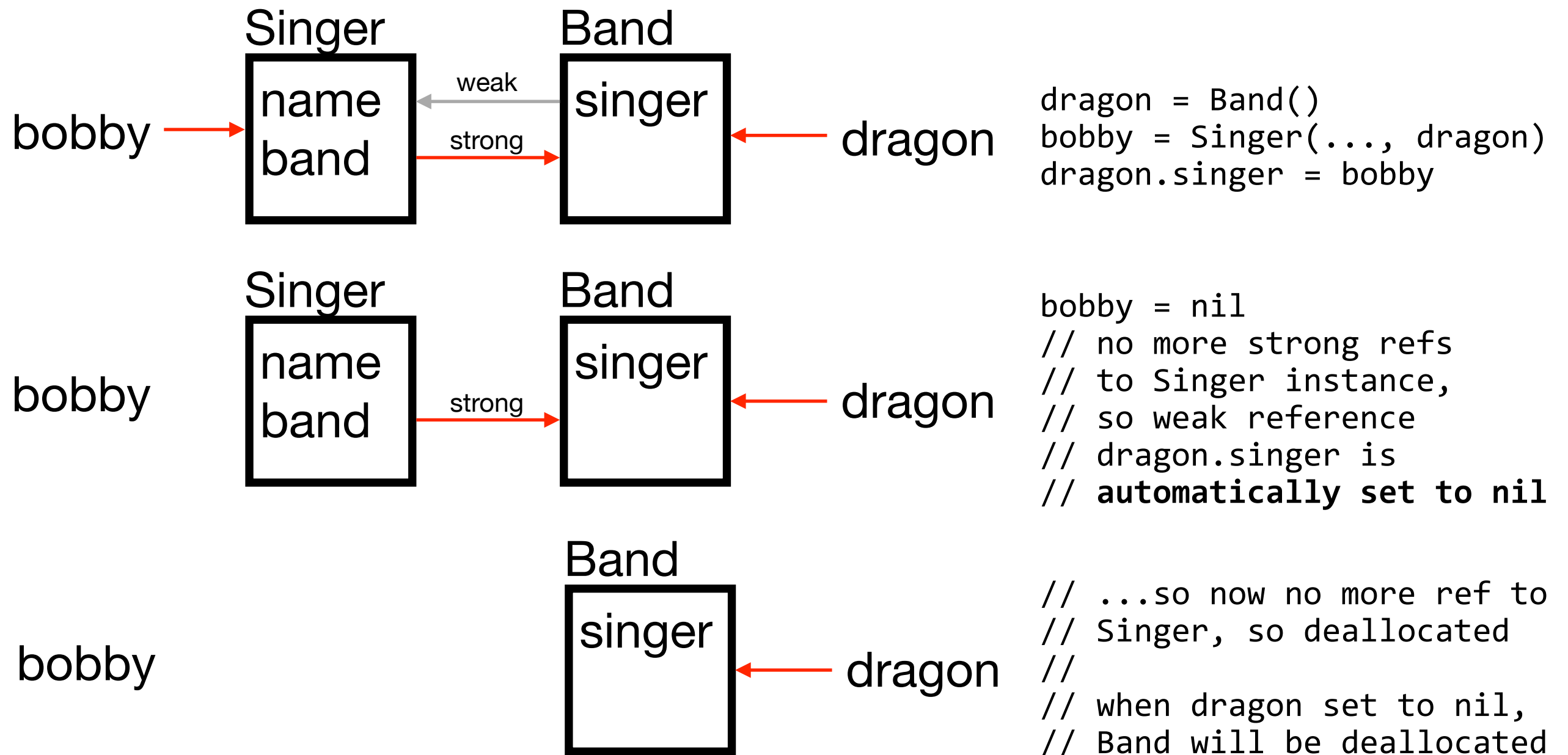
  deinit {
    print("\(Self.self)", #function)
  }
}
```

```
// 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
```

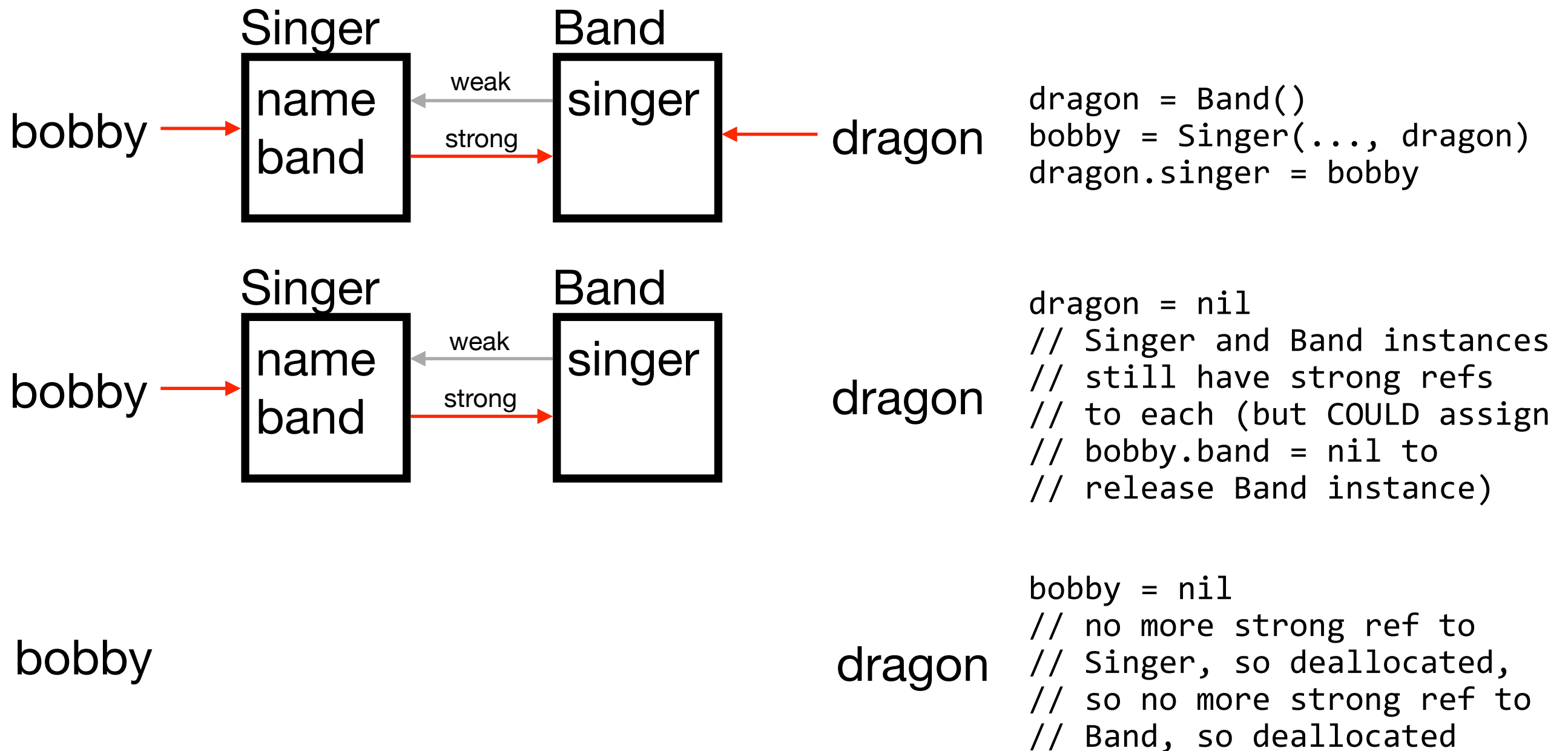
Weak reference

use when instance lifetime > target lifetime



Weak reference

what if we release the other one first?



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
    }

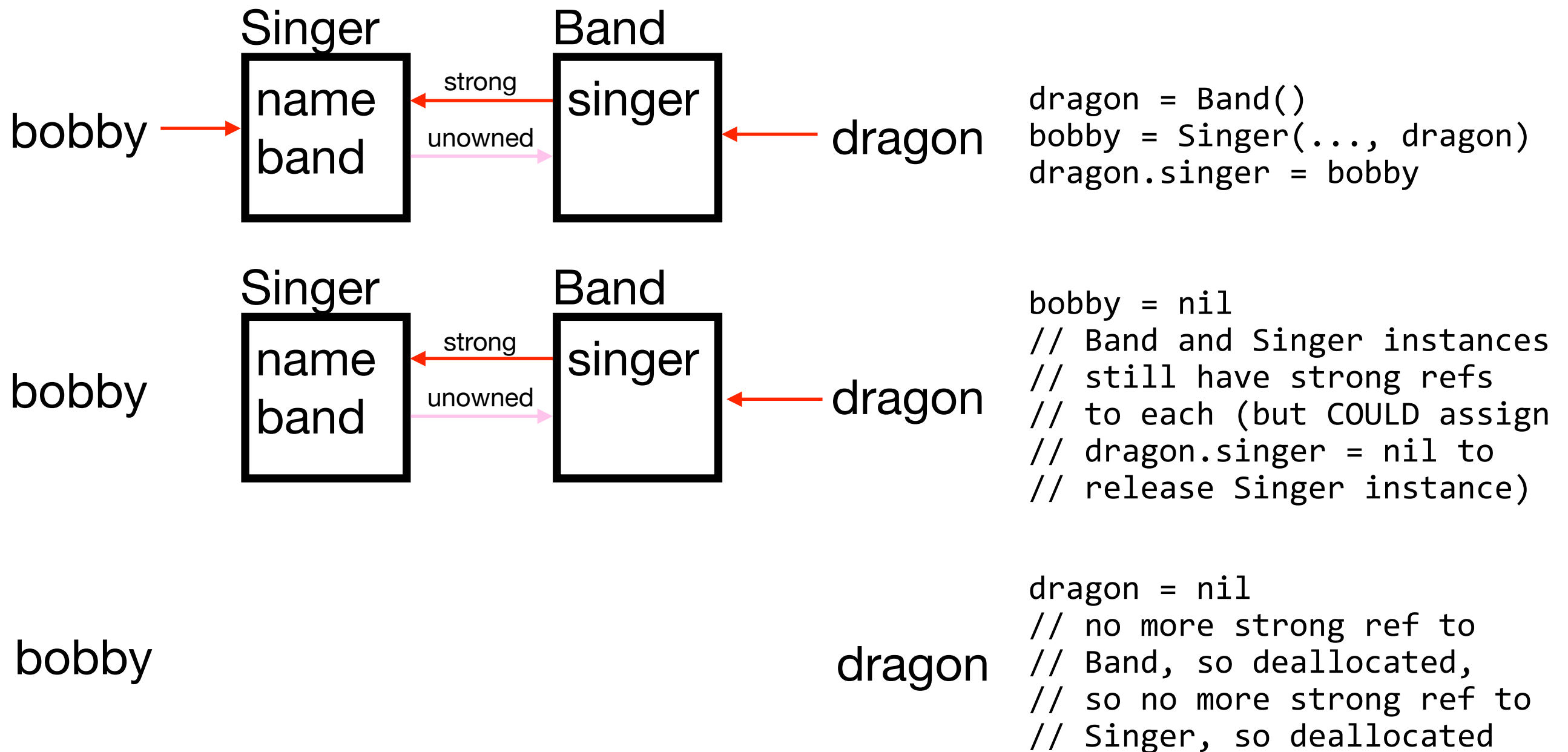
    deinit {
        print("\(Self.self)", #function)
    }
}
```

```
// 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
```

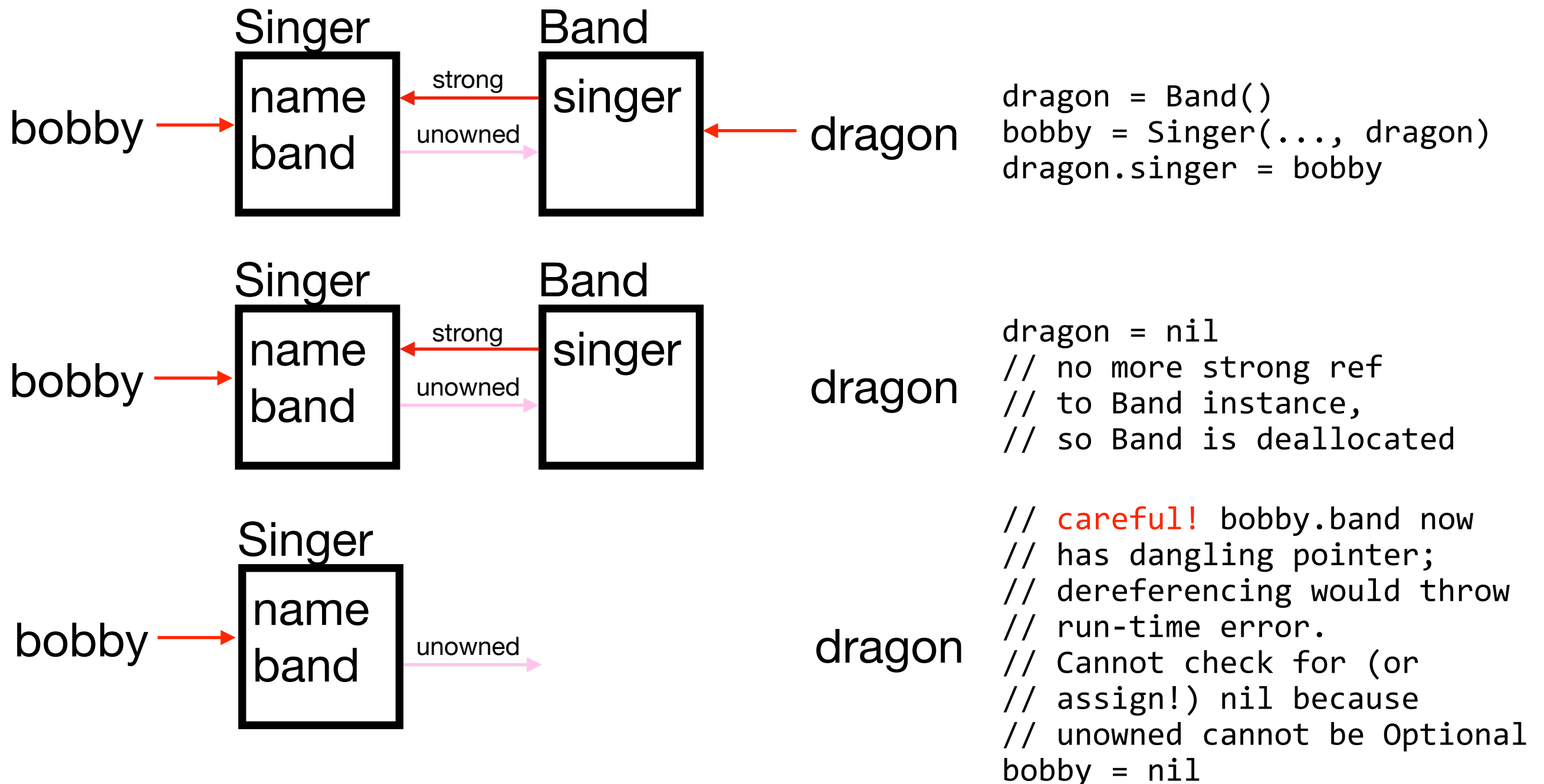
Unowned reference

use when instance lifetime \leq target lifetime



Unowned reference

what if we release the target first?



References

```
// Examples based on Scott Gardner's article,  
// "Conquering Capture Lists:  
//
```

```
https://scotteg.github.io/capture-lists
```

```
// Additional guidance from the official documentation  
// "Automatic Reference Counting"  
//
```

```
https://docs.swift.org/swift-book/LanguageGuide/  
AutomaticReferenceCounting.html
```