Swift Reference Types

reference cycles: strong, weak, and unowned

Glenn Cole, April 2020

Strong reference

releases claim when set to nil* or owner deallocated

```
class Band {
    var singer: Singer?

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

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

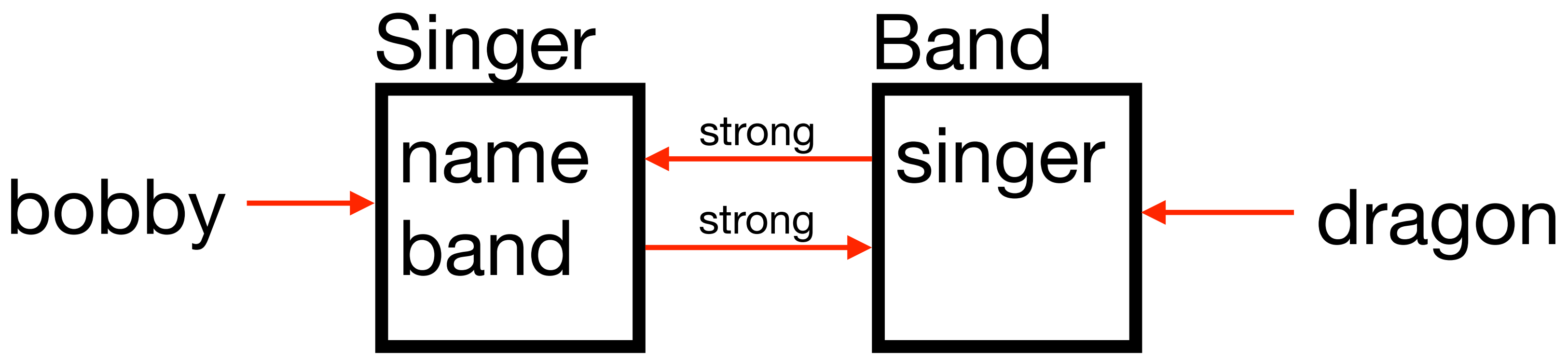
```
class Singer {
    let name: String
    let band: Band

    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 above are Optionals so we can
// assign nil, and force-unwrapped for
// ease of use
```

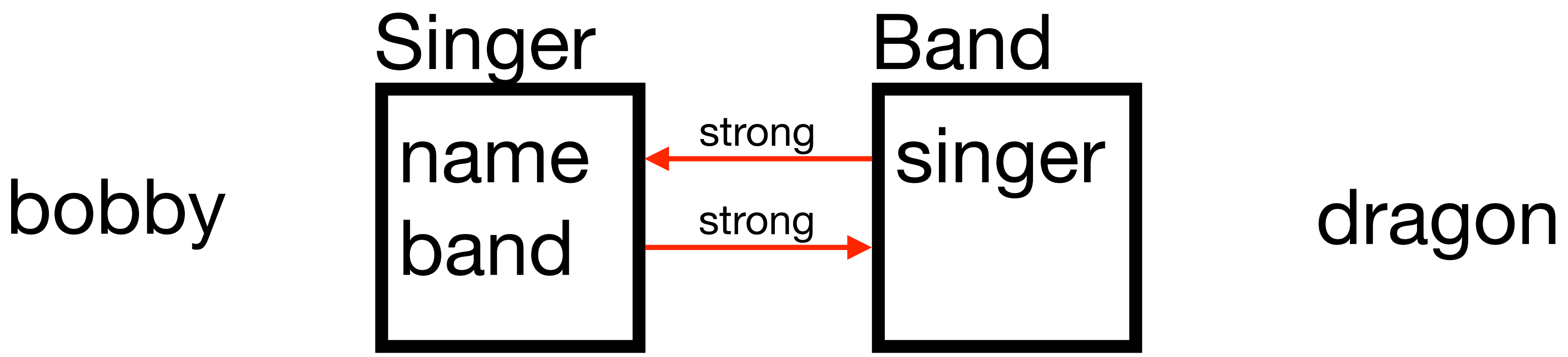
```
// * can be set to nil only if optional;
// may also point to a different object
//   to release claim to the first

// The target instance is deallocated
// where there is no strong reference
// that points to it.
```

Strong reference

target deallocated only when no strong reference



```
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

expects instance variable to outlive target

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

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

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

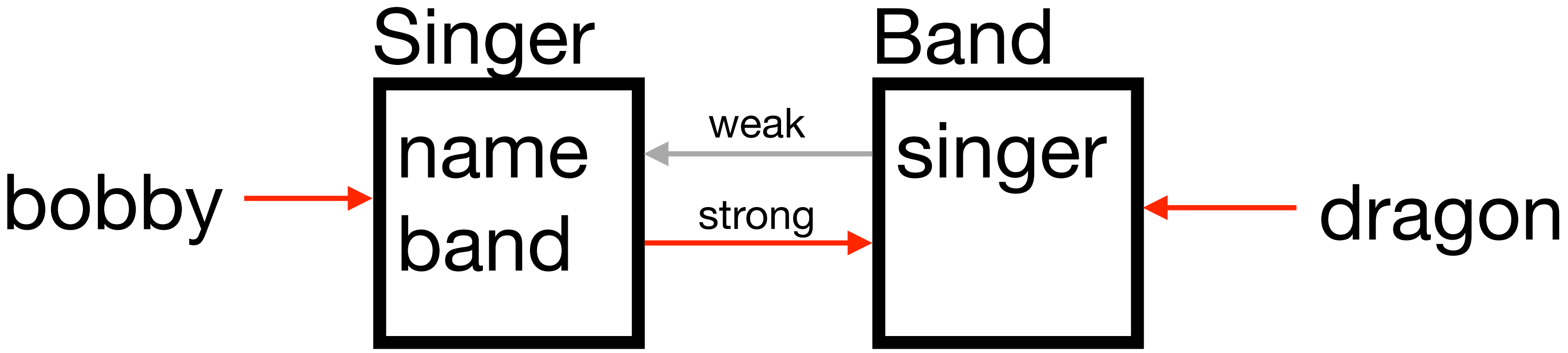
```
class Singer {
    let name: String
    let band: Band

    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 above are Optionals so we can
// assign nil, and force-unwrapped for
// ease of use
```

```
// weak vars MUST be Optional,
// because they are automatically
// assigned nil when their target
// has no more strong references
// and thus gets deallocated
```

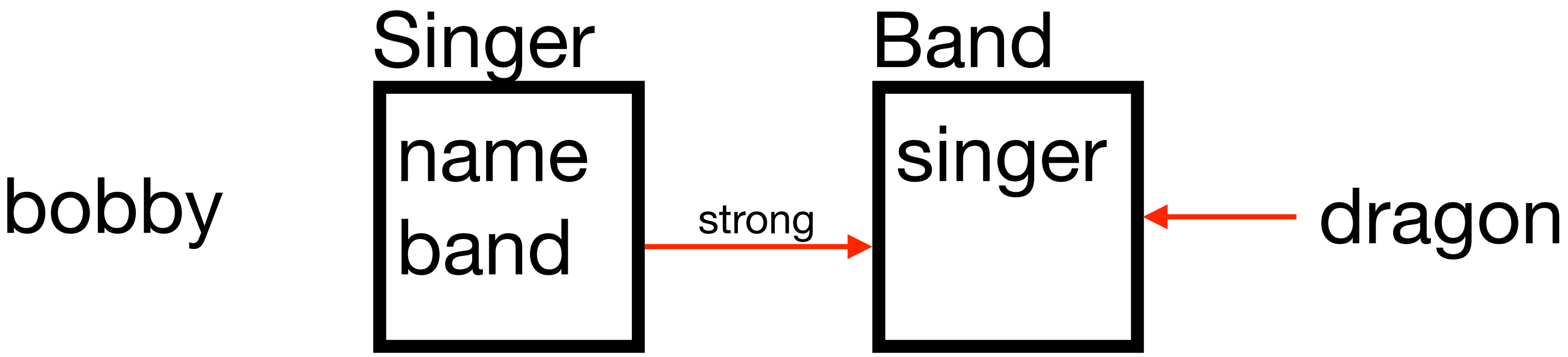
Weak reference

use when instance lifetime > target lifetime



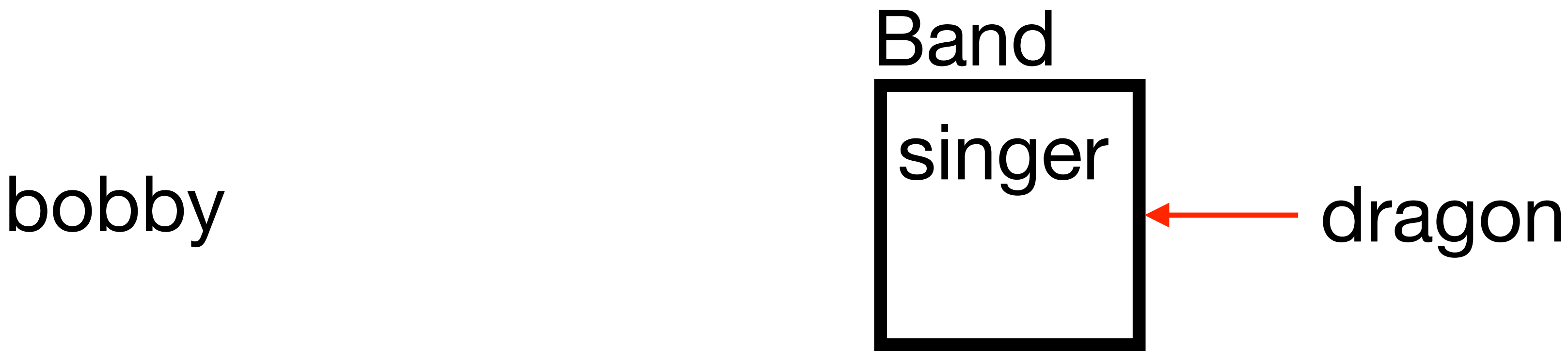
Singer
Band
bobby
name
band
weak
strong
singer
dragon

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



Singer
Band
bobby
name
band
strong
singer
dragon

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

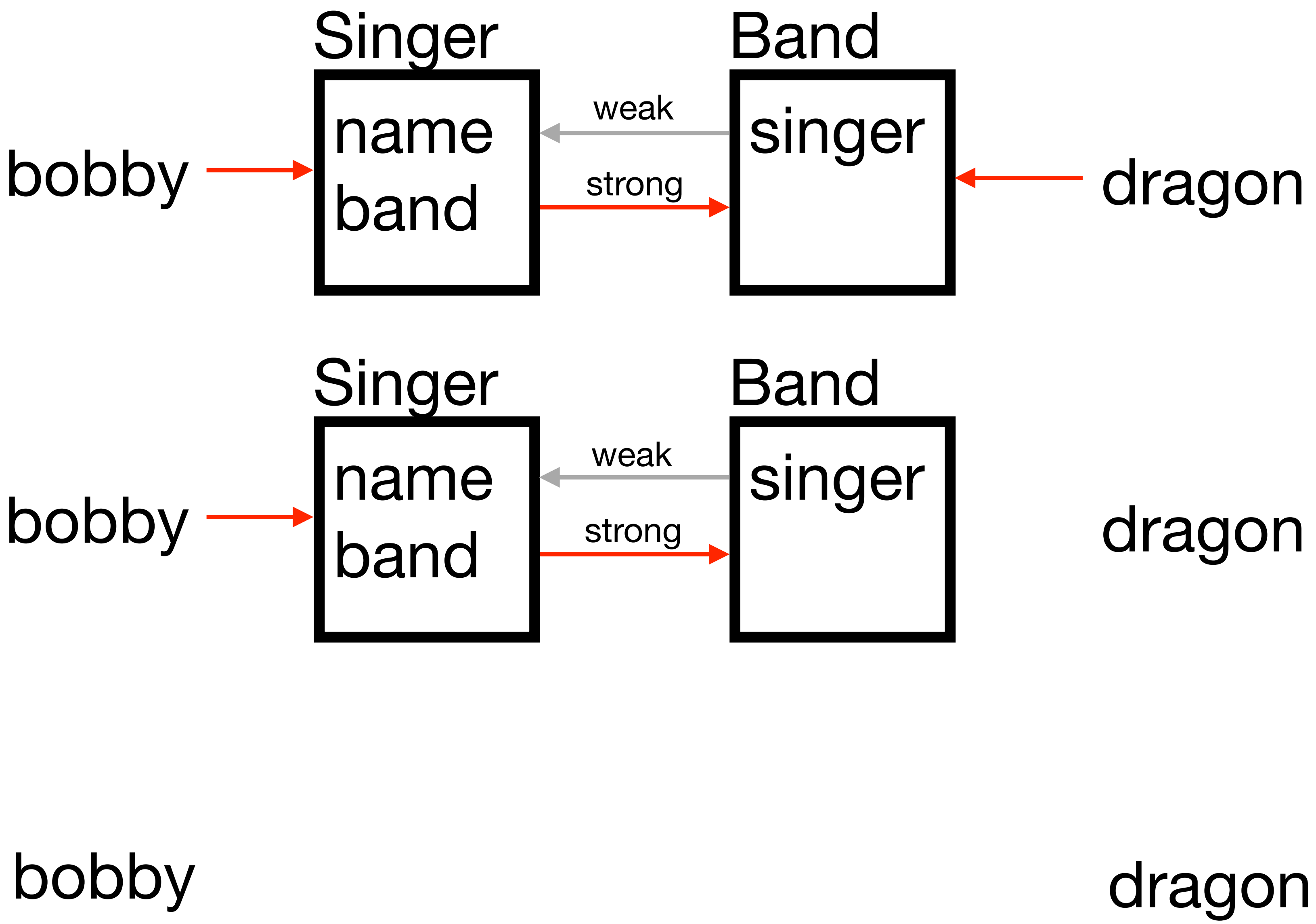


Band
bobby
singer
dragon

```
// ...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 = nil
// Singer and Band instances
// still have strong refs
// to each (If bobby.band was
// optional, could assign nil
// to release Band instance)
```

```
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

```
class Band {
    var singer: Singer?

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

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

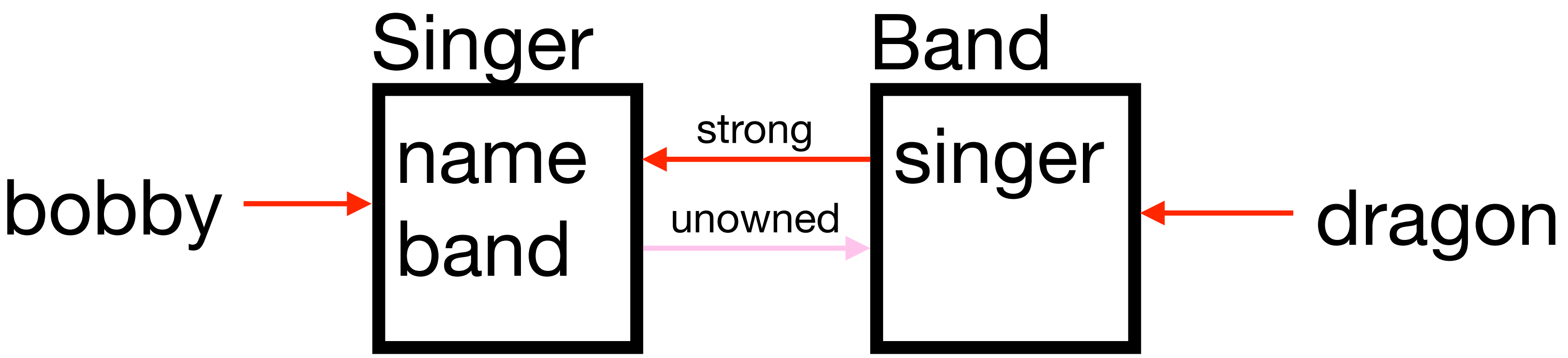
```
class Singer {
    let name: String
    unowned let band: Band

    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 above are Optionals so we can
// assign nil, and force-unwrapped for
// ease of use
```

```
// unowned vars CANNOT be Optional,
// because it is assumed their target
// will outlive their instance
```

Unowned reference

use when instance lifetime ≤ target lifetime



bobby

Singer

name
band

strong

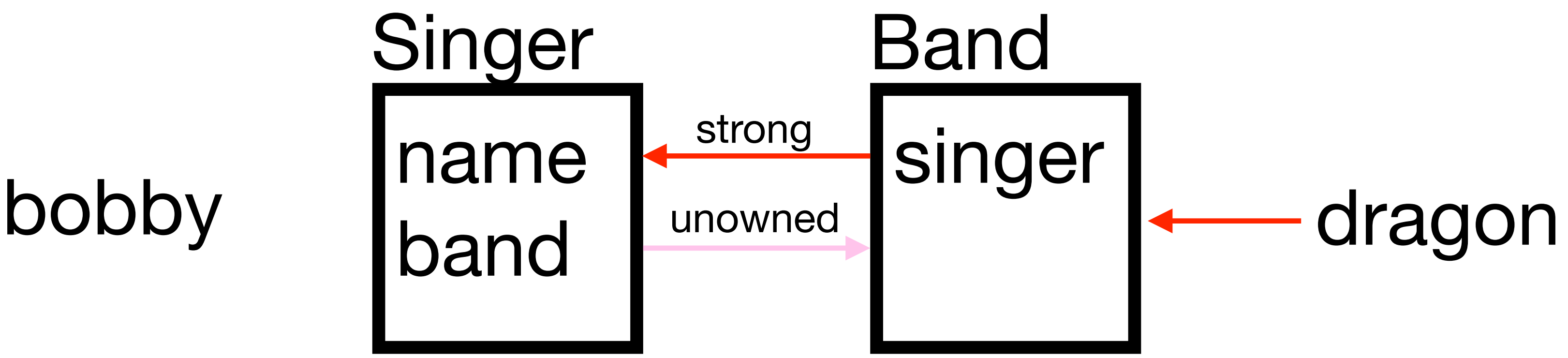
unowned

Band

singer

dragon

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



bobby

Singer

name
band

strong

unowned

Band

singer

dragon

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

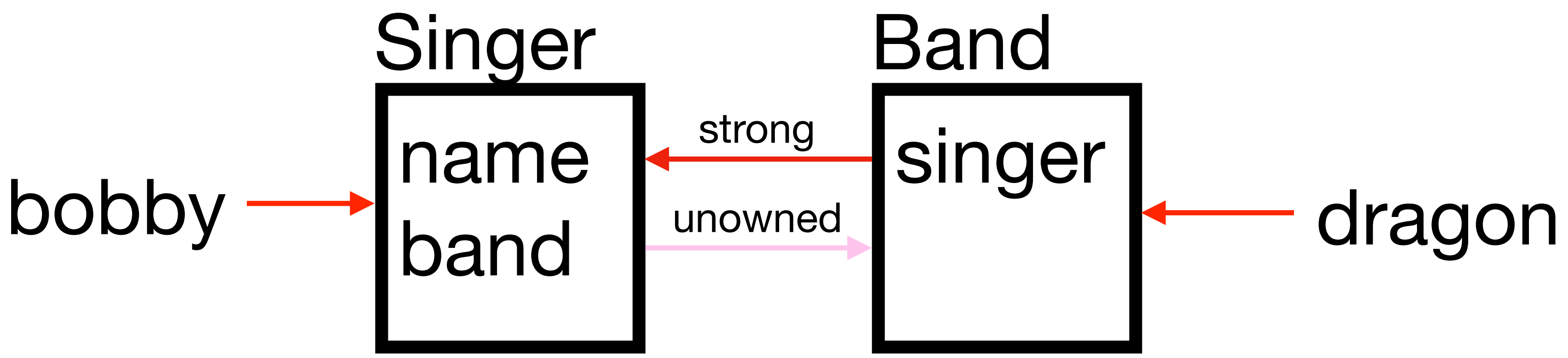
bobby

dragon

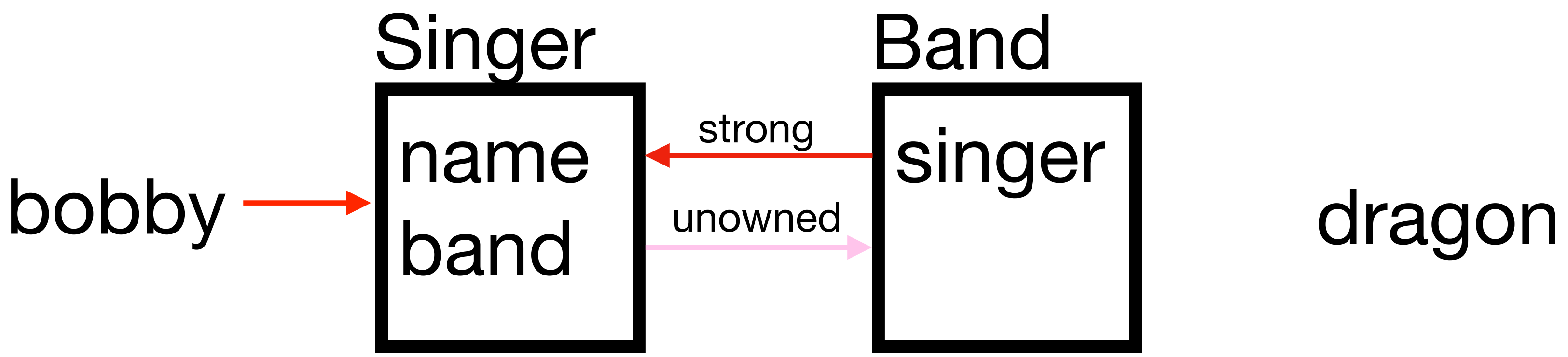
```
dragon = nil
// no more strong ref to
// Band, so deallocated,
// so no more strong ref to
// Singer, so deallocated
```

Unowned reference

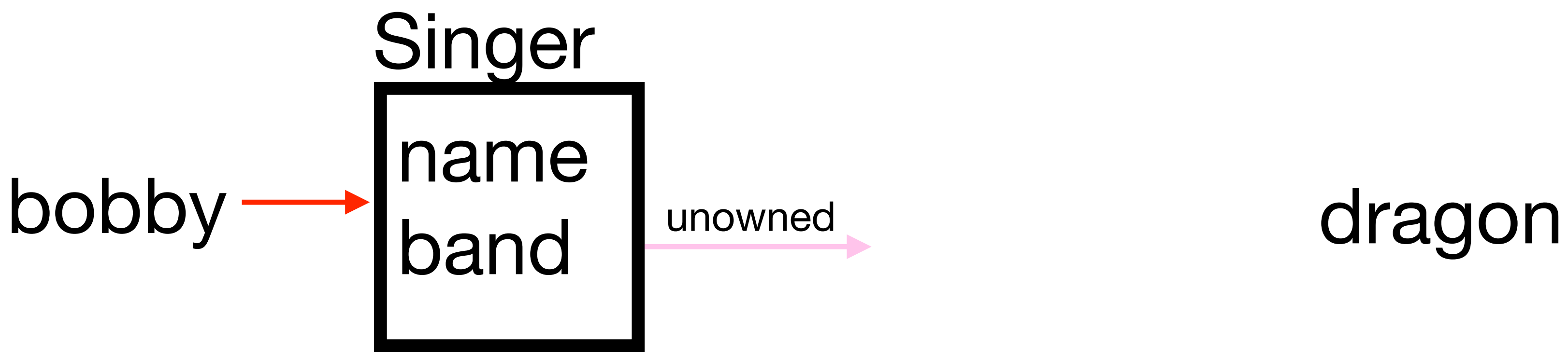
what if we release the unowned target first?



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



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



```
// 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/capture-lists

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