Aggregate Data Types

Michael VanSickle

@vansimke

Introduction

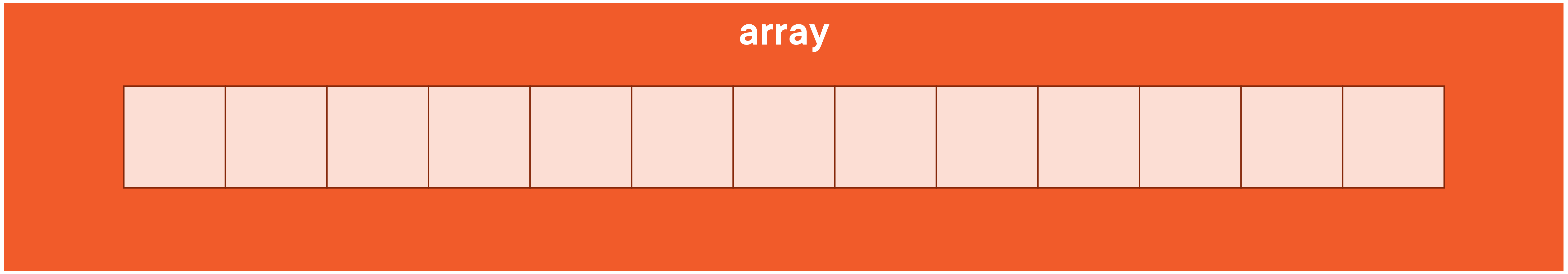
Arrays

Slices

Maps

Structs

Array



Array

array

value	0	0	0	0	0	0	0	0	0	0	0	0	0
index	0	1	2	3	4	5	6	7	8	9	10	11	12

Arrays in Go

```go
var arr [3]int              // array of 3 ints
fmt.Println(arr)            // [0 0 0]
arr = [3]int{1, 2, 3}       // array literal

fmt.Println(arr[1])         // 2
arr[1] = 99                 // update value
fmt.Println(arr)            // [1 99 3]

fmt.Println(len(arr))       // 3
```

Arrays in Go

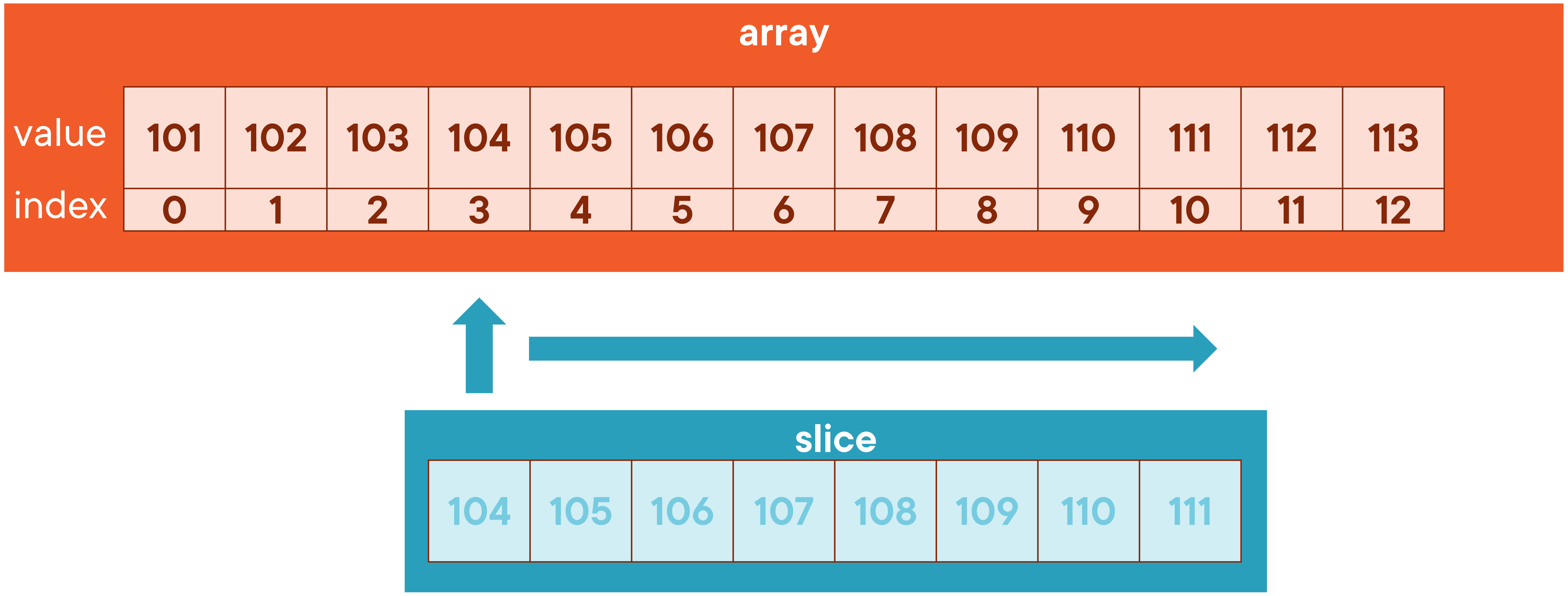
```
arr := [3]string{“foo”, “bar”, “baz”}

arr2 := arr                                   // arrays are copied by value
fmt.Println(arr2)                             // {“foo” “bar” “baz”}

arr[0] = “quux”
fmt.Println(arr)                              // {“quux” “bar” “baz”}
fmt.Println(arr2)                             // {“foo” “bar” “baz”}

arr == arr2                                   // false – arrays are comparable
```

Demo

arrays

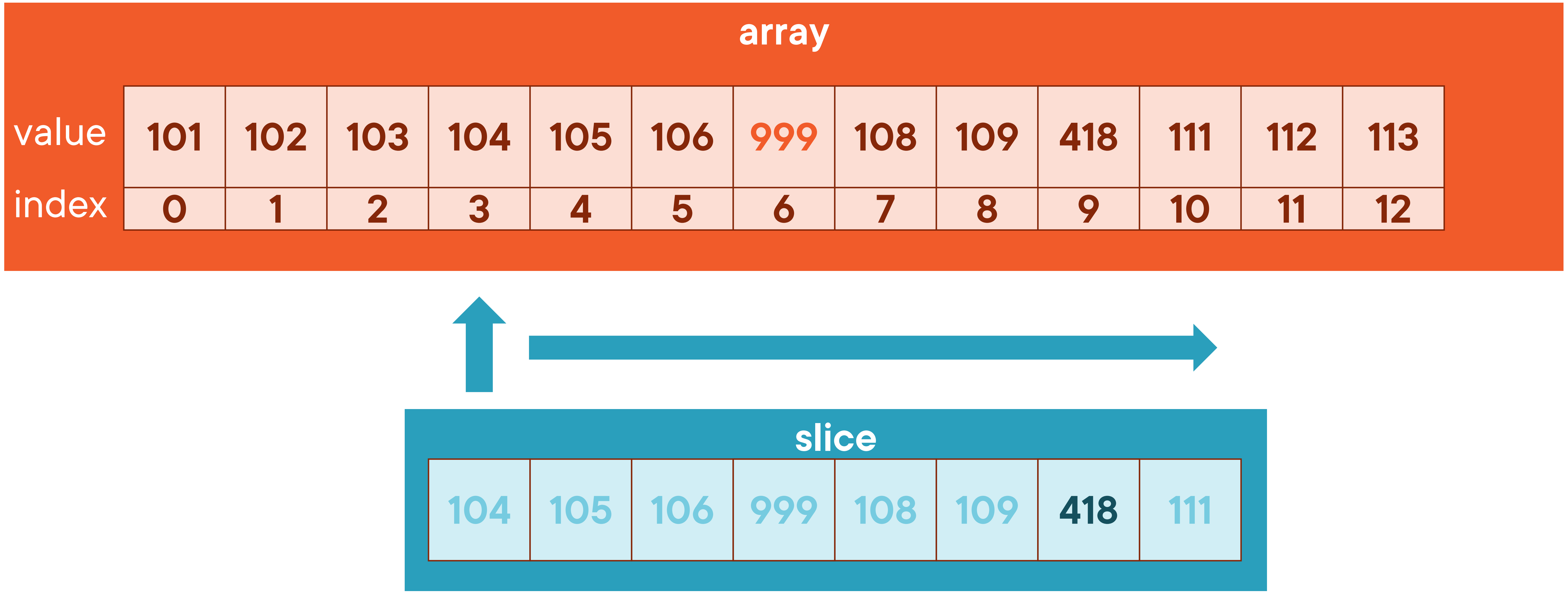
- declare and intialize

Slices



array
value
index
101
102
103
104
105
106
107
108
109
110
111
112
113
0
1
2
3
4
5
6
7
8
9
10
11
12
slice
104
105
106
107
108
109
110
111

Slices


array
value
index
101
102
103
104
105
106
999
108
109
418
111
112
113
0
1
2
3
4
5
6
7
8
9
10
11
12
slice
104
105
106
999
108
109
418
111

Slices in Go

```
var s []int                 // slices of ints
fmt.Println(s)              // [] (nil)
s = []int{1, 2, 3}          // slice literal

fmt.Println(s[1])           // 2
s[1] = 99                   // update value
fmt.Println(s)              // [1 99 3]

s = append(s, 5, 10, 15)    // add elements to the slice
fmt.Println(s)              // [1 99 3 5 10 15]

s = slices.Delete(s, 1, 3) // remove indices 1, 2 from slice (golang.org/x/exp/slices)
fmt.Println(s)              // [1 5 10 15]
```

Slices in Go

```
s := []string{“foo”, “bar”, “baz”}
s2 := s                                // slices are copied by reference
                                       // use slices.Clone to clone

s[0], s2[2] = “qux”, “fred”            // update values in slices
fmt.Println(s, s2)                     // [“qux” “bar” “fred”] [“qux” “bar” “fred”]
                                       // data is shared


s == s2                                // compile time error – slices are not comparable
```

Demo

slices

- don't show slicing ops

Maps

array

value	0	0	0	0	0	0	0	0	0	0	0	0	0
index	0	1	2	3	4	5	6	7	8	9	10	11	12

Maps

array

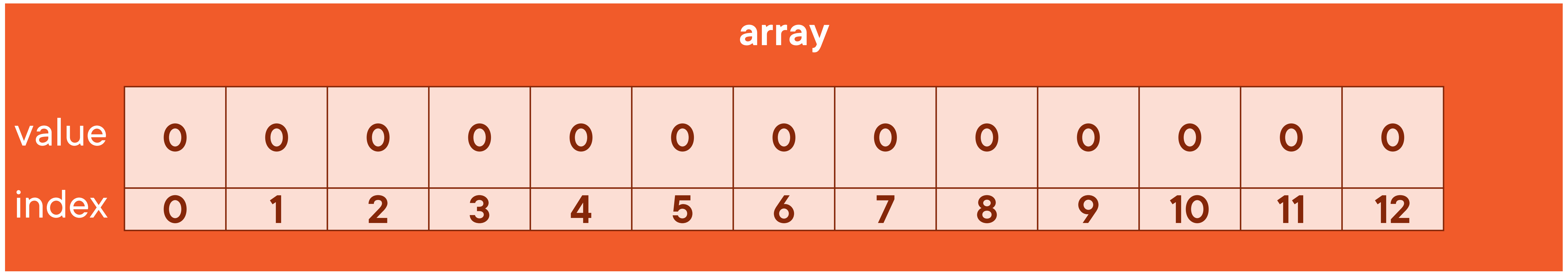
value	0	0	0	0	0	0	0	0	0	0	0	0	0
index	0	1	2	3	4	5	6	7	8	9	10	11	12

map

value {string}

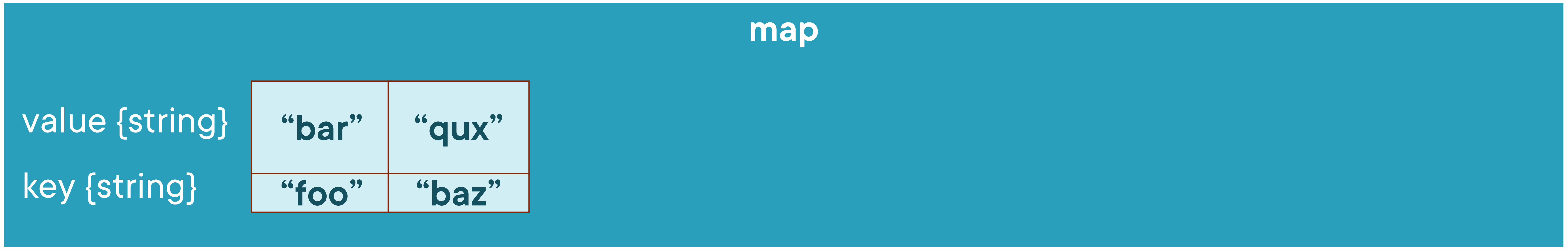
key {string}

Maps



array

value	0	0	0	0	0	0	0	0	0	0	0	0	0
index	0	1	2	3	4	5	6	7	8	9	10	11	12



map

value {string}	“bar”	“qux”
key {string}	“foo”	“baz”

Map

```
var m map[string]int                                // declare a map
fmt.Println(m)                                      // map[] (nil)
m = map[string]int{“foo”: 1, “bar”: 2}              // map literal
fmt.Println(m)                                      // map[foo:1 bar:2]

fmt.Println(m[“foo”])                               // lookup value in map
m[“bar”] = 99                                       // update value in map

delete(m, “foo”)                                    // remove entry from map
m[“baz”] = 418                                      // add value to map

fmt.Println(m)                                      // map[bar:99 baz: 418]

fmt.Println(m[“foo”])                               // 0 - queries always return results
v, ok := m[“foo”]                                   // comma okay syntax verifies presents
fmt.Println(v, ok)                                  // 0, false
```

https://pkg.go.dev/golang.org/x/exp/maps

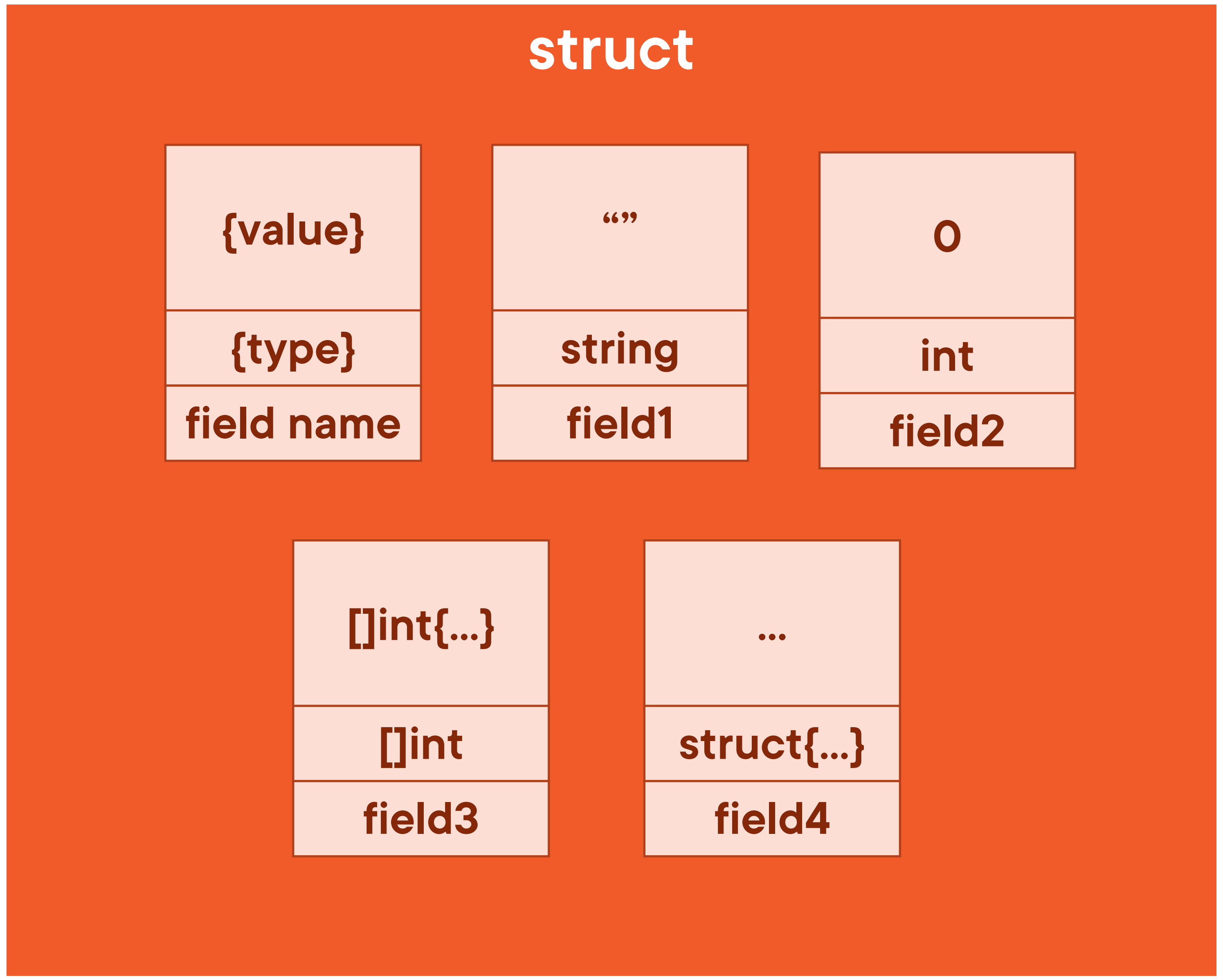
Map

```
m := map[string]int{
     “foo”:1,
     “bar”:2,
     “baz”:3}
m2 := m                             // maps are copied by reference
                                    // use maps.Clone to clone

m[“foo”], m2[“bar”] = 99, 42        // update values in maps
fmt.Println(m)                      // map[foo:99 bar:42 baz:3]
fmt.Println(m2)                     // map[foo:99 bar:42 baz:3]
                                    // data is shared


m == m2                             // compile time error – maps are not comparable
```

Demo

maps

Structs


struct
{value}
{type}
field name
“”
string
field1
0
int
field2
[]int{...}
[]int
field3
...
struct{...}
field4

Structs in Go

```
var s struct{                        // declare an anonymous struct
    name     string
    id       int
}
fmt.Println(s)                       // {“” 0}

s.name = “Arthur”                    // assign value to field
fmt.Println(s.name)                  // query value of field
```

Structs in Go

```
type myStruct struct {                  // create custom type based on struct
     name     string
     id       int
}

var s myStruct                          // declare variable with custom type
fmt.Println(s)                          // {“” 0}

s = myStruct{                           // struct literal
     name: “Arthur”,
     id: 42}
fmt.Println(s)                          // {“Arthur” 42}
```

Structs in Go

```
type myStruct struct {
     name      string
     id        int
}
var s myStruct
s = myStruct{
     name: “Arthur”,
     id: 42}
s2 := s
s.name = “Tricia”                  // structs are copied by value
fmt.Println(s, s2)                 // {“Tricia” 42} {“Arthur” 42}

s == s2                            // false – structs are comparable
```

Demo

structs

Summary

Arrays

Slices

Maps

Structs