Building a go tool to modify struct tags

Fatih Arslan
Software Engineer @DigitalOcean

Why do we need **tooling?**

```
type Example struct {
    Foo string
}
```

```
type Example struct {
    Foo string `json:"foo"`
}
```

What about structs with
many fields?

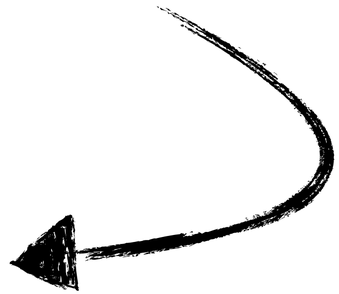
```
type Example struct {
    StatusID int64
    Foo      string
    Bar      bool

    Server struct {
        Address string
        TLS     bool
    }

    DiskSize int64
    Volumes  []string
}
```

```
type Example struct {
    StatusID int64  `json:"status_id"`
    Foo      string `json:"foo" xml:"foo"`
    Bar      bool   `json:"bar" xml:"bar"`

    Server struct {
        Address string `json:"address"`
        TLS     bool   `json:"tls",xml:"tls"`
    } `json="server"`

    DiskSize int64    `json:disk_size`
    Volumes  []string `"json":"volumes"`
}
```

Working with struct tags
is **not** fun 😝

So, what's a **struct tag**?

```
type Example struct {
  Foo string `json:"foo"`
}
```



There is **no official spec** of the **struct tag**

Struct tag definition

But fortunately, we have an **inofficial** one in **stdlib**!

Spec is defined in the

reflect package

reflect.StructTag

type **StructTag**

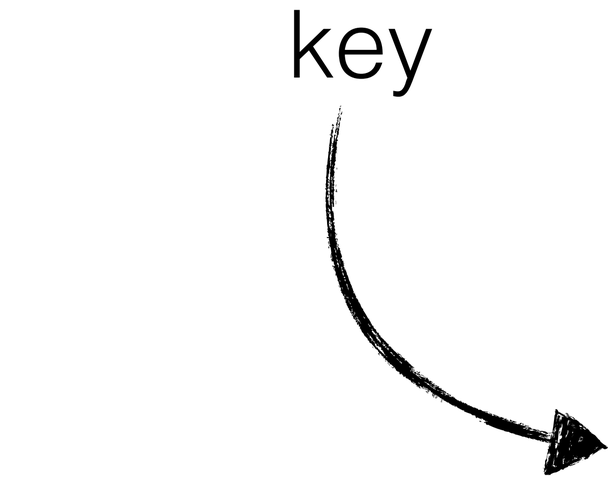
A StructTag is the tag string in a struct field.

By convention, tag strings are a concatenation of optionally space-separated key:"value" pairs. Each key is a non-empty string consisting of non-control characters other than space (U+0020 ' '), quote (U+0022 '"'), and colon (U+003A ':'). Each value is quoted using U+0022 '"' characters and Go string literal syntax.

```
type StructTag string
```

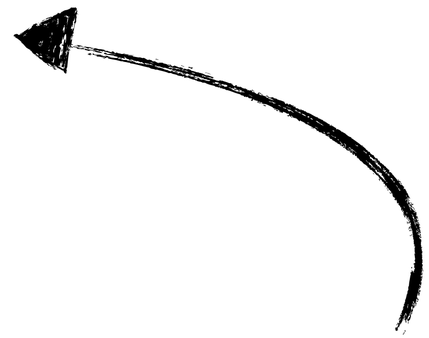
Let's **decompose** the spec

json:"foo"



json:"foo"

json:"**foo**"



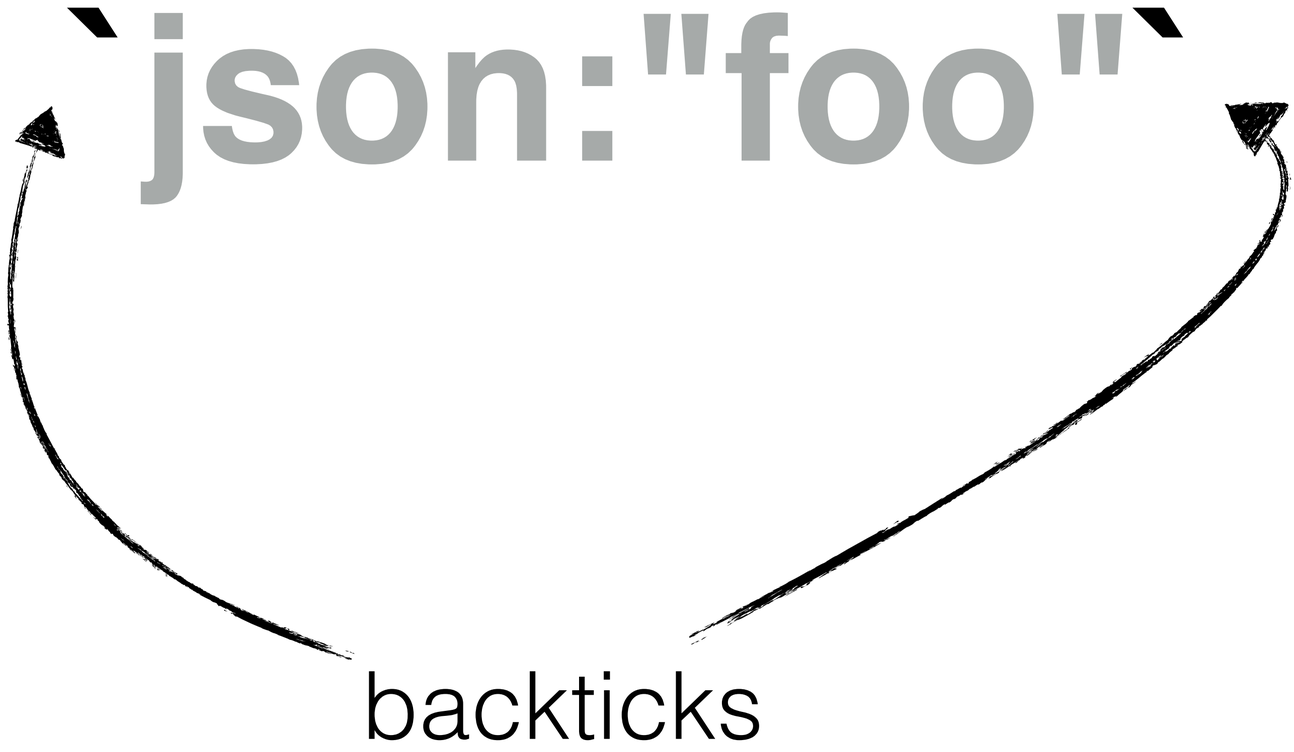
value
(quoted)

json:"foo"

colon
(not specified clearly)

space separation

json:"foo" xml:"foo"


`json:"foo"`
backticks

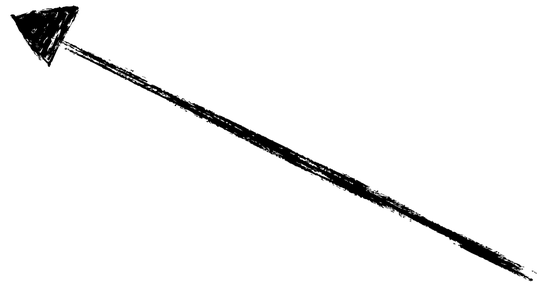
```
type Example struct {
    Foo string `json:"foo"`
}
```

```
type Example struct {
    Foo string "json:\"foo\""
}
```

quotes instead of backticks

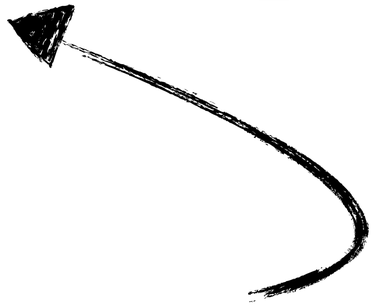
(works, but not fun to deal with)

json:"**foo,omitempty**"



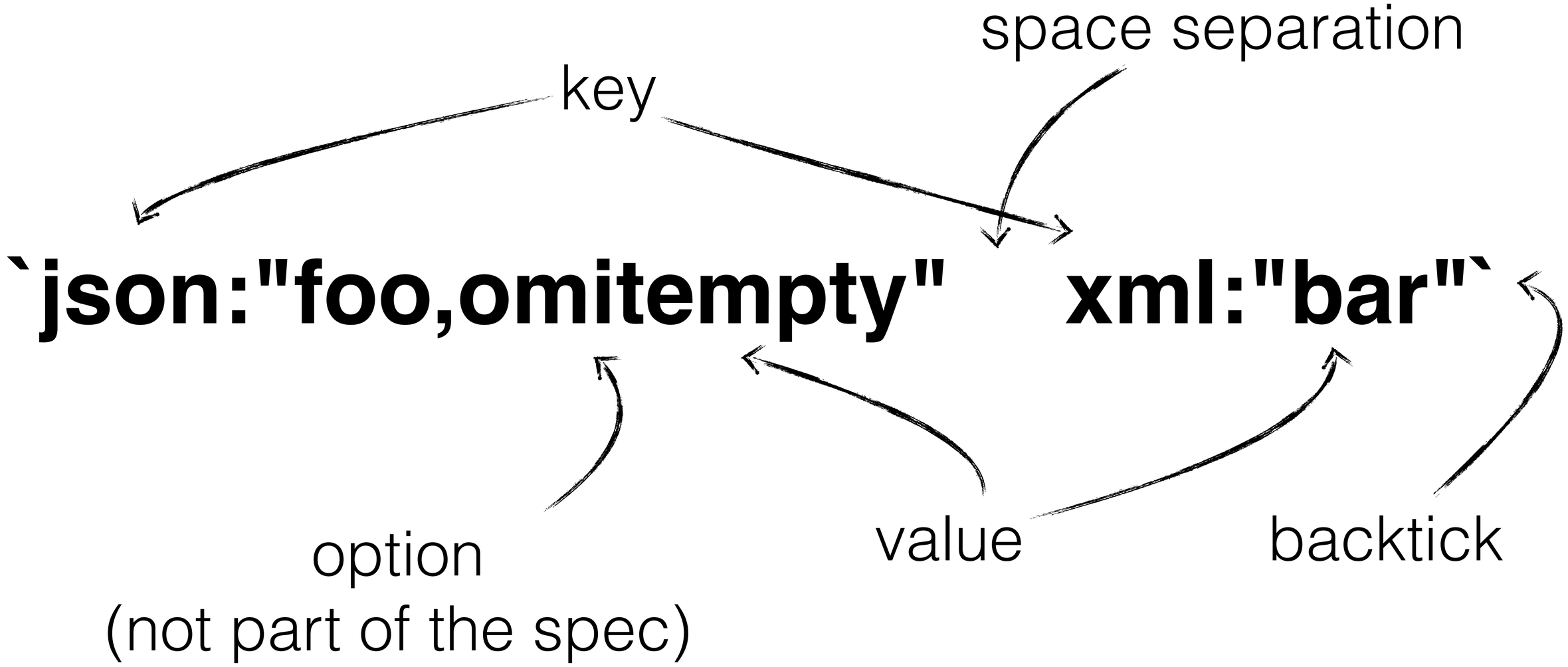
value

json:"foo,**omitempty**"



option
(not part of the spec)

Recap


space separation
key
`json:"foo,omitempty" xml:"bar"`
option
(not part of the spec)
value
backtick

Back to our **initial slide**!

```
type Example struct {
    StatusID int64  `json:"status_id"`
    Foo      string `json:"foo" xml:"foo"`
    Bar      bool   `json:"bar" xml:"bar"`

    Server struct {
        Address string `json:"address"`
        TLS     bool   `json:"tls",xml:"tls"`
    } `json="server"`

    DiskSize int64    `json:disk_size`
    Volumes  []string `"json":"volumes"`
}
```

It has some **errors** 

```
type Example struct {
    StatusID int64  `json:"status_id"`
    Foo      string `json:"foo" xml:"foo"`
    Bar      bool   `json:"bar" xml:"bar"`

    Server struct {
        Address string `json:"address"`
        TLS     bool   `json:"tls",xml:"tls"`
    } `json="server"`

    DiskSize int64    `json:disk_size`
    Volumes  []string `"json":"volumes"`
}
```

```
type Example struct {
    StatusID int64  `json:"status_id"`
    Foo      string `json:"foo" xml:"foo"`
    Bar      bool   `json:"bar" xml:"bar"`

    Server struct {
        Address string `json:"address"`
        TLS     bool   `json:"tls",xml:"tls"`
    } `json="server"`

    DiskSize int64    `json:disk_size`
    Volumes  []string `"json":"volumes"`
}
```

`json:"tls" xml:"tls"`

```
type Example struct {
    StatusID int64  `json:"status_id"`
    Foo      string `json:"foo" xml:"foo"`
    Bar      bool   `json:"bar" xml:"bar"`

    Server struct {
        Address string `json:"address"`
        TLS     bool   `json:"tls",xml:"tls"`
    } `json="server"`

    DiskSize int64    `json:disk_size`
    Volumes  []string `"json":"volumes"`
}
```

```
type Example struct {
    StatusID int64  `json:"status_id"`
    Foo      string `json:"foo" xml:"foo"`
    Bar      bool   `json:"bar" xml:"bar"`

    Server struct {
        Address string `json:"address"`
        TLS     bool   `json:"tls",xml:"tls"`
    } `json="server"`
      `json:"server"`
    DiskSize int64    `json:disk_size`
    Volumes  []string `"json":"volumes"`
}
```

```
type Example struct {
    StatusID int64  `json:"status_id"`
    Foo      string `json:"foo" xml:"foo"`
    Bar      bool   `json:"bar" xml:"bar"`

    Server struct {
        Address string `json:"address"`
        TLS     bool   `json:"tls",xml:"tls"`
    } `json="server"`

    DiskSize int64    `json:disk_size`
    Volumes  []string `"json":"volumes"`
}
```

```
type Example struct {
    StatusID int64  `json:"status_id"`
    Foo      string `json:"foo" xml:"foo"`
    Bar      bool   `json:"bar" xml:"bar"`

    Server struct {
        Address string `json:"address"`
        TLS     bool   `json:"tls",xml:"tls"`
    } `json="server"`

    DiskSize int64    ~~`json:disk_size`~~
    Volumes  []string `json:"disk_size"`
}
```

```
type Example struct {
    StatusID int64  `json:"status_id"`
    Foo      string `json:"foo" xml:"foo"`
    Bar      bool   `json:"bar" xml:"bar"`

    Server struct {
        Address string `json:"address"`
        TLS     bool   `json:"tls",xml:"tls"`
    } `json="server"`

    DiskSize int64    `json:disk_size`
    Volumes  []string `"json":"volumes"`
}
```

```
type Example struct {
    StatusID int64  `json:"status_id"`
    Foo      string `json:"foo" xml:"foo"`
    Bar      bool   `json:"bar" xml:"bar"`

    Server struct {
        Address string `json:"address"`
        TLS     bool   `json:"tls",xml:"tls"`
    } `json="server"`

    DiskSize int64    `json:disk_size`
    Volumes  []string ~~`"json":"volumes"`~~
}
```

`json:"volumes"`

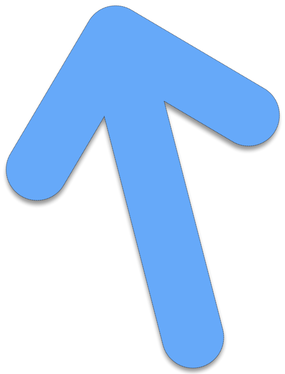
First attempt

(to automate it)

Old implementation

- From the cursor search backwards for the first 'struct {' literal
- Once found, do a forward search for the right hand brace }
- Get the line numbers between the two braces
- For each line, get the first identifier, convert it to camel_case word and then append to the same line
- A combination of Vim's search() function and regex is being used

```
type Example struct {
    Foo string
    Bar |bool
}
```



cursor

```
type Example struct {
    Foo string
    Bar bool
}
```

```
type Example struct {
    Foo string
    Bar bool
}
```

```
type Example struct {
	Foo string
	Bar bool
}
```

```
type Example struct {
        Foo string
        Bar bool
}
```

```
type Example struct {
        Foo string `json:"foo"`
        Bar bool `json:"bar"`
}
```

Problems

No Formatting

have

```
type Example struct {
  Bar string `json:"bar"`
  Foo bool `json:"foo"`
}
```

want

```
type Example struct {
  Bar string `json:"bar"`
  Foo bool   `json:"foo"`
}
```

In-line comments

have

```
type Example struct {
   Bar string // comment for bar `json:"bar"`
   Foo bool `json:"foo"`
}
```

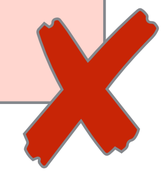
want

```
type Example struct {
   Bar string `json:"bar"` // comment for bar
   Foo bool   `json:"foo"`
}
```

Nested structs

have

```
type Example struct {
   Bar    bool `json:"bar"`
   Server struct { `json:"server"`
      Address string `json:"address"`
   }
}
```



want

```
type Example struct {
   Bar    bool `json:"bar"`
   Server struct {
      Address string `json:"address"`
   } `json:"server"`
}
```

Duplicate tags

have

```
type Example struct {
   Bar string `json:"bar"` `xml:"bar"`
   Foo bool `xml:"foo"`
}
```

want

```
type Example struct {
   Bar string `json:"bar" xml:"bar"`
   Foo bool   `xml:"foo"`
}
```

more quirks

- not able to remove tags
- not able to add or remove options
- field with interface{} types don't work
- comment with braces ({ and }) don't work
- ...

How do we **fix** it?

Second attempt

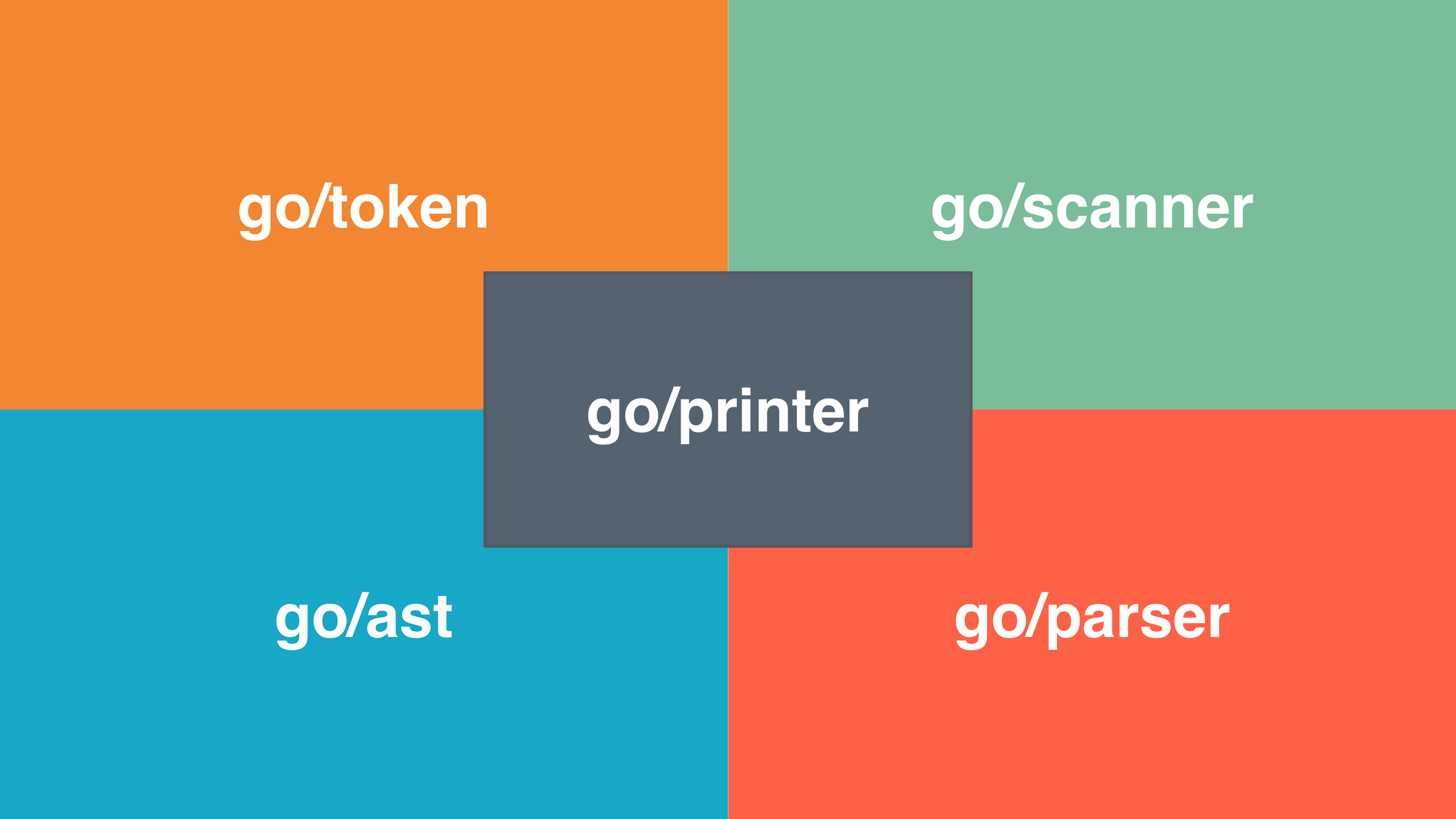
(and also the final one 🙃)

Two issues that need to be fixed

- ✗ Parse struct
- ✗ Parse struct tag

1. Parse Struct

The Go **Parser** Family

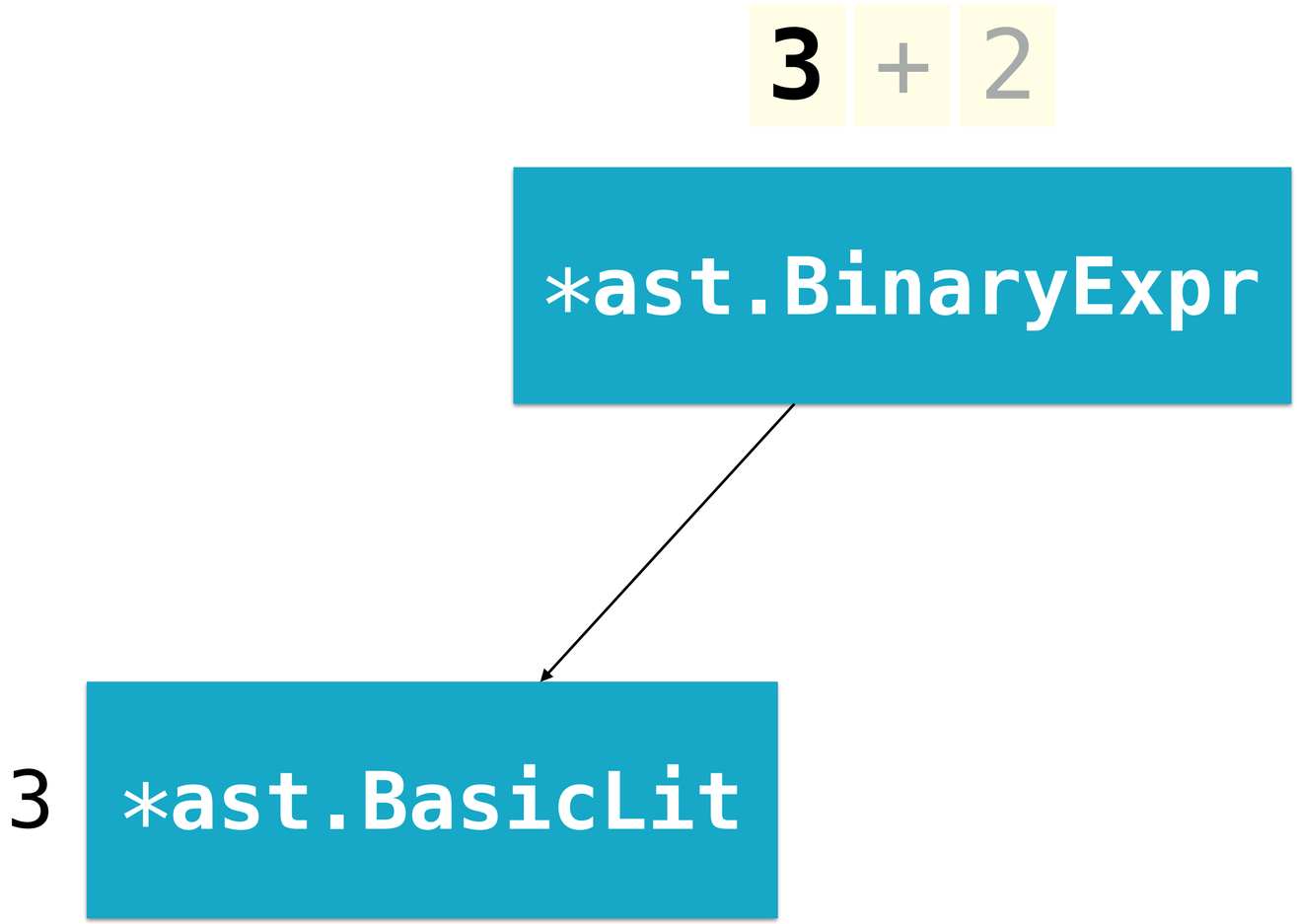

go/token
go/scanner
go/printer
go/ast
go/parser

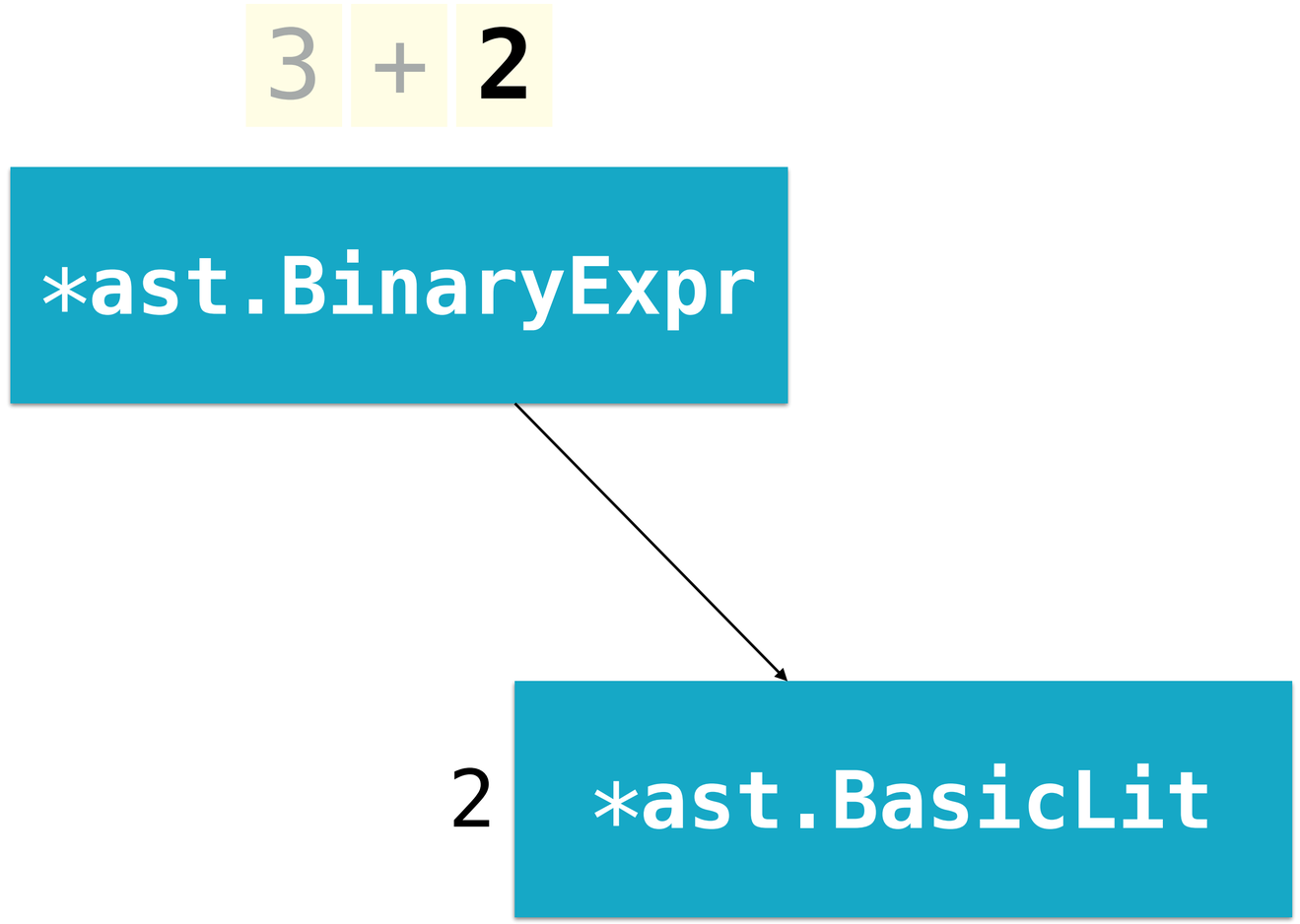
First,

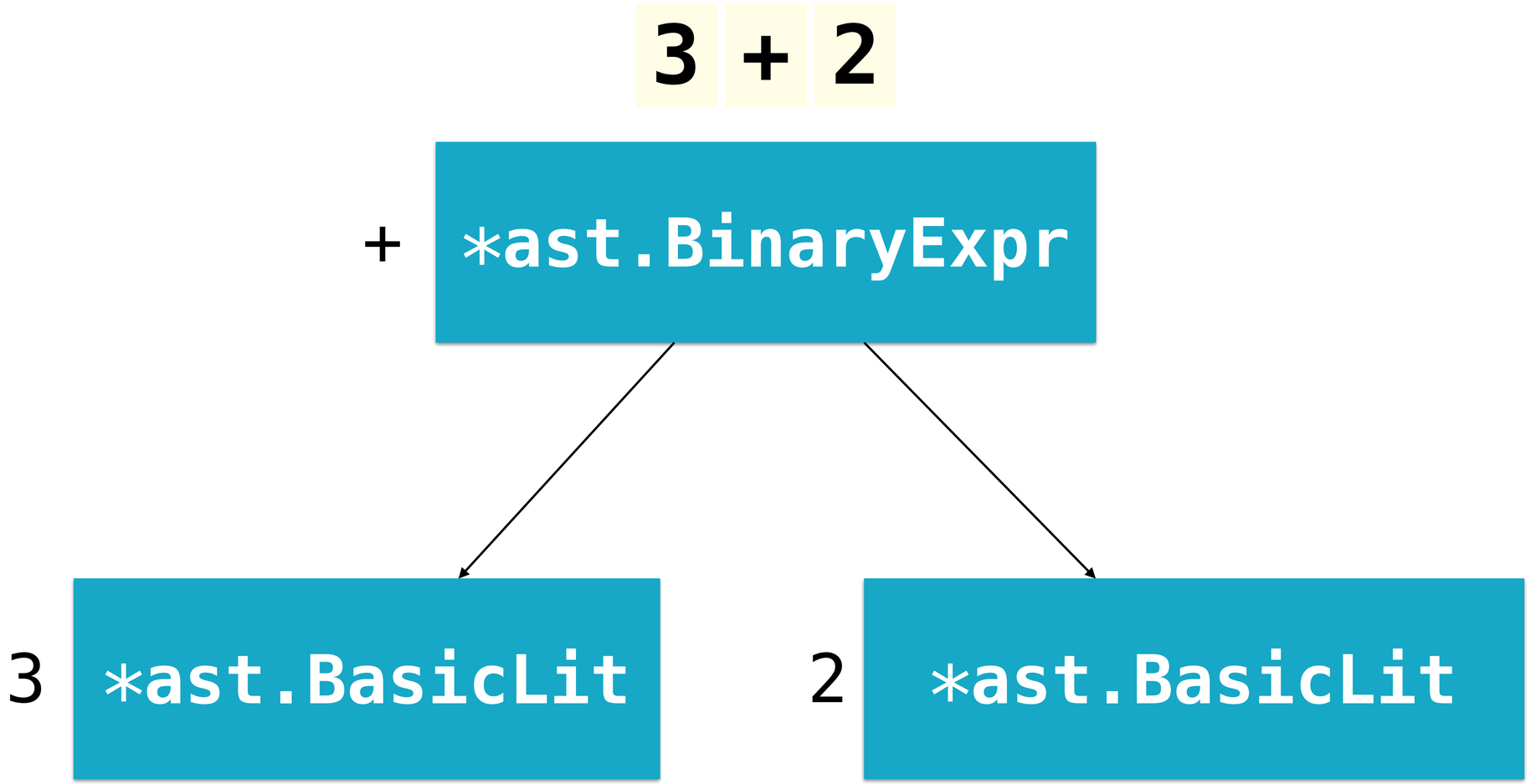
a quick tutorial about **go/ast**

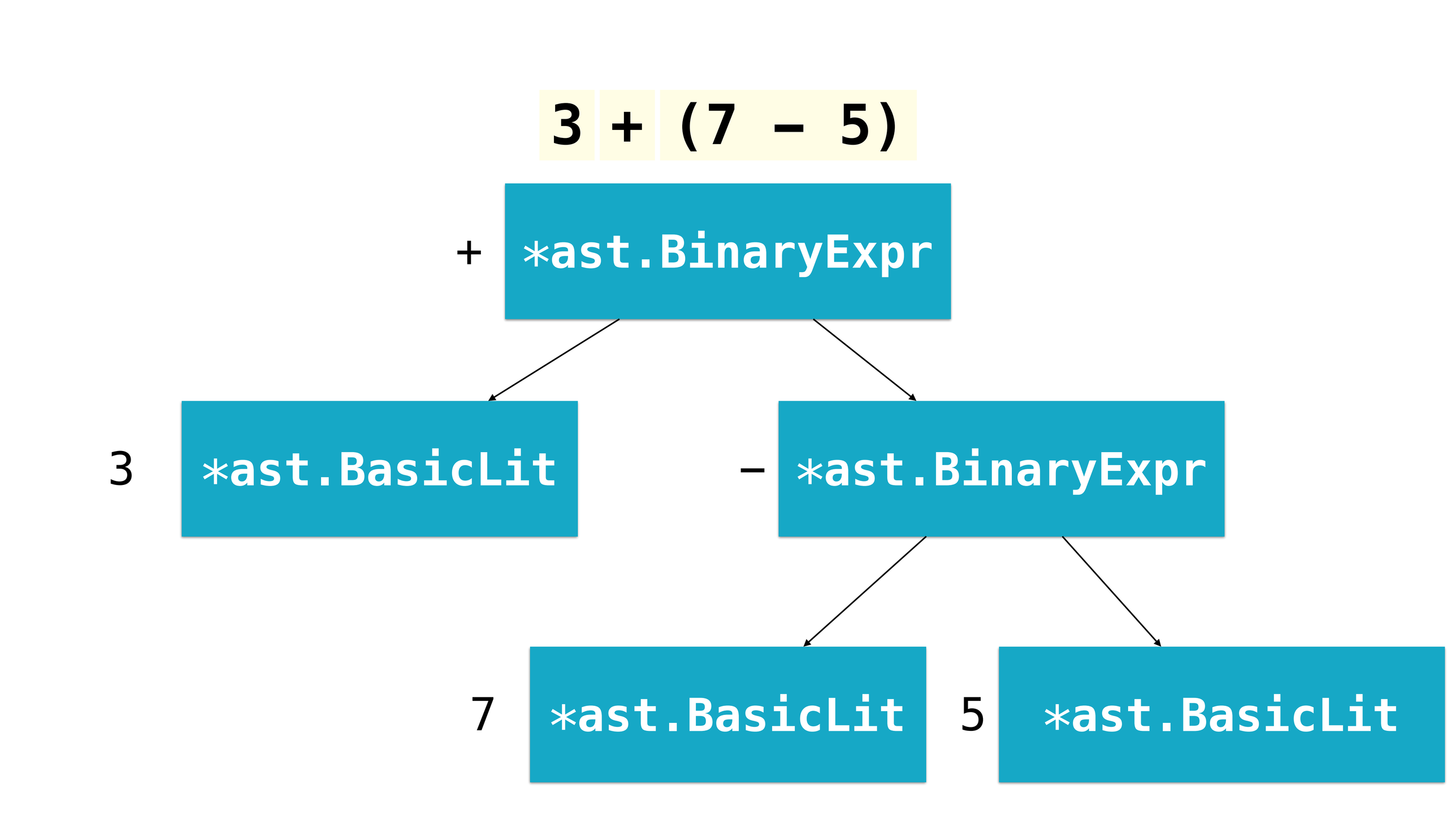
```
3 + 2
```

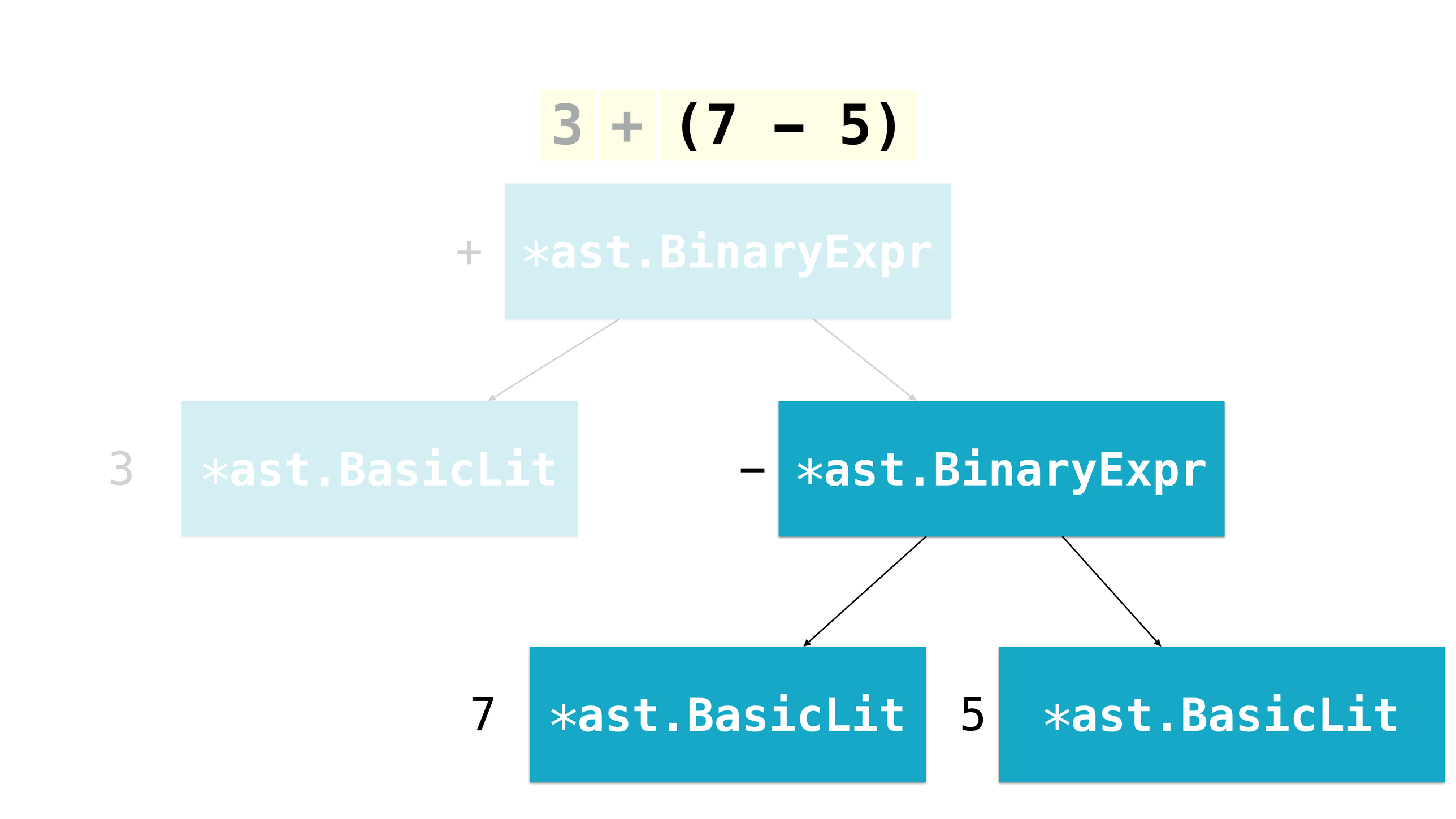
`*ast.BinaryExpr`


3 + 2
*ast.BinaryExpr
3
*ast.BasicLit


3 + 2
*ast.BinaryExpr
2
*ast.BasicLit


3 + 2
+
*ast.BinaryExpr
3
*ast.BasicLit
2
*ast.BasicLit

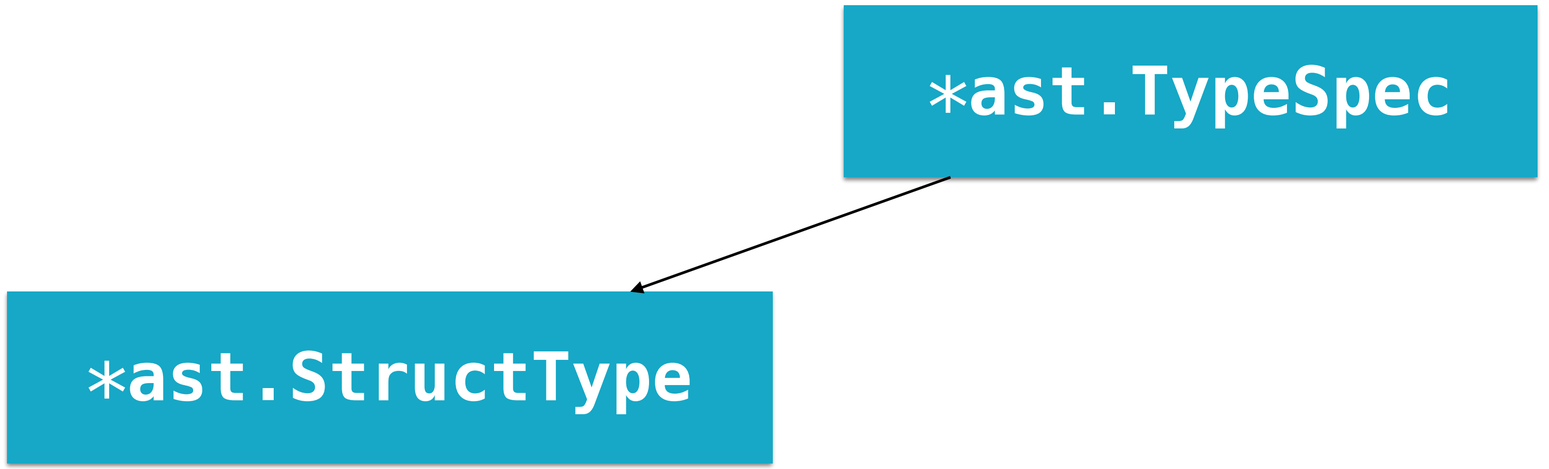

3 + (7 - 5)
+
*ast.BinaryExpr
3
*ast.BasicLit
-
*ast.BinaryExpr
7
*ast.BasicLit
5
*ast.BasicLit


3 + (7 - 5)
+
*ast.BinaryExpr
3
*ast.BasicLit
-
*ast.BinaryExpr
7
*ast.BasicLit
5
*ast.BasicLit

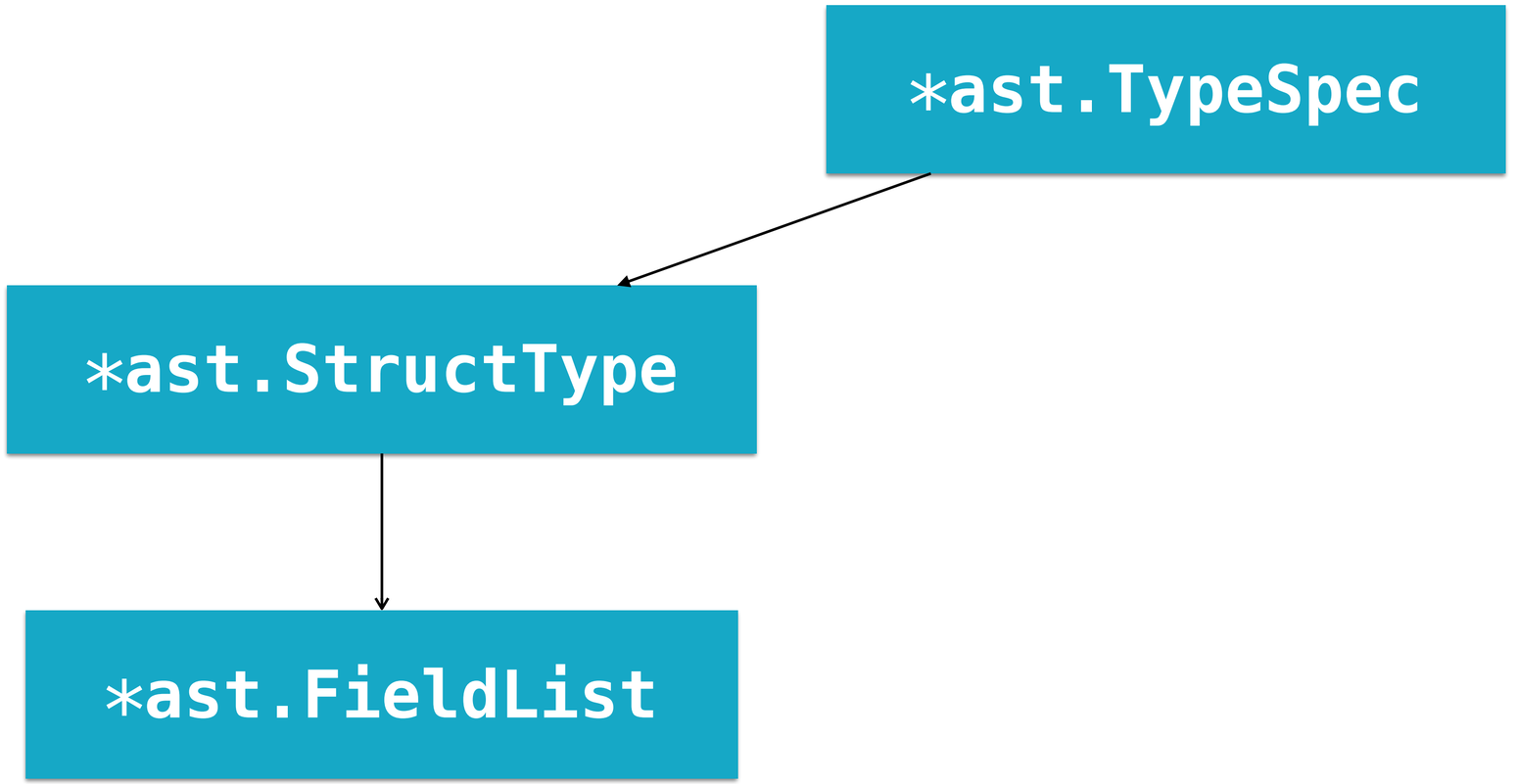
```
type Example struct {
    Foo string `json:"foo"`
}
```

***ast.TypeSpec**

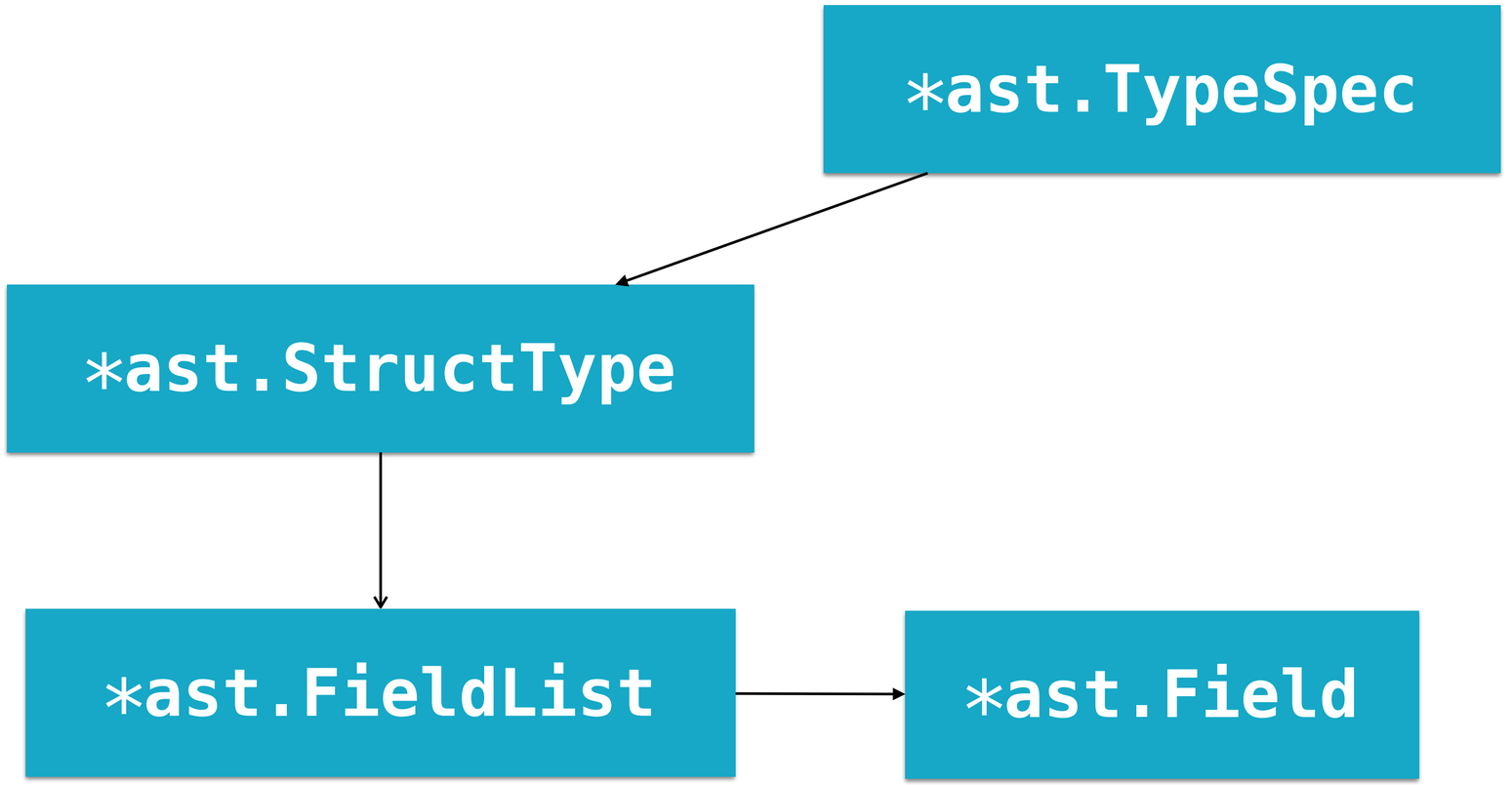
```
type Example struct {
    Foo string `json:"foo"`
}
```



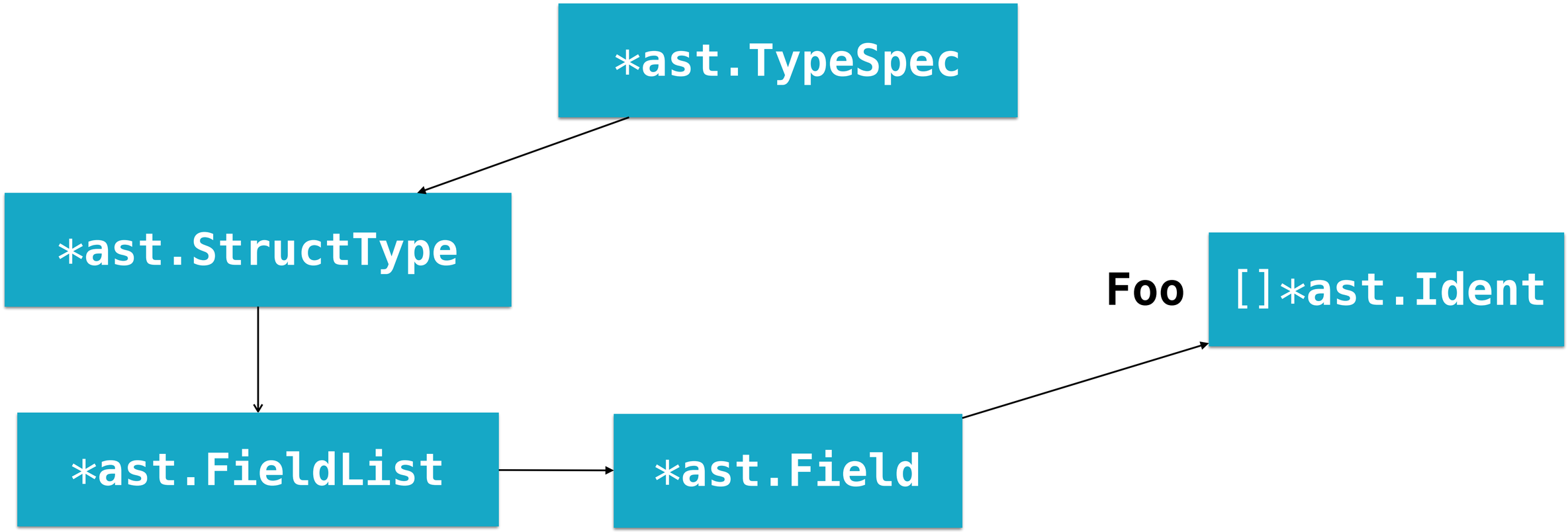
```
type Example struct {
    Foo string `json:"foo"`
}
```



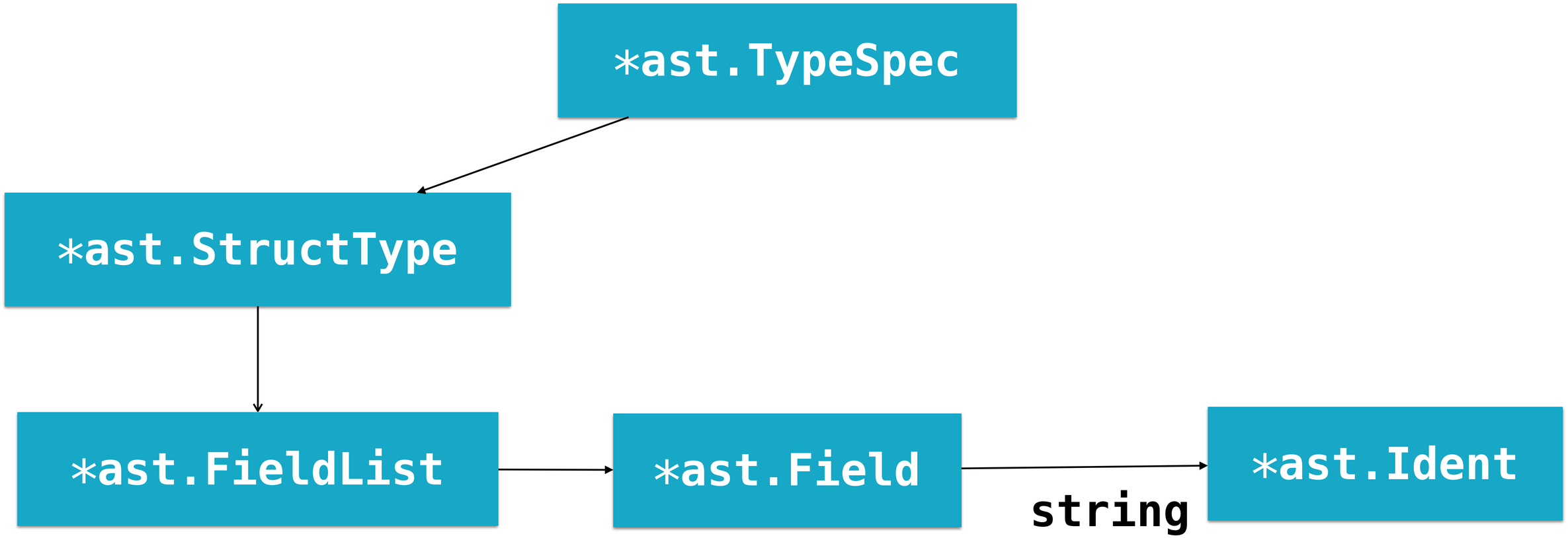
```
type Example struct {
    Foo string `json:"foo"`
}
```



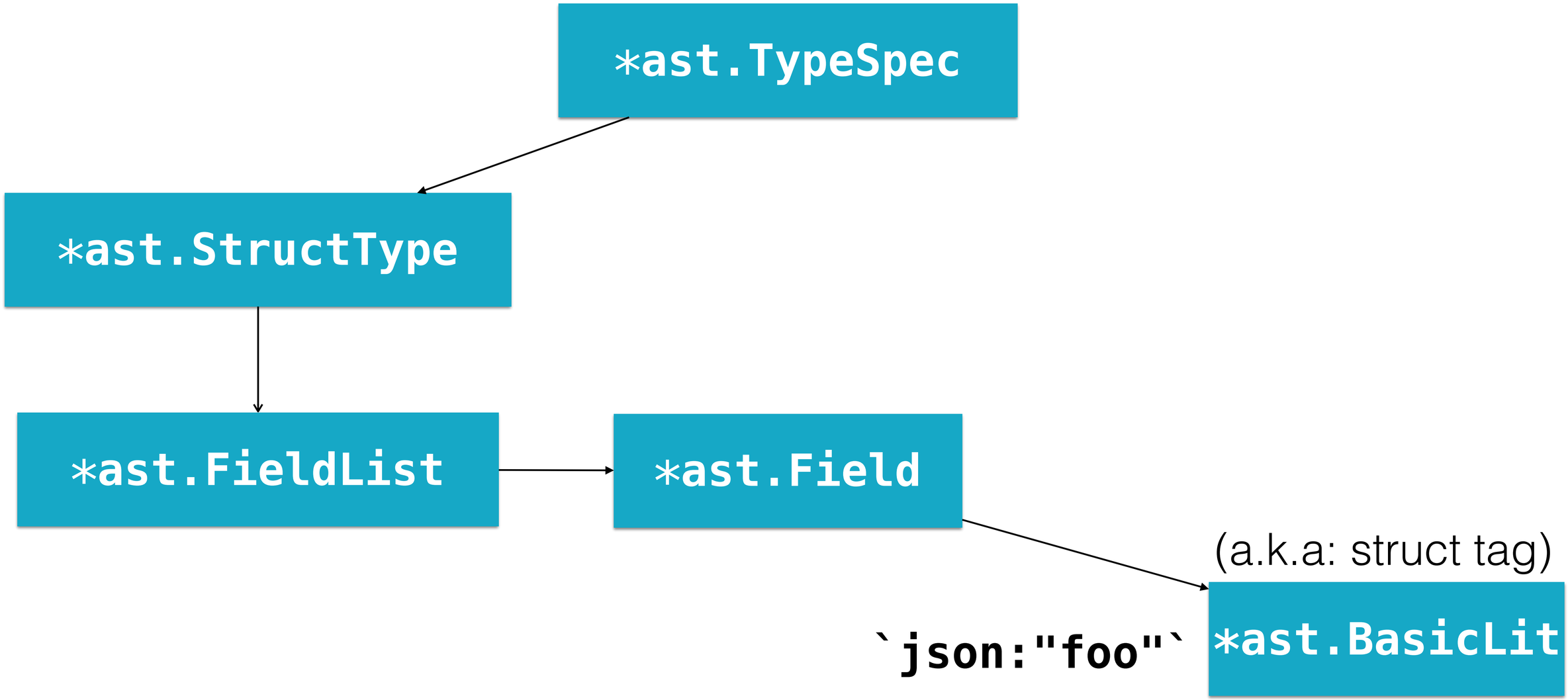
```
type Example struct {
    Foo string `json:"foo"`
}
```



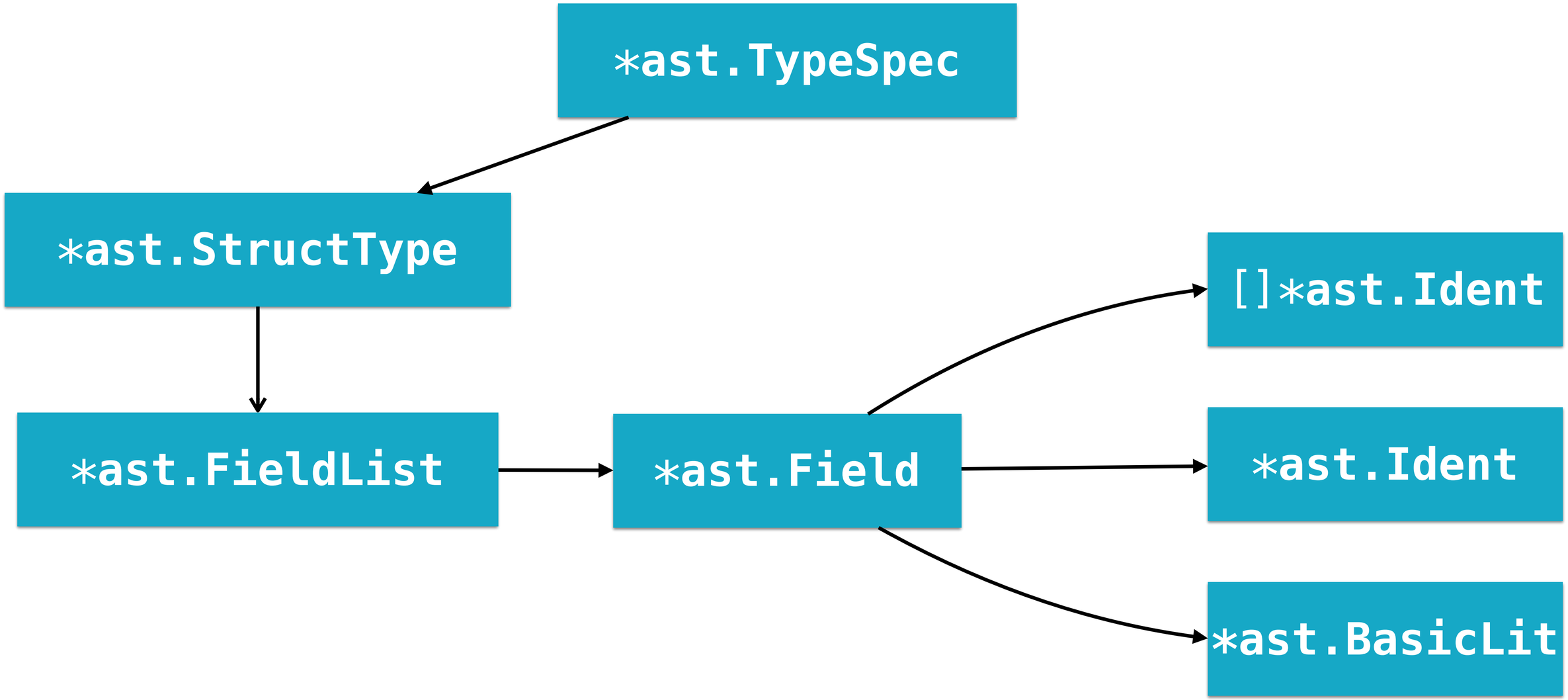
```
type Example struct {
    Foo string `json:"foo"`
}
```



```
type Example struct {
    Foo string `json:"foo"`
}
```



```
type Example struct {
    Foo string `json:"foo"`
}
```



ast.Node

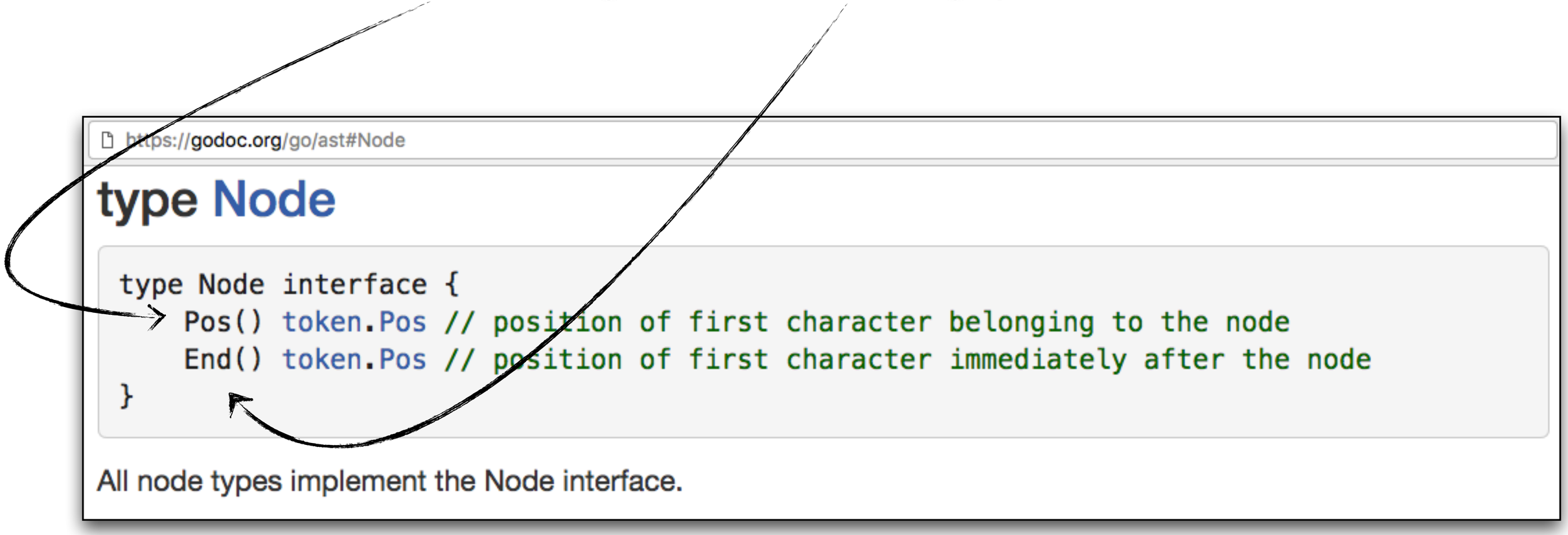
https://godoc.org/go/ast#Node

type Node

```
type Node interface {
    Pos() token.Pos // position of first character belonging to the node
    End() token.Pos // position of first character immediately after the node
}
```

All node types implement the Node interface.

All nodes define the **starting** and **ending** positions



How do we **parse a Struct**?

```
src := `package main
type Example struct {
   Foo string` + " `json:\"foo\"` }"

fset := token.NewFileSet()
file, err := parser.ParseFile(fset, "demo", src, parser.ParseComments)
if err != nil {
   panic(err)
}

ast.Inspect(file, func(x ast.Node) bool {
   s, ok := x.(*ast.StructType)
   if !ok {
      return true
   }

   for _, field := range s.Fields.List {
      fmt.Printf("Field: %s\n", field.Names[0].Name)
      fmt.Printf("Tag:   %s\n", field.Tag.Value)
   }
   return false
})
```

```
src := `package main
type Example struct {
   Foo string` + " `json:\"foo\"` }"

fset := token.NewFileSet()
file, err := parser.ParseFile(fset, "demo", src, parser.ParseComments)
if err != nil {
   panic(err)
}

ast.Inspect(file, func(x ast.Node) bool {
   s, ok := x.(*ast.StructType)
   if !ok {
      return true
   }

   for _, field := range s.Fields.List {
      fmt.Printf("Field: %s\n", field.Names[0].Name)
      fmt.Printf("Tag:   %s\n", field.Tag.Value)
   }
   return false
})
```

```
src := `package main
type Example struct {
   Foo string` + " `json:\"foo\"` }"

fset := token.NewFileSet()
file, err := parser.ParseFile(fset, "demo", src, parser.ParseComments)
if err != nil {
   panic(err)
}

ast.Inspect(file, func(x ast.Node) bool {
   s, ok := x.(*ast.StructType)
   if !ok {
      return true
   }

   for _, field := range s.Fields.List {
      fmt.Printf("Field: %s\n", field.Names[0].Name)
      fmt.Printf("Tag:   %s\n", field.Tag.Value)
   }
   return false
})
```

```
src := `package main
type Example struct {
   Foo string` + " `json:\"foo\"` }"

fset := token.NewFileSet()
file, err := parser.ParseFile(fset, "demo", src, parser.ParseComments)
if err != nil {
   panic(err)
}

ast.Inspect(file, func(x ast.Node) bool {
   s, ok := x.(*ast.StructType)
   if !ok {
      return true
   }

   for _, field := range s.Fields.List {
      fmt.Printf("Field: %s\n", field.Names[0].Name)
      fmt.Printf("Tag:   %s\n", field.Tag.Value)
   }
   return false
})
```

```
src := `package main
type Example struct {
   Foo string` + " `json:\"foo\"` }"

fset := token.NewFileSet()
file, err := parser.ParseFile(fset, "demo", src, parser.ParseComments)
if err != nil {
   panic(err)
}

ast.Inspect(file, func(x ast.Node) bool {
   s, ok := x.(*ast.StructType)
   if !ok {
      return true
   }

   for _, field := range s.Fields.List {
      fmt.Printf("Field: %s\n", field.Names[0].Name)
      fmt.Printf("Tag:   %s\n", field.Tag.Value)
   }
   return false
})
```

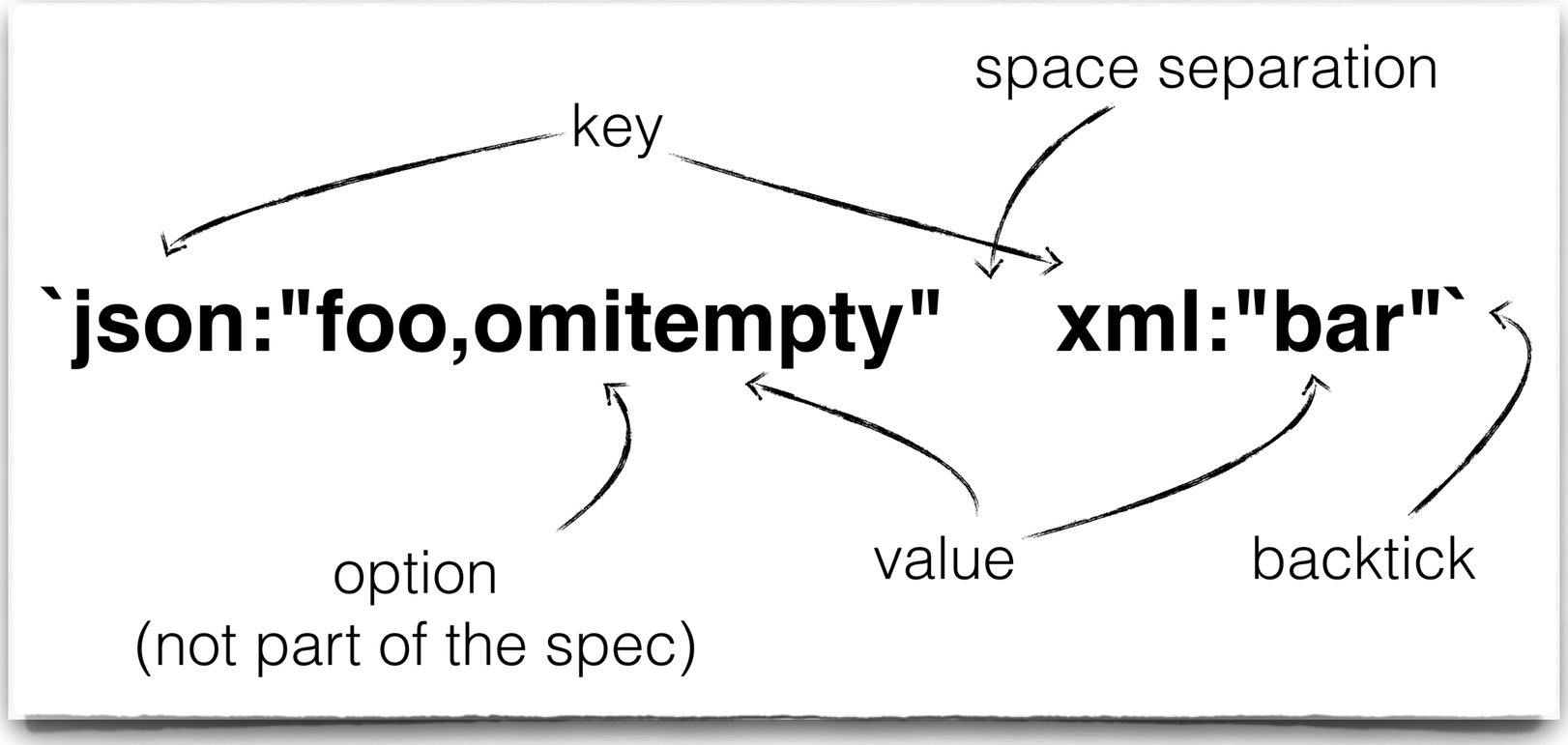
```
src := `package main
type Example struct {
   Foo string` + " `json:\"foo\"` }"

fset := token.NewFileSet()
file, err := parser.ParseFile(fset, "demo", src, parser.ParseComments)
if err != nil {
   panic(err)
}

ast.Inspect(file, func(x ast.Node) bool {
   s, ok := x.(*ast.StructType)
   if !ok {
      return true
   }

   for _, field := range s.Fields.List {
      fmt.Printf("Field: %s\n", field.Names[0].Name)
      fmt.Printf("Tag:   %s\n", field.Tag.Value)
   }
   return false
})
```

```
Field: Foo
Tag:   `json:"foo"`
```

2. Parse Struct Tag

How to **parse** a Struct Tag?

Remember this?



```
package main

import (
  "fmt"
  "reflect"
)

func main() {
  tag := reflect.StructTag(`json:"foo"`)
  value := tag.Get("json")

  fmt.Printf("value: %q\n", value)
}
```

```go
package main

import (
  "fmt"
  "reflect"
)

func main() {
  tag := reflect.StructTag(`json:"foo"`)
  value := tag.Get("json")

  fmt.Printf("value: %q\n", value)
}
```

```
$ go run main.go
value: "foo"
```

reflect.StructTag is not perfect ...

Issues with **reflect.StructTag**

- can't detect if the tag is malformed (only go vet knows that)
- doesn't know the semantics of options (i.e: omitempty)
- doesn't return all existing tags
- modifying existing tags is not possible

reflect

```
1151 // Lookup returns the value associated with key in the tag string.
1152 // If the key is present in the tag the value (which may be empty)
1153 // is returned. Otherwise the returned value will be the empty string.
1154 // The ok return value reports whether the value was explicitly set in
1155 // the tag string. If the tag does not have the conventional format,
1156 // the value returned by Lookup is unspecified.
1157 func (tag StructTag) Lookup(key string) (value string, ok bool) {
1158     // When modifying this code, also update the validateStructTag code
1159     // in cmd/vet/structtag.go.
1160
1161     for tag != "" {
1162         // Skip leading space.
1163         i := 0
1164         for i < len(tag) && tag[i] == ' ' {
1165             i++
1166         }
1167         tag = tag[i:]
1168         if tag == "" {
1169             break
1170         }
1171
```

cmd/vet

```
120 // validateStructTag parses the struct tag and returns an error if it is not
121 // in the canonical format, which is a space-separated list of key:"value"
122 // settings. The value may contain spaces.
123 func validateStructTag(tag string) error {
124     // This code is based on the StructTag.Get code in package reflect.
125
126     n := 0
127     for ; tag != ""; n++ {
128         if n > 0 && tag != "" && tag[0] != ' ' {
129             // More restrictive than reflect, but catches likely mistakes
130             // like `x:"foo",y:"bar"`, which parses as `x:"foo" ,y:"bar"` with
131             return errTagSpace
132         }
133         // Skip leading space.
134         i := 0
135         for i < len(tag) && tag[i] == ' ' {
136             i++
137         }
138         tag = tag[i:]
139         if tag == "" {
140             break
141         }
```

Let's **improve it**
with a custom package

import "**github.com/fatih/structtag**"

structtag

Index

Parse and list all tags

```
tags, err := structtag.Parse(`json:"foo,omitempty" xml:"foo"`)
if err != nil {
   panic(err)
}

// iterate over all key-value pairs
for _, t := range tags.Tags() {
   fmt.Printf("tag: %+v\n", t)
}
```

```
$ go run main.go
tag: json:"foo,omitempty"
tag: xml:"foo"
```

Get a single Tag

```
tags, err := structtag.Parse(`json:"foo,omitempty" xml:"foo"`)
if err != nil {
   panic(err)
}

jsonTag, err := tags.Get("json")
if err != nil {
   panic(err)
}
```

```
fmt.Println(jsonTag)          // Output: json:"foo,omitempty"
fmt.Println(jsonTag.Key)      // Output: json
fmt.Println(jsonTag.Name)     // Output: foo
fmt.Println(jsonTag.Options) // Output: [omitempty]
```

Change existing tag

```
jsonTag, err := tags.Get(`json:"foo,omitempty"`)
if err != nil {
   panic(err)
}

jsonTag.Name = "bar"
jsonTag.Options = nil
tags.Set(jsonTag)

fmt.Println(tags)
```

```
json:"bar"
```

Add new tag

```
tags, err := structtag.Parse(`json:"foo,omitempty" xml:"foo"`)

hclTag := &structtag.Tag{
   Key:     "hcl",
   Name:    "gopher",
   Options: []string{"squash"},
}

// add new tag
tags.Set(hclTag)

fmt.Println(tags)
```

```
json:"foo,omitempty" xml:"foo" hcl:"gopher,squash"
```

Add/remove options

```
tags, err := structtag.Parse(`json:"foo" xml:"bar,comment"`)
if err != nil {
   panic(err)
}

tags.AddOptions("json", "omitempty")

tags.DeleteOptions("xml", "comment")
```

```
fmt.Println(tags) // json:"foo,omitempty" xml:"bar"
```

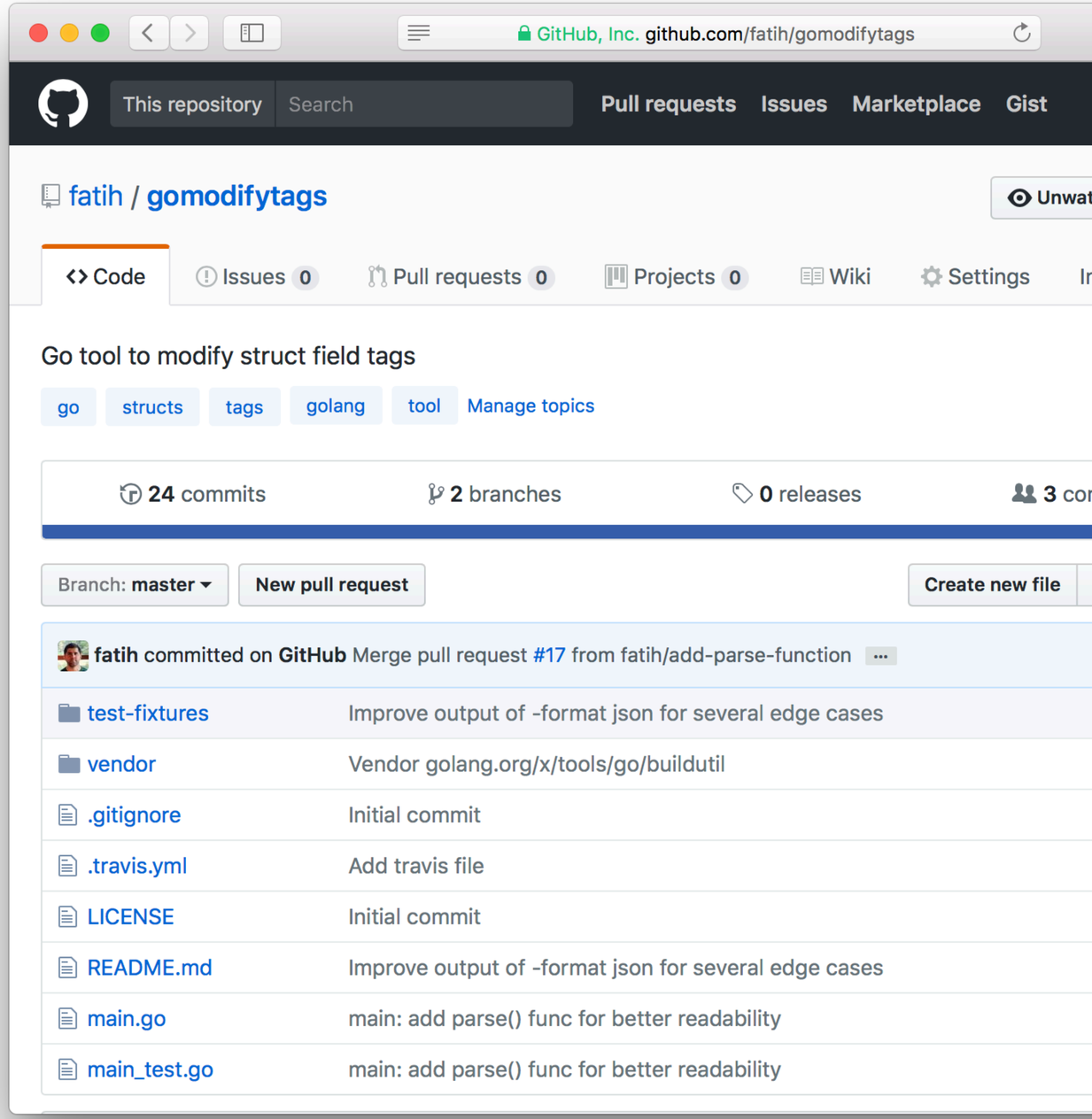
Both issues are fixed now 👍

✅ Parse struct (using **go/parser**)

✅ Parse struct tag (using **fatih/structtag**)

Write a **CLI tool**

gomodifytags



go get **github.com/fatih/gomodifytags**

1. **Fetch** configuration settings
2. **Parse** content
3. **Find** selection
4. **Modify** the struct tag
5. **Output** the result

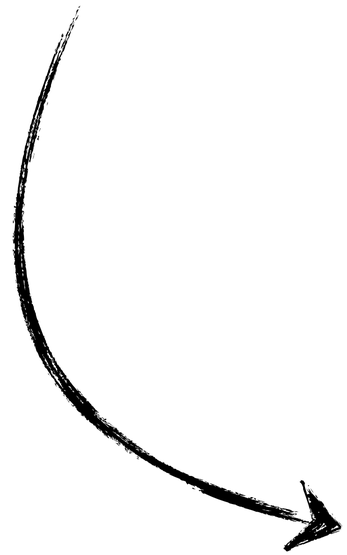
```
func main() {
   var cfg config

   node = cfg.parse()

   start, end = cfg.findSelection(node)

   rewritten = cfg.rewrite(node, start, end)

   out = cfg.format(rewritten)

   fmt.Println(out)
}
```

1. **Fetch** configuration settings
2. **Parse** content
3. **Find** selection
4. **Modify** the struct tag
5. **Output** the result

```
func main() {
   var cfg config

   node = cfg.parse()

   start, end = cfg.findSelection(node)

   rewritten = cfg.rewrite(node, start, end)

   out = cfg.format(rewritten)

   fmt.Println(out)
}
```

Fetch **configuration** settings

```
$ gomodifytags --file example.go ...
```



```
cfg := &config{
   file:         *flagFile,
   line:         *flagLine,
   structName:   *flagStruct,
   offset:       *flagOffset,
   output:       *flagOutput,
   write:        *flagWrite,
   clear:        *flagClearTags,
   clearOption:  *flagClearOptions,
   transform:    *flagTransform,
   sort:         *flagSort,
   override:     *flagOverride,
}
```

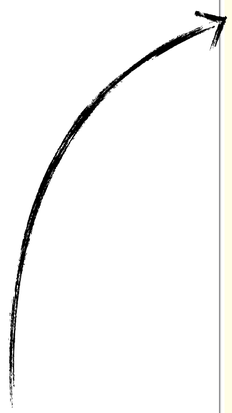
Use **flags** to set configuration

Things we need:

1. What **content** to process
2. Where and what to **output**
3. Which **struct** to modify
4. Which **tags** to modify

Things we need:

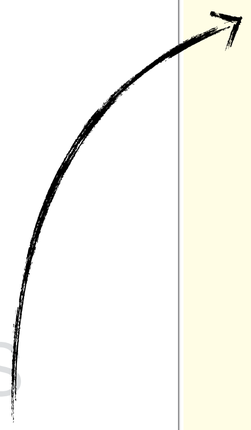
1. What **content** to process
2. Where and what to **output**
3. Which **struct** to modify
4. Which **tags** to modify



```
type config struct {
   file      string
   modified  io.Reader
   output    string
   write     bool

   offset      int
   structName  string
   line        string
   start, end  int

   remove    []string
   add       []string
   override  bool
   transform string
   sort      bool
   clear     bool

   addOpts    []string
   removeOpts []string
   clearOpt   bool
}
```

Things we need:

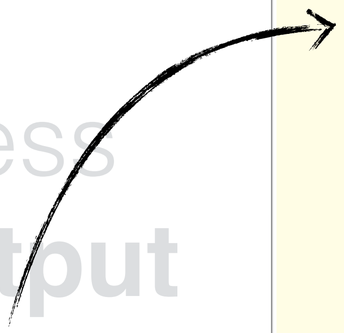
1. What **content** to process
2. Where and what to **output**
3. Which **struct** to modify
4. Which **tags** to modify



```
type config struct {
   file      string
   modified  io.Reader
   output    string
   write     bool

   offset      int
   structName  string
   line        string
   start, end  int

   remove    []string
   add       []string
   override  bool
   transform string
   sort      bool
   clear     bool

   addOpts    []string
   removeOpts []string
   clearOpt   bool
}
```

Things we need:

1. What **content** to process
2. Where and what to **output**
3. Which **struct** to modify
4. Which **tags** to modify



```
type config struct {
   file      string
   modified  io.Reader
   output    string
   write     bool

   offset     int
   structName string
   line       string
   start, end int

   remove    []string
   add       []string
   override  bool
   transform string
   sort      bool
   clear     bool

   addOpts    []string
   removeOpts []string
   clearOpt   bool
}
```

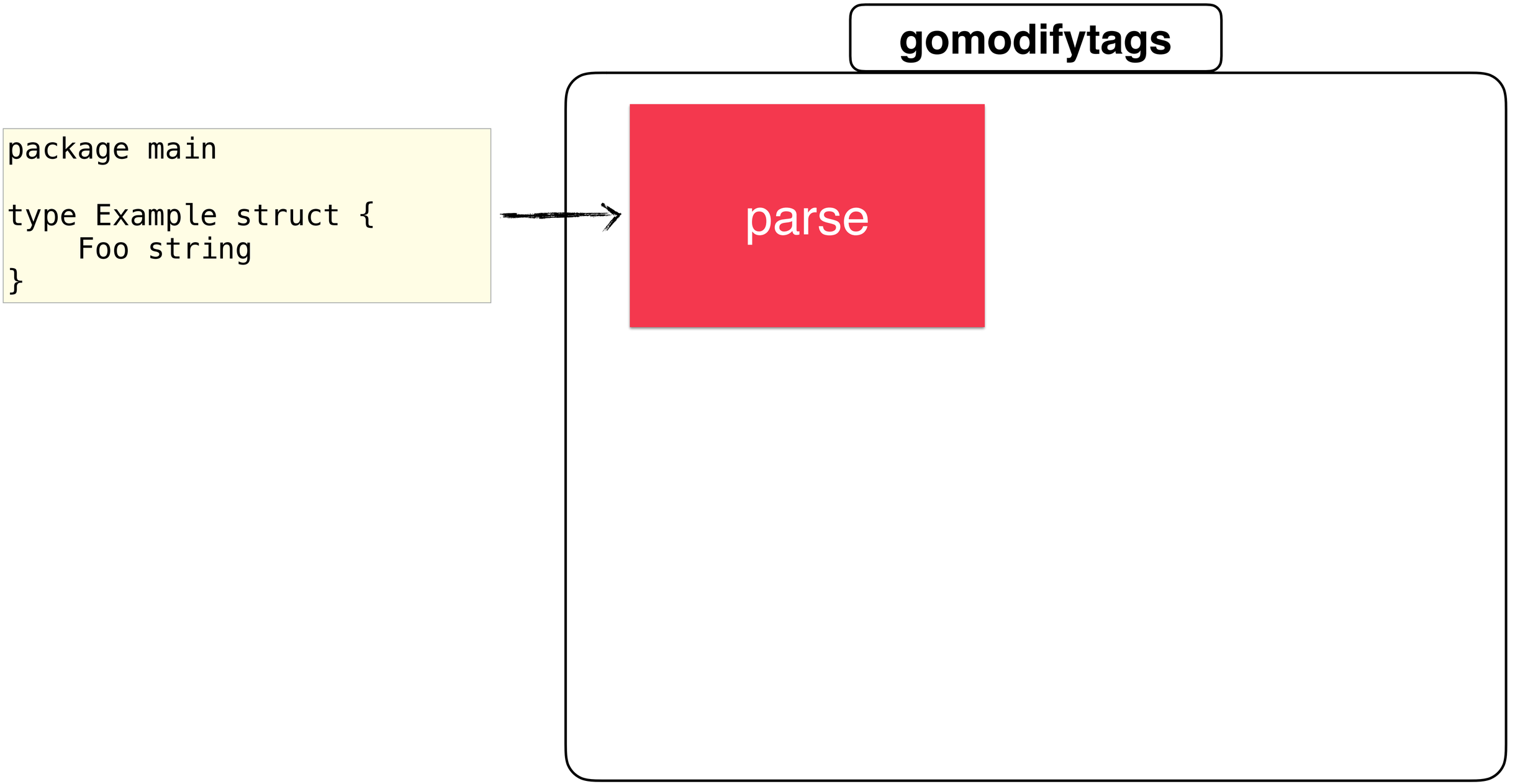
Things we need:

1. What **content** to process
2. Where and what to **output**
3. Which **struct** to modify
4. Which **tags** to modify

```
type config struct {
   file      string
   modified  io.Reader
   output    string
   write     bool

   offset      int
   structName  string
   line        string
   start, end  int

   remove    []string
   add       []string
   override  bool
   transform string
   sort      bool
   clear     bool

   addOpts    []string
   removeOpts []string
   clearOpt   bool
}
```

```
package main

type Example struct {
    Foo string
}
```

$ **gomodifytags**
-**file** example.go
-**struct** Example
-**add-tags** json

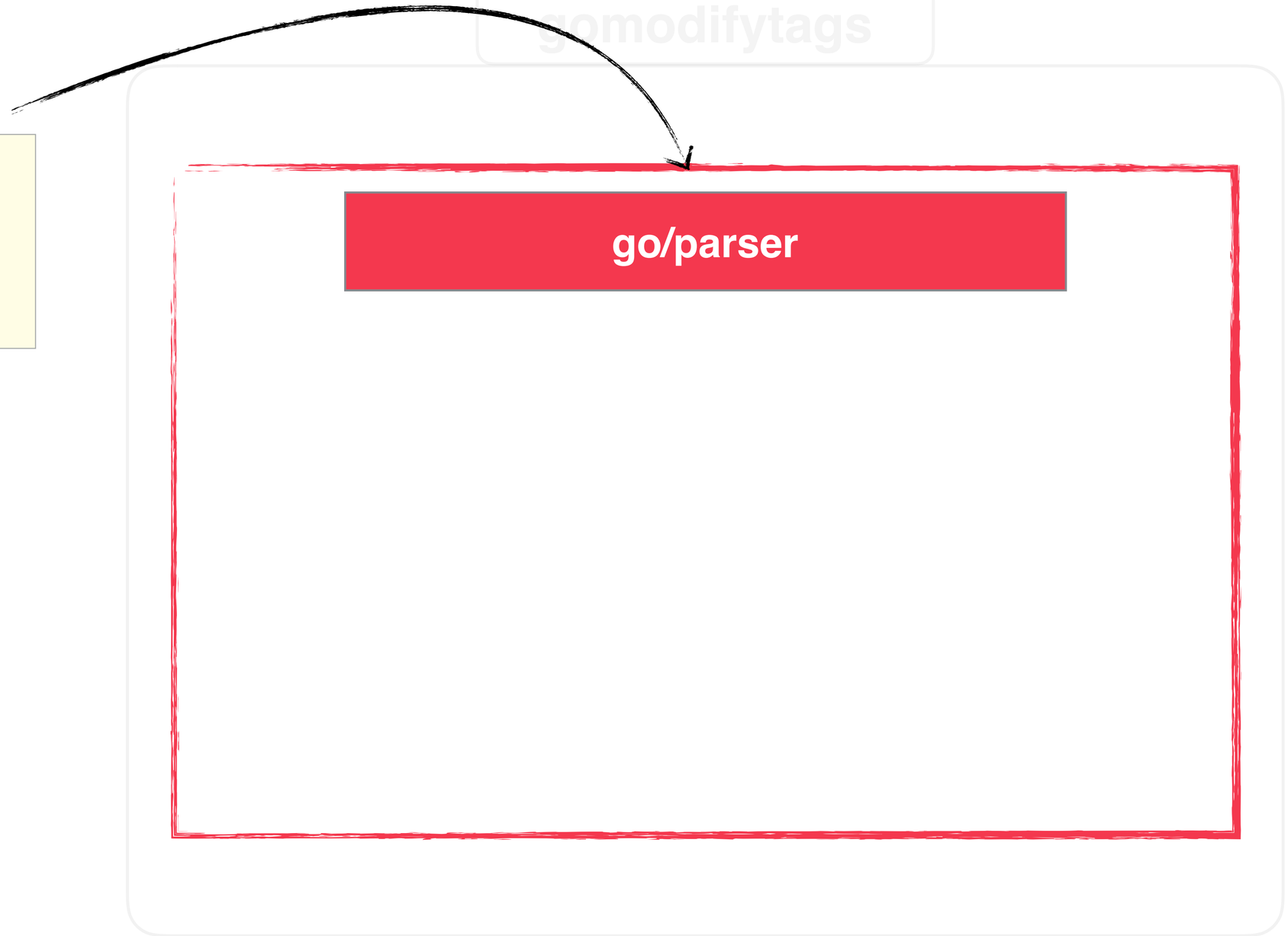

gomodifytags
package main
type Example struct {
Foo string
}
parse

Parse content

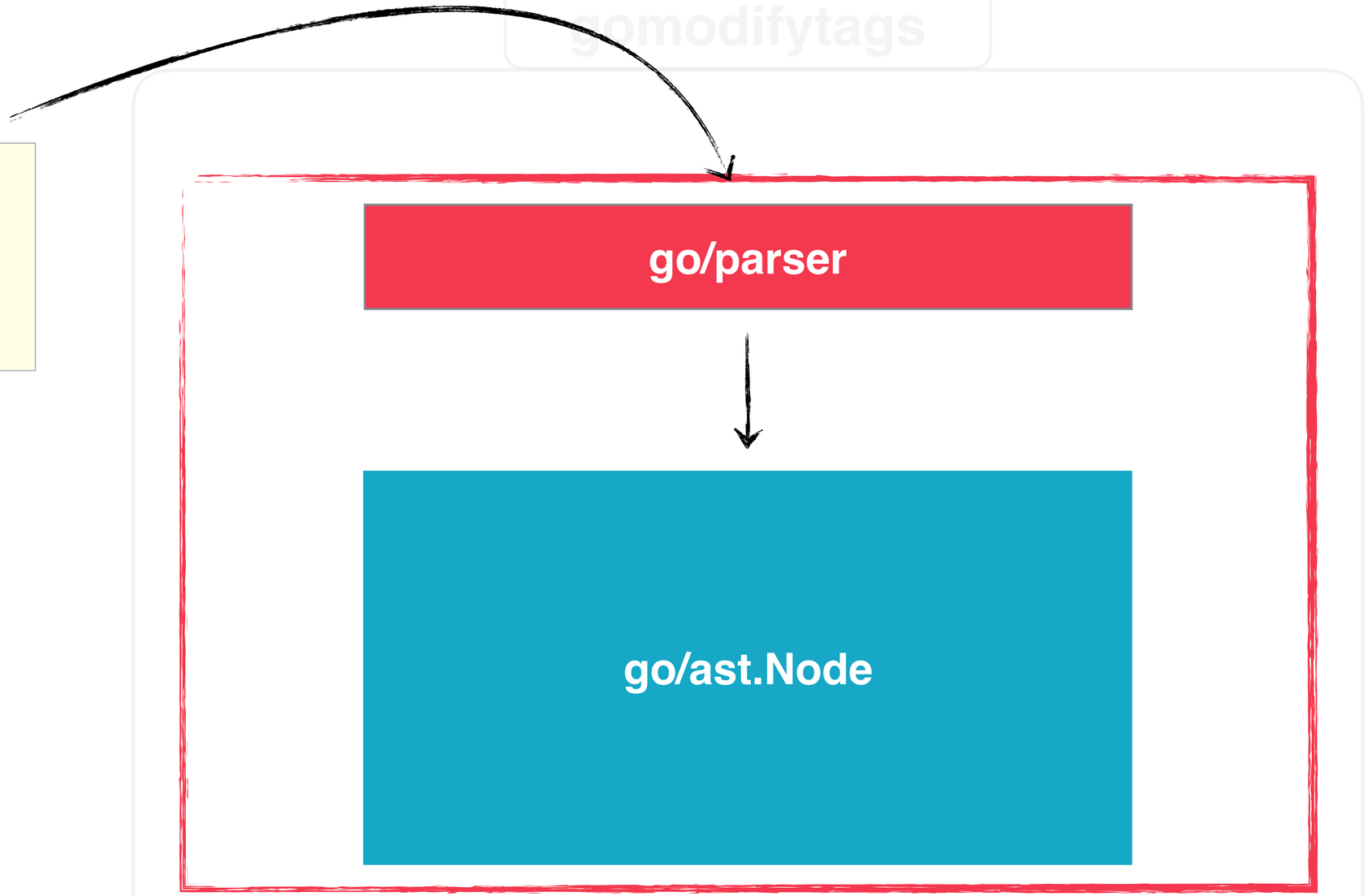
1. **Fetch** configuration settings
2. **Parse** content
3. **Find** selection
4. **Modify** the struct tag
5. **Output** the result

```
func main() {
   var cfg config

   node = cfg.parse()

   start, end = cfg.findSelection(node)

   rewritten = cfg.rewrite(node, start, end)

   out = cfg.format(rewritten)

   fmt.Println(out)
}
```

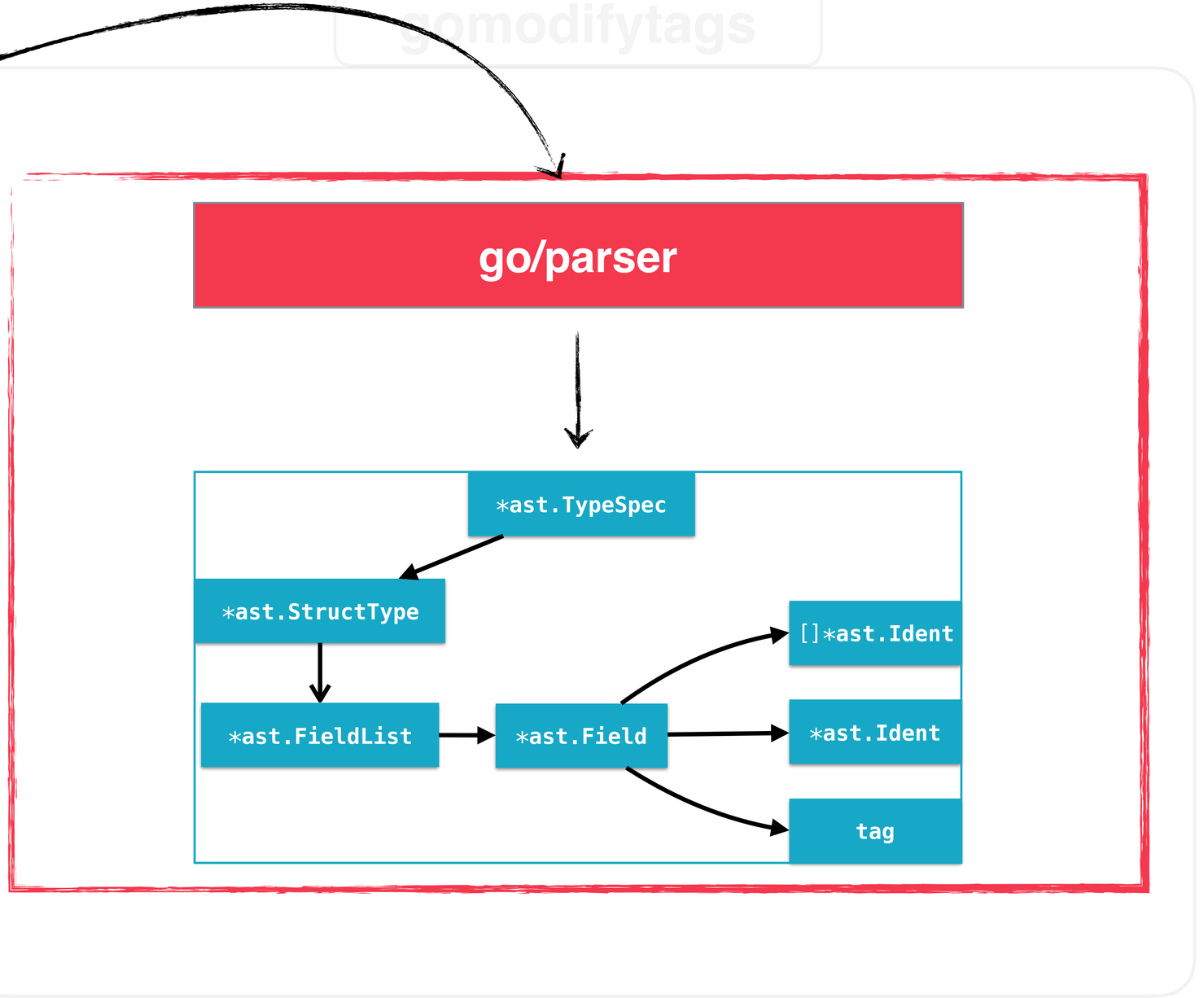
```
package main

type Example struct {
    Foo string
}
```



```
package main

type Example struct {
    Foo string
}
```



```
package main

type Example struct {
    Foo string
}
```



Parse content

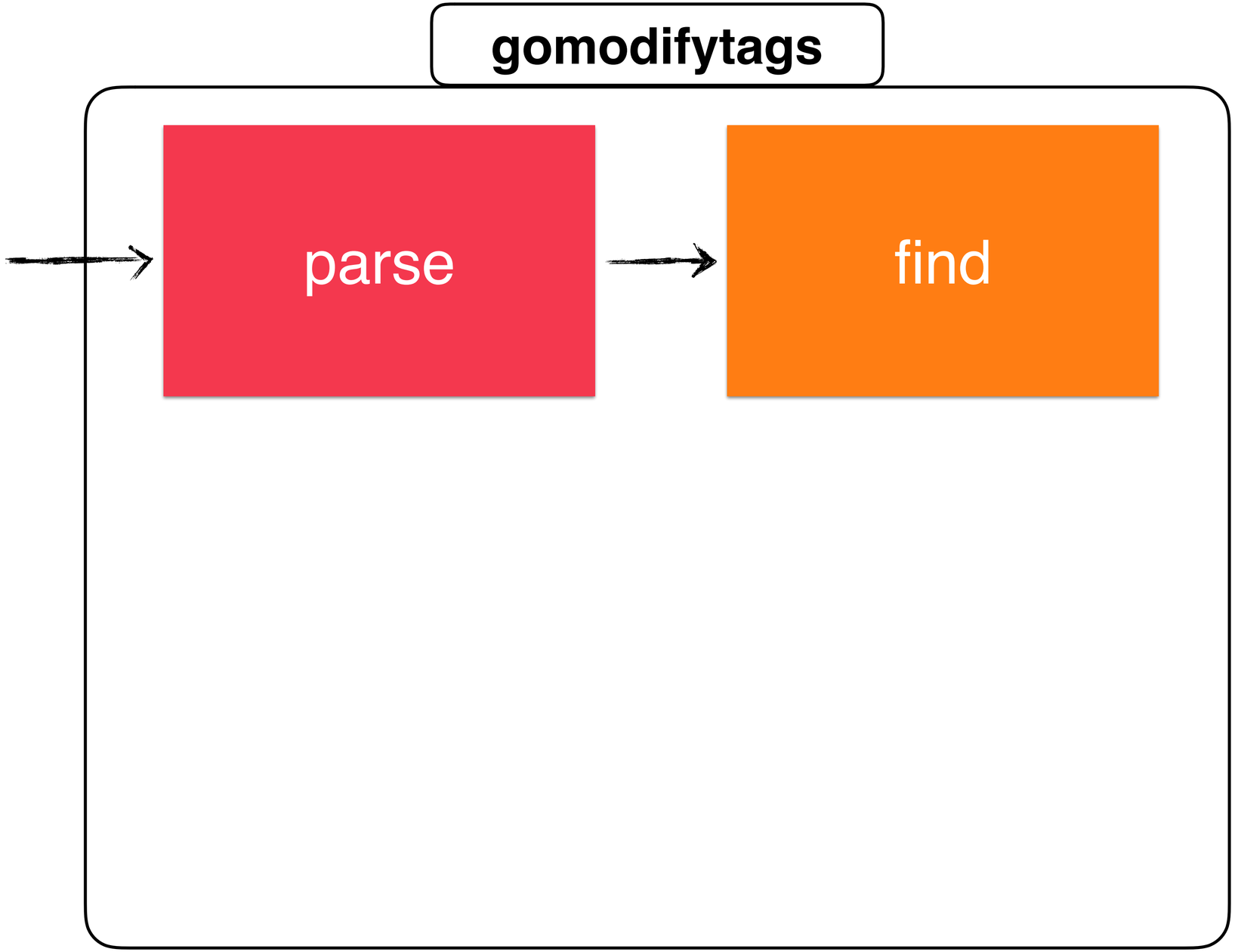
```
func (c *config) parse() (ast.Node, error) {
   c.fset = token.NewFileSet()
   var contents interface{}
   if c.modified != nil {
      archive, err := buildutil.ParseOverlayArchive(c.modified)
      if err != nil {
         return nil, fmt.Errorf("failed to parse -modified archive: %v", err)
      }
      fc, ok := archive[c.file]
      if !ok {
         return nil, fmt.Errorf("couldn't find %s in archive", c.file)
      }
      contents = fc
   }

   return parser.ParseFile(c.fset, c.file, contents, parser.ParseComments)
}
```

Parse content (cont.)

```
func (c *config) parse() (ast.Node, error) {
   c.fset = token.NewFileSet()
   var contents interface{}
   if c.modified != nil {
      archive, err := buildutil.ParseOverlayArchive(c.modified)
      if err != nil {
         return nil, fmt.Errorf("failed to parse -modified archive: %v", err)
      }
      fc, ok := archive[c.file]
      if !ok {
         return nil, fmt.Errorf("couldn't find %s in archive", c.file)
      }
      contents = fc
   }

   return parser.ParseFile(c.fset, c.file, contents, parser.ParseComments)
}
```

gomodifytags

```
package main

type Example struct {
    Foo string
}
```



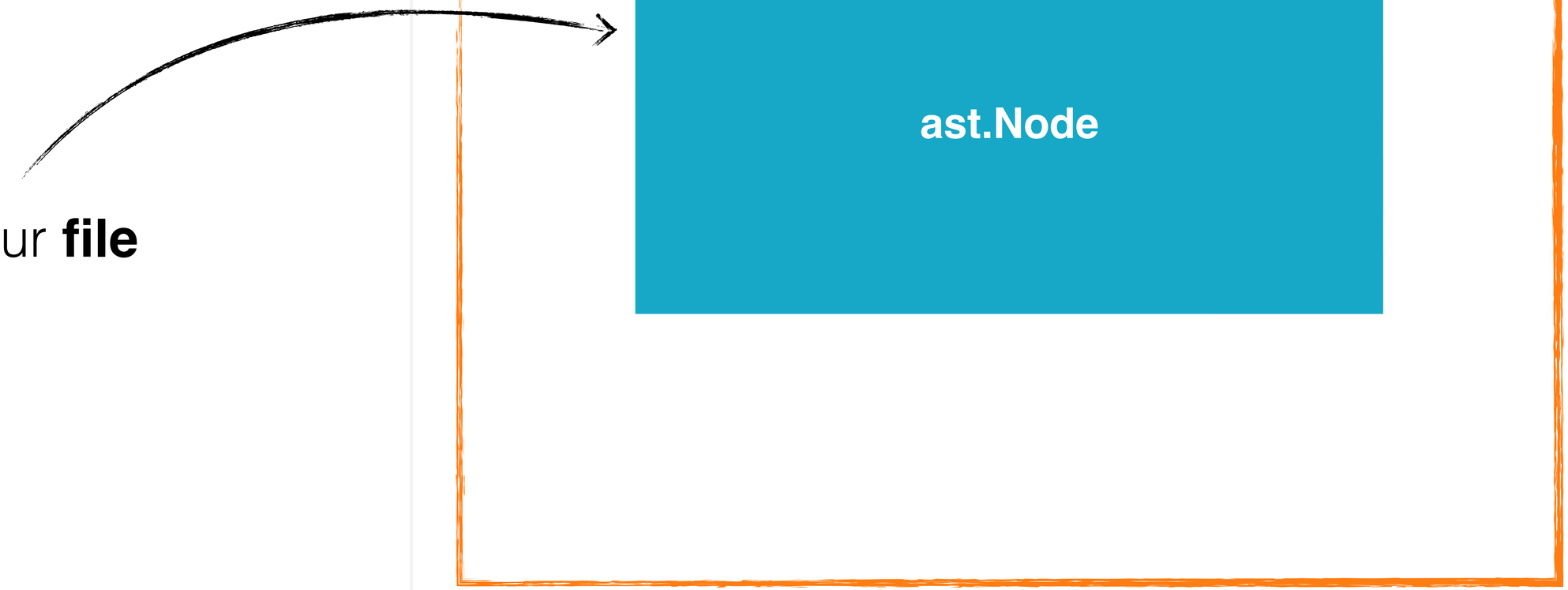
Find **start** and **end** positions

1. **Fetch** configuration settings
2. **Parse** content
3. **Find** selection
4. **Modify** the struct tag
5. **Output** the result

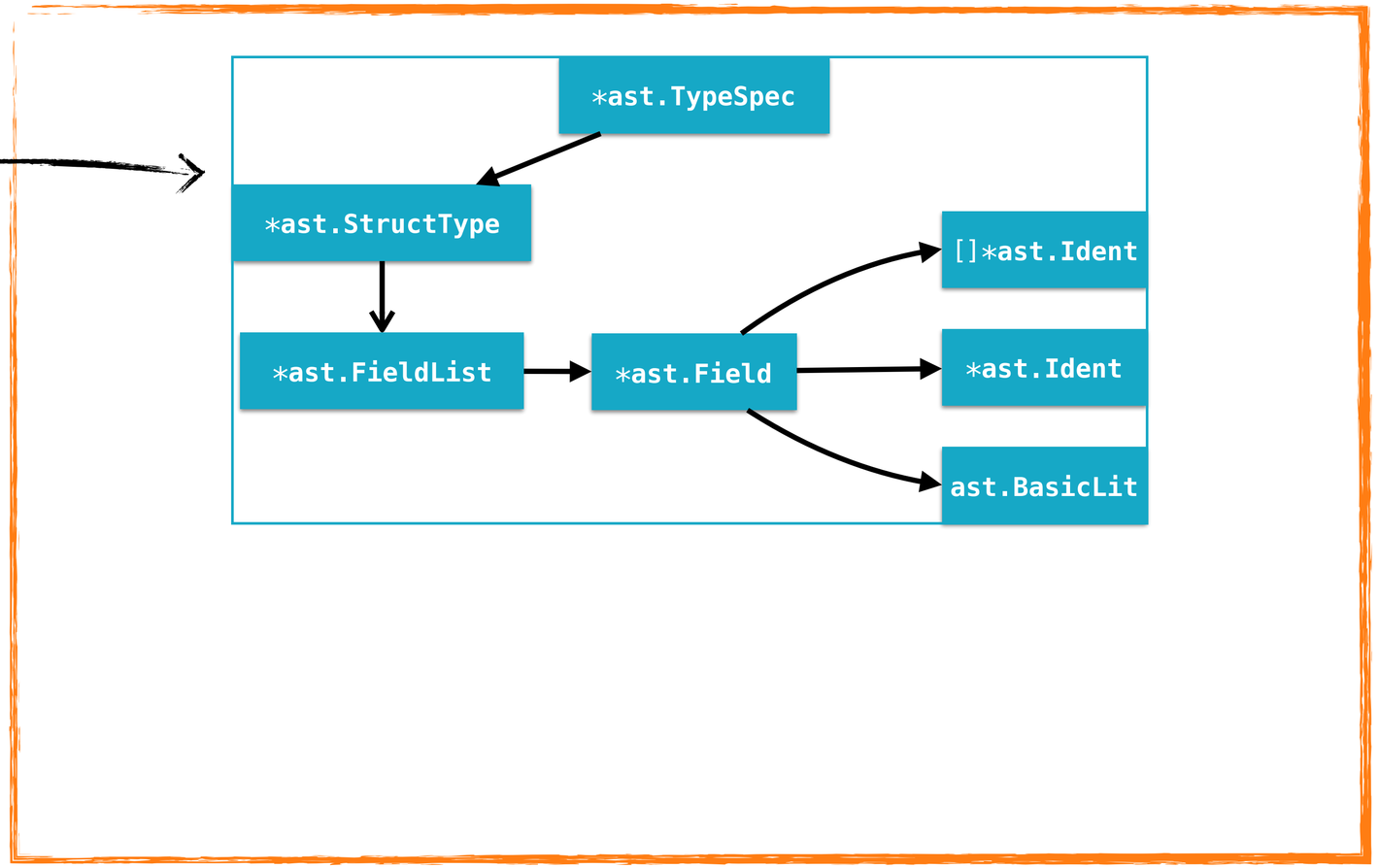
```
func main() {
   var cfg config

   node = cfg.parse()

   start, end = cfg.findSelection(node)

   rewritten = cfg.rewrite(node, start, end)

   out = cfg.format(rewritten)

   fmt.Println(out)
}
```

gomodifytags

this is our **file**

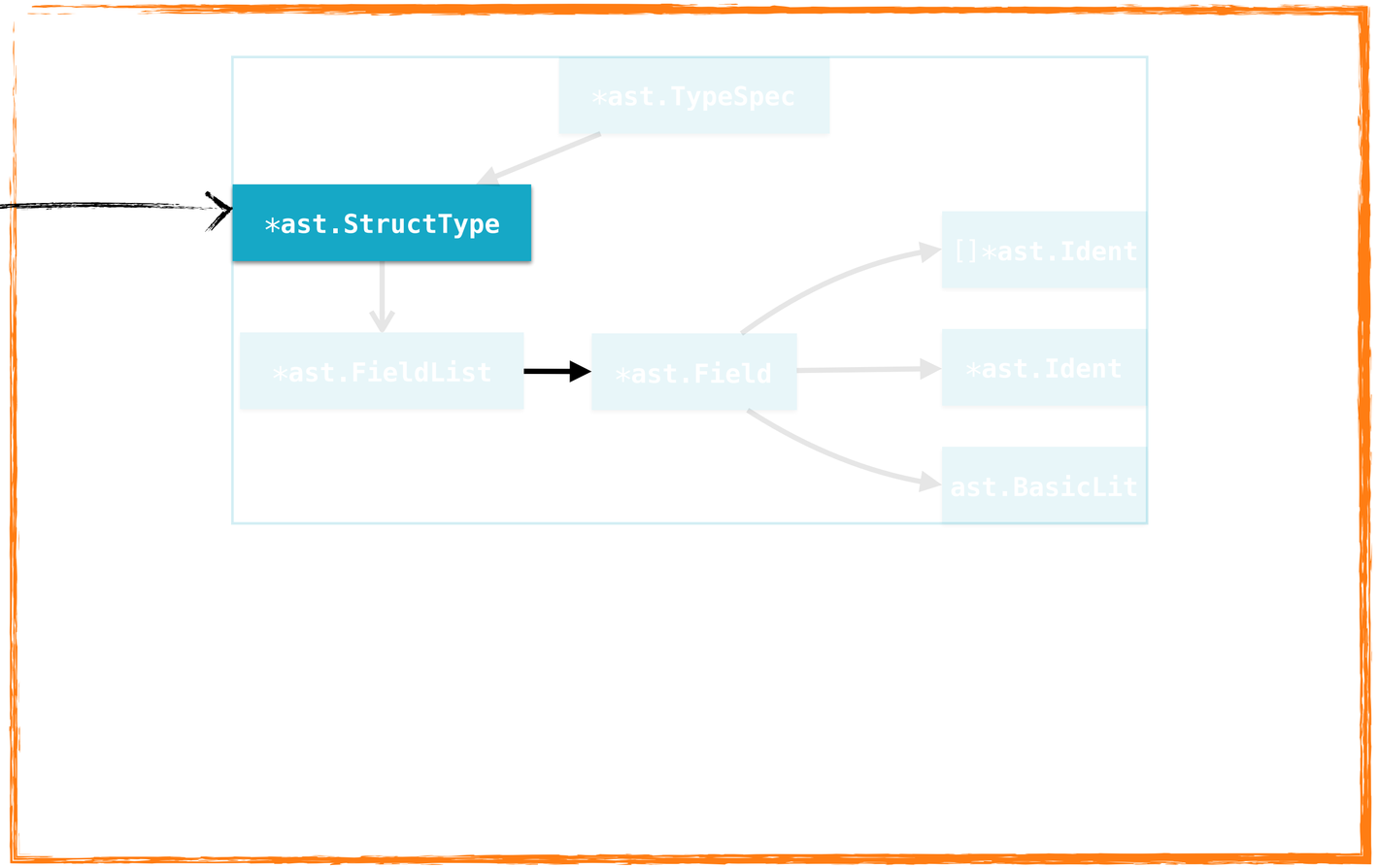


which is an **AST** tree



gomodifytags

select **struct** that **matches** our criteria ...

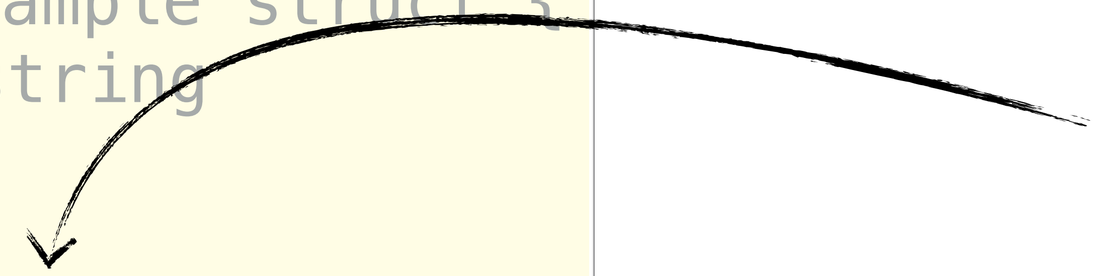


There are **many ways** to find a struct

```
 1 package main
 2
 3 type Example struct {
 4   Foo string
 5 }
 6
 7 type Server struct {
 8   Name       string
 9   Port       int
10   EnableLogs bool
11 }
12
13 type Person struct {
14   Name string
15 }
```

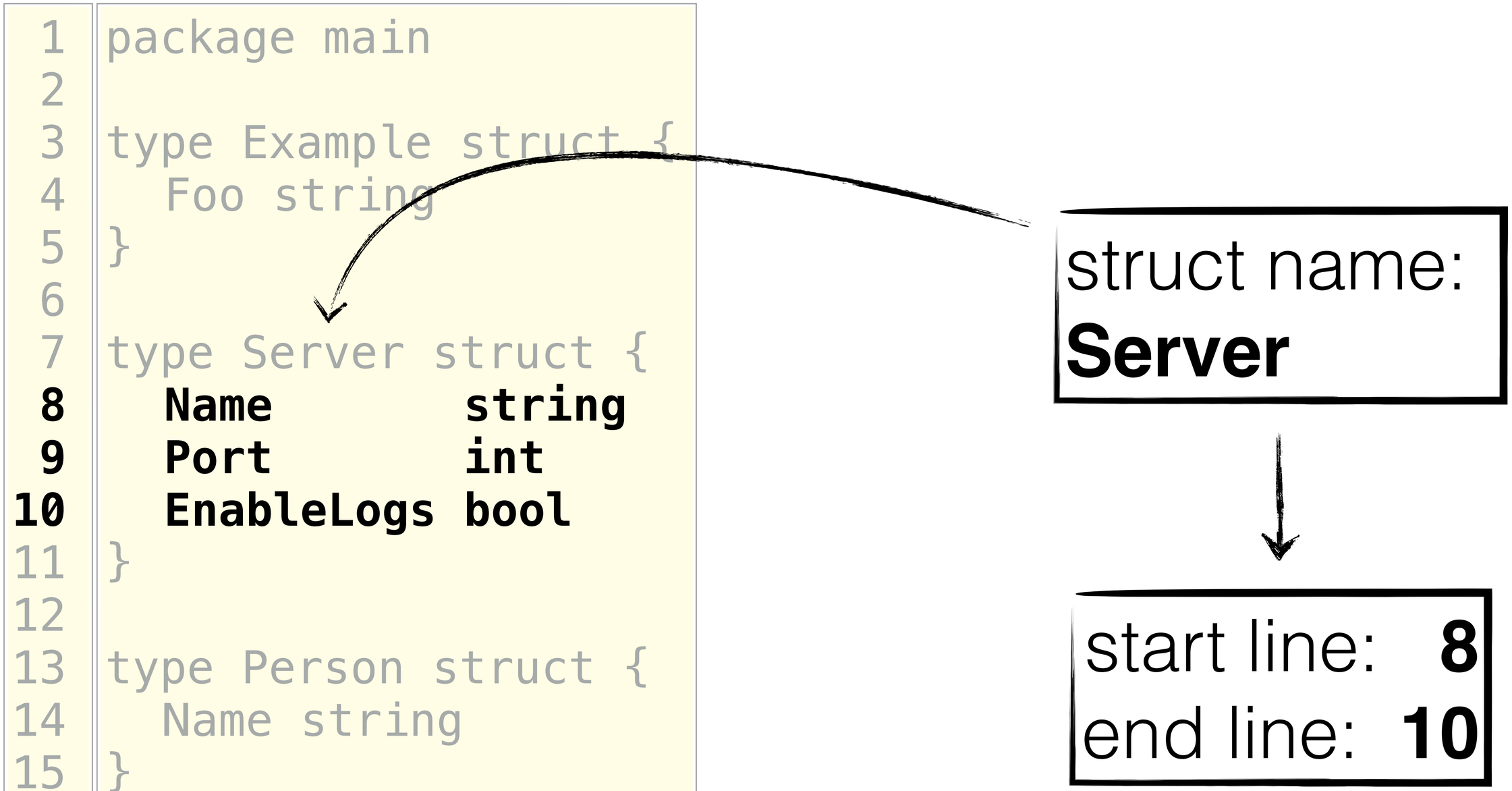
```
 1 package main
 2
 3 type Example struct {
 4   Foo string
 5 }
 6
 7 type Server struct {
 8   Name        string
 9   Port        int
10   EnableLogs bool
11 }
12
13 type Person struct {
14   Name string
15 }
```

```
 1 package main
 2
 3 type Example struct {
 4   Foo string
 5 }
 6
 7 type Server struct {
 8   Name        string
 9   Port        int
10   EnableLogs bool
11 }
12
13 type Person struct {
14   Name string
15 }
```



struct name:

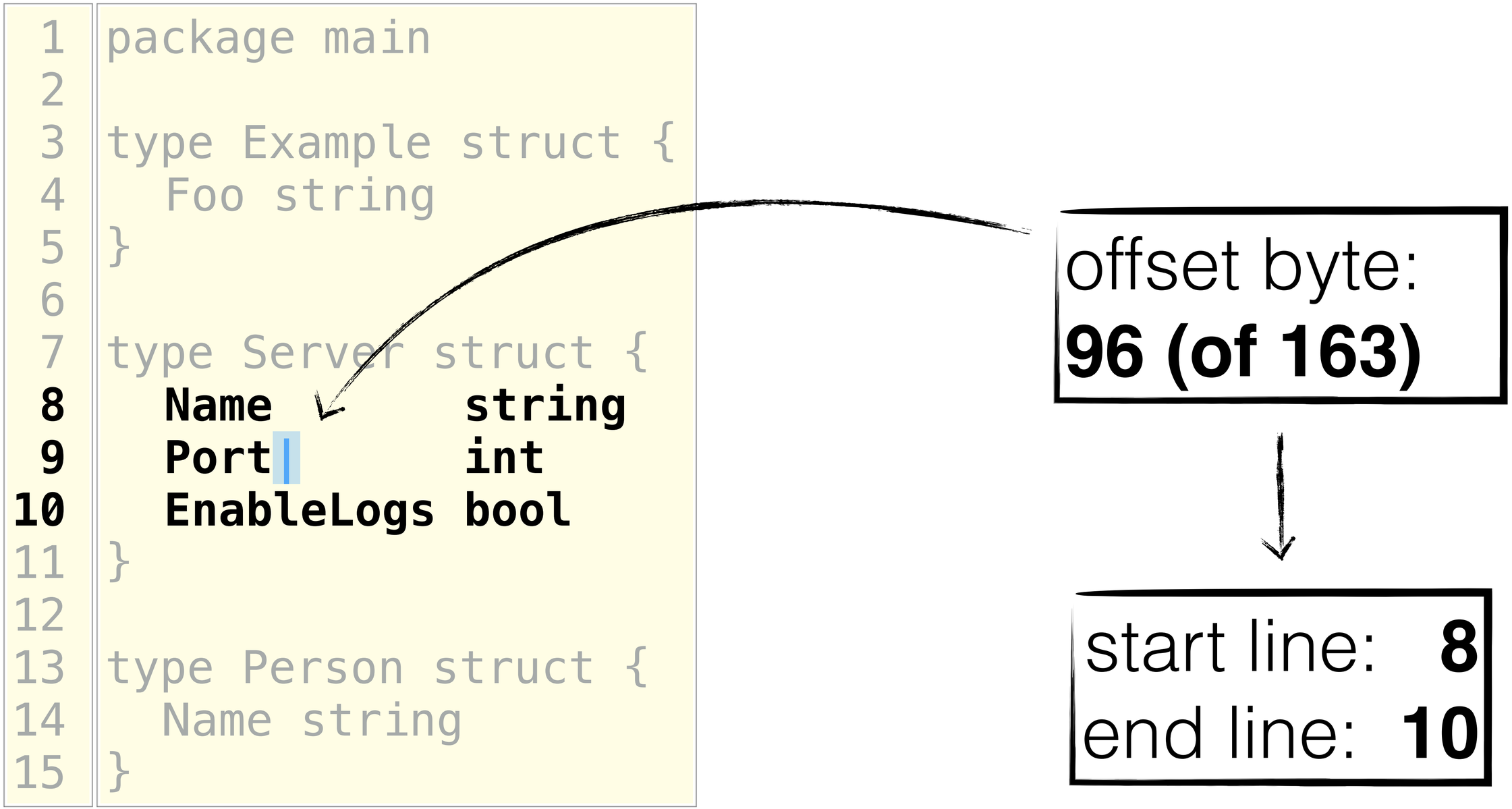
Server


1 package main
2
3 type Example struct {
4 Foo string
5 }
6
7 type Server struct {
8 Name string
9 Port int
10 EnableLogs bool
11 }
12
13 type Person struct {
14 Name string
15 }
struct name:
Server
start line: 8
end line: 10

```
 1 package main
 2
 3 type Example struct {
 4   Foo string
 5 }
 6
 7 type Server struct {
 8   Name        string
 9   Port        int
10   EnableLogs bool
11 }
12
13 type Person struct {
14   Name string
15 }
```

offset byte:

96 (of 163)


1 package main
2
3 type Example struct {
4 Foo string
5 }
6
7 type Server struct {
8 Name string
9 Port int
10 EnableLogs bool
11 }
12
13 type Person struct {
14 Name string
15 }
offset byte:
96 (of 163)
start line: 8
end line: 10

```
 1 package main
 2
 3 type Example struct {
 4   Foo string
 5 }
 6
 7 type Server struct {
 8   Name        string
 9   Port        int
10   EnableLogs bool
11 }
12
13 type Person struct {
14   Name string
15 }
```

.. or **specify explicit** lines

start line: **8**
end line: **10**

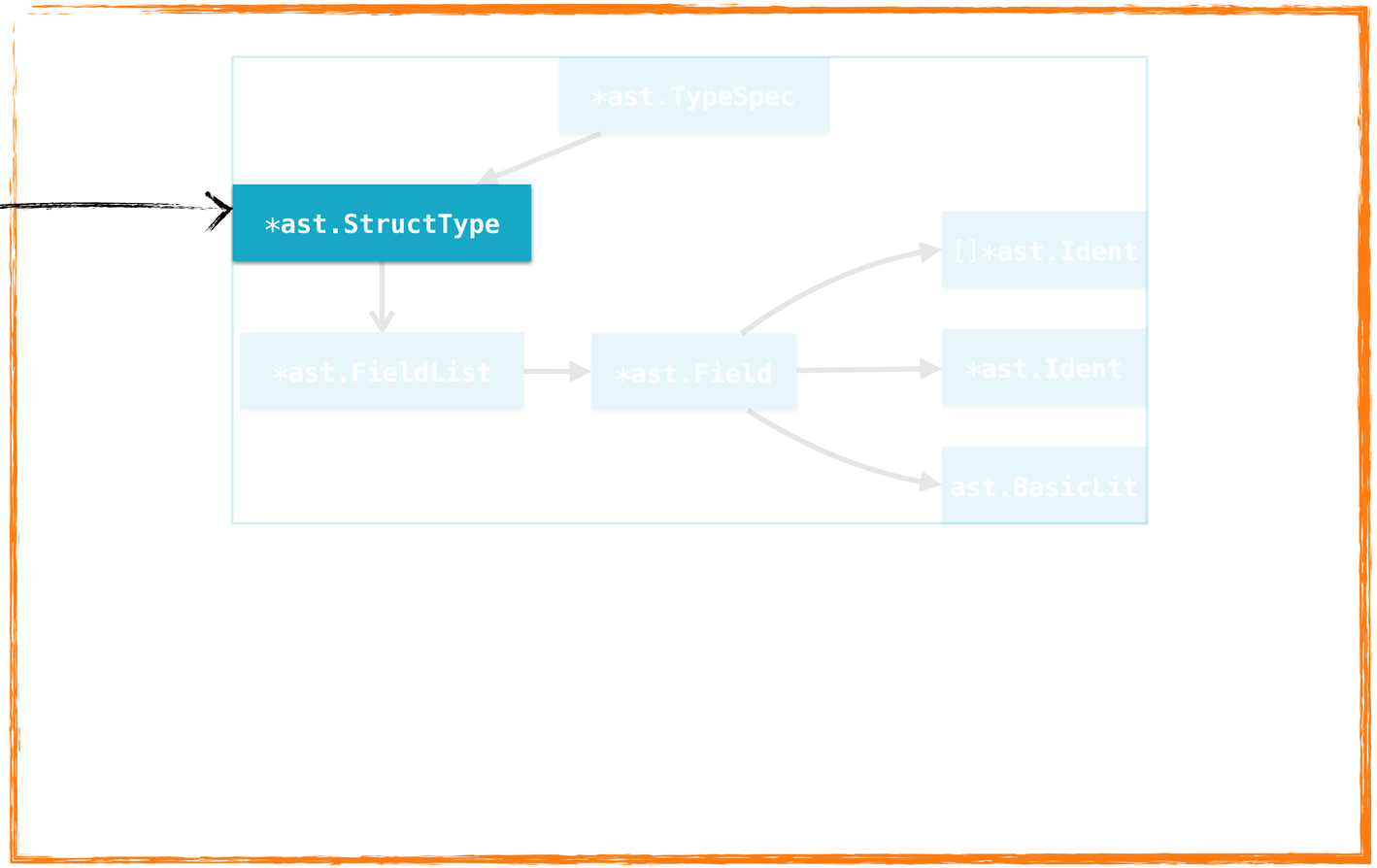
```
 1 package main
 2
 3 type Example struct {
 4   Foo string
 5 }
 6
 7 type Server struct {
 8   Name        string
 9   Port        int
10   EnableLogs bool
11 }
12
13 type Person struct {
14   Name string
15 }
```

.. or **specify explicit** lines

start line: **9**
end line: **9**

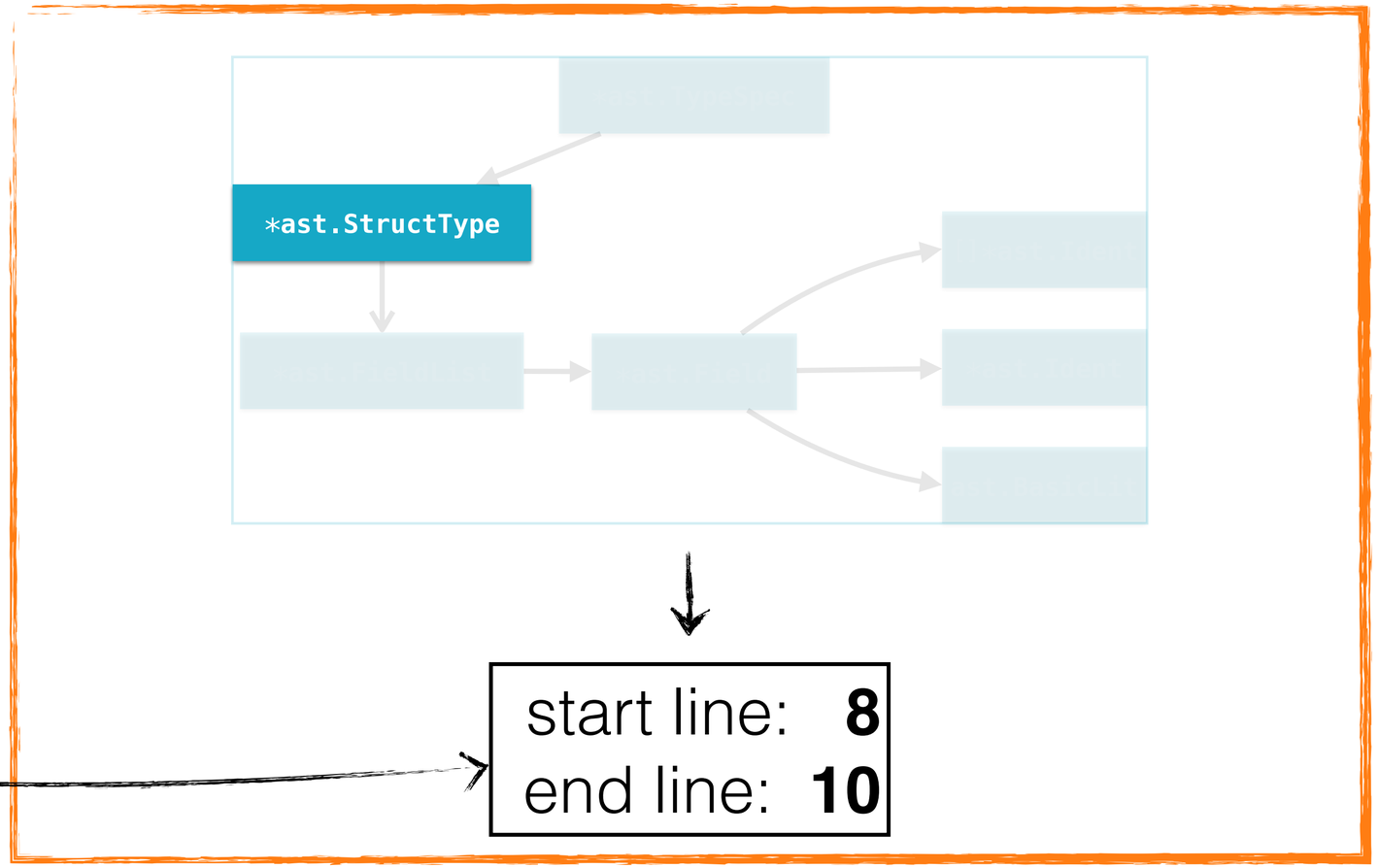
gomodifytags

select **struct** that **matches** our criteria ...



gomodifytags

... and get the
start and **end** lines



Find selection

```
func (c *config) findSelection(node ast.Node) (int, int, error) {
   if c.line != "" {
      return c.lineSelection(node)
   } else if c.offset != 0 {
      return c.offsetSelection(node)
   } else if c.structName != "" {
      return c.structSelection(node)
   }

   return 0, 0, errors.New("-line, -offset or -struct is not passed")
}
```

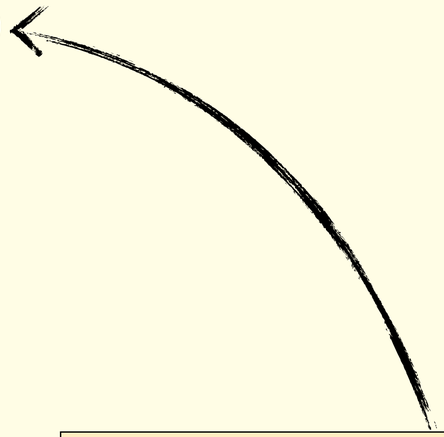
Find selection

```
func (c *config) findSelection(node ast.Node) (int, int, error) {
   if c.line != "" {
      return c.lineSelection(node)
   } else if c.offset != 0 {
      return c.offsetSelection(node)
   } else if c.structName != "" {
      return c.structSelection(node)
   }

   return 0, 0, errors.New("-line, -offset or -struct is not passed")
}
```

Collecting structs

```
// collectStructs collects and maps structType nodes to their positions
func collectStructs(node ast.Node) map[token.Pos]*structType {
   structs := make(map[token.Pos]*structType, 0)
   collectStructs := func(n ast.Node) bool {
      t, ok := n.(*ast.TypeSpec)
      if !ok {
         return true
      }

      structName := t.Name.Name

      x, ok := t.Type.(*ast.StructType)
      if !ok {
         return true
      }

      structs[x.Pos()] = &structType{
         name: structName,
         node: x,
      }
      return true
   }
   ast.Inspect(node, collectStructs)
   return structs
}
```

```go
// collectStructs collects and maps structType nodes to their positions
func collectStructs(node ast.Node) map[token.Pos]*structType {
   structs := make(map[token.Pos]*structType)
   collectStructs := func(n ast.Node) bool {
      t, ok := n.(*ast.TypeSpec)
      if !ok {
         return true
      }

      structName := t.Name.Name

      x, ok := t.Type.(*ast.StructType)
      if !ok {
         return true
      }

      structs[x.Pos()] = &structType{
         name: structName,
         node: x,
      }
      return true
   }
   ast.Inspect(node, collectStructs)
   return structs
}
```



```go
type structType struct {
   name string
   node *ast.StructType
}
```

```
// collectStructs collects and maps structType nodes to their positions
func collectStructs(node ast.Node) map[token.Pos]*structType {
   structs := make(map[token.Pos]*structType)
   collectStructs := func(n ast.Node) bool {
      t, ok := n.(*ast.TypeSpec)
      if !ok {
         return true
      }

      structName := t.Name.Name

      x, ok := t.Type.(*ast.StructType)
      if !ok {
         return true
      }

      structs[x.Pos()] = &structType{
         name: structName,
         node: x,
      }
      return true
   }
   ast.Inspect(node, collectStructs)
   return structs
}
```

```
// collectStructs collects and maps structType nodes to their positions
func collectStructs(node ast.Node) map[token.Pos]*structType {
   structs := make(map[token.Pos]*structType)
   collectStructs := func(n ast.Node) bool {
      t, ok := n.(*ast.TypeSpec)
      if !ok {
         return true
      }

      structName := t.Name.Name

      x, ok := t.Type.(*ast.StructType)
      if !ok {
         return true
      }

      structs[x.Pos()] = &structType{
         name: structName,
         node: x,
      }
      return true
   }
   ast.Inspect(node, collectStructs)
   return structs
}
```

```
type structType struct {
   name string
   node *ast.StructType
}
```

```
// collectStructs collects and maps structType nodes to their positions
func collectStructs(node ast.Node) map[token.Pos]*structType {
   structs := make(map[token.Pos]*structType)
   collectStructs := func(n ast.Node) bool {
      t, ok := n.(*ast.TypeSpec)
      if !ok {
         return true
      }

      structName := t.Name.Name

      x, ok := t.Type.(*ast.StructType)
      if !ok {
         return true
      }

      structs[x.Pos()] = &structType{
         name: structName,
         node: x,
      }
      return true
   }
   ast.Inspect(node, collectStructs)
   return structs
}
```

Struct selection

```
var encStruct *ast.StructType
for _, st := range collectStructs() {
  if st.name == c.structName {
    encStruct = st.node
  }

  start = c.fset.Position(encStruct.Pos()).Line
  end = c.fset.Position(encStruct.End()).Line
}
```

Struct selection

```
var encStruct *ast.StructType
for _, st := range collectStructs() {
  if st.name == c.structName {
    encStruct = st.node
  }

  start = c.fset.Position(encStruct.Pos()).Line
  end = c.fset.Position(encStruct.End()).Line
}
```

Struct selection

```
var encStruct *ast.StructType
for _, st := range collectStructs() {
  if st.name == c.structName {
    encStruct = st.node
  }

  start = c.fset.Position(encStruct.Pos()).Line
  end = c.fset.Position(encStruct.End()).Line
}
```

Struct selection

```
var encStruct *ast.StructType
for _, st := range collectStructs() {
  if st.name == c.structName {
    encStruct = st.node
  }

  start = c.fset.Position(encStruct.Pos()).Line
  end = c.fset.Position(encStruct.End()).Line
}
```

Offset selection

```
var encStruct *ast.StructType
for _, st := range collectStructs() {
  structBegin := c.fset.Position(st.node.Pos()).Offset
  structEnd := c.fset.Position(st.node.End()).Offset

  if structBegin <= c.offset && c.offset <= structEnd {
    encStruct = st.node
    break
  }
}

start = c.fset.Position(encStruct.Pos()).Line
end = c.fset.Position(encStruct.End()).Line
```

Offset selection

```
var encStruct *ast.StructType
for _, st := range collectStructs() {
  structBegin := c.fset.Position(st.node.Pos()).Offset
  structEnd := c.fset.Position(st.node.End()).Offset

  if structBegin <= c.offset && c.offset <= structEnd {
    encStruct = st.node
    break
  }
}

start = c.fset.Position(encStruct.Pos()).Line
end = c.fset.Position(encStruct.End()).Line
```

Offset selection

```
var encStruct *ast.StructType
for _, st := range collectStructs() {
  structBegin := c.fset.Position(st.node.Pos()).Offset
  structEnd := c.fset.Position(st.node.End()).Offset

  if structBegin <= c.offset && c.offset <= structEnd {
    encStruct = st.node
    break
  }
}

start = c.fset.Position(encStruct.Pos()).Line
end = c.fset.Position(encStruct.End()).Line
```

Offset selection

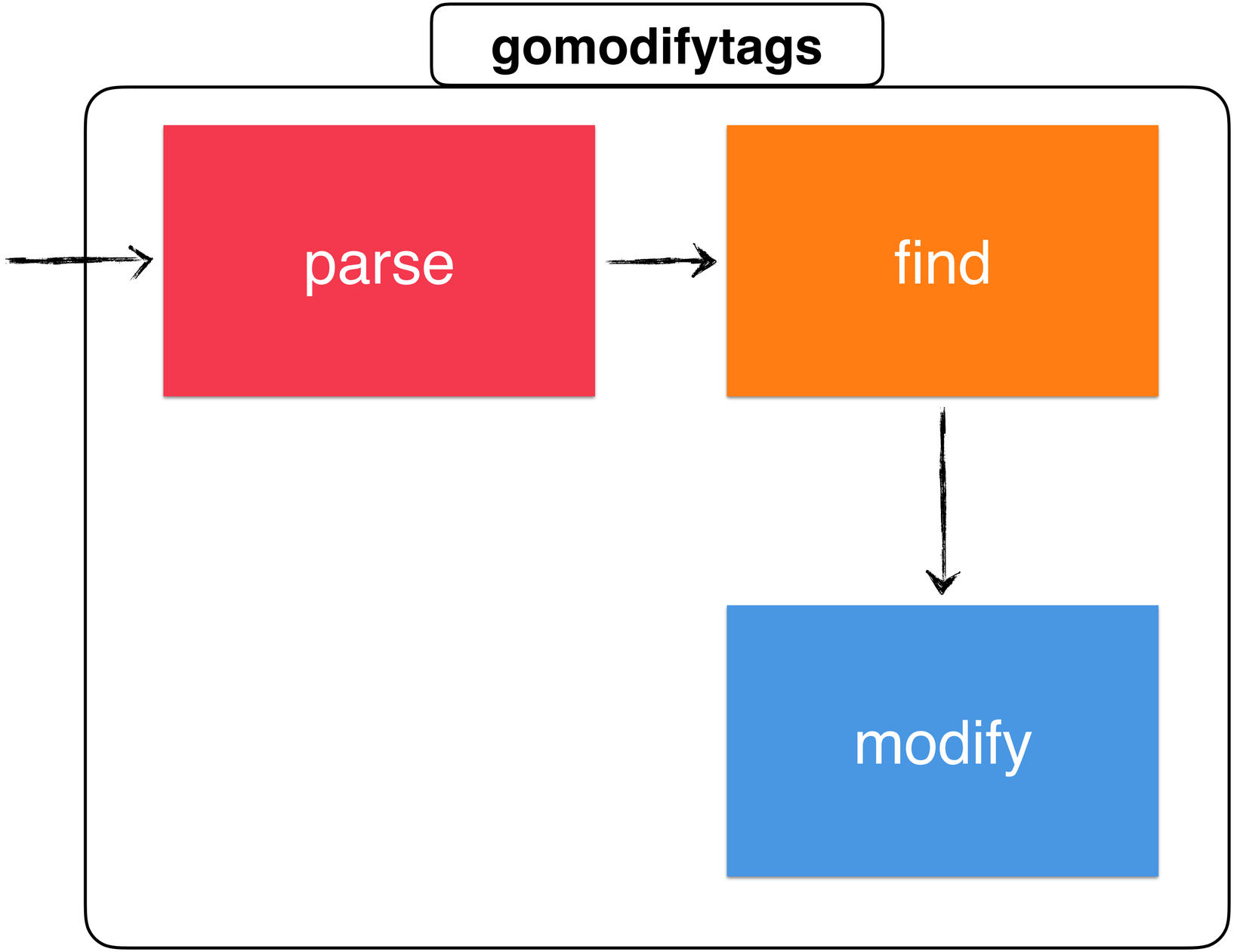
```
var encStruct *ast.StructType
for _, st := range collectStructs() {
  structBegin := c.fset.Position(st.node.Pos()).Offset
  structEnd := c.fset.Position(st.node.End()).Offset

  if structBegin <= c.offset && c.offset <= structEnd {
    encStruct = st.node
    break
  }
}

start = c.fset.Position(encStruct.Pos()).Line
end = c.fset.Position(encStruct.End()).Line
```

Offset selection

```
var encStruct *ast.StructType
for _, st := range collectStructs() {
  structBegin := c.fset.Position(st.node.Pos()).Offset
  structEnd := c.fset.Position(st.node.End()).Offset

  if structBegin <= c.offset && c.offset <= structEnd {
    encStruct = st.node
    break
  }
}

start = c.fset.Position(encStruct.Pos()).Line
end = c.fset.Position(encStruct.End()).Line
```

gomodifytags

```
package main

type Example struct {
    Foo string
}
```



Rewrite the struct tag

1. **Fetch** configuration settings
2. **Parse** content
3. **Find** selection
4. **Modify** the struct tag
5. **Output** the result

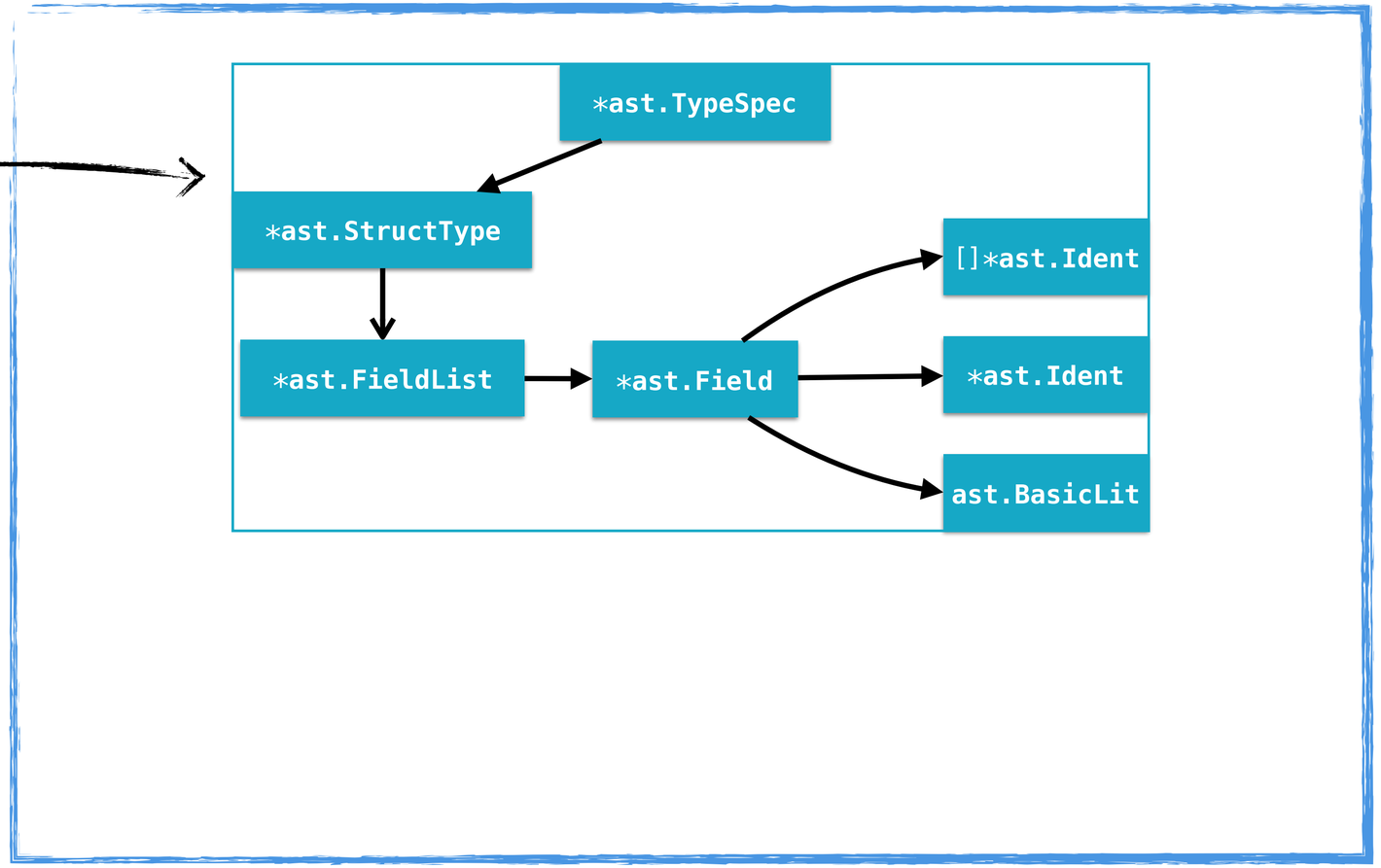
```
func main() {
   var cfg config

   node = cfg.parse()

   start, end = cfg.findSelection(node)

   rewritten = cfg.rewrite(node, start, end)

   out = cfg.format(rewritten)

   fmt.Println(out)
}
```

start line: **8**
end line: **10**

start and **end** lines
we're going to **modify**

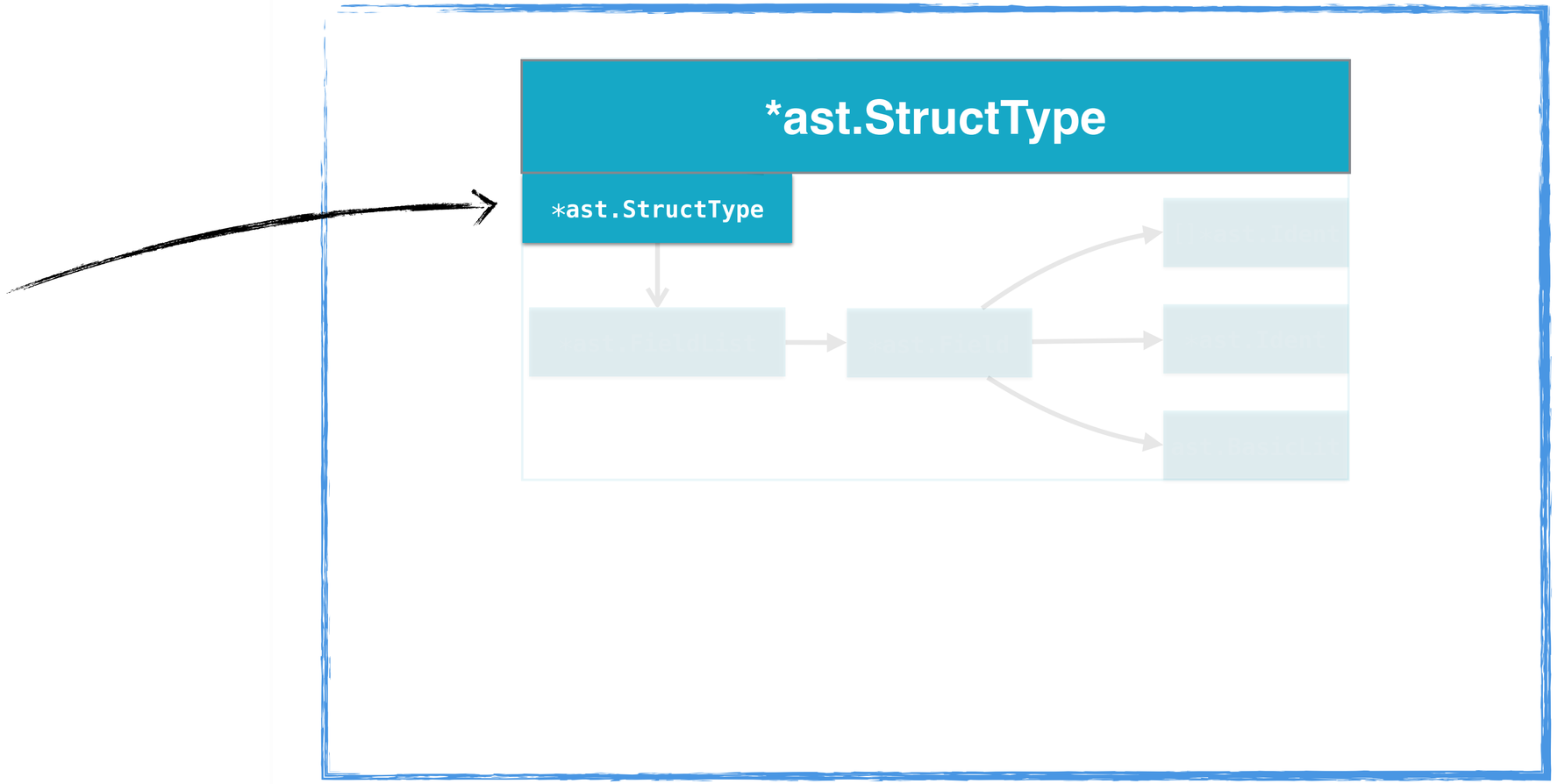
start line: **8**
end line: **10**

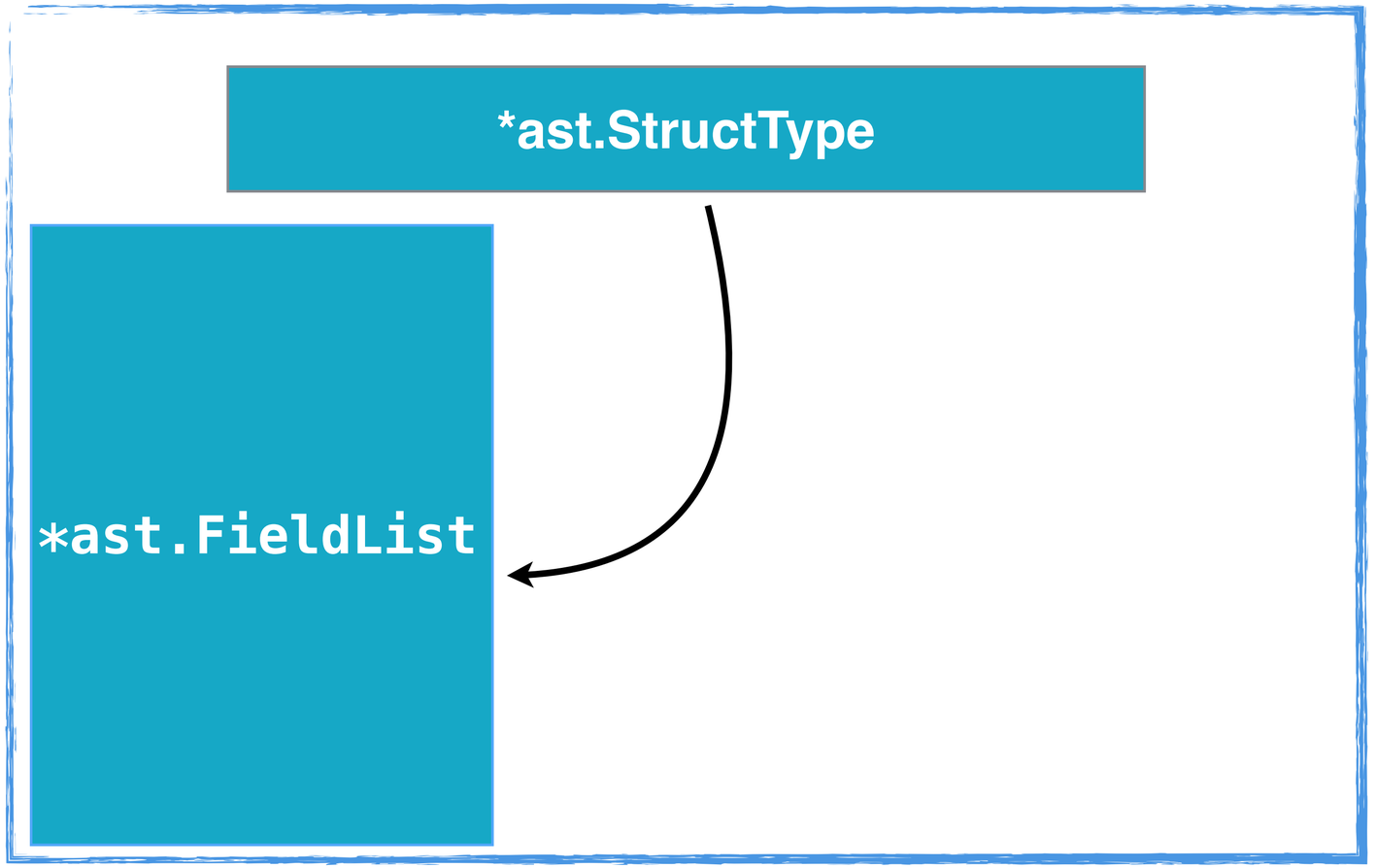
our parsed **AST**

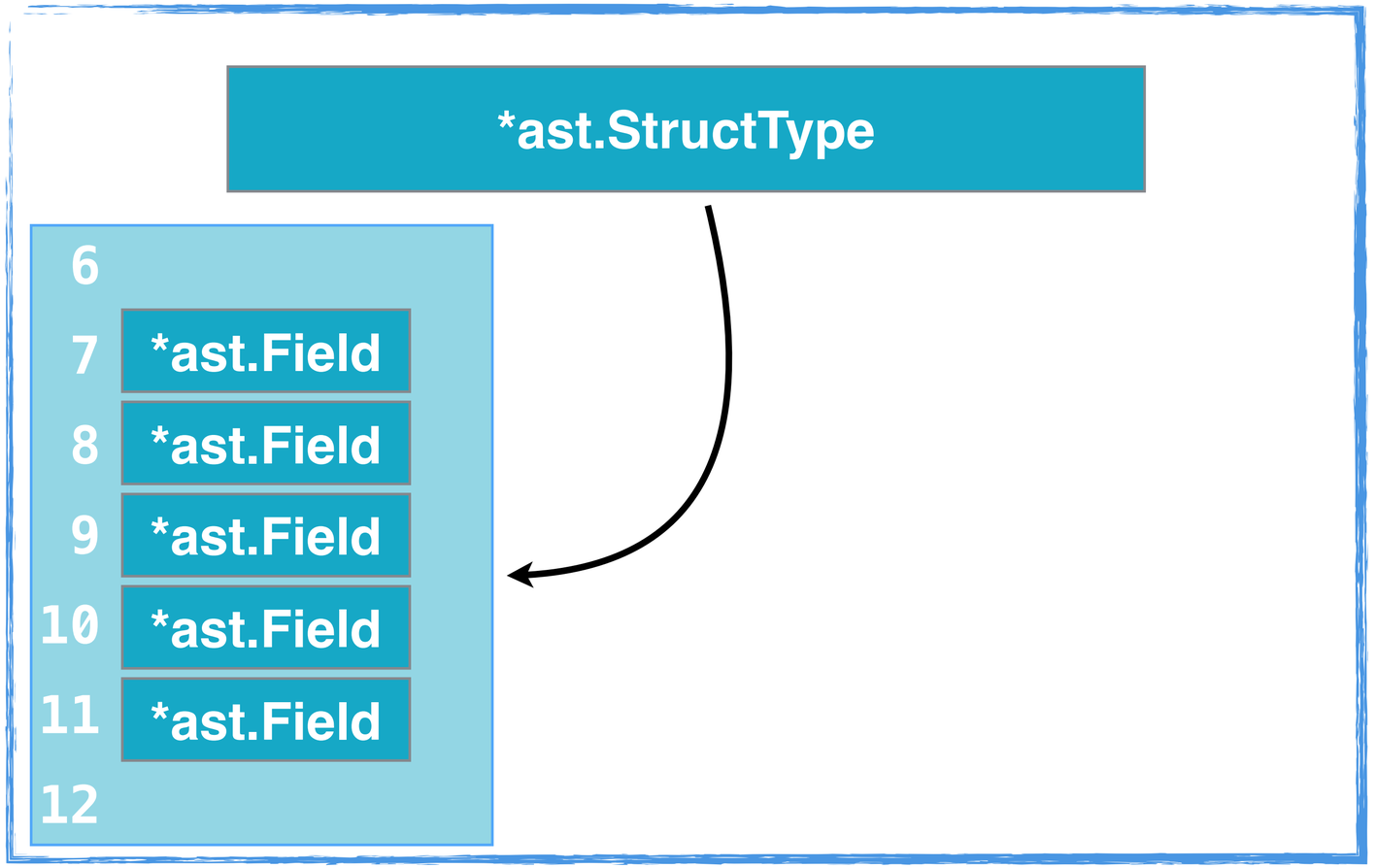


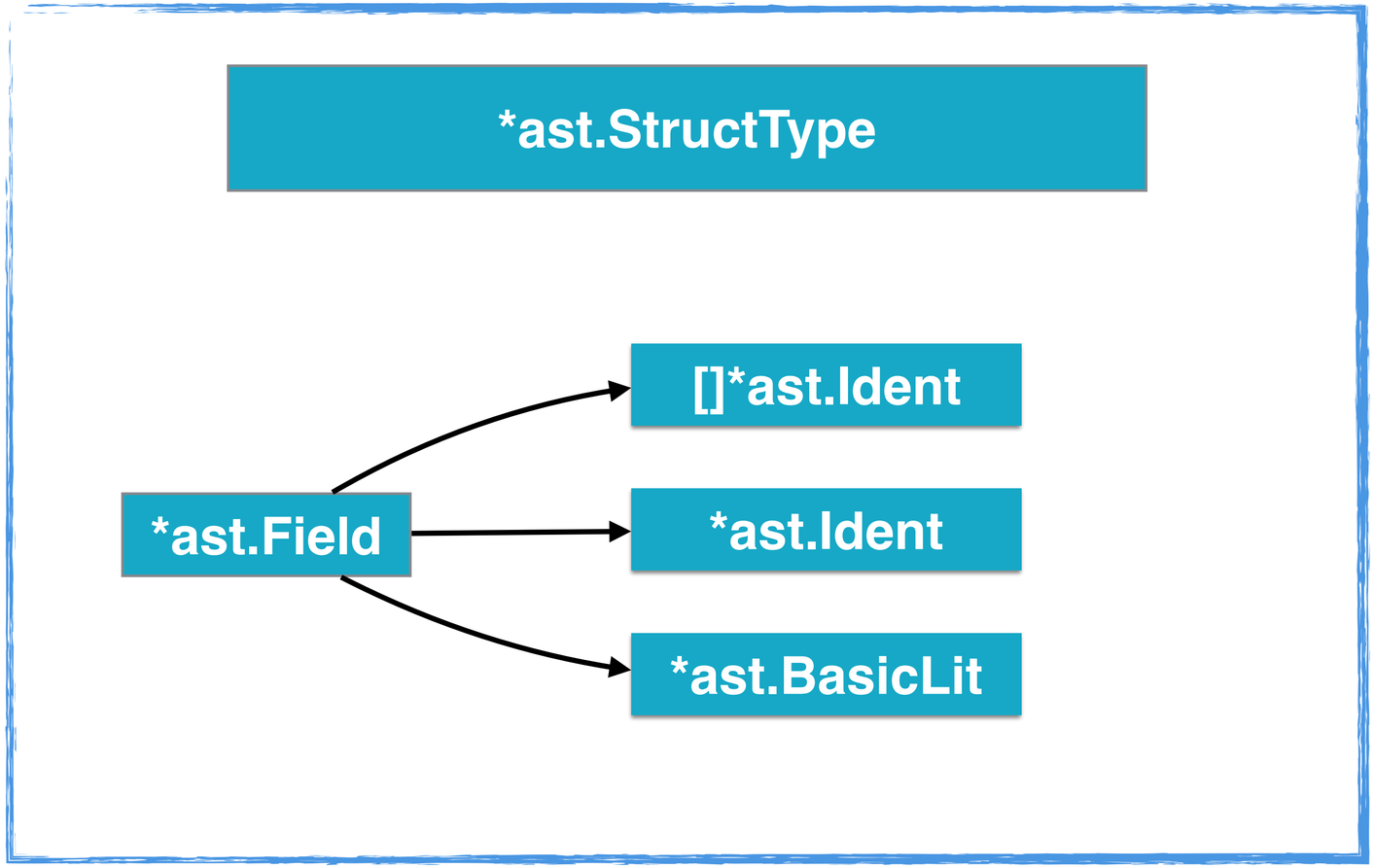
start line: **8**
end line: **10**

struct we're going to modify

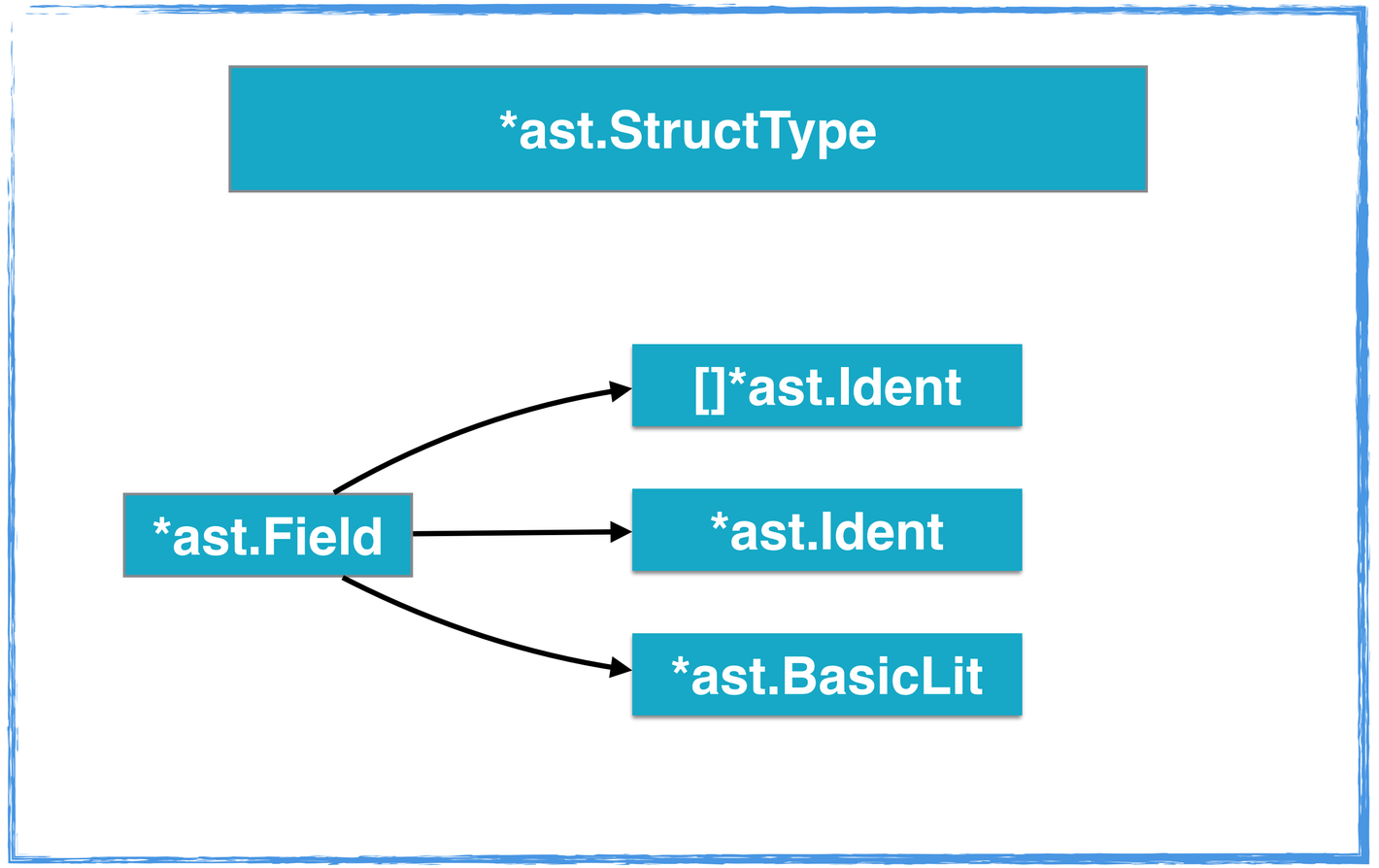



*ast.StructType
*ast.FieldList

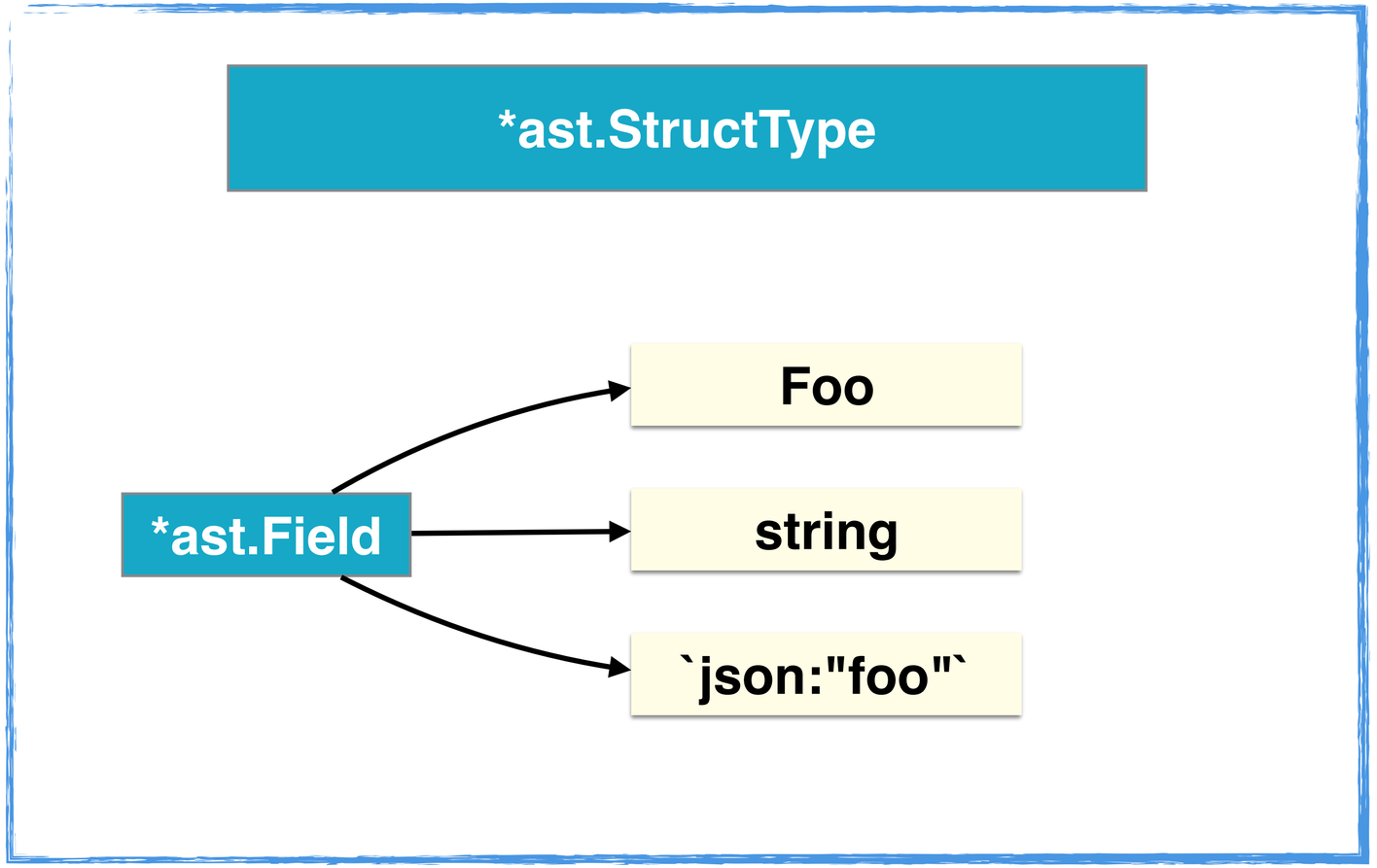

*ast.StructType
6
7
*ast.Field
8
*ast.Field
9
*ast.Field
10
*ast.Field
11
*ast.Field
12


*ast.StructType
[]*ast.Ident
*ast.Field
*ast.Ident
*ast.BasicLit

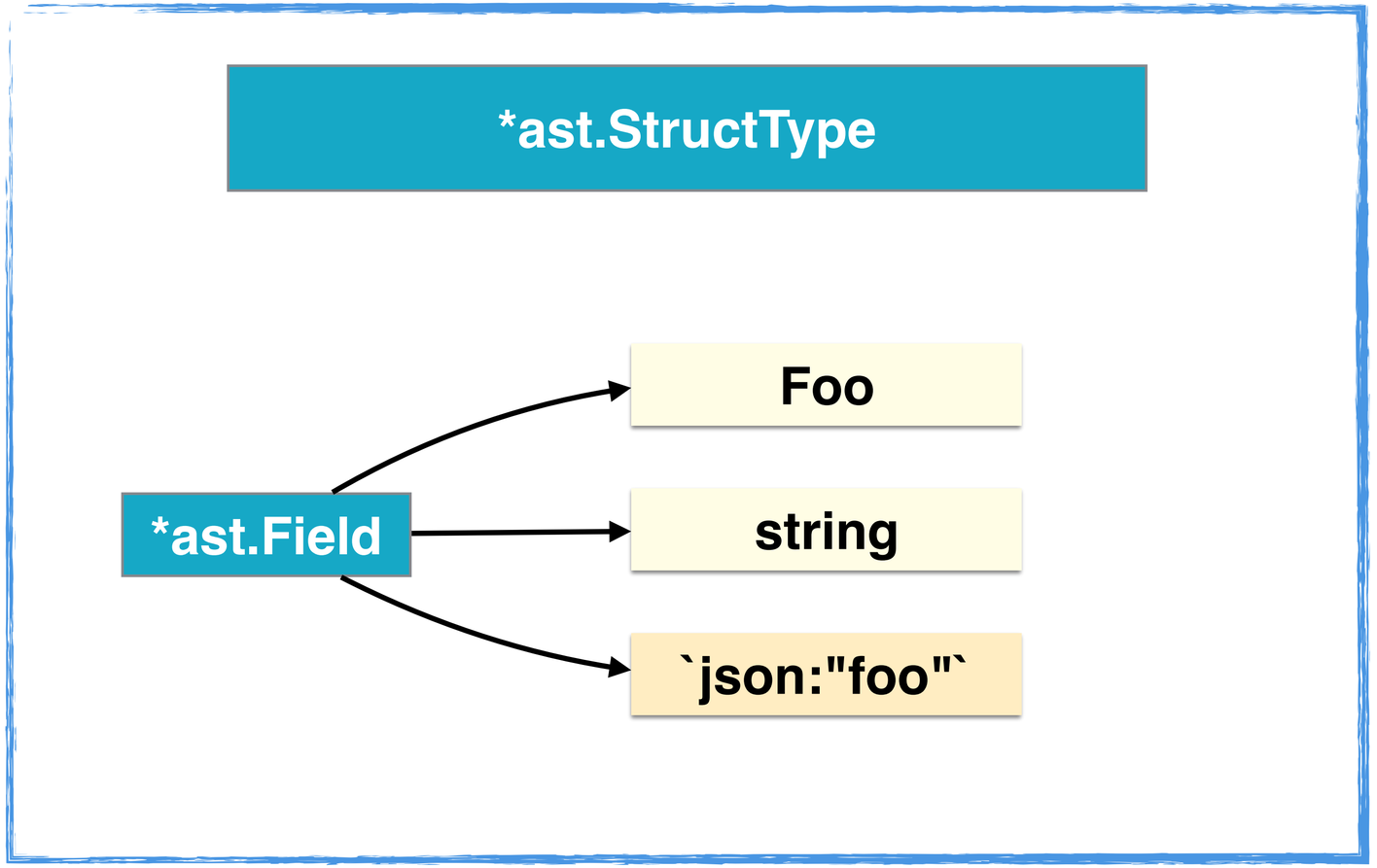
```
type Example struct {
    Foo string `json:"foo"`
}
```



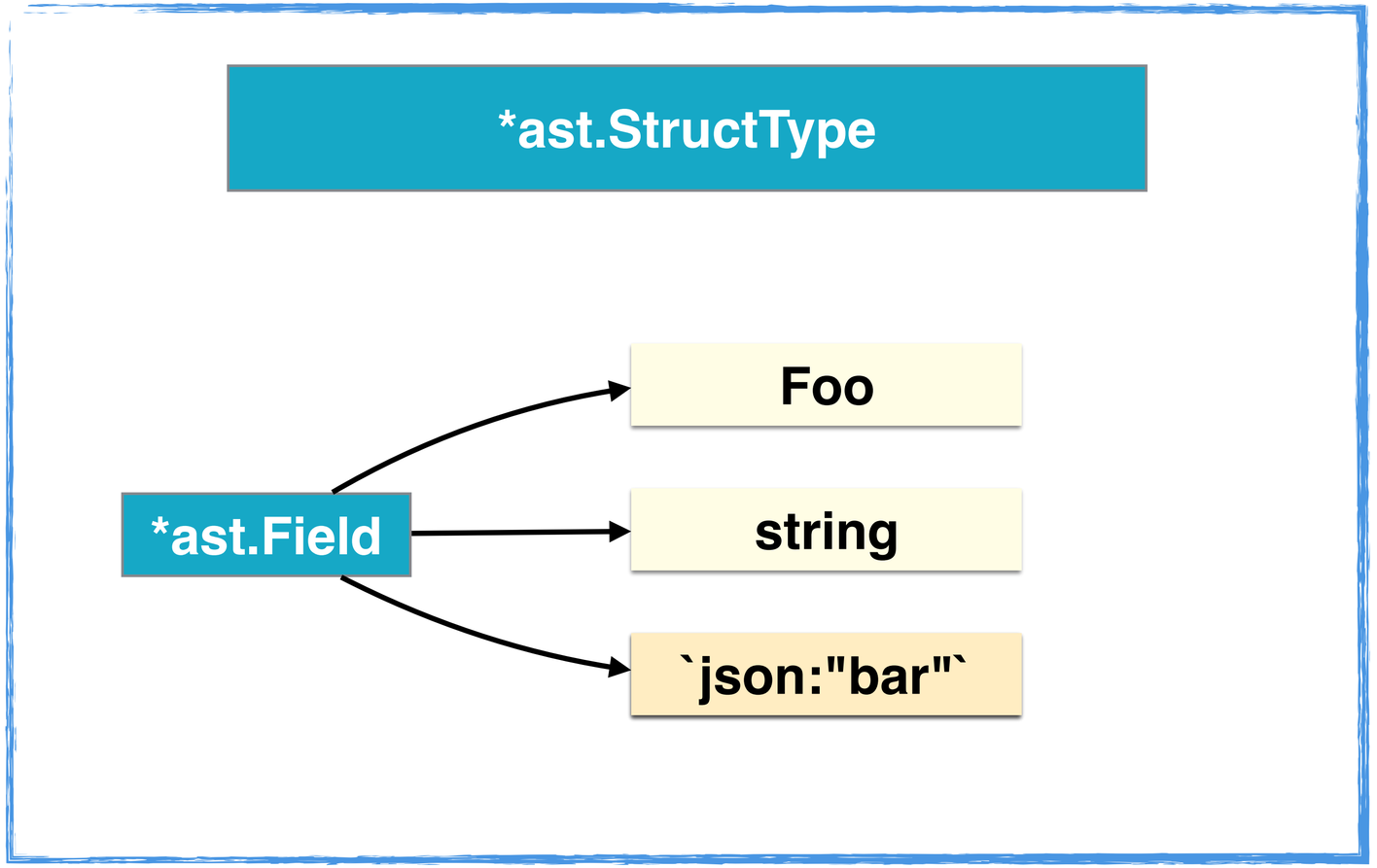
```
type Example struct {
    Foo string `json:"foo"
}
```



```
type Example struct {
    Foo string `json:"foo"`
}
```



```
type Example struct {
    Foo string `json:"foo"`
}
```



How do we **rewrite** a struct tag?

```
src := `package main
type Example struct {
   Foo string` + " `json:\"foo\"` }"

fset := token.NewFileSet()
file, err := parser.ParseFile(fset, "demo", src, parser.ParseComments)
if err != nil {
   panic(err)
}

ast.Inspect(file, func(x ast.Node) bool {
   s, ok := x.(*ast.StructType)
   if !ok {
      return true
   }

   for _, field := range s.Fields.List {
      // found field!
      field.Tag.Value = `json:"bar"`
   }

   return false
})
```

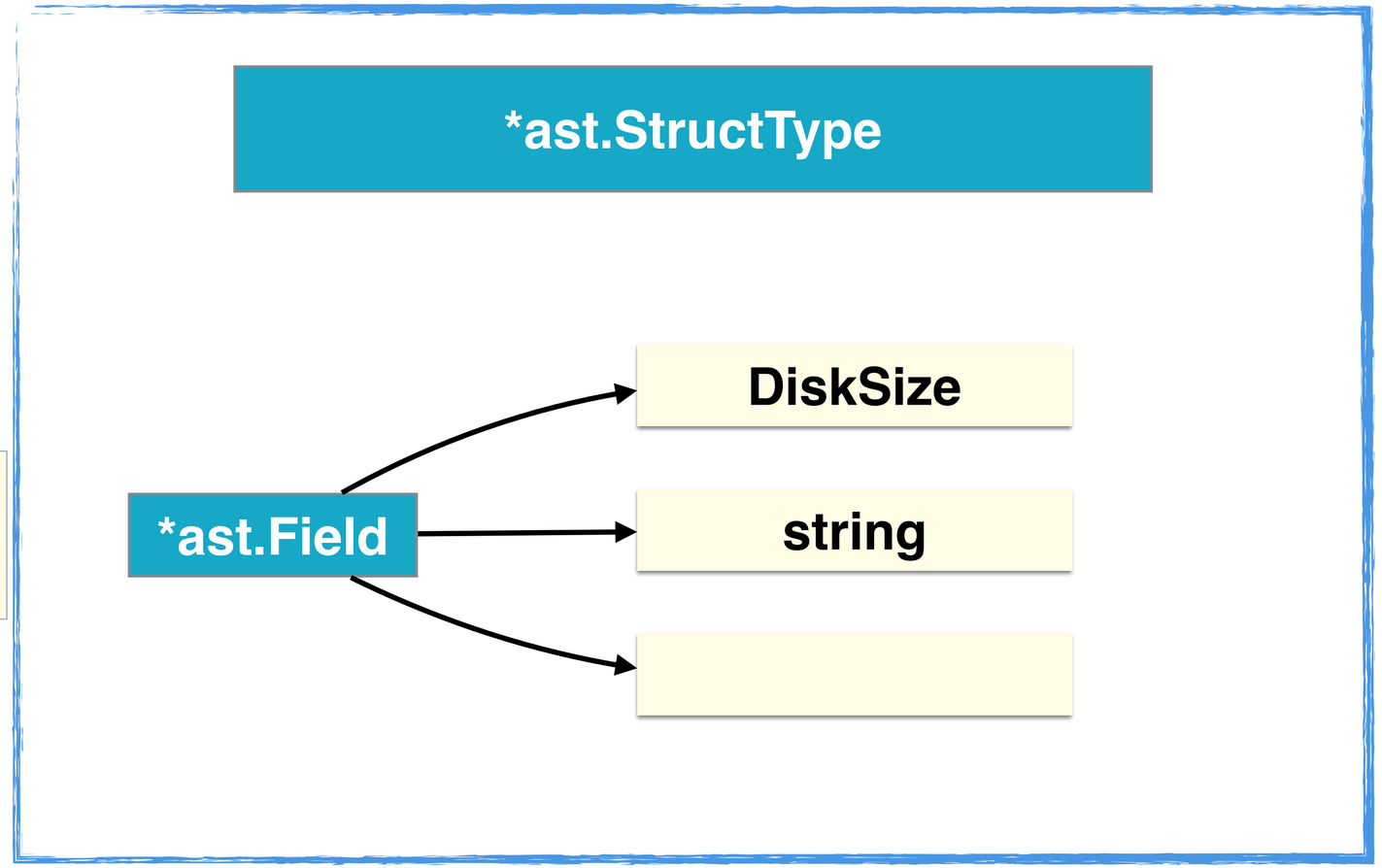
After **finding** a field,

```
src := `package main
type Example struct {
   Foo string` + " `json:\"foo\"` }"

fset := token.NewFileSet()
file, err := parser.ParseFile(fset, "demo", src, parser.ParseComments)
if err != nil {
   panic(err)
}

ast.Inspect(file, func(x ast.Node) bool {
   s, ok := x.(*ast.StructType)
   if !ok {
      return true
   }

   for _, field := range s.Fields.List {
      // found field!
      field.Tag.Value = `json:"bar"`
   }

   return false
})
```

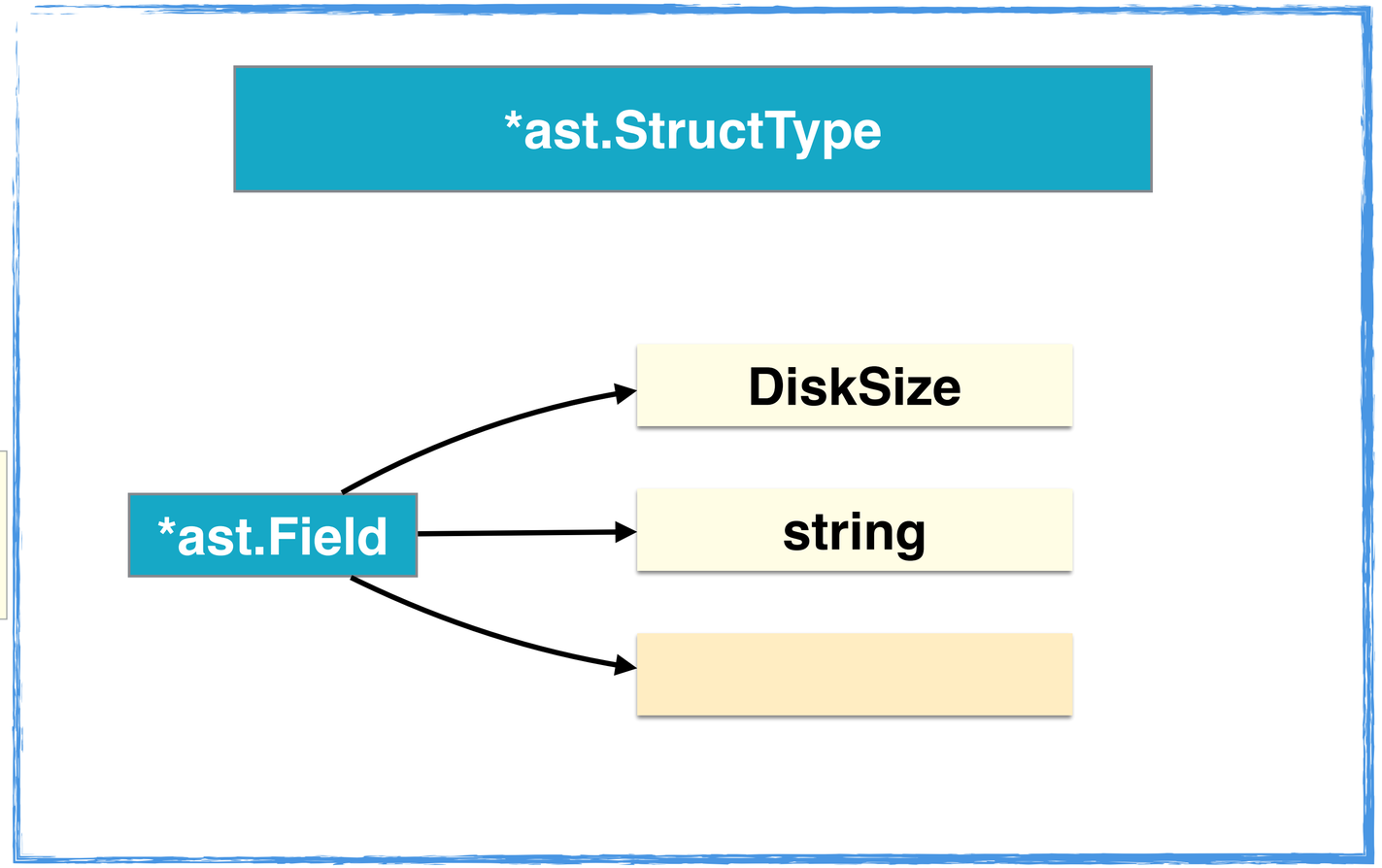
After **finding** a field, **replace** the value

~~Tag: `json:"foo"`~~
Tag: `json:"bar"`

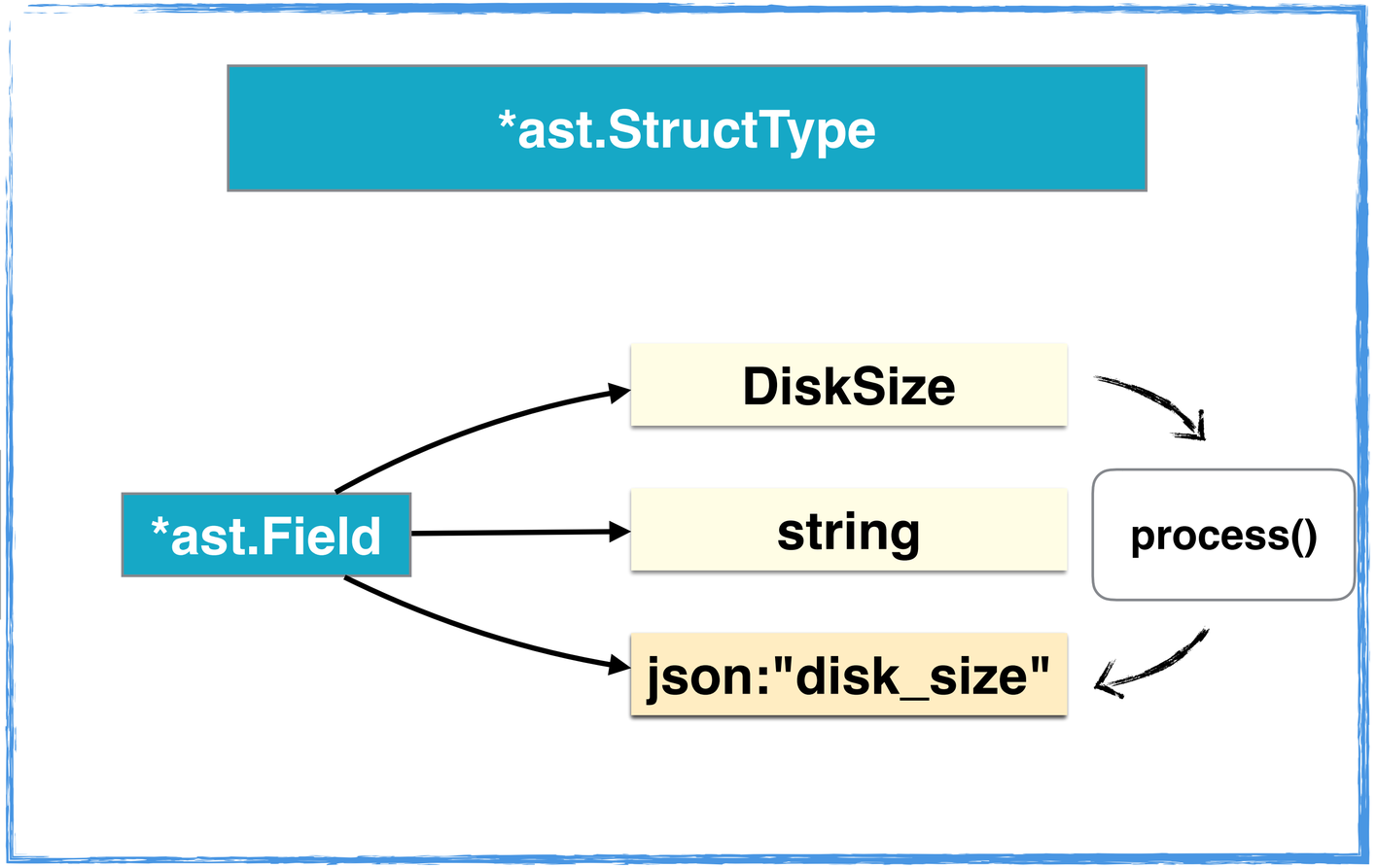
```
type Example struct {
  DiskSize string
}
```



```
type Example struct {
  DiskSize string
}
```



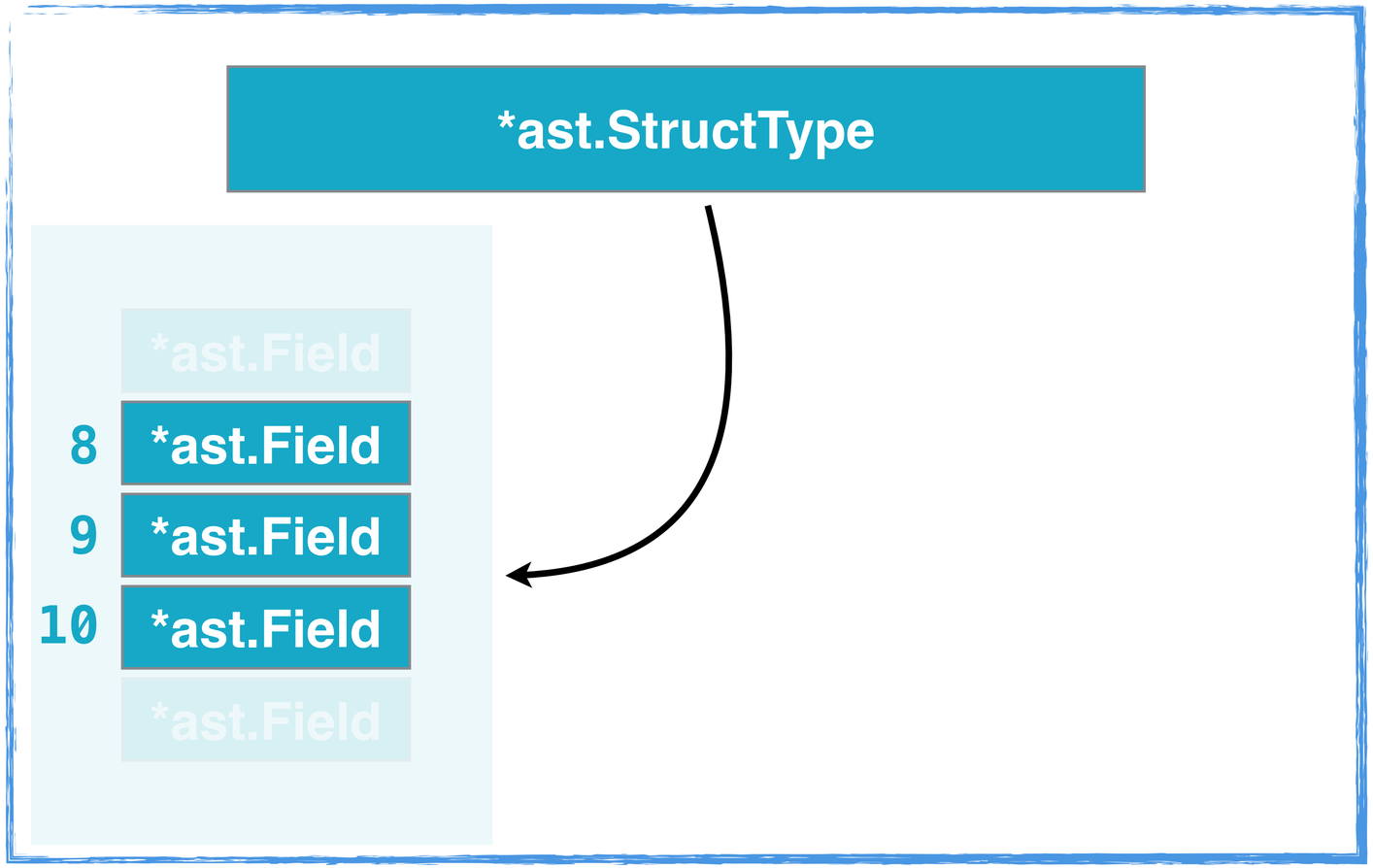
```
type Example struct {
  DiskSize string `json:"disk_size"`
}
```



```
src := `package main
type Example struct {
   Foo string` + " `json:\"foo\"` }"

fset := token.NewFileSet()
file, err := parser.ParseFile(fset, "demo",
if err != nil {
   panic(err)
}

ast.Inspect(file, func(x ast.Node) bool {
   s, ok := x.(*ast.StructType)
   if !ok {
      return true
   }

   for _, field := range s.Fields.List {
      // found field!
      field.Tag.Value = process(field)
   }

   return false
})
```

```
tags = c.removeTags(tags)
tags, err = c.removeTagOptions(tags)
if err != nil {
   return "", err
}

tags = c.clearTags(tags)
tags = c.clearOptions(tags)

tags, err = c.addTags(fieldName, tags)
if err != nil {
   return "", err
}

tags, err = c.addTagOptions(tags)
if err != nil {
   return "", err
}

if c.sort {
   sort.Sort(tags)
}
```

Modify overview

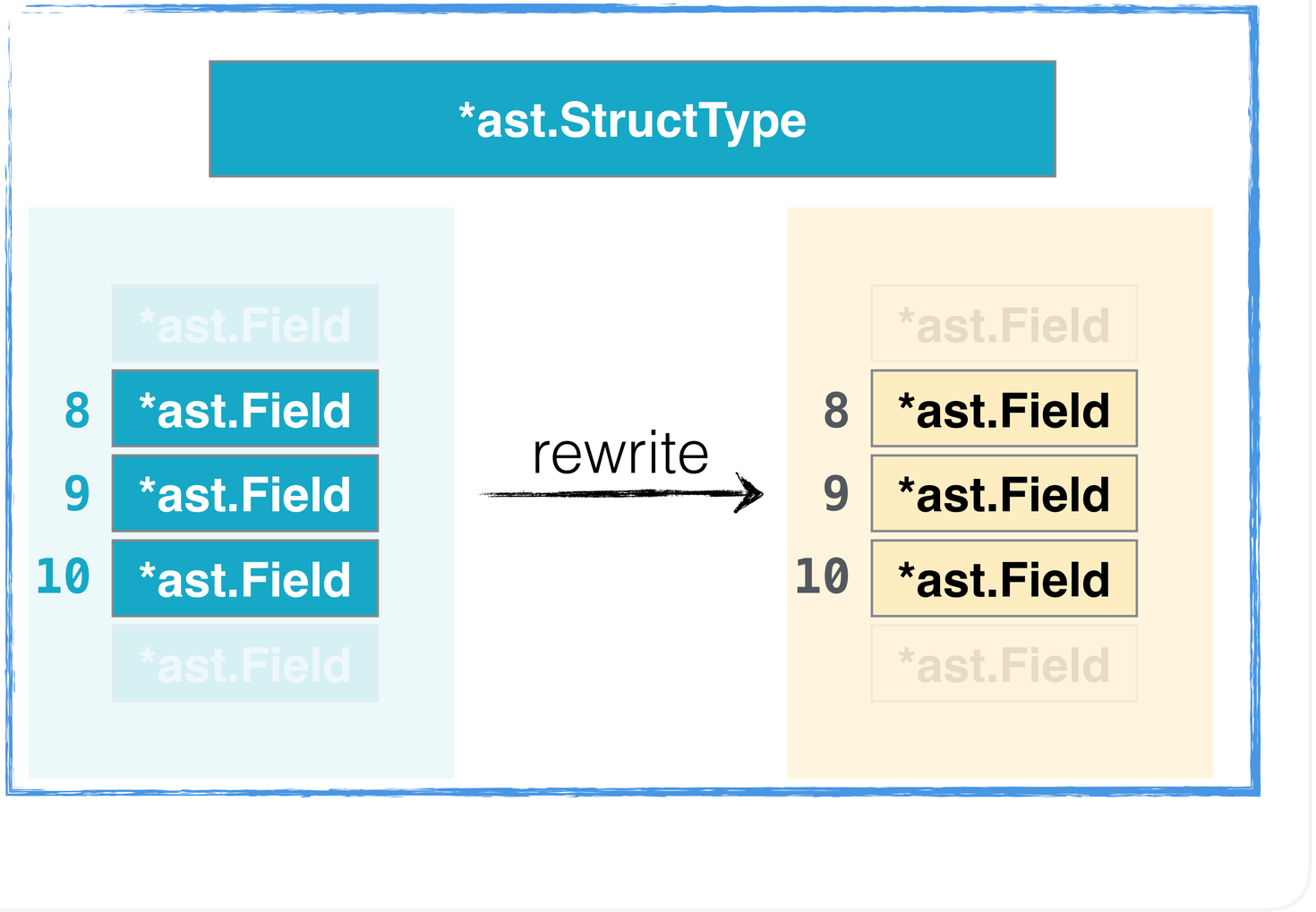


select field between
start and **end** lines ...

gomodifytags

Modify overview

... and **rewrite** selected fields **tags**



gomodifytags

```
func (c *config) rewriteFields(node ast.Node) (ast.Node, error) {
   var rewriteErr error
   rewriteFunc := func(n ast.Node) bool {
      x, ok := n.(*ast.StructType)
      if !ok {
         return true
      }

      for _, f := range x.Fields.List {
         // process each field
         // ...
      }

      return true
   }

   ast.Inspect(node, rewriteFunc)
   return node, rewriteErr
}
```

```
func (c *config) rewriteFields(node ast.Node) (ast.Node, error) {
   var rewriteErr error
   rewriteFunc := func(n ast.Node) bool {
      x, ok := n.(*ast.StructType)
      if !ok {
         return true
      }

      for _, f := range x.Fields.List {
         // process each field
         // ...
      }

      return true
   }

   ast.Inspect(node, rewriteFunc)
   return node, rewriteErr
}
```

```
func (c *config) rewriteFields(node ast.Node) (ast.Node, error) {
   var rewriteErr error
   rewriteFunc := func(n ast.Node) bool {
      x, ok := n.(*ast.StructType)
      if !ok {
         return true
      }

      for _, f := range x.Fields.List {
         // process each field
         // ...
      }

      return true
   }

   ast.Inspect(node, rewriteFunc)
   return node, rewriteErr
}
```

```
for _, f := range x.Fields.List {
   line := c.fset.Position(f.Pos()).Line

   if !(c.start <= line && line <= c.end) {
      continue
   }

   if f.Tag == nil {
      f.Tag = &ast.BasicLit{}
   }

   // ...

}
```

```
for _, f := range x.Fields.List {
   line := c.fset.Position(f.Pos()).Line

   if !(c.start <= line && line <= c.end) {
      continue
   }

   if f.Tag == nil {
      f.Tag = &ast.BasicLit{}
   }

   // ...

}
```

```
for _, f := range x.Fields.List {
   line := c.fset.Position(f.Pos()).Line

   if !(c.start <= line && line <= c.end) {
      continue
   }

   if f.Tag == nil {
      f.Tag = &ast.BasicLit{}
   }

   // ...

}
```

```
for _, f := range x.Fields.List {
   fieldName := ""
   if len(f.Names) != 0 {
      fieldName = f.Names[0].Name
   }

   if f.Names == nil {
      ident, ok := f.Type.(*ast.Ident)
      if !ok {
         continue // anonymous field
      }

      fieldName = ident.Name
   }

   res, err := c.process(fieldName, f.Tag.Value)
   if err != nil {
      rewriteErr = err
      return true
   }
   f.Tag.Value = res
}
```

```
for _, f := range x.Fields.List {
   fieldName := ""
   if len(f.Names) != 0 {
      fieldName = f.Names[0].Name
   }

   if f.Names == nil {
      ident, ok := f.Type.(*ast.Ident)
      if !ok {
         continue // anonymous field
      }

      fieldName = ident.Name
   }

   res, err := c.process(fieldName, f.Tag.Value)
   if err != nil {
      rewriteErr = err
      return true
   }
   f.Tag.Value = res
}
```

```
for _, f := range x.Fields.List {
   fieldName := ""
   if len(f.Names) != 0 {
      fieldName = f.Names[0].Name
   }

   if f.Names == nil {
      ident, ok := f.Type.(*ast.Ident)
      if !ok {
         continue // anonymous field
      }

      fieldName = ident.Name
   }

   res, err := c.process(fieldName, f.Tag.Value)
   if err != nil {
      rewriteErr = err
      return true
   }
   f.Tag.Value = res
}
```

Processing the tags

```
func (c *config) process(fieldName, tagVal string) (string, error) {
   var tag string
   if tagVal != "" {
      var err error
      tag, err = strconv.Unquote(tagVal)
      if err != nil {
         return "", err
      }
   }

   tags, err := structtag.Parse(tag)
   if err != nil {
      return "", err
   }

   // process tags ...

   res := tags.String()
   if res != "" {
      res = quote(tags.String())
   }
   return res, nil
}
```

```
func (c *config) process(fieldName, tagVal string) (string, error) {
   var tag string
   if tagVal != "" {
      var err error
      tag, err = strconv.Unquote(tagVal)
      if err != nil {
         return "", err
      }
   }

   tags, err := structtag.Parse(tag)
   if err != nil {
      return "", err
   }

   // process tags ...

   res := tags.String()
   if res != "" {
      res = quote(tags.String())
   }
   return res, nil
}
```

```
func (c *config) process(fieldName, tagVal string) (string, error) {
   var tag string
   if tagVal != "" {
      var err error
      tag, err = strconv.Unquote(tagVal)
      if err != nil {
         return "", err
      }
   }

   tags, err := structtag.Parse(tag)
   if err != nil {
      return "", err
   }

   // process tags ...

   res := tags.String()
   if res != "" {
      res = quote(tags.String())
   }
   return res, nil
}
```

```
func (c *config) process(fieldName, tagVal string) (string, error) {
   var tag string
   if tagVal != "" {
      var err error
      tag, err = strconv.Unquote(ta
      if err != nil {
         return "", err
      }
   }

   tags, err := structtag.Parse(ta
   if err != nil {
      return "", err
   }

   // process tags ...

   res := tags.String()
   if res != "" {
      res = quote(tags.String())
   }
   return res, nil
}
```

```
tags = c.removeTags(tags)
tags, err = c.removeTagOptions(tags)
if err != nil {
   return "", err
}

tags = c.clearTags(tags)
tags = c.clearOptions(tags)

tags, err = c.addTags(fieldName, tags)
if err != nil {
   return "", err
}

tags, err = c.addTagOptions(tags)
if err != nil {
   return "", err
}

if c.sort {
   sort.Sort(tags)
}
```

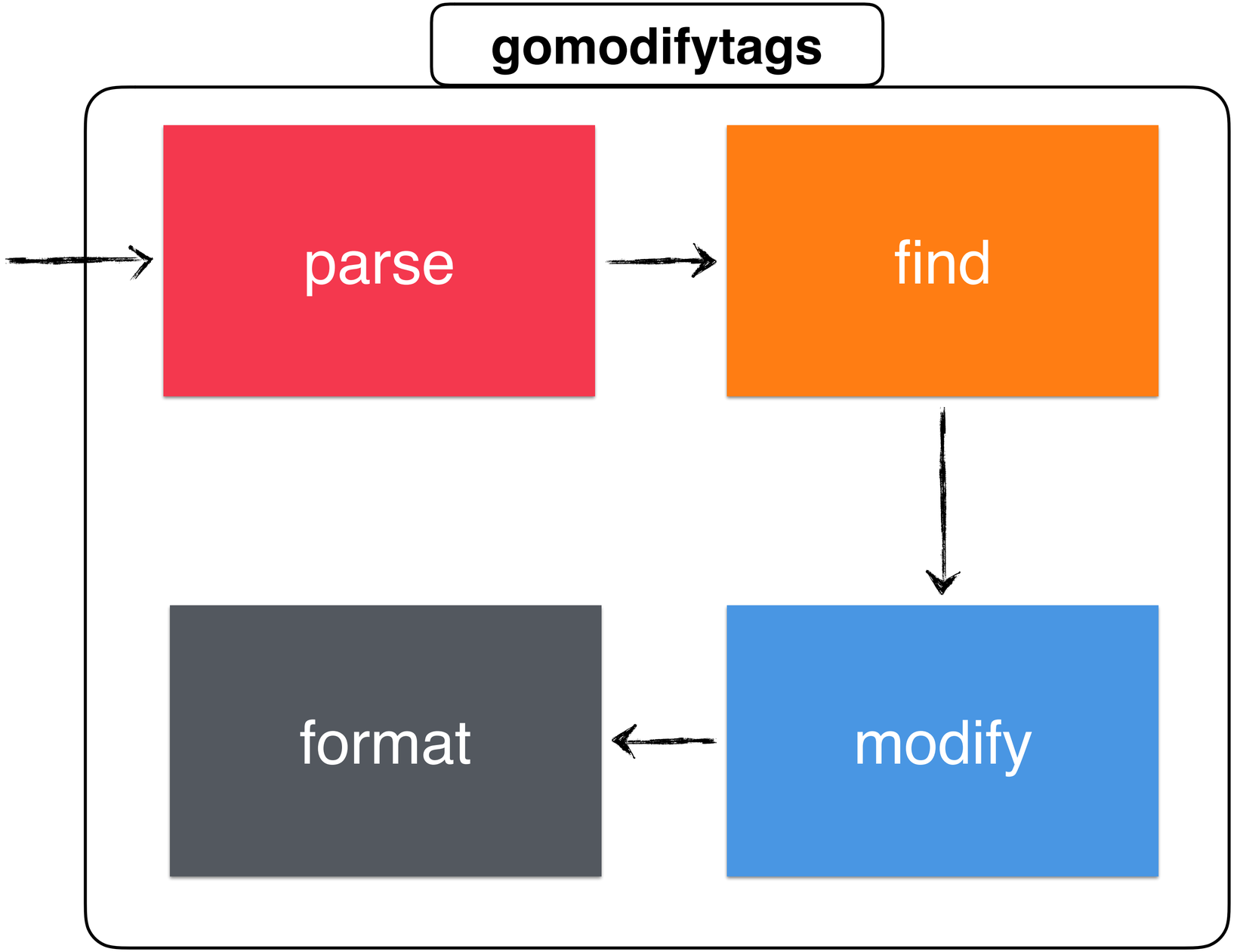
```
func (c *config) process(fieldName, tagVal string) (string, error) {
   var tag string
   if tagVal != "" {
      var err error
      tag, err = strconv.Unquote(ta
      if err != nil {
         return "", err
      }
   }

   tags, err := structtag.Parse(ta
   if err != nil {
      return "", err
   }

   // process tags ...

   res := tags.String()
   if res != "" {
      res = quote(tags.String())
   }
   return res, nil
}
```

```
tags = c.removeTags(tags)
tags, err = c.removeTagOptions(tags)
if err != nil {
   return "", err
}

tags = c.clearTags(tags)
tags = c.clearOptions(tags)

tags, err = c.addTags(fieldName, tags)
if err != nil {
   return "", err
}

tags, err = c.addTagOptions(tags)
if err != nil {
   return "", err
}

if c.sort {
   sort.Sort(tags)
}
```

```
func (c *config) process(fieldName, tagVal string) (string, error) {
   var tag string
   if tagVal != "" {
      var err error
      tag, err = strconv.Unquote(tagVal)
      if err != nil {
         return "", err
      }
   }

   tags, err := structtag.Parse(tag)
   if err != nil {
      return "", err
   }

   // process tags ...

   res := tags.String()
   if res != "" {
      res = quote(tags.String())
   }
   return res, nil
}
```

```
func (c *config) rewriteFields(node ast.Node) (ast.Node, error) {
   var rewriteErr error
   rewriteFunc := func(n ast.Node) bool {
      x, ok := n.(*ast.StructType)
      if !ok {
         return true
      }

      for _, f := range x.Fields.List {
         // process each field
         // ...
      }

      return true
   }

   ast.Inspect(node, rewriteFunc)
   return node, rewriteErr
}
```

```
package main

type Example struct {
    Foo string
}
```



Output the result

1. **Fetch** configuration settings
2. **Parse** content
3. **Find** selection
4. **Modify** the struct tag
5. **Output** the result

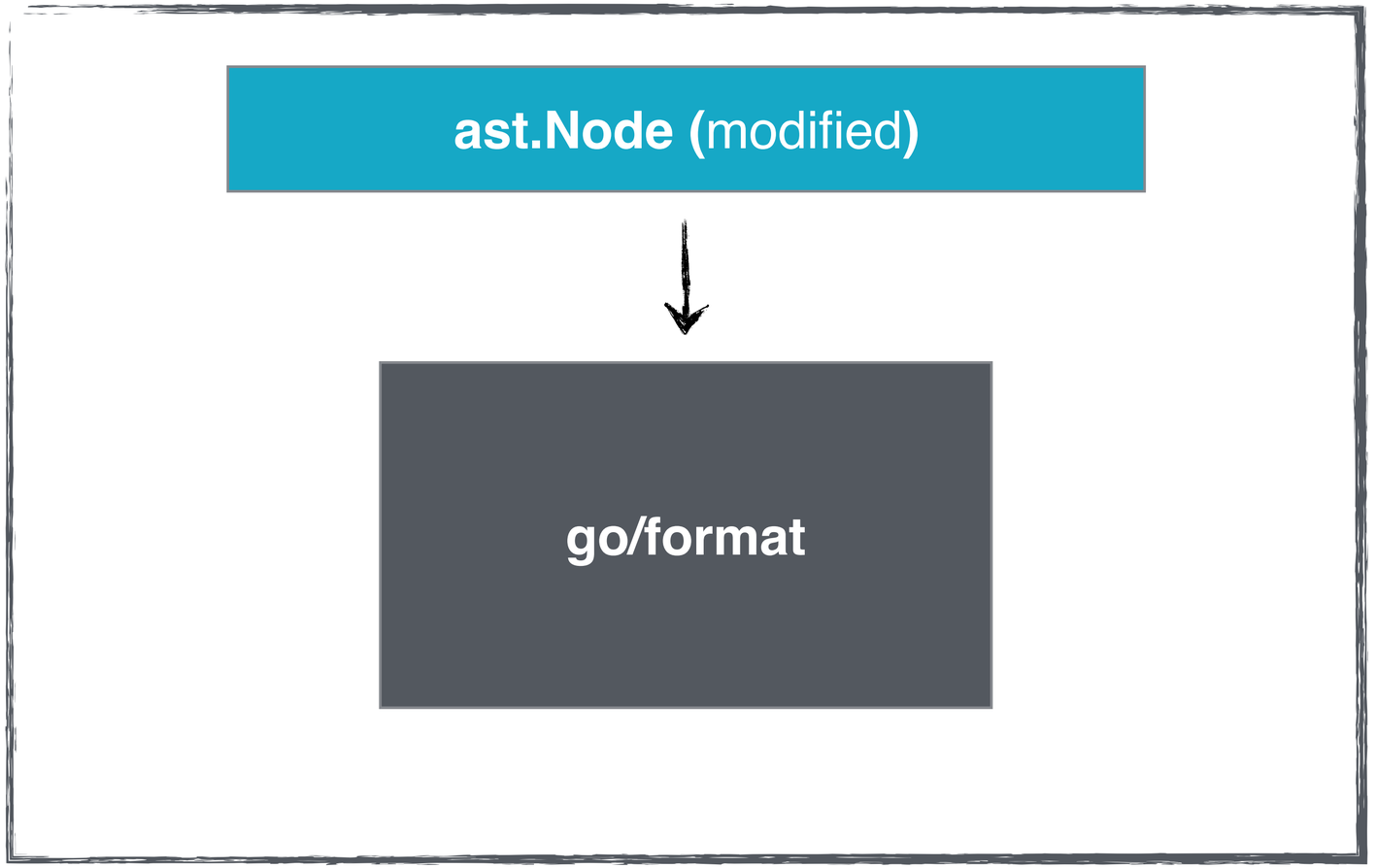
```
func main() {
   var cfg config

   node = cfg.parse()

   start, end = cfg.findSelection(node)

   rewritten = cfg.rewrite(node, start, end)

   out = cfg.format(rewritten)

   fmt.Println(out)
}
```

gomodifytags

ast.Node (modified)

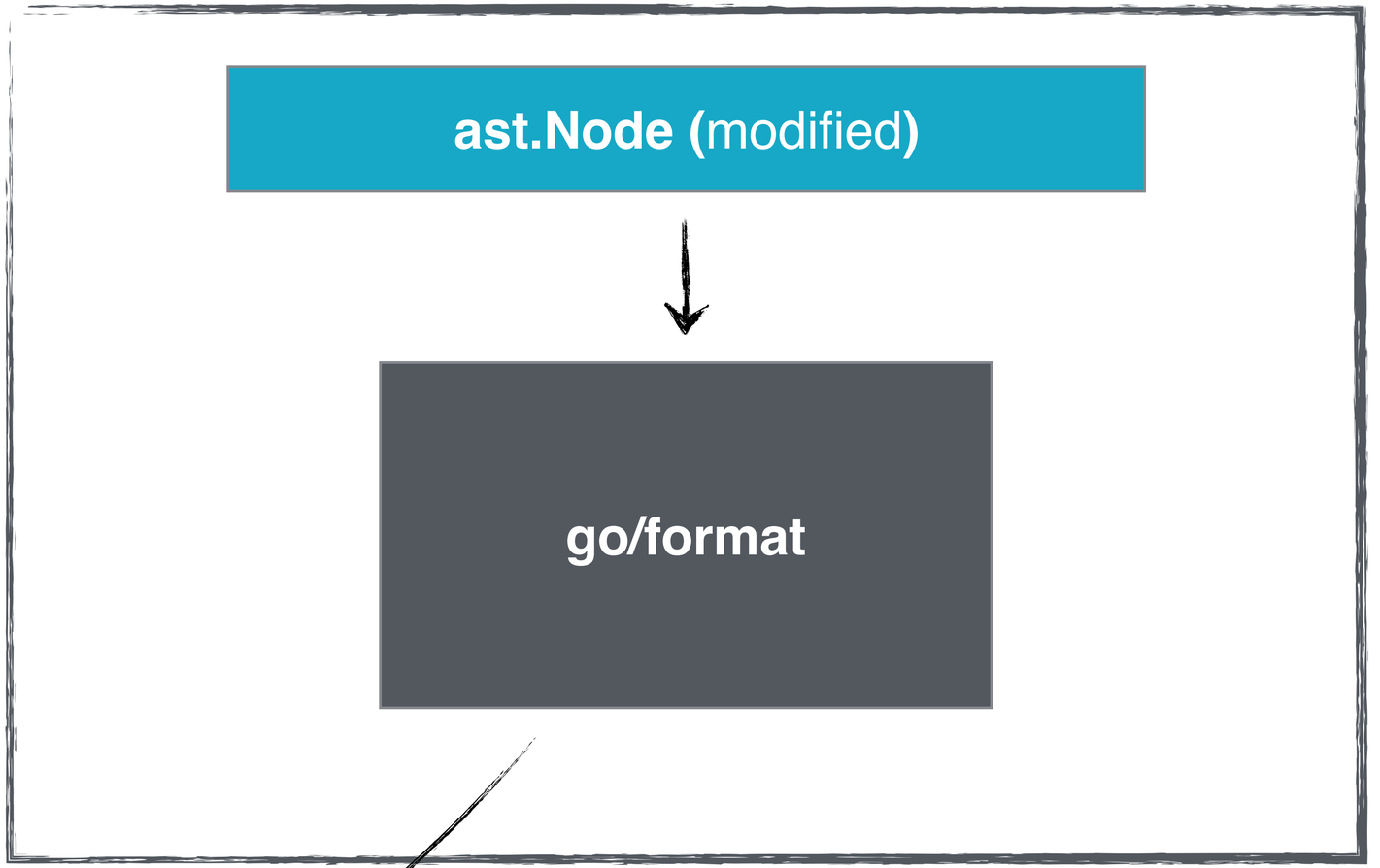
gomodifytags

Pass the modified node
to go/format



gomodifytags

... and output the **result** to **stdout**




```
package main

type Example struct {
    Foo string `json:"foo"`
}
```

Input

```
package main

type Server struct {
   Name         string
   Port         int
   EnableLogs   bool
   BaseDomain   string
   Credentials struct {
      Username string
      Password string
   }
}
```

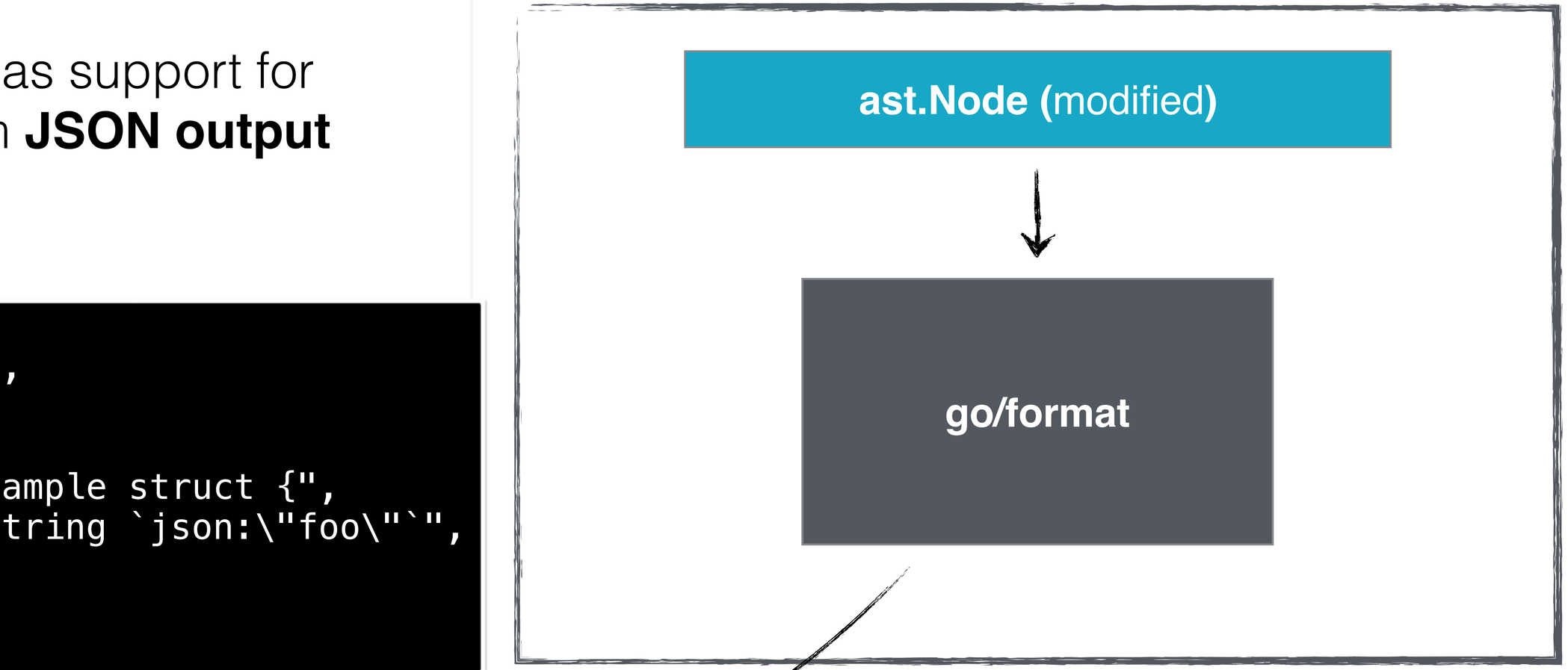
Output

```
package main

type Server struct {
   Name         string `json:"name"`
   Port         int    `json:"port"`
   EnableLogs   bool   `json:"enable_logs"`
   BaseDomain   string `json:"base_domain"`
   Credentials struct {
      Username string `json:"username"`
      Password string `json:"password"`
   } `json:"credentials"`
}
```

$ **gomodifytags**
-**file** example.go
-**struct** Server
-**add-tags** json

gomodifytags

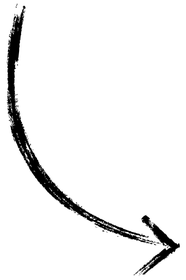
Also has support for
custom **JSON output**




```
{
  "start": 3,
  "end": 5,
  "lines": [
    "type Example struct {",
    "  Foo string `json:\"foo\"`",
    "}"
  ]
}
```

Input

```
package main

type Server struct {
   Name        string
   Port        int
   EnableLogs  bool
   BaseDomain  string
   Credentials struct {
      Username string
      Password string
   }
}
```



```
$ gomodifytags
  -file example.go
  -struct Server
  -add-tags json
  -format json
```

JSON Output

```
{
  "start": 3,
  "end": 12,
  "lines": [
    "type Server struct {",
    "  Name        string `xml:\"name\"`",
    "  Port        int    `xml:\"port\"`",
    "  EnableLogs  bool   `xml:\"enable_logs\"`",
    "  BaseDomain  string `xml:\"base_domain\"`",
    "  Credentials struct {",
    "    Username string `xml:\"username\"`",
    "    Password string `xml:\"password\"`",
    "  } `xml:\"credentials\"`",
    "}"
  ]
}
```

Why **stdout?**

- by default, it means "**dry-run**"
- immediate **feedback**
- tools can **redirect** the output
- **composable** (gomodifytags myfile.go > newfile.go)

Output the result

```
func (c *config) format(file ast.Node) (string, error) {
   switch c.output {
   case "source":
      // return formatted source

   case "json":
      // return only changes in json

   default:
      return "", fmt.Errorf("unknown output mode: %s", c.output)
   }
}
```

Output the result

```
func (c *config) format(file ast.Node) (string, error) {
   switch c.output {
   case "source":
      // return formatted source

   case "json":
      // return only changes in json

   default:
      return "", fmt.Errorf("unknown output mode: %s", c.output)
   }
}
```

Source file

```
var buf bytes.Buffer
err := format.Node(&buf, c.fset, file)
if err != nil {
   return "", err
}

if c.write {
   err = ioutil.WriteFile(c.file, buf.Bytes(), 0)
   if err != nil {
      return "", err
   }
}

return buf.String(), nil
```

Source file

```
var buf bytes.Buffer
err := format.Node(&buf, c.fset, file)
if err != nil {
   return "", err
}

if c.write {
   err = ioutil.WriteFile(c.file, buf.Bytes(), 0)
   if err != nil {
      return "", err
   }
}

return buf.String(), nil
```

Source file

```
var buf bytes.Buffer
err := format.Node(&buf, c.fset, file)
if err != nil {
   return "", err
}

if c.write {
   err = ioutil.WriteFile(c.file, buf.Bytes(), 0)
   if err != nil {
      return "", err
   }
}

return buf.String(), nil
```

JSON output

```
var buf bytes.Buffer
err := format.Node(&buf, c.fset, file)
if err != nil {
   return "", err
}

var lines []string
scanner := bufio.NewScanner(bytes.NewBufferString(buf.String()))
for scanner.Scan() {
   lines = append(lines, scanner.Text())
}

if c.start > len(lines) {
   return "", errors.New("line selection is invalid")
}


...
```

JSON output

```
var buf bytes.Buffer
err := format.Node(&buf, c.fset, file)
if err != nil {
   return "", err
}

var lines []string
scanner := bufio.NewScanner(bytes.NewBufferString(buf.String()))
for scanner.Scan() {
   lines = append(lines, scanner.Text())
}

if c.start > len(lines) {
   return "", errors.New("line selection is invalid")
}


...
```

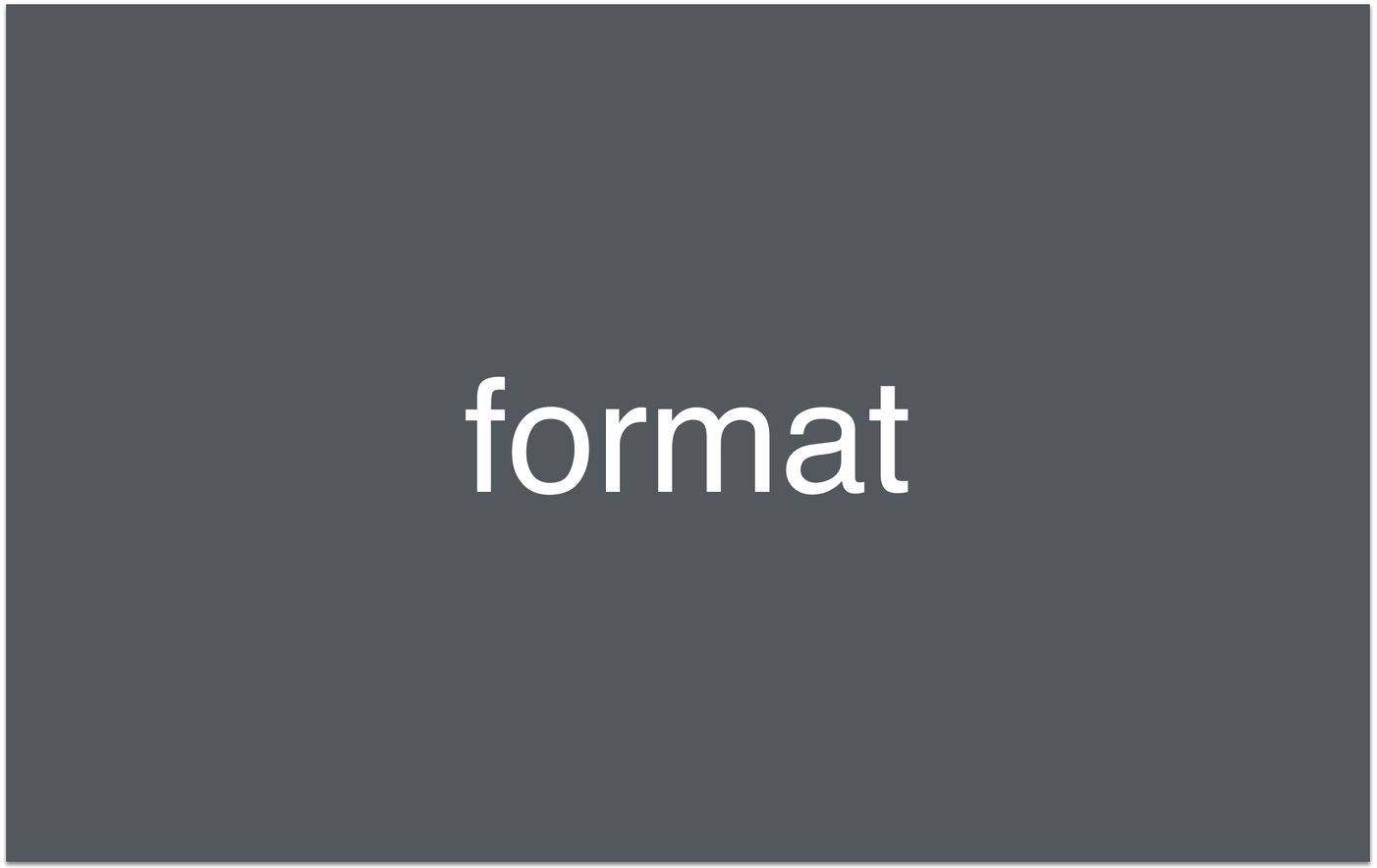
JSON output (cont.)

```
out := &output{
   Start: c.start,
   End:   c.end,
   Lines: lines[c.start-1 : c.end],
}

o, err := json.MarshalIndent(out, "", "  ")
if err != nil {
   return "", err
}

return string(o), nil
```

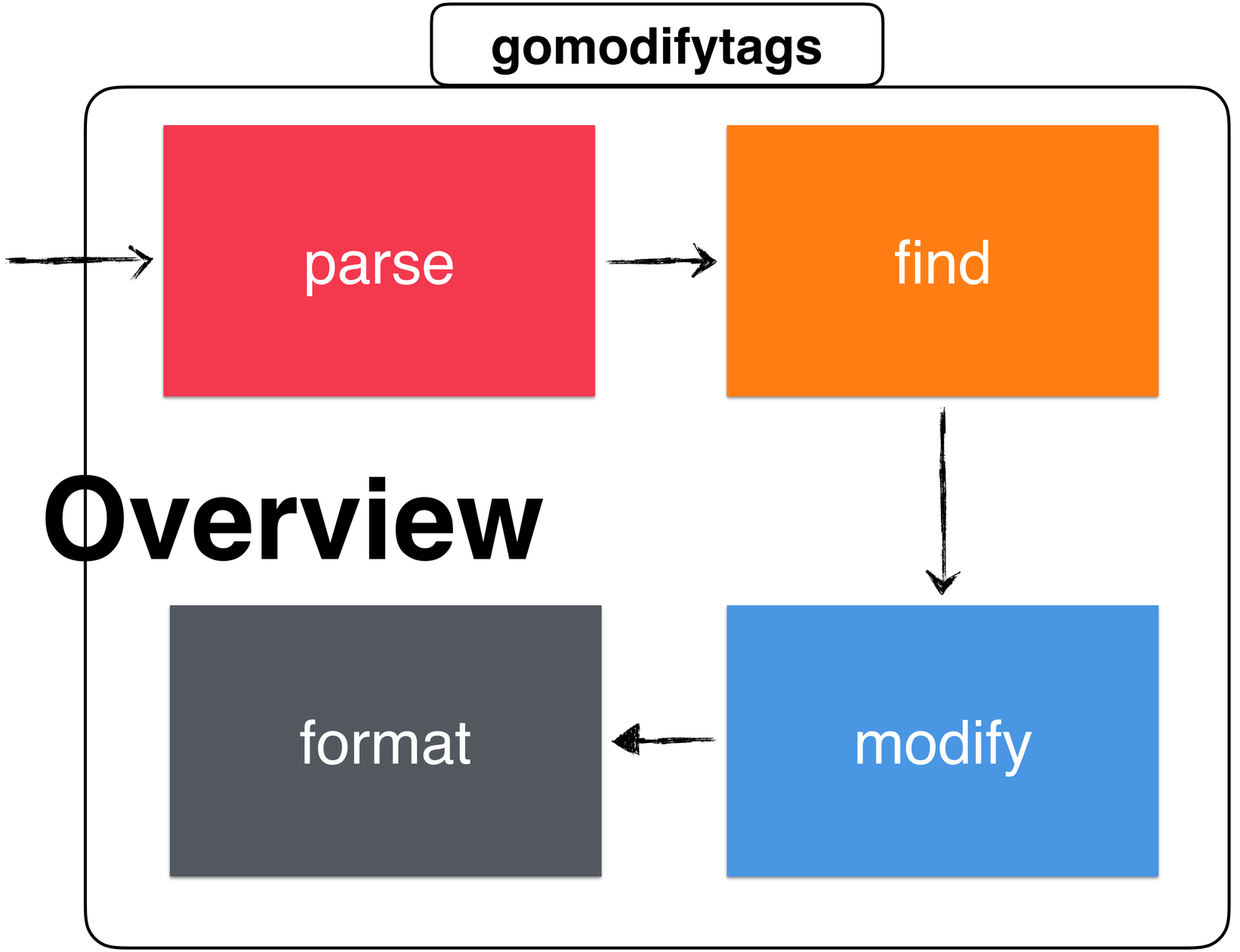
JSON output (cont.)

```
out := &output{
   Start: c.start,
   End:   c.end,
   Lines: lines[c.start-1 : c.end],
}

o, err := json.MarshalIndent(out, "", "  ")
if err != nil {
   return "", err
}

return string(o), nil
```

gomodifytags



```
package main

type Example struct {
    Foo string
}
```



Overview

gomodifytags

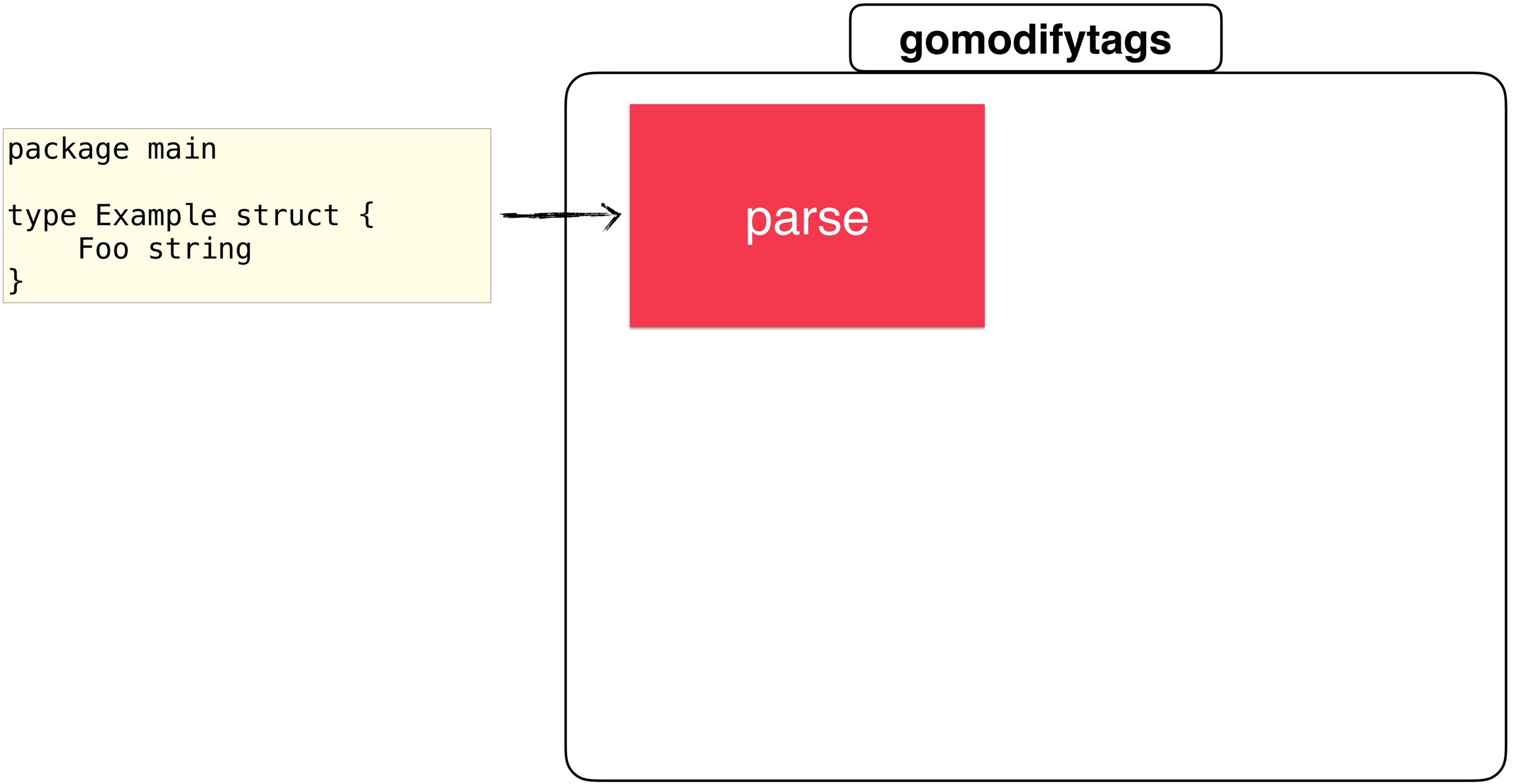
```
package main

type Example struct {
    Foo string
}
```

gomodifytags

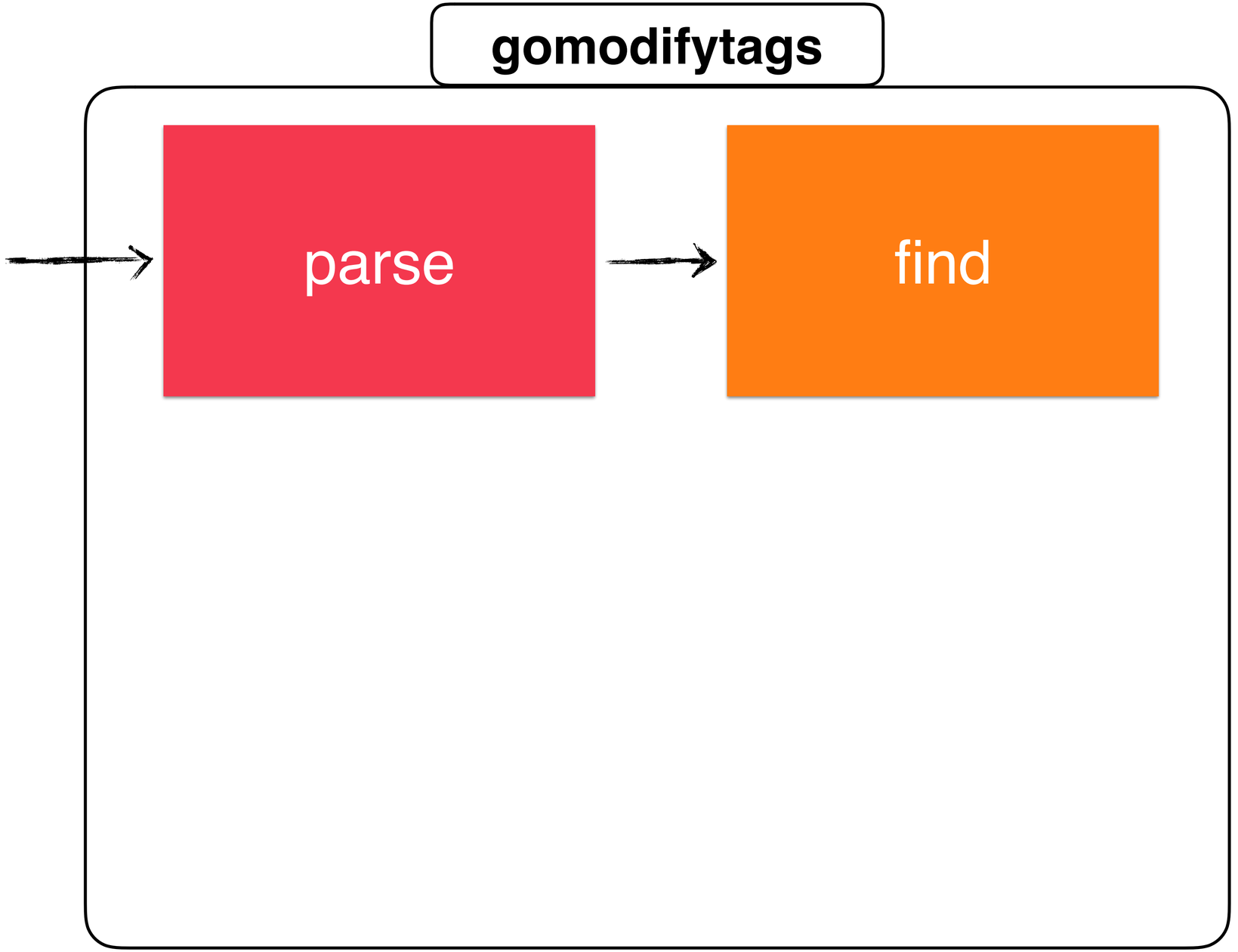
```
package main

type Example struct {
    Foo string
}
```

gomodifytags

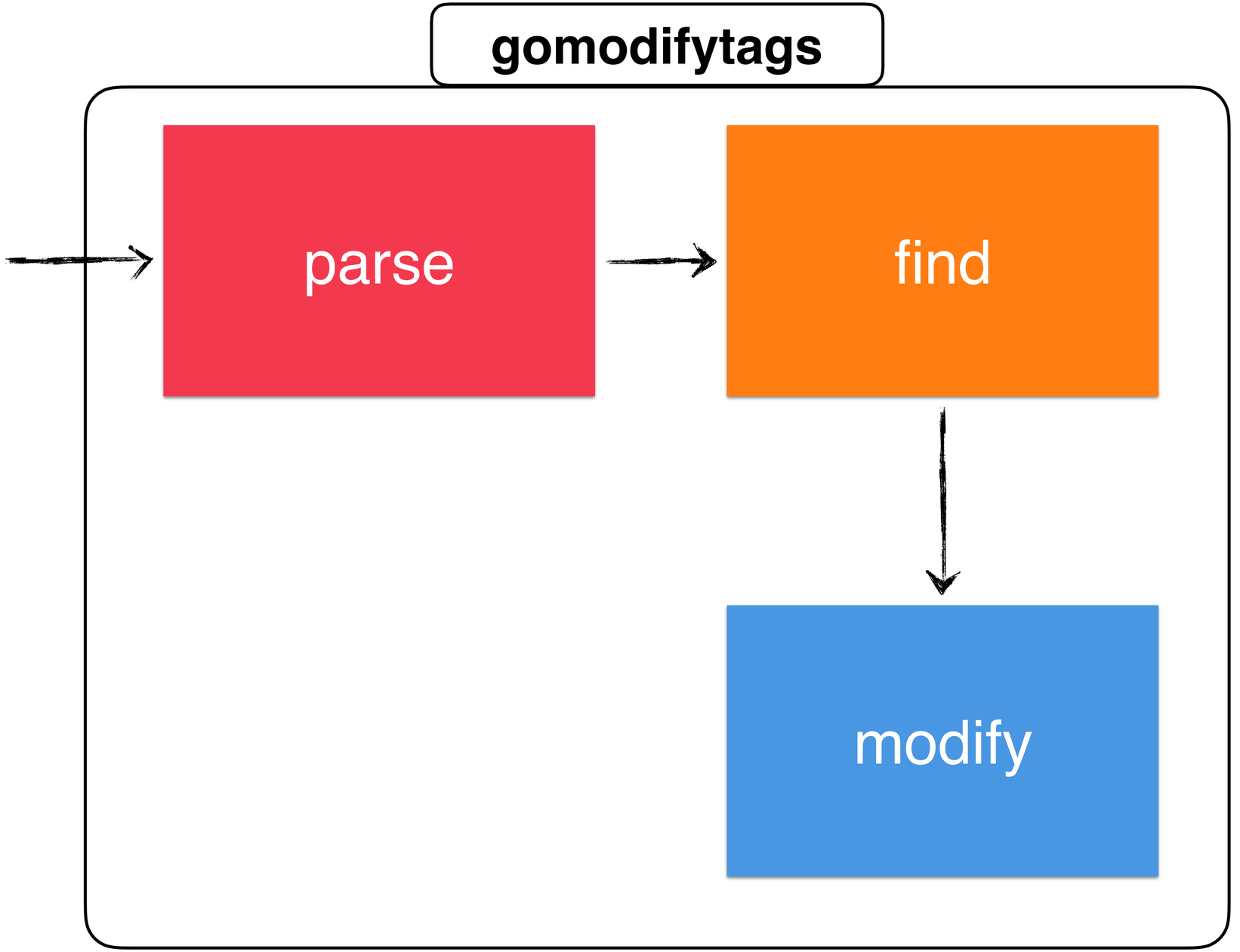
-**file** example.go
-**struct** Example
-**add-tags** json


gomodifytags
package main
type Example struct {
Foo string
}
parse

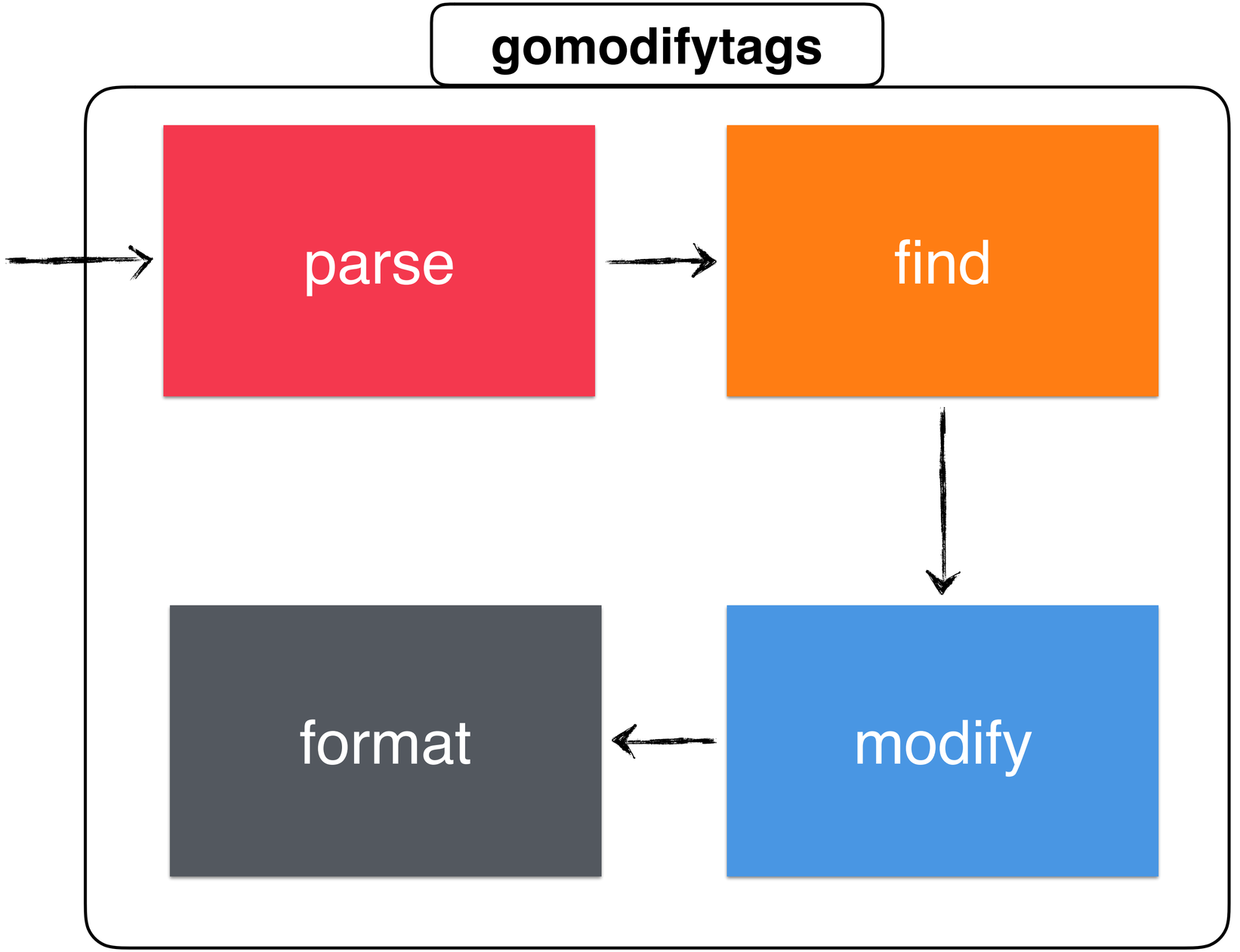
```
package main

type Example struct {
    Foo string
}
```

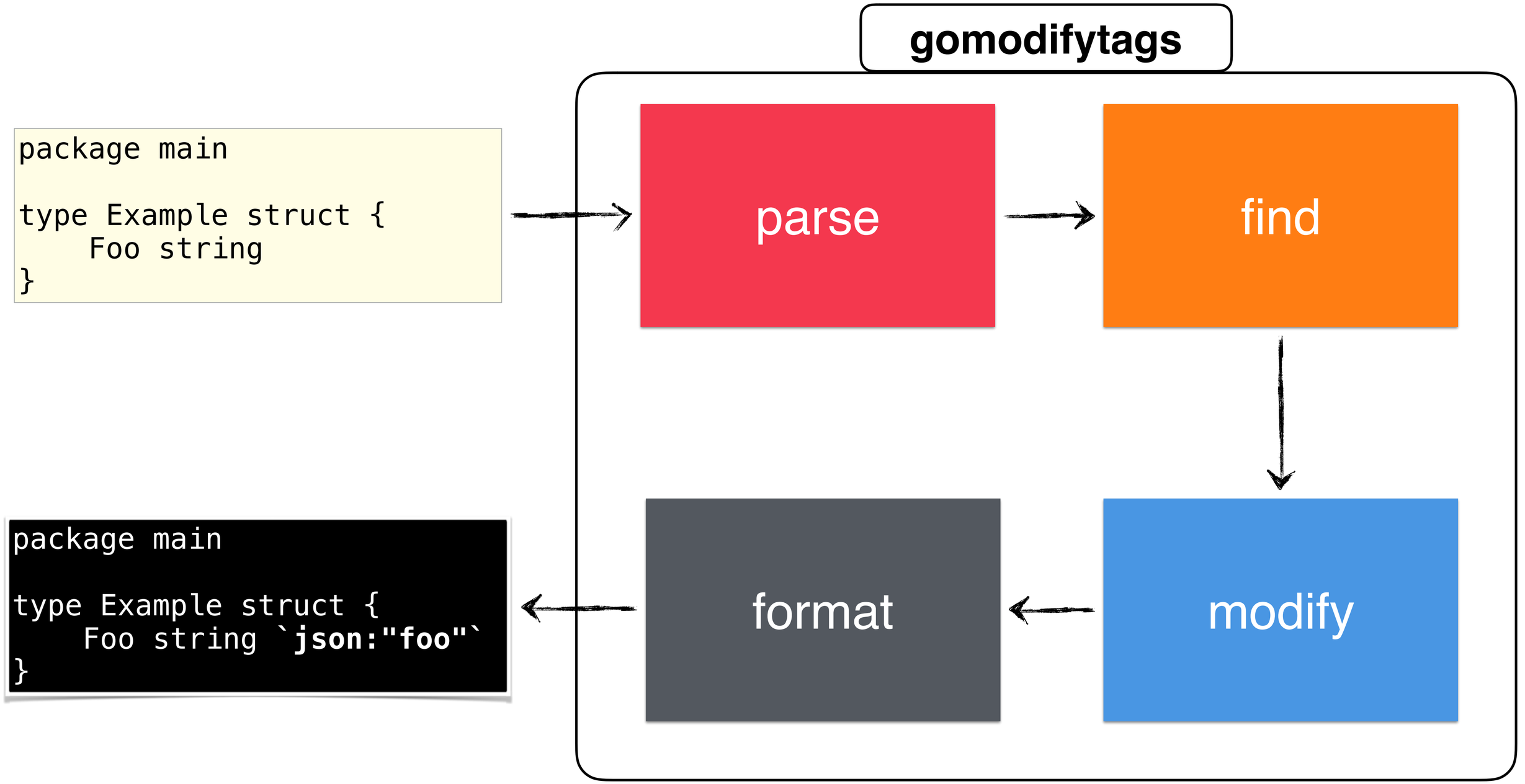


```
package main

type Example struct {
    Foo string
}
```



```
package main

type Example struct {
    Foo string
}
```




gomodifytags
package main
type Example struct {
Foo string
}
parse
find
modify
format
package main
type Example struct {
Foo string `json:"foo"`
}

Demo time

Built from the ground up **for editors**

- selecting structs based on offset or range of lines
- write result to stdout or file
- json output for easy parsing
- support for unsaved buffers
- individual cli flags for all features

Supported **editors**

- **vim** with vim-go
- **atom** with go-plus (thanks **Zac Bergquist**)
- **vscode** with vscode-go (thanks **Ramya Rao)**
- **emacs** - WIP (https://github.com/syohex/emacs-go-add-tags/issues/6)

Fixed bugs

	0 Open	✓ **8 Closed**	Author	Labels	Milestones	Assignee	Sort
	:GoAddTags do not support camelcase improvement tool/goaddtags #1265 by 812lcl was closed 25 days ago						4
	Goaddtags if the field comments contains braces bug tool/goaddtags #1189 by fatih was closed on Feb 14						1
	goaddtags should append to a one line comment bug tool/goaddtags #1188 by fatih was closed on Feb 14						1
	GoAddTags bug when using *interface{} fields bug tool/goaddtags #1091 by marcosnils was closed on Feb 14						1
	GoAddTags should not add twice bug tool/goaddtags #1064 by gnhuy91 was closed on Feb 14						2
	:GoAddTags with nested structs improvement tool/goaddtags #990 by dvcrn was closed on Feb 14						4
	GoAddTags: tag options improvement tool/goaddtags #985 by tobstarr was closed on Feb 14						6
	GoAddTags: allow modifying existing comments improvement tool/goaddtags #984 by tobstarr was closed on Feb 14						1

Recap

- A **single** tool to rule **all editors** 👑
- **Easy** maintenance 🛠️
- **Free of bugs** due stdlib tooling (go/parser family) 🐛
- **Happy** and **productive** users 😊

Thanks!

Fatih Arslan

 @fatih

 @fatih

 fatih@arslan.io