

GopherCon Africa 2026



Hi

ainsley.dev

godaily.dev

ainsleyclark.com

//go:embed

Why?

“ The missing piece that helps us keep a single binary...

— Barak Amar

What it is

The directive

How to use it

The variable

How it's built

The binary

What you take on

The tradeoffs

What it is

The directive

How

Th

How to use it

The variable

How

Th

How it's built

The binary

What

Th

What you take on

The tradeoffs

01

WHAT IT IS

The directive

```
1 $ docker run myapp
2 open ./static/app.css: no such file or directory
```

```
1 $ docker run myapp
2 open ./static/app.css: no such file or directory
```

```
1 $ docker run myapp
2 open ./static/app.css: no such file or directory
```

```
1 //go:embed version.txt  
2 var version string
```

```
1 //go:embed version.txt  
2 var version string
```

```
1 //go:embed version.txt
2 var version string
```

What it is not

What it is not

02

HOW TO USE IT

The variable

```
1  import "embed" // Or import _ "embed"
2
3  //go:embed version.txt
4  var version string
5
6  //go:embed schema.sql
7  var schema []byte
8
9  //go:embed assets/*
10 var assets embed.FS
11
12 func main() {
13     fmt.Println(version) // v1.4.2
14     db.Exec(string(schema))
15
16     css, _ := assets.ReadFile("assets/index.css")
17     fmt.Println(string(css)) // body { margin: 0 }
18 }
```

```
1  import "embed" // Or import .
2
3  //go:embed version.txt
4  var version string
```

```
1  import "embed" // Or import
2
3  //go:embed version.txt
4  var version string
5
6  //go:embed schema.sql
```

```
4    var version string
5
6    //go:embed schema.sql
7    var schema []byte
8
9    //go:embed assets/*
```

```
7      var schema []byte
8
9      //go:embed assets/*
10     var assets embed.FS
11
12     func main() {
```

```
10  var assets embed.FS
11
12  func main() {
13      fmt.Println(version) // v1.4.2
14      db.Exec(string(schema))
15
16      css, _ := assets.ReadFile("assets/index.css")
17      fmt.Println(string(css)) // body { margin: 0 }
18  }
```

```
1 // An FS is a read-only collection of files.  
2 type FS struct {}  
3  
4 func (f FS) Open(name string) (fs.File, error)  
5 func (f FS) ReadDir(name string) ([]fs.DirEntry, error)  
6 func (f FS) ReadFile(name string) ([]byte, error)
```

```
1 // An FS is a read-only collection of files.  
2 type FS struct {}  
3  
4 func (f FS) Open(name string) (fs.File, error)  
5 func (f FS) ReadDir(name string) ([]fs.DirEntry, error)  
6 func (f FS) ReadFile(name string) ([]byte, error)
```

Simple as that...

03

HOW IT'S BUILT

The binary

Preface

go build

Packages

go build

Lexer

go build

Patterns

cmd/compile

Parser

cmd/compile

Compiler

cmd/link

Linker

go build

Packages

go build

Lexer

go build

Patterns

cmd/compile

Parser

cmd/compile

Compiler

cmd/link

Linker

go build

Packages

go bu:

Lexo

go build

Lexer

go bu:

Patt

go build

Patterns

cmd/compile

Parser

cmd/co

Con

cmd/compile

Compiler

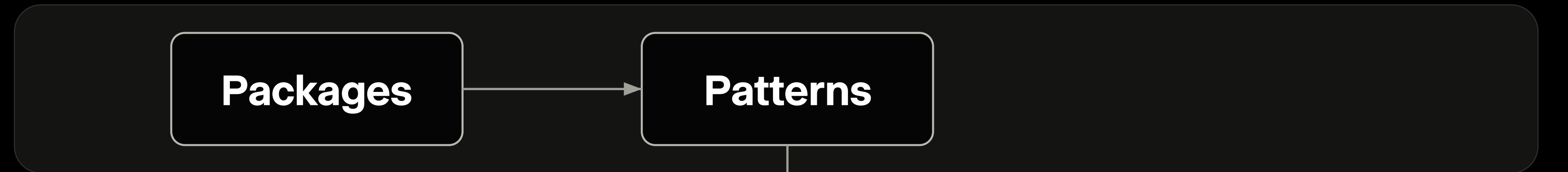
cmd/link

Linker

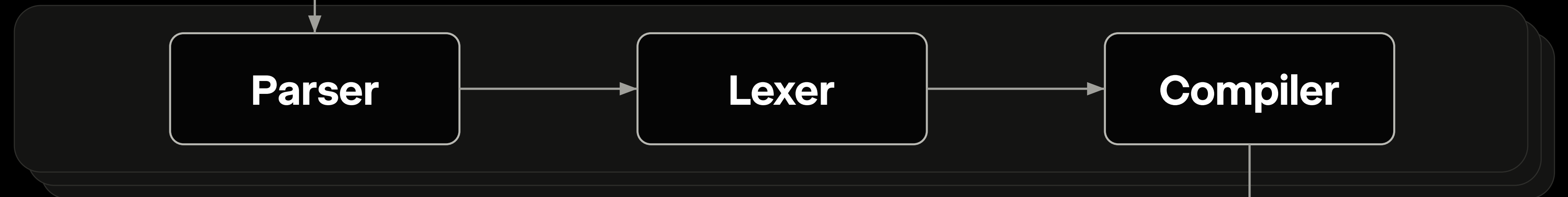
cmd/link

Linker

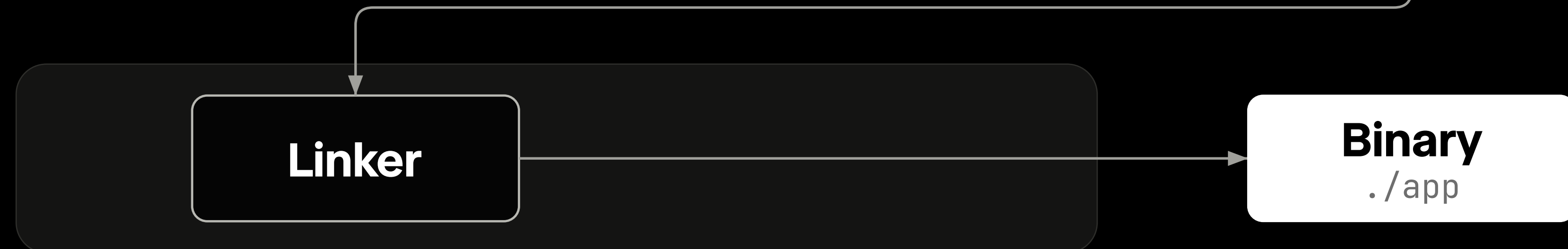
go
runs once



compile
once per package



link
runs once



go

runs once

Packages

Patterns

compile

once per package

Parser

Lexer

Compiler

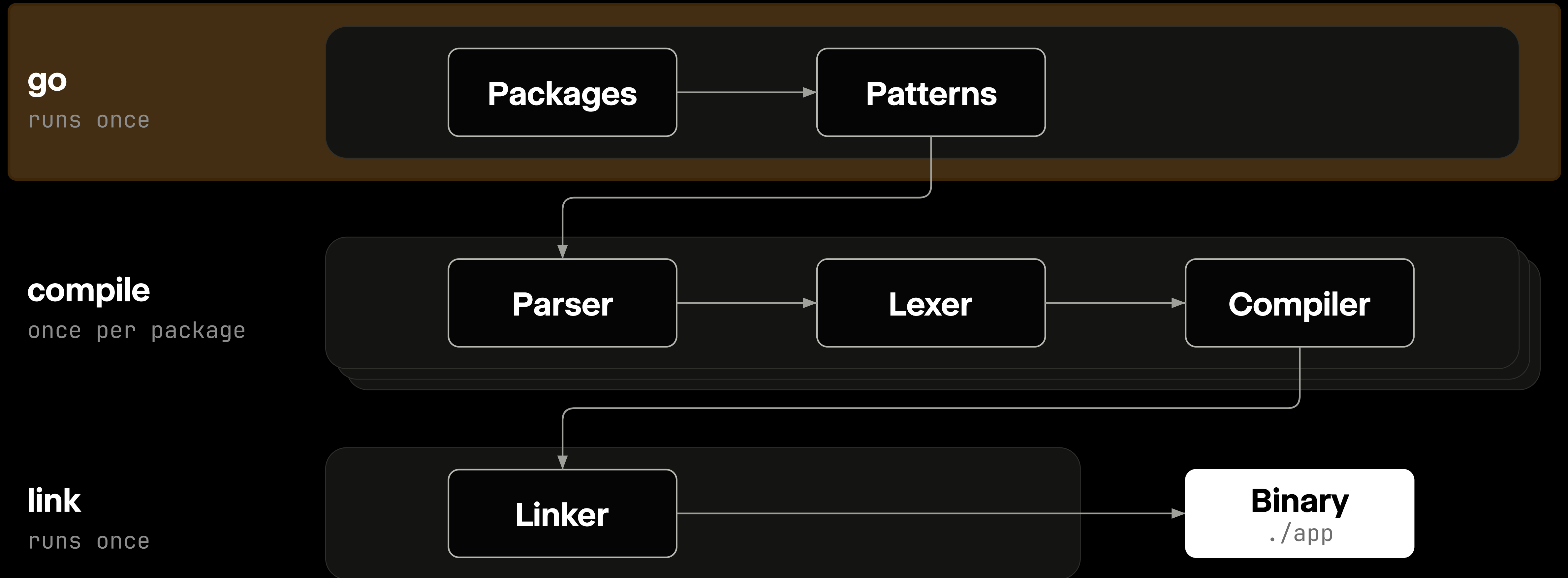
link

runs once

Linker

Binary

./app



go
runs once

Packages

Patterns

compile
once per package

Parser

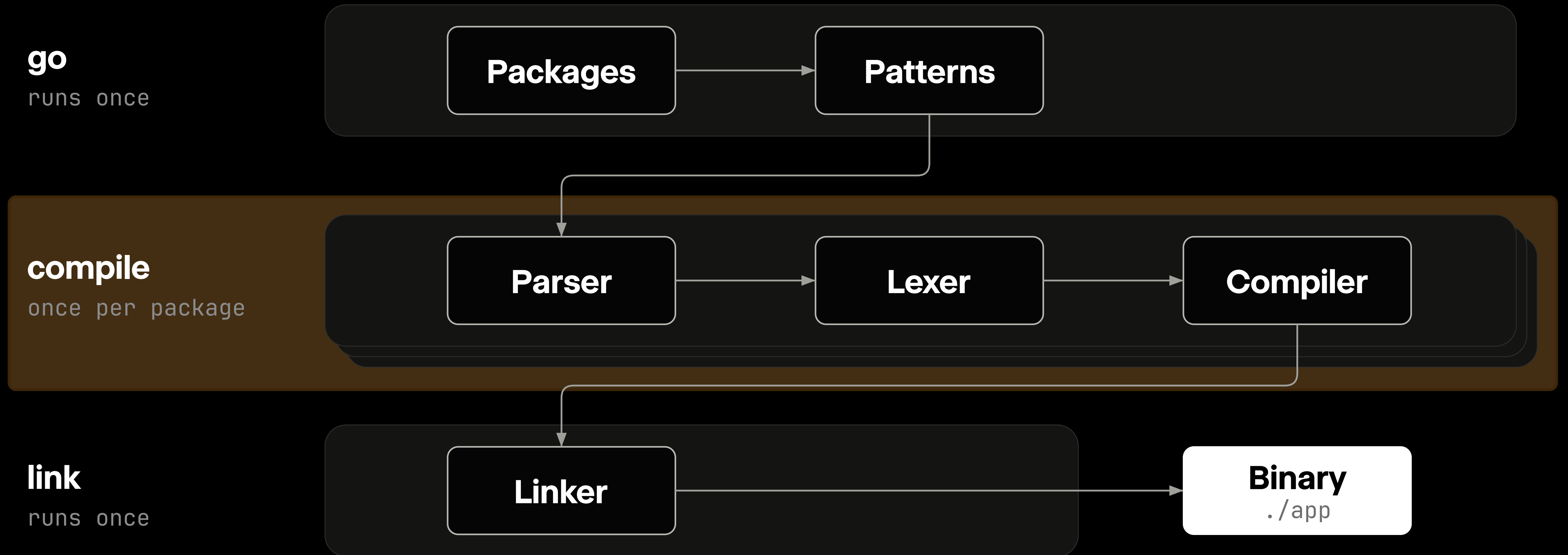
Lexer

Compiler

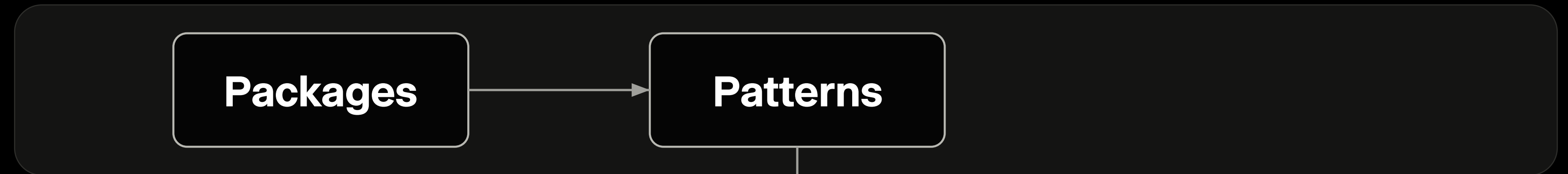
link
runs once

Linker

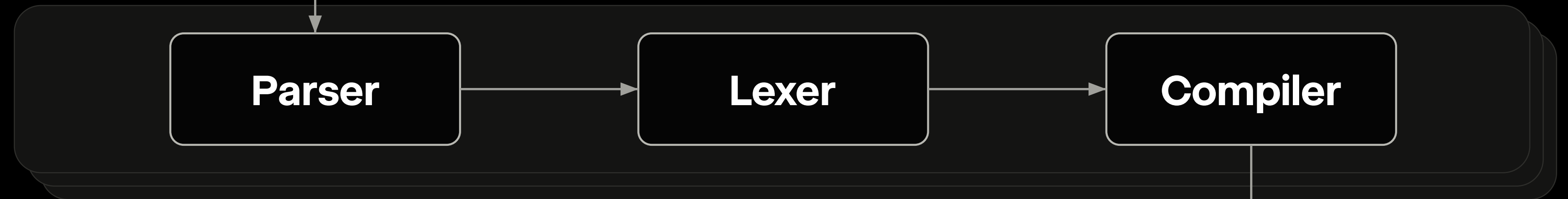
Binary
./app



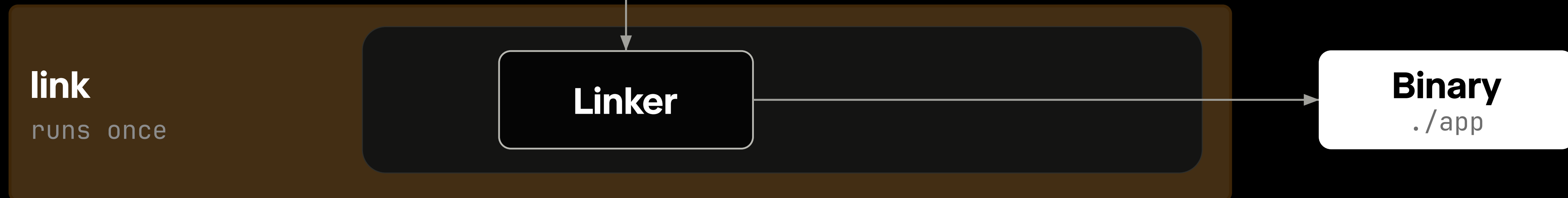
go
runs once



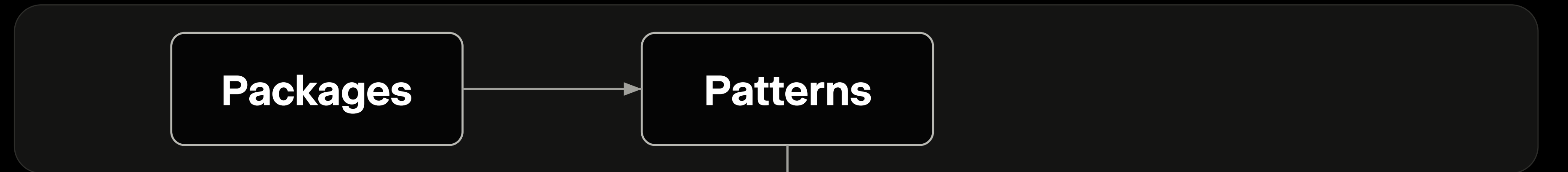
compile
once per package



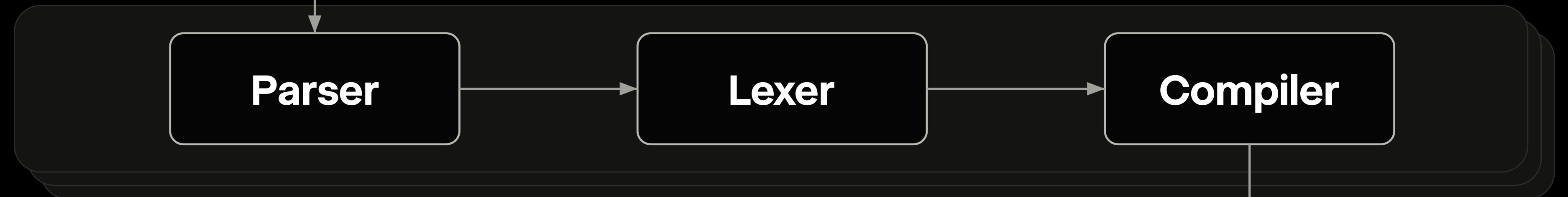
link
runs once



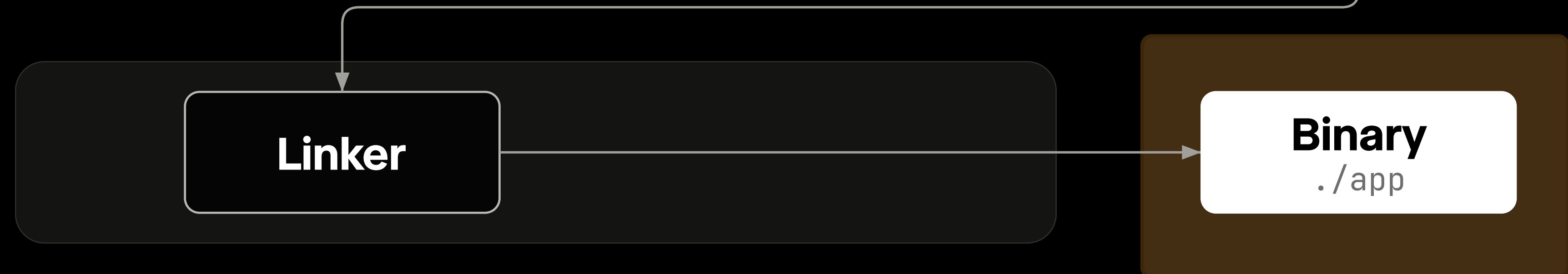
go
runs once



compile
once per package



link
runs once



```
1 .
2 |
3 |— main.go
4 |— index.html (<h1>Hello</h1>)
5 |— assets
6 |   |— index.css
   |   |— index.js
```

```
1 .
2 |
3 | main.go
4 | index.html (<h1>Hello</h1>)
5 | assets
6 |   |
   |   | index.css
   |   |
   |   | index.js
```

```
1  
2  └─ main.go  
3  └─ index.html (<h1>Hello</h1>)  
4  └─ assets  
5     └─ index.css  
6     └─ index.js
```

```
3  └─ index.html (<h1>Hello</h1>
4  └─ assets
5     └─ index.css
6     └─ index.js
```

```
1  package main
2
3  import "embed"
4
5  //go:embed index.html
6  var indexHTML string
7
8  //go:embed assets/*
9  var assets embed.FS // (index.css & index.js)
```

```
3  import "embed"
4
5  //go:embed index.html
6  var indexHTML string
7
8  //go:embed assets/*
```

```
6      var indexHTML string
7
8      //go:embed assets/*
9      var assets embed.FS // (index
```

```
1  package main
2
3  import "embed"
4
5  //go:embed index.html
6  var indexHTML string
7
8  //go:embed assets/*
9  var assets embed.FS // (index.css & index.js)
```

go build

Packages

go build

Lexer

go build

Patterns

cmd/compile

Parser

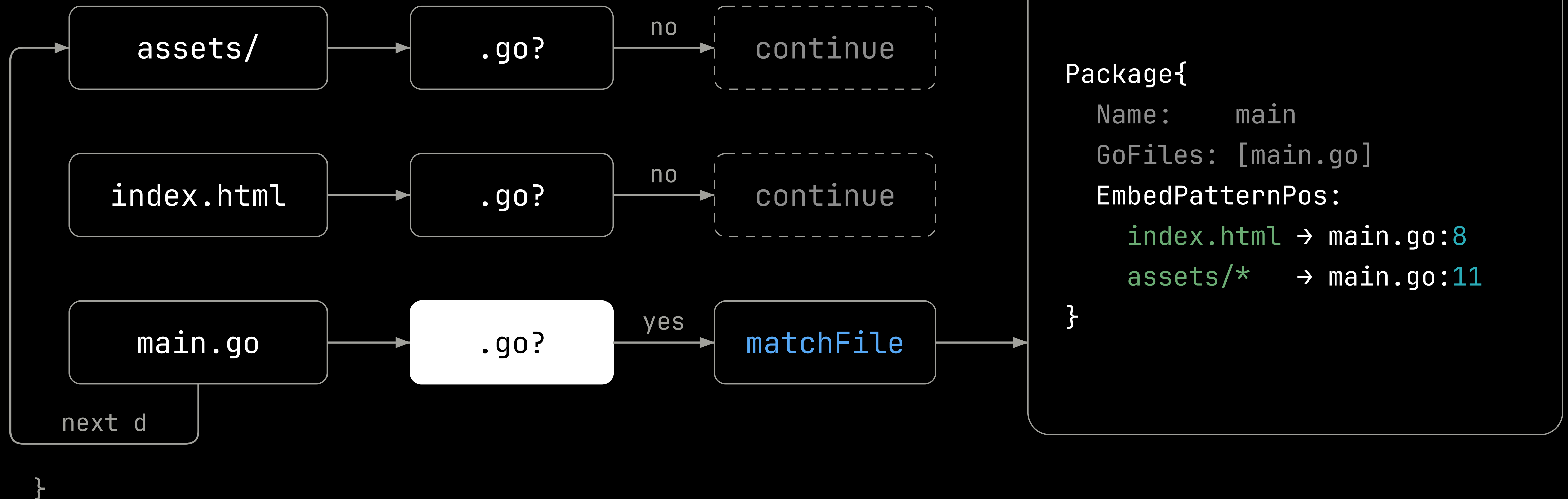
cmd/compile

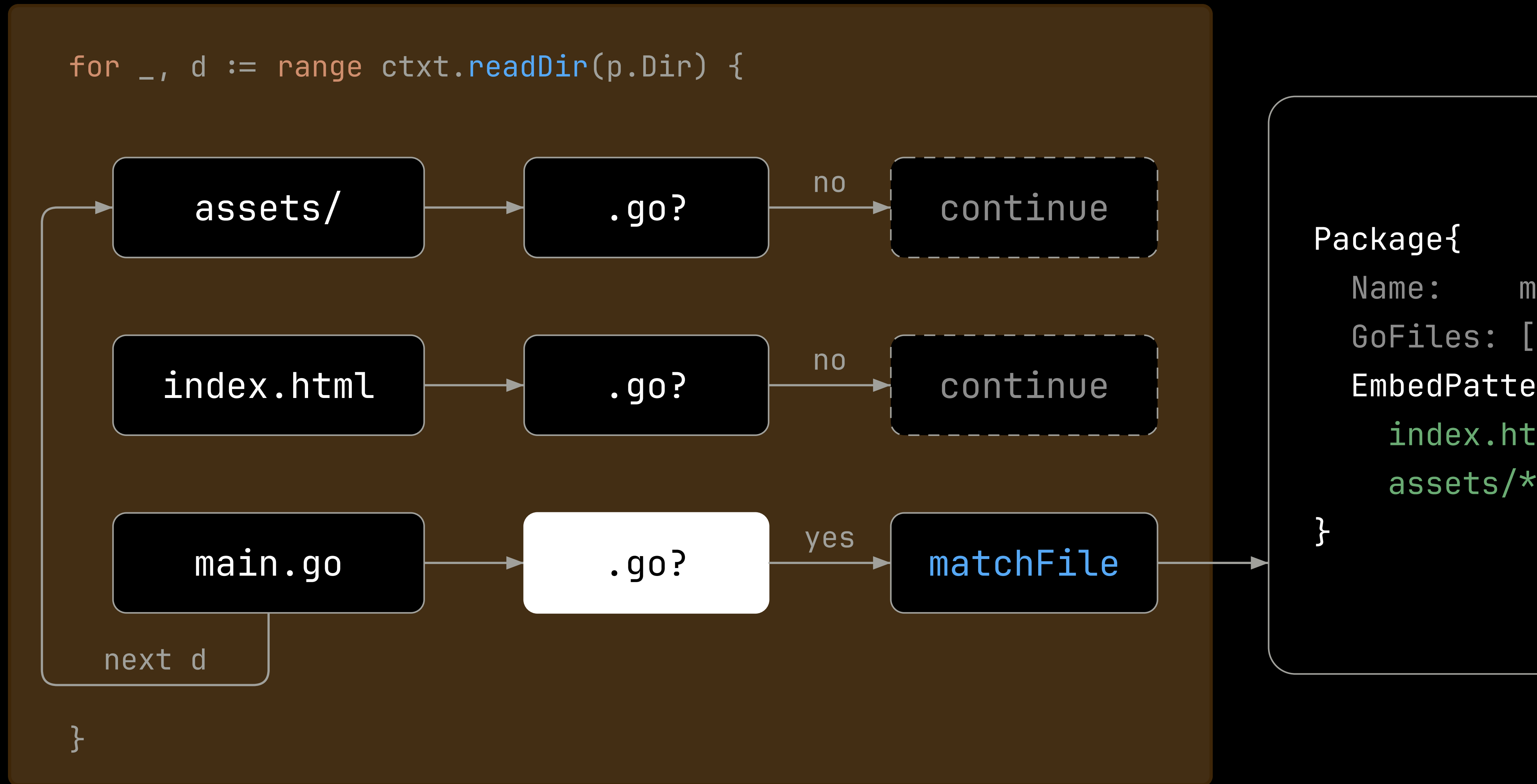
Compiler

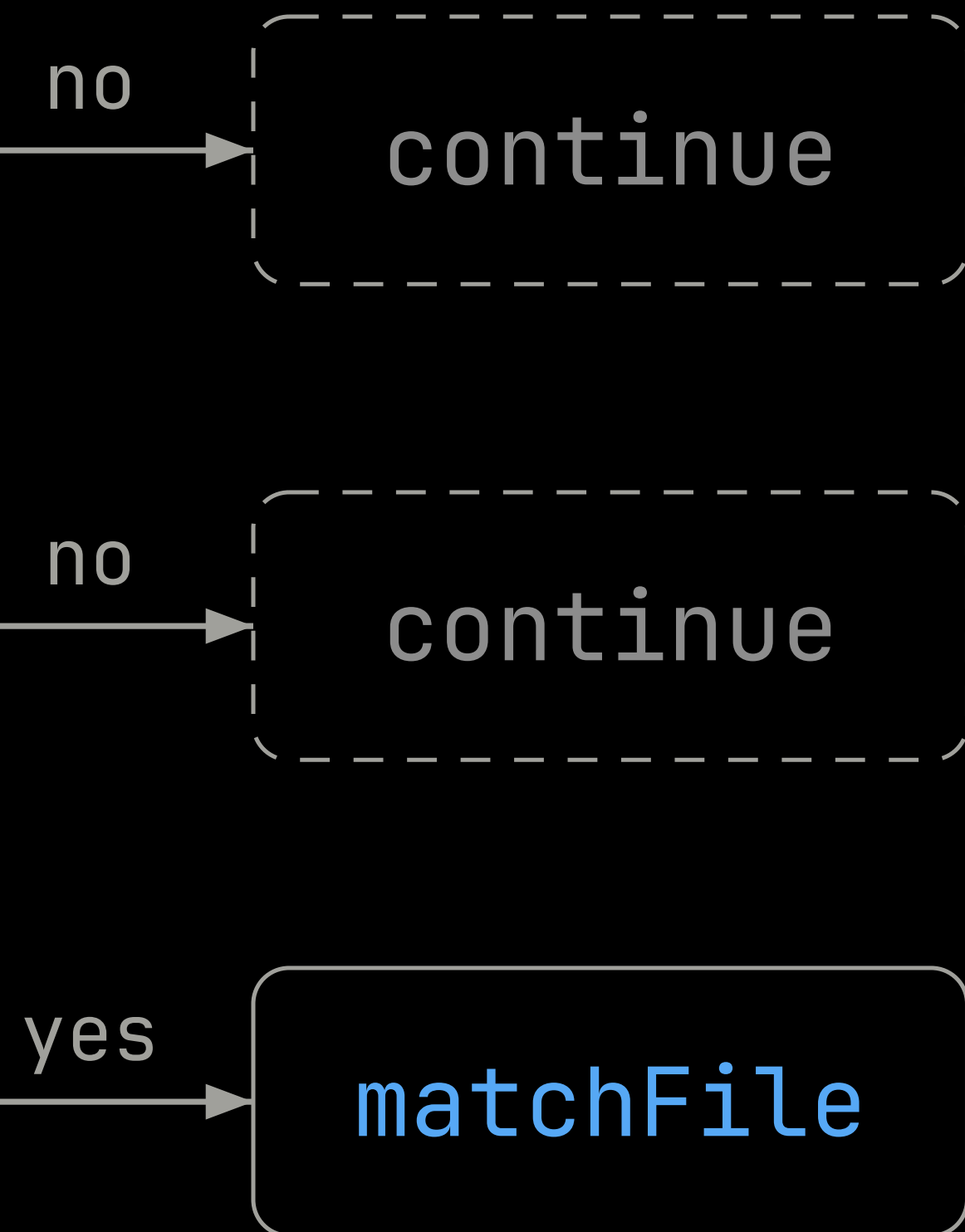
cmd/link

Linker

```
for _, d := range ctxt.readDir(p.Dir) {
```







```
Package{
  Name:    main
  GoFiles: [main.go]
  EmbedPatternPos:
    index.html → main.go:8
    assets/*   → main.go:11
}
```

Name: main

GoFiles: [main.go]

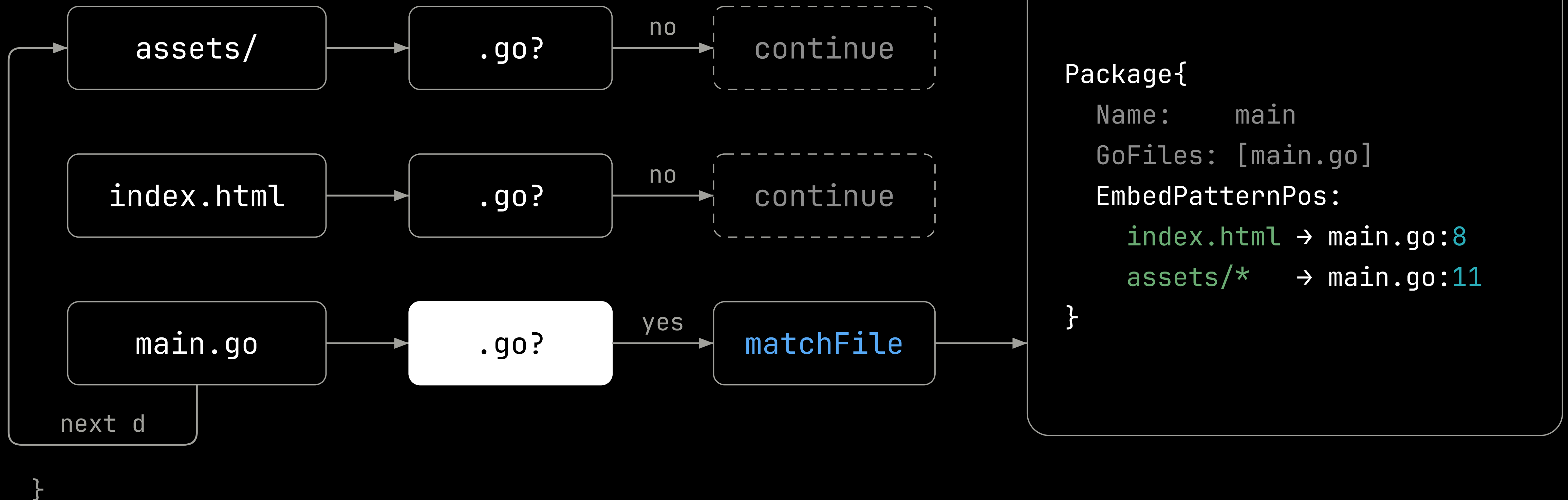
EmbedPatternPos:

index.html → main.go:8

assets/* → main.go:11

}

```
for _, d := range ctxt.readDir(p.Dir) {
```



```
1  func (ctxt *Context) Import(path string, srcDir string, mode ImportMode) (*Package, error) {
2      for _, d := range ctxt.readDir(p.Dir) {           // walk the package dir
3          name := d.Name()
4          if filepath.Ext(name) != ".go" {               // main.go, index.html, assets/...
5              continue
6          }
7          info, _ := ctxt.matchFile(p.Dir, name, ...)    // reads header, scans comments & embed
8          p.GoFiles = append(p.GoFiles, name)            // ["main.go"]
9          p.Name = info.parsed.Name.Name                 // main
10
11         for _, emb := range info.embeds {
12             // {"index.html": [main.go:5], "assets/*": [main.go:8]}
13             embedPos[emb.pattern] = append(embedPos[emb.pattern], emb.pos)
14         }
15     }
16 }
```

```
1  func (ctxt *Context) Import(path string, srcDir string, mode ImportMode) (*Package, error) {
2      for _, d := range ctxt.readDir(p.Dir) {           // walk the package dir
3          name := d.Name()
4          if filepath.Ext(name) != ".go" {              // main.go, index.html, assets/...
5              continue
6          }
7          info, _ := ctxt.matchFile(p.Dir, name, ...)   // reads header, scans comments & embed
8          p.GoFiles = append(p.GoFiles, name)           // ["main.go"]
9          p.Name = info.parsed.Name.Name                // main
10
11          for _, emb := range info embeds {
12              // {"index.html": [main.go:5], "assets/*": [main.go:8]}
```

```
1 func (ctxt *Context) Import(path string, srcDir string) error {
2     for _, d := range ctxt.readDir(p.Dir) {
3         name := d.Name()
4         if filepath.Ext(name) != ".go" {
5             continue
6         }
7         info, _ := ctxt.matchFile(p.Dir, name, ...)
```

```
1 func (ctxt *Context) Import(path string, srcDir
2     for _, d := range ctxt.readDir(p.Dir) {
3         name := d.Name()
4         if filepath.Ext(name) != ".go" {
5             continue
6         }
7         info, _ := ctxt.matchFile(p.Dir, name,
8         p.GoFiles = append(p.GoFiles, name)
9         p.Name = info.parsed.Name.Name
```

```
1 func (ctxt *Context) Import(path string, srcDir string, mode string) {
2     for _, d := range ctxt.readDir(p.Dir) { // walk
3         name := d.Name()
4         if filepath.Ext(name) != ".go" { // main
5             continue
6         }
7         info, _ := ctxt.matchFile(p.Dir, name, ...) // read
8         p.GoFiles = append(p.GoFiles, name) // ["ma
9         p.Name = info.parsed.Name.Name // main
10
11     for _, emb := range info.embeds {
12         // {"index.html": [main.go:5], "assets/*": [main
13         embedPos[emb.pattern] = append(embedPos[emb.patt
```

```
4         if filepath.Ext(name) != ".go" {                               // main.go, index.html, assets/...
5             continue
6         }
7         info, _ := ctxt.matchFile(p.Dir, name, ...) // reads header, scans comments &
8         p.GoFiles = append(p.GoFiles, name)         // ["main.go"]
9         p.Name = info.parsed.Name.Name              // main
10
11         for _, emb := range info.embeds {
12             // {"index.html": [main.go:5], "assets/*": [main.go:8]}
13             embedPos[emb.pattern] = append(embedPos[emb.pattern], emb.pos)
14         }
15     }
16 }
```

go build

Packages

go build

Lexer

go build

Patterns

cmd/compile

Parser

cmd/compile

Compiler

cmd/link

Linker

/	/	g	o	:	e	m	b	e	d		i	n	d	e	x	.	h	t	m	l
---	---	---	---	---	---	---	---	---	---	--	---	---	---	---	---	---	---	---	---	---

one character at a time →

//go:embed index.html

a comment is a real token

COMMENT

v	a	r		i	n	d	e	x	H	T	M	L		s	t	r	i	n	g	↵
---	---	---	--	---	---	---	---	---	---	---	---	---	--	---	---	---	---	---	---	---

the newline becomes a ;

var

VAR

indexHTML

IDENT

string

IDENT

;

SEMICOLON

/ / g o : e m b e d i n d e x . h t m l

one character at a time →

//go:embed index.html

a comment is a real token

COMMENT

v a r i n d e x H T M L s t r i n g ↵

the newline becomes a ;

var

VAR

indexHTML

IDENT

string

IDENT

;

SEMICOLON

/ / g o : e m b e d i n d e x . h t m l

one character at a time →

//go:embed index.html

a comment is a real token

COMMENT

v a r i n d e x H T M L s t r i n g ↵

the newline becomes a ;

var

VAR

indexHTML

IDENT

string

IDENT

;

SEMICOLON

/ / g o : e m b e d i n d e x . h t m l

one character at a time →

//go:embed index.html

a comment is a real token

COMMENT

v a r i n d e x H T M L s t r i n g ↵

the newline becomes a ;

var

indexHTML

string

;

VAR

IDENT

IDENT

SEMICOLON

```
1  if hasEmbed {
2      r.readRest() // keep going: read the rest of the file after all
3      sc.Init(file, r.buf, nil, scanner.ScanComments) // get comments as tokens
4
5      for {
6          pos, tok, lit := sc.Scan()
7          if tok == token.COMMENT && strings.HasPrefix(lit, "//go:embed") {
8              // in: `//go:embed index.html`
9              // out: {pattern: "index.html", pos: ...}
10             embs, _ := parseGoEmbed(fset, pos, lit)
11             info.embeds = append(info.embeds, embs...)
12         }
13     }
14 }
```

```
1  if hasEmbed {  
2      r.readRest() // keep goi  
3      sc.Init(file, r.buf, nil
```

```
1  if hasEmbed {
2      r.readRest() // keep going: read the rest of the file after all
3      sc.Init(file, r.buf, nil, scanner.ScanComments) // get comments as tokens
4
5      for {
6          pos, tok, lit := sc.Scan()
7          if tok == token.COMMENT && strings.HasPrefix(lit, "//go:embed") {
8              // in: `//go:embed index.html`
9              // out: {pattern: "index.html", pos: ...}
10             embs, _ := parseGoEmbed(fset, pos, lit)
11             info.embeds = append(info.embeds, embs...)
12         }
13     }
14 }
```

```
2      r.readRest() // keep going: read the rest of the file
3      sc.Init(file, r.buf, nil, scanner.Scanner{
4          // ...
5      })
6      for {
7          pos, tok, lit := sc.Scan()
8          if tok == token.COMMENT && strings.HasPrefix(lit, `//go:embed index.html`) {
9              // out: {pattern: "index.html"}
10             embs, _ := parseGoEmbed(fset, lit)
```

```
1  if hasEmbed {
2      r.readRest() // keep going: read the rest of the file after all
3      sc.Init(file, r.buf, nil, scanner.ScanComments) // get comments as tokens
4
5      for {
6          pos, tok, lit := sc.Scan()
7          if tok == token.COMMENT && strings.HasPrefix(lit, "//go:embed") {
8              // in: `//go:embed index.html`
9              // out: {pattern: "index.html", pos: ...}
10             embs, _ := parseGoEmbed(fset, pos, lit)
11             info.embeds = append(info.embeds, embs...)
12         }
13     }
14 }
```

```
3      sc.Init(file, r.buf, nil, scanner.ScanComments) // get comments
4
5      for {
6          pos, tok, lit := sc.Scan()
7          if tok == token.COMMENT && strings.HasPrefix(lit, "//go") {
8              // in: `//go:embed index.html`
9              // out: {pattern: "index.html", pos: ...}
10             embs, _ := parseGoEmbed(fset, pos, lit)
11             info.embeds = append(info.embeds, embs...)
12         }
13     }
14 }
```

```
1  []fileEmbed{
2      {pattern: "index.html", pos: main.go:5:12},
3      {pattern: "assets/*", pos: main.go:8:12},
4  }
```

```
1 []fileEmbed{
2     {pattern: "index.html", pos: main.go:5:12},
3     {pattern: "assets/*", pos: main.go:8:12},
4 }
```

```
1  []fileEmbed{
2      {pattern: "index.html", pos: main.go:5:12},
3      {pattern: "assets/*", pos: main.go:8:12},
4  }
```

```
1  []fileEmbed{
2      {pattern: "index.html", pos: main.go:5:12},
3      {pattern: "assets/*", pos: main.go:8:12},
4  }
```

go build

Packages

go build

Lexer

go build

Patterns

cmd/compile

Parser

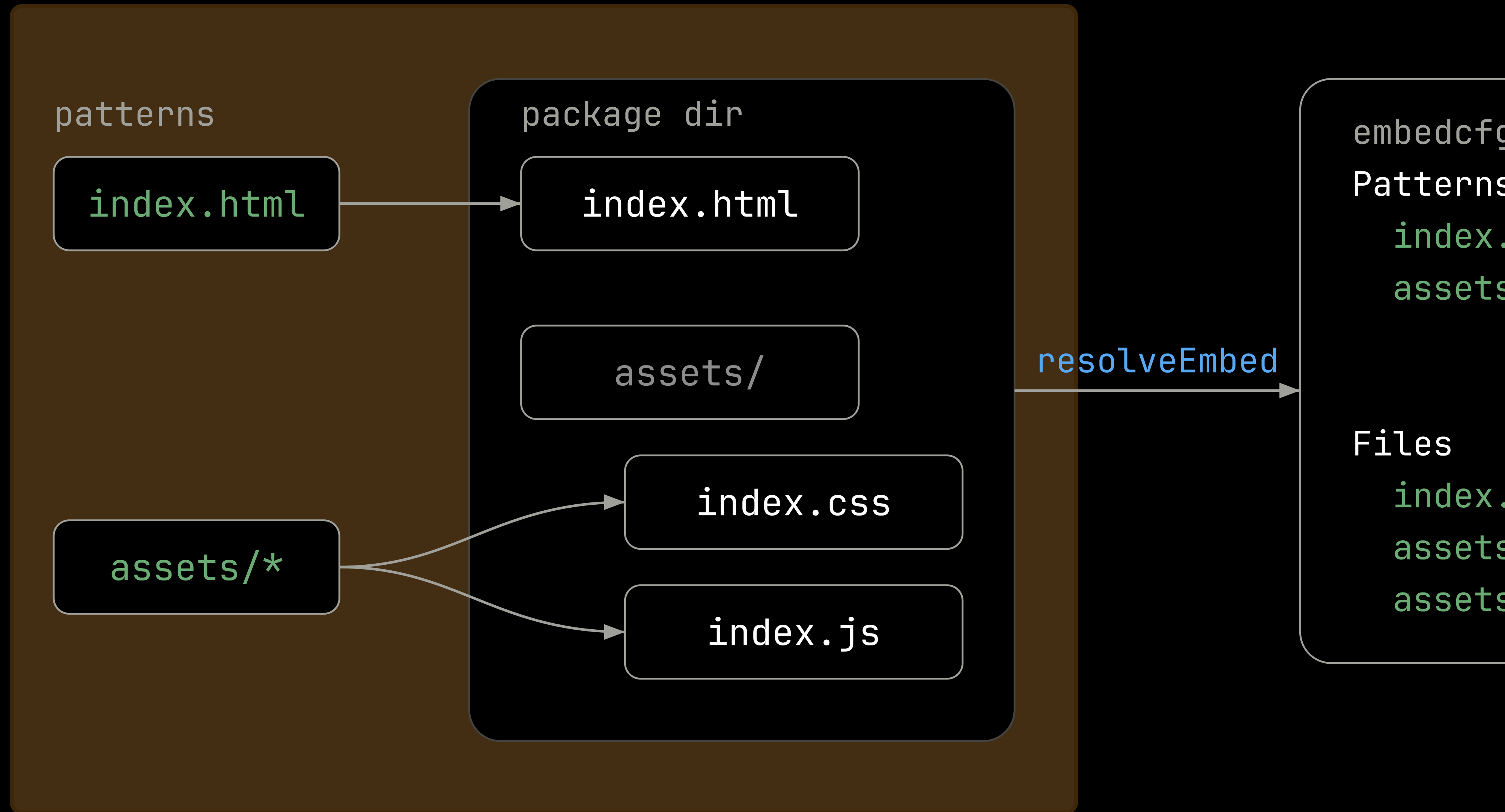
cmd/compile

Compiler

cmd/link

Linker





s

resolveEmbed



inde

asse

Files

inde

embedcfg

Patterns

`index.html` → [`index.html`]

`assets/*` → [`assets/index.css`,
`assets/index.js`]

Files

`index.html` → `~/app/index.html`

`assets/index.css` → `~/app/assets/index.css`

`assets/index.js` → `~/app/assets/index.js`

resolveEmbed

[5]

-embedcfg

embedcfg

Patterns

`index.html` → `[index.html]`

`assets/*` → `[assets/index.css,`
`assets/index.js]`

Files

`index.html` → `~/app/index.html`

`assets/index.css` → `~/app/assets/index.css`

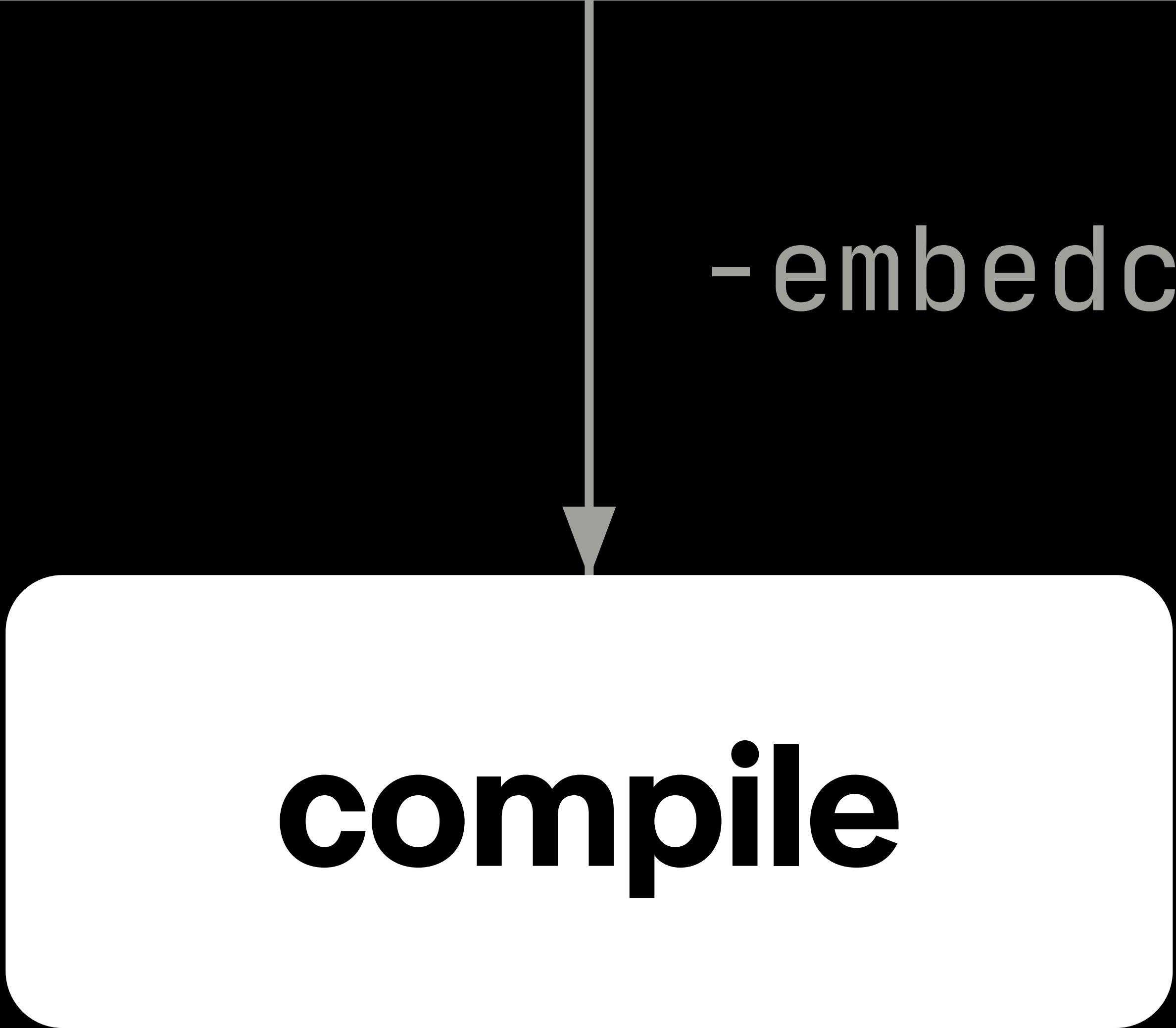
`index.html` → `[index.html]`
`assets/*` → `[assets/index.css,`
`assets/index.js]`

Files

`index.html` → `~/app/index.html`
`assets/index.css` → `~/app/assets/index.css`
`assets/index.js` → `~/app/assets/index.js`

-embedcfg

-embedcfg



compile

```
1  {
2    "Patterns": {
3      "assets/*": [
4        "assets/index.css",
5        "assets/index.js"
6      ],
7      "index.html": [
8        "index.html"
9      ]
10   },
11   "Files": {
12     "assets/index.css": "/home/user/project/assets/index.css",
13     "assets/index.js": "/home/user/project/assets/index.js",
14     "index.html": "/home/user/project/index.html"
15   }
16 }
```

```
1  {
2    "Patterns": {
3      "assets/*": [
4        "assets/index.css",
5        "assets/index.js"
6      ],
7      "index.html": [
8        "index.html"
9      ]
10   },
11   "Files": {
12     "assets/index.css": "/home/user/project/assets/index.css",
13     "assets/index.js": "/home/user/project/assets/index.js",
14     "index.html": "/home/user/project/index.html"
15   }
16 }
```

```
1  {
2    "Patterns": {
3      "assets/*": [
4        "assets/index.css",
5        "assets/index.js"
6      ],
7      "index.html": [
8        "index.html"
9      ]
10   },
11   "Files": {
12     "assets/index.css": "/home/user/project/assets/index.css",
13     "assets/index.js": "/home/user/project/assets/index.js",
14     "index.html": "/home/user/project/index.html"
15   }
16 }
```

```
1  {
2    "Patterns": {
3      "assets/*": [
4        "assets/index.css",
5        "assets/index.js"
6      ],
7      "index.html": [
8        "index.html"
9      ]
10   },
11   "Files": {
12     "assets/index.css": "/home/user/project/assets/index.css",
13     "assets/index.js": "/home/user/project/assets/index.js",
14     "index.html": "/home/user/project/index.html"
15   }
16 }
```

```
1  {
2    "Patterns": {
3      "assets/*": [
4        "assets/index.css",
5        "assets/index.js"
6      ],
7      "index.html": [
8        "index.html"
9      ]
10   },
11   "Files": {
12     "assets/index.css": "/home/user/project/assets/index.css",
13     "assets/index.js": "/home/user/project/assets/index.js",
14     "index.html": "/home/user/project/index.html"
15   }
16 }
```

```
1  {
2    "Patterns": {
3      "assets/*": [
4        "assets/index.css",
5        "assets/index.js"
6      ],
7      "index.html": [
8        "index.html"
9      ]
10   },
11   "Files": {
12     "assets/index.css": "/home/user/project/assets/index.css",
13     "assets/index.js": "/home/user/project/assets/index.js",
14     "index.html": "/home/user/project/index.html"
15   }
16 }
```

embedcfg

embedcfg

embedcfg



go build

Packages

go build

Lexer

go build

Patterns

cmd/compile

Parser

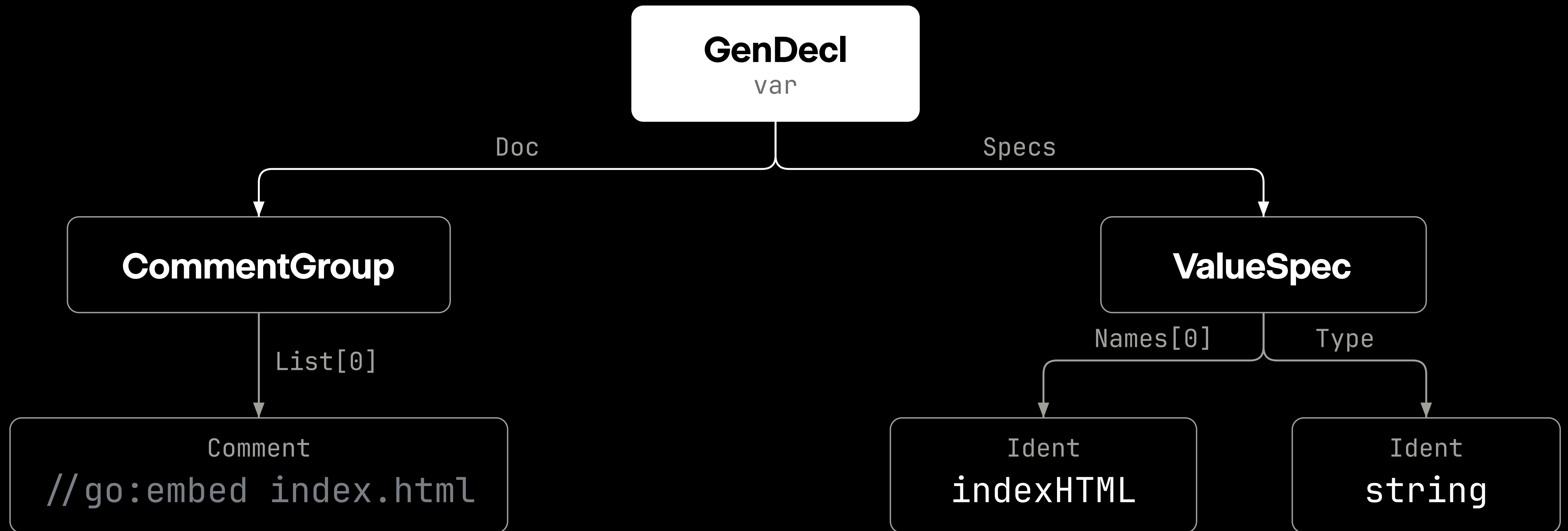
cmd/compile

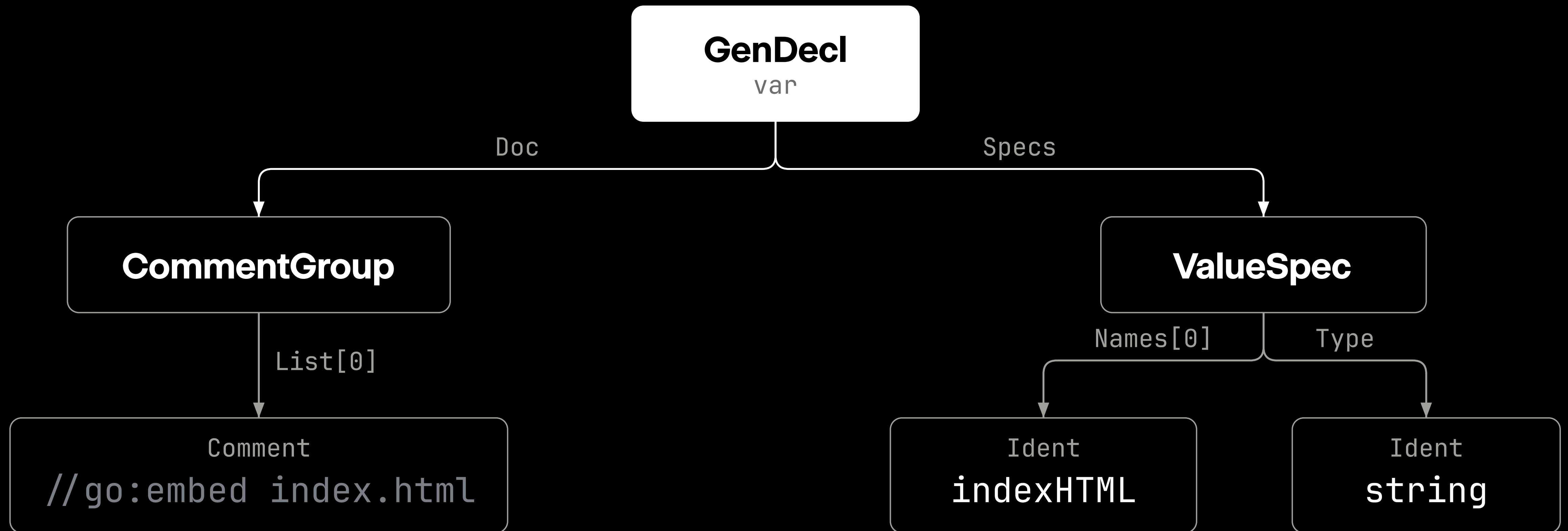
Compiler

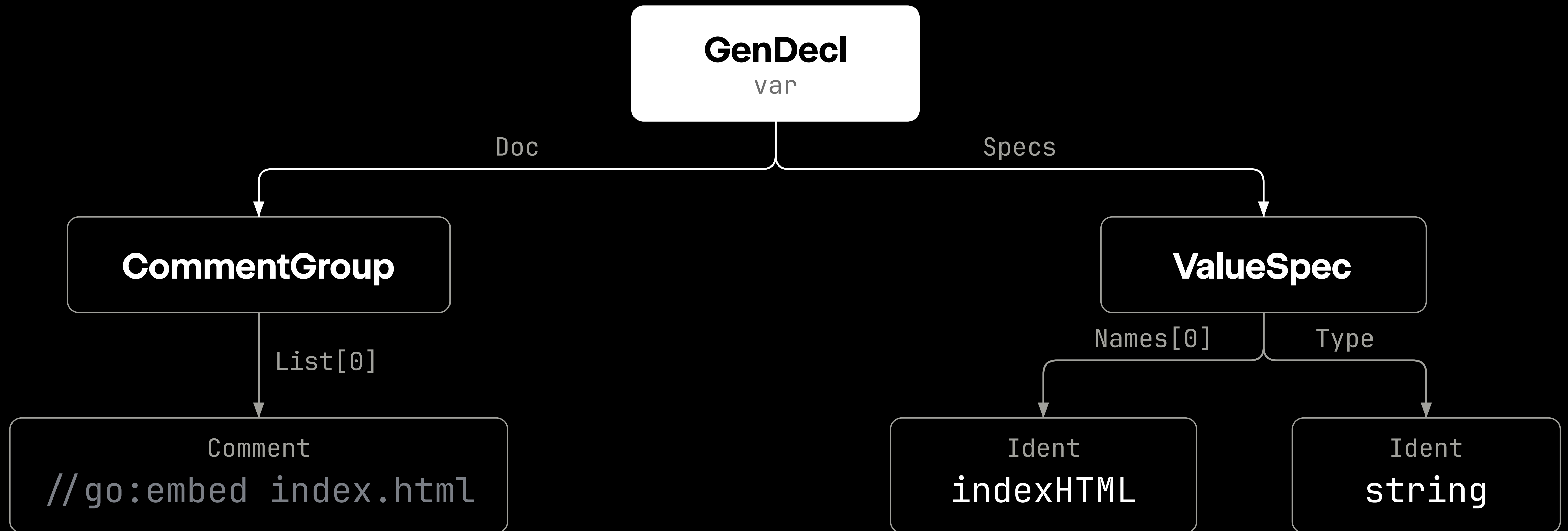
cmd/link

Linker

```
1 //go:embed index.html
2 var indexHTML string
```

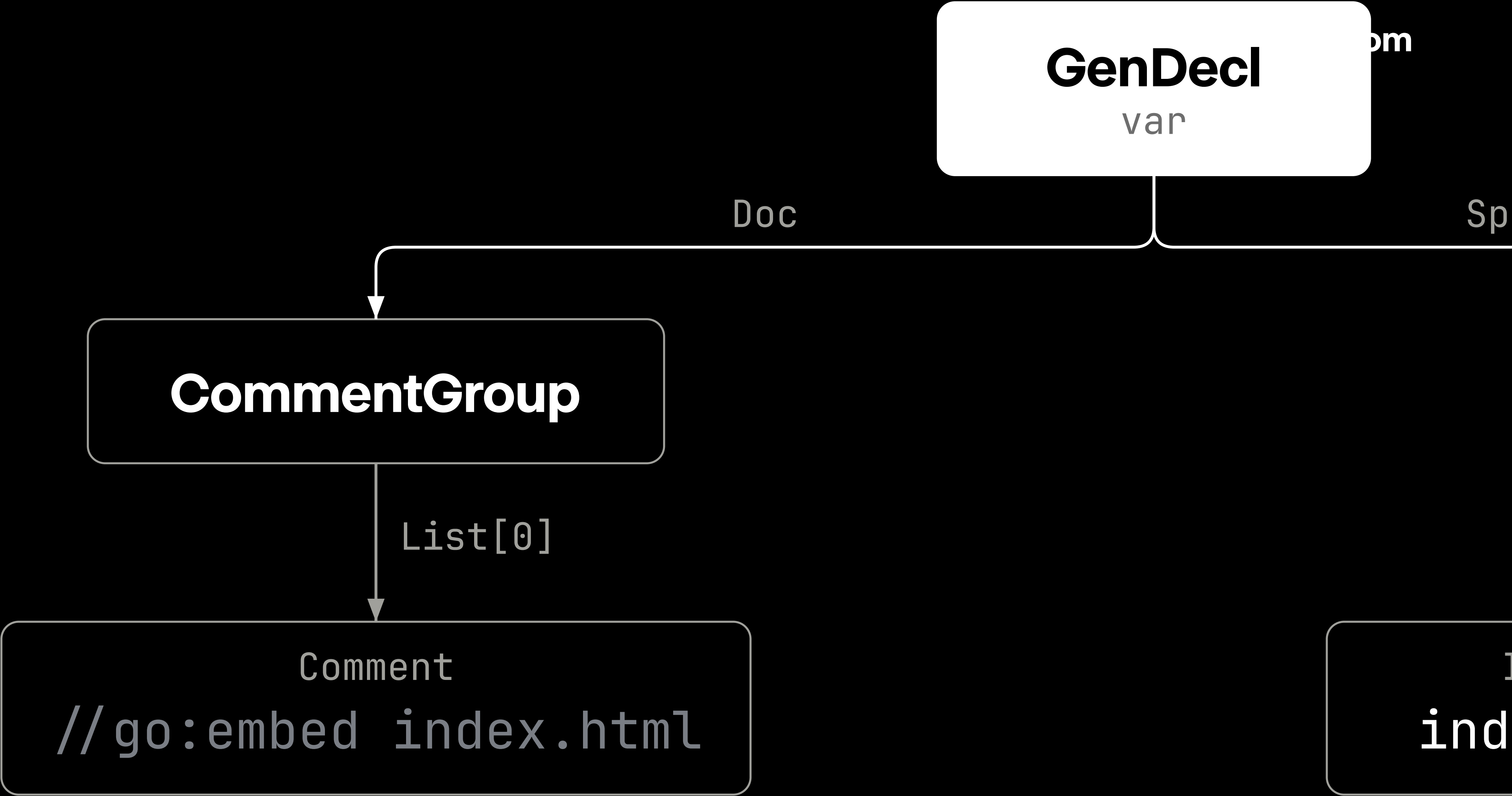






GenDecl

var



GenDecl

var

Specs

ValueSpec

Names[0]

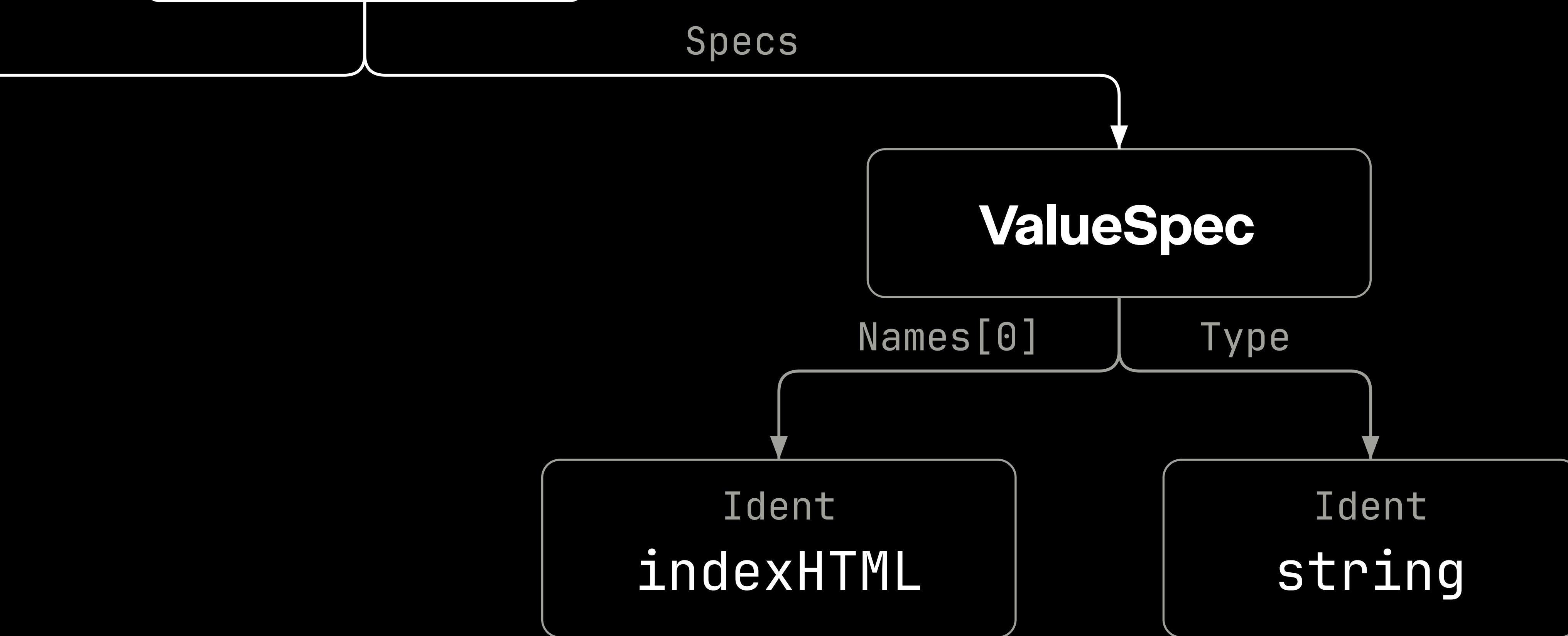
Type

Ident

indexHTML

Ident

string



```
1  1: *ast.GenDecl {
2    . Doc: *ast.CommentGroup {
3      . List: []*ast.Comment (len = 1) {
4        . 0: *ast.Comment {
5          . Slash: main.go:8:1
6          . Text: "//go:embed index.html"
7        }
8      }
9    }
10   . Specs: []*ast.Spec (len = 1) {
11     . 0: *ast.ValueSpec {
12       . Names: []*ast.Ident (len = 1) {
13         . 0: *ast.Ident {
14           . NamePos: main.go:9:5
15           . Name: "indexHTML"
16         }
17       }
18       . Type: *ast.Ident {
19         . NamePos: main.go:9:15
20         . Name: "string"
21       }
22     }
23   }
24 }
```

```
1      1: *ast.GenDecl {
2      .   Doc: *ast.CommentGroup {
3      .   .   List: []*ast.Comment (
4      .   .   .   0: *ast.Comment {
```

```
1  1: *ast.GenDecl {
2    . Doc: *ast.CommentGroup {
3      . List: []*ast.Comment (len = 1) {
4        . . 0: *ast.Comment {
5          . . . Slash: main.go:8:1
6          . . . Text: "//go:embed index.html"
7          . . . }
8        . . }
9      . }
10 . Specs: []*ast.Spec (len = 1) {
11 . . 0: *ast.ValueSpec {
```

```
2      . Doc: *ast.CommentGroup {
3      .   . List: []*ast.Comment (len = 1) {
4      .   .   . 0: *ast.Comment {
5      .   .   .   . Slash: main.go:8:1
6      .   .   .   . Text: "//go:embed index.html"
7      .   .   .   }
8      .   .   }
9      .   }
10     . Specs: []ast.Spec (len = 1) {
11     .   . 0: *ast.ValueSpec {
```

```
9      .  }
10     .  Specs: []ast.Spec (len = 1) {
11     .    .  0: *ast.ValueSpec {
12     .    .    .  Names: []ast.Ident (len = 1) {
13     .    .    .    .  0: *ast.Ident {
14     .    .    .    .    .  NamePos: main.go:9:5
15     .    .    .    .    .  Name: "indexHTML"
16     .    .    .    .  }
17     .    .    .  }
18     .    .    .  Type: *ast.Ident {
19     .    .    .    .  NamePos: main.go:9:15
20     .    .    .    .  Name: "string"
21     .    .    .  }
22     .    .  }
23     .  }
24 }
```

```
10 . Specs: []ast.Spec (len = 1) {
11 . . 0: *ast.ValueSpec {
12 . . . Names: []*ast.Ident (len = 1) {
13 . . . . 0: *ast.Ident {
14 . . . . . NamePos: main.go:9:5
15 . . . . . Name: "indexHTML"
16 . . . . }
17 . . . }
18 . . Type: *ast.Ident {
19 . . . NamePos: main.go:9:15
```

```
16      . . . . }
17      . . . }
18      . . . Type: *ast.Ident {
19      . . . . NamePos: main.go:9:15
20      . . . . Name: "string"
21      . . . }
22      . . }
23      . }
```

```
1  1: *ast.GenDecl {
2    . Doc: *ast.CommentGroup {
3      . . List: []*ast.Comment (len = 1) {
4        . . . 0: *ast.Comment {
5          . . . . Slash: main.go:8:1
6          . . . . Text: "//go:embed index.html"
7          . . . }
8        . . }
9      . }
10   . Specs: []*ast.Spec (len = 1) {
11     . . 0: *ast.ValueSpec {
12       . . . Names: []*ast.Ident (len = 1) {
13         . . . . 0: *ast.Ident {
14           . . . . . NamePos: main.go:9:5
15           . . . . . Name: "indexHTML"
16           . . . . }
17         . . . }
18         . . . Type: *ast.Ident {
19           . . . . NamePos: main.go:9:15
20           . . . . Name: "string"
21           . . . }
22         . . }
23       . }
24     }
```

go build

Packages

go build

Lexer

go build

Patterns

cmd/compile

Parser

cmd/compile

Compiler

cmd/link

Linker

```
1 func checkEmbed(decl *syntax.VarDecl, haveEmbed, withinFunc bool) error {
2     switch {
3     case !haveEmbed: // The file never imported "embed".
4         return errors.New(`go:embed only allowed in Go files that import "embed"`)
5     case len(decl.NameList) > 1: // var a, b string
6         return errors.New("go:embed cannot apply to multiple vars")
7     case decl.Values != nil: // var s string = "hello"
8         return errors.New("go:embed cannot apply to var with initializer")
9     case withinFunc: // Declared inside a func body.
10        return errors.New("go:embed cannot apply to var inside func")
11    }
12    return nil
13 }
```

```
1  func checkEmbed(decl *syntax.VarDecl, haveEmbed, withinFunc bool) error {
2      switch {
3      case !haveEmbed: // The file never imported "embed".
4          return errors.New(`go:embed only allowed in Go files that import "embed"`)
5      case len(decl.NameList) > 1: // var a, b string
6          return errors.New("go:embed cannot apply to multiple vars")
7      case decl.Values != nil: // var s string = "hello"
8          return errors.New("go:embed cannot apply to var with initializer")
9      case withinFunc: // Declared inside a func body.
10         return errors.New("go:embed cannot apply to var inside func")
11     }
12     return nil
13 }
```

`var indexHTML string`

`a []byte` adds a cap field

on disk



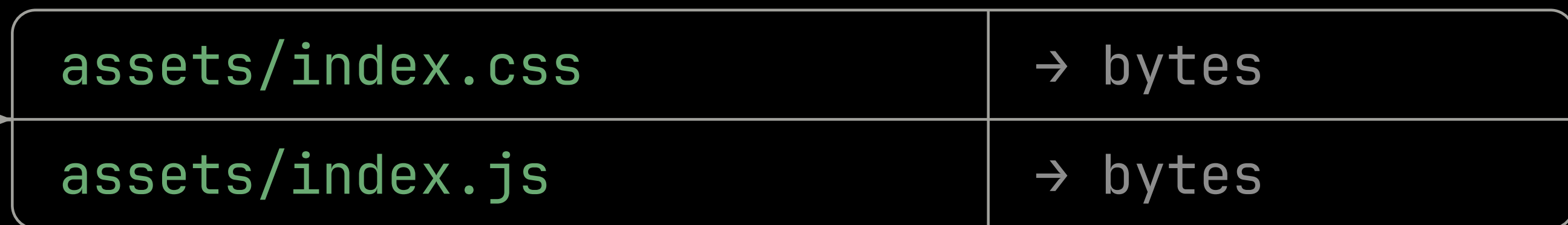
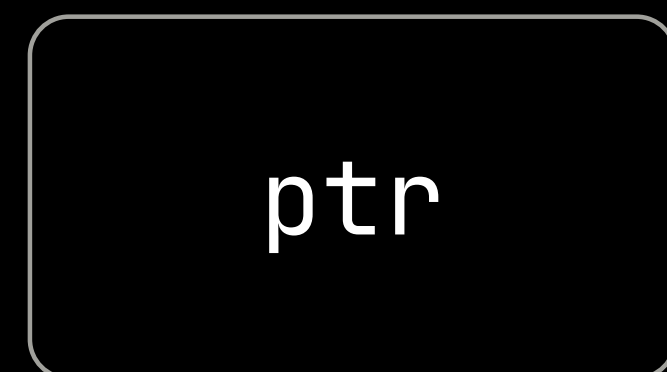
`read once, Files["index.html"]`

< h 1 > H e l l o < / h 1 >

read-only data · 14 bytes

`var static embed.FS`

`[]file` · one record per file



```
var indexHTML string
```

ptr	len 14
-----	--------

on disk

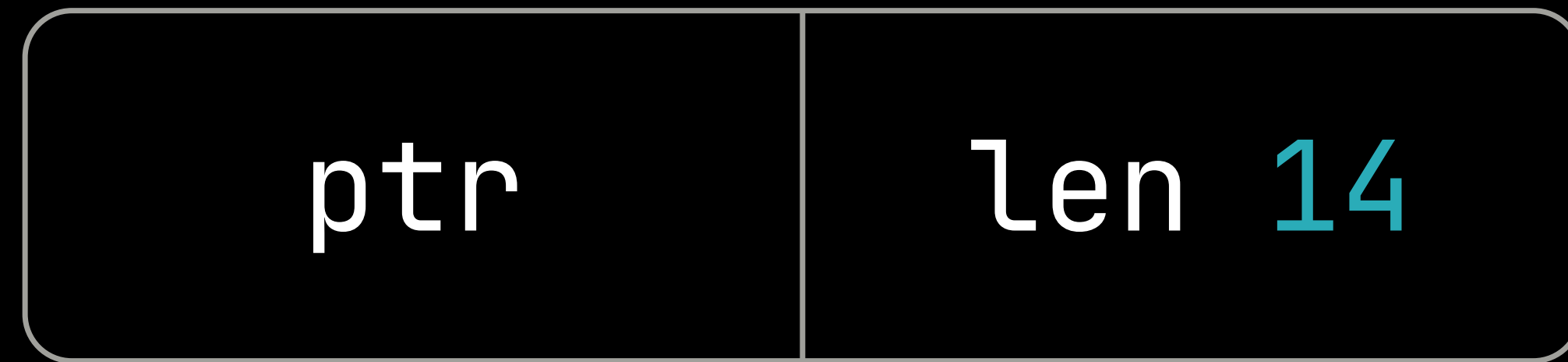
~/app/index.html

"]

```
var indexHTML string
```

```
a []byte adds a cap field
```

ainsleyclark.com



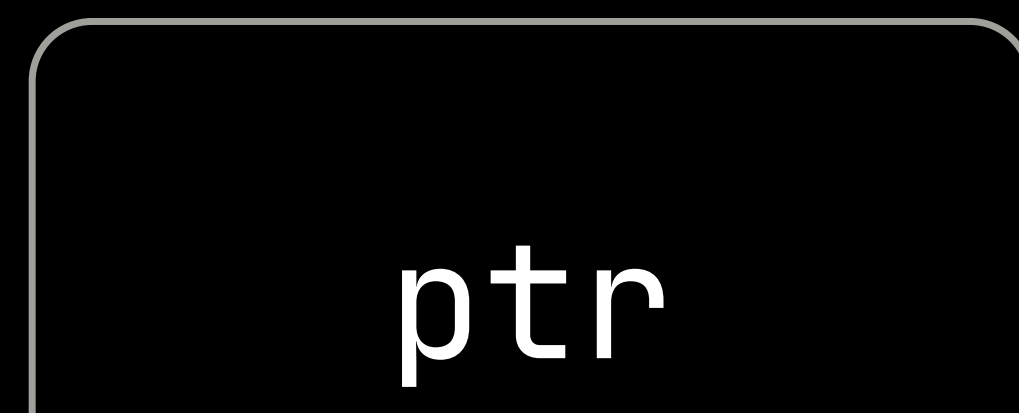
```
read once, Files["index.html"]
```

`< h 1 > H e l l o < / h 1 >`

```
read-only data · 14 bytes
```

```
var static embed.FS
```

```
[]file · one record per file
```





read once, Files["index.html"]

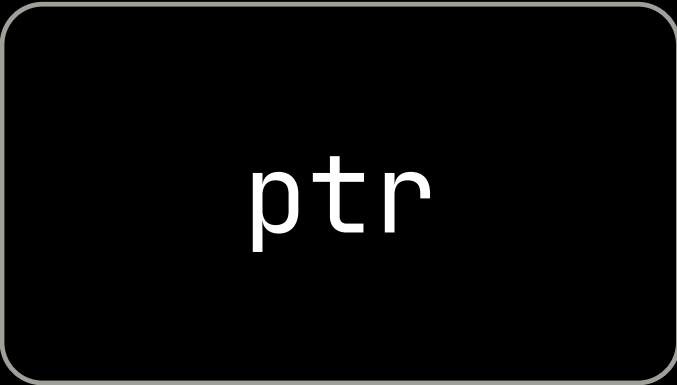
ainsleyclark.com

< h 1 > H e l l o < / h 1 >

read-only data · 14 bytes

var static embed.FS

[]file · one record per file



assets/index.css	→ bytes
assets/index.js	→ bytes

```
1 // WriteEmbed, for our `var indexHTML string`.
2 case embedString, embedBytes:
3     file := files[0] // "index.html", from the embedcfg JSON
4     path := base.Flag.Cfg.Embed.Files[file] // "/home/user/project/index.html"
5     fsym, size, _ := fileStringSym(v.Pos(), path, ...) // read it, name the bytes
6
7     sym, off := v.Linksym(), 0 // indexHTML itself
8     off = objw.SymPtr(sym, off, fsym, 0) // field 1: pointer to those bytes
9     off = objw.Uintptr(sym, off, uint64(size)) // field 2: length
10    if kind == embedBytes {
11        objw.Uintptr(sym, off, uint64(size)) // field 3: cap, because a slice has one
12    }
13 case embed.FS:
14     // See source files below...
```

```
1 // WriteEmbed, for our `var indexHTML string`.
2 case embedString, embedBytes:
3     file := files[0] // "index.html", from the embedcfg JSON
4     path := base.Flag.Cfg.Embed.Files[file] // "/home/user/project/index.html"
5     fsym, size, _ := fileStringSym(v.Pos(), path, ...) // read it, name the bytes
6
7     sym, off := v.Linksym(), 0 // indexHTML itself
8     off = objw.SymPtr(sym, off, fsym, 0) // field 1: pointer to those bytes
9     off = objw.Uintptr(sym, off, uint64(size)) // field 2: length
10    if kind == embedBytes {
11        objw.Uintptr(sym, off, uint64(size)) // field 3: cap, because a slice has one
12    }
13 case embed.FS:
14     // See source files below...
```

```
1 // WriteEmbed, for our `var indexH
2 case embedString, embedBytes:
3     file := files[0]
4     path := base.Flag.Cfg.Embed.Fi
5     fsym, size, _ := fileStringSymC
```

```
1 // WriteEmbed, for our `var indexHTML string`.
2 case embedString, embedBytes:
3     file := files[0] // "ind
4     path := base.Flag.Cfg.Embed.Files[file] // "/ho
5     fsym, size, _ := fileStringSym(v.Pos(), path, .
6
7     sym, off := v.Linksym(), 0 // i
8     off = objw.SymPtr(sym, off, fsym, 0) // f
```

```
1 // WriteEmbed, for our `var indexHTML string`.
2 case embedString, embedBytes:
3     file := files[0] // "index.html", from
4     path := base.Flag.Cfg.Embed.Files[file] // "/home/user/proje
5     fsym, size, _ := fileStringSym(v.Pos(), path, ...) // read i
6
7     sym, off := v.Linksym(), 0 // indexHTML itse
8     off = objw.SymPtr(sym, off, fsym, 0) // field 1: point
9     off = objw.Uintptr(sym, off, uint64(size)) // field 2: lengt
10    if kind == embedBytes {
11        objw.Uintptr(sym, off, uint64(size)) // field 3: cap,
```

```
1 // WriteEmbed, for our `var indexHTML string`.
2 case embedString, embedBytes:
3     file := files[0] // "index.html", from
4     path := base.Flag.Cfg.Embed.Files[file] // "/home/user/proje
5     fsym, size, _ := fileStringSym(v.Pos(), path, ...) // read i
6
7     sym, off := v.Linksym(), 0 // indexHTML itse
8     off = objw.SymPtr(sym, off, fsym, 0) // field 1: point
9     off = objw.Uintptr(sym, off, uint64(size)) // field 2: lengt
10    if kind == embedBytes {
11        objw.Uintptr(sym, off, uint64(size)) // field 3: cap,
```

```
4      path := base.Flag.Cfg.Embed.Files[file] //
5      fsym, size, _ := fileStringSym(v.Pos(), pat
6
7      sym, off := v.Linksym(), 0
8      off = objw.SymPtr(sym, off, fsym, 0)
9      off = objw.Uintptr(sym, off, uint64(size))
10     if kind == embedBytes {
11         objw.Uintptr(sym, off, uint64(size))
12     }
```

```
4      path := base.Flag.Cfg.Embed.Files[file] // "/home/
5      fsym, size, _ := fileStringSym(v.Pos(), path, ...)
6
7      sym, off := v.Linksym(), 0                // index
8      off = objw.SymPtr(sym, off, fsym, 0)      // field
9      off = objw.Uintptr(sym, off, uint64(size)) // field
10     if kind == embedBytes {
11         objw.Uintptr(sym, off, uint64(size)) // field
12     }
13     case embed.FS:
14         // See source files below...
```

Object archives

go build

Packages

go build

Lexer

go build

Patterns

cmd/compile

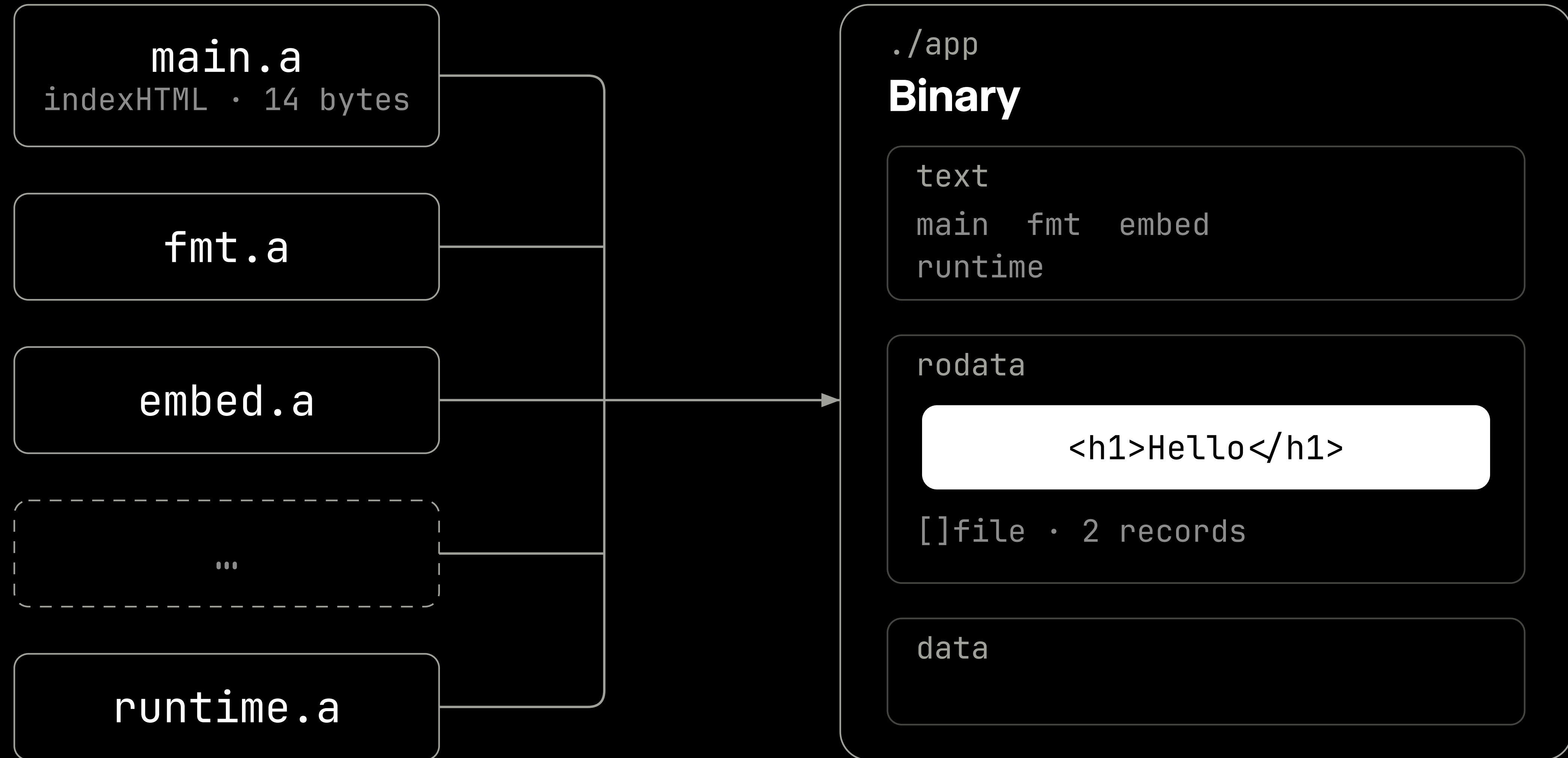
Parser

cmd/compile

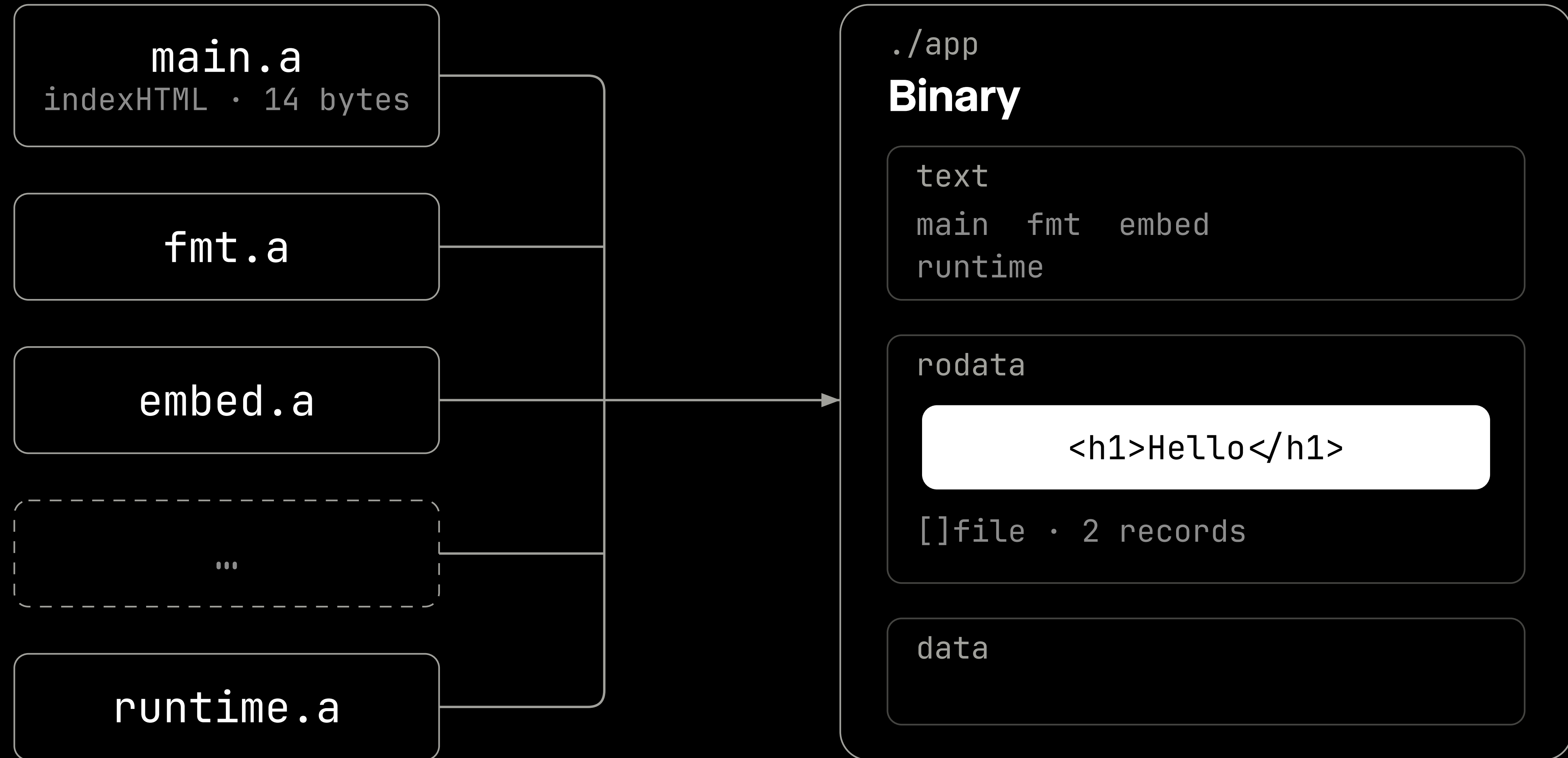
Compiler

cmd/link

Linker



the Go runtime, added here



the Go runtime, added here

main.a

indexHTML · 14 bytes

fmt.a

embed.a

...

runtime.a

the Go runtime, added here

./app

Binary

text

main fmt embed

runtime

rodata

<h1>Hello</h1>

[]file · 2 records

data

main.a
indexHTML · 14 bytes

fmt.a

embed.a

...

runtime.a

the Go runtime, added here

./app

Binary

text

main fmt embed
runtime

rodata

<h1>Hello</h1>

[]file · 2 records

data



04.

WHAT YOU TAKE ON

The tradeoffs

```
1 $ strings myapp | grep -o 'API_KEY=[^ ]*'  
2 API_KEY=sk_live_4eC39HqLyjWDarjtT1zdp7dc
```

```
1 $ strings myapp | grep -o 'API_KEY=[^ ]*'  
2 API_KEY=sk_live_4eC39HqLyjWDarjtT1zdp7dc
```

```
1 //go:embed all:assets
2 var assets embed.FS
3
4 http.Handle("/", http.FileServer(http.FS(assets)))
5
6 // $ curl localhost:8080/assets/.env
7 // API_KEY=sk_live_4eC39HqLyjWDarjtT1zdp7dc
```

```
1 //go:embed all:assets
2 var assets embed.FS
3
```

```
2  var assets embed.FS
3
4  http.Handle("/", http.FileServer(http.FS(assets)))
5
6  // $ curl localhost:8080/assets/.env
7  // API_KEY=sk_live_4eC39HqLyjWDarjtT1zdp7dc
```

Rebuild to change a byte

```
1 //go:embed assets/*
2 var assets embed.FS
3
4 func assetsFS(development bool) fs.FS {
5     if development {
6         return os.DirFS(".")
7     }
8     return assets
9 }
```

Testing

```
1 func TestAssetsArePresent(t *testing.T) {  
2     // Extras are ignored, so adding a file needs no test change.  
3     err := fstest.TestFS/assets,  
4         "assets/index.css",  
5     )  
6     if err != nil {  
7         t.Fatalf("files are not found: %v", err)  
8     }  
9 }
```

```
1 func TestAssetsArePresent(t *testing.T)
2     // Extras are ignored, so adding a
3     err := fstest.TestFS(assets,
4         "assets/index.css",
5     )
6     if err != nil {
7         t.Fatalf("files are not found:
```

```
1 func TestAssetsArePresent(t *testing.T) {  
2     // Extras are ignored, so adding a file needs no test change.  
3     err := fstest.TestFS/assets,  
4         "assets/index.css",  
5     )  
6     if err != nil {  
7         t.Fatalf("files are not found: %v", err)  
8     }  
9 }
```

```
1 // Take the interface, not the implementation.
2 func NewServer(assets fs.FS) *Server {
3     return &Server{assets: assets}
4 }
5
6 func TestIndexHandlesAnEmptyTemplate(t *testing.T) {
7     srv := NewServer(fstest.MapFS{
8         "index.html": &fstest.MapFile{Data: []byte("")},
9     })
10
11     // No files on disk, no embed, no build.
12     if err := srv.Render(io.Discard, "index.html"); err == nil {
13         t.Fatal("expected an error for an empty template")
14     }
15 }
```

```
1 // Take the interface, not the implementation.
2 func NewServer(assets fs.FS) *Server {
3     return &Server{assets: assets}
4 }
5
6 func TestIndexHandlesAnEmptyTemplate(t *testing.T) {
7     srv := NewServer(fstest.MapFS{
```

```
2 func NewServer(assets fs.FS) *Server {
3     return &Server{assets: assets}
4 }
5
6 func TestIndexHandlesAnEmptyTemplate(t *testing.T) {
7     srv := NewServer(fstest.MapFS{
8         "index.html": &fstest.MapFile{Data: []byte("")},
9     })
10
11     // No files on disk, no embed, no build.
12     if err := srv.Render(io.Discard, "index.html"); err == nil {
13         t.Fatalf("expected an error for an empty template")
14     }
```

```
6 func TestIndexHandlesAnEmptyTemplate(t *testing.T) {
7     srv := NewServer(fstest.MapFS{
8         "index.html": &fstest.MapFile{Data: []byte("")},
9     })
10
11     // No files on disk, no embed, no build.
12     if err := srv.Render(io.Discard, "index.html"); err == nil {
13         t.Fatal("expected an error for an empty template")
14     }
15 }
```

```
1 // Take the interface, not the implementation.
2 func NewServer(assets fs.FS) *Server {
3     return &Server{assets: assets}
4 }
5
6 func TestIndexHandlesAnEmptyTemplate(t *testing.T) {
7     srv := NewServer(fstest.MapFS{
8         "index.html": &fstest.MapFile{Data: []byte("")},
9     })
10
11     // No files on disk, no embed, no build.
12     if err := srv.Render(io.Discard, "index.html"); err == nil {
13         t.Fatal("expected an error for an empty template")
14     }
15 }
```

Recap

It can look scary...

It can look scary...

Thanks

