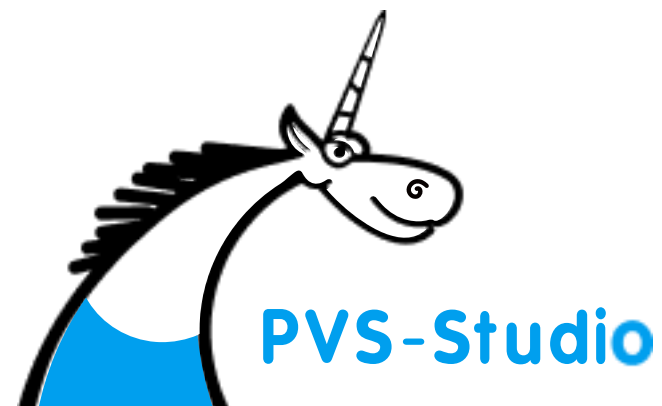






Go vet can't go: How to create your own Go analyzer?



Gleb Aslamov
Senior Developer Advocate



Gleb Aslamov

Senior Developer Advocate

- C-Sharper, Gopher
- Static analyzer developer
- I look for bugs in code and share my findings



PVS-Studio

Static analysis in the world of Go



Static analysis in Go

5

- Static analysis is the process of detecting flaws, bugs, and potential vulnerabilities in the source code of programs. Static analysis can be thought of as an automated code review process.

Static analysis in Go

6

- Static analysis is the process of detecting flaws, bugs, and potential vulnerabilities in the source code of programs. Static analysis can be thought of as an automated code review process.
- But I think you know this better than I do...
Right?

A bit of the basics

7

- go vet *as* part of the toolchain

- go vet *is* part of the toolchain

No, not yours

The Go toolchain Go, it's not a separate linter

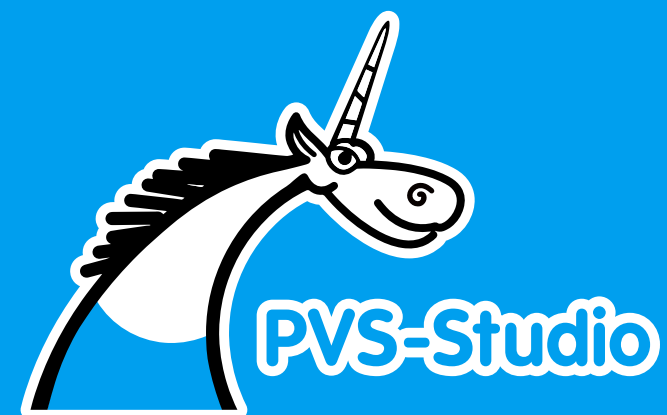
- go vet *is* part of the toolchain

No, not yours

The Go toolchain Go, it's not a separate linter

- Have you run go build? go test? You're a confident static analyzer user!

You're already using it



Synthetic example

11

```
if A == B {  
    if A == B {  
        . . .  
    }  
}
```

Synthetic example

12

```
if A == B {  
    if A == B {  
        . . .  
    }  
}
```

PVS-Studio warning:

V8020. Recurring check. This condition was already verified on a previous line.

- He-he, it's such a silly error, I definitely won't misstep here!

Synthetic example

13

```
if A == B {  
    if A == B {  
        . . .  
    }  
}
```

PVS-Studio warning:

V8020. Recurring check. This condition was already verified on a previous line.

- He-he, it's such a silly error, I definitely won't misstep here!
...Right?

Example from the Incus project

14

```
err = p.Save(pidPath)
if err != nil {
    err2 := p.Stop()
    if err != nil {
        return fmt.Errorf("...: %s: %s", err, err2)
    }
    ...
}
```

Example from the Incus project

15

```
err = p.Save(pidPath)
if err != nil {
    err2 := p.Stop()
    if err != nil {          // <= err2, maybe?
        return fmt.Errorf("...: %s: %s", err, err2)
    }
    ...
}
```

PVS-Studio warning:

V8020 Recurring check. The 'err != nil' condition was already verified on line 407

"What's the point of all this talk?
go vet can catch it!
Why do I need other tool/my own
tool?"



Can it...?

17

```
func rgb1(r float32, g float32, b float32) {  
    if r > 1 || g > 1 || r > 1 {  
        . . .  
    }  
}
```

Can it...?

18

```
func rgb1(r float32, g float32, b float32) {  
    if r > 1 || g > 1 || r > 1 {  
        . . .  
    }  
}
```

Of course it can!

```
go vet: redundant or: r > 1 || r > 1
```

Can it...?

19

```
func rgb1(r float32, g float32, b float32) {  
    if r > 1 || g > 1 || 1 < r {  
        . . .  
    }  
}
```

But what about
this...

Can it...?

20

```
func rgb1(r float32, g float32, b float32) {  
    if r > 1 || g > 1 || 1 < r {  
        . . .  
    }  
}
```

But what about
this...

Of course it can!

Can it...?

21

```
func rgb1(r float32, g float32, b float32) {  
    if r > 1 || g > 1 || 1 < r {  
        . . .  
    }  
}
```

But what about
this...

Of course it can!
...right?



go vet can't go

Can it...?

23

```
func rgb1(r float32, g float32, b float32) {  
    if r > 1 || g > 1 || 1 < r {  
        . . .  
    }  
}
```

But what about
this...

Of course it can't!

go vet: All good! 

Take a look at the real-world case

24

- Nuclei

```
func NewEntityParser(dir string) (*EntityParser, error) {
    cfg := &packages.Config{
        Mode:
            packages.NeedName      | packages.NeedFiles |
            packages.NeedImports | packages.NeedTypes |
            packages.NeedSyntax  | packages.NeedTypes |
            packages.NeedModule  | packages.NeedTypesInfo,
        . . . .
    }
}
```

Take a look at the real-world case

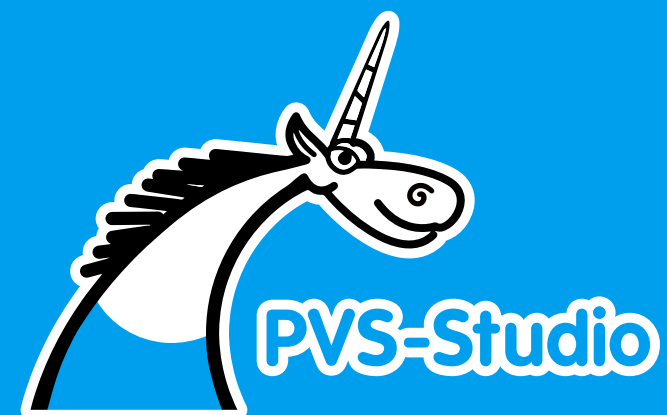
25

- Nuclei

```
func NewEntityParser(dir string) (*EntityParser, error) {  
    cfg := &packages.Config{  
        Mode:  
  
                packages.NeedName      | packages.NeedFiles |  
                packages.NeedImports | packages.NeedTypes |  
                packages.NeedSyntax  | packages.NeedTypes |  
                packages.NeedModule  | packages.NeedTypesInfo,  
  
        . . . .  
    }  
}
```

go vet: All good! 

"Well, that's probably a one-off case..."

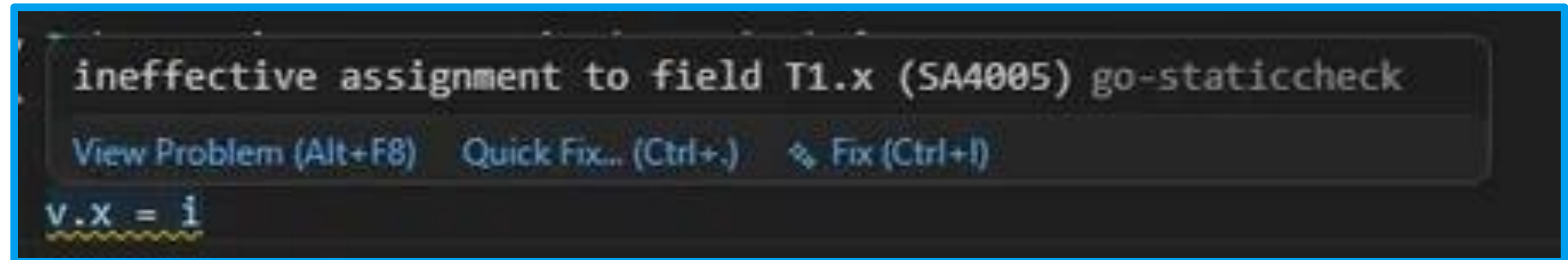


- Staticcheck

```
func (v T1) negative_range_string(s string) {  
    for i := range s {  
        fmt.Println(v.x)  
        v.x = i  
    }  
}
```

- Staticcheck

```
func (v T1) negative_range_string(s string) {  
    for i := range s {  
        fmt.Println(v.x)  
        v.x = i  
    }  
}
```



But is it actually a bug?

No!

- Another example with Staticcheck

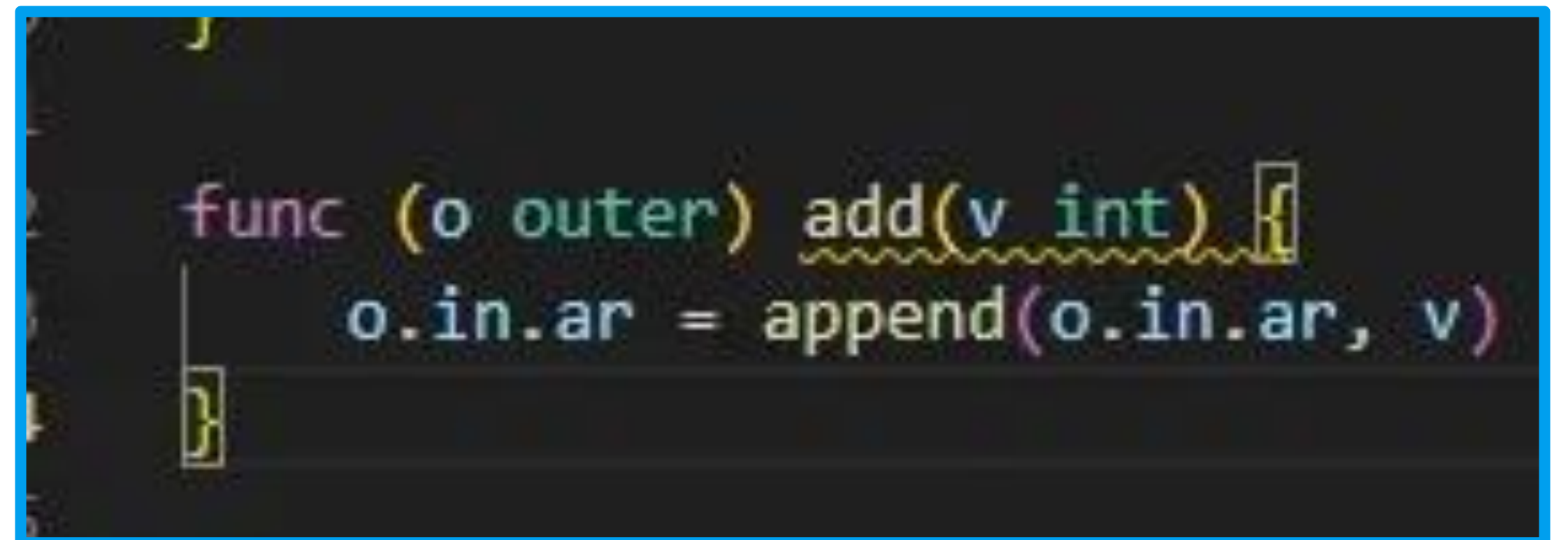
```
func (o outer) add(v int) {  
    o.in.ar = append(o.in.ar, v)  
}
```

False Negative

30

- Another example with Staticcheck

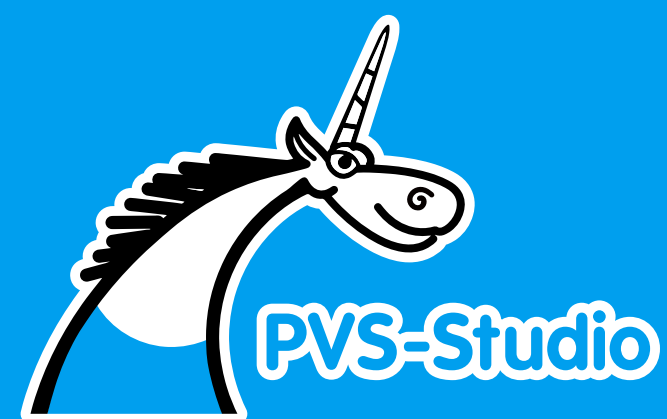
```
func (o outer) add(v int) {  
    o.in.ar = append(o.in.ar, v)  
}
```



But is it actually a bug?

Yes!

Let's set the goal!



Set the goal

32

Let's try to find the error

A method modifies a field of a receiver passed by value. Changes go into the copy; the original stays unchanged.

Set the goal

33

Let's try to find the error

A method modifies a field of a receiver passed by value. Changes go into the copy; the original stays unchanged.

That doesn't work

```
type Buffer struct {  
    data []byte  
}  
  
func (b Buffer) Append(value byte) {  
    b.data = append(b.data, value)  
}
```

Set the goal

34

Let's try to find the error

A method modifies a field of a receiver passed by value. Changes go into the copy; the original stays unchanged.

That doesn't work

```
type Buffer struct {  
    data []byte  
}  
  
func (b Buffer) Append(value byte) {  
    b.data = append(b.data, value)  
}
```

But this works

```
type Buffer struct {  
    data []byte  
}  
  
func (b *Buffer) Append(value byte) {  
    b.data = append(b.data, value)  
}
```

"Why do I need my own rules?"

35

+ Pros

- No need to choose one
- Customize for your needs
- Get a full control
- Set up for projects' specifics

"Why do I need my own rules?"

36

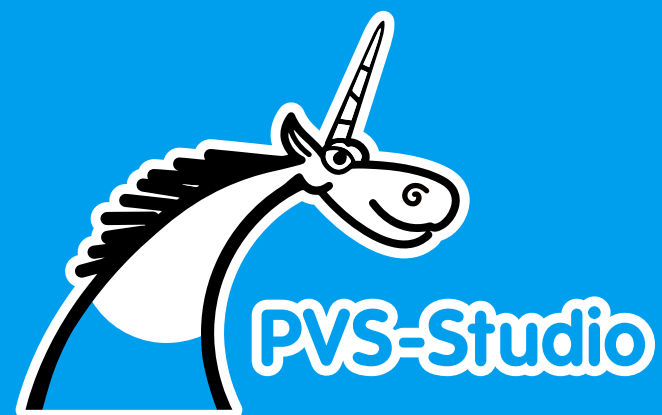
+ Pros

- No need to choose one
- Customize for your needs
- Get a full control
- Set up for projects' specifics

– Cons

- A mini-project
- Needs maintaining and updating
- Have to catch FPs and FNs manually

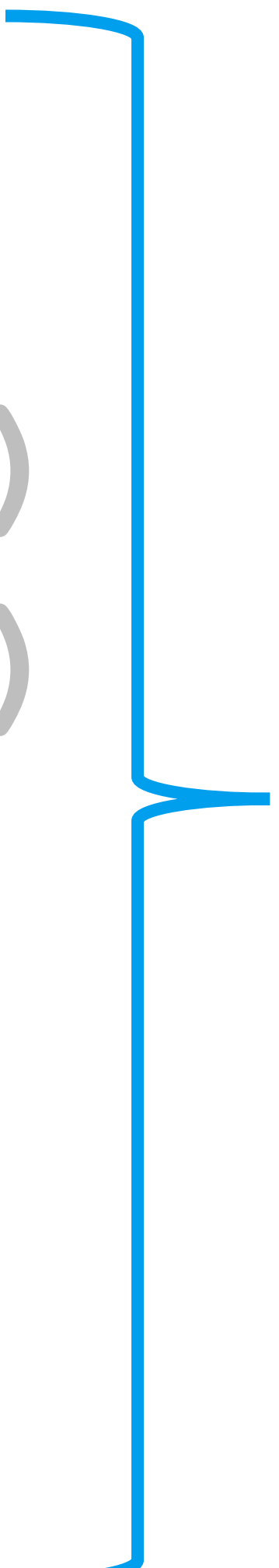
Go technologies and packages for analysis





```
func Foo() {  
    if Condition {  
        CallExpressionX()  
        CallExpressionY()  
    } else {  
        fmt.Println()  
    }  
}
```

```
func Foo() {  
    if Condition {  
        CallExpressionX()  
        CallExpressionY()  
    } else {  
        fmt.Println()  
    }  
}
```



FuncDecl
(function declaration)

```
func Foo() {  
    if Condition {  
        CallExpressionX()  
        CallExpressionY()  
    } else {  
        fmt.Println()  
    }  
}
```



IfStmt
(conditional expression)

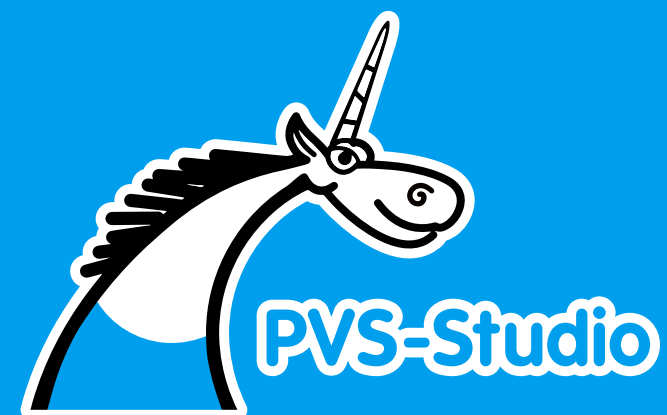
```
func Foo() {  
    if Condition {  
        CallExpressionX()  
        CallExpressionY()  
    } else {  
        fmt.Println()  
    }  
}
```

BlockStmt

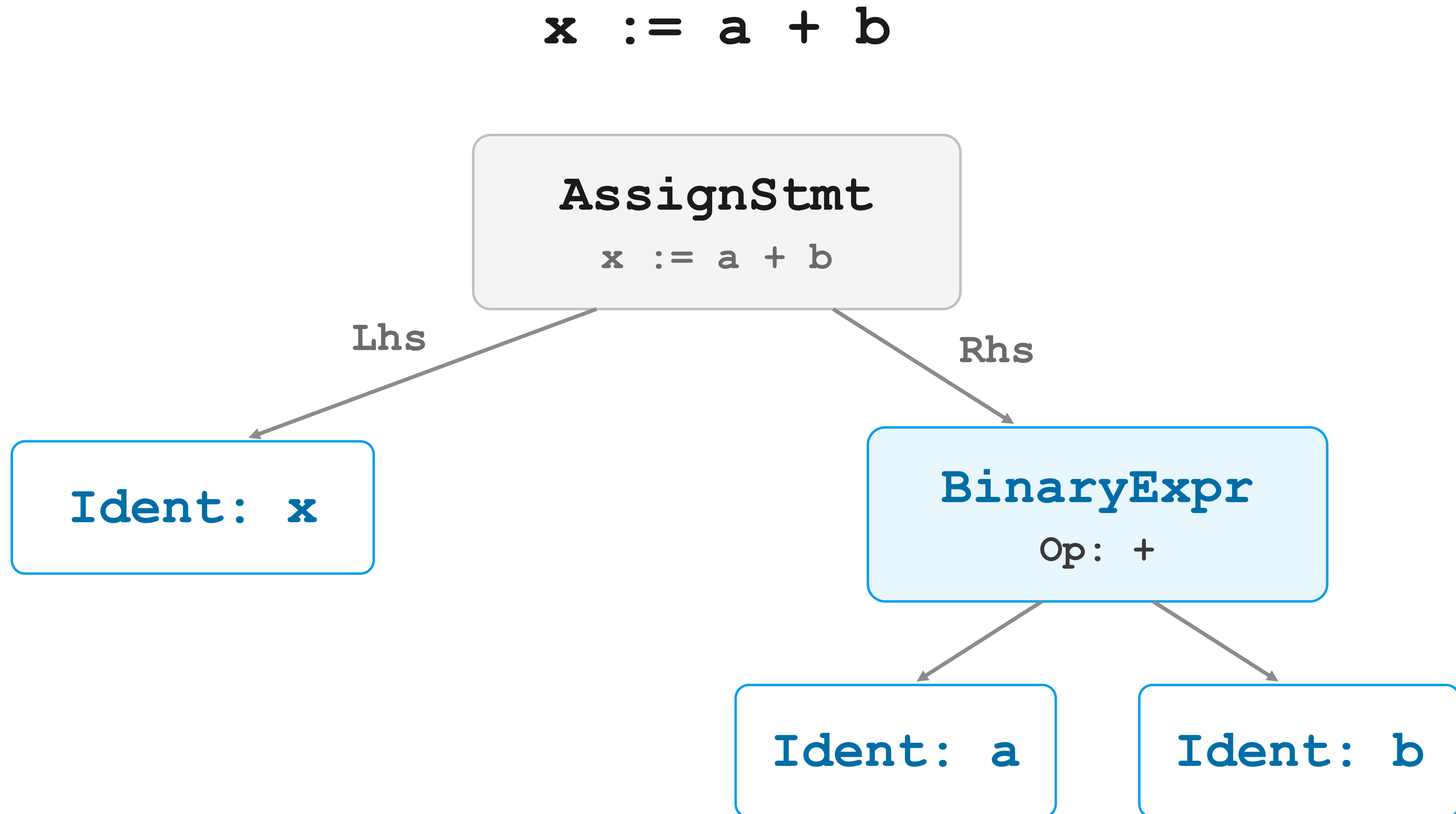
```
func Foo() {  
    if Condition {  
        CallExpressionX()  
        CallExpressionY()  
    } else {  
        fmt.Println()  
    }  
}
```

CallExpr
(function call)

Abstract Syntax Tree (AST)



x := a + b



How syntax finds bugs

47

- WOX

```
func(manifest StorePluginManifest) bool {  
    return manifest.Id == manifest.Id  
}
```

go vet: All good! 

How syntax finds bugs

48

- WOX

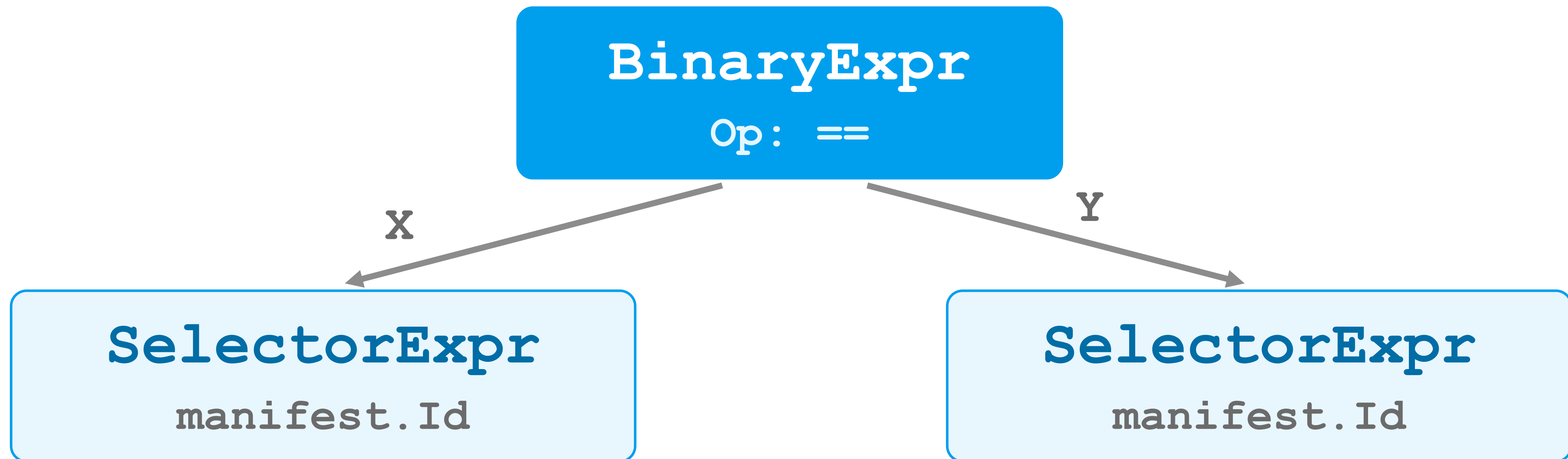
```
for _, manifest := range pluginManifest {  
  
    func(manifest StorePluginManifest) bool {  
        return manifest.Id == manifest.Id  
    }  
}
```

go vet: All good! 

How syntax finds bugs

49

`manifest.Id == manifest.Id`



Left and right are the same subtree.
Types (semantics) aren't needed

Inside the diagnostic rule

50

Source

```
func (b Buffer) Append(value byte) {  
    b.data = append(b.data, value)  
}
```

Inside the diagnostic rule

51

Source

```
func (b Buffer) Append(value byte) {  
    b.data = append(b.data, value)  
}
```

What go/ast sees

FuncDecl

Recv: FieldList

Field

Type: Ident "Buffer"

Name: Ident "Append"

Body: BlockStmt

Inside the diagnostic rule

52

Source

```
func (b Buffer) Append(value byte) {  
    b.data = append(b.data, value)  
}
```

What go/ast sees

FuncDecl

Recv: FieldList

Field

Type: Ident "Buffer"

Name: Ident "Append"

Body: BlockStmt

Look closer:

No asterisk == means **by value**

That's as far as syntax gets us...
It won't help with the rest.

Semantics



```
if m != nil {
    lastChar := line[m[1]-1]
    if lastChar == '.' {
        m[1]--
    } else if lastChar == ')' {
        ....
    } else if lastChar == ';' {
        i := m[1] - 2
        for ; i >= m[0]; i-- {
            if util.IsAlphaNumeric(line[i]) {
                continue
            }
            break
        }
        if i != m[1]-2 {
            if line[i] == '&' {
                m[1] -= m[1] - i
            }
        }
    }
}
```

```
if m != nil {  
    lastChar := line[m[1]-1]  
    if lastChar == '.' {  
        m[1]--  
    } else if lastChar == ')' {  
        ....  
    } else if lastChar == ';' {  
        i := m[1] - 2  
        for ; i >= m[0]; i-- {  
            if util.IsAlphaNumeric(line[i]) {  
                continue  
            }  
            break  
        }  
        if i != m[1]-2 {  
            if line[i] == '&' {  
                m[1] -= m[1] - i  
            }  
        }  
    }  
}
```

$m[1] -= i$



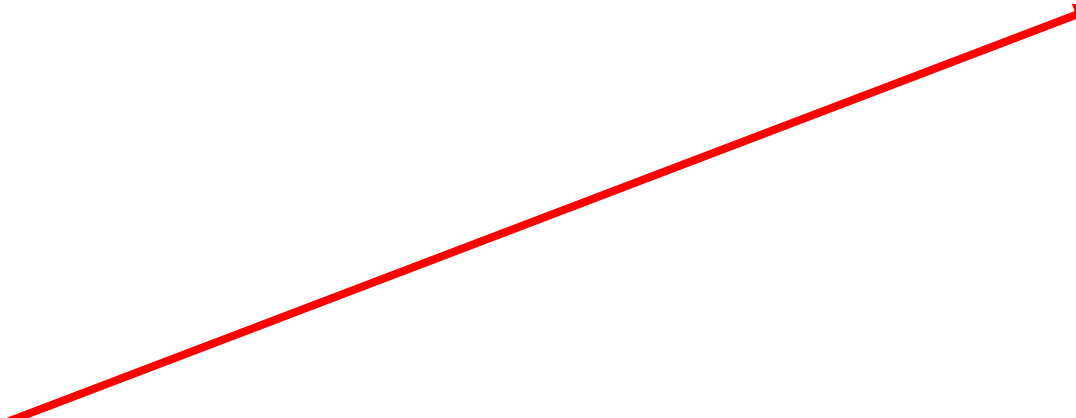
$m[1] = m[1] - i$

How semantics finds bugs

56

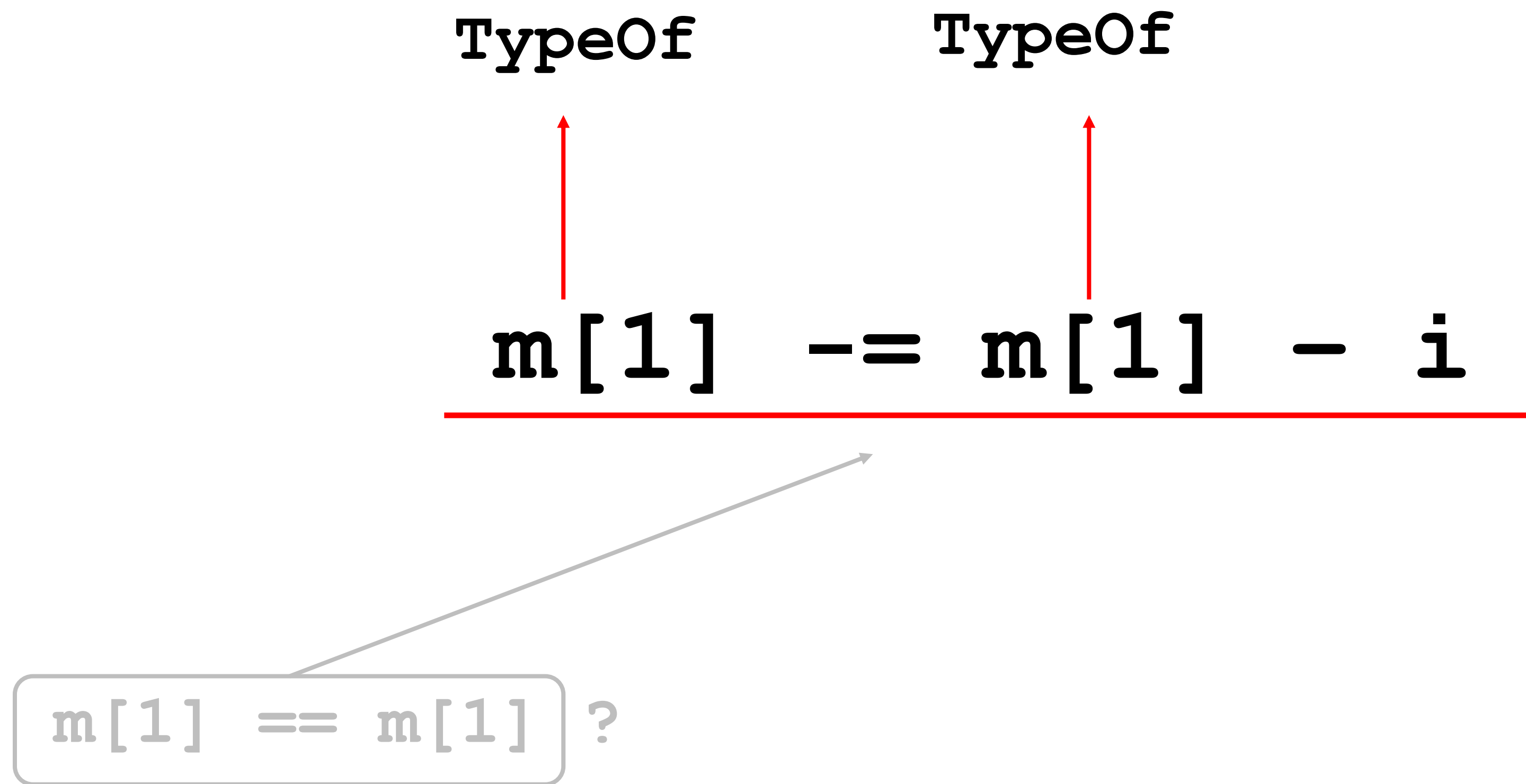
`m[1] -= m[1] - i`

`m[1] == m[1]` ?



How semantics finds bugs

57



Inside the diagnostic rule

58

Inside the analyzer, a field is a number

FieldAddr

X: **t0**

Field: **1**

Inside the diagnostic rule

59

Inside the analyzer, a field is a number

```
FieldAddr
```

```
  X:      t0
```

```
  Field: 1
```

Get the name from the type

```
elem := ptr.Elem().Underlying()
```

```
st    := elem.(*types.Struct)
```

```
st.Field(1).Name()
```

```
// "metricType"
```

Inside the diagnostic rule

60

Inside the analyzer, a field is a number

FieldAddr

X: t0

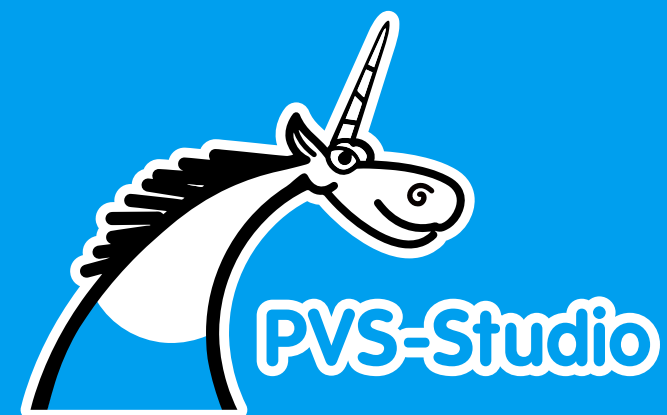
Field: 1

If we have no [go/types](#), the warning would show **field1** instead of a readable path to the field.

Get the name from the

```
elem := ptr.Elem().Underlying()  
st    := elem.(*types.Struct)  
st.Field(1).Name()  
// "metricType"
```

Integration with basic tools



go/ast
syntax tree

go/types
semantics

go/token
position in the file

analysis
golang.org/x/tools/go/
rule framework

go/analysis is a framework for analyzers

- A rule is an `analysis.Analyzer`
- It takes care of parsing and typing
- Singlechecker, multichecker, ready-made passes

go/ast
syntax tree

go/types
semantics

go/token
position in the file

analysis
golang.org/x/tools/go/
rule framework

Optional (when you need more extras)

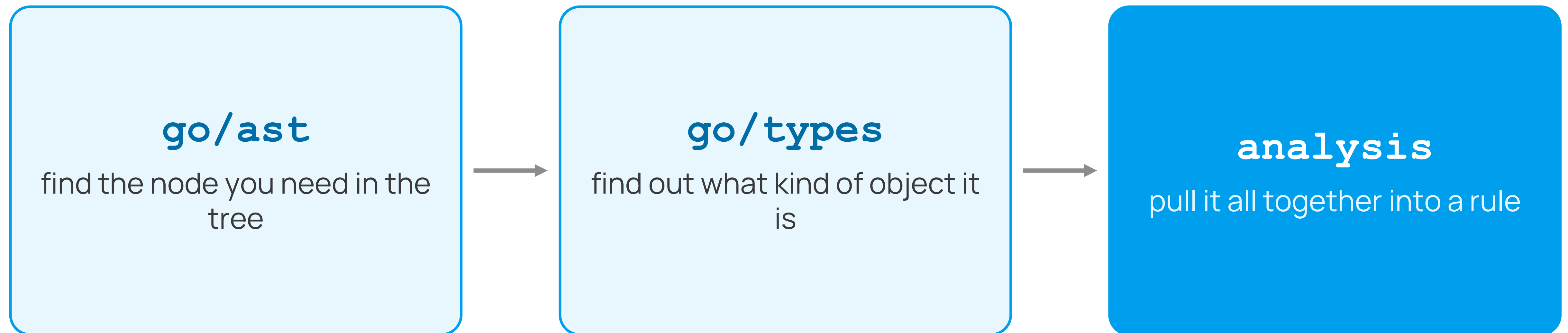
go/parser
manual parsing without the framework

go/packages
loading packages with types

go/ssa
data flow and interprocedural analysis

How it all works together

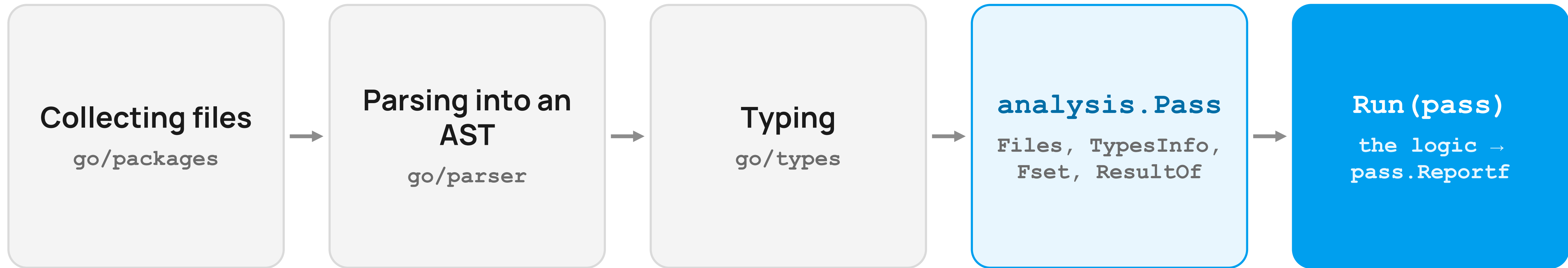
64



The tree tells **you how it's written**. Types **tell you what it means**

How the analysis works

65



The framework does it for you

A place where your rule lives

- Each rule is an object of the ***analysis.Analyzer*** type

- Each rule is an object of the ***analysis.Analyzer*** type
 - The name of the diagnostic rule and description
 - Dependencies on other analyzers
 - The **Run** function
 - Methods of error reporting

- Each rule is an object of the *analysis.Analyzer* type

```
var Analyzer = &analysis.Analyzer{
    Name: "mycheck",
    Doc:  " Description of the rule",
    Run:  run,
}

func run(pass *analysis.Pass) (any, error) {
    // logic check
    return nil, nil
}
```

- Each rule is an object of the *analysis.Analyzer* type

```
var Analyzer = &analysis.Analyzer{
    Name: "mycheck",
    Doc:  " Description of the rule",
    Run:  run,
}

func run(pass *analysis.Pass) (any, error) {
    // logic check
    return nil, nil
}
```

- *analysis.Pass*

- ***analysis.Pass***
 - *Files* is the list of AST files (***ast.File***), ready to be traversed

- ***analysis.Pass***
 - *Files* is the list of AST files (***ast.File***), ready to be traversed
 - ***TypesInfo*** is a type information (***types.Info***) for expressions in the tree

- ***analysis.Pass***

- *Files* is the list of AST files (***ast.File***), ready to be traversed
- ***TypesInfo*** is a type information (***types.Info***) for expressions in the tree
- ***Pkg*** is a current package

- ***analysis.Pass***

- *Files* is the list of AST files (***ast.File***), ready to be traversed
- ***TypesInfo*** is a type information (***types.Info***) for expressions in the tree
- ***Pkg*** is a current package
- ***Reportf*** and ***Report*** are methods for reporting errors

■ *analysis.Pass*

- *Files* is the list of AST files (*ast.File*), ready to be traversed
- *TypeInfo* is a type information (*types.Info*) for expressions in the tree
- *Pkg* is a current package
- *Reportf* and *Report* are methods for reporting errors
- *Fset* is the position table (*token.FileSet*)

■ *analysis.Pass*

- *Files* is the list of AST files (*ast.File*), ready to be traversed
- *TypeInfo* is a type information (*types.Info*) for expressions in the tree
- *Pkg* is a current package
- *Reportf* and *Report* are methods for reporting errors
- *Fset* is the position table (*token.FileSet*)
- *ResultOf* is a data from other analyzers, if the rule has dependencies

- *singlechecker* and *multichecker*

- *singlechecker* and *multichecker*

```
package main
import (
    "golang.org/x/tools/go/analysis/singlechecker")
func main() {
    singlechecker.Main(Analyzer)
}
```

```
go vet -vettool=./mycheck ./...
```

- *singlechecker* and *multichecker*

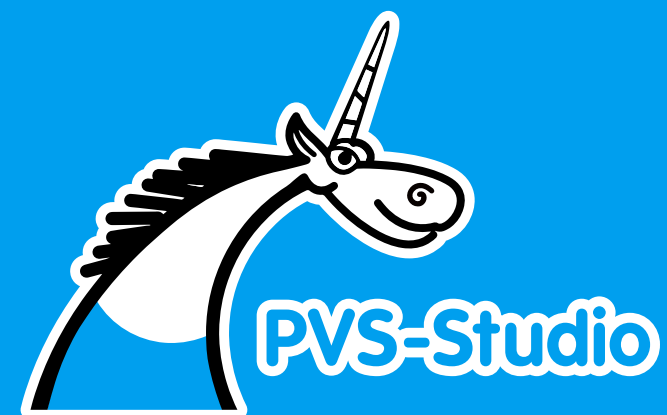
```
multichecker.Main(  
    rule1.Analyzer,  
    rule2.Analyzer,  
    rule3.Analyzer,  
    ...  
)
```

- Running through *go vet*

```
go build -o mycheck ./cmd/mycheck
```

```
go vet -vettool=./mycheck ./...
```

How to write your own analyzer?



Let's write our own diagnostic rule

83

Diagnostic rule should find empty **if** blocks like that:

```
if cond {  
}
```

The rule uses the **go/ast** package to traverse the syntax tree

Let's write our own diagnostic rule

84

The code in `main.go`:

```
package main
import (
    "emptyif"
    "golang.org/x/tools/go/analysis/singlechecker"
)
func main() {
    singlechecker.Main(emptyif.Analyzer)
}
```

Let's write our own diagnostic rule

85

The code of the diagnostic rule in the `emptyif.go` file

```
package emptyif

import (
    "go/ast"
    "golang.org/x/tools/go/analysis"
)

var Analyzer = &analysis.Analyzer{
    Name: "emptyif",
    Doc:  "reports empty if statements",
    Run:  run,
}

func run(pass *analysis.Pass) (any, error) {
    for _, file := range pass.Files {
        ast.Inspect(file, func(n ast.Node) bool {
            if stmt, ok := n.(*ast.IfStmt); ok {
                if stmt.Body != nil && len(stmt.Body.List) == 0 {
                    pass.Reportf(stmt.Pos(), "empty if block")
                }
            }
            return true
        })
    }
    return nil, nil
}
```

What this rule consists of

86

- **AST traversal**
- **Check for IfStmt**
- **Analysis the block body**
- **Report generation**

What this rule consists of

87

- **AST traversal**

Uses `ast.Inspect`, which recursively iterates through the tree's nodes

```
for _, file := range pass.Files {  
    ast.Inspect(file, func(n ast.Node) bool {
```

- Check for `IfStmt`
- Analysis of the block body
- Report generation

What this rule consists of

88

- AST traversal
- **Check for IfStmt**

The node is matched via a type assertion: `n.(*ast.IfStmt)`

```
if stmt, ok := n.(*ast.IfStmt); ok {
```

- Analysis of the block body
- Report generation

What this rule consists of

89

- AST traversal
- Check for `IfStmt`
- **Analysis of the block body**

`stmt.Body.List` contains a list of statements. If it is empty, we issue a warning.

```
if stmt.Body != nil && len(stmt.Body.List) == 0 {
```

- Report generation

What this rule consists of

90

- AST traversal
- Check for IfStmt
- Analysis of the block body
- **Report generation**

pass.Reportf outputs a message tied to a location in the source.

```
pass.Reportf(stmt.Pos(), "empty if block")
```

So what do we get?

91

Everything right? Good job! Build and run your analyzer!

```
go build -o emptyif.exe ./cmd/emptyif
go vet -vettool=D:\emptyif\emptyif.exe ./...
```

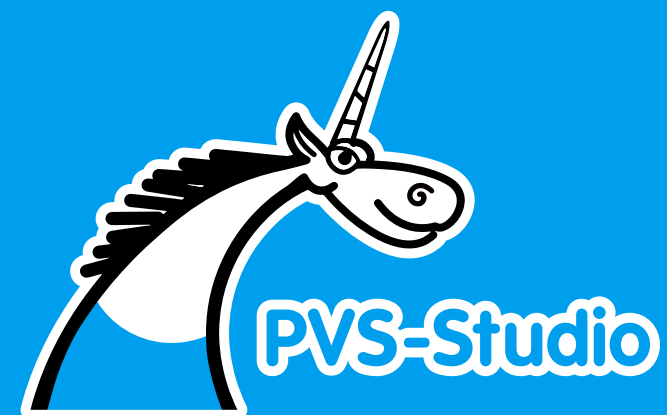
So what do we get?

92

And...the output will look roughly like this:

```
tests\emptyif_test\file1.go:12:2: empty if block
```

Back to our goal



Logic of diagnostic rule

94

1. A method whose receiver is passed by value
2. An entry is inside a field of that receiver
3. Nothing later in the code reads that field
4. So the write is lost: issue the warning

One tree alone isn't enough. Let's look at execution order and SSA

The same thing, but in code

95

```
recv := fn.Receiver()
// 1. only value receivers are needed
if recv == nil || isPointer(recv) {
    return
}
// 2. all entries to fields of this copy
for _, w := range writesToFields(recv) {
    // 3. is this field read anywhere after the entry?
    if !hasReadAfter(w) {
        // 4. non-read, so the write is lost
        report(w, "copy's field, entry is lost")
    }
}
```

What can we find: example

96

- Milvus

```
func (gi GenericIndex) WithMetricType(metricType MetricType) {  
    gi.baseIndex.metricType = metricType  
}
```

- go vet and staticcheck stay silent

What can we find: example

97

- Milvus

```
func (gi GenericIndex) WithMetricType(metricType MetricType) {  
    gi.baseIndex.metricType = metricType  
}
```

- go vet and staticcheck stay silent

What the diagnostic rule outputs:

Modifying the 'gi.baseIndex.metricType' field of the 'gi' value receiver has no effect.
Consider using a pointer receiver.

The copy's field changes — the original object stays unchanged

PVS-Studio 8.0

98

- Support for Go, JS, and TS
(in addition to C/C++/C#/Java analyzers)
- ~40 new rules per language
(on top of 1,000+ already available)
- New and updated IDE plugins
(major VS Code extension overhaul)



Q/A

