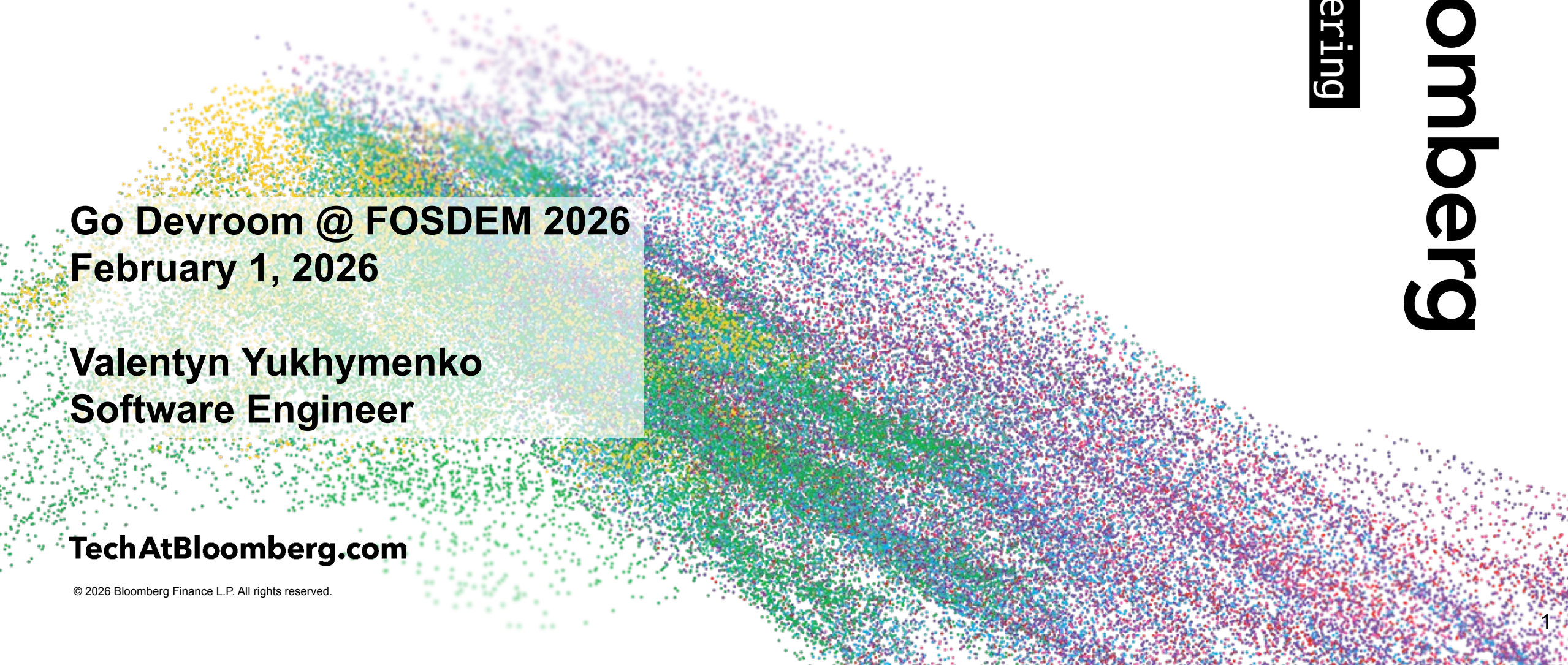
Inside Reflection

Bloomberg

Engineering



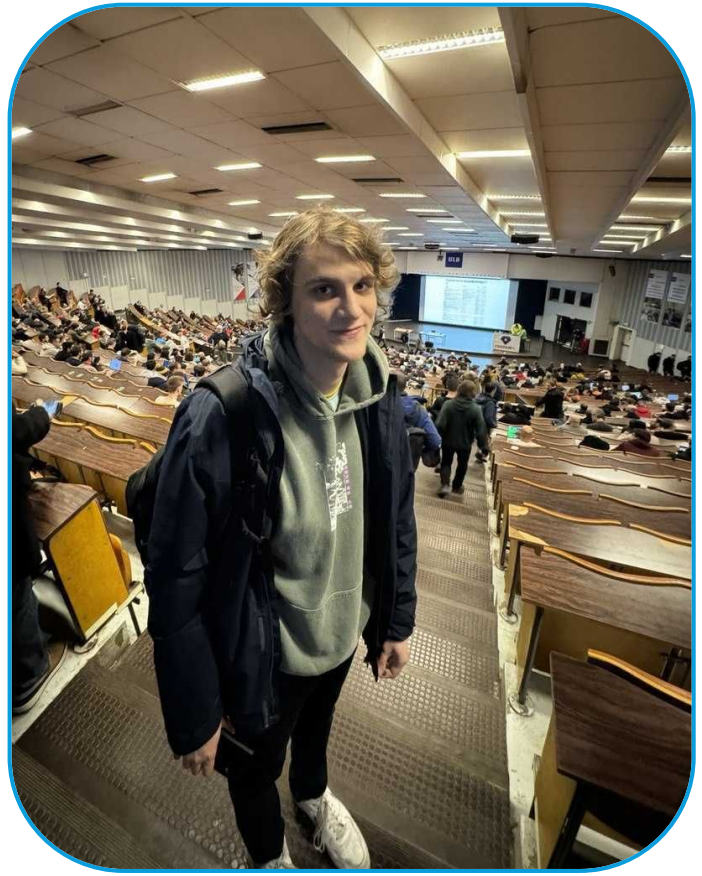
Go Devroom @ FOSDEM 2026
February 1, 2026

Valentyn Yukhymenko
Software Engineer

TechAtBloomberg.com

About

- Helps improve deployment process at Bloomberg by day
- Tries to understand compilers by night (literally 🥲)
- Has small contribution experience with C++ Reflection implementations
- Uses Go regularly, so decided to see how Reflection is implemented there!



My first FOSDEM 2 years ago

TechAtBloomberg.com

Bloomberg

Engineering

Agenda

1. What is Reflection?
2. What's inside the Go interface?
3. What's inside the reflect package?

TechAtBloomberg.com

Bloomberg

Engineering

What is Reflection?

Ability* of a program to:

- 🔍 **Introspect** existing types and behaviour
- 🔨 **Modify** existing types and behaviour
- 🔮 **Create** new types or instances from scratch

** Usually in runtime but some languages support compile time reflection*

TechAtBloomberg.com

Source of AI generated image on this slide: ChatGPT

Bloomberg

Engineering

State of Reflection

Python	Java	C++	Go
📜 Bytecode + Interpreted (VM)	📜 Bytecode + JIT (JVM)	📦 AOT compiled (native)	📦 AOT compiled (native)
🧩Dynamic types	🔒 Static types	🔒 Static types	🔒 Static types
🥇 Runtime Reflection	🥈 Runtime Reflection	🏗️ Compile-time Reflection (C++ 26)	🥉 Runtime Reflection

TechAtBloomberg.com

Bloomberg

Engineering

So, how is runtime reflection achieved in Go?

“Interfaces are, for me, the most exciting part of Go from a language design point of view. If I could export one feature of Go into other languages, it would be interfaces.”

– Russ Cox

“Go Data Structures: Interfaces” blog post (2009)

TechAtBloomberg.com

Bloomberg

Engineering

Inspired by go.dev/blog/laws-of-reflection

Interfaces 101

```
file, _ := os.OpenFile("/dev/null", os.O_RDWR, 0)
fmt.Printf("Type of `file`: %T; Value: %v\n", file, file)
// Type of `file`: *os.File; Value: &{0xc0000941e0}
```

TechAtBloomberg.com

Bloomberg

Engineering

Inspired by go.dev/blog/laws-of-reflection

Interfaces 101

```
file, _ := os.OpenFile("/dev/null", os.O_RDWR, 0)
fmt.Printf("Type of `file`: %T; Value: %v\n", file, file)
// Type of `file`: *os.File; Value: &{0xc0000941e0}

var reader io.Reader = file
fmt.Printf("Type of `reader`: %T; Value: %v\n", reader, reader)
// Type of `reader`: *os.File; Value: &{0xc0000941e0}
```


Bloomberg

Engineering

Inspired by go.dev/blog/laws-of-reflection

Interfaces 101

```
file, _ := os.OpenFile("/dev/null", os.O_RDWR, 0)
fmt.Printf("Type of `file`: %T; Value: %v\n", file, file)
// Type of `file`: *os.File; Value: &{0xc0000941e0}

var reader io.Reader = file
fmt.Printf("Type of `reader`: %T; Value: %v\n", reader, reader)
// Type of `reader`: *os.File; Value: &{0xc0000941e0}

var writer io.Writer = reader.(io.Writer)
fmt.Printf("Type of `writer`: %T; Value: %v\n", writer, writer)
// Type of `writer`: *os.File; Value: &{0xc0000941e0}
```


Inspired by go.dev/blog/laws-of-reflection

Interfaces 101

```
var empty any = file // `any` is the same as `interface{}`
fmt.Printf("Type of `empty`: %T; Value: %v\n", empty, empty)
// Type of `empty`: *os.File; Value: &{0xc0000941e0}
```

TechAtBloomberg.com

Bloomberg

Engineering

Inspired by go.dev/blog/laws-of-reflection

Interfaces 101

```
var empty any = file // `any` is the same as `interface{}`
fmt.Printf("Type of `empty`: %T; Value: %v\n", empty, empty)
// Type of `empty`: *os.File; Value: &{0xc0000941e0}

empty = 42
fmt.Printf("Type of `empty`: %T; Value: %v\n", empty, empty)
// Type of `empty`: int; Value: 42
```


Bloomberg

Engineering

Internals of Interface: Fat Pointer approach

```
// Type of `reader`: *os.File; Value: &{0xc0000941e0}
```

```
type iface struct {
    tab  *abi.ITab
    data unsafe.Pointer
}
```

Source: go/src/runtime/runtime2.go

**ABI stands for Application Binary Interface*

TechAtBloomberg.com

Bloomberg

Engineering

Internals of Interface: Fat Pointer approach

// Type of `reader`: ***os.File**; Value: **&{0xc0000941e0}**

```
type iface struct {
    tab  *abi.ITab
    data unsafe.Pointer
}
```

Pointer to **interface table**

Raw pointer to underlying **value**

Source: [go/src/runtime/runtime2.go]

**ABI stands for Application Binary Interface*

TechAtBloomberg.com

Bloomberg

Engineering

Internals of Interface: Interface Table

```
// Type of `reader`: *os.File; Value: &{0xc0000941e0}
```

```
type ITab struct {
    Inter *InterfaceType
    Type  *Type
    Hash  uint32
    Fun   [1]uintptr
}
```

Source: go/src/internal/abi/iface.go

**ITab stands for Interface Table*

TechAtBloomberg.com

Bloomberg

Engineering

Internals of Interface: Interface Table

```
// Type of `reader`: *os.File; Value: &{0xc0000941e0}

type ITab struct {
    Inter *InterfaceType
    Type  *Type
    Hash  uint32
    Fun   [1]uintptr
}
```

Definition of interface → **type** + **methods**

Underlying type which **implements** interface
e.g. **io.Reader** = ***os.File**

Source: [go/src/internal/abi/iface.go]

**ITab stands for Interface Table*

TechAtBloomberg.com

Bloomberg

Engineering

Internals of Interface: Interface Table

```
type ITab struct {
    Inter *InterfaceType
    Type  *Type
    Hash  uint32
    Fun   [1]uintptr
}
```

```
type InterfaceType struct {
    Type
    PkgPath Name
    Methods []Imethod
}
```

Source: go/src/internal/abi/iface.go

**ITab stands for Interface Table*

TechAtBloomberg.com

Bloomberg

Engineering

Internals of Interface: Interface Table

```
type ITab struct {
    Inter *InterfaceType
    Type  *Type
    Hash  uint32
    Fun   [1]uintptr
}
```

Copy of **Type.Hash.** Used for **type switches**

```
func do(i any) {
    switch v := i.(type) {
    case int: // case 12345
        // TODO:
    case string: // case 54321
        // TODO:
    default:
        panic("unexpected")
    }
}
```

Source: [go/src/internal/abi/iface.go]

TechAtBloomberg.com

Bloomberg

Engineering

Internals of Interface: Interface Table

```
type ITab struct {
    Inter *InterfaceType
    Type  *Type
    Hash  uint32
    Fun   [1]uintptr
}
```

Function pointer table.

Variable sized array. **fun[0] == 0** means **Type does not** implement **InterfaceType**.

Source: [go/src/internal/abi/iface.go]

Internals of Type

Common information about all **types**

```
type Type struct {
    Str         NameOff
    Size_       uintptr
    Hash        uint32
    ...
    Kind_       Kind
    ...
}
```

Source: [go/src/internal/abi/type.go]

Internals of Type

Common information about all **types**

```
type Type struct {
    Str       NameOff
    Size_     uintptr
    Hash      uint32
    ...
    Kind_     Kind
    ...
}
```

Source: [go/src/internal/abi/type.go]

TechAtBloomberg.com

Bloomberg

Engineering

Internals of Type

Common information about all **types**

```
type Type struct {
    Str       NameOff
    Size_     uintptr
    Hash      uint32
    ...
    Kind_     Kind
    ...
}
```

Bool
Int
...
Array
Chan
Func
Interface
Map
Pointer
Slice
String
Struct
UnsafePointer

Enables access to **specific** **type** information

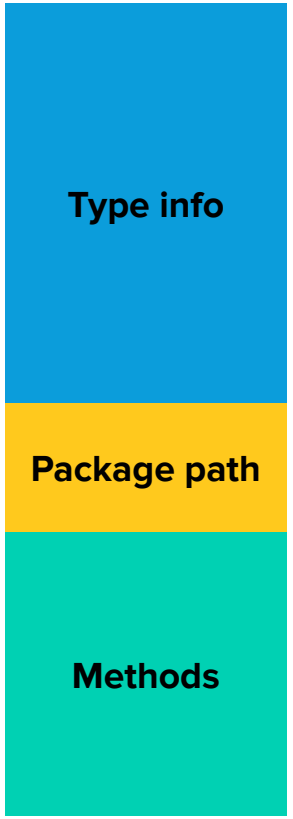
Source: [go/src/internal/abi/type.go]

Internals of Type: How to get Interface methods?

```
func (t *Type) InterfaceType() *InterfaceType
{
    if t.Kind() != Interface {
        return nil
    }
    return (*InterfaceType)(unsafe.Pointer(t))
}
```

```
type InterfaceType struct {
    Type
    PkgPath Name
    Methods []Imethod
}
```

Memory layout



Source: go/src/internal/abi/type.go

Internals of Type: How to get Struct fields?

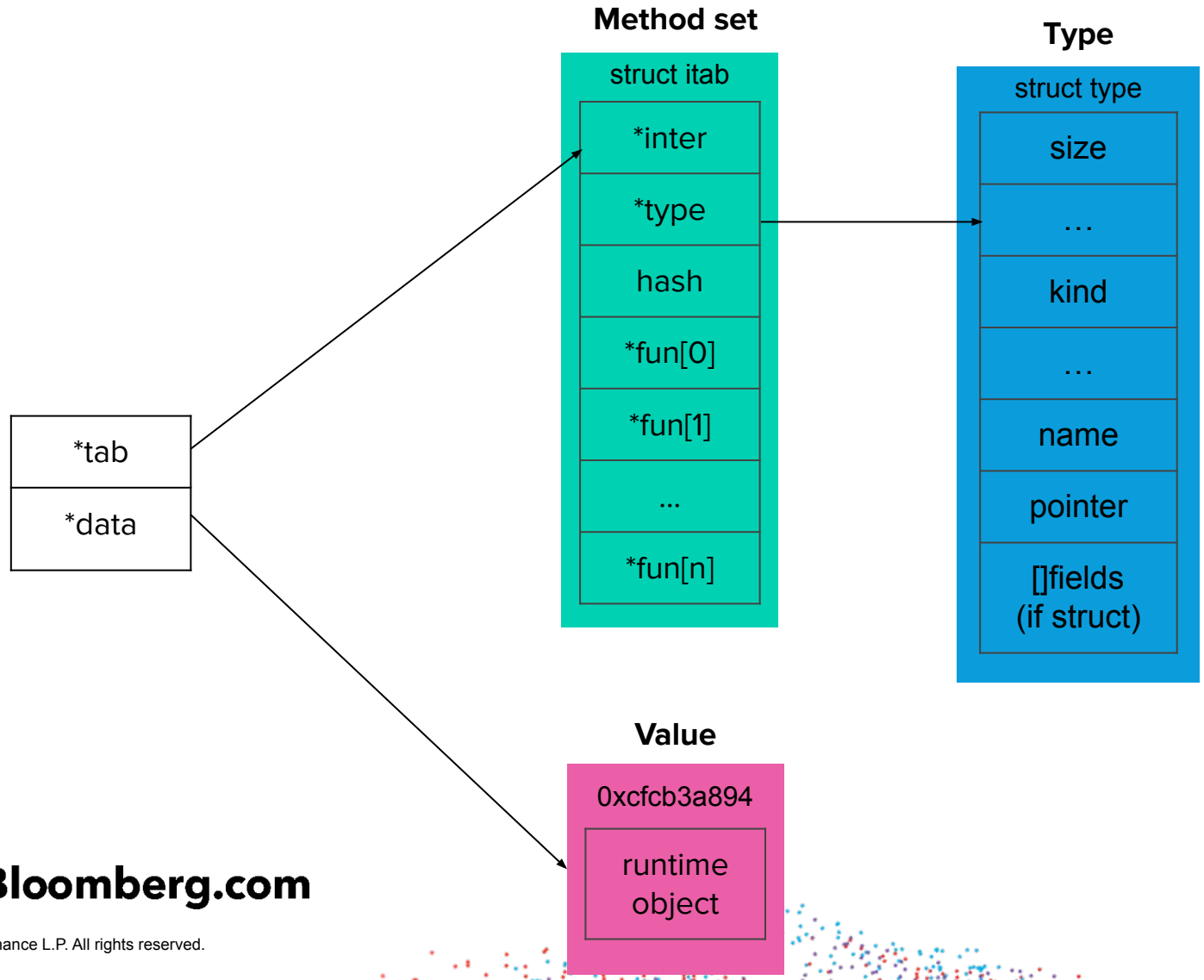
```
func (t *Type) StructType() *StructType
{
    if t.Kind() != Struct {
        return nil
    }
    return (*StructType)(unsafe.Pointer(t))
}
```

```
type StructType struct {
    Type
    PkgPath Name
    Fields  []StructField
}
```

Source: [go/src/internal/abi/type.go]

Inspired by: blog.gopheracademy.com/advent-2018/interfaces-and-reflect

Connection between Type and Interface



TechAtBloomberg.com

Bloomberg

Engineering

Edge case: Empty Interface

```
// Interface any


type eface struct {
    type  *abi.Type
    data unsafe.Pointer
}
```

```
// Interface io.Reader


type iface struct {
    tab  *abi.ITab
    data unsafe.Pointer
}
```

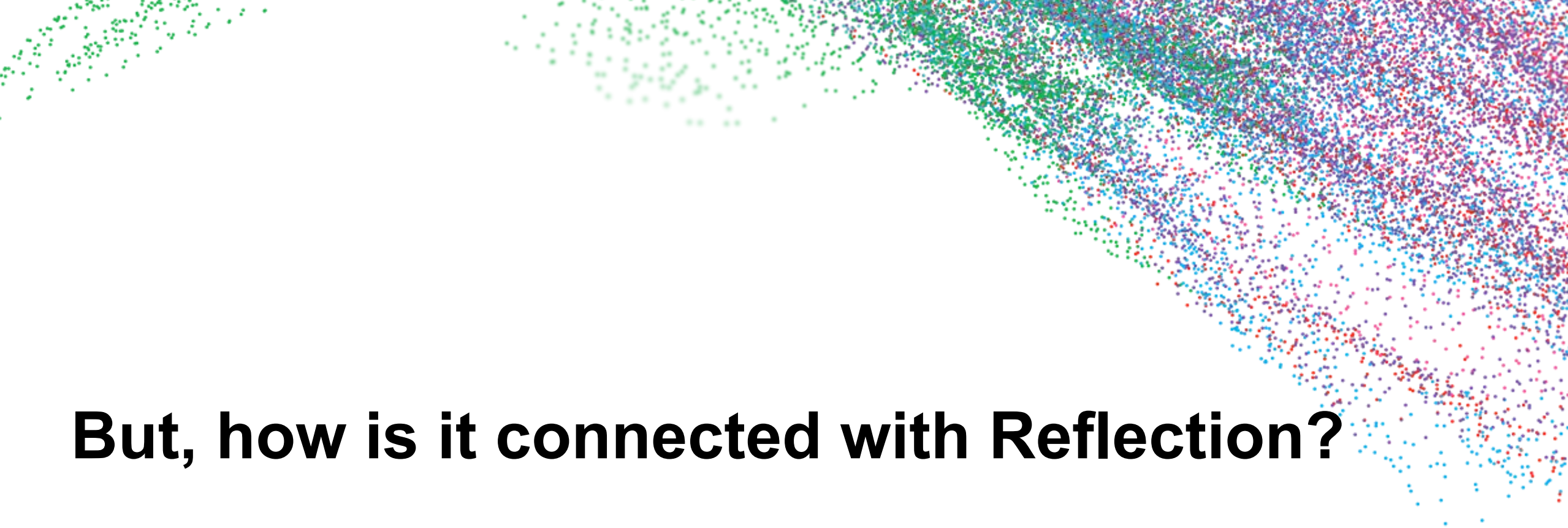
Source: go/src/runtime/runtime2.go

**ABI stands for Application Binary Interface*

TechAtBloomberg.com

Bloomberg

Engineering



But, how is it connected with Reflection?

Reflection API in Go

reflect.TypeOf(i any)

reflect.ValueOf(i any)

pkg.go.dev/reflect

TechAtBloomberg.com

Bloomberg

Engineering

Reflection API in Go

reflect.TypeOf(i any)

reflect.ValueOf(i any)

Looks familiar?

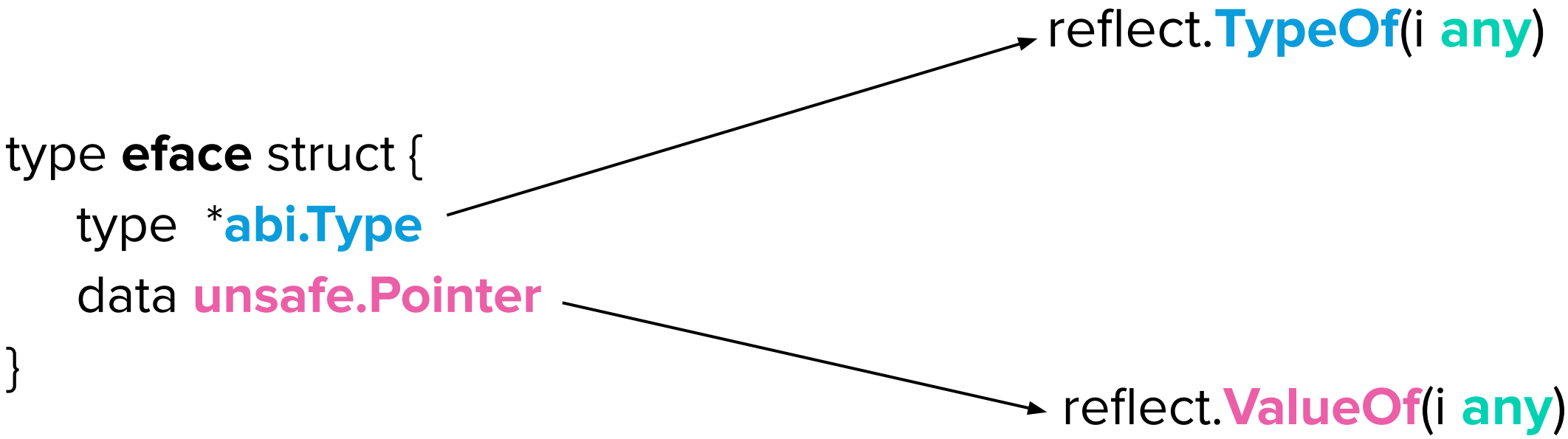
pkg.go.dev/reflect

TechAtBloomberg.com

Bloomberg

Engineering

Reflection API in Go



pkg.go.dev/reflect

TechAtBloomberg.com

Bloomberg

Engineering

Inside reflect.TypeOf

```
func TypeOf(i any) reflect.Type {
    return toType(abi.TypeOf(i))
}

func toType(t *abi.Type) reflect.Type {
    if t == nil {
        return nil
    }
    return (*rtype)(unsafe.Pointer(t))
}
```

```
type reflect.Type interface {
    Align() int
    Method(int) Method
    Kind() Kind
    Field(i int) StructField
    ...
}
```

```
type rtype struct {
  t  abi.Type
}
```

Source: go/src/reflect/type.go

Inside reflect.ValueOf

```
func ValueOf(i any) reflect.Value {
    if i == nil {
        return Value{}
    }
    return unpackEface(i)
}
```

```
type reflect.Value struct {
    typ_    *abi.Type
    ptr     unsafe.Pointer
    flag    uintptr
}
```

Source: go/src/reflect/value.go

reflect package: Some examples

- reflect.Type.Kind();

  reflect.Type.Size();

```
func (t *rtype) Kind() Kind {
    return Kind(t.t.Kind())
}
```

```
func (t *rtype) Size() uintptr{
    return t.t.Size()
}
```

Source: [go/src/reflect/type.go]

reflect package: Some examples

- reflect.**Type**.Kind();

  reflect.**Type**.Size();

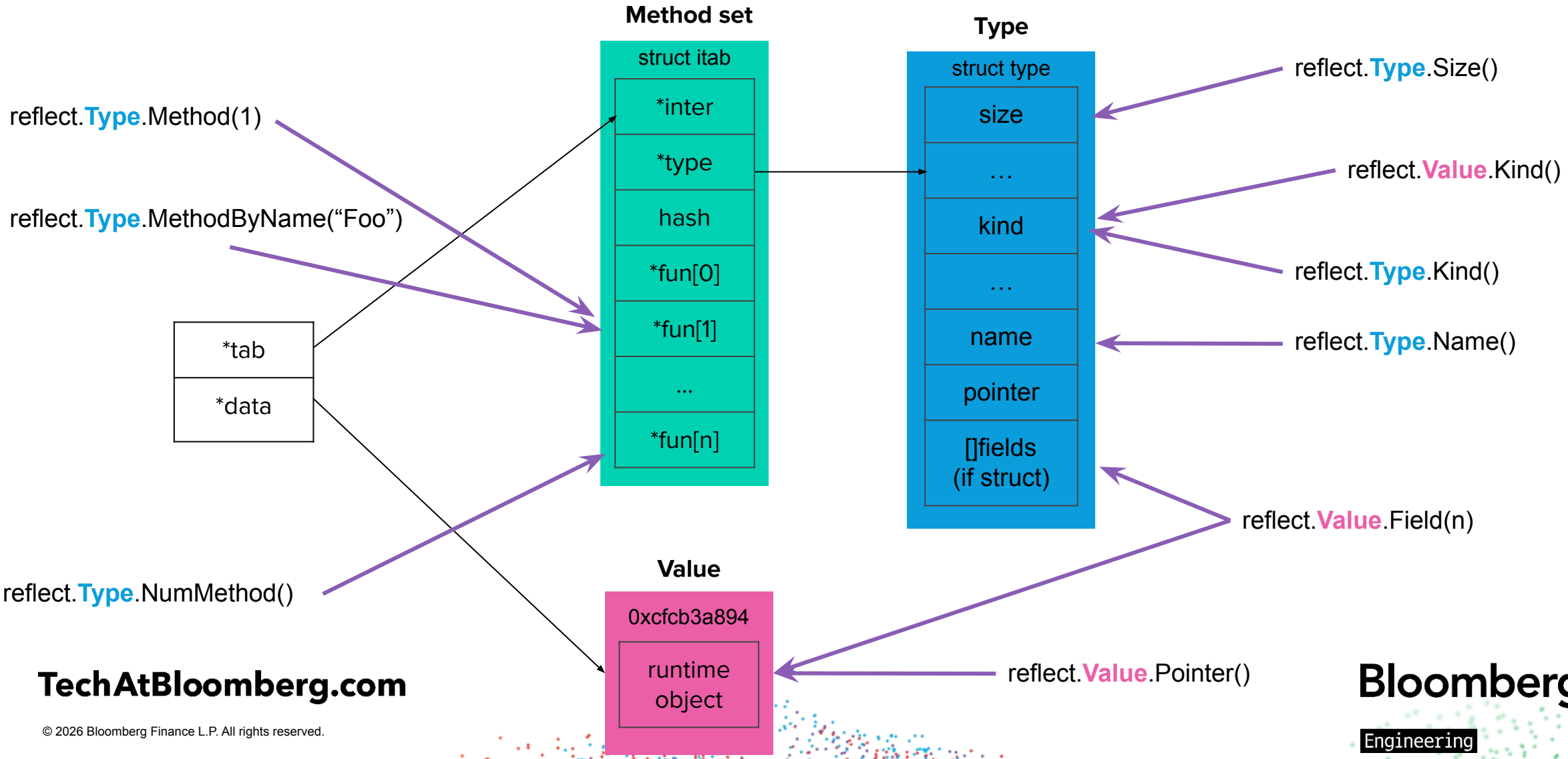
- reflect.**Value**.SetBool(bool);

```
func (v Value) SetBool(x bool) {
    v.mustBeAssignable()
    v.mustBe(Bool)
    *(*bool)(v ptr) = x
}
```

Source: [go/src/reflect/type.go]

Inspired by: blog.gopheracademy.com/advent-2018/interfaces-and-reflect

Connection between Type and Interface



TechAtBloomberg.com

Bloomberg

Engineering

Conclusions

- Interface **any** enables runtime reflection capabilities in Go.
- **reflect** package provides a **safe thin API** over underlying information of **interface**.
- It’s not magic! Just good simple design.

TechAtBloomberg.com

Bloomberg

Engineering

What haven’t we covered?

We have looked only at how runtime information is **accessed**.

But how is this information **generated and propagated** into interfaces during compilation?

TechAtBloomberg.com

Bloomberg

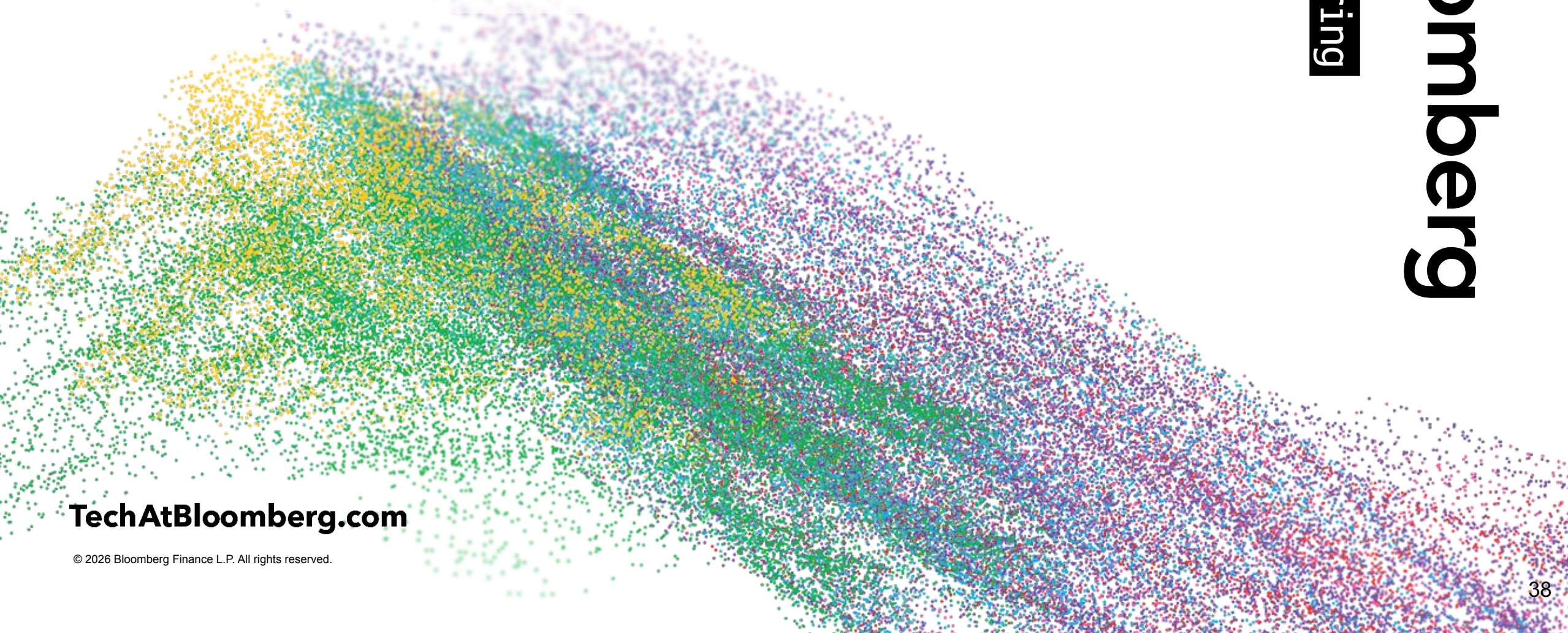
Engineering

Thank you!

Speak to me in between other talks if you have any feedback!

Bloomberg

Engineering



TechAtBloomberg.com