Interface Internals

By Evan Shaw

Normal Method Calls

```
func ReadFull(r *os.File, buf []byte) (n int, err error) {
      for n < len(buf) && err == nil {
            var nn int
            nn, err = r.Read(buf[n:])
            n += nn
      }
      return
}
```

Static dispatch

We know exactly which method body we’re jumping to

Interface Method Calls

```
func ReadFull(r io.Reader, buf []byte) (n int, err error) {
      for n < len(buf) && err == nil {
            var nn int
            nn, err = r.Read(buf[n:])
            n += nn
      }
      return
}
```

Dynamic dispatch

No way of knowing which Read method we’re calling

What an Interface Looks Like on the Inside

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

type itab struct {
      inter  *interfacetype
      _type  *_type
      link   *itab
      hash   uint32 // copy of _type.hash. Used for type switches.
      bad    bool   // type does not implement interface
      inhash bool   // has this itab been added to hash?
      unused [2]byte
      fun    [1]uintptr // variable sized
}
```

Where Do `itabs` Come From?

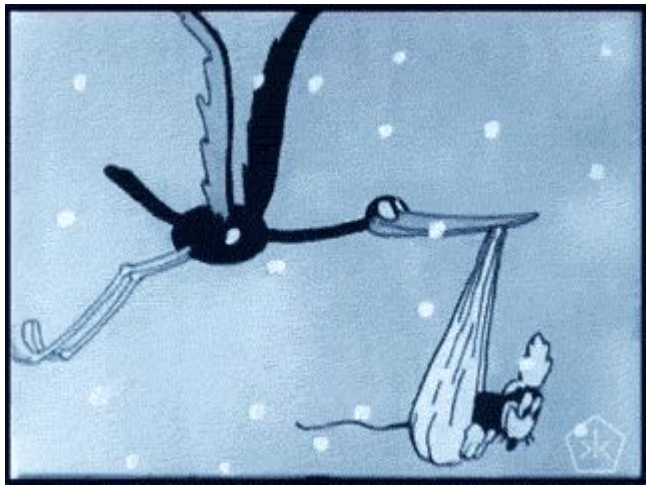
Compiler makes them:

```
var r io.Reader
r, _ = os.Open(“file.txt”)
```

Runtime makes them, dynamically:

```
r.(io.ReadCloser)
```

(Uses reflection data, caches result)



Hidden Costs, Part 1

Compiler can’t inline interface method calls because it doesn’t know what to inline

Hidden Costs, Part 2

```
var global []byte

type evilReader int

func (r *evilReader) Read(b []byte) error {
    global = b
}

func readSomething(r io.Reader) {
    var buf [8]byte // Want to allocate on the stack
    r.Read(buf[:])  // But this might be an evilReader!
}
```

Compiler can’t run escape analysis through an interface call, because it doesn’t know what to analyze and has to assume the worst.

Hidden Costs, Part 3

Putting a non-pointer inside an interface requires a heap allocation, which must later be garbage-collected.

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

This can only be a pointer, for garbage collection reasons

(Normal escape analysis applies for pointer values.)

Conclusion

Interfaces are a great abstraction!

Costs are still cheap compared to method calls in many dynamic languages.

It’s still good to know the costs and recognize when they aren’t worth it.