GOLANG UPDATE

ERIK LUPANDER

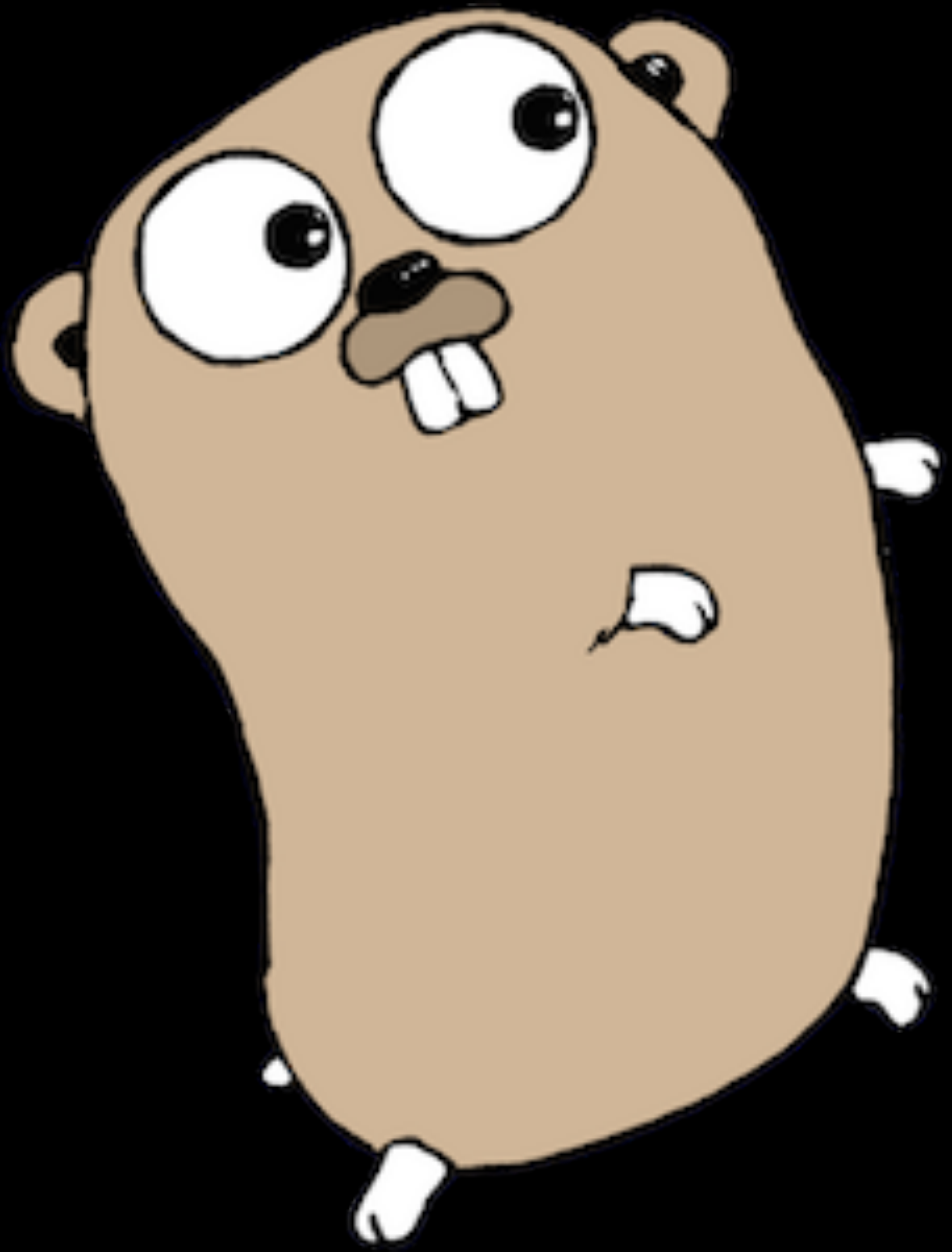
CADEC 2019.01.24 & 2019.01.30 | CALLISTAENTERPRISE.SE

CALLISTA

— ENTERPRISE —

AGENDA

- Where is Go in 2019?
- Points of criticism
- Go modules
- Go 2.0 drafts
- Summary



GOLANG IN 2019?

BACK TO 2017…

"Go is an attempt to combine the ease of programmin
interpreted, dynamically typed language, with the eff
and safety of a statically typed, compiled languag
Go offic
CALL

CADEC 2017 RECAP - WHY GO FOR MICROSERVICES?

- Runtime efficiency
- Statically linked binaries
- Lightning fast compilation
- Cross platform
- Pragmatic language favoring simplicity and productivity
- Great standard lib and community

BACK TO 2017…

- Go had seen a huge increase in popularity
- Commonly used in cloud infrastructure software
- The advent of microservices seemed a natural fit

WHERE IS GO IN 2019?

- Ubiquitous in cloud and systems programming
- Tools, network apps
- Microservices, APIs
  - 1000+ companies listed on the Go wiki using Go
- Much better IDEs
- Popularity in rankings stabilized ~#15

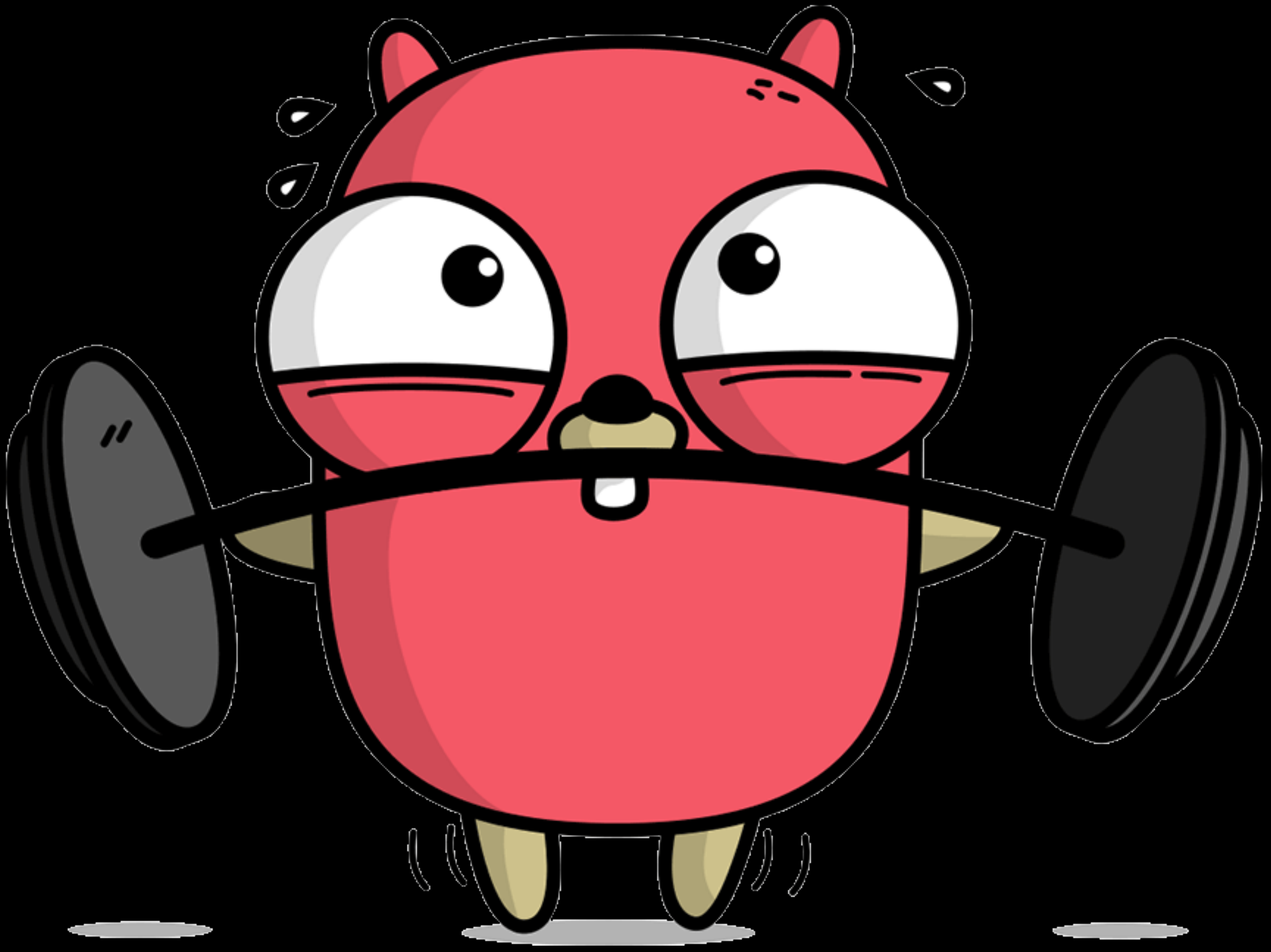
GO IN THE YEARS AHEAD

- Will Go stay relevant?
  - Definitely in the cloud infrastructure space
  - Fierce competition in the microservice / APIs space
    - » JVM
      - Micronaut, GraalVM
    - » Rust, NodeJS, PHP 7, .NET Core, …
  - Go 2.0 on the horizon…

POINTS OF CRITICISM

No generics? No exceptions? Wut!?

HTTPS://WWW.SLIDESHARE.NET/ARHAN/SOMETHING-ABOUT-GOLANG-53392734



HTTPS://GOPHERCISES.COM/

1
DEPENDENCY MANAGEMENT

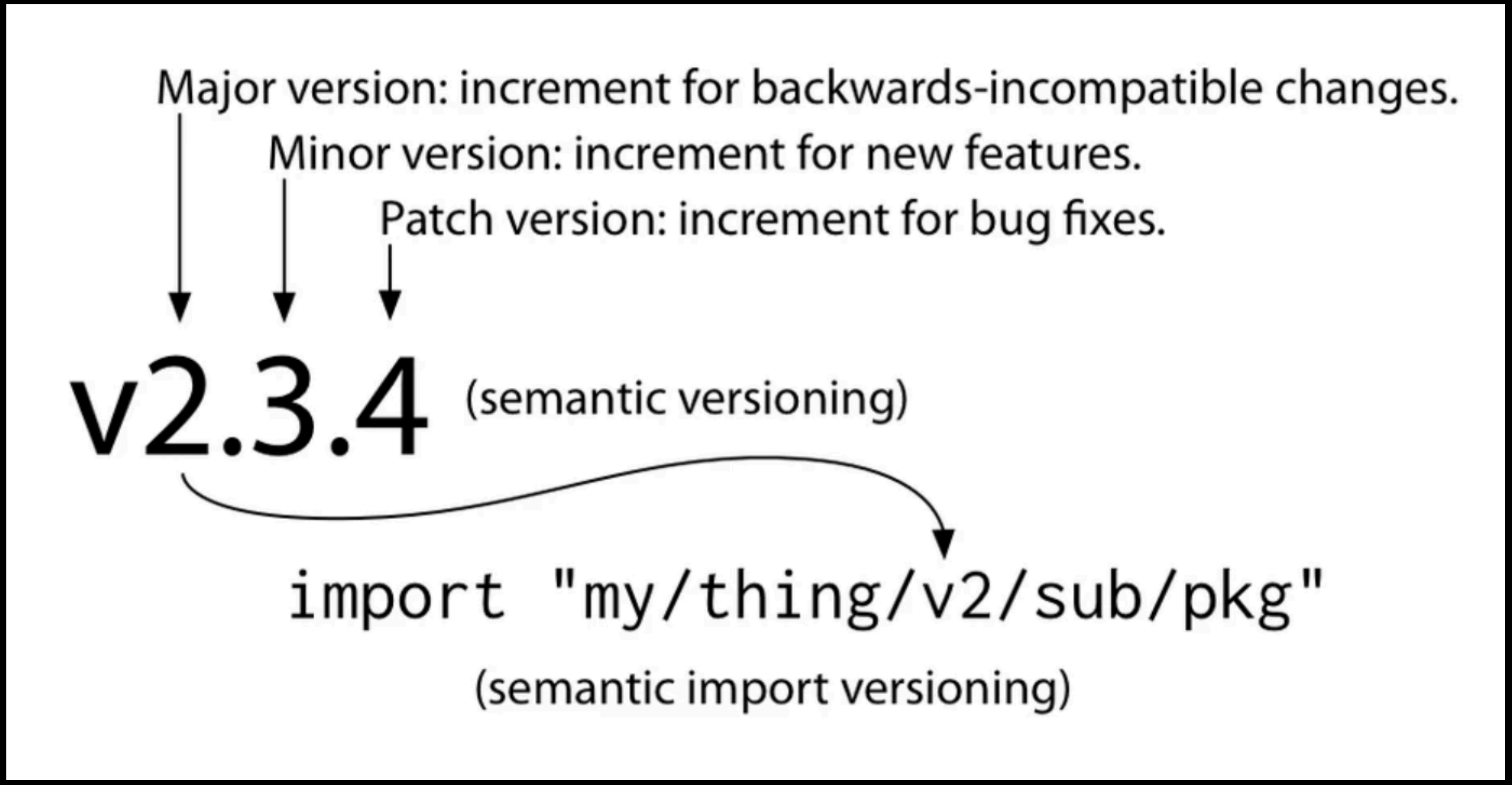
DEPENDENCY MANAGEMENT

- Go 1.0 shipped with rudimentary dependency handling
  - Downloads source from source repositories
    - » Which version?
    - » Transitive dependencies?
- 10+ 3rd-party dependency managers

GO MODULES

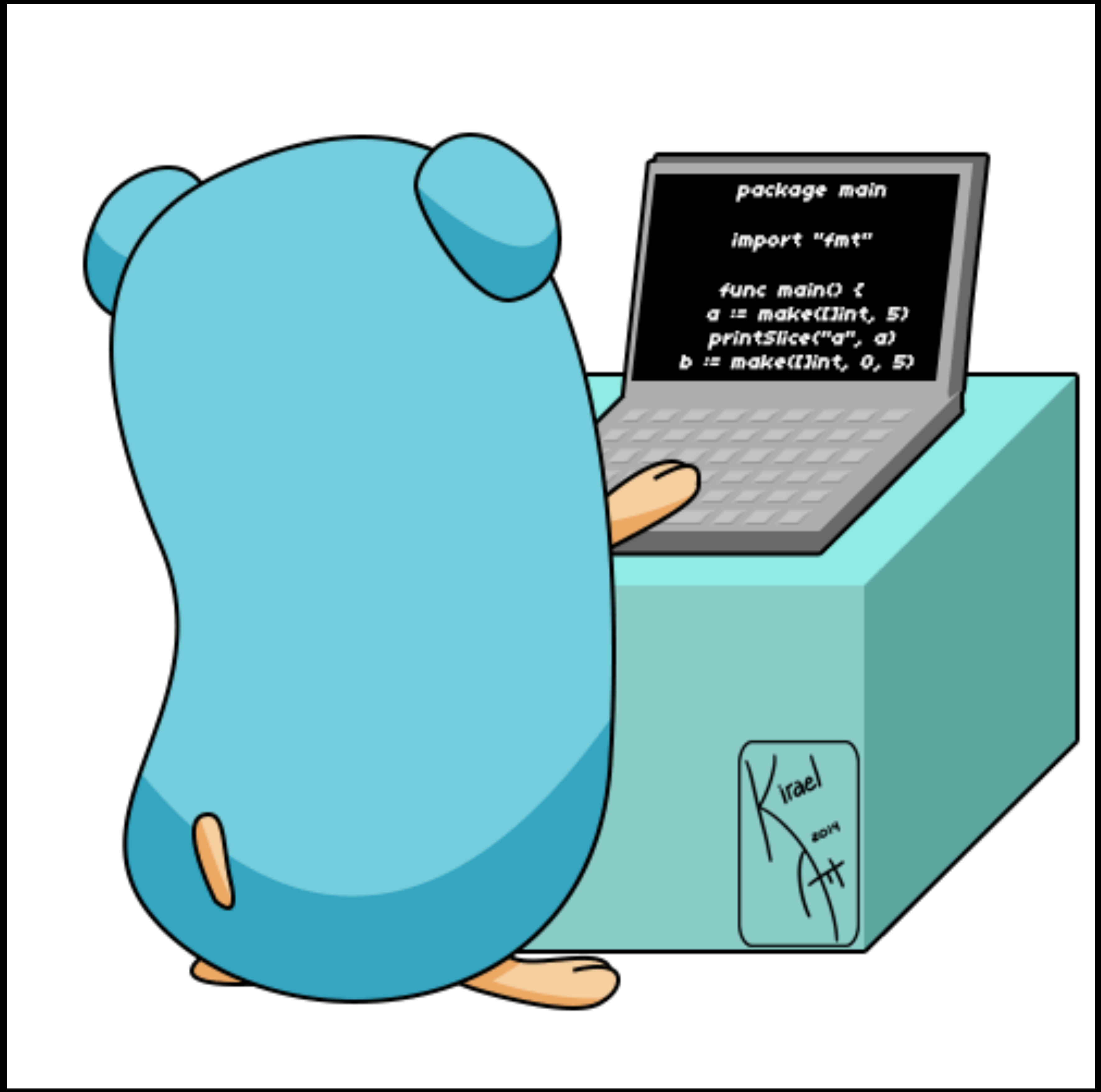
GO MODULES

- Experimental support added in Go 1.11
  - Release planned in Go 1.13
  - Already 100% functional
- A module is a versioned unit of one or more packages
- Records exact versions of dependencies
  - Typically git tags
    - » v2.3.4
  - Semantic versioning
- Reproducible builds


Major version: increment for backwards-incompatible changes.
Minor version: increment for new features.
Patch version: increment for bug fixes.
v2.3.4
(semantic versioning)
import "my/thing/v2/sub/pkg"
(semantic import versioning)

HTTPS://BLOG.SPIRALSCOUT.COM

DEMO TIME!



```
package main

import "fmt"

func main() {
a := make([]int, 5)
printSlice("a", a)
b := make([]int, 0, 5)
```

HTTPS://WWW.DEVIANTART.COM/KIRAEL-ART/ART/GO-LANG-MASCOT-458285682

POINTS OF CRITICISM

CONTINUED…

2
ERROR HANDLING

3
(NO)
GENERICS

GO 2.0 DRAFTS

GO 2.0

- Draft designs for 2.0 released to the community for feedback in sept 2018
  - Error handling
  - Error values
  - Generics
- Community feedback so far…
  - The right stuff
    - » Opinions on draft designs vary…

RANDOM STOCK PHOTO

2
ERROR HANDLING

ERROR HANDLING - GO 1.X

```go
func DoHttpPost(targetUrl string, payload MyStruct) ([]byte, error) {

    parsedUrl, err := url.ParseRequestURI(targetUrl)
    if err != nil {
        return handleError(err)
    }

    jsondata, err := json.Marshal(payload)
    if err != nil {
        return handleError(err)
    }

    resp, err := http.Post(parsedUrl.String(), "application/json", bytes.NewBuffer(jsondata))
    if err != nil {
        return handleError(err)
    }

    respData, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return handleError(err)
    }
    . . .
```

ERROR HANDLING - GO 1.X

```go
func DoHttpPost(targetUrl string, payload MyStruct) ([]byte, error) {

    parsedUrl, err := url.ParseRequestURI(targetUrl)
    if err != nil {
        return handleError(err)
    }

    jsondata, err := json.Marshal(payload)
    if err != nil {
        return handleError(err)
    }

    resp, err := http.Post(parsedUrl.String(), "application/json", bytes.NewBuffer(jsondata))
    if err != nil {
        return handleError(err)
    }

    respData, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return handleError(err)
    }
```

ERROR HANDLING - GO 2.0 PROPOSAL - WITH EXPLICIT HANDLER

```go
func DoHttpPost(targetUrl string, payload MyStruct) ([]byte, error) {
    handle err {
        return fmt.Errorf("DoHttpPost failed with %v", err)
    }
    parsedUrl := check url.ParseRequestURI(targetUrl)

    jsondata := check json.Marshal(payload)

    resp := check http.Post(parsedUrl.String(), "application/json", bytes.NewBuffer(jsondata))

    respData := check ioutil.ReadAll(resp.Body)

    // Handle response
}
```

ERROR HANDLING - GO 2.0 PROPOSAL - WITH IMPLICIT HANDLER

```go
func DoHttpPost(targetUrl string, payload MyStruct) ([]byte, error) {



    parsedUrl := check url.ParseRequestURI(targetUrl)

    jsondata := check json.Marshal(payload)

    resp := check http.Post(parsedUrl.String(), "application/json", bytes.NewBuffer(jsondata))

    respData := check ioutil.ReadAll(resp.Body)

    // Handle response
}
```

3
NO
GENERICS

RUSS COX
GOLANG TEAM

RUSS COX ON GENERICS IN 2009

”do you want ***slow programmers,***

or

slow compilers *and* ***bloated binaries,***

or

slow execution times”

RUSS COX ON GENERICS IN 2009

"do you want ***slow programmers,***

or

~~***slow compilers*** *and* ***bloated binaries***~~***,***

or

~~***slow execution times***"~~

NO GENERICS!

GO 1.0 CODE SAMPLE

```go
func Filter(items []MyStruct, filterFunc func(MyStruct) bool) []MyStruct {
    output := make([]MyStruct, 0)
    for _, item := range items {
        if filterFunc(item) {
            output = append(output, item)
        }
    }
    return output
}
```

GO 1.0 SAMPLE

```go
func Filter(items []OtherStruct, filterFunc func(OtherStruct) bool) (output []OtherStruct) {
    output := make([]OtherStruct, 0)
    for _, item := range items {
        if filterFunc(item) {
            output = append(output, item)
        }
    }
    return output
}
```

GO 2.0 CODE SAMPLE

```go
func Filter(type T)(items []T, filterFunc func(T) bool) []T {
    output := make([]T, 0)
    for _, item := range items {
        if filterFunc(item) {
            output = append(output, item)
        }
    }
    return output
}


items := []AnyStruct {{"blue", false}, {"red", true},}

// Type inferred by compiler
output := Filter(items, func(s1 AnyStruct) bool {
    return s1.IsBlue()
})

// Explicit type
output := Filter(type AnyStruct)(items, func(s1 AnyStruct) bool {
    return s1.IsBlue()
})
```

GO 2.0 GENERICS - CONTRACTS

- How to enforce that a type T has certain traits that a generic function needs to operate on?

```
private <T extends Serializable> void serializeToDisk(T item) {
```

GO 2.0 GENERICS - CONTRACTS

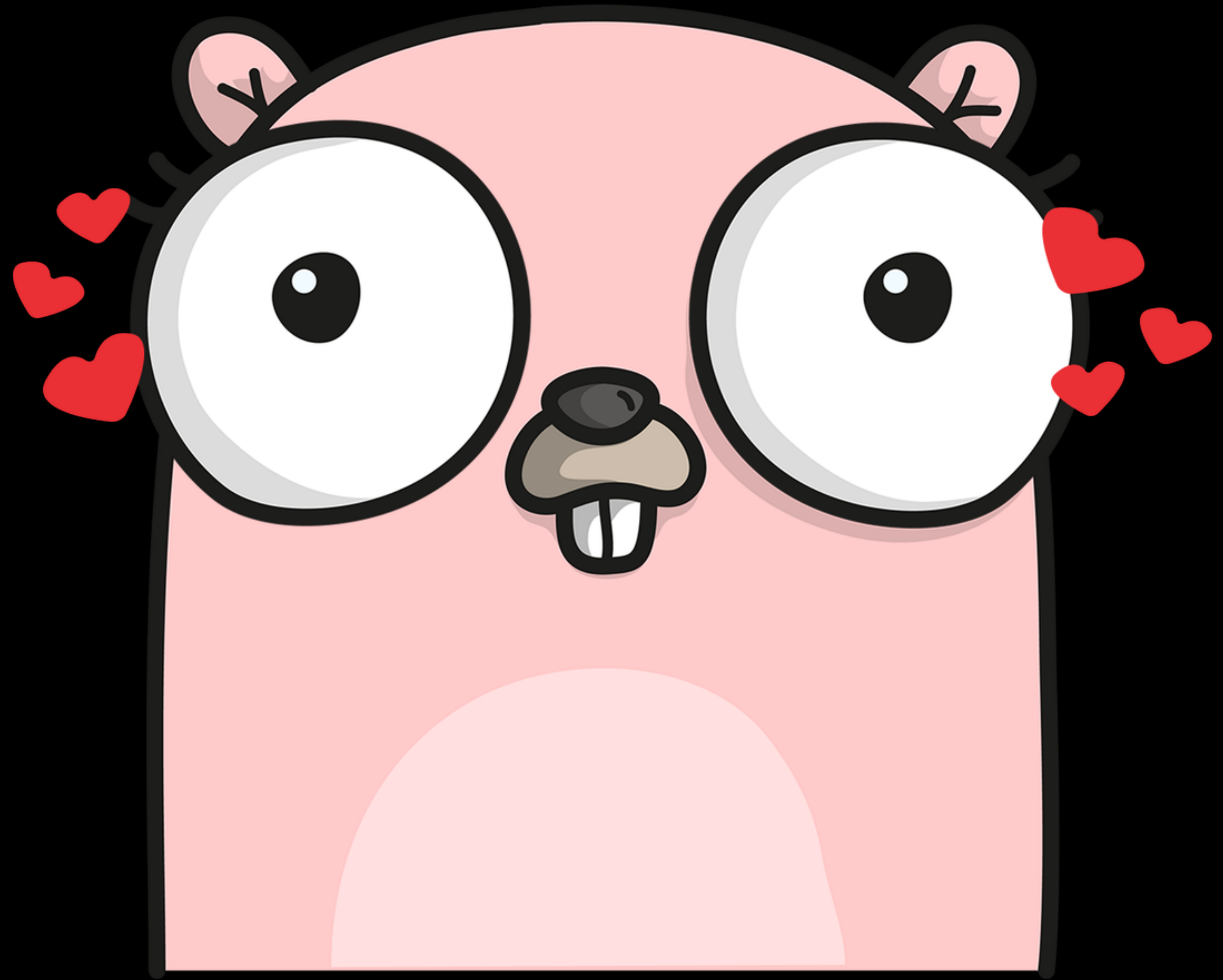
- Contracts defines what a generic type T must fulfill
  - Operators, Fields, Methods, Interface, …
- Type declarations on methods can then specify a contract for a generic type

```
contract comparable(x T) {
    x == x
}

contract stringer(x T) {
    var string = x.String()
}

func Contains(type T comparable)(items []T, element T) bool {
    for _, item := range items {
        if item == element {
            return true
        }
    }
    return false
}
```

SUMMARY

- In 2019, Go is still popular and a viable option in many domains.
- Go modules fixes one of the most common points of criticism.
- Go 2.0 drafts are definitely promising.
  - Timeline is largely unknown.
- Go should absolutely be around for many years to come.

THANK YOU!



HTTPS://GOPHERIZE.ME/