Digital Publishing for Scale

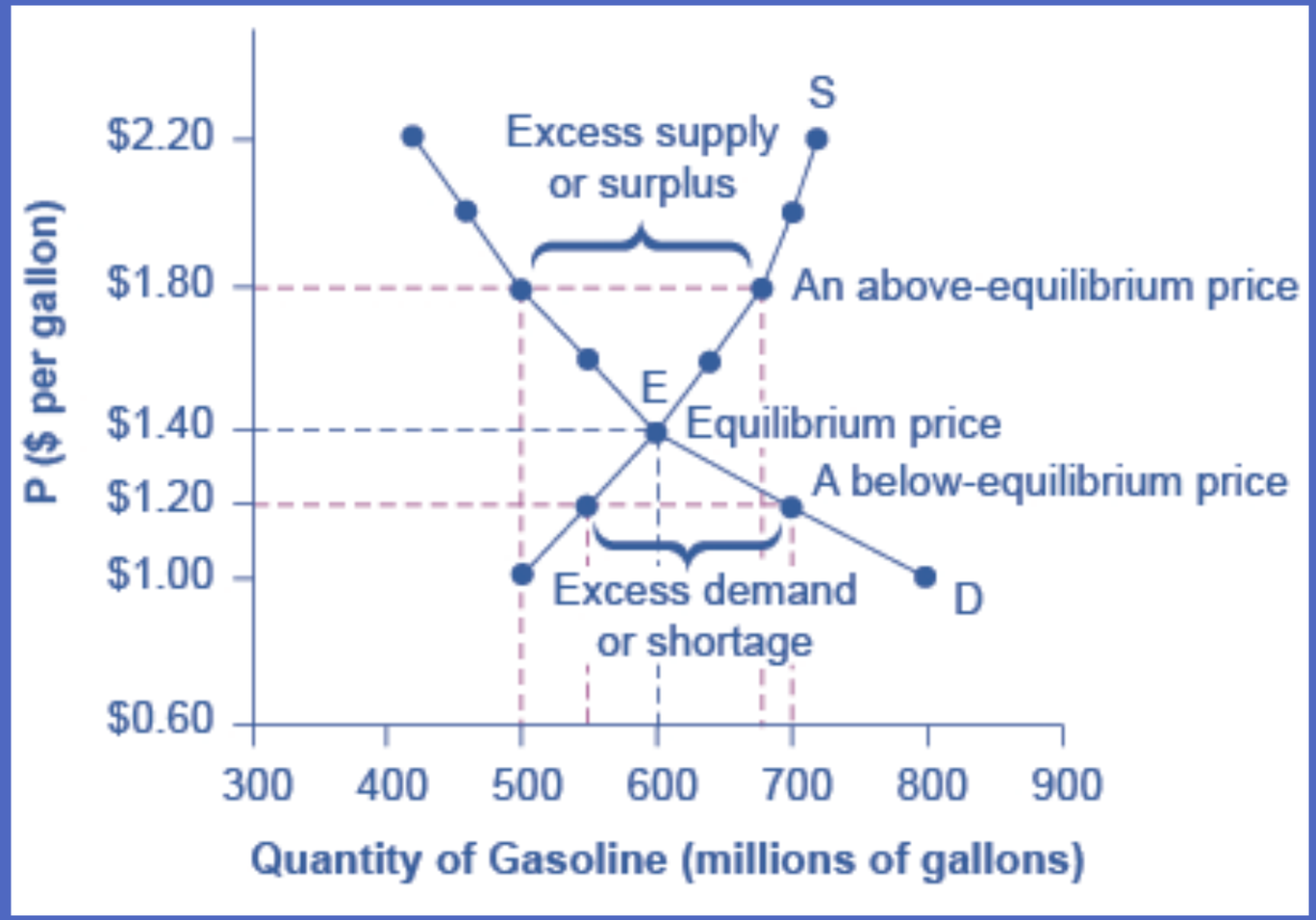
The Economist and Go

Jonas

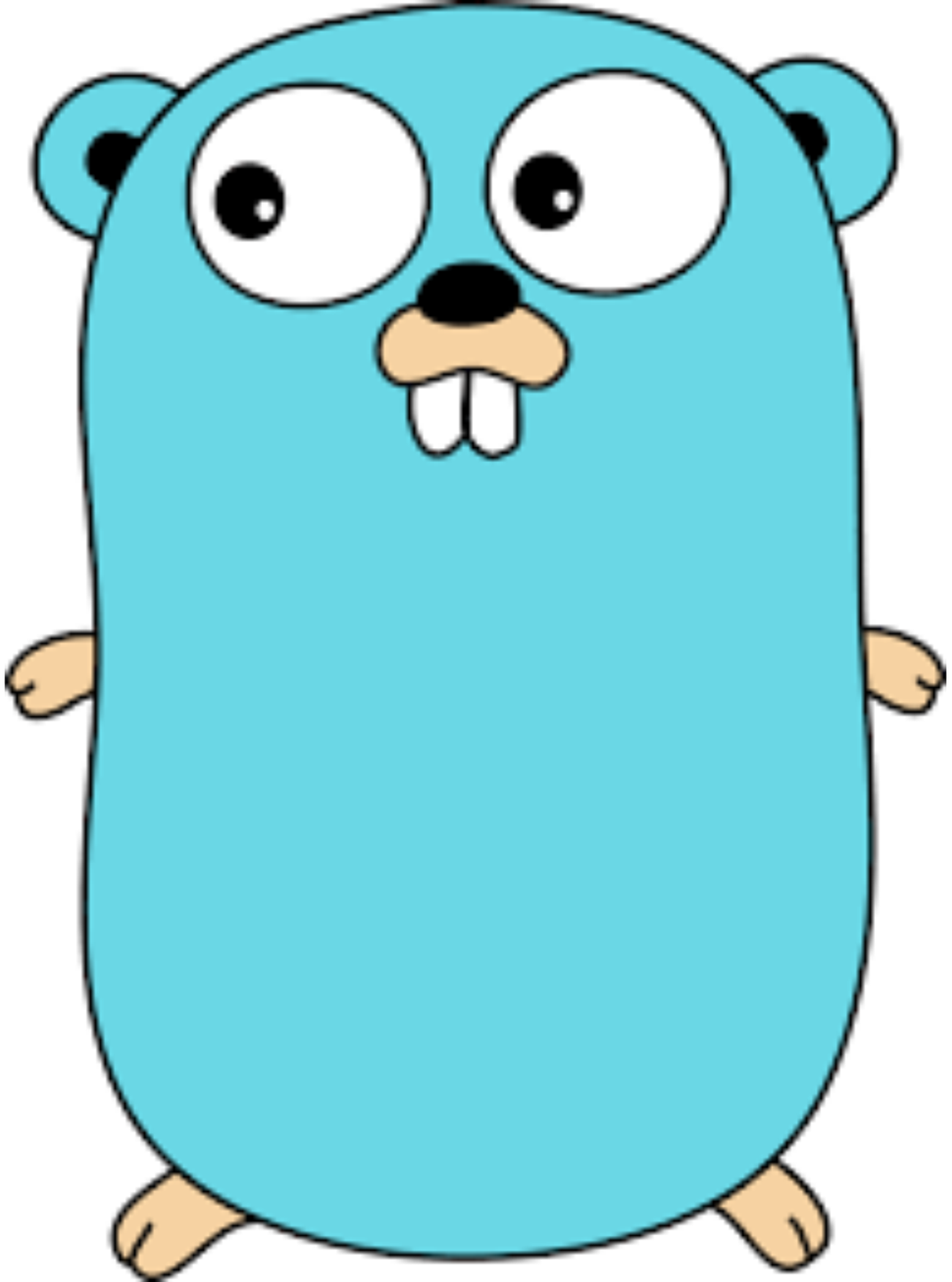
Lead Engineer, Content Platform

@yigenana

Print Pressing Forward



Print Pressing Forward



The Happy Path to Go

The Platform
* AWS hosted
* Event Messaging
* Worker microservices
* Distributed
* S3, DynamoDB, and ElasticSearch data stores
* RESTful API & GraphQL

The Happy Path to Go

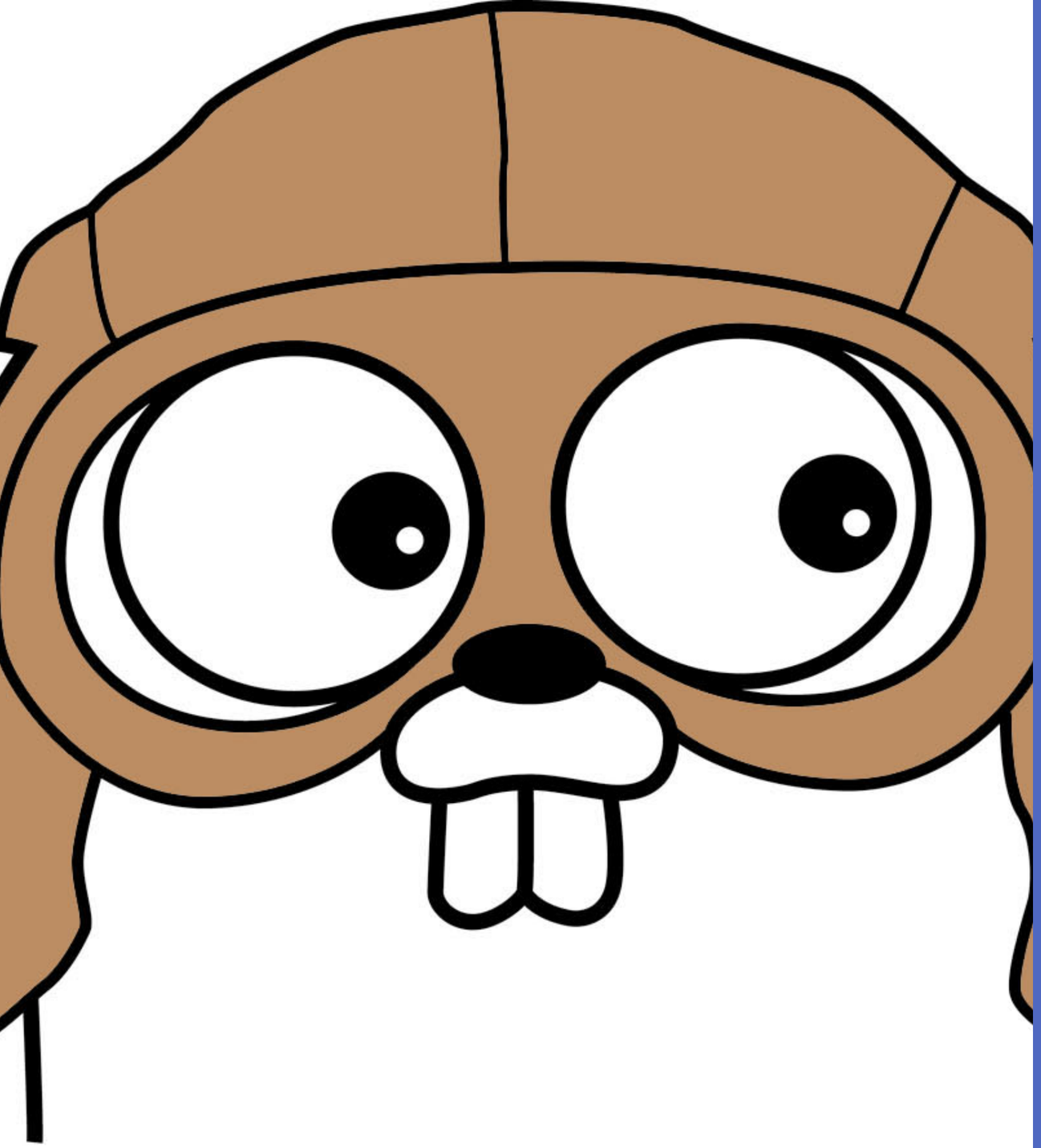
Key factors

* Built in concurrency support enables performance at scale
* Strong networking and API support
* Compiled language simple to deploy across platforms
* Simple design and syntax quickly enables developers

Fail Fast: Language Design

Application Principles

* Minimize startup time
* Fail fast
* Continuous Integration & Delivery



Fail Fast: Errors vs Exceptions

Errors vs Exception

* No exceptions
* Error as a type
* Failing fast the responsibility of the developer

Fail Fast: Error Handling

```
type error interface {
    Error() string
}
```

Fail Fast: Error Handling

```
// New returns an error that formats as the given text.
func New(text string) error {
    return &errorString{text}
}

// errorString is a trivial implementation of error.
type errorString struct {
    s string
}

func (e *errorString) Error() string {
    return e.s
}
```

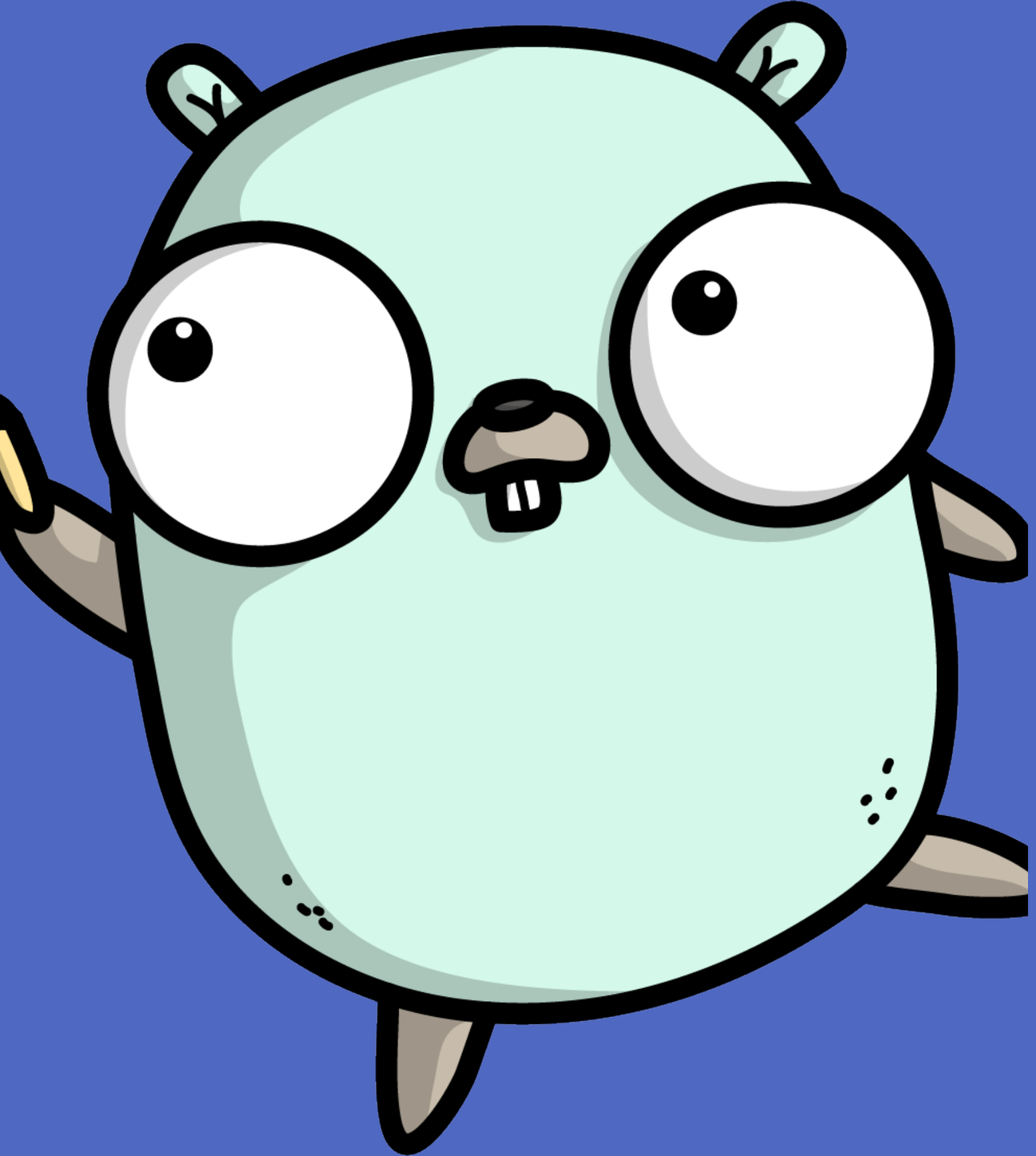
Fail Fast: Error Handling

```
func fetchContent(id string) (string, error) {
    content, err := fetch(id)
    if err != nil {
        // handle the error.
    }
    // happy path continues.
}
```

Fail Fast: Error Handling

```
package net

type Error interface {
    error
    Timeout() bool   // Is the error a timeout?
    Temporary() bool // Is the error temporary?
}

if nerr, ok := err.(net.Error); ok && nerr.Temporary() {
    time.Sleep(1e9)
    continue
}

if err != nil {
    log.Fatal(err)
}
```

From Error Handling in Go



Fail Fast: Error Handling

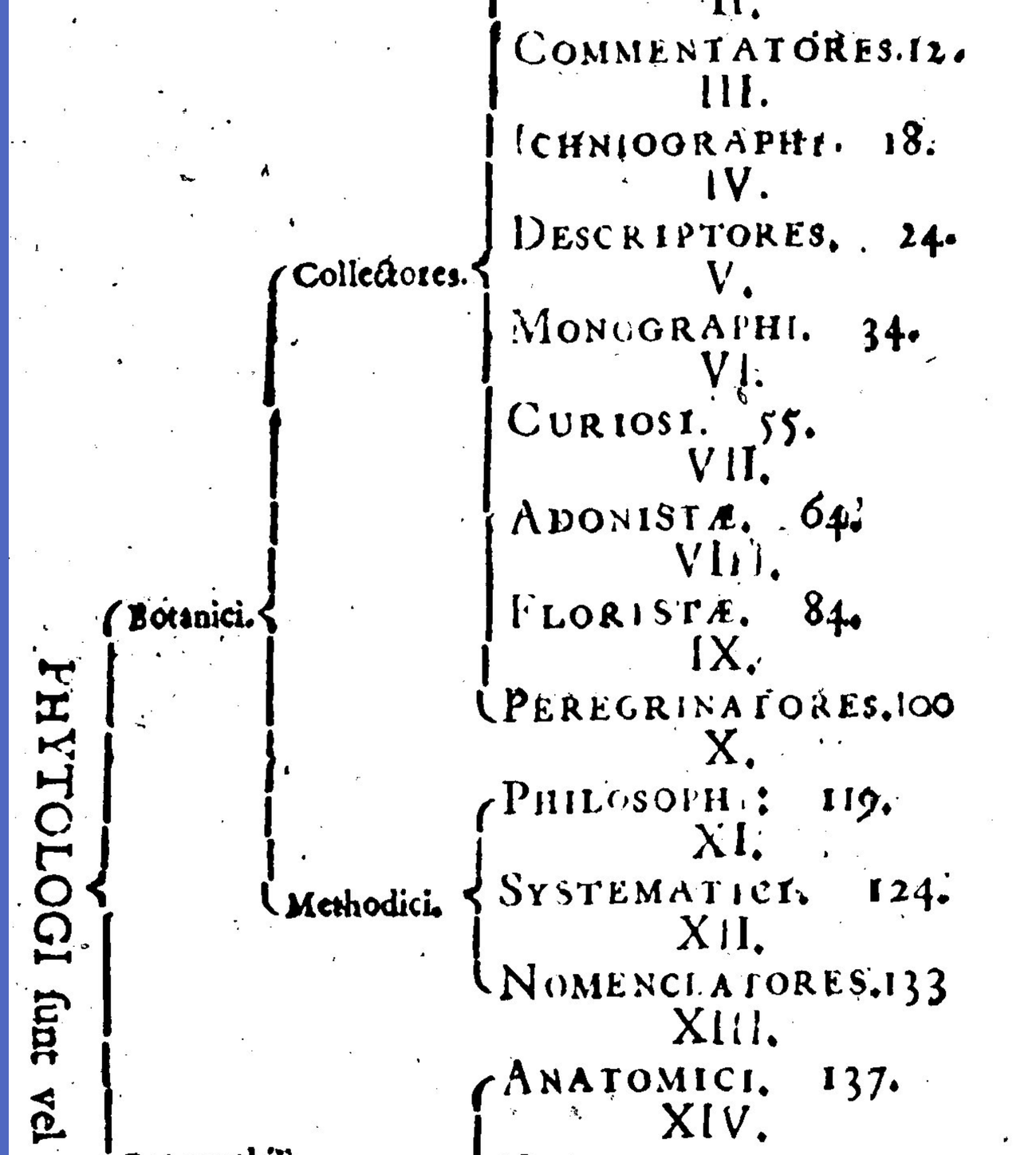
Go Error Key Benefits

* Handle errors gracefully
* Improve debugging and visibility
* More user friendly errors responses
* Greater reliability

Consistency: Language Design

The Canonical Article

* Standard data model for all content
* Aligned to schema.org standards
* Hypermedia references to associated content
* Query via GraphQL and RESTful API



Consistency: Testing

main.go

```
package main

func Sum(x int, y int) int {
    return x + y
}
```

Consistency: Testing

main_test.go

```
package main

import "testing"

func TestSum(t *testing.T) {
    total := Sum(1,2)
    if total != 3 {
        t.Errorf("Sum incorrect, got: %d, want: %d.", total, 3)
    }
}
```

Consistency: Testing

Go Test Features

* Cover: Code test coverage
* Bench: Runs benchmark tests (Benchxxx)
* TestMain: Add extra setup or teardown for tests
* Table tests and mocks

Consistency: The Challenges

One moment an empty value is 0.

```
{
    "id": "test123",
    "type": "article",
    "position": 0
}
```

The next moment it's an empty string!

```
{
    "id": "test123",
    "type": "article",
    "position": ""
}
```

Consistency: Serializing Dynamic Content

```
package json

// decodeState represents the state while decoding a JSON value.
type decodeState struct {
    data         []byte
    off          int // next read offset in data
    opcode       int // last read result
    scan         scanner
    errorContext struct { // provides context for type errors
        Struct string
        Field  string
    }
    savedError            error
    useNumber             bool
    disallowUnknownFields bool
}
```

Consistency: Serializing Dynamic Content

```
func (d *decodeState) value(v reflect.Value) error {
    case scanBeginArray:
        if v.IsValid() {
            if err := d.array(v); err != nil {
                return err
            }
        } else {
            d.skip()
        }
        d.scanNext()
    case scanBeginObject:
        if v.IsValid() {
            if err := d.object(v); err != nil {
                return err
            }
        } else {
            d.skip()
        }
        d.scanNext()
    return nil
}
```

Consistency: Serializing Dynamic Content

```
// convertNumber converts the number literal s to a float64 or a Number
// depending on the setting of d.useNumber.
func (d *decodeState) convertNumber(s string) (interface{}, error) {
    if d.useNumber {
        return Number(s), nil
    }
    f, err := strconv.ParseFloat(s, 64)
    if err != nil {
        return nil, &UnmarshalTypeError{Value: "number " + s, Type: reflect.TypeOf(0.0), Offset: int64(d.off)}
    }
    return f, nil
}
```

Consistency: Serializing Dynamic Content

```
package canonical

// traverseState represents current state of tree traversal.
type traverseState struct {
    object   *s3.Object
    tags     *traverseTags
    field    reflect.StructField
    treePath string
}
```

Consistency: Serializing Dynamic Content

```
// traverse is a top level tree traversal function.
func (t traverseState) traverse(v reflect.Value) {
    switch v.Kind() {
    case reflect.Struct:
        switch {
        case t.field.Tag.Get(refTag) != "":
            t.setRef(v)
        default:
            t.traverseStruct(v)
        }
    case reflect.Slice:
        if t.field.Tag.Get(pathTag) != "" {
            t.setSlice(v)
        }
    case reflect.String, reflect.Bool, reflect.Int, reflect.Float64:
        if t.field.Tag.Get(pathTag) != "" {
            t.setPrimitive(v)
        }
    default:
        logger.Warnln("Unknown kind:", v)
    }
}
```

Consistency: Serializing Dynamic Content

```
func toInt(v reflect.Value) int64 {
    switch v.Type {
    case int:
        return v.Int()
    case string:
        i, err := strconv.Atoi(v.String())
    default:
        return 0
    }
}
```

Consistency: Serializing Dynamic Content

```
BenchmarkCustomSerializer-1          1000          1345080 ns/op
BenchmarkCustomSerializer-2          1000          1410146 ns/op
BenchmarkCustomSerializer-3          1000          1348061 ns/op
BenchmarkCustomSerializer-4          1000          1343668 ns/op
BenchmarkCustomSerializer-5          1000          1411710 ns/op

BenchmarkStandardUnmarshal-1         1000          1451230 ns/op
BenchmarkStandardUnmarshal-2         1000          1771896 ns/op
BenchmarkStandardUnmarshal-3         1000          1396645 ns/op
BenchmarkStandardUnmarshal-4         1000          1720717 ns/op
BenchmarkStandardUnmarshal-5         1000          1475672 ns/op
```

Scale: HTTP and concurrency

```
package main

import (
    "fmt"
    "log"
    "net/http"
)

func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hello World!")
}

func main() {
    http.HandleFunc("/", handler)
    log.Fatal(http.ListenAndServe(":8080", nil))
}
```

Scale: HTTP and concurrency

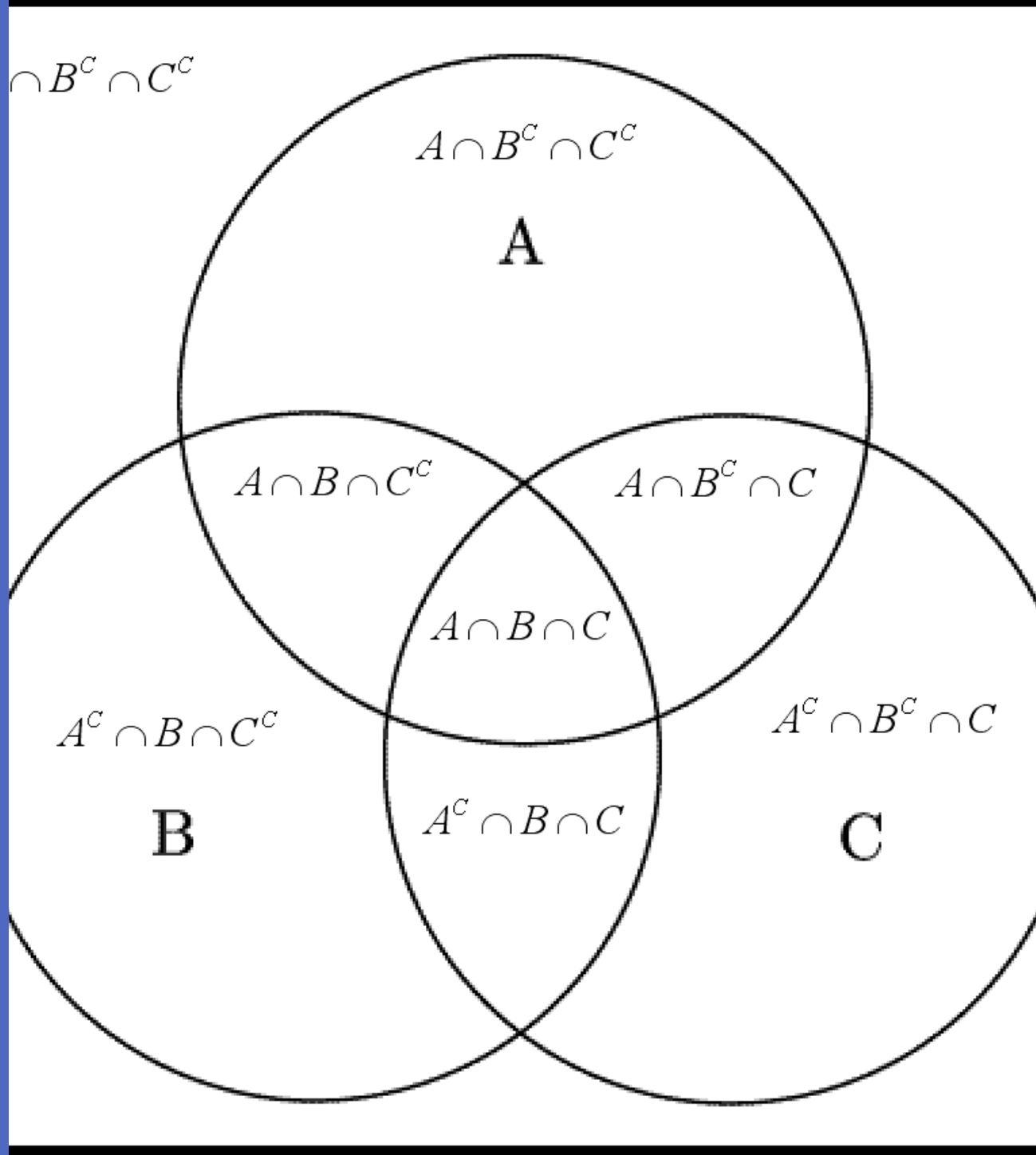
```
func (srv *Server) Serve(l net.Listener) error {
    baseCtx := context.Background() // base is always background, per Issue 16220
    ctx := context.WithValue(baseCtx, ServerContextKey, srv)
    for {
        rw, e := l.Accept()
        if e != nil {
            select {
            case <-srv.getDoneChan():
                return ErrServerClosed
            default:
            }
            if ne, ok := e.(net.Error); ok && ne.Temporary() {
                continue
            }
            return e
        }
        tempDelay = 0
        c := srv.newConn(rw)
        c.setState(c.rwc, StateNew) // before Serve can return
        go c.serve(ctx)
    }
}
```

Scale: Content Guarantees

CAP Theorum

Pick two

* Consistency
* Availiabity
* Partition Tolerance

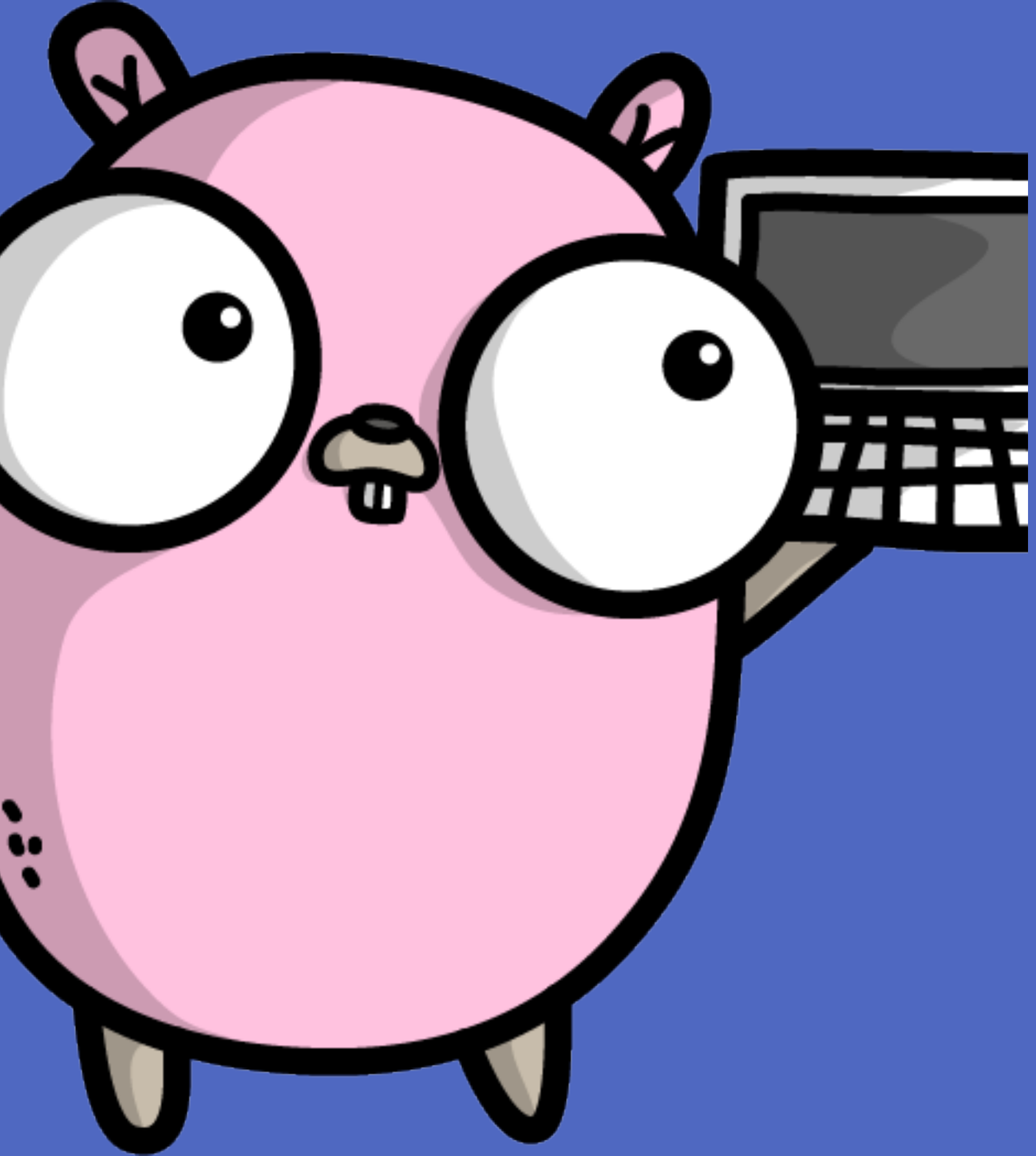


Scale: Content Guarantees

```
func reprocess(searchResult *http.Response) (int, error) {
    responses := make([]response, len(searchResult.Hits.Hits))
    var wg sync.WaitGroup
    wg.Add(len(responses))

    for i, hit := range searchResult.Hits.Hits {
        wg.Add(1)
        go func(i int, item elastic.SearchHit) {
            defer wg.Done()
            code, err := reprocessItem(item)
            responses[i].code = code
            responses[i].err = err
        }(i, *hit)
    }
    wg.Wait

    return http.StatusOK, nil
}
```

Scale: Content Guarantees

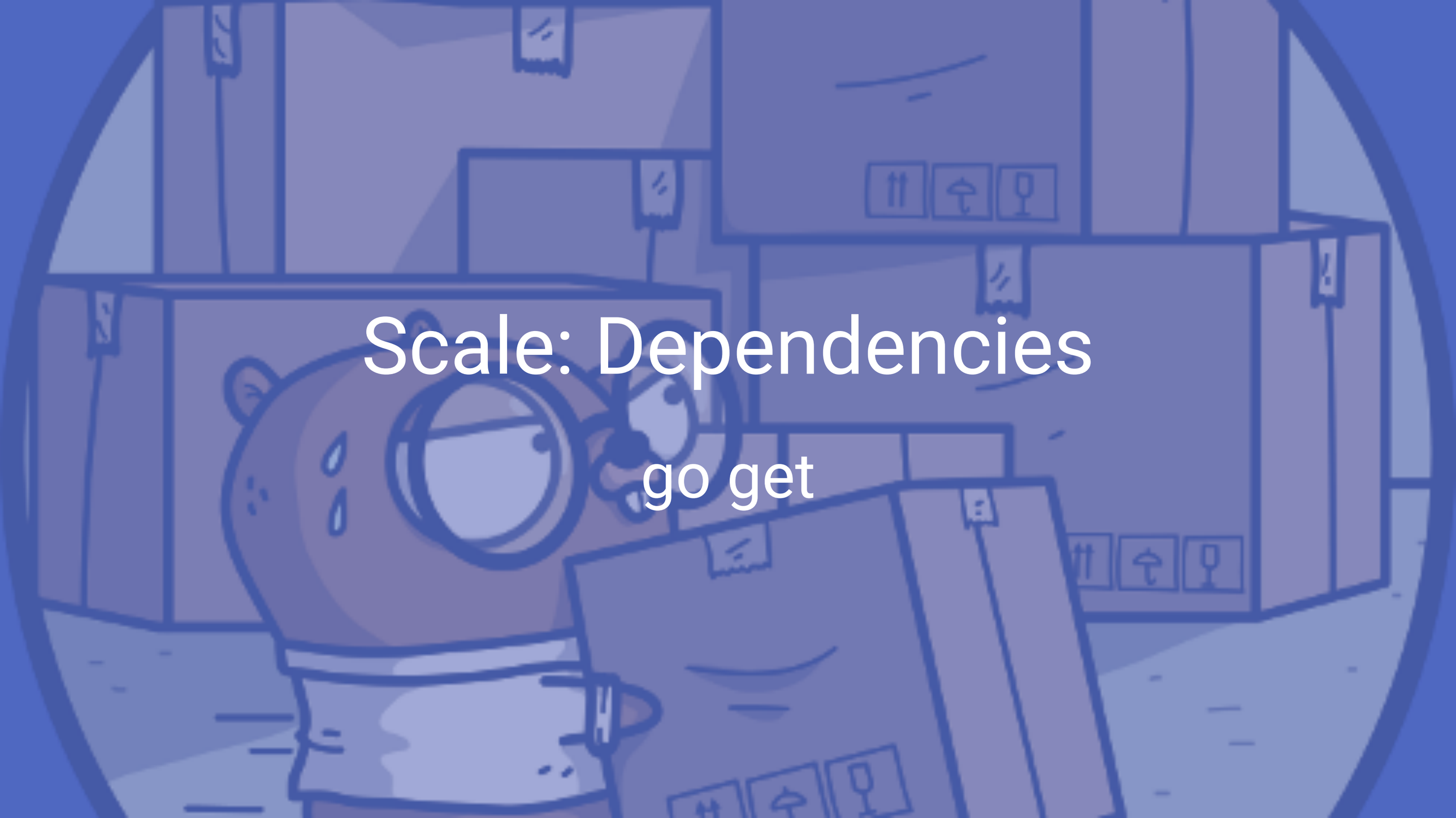
```
ProcessTimeAsync-1          20      564.030301ms
ProcessTimeAsync-2          20      813.193206ms
ProcessTimeAsync-3          20      564.536223ms
ProcessTimeAsync-4          20      830.068246ms
ProcessTimeAsync-5          20      865.895741ms

ProcessTimeSync-1           20      3.806562215s
ProcessTimeSync-2           20      4.666270193s
ProcessTimeSync-3           20      4.206750535s
ProcessTimeSync-4           20      3.745507495s
ProcessTimeSync-5           20      4.109966063s
```



Scale: Visibility with Profiling

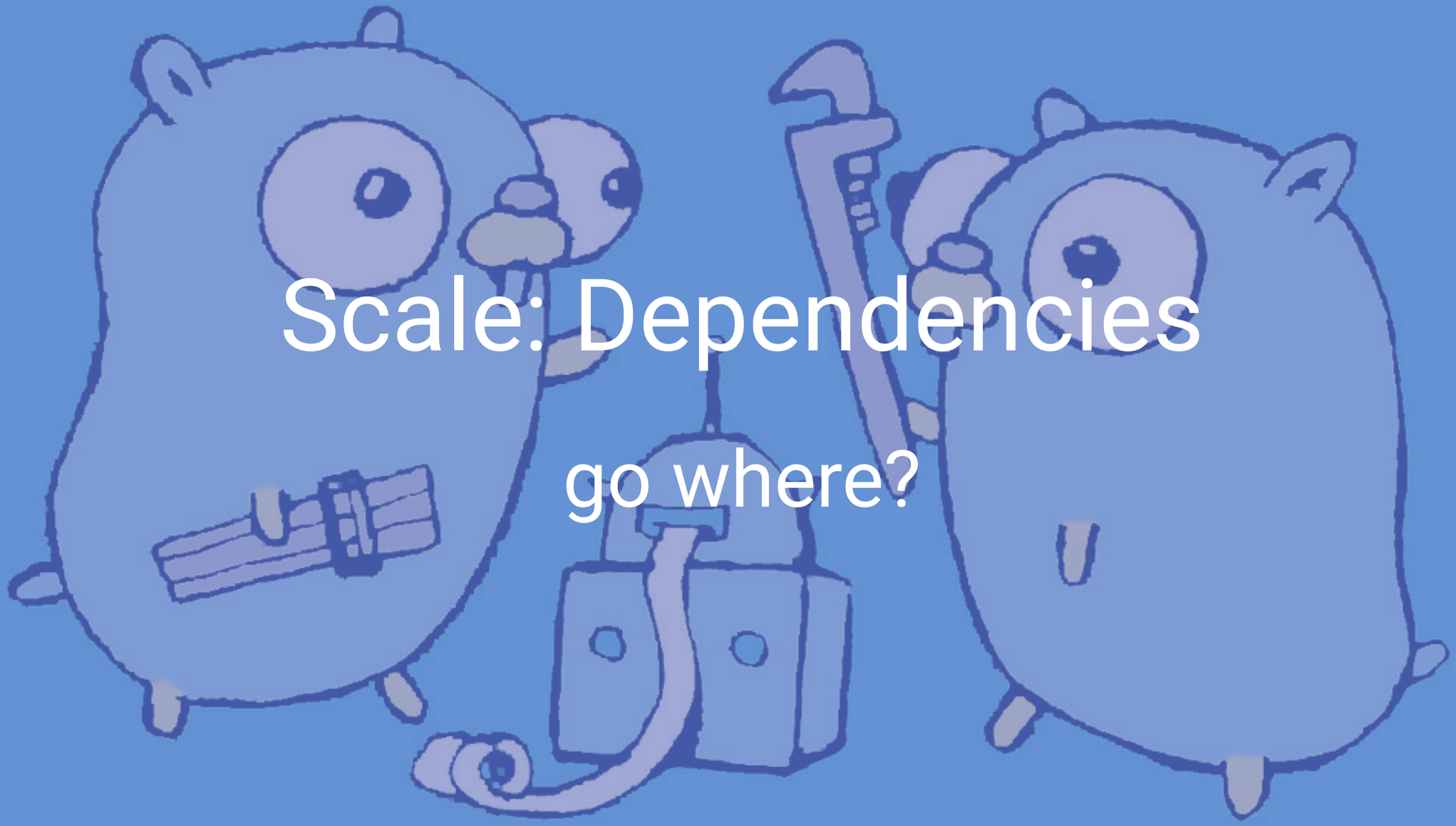
PPROF

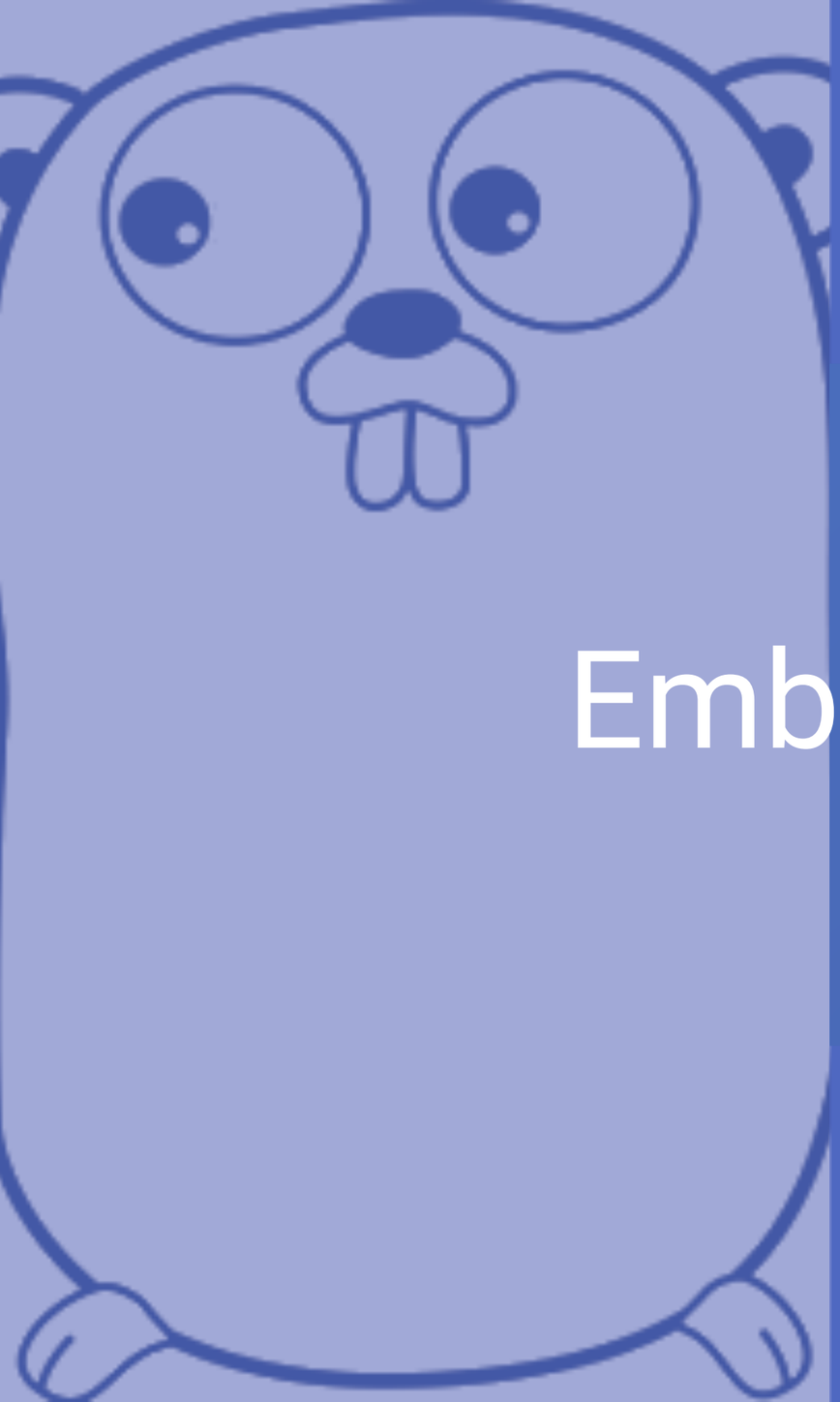
* CPU profiles, traces, heap profiles, and mutex profiles.
* CLI tool
* HTTP Endpoint
* Visualizations


Scale: Dependencies
go get

Scale: Dependencies

go where?





Embracing Limitations

Embracing Limitations: Metadata Parsing

```
// runExif runs the exiftool command with a set of arguments.
func runExif(args []string) ([]byte, error) {
    cmdOut, err := exec.Command("exiftool", args...).Output()
    if err != nil {
        return nil, err
    }
    return cmdOut, nil
}
```

Embracing Limitations: HTML Maddness

```
<br><em>• Percentage change: shows the change in
inflation-adjusted prices between two selected dates</em></p>
<p><em><span class="fontcolor-red"><span class="fontcolor-red">
<span class="fontcolor-black"><span class="fontcolor-red">
We also publish interactive house-price guides to</span>
&amp;nbsp;</span><a href="\"http://www.economist.com/houseprices\&amp;quot;">global markets</a></span>
&amp;nbsp;<span class="fontcolor-red">and&amp;nbsp;</span>
<em><span class="fontcolor-red"><span class="fontcolor-red">
<span class="fontcolor-black"><a href="http://www.economist.com/blogs/graphicdetail/2014/04/british-house-prices">British regions</a>.
</span></span></span></em></span></em></p>
```

Embracing Limitations: Simple JSON handling

```
// getLane returns lane information from the
// envelope. It uses jsonpointer to access the field
// as we want to build on this in a subsequent iteration.
const getLane = (event) => {
  const target = '/envelope/responseElements/queue-source';
  return pointer.has(event, target) && pointer.get(event, target);
}
```

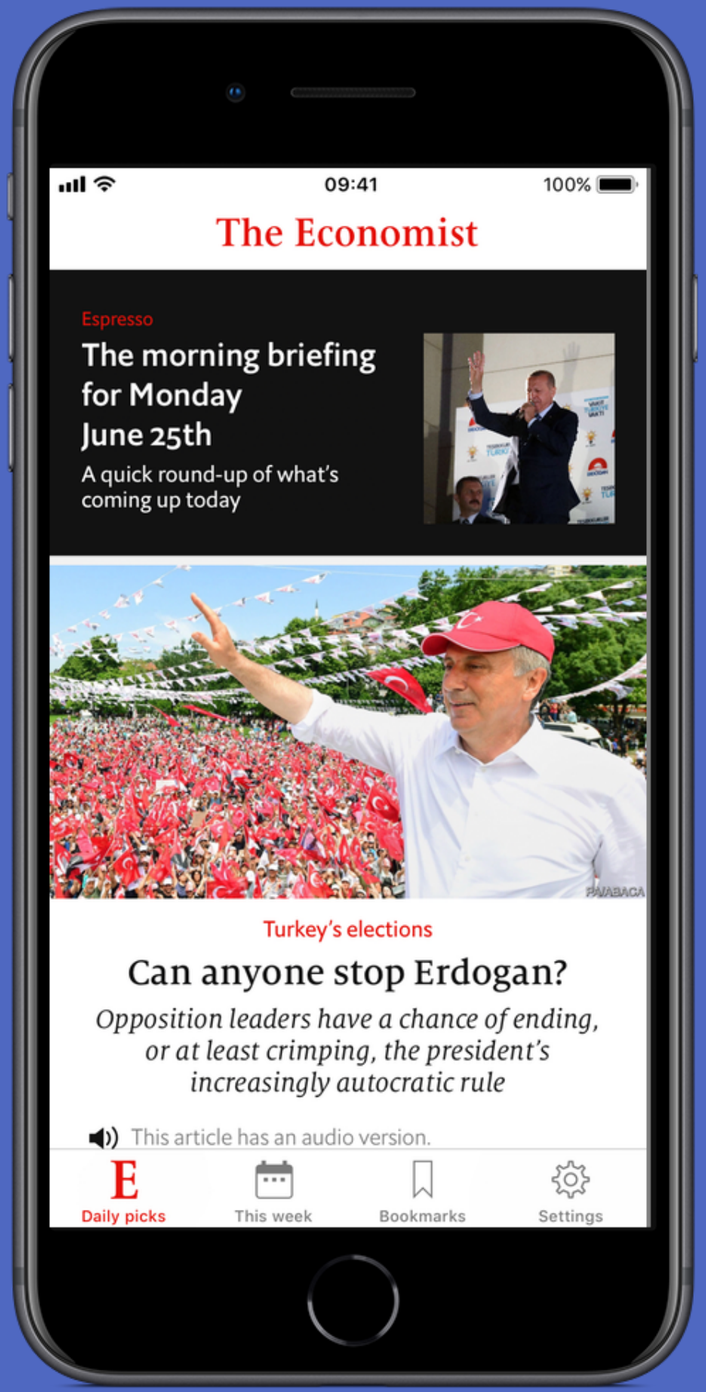
GOing Retro

What Went Well?

* Key language design elements for distributed systems
* Concurrency model quick to implement
* Robust Go tooling that fits into workflow
* Fun to write and fun community

What Could be Improved?

* Versioning and vendoring lack standards
* Lacks maturity in some areas
* Verbose for certain use cases

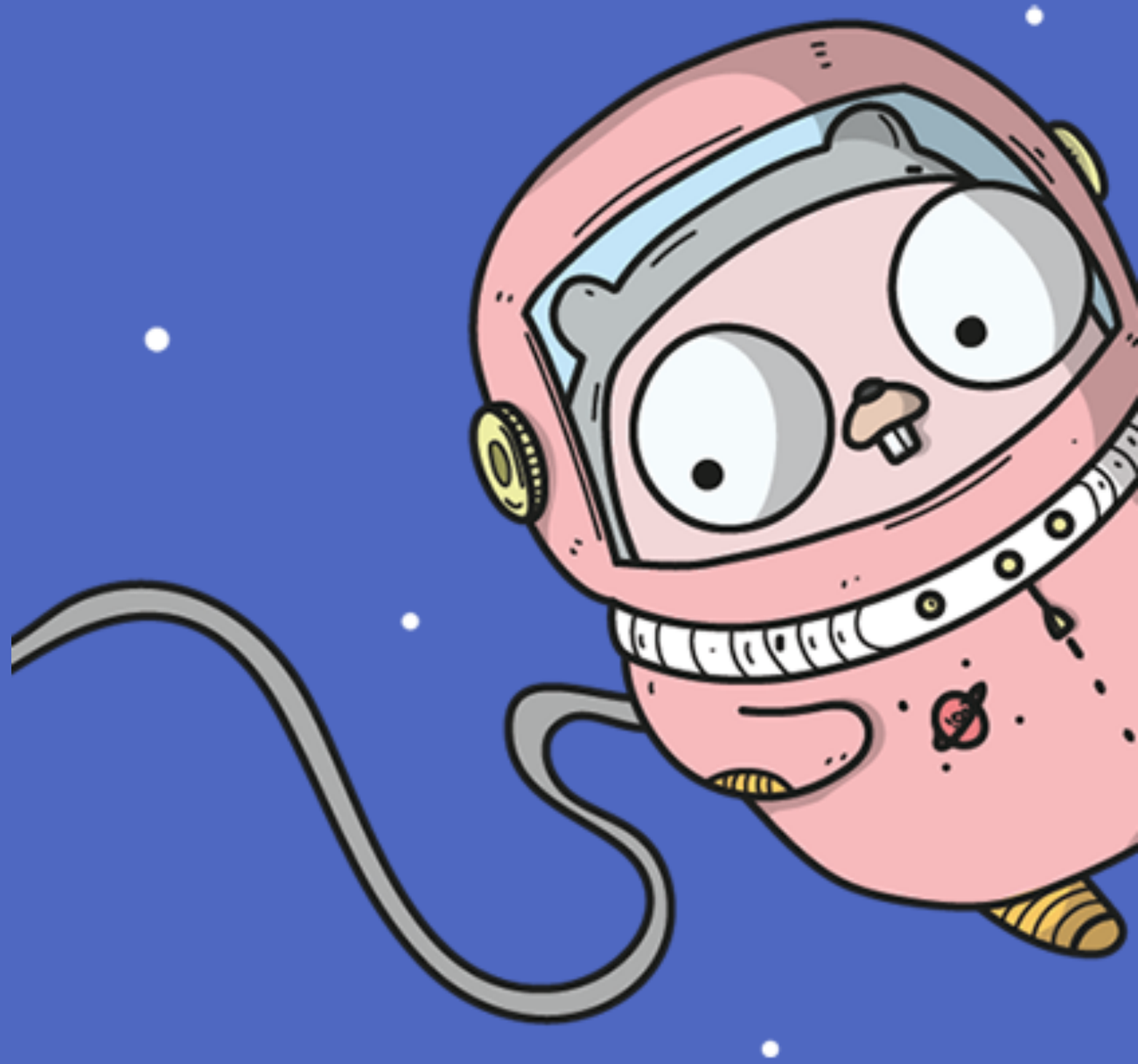


GOing Retro: Scale

GOing Retro: Design

Systems Design

* What are your system goals?
* What guarantees are you providing your consumers?
* What architecture and patterns are right for your system?
* How will your system need to scale?



Thank You!

@yigenana

Adorable Gophers by Ashley McNamara