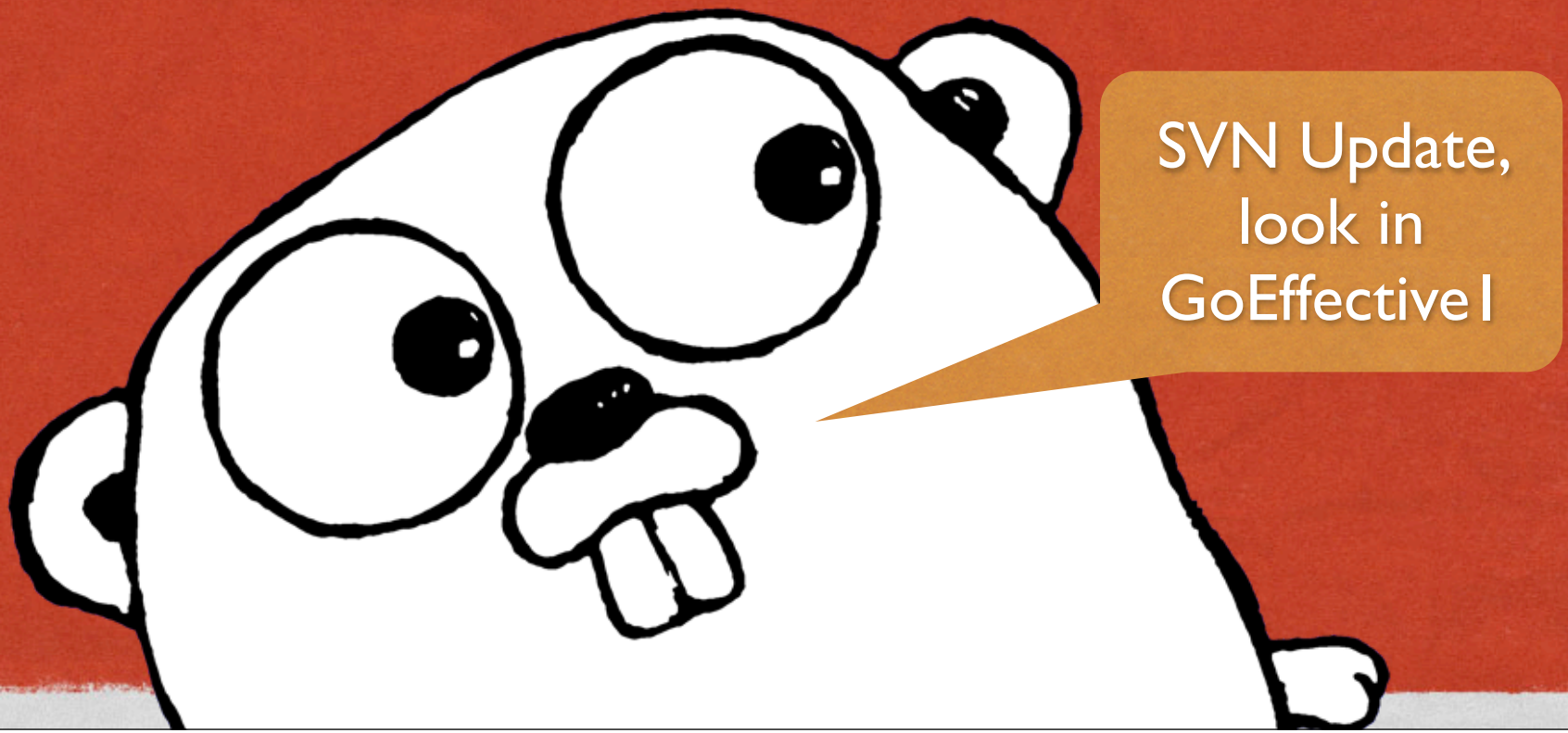
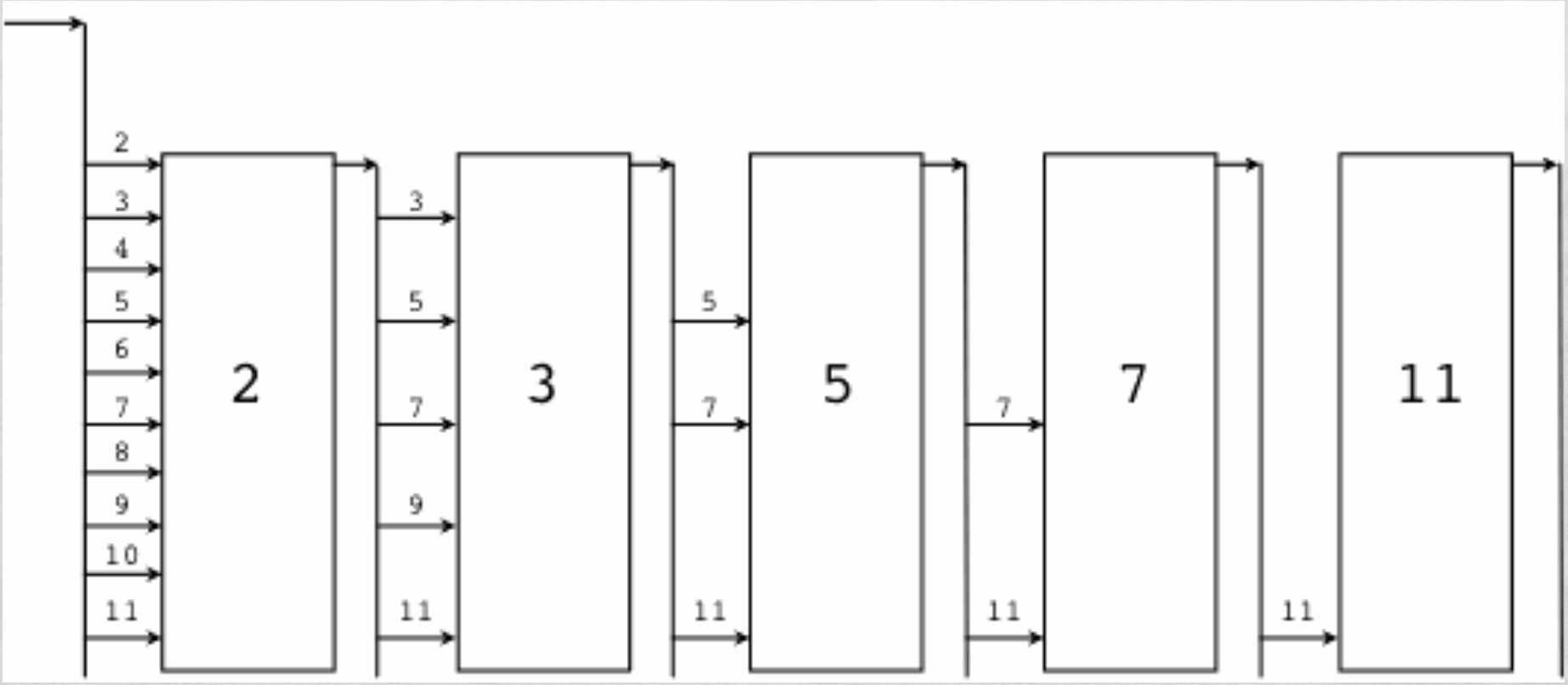
EFFECTIVE GO

Curt Clifton
Rose-Hulman Institute of Technology



PRIME NUMBER SIEVE



http://golang.org/doc/go_tutorial.html

SERVER

- Multiplexing with channels
- Control channels

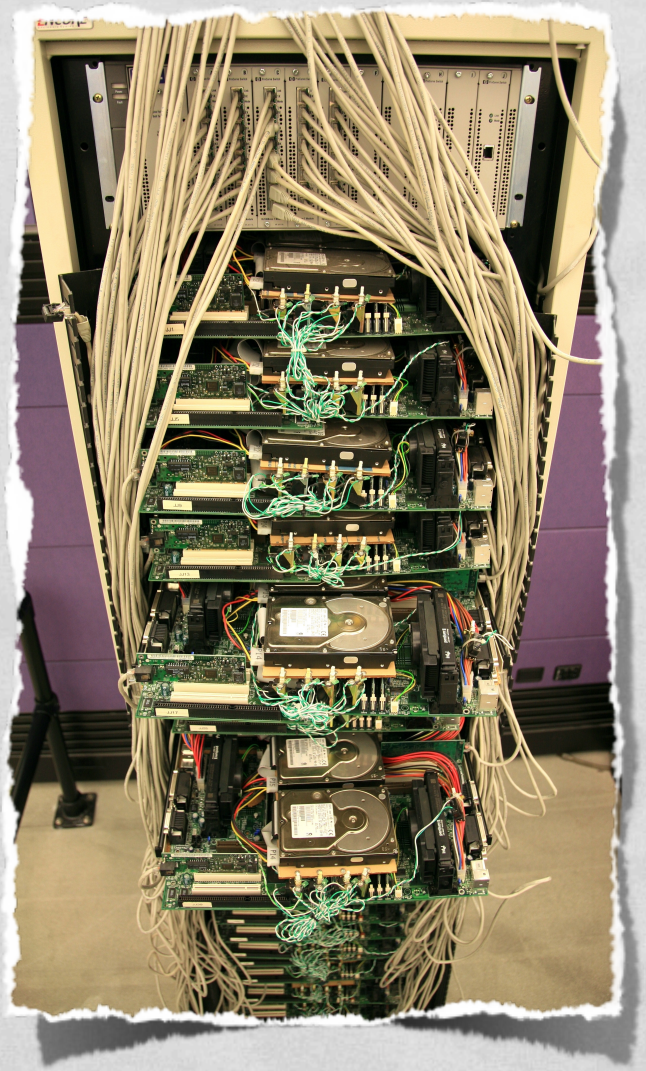


Photo by jurvetson - http://flic.kr/p/eWnwZ

GOLANG RESOURCES

- Package search
- FAQ
- Spec

GODOC

GODOC – PACKAGE

- Block comment *immediately* before *package* clause
- Introduces entire package
- No extra formatting!

GODOC – FUNCTION

- Document every exported function
- Use complete setences.
- Start with one-sentence summary, that starts with function name

GODOC — VAR DECLS

- Group related vars and use single comment

```
// Error codes returned by failures to parse an expression.
var (
    ErrInternal      = os.NewError("internal error")
    ErrUnmatchedLpar = os.NewError("unmatched '('")
    ErrUnmatchedRpar = os.NewError("unmatched ')'")
    ...
)
```

Photo by dorena-wm - http://flic.kr/p/8jvsso

WHAT'S IN A NAME

NAMES – PACKAGES

- Lower case
- Single word
- Err toward brevity

- Package location:
  - *src/pkg/container/vector*
- Declared with:
  - *package "vector"*
- Imported with:
  - *import "container/vector"*
- Used like:
  - *iv := new(vector.IntVector)*

NAMES – TYPES, FUNCTIONS

- Don’t stutter
- To users, package name is part of name
- Use CamelCase, not python_style_names

CONTROL

CONTROL – IF

- Use initialization statement

```
if err := file.Chmod(0664); err != nil {
    log.Print(err)
    return err
}
```

CONTROL – IF

- Omit unnecessary *else*

```
f, err := os.Open(name, os.O_RDONLY, 0)
if err != nil {
    return err
}
d, err := f.Stat()
if err != nil {
    return err
}
codeUsing(f, d)
```

CONTROL – FOR

- Like C's for:
  - **for init; condition; post { ... }**
- Like C's while:
  - **for condition { ... }**
- Like C forever:
  - **for { ... }**

CONTROL – FOR

- Multiple loop variables?
- Use parallel assignment:
  - ```
    for i, j := 0, len(a)-1; i < j; i, j = i+1, j-1 {
        a[i], a[j] = a[j], a[i]
    }
    ```
    ```

CONTROL – SWITCH

- Naked switch is essentially if-else-if

```
func unhex(c byte) byte {
    switch {
    case '0' <= c && c <= '9':
        return c - '0'
    case 'a' <= c && c <= 'f':
        return c - 'a' + 10
    case 'A' <= c && c <= 'F':
        return c - 'A' + 10
    }
    return 0
}
```

CONTROL – SWITCH

- No auto fall-through

"Standard" switch

```
func shouldEscape(c byte) bool
{
    switch c {
    case ' ', '?', '&', '=', '#', '+', '%':
        return true
    }
    return false
}
```

Commas to group cases

CONTROL – TYPE SWITCH

Default can come first!

Special syntax

```
switch t := interfaceValue.(type) {
default:
    fmt.Printf("unexpected type %T", t)  // %T prints type
case bool:
    fmt.Printf("boolean %t\n", t)
case int:
    fmt.Printf("integer %d\n", t)
case *bool:
    fmt.Printf("pointer to boolean %t\n", *t)
case *int:
    fmt.Printf("pointer to integer %d\n", *t)
}
```

Variable has type from the case

DEFER MADNESS

```
func Contents(filename string) (string, os.Error) {
    f, err := os.Open(filename, os.O_RDONLY, 0)
    if err != nil {
        return "", err
    }
    defer f.Close()

    var result []byte
    buf := make([]byte, 100)
    for {
        n, err := f.Read(buf[0:])
        result = bytes.Add(result, buf[0:n])
        if err != nil {
            if err == os.EOF {
                break
            }
            return "", err
        }
    }
    return string(result), nil
}
```

Close later!

Either exit path is followed by f.Close()

DEFER – LIFO ORDER

```
for i := 0; i < 5; i++ {
    defer fmt.Printf("%d ", i)
}
```

4 3 2 1 0

DEFER – ARGS EARLY, FN LATE

```
func trace(s string) string {
    fmt.Println("entering:", s)
    return s
}

func un(s string) {
    fmt.Println("leaving:", s)
}

func a() {
    defer un(trace("a"))
    fmt.Println("in a")
}
```

```
func b() {
    defer un(trace("b"))
    fmt.Println("in b")
    a()
}

func main() {
    b()
}
```

```
entering: b
in b
entering: a
in a
leaving: a
leaving: b
```