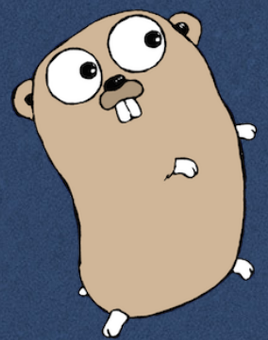


on a Plane

Practical Considerations for Instrument Control in Go

Where are we going

- *Overview of AVAPS® instrumentation*
- *Design criteria*
- *Ok.... so we chose the blue pill*
- *Lessons learned*
- *Further resources*

AVAPS®
Instrumentation

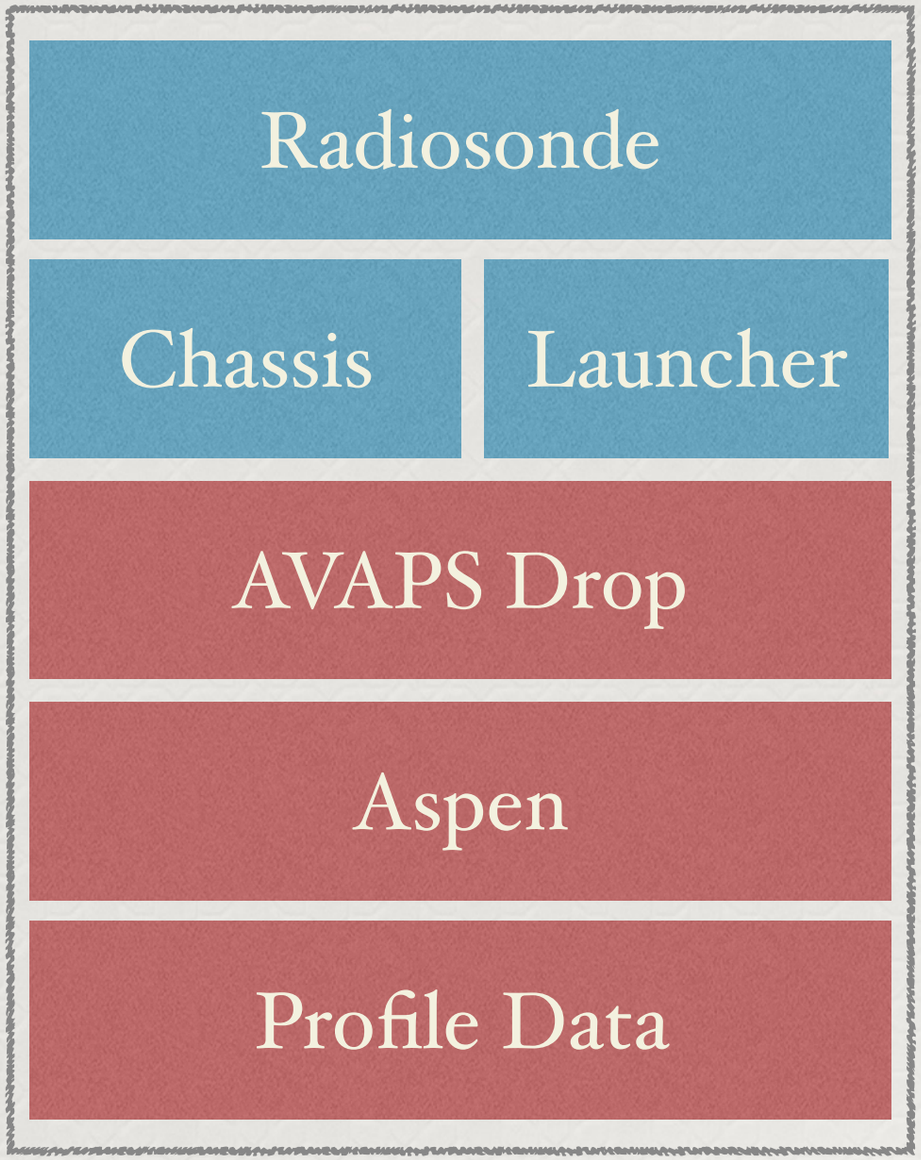
Tooling for targetable in-situ atmospheric profiles

AVAPS® Instrumentation

- *Targetable, high resolution atmospheric profiling instrument*
- *Expendable are ejected from aircraft while samples are radioed back to plane*
- *Main users are operationally driven for hurricane landfall / evacuation:*
  - *NOAA Aircraft Operations Center (Tampa/Lakeland, Fl)*
  - *US Airforce 53rd Weather Recon Wing (Biloxi, Ms)*
- *Other Research Platforms*
  - *NCAR/NSF G-V, C-130*
  - *NASA Global Hawk*
  - *DLR Falcon & G-550*
  - *UK Met Office*
  - *University of Taiwan*

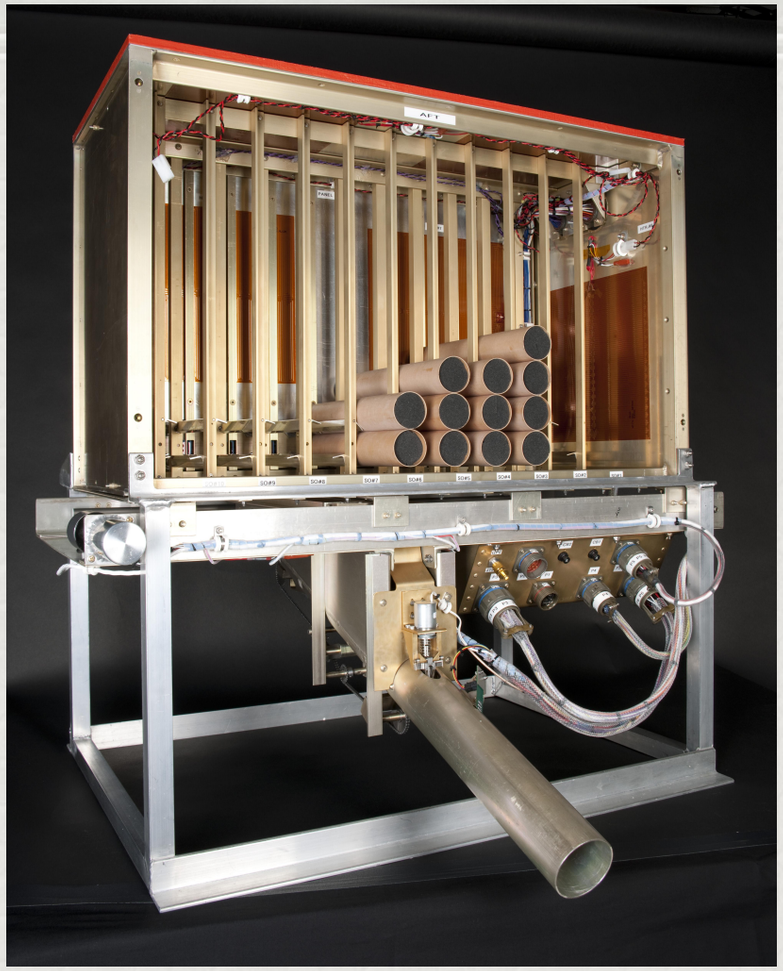
System Components

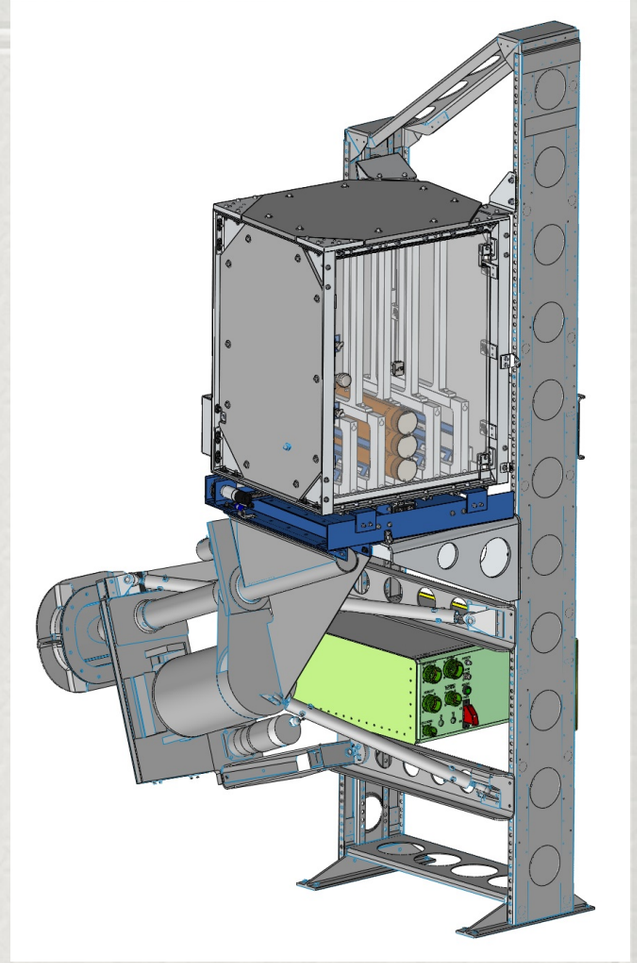
- *AVAPS Drop controls hardware elements, and records sonde data*
- *Aspen performs QC on raw data*
- *Profile data is final product*




WESTCOTT
STAINLESS STEEL

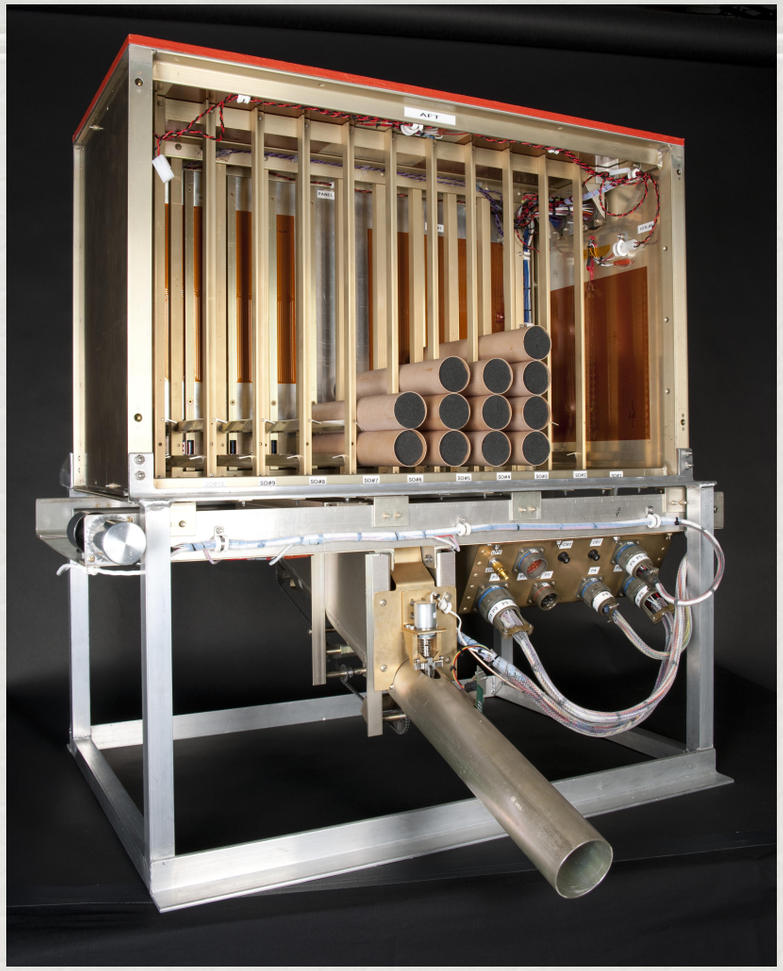

WESTCOTT
STAINLESS STEEL

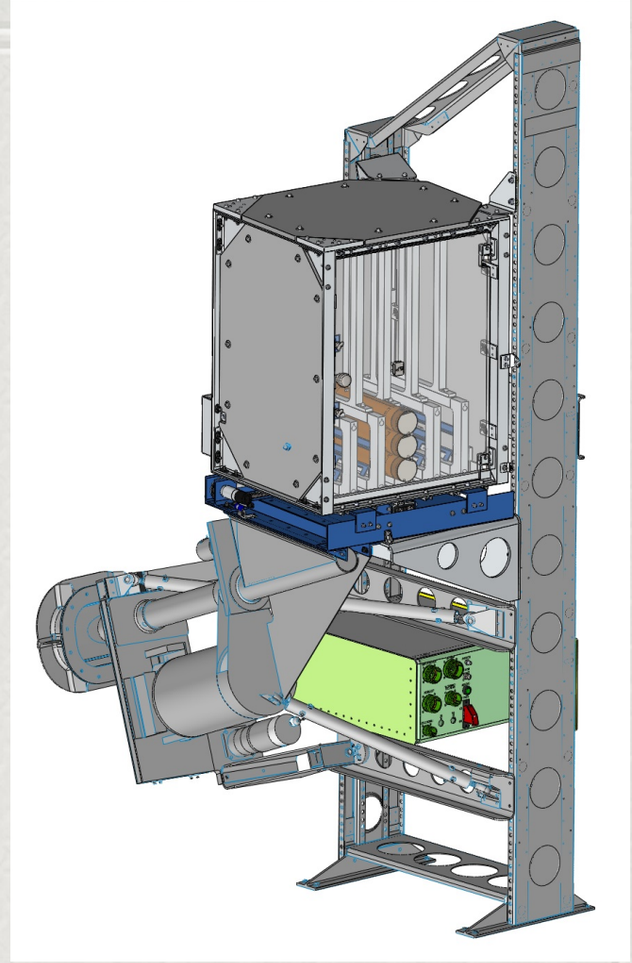

AFT

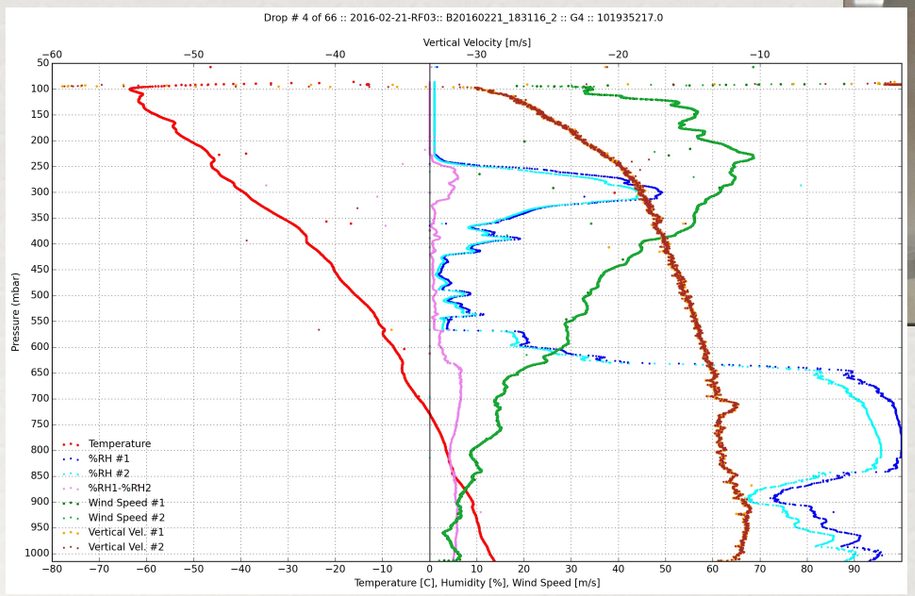



ABSOPULSE
ELECTRONICS LTD.
T.P.
+
−
Vadj
Vout
POWER SUPPLY
SIGNAL STRENGTH
DATA LOCK
15VDC/13A
TELEMETRY RECEIVER
LAUNCH
TX
RH HEAT
SONDE UMBILICAL
VCC
Slot Active
Slot 3
Slot 2
Slot 1
SONDE INTERFACE
EtherCOM
EtherCOM


WESTCOTT
STAINLESS STEEL


AFT

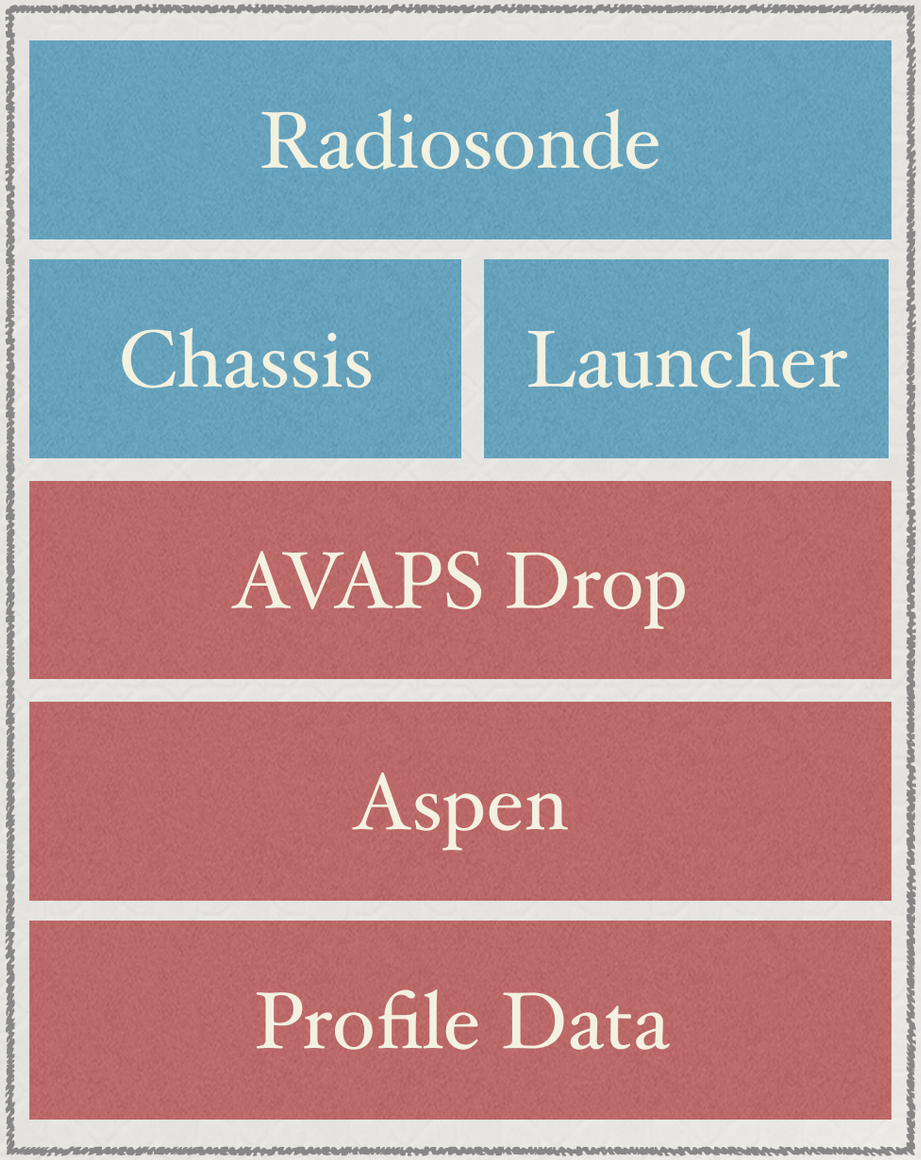



Drop # 4 of 66 :: 2016-02-21-RF03:: B20160221_183116_2 :: G4 :: 101935217.0
Vertical Velocity [m/s]
Pressure (mbar)
Temperature [C], Humidity [%], Wind Speed [m/s]
Temperature
%RH #1
%RH #2
%RH1-%RH2
Wind Speed #1
Wind Speed #2
Vertical Vel. #1
Vertical Vel. #2


ABSOPULSE
T.P.
Vadj
Vout
POWER SUPPLY
SIGNAL STRENGTH
DATA LOCK
15VDC/13A
TELEMETRY RECEIVER
LAUNCH
TX
RH HEAT
SONDE UMBILICAL
SONDE INTERFACE
EtherCOM
EtherCOM

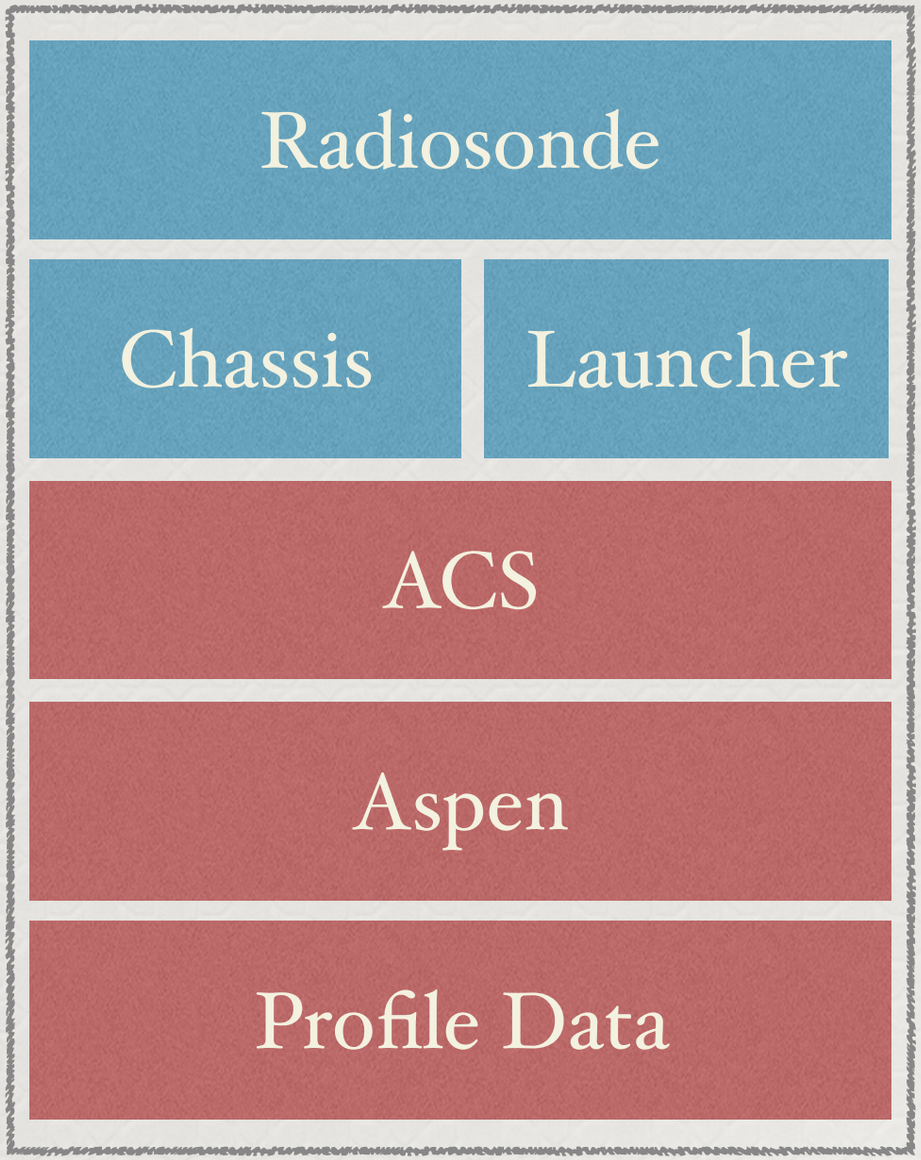
Tear it up.
Build it better.

- *Radiosonde: 2008 & 2017*
- *Chassis: 2008*
- *Launcher: 2010*
- *Aspen: 2010*
- *AVAPS Drop: 1995*



Tear it up. Build it better.

- *Radiosonde: 2008 & 2017*
- *Chassis: 2008*
- *Launcher: 2010*
- *Aspen: 2010*
- *AVAPS Drop: 1995*



Design Criteria

What exactly are we making again?

Revamping

- *Current software stack is doing too much in one place*
  - *Emitting commands to/from equipment*
  - *Data Collection with some data QC*
    - *Various GUI elements: control interface, data graphics*
- *Functionality being rearranged::*
  - *Moving all data QC work into Aspen*
  - *ACSd: performs equipment controlling, data collection*
  - *ACS-fe: handle data displays, controller GUI*

Deficiencies

- *No unit tests or CI/CT: change needs manual effort - (eg, Aircraft Hours)*
- *Inherently single user - multiple ugly hacks for "multi-user" access*
- *Requires arbitrarily precise time with invasive OS hooks*
- *Lots of baked in checks and tunable config*
- *Runs only on x86 Window. Aircraft platforms are migrating to Linux*
- *Logging is missing critical chunks - difficult to troubleshooting.*
- *Sounding data spread all over unstructured data files*

Goals

- *Support multiple users and roles without the need to install anything*
- *Deploy via automated software builds, tests, and releases.*
- *Be as flexible as possible:*
  - *Expose functionality and options as a set of config files*
  - *Break unnecessary state machines and simplify the required ones*
  - *Be able to run on multiple architectures easily (particularly arm & x86)*
  - *Publish and maintain an API for others to utilize*
- *Cease tampering with raw data and use standard containers, such as NetCDF*

ACS-fe :: Front End

- *All users have modern, evergreen web browsers: Any recent builds of Chrome, Firefox, Edge, or Safari should suffice*
- *No attempt for supporting Internet Explorer anything.*
- *Front end should be a standard HTML/CSS/Javascript stack*
- *Use a responsive UI / 'mobile first' layout manager*
- *Create and adhere a HTTP REST API*

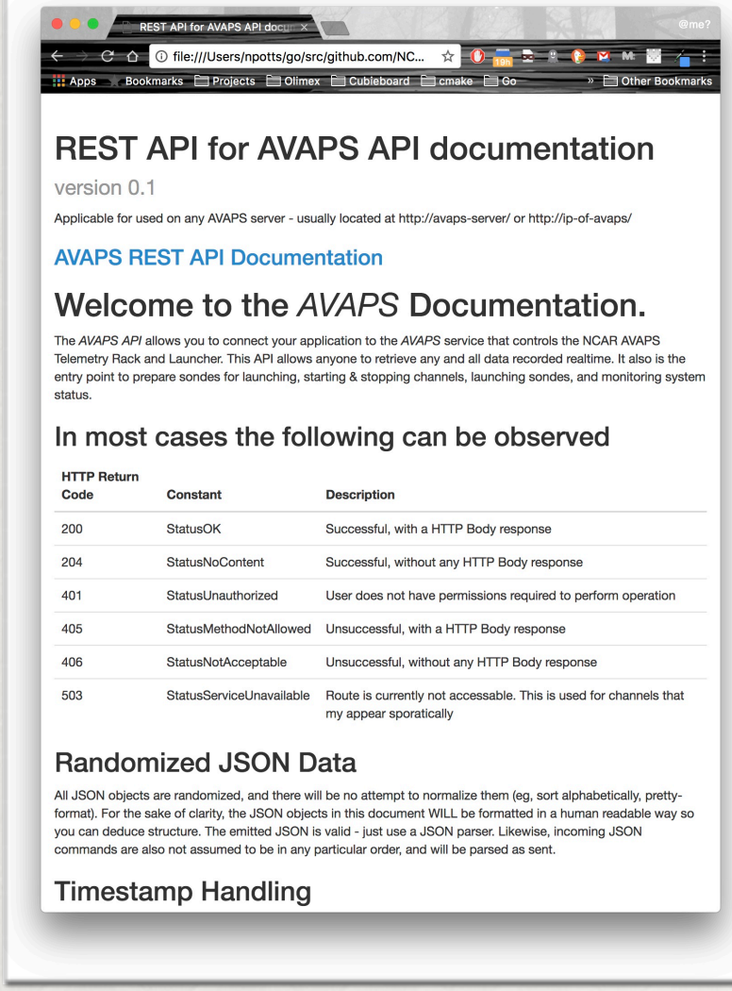
API Design

API Design

Eat your own dog food

API Design

- *Top Down design: developed a "paper-doll" API to design backend services to*
  - *Used RAML 0.8, consider Swagger*
    - *Treated at living document*
- *Mapping system actions to HTTP Verbs (GET, POST, PUT, DELETE)*
  - *http://www.restapitutorial.com/index.html*
  - *Eg retrieve data recording channel's status / recent data / etc: "GET /channel"*
- *Document inputs and outputs, provide examples*



ACSd :: Back End

- *Functions as HTTP server and manages REST API endpoints*
- *Performs command & control functionality; collects data and metadata.*
- *Language selection criteria:*
  - *Readable - code is read more often than written (eg. no Perl)*
  - *Ease of deploying without revealing sources*
    - *Copying a small binary executable preferred*
    - *Docker (containerization) is a fallback*
  - *Function as a HTTP server with some sort of route (mux) handler*
  - *Multi-arch support - minimum of amd64 & arm.*
- *Plug into a Database for immediate data storage, export netCDF at conclusion*

ACSd Language Options (2015)

	Readable	Deploy	HTTP	Arch's
Python	Y	Source	Twisted Native	amd64, *arm*
C/C++	Maybe	Binary	3rd party	*
Rust	Y	Binary w/ libs	*No*	amd64 -arm
Go	Y	Binary	Yes	amd64, arm
RoR	Y	Source	Yes	amd64, -arm
NodeJS	N	Source	Yes	amd64, arm

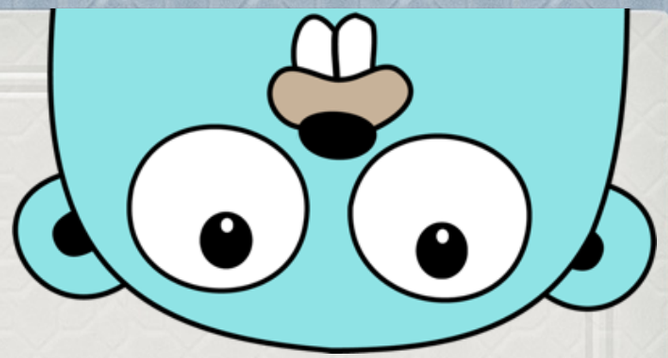
ACSd Language Options (2015)

	Readable	Deploy	HTTP	Arch's
Python	Y	Source	Twisted Native	amd64, *arm*
C/C++	Maybe	Binary	3rd party	*
Rust	Y	Binary w/ libs	*No*	amd64 -arm
Go	Y	Binary	Yes	amd64, arm
RoR	Y	Source	Yes	amd64, -arm
NodeJS	N	Source	Yes	amd64, arm

Ok...
We Chose the Blue Pill

A brief introduction to Go

What is Go



- *Features:*
  - *A compiled, statically typed, procedural language with pointers and garbage collection*
    - *Guaranteed type sizes, functions as first class citizens, variadic functions, & closures*
    - *Built in concurrency via go-routines*
    - *Solid standard library: DocStrings, unit tests, benchmarking, code reflection, ready-for-production HTTP server, cgo (C as Go packages), and dependency vendoring*
- *Feels like a scripting engine - extremely fast statically linked builds*
  - *ACS from **100% source files** (~817 files): **~43secs**. utilizing libs: (~250 file): **1.5s***

Origin and Users

- *Google release Go 1.0 into the wild in 2009.*
- *Used heavily for backend services by household names*

1password	BBC	Canonical	Digital Ocean
Docker	Dropbox	Facebook	Google
IBM	Intel	Lyft	Mozilla
Netflix	SendGrid	Stack Exchange	Uber

- *If you work at UCAR, you are using Go*

- *Integrates OOB with GitHub, BitBucket, Launchpad, IBM DevOps Services and with Bazaar, Git, Mercurial, and Subversion*
- *Forced hierarchy rooted at $GOPATH:*
  - *Binaries : $GOPATH/bin*
  - *Sources : $GOPATH/src*
  - *Compiled Libs: $GOPATH/pkg*

Idiomatic Go

- *Go was designed for large projects with multiple developers.*
- *Go is rather opinionated. Give in - it is sage advice*
  - *Naming - https://blog.golang.org/package-names*
  - *DocString, Interface names, etc - https://golang.org/doc/effective_go.html*
  - *Use gofmt - no need for* astyle *wars*
  - *Generally, 'magic' is heavily frowned on.*
- *Most, if not all, go code you see all follows these conventions*

Learning Curve

- *Go does not support inheritance or subclassing. Instead it uses interfaces*
- *Go's object model of interfaces is a bit odd at first, but is powerful. Given:*

```
type Reader interface {
  Read(p []byte) (n int, err error)
}
```

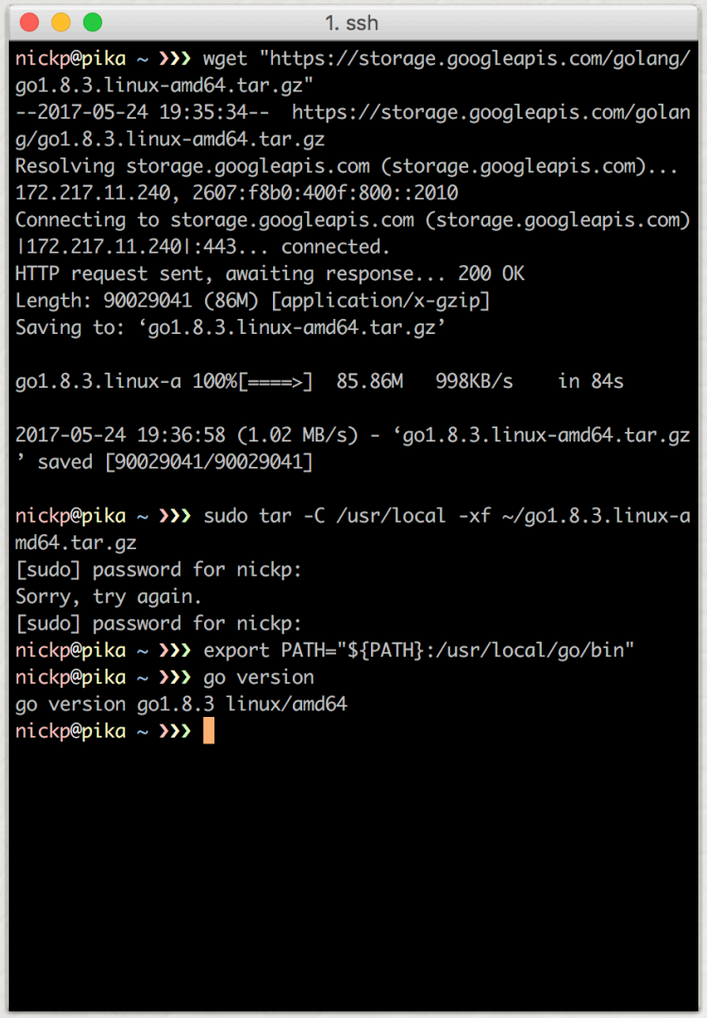
Any type that has the same read signature is a "Reader" and can utilize functions that take Reader arguments.

- *Go does lacks generics. There are libraries that alleviate some of the burden, but in practice I haven't found this to be too painful, but it does break DRY principles.*

Getting Started

- *Download from*
  https://golang.org/dl
- *Install instructions*
  https://golang.org/doc/install
- *Usually little more than:*

```
>tar -C /usr/local  -xf <release>
>export PATH="${PATH}:/usr/local/go/bin"
>go version
```

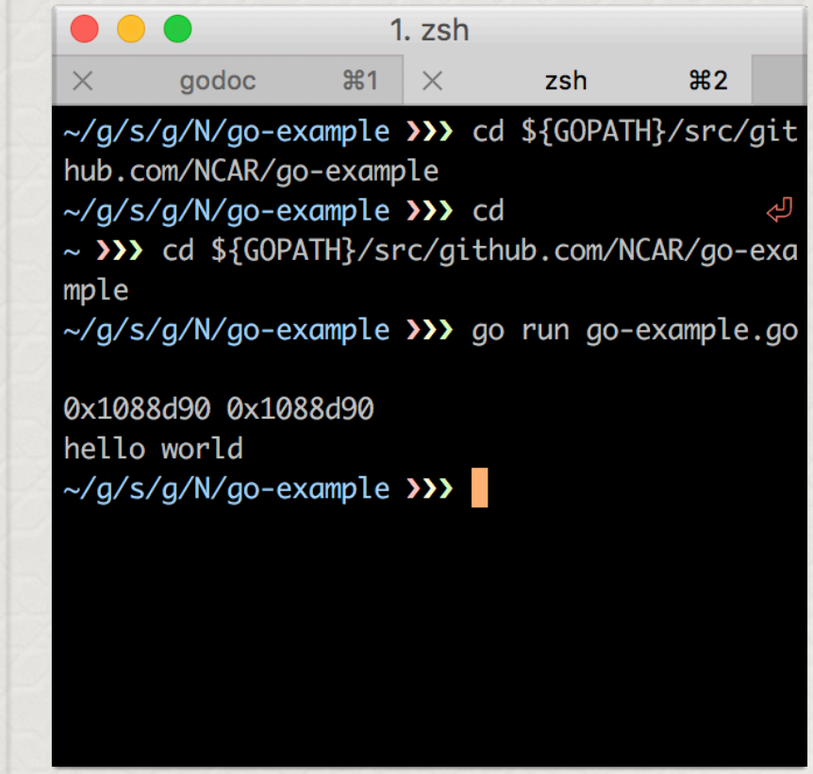


Hello World

```go
package main

import (
    "fmt"
)

func main() {
    f := func(s string) func() string {
        r := func() string {
            return s
        }
        return r
    }
    hello := f("hello")
    world := f("world")
    fmt.Printf("%v %v\n", hello, world)
    fmt.Printf("%s %s\n", hello(), world())
}
```



Packages as Compile Units

```go
/*Package say is an example go package with no real functionality

It covers the basics of:
 - interfaces
 - type syntax
*/
package say

import (
    "fmt"
    sc "strconv" //import strconv package as "sc"
)

/*I typedef for int64 (64 bit int).  Typedef'd parameters can add any sort of functionality*/
type I int64

var q I      //q is of type() Isay defaults to 0
var r *I     //r defaults to a null pointer, which is  nil
var s = I(4) // s is a pointer to an Isay struct of value 5

/*init is called on module init at runtime in order:
- Package wide variables initialized (such as q, r, and s)
- whatever actions init() performs
*/
func init() {
    var val = I(54) // syntax is allowed
    val2 := I(55)   // but this form is better
    q = val2 - val  //setting q to something non-zero

    r = &val //this is both syntactically ok and safe
    if *r != val {
        panic("r is the same as I")
    }
}

/*Sayer is an exported interface declaration. Title case declarations are exported.*/
type Sayer interface {
    Say() string
}

/*usay is a un-exported by way of being a lower case character: non-package member cannot (easily) access.*/
type usay uint

/*Say is a function receiver attached to the definition of usay. u, in this context, is unalterable but accessible for comparison and use. usay is a Sayer via the same signature*/
func (u usay) Say() string {
    switch u {
    case 2, 4, 6:
        fallthrough //case blocks don't 'fallthrough' automatically
    case 8, 10:
        return "Even!"
    case 1, 3, 5, 7, 9:
        return "Odd!"
    case 0: //ordering in case is irrelevant
        return "Zero!!!!"
    default:
        return "Too Big!!"
    }
}
```

Packages as Compile Units

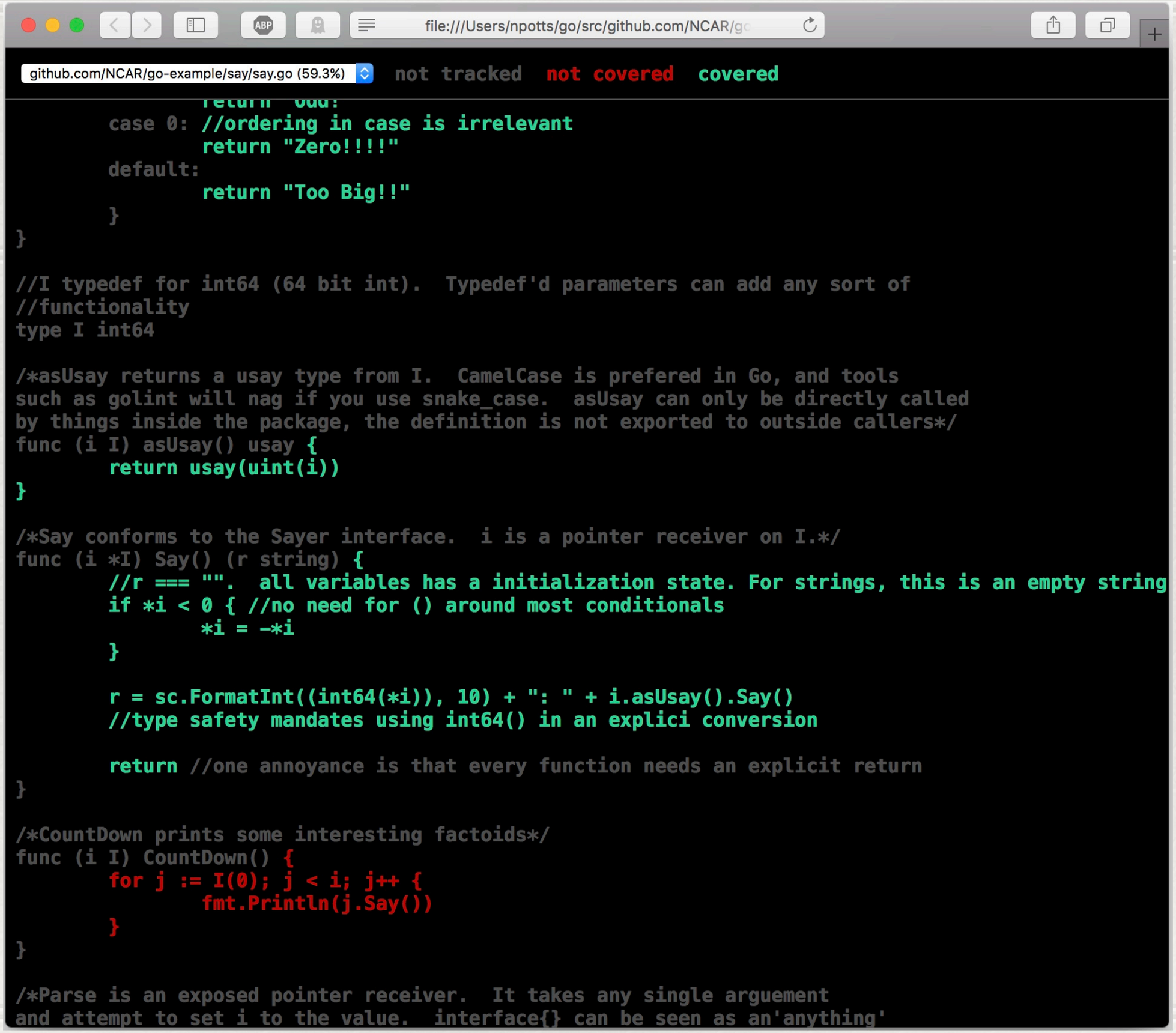
```
65 /*NewI returns a instantiated I from the passed value i*/
66 func NewI(i int64) *I {
67     r := I(i)
68     return &r //safe & legal
69 }
70
71 /*asUsay returns a usay type from I.  CamelCase is preferred
72 and tools such as golint will nag if you use snake_case.
73 asUsay can only be directly called by things inside the pkg,
74 the definition is not exported to outside callers*/
75 func (i I) asUsay() usay {
76     return usay(uint(i))
77 }
78
79 /*Say conforms to the Sayer interface. Say is a pointer
receiver on I.*/
80 func (i *I) Say() (r string) {
81     /*r is equal to "" - all variables have a initialization
82     state. For strings, this is an empty string*/
83     if *i < 0 { //no need for () around most conditionals
84         *i = -*i
85     }
86
87     r = sc.FormatInt((int64(*i)), 10) + ": " +
i.asUsay().Say()
88     //type safety mandates using int64() explicitly
89
90     //one annoyance is that bare functions needs a return
91     return
92 }
93
94 /*CountDown prints some interesting factoids*/
95 func (i I) CountDown() {
96     for j := I(0); j < i; j++ {
97         fmt.Println(j.Say())
98     }
99 }
100
101 /*Parse is an exposed pointer receiver.  It takes any single
102 argument and attempt to set i to the value.  interface{} can
103 be seen as 'anything' variable, but is a bit more nuanced.*/
104 func (i *I) CanConvert(iface interface{}) bool {
105     //type switch:  uses introspection internally
106     switch v := iface.(type) {
107     //various forms of ints. A int is always 32 bits.
108     case uint8, int8, int16, uint16, int32, uint32, int,
uint, int64, uint64:
109         return true
110     case float32, float64:
111         return true
112     case []byte, string:
113         return false
114     case usay, I:
115         return true
116     case Sayer: //can check for a sayer interface
117         return true
118     case func() int:
119         //functions are first class citizens, and can be
treated as POD
120         return true
121     default:
122         fmt.Println(`This wont compile unless v is used.
123 backtick quotes span multiple lines`, v)
124     }
125     return false
126 }
```

Unit Tests

```
 1 /*Package say is an example go package with no real
functionality
 2
 3 It covers the basics of:
 4  - interfaces
 5  - type syntax
 6 */
 7 package say
 8
 9 import (
10     // "fmt" //unused imports are compile failures
11     "testing"
12 )
13
14 func TestUsay_Say(t *testing.T) {
15     //A map (of keys of type usay, of values string)
is a hash Table
16     tests := map[usay]string{
17         0:  "Zero!!!!",
18         1:  "Odd!",
19         2:  "Even!",
20         3:  "Odd!",
21         4:  "Even!",
22         5:  "Odd!",
23         6:  "Even!",
24         7:  "Odd!",
25         8:  "Even!",
26         9:  "Odd!",
27         10: “Even!",
28         11: "Too Big!!”,
29     }
30
31     //this rendition of for, with the range keyword,
which iterates through
32     //all the keys in the above map in random order
33     for key, value := range tests {
34         if got := key.Say(); got != value {
35             t.Errorf("Say() on %d failed:  Got %q, not
%q as expected", key, got, value)
36         }
37     }
38 }
39
40 func TestI_Say(t *testing.T) {
41     //A map (of keys of type usay, of values string)
is a hash Table
42     tests := map[I]string{
43         0:  "0: Zero!!!!",
44         1:  "1: Odd!",
45         -1: "1: Odd!",
46         2:  "2: Even!",
47     }
48
49     //this rendition of for, with the range keywork,
which iterates through
50     // all the keys in the above map in random order
51     for key, value := range tests {
52         if got := key.Say(); got != value {
53             t.Errorf("Say() on %d failed:  Got %q, not
%q as expected", key, got, value)
54         }
55     }
56 }
```

Unit Tests

```
> cd ${GOPATH}/github.com/NCAR/go-example/say
> go test -coverprofile=cover.out
> go tool cover -html=cover.out -o cover.html
> open cover.html
```


file:///Users/npotts/go/src/github.com/NCAR/g
github.com/NCAR/go-example/say/say.go (59.3%)
not tracked
not covered
covered
return "Odd!"
case 0: //ordering in case is irrelevant
return "Zero!!!!"
default:
return "Too Big!!"
}
}
//I typedef for int64 (64 bit int). Typedef'd parameters can add any sort of
//functionality
type I int64
/*asUsay returns a usay type from I. CamelCase is prefered in Go, and tools
such as golint will nag if you use snake_case. asUsay can only be directly called
by things inside the package, the definition is not exported to outside callers*/
func (i I) asUsay() usay {
return usay(uint(i))
}
/*Say conforms to the Sayer interface. i is a pointer receiver on I.*/
func (i *I) Say() (r string) {
//r === "". all variables has a initialization state. For strings, this is an empty string
if *i < 0 { //no need for () around most conditionals
*i = -*i
}
r = sc.FormatInt((int64(*i)), 10) + ": " + i.asUsay().Say()
//type safety mandates using int64() in an explici conversion
return //one annoyance is that every function needs an explicit return
}
/*CountDown prints some interesting factoids*/
func (i I) CountDown() {
for j := I(0); j < i; j++ {
fmt.Println(j.Say())
}
}
/*Parse is an exposed pointer receiver. It takes any single arguement
and attempt to set i to the value. interface{} can be seen as an'anything'

Concurrency

```
21 type data struct {
22     When time.Time
23     s    say.Sayer //remember, say.Sayer is an
interface
24 }
25
.....
35 /*Basic shows a very basic usage of channels*/
36 func Basic(writers, readers int) {
37     // the make keywork creates the channel on the
heap of type "chan data"
38     chanData := make(chan data)
39
40     writer := func(id int) {
41         //wait's type are determined by the return
value of randDuration(),
42         // and the second returned argument is
being discarded via the _
43         // keyword
44         wait, _ := randDuration()
45
46         // time.After() returns a (single fire)
channel , which
47         // this will block on until it can be read
(and discarded)
48         // via the <- operator
49         <-time.After(wait)
50
51         //closure here in accessing chanData.
This blocks
52         // until someone does a synchronous read
from chanData
53         chanData <- data{When: time.Now(), s:
say.NewI(int64(id))}
54     }
55
56     now := time.Now()
57
58     for i := 0; i < writers; i++ {
59         //the 'go' keyword invokes calling the
passed function on a separate
60         // goroutine.  In this case, we are
attempting to write data objects
61         // to chanData 'writers' times
62         go writer(i)
63     }
64
65     for i := 0; i < readers; i++ {
66         // d's type is determined from the
definition of chanData
67         // this time, we are using the <- operator
to synchronously read
68         // from one of the above goroutines.
Which is not
69         d := <-chanData
70         fmt.Printf("%s: %v\n", time.Since(d.When),
d.s.Say())
71     }
72     fmt.Println("Operation took ",
time.Since(now))
73 }
```

```
14.622697ms: 4812: Too Big!!
14.612568ms: 4853: Too Big!!
14.609019ms: 4852: Too Big!!
14.522856ms: 4678: Too Big!!
Operation took  28.334608ms
PASS
ok      github.com/NCAR/go-example/busy 0.036s
~/g/s/g/N/g/busy ››› go test -run Basic
```

```
21 type data struct {
22     When time.Time
23     s    say.Sayer //remember, say.Sayer is an
interface
24 }
25
.....
35 /*Basic shows a very basic usage of channels*/
36 func Basic(writers, readers int) {
37     // the make keywork creates the channel on the
heap of type "chan data"
38     chanData := make(chan data)
39
40     writer := func(id int) {
41         //wait's type are determined by the return
value of randDuration(),
42         // and the second returned argument is
being discarded via the _
43         // keyword
44         wait, _ := randDuration()
45
46         // time.After() returns a (single fire)
channel , which
47         // this will block on until it can be read
(and discarded)
48         // via the <- operator
49         <-time.After(wait)
50
51         //closure here in accessing chanData.
This blocks
52         // until someone does a synchronous read
from chanData
53         chanData <- data{When: time.Now(), s:
say.NewI(int64(id))}
54     }
55
56     now := time.Now()
57
58     for i := 0; i < writers; i++ {
59         //the 'go' keyword invokes calling the
passed function on a separate
60         // goroutine.  In this case, we are
attempting to write data objects
61         // to chanData 'writers' times
62         go writer(i)
63     }
64
65     for i := 0; i < readers; i++ {
66         // d's type is determined from the
definition of chanData
67         // this time, we are using the <- operator
to synchronously read
68         // from one of the above goroutines.
Which is not
69         d := <-chanData
70         fmt.Printf("%s: %v\n", time.Since(d.When),
d.s.Say())
71     }
72     fmt.Println("Operation took ",
time.Since(now))
73 }
```

Data Races

```
75 /*MapRace purposefully forces race conditions, to demonstrate the
76 functionality of the 'go test -race' race detector*/
77 func MapRace(N int) {
78     //wait group, so we don't exit MapRace while data is being modified
79     wg := sync.WaitGroup{}
80     wg.Add(N)
81
82     //data is an empty has map of int->timestamps
83     //maps are not thread safe by definition
84     data := map[int]time.Time{}
85
86     f := func() {
87         d, _ := randDuration()
88         <-time.After(d) //random delay
89         //Pick a random int from 0:9 and attach the current timestamp
90         data[rand.Intn(10)] = time.Now()
91         wg.Done() //tells the wait group this is done
92     }
93
94     for i := 0; i < N; i++ {
95         go f()
96     }
97
98     wg.Wait() //waits for all go routines to return
99 }
```

Data Races

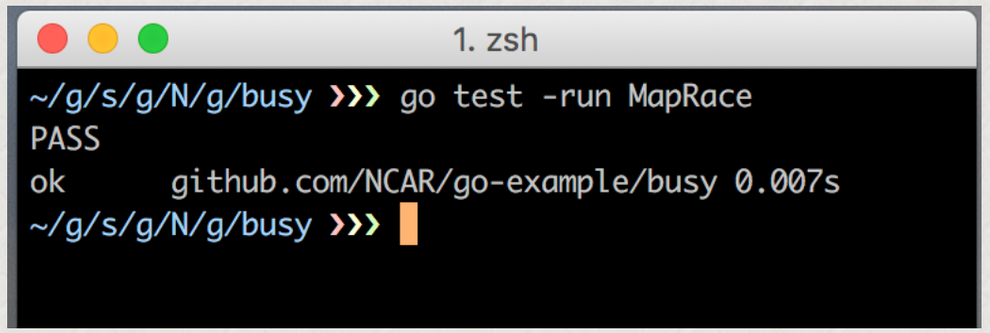
```go
/*MapRace purposefully forces race conditions, to demonstrate the
functionality of the 'go test -race' race detector*/
func MapRace(N int) {
    //wait group, so we don't exit MapRace while data is being modified
    wg := sync.WaitGroup{}
    wg.Add(N)

    //data is an empty has map of int->timestamps
    //maps are not thread safe by definition
    data := map[int]time.Time{}

    f := func() {
        d, _ := randDuration()
        <-time.After(d) //random delay
        //Pick a random int from 0:9 and attach the current timestamp
        data[rand.Intn(10)] = time.Now()
        wg.Done() //tells the wait group this is done
    }

    for i := 0; i < N; i++ {
        go f()
    }

    wg.Wait() //waits for all go routines to return
}
```



```
75 /*Ma
76 func
77 func
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99 }
```

1. zsh

```
~/g/s/g/N/g/busy ››› go test -run MapRace
PASS
ok      github.com/NCAR/go-example/busy 0.008s
~/g/s/g/N/g/busy ››› clear && go test -run MapRace
fatal error: concurrent map writes

goroutine 6 [running]:
runtime.throw(0x113abf0, 0x15)
        /usr/local/go/src/runtime/panic.go:596 +0x95 fp=0xc42002e6e0 sp=0xc42002e6c0
runtime.mapassign(0x11136e0, 0xc420018a20, 0xc42002e7b0, 0x0)
        /usr/local/go/src/runtime/hashmap.go:499 +0x667 fp=0xc42002e780 sp=0xc42002e6e0
github.com/NCAR/go-example/busy.MapRace.func1()
        /Users/npotts/go/src/github.com/NCAR/go-example/busy/busy.go:90 +0xc3 fp=0xc42002e7e0 sp=0xc42002e780
runtime.goexit()
        /usr/local/go/src/runtime/asm_amd64.s:2197 +0x1 fp=0xc42002e7e8 sp=0xc42002e7e0
created by github.com/NCAR/go-example/busy.MapRace
        /Users/npotts/go/src/github.com/NCAR/go-example/busy/busy.go:95 +0xfd

goroutine 1 [chan receive]:
testing.(*T).Run(0xc420070750, 0x1138977, 0xb, 0x1140478, 0x1053301)
        /usr/local/go/src/testing/testing.go:698 +0x2f4
testing.runTests.func1(0xc420070750)
        /usr/local/go/src/testing/testing.go:882 +0x67
testing.tRunner(0xc420070750, 0xc420041de0)
        /usr/local/go/src/testing/testing.go:657 +0x96
testing.runTests(0xc42000ac60, 0x11ce480, 0x2, 0x2, 0x1140a60)
        /usr/local/go/src/testing/testing.go:888 +0x2c1
testing.(*M).Run(0xc420041f20, 0xc420041f20)
        /usr/local/go/src/testing/testing.go:822 +0xfc
main.main()
        github.com/NCAR/go-example/busy/_test/_testmain.go:44 +0xf7

goroutine 5 [semacquire]:
sync.runtime_Semacquire(0xc420014f4c)
        /usr/local/go/src/runtime/sema.go:47 +0x34
sync.(*WaitGroup).Wait(0xc420014f40)
        /usr/local/go/src/sync/waitgroup.go:131 +0x7a
github.com/NCAR/go-example/busy.MapRace(0x5)
        /Users/npotts/go/src/github.com/NCAR/go-example/busy/busy.go:98 +0x128
github.com/NCAR/go-example/busy.TestMapRace(0xc420070820)
        /Users/npotts/go/src/github.com/NCAR/go-example/busy/busy_test.go:14 +0x2a
testing.tRunner(0xc420070820, 0x1140478)
        /usr/local/go/src/testing/testing.go:657 +0x96
created by testing.(*T).Run
```

Go's Race Detector

go has a built in race detector:

go test -race

```
Goroutine 10 (finished) created at:
  github.com/NCAR/go-example/busy.MapRace()
      /Users/npotts/go/src/github.com/NCAR/go-example/busy/busy.go:95 +0x16b
  github.com/NCAR/go-example/busy.TestMapRace()
      /Users/npotts/go/src/github.com/NCAR/go-example/busy/busy_test.go:14 +0x37
  testing.tRunner()
      /usr/local/go/src/testing/testing.go:657 +0x107
==================
==================
WARNING: DATA RACE
Write at 0x00c42007c9f0 by goroutine 8:
  runtime.mapassign()
      /usr/local/go/src/runtime/hashmap.go:485 +0x0
  github.com/NCAR/go-example/busy.MapRace.func1()
      /Users/npotts/go/src/github.com/NCAR/go-example/busy/busy.go:90 +0xda

Previous write at 0x00c42007c9f0 by goroutine 10:
  runtime.mapassign()
      /usr/local/go/src/runtime/hashmap.go:485 +0x0
  github.com/NCAR/go-example/busy.MapRace.func1()
      /Users/npotts/go/src/github.com/NCAR/go-example/busy/busy.go:90 +0xda

Goroutine 8 (running) created at:
  github.com/NCAR/go-example/busy.MapRace()
      /Users/npotts/go/src/github.com/NCAR/go-example/busy/busy.go:95 +0x16b
  github.com/NCAR/go-example/busy.TestMapRace()
      /Users/npotts/go/src/github.com/NCAR/go-example/busy/busy_test.go:14 +0x37
  testing.tRunner()
      /usr/local/go/src/testing/testing.go:657 +0x107

Goroutine 10 (finished) created at:
  github.com/NCAR/go-example/busy.MapRace()
      /Users/npotts/go/src/github.com/NCAR/go-example/busy/busy.go:95 +0x16b
  github.com/NCAR/go-example/busy.TestMapRace()
      /Users/npotts/go/src/github.com/NCAR/go-example/busy/busy_test.go:14 +0x37
  testing.tRunner()
      /usr/local/go/src/testing/testing.go:657 +0x107
==================
--- FAIL: TestMapRace (0.00s)
        testing.go:610: race detected during execution of test
FAIL
exit status 1
FAIL    github.com/NCAR/go-example/busy 0.014s
~/g/s/g/N/g/busy ❯❯❯
```

Cross-OS Builds

- *Go emits statically linked binaries for $GOOS /$GOARCH pair.*
- *Building for a different arch and os is dead simple:*

```
GOOS="linux" GOARCH="amd64" go build
```

Cross-OS Builds

- *Go emits statically linked binaries for $GOOS /$GOARCH pair.*
- *Building for a different arch and os is dead simple:*

```
GOOS="linux" GOARCH="amd64" go build
```

- In fact, `go build` is just a special case where $GOOS and $GOARCH are set to the host compiler's $GOOS and $GOARCH - functionally always cross compiling

On a RHEL box:

1. npotts@tikal: ~/code/go/src/github.com/NCAR/ACSd (ssh)

tikal: ...om/NCAR/AC... ⌘1 | ssh ⌘2

```
npotts@tikal ~/c/g/s/g/N/ACSd ››› uname -a
Linux tikal.eol.ucar.edu 3.10.0-514.16.1.el7.x86_64 #1 SMP Wed Apr 12 15:04:24 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
npotts@tikal ~/c/g/s/g/N/ACSd ››› shasum packager
8e7aa0f932089252fd24b32382b9c1e33eb04fd6  packager
npotts@tikal ~/c/g/s/g/N/ACSd ››› ./packager --help
Usage of ./packager:
  -all
        Same as --vendored --embed --pkgs and installs various binaries into the "build" folder
  -clean
```

Copy the Binary to an Ubuntu box:

1. ssh

tikal: ...om/NCAR/... ⌘1 | ssh ⌘2

```
nickp@avaps-dev ~/c/g/s/g/N/ACSd ››› uname -a
Linux avaps-dev.eol.ucar.edu 4.4.0-77-generic #98-Ubuntu SMP Wed Apr 26 08:34:02 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
nickp@avaps-dev ~/c/g/s/g/N/ACSd ››› sha1sum packager
8e7aa0f932089252fd24b32382b9c1e33eb04fd6  packager
nickp@avaps-dev ~/c/g/s/g/N/ACSd ››› ./packager --help
Usage of ./packager:
  -all
        Same as --vendored --embed --pkgs and installs various binaries into the "build" folder
  -clean
```

On a Ubuntu box:

1. ssh

tikal: ...hub.com/NCA... ⌘1 | ssh ⌘2

```
#98-Ubuntu SMP Wed Apr 26 08:34:02 UTC 2017
nickp@avaps-dev ~/c/g/s/g/N/ACSd ››› GOOS="darwin" GOARCH="amd64" go build packager.go
nickp@avaps-dev ~/c/g/s/g/N/ACSd ››› ./packager
zsh: exec format error: ./packager
nickp@avaps-dev ~/c/g/s/g/N/ACSd ››› sha1sum packager
b07abebbc69d6d21352f89c375859d19a68da4f2  packager
nickp@avaps-dev ~/c/g/s/g/N/ACSd ›››
```

On a macOS box:

2. zsh

```
~ ››› uname -s
Darwin
~ ››› shasum packager
b07abebbc69d6d21352f89c375859d19a68da4f2  packager
~ ››› ./packager
This is the AVAPS Build tool.  No more, no less

Moving to ${GOPATH}/src/github.com/NCAR/ACSd
Invalid usage.  See --help
~ ›››
```

Documentation

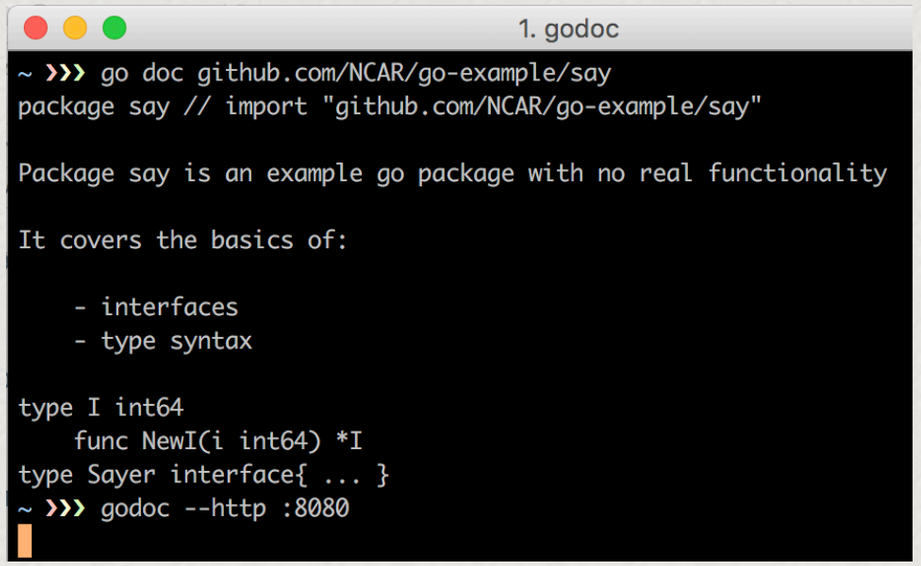
Assuming you follow conventions:

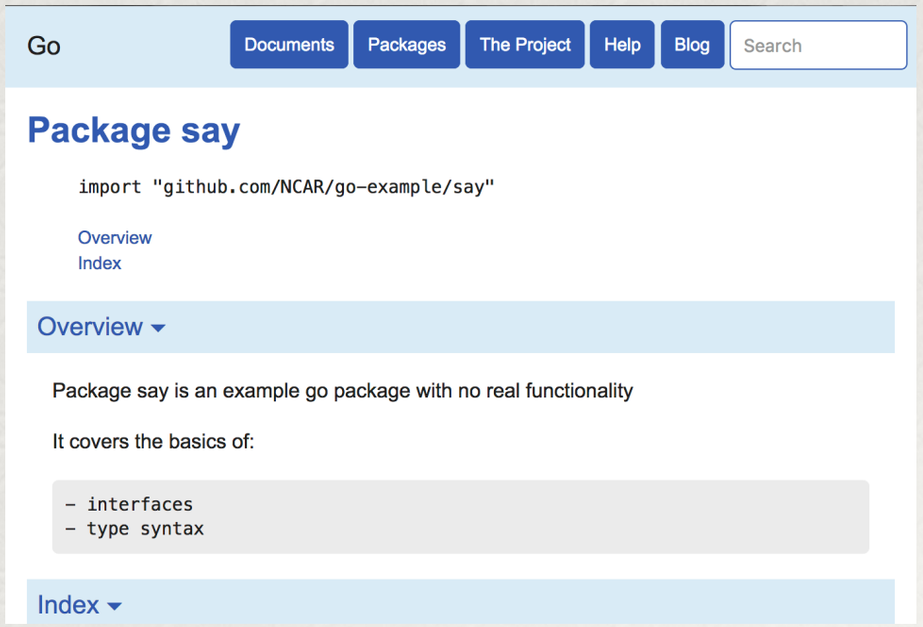
- *Go tool package docs*

  `go doc github.com/NCAR/go-example/say`

- *Godoc tool:*

  ```
  go get -v golang.org/x/tools/cmd/godoc
  godoc -http :8080
  open http://localhost:8080/pkg/
  github.com/NCAR/go-example/say/
  ```





Lessons Learned

...or what not to do

- *Use the provided tooling. They are there to help*
  - *Use* `go test -race`. *Google does. You should too.*
  - `go vet` *detect potential compile and runtime issues*
  - `oracle` *helps navigate source trees*
    - *Most editors tie into the above tools*
- *Close HTTP handlers - They default to remaining open, and you can quickly chew up 65536 sockets*
- *Logging all HTTP traffic, though nice, its a bit excessive and slow*

Resources

Jumpstart your Designs

- *API Design*
  - *http://www.restapitutorial.com/index.html*
  - *https://annevankesteren.nl/2007/10/http-methods*
  - *https://raml.org & http://swagger.io*
- *Go Intro*
  - *https://golang.org/dl & https://golang.org/doc/install*
  - *https://github.com/NCAR/go-example*
  - *https://awesome-go.com*
  - *https://blog.golang.org/race-detector*

Confused or Excited?

(Questions?)