strikr

unsafe, cgo and go plugins

saifi@acm.org

In the world of 'go'

strikr

- Safety guarantees
  - Static checks
    - Type checks
    - Strict rules for type conversions
  - Dynamic checks
    - Out-of-array bounds
    - Nil pointer references

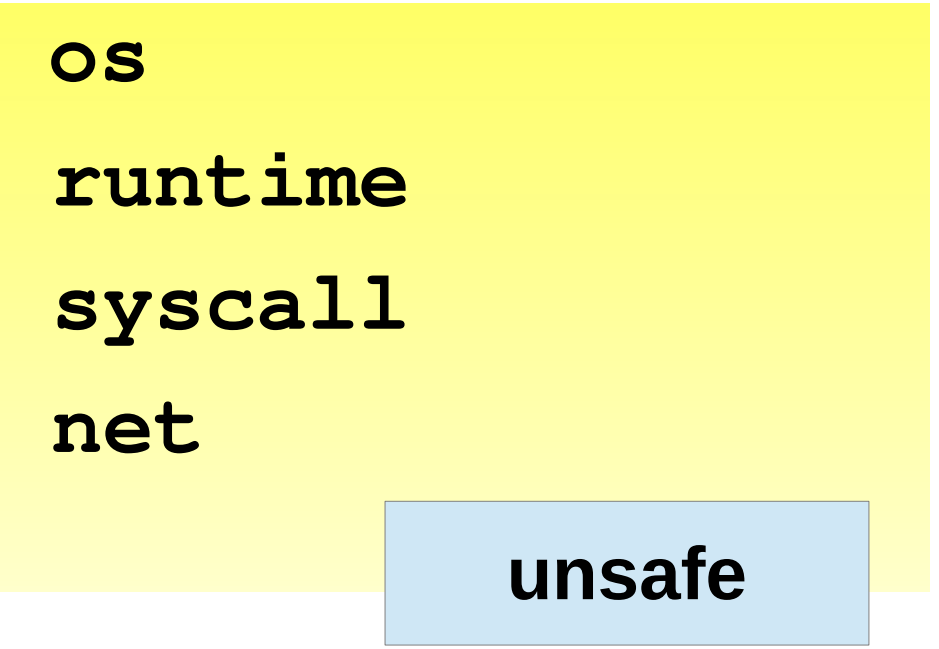
In the world of 'go' ...

strikr

- Implementation details are not accessible
  - Discover layout of struct
  - Pointer identifies a variable without revealing its address.
  - Identity of the OS thread on which the current stackless function (g*) is running
  - Go scheduler moves stackless function (g*) from one thread to another.
  - Address changes and pointer updates as garbage collector moves variables

Side stepping the safety

- **unsafe**
- Expose details of Go memory layout
- Looks like a regular package
- Import “unsafe”
- *Actually* implemented by the compiler



unsafe

strikr

`src/unsafe/unsafe.go`

```
type ArbitraryType int
type Pointer *ArbitraryType

func Sizeof   (x ArbitraryType) uintptr
func Offsetof (x ArbitraryType) uintptr
func AlignOf  (x ArbitraryType) uintptr
```

`go vet` may help but can't depend on it.

unsafe pointer manipulation

```
func get_address (f float64) uint64 {

    pT := unsafe.Pointer (&f)

    p  := (*uint64)(pT)

    *p = 7.0

    return *p
}
```

```
func main () {
    num := 1.0

    fmt.Printf ("%f \n", num)
    fmt.Printf ("%#016x \n", get_address (num))
    fmt.Printf ("%f \n", num)
}
```

Code organization 101

strikr

- As functionality grows, functions are categorized together
- Unit of organization is then
  - Static library
  - Dynamic library
- Calling convention defined to support linkage considerations
  - Who cleans up the stack
  - What happens to name mangling

A C function calling a C function from a library

file_libname.c

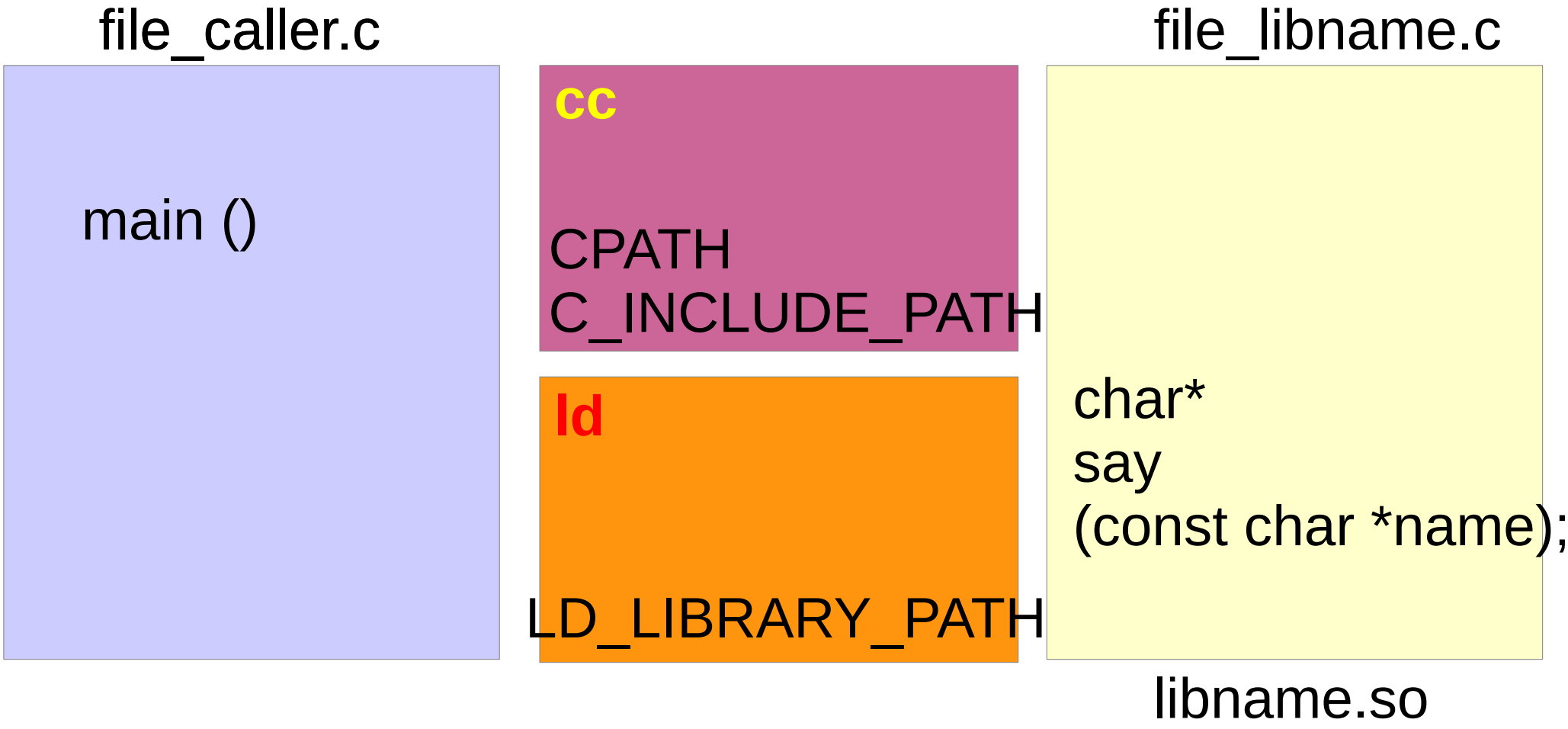
```
char*
say
(const char *name);
```

Code gen option
-f**PIC**

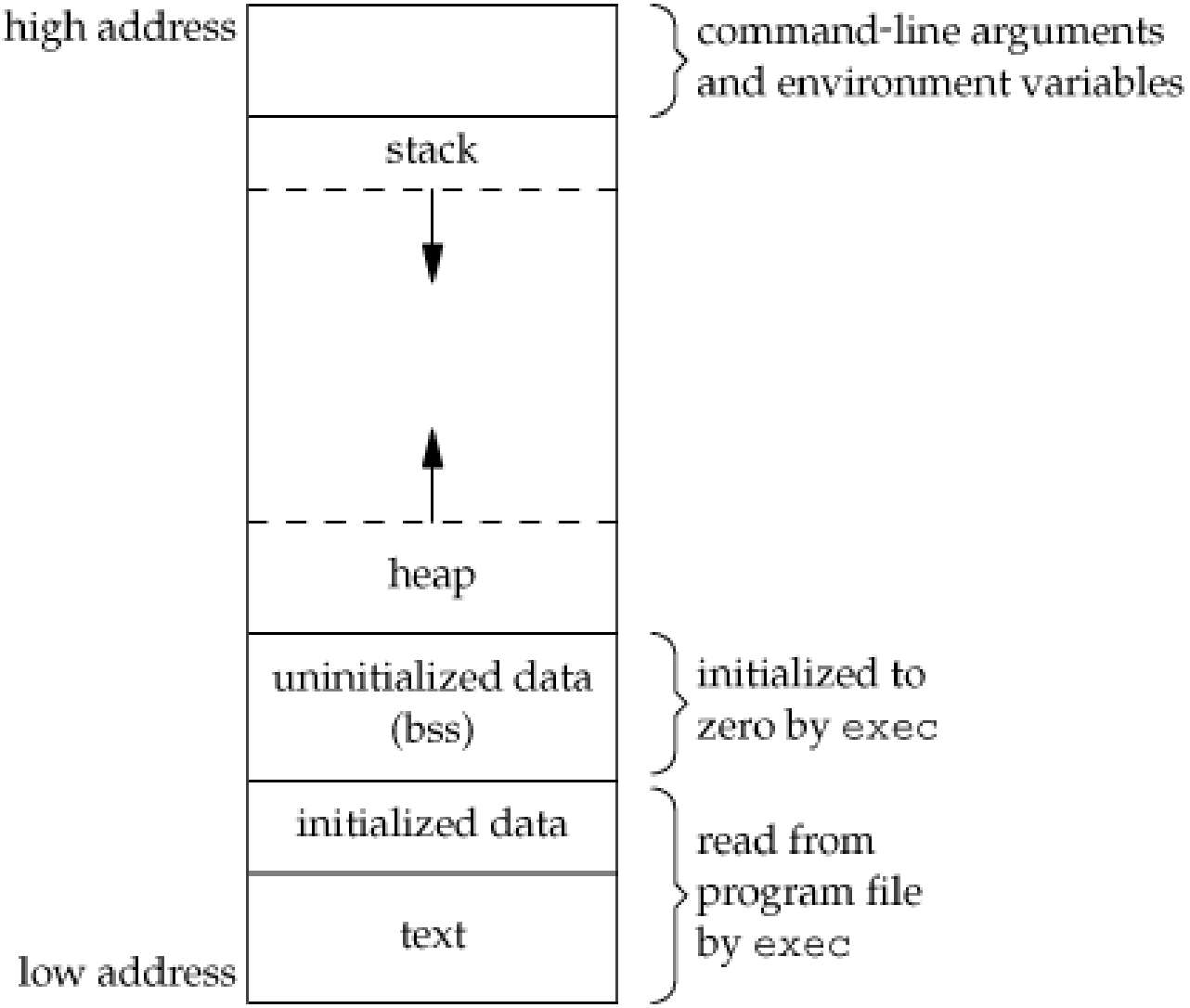
linker option
-shared

libname.so

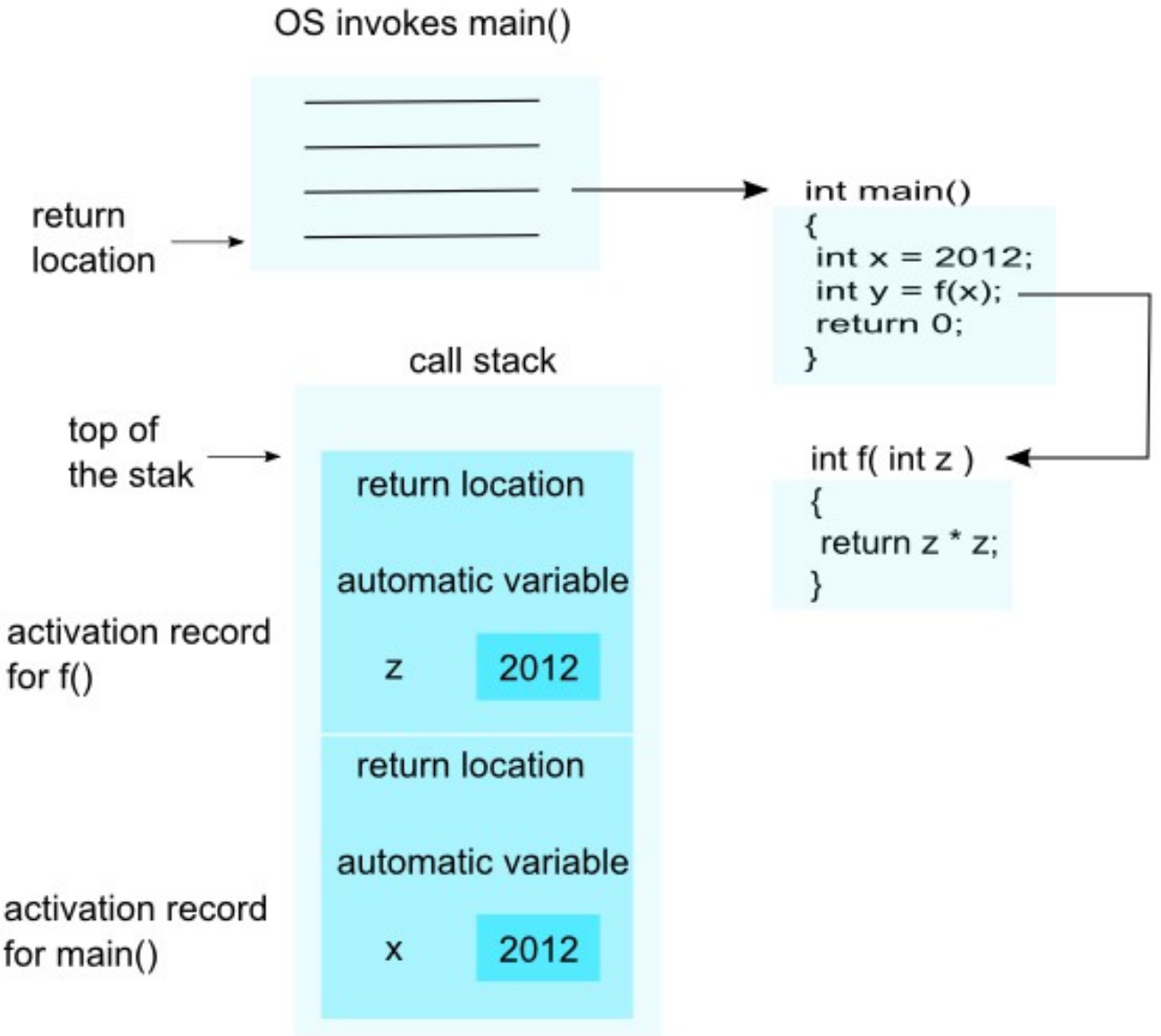
A C function calling a C function from a library



Memory layout of C program



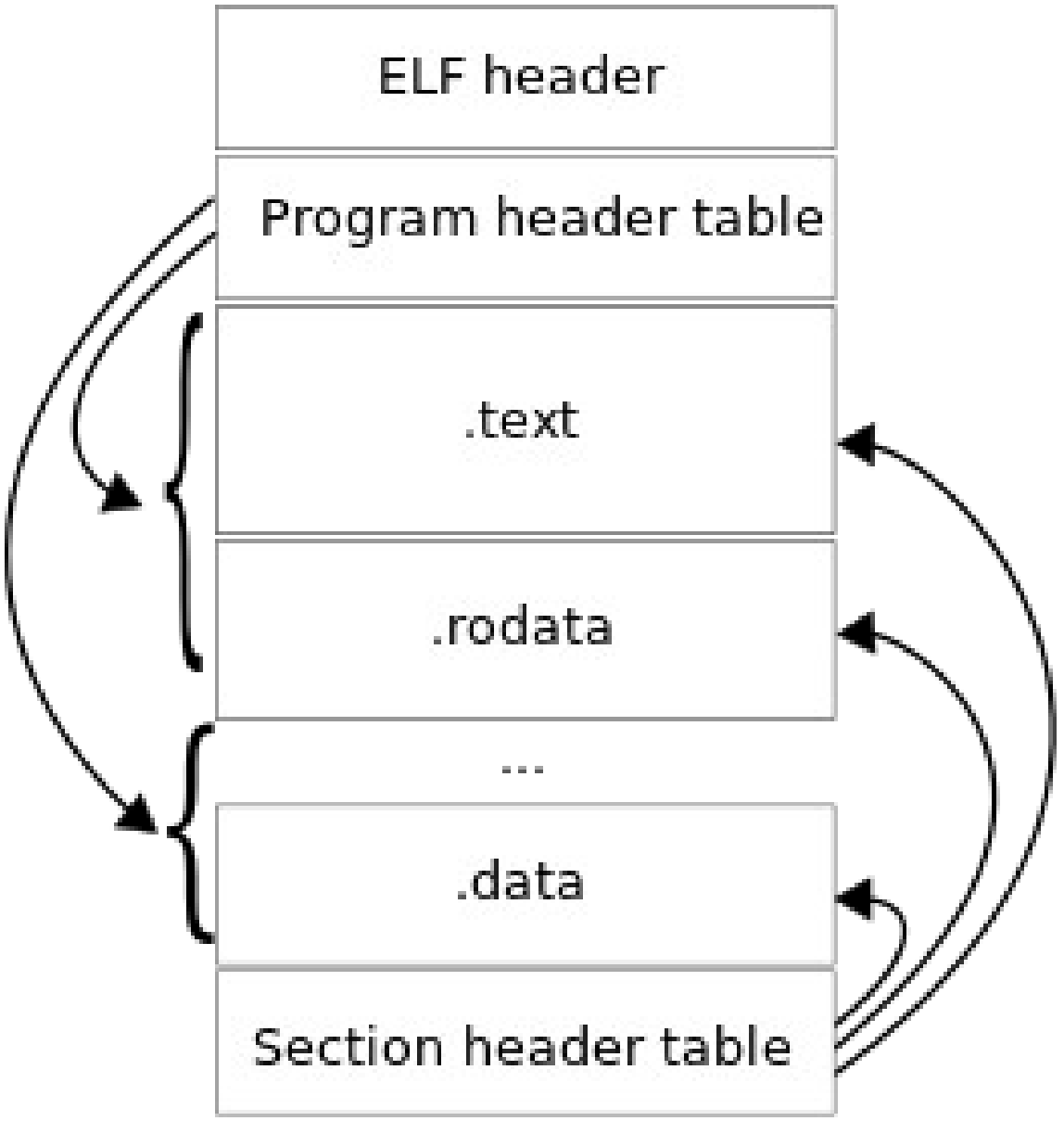
A C function calling a C function



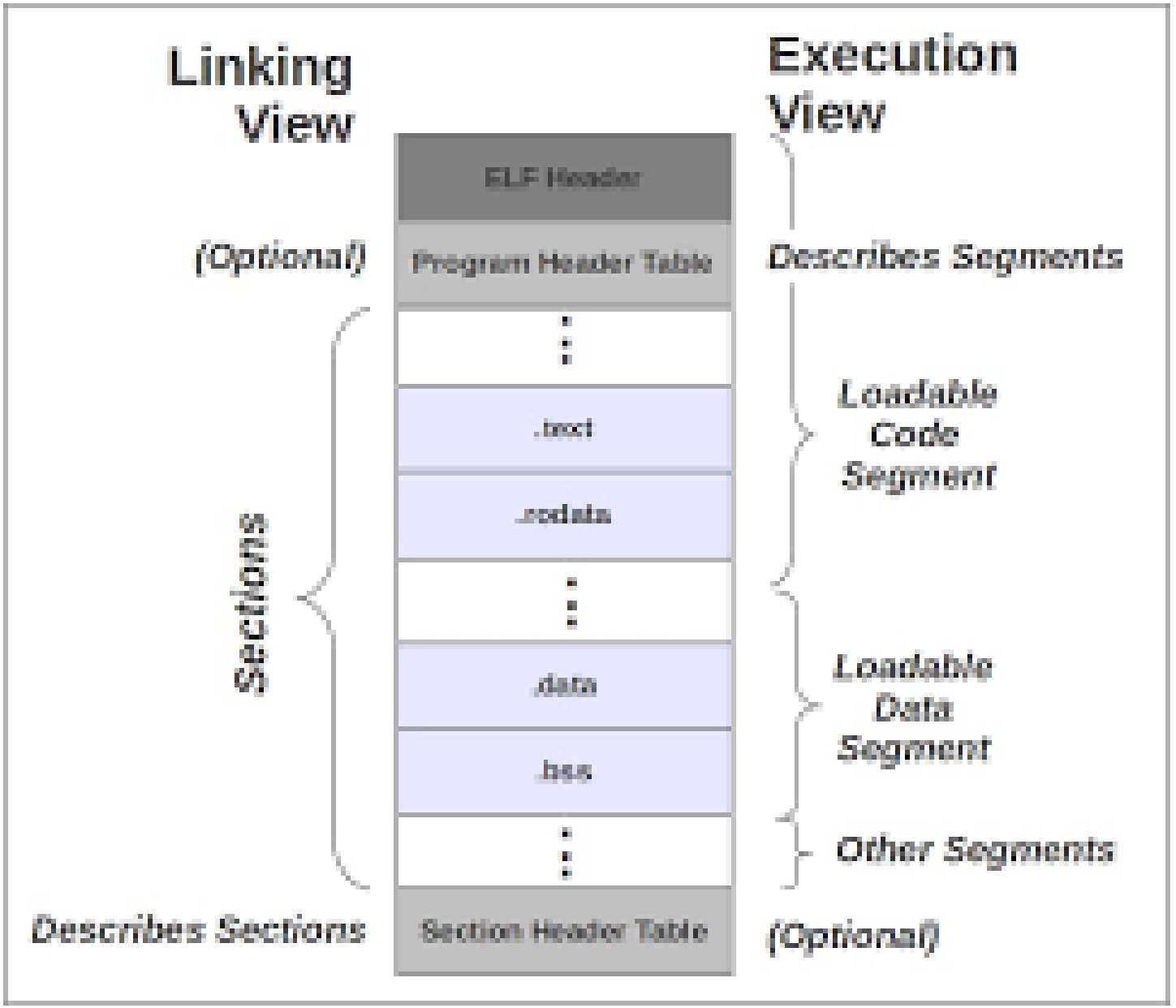
Code organization 202

- Binary generated code is arranged in ELF format
- **E**xecutable and **L**inkable **F**ormat
- Expressed in terms of sections
  - .text
  - .data
  - .rodata
  - .bss
- DWARF for debugging

ELF



Two views



Code organization 303

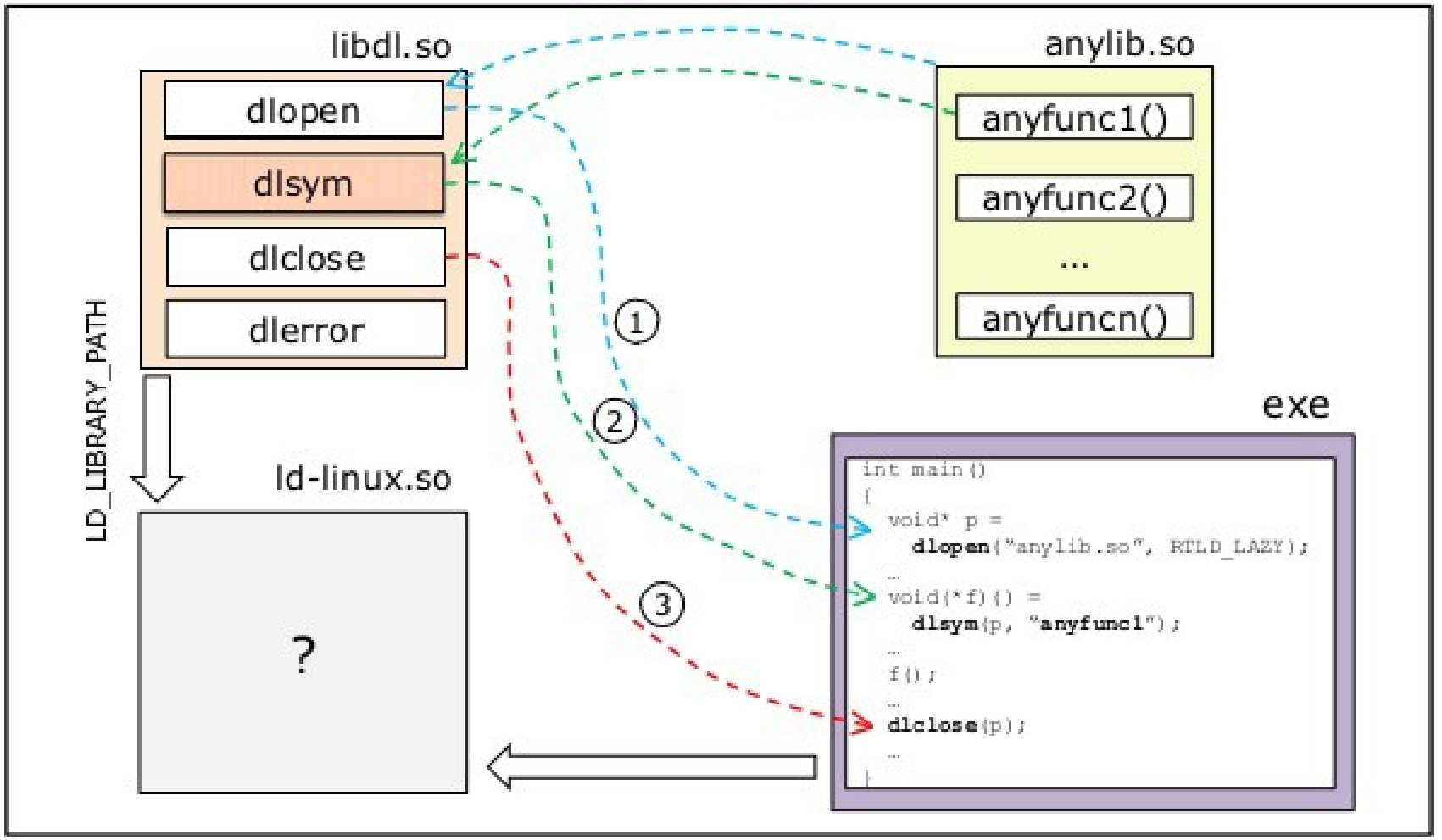
- Dynamically loaded libraries
  - Loaded on demand
  - Used to implement plugins, modules
  - On Linux built as standard object modules
- Linkage
  - extern “**C**”
  - no name mangling

Code organization 303

- Infrastructure to work with sections and symbols
  - `#include <dlfcn.h>`
  - `/usr/include/dlfcn.h`
  - Same as in Solaris
- API
  - `dlopen()`
  - `dlsym()`
  - `dlerror()`
  - `dlclose()`

Infrastructure to work with ELF

Dynamic Loading



Shared object API

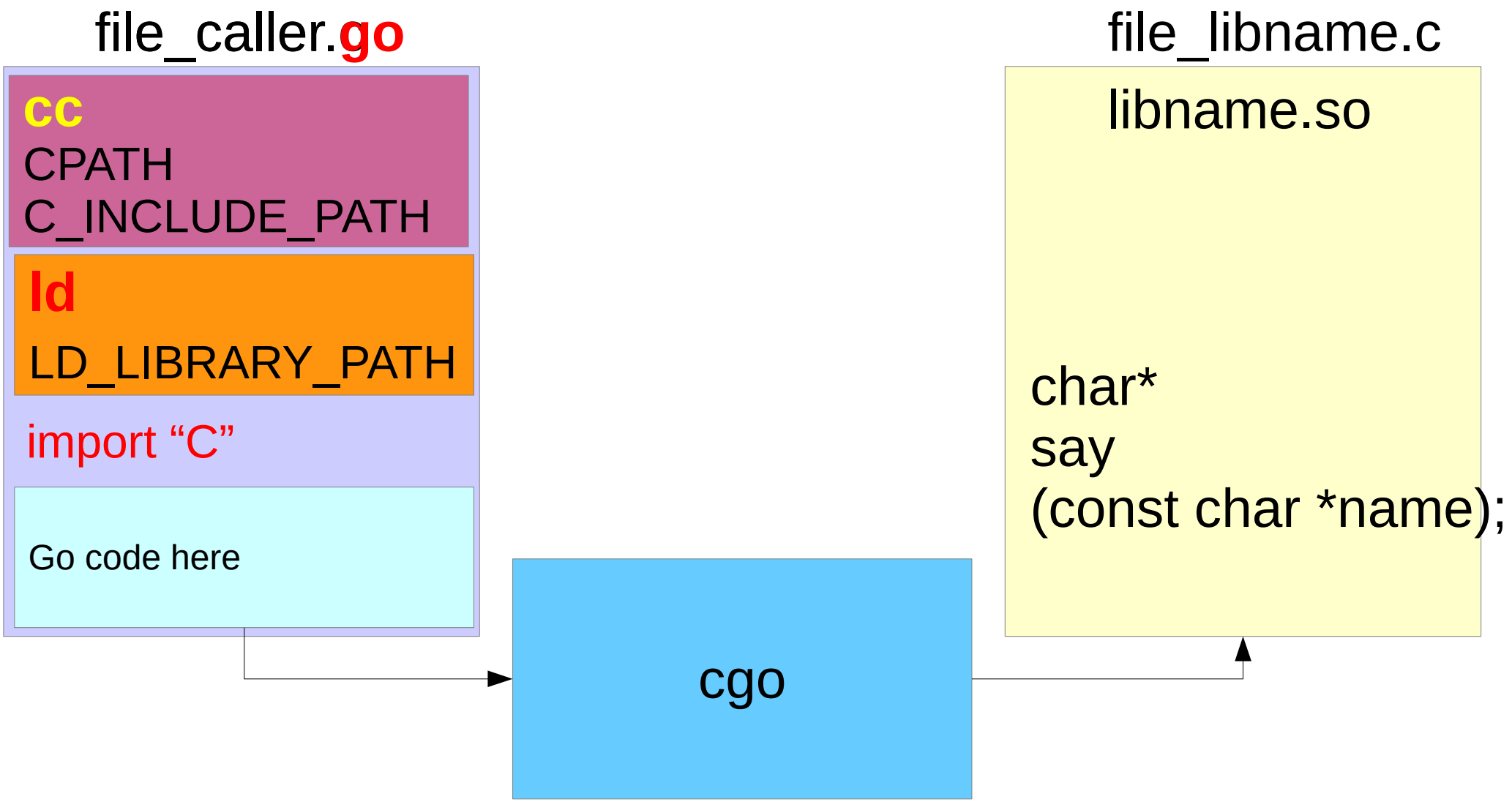
- void * dlopen(const char *filename, int flag);
  - Open the shared file and map it.
- void * dlsym(void *handle, char *symbol);
  - Find the run-time address in shared object
- char *dlerror (void);
  - Return a string describing the error.
- int dlclose (void *__handle);
  - Unmap and close a shared object

cgo

- Useful technology that allows Go to interoperate with C libraries
- Generates Thunks and Stubs to bind
    - C to Go
    - Go to C
- Challenges
    - Different stacks for Go and C
    - Garbage collector 'drama'
-

cgo

- package
  - https://golang.org/cmd/cgo/
- Rules for passing pointers
  - https://github.com/golang/proposal/blob/master/de
-


file_caller.go
cc
CPATH
C_INCLUDE_PATH
ld
LD_LIBRARY_PATH
import “C”
Go code here
cgo
file_libname.c
libname.so
char*
say
(const char *name);

cgo Pointer 'passing' restrictions

strikr

- Go code may pass a Go pointer to C provided the Go memory to which it points does not contain any Go pointers.
- C code must not store any Go pointers in Go memory, even temporarily.
- C code may not keep a copy of a Go pointer after the call returns.
- Go code may not store a Go pointer in C memory.
- Checked dynamically at runtime GODEBUG=cgocheck=2

go packages and 'cgo'

strikr

```
crypto/x509
os/user
go/internal
cmd
net
runtime
runtime/cgo
runtime/race
```

src/net

conf_netcgo.go
cgo_stub.go

cgo_resnew.go
cgo_resold.go

cgo_sockold.go
cgo_socknew.go

cgo_linux.go
cgo_windows.go

cgo_openbsd.go
cgo_bsd.go
cgo_netbsd.go

cgo_solaris.go
cgo_unix.go

cgo_unix_test.go
cgo_android.go

cgo – what to know

strikr

- Unavoidable when working with binary blob
  - Graphics driver
  - Windowing system
- Slower build times
  - C compiler in focus, works on every C file across packages to create a single .o file
  - Linker works through .o file to resolve the shared objects referenced
  - Cross compiling not possible

cgo – what to know

strikr

- cgo is not go, both cc and go compiler needed
- Cross-compiling is disabled when cgo is operational
- Go tools don't work
- Performance issues due mis-matched 'call' stacks
- C decides not Go (addr, sig, tls)
- No longer single static binary
- What was the garbage collector upto ;P

cgo usage references

- 37: error : use of undeclared identifier
  http://www.mischiefblog.com/2014/06/24/a-go-cgo-g
- Using C libraries with Go
  https://jamesadam.me/2014/11/23/using-c-libraries-w
- cgo is not Go
  https://dave.cheney.net/2016/01/18/cgo-is-not-go

Gotcha's

- Why use unicode characters in function names in the Go source code https://groups.google.com/forum/#!msg/golang-nuts/
- Slashes and dots in function names in prototypes https://stackoverflow.com/questions/13475908/slash
- errno http://noeffclue.blogspot.in/2011/10/experimenting-w
-

plugin

strikr

- A plugin is a Go main package with exported functions and variables that has been built with
  - `go build -buildmode=plugin`
- Isomorphic to dlfcn design
- Plugins work only on Linux
- `import “plugin”`
- A plugin is only initialized once, and cannot be closed.
- `https://golang.org/pkg/plugin/`

plugins

- Export symbol
  - using “//export”
- Leverage -buildmode argument
  - go build -buildmode=

```
archive
c-archive
c-shared
default
shared
exe
pie
plugin
```

plugin

plugin.go

plugin_dlopen.go
plugin_stubs.go

```
type Plugin struct {
   pluginpath string
   loaded     chan struct{} // closed when loaded
   syms       map[string]interface{}
}


func Open(path string) (*Plugin, error) {
   return open(path)
}


func (p *Plugin) Lookup(symName string) (Symbol, error) {
   return lookup(p, symName)
}


type Symbol interface{}
```

plugin_dlopen.go

C part

```
static uintptr_t pluginOpen(const char* path, char** err);
static void* pluginLookup(uintptr_t h, const char* name, char** err);
```

```
import "C"
```

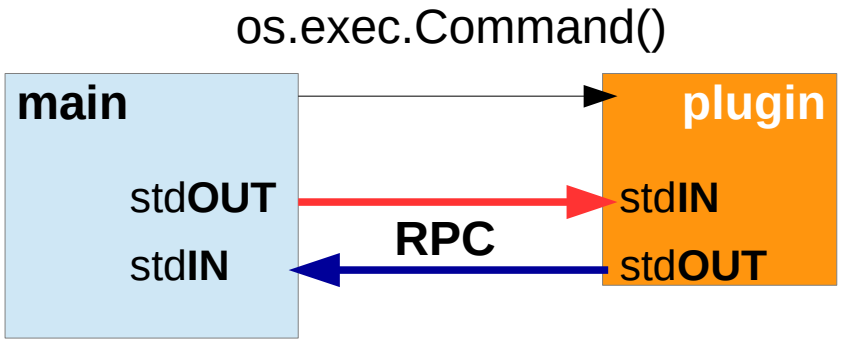
Go part

```
func pathToPrefix(s string) string
func open(name string) (*Plugin, error)
func lookup(p *Plugin, symName string) (Symbol, error)
func lastmoduleinit() (pluginpath string, syms map[string]interface{}, mismatchpkg string)
```

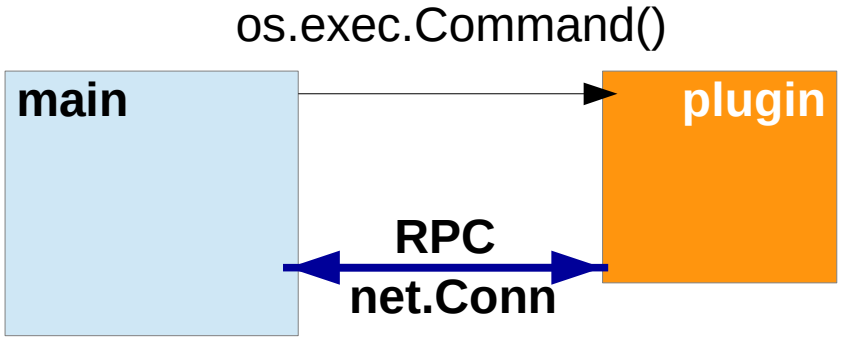
plugin patterns

strikr

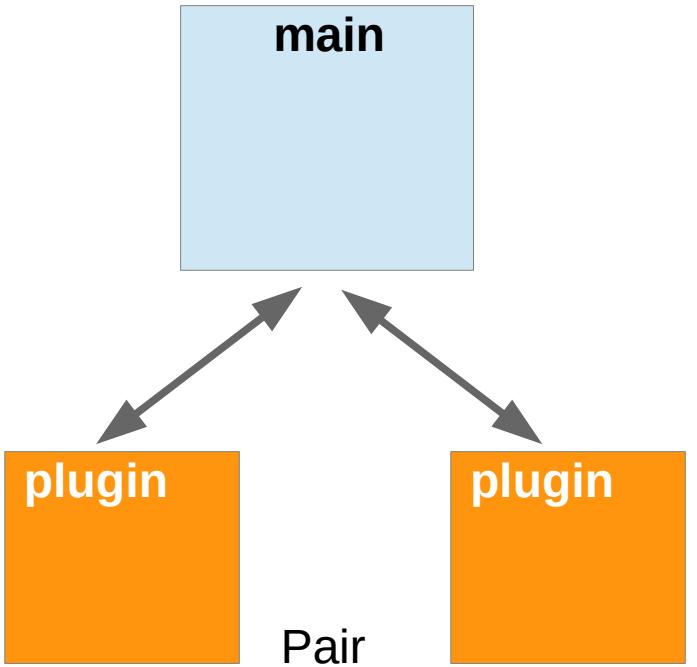
External process using RPC via std IN/OUT



External process using RPC via network



nanomsg “scalability protocols”



Pair
PubSub
Bus
Survey
Pipeline
ReqRep

Code references

- A toolkit for creating plugins for Go applications
  https://github.com/natefinch/pie
- nanomsg socket library for several communication patterns
  http://nanomsg.org/
- scalable protocol implementations in Go
  https://github.com/go-mangos/mangos
- Hashicorp go plugin system
  https://github.com/hashicorp/go-plugin

References

- Go plugins are easy as a pie
  https://npf.io/2015/05/pie/

- Go lang plugin system over RPC
  https://github.com/hashicorp/go-plugin

- Plugin in Go
  https://appliedgo.net/plugins/

Thank You

http://strikr.in/

Thank You

http://strikr.in/