Technology Consulting Company
Research, Development &
Global Standard

Using DPDK with Go

Takanari Hayama
taki@igel.co.jp

BACKGROUND

Background

- Lagopus (https://github.com/lagopus/lagopus)
  - Open Source OpenFlow 1.3 Software Switch
  - DPDK or Raw Socket
  - C

- Lagopus2 (https://github.com/lagopus/vsw)
  - OpenSource Software Router (VLAN, IPsec, Match-Action)
  - DPDK Only
  - Go + C

Goals of Lagopus2

- Performance
- Maintainability + Extensibility

Goals of Lagopus2

- Performance → DPDK + C
- Maintainability + Extensibility → Go

What is Go?

Open Source Programming Language

- Simple
- Strong Type System
- Statically Typed with Flexibility
- Concurrency
- Garbage Collection
- Compiled Language
- Can use C Library via CGo

Go vs C

	Go	C
Complexity	Simple by Design	Can Become Complex
Performance	Moderate	Very Fast
Key-Value Data Type	Yes (Map)	No (requires other library)
Concurrency	Yes (channel and go func)	No (requires other tools)
Memory Management	Yes (Garbage Collection)	No
Compiled Language	Yes	Yes
Build System	Built-in	Your Choice

Performance

Goal

- Data Plane shall run fast
- Control Plane can be slow
- Control Plane shall not disturb Data Plane

Design

- Use C + DPDK directly where we need performance
- Let C to focus on packet processing
- Complicated tasks to be offloaded to Go
- Use DPDK Ring for communication between C and Go codes
- Make lock-free where possible

Maintainability + Extensibility

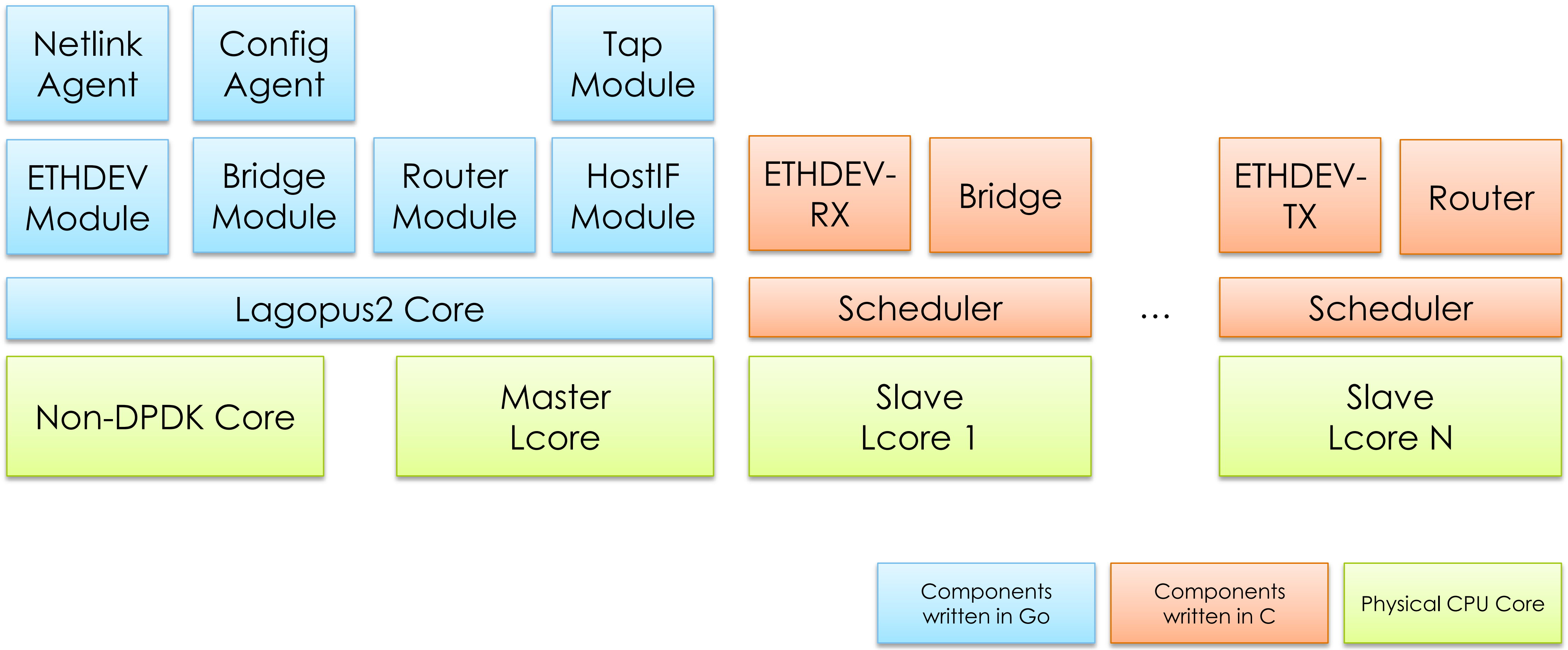
Goal

- Keep the code simple

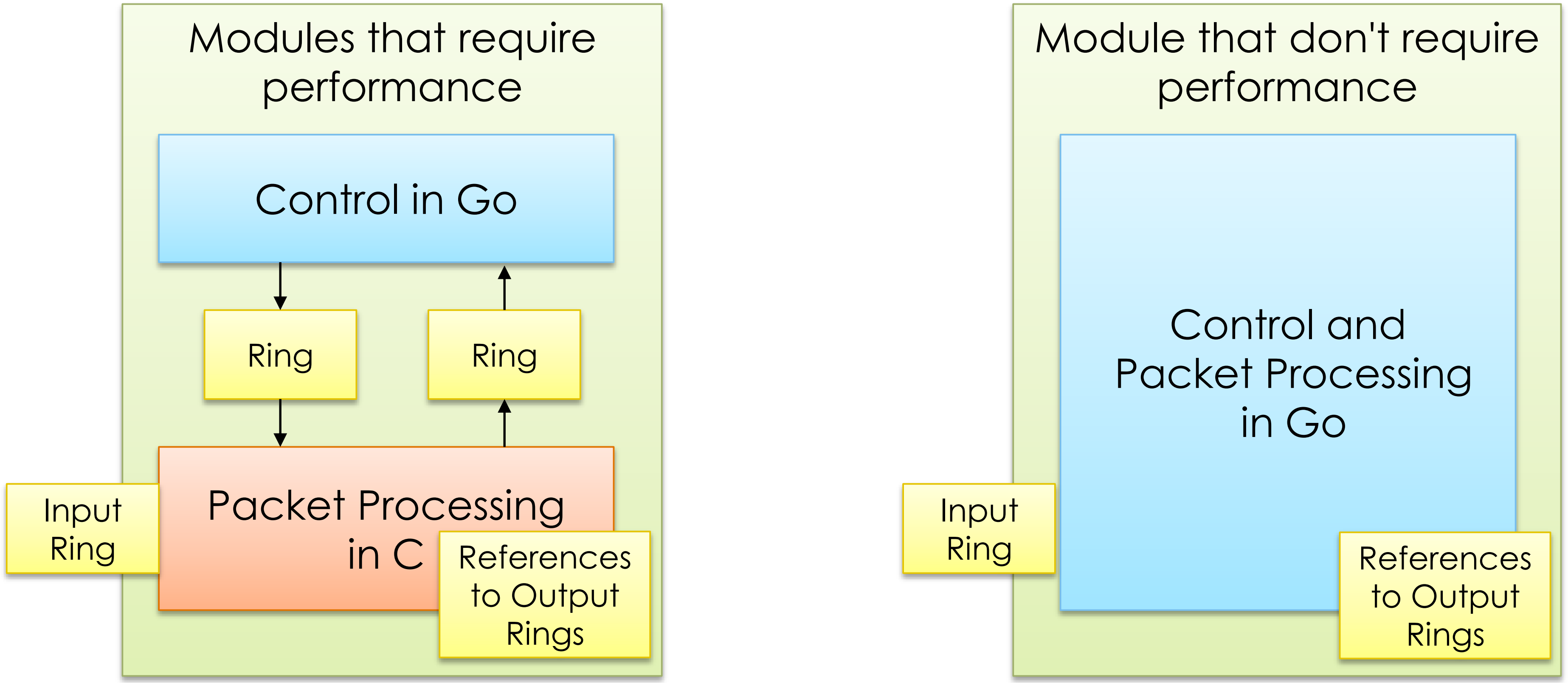
Design

- Anything performance is not that important, do it in Go
- Make C part as simple as possible
- Make good use of Go types, i.e. Slice and Map, to make code simple
- Make good use of existing library, i.e. DPDK

Lagopus2 Architecture



Architecture


Modules that require performance
Control in Go
Ring
Ring
Input Ring
Packet Processing in C
References to Output Rings
Module that don't require performance
Control and Packet Processing in Go
Input Ring
References to Output Rings

USING DPDK FROM GO

Making Good Use of Go

- Type Safety
- Simplicity
- Performance

Type Safety

DPDK API make heavy use of generic types, such as unsigned, int, uint8_t, like any other C based library.

For Go, we should have type safety.

- e.g. Make sure port_id passed to rte_eth_dev_* APIs is always valid port ID.

Example: Type Safety

```
type EthDev struct {
	port_id   uint
	socket_id int
}

type EthDevInfo C.struct_rte_eth_dev_info

func EthDevOpen(port_id uint) (*EthDev, error) {
	pid := C.uint8_t(port_id)
	if int(C.rte_eth_dev_is_valid_port(pid)) == 0 {
		return nil, fmt.Errorf("Invalid port ID: %v", port_id)
	}
	return &EthDev{port_id, int(C.rte_eth_dev_socket_id(pid))}, nil
}

func (re *EthDev) DevInfo() *EthDevInfo {
	var di EthDevInfo
	C.rte_eth_dev_info_get(C.uint8_t(re.port_id), (*C.struct_rte_eth_dev_info)(&di))
	return &di
}
```

Simplicity

Most of DPDK API such as rte_ring passes around handles.

Define API as Methods, not Functions, to wrap DPDK API for particular types.

- Clarify that the APIs are for particular types.
- Hide details that are not necessary for callers.
- Minimize the risks for anything may go wrong.

Example: Simplicity

```
type Ring C.struct_rte_ring
type RingFlags uint

const (
	RING_F_SP_ENQ = RingFlags(C.RING_F_SP_ENQ)
	RING_F_SC_DEQ = RingFlags(C.RING_F_SC_DEQ)
)

func RingCreate(name string, count uint, socket_id int, flags RingFlags) *Ring {
	cname := C.CString(name)
	defer C.free((unsafe.Pointer)(cname))
	return (*Ring)(C.rte_ring_create(cname, C.unsigned(count),
				C.int(socket_id), C.unsigned(flags)))
}

func (r *Ring) Free() {
	C.rte_ring_free((*C.struct_rte_ring)(r))
}

func (r *Ring) Enqueue(obj unsafe.Pointer) bool {
	return int(C.rte_ring_enqueue((*C.struct_rte_ring)(r), obj)) == 0
}
```

Performance

Even though we can't achieve real performance in Go, we definitely want relatively good performance.

Avoiding memory copy is cruicial.

Example: Performance

```
type EtherHdr []byte

func (mb *Mbuf) EtherHdr() EtherHdr {
        len := C.sizeof_struct_ether_hdr
        mb.checkAndUpdateMbufLen()
        return (EtherHdr)((*[1 << 30]byte)(unsafe.Pointer(uintptr(mb.buf_addr) +
                                              uintptr(mb.data_off)))[:len:len])
}
```

You can create a Go slice from the underlying C array without copying the array.
When the slice is released, only the reference to the C array is released.
Underlying C array remains until the array is explicitly released in C.

But… You Need to be Careful

Go automatically releases memory allocated in Go when they're not needed anymore.

HOWEVER, anything allocated in C shall be released explicitly. You have full responsibility!

- E.g., you must explicitly free ring when you don't need it anymore.

No destructor, deinit or something similar to free C memory automatically in Go.

Your C type may be different from mine…

```
type Ring C.struct_rte_ring
```

tells, that the type Ring is an alias to struct rte_ring in C.

However, if the type is defined in different package, Go can't check the identity of C types.

```
ring := dpdk.RingCreate("ring", 10, dpdk.SOCKET_ID_ANY, 0)
var cring *C.struct_rte_ring

cring = ring // Error
cring = (*C.struct_rte_ring)(ring) // Error
cring = (*C.struct_rte_ring)(unsafe.Pointer(ring)) // Ok!☹
```

Not quite type safe here… unsafe is really unsafe.

Regular C struct members are invisible

Any name starting with upper characters are exported in Go, i.e. has a global scope.

```
/*
struct my_struct {
        int Visible;
        int invisible;
}
*/
import "C"
type MyStruct C.struct_my_struct
```

You can access to MyStruct.Visible but not to MyStruct.invisible from outside the package.

Should define setter/getter where needed.

```
func (di *EthDevInfo) DefaultRxConf() *EthRxConf {
        rc := di.default_rxconf
        return (*EthRxConf)(&rc)
}
```

Conclusions

Could make DPDK API Go friendly.

Memory management and type conversion requires extra care.

Heavy use of unsafe may cause lots of problem, but sometime they're inevitable.

Useful References

Command cgo - https://golang.org/cmd/cgo/
C? Go? Cgo! - https://blog.golang.org/c-go-cgo
cgo - https://github.com/golang/go/wiki/cgo

QUESTIONS?