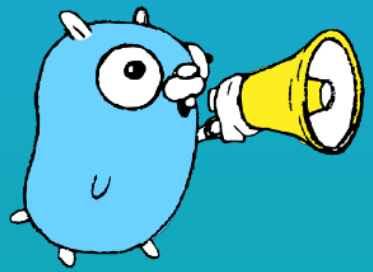
Seoul, MAR 30 2019

Introduction to Go Programming

홍혜종

Golang Korea



목차

SECTION ONE

Go의 역사

History

로버트 그리즈머
(Hotspot JVM)

롭 파이크
(Unix/UTF-8)

켄 톰슨
(B and C lang/Unix/UTF-8)

2007년 9월 21일

분산 운영체제와 관련된 작업을 하다 깊은 고민에 빠진 3명의 프로그래머

History

C++ 및 기존의 언어들은 너무 어렵고
언어 선택에 대한 트레이드 오프가 있네?

멀티 코어 CPU가 보편화되고 있지만
효율적이고 안전하게 프로그래밍을 할 수 있게
서포터해주는 언어가 거의 없네??

두 가지의 고민에 빠졌는데...

1. C++ 및 기존의 언어들은 너무 어렵고

언어 선택에 대한 트레이드 오프가 있다.

빠른 컴파일 or 효율적인 실행 or 프로그래밍의 용이성

주류 언어 중 세가지 특징 모두 다 포함하고 있는 언어는 없었다.

2. 멀티 코어 CPU가 보편화되고 있지만

효율적이고 안전하게 프로그래밍을 할 수 있게

서포터해주는 언어가 거의 없네??

멀티 코어 CPU의 등장은
동시성이나 병렬성에 대한 복잡성을 증가시켰고

또한 대규모 프로그램에서
수동으로 메모리 관리는 어렵고 안전하지 않을 수 있다.

그래서 시작한 이 프로젝트는

History

Subject: Re: prog lang discussion

From: Rob 'Commander' Pike

Date: Tue, Sep 25, 2007 at 3:12 PM

To: Robert Griesemer, Ken Thompson

i had a couple of thoughts on the drive home.

1. name

'go'. you can invent reasons for this name but it has nice properties. it's short, easy to type. tools: goc, gol, goa. if there's an interactive debugger/interpreter it could just be called 'go'. the suffix is .go

...

2007년 9월 25일 메일을 통해 go라는 이름이 지어졌고

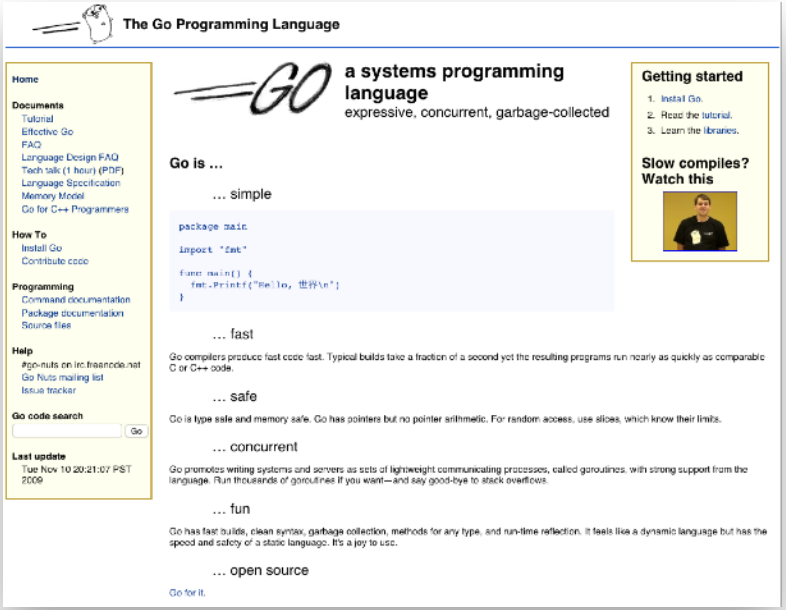
History

이안 랜스 테일러
(GCC)

러스 콕스
(Plan9/CSP)

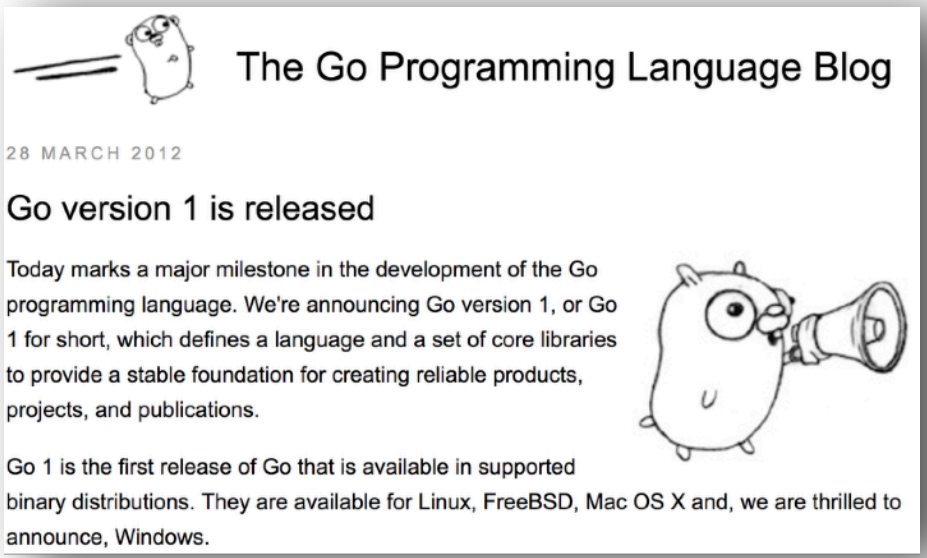
2008년에는 이안 랜스 테일러와 러스 콕스가 합류했고

History



2009년 11월 10일 오픈소스로 공개되었으며

History



The Go Programming Language Blog

28 MARCH 2012

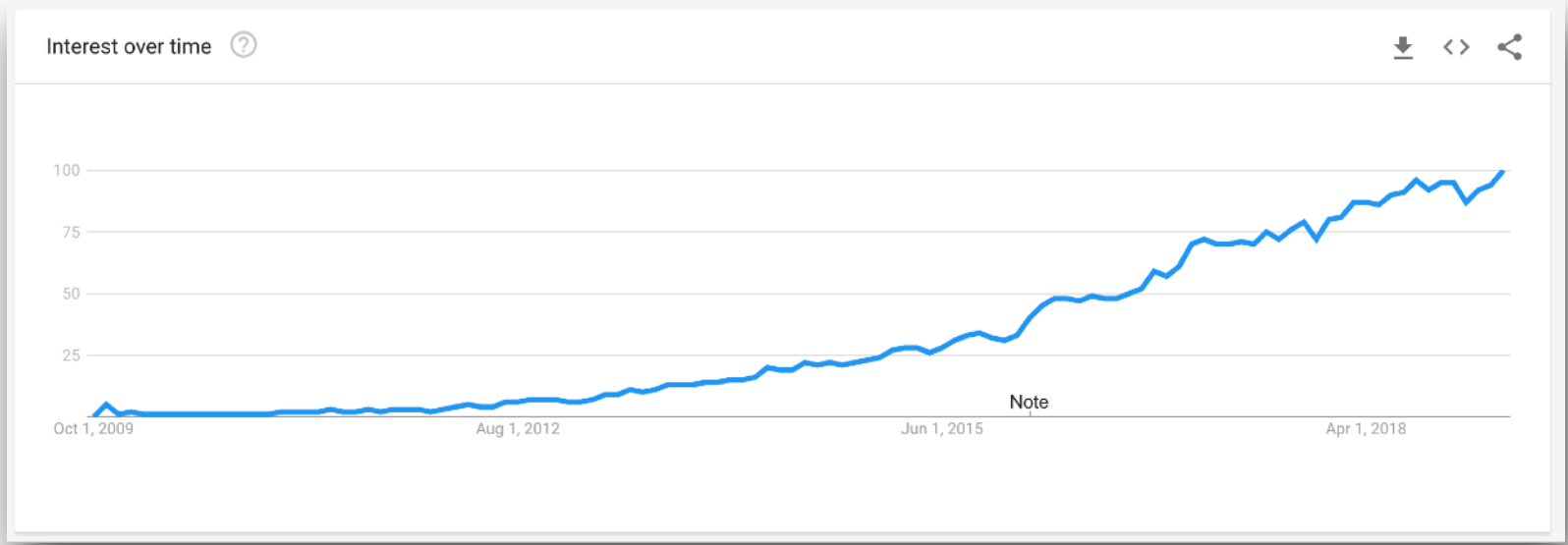
Go version 1 is released

Today marks a major milestone in the development of the Go programming language. We're announcing Go version 1, or Go 1 for short, which defines a language and a set of core libraries to provide a stable foundation for creating reliable products, projects, and publications.

Go 1 is the first release of Go that is available in supported binary distributions. They are available for Linux, FreeBSD, Mac OS X and, we are thrilled to announce, Windows.

2012년 3월 28일에는 버전 1이 릴리즈 됨.

History



그리고 떡상중

Source: trends.google.com

Go의 대표적인 프로젝트

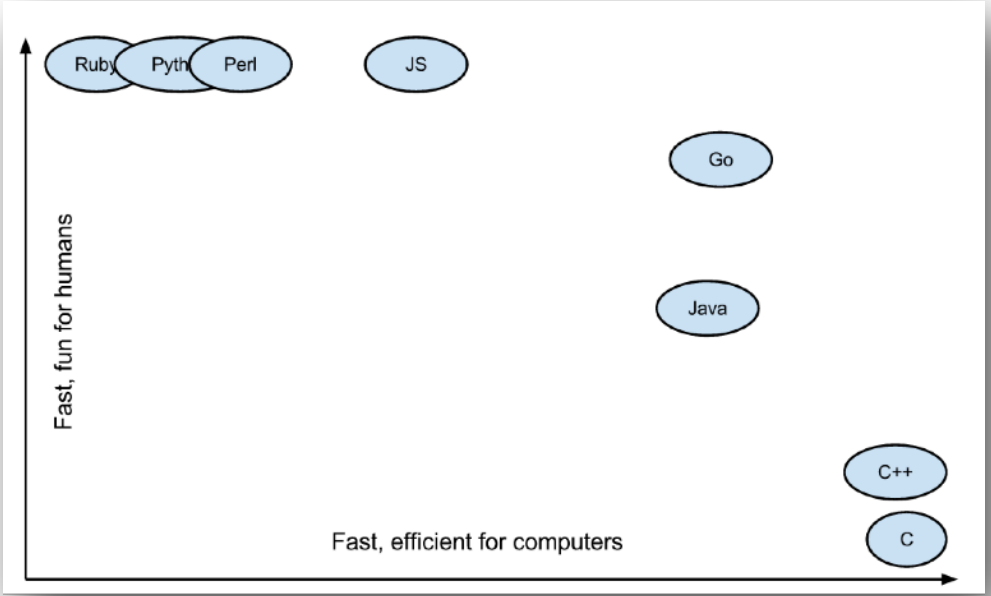
SECTION TWO

Go의 간략한 소개

Features

1. 단순하면서 쉬운 프로그래밍
2. 빠른 컴파일 속도
3. 정적 타입(+interface)
4. 내장 런타임(garbage collector, scheduler, etc)
5. 동시성 지원(CSP)
6. 기본 툴들(build, fetch, test, document, profile, format)
7. 풍부하고 강력한 표준 라이브러리

Go의 포지션



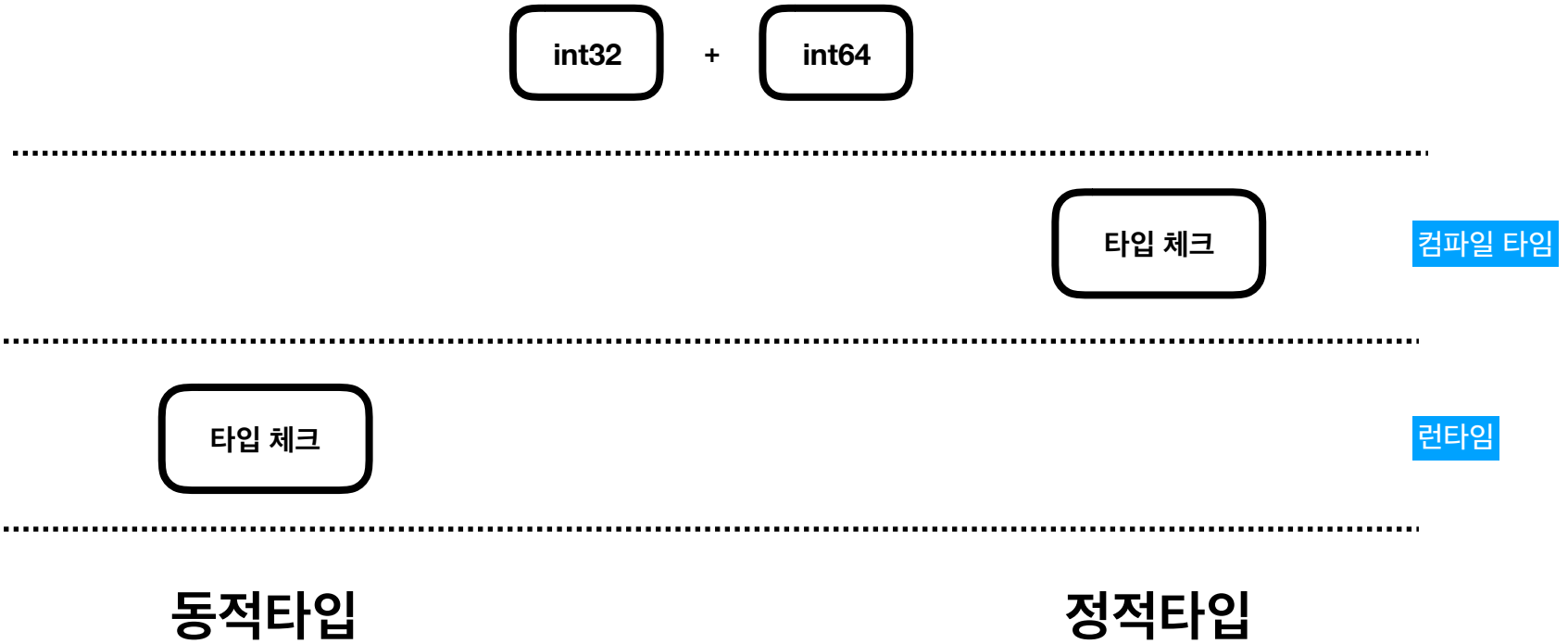
적당히 쉽고!? 적당히 빠른!?

Go의 포지션

break default func interface select
case defer go map struct
chan else goto package switch
const fallthrough if range type
continue for import return var

25개밖의 되지않는 문법 키워드

타입체크



헤더 파일이 없고

전방 선언이 없고

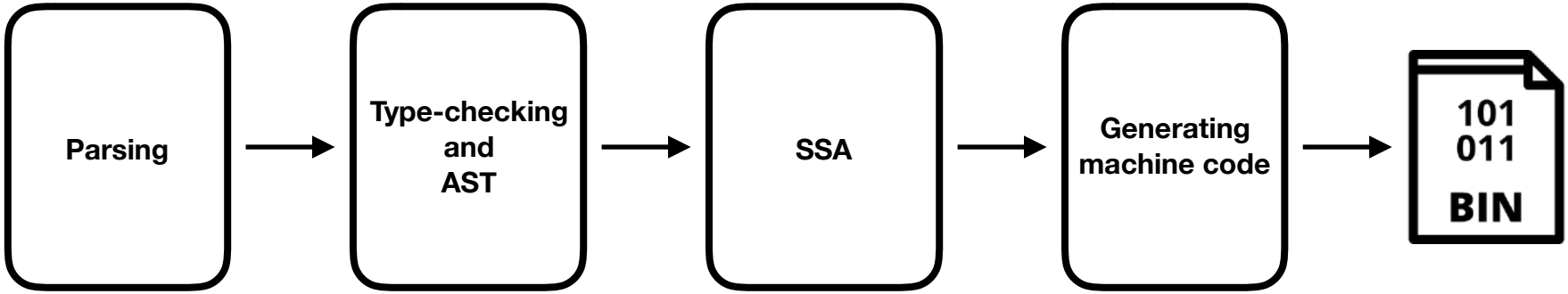
제네릭이 없고

문법이 파싱하기
좋게 디자인 되었고

컴파일러 구현을
속도에 초점을 맞췄고

컴파일이 빠른 이유

컴파일 단계

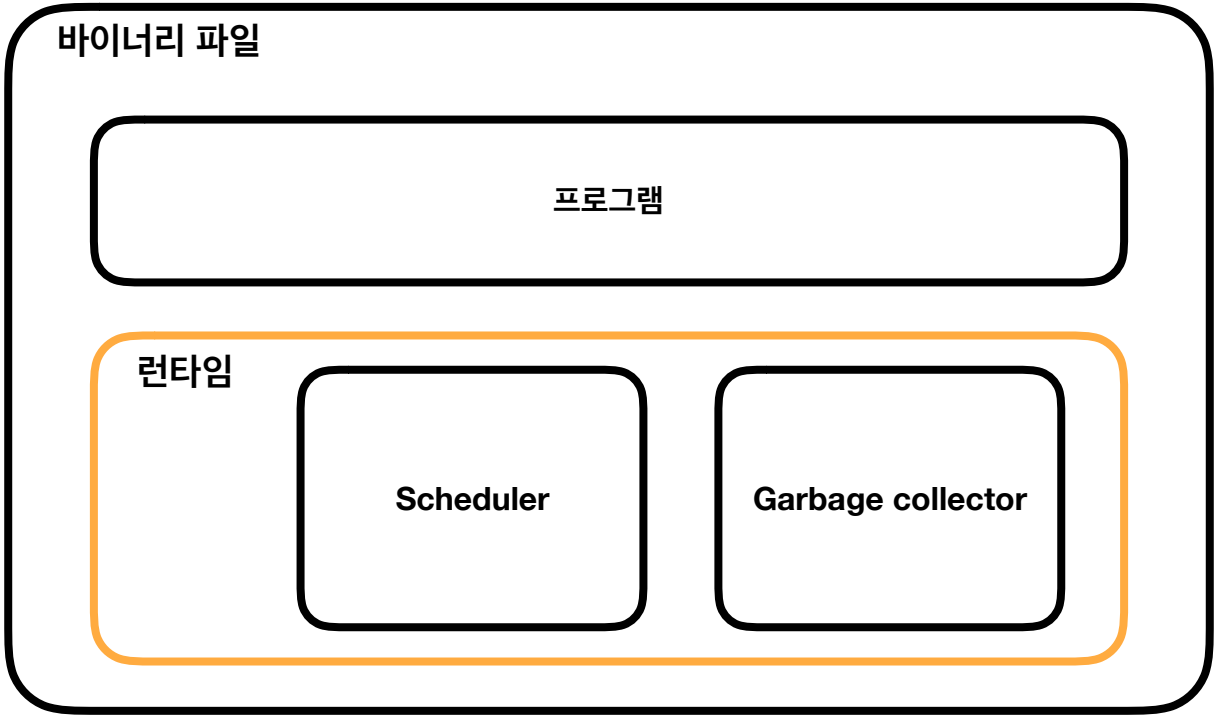

Parsing
Type-checking and AST
SSA
Generating machine code
101
011
BIN

크로스 컴파일

$GOOS	$GOARCH	
darwin	386	-- 32 bit MacOSX
darwin	amd64	-- 64 bit MacOSX
freebsd	386	
freebsd	amd64	
linux	386	-- 32 bit Linux
linux	amd64	-- 64 bit Linux
linux	arm	-- RISC Linux
netbsd	386	
netbsd	amd64	
openbsd	386	
openbsd	amd64	
plan9	386	
windows	386	-- 32 bit Windows
windows	amd64	-- 64 bit Windows

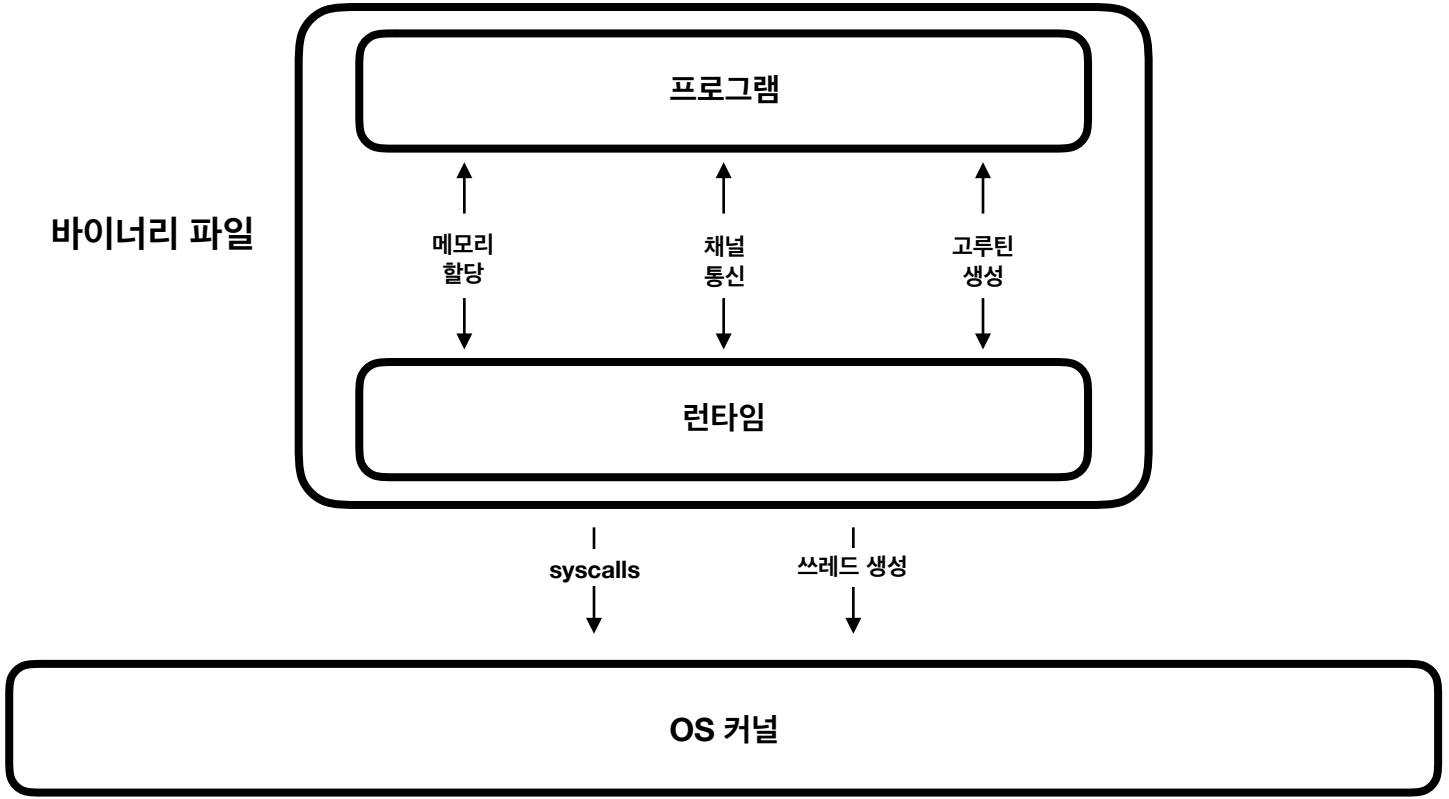
원하는 OS와 CPU 타입을 지정해서
해당 바이너리 파일을 만들 수 있음

바이너리 구조

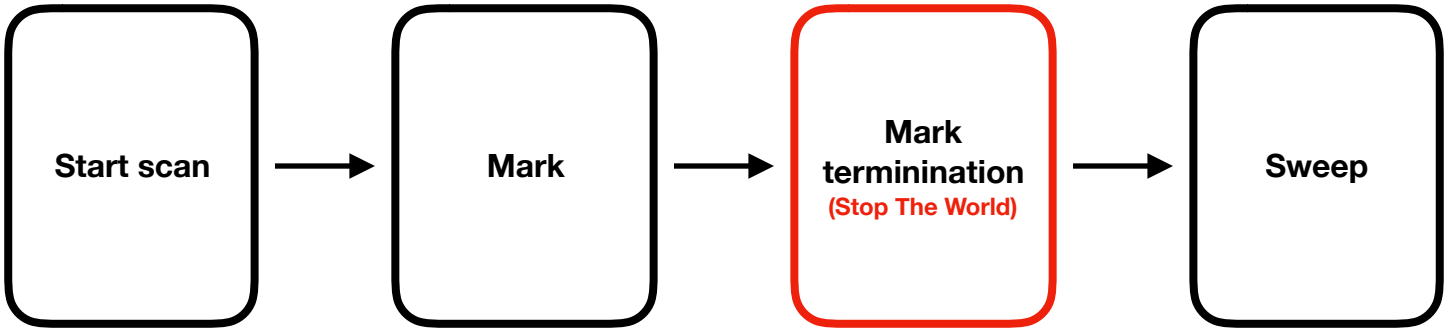


런타임 내장

작동 구조



Garbage Collection

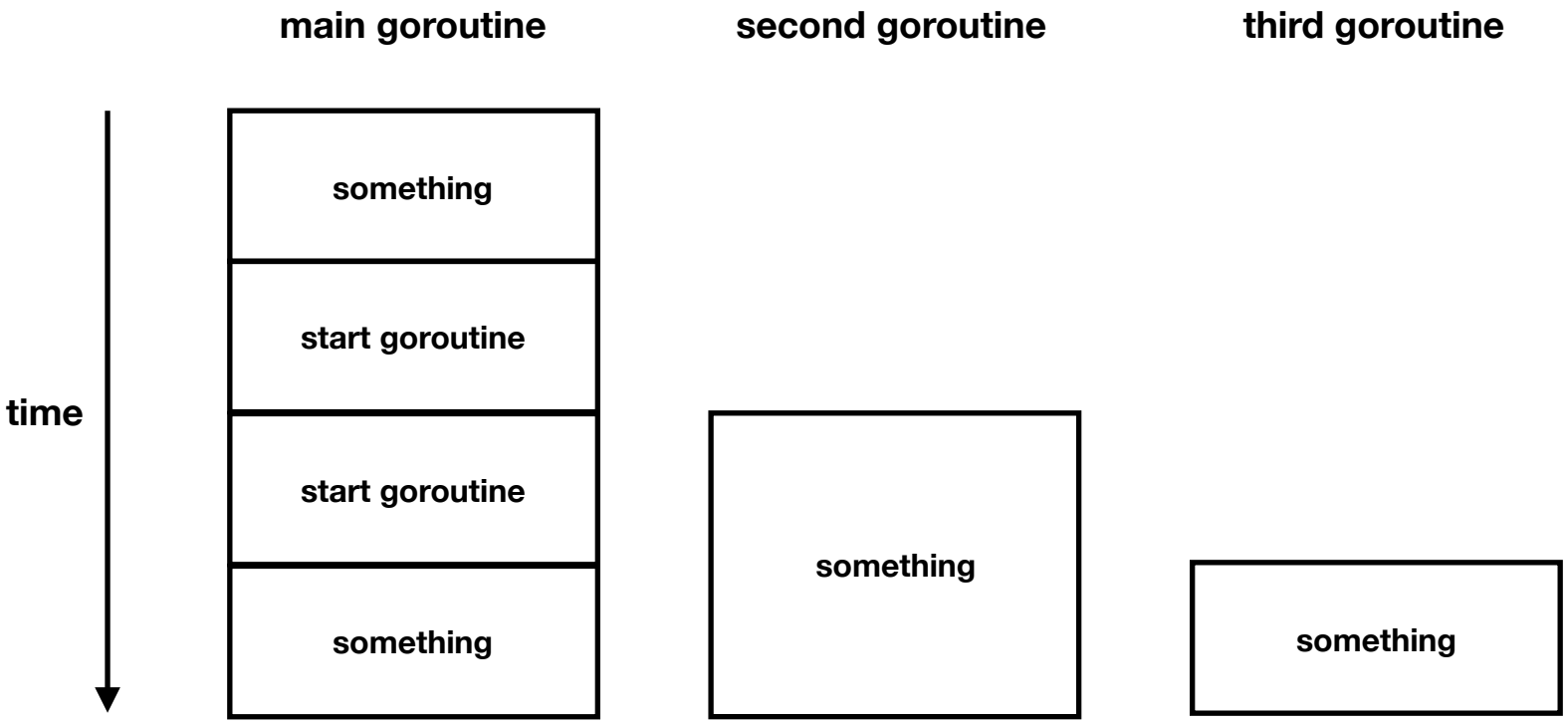


Go의 GC는

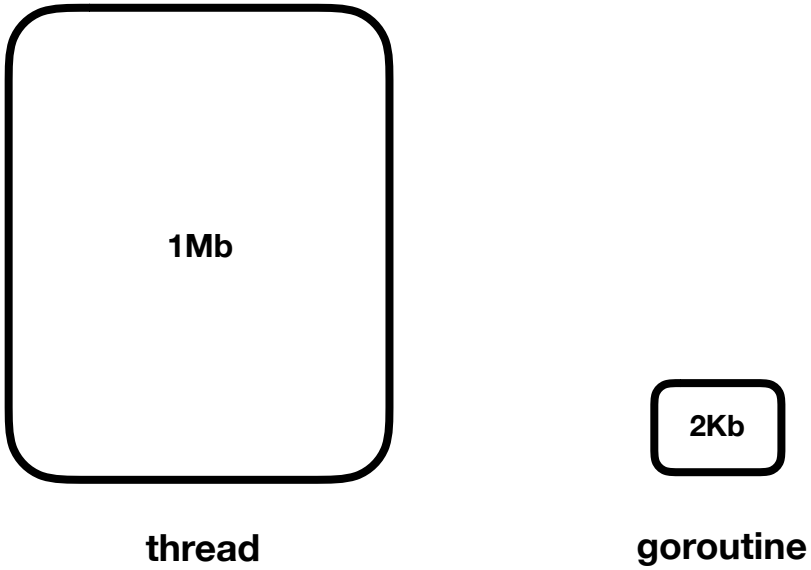
Concurrent Mark & Sweep

(컨셉: 대기시간 > 처리량)

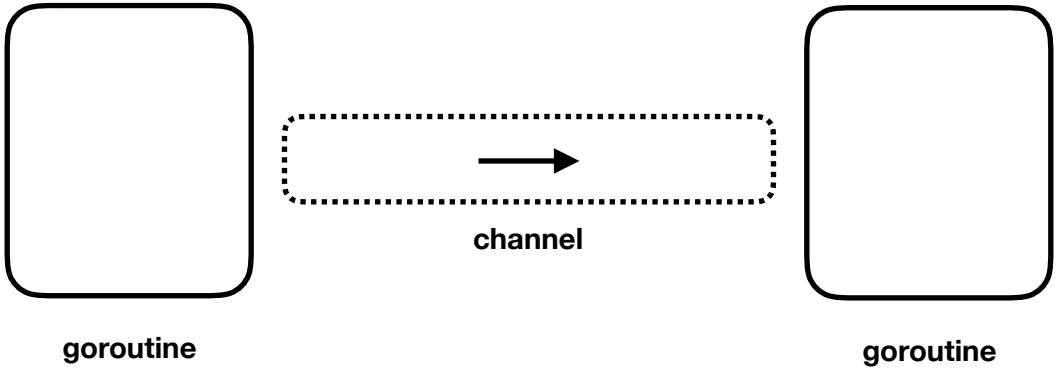
고루틴


main goroutine
second goroutine
third goroutine
time
something
start goroutine
start goroutine
something
something
something

고루틴의 크기



채널



기본 커맨드 툴

- go run
- go build
- go install
- go test
- go clean
- go tool
- go get
- go mod
- go fmt
- go vet
- go fix
- go generate
- gorename
- godoc
- go bug
- go version

https://golang.org/cmd/go/

SECTION THREE

Hello World로 바라보는 Go

Hello World

```
package main

import "fmt"

func main(){
  fmt.Println("Hello world.")
}
```

컴파일/실행 방법

go run	컴파일 & 실행
go build	컴파일
go install	컴파일 된 패키지를 $GOPATH/pkg에 캐시 컴파일 된 바이너리를 $GOPATH/bin에 캐시 Go 1.9부터 build -i와 동일한 효과

GOPATH란?

```
$GOPATH
|- src : 내가 작성한&외부 패키지의 소스코드 저장
|- pkg : 컴파일된 패키지의 캐시 저장
|- bin : 컴파일된 바이너리 저장
```

GOPATH는 워킹 디렉토리를 가르키는 환경변수

1.8부터 기본으로 $HOME/go

GOPATH란?

```
$GOPATH
|- pkg
|- bin
|- src
    |- github.com
        |- user
            |- package
                |- main.go
```

내가 작성한&외부 패키지의 소스코드 저장

GOPATH란?

```
$GOPATH
|- bin
|- src
|- pkg
   |-{OS}_{CPU}
         |- github.com
               |- user
                     |- package
                           |- main.a
```

컴파일된 패키지의 캐시 저장

GOPATH란?

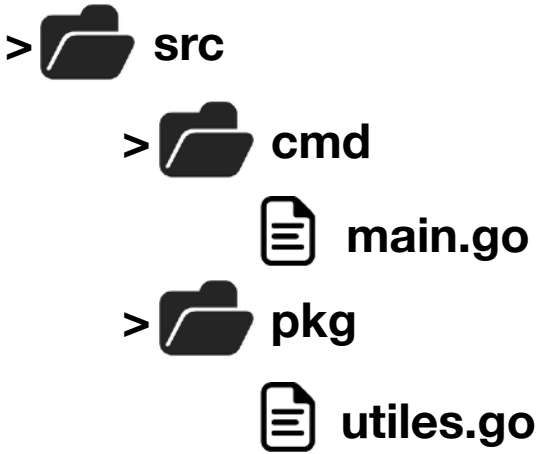
```
$GOPATH
|- pkg
|- src
|- bin
    |- binary
    |- binary
    |- binary
```

컴파일&설치된 바이너리 저장

Hello World

```
package main

import "fmt"

func main(){
  fmt.Println("Hello world.")
}
```

패키지란?



Go의 소스파일은 패키지(디렉토리) 안에 있어야 함.

패키지에는 타입, 함수, 변수 및 상수가 포함되어 있음.

Hello World

```
package main

import "fmt"

func main(){
  fmt.Println("Hello world.")
}
```

import의 특징

- 대소문자로 접근 권한 구분.
- 표준 라이브러리 접근($GOROOT/src)
- 외부 패키지 접근($GOPATH/src)
- 상대 경로로 접근(“../foo”)
- 식별자 재정의(h “text/template”)
- 빈 식별자(_ "lib/math")

```
package main

import "fmt"

func main(){
  fmt.println("Hello world.")
}
```

식별자 재정의

```
package main

import (
    t "text/template"
    h "html/template"
)

func main() {
    t.New("foo").Parse(`{{define "T"}}Hello, {{.}}!{{end}}`)
    h.New("foo").Parse(`{{define "T"}}Hello, {{.}}!{{end}}`)
}
```

빈 식별자

```
package main

import (
    "database/sql"
    "fmt"

    _ "github.com/go-sql-driver/mysql"
)

func main() {
    db, err := sql.Open("mysql", "user:password@/database")
    if err != nil {
        panic(err.Error())
    }
    defer db.Close()

    fmt.Println(db.Ping())
}
```

SECTION FOUR

Go의 문법

- Function
- Type
- Container

Functions

```
func double(x int) int { return 2 *x }
func div(x, y int) (int, error) { return 5, nil }
func splitHostIP(s string) (host, ip string)  { return }

var even func(x int) bool
even := func(x int) bool { return x%2 == 0 }
```

클로저(Closure)

```
func intSeq() func() int {
    i := 0
    return func() int {
        i++
        return i
    }
}

func main(){
    nextInt := intSeq()
    fmt.Println(nextInt())
    fmt.Println(nextInt())
    fmt.Println(nextInt())

    newInts := intSeq()
    fmt.Println(newInts())
}
```

미리 선언된 함수들(Predefined functions)

1. init

2. defer

3. panic, recover

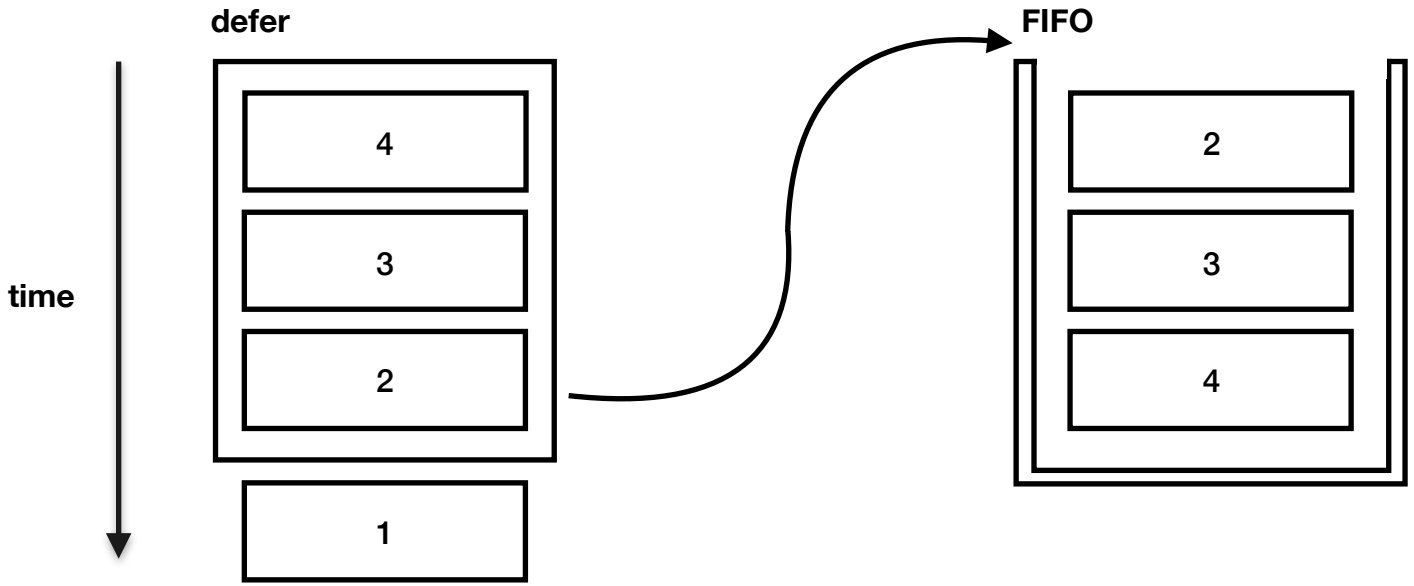
init

```
package main

import "fmt"

func init(){
   fmt.Println("before")
}

func main(){
   fmt.Println("after")
}
```

defer

```
package main

import "fmt"

func main(){
   defer fmt.Println("4")
   defer fmt.Println("3")
   defer func() {
       fmt.Println("2")
   }()

   fmt.Println("1")
}
```

defer 작동 흐름



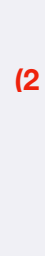
Quiz 1

```go
func watDefer(i int) (result int) {
    defer func() {
        result = result + 1
    }()
    defer func() {
        result = result * 3
    }()
    result = i * 2
    return 20
}

func main() {
    result := watDefer(5)
    fmt.Println(result)
}
```

출력되는 값과 왜 그렇게 출력되는지 이유를 설명해주세요.

Quiz 1(해설)

```go
func watDefer(i int) (result int) {
    defer func() {
        result = result + 1
    }()
    defer func() {
        result = result * 3
    }()
    result = i * 2
    return 20
}

func main() {
    result := watDefer(5)
    fmt.Println(result)
}
```



61 = (20*3)+1

panic

```
package main

import "fmt"

func main(){
   panic("Error!!")

   fmt.Println("Hello World.")
}
```

recover

```
func Something(){
   defer func(){
        if r := recover(); r != nil {
            fmt.Println(r)
        }
   }()

   panic("Error!!")
}

func main(){
   Something()

   fmt.Println("Hello World.")
}
```

Predefined types

- string
- bool
- int int8, int16, int32, int64
- uint uint8, uint16, uint32, uint64, uintptr(uint)
- byte(uint8)
- rune(int32)
- float32, folat64
- complex64, complex128(복소수)
- error
- nil

Predefined types

숫자 타입

int int8, int16, int32(rune), int64

uint uint8(byte), uint16, uint32, uint64, uintptr(uint)

float32, folat64, complex64, complex128

기타

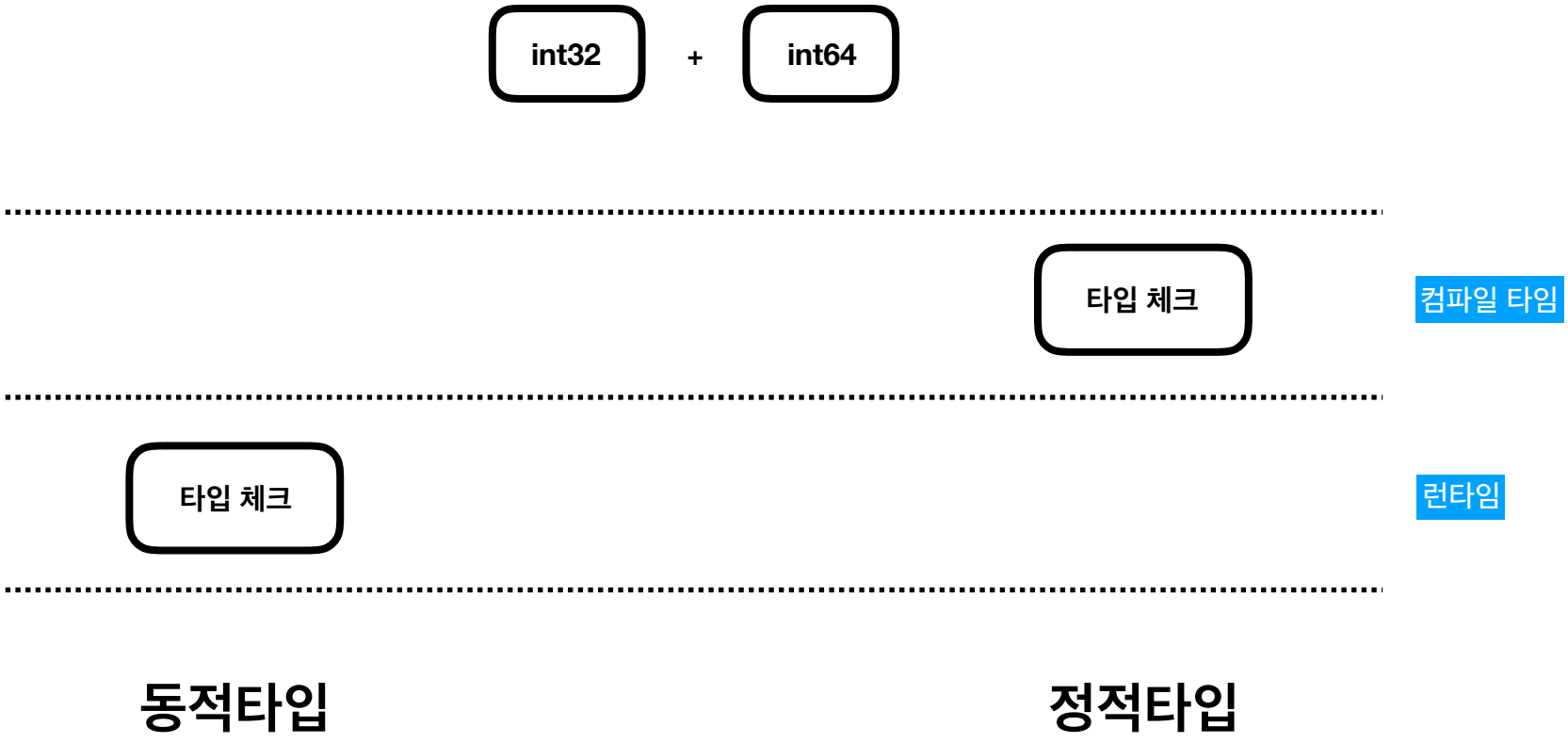
string, bool, error, nil

$GOROOT/src/builtin/builtin.go

타입 체크 시스템

```
package main

import "fmt"

func main(){
   var i32 int32 = 0
   var i64 int64= 0

   sum := i32 + i64

   fmt.Println(sum)
}
```

타입 체크 시스템



변수 선언의 4가지 방법

```
package main

import “fmt"

func main() {
   var foo byte
   var bar = 1
   var i, j int = 1, 2
   c, python, java := true, false, "no!"

   fmt.Println(foo, bar, i, j, c, python, java)
}
```

짧은 변수 선언(Short variable declaration)

```
package main

import "fmt"

func main(){

   i := 5

   fmt.Println(i)
}
```

재선언(Redeclaration)

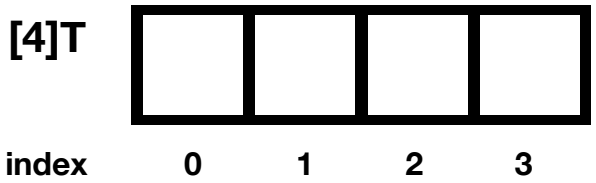
```
package main

import "fmt"

func main() {
   foo := 1
   foo := 2  // ???

   fmt.Println(foo)
}
```

Container types

- Array
- Slice
- Map

Container types

- **Array : 정적 배열**
- **Slice : 동적 배열**
- **Map : Hash 맵**

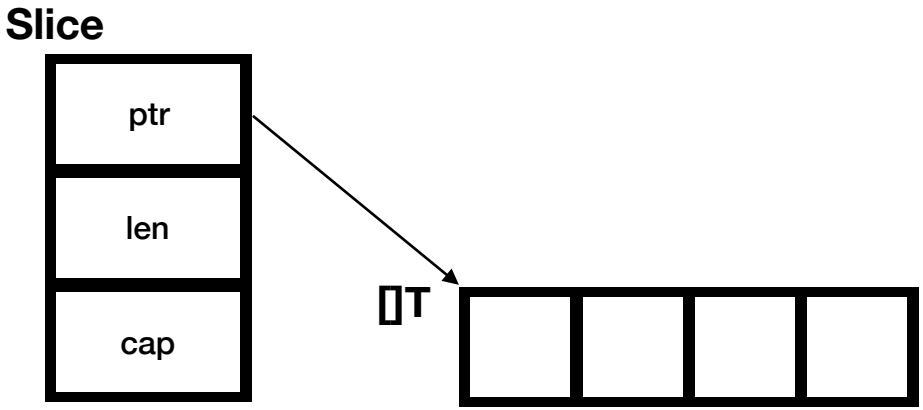


크기가 정해진 배열

Array 선언의 3가지 방법

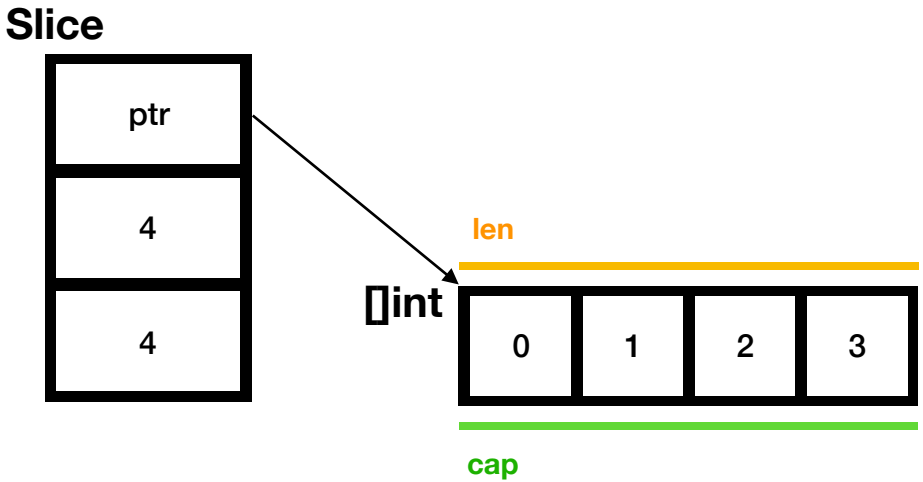
```
package main

import "fmt"

func main(){
    var a [4]int
    b := [2]string{"Penn", "Teller"}
    c := [...]bool{true, false, false}

    fmt.Println(a,b,c)
}
```

Slice



크기가 동적인 배열

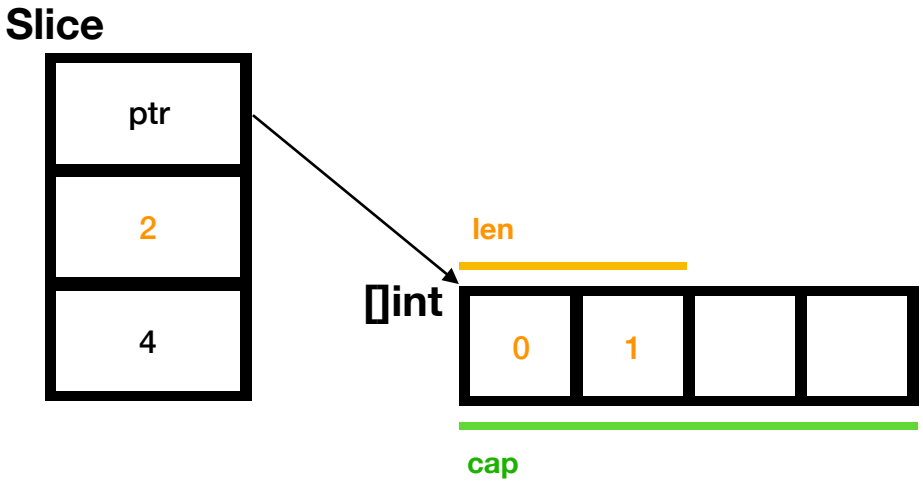
Slice



Slice 선언의 3가지 방법

```
package main

import "fmt"

func main(){
    letters := []string{"a", "b", "c", "d"}
    var bytes  = make([]byte, 5, 5)
    b := make([]int, 3, 3)

   fmt.Println(letters, bytes, b)
}
```

Slice



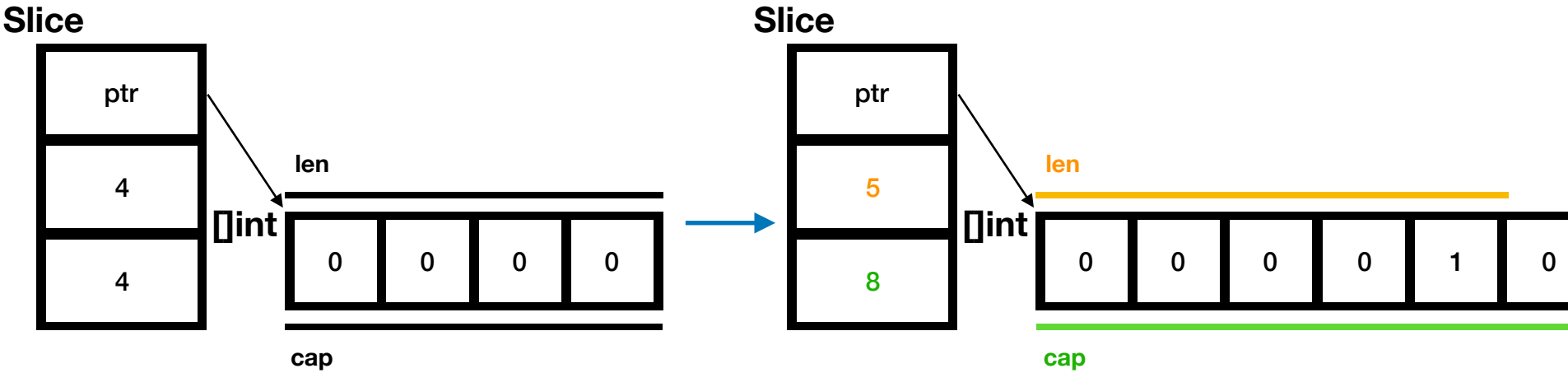

```
b := make([]int, 2, 4)
```

Slice의 확장

```
package main

import "fmt"

func main(){
   s := make([]int, 4, 4)
   s = append(s, 1)

   fmt.Println(s)
}
```

func append(s []T, x ...T) []T

Slice의 확장

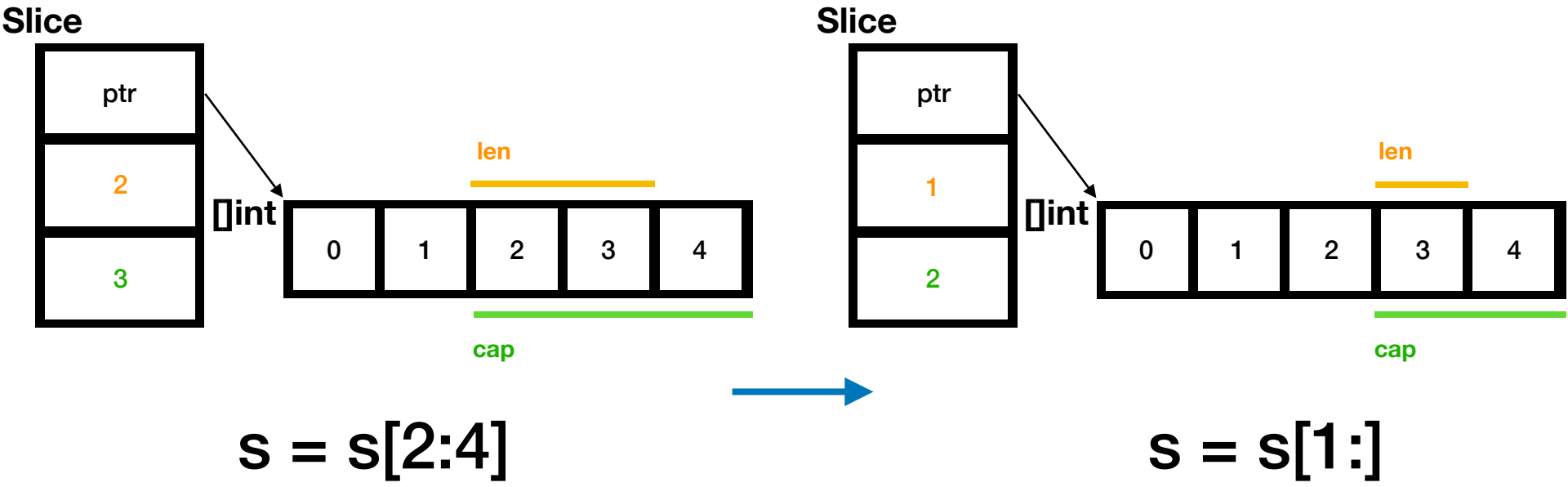


s = append(s, 1)

Sub-slicing

```
package main

import "fmt"

func main(){
   s := []int{0, 1, 2, 3, 4}
   s = s[2:4]
   fmt.Println(s, len(s), cap(s))

   s = s[1:]
   fmt.Println(s, len(s), cap(s))
}
```

Sub-slicing


Slice
ptr
2
3
[]int
len
0
1
2
3
4
cap
Slice
ptr
1
2
[]int
len
0
1
2
3
4
cap
s = s[2:4]
s = s[1:]

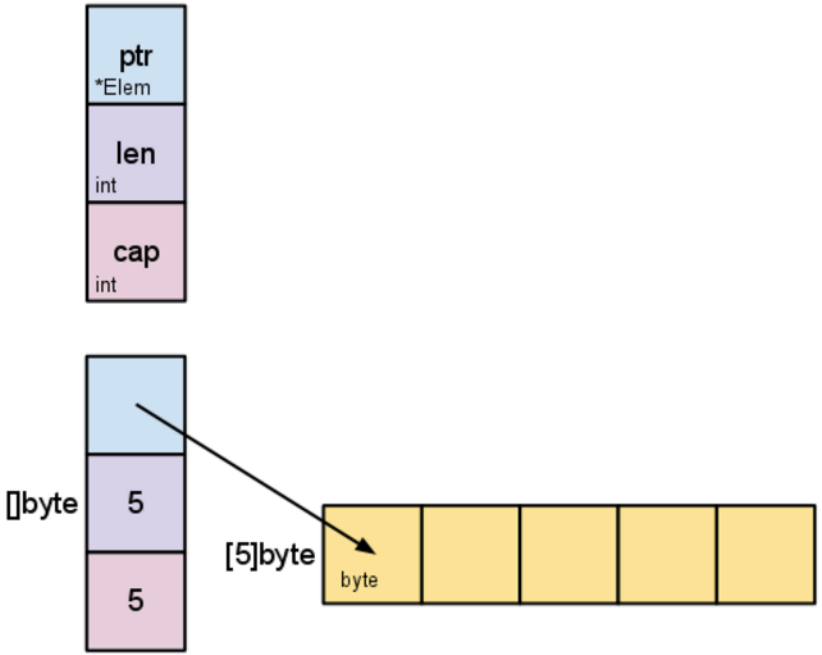
Quiz 2

```go
func grow(s []int) {
    s = append(s, 4, 5, 6)
}

func main() {
    s := []int{1, 2, 3}
    fmt.Println(s)
    grow(s)
    fmt.Println(s)
}
```

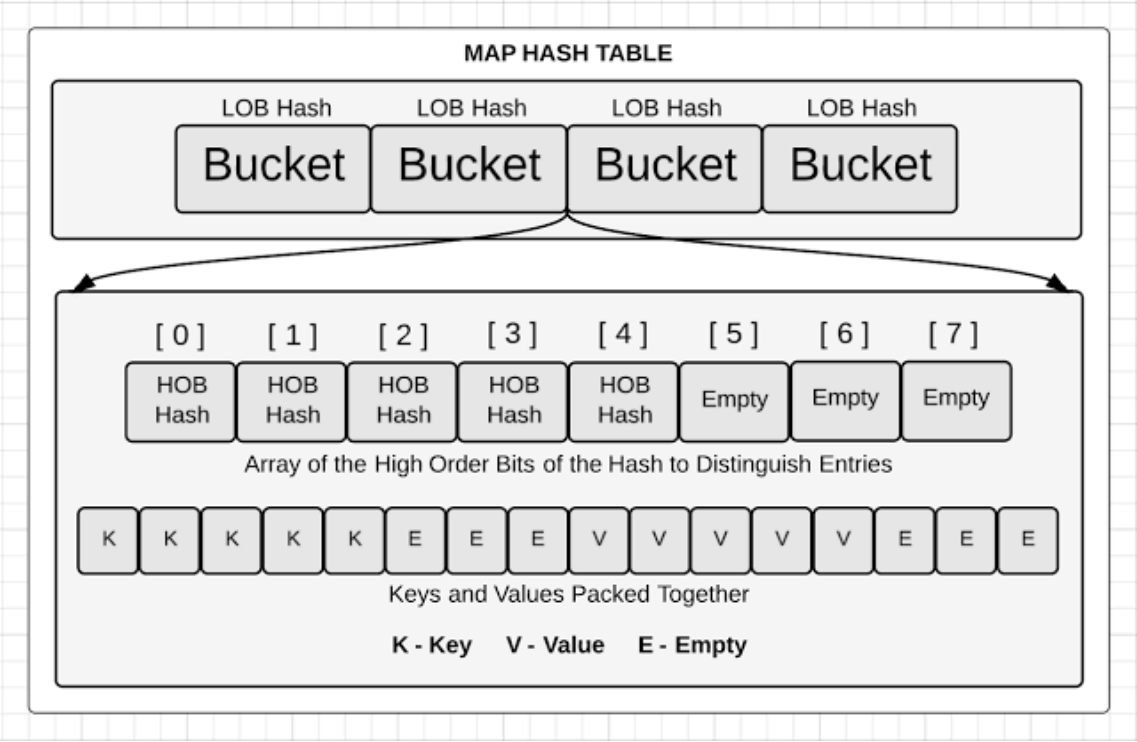
출력되는 값과 왜 그렇게 출력되는지 이유를 설명해주세요.

Quiz 2(해설)

```
type slice struct {
        array unsafe.Pointer
        len   int
        cap   int
}
```



https://github.com/golang/go/blob/master/src/runtime/slice.go



Map

https://www.ardanlabs.com/blog/2013/12/macro-view-of-map-internals-in-go.html

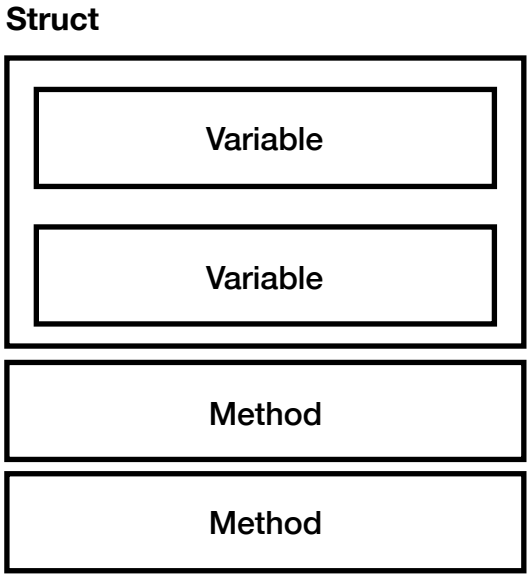
Map 선언의 3가지 방법

```
package main

import "fmt"

func main(){
    var m map[string]int
    m = make(map[string]int)

    names := map[string]int{
        "rsc": 3711,
    }

   tables := make(map[string]int)

   fmt.Println(m, names, tables)
}
```

Map 연산자(read, write, delete)

```
package main

import "fmt"

func main(){
    m := make(map[string]int)

    m["foo"] = 5 // write

    _ = m["foo"] // read

    foo , exist := m["foo"] // second read
    fmt.Println(foo, exist)

    delete(m, "foo") // delete
}
```

Struct



하나 이상의 변수와 메서드를 그룹 지어서
새로운 자료형을 정의하는 것

Struct

```
package main

import "fmt"

type Person struct {
    FirstName string
    LastName  string
    Age       int
}

func main(){
    var h1 Person = Person{FirstName:"Hyejong", LastName:"Hong",Age:29}
    var h2 Person = Person{"Minjae", "Kwon",25}

    fmt.Println(h1, h2)
}
```

Struct 메서드 선언

```
type Person struct {
    FirstName string
    LastName  string
    Age       int
}

func (p Person) Major() bool { return p.Age >= 18 }
func (p *Person) Birthday() { p.Age++ }

func main(){
    var h1 Person = Person{FirstName:"Hyejong", LastName:"Hong",Age:17}
    h1.Birthday()
    fmt.Println(h1.Major())
}
```

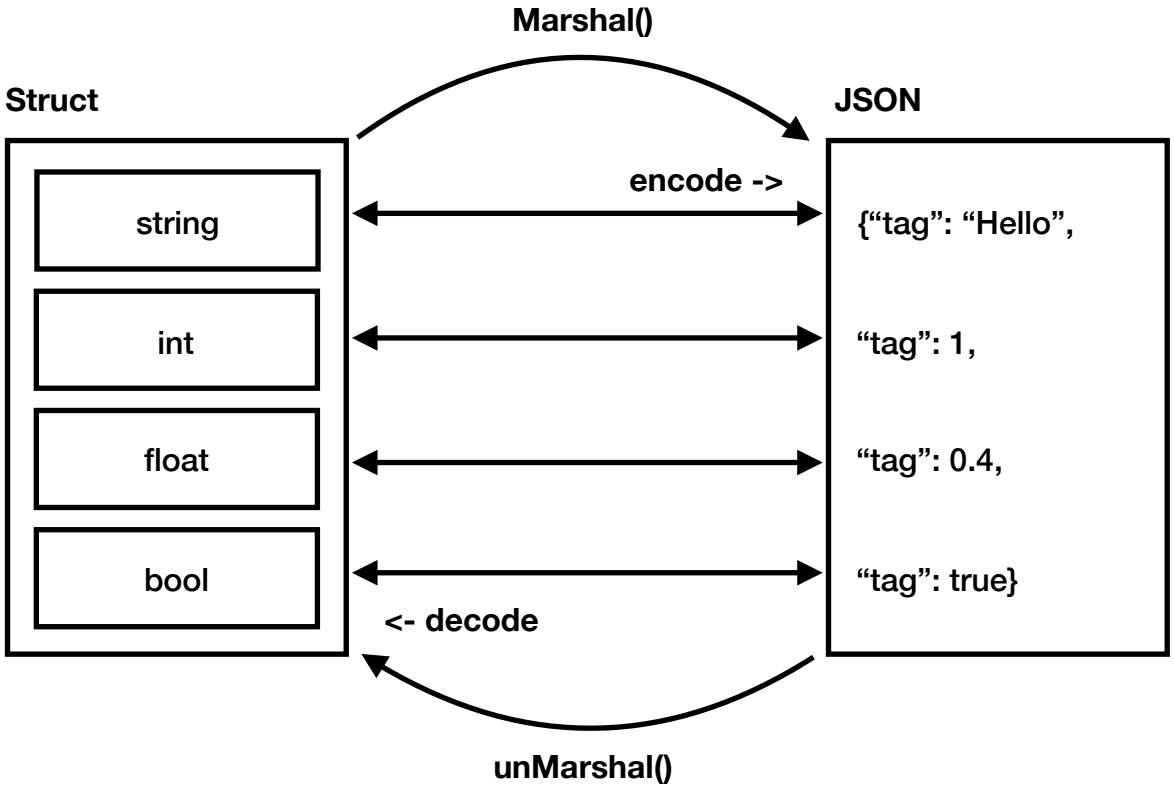
Struct 임베디드 필드

```
type Person struct {
    FirstName string
    LastName  string
    int
}

func main(ㅁ{
    var h1 Person = Person{FirstName:"Hyejong", LastName:"Hong",int:29}
    h1.int = 30
    fmt.Println(h1)
}
```

Struct 태그

```
type T struct {
    f1    string    "f one"
    f2, f3 int64    `f four and five`
    f4 string       `one:"1"`
}
func main() {
    t := reflect.TypeOf(T{})
    f1, _ := t.FieldByName("f1")
    fmt.Println(f1.Tag)
    f2, _ := t.FieldByName("f2")
    f3, _ := t.FieldByName("f3")
    fmt.Println(f2.Tag, f3.Tag)
    f4, _ := t.FieldByName("f4")
    fmt.Println(f4.Tag)
    v, ok := f4.Tag.Lookup("one")
    fmt.Printf("%s, %t\n", v, ok)
}
```

Struct 태그


Marshal()
Struct
JSON
encode ->
string
{“tag”: “Hello”,
int
“tag”: 1,
float
“tag”: 0.4,
bool
“tag”: true}
<- decode
unMarshal()

Struct JSON 인코딩

```
type T struct {
    F1 int `json:"f1"`
    F2 int `json:"F2"`
    F3 int `json:"-"`
}

func main() {
    t := T{1, 0,  3}
    b, err := json.Marshal(t)
    if err != nil {
        panic(err)
    }
    fmt.Printf("%s\n", b) // {"f_1":1,"f_3":2}
}
```

Struct JSON 디코딩

```
type T struct {
    F1 int `json:"f1"`
    F2 int `json:"F2"`
    F3 int `json:"-"`
}

func main() {
    b := []byte(`{"f1":1,"F2":2, "F3":3}`)
    t := T{}
    if err := json.Unmarshal(b, &t); err != nil {
        panic(err)
    }
    fmt.Println(t) // {1 2 0}
}
```

Go 에는

상속이 없고

서브 클래스도 없고

계층 구조도 없습니다.

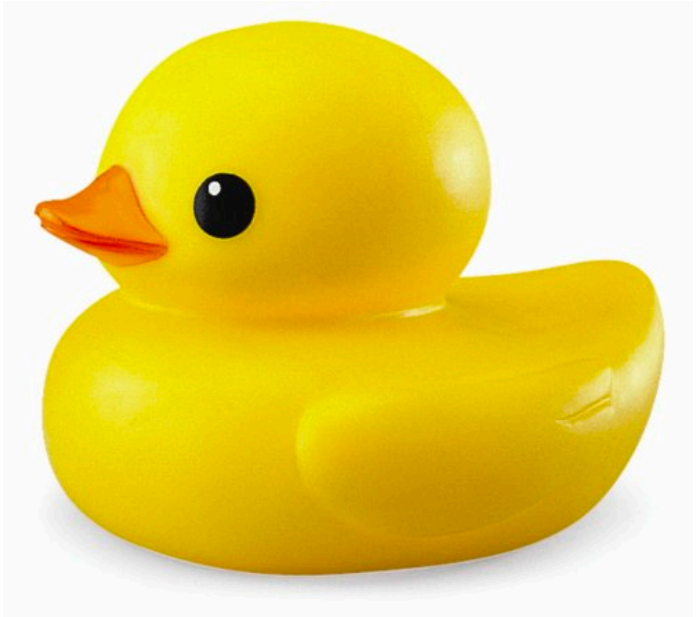
그럼 추상화는

어떻게 하죠?

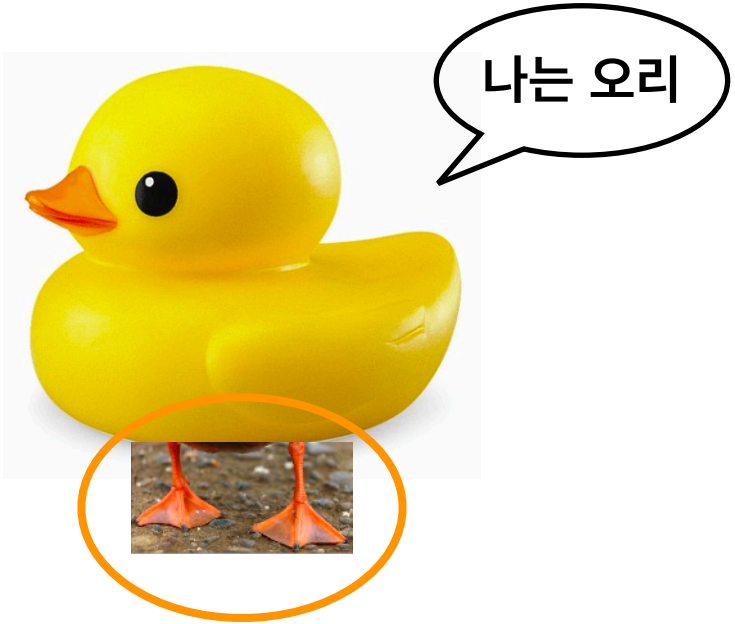
Interface

유저가 구현 해야하는
메서드 원형을 정의한다.

해당 타입이
메서드를 구현한다.



이런 걸 **덕타이핑**라고 부릅니다.



오리 발이라는 메서드 원형을 정의하고
다른 타입이 오리발을 구현한다면...

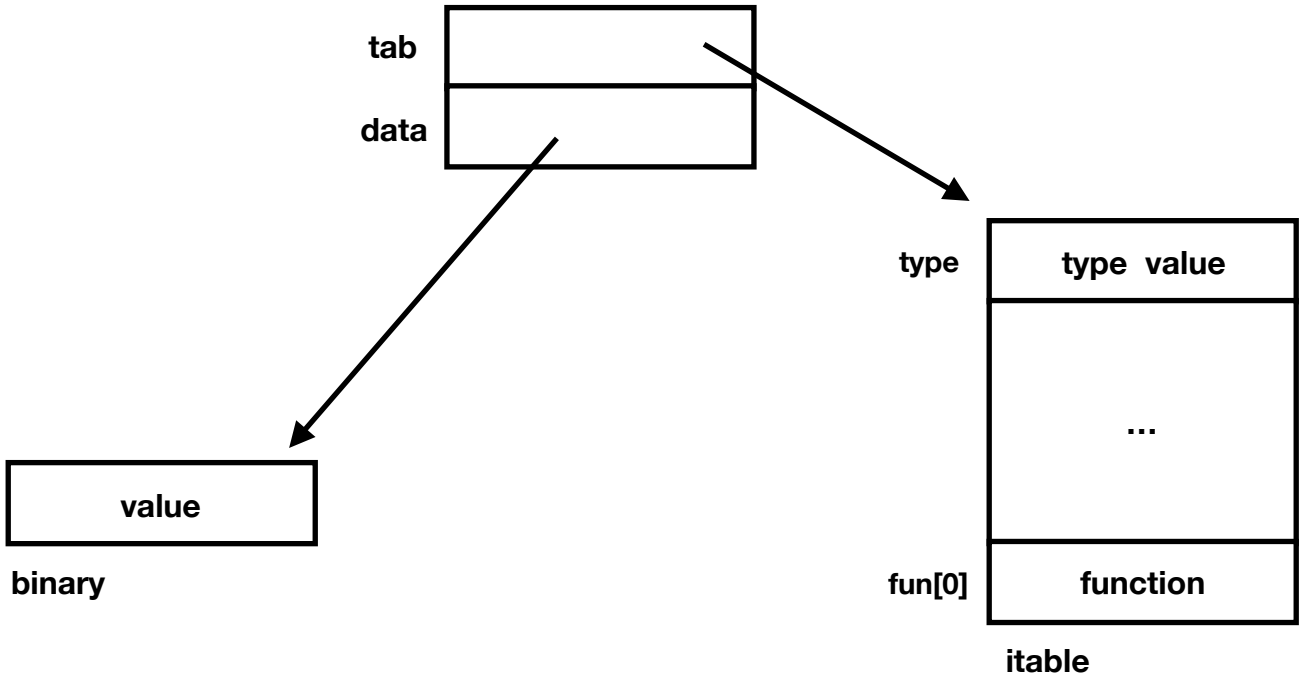
Interface

너도 오리야 ^^

Interface

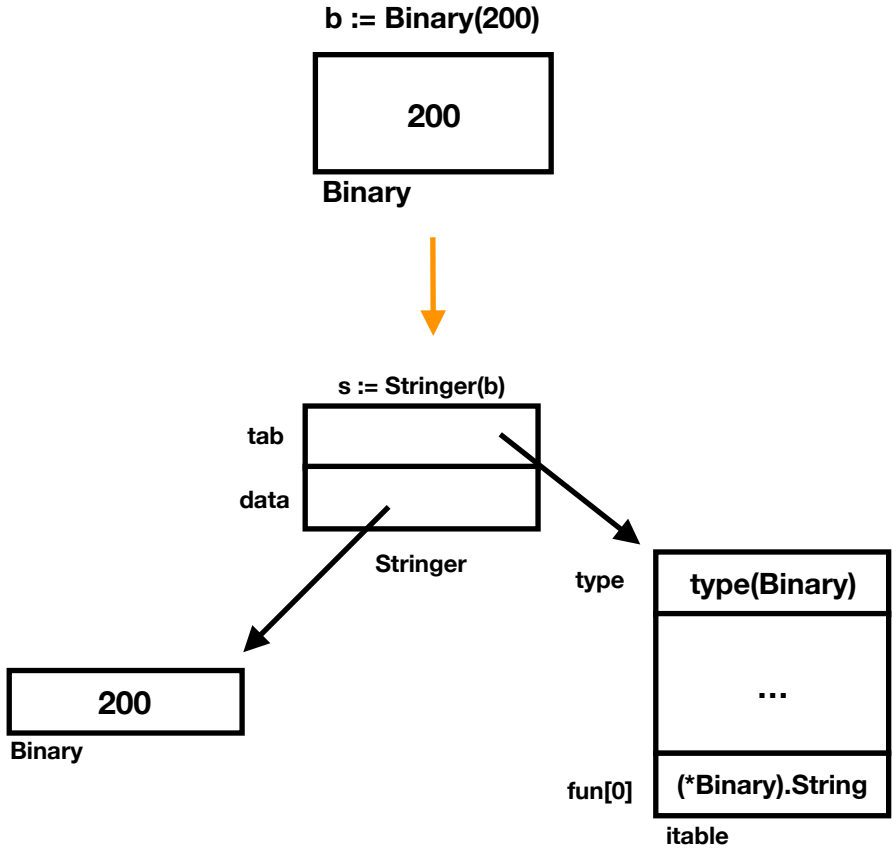
```
type Stringer interface {
    String() string
}

type Binary uint64

func (i Binary) String() string {
    return strconv.FormatUint(uint64(i), 2)
}

func main() {
    b := Binary(200)
    s := Stringer(b)
    fmt.Println(s.String())
}
```

Interface의 구조



Interface의 구조

```
func main() {
    b := Binary(200)
    s := Stringer(b)
    fmt.Println(s.String())
}
```

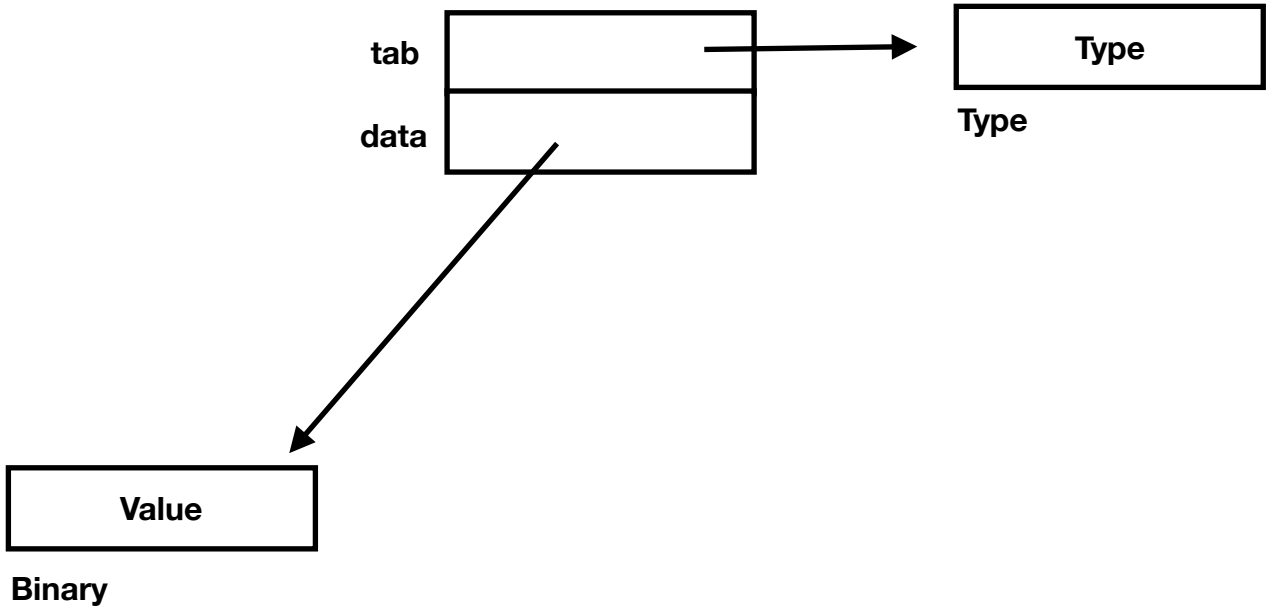


+ interface는 타입이다.

Interface는 타입이다.

```
package main

import "fmt"

func main() {
    var i interface{}
    fmt.Println(i)

    i = 5
    fmt.Println(i)
    i = 1.6
    fmt.Println(i)
    i = "interface"
    fmt.Println(i)
}
```

Interface는 타입이다.



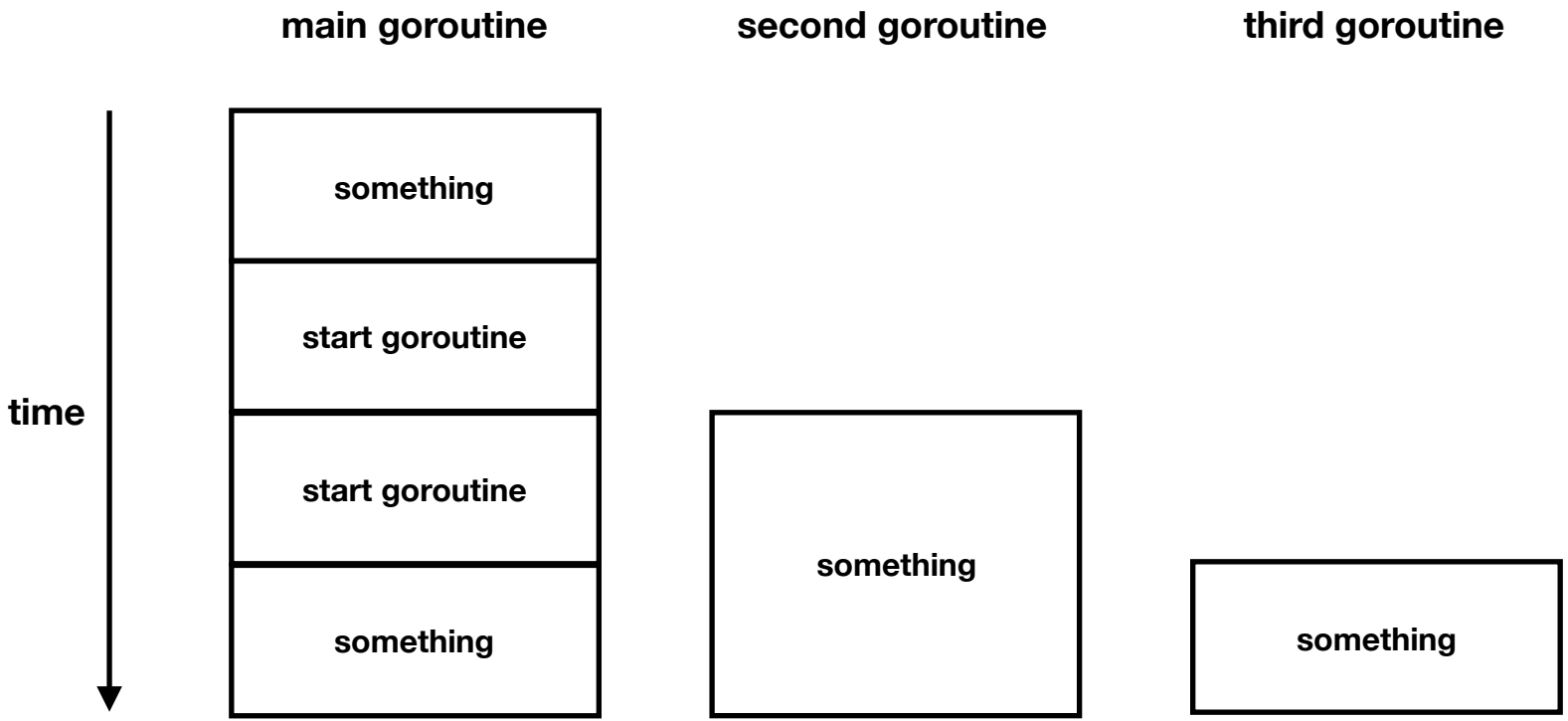
SECTION FIVE

Go의 Concurrency

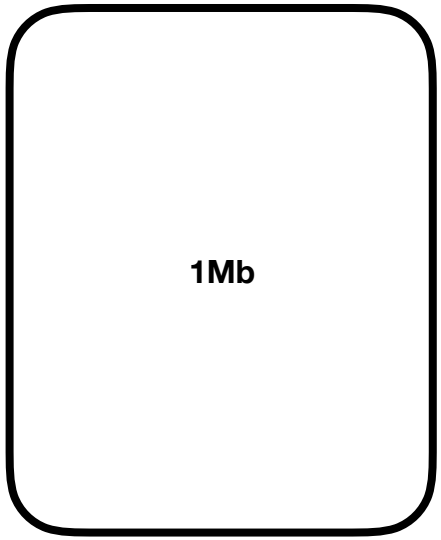
동시성(Concurrency)

- goroutine
- channel

고루틴이란


main goroutine
second goroutine
third goroutine
time
something
start goroutine
start goroutine
something
something
something

그린 쓰레드(lightweight thread)



Thread

goroutine

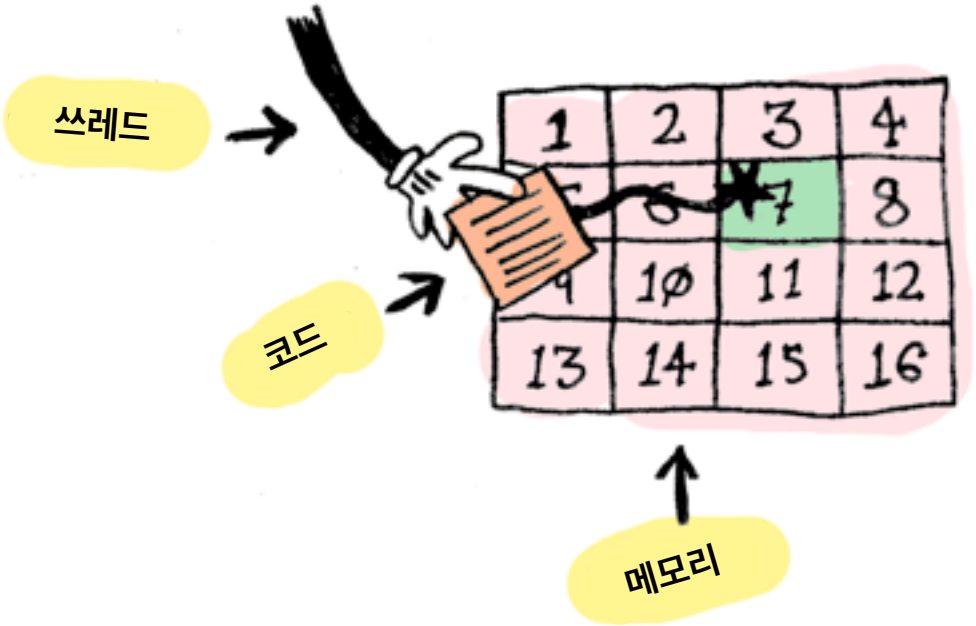
goroutine 예제

```
func f(from string) {
    for i := 0; i < 3; i++ {
        fmt.Println(from, ":", i)
    }
}

func main() {

    f("direct")
    go f("goroutine")
    go func(msg string) {
        fmt.Println(msg)
    }("going")

    fmt.Scanln()
    fmt.Println("done")
}
```

Concurrency problem

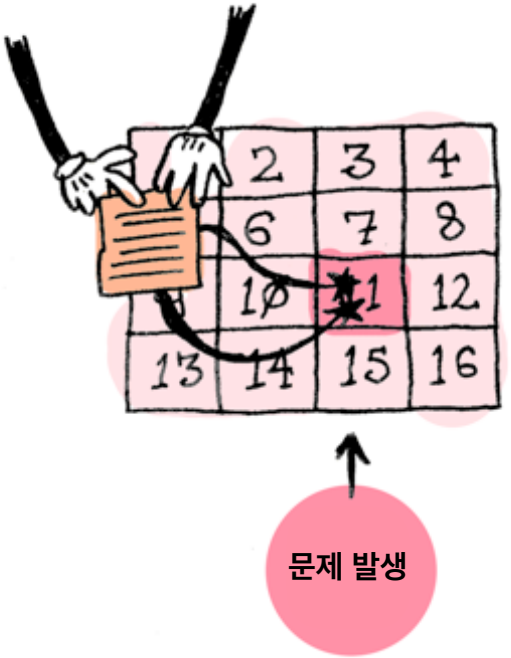
http://adit.io/posts/2013-05-15-Locks,-Actors,-And-STM-In-Pictures.html

Concurrency problem



http://adit.io/posts/2013-05-15-Locks,-Actors,-And-STM-In-Pictures.html

Concurrency problem



http://adit.io/posts/2013-05-15-Locks,-Actors,-And-STM-In-Pictures.html