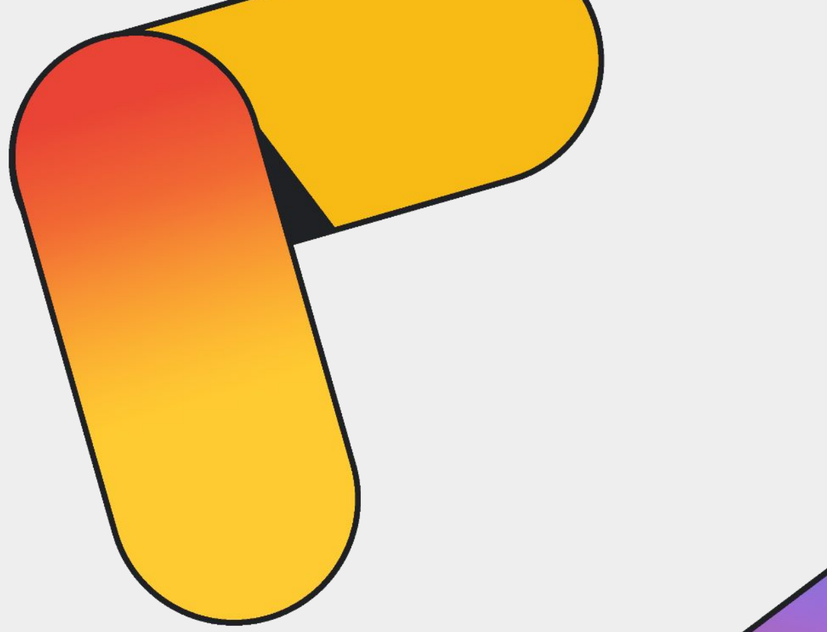
Google I/O Extended

Boost performance of Go applications with profile guided optimization

GDG Golang Korea



Google I/O Extended

Table of Contents

Google I/O Extended

Section 1

What is Go?

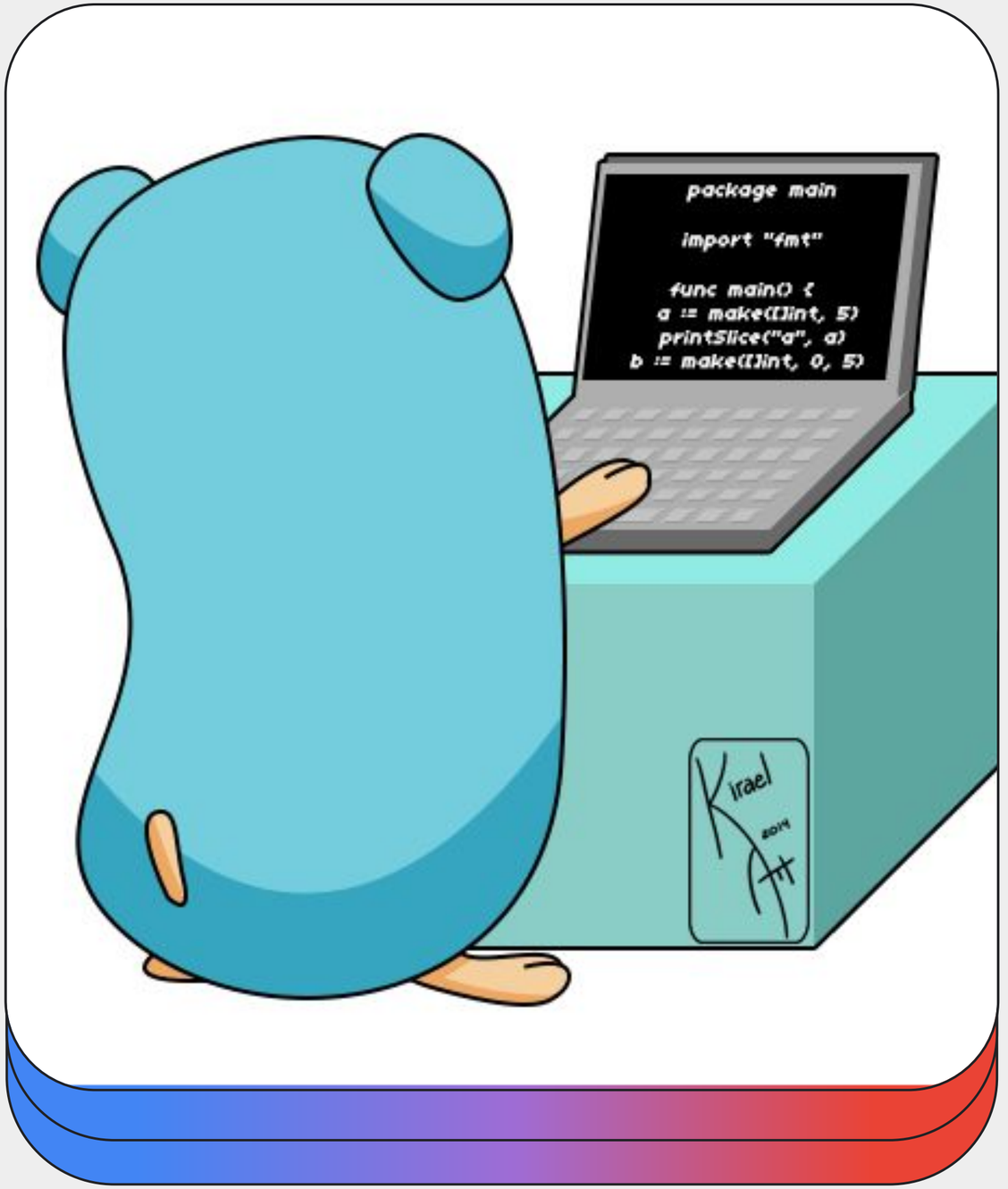
Go언어란?

Section 1

What is Go?

2009년 11월에 구글에서 처음 발표된 후 2012년 3월에 정식 발표된 프로그래밍 언어입니다.
로버트 그리즈머, 롭 파이크, 케네스 톰슨이 디자인하였고, 2007년 경에 이들 세 사람이 새로운 언어에 대한 스케치를 하면서 프로젝트가 시작되었다고 합니다. 세 사람 다 C++의 복잡함이 싫어서 Go를 만들었다고 합니다.
지금도 패키지에 무엇을 포함할지는 이 세 사람이 만장일치로 합의해야 이뤄진다고 합니다.

Go(Golang)는 Google에서 개발한 오픈 소스 프로그래밍 언어입니다. Go는 간결하고 효율적인 문법, 동시성 지원,
빠른 컴파일 속도 등을 특징으로 하며, 시스템 프로그래밍부터 웹 개발까지 다양한 분야에서 사용됩니다.



Google I/O Extended

간결하고 읽기 쉬운 문법

Google I/O Extended

Goroutine(고루틴)

Google I/O Extended

가비지 컬렉션(GC)

Google I/O Extended

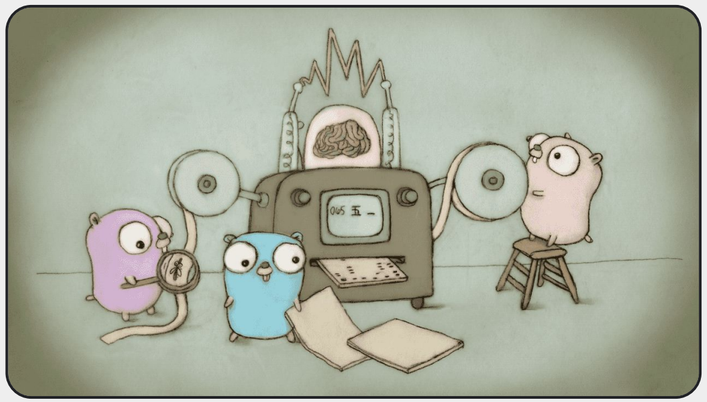
크로스 플랫폼 지원

Section 1

Go is productive platform for building production systems

Go is productive

Go는 배우기 쉽고 유지 관리가 용이하며, 읽기 쉬우며, 팀, 작업량 및 사용 사례에 걸쳐 확장 가능합니다.



Go is a platform

Go는 단순한 언어가 아니라, 개발 생애 주기 전반에 걸쳐 완전한 개발자 경험을 제공합니다.

Go is production ready

Go는 더 신뢰할 수 있고, 효율적이며, 안정적이고, 보안성이 뛰어나 중요한 비즈니스 시스템과 인프라에 적합합니다.

Section 1

Go’s developer platform provides end-to-end solutions out of the box

Google I/O Extended

Section 2

PGO

Profile Guided Optimization

Section 2

PGO란?

프로파일 가이드 최적화(Profile Guided Optimization, PGO)는 애플리케이션의 성능을 향상시키기 위해 실행 중에 수집된 프로파일 데이터를 사용하는 최적화 기법

Section 2

PGO의 이점

성능 향상

실행 시간 단축: 코드의 핫스팟(Hotspot)을 집중적으로 최적화해 중요한 코드 경로의 실행을 빠르게 합니다.

메모리 사용 최적화: 자주 사용되지 않는 코드는 덜 최적화하여 메모리와 CPU 사용 효율성을 높입니다.

리소스 효율성

에너지 절약: 최적화된 코드로 CPU 사이클을 절약하여 전력 소비를 줄입니다.

하드웨어 비용 절감: 효율적인 리소스 활용으로 하드웨어 업그레이드 필요성을 감소시킵니다.

Section 2

컴파일러

컴파일러(compiler)는 프로그래밍 언어로 작성된 소스 코드를 컴퓨터가 이해할 수 있는 기계어 또는 바이트코드로 번역하는 소프트웨어입니다.
소스 코드는 사람이 읽을 수 있는 고급 프로그래밍 언어로 작성되며,
컴파일러는 이를 기계가 실행할 수 있는 저수준 언어로 변환함

Section 2

Standard Compilation flow in Go

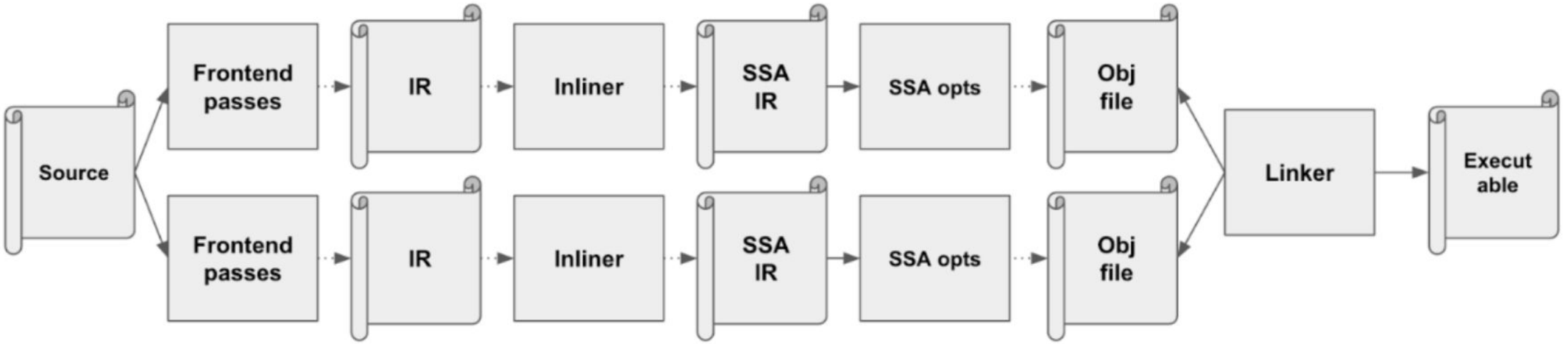


Figure 1. Go compiler.

Section 2

Go

```go
func Add(a, b int) int {
   return a + b
}

func main() {
   c = Add(1, 2)
}
```

Assembly

```
TEXT main.Add(SB)
   ADDQ BX, AX
   RET
```

```
TEXT main.main(SB)
   MOVL $0x1, AX
   MOVL $0x2, BX
   CALL main.Add(SB)
   MOVQ AX, main.c(SB)
```

Section 2

Inlining

```
func Add(a, b int) int {
   return a + b
}

func main() {
   c = Add(1, 2)
}
```

```
func Add(a, b int) int {
   return a + b
}

func main() {
   a := 1
   b := 2
   c = a + b
}
```

Section 2

Constant Propagation

```
func Add(a, b int) int {
   return a + b
}

func main() {
   a := 1
   b := 2
   c = a + b
}
```

```
func Add(a, b int) int {
   return a + b
}

func main() {
   c = 1 + 2
}
```

Section 2

Constant folding

```
func Add(a, b int) int {
   return a + b
}

func main() {
   c = 1 + 2
}
```

```
func Add(a, b int) int {
   return a + b
}

func main() {
   c = 3
}
```

Section 2

```
TEXT main.Add(SB)
   ADDQ BX, AX
   RET
```

```
TEXT main.main(SB)
   MOVL $0x1, AX
   MOVL $0x2, BX
   CALL main.Add(SB)
   MOVQ AX, main.c(SB)
```

```
TEXT main.Add(SB)
   ADDQ BX, AX
   RET

TEXT main.main(SB)
   MOVQ $0x3, main.c(SB)
```

Section 2

Golang에서의 프로파일링 방법

pprof 패키지

pprof는 Go 언어에서 제공하는 기본적인 프로파일링 도구입니다. CPU 프로파일링, 메모리 프로파일링, goroutine 덤프 등을 지원

runtime/trace 패키지

runtime/trace 패키지를 사용하면 더 자세한 실행 추적 정보를 얻을 수 있음

Benchmarking

벤치마킹을 통해서도 프로파일링을 할 수 있습니다. 벤치마크 테스트를 작성한 후 -cpuprofile, -memprofile 플래그를 사용하여 프로파일링할 수 있음

Section 2

pprof

pprof는 Go에서 성능 프로파일링을 위한 패키지로,
Go 애플리케이션의 CPU, 메모리, goroutine, 힙 사용 등을
분석할 수 있는 도구

https://pkg.go.dev/net/http/pprof

https://pkg.go.dev/runtime/pprof

Section 2

pprof

net/http/pprof

이 패키지는 HTTP 인터페이스를 통해 실시간으로 프로파일링 데이터를 수집할 수 있도록 합니다. 주로 애플리케이션에 간단히 임포트하여 HTTP 서버를 통해 프로파일 데이터를 노출시키는 방식으로 사용됨

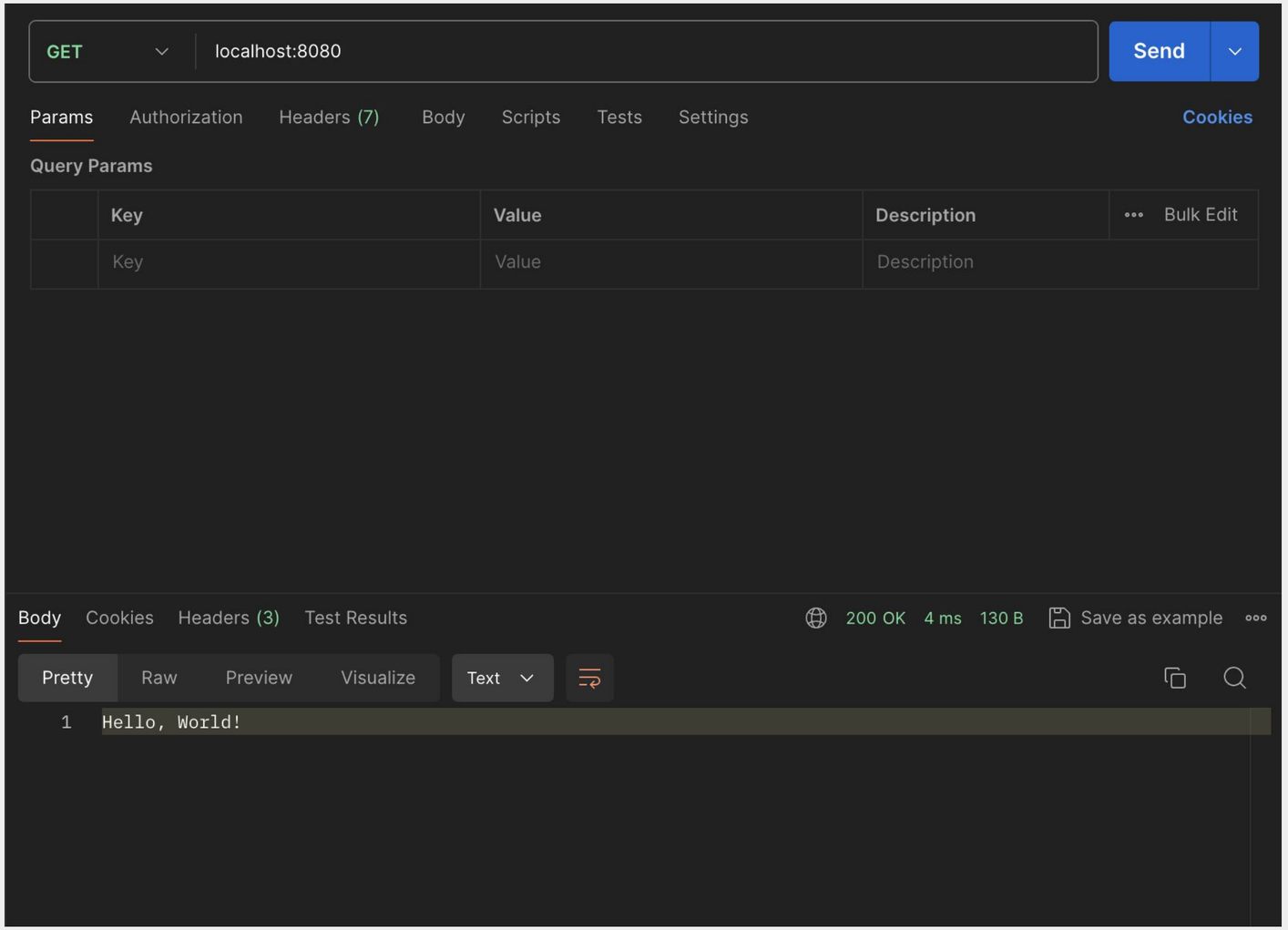
runtime/pprof

이 패키지는 코드 내에서 직접 프로파일링 데이터를 수집하고 제어할 수 있는 인터페이스를 제공

Section 2

```go
package main

import (
    "log"
    "net/http"
    _ "net/http/pprof"
)

func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        w.Write([]byte("Hello, World!"))
    })

    go func() {
        log.Println(http.ListenAndServe("localhost:6060", nil))
    }()

    log.Println("Starting server on :8080")

    if err := http.ListenAndServe(":8080", nil); err != nil {
        log.Fatalf("could not start server: %v\n", err)
    }
}
```

Section 2

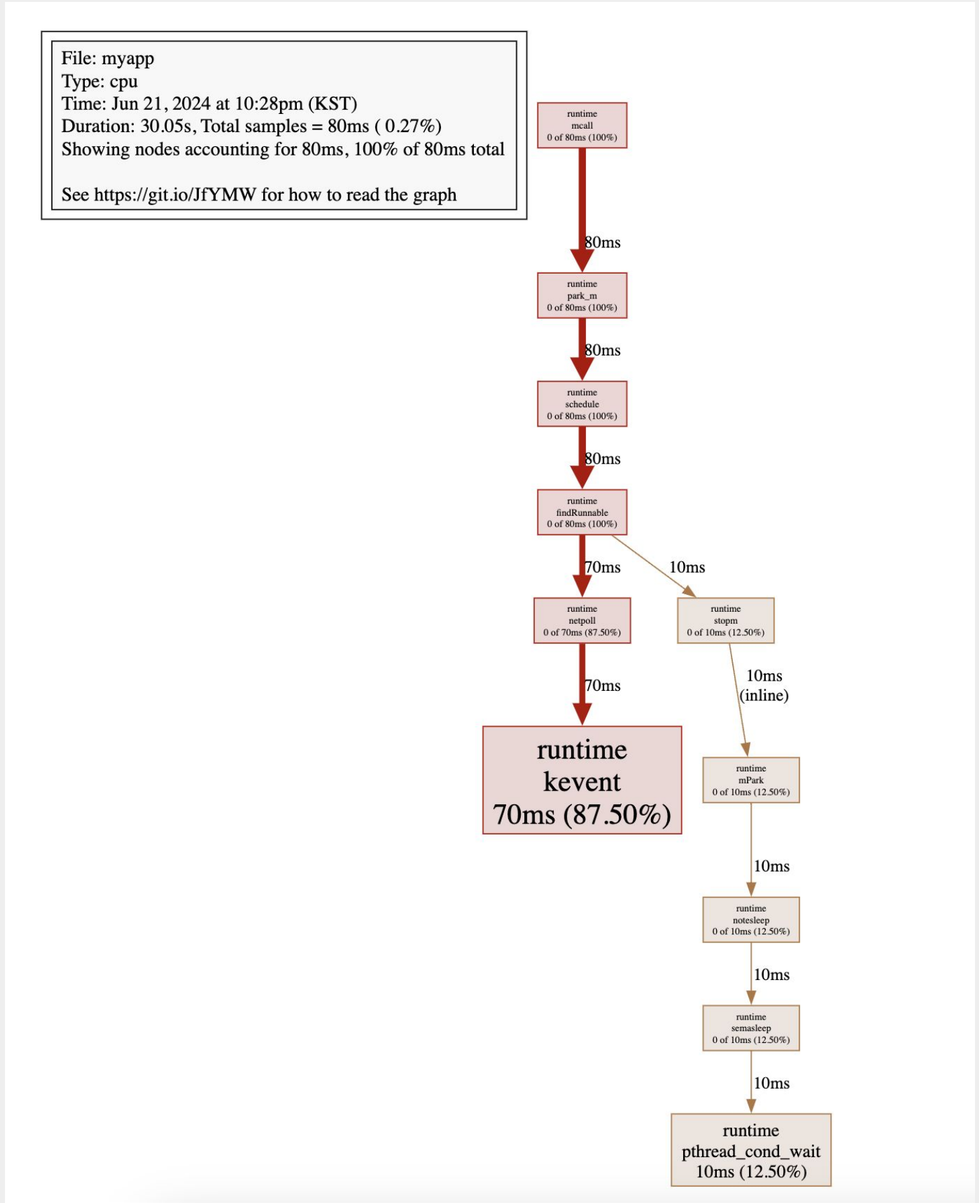

GET
localhost:8080
Send
Params
Authorization
Headers (7)
Body
Scripts
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (3)
Test Results
200 OK
4 ms
130 B
Save as example
Pretty
Raw
Preview
Visualize
Text
1
Hello, World!

Section 2

```
go tool pprof http://localhost:6060/debug/pprof/profile\?seconds\=30

Fetching profile over HTTP from http://localhost:6060/debug/pprof/profile?seconds=30
Saved profile in /Users/hyunseojung/pprof/pprof.myapp.samples.cpu.001.pb.gz
```

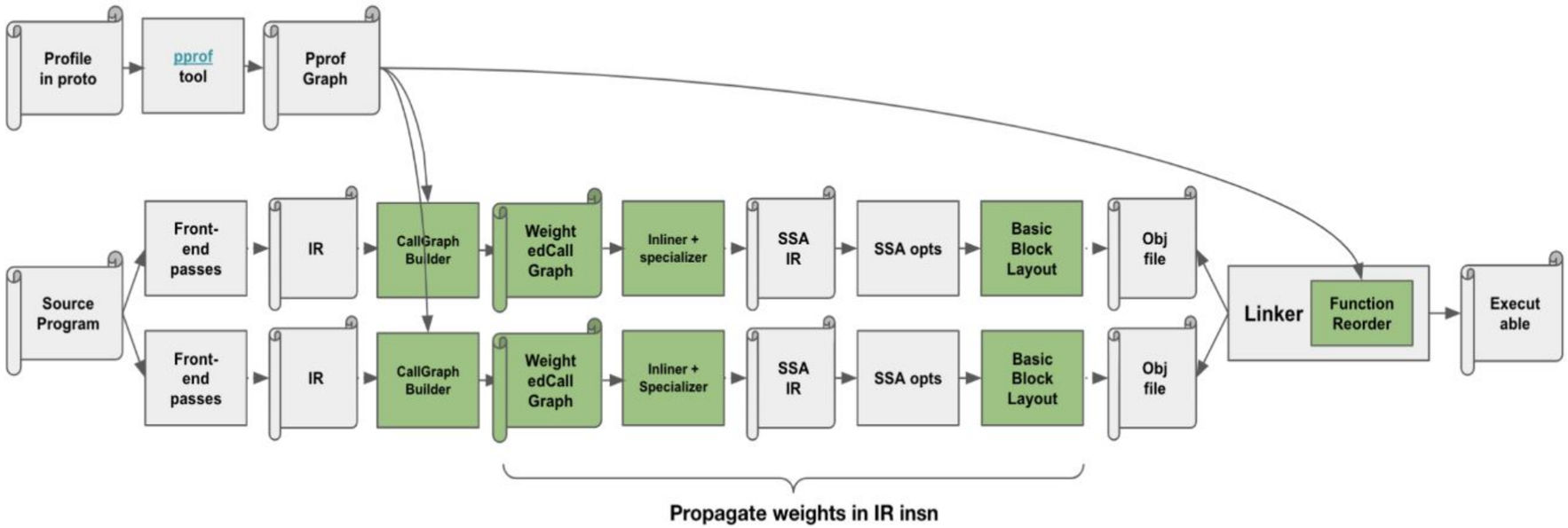
```
File: myapp
Type: cpu
Time: Jun 21, 2024 at 10:28pm (KST)
Duration: 30.05s, Total samples = 80ms ( 0.27%)
Entering interactive mode (type "help" for commands, "o" for options)
```

Section 2


File: myapp
Type: cpu
Time: Jun 21, 2024 at 10:28pm (KST)
Duration: 30.05s, Total samples = 80ms (0.27%)
Showing nodes accounting for 80ms, 100% of 80ms total
See https://git.io/JfYMW for how to read the graph
runtime
mcall
0 of 80ms (100%)
80ms
runtime
park_m
0 of 80ms (100%)
80ms
runtime
schedule
0 of 80ms (100%)
80ms
runtime
findRunnable
0 of 80ms (100%)
70ms
10ms
runtime
netpoll
0 of 70ms (87.50%)
runtime
stopm
0 of 10ms (12.50%)
70ms
10ms
(inline)
runtime
kevent
70ms (87.50%)
runtime
mPark
0 of 10ms (12.50%)
10ms
runtime
notesleep
0 of 10ms (12.50%)
10ms
runtime
semasleep
0 of 10ms (12.50%)
10ms
runtime
pthread_cond_wait
10ms (12.50%)

Section 2

New compilation flow proposed in Go for PGO (Fig-2):



Section 2

```go
package main

import (
    "fmt"
    "log"
    "math/big"
    "net/http"
    _ "net/http/pprof"
)

tabnine: test | explain | document | ask
func fibonacci(n int) *big.Int {
    if n < 2 {
        return big.NewInt(int64(n))
    }
    a, b := big.NewInt(0), big.NewInt(1)
    for i := 2; i <= n; i++ {
        a.Add(a, b)
        a, b = b, a
    }
    return b
}

tabnine: test | explain | document | ask
func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        n := 50000
        result := fibonacci(n)
        w.Write([]byte(fmt.Sprintf("Fibonacci(%d) = %s", n, result.String())))
    })

    go func() {
        log.Println(http.ListenAndServe("localhost:6060", nil))
    }()

    log.Println("Starting server on :8080")
    if err := http.ListenAndServe(":8080", nil); err != nil {
        log.Fatalf("could not start server: %v\n", err)
    }
}
```

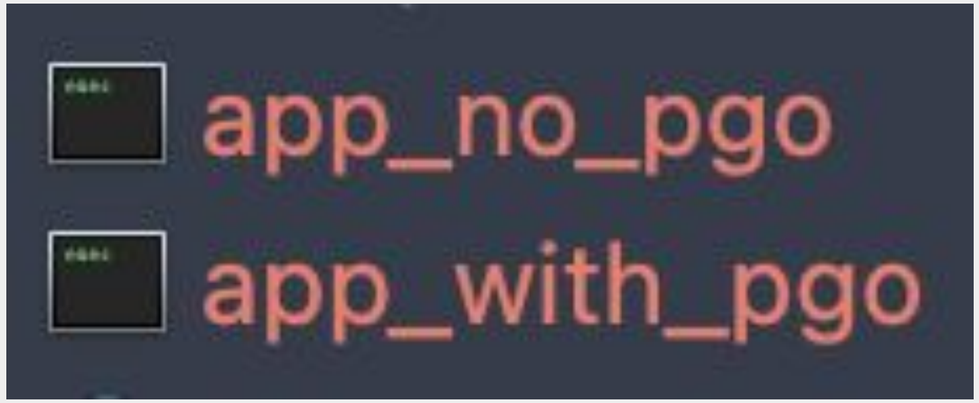
Section 2

```
go run myapp.go
go tool pprof -proto -output=profile.pb.gz http://localhost:6060/debug/pprof/profile?seconds=30
```

Section 2

```
go build -o app_with_pgo -pgo=profile.pb.gz myapp.go
```

Section 2



Section 2

```
./app_no_pgo &
wrk -t12 -c400 -d30s http://localhost:8080/
pkill app_no_pgo
```

```
Running 30s test @ http://localhost:8080/
  12 threads and 400 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   349.37ms  381.88ms   1.99s    83.86%
    Req/Sec   121.49     55.08   360.00     69.40%
  43586 requests in 30.08s, 440.90MB read
  Socket errors: connect 0, read 475, write 0, timeout 161
Requests/sec:   1448.90
Transfer/sec:     14.66MB
```

Section 2

```
./app_with_pgo &
wrk -t12 -c400 -d30s http://localhost:8080/
pkill app_with_pgo
```

```
Running 30s test @ http://localhost:8080/
  12 threads and 400 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   342.74ms  371.21ms   1.99s    84.53%
    Req/Sec   123.92     52.27   340.00     65.97%
  44453 requests in 30.05s, 449.67MB read
  Socket errors: connect 0, read 576, write 11, timeout 122
Requests/sec:   1479.08
Transfer/sec:     14.96MB
```

Section 2

```
Running 30s test @ http://localhost:8080/
  12 threads and 400 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   349.37ms  381.88ms   1.99s    83.86%
    Req/Sec   121.49     55.08   360.00     69.40%
  43586 requests in 30.08s, 440.90MB read
  Socket errors: connect 0, read 475, write 0, timeout 161
Requests/sec:   1448.90
Transfer/sec:     14.66MB
```

```
Running 30s test @ http://localhost:8080/
  12 threads and 400 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   342.74ms  371.21ms   1.99s    84.53%
    Req/Sec   123.92     52.27   340.00     65.97%
  44453 requests in 30.05s, 449.67MB read
  Socket errors: connect 0, read 576, write 11, timeout 122
Requests/sec:   1479.08
Transfer/sec:     14.96MB
```

Google I/O Extended

Q&A

자유롭게 질문 해주세요!

Google I/O Extended

Thank You

Hyunseo Jung organizer

GDG Golang Korea