Guide to Profile-Guided Optimization:

inlining, devirtualizing, and profiling

Satoru Kawahara

@Go Conference 2024

Who talks

Hello, I’m bisho-jo!

- 🏢 **Kanmu,Inc.**
- 📎 Nymphium
- 🔰 2nd-year Golang student

Today's Topic

By the way, do you ...

Today's Topic

By the way, do you ...

 Use the Go Compiler?

Today's Topic

By the way, do you ...

- 🙋‍♀️ Use the Go Compiler?
- 🙋‍♀️ Use its Optimizations?

Today's Topic

By the way, do you ...

- 🙋‍♀️ Use the Go Compiler?
- 🙋‍♀️ Use its Optimizations?

Then,

Today's Topic

By the way, do you ...

- 🙋‍♀️ Use the Go Compiler?
- 🙋‍♀️ Use its Optimizations?

Then,

Do you use

Profile-Guided Optimization?

Today’s Topic

Today’s Topic

Learn about

Today's Topic

Learn about

Profile-Guided Optimization

Today's Topic

Learn about

Profile-Guided Optimization

and

its associated optimizations

Today's Topic

Profile-Guided Optimization (**PGO**, abbrev.)
is an optimization method that[*1]:

[*1] `https://go.dev/doc/pgo`

Today's Topic

Profile-Guided Optimization (**PGO**, abbrev.) is an optimization method that[*1]:

- Uses *profiling information* from **program execution**

[*1] `https://go.dev/doc/pgo`

Today's Topic

Profile-Guided Optimization (**PGO**, abbrev.) is an optimization method that[*1]:

- Uses *profiling information* from **program execution**
- Enables *more aggressive optimizations*, such as inlining and devirtualization

[*1] `https://go.dev/doc/pgo`

Today's Topic

Profile-Guided Optimization (**PGO**, abbrev.) is an optimization method that[*1]:

- Uses *profiling information* from **program execution**
- Enables *more aggressive optimizations*, such as **inlining** and **devirtualization**

[*1] `https://go.dev/doc/pgo`

Today's Topic

Learn about

Profile-Guided Optimization

and

its associated optimizations

Today's Topic

Learn about

Profile-Guided Optimization

and

👉 **its associated optimizations**

- 🤔 *Function Inlining*
- 🤔 *Devirtualization*

Function Inlining

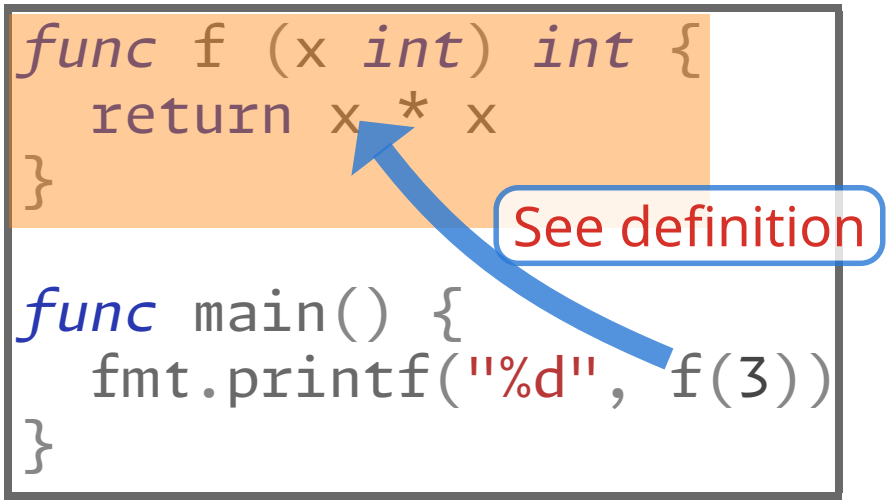
Function inlining, or simply *inlining*, is the process of replacing a func call with its body.

```
func f (x int) int {
  return x * x
}

func main() {
  fmt.printf("%d", f(3))
}
```

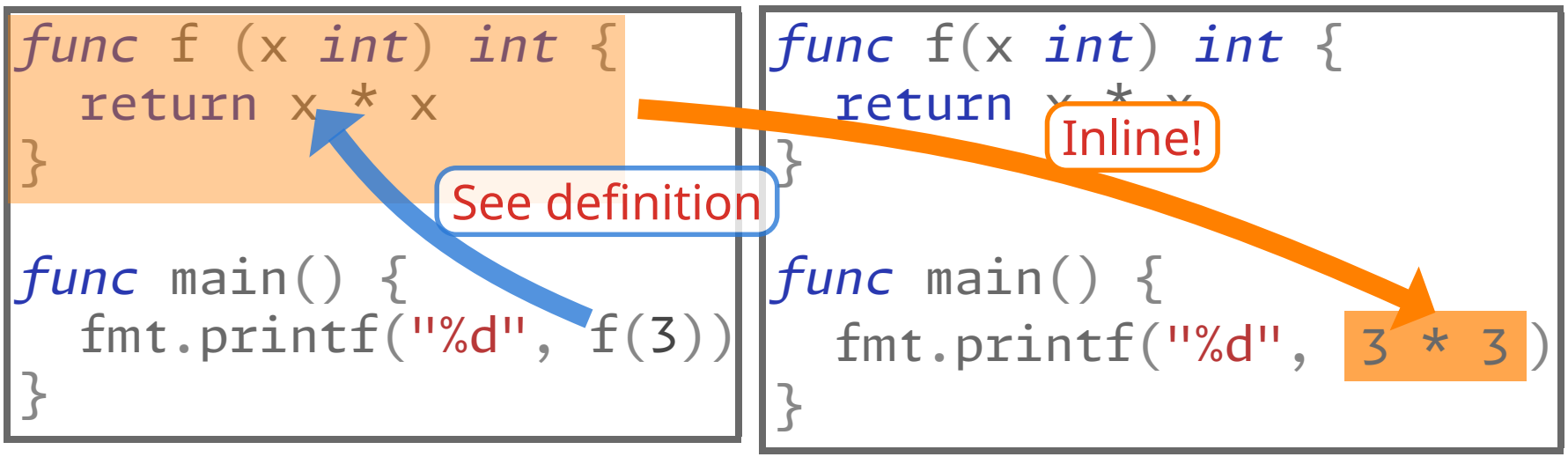
Function Inlining

Function inlining, or simply *inlining*, is the process of replacing a func call with its body.



Function Inlining

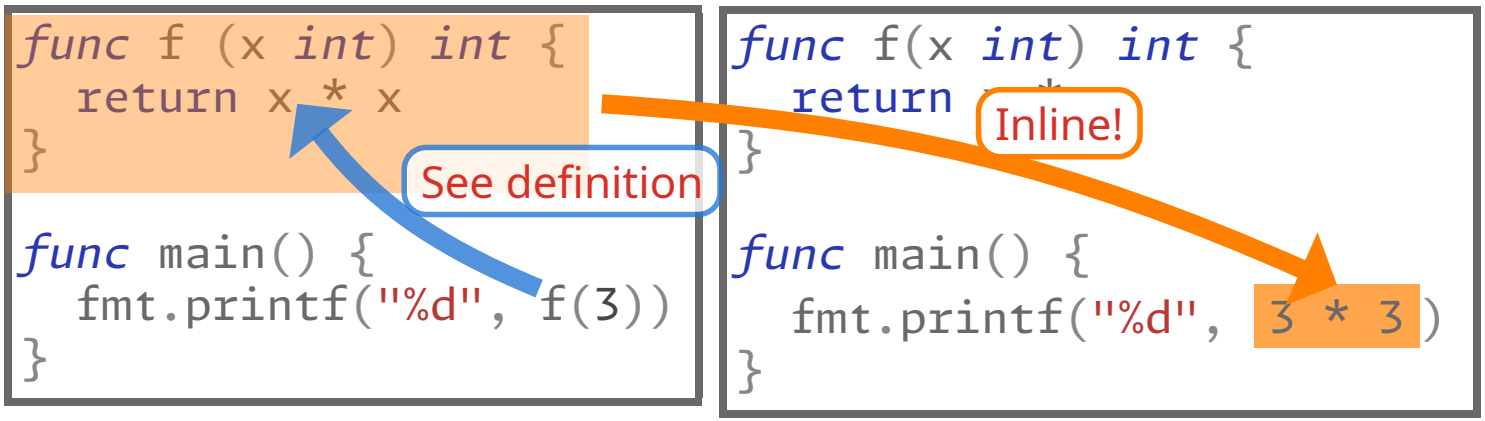
Function inlining, or simply *inlining*, is the process of replacing a func call with its body.



Function Inlining

Function inlining, or simply *inlining*, is the process of replacing a func call with its body.

- Reduces ***function call overhead:***
  No stack frame setup, no return address,
  no arguments copying
- Enables ***further optimizations***:
  E.g., *constant propagation*, *dead code elimination*



Conditions for Inlining

Several conditionals to be applied[*2]:

[*2] `https://github.com/golang/go/blob/go1.21.0/src/cmd/compile/internal/inline/inl.go`

Conditions for Inlining

Several conditionals to be applied[*2]:

- **Non-leaf** function:

[*2] `https://github.com/golang/go/blob/go1.21.0/src/cmd/compile/internal/inline/inl.go`

●Conditions for Inlining

Several conditionals to be applied[*2]:

- **Non-leaf** function:
  The func shouldn't call other funcs.

```
func f (x int) int {
  return x * x
}

func g (x int) int {
  return f(x) + 1
}
```

[*2] https://github.com/golang/go/blob/go1.21.0/src/cmd/compile/internal/inline/inl.go

●Conditions for Inlining

Several conditionals to be applied[*2]:

- **Non-leaf** function:
  The func shouldn't call other funcs.

```
func f (x int) int {
  return x * x
}

func g (x int) int {
  return f(x) + 1
}
```

calls f → g is **non-leaf function**

[*2] https://github.com/golang/go/blob/go1.21.0/src/cmd/compile/internal/inline/inl.go

Conditions for Inlining

Several conditionals to be applied[*2]:

- **Non-leaf** function:
  The func shouldn't call other funcs.
- **Small function**, *"Budget"* ≤ 80:
  Constructs are rated by their cost:
  - 57 for non-leaf func call
  - 1 for `panic`
  - etc.

  The *budget* is the total cost of func body.

[*2] `https://github.com/golang/go/blob/go1.21.0/src/cmd/compile/internal/inline/inl.go`

Conditions for Inlining

Several conditionals to be applied[*2]:

- **Non-leaf** function:
  The func shouldn't call other funcs.
- **Small function**, *"Budget"* ≤ 80:
  Constructs are rated by their cost:
  - 57 for non-leaf func call
  - 1 for `panic`
  - etc.

  The budget is the total cost of func body.
- And so on $\cdots$
  - Not external function (e.g., C functions)
  - No specific tags set, *`//go:noinline`*, *`//go:systemstack`*, etc.
  - Not a complex body, including `defer`, `select`, etc.

[*2] `https://github.com/golang/go/blob/go1.21.0/src/cmd/compile/internal/inline/inl.go`

Devirtualization

Devirtualization is an optimizations that converts an interface method call into concrete func call.

Devirtualization

Devirtualization is an optimizations that converts an **interface method call** into concrete func call.

WAIT!

Can you explain

how

interface method call

works?

•Interface Method Call

Interface method calls in Go is perforemed based on *dynamic dispatch*[*3].

```
var r io.Reader
r = strings.NewReader("Hello")
buf := make([]byte, 5)
n, _ := r.Read(buf)
```

[*3] https://research.swtch.com/interfaces

•Interface Method Call

Interface method calls in Go is perforemed based on *dynamic dispatch*[*3].

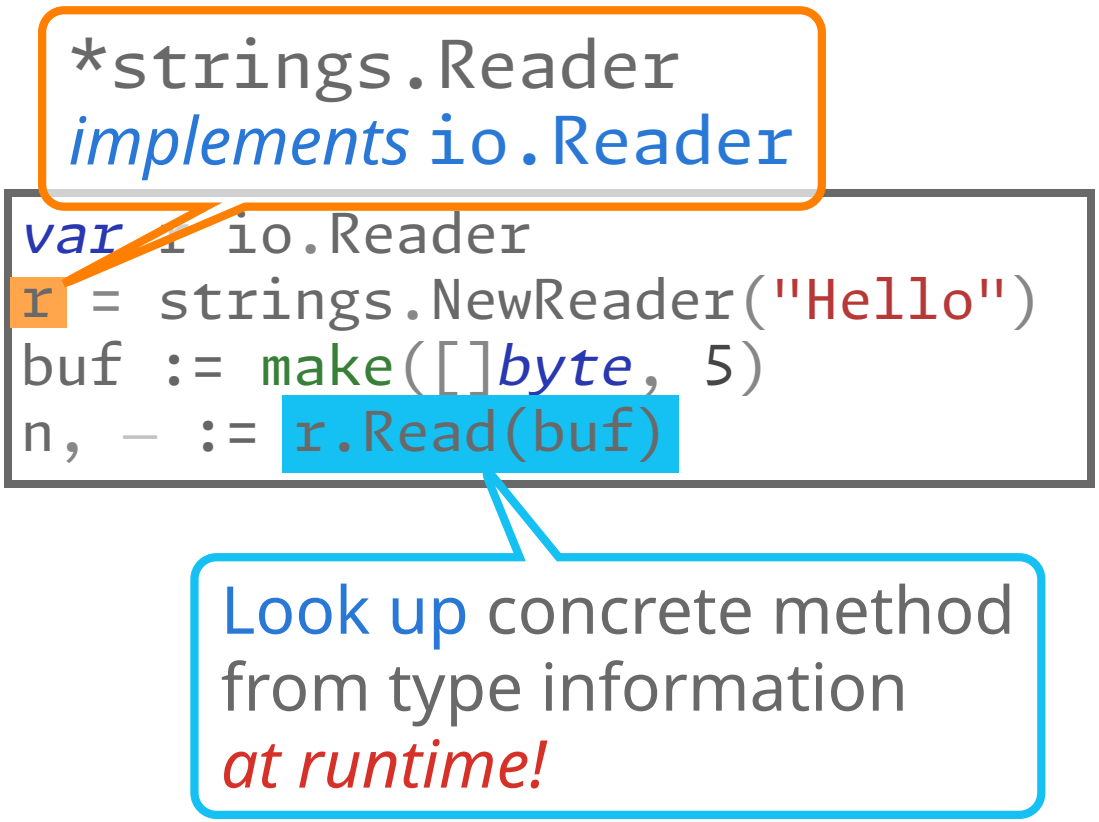
*strings.Reader
implements io.Reader

```
var r io.Reader
r = strings.NewReader("Hello")
buf := make([]byte, 5)
n, _ := r.Read(buf)
```

[*3] https://research.swtch.com/interfaces

•Interface Method Call

Interface method calls in Go is perforemed based on *dynamic dispatch*[*3].



[*3] https://research.swtch.com/interfaces

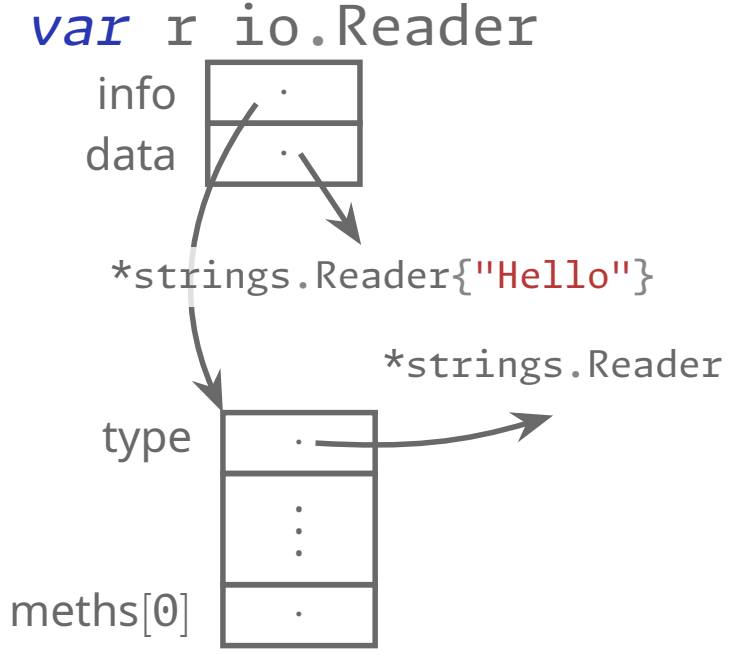
•Interface Method Call

Interface method calls in Go is perforemed based on *dynamic dispatch*[*3].

*strings.Reader
implements io.Reader

```
var r io.Reader
r = strings.NewReader("Hello")
buf := make([]byte, 5)
n, _ := r.Read(buf)
```

Look up concrete method from type information *at runtime!*



[*3] https://research.swtch.com/interfaces

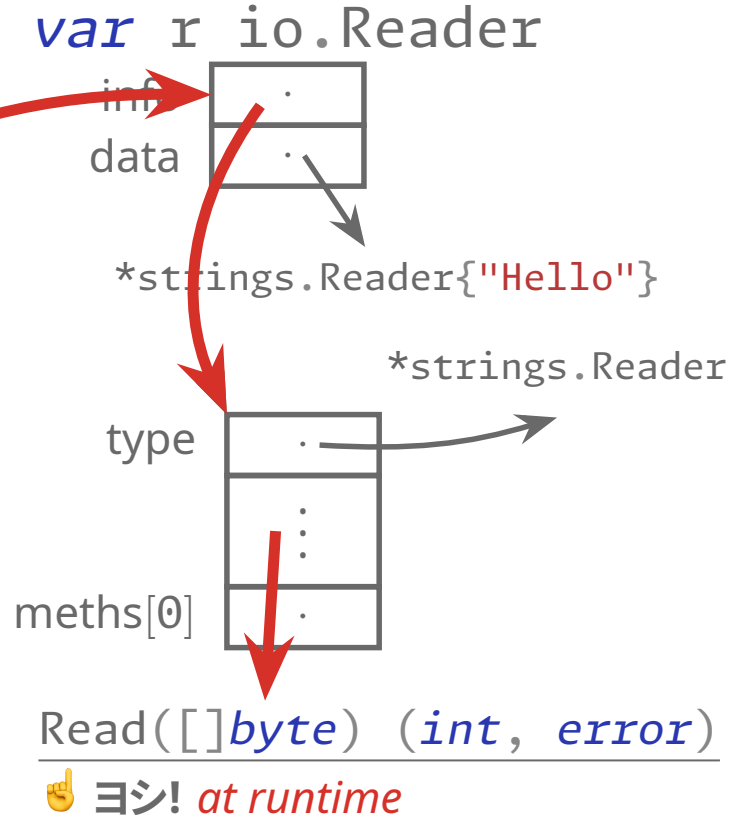
•Interface Method Call

Interface method calls in Go is perforemed based on *dynamic dispatch*[*3].

*strings.Reader *implements* io.Reader

```
var r io.Reader
r = strings.NewReader("Hello")
buf := make([]byte, 5)
n, _ := r.Read(buf)
```

Look up concrete method from type information *at runtime!*



[*3] https://research.swtch.com/interfaces

Devirtualization

Devirtualization is a kind of optimizations that an converts **interface method call** 👍 with concrete func call.

```
var r io.Reader
r = strings.NewReader("Hello")
buf := make([]byte, 5)
n, _ := r.Read(buf)
```

Devirtualization

Devirtualization is a kind of optimizations that an converts **interface method call** 👍 with concrete func call.

```
var r io.Reader
r = strings.NewReader("Hello")
buf := make([]byte, 5)
n, _ := r.Read(buf)
```

Look up concrete method from type information at runtime

Devirtualization

Devirtualization is a kind of optimizations that an converts **interface method call** 👍 with concrete func call.

```
var r io.Reader
r = strings.NewReader("Hello")
buf := make([]byte, 5)
n, _ := r.Read(buf)
```

Look up concrete method from type information **at runtime** ...? 🤔

Devirtualization

Devirtualization is a kind of optimizations that an converts **interface method call** 👍 with concrete func call.

```
var r io.Reader
r = strings.NewReader("Hello")
buf := make([]byte, 5)
n, _ := r.Read(buf)
```

👇👇👇

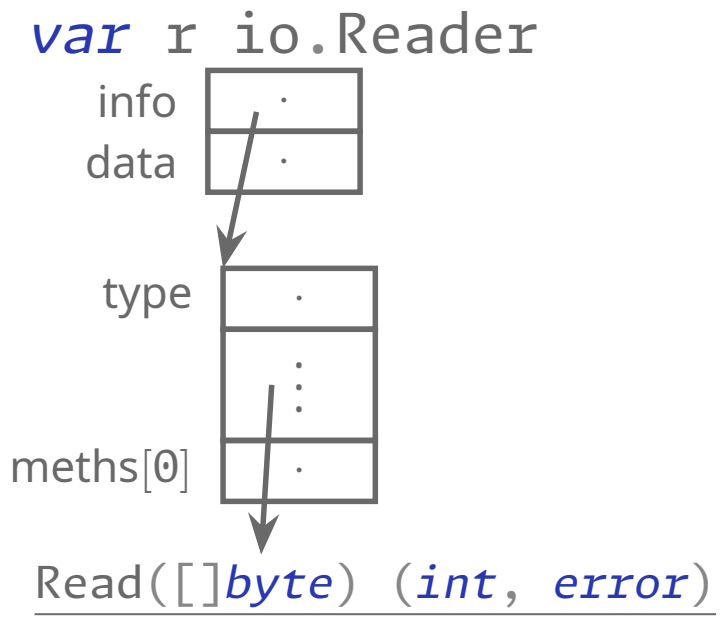
Look up concrete method from type information **at runtime** ...? 🤔

→ Can analyze it calls `(*strings.Reader).Read` at ***compile time!***

Devirtualization

Devirtualization is a kind of optimizations that an converts **interface method call** 👍 with concrete func call.

```
var r io.Reader
r = strings.NewReader("Hello")
buf := make([]byte, 5)
n, _ := r.Read(buf)
```

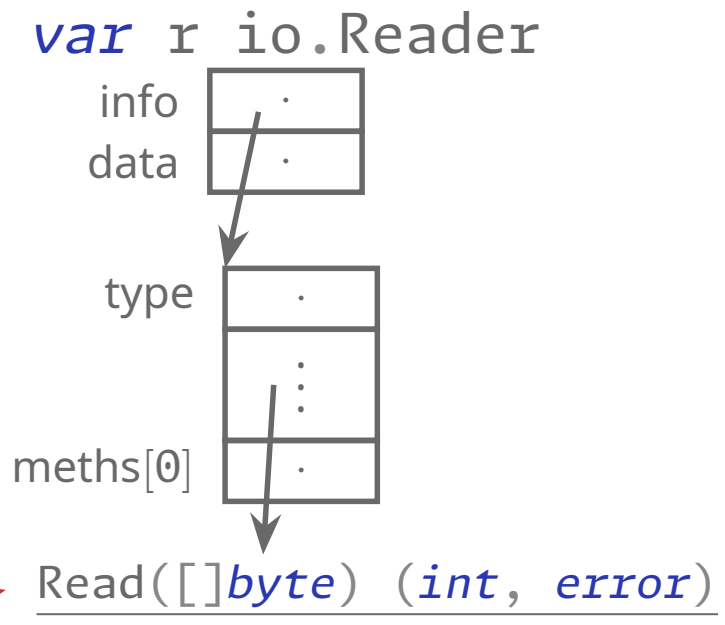


Devirtualization

Devirtualization is a kind of optimizations that an converts **interface method call** 👍 with concrete func call.

```
var r io.Reader
r = strings.NewReader("Hello")
buf := make([]byte, 5)
n, _ := r.Read(buf)
```

Call directly!



☝ ヨシ! ***at compile runtime***

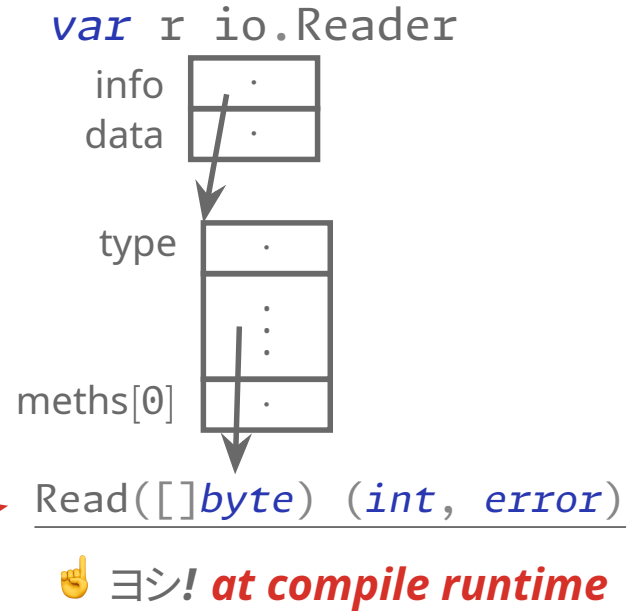
■Devirtualization

Devirtualization is a kind of optimizations that an converts **interface method call** 👍 with concrete func call.

- Reduces ***interface method call overhead:***
  No look up and typechecking at runtime
- Enables ***further optimizations:***
  Same to inlining

```
var r io.Reader
r = strings.NewReader("Hello")
buf := make([]byte, 5)
n, _ := r.Read(buf)
```

Call directly!



☝ ヨシ! ***at compile runtime***

Limitation for Devirtualization

⚠**Limitation**: can only be applied if concrete method is **determined statically**.

```
var r io.Reader
if os.Getenv("MODE") == "string" {
  r = strings.NewReader("Hello")
} else {
  r = bytes.NewReader([]byte("Hello"))
}
buf := make([]byte, 5)
n, _ := r.Read(buf)
```

Limitation for Devirtualization

⚠ **Limitation**: can only be applied if concrete method is **determined statically**.

```
var r io.Reader
if os.Getenv("MODE") == "string" {
  r = strings.NewReader("Hello")
} else {
  r = bytes.NewReader([]byte("Hello"))
}
buf := make([]byte, 5)
n, _ := r.Read(buf)
```

dynamic conditional

Limitation for Devirtualization

⚠ **Limitation**: can only be applied if concrete method is **determined statically**.

```
var r io.Reader
if os.Getenv("MODE") == "string" {
  r = strings.NewReader("Hello")
} else {
  r = bytes.NewReader([]byte("Hello"))
}
buf := make([]byte, 5)
n, _ := r.Read(buf)
```

dynamic conditional

Limitation for Devirtualization

⚠ **Limitation**: can only be applied if concrete method is **determined statically**.

```
var r io.Reader
if os.Getenv("MODE") == "string" {
  r = strings.NewReader("Hello")
} else {
  r = bytes.NewReader([]byte("Hello"))
}
buf := make([]byte, 5)
n, _ := r.Read(buf)
```

dynamic conditional

Can't resolve ***statically***, which should call `(*strings.Reader).Read` or `(*bytes.Reader).Read`?

Profile-Guided Optimization

Go compiler performs several optimizations,

Profile-Guided Optimization

Go compiler performs several optimizations, *and* there are **still room for more!**

- *Conditionals* for inlining
- *Limitation* for devirtualization

Profile-Guided Optimization

Go compiler performs several optimizations,
and there are **still room for more!**

- *Conditionals* for inlining
- *Limitation* for devirtualization

This is where

Profile-Guided Optimization

comes in 😎

Profile-Guided Optimization

PGO is an optimization method that:

- Uses *profiling information* from **program execution**
- Enables *more aggressive optimizations*, such as inlining and devirtualization

Profile-Guided Optimization

PGO is an optimization method that:

- Uses *profiling information* from **program execution**
- Enables *more aggressive optimizations*, such as **inlining** and **devirtualization**


Profile-Guided Optimization

PGO is an optimization method that:

- Uses *profiling information* from **program execution**

So,
***how to* collect profiles?**

- Enables *more aggressive optimizations*, such as inlining and devirtualization

👍 👍

Creating and Collecting Profiles

Profile data is represented as *pprof* format.
There are several choices to create:

- `runtime/pprof`
  Writes out profile files
- `net/http/pprof`
  Runs HTTP server for get profiling data
- gopkg.in/DataDog/dd-trace-go.v1/profiler
  Sends profiles to the Datadog API

■#PR

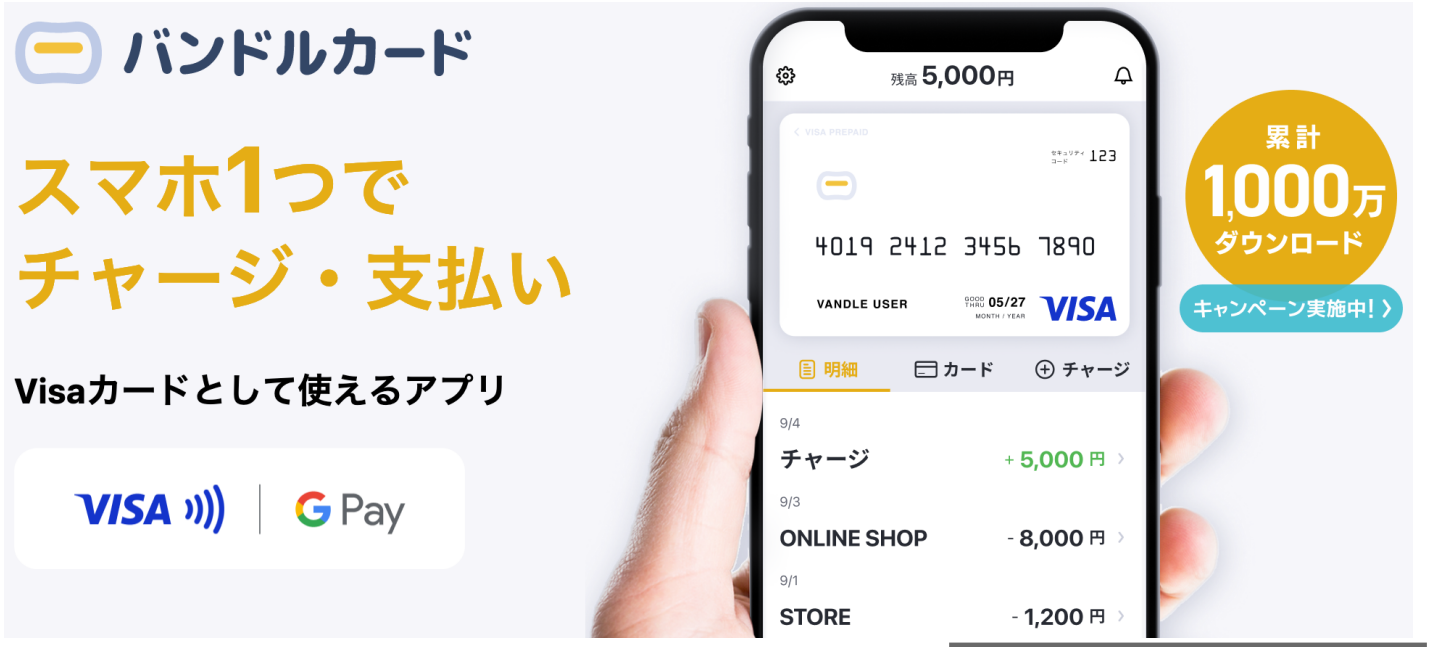
突然ですが

ここで宣伝です

■#PR


バンドルカード
スマホ1つで
チャージ・支払い
Visaカードとして使えるアプリ
VISA
Pay
残高 5,000円
VISA PREPAID
123
4019 2412 3456 7890
VANDLE USER
05/27
VISA
明細
カード
チャージ
9/4
チャージ
+ 5,000 円
9/3
ONLINE SHOP
- 8,000 円
9/1
STORE
- 1,200 円
累計
1,000万
ダウンロード
キャンペーン実施中!

■#PR



Vandle API server → 🐶**Datadog**

profiles,logs
via `dd-trace-go/profiler`

●Creating and Collecting Profiles

To fetch profiles from 🐶Datadog……

- **datadog-pgo**
  Can fetch many profiles(up to 30?) at once!

●Creating and Collecting Profiles

To fetch profiles from 🐶Datadog.....

- **datadog-pgo**
  Can fetch many profiles(up to 30?) at once!
  At build phase:

```
ENV DD_API_KEY=${DD_API_KEY}
ENV DD_APP_KEY=${DD_APP_KEY}

RUN datadog-pgo -profiles 30 \
  'service:vandle-api env:prd' ./default.pgo
```

Creating and Collecting Profiles

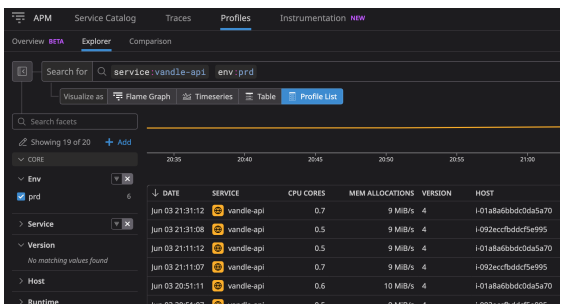
To fetch profiles from 🐶Datadog……

- **datadog-pgo**
  Can fetch many profiles(up to 30?) at once!
  At build phase:

```
ENV DD_API_KEY=${DD_API_KEY}
ENV DD_APP_KEY=${DD_APP_KEY}

RUN datadog-pgo -profiles 30 \
  'service:vandle-api env:prd' ./default.pgo
```

- ✋ Pick from APM profile list BY HAND



Difficult to get many profiles😭

Compiling with PGO

Compiler flags for PGO:

```
$ go build -pgo -gcflags='-m=2 -l=4' ./vandle-server
```

Compiling with PGO

Compiler flags for PGO:

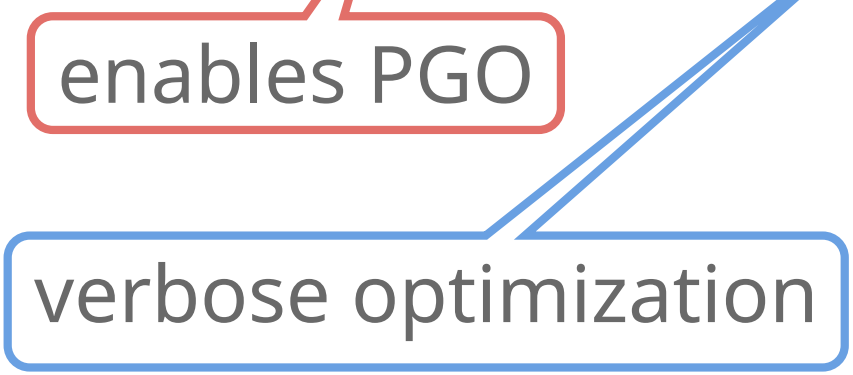
```
$ go build -pgo -gcflags='-m=2 -l=4' ./vandle-server
```

enables PGO

Compiling with PGO

Compiler flags for PGO:

```
$ go build -pgo -gcflags='-m=2 -l=4' ./vandle-server
```



Compiling with PGO

Compiler flags for PGO:

```
$ go build -pgo -gcflags='-m=2 -l=4' ./vandle-server
```

enables PGO

controls inlining, `-l=4` enables inlining **non-leaf functions** (!)

verbose optimization

Compiling with PGO

Compiler flags for PGO:

```
$ go build -pgo -gcflags='-m=2 -l=4' ./vandle-server
```

Compiling with PGO

Compiler flags for PGO:

```
$ go build -pgo -gcflags='-m=2 -l=4' ./vandle-server
......
internal/reflectlite/type.go:414:28:
    PGO devirtualizing interface call
    u.common to rtype.common
......
runtime/mgcsweep.go:499:6:
    cannot inline (*sweepLocked).sweep:
    function too complex:
    cost 2030 exceeds budget 2000
......
```

Compiling with PGO

Compiler flags for PGO:

```
$ go build -pgo -gcflags
......
internal/reflectlite/type.go:414:28:
    PGO devirtualizing interface call
    u.common to rtype.common
......
runtime/mgcsweep.go:499:6:
    cannot inline (*sweepLocked).sweep:
    function too complex:
    cost 2030 exceeds budget 2000
......
```

Devirtualize statically ambiguous interface call!

Compiling with PGO

Compiler flags for PGO:

```
$ go build -pgo -gcflags
......
internal/reflectlite/type.go:414:28:
    PGO devirtualizing interface call
    u.common to rtype.common
......
runtime/mgcsweep.go:49
    cannot inline (*sw
    function too complex:
    cost 2030 exceeds budget 2000
......
```

Devirtualize statically ambiguous interface call!

Not inlined, but budget is lifted to 2000!

Evaluation

≈ 240000 lines

Compiling **Vandle** API server,
with or without PGO and `-l`, then count output

```
$ go build ..... \
  | grep -E '(can inline|PGO devirtualizing)' \
  | wc -l
```

Evaluation

```
$ go build ..... \
  | grep -E '(can inline|PGO devirtualizing)' \
  | wc -l
```

flags	result (lines)	inlining	PGO devirt
`-pgo=off` (default)	110313	110313	-
`-pgo=off -l=4`	+67561	177874	-
`-pgo`	-418	109879	16
`-pgo -l=4`	+68851	179147	17

Evaluation

```
$ go build ..... \
  | grep -E '(can inline|PGO devirtualizing)' \
  | wc -l
```

flags	result (lines)	inlining	PGO devirt
`-pgo=off` (default)	110313	110313	-
`-pgo=off -l=4`	+67561	177874	-
`-pgo`	-418	109879	16
`-pgo -l=4`	+68851	179147	17

- `-l=4` works well!

Evaluation

```
$ go build ..... \
  | grep -E '(can inline|PGO devirtualizing)' \
  | wc -l
```

flags	result (lines)	inlining	PGO devirt
`-pgo=off` (default)	110313	110313	-
`-pgo=off -l=4`	+67561	177874	-
`-pgo`	-418	109879	16
`-pgo -l=4`	+68851	179147	17

- `-l=4` works well!
  - with PGO performs inlining +1273

Evaluation

```
$ go build ..... \
  | grep -E '(can inline|PGO devirtualizing)' \
  | wc -l
```

flags	result (lines)	inlining	PGO devirt
`-pgo=off` (default)	110313	110313	-
`-pgo=off -l=4`	+67561	177874	-
`-pgo`	-418	109879	16
`-pgo -l=4`	+68851	179147	17

- `-l=4` works well!
  - with PGO performs inlining +1273
- `-l=4` accelerates devirtualization

Evaluation

```
$ go build ..... \
  | grep -E '(can inline|PGO devirtualizing)' \
  | wc -l
```

flags	result (lines)	inlining	PGO devirt
`-pgo=off` (default)	110313	110313	-
`-pgo=off -l=4`	+67561	177874	-
`-pgo`	-418	109879	16
`-pgo -l=4`	+68851	179147	17

- `-l=4` works well!
  - with PGO performs inlining +1273
- `-l=4` accelerates devirtualization
- WHAT?!

  Optimizations is so complicated ... 😌

Conclusion

- Introduces **profile-guided optimization**, **inlining** and **devirtualization**
- Evaluates PGO with `-l` flag, using production code
- Vandle card is running with PGO build!
- **Optimizations are so deep and intereseting!**

Conclusion

- Introduces **profile-guided optimization**, **inlining** and **devirtualization**
- Evaluates PGO with `-l` flag, using production code
- Vandle card is running with PGO build!
- **Optimizations are so deep and intereseting!**
- Couldn't talk today 😭:
  - Pprof for visualizing call site
  - PGO calculates hotness by **CFD**
  - **AutoFDO**, Continuous compiling with profiles
  - Comparison with other languages' compilers
    - Haskell, resolves *all* typeclass constraints
    - JVM, *JIT*
    - PGO in .NET
  - Interface with Generics