

개발 생산성 10배 높이기 : from C++ to Golang

김덕일 NAVER / Search Serving Platform
조현준 NAVER / Search Serving Platform

CONTENTS

1. Let Me GO
2. Let's GO with SWIG
3. How We GO
4. You Can GO

1. Let Me GO

1.1 검색 서비스를 아시나요?

- 블로그, 카페, 지식인 등 200여개의 서비스가 운영중
- 대부분의 서비스는 검색 기능을 제공하고 있고 검색 서비스는 이를 지칭함



1.1 검색 서비스를 아시나요?

검색 서비스 구조

2020 DEVIEW 검색



1.1 검색 서비스를 아시나요?

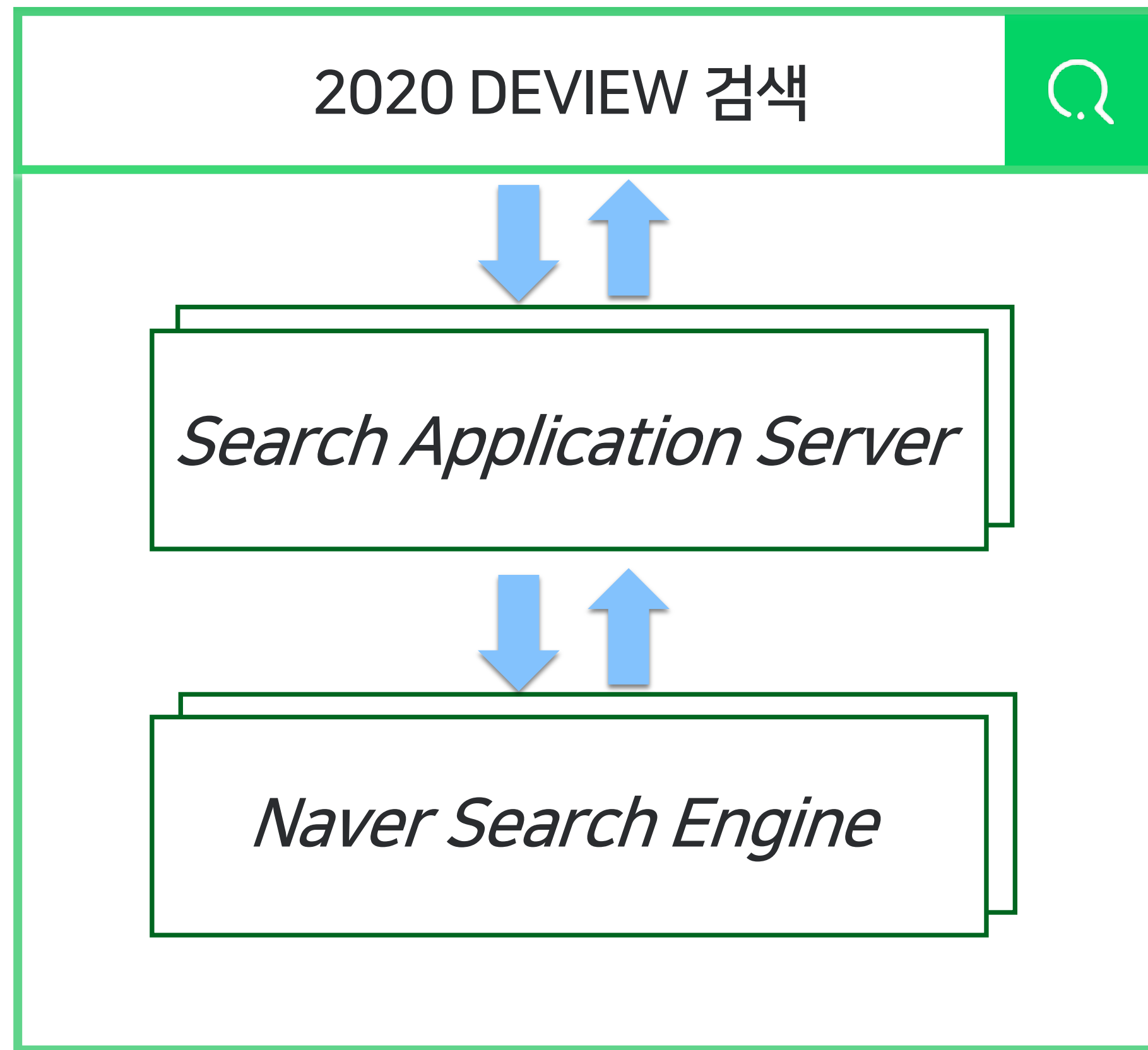
검색 서비스 구조



Apache MPM Prefork 방식으로 구성
질의 분석, 정렬 방식, 랭킹 모델링 로직 등이 구현
각 서비스들 마다 고유한 검색 로직을 구현

1.1 검색 서비스를 아시나요?

검색 서비스 구조

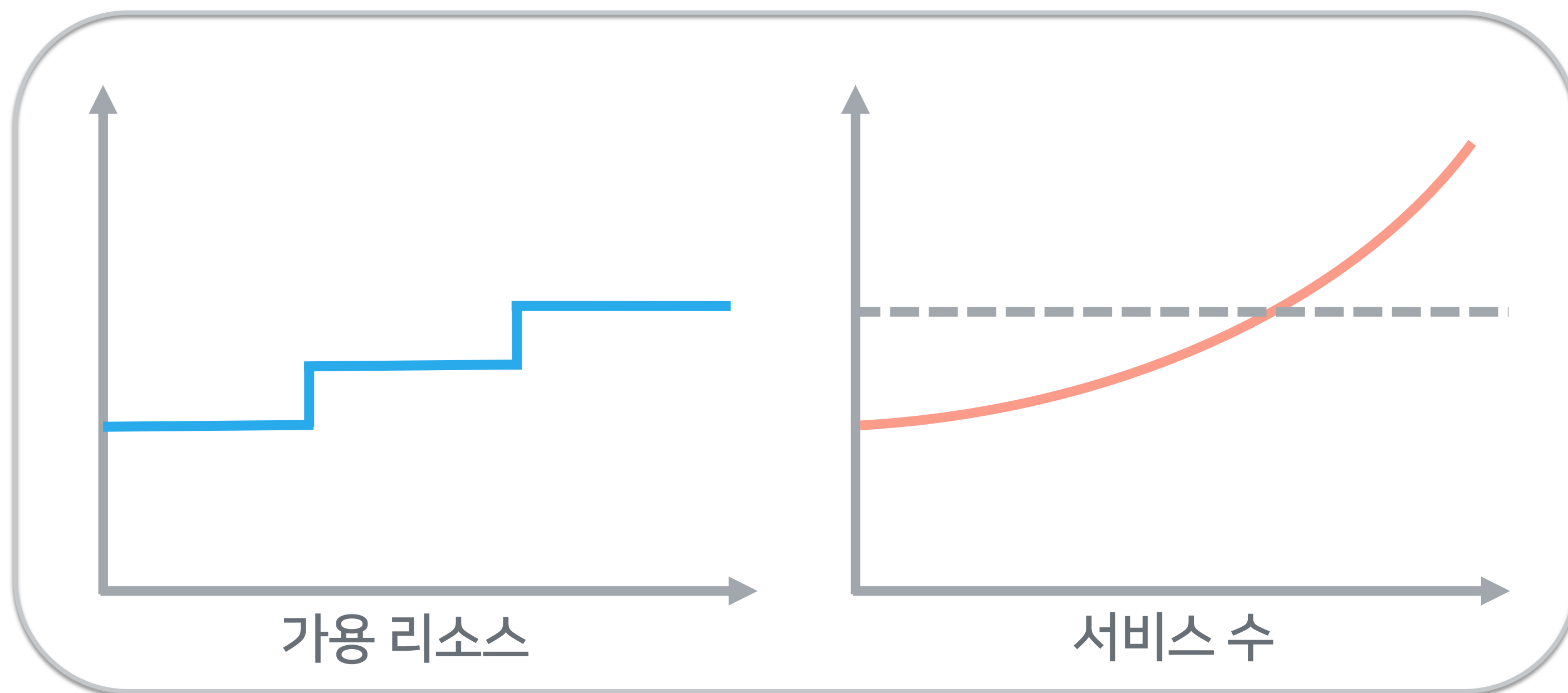


Apache MPM Prefork 방식으로 구성
질의 분석, 정렬 방식, 랭킹 모델링 로직 등이 구현
각 서비스들 마다 고유한 검색 로직을 구현

C++로 자체 개발한 고성능 검색 엔진 사용

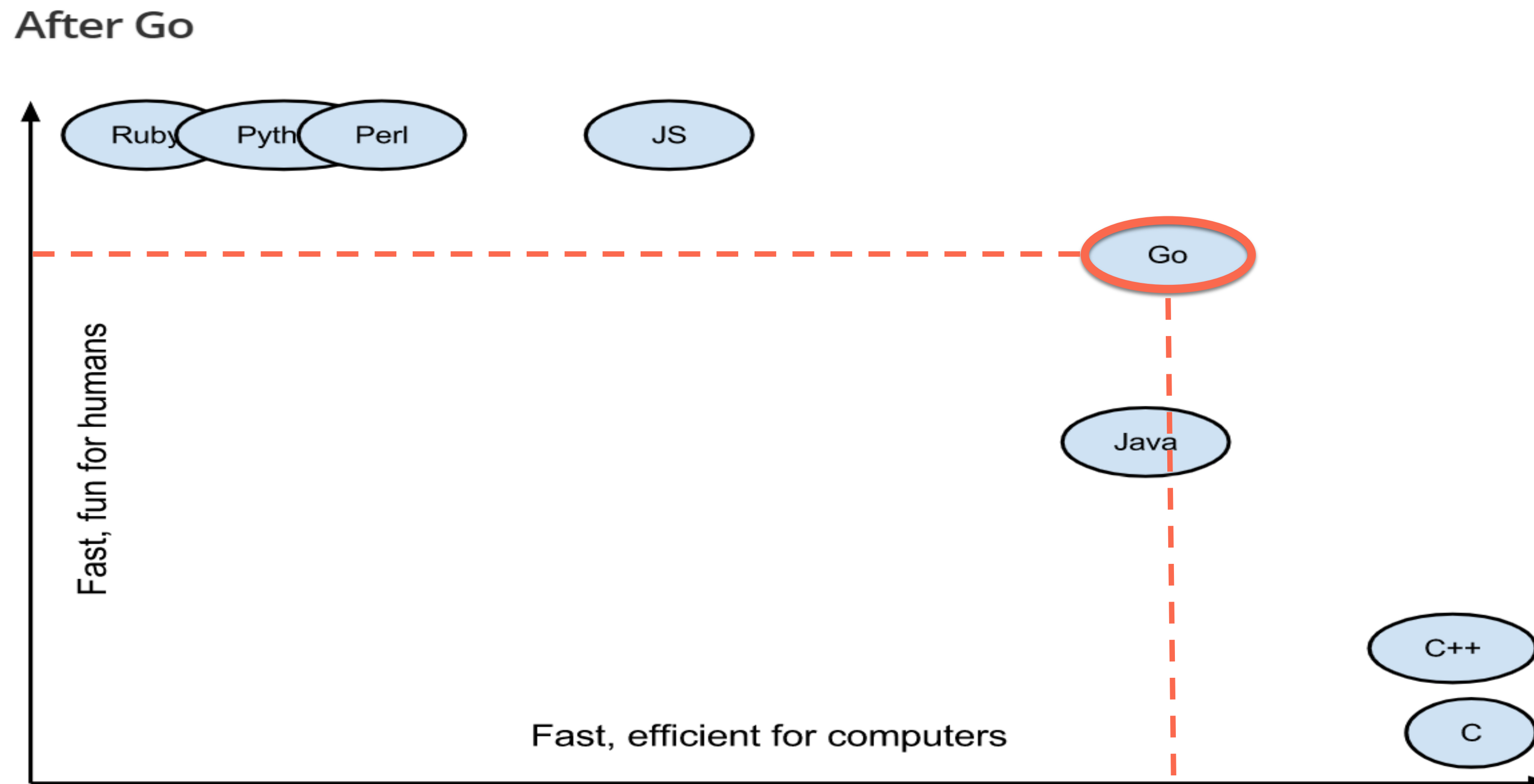
1.2 생산성에 대한 고민

- 기존 검색 서비스 규모는 커지고, 신규 검색 서비스는 계속 증가
- 적은 리소스로 C++과 유사한 성능을 보장하면서 생산성을 향상시킬수 있는 방법을 고민



1.3 GO언어를 도입해보자

- 성능이 보장되고 생산성을 늘릴수 있는 DevOps에 적합한 언어



Brad Fitzpatrick GoCon 2014

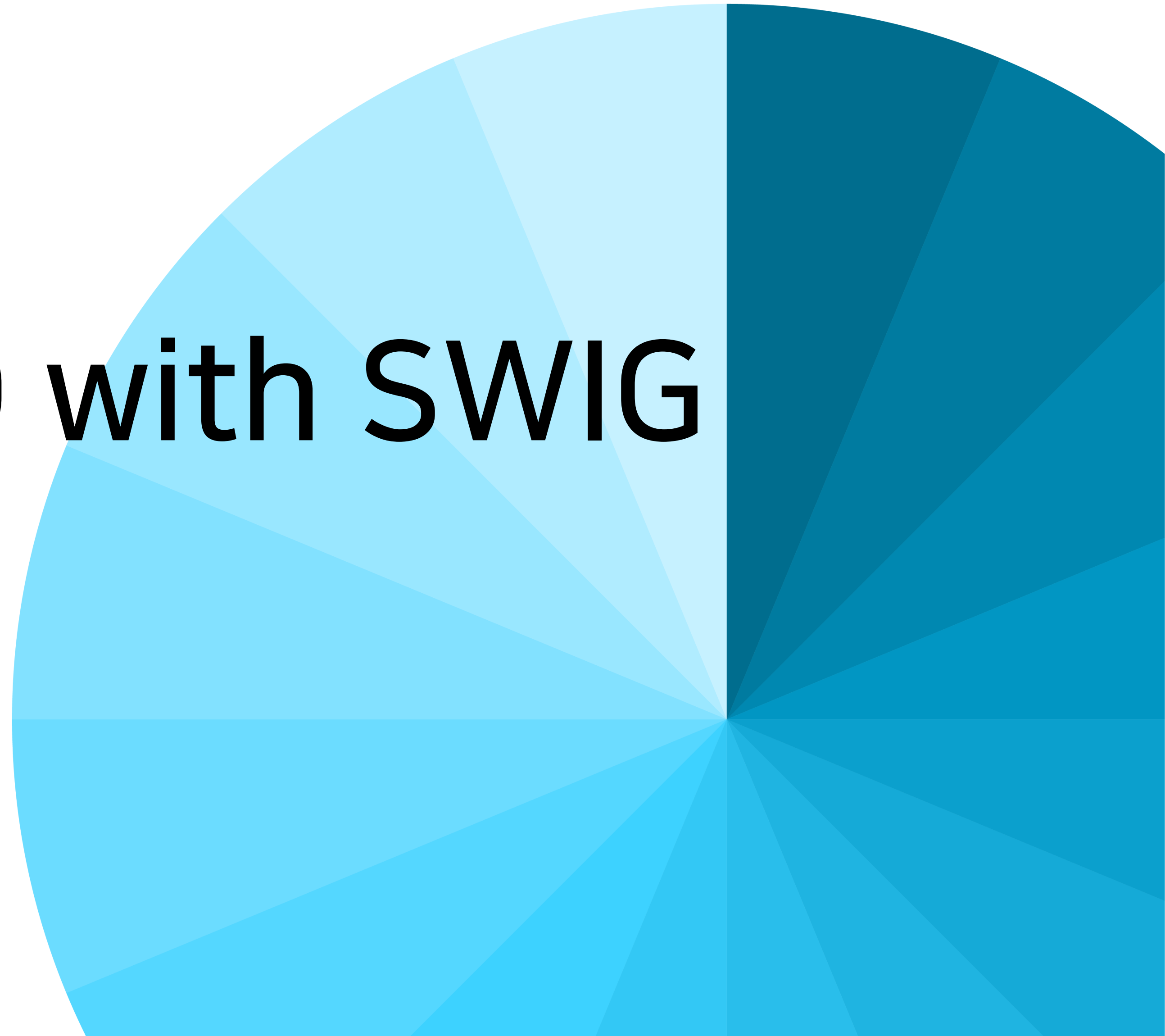
1.3 GO언어를 도입해보자

- C++과 GO를 여러 방면에서 비교해봤을 때 장점이 많음

	C++		GO
성능	매우 좋음	≥	C++과 유사함
러닝 커브	어려운 편에 속함	<	쉬운 편에 속함
라이브러리 추가	일일이 추가	<	명령어 한줄로 쉽게 추가
메모리 관리	관리가 필요함	<	GC 지원
유닛 테스트	직접 구축	<	기본 지원
숙련도에 따른 편차	큰 편에 속함	<	적은 편에 속함

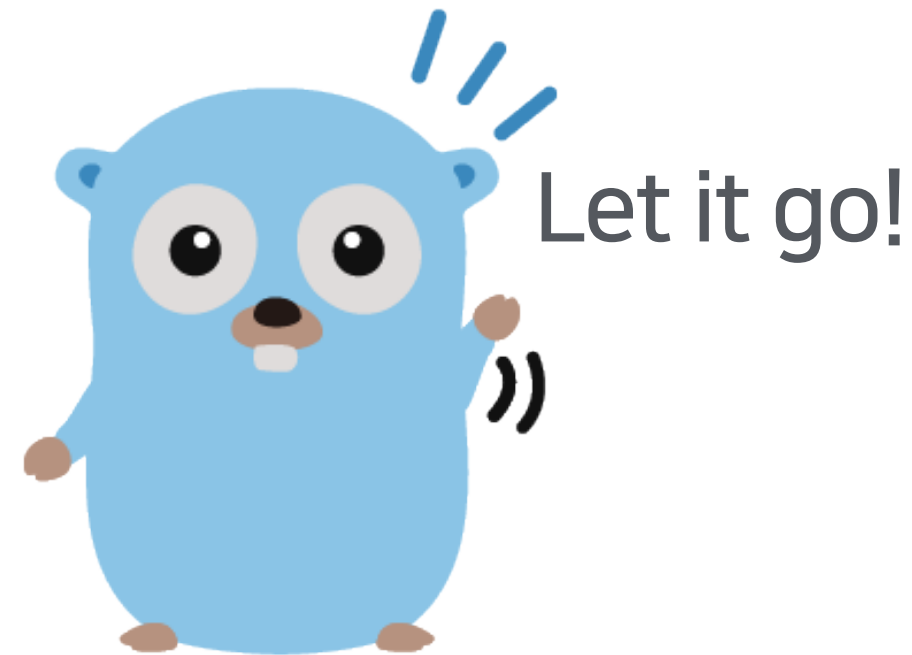
하지만 검색 엔진 라이브러리는
C++인데 어떻게 해야되지?

2. Let's GO with SWIG



2.1 SWIG를 사용하자

- C/C++ 라이브러리를 다양한 언어에서 사용할 수 있도록 추상화 시켜주는 오픈소스
- GO 뿐만 아니라 Python, C#, Java, Perl, PHP, R, Ruby 등 다양한 언어 지원
- 4.0 버전까지 릴리즈, 계속해서 기능이 추가 되고 있음

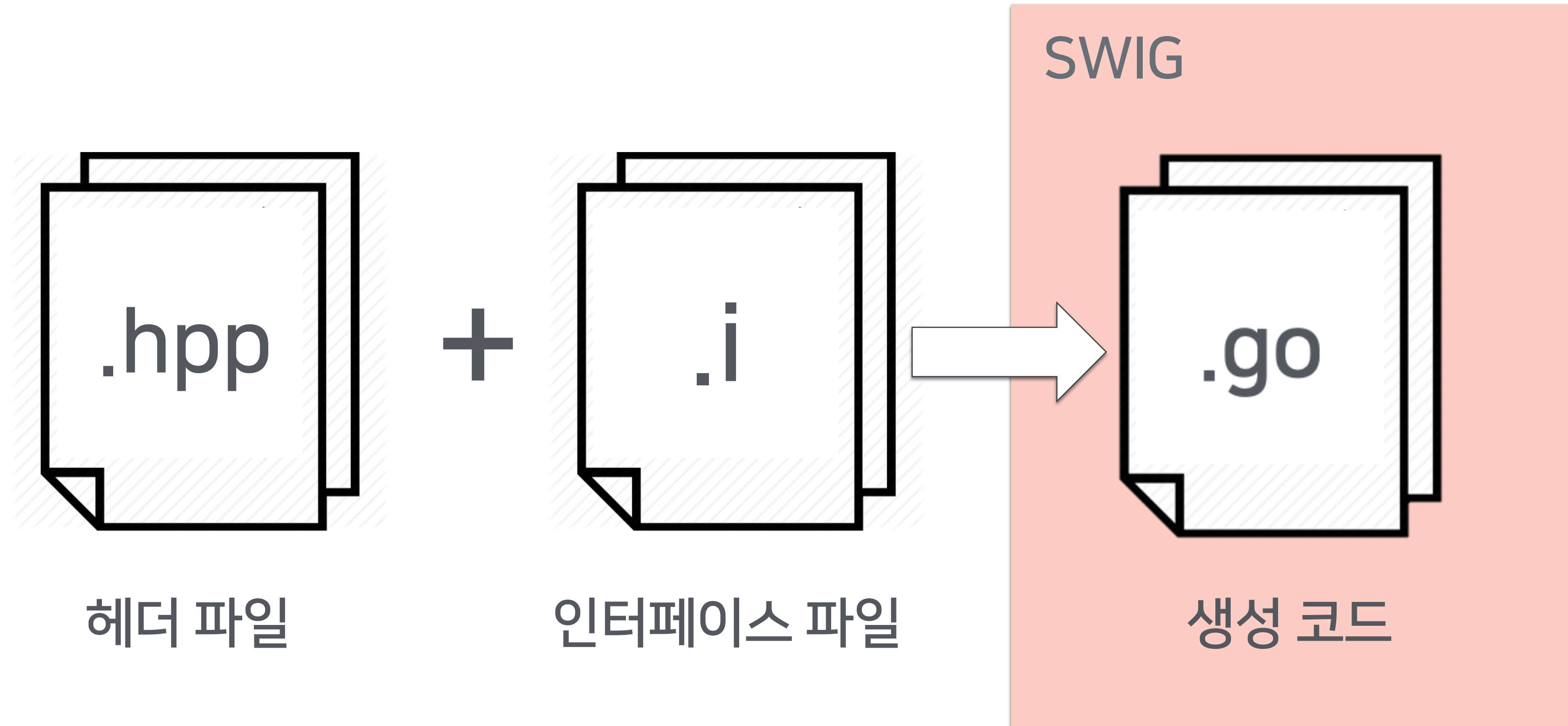


Gopher by Renée French, and tenntenn

+

Simplified Wrapper and Interface Generator

2.2 SWIG는 세가지만 숙지하면 끝



2.2 SWIG는 세가지만 숙지하면 끝

헤더 파일

- GO 에서 사용할 C++함수원형이 존재하는 파일, 래핑 타겟 파일
- 주로 C++ 라이브러리에서 필요한 부분을 프록시 코드로 만들어서 사용

```
#include <iostream>
void printHello();
```

[device.hpp]

```
#include "device.hpp"
void printHello() {
    std::cout << "Hello Devview!!" << std::endl;
}
```

[device.cpp]

libdevice.so

2.2 SWIG는 세가지만 숙지하면 끝

헤더 파일

- GO 에서 사용할 C++함수원형이 존재하는 파일, 래핑 타겟 파일
- 주로 C++ 라이브러리에서 필요한 부분을 프록시 코드로 만들어서 사용

```
#include <iostream>
void printHello();
```

[device.hpp]

```
#include "device.hpp"
void printHello() {
    std::cout << "Hello Devview!!" << std::endl;
}
```

[device.cpp]

libdevview.so

2.2 SWIG는 세가지만 숙지하면 끝

인터페이스 파일

- C++ 코드를 어떤 방식으로 래핑할지 결정, 래핑을 위한 다양한 문법을 지원
- 대부분의 작업은 해당 파일에서 이루어짐

```
%module devview  
%include devview.hpp
```

```
%ignore ...  
%rename ...  
%template ...
```

```
%insert (go_wrapper) %{  
...  
%}
```

코드 생성시 참조하는 영역

코드 생성시 삽입하는 영역

[devview.i]

2.2 SWIG는 세가지만 숙지하면 끝

인터페이스 파일

- C++ 코드를 어떤 방식으로 래핑할지 결정, 래핑을 위한 다양한 문법을 지원
- 대부분의 작업은 해당 파일에서 이루어짐

```
%module devview  
%include devview.hpp
```

```
%ignore ...  
%rename ...  
%template ...
```

```
%insert (go_wrapper) %{  
...  
%}
```

코드 생성시 참조하는 영역

코드 생성시 삽입하는 영역

[devview.i]

2.2 SWIG는 세가지만 숙지하면 끝

인터페이스 파일

- C++ 코드를 어떤 방식으로 래핑할지 결정, 래핑을 위한 다양한 문법을 지원
- 대부분의 작업은 해당 파일에서 이루어짐

```
%module devview  
%include devview.hpp
```

```
%ignore ...  
%rename ...  
%template ...
```

코드 생성시 참조하는 영역

```
%insert (go_wrapper) %{  
...  
%}
```

코드 생성시 삽입하는 영역

[devview.i]

2.2 SWIG는 세가지만 숙지하면 끝

생성 코드

- SWIG를 통해서 생성되는 코드, 디버깅시 주로 확인
- 패키지로 import 하여 사용, CGO를 통해 자동으로 빌드

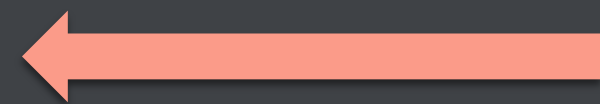
```
package devview
```

```
...
```

```
func PrintHello() {
```

```
    Swig_cxx_printHello(...)
```

```
}
```



생성된 패키지의 이름



호출시

Hello Devview!! 출력

[generated devview.go]

2.3 SWIG를 적용하자

생성 코드 호출 과정

- GO영역에서...

GO

```
package main  
import devview
```

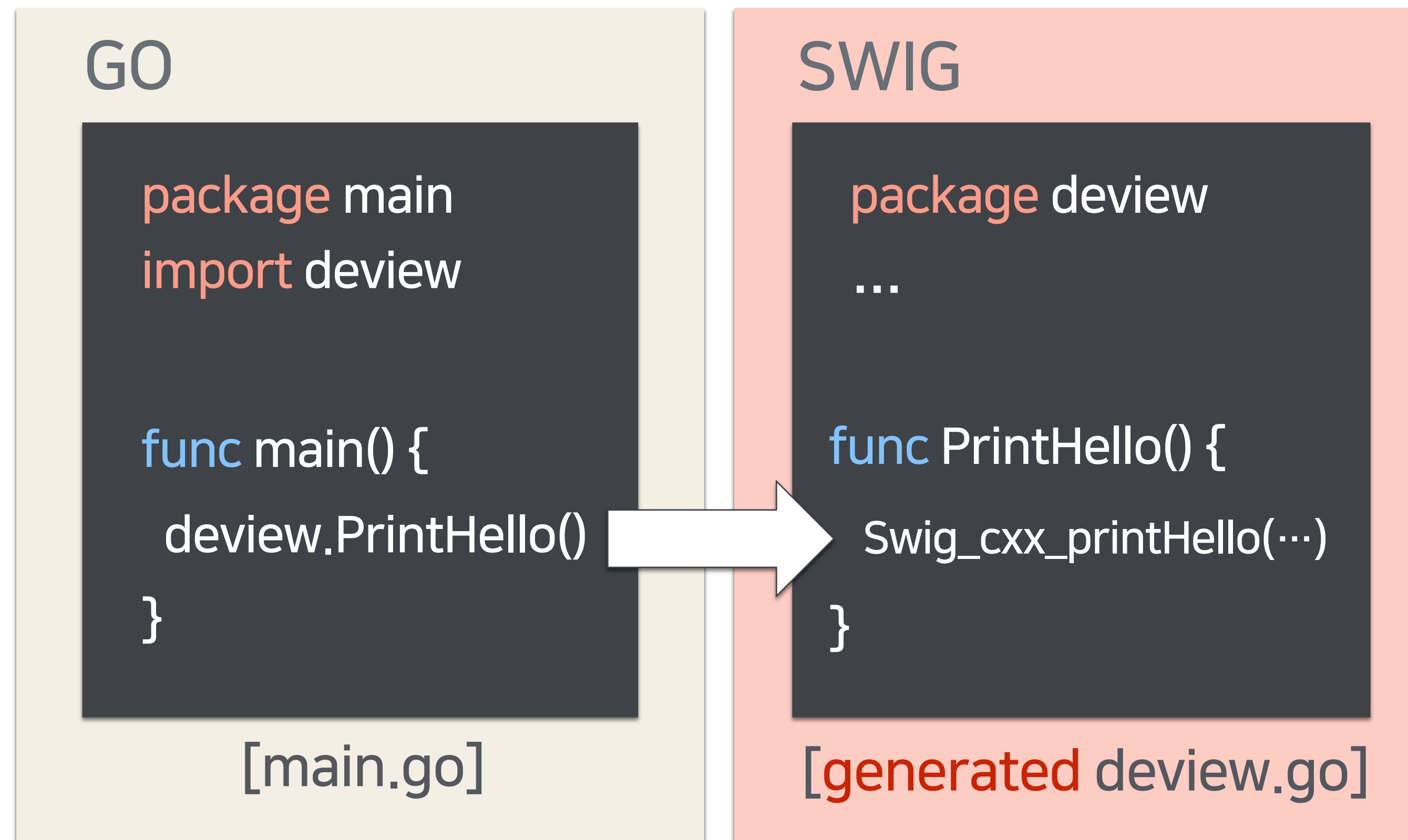
```
func main() {  
    devview.PrintHello()  
}
```

[main.go]

2.3 SWIG를 적용하자

생성 코드 호출 과정

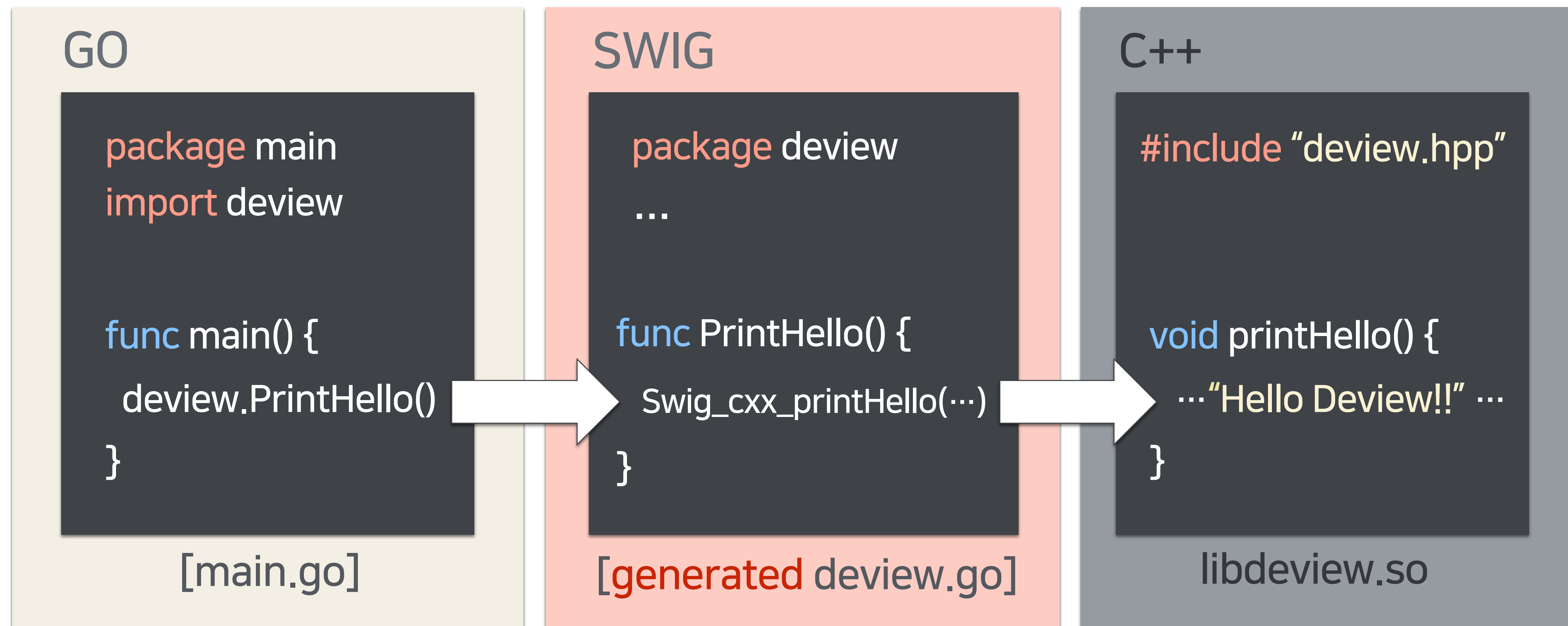
- GO영역에서...



2.3 SWIG를 적용하자

생성 코드 호출 과정

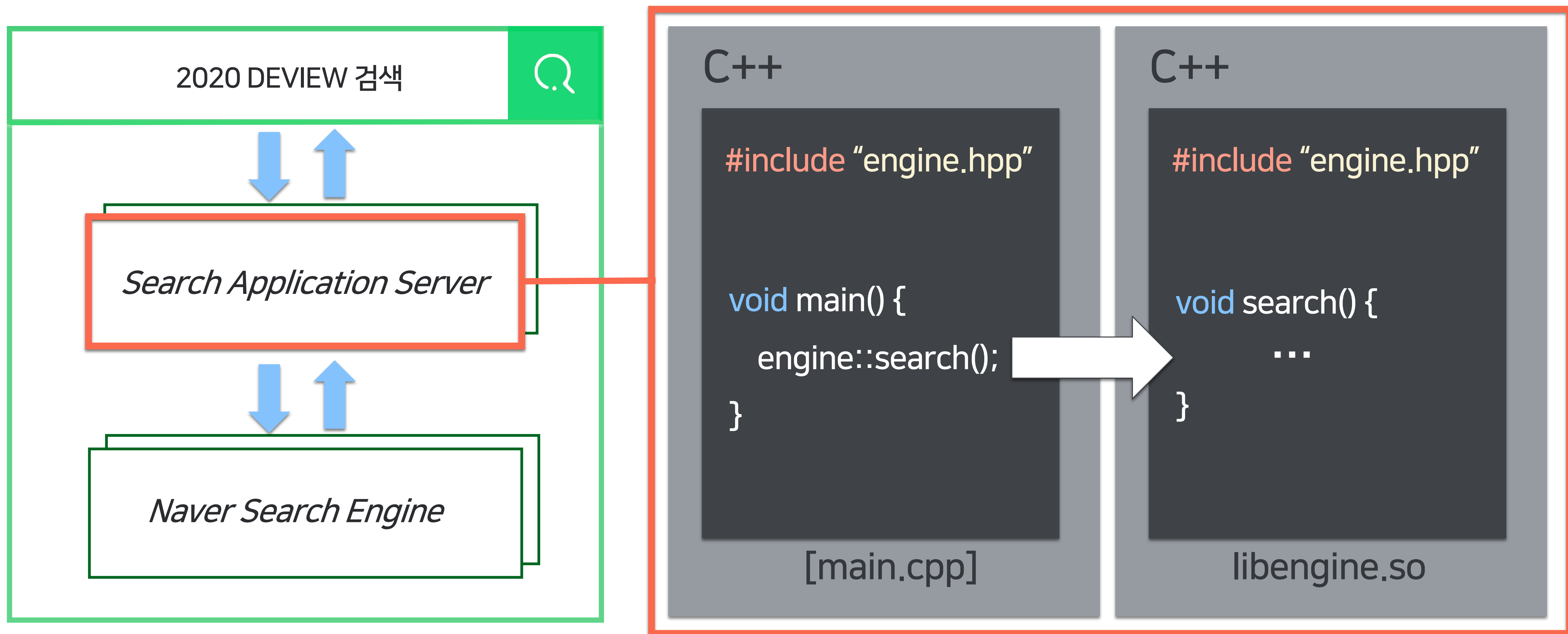
- GO영역에서 C++영역으로 호출



2.3 SWIG를 적용하자

검색 서비스에는 어떻게 적용할까?

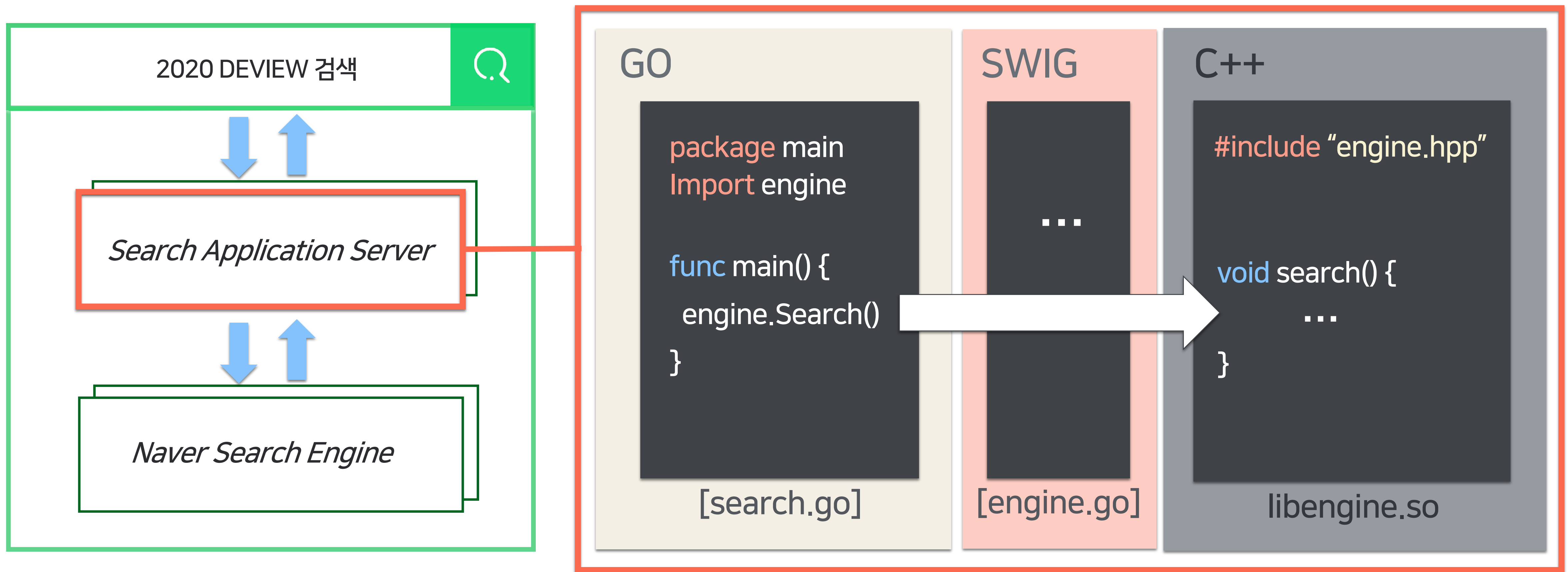
- 기존에는 C++ 검색 엔진 라이브러리를 이용하여 검색하던 구조



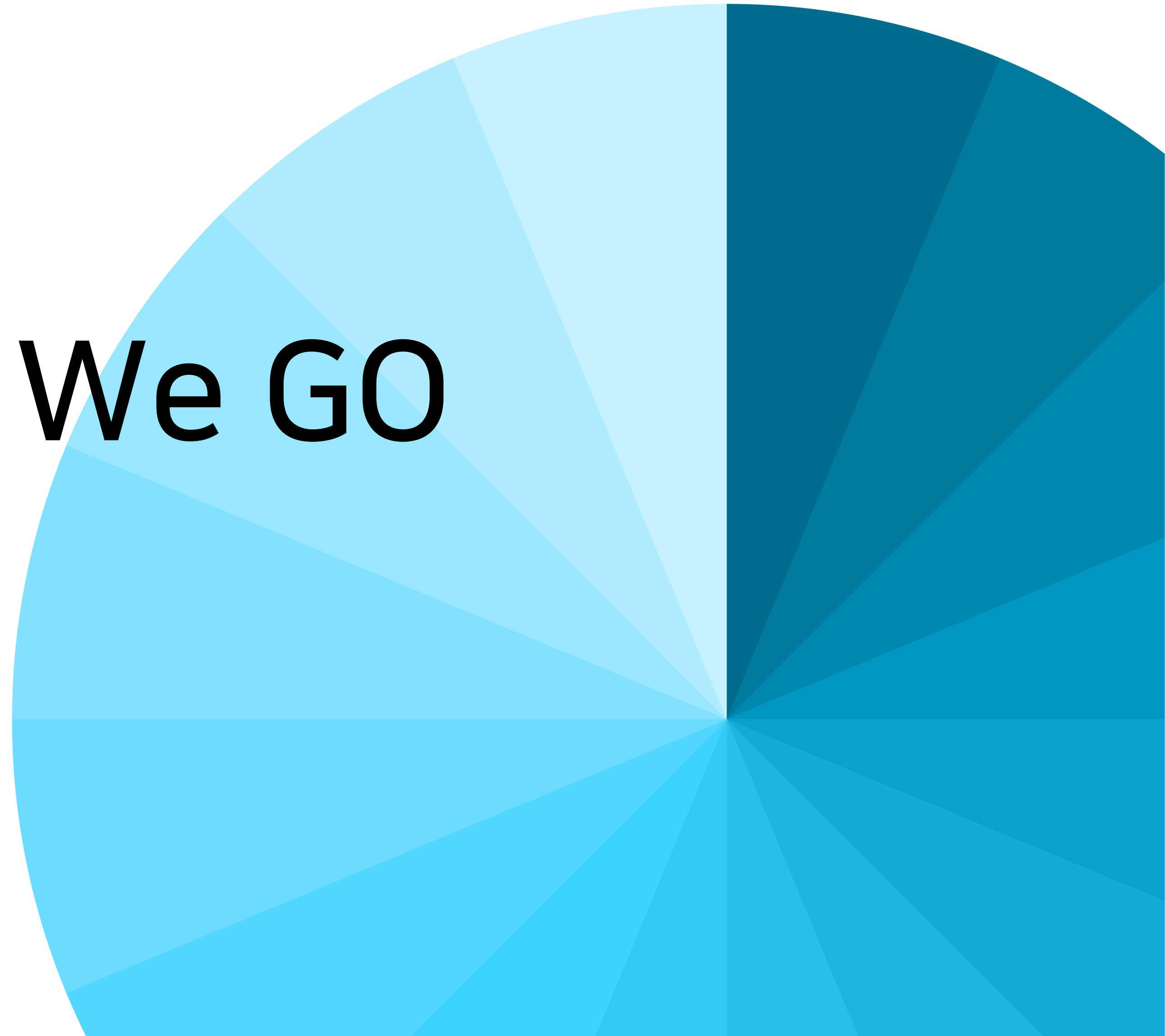
2.3 SWIG를 적용하자

검색 서비스에도 그대로 적용

- C++ 엔진 라이브러리와 의존성을 끊을수 있고, 유지보수하기 좋은 구조



3.How We GO



목표 GO는 GO 답게



3.1 Let Them Go

GO답게 쓰는 첫번째 길

- Ignore Reference, Move, Iterator, Operator Overload, etc

```
%ignore Row::getColumn<T>(std::string const&) const;  
%ignore Table::move;  
%ignore Table::begin();  
%ignore Table::end();  
%ignore Table::erase(const_iterator);  
%ignore Table::operator [](std::size_t) const;
```

[devview.i]

3.1 Let Them Go

GO답게 쓰는 첫번째 길

- Ignore Reference, Move, Iterator, Operator Overload, etc

```
%ignore Row::getColumn<T>(std::string const&) const;  
%ignore Table::move;  
%ignore Table::begin();  
%ignore Table::end();  
%ignore Table::erase(const_iterator);  
%ignore Table::operator [](std::size_t) const;
```

[devview.i]

3.1 Let Them Go

GO답게 쓰는 첫번째 길

- Ignore Reference, Move, Iterator, Operator Overload, etc

```
%ignore Row::getColumn<T>(std::string const&) const;  
%ignore Table::move;  
%ignore Table::begin();  
%ignore Table::end();  
%ignore Table::erase(const_iterator);  
%ignore Table::operator [](std::size_t) const;
```

[devview.i]

3.1 Let Them Go

GO답게 쓰는 첫번째 길

- Ignore Reference, Move, Iterator, Operator Overload, etc

```
%ignore Row::getColumn<T>(std::string const&) const;  
%ignore Table::move;  
%ignore Table::begin();  
%ignore Table::end();  
%ignore Table::erase(const_iterator);  
%ignore Table::operator [](std::size_t) const;
```

[devview.i]

3.1 Let Them Go

GO답게 쓰는 첫번째 길

- Ignore Reference, Move, Iterator, Operator Overload, etc

```
%ignore Row::getColumn<T>(std::string const&) const;  
%ignore Table::move;  
%ignore Table::begin();  
%ignore Table::end();  
%ignore Table::erase(const_iterator);  
%ignore Table::operator [](std::size_t) const;
```

```
%rename(assign) Table::operator=;
```

[devview.i]

3.2 Switch(.type)

직접 사용할 타입 위주로 SWIG에게 가이드

C/C++ type	Go type
bool	bool
char	byte
signed char	int8
unsigned char	byte
short	int16
unsigned short	uint16
int	int
unsigned int	uint
long	int64
unsigned long	uint64
long long	int64
unsigned long long	uint64
float	float32
double	float64
char * char []	string

3.2 Switch(.type)

C/C++ type	Go type
bool	bool
char	byte
signed char	int8
unsigned char	byte
short	int16
unsigned short	uint16
int	int
unsigned int	uint
long	int64
unsigned long	uint64
long long	int64
unsigned long long	uint64
float	float32
double	float64
char * char []	string

직접 사용할 타입 위주로 SWIG에게 가이드

```
%apply double {long double}
%typemap(gotype) int, const int & "int32"
%template(VectorString) std::vector<std::string>;
```

[devview.i]

3.2 Switch(.type)

C/C++ type	Go type
bool	bool
char	byte
signed char	int8
unsigned char	byte
short	int16
unsigned short	uint16
int	int
unsigned int	uint
long	int64
unsigned long	uint64
long long	int64
unsigned long long	uint64
float	float32
double	float64
char * char []	string

직접 사용할 타입 위주로 SWIG에게 가이드

```
%apply double {long double}
%typemap(gotype) int, const int & "int32"
%template(VectorString) std::vector<std::string>;
```

[devview.i]

3.2 Switch(.type)

C/C++ type	Go type
bool	bool
char	byte
signed char	int8
unsigned char	byte
short	int16
unsigned short	uint16
int	int
unsigned int	uint
long	int64
unsigned long	uint64
long long	int64
unsigned long long	uint64
float	float32
double	float64
char * char []	string

직접 사용할 타입 위주로 SWIG에게 가이드

```
%apply double {long double}
%typemap(gotype) int, const int & "int32"
%template(VectorString) std::vector<std::string>;
```

[devview.i]

3.2 Switch(.type)

C/C++ type	Go type
bool	bool
char	byte
signed char	int8
unsigned char	byte
short	int16
unsigned short	uint16
int	int
unsigned int	uint
long	int64
unsigned long	uint64
long long	int64
unsigned long long	uint64
float	float32
double	float64
char * char []	string

직접 사용할 타입 위주로 SWIG에게 가이드

```
%apply double {long double}
%typemap(gotype) int, const int & "int32"
// %template(VectorString) std::vector<std::string>
```

[devview.i]

```
type Std_vector_Sl_std_string_Sg_ interface {
    Swigcptr() uintptr
}
```

[generated devview.go]

3.3 Exception? Recover!

Throw => Catch => Panic

```
%exception {  
  try {  
    $action;  
  } catch (std::exception &e) {  
    _swig_gopanic(e.what());  
  }  
}
```

[devview.i]

3.3 Exception? Recover!

Throw => Catch => Panic => Recover

```
%exception {
    try {
        $action;
    } catch (std::exception &e) {
        _swig_gopanic(e.what());
    }
}
```

[devview.i]

```
func catch(err *error) {
    if r := recover(); r != nil {
        *err = fmt.Errorf(format: "%v", r)
    }
}
```

[main.go]

3.4 Template + Macro

Template은 필요한 만큼 미리 선언 하자

```
%define TEMPLATE_WRAP(Type, T...)
    %template(## ColumnName ## Type) ColumnName<T>;          // type
    %template(## AddColumn ## Type) Table::addColumn<T>;      // func
    %template(## HasColumn ## Type) Table::hasColumn<T>;      // func
%enddef

TEMPLATE_WRAP(Bool,      bool);
TEMPLATE_WRAP(String,    std::string);
TEMPLATE_WRAP(VectorString, std::vector<std::string>);
```

3.4 Template + Macro

Template은 필요한 만큼 미리 선언 하자

```
%define TEMPLATE_WRAP(Type, T...)
    %template(## ColumnName ## Type) ColumnName<T>;          // type
    %template(## AddColumn ## Type) Table::addColumn<T>;      // func
    %template(## HasColumn ## Type) Table::hasColumn<T>;      // func
%enddef

TEMPLATE_WRAP(Bool, bool);
TEMPLATE_WRAP(String, std::string);
TEMPLATE_WRAP(VectorString, std::vector<std::string>);
```

3.4 Template + Macro

Template은 필요한 만큼 미리 선언 하자

```
%define TEMPLATE_WRAP(Type, T...)
    %template(## ColumnName ## Type) ColumnName<T>;          // type
    %template(## AddColumn ## Type) Table::addColumn<T>;      // func
    %template(## HasColumn ## Type) Table::hasColumn<T>;      // func
%enddef
```

```
TEMPLATE_WRAP(Bool, bool);
TEMPLATE_WRAP(String, std::string);
TEMPLATE_WRAP(VectorString, std::vector<std::string>);
```


3.4 Template + Macro

Template은 필요한 만큼 미리 선언 하자

```
%define TEMPLATE_WRAP(Type, T...)
    %template(## ColumnName ## Type) ColumnName<T>;          // type
    %template(## AddColumn ## Type) Table::addColumn<T>;      // func
    %template(## HasColumn ## Type) Table::hasColumn<T>;      // func
%enddef

TEMPLATE_WRAP(Bool,      bool);
TEMPLATE_WRAP(String,    std::string);
TEMPLATE_WRAP(VectorString, std::vector<std::string>);
```

3.4 Template + Macro

Template은 필요한 만큼 미리 선언 하자

```
%define TEMPLATE_WRAP(Type, T...)
    %template(## ColumnName ## Type) ColumnName<T>;          // type
    %template(## AddColumn ## Type) Table::addColumn<T>;      // func
    %template(## HasColumn ## Type) Table::hasColumn<T>;      // func
%enddef

TEMPLATE_WRAP(Bool,      bool);
TEMPLATE_WRAP(String,    std::string);
TEMPLATE_WRAP(VectorString, std::vector<std::string>);
```


3.4 (Rename) + Template + Macro

SWIG는 `interface{}` 를 너무나 좋아 한다

```
bool hasColumn(const std::string& columnName) const;  
template <typename T>  
bool hasColumn(const ColumnName<T>& name) const;
```

3.4 (Rename) + Template + Macro

SWIG는 `interface{}` 를 너무나 좋아 한다

```
bool hasColumn(const std::string& columnName) const;  
template <typename T>  
bool hasColumn(const ColumnName<T>& name) const;
```

3.4 (Rename) + Template + Macro

SWIG는 `interface{}` 를 너무나 좋아 한다

```
bool hasColumn(const std::string& columnName) const;
template <typename T>
bool hasColumn(const ColumnName<T>& name) const;
```

```
HasColumn(name interface{}) bool
```

[generated devview.go]

3.4 (Rename) + Template + Macro

SWIG는 `interface{}` 를 너무나 좋아 한다

```
bool hasColumn(const std::string& columnName) const;
template <typename T>
bool hasColumn(const ColumnName<T>& name) const;
```

```
HasColumn(name interface{}) bool
```

[generated devview.go]

미리 분리 시키자

```
%rename(HasColumn) Table::hasColumn;
%rename(## HasColumn ## Type) Table::hasColumn<T>;
```

[devview.i]

3.4 (Rename) + Template + Macro

SWIG는 `interface{}` 를 너무나 좋아 한다

```
bool hasColumn(const std::string& columnName) const;
template <typename T>
bool hasColumn(const ColumnName<T>& name) const;
```

```
HasColumn(name interface{}) bool
```

[generated devview.go]

미리 분리 시키자

```
%rename(HasColumn)      Table::hasColumn;
%rename(## HasColumn ## Type) Table::hasColumn<T>;
```

[devview.i]

3.5 Enum

Enum에 붙는 Namespace는 뗄 수가 없다

```
namespace npp
{
    namespace sc2
    {
        enum ColumnType
        {
```

||

```
type NppSc2ColumnType int
```


3.5 Enum

Enum에 붙는 Namespace는 뗄 수가 없다

```
namespace npp
{
    namespace sc2
    {
        enum ColumnType
        {
```

||

```
type NppSc2ColumnType int
```

=

```
%insert(go_wrapper) %{
    type ColumnType = NppSc2ColumnType
%}
```

그러니 별명을 주자

3.6 Inception - Preprocess

SWIG가 자체 Parser를 사용해서 당황스러운 경우

```
const ColumnName<std::string> CN_KEY;           // OK
const ColumnName<std::string> CN_KEY();          // OK
const ColumnName<std::string> CN_KEY("key");     // Syntax Error
const ColumnName<std::string> CN_KEY(string x);  // OK
const ColumnName<std::string> CN_KEY = "key";    // OK
```


3.6 Inception - Preprocess

SWIG가 자체 Parser를 사용해서 당황스러운 경우

```
const ColumnName<std::string> CN_KEY;           // OK
const ColumnName<std::string> CN_KEY( );         // OK
const ColumnName<std::string> CN_KEY( "key" );    // Syntax Error
const ColumnName<std::string> CN_KEY( string x ); // OK
const ColumnName<std::string> CN_KEY = "key";    // OK
```

전처리로 회피 가능하다

```
// CN_KEY("key");
#define CN_KEY(x) CN_KEY = x           // 전처리
#include "npp_search_result.hpp"       // 처리 시작
```

[devview.i]

3.6 Inception - Preprocess

SWIG가 자체 Parser를 사용해서 당황스러운 경우

```
const ColumnName<std::string> CN_KEY;           // OK
const ColumnName<std::string> CN_KEY();          // OK
const ColumnName<std::string> CN_KEY("key");     // Syntax Error
const ColumnName<std::string> CN_KEY(string x);  // OK
const ColumnName<std::string> CN_KEY = "key";    // OK
```

전처리로 회피 가능하다

```
// CN_KEY("key");
#define CN_KEY(x) CN_KEY = x           // 전처리
#include "npp_search_result.hpp"       // 처리 시작
```

[devview.i]

3.6 Inception - Extend

레퍼런스 타입으로 값을 변경해야 하여 당황스러운 경우

```
template <typename T>  
T const& getColumn(const std::string& columnName) const;
```


3.6 Inception - Extend

레퍼런스 타입으로 값을 변경해야 하여 당황스러운 경우

```
template <typename T>  
T const& getColumn(const std::string& columnName) const;  
  
auto& r = t1.addRow();  
r.getColumn(a_idx) = 3; r.getColumn(b_idx1) = 1;
```

3.6 Inception - Extend

레퍼런스 타입으로 값을 변경해야 하여 당황스러운 경우

```
template <typename T>
T const& getColumn(const std::string& columnName) const;

auto& r = t1.addRow();
r.getColumn(a_idx) = 3; r.getColumn(b_idx1) = 1;
```

코드 조각을 심을 수 있다

```
%extend npp::sc2::Row {
    template <typename T>
    void setColumn(std::string name, T val) {
        self->getColumn< T >(name) = val;
    }
}
```

[devview.i]

```
r, _ := t1.AddRow()
r.SetColumnInt32(aIdx, int32(3))
r.SetColumnInt32(bIdx1, int32(1))
```

[main.go]

3.7 Shared Memory

Apache MPM Prefork에서 Shared Cache가 필요(READ >> Write)

- Key-Value Map방식이 필요
- GO는 Communication을 권장, 멀티프로세스가 필요 없음(Goroutine!)
- 서버/클라이언트, gRpc를 통한 통신이 많음

3.7 Shared Memory

Apache MPM Prefork에서 Shared Cache가 필요(READ >> Write)

- sqlite3 사용
- sqlite3에서 Write-Ahead-Logging 활성화(Single Write)
- /dev/shm 영역에 db 생성

3.8 Calm Down, Mem

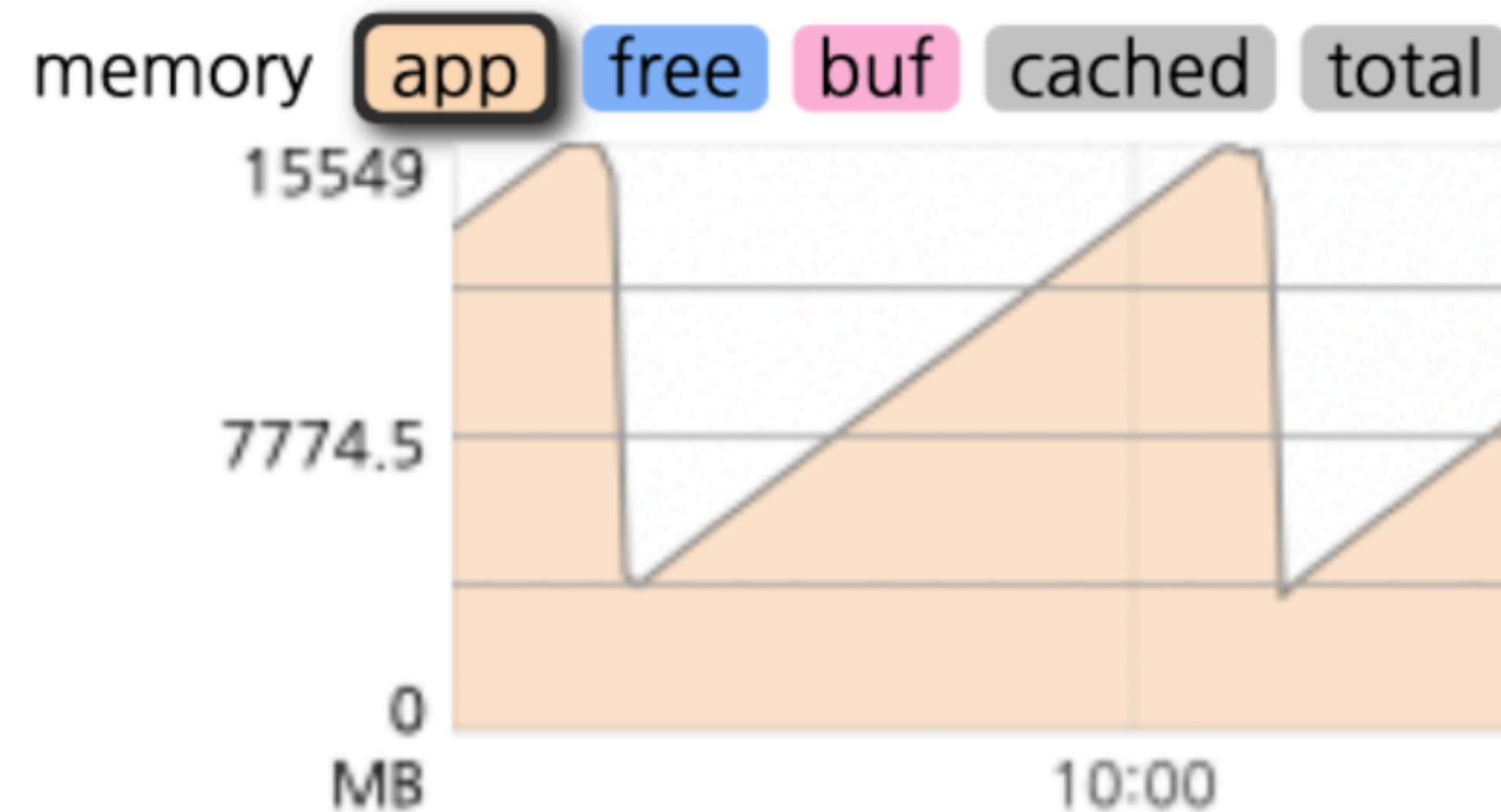
Memory Wave

- GO의 GC는 CPP로부터 올라온 메모리에는 관심이 없습니다

```
t1, _ = NewTable()
```

<=====

```
(npp::sc2::Table *)new npp::sc2::Table(
```



[Seer Collector]

3.8 Calm Down, Mem

후보1) 직접 Delete

```
t1, _ = NewTable()  
defer DeleteTable(t1)
```

- 사용자가 직접 Delete
- 즉각 회수
- GO 서버에서 사용량 일정
- Apache에서 사용량 일정
- Pointer형식 아니어도 됨

후보2) SetFinalizer

```
t1, _ = NewOrgTable()  
runtime.SetFinalizer(t1, t1.Delete)  
return t1
```

- 사용자는 New만 사용
- Queue에 쌓이다가 GC 때 회수(예측불가)
- GO 서버에서 메모리 사용량 조금씩 증가
- Apache에서 사용량 일정(메모리 사용량 UP)
- Pointer만 지원

3.8 Calm Down, Mem

후보1) 직접 Delete

```
t1, _ = NewTable()
defer DeleteTable(t1)
```

- 사용자가 직접 Delete
- 즉각 회수
- GO 서버에서 사용량 일정
- Apache에서 사용량 일정
- Pointer형식 아니어도 됨

후보2) SetFinalizer

```
t1, _ = NewOrgTable()
runtime.SetFinalizer(t1, t1.Delete)
return t1
```

- 사용자는 New만 사용
- Queue에 쌓이다가 GC 때 회수(예측불가)
- GO 서버에서 메모리 사용량 조금씩 증가
- Apache에서 사용량 일정(메모리 사용량)
- Pointer만 지원



3.8 Calm Down, Mem

후보1) 직접 Delete

```
t1, _ = NewTable()  
defer DeleteTable(t1)
```

- 사용자가 직접 Delete
- 즉각 회수
- GO 서버에서 사용량 일정
- Apache에서 사용량 일정
- Pointer형식 아니어도 됨

후보3) SetFinalizer + runtime.GC()

```
t1, _ = NewOrgTable()  
runtime.SetFinalizer(t1, t1.Delete)  
return t1
```

- 사용자는 New만 사용
- 회수 타이밍 조절 가능 하나 퍼포먼스 영향
- GO 서버에서 메모리 사용량 조금씩 증가
- Apache에서 사용량 일정(메모리 사용량 UP)
- Pointer만 지원

3.8 Calm Down, Mem

후보1) 직접 Delete

```
t1, _ = NewTable()
defer DeleteTable(t1)
```

- 사용자가 직접 Delete
- 즉각 회수
- GO 서버에서 사용량 일정
- Apache에서 사용량 일정
- Pointer형식 아니어도 됨

후보3) SetFinalizer + runtime.GC()

```
t1, _ = NewOrgTable()
runtime.SetFinalizer(t1, t1.Delete)
return t1
```

- 사용자는 New만 사용
- 회수 타이밍 조절 가능 하나 퍼포먼스 영향
- GO 서버에서 메모리 사용량 조금씩 증가
- Apache에서 사용량 일정(메모리)
- Pointer만 지원



3.8 Calm Down, Mem

후보1) 직접 Delete

```
t1, _ = NewTable()  
defer DeleteTable(t1)
```

- 사용자가 직접 Delete
- 즉각 회수
- GO 서버에서 사용량 일정
- Apache에서 사용량 일정

후보4) 채널로 자체 Destructor 구현

```
t1, _ = NewOrgTable()  
deleteChan <- t1.Delete  
return t1
```

- 사용자는 New만 사용
- 회수 타이밍 조절 가능 하나 퍼포먼스 영향
- GO 서버에서 사용량 일정(메모리 사용량 UP)
- Apache에서 사용량 일정(메모리 사용량 UP)

3.8 Calm Down, Mem

후보1) 직접 Delete

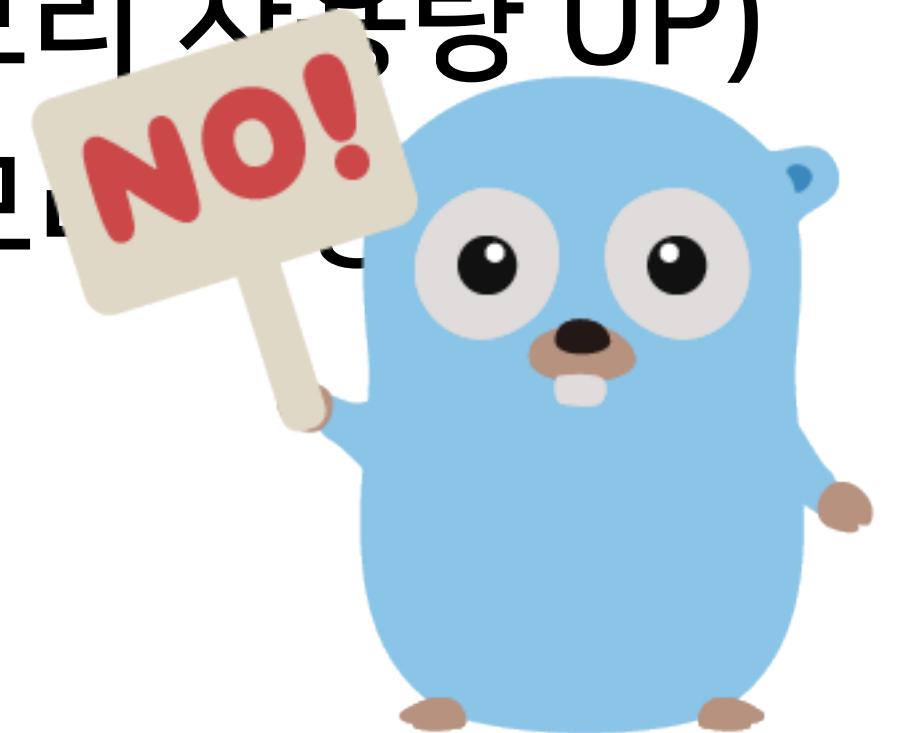
```
t1, _ = NewTable()  
defer DeleteTable(t1)
```

- 사용자가 직접 Delete
- 즉각 회수
- GO 서버에서 사용량 일정
- Apache에서 사용량 일정

후보4) 채널로 자체 Destructor 구현

```
t1, _ = NewOrgTable()  
deleteChan <- t1.Delete  
return t1
```

- 사용자는 New만 사용
- 회수 타이밍 조절 가능 하나 퍼포먼스 영향
- GO 서버에서 사용량 일정(메모리 사용량 UP)
- Apache에서 사용량 일정(메모리 사용량 UP)

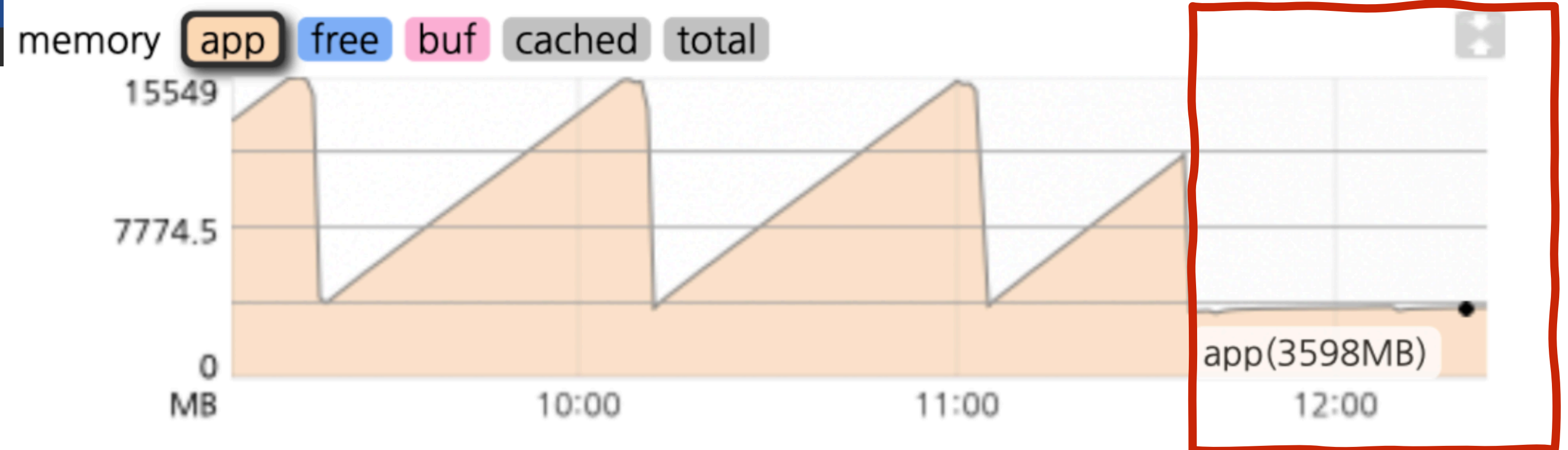


3.8 Calm Down, Mem

후보1) 직접 Delete

```
t1, _ = NewTable()  
defer DeleteTable(t1)
```

- 사용자가 직접 Delete
- 즉각 회수
- GO 서버에서 사용량 일정
- Apache에서 사용량 일정



[Seer Collector]

3.9 Performance

GO가 C++보다 나은 성능을 보여주는가?

3.9 Performance

~~GO가 C++보다 나은 성능을 보여주는가?~~

GO가 동일한 환경에서 C++만큼 처리 할 수 있는가?

- 동일한 장비 수
- 동일한 서버 설정(Apache, etc)

3.9 Performance

GO기반, C++기반 아파치 모듈로 각각 초당 500개의 질의 처리시

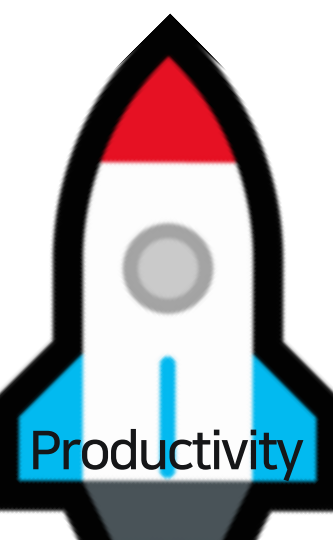
	GO		C++
CPU	9.2%	<	10.9%
Memory	1450MB	>	1080MB
I/O Write	8KB/s	=	8KB/s
Apache Worker	~13	<	~15

3.9 Performance

2020.07 Line Matome 실서비스 전환

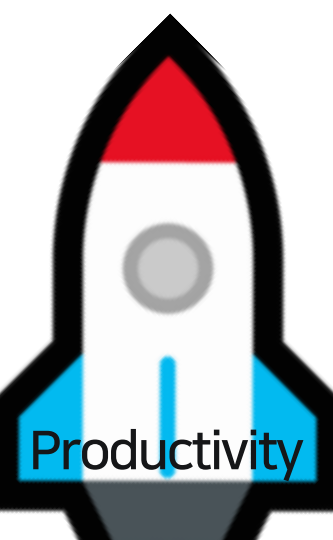


4. You Can GO

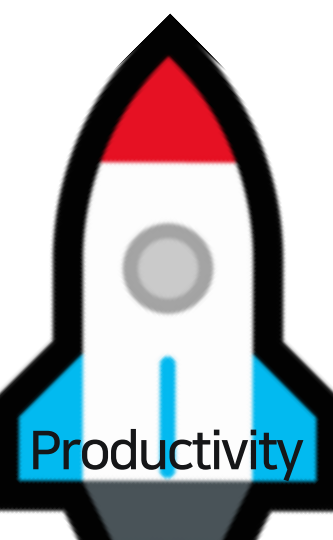


4.0 개발부터 실서비스 적용까지

- 2020.02.14 Kick-off(김준영, 김덕일, 장원희, 조현준)
- 2020.05.17 v0.1.0 릴리즈
- 2020.07.02 첫번째 실서비스 GO 기반 모듈로 전환 완료
- 2020.07.10 v1.0.0 릴리즈
- 2020.10.16 v1.3.2 릴리즈



프로젝트를 마치고 보니
생산성이 증가 되었는가?



4.1 생산성 - 빠르게 효율적으로 목표 달성

- 새 멤버가 실제 업무에 투입되기까지의 노력과 시간(적게!)
#러닝커브 #코드리뷰 #빌드환경 #개발문서
- 중복 업무가 아닌 각자의 고유 업무에 들이는 노력과 시간(많게!)
#코드재사용 #이슈공유
- 기능 개발 자체에 사용하는 노력과 시간(많게!)
#러닝커브 #개발문서 #코드리뷰
- 서비스에 투입할 만한 코드가 나오기까지 걸리는 노력과 시간(적게!)
#테스트 #성능테스트 #코드배포 #코드버저닝

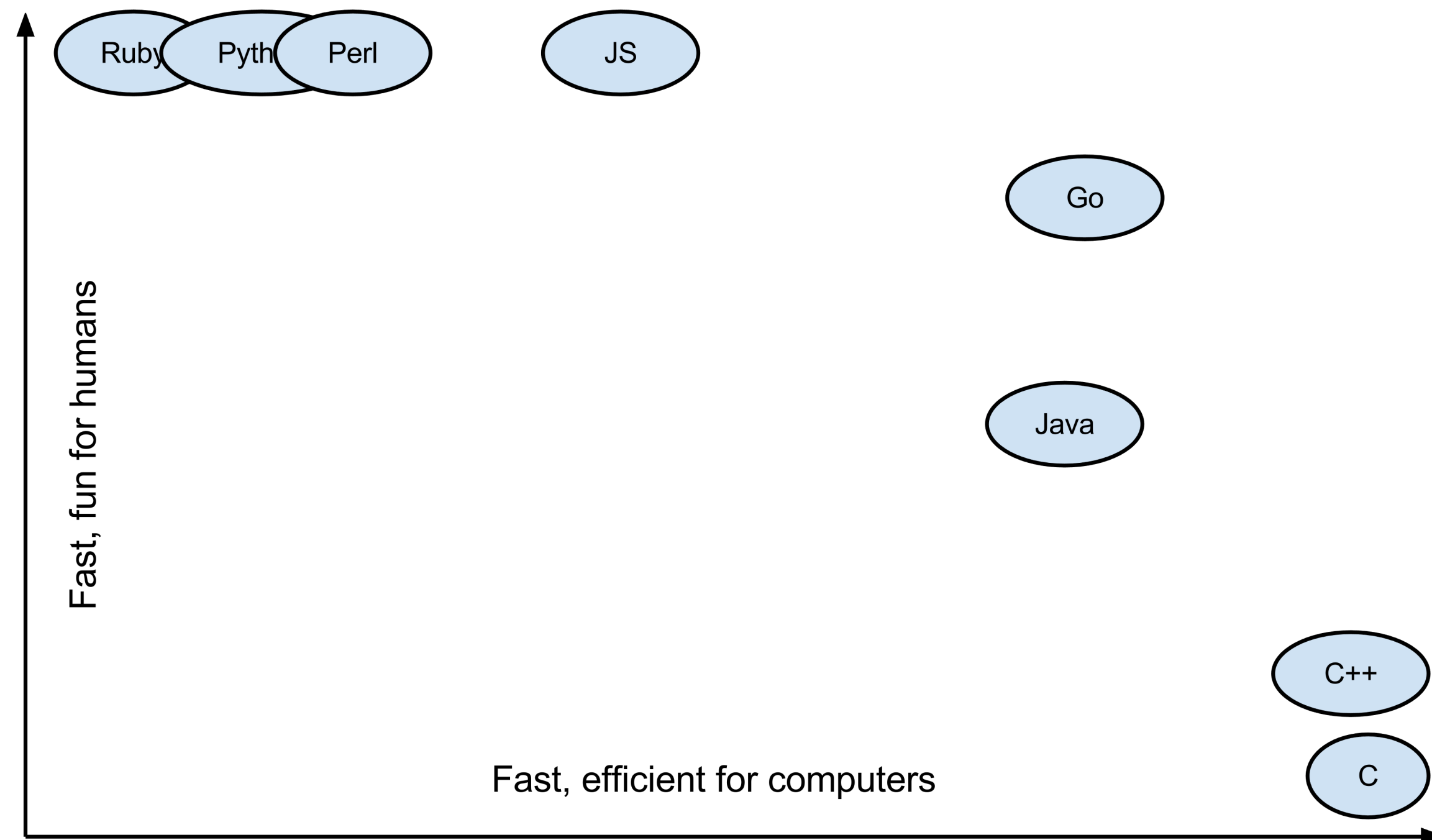


4.2 Easy, Fun, Fast

#러닝커브 #개발문서 #코드리뷰

Language Specification

- GO 100 pages
- C++20 1841 pages
(Core 500 pages)



Brad Fitzpatrick GoCon 2014



4.3 Hello, GO World!

#빌드환경 #테스트 #성능테스트 #개발문서

빌드, 테스트 그리고 문서화

- go build, go test(-bench), godoc
- Example* => 테스트 + 문서화

```
func ExampleBool_GetBool() {
    value := parser.Parse(str: "false")
    fmt.Println(value)
    // Output: false
}
```

=

▼ Example

Code:

```
value := parser.Parse("false")
fmt.Println(value)
```

Output:

```
false
```



4.4 코드 품질, 컨벤션

#러닝커브 #코드리뷰

간결한 Compiler, 통일된 Style

- gofmt, goimport





4.5 패키지 배포 그리고 버저닝

#코드배포 #코드버저닝 #이슈공유 #코드재사용

Github 기반 배포와 go.mod

- Repository에 접근만 가능하면 한 줄로 import
- Release 기반으로 배포자/사용자 모두 편리한 버전 관리
- 자연스러운 이슈 공유 및 파악



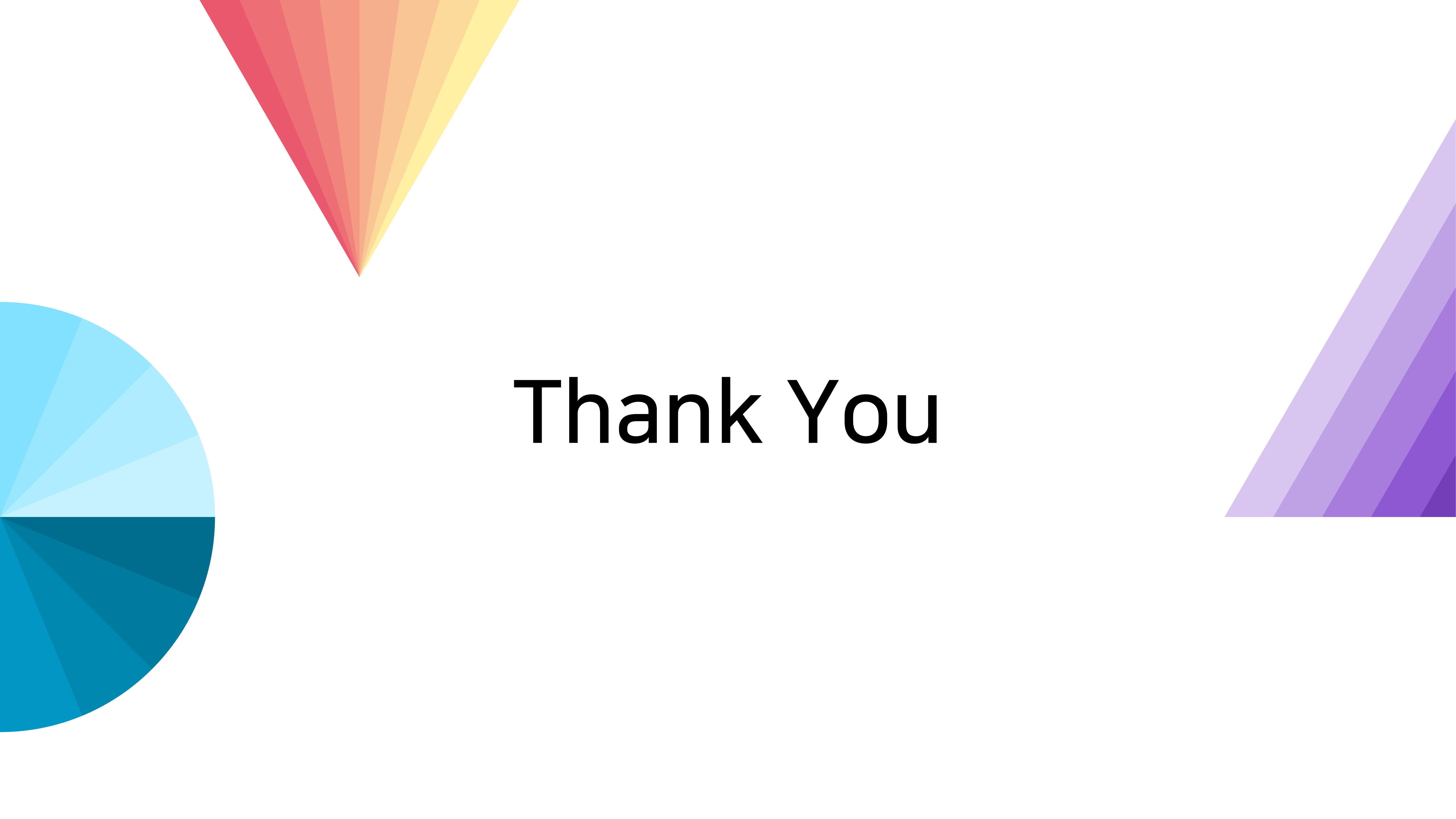
4.6 GO객의 소리

- GO언어 쪽이 기존 **로직 파악**하는데 더 수월 했다.
- 다른 개발자의 코드를 **쉽게 Import**할 수 있다.
- 어메이징한 **defer**
- GO에서는 마음이 급하더라도 **Test Driven Development**를 하게 된다.
- C++은 표준에 따라 작성된 코드가 (많이) 다른 편인데, GO는 그런 현상이 적다.
- 파워풀한 GO **표준 라이브러리**(특히 Marshaling)
- GO가 단순하니 이제는 **타 언어가 복잡해** 보인다.
- C++에 비해서 **학습량이 적어서**, 바로 서비스 개발에 적용 할 수 있었다.
- 신규 기능 개발 시 **부담감이 줄어**들었다.

Give it a GO!



Q & A



Thank You

References

- <http://www.swig.org/Doc4.0/SWIG.html>
- <https://github.com/swig/swig/tree/master/Examples/go>
- <https://www.softwariness.com/articles/api-design-for-swig>
- <https://github.com/Softwariness/SwigUsageExample>
- <https://github.com/golang/go/tree/master/misc/swig/callback>
- <https://github.com/jsolmon/go-swig-exceptions>
- <https://github.com/garrettsickles/goapache>
- <https://medium.com/a-journey-with-go/go-finalizers-786df8e17687>
- <https://github.com/swig/swig/issues/289>