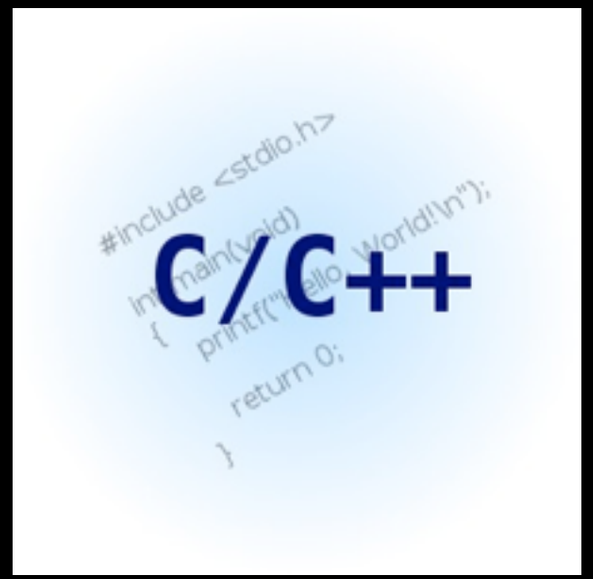
Why Go

Võ Anh Duy
@voanhduy1512

Agenda

- Why another language ?
- Enter Go
- Q & A

Why another language?


#include <stdio.h>
int main(void)
{
printf("Hello, World!\n");
return 0;
}
C/C++


python™


php


C#


Java™

What will you chose?

	C/C++	C#/Java	Ruby/Python/ Php
Build Time	slow	efficient	
Speed	fast	not so fast	slow
Programming experience	terrible	good	awesome

What will you chose?

C/C++	C#/Java	Ruby/Python/Php
efficient execution	efficient compilation	ease of programming

“When builds are slow, there is time to think. The origin myth for Go states that it was during one of those 45 minute builds that Go was conceived.”

Rob Pike,
Go at Google: Language Design in the Service of Software Engineering
https://talks.golang.org/2012/splash.article

Why another language?

- Computers are enormously quicker but software development is not faster.
- Dependency management is a big part of software development today but the “header files” of languages in the C tradition are antithetical to clean dependency analysis—and fast compilation.
- There is a growing rebellion against cumbersome type systems like those of Java and C++, pushing people towards dynamically typed languages such as Python and JavaScript.
- Some fundamental concepts such as garbage collection and parallel computation are not well supported by popular systems languages.
- The emergence of multicore computers has generated worry and confusion.

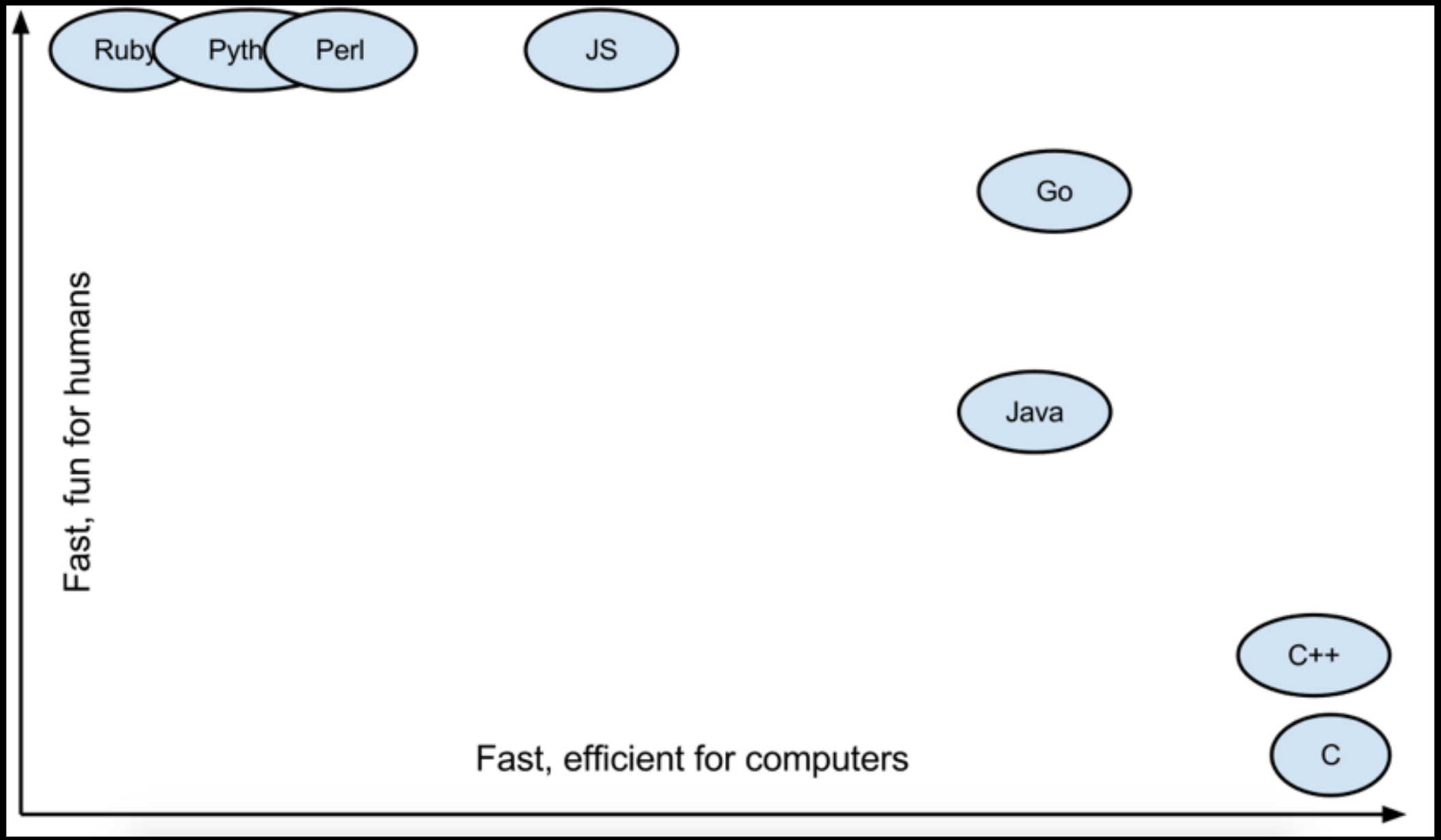
Enter Go

Go’s targets

- It is possible to compile a large Go program in a few seconds on a single computer.
- Go provides a model for software construction that makes dependency analysis easy and avoids much of the overhead of C-style include files and libraries.
- Go's type system has no hierarchy, so no time is spent defining the relationships between types. Also, although Go has static types the language attempts to make types feel lighter weight than in typical OO languages.
- Go is fully garbage-collected and provides fundamental support for concurrent execution and communication.
- By its design, Go proposes an approach for the construction of system software on multicore machines.

Go’s targets

- Fast build
- Approach the performance of C
- Ease of programming
- Effective package management
- Networked-communication, concurrency and parallelization



http://talks.golang.org/2014/gocon-tokyo/funfast.svg

Design principles

- Minimal amount of keywords
- Minimalist approach
- Explicit specification

Characteristics

- Imperative language
- Object-oriented: yes and no
- Statically typed, strongly typed
- No type hierarchy
- Compile to native code
- Concurrency
- Dynamically (through var)

Syntax

```go
// Package stringutil contains utility functions for working with strings.
package stringutil

// Reverse returns its argument string reversed rune-wise left to right.
func Reverse(s string) string {
  r := []rune(s)
  for i, j := 0, len(r)-1; i < len(r)/2; i, j = i+1, j-1 {
    r[i], r[j] = r[j], r[i]
  }
  return string(r)
}
```

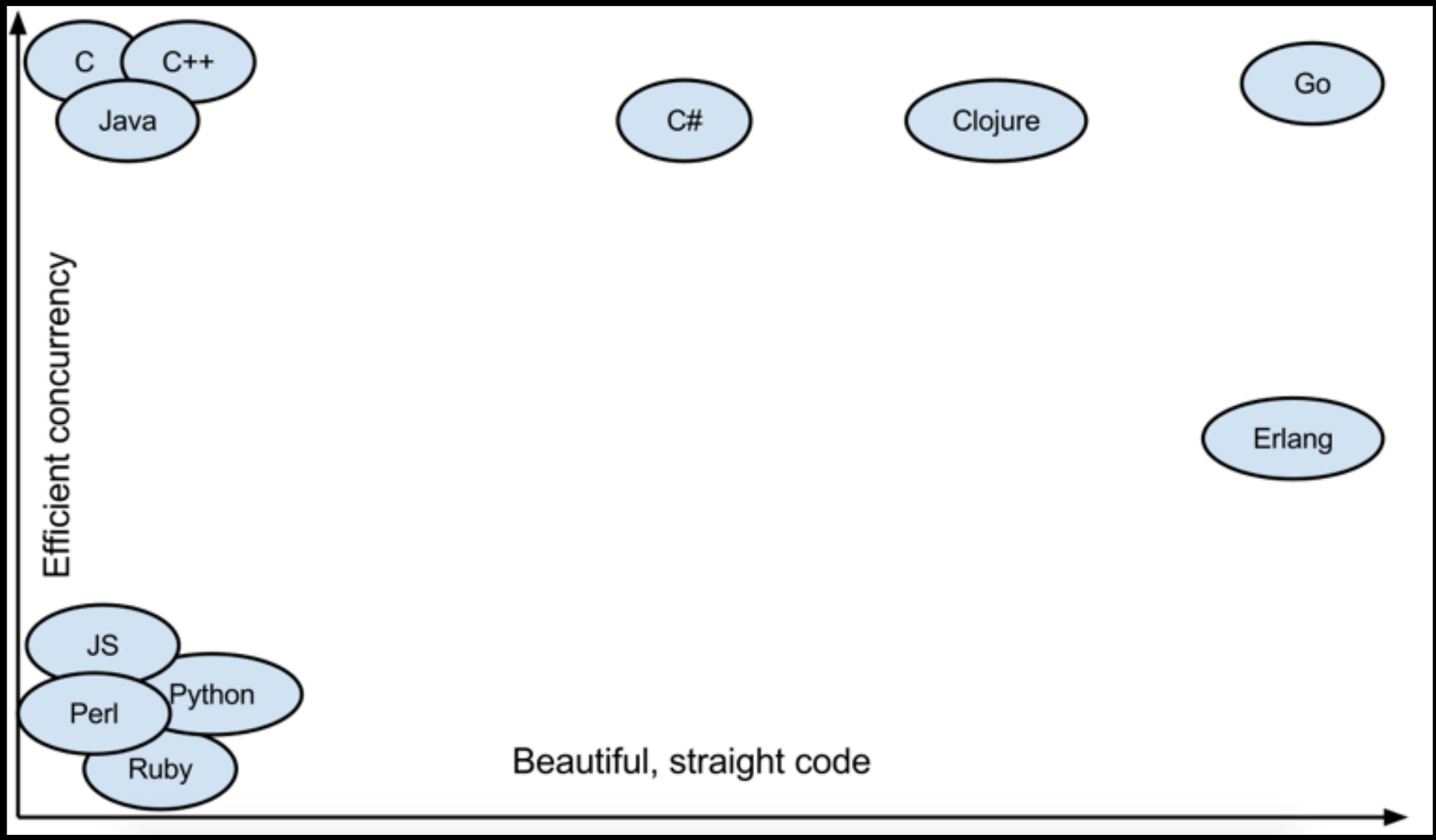
Package

- Packages define boundaries for compilation and reuse.
- Packages are directories containing source code.
- The unit of compilation is the package, not the file.
- You import a whole package, not a type, or a symbol.

```
import "fmt"

func main() {
    fmt.Printf("Hello %s\n", "world)
}
```

Concurrency



http://talks.golang.org/2014/gocon-tokyo/funfast.svg

Goroutines

- The go statement launches a function call as a goroutine
- A goroutine runs concurrently (but not necessarily in parallel)
- A goroutine is a thread of control within the program, with its own local variables and stack. Much cheaper to create and schedule than operating system threads.

```
go f()
go f(x, y, ...)
```

Channels

```
var ch chan int
ch := make(chan int)  // declare and initialize with newly made channel
```

```
ch <- 1   // send value 1 on channel ch
x = <-ch  // receive a value from channel ch (and assign to x)
```

Use cases

Use case

- Run Fast but easy to write
- Highly Concurrency
- No deployment pain

Who use go?

- BBC Worldwide - source
- Beachfront Media article
- Betable - talk #1, talk #2
- Bitbucket - source
- bitly - github blog
- Canonical - source
- Carbon Games - source
- CloudFlare - blog article
- Cloud Foundry - blog github
- CloudWalk - github
- clypd - blog
- Conformal Systems - blog post github
- Crashlytics - tweet
- Cupcake - tweet github
- CustomerIO - tweet
- Digital Ocean - blog
- Disqus - blog
- DNSimple - blog
- domainr - tweet
- dotCloud - docker slides
- drone.io - post github
- Dropbox - blog, github
- Embedly - blog
- ErrPlane - blog
- Foize - github
- Flipboard - source (job post)
- Getty Images - tweet tweet
- GitHub - blog post

- https://code.google.com/p/go-wiki/wiki/GoUsers

Q & A