* A new open source language
* A concurrent garbage collected language
* Builds large programs fast.

Not owned by Google.
“Go compilers produce fast code fast. Typical builds take a fraction of a second yet the resulting programs run nearly as quickly as comparable C or C++ code.”
“Go is type safe and memory safe. Go has pointers but no pointer arithmetic. For random access, use slices, which know their limits.”
“Do not communicate by sharing memory. Instead, share memory by communicating.”

“Go promotes writing systems and servers as sets of lightweight communicating processes, called goroutines, with strong support from the language. Run thousands of goroutines if you want—and say good-bye to stack overflows.”
Go ~ Open Source

BSD licence code

Creative commons attribution documentation

Majority of committers are outside Google
“Go has fast builds, clean syntax, garbage collection, methods for any type, and run-time reflection.

It feels like a dynamic language but has the speed and safety of a static language.

It's a joy to use.”
Go ~ Deeper

- Interfaces ~ pure classless duck-typing
- Reflection ~ strong static types
- Higher-order functions & closures
- Concurrency ~ CSP

✗ Immutability
✗ Genereics
package main

import "fmt"

func main() {
    fmt.Println("Hello, 世界")
}
A type implements an interface by defining the required methods.

Example: Printing Stringers
Docs: fmt.Stringer
Go ~ Interfaces 2

- Eg. http://golang.org/pkg/time/#Weekday
- Eg. http://play.golang.org/p/FBIEp60oXP

- No 'implements' declarations
  - These types don't declare that they implement fmt.Stringer
  - They just do, simply by declaring a String() string method.

- Go interfaces are simple, lightweight entities.
  - Very little coupling
  - Can be added later
Go ~ IO Writer

- Writing bytes: `io.Writer`
  ```go
type Writer interface {
    Write(p []byte) (n int, err error)
}
```
- Example: `os.Stdout`
- Example: `32bit CRC`
- Example: `using MultiWriter`
Go ~ IO Reader

- Reading bytes: `io.Reader`
  
  ```go
  type Reader interface {
      Read(p []byte) (n int, err error)
  }
  ```

- `os.File`
- `bufio.Reader`
- `net.Conn`
- `Compress/gzip, crypto/tls, ...`
- `bytes.Buffer`
Go ~ Reflection

- Type information & basic operations are available at runtime
- A little goes a long way
  - Only needed in one or two key places
- Example – implementation of `Printf` as ordinary Go code
- Lots of flexibility, e.g. for JSON and XML processing
  - Step 1: printing a struct
  - Step 2: JSON and XML marshalling
  - Step 3: generalisation with higher-order function
    - note `Lang` didn't need anything clever to map it to JSON or XML
    - JSON and XML marshalling libs contain some reflection code
Go ~ Web Crawl

- Example: toy web crawler counting webpage bytes (ex10webcrawl1)

  Python 20806 [2.95s]
  Ruby 10330 [1.94s]
  Scala 46318 [0.71s]
  Go 6400 [0.42s]
  6.01s total

- But we could make these requests in parallel and speed it all up...
Go ~ Concurrency

● Parallelism
  – Running multiple things at the same time

● Concurrency
  – A way to deal with multiple things simultaneously
  – The coordination of parallel computations

● Go provides both but the emphasis is on concurrency
Go ~ Concurrency

- **Goroutines** let you run multiple computations simultaneously
- **Channels** let you coordinate the computations, by explicit communication.
Go ~ Concurrent Web Crawl

- Step 1: toy web crawler with goroutines ... and a rubbish sleep at the end (ex11webcrawl2)
- Step 2: add some channels. No delays needed – Nice! (ex12webcrawl3)
- Step 3: what if we want to give up after n seconds? Simple: use a timeout with a select (aka. 'alternative' in CSP speak) (ex13webcrawl4)
  - No callbacks
  - No condition variables, mutexes, semaphores
    - although they exist under the hood in the runtime
  - What you see is what you get – simple!
Go ~ closing remarks

- Go ~ a fast, fun and productive language
- V1.1.2 released Aug 2013
- Choice of two compilers coordinated by spec
- Medium-sized community of open-source add-on apis
- Eclipse & IntelliJ plugins ('Goclipse' is currently better)
- Testing, debugging, profiling tools
  - quite usable but not very snazzy yet.
- See A Tour of Go, Russ Cox (the basis of this talk)
- Start here:
  - http://tour.golang.org/
  - https://bitbucket.org/rickb777/go-talk