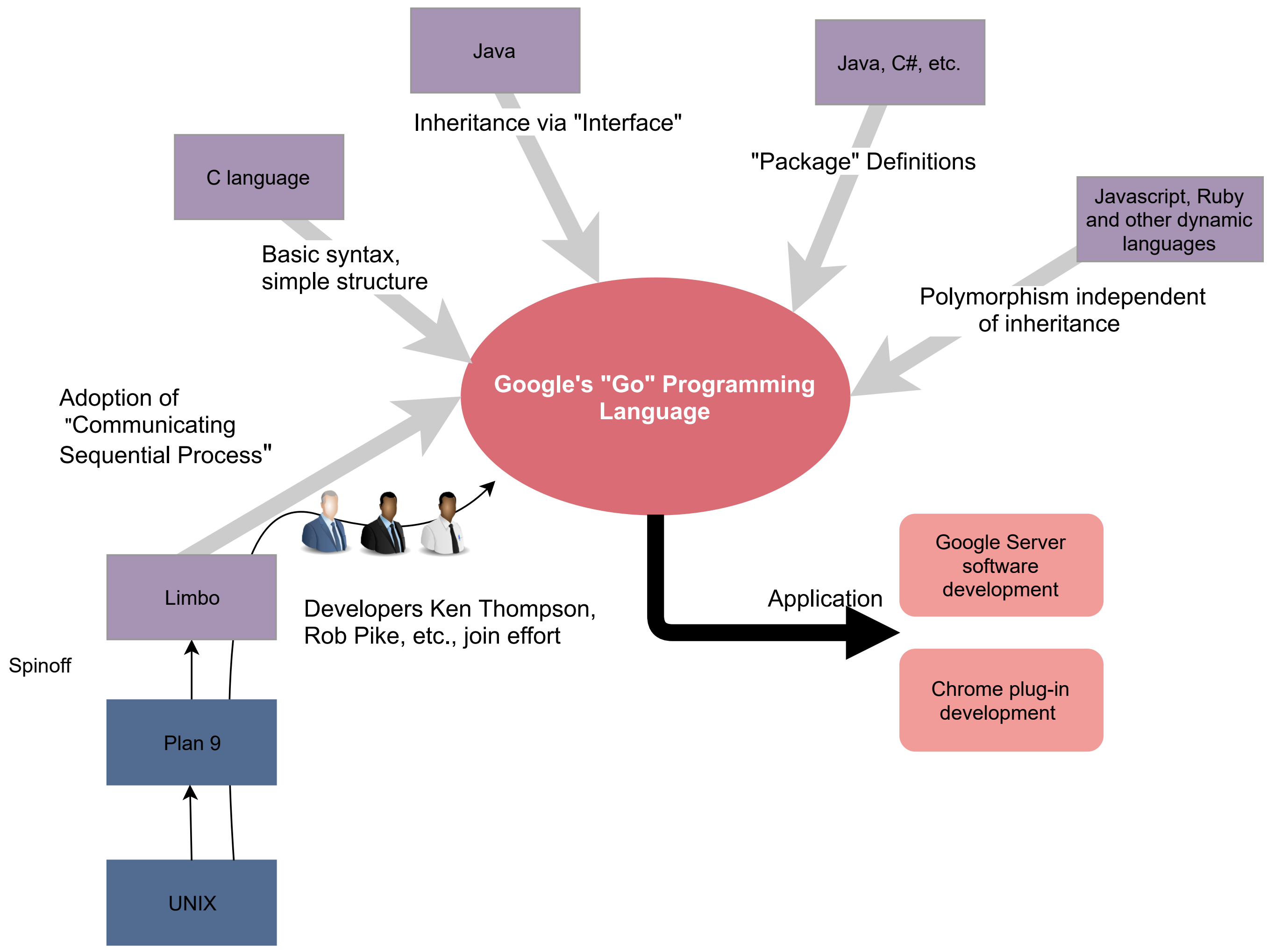
Reasons for Developing Go

This lesson covers some reasons why there was a need for Go when other languages were doing their job.

Languages that influenced Go

Go belongs to the *C-family*, like C++, Java, and C#, and is inspired by many other languages created and used by its designers. The resemblance with the syntax of C language was maintained to gain an advantage of familiarity among developers. However, compared to C/C++, the syntax is made more incisive (concise). Additionally, Go has the features of a dynamic language, so Python and Ruby programmers feel more comfortable with it. The following figure shows some of the influences on the Go programming language:


Java
Java, C#, etc.
Inheritance via "Interface"
"Package" Definitions
C language
Javascript, Ruby and other dynamic languages
Basic syntax, simple structure
Polymorphism independent of inheritance
Google's "Go" Programming Language
Adoption of "Communicating Sequential Process"
Google Server software development
Application
Limbo
Developers Ken Thompson, Rob Pike, etc., join effort
Chrome plug-in development
Spinoff
Plan 9
UNIX

Influence on Go

Why a new language?

Following are the reasons that led to the development of Go:

- Evolving with computing landscape
- Need for faster software development
- Need for efficiency and ease of programming

Let’s discuss each need one by one.

Evolving with computing landscape

Programming languages like C/C++ did not evolve with the computing landscape, so there is a need for a new systems language, appropriate for the needs of our computing era.

Need for faster software development

In contrast to computing power, software development has not become considerably faster or more successful (considering the number of failed projects), whereas applications still grow in size. Therefore, a new low-level language was needed, equipped with higher concepts.

Need for efficiency and ease of programming

Before Go, a developer had to choose between fast execution but slow and inefficient building (like C++) or efficient compilation but not so fast execution (like .NET or Java), or ease of programming but slower execution (like dynamic languages such as Python, Ruby or JavaScript). Go is an attempt to combine all the *three* wishes: efficient and fast compilation, fast execution, and ease of programming.

Targets of Go

The main target of Golang's design was to combine the *efficacy, speed, and safety of a statically typed and compiled language* with the *ease of programming of a dynamic language* to make programming more fun again.

Some other targets that Go was meant to meet were:

- Support for network communication, concurrency, and parallelization
- Support for excellent building speed
- Support for memory management

Support for network communication, concurrency, and parallelization

To get the most out of distributed and multi-core machines excellent support for networked-communication, concurrency, and parallelization. Golang was expected to achieve this target for internal use in Google, and this target is achieved through the concepts of **goroutines**. Don't worry if you do not have an idea about goroutines. We'll cover them in Chapter 12 of this course. It is undoubtedly a great stronghold of Go as compared to other programming

languages keeping in mind the importance of multicore and multiprocessor computers and the minimal support for them in existing programming languages.

Support for excellent building speed

There was a growing concern to improve the *building speed* (compilation and linking to produce machine code) of C++ projects, which are heavily used in Google's infrastructure. This concern gave birth to the idea of developing the Go programming language. In particular, *dependency management* is a very important part of software development today. The "header files" of languages caused considerable overhead leading in order to build times of hours for the biggest projects. Developers felt the need for clean dependency analysis and fast compilation, which Go language provides with its **package model**.

The entire Go standard library compiles in less than 20 seconds. Typical projects compile in half a second. This lightning-fast compiling process is even faster than C or Fortran, making compilation a non-issue. Until now, this was regarded as one of the great benefits of using dynamic languages for development because the long compile/link step of C++ could be skipped. However, with Go, this is no longer an issue! Compilation times are negligible, and we have the same productivity as in the development cycle of a scripting or dynamic language. In addition to this, the execution speed of the native code is comparable to C/C++.

Support for memory management

Because memory problems (memory leaks) are a long-time problem of C++, Go's designers decided that memory management should not be the responsibility of a developer. So although Go executes native code, it runs in a small runtime, which takes care of an efficient and fast garbage collection. Go also has a built-in runtime reflection capability.

That's it about the reasons and targets of Go. Now let's study Go's characteristics.