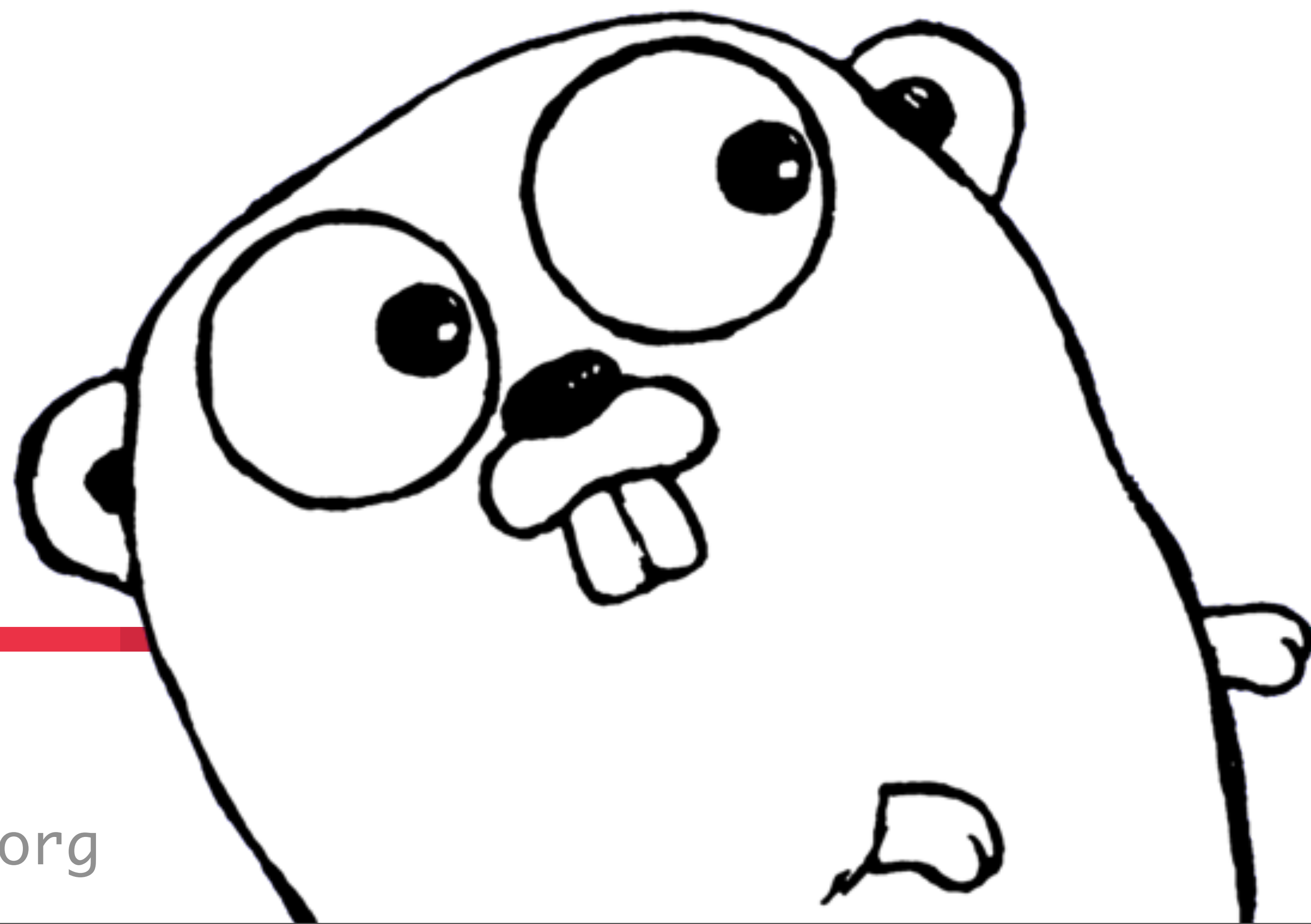
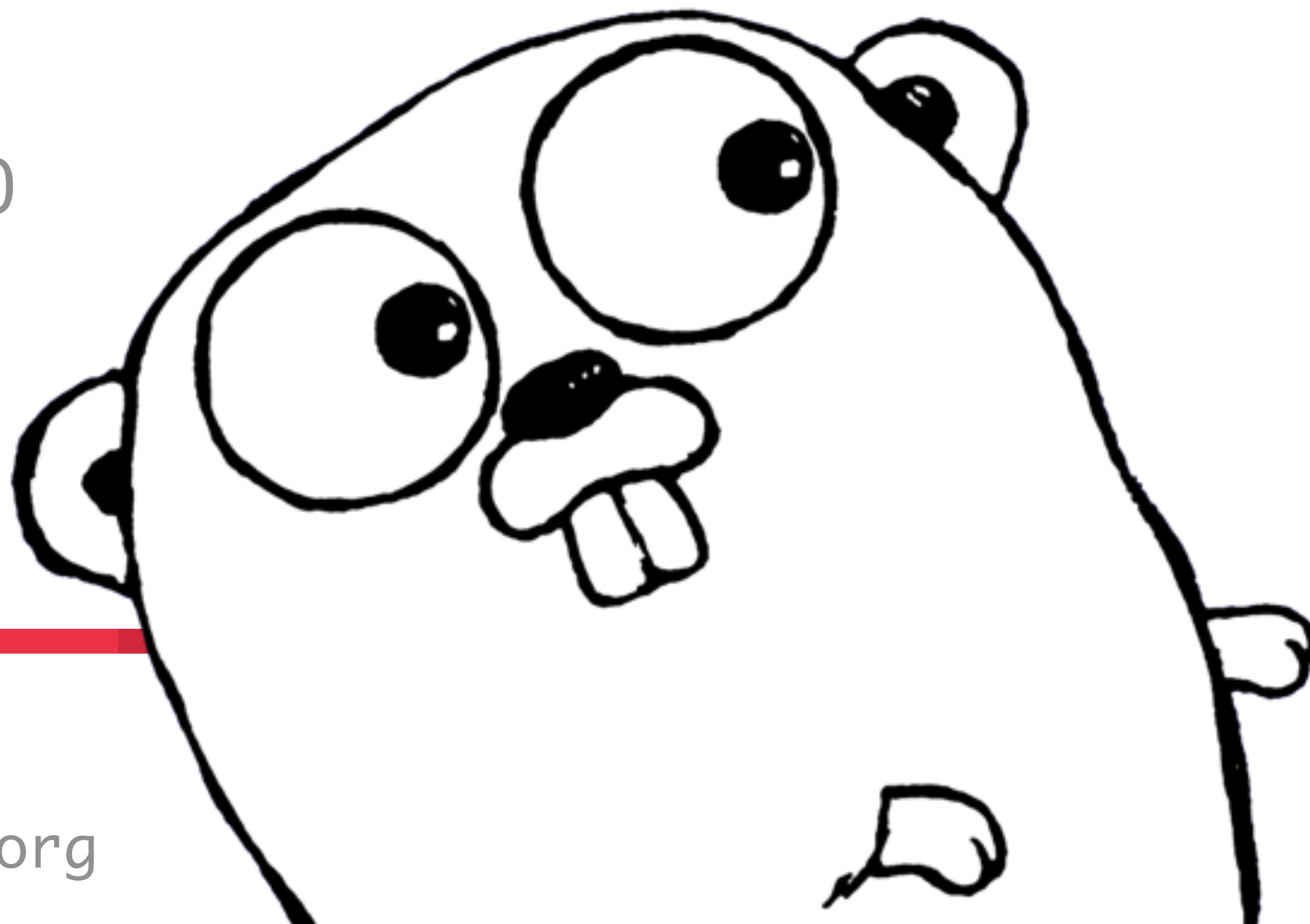


http://golang.org

Public Static Void

Rob Pike
OSCON
July 22, 2010



http://golang.org

History

I'm always delighted by the light touch and stillness of early programming languages. Not much text; a lot gets done. Old programs read like quiet conversations between a well-spoken research worker and a well-studied mechanical colleague, not as a debate with a compiler. Who'd have guessed sophistication bought such noise?

- Dick Gabriel

Sophistication

If more than one function is selected, any function template specializations in the set are eliminated if the set also contains a non-template function, and any given function template specialization F1 is eliminated if the set contains a second function template specialization whose function template is more specialized than the function template of F1 according to the partial ordering rules of 14.5.6.2. After such eliminations, if any, there shall remain exactly one selected function.
(C++0x, §13.4 [4])

Sophistication

Which Boost templated pointer type should I use?

```
- linked_ptr
- scoped_ptr
- shared_ptr
- smart_ptr
- weak_ptr
- intrusive_ptr
- exception_ptr
```


Google

Noise

```
public static <I, O> ListenableFuture<O> chain
(ListenableFuture<I> input, Function<? super I, ?
extends ListenableFuture<? extends O>> function)
dear god make it stop
```

- a recently observed chat status

```
foo::Foo *myFoo = new foo::Foo(foo::FOO_INIT)
```

- but in the original `Foo` was a longer word

Google

How did we get here?

A personal analysis:

1) C and Unix became dominant in research.

2) The desire for a higher-level language led to C++, which grafted the Simula style of object-oriented programming onto C. It was a poor fit but since it compiled to C it brought high-level programming to Unix.

3) C++ became the language of choice in parts of industry and in many research universities.

4) Java arose as a clearer, stripped-down C++.

5) By the late 1990s, a teaching language was needed that seemed relevant, and Java was chosen.

Google

Programming became too hard

These languages are hard to use.

They are subtle, intricate, and verbose.

Their standard model is oversold, and we respond with add-on models such as "patterns".

(Norvig: patterns are a demonstration of weakness in a language.)

Programming as bureaucracy

Every step in the program must be justified to the compiler.

The repetition in

```
foo::Foo *myFoo = new foo::Foo(foo::FOO_INIT)
```

is like filling out a form in triplicate.

Repetition became a placeholder for safety.

Programs end up looking like credit card applications.

Google

Bureaucracy is institutionalized

```
static {
    defaultPorts.put("http",       new Integer(80));
    defaultPorts.put("shttp",      new Integer(80));
    defaultPorts.put("https",      new Integer(443)); // ...
    usesGenericSyntax.put("http",  Boolean.TRUE);
    usesGenericSyntax.put("https", Boolean.TRUE);
    usesGenericSyntax.put("shttp", Boolean.TRUE);   // ...
}

private static final boolean pathsEqual(String p1, String p2)
```

This is the look of industrial programming.
Why do we type so much?
Why is this style not bothersome to programmers?
Why don't the languages help?

Yet somehow... these languages are successful and vital.

Google

A reaction

The inherent clumsiness of the main languages has caused a reaction.

A number of successful simpler languages (Python, Ruby, Lua, JavaScript, Erlang, ...) have become popular, in part as a rejection of the standard languages.

Some beautiful and rigorous languages designed by domain experts (Scala, Haskell, ...) have also arisen, although they are not as widely adopted.

So despite the standard model, other approaches are popular and there are signs of a growth in "outsider" languages, a renaissance of language invention.

Google

A confusion

The standard languages (Java, C++) are statically typed.

Most outsider languages (Ruby, Python, JavaScript) are interpreted and dynamically typed.

Perhaps as a result, some programmers have confused "ease of use" with interpretation and dynamic typing.

This confusion arose because of how we got here: grafting an orthodoxy onto a language that couldn't support it cleanly.

The world has split into a false dichotomy: nice, interpreted, dynamic versus ugly, compiled, static.

Time to put things right.

Google

The good

The standard languages are very strong: type-safe, effective, efficient.

In the hands of experts, they are great.

Huge systems and large companies are built on them.

In practice they work well for large scale programming: big programs, many programmers.

Google

The bad

The standard languages are hard to use.

Compilers are slow and fussy. Binaries are huge.

Effective work needs language-aware tools, distributed compilation farms, ...

Many programmers prefer to avoid them.

These languages are at least 10 years old and poorly adapted to the current computing environment: clouds of networked multicore CPUs.

Google

Flight to the suburbs

This is partly why Python et al. have become so popular:
They don't have much of the "bad".

- dynamically typed (fewer noisy keystrokes)
- interpreted (no compiler to wait for)
- good tools (interpreters make things easier)

But they also don't have the "good". They are

- slow
- not type-safe (static errors occur at runtime)
- very poor at scale

And they're also not very modern.

A niche

There is a niche to be filled: a language that has the good, avoids the bad, and is suitable to modern computing infrastructure:

- comprehensible
- statically typed
- light on the page
- fast to work in
- scales well
- doesn't require tools, but supports them well
- good at networking and multiprocessing

Google

Go

An attempt to fill the niche

Google

The target

Go aims to combine the safety and performance of a statically typed compiled language with the expressiveness and convenience of a dynamically typed interpreted language.

It also aims to be suitable for modern systems - large scale - programming.

Google

How does Go fill the niche?

General purpose

Concise syntax

Expressive type system

Concurrency

Garbage collection

Fast compilation

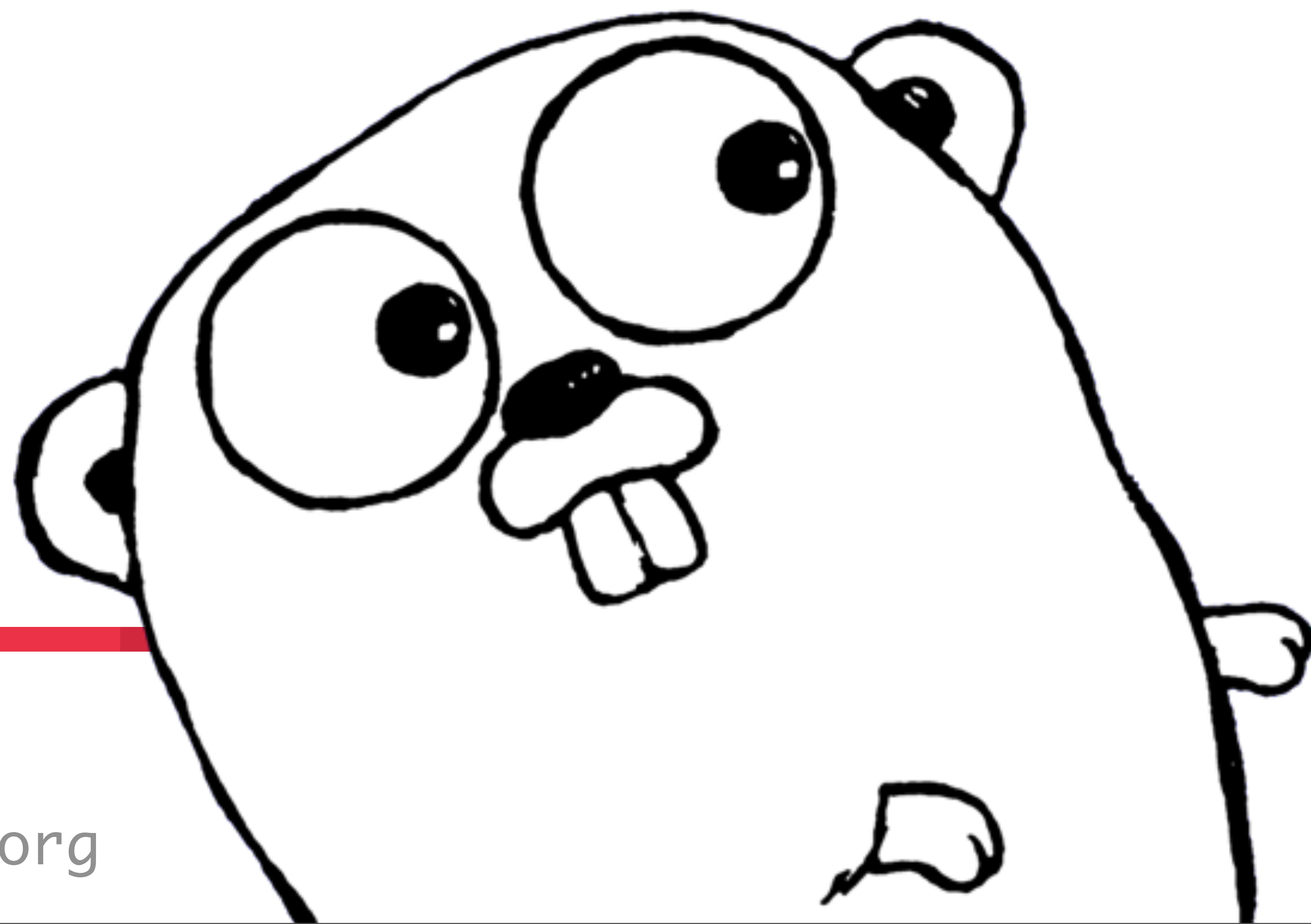
Efficient execution

And of course... Open Source

More details in my next talk

And at `http://golang.org`

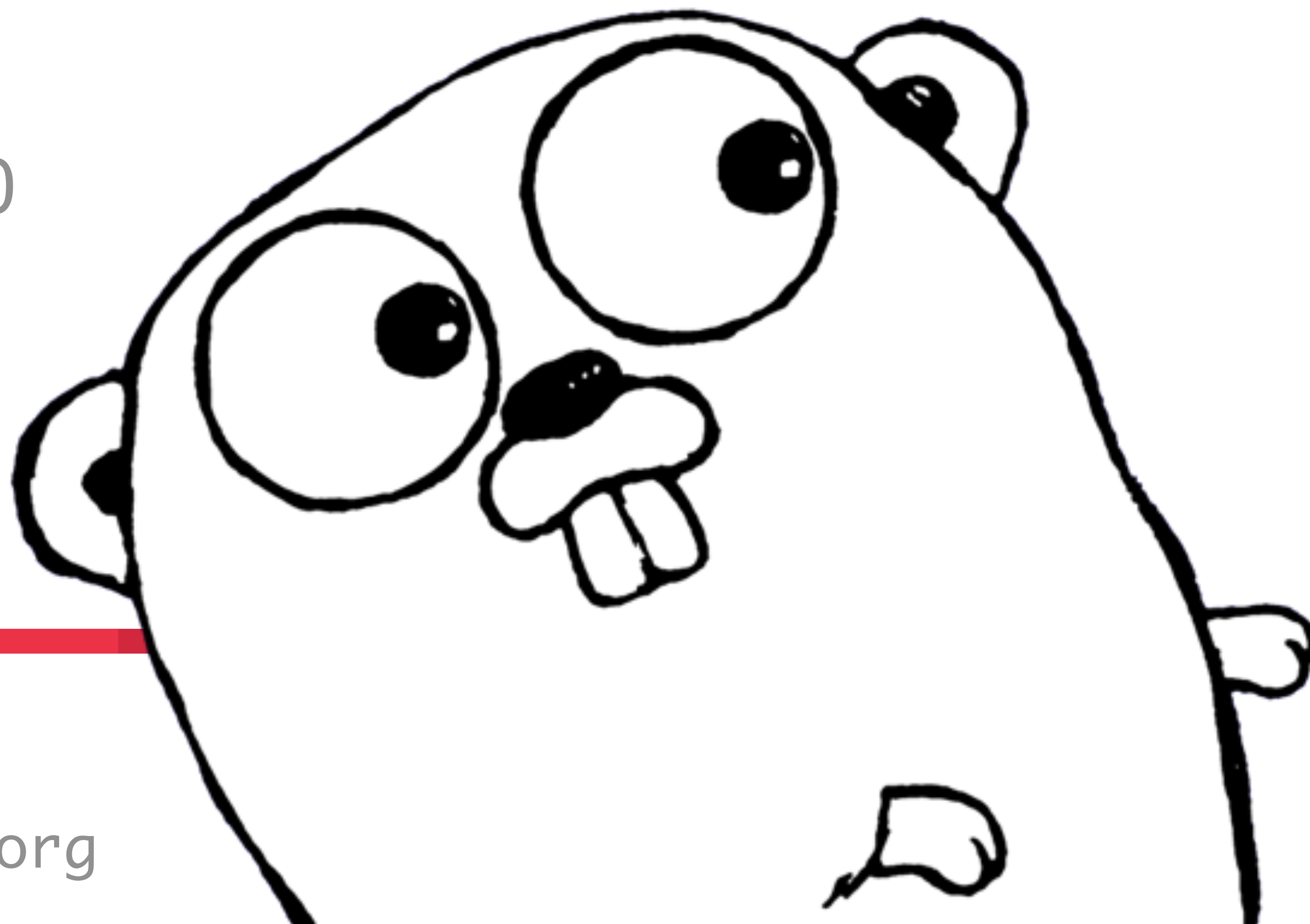
Google



http://golang.org

Public Static Void

Rob Pike
OSCON
July 22, 2010



http://golang.org