# Rust programming language

(a.k.a. “Project Servo”)

Technology from the past,
come to save the future
from itself.

Mozilla Annual Summit, July 2010

<graydon@mozilla.com>

# Oh no, not again

- Yes, I'm terribly sorry
- It's not for the web
- it's not for rapid prototyping
- it's not for casual programming
- It's not for lone genius hackers
- It's not for “managed runtimes”
- If you need these things, there are 9000 other languages, pick one of them.

# Ok then, for what?

It's for writing
**large**,
**systems-level,**
**concurrent**
programs that are
**very safe**,
**predictable**,
**maintainable**,
and
**efficient**.

# Hyperbole

Many languages get written to show off
one new favored feature
or to explore a clever paradigm
and everything else
is an afterthought
a cobbled-together mess.

# Bold claims

Rust does the opposite.

There is **nothing new** in Rust, at all.

Intent is to pick stuff widely known to be good, and be thorough, not botch any one part so bad.

To have a language that doesn't make us cry.

# *Nothing* new?

- Hardly anything. Maybe a keyword or two.
- Many older languages *better* than newer ones:
  - eg. Mesa (1977), BETA (1975), CLU (1974) ...
    - We keep forgetting already-learned lessons
- Rust picks from 80s / early 90s languages:
  - **Nil** (1981), **Hermes** (1990)
  - **Erlang** (1987)
  - **Sather** (1990)
  - **Newsqueak** (1988), **Alef** (1995), **Limbo** (1996)
  - **Napier** (1985, 1988)

# Details! (#1)

- **Static** safety:
  - memory safety, *no wild pointers*
  - typestate system*, no null pointers*
  - mutability control, *immutable by default*
  - side-effect control, *pure by default*

# Details! (#2)

- **Dynamic** safety:
  - Bounds-checked indexing, trapped signals, etc.
  - Dynamic *assertions* drive typestates
  - All errors cause *failure*, unwinding
    - “Expected errors”? Use a disjoint union return
  - Failure of a task is *non-recoverable*
    - “Crash-only” tasks with isolation, trapping
    - Pervasive logging, annotations for unwinding
    - Supervision / restart task ownership tree

# Details! (#3)

- **Structural** type bestiary:
  - Records, tuples, vectors
  - Tagged disjoint unions
  - First class functions (with bindings)
  - *Structural* objects
    - Lightweight
    - Immutable by default also
    - No classes, no class hierarchy
    - Just object types and objects that conform to them

# Details! (#4)

- **Actor** language bestiary:
  - Lightweight tasks (spawn 1 million tasks = ~1sec)
  - Async, half-duplex channels (“buffered capabilities”)
  - **No shared mutable state**
  - Can only pass immutable messages
  - Idempotent task failure, failure-signal linkage

# Details! (#5)

- **Systems** language bestiary:
  - Fast calling of C (~8 insns, switch stacks)
  - Fast and safe stack-iterators (no cursor objects)
  - **No global GC** to fight (only per-task, mutable bits)
  - Real data structures (incl. nested structures)
    - Stack allocation, destructors, RAII
  - Multi-file compilation / optimization
    - ELF/MachO/PE + DWARF
    - works with GDB, valgrind, shark, etc.

# Details! (#6)

- Other useful bits (trying to be thorough)
  - Generics
  - Bignums
  - Nested modules with import/export control
  - UTF8 strings (not UCS2)
  - Marked syntax-extension system
  - Reflection, dynamic type, type-switch
- None of this stuff is surprising or unique!

# Implementation status

- Young, immature, hobby project until lately
  - Mostly-done design by now, heads down
  - ~80% language features working
  - ~70% runtime working
- 35kloc bootstrap compiler (ocaml)
  - Built-in x86 backend for linux, win32, OSX
  - LLVM backend in progress
- Minimal standard library
  - Hello there, interns!

# Mozilla Involvement

- Until last summer, I worked on this alone.
- Mozilla has strategic interest safer languages for the future. Memory / concurrency bug whack-a-mole gets stale. Investing in Rust.
- No concrete plans regarding specific use. Investing in project to see what develops.
- Small team presently working within labs.
- Volunteers (and non-mozillans) welcome.

# Huh? No code samples?

- Not in this talk, it'd be a distraction
  - It reads really predictably, C-family-ish
  - Syntax is very secondary to semantics
  - Also easy(-ish) to tweak as we go along
  - Had to leave you curious after the talk :)
- Besides, do you really want to spend your hours on planet earth arguing over syntax?
  - *Please keep this thought in mind when posting to the mailing list*

# Inevitable question: is this like “Go”?

- No
  - I've been working on Rust for years. There are dozens of actor languages in the pipeline. Go to a PL conference and ask around.
- Go seems to be barking up a different tree?
  - Has coroutines, but *kept shared mutable state*
  - Has memory safety, but *kept null pointers*
  - Has unwinding, but *no destructors or RAII*
  - Has message passing, but *no immutability*
  - Has some built-in generics, but *not in user code*

# Immediate plans

- Keep hacking on compiler and runtime
  - Eventually transition to self-hosted frontend, LLVM backend
- Build out libraries and bindings
- Need help: experienced language implementors best, plus an army of worker-drones
  - Please: no research or novelty! There's plenty of known-good technology in the literature.

# Fini

Q and A time!