ripple

Make waves

The Team

Amar Dhingra
Project Manager

Spencer Brown
Language Guru

Alex Roth
System Architect

Artur Renault
System Integrator

Tom Segarra
Tester & Verifier

The Mission

- Build a Reactive, Imperative Programming Language

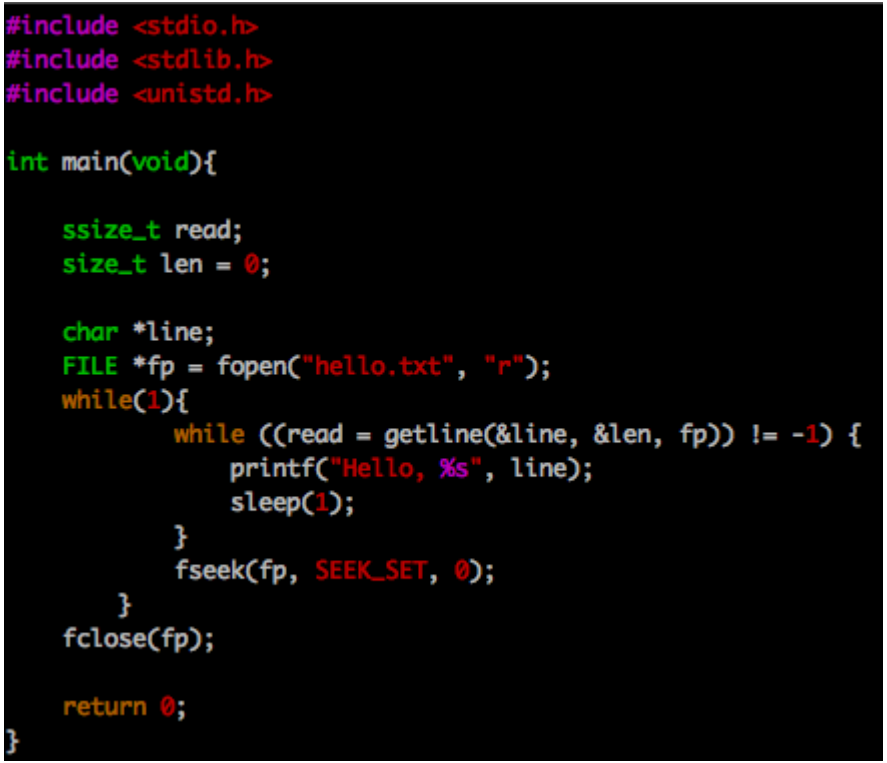
Why Ripple?

Data is no longer local to a program

Data comes in all sorts of shapes and forms

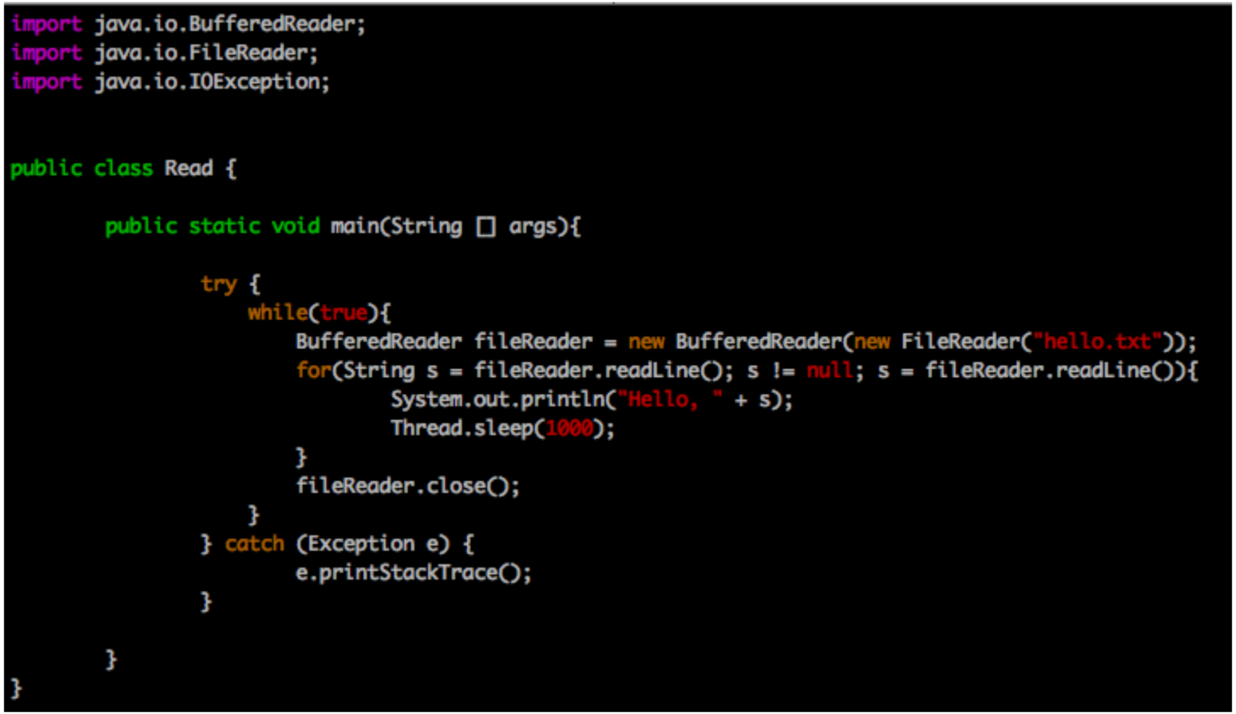
No one wants to write long, complex code

Reading in a file - C



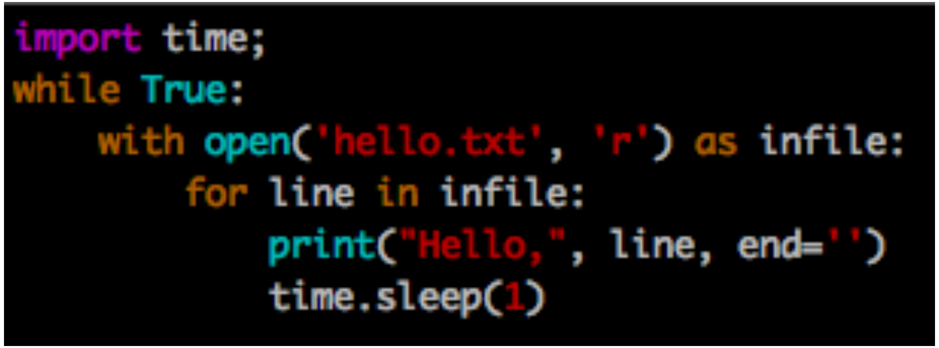
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void){

    ssize_t read;
    size_t len = 0;

    char *line;
    FILE *fp = fopen("hello.txt", "r");
    while(1){
            while ((read = getline(&line, &len, fp)) != -1) {
                printf("Hello, %s", line);
                sleep(1);
            }
            fseek(fp, SEEK_SET, 0);
        }
    fclose(fp);

    return 0;
}
```

Reading a file - Java



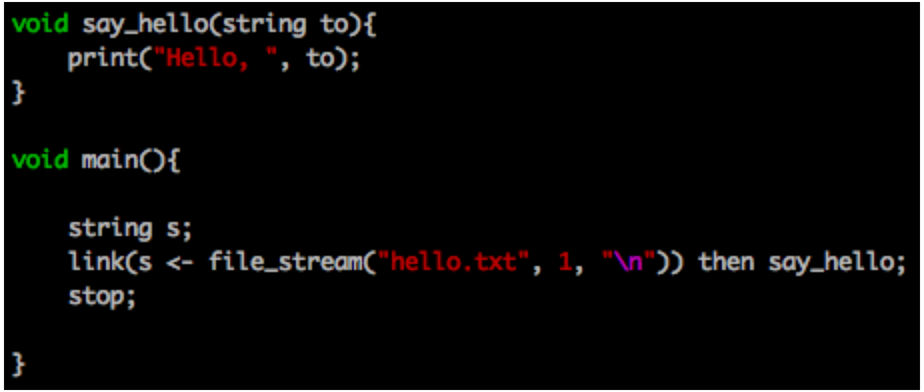
```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;


public class Read {

        public static void main(String [] args){

                try {
                    while(true){
                        BufferedReader fileReader = new BufferedReader(new FileReader("hello.txt"));
                        for(String s = fileReader.readLine(); s != null; s = fileReader.readLine()){
                                System.out.println("Hello, " + s);
                                Thread.sleep(1000);
                        }
                        fileReader.close();
                    }
                } catch (Exception e) {
                        e.printStackTrace();
                }

        }
}
```

Reading a file - Python



```python
import time;
while True:
    with open('hello.txt', 'r') as infile:
        for line in infile:
            print("Hello,", line, end='')
            time.sleep(1)
```

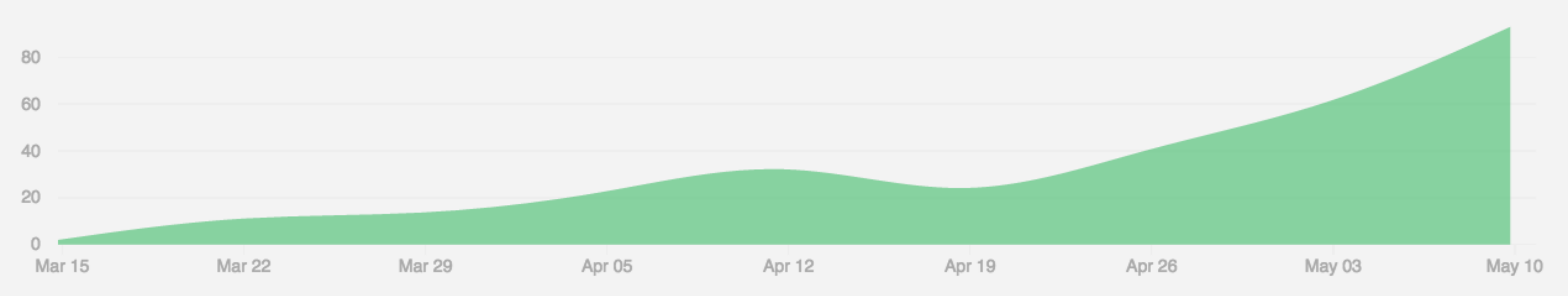
Reading a file in Ripple



```
void say_hello(string to){
    print("Hello, ", to);
}

void main(){

    string s;
    link(s <- file_stream("hello.txt", 1, "\n")) then say_hello;
    stop;

}
```

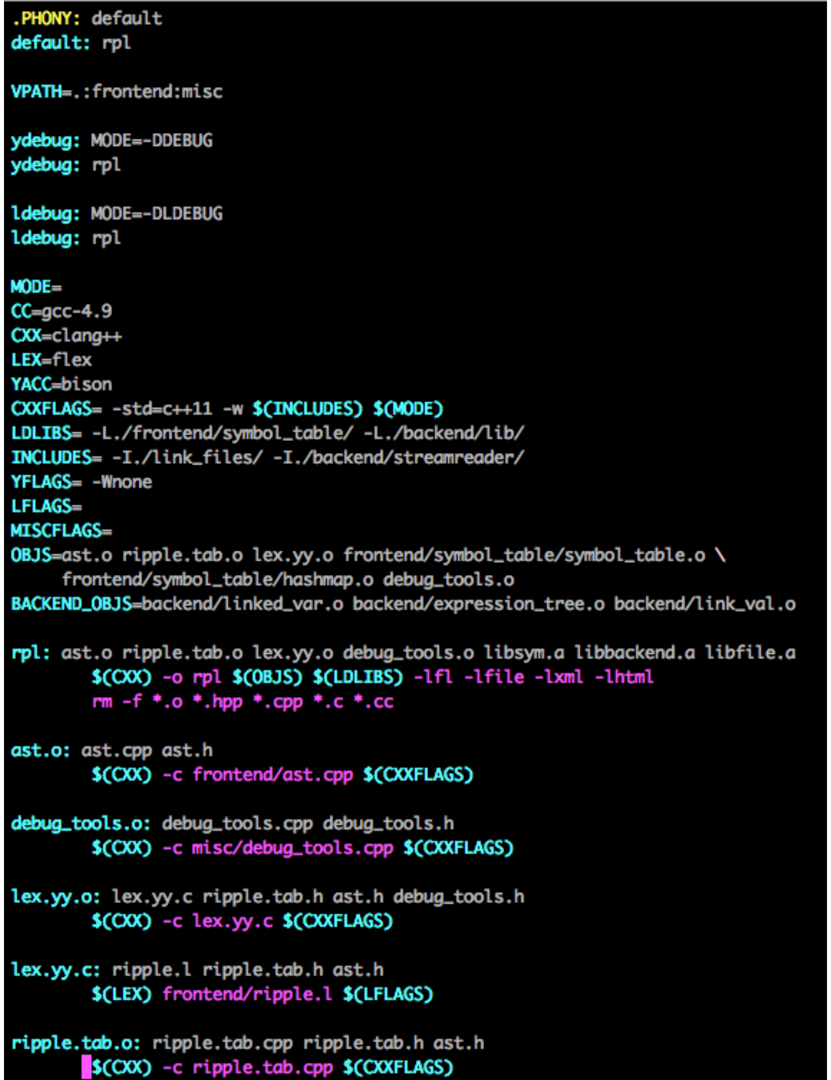
Project Management

Git History



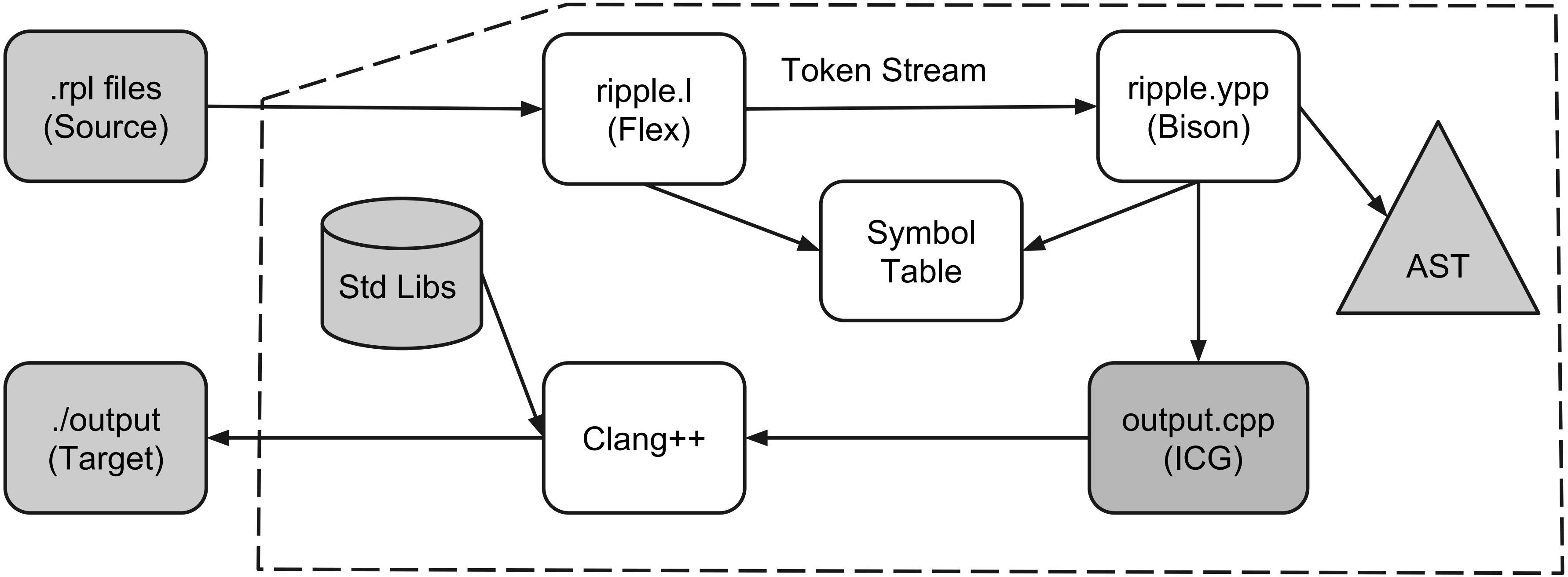
80
60
40
20
0
Mar 15
Mar 22
Mar 29
Apr 05
Apr 12
Apr 19
Apr 26
May 03
May 10

System Integration



```
.PHONY: default
default: rpl

VPATH=.:frontend:misc

ydebug: MODE=-DDEBUG
ydebug: rpl

ldebug: MODE=-DLDEBUG
ldebug: rpl

MODE=
CC=gcc-4.9
CXX=clang++
LEX=flex
YACC=bison
CXXFLAGS= -std=c++11 -w $(INCLUDES) $(MODE)
LDLIBS= -L./frontend/symbol_table/ -L./backend/lib/
INCLUDES= -I./link_files/ -I./backend/streamreader/
YFLAGS= -Wnone
LFLAGS=
MISCFLAGS=
OBJS=ast.o ripple.tab.o lex.yy.o frontend/symbol_table/symbol_table.o \
     frontend/symbol_table/hashmap.o debug_tools.o
BACKEND_OBJS=backend/linked_var.o backend/expression_tree.o backend/link_val.o

rpl: ast.o ripple.tab.o lex.yy.o debug_tools.o libsym.a libbackend.a libfile.a
        $(CXX) -o rpl $(OBJS) $(LDLIBS) -lfl -lfile -lxml -lhtml
        rm -f *.o *.hpp *.cpp *.c *.cc

ast.o: ast.cpp ast.h
        $(CXX) -c frontend/ast.cpp $(CXXFLAGS)

debug_tools.o: debug_tools.cpp debug_tools.h
        $(CXX) -c misc/debug_tools.cpp $(CXXFLAGS)

lex.yy.o: lex.yy.c ripple.tab.h ast.h debug_tools.h
        $(CXX) -c lex.yy.c $(CXXFLAGS)

lex.yy.c: ripple.l ripple.tab.h ast.h
        $(LEX) frontend/ripple.l $(LFLAGS)

ripple.tab.o: ripple.tab.cpp ripple.tab.h ast.h
        $(CXX) -c ripple.tab.cpp $(CXXFLAGS)
```

The Architecture



./rpl

Language Guru

Key Syntactic Elements:

1. Link Statements
2. Streams

The Link and The Stream

```
link(var <- stream()) function;
```

```
link(var <- filter_function <- stream()) then auxiliary_function;
```

The Link Statement

```
link(                              )                    ;
```

The Link Statement

```
link(                    stream() )                ;
```

The Link Statement

```
link(        filter <- stream() )                 ;
```

The Link Statement

```
link( var <- filter <- stream() )                  ;
```

The Link Statement

```
link( var <- filter <- stream() ) then auxiliary;
```

Testing

1. Write tests

2. Write good tests

3. Write good tests early

Testing

- should not be thought of as a trade-off against the implementation of features.
- cannot be avoided.
- can be both modular and integrated.
- often produces unexpected benefits.

What We Learned

- Amar: Communication
- Tom: Scope
- Spencer: Action
- Alex: Dependability
- Artur: Expectations

Thank you!