The Fly Language

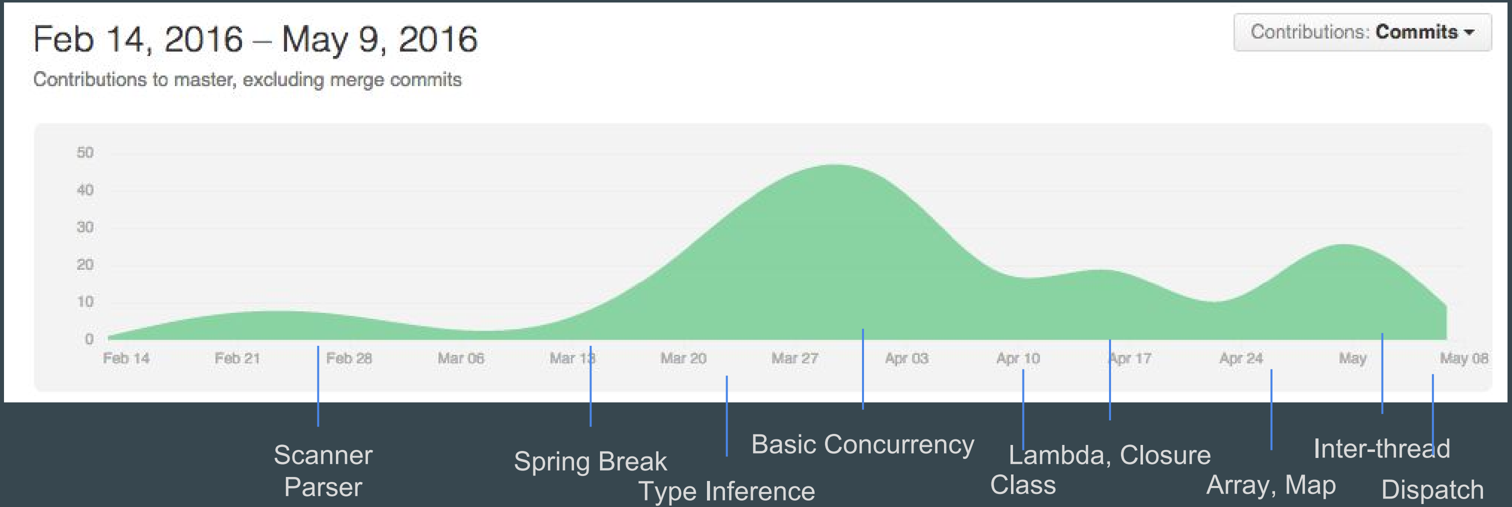
Carolyn Sun   Hsiang-Ho Lin   Shenlong Gu   Xin Xu

Introduction

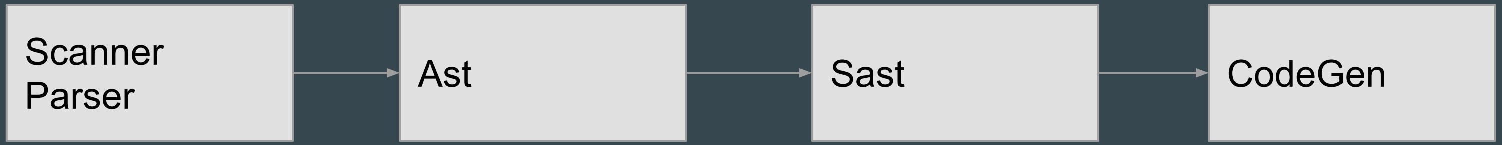
- Motivation
- Compile down to C++ code
- Type inference
- Concurrency primitives: thread, channel, signal
- Thread-safe container types
- Capability for code to be dispatched and executed across systems
- Functional programming features such as lambda and clojure
- Network Library

Project Status

- 3217 lines of OCaml code
- 497 lines of C++ code
- 276 git commits
- 48 test cases, 1051 lines of test code



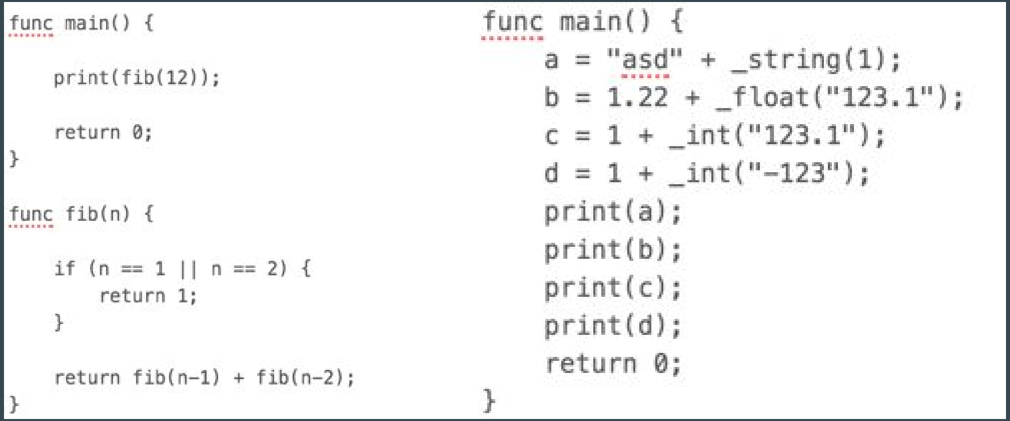
Architecture



Type Inference

Variables are static typed. Functions are typed according to all kinds of calls that invoked on the functions.

Tech: we infer a function result when a function is called with typed parameters.



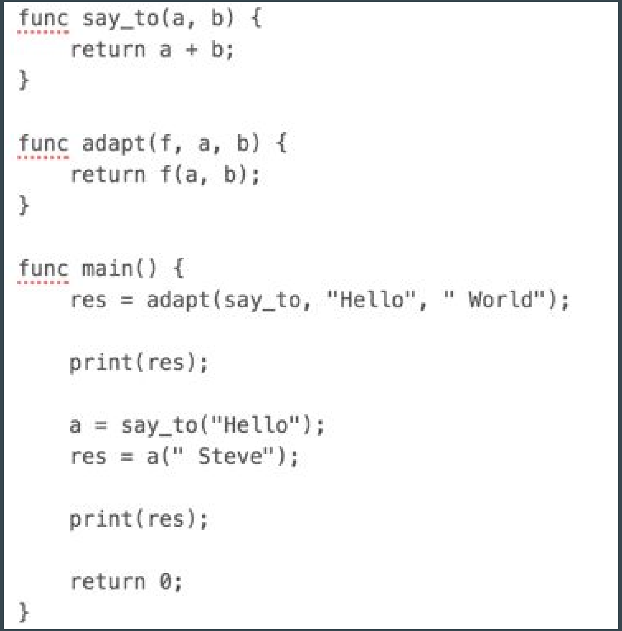
```
func main() {

    print(fib(12));

    return 0;
}

func fib(n) {

    if (n == 1 || n == 2) {
        return 1;
    }

    return fib(n-1) + fib(n-2);
}
```

```
func main() {
    a = "asd" + _string(1);
    b = 1.22 + _float("123.1");
    c = 1 + _int("123.1");
    d = 1 + _int("-123");
    print(a);
    print(b);
    print(c);
    print(d);
    return 0;
}
```

Closure

Each function can be called with some parameters to generate a closure (a function binded with some parameters)

Tech: Use a class to hold the variables and functions.



```
func say_to(a, b) {
    return a + b;
}

func adapt(f, a, b) {
    return f(a, b);
}

func main() {
    res = adapt(say_to, "Hello", " World");

    print(res);

    a = say_to("Hello");
    res = a(" Steve");

    print(res);

    return 0;
}
```

Lambda

We support some basic lambda usage.

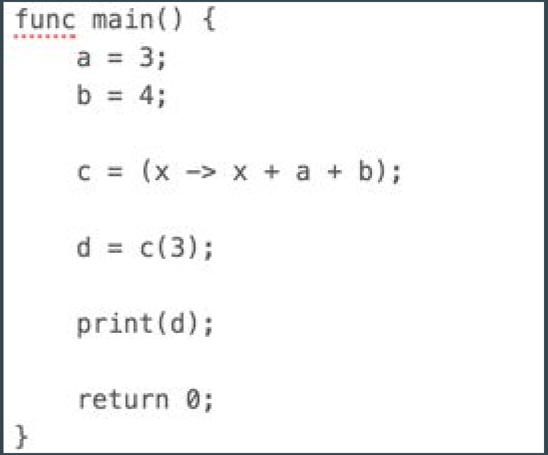
Variables are passed by referrence for the class, map, array.

Variables are passed by value for int, float, string.

Tech: we keep track of all variables used in the

lambda and generate a new function for C++ with

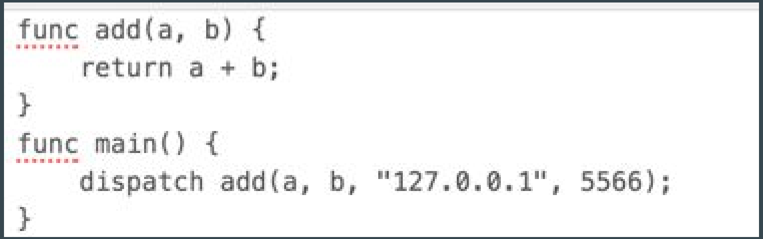
these local variables wrapped like clojure.



```
func main() {
    a = 3;
    b = 4;

    c = (x -> x + a + b);

    d = c(3);

    print(d);

    return 0;
}
```

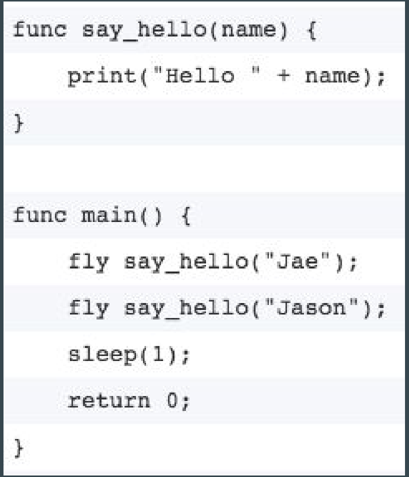
Dispatch/Exec

We can send a function with some parameters to another machine to execute and wait for the result to be returned.

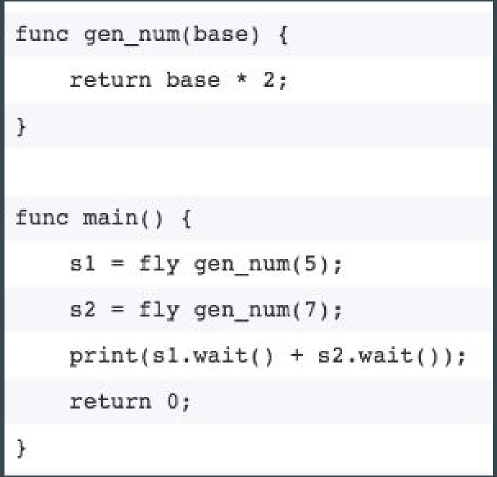


```
func add(a, b) {
    return a + b;
}
func main() {
    dispatch add(a, b, "127.0.0.1", 5566);
}
```

Concurrency: threading



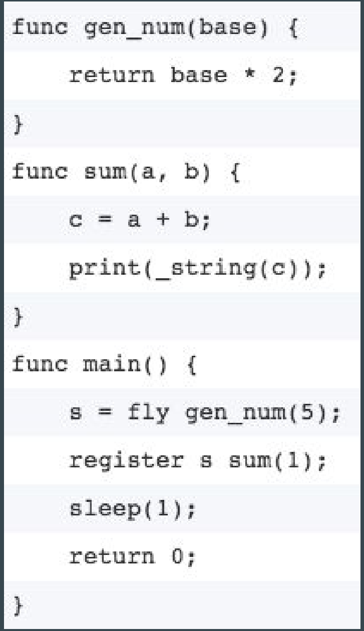
```
func say_hello(name) {
    print("Hello " + name);
}

func main() {
    fly say_hello("Jae");
    fly say_hello("Jason");
    sleep(1);
    return 0;
}
```



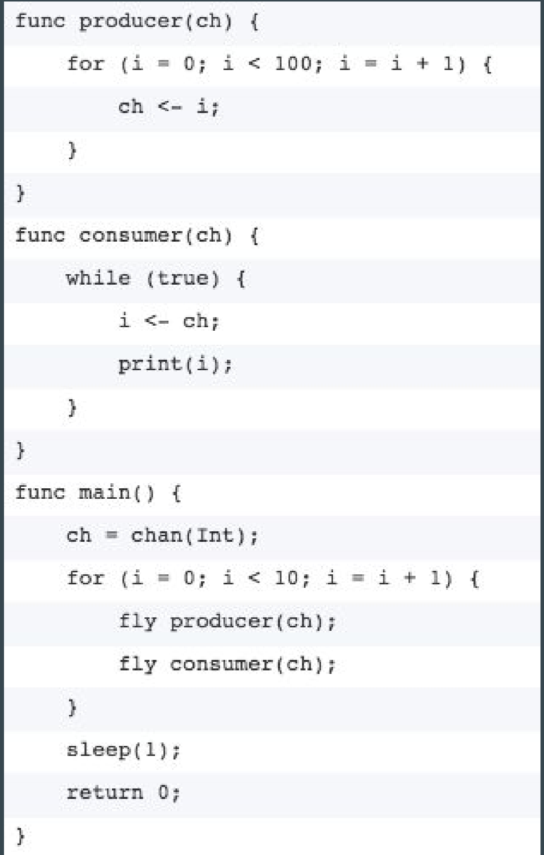
```
func gen_num(base) {
    return base * 2;
}

func main() {
    s1 = fly gen_num(5);
    s2 = fly gen_num(7);
    print(s1.wait() + s2.wait());
    return 0;
}
```

Concurrency: Inter-thread communication

Signal



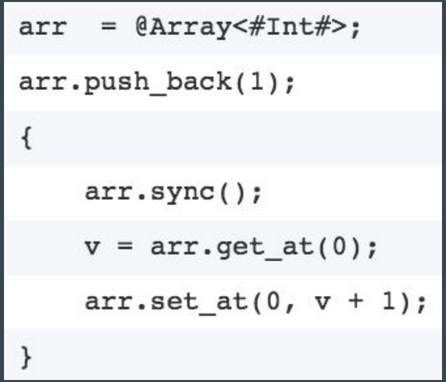
```
func gen_num(base) {
    return base * 2;
}
func sum(a, b) {
    c = a + b;
    print(_string(c));
}
func main() {
    s = fly gen_num(5);
    register s sum(1);
    sleep(1);
    return 0;
}
```



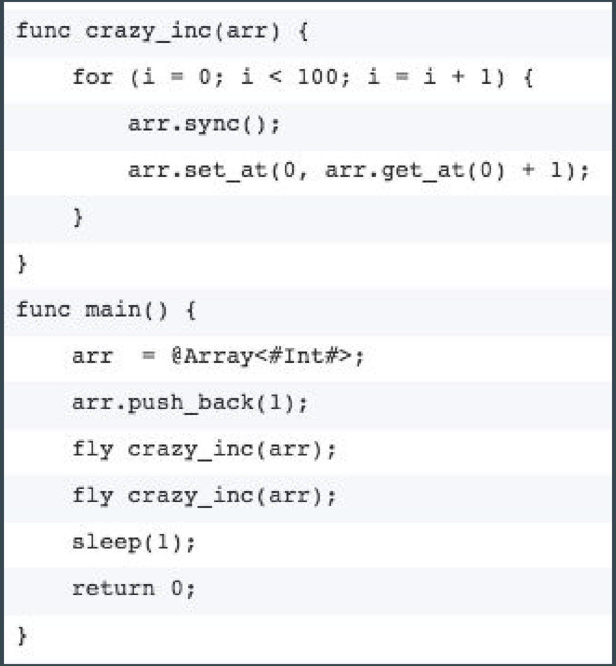
```
func producer(ch) {
    for (i = 0; i < 100; i = i + 1) {
        ch <- i;
    }
}
func consumer(ch) {
    while (true) {
        i <- ch;
        print(i);
    }
}
func main() {
    ch = chan(Int);
    for (i = 0; i < 10; i = i + 1) {
        fly producer(ch);
        fly consumer(ch);
    }
    sleep(1);
    return 0;
}
```

Channel

Concurrency: Thread-Safe Containers



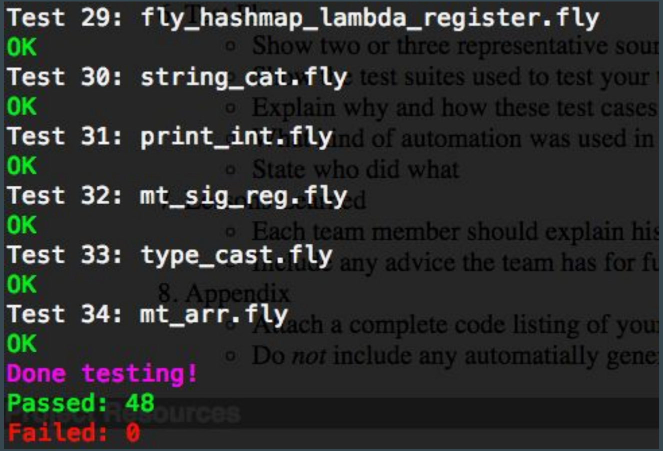
```
arr  = @Array<#Int#>;
arr.push_back(1);
{
    arr.sync();
    v = arr.get_at(0);
    arr.set_at(0, v + 1);
}
```



```
func crazy_inc(arr) {
    for (i = 0; i < 100; i = i + 1) {
        arr.sync();
        arr.set_at(0, arr.get_at(0) + 1);
    }
}
func main() {
    arr  = @Array<#Int#>;
    arr.push_back(1);
    fly crazy_inc(arr);
    fly crazy_inc(arr);
    sleep(1);
    return 0;
}
```

Automated Integration Tests

- 48 Test cases, 14 for should-fail, 34 for should-pass
- Use python script to automate the process
- Verifies all the test cases are passed before committing



```
Test 29: fly_hashmap_lambda_register.fly
OK
Test 30: string_cat.fly
OK
Test 31: print_int.fly
OK
Test 32: mt_sig_reg.fly
OK
Test 33: type_cast.fly
OK
Test 34: mt_arr.fly
OK
Done testing!
Passed: 48
Failed: 0
```

Team Responsibilities

Carolyn Sun: Testing automation, Debug module, Documentation

Hsiang-Ho Lin: Compiler Front end, Code generation, C++ Library, Test case creation, Documentation

Shenlong Gu: Compiler Front end, Semantics, Code generation, C++ Library, Documentation

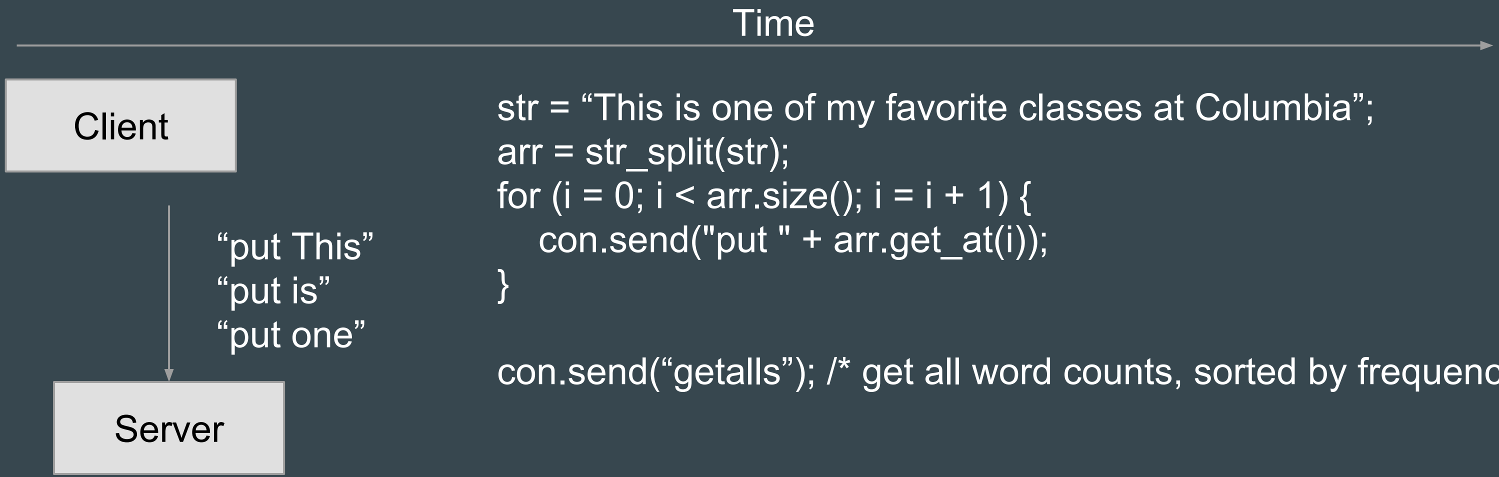
Xin Xu: Test case creation, Debug module, Documentation

Lesson Learned

- Time Management
  - Start Early
  - Meet Regularly
- Communication
  - Listen and Share Ideas
- Collaboration
  - Github
  - Clean Code
  - Don’t commit broken code
- Testing
  - Automate

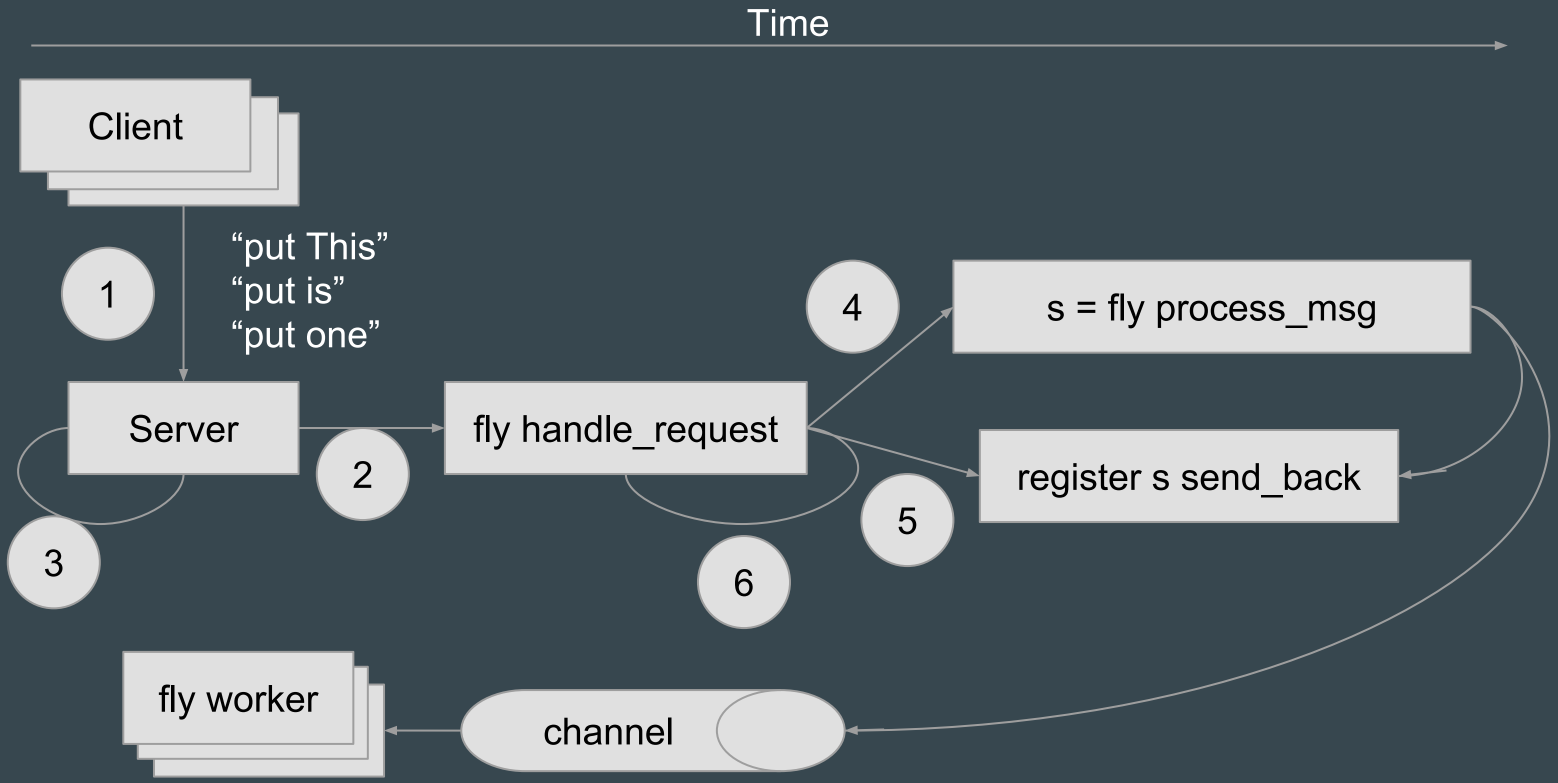
Demo

Word Count Server and Client



```
str = “This is one of my favorite classes at Columbia”;
arr = str_split(str);
for (i = 0; i < arr.size(); i = i + 1) {
    con.send("put " + arr.get_at(i));
}

con.send(“getalls”); /* get all word counts, sorted by frequency */
```

Word Count Server and Client


Time
Client
"put This"
"put is"
"put one"
1
Server
2
3
fly handle_request
4
s = fly process_msg
5
register s send_back
6
fly worker
channel