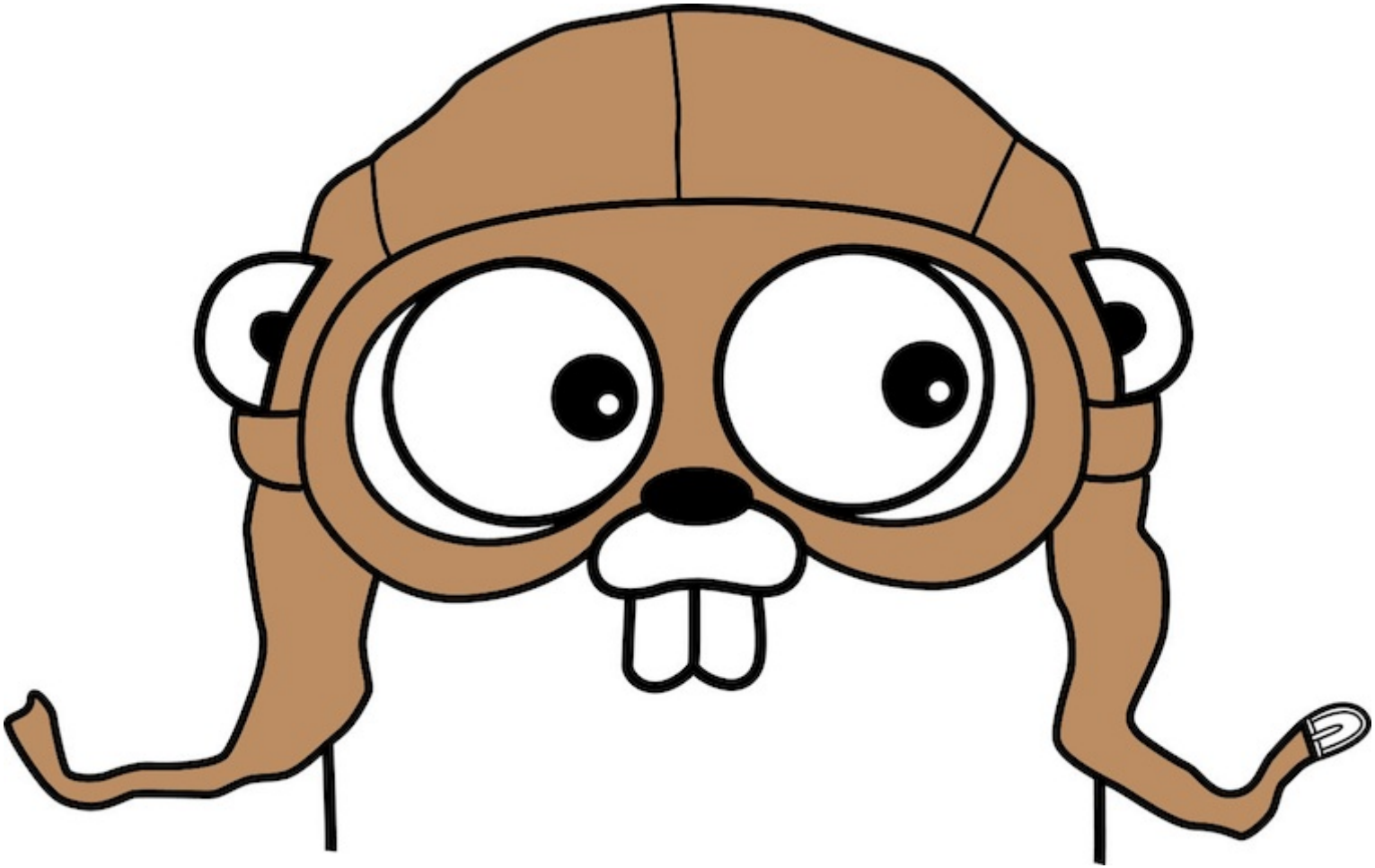
Composing Channels for Easy Concurrency

September 10, 2015, Dallas, Texas, USA

John Scott
Consultant, American Messaging, 2006-now
Founder, SetSpace, Inc, 1998-now

The single biggest problem in communication is the illusion that it has taken place.

George Bernard Shaw



Share Memory by Communicating

Go is a Blend of Sequential and Concurrent Coding

Sequential Fibonacci Series

f(n) = f(n-1) + f(n-2)

```
package main
import . "fmt"

func fib(i int) int {

    if i <= 2 {
        return 1
    }

    return fib(i - 1) + fib(i - 2)
}

func main() {

    Println(fib(42))

}
```

Run

Multiple, Sequential Fibonacci Series

f(n) = f(n-1) + f(n-2)

```
package main
import . "fmt"

func fib(i int) int {

    if i <= 2 {
        return 1
    }

    return fib(i - 1) + fib(i - 2)
}

func main() {

    Println(fib(42), fib(42), fib(42), fib(42))
}
```

Run

Concurrent Fibonacci Series

f(n) = f(n-1) + f(n-2)

```
func main() {

    answer := make(chan int)

    fib_worker := func(i int) {
        answer <- fib(i)
    }

    go fib_worker(42)               // Just add "go" in front of function call
    go fib_worker(42)
    go fib_worker(42)
    go fib_worker(42)

    Println(<- answer, <- answer, <- answer, <- answer)
}
```

Run

The Big Threes of Systems Programming

- Modularity
- Composition
- Concurrency

Modular

No Side Effects, Code is Local

Fibonacci

```
func fib(i int) int {

    if i <= 2 {
        return 1
    }

    return fib(i - 1) + fib(i - 2)
}
```

Square

```
func square(i int) int {

    return i * i
}
```

Composition

Create New Modules from Existing

Squib

```
func squib(i int) int {

    return square(fib(i))
}

func main() {

    Println(squib(42))
}
```

Run

Concurrency

Concurrency is the composition of independently executing processes.

Parallelism is the simultaneous execution of any processes.

Unix Shell Pipeline is Concurrency

```
find . -name '*.txt'  |  bzip2 >files.bz2
```

Find All text Files and Compress into files.bz2

```
ps -el  |  grep postgres  |  grep -v grep  |  mail -s 'Postgres Processes' jmscott@setspace.com
```

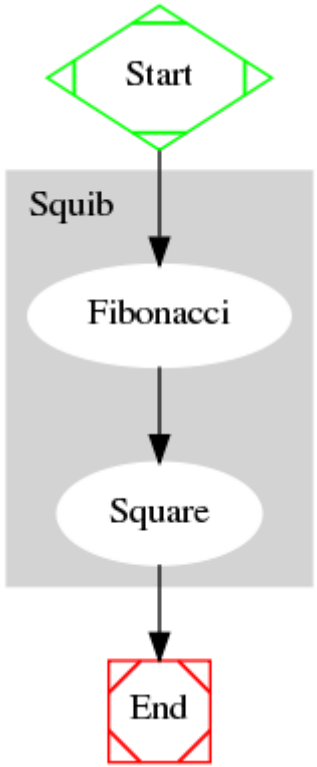
Mail List of All PostgreSQL Processes to jmscott@setspace.com

Background Processes Running in Parallel

```
slow_job1 &
slow_job2 &
slow_job3 &
```

Not Concurrent Since Not Composed into Single Process

Fibonacci + Squaring as a Pipeline



New Service Christianed "Squib"

Fibonacci as a Pipeline

```
func fib_pipe(in chan int) (out chan int) {

    out = make(chan int)

    go func() {
        defer close(out)

        for i := range in {
            out <- fib(i)
        }
    }()
    return out
}
```

First Node in Previous Graph

Squaring as a Pipeline

```
func square_pipe(in chan int) (out chan int) {

    out = make(chan int)

    go func() {
        defer close(out)

        for i := range in {
            out <- square(i)
        }
    }()
    return out
}
```

Second Node in Previous Graph

Composing Squib into a Pipeline

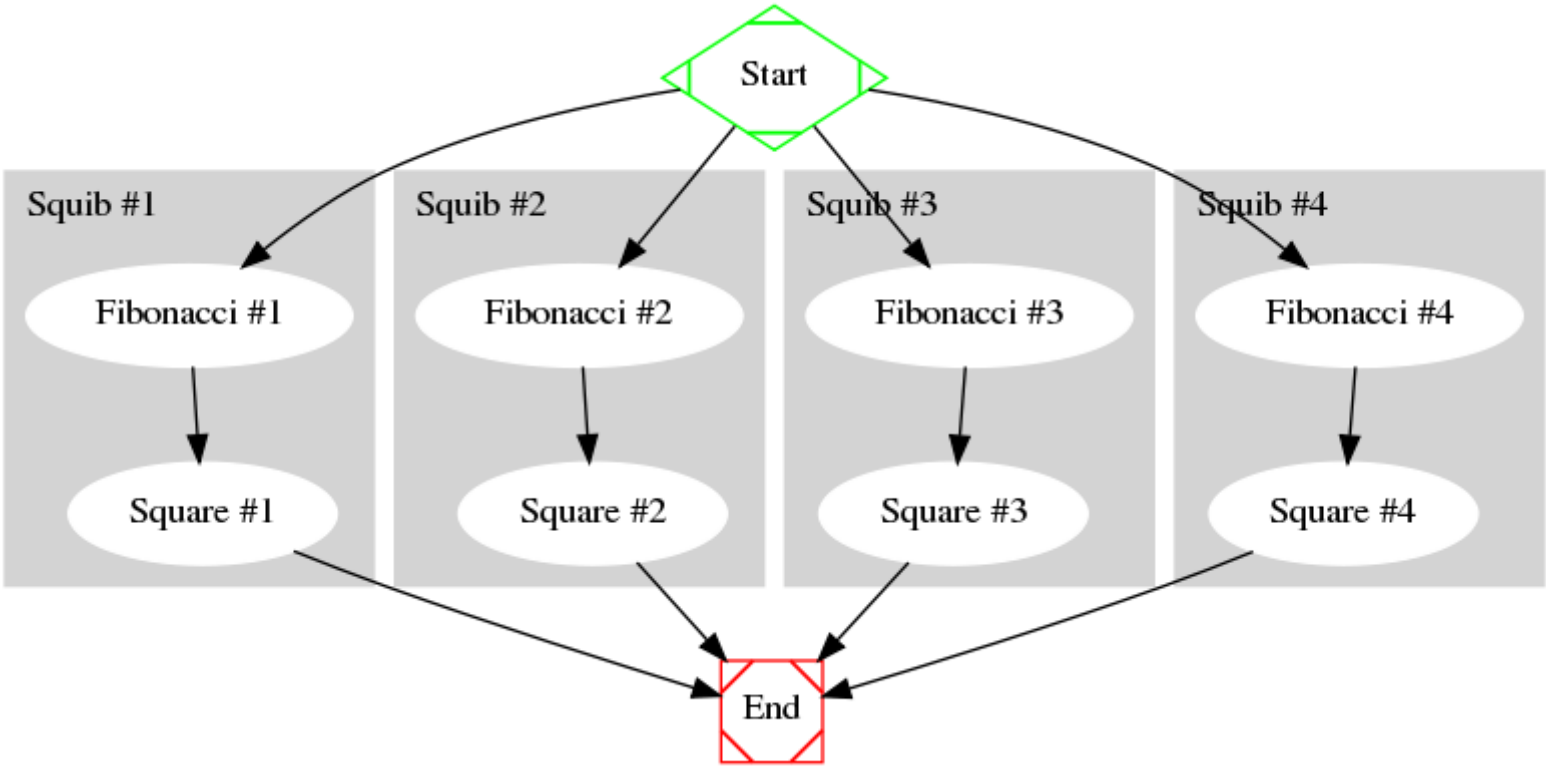
```
func main() {

    in := make(chan int)

    out := square_pipe(fib_pipe(in))

    for i := 1;  i <= 42;  i++ {
        in <- i
        Println(i, <-out)
    }
}
```

Run

Working Stealing

- Writes and Reads on Channels are Atomic
- Simulaneous Readers Compete on Channel for Next Available Datum

Squib as a Queue of Workers



Each Squib Runs in Parallel

Squib as a Queue of Workers

```
func squib_queue(in chan int, worker_count int) (merge chan int) {

    merge = make(chan int)

    for i := 0;  i < worker_count;  i++ {

        go func() {

            out := square_pipe(fib_pipe(in))
            for {
                merge <- (<- out)
            }

        }()
    }
    return merge
}
```

Run Squib as Work Queue

```
func main() {

    in := make(chan int)

    pump := func(limit int) {
        for i := 1;  i <= limit;  i++ {
            in <- i
        }
    }
    go pump(42)

    out := squib_queue(in, 4)

    for i := 1;  i <= 42;  i++ {
        Println(<- out)
    }
}
```

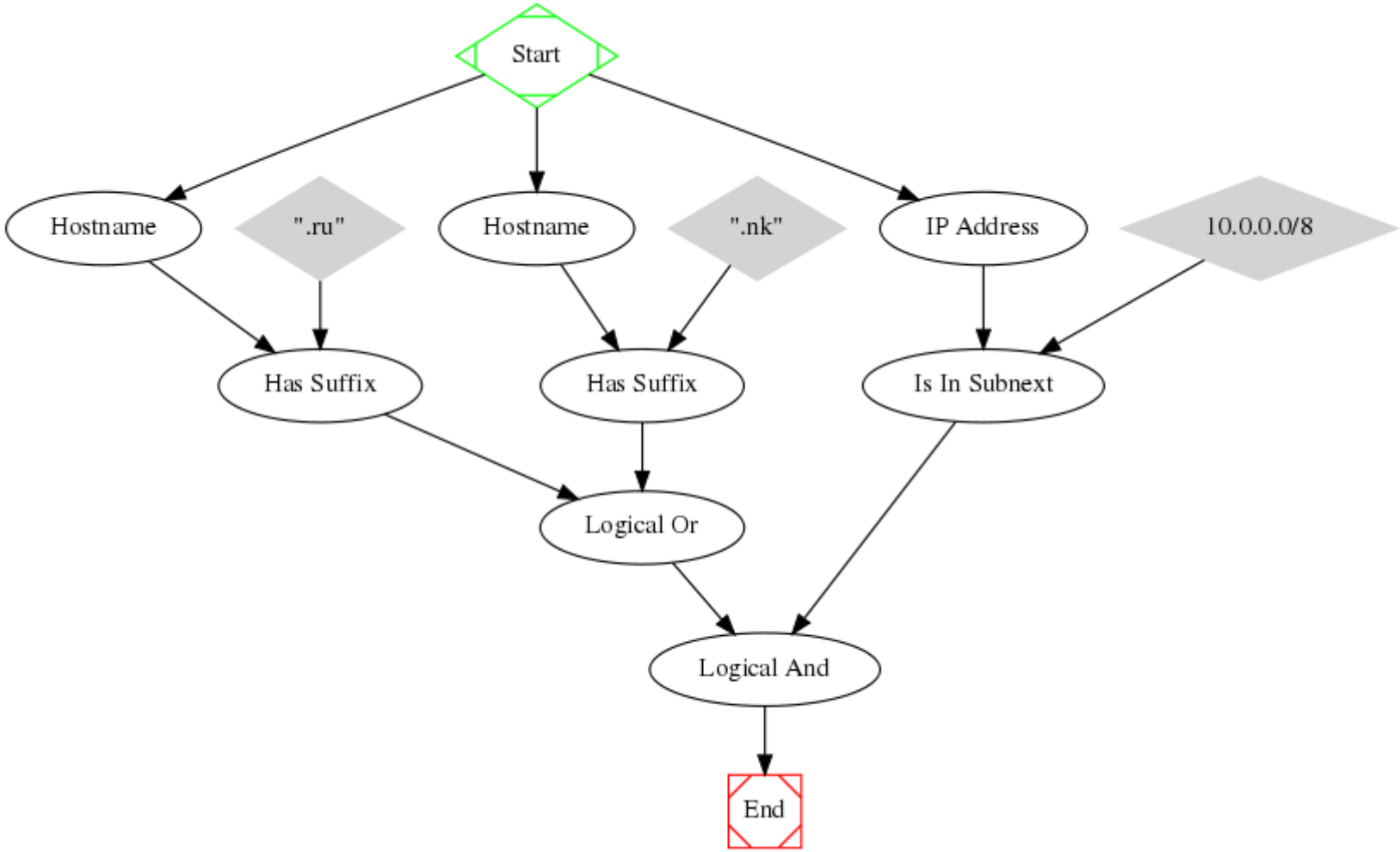
Run

Cheap Broadcast by Closing Channel

Read on Closed Channel Returns Nil Or Zero

```
func pipe_element(in chan int) (out chan int) {

    out = make(chan int)

    go func() {
        defer close(out)

        for {
            request := <-in
            if request == 0 {
                return
            }

            //  ... do some work

            out <- 42
        }
     }()
     return out
}
```

Complex Flow Graph Coordinate Work



Query: (hostname ~ 'ru$' or hostname ~ 'kp$') and ip4 <<= '^10.0.0.0/8'

Links

Interactive Tour of Go (http://tour.golang.org/)

An Introduction to Programming in Go - Online Book (http://www.golang-book.com/)

Meet the Go Team - Q/A at Google I/O 2012 (https://www.youtube.com/watch?v=sln-gJaURzk&list=PLoJWLKOp927tFsMbO2onhp26NnOAKAtO#t=921)

Concurrency is Not Parallelism by Rob Pike @ Vimeo (http://vimeo.com/49718712)

Communication Sequential Processes - The Theory That Inspired Go @ Wikipedia (http://en.wikipedia.org/wiki/Communicating_sequential_processes)

Communicating Sequential Processes - Readable Intro to CSP Algebra (PDF) (http://www.usingcsp.com/cspbook.pdf)

Searchable Documentation on Popular Go Packages (http://www.godoc.org)

Thank you

September 10, 2015, Dallas, Texas, USA

John Scott
Consultant, American Messaging, 2006-now
Founder, SetSpace, Inc, 1998-now

jmscott@setspace.com (mailto:jmscott@setspace.com)
john.scott@americanmessaging.net (mailto:john.scott@americanmessaging.net)