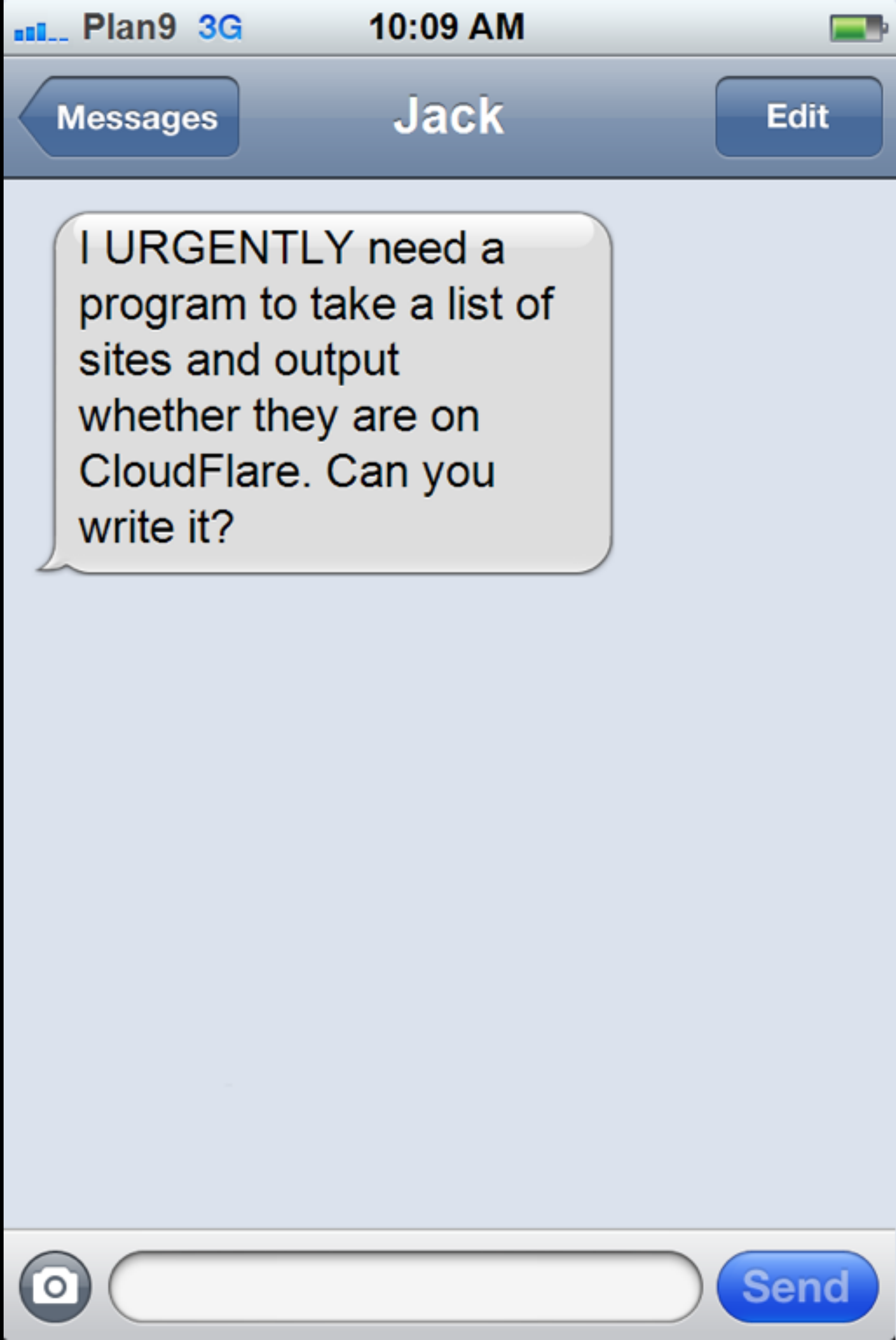
# I came for the easy concurrency

# I stayed for the easy composition

John Graham-Cumming

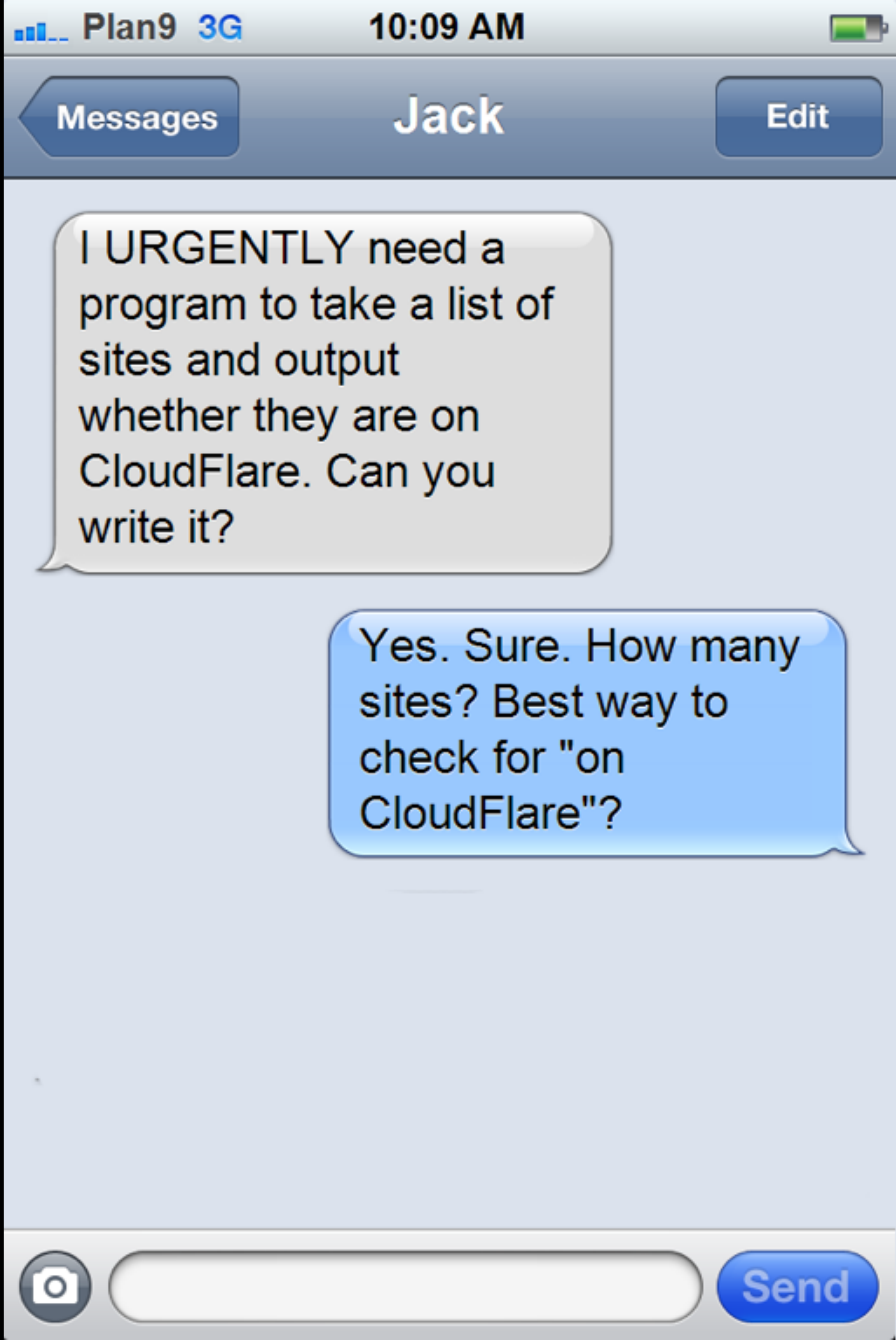
Happily working quietly when…

Happily working quietly when…



I URGENTLY need a program to take a list of sites and output whether they are on CloudFlare. Can you write it?

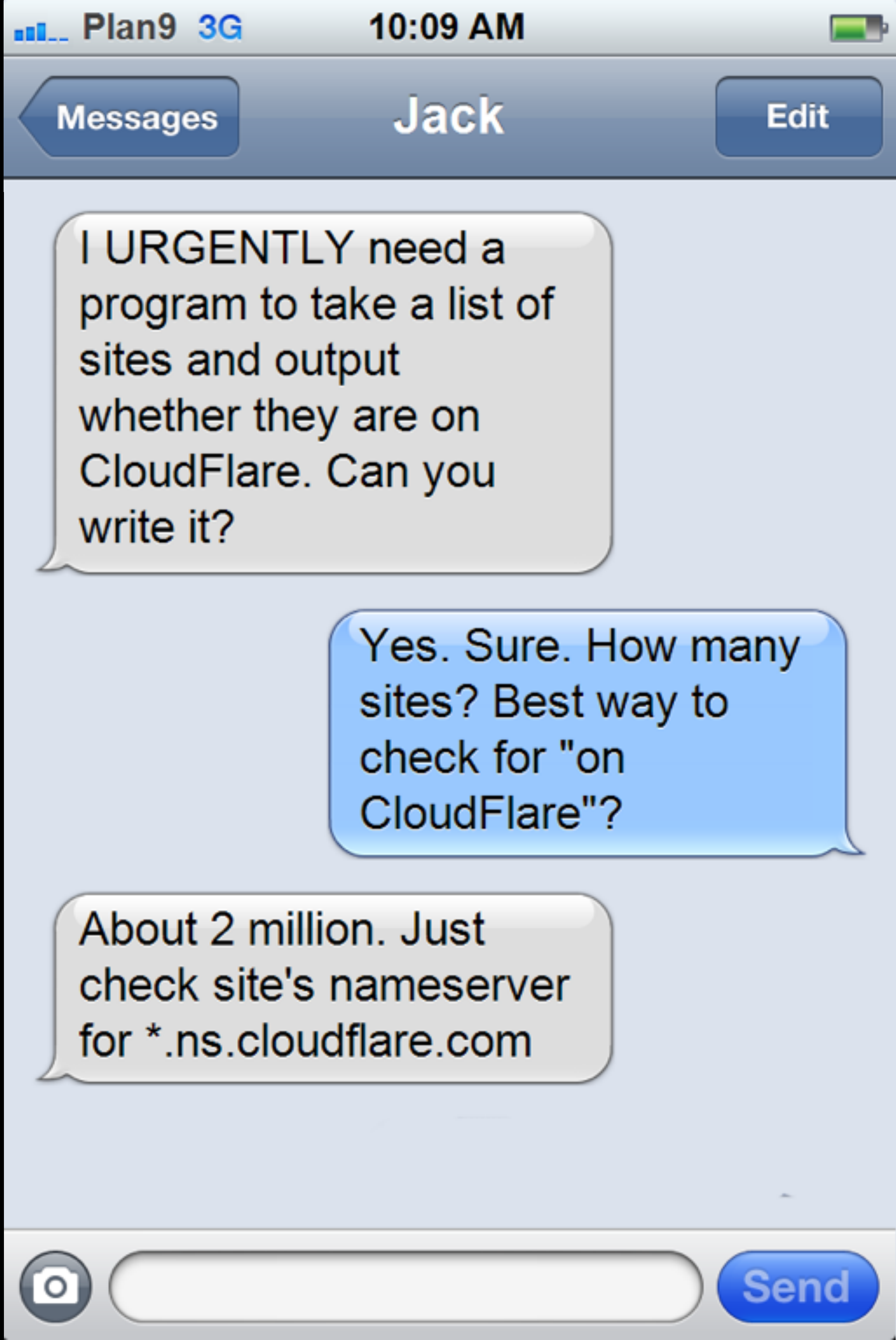
Happily working quietly when…



I URGENTLY need a program to take a list of sites and output whether they are on CloudFlare. Can you write it?

Yes. Sure. How many sites? Best way to check for "on CloudFlare"?

Happily working quietly when…

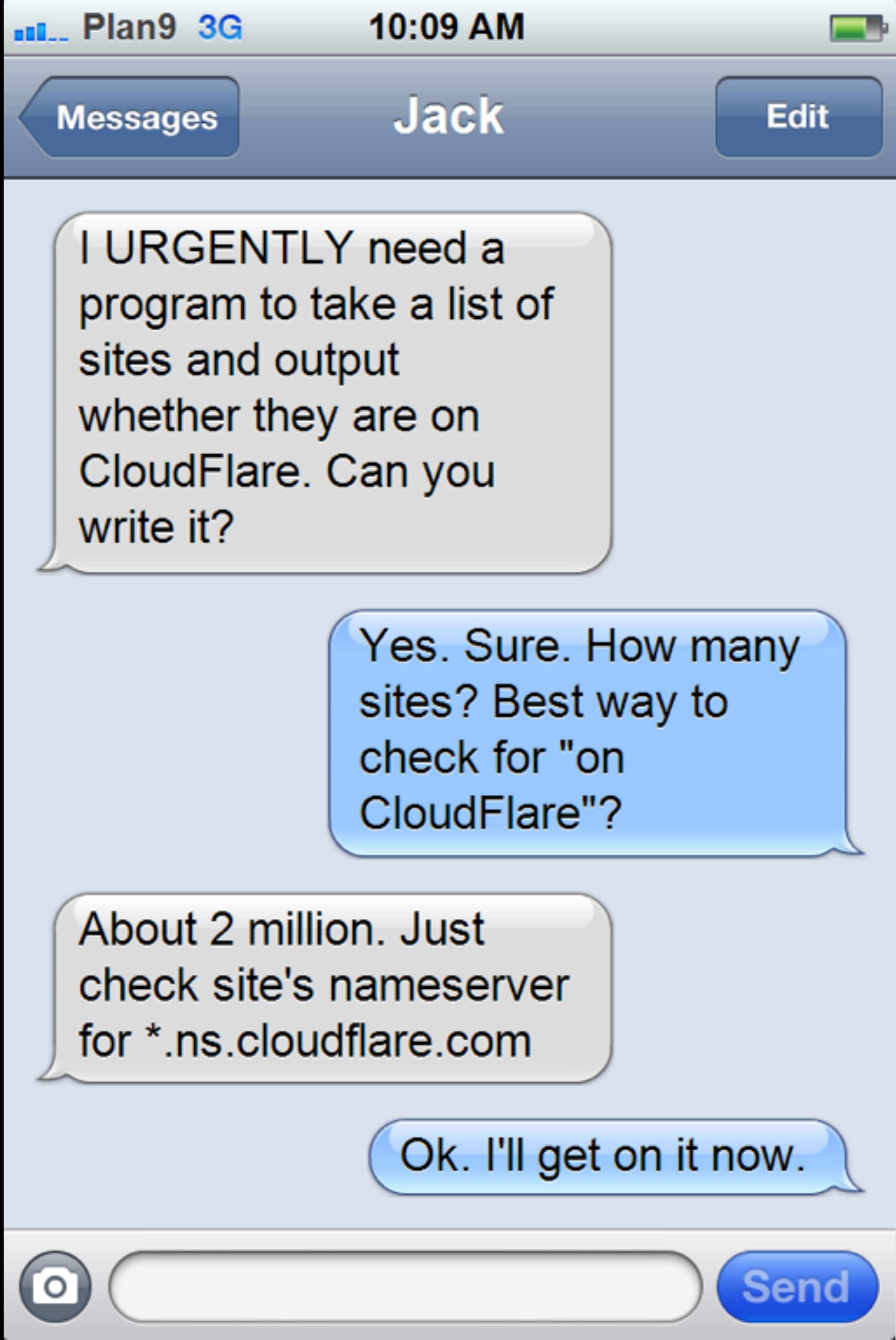


I URGENTLY need a program to take a list of sites and output whether they are on CloudFlare. Can you write it?

Yes. Sure. How many sites? Best way to check for "on CloudFlare"?

About 2 million. Just check site's nameserver for *.ns.cloudflare.com

Happily working quietly when…



I URGENTLY need a program to take a list of sites and output whether they are on CloudFlare. Can you write it?

Yes. Sure. How many sites? Best way to check for "on CloudFlare"?

About 2 million. Just check site's nameserver for *.ns.cloudflare.com

Ok. I'll get on it now.

# First thought

```
cat zones.txt | xargs -I{} dig NS {}
```

# First thought

```
cat zones.txt | xargs -I{} dig NS
{} | grep "IN\s*NS.*\.ns
\.cloudflare\.com"
```

# First thought

```
cat zones.txt | xargs -I{} dig NS
{} | grep "IN\s*NS.*\.ns
\.cloudflare\.com" | cut  -f1
```

# First thought

```
cat zones.txt | xargs -I{} dig NS
{} | grep "IN\s*NS.*\.ns
\.cloudflare\.com" | cut  -f1 |
sort
```

# First thought

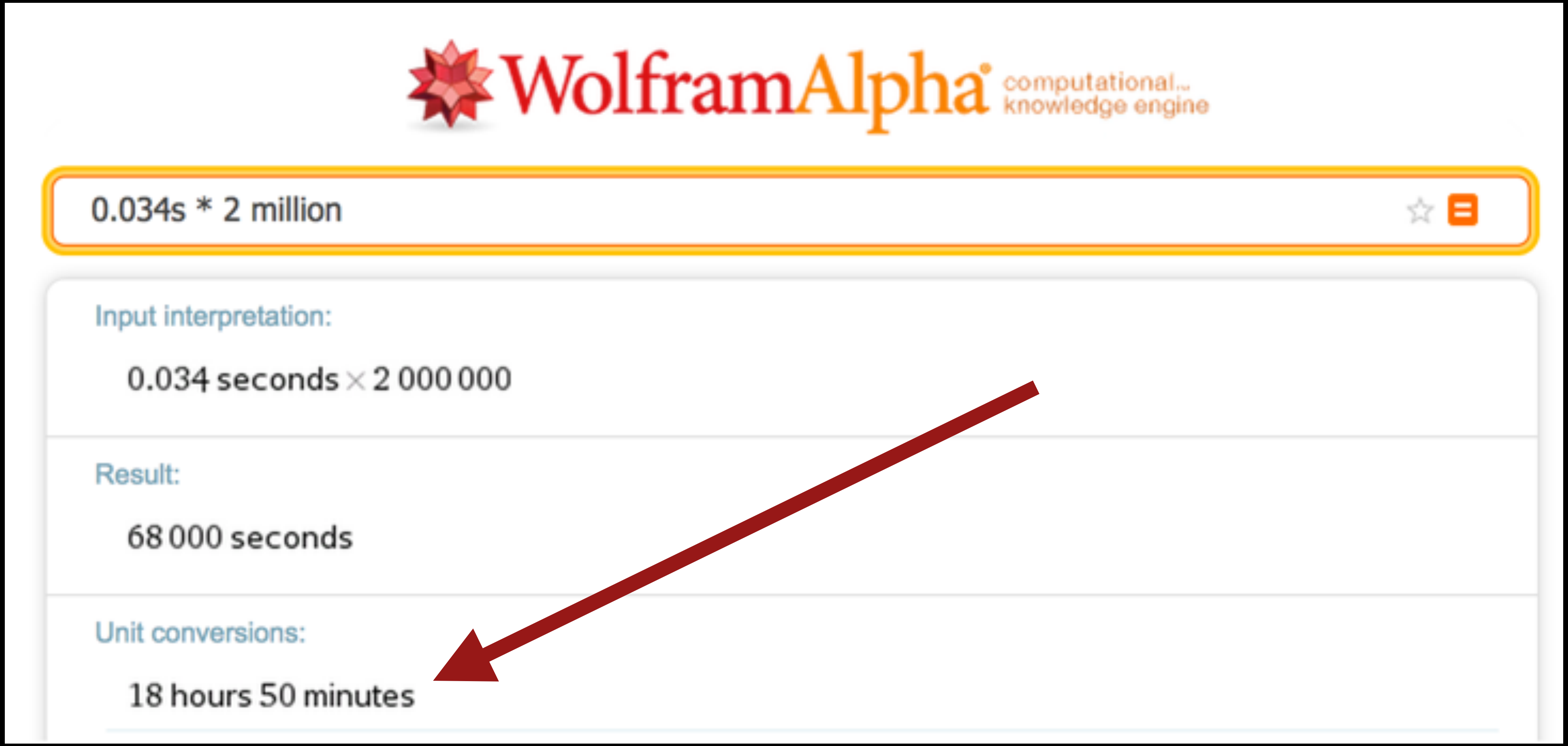
```
cat zones.txt | xargs -I{} dig NS
{} | grep "IN\s*NS.*\.ns
\.cloudflare\.com" | cut  -f1 |
sort | uniq
```

# But then...

```
% time dig ns jgc.org +short
sid.ns.cloudflare.com.
elsa.ns.cloudflare.com.
0.01s user 0.00s system 25% cpu 0.034 total
```

# But then...

```
% time dig ns jgc.org +short
sid.ns.cloudflare.com.
elsa.ns.cloudflare.com.
0.01s user 0.00s system 25% cpu 0.034 total
```



0.034s * 2 million

Input interpretation:

0.034 seconds × 2 000 000

Result:

68 000 seconds

Unit conversions:

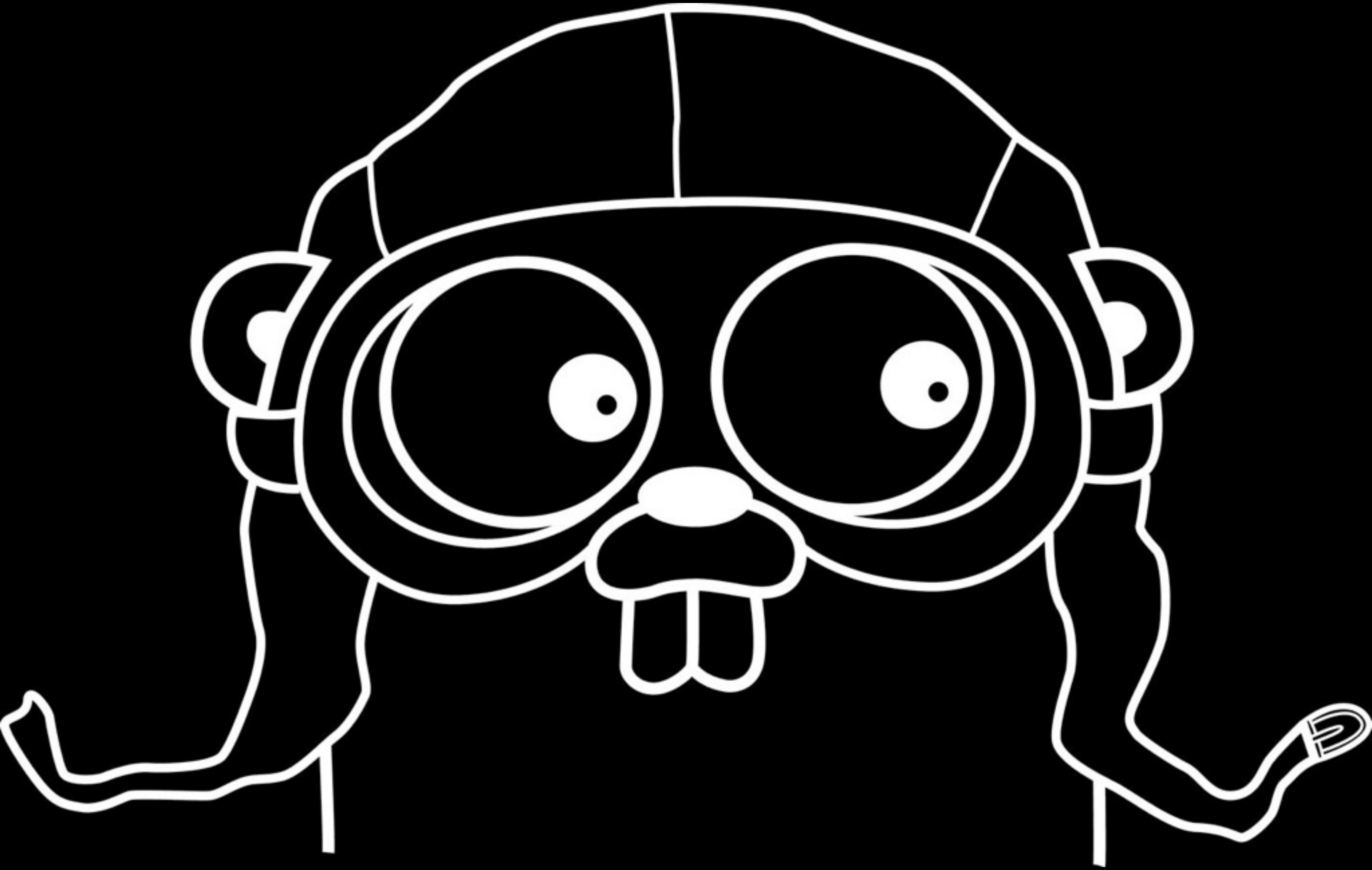
18 hours 50 minutes

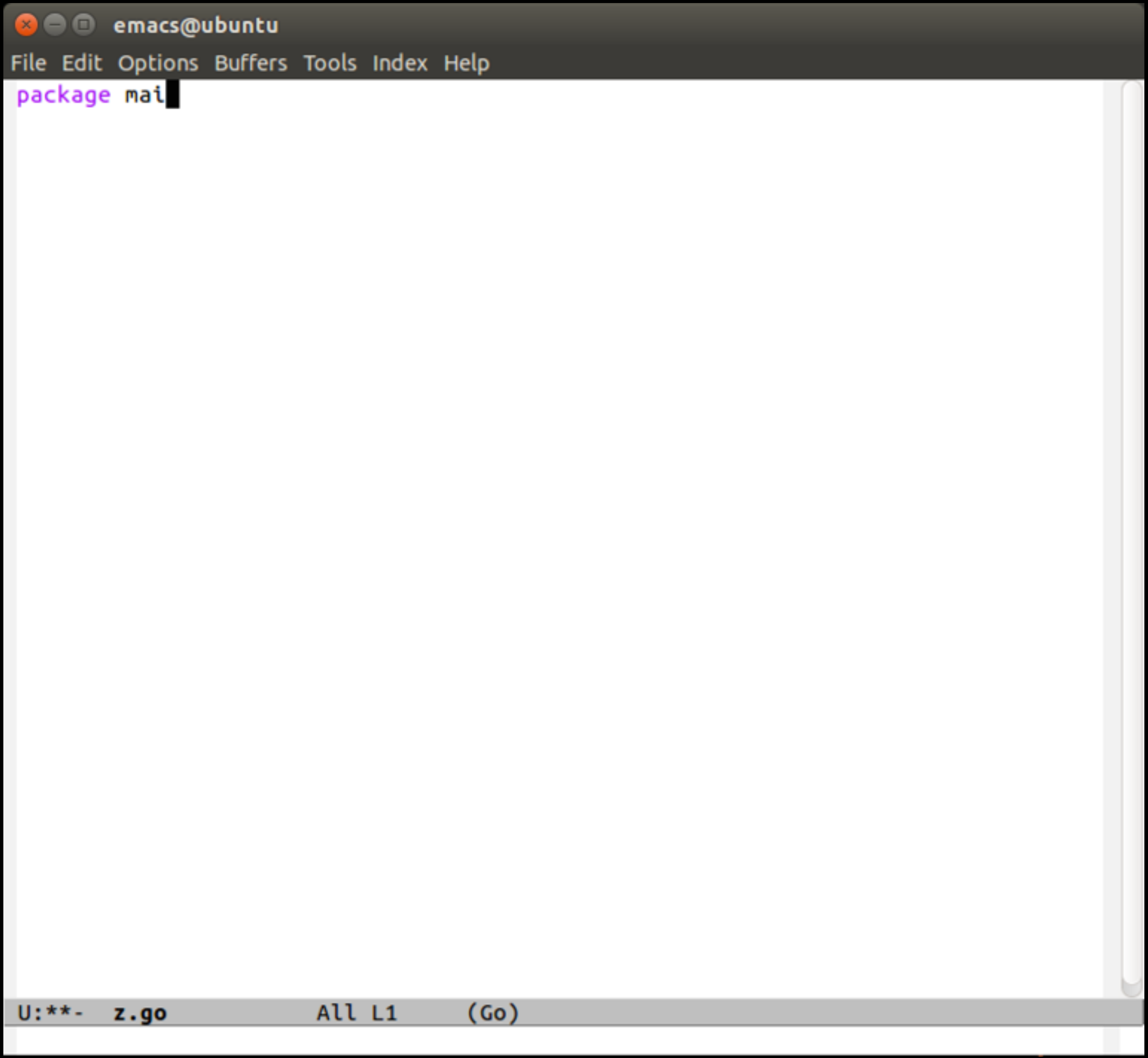
DAMMIT CHLOE
WE DON'T HAVE 18 HOURS
memegenerator.net

# Also…

- Could have used GNU parallel
- But what about errors?
- Also dig output not structured

This looks like a job for…

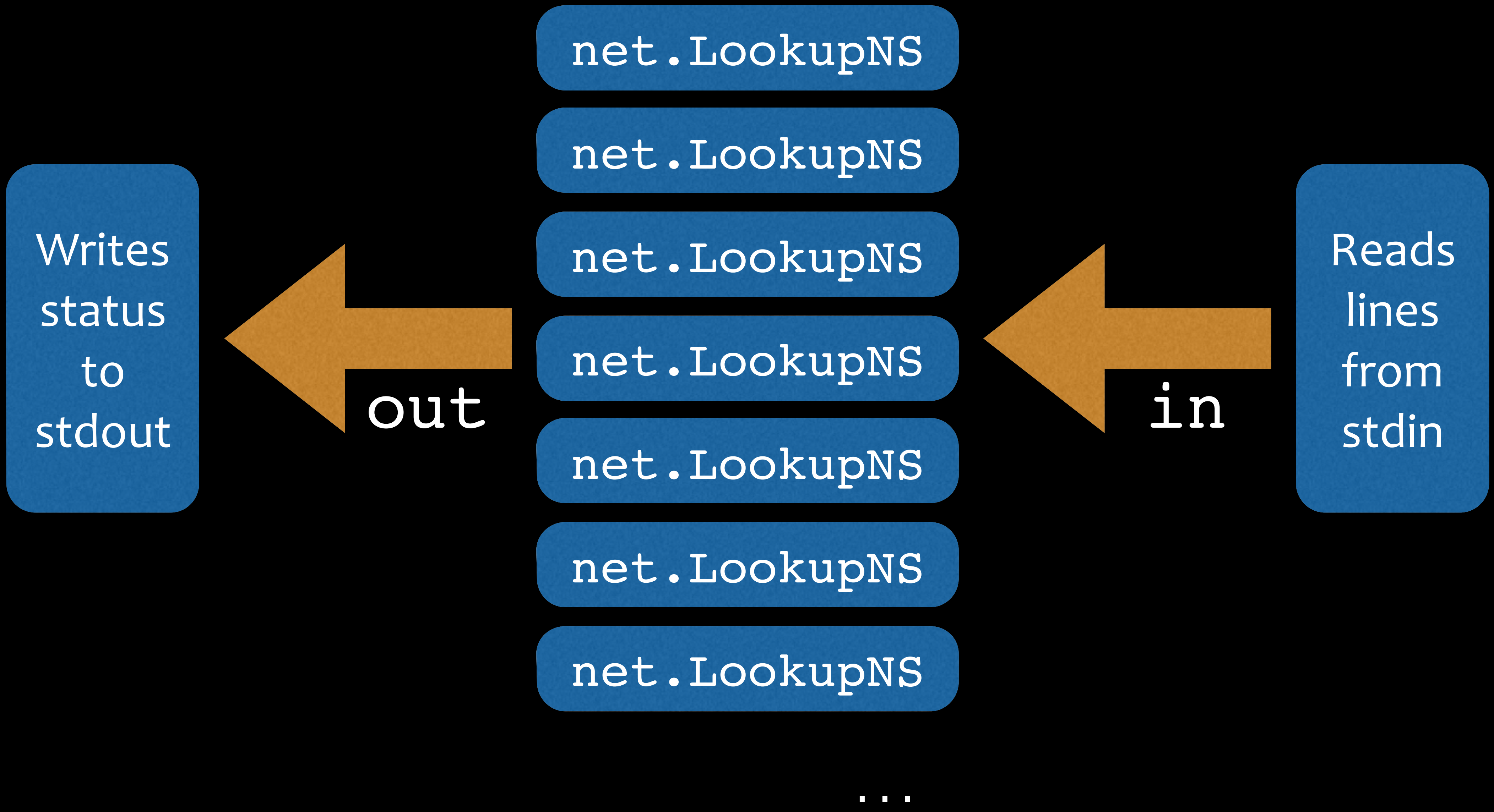




```
package mai
```

```
U:**-  z.go           All L1      (Go)
```

# Rough Architecture



Share `in` and `out` channels across goroutines

# Quick type to encapsulate work and results

```
type lookup struct {
    name string

    // Filled in when NS looked up

    err error
    cloudflare bool
}
```

# Read stdin, stuff down channel

```
var wg sync.WaitGroup

in := make(chan lookup)

wg.Add(1)
go func() {
    s := bufio.NewScanner(os.Stdin)
    for s.Scan() {
        in <- lookup{name: s.Text()}
    }
    if s.Err() != nil {
        log.Fatalf("Error reading STDIN: %s", s.Err())
    }
    close(in)
    wg.Done()
}()
```

Reads lines from stdin

# Read stdin, stuff down channel

```
var wg sync.WaitGroup

in := make(chan lookup)

wg.Add(1)
go func() {
    s := bufio.NewScanner(os.Stdin)
    for s.Scan() {
        in <- lookup{name: s.Text()}
    }
    if s.Err() != nil {
        log.Fatalf("Error reading STDIN: %s", s.Err())
    }
    close(in)
    wg.Done()
}()
```

Reads lines from stdin

# Read stdin, stuff down channel

```
var wg sync.WaitGroup

in := make(chan lookup)

wg.Add(1)
 go func() {
    s := bufio.NewScanner(os.Stdin)
    for s.Scan() {
        in <- lookup{name: s.Text()}
    }
    if s.Err() != nil {
        log.Fatalf("Error reading STDIN: %s", s.Err())
    }
    close(in)
    wg.Done()
 }()
```

Reads lines from stdin

# Read results; write to stdout

Writes status to stdout

```
out := make(chan lookup)

go func() {
    for l := range out {
        state := "OTHER"
        switch {
        case l.err != nil:
            state = "ERROR"
        case l.cloudflare:
            state = "CLOUDFLARE"
        }

        fmt.Printf("%s,%s\n", l.name, state)
    }
}()
```

# Read results; write to stdout

Writes status to stdout

```
out := make(chan lookup)

go func() {
    for l := range out {
        state := "OTHER"
        switch {
        case l.err != nil:
            state = "ERROR"
        case l.cloudflare:
            state = "CLOUDFLARE"
        }

        fmt.Printf("%s,%s\n", l.name, state)
    }
}()
```

# Read results; write to stdout

Writes status to stdout

```
out := make(chan lookup)

go func() {
    for l := range out {
        state := "OTHER"
          switch {
          case l.err != nil:
               state = "ERROR"
          case l.cloudflare:
               state = "CLOUDFLARE"
          }

          fmt.Printf("%s,%s\n", l.name, state)
    }
}()
```

# Do the work

```
for i := 0; i < 1000; i++ {
    wg.Add(1)
    go func() {
        for l := range in {
            nss, err := net.LookupNS(l.name)
            if err != nil {
                l.err = err
            } else {
                for _, ns := range nss {
                    if strings.HasSuffix(ns.Host,
                                        ".ns.cloudflare.com.") {
                        l.cloudflare = true
                        break
                    }
                }
            }
            out <- l
        }
        wg.Done()
    }()
}
```

net.LookupNS

# Do the work

```
for i := 0; i < 1000; i++ {
    wg.Add(1)
    go func() {
        for l := range in {
            nss, err := net.LookupNS(l.name)
            if err != nil {
                l.err = err
            } else {
                for _, ns := range nss {
                    if strings.HasSuffix(ns.Host,
                                         ".ns.cloudflare.com.") {
                        l.cloudflare = true
                        break
                    }
                }
            }
            out <- l
        }
        wg.Done()
    }()
}
```

net.LookupNS

# Do the work

```
for i := 0; i < 1000; i++ {
    wg.Add(1)
    go func() {
        for l := range in {
            nss, err := net.LookupNS(l.name)
            if err != nil {
                l.err = err
            } else {
                for _, ns := range nss {
                    if strings.HasSuffix(ns.Host,
                                        ".ns.cloudflare.com.") {
                        l.cloudflare = true
                        break
                    }
                }
            }
            out <- l
        }
        wg.Done()
    }()
}
```

net.LookupNS

# Do the work

```
for i := 0; i < 1000; i++ {
    wg.Add(1)
    go func() {
        for l := range in {
            nss, err := net.LookupNS(l.name)
              if err != nil {
                  l.err = err
              } else {
                  for _, ns := range nss {
                      if strings.HasSuffix(ns.Host,
                                 ".ns.cloudflare.com.") {
                          l.cloudflare = true
                          break
                      }
                  }
              }
            out <- l
        }
        wg.Done()
    }()
}
```

net.LookupNS

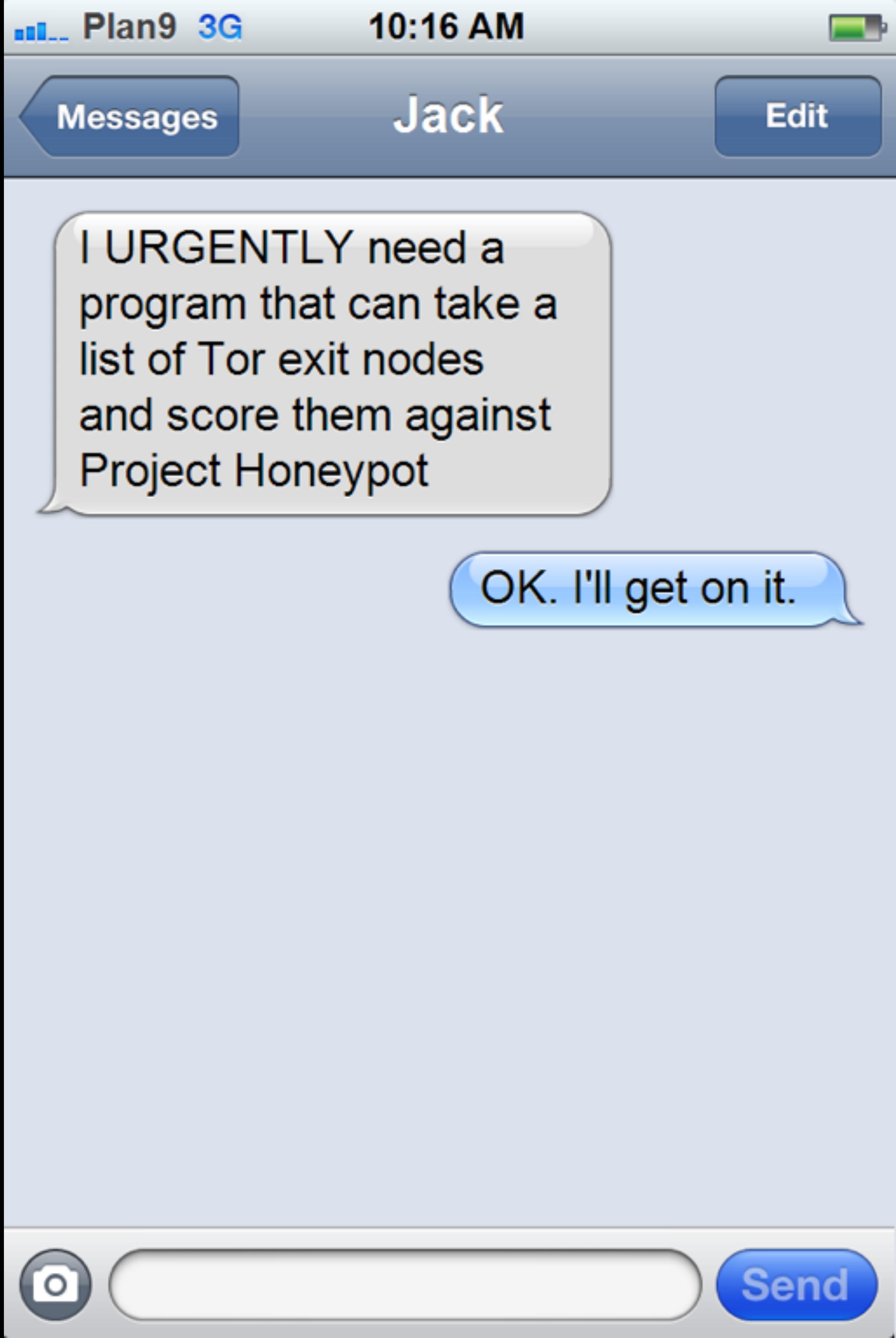
# Easy concurrency

- 75 lines of Go
- Highly concurrent
- Simple to understand
- Go standard packages are great

```
go run z.go < zones.txt
```

```go
package main

import (
    "bufio"
    "fmt"
    "log"
    "net"
    "os"
    "strings"
    "sync"
)

type lookup struct {
    name string
    err error
    cloudflare bool
}

func main() {
    var wg sync.WaitGroup

    in := make(chan lookup)

    wg.Add(1)
    go func() {
        s := bufio.NewScanner(os.Stdin)
        for s.Scan() {
            in <- lookup{name: s.Text()}
        }
        if s.Err() != nil {
            log.Fatalf("Error reading STDIN: %s", s.Err())
        }
        close(in)
        wg.Done()
    }()

    out := make(chan lookup)

    go func() {
        for l := range out {
            state := "OTHER"
            switch {
            case l.err != nil:
                state = "ERROR"
            case l.cloudflare:
                state = "CLOUDFLARE"
            }
            fmt.Printf("%s,%s\n", l.name, state)
        }
    }()

    for i := 0; i < 1000; i++ {
        wg.Add(1)
        go func() {
            for l := range in {
                nss, err := net.LookupNS(l.name)
                if err != nil {
                    l.err = err
                } else {
                    for _, ns := range nss {
                        if strings.HasSuffix(ns.Host, ".ns.cloudflare.com.") {
                            l.cloudflare = true
                            break
                        }
                    }
                }
                out <- l
            }
            wg.Done()
        }()
    }

    wg.Wait()
    close(out)
}
```

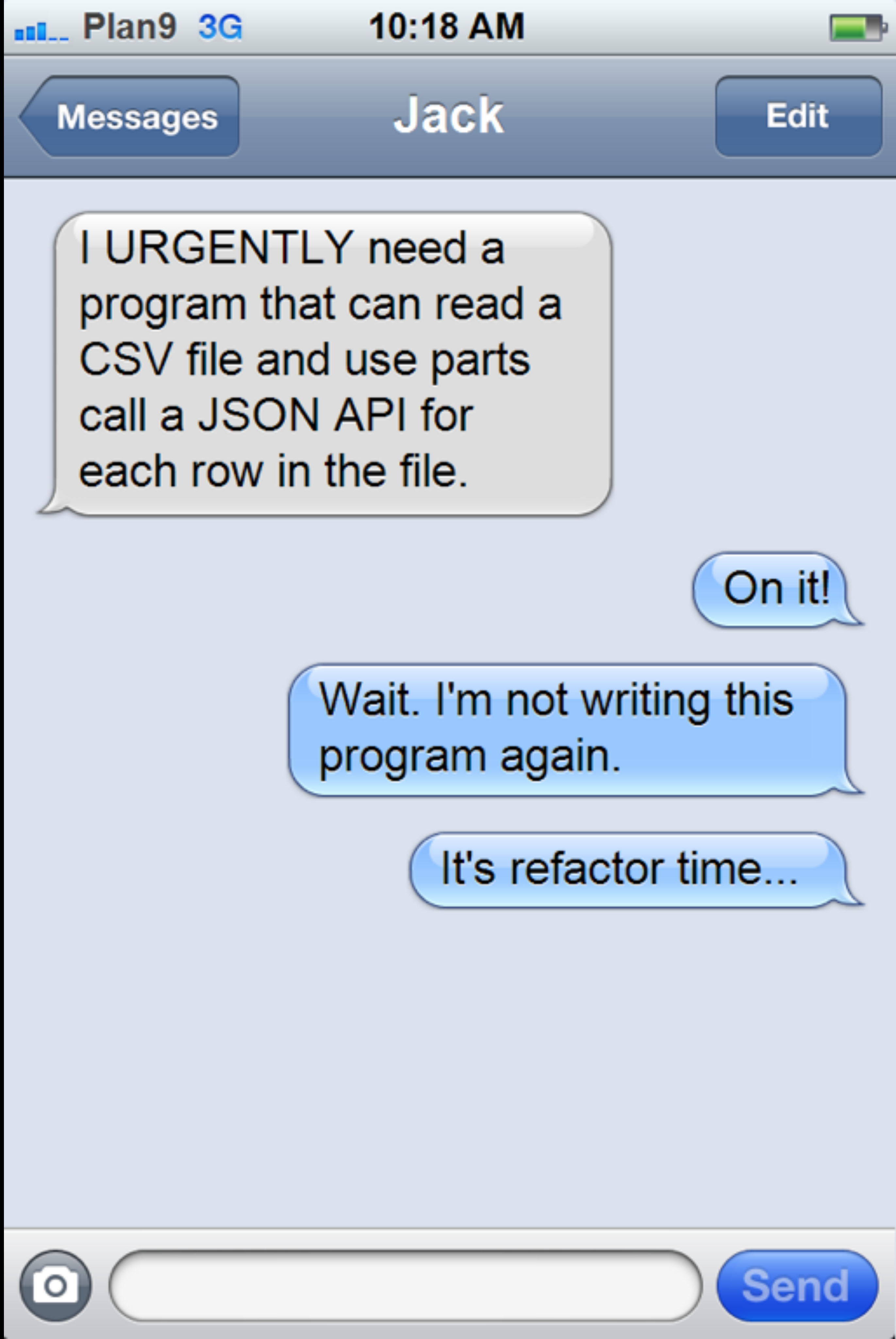
Five minutes later…



I URGENTLY need a program that can take a list of Tor exit nodes and score them against Project Honeypot

OK. I'll get on it.

Two minutes later…



I URGENTLY need a program that can read a CSV file and use parts call a JSON API for each row in the file.

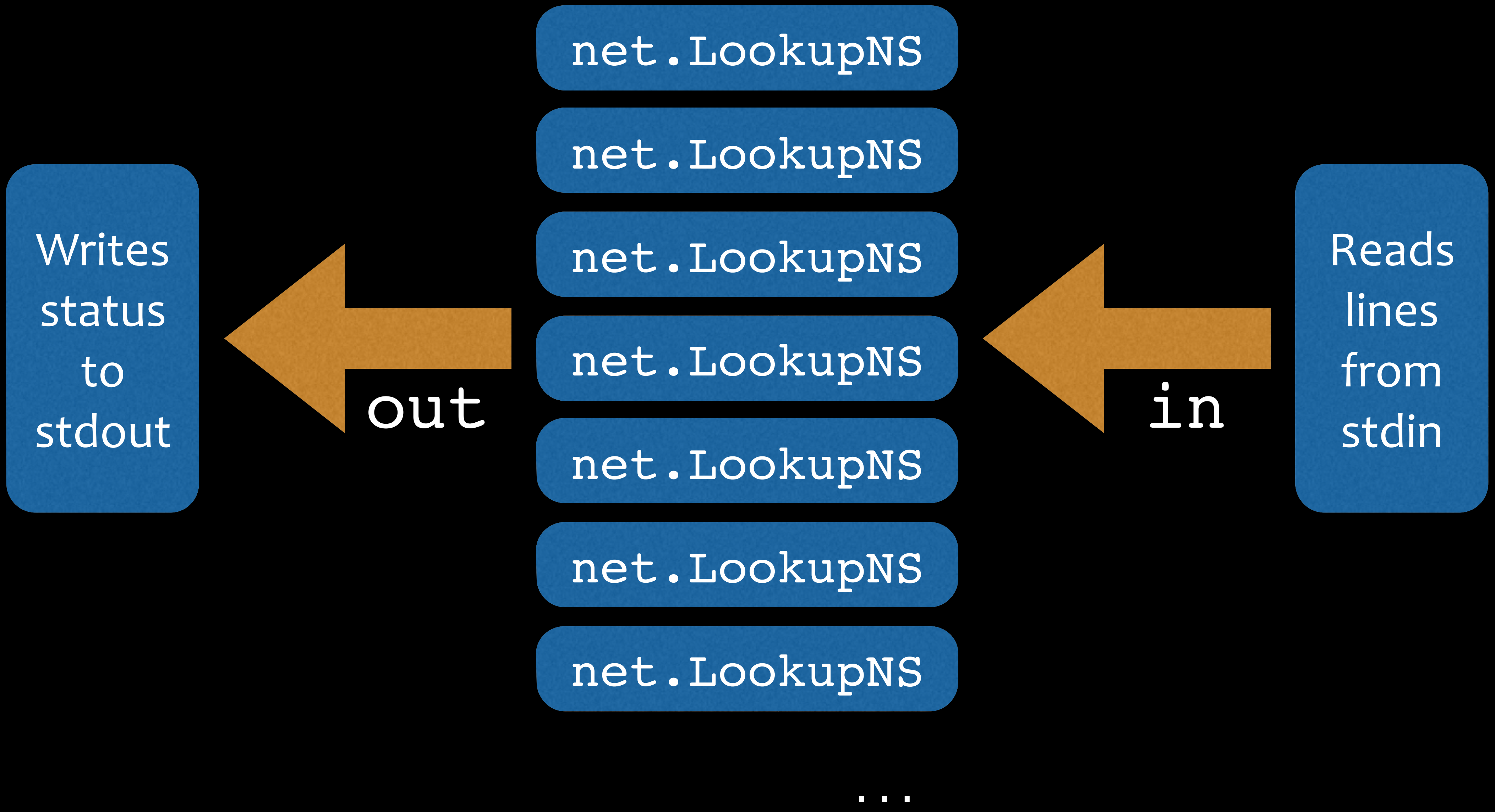
On it!

Wait. I'm not writing this program again.

It's refactor time...

It’s **`interface`** time!

# Rough Architecture



Writes status to stdout

out

net.LookupNS

net.LookupNS

net.LookupNS

net.LookupNS

net.LookupNS

net.LookupNS

net.LookupNS

...

in

Reads lines from stdin

# factory and task

```
type factory interface {
    make(line string) task
}

type task interface {
    process()
    print()
}
```

# Implement `factory`

```
type lookupFactory struct {
}

func (f *lookupFactory) make(line string) task {
    return &lookup{name: line}
}
```

# Implement task

```
type lookup struct {
    name string
    err error
    cloudflare bool
}

func (l *lookup) process() {
    nss, err := net.LookupNS(l.name)
    if err != nil {
        l.err = err
    } else {
        for _, ns := range nss {
            if strings.HasSuffix(ns.Host, ".ns.cloudflare.com.") {
                l.cloudflare = true
                break
            }
        }
    }
}
```

# Implement task

```
func (l *lookup) print() {
    state := "OTHER"
    switch {
    case l.err != nil:
        state = "ERROR"
    case l.cloudflare:
        state = "CLOUDFLARE"
    }
    fmt.Printf("%s,%s\n", l.name, state)
}
```

# run

```
func run(f factory) {
    var wg sync.WaitGroup

    in := make(chan task)

    wg.Add(1)
    go func() {
        s := bufio.NewScanner(os.Stdin)
        for s.Scan() {
            in <- f.make(s.Text())
        }
        if s.Err() != nil {
            log.Fatalf("Error reading STDIN: %s", s.Err())
        }
        close(in)
        wg.Done()
    }()
```

# run

```
func run(f factory) {
    var wg sync.WaitGroup

    in := make(chan task)

    wg.Add(1)
    go func() {
        s := bufio.NewScanner(os.Stdin)
        for s.Scan() {
                in <- f.make(s.Text())
           }
        if s.Err() != nil {
            log.Fatalf("Error reading STDIN: %s", s.Err())
        }
        close(in)
        wg.Done()
    }()
```

# run

```
out := make(chan task)

go func() {
    for t := range out {
        t.print()
    }
}()
```

…

# run

```
out := make(chan task)

go func() {
    for t := range out {
        t.print()
    }
}()
```

…

# run

```
    for i := 0; i < 1000; i++ {
        wg.Add(1)
        go func() {
            for t := range in {
                t.process()
                out <- t
            }
            wg.Done()
        }()
    }

    wg.Wait()
    close(out)
}
```

# run

```
    for i := 0; i < 1000; i++ {
        wg.Add(1)
        go func() {
            for t := range in {
                t.process()
                out <- t
            }
            wg.Done()
        }()
    }

    wg.Wait()
    close(out)
}
```

# run

```
    for i := 0; i < 1000; i++ {
        wg.Add(1)
        go func() {
            for t := range in {
                t.process()
                out <- t
            }
            wg.Done()
        }()
    }

    wg.Wait()
    close(out)
}
```

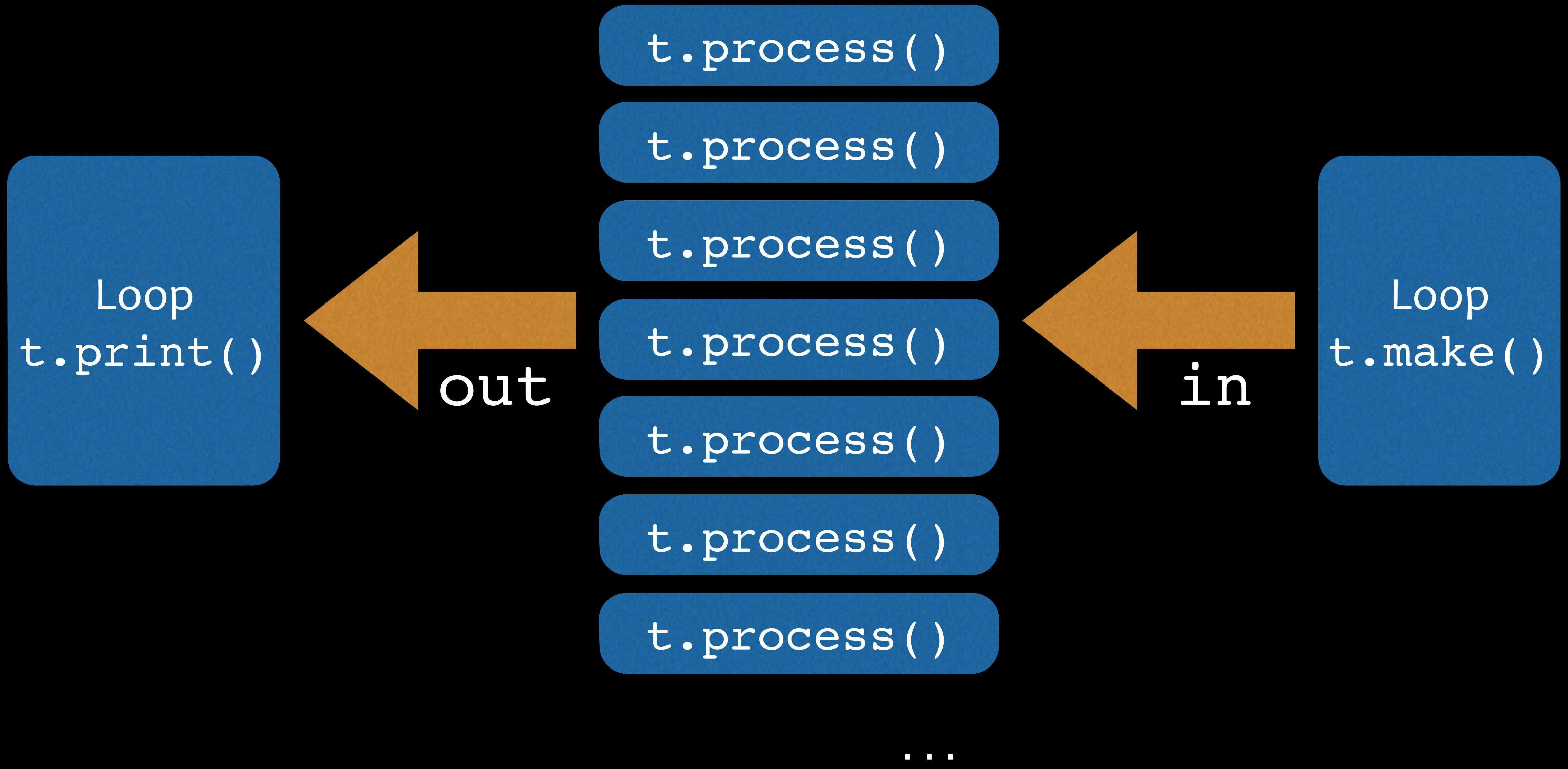
# run

```
    for i := 0; i < 1000; i++ {
        wg.Add(1)
        go func() {
            for t := range in {
                t.process()
                out <- t
            }
            wg.Done()
        }()
    }

    wg.Wait()
    close(out)
}
```

# main()

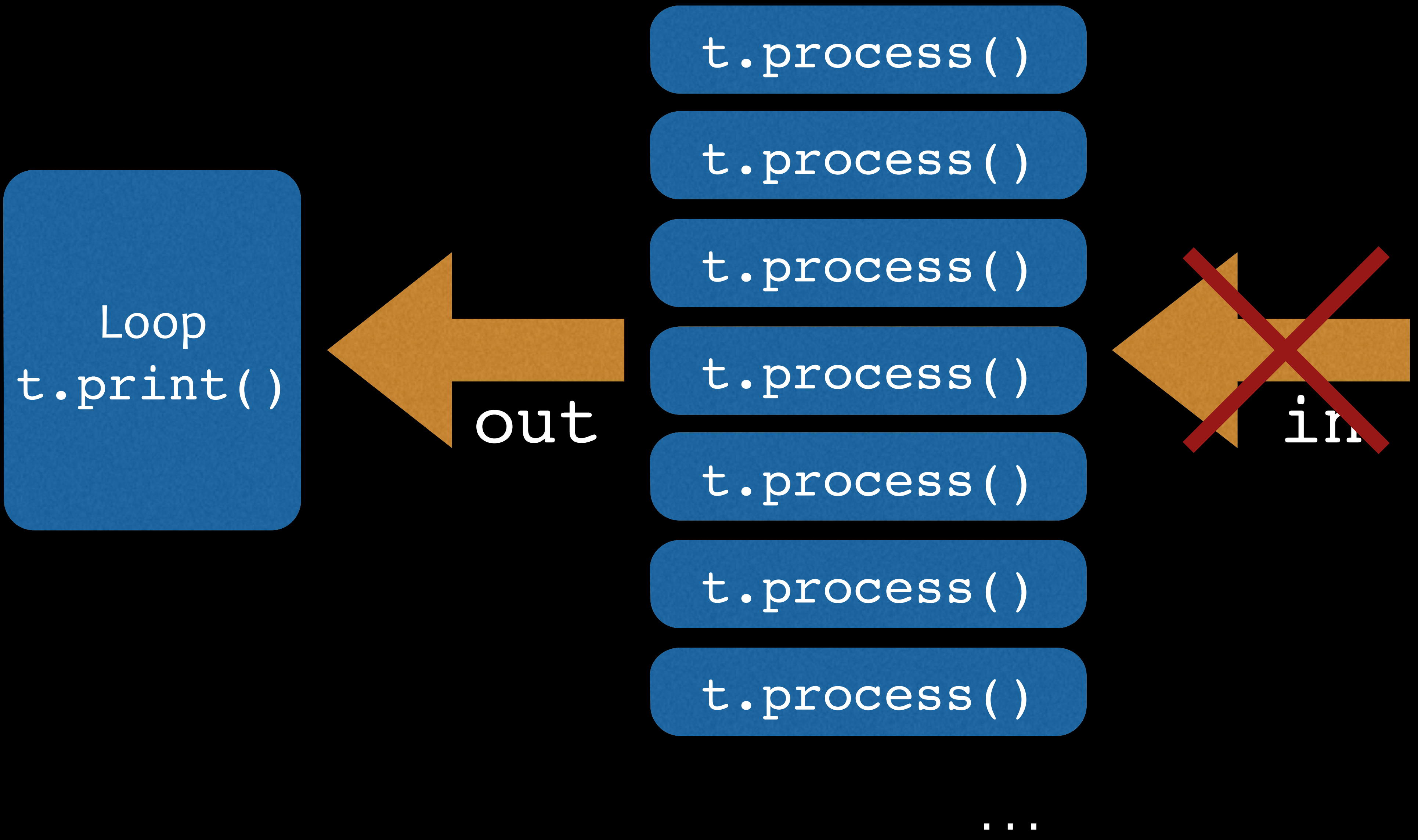
```
func main() {
    run(&lookupFactory{})
}
```

# Starting State



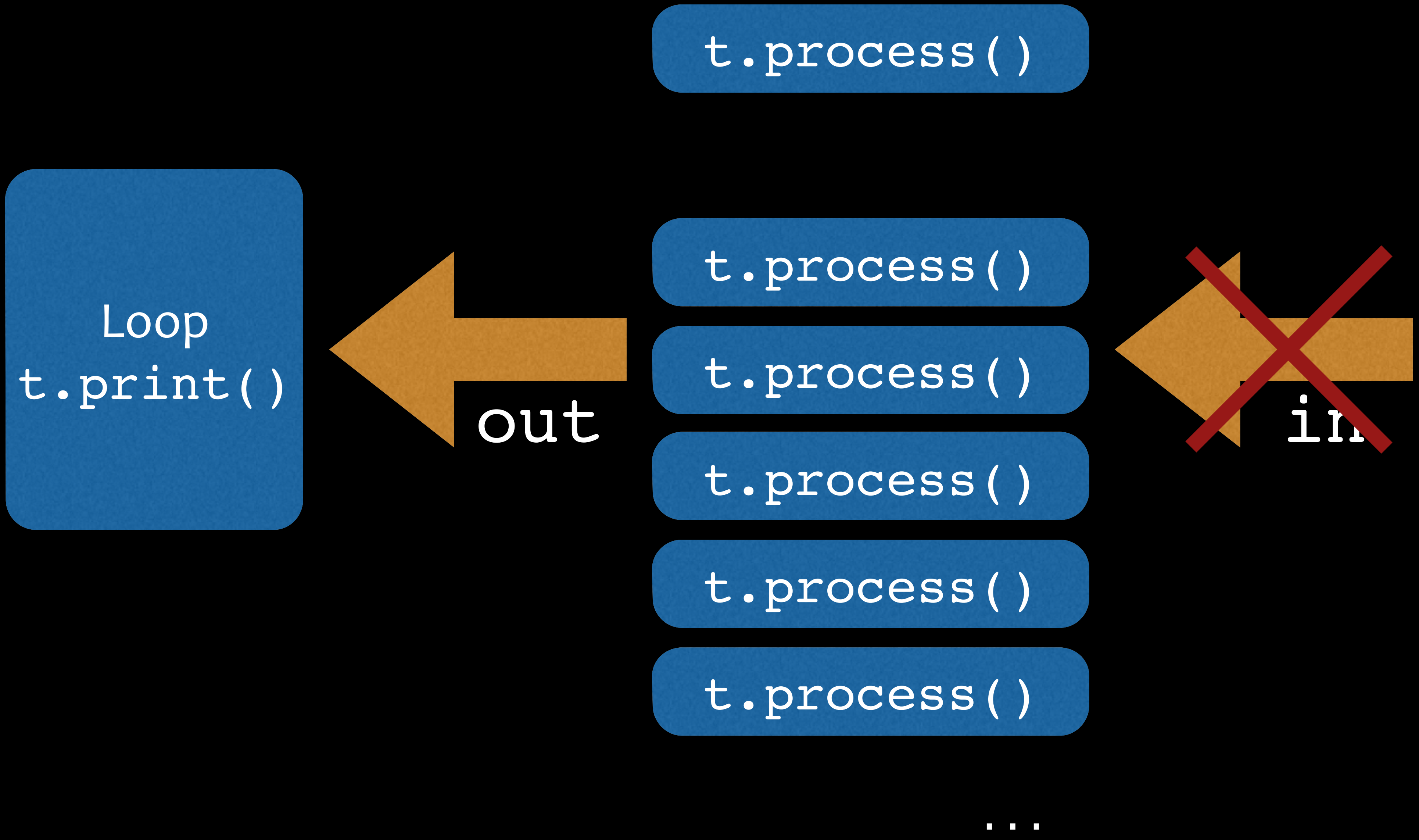
`wg` has count of 1001

# stdin empty; `in` closed



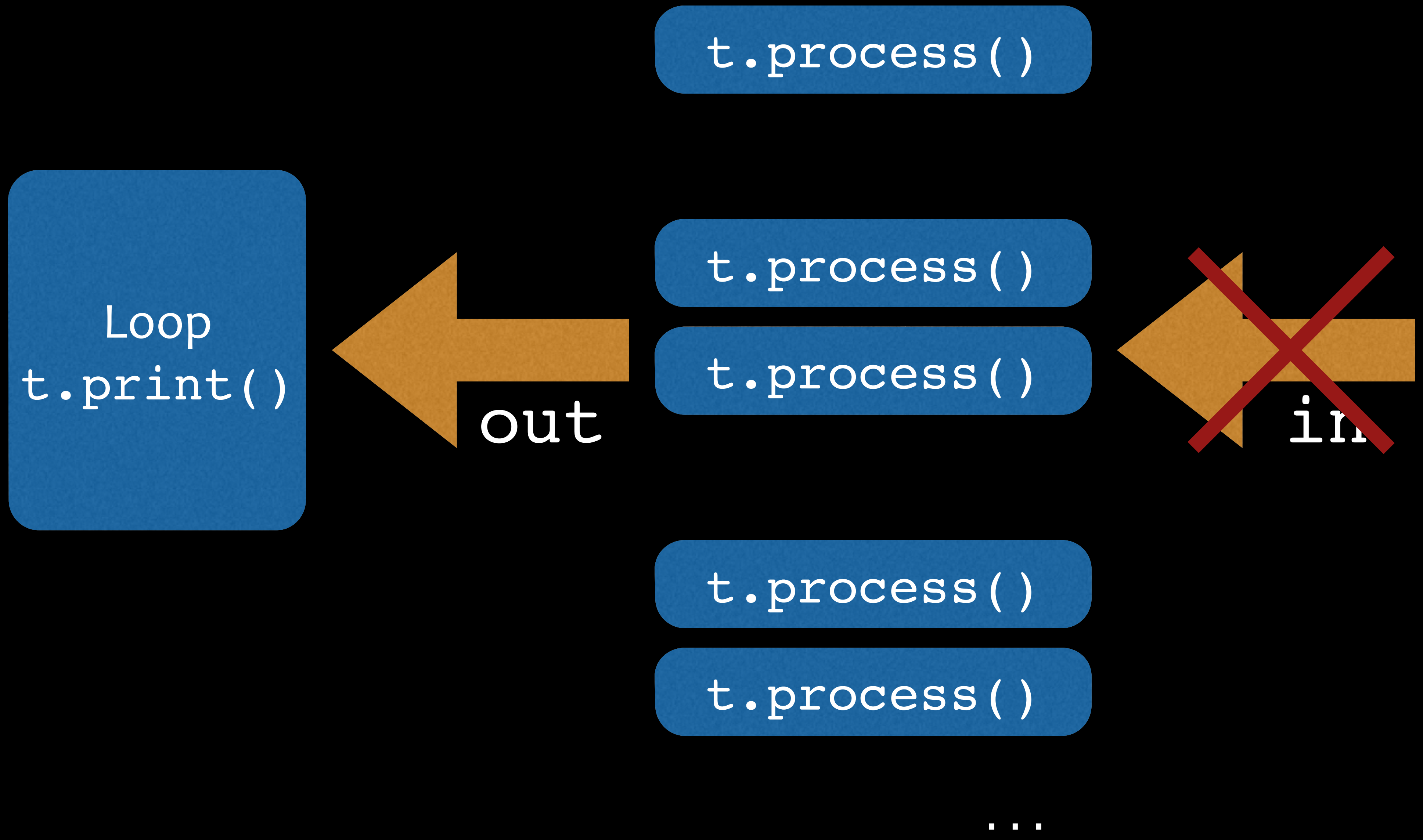
`wg` has count of 1000

# Workers start terminating



`wg` has count of 999

# Workers start terminating



t.process()

Loop
t.print()

out

t.process()

t.process()

in

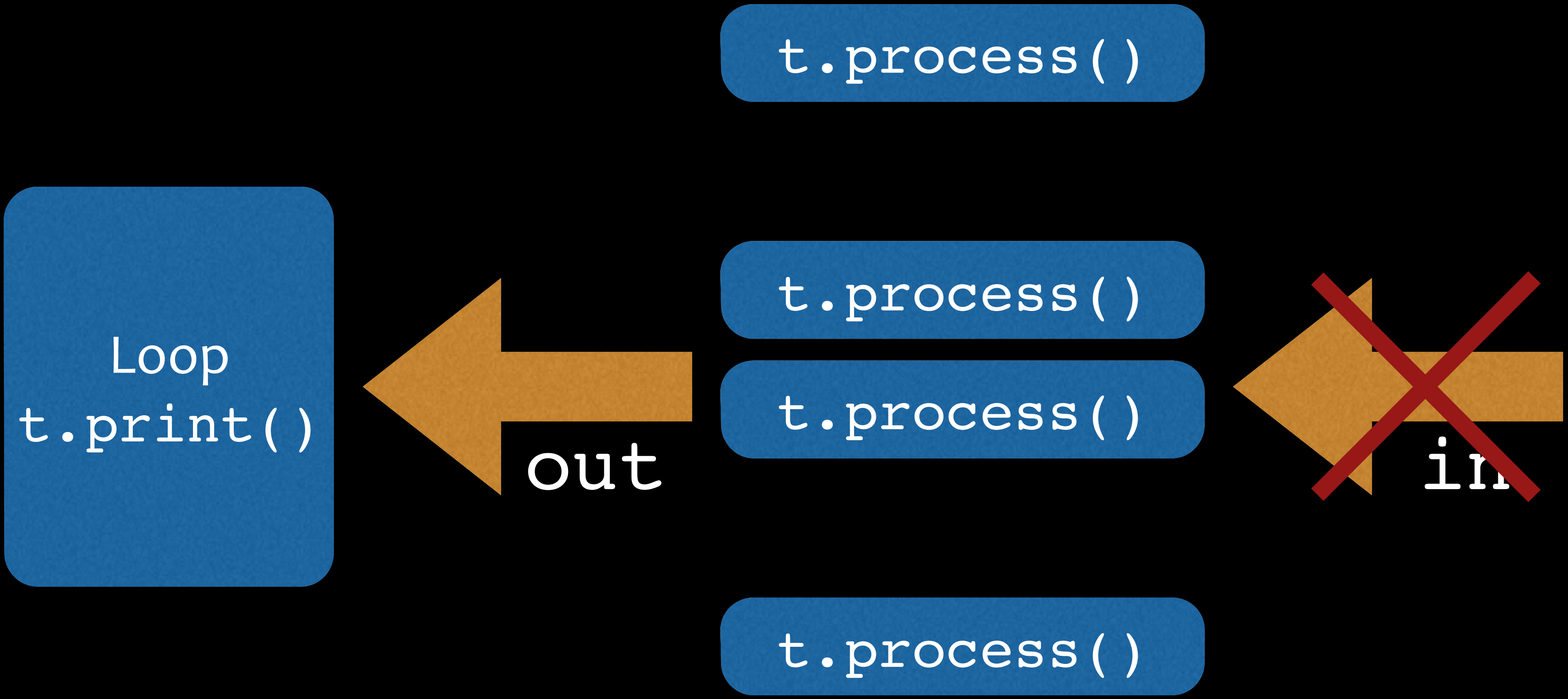
t.process()

t.process()

...

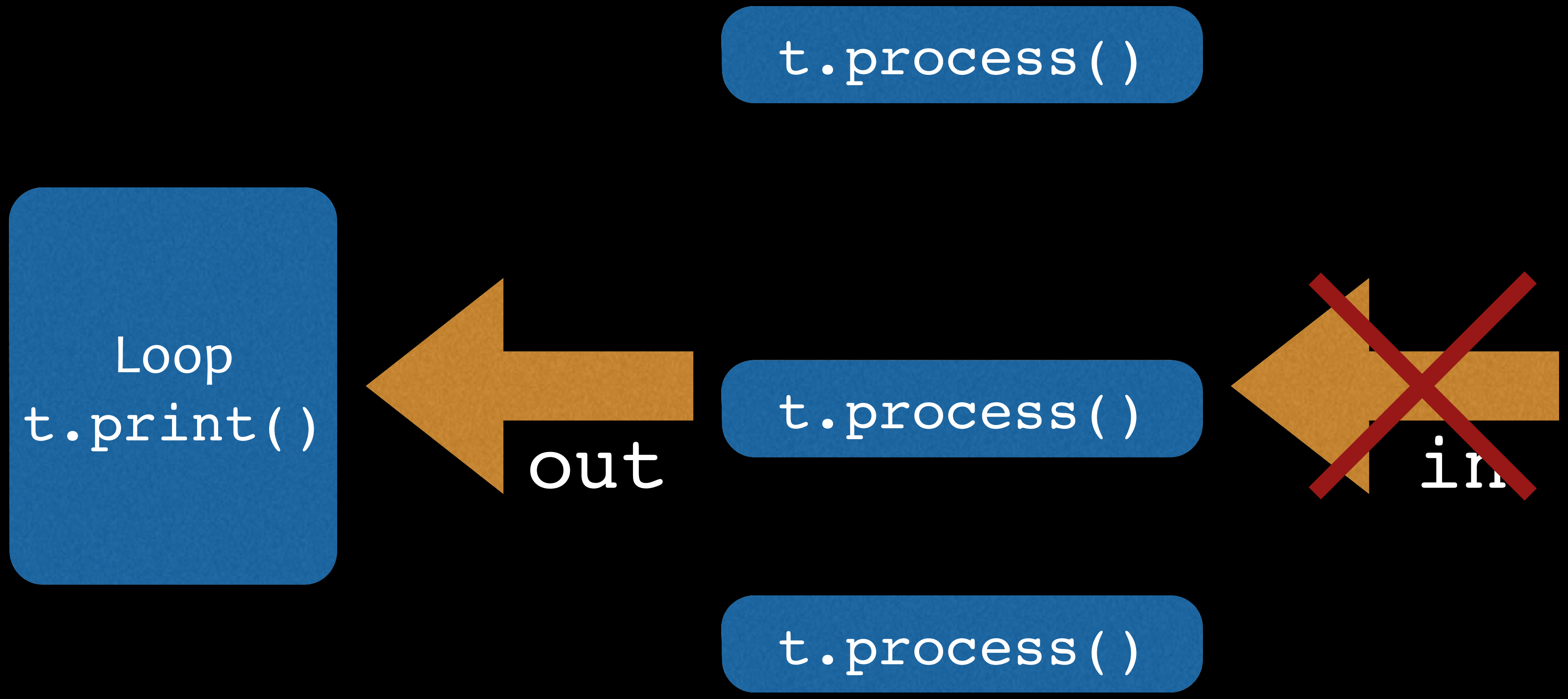
`wg` has count of 998

# Workers start terminating



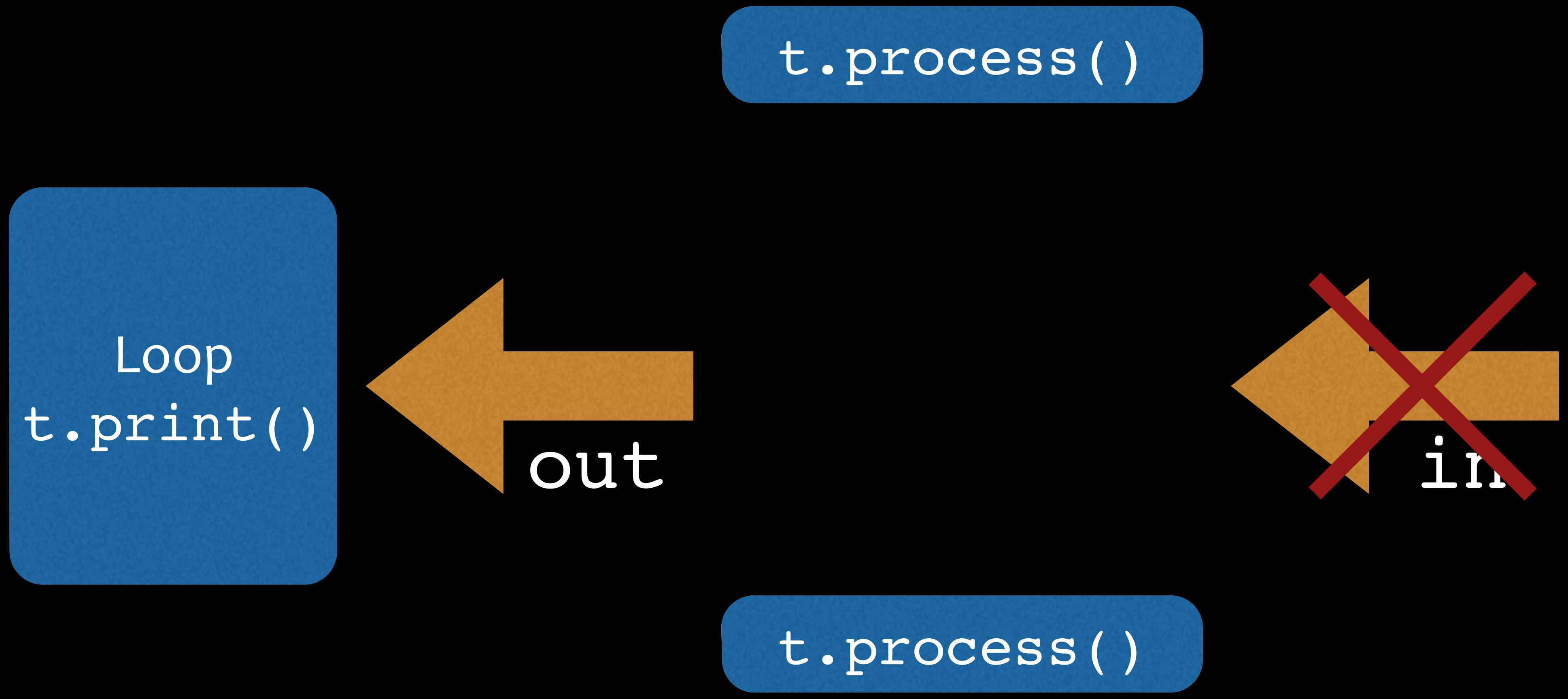
`wg` has count of 4

# Workers start terminating



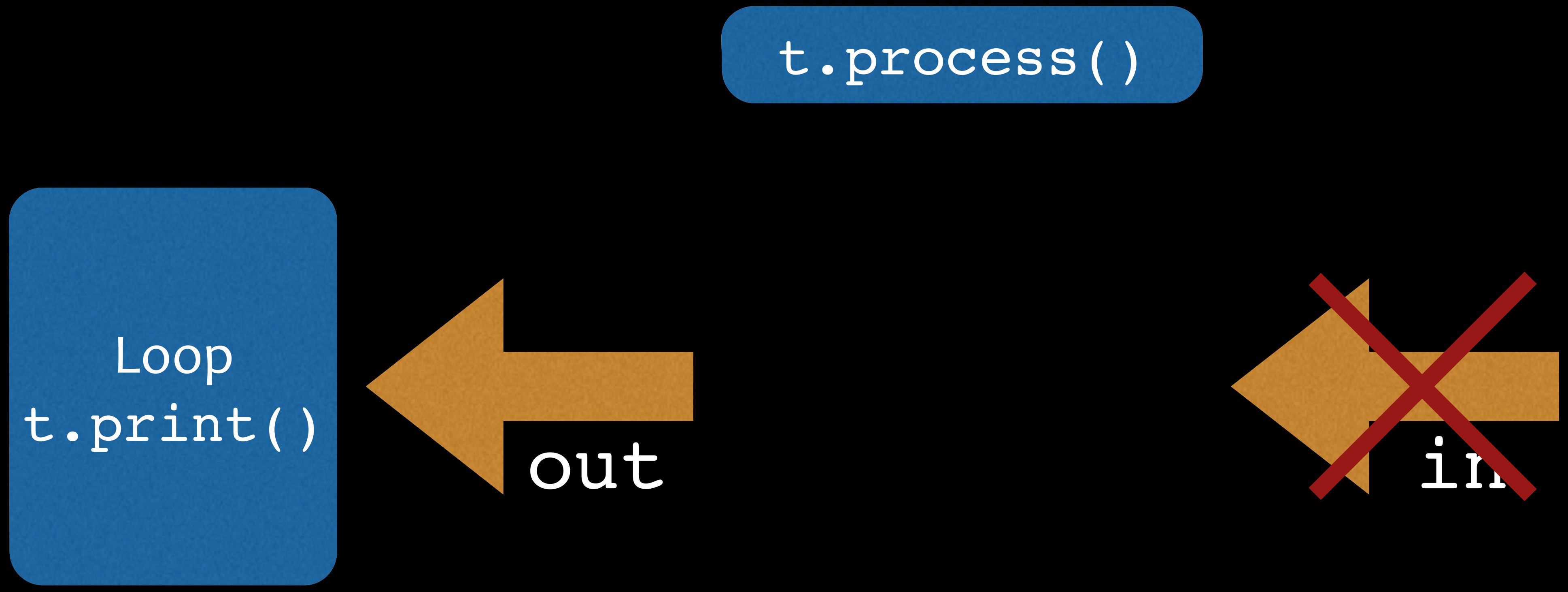
`wg` has count of 3

# Workers start terminating



`wg` has count of 2

# Workers start terminating



t.process()

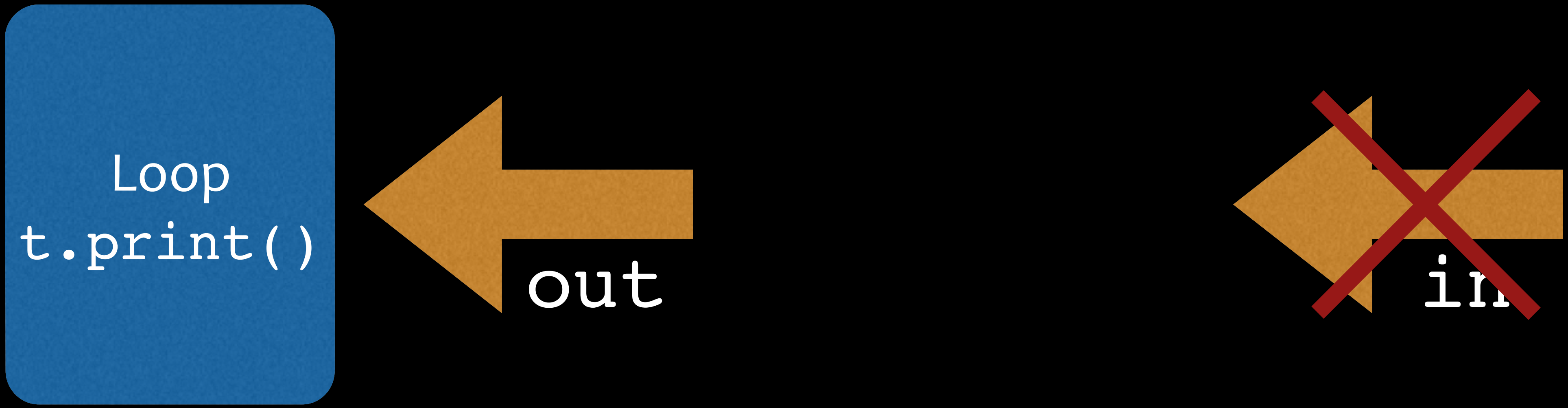
Loop
t.print()

out

in

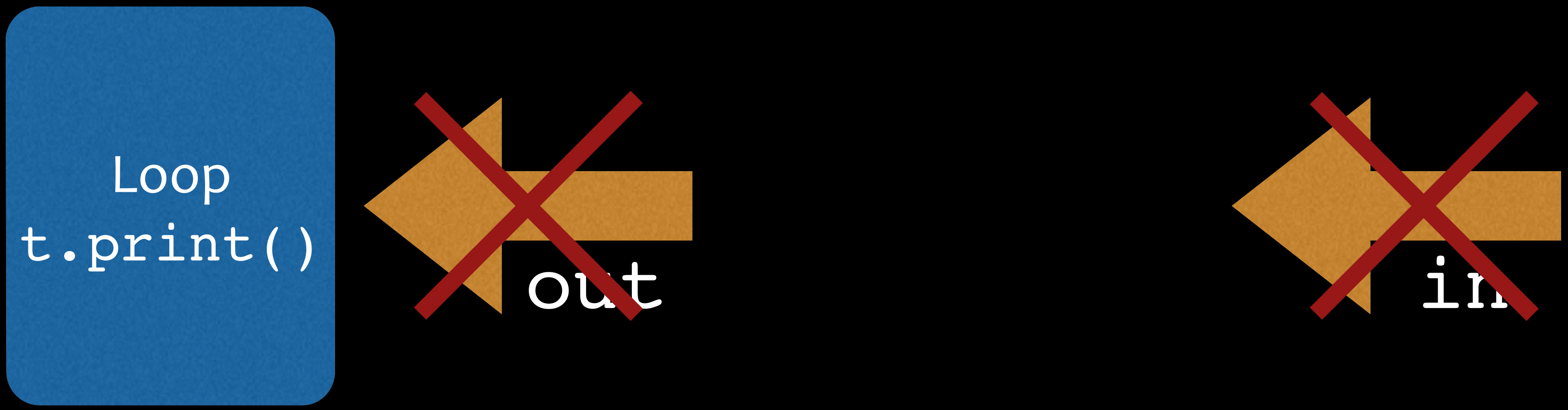
`wg` has count of 1

# Workers finish terminating

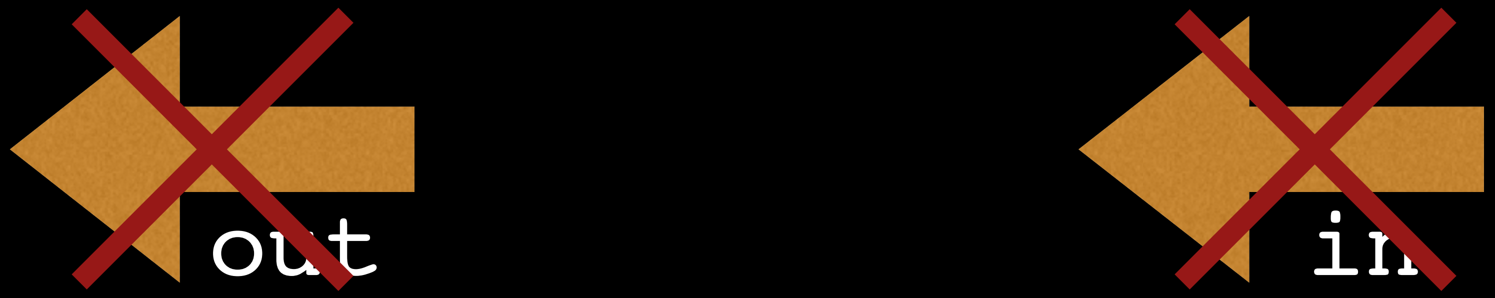


`wg` has count of 0

# wg.Wait(); close(out)



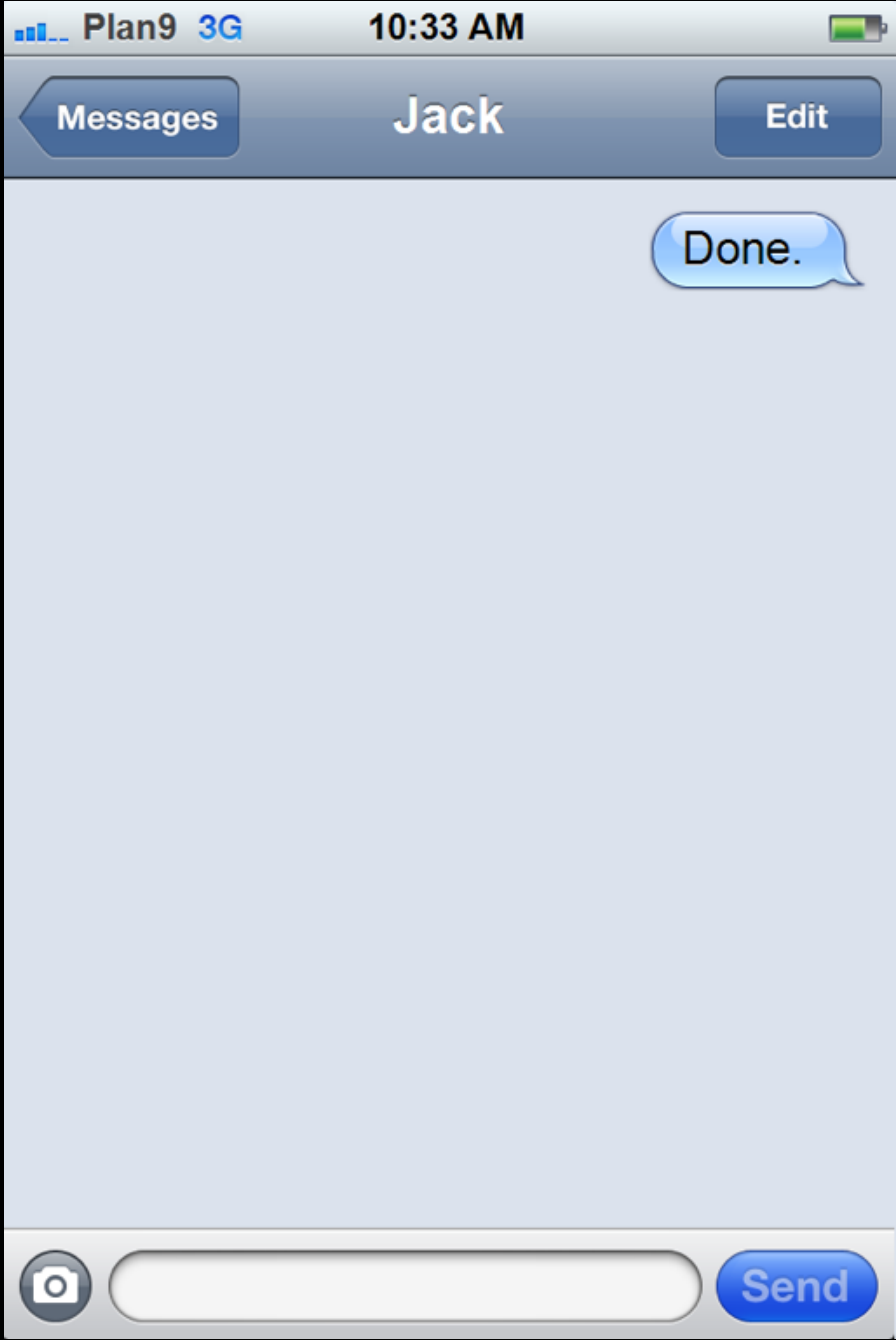
# Termination



# Conclusion

- Trivially easy concurrency
- Refactor for generality with minor code changes
- Only went from 75 to 103 lines
- Go is good for big and small programs

https://github.com/jgrahamc/dotgo



Done.