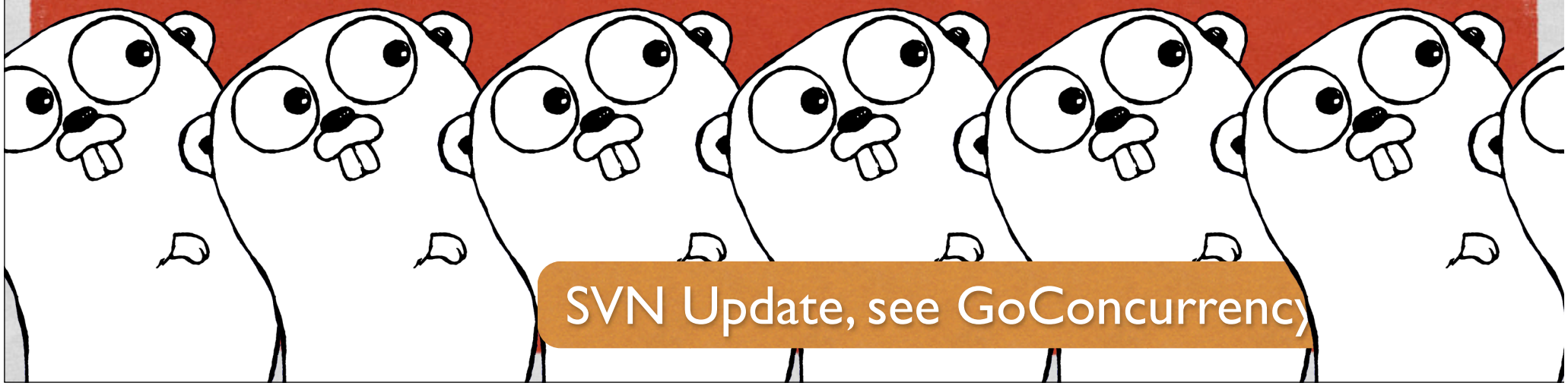
EMBEDDING AND CONCURRENCY

Curt Clifton
Rose-Hulman Institute of Technology



SVN Update, see GoConcurrency

EMBEDDING: INTERFACES AND STRUCTS

EMBEDDING INTERFACES

```
type ReadWriter interface {
    Reader
    Writer
}
```

```
type ReadWriter interface {
    Read(p []byte) (n int, err os.Error)
    Write(p []byte) (n int, err os.Error)
}

type Reader interface {
    Read(p []byte) (n int, err os.Error)
}

type Writer interface {
    Write(p []byte) (n int, err os.Error)
}
```

EMBEDDING STRUCTS

```
type Lockable struct {
    locked bool
}

type T struct {
    name string
    Lockable
}
```

T "inherits" Lockable's fields and methods!

CONCURRENCY PATTERNS

DON’T COMMUNICATE BY SHARING MEMORY

SHARE MEMORY BY COMMUNICATING

Q1

GOROUTINES

- Run in parallel
- Share address space
- Lightweight
- Multiplexed onto multiple threads automatically

IDIOM: ASYNC EXECUTION

```
func Announce(message string, delay int64) {
    go func() {
        time.Sleep(delay)
        fmt.Println(message)
    }()
}
```

Q2

IDIOM: FUTURES

```
done := make(chan int)
go func() {
    done <- list.Foldl(sum, 0)
}()
doSomethingForAWhile()
sum <- done
```

Q3

IDIOM: CHANNELS AS SEMAPHORES

```
var sem = make(chan int, MaxAllowed)

func handle(r *Request) {
    sem <- 1    // Blocks if MaxAllowed process calls are running
    process(r)  // Guarded resource
    <-sem       // Done; enable next request to run.
}

func Serve(queue chan *Request) {
    for {
        req := <-queue
        go handle(req)
    }
}
```

Q4

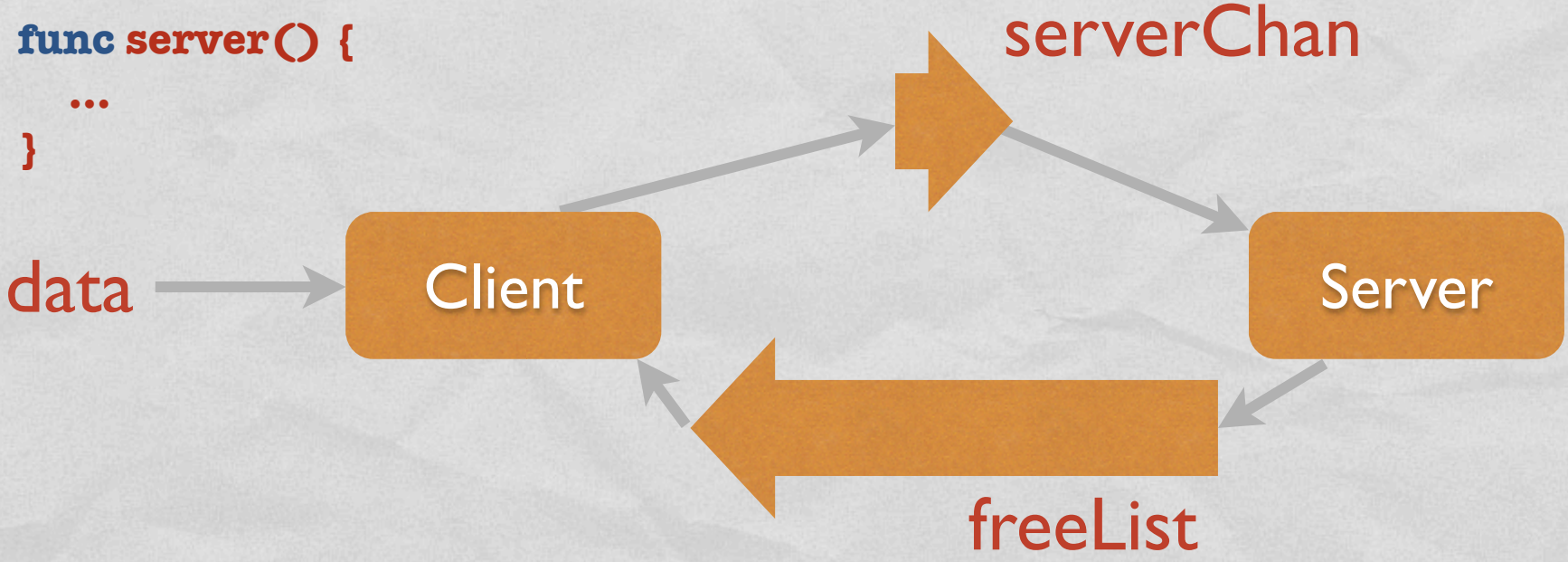
IDIOM: AVOIDING SEMAPHORES

```
func handle(queue chan *Request) {
    for r := range queue {
        process(r)
    }
}

func Serve(queue chan *Request, quit chan bool) {
    // Start handlers
    for i := 0; i < MaxAllowed; i++ {
        go handle(queue)
    }
    <-quit // keep server running until told
}
```

Q5

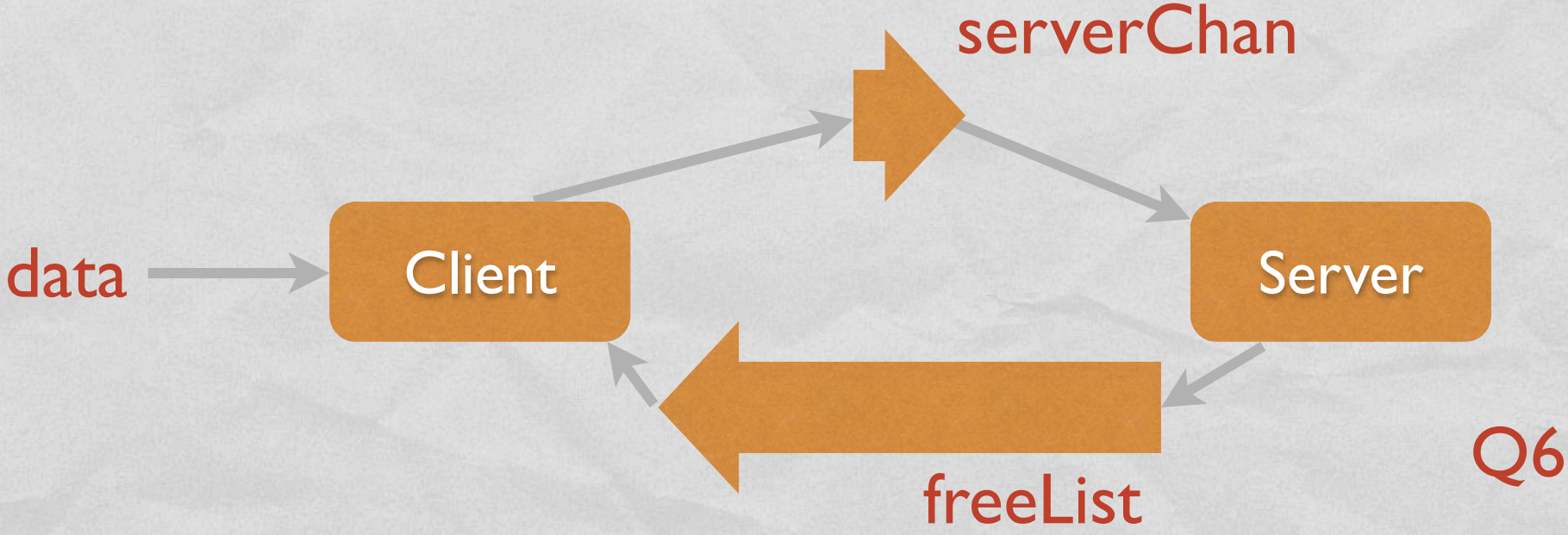
IDIOM: RESOURCE POOL

```
var freeList = make(chan *Buffer, 100)
var serverChan = make(chan *Buffer)

func client() {
    ...
}

func server() {
    ...
}
```



```
var freeList = make(chan *Buffer, 100)
var serverChan = make(chan *Buffer)

func client() {
    for {
        b, ok := <-freeList  // nonblocking
        if !ok {
            b = new(Buffer)
        }
        load(b)
        serverChan <- b
    }
}
```

```
func server() {
    for {
        b := <-serverChan    // blocking
        process(b)
        _ = freeList <- b    // non-blocking
    }
}
```



Q6

Photo by Ian Sane - http://flic.kr/p/86XAuo

EXERCISES

See GoConcurrency/parmap.go and parsort.go