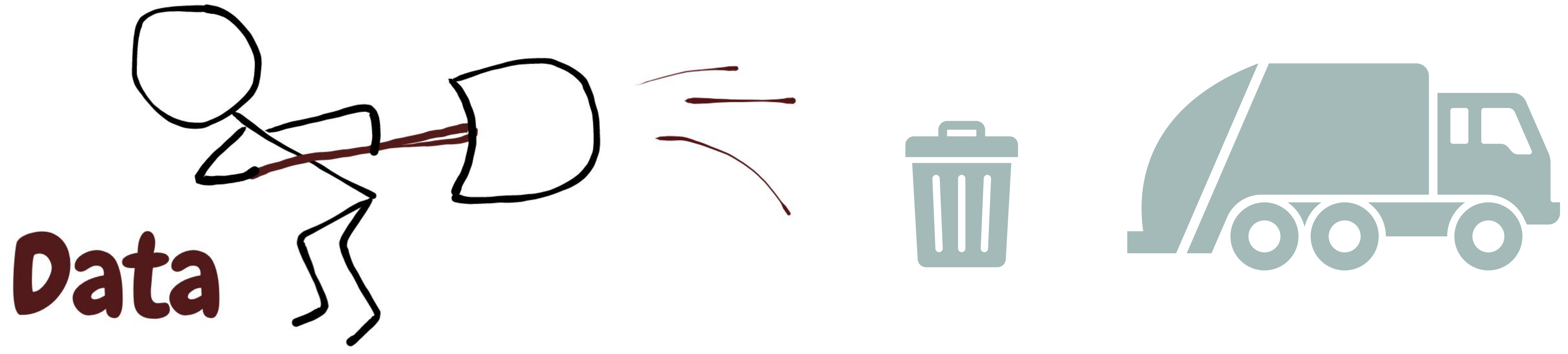
THE GOLANG GARBAGE COLLECTOR

Maya Rosecrance



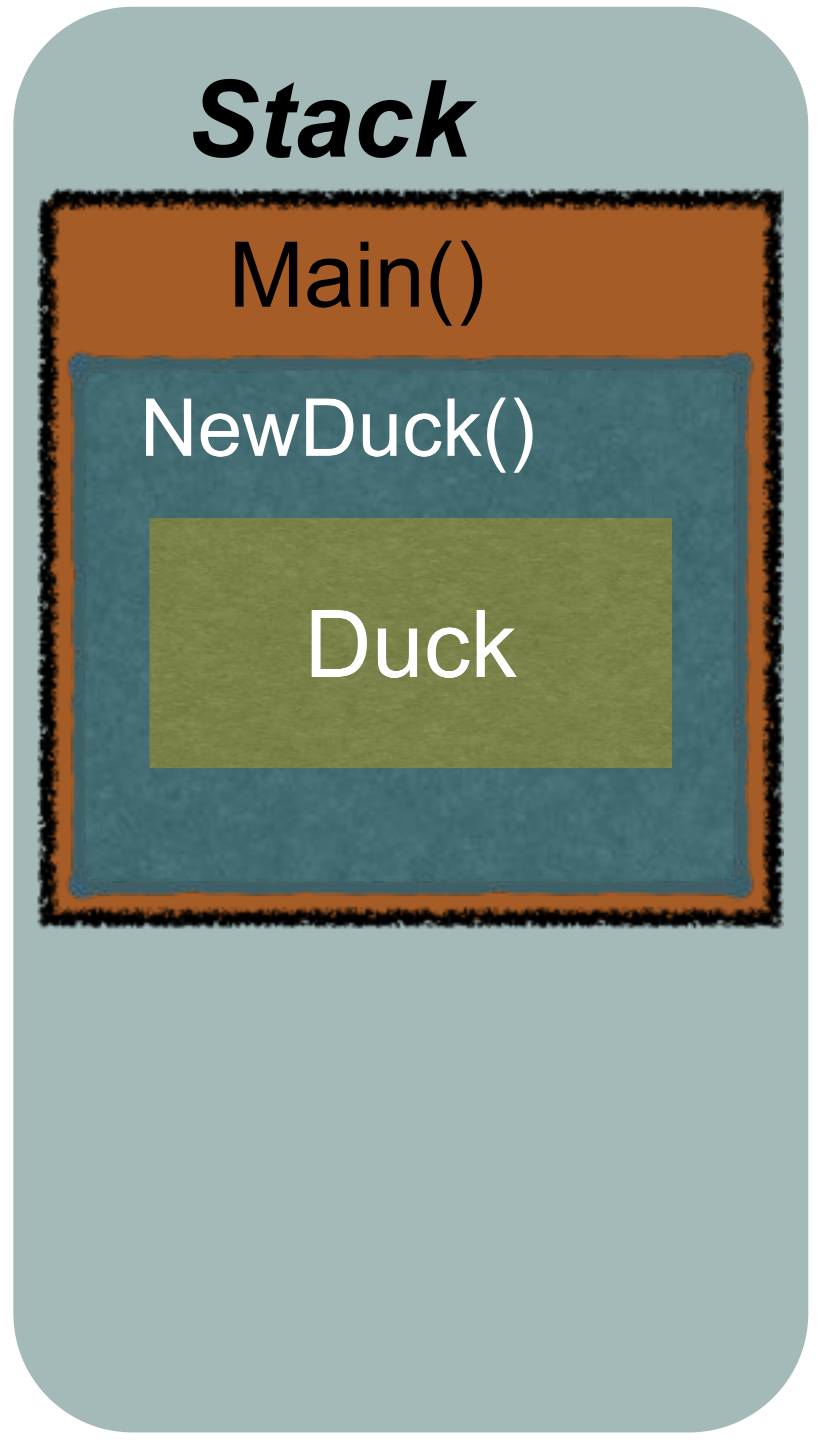
MEMORY MANAGEMENT SYSTEM

- Memory Allocation
- Marking of live objects
- Freeing space used by dead objects

CONTENTS

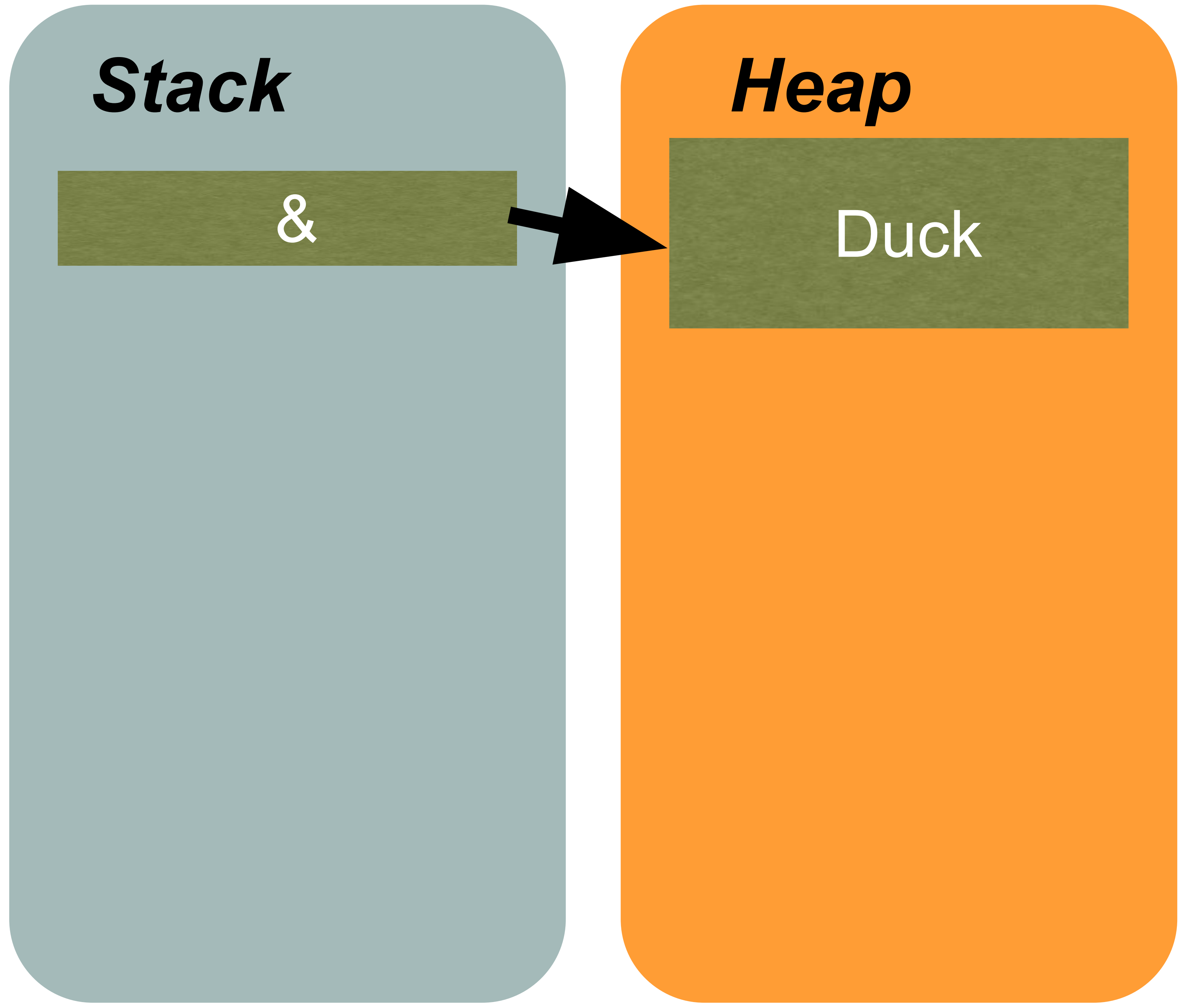
WHEN IS GARBAGE GENERATED

```
func main() {
 NewDuck()
}


type Duck struct {}


func NewDuck() Duck {
 return Duck{}
}
```



WHEN IS GARBAGE GENERATED

```go
func main() {
 NewDuck()
}


type Duck struct {}


//go:noinline
func NewDuck() *Duck {
 return &Duck{}
}
```



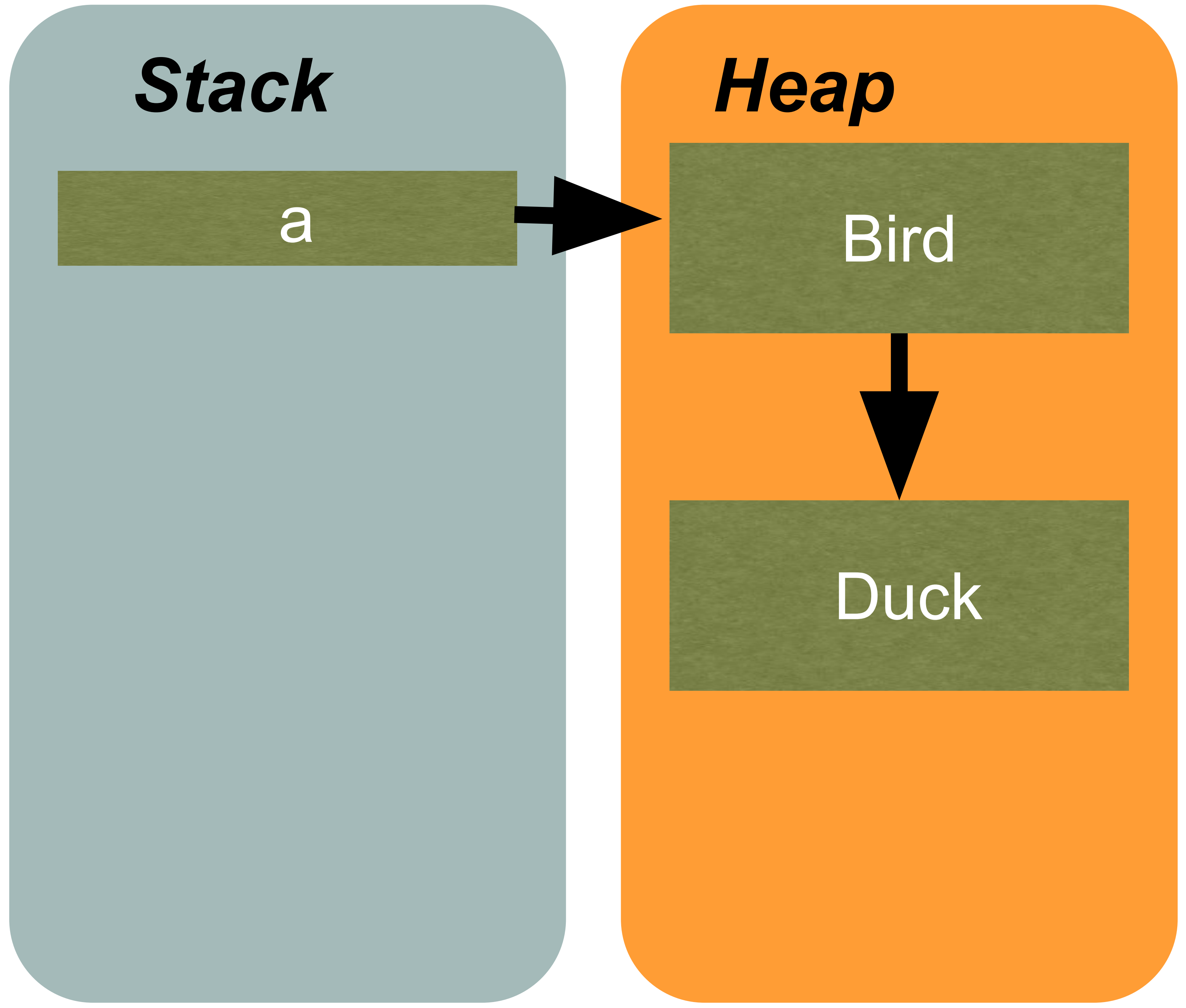
WHEN IS GARBAGE GENERATED

- go run -gcflags -m main.go

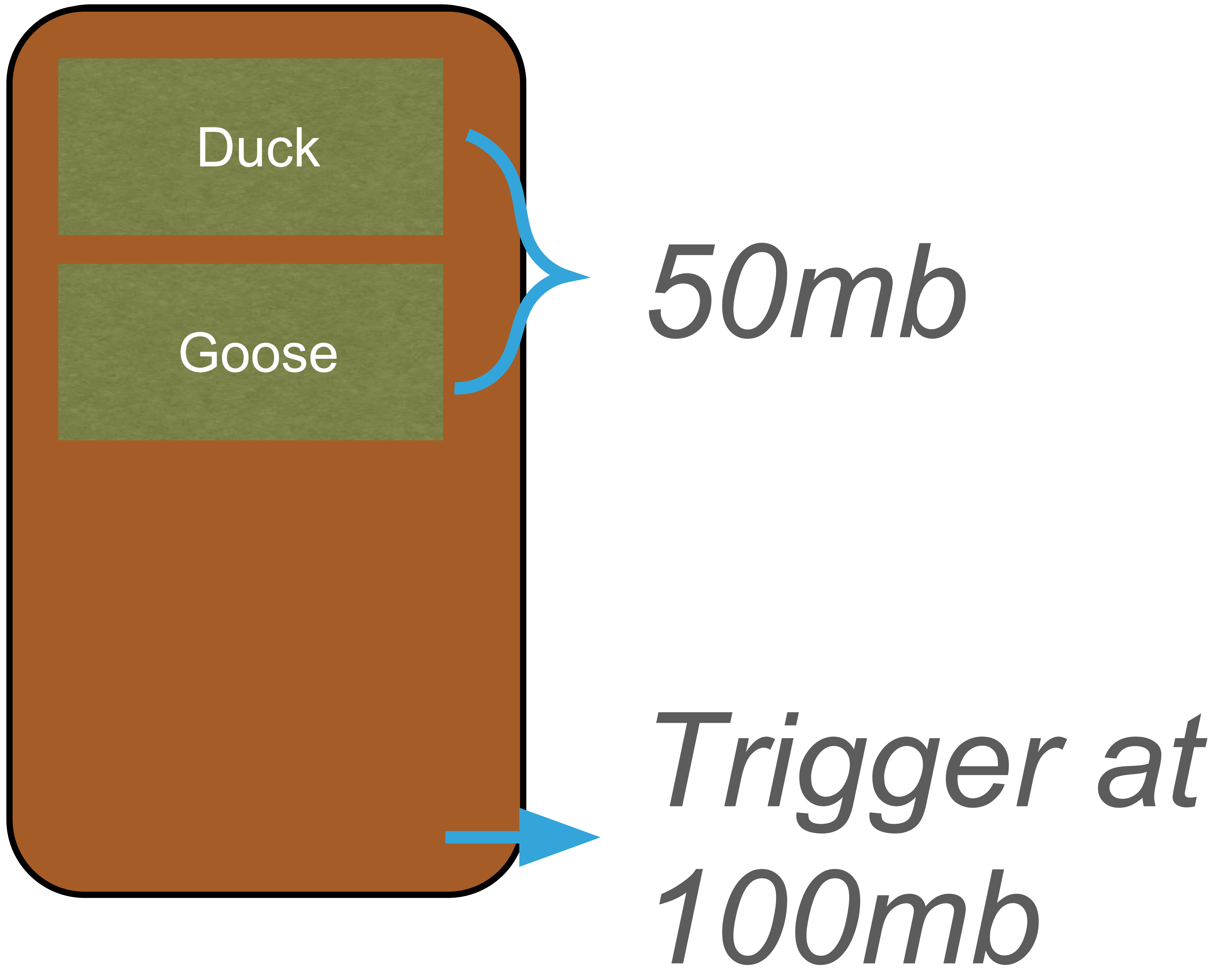
```
# command-line-arguments
./main.go:16:6: cannot inline NewDuck: marked
go:noinline
./main.go:9:6: cannot inline main: non-leaf function
./main.go:17:15: &Duck literal escapes to heap
./main.go:17:15: from ~r0 (return) at ./main.go:17:2
```

WHEN IS GARBAGE GENERATED

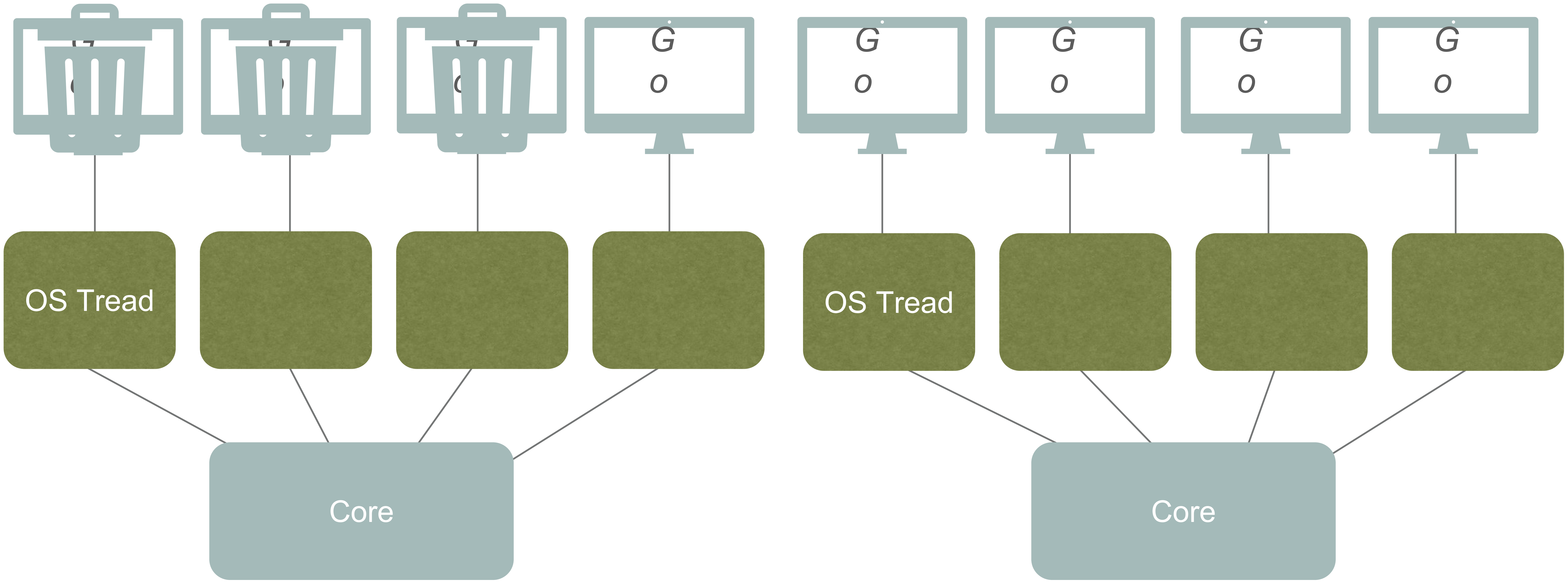
```
func main() {
    var a Bird
    a = NewDuck()
     a.Tweet()
}


type Bird interface {
   Tweet()
}
```



TRIGGER THE GARBAGE COLLECTOR



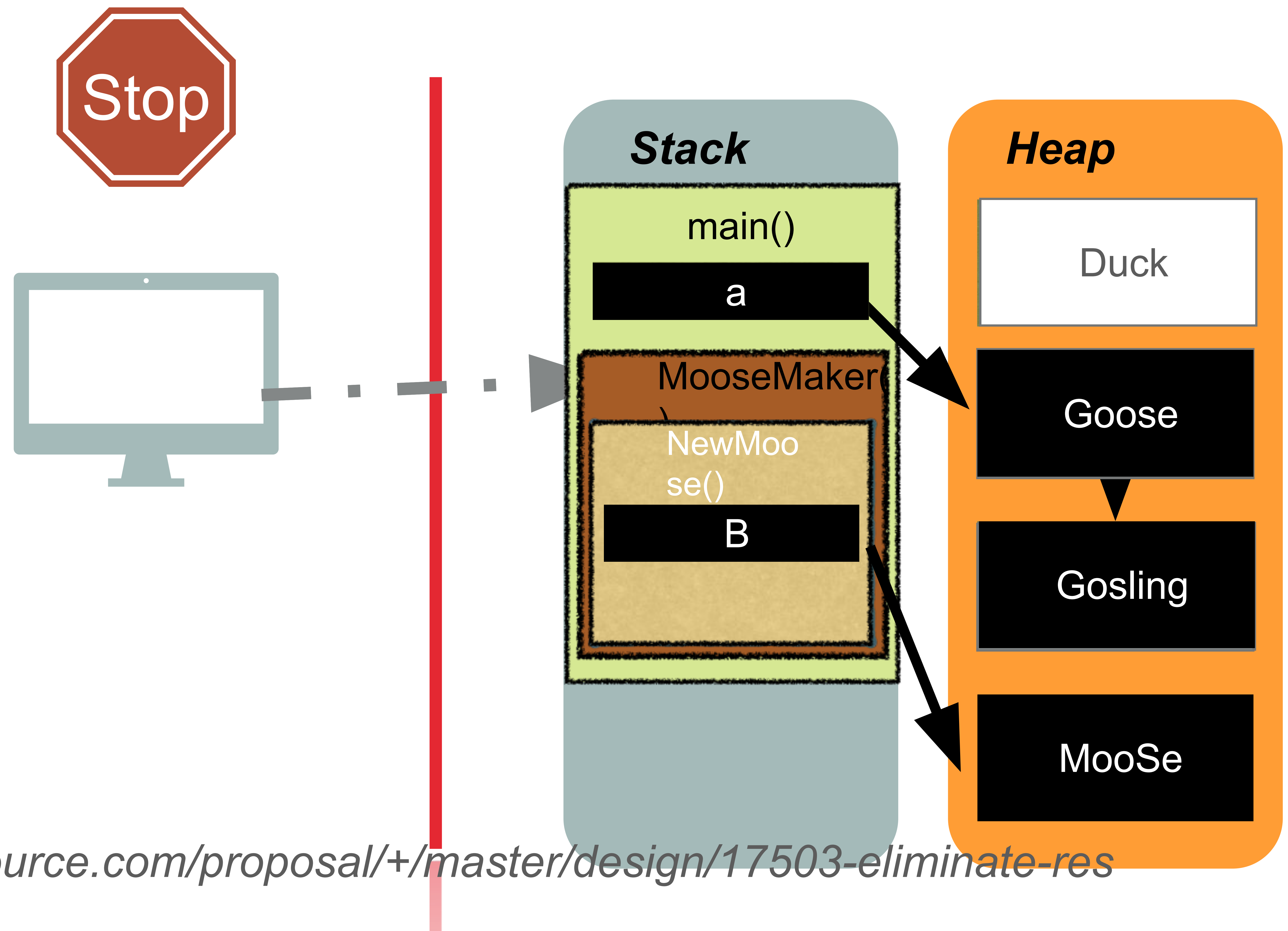
GC PACER



CONCURRENT, TRICOLOR MARK AND SWEEP

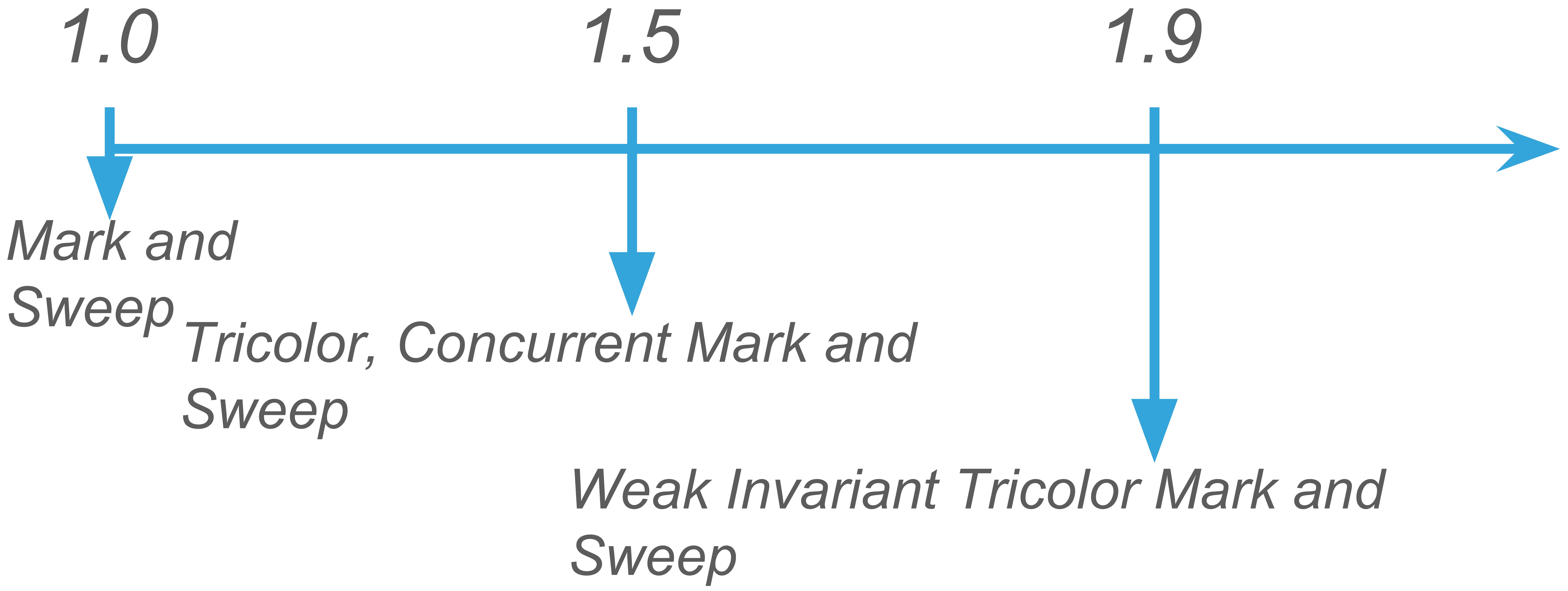
*NON-GENERATIONAL, NON-MOVING

CONCURRENT MARK AND SWEEP



https://go.googlesource.com/proposal/+/master/design/17503-eliminate-rescan.md

TIMELINE


1.0
1.5
1.9
Mark and Sweep
Tricolor, Concurrent Mark and Sweep
Weak Invariant Tricolor Mark and Sweep

TRADEOFFS

- GC throughput
- Program throughput
- Pause times (Latency)

MEASUREMENT

Performance

BENCHMARKS

- Be wary of toy Garbage Collection tests
  - Don’t behave the same a “real” programs
  - locality effects

- Optimum dataset for any algorithm

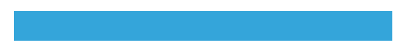
TOOLS

- pprof
- gctrace=1
- sync.Pool

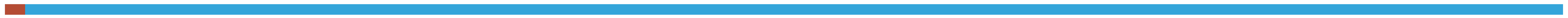
- https://github.com/golang/go/wiki/Performance
- https://golang.org/pkg/runtime/
- https://www.ardanlabs.com/blog/2017/06/language-mechanics-on-memory-profiling.html

TIME

Stack Allocation Alone - 10 ms



Original - 300 ms



TIME

GC STW- 100 microseconds

Humans process visual stimuli - 13 ms

Speed of light from Tel Aviv to San Francisco - 40ms

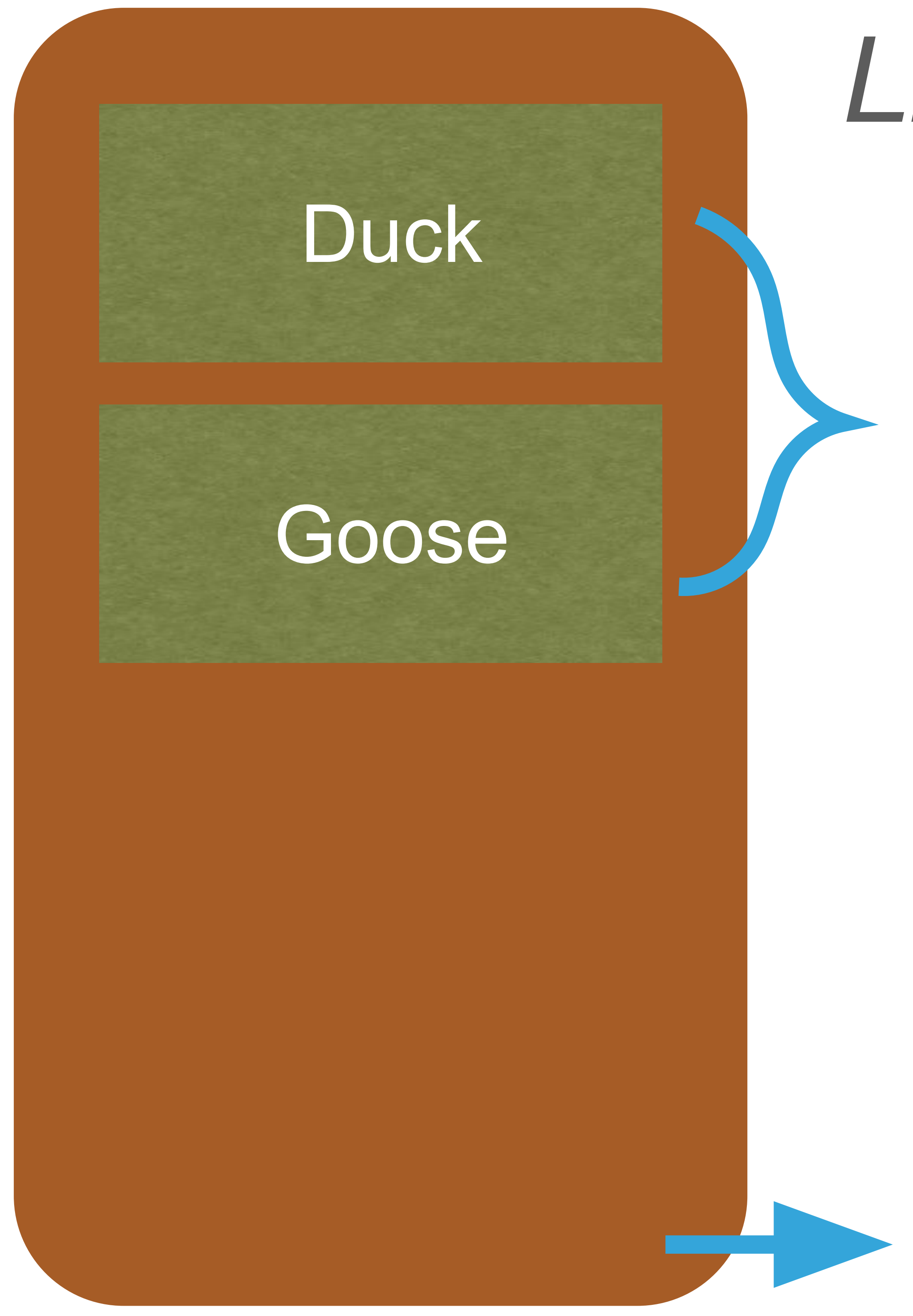
GARBAGE COLLECTOR KNOBS

Tweak the GC

GCPERCENT

- GOGC=100 go run foo.go

*Live data + (GCPercent/100 * Live data)*

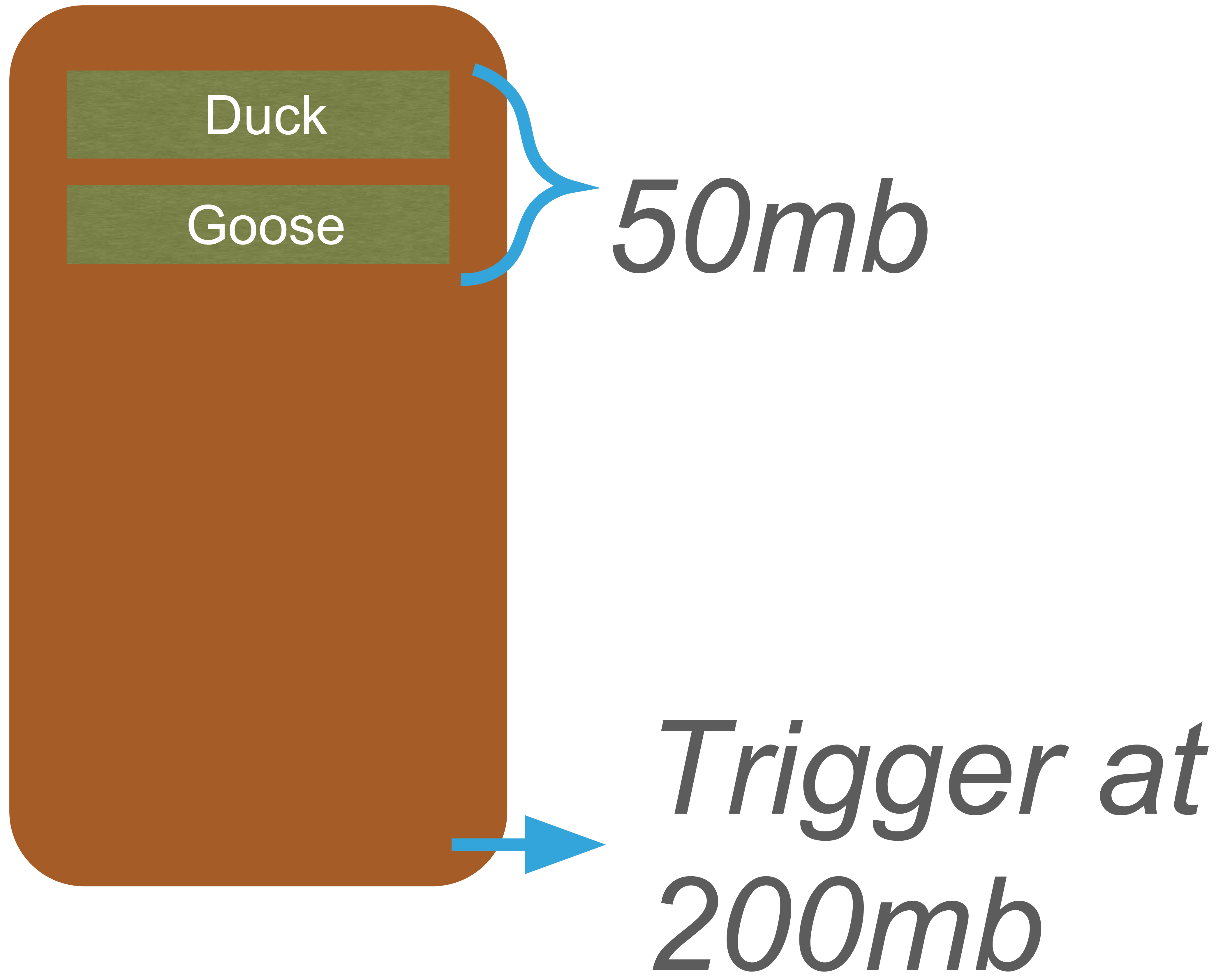


50mb

Trigger at 100mb

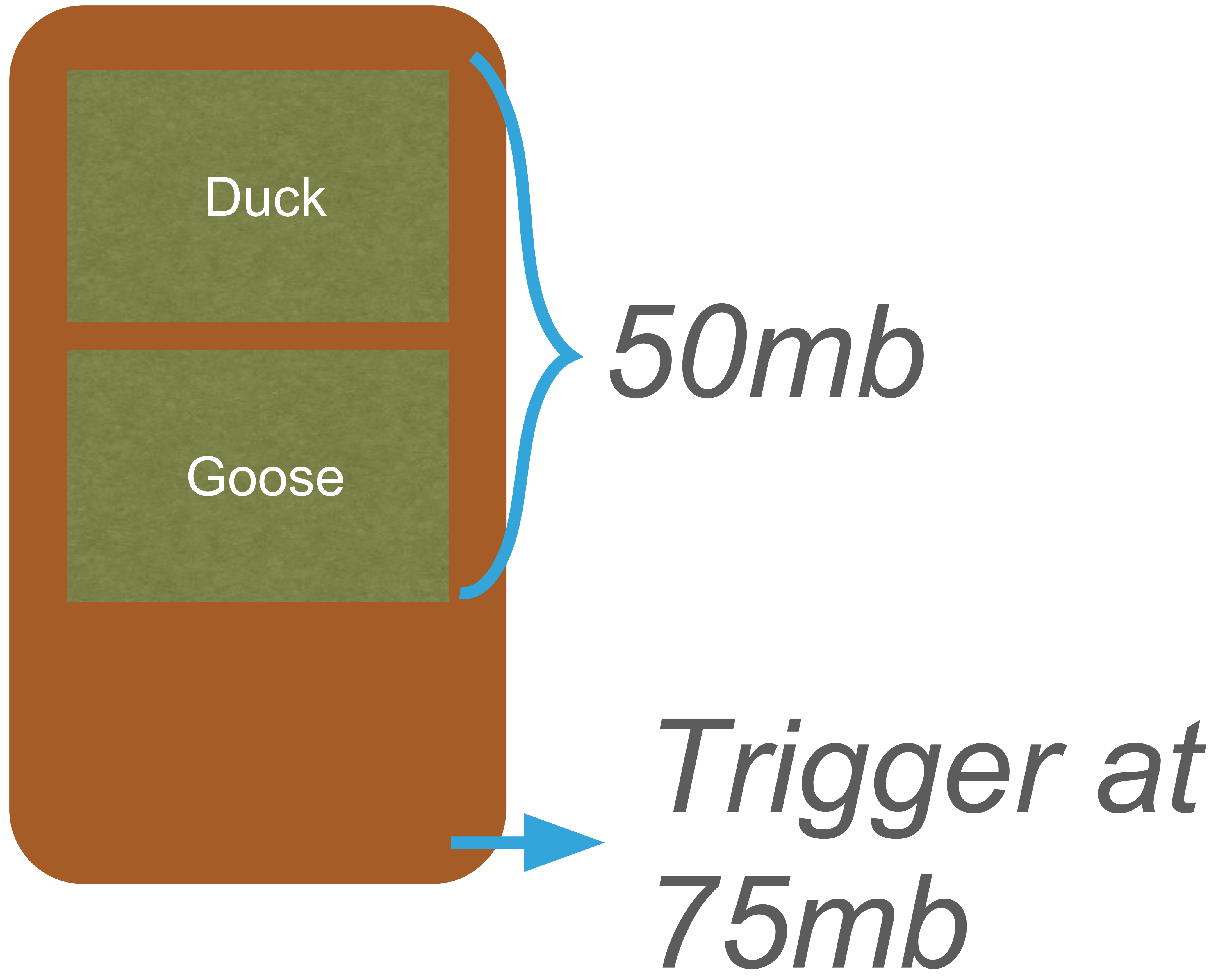
GCPERCENT

- GOGC=300 go run foo.go



GCPERCENT

- GOGC=50 go run foo.go



MAX HEAP SIZE

- Coming Soon.. maybe
- RSS = heap + stacks + globals
- Similar behavior with tuning GCPercent

https://github.com/golang/go/issues/16843

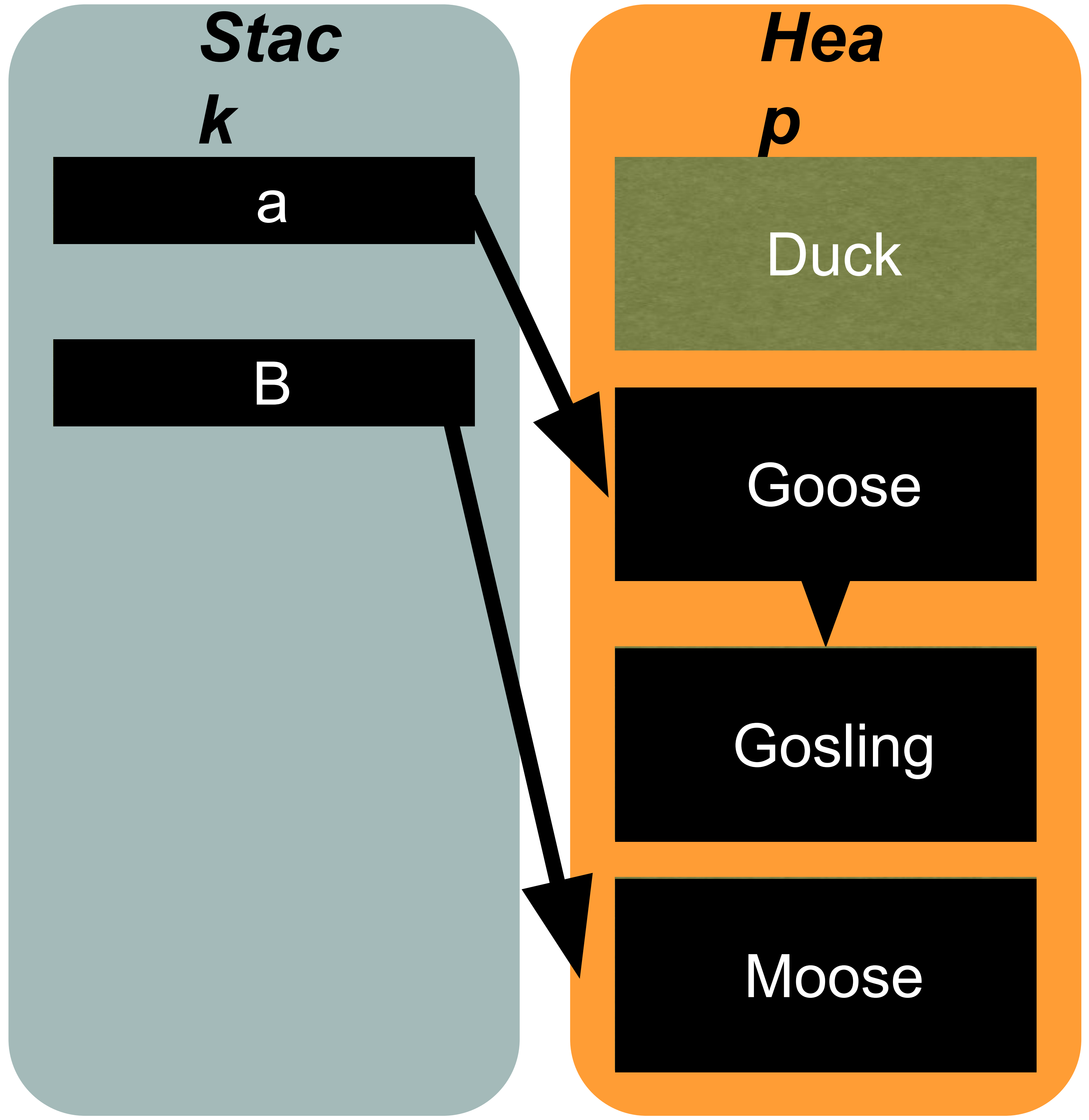
THANK YOU!

REFERENCES

- The Garbage Collection Handbook: The Art of Automatic Memory Management
- Rick Hudson: https://www.youtube.com/watch?v=aiv1JOfMjm0&t=1s and https://blog.golang.org/ismmkeynote
- Bill Kennedy: https://www.ardanlabs.com/blog/2017/06/language-mechanics-on-memory-profiling.html
- Proposal on 1.9 algorithm: https://go.googlesource.com/proposal/+/master/design/17503-eliminate-rescan.md

MARK AND SWEEP COLLECTION

Shrink Stack?



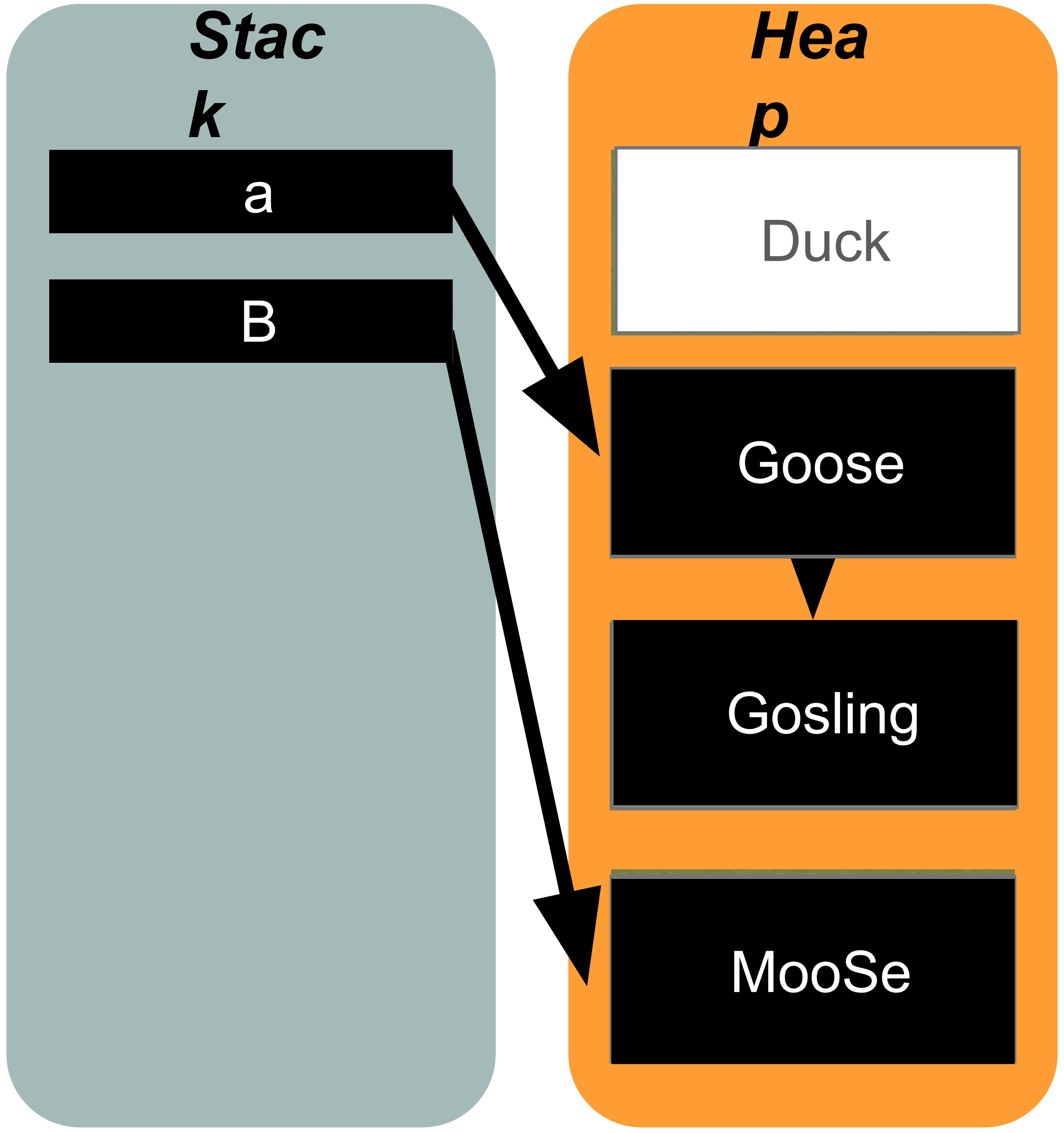
OTHER GARBAGE COLLECTOR ALGORITHMS

- Mark-compact
- Copying
- Reference counting

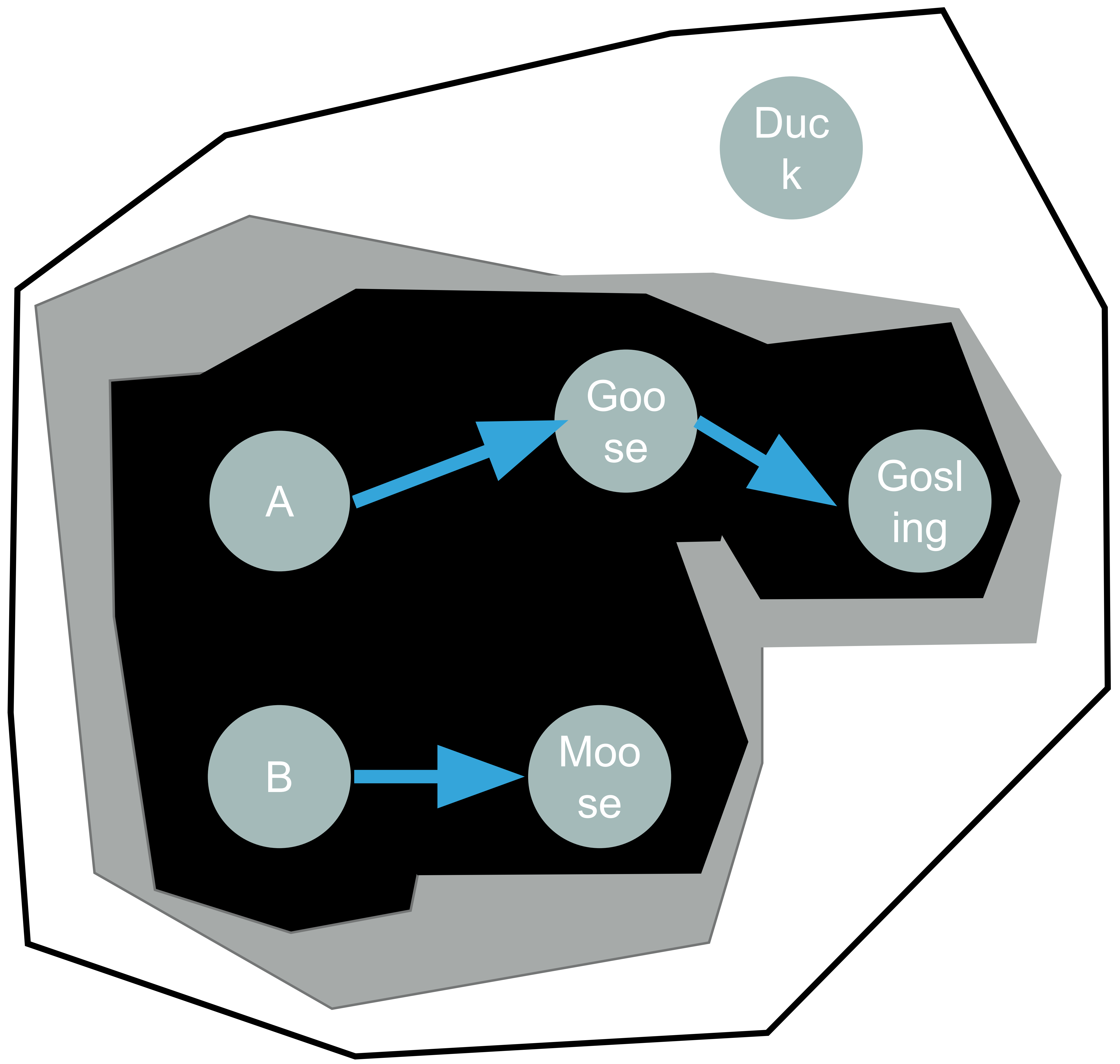
- Manual garbage collection

TRICOLOR MARK AND SWEEP

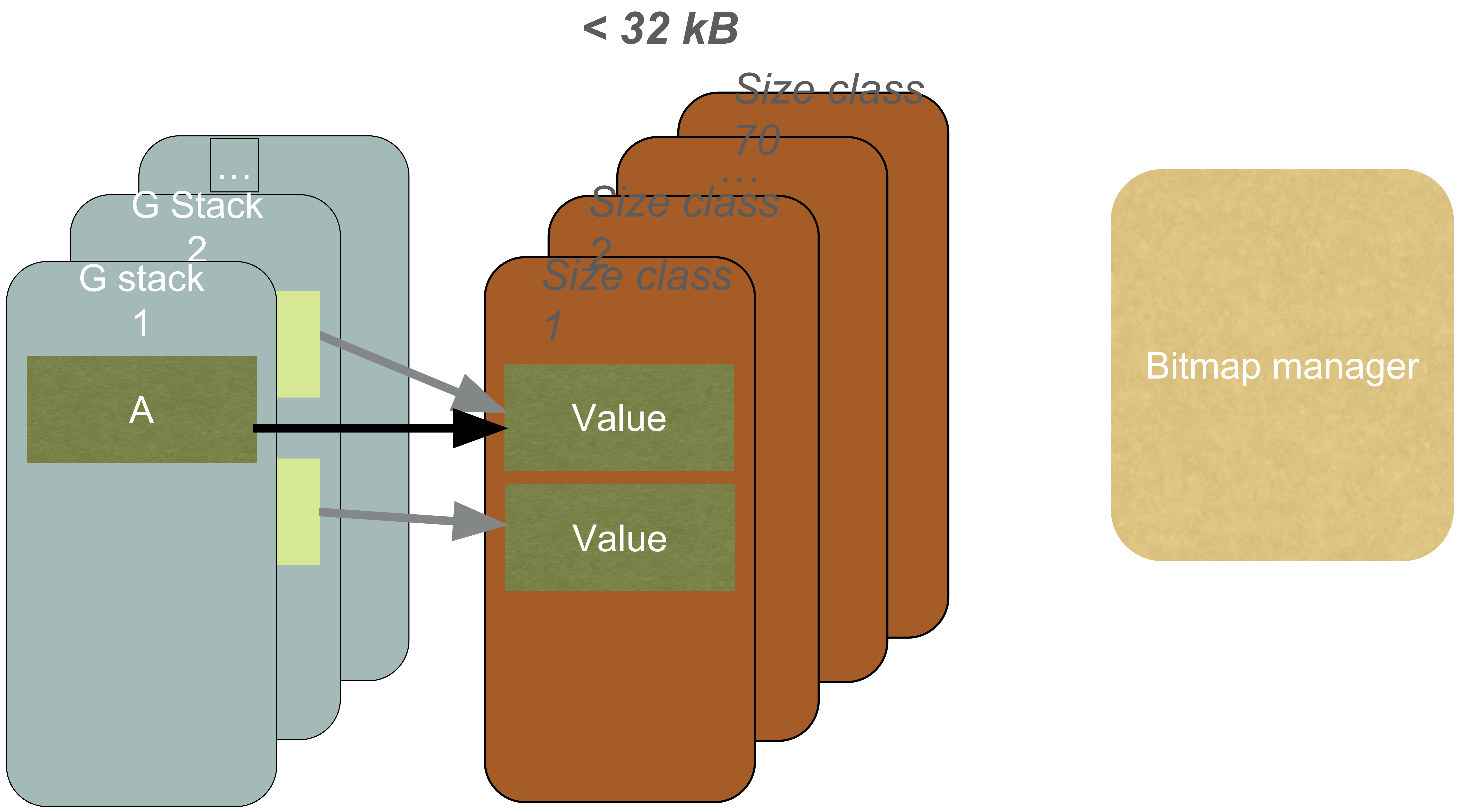
Strong Invariant: No black node can point to a white node



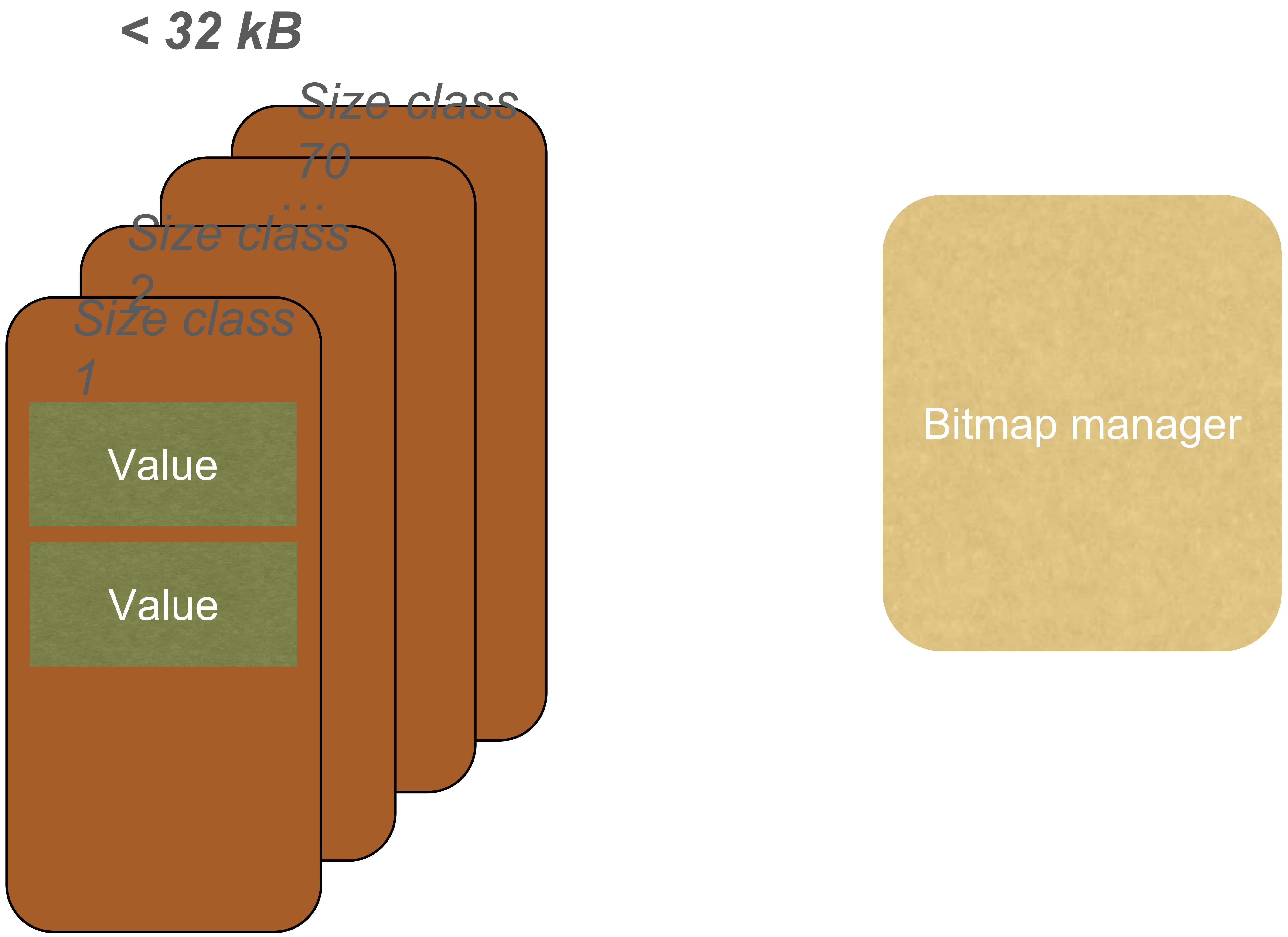
TRICOLOR MARK AND SWEEP



ALLOCATION


< 32 kB
Size class 70
…
Size class 2
Size class 1
…
G Stack 2
G stack 1
A
Value
Value
Bitmap manager

NON-MOVING


< 32 kB
Size class 70
...
Size class 2
Size class 1
Value
Value
Bitmap manager