Garbage Collection Goodies

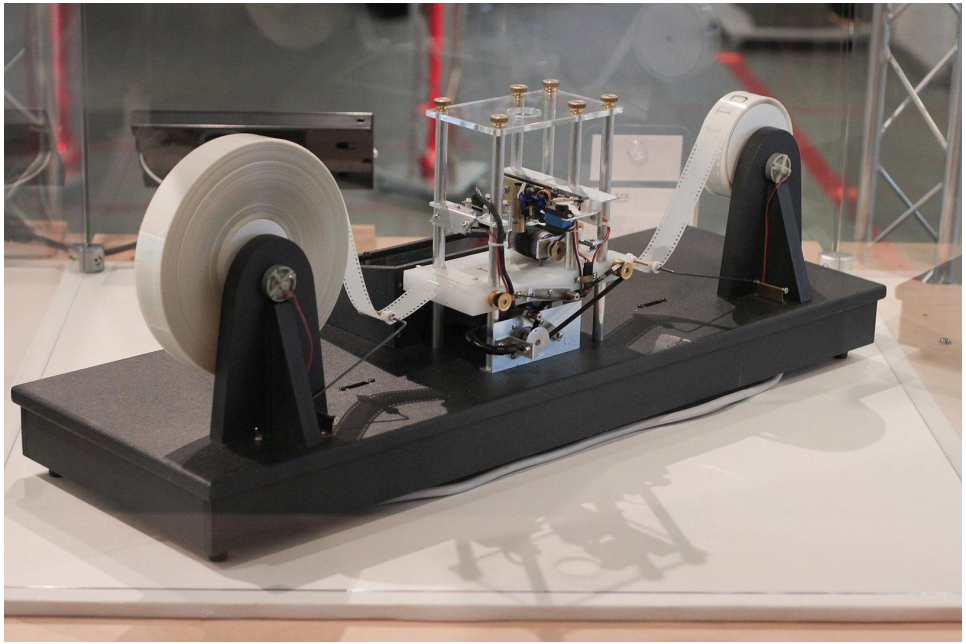
what are we doing with all this garbage anyway

In the beginning…

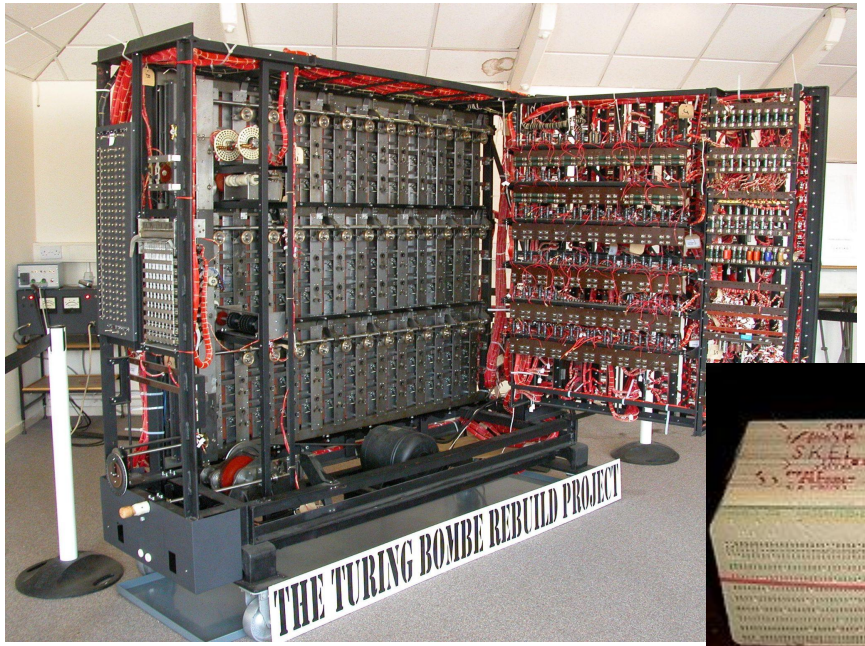
The Entscheidungsproblem

“We can know, we must know”

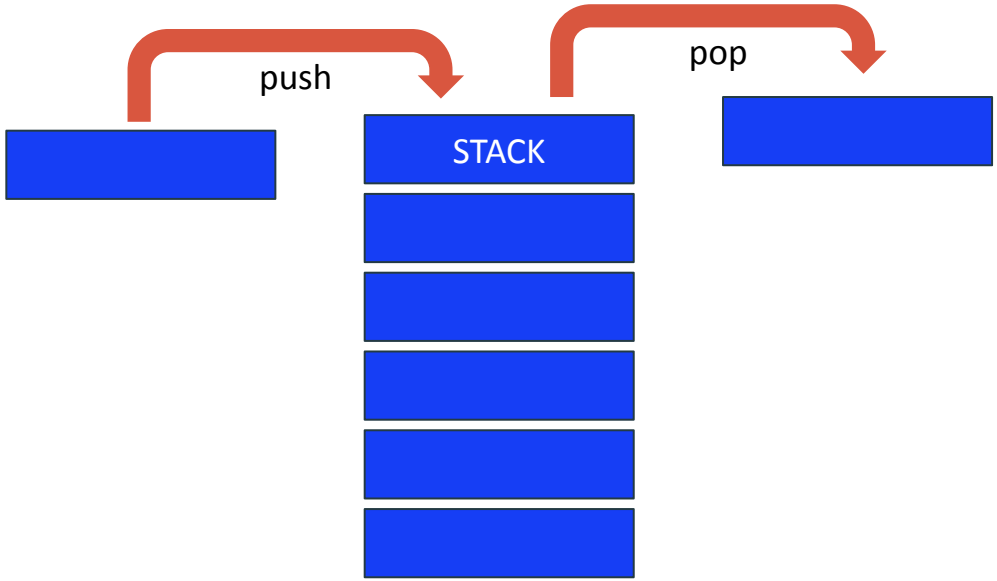
Church-Turing Thesis



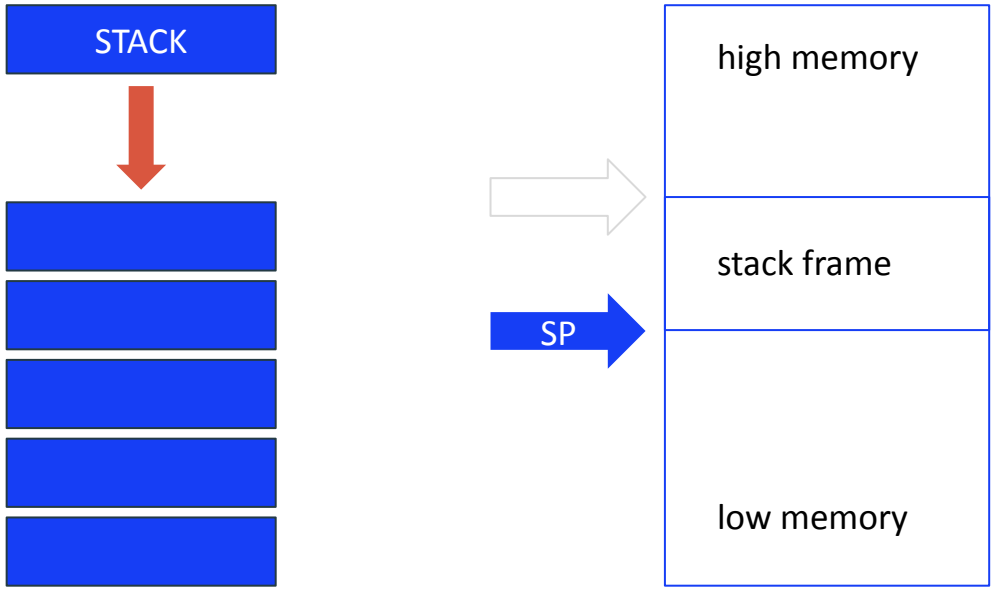
Early Computers



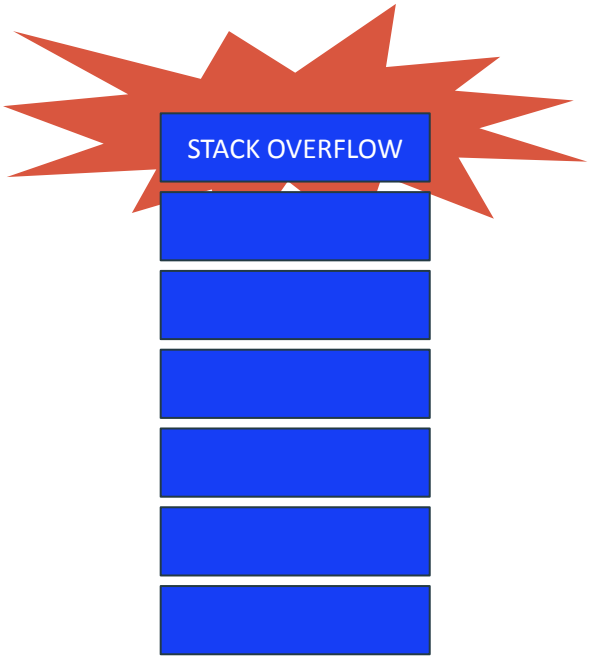
The Stack



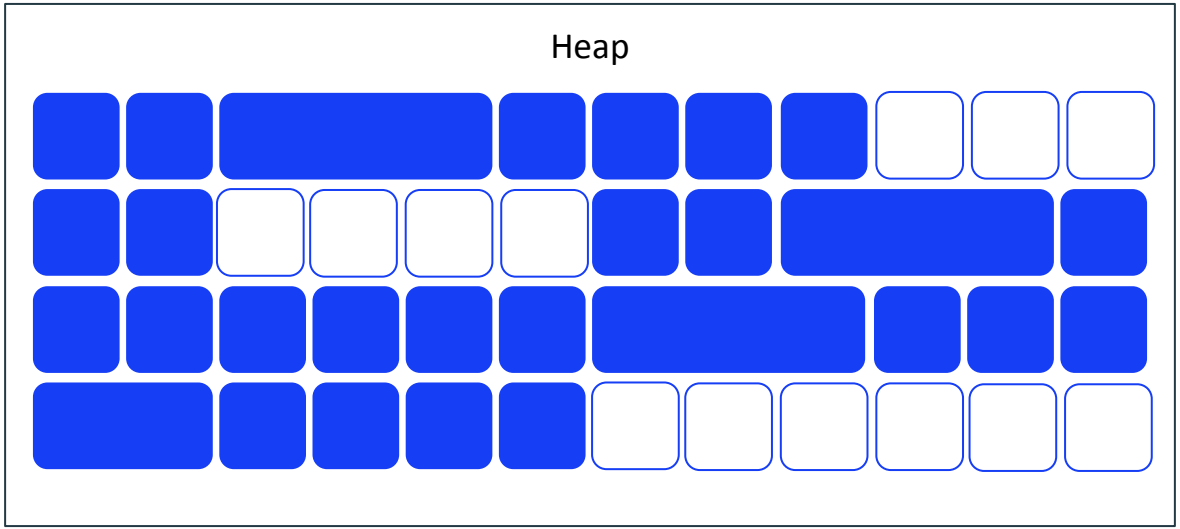
The Stack



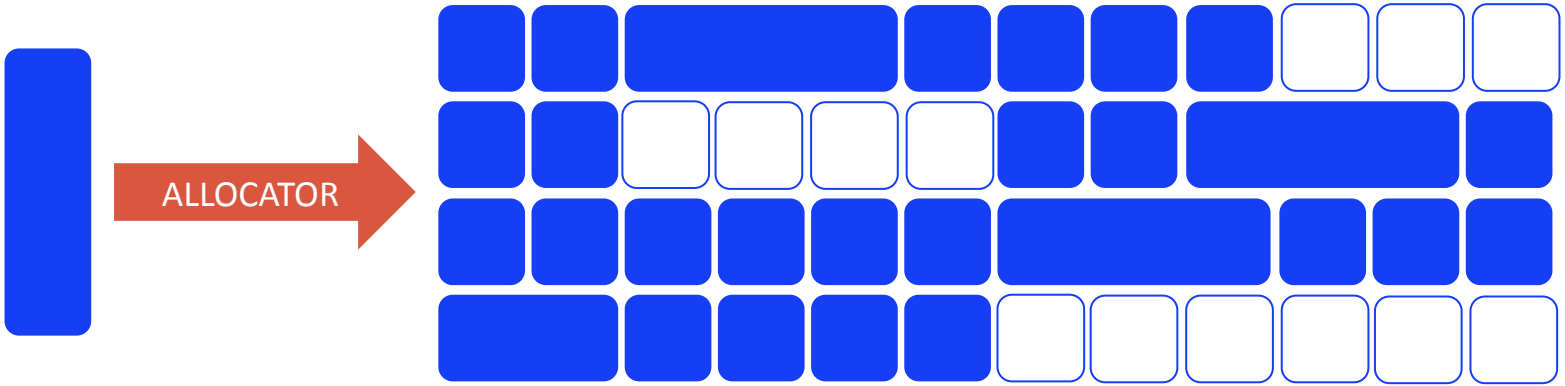
The Stack



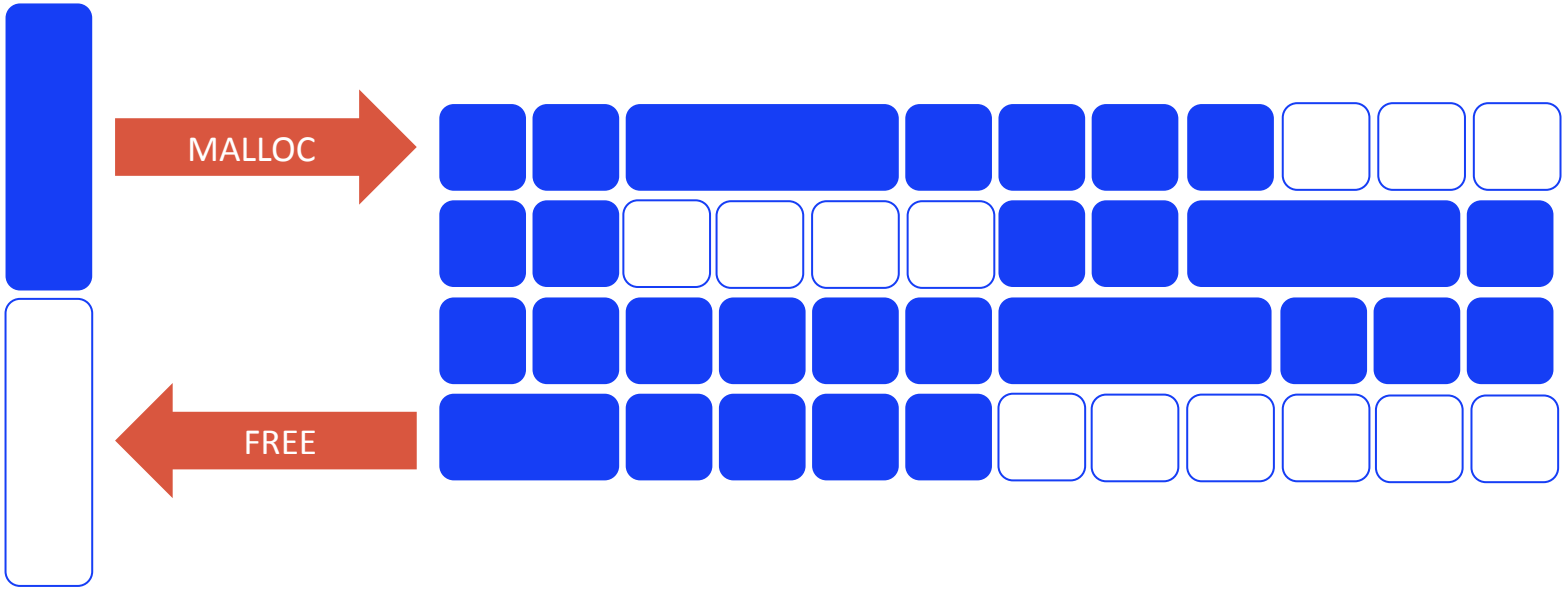
The Heap



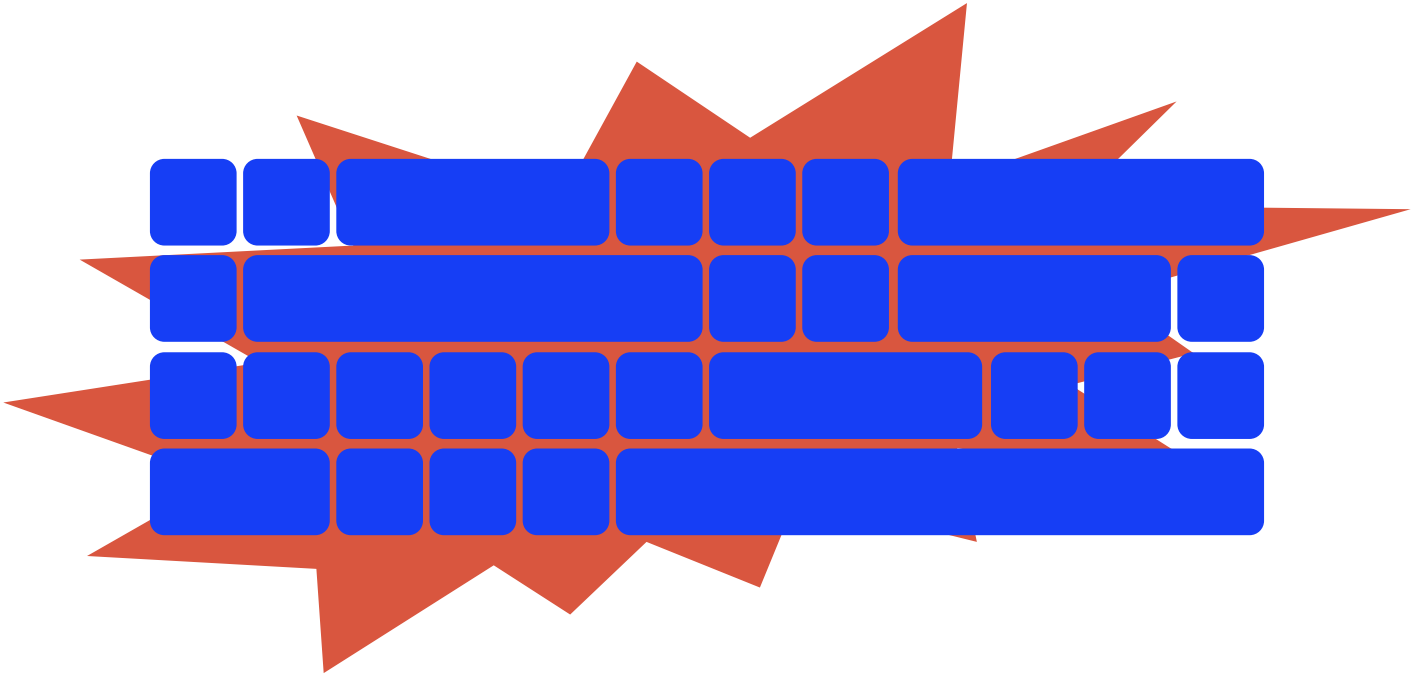
The Heap



The Heap



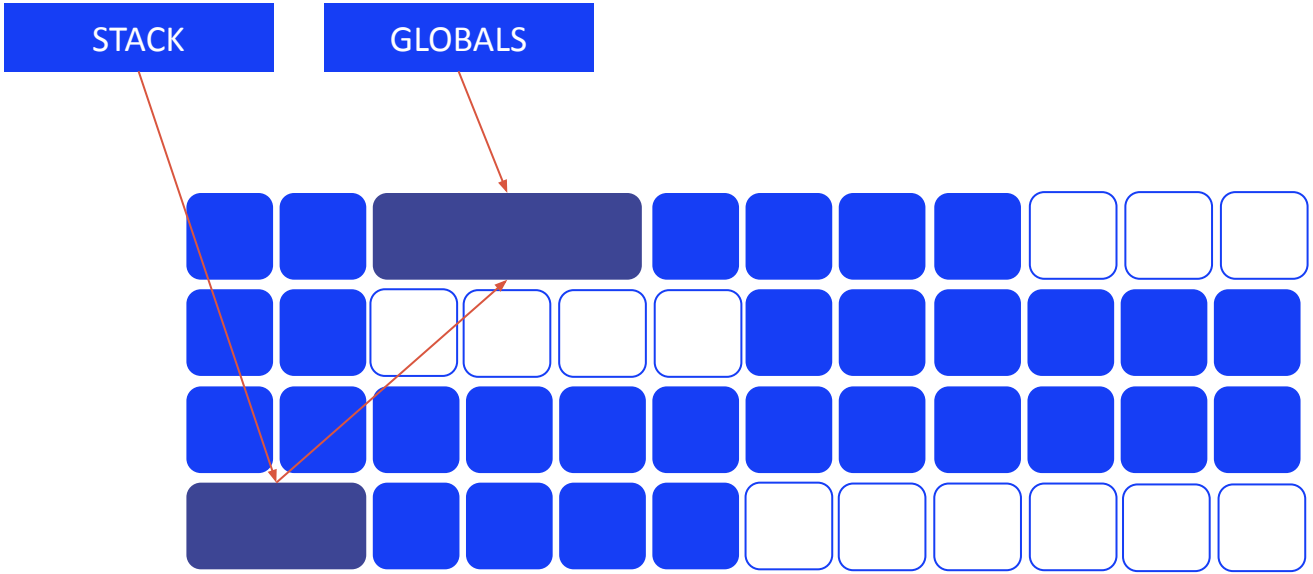
The Heap



Garbage Collection

"The Goal of the Garbage Collection provides the illusion that a language runtime has infinite memory."

Tracing



Scheduling

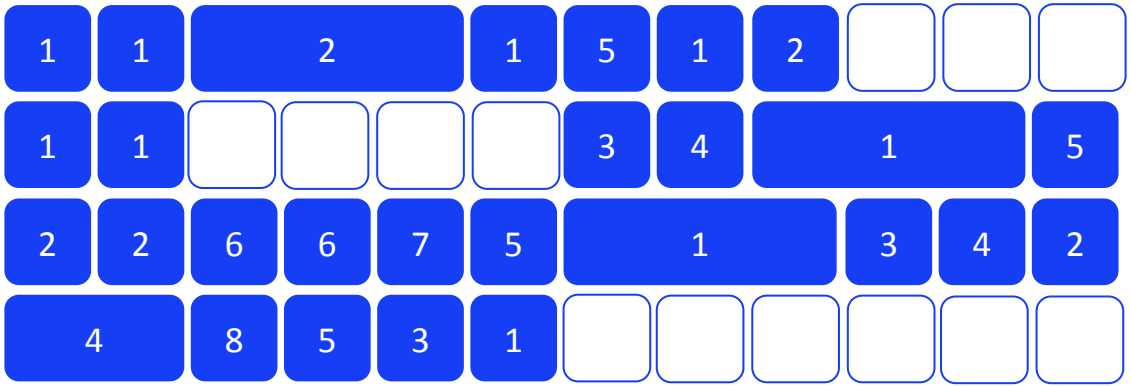
How often do we run our garbage collector?

Escape Analysis

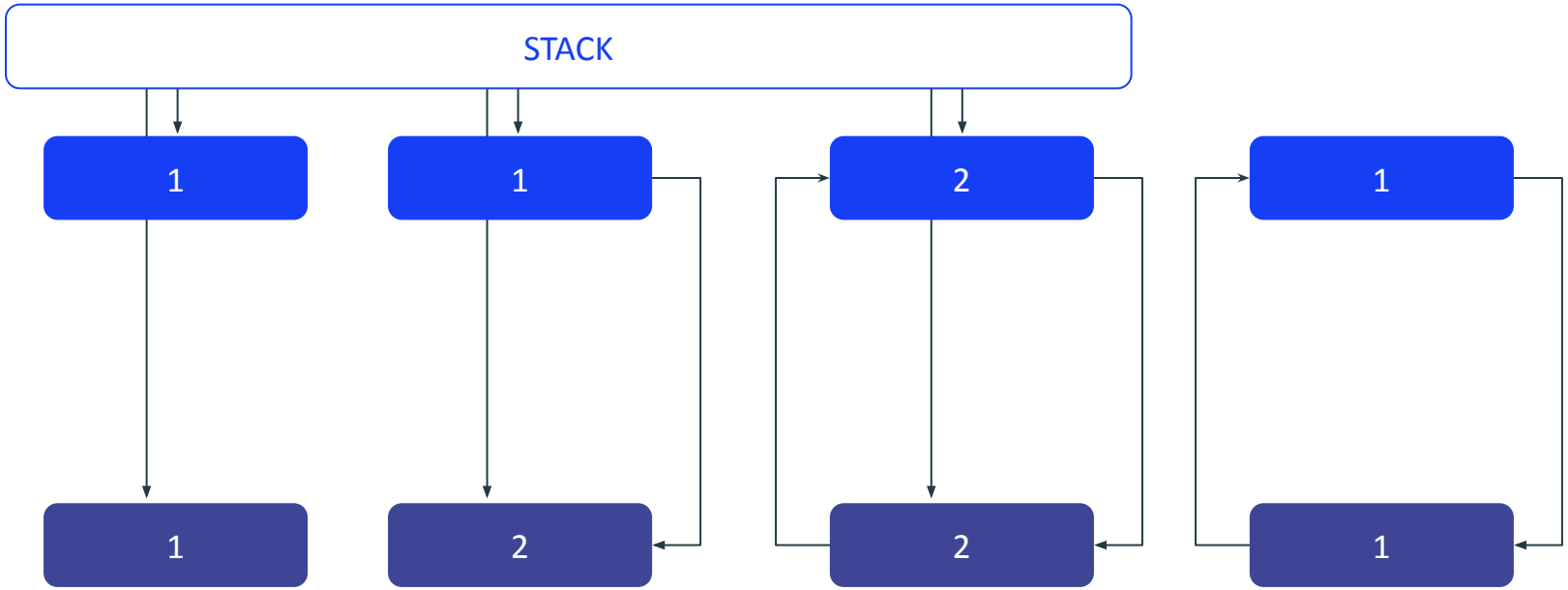
```go
package main

func main() {
    m := *heap()
    m++
}

//go:noinline
func heap() *int {
    i := 42
    return &i
}
```

```
brandon@brandon-OMEN-by-HP-Laptop-15z-en100:~/src/garbage_collectors$ go build -gcflags "-m -m" .
# github.com/bjatkin/garbage_collectors
./main.go:9:6: cannot inline heap: marked go:noinline
./main.go:3:6: can inline main with cost 67 as: func() { m := *heap(); m++ }
./main.go:10:2: i escapes to heap:
./main.go:10:2:   flow: ~r0 = &i:
./main.go:10:2:     from &i (address-of) at ./main.go:11:9
./main.go:10:2:     from return &i (return) at ./main.go:11:2
./main.go:10:2: moved to heap: i
```

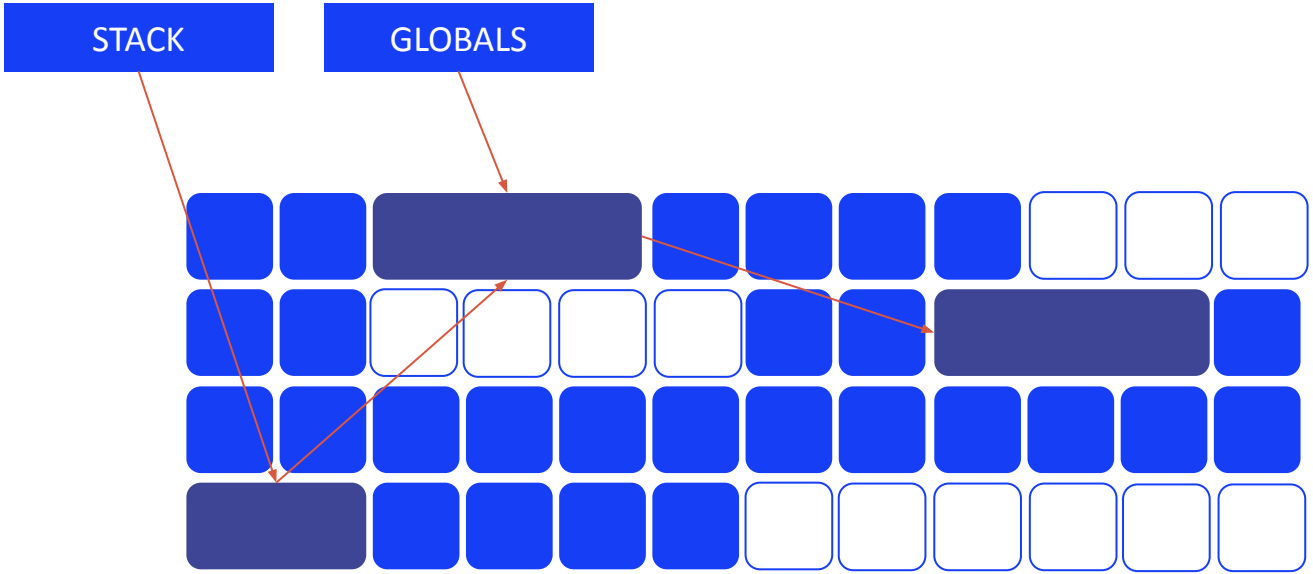
Reference Counting



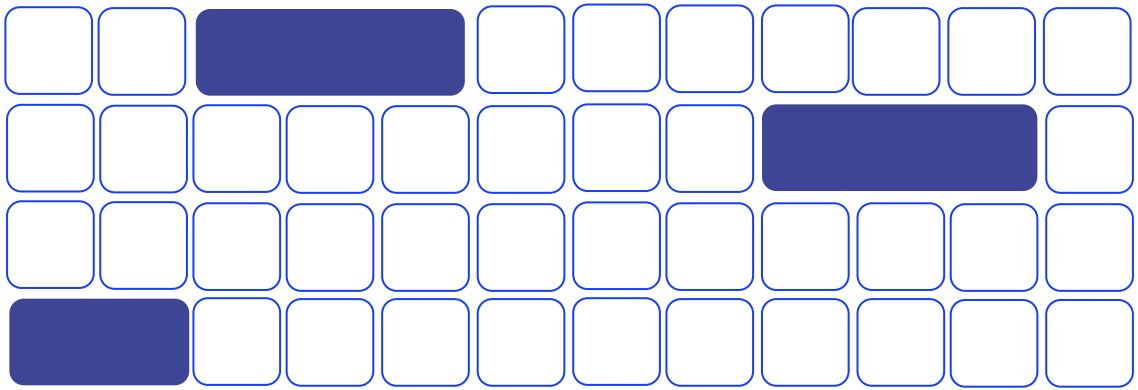
Reference Counting



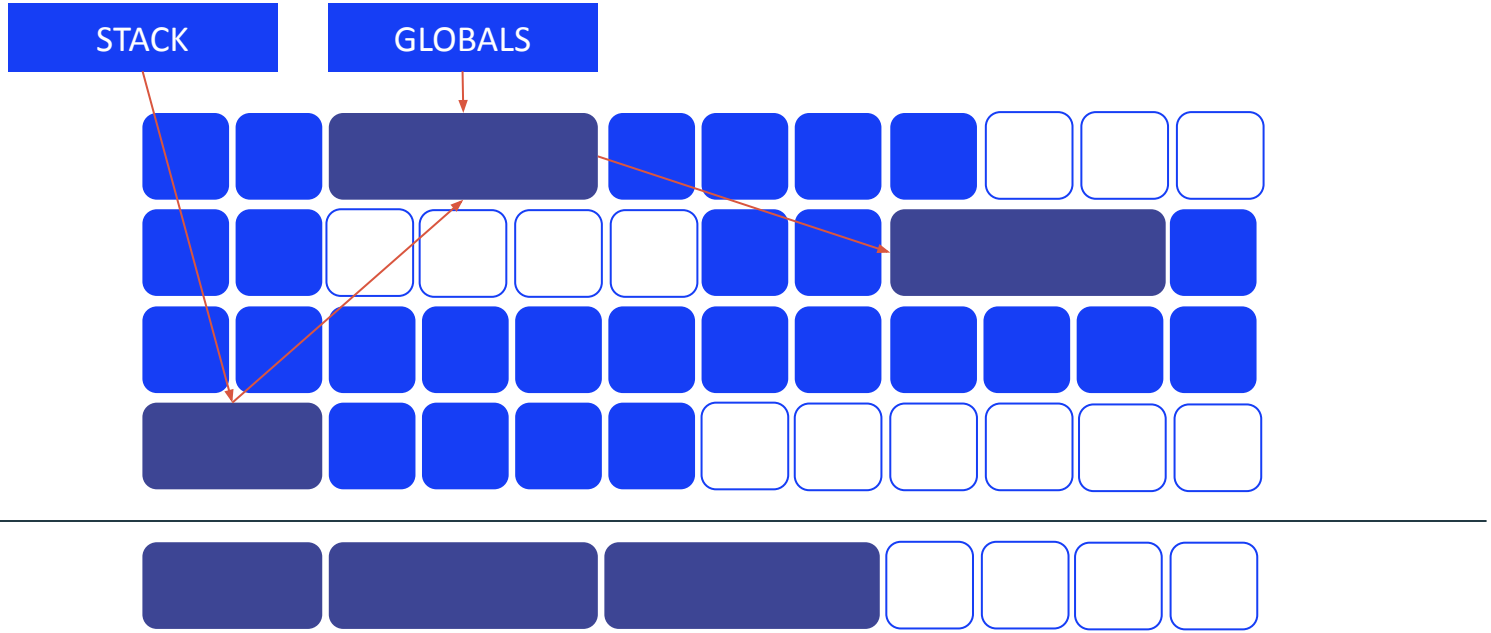
Mark and Sweep



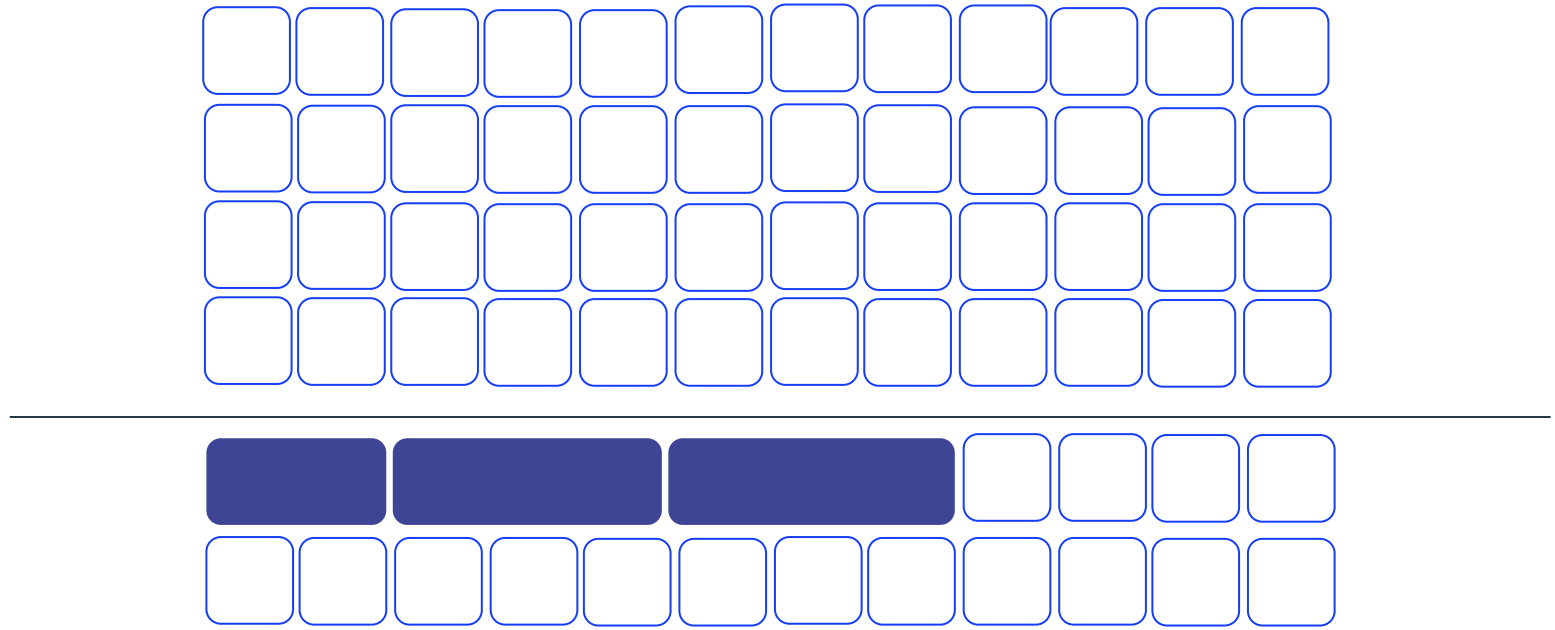
Mark and Sweep



Stop and Copy



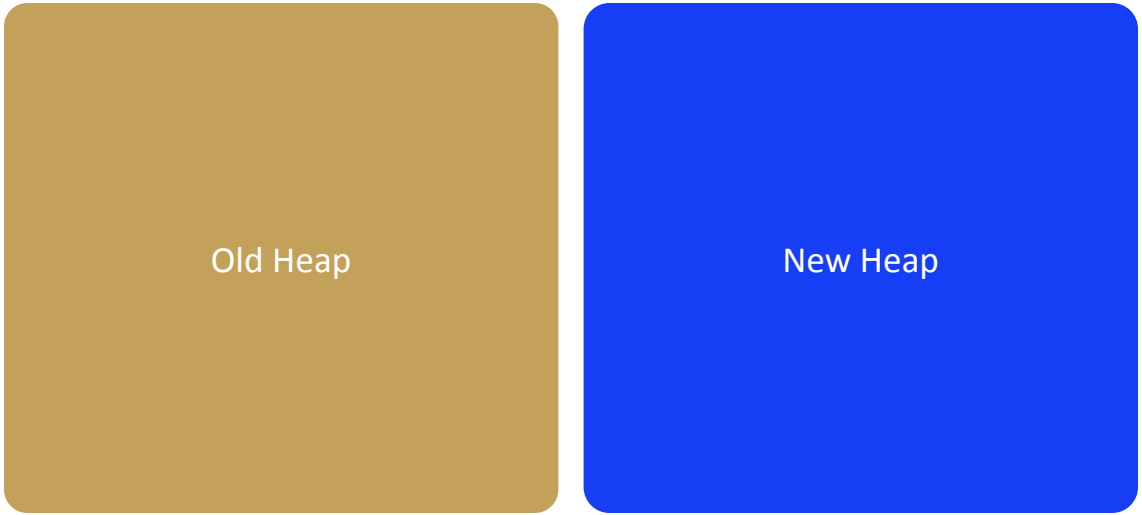
Stop and Copy



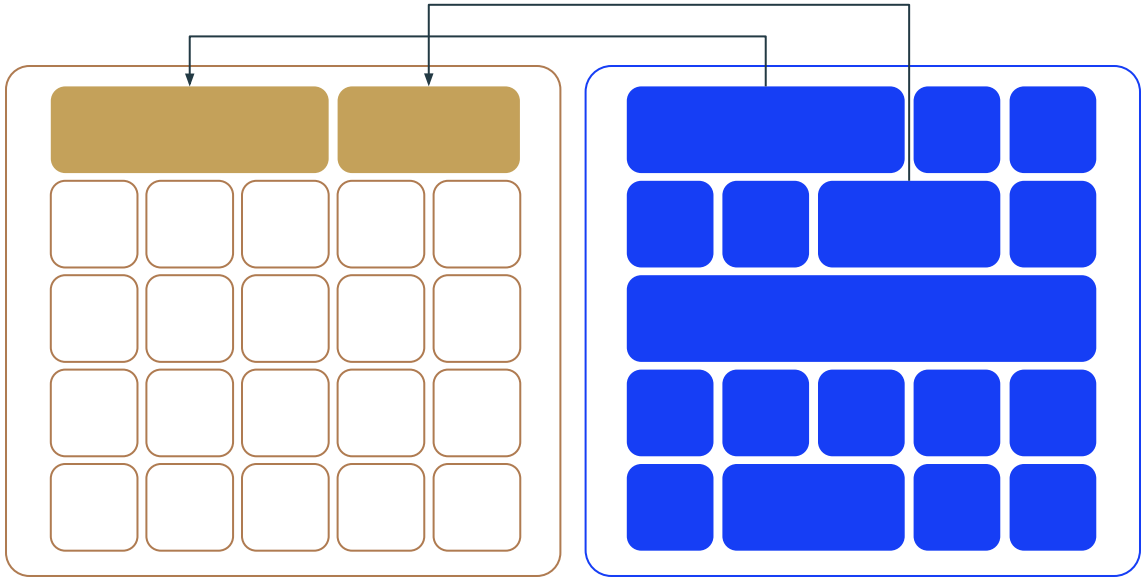
Generational Garbage Collection

“Most Object Die Young”

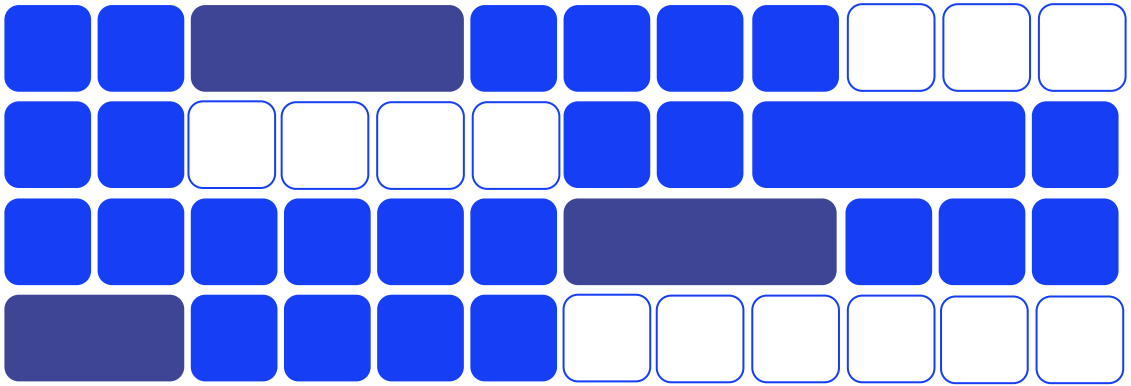
Generational Garbage Collection



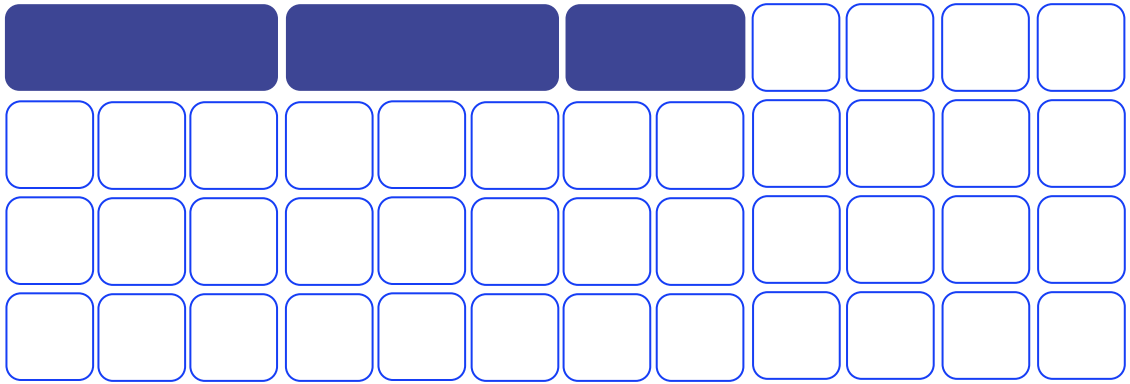
Generational Garbage Collection

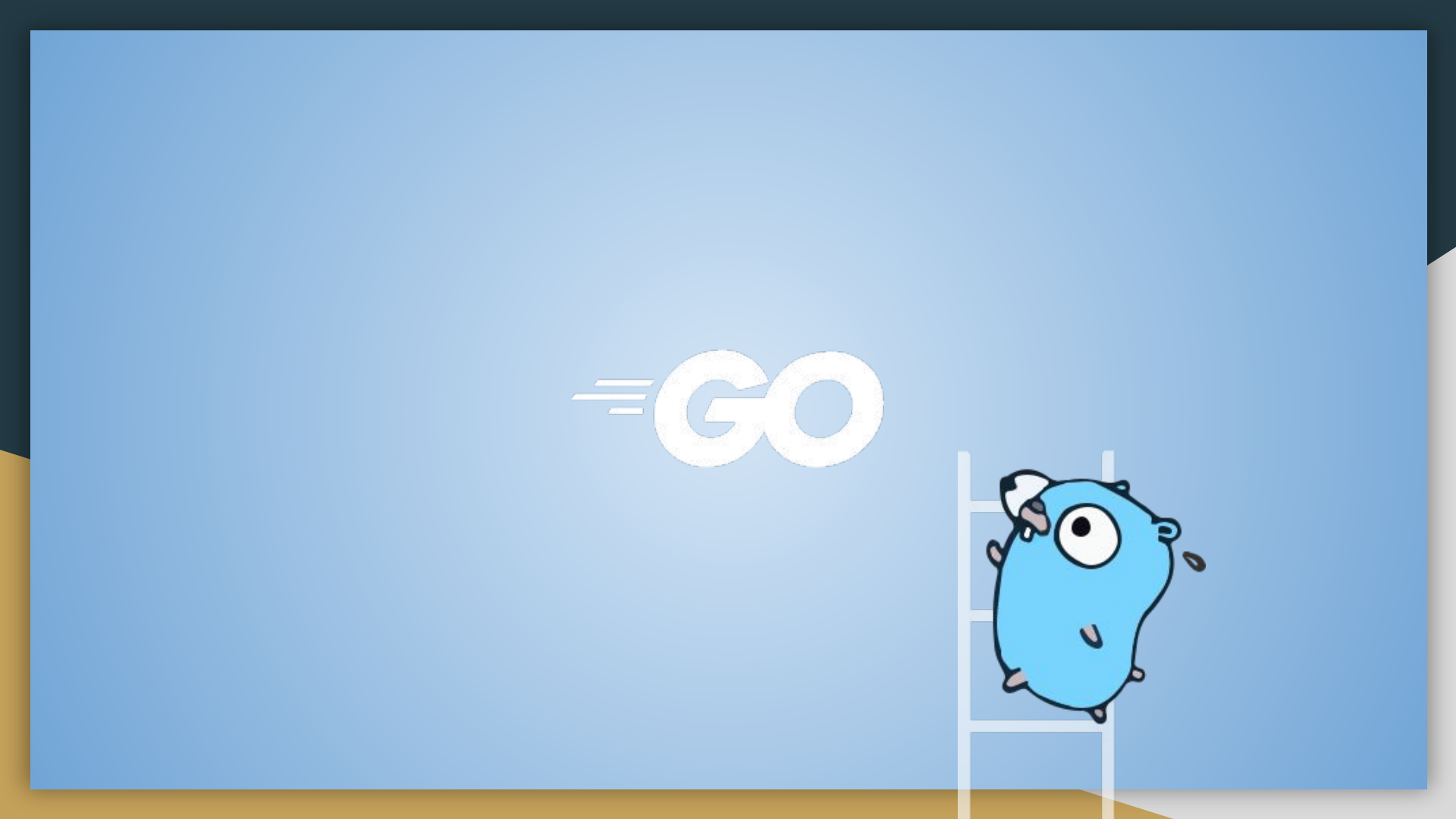


Mark Compact Garbage Collection

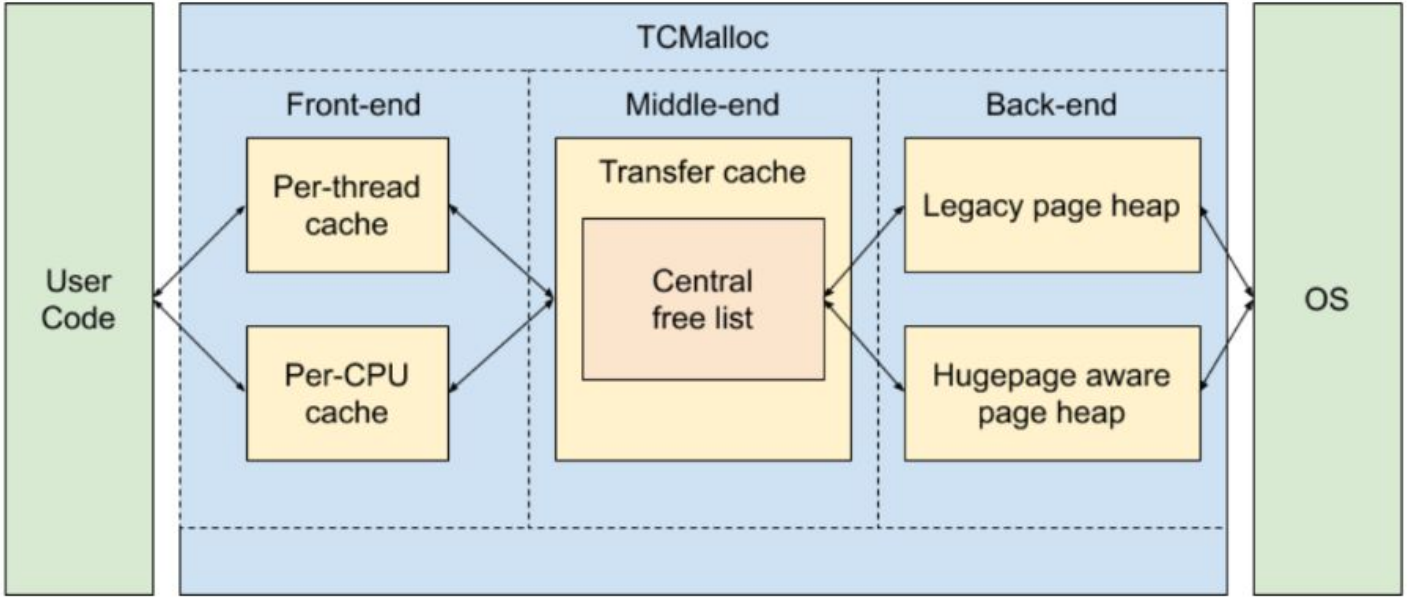


Mark Compact Garbage Collection




GO

TCMalloc


TCMalloc
Front-end
Middle-end
Back-end
User Code
Per-thread cache
Per-CPU cache
Transfer cache
Central free list
Legacy page heap
Hugepage aware page heap
OS

Garbage Collection In Go

“This omission is intentional and enables the use of radically different memory management techniques.“

A Guide to the Go Garbage Collector

Garbage Collection In Go

- Escape Analysis
- Tracing
- GC Scheduling (CPU usage vs Memory Usage)

Memory vs CPU tradeoff

“doubling GOGC will double heap memory overheads and roughly halve GC CPU cost”

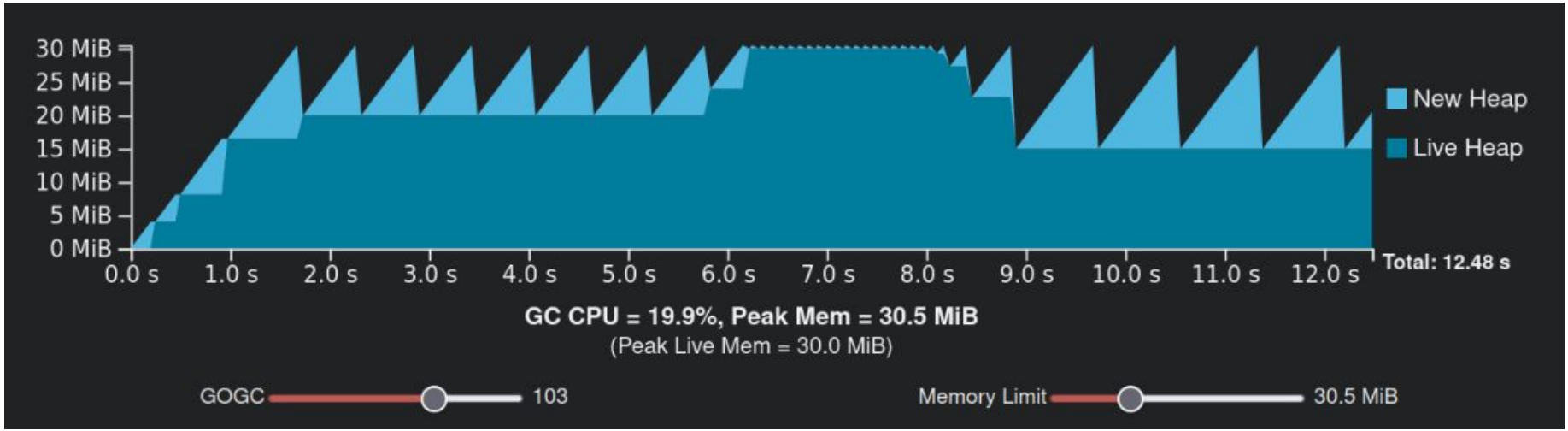
Memory vs CPU tradeoff

https://tip.golang.org/doc/gc-guide

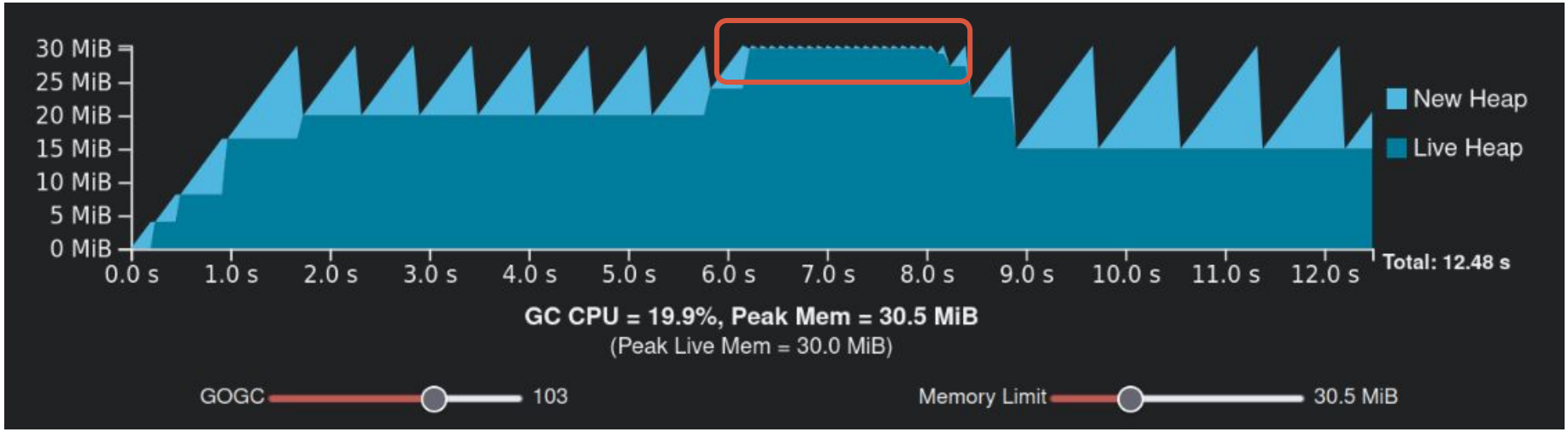
Optimizing The Go GC

- Make sure the GC is actually a limiting factor for your program (just like any other performance issue)
  - CPU profiles
  - Execution traces
  - GC traces
- Eliminate Heap Allocations
  - `$ go build -gcflags=-m=3 [package]`
- Configure your environment
  - `$GOGC` / `runtime.SetGCPercent()`
  - `$GOMEMLIMIT` / `runtime.SetMemoryLimit()`

GC Thrashing



GC Thrashing

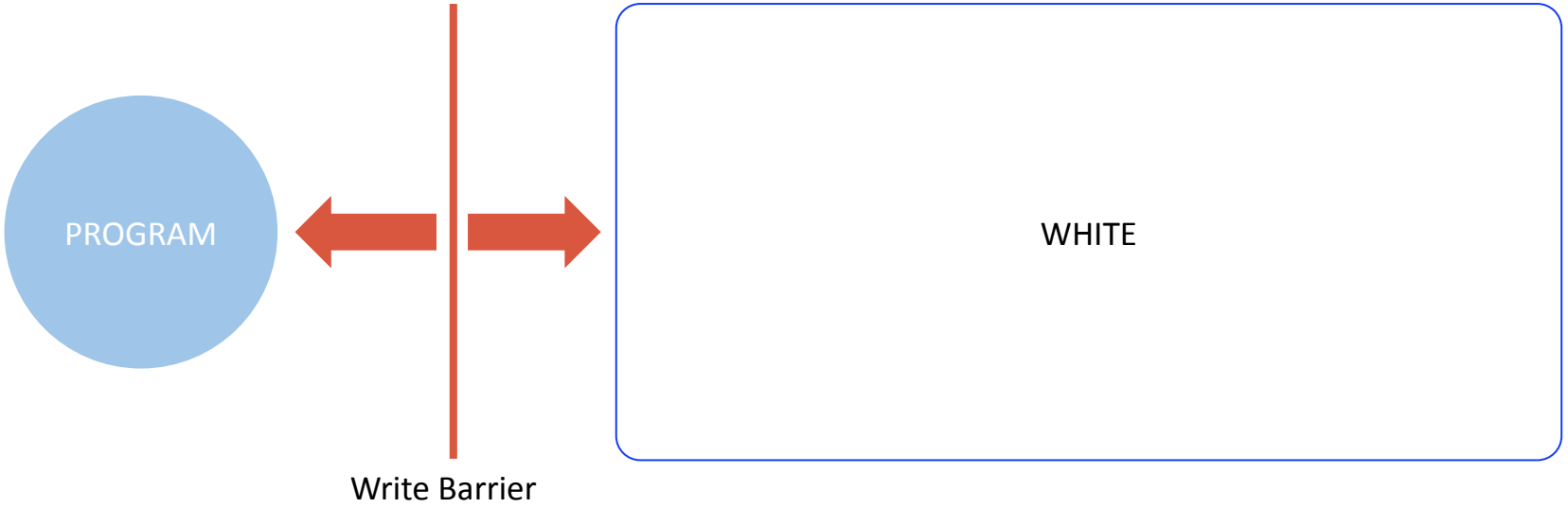


Latency Vs CPU usage

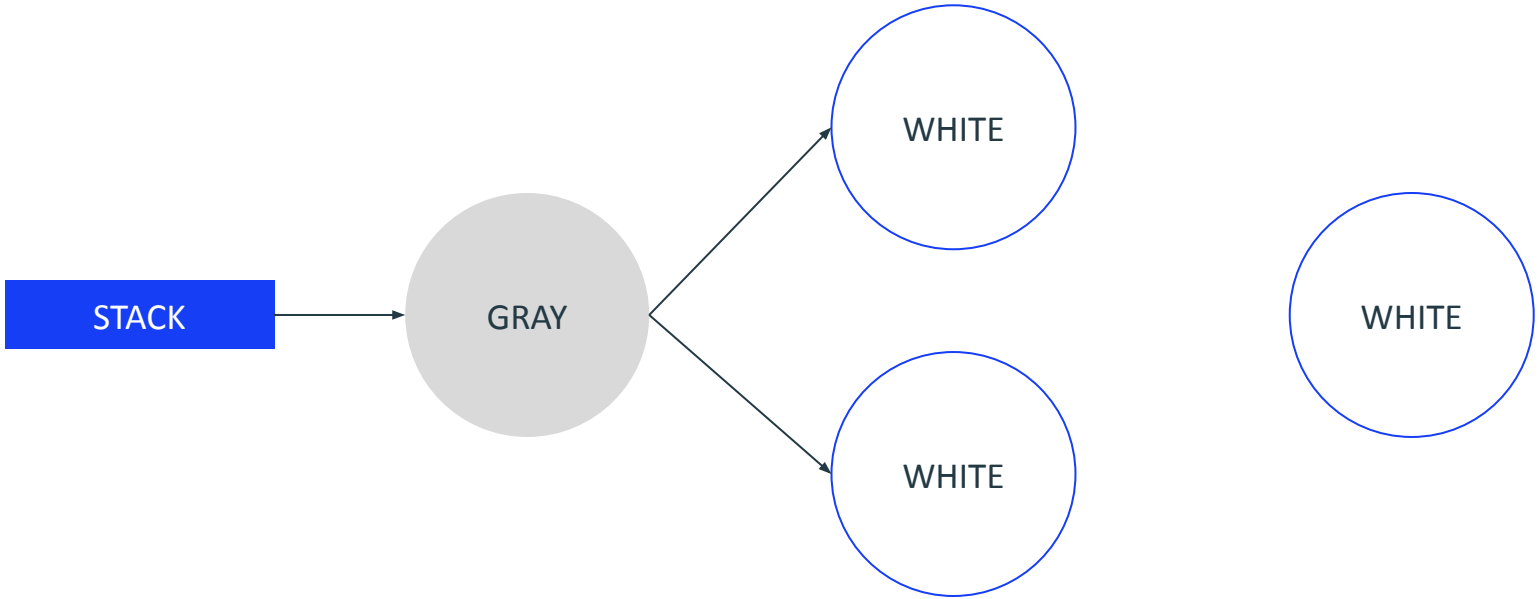
“reducing GC frequency may also lead to latency improvements”

Tricolor Concurrent Nonmoving Mark and Sweep

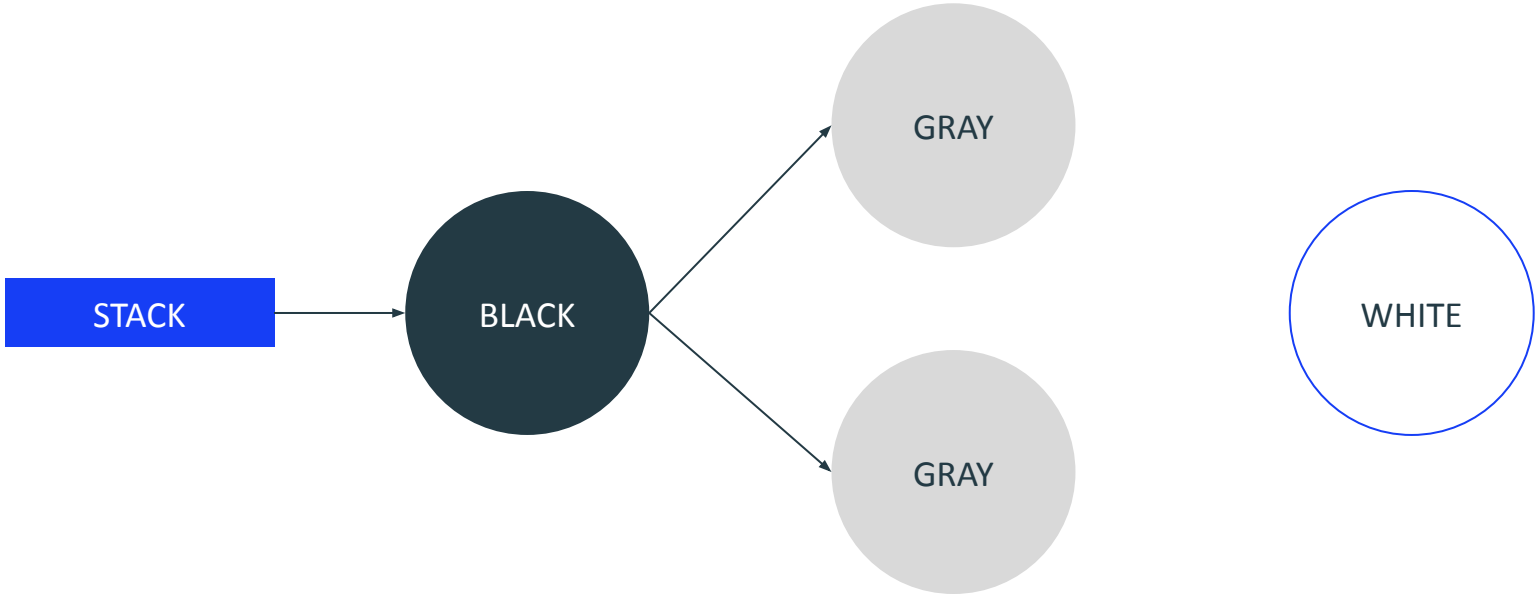
Tricolor Concurrent Nonmoving Mark and Sweep



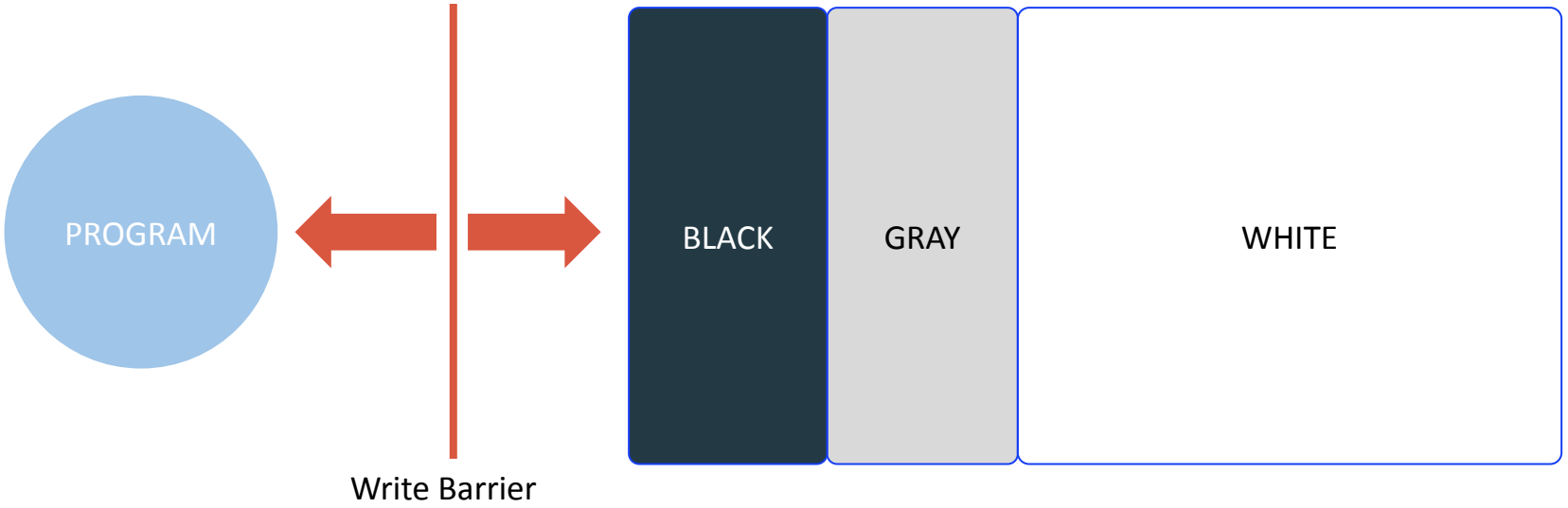
Tricolor Concurrent Nonmoving Mark and Sweep



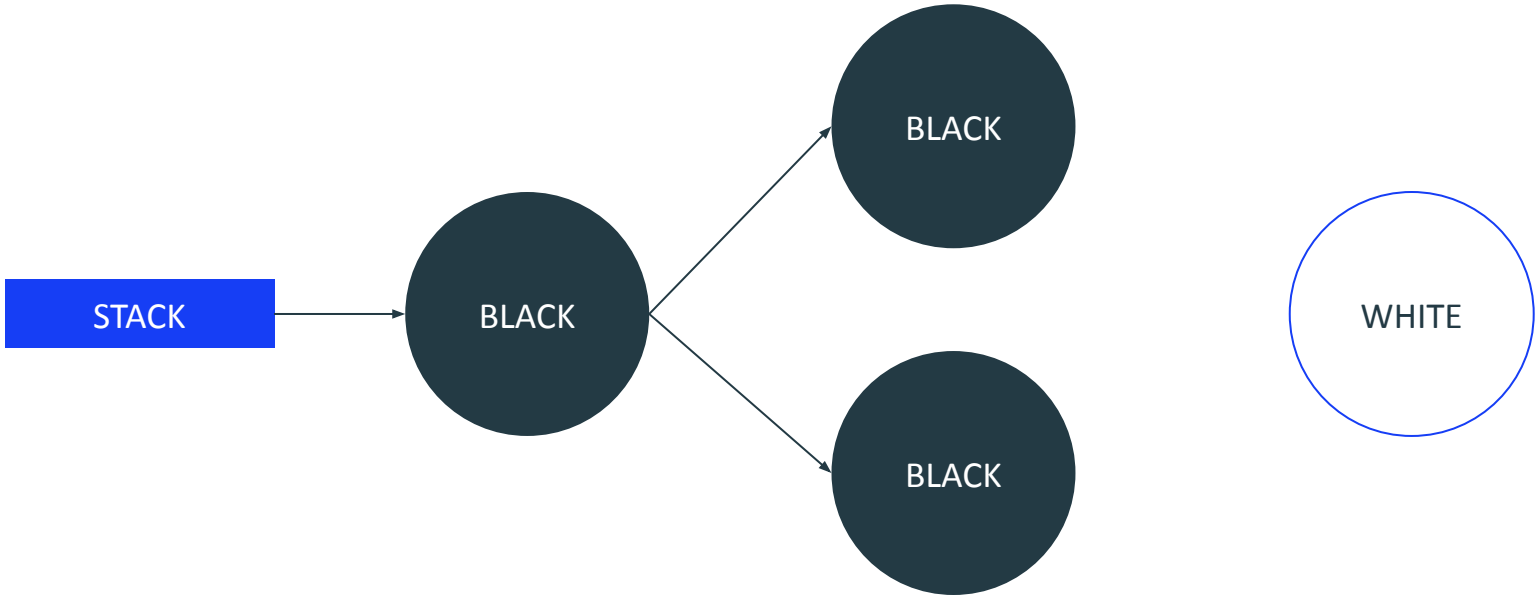
Tricolor Concurrent Nonmoving Mark and Sweep



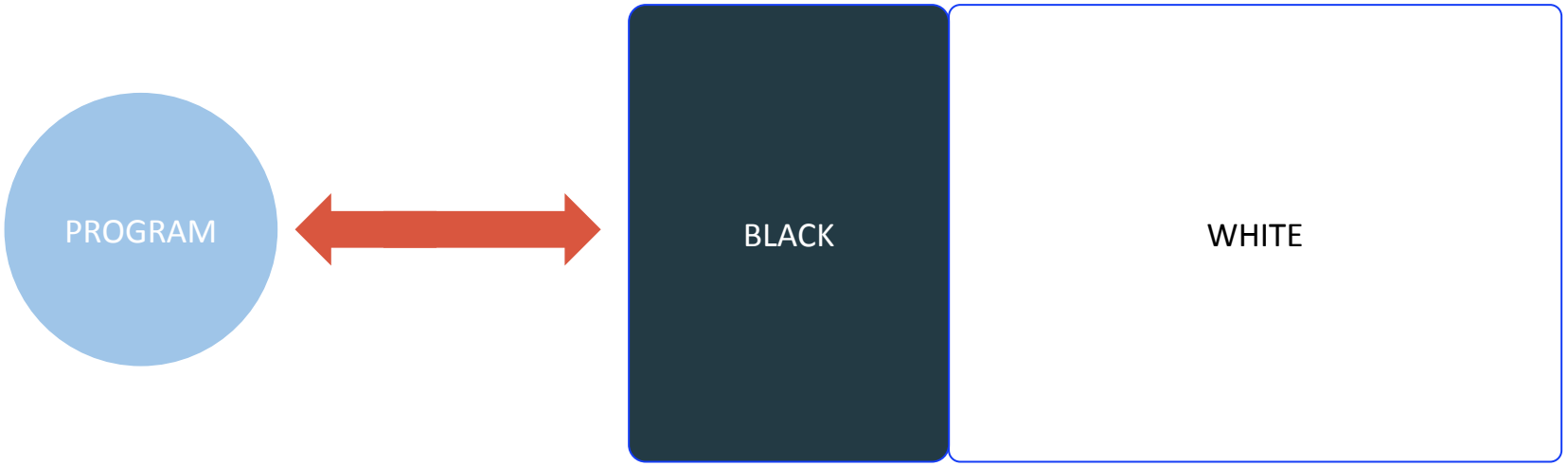
Tricolor Concurrent Nonmoving Mark and Sweep



Tricolor Concurrent Nonmoving Mark and Sweep



Tricolor Concurrent Nonmoving Mark and Sweep



Thanks!