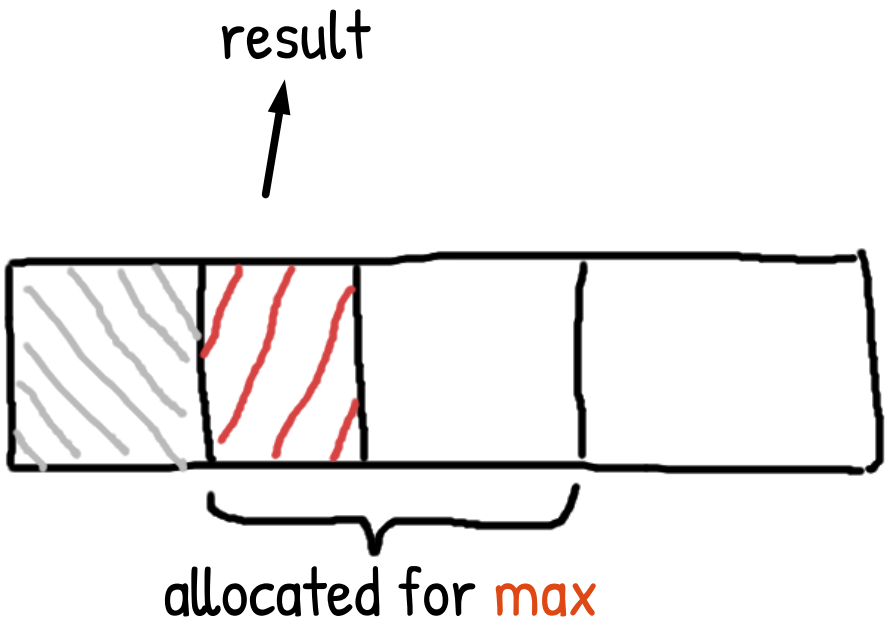
The garbage collector

What to expect

- Some history: Memory allocation in C
- Go memory allocation
- GC in action
- What's under the hood
- Conclusion

History: the stack

```c
#include <stdio.h>

int max(int first, int second) {
  int result = first;
  if (second > first) {
    result = second;
  }
  return result;
}

int main(int argc, char *argv[]) {
  printf("max of %d and %d is %d\n",
    2, 5, max(2,5));
}
```



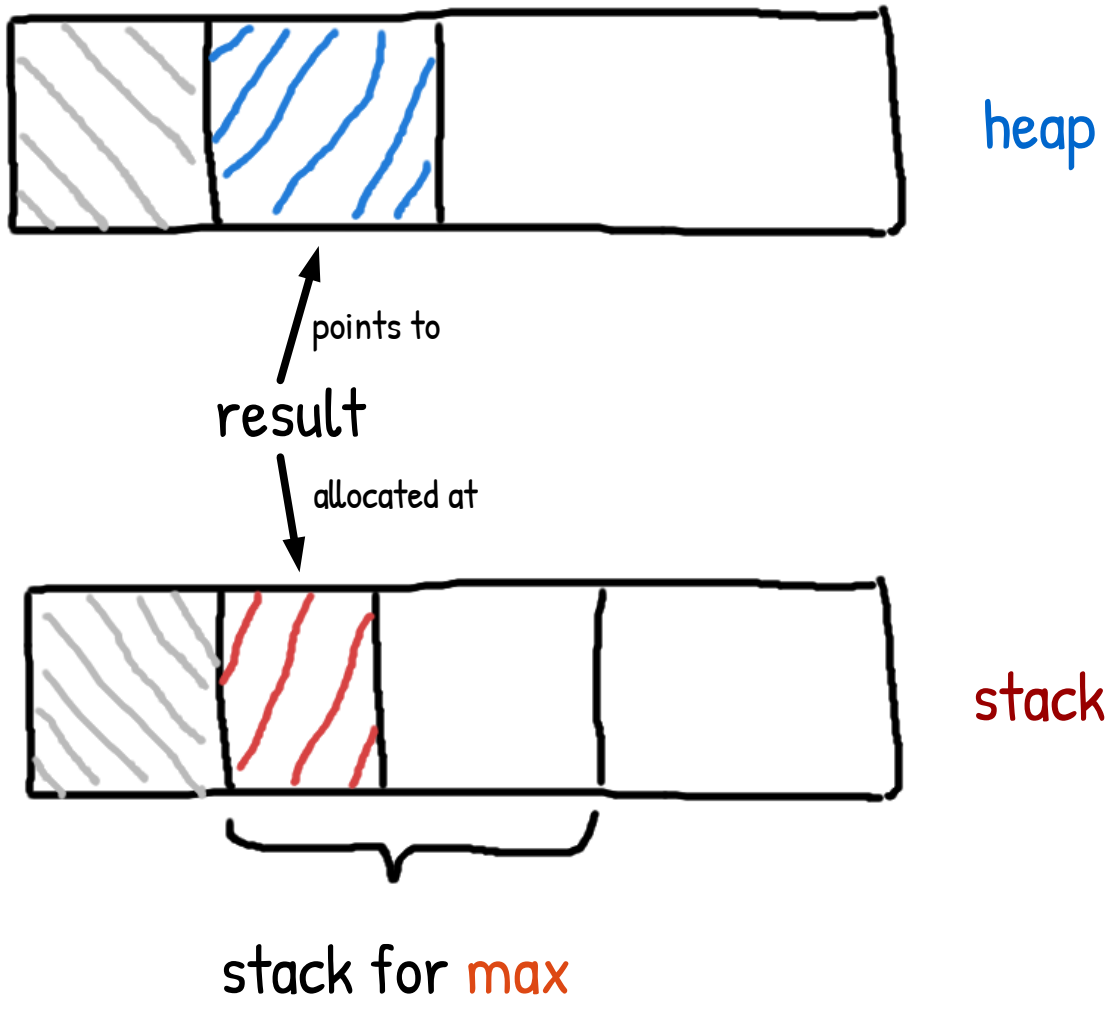
History: the stack

```c
#include <stdio.h>

int max(int first, int second) {
  int result = first;
  if (second > first) {
    result = second;
  }
  return result;
}

int main(int argc, char *argv[]) {
  printf("max of %d and %d is %d\n",
    2, 5, max(2,5));
}
```

memory for max deallocated

History: the heap

```c
#include <stdio.h>
#include <stdlib.h>

int *max(int first, int second) {
  int *result = malloc(sizeof(int));
  *result = first;
  if (second > first) {
    *result = second;
  }
  return result;
}

int main(int argc, char *argv[]) {
  int *myMax = max(2,5);
  printf("max of %d and %d is %d\n",
    2, 5, *myMax);
  free(myMax);
}
```



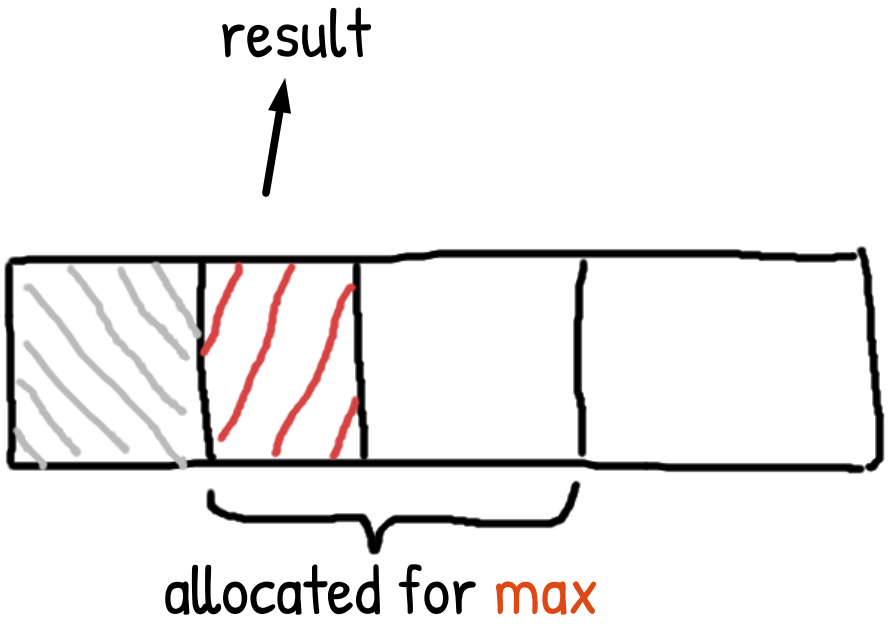
Memory allocation in Go

```
package main

import `fmt`

func max(first, second int) int {
  var result int = first
  if second > first {
    result = second
  }
  return result
}

func main() {
  fmt.Printf("max of %d and %d is %d\n", 2,
   5, max(2, 5))
}
```

Where is result allocated?

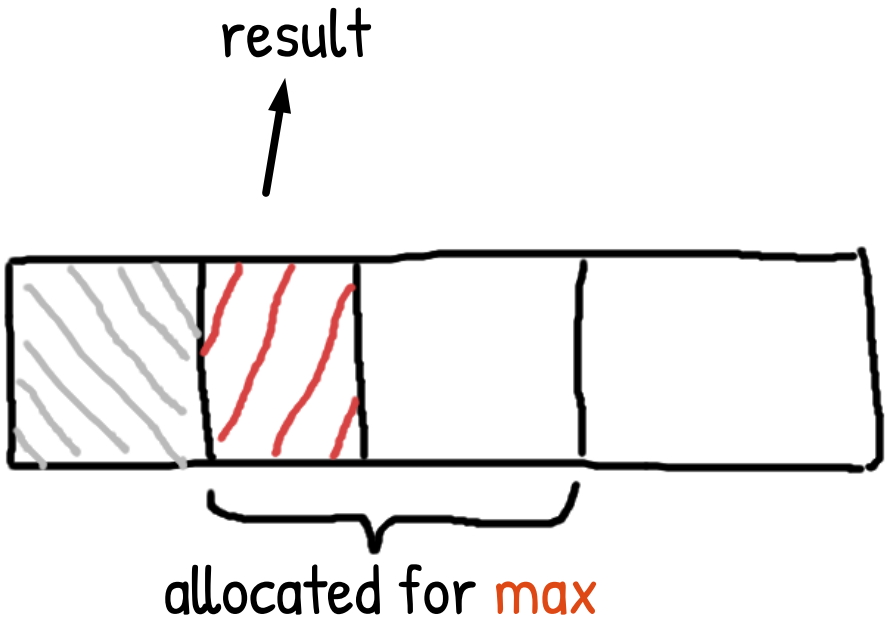
Memory allocation in Go

```
package main

import `fmt`

func max(first, second int) int {
  var result int = first
  if second > first {
    result = second
  }
  return result
}

func main() {
  fmt.Printf("max of %d and %d is %d\n", 2,
   5, max(2, 5))
}
```



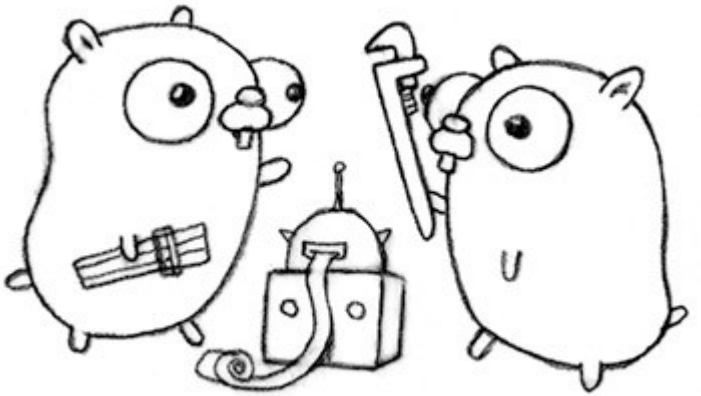
Memory allocation in Go

```
package main

import `fmt`

func max(first, second int) int {
  var result int = first
  if second > first {
    result = second
  }
  return result
}

func main() {
  fmt.Printf("max of %d and %d is %d\n", 2,
   5, max(2, 5))
}
```



Memory allocation in Go

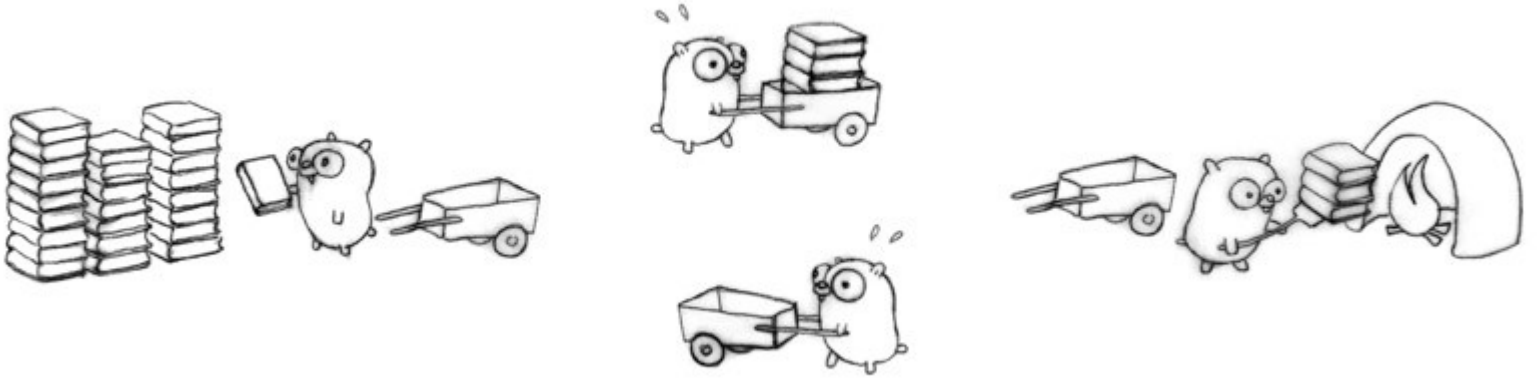
```
package main
import `fmt`

type Gopher struct {
  Weight int
}

func breed(first, second *Gopher) *Gopher {
  return &Gopher{(first.Weight+second.Weight)/2}
}

func main() {
  dad := Gopher{200}
  mom := Gopher{250}
  fmt.Printf("%v plus %v is %v\n",
    dad, mom, breed(&dad, &mom))
}
```



Where is the kid Gopher?

the Garbage Collector

- To clean up heap memory that nobody cares about



GC in action

Why should you care?

- Go GC is a stop-the-world mark & sweep garbage collector
  -> i.e: no code running when GC
  -> i.e: slow response or timeout

i.e: timeout spike

Why should you care?

- Example: money transfer:

A -> [send money from user X to Y] -> B

A (waiting for ACK)..... -> B (GC. stopped)

A (timeout->failed)..... -> B (ran & ACK)

A (resend before getting ACK) -> B

Tuning

- GOGC= X (default = 100)
- GOGC=off
- runtime/debug/#SetGCPercent.
- -gctrace, -gcdead

Questions?