Garbage Collector combines the following approaches:

1. Concurrency
2. Tri-colour
3. Mark and Sweep

## Tri-colour

- white objects are not yet processed by the GC
  * May/may not be discoverable
- Grey objects are discovered and reachable from the roots

- Black objects are reachable and fully processed
  - It's descendants are also reachable

## Write Barriers

When a pointer that references a white object is written to a black object, that object is marked as grey.

- This prevents objects being swept prematurely
- Allows the program to run concurrently

*Prevents race conditions of an object being swept when a white objects pointer gets written to a black object

# Phases of a GC cycle

1. Setup
2. Mark
3. Mark Termination
4. Sweep

Mark phase involves tracing the heap and identifying reachable objects.

Mark Termination ensures all goroutines reach a 'Garbage Collection safepoint' and scans remaining grey objects in the worklist and the stack.

Mark collection is the only phase that is not concurrent. This is because once all goroutines reach a GC safepoint, they're all stopped while the stack and remaining grey objects are scanned.

All other phases are concurrent!

- Go's GC is both concurrent and parrallel

Parrallelisation occurs by GC tasks occuring across

# Stack Frames

- A stack frame is a portion of the call stack

- Each time a function is called, a new stack frame is allocated

- A stack frame holds a function's variables, return addresses and other function-specific data

- Deallocated once the function completes

## Stack Size

- Each goroutine is assigned a small stack

- The size of a stack is dynamic

## Stack Resizing

- Occurs when a function needs more space in the stack

1. A larger stack is allocated

2. All stack contents are copied over and pointers are updated

3. Old stack is deallocated

- Stack shrinking occurs when GC cycle notices large unused stack space

- Memory deallocated from stack shrinking is given to the heap

- Stack shrinking increases efficiency in memory management

- process of shrinking/growing stack space incurs overhead!

* This is because when a stack is increased in size, a new stack must be created and its contents moved

- Due to resizing overhead, stack space is not shrunk immediately, only when a large portion is not used

# Heap Growth Ratio

- GOGC is used to trigger the garbage collection cycle when the heap grows a certain multiplier w.r.t. heap size at end of last GC cycle
- GOGC = 100 will mean that the GC cycle will start when the heap grows twice the size of the heap at the end of last GC cycle!

- GOGC = 50 will mean GC is triggered when heap grows 1.5 times of heap size at the end of last GC cycle.

# Finalisers

- Execute cleanup ops before GC deallocates an object

- Used to free non-memory resources

  * File descriptors
  * Network connections
  * Database Handlers

- Invoked when GC notices an object is not reachable

- Go doesn't guarantee that a finaliser will be run
  * when the program doesn't terminate cleanly for eg: os.Exit() is called

- Not deterministic
- Adds complexity