.NET Memory Management Poster II [1)2)]

by @konradkokosa https://prodotnetmemory.com

Allocation triggers:

AllocSmall - running out of budget on Gen 0 during object allocation.

AllocLarge - running out of budget on LOH during large object allocation

OutOfSpaceSOH - running out of SOH ephemeral segment space

OutOfSpaceLOH - running out of LOH segments space

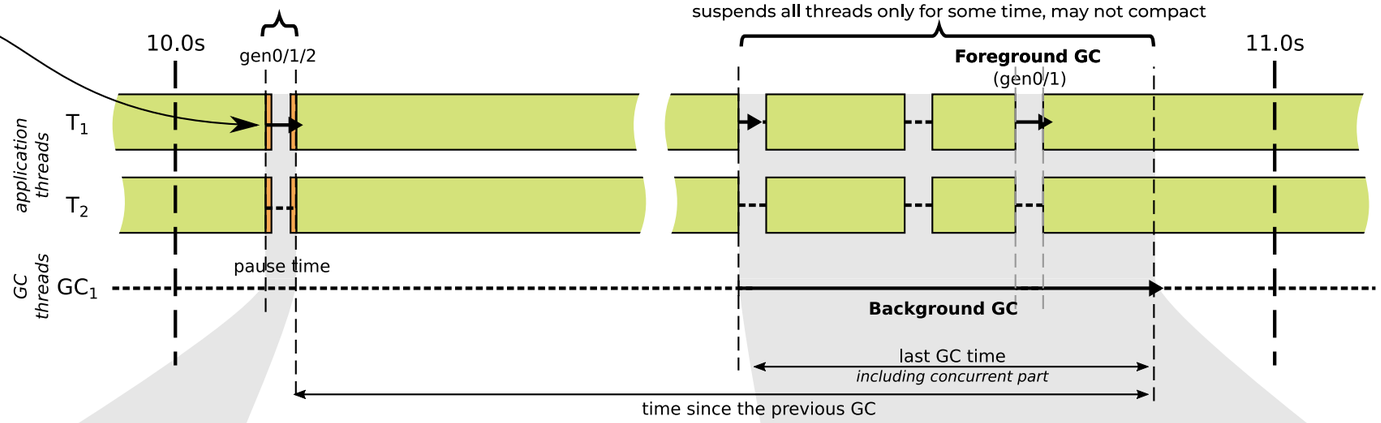
Induced - called explictly from code

LowMemory - operating system has sent *low-memory notification*

Internal - various GC reasons, like: *AppDomain* unload or cleaning up *Thread* objects

"Stop-the-world" GC

suspends **all** threads, may compact



Background GC

with optional Foreground GCs

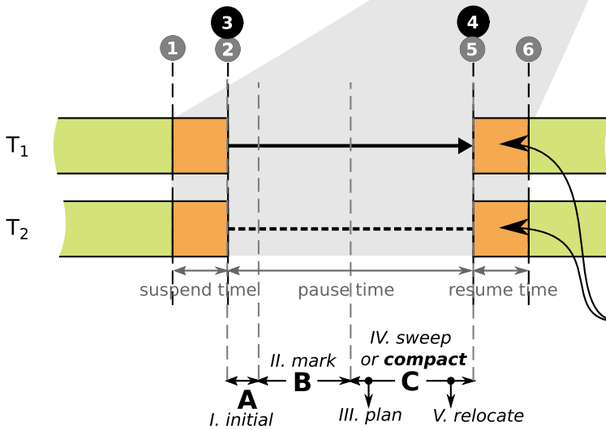
suspends **all** threads only for some time, may not compact

$$\%\ \text{Time in GC} = \frac{\text{last GC time}}{\text{since previous GC time}}\ [\%]$$

GC types:

(as shown in PerfView)

N - non-concurrent GC (blocking)
B - Background GC
F - Foreground GC (blocking collection of an ephemeral generations during Background GC)
I - induced (manually triggered) blocking GC
i - induced non-blocking GC



I. Choose **condemned generation**
II. Marking of reachable objects in the condemned and younger generations
III. Plan - the GC decides whether compacting is worth doing or maybe sweeping is just enough
IV. Sweep or...
V. Relocate and **Compact** - to update all addresses to the new ones

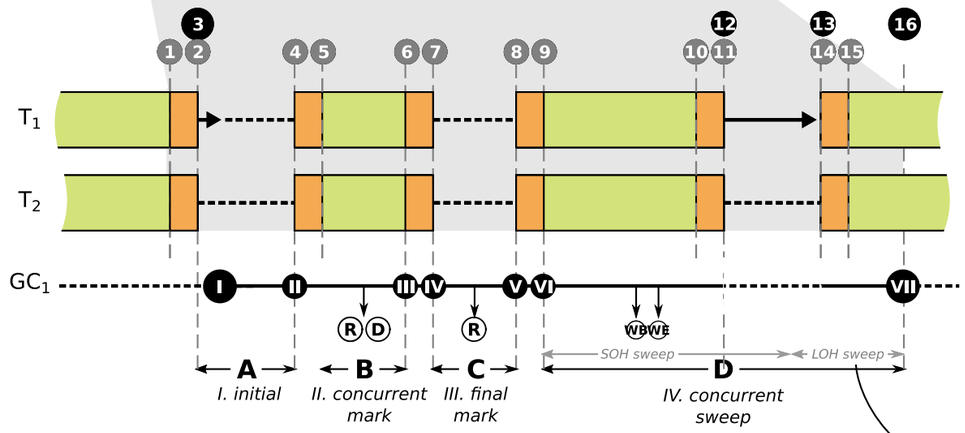
ETW/LTTng events

1 GC/SuspendEEStart
2 GC/SuspendEEStop
3 GC/Start
4 GC/Stop
5 GC/RestartEEStart
6 GC/RestartEEStop

Thread suspend/resume

Every managed method may be:
Fully interruptible - thread running it may be suspended at any time
Partially interruptible - thread running it may be suspended at specified *safe points* (mostly, method calls)

GCInfo data knows where *GC roots* live (what registers, where on stack) for the current method instruction



Ia. Choose **condemned generation**
Ib. Initial marking - finding GC roots
II. Concurrent Marking of reachable objects in the condemned and younger generations, starting from the initial GC roots
III. Final Mark - revisit reachability from pages that has been modified during concurrent marking
IV. Concurrent **Sweep**

ETW/LTTng events

1 GC/SuspendEEStart
2 GC/SuspendEEStop
3 GC/Start
4 GC/RestartEEStart
5 GC/RestartEEStop
6 GC/SuspendEEStart
7 GC/SuspendEEStop
8 GC/RestartEEStart
9 GC/RestartEEStop
10 GC/SuspendEEStart
11 GC/SuspendEEStop
12 GC/Start
13 GC/Stop
14 GC/RestartEEStart
15 GC/RestartEEStop
16 GC/Stop

0+ Foreground GCs

I BGCStart
II BGC1stNonCondStop
R BGCRevisit
D BGCDrainMark
III BGC1stConStop
IV BGC2ndNonConStart
R BGCRevisit
V BGC2ndNonConStop
VI BGC2ndConStart
VII BGC2ndConStop

LOH allocations not allowed

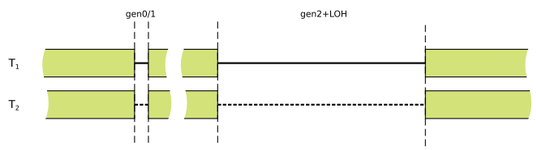
(produces *LOH Allocation Pause (due to background GC) > 200 Msec* report in PerfView)

WB BGCAllocWaitBegin
WE BGCAllocWaitEnd

Workstation GC

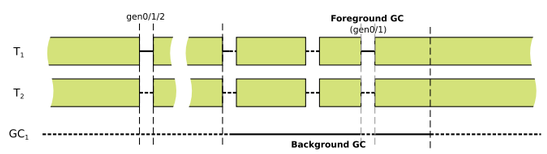
minimize pauses for better **interactivity**

Workstation Non-concurent



- No GC threads
- Only "stop-the-world" GC (may compact)

Background Workstation

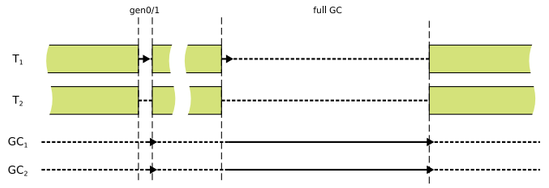


- Single GC thread
- Gen 0/1 (and sometimes 2) - "stop-the-world" GC (may compact)
- Mostly - Background GC (concurrent sweep)

Server GC

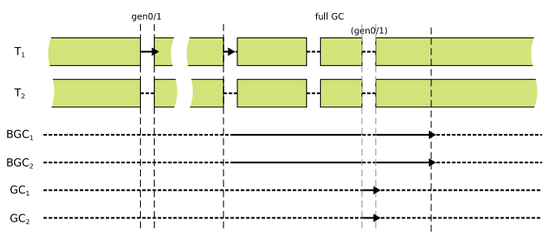
maximize resources usage for better **throughput**

Server Non-concurrent



- By default, N GC threads (N - # of cores)
- Only "stop-the-world" GC (may compact)

Background Server



- By default, N GC threads (N - # of cores)
- Gen 0/1 (and sometimes 2) - "stop-the-world" GC (may compact)
  Mostly - Background GC (concurrent sweep)

1) Based on .NET Core and 2 CPU cores with 2 user threads, 64-bit Windows
2) Many identifiers (like **AllocSmall** or **% Time in GC**) are used as defined in *PerfView* tool

Legend: Working user thread | Suspending thread | Working GC thread | Suspended thread

ver 1.0.2