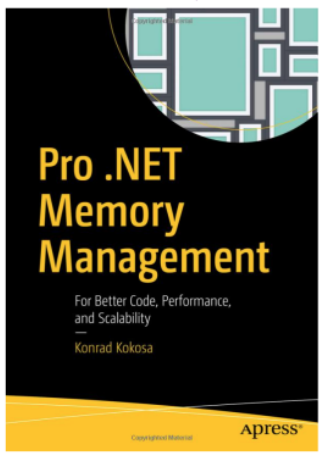
.NET GC Internals

Introduction

@konradkokosa / @dotnetosorg

About me

.NET freelancer/ trainer/ consultant/ speaker/ ...



prodotnetmemory.com

.NET meetups/ conferences/ webinars/ ...

diagnosticsexpert.com

asyncexpert.com

OutOfMemory card game

(still prototyping)

playoutofmemory.com

Microsoft® Most Valuable Professional

TooSlowException.com

@konradkokosa

.NET GC Internals series

- what it is NOT...?

.NET GC Internals series

- what it is NOT...?
  - practical debugging/diagnosing/measuring series

.NET GC Internals series

- what it is NOT...?
  - practical debugging/diagnosing/measuring series
  - practical best-practices programming series

.NET GC Internals series

- what it is NOT...?
  - practical debugging/diagnosing/measuring series
  - practical best-practices programming series
- what it is...?

.NET GC Internals series

- what it is NOT...?
  - practical debugging/diagnosing/measuring series
  - practical best-practices programming series
- what it is...?
  - in DEPTH explanation of the most important .NET GC parts 🥹

.NET GC Internals series

- what it is NOT...?
  - practical debugging/diagnosing/measuring series
  - practical best-practices programming series
- what it is...?
  - in DEPTH explanation of the most important .NET GC parts 🥹
  - the thing that is **always** too short during trainings due to the lack of time...

.NET GC Internals series

- what it is NOT...?
  - practical debugging/diagnosing/measuring series
  - practical best-practices programming series
- what it is...?
  - in DEPTH explanation of the most important .NET GC parts 🥹
  - the thing that is **always** too short during trainings due to the lack of time...
  - "interactive ~1h lectures" - *ad hoc* drawings, **do not hesitate** to Q&A

.NET GC Internals series

- what it is NOT...?
  - practical debugging/diagnosing/measuring series
  - practical best-practices programming series
- what it is...?
  - in DEPTH explanation of the most important .NET GC parts 🥹
  - the thing that is **always** too short during trainings due to the lack of time...
  - "interactive ~1h lectures" - *ad hoc* drawings, **do not hesitate** to Q&A
  - a preliminary material before future trainings

01. .NET GC Internals - Introduction

This module agenda:

- mini-series roadmap
- fundamentals
  - manual vs automatic memory management
  - reference counting and... a little of Rust
  - tracing Garbage Collection
- GC in .NET basics
  - types
  - history
- first dive into the .NET 5 runtime source code
  - building & debugging CoreCLR
  - gc.cpp 😍
- materials

.NET GC Internals

Agenda:

- Introduction
  - roadmap and fundamentals, source code, ...
- **Mark** phase
  - roots, object graph traversal, *mark stackc, mark/pinned flag,* mark list*, ...
- **Concurrent Mark** phase
  - *mark array/mark word*, concurrent visiting, *floating garbage*, *write watch list*, ...
- **Plan** phase
  - *gap*, *plug*, *plug tree*, *brick table*, *pinned plug*, *pre/post plug*, ...
- **Sweep** phase
  - *free list threading*, concurrent sweep, ...
- **Compact** phase
  - *relocate* references, compact, ...
- **Generations**
  - physical organization, *card tables*, ...
- **Allocations**
  - *bump pointer allocator*, free list allocator, *allocation context*, ...
- ...?!

Fundamentals

Explicit allocation/deallocation

```c
#include<stdio.h>
int main()
{
    int *ptr;
    ptr = (int*)malloc(sizeof(int));
    if (ptr == 0)
    {
        printf("ERROR: Out of memory\n");
        return 1;
    }
    *ptr = 25;
    printf("%d\n", *ptr);
    free(ptr);
    return 0;
}
```

Dangers:

Explicit allocation/deallocation

```c
#include<stdio.h>
int main()
{
    int *ptr;
    ptr = (int*)malloc(sizeof(int));
    if (ptr == 0)
    {
        printf("ERROR: Out of memory\n");
        return 1;
    }
    *ptr = 25;
    printf("%d\n", *ptr);
    free(ptr);
    return 0;
}
```

Dangers:

- Memory leak
- Dangling pointer

Automatic memory management

Garbage Collector – the old concept:

Automatic memory management

Garbage Collector – the old concept:

- John McCarthy. *Recursive functions of symbolic expressions and their computation by machine.* - 1958
- George E. Collins. *A method for overlapping and erasure of lists.* - 1960

Automatic memory management

Garbage Collector – the old concept:

- John McCarthy. *Recursive functions of symbolic expressions and their computation by machine.* - 1958
- George E. Collins. *A method for overlapping and erasure of lists.* - 1960

so... Lisp

Automatic memory management

- reference counting
- tracing

Automatic memory management

- reference counting
- tracing
- Rust

Reference counting

- for every object, maintain a *counter* of *references* pointing to it
- C++ "shared pointers":

```
int main()
{
  std::shared_ptr<Foo> sh2(new Foo);
  std::cout << sh2.use_count() << '\n';
}
```

- COM
- ...

Reference counting

- pros:
  - deterministic/immediate deallocation
  - incremental work
  - simple implementation (well...)

Reference counting

- pros:
  - deterministic/immediate deallocation
  - incremental work
  - simple implementation (well...)
- cons:
  - mutator's overhead

Reference counting

- pros:
  - deterministic/immediate deallocation
  - incremental work
  - simple implementation (well...)
- cons:
  - mutator's overhead - although *"recent work narrowed the gap between reference counting and the fastest tracing collectors to **within 10%**"*

Reference counting

- pros:
  - deterministic/immediate deallocation
  - incremental work
  - simple implementation (well...)
- cons:
  - mutator's overhead - although *"recent work narrowed the gap between reference counting and the fastest tracing collectors to **within 10%**"*
  - cyclic references

Reference counting

- pros:
  - deterministic/immediate deallocation
  - incremental work
  - simple implementation (well...)
- cons:
  - mutator's overhead - although *"recent work narrowed the gap between reference counting and the fastest tracing collectors to **within 10%**"*
  - cyclic references
- unless something more sophisticated:
  - deferred reference counting - avoiding counting reference stored on the stack
  - mixed - Python (and you can disable the GC)
  - RC Immix - Taking Off the Gloves with Reference Counting Immix paper, '2013

Affine type system - Rust

```
let s1 = String::from("hello");
let s2 = s1;

println!("{}, world!", s1);
```

Affine type system - Rust

```rust
let s1 = String::from("hello");
let s2 = s1;

println!("{}, world!", s1);
```

```
error[E0382]: use of moved value: `s1`
 --> src/main.rs:5:28
  |
3 |     let s2 = s1;
  |         -- value moved here
4 |
5 |     println!("{}, world!", s1);
  |                            ^^ value used here after move
  |
  = note: move occurs because `s1` has type `std::string::String`, which does
  not implement the `Copy` trait
```

Affine type system - Rust

```
fn main() {
    let s = String::from("hello");  // s comes into scope

    takes_ownership(s);             // s's value moves into the function...
                                    // ... and so is no longer valid here


}
fn takes_ownership(some_string: String) { // some_string comes into scope
    println!("{}", some_string);
} // Here, some_string goes out of scope and `drop` is called. The backing
  // memory is freed.
```

Affine type system - Rust

```
fn main() {
    let s = String::from("hello");

    use(&s);
}

fn use(some_string: &String) {
    // …
}
```

Tracing

- occasionally *trace* what is being used and reclaim unused memory

Tracing

- occasionally *trace* what is being used and reclaim unused memory
- pros:
  - simple to use
  - "faster" - smaller mutators' overhead, only occasional pauses
  - handles cyclic references out of the box

Tracing

- occasionally *trace* what is being used and reclaim unused memory
- pros:
  - simple to use
  - "faster" - smaller mutators' overhead, only occasional pauses
  - handles cyclic references out of the box
- cons:
  - non-deterministic deallocation
  - not completly "pauseless"
  - hard to implement

GC in .NET

GC in .NET

There are various runtimes but a few GC implementations:

- .NET Framework
- .NET Core/.NET 5
- Mono
- .NET Compact Framework
- Silverlight

GC in .NET

There are various runtimes but a few GC implementations:

- .NET Framework
- .NET Core/.NET 5
- Mono
- .NET Compact Framework
- Silverlight

And in .NET Framework/Core we have four main GCs "flavours" available:

	Concurrent (false)	**Concurrent (true)**
Workstation	Non-Concurrent Workstation	Background Workstation
Server	Non-Concurrent Server	Background Server

GC in .NET

There are various runtimes but a few GC implementations:

- .NET Framework
- .NET Core/.NET 5
- Mono
- .NET Compact Framework
- Silverlight

And in .NET Framework/Core we have four main GCs "flavours" available:

	Concurrent (false)	**Concurrent (true)**
Workstation	Non-Concurrent Workstation	Background Workstation
Server	Non-Concurrent Server	Background Server

They are all some (important) modifications of **the tracing garbage collection, without reference counting usage**.

GC in .NET

- application "type":

GC in .NET

- application "type":
  - Workstation - designed for responsiveness needed in interactive, UI-based applications
    - pauses as short as possible
    - good citizen in the whole interactive environment - single Managed Heap (and up to single CPU core usage)

GC in .NET

- application "type":
  - Workstation - designed for responsiveness needed in interactive, UI-based applications
    - pauses as short as possible
    - good citizen in the whole interactive environment - single Managed Heap (and up to single CPU core usage)
  - Server - designed for simultaneous, request-based processing
    - maximizing throughput
    - "give me all" citizen in the system - multiple Managed Heaps and GC threads

GC in .NET

- application "type":
  - Workstation - designed for responsiveness needed in interactive, UI-based applications
    - pauses as short as possible
    - good citizen in the whole interactive environment - single Managed Heap (and up to single CPU core usage)
  - Server - designed for simultaneous, request-based processing
    - maximizing throughput
    - "give me all" citizen in the system - multiple Managed Heaps and GC threads
- concurrency:

GC in .NET

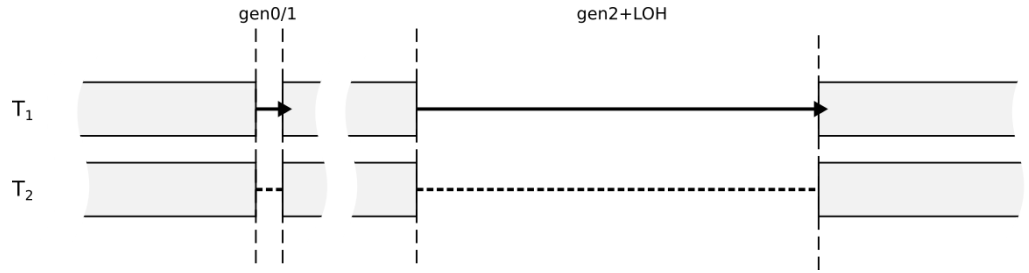
- application "type":
  - Workstation - designed for responsiveness needed in interactive, UI-based applications
    - pauses as short as possible
    - good citizen in the whole interactive environment - single Managed Heap (and up to single CPU core usage)
  - Server - designed for simultaneous, request-based processing
    - maximizing throughput
    - "give me all" citizen in the system - multiple Managed Heaps and GC threads
- concurrency:
  - Non-Concurrent - simple, "stop the world" and do the job
    - optimal, no-one interrupts

GC in .NET

- application "type":
  - Workstation - designed for responsiveness needed in interactive, UI-based applications
    - pauses as short as possible
    - good citizen in the whole interactive environment - single Managed Heap (and up to single CPU core usage)
  - Server - designed for simultaneous, request-based processing
    - maximizing throughput
    - "give me all" citizen in the system - multiple Managed Heaps and GC threads
- concurrency:
  - Non-Concurrent - simple, "stop the world" and do the job
    - optimal, no-one interrupts
  - Background - some parts of the GC run concurrently with the application
    - almost pauseless
    - (currently) non-compacting

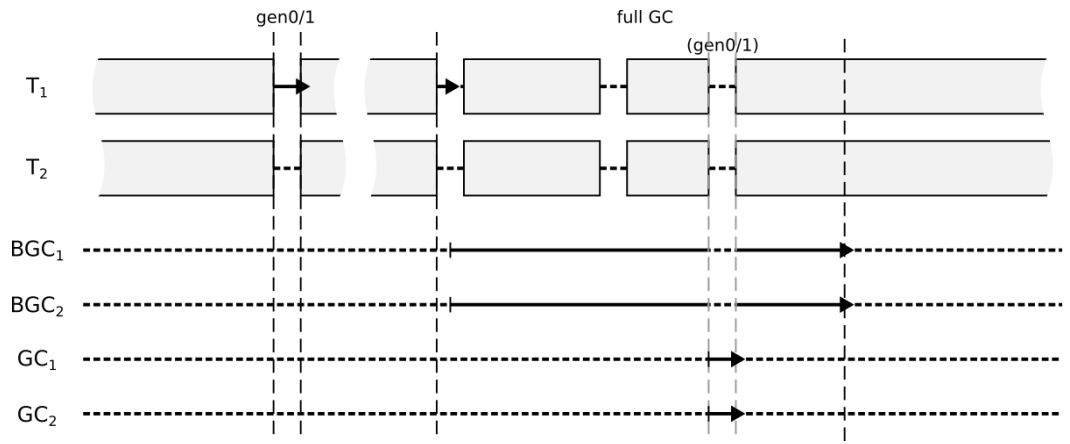
GC in .NET

Non-Concurrent Workstation:



vs

Background Server:



GC in .NET history

- Patric Dussud is the author of the .NET GC
  - *"My dad collects garbage at Microsoft"* - Patric Dussud's son

GC in .NET history

- Patric Dussud is the author of the .NET GC
  - *"My dad collects garbage at Microsoft"* - Patric Dussud's son
- So the story goes...:

GC in .NET history

- Patric Dussud is the author of the .NET GC
  - *"My dad collects garbage at Microsoft"* - Patric Dussud's son
- So the story goes...:
  - JScript - Microsoft's dialect of the ECMAScript standard (avoiding JavaScript trademark issues :))
    - "four person over a few weekends", with **conservative GC prototype** written by Patric

GC in .NET history

- Patric Dussud is the author of the .NET GC
  - *"My dad collects garbage at Microsoft"* - Patric Dussud's son
- So the story goes...:
  - JScript - Microsoft's dialect of the ECMAScript standard (avoiding JavaScript trademark issues :))
    - "four person over a few weekends", with **conservative GC prototype** written by Patric
  - -> "JVM" 0.1 - Microsoft's experimental JVM implementation - based on the same conservative GC prototype

GC in .NET history

- Patric Dussud is the author of the .NET GC
  - *"My dad collects garbage at Microsoft"* - Patric Dussud's son
- So the story goes...:
  - JScript - Microsoft's dialect of the ECMAScript standard (avoiding JavaScript trademark issues :))
    - "four person over a few weekends", with **conservative GC prototype** written by Patric
  - -> "JVM" 0.1 - Microsoft's experimental JVM implementation - based on the same conservative GC prototype
  - -> "JVM" 0.2 (Lisp to C++) - prototype of a better, generational GC in Lisp (because he felt comfortable in Lisp for experiments). Then **transpiled to C++**.

GC in .NET history

- Patric Dussud is the author of the .NET GC
  - *"My dad collects garbage at Microsoft"* - Patric Dussud's son
- So the story goes...:
  - JScript - Microsoft's dialect of the ECMAScript standard (avoiding JavaScript trademark issues :))
    - "four person over a few weekends", with **conservative GC prototype** written by Patric
  - -> "JVM" 0.1 - Microsoft's experimental JVM implementation - based on the same conservative GC prototype
  - -> "JVM" 0.2 (Lisp to C++) - prototype of a better, generational GC in Lisp (because he felt comfortable in Lisp for experiments). Then **transpiled to C++**.
  - ~> C++ (CLR) - based on the experiments and the same algorithm

GC in .NET history

- Patric Dussud is the author of the .NET GC
  - *"My dad collects garbage at Microsoft"* - Patric Dussud's son
- So the story goes...:
  - JScript - Microsoft's dialect of the ECMAScript standard (avoiding JavaScript trademark issues :))
    - "four person over a few weekends", with **conservative GC prototype** written by Patric
  - -> "JVM" 0.1 - Microsoft's experimental JVM implementation - based on the same conservative GC prototype
  - -> "JVM" 0.2 (Lisp to C++) - prototype of a better, generational GC in Lisp (because he felt comfortable in Lisp for experiments). Then **transpiled to C++**.
  - ~> C++ (CLR) - based on the experiments and the same algorithm
- at the times of ~.NET Framework 2.0 taken over by Maoni Stephens

.NET Core source code

.NET Core source code

- there was an important repositories merge:
  - https://github.com/dotnet/runtime - .NET 5.0+
  - https://github.com/dotnet/coreclr & https://github.com/dotnet/corefx - up to .NET Core 3.1
- choose proper branch/tag:
  - https://github.com/dotnet/runtime/tree/release/5.0
- the GC code is there:
  - https://github.com/dotnet/runtime/tree/release/5.0/src/coreclr/src/gc

Building .NET Core

- Instructions: https://github.com/dotnet/runtime/blob/master/docs/workflow/README.md
  - including requirements, fe. for Windows
- checkout interesting tag/release: **`git checkout release/5.0`**
- build in current arch/target, **`Debug`**, both runtime & libs, without tests: **`build.cmd`**
  - or not... issue #41886
  - manually applying PR #41900
- Visual Studio solution file is created as a build artifact under:
  `<reporoot>\artifacts\obj\coreclr\windows.<platform>.<configuration>\CoreCLR.sln`

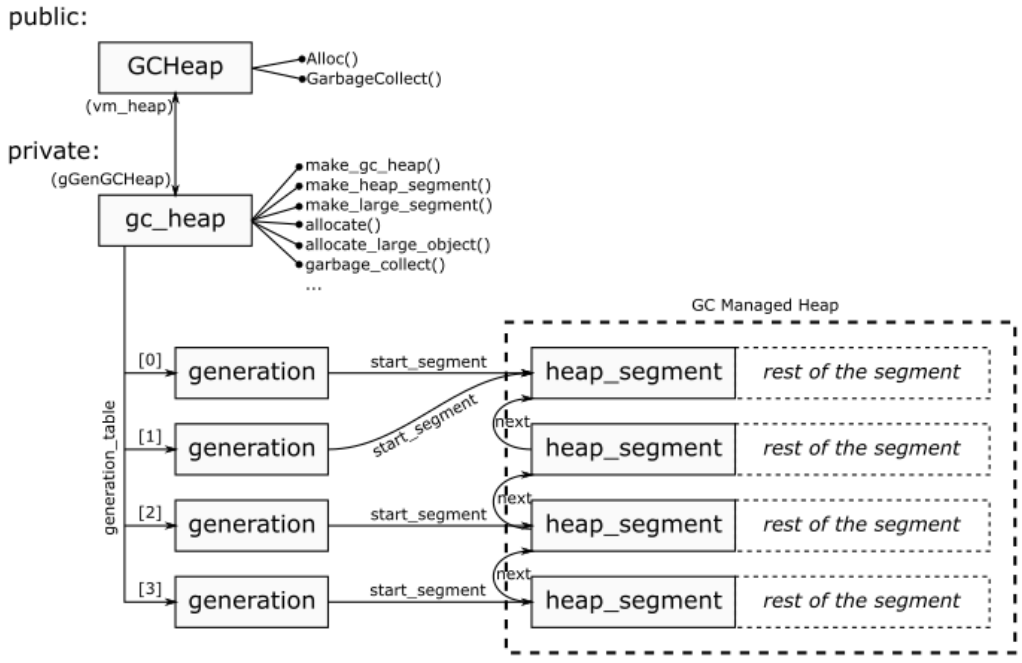
Debugging .NET Core

- Instructions are pretty straightforward
- we can use Visual Studio (nice experience) or WinDbg (nice SOS)

.NET GC source code

- But...
  - gc.cpp -> few classes
  - gc.cpp -> ~39000 lines, ~1.35 MB
  - `#ifdef`, `#ifdef`, `#ifdef`, ...

.NET GC source code



- **`GCHeap`** - public API for the Execution Engine (methods like **`Allocate`** or **`GarbageCollect`**)
  - Workstation mode - only single instance
  - Server mode - additional instances per every Managed Heap
- **`gc_heap`** - internal API used by **`GCHeap`** (**`allocate`**, **`garbage_collect`**, **`make_gc_heap`**, ...)
  - Workstation mode - all relevants methods are compiled as static
  - Server mode - as many as Managed Heaps

.NET GC source code - Server/Workstation GC

gc.cpp has <40 kLOC of C++

.\src\gc\gcsvr.cpp defines **SERVER_GC** constant and **SVR** namespace:

```
#define SERVER_GC 1
namespace SVR {
#include "gcimpl.h"    // <-- defines MULTIPLE_HEAPS
#include "gc.cpp"
}
```

.\src\gc\gcwks.cpp defines **WKS** namespace:

```

namespace WKS {
#include "gcimpl.h"
#include "gc.cpp"
}
```

.NET GC source code - Server/Workstation GC

...and then the whole **gc.cpp** begins...

```
heap_segment* gc_heap::get_segment_for_loh (size_t size
  #ifdef MULTIPLE_HEAPS
                                            , gc_heap* hp
#endif //MULTIPLE_HEAPS
                                            )
{
  #ifndef MULTIPLE_HEAPS
    gc_heap* hp = 0;
#endif //MULTIPLE_HEAPS
    heap_segment* res = hp->get_segment (size, TRUE);
```

.NET GC source code - Non-Concurrent/Background GC

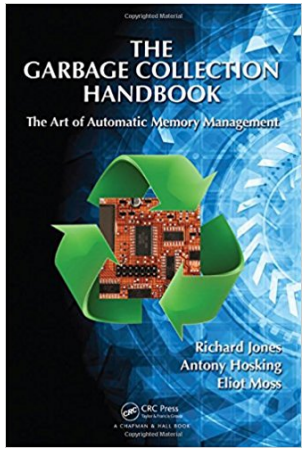
- **.\src\gc\gc.cpp** consumes **BACKGROUND_GC** constant
- always defined in both SVR and WKS versions
- dynamic flag checked

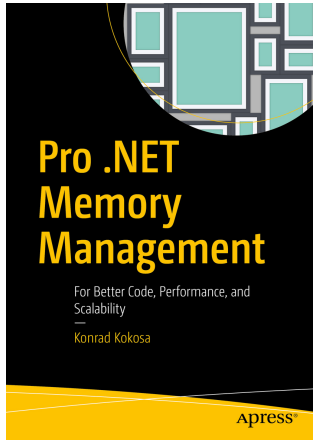
```
void GCStatistics::AddGCStats(const gc_mechanisms& settings, size_t timeInMSec)
{
  #ifdef BACKGROUND_GC
     if (settings.concurrent)
    {
        bgc.Accumulate((uint32_t)timeInMSec*1000);
        cntBGC++;
    }
    else if (settings.background_p)
    {
        // ...
```

Materials

Books:

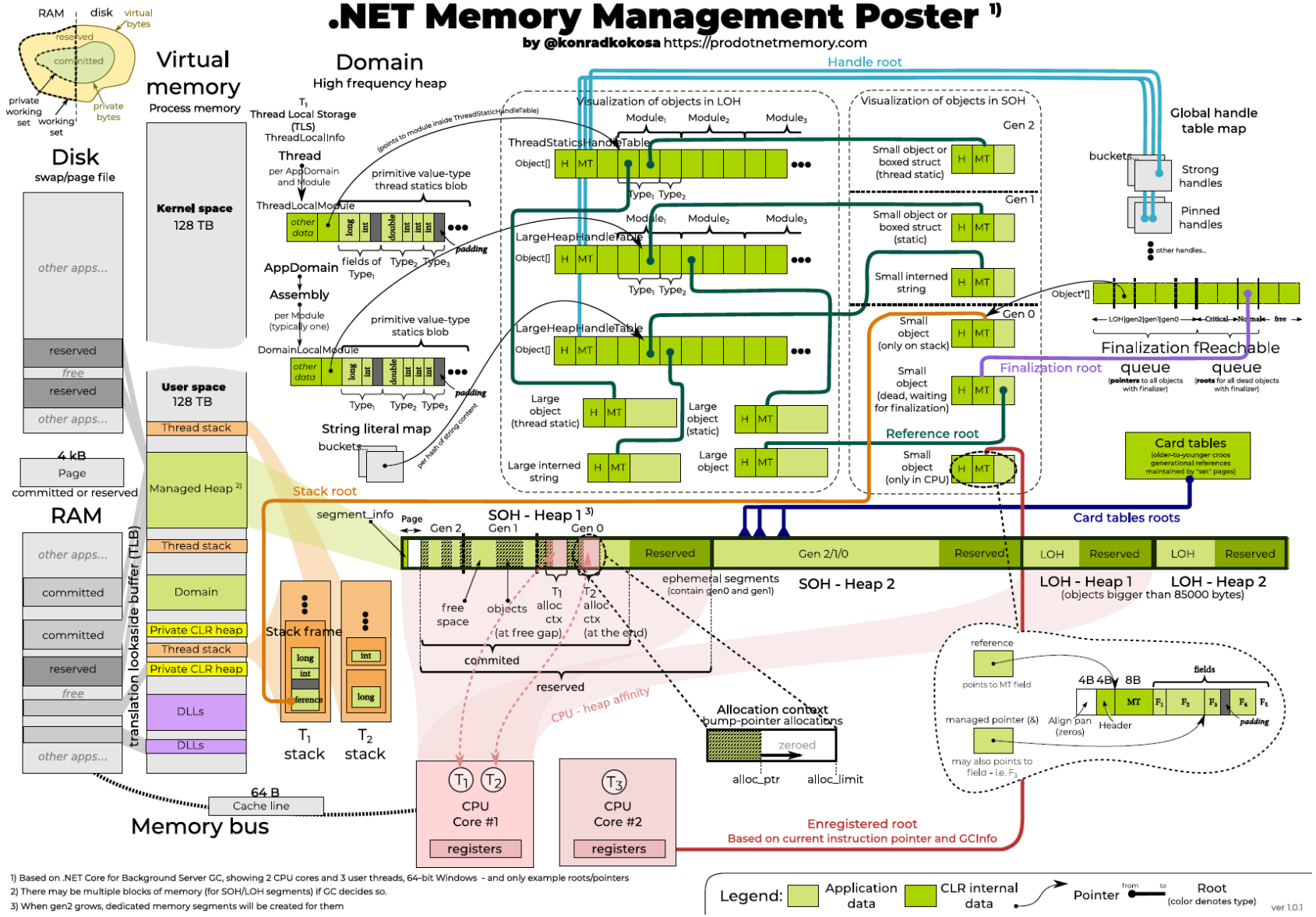
- The Garbage Collection Handbook - Richard Jones, Antony Hosking, Eliot Moss
- Pro .NET Memory Management Management - Konrad Kokosa





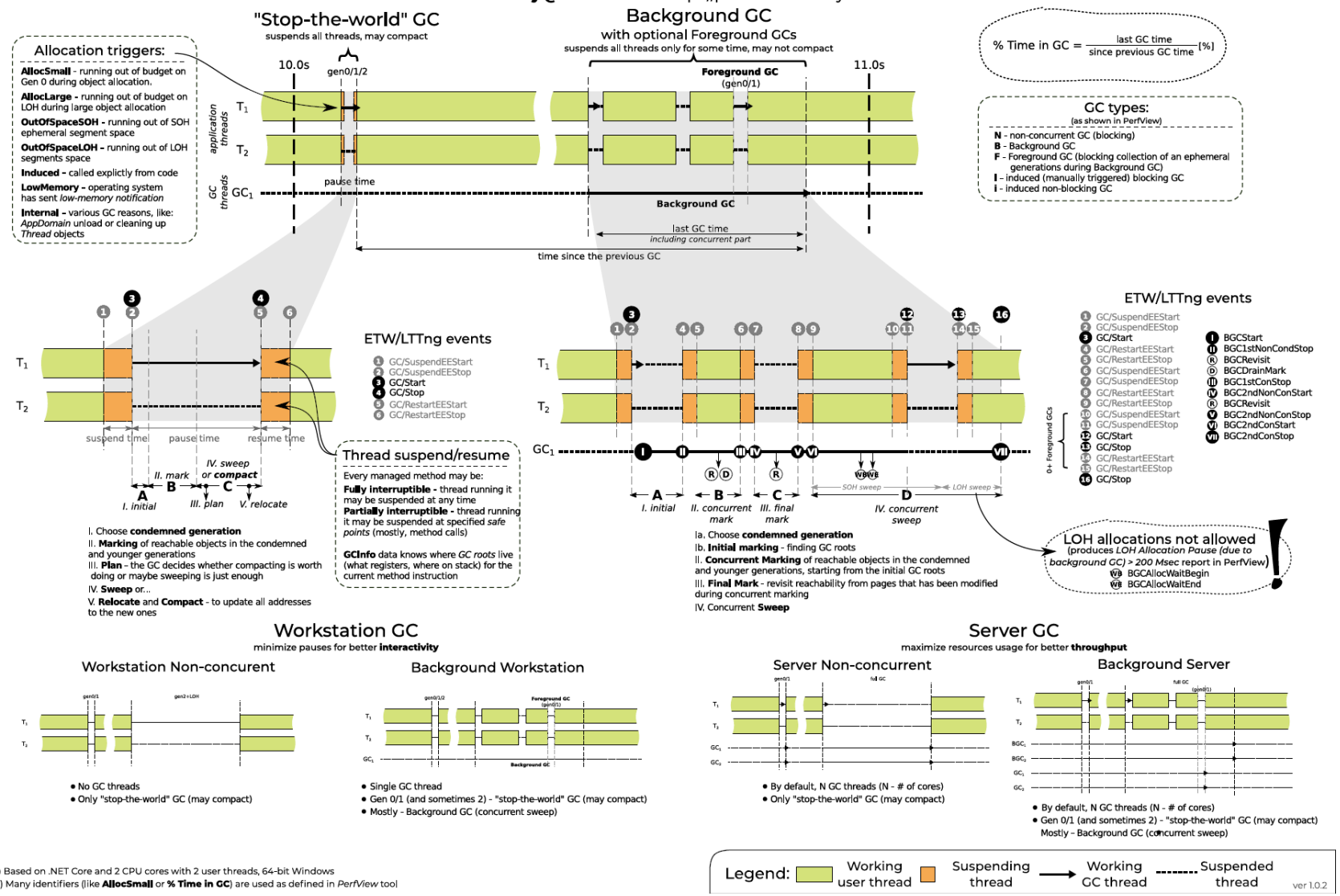
Sites:

- TooSlowException.com & Pro .NET Memory book site - Konrad Kokosa
- Maoni Stephens blog
- Maoni's awesome .NET Memory Performance Analysis document
- The Book of the Runtime


.NET Memory Management Poster 1)
by @konradkokosa https://prodotnetmemory.com
RAM
disk
virtual bytes
reserved
committed
private working set
working set
private bytes
Virtual memory
Process memory
Disk
swap/page file
other apps...
reserved
free
reserved
other apps...
4 kB
Page
committed or reserved
RAM
other apps...
committed
committed
reserved
free
other apps...
translation lookaside buffer (TLB)
64 B
Cache line
Memory bus
Kernel space
128 TB
User space
128 TB
Thread stack
Managed Heap 2)
Thread stack
Domain
Private CLR heap
Thread stack
Private CLR heap
DLLs
DLLs
Domain
High frequency heap
Thread Local Storage (TLS)
ThreadLocalInfo
Thread
per AppDomain and Module
ThreadLocalModule
other data
long
int
double
padding
fields of Type1
primitive value-type thread statics blob
AppDomain
Assembly
per Module (typically one)
DomainLocalModule
primitive value-type statics blob
String literal map
buckets...
per hash of string content
Handle root
Visualization of objects in LOH
ThreadStaticsHandleTable
LargeHeapHandleTable
Large object (thread static)
Large object (static)
Large interned string
Large object
Visualization of objects in SOH
Gen 2
Small object or boxed struct (thread static)
Gen 1
Small object or boxed struct (static)
Small interned string
Gen 0
Small object (only on stack)
Small object (dead, waiting for finalization)
Small object (only in CPU)
Reference root
Global handle table map
buckets...
Strong handles
Pinned handles
other handles...
Finalization root
Finalization fReachable queue
queue
Card tables
Card tables roots
Stack root
segment_info
Page
Gen 2
Gen 1
Gen 0
SOH - Heap 1 3)
Reserved
Gen 2/1/0
Reserved
LOH
Reserved
LOH
Reserved
ephemeral segments (contain gen0 and gen1)
SOH - Heap 2
LOH - Heap 1
LOH - Heap 2
(objects bigger than 85000 bytes)
free space
objects (at free gap)
T1 alloc ctx
T2 alloc ctx (at the end)
commited
reserved
CPU - heap affinity
Stack frame
long
int
reference
int
long
T1 stack
T2 stack
Allocation context
bump-pointer allocations
zeroed
alloc_ptr
alloc_limit
reference
points to MT field
managed pointer (&)
may also points to field - i.e. F3
Align pan (zeros)
Header
4B 4B 8B
fields
MT
F1
F2
F3
F4
F5
padding
CPU Core #1
registers
CPU Core #2
registers
Enregistered root
Based on current instruction pointer and GCInfo
1) Based on .NET Core for Background Server GC, showing 2 CPU cores and 3 user threads, 64-bit Windows - and only example roots/pointers
2) There may be multiple blocks of memory (for SOH/LOH segments) if GC decides so.
3) When gen2 grows, dedicated memory segments will be created for them
Legend:
Application data
CLR internal data
Pointer
from
to
Root
(color denotes type)
ver 1.0.1

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

by @konradkokosa https://prodotnetmemory.com



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

Thank you! Any questions?!