SELA | DEVELOPER | PRACTICE
July 3-5, 2018

Kevin Gosse @kookiz
Gregory Léocadie @gleocadie
Christophe Nasarre @chnasarre

.NET Memory Models

.NET Basics: reference = pointer… that moves

- Value type = struct / enum
  - Allocated on stack or embedded inside a type instance
  - Not controlled by GC
  - Passed by value to methods
- Reference type = class
  - Allocated on the *Managed Heap*
  - Can be moved in memory by the GC
  - Passed by reference to methods

.NET Basics: Common (strongly) Type System

- Value types
  - Primitive `int i;`
  - Enums `enum State { Off, On }`
  - Structs `struct Point { int x, y; }`
- Reference types
  - Classes `class Foo: Bar, Ifoo { … }`
  - Interfaces `interface IFoo: IBar { … }`
  - Arrays `string[] a = new string[10];`
  - Delegates `delegate void Empty();`

GC: an history of generation and size

- The CLR allocates *segments* in process address space
  - Normal Heap: objects < 85,000 bytes (managed by GC)
  - Large Object Heap: objects > 85,000 bytes (managed by GC but not compacted - fragmentation)

- A garbage collection is started when an allocation is requested
  1. Look for referenced objects
  2. Move referenced objects to avoid holes (= compaction)
  3. Go back to the initial allocation

- Each time an object survives a collection, it goes to the next generation
  - Gen 0: short lived objects
  - Gen 2: long lived objects or… memory leak
  - Gen 1: a kind of purgatory between the other two generations

GC: not so “behind the scene” as advertised

- Look for ALL referenced objects
- Copy memory block during compaction phase
  - This is why LOH exists but risk of fragmentation (less important issue in 64 bit)
- ALL threads in the process are frozen (STW syndrom)
- Non deterministic: might occur on any allocation
- Impact of types that implement a `~Finalize` method
- Don’t call `GC.Collect()`: could break the GC self-tuning algorithms

Questions