GFT

GC Explained

Maciej Paszta

GFT

AGENDA

1. Object layout
2. Memory model
3. Generations
4. GC Workflow
5. Different types of GC

Object layout

- SyncBlk or an index to SyncBlk table, stores
  - information about lock - lock(this)
  - index of object in AppDomain
  - hash code
  - COM interop
- TypeHandle - describes the nature of the object's type, reference to MethodTable
  - MethodTable holds information about available methods, default constructor etc. Single instance per every type.
    - MethodDesc - describes a single method and it's IL code.
    - Interface Map / Interface VTable Map - maps type to interface it's implementing
    - static variables
    - EEClass - complimentary to MethodTable; also describes a type, but it contains less frequently used data (usually needed for JIT compilation)
  - Instance fields
  - String literals - actually points to global list of all string l

Memory model

- Each process works in its own virtual address space
  - developer doesn't operate on physical memory
  - can be fragmented - that is the address space contains holes between allocated space
- Stack
  - where value types (int, decimal, enum, struct etc.) are stored.
  - values are removed from the stack as soon as they fall out of scope (stack frame)
  - not subject of Garbage Collection
- GC Heap (or more precisely managed heap)
  - where reference types are stored and allocated.
  - every memory segment allocated on heap represents a type - that's the criteria for determining whether heap is in a valid state
  - heap divided into 2 groups:
    - Small object heap (< 85,000 bytes): Generation 0, Generation 1, Generation 2
    - Large object heap (>= 85,000 bytes)
      - not compacted (in .NET 4.5.1 developer can override this behavior)
      - only collected on full garbage collection runs

Memory model - internals

- LoaderHeap (High Frequency Heap)
  - CLR artefacts like MethodTable, MethodDesc, FieldDesc - basically the type system.
  - static variables - whether they are reference or value types
    - predictable allocation - decrease the chance of fragmentation
    - not a subject of GC
- StubHeap
  - COM wrapper classes
  - P/Invoke wrappers
  - Code Access Security

Generations

- Generation 0
  - new objects are allocated here
  - most frequently collected generation
  - allocated in ephemeral memory segment
  - ephemeral generation
  - cheap to collect
- Generation 1
  - contains short-lived objects
  - acts as a buffer between gen 0 and gen 1
  - cheap to collect
- Generation 2
  - long-lived objects
  - expensive to collect

Generations

- Flow: Gen0 -> Gen1 -> Gen2
- Ephemeral memory segment:
  - every new memory segment requested by GC
  - designated to store ephemeral generations (Gen0 and Gen1)
  - old ephemeral segments are designated to store Gen2 objects

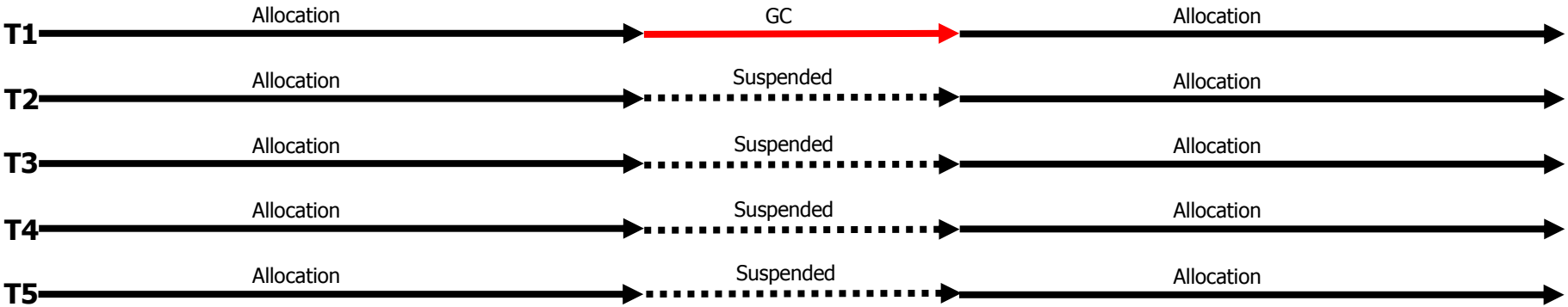
GC Triggers

- The system is low on physical memory
- The amount of allocated memory on managed heap has surpassed the certain threshold
  - threshold is adjusted in runtime (depending on the survival rate of objects)
- Developer called the GC.Collect() method – not advised!
- Developer can prevent GC from running for a small amount of time – GCSettings.LatencyMode

GC Phases

1. Mark - marks objects that are still alive, that is they are reachable from GC Roots:
   - static variables
   - global variables
   - stack objects with references to managed heap objects
2. Relocate updates the references to the objects
   - Objects promoted to higher generation
   - Compacted objects
3. Compaction
   - reclaims memory occupied by dead objects
   - compacts the memory

GC Process


T1
Allocation
GC
Allocation
T2
Allocation
Suspended
Allocation
T3
Allocation
Suspended
Allocation
T4
Allocation
Suspended
Allocation
T5
Allocation
Suspended
Allocation

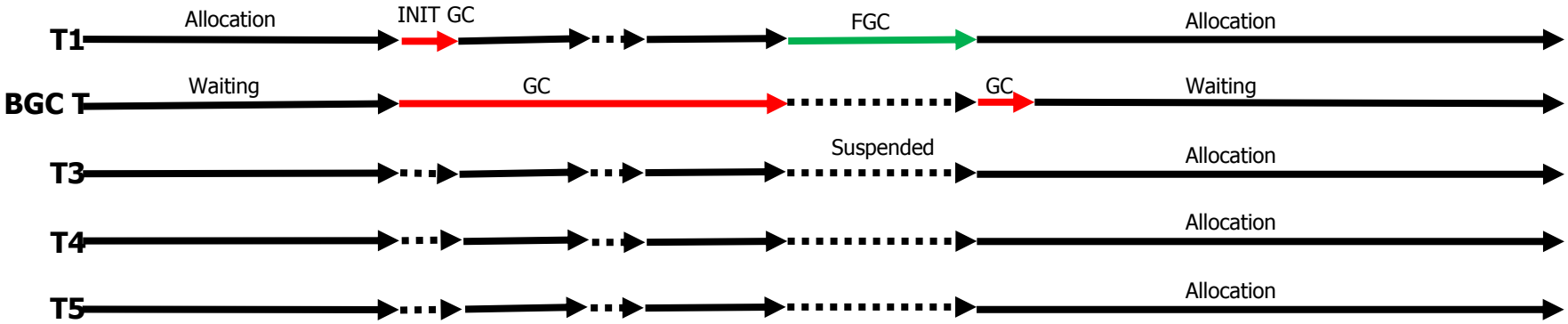
Workstation vs Server GC

- Workstation GC:
  - Default for desktop and console apps
  - Default on 1 CPU machines
  - GC runs on user thread (with normal priority), thus must compete with other threads for CPU time
  - Collections occur less frequently – improved UI responsiveness
  - Can be concurrent (Gen2 collection happens in the background)
- Server GC
  - Default for services
  - Can be turned on by <gcServer> setting
  - Dedicated heap and GC thread for each CPU
  - GC thread running with highest priority
  - Frequent collections – to collect as much temporary objects as possible
  - Can be concurrent (Gen2 collection happens in the background)
- Concurrent GC swapped for Background workstation/server GC in .NET 4.5.1

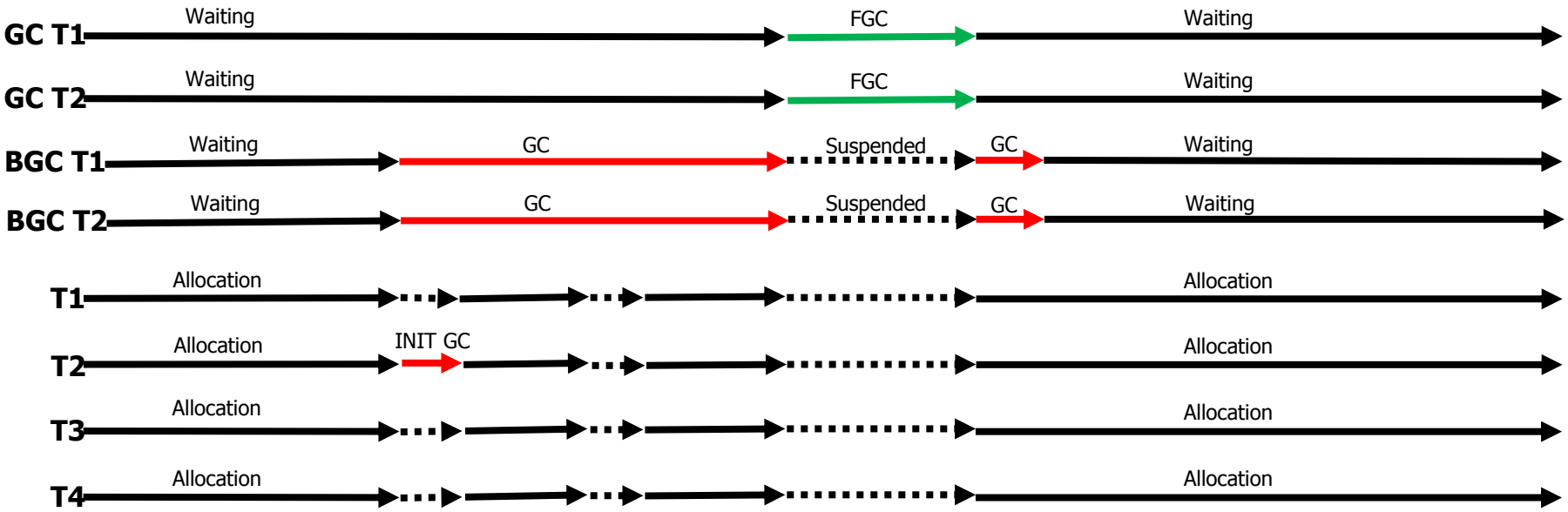
Background GC

- New GC type introduces in .NET 4.5.1
- Dedicated thread to collect Gen2
- Background GC thread doesn’t perform compaction
- Gen0 and Gen1 collections require world-stop (thay can temporarily suspend BGC collections – this is the main difference when compared to concurrent collection)
- Foreground thread swaps ephemeral segments rather than copying individual objects

Background Workstation GC



Background Server GC



Notes

- GC doesn’t affect unmanaged resources
- GC can’t move pinned objects

```
byte[] data = new byte[3];
fixed(byte *ptr = data)
{

}
```

- Types treated in a special way
  - WeakReference
  - ConditionalWeakTable<TKey, TValue>

Useful links

- Background and Foreground GC in .NET 4: http://blogs.msdn.com/b/salvapatuel/archive/2009/06/10/background-and-foreground-gc-in-net-4.aspx
- GC Fundamentals: http://msdn.microsoft.com/en-us/library/0xy59wtx(v=vs.110).aspx
- Large Object Heap Uncovered: http://msdn.microsoft.com/en-us/magazine/cc534993.aspx
- Latency Modes: https://msdn.microsoft.com/en-us/library/bb384202.aspx
- Constrained Execution Regions: https://msdn.microsoft.com/en-us/library/ms228973.aspx

Thank You!

GC Explained

GFT Poland Sp. z o.o.
Maciej Paszta
Technical Architect

Okraglak, Mielzynskiego 14
61-725 Poznan, Polska
maciej.paszta@gft.com