Module 4

.NET GC fundamentals

GC in .NET

There are various runtimes but a few GCs:

- .NET
- .NET Core
- Mono
- .NET Compact Framework
- Silverlight (?)

GC in .NET

There are various runtimes but a few GCs:

- .NET
- .NET Core
- Mono
- .NET Compact Framework
- Silverlight (?)
- .NET Core is open-sourced! - https://github.com/dotnet/coreclr/tree/master/src/gc

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

GC in .NET

Main features:

GC in .NET

Main features:

- **driven by application behavior** - internal heuristics try to tune to your application, e.g. how often it allocates, for how long etc.

GC in .NET

Main features:

- **driven by application behavior** - internal heuristics try to tune to your application, e.g. how often it allocates, for how long etc.
    - we can influence it by changing GC configuration and/or application behavior (reducing allocations, caching, ...)

GC in .NET

Main features:

- **driven by application behavior** - internal heuristics try to tune to your application, e.g. how often it allocates, for how long etc.
    - we can influence it by changing GC configuration and/or application behavior (reducing allocations, caching, ...)
    - *"there's a balance between how much work you need to do as a performance engineer and how much GC handles automatically"* - Maoni

GC in .NET

Main features:

- **driven by application behavior** - internal heuristics try to tune to your application, e.g. how often it allocates, for how long etc.
    - we can influence it by changing GC configuration and/or application behavior (reducing allocations, caching, ...)
    - *"there's a balance between how much work you need to do as a performance engineer and how much GC handles automatically"* - Maoni
- **tracing** (**sweeping** or **compacting**) - it needs to discover objects *reachability*:

GC in .NET

Main features:

- **driven by application behavior** - internal heuristics try to tune to your application, e.g. how often it allocates, for how long etc.
    - we can influence it by changing GC configuration and/or application behavior (reducing allocations, caching, ...)
    - *"there's a balance between how much work you need to do as a performance engineer and how much GC handles automatically"* - Maoni
- **tracing** (**sweeping** or **compacting**) - it needs to discover objects *reachability*:
    - the more objects and bigger graph to traverse, the more work to do

GC in .NET

Main features:

- **driven by application behavior** - internal heuristics try to tune to your application, e.g. how often it allocates, for how long etc.
  - we can influence it by changing GC configuration and/or application behavior (reducing allocations, caching, ...)
  - *"there's a balance between how much work you need to do as a performance engineer and how much GC handles automatically"* - Maoni
- **tracing** (**sweeping** or **compacting**) - it needs to discover objects *reachability*:
  - the more objects and bigger graph to traverse, the more work to do
  - *"the amount of GC work is proportional to how much memory is live, i.e., the survivors"* - Maoni

GC in .NET

Main features:

- **driven by application behavior** - internal heuristics try to tune to your application, e.g. how often it allocates, for how long etc.
    - we can influence it by changing GC configuration and/or application behavior (reducing allocations, caching, ...)
    - *"there's a balance between how much work you need to do as a performance engineer and how much GC handles automatically"* - Maoni
- **tracing** (**sweeping** or **compacting**) - it needs to discover objects *reachability*:
    - the more objects and bigger graph to traverse, the more work to do
    - *"the amount of GC work is proportional to how much memory is live, i.e., the survivors"* - Maoni
- **generational** - it groups objects into *generations* to not consider them all at the same time:

GC in .NET

Main features:

- **driven by application behavior** - internal heuristics try to tune to your application, e.g. how often it allocates, for how long etc.
  - we can influence it by changing GC configuration and/or application behavior (reducing allocations, caching, ...)
  - *"there's a balance between how much work you need to do as a performance engineer and how much GC handles automatically"* - Maoni
- **tracing** (**sweeping** or **compacting**) - it needs to discover objects *reachability*:
  - the more objects and bigger graph to traverse, the more work to do
  - *"the amount of GC work is proportional to how much memory is live, i.e., the survivors"* - Maoni
- **generational** - it groups objects into *generations* to not consider them all at the same time:
  - objects can live much longer than since they became unreachable - if we don't look at a given generation!

GC in .NET

Main features:

- **driven by application behavior** - internal heuristics try to tune to your application, e.g. how often it allocates, for how long etc.
  - we can influence it by changing GC configuration and/or application behavior (reducing allocations, caching, ...)
  - *"there's a balance between how much work you need to do as a performance engineer and how much GC handles automatically"* - Maoni
- **tracing** (**sweeping** or **compacting**) - it needs to discover objects *reachability*:
  - the more objects and bigger graph to traverse, the more work to do
  - *"the amount of GC work is proportional to how much memory is live, i.e., the survivors"* - Maoni
- **generational** - it groups objects into *generations* to not consider them all at the same time:
  - objects can live much longer than since they became unreachable - if we don't look at a given generation!
  - how many objects survive strongly influence whether we should collect it again:
    - *"If GC collects a generation and discovers most objects survived, it wouldn't make sense to collect it so soon again because the goal of a GC is to reclaim memory"* - Maoni
    - in other words, if many survived - it was premature

GC in .NET

Main features:

- **driven by application behavior** - internal heuristics try to tune to your application, e.g. how often it allocates, for how long etc.
  - we can influence it by changing GC configuration and/or application behavior (reducing allocations, caching, ...)
  - *"there's a balance between how much work you need to do as a performance engineer and how much GC handles automatically"* - Maoni
- **tracing** (**sweeping** or **compacting**) - it needs to discover objects *reachability*:
  - the more objects and bigger graph to traverse, the more work to do
  - *"the amount of GC work is proportional to how much memory is live, i.e., the survivors"* - Maoni
- **generational** - it groups objects into *generations* to not consider them all at the same time:
  - objects can live much longer than since they became unreachable - if we don't look at a given generation!
  - how many objects survive strongly influence whether we should collect it again:
    - *"If GC collects a generation and discovers most objects survived, it wouldn't make sense to collect it so soon again because the goal of a GC is to reclaim memory"* - Maoni
    - in other words, if many survived - it was premature

"The one rule to remember" - ***"What survives usually determines how much work GC needs to do; what doesn't survive usually determines how often a GC is triggered"*** - Maoni

GC in .NET

Main features:

GC in .NET

Main features:

- **concurrent** - it is able to reclaim memory while the application is running 🖤

GC in .NET

Main features:

- **concurrent** - it is able to reclaim memory while the application is running 💜
  (but at the time of .NET 6, it is still not able to compact in a concurrent way)

GC in .NET

Main features:

- **concurrent** - it is able to reclaim memory while the application is running 🖤
  (but at the time of .NET 6, it is still not able to compact in a concurrent way)
- **per process** - but it is aware of physical memory load on the machine (*"high memory load"* situation)

GC in .NET

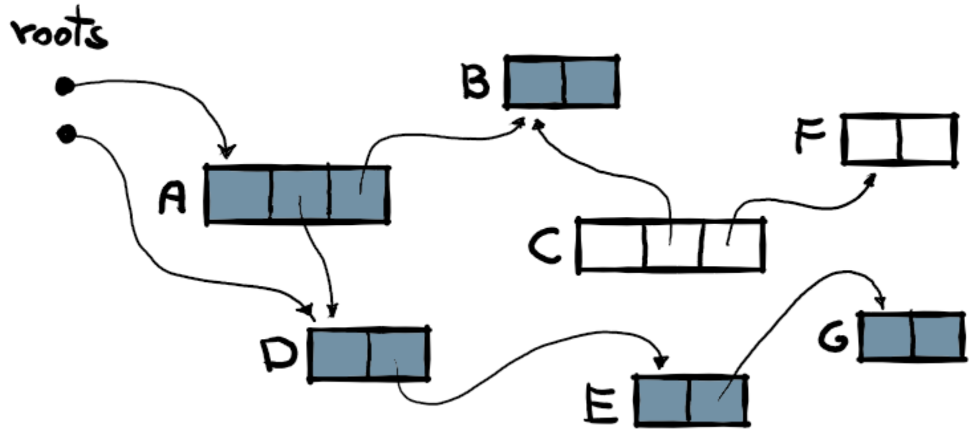
Main features:

- **concurrent** - it is able to reclaim memory while the application is running 💜
  (but at the time of .NET 6, it is still not able to compact in a concurrent way)
- **per process** - but it is aware of physical memory load on the machine (*"high memory load"* situation)
- **multiplatform** - as the .NET itself, it is shared between Windows, Linux and macOS (with some thin OS-dependent layer)

GC in .NET

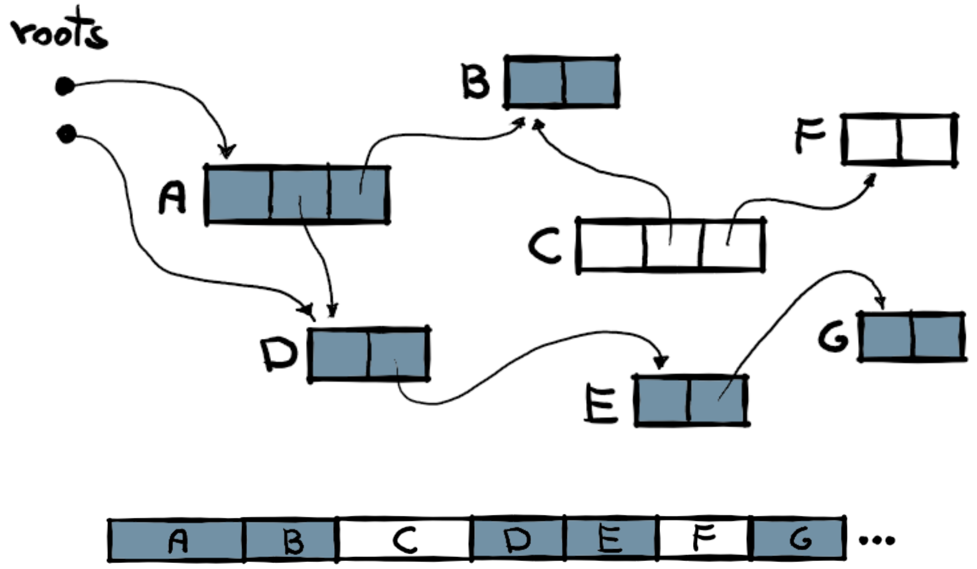
GC in .NET

Mark



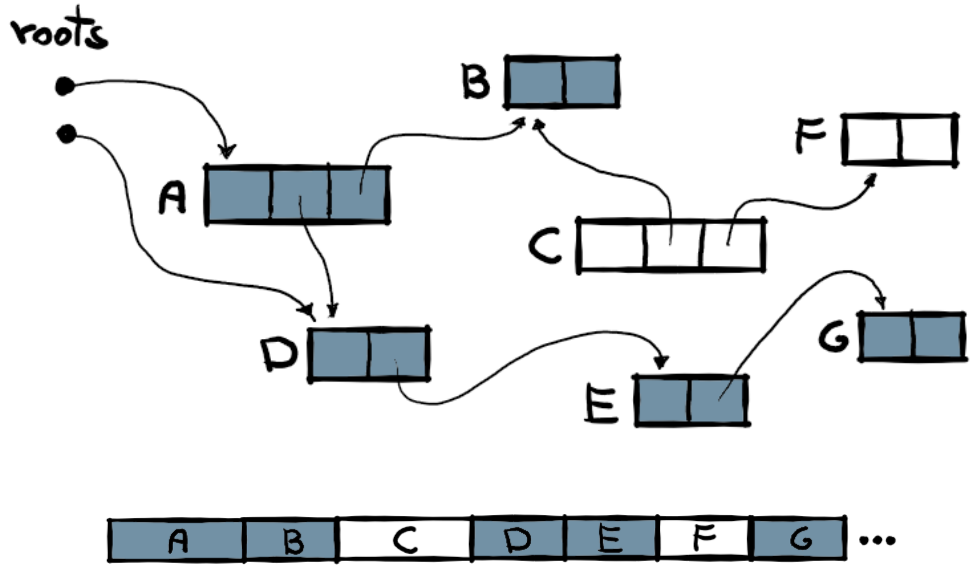
GC in .NET

Mark

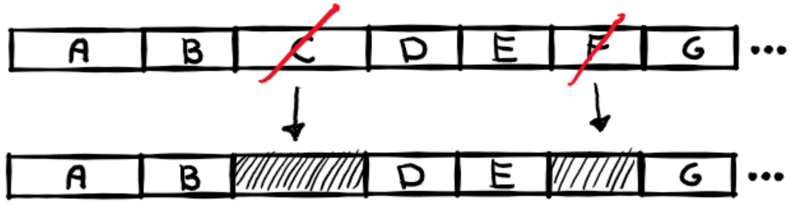


GC in .NET

Mark

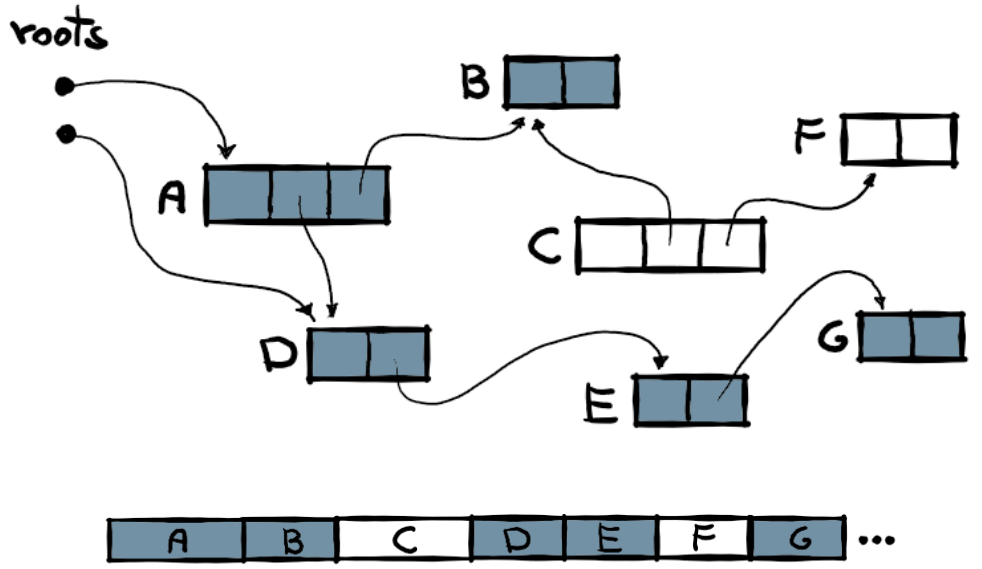


Sweep

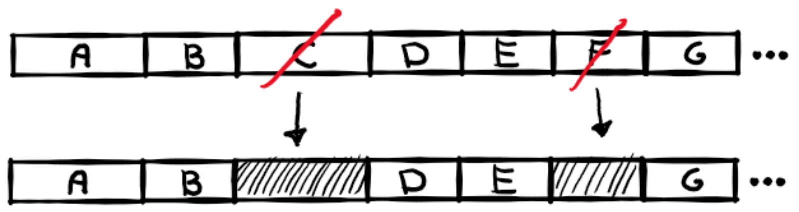


GC in .NET

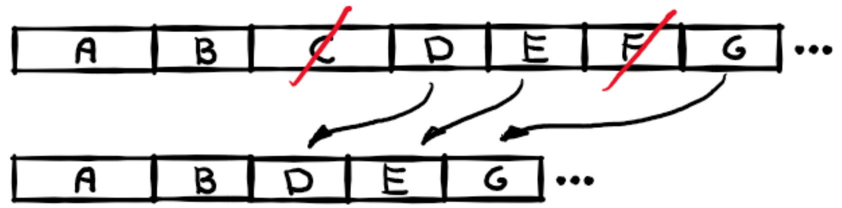
Mark



Sweep



Compact



GC in .NET

- Mark & Sweep or Mark & Compact

GC in .NET

- Mark & Sweep or Mark & Compact
- Mark-**Plan**-Sweep/Compact

GC in .NET

- Mark & Sweep or Mark & Compact
- Mark-**Plan**-Sweep/Compact
    - *"The plan phase simulates a compaction to determine the effective result. If compaction is productive the GC starts an actual compaction; otherwise it sweeps."*

GC in .NET

- Mark & Sweep or Mark & Compact
- Mark-**Plan**-Sweep/Compact
    - *"The plan phase simulates a compaction to determine the effective result. If compaction is productive the GC starts an actual compaction; otherwise it sweeps."*
    - *productive* - is it worth to reduce fragmentation?

GC in .NET

- Mark & Sweep or Mark & Compact
- Mark-**Plan**-Sweep/Compact
    - *"The plan phase simulates a compaction to determine the effective result. If compaction is productive the GC starts an actual compaction; otherwise it sweeps."*
    - *productive* - is it worth to reduce fragmentation? How big is useless fragmentation?

GC in .NET

- Mark & Sweep or Mark & Compact
- Mark-**Plan**-Sweep/Compact
    - *"The plan phase simulates a compaction to determine the effective result. If compaction is productive the GC starts an actual compaction; otherwise it sweeps."*
    - *productive* - is it worth to reduce fragmentation? How big is useless fragmentation? Is it worth to pause an app (remember, currently there is no concurrent compacting GC mode)?

GC triggers

GC triggers

- allocation ("memory shortage") - 99%

GC triggers

- allocation ("memory shortage") - 99%
- explicit `GC.Collect` call - 0.9%

GC triggers

- allocation ("memory shortage") - 99%
- explicit `GC.Collect` call - 0.9%
- *low memory notification* from the OS - 0.1%

GC triggers

- allocation ("memory shortage") - 99%
- explicit `GC.Collect` call - 0.9%
- *low memory notification* from the OS - 0.1%
  - high memory load situations - triggering more aggressive GC (Full, blocking GCs)

GC triggers

- allocation ("memory shortage") - 99%
- explicit `GC.Collect` call - 0.9%
- *low memory notification* from the OS - 0.1%
  - high memory load situations - triggering more aggressive GC (Full, blocking GCs)
  - 90% threshold for less than 80 GiB RAM machines, 90-97% for bigger

GC triggers

- allocation ("memory shortage") - 99%
- explicit `GC.Collect` call - 0.9%
- *low memory notification* from the OS - 0.1%
    - high memory load situations - triggering more aggressive GC (Full, blocking GCs)
    - 90% threshold for less than 80 GiB RAM machines, 90-97% for bigger
    - can be configure with `COMPlus_GCHighMemPercent`

GC triggers

- allocation ("memory shortage") - 99%
- explicit `GC.Collect` call - 0.9%
- *low memory notification* from the OS - 0.1%
  - high memory load situations - triggering more aggressive GC (Full, blocking GCs)
  - 90% threshold for less than 80 GiB RAM machines, 90-97% for bigger
  - can be configure with `COMPlus_GCHighMemPercent`
  - *"For processes running in a container, GC would consider the physical memory based on the container limit."*

GC triggers

- allocation ("memory shortage") - 99%
- explicit `GC.Collect` call - 0.9%
- *low memory notification* from the OS - 0.1%
    - high memory load situations - triggering more aggressive GC (Full, blocking GCs)
    - 90% threshold for less than 80 GiB RAM machines, 90-97% for bigger
    - can be configure with `COMPlus_GCHighMemPercent`
    - *"For processes running in a container, GC would consider the physical memory based on the container limit."*
- no periodic calls!

GC triggers

- allocation ("memory shortage") - 99%
- explicit `GC.Collect` call - 0.9%
- *low memory notification* from the OS - 0.1%
  - high memory load situations - triggering more aggressive GC (Full, blocking GCs)
  - 90% threshold for less than 80 GiB RAM machines, 90-97% for bigger
  - can be configure with `COMPlus_GCHighMemPercent`
  - *"For processes running in a container, GC would consider the physical memory based on the container limit."*
- no periodic calls!
  - the goal of a GC is to reclaim memory, it would be unproductive just to blindly call it periodically

Materials

- Fundamentals of garbage collection
- Garbage collection documentation