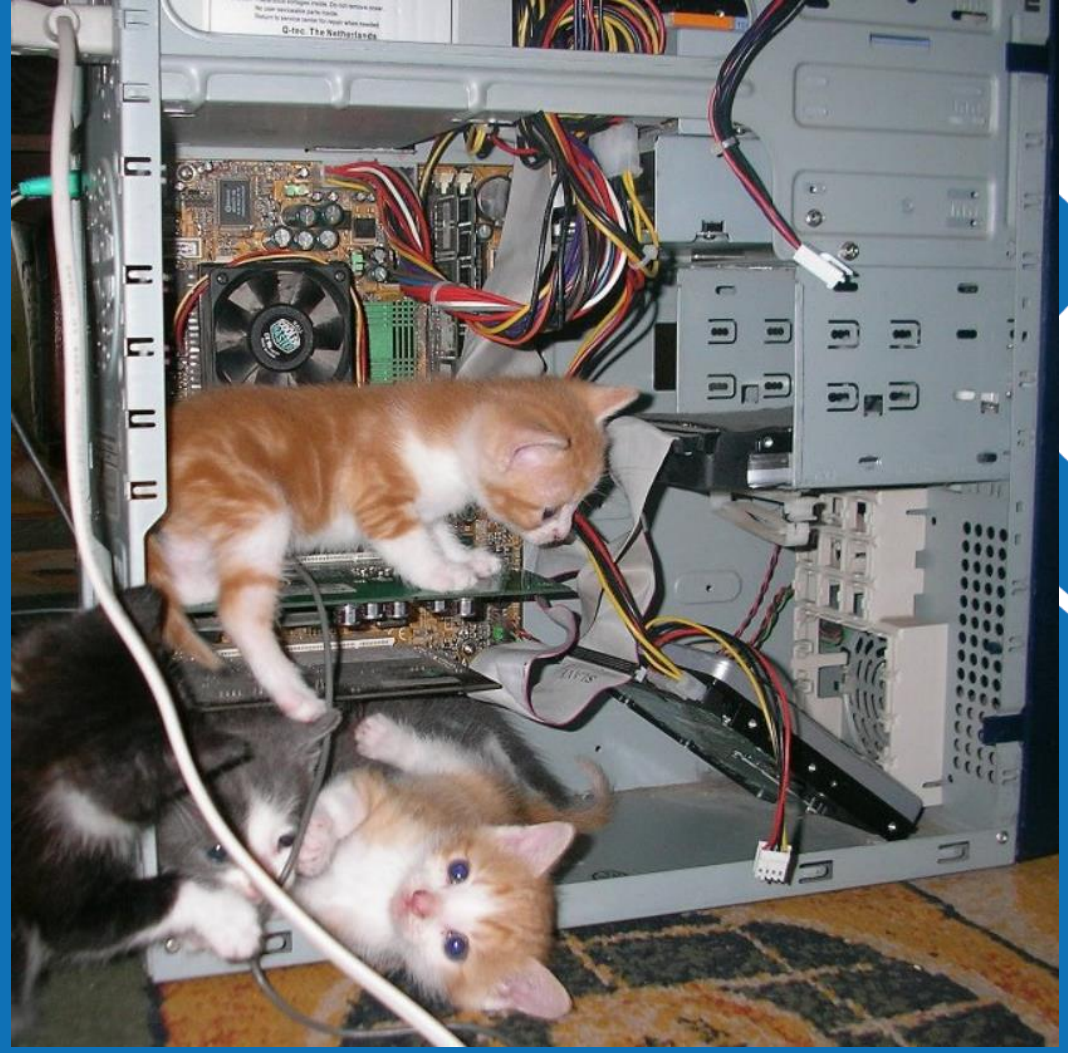


.NET DEVELOPER DAYS

.NET Memory Management

Kevin Gosse **@kookiz**
Christophe Nasarre **@chnasarre**

AGENDA

1. What is *memory*?

Let’s go back to the roots

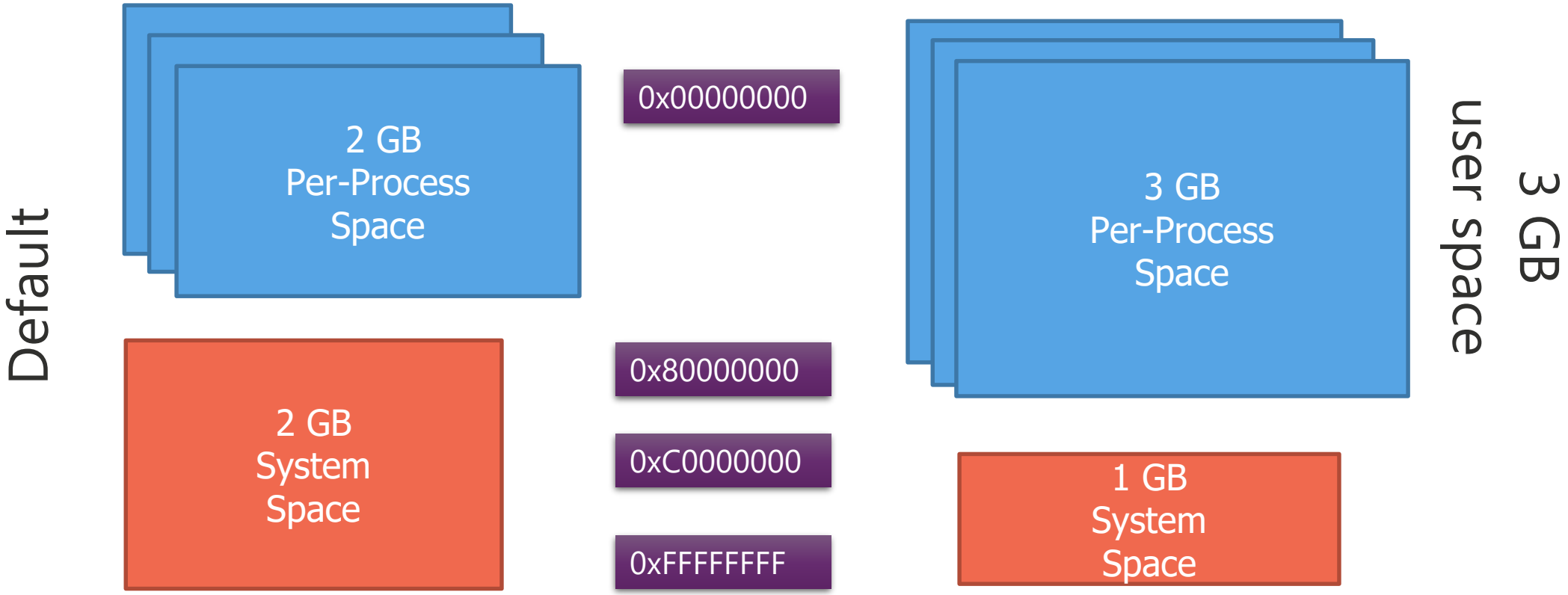
2. GC Internals

How the CLR is managing memory

criteo

Memory on Windows: *address space* in 32-bit

- 32-bits = 2^32 = 4 GB
  - /3GB and /USERVA can extend process address up to 3 GB
  - Process must be marked “large address space aware” to use memory above 2 GB

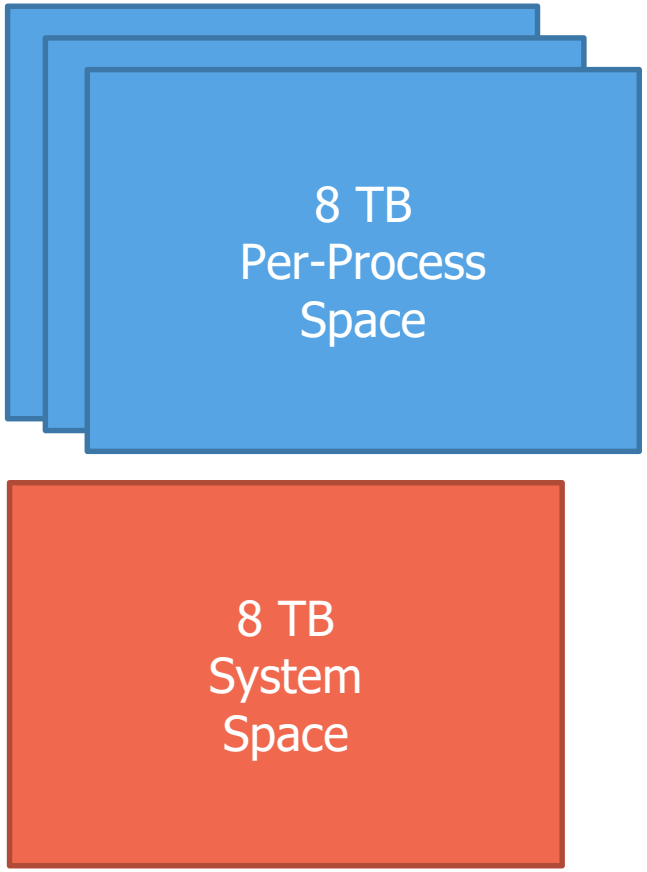


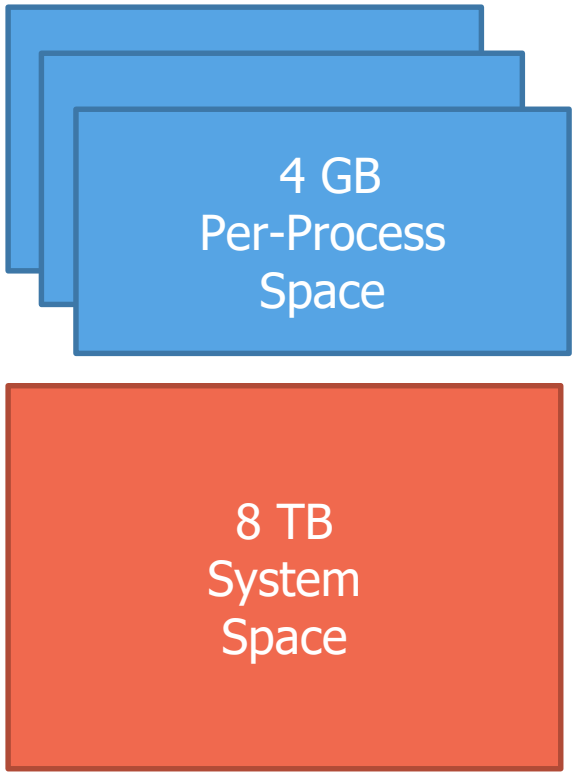
criteo

Memory on Windows: *address space* in 64-bit

- 64-bits = 2^64 = 17,179,869,184
  - But limited physically to 24 TB (Windows Server) and 6 TB (Windows 10)

x64
- AMD64
- Intel 64



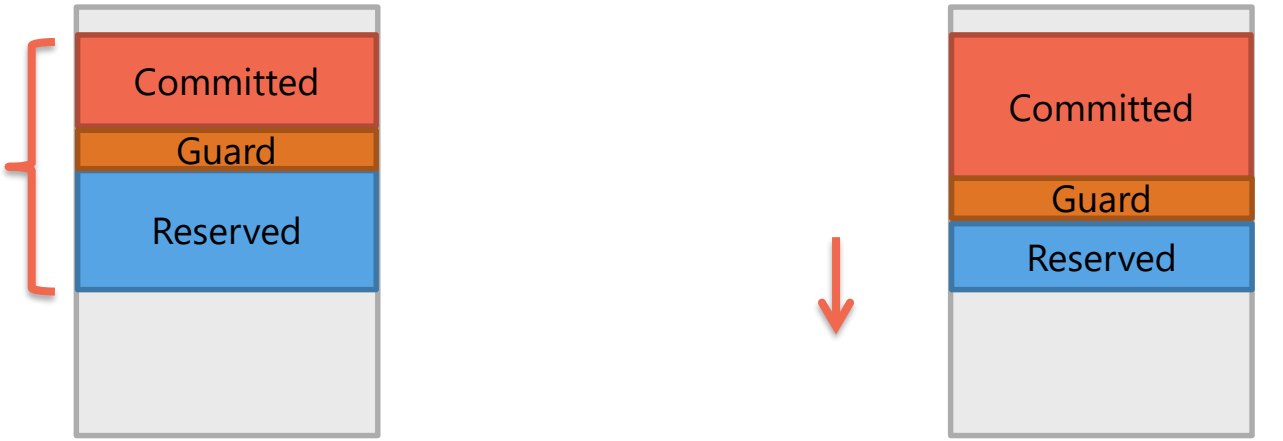


32-bit
process
on x64

criteo

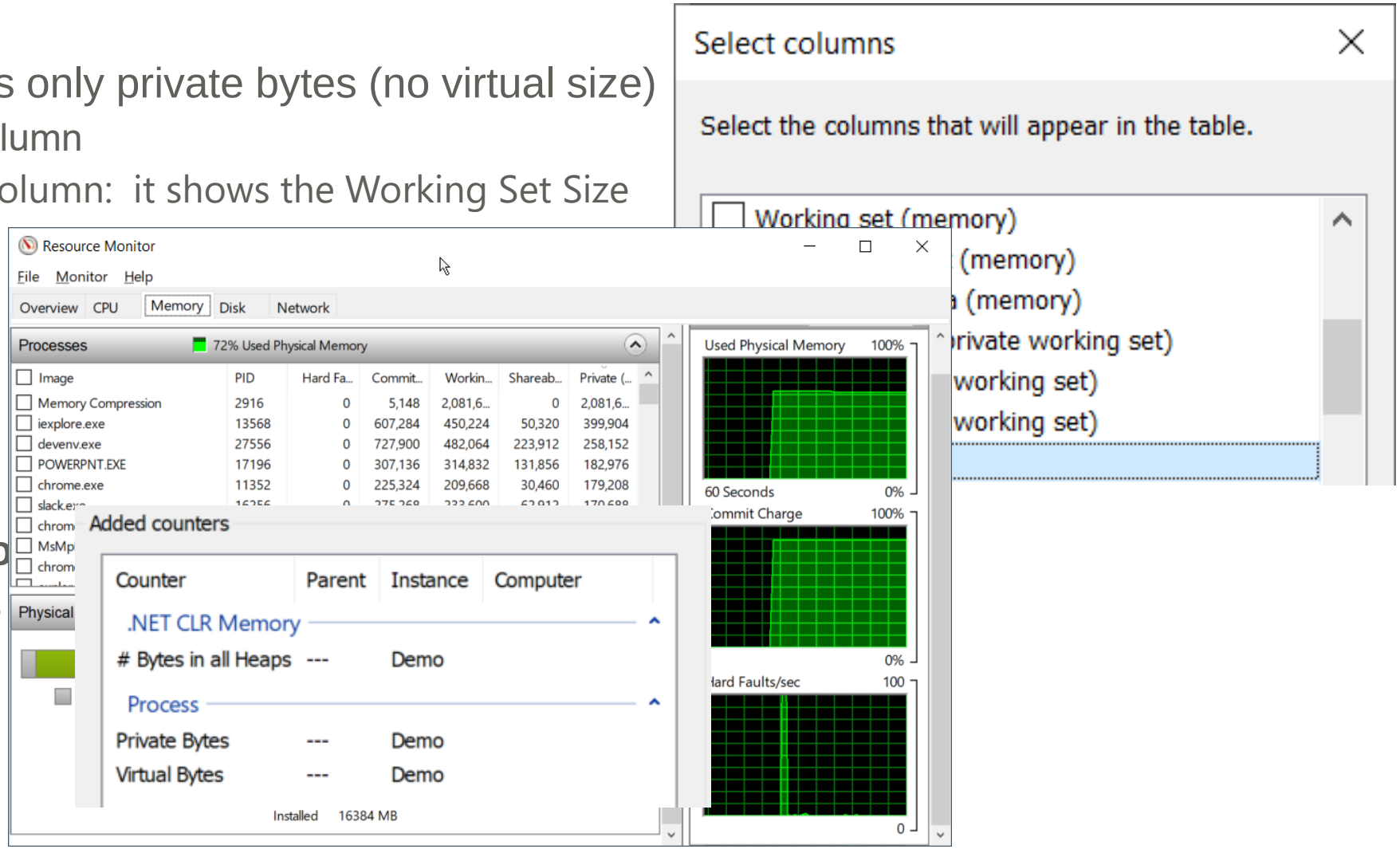
Different words to describe the notion of “memory”

- Windows protects memory (i.e. process isolation)
- Windows allows memory allocation in two steps
  1. *Reserve* address space
  2. *Commit* storage in that address space
- Reserved memory lets an application lazily commit **contiguous** memory (used for heap and stack expansion)



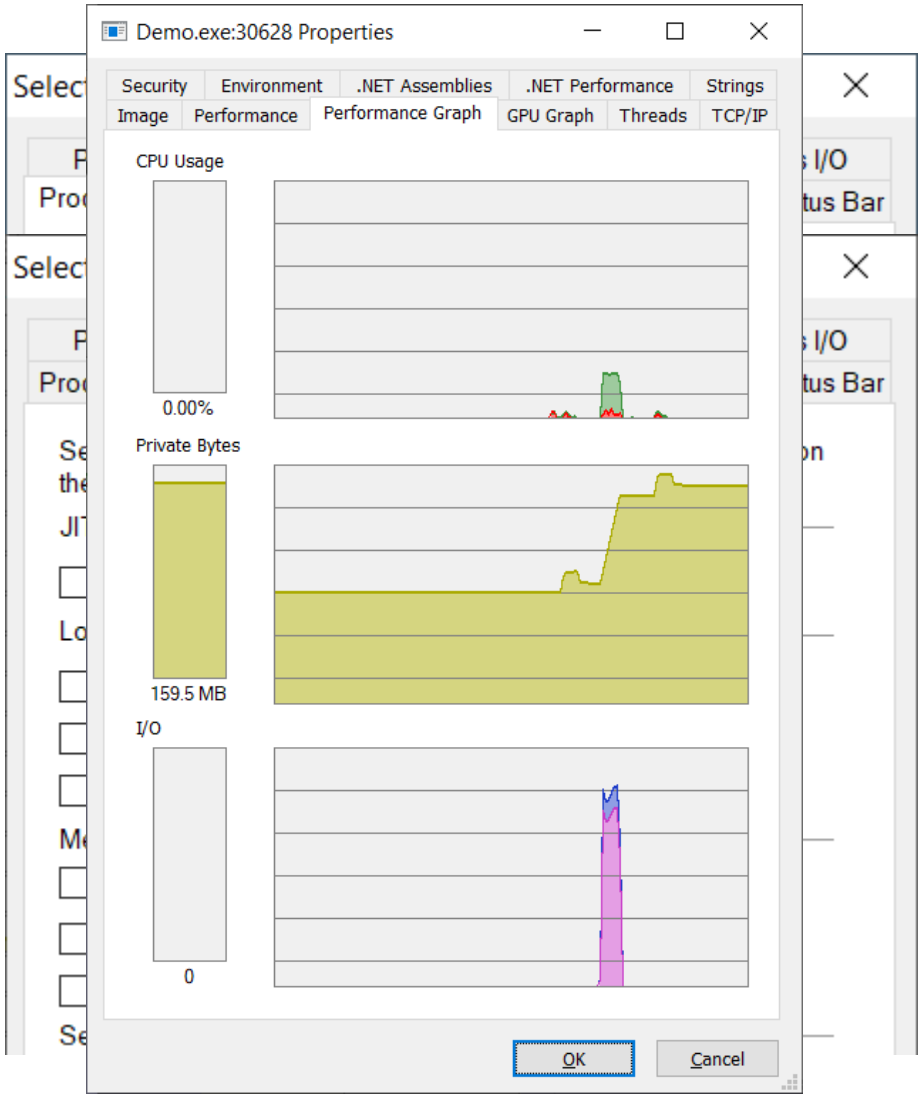
Which tool?

- **Task Manager** shows only private bytes (no virtual size)
  - Pick “Commit Size” column
  - Forget about “Mem” column: it shows the Working Set Size
- **Resource Monitor**
  - from Task Manager
  - “perfmon.exe /res”
- **Performance Monito**
  - Pick the right counters



… or just use **Process Explorer** from SysInternals

- “Native” view
- “Managed” view
- Global view



.NET Basics (1/2)

a reference = a pointer… that moves

- **Value type** = struct / enum
  - Allocated on stack or embedded inside a type
  - 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

criteo

.NET Basics (2/2)

Common Type System = everything is strongly typed

- **Value types**
  - Primitive `int i;`
  - Enums `enum State { Off, On }`
  - Structs `struct Point { int x, y; }`

- **Reference type**s
  - Classes `class Foo: Bar, Ifoo { … }`
  - Interfaces `interface IFoo: IBar { … }`
  - Arrays `string[] a = new string[10];`
  - Delegates `delegate void Empty();`

criteo

GC Internals (1/2)

Managed Heap: an history of generations 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

criteo

GC Internals (2/2)

Garbage Collection and hidden costs

- 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 (except the one doing the allocation)
- Non deterministic: might occur on any allocation
- Impact of types that implement a `~Finalize` method (more on this later on)

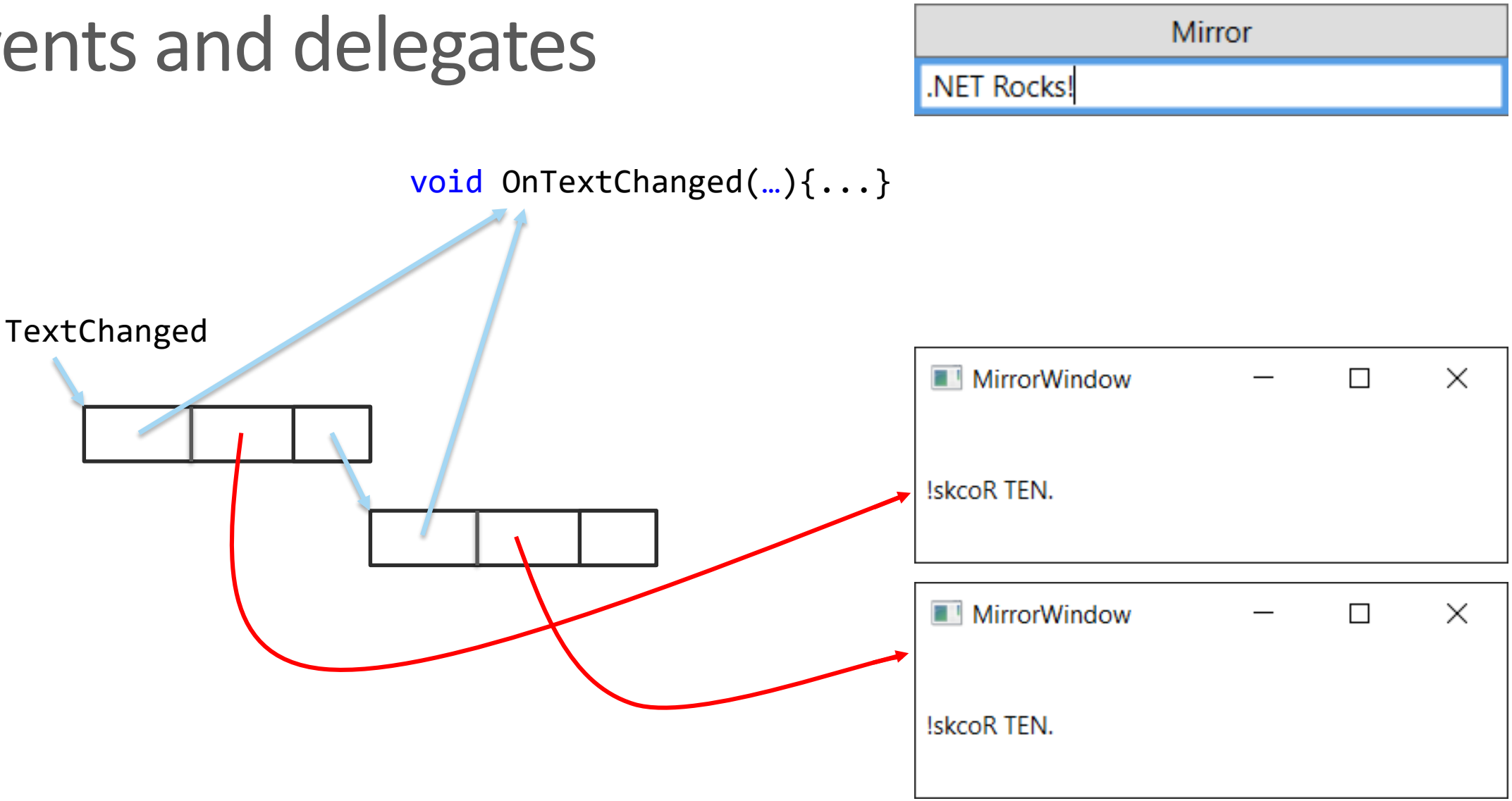
- Don't call `GC.Collect()`: could break the GC self-tuning algorithms
- Default sizes for segments could be too laaaaaarge (Server/Workstation GC)
  - https://devblogs.microsoft.com/dotnet/middle-ground-between-server-and-workstation-gc/
  - **DEMO**: show .NET memory segments with **VMMap** and **ClrMD Studio**

criteo

DEMO

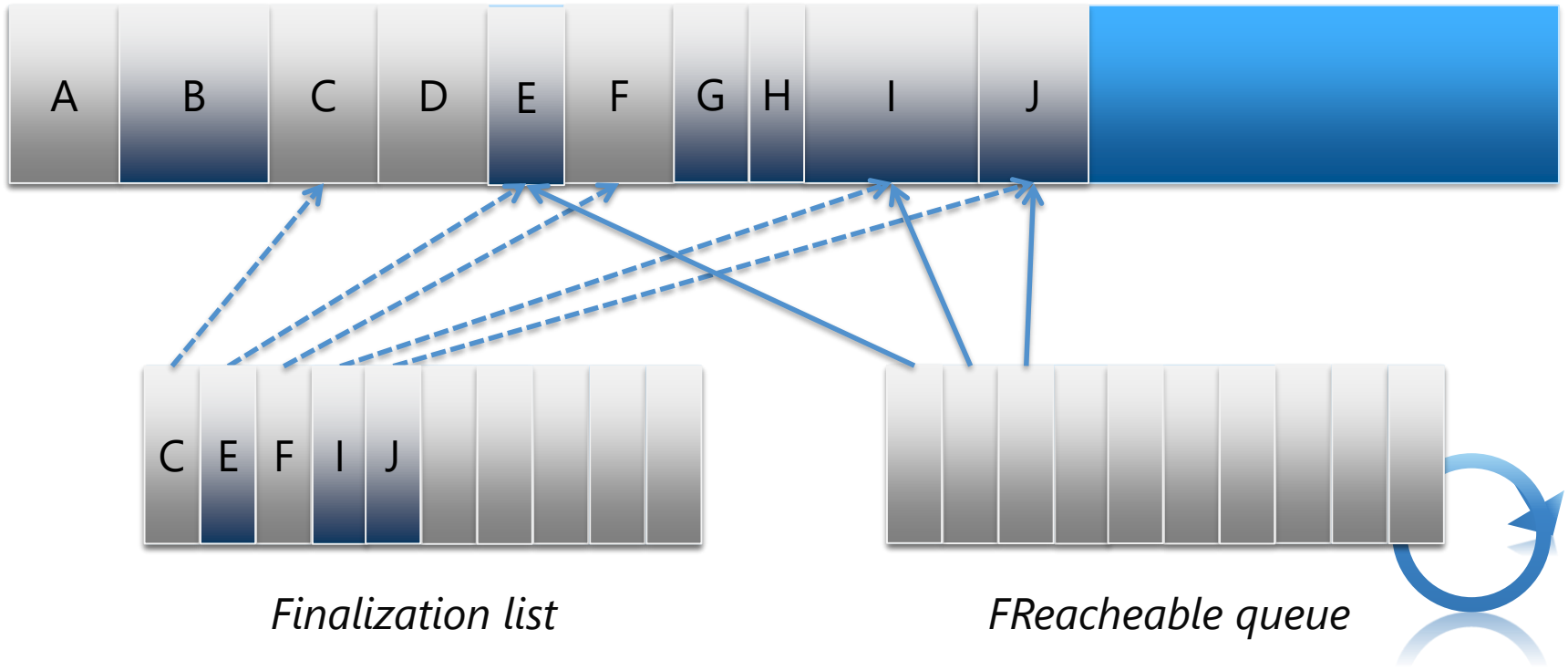
Chasing a memory leak

Events and delegates

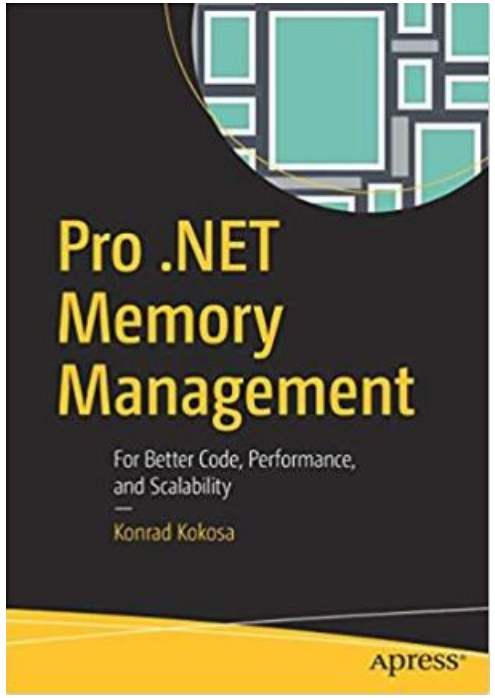


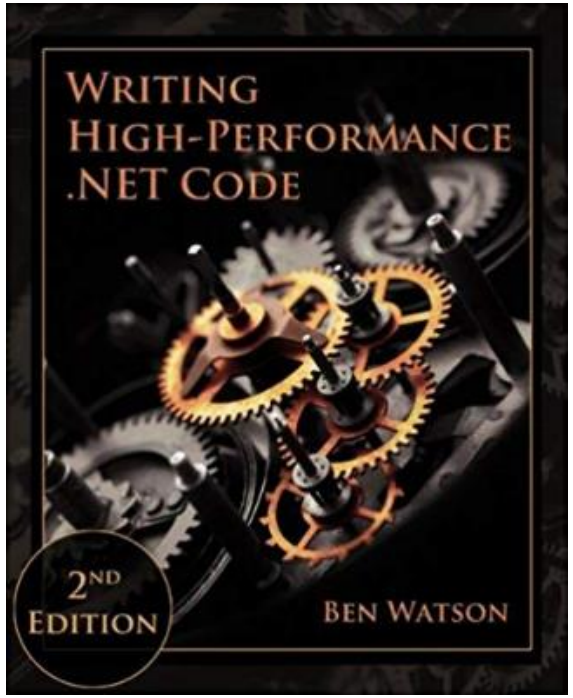
Finalizers and the Garbage Collector

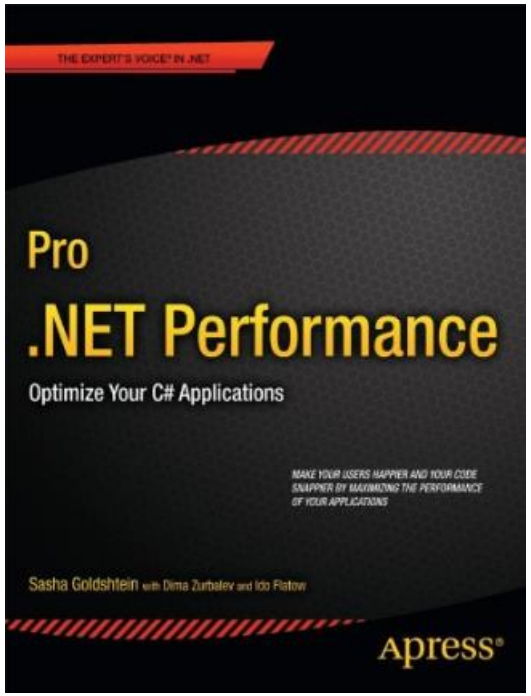
- What is the cost of a `~Finalize()` method?



Resources







Documentation

https://github.com/dotnet/coreclr/blob/master/Documentation/botr/garbage-collection.md

https://docs.microsoft.com/en-us/dotnet/standard/garbage-collection/fundamentals

DebuggingExtensions on github

https://github.com/chrisnas/DebuggingExtensions

Thank you.

criteo