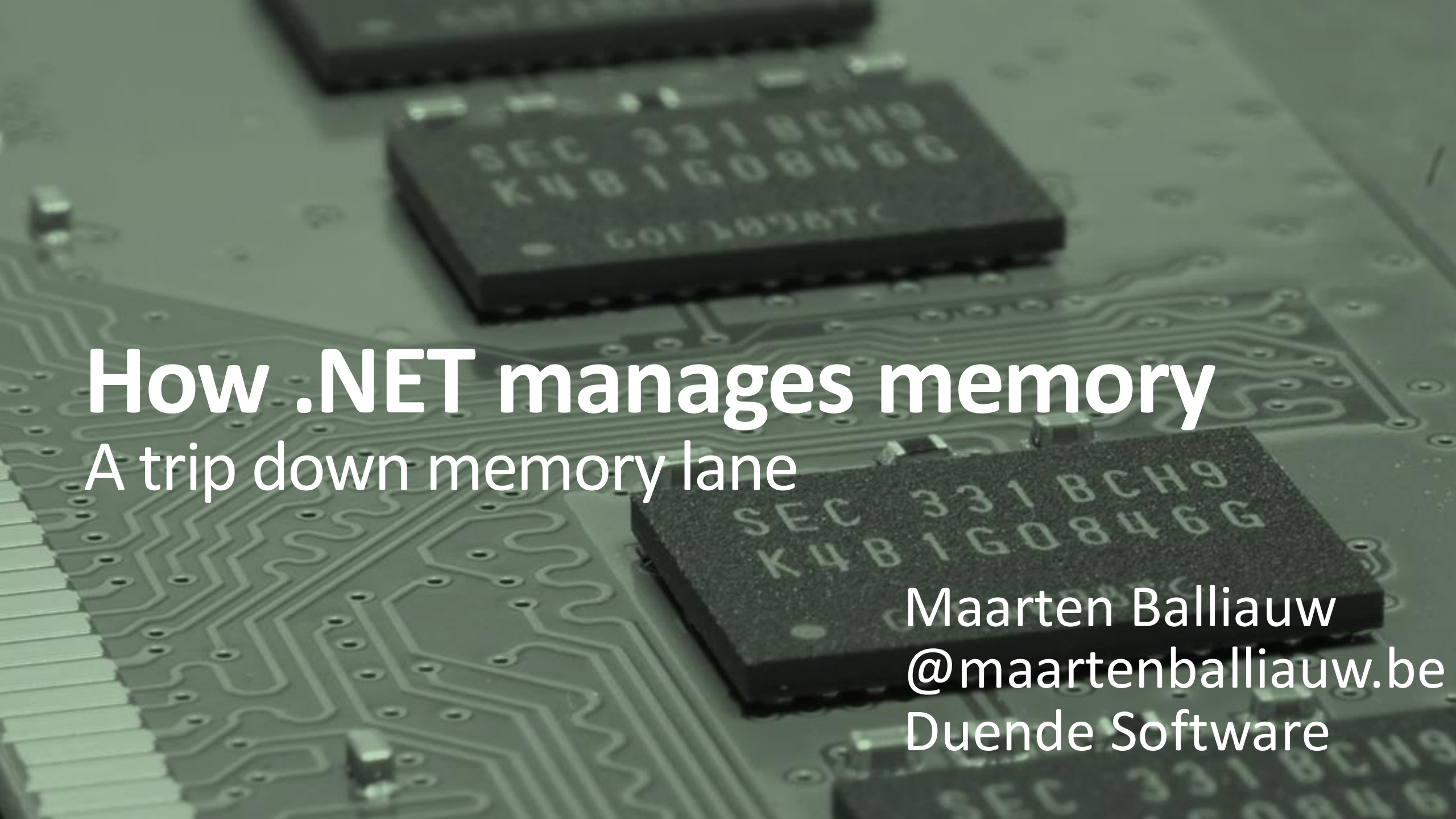


How .NET manages memory

A trip down memory lane

Maarten Balliauw
@maartenballiauw.be
Duende Software


Garbage Collector

The goal of managed memory

"Virtually unlimited memory for our applications"

.NET runtime manages pre-allocated memory
- Allocations happen in that heap
- Garbage Collector (GC) reclaims unused memory

.NET memory management 101

Memory allocation
- Objects allocated in “managed heap”
- **Cheap and fast**, it’s just adding a pointer
- If more space is needed, the heap is expanded

Memory release or “Garbage Collection” (GC)

Generations

Large Object Heap

Pinned Object Heap

Frozen Object Heap

.NET memory management 101

Memory allocation

Memory release or “Garbage Collection” (GC)

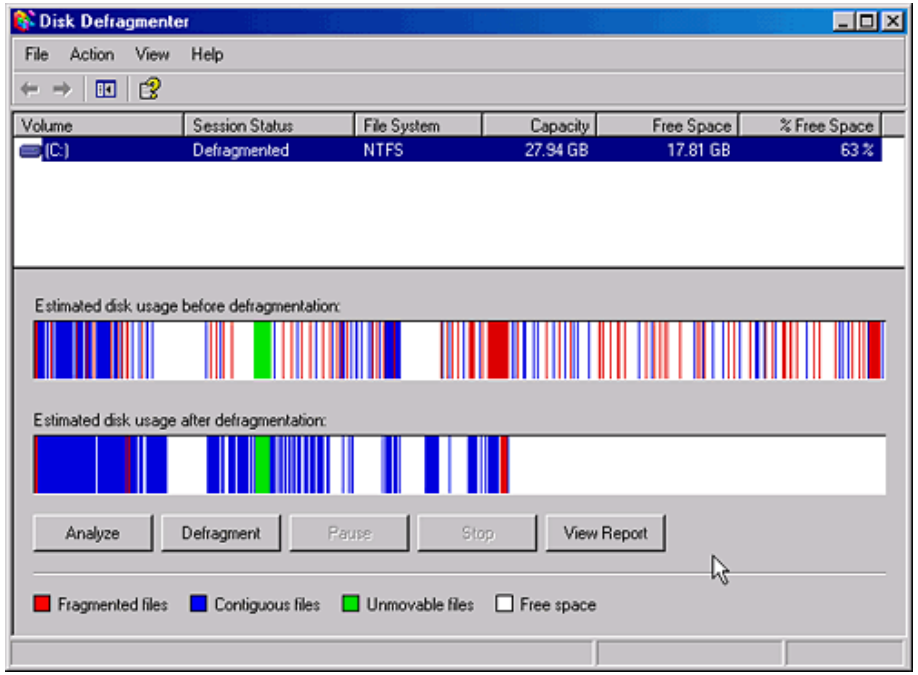
- GC releases memory that’s no longer in use
- Expensive!
  - *Pause application (not always the case)*
  - *Build a graph of objects*
  - *Remove unreachable objects*
  - *Compact memory*

Generations

Large Object Heap

Pinned Object Heap

Frozen Object Heap

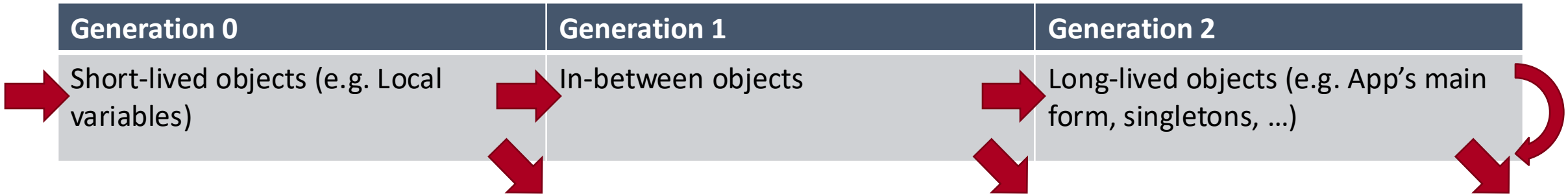


.NET memory management 101

Memory allocation

Memory release or “Garbage Collection” (GC)

Generations



Large Object Heap

Pinned Object Heap

Frozen Object Heap

.NET memory management 101

Memory allocation

Memory release or “Garbage Collection” (GC)

Generations

Large Object Heap (LOH)

- Large objects (>85KB)
- Collected only during full garbage collection
- Not compacted (by default)
- Fragmentation can cause *OutOfMemoryException*

Pinned Object Heap

Frozen Object Heap

.NET memory management 101

Memory allocation

Memory release or “Garbage Collection” (GC)

Generations

Large Object Heap

Pinned Object Heap (POH)

- Objects you know won’t need GC
- Tell the GC to skip an object (e.g. a pointer with PInvoke)
- Can complicate the work of the GC due to fragmentation, use them sparingly

Frozen Object Heap

.NET memory management 101

Memory allocation

Memory release or “Garbage Collection” (GC)

Generations

Large Object Heap

Pinned Object Heap

Frozen Object Heap (FOH)

- Immutable objects that will never need to be collected
- E.g. type info, string literals, reference data, …
- Recommended for very specific cases

The .NET garbage collector

- Runs often for gen 0, less often for higher generations
- Background GC (enabled by default)
  - Concurrent with application threads, pauses usually just for one thread
- When does it run?
  - Out of memory condition – when the system fails to allocate or re-allocate memory
  - After some significant allocation – threshold changes dynamically
  - Profiler, forced, internal events
  - Not fully predictable

https://raw.githubusercontent.com/dotnet/coreclr/master/src/gc/gc.cpp 😬

Throughput

More allocations, more garbage collections

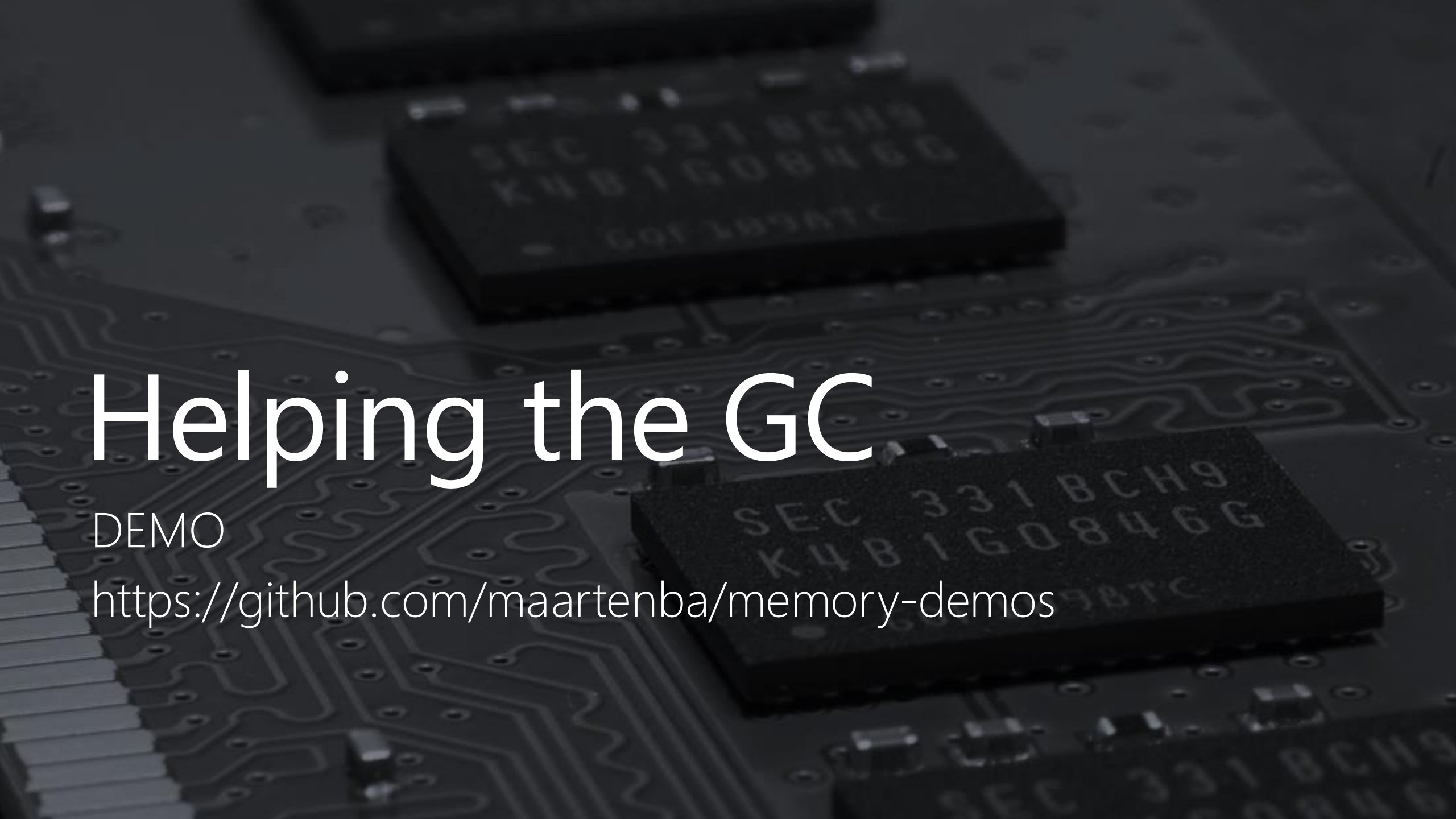
Garbage collection means pauses can happen

Pauses are bad – they may be experienced by your users!

- Mobile apps or in games (lag)
- Server (“resource sharing” with others)
- Trading
- …

Can we help the GC avoid pauses?

- Allocating is cheap, collecting is expensive
  - Use struct when it makes sense, `Span<T>`, `ValueTuple<T>`, object pooling, …
  - Make use of `IDisposable` / `using` statement & clean up manually
- Weak references
  - Allow the GC to always collect these objects, no need for checks
- Finalizers
  - Beware the finalizer queue!



Helping the GC

DEMO

https://github.com/maartenba/memory-demos

SLIDES TODO FROM HERE, DEMOS ARE CURRENT

ALSO CONSIDER SUDOKU APP


Allocations

Types

REFERENCE TYPES

`class`, `string`

Variable is a pointer to an object

Passed around by reference

Assignment copies the reference

Allocated on heap

GC involved, plenty of space

VALUE TYPES

`int`, `bool`, `struct`, `decimal`, `enum`, `float`, `byte`, `long`, …

Variable is the value

Passed around by value (copied)

Assignment copies the value

Allocated on stack

No GC involved, limited space

Hidden allocations!

Boxing

```
int i = 42;

// boxing - wraps the value type in an "object box"
// (allocating a System.Object)
object o = i;
```

Lambda's/closures

- Allocate compiler-generated `DisplayClass` to capture state

Params arrays (depending on compiler version)

Async/await

…

How to find them?

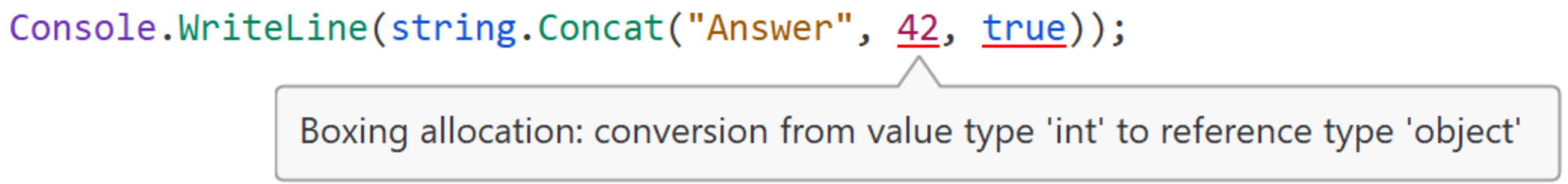
Intermediate Language (IL)

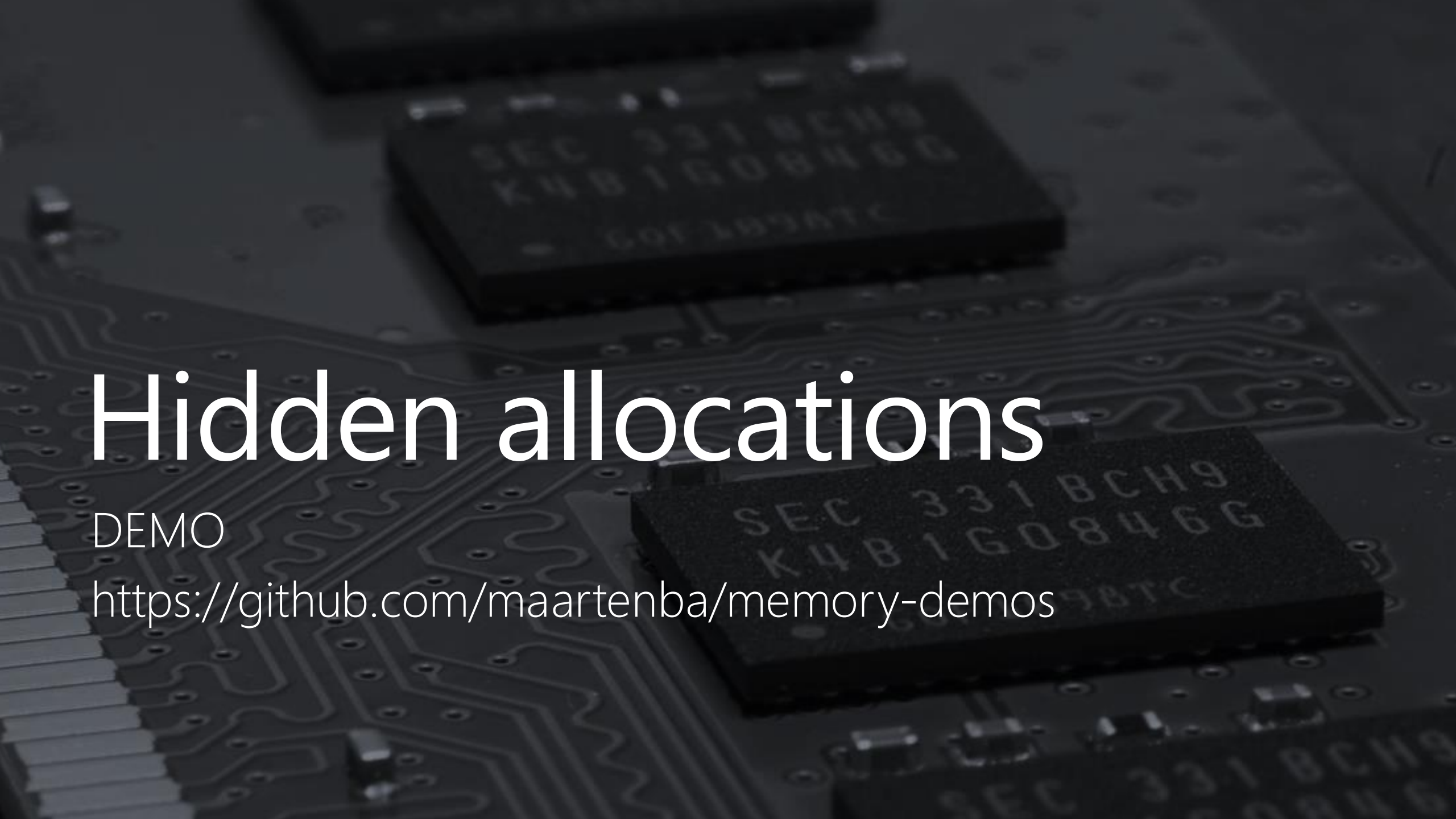
Profiler

“Heap allocations viewer”

[ReSharper Heap Allocations Viewer plugin]

[Roslyn’s Heap Allocation Analyzer]





Hidden allocations

DEMO

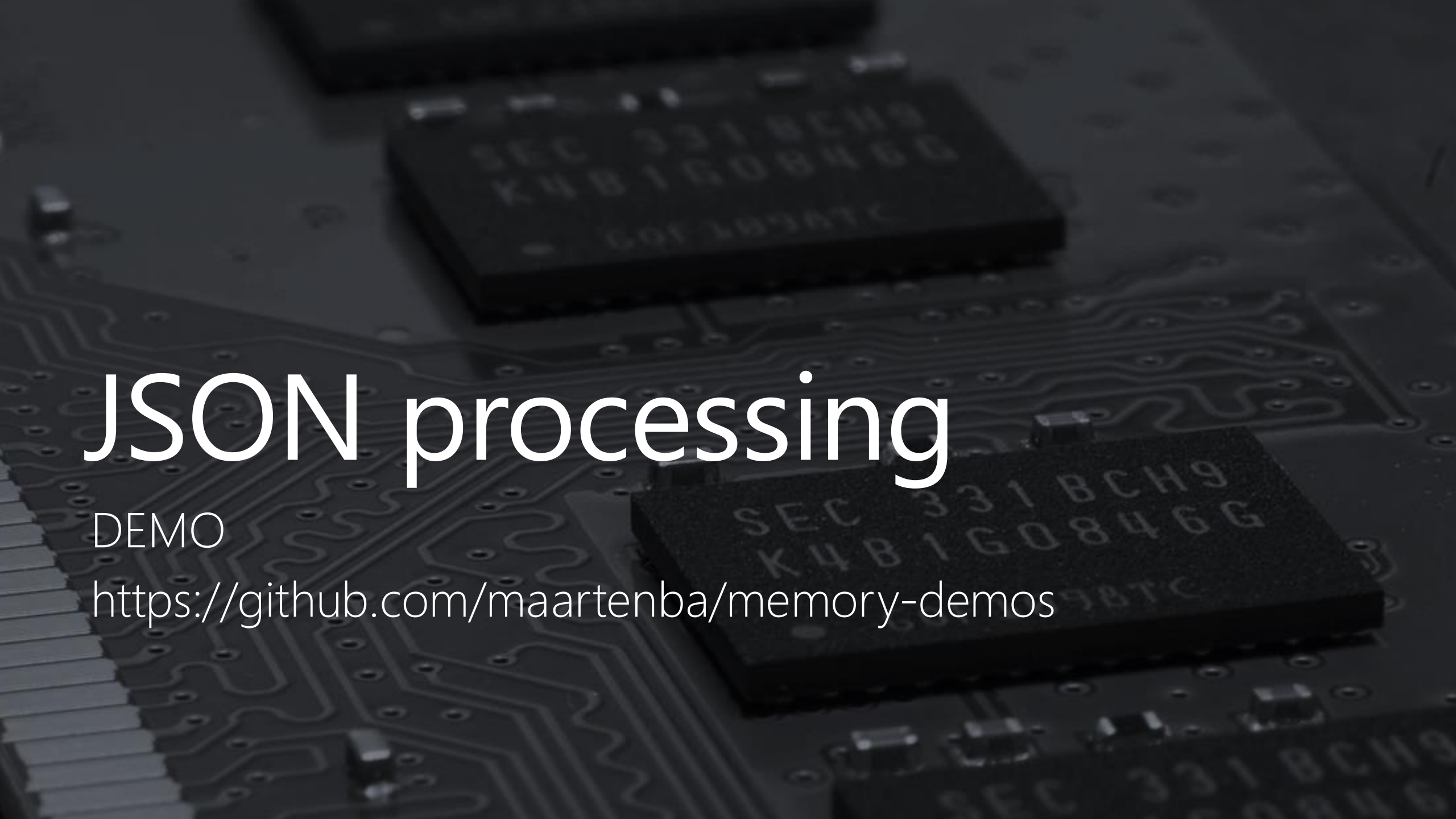
https://github.com/maartenba/memory-demos

Measure!

Don’t do premature optimization – measure!
- Allocations don’t always matter (that much)
- .NET has a lot of free performance gains in every release

Performance counters:
- How frequently are we allocating?
- How frequently are we collecting?
- What generation do we end up on?
- Are our allocations introducing pauses?

Use a profiler like dotMemory, dotTrace, …



JSON processing

DEMO

https://github.com/maartenba/memory-demos

09:48

Maarten Ballauw

TOTAL
2028

UNIQUE
1058

maartenba

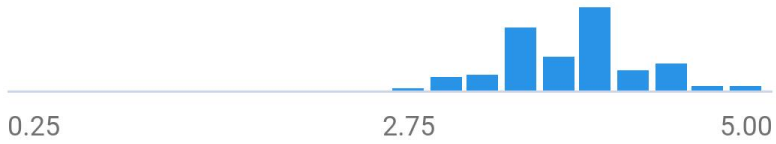
Joined Dec 9, 2013

PHOTOS >





RATINGS > 1,993 TOTAL



LISTS > NEW LIST

09:50

Beer

Westmalle Trappist Tripel
Brouwerij der Trappisten van W...
Belgian Tripel >
9.5% ABV • 36 IBU

FIND IT | ADD TO LIST

CHECK-IN

YOU >	69 Check-ins	5.00
FRIENDS >	37 rating	4.09
EVERYONE >	147k rating	3.87

HIGHLIGHTS

Belgian Beer Factory
Still one of the best Triple beers in the world
https://www.belgianbeerfactory.com/en/belgian-beer/per-brewery/trappist-westmalle/

```
[
  { ... },
  {
    "name": "Westmalle Tripel",
    "brewery": "Brouwerij der Trappisten van Westmalle",
    "votes": 17658,
    "rating": 4.7
  },
  { ... }
]
```

Object pools / object re-use

Re-use objects / collections (when it makes sense)

- Fewer allocations, fewer objects for the GC to clean
- Less memory traffic (maybe no need for a full GC)

Object pooling - object pool pattern

- `System.Buffers.ArrayPool / MemoryPool`
- `Microsoft.Extensions.ObjectPool`
- "Borrow objects that have been allocated before"

Garbage Collector summary

- Don’t fear allocations!
  - GC is optimized for high memory traffic in short-lived objects
  - Don’t optimize what should not be optimized…
- Know when allocations happen
  - GC is awesome
  - Gen2 collection that stop the world not so much…
  - Measure!

Exploring the heap

for fun and profit

How would you…

…build a **managed type system**, store in memory, **CPU/memory friendly**

Probably:

- Store type info (what's in there, what's the offset of fieldN, …)
- Store field data (just data)
- Store method pointers
- Inheritance information

Managed Heap

Instance Id	Pointer
0x40	0x30

Address	Value
0x1C	(sync block address)
0x30	0x60 (RTTI address)
0x..	33 (field 1 value)
0x..	Maarten (field N value)

Method Table Structure

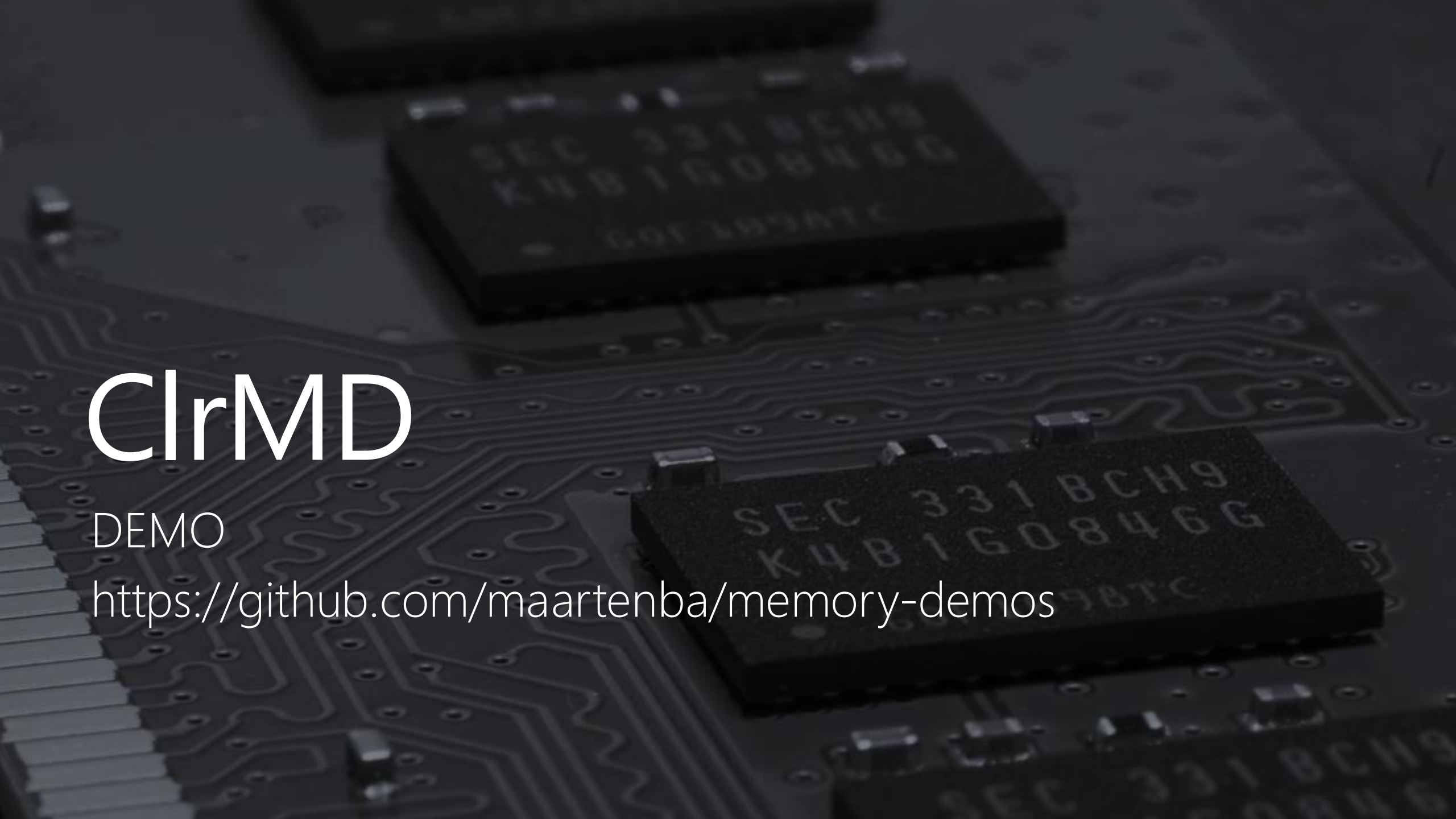
Address	Value
0x60	
0x6C	Interface Map Table Address
0x..	Inherited virtual method addresses
0x..	Introduced virtual method addresses
0x..	Instance method addresses
0x..	Static method addresses
0x..	Static field 1 value

(scroll down for more...)

“LINQ to Heap”

Microsoft.Diagnostics.Runtime (ClrMD)

“ClrMD is a set of advanced APIs for programmatically inspecting a crash dump of a .NET program much in the same way that the SOS Debugging Extensions (SOS) do. This allows you to write automated crash analysis for your applications as well as automate many common debugger tasks. In addition to reading crash dumps ClrMD also allows supports attaching to live processes.”



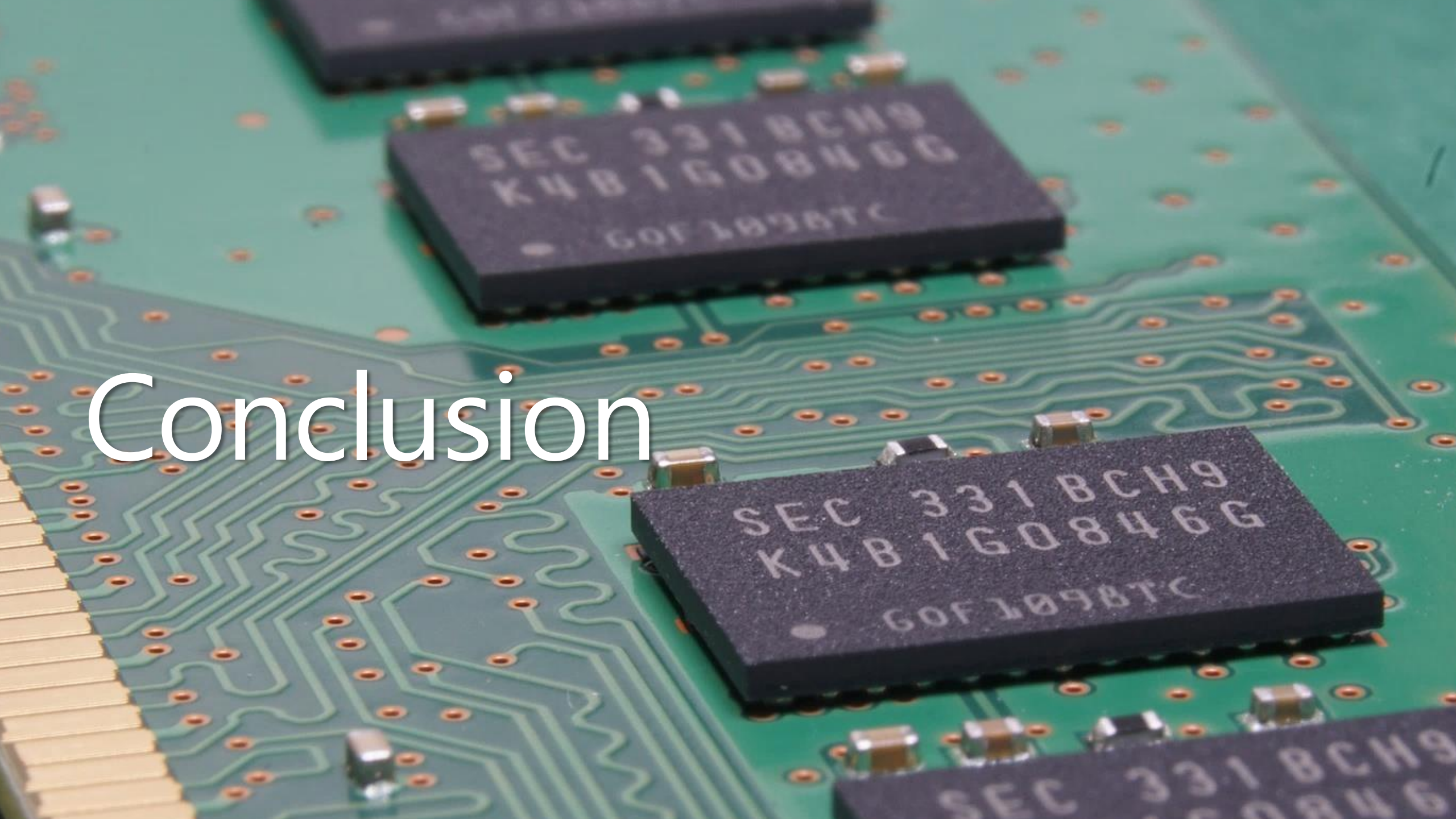
ClrMD

DEMO

https://github.com/maartenba/memory-demos

But... Why?

- Programmatic insight into memory space of a running project
  - Unit test critical paths and assert behavior (did we clean up what we expected?)
  - Capture memory issues in running applications
- Other (easier) options in this space
  - dotMemory Unit (JetBrains)
  - Benchmark.NET



Conclusion

Conclusion

Garbage Collector (GC) optimized for high memory traffic + short-lived objects

Don't fear allocations! But beware of gen2 "stop the world"

Don't optimize what should not be optimized…

- Measure!
  - *Using a profiler/memory analysis tool*
  - *ClrMD to automate inspections*
  - *dotMemory Unit, Benchmark.NET, … to profile unit tests*

Blog series: https://blog.maartenballiauw.be

Thank you!

Maarten Balliauw
@maartenballiauw.be
Duende Software