

Robert Haken

software & cloud architect, HAVIT, s.r.o.

haken@havit.cz, @RobertHaken, <http://knowledge-base.havit.cz>

.NET

Memory Internals

Architektura paměti aplikací

instrukční paměť

zásobník (stack)

halda (heap)

- Managed Heap
- Native Heap



Zásobník

Stack



Zásobník (Stack)

per thread

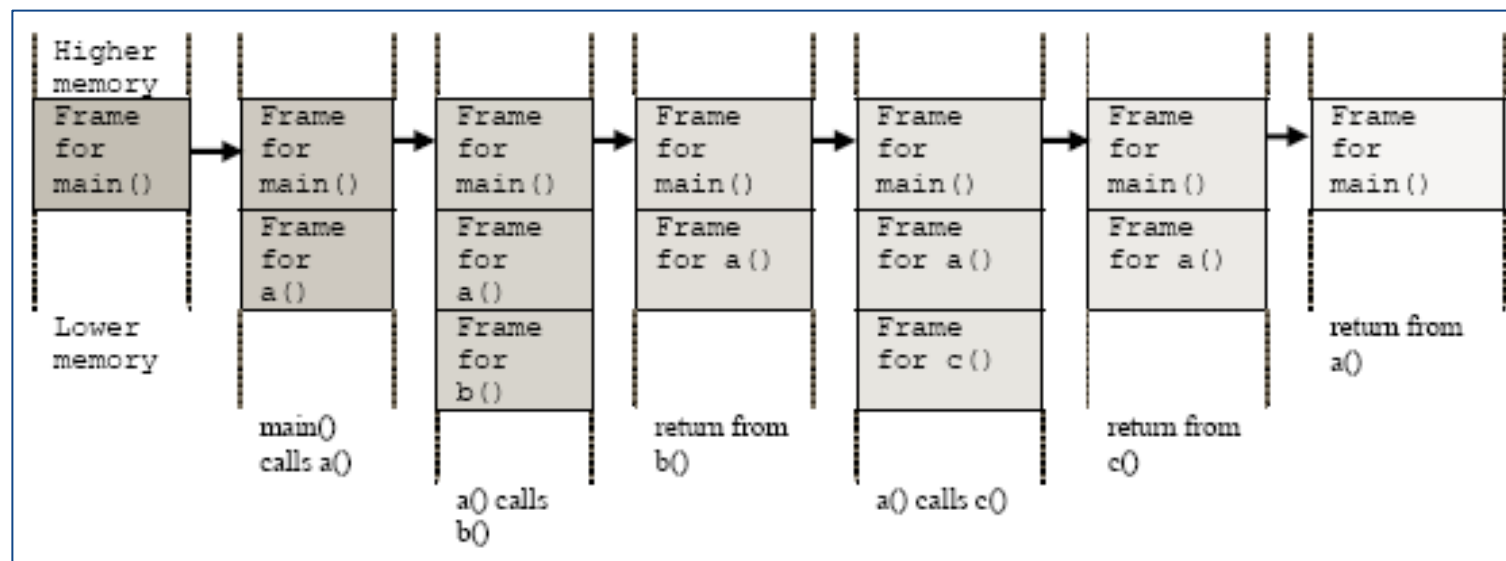
fixed size (konfigurovatelný, default 1 MB)

LIFO – Last In, First Out

Activation Records (Stack Frames)

- parametry (argumenty)
- návratová adresa, registry CPU, ...
- lokální proměnné

```
void main()  
{  
    a();  
}  
  
void a()  
{  
    b();  
    c();  
}
```



Stack Frames, Hodnotové/datové typy (01-StackHeap)

DEMO



Datové typy

Hodnotové

in-place hodnota

primitivní typy

Int16/32/64, Byte, Boolean, Double, ...

char, Decimal

struct - DateTime, Nullable<T>, vlastní,

...

Referenční

„hodnotou“ je odkaz na instanci (tj. adresa)

class (vč. Object)

pole – Array, type[]

string

na jednu instanci může ukazovat více referencí



Záludnosti

`string` je referenční datový typ

- ale je immutable!
- změna => nová instance
- instance se sdílí, dokud se nemění

`Array` je v .NET referenční datový typ

- ...a taky se tak chová!
- paměť je alokována při vzniku instance, ne při deklaraci proměnné



Operace přiřazení (=)

vždy zkopírování paměťové buňky

hodnotový typ → zkopírování vlastní hodnoty

referenční typ → zkopírování odkazu na instanci



Předávání parametrů do metod

předání parametru „hodnotou“ (default)

→ zkopírování paměťové buňky

předání parametru „referencí“ (ref, out)

→ odkaz na zdrojovou paměťovou buňku (stack)

návratová hodnota metody

→ zkopírování paměťové buňky (~ přiřazení)



Předávání parametrů do metod (02-CallingMethods)

DEMO



Halda

Heap



Halda (Managed Heap)

Garbage Collector ~ 1959 LISP

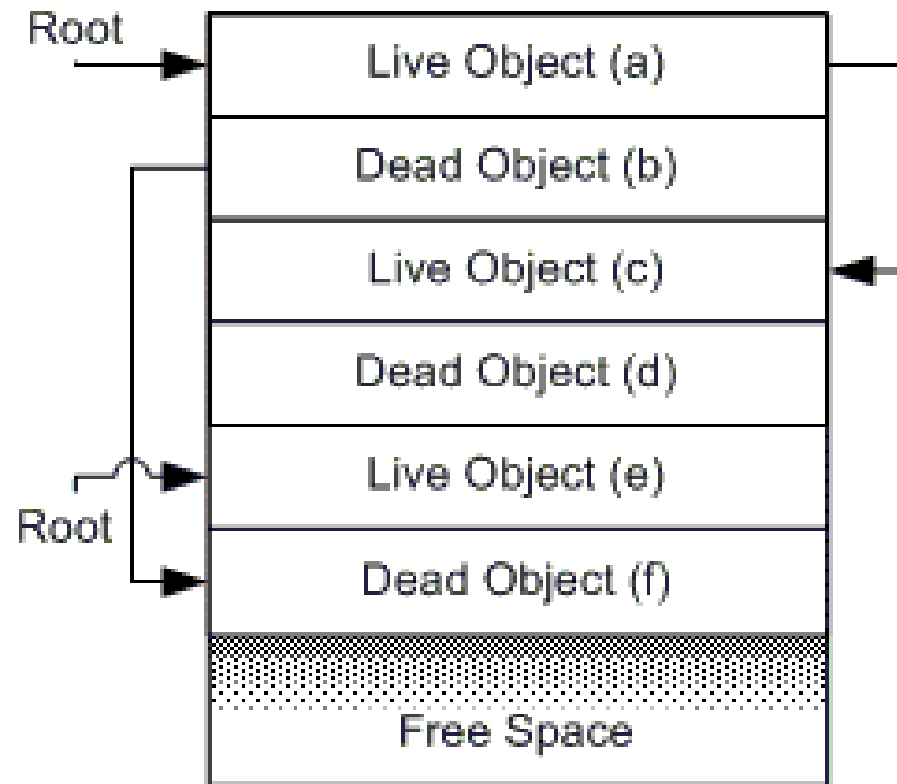
Garbage Collection

- překročení thresholdu generace při alokaci
- system-wide memory pressure
- GC.Collect() API (never-ever!)

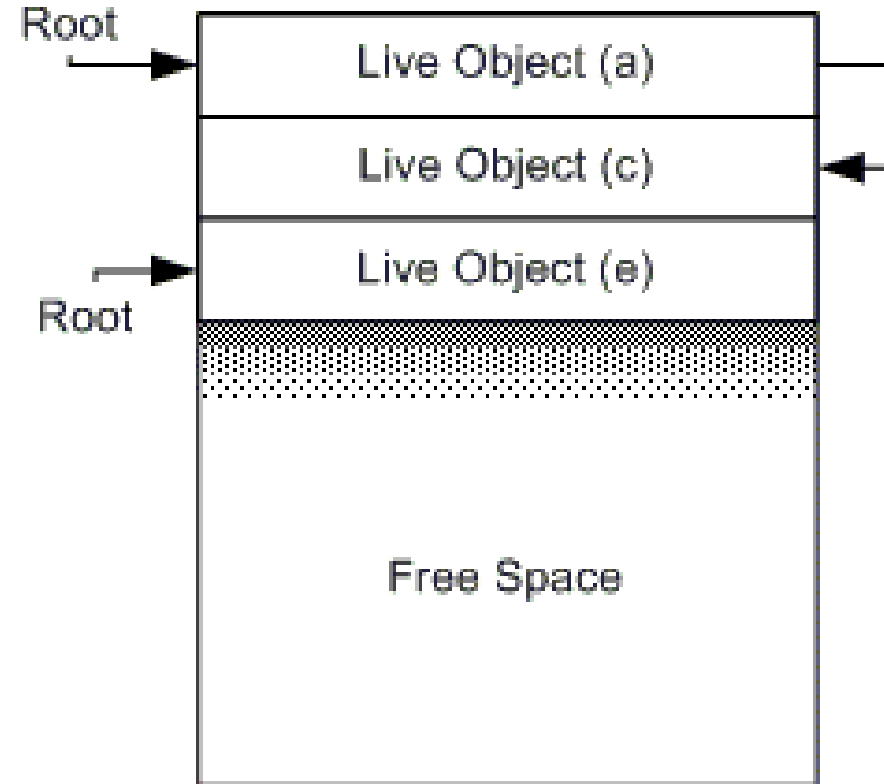
sesypání, update referencí



Before

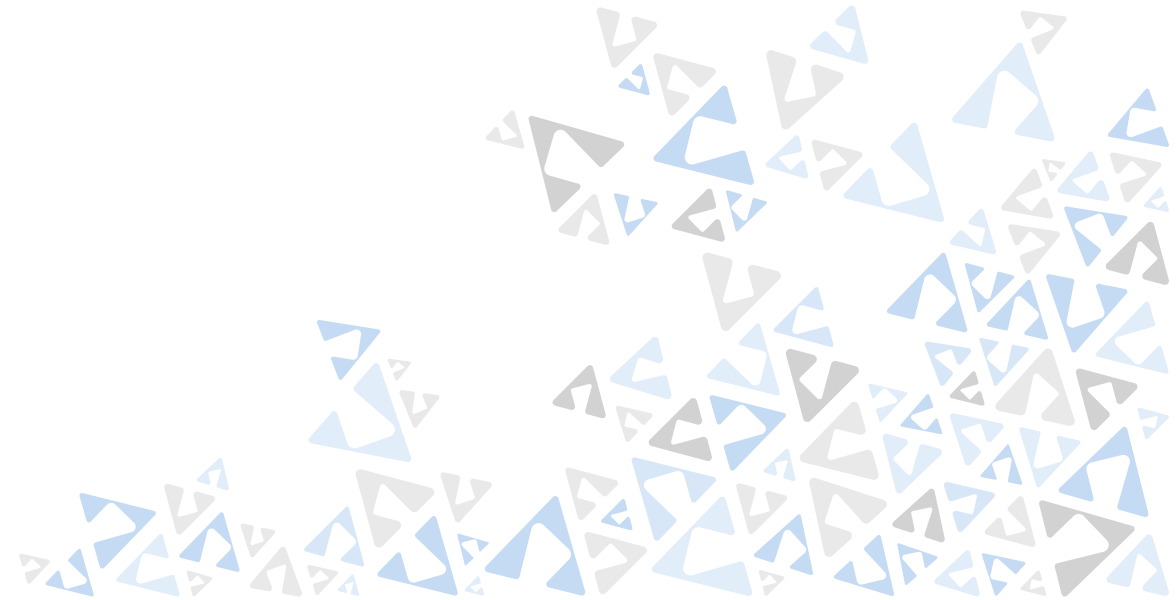


After



Garbage Collection (03-SimpleGarbageCollection)

DEMO



Roots

kořeny grafu objektů pro zjištění dosažitelnosti

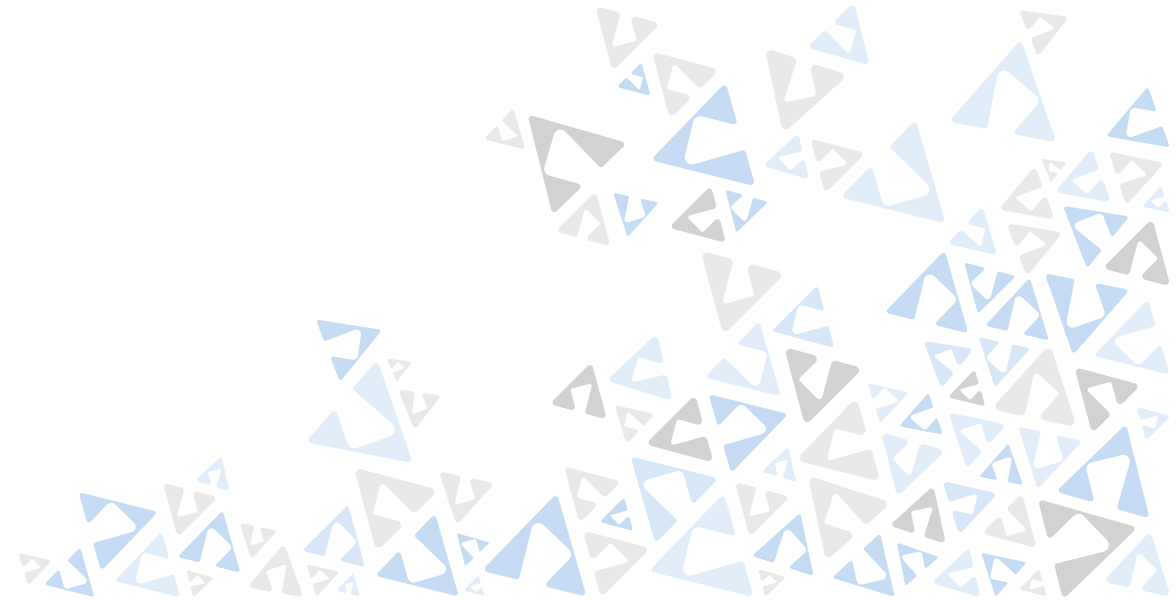
- zásobník (lokální proměnné, parametry metod)
- registry procesoru
- GCHandles
 - globální statické fieldy
 - pinned objects
 - ...
- F-reachable queue

BONUS: WeakReference



GC Roots (04-GCRoots)

DEMO



Generations

výkonová optimalizace

soustředí se objekty s krátkou životností

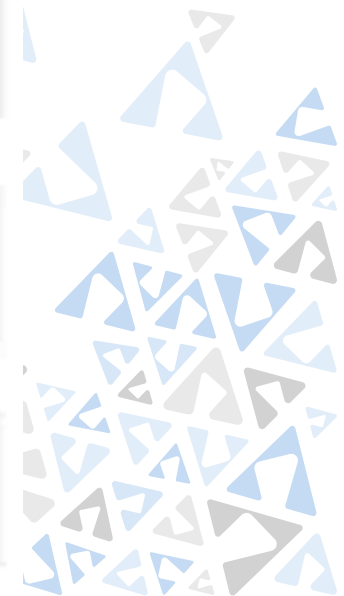
Gen 0 – nové, Gen 1 & 2 – přežily 1/více-krát

Gen 0 + 1 = ephemeral segment („fixed“ size)

Gen 2 = variable size

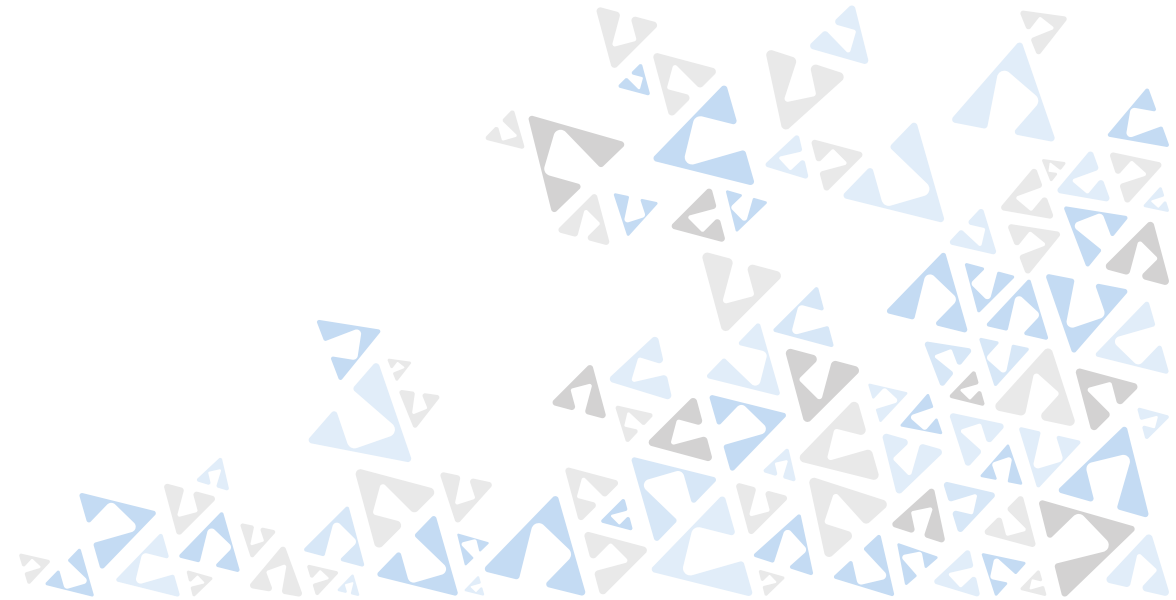


Generations



Generations (05-Generations)

DEMO



Finalization

explicitní úklid unmanaged zdrojů

Finalize() ~ C# destructor

Finalization Queue => vždy Gen 1

F-reachable Queue

Finalization Thread

IDisposable, ResourceWrapper pattern



Large Object Heap

výkonová optimalizace

objekty větší než 85 000 bytů (default)

součást Gen 2 collection

nesetřásá se (lze vynutit jednorázově)

Free List



xPlat - Linux, ...

```
dotnet tool install --global dotnet-dump  
dotnet-dump collect  
dotnet-dump analyze <dump_file.dmp>
```



REFERENCES

Demos

<https://github.com/hakenr/NetCoreMemoryInternals>

<https://github.com/hakenr/MemoryAndTypes>

<https://github.com/hakenr/GarbageCollectorDemo>



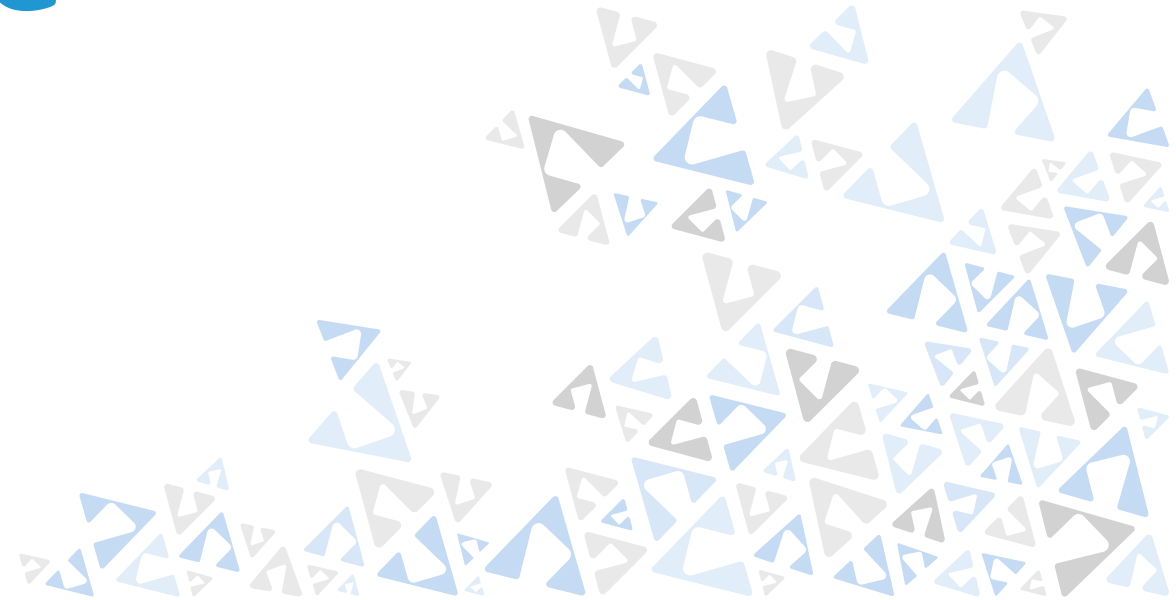
Blog – HAVIT Knowledge Base- <http://knowledge-base.havit.cz/> + .eu

Twitter - [@RobertHaken](https://twitter.com/RobertHaken)

YouTube - <https://www.youtube.com/user/HAVITcz>



WinDbg Reference



Debugger Extensions pro .NET

`.load C:\path\to\extension.dll`

SOS.dll – Son of Strike, součást .NET

`.loadby sos mscorwks` (.NET < 4)

`.loadby sos clr` (.NET >= 4)

`.loadby sos coreclr` (.NET Core)

PSSCOR2/PSSCOR4 – širší SOS (web)

SOSEX, NetEx – 3rd party

`!help [<command>]`



Záludnosti použití Debuggeru

Platform - x86 vs. x64

Symbols

`.symfix` (MSFT Symbols Server), `.sympath`, `.sympath+`
`.reload`

.NET Data Access Layer (mscordacwks.dll)

`.cordll -ve -u -l`

stejná verze, jako na laděném stroji (dtto SOS)



Režimy práce s Debuggerem

Open Executable... (**g** pro Run)

Attach to a Process...

Open Crash Dump...

- Task Manager / Create Dump File (!!32-bit vs. x64 stroj)
- DebugDiag / ADPLUS
- SysInternals PROCDUMP.EXE
- Windows Error Reporting
- Windows Crash Dump
- WIN32 API (extern v .NET)



DebugDiag

- „user“-friendly UI
- připravené analýzy
- sběr dat/dumpů
- pod pokličkou debugger services
- voláno např. i z Azure Web Apps KUDU



Stack Examination

`!ClrStack [-i] [-a] [-l] [-p]`

`!DumpStack [-EE]`

`!EEStack [-EE] (all threads)`

`!DumpStackObjects (typy)`



Heap Examination

!DumpHeap [-stat] [-type <name>]
 [-mt <MTaddr>] [-live|dead]

!HeapStat [-inclUnrooted]

!GCRoot <ObjAddr> !GCHandles

!EEHeap -gc

!FinalizationQueue [-allReady]

!FindAppDomain <ObjAddr>



Object Inspection

!DumpObject <ObjAddr>

!DumpArray <ObjAddr>

!DumpVC <MTAddr> <ObjAddr>

dd <addr>

dq <addr>

!ObjSize <ObjAddr>



Error Diagnostics

`!PrintException [ObjAddr] [-nested]`

`!DumpAllExceptions` (PSSCOR4)

`!wdae` (NETEXT)

`!wpe` (NETEXT)

`!VerifyHeap`

`!VerifyObj <ObjAddr>`

`!analyze -v` (native)



Threads

!Threads

~123s

!ThreadPool

!ThreadState <state>

