Aus Erfahrung lernen:

„MemoryLeaks“ in C#
oder
Wenn Objekte ewig leben

27.10.2015

Dietmar Mayer

EMail: mayer@smartdevelop.net
Website: http://www.smartdevelop.net

Über mich

- Über 30 Jahre Software-Entwicklung

- Branche: Technische Software
  - CAD, Stahlbau (rzindustriebau, LOGOCAD)
  - Gebäudetechnik (Trimble / Plancal Nova, BIM, technische Gebäudeberechnungen)
  - Prozesstechnik (COMOS)

Über mich: Erfahrungen / Umfeld

- Klassisch: DOS + Windows, C++
  - MFC, COM, ODBC, DAO, …
- seit 2007 C# / Managed .NET
  - UI: WPF, XAML, Silverlight
  - .NET-Interop: Win32-MFC / Managed-.NET
  - Retten von bestehendem Code in die .NET-Welt

Motivation für die Leak-Suche

- Kontinuierlich steigender Speicherverbrauch während der Programmlaufzeit

- → Projekt:(Mem-) Leaks finden und beheben

- → Präsentation für KollegenInnen:
  - Wie geht man da ran ?
  - Gefundene Leaks aufzeigen, um sie zukünftig von Anfang an zu vermeiden.

Thema heute:

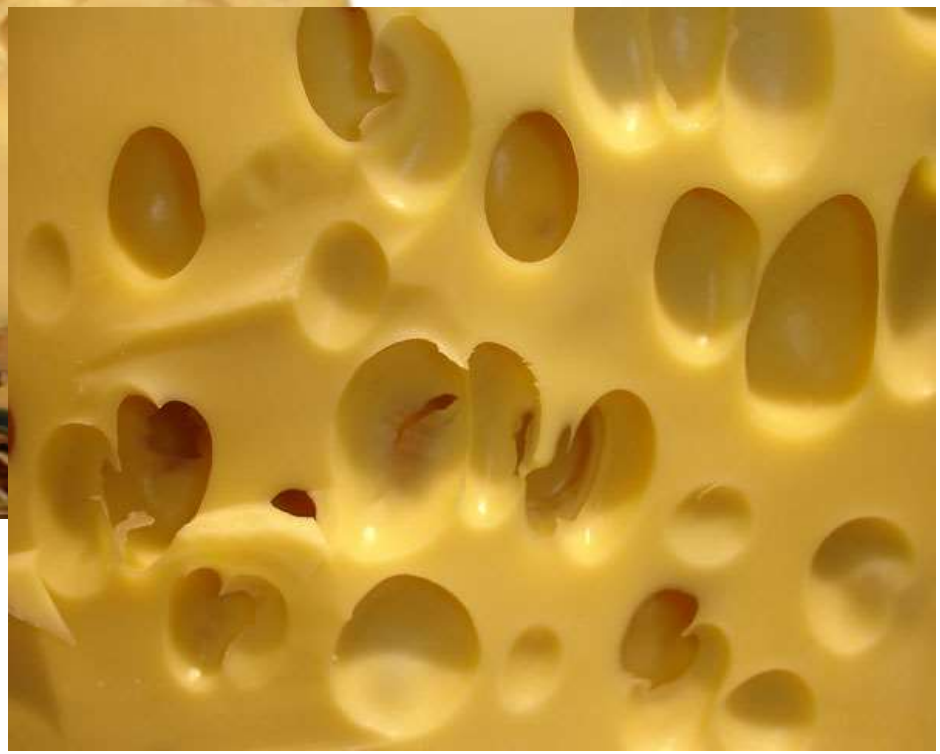
- Wie findet man Leaks
  - Vorgehen
  - Tool-Unterstützung im .NET Managed Umfeld
- Beispiele für Leaks aus der Praxis
  - Closures
  - EventHandler

memory leak – Was ist das ?

- **Speicherleck** (englisch *memory leak*, gelegentlich auch *Speicherloch* oder kurz *memleak*)
- bezeichnet einen Fehler in der Speicherverwaltung eines Computerprogramms, der dazu führt, dass es einen Teil des Arbeitsspeichers zwar belegt, diesen jedoch weder freigibt noch nutzt.

http://de.wikipedia.org/wiki/Speicherleck

leaks ... . anschaulich



http://media.kuechengoetter.de/media/38/12072983546380/edamer2.jpg
http://media.agrarheute.com/05/416105.jpg
http://www.epicurus.com/Glossary/wp-content/uploads/2011/07/emmental.jpg

leaks finden – genereller Ablauf (1)

- **Reproduzierbaren Ablauf für einen Speicher- / Resourcenverlust finden.**

- Programm neu starten.

- Einmal Ablauf vollständig durchführen, damit Dlls, Daten etc. passend geladen sind.

leaks finden – genereller Ablauf (2)

- **1. Messpunkt erstellen**

- Ablauf zur Reproduktion 1 x vollständig durchlaufen.
  - z.B. Projekt aus IFC-File importieren und Projekt wieder schließen

- **2. Messpunkt erstellen**

leaks finden – genereller Ablauf (3)

Differenzen zwischen Messpunkt 1 und Messpunkt 2 analysieren

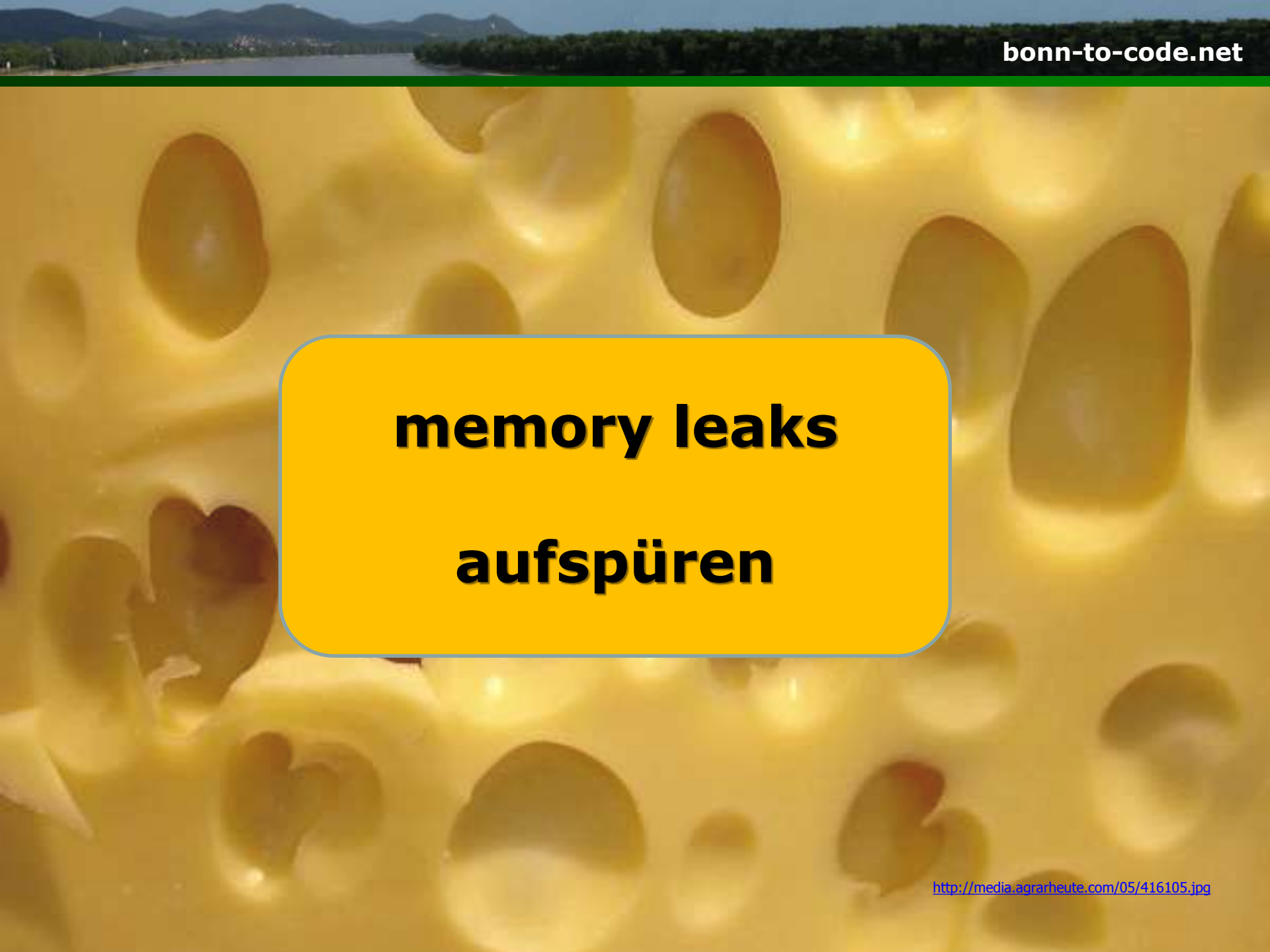
leaks finden – genereller Ablauf (4)

DeltaState (Messpunkt2 – Messpunkt1)

- Welche Objekte sind jetzt neu hinzugekommen ?
- Ist das OK? gewünscht? => **kein Leak**

- **Warum wurden sie nicht freigegeben ?**
- Ursache finden und beseitigen.

memory leaks aufspüren



http://media.agrarheute.com/05/416105.jpg

leaks - Aufspüren

Grober Indikator: **Hauptspeicherverlauf**

- Taskmanager - Details

- **Sysinternals-Suite** - **ProcessExplorer**
  - https://technet.microsoft.com/de-de/sysinternals/bb545021.aspx
  - **ProcExp** - ProcessExplorer (⇔ Taskmanager)
  - **ProcMon** - ProcessMonitor (Trace Filesystem&Registry)
  - **DbgView** - DebugViewer (Debug-Output ohne VS)
  - **ZoomIt** - ScreenLupe ([Ctrl]-[1], [Ctrl]-[4])


novaBIMConverter.exe:6708 Properties
Image
Performance
Performance Graph
GPU Graph
Threads
TCP/IP
Security
Environment
.NET Assemblies
.NET Performance
Strings
CPU Usage
0.00%
Sysinternals - ProcessMonitor - Memory nur von "novaBIMConverter"
Projekt (neu-erstellen von) IFC-File: ,
U:\Union\Union Testcases\IFC\IFC_test_011-AC11-FZK-Haus_aender.ifc
dann Projekt schließen ... Vorgang 5x wiederholt (Release-Version)
Private Bytes
338.4 MB
288 MB
309 MB
332 MB
327 MB
340 MB
I/O
0
OK
Cancel

leaks - Aufspüren - Tipps (1)

- Nebeneinflüsse / Einmaleinflüsse minimieren
  - daher immer neu starten, 1x durchlaufen, erst danach Messpunkte setzen
- Kleine Beispiele ausarbeiten
  - offensichtliche Leaks aus großen Beispielen mit kleinen Beispielen nachprüfen und diese beheben.
  - Tooling / Memory-Analyse sehr zeitaufwändig.
  - Danach erst mit großem Beispiel analysieren.

leaks - Aufspüren - Tipps (2)

- Release-Version bzw.
- ReleaseDebugInfo - Build nutzen (C++)
  - Release-Build (mit PDB's)
  - Hintergrund: Debug-Speicherverwaltung (native C++) hat anderes Speicherlayout, mehr Checks, Landmark-Speicherblöcke.
  - andere Optimierungen im Code.

Nachtrag: C++ - Windows Debug Heap und Release-Builds

- In C++/MFC werden in der Regel via **#define new DEBUG_NEW** die new-Operationen überschrieben und im DebugBuild (_DEBUG) werden dann noch **filename** und **sourcecode line number** mit übergeben und verwaltet. – Das ist beim Release-Build nicht gegeben.
  - https://msdn.microsoft.com/de-de/library/tz7sxz99.aspx

- Windows Heap-Operationen sind deutlich langsamer, wenn der Prozess über den VisualStudio-Debugger gestartet wurde, insbesondere auch für Programme, die im Release-Build erstellt wurden. Dann wird automatisch der **Windows-Debug-Heap** aktiviert, der diverse Integritätschecks etc. ausführt.

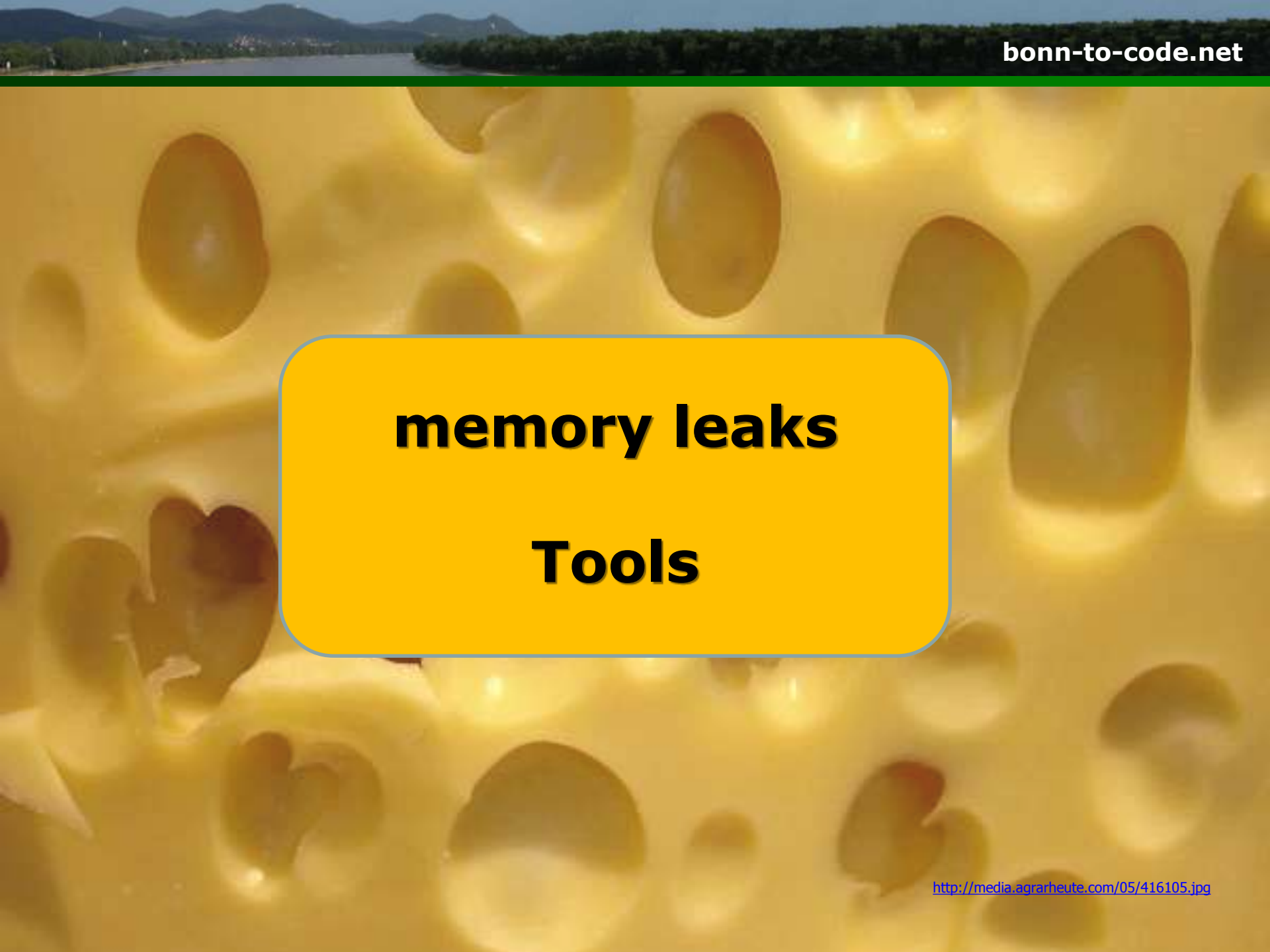
  - http://preshing.com/20110717/the-windows-heap-is-slow-when-launched-from-the-debugger/
  - http://ofekshilon.com/2014/09/20/accelerating-debug-runs-part-1-_no_debug_heap-2/
  - In den Artikeln steht auch drin, wie man das abstellen kann (Environment-Variable ***_NO_DEBUG_HEAP=1***)
  - Ob unmanaged HeapTesttools ebenfalls den Windows-Debug-Heap nutzen, oder eigene Mechanismen verwenden, kann ich nicht generell beantworten.
  - Häufig muss man für unmanaged Leak-Suche den Quelllcode haben und bestimmte Header/Libs hinzulinken.

- Da die Suche nach den Leaks generell zeitaufwändig und resourcenintensiv ist, war meine Idee, alles, was vom Leak-Such-Tool nicht benötigt wird, wegzulassen.

Nachträgliche Folie: 30.10.2015

memory leaks

Tools



http://media.agrarheute.com/05/416105.jpg

leaks - Tools generell (1)

- Überwachen und Protokollieren der Speicherverwaltung während des Programmlaufs
  - jedes **new / malloc**
  - jedes **delete / free**
  - in C# zusätzlich **IDisposable (Dispose() / using)**
  - in C# zusätzlich **GarbageCollector - Collect**

leaks - Tools generell (2)

- Einklinken ins Laufzeitsystem des Prozesses
- Dabei "Buchhaltung" über den Speicher
  - Ein- / Austragen in "Verwendungslisten"
- Speichern von
  - Objekt (HeapAdresse, .NET-ObjectId)
  - Allocated Size
  - CallStack der Allocation

leaks - Tools generell (3)

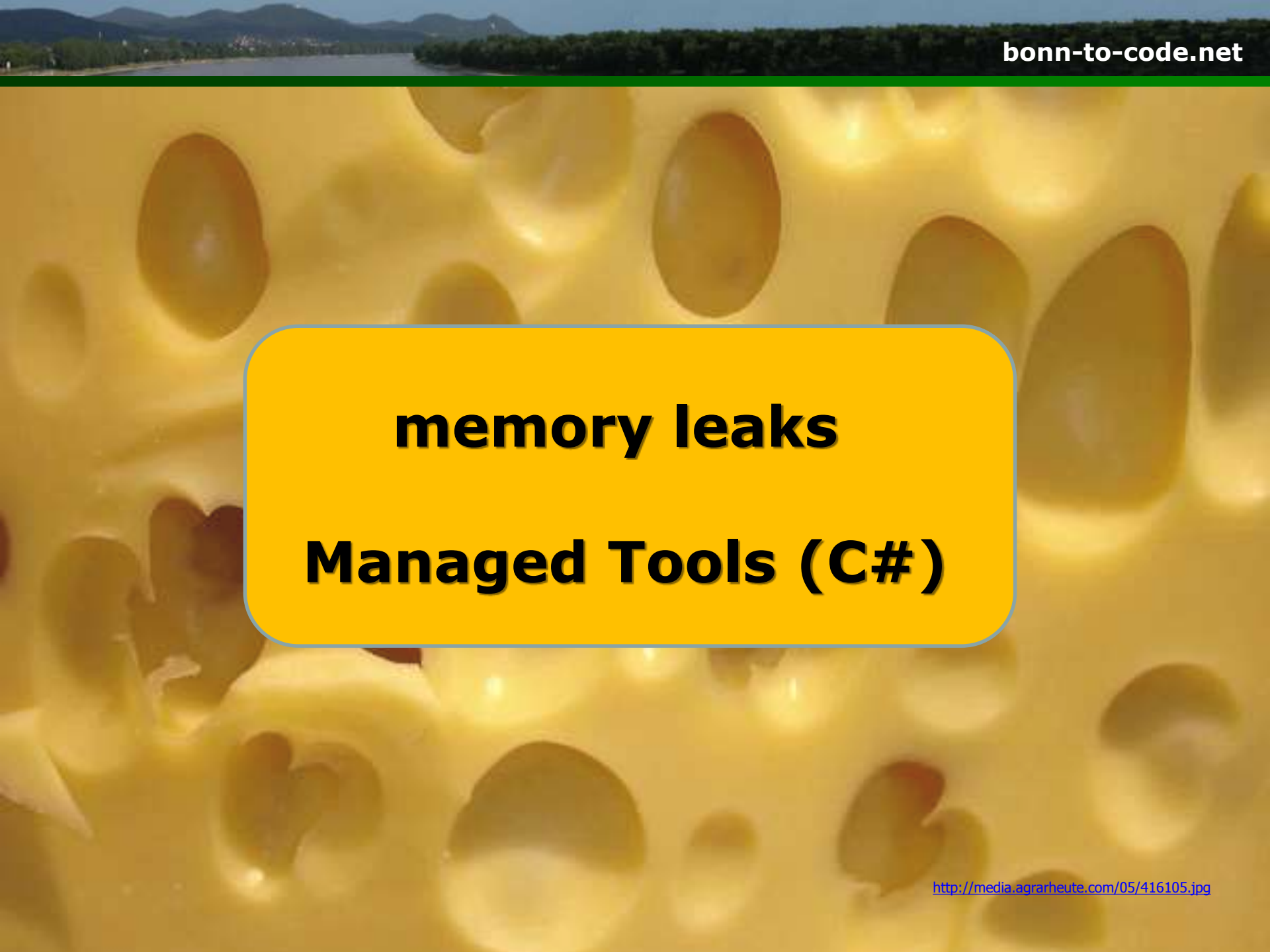
- Snapshots erstellen und merken
  - Liste der “aktiven” Objekte/Speicher zum Zeitpunkt x
- Snapshots vergleichen
- Snapshot - Delta analysieren
  - Welche Objekte sind hinzugekommen
  - Welche Objekte wurden freigegeben
  - aufbereitet anzeigen

leaks - Tools generell (4)

- Aggregieren / Sortieren der Objekte
  - nach Callstacks
  - nach Größen
  - nach Heap-Arten (“LargeObjectHeap”, “native”..)
  - nach “Eigenheiten”
    - schon disposed
    - Callstack nur von EventHandler gehalten
    - ….
- Rückverfolgung und Call-Tree-Anzeige bis auf die einzelne Instanz

memory leaks

Managed Tools (C#)



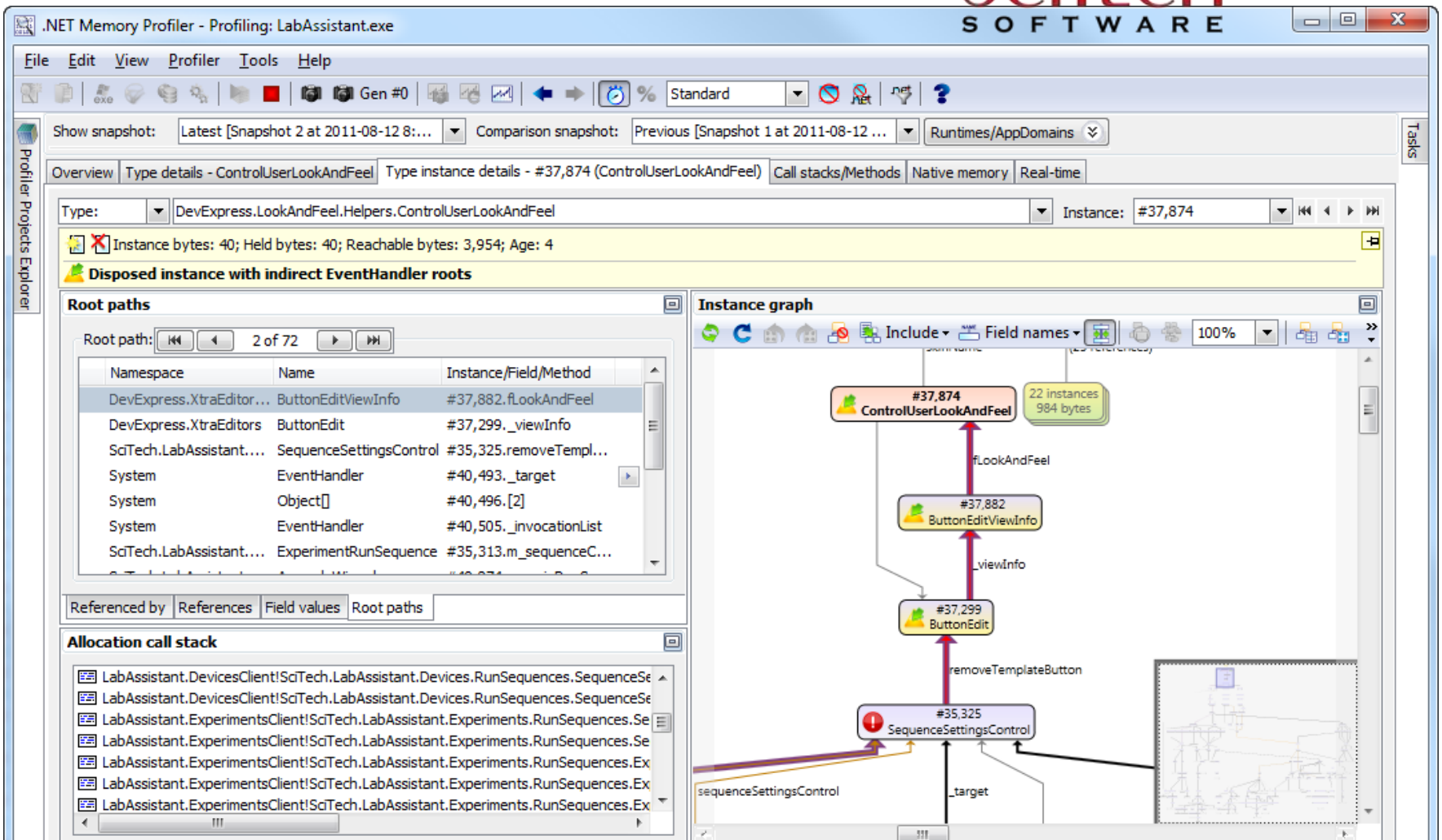
http://media.agrarheute.com/05/416105.jpg

memleaks - Managed Tools (C#)

- Sehr gute Toolunterstützung
- Wegen Reflection immer klar
  - Wer (class), Wo (methodinfo), Warum (callstack)
- Tools:
  - SciTech .NET Memory Profiler (super)
  - Microsoft .NET memory allocation profiler
  - Telerik Just Trace
  - ….

SciTech .NET Memory Profiler (1)

http://memprofiler.com/



SciTech .NET Memory Profiler (2)

- Preise: je nach Edition zwischen 200-600 €
  - http://memprofiler.com/order.aspx
- Dann noch Hinweis aus dem Publikum, dass dieser **Profiler auch eine API** hat
  - http://memprofiler.com/OnlineDocs/default.htm?turl=netmemoryprofilerapi.htm
- und **im Unittesting eingesetzt** werden kann.
  - http://memprofiler.com/OnlineDocs/default.htm?turl=performunittestingtogetherwiththeprofiler.htm

```
void ShowDialog()
{
        MemProfiler.FastSnapshot();
        // Show the dialog
        MemAssertion.NoNewInstances( ... );
}
```

Nachträgliche Folie: 30.10.2015

memleaks - Microsoft OnBoard

- Microsoft Performance and Diagnostics Hub

  http://blogs.msdn.com/b/visualstudioalm/archive/2013/07/12/performance-and-diagnostics-hub-in-visual-studio-2013.aspx

- Microsoft .NET memory allocation profiler

  http://blogs.msdn.com/b/dotnet/archive/2013/04/04/net-memory-allocation-profiling-with-visual-studio-2012.aspx

  http://blogs.msdn.com/b/visualstudioalm/archive/2014/04/02/diagnosing-memory-issues-with-the-new-memory-usage-tool-in-visual-studio.aspx

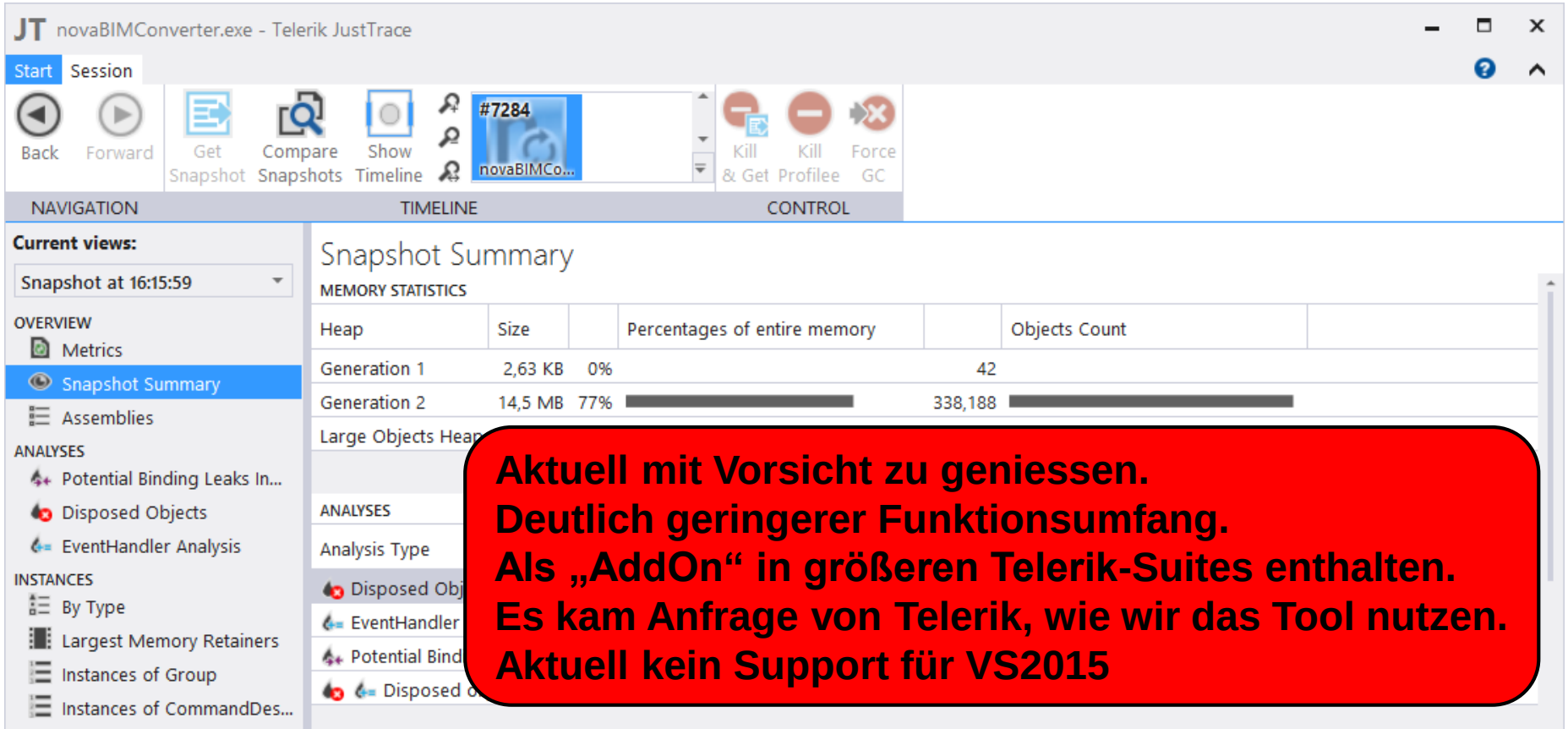
  Using the Profiling Tools From the Command-Line

  https://msdn.microsoft.com/en-us/library/ms182401.aspx

memleaks - Telerik Just Trace

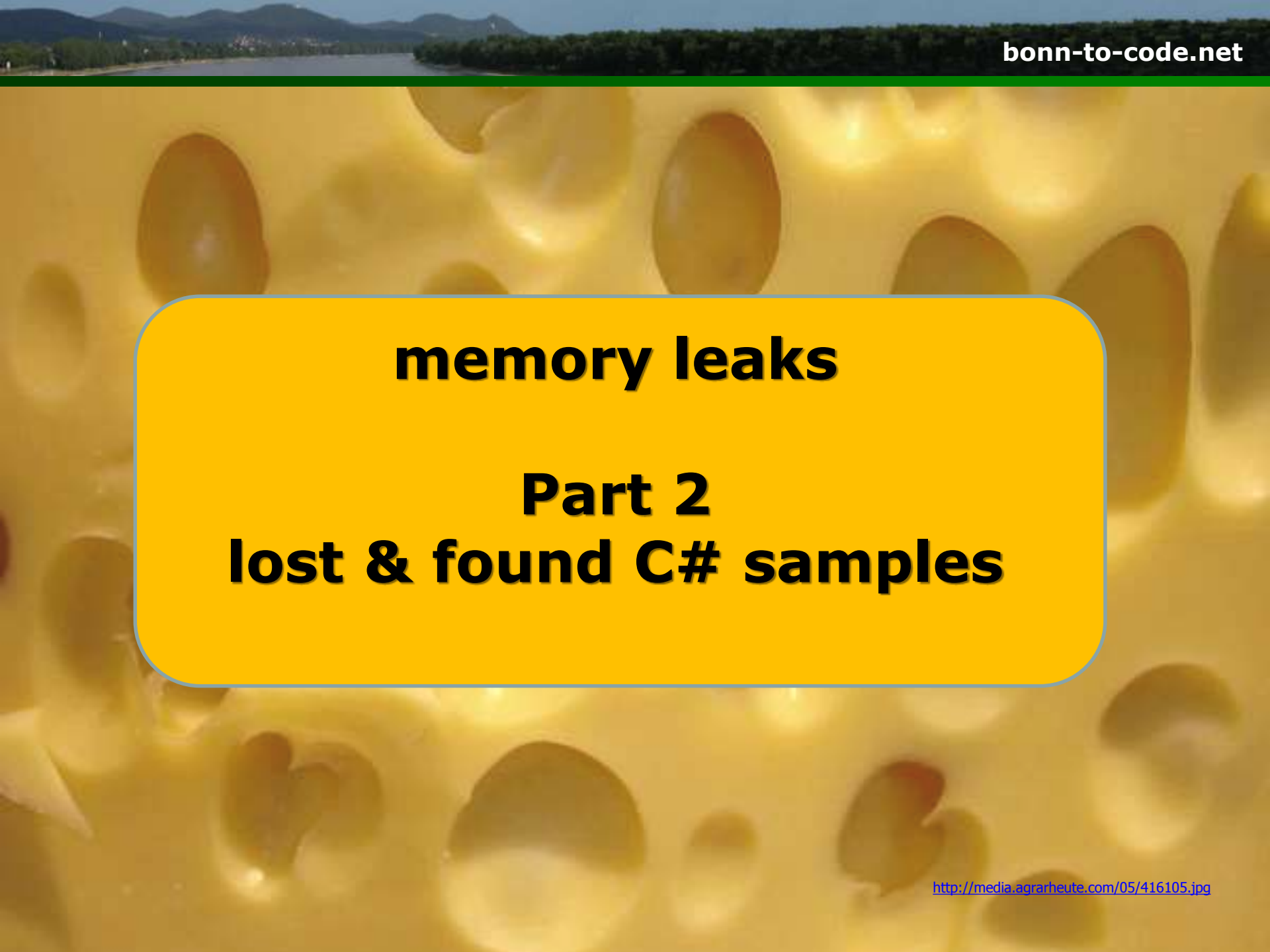
http://www.telerik.com/products/memory-performance-profiler.aspx

- Performance & Memory Analyse, Visual-Studio Plugin
- Sample Videos http://www.telerik.com/videos/justtrace



memory leaks

Part 2
lost & found C# samples



http://media.agrarheute.com/05/416105.jpg

Disclaimer:

Nachfolgende Codestellen bitte ohne Wertung
und unvoreingenommen betrachten.

Explizit kein “blame” !!!

sondern: Aus Fehlern lernen…

Lösungsansätze bitte nicht als “absolut” ansehen.

memory leaks

lost & found C# samples
Closures und EventHandler

http://media.agrarheute.com/05/416105.jpg

Event-Delegates mit Closures (1)

```
TreeItemViewModel it1 = new TreeItemViewModel
        { Header = SideViewNameToDisplayName(item),
          CommandParameter = item, ... };


it1.ContextMenuOpening +=
        (sender, args) => this.CreateViewMenu(it1);
```

delegate / closure hält das neu angelegte TreeItemViewModel.

Exkurs: Closure - Was ist das ? (1)

http://stackoverflow.com/questions/595482/what-are-closures-in-c
http://stackoverflow.com/questions/9591476/are-lambda-expressions-in-c-sharp-closures

http://diditwith.net/PermaLink,guid,235646ae-3476-4893-899d-105e4d48c25b.aspx

A **closure** is a function that is bound to the environment in which it is declared. Thus, **the function can reference elements from the environment within it's body.** …

This means that **local variables from the parenting method body can be referenced within the anonymous method's body**.

Exkurs: Closure - Was ist das ? (2)

```
delegate void      Action();

static void Main(string[]   args)
{
   int x = 0;
   Action a = delegate
              { Console.WriteLine(x); };
   a();
}
```

⇒ gibt **0** aus !

```
delegate void      Action();

static void Main(string[]   args)
{
   int x = 0;
   Action a = delegate
              { Console.WriteLine(x); };
   x = 1;
   a();
}
```

⇒ gibt **0 oder 1** aus ???

Exkurs: Closure - Was ist das ? (3)

http://diditwith.net/PermaLink,guid,235646ae-3476-4893-899d-105e4d48c25b.aspx

It turns out that **the answer is 1**, not 0.

The reason for this is that the anonymous method is a closure and is bound to its parenting method body and the local variables in it.

The important **distinction is that it is bound to variables, not to values.** In other words, the value of "x" is not copied in when "a" is declared. Instead, a reference to "x" is used so that "a" will always use the most recent value of "x". **In fact, this reference to "x" will be persisted even if "x" goes out of scope.**

Exkurs: Closure - Was ist das ? (4)

- Ein Closure ist eine **Funktion,**
- **gebunden zur umgebenden Methode**, in der sie deklariert wurde**.**
- **Lokale Variablen** der umgebenden Parent-Methode können innerhalb der anonymen Closure-Methode referenziert und genutzt werden.
- **Gebunden an Variablen, nicht an Werte**.
- Diese **Referenz zu „der lokalen Variable" wird länger beibehalten (persisted)** auch dann, wenn die umgebende Methode verlassen wurde und die „lokale Variable" out of scope gegangen wäre.

Exkurs: Closure - Was ist das ? (5)

```
delegate void         Action();

static void Main(string[]        args)
{
   int x = 0;
   Action a = delegate
          { Console.WriteLine(x); };
   x = 1;

   a();
}
```

```
delegate void Action();

sealed class ActionClosure {
        public int x;
        public void    AnonMethod() {
            Console.WriteLine(x);}
      }

static Action GetAction()        {
        ActionClosure closure = new ActionClosure();
        closure.x = 0;
        Action a = new Action(closure.AnonMethod);
        closure.x = 1;
        return a;      }

static void Main(string[]        args)      {
        Action a = GetAction();

        a();}
```

Exkurs: Closure - Was ist das ? (6)

```
delegate void

static void Main(stri
{
    int x = 0;
    Action a = delegat
            { Console.W
    x = 1;

    a();
}
```

```
delegate void Action();

sealed class ActionClosure {
        public int x;
        public void    AnonMethod() {
            Console.WriteLine(x)
      }

static Action GetAction()
        ActionClosure closure =
        closure.x = 0;
        Action a = new Action(
        closure.x = 1;
        return a;      }

static void Main(string[]
        Action a = GetAction()

        a();}
```

Vom Compiler erstellte **Closure-Klasse** mit den benutzten **„lokalen Variablen"** und der **„anonymen Methode"**

Vom Compiler erstellte **statische „Setup"-Funktion**, die das ClosureObjekt erstellt, die „lokalen Variablenzuweisungen ausführt und die Methode als Action zurückgibt.

In der eigentlichen **„Closure-Parent-Function"** dann nur noch das „Setup"-der ClosureAction.

Damit sind dann die „lokalen Variablen" aus dem lokalen Scope in ein losgelöstes Objekt gewandert !

Event-Delegates mit Closures (1)

```
TreeItemViewModel it1 = new TreeItemViewModel
        { Header = SideViewNameToDisplayName(item),
          CommandParameter = item, ... };


it1.ContextMenuOpening +=
        (sender, args) => this.CreateViewMenu(it1);
```

Zurück zum Beispiel

xxxxViewModel:

delegate / closure hält das neu angelegte TreeItemViewModel.

Event-Delegates mit Closures (2)

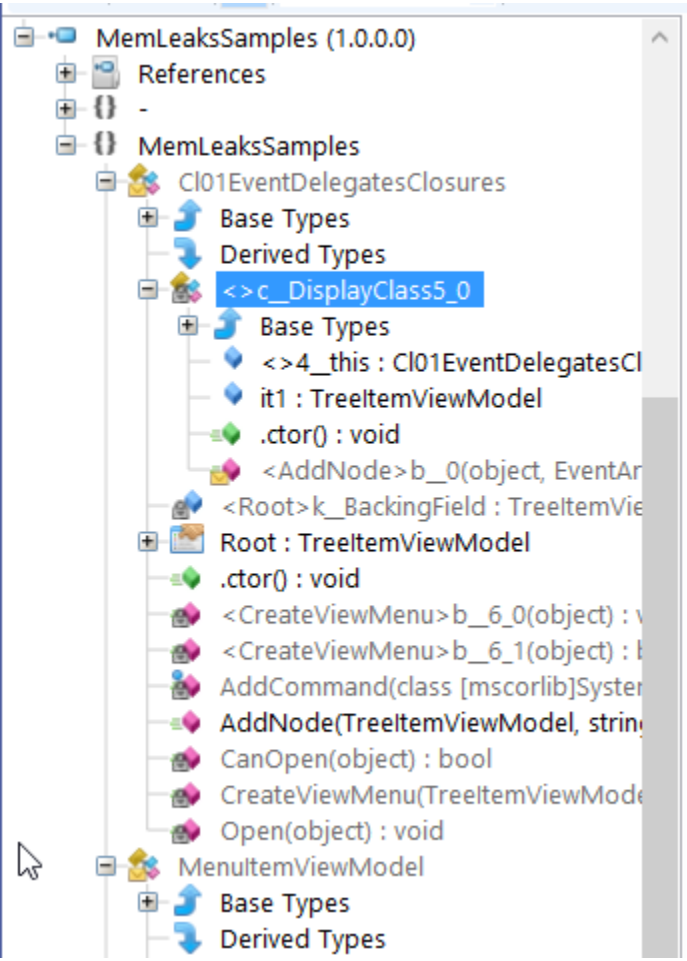
```
delegate void Action(object, EventArgs);

void AddNode(...)
{
  TreeItemViewModel it1 = new
                TreeItemViewModel(..);

  it1.ContextMenuOpening +=
    (sender, args) =>
      this.CreateViewMenu(it1);
}

irgendwann Aufruf aus UI:
ContextMenuOpeningAction(this, args);
```

```
delegate void Action(object, EventArgs);

nested sealed class ActionClosure {
    public TreeItemViewModel it1;
    public EmbeddingThisClass thisClass;
    public void AnonMethod(object, EventArgs) {
            thisClass.CreateViewMenu(it1);}
    }

static Action<object, EventArgs> GetAction() {
        ActionClosure closure = new ActionClosure();
        closure.it1 = new TreeItemViewModel(..);
        closure.thisClass = this;
        Action a = new Action<object,EventArgs>
                              (closure.AnonMethod);
        return a;      }

void AddNode(...)      {
        Action<object, EventArgs> a = GetAction();
        it1.ContextMenuOpening += new EventHandler(a);
        }
```

nochmal komplizierter wg. EventHandler und „this“-Zeiger

(Pseudocode –ohne Gewähr)

Event-Delegates mit Closures (3)




```
.class nested private auto ansi sealed beforefieldinit '<>c__DisplayClass5_
    extends [mscorlib]System.Object
{
    .custom instance void [mscorlib]System.Runtime.CompilerServices.Compile
        01 00 00 00
    )
    // Fields
    .field public class MemLeaksSamples.TreeItemViewModel it1
    .field public class MemLeaksSamples.Cl01EventDelegatesClosures '<>4__th

    // Methods
    .method public hidebysig specialname rtspecialname
        instance void .ctor () cil managed
    {
        // Method begins at RVA 0x21cc
        // Code size 8 (0x8)
        .maxstack 8

        IL_0000: ldarg.0
        IL_0001: call instance void [mscorlib]System.Object::.ctor()
        IL_0006: nop
        IL_0007: ret
    } // end of method '<>c__DisplayClass5_0'::.ctor

    .method assembly hidebysig
        instance void '<AddNode>b__0' (
            object sender,
            class [mscorlib]System.EventArgs args
```

C# / IlSpy –Sample

Event-Delegates mit Closures (fixed)

```
private void CreateContextMenuOnContextMenuOpening(object sender, EventArgs e) {
   TreeItemViewModel tivm = sender as TreeItemViewModel;
   Contract.Assert(tivm != null);
   this.CreateViewMenu(tivm);
}

void AddNode(...){
    TreeItemViewModel it1 = new TreeItemViewModel {
       Header = SideViewNameToDisplayName(item),
       CommandParameter = item, ... };

it1.ContextMenuOpening +=
    CreateContextMenuOnContextMenuOpening;
        (sender, args) => this.CreateViewMenu(it1);
```

meine Lösung:

expliziter Eventhandler-delegate, der aus dem **sender** das betroffene TreeItemViewModel auswertet.

memory leaks

EventHandler und Lambda‘s
+= / -=

http://media.agrarheute.com/05/416105.jpg

Exkurs: EventHandler und Lambda's (1)

```
button.Click += (s, e) => MessageBox.Show("Woho");
button.Click -= (s, e) => MessageBox.Show("Woho");
```

Bedeutung eines Lambda's:

Lambda expressions are nothing more than a language feature that the compiler translates into the exact same code that you are used to working with.

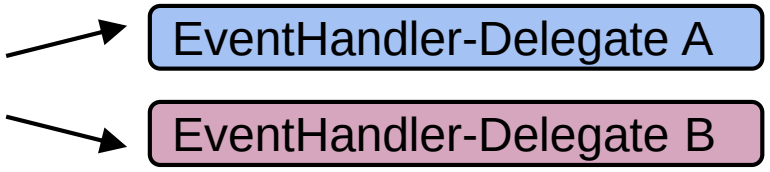
```
button.Click += new EventHandler
        (delegate(Object s, EventArgs e)
                {MessageBox.Show("Woho");});
```

http://stackoverflow.com/questions/2465040/using-lambda-expressions-for-event-handlers

Exkurs: EventHandler und Lambda's (2)

Achtung: jedes Lambda ist ein eigener Delegate:

```
button.Click += (s, e) => MessageBox.Show("Woho");
button.Click -= (s, e) => MessageBox.Show("Woho");
```



Daher Achtung:

EventHandler-Delegate A ist nach dem -= immer noch attached !!!

http://stackoverflow.com/questions/2465040/using-lambda-expressions-for-event-handlers

Exkurs: EventHandler und Lambda's (3)

Richtig: Explizites EventHandler-Object:

```
EventHandler handler = (s, e) => MessageBox.Show("Woho");

    button.Click += handler;
    button.Click -= handler;
```

oder: Explizite EventHandler-Method:

```
private void OnContextMenuOpening(object sender, ContextMenuEventArgs e){...}

    this.AssociatedObject.ContextMenuOpening += this.OnContextMenuOpening;
    this.AssociatedObject.ContextMenuOpening -= this.OnContextMenuOpening;
```

http://stackoverflow.com/questions/2465040/using-lambda-expressions-for-event-handlers

memory leaks

Static Object (Singleton) mit "this.MemberFunc-Delegate"

http://media.agrarheute.com/05/416105.jpg

Static Object mit this.MemberFunc-Delegate

- static (Command) - Object
- dynamisch angelegt aus einem PropertyGetter beim ersten Zugriff
- mit einem this.MemberFunc/Action delegate, der im neuen static-Object (Command) gespeichert wird

⇒ **bewirkt, dass das erzeugende this-Objekt nie freigegeben wird.**

Umgebung: DocumentExplorer - TreeItems....

```
private static ICommand showInfoCommand;

private void ShowInfoCommandExecute(object param)
    { ... }
private bool ShowInfoCommandCanExecute(object param)
    { ... }

public ICommand ShowInfoCommand
{ get {
        return CommandHelpers.LookupOrCreateCommand(
                ref showInfoCommand,
                () => new RelayCommand(
                   this.ShowInfoCommandExecute,
                   this.ShowInfoCommandCanExecute));
        }
}
```

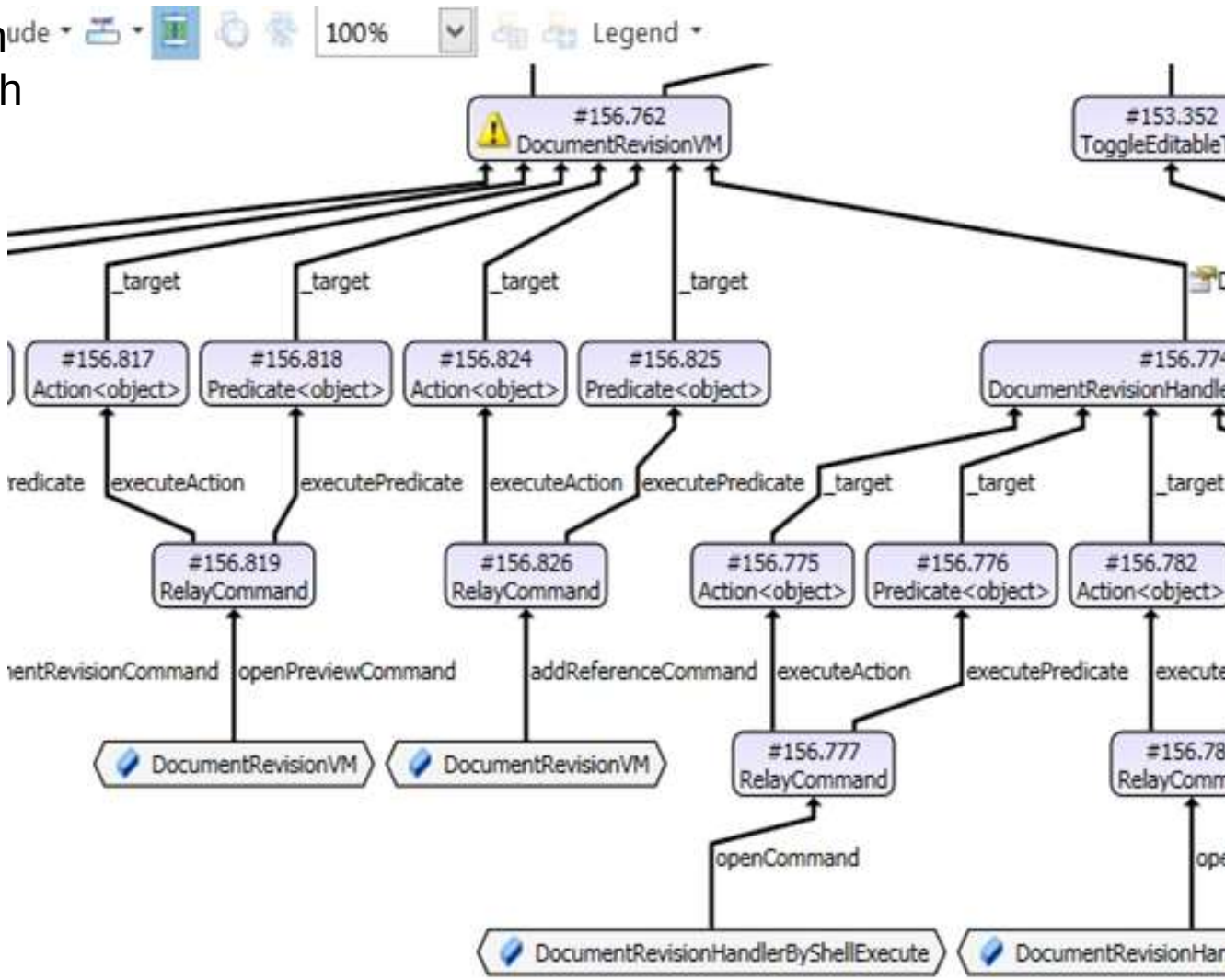
Die erste Anforderung von ShowInfoCommand erstellt die static Instance.

Diese verwendet das this-Object !!

Dadurch bleibt das 1. TreeItem (this-Object) auf immer und ewig eingeladen, auch dann wenn längst anderes Projekt geöffnet wurde.

Ansicht des Call-Graph für das Leak im SciTech .NET Memory Profiler

http://memprofiler.com/



Static Object mit this.MemberFunc-Delegate 1. Lösungsmöglichkeit

```
private static ICommand closeCommand;

private static void CloseCommandExecute(object pa
   {...}
private static bool CloseCommandCanExecute(object
   {...}

public ICommand CloseCommand {
        get  {
         return CommandHelpers.LookupOrCreateCommand(
                           ref closeCommand,
                 () => new RelayCommand(
                     this.CloseCommandExecute,
                     this.CloseCommandCanExecute));
               }
}
```

1. Lösungsmöglichkeit:

Die Handlerfunktionen auch static machen…
… wenn inhaltlich möglich.

Vorteil: Es gibt nur ein Command-Object für alle Treeitems

Static Object mit this.MemberFunc-Delegate 2. Lösungsmöglichkeit

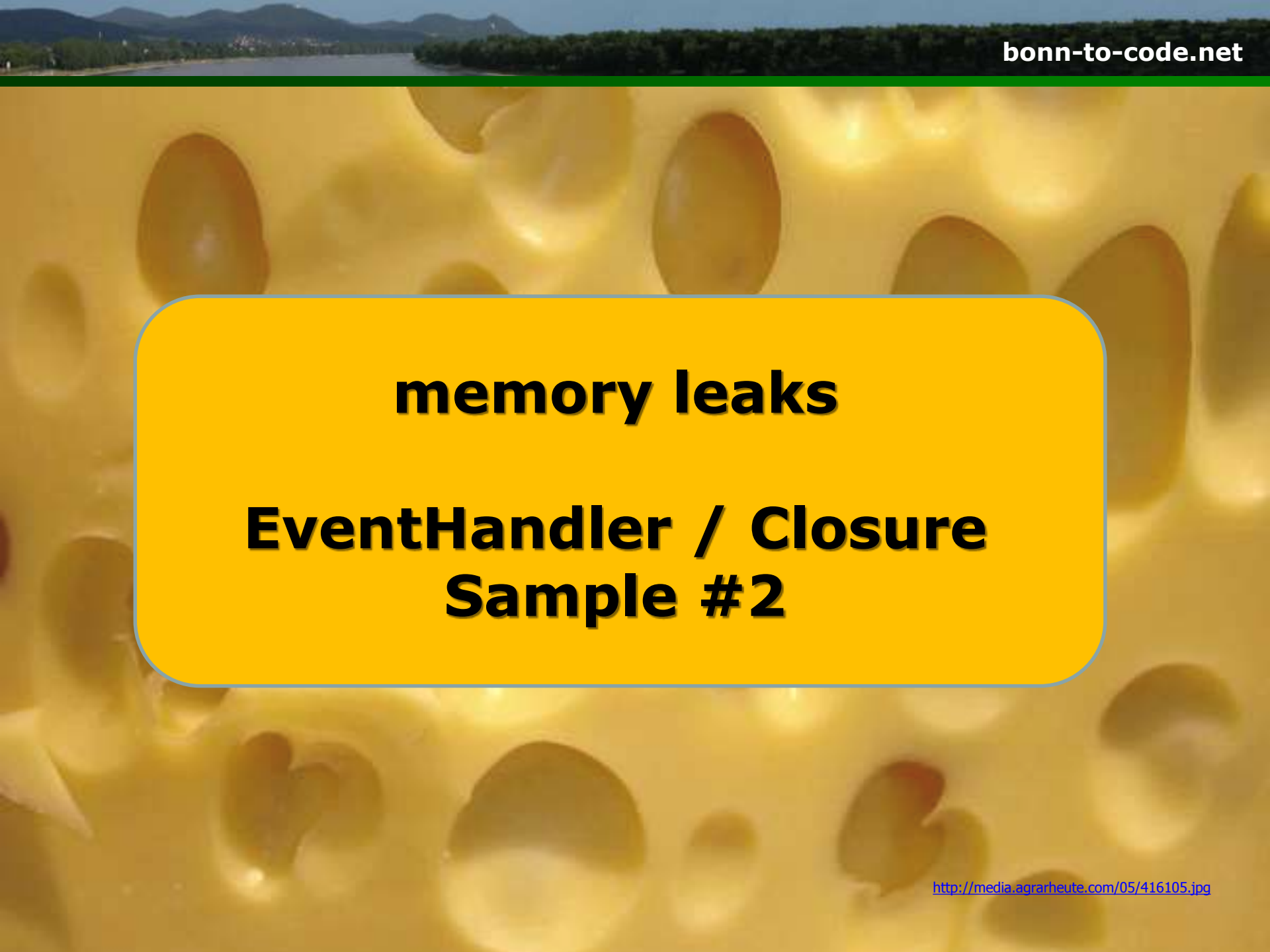
```
private ~~static~~ ICommand closeCommand;

private void CloseCommandExecute(object param)
   {...}
private bool CloseCommandCanExecute(object param)
   {...}

public ICommand CloseCommand {
        get  {
         return CommandHelpers.LookupOrCreateCommand(
                     ref this.closeCommand,
                     () => new RelayCommand(
                          this.CloseCommandExecute,
                          this.CloseCommandCanExecute));
            }         }
```

2. Lösungsmöglichkeit:

Das static closeCommand zum Member machen.

memory leaks

EventHandler / Closure Sample #2



http://media.agrarheute.com/05/416105.jpg

Eventhandler-Closure Sample 2 (1)

Context: in einem WPF - Behavior (OnAttached / OnDetaching)

```
private void AssociatedObject_SelectedItemChanged(object send
    var trv = this.AssociatedObject;
    var offset = trv.HorizontalScrollOffset;
    if (!Double.IsNaN(offset))  {
       var scrollViewer = trv.GetVisualDescendants<ScrollViewer>().FirstOrDefa
       if (scrollViewer != null) {
          scrollViewer.ScrollToHorizontalOffset(offset);   // and because that wasn't
          var scrollBar  = scrollViewer.GetVisualDescendants<ScrollBar>().
                    .FirstOrDefault(cur => cur.Orientation == Orientation.Horizontal
          if (scrollBar != null) {
             RoutedPropertyChangedEventHandler<double> handler = null;
             handler = (sender1, e1) =>
             {
                          scrollBar.ValueChanged -= handler;
                          scrollViewer.ScrollToHorizontalOffset(offset);
             };
             scrollBar.ValueChanged += handler;
          }
       }
    }
}
```

BehaviorKlasse und TreeItem bleiben geladen. Die ScrollBar, ScrollViewer und der Handler selbst werden noch von dem Closure referenziert.

Wenn ein schmales Item oder in einem breiten Tree selektiert wurde, ist beim “BringItemIntoView” der ScrollBarValue nicht geändert worden, deswegen lief der ValueChanged-Delegate nicht an und deswegen ist das -= Abhängen des Handler nicht ausgeführt worden.

Closures auflösen - Sample 2 (2)

Lösung: - ScrollViewer und ScrollBar in BehaviorKlasse als Member,
- Explizite EventHandlerDelegateFunc
- und passende Attach/Detach-Funktionen
- und passend Freigaben bei SelectionChanged

```
private ScrollViewer myScrollViewer;
private ScrollBar horizontalScrollBar;

private void ScrollBarAdditionalHandler(object sender,
                    routedPropertyChangedEventArgs<double> args)

private void AttachAddtionalHandlerToScrollBar(ScrollBar scrollBar, ScrollViewer scrollViewe
private void DetachAddtionalHandlerFromScrollBar()
```

TFS - CS 31987

Closures - Möglichst einfache lokale Daten halten

```
private void CreateDomain(ProjectFacade projectFacade, ProjectModuleRegisterEntry item, Storage storage)
        {...
    var projectDataDomainFacade = new ProjectDataDomainFacade(moduleId,
          this.environment,
          () => this.GetProjectDataDomain(projectFacade.Id, moduleId) );
    projectFacade.AddDataDomain(projectDataDomainFacade);
        ... }
```

Komplettes ProjectFacade- Objekt wird im Closure gehalten.

Blockiert dadurch Freigabe.

Verbesserte Variante:

```
        {...
        string projectFacadeId = projectFacade.Id;
    var projectDataDomainFacade = new ProjectDataDomainFacade(moduleId,
          this.environment,
          () => this.GetProjectDataDomain(projectFacadeId, moduleId) );
    projectFacade.AddDataDomain(projectDataDomainFacade);
        ... }
```

Hält nur den ProjectID-String im Closure.

Objekte werden korrekt entladen.

WPF-Behaviors : OnDetaching

Bei Behaviors auch ans Freigeben der Events/Bindings im OnDetaching denken:

```
public class TreeViewBindableClipboardBehavior : Behavior<TreeView>   {
 private CommandBinding cutCommandBinding; (copy/paste dto)

 protected override void OnAttached(){
        base.OnAttached();
        this.cutCommandBinding = new CommandBinding(...);
        this.AssociatedObject.CommandBindings.Add(this.cutCommandBinding);
}
```

```
 protected override void OnDetaching() {
        if (this.AssociatedObject != null) {

this.AssociatedObject.CommandBindings.Remove(this.cutCommandBinding); }
        base.OnDetaching();        }
```

OnDetaching() fehlte !!!
TreeItems wurden wegen des CommandBindings im Speicher behalten !

Xaml: MemLeak (WPF) (1)

Einfache Messungen: OpenProject, CloseProject (mit existing Projects)

-> genau nur noch **1 RecentView wird pro Aufruf des Backstage-Views zuviel beibehalten** von:

- **Grid._parent.ContentPresenter._templatedParent**
- **Grid._parent** direct
- via **ColumnDefinitionCollection._owner.Grid_parent**...

Nach diverse Versuchen Ursache gefunden:

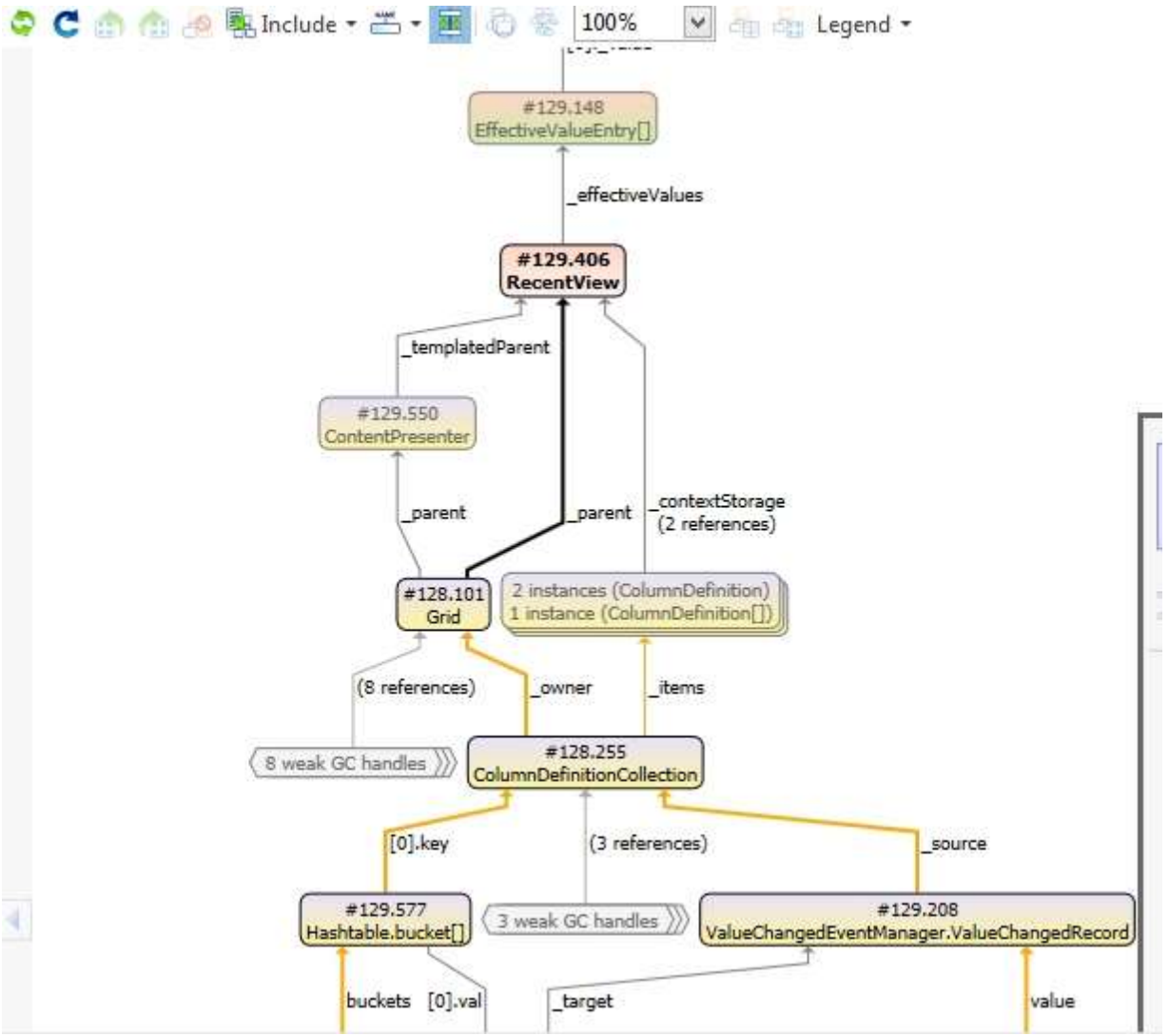
RecentView.Xaml:

Im WPF-Style für Linien wird auf das Grid.ColumnDefinitions.Count gebunden....

Xaml: MemLeak (WPF) (2)

Ansicht des Call-Graph für das Leak im SciTech .NET Memory Profiler

http://memprofiler.com/



Xaml: MemLeak (WPF) (3)

```
<Style x:Key="horizontalLineStyle"
        TargetType="Line" BasedOn="{StaticResource lineStyle}">
        <Setter Property="X2" Value="1" />
        <Setter Property="VerticalAlignment" Value="Bottom" />
        <Setter Property="Grid.ColumnSpan"
         Value="{Binding
                  Path=ColumnDefinitions.Count,
                  RelativeSource={RelativeSource AncestorType=Grid}}"/>
</Style>
```

Vermutung: ist auch **in Richtung Closures** bei der Definition des Bindings für den PropertySetter ?!?

Dann noch: Grid.ColumnSpan wurde an der Style-Verwendungsstelle eh fest auf 1 bzw. 2 gesetzt, damit ist dieser Setter komplett überflüssig !

WPF - Bitmap-Images

http://stackoverflow.com/questions/1684489/how-do-you-make-sure-wpf-releases-large-bitmapsource-from-memory
http://code.logos.com/blog/2008/04/memory_leak_with_bitmapimage_and_memorystream.html

Instead of loading from a Uri, just construct a Stream and pass it to BitmapFrame's constructor:

```
var source = new BitmapImage();
using(Stream stream = ...)
{
            source.BeginInit();
            source.StreamSource = stream;
            source.CacheOption = BitmapCacheOption.OnLoad; // not a mistake - see below
            source.EndInit();
}
```

TODO nochmal genauer recherchieren, warum...

LoadImageSource... das **BitmapImage noch Image.Freeze();**

Hintergrund: Freezed-Objects können mehrfach im WPF-UI genutzt werden, da sie sich nicht mehr ändern können.

memory leaks

Zusammenfassung

http://media.agrarheute.com/05/416105.jpg

leaks - Zusammenfassung (1)

Vorgehensweise zur Beseitigung von Leaks:

- reproduzierbarer, möglichst kleiner Fall herstellen
- Snapshots erstellen
- Snapshot-Delta - analysieren
- Ursachen beheben

Tools helfen bei der Analyse

leaks - Zusammenfassung (2)

Geduld, Geduld, Geduld ....

- Die Messläufe, insbesondere unmanaged sind langandauernd.

Ergebnisse hinterfragen ....

- Häufig haben die Tools recht, auch wenn man es nicht direkt versteht.

Zähigkeit siegt ....

memleaks - Zusammenfassung (3)

Typische Leak-Fälle vermeiden - Managed C#

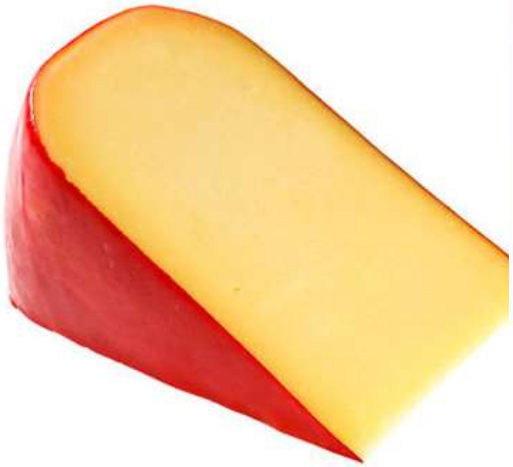
- Closures bei Lambda-Ausdrücken - (lokale Variablen)
- explizite EventHandler-Delegates oder EventHandler-Objekt bevorzugen.
- EventHandler -= Lambda-Ausdruck ist falsch !
- EventHandler wenn möglich wieder abmelden (Dispose…) Hilft dem GC!
- StaticObject z.B.Command mit this.Member-Delegates halten this-Objekt !
- versuchen, Closures zu umgehen oder klein zu halten (string id statt Obj)
- WPF-Behaviors: OnAttached() - Items auch im OnDetaching() freigeben.
- WPF-XAML-Binding-Leaks via RelativeSource (schwer zu lokalisieren)
- WPF-BitmapImages via Stream laden (⇒ LoadImageSource), Freeze()
- StringBuilder bzw. String.Format nutzen (Speicherfragmentierung)
- Large Object Heap (⇒ Speicherfragmentierung beachten)

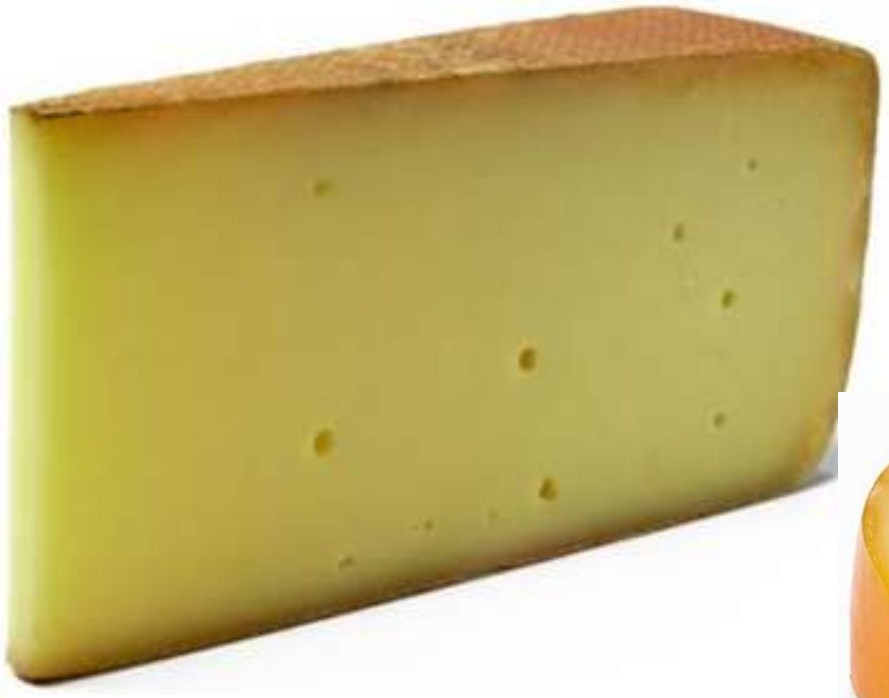
memory leaks

Alles klar – oder ?

http://media.agrarheute.com/05/416105.jpg

zukünftig dann….





https://www.ndr.de/ratgeber/kochen/kaese169_v-contentgross.jpg
http://mhstatic.de/fm/1/thumbnails/sh_Mediashow_Zinkhaltige_Produkte_Edamer_800x533.jpg.3018478.jpg
http://www.bioprodukte.de/files/2011/04/K%C3%A4se.jpg

memory leaks

Danke für die Aufmerksamkeit

http://media.agrarheute.com/05/416105.jpg

memleaks - Disclaimer

Der Vortrag erhebt keinen Anspruch auf Vollständigkeit, Eignung, Garantie oder jegliche Art von Gewährleistung überhaupt.

Der Vortrag wird “so wie gesehen” bereitgestellt.

- No claim of suitability, guarantee, or any warranty whatsoever is provided.
- The presentation is provided "as-is".