Porting BETA to ROTOR/sscli

ROTOR Capstone Workshop,

Sept 19 - 21 2005

by Peter Andersen

ROTOR RFP II

1. “hello-world” up to complete compiler test suite
   - Almost OK at time of RFP II
2. Implement (some) missing features in language mapping and libraries
3. Bootstrap the BETA compiler to ROTOR and .NET
4. Possibly develop a GUI framework on top of ROTOR and .NET.
   - System.Windows.Forms and System.Drawing not available on ROTOR (but Views available)
5. Investigate mechanisms for Simula/BETA-style coroutines

[Re 1-2] BETA.Net status

- **Most language features implemented**
- **Patterns** mapped to classes, nested patterns become nested classes with explicit uplevel link
- **Enter-do-exit semantics** implemented by generating separate methods for Enter(), Do(), and Exit()
- Use of **patterns as methods** supported by generated convenience methods
- **Virtual classes** – corresponding to generics (.NET 2.0) implemented with virtual instantiation methods and a lot of (unnecessary) casting.
- **INNER** semantics implemented with multiple virtual method chains

[Re 1-2] BETA.Net status

- **Pattern variables**: Classes and methods as first-class values implemented with reflection
- **Leave/restart** out of nested method activations implemented with exceptions (expensive!)
- **Multiple return values** – implemented with extra fields
- **Interface to external classes** - Rudimentary support for overloading, constructors etc. Offline batch tool **`dotnet2beta`** implemented using reflection
- **Coroutines** and **concurrency** - More on this later…
- **Basic libraries** (text, file, time etc.), implemented on top of .NET BCL

[Re 3] Bootstrapped compiler

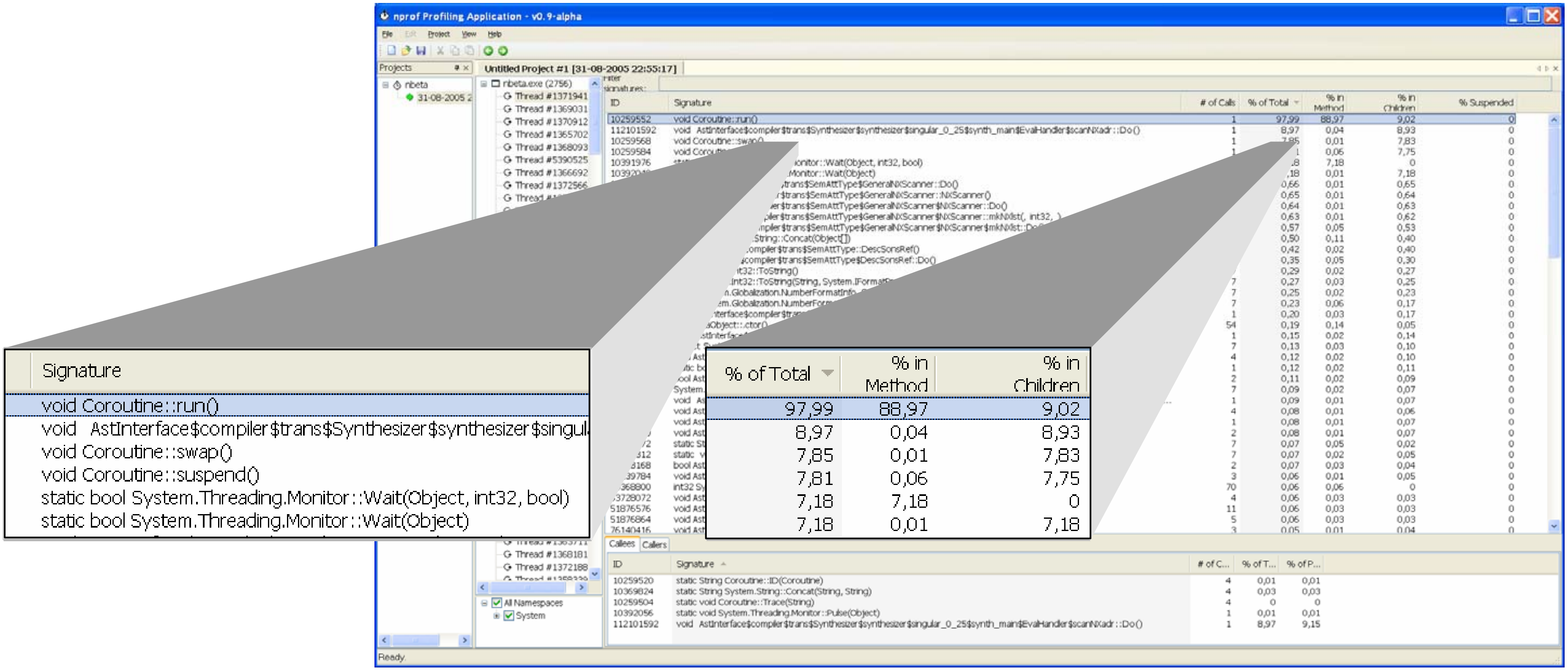
- 122.000 lines BETA source, including used libraries
- Bootstrapped compiler up-n-running ☺
  - Download: http://www.daimi.au.dk/~beta/ooli/download/
  - Very slow!
- Managed compiler running on .NET CLR:
  - Compiles small programs nicely
  - Crashes on larger programs with `System.OutOfMemoryException`
- Perfect case for debugging via ROTOR (SOS extension)
  - ”what is the actual reason that the EE throws that exception?”
- BUT: Managed compiler does *not* fail on ROTOR ☺ ☹ ?

[Re 3] Compiler statistics

- Some statistics: Compilation of complete test suite on 1.7GHz laptop:
- About 12000 lines of BETA code, including parsing, semantic checking, code generation and 75 calls of ilasm. 96000 lines of IL generated (!).
- **Native (win32) nbeta**:
  - 21 seconds
  - 11Mb memory consumption
- **.NET CLR:**
  - Fails about halfway with **`System.OutOfMemoryException`**
  - Memory consumption 110Mb (> 100Mb of *physical* memory free!?)
  - Number of threads created: 7872
- **sscli (win32) checked**:
  - 2 hours 3 minutes ~ slowdown 350 !!
  - 160Mb max mem. consumption.
  - Number of threads created: 25502
- **sscli (win32) fastchecked**:
  - 54 minutes ~ slowdown 154
- **sscli (win32) free**:
  - 17 minutes ~ slowdown 48
  - 145Mb max mem. consumption.

[Re 3] Why compiler slow?

- Nprof screenshot:



[Re 3] Bootstrapped compiler

- Indicates that current Coroutine implementation is major bottleneck
- Other measurements also indicate that Coroutine switching contributes about a factor 100 more than other BETA constructs to slow down
- *So we need to look more at Coroutines!!*

Go to coroutine appendix

[Re 5]: Coroutines in C#

- Imagine:

```
abstract class Coroutine  // Similar to Thread
{ ...
  public void call() { ... }  // a.k.a. attach/resume
  public void suspend() { ... }
  public abstract void Do();   // Similar to Run()
}
SpecificCoroutine: Coroutine{ … }
Coroutine S = new SpecificCoroutine();
```

- Do() is action part of coroutine
- First S.call() will invoke S.Do()
- S.suspend() will return to the point of S.call() and resume execution after S.call()
- Subsequent S.call() will resume execution in S *where it was last suspended*

[Re 5] Current impl. of class Coroutine

- **`class Coroutine`** implemented by means of **`System.Threading.Thread`** and **`System.Threading.Monitor`**

```
public class Coroutine {
  public static Coroutine current;
  private Coroutine caller; // backlink; this when suspended
  private System.Threading.Thread myThread; // notice private
  public Coroutine ()
   { … Constructor: allocate myThread starting in run; set up caller etc. }
  private void run()
   { … Thread entry point: call Do()and then terminate myThread … }
  public void swap()
   { … Main call() / suspend() handling; next slide … }
  public abstract void Do();
}
```

[Re 5] Current impl. of **`Coroutine.swap()`**

- Used asymmetrically:
  - Call: this == to become current; this.caller == this
  - Suspend: this == current; this.caller to be resumed

```
public void swap()
{
  lock (this){
    Coroutine old_current = current;
    current = caller;
    caller = old_current;
    if (!myThread.IsAlive) {
      myThread.Start();
    } else {
      System.Threading.Monitor.Pulse(this);
    }
    System.Threading.Monitor.Wait(this);
    }
 }
```

Currently executing Component/Coroutine

Swap pointers

Start or resume new current

Suspend old current

[Re 5] Coroutine problems?

- Measurements from JVM indicate that thread *allocation* is the culprit – use of threadpool for reusing threads gave significant speed up
  - .NET / ROTOR same problem?
  - Did not (yet) try this optimization for .NET
- Otherwise unreferenced threads with unfinished ThreadStart methods count as GC roots?
  - Lots of such coroutines in BETA execution

[Re 5] Coroutine support in .NET/ROTOR?

- Direct light-weight user defined scheduling desirable
  - C# 2.0 `yield`?
  - P/Invoke of WIN32 Fibers?
  - ROTOR extension?

[Re 5] Comparison with C# 2.0 `yield`

- C# 2.0 has new feature called yield return
  - Yield corresponds to suspend()
- Used for implementing enumerator pattern
- May be considered ”poor man’s coroutine”
- Implemented as a simple state-machine
- Can only ”save” one stack frame

[Re 5] P/Invoke of WIN32 Fibers

- Described in
  - Ajai Shankar: Implementing e Unmanaged Fiber API http://msdn.microsoft.com/msdnmag/issues/03/09/CoroutinesinNET

Update - 9/16/2005: The solution described in this article relies on undocumented functionality that is not supported by Microsoft at this time

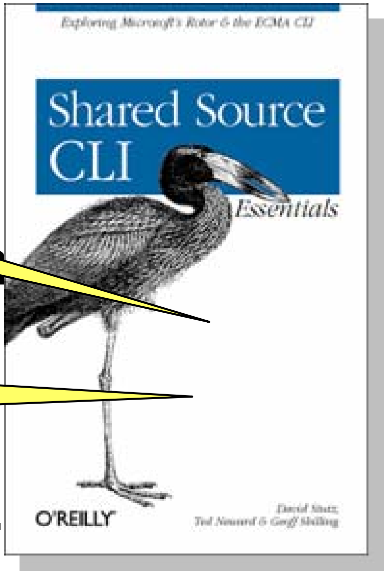
- Pretty ”hairy” code, inclusing use of undocumented APIs
- http://blogs.msdn.com/greggm/archive/2004/06/07/150298.aspx :
  - *”**DON’T USE FIBERS IN A MANAGED APPLICATION**. The 1.1/1.0 runtime will deadlock if you try to managed debug a managed application that used fibers. The CLR team did a lot of work for **fiber support in the 2.0 runtime**, but it still won't support debugging”*
- Sample (not?) available for .Net 2.0:
  - http://msdn2.microsoft.com/en-us/library/sdsb4a8k (CoopFiber)
  - (thank you Fabio)

[Re 5] ROTOR extension?

- ROT… **coswap** bytecode?
  - Addition of … presumably straigh…
  - W… …ith managed P… …s, thread synchron… …ception handling etc.

The concurrency model is quite complex...

As promised, this [aborting a thread] is a pretty hefty chunk of code...



- We read "Shared Source CLI Essentials" and browsed the 5M lines of ROTOR source a lot.
- A little overwhelmed with the challenge!
- Needed pre-study with simpler architecture

[Re 5] pre-vm

- Joined forces with another ongoing project: PalCom (http://www.ist-palcom.org)
- As part of PalCom Runtime Environment: pre-vm virtual machine
- Simple dynamically typed (a la Smalltalk) interpreted runtime system, <20 bytecodes
- Prototype implemented in Java, currently being re-implemented in C++ for use in small devices
- (Partial) language mappings for BETA, Java, Smalltalk

[Re 5] pre-vm: coroutines

- Coroutine-based environment
  - Coroutines (not threads) are the basic scheduling unit
  - Coroutines scheduled by user-programmed schedulers
    - (Somewhat like Fibers in WIN32)
  - Default (replaceable) schedulers included in library
  - Different scheduling strategies can be used for (disjunct) sets of coroutines, e.g. hierarchical schedulers
  - Preemptively scheduled coroutines (i.e. threads) programmed using interrupt/timer mechanism

[Re 5] pre-vm: implementation

- VM support for coroutines:
  - **`Coroutine`** VM-defined entity which includes a stack, a current execution point and a backlink to coroutine that attached it
  - Bytecode for coroutine swap:
    - Attach(x) → push x; coswap
    - Suspend(x) → push x; coswap
    - Notice: A coroutine may suspend another (which needs to be active)
  - Primitives for setting an interrupt interval and an interrupt handler

[Re 5] pre-vm: preemptive scheduling

- Preemptive scheduling:
  - Set an interrupt interval
  - Set an interrupt handler: Must include a `void handle(Object)` method
  - In the handler call Suspend() on the currently active coroutine and Attach() on the next coroutine to run
- Interrupts only detected at the so-called *safe-points* (backward-branches, method entries, and I/O calls)
  - Comparable with GC safe-points in Rotor

[Re 5] pre-vm: synchronization and I/O

- Synchronization:
  - Critical regions, mutexes, semaphores etc. built using a single Lock() primitive
  - Currently no need for e.g. test-and-set bytecode, as interrupts only occur at well-known safe-points
  - May be needed if more interrupt-places added to reduce latency; simple to implement
- Blocking I/O impl: Two approaches:
  - If an interrupt is detected at the I/O call, interpreter continues on a fresh (native) thread, and blocking I/O thread stops after I/O call completed (current strategy)
  - Programmer must distinguish between potentially blocking and non-blocking I/O calls. Blocking calls automatically done by another thread (considered)

[Re 5] Coroutines: status

- Pre-vm is still very much work-in-progress (project on second year out of four)
- Results so far look promising; i.e. the idea of using coroutines as the sole scheduling entity seems realizable
  - Simple VM-level semantics
  - Simple implementation
- Problem with unterminated coroutines staying alive can be completely controlled by user-programmed scheduler
- Potential problem:
  - Different user-programmed (preemptive) schedulers in separate components may conflict – especially if the need to synchronize between components

[Re 5] Coroutines: status

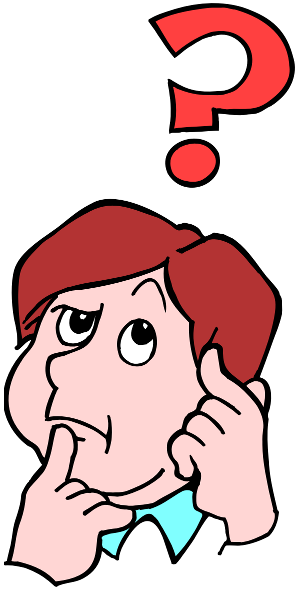
- Difficult (yet) to say how much of this can be applied to ROTOR/.NET
  - Same ideas could probably be realized if coroutine systems always reside within one managed thread and synchronization of coroutines with managed threads is not considered
- Interesting to see how far we can get in ROTOR.
  - Probably much better ”dressed” when we have the embedded C++ implementation of pre-vm implemented and example applications running on top of it
- If a Fiber API actually gets into Whidbey, presumably this will get much easier

Future plans

- Obvious optimizations in current C# implementation of Coroutines (e.g. ThreadPool)
- More lessons to learn from pre-vm work
- Perhaps co-operation with Cambridge?
  - Previous contact to MSR Cambridge guys who patched a JVM to include support for Coroutines
- Perhaps co-operation with Redmond?
  - Contacts within C# team and CLR team. Coroutine co-operation suggested.
- Perhaps co-operation with PUC-Rio
- Exciting to see what things look like after .Net 2.0 (and later ROTOR 2.0)

Contacts: Questions?

- Peter Andersen (that’s me)
  - mailto:datpete@daimi.au.dk
- Prof. Ole Lehrmann Madsen
  - mailto:olm@daimi.au.dk
- Info & download:
  - http://www.daimi.au.dk/~beta/ooli



Appendices

- The following slides not presented at Capstone workshop
- Added as backgound material
- Appendix A describes a basic BETA program and how it is mapped to .NET
- Appendix B describes coroutines in general, here expressed in C#

App. A: BETA Language Mapping

- Object-oriented programming language
  - Scandinavian school of OO, starting with the Simula languages
  - Simple example:

```
Calculator:
 (# R: @integer;
    set:
     (# V: @integer enter V do V → R #);
    add:
     (# V: @integer enter V do R+V → R exit R #);
 #);
```

A *pattern* named `Calculator`

Static instance variable named R

Internal *pattern* named `set` with an input variable V

Internal *pattern* named add with an input variable V and a return value named R

App. A: BETA example use

```
Calculator:
 (# R: @integer;
    set:
     (# V: @integer enter V do V → R #);
    add:
     (# V: @integer enter V do R+V → R exit R #);
 #);
```

Use of add as a method:

```
C: @Calculator;
X: @integer;
5 → C.add → X
```

Use of add as a class:

```
C: @Calculator;
X: @integer;
A: ^C.add;
&C.add[] → A[];
5 → A → X
```

Creation of an instance of C.add

Execution of the C.add instance

App. A: BETA vs. CLR/CLS

- Class and method unified in *pattern*
- General nesting of patterns, i.e. also of methods
  - Uplevel access to fields of outer patterns
- INNER instead of super
- Enter-Do-Exit semantics
- Genericity in the form of virtual patterns
- Multiple return values
- Active objects in the form of Coroutines
- No constructors, no overloading
- No dynamic exceptions

App. A: BETA.Net/Rotor Challenges

- Mapping must be *complete* and *semantically correct*
- BETA should be able to *use* classes from other languages and visa versa
- BETA should be able to *inherit* classes from other languages and visa versa
- In .NET terminology:
  - BETA compliant with Common Language Specification (CLS)
  - BETA should be a *CLS Extender*
- The BETA mapping should be ’nice’ when seen from other languages
- Existing BETA source code should compile for .NET

App. A: Mapping patterns: nested classes

```
public class Calculator: System.Object {
  public int R;
  public class add: System.Object {
    public int V;
    Calculator origin;
    public add(Calculator outer) { origin = outer; }
    public void Enter(int a) { V = a; }
    public void Do() { origin.R = origin.R + V; }
    public int Exit() { return origin.R; }
  }
  public int call_add(int V){
    add A = new add(this);
    A.Enter(V);
    A.Do();
    return A.Exit();
  }
  …
}
```

CLS does not allow for this to be called just add()

```
Calculator:
 (# R: @integer;
    …
    add:
     (# V: @integer
     enter V
     do R+V → R
     exit R
     #);
   #);
```

App. A: Use of add as a class:

```
C: @Calculator;

X: @integer;
A: ^C.add;
&C.add[] → A[];
5 → A → X
```

```
Calculator C
   = new Calculator()
int X;
Calculator.add A;
A = new Calculator.add(C);
A.Enter(5);
A.Do()
X = A.Exit();
```

App. A: Use of add as a method

```
C: @Calculator;

X: @integer;
5 → C.add → X
```

```
Calculator C
   = new Calculator()
int X;
X = C.call_add(5);
```

App. A: Not described here…

- **Virtual classes** – corresponding to generics (.NET 2.0) – implemented with virtual instantiation methods and a lot of (unnecessary) casting.
- **Coroutines** and **concurrency** - More on this later…
- **Pattern variables**: Classes and methods as first-class values – implemented with reflection
- **Leave/restart** out of nested method activations – implemented with exceptions (expensive!)
- **Multiple return values** – implemented with extra fields
- **Interface to external classes** - Rudimentary support for overloading, constructors etc. Offline batch tool `dotnet2beta` implemented using reflection
- Numerous minor details!

Go back to coroutine implementation

App. B: Coroutines in C#

- Given the C# Coroutine definition included in the main part of these slides:

```
abstract class Coroutine  // Similar to Thread
{ ...
  public void call() { ... }
  public void suspend() { ... }
  public abstract void Do();   // Similar to Run()
}
SpecificCoroutine: Coroutine{ … }
Coroutine S = new SpecificCoroutine();
```

App. B: Example: Adder

- Produces sequence start + start, (start+1)+(start+1) …
- By using (infinite) recursion
- Suspends after each computation

```
class Adder: Coroutine {
    public int res;
    int start;
    public Adder(int s) {
        start = s;
    }
    void compute(int V){
        res = V+V;
        suspend();
        compute(V+1);
    }
    public override void Do() {
        compute(start);
    }
}
```

App. B: Example: Multiplier

- Produces sequence start * start, (start+1) * (start+1) …
- By using (infinite) recursion
- Suspends after each computation

```
class Multiplier: Coroutine {
    public int res;
    int start;
    public Multiplier(int s) {
        start = s;
    }
    void compute(int V){
        res = V*V;
        suspend();
        compute(V+1);
    }
    public override void Do() {
        compute(start);
    }
}
```

App. B: Merger

- Merge sequences produced by Adder instance and Multiplier instance
- Sort in ascending order
- First 6 values

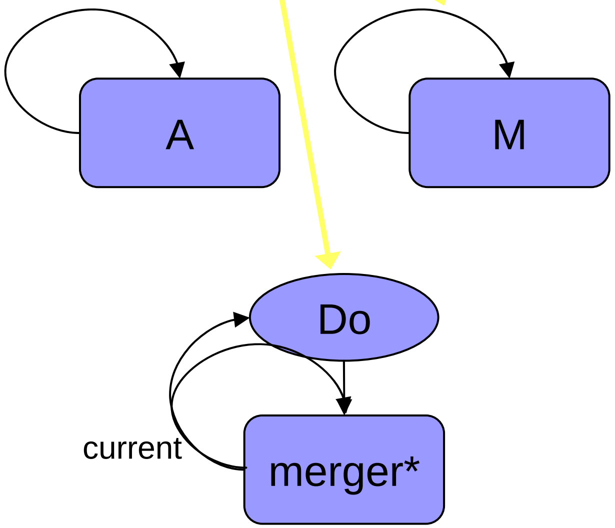
```
class Merger: Coroutine {
    Adder A = new Adder(3);
    Multiplier M = new Multiplier(2);
    public override void Do() {
      A.call(); M.call();
      for (int i=0; i<6; i++){
        if (A.res < M.res) {
          Console.WriteLine("A: " + A.res);
          A.call();
        } else {
          Console.WriteLine("M: " + M.res);
          M.call();
        }
      }
    }
    public static void Main(String[] args) {
      (new Merger()).call()
    }
}
```

Adder | Multiplier | **Merger**

```
class Merger: Coroutine {
   Adder A = new Adder(3);
   Multiplier M = new Multiplier(2);
   public override void Do() {
     A.call(); M.call();
     for (int i=0; i<6; i++){
        if (A.res < M.res) {
          Console.WriteLine("A: " + A.res);
          A.call();
        } else {
          Console.WriteLine("M: " + M.res);
          M.call();
        }
     }
   }
   public static void Main(String[] args) {
     (new Merger()).call()
   }
}
```

Caller link (back-link) – initially self

Method invocation

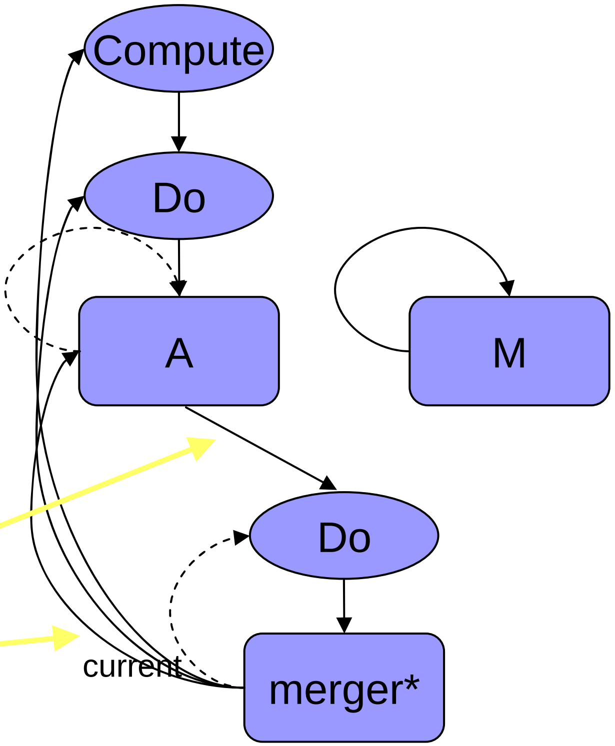


Coroutine

Adder | Multiplier | Merger

```
class Adder: Coroutine {
    public int res;
    int start;
    public Adder(int s) {
      start = s;
    }
    void compute(int V){
        res = V+V;
        suspend();
        compute(V+1);
    }
    public override void Do() {
        compute(start);
    }
}
```

Call() is basically just a swap of two pointers

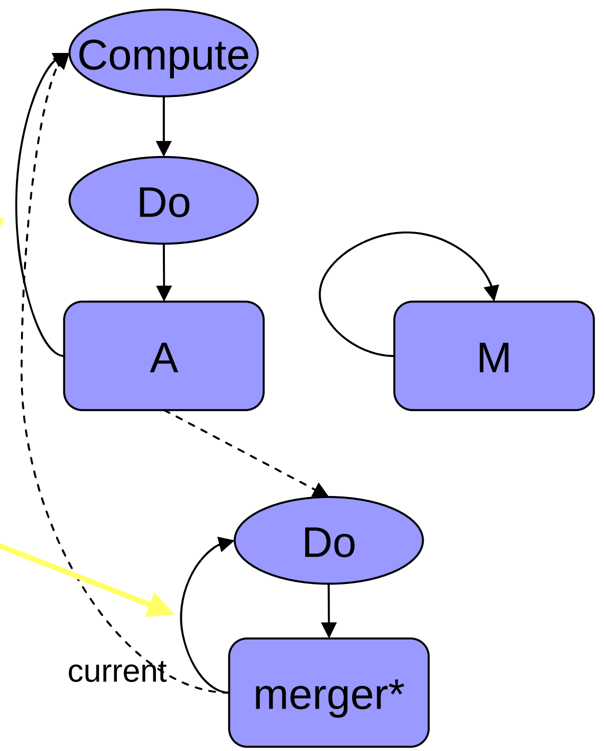


Go back to coroutine implementation

Adder | Multiplier | **Merger**

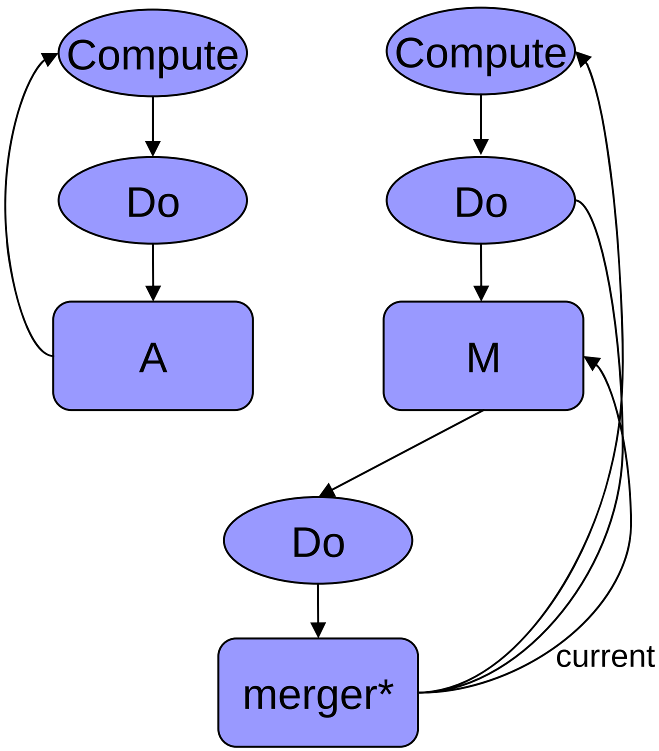
```
class Merger: Coroutine {
    Adder A = new Adder(3);
    Multiplier M = new Multiplier(2);
    public override void Do() {
       A.call();  ➔ M.call();
       for (int i=0; i<6; i++){
           if (A.res < M.res) {
              Console.WriteLine("A: " + A.res);
              A.call();
           } else {
              Console.WriteLine("M: " + M.res);
              M.call();
          }
       }
    }
    public static void Main(String[] args) {
        (new Merger()).call()
    }
}
```

suspend() is also basically just a swap of two pointers



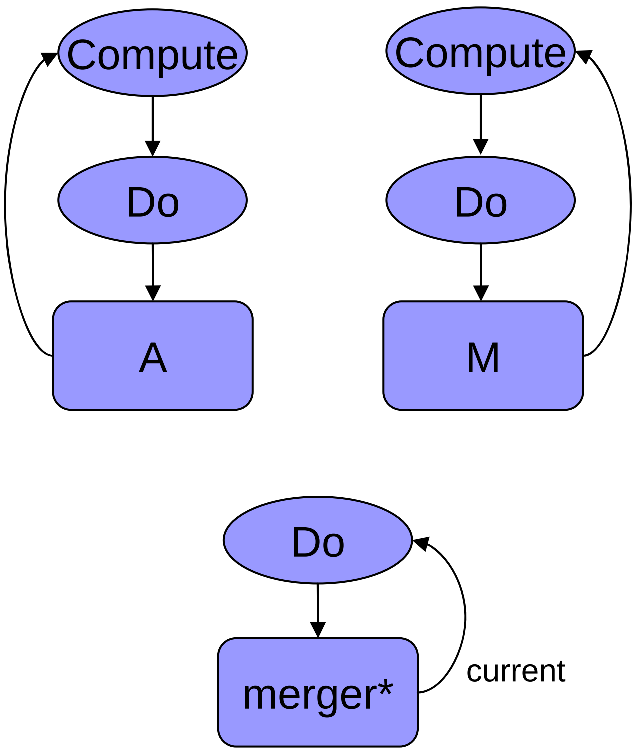
Adder | **Multiplier** | Merger

```
class Multiplier: Coroutine {
    public int res;
    int start;
    public Multiplier(int s) {
      start = s;
    }
    void compute(int V){
        → res = V*V;
        → suspend();
          compute(V+1);
    }
→ public override void Do() {
        → compute(start);
    }
}
```



Adder | Multiplier | **Merger**

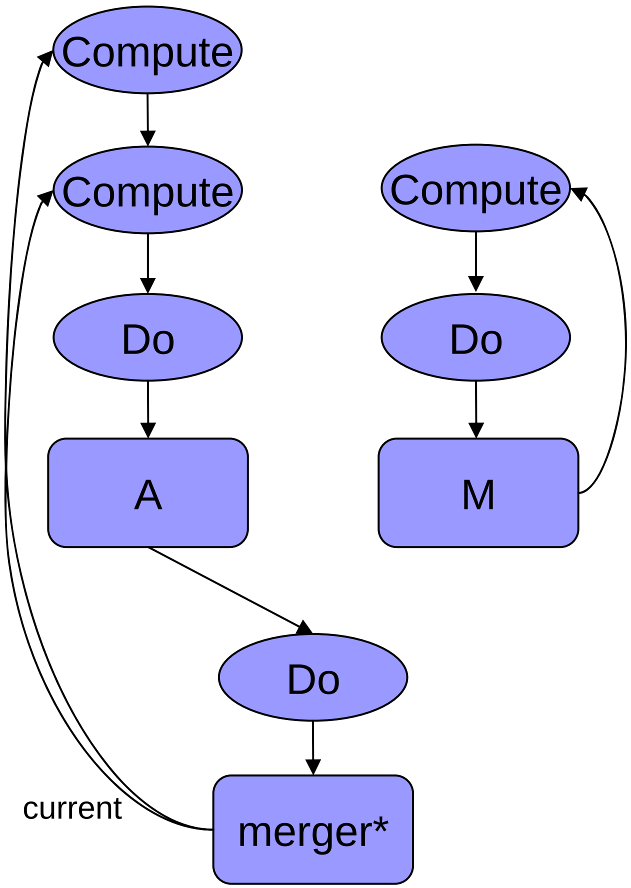
```
class Merger: Coroutine {
    Adder A = new Adder(3);
    Multiplier M = new Multiplier(2);
    public override void Do() {
       A.call(); M.call();
       for (int i=0; i<6; i++){
          if (A.res < M.res) {
             Console.WriteLine("A: " + A.res);
             A.call();
          } else {
             Console.WriteLine("M: " + M.res);
             M.call();
          }
       }
    }
    public static void Main(String[] args) {
       (new Merger()).call()
    }
}
```



Go back to coroutine implementation

Adder | Multiplier | Merger

```
class Adder: Coroutine {
    public int res;
    int start;
    public Adder(int s) {
      start = s;
    }
    void compute(int V){
        → res = V+V;
        → suspend();
        → compute(V+1);
    }
    public override void Do() {
            compute(start);
    }
}
```



Go back to coroutine implementation

Adder | Multiplier | **Merger**

```
class Merger: Coroutine {
    Adder A = new Adder(3);
    Multiplier M = new Multiplier(2);
    public override void Do() {
       A.call(); M.call();
       for (int i=0; i<6; i++){
           if (A.res < M.res) {
              Console.WriteLine("A: " + A.res);
              A.call();
           } else {
              Console.WriteLine("M: " + M.res);
              M.call();
          }
       }
    }
    public static void Main(String[] args) {
       (new Merger()).call()
    }
}
```

… and so on