Writing High Performance Delphi Application

Primož Gabrijelčič

About me

- Primož Gabrijelčič
- http://primoz.gabrijelcic.org
- programmer, MVP, writer, blogger, consultant, speaker
- Blog *http://thedelphigeek.com*
- Twitter *@thedelphigeek*
- Skype *gabr42*
- LinkedIn *gabr42*
- GitHub *gabr42*
- SO *gabr*
- Google+ *Primož Gabrijelčič*

The Delphi Geek

random ramblings on Delphi, programming, Delphi programming, and all the rest

Sunday, May 20, 2018

Introducing MultiBuilder

When I'm working on OmniThreadLibrary, I have to keep backwards compatibility in mind. And man, is that hard! OmniThreadLibrary supports every Delphi from 2007 onward and that means lots of IFDEFs and some ugly hacks.

Typically I develop new stuff on Berlin or Tokyo and then occasionally start a batch script that tests if everything compiles and runs unit tests for all supported platforms in parallel. (I wrote about that system two years ago in article Setting Up a Parallel Build System.) Dealing with 14 DOS windows, each showing compilation log, is cumbersome, though, and that's why I do this step entirely too infrequently.

For quite some time I wanted to write a simple framework that would put my homebrew build batch into a more formal framework and which would display compilation results in a nicer way. Well, this weekend I had some time and I sat down and put together just that - a MultiBuilder. I can promise that it will be extensively used in development of OmniThreadLibrary v4. (Which, incidentally, will drop support for 2007 to XE. You've been notified.)

The rest of this post represents a short documentation for the project, taken from its GitHub page.

I don't plan to spend much time on this project. If you find it useful and if you would like to make it better, go ahead! Make the changes, create a pull request. I'll be very happy to consider all improvements.

Read more »

Posted by Primož Gabrijelčič at 21:46 1 comment: Links to this post

Labels: build, compiler, Delphi, FireMonkey

embarcadero MVP

Pages

Presentations

PERFORMANCE

Performance

- What is performance?
- How do we “add it to the program”?
  - There is no silver bullet!

What is performance?

- Running “fast enough”
- Raw speed
- Responsiveness
  - Non-blocking

Improving performance

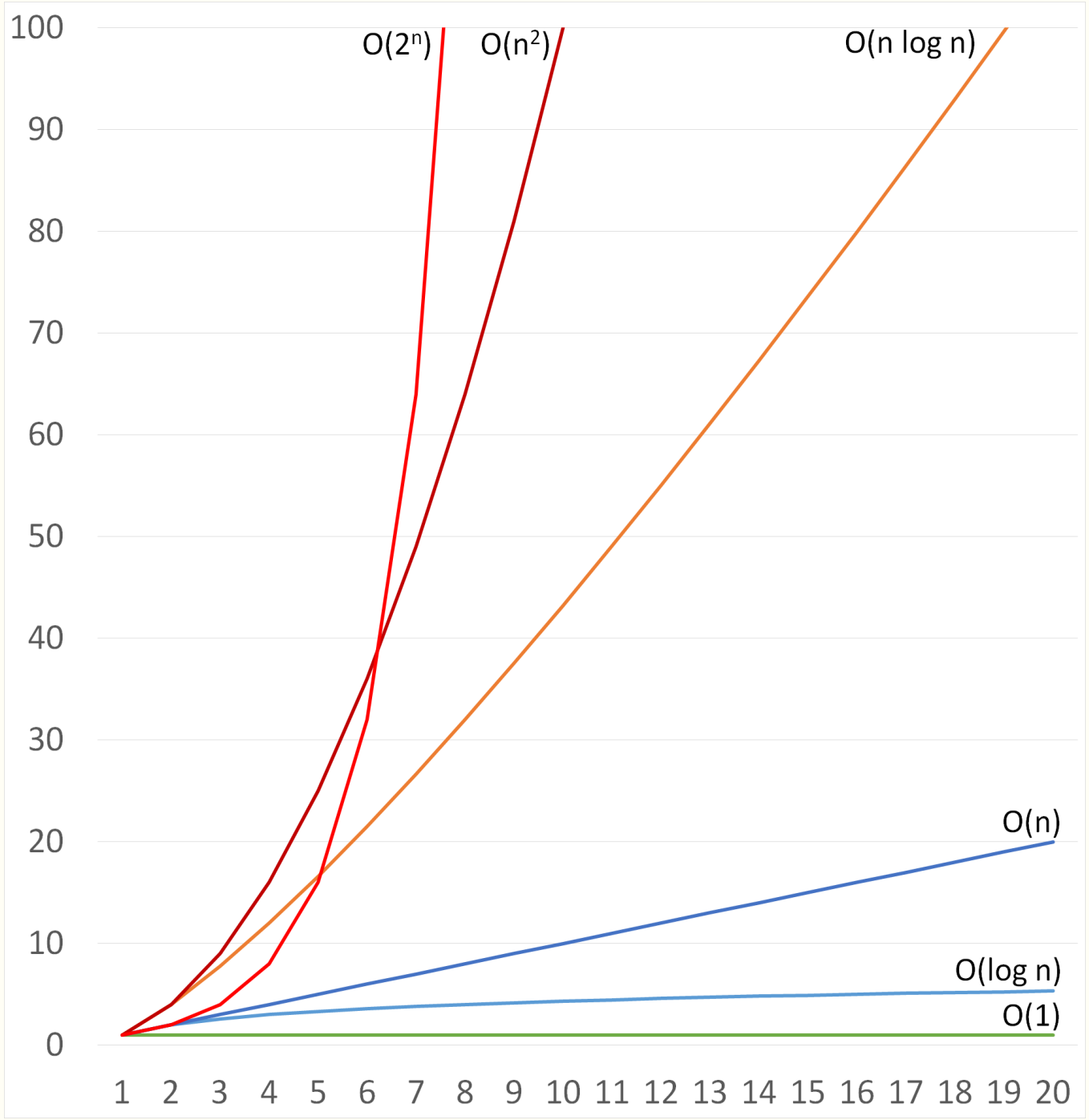
- Analyzing algorithms
- Measuring execution time
- Fixing algorithms
- Fine tuning the code
- Mastering memory manager
- Writing parallel code
- Importing libraries

Algorithm Complexity

Algorithm complexity

- Tells us how algorithm slows down if data size is increased by a factor of *n*
- O()
  - O(n), O(n^2), O(n log n) …
- Time **and** space complexity

- O(1) accessing array elements
- O(log n) searching in ordered list
- O(n) linear search
- O(n log n) quick sort (average)
- $O(n^2)$ quick sort (worst), naive sort (bubblesort, insertion, selection)
- $O(c^n)$ recursive Fibonacci, travelling salesman



Comparing complexities

Data size	O(1)	O(log n)	O(n)	O(n log n)	$O(n^2)$	$O(c^n)$
1	1	1	1	1	1	1
10	1	4	10	43	100	512
100	1	8	100	764	10.000	10^{29}
300	1	9	300	2.769	90.000	10^{90}

O(RTL)

- Lists
  - access O(1)
  - Search O(n) / O(n log n)
  - Sort O(n log n)
- Dictionary
  - * O(1)
  - Search by value O(n)
  - Unordered!
    - Spring4D 1.3?
- Trees
  - * O(log n)
  - Spring4D 1.2.2

http://bigocheatsheet.com/

Measuring Performance

Measuring

- Manual
  - GetTickCount
  - QueryPerformanceCounter
  - TStopwatch

- Automated - Profilers
  - Sampling
  - Instrumenting
    - Binary
    - Source

Free profilers

- AsmProfiler
  - Instrumenting and sampling
  - 32-bit
  - https://github.com/andremussche/asmprofiler
- Sampling Profiler
  - Sampling
  - 32-bit
  - https://www.delphitools.info

Commercial profilers

- AQTime
  - Instrumenting and sampling
  - 32- and 64-bit
  - https://smartbear.com/
- Nexus Quality Suite
  - Instrumenting
  - 32- and 64-bit
  - https://www.nexusdb.com
- ProDelphi
  - Instrumenting (source)
  - 32- and 64-bit
  - http://www.prodelphi.de/

Fixing the Algorithm

Fixing the algorithm

- Find a better algorithm
- If a part of program is slow, don’t execute it so much
- If a part of program is slow, don’t execute it at all

“Don’t execute it so much”

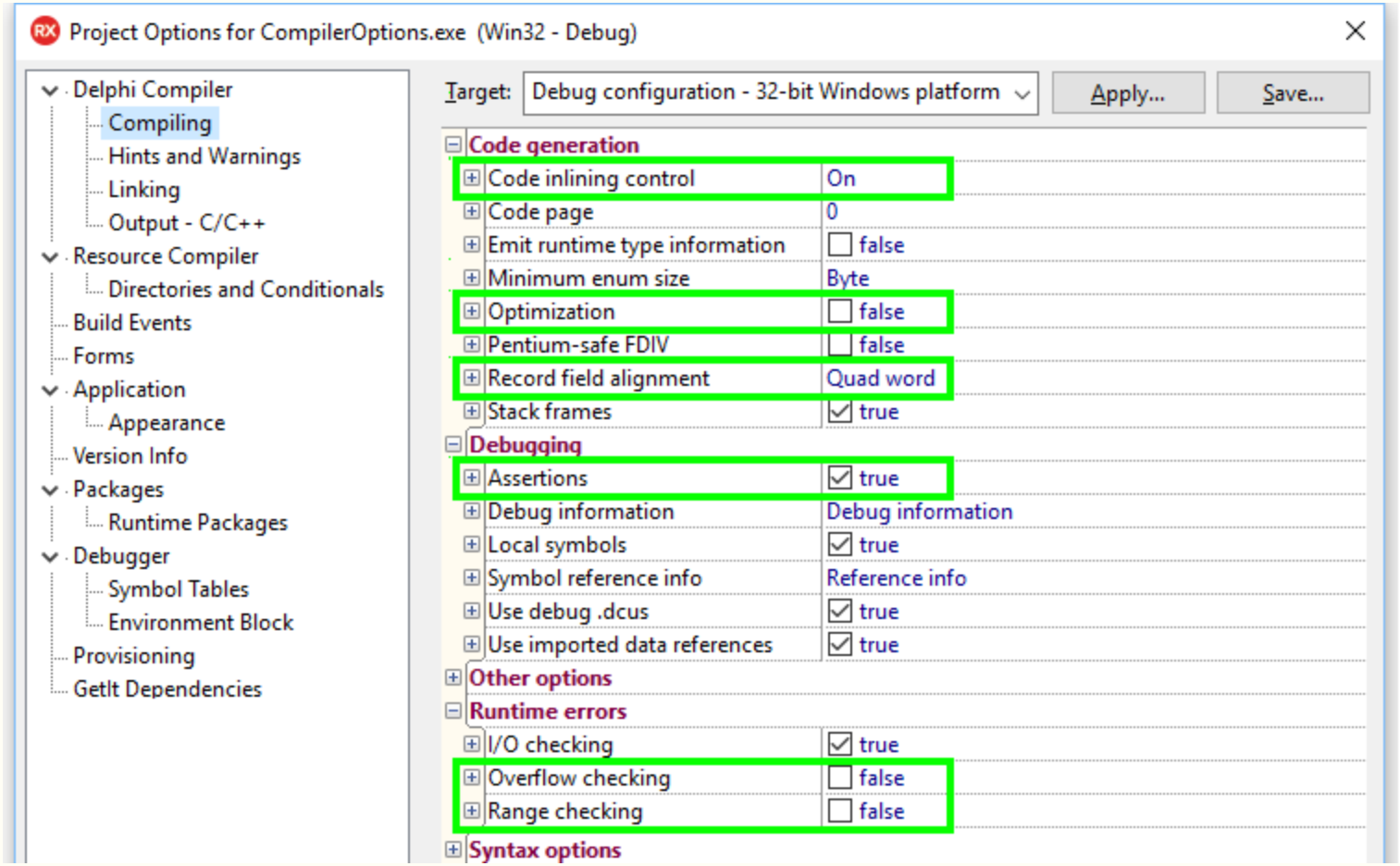
- Don’t update UI thousands of times per second
- Call BeginUpdate/EndUpdate
- Don’t send around millions of messages per second

“Don’t execute it at all”

- UI virtualization
  - Virtual listbox
  - Virtual TreeView
- Memoization
  - Caching
  - Dynamic programming
  - TGpCache<K,V>
    - O(1) all operations
    - GpLists.pas, https://github.com/gabr42/GpDelphiUnits/

Fine Tuning the Code

Compiler settings



Behind the scenes

- Strings
  - Reference counted, Copy on write
- Arrays
  - Static
  - Dynamic
    - Reference counted, Cloned
- Records
  - Initialized if managed
- Classes
  - Reference counted on ARC
- Interfaces
  - Reference counted

Calling methods

- Parameter passing
  - Dynamic arrays are strange
- Inlining
  - Single pass compiler!

Mastering Memory Manager

58 memory managers in one!

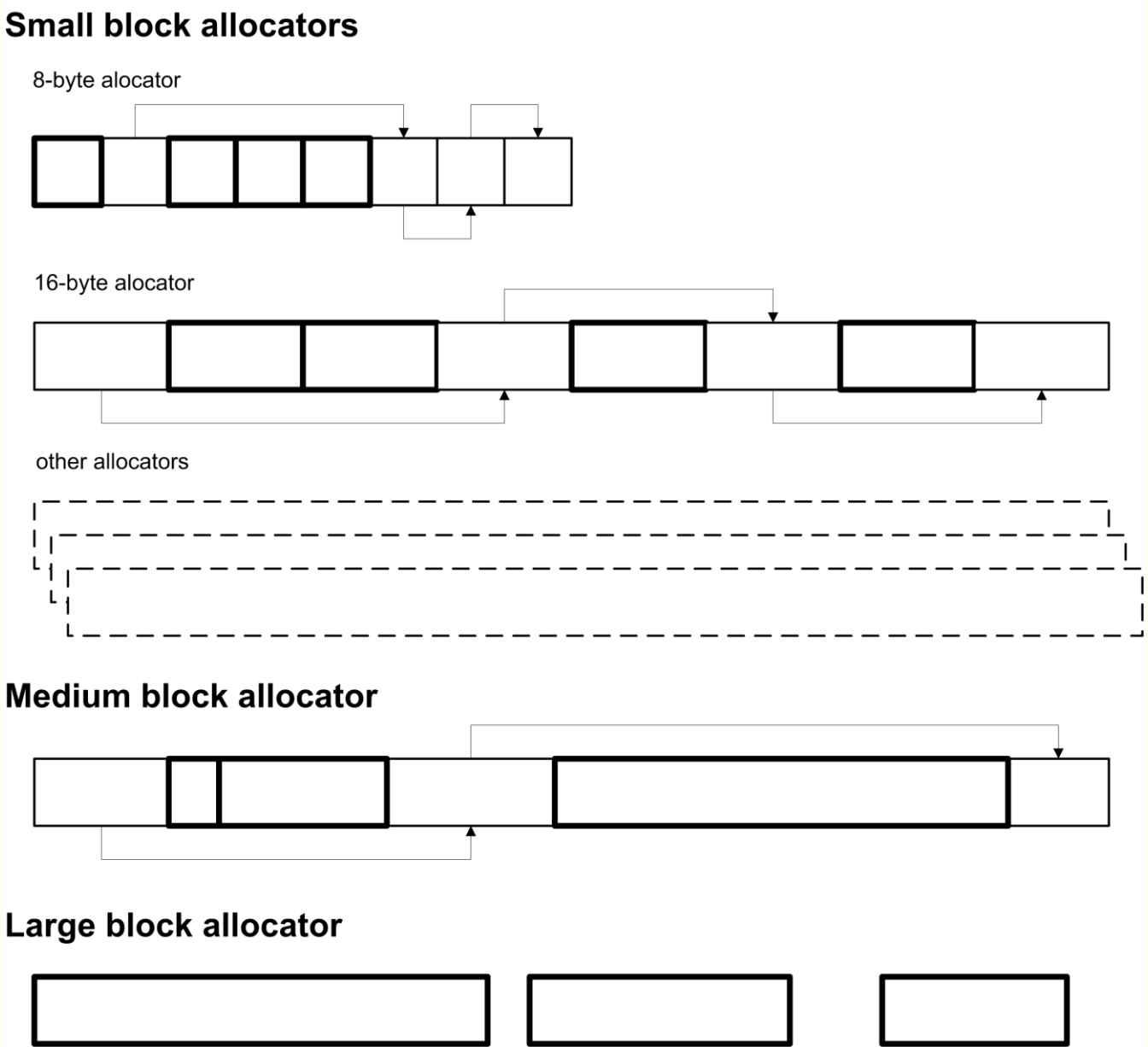


Image source: ‘Delphi High Performance’

Optimizations

- Reallocation
  - Small blocks: New size = at least 2x old size
  - Medium blocks: New size = at least 1.25x old size
- Allocator locking
  - Small block only
  - Will try 2 ‘larger’ allocators
  - Problem: Freeing memory

```
allocIdx := find best allocator for the memory block
repeat
  if can lock allocIdx then
    break;
  Inc(allocIdx);
  if can lock allocIdx then
    break;
  Inc(allocIdx);
  if can lock allocIdx then
    break;
  Dec(allocIdx, 2)
until false
```

Optimizing parallel allocations

- FastMM4 from GitHub
  - https://github.com/pleriche/FastMM4
- DEFINE LogLockContention
- DEFINE UseReleaseStack

Alternatives

- ScaleMM
  - https://github.com/andremussche/scalemm
- TBBMalloc
  - https://www.threadingbuildingblocks.org
  - https://sites.google.com/site/aminer68/intel-tbbmalloc-interfaces-for-delphi-and-delphi-xe-versions-and-freepascal
  - http://tiny.cc/tbbmalloc

Writing Parallel Code

When to parallelize?

- When other means are exhausted

- Pushing long operations into background
  - “Unblock” the user interface
- Supporting multiple clients in parallel
- Speeding up the algorithm
  - Hard!

Common problems

- Accessing UI from a background thread

“Never access UI from a background thread!”

“Never access UI from a background thread!”

Common problems

- Accessing UI from a background thread
- Reading/writing shared data
  - Structured data
  - Simple data

Synchronization

- Critical section
- Spinlock
- Monitor
- Readers-writers / MREW / SWMR
  - TMREWSync / TMultiReadExclusiveWriteSynchronizer
    - Terribly slow
  - Slim Reader/Writer (SRW)
    - Windows only
    - http://tiny.cc/winsrw

Synchronization problems

- Slowdown
- Deadlocks

Interlocked operations

- “Microlocking”
- Faster
- Limited use

Communication

Communication

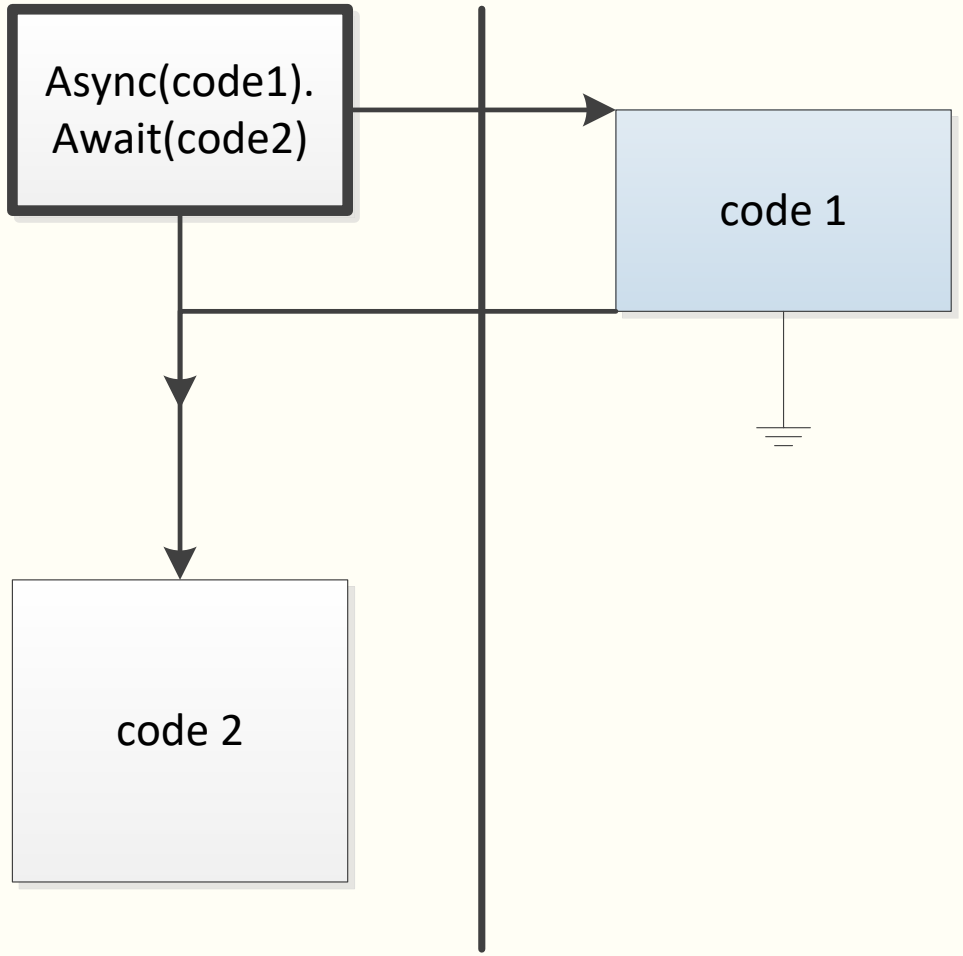
- Windows messages
- TThread.Queue
  - TThread.Synchronize too, but …
- polling

Parallel Patterns

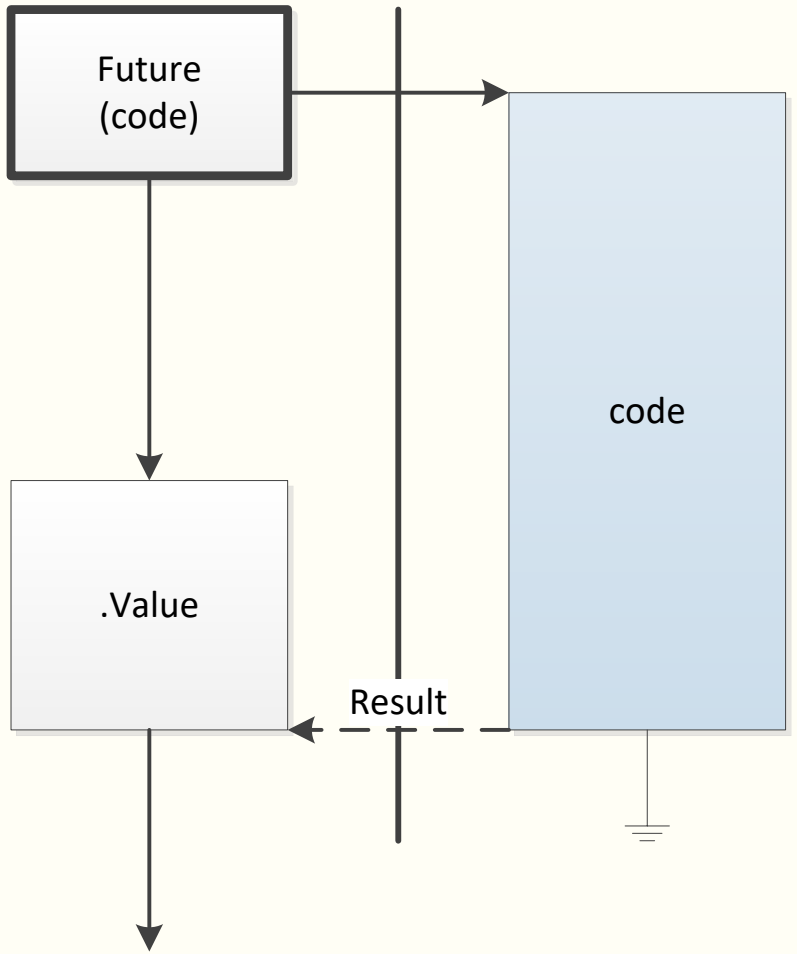
Patterns

- Async/Await
- Join
- Future
- Parallel For
- Pipeline
- Map
- Timed task
- Parallel task
- Background worker
- Fork/Join

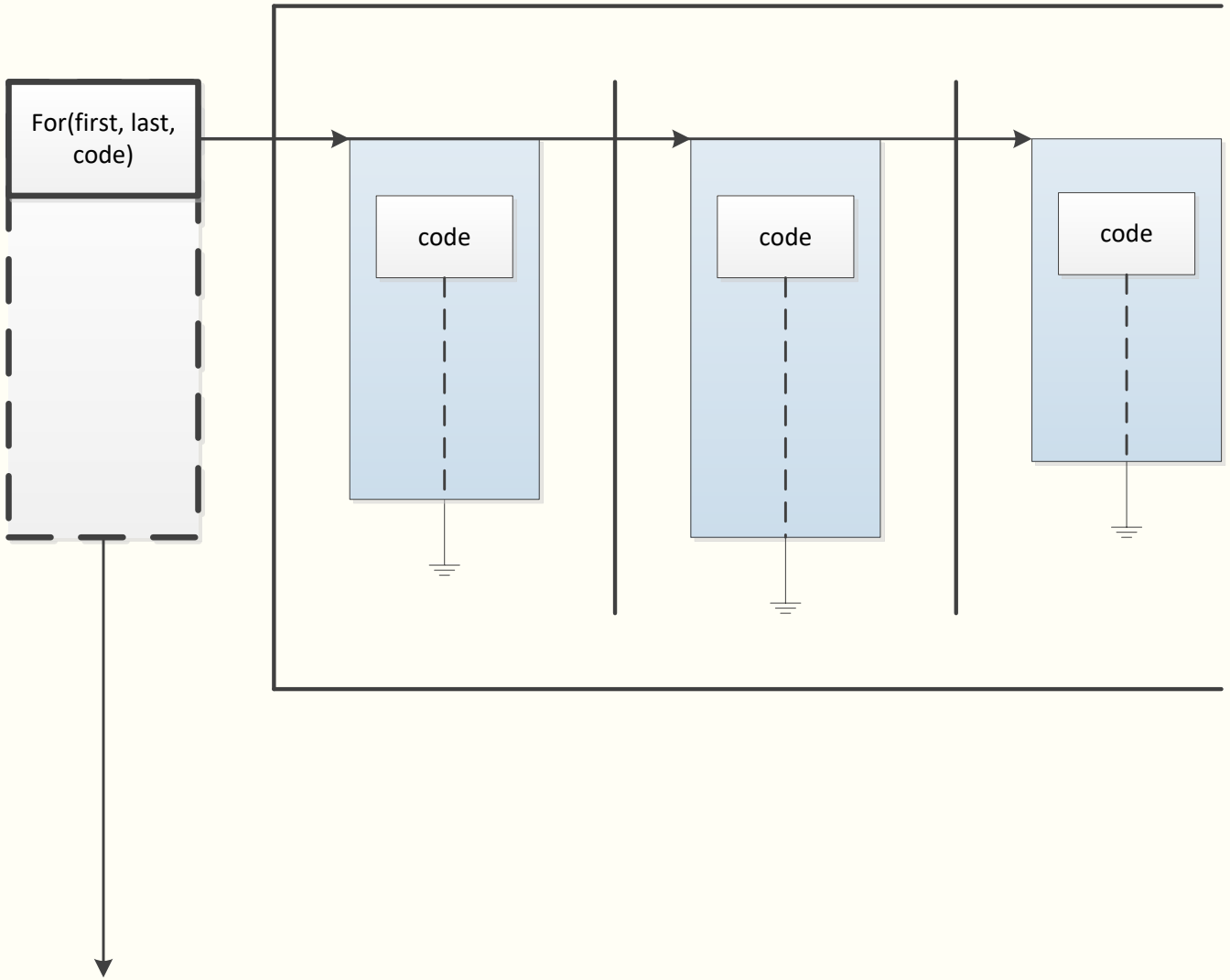
Async / Await



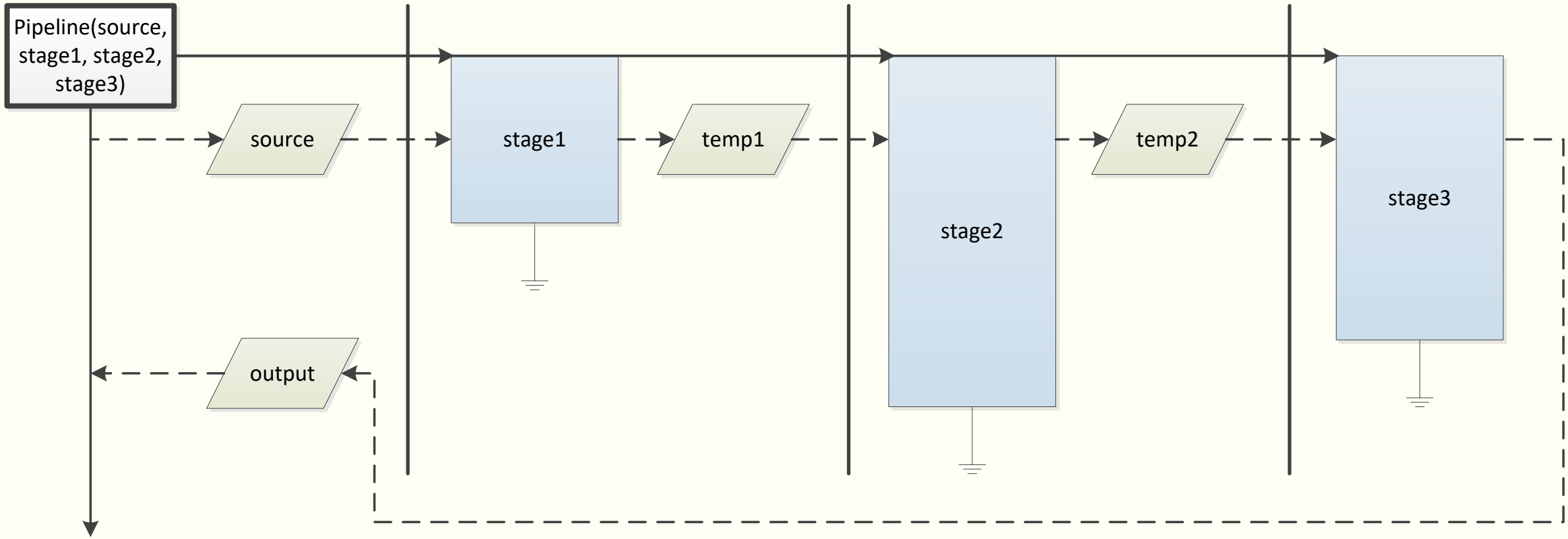
Future



Parallel For



Pipeline


Pipeline(source, stage1, stage2, stage3)
source
stage1
temp1
stage2
temp2
stage3
output

High Performance in a Nutshell

Steps to faster code

1. Understand the problem. What do you want to achieve?
2. Find the problematic code. Measure!
3. If possible, find a better algorithm.
4. If that fails, fine tune the code.
5. As a last resort, parallelize the solution.
   Danger, Will Robinson!
6. Find/write faster code in a different language and link it into the application.

It's Question Time!