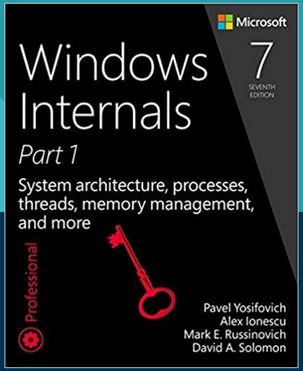
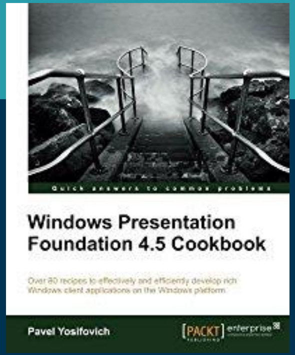
Building a .NET Cross Platform Profiler (in an hour)

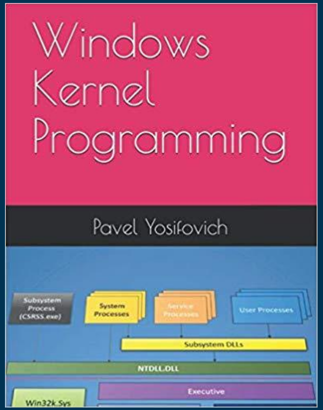
Pavel Yosifovich
@zodiacon

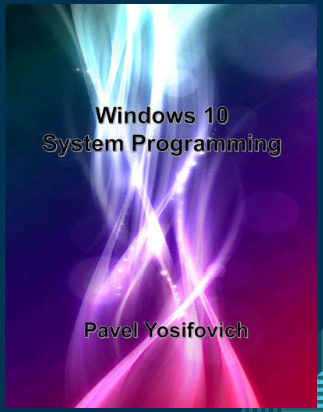
About Me

- Developer, Trainer, Author and Speaker
- Book author
  - “Windows Kernel Programming” (2019)
  - “Windows Internals 7th edition, Part 1” (co-author, 2017)
  - “Windows 10 System Programming” (WIP)
- Pluralsight author
- Author of several open-source tools (http://github.com/zodiacon)
- Blogs: http://blogs.microsoft.co.il/pavely, http://scorpiosoftware.net









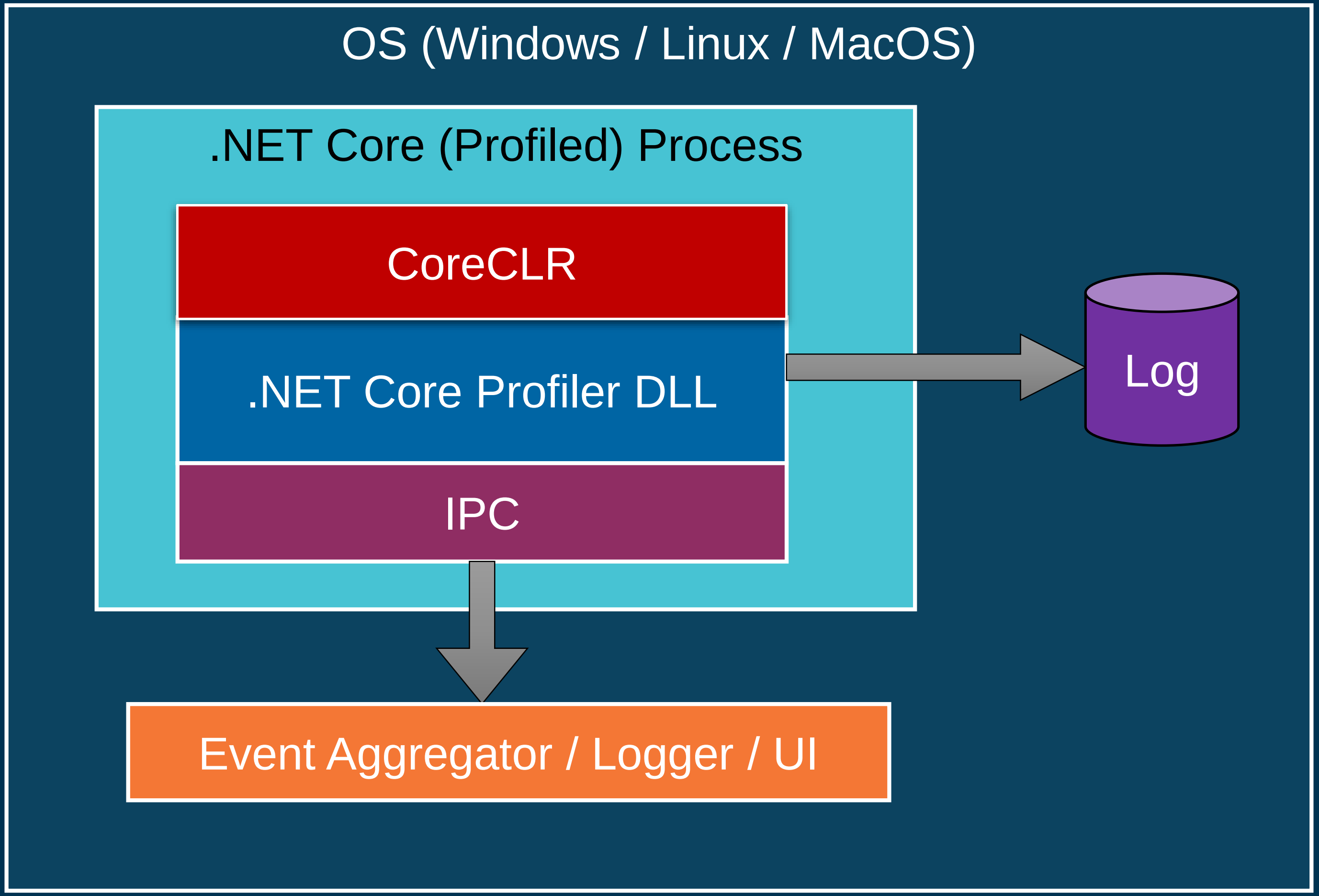
Agenda

- Overview
- Profiler Architecture
- Loading a Profiler
- Implementing a Profiler
- Summary

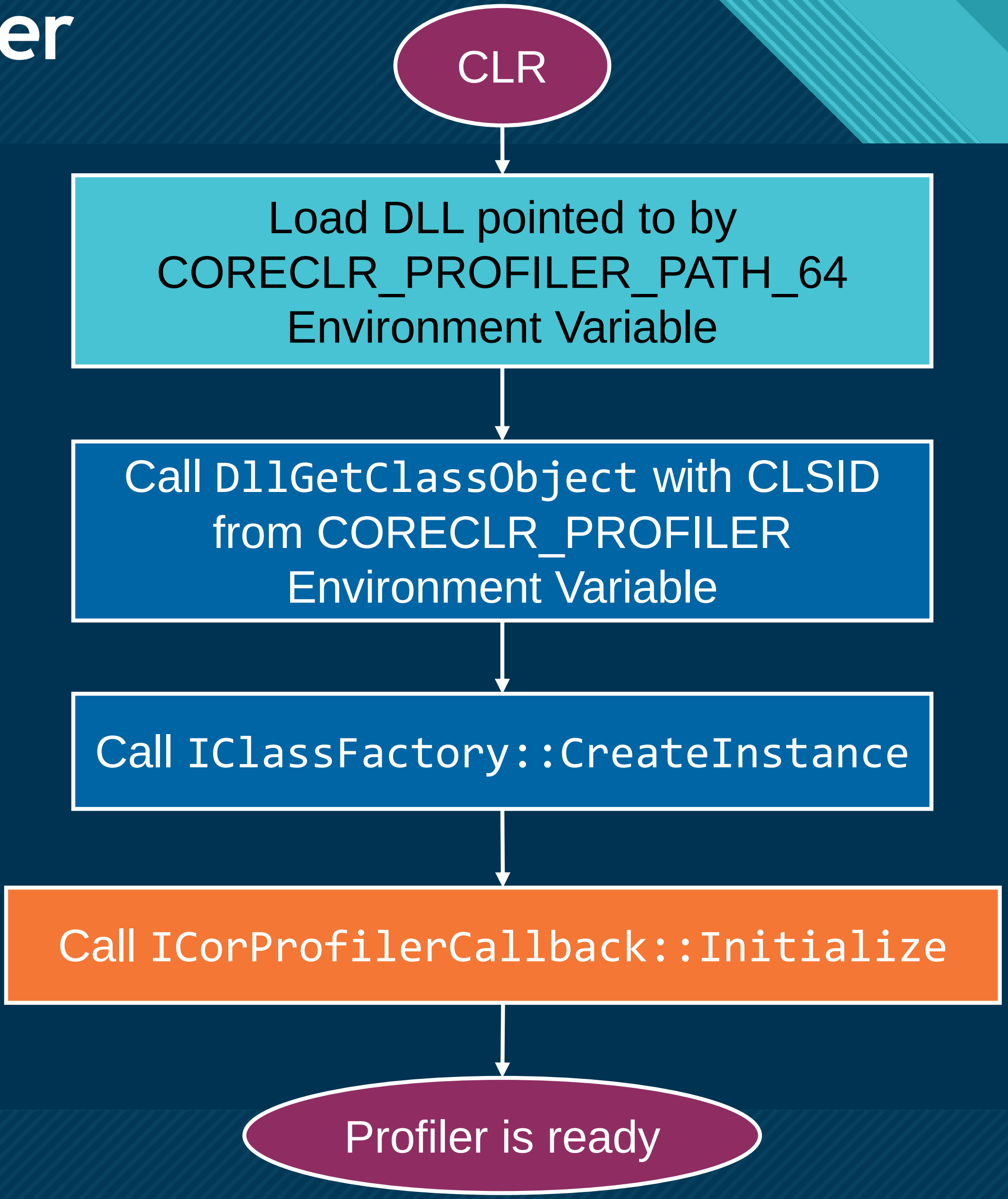
Overview

- The .NET CLR supports loading an instrumentation profiler
  - COM class implemented in C++ and hosted in a DLL
  - Windows only
- The .NET Core CoreCLR supports the same model
  - With the same interfaces
  - Windows / Linux / MacOS
- We’ll built a simple, yet functional, profiler

Profiler Basics


OS (Windows / Linux / MacOS)
.NET Core (Profiled) Process
CoreCLR
.NET Core Profiler DLL
IPC
Log
Event Aggregator / Logger / UI

Loading a Profiler


CLR
Load DLL pointed to by CORECLR_PROFILER_PATH_64 Environment Variable
Call `DllGetClassObject` with CLSID from CORECLR_PROFILER Environment Variable
Call `IClassFactory::CreateInstance`
Call `ICorProfilerCallback::Initialize`
Profiler is ready

The Profiler and COM

- A profiler is not loaded by calling **`CoCreateInstance`**
  - Requires a threading model to be set
- COM is a Windows technology
- For non-Windows platforms, Microsoft created the Platform Adaptation Layer (PAL)
  - Insulates the profiler developer from platform differences as far as COM and CLR are concerned
- The PAL is currently bundled with the CoreCLR source code itself

Project Structure and Build

- Using CMake
  - CoreCLR uses CMake as its build system
  - VS support for CMake is not good enough (IMHO)
- Using MSBuild
  - Shared items project holds majority of code
  - Specific projects for Windows and Linux
- Compiler for Linux
  - Use Clang (not gcc)
    - Supports some Microsoft extensions

Testing

- Set up environment variables
- Launch application
- Linux testing options (assuming developing on Windows)
    - Deploy and run on a Linux VM
    - Use the Windows Subsystem for Linux (WSL)
        - Requires windows 10 version 1607 and later

Environment Variables

- CORECLR_ENABLE_PROFILING=1
- CORECLR_PROFILER={ProfilerGuid}
- CORECLR_PROFILER_PATH_64={64bitProfilerPath}
- CORECLR_PROFILER_PATH_32={32bitProfilerPath}

Code!

Summary

- .NET Core is cross-platform
  - So is a .NET Core profiler
- Use of PAL and standard C++ can help cope with platform differences
- Profiling is just one part of the job
  - The other is actually making good use of the gathered data

Resources

- CLR Profiler samples on Github
- David Broman’s CLR Profiling Blog
- Old CLR Profiler on Github

Q&A

Thank You!