Samsung R&D Institute Russia
Center of System Software

Challenges in debugging dynamically compiled languages
as exemplified by C# debugger for Tizen

Dmitri Botcharnikov

Agenda

- Dynamically compiled languages
- Tizen .NET
- Debugging Challenges
- Tizen .NET Debugger internals
- Future plans

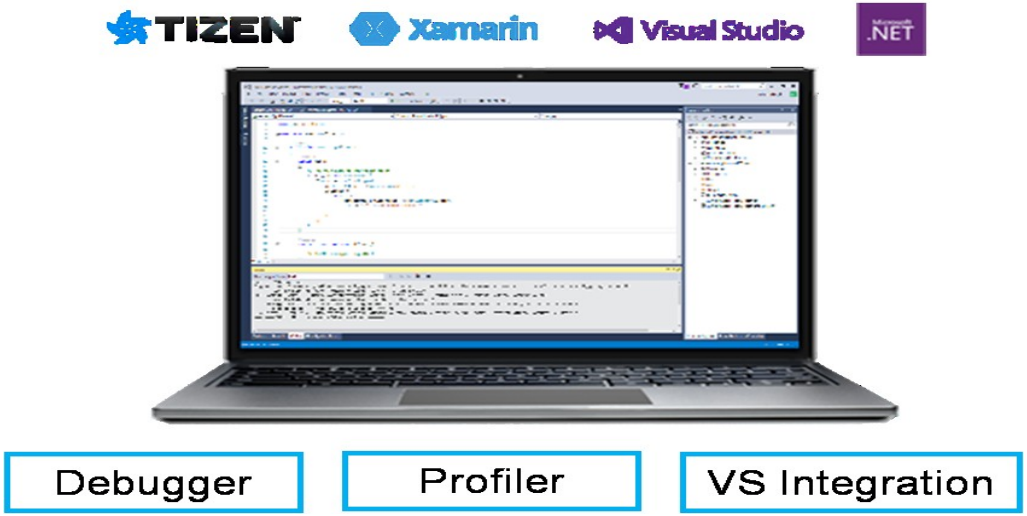
Dynamically compiled languages

- Dynamically (Just-In-Time) compiled languages
- VM manages low-level details: memory allocation, exception handling
- But for debuggers…

Tizen .NET

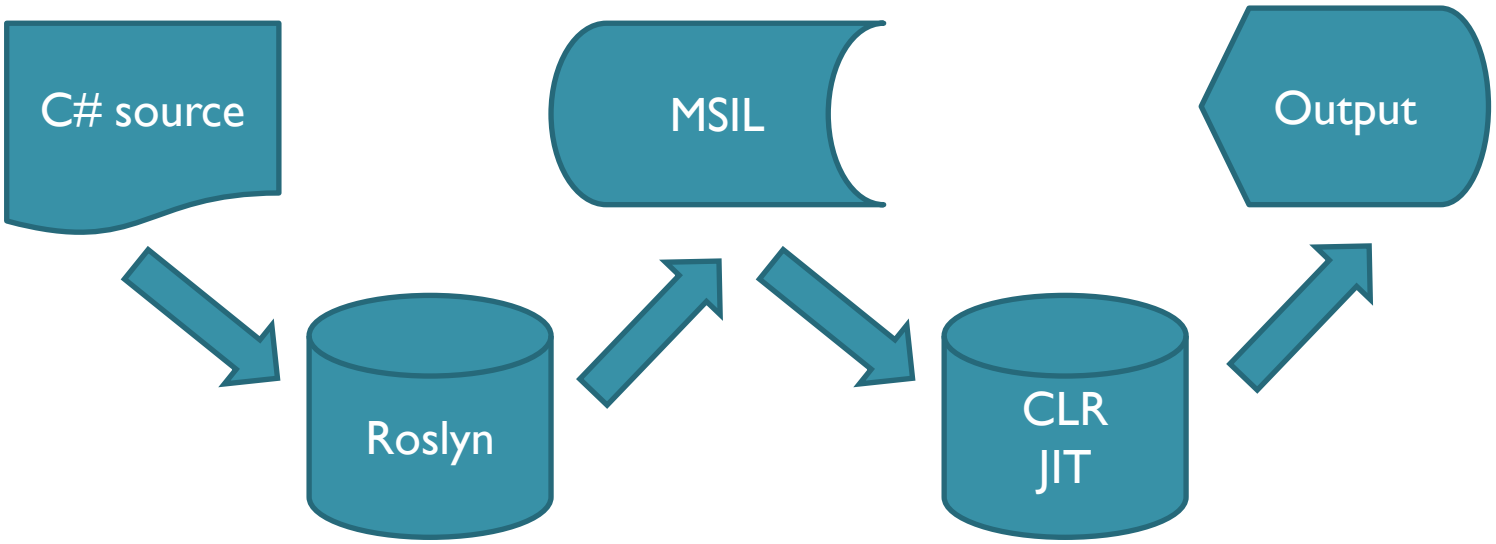
- Visual Studio Tools for Tizen preview were released
- C# was added to Tizen
- Over 1,400,000 C# developers worldwide
- Tizen running on 50 millions Samsung devices (TV, wearables, mobile, IoT)
- http://developer.tizen.org

Technologies



- Tizen OS (emulator, platform-specific APIs)
- Xamarin.Forms
- .NET Core (CoreCLR, CoreFX, Roslyn)
- Visual Studio 2015 (Windows)

C# Compilation & Execution



- Language-specific compiler: C# => MSIL
- CLR JIT compiler: MSIL => native code

Debugging Challenges

- Source code to native code mapping
  - C# compiler generates debugging information for source code to MSIL mapping
- Stepping in and over
  - Stepping into not yet compiled code
  - Managed exception handlers
  - Lambdas, closures & iterators
- Local variables & arguments inspection
  - C# compiler generates debugging information for MSIL variables

LLDB

- Subproject of LLVM (http://lldb.llvm.org)
- Native debugger builds on LLVM and Clang libraries
- Supports X86 and ARM architectures

SOS debugger plug-in

- Plug-in for LLDB (libsosplugin.so, libsos.so)
- Port of SOS.dll (SOS Debugging extension) to Linux platform
- Provides low-level information about internals of CLR environment
- Useful for CoreCLR developers, but not so for application developers

GDB JIT Interface

- Interface for registering JITed code with debuggers
- Initially designed for GDB, now supported by LLDB
- VM should construct in-memory ELF+DWARF image and call predefined function
  - __jit_debug_register_code
- Debugger puts breakpoint on this function
- On breakpoint hit loads constructed image and resume execution

GDB JIT: Pro & Cons

- Pro
  - Supported by both GDB and LLDB
  - Integrated into debugger infrastructure
  - The easiest way to add support for JITed language
- Cons
  - Invasive (only needed for debugging)
  - Memory consuming (~700 b on ARM, ~1kb on x86_64)
  - Inherently static: generated before execution

Stepping over and in

- Stepping in and over
  - Stepping into still not compiled code
  - Managed exception handlers: stack unwinding
  - Lambdas, closures & iterators
- CoreCLR implements calls through stubs dispatch which is dynamically changed
- Solution
  - Generate symbols for stubs in GDB JIT in-memory image
  - Modify LLDB thread plans to follow these symbols

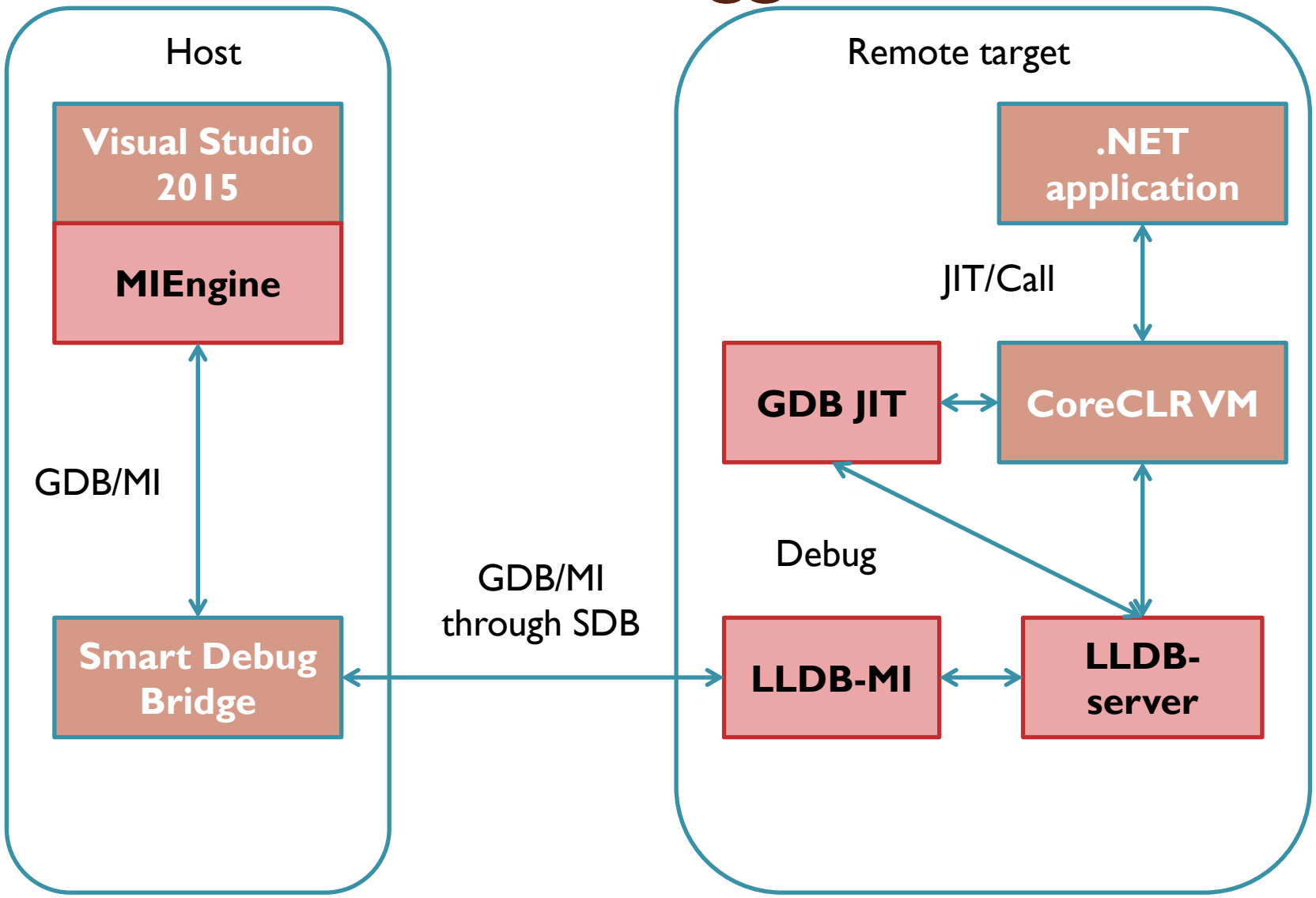
GDB/MI & Microsoft MIEngine

- GDB/MI: machine oriented text interface
- Supported by Eclipse CDT, Emacs & others
- Visual Studio MI Debug Engine is an open source VS extension that provides support for GDB/MI
- Modified to support Tizen Application Framework

Historical debugging PoC

- Allows you to move backward and forward through the execution of your application and inspect its state
- Implemented in CoreCLR through ICorProfiler interface
- Requires implementation of platform-specific profiler hooks (OS + arch)
- Developed Proof-of-Concept realization for ARM & x86_64 Linux

Tizen .NET Debugger



Future plans

- Develop C# language type plug-in for LLDB
  - Get LLDB knows about C# type system
- Develop .NET runtime support plug-in for LLDB
  - Generic instantiation types available during method execution
  - Better support for CoreCLR stubs
- Develop full-fledged Historical debugger

Thank you!