

Towards a HiPEAC Virtualisation Framework

Jonas Maebe

(with much input from Bjorn De Sutter & Albert Cohen)

June 4th, 2009



Challenge: optimisation vs. flexibility

- Fully specialising software is hard
 - Take full advantage of HW features
- Compiled software has limited adaptability
 - HW paras & variability, input, attacks
- Software layers cause overhead

Solution: virtualisation

- Distribute intermediate code, specialise at load/run time
 - Split compilation
 - Use run-time information (hw & sw)
- Include lots of metadata
- Integrate with system-level virtualisation

Our goals

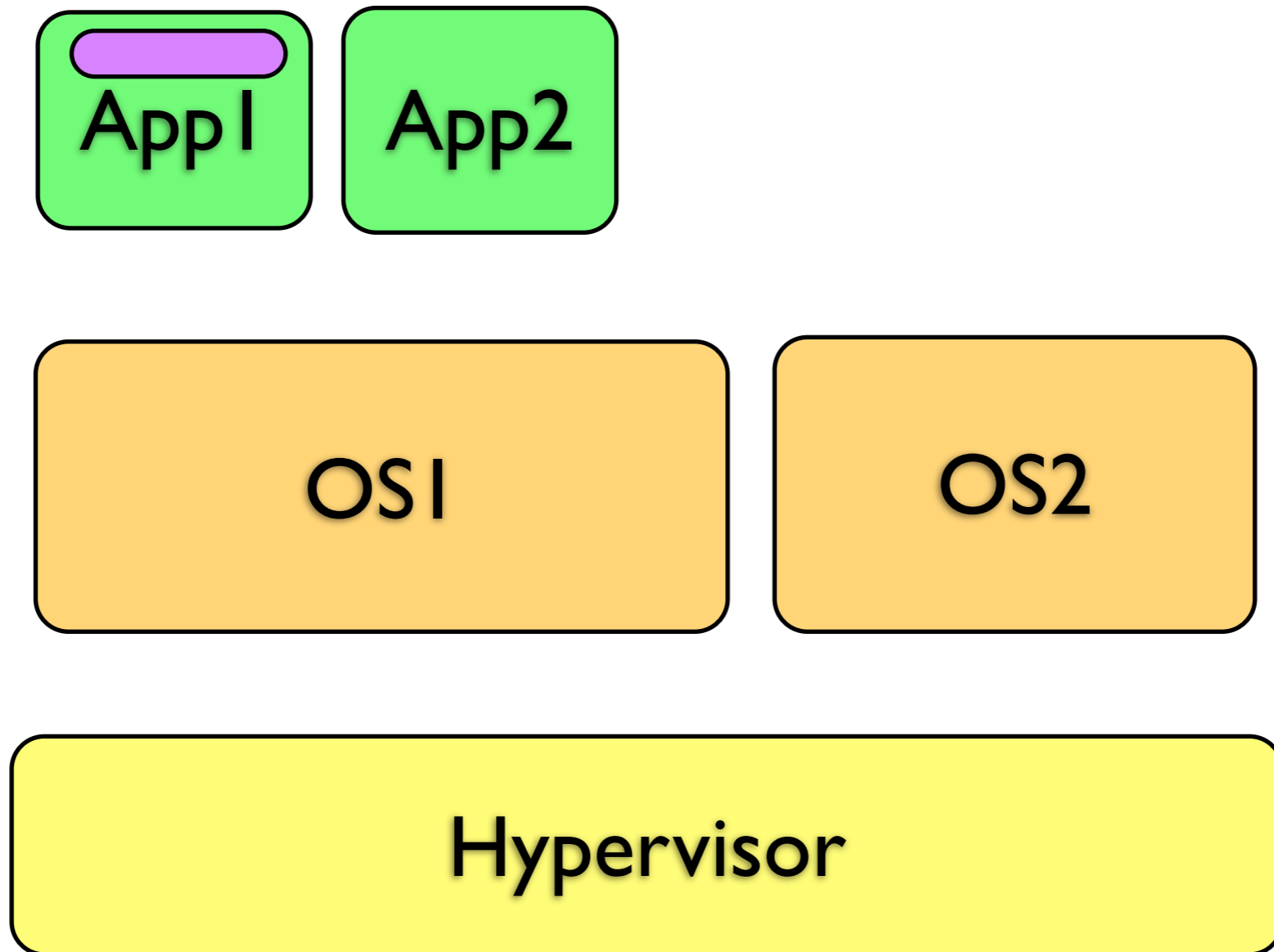
- Define HiPEAC virtualisation research focus
- Create common research infrastructure
 - Effective target-dependent optimisations
 - Flexible dynamic compilation (wpo, fb)
 - Low and high latency environments?
- Portable performance

Basis: GCC / CIL / Mono

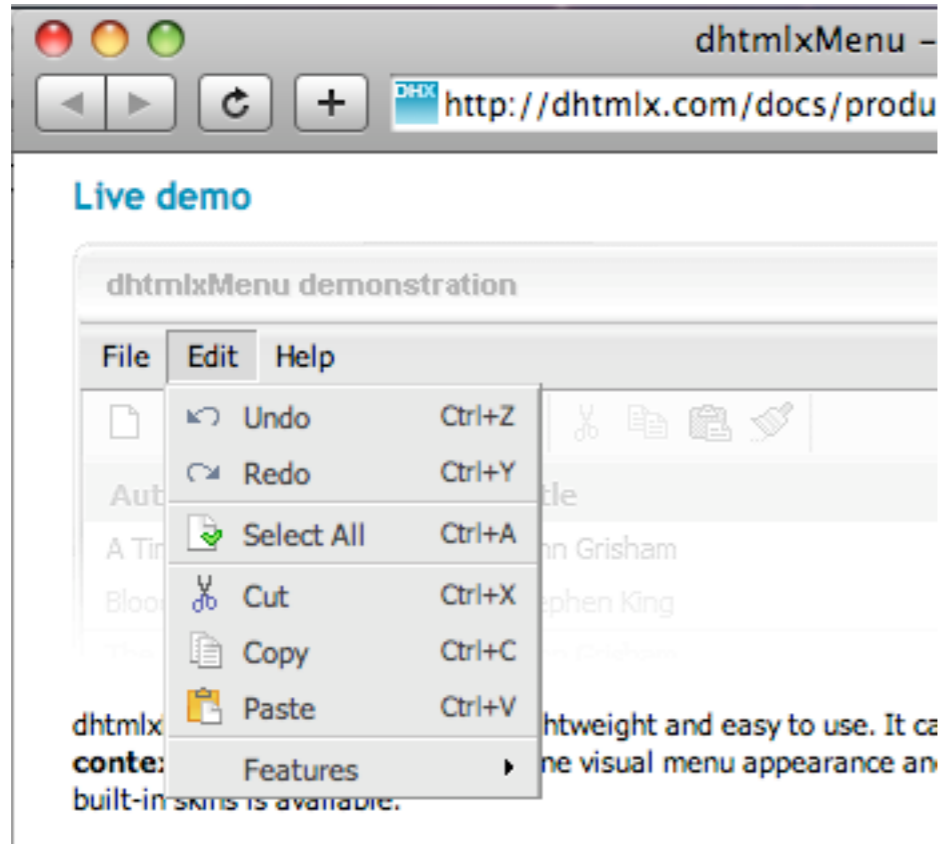
- Reasons
 - Extensible, proven format and tools
 - Existing expertise in HiPEAC
 - Interest by HiPEAC member companies (ST, IBM, NXP)
 - Available for ARM/PPC/x86/... and Linux/Windows/Mac OS X/...

- Execution context
- Frameworks/techniques overview
- Summary

Virtualisation level



Execution types



FFT

- Low latency
- Unpredictable

- Overall efficiency more important
- More predictable

- Execution context
- **Frameworks/techniques overview**
- Summary

Intermediate code

Framework	Code format	Properties
Java	Java bytecode	stack-based Java-specific <i>(Da Vinci Machine)</i>
CLR	CIL	stack-based generic (C++) rich type info
LLVM	LLVM IR	SSA-based low level type info ($\emptyset\emptyset\emptyset$)

Metadata

Framework	Metadata form	Granularity
Java	interface <i>(attribute)</i>	class, method, field
CLR	class <i>(annotation)</i>	assembly, class, ... , method, field, para
LLVM	free form string <i>(annotation)</i>	symbolic address, source line

Mono (CIL/CLI/CLR)

- Implements CLI and .NET 2.0
- JIT/regular compiler for all archs/OSes
 - SSA-based optimisation framework
- OS-level: MS Singularity project
- Moonlight

Java

- JikesRVM, OpenJDK, Android/Dalvik, ...
- Many advanced JIT compilers and garbage collectors
- Limited to Java semantics and managed code
- But: Da Vinci Machine Project

LLVM

- Own IL (LLVM IR), lower-level than CIL
- JIT/non-JIT compiler for all archs/OSes
 - SSA-based IR and optimisation framework
- OS-level: Toskana-VM (L4 running LLVM IR Linux kernel)

dotGNU

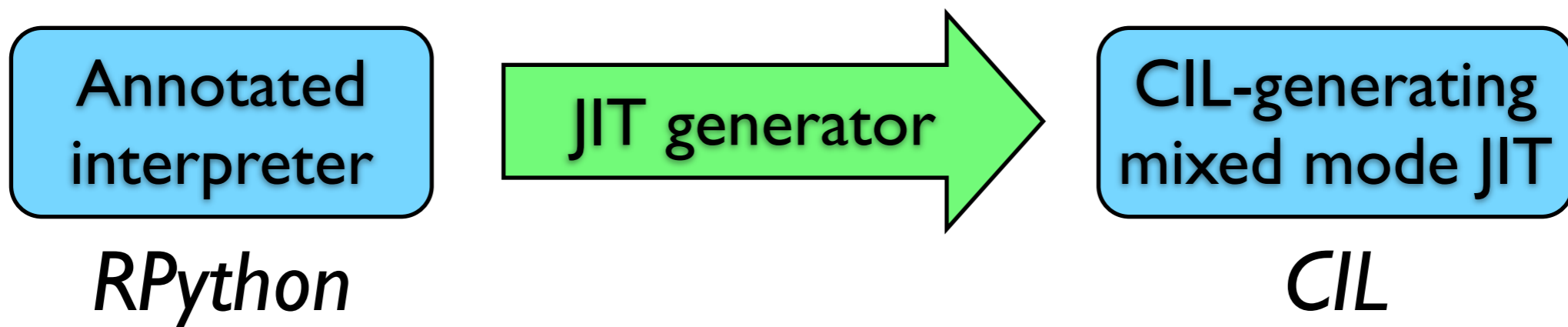
- Re-implementation of .NET in Free Software
- Some components/related projects:
 - Class library (Portable.NET)
 - libjit: generic low-level assembler/code gen.
 - ILDJIT: GPL JIT compiler for .NET
 - Use spare cores for predictive compilation

Portmeirion

- Sun project to co-design HW/virtualised software stack
- Whole-system optimisation (“inline system call in Java application”)
- Adapters for legacy code
- Early planning stage last year, now: unknown

PyPy

- Originally JIT for Python, now more generic
- Switches between interpreting and JITing
- Interesting structure:



Javascript

- WebKit Squirrelfish, Google V8, Tamarin
 - Polymorphic inline cache (SELF)
 - Lightweight JIT
- Mozilla Tracemonkey
 - Build Trace Tree
 - Lightweight JIT optimises hot traces

A few more

- IronPython/Ruby: translate into CIL (part statically, part dynamically)
- Parrot
 - Own JIT, focusses on dynamic languages
 - 2 intermediate formats (low/higher level)
- ST: PVM, Linear Assembler Optimizer

- Execution context
- Frameworks/techniques overview
- **Summary**

Summary

Bytecode

- Different levels? (polyhedral, stack, ssa, ...)
 - Split compilation
 - Via metadata
- Metadata
 - Structured vs. unstructured
 - Granularity (feature-specific, pluggable?)

Summary

OS-level virtualisation

- Realistic:
 - Singularity
 - Toskana-VM
 - Cosmos
 - (Portmeirion)
- Performance: little or no data