Compilers and Tools for Software Stack Optimisation

EJCP 2014

2014/06/20

christophe.guillon@st.com

Outline

Compilers for a Set-Top-Box

Compilers Potential

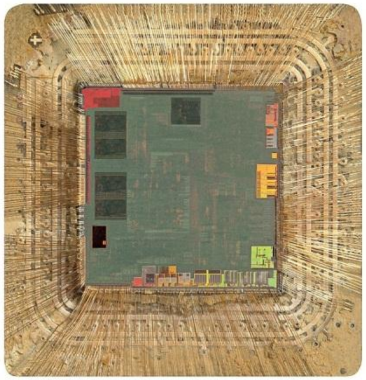
Auto Tuning Tools

Dynamic Program instrumentation

System calls interposition

COMPILERS FOR A SET-TOP-BOX

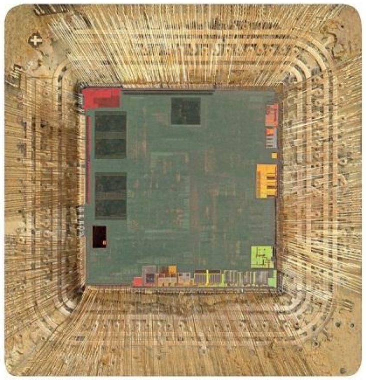
Set-Top-Box at Home



System on Chip (SoC)

Video Encode & Decode
Audio Encode & Decode
Connectivity and Storage
Drivers and Application Middlewares
Rich Content Applications

Set-Top-Box System on Chip



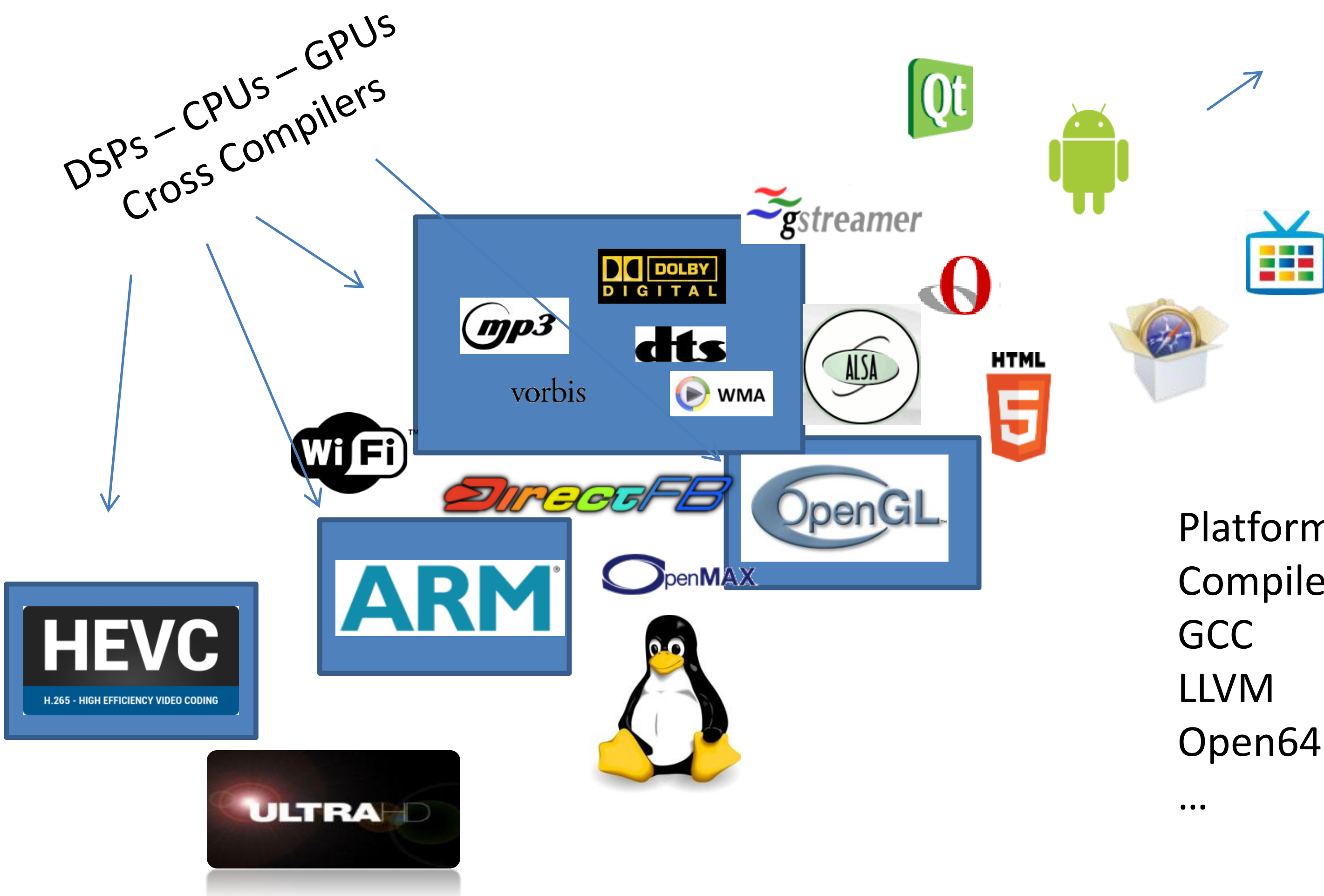
SoC / ASICs / DSPs – CPUs – GPUs

Applications

Kernel – Drivers – Middlewares – Browsers - Frameworks

~20 programmable cores
2x ARM
4x ST231
2x STxP70
9 x SLIM
1 ARM Mali …

Cross Compilers



Applications

Platform Development Compilers:

GCC

LLVM

Open64

…

Embedded Compilers

OpenGL – OpenCL
Kernel - JavaScript – Java – ActionScript
Dynamic Compilers

Applications

Runtime Compilers:
OpenCL/LLVM - AOT
OpenGL ARM - AOT
Kernel opts - AOT
V8 JSC - JIT
SFX JSC - JIT
Adobe ASC - JIT
Dalvik JC - JIT
HotSpot JC - JIT
Android NCI - AOT
…

Other Compilers

Platform and Processor Emulation Compilers

Applications

Emulation Tools Compilers:
QEMU (DBT)
Valgrind (DBT)
ARM FastSIM (SIM)
HCE Compilers
…

x86 Host

COMPILERS POTENTIAL

Compilers Potential and Cost

When?

- Static
- Ahead of Time
- Just In Time

Increasing Knowledge

Increasing Runtime Cost

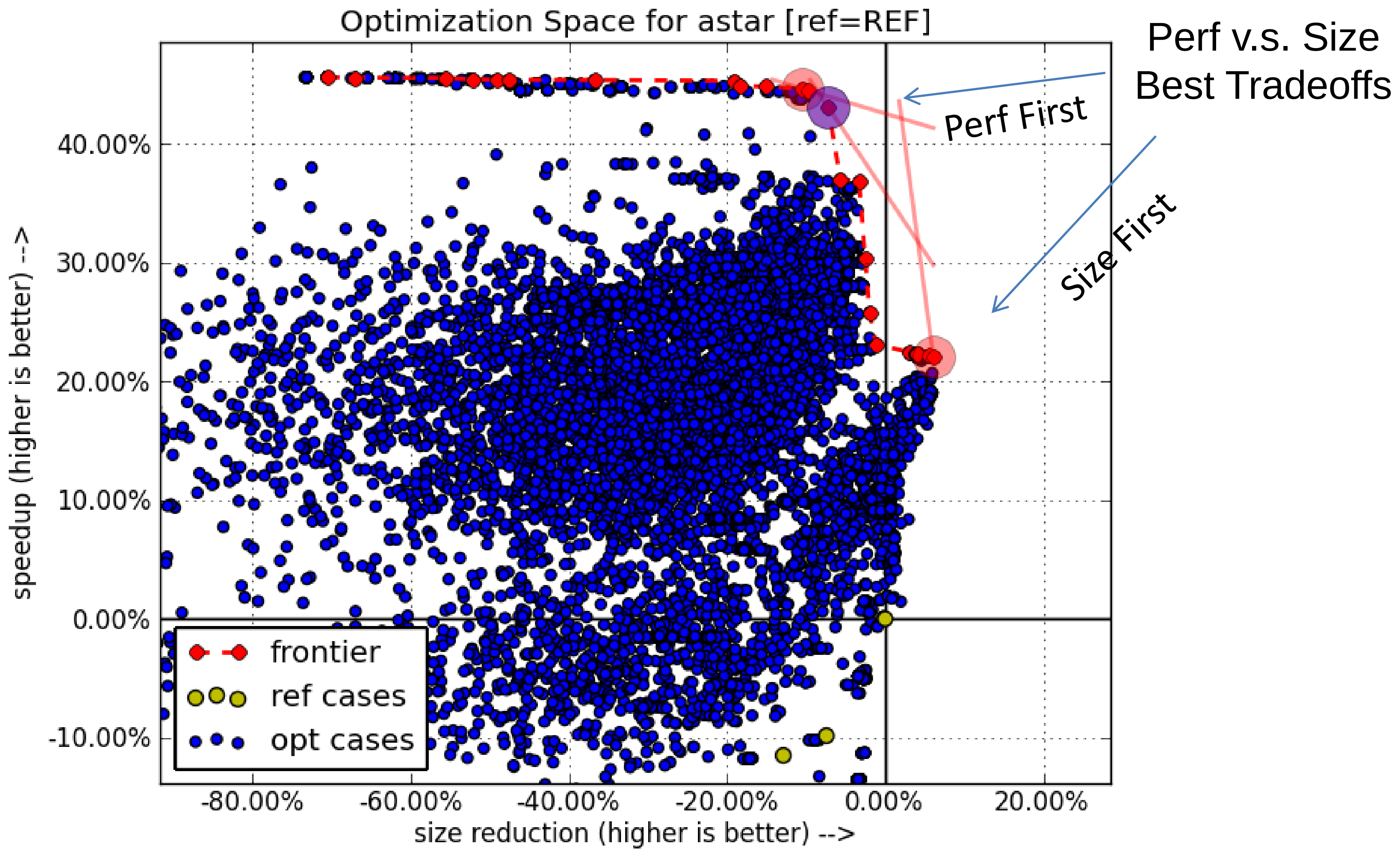
How Often?

- One shot
- Iterative
- Continuous

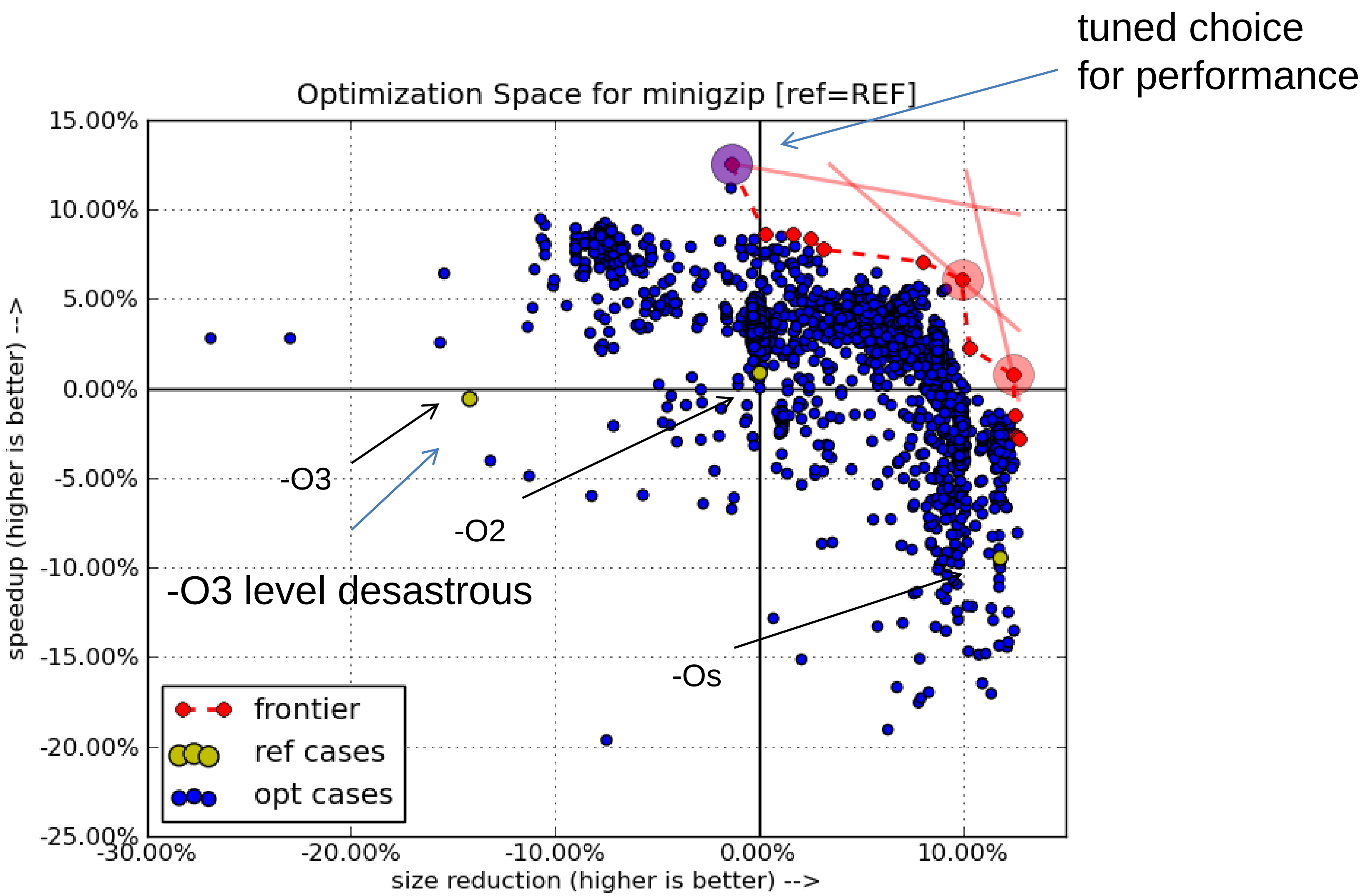
Increasing Potential

Increasing Potential Cost

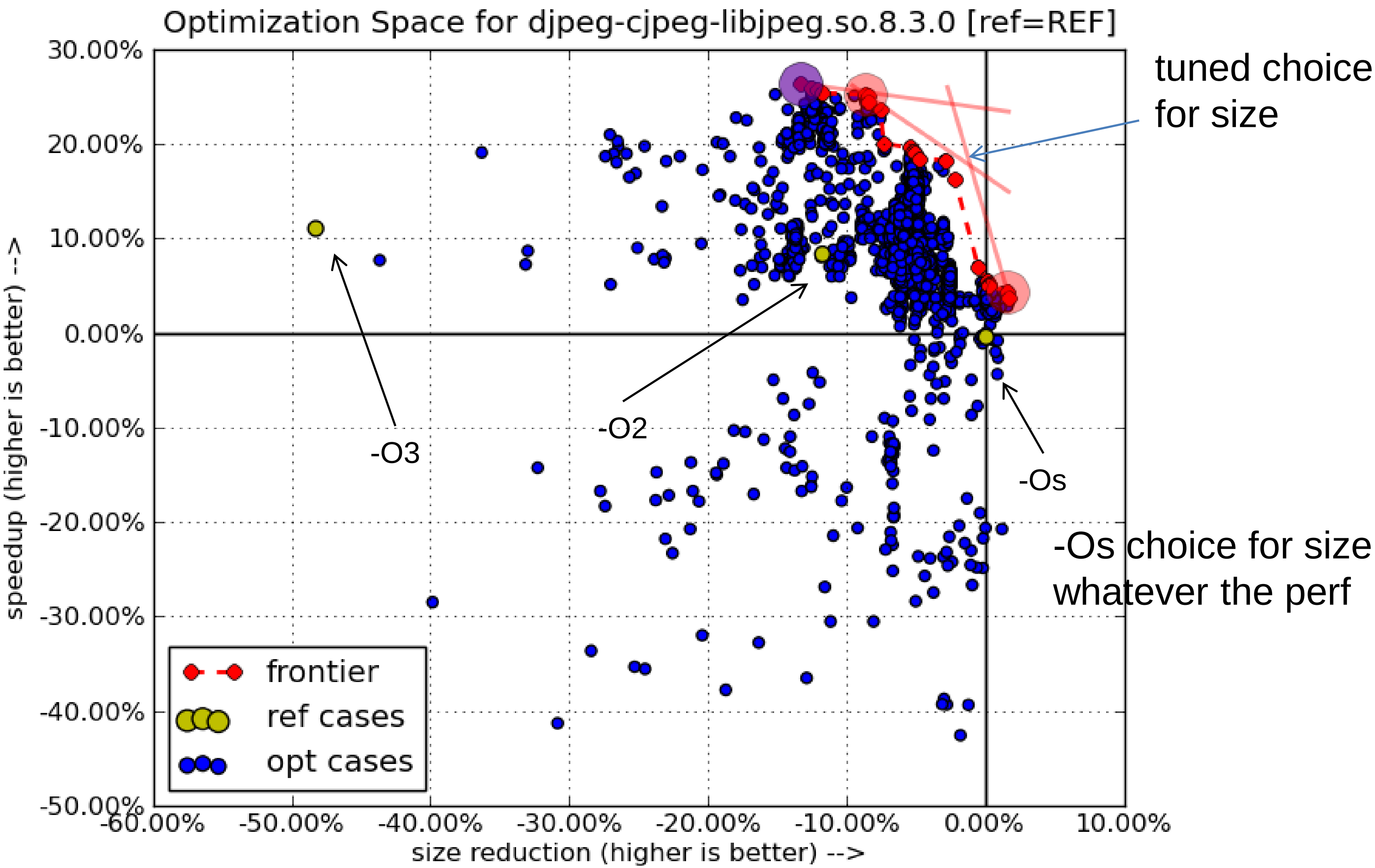
Optimizations Space Exploration



Defaults are Uneffectives



Defaults Choices are Limited



Optimizing Applications

Choose the right domain specific language

Fallback to C/C++ ☺

Ensure Functionality and Unitary Testing at First

Develop Representative Benchmarks

Profile against the Benchmarks

Use your Domain Expertise

Use your Target Architecture Expertise ☹

Exploit the Compiler potential ☹

Iterate, but preserve maintainability

Should you care?

AUTO TUNING TOOLS

Motivations

Automate the process of optimization

Build, run, observe, iterate, …

Complement programmer’s optimisation

Reproduce as changes are made

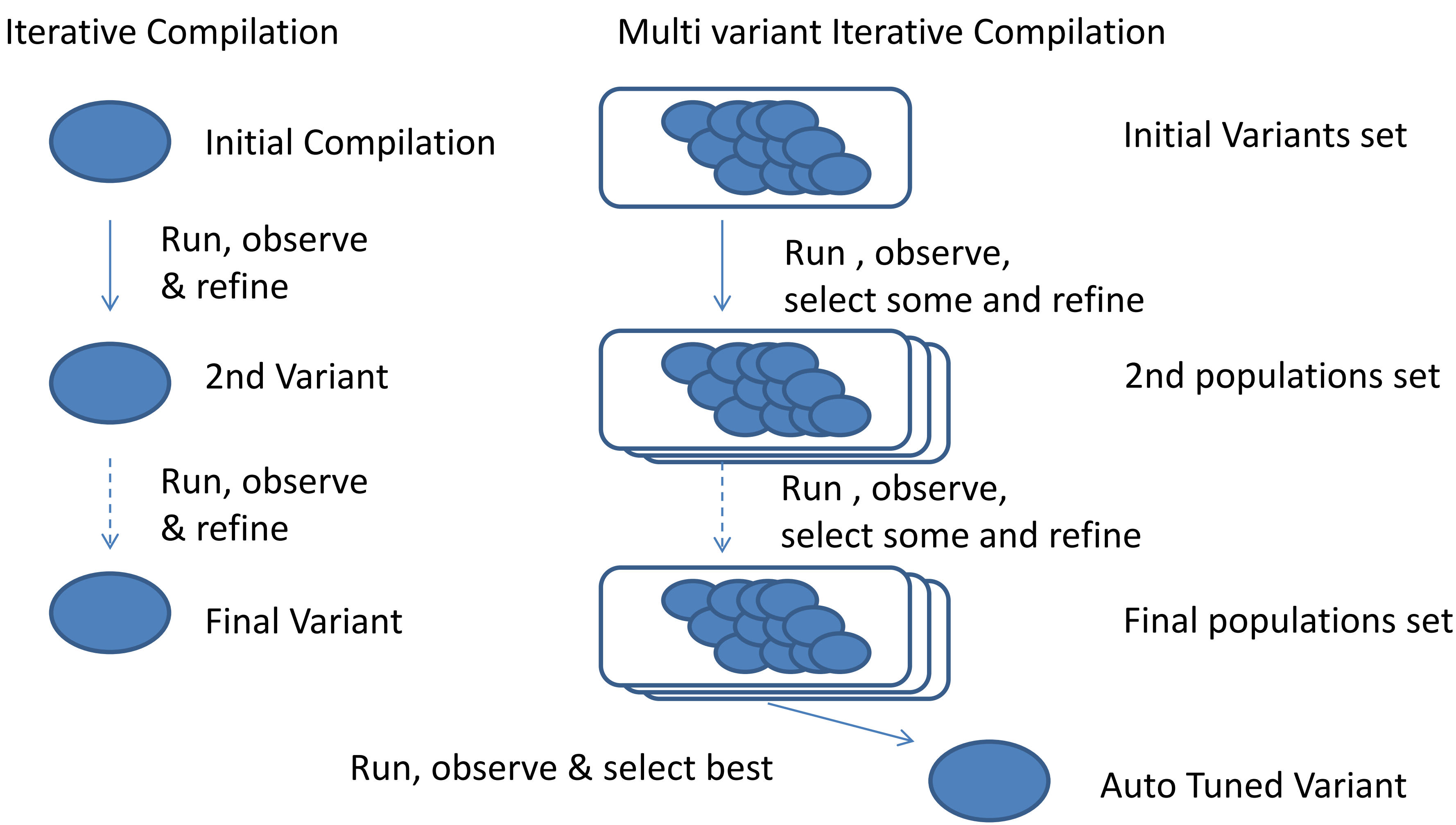
Explore more routes

Exploit knowledge of compilers and architectures

Explore transparently

Inject optimisations without sources modification

Auto Tuning through Iterative Compilation



Cost of Auto Tuning

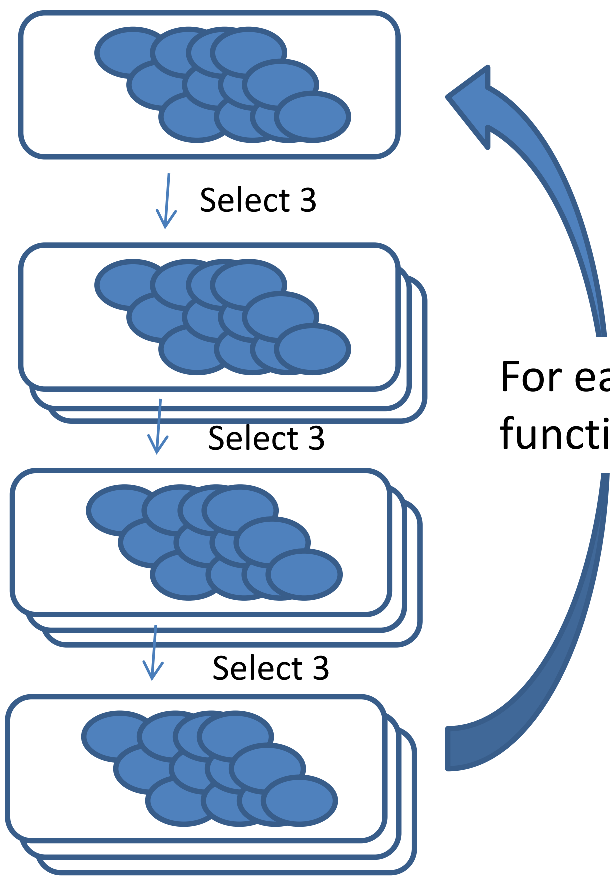
Populations sets (size P)

1st: base
(Ox X fdo X lto)

2nd: inlining
variants

3nd: loop transfo
variants

4th: back-end
variants



For each hot
function (H functions)

Complexity:

0(10.P.H) compilation+run

Typical total time cost (P=100, H=10):

=> 10 000 compilation+run

Parallelizing Auto Tuning

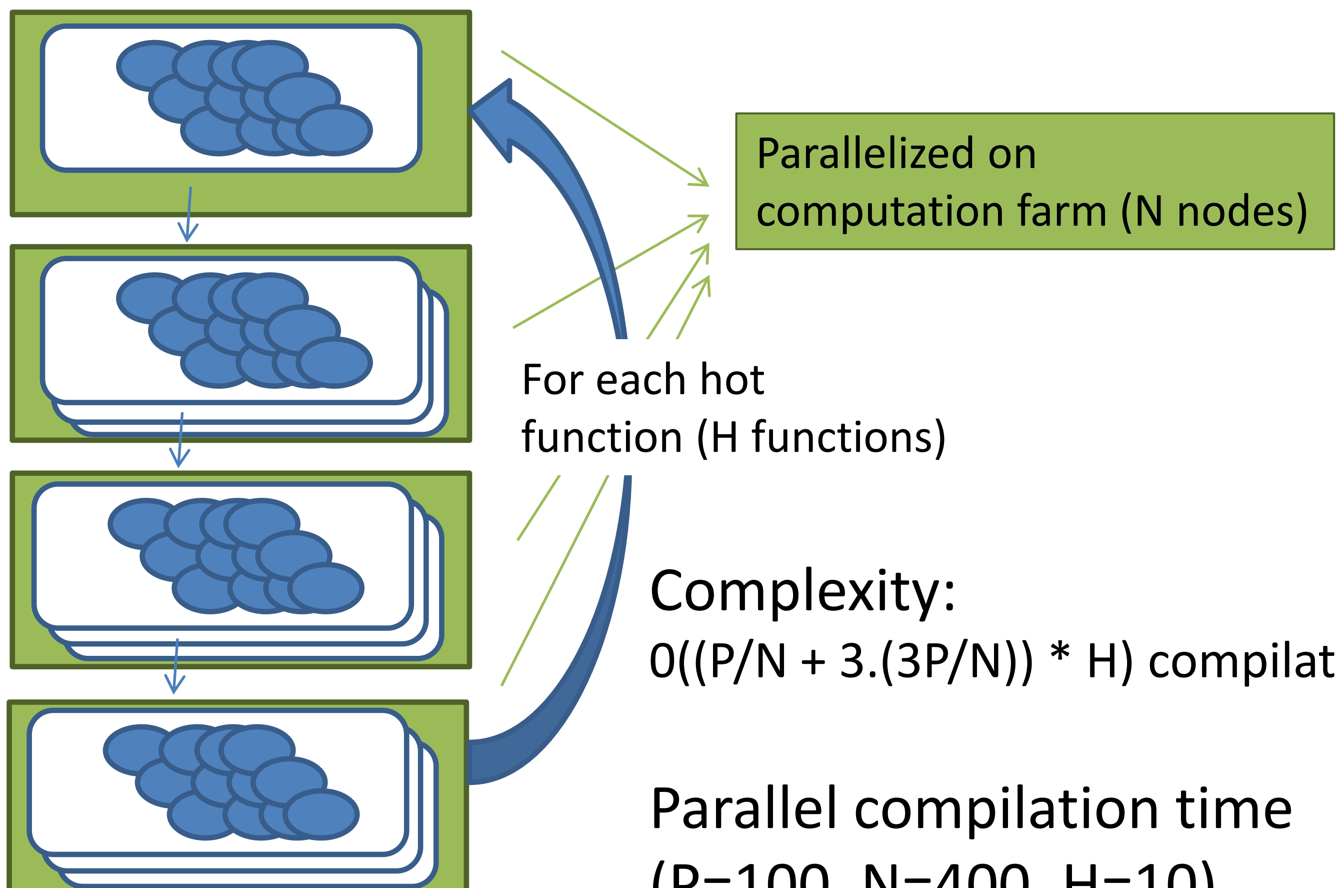
Populations sets (size P)

1st: base
(Ox X fdo X lto)

2nd: inlining
variants

3nd: loop transfo
variants

4th: back-end
variants



Complexity:

0((P/N + 3.(3P/N)) * H) compilation+run

Parallel compilation time

(P=100, N=400, H=10)

=> 40 compilation+run (not 25)

Tools for Auto Tuning

Achieving Seamless Auto Tuning on Computation Farm

Compilation: audit build & produce variants *

Run: on chip or emulation **

Profiling: instrumentation ***

Observation: system perf/size ****

For instance
* with Proot + CARE
** with QEMU for instance
*** with QEMU + dynamic instrumentation
**** with bin/size and bin/perf tools for instance

Demo & Refs

Ref to auto tuning video:
http://youtu.be/l5qpWGIAjUs

Auto tuning frameworks

TACT

ATOS (not yet disclosed)

Genetic exploration Tools

DYNAMIC
PROGRAM INSTRUMENTATION

Motivations

Program performance analysis

cycles count, profiles, call graphs, …

Profile driven compiler optimizations

I-cache placement, edge profiling, data dep. estimation, …

Program debugging

call traces, syscall traces, memory checks, …

Processor architecture analysis

instructions usage, cache behavior, …

Dynamic Binary Translation

Program Loader

Disassembler

Compiler

Target Buffer Manager

Virtual Memory Manager

System calls Emulator

Debugger Bridge

Instrumentation Interface

Processes, Threads and Filesystem Monitor

Example with QEMU

QEMU, a versatile open source tool used for:

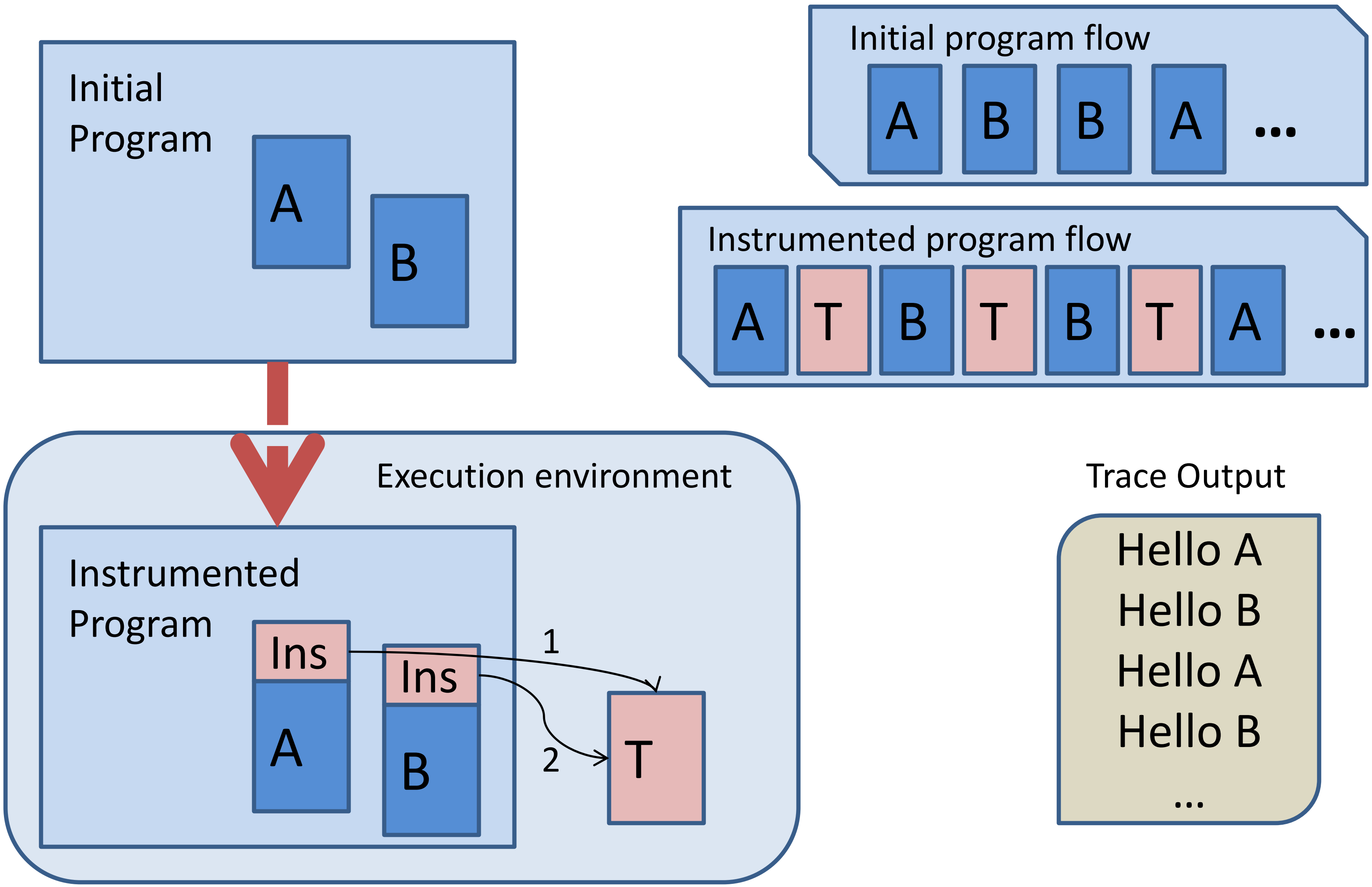
Platform emulation	(Google SDK,…)
Devices emulation	(VirtualBox,…)
Program emulation	(ScratchBox,…)

QEMU is a Dynamic Binary Translator (DBT)

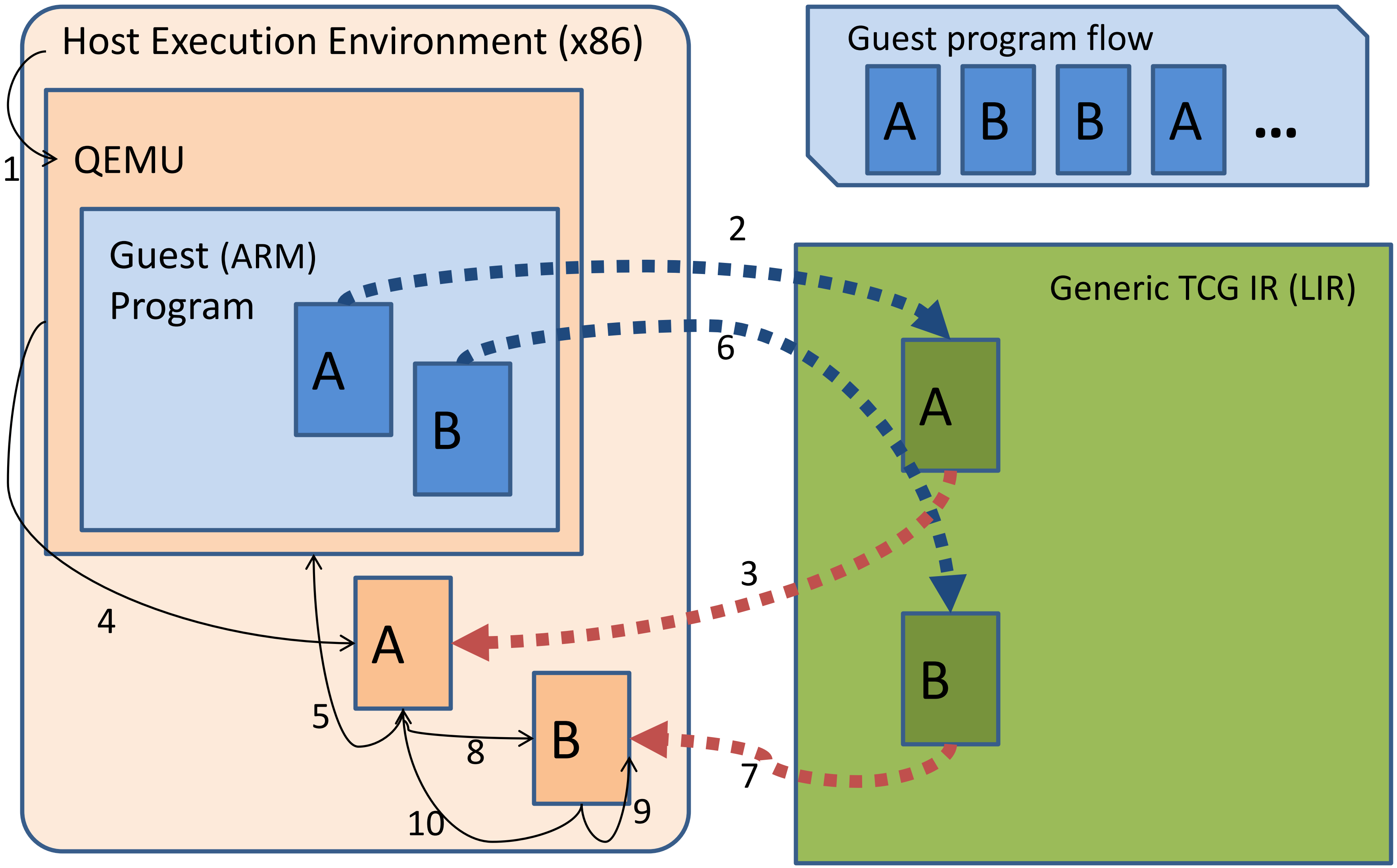
Hence it contains a compiler: TCG

(Tiny Code Generator)

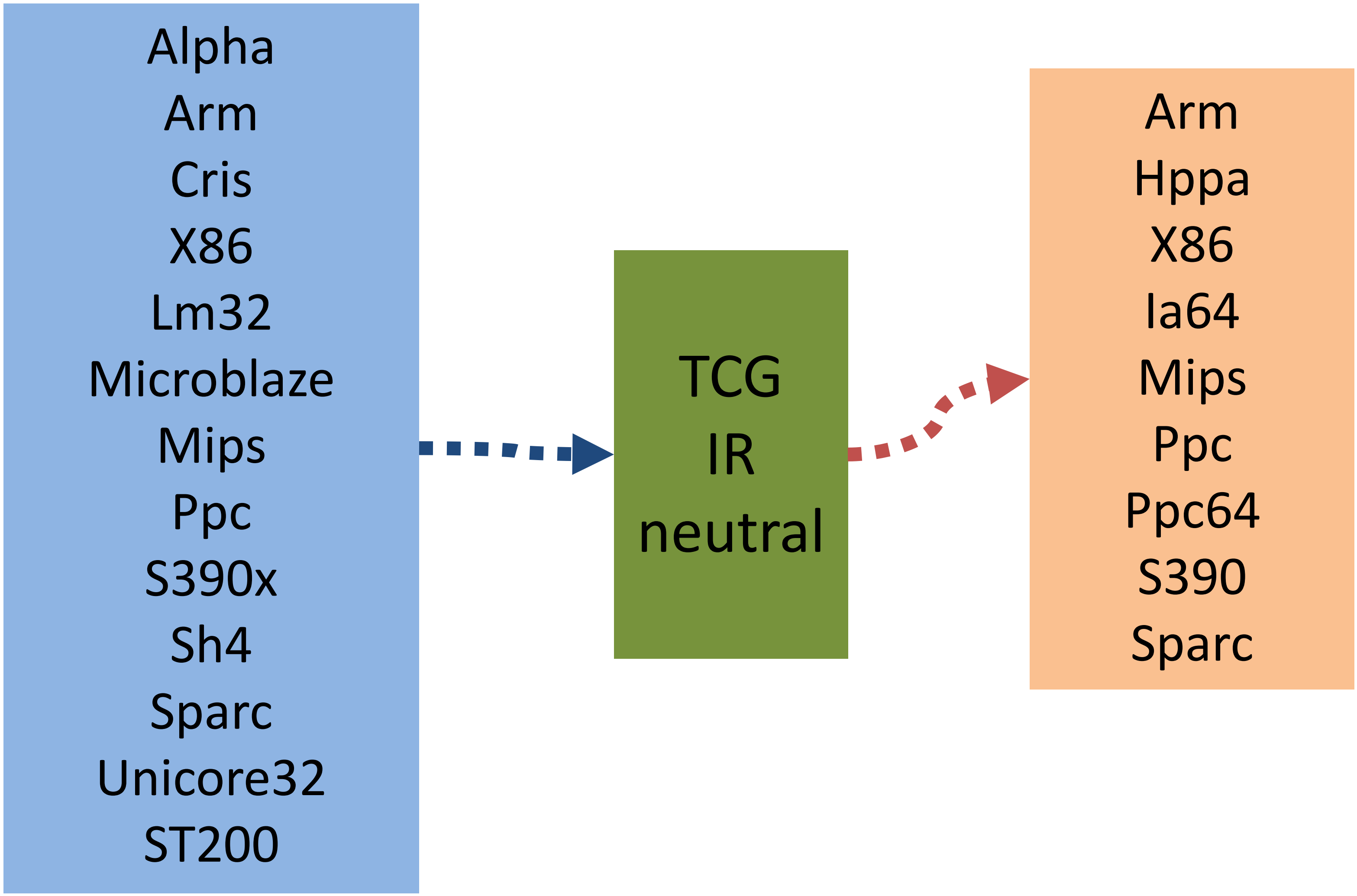
Program Instrumentation



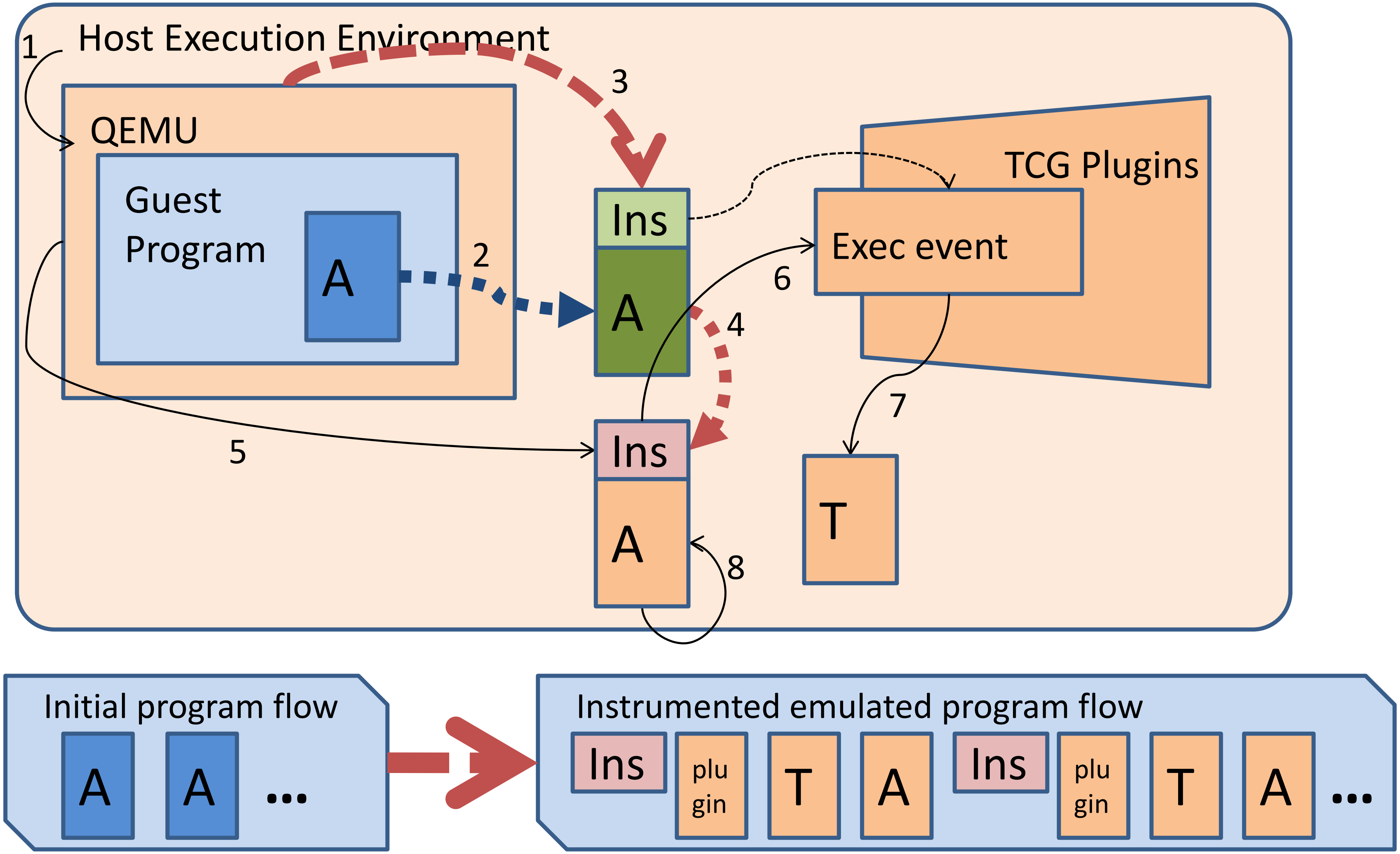
Binary Translation (i.e. QEMU ARM -> x86)



QEMU guests & hosts



Execution Time Instrumentation



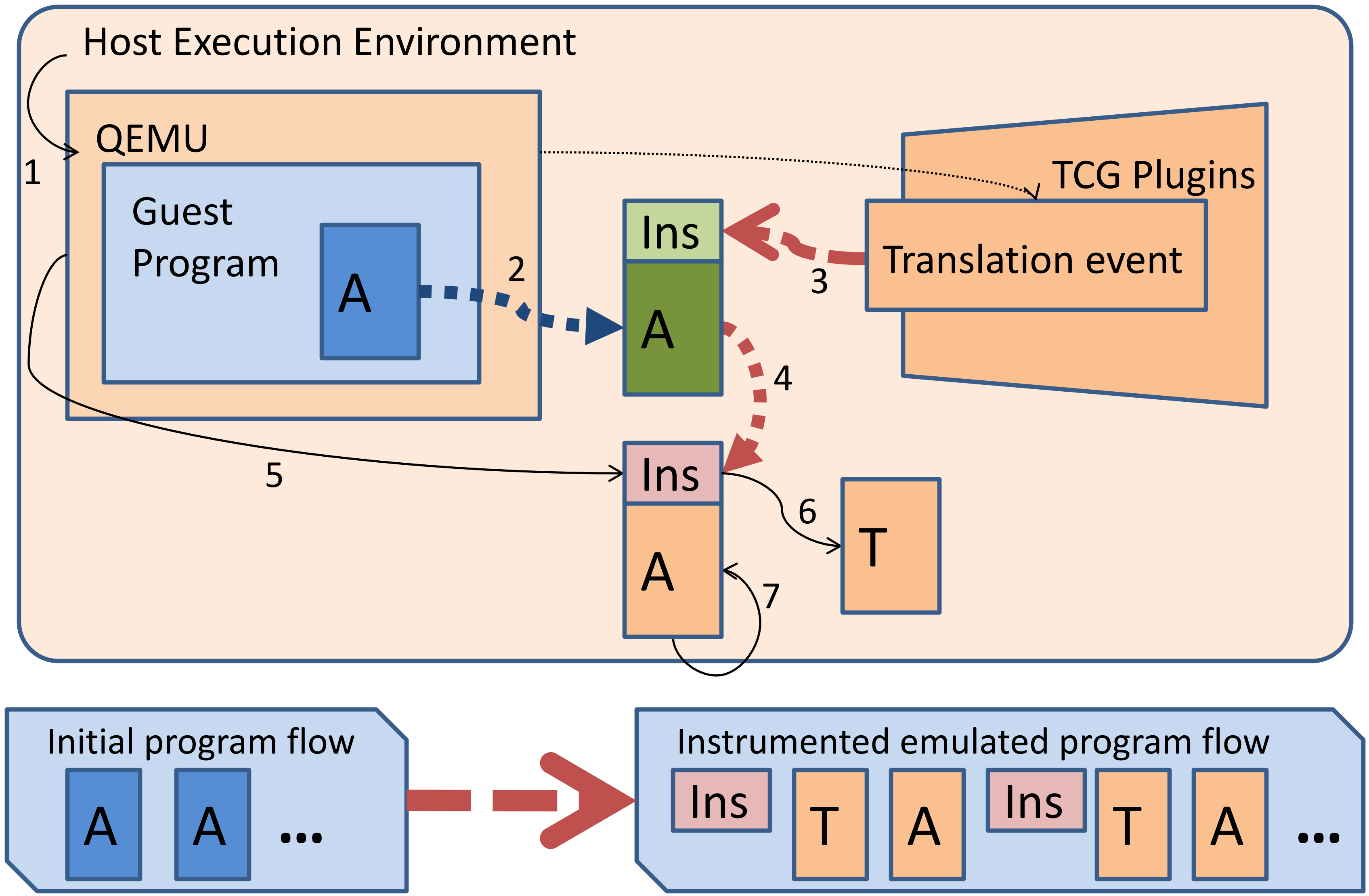
Instruction Count Plugin

```
/* A simple plugin for counting executed guest instructions.
 * Usage : cc shared -o icount.so icount.c
 * qemu-i386 -tcg-plugin ./icount.so the_program
 */
#include <stdint.h>
#include <stdio.h>
#include « qemu_tpi.h »

uint64_t total_icount;

/* Instruction count updated at each block execution. */
void qemu_tpi_block_execution_event(qemu_tpi_t *tpi) {
   total_icount += TPI_tb_icount(tpi);
}
void qemu_tpi_fini(qemu_tpi_t *tpi) {
   printf("Instructions: %"PRIu64"\n", total_icount);
}
```

Translation Time Instrumentation



Inlined Instruction Count Plugin

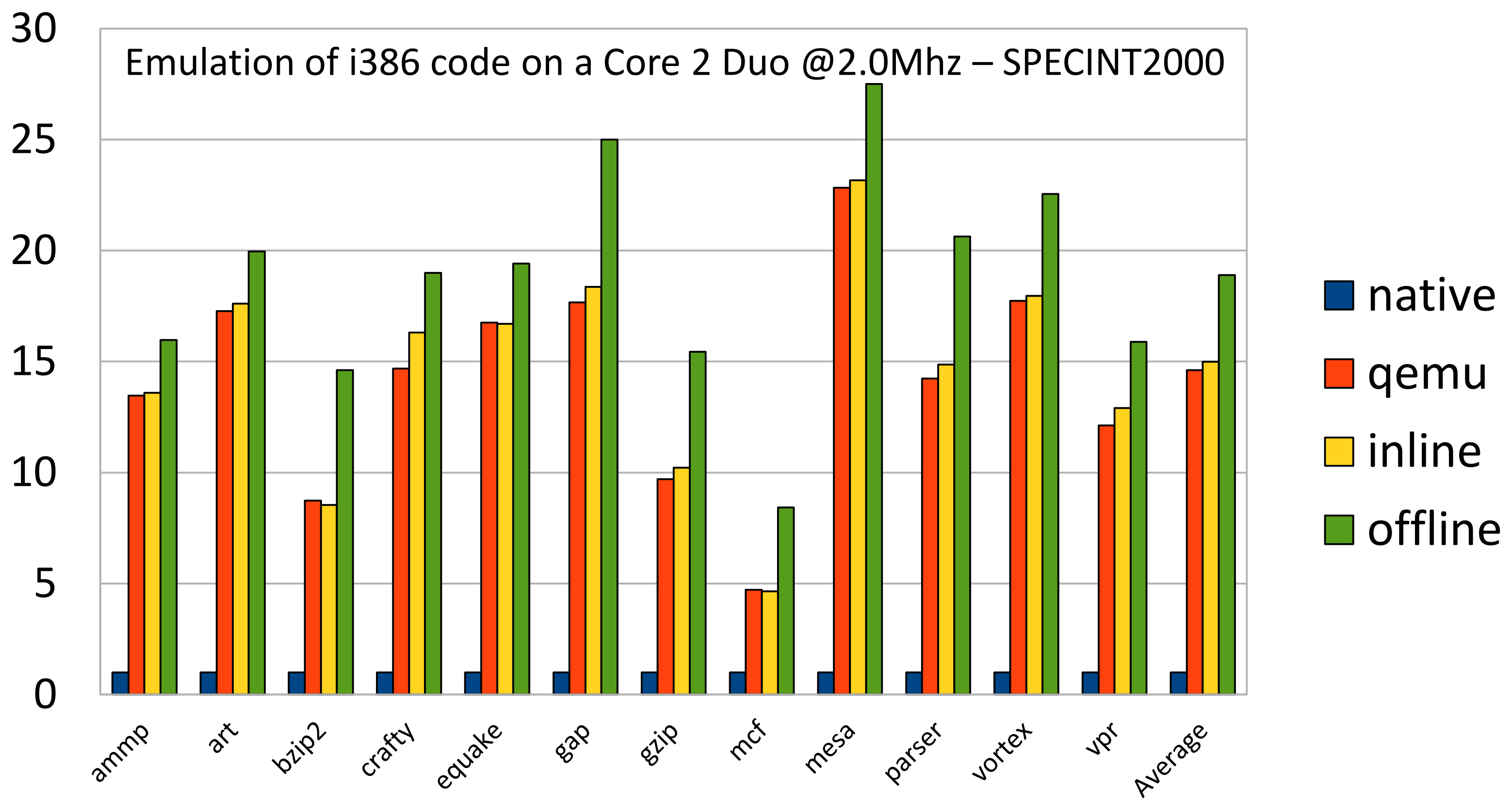
```
/* The translation event based code for the icount.c plugin. */

void qemu_tpi_block_translation_event(qemu_tpi_t *tpi) {
   /* Instruction count update inlined at each block translation.
    * Code is generated into the TCG IR buffer directly. */
   TPIv_ptr ptr = TPI_const_ptr(tpi, &total_icount);
   TPIv_i64 total = TPI_temp_new_i64(tpi);
   TPI_gen_ld_i64(tpi, total, ptr, 0);
   TPIv_i64 icount = TPI_const_i64(tpi, TPI_tb_icount(tpi));
   TPI_gen_add_i64(tpi, total, total, icount);
   TPI_temp_free_i64(tpi, icount);
   TPI_gen_st_i64(tpi, total, ptr, 0);
   TPI_temp_free_i64(tpi, total);
   TPI_temp_free_ptr(tpi, ptr);
}
```

Profile Plugin Example

```
$ qemu-i386 -tcg-plugin profile.so sha1-386.exe
SHA1=15dd99a1991e0b3826fede3deffc1feba42278e6
Instrs          bytes           blocks          symbol
106497664       364229691       1792028         SHA1Transform
571297          1815640         133175          SHA1Update
961             3399            168             SHA1Final
23590           85199           6141            main
18              40              3               __libc_csu_init
16              43              5               .init
2093            12552           2087            .plt
55              146             10              .text
12              28              3               .fini
175102          504418          50840           unknown/libc.so.6
```

Instrumentation Overhead



Emulation cost is x15 (qemu)
Instrumentation overhead is 30% (offline) for a I-count plugin
Overhead reduced to 3% with translation time interface (inline)

Demo & Refs

Ref to live demo

Dynamic Binary Translators and Intrumentation Tools

Valgrind

Pine

DynamoRIO

QEMU

SYSTEM CALLS INTERPOSITION

Motivation

Intercept communications between user programs and the kernel

Modify the behavior (side effects) of the user program without recompiling

Take full control (debugger)

Linux ptrace inteface

Attach

fork(), ptrace(getpid(), …) and exec(), or

ptrace(program_pid, …)

Monitor

wait() and catch TRAP signal

Handle

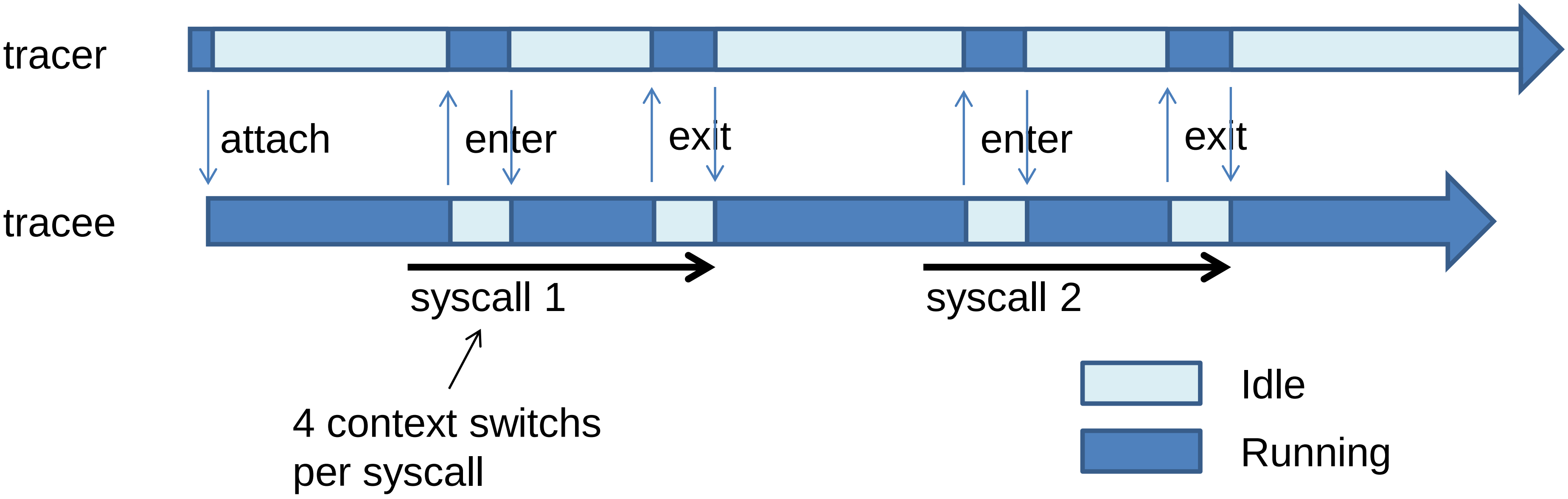
get syscall() info and program state, and

modify syscall()/state and continue

Interposition and Context Switch

All syscall() from the tracee are notified

- just before start
- just before end



Interposition Example

Emulate a change of the ‘/’ (root) location

- equivalent to chroot() but in user-land
- tracee behaves as in ‘/’ but in your /mydir

Trace open(/file_path) (and all calls with paths)

- Tracee: open(“/etc/passwd”)
- Tracer: “/etc/passwd” -> “/mydir/etc/passwd”

Well, it’s not that easy to fake passwords, why?

Also what about the loader, “/mydir/lib/ld.so” ?

Applications

Debugging

gdb, strace

Instrumentation

Valgrind

Cross filesystem execution

QEMU, Proot

Archiving

CARE, CDE

CARE

Comprehensive Archiving

For each path seen in a syscall

if (not_seen(path) && not_existing(path))

copy path to /archive/path

For Reproducible Execution

For each path seen in a syscall

transform path into /archive/path

Demo and Refs

See CARE demo video http://youtu.be/XIPzm66_3w0

CARE: http://reproducible.io

PRoot: http://proot.me