CS 6V81-05: System Security and Malicious Code Analysis
Dynamic Binary Instrumentation

Zhiqiang Lin

Department of Computer Science
University of Texas at Dallas

January 30th, 2012

Outline

Outline

What Is Instrumentation

- A technique that inserts extra code into a program to collect runtime information

What Is Instrumentation

- A technique that inserts extra code into a program to collect runtime information

```
icount++
sub $0xff, %edx
icount++
cmp %esi, %edx
icount++
jle <L1>
icount++
mov $0x1, %edi
icount++
add $0x10, %eax
```

What Is Instrumentation

```
Max = 0;
for (p = head; p; p = p->next)
{

    if (p->value > max)
    {

        max = p->value;
    }
}
```

What Is Instrumentation

```
Max = 0;
for (p = head; p; p = p->next)
{
    printf("In loop\n");
    if (p->value > max)
    {
      printf("True branch\n");
      max = p->value;
    }
}
```

What Is Instrumentation

```
Max = 0;
for (p = head; p; p = p->next)
{
    count[0]++;
    if (p->value > max)
    {
      count[1]++;
      max = p->value;
    }
}
```

What Can Instrumentation Do?

- Profiler for compiler optimization:
  - Basic-block count
  - Value profile
- Micro architectural study:
  - Instrument branches to simulate branch predictors
  - Generate traces
- Bug checking/Vulnerability identification/Exploit generation:
  - Find references to uninitialized, unallocated address
  - Inspect argument at particular function call
  - Inspect function pointers and return addresses
- Software tools that use dynamic binary instrumentation:
  - Valgrind, Pin, QEMU, DynInst, ...

Binary Instrumentation Is Dominant

- Libraries are a big pain for source code level instrumentation
  - Proprietary libraries: communication (MPI, PVM), linear algebra (NGA), database query (SQL libraries).
- Easily handle multi-lingual programs
  - Source code level instrumentation is heavily language dependent.
    - More complicated semantics
- Turning off compiler optimizations can maintain an almost perfect mapping from instructions to source code lines
- Worms and viruses are rarely provided with source code
- ...

Instrumentation approaches: source vs. binary

- Source instrumentation:
  - Instrument source programs
- Binary instrumentation:
  - Instrument executables directly
- Advantages for binary instrumentation
  - Language independent
  - Machine-level view
  - Instrument legacy/proprietary software

Instrumentation approaches: static vs. dynamic

- When to instrument
  - Instrument statically - before runtime
  - Instrument dynamically - during runtime
- Advantages for dynamic instrumentation
  - No need to recompile or relink
  - Discover code at runtime
  - Handle dynamically-generated code
  - Attach to running processes

How is Instrumentation used in Program Analysis?

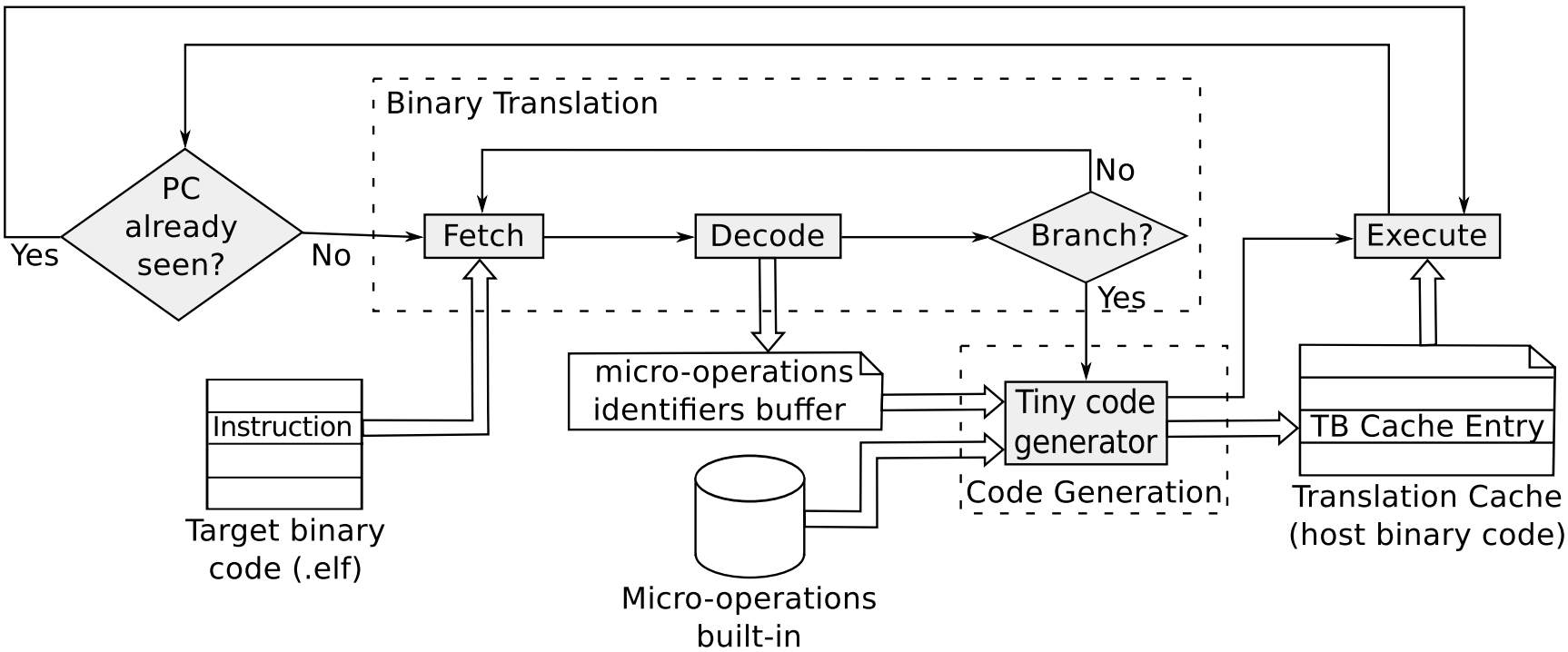
- Code coverage
- Call-graph generation
- Memory-leak detection
- Vulnerability identification
- Instruction profiling
- Data dependence profiling
- Thread analysis
  - Thread profiling
  - Race detection

Outline

QEMU

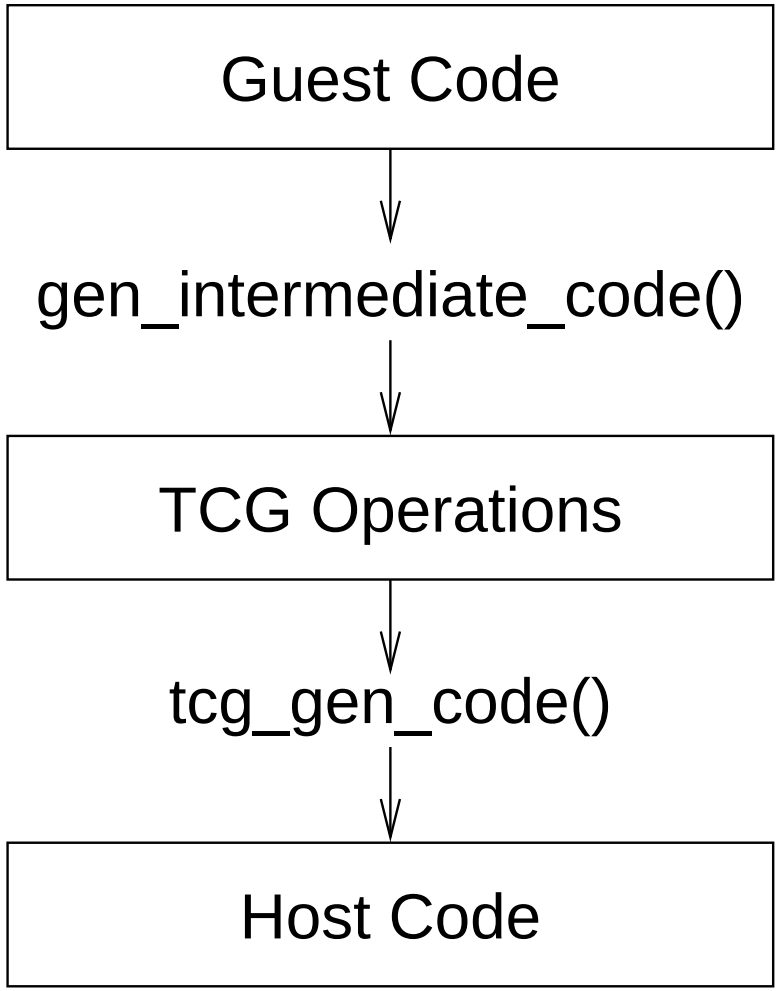
- QEMU is a generic and open source machine emulator and virtualizer.
- As a machine emulator, QEMU can run OSes and programs made for one machine (e.g. an ARM board) on a different machine (e.g. your own PC). By using dynamic translation, it achieves very good performance.
- As a virtualizer, QEMU achieves near native performances by executing the guest code directly on the host CPU. QEMU supports virtualization when executing under the Xen hypervisor or using the KVM kernel module in Linux. When using KVM, QEMU can virtualize x86, server and embedded PowerPC, and S390 guests.

QEMU Internals

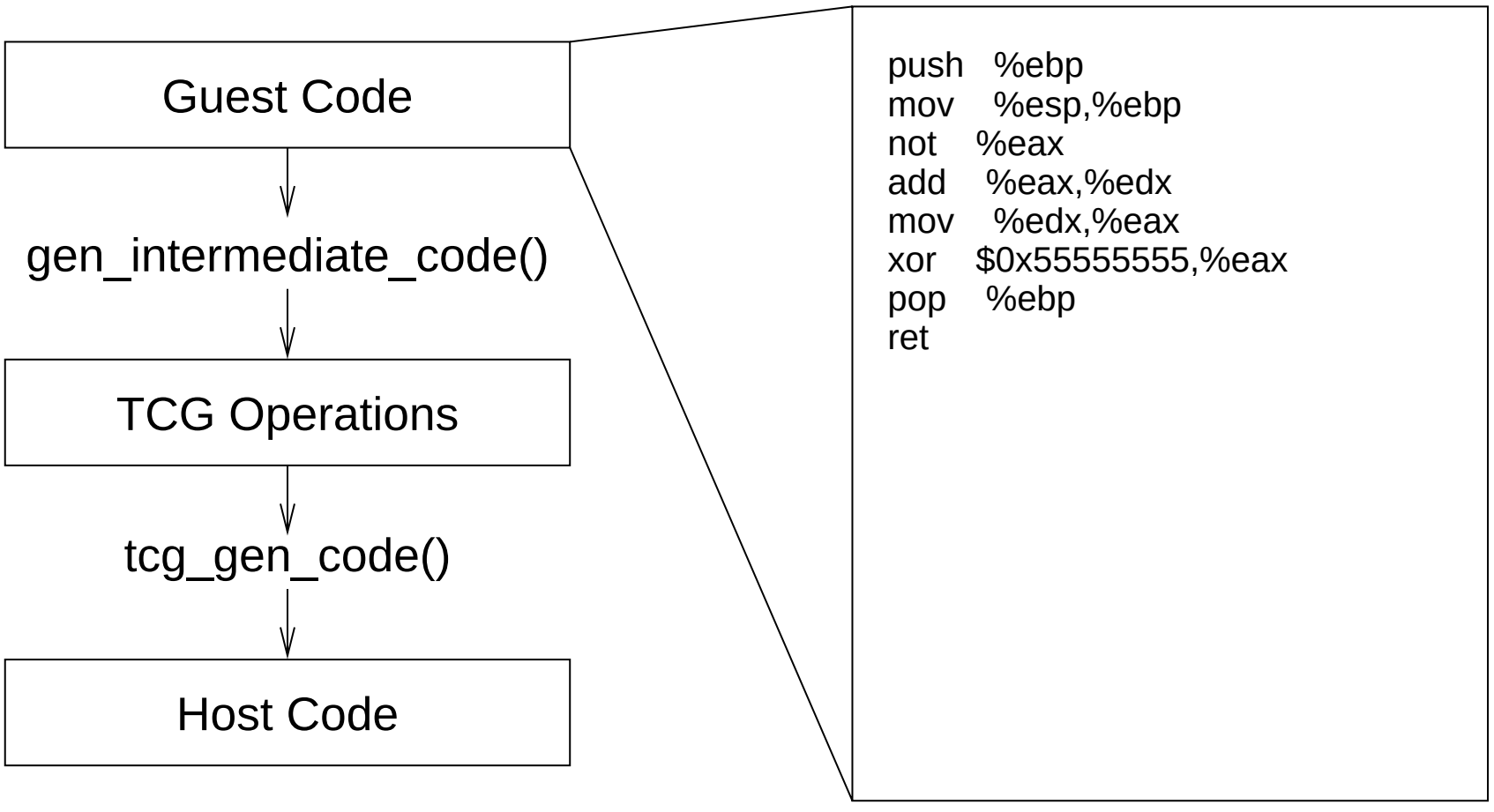


QEMU-Code Translation

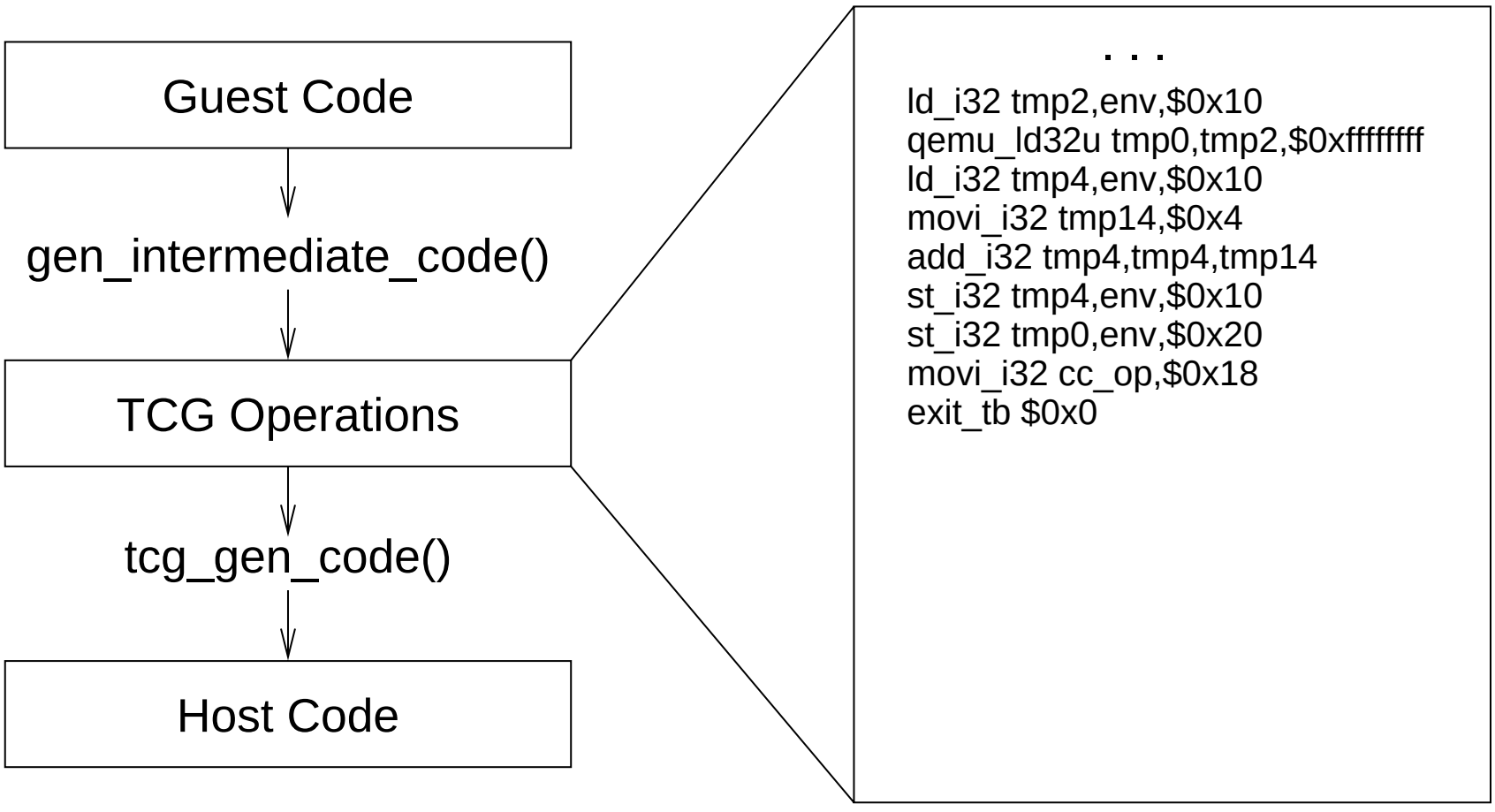
- QEMU uses an intermediate form.
- Frontends are in target-*/, includes alpha, arm, cris, i386, m68k,mips, ppc, sparc, etc.
- Backends are in tcg/*, includes arm/, hppa/, i386/, ia64/, mips/, ppc/, ppc64/, s390/, sparc/, tcg.c, tcg.h, tcg-opc.h, tcg-op.h, tcg-runtime.h



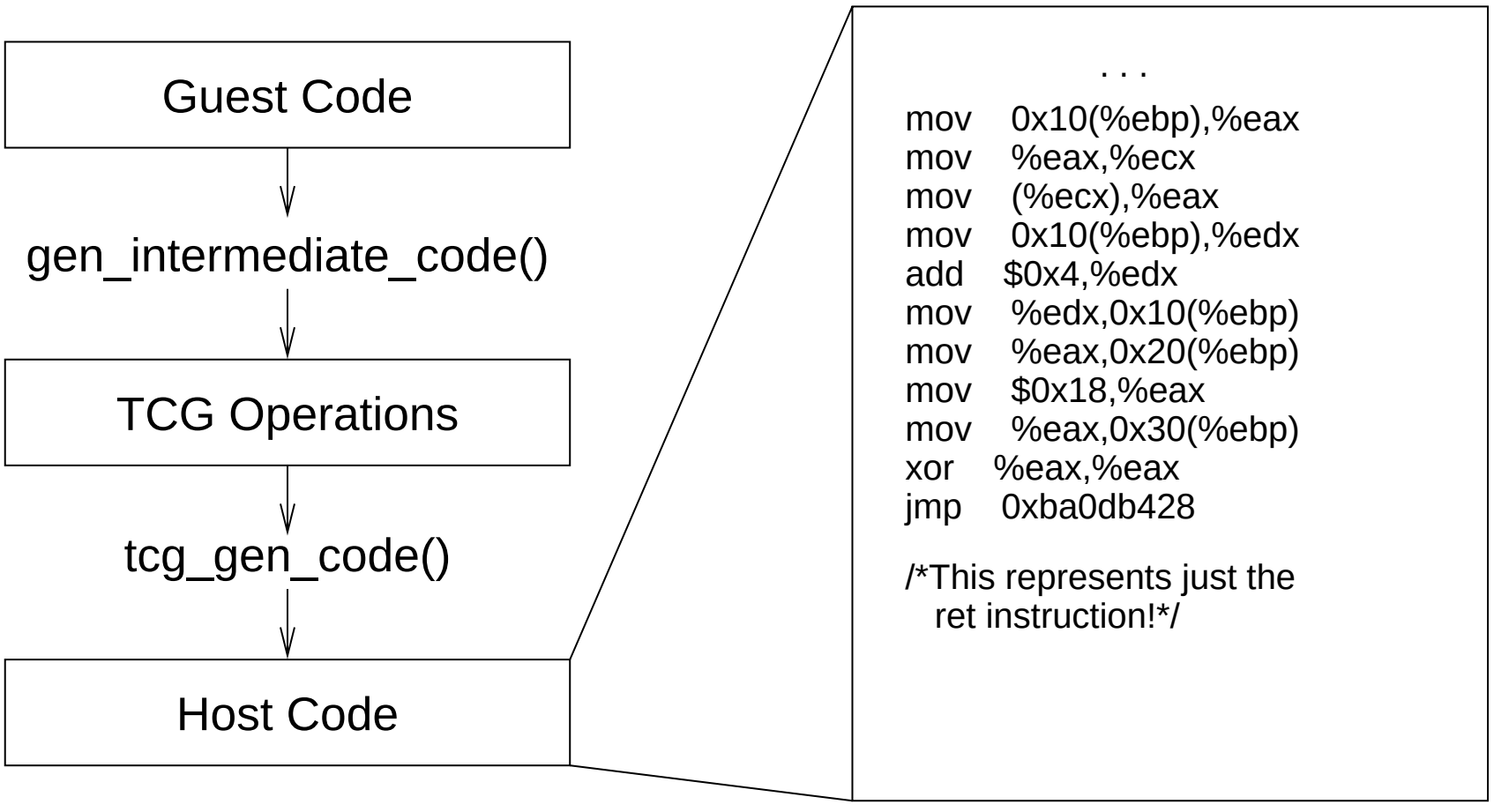
QEMU-Code Translation



QEMU-Code Translation

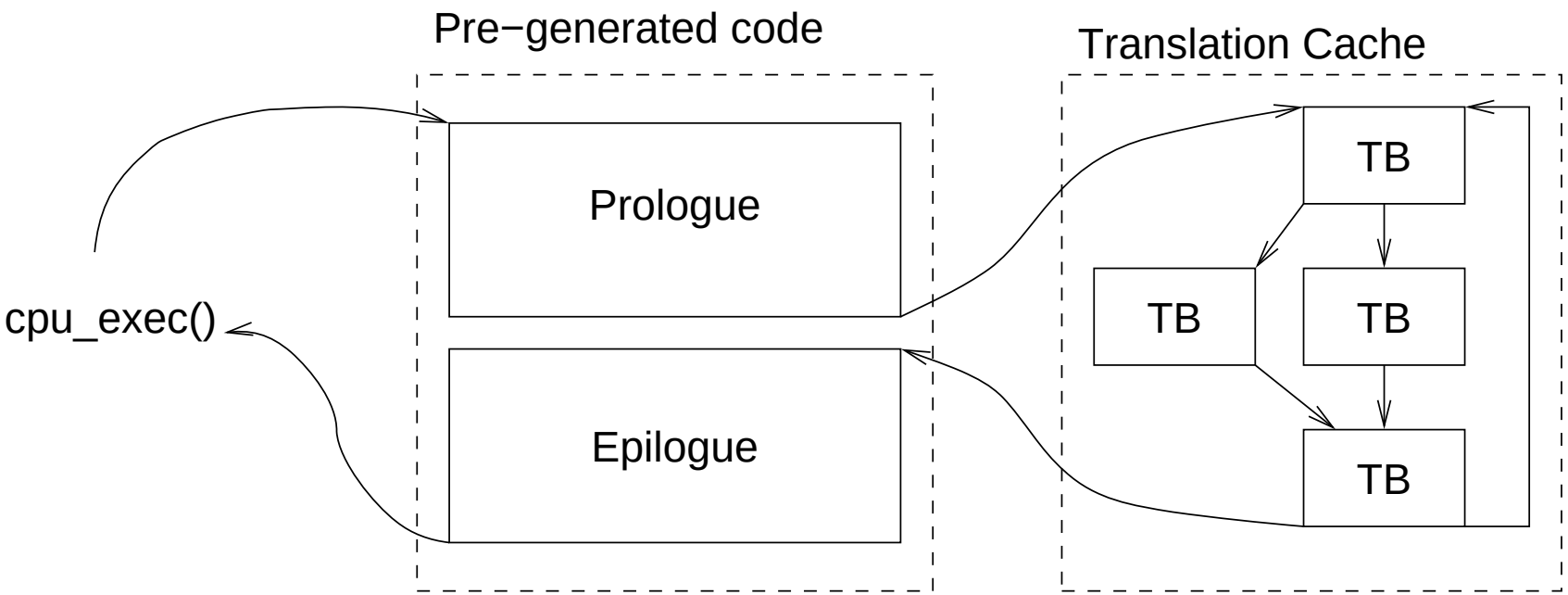


QEMU-Code Translation

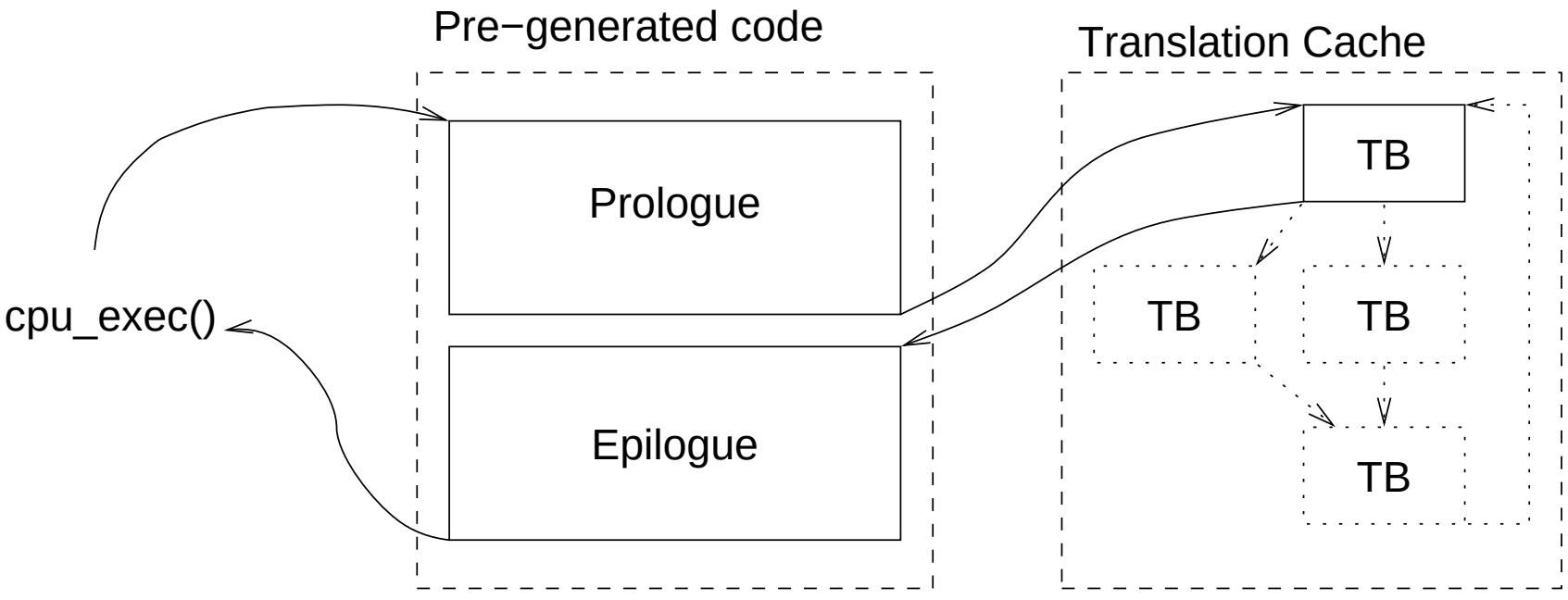


QEMU-Basic Block Chaining

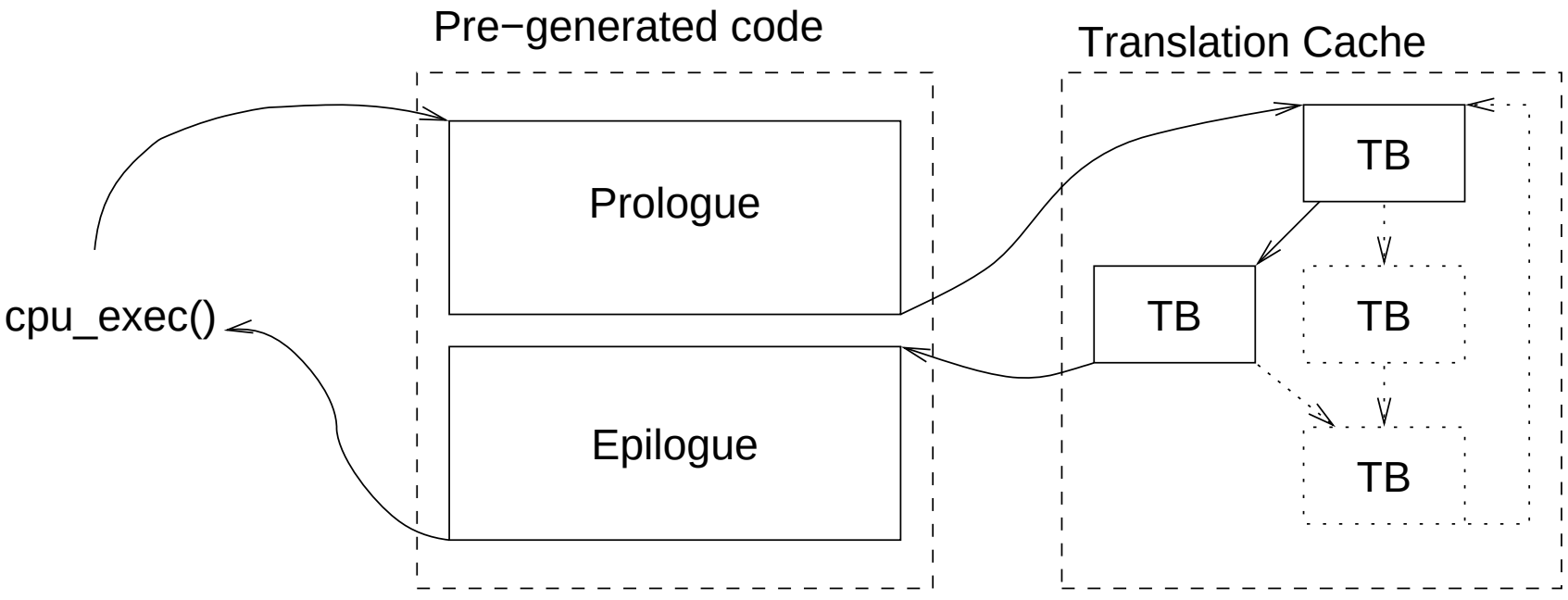
- Returning from code cache is slow.
- Solution: jump directly between basic blocks!
- Make space for a jump, follow by a return to the epilogue.
- Every time a block returns, try to chain it.



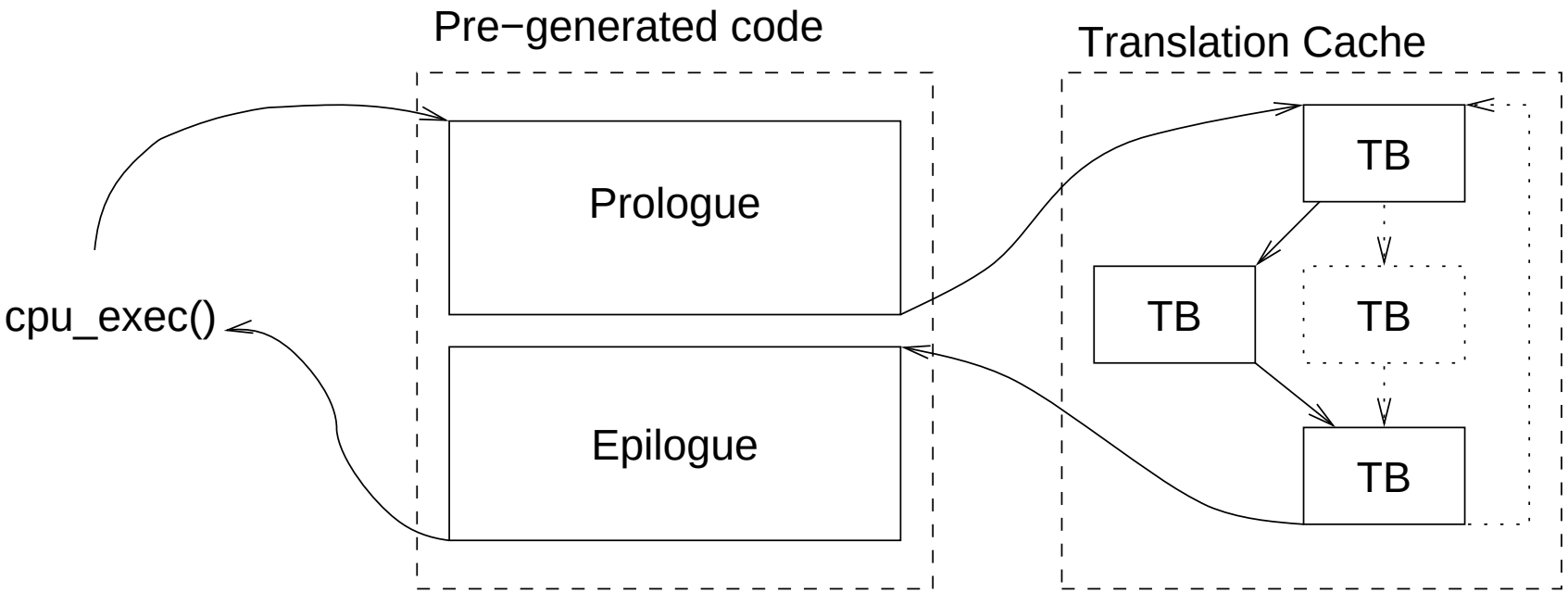
QEMU-Basic Block Chaining: Step 1



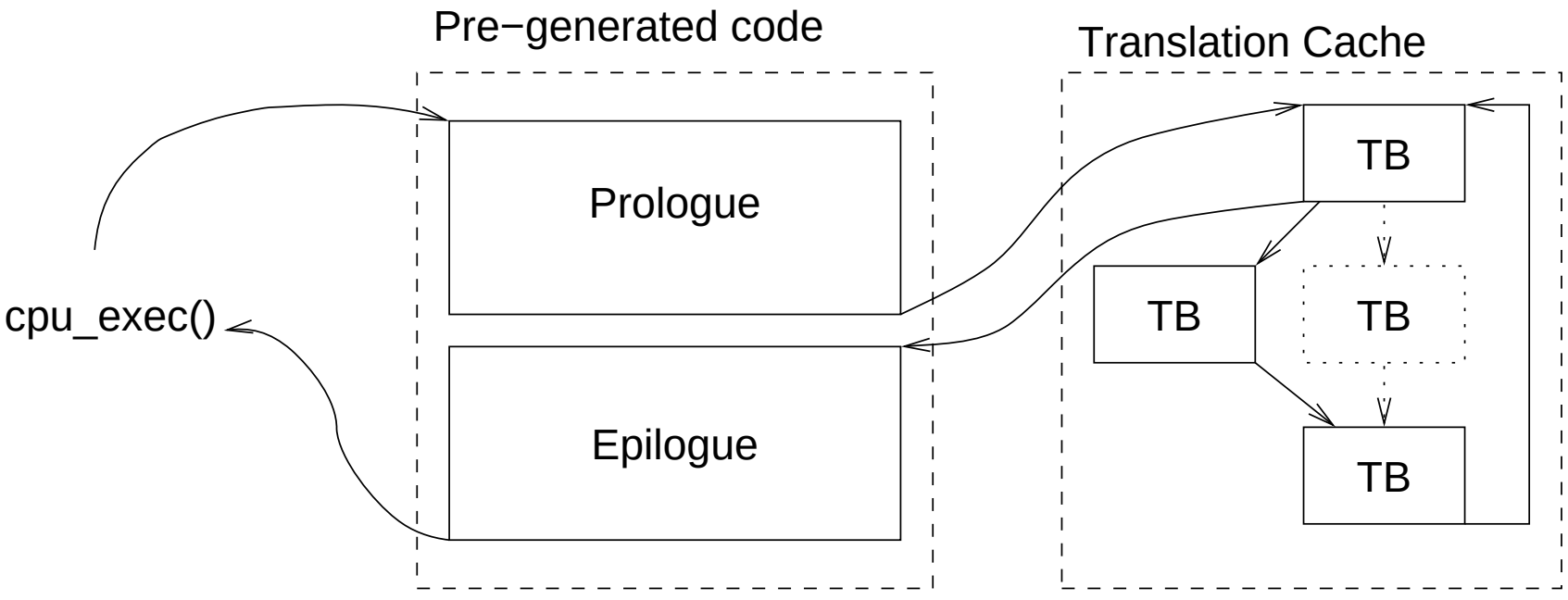
QEMU-Basic Block Chaining: Step 2



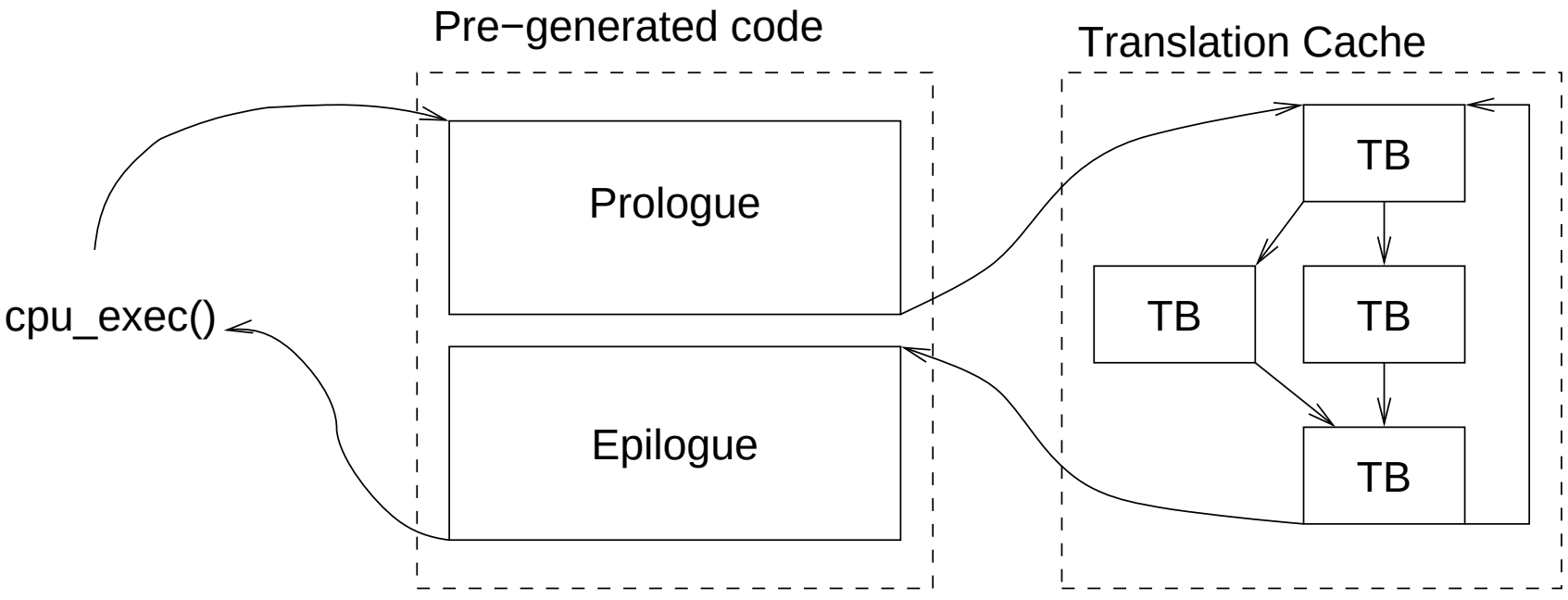
QEMU-Basic Block Chaining: Step 3



QEMU-Basic Block Chaining: Step 4



QEMU-Basic Block Chaining: Step 5



QEMU-Code Base

- TranslationBlock structure in `translate-all.h`
- Translation cache is code gen buffer in `exec.c`
- `cpu-exec()` in cpu-exec.c orchestrates translation and block chaining.
- `target-*/translate.c`: guest ISA specific code.
- `tcg-*/*/`: host ISA specific code.
- `linux-user/*`: Linux usermode specific code.
- `vl.c`: Main loop for system emulation.
- `hw/*`: Hardware, including video, audio, and boards.

Outline

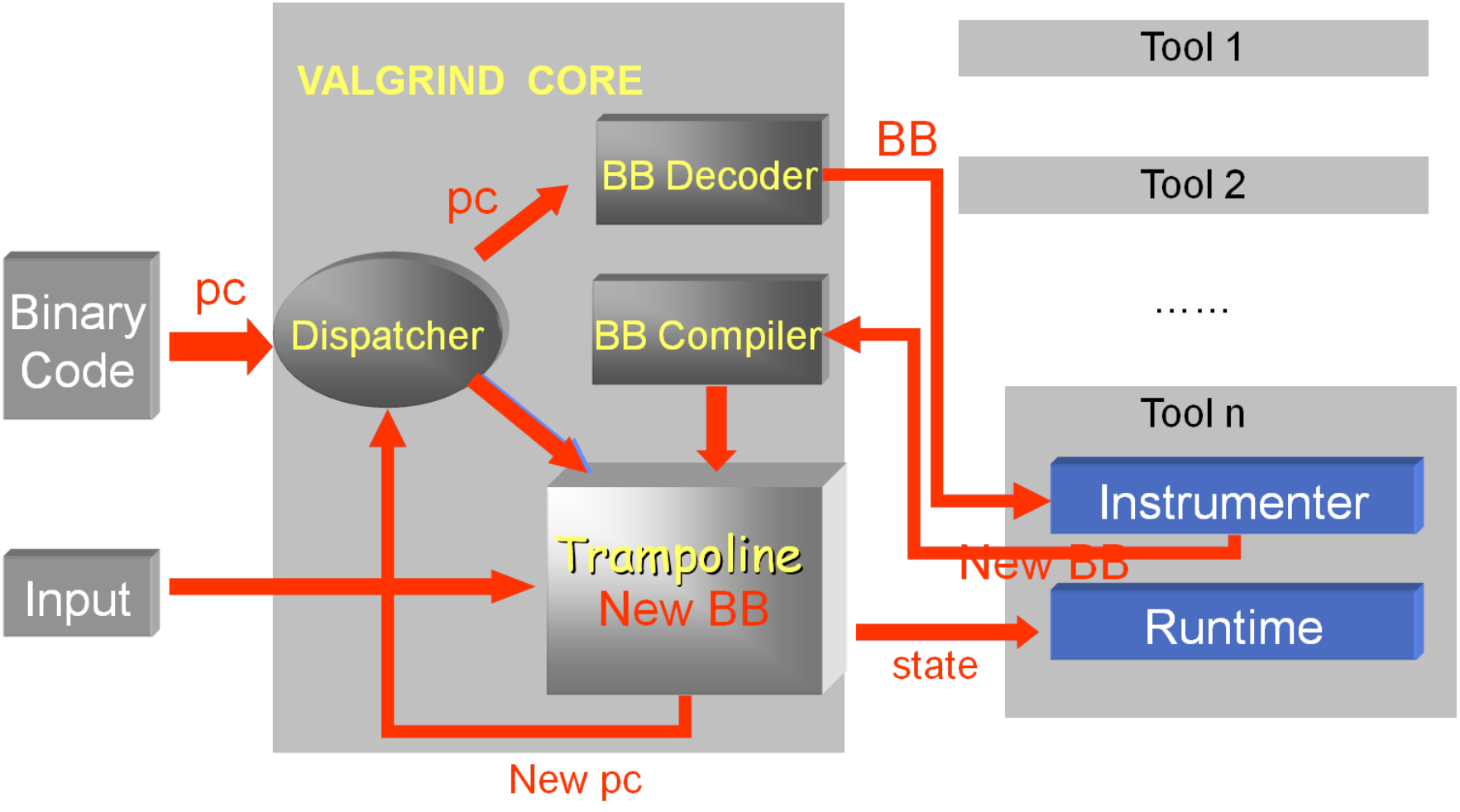
Valgrind

- Valgrind is an instrumentation framework for building dynamic analysis tools.
- Developed by Julian Seward at/around Cambridge University,UK
  - Google-O'Reilly Open Source Award for "Best Toolmaker" 2006
  - A merit (bronze) Open Source Award 2004
  - Open source
    - works on x86, AMD64, PPC code

Dynamic Instrumentation - Valgrind

- Easy to execute, e.g.:
  - valgrind –tool=memcheck ls
- It becomes very popular
  - One of the two most popular dynamic instrumentation tools
    - Pin and Valgrind
  - Very good usability, extendibility, robust
    - 25MLOC
  - Mozilla, MIT, CMU-security, Me, and many other places
- Overhead is the problem
  - 5-10X slowdown without any instrumentation

Valgrind Infrastructure



Dynamic Instrumentation Characteristics

- A trampoline is required.
- Does not require recompiling or relinking
  - Saves time: compile and link times are significant in real systems.
  - Can instrument without linking (relinking is not always possible).
- Dynamically turn on/off, change instrumentation
  - From t1-t2, I want to execute F', t3-t4, I want F"
    - Can be done by invalidating the mapping in the dispatcher.
- Can instrument running programs (such as Web or database servers)
  - Production systems.
- Can instrument self-mutating code.
  - Obfuscation can be easily get around.

Outline

PIN

- Pin is a tool for the instrumentation of programs. It supports Linux* and Windows* executables for IA-32, Intel(R) 64, and IA-64 architectures.
- Pin allows a tool to insert arbitrary code (written in C or C++) in arbitrary places in the executable. The code is added dynamically while the executable is running. This also makes it possible to attach Pin to an already running process.

Advantages of Pin Instrumentation

- Easy-to-use Instrumentation:
  - Uses dynamic instrumentation
    - Does not need source code, recompilation, post-linking
- Programmable Instrumentation:
  - Provides rich APIs to write in C/C++ your own instrumentation tools (called Pintools)
- Multiplatform:
  - Supports x86, x86-64, Itanium
  - Supports Linux, Windows
- Robust:
  - Instruments real-life applications: Database, web browsers,. . .
  - Instruments multithreaded applications
  - Supports signals
- Efficient:
  - Applies compiler optimizations on instrumentation code

Using Pin

- Launch and instrument an application:

```
$pin -t pintool.so  - - application
```

1. instrumentation engine (provided)
2. instrumentation tool (write your own, or use a provided sample)

- Attach to a running process, and instrument application:

```
$pin -t pintool.so  -pid 1234
```

Pin Instrumentation APIs

- Basic APIs are architecture independent:
  - Provide common functionalities like determining:
    - Control-flow changes
    - Memory accesses
- Architecture-specific APIs
  - e.g., Info about opcodes and operands
- Call-based APIs:
  - Instrumentation routines
  - Analysis routines

Instrumentation vs. Analysis

- Concepts borrowed from the ATOM tool:
- Instrumentation routines define where instrumentation is inserted
  - e.g., before instruction
  - Occurs first time an instruction is executed
- Analysis routines define what to do when instrumentation is activated
  - e.g., increment counter
  - Occurs every time an instruction is executed

Pintools 2: Instruction trace

- Need to pass the ip argument to t he printip analysis routine

```
printip(ip)
sub $0xff, %edx
printip(ip)
cmp %esi, %edx
printip(ip)
jle <L1>
printip(ip)
mov $0x1, %edi
printip(ip)
add $0x10, %eax
```

Running itrace tool

```
$ /opt/pin/pin
-t /opt/pin/source/tools/ManualExamples/
obj-intel64/itrace.so
-- /bin/ls

(...)
```

```
$ head itrace.out
0x7f907b188af0
0x7f907b188af3
0x7f907b189120
0x7f907b189121
0x7f907b189124
```

ManualExamples/itrace.cpp

```
#include <stdio.h>
%#include "pin.h"
FILE * trace;
void printip(void *ip) { fprintf(trace, "%p\n", ip); }

void Instruction(INS ins, void *v) {
    INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)printip, IARG_INST_PTR, IARG_END);
}
void Fini(INT32 code, void *v) { fclose(trace); }
int main(int argc, char * argv[]) {
    trace = fopen("itrace.out", "w");
    PIN_Init(argc, argv);
     INS_AddInstrumentFunction(Instruction, 0);

    PIN_AddFiniFunction(Fini, 0);
    PIN_StartProgram();
    return 0;
}
```

Examples of Arguments to Analysis Routine

- IARG_INST_PTR
  - Instruction pointer(program counter) value
- IARG_UINT32 <value>
  - An integer value
- IARG_REG_VALUE <register name>
  - Value of the register specified
- IARG_BRANCH_TARGET_ADDR
  - Target address of the branch instrumented
- IARG_MEMORY_READ_EA
  - Effective address of a memory read
- And many more . . . (refer to the Pin manual for details)

Instrumentation Granularity

- Instrumentation can be done at three different granularities:
  - Instruction
  - Basic block
    - A sequence of instructions terminated at a control-flow changing instruction
    - Single entry, single exit
  - Trace
    - A sequence of basic blocks terminated at an unconditional control-flow changing instruction
    - Single entry, multiple exits

```
sub $0xff, %edx
cmp %esi, %edx
jle <L1>
```

```
mov $0x1, %edi
add $0x10, %eax
jmp <L2>
```

1 Trace, 2 BBs, 6 insts

Recap of Pintool 1: instruction count

- The straightforward implementation works, but counting can be more effiicent

```
icount++
sub $0xff, %edx
icount++
cmp %esi, %edx
icount++
jle <L1>
icount++
mov $0x1, %edi
icount++
add $0x10, %eax
```

ManualExamples/inscount1.cpp

```
#include <stdio.h>
#include "pin.H"
UINT64 icount = 0;
void docount(INT32 c) { icount += c; }
void Trace(TRACE trace, void *v) {
  for (BBL bbl = TRACE_BblHead(trace);
    BBL_Valid(bbl); bbl = BBL_Next(bbl)) {
    BBL_InsertCall(bbl, IPOINT_BEFORE, (AFUNPTR)docount,
          IARG_UINT32, BBL_NumIns(bbl), IARG_END);
  }
}
void Fini(INT32 code, void *v) {
  fprintf(stderr, "Count %lld\n", icount);
}
int main(int argc, char * argv[]) {
   PIN_Init(argc, argv);
   TRACE_AddInstrumentFunction(Trace, 0);
    PIN_AddFiniFunction(Fini, 0);
   PIN_StartProgram();
   return 0;
}
```

Instrumentation vs. Analysis

- Sample tools in the Pin distribution:
  - Cache simulators, branch predictors, address tracer, syscall tracer, edge profiler, stride profiler
- Some tools developed and used inside Intel:
  - Opcodemix (analyze code generated by compilers)
  - PinPoints (find representative regions in programs to simulate)
- Companies are writing their own Pintools
- Universities use Pin in teaching and research

Instrumentation-Driven Simulation

- Fast exploratory studies
  - Instrumentation = native execution
  - Simulation speeds at MIPS
- Characterize complex applications
  - E.g. Oracle, Java, parallel data-mining apps
- Simple to build instrumentation tools
  - Tools can feed simulation models in real time
  - Tools can gather instruction traces for later use

Writing your own Pintool

- It's easier to modify one of the existing tools, and re-use the existing makefile
- Install PIN package in your home directory, and work from there
  - /opt/pin-<version>-<architecture>-<os>.tar.gz

Outline

Summary

References

- http://en.wikipedia.org/wiki/QEMU
- http://wiki.qemu.org/Main_Page
- http://valgrind.org/
- QEMU Internals by Chad D. Kersey
  http://lugatgt.org/content/qemu_internals/downloads/slides.pdf
- http://www.pintool.org/
- Pin 2.10 User Guide
  http://www.pintool.org/docs/45467/Pin/html/
- http://www.dyninst.org/