ISA 563: Fundamentals of Systems Programming

Code Instrumentation with Pin

April 1, 2010

Goals

- Introduction to Pin
- Highlights of the Pin API

- Building a simple tool (strace clone)
- Building Mizer, a memoization tool

- Wrap-up

Motivation

What the heck is program P doing?

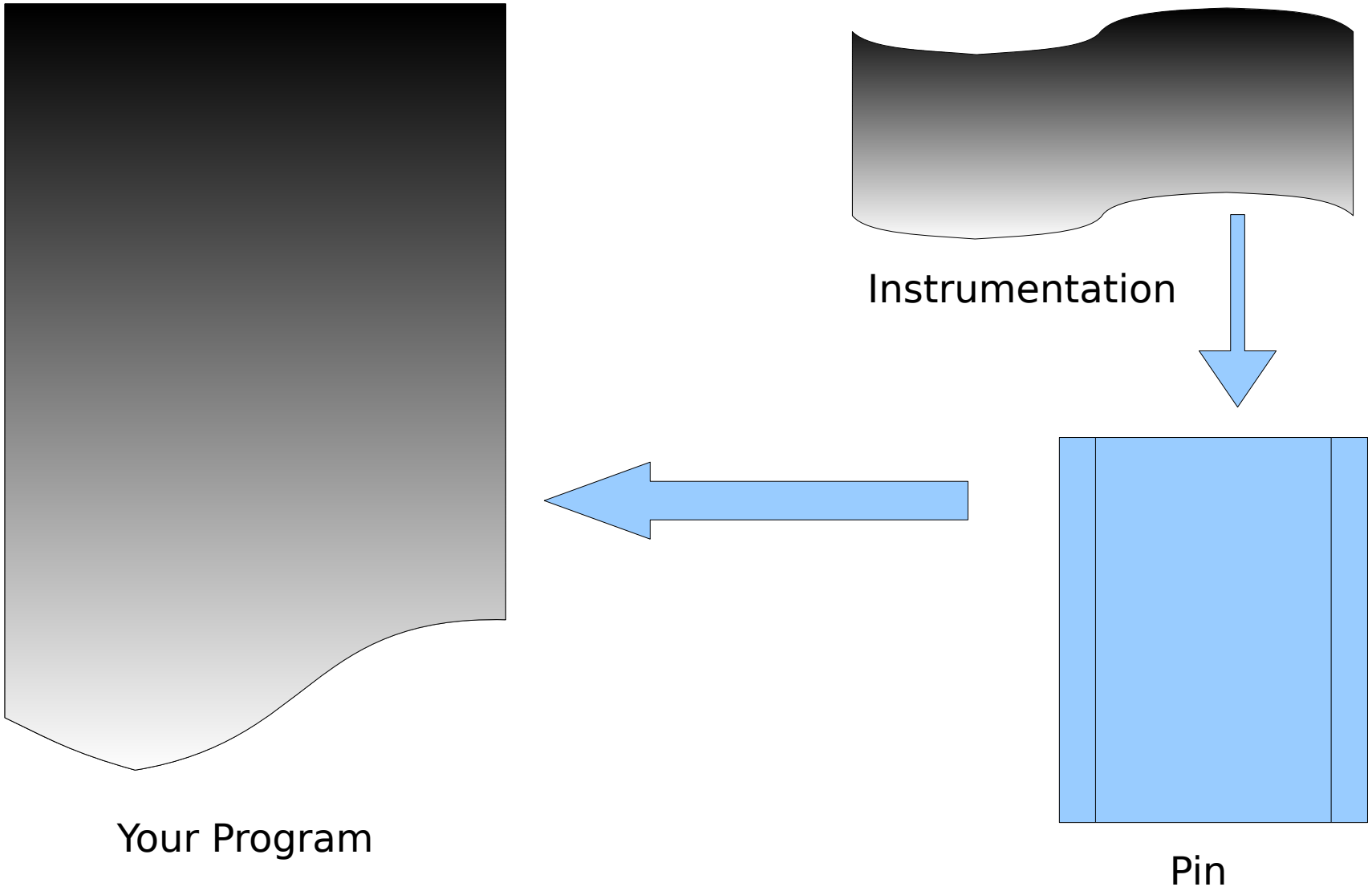
Previous Approaches to Debugging & Instrumentation

- Self-written (printf, System.out, writeln)
  - Unreliable (I/O flushing, corrupted state, etc.)
  - Intrusive (performance, complicates source)
  - Breaks flow (requires recompilation, meta-debugging)
- Debuggers (IDAPro, jdb, gdb, Acme IDE)
  - Programmable interface is limited
- Code profilers (ptrace, gprof, strace, purify)

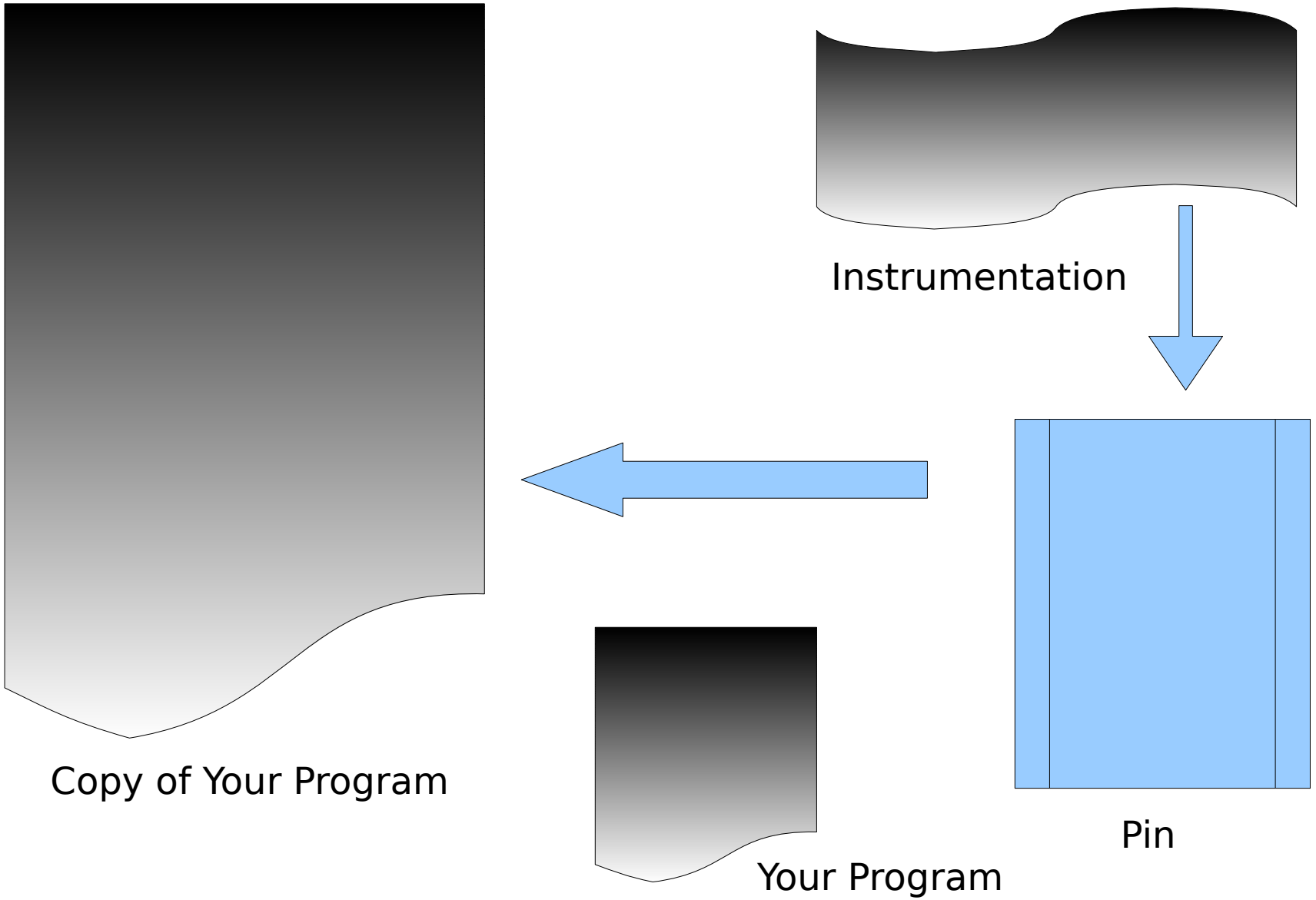
Dynamic Binary Supervision

- Valgrind
  - Great shadow memory subsystem
  - Multi-stage translation, IR
- Dyninst
  - Trampolines & probes
- Pin
  - Dynamic x86 to x86 compilation
  - attach/detach

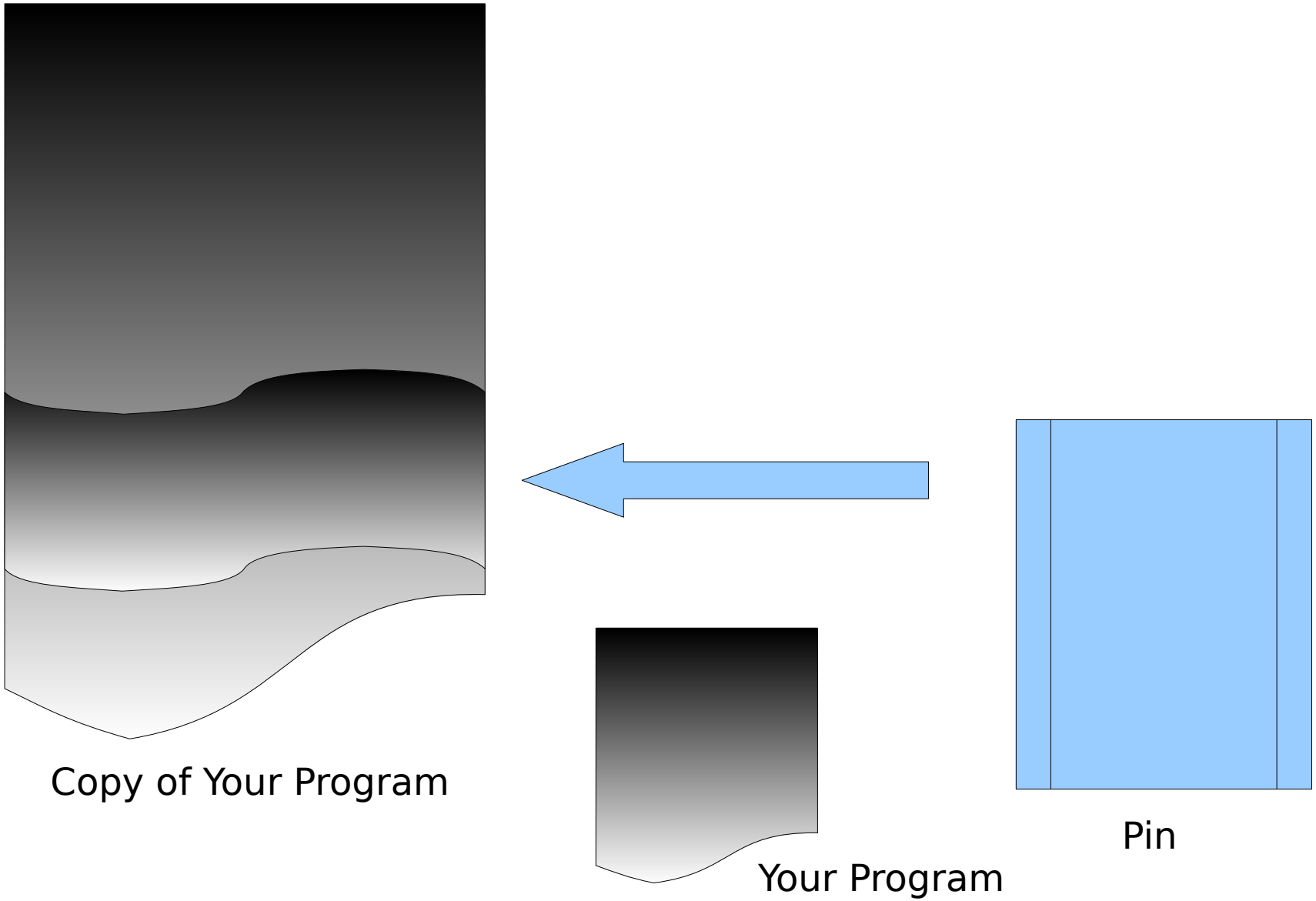
Overview: Injecting Instrumentation



Overview: Injecting Instrumentation



Overview: Injecting Instrumentation



Steps

- Write a Pin tool (C/C++)
  - Write a series of **instrumentation** routines
  - Register **analysis** routines at "interesting" events / points
  - Write analysis routines
- Compile against Pin API
- Run (*unmodified!)* program
  - [you@node]$ pin -t mytool -- /bin/ls -Shl
- Observe output / debug

Pin Benefits

- Written with Intel's blessing, support, etc.
- Can attach after program startup
- Notices all interesting events, instructions, etc.
  - Static and dynamic instrumentation
- Fairly complete API to extract all this information
- Dynamically recompiles: only a 1 time instrumentation overhead

And now the good stuff...

Writing an strace(1) clone

Pin Tool Design

- Examine every *encountered* instruction
- Does it signify a system call?
  - If yes, print out a record of it
  - If not, ignore it
  - Deal with returning from a system call
- Clean up

```
int main(int argc, char *argv[])
{
  PIN_InitSymbols();
  if(PIN_Init(argc, argv))
     return -1;
  INS_AddInstrumentFunction(InjectInstr, 0);
  PIN_AddFiniFunction(Fini, 0);
  PIN_StartProgram();
  return 0;
}
```

```
int main(int argc, char *argv[])
{
  PIN_InitSymbols();
  if(PIN_Init(argc, argv))
     return -1;
  INS_AddInstrumentFunction(InjectInstr, 0);
  PIN_AddFiniFunction(Fini, 0);
  PIN_StartProgram();
  return 0;
}
```

```
VOID
Fini(INT32 code,
     VOID *v)
{
   fprintf(stderr,
           “\nstrace done with code %d\n”,
           code);
}
```

InjectInstr (part 1)

```
VOID
InjectInstr(INS ins,
            VOID *v)
{
  if(INS_IsSyscall(ins))
  {
    ... //next slide
  }
}
```

InjectInstr (part 2)

```
if(INS_IsSyscall(ins))
{
  INS_InsertCall(ins, IPOINT_BEFORE,
                 AFUNPTR(SyscallEnter),
                 IARG_INST_PTR,
                 IARG_SYSCALL_NUMBER,
                 IARG_SYSARG_VALUE, 0,
                 ...
                 IARG_END);
  INS_InsertCall(ins, IPOINT_AFTER,
                 AFUNPTR(SyscallExit),
                 IARG_SYSRET_VALUE,
                 IARG_END);
```

InjectInstr (part 2)

```
if(INS_IsSyscall(ins))
{
  INS_InsertCall(ins, IPOINT_BEFORE,
                 AFUNPTR(SyscallEnter),
                 IARG_INST_PTR,
                 IARG_SYSCALL_NUMBER,
                 IARG_SYSARG_VALUE, 0,
                 ...
                 IARG_END);
  INS_InsertCall(ins, IPOINT_AFTER,
                 AFUNPTR(SyscallExit),
                 IARG_SYSRET_VALUE,
                 IARG_END);
```

```
VOID
SyscallEnter(VOID *invoked_at_addr,
             INT32 syscall_number,
             VOID *arg0,
             ...
             VOID *arg5)
{
   fprintf(stdout,
           “%d(%p, %p, %p, %p, %p, %p)”,
           syscall_number,
           arg0, arg1, arg2,
           arg3, arg4, arg5);
}
```

```
VOID
SyscallExit(VOID *ret)
{
   fprintf(stdout,
           " = %p\n",
           ret);
   fflush(stdout);
}
```

```
[me@host stclone]$ ../Bin/pin -t strace -- /bin/true

uname(0xbff787dc, 0xbff7896c, ...) = (nil)
brk((nil), 0x9a5fb4, ...) = 0x98d3000
access(0x9a18f8, 0x4, ...) = 0xffffffff
open(0x9a2570, (nil), ...) = 0x3
fstat64(0x3, 0xbff77ef8, ...) = (nil)
...
brk(0x98f4000, 0xacfff4, ...) = 0x98f4000
open(0xab0320, 0x8000, 0x1, ...) = 0x3
fstat64(0x3, 0xad11a0, ...) = (nil)
mmap2((nil), 0x200000, 0x1, ...) = 0xb635b000
close(0x3, 0xacfff4, 0xad1184, ...) = (nil)
exit_group((nil), (nil), (nil), ...)

strace-clone done with code 0
```

Objects to Instrument

- Instructions (INS)
- Basic Blocks (BBL)
- Traces (TRACE)
- Routine/Function (RTN)
- Image (IMG)
- Section (SEC)

Other Fun Stuff

- CPU checkpoint/restart
- Thread support
- Superpin: automatic parallelization
- KNOBS: cmd line switches made easy
- Alarms, Filters, Controllers...