SystemTap

Full System Observability for GNU/Linux

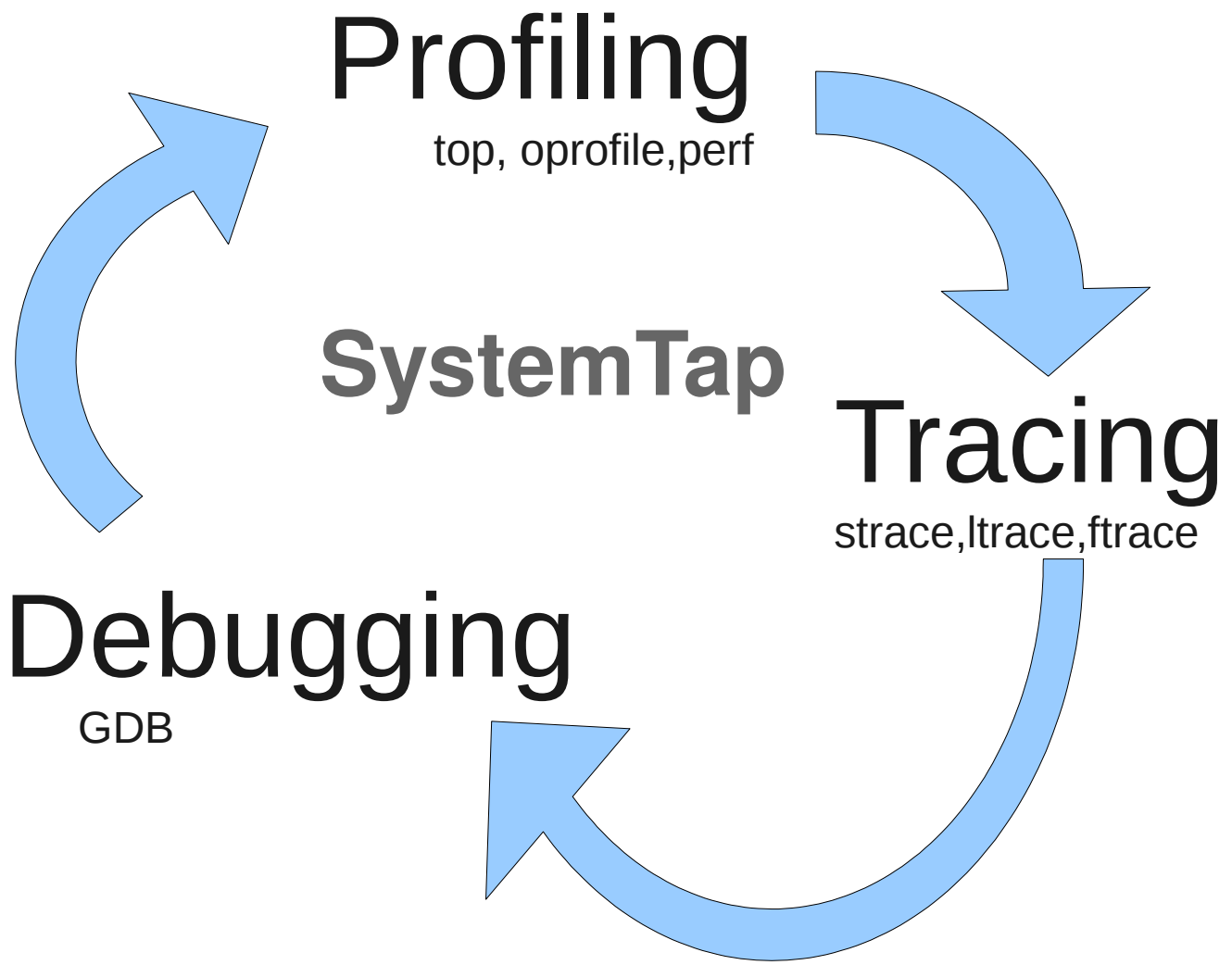
Mark Wielaard

Fosdem 2010

This Talk

- About SystemTap
  - The circle of observability
  - tracing, profiling, debugging
  - Some examples
- Not just SystemTap
  - Working with others
  - Working with you!
  - Observing Scripting & Language runtimes
  - More examples

The circle of observability



Tracing

- Nice to have
  - Provides info while running
  - Did I pass GO?
  - Quick overview of code flow
- Limitations
  - Often specialized tools (strace, ltrace, ftrace)
  - Limited filtering
  - Overwhelming amount of information
    - Trying to “grep it out” can alter sytem under observation

Profiling

- Nice to have
  - Monitors while running (sampling)
  - Can (often) see systemwide
- Limitations
  - Often one kind of event (time)
  - After the fact analysis

Debugging

- Nice to have
  - Full context (variables, parameters, memory, registers, backtrace)
  - Conditional breakpoints
- Limitations
  - Stops the program under inspection
  - One program at a time (not system wide)

One tool to rule them all

- Unobtrusive, non-stop
- System wide
- Monitoring multiple event types

  Synchronous and Asynchronous
- Scriptable (in-place) filtering and statistics collection
- Safe (enforces limits, stops dumb things)

How does it look

- probe <event> { handler }
  - Where event is kernel.function, process.statement, timer.ms, begin, end, (tapset) aliases
- handler can have:
  - filtering/conditionals (if ... next)
  - control structures (foreach, while)
- Variables are primitive (number, string), associate arrays or statistical aggregate
- Helper functions (log, printf, gettimeofday, pid)

How to run

- stap -e '<script>' [-c <target program>]

  stap script.stp [-c <target program>]

  stap -l '<event*>'

Examples!

- No tricks!
  - Fedora 12
- OK, some tricks...
  - Installed all debuginfo beforehand
  - Added myself to stapdev group
  - Python package & Java package from rawhide

EXAMPLES

Tracing all functions in a program:

```
$ stap -e 'probe process("/bin/ls").function("*")
{ log(pp()) }' -c /bin/ls

process("/bin/ls").function("main@/usr/src/debug/
coreutils-7.6/src/ls.c:1225")
process("/bin/ls").function("set_program_name@/us
r/src/debug/coreutils-7.6/lib/progname.c:35")
process("/bin/ls").function("initialize_exit_fail
ure@/usr/src/debug/coreutils-
7.6/src/system.h:118")
process("/bin/ls").function("decode_switches@/usr
/src/debug/coreutils-7.6/src/ls.c:1497")
[...]
```

Showing all parameters of probed functions

```
$ stap -e 'probe process("/bin/ls").function("*")
{ log(probefunc() . ": " . $$parms) }' -c /bin/ls

main: argc=0x1 argv=0xbfd4c504
set_program_name: argv0=0xbfd4d568
initialize_exit_failure: status=0x2
[...]
```

Using thread_indent() to show control flow and timings (1)

```
$ cat trace-ls.stp

probe process("/bin/ls").function("*").call
{
  log(thread_indent(1) . "=> " . probefunc())
}

probe process("/bin/ls").function("*").return
{
  log(thread_indent(-1) . "<= " . probefunc())
}
```

Using thread_indent() to show control flow and timings (2)

```
$ stap trace-ls.stp -c /bin/ls

     0 ls(20718):=> main
    28 ls(20718): => set_program_name
    53 ls(20718): <= set_program_name
   221 ls(20718): => human_options
   243 ls(20718): <= human_options
   266 ls(20718): => clone_quoting_options
   285 ls(20718):  => xmemdup
   303 ls(20718):   => xmalloc
```

System wide syscall probing

```
$ stap -e 'probe syscall.open
  {log(execname() . ": " . filename)}'

gnome-settings-: /proc/mounts
hald-addon-stor: /dev/sr0
devkit-disks-da: /dev/sr0
gpm: /dev/tty0
sendmail: /proc/loadavg
gnome-settings-: /proc/mounts
hald-addon-stor: /dev/sr0
devkit-disks-da: /dev/sr0
[...]
```

Top applications doing vfs read/write
See examples/io/iotop.stp

```
$ stap iotop.stp

       Process          KB Read       KB Written
        stapio             3328                 0
  simpress.bin             1560                 0
          Xorg              226                 0
gnome-terminal              199                 0
      metacity               48                 0
gnome-settings-              36                 0
hald-addon-inpu              14                 0
gnome-power-man              12                 0
```

Keeping statistics for vfs.read (unread global aggregate variables are printed at and)

```
$ cat iotop2.stp
global reads
probe vfs.read {reads[execname()] <<< $count}

$ stap iotop2.stp
reads["simpress.bin"] @count=0x28e @min=0x4f
@max=0x2f060 @sum=0x4a3296 @avg=0x1d0b
reads["firefox"] @count=0x10c @min=0x1
@max=0x1000 @sum=0xab061 @avg=0xa35
reads["Xorg"] @count=0x2d @min=0x100 @max=0x1018
@sum=0x20c58 @avg=0xba6
reads["dbus-daemon"] @count=0x20 @min=0x800
@max=0x800 @sum=0x10000 @avg=0x800
```

Not just SystemTap

- Observability is not just fancy scripting
- The events and context need to be there
- Low Level & High Level

Low Level (events & contexts)

- Nicely covered
- Timers, processes, kprobes, uprobes (perf-counters, data watches)
- GCC debuginfo keeps improving
- Trick is to get more tools using them

High level (events & contexts)

- Through TapSets (aliases)
  - e.g. vfs, nfs, tcp tapsets (mapping to one or more alternative low-level function/statement probes)
- Through markers (in source)
  - Developer guided: “this might be interesting” points
- We need your help!

Kernel

- Two solutions!
  - Markers – stap -l 'kernel.mark(“*”)'
  - Tracepoints – stap -l 'kernel.trace(“*”)'
- Plus lots of tapsets (nfs,tcp,scheduler,vfs,etc.)

A long lost friend

- dtrace – similar in spirit to SystemTap
- Not suitable for GNU/Linux

  (long sad licensing story)
- But they had a great idea:

  User Static Dynamic probes

#include <sys/sdt.h>

- Deprecate/Discourage STAP_PROBE
- Made DTRACE_PROBE macros source compatible
- Provide dtrace (python) script that emulates build setup
- Often all you need is:

  ./configure –enable-dtrace

A nice addition to -- verbose

- Add static trace points (also) where you add verbose output or logging
- Let your users surprise you!

Stap on Stap

- stap -v prints for each pass timing/mem usage
- But you can (also) get such things with

  proc_mem_string() and task_time_string()
- And not just strings, actual values!
- So we added probes to indicate each pass start/end
- Now we can run a stap script systemwide while the testsuite is running and collect full statistics!

Make your scripting/ language runtime observable

- Example python
- Example java

Python (1)

```
$ cat python_trace.stp
probe process("/usr/lib/libpython2.6.so").mark
("function__entry") {
  printf("%s -> %s %s:%d\n",
         thread_indent(1), user_string($arg2),
         user_string($arg1), $arg3) }
probe process("/usr/lib/libpython2.6.so").mark
("function__return") {
  printf("%s <- %s %s:%d\n",
         thread_indent(-1), user_string($arg2),
         user_string($arg1), $arg3) }
```

Python (2)

```
$ stap python-trace -c python

  31 python(85):-> <module> UserDict.py:1
  60 python(85): -> UserDict UserDict.py:3
 103 python(85): <- UserDict UserDict.py:72
 137 python(85): -> IterableUserDict
UserDict.py:78
 160 python(85): <- IterableUserDict
UserDict.py:79
 303 python(85): -> <module> _abcoll.py:9
1644 python(85):  -> <module> abc.py:4
1680 python(85):   -> abstractproperty abc.py:28
```

Java – full backtrace on jni function call

```
$ stap -e 'probe hotspot.jni.GetArrayLength {print_jstack_full()}'
-c 'java Hello' | c++filt
jni_GetArrayLength
<Interpreter@0x14c14aa>
java/io/FileOutputStream.writeBytes([BII)V<Interpreter@0x14b9e61>
java/io/FileOutputStream.write([BII)V<Interpreter@0x14b9e61>
java/io/BufferedOutputStream.flushBuffer()V<Interpreter@0x14b9e61>
java/io/BufferedOutputStream.flush()V<Interpreter@0x14b9e61>
java/io/PrintStream.write([BII)V<Interpreter@0x14b9e61>
sun/nio/cs/StreamEncoder.writeBytes()V<Interpreter@0x14b9e61>
sun/nio/cs/StreamEncoder.implFlushBuffer()V<Interpreter@0x14b9e61>
sun/nio/cs/StreamEncoder.flushBuffer()V<Interpreter@0x14b9e61>
java/io/OutputStreamWriter.flushBuffer()V<Interpreter@0x14b9e61>
java/io/PrintStream.newLine()V<Interpreter@0x14b9e61>
java/io/PrintStream.println(Ljava/lang/String;)V<Interpreter@0x14b
9e61>
Hello.main([Ljava/lang/String;)V<StubRoutines_(1)@0x14b734c>
.L176
os::os_exception_wrapper(void (*)(JavaValue*, methodHandle*,
JavaCallArguments*, Thread*), JavaValue*, methodHandle*,
JavaCallArguments*, Thread*)
[...]
```

SystemTap – Q/A

- http://sourceware.org/systemtap/
  - Tutorial, Beginners Guide, Language reference, Examples, man pages, and more...
- http://sourceware.org/systemtap/wiki/
  - AddingUserSpaceProbingToApps