SystemTap Sans Kernel

A pure userspace backend

Josh Stone <jistone@redhat.com>
Software Engineer, Red Hat
April 4, 2012

TL;DR

+ Dyninst

SystemTap Sans Kernel

- SystemTap status quo
- Introducing Dyninst
- Putting them together
- Status and plans

SystemTap is:

- A Linux command-line tool, driven by scripts
- Instrument and run code at many events
  - Arbitrary kernel locations (kprobes)
  - Arbitrary userspace locations (uprobes)
  - Predefined locations (kernel tracepoints, SDT)
  - Hardware performance counters
  - Timers, syscalls, process lifetimes, etc.

- http://sourceware.org/systemtap

Example 1 – top syscalls

```
global syscount

probe syscall.* {
  syscount[name] += 1
}

probe end {
  foreach (count = name in syscount- limit 10)
    printf("%6d  %s\n", count, name)
}
```


Example 2 – scheduler tracing

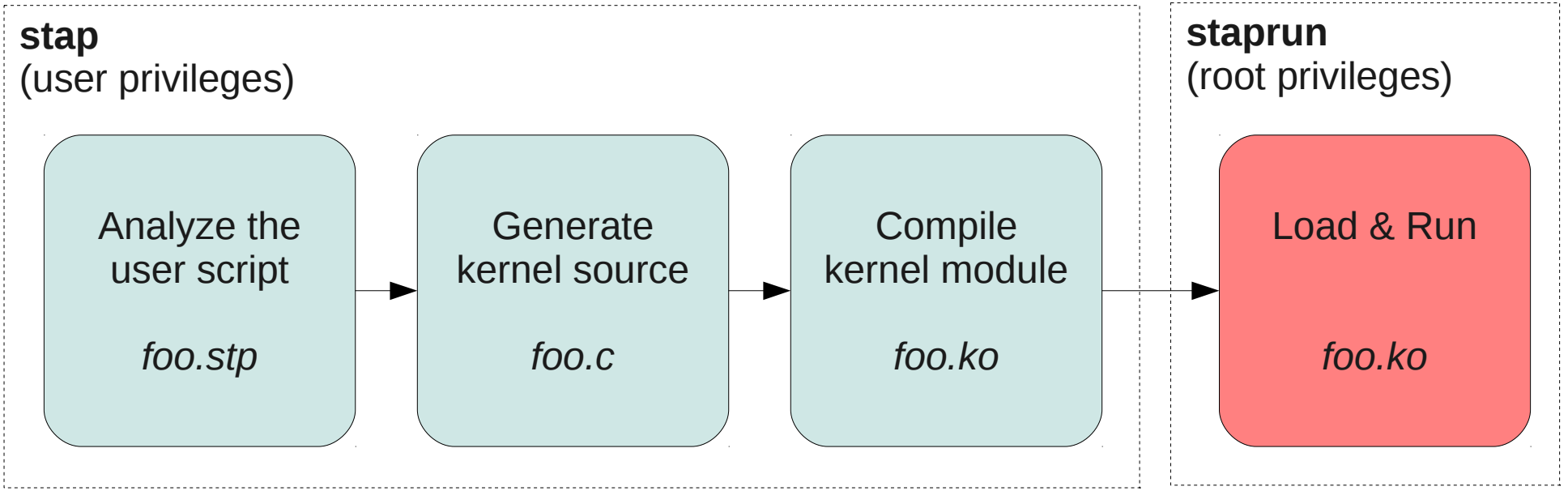
```
global on_times

probe scheduler.cpu_on {
  if (pid() == target())
    on_times[tid()] = local_clock_ns()
}

probe scheduler.cpu_off {
  if (tid() in on_times) {
    delta = local_clock_ns() - on_times[tid()]
    printf("%3d  %5d  %5d  %10d\n",
           cpu(), pid(), tid(), delta)
    delete on_times[tid()]
  }
}

probe begin {
  printf("%3s  %5s  %5s  %10s\n",
         "CPU", "PID", "TID", "DELTA(ns)")
}
```


SystemTap operation



SystemTap limitations

- Generating kernel modules
  - Kernel doesn't keep a fixed API
  - Inherently out-of-tree, thus politically incorrect
- Running kernel modules
  - Requires privilege to load
    - Root, also groups stapdev and stapusr
  - Mistakes are fatal
    - Mitigated by protected stap language

Introducing Dyninst

- API to insert code into a running program
  - High performance
  - Machine independent
- Examples tools:
  - Open|SpeedShop, TAU, COBI, Extrae, STAT, codeCoverage, unstrip
- Project lead by the University of Maryland and the University of Wisconsin-Madison
- Released under LGPL 2.1+
- http://www.dyninst.org/

Dyninst components

Example Dyninst Tools

- Open|SpeedShop
  - Performance analysis tool
  - From single nodes to whole clusters
- unstrip
  - Restore symbols *heuristically*
- CRAFT
  - Floating point analysis, patch doubles to singles

Dyninst attractions for SystemTap

- No special privilege required for own processes
- Dynamic operation
  - Run processes directly
  - Attach to live processes
- Insert arbitrary code
  - e.g. Run a SystemTap handler
- Runs instrumentation in-process
  - Not even a ring transition
- Fast!

Microbenchmark

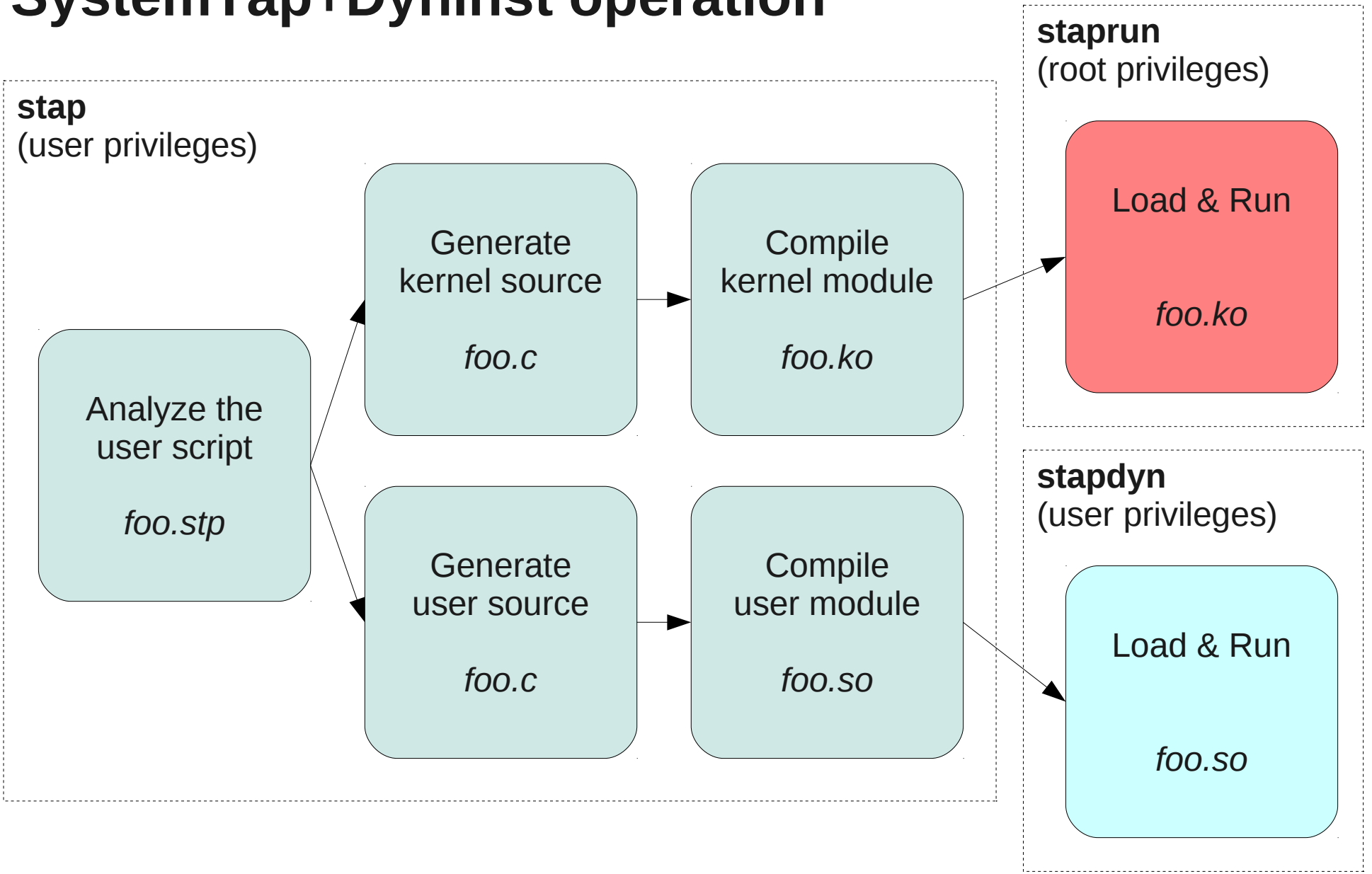
Single-threaded – 10M NOP loops

	User (ms)	System (ms)	µs/probe
base	0	0	
uprobes	870	5780	0.67
dyninst	590	0	0.06

Multi-threaded – 8 simultaneous 10M NOP loops

	User (ms)	System (ms)	µs/probe
base	40	0	
uprobes	9060	342680	4.40
dyninst	105290	8470	1.42

SystemTap+Dyninst operation



SystemTap+Dyninst limitations

- Limited process visibility
  - Only what's accessible by ptrace
  - No practical system-wide monitoring
- Subset of SystemTap events
  - YES: process.*, timers
  - NO: kernel.*, perf
- Only a single mutator for any given process
  - Thanks ptrace!

Example: Tracing SDT

- SDT = Statically Defined Tracepoints
- dynsdt: standalone dyninst app
  - Discover SDT in target app and libraries
  - Instrument each point with a printf
  - Run and trace!

- Becomes a one-liner in SystemTap:

```
probe process.mark(“*”) { println($$name) }
```


Integration status

- Generating userspace modules
- Connecting probe address to a handler function
- (in-progress) stapdyn loader

- TODO
  - Data transport to mutator
  - Split runtime per-thread
  - Combine state across processes

Contact

- http://sourceware.org/systemtap
- systemtap@sourceware.org
- Josh Stone <jistone@redhat.com>

- http://www.dyninst.org/