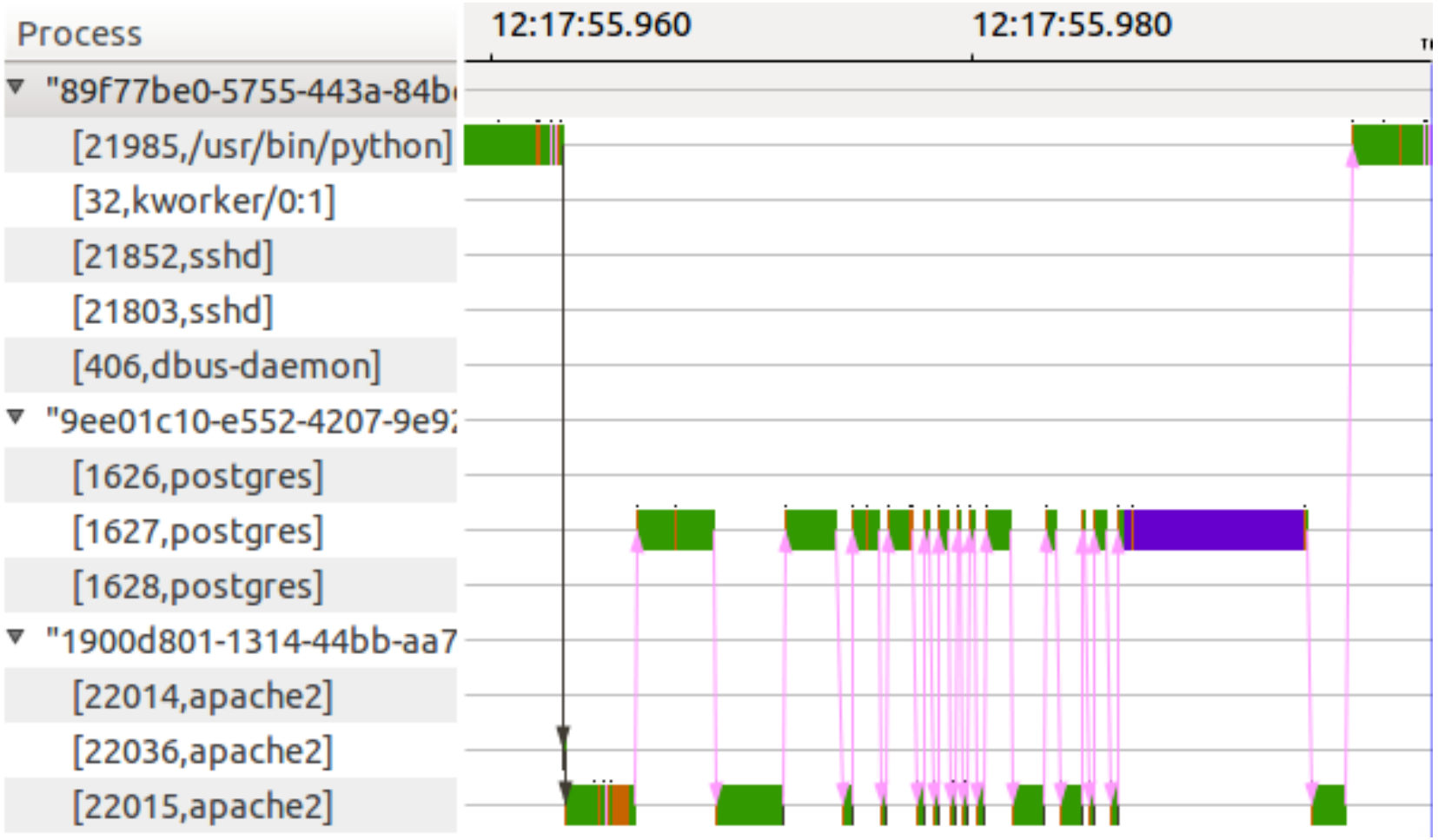
Execution path profiling

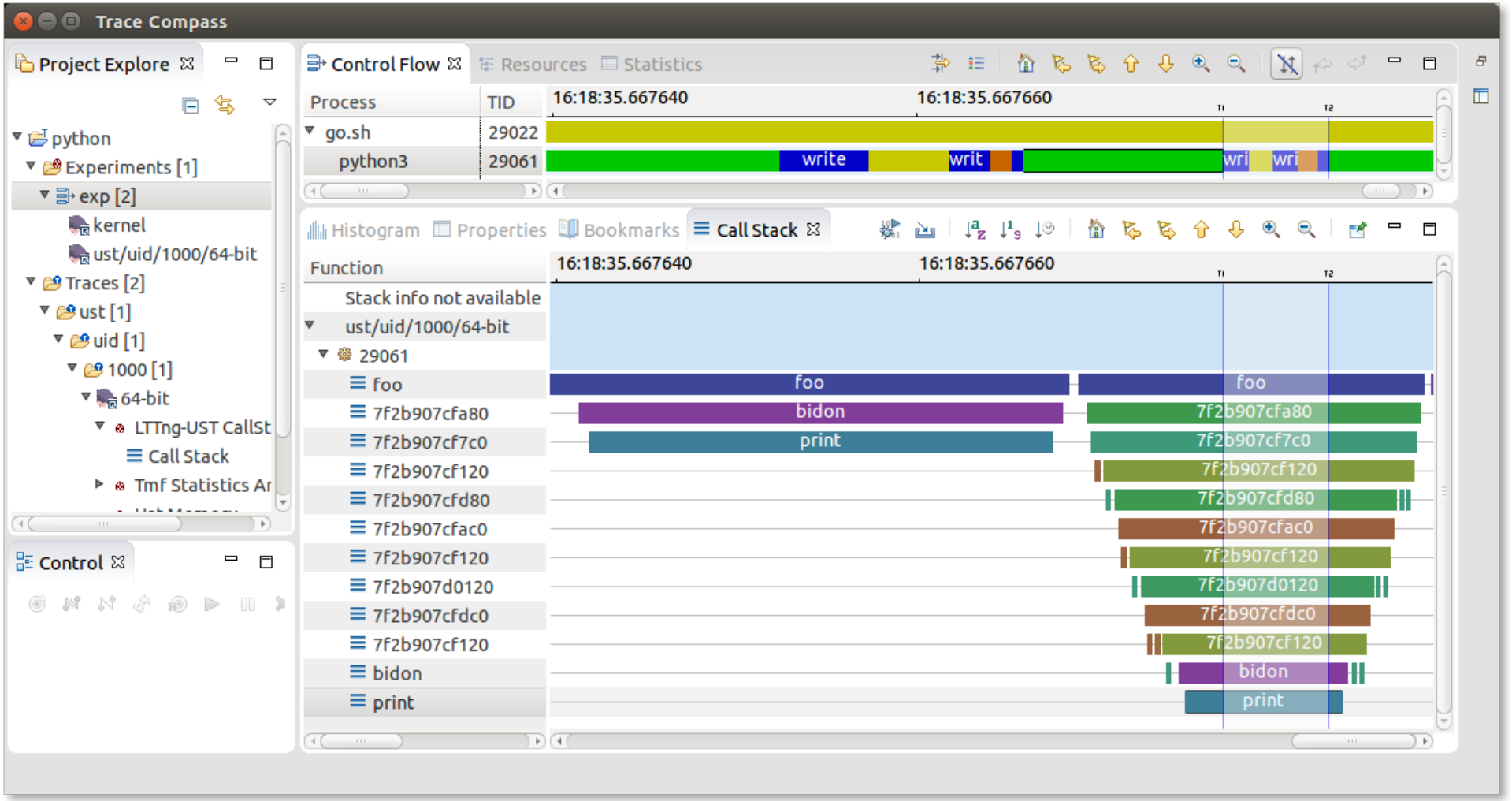
Progress Report Meeting
May 13th 2015

Francis Giraldeau
francis.giraldeau@polymtl.ca

Under the direction of Michel Dagenais
DORSAL Lab, École Polytechnique de Montréal

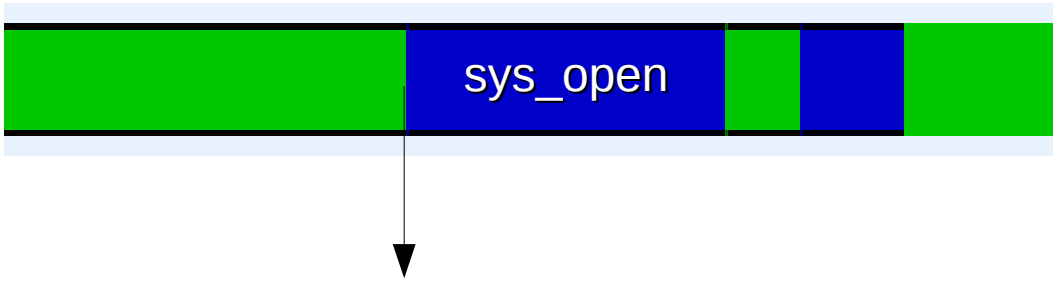

Process
12:17:55.960
12:17:55.980
"89f77be0-5755-443a-84b
[21985,/usr/bin/python]
[32,kworker/0:1]
[21852,sshd]
[21803,sshd]
[406,dbus-daemon]
"9ee01c10-e552-4207-9e9
[1626,postgres]
[1627,postgres]
[1628,postgres]
"1900d801-1314-44bb-aa7
[22014,apache2]
[22036,apache2]
[22015,apache2]

LTTng-UST cyg profile



Bridge the gap between kernel trace and executing code (1)

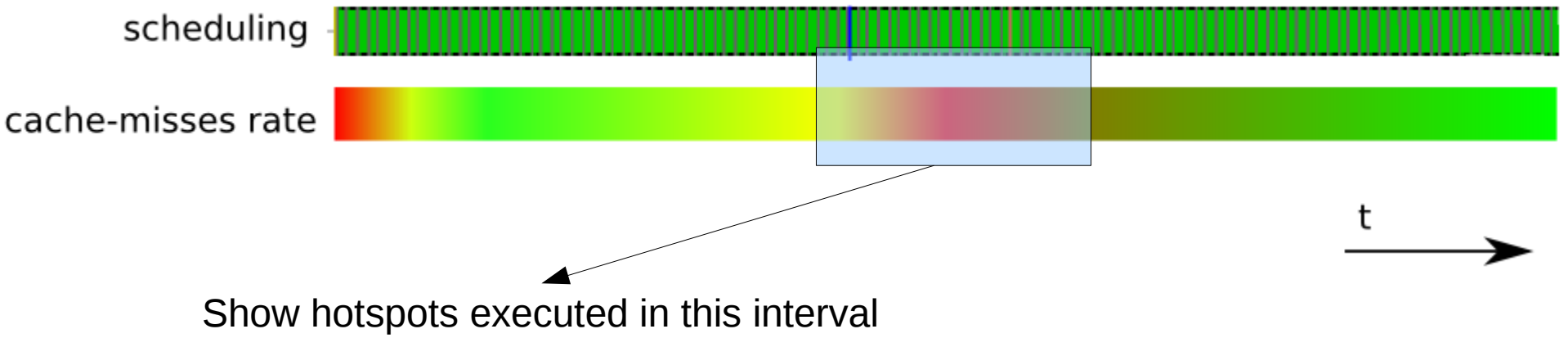
- Balance between resolution and overhead
- Stack trace on system event
  - As an event context for the general case




```
#0  sys_open in libc6
#1  some_func at mylib.c
#2  other_func at myapp.c
#3  main at main.c
```

Bridge the gap between kernel trace and executing code (2)

- PMU according to time
  - Global count map per symbol flatten the time



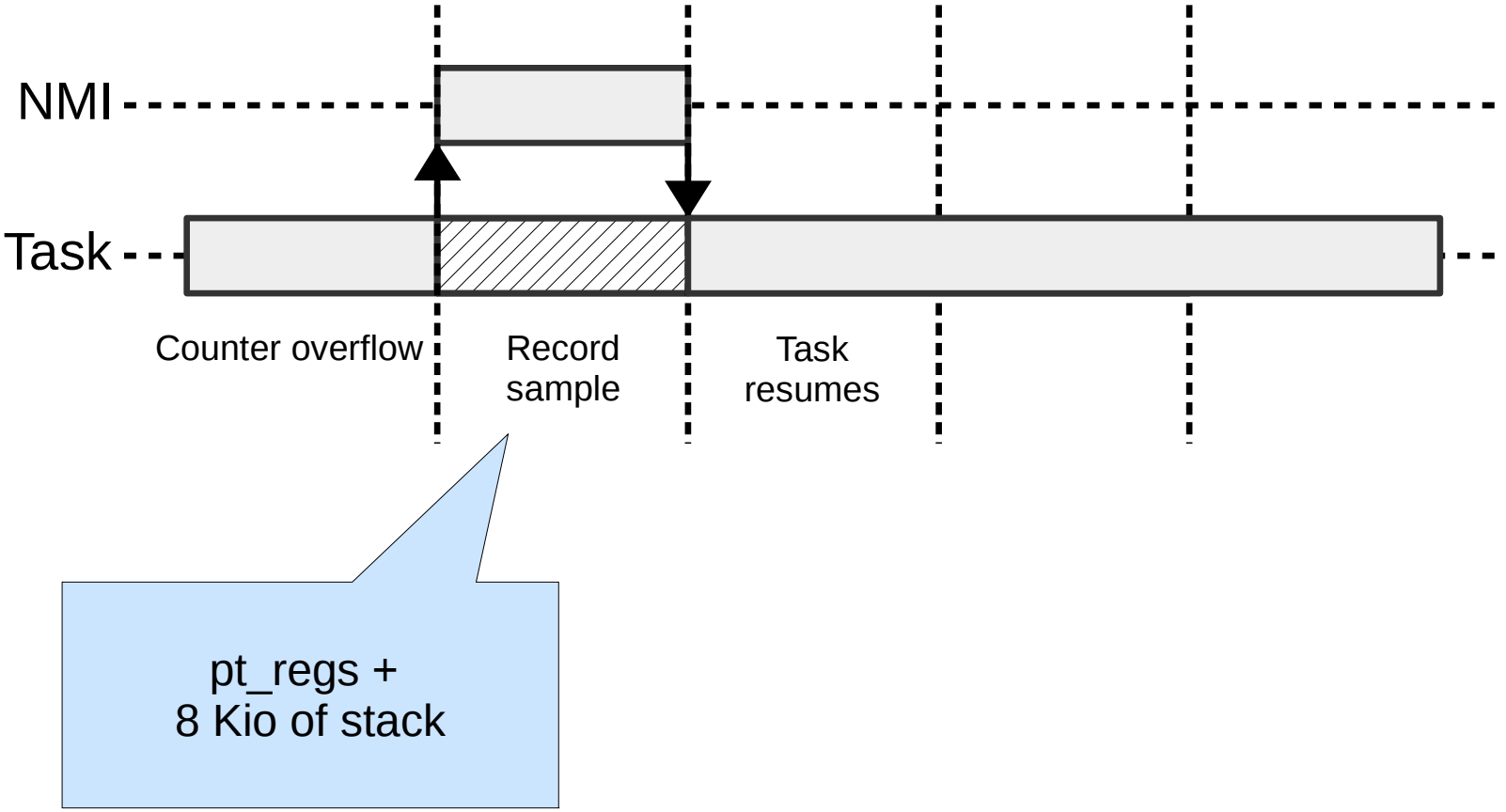
Profilers

- Instrumenting, compiler-assisted (ex. gprof)
  - Block level and/or function level
  - Function counts, timestamps and/or hardware counter values
- Dynamic binary translation (ex. valgrind)
  - Generic of instructions, misses
- Sampling (ex: perf)
  - Timer based (SIGPROF)
  - Hardware counter overflow

```c
#include <stdio.h>
#include <stdlib.h>

static volatile long x = 0;

static void foo(int depth);

__attribute__((noinline))
static void baz(int depth)
{
    if (depth > 0)
        foo(--depth);
    x++;
}

__attribute__((noinline))
static void bar(int depth)
{
    baz(depth);
}

__attribute__((noinline))
static void foo(int depth)
{
    bar(depth);
}

int main(void)
{
    printf("start\n");
    int i = 0;

    for(i=0; i<0xffffff; i++)
        foo(10);

    printf("done x=%ld\n", x);
    return 0;
}
```

Experiment	Elapsed	Overhead
baseline	2.84	-
gprof	6.13	115.8%
valgrind	24.71	770.1%
perf	2.96	4.2%

Perf internals



Runtime support

- C/C++ (built-in)
- Java
  - Agent to save JIT code symbol table:
  - https://github.com/jrudolph/perf-map-agent
- JavaScript
  - V8 can save JIT code and symbols to ELF
  - https://codereview.chromium.org/70013002
- What about CPython and interpreted languages in general?

Interpreter

“An interpreter appears to directly execute the operations specified in the source program in inputs supplies by the user.”

Compilers, Principles, Techniques and Tools, 2nd Edition

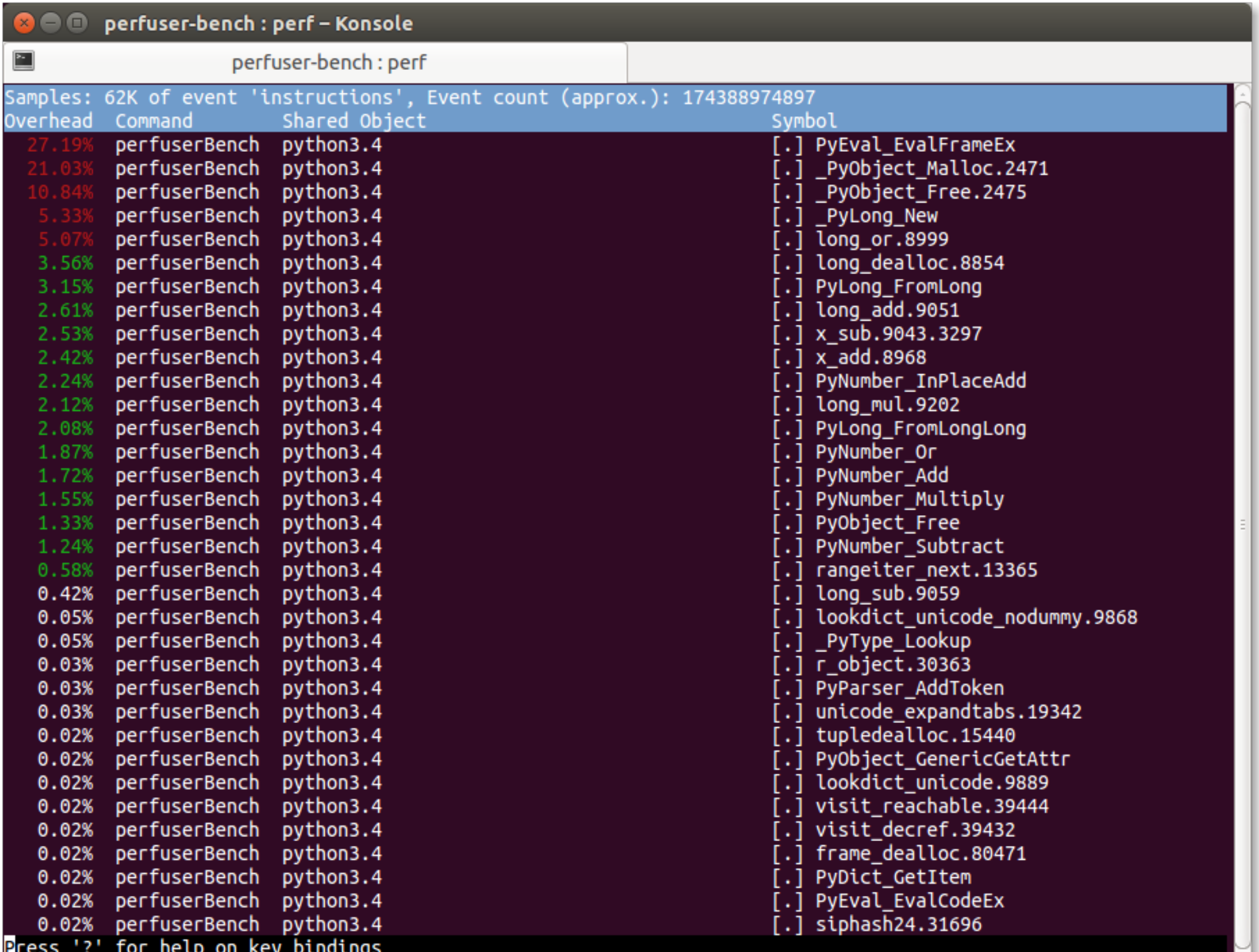
CPython interpreter

```
import sys, traceback
from linuxProfile import api

class Foo(object):
    def __init__(self):
        pass
    def foo(self):
        self.bar()
    def bar(self):
        self.baz()
    def baz(self):
        api.traceback()

def test_traceback_ust():
    foo = Foo()
    foo.foo()
```

```
$ gdb python3

(gdb) break __do_traceback

(gdb) run /usr/bin/nosetests3 -s
nosetests/test_traceback.py

Breakpoint 1, __do_traceback

(gdb) bt
#0  __do_traceback
#1  do_traceback
#2  call_function
#3  PyEval_EvalFrameEx
#4  fast_function
#5  call_function
#6  PyEval_EvalFrameEx
[…]
#112 run_file
#113 Py_Main
#114 main
```

baz

bar


perfuser-bench : perf – Konsole
perfuser-bench : perf
Samples: 62K of event 'instructions', Event count (approx.): 174388974897
Overhead Command Shared Object Symbol
27.19% perfuserBench python3.4 [.] PyEval_EvalFrameEx
21.03% perfuserBench python3.4 [.] _PyObject_Malloc.2471
10.84% perfuserBench python3.4 [.] _PyObject_Free.2475
5.33% perfuserBench python3.4 [.] _PyLong_New
5.07% perfuserBench python3.4 [.] long_or.8999
3.56% perfuserBench python3.4 [.] long_dealloc.8854
3.15% perfuserBench python3.4 [.] PyLong_FromLong
2.61% perfuserBench python3.4 [.] long_add.9051
2.53% perfuserBench python3.4 [.] x_sub.9043.3297
2.42% perfuserBench python3.4 [.] x_add.8968
2.24% perfuserBench python3.4 [.] PyNumber_InPlaceAdd
2.12% perfuserBench python3.4 [.] long_mul.9202
2.08% perfuserBench python3.4 [.] PyLong_FromLongLong
1.87% perfuserBench python3.4 [.] PyNumber_Or
1.72% perfuserBench python3.4 [.] PyNumber_Add
1.55% perfuserBench python3.4 [.] PyNumber_Multiply
1.33% perfuserBench python3.4 [.] PyObject_Free
1.24% perfuserBench python3.4 [.] PyNumber_Subtract
0.58% perfuserBench python3.4 [.] rangeiter_next.13365
0.42% perfuserBench python3.4 [.] long_sub.9059
0.05% perfuserBench python3.4 [.] lookdict_unicode_nodummy.9868
0.05% perfuserBench python3.4 [.] _PyType_Lookup
0.03% perfuserBench python3.4 [.] r_object.30363
0.03% perfuserBench python3.4 [.] PyParser_AddToken
0.03% perfuserBench python3.4 [.] unicode_expandtabs.19342
0.02% perfuserBench python3.4 [.] tupledealloc.15440
0.02% perfuserBench python3.4 [.] PyObject_GenericGetAttr
0.02% perfuserBench python3.4 [.] lookdict_unicode.9889
0.02% perfuserBench python3.4 [.] visit_reachable.39444
0.02% perfuserBench python3.4 [.] visit_decref.39432
0.02% perfuserBench python3.4 [.] frame_dealloc.80471
0.02% perfuserBench python3.4 [.] PyDict_GetItem
0.02% perfuserBench python3.4 [.] PyEval_EvalCodeEx
0.02% perfuserBench python3.4 [.] siphash24.31696
Press '?' for help on key bindings

Python call stack

```
static size_t
__do_traceback(PyFrameObject *frame, struct frame *tsf, int max)
{
    size_t depth = 0;
    while (frame != NULL && depth < max) {
        frame->f_code->co_filename;
        frame->f_code->co_name;
        item->lineno = PyFrame_GetLineNumber(frame);
        frame = frame->f_back;
        depth++;
    }
    return depth;
}
```

Approaches

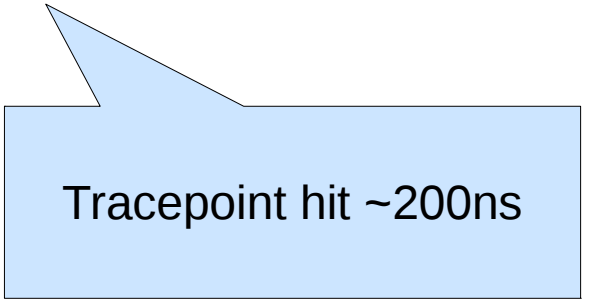
- Kernel space
  - Needs to walk user space structure
  - Depends on arch, compiler flags, Python version, etc.
  - Structure may not be coherent (barrier required between field and frame assign)
- User space
  - Trivial and safe
  - But the NMI occurs in kernel space

Research questions

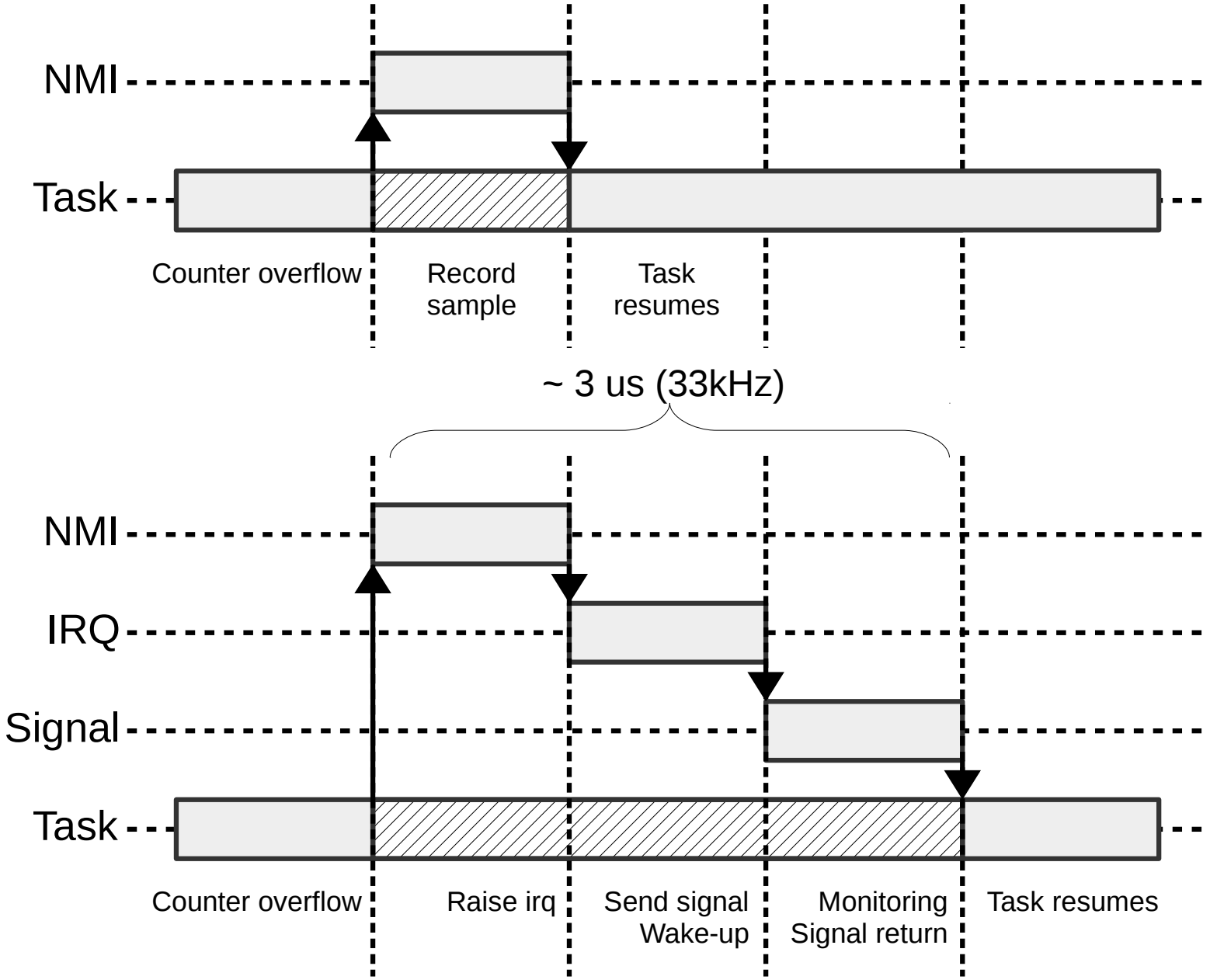
- How can we forward the NMI to user space?
- Can we use do that for performance profiling?
- What is the cost and overhead?

Signals

- The user space register a signal handler
- The kernel saves registers, setups a signal frame and sets the RIP to the signal handler
- On return to user space, the signal handler executes
- At the end of the signal handler, sigreturn is called, the kernel undo the signal frame and restore registers
- The program resumes
- SIGUSR1 microbenchmark: 879 ns



Architecture



```
static int
sampling_do_open(PyObject *obj)
{
    […]
    /* install handler */
    sigact.sa_sigaction = handle_sigio;
    sigact.sa_flags = SA_SIGINFO;
    sigaction(SIGIO, &sigact, &oldsigact);

    /* Open the perf event */
    attr.disabled = 0; /* do not start the event yet */
    attr.watermark = 1;
    attr.wakeup_events = 1;
    fd = sys_perf_event_open(&attr, tid, -1, -1, 0);

    /* Configure fasync */
    fown.type = F_OWNER_TID;
    fown.pid = tid;
    fcntl(fd, F_SETOWN_EX, &fown)
    flags = fcntl(ev->fd, F_GETFL);
    fcntl(ev->fd, F_SETFL, flags | FASYNC | O_ASYNC)

    /* enable counter for one shot */
    ioctl(xev->fd, PERF_EVENT_IOC_REFRESH, 1);
    ioctl(xev->fd, PERF_EVENT_IOC_ENABLE, 0);
}
```

```c
static void
handle_sigio(int signo, siginfo_t *info, void *data)
{
    struct frame tsf[DEPTH_MAX];
    size_t depth = 0;

    depth = __do_traceback(get_top_frame(), tsf, DEPTH_MAX);
    tracepoint(python, traceback, tsf, depth);
    ioctl(fd, PERF_EVENT_IOC_REFRESH, 1);
}
```

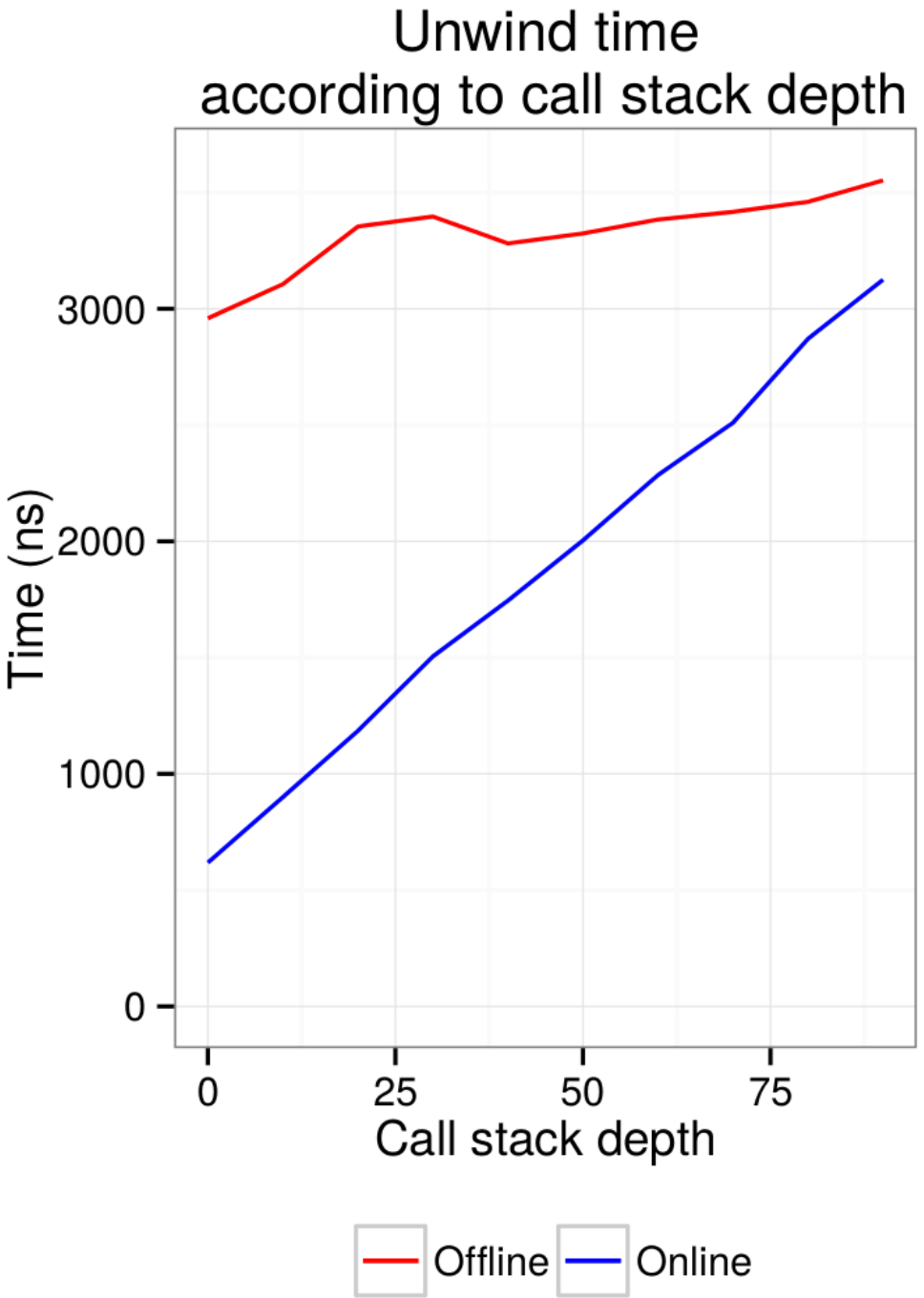
Factoring out monitoring cost

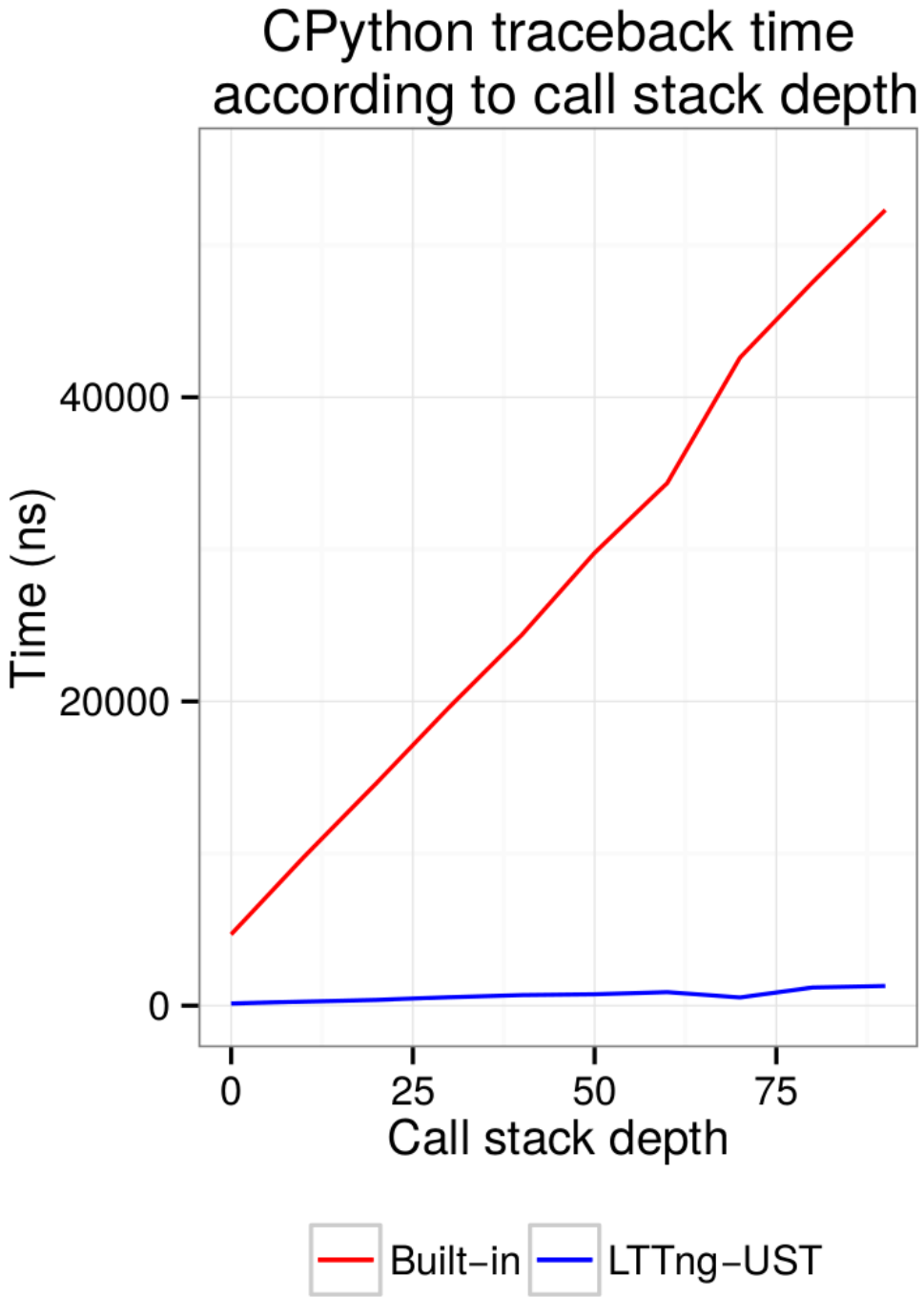
- One shot enabling means signal handler does not increment performance counter.
- If monitoring cost increases, the overhead increases, but does not affect directly the results.
- There is no correlation between the work performed inside the signal handler and the number of samples produced. $cor = 0.02, R^2 = 6.9 \times 10^{-4}$

```
ioctl(fd, PERF_EVENT_IOC_REFRESH, 1);
```

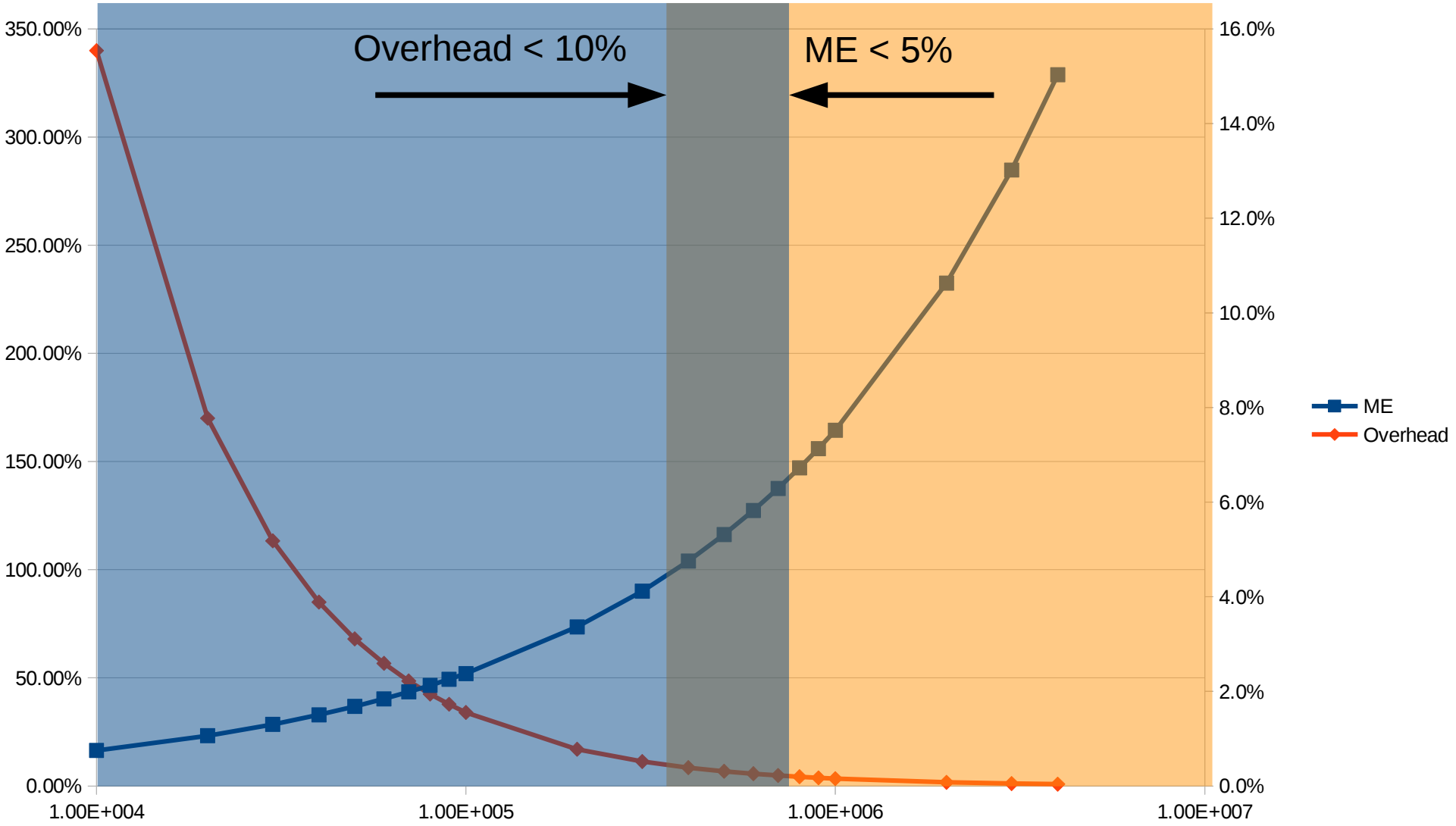
TABLE 1: Profile measurement accuracy

Fn	Exp.	cProfile		PyPerf	
		time (s.)	%	count (10^3)	%
f0	50.0	1.997	50.1	186.6	50.0
f1	20.0	0.799	20.0	74.6	20.2
f2	15.0	0.599	15.0	55.8	14.9
f3	5.0	0.199	5.0	18.6	5.0
f4	5.0	0.199	5.0	18.6	5.0
f5	1.0	0.040	1.0	3.8	1.0
f6	1.0	0.039	1.0	3.7	1.0
f7	1.0	0.039	1.0	3.6	1.0
f8	1.0	0.039	1.0	3.7	1.0
f9	1.0	0.039	1.0	3.8	1.0
RMS (%)			0.1		0.1


Unwind time
according to call stack depth
Time (ns)
3000
2000
1000
0
0
25
50
75
Call stack depth
Offline
Online


CPython traceback time
according to call stack depth
Time (ns)
40000
20000
0
0
25
50
75
Call stack depth
Built−in
LTTng−UST

Overhead and margin of error according to sampling period



Caveats

- Signals can be masked (and the sample delayed, same as SIGPROF)
- Signal SIGIO can be already used for other purpose (use an RT signal instead)

Demo

Conclusion

- Forward counter events to user-space possible
- Provides accurate profiles
- Reasonable compromise between resolution and user-space

Future work

- Integrate comprehensive call stack support to LTTng-UST and TraceCompass
- Profile report for a given execution path
- Reduce redundancy by using sentinel, save only what changed

One more thing...


01 WINS
84
00 WINS
SHANG TSUNG
KUNG LAO
FINISH HIM!!

SystemTap

- Sampling of Python code from kernel
- Does online unwind and Python traceback
- Uses 2 uprobes on frame entry/exit (~3us per hit)
- PI calculation benchmark: **3.4x slowdown**
- Maximum frequency: **3 kHz (throttled by perf)**
- And requires **root privileges**
- And saves to **text files**
- And required to **patch to Python interpreter**
- And we **can't control** the profiler in the program

Thanks to Professor Michel Dagenais and our partners EfficiOS and Ericsson.

Special thanks to Geneviève Bastien for her excellent work on Trace Compass.

Software:

http://secretaire.dorsal.polymtl.ca/~fgiraldeau/workload-kit/

http://secretaire.dorsal.polymtl.ca/~fgiraldeau/traceset/

https://github.com/giraldeau