Linux/ia64 support for performance monitoring

Stéphane Eranian
HP Labs
Gelato Meeting, May 2004 – UIUC, IL

Outline

- Goals of the interface
- The perfmon-2 kernel interface overview
- Overview of existing tools
- Examples of measurements

Goals of the interface

- Provides a generic interface to access PMU
  - Not dedicated to one app, avoid fragmentation
- Must be portable across PMU models:
  - Almost all PMU-specific knowledge in user level libraries
- Supports per-process monitoring
  - Self-monitoring, unmodified binaries, attach/detach
  - multi-threaded and multi-process workloads
- Supports system-wide monitoring
- Supports counting and sampling
- No modification to applications or system
- Builtin, efficient, robust, secure, simple,documented

Current implementations

- In Linux/ia64 since 2.4.0
- In all 2.4-based kernels: perfmon-1
  - First generation interface
  - Included in SLES-8, RHAS-2.1, RHEL-3.0 (but broken)
  - Several limitations : no monitoring across fork()
- In all 2.6-based kernels: perfmon-2
  - Second generation interface
  - Not backward compatible with perfmon-1
  - Lots of new features and more flexibility
  - Easier to port existing applications (Oprofile, VTUNE)

Perfmon-2 new features

- Perfmon context identified with file descriptor
- Monitoring across pthread_create/fork (i.e., clone2)
  - Using ptrace() to help track creation/exit
- Custom sampling buffer formats
  - Easy to port existing monitoring tools
- Can attach/detach to/from running process
  - Useful for long running daemons
- Event-set multiplexing
  - Minimize effect of limited number of counters
- Scalable counter overflow notification mechanism
  - Exploit Linux file system interface

Perfmon-2 interface

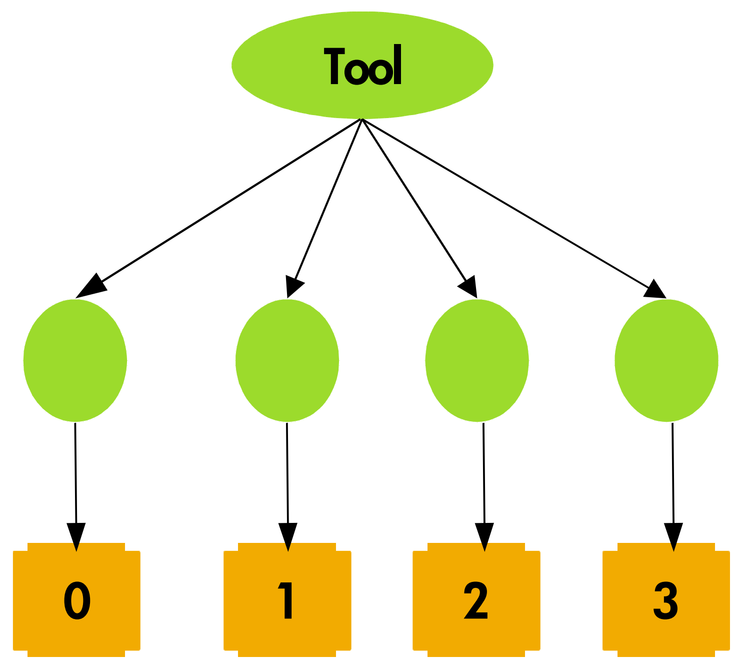
- Uses a system call instead of drivers interface
  - More fexibility (e.g., # arguments)
  - Close ties with the context switch code, exit, fork
  - Kernel compile-time option (CONFIG_PERFMON)
- Perfmon context to enscapsulate PMU state
  - Each context uniquely identified by file descriptor
- Argument similar to ioctl() but you can pass vectors
- Counters are always exported as 64-bit wide

```
int perfmonctl(int fd, int cmd, void *arg, int narg)
```

PFM_CREATE_CONTEXT	PFM_READ_PMDS	PFM_START
PFM_WRITE_PMCS	PFM_LOAD_CONTEXT	PFM_STOP
PFM_WRITE_PMDS	PFM_UNLOAD_CONTEXT	PFM_RESTART

System wide monitoring

- Monitoring of all processes
- Union of cpu-wide sessions:
  - flexibility: different events on each CPU
  - simplicity for implementations
  - Uses process pinning capabilities
- Ability to exclude the idle task
- Does not currently coexist with per-process sessions



Support for event-based sampling

- 64-bit counter overflow notification via message
  - Extracted via read(), supports select/poll, SIGIO
- Number of sampling periods = number of counters
  - Sampling period can be randomized (reproduceable)
  - Each counter has long and short period to hide recovery cost
- task can be stopped on overflow notification
- Kernel level sampling buffer
  - Variable size
  - Remapped to user level to avoid massive copying
- Sampling buffer format controlled by kernel module
  - Perfmon core does not know how samples are recorded

Custom sampling buffer formats

- Why?
  - No single format can satisfy all needs
  - Make it easy to port existing monitoring tools
- How?
  - Each format is implemented as a kernel module
  - Each format is uniquely identified with 128-bit UUID
  - Each format registers a set of callbacks with perfmon core
  - Each format provides a handler callback invoked on counter overflow
- What can a format use?
  - May use buffer allocation+remapping service
  - May use overflow notification mechanism

Event set multiplexing

- To overcome limitation on number of counters
- Each set encapsulates PMU state (PMC/PMD/IBR/DBR)
  - Can define up to 65K sets
  - Possiblity to exclude idle task per set (system-wide only)
- Supported in per-task and system-wide modes
  - counting and sampling support
- Cycling and Switching:
  - Cycling is round-robin
  - Per-set time-based: timeout granularity of clock tick
  - Per-set overflow-based: “after n overflows of counter X”
  - Multiple counters can be triggers in overflow mode
- Can be used to implement counter cascading

Linux/ia64 monitoring tools

- gprof:
  - Flat profile, call graph, recompilation, applications only
- Kernel profiler:
  - Kernel only flat profile, builtin (cmdline profile=)
- Prospect(HP)/OProfile:
  - PMU-based, system-wide flat profile
- VTUNE(Intel):
  - PMU-based, system-wide flat profile, Windows-side GUI
- pfmon/libpfm, qprof, q-syscollect/q-view (HP Labs)
- PAPI toolkit (U. of Tenessee):
  - PMU-based, counting, sampling, uses libpfm

Monitoring complicated workloads

- Can follow across fork/vfork, pthread_create
- Works for counting and sampling
- Supports regular expression to filter binaries of interest
- Example: elapsed cycles for a simple compile

```
$ pfmon --us-c -u -k --follow-all -ecpu_cycles,ia64_inst_retired \
  -- cc e.c -o e

 1,164,772   CPU_CYCLES        /usr/lib/gcc-lib/ia64-linux/2.96/cpp0
 1,295,480   IA64_INST_RETIRED /usr/lib/gcc-lib/ia64-linux/2.96/cpp0
13,758,346   CPU_CYCLES        /usr/lib/gcc-lib/ia64-linux/2.96/cc1
21,863,635   IA64_INST_RETIRED /usr/lib/gcc-lib/ia64-linux/2.96/cc1
 5,708,731   CPU_CYCLES        as
 7,165,599   IA64_INST_RETIRED as
27,046,535   CPU_CYCLES        /usr/bin/ld
35,247,760   IA64_INST_RETIRED /usr/bin/ld
 1,381,134   CPU_CYCLES        /usr/lib/gcc-lib/ia64-linux/2.96/collect2
 1,508,977   IA64_INST_RETIRED /usr/lib/gcc-lib/ia64-linux/2.96/collect2
 1,913,253   CPU_CYCLES        cc
 1,976,590   IA64_INST_RETIRED cc
```

Detailed cycle breakdown

- Can use current pfmon with wrapper script
  - i2prof.pl written by Per Ekman
- Using the current development version of pfmon:
  - exploits event sets multiplexing capabilities

```
$ pfmon -m itanium2-stalls -k -u –system-wide –print-interval – mcf inp.in

# ---------------------------------------------------------------------------------
# %itlb %icache %bra  %unstall %BE   %score %RSE  --------------- D-access ------
#                     exec     flush board        %d1tlb %d2tlb %cache -loaduse-
#                                                               res     %gr   %fr
# ---------------------------------------------------------------------------------
   0.00 0.02     2.81  32.08    10.06 1.19   0.00  0.57    5.28   4.19    43.80 0.00
   0.00 0.02     2.81  32.12    10.06 1.19   0.00  0.57    5.28   4.19    43.77 0.00
   0.00 0.02     2.81  32.09    10.06 1.19   0.00  0.57    5.28   4.19    43.78 0.00
   0.00 0.00     0.08  59.29     0.22 0.05   0.00  0.03    0.01   1.75    38.57 0.01
   0.00 0.00     0.06  54.49     0.16 1.16   0.00  0.46    3.16   3.74    36.76 0.00
   0.00 0.05     2.83  42.14    10.08 1.06   0.02  0.68    4.77   5.69    32.69 0.00
   0.00 0.05     2.79  42.27     9.97 1.07   0.02  0.69    4.88   5.67    32.59 0.00
   0.00 0.03     2.44  41.42     8.74 1.11   0.00  0.55    4.30   4.32    37.09 0.00
   0.00 0.02     2.82  32.07    10.07 1.16   0.00  0.62    5.69   4.46    43.08 0.00
```

Data load cache misses profiles

- Obtained using the Data EARS
- Provides instruction AND data views
- Careful with interpretation: not all misses lead to stalls
- To be included in the next version of pfmon
- Example: mcf instruction and data views

```
#count %self  %cum   %L2    %L3    %RAM   instruction addr
 6358  11.11% 11.11%  3.05%  5.17% 91.77% price_out_impl+0x820<mcf>
 6238  10.90% 22.01% 26.74% 69.93%  3.33% price_out_impl+0x850<mcf>
 5404   9.44% 31.45% 74.43% 24.94%  0.63% bea_compute_red_cost+0x50<mcf>
 5016   8.77% 40.22% 46.69% 33.77% 19.54% bea_compute_red_cost+0xa1<mcf>
 4968   8.68% 48.90% 42.43%  9.98% 47.58% primal_bea_mpp+0x7b1<mcf>
 4878   8.52% 57.42% 36.67% 51.87% 11.46% bea_compute_red_cost+0x90<mcf>

#count %self %cum  %L2    %L3    %RAM  data addr
   37  0.06% 0.06% 62.16% 32.43% 5.41% 0x200000000017ebd0
   32  0.06% 0.12% 75.00% 18.75% 6.25% 0x20000000000d07b0
   29  0.05% 0.17% 68.97% 24.14% 6.90% 0x20000000000e2438
   28  0.05% 0.22% 96.43%  3.57% 0.00% 0x20000000000d3708
   26  0.05% 0.27% 88.46% 11.54% 0.00% 0x20000000000d8c58
```

Kernel level call stack sampling

- Combines kernel stack unwinder with perfmon:
  - On counter overflow, record the call stack
  - Uses a custom sampling buffer format
- Example using the development version of pfmon:

```
$ pfmon -el3_misses --long-smpl-periods=2000 --smpl-periods-random=0xff:10 -k \
  --smpl-module=kcall-stack-ia64 --resolve-addr --system-wide

__copy_user,file_read_actor,do_generic_mapping_read,__generic_file_aio_read,generic_file_aio_read,
do_sync_read,vfs_read,sys_read,ia64_ret_from_syscall

do_anonymous_page,do_no_page,handle_mm_fault,ia64_do_page_fault,ia64_leave_kernel

clear_page,do_anonymous_page,do_no_page,handle_mm_fault,ia64_do_page_fault,ia64_leave_kernel

bh_lru_install,__find_get_block,__getblk,ext3_get_inode_loc,ext3_reserve_inode_write,
ext3_mark_inode_dirty,ext3_dirty_inode,__mark_inode_dirty,update_atime,link_path_walk,open_namei,
filp_open,sys_open,ia64_ret_from_syscall

end_bio_bh_io_sync,bio_endio,__end_that_request_first,scsi_end_request,scsi_io_completion,
sd_rw_intr,scsi_finish_command,scsi_softirq,do_softirq,ia64_handle_irq,ia64_leave_kernel

filemap_nopage,do_no_page,handle_mm_fault,ia64_do_page_fault,ia64_leave_kernel

scsi_finish_command,scsi_softirq,do_softirq,ia64_handle_irq,ia64_leave_kernel

end_page_writeback,end_buffer_async_write,end_bio_bh_io_sync,bio_endio,__end_that_request_first,
scsi_end_request,scsi_io_completion,sd_rw_intr,scsi_finish_command,scsi_softirq,do_softirq,
ia64_handle_irq,ia64_leave_kernel
```

Data structure profiling(experimental)

- Statistical profile of data structure to improve layout
- combines loads+stores retired events,range restrictions
- Currently user level but can be made to work in kernel

```
static void
walk_list(node_t *list)
{
  while(list) {
    list->total = list->val1 + list->val2;
    list = list->next;
  }
}

field name     :size: offs : line :  cover  :   loads : stores  :
node_t->prev  :  8 : 8    : 0    :  19.98% :   0.00% :   0.00% :
node_t->next  :  8 : 0    : 0    :  19.67% :  32.86% :   0.00% :
node_t->val1  :  8 : 16   : 0    :  20.35% :  33.97% :   0.00% :
node_t->val2  :  8 : 24   : 0    :  20.00% :  33.17% :   0.00% :
node_t->total :  8 : 32   : 0    :  20.01% :   0.00% : 100.00% :
```

Conclusions

- The Itanium® 2 PMU is very powerful
- Monitoring is key to achieving world-class performance
- Linux/ia64 has the most advanced monitoring subsystem of all Linux implementations
- Linux/ia64 already supports a variety of monitoring tools
- We need to develop better, smarter tools for non-experts

Linux/ia64 perfmon resources

Linux/ia64 perfmon resources

- Performance tools developed by HPLabs:
  - pfmon/libpfm, q-tools, q-prof
  - http://www.hpl.hp.com/research/linux
- Intel VTUNE:
  http://ww.intel.com/software/products/vtune
- PAPI
  http://icl.cs.utk.edu/projects/papi
- Prospect:
  http://prospect.sf.net
- OProfile
  http://oprofile.sf.net

Linux/ia64 perfmon resources

- i2prof.pl:
  http://www.pdc.kth.se/~pek/i2prof.pl
- IPF PMU architecture:
  http://developer.intel.com/design/itanium/
- Itanium® 2 PMU specification:
  http://developer.intel.com/design/itanium/manuals.htm

Backup slides

Opcode matching with pfmon

- Constrains monitoring to instructions or patterns
  - Based on opcode, e.g., st8.*
  - Based on functional unit, e.g., M,F,I,B
  - Pattern uses a match+mask fields
  - Not all instructions can be uniquely identified
  - Two opcode matching registers on Itanium® 1 & 2
- Ex.: counting the number of br.cloop instructions:

```
$ pfmon -us-c --opc-match8=0x1400028003fff1fa \
   -e IA64_TAGGED_INST_RETIRED_IBRP0_PMC8 -- foo
          4,999,950,164 IA64_TAGGED_INST_RETIRED_IBRP0_PMC8
```

Range restrictions

- Constrains monitoring to range of data or code
  - Implemented via debug registers (not used as breakpoints)
  - Can specify a range inside the kernel
  - Works for both per-process and system-wide
  - Not all events support range restrictions
- Range must be aligned on size for exact measurements
  - gcc -falign-functions= option can be useful
- Pfmon supports symbol name for the range
- Ex.: how many L2 misses while executing init_tab()

```
$ pfmon –us-c -el2_misses -- foo
           1,245,516 L2_MISSES (misses for the entire execution)
$ pfmon –us-c –irange=init_tab -el2_misses -- foo
              14,456 L2_MISSES (misses for init_tab() only)
```

Sampling branches (BTB)

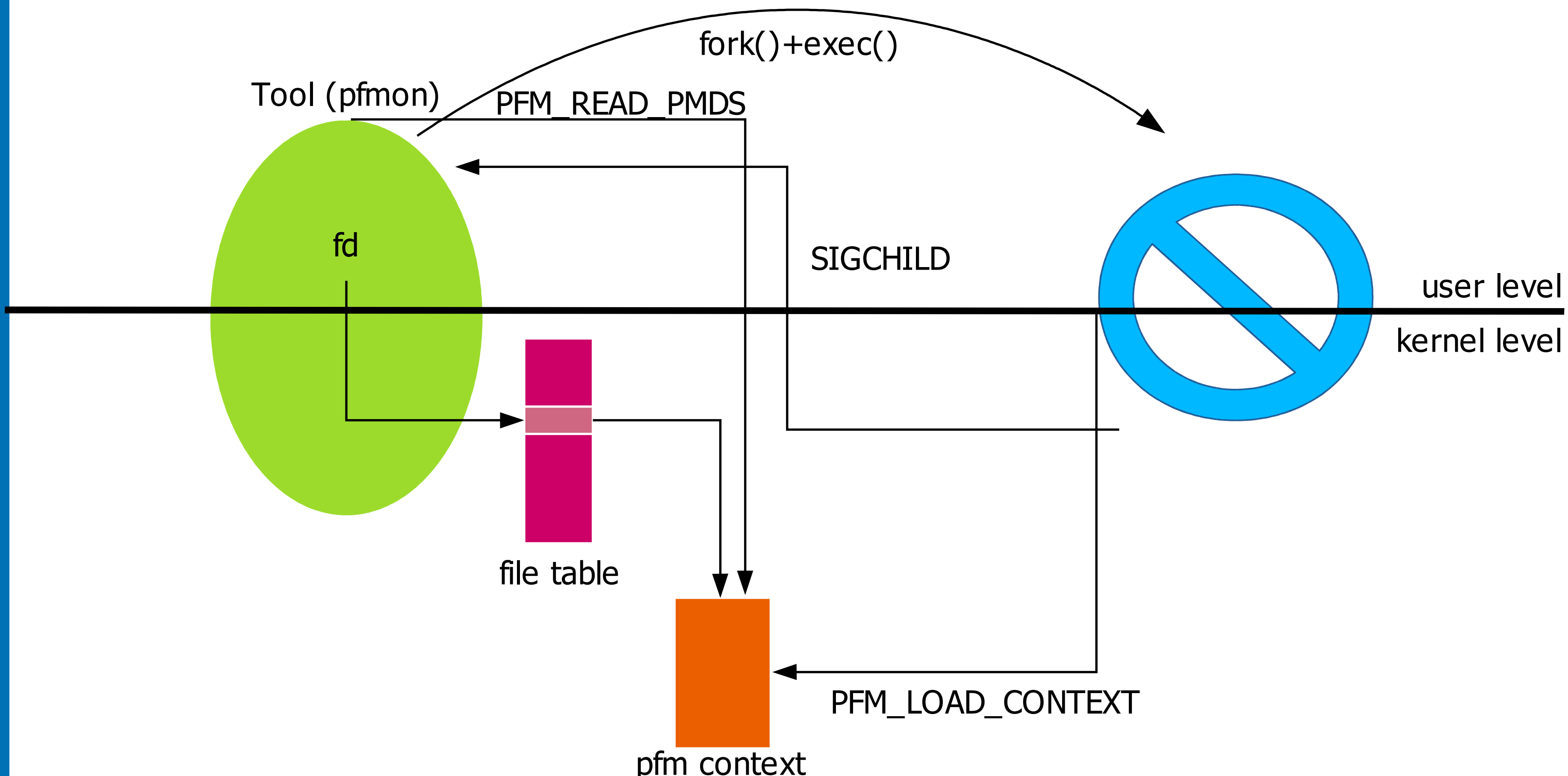
- Capture up to the last 4 branches:
  - Each entry contains source/target addr., prediction outcome
  - Possible to filter branches: taken/not taken, mispredicted
  - Can be combined with EAR to build a path to a cache/tlb miss
- Ex.: sample every 1000 taken branch, record last 4

```
$ pfmon --smpl-periods-random=5:0xff --btb-tm-tk \
    --long-smpl-periods=1000 -ebranch_event -- foo

  entry 231 PID:673 CPU:0 STAMP:0x12957325ac49 IIP:0x40000000000004d0
     last reset : 1004
     branch source address: 0x40000000000004f2
     branch target address: 0x40000000000004c0
     branch taken : yes, prediction: success, pipe flush: no
     ...
  40000000000004f0:[MFB]        nop.m 0x0
  40000000000004f1:             nop.f 0x0
  40000000000004f2:             br.cloop.sptk.few 40000000000004c0
```

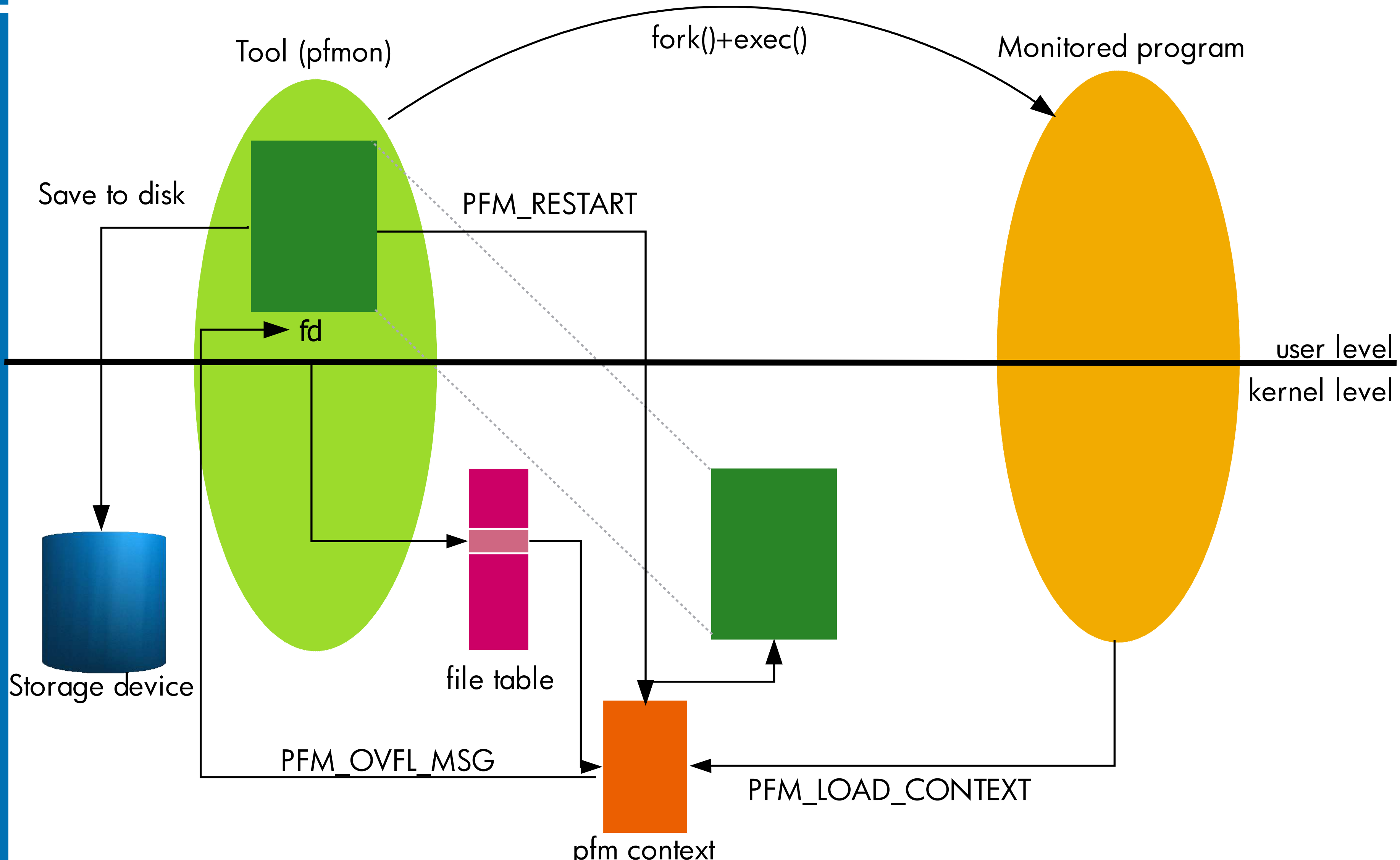
Simple self-monitoring example

```
pfarg_context_t ctx;
pfarg_reg_t pmcs[1], pmds[1];
pfarg_load_t load_args;
pfmlib_input_param_t inp;
pfmlib_output_param_t outp;
pfm_find_event("CPU_CYCLES", &inp.pfp_events[0]);
inp.pfp_plm = PFM_PLM3; inp.pfp_count = 1;
pfm_dispatch_events(&inp, NULL, &outp, NULL);
pmds[0].reg_num = pmcs[0].reg_num = outp.pfp_pmcs[0].reg_num;
pmcs[0].reg_value = outp_pfp_pmcs[0].reg_value;
perfmonctl(0, PFM_CREATE_CONTEXT, &ctx,1);
fd = ctx.ctx_fd;
perfmonctl(fd, PFM_WRITE_PMCS, pmcs, 1);
perfmonctl(fd, PFM_WRITE_PMDS, pmds, 1);
load_arg.load_pid = getpid();
perfmonctl(fd, PFM_LOAD_CONTEXT, &load_args, 1);
perfmonctl(fd, PFM_START, NULL, 0);

/* run code to measure */

perfmonctl(fd, PFM_STOP, NULL, 0);
perfmonctl(fd, PFM_READ_PMDS, pmds, 1);
close(fd);
```

Monitoring an unmodified binary



Typical sampling setup



Event-based sampling with pfmon

- Can sample on any event
- Multiple simultaneous sampling periods supported
  - Can collect several profiles at the same time
- Sampling period can be randomized
  - Very important to avoid biased results
- Ex.: every 100,000 cycles, record ia64_inst_retired

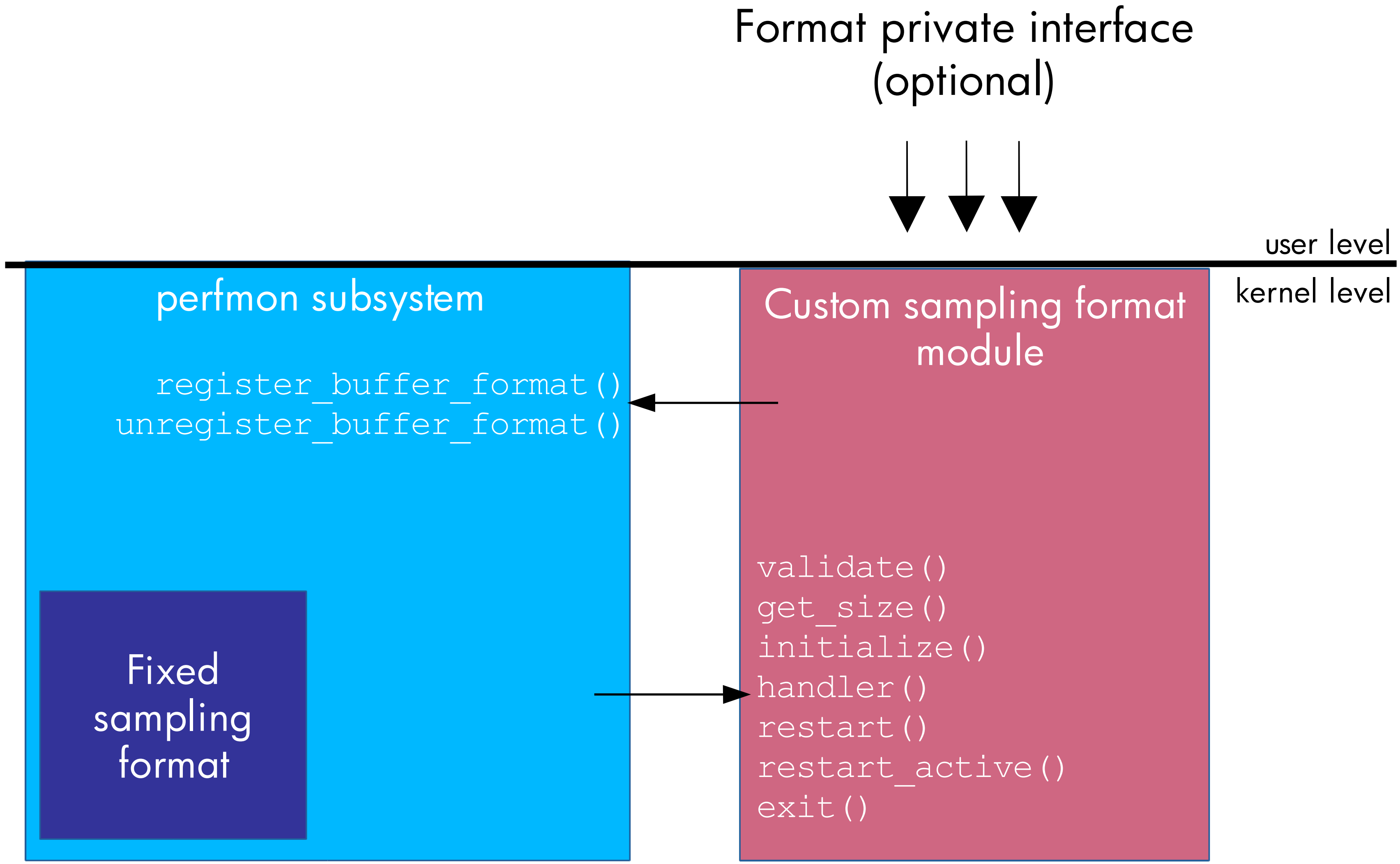
```
$ pfmon -reset-non-smpl -long-smpl-periods=100000 -ecpu_cycle,
  ia64_inst_retired -- foo

entry 6138 PID:4220 CPU:0 STAMP:0x480af599a15a IIP:0x40000000000003a0
        OVFL: 4 LAST_VAL: 100000
        PMD5 : 0x0000000000036ec4 (224964 instructions retired)
entry 6139 PID:4220 CPU:0 STAMP:0x480af59b2c83 IIP:0x40000000000003a0
        OVFL: 4 LAST_VAL: 100000
        PMD5 : 0x0000000000036ec4
```

Sampling cache and TLB misses (EARS)

- Very useful to find where cache/TLB misses occur
  - Cannot be done with naïve IP-based sampling
  - Pinpoint the source of a miss, not the consequence
  - Must be used with sampling
- Ex.: sample every 1000 cache misses with latency > 4 cycles

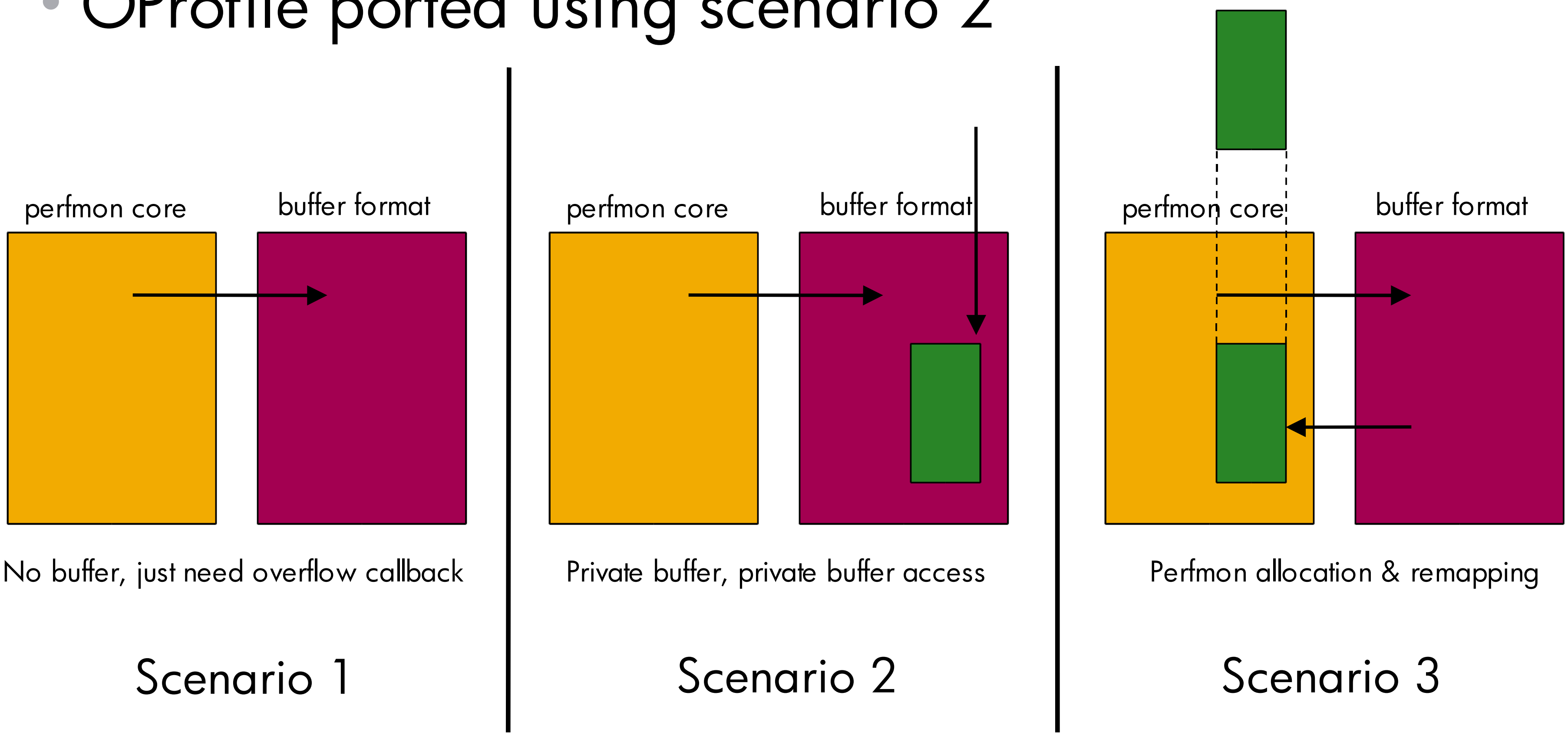
```
$ pfmon --long-smpl-periods=1000 -edata_ear_cache_lat4 – foo


  entry 2000 PID:608 CPU:0 STAMP:0xfe3e1212e5 IIP:0x4000000000000990
          accessed data: 0x2000000000357000
          miss latency : 16 cycles
          inst address : 0x4000000000000981

  4000000000000980:          [MMI]        ld8 r15=[r16]
  4000000000000981:                       ld8 r14=[r17]← miss source
  4000000000000982:                       nop.i 0x0;;
  4000000000000990:          [MMI]        cmp.ltu p7,p6=r14,r15;; ← stall
```

Custom sampling buffer interface



Sampling buffer formats scenarios

- Formats can manage their own private buffers
- Formats can export their data anyway they want
- OProfile ported using scenario 2



pfmon/libpfm for perfmon-2

- Pfmon-3.0 (GNU GPL):
  - Monitoring of unmodified binaries or system-wide
  - Supports counting and event-based sampling
  - Supports all PMU features of Itanium® 1 & 2
  - Uses libpfm for setup
- Libpfm-3.0 (MIT-style):
  - Helps setup PMC registers
  - Provides common interface across PMU models
  - Contains all PMU specific knowledge (event encoding,…)
  - Provides model specific interface for advanced features
  - Does not invoke perfmonctl()
- Use pfmon-2.0/libpfm-2.0 for all 2.4 kernels

i n v e n t