# Perfmon2:
# a standard performance monitoring interface for Linux

**Stéphane Eranian**

<eranian@gmail.com>

# Agenda

- PMU-based performance monitoring
- Overview of the interface
- Current status
- Tools
- Challenges

# What is performance monitoring?

*the action of collecting information related to how an application or system performs*

- Information obtained by instrumenting the code:
  - extract program-level or system-level information
  - statically: compilers (-pg option), explicit code (DTRACE)
  - dynamically (code rewrite): HP Caliper, Intel PIN tool
  - e.g.: count basic-block executions, number of ctxsw/s
- Information obtained from CPU/chipset:
  - extract micro-architectural level information
  - exploit hardware performance counters
  - e.g.: count TLB misses, stall cycles, memory access latency

# Performance Monitoring Unit (PMU)

- CPU HW collecting micro-architectural events:
  - sources: pipeline, system bus, caches, thermal condition, ...
- All modern ISA have a PMU:
  - architected for IA-64, AMD64
  - now finally for Intel IA-32 (starting with Yonah)
  - important for SW to maximize code reuse
- PMU is highly specific to a CPU implementation
  - events
  - extended features

# Diversity of PMU HW

- Dual-core Itanium 2: PMC, PMD, 12 counters (47bits)
  - atomic freeze, opcode filters, range restrictions
  - where cache/TLB misses are (DEAR), Branch Trace Buffer
- AMD64: MSR, 4 counters (40 bits)
  - no atomic freeze
  - Instruction-Based Sampling (Barcelona)
- Intel Core-based: MSR, 5 counters (31 bits)
  - possible atomic freeze
  - fixed counters, Precise Event-Based Sampling (PEBS)
- IBM Power 6: SPRN, 4 counters (32 bits)
  - preset event groups
  - fixed counters (cycles,instr) with no interrupt on overflow

# Diversity of usage models

- Types of measurement:
  - counting or sampling
- Scopes of measurement:
  - system-wide: across all threads running on a CPU
  - per-thread: a designated thread (modified or unmodified)
  - virtualized environments (Xen, KVM)
- Scopes of control:
  - user level programs: monitoring tools, compilers, MRE
  - kernel: SystemTap or VMM
- Scopes of processing:
  - offline: profile-guided optimization (PGO), manual tuning
  - online: dynamic optimization (DPGO)

# Why a kernel interface?

- PMU interface is composed of registers
- Registers writeable only at most privileged level
  - may be readable at the user level (X86, IA64, Power)
- PMU can generate interrupts
  - need a kernel level interrupt handler
- Per-thread monitoring requires kernel hooks
  - context switch, thread creation/termination

# Existing Linux monitoring interfaces

- OProfile (John Levon):
  - included in mainline kernel, most distributions
  - system-wide profiling only, supports all major platforms
- Perfctr (Mikael Pettersson)
  - separate kernel patch
  - provides per-thread, system-wide monitoring
  - designed for self-monitoring (e.g., PAPI), basic profiling
  - supports all IA-32, PowerPC
- VTUNE driver (Intel)
  - open-source driver but specific to VTUNE

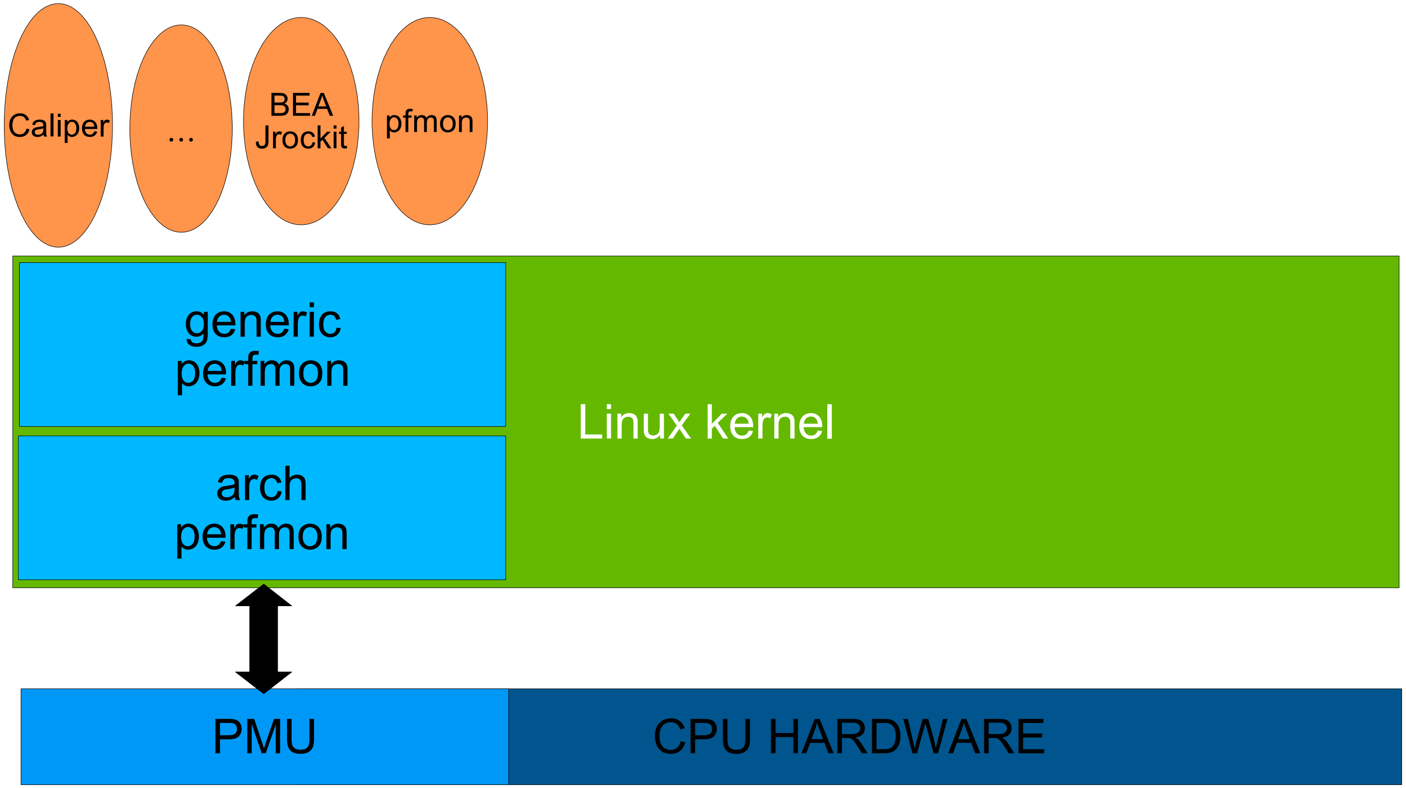
fragmentation, no coordination,

no standard and generic interface exists

# Why a standard interface?

- Currrent HW trends make perf. monitoring crucial
  - SW must change for multi-core, multi-thread, NUMA, power
- Need monitoring tools to understand SW performance
  - requires portable, flexible kernel-level infrastructure
  - support needed across all processor architectures
- Standard attractive for ISV/tool developers
  - can write portable tools, improve code reuse
  - broader market for monitoring products (multi-arch)
- Easier to get accepted in mainline kernel
  - no kernel patching, improved support
  - integration into commercial Linux distributions

# Overall monitoring architecture



Caliper

...

BEA Jrockit

pfmon

generic perfmon

arch perfmon

Linux kernel

PMU

CPU HARDWARE

# Why use Linux?

- Need to build a community to push a new standard
- Open-source provides key advantages:
  - easily available
  - easy to share code
  - easy to involve other developers
  - collaborative effort to speed up development/adoption
- Neutral operating system
- Support for multiple architectures
- Runs on very small to very large systems
- Linux is lacking a good monitoring interface

# Goals of the perfmon2 interface

- A generic programming interface to access the PMU
  - bottom-up approach, no tool in mind
- Be portable across all PMU models/architectures
- Supports system-wide and per-thread monitoring
  - self-monitoring, unmodified binaries, attach/detach
  - multi-threaded and multi-process workloads
- Supports counting and sampling
- No recompilation
- Builtin, efficient, robust, secure, documented

# Perfmon2 interface (1)

- Core interface allows read/write of PMU registers
- Uses the system call approach (rather than driver)
- Perfmon2 context encapsulates all PMU state
  - context uniquely identified by file descriptor
- Leverages existing mechanisms wherever possible
  - file descriptors, signals, `mmap()`, `ptrace()`

```
int pfm_create_context(pfarg_ctx_t *ctx, char *s, void *a, size_t sz)
int pfm_write_pmcs(int fd, pfarg_pmc_t *pmcs, int n);
int pfm_write_pmds(int fd, pfarg_pmd_t *pmcs, int n);
int pfm_read_pmds(int fd, pfarg_pmd_t *pmcs, int n);
int pfm_load_context(int fd, pfarg_load_t *ld);
int pfm_start(int fd, pfarg_start_t *st);
int pfm_stop(int fd);
int pfm_restart(int fd);
int pfm_create_evtsets(int fd, pfarg_setdesc_t *st, int n);
int pfm_delete_evtsets(int fd, pfarg_setdesc_t *st, int n);
int pfm_getinfo_evtsets(int fd, pfarg_setinfo_t *it, int n);
int pfm_unload_context(int fd);
int close(int fd);
```

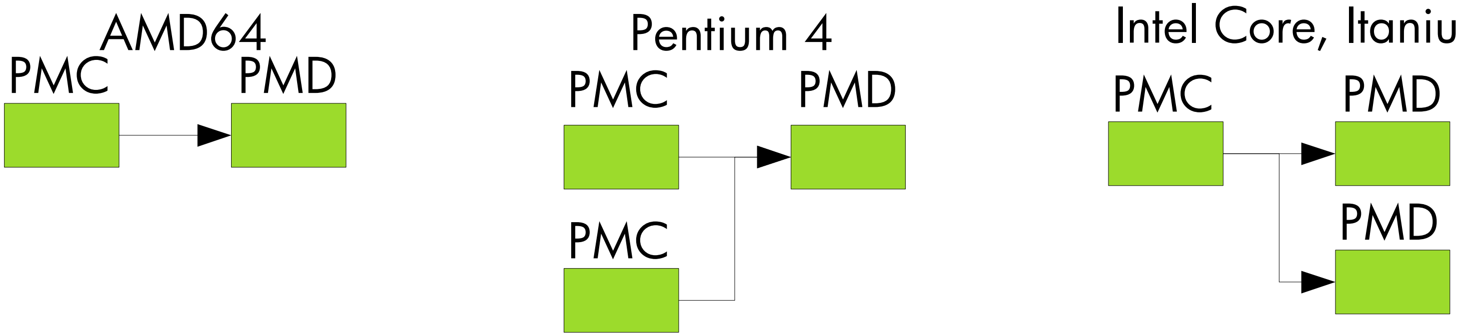
# Perfmon2 interface (2)

*Uniformity makes it easier to write portable tools*

- Registers are always exported as 64-bit wide
  - 64-bit counters emulated via overflow interrupt capability
- Exports logical view of PMU registers
  - PMC: configuration registers
  - PMD: data registers (counters, buffers)
  - use indexed-register style naming: PMC[1], PMC[2],...
- Mapping to actual registers depends on PMU model
  - defined by PMU description kernel module
  - visible in `/sys/kernel/perfmon/pmu_desc`

# Perfmon2 interface (3)

- Same ABI between ILP32 and LP64 models
  - x86_64, ppc64: 32-bit tools run unmodified on 64-bit kernel
  - all syscall arguments have fixed size
- Vector arguments for read/write of PMU registers:
  - portable: decoupled PMC/PMD = no dependency knowledge
  - extensible: no knowledge of number of registers
  - efficient and flexible: can write one or multiple regs per call



# Per-thread session

*thread = kernel visible thread (task)*

- PMU state is saved/restored on context switches
  - multiple per-thread sessions can run concurrently
- Supports one session per thread
- Thread must be stopped to access PMU state
  - leverages `ptrace()`
  - except for self-monitoring
- No inheritance across fork/pthread_create
  - can use `ptrace()`options (`PTRACE_O_TRACE*`)
  - aggregation done by the tool, if needed
- Available to regular users

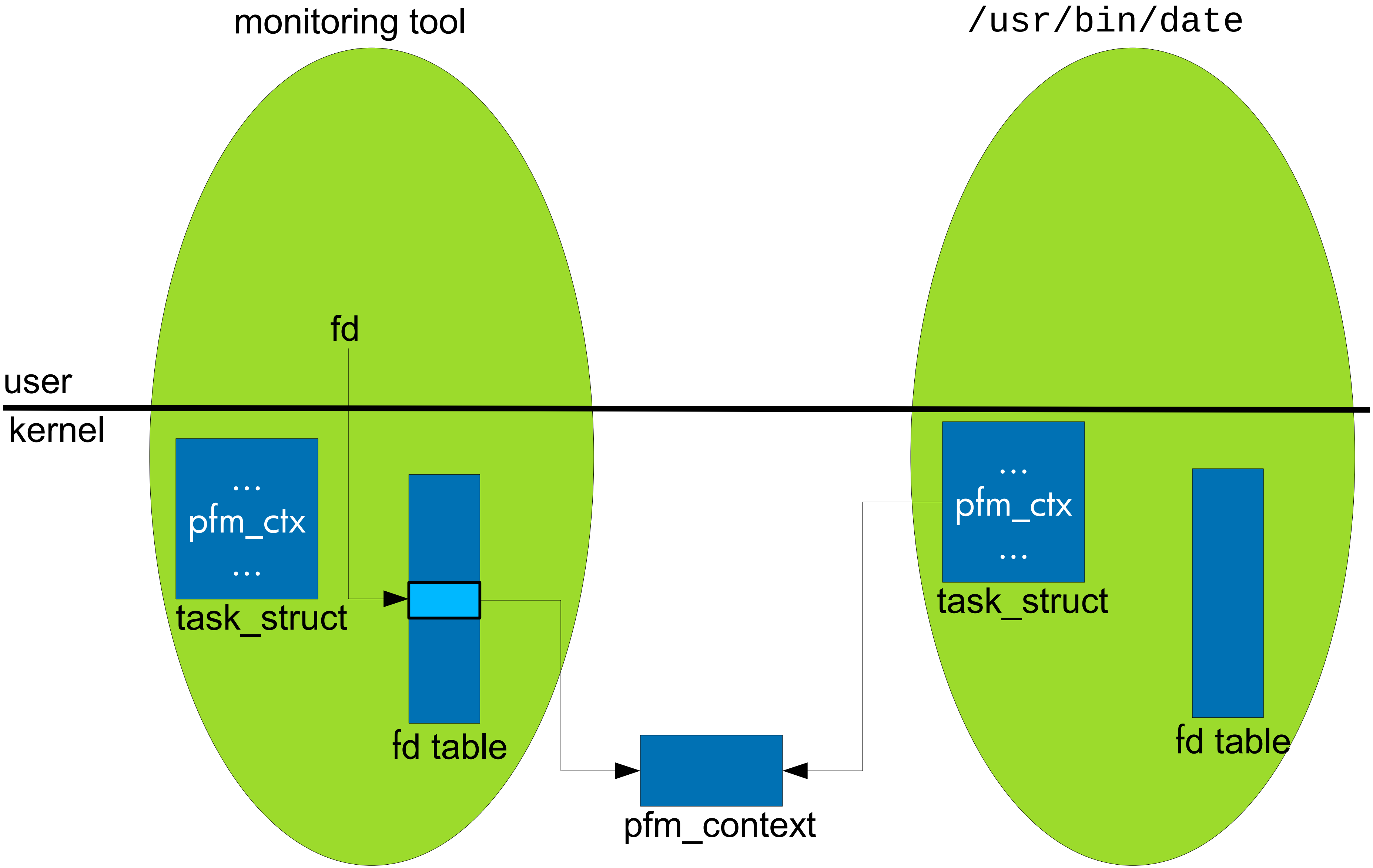
# Why a per-thread mode?

- Needed for simple counting
- Need to follow thread if it migrates to another CPU
  - monitoring cannot change thread's affinity settings
  - if using system-wide => must monitor all CPUs
- Security
  - only see data related to you threads/processes
  - no need to be super-user
- Allow support of certain PMU HW features:
  - Intel Core: PEBS hardware buffer does not record pid/tid
  - Intel Itanium2: Branch Trace Buffer does not record pid/tid
  - Intel dual-core Itanium2 IP-EAR does not record no pid/tid

# Per-thread implementation

- No cost in space if unused:
  - PMU state encapsulated in independent structure
  - adds one pointer to thread structure
- No cost in time if unused:
  - register access latency high,wrmsr/rdmsr slow (>100 cycles)
- lazy context switch:
  - only when PMU used by the thread
  - only save/restore registers actually used
  - SMP uses lazy state restore, similar to fp state

# Typical monitoring session

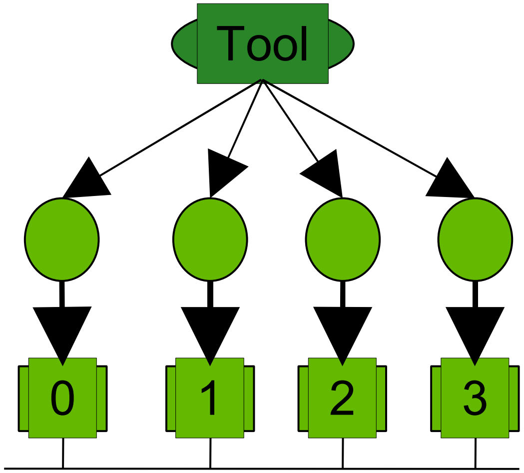


# Basic self-monitoring per-thread session

```
pfarg_ctx_t ctx; int fd;
pfarg_load_t load;
pfarg_pmd_t pd[1]; pfarg_pmc_t pc[1];
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);
pd[0].reg_num = out.pfp_pd[0].reg_num;
pc[0].reg_num = outp.pfp_pc[0].reg_num;
fd = pfm_create_context(&ctx, NULL, 0, 0);
pfm_write_pmcs(fd, pc, 1);
pfm_write_pmds(fd, pd, 1);
load.load_pid = getpid();
pfm_load_context(fd, &load);
pfm_start(fd, NULL);
/* run code to measure */
pfm_stop(fd);
pfm_read_pmds(fd, pd, 1);
printf(“total cycles %”PRIu64”\n”, pd[0].reg_value);
close(fd);
```

# System-wide session

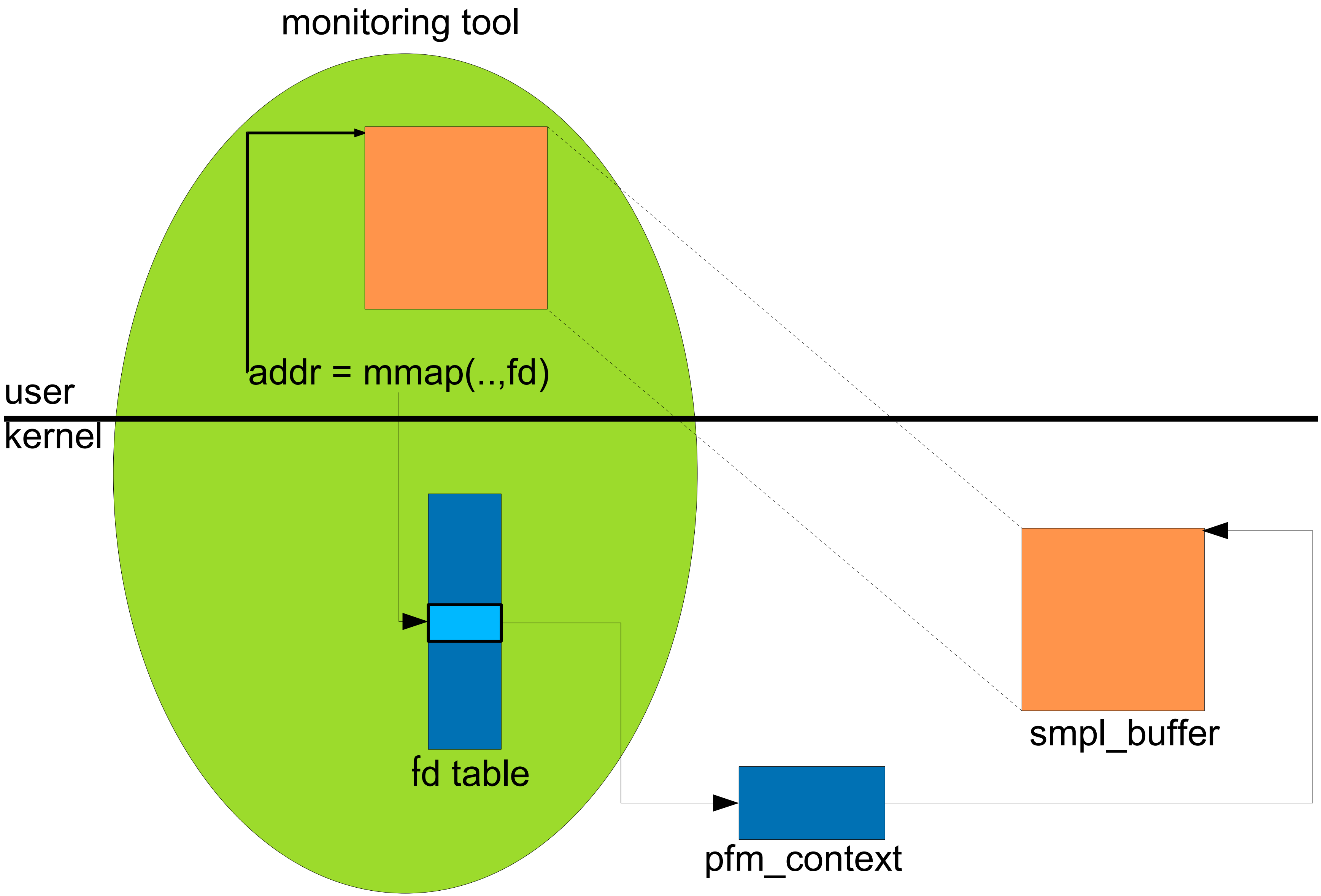
- Monitors across all threads running on one CPU
  - same programming sequence as per-thread
- System-wide SMP = union of CPU-wide sessions:
  - flexibility: measure different metrics on different CPUs
  - scalability: strong affinity (processor, cache)
  - ready for HW buffer, e.g., Intel PEBS
- Mutual exclusion with per-thread session



# Support for sampling

- Supports Event-Based Sampling (EBS)
  - period p expressed as $2^{64}$`-p` occurrences of an event
  - #sampling periods = number of counters
- Can request notification when 64-bit counter overflows
  - notification = message, extracted via `read()`
  - support for `select()`,`poll()`,`SIGIO`
- Optional support for kernel level sampling buffer
  - amortize cost by notifying only when buffer full
  - buffer remapped read-only to user with `mmap()`= zero copy
  - periods can be randomized to avoid biased sample

# Kernel level sampling buffer



# Sampling buffer formats

- No single format can satisfy all needs
  - must keep complexity low and extensibility high
- Extract buffer format from core to a kernel module
- Each format provides at least:
  - string for identification (passed on context creation)
  - counter overflow handler
- Each format controls:
  - where and how samples are stored
  - what gets recorded, how the samples are exported
  - when a user notification must be sent to user

# Existing sampling formats

- Default format (builtin):
  - linear buffer, fixed header followed by optional PMDs values
- OProfile format (IA-64, X86)
  - 10 lines of C, reuse all generic code, small user level changes
- Two-way sampling format:
  - process one part while storing in the other: limit blind spots
- Kernel call stack format (experimental, IA-64):
  - records kernel call stacks (unwinder) on counter overflow
- Precise Event Based Sampling (P4, Intel Core 2 Duo)
  - 100 lines of C, first interface to provide access to feature!

# Event sets and multiplexing

- What is the problem?
  - number of counters is often limited (4 on Itanium®2 PMU)
  - some events cannot be measured together
- What is the solution?
  - create register sets which encapsulate the full PMU state
  - multiplex sets on actual PMU HW
  - sets identified by simple integer
  - sets managed in round-robin fashion
- global counts approximated by simple scaling
- Kernel support needed to minimize overhead
- Supports timeout or overflow-based switching
- Works with counting and sampling

# Security

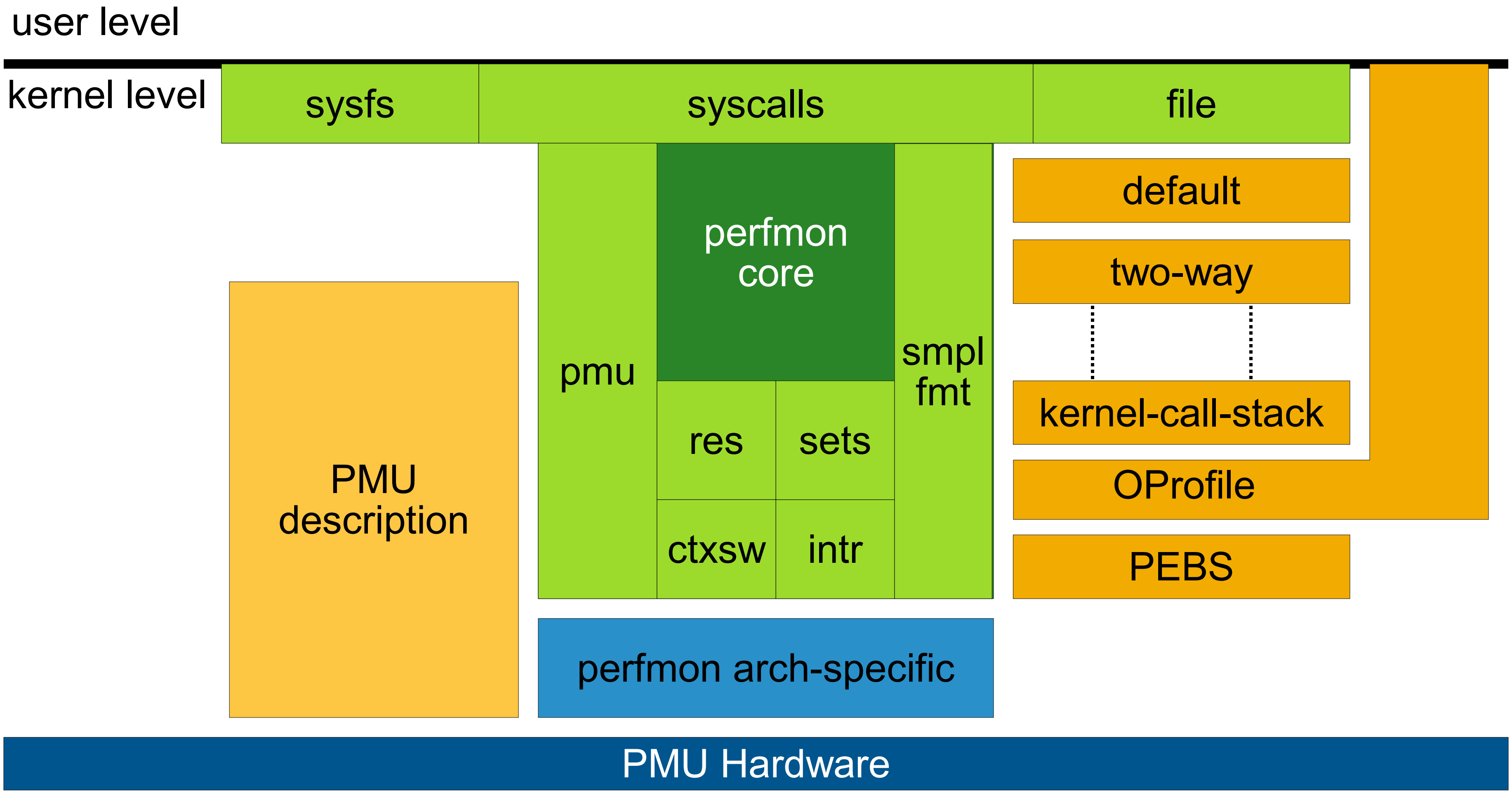
*Cannot assume tools/users are well-behaved*

- Vector arguments, sampling buffers have max. size
  - tuneable via `/sys`
- Per-thread and system-wide contexts
  - can only attach to thread owned by caller
  - each type can be limited to a users group (via `/sys`)
- Reading of PMU registers
  - direct access (some arch):limited to self-monitoring
  - interface access: can only read registers declared used
- PMU interrupt flooding
  - need to add interrupt throttling mechanism

# PMU description module

- Why?
  - Hardware release cycles ≠ Linux distros release cycles
  - needs new hardware support quickly
  - easier bug fixing
- How?
  - move model specific code/data to kernel module
  - keep architecture specific code/data in the core kernel
  - relies on having a PMU architecture: IA-64, AMD64, IA-32?
  - auto-loading on first context creation
- Model-specific data:
  - physical to logical PMU register mappings
  - register sanity checker, if needed

# Perfmon2 architecture summary



# Supported Processors

| HW Vendors | Model | Contributors |
|---|---|---|
| AMD | AMD64 family 6 | Cornell U. |
| AMD | AMD64 family 15 | HPLabs |
| AMD | AMD64 family 16 | AMD |
| Intel | Itanium (all models) | HPLabs |
| Intel | Pentium II, Pentium Pro | Cornell U. |
| Intel | Pentium III, Pentium M | HPLabs |
| Intel | Core Duo/Core Solo | HPLabs |
| Intel | Pentium 4 | Intel |
| Intel | Core 2 Duo | HPLabs |
| MIPS | various | Phil Mucci, SiCortex, Broadcom, Cornell U. |
| IBM | Power4, Power5, PPC970 | IBM |
| IBM | Power 6 | IBM |
| IBM | Cell | IBM, Sony, Toshiba |
| Cray | X2, XT | Cray |
| Sun | Ultra12,Ultra3* | David S. Miller |
| Sun | Niagara1, Niagara2 | David S. Miller |

still missing: ARM, Hitachi SH, Alpha (;->)

# Status

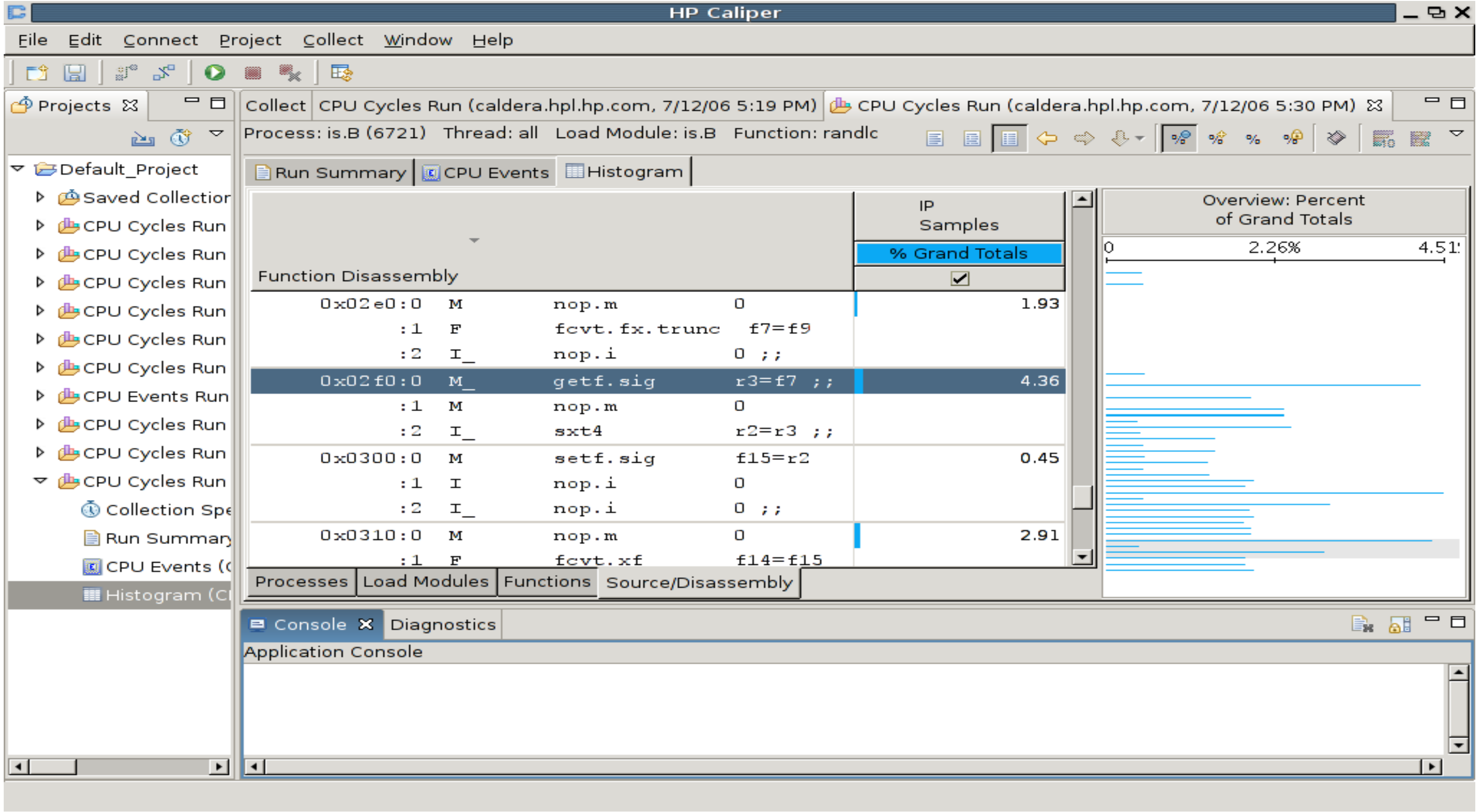
- 1st generation (v2.0) shipping on Linux/ia64 only
  - available since RHEL4/SLES9
- Multi-arch version (v2.8) in development:
  - selected to become Linux monitoring interface
  - reviewed on LKML multiple times, still needs one last pass
  - supported publicly by many HW, Linux vendors
  - once in mainline ⇒ commercial distros (RHEL6, SLES11?)
- In production on Cray X2 and SiCortex systems
- Kernel patch stats:
  - 1.1MB over 5 processor architectures
  - adds ≈7000 lines of C

# Tools(1)

- pfmon/libpfm 3.2 (HP Labs) (GPL/MIT)
  - pfmon: count, collect profiles per-thread or system-wide
  - libpfm helper library: what to measure □□values of PMC
  - IA-64 features: opcode match, DEAR,BTB, range restrictions
  - supports all IA64, X86 (Intel, AMD)
  - contributions for Cell, Power*, SPARC64*, MIPS
  - available on commercial distributions
- PAPI toolkit (U. of Tennessee)
  - popular toolkit to write portable monitoring tools
- BEA JRockit 1.4.2 for Linux/ia64:
  - Dynamic Profile Guided Optimization (DPGO)

# Tools(2)

- Caliper(HP) 4.3 (free for non commercial use)
  - Per-thread, source level profiles, preset metrics, IA-64 ONLY
  - Java-based standalone GUI (local,remote) or Eclipse plug-in

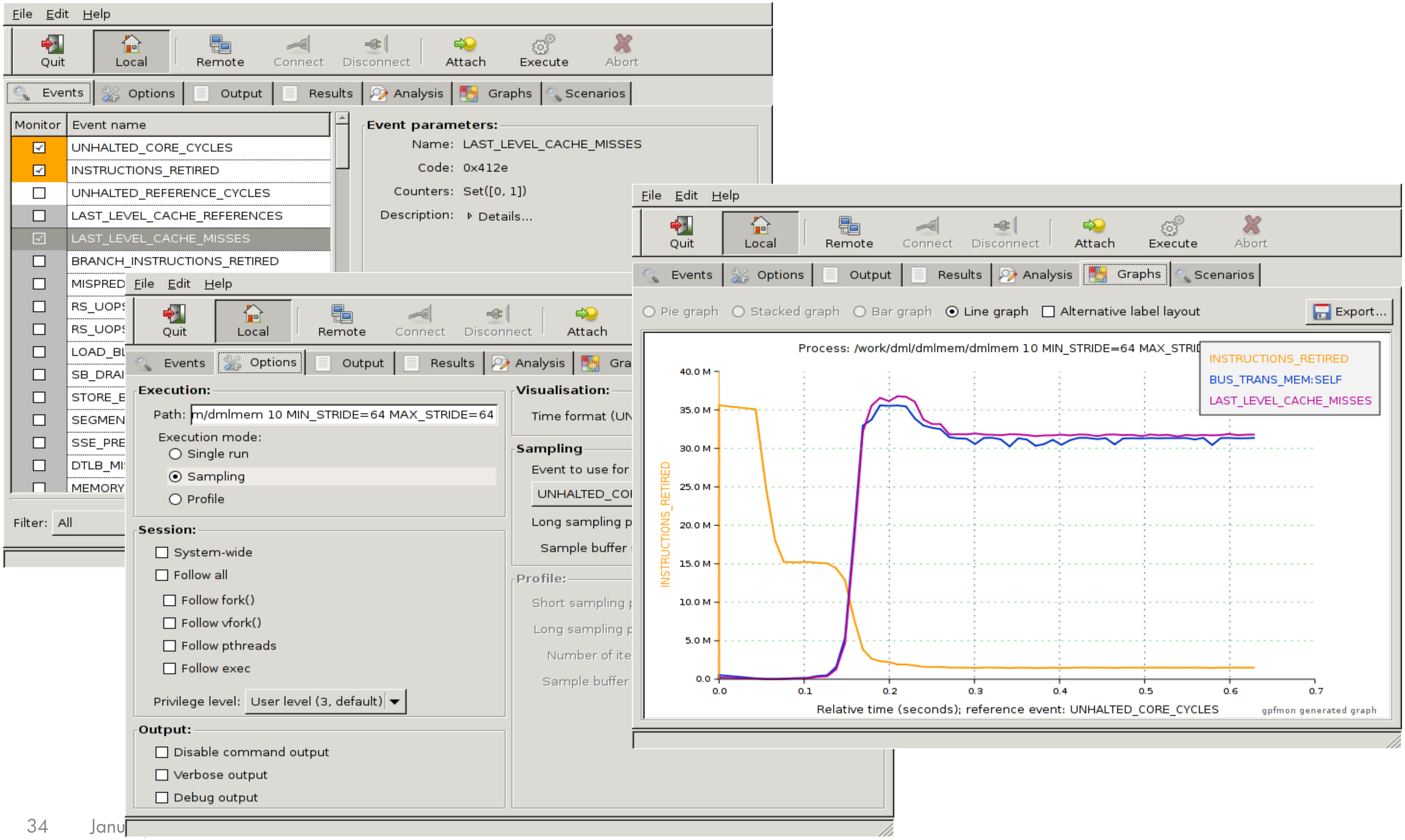


Process: is.B (6721) Thread: all Load Module: is.B Function: randlc

```
Function Disassembly                                  IP Samples
                                                      % Grand Totals
                                                      ☑
    0x02e0:0  M    nop.m            0                 1.93
          :1  F    fcvt.fx.trunc    f7=f9
          :2  I_   nop.i            0 ;;
    0x02f0:0  M_   getf.sig         r3=f7 ;;          4.36
          :1  M    nop.m            0
          :2  I_   sxt4             r2=r3 ;;
    0x0300:0  M    setf.sig         f15=r2            0.45
          :1  I    nop.i            0
          :2  I_   nop.i            0 ;;
    0x0310:0  M    nop.m            0                 2.91
          :1  F    fcvt.xf          f14=f15
```

# Tools(3)

- gpfmon (CERN): python-based GUI for pfmon



| Monitor | Event name |
|---|---|
| ☑ | UNHALTED_CORE_CYCLES |
| ☑ | INSTRUCTIONS_RETIRED |
| ☐ | UNHALTED_REFERENCE_CYCLES |
| ☐ | LAST_LEVEL_CACHE_REFERENCES |
| ☑ | LAST_LEVEL_CACHE_MISSES |
| ☐ | BRANCH_INSTRUCTIONS_RETIRED |
| ☐ | MISPRED |
| ☐ | RS_UOPS |
| ☐ | RS_UOPS |
| ☐ | LOAD_BL |
| ☐ | SB_DRAI |
| ☐ | STORE_E |
| ☐ | SEGMEN |
| ☐ | SSE_PRE |
| ☐ | DTLB_MI |
| ☐ | MEMORY |

Filter: All

**Event parameters:**

Name: LAST_LEVEL_CACHE_MISSES
Code: 0x412e
Counters: Set([0, 1])
Description: ▸ Details...

**Execution:**

Path: m/dmlmem 10 MIN_STRIDE=64 MAX_STRIDE=64

Execution mode:
○ Single run
● Sampling
○ Profile

**Session:**

☐ System-wide
☐ Follow all
☐ Follow fork()
☐ Follow vfork()
☐ Follow pthreads
☐ Follow exec

Privilege level: User level (3, default)

**Output:**

☐ Disable command output
☐ Verbose output
☐ Debug output

**Visualisation:**

Time format (UN

**Sampling**

Event to use for
UNHALTED_COR
Long sampling p
Sample buffer

**Profile:**

Short sampling p
Long sampling p
Number of ite
Sample buffer

○ Pie graph ○ Stacked graph ○ Bar graph ● Line graph ☐ Alternative label layout

# Current challenges

- Merge with mainline
- Sharing the PMU resource
  - between different subsystems: watchdog, OProfile, perfmon2
  - between conflicting users: per-thread and system-wide
  - mutual-exclusion is too restrictive, especially on large systems
  - workaround via affinity restriction is invalid
- PMU access in virtualized environments
  - PMU usage is never for correctness but for performance
  - usage model evolving: from development only to always on
  - used by monitoring, tools, managed runtimes, OS kernels
  - must ensure continuity of service
  - need guest PMU virtualization + system-wide monitoring

# Summary

- Performance monitoring is not optional anymore
- No multicore performance without monitoring!

- Perfmon2 is a very advanced monitoring interface
- Perfmon2 support all major hardware platforms
- Perfmon2 has an active community of users/dev
- Perfmon2 embraced by all HW vendors
- Perfmon2 to become the Linux monitoring interface

# References

- Perfmon2 web site/mailing lists:
  - visit http://perfmon2.sf.net
- Caliper web site
  - visit http://www.hp.com/go/caliper
- gpfmon (CERN):
  - https://andrzejn.web.cern.ch/andrzejn