Deep dive into Linux perf tool

Like Xu

Linux v5.14

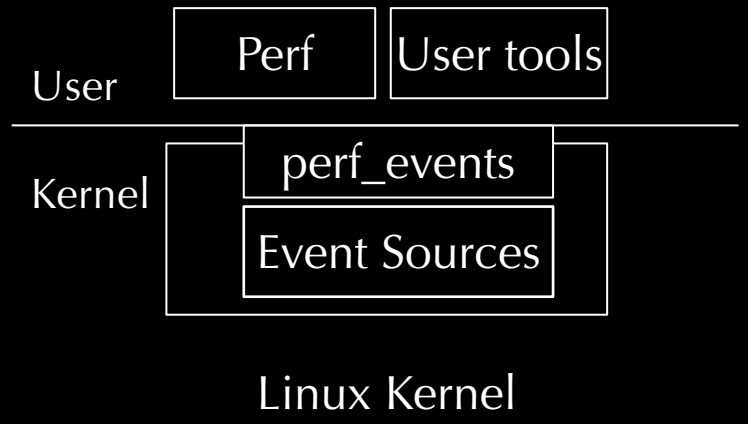
Agenda

1. Perf Overview
2. Perf Roadmap
3. Event types
4. Profiling modes
5. Hardware PMU support
6. From perf-config to perf-diff
7. Sample Data Fields
8. Perf Miscellaneous

Uncovered topics :

1. perf-probe/kvm/timechart
2. JIT Support
3. eBPF + LLVM
4. External scripts on samples
5. Any specific hw/sw events
   (e.g topdown or hyprid cpu)

Perf Overview



- A performance analysing tool for Linux,
  - Integrated into the Linux kernel
  - Available from Linux kernel version 2.6.31 in 2009
  - kernel driver (perf_event subsystem) + user-space perf tool
- Supported Architectures
  - x86, PowerPC, ARM, Sparc, MIPS, Alpha, Blackfin, sh
  - Aims to abstract the hardware and be easy to use
- Scope
  - system wide、per-numa、per-cpu、
  - perf-thread、per-container、per-dso(dynamic shared object (DSO))
- Target
  - hardware event + software event + trace-point + call-graph
- Profile modes
  - Count, Sample, Trace, Snapshot

Profiling Target

- --all-cpus (-a)
- --pid (-p)
- --tid (-t)
  - --no-inherit :: child tasks do not inherit counters.
  - --inherit :: perf-ftrace/record/top
- --uid (-u)
- --namespaces and --cgroup
- --comm
- --exclude-perf

Build Perf :: perf-version

- Keep it consistent with the kernel
  - make sure the lib installed correctly
  - cd linux/tools/perf
  - make LIBPFM4=1
- $ perf version --build-options
  - glibc
  - libelf
  - dwarf + dwarf_getlocations
    - Debugging With Arbitrary Record Formats
    - standardized debugging data format
  - syscall_table
  - libbfd
    - Binary File Descriptor library (BFD)
    - 25 instruction set architectures
  - libnuma + numa_num_possible_cpus
  - libperl + libpython
  - libslang
    - rapid alpha-numeric terminal
  - libcrypto :: buildid from md5
  - libunwind + libdw-dwarf-unwind
    - enable clang++ to port to platforms
  - zlib + lzma + zstd
    - perform lossless data compression
  - get_cpuid
  - bpf
  - aio
    - asynchronoes I/O library
  - libpfm4
    - help convert event name

perf-test :: Runs sanity tests

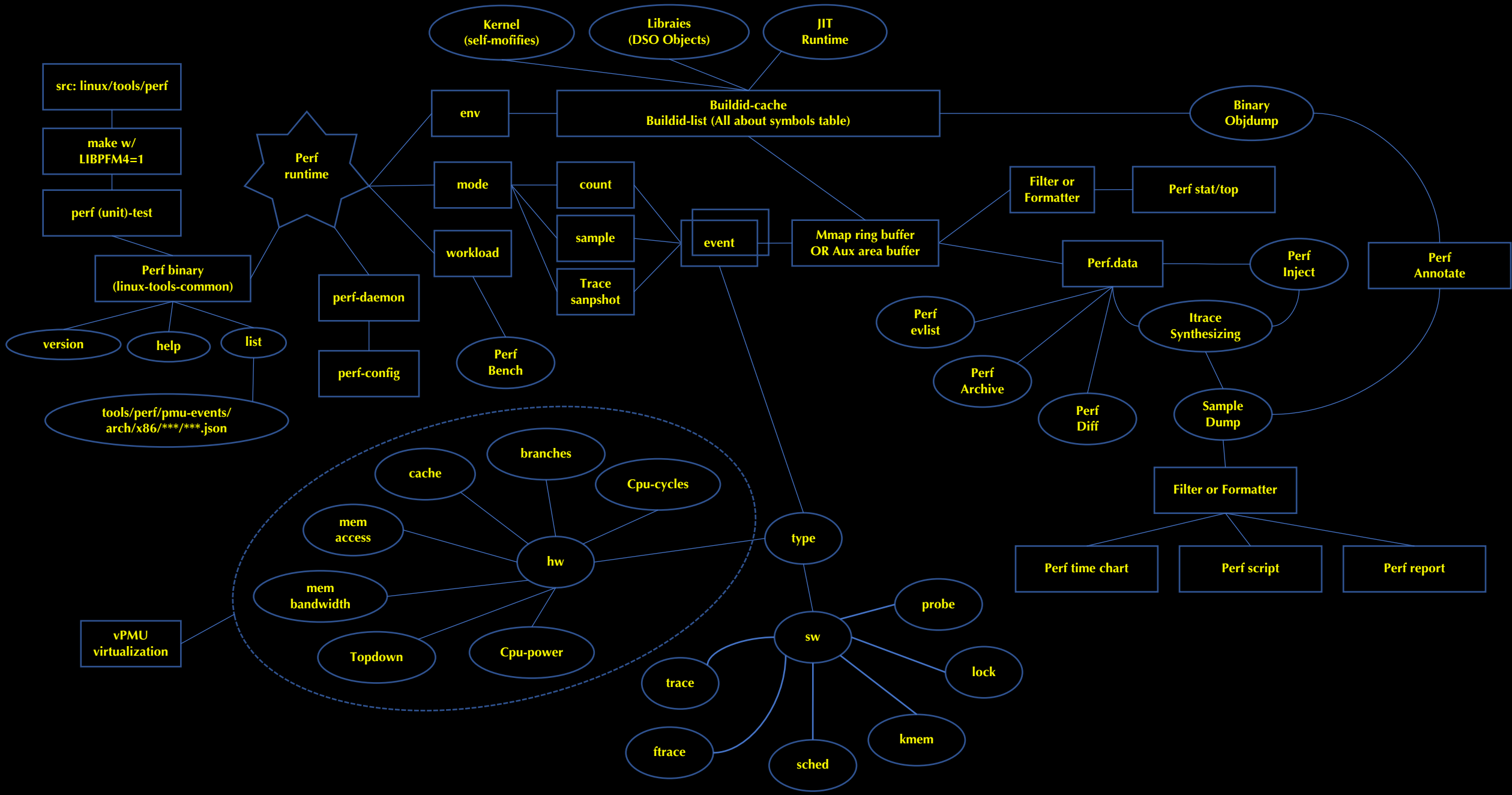
run all tests within single process:

$ perf test --skip 17,23,24,58 --dont-fork

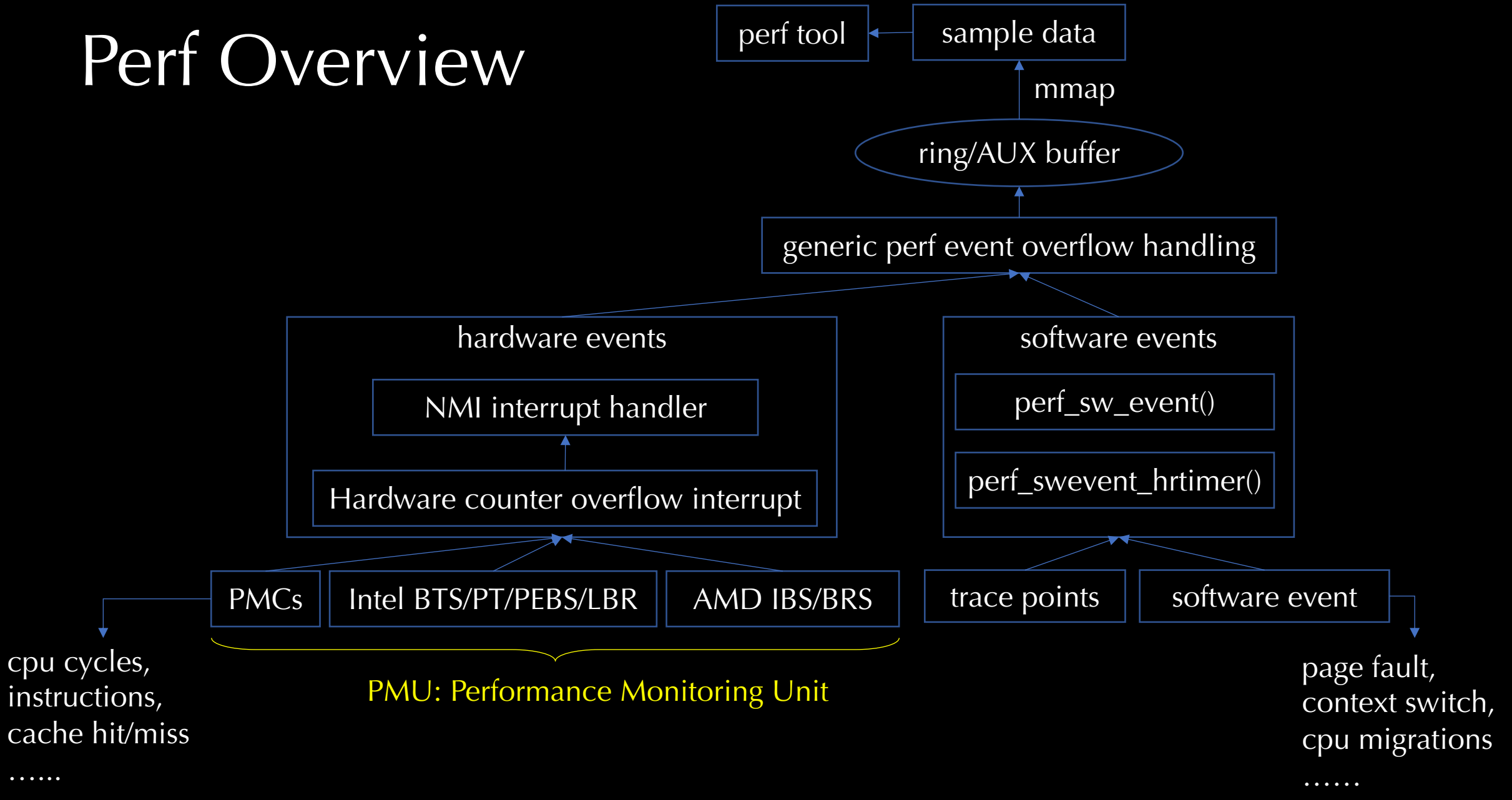
```
10: PMU events                                                      :
10.1: PMU event table sanity                                        : Ok
10.2: PMU event map aliases                                         : Ok
10.3: Parsing of PMU event table metrics                            : Ok
10.4: Parsing of PMU event table metrics with fake PMUs             : Ok
11: DSO data read                                                   : Ok
12: DSO data cache                                                  : Ok
13: DSO data reopen                                                 : Ok
14: Roundtrip evsel->name                                           : Ok
15: Parse sched tracepoints fields                                  : Ok
16: syscalls:sys_enter_openat event fields                          : Ok
17: Setup struct perf_event_attr                                    : Skip (user override)
```

perf builtin commands

```
The most commonly used perf commands are:
  annotate        Read perf.data (created by perf record) and display annotated code
  archive         Create archive with object files with build-ids found in perf.data file
  bench           General framework for benchmark suites
  buildid-cache   Manage build-id cache.
  buildid-list    List the buildids in a perf.data file
  c2c             Shared Data C2C/HITM Analyzer.
  config          Get and set variables in a configuration file.
  daemon          Run record sessions on background
  data            Data file related processing
  diff            Read perf.data files and display the differential profile
  evlist          List the event names in a perf.data file
  ftrace          simple wrapper for kernel's ftrace functionality
  inject          Filter to augment the events stream with additional information
  iostat          Show I/O performance metrics
  kallsyms        Searches running kernel for symbols
  kmem            Tool to trace/measure kernel memory properties
  kvm             Tool to trace/measure kvm guest os
  list            List all symbolic event types
  lock            Analyze lock events
  mem             Profile memory accesses
  record          Run a command and record its profile into perf.data
  report          Read perf.data (created by perf record) and display the profile
  sched           Tool to trace/measure scheduler properties (latencies)
  script          Read perf.data (created by perf record) and display trace output
  stat            Run a command and gather performance counter statistics
  test            Runs sanity tests.
  timechart       Tool to visualize total system behavior during a workload
  top             System profiling tool.
  version         display the version of perf binary
  probe           Define new dynamic tracepoints
  trace           strace inspired tool
```


Kernel
(self-mofifies)
Libraies
(DSO Objects)
JIT
Runtime
src: linux/tools/perf
make w/
LIBPFM4=1
perf (unit)-test
Perf binary
(linux-tools-common)
version
help
list
tools/perf/pmu-events/
arch/x86/***/***.json
Perf
runtime
perf-daemon
perf-config
env
Buildid-cache
Buildid-list (All about symbols table)
Binary
Objdump
mode
count
sample
Trace
sanpshot
workload
Perf
Bench
event
Mmap ring buffer
OR Aux area buffer
Filter or
Formatter
Perf stat/top
Perf.data
Perf
Inject
Perf
Annotate
Perf
evlist
Perf
Archive
Itrace
Synthesizing
Perf
Diff
Sample
Dump
Filter or Formatter
Perf time chart
Perf script
Perf report
branches
cache
Cpu-cycles
mem
access
hw
mem
bandwidth
Topdown
Cpu-power
vPMU
virtualization
type
sw
probe
lock
trace
ftrace
sched
kmem

Perf Overview



Profiling modes on events

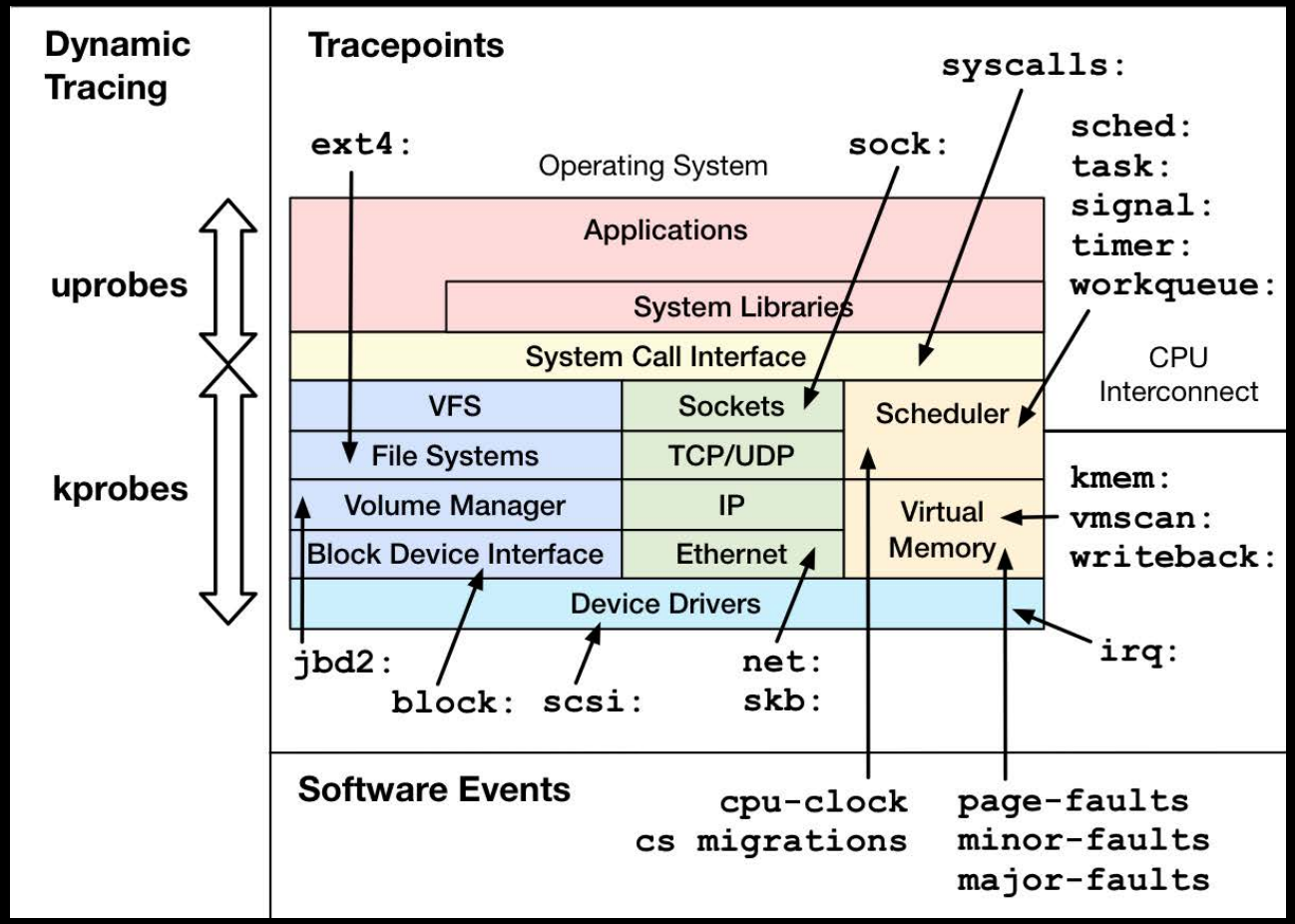
- Count mode --> $ perf stat --metrics IPC …
- Sample mode --> $ perf record …
  - based on the occurrence of a particular event such as a timer or interrupt
  - Record w/ -F, -c, -I
- Full-trace mode
  - Get all execution records
  - Software trace
    - $ perf trace/ftrace/probe/kmem/lock/sched/kvm …
  - Hardware trace (more on next slide)
    - Intel branch trace store/Processor Trace/LBR call-stack mode
- Snapshot mode :: let trace run and overwrite older data in the buffer
  - $ perf record –S
- Mixed mode (such as, count + sample)
  - $ perf stat record …

Sample Data Fields

Event types

- Hardware events (Kernel supplied PMU events)
  - Hardware cache event
  - Hardware event
  - metric, metricgroup
- Software event
  - System events
  - SDT event (Statically Defined Tracepoint events)
    - sdt_glib:mem__alloc
    - sdt_glib:mem__free
  - Tracepoint event
    - kvm:kvm_entry
- $ perf list --desc --long-desc --details (--raw-dump)

Kernel Space Events



$ perf list software

```
alignment-faults
bpf-output
cgroup-switches
context-switches OR cs
cpu-clock
cpu-migrations OR migrations
dummy
emulation-faults
major-faults
minor-faults
page-faults OR faults
task-clock
```

$ perf list tracepoint

- $ perf trace
- $ perf probe
- $ perf ftrace
- $ perf kmem
- $ perf lock
- $ perf sched
- $ perf kvm

perf-trace

Perf user need to access to /sys/kernel/debug/tracing

Or set kernel.perf_event_paranoid sysctl to -1

- add_events
- print syscall argument names
- Show syscall duration
- Show syscall start timestamp.
- Print syscall arguments that are equal to zero
- trace.tracepoint_beautifiers

  argument beautifiers, "libbeauty" as default

```
openat(dfd: CWD, filename: 0x9b2b9ee0)                                  = 6
read(fd: 6, buf: 0x7ff4c2e86010, count: 131072)                         = 0
```

```
1548.904 (499.097 ms): qemu-system-x8/2714250  ... [continued]: ppoll())
1548.847 (         ): CPU 57/KVM/2714313 ioctl(fd: 72<anon_inode:kvm-vcpu:5
 141.433 (2047.319 ms): CPU 8/KVM/2714264  ... [continued]: ioctl())
2048.014 (         ): qemu-system-x8/2714250 ppoll(ufds: 0x55b7a6965810, nf
2188.763 ( 0.011 ms): CPU 8/KVM/2714264 ioctl(fd: 23<anon_inode:kvm-vcpu:8>
2188.780 ( 0.009 ms): CPU 8/KVM/2714264 ioctl(fd: 23<anon_inode:kvm-vcpu:8>
```

```
438.220 ( 0.004 ms): sap1010/13888 rt_sigaction(
438.228 ( 0.017 ms): sap1010/13888 getuid()
438.249 ( 0.005 ms): sap1010/13888 migrate_page
438.259 ( 0.004 ms): sap1010/13888 rt_sigaction(
```

perf-ftrace

- $ perf ftrace --funcs (-F) # List available functions to trace
- --trace-funcs (-T) or --notrace-funcs (-N)
- --tracer (-t) 'function' or 'function_graph'.

```
  TIMESTAMP   FUNCTION
     |         |
583549.253981: finish_task_switch <-__schedule
583549.253982: rcu_read_lock_sched_held <-lock_acquire
583549.253982: rcu_read_lock_held_common <-rcu_read_lock_sched_held
583549.253982: rcu_lockdep_current_cpu_online <-rcu_read_lock_held_common
583549.253983: arch_irq_work_raise <-irq_work_queue
583549.253983: x2apic_send_IPI_self <-arch_irq_work_raise
583549.253984: __balance_callbacks <-finish_task_switch
583549.253984: do_balance_callbacks <-finish_task_switch
583549.253984: _raw_spin_unlock <-finish_task_switch
```

```
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
  66)               |  rcu_read_lock_sched_held() {
  66)               |    rcu_read_lock_held_common() {
  66)   0.138 us    |      rcu_lockdep_current_cpu_online();
  66)   0.730 us    |    }
  66)   2.405 us    |  }
  66)               |  switch_mm_irqs_off() {
  66)   0.251 us    |    load_new_mm_cr3();
  66)               |    rcu_read_lock_sched_held() {
  66)               |      rcu_read_lock_held_common() {
  66)   0.109 us    |        rcu_lockdep_current_cpu_online();
  66)   0.312 us    |      }
  66)   0.578 us    |    }
  66)   0.118 us    |    switch_ldt();
  66)   1.814 us    |  }
  66)               |  rcu_read_lock_sched_held() {
  66)               |    rcu_read_lock_held_common() {
  66)   0.117 us    |      rcu_lockdep_current_cpu_online();
  66)   0.370 us    |    }
  66)   0.800 us    |  }
```

perf-kmem

- a perf version of kmemtrace-user
  - alloc/free
  - Slab → need a a specific cache or is there significantly fragmented, memory leak
  - Page → check the amounts of Taking lock on zone->lock, disabling interrupts, dirtying cache line, pages are freed in batch, memory pressure
  - Per-CPU Allocator → contention on the zone->lock, imbalance between CPUs, large amounts of cache line bounces
  - Fragmentation → resize the memory pool, high-order allocations, the default hugepage size
- Record demo
  - $ perf kmem --verbose --caller --alloc --slab --page record -- perf bench mem memcpy
- Stat demo
  - perf kmem --caller --alloc --slab --page --live stat
    - Callsite or Alloc Ptr
    - Total_alloc/Per :: the total amount of memory allocated per call-site or alloc-ptr
    - Hit ::: if alloc_stat->ptr == ptr
    - Ping-pong :: statistics for allocations and frees on different cpus
    - Frag
    - Live alloc (KB) ::: shows live (currently allocated) pages
    - Order
    - GFP flags :: GFP_KERNEL, GFP_NOWAIT ….
    - PFN
    - SUMMARY (SLAB allocator)
    - SUMMARY (page allocator)

perf-lock - Analyze lock events

- $ perf lock record perf bench futex wake
- $ perf lock report --key=acquired

Name	acquired	contended	avg wait (ns)	total wait (ns)	max wait (ns)	min wait (ns)
&pmus_srcu	829451	0	0	0	0	0
rcu_read_lock	405316	0	0	0	0	0
&____s->seqcount...	46130	0	0	0	0	0
&____s->seqcount...	39930	0	0	0	0	0
event_mutex.wait...	11163	5027	2347	11803268	65806	1442
batched_entropy_...	9578	0	0	0	0	0
&obj_hash[i].loc...	7620	0	0	0	0	0
&obj_hash[i].loc...	7397	0	0	0	0	0
&obj_hash[i].loc...	7366	0	0	0	0	0
&n->list_lock	7306	2	1578	3157	1594	1563
&obj_hash[i].loc...	7161	0	0	0	0	0
&obj_hash[i].loc...	7110	0	0	0	0	0
&obj_hash[i].loc...	7018	0	0	0	0	0
&n->list_lock	6798	6	1651	9909	1795	1534

perf-kmem

- Tool to trace/measure kernel memory properties
- $ perf kmem --verbose --caller --alloc --slab --page record -- perf bench mem memcpy
- $ perf kmem --caller --alloc --slab --page --live stat

```
SUMMARY (page allocator)
========================
Total allocation requests
Total free requests

Total alloc+freed requests
Total alloc-only requests
Total free-only requests

Total allocation failures
```

Callsite	Total_alloc/Per	Total_req/Per	Hit	Ping-pong	Frag
proc_sys_call_handler+ce	8232/1372	4127/687	6	1	49.866%
__alloc_skb+7c	21504/1024	13248/630	21	21	38.393%
load_elf_phdrs+3a	10240/1024	6328/632	10	0	38.203%

PFN	Live alloc (KB)	Hits	Order	Mig.type	GFP flags	Callsite
4854404	16	1	2	MOVABLE	I|F|NWR|NR|C|NMA|RC	ffffffff021e3d40
69882200	8	1	1	UNMOVABL	I|F|NWR|NR|C|NMA	ffffffff021e3d40
68807760	8	1	1	UNMOVABL	KAC|Z	ffffffff021e3d40

perf-sched

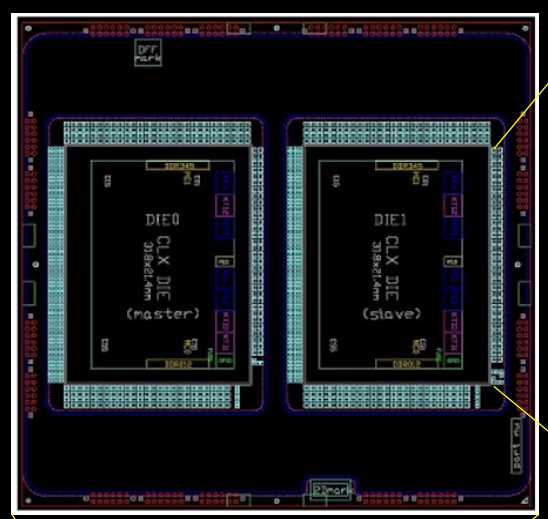
- Tool to trace/measure scheduler properties (latencies)
- $ perf sched record -- sleep 1
- $ perf sched timehist

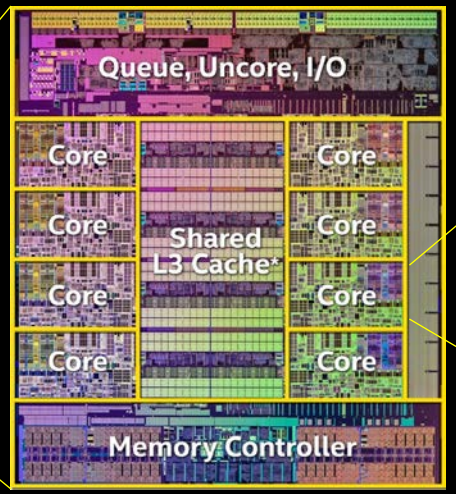
```
             time    cpu  task name                       wait time  sch delay   run time
                          [tid/pid]                          (msec)     (msec)     (msec)
--------------- ------  ------------------------------  ---------  ---------  ---------
  108804.197065 [0000]  perf[538387]                        0.000      0.000      0.000
  108804.197090 [0000]  migration/0[16]                     0.000      0.006      0.025
  108804.197151 [0001]  perf[538387]                        0.000      0.000      0.000
  108804.197177 [0001]  migration/1[19]                     0.000      0.006      0.025
  108804.197250 [0002]  perf[538387]                        0.000      0.000      0.000
  108804.197264 [0002]  migration/2[24]                     0.000      0.007      0.013
  108804.197345 [0003]  perf[538387]                        0.000      0.000      0.000
  108804.197359 [0003]  migration/3[31]                     0.000      0.006      0.013
  108804.197423 [0004]  perf[538387]                        0.000      0.000      0.000
  108804.197444 [0004]  migration/4[36]                     0.000      0.005      0.020
  108804.197514 [0005]  perf[538387]                        0.000      0.000      0.000
```

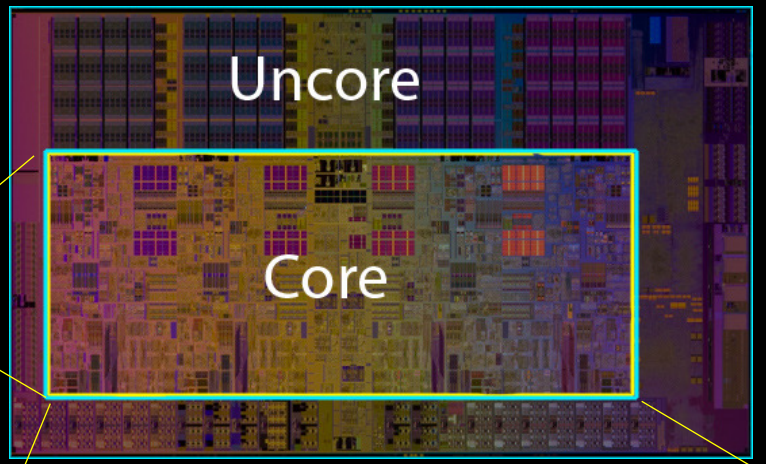
Event switch-on/off tracing mode

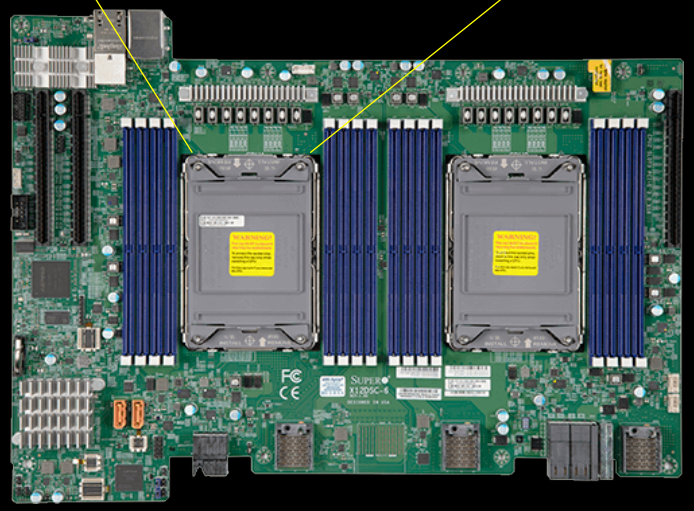
```
$ perf report/script/top/trace ...
--switch-on EVENT_NAME
--switch-off EVENT_NAME
--show-on-off-events
$ perf trace -e "sched:*,syscalls:*sleep*" \
--switch-on="syscalls:sys_enter_nanosleep" \
--switch-off="syscalls:sys_exit_nanosleep" --show-on-off sleep 1
```

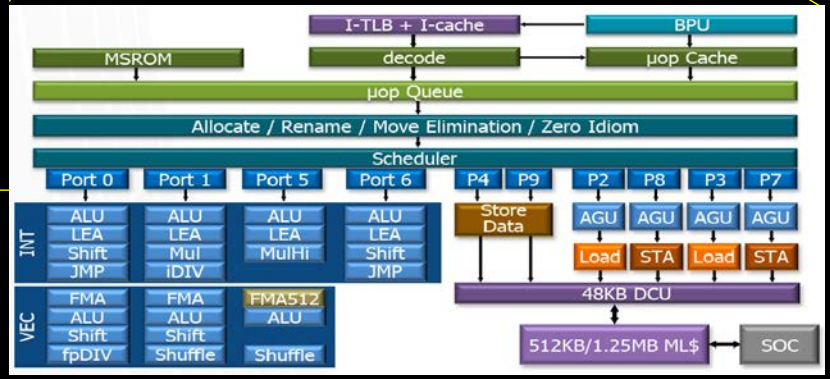
Hardware Events












```
branch-instructions OR branches
branch-misses
bus-cycles
cache-misses
cache-references
cpu-cycles OR cycles
instructions
ref-cycles
stalled-cycles-backend OR idle-cycles-backend
stalled-cycles-frontend OR idle-cycles-frontend
```

IPC

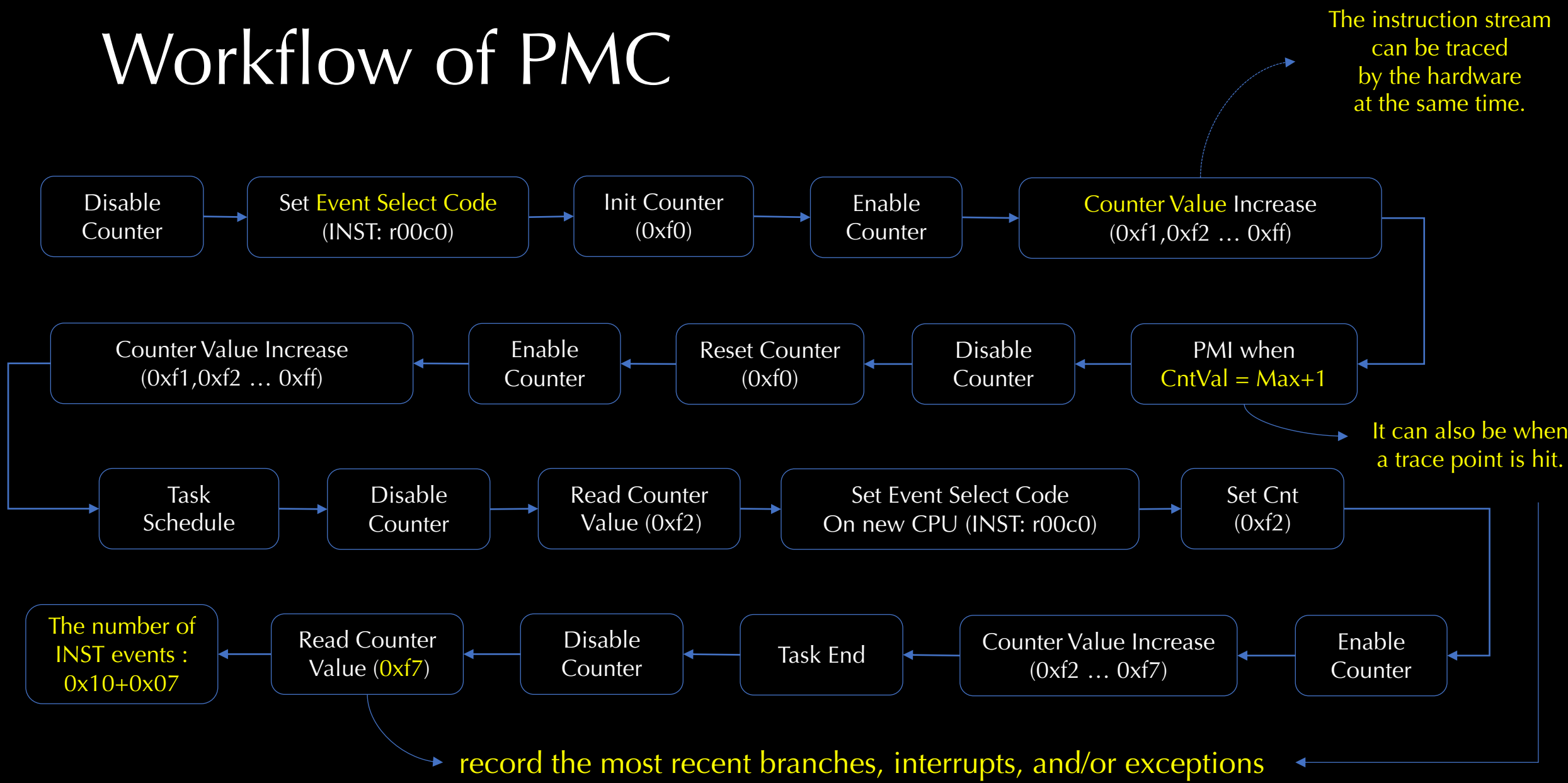
https://perfmon-events.intel.com/

Uncore hardware events

- Uncore events generally cannot be sampled,
  - but only counted globally with perf stat -a.
- E.g. memory read bandwidth every second
  - $ perf stat -C 0 -a -e uncore_imc_0/cas_count_read/ -I 1000
- $ perf iostat list → List all PCIe root ports
- $ perf iostat '0000:16,0000:20' -I 1000

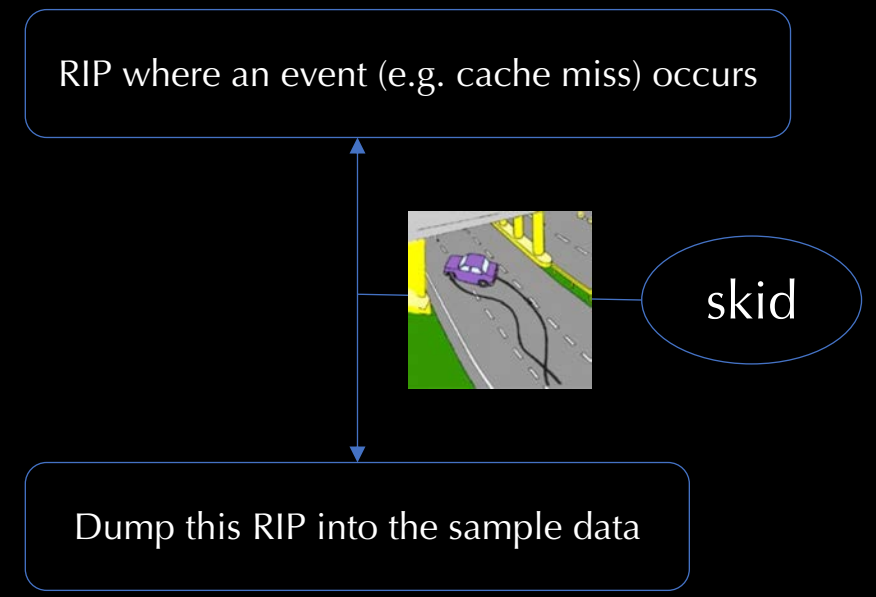
		I/O devices ←→ memory		I/O devices ←→ CPU	
# time	port	Inbound Read(MB)	Inbound Write(MB)	Outbound Read(MB)	Outbound Write(MB)
1.000502746	0000:16	1	0	1	0
1.000502746	0000:20	0	0	0	0
2.001499628	0000:16	1	0	1	0
2.001499628	0000:20	0	0	0	0

Workflow of PMC



PMC skid issue on instruction pointer (IP)

```
400400: push %rbp
400401: mov %rsp,%rbp
400404: push %rbx
400405: sub $0x8,%rsp
400409: cmpb $0x0,0x200420(%rip)
400410: jne 40045d
400412: mov $0x600648,%ebx
400417: mov 0x20041a(%rip),%rax
40041e: sub $0x600640,%rbx
400425: sar $0x3,%rbx
400429: sub $0x1,%rbx
40042d: cmp %rbx,%rax
400430: jae 400456
400432: nopw 0x0(%rax,%rax,1)
```



Avoid skid with Precise Event Based Sampling (PEBS)

- When a PMC is configured for PEBS,
  - a PEBS record is stored in the PEBS buffer in the DS save area after the counter overflow occurs.
    - the architectural state of the processor (state of the 8 general purpose registers, EIP register, and EFLAGS register)
- Precise level
  - no p - arbitrary skid
  - :p - constant skid
  - :pp - requested to have 0 skid (Intel PEBS events)
  - :ppp - must have 0 skid (only special case)
- $ perf top -e cache-misses:P
- $ perf record -g --call-graph lbr --group -e 'branch-instructions:P,branch-misses:P' -- ./workload

Intel Load latency & Precise store facility

- Load latency
  - Load data linear address
  - Latency value
    - For Intel, the latency includes any pipeline queueing delays
    - in addition to the memory subsystem latency.
  - Data source
- Precise store
  - Store data linear address
  - Store status
- list available events : $ perf mem/c2c record -e list
- $ perf mem record … # Profile memory accesses
  - $ perf mem --phys-data --data-page-size record --all-user --ldlat 10 -- ./workload
  - $ perf mem --data --phys-data --data-page-size report
- $ perf c2c record …# c2c: cache to cache – Detect False-Sharing cache-lines
  - Why memory load so slow? Not hit in LLC? Not hit in local memory? Cache-line false-sharing issue?

perf-c2c

- Wrapped by “-W, -d,--phys-data,--sample-cpu”
- $ perf c2c record -e list
- $ perf c2c record -- --ldlat=10 --call-graph lbr -a -- sleep 1
- $ perf c2c report --node-info --coalesce dso --full-symbols --display lcl --stdio --call-graph='callee'
  - Summary
    - Trace Event Information
    - Global Shared Cache Line Event Information
  - Shared Data Cache Line Table
    - "PA cnt” : the number of times we got different address than we currently hold’
  - Shared Cache Line Distribution Pareto

```
=================================================
          Shared Data Cache Line Table
=================================================
#
#        ---------- Cacheline ----------      Tot  ------- Load Hitm -------   Total    Total    Total  ---- Stores ----  ----- Core Load Hit -----  - LLC Load Hit --  - RMT Load Hit --  --- Load Dram ----
# Index            Address  Node  PA cnt     Hitm    Total  LclHitm  RmtHitm  records   Loads  Stores   L1Hit   L1Miss      FB      L1      L2  LclHitm  RmtHitm  RmtHit  RmtHitm      Lcl      Rmt
# .....  .................  ....  ......  .......  .......  .......  .......  .......  .......  .......  .......  .......  .......  .......  .......  .......  .......  .......  .......  .......  .......
#
      0  0xffffffff82e835c0     0      23    4.82%        8        4        4       35       35        0        0        0        7        3        0        8        4        0        4        0        9
      1  0xffffffff82f92c40     0      13    4.22%        7        7        0       16       16        0        0        0        4        4        0        1        7        0        0        0        0
      2  0xffffffff82f931c0     0      10    3.61%        6        4        2       35       35        0        0        0        3       14        1        9        4        0        2        0        2
      3  0xffffffff82f92700     0      14    3.01%        5        5        0       11       11        0        0        0        3        1        0        2        5        0        0        0        0
      4  0xffffffff82f95100     0      11    3.01%        5        5        0       13       13        0        0        0        1        5        0        2        5        0        0        0        0
      5  0xffffffff8313afc0     0      10    3.01%        5        4        1       18       18        0        0        0        6        6        0        0        4        0        1        0        1


=================================================
      Shared Cache Line Distribution Pareto
=================================================
#
#        ----- HITM -----  -- Store Refs --  --------- Data address ---------  ---------- cycles ----------    Total       cpu           Shared
#   Num  RmtHitm  LclHitm   L1 Hit  L1 Miss              Offset  Node  PA cnt  rmt hitm  lcl hitm     load  records       cnt           Object  Node{cpus %hitms %stores}
# .....  .......  .......  .......  .......  ..................  ....  ......  ........  ........  ........  .......  ........  .................  ....
#
  ------------------------------------------------------------
      0        4        7        0        0  0xffffffff83c63940
  ------------------------------------------------------------
        100.00%  100.00%    0.00%    0.00%                0x10     0       1       395       180         0       15       14  [kernel.vmlinux]   0{ 8  57.1%    n/a}  1{ 6  42.9%    n/a}
           |
           |--41.42%--_raw_spin_unlock
           |          event_function
           |          remote_function
           |          generic_exec_single
           |          smp_call_function_single
           |          event_function_call
           |          perf_event_for_each_child
           |          _perf_ioctl
           |          perf_ioctl
           |          __x64_sys_ioctl
           |          do_syscall_64
           |          entry_SYSCALL_64
           |          ioctl@plt
           |          perf_evsel__enable_cpu
           |          __evlist__enable
```

Advanced hardware tracing facilities

- Branch trace store (BTS)
  - trace all branches taken
  - $ perf record --per-thread -e intel_bts// ./workload
- Processor Trace (PT)
  - trace both cycle count and timestamp, branches, power, processor state
  - $ perf record -e intel_pt// ./workload
- The last branch record (LBR) call stack mode
  - trace function call instructions that do not return
  - Timed LBR :: the cycles of code block between 2 branches
  - $ perf record -g --call-graph lbr ./workload

Intel PT event config options

- $ perf record -e intel_pt/tsc=1,noretcomp=1,psb_period=5,mtc=1,mtc_period=9,cyc=1,cyc_thresh=12,pt=1,branch=1,ptw=0,fup_on_ptw=0,pwr_evt=0/ uname
- tsc (TSC packets using a TMA packet)
  - Produces TSC timestamp
- noretcomp :: Disables "return compression", more packets but more reliable.
- psb_period (0-5):: “`cat /sys/bus/event_source/devices/intel_pt/caps/psb_cyc` == 1”
  - The number of trace bytes between PSB packets
  - a TSC packet is produced with PSB
- mtc :: MTC timing packets, the hardware crystal clock (CTC)
- mtc_period (0,3,6,9)
- cyc (even finer grain timestamp) :: CPU cycles since the last CYC packet
- cyc_thresh (0-12)
- branch
- ptw :: A ptwrite instruction is executed.
- fup_on_ptw :: provides the address of the ptwrite
- pwr_evt ::power changes to the CPU C-state

Hardware PMU driver initialization

```
Performance Events: PEBS fmt4+-baseline,  AnyThread deprecated, Icelake events, 32-deep LBR, full-width counters, Intel PMU driver.
... version:                5
... bit width:              48
... generic registers:      8
... value mask:             0000ffffffffffff
... max period:             00007fffffffffff
... fixed-purpose events:   4
... event mask:             0001000f000000ff
```

```
# root @ amd-zen3 in ~ [17:18:59] C:130
$ lscpu | grep -ioE "pebs|bts|ibs"
ibs
```

```
# root @ intel-icx in /data [17:18:17]
$ lscpu | grep -ioE "pebs|bts|ibs"
pebs
bts
```

```
Performance Events: Fam17h+ core perfctr, AMD PMU driver.
... version:                0
... bit width:              48
... generic registers:      6
... value mask:             0000ffffffffffff
... max period:             00007fffffffffff
... fixed-purpose events:   0
... event mask:             000000000000003f
```

A typical usage

$ perf config …

$ echo 0 > /proc/sys/kernel/watchdog

$ echo 25 > /proc/sys/kernel/perf_cpu_time_max_percent

$ echo 10000 > /proc/sys/kernel/perf_event_max_sample_rate

$ echo 0 > /proc/sys/kernel/perf_cpu_time_max_percent

$ perf top -e …

$ perf (c2c/mem) record …

$ perf script --itrace=ibxwpe -F+flags …

$ perf report …

$ perf annotate …

perf-config

- Configuration for perf tools
  - colors、buildid、annotate、call-graph、report、record
- FILE
  - per-user :: $HOME/.perfconfig
  - system-wide :: $(sysconfdir)/perfconfig
- Example
  - $ perf config --user report.sort-order=srcline ui.show-headers=false kmem.default=slab
  - $ perf config call-graph.record-mode

```
[colors]
        top = red, default
        medium = green, default
        normal = lightgray, default
        selected = black, lightgray
        jump_arrows = blue, default
        addr = magenta, default
        root = white, blue

[annotate]
        disassembler_style = intel
        hide_src_code = true
        use_offset = true
        jump_arrows = true
        show_linenr = false
        show_nr_jumps = true
        show_total_period = false
        show_nr_samples = true
        offset_level = 2
        demangle = true
        demangle_kernel = true
[trace]
        args_alignment = 70
        no_inherit = true
        show_arg_names = true
        show_duration = true
        show_prefix = true
        add_events = sched:*
        show_timestamp = true
        show_zeros = true
        tracepoint_beautifiers = libtraceevent
```

perf buildid & buildid-cache

- $ perf buildid-list

f20952de346f580a702a77c51f3ab84ee92b75ce [kernel.kallsyms]

c2c416f4beec296b3f2cc0fdcfdc086cf84dbc71 /root/br_instr

9a5dbe7bce9ad9bb60f3e069b5fa9739499e1283 /usr/lib64/ld-2.28.so

4e546bd9980f7a46f0139716f1ba86d57aa87faa [vdso]

- --buildid-all :: Record build-id of all DSOs regardless whether it's actually hit or not.
- --buildid-dir (perf config buildid.dir)
  - Disable it, set buildid.dir to /dev/null
  - Default, in a per-user directory, $HOME/.debug/
    - Stores a hard link or copy
  - $ perf buildid-cache -v --kcore vm0/proc/kcore
- buildid-cache.debuginfod=URLs
  - Specify debuginfod URLs to be used
  - $ perf config buildid-cache.debuginfod=https://debuginfod.fedoraproject.org/
- perf config record.build-id
  - cache :: post-process data and save/update the binaries into the build-id cache
  - no-cache :: not update the build-id cache
  - skip :: skips post-processing and does not update the cache
  - mmap :: skips post-processing and reads build-ids from MMAP events

$ perf kallsyms -v vmx_exit # Searches running kernel for symbols

```
$ perf kallsyms -v vmx_exit
/proc/{kallsyms,modules} inconsistency while looking for "[__builtin__ftrace]" module!
vmx_exit: [kernel] [kernel.kallsyms] 0xffffffff8107fca0-0xffffffff8107fd60 (0xffffffff8107fca0-0xffffffff8107fd60)
```

How to program an event via perf

- Check event code $ perf list --details for each event
- $ perf record -e branch-misses …
- Raw encoding
  - $ perf stat -e r1a8 …
- Raw parameters (ls /sys/devices/cpu/format)
  - $ perf record -e cpu/event=0xa8,umask=0x1,name=LSD.UOPS_CYCLES,cmask=0x1/ …
  - $ perf stat -e cpu/event=0,umask=0x3,percore=1/
  - $ perf record -e intel_pt/tsc=1,noretcomp=1,psb_period=5/u …
- EVENT_MODIFIERS :: $ perf record -e branch-misses:IPDe …
  - u - user-space counting
  - k - kernel counting
  - h - hypervisor counting (ARM only)
  - G - guest counting (in KVM guests)
  - H - host counting (not in KVM guests)
  - I - non idle counting (exclude_idle)
  - P - use maximum detected precise level
    - p - precise level, reduce arbitrary skid
  - S - read sample value (PERF_SAMPLE_READ)
  - D - pin the event to the PMU
  - W - group is weak and will fallback to non-group if not schedulable
  - e - group or event are exclusive and do not share the PMU

```
itlb_misses.stlb_hit
     [Instruction fetch requests that miss the ITl
      cpu/(null)=0x186a3,umask=0x20,event=0x85/
itlb_misses.walk_active
     [Cycles when at least one PMH is busy with a
      cpu/cmask=0x1,(null)=0x186a3,umask=0x10,evel
itlb_misses.walk_completed
     [Code miss in all TLB levels causes a page wa
      cpu/(null)=0x186a3,umask=0xe,event=0x85/
itlb_misses.walk_completed_2m_4m
     [Code miss in all TLB levels causes a page wa
      cpu/(null)=0x186a3,umask=0x4,event=0x85/
itlb_misses.walk_completed_4k
     [Code miss in all TLB levels causes a page wa
      cpu/(null)=0x186a3,umask=0x2,event=0x85/
itlb_misses.walk_pending
     [Number of page walks outstanding for an outs
      cpu/(null)=0x186a3,umask=0x10,event=0x85/
```

Profile using event groups

- Ensure some events are always measured together as a group
  - to minimize multiplexing errors.
  - The number of available performance counters depend on the CPU.
- Leader sampling
  - $ perf record -e '{cycles,instructions}:S' ...
    - the first event (the leader) samples,
    - and it only reads the values of the other events in the group.
  - $ perf report –group
- Advanced (Vtuned) metric, $ perf list metric or metricgroup
  - $ perf stat -I 1000 -M Average_Frequency -C 0

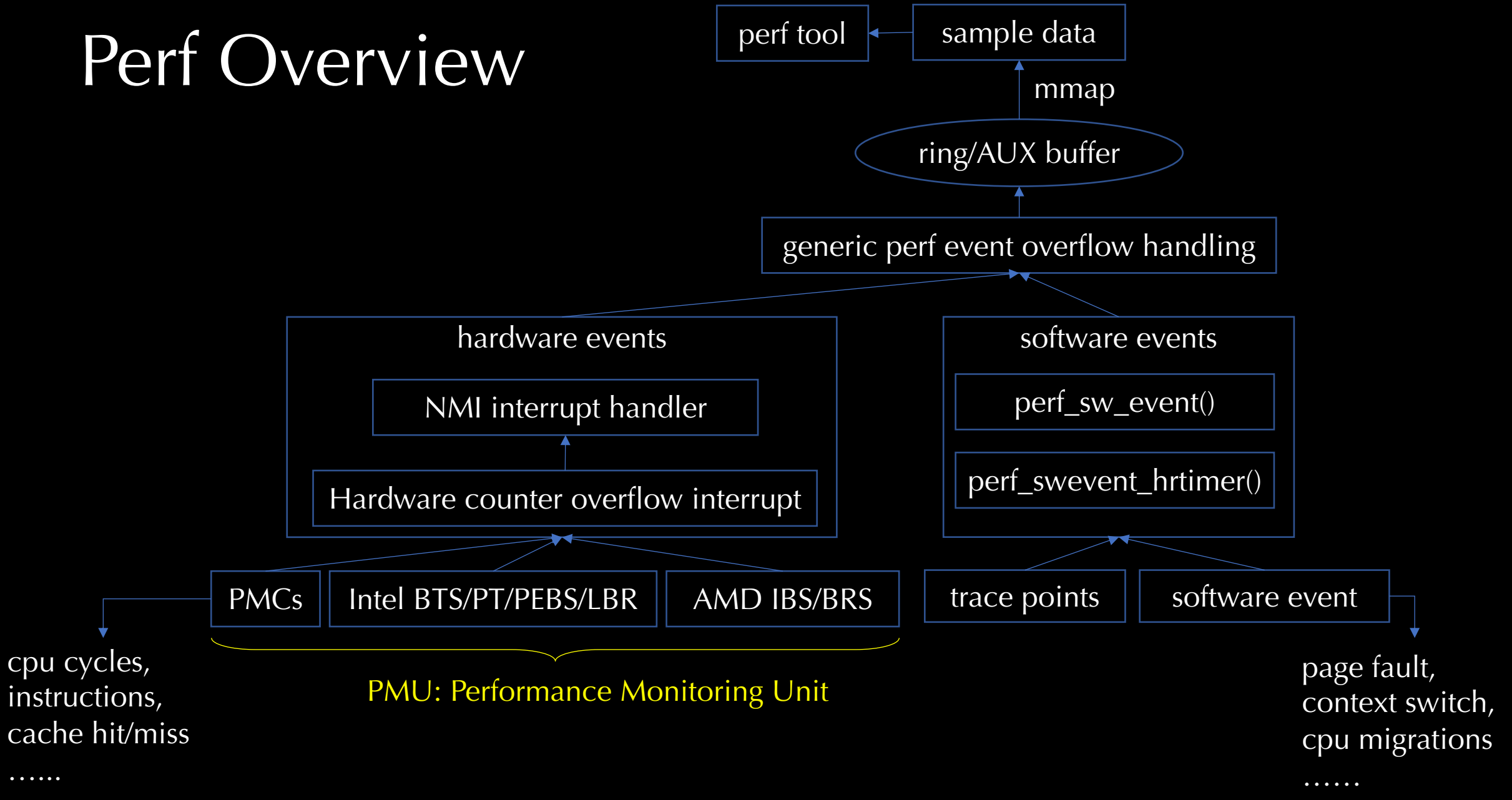
Little-known perf-record parameters

- --kcore
  - A fairly good kernel image is available in /proc/kcore but to get an accurate image a copy of /proc/kcore needs to be made under the same conditions as the data capture.
  - Save copies of /proc/kcore, /proc/kallsyms and /proc/modules
  - $ perf record --kcore uname
- --realtime (-r) :: Collect data with this RT SCHED_FIFO priority.
- --timestamp-boundary
  - Record timestamp boundary, time of first/last samples
- --delay
  - After starting the program, wait msecs before measuring. This is useful to filter out the startup phase of the program, which is often very different.
  - Also for perf trace/ftrace/stat/top

Enable call-graph recording
call chain/stack chain/backtrace

- --call-graph [option]
- Valid options:
  - "fp" (frame pointer, user space)
    - gcc --fomit-frame-pointer
  - "dwarf" (DWARF's CFI - Call Frame Information)
    - kernel space
      - CONFIG_UNWINDER_FRAME_POINTER (fp)
      - CONFIG_UNWINDER_ORC (orc)
    - also records (user) stack dump
      - Default size of the stack dump is 8192 (bytes).
    - perf + the libunwind or libdw library
  - "lbr" (Hardware Last Branch Record facility)
    - doesn't require any compiler options
    - doesn't work with branch stack sampling at the same time.
- $ perf record -g --call-graph dwarf,4096 -- workload

Perf Overview



Control mmap buffer & perf.data output

- Buffer options
  - --no-buffering
  - --mmap-pages=1M :: Number of mmap data pages
  - --mmap-flush=16M
  - --overwrite :: use an overwritable ring buffer
  - --buffer-size (-m, perf-ftrace only)
- Output options
  - --output
  - --max-size=1M
  - --timestamp-filename
  - --switch-output :: generate perf.data files on signal

Sample Data Fields

$ perf script -F comm,tid,pid,cpu,time,event,ip,dso,sym,symoff,srcline,period,insnlen,insn,misc

```
  CPU 115/KVM 466675/466796 [076] K     106883.295520:      35940   cycles:  ffffffff81e31d40 vmx_vmexit+0x0 (/lib/modules/5.15.0-rc2/build/vmlinux)
vmenter.S:79 ilen: 2 insn: eb 30

      branches:   tr strt
      branches:   call          IPC: 0.07 (124/1637)
      branches:   tr strt       IPC: 0.03 (17/475)
      branches:   tr strt

                                          ABI:2    AX:0xff    BX:0xff11003f772137e0
                                          ABI:2    AX:0xff    BX:0xff11003f772137e0
                                          ABI:2    AX:0xff    BX:0xff11003f772137e0
 lock_acquire
     rcu_read_lock_sched_held
         __fentry__                                    2M
         __mem_cgroup_uncharge_list                    1024M
             uncharge_page                             4K
                 __fentry__
                 release_pages                         ffffffff811f8267      10e5d0e9a
                     free_unref_page_list              ffffffff811d3003      3f77fedc08
                         __fentry__                    ffffffff811d3031      3f7f7ed18c
                         tlb_flush_mmu
                             tlb_flush_mmu     perf_event_exec+308:
                             tlb_finish_mmu    ffffffff81359254      insn: 48 8b 7c 24 20
                             __vma_adjust      ffffffff81359259      insn: e8 e2 fc ae 00          # PRED 9 cycles [260] 0.22 IPC
```

comm,tid,pid,cpu,time,event,ip,dso,sym,symoff,srcline,period,insnlen,insn,misc

flags ,ipc,callindent # -e intel_pt//, --itrace=cr -F +callindent

addr # sample address, --data

tod # --clockid CLOCK_MONOTONIC

iregs, # --intr-regs

uregs, # --user-regs

data_page_size, # -e "mem-loads,mem-stores" --data-page-size

code_page_size # -e "mem-loads,mem-stores" --code-page-size

phys_addr # -e "mem-loads,mem-stores" --phys-data

brstack # -b

brstacksym,brstackoff # -b

brstackinsn # -b

Dump event raw sample/trace

- perf-script/sched/report/lock/diff
  - --dump-raw-trace
  - --per-event-dump
- perf-mem
  - --dump-raw-samples

```
0x600 [0x78]: event: 1
.
. ... raw event: size 120 bytes
.  0000:  01 00 00 00 01 00 78 00 ff ff ff ff 00 00 00 00  ......x.<FF><FF><FF><FF>....
.  0010:  00 e0 02 c0 ff ff ff ff 00 70 00 00 00 00 00 00  .<E0>.<C0><FF><FF><FF><FF>.p
.  0020:  00 00 00 00 00 00 00 00 2f 6c 69 62 2f 6d 6f 64  ......../lib/mod
.  0030:  75 6c 65 73 2f 35 2e 31 34 2e 30 2d 72 63 32 2b  ules/5.14.0-rc2+
.  0040:  2f 6b 65 72 6e 65 6c 2f 6e 65 74 2f 69 70 76 34  /kernel/net/ipv4
.  0050:  2f 6e 65 74 66 69 6c 74 65 72 2f 69 70 5f 74 61  /netfilter/ip_ta
.  0060:  62 6c 65 73 2e 6b 6f 00 00 00 00 00 00 00 00 00  bles.ko.........
.  0070:  00 00 00 00 00 00 00 00                          ........

0 0x600 [0x78]: PERF_RECORD_MMAP -1/0: [0xffffffffc002e000(0x7000) @ 0]: x /lib/modules/
```

perf-script :: check all traces/samples

- --time:: (Only analyze samples within given time window:)
- --deltatime:: (Print time stamps relative to previous event.)
- --show-lost-events::
- --script=:: (Process trace data with the given script)
- --ns:: (Use 9 decimal places when displaying time)
- --deltatime:: (Print time stamps relative to previous event.)
- --call-trace::
- --call-ret-trace::
- --inline:: (If a callgraph address belongs to an inlined function)

Dump instruction traces with filter

- $ perf script --time starttime,stoptime --insn-trace --xed

```
$ perf script -i perf.data.kvm --guestkallsyms $KALLSYMS --insn-trace --xed -F+ipc,+flags | grep -C10 vmresume | head -21
      CPU 9/KVM  8614                      ffffffffa053a7ad __vmx_vcpu_run+0x3d ([kernel.kallsyms])      movq  0x48(%rax), %r9
      CPU 9/KVM  8614                      ffffffffa053a7b1 __vmx_vcpu_run+0x41 ([kernel.kallsyms])      movq  0x50(%rax), %r10
      CPU 9/KVM  8614                      ffffffffa053a7b5 __vmx_vcpu_run+0x45 ([kernel.kallsyms])      movq  0x58(%rax), %r11
      CPU 9/KVM  8614                      ffffffffa053a7b9 __vmx_vcpu_run+0x49 ([kernel.kallsyms])      movq  0x60(%rax), %r12
      CPU 9/KVM  8614                      ffffffffa053a7bd __vmx_vcpu_run+0x4d ([kernel.kallsyms])      movq  0x68(%rax), %r13
      CPU 9/KVM  8614                      ffffffffa053a7c1 __vmx_vcpu_run+0x51 ([kernel.kallsyms])      movq  0x70(%rax), %r14
      CPU 9/KVM  8614                      ffffffffa053a7c5 __vmx_vcpu_run+0x55 ([kernel.kallsyms])      movq  0x78(%rax), %r15
      CPU 9/KVM  8614                      ffffffffa053a7c9 __vmx_vcpu_run+0x59 ([kernel.kallsyms])      movq  (%rax), %rax
      CPU 9/KVM  8614   call               ffffffffa053a7cc __vmx_vcpu_run+0x5c ([kernel.kallsyms])      callq  0xffffffffa053a740
      CPU 9/KVM  8614   jcc                ffffffffa053a740 vmx_vmenter+0x0 ([kernel.kallsyms])          jz 0xffffffffa053a746
      CPU 9/KVM  8614   vmentry            ffffffffa053a742 vmx_vmenter+0x2 ([kernel.kallsyms])          vmresume     IPC: 0.09 (41/455)
          :8614  8614                      ffffffff81d754c9 cpu_idle_poll.isra.0+0x39 ([guest.kernel.kallsyms])   movq  (%rbx), %rax
          :8614  8614                      ffffffff81d754cc cpu_idle_poll.isra.0+0x3c ([guest.kernel.kallsyms])   test $0x8, %al
          :8614  8614   jcc                ffffffff81d754ce cpu_idle_poll.isra.0+0x3e ([guest.kernel.kallsyms])   jnz 0xffffffff81d754e3      IPC: 0.05 (3/51)
          :8614  8614                      ffffffff81d754d0 cpu_idle_poll.isra.0+0x40 ([guest.kernel.kallsyms])   movl  0x12d4bba(%rip), %eax
          :8614  8614                      ffffffff81d754d6 cpu_idle_poll.isra.0+0x46 ([guest.kernel.kallsyms])   test %eax, %eax
          :8614  8614   jcc                ffffffff81d754d8 cpu_idle_poll.isra.0+0x48 ([guest.kernel.kallsyms])   jnz 0xffffffff81d754c7
          :8614  8614                      ffffffff81d754c7 cpu_idle_poll.isra.0+0x37 ([guest.kernel.kallsyms])   pause
          :8614  8614                      ffffffff81d754c9 cpu_idle_poll.isra.0+0x39 ([guest.kernel.kallsyms])   movq  (%rbx), %rax
          :8614  8614                      ffffffff81d754cc cpu_idle_poll.isra.0+0x3c ([guest.kernel.kallsyms])   test $0x8, %al
          :8614  8614   jcc                ffffffff81d754ce cpu_idle_poll.isra.0+0x3e ([guest.kernel.kallsyms])   jnz 0xffffffff81d754e3
```

Synthesize samples from traces

- Sometime, samples can be synthesized after-the-fact, the sampling period can be selected for reporting.
- $ perf inject --itrace -i perf.data -o perf.data.new
- $ perf report --itrace=i1usge
  - e.g. sample every microsecond
- $ perf script --itrace=ibxwp
- itrace options
  - i synthesize instructions events
  - b synthesize branches events (branch misses for Arm SPE)
  - c synthesize branches events (calls only)
  - r synthesize branches events (returns only)
  - x synthesize transactions events
  - w synthesize ptwrite events
  - p synthesize power events (incl. PSB events for Intel PT)
  - o synthesize other events recorded due to the use
  - of aux-output (refer to perf record)
  - e synthesize error events
  - d create a debug log
  - f synthesize first level cache events
  - m synthesize last level cache events
  - M synthesize memory events
  - t synthesize TLB events
  - a synthesize remote access events
  - g synthesize a call chain (use with i or x)
  - G synthesize a call chain on existing event records
  - l synthesize last branch entries (use with i or x)
  - L synthesize last branch entries on existing event records
  - s skip initial number of events
  - q quicker (less detailed) decoding
  - Z prefer to ignore timestamps (so-called "timeless" decoding)

perf-annotate :: connect samples to code

- specify the foreground and background colors

- disassembler_style
  - Intel or amd64
  - see the '-M' option help in the 'objdump' man page.
- hide/show_src_code
- use_offset
  - From ffffffff816250b8: | mov 0x8(%r14),%rdi
  - to 368: | mov 0x8(%r14),%rdi
- show_nr_jumps
  - the number of branches jumping to that address

- jump_arrows
- show_total_period
  - total periods are printed instead of percent values as below.
- show_nr_samples
- --skip-missing
- --group
- --percent-type
  - global-hits, local-hits

```
Percent       mov    r13d,DWORD PTR [rdi+0x4]
        1  4f:   mov    edi,0xc5c8a0
         54: → call  pthread_mutex_lock@plt
                 sub    DWORD PTR [rip+0x7c0cd8],0x1
               ↓ je     18f
        1  66:   mov    edi,0xc5c7e0
                 mov    esi,0xc5c8a0
  0.03   70: → call  pthread_cond_wait@plt
                 mov    edi,0xc5c8a0
         7a: → call  pthread_mutex_unlock@plt
                 nop
  0.21  2  80:   mov    ecx,r14d
  0.41           mov    edx,0x1
  1.67           lea    rsi,[rbp-0x44]
  0.19           mov    edi,r13d
  0.22   8f: → call  epoll_wait@plt
  4.15           test   eax,eax
               ↓ js     130
  3.48           mov    ebx,DWORD PTR [rbp-0x40]
  1.19         ↓ jmp    b6
                 nop
  2.12  1  a8:   test   eax,eax
               ↓ jns    d3
  0.04   ac: → call  __errno_location@plt
  6.47           cmp    DWORD PTR [rax],0xb
 29.91           jne    d3
 28.05  1  b6:   mov    edx,0x8
  1.80           lea    rsi,[rbp-0x50]
  1.47           mov    edi,ebx
  4.63   c1: → call  read@plt
  2.81           movzx r15d,BYTE PTR [rip+0x7c0cc9]        # c5c8e7 <done>
  5.02           test   r15b,r15b
               ↑ je     a8
  0.12  2  d3:   cmp    BYTE PTR [rip+0x7c0ca0],0x0
  3.94         ↓ jne    170
  0.46  1  e0:   cmp    BYTE PTR [rip+0x7c0c92],0x0
  0.65         ↓ jne    150
  0.62  1  e9:   add    r12,0x1
  0.03           test   r15b,r15b
  0.00         ↑ je     80
  0.01           cmp    BYTE PTR [rip+0x7c0c7f],0x0
  0.01         ↓ jne    19e
```

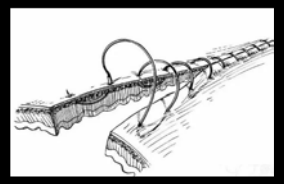
```
Samples           do {
                  ret = epoll_wait(efd, &ev, 1, to);
   244  49bbd0:   mov    ecx,r14d
   150  49bbd3:   mov    edx,0x1
  1822  49bbd8:   lea    rsi,[rbp-0x44]
   219  49bbdc:   mov    edi,r13d
   256  49bbdf: → call   435010 <epoll_wait@plt>
                  } while (ret < 0 && errno == EINTR);
  4135  49bbe4:   test   eax,eax
        49bbe6: ↓ js     49bc80 <workerfn+0x130>
                  if (ret < 0)
                  err(EXIT_FAILURE, "epoll_wait");

                  fd = ev.data.fd;
  3663  49bbec:   mov    ebx,DWORD PTR [rbp-0x40]
  1205  49bbef: ↓ jmp    49bc06 <workerfn+0xb6>
        49bbf1:   nop    DWORD PTR [rax+0x0]

                  do {
                  r = read(fd, &val, sizeof(val));
                  } while (!done && (r < 0 && errno == EAGAIN));
   416  49bbf8:   test   eax,eax
        49bbfa: ↓ jns    49bc23 <workerfn+0xd3>
     2  49bbfc: → call   436060 <__errno_location@plt>
   287  49bc01:   mp     DWORD PTR [rax],0xb
  1301  49bc04:   ne     49bc23 <workerfn+0xd3>
                  read():
                  return __read_chk (__fd, __buf, __nbytes, __bos0 (__buf));

                  if (__nbytes > __bos0 (__buf))
                  return __read_chk_warn (__fd, __buf, __nbytes, __bos0 (__buf));
                  }
                  return __read_alias (__fd, __buf, __nbytes);
  2226  49bc06:   mov    edx,0x8
  1994  49bc0b:   lea    rsi,[rbp-0x50]
    65  49bc0f:   mov    edi,ebx
   231  49bc11:   call   435650 <read@plt>
                  workerfn():
   728  49bc16:   movzx  r15d,BYTE PTR [rip+0x7c0cc9]        # c5c8e7 <done>
  3204  49bc1e:   test   r15b,r15b
        49bc21:   je     49bbf8 <workerfn+0xa8>

                  if (et) {
   131  49bc23:   mp     BYTE PTR [rip+0x7c0ca0],0x0        # c5c8ca <et>
  4025  49bc2a: ↓ jne    49bcc0 <workerfn+0x170>
                  ev.events = EPOLLIN | EPOLLET;
                  ret = epoll_ctl(efd, EPOLL_CTL_ADD, fd, &ev);
                  }
```

Display call chains

--call-graph=<print_type,threshold[,print_limit],order,sort_key[,branch],value>::

- print_type
  - $ perf report lfsr_cond -g fractal --stdio
- threshold[,print_limit],
  - a minimum percent to be output.
  - Default is 0.5 (%).
- order
  - callee: callee based call graph. == “--no-children”
  - caller: inverted caller based call graph. == “--children ”
- sort_key[,branch],
  - function, address, srcline
- value
  - percent, period, count
- perf report -g graph,percent --no-children --branch-history --stdio
- --stitch-lbr

```
43.30%  br_instr.c:50   [.] cmp_end                      br_instr
        |
        |--34.31%--br_random_cond_jump br_instr.c:50 (inlined)
        |          main br_instr.c:522
        |          __libc_start_main
        |          _start
        |
         --8.99%--br_random_cond_jump br_instr.c:50 (inlined)
                   main br_instr.c:522
                   __libc_start_main
                   _start
```



perf-diff :: display the differential profile

- --dsos
- --comms
- --symbols
- --baseline-only
  - Show only items with match in baseline.
- --formula
  - Show formula for given computation.

```
# Event 'cycles'
#
# Data files:
#  [0] perf.data (Baseline)
#  [1] perf.data.old
#  [2] perf.data.old.2
#
# Baseline/0  Delta Abs/1  Delta Abs/2  Shared Object      Symbol
# ..........  ...........  ...........  .................  ................................
#
     41.36%       +1.31%       +1.47%  br_instr           [.] cmp_end
     57.07%       -0.91%       -0.27%  br_instr           [.] lfsr_cond
      0.10%       +0.14%               [kernel.kallsyms]  [k] lock_is_held_type
                  +0.10%       +0.06%  [kernel.kallsyms]  [k] mark_lock
      0.15%       -0.10%       -0.14%  [kernel.kallsyms]  [k] lock_release
      0.10%       -0.10%       -0.10%  [kernel.kallsyms]  [k] native_write_msr
                  +0.10%               [kernel.kallsyms]  [k] native_apic_msr_eoi_write
                  +0.05%               [kernel.kallsyms]  [k] debug_check_no_obj_freed
      0.05%       +0.05%       +0.07%  [kernel.kallsyms]  [k] match_held_lock
      0.10%       -0.05%               [kernel.kallsyms]  [k] rcu_lockdep_current_cpu_online
                  +0.05%               [kernel.kallsyms]  [k] irqentry_enter
                  +0.05%               [kernel.kallsyms]  [k] is_module_text_address
                  +0.05%       +0.05%  [kernel.kallsyms]  [k] do_raw_spin_trylock
      0.10%       +0.05%       -0.07%  [kernel.kallsyms]  [k] __lock_acquire
                  +0.03%               [kernel.kallsyms]  [k] change_protection
      0.10%       +0.00%               [kernel.kallsyms]  [k] lock_acquired
      0.05%       -0.00%               [kernel.kallsyms]  [k] native_sched_clock
                  +0.00%               [kernel.kallsyms]  [k] rcu_read_lock_sched_held
      0.25%                            [kernel.kallsyms]  [k] __hrtimer_run_queues
      0.15%                            [kernel.kallsyms]  [k] native_irq_return_iret
      0.10%                            [kernel.kallsyms]  [k] __handle_mm_fault
      0.05%                            [kernel.kallsyms]  [k] lock_acquire
      0.05%                            [kernel.kallsyms]  [k] do_dentry_open
      0.05%                            [kernel.kallsyms]  [k] __pte_alloc
      0.05%                            [kernel.kallsyms]  [k] next_uptodate_page
      0.05%                            [kernel.kallsyms]  [k] __softirqentry_text_start
      0.05%                            [kernel.kallsyms]  [k] timerqueue_add
      0.02%                            [kernel.kallsyms]  [k] do_raw_spin_unlock
      0.01%                            [kernel.kallsyms]  [k] guarantee_online_cpus
      0.00%                            [kernel.kallsyms]  [k] native_flush_tlb_one_user
      0.00%                            [kernel.kallsyms]  [k] acpi_os_read_memory
                             +0.06%    [kernel.kallsyms]  [k] perf_iterate_ctx
                             +0.05%    [kernel.kallsyms]  [k] lockdep_hardirqs_on
                             +0.00%    [kernel.kallsyms]  [k] native_set_fixmap
```

perf-report

- --sort (-s) :: default sort order from "comm,dso,symbol"
- --percent-limit
  - not print overhead lower than this, Default is '0'.
- --group
  - show event group information together
- --skip-empty
  - If it's set true, 'perf report --stat' will not show 0 stats.
- $ perf report -F +overhead_sys,overhead_us,overhead_children --verbose --show-info --ns --percentage=relative --percent-type local-hits --disassembler-style=intel --show-cpu-utilization --skip-empty --hide-unresolved --branch-stack --branch-history --children --call-graph=fractal,callee,percent --show-ref-call-graph --inline --stitch-lbr --group --mem-mode --raw-trace --pretty=normal

perf-data


```
"timestamp": 244187521279000,
"pid": 1178672,
"tid": 1178672,
"comm": "perf-exec",
"callchain": [
        {
                "ip": "0xffffffff81077524",
                "symbol": "native_write_msr",
                "dso": "[kernel.vmlinux]"
        }
]
```


- JSON conversion
  - $ perf data convert --to-json output.json
- CTF conversion
  - The Common Trace Format (CTF) is a binary trace format designed to be very fast to write without compromising great flexibility. It allows traces to be natively generated by any C/C++ application or system, as well as by bare-metal (hardware) components.
  - $ perf data convert --all --tod --to-ctf=./ctf-data/
    - $ perf record --clockid CLOCK_MONOTONIC -e 'sched:*,raw_syscalls:*' -a -- sleep 1
  - babeltrace ./ctf-data/
    - a command-line tool which makes it very easy for mere mortals to view, convert, transform, and analyse traces.

perf-evlist

- List the event names in a perf.data file
- $ perf evlist --freq --trace-fields

```
$ perf evlist --freq --trace-fields -i perf.data.old
branch-misses:IPDe: sample_freq=4000, (not a tracepoint)
```

perf-archive

- Create archive with object files with build-ids found in perf.data file, analysis of perf.data contents can be possible on another machine.
- Demo

```
$ perf archive perf.data
Now please run:

$ tar xvf perf.data.tar.bz2 -C ~/.debug

wherever you need to run 'perf report' on.
```

```
$ tar xvf perf.data.tar.bz2 -C ~/.debug
.build-id/ff/0c0de44061a24029985d3b35856080556e865f
[kernel.kallsyms]/ff0c0de44061a24029985d3b35856080556e865f/
[kernel.kallsyms]/ff0c0de44061a24029985d3b35856080556e865f/kallsyms
[kernel.kallsyms]/ff0c0de44061a24029985d3b35856080556e865f/probes
.build-id/86/a266c2e46b80efab7cfde2626e630013e70798
usr/bin/perf/86a266c2e46b80efab7cfde2626e630013e70798/
usr/bin/perf/86a266c2e46b80efab7cfde2626e630013e70798/elf
usr/bin/perf/86a266c2e46b80efab7cfde2626e630013e70798/probes
.build-id/9a/5dbe7bce9ad9bb60f3e069b5fa9739499e1283
usr/lib64/ld-2.28.so/9a5dbe7bce9ad9bb60f3e069b5fa9739499e1283/
usr/lib64/ld-2.28.so/9a5dbe7bce9ad9bb60f3e069b5fa9739499e1283/elf
usr/lib64/ld-2.28.so/9a5dbe7bce9ad9bb60f3e069b5fa9739499e1283/probes
.build-id/95/5182d354056b626e6e387340fc69498f365ac6
usr/lib64/libcrypto.so.1.1.1g/955182d354056b626e6e387340fc69498f365ac6/
usr/lib64/libcrypto.so.1.1.1g/955182d354056b626e6e387340fc69498f365ac6/elf
usr/lib64/libcrypto.so.1.1.1g/955182d354056b626e6e387340fc69498f365ac6/probes
.build-id/c3/950777be21a85ee94da00b27b7824aa487993b
usr/lib64/libc-2.28.so/c3950777be21a85ee94da00b27b7824aa487993b/
usr/lib64/libc-2.28.so/c3950777be21a85ee94da00b27b7824aa487993b/elf
usr/lib64/libc-2.28.so/c3950777be21a85ee94da00b27b7824aa487993b/probes
.build-id/6e/157cb1e296136d37feaa4ab2bc1dde3d973c41
usr/lib64/libssh.so.4.8.5/6e157cb1e296136d37feaa4ab2bc1dde3d973c41/
usr/lib64/libssh.so.4.8.5/6e157cb1e296136d37feaa4ab2bc1dde3d973c41/elf
usr/lib64/libssh.so.4.8.5/6e157cb1e296136d37feaa4ab2bc1dde3d973c41/probes
```

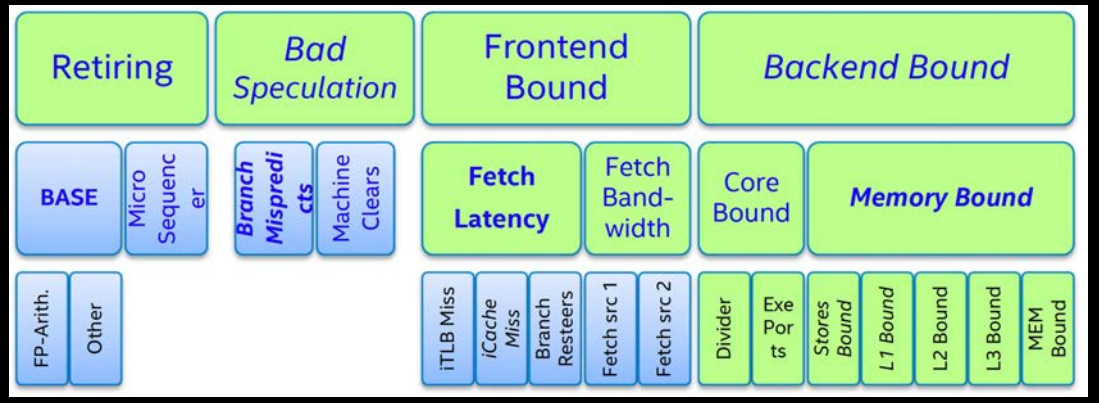
Profiling modes on events

- Count mode --> $ perf stat --metrics IPC …
- Sample mode --> $ perf record …
  - based on the occurrence of a particular event such as a timer or interrupt
  - Record w/ -F, -c, -I
- Full-trace mode
  - Get all execution records
  - Software trace
    - $ perf trace/ftrace/probe/kmem/lock/sched/kvm …
  - Hardware trace (more on next slide)
    - Intel branch trace store/Processor Trace/LBR call-stack mode
- Snapshot mode :: let trace run and overwrite older data in the buffer
  - $ perf record –S
- Mixed mode (such as, count + sample)
  - $ perf stat record …

Sample Data Fields

Count mode :: perf-stat

- $ perf stat --interval-clear -I 1000 --metrics IPC -C 0
- --metrics
- --per-socket/die/core/thread/node
- --repeat=<n> :: repeat command and print average
- --null :: Don't start any counters.
- --transaction (-T) :: Print statistics of transactional execution if supported.
- --topdown and --td-level
  - $ perf stat --topdown --td-level=2 --no-metric-only
- --smi-cost :: Measure SMI cost
- --interval-clear (Clear the screen before next interval.)
- --interval-print msecs (Print count deltas every N milliseconds)
- --interval-count times (Print count deltas for fixed number of times.)
- --summary :: Print summary for interval mode (-I)



Snapshot Mode

- let the trace run and overwrite older data in the buffer
  - grab a snapshot of what was going on around that interesting moment.
- Snapshot mode (intel_pt PMU) and sample trace
  - cannot be used together
- The default snapshot size is the auxtrace mmap size
  - -S0x100000, 4MiB for privileged users, 128KiB for unprivileged users.
- $ perf record -v -e intel_pt//u -S ./loopy 1000000000 &
- [1] 11435
- $ kill -USR2 11435 # Recording AUX area tracing snapshot

Perf top

- # Sample CPUs at 49 Hertz, and show top addresses and symbols, live (no perf.data file):
  - $ perf top -F 49
- # Sample CPUs at 49 Hertz, and show top process names and segments, live:
  - $ perf top -e cache-misses -F 49 -ns comm,dso
  - $ perf top -e raw_syscalls:sys_enter -ns comm
- $ perf top -e 'cache-misses,cycles' -v --force -F 49 --realtime=1 -u root -s overhead,comm --call-graph lbr --no-children --percentage relative --show-nr-samples

perf-daemon :: run record on background

- Standard configuration :: $ perf config --list
  - $ perf record --mmap-pages 20M -e sched:* --overwrite --switch-output -a
- START :: $ perf daemon start (--foreground)
  - Not every 'perf record' session is suitable for running under daemon.
  - produces data on query or periodically
- Check sessions :: $perf daemon -v
  - output: /opt/perfdata/session-sched/output # output for specific session.
    - tail -2 /opt/perfdata/session-cycles/output
  - control: /opt/perfdata/session-sched/control # perf control files.
  - up: 1 minutes # minutes daemon/session is running.
- Verify session is up and operational :: $ perf daemon ping
- Generate perf.data file
  - $ perf daemon signal
  - $ perf daemon signal --session cycles
  - $ tail -2 /tmp/perfdata/session-cycles/output
- STOP :: $ perf daemon stop

Tips to reduce perf record overhead

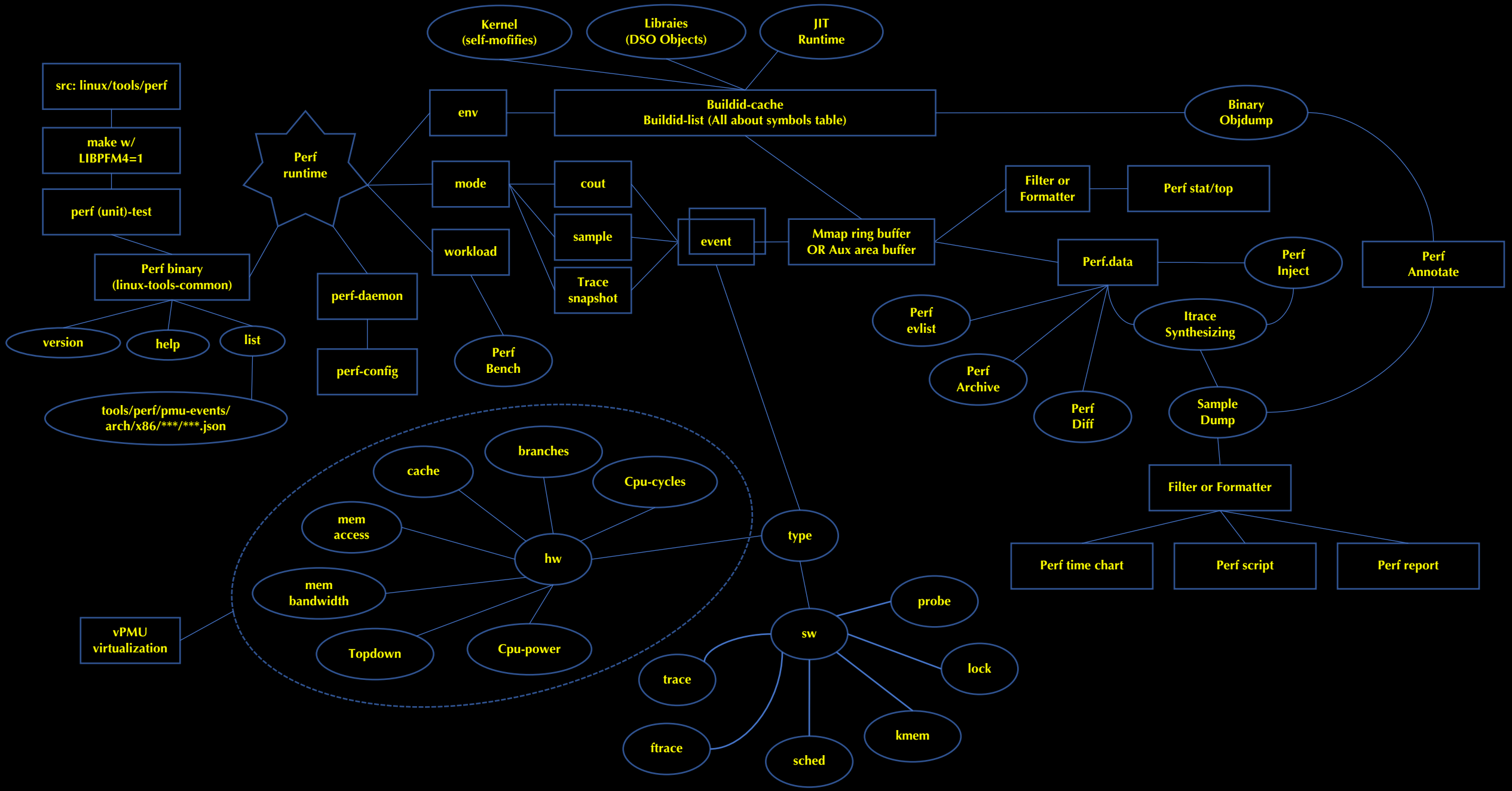
- Multi AIO trace writing allows caching more kernel data into userspace memory postponing trace writing for the sake of overall profiling data thruput increase.
  - --aio[=n], (default: 1, max: 4)
- Set affinity mask of trace reading thread
  - --affinity=node
- Produce compressed trace
  - --compression-level[=n]
    - (default: 1 - fastest compression, 22 - smallest trace)
- --no-buildid-cache, --no-buildid, --proc-map-timeout

perf-bench :: framework for benchmark

- Demo
  - $ perf bench --repeat 3 --format='simple' mem memcpy --function x86-64-movsq --size 4MB --nr_loops 10 --cycles
- sched: Scheduler and IPC benchmarks
- syscall: System call benchmarks
- mem: Memory access benchmarks
- numa: NUMA scheduling and MM benchmarks
- futex: Futex stressing benchmarks
- epoll: Epoll stressing benchmarks
- internals: Perf-internals benchmarks

So, why we need guest PMU ?

- Add all perf capabilities for cloud developers
- The profiling data for same workload are different over guest/host
  - due to vm-exit/entry world switch and some secret hardware details
- Allow users on cloud and users on host to use PMU hw resources in a harmony way
  - reduce virtualization overhead
  - keep both performance profiling data as accurate as possible
  - reduces security risks and side channel attack surface


Kernel
(self-mofifies)
Libraies
(DSO Objects)
JIT
Runtime
src: linux/tools/perf
make w/
LIBPFM4=1
perf (unit)-test
Perf binary
(linux-tools-common)
version
help
list
tools/perf/pmu-events/
arch/x86/***/***.json
Perf
runtime
perf-daemon
perf-config
env
Buildid-cache
Buildid-list (All about symbols table)
Binary
Objdump
mode
cout
workload
sample
Trace
snapshot
Perf
Bench
event
Mmap ring buffer
OR Aux area buffer
Filter or
Formatter
Perf stat/top
Perf.data
Perf
Inject
Perf
Annotate
Perf
evlist
Itrace
Synthesizing
Perf
Archive
Perf
Diff
Sample
Dump
Filter or Formatter
Perf time chart
Perf script
Perf report
branches
cache
Cpu-cycles
mem
access
hw
mem
bandwidth
Topdown
Cpu-power
vPMU
virtualization
type
probe
sw
lock
trace
kmem
ftrace
sched

Q & A

Thank you to join me.

Like Xu