Efficient Performance Profiling for Virtual Machines

Sandipan Das

AMD
together we advance_

Motivation

Performance profiling inside Virtual Machines (VMs) is no longer an afterthought

- Due to a high degree of cloud adoption

Inexpensive and accurate profiling are highly desirable

- Creates opportunities for both cloud vendors and users to optimize
  - Provisioning
  - Quality-of-service (QoS)
  - Capacity planning
  - Billing and costs

Performance Profiling

Running perf on a typical Linux system

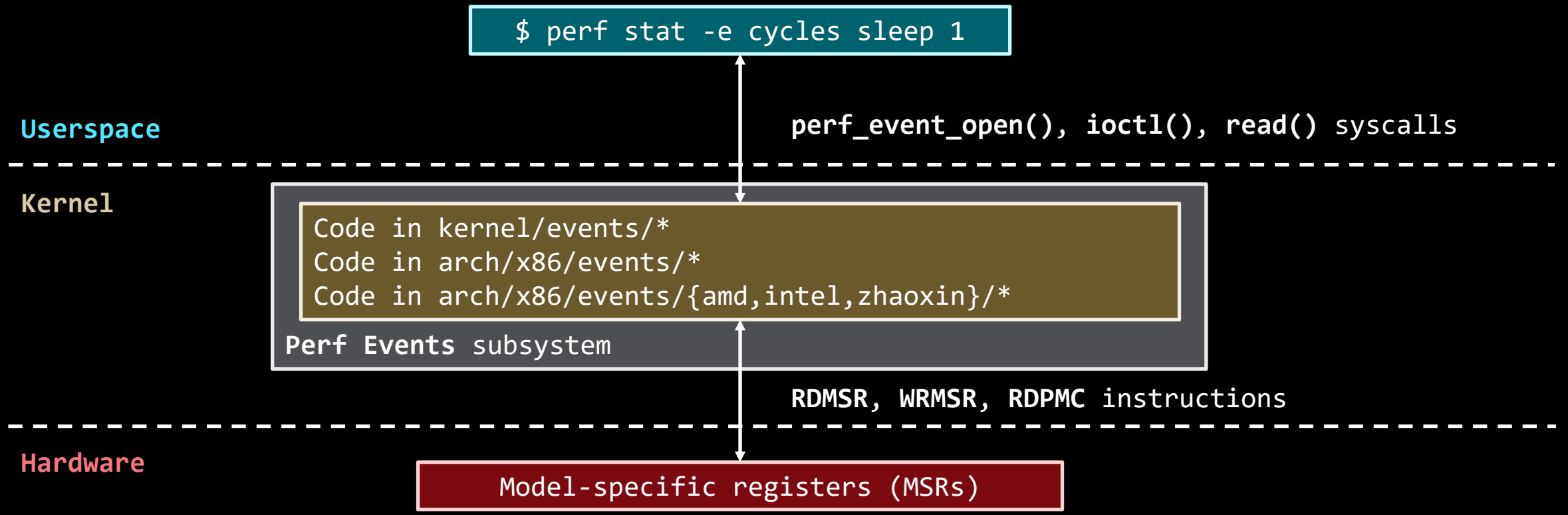
```
$ perf stat -e cycles sleep 1

 Performance counter stats for 'sleep 1':

         2,082,889      cycles:u

       1.002348165 seconds time elapsed

       0.000000000 seconds user
       0.002138000 seconds sys
```

Workload to be profiled

Event to be counted

Performance Profiling

Running perf on a typical Linux system



Performance Profiling

Dealing with multiple `perf` events

```
$ perf stat -e cpu-cycles,ref-cycles,branch-instructions,branch-misses,cache-misses sleep 1
```

Need for an event scheduling mechanism

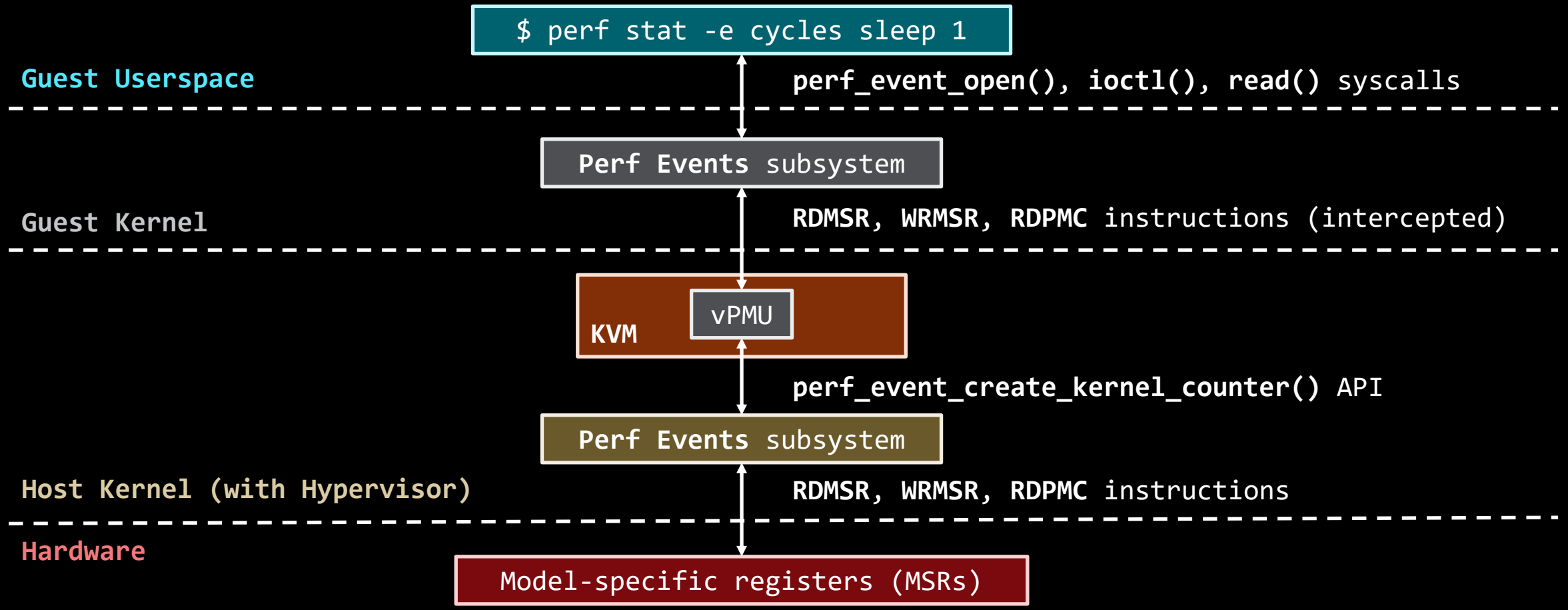
- The number of available hardware counters is limited, so they need to be rationed
- Some events may be programmable only on a specific subset of counters (events with constraints)
- Some events may be programmable only alongside other events (events that are grouped)
- Some events are tied to a process or thread (events that are task-specific)

The **Perf Events** subsystem has an event scheduler which

- Manages assignment of hardware counters to events and time-tracking
- Handles switching of event groups (time-based multiplexing)
- Switching of task-specific events during context switches (through scheduler hooks)

Performance Profiling – Inside Guests

Running perf inside a KVM guest



Performance Profiling – Inside Guests

Running `perf` inside a KVM guest

Guests do not have direct access to hardware

- MSR accesses are intercepted by KVM
- High overhead of world switches (transitions between host and guest contexts) due to interceptions

Backing events are created on behalf of the guest

- They exist alongside other host perf events (including the event used for the NMI watchdog)
- They may be silently disabled by the host due to unavailability of free hardware counters
- Guests have no control over how the host Perf Events subsystem is handling the backing events
- Guests do not know if their backing events are even getting hardware counters allotted

Performance Profiling – Conflicts between Host and Guest

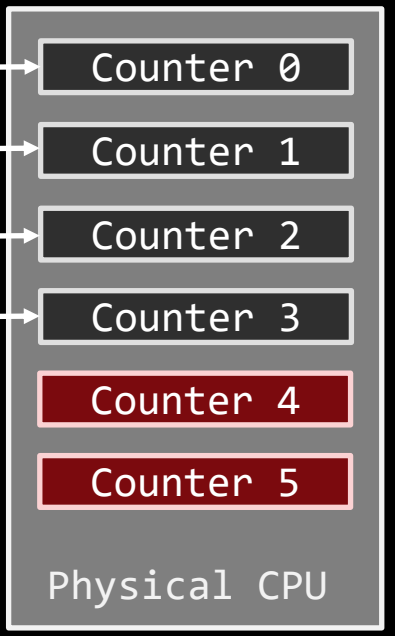
Best Case – Running `perf` inside a KVM guest while `perf` is not running on the host

```
Guest instance
$ perf stat -e cycles,instructions,branches,cache-misses sleep 1


 Performance counter stats for 'sleep 1':

          377,066      cycles:u
          311,251      instructions:u
           69,108      branches:u
            6,206      cache-misses:u

      1.010779420 seconds time elapsed

      0.002668000 seconds user
      0.008005000 seconds sys
```

Performance Profiling – Conflicts between Host and Guest

Best Case – Running `perf` inside a KVM guest while `perf` is not running on the host

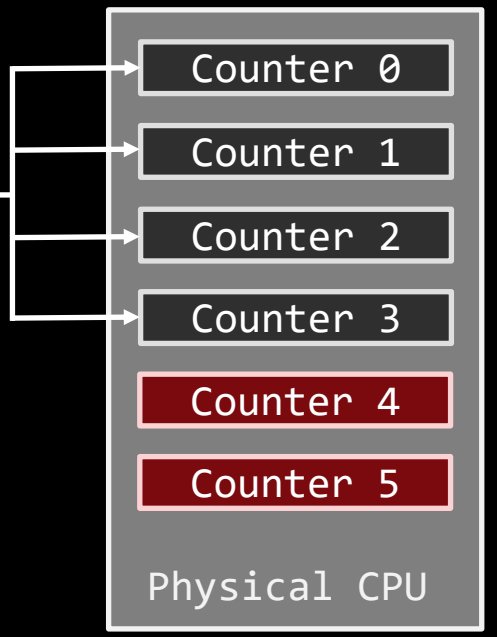
```
Guest instance
$ perf stat -e cycles,instructions,branches,cache-misses sleep 1

 Performance counter stats for 'sleep 1':

          377,066      cycles:u
          311,251      instructions:u
           69,108      branches:u
            6,206      cache-misses:u

      1.010779420 seconds time elapsed

      0.002668000 seconds user
      0.008005000 seconds sys
```



Performance Profiling – Conflicts between Host and Guest

Worst Case – Running `perf` inside a KVM guest while `perf` is also running on the host

```
Host instance
$ sudo perf stat -e cycles:D,instructions:D,branches:D,cache-misses:D -a
```

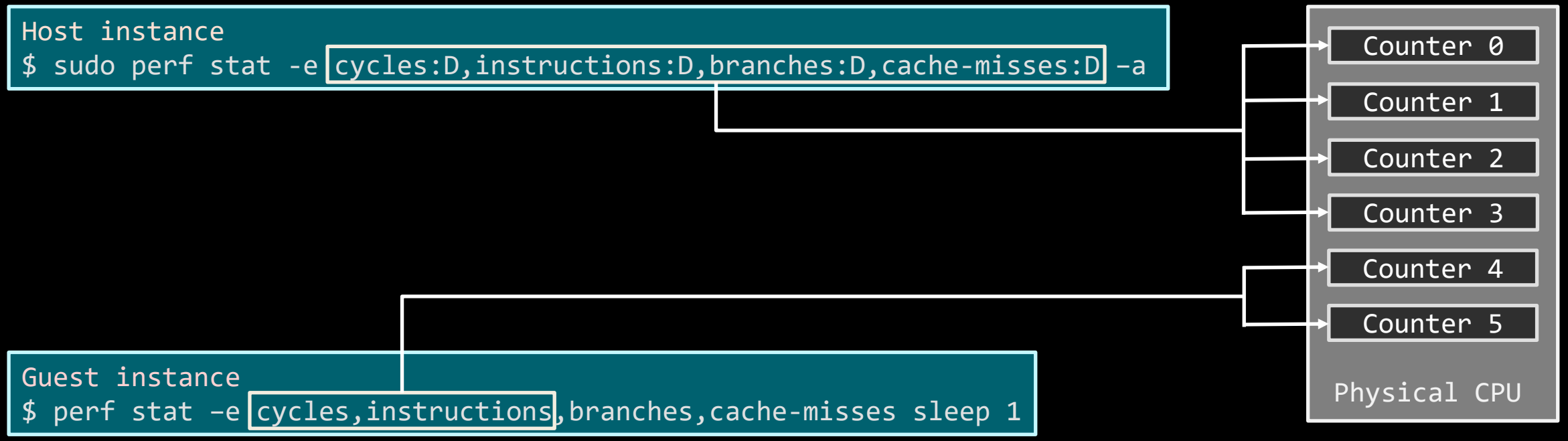


Pinning ensures that host events get permanent counter allotments

- No longer subject to time-based multiplexing
- Backing events for guests can no longer use these counters

Performance Profiling – Conflicts between Host and Guest

Worst Case – Running `perf` inside a KVM guest while `perf` is also running on the host



Performance Profiling – Conflicts between Host and Guest

Worst Case – Running `perf` inside a KVM guest while `perf` is also running on the host

```
Guest instance
$ perf stat -e cycles,instructions,branches,cache-misses sleep 1


 Performance counter stats for 'sleep 1':

          375,834      cycles:u
          311,372      instructions:u
                0      branches:u
                0      cache-misses:u

       1.010497211 seconds time elapsed

       0.000000000 seconds user
       0.010376000 seconds sys
```

vs. Best Case

```
377,066      cycles:u
311,251      instructions:u
 69,108      branches:u
  6,206      cache-misses:u
```

Performance Profiling – Conflicts between Host and Guest

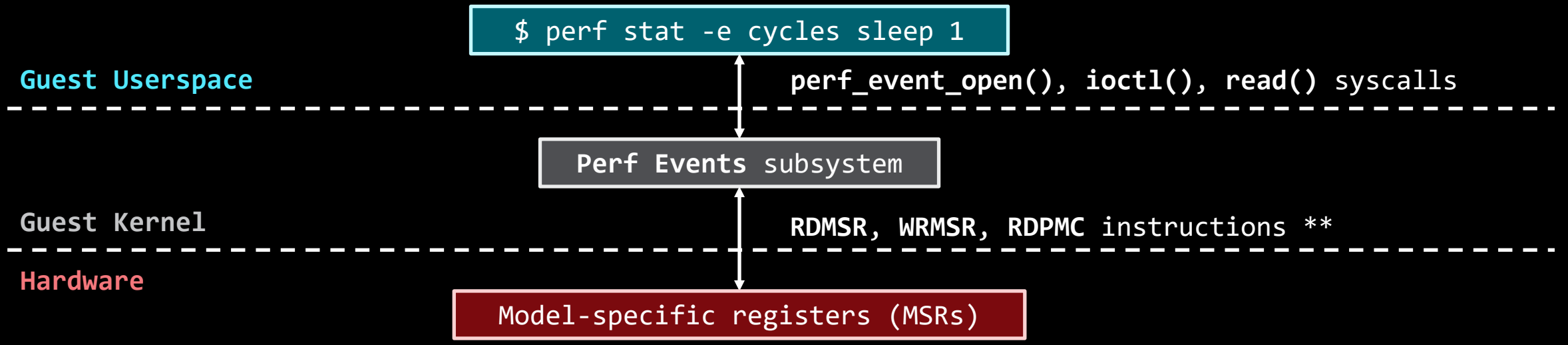
Solution – Mediated PMU

Switch the ownership and the state of hardware counters during world switches

- Before VM-Entry
  - Pause counting of host events and schedule them out; host perf events subsystem relinquishes control
  - Guest counter state is restored
- After VM-Exit
  - Guest counter state is saved
  - Schedule in host events and resume counting; host perf events subsystem regains control
- Guests get direct access to hardware (MSRs) in most cases

Performance Profiling – Inside Guests with Mediated PMU

Running perf inside a KVM guest



** Some MSRs like the Event Selectors are still intercepted for guest event filtering functionality

Performance Profiling – Overhead with Mediated PMU

Comparison – Typical workload analysis usage

```
Guest instance (single vCPU) – Perform a top-down analysis
$ sudo perf stat –M PipelineL1 true
```

```
Host instance – Count VM-Exits for hardware counter accesses
$ sudo perf stat –e kvm:kvm_msr --filter="((ecx >= 0xc0010000 && ecx <= 0xc0010007) || (ecx >=
0xc0010200 && ecx <= 0xc001020b) || (ecx >= 0xc0000300 && ecx <= 0xc0000303))" -e
probe:kvm_emulate_rdpmc –a
```

```
63      kvm:kvm_msr
11      probe:kvm_emulate_rdpmc
```

Legacy – 74 VM-Exits

```
22      kvm:kvm_msr
 0      probe:kvm_emulate_rdpmc
```

Mediated PMU – 22 VM-Exits

Performance Profiling – Overhead with Mediated PMU

Comparison – Typical performance dashboard usage

```
Guest instance (single vCPU) – Measure IPC every second for the next 10 seconds
$ sudo perf stat –e cycles,instructions –-interval-print 1000 –a sleep 10
```

```
Host instance – Count VM-Exits for hardware counter accesses
$ sudo perf stat –e kvm:kvm_msr --filter="((ecx >= 0xc0010000 && ecx <= 0xc0010007) || (ecx >=
0xc0010200 && ecx <= 0xc001020b) || (ecx >= 0xc0000300 && ecx <= 0xc0000303))" -e
probe:kvm_emulate_rdpmc –a
```

```
26      kvm:kvm_msr
23      probe:kvm_emulate_rdpmc
```

Legacy – 49 VM-Exits

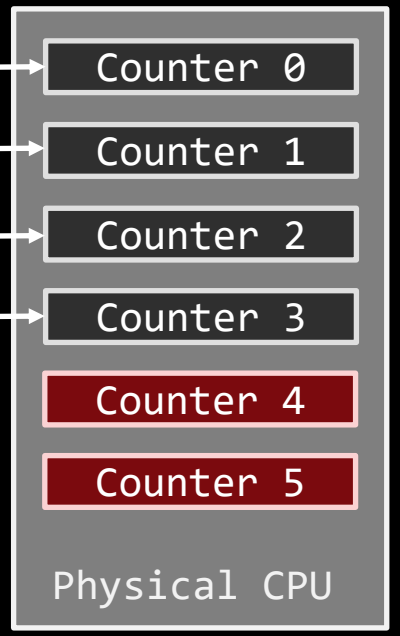
```
 6      kvm:kvm_msr
 0      probe:kvm_emulate_rdpmc
```

Mediated PMU – 6 VM-Exits

Performance Profiling – Accuracy with Mediated PMU

Worst Case – Running `perf` inside a KVM guest while `perf` is also running on the host

```
Host instance
$ sudo perf stat -e cycles:D,instructions:D,branches:D,cache-misses:D -a
```

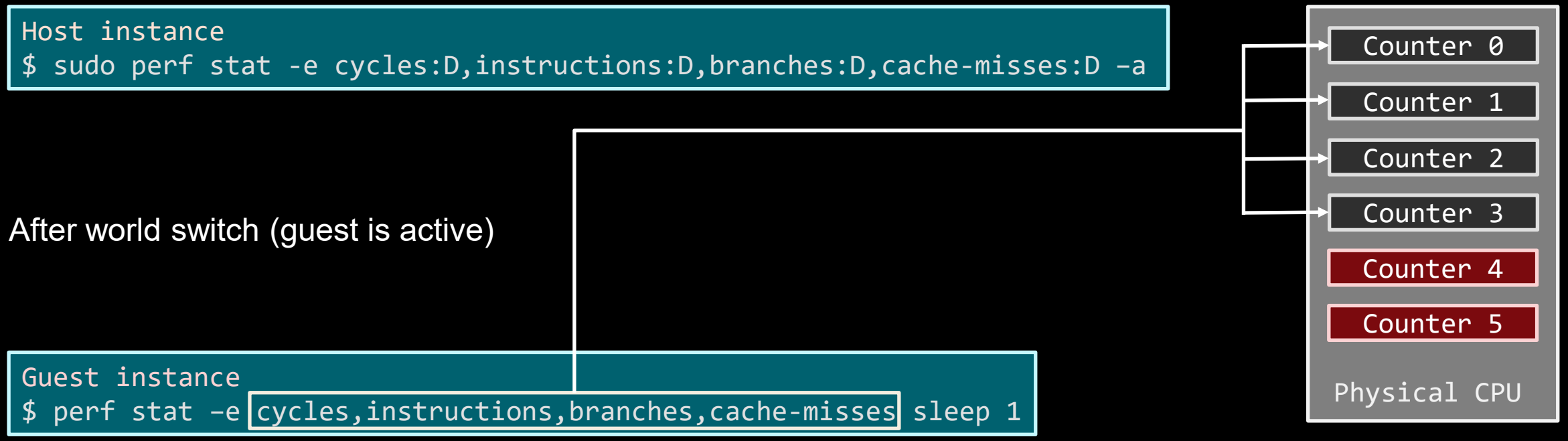


Before world switch (guest is inactive)

```
Guest instance
$ perf stat -e cycles,instructions,branches,cache-misses sleep 1
```

Performance Profiling – Accuracy with Mediated PMU

Worst Case – Running `perf` inside a KVM guest while `perf` is also running on the host



Performance Profiling – Accuracy with Mediated PMU

Worst Case – Running `perf` inside a KVM guest while `perf` is also running on the host

```
Guest instance
$ perf stat -e cycles,instructions,branches,cache-misses sleep 1


 Performance counter stats for 'sleep 1':

          375,852      cycles:u
          311,497      instructions:u
           69,157      branches:u
            6,233      cache-misses:u

      1.010792516 seconds time elapsed

      0.001689000 seconds user
      0.008446000 seconds sys
```

vs. Best Case

```
377,066      cycles:u
311,251      instructions:u
 69,108      branches:u
  6,206      cache-misses:u
```

Performance Profiling – Inside Guests with Mediated PMU

Pros

Low runtime overhead

- Direct access to most MSRs
- Avoid expensive VM-Exits caused by MSR interceptions and their side-effects

Native accuracy

- Avoid conflicts between host and guest perf events due to mutual exclusion
- Consistent results (guest events are not silently rejected due to scheduling conflicts)

Performance Profiling – Inside Guests with Mediated PMU

Cons

High world switch overhead

- The guest counter state has to be saved and restored for every world switch
- If a CPU mostly executes in guest context and runs for long uninterrupted stretches, like in the case of compute-bound cloud instances, the cost is amortized over a large period of guest runtime

Loss of host-side observability of guests

- Host perf events can no longer monitor guest activity due to mutual exclusion (design trade-off)
- Breaks tools like `perf kvm`

Performance Profiling – Inside Guests with Mediated PMU

Optimization and Future Work

Offload context switch of guest counter state to hardware

- Faster switching of host and guest states
- Reclaim part of the additional world switch overhead

Confidential compute requires mutual exclusion

- Prevent host from snooping guest performance metrics and potentially launching side-channel attacks
- Hardware can encrypt guest counter state and protect it from attacks

Adoption by other architectures

- Currently an x86-only feature

Other Takeaways

Patience

Multiple rounds of collaborative development, reviews and testing over a period of around 2 years to go from

```
Subject: [RFC PATCH 00/41] KVM: x86/pmu: Introduce passthrough vPMU
Date: Fri, 26 Jan 2024 16:54:03 +0800
Message-ID: <20240126085444.324918-1-xiong.y.zhang@linux.intel.com>
```

to

```
commit cb5573868ea85ddbc74dd9a917acd1e434d21390
Merge: c87c79345ea8 b1195183ed42
Author: Linus Torvalds <torvalds@linux-foundation.org>
Date: Fri Feb 13 11:31:15 2026 -0800

    Merge tag 'for-linus' of git://git.kernel.org/pub/scm/virt/kvm/kvm
```

Acknowledgements

Dapeng Mi
Kan Liang
Mingwei Zhang
Peter Ziljstra
Sean Christopherson
Xiong Zhang
Binbin Wu
Chao Gao
Manali Shukla
Xudong Hao
Yongwei Ma

…and more

References

Patch series

- RFC – https://lore.kernel.org/all/20240126085444.324918-1-xiong.y.zhang@linux.intel.com/
- Version 2 – https://lore.kernel.org/all/20240506053020.3911940-1-mizhang@google.com
- Version 3 – https://lore.kernel.org/all/20240801045907.4010984-1-mizhang@google.com
- Version 4 – https://lore.kernel.org/all/20250324173121.1275209-1-mizhang@google.com
- Version 5 – https://lore.kernel.org/all/20250806195706.1650976-1-seanjc@google.com
- Version 6 – https://lore.kernel.org/all/20251206001720.468579-1-seanjc@google.com/

Copyright and Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate releases, for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED 'AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION.


AMD