Perfctr-Xen: A framework for Performance Counter Virtualization

Ruslan Nikolaev and Godmar Back
Virginia Polytechnic Institute
Blacksburg

Overview

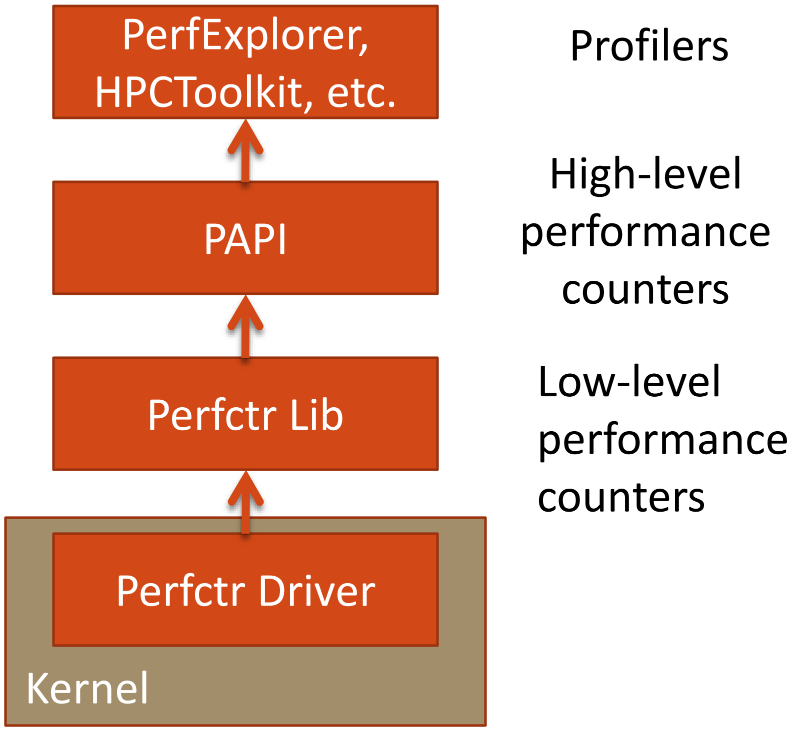
- IaaS widely use virtual machine monitors
  - Type 1 hypervisors: Xen, KVM, ESX …
- Commonly used performance analysis tools (e.g., PAPI) cannot be used because existing VMM and guests do not provide necessary per-thread virtualization support for hardware event counters
- Our contribution: Perfctr-Xen:
  - Framework for performance counter virtualization
  - Software-compatible with widely used perfctr library
  - Techniques for collaboration of guest and hypervisor
- Experimental validation

Existing Performance Counter Virtualization Solutions

- XenoProf
  - Extension of Oprofile system-wide profiler
  - Does not provide per-domain abstraction of hardware counter facilities (supports only 1 domain at a time)
- VPMU driver
  - Treats PMU registers like ordinary registers (saved/restored by VMM)
  - Requires use of hardware assisted virtualization mode; support for limited number of architecture generations since VMM must contain architecture-specific code
  - Not compatible with all architectures

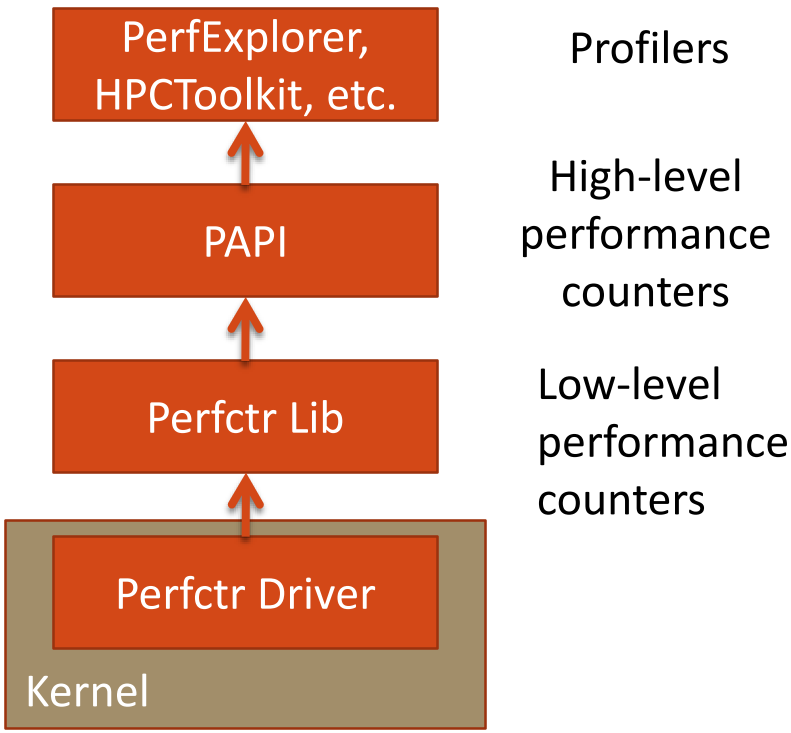
Perfctr-Xen

- Perfctr (Native)

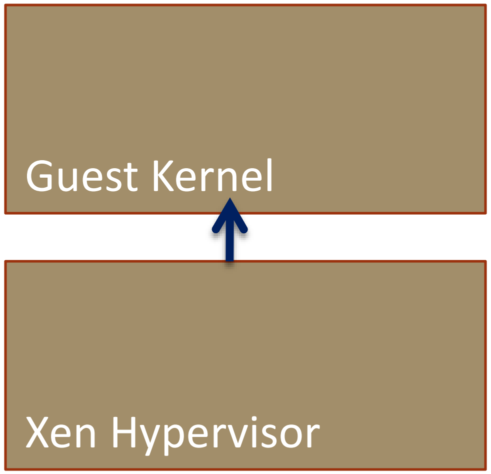


Perfctr-Xen

- Perfctr (Native)

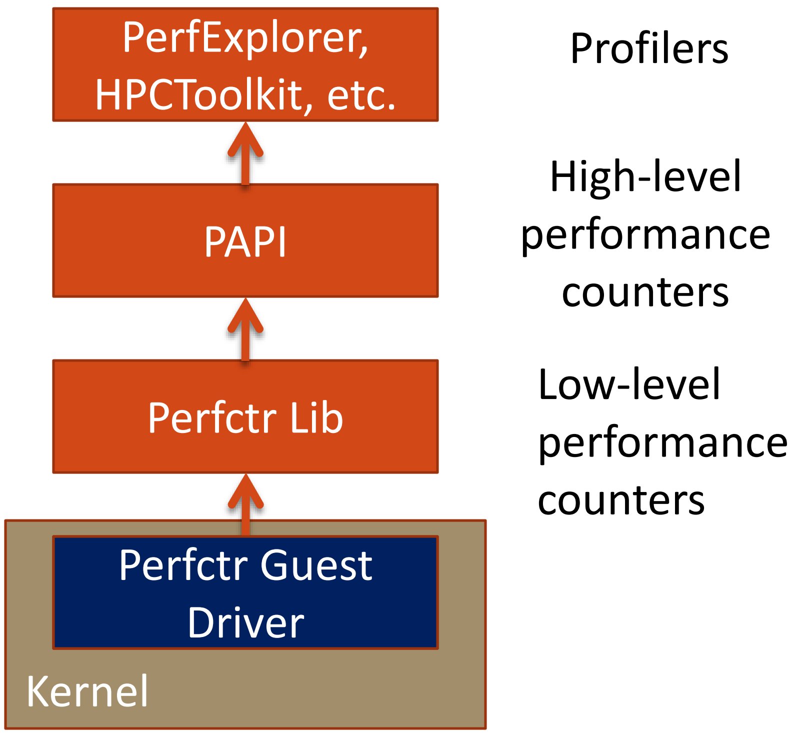


- Perfctr-Xen

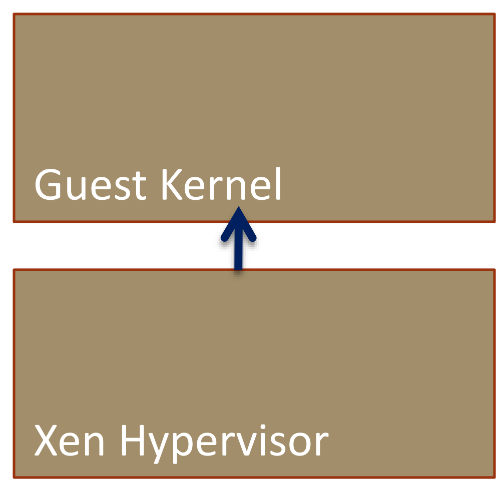


Perfctr-Xen

- Perfctr (Native)

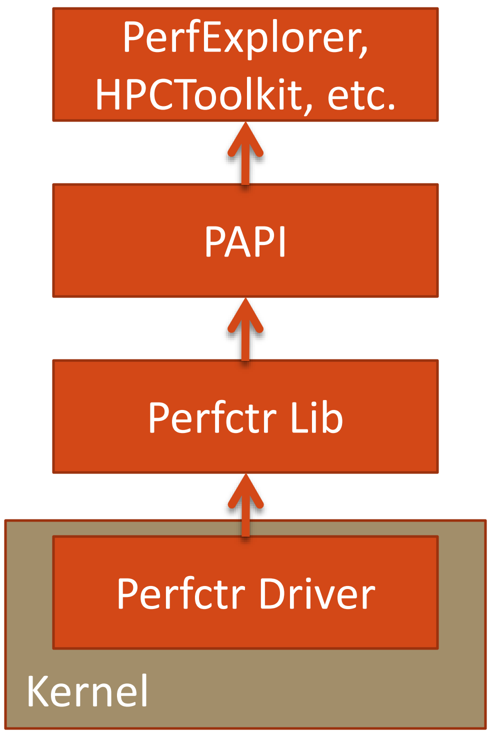


- Perfctr-Xen



Perfctr-Xen

- Perfctr (Native)

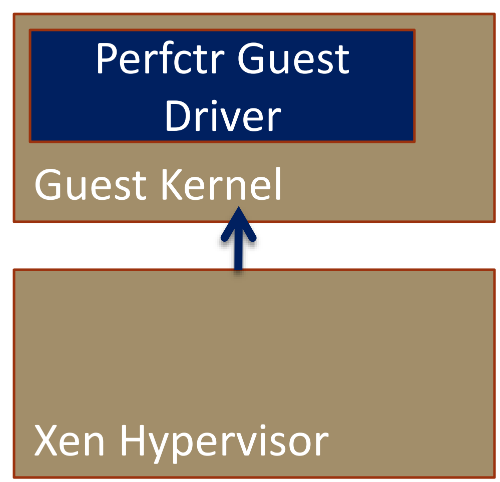


Profilers

High-level performance counters

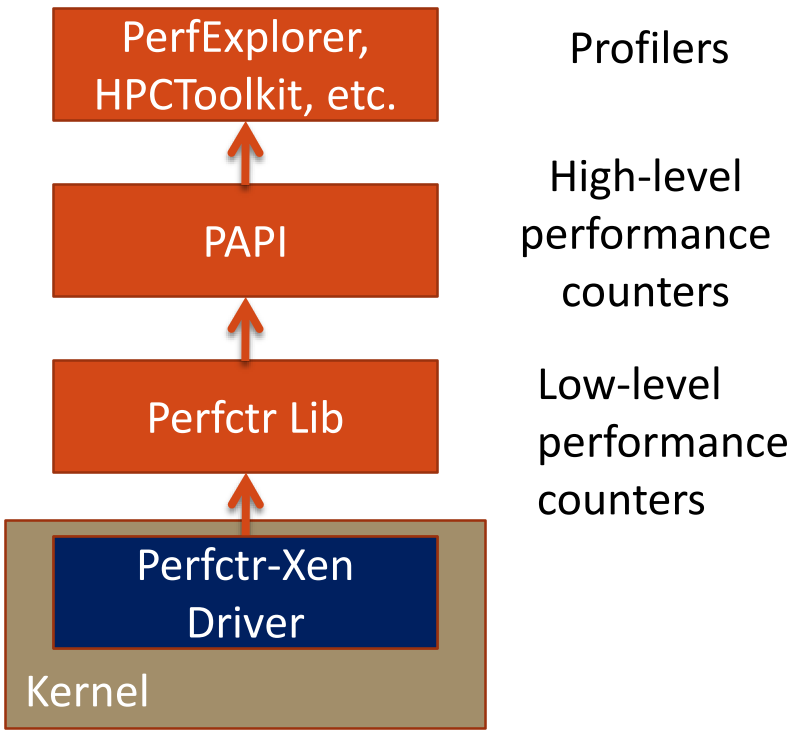
Low-level performance counters

- Perfctr-Xen

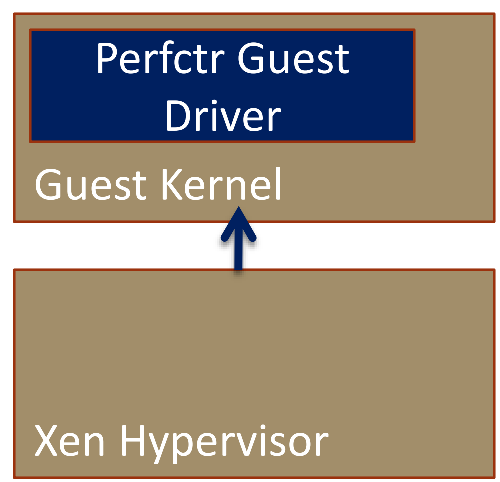


Perfctr-Xen

- Perfctr (Native)

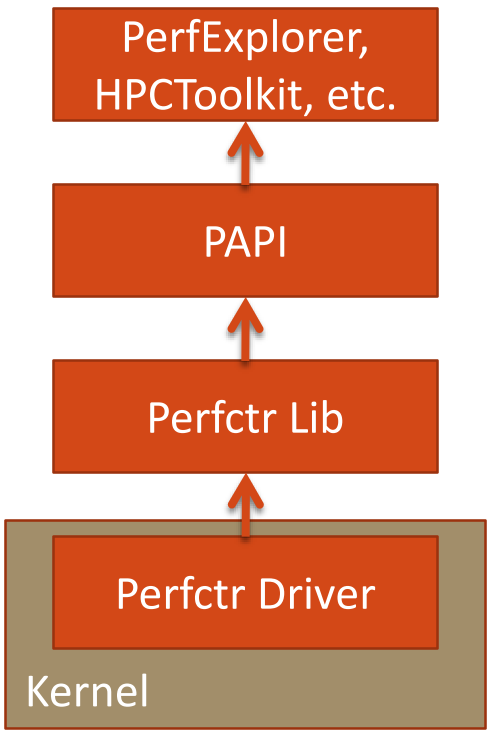


- Perfctr-Xen



Perfctr-Xen

- Perfctr (Native)

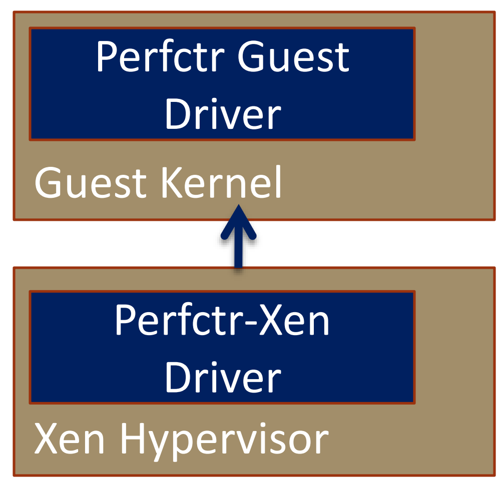


Profilers

High-level performance counters

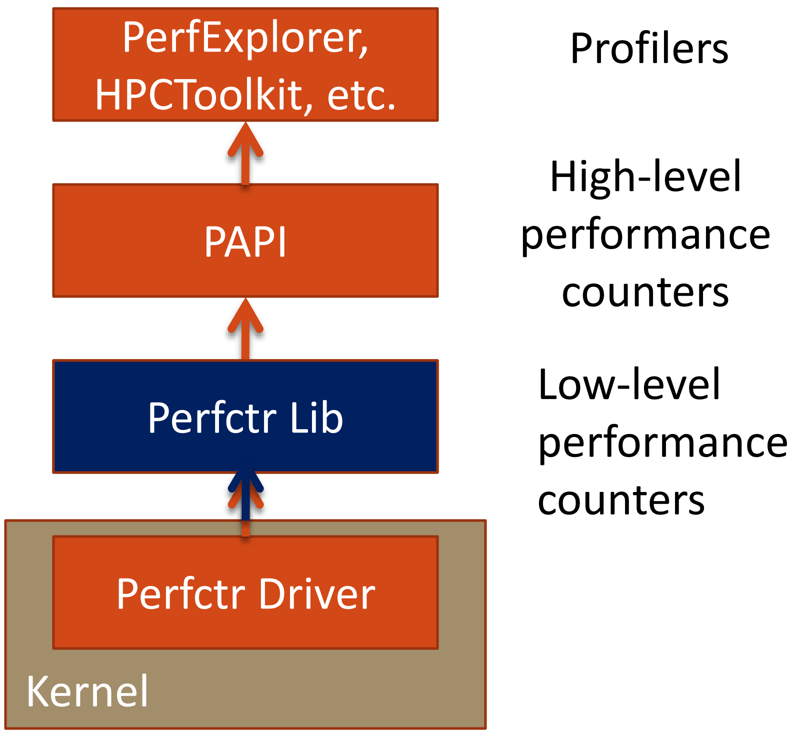
Low-level performance counters

- Perfctr-Xen

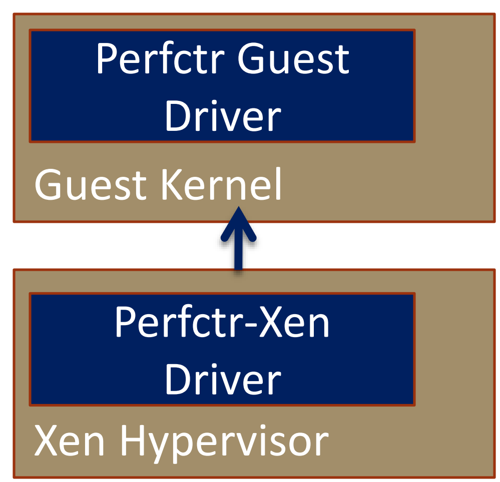


Perfctr-Xen

- Perfctr (Native)

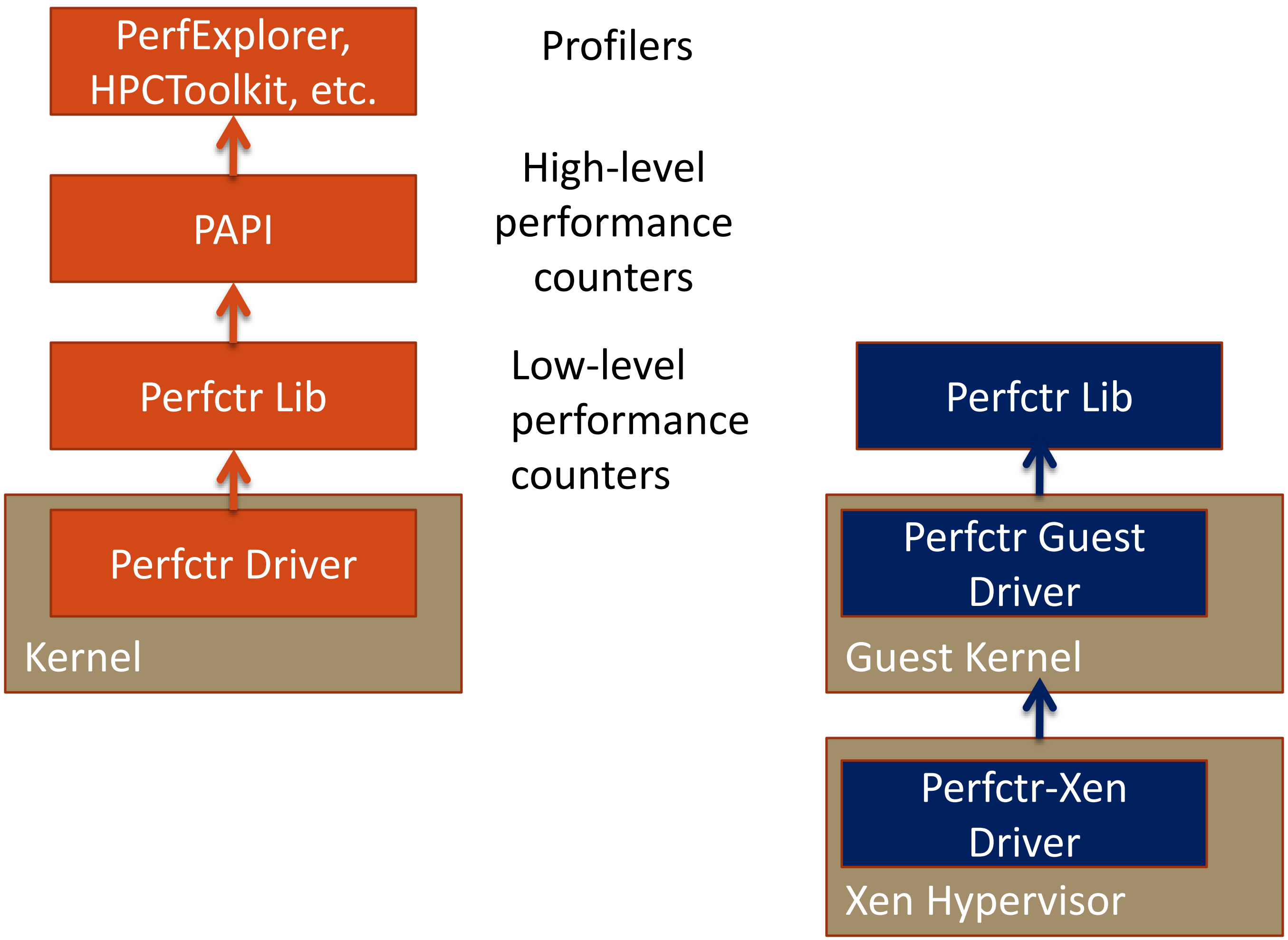


- Perfctr-Xen



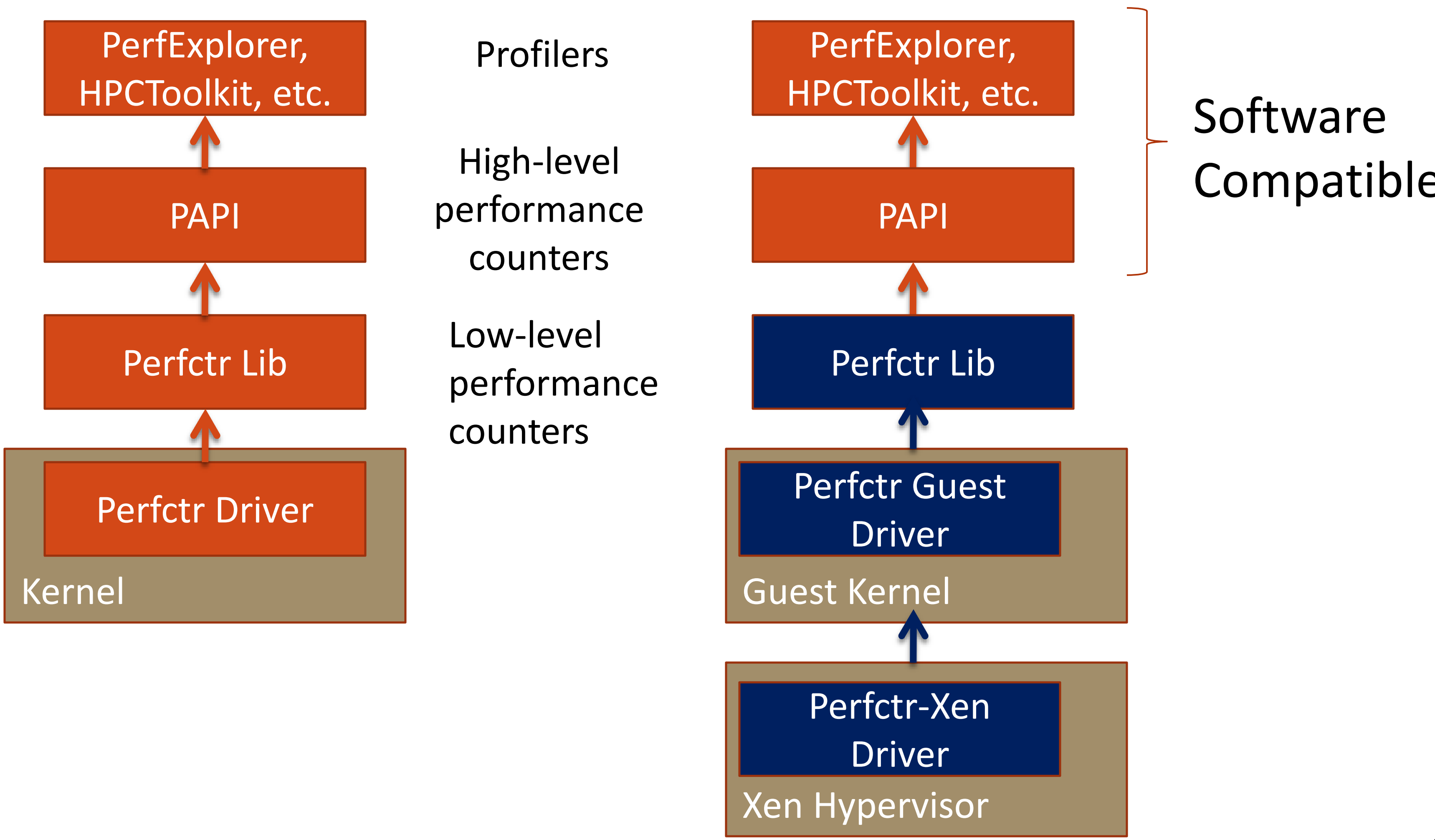
Perfctr-Xen

- Perfctr (Native)
- Perfctr-Xen



Perfctr-Xen

- Perfctr (Native)
- Perfctr-Xen



Per-thread PMU Virtualization

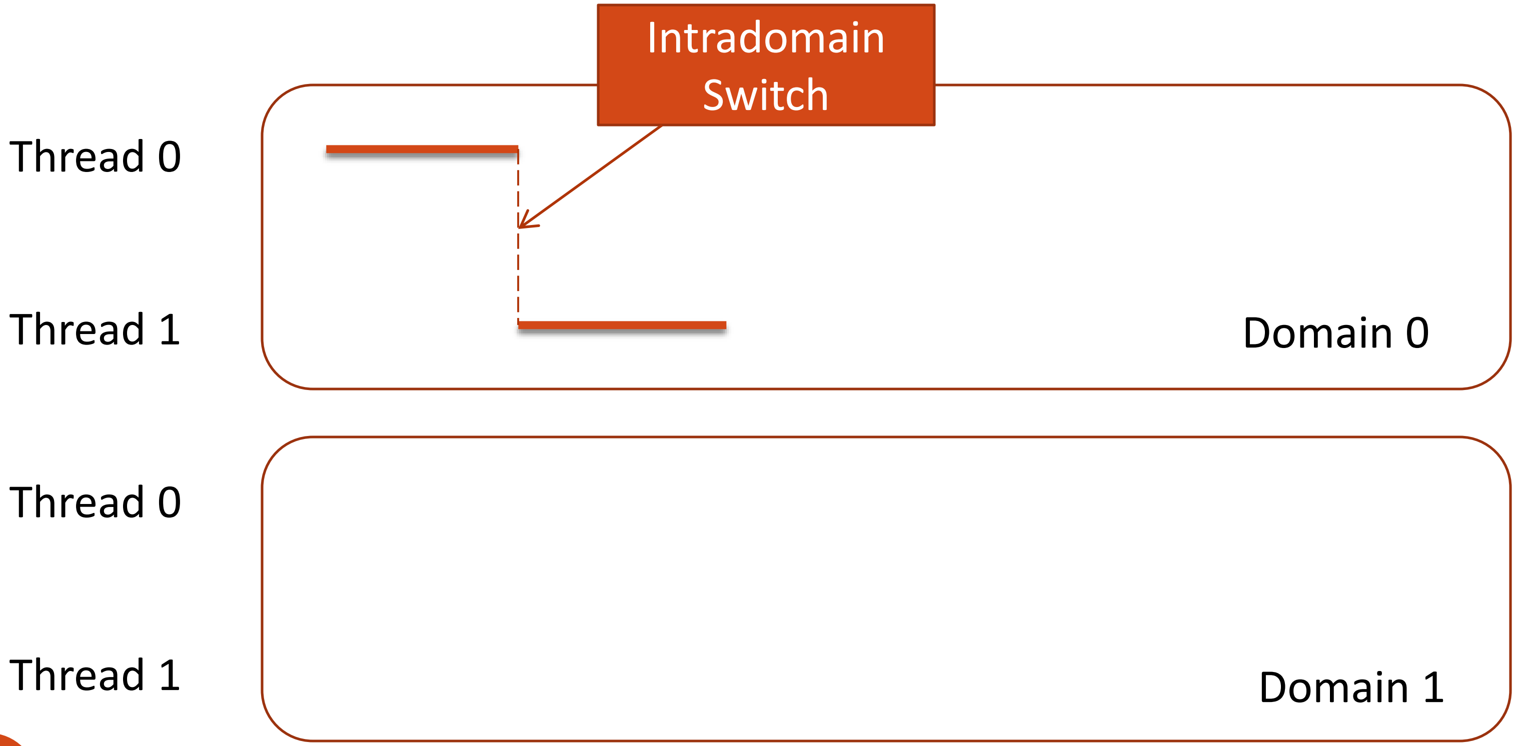
Logical per-thread value includes only events incurred during the thread execution

Per-thread PMU Virtualization

Logical per-thread value includes only events incurred during the thread execution

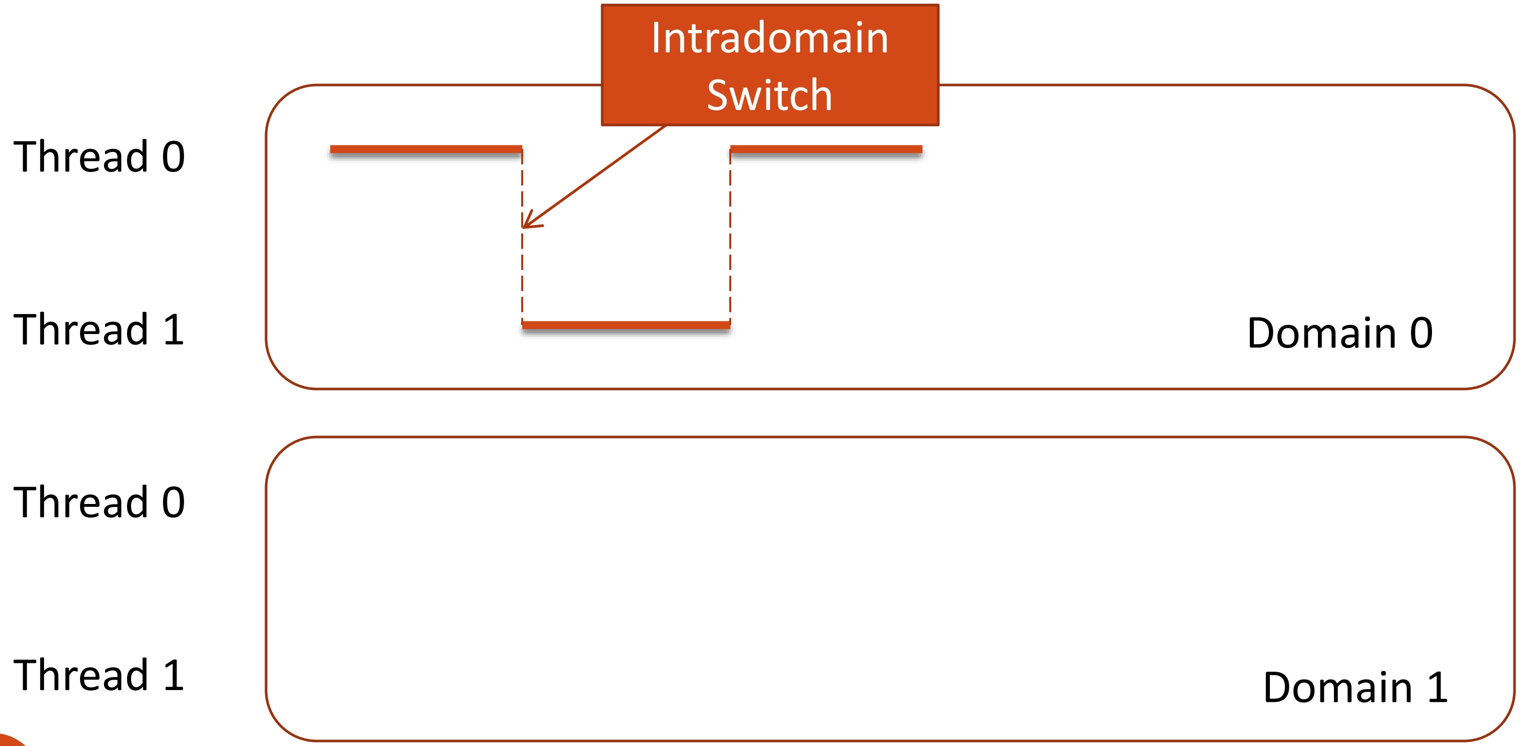
Per-thread PMU Virtualization

Logical per-thread value includes only events incurred during the thread execution



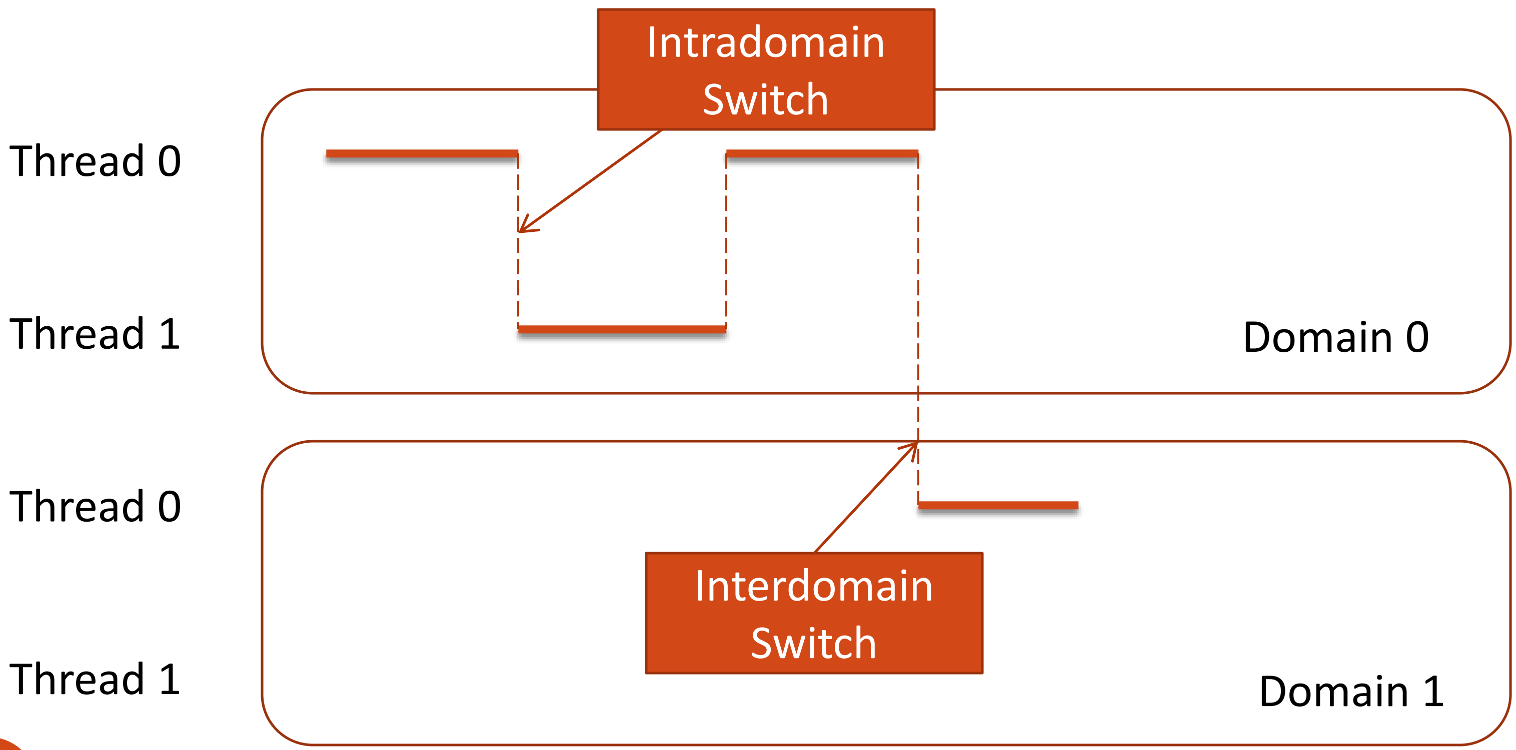
Per-thread PMU Virtualization

Logical per-thread value includes only events incurred during the thread execution



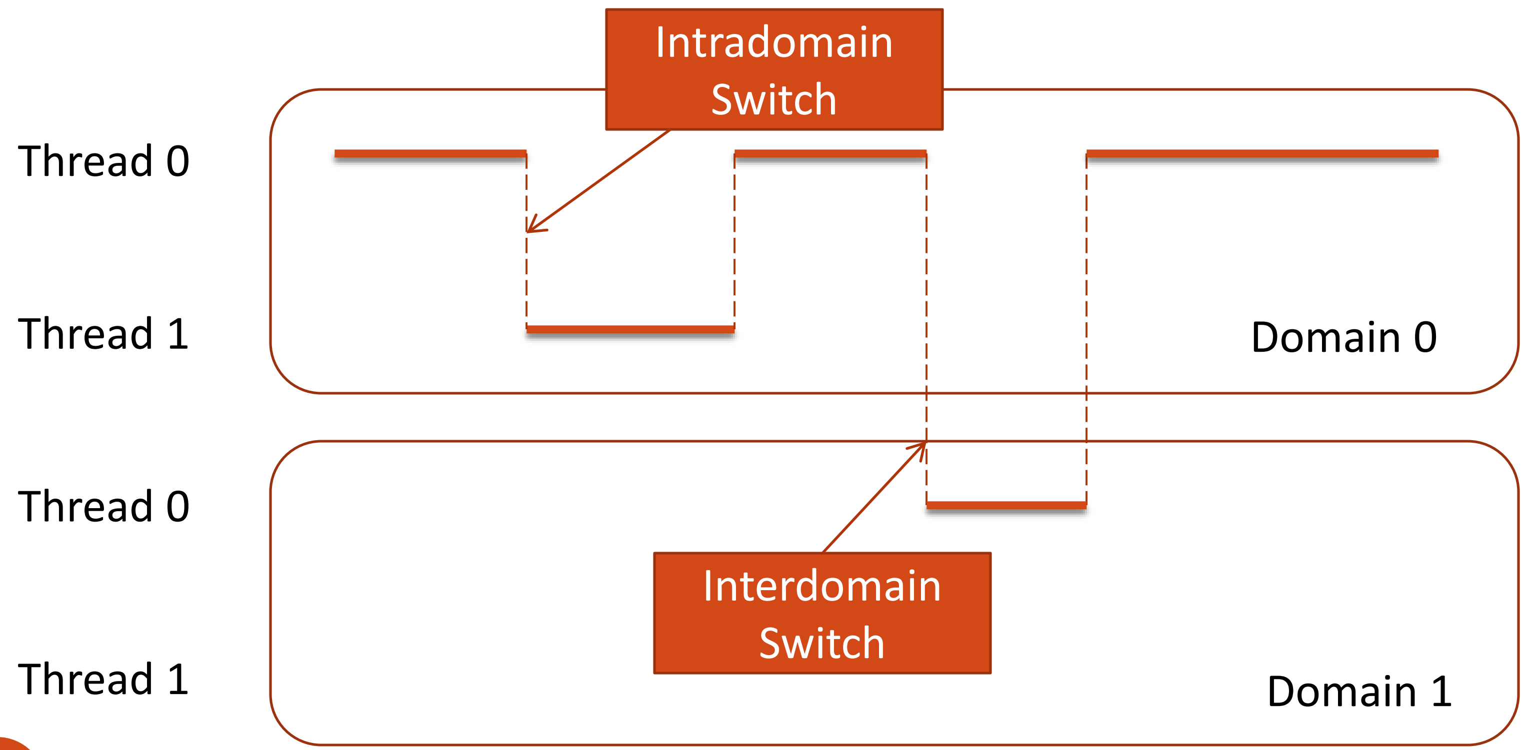
Per-thread PMU Virtualization

Logical per-thread value includes only events incurred during the thread execution



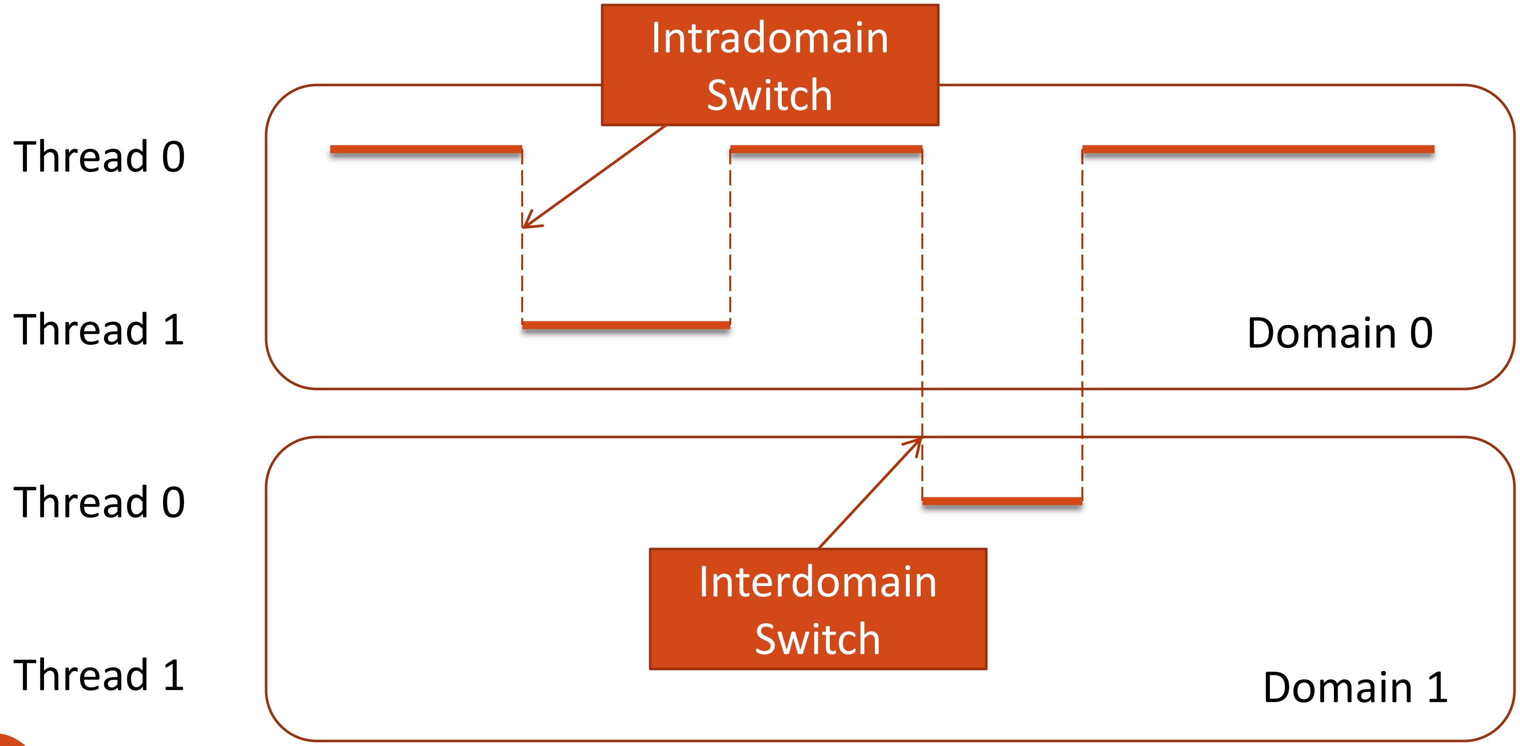
Per-thread PMU Virtualization

Logical per-thread value includes only events incurred during the thread execution



Per-thread PMU Virtualization

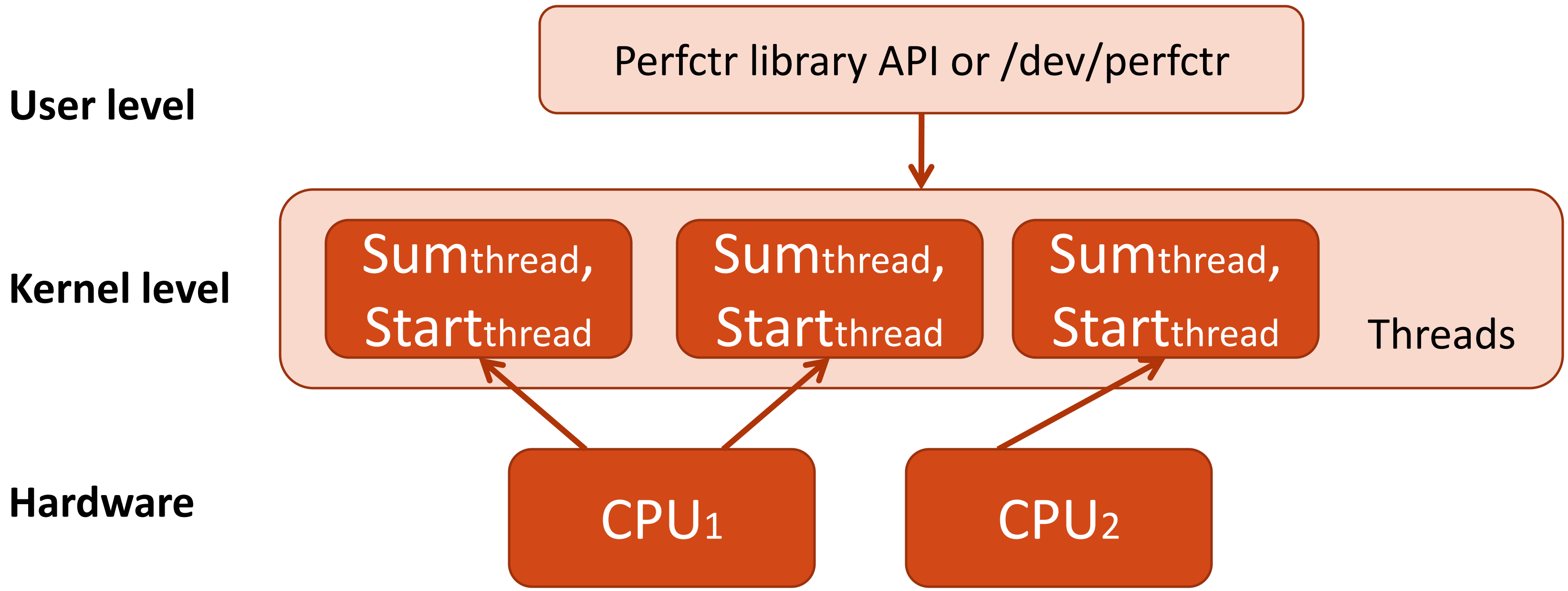
Logical per-thread value includes only events incurred during the thread execution



Perfctr Library

Modes of operation

- A-mode: an event count in some region of a program
- I-mode: an interrupt after a certain number of events has occurred



Perfctr: A-mode counters

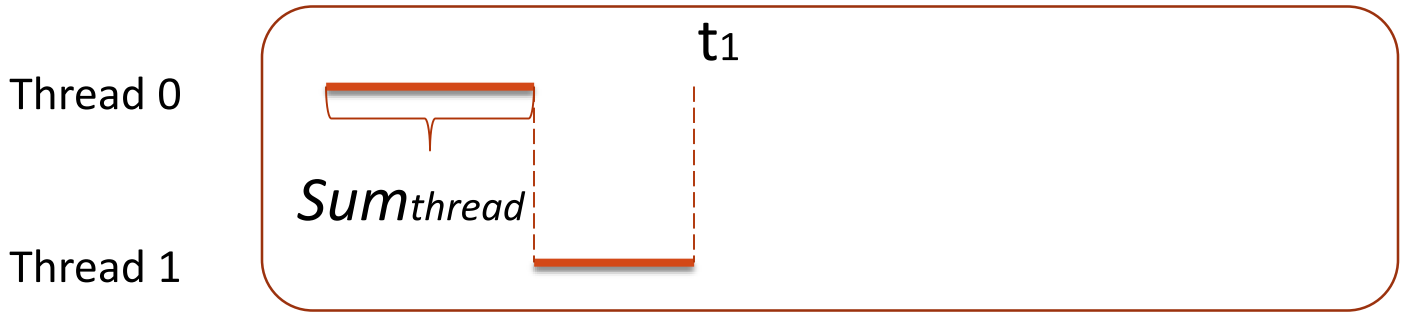
Perfctr: A-mode counters

Sum_{thread} records accumulated event count

Perfctr: A-mode counters

Sum_{thread} records accumulated event count

Kernel records physical value to $Start_{thread}$ = Phys(t_1)



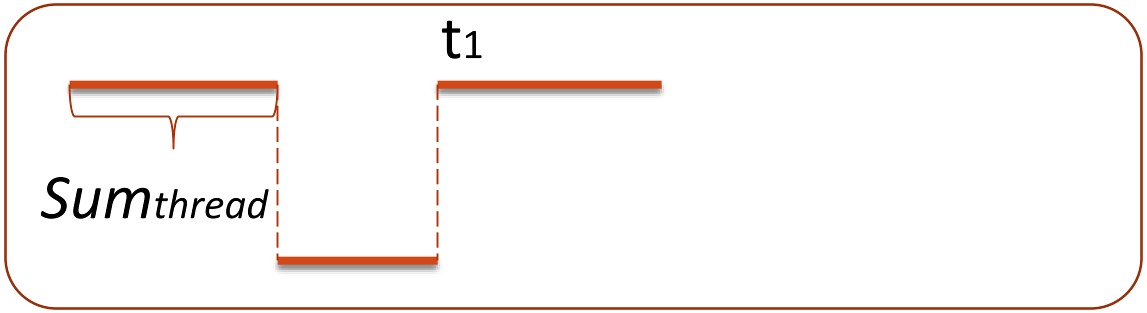
Perfctr: A-mode counters

Sum_{thread} records accumulated event count

Kernel records physical value to $Start_{thread}$ = Phys(t_1)

Thread 0

Thread 1



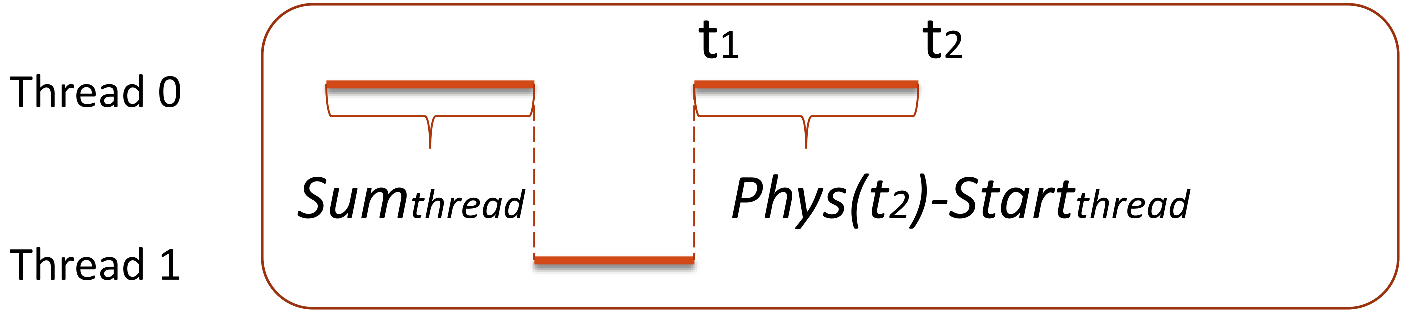
Perfctr: A-mode counters

Sum_{thread} records accumulated event count

Kernel records physical value to $Start_{thread} = Phys(t_1)$

Thread samples physical value $Phys(t_2)$, computes

Logical value $Log_{thread} = Sum_{thread} + (Phys(t_2) - Start_{thread})$



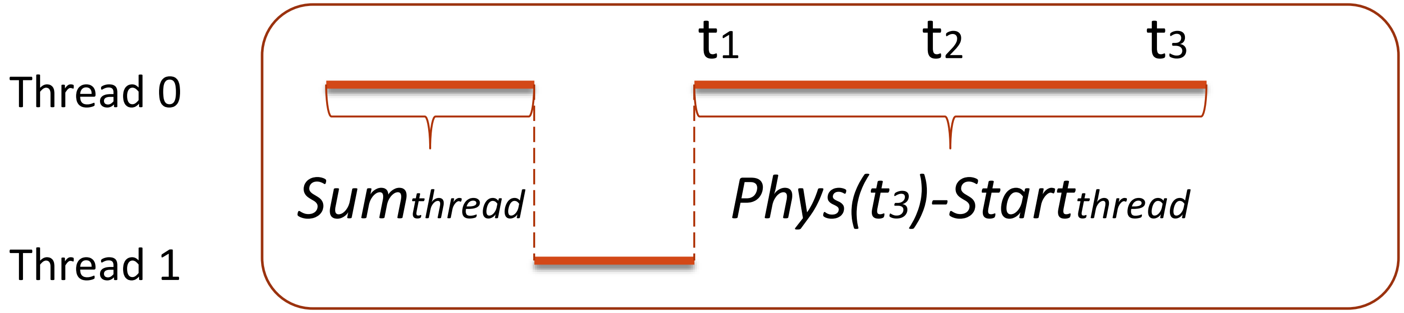
Perfctr: A-mode counters

Sum_{thread} records accumulated event count

Kernel records physical value to $Start_{thread} = \mathrm{Phys}(t_1)$

Thread samples physical value $\mathrm{Phys}(t_2)$, computes

Logical value $Log_{thread} = Sum_{thread} + (\mathrm{Phys}(t_2) - Start_{thread})$



Perfctr: A-mode counters

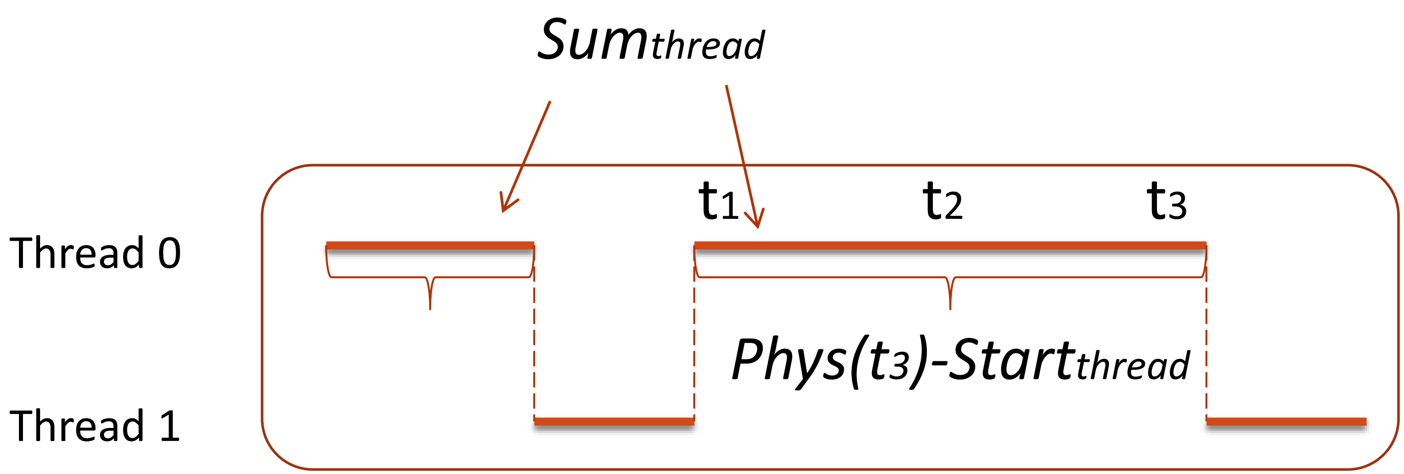
Sum_{thread} records accumulated event count

Kernel records physical value to $Start_{thread}$ = Phys(t_1)

Thread samples physical value Phys(t_2), computes

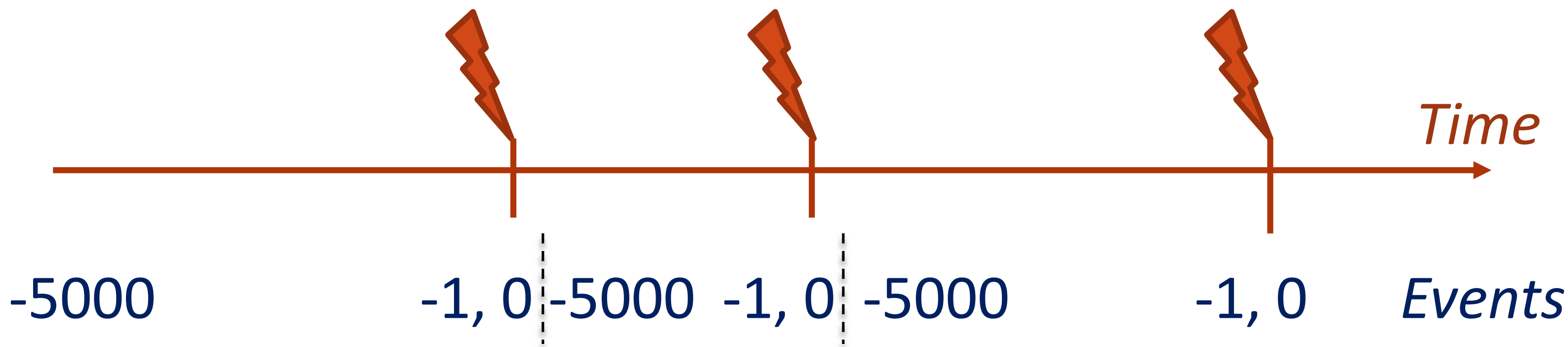
Logical value $Log_{thread} = Sum_{thread} + (Phys(t_2) - Start_{thread})$

Kernel increments $Sum_{thread} = Sum_{thread} + (Phys(t_3) - Start_{thread})$



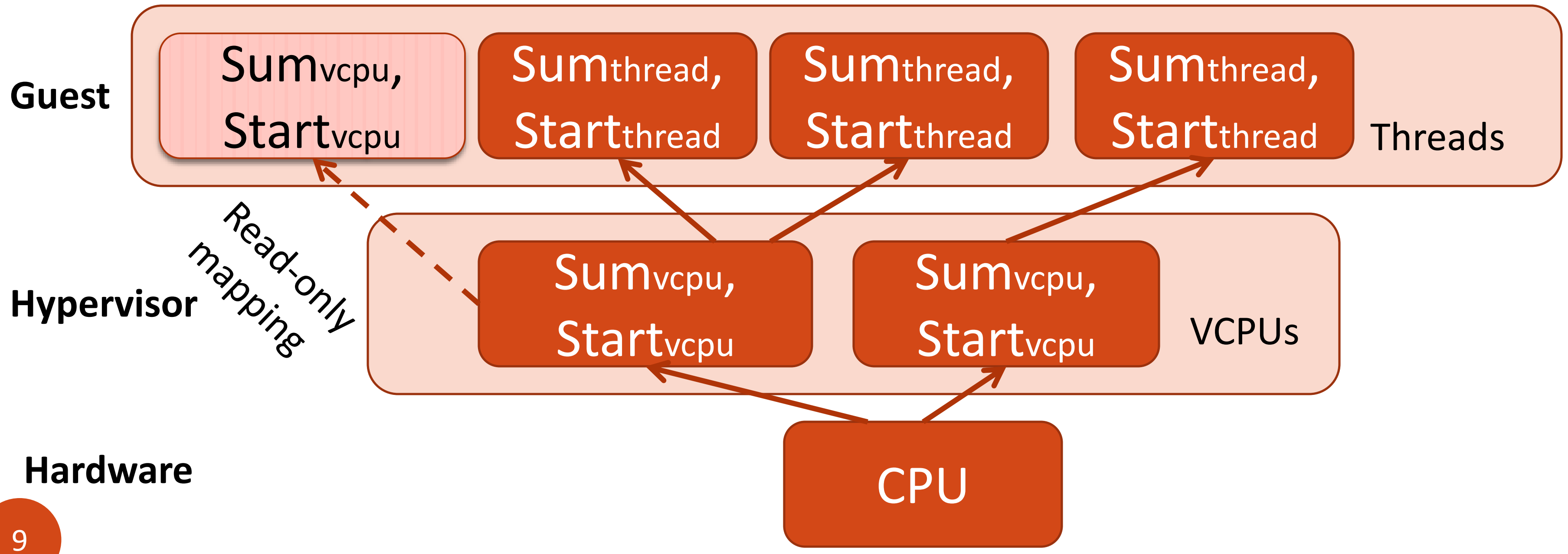
Perfctr: I-mode counters

- PMU registers trigger interrupt on zero-overflow
- Physical register initialized to negated sample period
- Requires that physical value be saved & restored on each context switch
- Compute logical accumulated value similar to a-mode



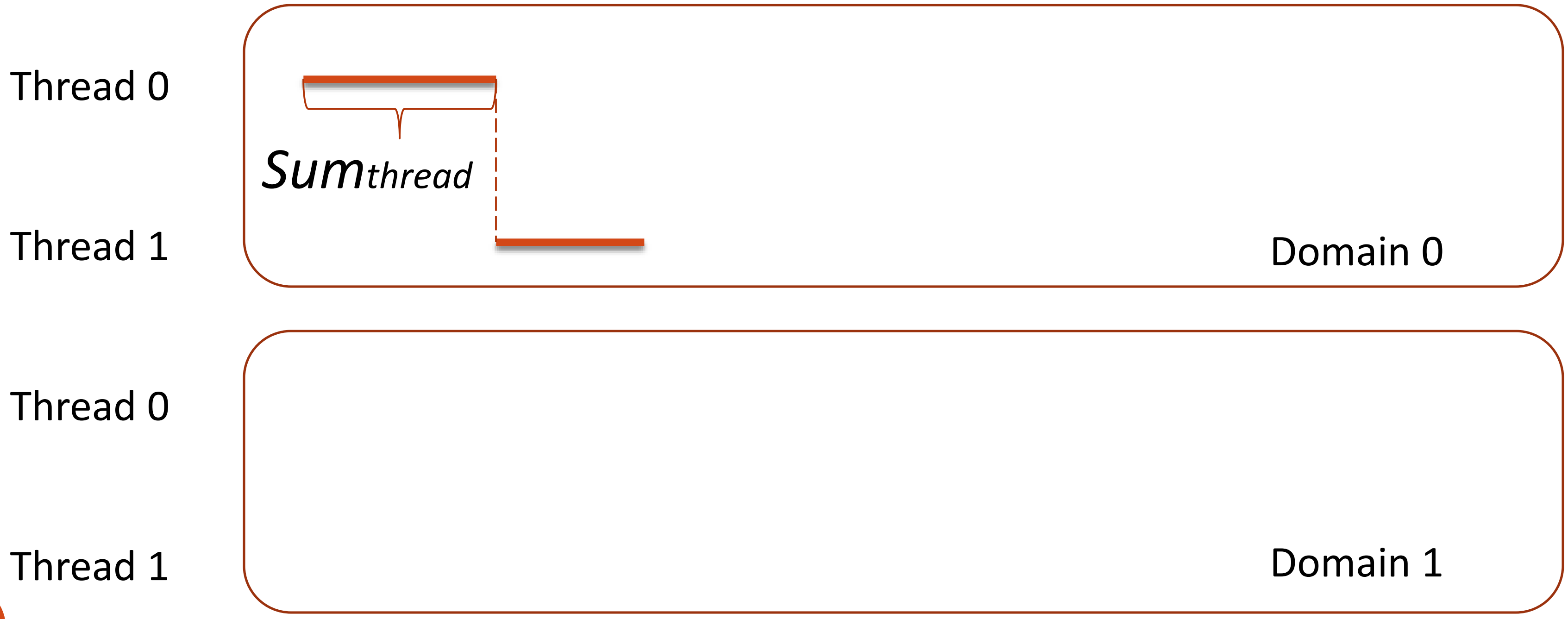
Perfctr-Xen: A-mode counters

- Requires cooperation of guest kernel and hypervisor:
  - Guest: maintains per-thread state: Sum_{thread}, $Start_{thread}$
  - Hypervisor: a per-VCPU (Virtual CPU) state: Sum_{vcpu}, $Start_{vcpu}$
- Guest kernel makes per-VCPU state available user threads

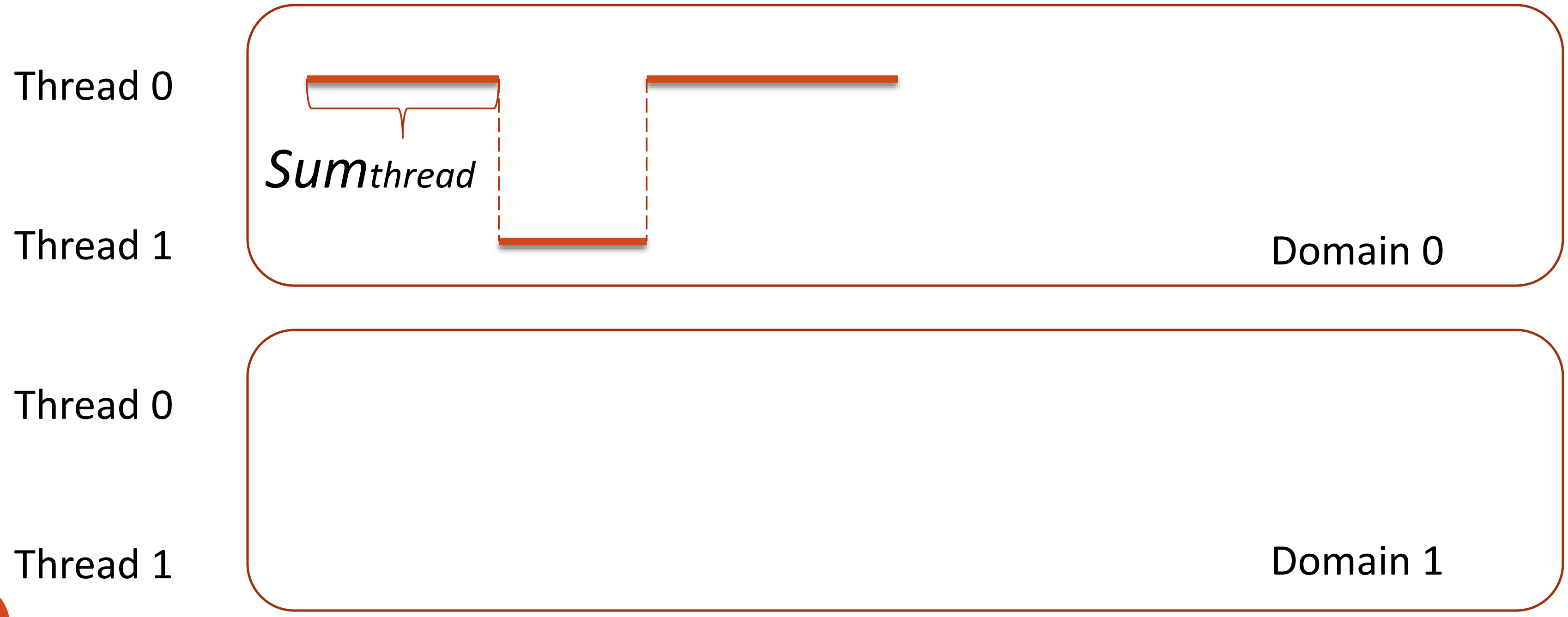


Perfctr-Xen: A-mode counters

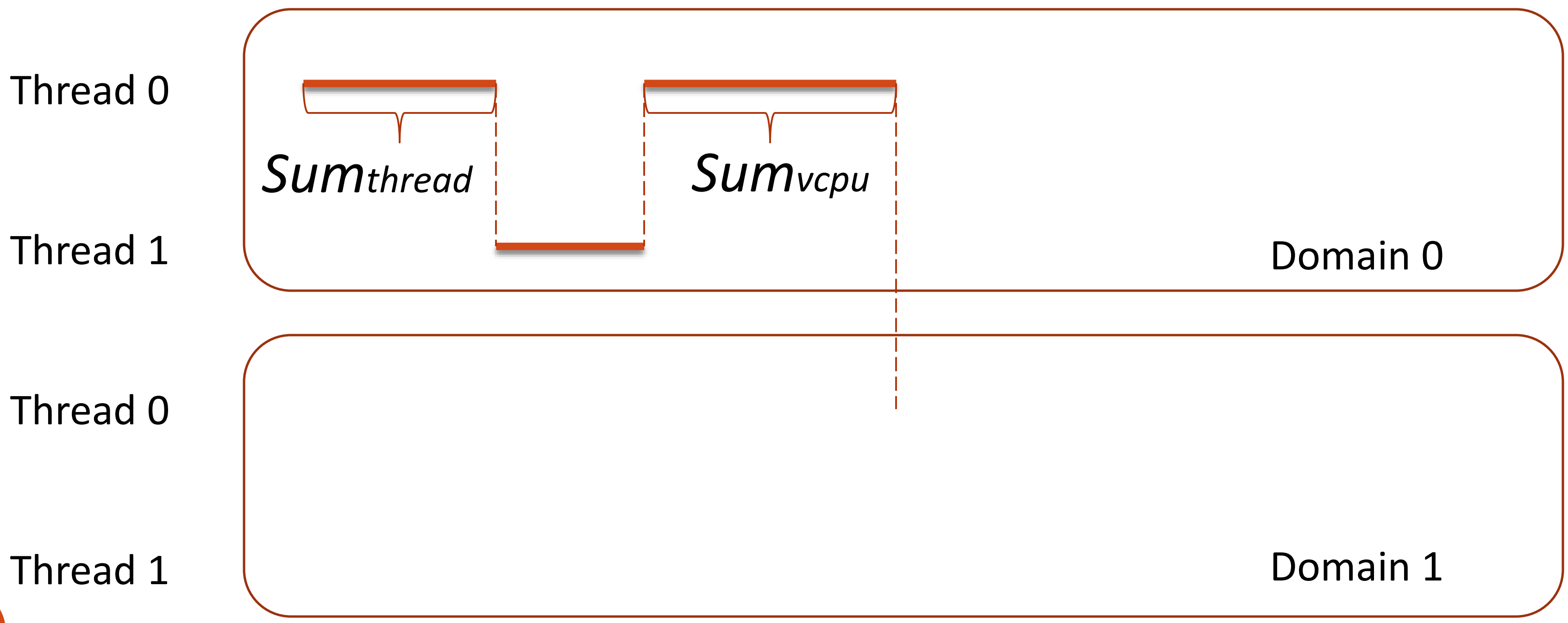
Perfctr-Xen: A-mode counters



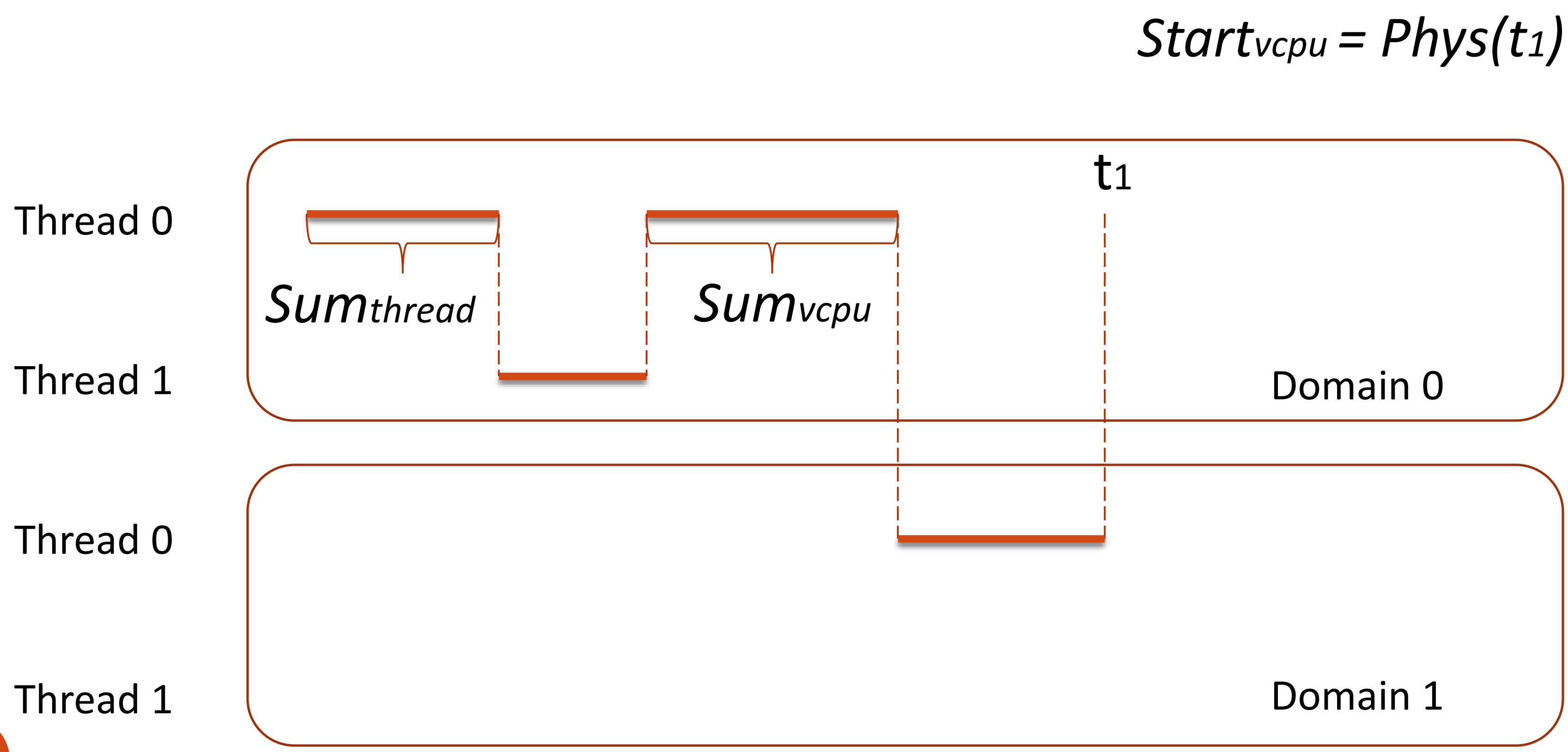
Perfctr-Xen: A-mode counters



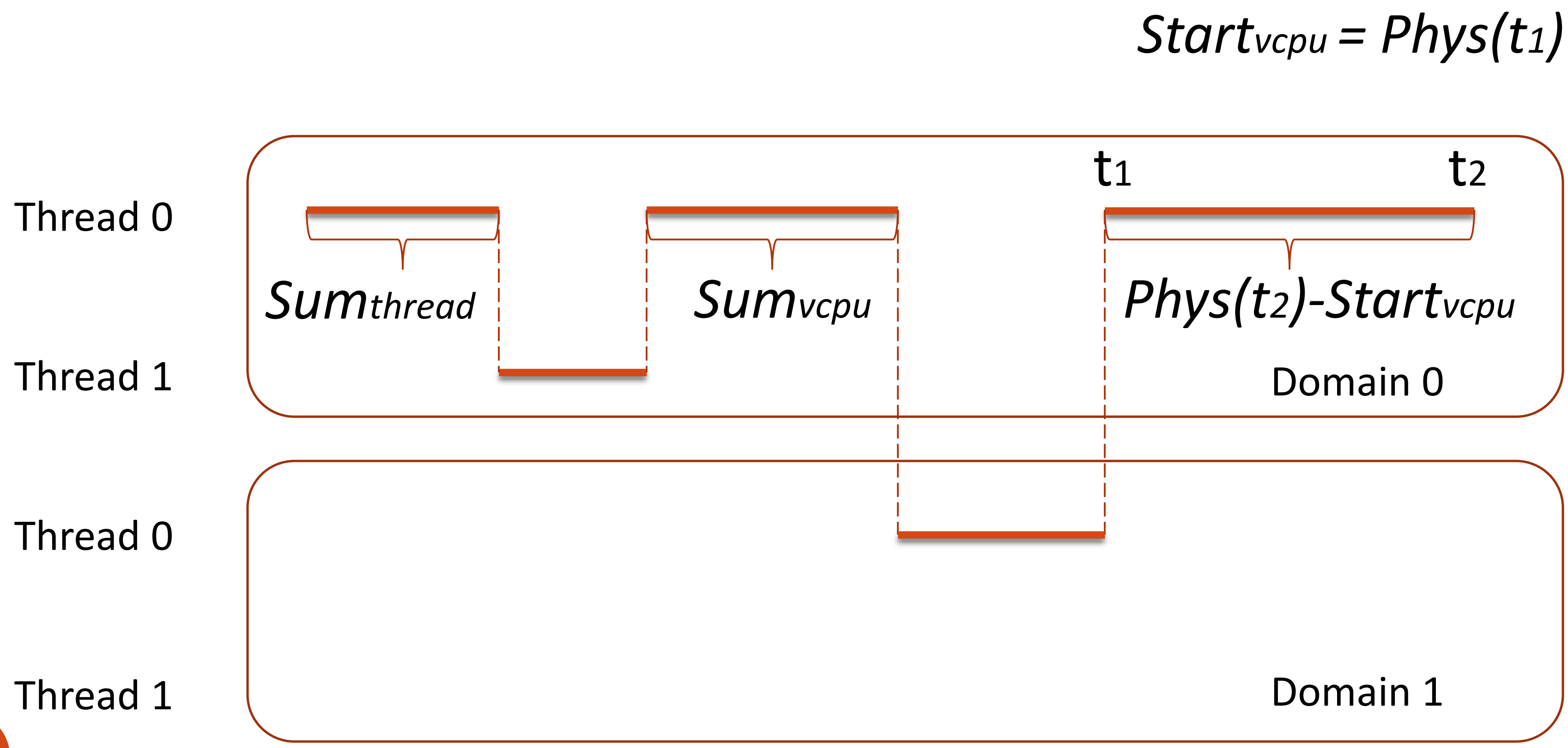
Perfctr-Xen: A-mode counters



Perfctr-Xen: A-mode counters



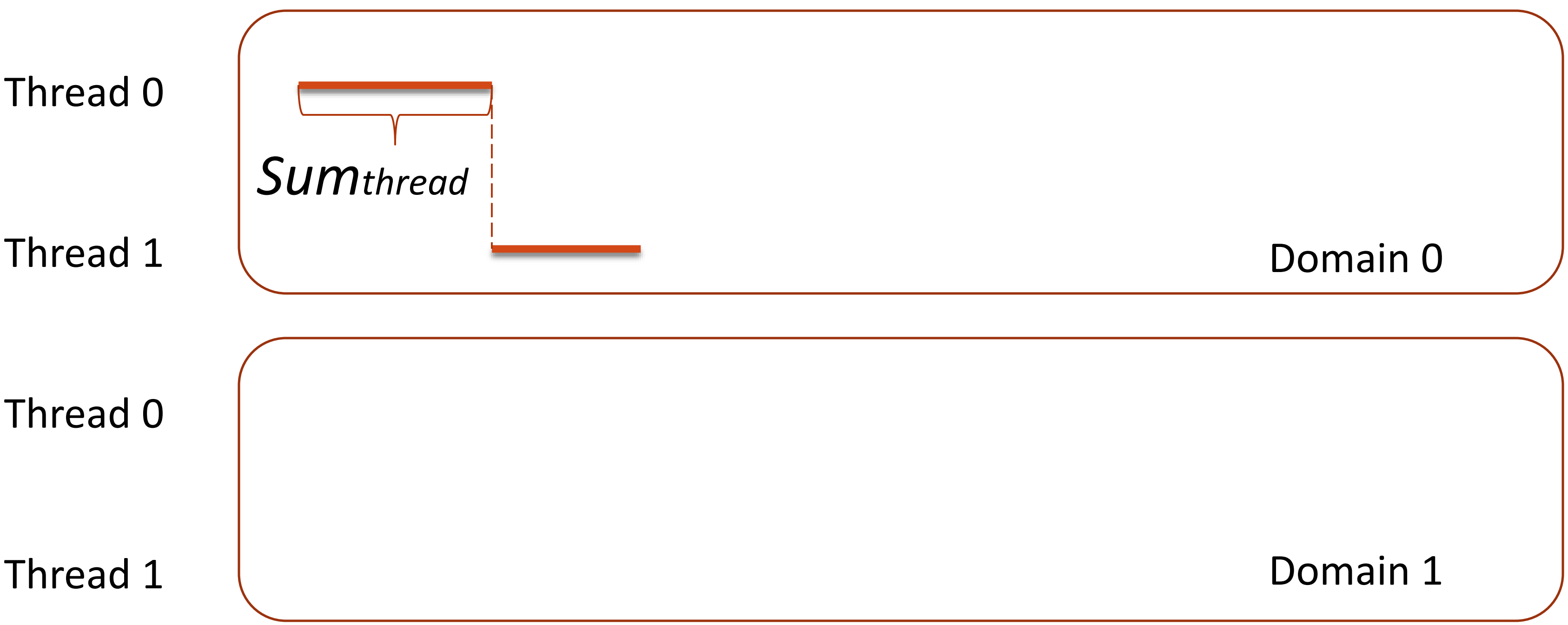
Perfctr-Xen: A-mode counters



Perfctr-Xen: A-mode counters

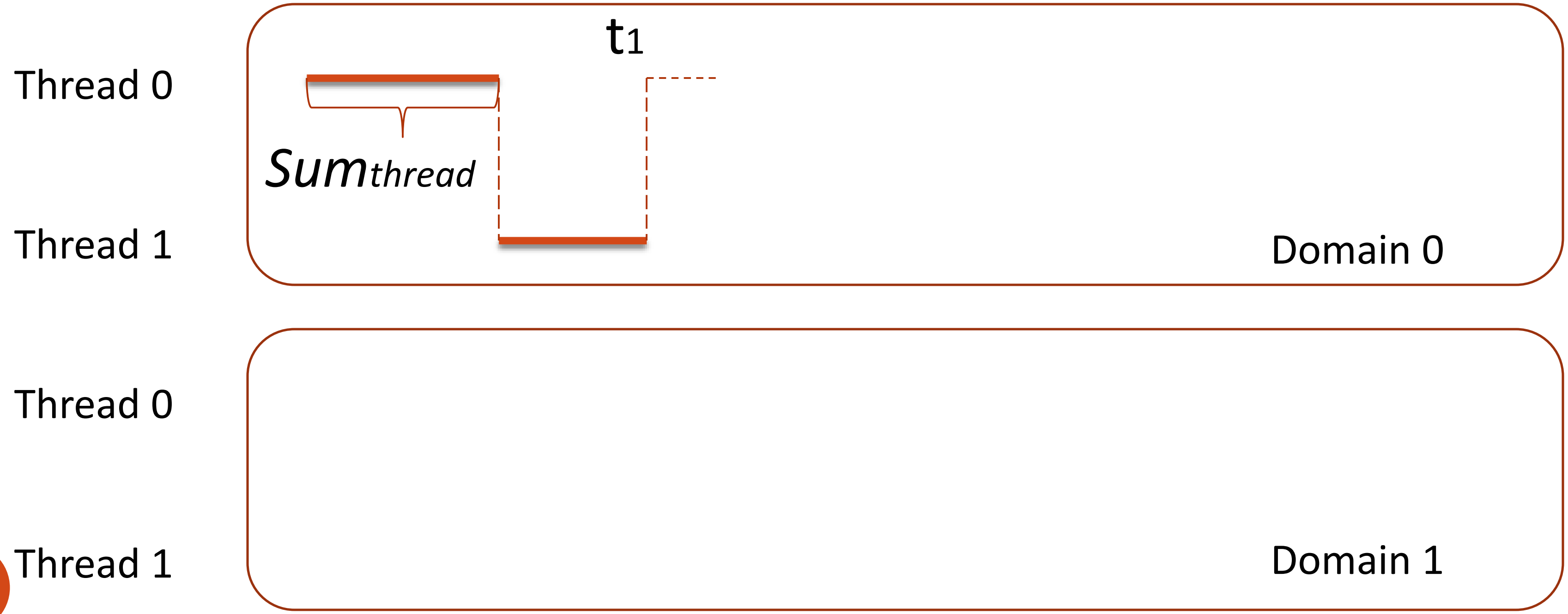
Perfctr-Xen: A-mode counters

Perfctr-Xen: A-mode counters



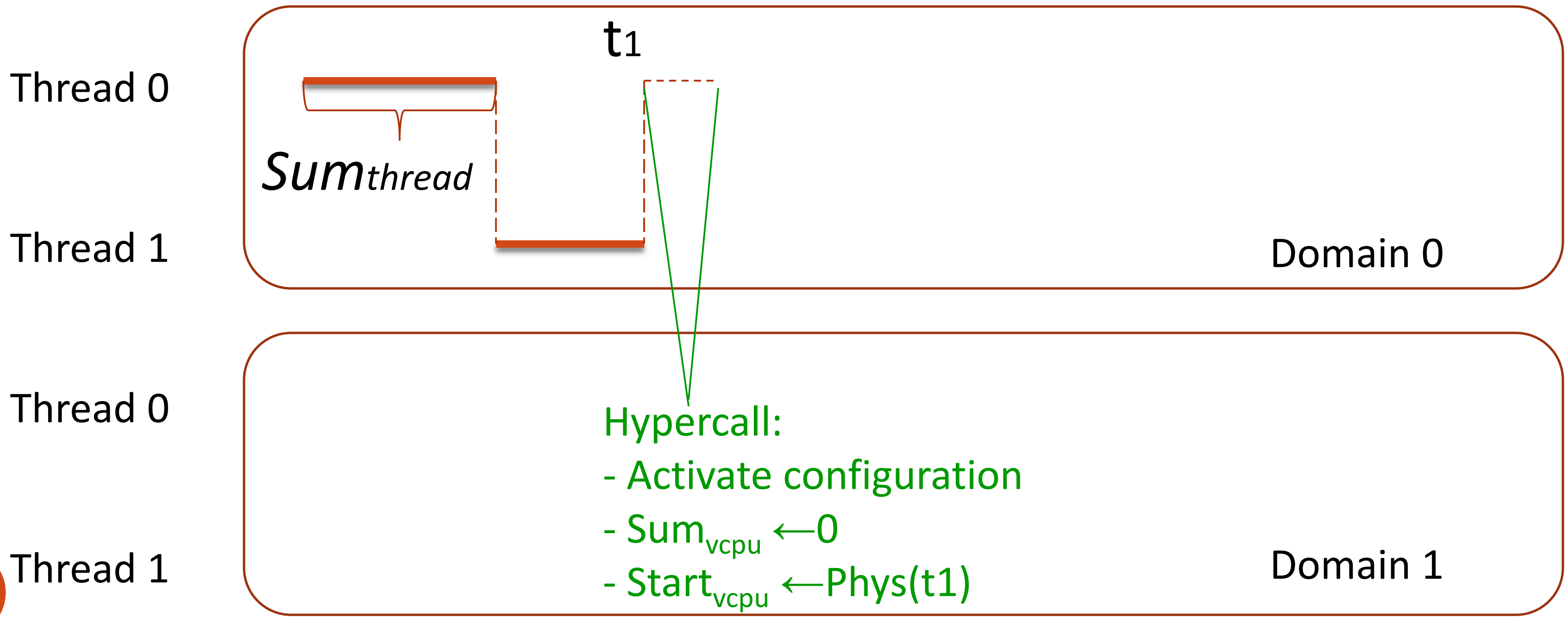
Perfctr-Xen: A-mode counters

Hypervisor: $Sum_{vcpu} = 0$, $Start_{vcpu} = Phys(t_1)$



Perfctr-Xen: A-mode counters

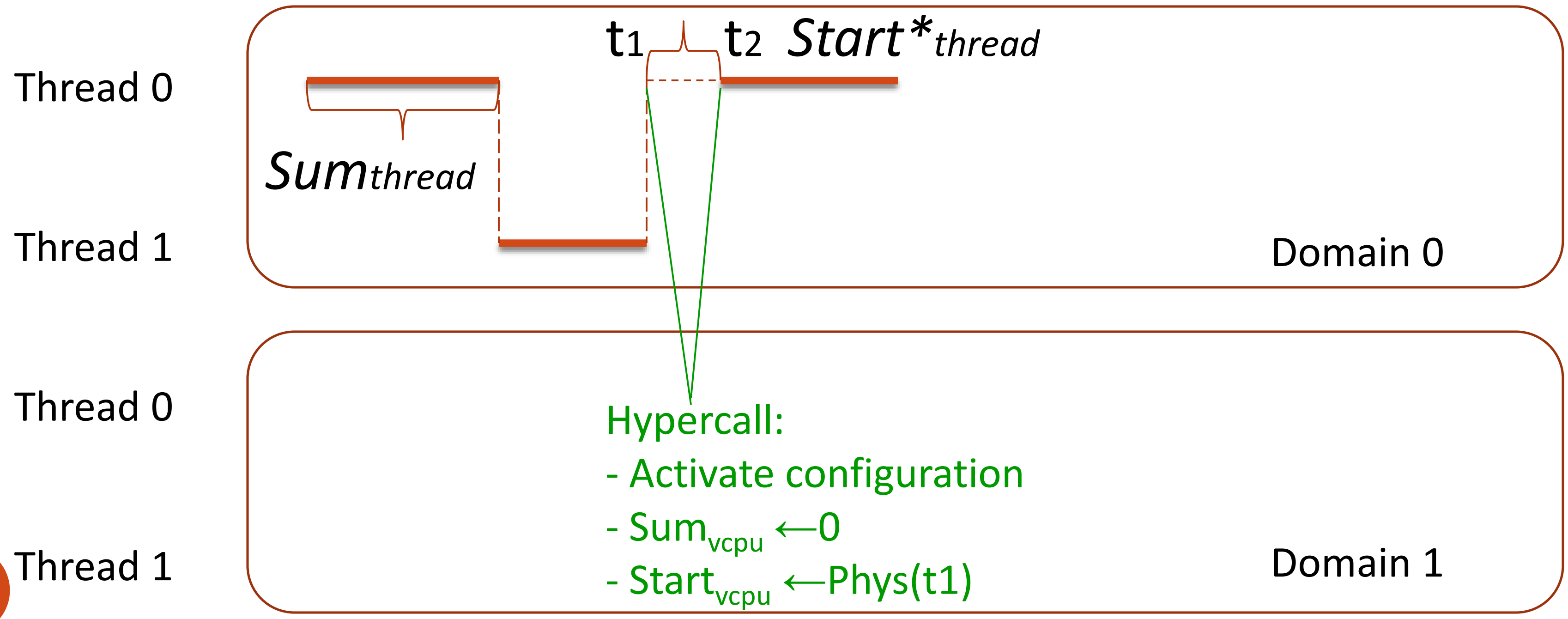
Hypervisor: Sum_{vcpu} = 0, $Start_{vcpu}$ = Phys(t_1)



Perfctr-Xen: A-mode counters

Hypervisor: $Sum_{vcpu} = 0$, $Start_{vcpu} = \text{Phys}(t_1)$

Guest: $Start^*_{thread} = Sum_{vcpu} + (\text{Phys}(t_2) - Start_{vcpu})$

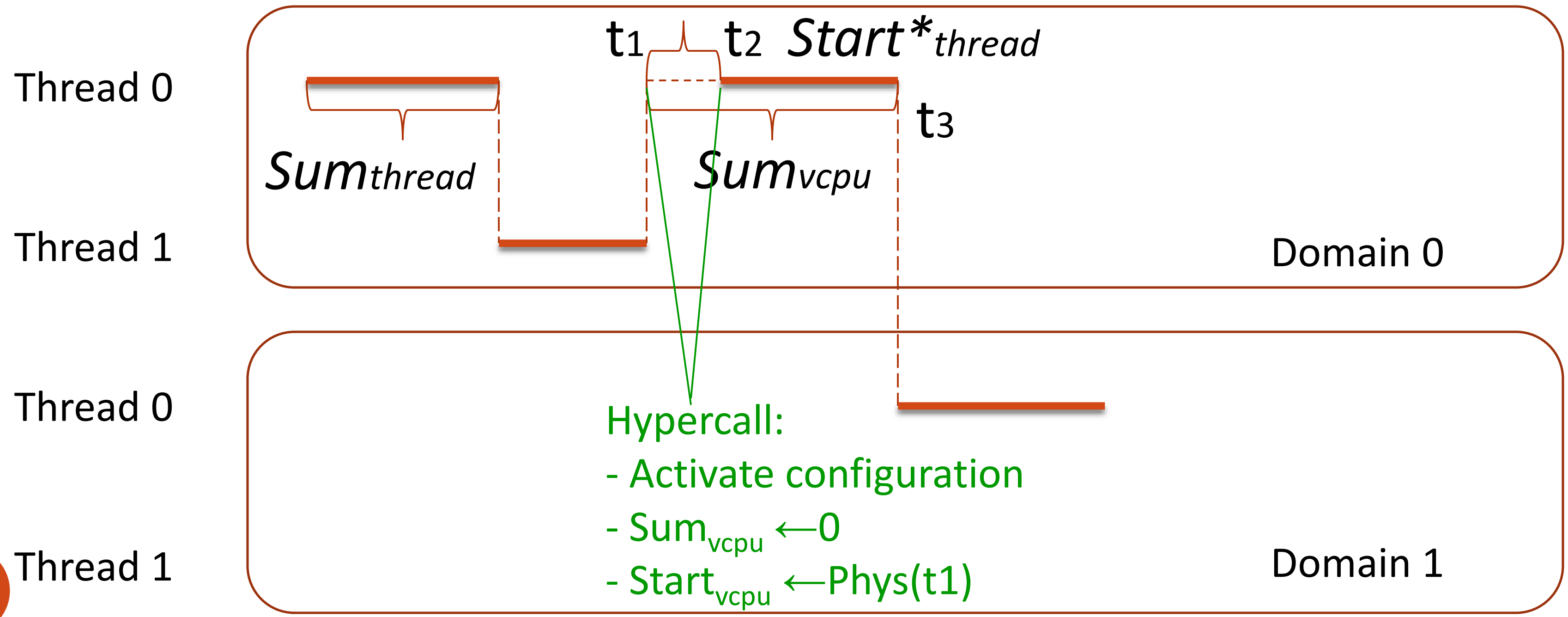


Perfctr-Xen: A-mode counters

Hypervisor: $Sum_{vcpu} = 0$, $Start_{vcpu} = Phys(t_1)$

Guest: $Start^*_{thread} = Sum_{vcpu} + (Phys(t_2) - Start_{vcpu})$

Hypervisor: $Sum_{vcpu} = Sum_{vcpu} + (Phys(t_3) - Start_{vcpu})$



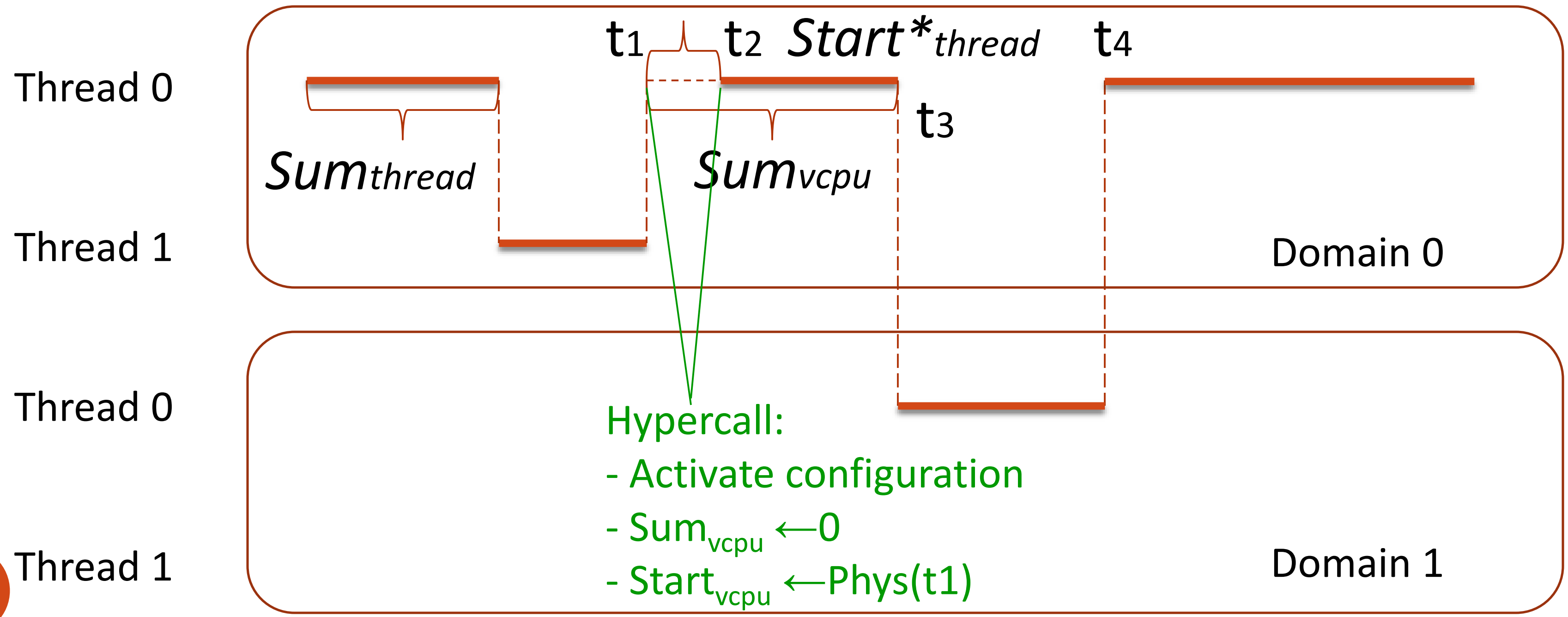
Perfctr-Xen: A-mode counters

Hypervisor: $Sum_{vcpu} = 0$, $Start_{vcpu} = Phys(t_1)$

Guest: $Start^*_{thread} = Sum_{vcpu} + (Phys(t_2) - Start_{vcpu})$

Hypervisor: $Sum_{vcpu} = Sum_{vcpu} + (Phys(t_3) - Start_{vcpu})$

Hypervisor: $Start_{vcpu} = Phys(t_4)$



Perfctr-Xen: A-mode counters

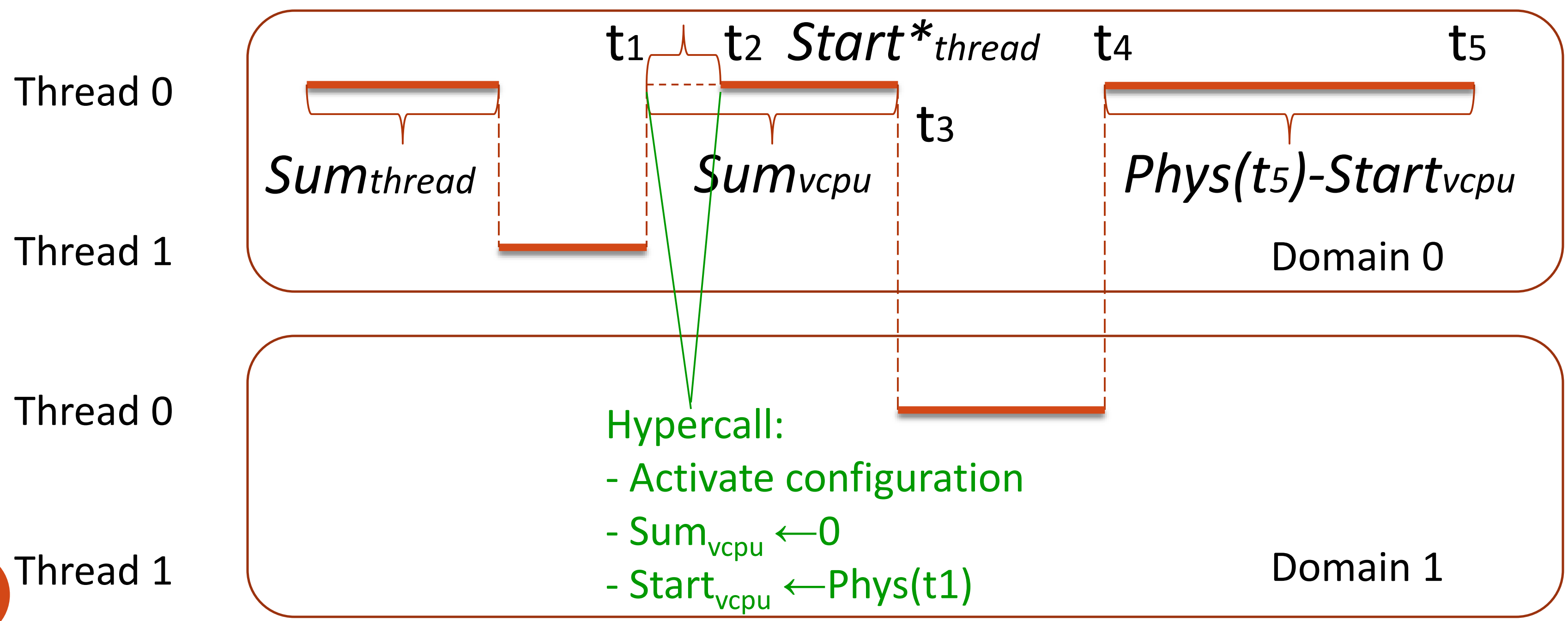
Hypervisor: $Sum_{vcpu} = 0$, $Start_{vcpu} = \mathrm{Phys}(t_1)$

Guest: $Start^*_{thread} = Sum_{vcpu} + (\mathrm{Phys}(t_2) - Start_{vcpu})$

Hypervisor: $Sum_{vcpu} = Sum_{vcpu} + (\mathrm{Phys}(t_3) - Start_{vcpu})$

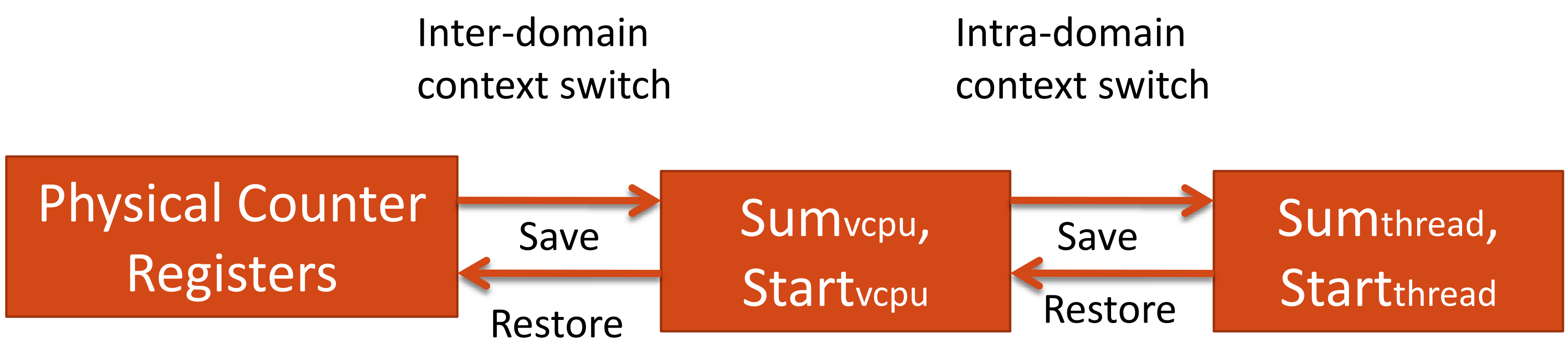
Hypervisor: $Start_{vcpu} = \mathrm{Phys}(t_4)$

$Log_{thread} = Sum_{thread} + Sum_{vcpu} + (\mathrm{Phys}(t_5) - Start_{vcpu}) - Start^*_{thread}$



Perfctr-Xen: I-mode counters

- Suspension hypercall to increment Sum_{vcpu} and sample $Start_{vcpu}$
- Resumption hypercall to restore per-VCPU values



$$Log_{thread} = Sum_{vcpu} + (\mathrm{Phys}(t) - Start_{vcpu})$$

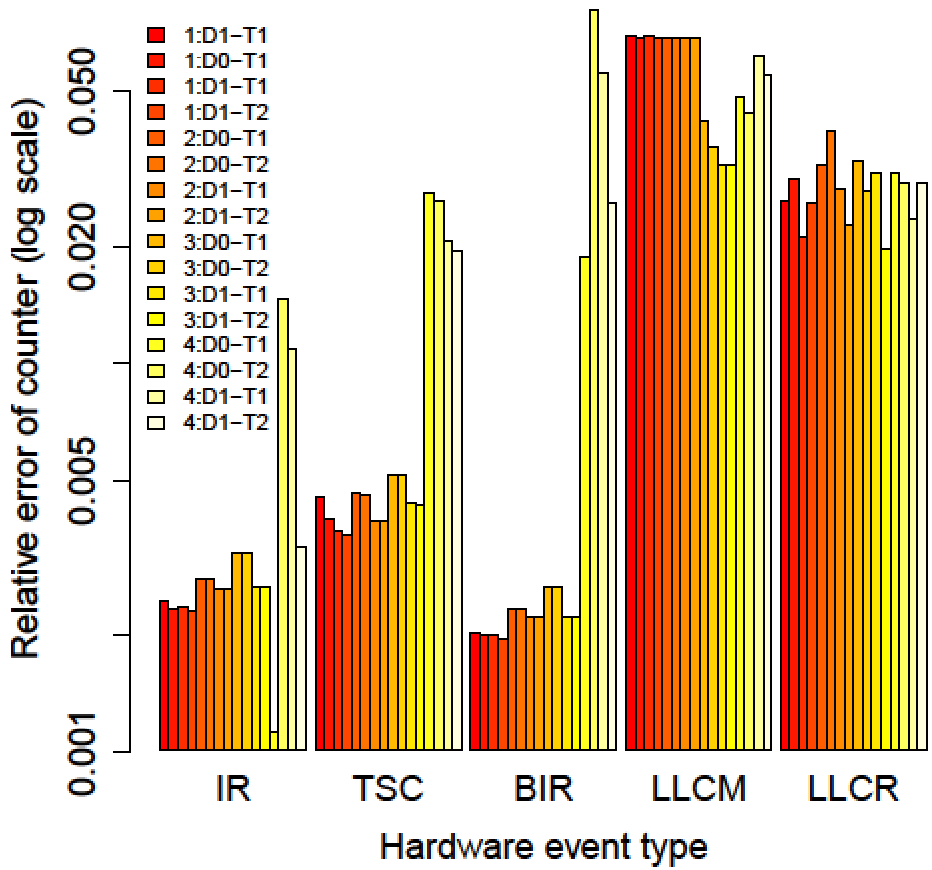
Perfctr-Xen: Interrupt delivery

- Hypervisor delivers overflow interrupts to guest via VIRQ_PERFCTR virtual interrupts
- Upon receipt, guest kernel signals user thread
- Virtual interrupts are delivered asynchronously (as soft interrupts)
- Guest must ensure that overflow interrupt is delivered to correct thread by rechecking overflow status
  - If thread causing overflow is suspended before virtual interrupt arrives at guest, mark as pending and deliver on next resume

Experimental Results

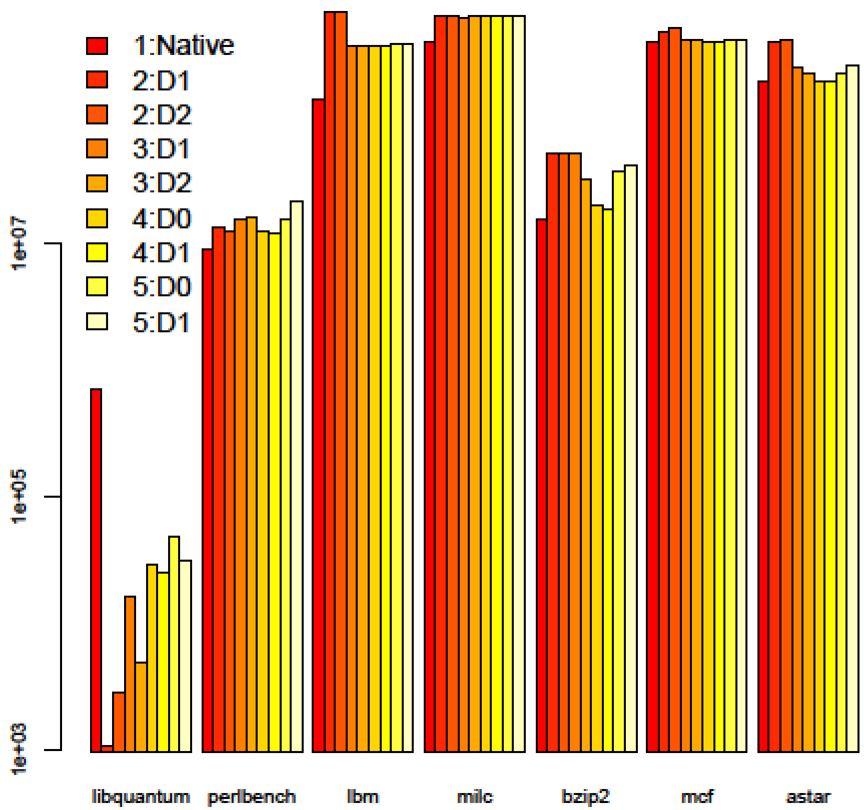
- Baseline: native execution
- Exercise multiple VCPU/PCPU scenarios
- Exercise multiple virtualization modes
  - Paravirtualization
  - Hardware-assisted virtualization (HVM)
  - Hybrid mode (HVM + guest enhancement)
- Correctness of implementation and accuracy of results
  - Microbenchmarks for a-mode, PAPI test for i-mode
  - Macrobenchmarks: SPEC CPU 2006
  - Verify Profiling (HPCToolkit)

Microbenchmarks



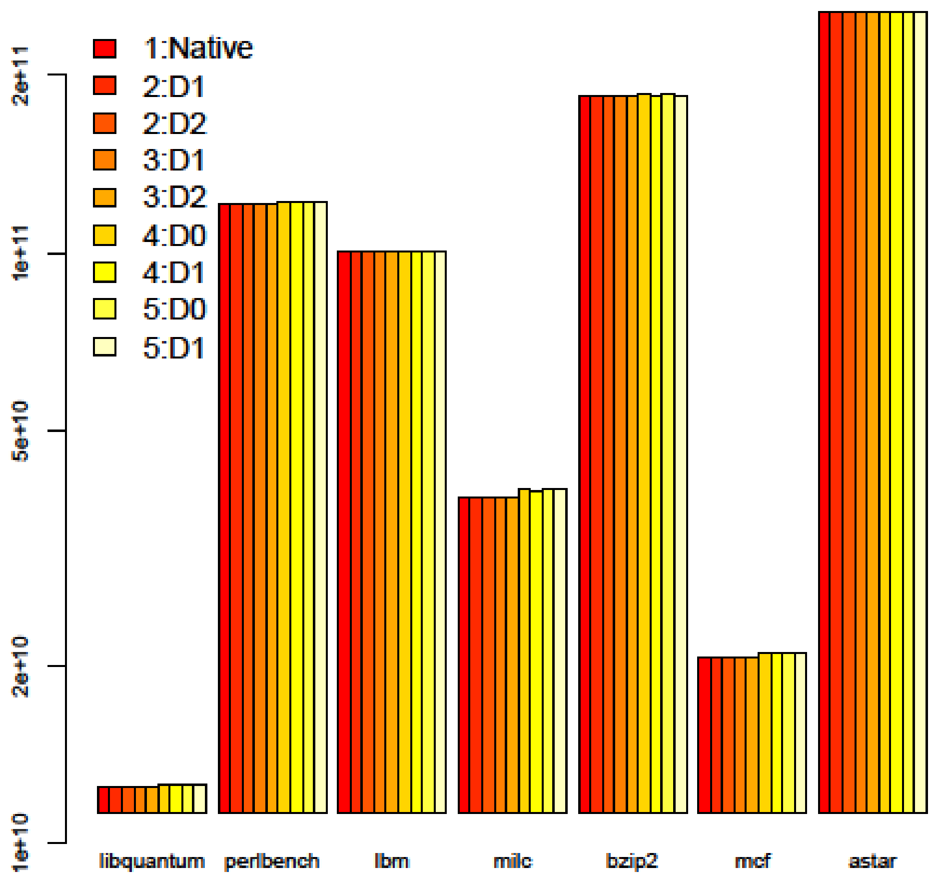
1. Each domain on 2 dedicated PCPUs; each thread on a dedicated VCPU.
2. Each domain on a dedicated PCPU; all threads in a domain on a shared VCPU.
3. All domains on a shared PCPU; all threads on a shared VCPU.
4. Random migration PCPUs and VCPUs

SPEC CPU2006: L2 Cache Misses



1. Native mode
2. Fully-virtualized Dom1 and Dom2, each on a dedicated core
3. Fully-virtualized Dom1 and Dom2 on the same core
4. Paravirtualized Dom0 and Dom1, each on a dedicated core
5. Paravirtualized Dom0 and Dom1 on the same core

SPEC CPU2006: L2 Cache References



1. Native mode
2. Fully-virtualized Dom1 and Dom2, each on a dedicated core
3. Fully-virtualized Dom1 and Dom2 on the same core
4. Paravirtualized Dom0 and Dom1, each on a dedicated core
5. Paravirtualized Dom0 and Dom1 on the same core

Related Work

- Performance counter support for VMM
  - XenoProf [Menon 2005]
  - Counter Virtualization for KVM [Du 2010, 2011]
  - VTSS++ system [Bratanov 2009]
- Performance counters in non-virtualized systems

  perf_counter, Perfmon [Eranian 2006], Intel VTune, AMD Code Analyst
- Higher-level libraries:
  - PAPI [Browne 1999]

Conclusion

- PerfCtr-Xen
  - Efficient and accurate per-thread virtualization of hardware event counters
  - Supports all commonly used virtualization modes
  - Plug-in Compatibility with PAPI, HPCToolkit, etc.
  - Techniques extend to other Type I hypervisors and low-level virtualization libraries

- Available at http://people.cs.vt.edu/~rnikola/

(LGPL license)