Real-time KVM from the ground up

LinuxCon NA 2016

Rik van Riel
Red Hat

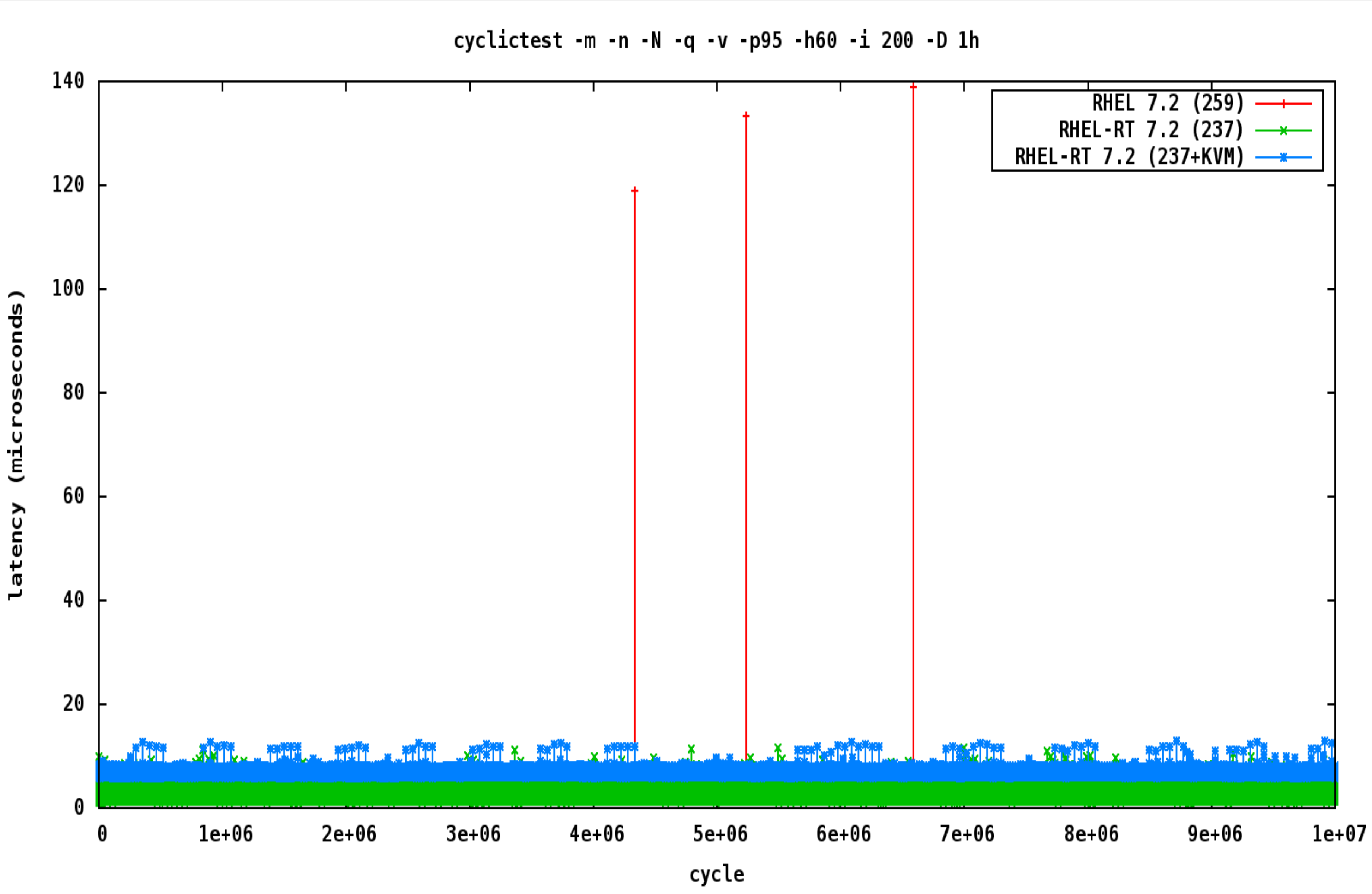
Real-time KVM

- What is real time?
- Hardware pitfalls
- Realtime preempt Linux kernel patch set
- KVM & qemu pitfalls
- KVM configuration
- Scheduling latency performance numbers
- Conclusions

What is real time?

- Real time is about determinism, not speed
- Maximum latency matters most
  - Minimum / average / maximum
- Used for workloads where missing deadlines is bad
  - Telco switching (voice breaking up)
  - Stock trading (financial liability?)
  - Vehicle control / avionics (exploding rocket!)
- Applications may have thousands of deadlines a second
- Acceptable max response times vary
  - For telco & stock cases, a few dozen microseconds
  - Very large fraction of responses must happen within that time frame (eg. 99.99%)

RHEL7.x Real-time Scheduler Latency Jitter Plot



Hardware pitfalls

- Biggest problems: BIOS, BIOS, and BIOS
- System Management Mode (SMM) & Interrupt (SMI)
  - Used to emulate or manage things, eg:
  - USB mouse PS/2 emulation
  - System management console
- SMM runs below the operating system
  - SMI traps to SMM, runs firmware code
- SMIs can take milliseconds to run in extreme cases
  - OS and real time applications interrupted by SMI
- Realtime may require BIOS settings changes
  - Some systems not fixable
  - Buy real time capable hardware
- Test with hwlatdetect & monitor SMI count MSR

Realtime preempt Linux kernel

- Normal Linux has similar latency issues as BIOS SMI
- Non-preemptible critical sections: interrupts, spinlocks, etc
- Higher priority program can only be scheduled after the critical section is over
- Real time kernel code has existed for years
  - Some of it got merged upstream
    - CONFIG_PREEMPT
  - Some patches in a separate tree
    - CONFIG_PREEMPT_RT
- https://rt.wiki.kernel.org/
- https://osadl.org/RT/

Realtime kernel overview

- Realtime project created a LOT of kernel changes
  - Too many to keep in separate patches
- Already merged upstream
  - Deterministic real time scheduler
  - Kernel preemption support
  - Priority Inheritance mutexes
  - High-resolution timer
  - Preemptive Read-Copy Update
  - IRQ threads
  - Raw spinlock annotation
  - NO_HZ_FULL mode
- Not yet upstream
  - Full realtime preemption

PREEMPT_RT kernel changes

- Goal: make every part of the Linux kernel preemptible
  - or *very* short duration
- Highest priority task gets to preempt everything else
  - Lower priority tasks
  - Kernel code holding spinlocks
  - Interrupts
- How does it do that?

PREEMPT_RT internals

- Most spinlocks turned into priority inherited mutexes
  - “spinlock” sections can be preempted
  - Much higher locking overhead
- Very little code runs with raw spinlocks
- Priority inheritance
  - Task A (prio 0), task B (prio 1), task C (prio 2)
  - Task A holds lock, task B running
  - Task C wakes up, wants lock
  - Task A inherits task C's priority, until lock is released
- IRQ threads
  - Each interrupt runs in a thread, schedulable
- RCU tracks tasks in grace periods, not CPUs
- Much, much more...

KVM & qemu pitfalls

- Real time is hard
- Real time virtualization is much harder

- Priority of tasks inside a VM are not visible to the host
  - The host cannot identify the VCPU with the highest priority program
- Host kernel housekeeping tasks extra expensive
  - Guest exit & re-entry
  - Timers, RCU, workqueues, …
- Lock holders inside a guest not visible to the host
  - No priority inheritance possible
- Tasks on VCPU not always preemptible due to emulation in qemu

Real time KVM kernel changes

- Extended RCU quiescent state in guest mode
- Add parameter to disable periodic kvmclock sync
  - Applying host ntp adjustments into guest causes latency
  - Guest can run ntpd and keep its own adjustment
- Disable scheduler tick when running a SCHED_FIFO task
  - Not rescheduling? Don't run the scheduler tick
- Add parameter to advance tscdeadline hrtime parameter
  - Makes timer interrupt happen “early” to compensate for virt overhead
- Various isolcpus= and workqueue enhancements
  - Keep more housekeeping tasks away from RT CPUs

Priority inversion & starvation

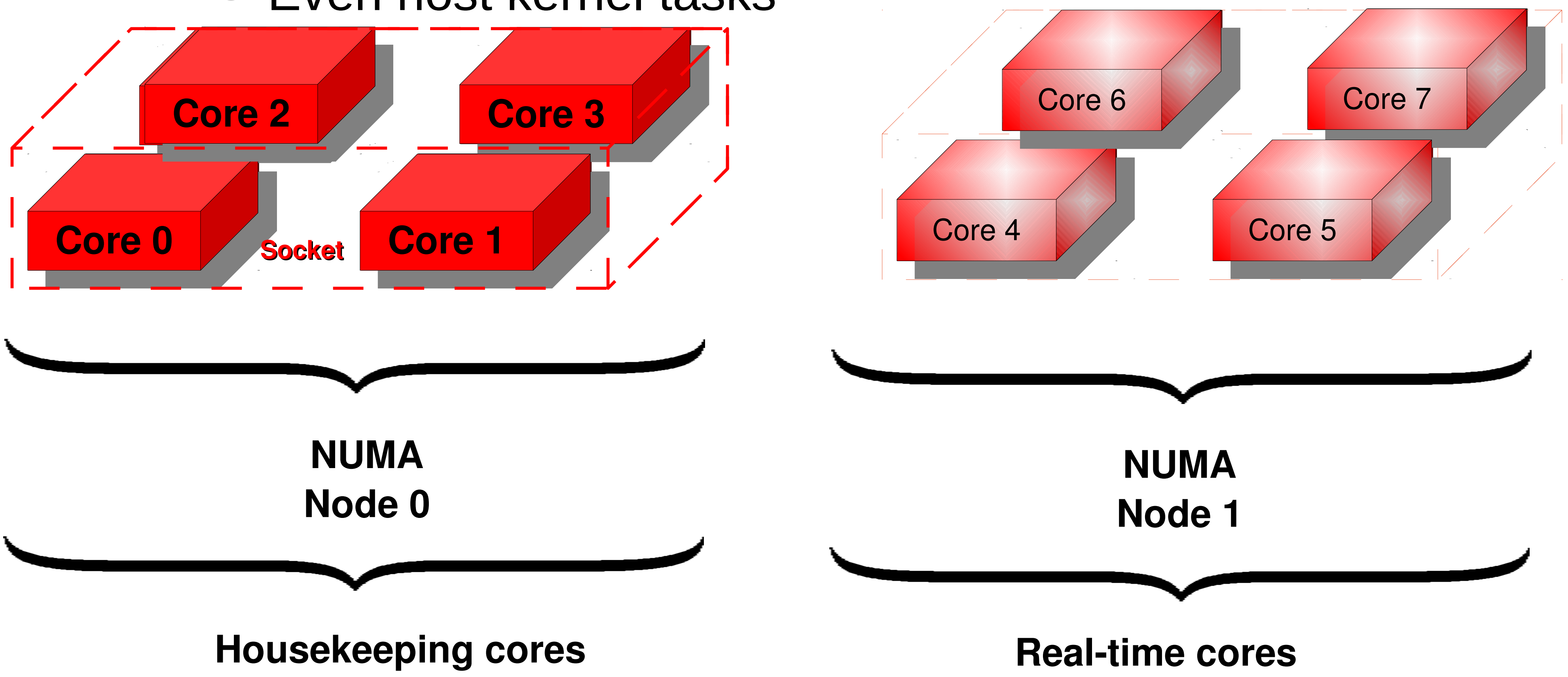
- Host & guest separated by clean(ish) abstraction layer
- VCPU thread needs a high real time priority on the host
  - Guarantee that real time app runs when it wants
- VCPU thread has same high real time host priority when running unimportant things...
- Guest could be run with idle=poll
  - VCPU uses 100% host CPU time, even when idle
- Higher priority things on the same CPU on the host are generally unacceptable – could interfere with real time task
- Lower priority things on the same CPU on the host could starve forever – could lead to system deadlock

KVM real time virtualization host partitioning

- Avoid host/guest starvation
  - Run VCPU threads on dedicated CPUs
  - No host housekeeping on those CPUs, except ksoftirqd for IPI & VCPU IRQ delivery
- Boot host with isolcpus and nohz_full arguments
- Run KVM guest VCPUs on isolated CPUs
- Run host housekeeping tasks on other CPUs

KVM real time virtualization host partitioning

- Run VCPUs on dedicated host CPUs
- Keep everything else out of the way
  - Even host kernel tasks

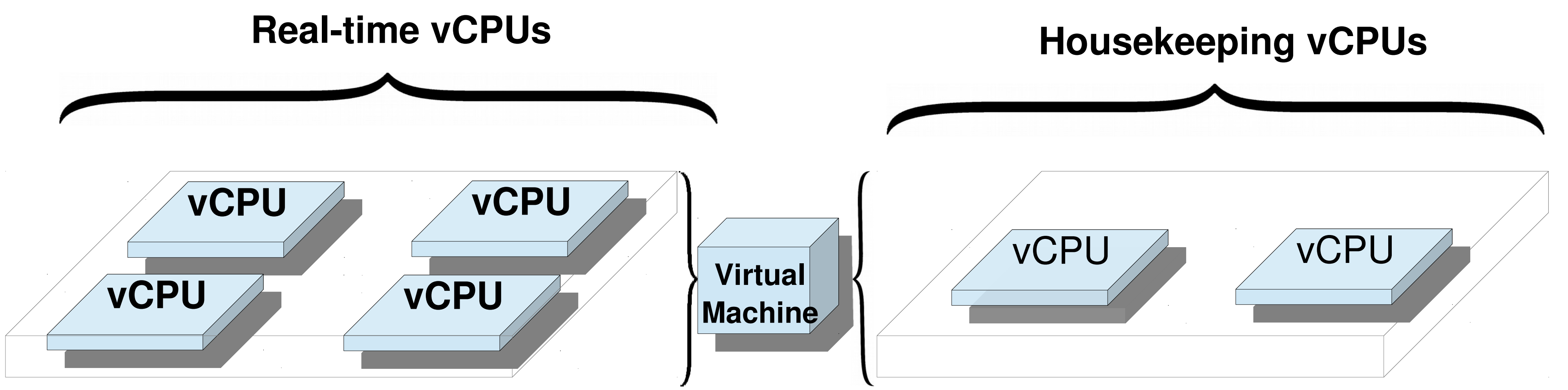


KVM real time virtualization guest partitioning

- Partitioning the host is not enough
- Tasks on guest can do things that require emulation
  - Worst case: emulation by qemu userspace on host
  - Poking I/O ports
  - Block I/O
  - Video card access
  - ...
- Emulation can take hundreds of microseconds
  - Context switch to other qemu thread
  - Potentially wait for qemu lock
  - Guest blocked from switching to higher priority task
- Guest needs partitioning, too!

KVM real time virtualization guest partitioning

- Guest booted with isolcpus
- Real time tasks run on isolated CPUs
- Everything else runs on system CPUs

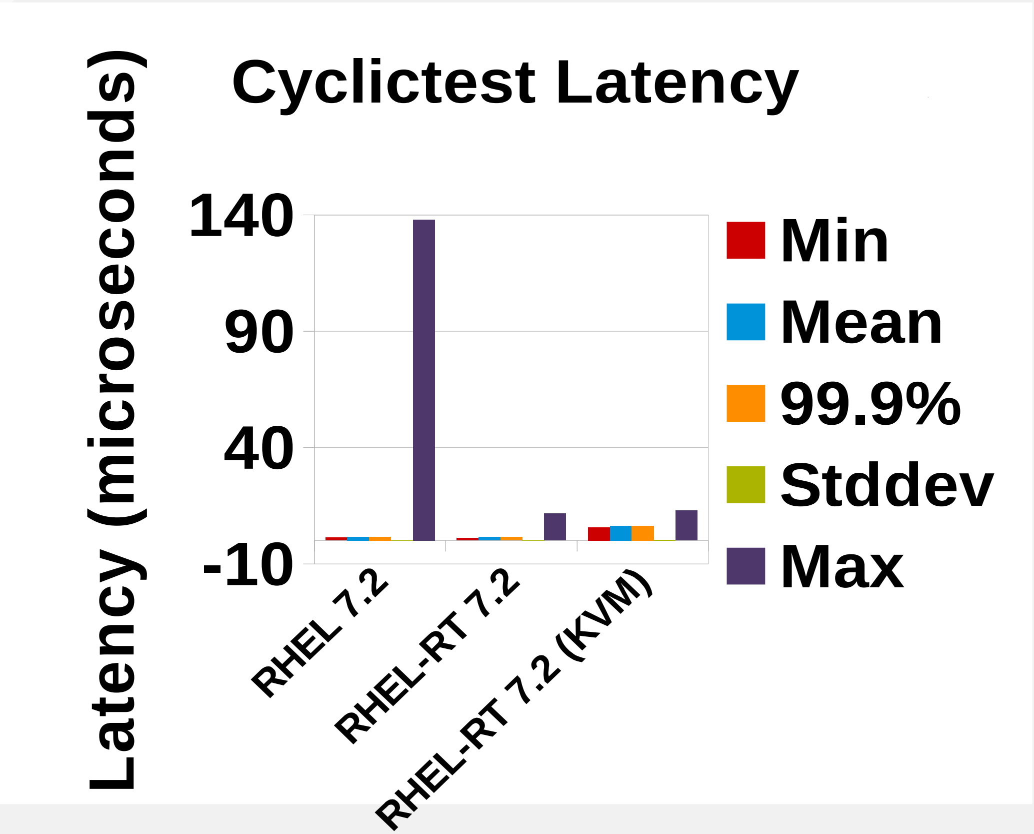


Real time KVM performance numbers

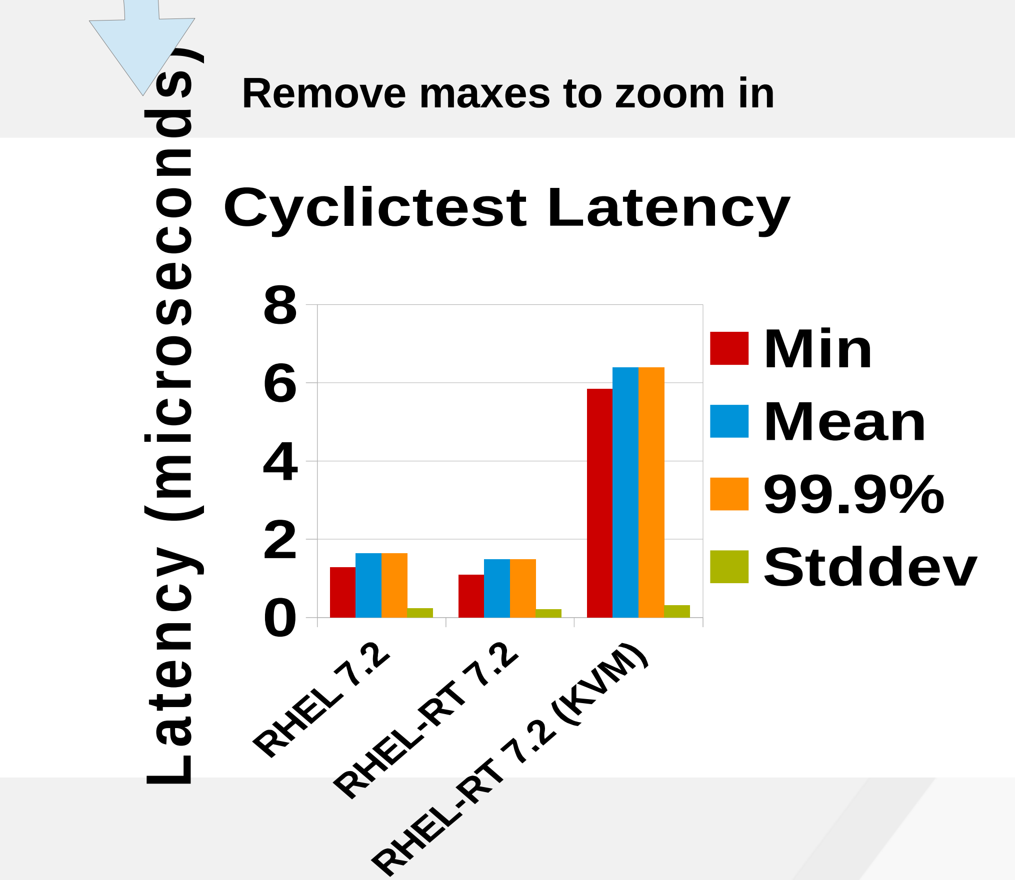
- Dedicated resources are ok
  - Modern CPUs have many cores
  - People often disable hyperthreading
- Scheduling latencies with cyclictest
  - Real time test tool
- Measured scheduling latencies inside KVM guest
  - Minimum: 5us
  - Average: 6us
  - Maximum: 14us

RHEL7.x Scheduler Latency (cyclictest)

Intel Ivy Bridge 2.4 Ghz, 128 GB mem



Remove maxes to zoom in



“Doctor, it hurts when I ...”

All kinds of system operations can cause high latencies

- CPU frequency change
- CPU hotplug
- Loading & unloading kernel modules
- Task migration between isolated and system CPUs
    - TLB flush IPI may get queued behind a slow op
    - Keep real time and system tasks separated
- Host clocksource change from TSC to !TSC
    - Use hardware with stable TSC
- Page faults or swapping
    - Run with enough memory
- Use of slow devices (eg. disk, video, or sound)
    - Only use fast devices from realtime programs
    - Slow devices can be used from helper programs

Cache Allocation Technology

- Single CPU can have many CPU cores, sharing L3 cache
- Cannot load lots of things from RAM in 14us
  - ~60ns for a single DRAM access
  - Uncached context switch + TLB loads + more could add up to >50us
- Low latencies depend on things being in CPU cache
- Latest Intel CPUs have Cache Allocation Technology
  - CPU cache “quotas”
  - Per application group, cgroups interface
  - Available on some Haswell CPUs
- Prevents one workload from evicting another workload from the cache
- Helps improve the guarantee of really low latencies

Future

- Use task isolation patches developed by Chris Metcalf?
- Change KVM to have guests run continuously on such fully isolated CPUs
- Let guests do some of their own CPU power saving (shallow c-states most of the time) on those CPUs?
- Enhance libvirt to have emulator threads run on different CPUs than the VCPU threads

Comparison with Jailhouse

- Jailhouse
  - Partitioning hypervisor
  - Typically used with assigned devices
  - Requires some custom setup
- KVM
  - Typically used as timeslicing hypervisor
  - Can be configured closer to partitioned system
    - Not perfect (yet), more kernel enhancements
    - Close enough for many uses
  - Can be managed through standard tools
    - Libvirt, OpenStack, Ovirt, etc
  - Real Time & normal hosts & guests managed with the same tools, as part of the same cloud

Conclusions

- Real time KVM is actually possible
  - Achieved largely through system partitioning
  - Overcommit is not an option
- Latencies low enough for various real time applications
  - 14 microseconds max latency with cyclictest
- Real time apps must avoid high latency operations
- Virtualization helps with isolation, manageability, hardware compatibility, …
- Requires very careful configuration
  - Can be automated with libvirt, openstack, etc