# As time goes by -

Analysing Where We Spend Our Cycles During Exits


Christian Bornträger

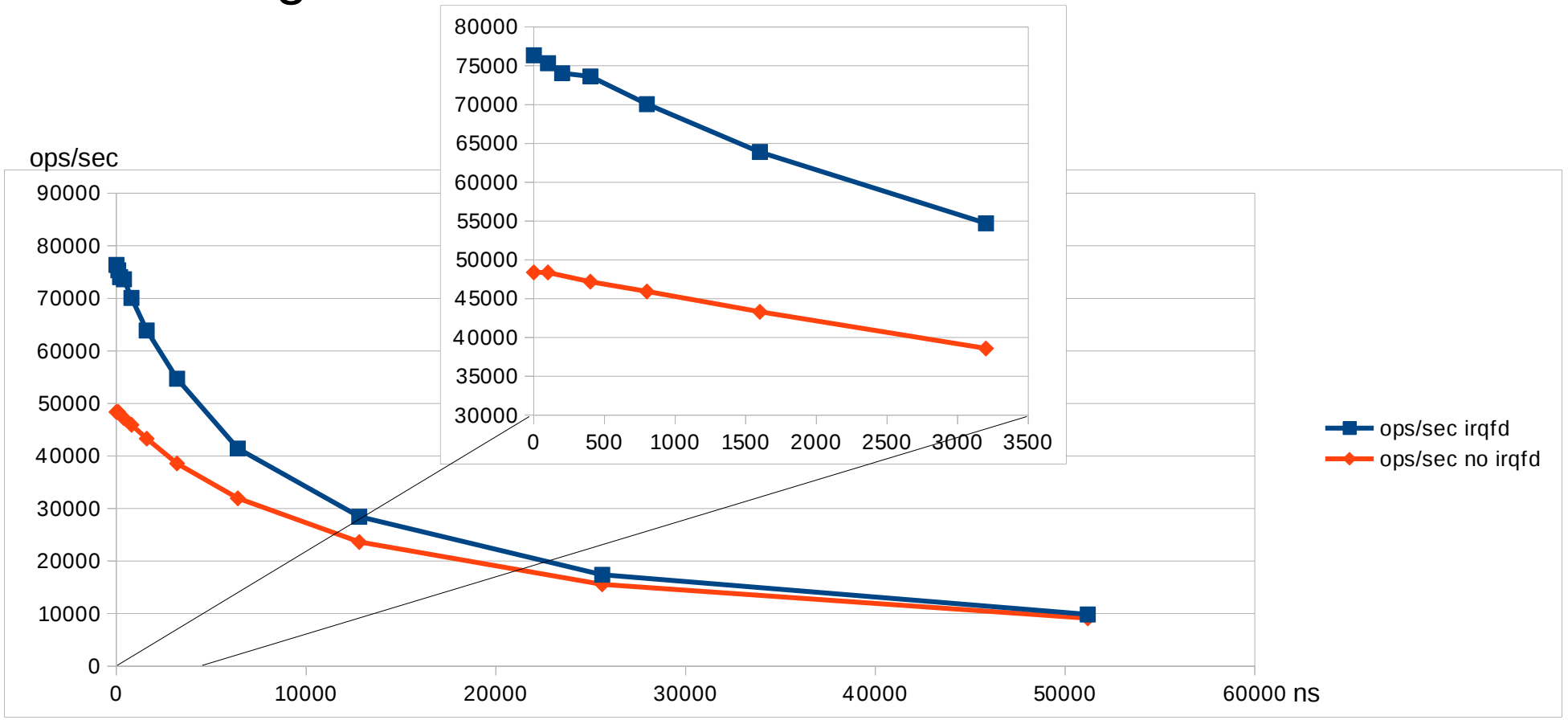
IBM Deutschland Research & Development GmbH
borntraeger@de.ibm.com
Co-maintainer KVM and
QEMU/KVM for s390x
(aka IBM zSystems)


# Motivation (1)

“People say that you should not micro-optimize. But if what you love is micro-optimization... that's what you should do.” - Linus Torvalds

# Motivation (2)

- Sample uperf TCP RR 1:1
- 1 byte tcp ping pong
- Let's see what happens when you add ndelay() after each guest exit



# What did I do?

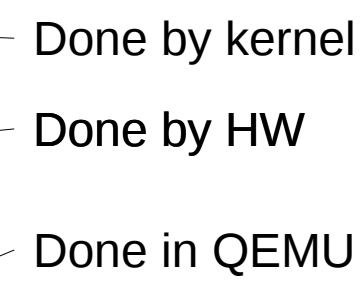
- Measuring how long it takes from guest->host->guest (guest exit handling exit and return to guest)
  - Initially with kernel module with timing and retries on s390
- Initial round trip time was 760ns (*)
  - Was surprised how far we exceed the HW overhead of entry/exit handling
  - Quickly identified several things down to 500ns
  - → Measuring and analysing can bring benefits very quickly for new architectures
- A lot more things after that
  - Fight against old code
  - Fight against new code
- Now at ~300ns (*) on my test system
  - Still much more than pure HW overhead

(*) Disclaimer: all numbers based on my as-is kernel config and my test systems (uncontrolled environment)

# Measuring 1/3

- Kvm-unit-tests
  - Available for most platforms
  - Gives times for typical exits
  - How long – not why (in cycles)

```
$ ./x86/run x86/vmexit.flat
[…]
cpuid 1552
vmcall 1448
mov_from_cr8 1
mov_to_cr8 15
inl_from_pmtimer 7220
inl_from_qemu 7002
[…]
```




- Kernel module for s390 as outlined

# Measuring 2/3

- So let's have a look at the why
  - Use ftrace!
  - Resolution for function tracer is microseconds

```
qemu-system-s39-4797  [000] ....   195.732618: kvm_s390_deliver_pending_inte...
```

  - Resolution for function graph is nanoseconds

```
0)   0.034 us    |          mutex_lock_killable();
```

  - Overhead > subject of measurement
    - Simple hypercall 300 ns → 1800 ns for function tracer
    - Simple hypercall 300 ns → 4800 ns for function_graph
    - Software uncertainty principle?
  - Still useful for finding interesting spots
  - Some functions (.s files) not prepared for ftrace :-/

# Measuring 3/3

- Use perf top/annotate
  - staring at samples in disassembly
  - Looking closer at hot samples
- Hand written “hacks”
- Disable “optional” code and retest

# History: early exits

- Request handling has many test_bits, clear_bits and memory barriers
- Requests are not the fast path, early exit if there are not requests
  - saves about 10ns for the common case on s390

```
 static int kvm_s390_handle_requests(struct kvm_vcpu *vcpu)
 {
[…]
+       if (!vcpu->requests)
+               return 0;
[…]
        if (kvm_check_request(KVM_REQ_MMU_RELOAD, vcpu)) {
[…]
```

- There was a followup idea from Paolo to pull this if into kvm_check_request such that gcc can optimize
  - I forgot about that
  - X86 and power already have a similar “if(vcpu->requests)”
  - Mips has only KVM_REQ_UNHALT
  - Arm sets KVM_REQ_VCPU_EXIT (but never checks?)

# History: irqsave/restore

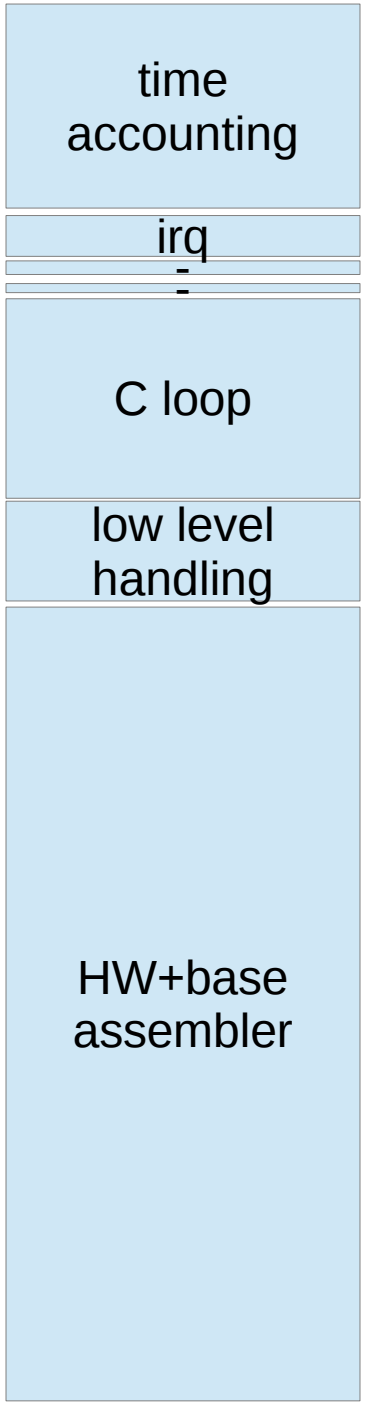
- irq_save/restore vs. irq_disable/enable
  - save/restore is about 5-8x slower than disable/enable on s390/x86
- How often when running KVM?
  - Around guest_enter_irqoff
  - rcu_note_context_switch might do it
  - Inside exit handlers
  - In scheduler code
- KVM now does disable/enable

# History: more s390 code

- S390 debug feature: pull condition check into header file
- S390 interrupt handling: do early exits
- Built-in vs. module
- Optimize irq_restore (ssm vs. stosm)

# Today

- Upstream s390 kernel, default config
- simple hypercall: ~300ns
- Lets start to remove code
  - Remove vtime_account_system: 255ns
    - About 50% arch code / 50% core code
  - + get rid of irq_disable_enable around guest_enter/exit: 246ns
  - + do not care about srcu locking 243ns
  - +get rid of tracing calls: 241ns
  - +shortcut in C (if special case just rerun the sie function): 197ns
  - +shortcut in assembler: 175 ns
- Still larger than pure HW time
  - possibly some misses/restarts in pipeline, caches, TLB and branch prediction
  - Still some code in hypervisor that needs to run



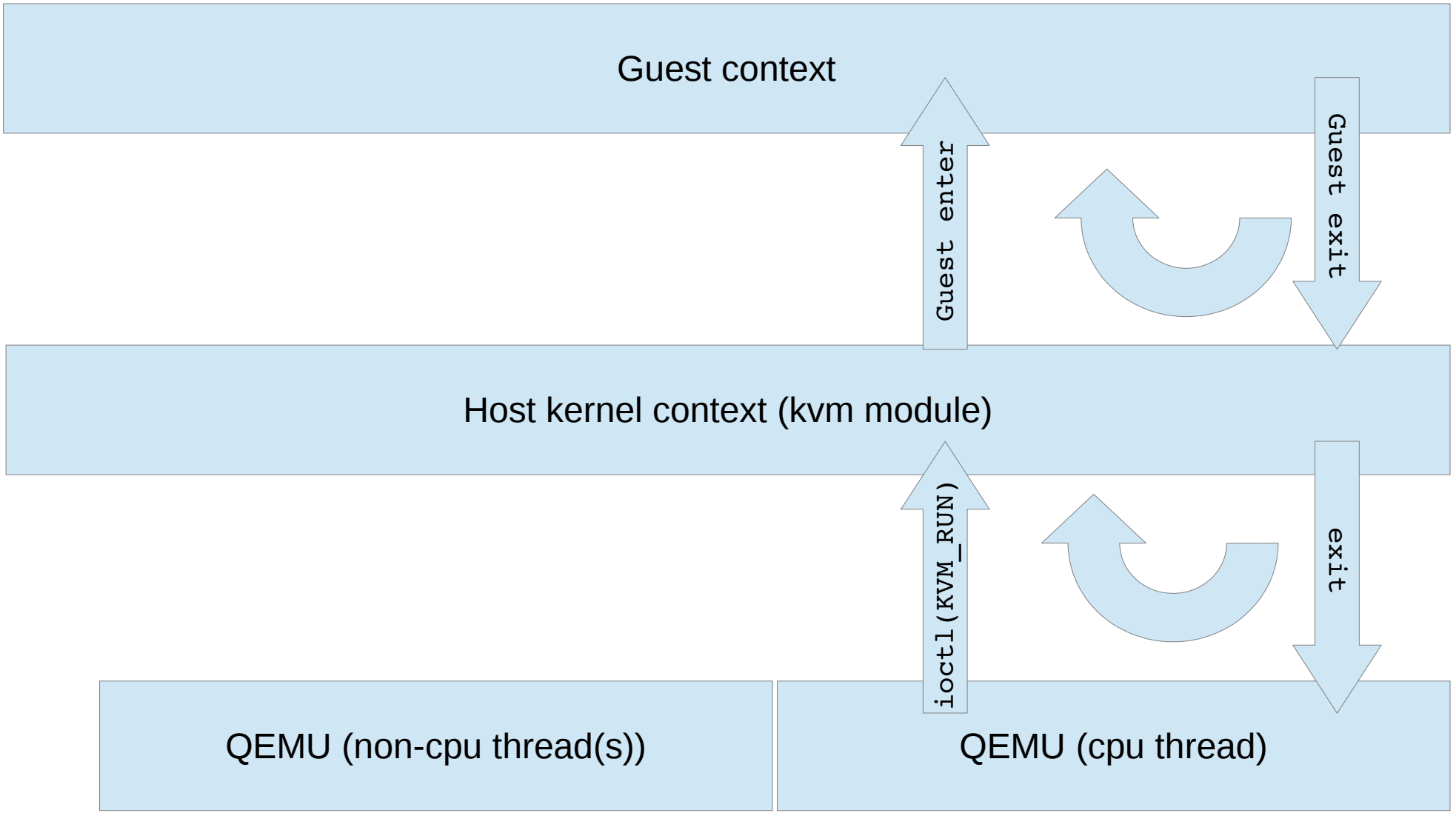
# QEMU

- Additional overhead of ~3000 cycles on x86 broadwell (~6000 on my ivy bridge)
  - Some things are known
    - Base overhead as seen before
    - signal mask restore
    - system call return
    - Glibc ioctl routine
    - KVM low level ioctl handling
    - KVM main loop
    - Glibc ioctl routine
    - system call enter
    - signal mask set
  - Some things can be hw related due to context change
    - Branch prediction
    - TLB
    - Caches
  - Some overhead due to horribly expensive things in QEMU

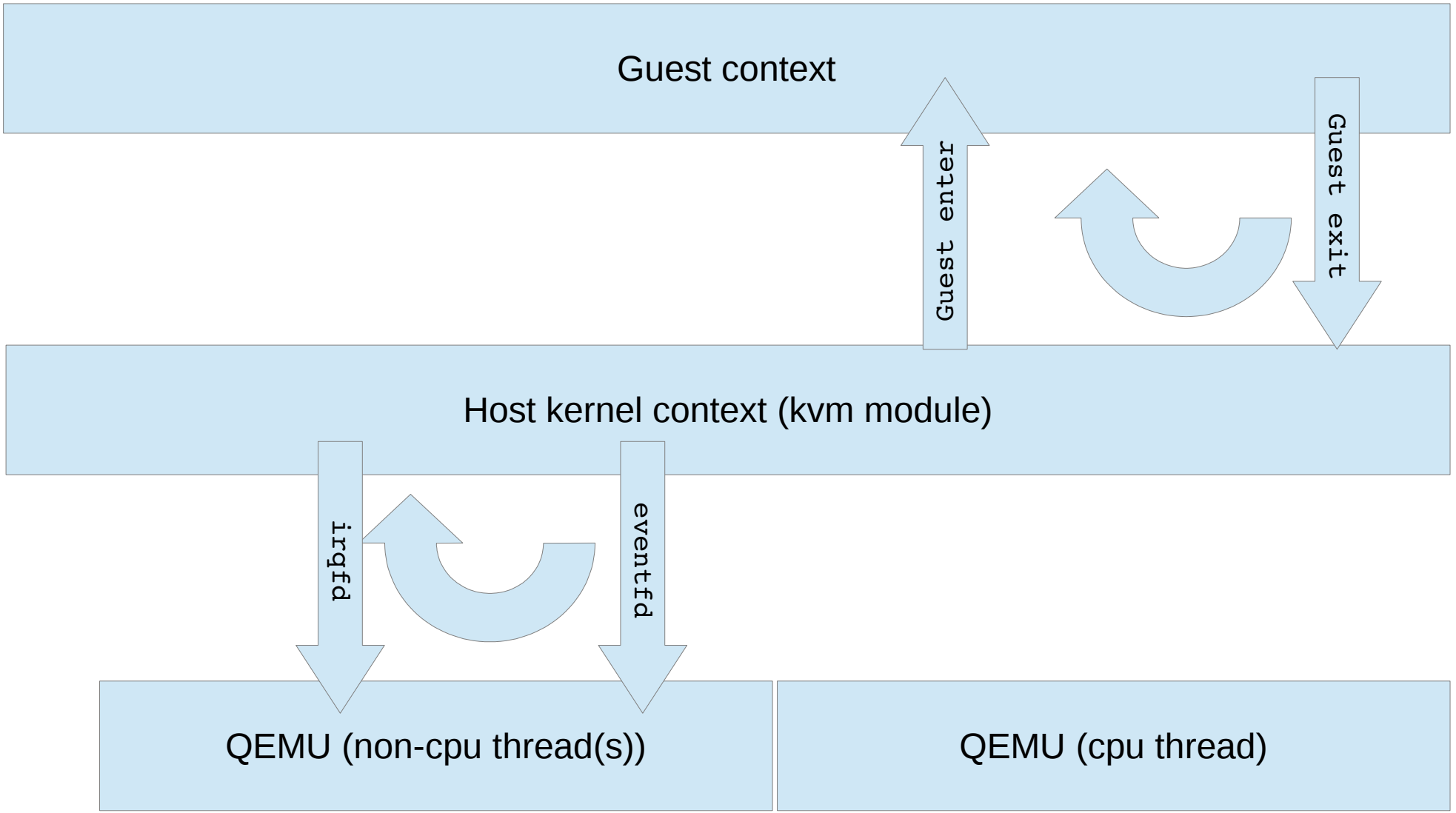
# QEMU

- Can we stay in the kernel for most exits?

# KVM exits


Guest context
Guest enter
Guest exit
Host kernel context (kvm module)
ioctl(KVM_RUN)
exit
QEMU (non-cpu thread(s))
QEMU (cpu thread)

# eventfd


Guest context
Guest enter
Guest exit
Host kernel context (kvm module)
irqfd
eventfd
QEMU (non-cpu thread(s))
QEMU (cpu thread)

# eventfd

- Using eventfd: 1400ns->400ns for the guest exit of the virtio kick
- Exit time seems to be constant, no matter how many devices (eventfd file descriptors) are being used → write to eventfd
- Performance (fio) also seems “flat”, as long as every disks has its own iothread

# vcpu_load vcpu_put

- With eventfd, most exits become lightweight exits
  - Can we avoid some things for lightweight exits?
  - The kernel does not use floating point
  - vcpu_load/vcpu_put
    - Floating point registers
    - Access registers
    - ...
  - Preempt notifier will ensure data integrity

# QEMU

- You said “Some overhead due to horribly expensive things in QEMU” on slide 12
  - Any examples?

# cpu_synchronize state

- cpu_synchronize_state when you need to read/write any CPU register state
  - call kvm_arch_get_registers
  - Schedules kvm_arch_put_registers
  - Two ioctl per register class (GET and SET)
    - KVM_GET_REGS
    - KVM_GET_XSAVE
    - KVM_GET_XCRS
    - KVM_GET_SREGS
    - KVM_GET_MSRS
    - KVM_GET_MP_STATE
    - KVM_GET_LAPIC
    - KVM_GET_PIT2
    - KVM_SET_REGS
    - KVM_SET_XSAVE
    - KVM_SET_XCRS
    - KVM_SET_SREGS
    - KVM_SET_MSRS
    - KVM_SET_MSRS
    - KVM_SET_PIT2

inl_from_qemu: 9183

inl_from_qemu: 59660

```
--- a/kvm-all.c
+++ b/kvm-all.c
@@ -1833,4 +1833,5 @@                    CPUState *cpu)

          run_ret = kvm_vcpu_ioctl(cpu, KVM_RUN, 0);
+         cpu_synchronize_state(cpu);
[…]
```

# Sync regs

- On s390 we often need one or the other register
  - Only one exit type (we would need one for each instructions)
  - We do call cpu_synchronize_state OFTEN
  - Why not use kvm_run as place for registers?

# Is this good enough?

- With all optimizations, arch_put/get_registers still visible in samples

```
for (i = 0; i < 32; i++) {
    cs->kvm_run->s.regs.vrs[i][0] = env->vregs[i][0].ll;
    cs->kvm_run->s.regs.vrs[i][1] = env->vregs[i][1].ll;
}
```

  - Due to aliasing rules and other things, gcc creates a loop with loads/stores instead of one big memcpy
  - Some cache effect as we are at the end of a context
- Long term solution could be to use access functions for registers
  - No mirroring necessary

# Object model

- Resolving an object is extremely expensive

inl_from_qemu: 9183

```
--- a/kvm-all.c
+++ b/kvm-all.c
@@ -1833,4 +1833,5 @@                    CPUState *cpu)

          run_ret = kvm_vcpu_ioctl(cpu, KVM_RUN, 0);
+         object_resolve_path_type("", TYPE_ACCEL, NULL);
[…]
```

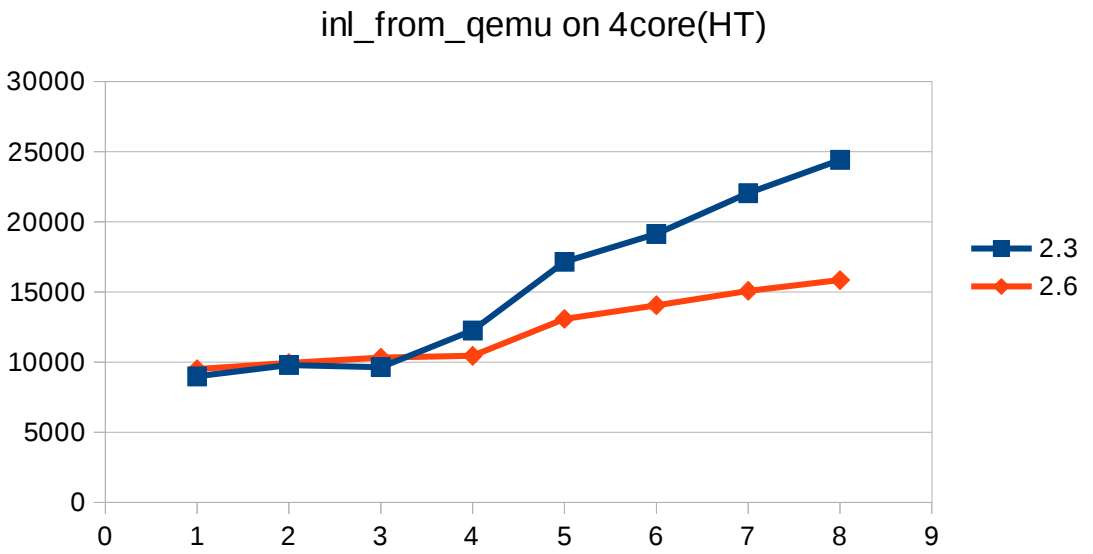
inl_from_qemu: 119780

# Big qemu lock

- Until QEMU 2.4, all KVM exits were handled serialized

```
qemu_mutex_unlock_iothread();
run_ret = kvm_vcpu_ioctl(cpu, KVM_RUN, 0);
qemu_mutex_lock_iothread()
```

- Pushdown efforts started in 2.4



# Future improvements

- Avoid exits
  - Use HW features
  - suggest HW features
  - Improve interfaces (e.g. virtio)
- On x86/s390 kernel offers only small potential
  - Request handling optimization in common code
  - Signal mask handling
- QEMU
  - Identify additional BQL pushdown areas
  - Understand object model cpu usage
  - Avoid/Optimize synchronize_state
  - Extend eventfd to other devices

# Fun facts

- Plugging in power cable in a Thinkpad W530 laptop improves exit times significantly even if the clock rate is the same
- Found 2 bugs in the s390 code while preparing these slides


IBM
Thank you!
ibm.com/linux
IBM

# Trademarks

**The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.**

| | | | | | | |
|---|---|---|---|---|---|---|
| DB2* | ECKD | IBM* | LinuxONE | PR/SM | z13 | z Systems |
| DB2 Connect | FICON* | Ibm.com | LinuxONE Emperor | Storwize* | zEnterprise* | z/VSE* |
| DS8000* | FlashSystem | IBM (logo)* | LinuxONE Rockhopper | XIV* | z/OS* | z/VM* |

* Registered trademarks of IBM Corporation

**The following are trademarks or registered trademarks of other companies.**

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.
Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.
Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.
ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.
Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.
Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
OpenStack is a trademark of OpenStack LLC. The OpenStack trademark policy is available on the OpenStack website.
TEALEAF is a registered trademark of Tealeaf, an IBM Company.
Windows Server and the Windows logo are trademarks of the Microsoft group of countries.
Worklight is a trademark or registered trademark of Worklight, an IBM Company.
UNIX is a registered trademark of The Open Group in the United States and other countries.

* Other product and service names might be trademarks of IBM or other companies.

**Notes**:
Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.
IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.
All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.
This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.
All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.
Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.
Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.
This information provides only general descriptions of the types and portions of workloads that are eligible for execution on Specialty Engines (e.g, zIIPs, zAAPs, and IFLs) ("SEs"). IBM authorizes customers to use IBM SE only to execute the processing of Eligible Workloads of specific Programs expressly authorized by IBM as specified in the “Authorized Use Table for IBM Machines” provided at www.ibm.com/systems/support/machine_warranties/machine_code/aut.html (“AUT”). No other workload processing is authorized for execution on an SE. IBM offers SE at a lower price than General Processors/Central Processors because customers are authorized to use SEs only to process certain types and/or amounts of workloads as specified by IBM in the AUT.