


# Integrated Testing in QEMU

## An overview of qtest and qemu-test

**Anthony Liguori – aliguori@us.ibm.com**

**IBM Linux Technology Center**

***Aug 2010***



# But testing is a solved problem...

(What about KVM autotest)

# Continuous Growth

QEMU sees ~50% annual growth rate in commits

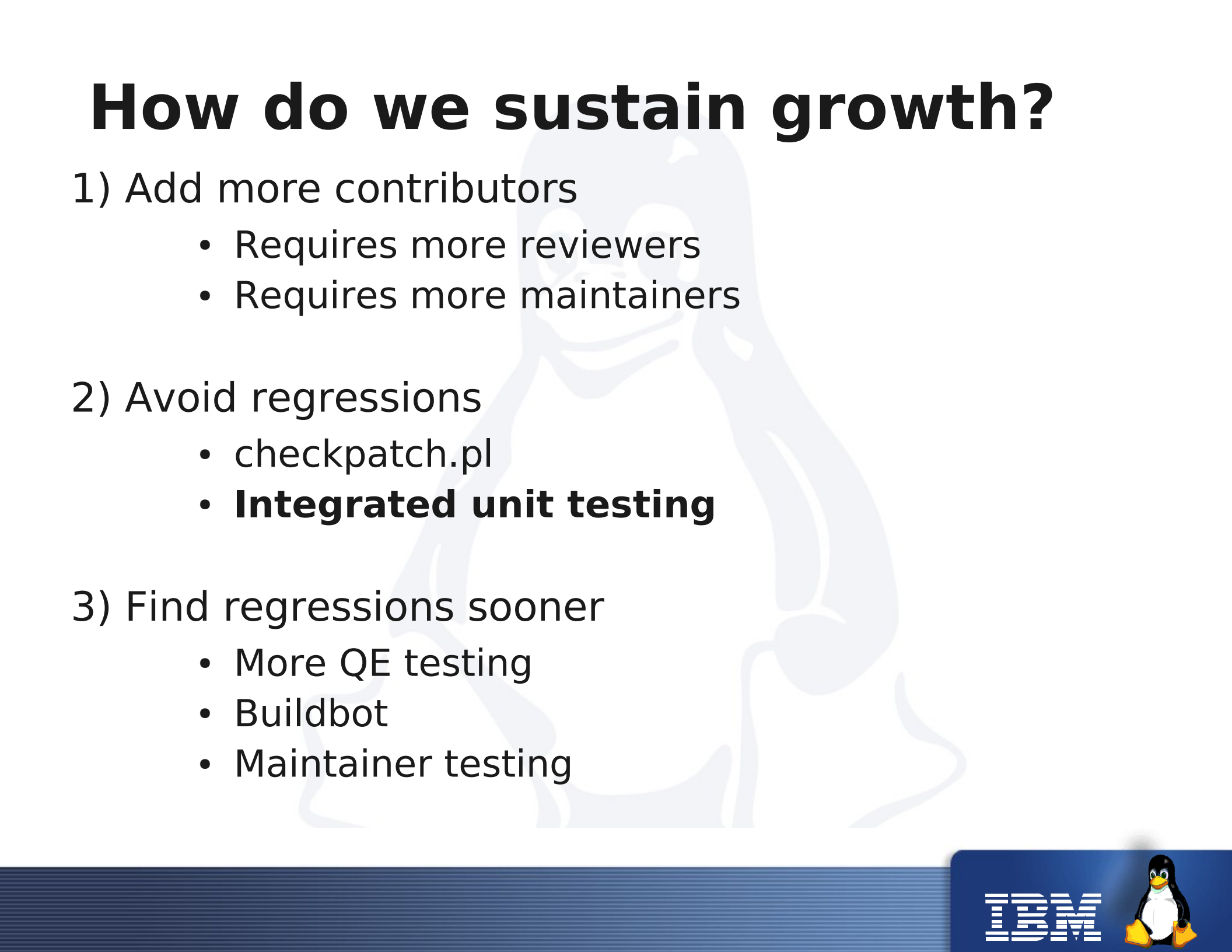
```
Total Physical Source Lines of Code (SLOC)                = 1,227,795
Development Effort Estimate, Person-Years (Person-Months) = 350.44 (4,205.27)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                         = 4.96 (59.56)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  = 70.60
Total Estimated Cost to Develop                           = $ 47,339,512
 (average salary = $56,286/year, overhead = 2.40).
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
SLOCCount is Open Source Software/Free Software, licensed under the GNU GPL.
SLOCCount comes with ABSOLUTELY NO WARRANTY, and you are welcome to
redistribute it under certain conditions as specified by the GNU GPL license;
see the documentation for details.
Please credit this data as "generated using David A. Wheeler's 'SLOCCount'."
```

# How do we sustain growth?

1) Add more contributors
   - Requires more reviewers
   - Requires more maintainers

2) Avoid regressions
   - checkpatch.pl
   - **Integrated unit testing**

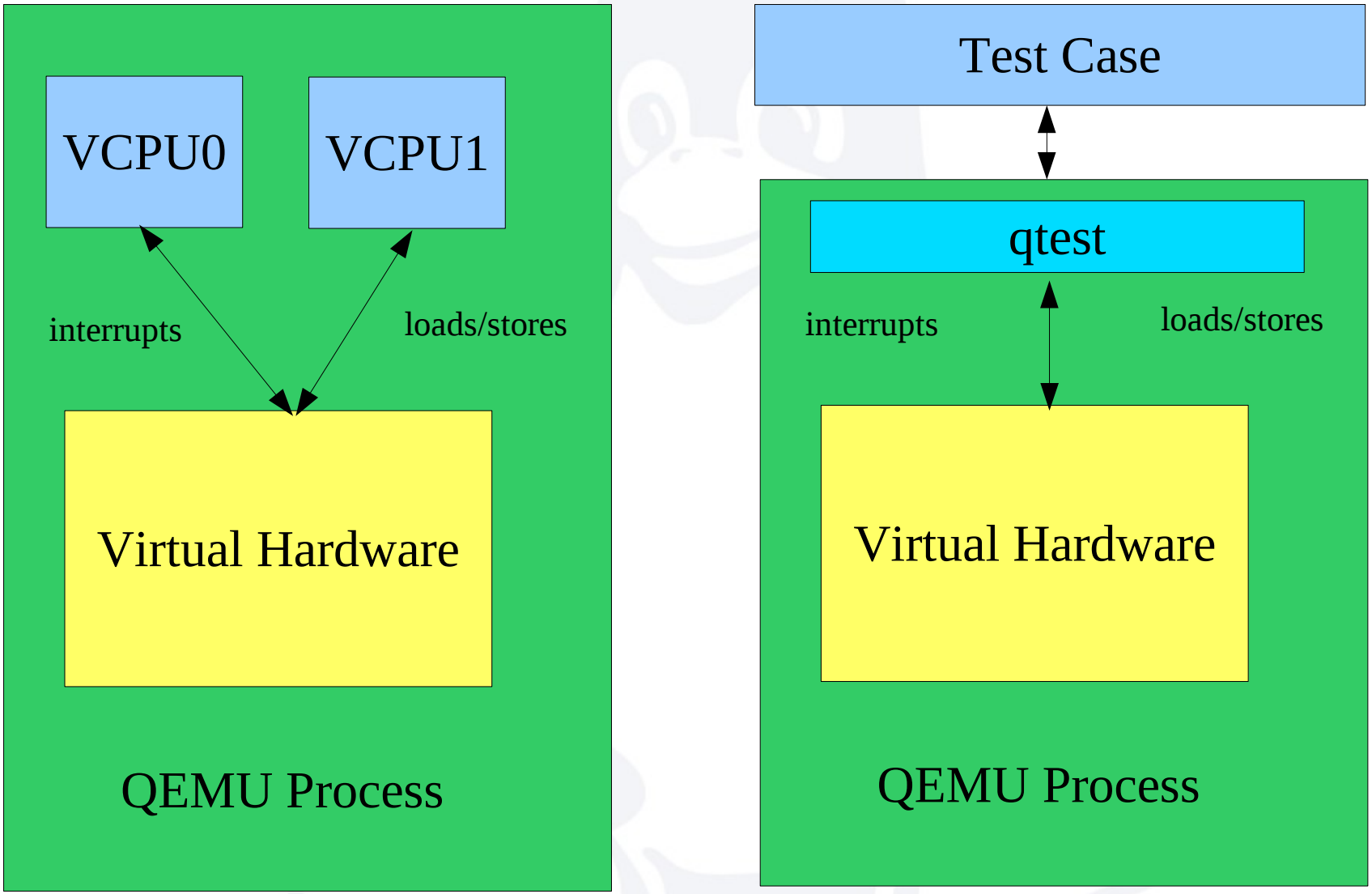
3) Find regressions sooner
   - More QE testing
   - Buildbot
   - Maintainer testing



# Rules of Unit Testing

- Developer convenience is top priority
  - If tests are hard to run, they won't be run
- Fitting a developer's work flow
  - Must not make permanent changes
  - Must not take a long time to setup
  - Must run quickly
  - Must not require root privileges
- Simplicity
  - Should require close to zero setup

# qtest


VCPU0
VCPU1
interrupts
loads/stores
Virtual Hardware
QEMU Process
Test Case
qtest
interrupts
loads/stores
Virtual Hardware
QEMU Process

# qtest overview

- All VCPU ↔ hardware communications happens over domain socket
  - PIO, MMIO, interrupts supported today
  - Extensible to hypercalls
  - Commands to control vm_clock progress
- Simple line-based protocol
  - Replay support?
- Test case runs as a separate process
  - Written in C compiled natively
  - Full access to libc

# Why not run guest code?

- Guest code needs cross compilers to build
  - Inconvenient
- Requires infrastructure to communicate test results
  - qtest uses gtester
- Challenging to test certain resources
  - Difficult to debug serial port if serial port is used for logging

# qtest in action

```
int main(int argc, char **argv)
{
    QTestState *s = NULL;
    int ret;

    g_test_init(&argc, &argv, NULL);

    s = qtest_start("-display none -rtc clock=vm");
    qtest_irq_intercept_in(s, "ioapic");

    qtest_add_func("/rtc/bcd/check-time", bcd_check_time);

    ret = g_test_run()
    if (s) {
        qtest_quit(s);
    }

    return ret;
}
```

# qtest in action

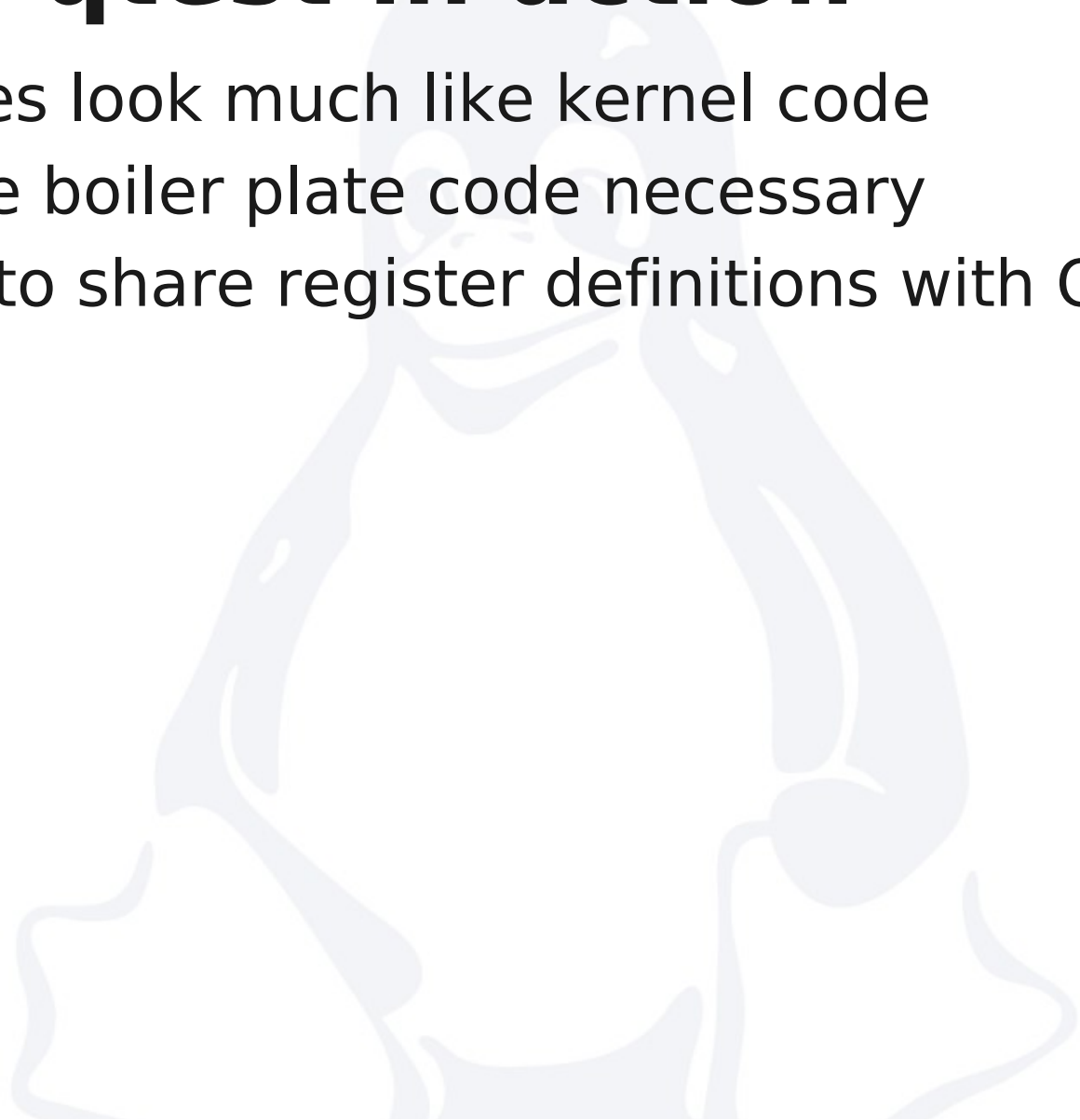
```
static uint8_t cmos_read(uint8_t reg)
{
    outb(base + 0, reg);
    return inb(base + 1);
}

static void cmos_write(uint8_t reg, uint8_t val)
{
    outb(base + 0, reg);
    outb(base + 1, val);
}

static void bcd_check_time(void)
{
    /* Set BCD mode */
    cmos_write(RTC_REG_B, cmos_read(RTC_REG_B) & ~REG_B_DM);
    check_time();
}
```

# qtest in action

- Test cases look much like kernel code
- Very little boiler plate code necessary
- Possible to share register definitions with QEMU



# libqos

- Simple test cases can be written with inb/outb
- More complicated devices require interaction with PCI bus and APIC

- libqos provides a mini-OS library for writing qtest cases
  - Meant to be reasonably portable

- PCI support
- Memory allocator

# libqos PCI

```
void qpci_device_foreach(QPCIBus *bus, int vendor_id, int device_id,
                         void (*func)(QPCIDevice *dev, int devfn, void *data),
                         void *data);
QPCIDevice *qpci_device_find(QPCIBus *bus, int devfn);

void qpci_device_enable(QPCIDevice *dev);
void qpci_device_disable(QPCIDevice *dev);

uint8_t qpci_config_readb(QPCIDevice *dev, uint8_t offset);
void qpci_config_writeb(QPCIDevice *dev, uint8_t offset, uint8_t value);
uint8_t qpci_io_readb(QPCIDevice *dev, void *data);
void qpci_io_writeb(QPCIDevice *dev, void *data, uint8_t value);
…

void *qpci_iomap(QPCIDevice *dev, int barno);
void qpci_iounmap(QPCIDevice *dev, void *data);
```

# libqos allocator

```
uint64_t guest_alloc(QGuestAllocator *allocator, size_t size);
void guest_free(QGuestAllocator *allocator, uint64_t addr);

void qtest_memread(QTestState *s, uint64_t addr, void *data, size_t size);
void qtest_memwrite(QTestState *s, uint64_t addr, const void *data, size_t size);
```

# libqos in action

```
cmdline = g_strdup_printf("-display none "
                          "-device virtio-blk-pci,drive=hd0,addr=04.0 "
                          "%s ",
                          block_info);

qs = qtest_start(cmdline);

pci_bus = qpci_init_pc();
ga = pc_alloc_init();

dev = qpci_device_find(pci_bus, QPCI_DEVFN(4, 0));
g_assert(dev != NULL);

bar0 = qpci_iomap(dev, 0);
qpci_device_enable(dev);

host_features = qpci_io_readl(dev, bar0);
g_assert(host_features & (1 << VIRTIO_BLK_F_SEG_MAX));
g_assert(host_features & (1 << VIRTIO_BLK_F_GEOMETRY));
g_assert(!(host_features & (1 << VIRTIO_BLK_F_RO)));
...
```

# qemu-test

- libqos has limitations
  - Unreasonable to write an AML interpreter for qtest
  - Impractical to test some very complex devices
- qemu-test attempts to use the Linux kernel as a “libos”
- Build system generates a Linux kernel + busybox environment
- Bootstraps cross compilers

# qemu-test Considerations

- Must be fully boot strapping
  - To compile with GPL
  - Desire to host binaries on qemu.org
- Must launch quickly
  - Rules out most distributions
- Test cases should be simple to write
  - Use shell script in host/guest
- Should have access to QMP
  - Ability to validate hotplug

# qemu-test in action

```
#!/bin/sh

in_host() {
    nic=`named_choose nic tier2 rtl8139 e1000 virtio`
    if test "$nic" = "tier2"; then
#   nic=`named_choose nic.tier2 ne2k_pci i82551 i82557b i82559er pcnet`
    nic=`named_choose nic.tier2 ne2k_pci i82551 i82557b i82559er`
    fi
    echo "Using networking card: $nic"
    qemu -nographic -enable-kvm -net user -net nic,model=$nic
}

in_guest() {
    udhcpc -i eth0 -f -n -q
    wget -O /dev/null http://www.google.com
}

if test $QEMU_TEST; then
    in_host
else
    in_guest
fi
```

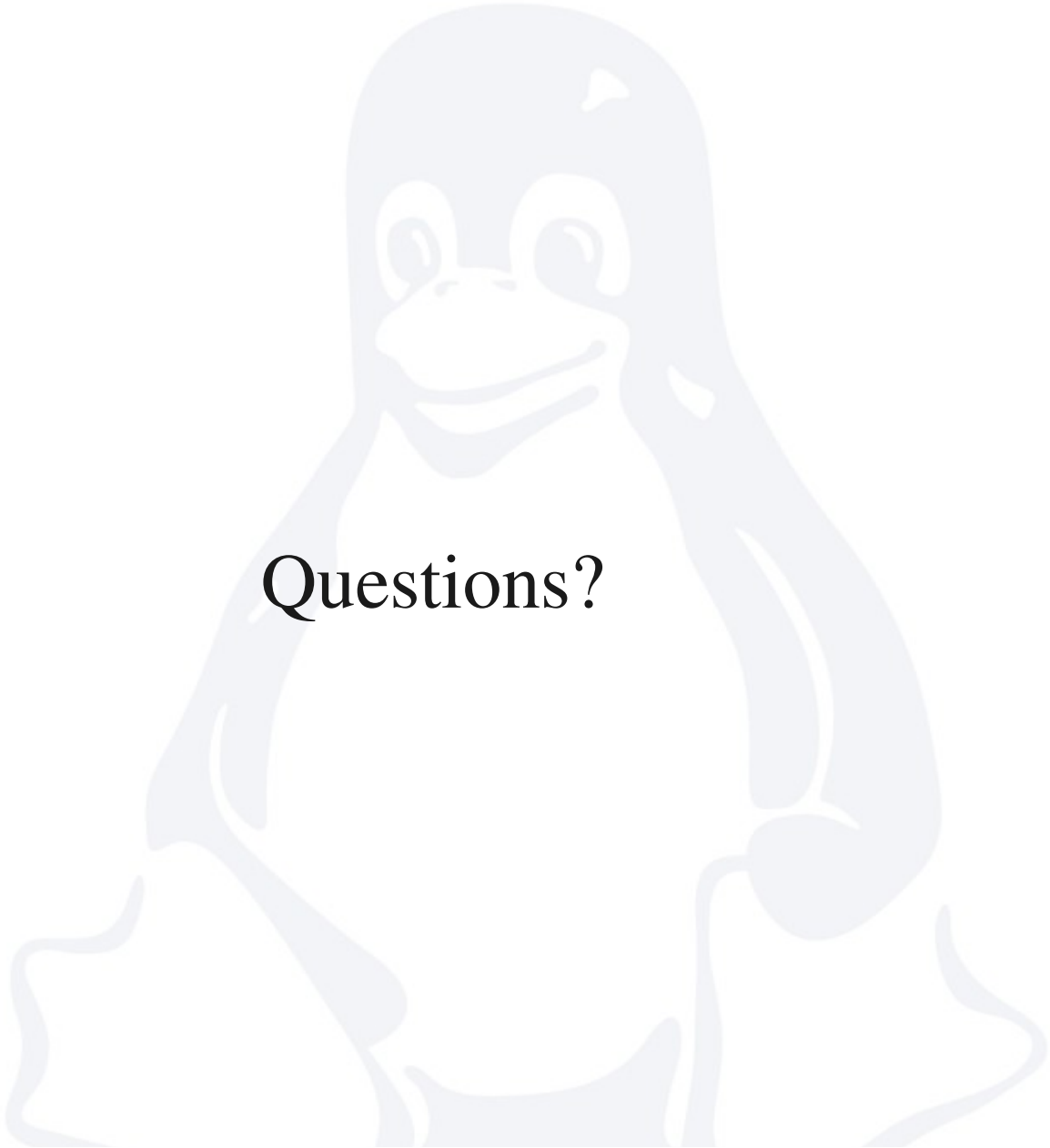
# Status

- qtest is merged
  - libqos is not yet
  - Tests for RTC, floppy controller, fw_cfg, hd-geo, i440fx, m48t59, virtio-blk-pci
  - Part of make check
  - Needs more tests!!

- qemu-test is not merged
  - ENOTIME
  - Tests for hot-plug, guest finger print, nodefaults, networking, virtio-blk serial, virtio-serial
  - Has been used for patch testing for months

# Next steps

- Write more tests
  - Merging libqos is key
- Enforce test writing for new devices
  - Either with qtest or qemu-test
- Fuzz testing and CVE regression testing
- In-tree testing is critical to QEMU's growth and maturity, but it is **only one piece** of the puzzle
  - Must focus on other pieces too!



Questions?