redhat.

A Driver Framework for qtest

Laurent Vivier

KVM Forum 2018

The driver framework for qtest project was sponsored by Google, as part of Google Summer of Code, and realized by Emanuele Esposito with mentoring from Paolo Bonzini and Laurent Vivier.

Overview

Past, Present and Future

Qtest in a nutshell

What is qtest, how it works...

A framework to rule them all

Run your test on every targets, simply

Future work

Missing parts and porting effort

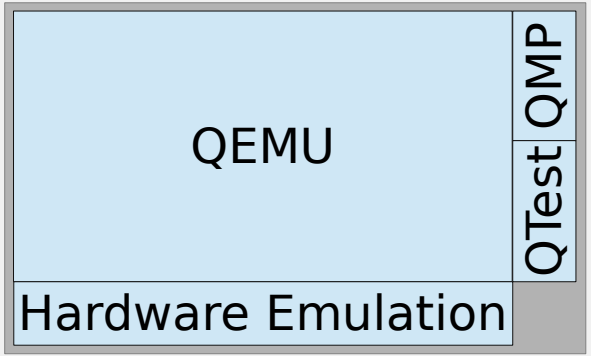
Qtest in a nutshell

What is qtest, how it works...

Qtest: QEMU side

Qtest is a library/plugin to access and control the devices of a Virtual Machine

- Qtest plugin executes commands sent by an external process
- Qtest commands are sent through a socket
- Qtest plugin takes control through the accelerator API

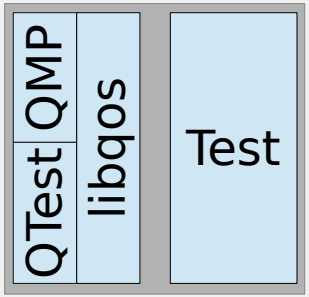



```
qemu-system-XXX ... -qtest <socket> -machine accel=qtest ...
```

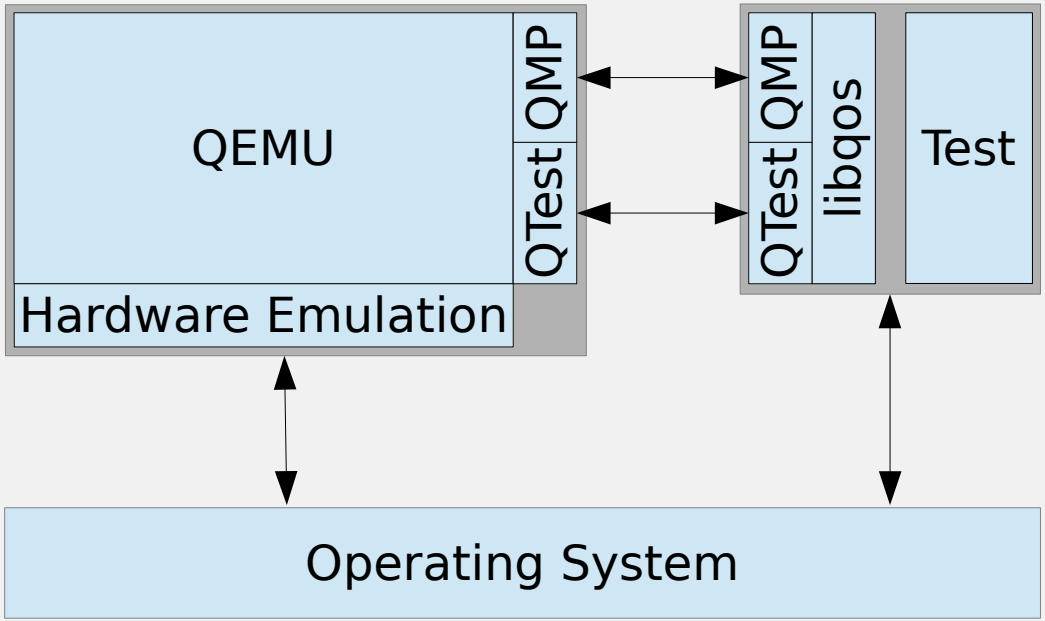
Qtest: Test application side

Libqtest provides the qtest API to the test program

- Libqtest provides a function to start the VM and connect the socket.
- Libqos provides mini-os library for writing more complicated qtest cases, on top of libqtest (to initialize guest memory allocator and access PCI API, virtio API, ...).



Overview



Example of Qtest protocol

/ppc64/rtas/get-time-of-day

```
...
[R +0.025177] rtas get-time-of-day 0 0x0 8 0x100000
[S +0.025189] OK 0
[R +0.025211] readl 0x100000
[S +0.025217] OK 0x0000000000000000
[R +0.025235] readl 0x100004
[S +0.025239] OK 0x00000000000007e2
...
```

RTAS: Run-Time Abstraction Services

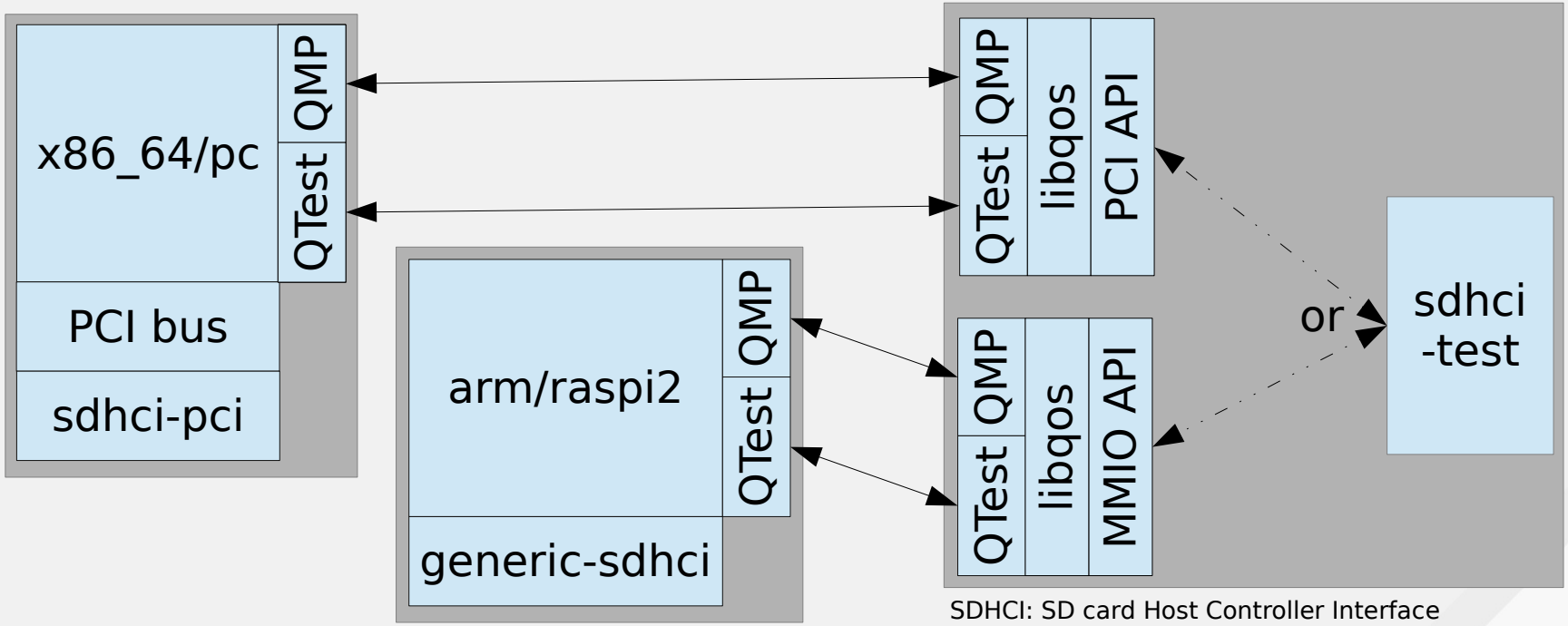
Example of QMP protocol

/i386/uhci/pci/hotplug

```
{"execute": "device_add",
 "arguments": {"port": "2", "bus": "uhci.0",
               "driver": "usb-tablet", "id": "usbdev2"}}
{"return": {}}
[R +0.017991] outl 0xcf8 0x8000e800
[S +0.017998] OK
[R +0.018054] inw 0xcfc
[S +0.018058] OK 0x8086
...
```

One test, several targets

Multiple test implementations for multiple targets



SDHCI: SD card Host Controller Interface

Multiple interfaces

/x86_64/sdhci/pc

```
[R +0.014232] outl 0xcf8 0x80002000
[S +0.014241] OK
[R +0.014263] inw 0xcfc
[S +0.014271] OK 0x1b36
[R +0.014293] outl 0xcf8 0x80002000
[S +0.014297] OK
...
```

/arm/sdhci/raspi2

```
[R +0.019111] readw 0x3f3000fe
[S +0.019121] OK 0x0000000000002402
[R +0.019150] readq 0x3f300040
[S +0.019156] OK 0x00000000052134b4
[R +0.019174] readq 0x3f300040
[S +0.019179] OK 0x00000000052134b4
...
```

Testing QEMU using qtest

One more thing...

You can find more details about qtest infrastructure in:

Testing QEMU emulated devices using qtest

Marc Marí Barceló <marc.mari.barcelo@gmail.com>

KVM Forum 2014

https://www.linux-kvm.org/images/4/43/03x09-TestingQEMU.pdf

A framework to rule them all

Run your test on every targets, simply

A framework to rule them all

The idea

Write once, test all:

- Write your test to test your hardware interface
- Describe which machines provide this interface
- Run automatically the test on all possible configurations

Abstract Concepts

Abstract concepts

The edges:

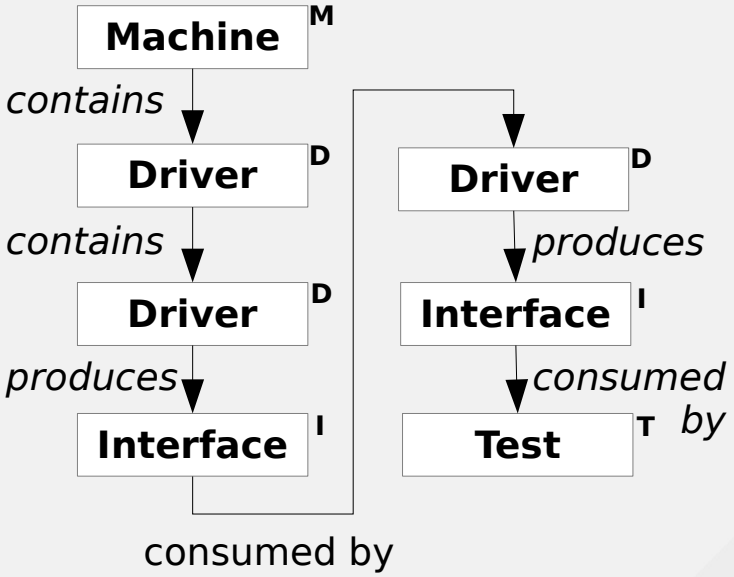
- “*contains*”, “*produces*” and “*consumed by*”

The nodes:

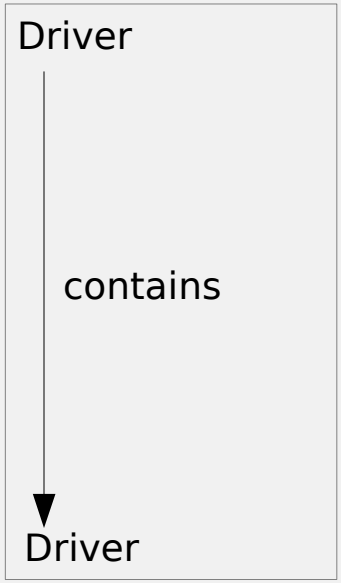
- “**machine**”, “**driver**”, “***interface***” and “**test**”

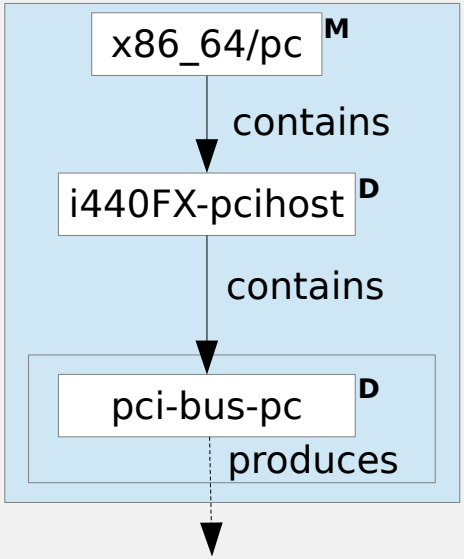
A graph is built with edges and nodes.

Tests to run can be discovered walking the graph.



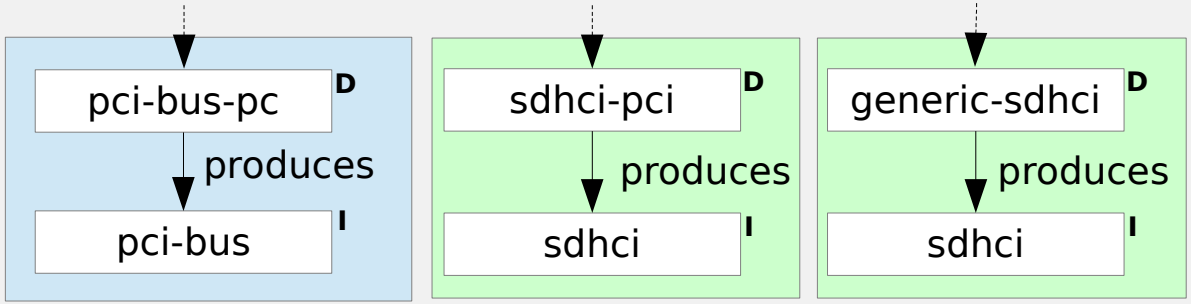
“contains” edge





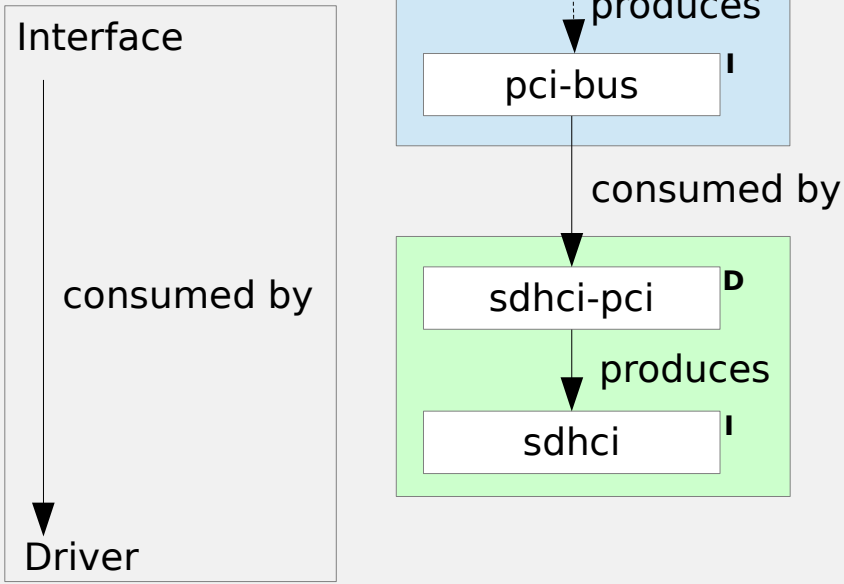
- A machine is described with **driver** *containing* **driver**

“produces” edge



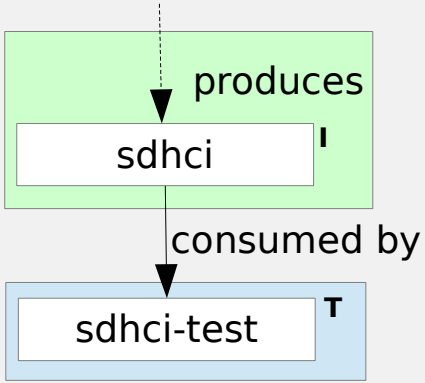
- A **driver** *produces* an **interface**

“consumed by” edge (driver)



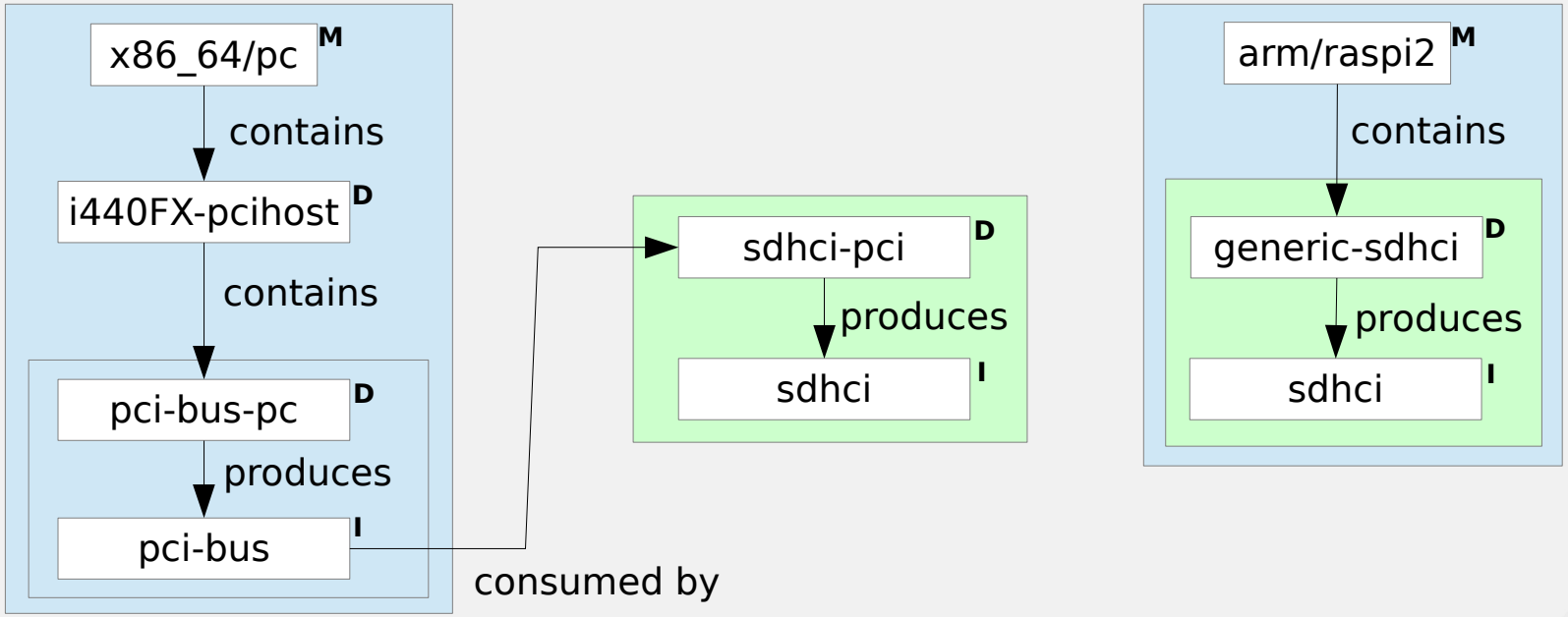
An **interface** can be *consumed by* a **driver**

“consumed by” edge (test)

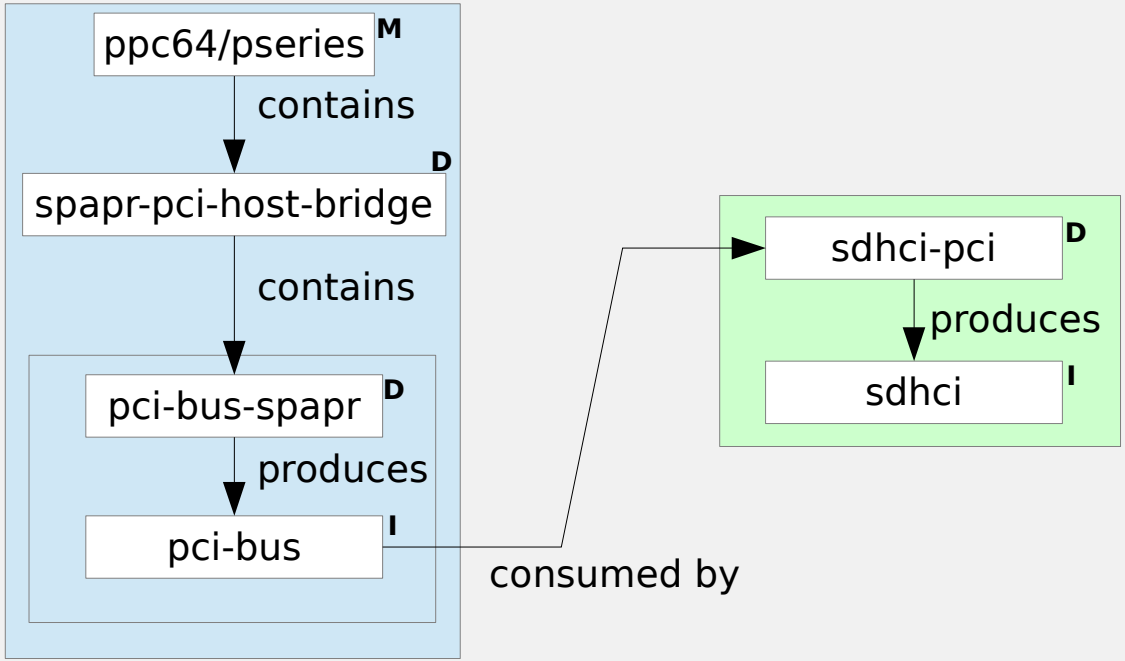


- An **interface** can be *consumed by* a **test**

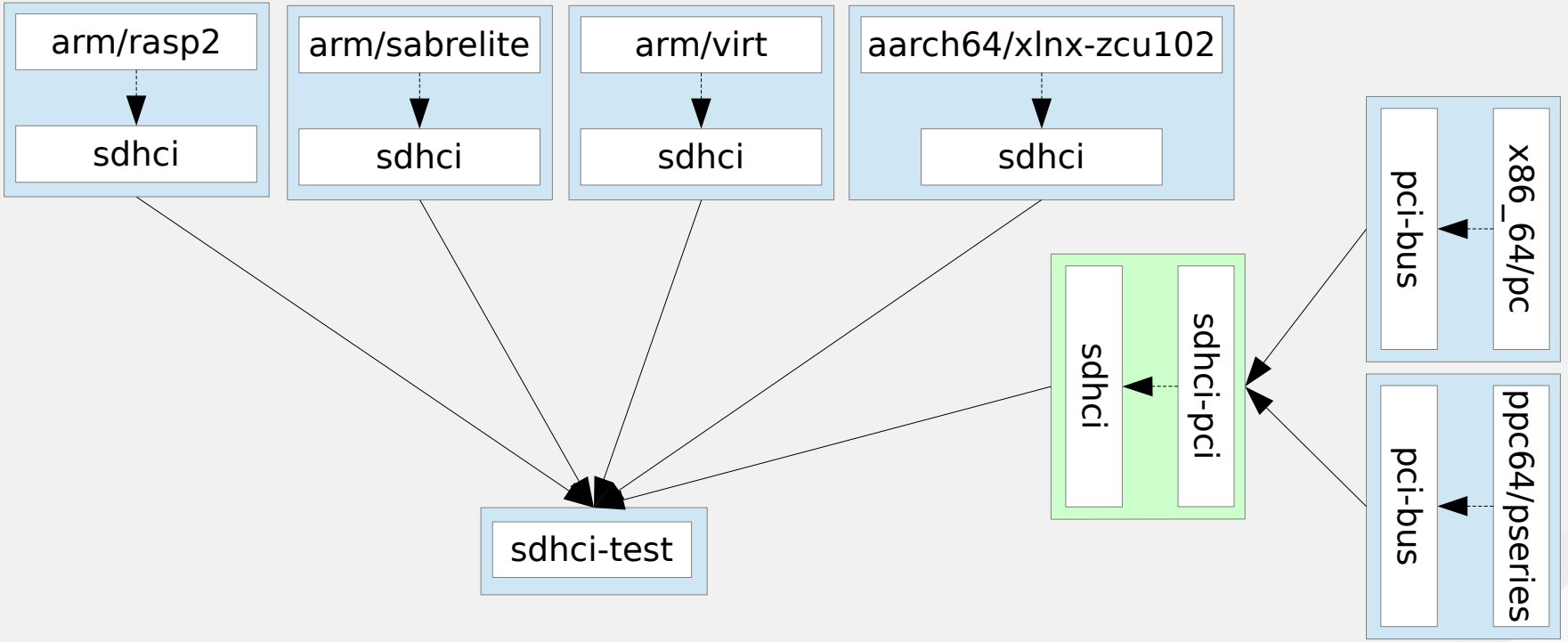
PC and Raspberry PI 2



Pseries machine



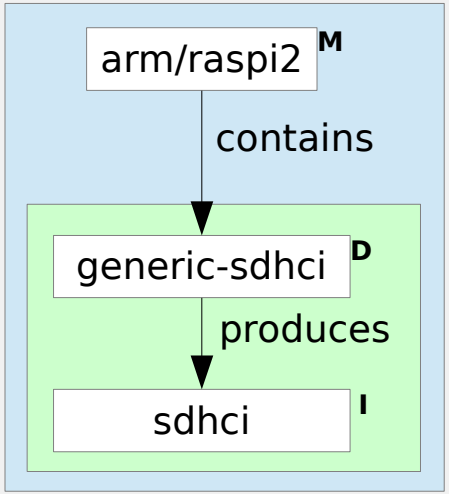
One test, 6 machines



Implementation Details

Raspberry Pi 2

Implementation example




```
qos_node_create_machine("arm/raspi2",
                        qos_create_machine_arm_raspi2);
qos_node_contains("arm/raspi2", "generic-sdhci", NULL);


qos_node_create_driver("generic-sdhci", NULL);
qos_node_produces("generic-sdhci", "sdhci");
```

Raspberry Pi 2

```
static void *qos_create_machine_arm_raspi2(QTestState *qts)
{
    QRaspi2Machine *machine = g_new0(QRaspi2Machine, 1);
...
    machine->obj.get_device = raspi2_get_device;
    qos_init_sdhci_mm(&machine->sdhci, 0x3f300000, props);
    return &machine->obj;
}
```

```
static QOSGraphObject *raspi2_get_device(void *obj, const char *device)
{
    QRaspi2Machine *machine = obj;
    if (!g_strcmp0(device, "generic-sdhci")) {
        return &machine->sdhci.obj;
    }
    g_assert_not_reached();
}
```

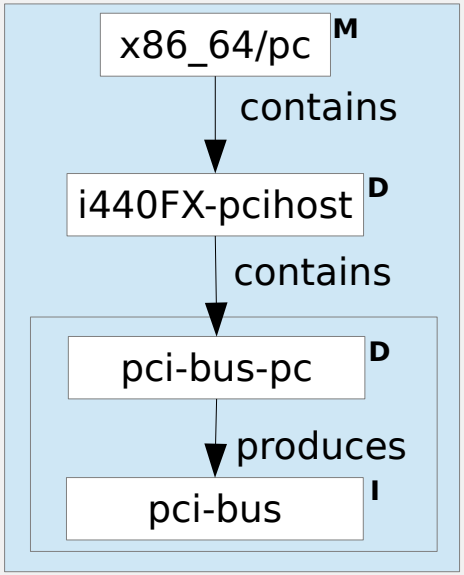
Raspberry Pi 2

```
void qos_init_sdhci_mm(QSDHCI_MemoryMapped *sdhci, QTestState *qts,
                                uint32_t addr, QSDHCIProperties *common)
{
   sdhci->obj.get_driver = sdhci_mm_get_driver;
   sdhci->sdhci.readw = sdhci_mm_readw;
   ...
   memcpy(&sdhci->sdhci.props, common, sizeof(QSDHCIProperties));
   sdhci->addr = addr;
}
```

```
static uint16_t sdhci_mm_readw(QSDHCI *s, uint32_t reg)
{
    QSDHCI_MemoryMapped *smm = container_of(s, QSDHCI_MemoryMapped, sdhci);
    return qtest_readw(smm->qts, smm->addr + reg);
}
```

PC

Implementation example




```
qos_node_create_machine("x86_64/pc", qos_create_machine_pc);
qos_node_contains("x86_64/pc", "i440FX-pcihost", NULL);

qos_node_create_driver("i440FX-pcihost", NULL);
qos_node_contains("i440FX-pcihost", "pci-bus-pc", NULL);

qos_node_create_driver("pci-bus-pc", NULL);
qos_node_produces("pci-bus-pc", "pci-bus");
```

PC

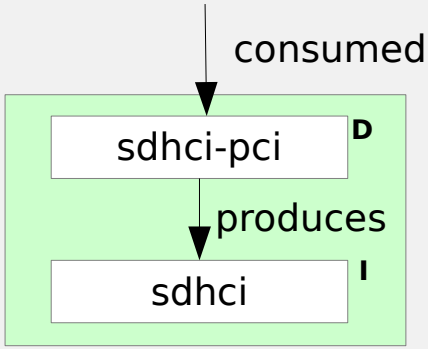
```
static void *qos_create_machine_pc(QTestState *qts)
{
    QX86_64_PCMachine *machine = g_new0(QX86_64_PCMachine, 1);
    machine->obj.get_device = pc_get_device;
    machine->obj.get_driver = pc_get_driver;
    machine->obj.destructor = pc_destructor;
    machine->alloc = pc_alloc_init_flags(qts, ALLOC_NO_FLAGS);
    qos_create_i440FX_host(&machine->bridge, qts, machine->alloc);

    return &machine->obj;
}
```

PCI bus PC

```
void qpci_init_pc(QPCIBusPC *qpci, QTestState *qts, QGuestAllocator *alloc)
{
    qpci->bus.pio_readb = qpci_pc_pio_readb;
 ...
    qpci->bus.pio_writeb = qpci_pc_pio_writeb;
 ...
    qpci->bus.memread = qpci_pc_memread;
    qpci->bus.memwrite = qpci_pc_memwrite;
    qpci->bus.config_readb = qpci_pc_config_readb;
...
    qpci->bus.config_writeb = qpci_pc_config_writeb;
...
    qpci->obj.get_driver = qpci_pc_get_driver;
}
```

SDHCI PCI card

Implementation example




```
qos_node_consumes("sdhci-pci", "pci-bus", &opts);

qos_node_create_driver("sdhci-pci", sdhci_pci_create);
qos_node_produces("sdhci-pci", "sdhci");
```

SDHCI PCI Card

```
static void *sdhci_pci_create(void *pci_bus, QGuestAllocator *alloc, void *addr)
{
    QSDHCI_PCI *spci = g_new0(QSDHCI_PCI, 1);
 ...
    spci->sdhci.readw = sdhci_pci_readw;
    spci->sdhci.readq = sdhci_pci_readq;
    spci->sdhci.writeq = sdhci_pci_writeq;

    spci->obj.get_driver = sdhci_pci_get_driver;
 ...
    return &spci->obj;
}
```

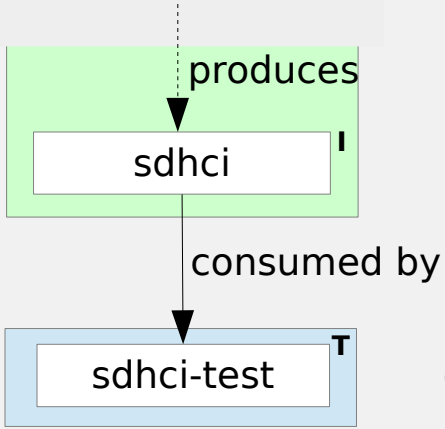
SDHCI PCI Card

```
static void *sdhci_pci_get_driver(void *object, const char *interface)
{
    QSDHCI_PCI *spci = object;
    if (!g_strcmp0(interface, "sdhci")) {
        return &spci->sdhci;
    }
    g_assert_not_reached();
}
```

```
static uint16_t sdhci_pci_readw(QSDHCI *s, uint32_t reg)
{
    QSDHCI_PCI *spci = container_of(s, QSDHCI_PCI, sdhci);
    return qpci_io_readw(&spci->dev, spci->mem_bar, reg);
}
```

SDHCI Test

Implementation Details




```
qos_add_test("registers", "sdhci", test_registers, NULL);
```

SDHCI Test

```
static void test_registers(void *obj, void *data, QGuestAllocator *alloc)
{
    QSDHCI *s = obj;

    check_specs_version(s, s->props.version);
    check_capab_capareg(s, s->props.capab.reg);
    check_capab_readonly(s);
    check_capab_v3(s, s->props.version);
    check_capab_sdma(s, s->props.capab.sdma);
    check_capab_baseclock(s, s->props.baseclock);
}
```

Future work

Missing parts and porting effort

Future work

- Merge Qtest Driver Framework in mainstream
- Port more tests to the framework
- Port more machine types (define accessors for PCI interface, specific bus, MMIO access)
- Autodetect machine architecture (“contains” relationship, with qtree or device-tree?)
- Write more generic tests that apply to all PCI devices
- Migration test support

References

- Features/qtest driver framework
  https://wiki.qemu.org/Features/qtest_driver_framework
- Features/QTest
  https://wiki.qemu.org/Features/QTest
- Testing
  https://wiki.qemu.org/Testing
- Testing QEMU emulated devices using qtest (KVM Forum 2014)
  https://www.linux-kvm.org/images/4/43/03x09-TestingQEMU.pdf
- Integrated Testing in QEMU (KVM Forum 2012)
  https://www.linux-kvm.org/images/8/89/2012-forum-Liguori-qtest.pdf

Questions

redhat.

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHat

youtube.com/user/RedHatVideos

BACKUP

Qtest in a nutshell

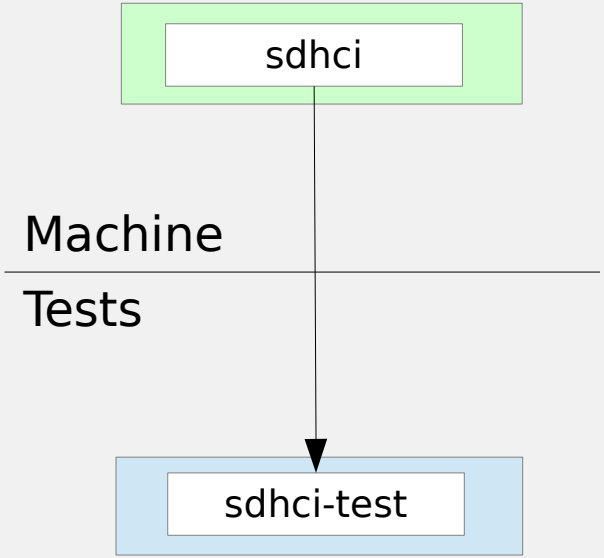
Where is Qtest?

Qtest is used when you do “make check-qtest” or “make check-qtest-<cpu>”

- Qtest is used by libqos to access VM devices
- Libqos adds a layer on top of qtest to allocate memory in the guest and to manage several buses (PCI, virtio, ...)
- In tests/ directory, C files implement drivers and tests to test VM devices using libqtest and libqos
- Tests are run through the gtester tool

A framework to rule them all

An example: SDHCI test

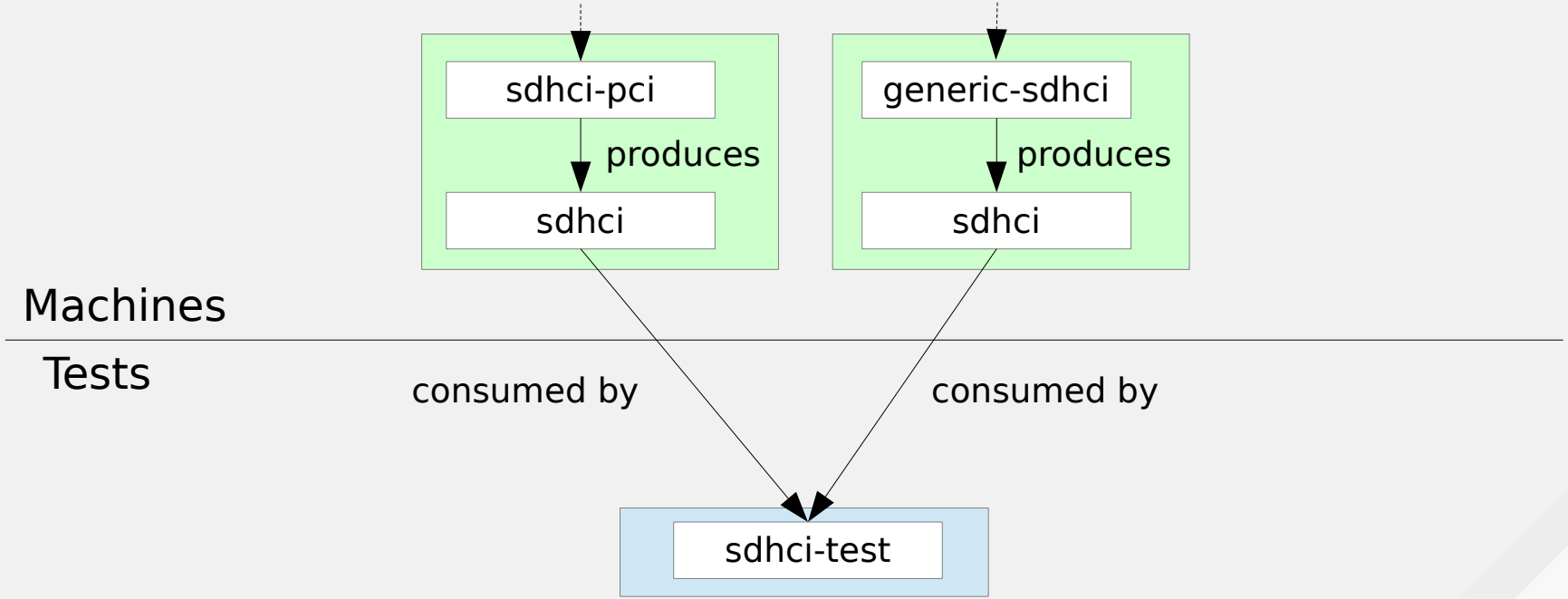


SDHCI test sequence

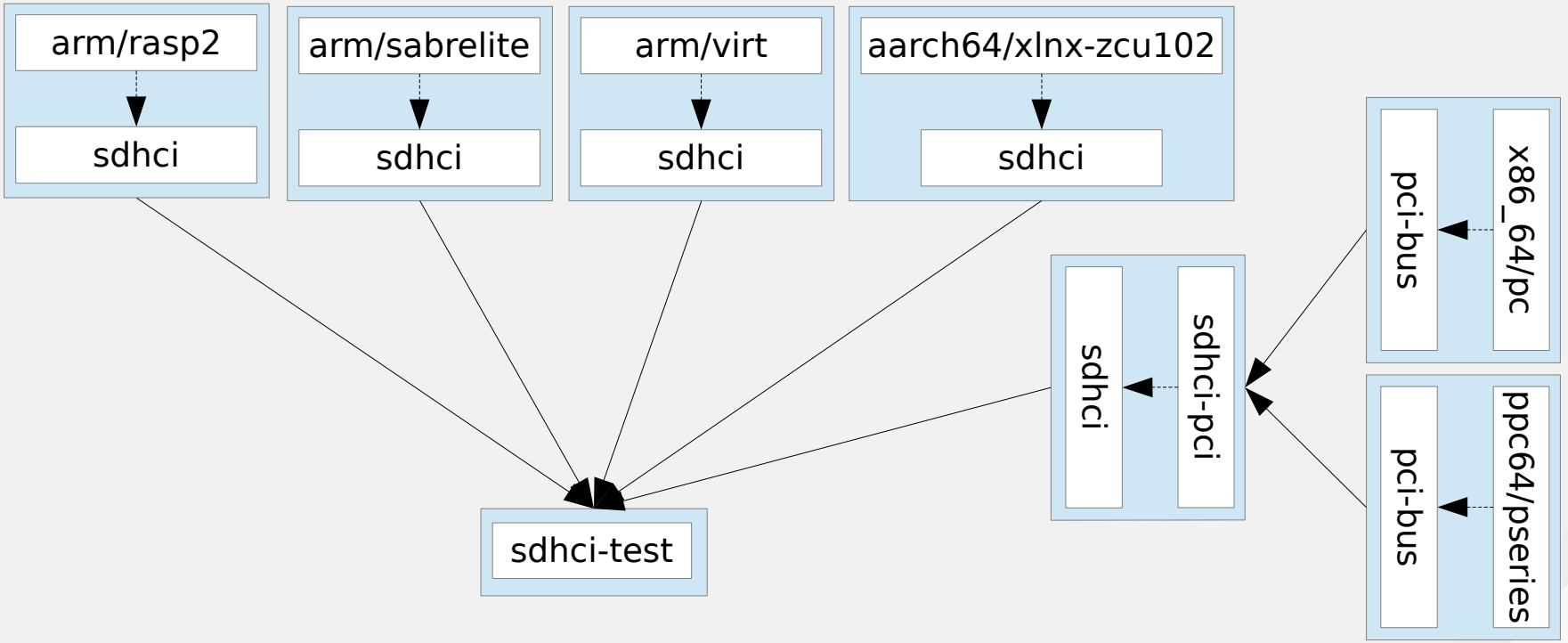
```
check_specs_version(s, s->props.version);
check_capab_capareg(s, s->props.capab.reg);
check_capab_readonly(s);
check_capab_v3(s, s->props.version);
check_capab_sdma(s, s->props.capab.sdma);
check_capab_baseclock(s, s->props.baseclock);
```

A framework to rule them all

An example: SDHCI test



A framework to rule them all



A framework to rule them all

Graph description

```
qos_node_create_machine("x86_64/pc", qos_create_machine_pc);
qos_node_create_driver("i440FX-pcihost", NULL);
qos_node_contains("x86_64/pc", "i440FX-pcihost", NULL);
qos_node_contains("i440FX-pcihost", "pci-bus-pc", NULL);
```

```
qos_node_create_machine("arm/raspi2", qos_create_machine_arm_raspi2);
qos_node_contains("arm/raspi2", "generic-sdhci", NULL);
```

```
qos_node_create_driver("generic-sdhci", NULL);
qos_node_produces("generic-sdhci", "sdhci");
qos_node_create_driver("sdhci-pci", sdhci_pci_create);
qos_node_produces("sdhci-pci", "sdhci");
qos_node_consumes("sdhci-pci", "pci-bus", &opts);
```

A framework to rule them all

sdhci-test

```
static void test_machine(void *obj, void *data, QGuestAllocator *alloc)
{
    QSDHCI *s = obj;

    check_specs_version(s, s->props.version);
...


static void check_specs_version(QSDHCI *s, uint8_t  version)
{
    uint32_t v;

    v = s->readw(s, SDHC_HCVER);
    v &= 0xff;
    v += 1;
    g_assert_cmpuint(v, ==, version);
}
```

A framework to rule them all

generic-sdhci

```
void qos_init_sdhci_smm(QSDHCI_MemoryMapped *sdhci,
                        uint32_t addr,
                        QSDHCIProperties *common)
{
    sdhci->obj.get_driver = sdhci_mm_get_driver;
    sdhci->sdhci.readw = sdhci_mm_readw;
    sdhci->sdhci.readq = sdhci_mm_readq;
    sdhci->sdhci.writeq = sdhci_mm_writeq;
    memcpy(&sdhci->sdhci.props, common,
           sizeof(QSDHCIProperties));
    sdhci->addr = addr;
}
```

A framework to rule them all

arm/raspi2

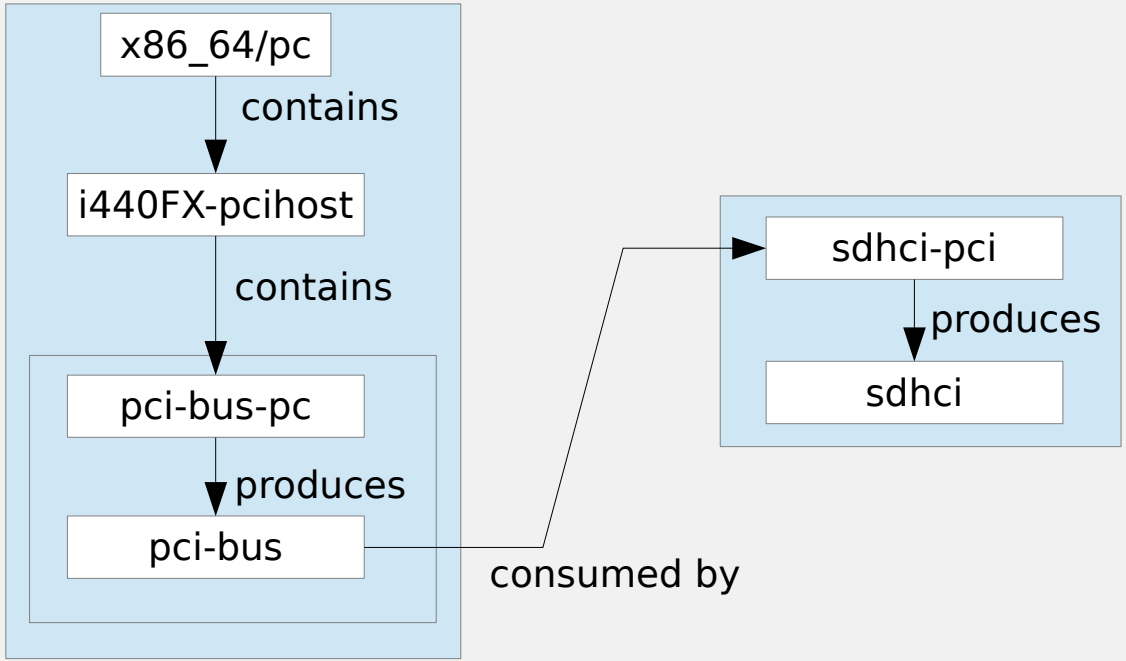
```
static void *qos_create_machine_arm_raspi2(void)
{
    QRaspi2Machine *machine = g_new0(QRaspi2Machine, 1);

    machine->obj.get_device = raspi2_get_device;
    machine->obj.get_driver = raspi2_get_driver;
    machine->obj.destructor = raspi2_destructor;
    qos_init_sdhci_smm(&machine->sdhci, 0x3f300000, &(QSDHCIProperties) {
        .version = 3,
        .baseclock = 52,
        .capab.sdma = false,
        .capab.reg = 0x052134b4
    });
    return &machine->obj;
}
```

A framework to rule them all

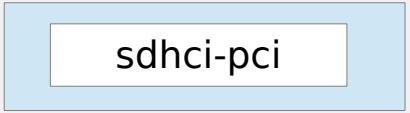
arm/raspi2

```
static QOSGraphObject *raspi2_get_device(void *obj, const char *device)
{
    QRaspi2Machine *machine = obj;
    if (!g_strcmp0(device, "generic-sdhci")) {
        return &machine->sdhci.obj;
    }

    fprintf(stderr, "%s not present in arm/raspi2\n", device);
    g_assert_not_reached();
}
```

A framework to rule them all



A framework to rule them all




```
static void *sdhci_pci_create(void *pci_bus, QGuestAllocator *alloc,
                              void *addr)
{
...
    qpci_device_init(&spci->dev, bus, addr);
    spci->mem_bar = qpci_iomap(&spci->dev, 0, &barsize);
    spci->sdhci.readw = sdhci_pci_readw;
    spci->sdhci.readq = sdhci_pci_readq;
    spci->sdhci.writeq = sdhci_pci_writeq;
...
    spci->obj.get_driver = sdhci_pci_get_driver;
    spci->obj.start_hw = sdhci_pci_start_hw;
    spci->obj.destructor = sdhci_destructor;
    return &spci->obj;
}
```

A framework to rule them all

x86_64/pc

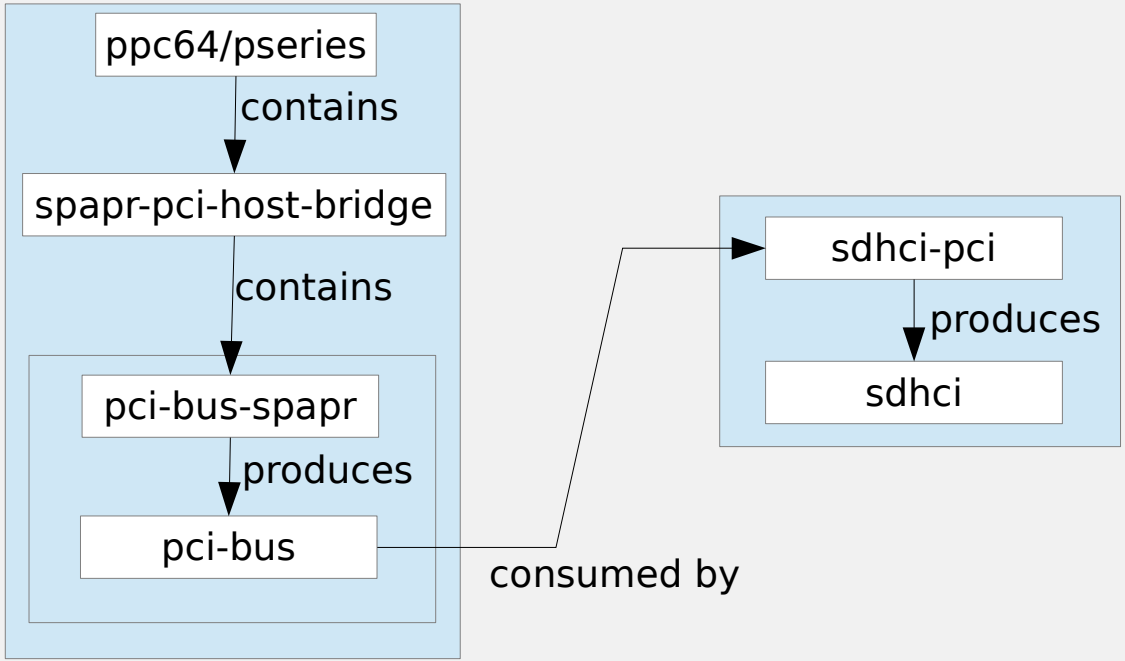
```
static void *qos_create_machine_pc(void)
{
    QX86_64_PCMachine *machine = g_new0(QX86_64_PCMachine, 1);
    machine->obj.get_device = pc_get_device;
    machine->obj.get_driver = pc_get_driver;
    machine->obj.destructor = pc_destructor;
    machine->alloc = pc_alloc_init_flags(global_qtest, ALLOC_NO_FLAGS);
    qos_create_i440FX_host(&machine->bridge, machine->alloc);

    return &machine->obj;
}
```

A framework to rule them all

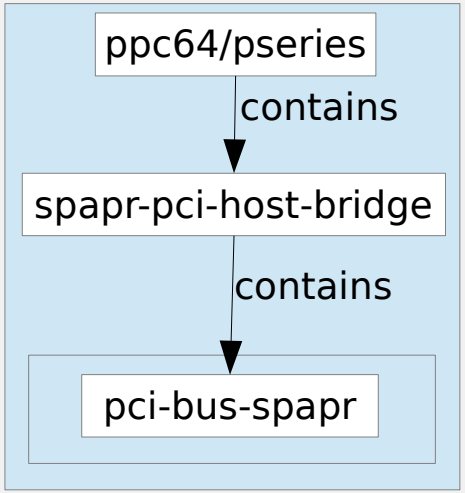
i440FX-pcihost

```
static void qos_create_i440FX_host(i440FX_pcihost *host,
                                   QGuestAllocator *alloc)
{
    host->obj.get_device = i440FX_host_get_device;
    qpci_init_pc(&host->pci, global_qtest, alloc);
}

static QOSGraphObject *i440FX_host_get_device(void *obj, const char *device)
{
    i440FX_pcihost *host = obj;
    if (!g_strcmp0(device, "pci-bus-pc")) {
        return &host->pci.obj;
    }
    fprintf(stderr, "%s not present in i440FX-pcihost\n", device);
    g_assert_not_reached();
}
```

A framework to rule them all

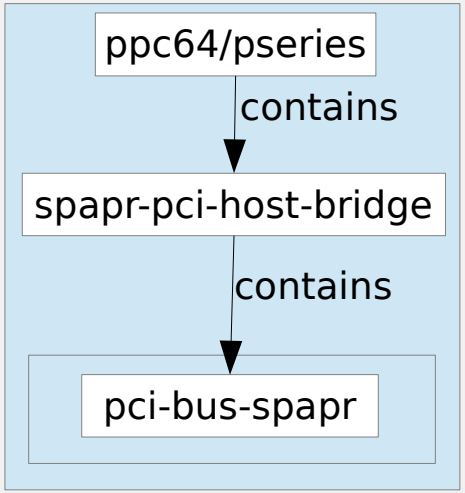


A framework to rule them all



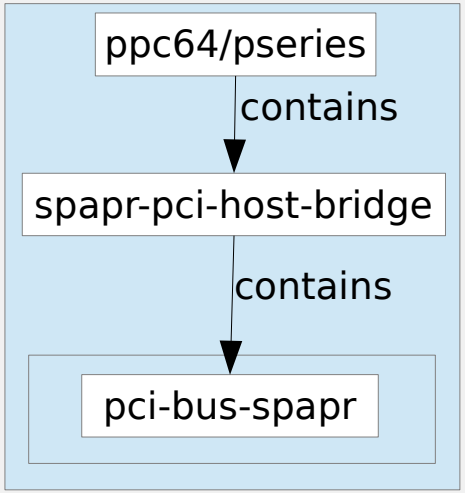

```
qos_node_create_machine("ppc64/pseries", qos_create_machine_spapr);
qos_node_create_driver("spapr-pci-host-bridge", NULL);
qos_node_contains("ppc64/pseries", "spapr-pci-host-bridge", NULL);
qos_node_contains("spapr-pci-host-bridge", "pci-bus-spapr", NULL);
```

A framework to rule them all




```
static void *qos_create_machine_spapr(void)
{
    Qppc64_pseriesMachine *machine = g_new0(Qppc64_pseriesMachine, 1);
    machine->obj.get_device = spapr_get_device;
    machine->obj.get_driver = spapr_get_driver;
    machine->obj.destructor = spapr_destructor;
    machine->alloc =  spapr_alloc_init_flags(global_qtest,
                                             ALLOC_NO_FLAGS);
    qos_create_QSPAPR_host(&machine->bridge, machine->alloc);

    return &machine->obj;
}
```

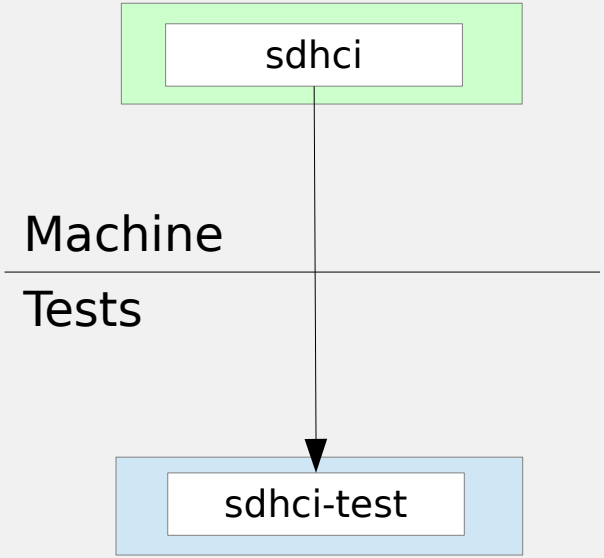
A framework to rule them all




```
static void qos_create_QSPAPR_host(QSPAPR_pci_host *host,
                                   QGuestAllocator *alloc)
{
    host->obj.get_device = QSPAPR_host_get_device;
    qpci_init_spapr(&host->pci, global_qtest, alloc);
}

static QOSGraphObject *QSPAPR_host_get_device(void *obj,
                                              const char *device)
{
    QSPAPR_pci_host *host = obj;
    if (!g_strcmp0(device, "pci-bus-spapr")) {
        return &host->pci.obj;
    }
    fprintf(stderr, "%s not present in QSPAPR_pci_host\n", device);
    g_assert_not_reached();
}
```

A framework to rule them all

An example: SDHCI test



SDHCI test sequence

```
check_specs_version(s, s->props.version);
check_capab_capareg(s, s->props.capab.reg);
check_capab_readonly(s);
check_capab_v3(s, s->props.version);
check_capab_sdma(s, s->props.capab.sdma);
check_capab_baseclock(s, s->props.baseclock);
```