PCI Interrupts for x86 Machines under FreeBSD

May 18, 2007

John Baldwin
jhb@FreeBSD.org

Introduction

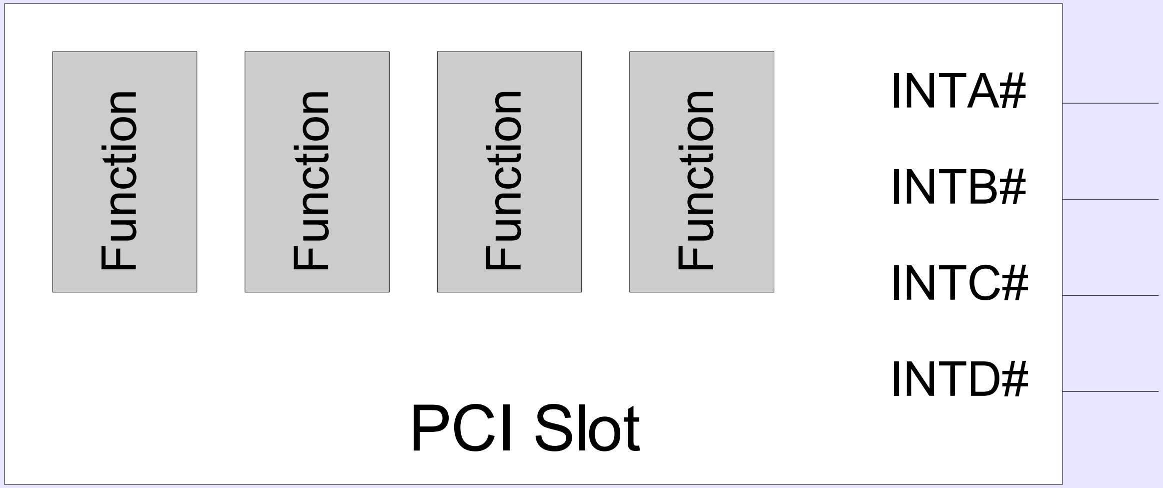
- Hardware for PCI INTx interrupts
  - x86 CPU interrupts
  - PCI INTx signals
  - x86 interrupt controllers
- Interrupt Routing
  - PCI-PCI bridge swizzle
  - Tables: $PIR, MP Table, ACPI _PRT
- Message Signaled Interrupts

x86 CPU Interrupts

- OS provides array of handlers called Interrupt Descriptor Table (IDT)
- Each interrupt contains vector which indexes table
- Vectors 0-31 reserved for Faults/Exceptions
- Vectors 32-255 available for device interrupts, IPIs, etc.

PCI INTx

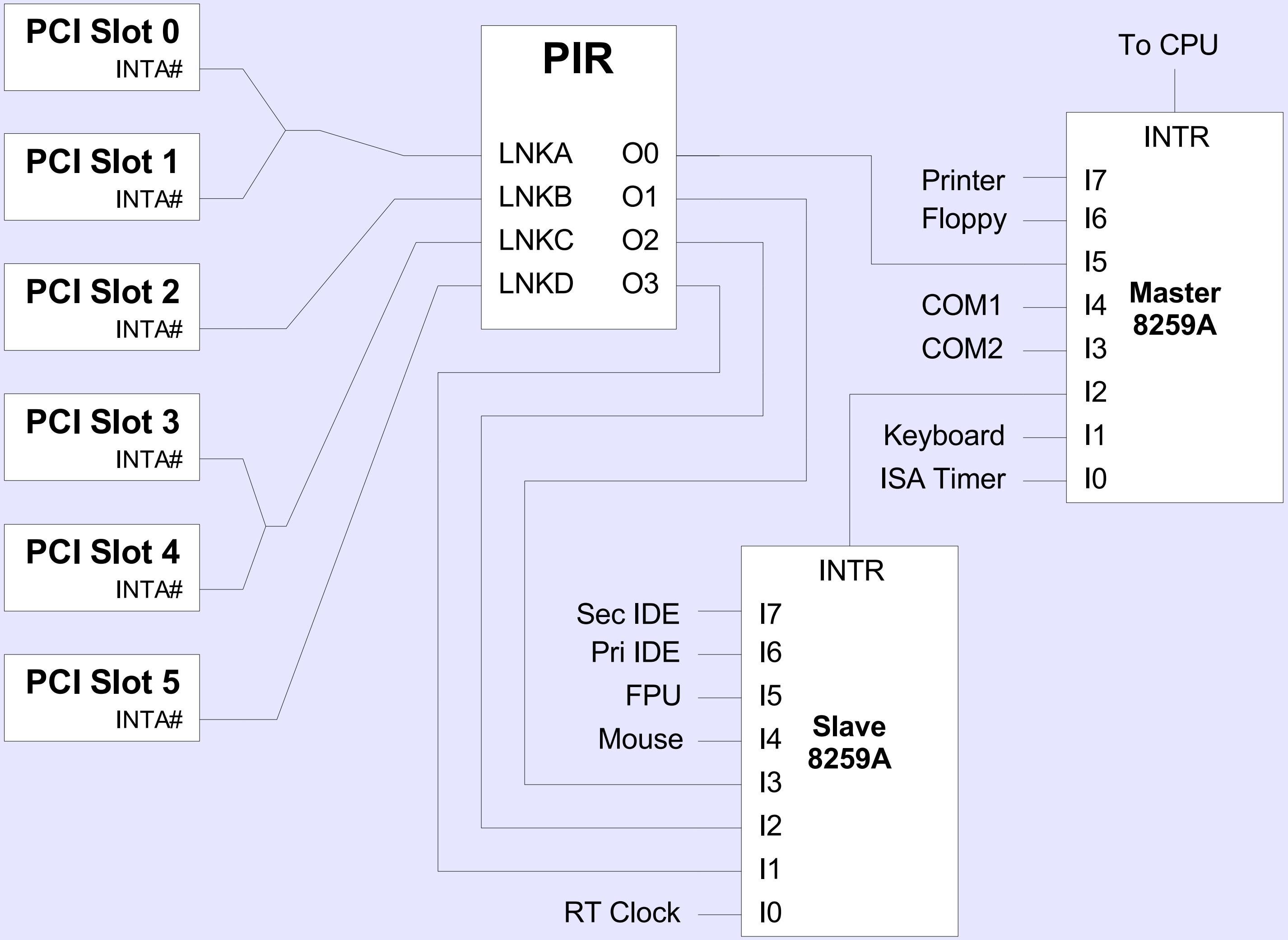
- PCI slots have 4 interrupts
  - INTA#, INTB#, INTC#, INTD#
- PCI functions use 1 interrupt from parent slot



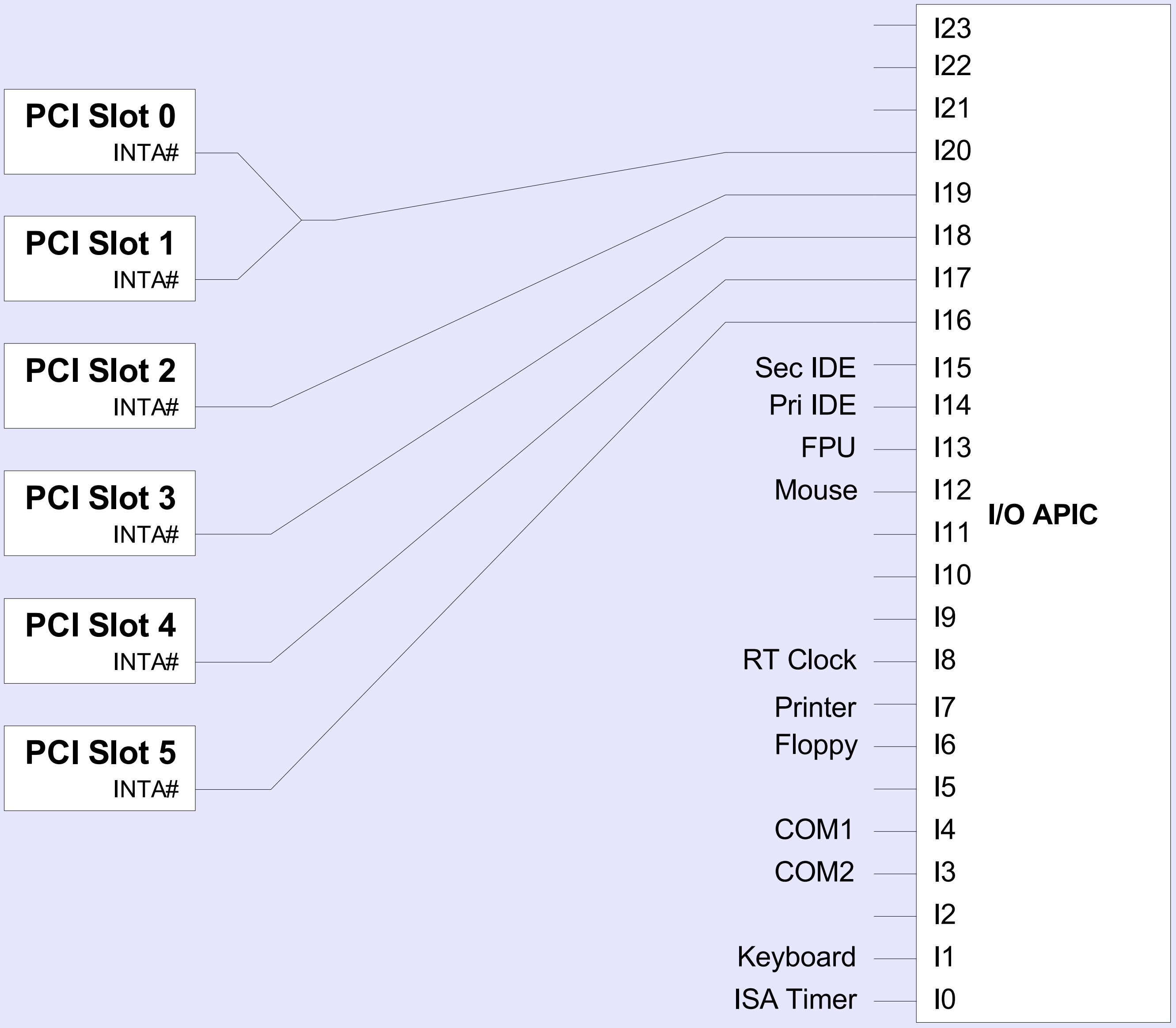
x86 Interrupt Controllers

- 8259A PICs (PC-AT)
- Programmable Interrupt Router
- APICs

8259A & PIR


PCI Slot 0
INTA#
PCI Slot 1
INTA#
PCI Slot 2
INTA#
PCI Slot 3
INTA#
PCI Slot 4
INTA#
PCI Slot 5
INTA#
PIR
LNKA
LNKB
LNKC
LNKD
O0
O1
O2
O3
To CPU
INTR
Printer
Floppy
COM1
COM2
Keyboard
ISA Timer
I7
I6
I5
I4
I3
I2
I1
I0
Master
8259A
INTR
Sec IDE
Pri IDE
FPU
Mouse
RT Clock
I7
I6
I5
I4
I3
I2
I1
I0
Slave
8259A

I/O APIC



PCI Slot 0
INTA#

PCI Slot 1
INTA#

PCI Slot 2
INTA#

PCI Slot 3
INTA#

PCI Slot 4
INTA#

PCI Slot 5
INTA#

I23
I22
I21
I20
I19
I18
I17
I16
Sec IDE I15
Pri IDE I14
FPU I13
Mouse I12
I11
I10
I9
RT Clock I8
Printer I7
Floppy I6
I5
COM1 I4
COM2 I3
I2
Keyboard I1
ISA Timer I0

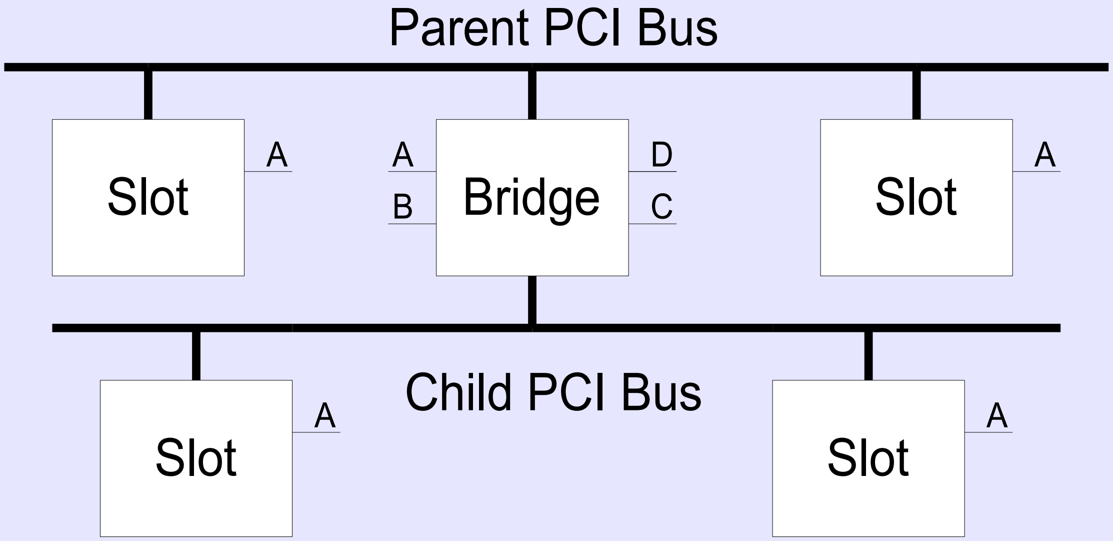
I/O APIC

Interrupt Routing

- How everything is hooked together
- Maps PCI (bus, slot, pin) to input pin on 8259A or I/O APIC
  - May have to detour through a PIR
- PCI-PCI bridge swizzle
- Various tables on x86
  - $PIR for 8259A & PIR
  - MP Table for APIC
  - ACPI for both

PCI-PCI Bridge Swizzle

- Routes interrupts across bridge from “downstream” bus to “upstream” bus
- For PCI-PCI bridges in add-on cards
- new_pin = (child_slot + child_pin) % 4



$PIR Table

```
Entry  Location  Bus Device Pin  Link  IRQs
  0    embedded    0    2    A   0x60  3 4 5 6 10 11 14 15
  1    embedded    1    0    A   0x60  3 4 5 6 10 11 14 15
  1    embedded    1    0    B   0x61  3 4 5 6 10 11 14 15
  1    embedded    1    0    C   0x62  3 4 5 6 10 11 14 15
  1    embedded    1    0    D   0x63  3 4 5 6 10 11 14 15
  2    embedded    0   29    A   0x60  3 4 5 6 10 11 14 15
  2    embedded    0   29    B   0x63  3 4 5 6 10 11 14 15
  2    embedded    0   29    C   0x62  3 4 5 6 10 11 14 15
  2    embedded    0   29    D   0x6b  3 4 5 6 10 11 14 15
  3    embedded    0   31    A   0x62  3 4 5 6 10 11 14 15
  4    embedded    4   13    A   0x61  3 4 5 6 10 11 14 15
  5    embedded    2    4    A   0x60  3 4 5 6 10 11 14 15
  6    embedded    4    3    A   0x68  3 4 5 6 10 11 14 15
  7    slot 1      3    7    A   0x62  3 4 5 6 10 11 14 15
  7    slot 1      3    7    B   0x63  3 4 5 6 10 11 14 15
  7    slot 1      3    7    C   0x60  3 4 5 6 10 11 14 15
  7    slot 1      3    7    D   0x61  3 4 5 6 10 11 14 15
```

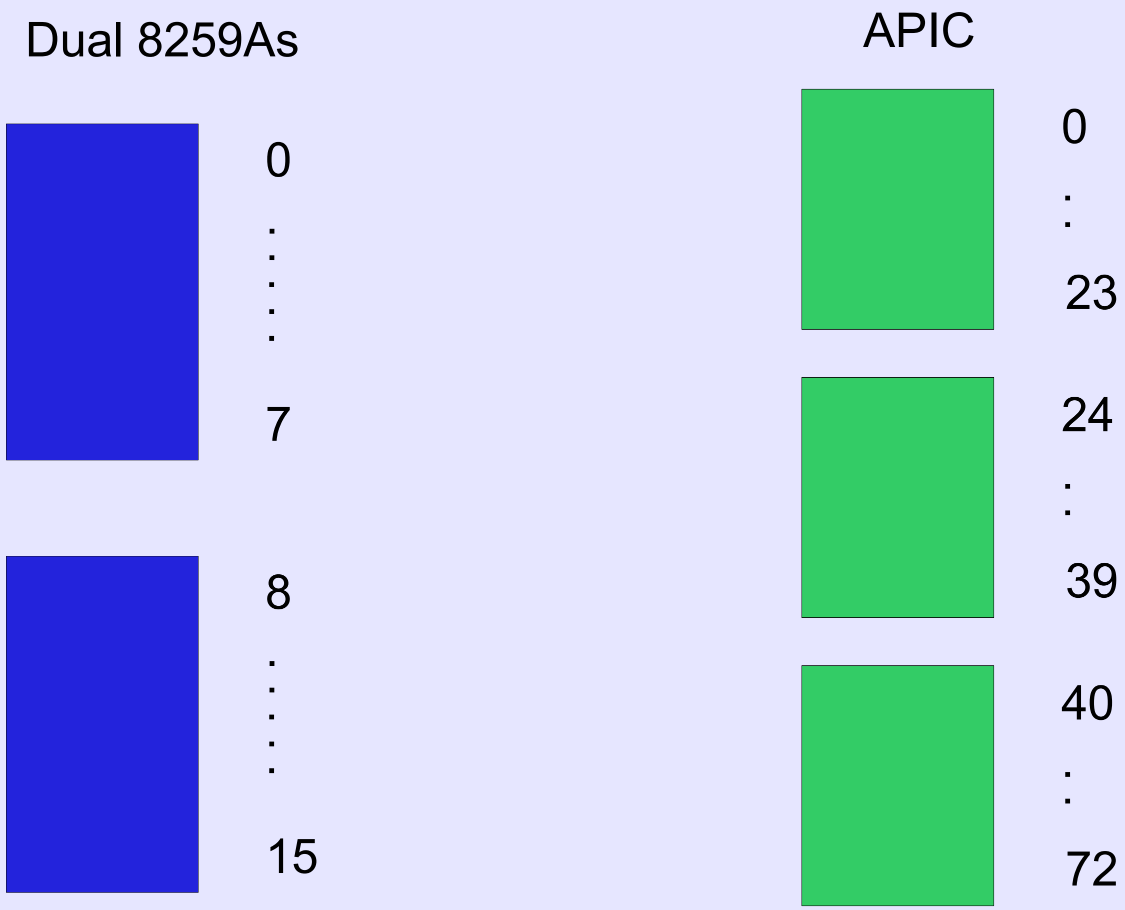
MP Table

```
I/O Ints:       Type    Polarity    Trigger     Bus ID   IRQ    APIC ID PIN#
                ExtINT  active-hi        edge        5     0          8    0
                INT      conforms    conforms        5     1          8    1
                INT      conforms    conforms        5     0          8    2
                INT      conforms    conforms        5     3          8    3
                INT      conforms    conforms        5     4          8    4
                INT      conforms    conforms        5     7          8    7
                INT      conforms    conforms        5     8          8    8
                INT      conforms    conforms        5     9          8    9
                INT      conforms    conforms        5    12          8   12
                INT      conforms    conforms        0   2:A          8   16
                INT      conforms    conforms        0  29:A          8   16
                INT      conforms    conforms        0  29:B          8   19
                INT      conforms    conforms        0  29:D          8   23
                INT      conforms    conforms        0  31:A          8   18
                INT      conforms    conforms        4  13:A          8   17
                INT      conforms    conforms        4   3:A          8   20
                INT      conforms    conforms        2   4:A          9    0
                INT      conforms    conforms        3   7:A         10    2
                INT      conforms    conforms        3   7:B         10    3
                INT      conforms    conforms        3   7:C         10    0
                INT      conforms    conforms        3   7:D         10    1
```

ACPI

- Global System Interrupts
  - 0-15 are ISA IRQs
  - 16+ are I/O APIC pins
- OS calls _PIC method to tell BIOS 8259A vs APIC
- Each PCI bus contains _PRT table
- Pins on Programmable Interrupt Router are Link Devices

Global System Interrupts



Dual 8259As

0

7

8

15

APIC

0

23

24

39

40

72

ACPI _PIC Method

```
Scope (\)
{
    Name (PICF, 0x00)
    Method (_PIC, 1, NotSerialized)
    {
        Store (Arg0, PICF)
    }
}
```

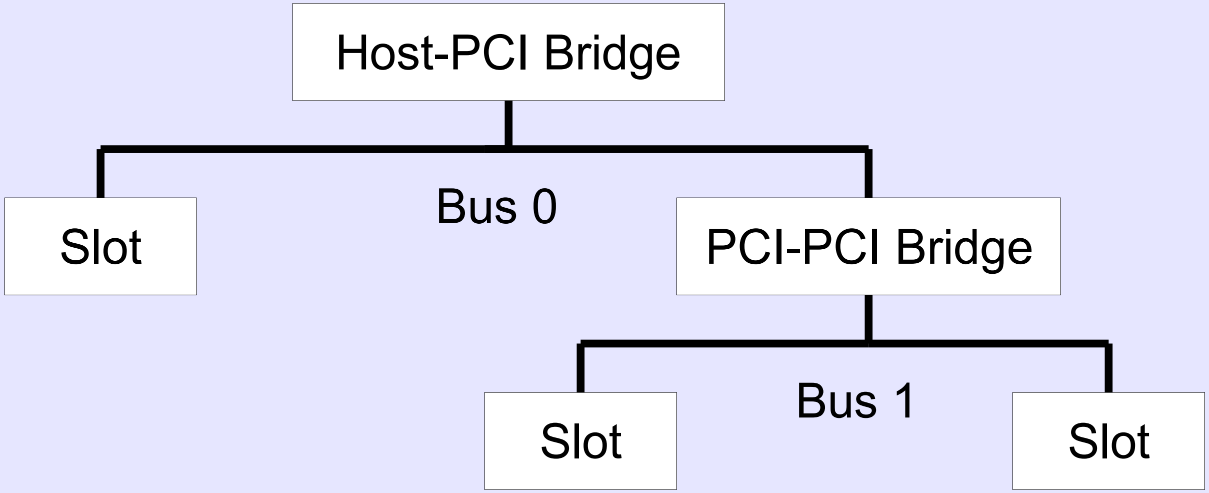
ACPI _PRT Method

```
Device (PXHB)
{
    ...
    Name (PIC3, Package (0x04)
    {
        Package (0x04)
        {
            0x0007FFFF,
            0x00,
            LNKC,
            0x00
        },
        ...
    })
    Name (APC3, Package (0x04)
    {
        Package (0x04)
        {
            0x0007FFFF,
            0x00,
            0x00,
            0x42
        },
        ...
    })
    Method (_PRT, 0, NotSerialized)
    {
        If (LNot (PICF))
        {
            Store (PIC3, Local0)
        }
        Else
        {
            Store (APC3, Local0)
        }

        Return (Local0)
    }
}
```

FreeBSD's PCI Interrupt Routing

- PCI bus asks parent bridge to route interrupt
- Requests propagate up device tree until answered



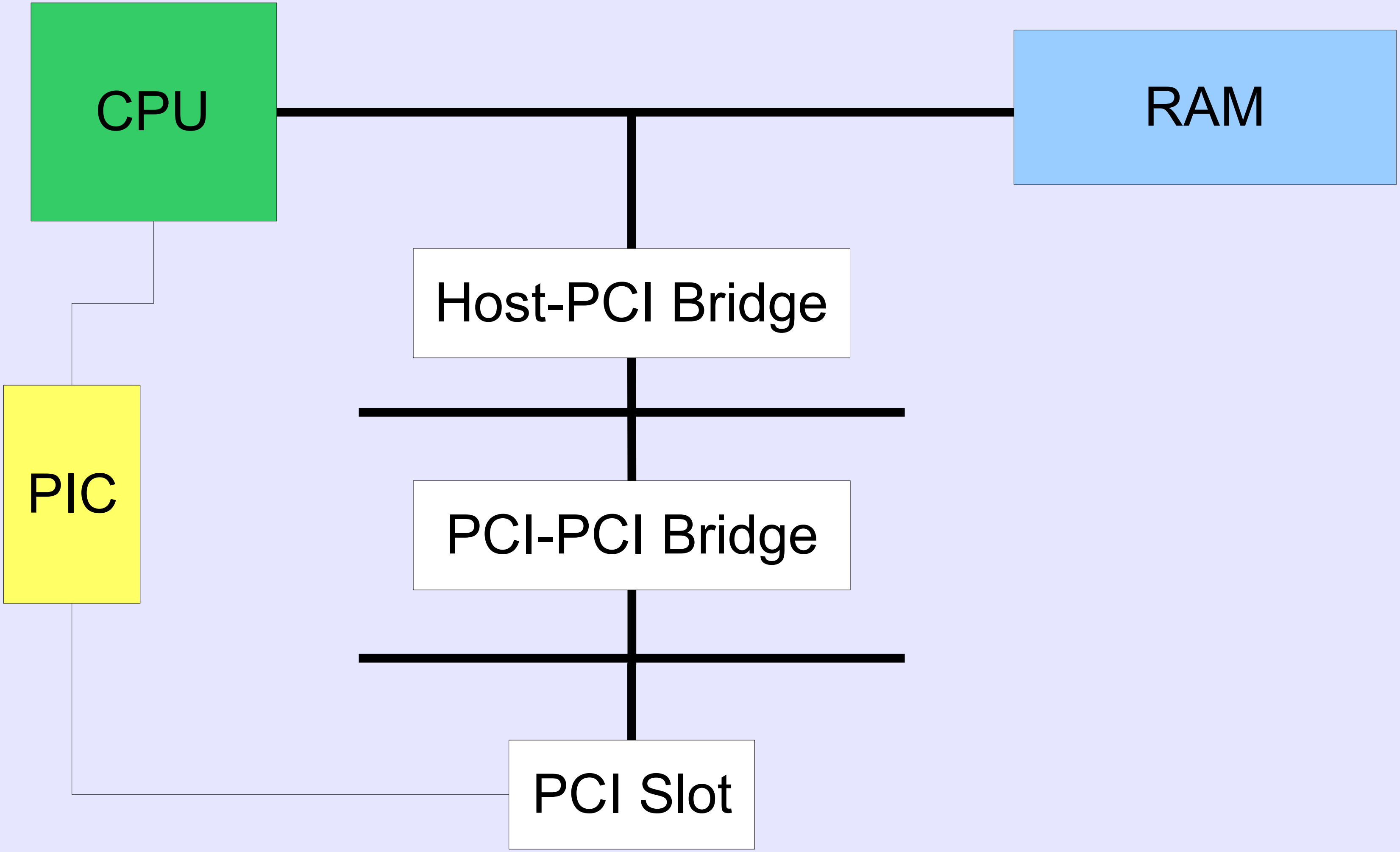
Mapping IRQs to IDT Vectors

- Interrupt routing routes PCI interrupt to interrupt controller pin (IRQ)
- FreeBSD uses GSI-like scheme to map interrupt controller pins to IRQs
- Allocate IDT vectors on-demand to IRQs

Message Signaled Interrupts

- Problems with INTx Interrupts
  - Single interrupt per function
  - Interrupt is a sideband signal
  - Interrupt routing is a pain
- MSI Addresses These Problems
  - Each function can have multiple messages
  - Interrupt triggered via memory write
  - Memory data contains IDT vector on x86

INTx Sideband Signals are Bad



FreeBSD's MSI Implementation

- PCI bus asks parent bridge for MSI IRQs
- x86 “nexus” driver creates MSI IRQs on the fly and assigns IDT vectors on-demand

Conclusion

- PCI INTx interrupts are messy
  - Interrupt Routing can be painful, esp. on x86
- MSI will hopefully be better
  - PCI-express mandates it for PCI-express devices

Q&A

- Paper and slides are available online
  - http://www.FreeBSD.org/~jhb/papers/bsdcan/2007/
- Questions?