Systems Design and Implementation

I.8 – Device Drivers

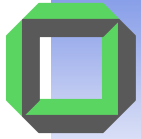
System Architecture Group, SS 2009

University of Karlsruhe

June 16, 2008

Jan Stoess

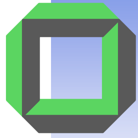
University of Karlsruhe



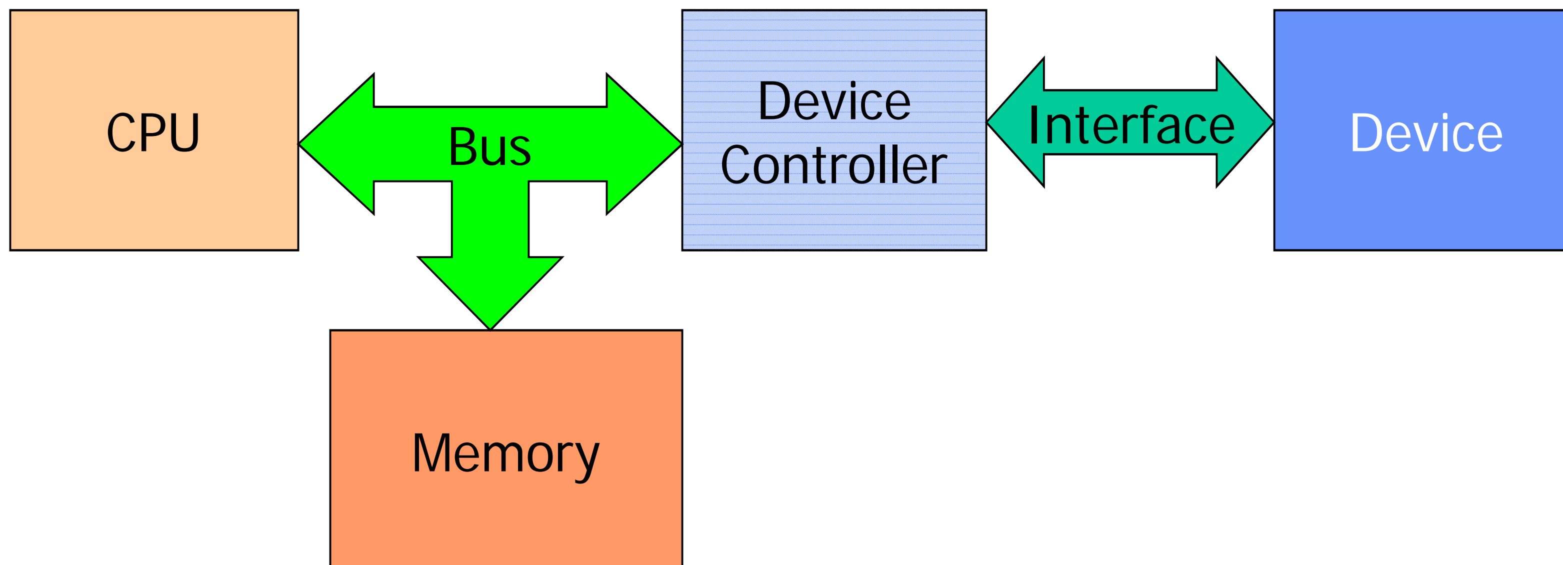
Overview

- Device I/O related concepts
- L4 support for interfacing to hardware
- Software issues and structure
- PC Screen and Keyboard

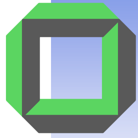
© 2009 University of Karlsruhe, System Architecture Group



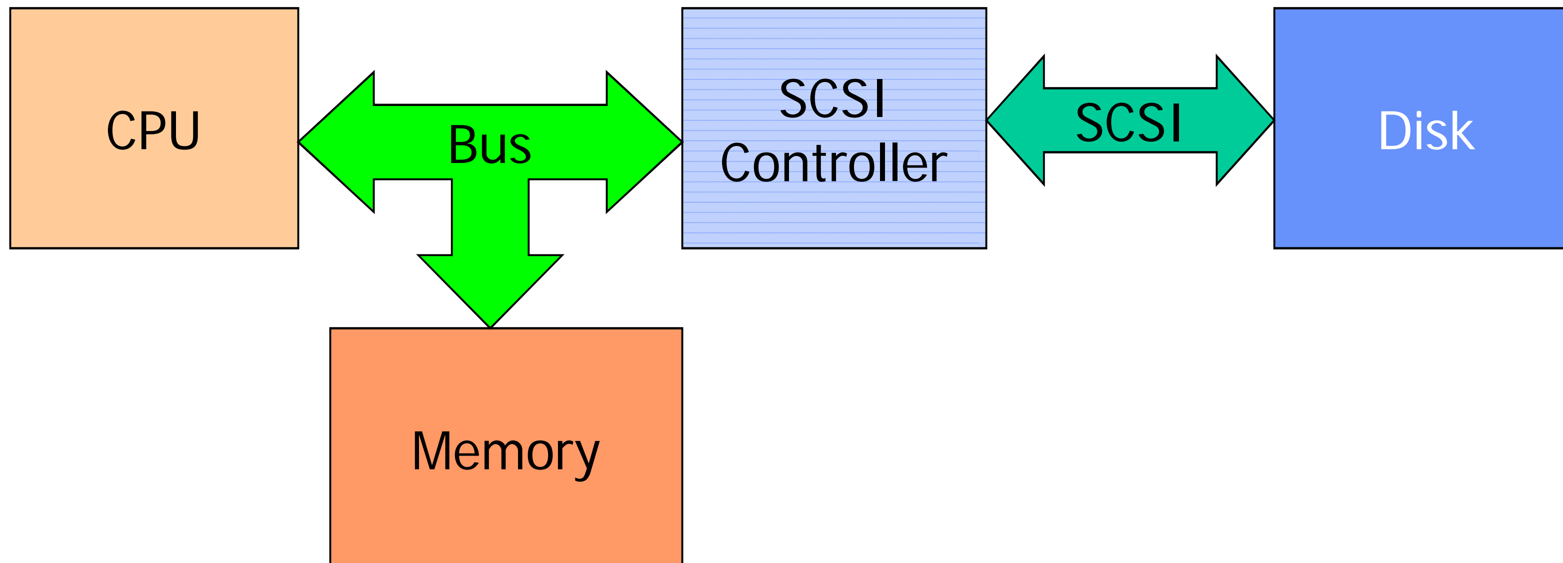
Simplified Hardware Model



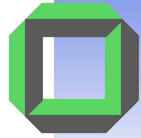
© 2009 University of Karlsruhe, System
Architecture Group



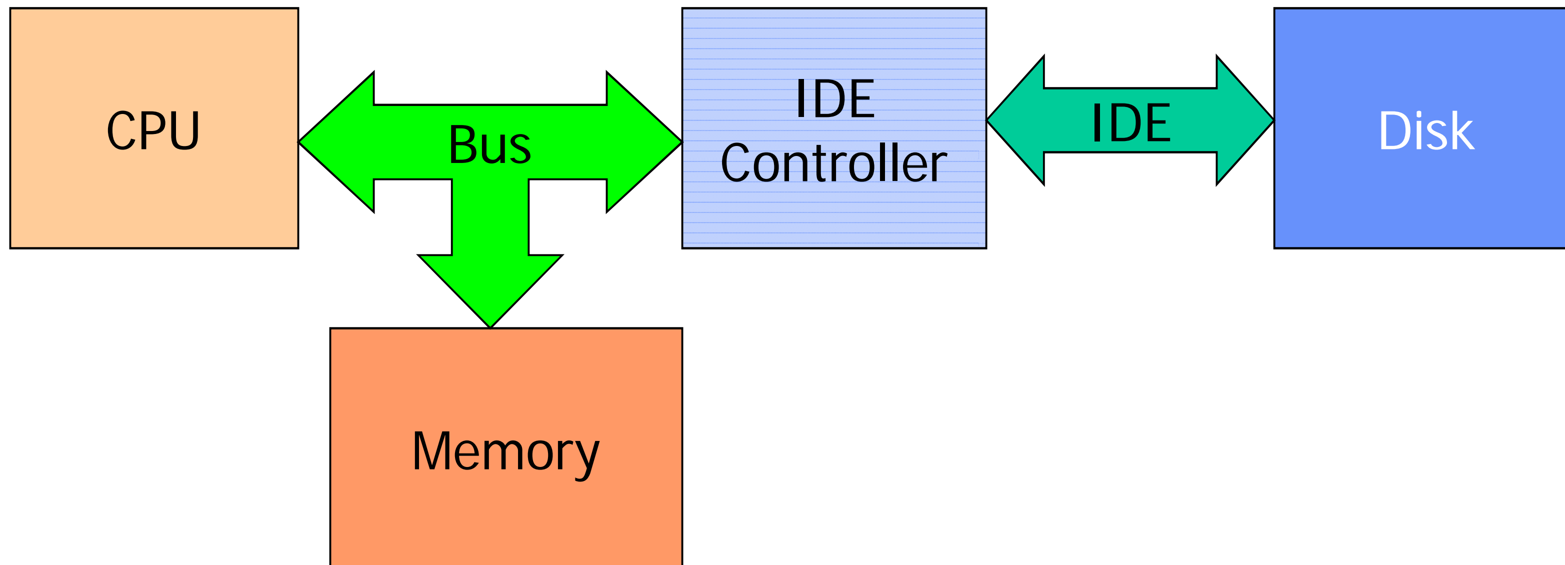
Example Hardware



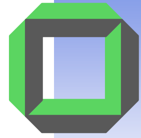
© 2009 University of Karlsruhe, System Architecture Group



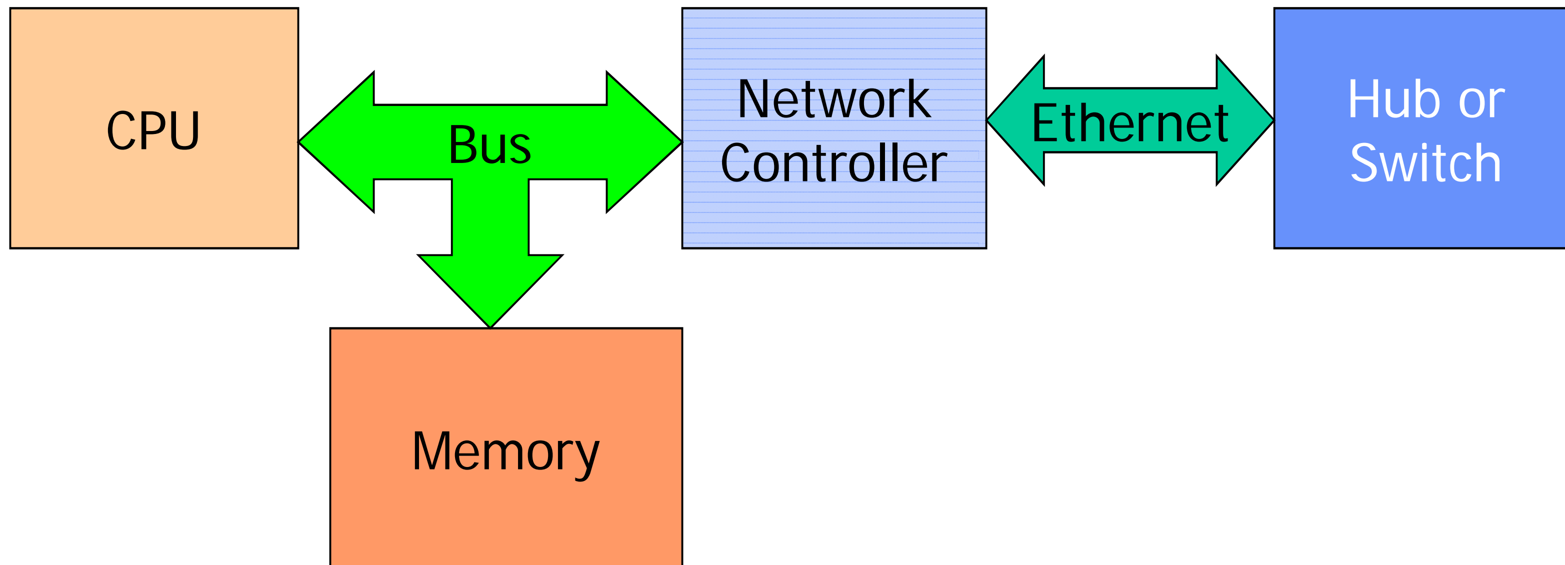
Example Hardware



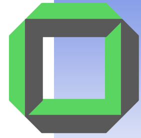
© 2009 University of Karlsruhe, System Architecture Group



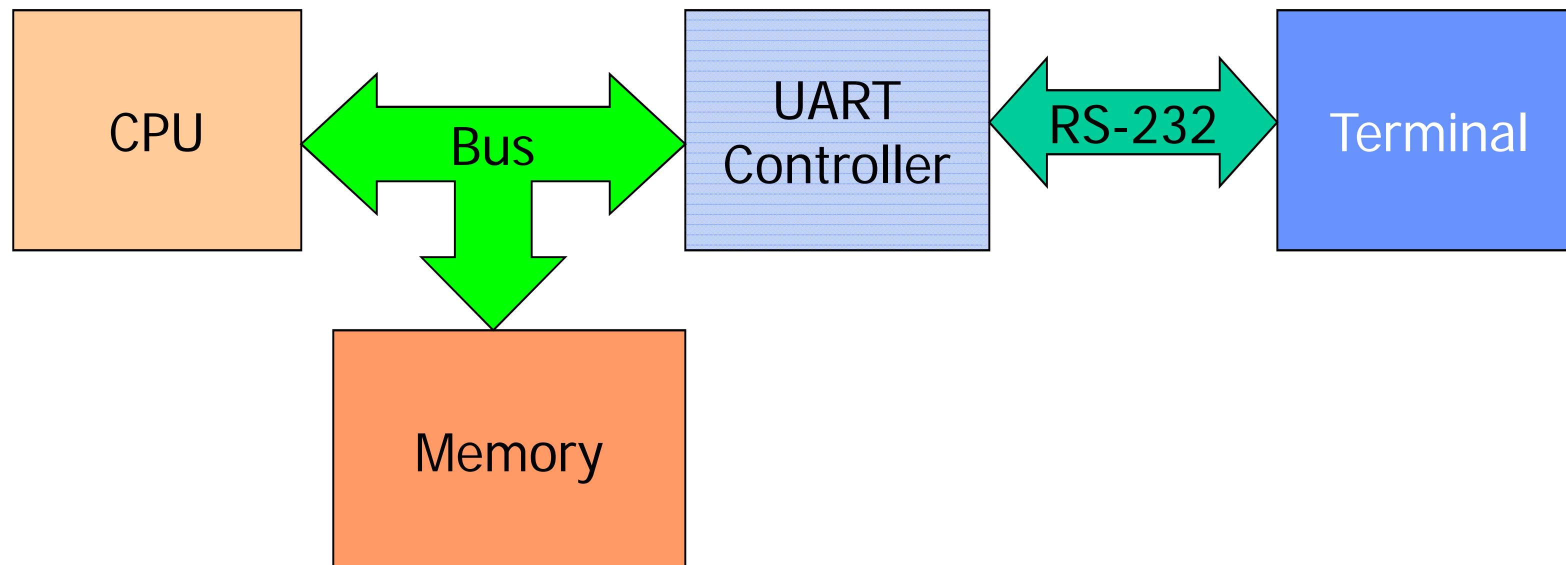
Example Hardware



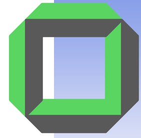
© 2009 University of Karlsruhe, System Architecture Group



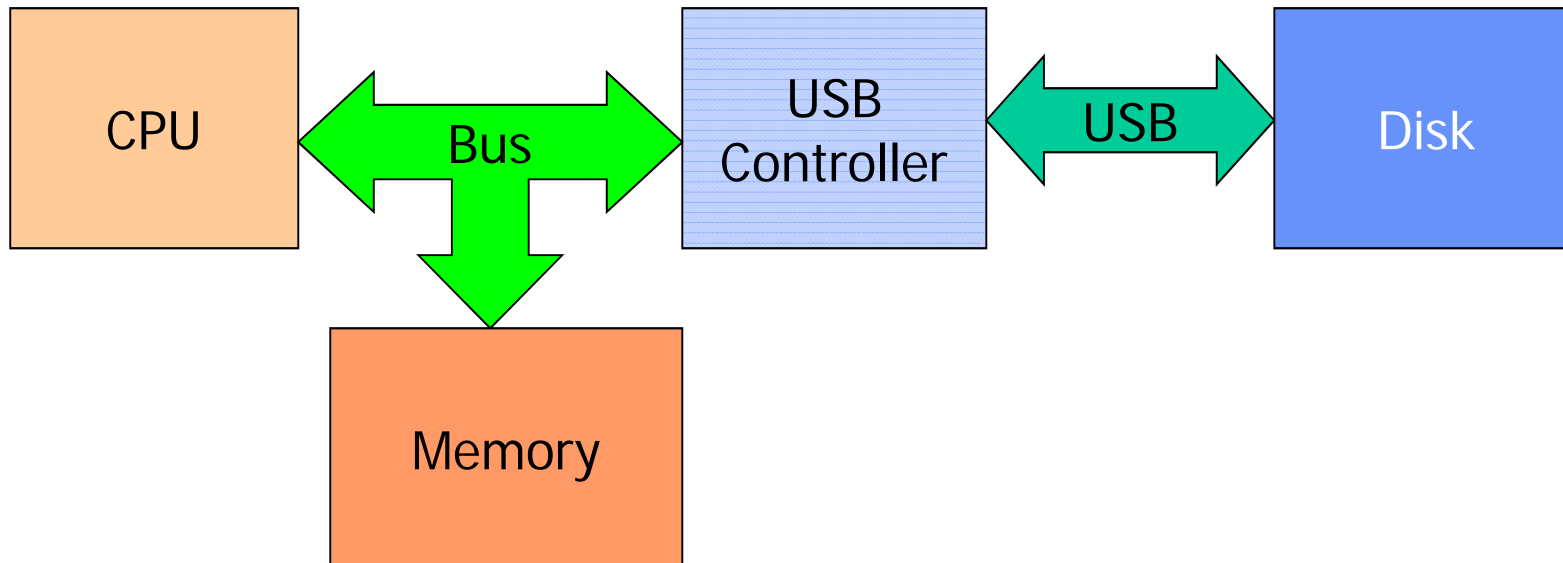
Example Hardware



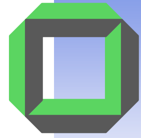
© 2009 University of Karlsruhe, System Architecture Group



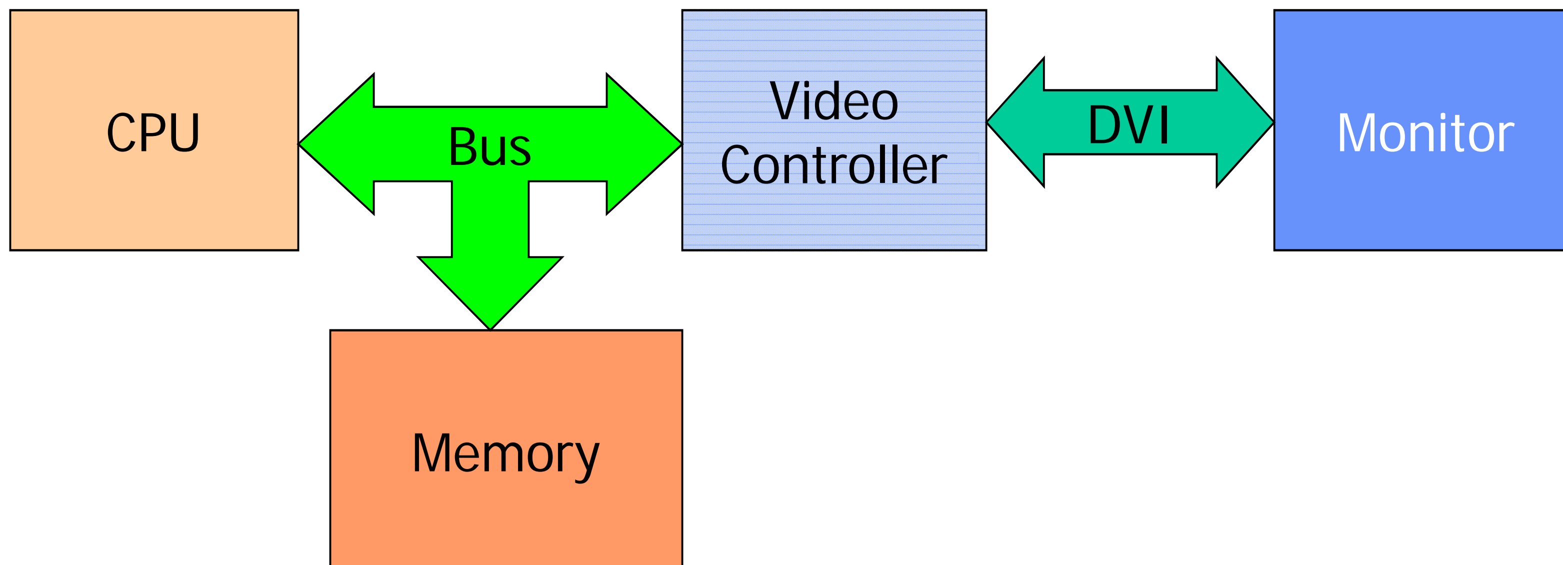
Example Hardware



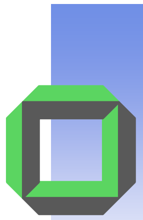
© 2009 University of Karlsruhe, System Architecture Group



Example Hardware



© 2009 University of Karlsruhe, System Architecture Group



Reality:
Intel I/O Controller Hub

Processor

Graphics Controller

Host Controller

Main Memory

Hub Interface

SMBus Device(s)

SMBus

PCI Slots

PCI Bus

AC'97 Codec(s) (optional)

AC'97 2.1

I/O Controller Hub 2

82801BA (ICH2) and 82801BAM (ICH2-M)

PCI Agent

ISA Bridge (optional)

ATA/100/66/33 4 IDE Drives

4xUSB

GPIO

Moon2 Docking Bridge (optional) (ICH2-M)

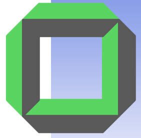
LPC I/F

LAN Controller

Super I/O (required)

FWH

© 2009 University of Karlsruhe, System Architecture Group

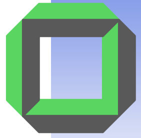


Device Controller Registers

- Writing values
  - Configures the controller
  - Commands the controller to perform I/O actions
  - Provides data needed by the controller
    - E.g. a network packet

- Reading values
  - Identifies controller configuration
  - Queries status information
  - Receives data provided by controller
    - E.g. a network packet

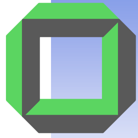
© 2009 University of Karlsruhe, System Architecture Group



Access to Device Registers

- Memory mapped I/O
  - Example: video ram
  - Device register access similar to physical memory
    - Normal load/store instructions used
      - Devices integrate cleanly with programming languages and normal protection mechanisms
    - Widely varied device speeds complicate the approach
      - Avoid slow devices on fast memory buses
    - Caches also complicate the approach
      - Don't cache device registers

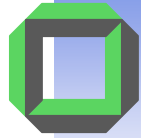
© 2009 University of Karlsruhe, System Architecture Group



Access to Device Registers

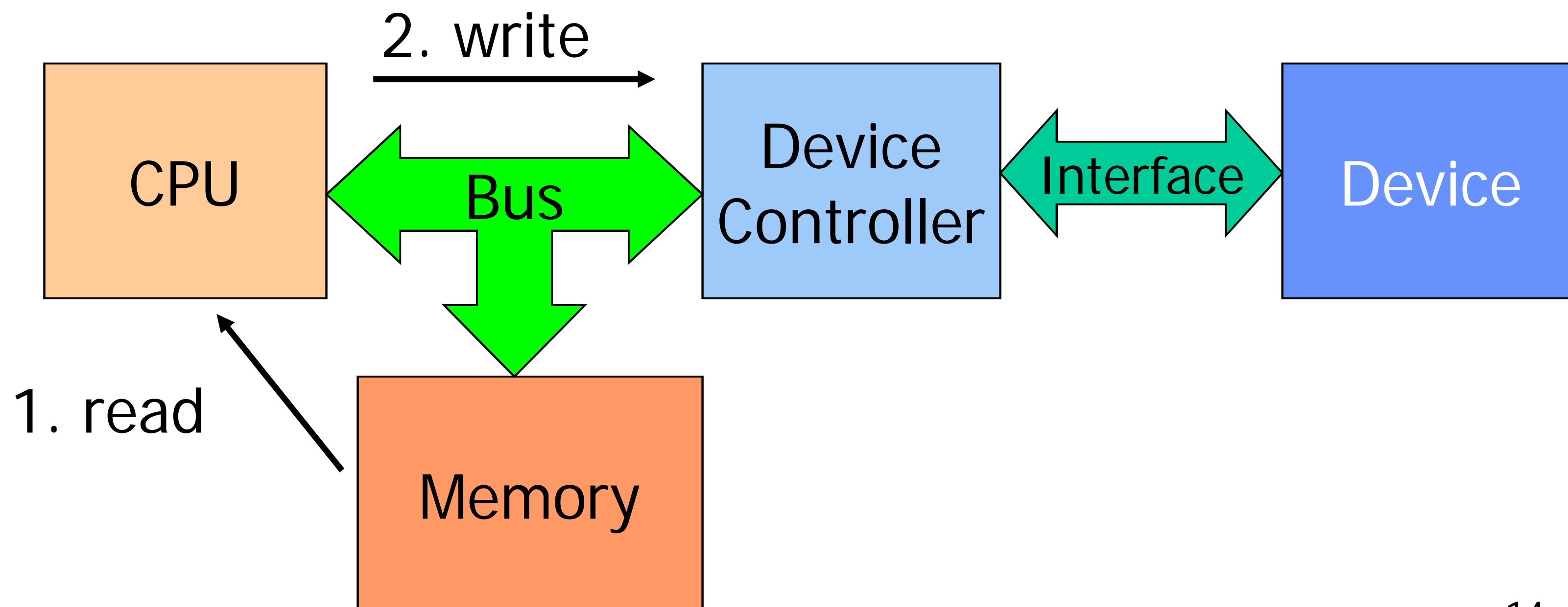
- I/O address space
  - Example: UART registers on standard PC
  - Device register access via special instructions
    - Awkward language integration
      - Must include special instructions in languages
    - Requires additional protection mechanisms
      - Kernel-protected I/O insns., I/O fpages
    - Hardware can adjust behavior for I/O operations
      - Avoid caches, dedicated I/O bus, special bus cycles
- IA-32
  - Combination of both methods

© 2009 University of Karlsruhe, System Architecture Group

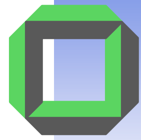


Memory-to-Device Transfers

- CPU reads from memory and writes to device controller
  - Two bus transactions per word transferred
  - CPU is busy
    - Typically fast CPU, slow device

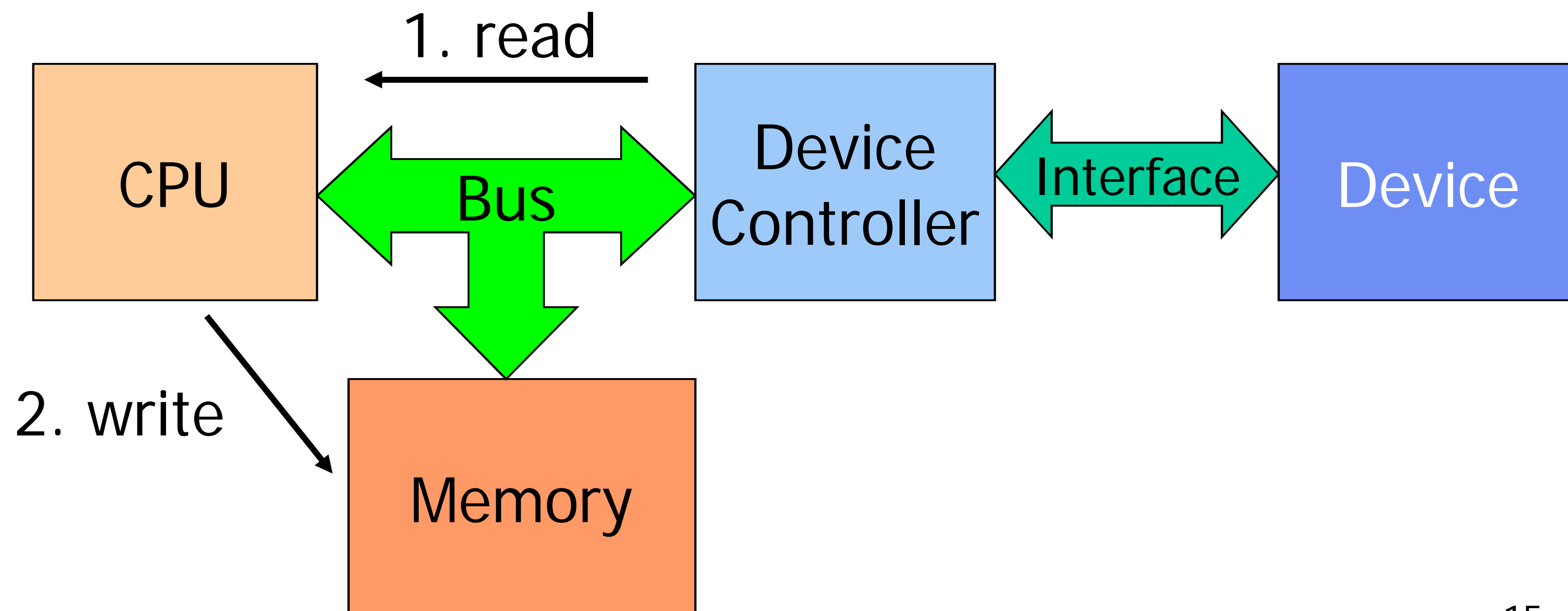


© 2009 University of Karlsruhe, System Architecture Group

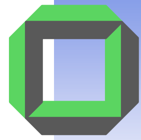


Device-to-Memory Transfers

- CPU reads from device controller and writes to memory
  - Two bus transactions per word transferred
  - CPU is busy
    - Typically fast CPU, slow device

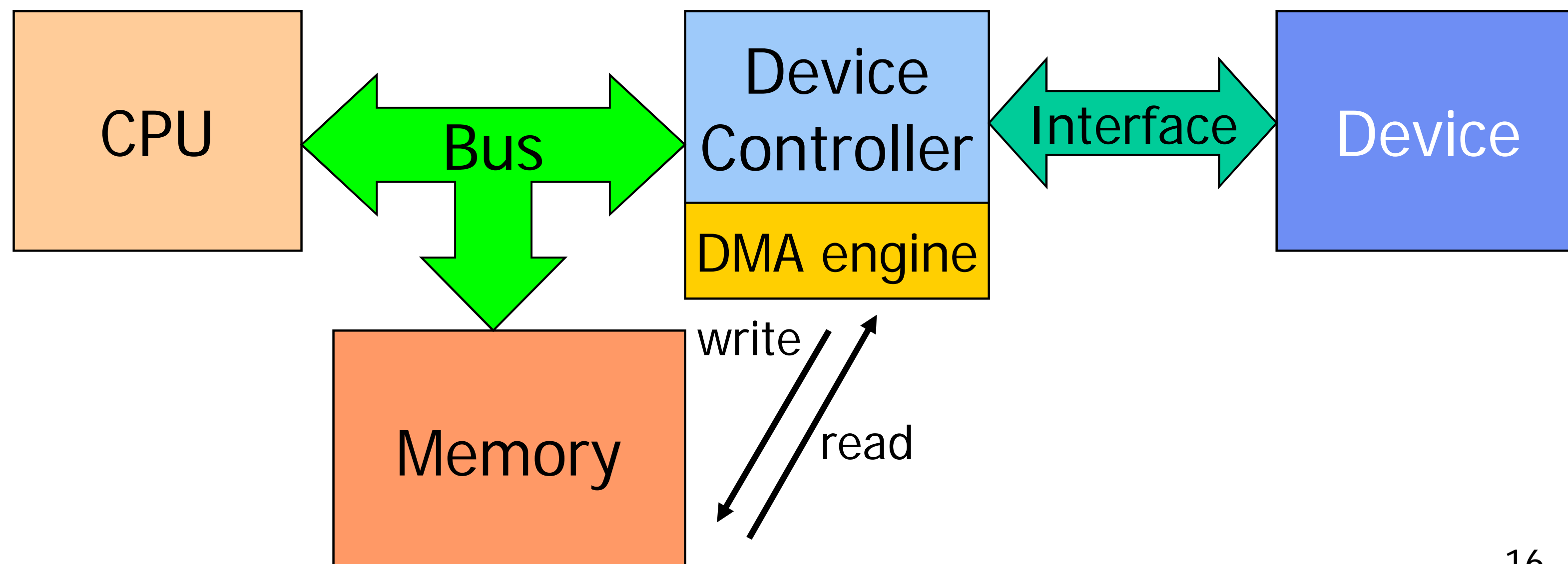


© 2009 University of Karlsruhe, System Architecture Group

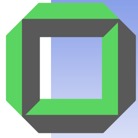


DMA – Direct Memory Access

- DMA engine reads/writes directly to memory
  - CPU specifies a DMA transfer (start address, size, etc.)
  - DMA engine transfers data independently of CPU
    - Cycle stealing, burst-mode
  - At most one bus transaction per word transferred
  - CPU free to process in parallel
    - Assuming no memory contention

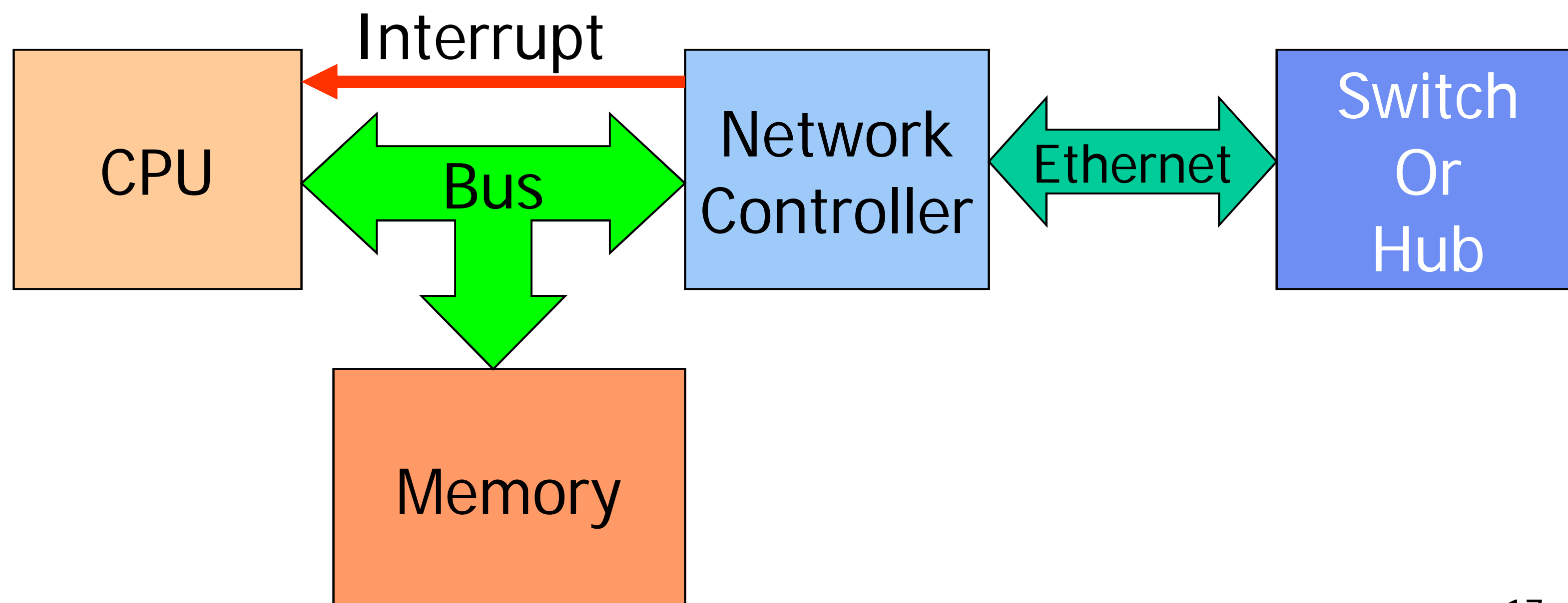


© 2009 University of Karlsruhe, System Architecture Group

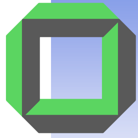


Interrupts

- Used to signal CPU that a device-related event has happened
  - Example: network packet received

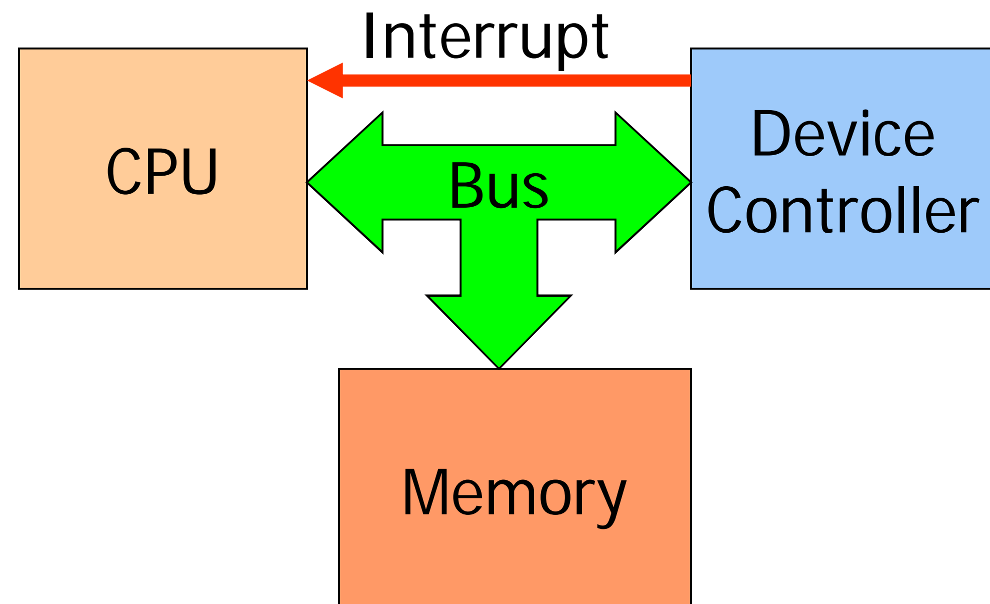


© 2009 University of Karlsruhe, System Architecture Group

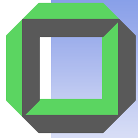


Simple Interrupt Model

- Normal computation is *interrupted*
- Control is transferred to the interrupt service routine
- The interrupt event is processed
- The interrupt is acknowledged
- Normal computation is resumed

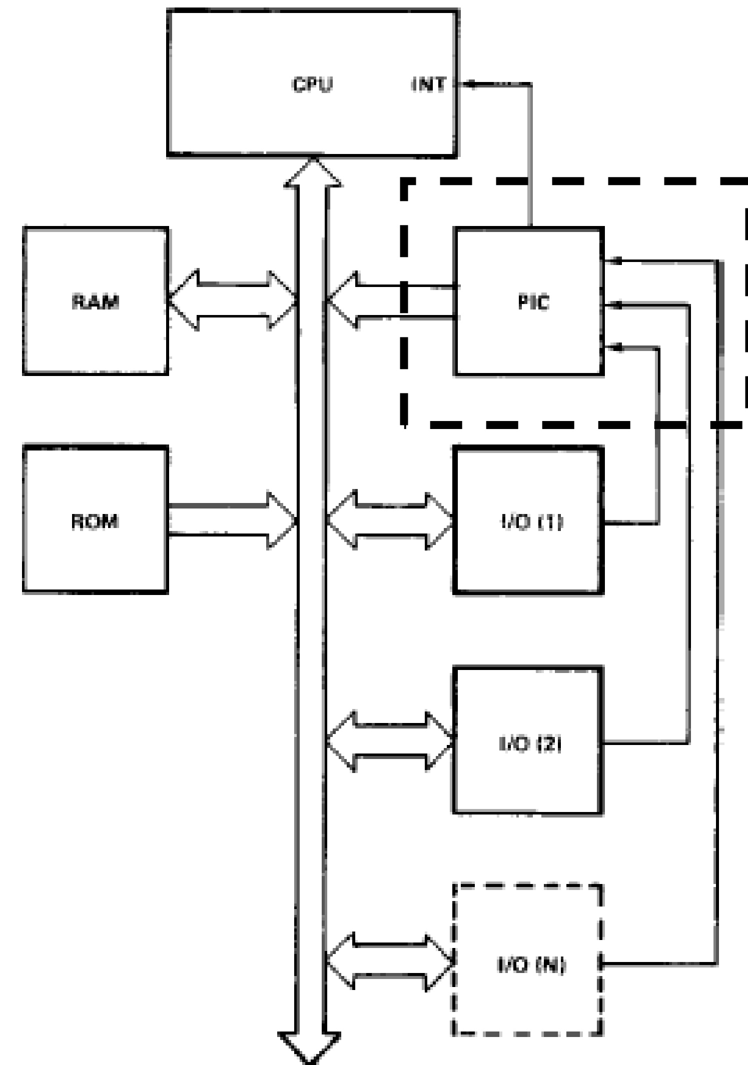


© 2009 University of Karlsruhe, System Architecture Group

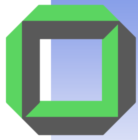


A little closer to reality – PIC

- Programmable interrupt controller
  - Many interrupt sources and few CPU interrupt input lines
  - Prioritizes interrupts from many devices

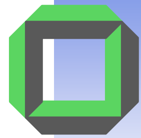


© 2009 University of Karlsruhe, System Architecture Group



PIC in more detail

© 2009 University of Karlsruhe, System Architecture Group



Cascading Interrupt Controllers

ADDRESS BUS (16)

CONTROL BUS

INT REQ

DATA BUS (8)

CS A0 D0-7 INTA INT
CAS 0
8259A
SLAVE A
CAS 1
CAS 2
SP/EN 7 6 5 4 3 2 1 0
GND

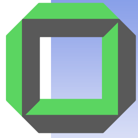
CS A0 D0-7 INTA INT
CAS 0
8259A
SLAVE B
CAS 1
CAS 2
SP/EN 7 6 5 4 3 2 1 0
GND

CS A0 D0-7 INTA INT
CAS 0
8259A
MASTER
CAS 1
CAS 2
SP/EN M7 M6 M5 M4 M3 M2 M1 M0
V_{CC}

7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0

INTERRUPT REQUESTS

© 2009 University of Karlsruhe, System Architecture Group



More recent: APICs and IO-APICs

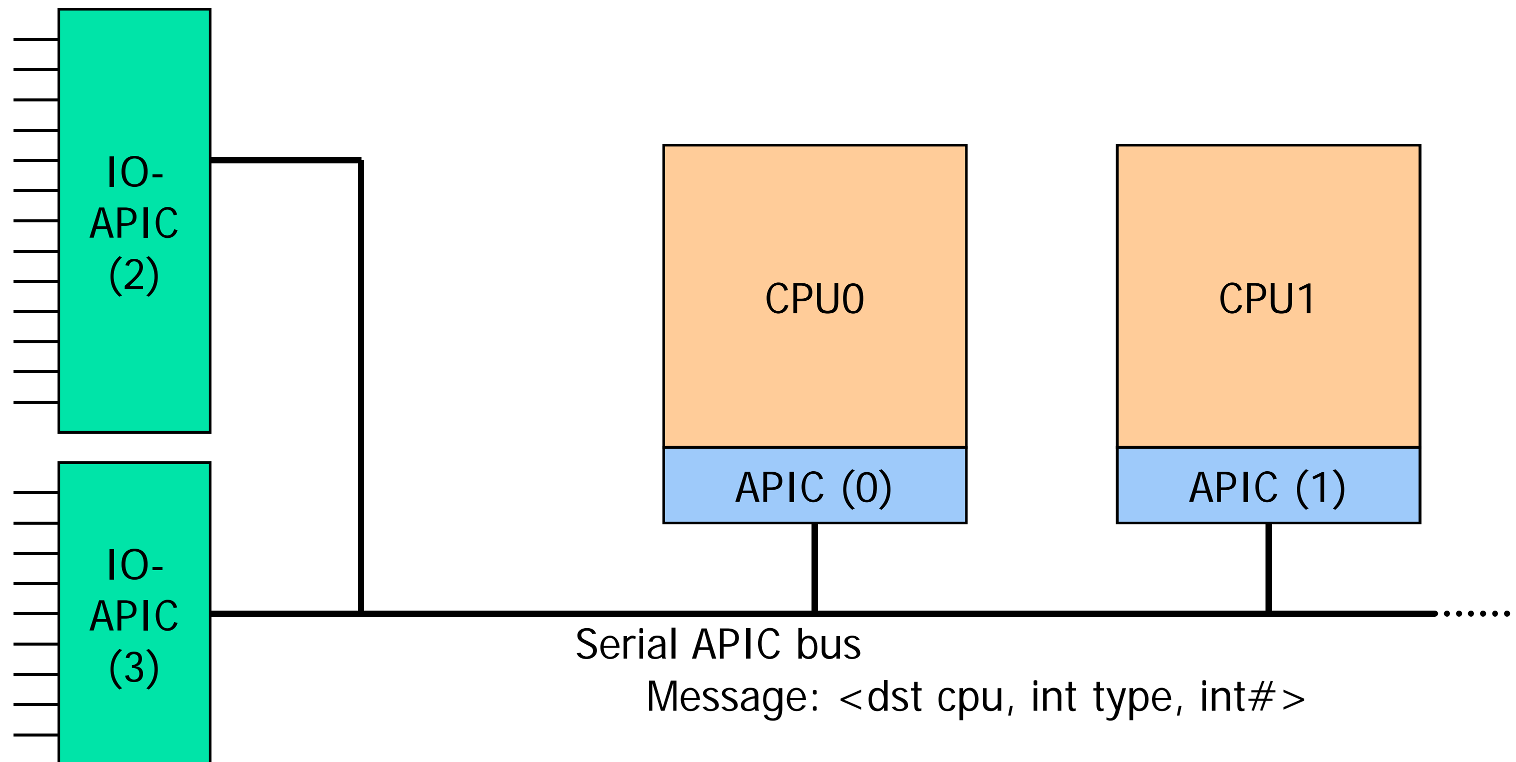


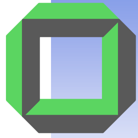
Table:

input PIN ⇒ CPU, int#

Pentium4+:

APIC messages use the memory bus.
Much faster!

© 2009 University of Karlsruhe, System Architecture Group



More recent: APICs and IO-APICs

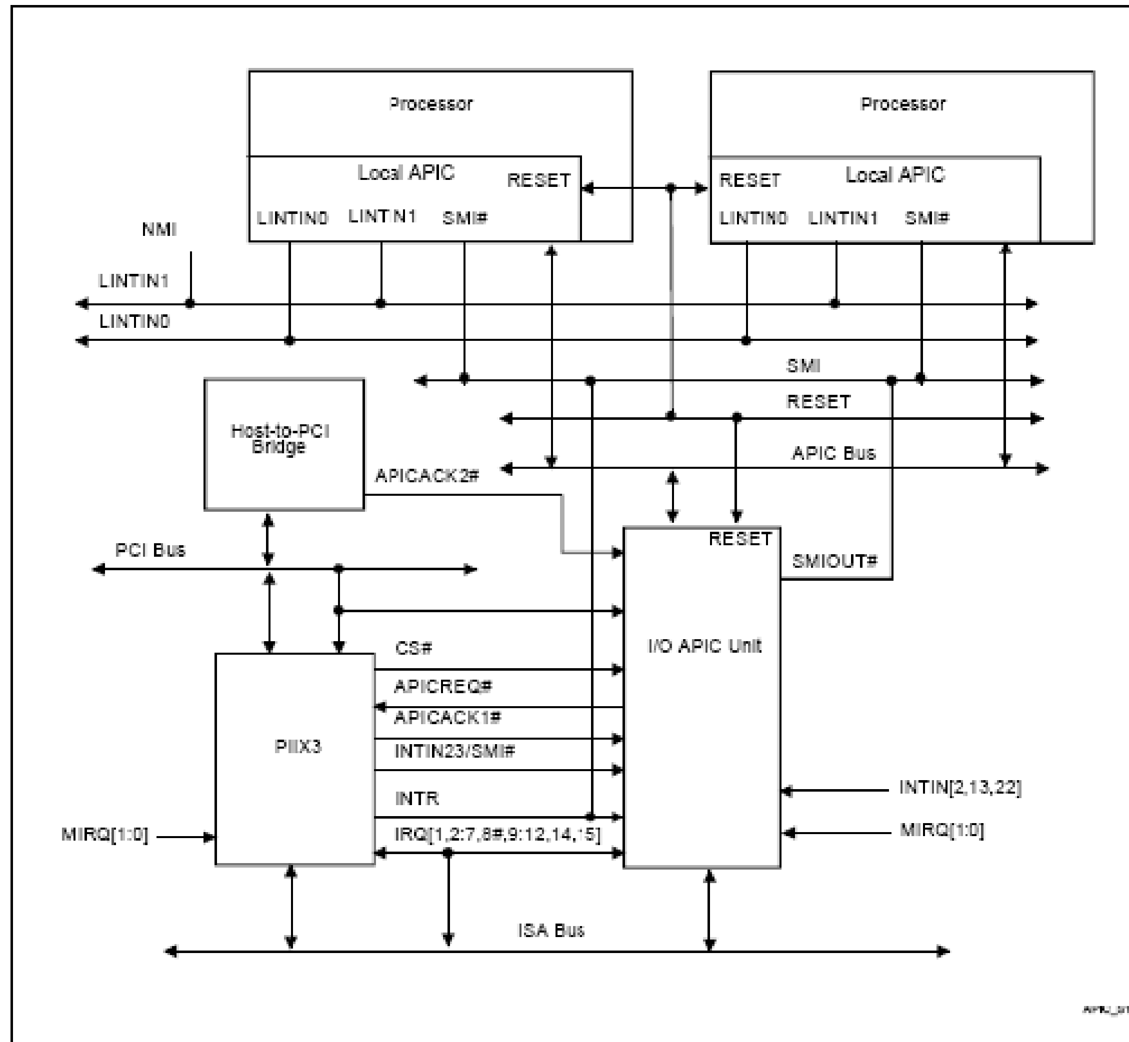
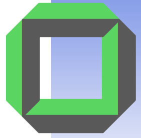


Figure 2. I/O And Local APIC Units

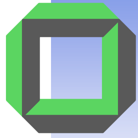
© 2009 University of Karlsruhe, System Architecture Group



L4 Support for Device Drivers

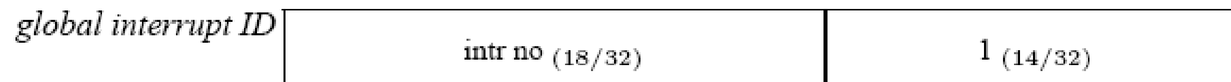
- Memory mapped devices
  - Accessed using normal instructions
  - Control access via virtual memory management functionality
- IA-32 I/O space instructions
  - Available to user-level
  - Can be restricted – I/O flexpages

© 2009 University of Karlsruhe, System Architecture Group



Abstraction: Interrupts as IPC

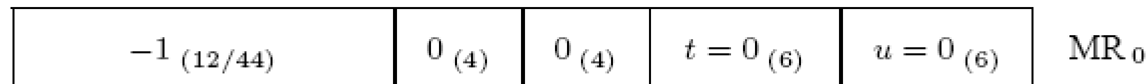
- In-kernel threads represent IRQ line



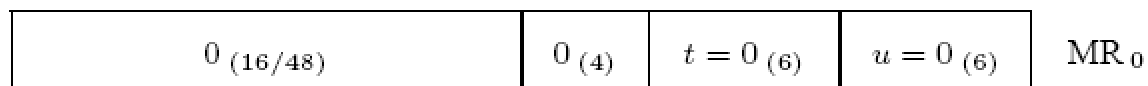
  - System interrupt ID
  - Interrupt handlers threads can register for interrupts

    `AssociateInterrupt (ThreadId InterruptTID, HandlerTID)`

- Interrupts are transformed into IPC messages
  - In-kernel thread performs IPC call to handler thread



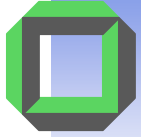
    - Kernel disables IRQ on PIC after IRQ fires
  - Interrupt handler replies to acknowledge interrupt



    - IRQ gets re-enabled on PIC

© 2009 University of Karlsruhe, System Architecture Group

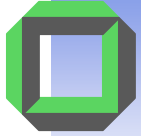
Software



I/O Methods

- Programmed I/O
- Interrupt-driven I/O
- DMA-based I/O

© 2009 University of Karlsruhe, System Architecture Group



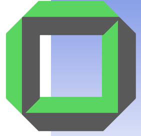
Programmed I/O

Output system call code

```
copy_from_user(buffer,p,count);
for (i = 0; i < count; i++) {
   while (*UART_status_reg != ready);
   *UART_data reg = p[i];
}
```

- Problem
  - If device is slower than CPU, CPU spends all its time *polling* or *busy waiting*

© 2009 University of Karlsruhe, System Architecture Group



Interrupt-driven I/O

Output system call code

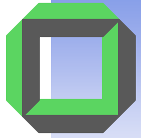
```
copy_from_user(buffer,p,count);
enable_interrupts();
while (*UART_status_reg != ready);
*UART_data_reg = p[i];
scheduler();
```

Interrupt service routine

```
if (count == 0) {
   unblock_user();
} else {
   *UART_data_reg = p[i];
   count--;
   i++;
}
acknowledge_interrupt();
return_from_interrupt();
```

- CPU can perform other work while “waiting”
- However, interrupts are not for free!!!

© 2009 University of Karlsruhe, System Architecture Group



DMA-based I/O

Output system call code

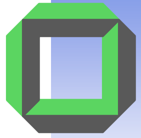
```
copy_from_user(buffer,p,count);
setup_DMA_controller();
trigger_device_operation();
scheduler();
```

Interrupt service routine

```
acknowledge_interrupt();
unblock_user();
return_from_interrupt();
```

- Avoids interrupt per transfer
- One interrupt per *buffer*

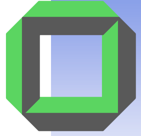
© 2009 University of Karlsruhe, System Architecture Group



Device Independence

- Avoid writing different applications for each device
- Categorize device into similar classes or types
  - Block versus character devices (?)
  - Disk, CDROM, DVD, Tape
  - Network
- Device drivers implement a standard interface for their device class
  - Abstracts from the details of particular hardware
  - Hides implementation details of managing the device

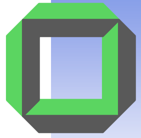
© 2009 University of Karlsruhe, System Architecture Group



Synchronous/Asynchronous Access

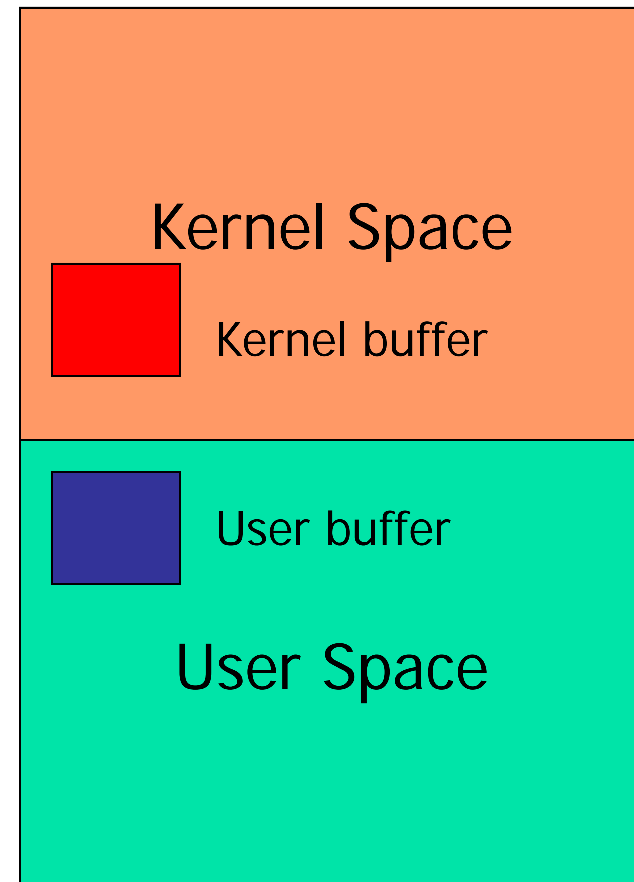
- Classical applications expect synchronous access to a device
  - Application waits for read() result
  - Easy error recovery
- Devices are usually of asynchronous nature
- Operating System must reconcile these differences
  - Application blocking
  - Asynchronous *signals*
  - WaitForObject, WaitForMultipleObjects

© 2009 University of Karlsruhe, System Architecture Group

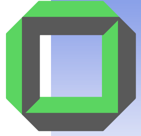


(Double) Buffering

- Write
  - Preserve data until I/O completes while app can continue
  - Batch enough data to supply device data rate
- Read
  - Store data until destination identified
    - E.g., network
  - Combine received data to a single *packet.*
    - E.g., avoid unblock per character
- However, copying is inefficient



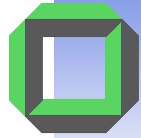
© 2009 University of Karlsruhe, System Architecture Group



Shareable vs. Exclusive Access

- Some devices are obviously shareable between users
  - Disk, network, ...
- Other device only make sense when used by one user at a time
  - Tape drive, CD-writer, ...
- OS must manage access to devices
  - Avoid deadlocks etc., ...

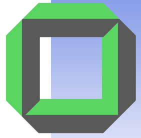
© 2009 University of Karlsruhe, System Architecture Group



Typical Software Model – Layers

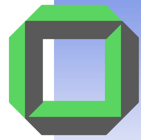
Layer	Example
User-Level I/O Software	Read/write on device
Device-independent OS software	Block device management
Device Driver	IDE driver
Interrupt Service Routine	IDE Interrupt Service Routine
Hardware	

© 2009 University of Karlsruhe, System Architecture Group



Problem

- The ISR unblocks the blocked thread
  - What happens if significant processing is required to unblock the thread?
  - We may miss the next interrupt!!!
- Particularly problematic when *burst-rate* can be much higher than processing rate (throughput).

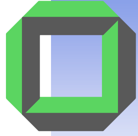


Typical Software Model

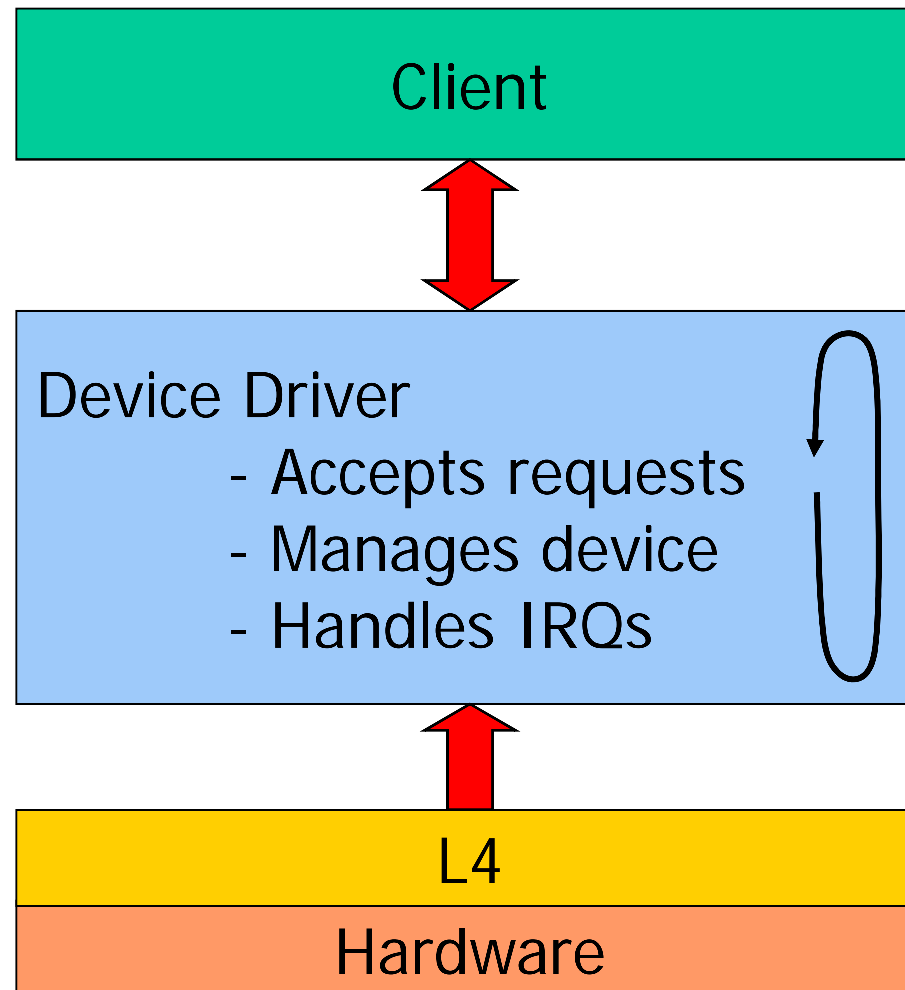
- ISR is as short as possible
  - Longer processing deferred until after burst
  - Linux: bottom half or tasklets
    Windows NT: deferred procedure calls

Layer		Example
User-Level I/O Software		Read/write on socket
Device-independent OS software		IP stack
Device Driver	Deferred ISR	Deferred packet processing
Interrupt Service Routine		Net ISR
Hardware		

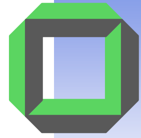
© 2009 University of Karlsruhe, System Architecture Group



Single-Threaded Device Driver



© 2009 University of Karlsruhe, System Architecture Group



Multi-Threaded Device Driver

Client

IPC/synchronization

Device Driver

Server Thread

Deferred ISR Thread

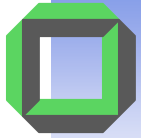
Interrupt Service Thread

L4

Hardware

© 2009 University of Karlsruhe, System Architecture Group

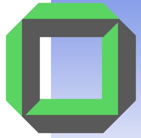
Screen and Keyboard in a PC



How to display something?

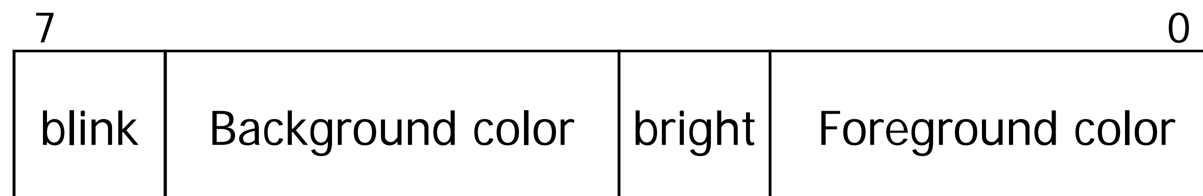
- Display controller
  - Human Interface Device
    - Converts contents of “virtual screen” into electrical signals for monitor
    - Video memory – addressable by CPU
    - Monitor image – readable by human
  - Supports different modes – Text vs. GFX
    - Mode defines interpretation of video memory contents – characters vs. pixels

© 2009 University of Karlsruhe, System Architecture Group



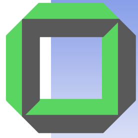
VGA Display

- VGA BIOS initializes text mode 80x25
- Video memory starts at physical address 0xb8000
- Memory format – 2 bytes per character
  - Address x+0: ASCII character code
  - Address x+1: Color code



  - 80 characters per line (160 bytes)
- Cursor control via controller registers

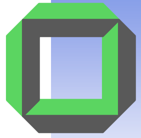
© 2009 University of Karlsruhe, System Architecture Group



How to type something?

- Keyboard controller
  - HID – converts depress/release of labeled key to scan-code
- Keyboard is now a PS/2 bus device
  - Shares bus with PS/2 mouse
  - Chipset simulates an i8042 controller
    - Status register: read IO port 0x64
    - Control register: write IO port 0x64
    - Data register: read/write IO port 0x60

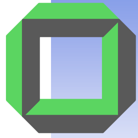
© 2009 University of Karlsruhe, System Architecture Group



i8042 Keyboard Controller

- Status register
  - Bit0: 1 = Output buffer full
  - Bit1: 1 = Input buffer full
- Data register
  - Bit0-6 = scan code
  - Bit7: 0 = key pressed
1 = key released
- Need to translate scan code to ASCII
- Shift keys – maintain state

© 2009 University of Karlsruhe, System Architecture Group



i8042 Keyboard Controller

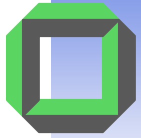
- Simple read loop:

```
kbdRead:
WaitLoop:     in      al, 64h       ; Read Status byte
              and     al, 10b       ; Test IBF flag (Status<1>)
              jz      WaitLoop      ; Wait for IBF = 1
              in      al, 60h       ; Read input buffer
```

- Simple write loop:

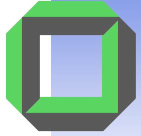
```
kbdWrite:
WaitLoop:     in      al, 64h       ; Read Status byte
              and     al, 01b       ; Test OBF flag (Status<0>)
              jnz     WaitLoop      ; Wait for OBF = 0
              out     60h, cl       ; Write to output buffer
```

© 2009 University of Karlsruhe, System Architecture Group



Console Server

- Allow clients to
  - Read from keyboard
  - Write to screen
- Focus client
  - Which client has the keyboard?
  - How to switch clients?



What's missing in SDIOS?

- Shell
  - Goal: Launch new programs from a minimal shell
    - Exercise file system, task server, pager, console
- Furthermore ...
  - Disk driver
  - File systems
  - Network driver
  - USB driver
  - IEEE1394 driver
  - ... Feel free to go wild!

© 2009 University of Karlsruhe, System Architecture Group