TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Systemarchitektur, Professur für Betriebssysteme

OPERATING-SYSTEM CONSTRUCTION

Material based on slides by Olaf Spinczyk, Universität Osnabrück

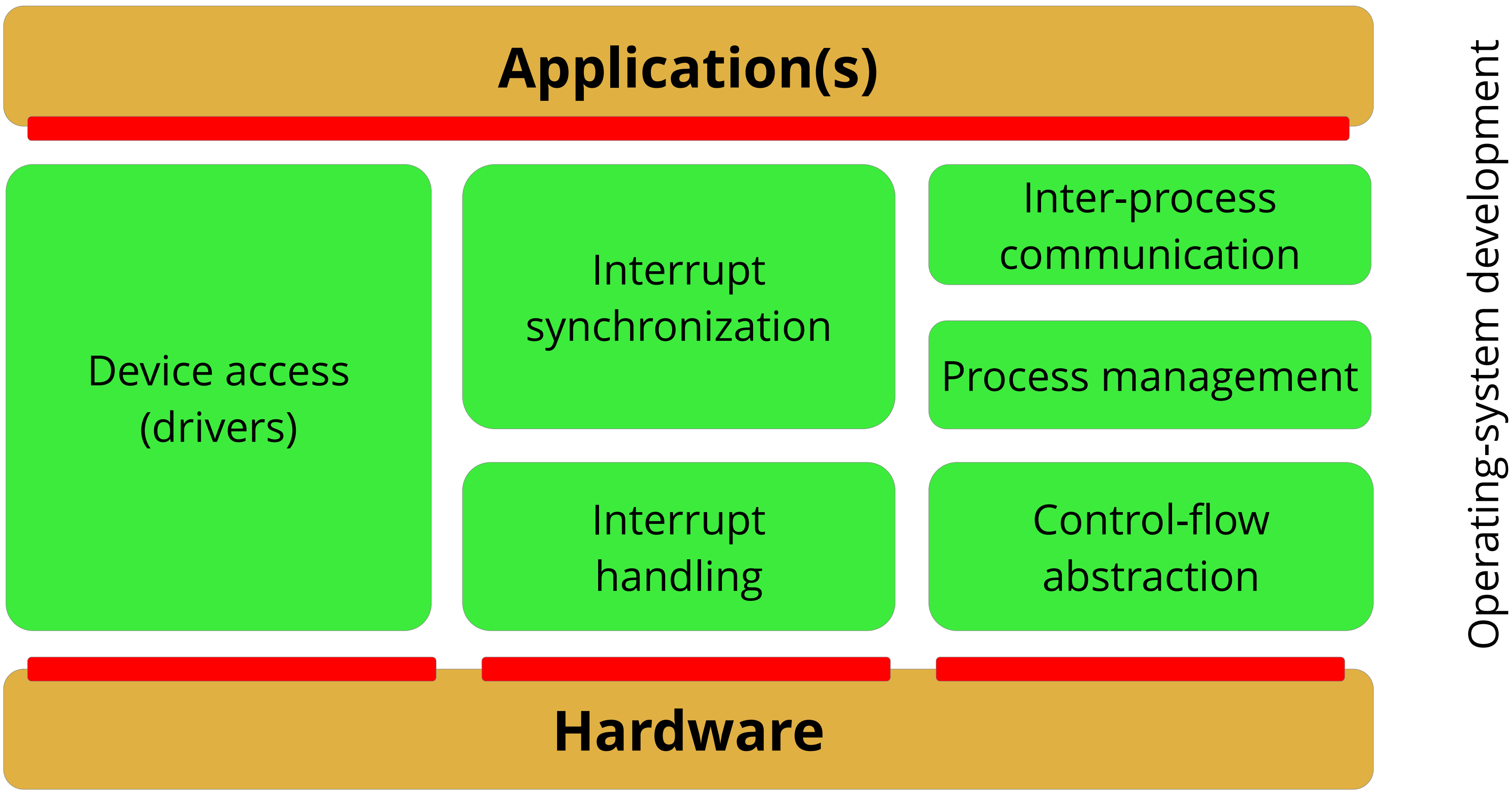
Interrupts – Hardware

https://tud.de/inf/os/studium/vorlesungen/betriebssystembau

HORST SCHIRMEIER

Overview: Lectures

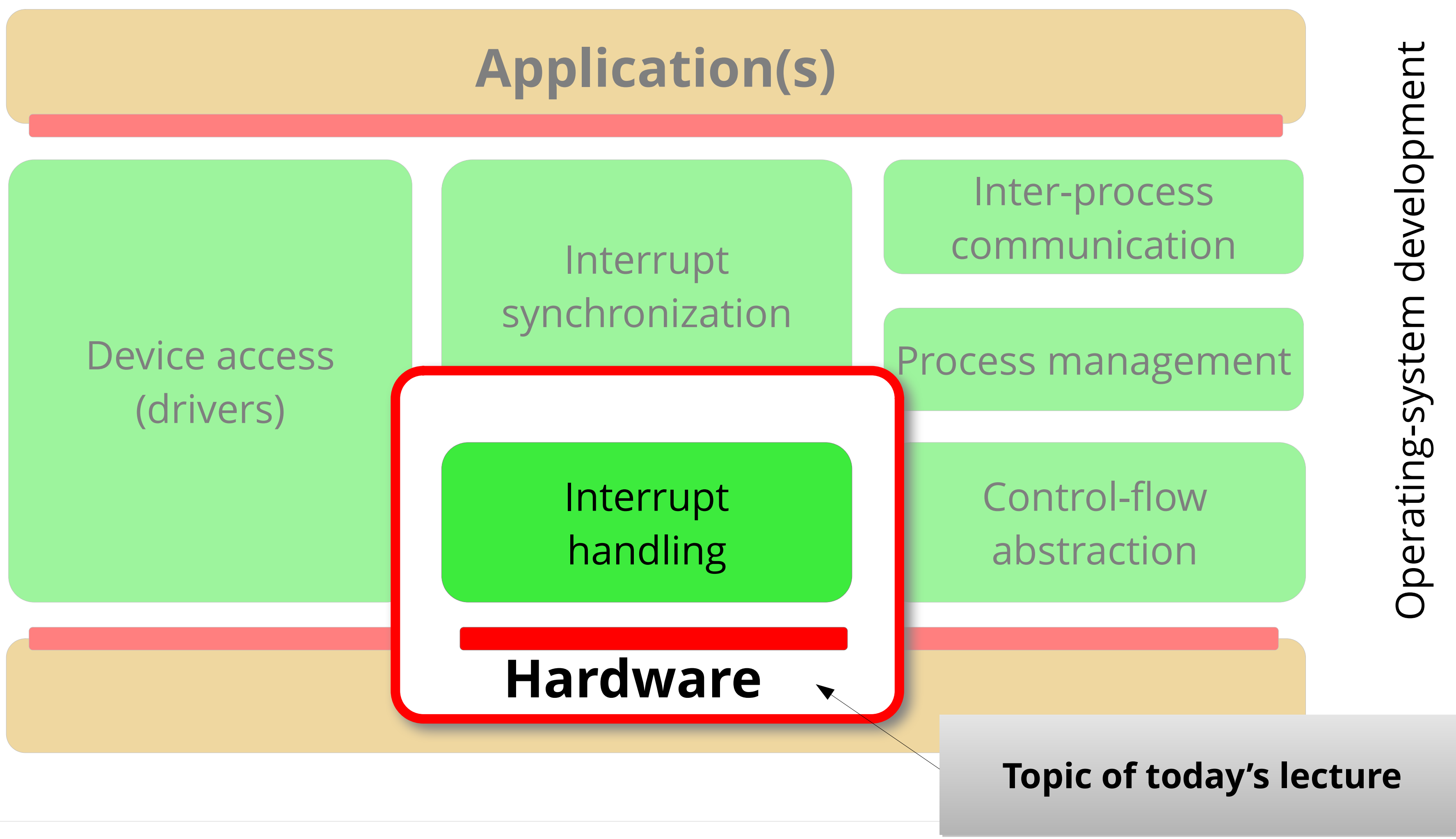
Structure of the "OO-StuBS" operating system:



TECHNISCHE UNIVERSITÄT DRESDEN

Overview: Lectures

Structure of the "OO-StuBS" operating system:



Overview

- Interrupts
  - Purpose
- General Discussion
  - Prioritization, Lost Interrupts, Dispatch, Saving State, Nested Interrupts, Interrupts in Multiprocessor Systems
- Hazards
  - “Spurious Interrupts”, “Interrupt Storms”
- Hardware-Architecture Examples
  - Motorola 68K, Pentium APIC

Overview

- **Interrupts**
  - Purpose
- General Discussion
  - Prioritization, Lost Interrupts, Dispatch, Saving State, Nested Interrupts, Interrupts in Multiprocessor Systems
- Hazards
  - “Spurious Interrupts”, “Interrupt Storms”
- Hardware-Architecture Examples
  - Motorola 68K, Pentium APIC

Purpose of Interrupts

Looking back in history …

- Overlapped I/O
  - Input: Wasting CPU cycles by (unpredictably long) **busy waiting**
  - Output: Autonomous device behavior (e.g. **DMA**) unloads CPU
- Time sharing
  - Timer interrupts allow the operating system to …
    - **preempt processes**
    - run time-driven activities

Overview

- Interrupts
  - Purpose
- **General Discussion**
  - Prioritization, Lost Interrupts, Dispatch, Saving State, Nested Interrupts, Interrupts in Multiprocessor Systems
- Hazards
  - "Spurious Interrupts", "Interrupt Storms"
- Hardware-Architecture Examples
  - Motorola 68K, Pentium APIC

Prioritization

- **Problem:**
  - Multiple interrupt requests can be signaled **at once**. *Which one is more important?*
  - While the CPU handles the most important request, **further requests** can be signaled.
- **Solution:** a **prioritization mechanism** …
  - **in software:** The CPU only has one IRQ (interrupt request) line and queries/services devices in a defined order.
  - **in hardware:** A prioritization circuit assigns priorities to devices and only forwards the most urgent request for handling.
  - **with static priorities:** each device statically gets assigned a priority
  - **with dynamic priorities:** priorities can be changed dynamically, e.g. cyclic

Lost Interrupts

- **Problem:**
  - During interrupt handling, and/or while interrupts are disabled, the CPU **cannot handle new interrupts**.
  - Memory for IRQs is (very!) limited
    - usually 1 bit per interrupt line
- **Solution:** in **software**
  - Interrupt handler routine should be **as (temporally) short as possible** to minimize probability for lost interrupts.
  - Interrupts should **not be disabled** longer than necessary by the CPU.
  - A device driver must handle the situation that an interrupt **signals more than one completed I/O operation**.

Interrupt Dispatch

- **Problem:**
  - Determine with little effort **which device** triggered the interrupt
    - **Sequential querying:**
      Time-consuming, modifies state of I/O buses and uninvolved devices
- **Solution: Interrupt vector**
  - Assign a number to each interrupt → index into vector
    - Vector number not necessarily related to priority
    - In practice, devices may have to share a vector number (**interrupt chaining**)
  - CPU-specific vector-table structure
    - Usually contains **pointers to functions**, rarely machine instructions

Saving State

- **Problem:**
  - After running the handler routine, we want to **return** to normal context
  - **Transparency:** Interrupt handling supposed to happen unnoticed
- **Solution: State save**
  - by hardware
    - Only essential state: e.g. return address and status register
    - State restore by special instruction, e.g. **`IRET`**, **`RTE`**, …
  - by software
    - Interrupts may occur at any time → handler routine also must save and restore state

Nested Interrupt Handling

- **Problem:**
  - To react promptly to important events, interrupt handlers should be **interruptible**.
  - … but we should avoid unlimited nesting. (Why?)
- **Solution:**
  - CPU only allows interrupts with *higher* priority
  - Current priority in status register
  - Previous priority on a stack

Multiprocessor Systems

- **Problem:**
  - Each interrupt can only be handled by one CPU. But which one?
  - Additional interrupt category: **Inter-processor interrupts** (IPIs)
- **Solution:** More complex interrupt-handling hardware for multiprocessors; design variants:
  - **static** destination
  - **random** destination
  - **programmable** destination
  - destination depending on **current CPU load**

  ... and combinations thereof.

Overview

- Interrupts
  - Purpose
- General Discussion
  - Prioritization, Lost Interrupts, Dispatch, Saving State, Nested Interrupts, Interrupts in Multiprocessor Systems
- **Hazards**
  - “Spurious Interrupts”, “Interrupt Storms”
- Hardware-Architecture Examples
  - Motorola 68K, Pentium APIC

Hazard: Spurious Interrupts

- **Problem:** Interrupt-handling mechanism can be presented with spurious* interrupts, caused e.g. by ...
  - Hardware errors
  - Incorrectly programmed devices
- **Solution:**
  - Avoid hardware and software errors 🙂
  - Program OS "defensively"
    - expect spurious interrupts

* "spurious" ≈ „falsch", „unecht"

Hazard: Interrupt Storms

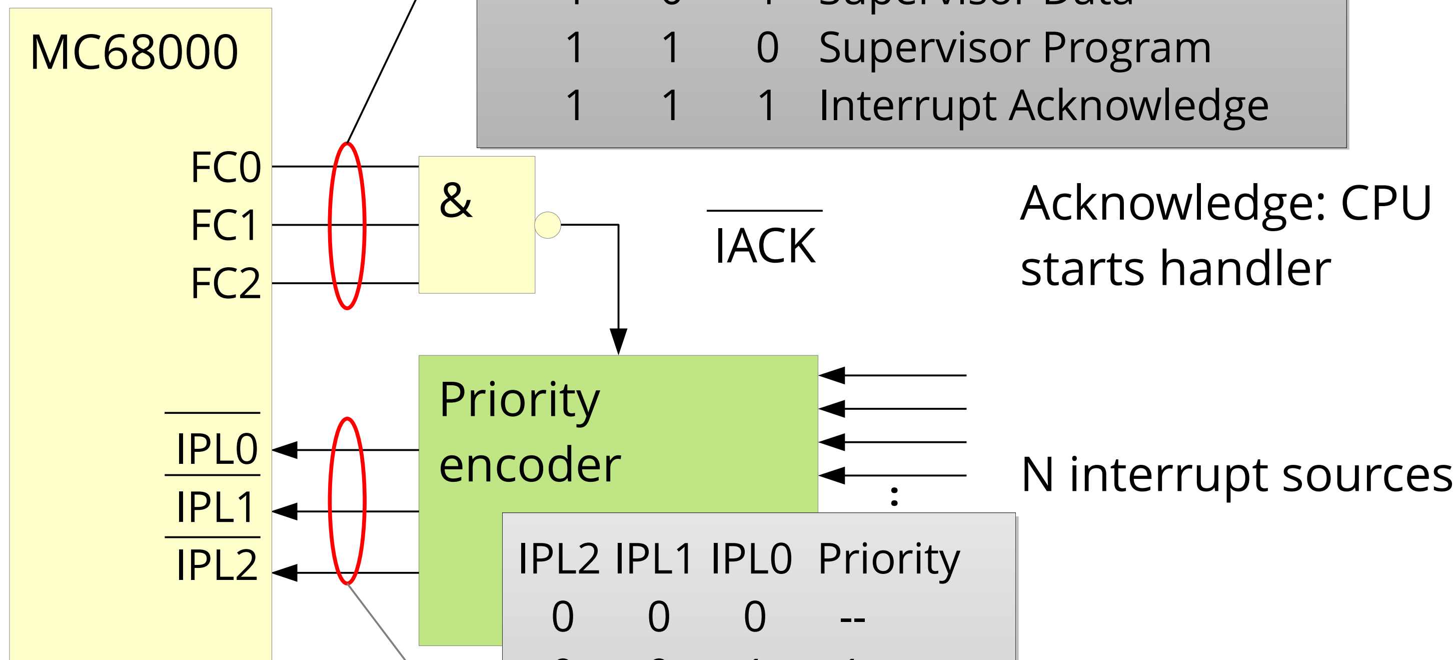
- **Problem:**
  - High interrupt frequency can overload or “freeze” a computer
  - **Cause:** Spurious interrupts, or too high I/O load
  - Can be mistaken for **thrashing** (similar symptoms).
- **Solution:** in the OS
  - Detect interrupt storms
  - Deactivate culprit device

Overview

- Interrupts
  - Purpose
- General Discussion
  - Prioritization, Lost Interrupts, Dispatch, Saving State, Nested Interrupts, Interrupts in Multiprocessor Systems
- Hazards
  - “Spurious Interrupts”, “Interrupt Storms”
- **Hardware-Architecture Examples**
  - Motorola 68K, Pentium APIC

Interrupts in the MC68000

Interrupts in the

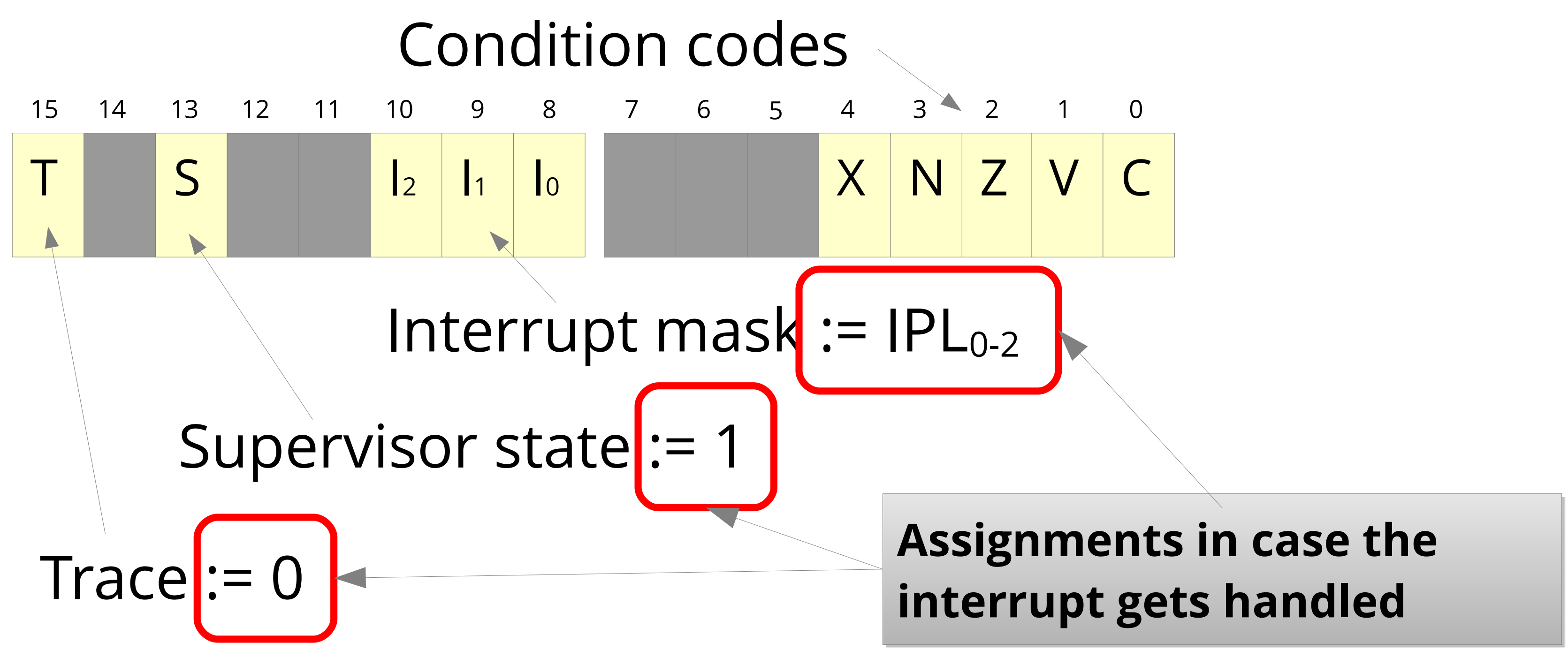


FC2	FC1	FC0	Address Space Type
0	0	0	*reserved*
0	0	1	User Data
0	1	0	User Program
0	1	1	*reserved*
1	0	0	*reserved*
1	0	1	Supervisor Data
1	1	0	Supervisor Program
1	1	1	Interrupt Acknowledge

IPL2	IPL1	IPL0	Priority
0	0	0	--
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7 (NMI)

MC68000 Status Register (SR)

- Contains current **interrupt mask** (among other things)
  - Interrupt → CPU tests whether $IPL_{0-2} > I_{0-2}$i
    No? → Interrupt is inhibited (for now).
  - However, interrupt with $IPL_{0-2} = 7$ is always handled (NMI)



TECHNISCHE UNIVERSITÄT DRESDEN

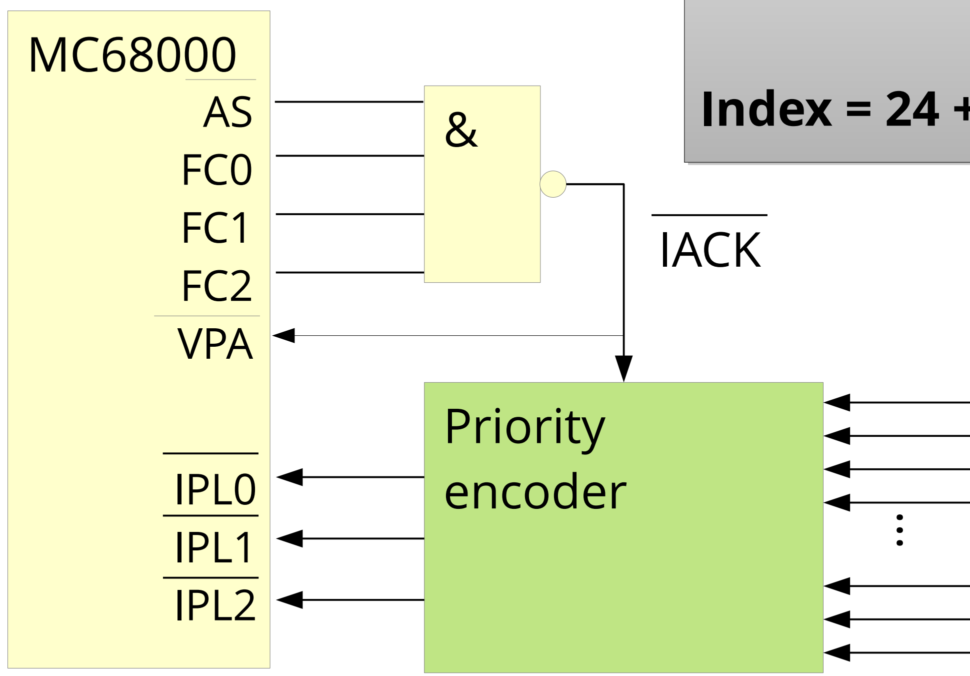
MC68000 Interrupt Vectors

Index	Address	Assignment
0	0x000	Reset: Initial Supervisor Stack Pointer
1	0x004	Reset: Initial PC
2	0x008	Bus Error
3	0x00c	Address Error
4	0x010	Illegal Instruction
5	0x014	Zero Divide
…		
24	0x060	Spurious Interrupt
25	0x064	**Level 1 Interrupt Autovector**
26	0x068	**Level 2 Interrupt Autovector**
…		
30	0x078	**Level 6 Interrupt Autovector**
31	0x07c	**Level 7 Interrupt Autovector (NMI)**
32-47	0x080	TRAP Instruction Vectors
48-63	0x0c0	*reserved*
64-255	0x100	**User Interrupt Vectors**

Autovectored Interrupts

External circuitry signals via **VPA** that the CPU should calculate the vector number **automatically**:

Index = 24 + IPL_{0-2}



Acknowledge: CPU starts handler

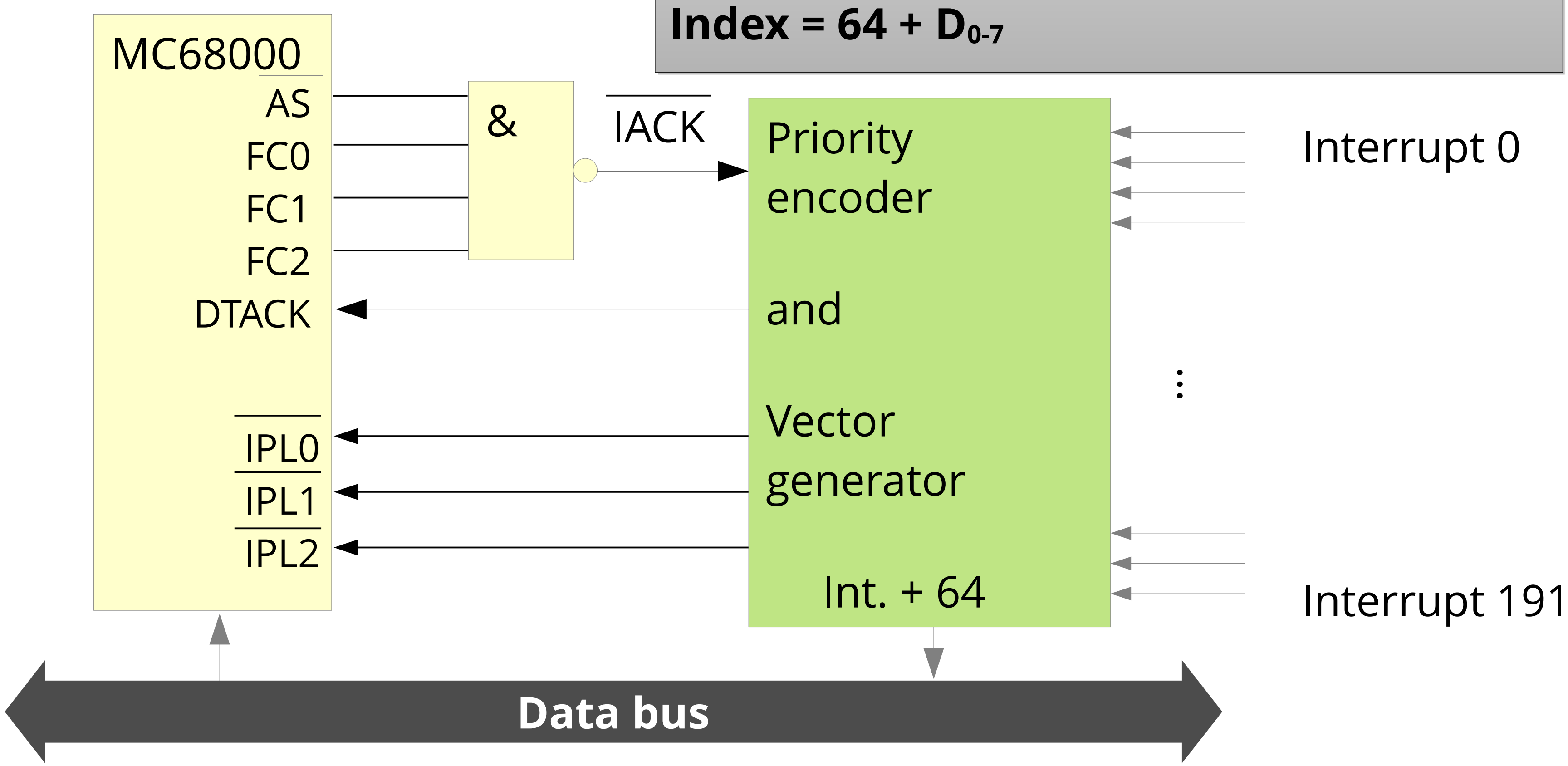
N interrupt sources

Problem: Only 6 vectors available. With more devices, **sharing** is unavoidable.

Non-Autovectored Interrupts

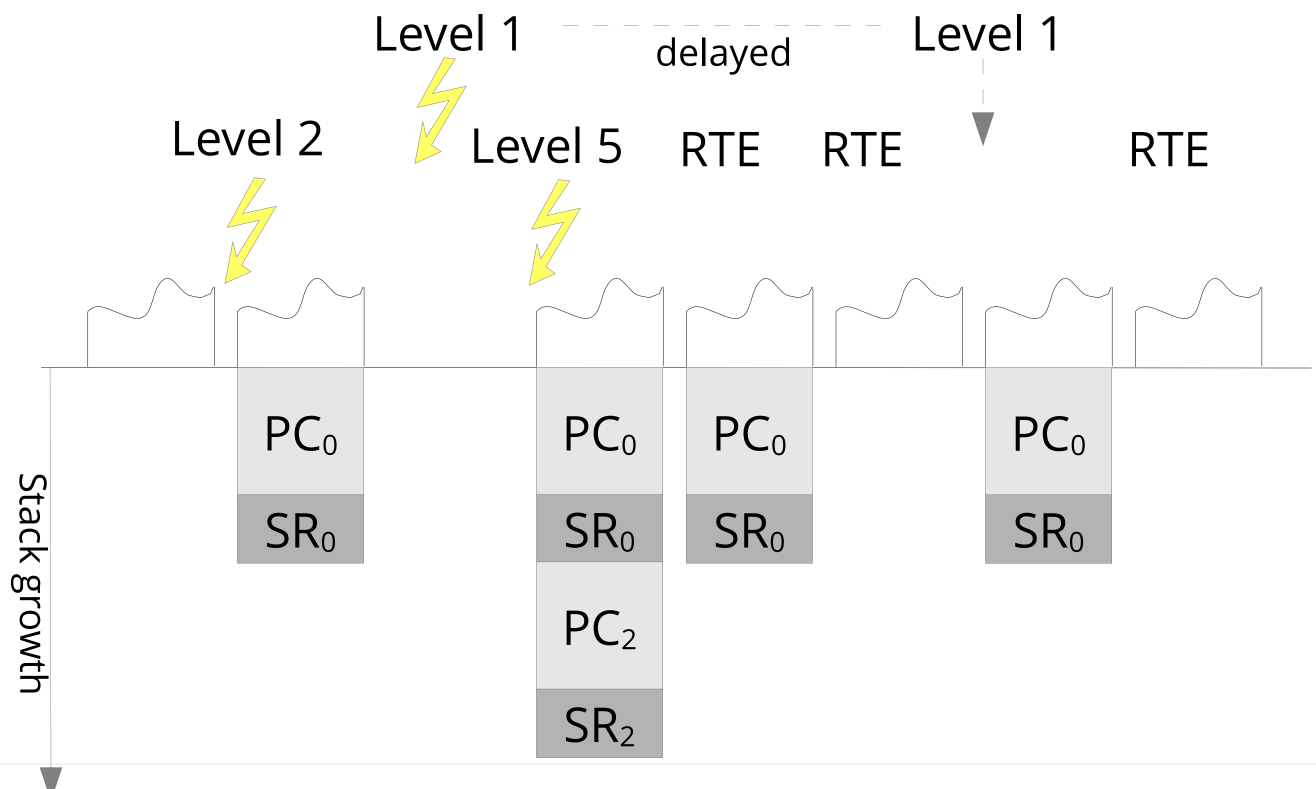
External circuitry signals via **DTACK** that the CPU should read the vector number **from the data bus**.

Index = 64 + $D_{0\text{-}7}$



MC68000 State Save

- Previous SR value and PC are saved on supervisor stack
- **`RTE`** instruction restores state



MC68000 – Summary

- **6 priority levels** for hardware interrupts + NMI
  - Interrupt level 1–6, NMI level 7
  - Masking possible via status register $I_{0\text{-}2}$
- Only interrupts with **higher priority** and NMI can interrupt running interrupt handler
  - Status register is adapted automatically
- **Automatic state save** on supervisor stack, **nested handling** possible
- Vector number generation …
  - either autovectored: Index = Priority + 24
  - or non-autovectored (by external hardware): Index = 64 … 255
- No multiprocessor support

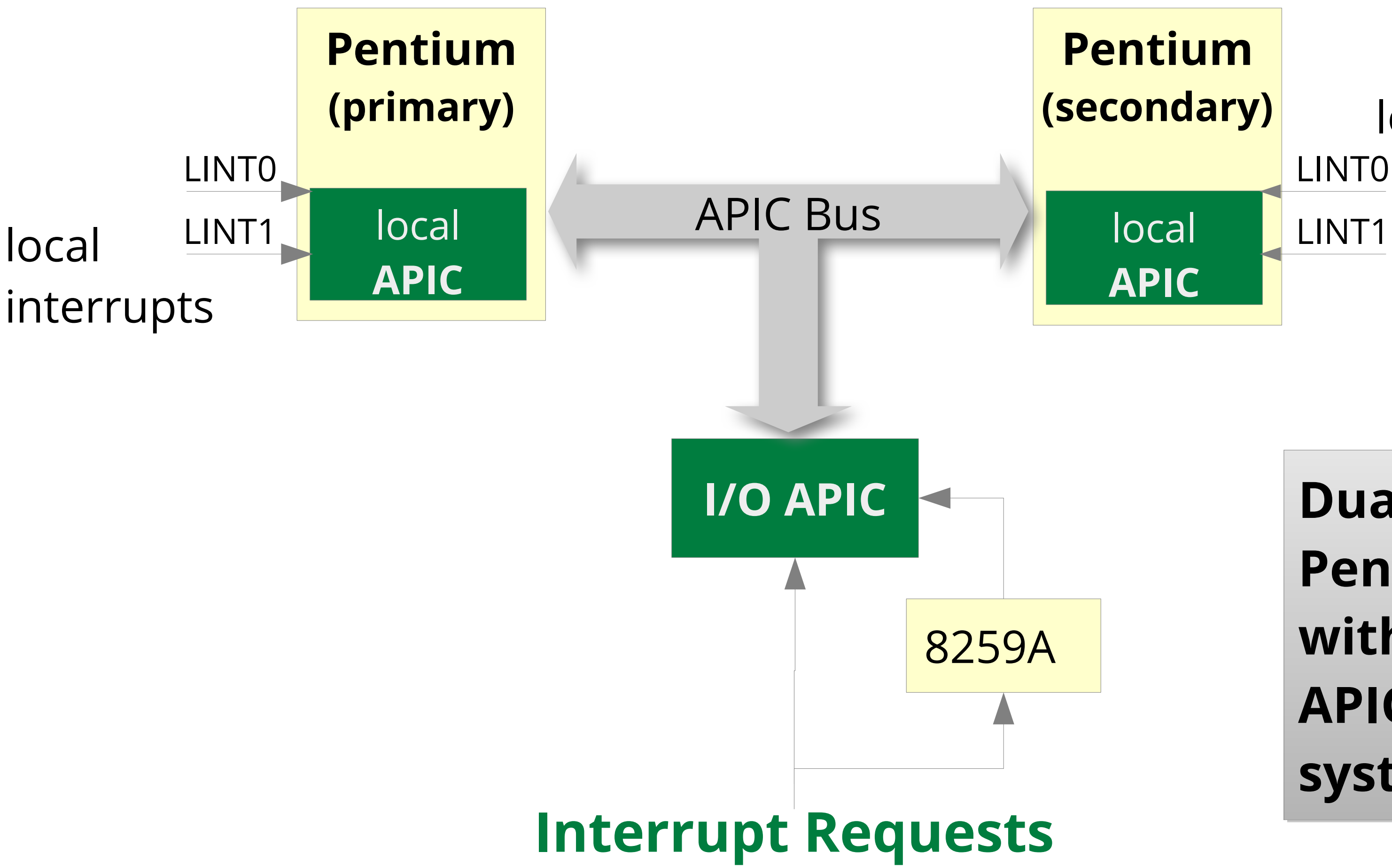
Interrupts in x86 CPUs

Interrupts in x86 CPUs

- Up and including i486, x86 CPUs had only 1 IRQ and 1 NMI line
- External hardware: prioritization, vector number generation
  - by a chip named **PIC 8259A**
    - 8 interrupt lines
    - 15 lines when cascading 2 PICs
    - no multiprocessor support
- Today's x86 processors contain the much more capable "Advanced Programmable Interrupt Controller" (**APIC**)
  - necessary for **multiprocessor systems**
  - completely superseded classic PIC 8259A
    - Compatibility: PIC interface still available in chipsets

APIC Architecture

- APIC interrupt system: **Local APIC** on each CPU, **I/O APIC**



Dual-processor Pentium system with distributed APIC interrupt system

I/O APIC

- Typically integrated in PC chipset's Southbridge
- Usually 24 interrupt lines
  - cyclic sensing (round-robin prioritization)
- **Interrupt Redirection Table:**
  64-bit entry for each interrupt line
  - Describes interrupt signal
  - Used for generating APIC bus message

I/O APIC

Structure (bits) of an *Interrupt Redirection Table* entry

63:56	**Destination Field** – R/W. 8-bit destination address depending on bit 11: APIC ID of a CPU (physical mode) or CPU group (logical mode)
55:17	*reserved*
16	**Interrupt Mask** – R/W. 1 = Do not forward this interrupt to a CPU.
15	**Trigger Mode** – R/W. 0 = Edge sensitive, 1 = Level sensitive
14	**Remote IRR** – RO. Type of received acknowledgment
13	**Interrupt Pin Polarity** – R/W. Signal polarity (high/low is active)
12	**Delivery Status** – RO. Interrupt message in flight?
11	**Destination Mode** – R/W. 0 = Physical mode, 1 = Logical mode
10:8	**Delivery Mode** – R/W. Affects destination APIC

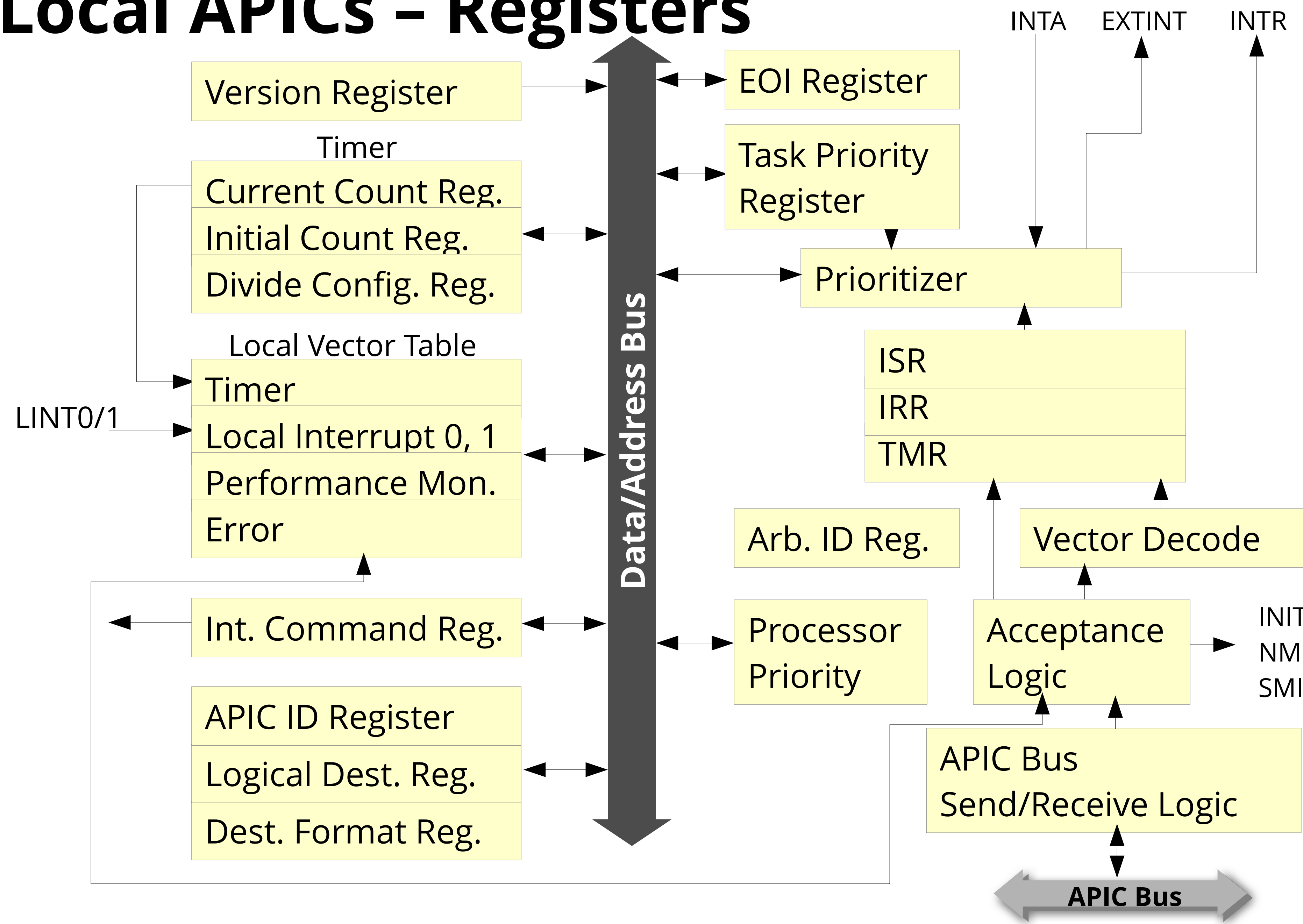
000 – Fixed	Deliver to all destination CPUs
001 – Lowest Priority	Deliver to CPU with currently lowest priority
010 – SMI	System Management Interrupt
100 – NMI	Non-Maskable Interrupt
101 – INIT	Initialize destination CPUs (reset)
111 – ExtINT	Answer to PIC 8259A

7:0 **Interrupt Vector** – R/W. 8-bit **Vector number between 16 and 254**

Local APICs

- Receive IRQs through APIC bus
- Also select/prioritize
- Can directly handle two local interrupts (lint0/lint1)
- Contain further functionality
  - Built-in timer, performance counters, thermal sensor
  - Command register:
    - Send own APIC messages
    - especially Inter-Processor Interrupt (IPI)
- Programmable via 32-bit registers (starting at 0xfee00000)
  - memory mapped (no external bus cycles)
  - Each CPU programs its own Local APIC

Local APICs – Registers



APIC Architecture – Summary

- Flexible IRQ distribution to CPUs in x86 MP system
  - fixed, groups, lowest task priority
  - multiple IRQs at once: prioritization with vector number
- Vector numbers 16–254 can be freely assigned
  - should be enough to avoid sharing
- Local APIC expects explicit EOI
  - Software must take care of this!
- With APIC, x86 in principle also supports priority levels
  - System software must act accordingly
    (re-enable interrupts, possibly use task priority register)

IRQ Sharing

- In practice, 24 IRQ lines proved to be insufficient
- … especially 4/8 lines for PCI devices:

PIRQ Line	#A	#B	#C	#D	#E	#F	#G	#H
AGP slot	shared							
PCI 1						shared		
PCI 2							used	
PCI 3					used			
PCI 4								shared
PCI 5						shared		
PCI 6			shared					
1. USB 1.1	shared							
2. USB 1.1				used				
3. USB 1.1			shared					
USB 2.0								shared
AC-97 Sound						shared		

- **Message-Signalled Interrupts (MSIs)** finally resolved this.

Summary

- Interrupt-handling hardware implements ...
  - Prioritization
  - Dispatch/execution of a handler routine
  - State save and nested execution
- Modern interrupt-handling hardware can ...
  - freely assign interrupt vectors,
  - avoid sharing vectors,
  - flexibly dispatch interrupts in multiprocessor systems.
- The operating system must ...
  - expect problems (spurious interrupts, interrupt storms)
  - pass on the signaled event to higher levels and finally to the application process.