

Betriebssysteme

Kapitel 1.1: Rechner-Architektur und Hardware-Grundlagen moderner Betriebssysteme

Stand: WS 10/11 (21.10.10)

Prof. Dr. Wolfgang Kuechlin

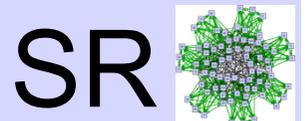
Dipl.-Inform., Dr. sc. techn. (ETH)

**Arbeitsbereich Symbolisches Rechnen
Wilhelm-Schickard-Institut für Informatik
Fakultät für Informations- und Kognitionswissenschaften**

Universität Tübingen

**Steinbeis Transferzentrum
Objekt- und Internet-Technologien (OIT)**

Wolfgang.Kuechlin@uni-tuebingen.de
<http://www-sr.informatik.uni-tuebingen.de>



Hardware Grundlagen

- Rechner-Architektur
 - single processor
 - (shared memory) multiprocessor
 - Hardware multi-threading (hyperthreading)
- Caches
- Interrupts und traps
- Timer
- DMA (direct memory access für externe Geräte)
- Prozessor Modes (kernel / user)
- Memory Management Unit MMU



Hardware Unterstützung für Effizienz

➤ Multi-Prozessor

- Parallele CPUs, heute Standard als Multicore
- Hyperthreading: Parallelität zw. Speicherzugriff und CPU

➤ Cache

- Zwischenspeicher, überbrücken Unterschiede in Zugriffszeiten
- Schneller first-level Cache auf Prozessor, second-level Cache zwischen Prozessor und Hauptspeicher/Bus

➤ Interrupt

- ermöglicht Parallelität zw. CPU und externem Gerät

➤ Direct Memory Access (DMA)

- ermöglicht effizienten direkten Transfer Hauptspeicher <--> Schnittstellen, parallel zur CPU



Hardware Unterstützung für Sicherheit

- Benutzerprozess soll Betrieb des Rechners nicht gefährden können.
- Prozessor-Modi
 - Ausschluss gewisser Prozessorinstruktionen (z.B. direkter Gerätezugriff) durch Benutzerprozess
 - Prozessor unterscheidet zwischen privilegiertem und unprivilegiertem Zustand
 - Zustandsübergang nur auf kontrollierte Art (System Call)
- Memory Management Unit (MMU)
 - ermöglicht Virtual Memory
 - rein privater Speicher, Größe unabhängig von realem Speicher
 - Effiziente Adressumrechnung virtuell → real



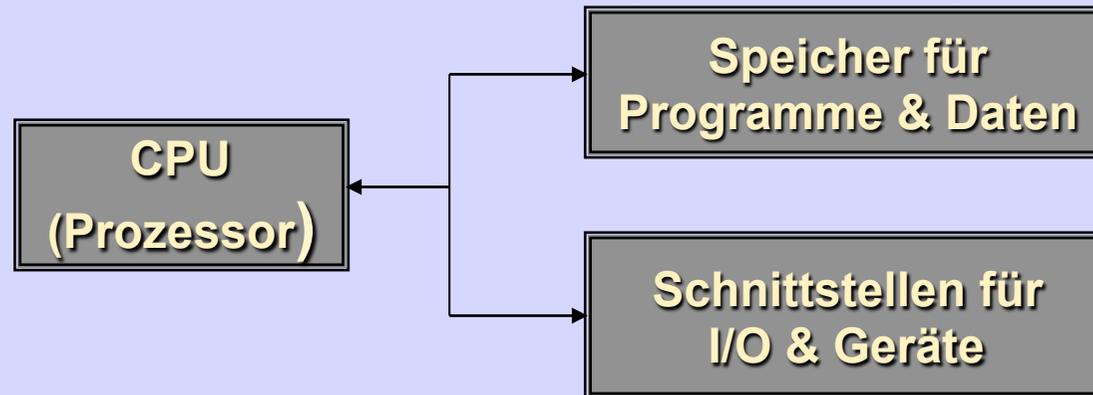
HW-Grundlagen moderner Systeme

- Von Neumann-Architektur
- Hardware-Unterstützung für Betriebssysteme dient
 - Effizienz
 - Sicherheit
- Effizienz
 - Parallelität
 - Caches
 - Interrupts
 - Timer
 - Direct Memory Access (DMA)
- Sicherheit
 - Processor-Modes (Kernel / User)
 - Traps
 - (Main) Memory Management Unit (MMU)



Rechner-Architektur

➤ Von Neumann Architektur (klassisch)



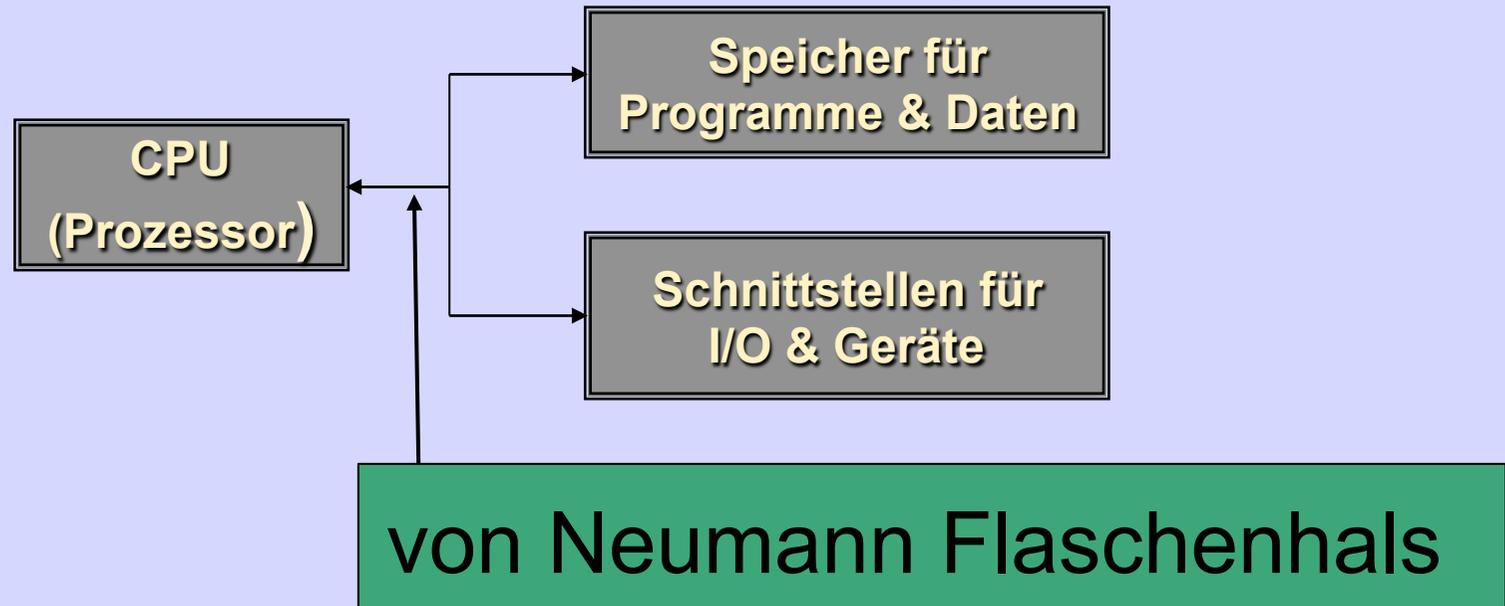
➤ SISD Single Instruction, Single Data

- Sequentielle Verarbeitung, Einprozessor-Rechner



Rechner-Architektur

➤ Von Neumann Architektur (klassisch)



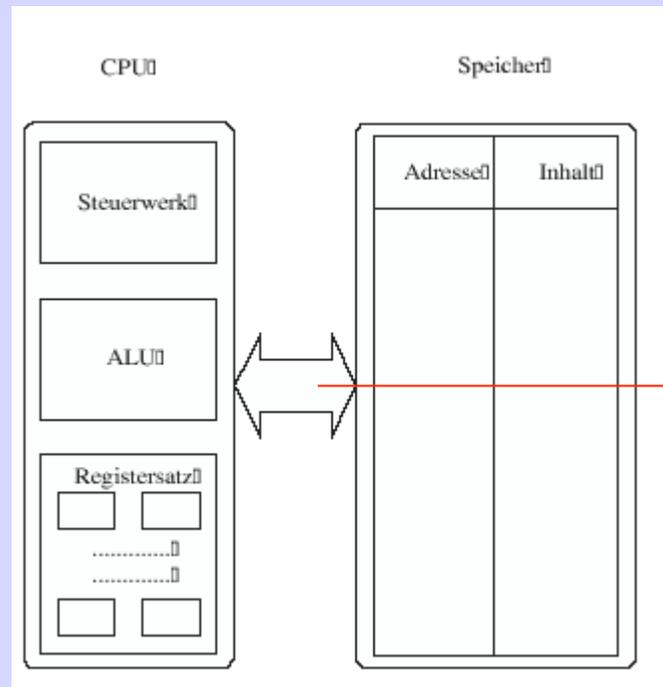
➤ SISD Single Instruction, Single Data

- Sequentielle Verarbeitung, Einprozessor-Rechner



von Neumann Architektur konkret

- Grundsätzlicher Aufbau verschiedener Rechnersysteme im Grundprinzip gleich

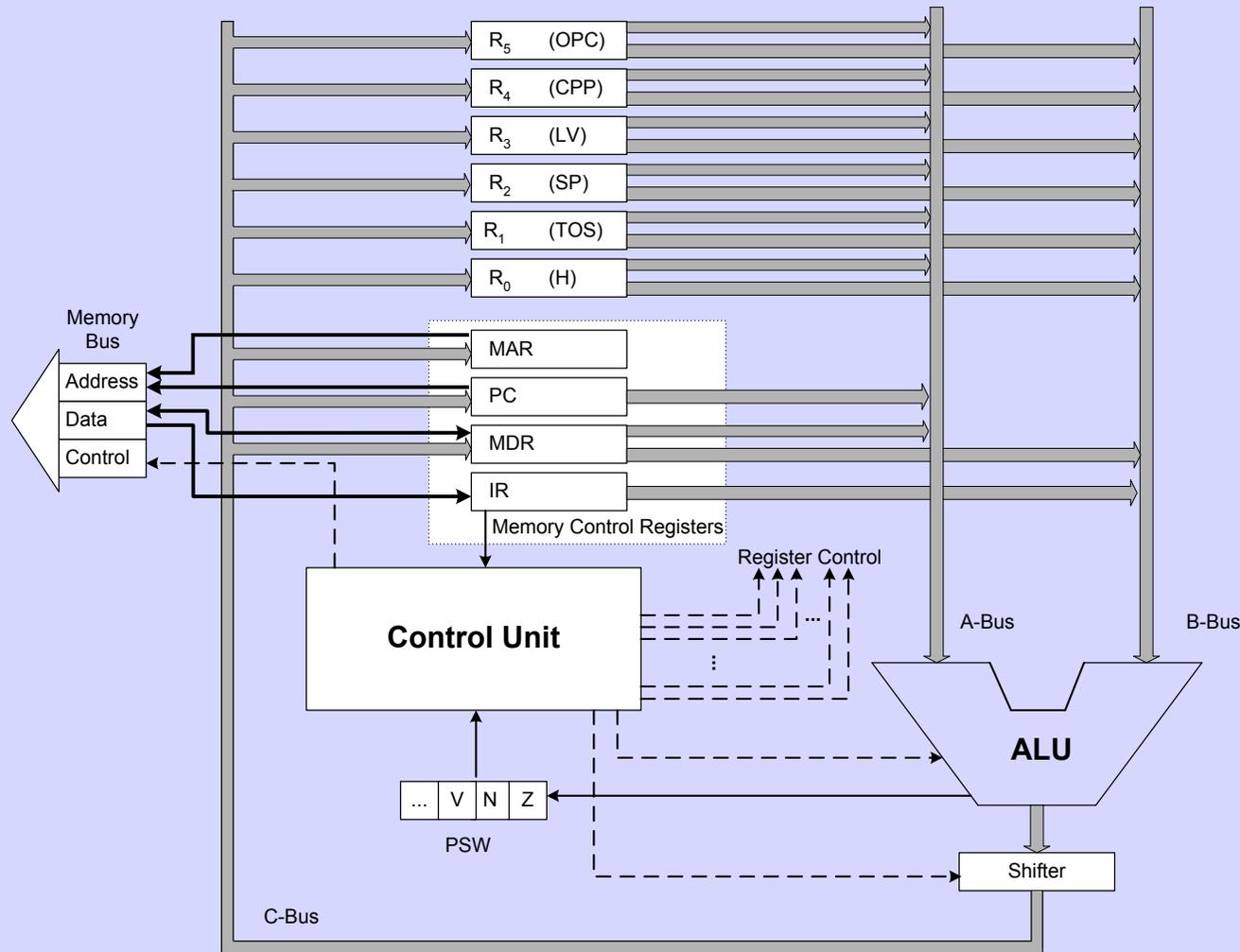


Bus mit Adress-, Daten-, und Steuerleitungen

Von Neumann Architektur



- Idealisierte Mikro-Architektur einer CPU mit externem Bus

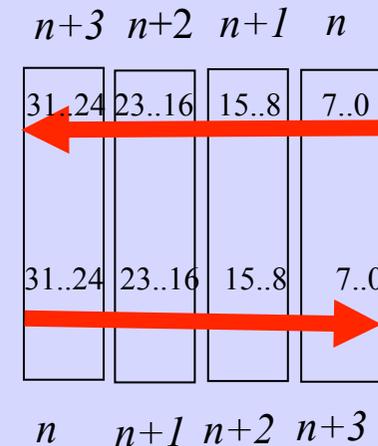


Big Endian, Little Endian

- Ein Integer i besteht aus 4 Bytes
 - i an Adresse n besteht aus Bytes $n+0, n+1, n+2, n+3$
 - Am Anfang von i ist MSB (Bits 31..24)
 - Am Ende von i ist LSB (Bits 7..0)
 - **Big Endian:** LSB hat Adresse $n+3$
 - **Little Endian:** LSB hat Adresse n
- Sowohl *Big Endian* als auch *Little Endian* werden benutzt
 - SUN SPARC und IBM Mainframes sind *Big Endian*
 - Die Intel Familie ist *Little Endian*
- Problem, wenn i byteweise zwischen *verschiedenen* Computern übermittelt wird

Little Endian

Big Endian



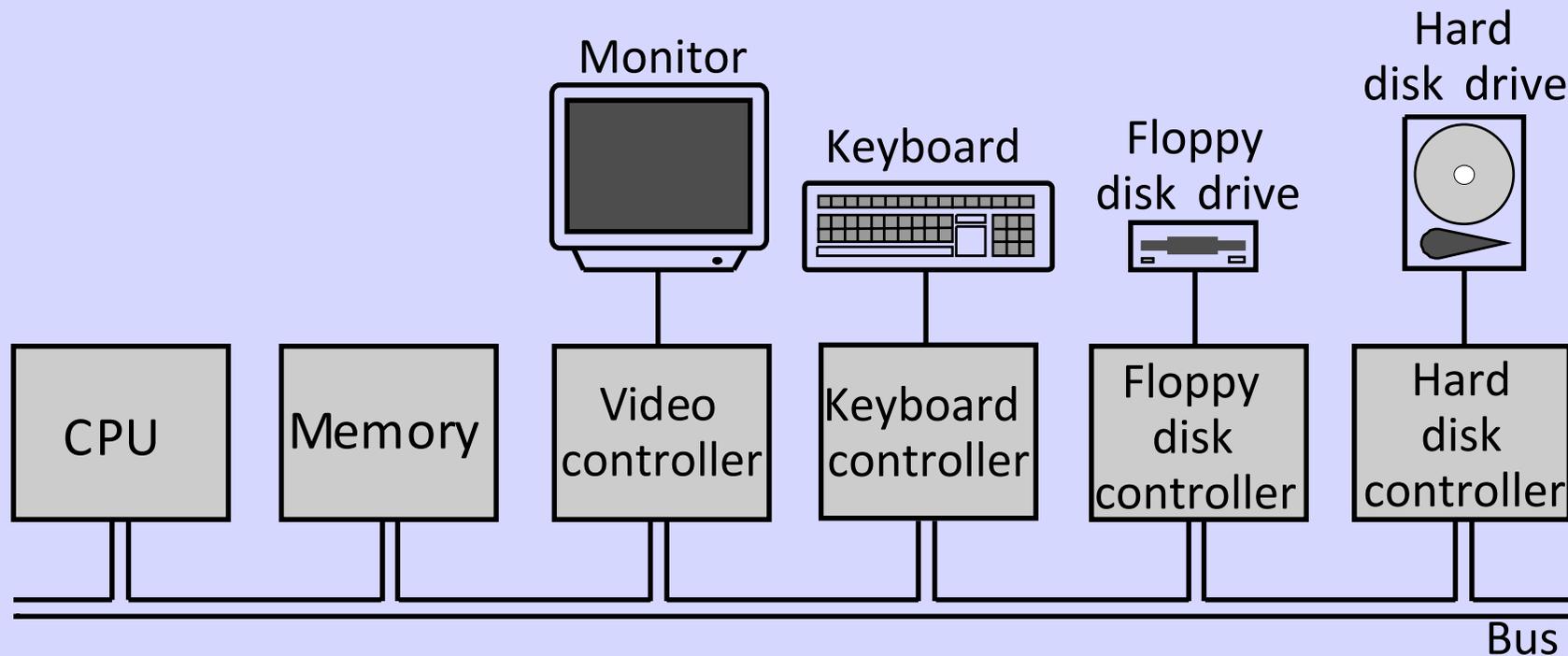
Beispiel: Repräsentation von 1025

Address	Big-Endian representation of 1025	Little-Endian representation of 1025
00	00000000	00000001
01	00000000	00000100
02	00000100	00000000
03	00000001	00000000



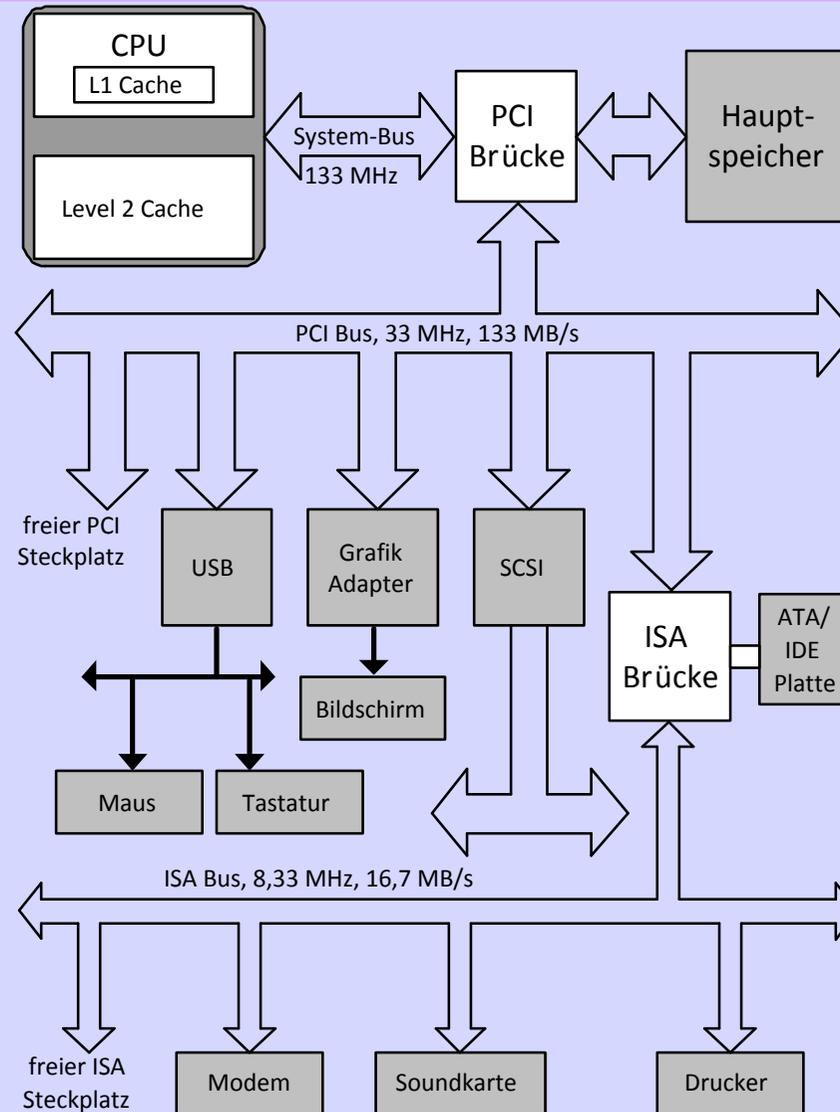
Organisation der Hardware

- Architektur eines einfachen Computersystems mit Bus
 - Controller bilden die digitale Schnittstelle der Geräte zum Bus
 - Sie empfangen Befehle als Bitmuster in Geräteregistern
 - Sie tauschen Daten als Bitmuster über Geräteregistern aus



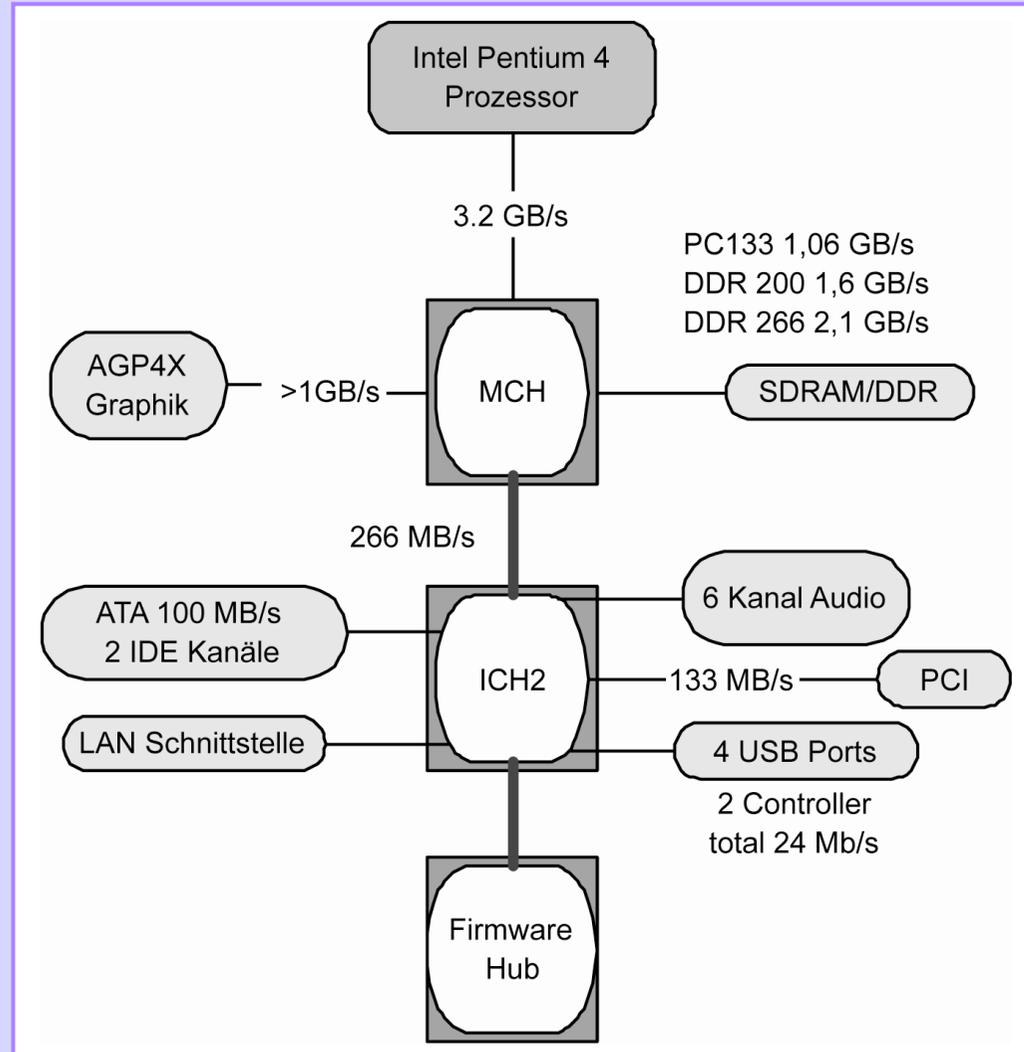
System-Architektur der Hardware

Architektur eines PC Systems
mit mehreren Bussen an Brücken



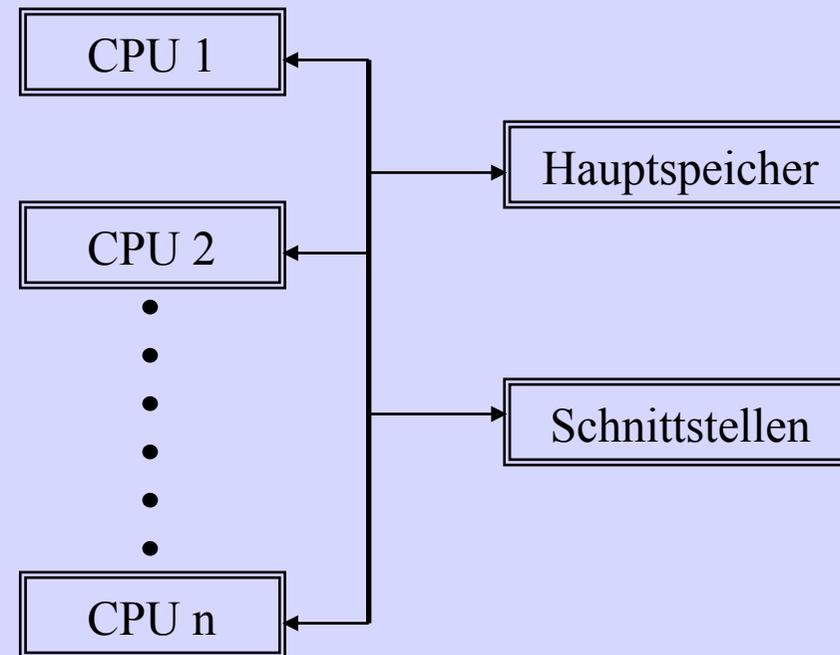
System-Architektur der Hardware

- Architektur eines modernen PC Systems mit mehreren Bussen an Hubs



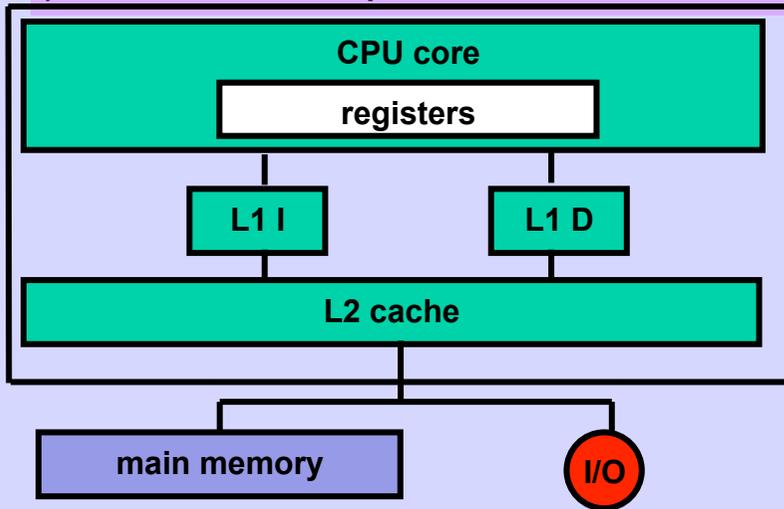
Rechner-Architektur

- Zukünftig: Shared memory Parallelrechner
 - Mehrere CPUs an Bus oder auf gleichem Chip
- MIMD: Multiple Instructions, Multiple Data
 - **Asynchron parallele Verarbeitung, Mehrprozessor-Rechner (oder verteiltes System)**

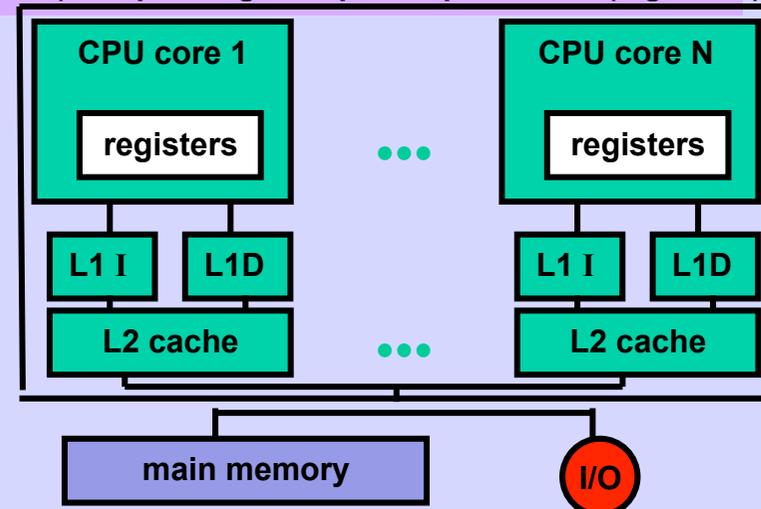


Multiprozessor-Architekturen

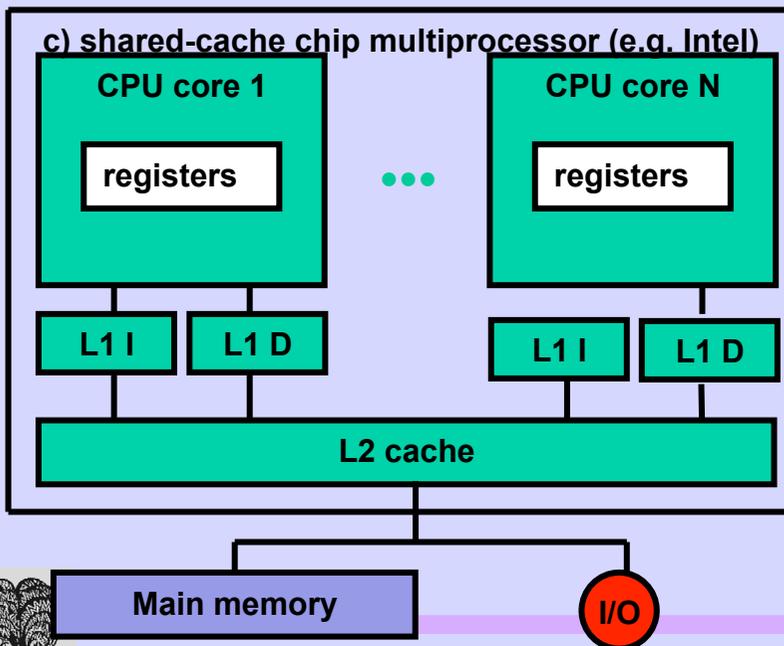
a) conventional microprocessor



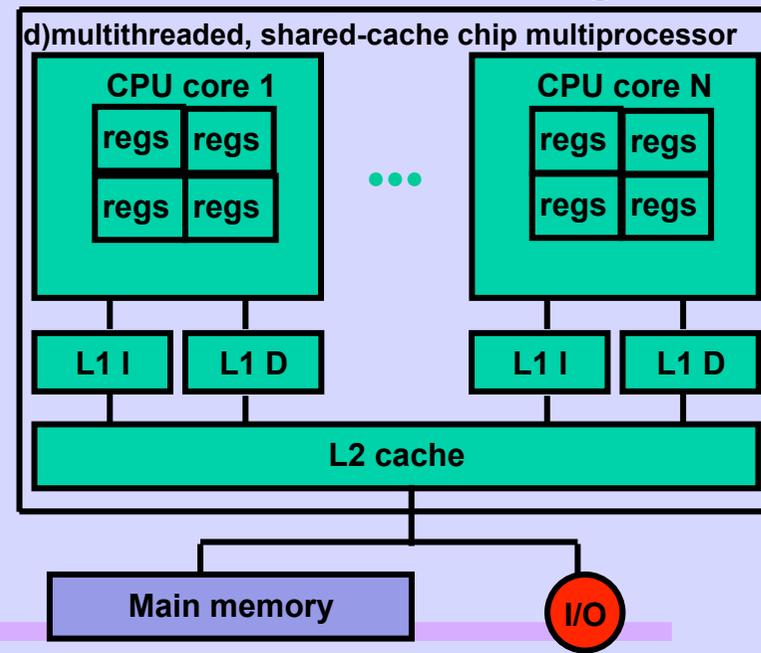
b) Simple single chip multiprocessor (e.g. AMD)



c) shared-cache chip multiprocessor (e.g. Intel)

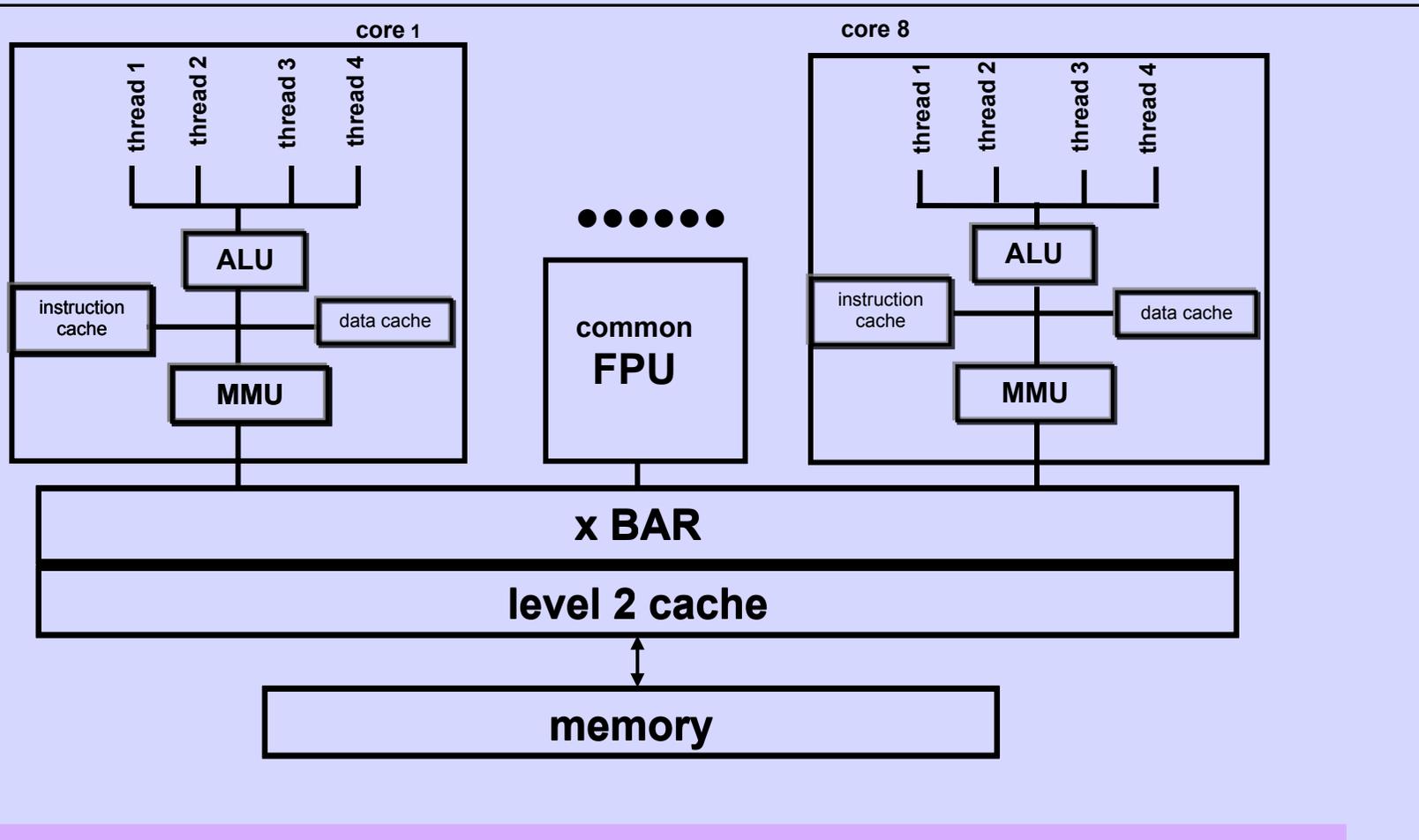


d) multithreaded, shared-cache chip multiprocessor



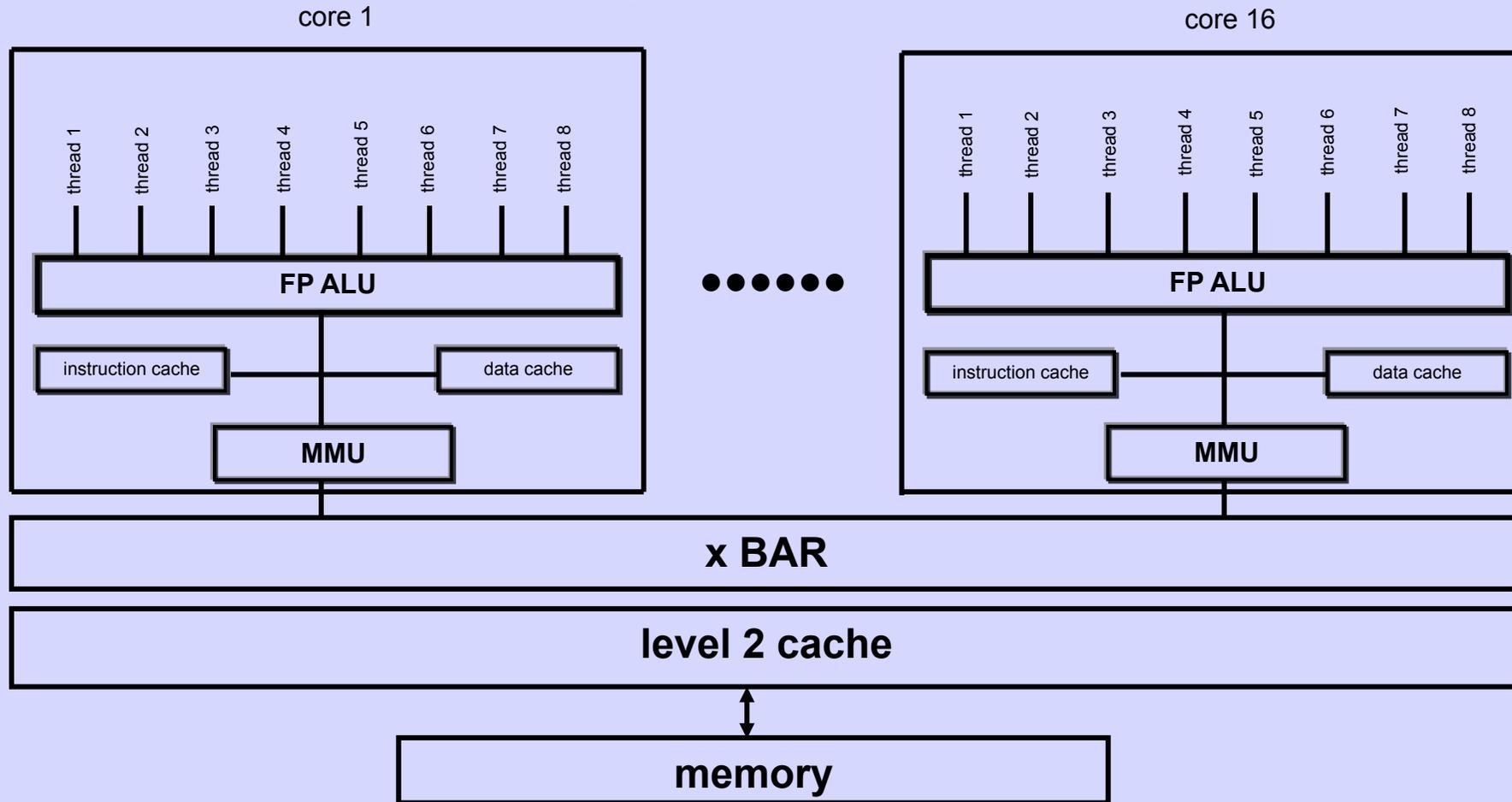
SUN Niagara 1 Multicore (T1) (single chip)

8 Way Multicore x 4 Way Multi-Threaded

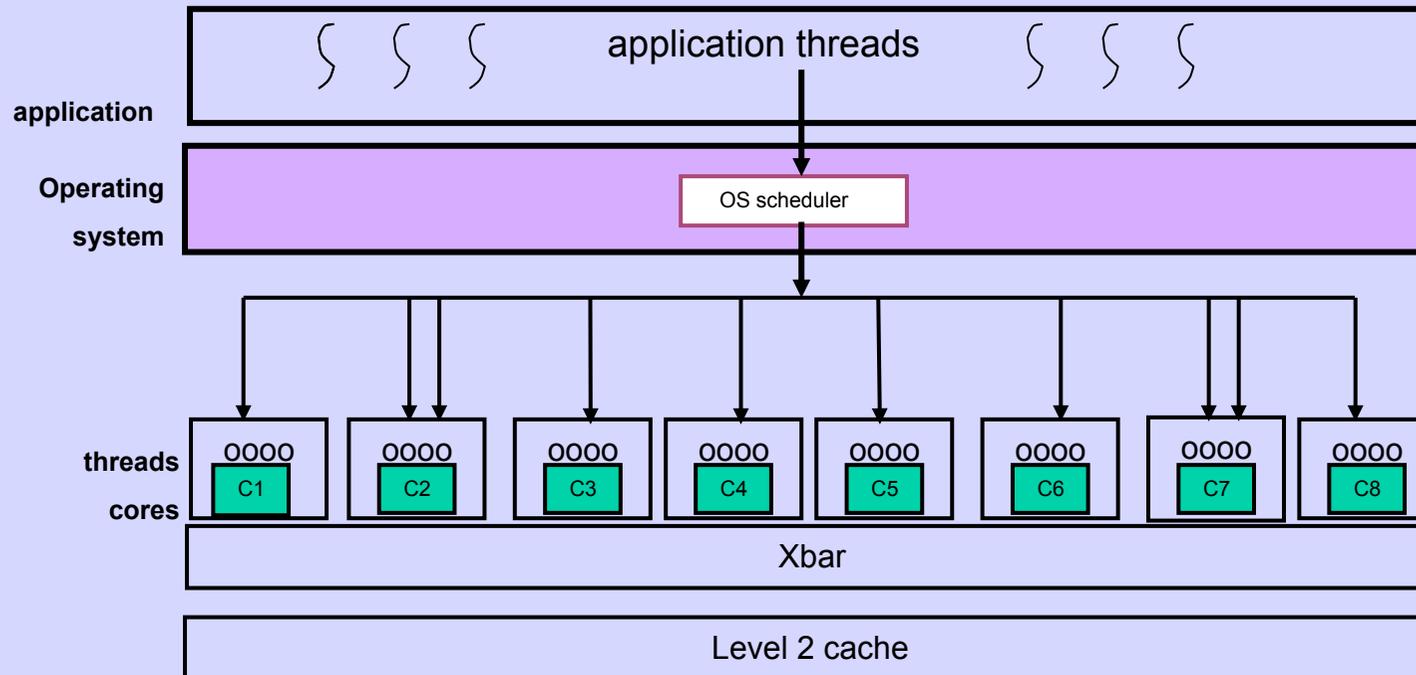


SUN Niagara 2 Multicore (T2) (single chip)

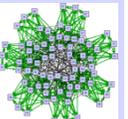
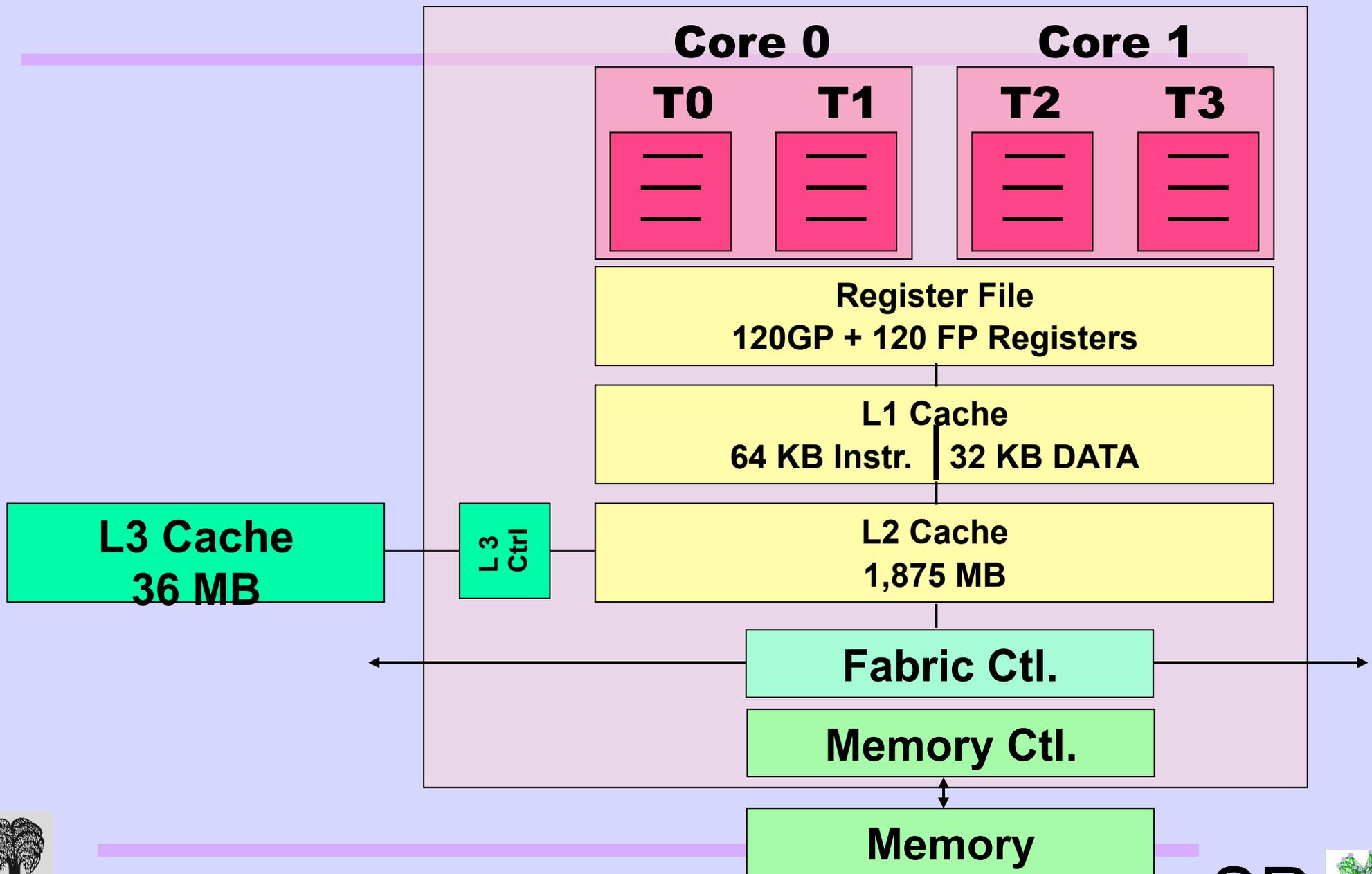
16 Way Multicore x 8 Way Multi-Threaded



Software Threads Scheduling on CMP Cores/ Threads



IBM Power 5 Processor

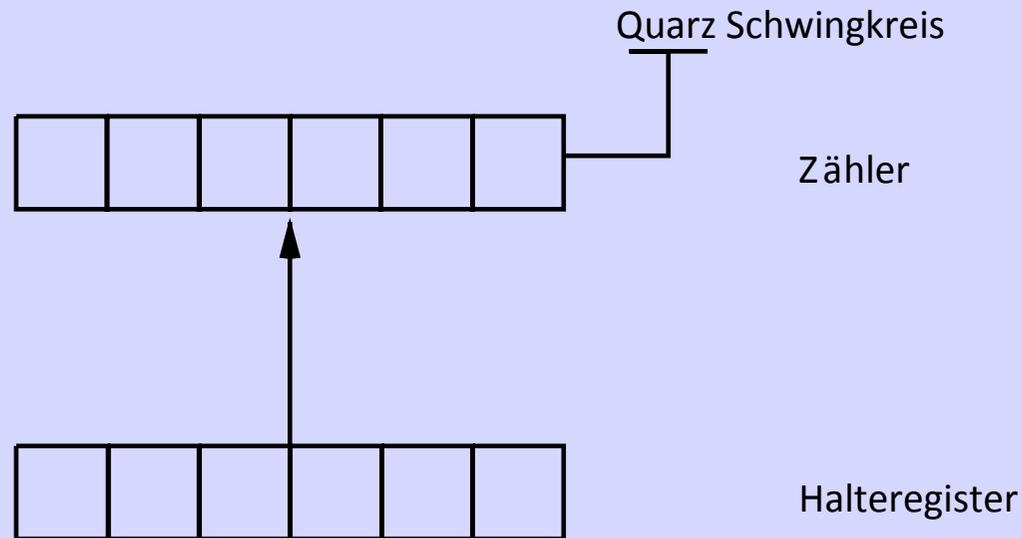


Unterbrechung

- **Interrupt** (*interruption*)
 - **Asynchrone Serviceanforderung ans Betriebssystem**
 - Ausgelöst durch Ereignisse (*events*)
(Quellen: Hardware und Software)
 - z.B.: Timer löst periodisch Interrupt aus (z.B. alle 10 ms)
 - » benutzt für Scheduling, Timesharing
 - **Ablauf (Grundprinzip)**
 - Prozessor wird durch Hardware-Signal bei Arbeit unterbrochen
 - Prozessor führt Interrupt Service Routine aus
 - Prozessor fährt mit Arbeit fort
 - **Zweck**
 - Kommunikation mit einem parallel arbeitenden Gerät
 - **Alternative**
 - periodische Abfrage (*polling*) des Geräts durch Prozessor



Timer



- Prozessor lädt Wert in Halteregister
- Zähler dekrementiert bis Null, parallel zur CPU
- Zähler == 0 löst (timer-)interrupt aus



Definitionen: Traps / Interrupts

➤ **Interrupt** = asynchrone Service-Anforderung

▪ **Hardware-Interrupt**

- durch CPU-externe Geräte oder andere CPU (Multiprozessor)

▪ **Software-Interrupt** (*software initiated interrupt*)

- Durch BS-Software veranlasster Interrupt
- Bei hoher Priorität: sofort (=synchron) ausgeführt → äquivalent zu SW-trap
- Bei niedrigerer Priorität: später (=asynchron) ausgeführt
(z.B. späteres asynchrones Erledigen sekundärer Arbeiten)

➤ **Trap** = synchrones Eintauchen ins Betriebssystem

ausgelöst durch laufendes Programm (benutzt evtl. interrupt Hardware)

▪ **Hardware-Trap** (*Exception*)

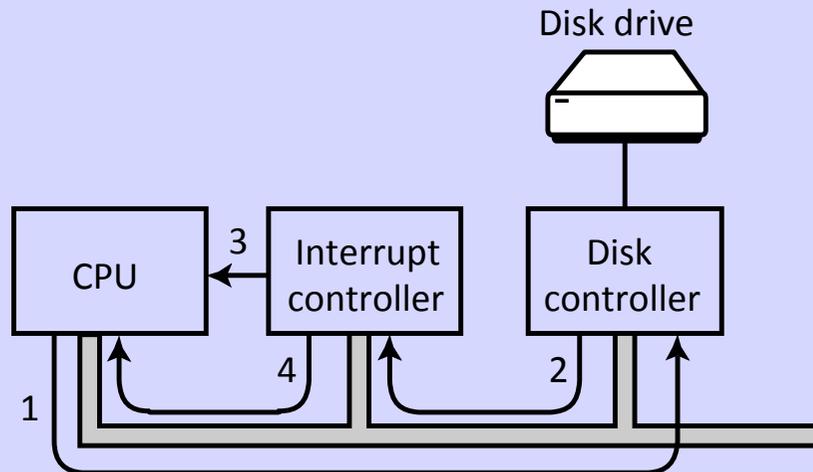
- synchron ausgelöst durch Hardware (Bsp.: Division durch 0)

▪ **Software-Traps**

- synchron ausgelöst durch Software (System Call)

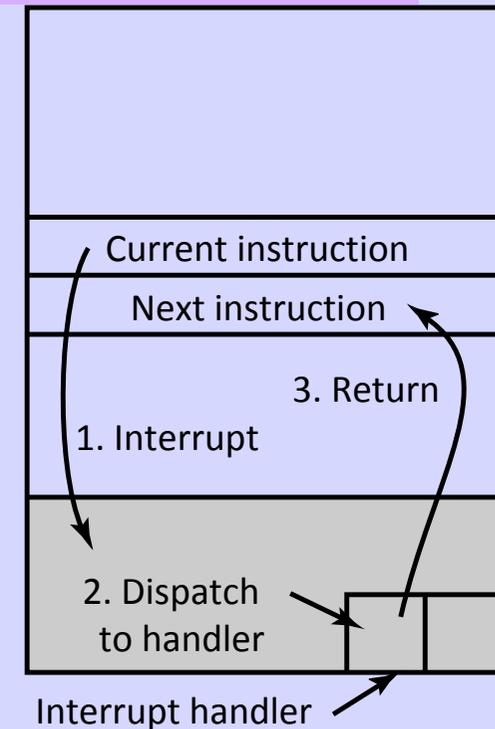


Klassische PC Interrupt-Architektur



(a)

1. Auftrag an disk (z.B. read)
2. Interrupt-request IRQ über interrupt-Leitung des I/O-Busses an PIC (programmable interrupt controller) auf der South-Bridge
3. Unterbrechung der CPU über ihren interrupt-Signaleingang
4. Index (IRQ+basis) des benötigten Handlers über Datenbus an CPU



(b)

How the CPU is interrupted

Quelle: Tanenbaum



Traditionelle IRQ-Belegung auf PC

➤ PIC 1 (Master)

➤ +-----+

➤ | IRQ 0 + <--- Timer

➤ | IRQ 1 + <--- Tastatur

➤ | IRQ 2 +-<-----+

➤ | IRQ 3 + <--- Seriell |

➤ | IRQ 4 + <--- Seriell |

➤ | IRQ 5 + <--- Soundkarte |

➤ | IRQ 6 + <--- Floppy |

➤ | IRQ 7 + <--- Parallel |

+-----+-----+ Port

|
|
+--->>> Zur CPU

PIC 2 (Slave)

+-----+

| IRQ 8 + <--- Echtzeituhr

| IRQ 9 + <--- ...

| IRQ 10 + <--- ...

| IRQ 11 + <--- ...

| IRQ 12 + <--- PS/2 Maus

| IRQ 13 + <--- Koprozessor

| IRQ 14 + <--- Festplatte

| IRQ 15 + <--- Festplatte

+-----+-----+

| |
+-----+



Advanced PC Interrupt Architecture

- Zweistufige Interrupt-Architektur für PC Systeme mit Advanced PIC
 - Local APIC auf jedem CPU-Kern
 - I/O-APIC im Chipsatz (an den I/O-Bussen)
- L-APICS können sich gegenseitig interrupt signalisieren
 - essentiell für Multi-CPU Systeme
- Nachrichten zwischen APICs werden über System-Bus ausgetauscht
 - Interrupts als Interrupt messages versendet
 - L-APIC leitet höchst-priore Interrupt-Nachricht an seine CPU weiter

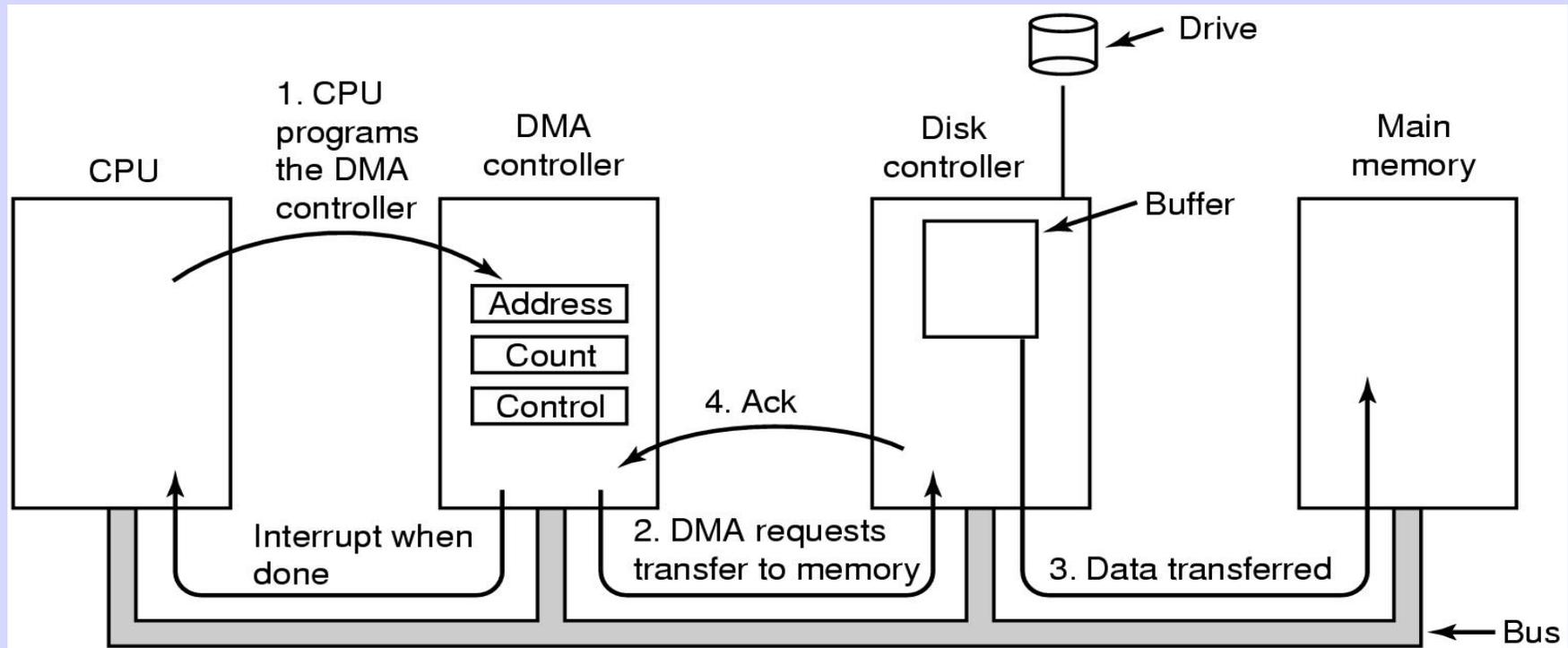


Behandlung eines Interrupts

- Prozessor unterbricht Arbeit und speichert (*save*) automatisch (Teil des) Registersatz ab (z.B. auf kernel stack, interrupt stack)
- Maskierung von Interrupts tieferer Priorität (heute im interrupt controller)
- Verzweigung zum Interrupthandler (im BS-Kern)
 - Interrupt-Vektor mit Startadressen der Handler
 - Handler-Index berechnet sich aus Interrupt-Nummer
 - Bedienen des Interrupt (interrupt service routine)
 - Sichern aller Register im Leitblock des unterbrochenen Prozesses
 - Aufsetzen eines Handler-Kontexts (Stack, MMU)
 - Quittierung des (Empfangs des) Interrupts an controller
 - Ggf. Suchen des auslösenden Geräts, Deblockieren des Geräts, Kommunikation
- Demaskierung von Interrupts tieferer Priorität (heute im interrupt controller)
- Zurückladen (*restore*) des Registersatzes
 - evtl. zunächst Prozesswechsel durch Betriebssystem (hoch-priorer Prozess könnte nach Interrupt jetzt lauffähig sein)



Direct Memory Access (DMA)



Quelle: Tanenbaum

Operation of a DMA transfer



Direct Memory Access (DMA)

➤ Funktion

- Datenübertragung von Gerät/Memory zu Gerät/Mem ohne CPU
- Traditionell: Gerät → DMA controller → Gerät
- fly-by mode: Gerät → Gerät

➤ Arbeitsweise

- CPU programmiert DMA controller (Adressen, Länge Daten)
- DMA steuert Geräte für Datenübertragung
- DMA meldet Vollzug über Interrupt an CPU

➤ Cycle-stealing

- DMA controller belegt Bus und „stiehlt“ Buszyklen von CPU



Prozessorzustände

➤ User-Mode

- Eingeschränkte Verwendung von Ressourcen
 - insbes. kein direkter Zugriff auf absolute Adressen oder auf Geräte
- Nur Instruktionen, welche keinen anderen Prozess beeinflussen, MMU erlaubt nur Zugriff auf bestimmte Speicherbereiche.
- Nach Trap (trap Instruktion in Assembler) in den Kernel-Mode wird Betriebssystem ausgeführt (system call)

➤ Kernel-Mode

- Betriebssystem prüft system call auf Korrektheit
- Volle Verwendung der Ressourcen



Prozessorzustände

- Feinere Aufteilung für geschichtetes BS
 - Mehrere Kernel-Modes (ab Intel 386: 4 Modi)
 - Für jeden Mode separaten Stack.
- Intel 8088: Keine Modi
 - MS-DOS kein sicheres BS
- Intel x86 ab 80286 bietet zwei Betriebsarten (diese Modes entsprechen nicht den oben erwähnten).
 - Protected Mode
 - In dieser Betriebsart (mind.) zwei Modi: Kernelmode und Usermode
 - Real Mode
 - In dieser Betriebsart ist der Prozessor kompatibel zu früheren Prozessormodellen, keine Prozessormodi.

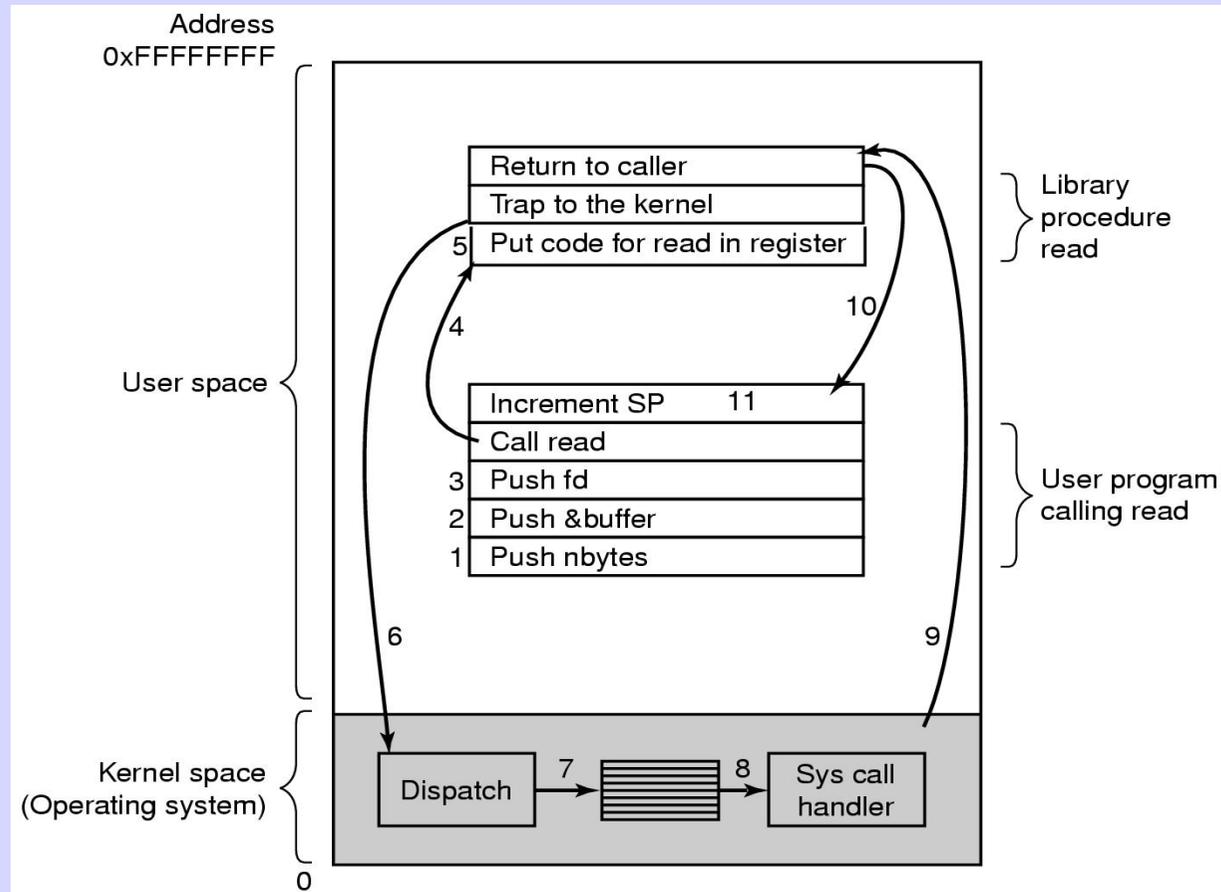


Prozessorzustände

- MS-DOS, Windows
 - Prozessor läuft im Real Mode
- Linux, OS/2, Windows NT
 - Prozessor läuft im Protected Mode.
- Prozessorfamilien
 - Motorola 68000, Power PC, SPARC und andere RISC
 - haben immer mindestens Kernel- und Usermode



Steps in Making a System Call



There are 11 steps in making the system call read (fd, buffer, nbytes)

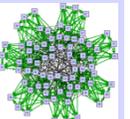
Quelle: Tanenbaum



Some System Calls For Process Management

Process management

Call	Description
<code>pid = fork()</code>	Create a child process identical to the parent
<code>pid = waitpid(pid, &statloc, options)</code>	Wait for a child to terminate
<code>s = execve(name, argv, environp)</code>	Replace a process' core image
<code>exit(status)</code>	Terminate process execution and return status



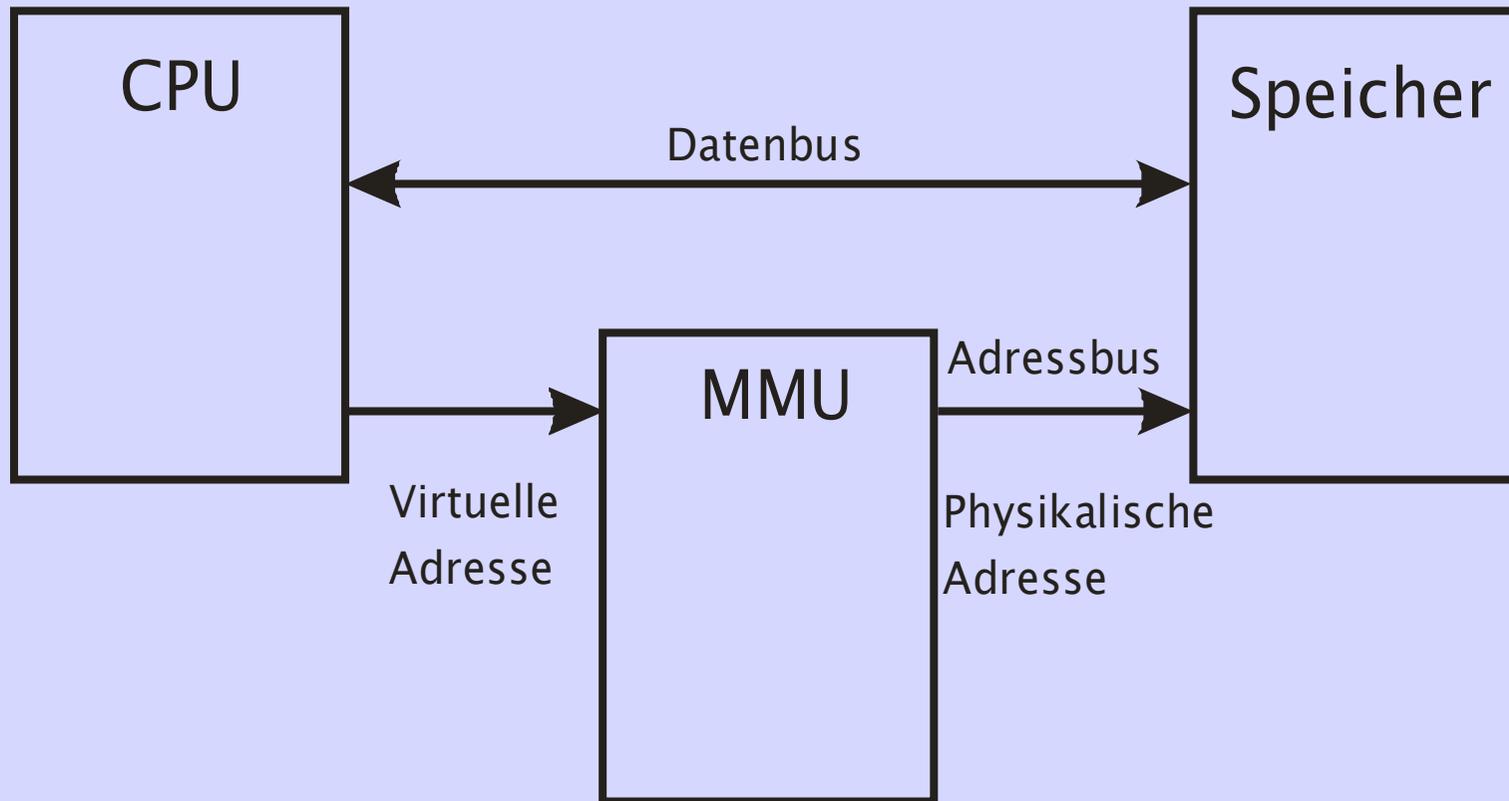
Some System Calls For File Management

File management

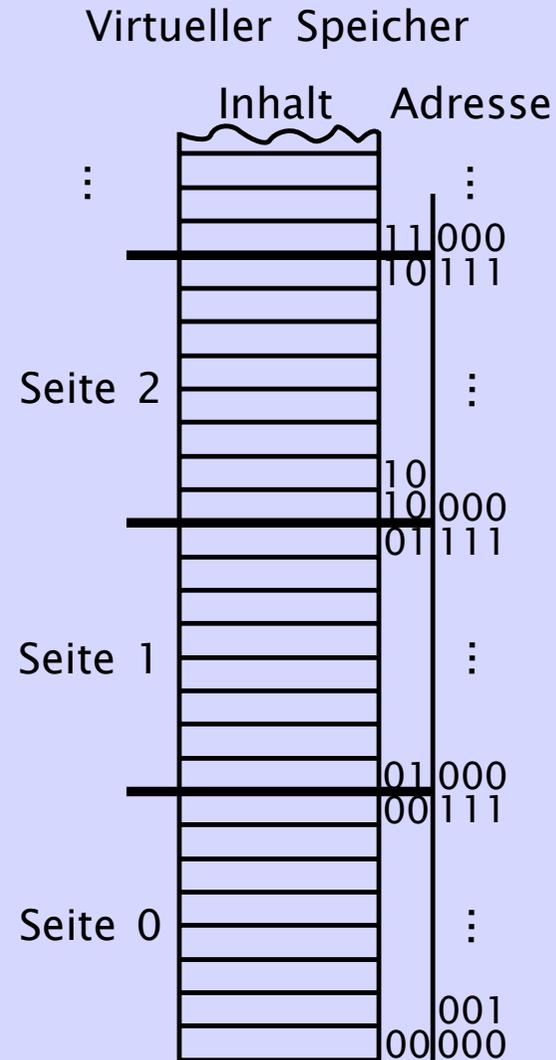
Call	Description
<code>fd = open(file, how, ...)</code>	Open a file for reading, writing or both
<code>s = close(fd)</code>	Close an open file
<code>n = read(fd, buffer, nbytes)</code>	Read data from a file into a buffer
<code>n = write(fd, buffer, nbytes)</code>	Write data from a buffer into a file
<code>position = lseek(fd, offset, whence)</code>	Move the file pointer
<code>s = stat(name, &buf)</code>	Get a file's status information



memory Management Unit (MMU)



Virtueller Speicher



Prinzip der Adressumsetzung

Bsp: $k=2$, $n=3$

