EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Enterprise Computing
Verarbeitungsgrundlagen

Prof. Dr.-Ing. Wilhelm G. Spruth
Dipl. Inf. Gerald Kreißig

WS2016/17

Unterschiedliche Begriffe

Die Mainframe Welt benutzt häufig andere Begriffe als die PC oder Unix/Linux Welt.
Hier ist ein kleiner Auszug aus meinem Wörterbuch:

z/OS, OS/390	Windows/Unix
Problem State	**User Mode**
Supervisor State	**Kernel Mode**
Region	**Virtueller Adressenraum**
Task	**Process**
Subtask	**Tread**
Data Set	**File**
DASD Direct Access Storage Device	**HDD, Disk, Plattenspeicher**
Program Status Word	**Status Register**
Firmware	**Microcode**

Verarbeitungsgrundlagen Teil 1

Mainframe Architektur

Was ist eine Rechnerarchitektur?

Als Rechnerarchitektur bezeichnen wir das Erscheinungsbild eines Rechners, wie ihn der Assembler (oder Maschinensprachen-) Programmierer sieht.

CISC – Complex Instruction Set Computer

- **x86** PCs und Laptops, größtenteils von Intel und AMD hergestellt
- **Itanium** wird in großen Hewlett Packard Unix Rechnern eingesetzt, hergestellt von Intel
- **System z** ausschließlich in Mainframes eingesetzt, heute fast nur noch von IBM hergestellt

RISC – Reduced Instruction Set Computer

- **PowerPC** IBM Unix "System P" Rechner, Sony Playstation 3, CISCO Internet Router, Microsoft X-Box, Nintendo, embedded Systems in der Automobilindustrie, hergestellt von Freescale und IBM
- **Sparc** wird in großen Sun/Oracle und Fujitsu Rechnern eingesetzt, hergestellt von Oracle (früher Sun)
- **ARM** Mobiltelefone, viele embedded Systems, wachsende Bedeutung
  entwickelt von Acron – Lizenzen an > 40 Chiphersteller

Die RISC Architektur wurde Ende der 1970-er Jahre von 2 Gruppen von Wissenschaftler entwickelt:

- **David Patterson** von der UC Berkely
- **John Cocke** von IBM Research in Yorktown Heights, NY

Historischer Hintergrund

Am 7. April 1964 kündigte IBM die S/360-Rechnerfamilie (System 360) an. Die Ziffer „360“ bezog sich dabei auf alle Richtungen eines Kompasses, um die universelle Anwendbarkeit, den weitgefächerten Bereich von Performance und Preis der Produkte sowie die Entwicklungsrichtungen des Unternehmens zu demonstrieren. Die Familie bestand ursprünglich aus 6 Zentraleinheiten und etwa 45 weiteren Ein-/Ausgabeeinheiten.

Die Existenz der S/360-Architektur ist den drei Wissenschaftlern *)

Gene Amdahl,
Gerry Blaauw,
Fred Brooks

und der damaligen IBM-Entwicklungs-Organisation unter der Leitung von IBM-Vizepräsident B.O. Evans zu verdanken, siehe http://www.informatik.uni-leipzig.de/cs/Literature/History/index.html .

Die Einführung der S/360-Architektur stellt einen Meilenstein in der Entwicklung des Computers dar. Zum damaligen Zeitpunkt waren die Unterschiede in den Rechner-Architekturen der einzelnen Hersteller sehr viel größer als dies heute der Fall ist. Viele Eigenschaften, die heute als selbstverständlich gelten, entstanden mit der Einführung der S/360-Architektur. Einige von vielen Beispielen sind:

- Die Entscheidung, dass ein Byte eine Länge von 8 Bit hat (und nicht z.B. 6 oder 7 Bit),
- die Tatsache, dass die Einheit der Hauptspeicheradressierung das Byte ist (und nicht ein 24-, 36- oder 48-Bit langes Wort),
- die Einführung von Mehrzweckregistern, die gleichzeitig als Adressregister und als Datenregister dienen,
- der Verzicht auf die direkte Hauptspeicher-Adressierung,
- der Unterschied zwischen Kernel (Supervisor)- und User (Problem)-Status,
- die Einführung des S/360-Kanals, der noch heute in der Form der SCSI und Ficon-Interfaces weiterlebt.

*) Amdahl, G.M., G.A. Blaauw, and F.P. Brooks, Jr. "Architecture of the IBM System/360." IBM Journal of Research and Development, vol. 8, no. 2 (April 1964): 87-101.

An Architecture to stand the Test of Time

Ein weiterer Meilenstein war die bewusste Entscheidung, die S/360-Architektur für eine sehr lange Lebenszeit auszulegen. Diesem dienten vor allem zwei Maßnahmen:

- Die Verpflichtung und Garantie, dass Maschinencode auf allen damaligen sowie zukünftigen Rechnermodellen unverändert lauffähig sein würde. Diese Strategie wurde bis zu den heutigen System z Modellen eingehalten.

- Die Einrichtung eines Architektur-Boards mit der Aufgabenstellung, die S/360-Architektur nach wissenschaftlichen Grundsätzen weiterzuentwickeln.

Diese und viele andere Entscheidungen von Amdahl, Blaauw und Brooks erwiesen sich als außerordentlich tragfähig und haben dazu geführt, dass sich die System z Architektur auch heute noch fortschrittlich und zukunftsorientiert darstellt.

Es existieren nur wenige Architektureigenschaften, die man mit dem heutigen Wissensstand anders und/oder besser machen würde. Dies ist besonders bemerkenswert, weil viele später entwickelte Rechnerarchitekturen mit neuartigen Entwicklungen versprochene Verbesserungen nicht liefern konnten, und in der Zwischenzeit wieder von der Bildfläche verschwunden sind. Ein Beispiel hierfür ist die Entwicklung der VAX-Architektur durch die Firma Digital Equipment Corporation (DEC).

DEC war in den 80er Jahren der weltweit zweitgrößte Computer Hersteller nach IBM. Im Jahre 1976 entwickelte DEC eine komplett neue Rechner-Architektur, die erheblich besser als die S/370 Architektur sein sollte. Nach größeren Anfangserfolgen musste DEC die VAX Architektur aufgeben, weil sie gegenüber S/370 schlicht nicht wettbewerbsfähig war. Die Firma Digital Equipment Corporation hat sich von diesem Desaster nie wieder erholen können.

1991 löste die Firma DEC ihre VAX-Architektur durch die Alpha-Architektur ab.

In dem Vorwort des Alpha-Architektur-Handbuches*) wurde explizit darauf hingewiesen, dass man die gleichen Entwurfsprinzipien angewendet habe, die von Amdahl, Blaauw und Brooks 1964 für die Einführung von S/360 entwickelt wurden:

The Alpha architecture is a RISC architecture that was designed for high performance and longevity. Following Amdahl, Blaauw, and Brooks, we distinguish between architecture and implementation:

- **Computer architecture is defined as the attributes of a computer seen by a machine language programmer. This definition includes the instruction set, instruction formats, operation codes, addressing modes and all registers and memory locations that may be directly manipulated by a machine language programmer.**

- **Implementation is defined as the actual hardware structure, logic design and datapath organization.**

This architecture book describes the required behavior of all Alpha implementations, as seen by the machine-language programmer.

*) Alpha Architecture Reference Manual, Digital Press, Digital Equipment Corporation, 1992

Lebensdauer von Anwendungssoftware

Ein wichtiger Unterschied zwischen Enterprise Computing und anderen Einsatzgebieten der Informatik ist die erwartete Lebensdauer der damit entwickelten Anwendungen. Bei manchen Smartphones und Tablets wird akzeptiert, dass Software für ältere Modelle aufgrund deren begrenzten Langzeitsupports nicht mehr verfügbar ist. Der ideale Kunde eines Smartphone Herstellers ersetzt sein Mobiltelefon alle 24 Monate durch ein neues, verbessertes Modell, obwohl das alte Modell noch voll funktionsfähig ist. Inkompatibilitäten zwischen aufeinander folgenden Softwareversionen sind in dieser Situation wegen des schnelllebigen Marktes kein gravierendes Problem.

Im Gegensatz dazu hat Enterprise Software eine Lebensdauer, die in Jahrzehnten gemessen wird. Ein Testfall waren die „Jahr 2000“ (y2k) Umstellungen von zweistelligen auf vierstellige Jahreszahlen. Obwohl der erforderliche Umstellungsaufwand von weltweit 300 Mrd. $ ein guter Anlass gewesen wäre, veraltete Software durch neuere, moderne Versionen zu ersetzen, ist dies fast nirgendwo geschehen.

In der Vergangenheit verstand man unter dem Begriff „Legacy Software“ viele Jahre alte Anwendungen, die auf Mainframe Rechnern ausgeführt wurden. In zunehmendem Maße existieren in den Unternehmen jedoch Legacy Anwendungen, die auf Windows (in unterschiedlichen Versionen), Linux, AIX, HP_UX, Solaris, Tru64 UNIX und vielen weiteren Betriebssystemen laufen.

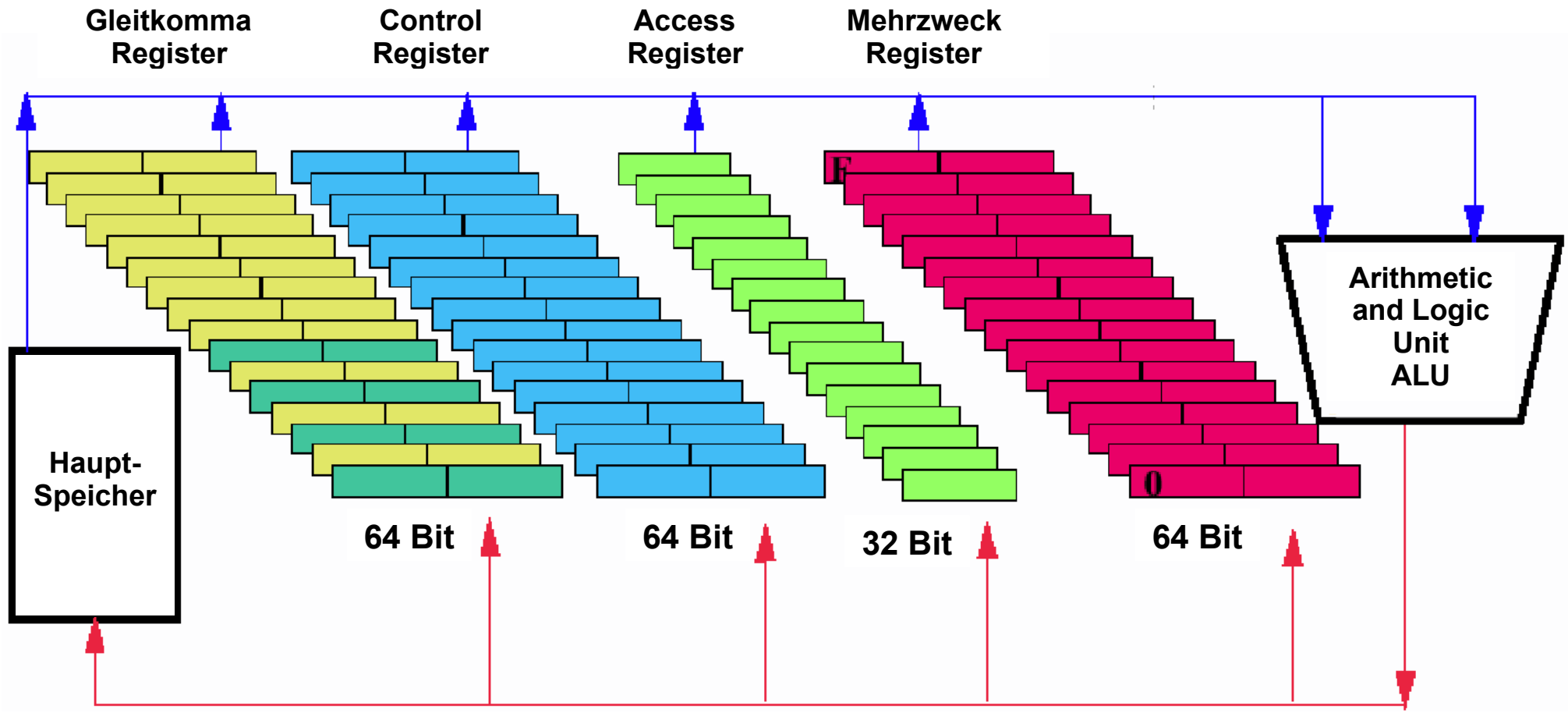
Es wird geschätzt, dass Unternehmen etwa 50% ihres jährlichen Budgets für Anwendungssoftware in Neuentwicklungen und 50% in die Administration und Pflege von Legacy Software investieren.

Anzahl der Register mehrerer Mainstream Architekturen

Architecture	Integer registers	Double FP registers
x86	8	8
x86-64	16	16
IBM/360	16	4
Z/Architecture	16	16
Itanium	128	128
UltraSPARC	32	32
POWER	32	32
Alpha	32	32
6502	3	0
ARM	16	16

Eine zu große Anzahl von Registern verringert die Performance, weil bei jedem Prozesswechsel der Inhalt der Register abgespeichert (saved) und neue Inhalte geladen werden müssen. Optimal sind durchschnittlich 25 Register, abhängig vom Profil der bearbeiteten Anwendungen. Da 25 keine Binärziffer ist, sind 16 oder 32 Register optimal.

System z Programmiermodell



Anmerkung: Die System z Architektur ist eine 64 Bit Architektur. Ähnlich wie x86, PowerPC und Sparc Architekturen ist sie aus einer ursprünglichen 32 Bit Version weiterentwickelt worden. Hierbei unterstützen die System z-Rechner einen 32-Bit- und einen 64-Bit-Modus. Das Umschalten zwischen beiden Modi ist sehr einfach.

Die obige Abbildung zeigt die Bestandteile einer Mainframe CPU, die für den Programmierer sichtbar sind und die von Maschinenbefehlen manipuliert werden können. Man bezeichnet diese Teile auch als das Programmiermodell. Zu beachten ist, dass die CPU außerdem viele Teile und Funktionen enthält, die für einen Programmierer nicht zugänglich und unsichtbar sind.

Das Programmiermodell besteht aus dem Hauptspeicher, einer Verarbeitungseinheit (ALU) die Funktionen wie z.B. Addition ausführen kann und einer ganze Reihe von Registern, die Daten, Hauptspeicheradressen oder Steuerfunktionen speichern können. Der Bestand an System z Registern umfasst:

- 16 Mehrzweck Register (General Purpose Register, GPR, 64 Bit) speichern Adressen oder Daten.
- 16 Gleitkommaregister (64 Bit) speichern Zahlen im Gleitkommaformat,
- 16 Control Register (64 Bit) speichern Steuerfunktionen. Ein Beispiel ist Steuerregister 1, welches die Anfangsadresse der Seitentafel im Hauptspeicher enthält,
- 16 Access Register (32 Bit) werden für spezielle Hauptspeicher Zugriffsfunktionen benutzt,

Zusätzlich existiert ein 64 Bit Befehlszähler Register und ein Flag Bit Register. Der Befehlszähler enthält die Adresse des nächstens auszuführenden Maschinenbefehls. Das Flag Register enthält individuelle Bits. Ein Beispiel ist ein Bit, welches den Rechner entweder in den Benutzerstatus oder in den Überwacherstatus versetzt.

Als eine Besonderheit der System z Architektur werden Befehlzähler und Flag Bits zu einem einzigen 128 Bit langen Register, dem „Programm Status Wort“ (PSW) Register zusammengefasst. Vom Standpunkt der Ausführung eines Programms definiert der Inhalt des PSW in jedem Augenblick den Status der CPU.

Bei einem CPU Chip mit mehreren Cores enthält jeder Core einen eigenen Satz der hier gezeigten Register.

16 und 32 Bit Maschinenbefehle

Die System z Maschinenbefehle haben alle eine einheitliche Länge von entweder 16 Bit, 32 Bit oder 48 Bit. Vor allem der geschickte Entwurf der 16 Bit Maschinenbefehle führt dazu, dass ein kompiliertes Programm auf einem System z Rechner fast immer 20 – 30 % weniger Platz im Hauptspeicher (Arbeitsspeicher) benötigt, als bei fast allen anderen Rechnerarchitekturen. Dies gilt spezifisch auch bei einem Vergleich der System z Architektur gegenüber der x86 Architektur.

Vor allem auf Grund der reduzierten Cache Bandbreite ist dies ein Performance Vorteil.

Es ist interessant zu sehen, dass die Firma Arm Holdings mit der Einführung der Thumb Technologieerweiterung für die ARM Architektur ein ähnliches Ziel anstrebt. Vor allem die jüngste Thumb-2 Technologie Erweiterung weist an dieser Stelle überraschende konzeptuelle Ähnlichkeiten mit der 1964 entstandenen S/360 Architektur auf.

Hiermit wiederholt sich die Frage: Wie konnte es sein, dass Amdahl, Blaauw und Brooks beim Entwurf der S/360 Architektur im Jahre 1964 alles richtig gemacht haben?

Decimal Floating Point

System z unterstützt drei Gleitkommaformate: Hexadezimal, IEEE 754 und Dezimal.

Das **hexadezimale Format** ist ein IBM proprietärer Binär-Standard und fast nur auf Mainframe Rechnern anzutreffen. Die meisten auf Mainframes gespeicherten Gleitkommadaten verwenden das hexadezimale Format.

Das **IEEE 754 Format** ist international weit verbreitet, und wird u.a. von der x86 Architektur benutzt. Auf Mainframes benutzt vor allem das zLinux Betriebssystem diesen Standard.

Das **Dezimale Gleitkommaformat (DFP)** ist erst seit wenigen Jahren verfügbar. Im Gegensatz zu den beiden anderen Formaten werden Mantisse und Exponent mit Dezimalziffern dargestellt. Dies führt zu einer gewissen Performance- und Genauigkeitseinbuße.
Die Benutzung von Dezimal Arithmetik ist in der Wirtschaft weit verbreitet. DFP vermeidet Rundungsfehler und andere Probleme bei der Konvertierung von Dezimaldaten in Binärdaten und umgekehrt, und gewinnt deshalb an Bedeutung.

Die Gleitkomma-Einheit des System z Prozessors unterstützt alle drei Gleitkomma-Arten. Ebenso speichert die z/OS DB2 Datenbank Daten in allen drei Formaten.

Darstellung Alphanumerischer Daten
ASCII / EBCDIC

Es existieren drei weit verbreitete Standards für die Darstellung alphanumerischer (Buchstaben, Ziffern, Sonderzeichen) Daten.

Die Darstellung von alphanumerischen Zeichen geht bei allen Rechnerarchitekturen auf uralte Wurzeln zurück. Bei vielen Rechnern ist dies die 7-Bit-**ASCII**-Darstellung (**American Standard Code for Information Interchange**), die ihren Ursprung in den Lochstreifen der Teletype-Maschinen hat und nachträglich auf 8 Bit erweitert wurde. Bei den Mainframe- (und einigen anderen) Rechnern ist dies die 8-Bit-**EBCDIC**-Darstellung (**Extended Binary Coded Decimal Interchange Code**, ausgesprochen [ebsidik]), die ihren Ursprung in den Lochkarten hat. Das Ergebnis ist eine Spaltung der IT-Welt. Etwa 60% aller geschäftlich relevanten alphanumerischen Daten sind im EBCDIC-Format gespeichert und etwa 40% im ASCII-Format. Wenn immer ASCII - und EBCDIC-Rechner miteinander kommunizieren, sind unschöne Konvertierungsverfahren erforderlich.

Mainframes unterstützen neben dem EBCDIC- auch das ASCII-Format. Letzteres wird z.B. von dem zLinux Betriebssystem genutzt, ist aber sonst nur wenig gebräuchlich. Man kann davon ausgehen, dass alphanumerische Daten auf einem Mainframe in der Regel im EBCDIC-Format vorliegen.

Neben EBCDIC und ASCII können Mainframes auch Daten im **UTF-8** bzw. **UTF-16** Format verarbeiten
Die Java Plattform verwendet UTF16 in Character Arrays und Strings.

ASCII- und EBCDIC Zeichentabellen

Beispiele: **ASCII** **R = Hex 52**
EBCDIC **R = Hex D9**

ASCII Table

Codepage 1252 - Latin 1 Windows

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-		0001	0002	0003	0004	0005	0006	0007	0008	0009	000A	000B	000C	000D	000E	000F
1-	0010	0011	0012	0013	0014	0015	0016	0017	0018	0019	001A	001B	001C	001D	001E	001F
2-	0020	! 0021	" 0022	# 0023	$ 0024	% 0025	& 0026	' 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
3-	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
4-	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
5-	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
6-	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
7-	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	007F
8-	€ 20AC	0081	‚ 201A	ƒ 0192	„ 201E	… 2026	† 2020	‡ 2021	ˆ 02C6	‰ 2030	Š 0160	‹ 2039	Œ 0152	008D	Ž 017D	008F
9-	0090	‘ 2018	’ 2019	“ 201C	” 201D	• 2022	– 2013	— 2014	˜ 02DC	™ 2122	š 0161	› 203A	œ 0153	009D	ž 017E	Ÿ 0178
A-	00A0	¡ 00A1	¢ 00A2	£ 00A3	¤ 00A4	¥ 00A5	¦ 00A6	§ 00A7	¨ 00A8	© 00A9	ª 00AA	« 00AB	¬ 00AC	- 00AD	® 00AE	¯ 00AF
B-	° 00B0	± 00B1	2 00B2	3 00B3	´ 00B4	µ 00B5	¶ 00B6	· 00B7	¸ 00B8	1 00B9	º 00BA	» 00BB	¼ 00BC	½ 00BD	¾ 00BE	¿ 00BF
C-	À 00C0	Á 00C1	Â 00C2	Ã 00C3	Ä 00C4	Å 00C5	Æ 00C6	Ç 00C7	È 00C8	É 00C9	Ê 00CA	Ë 00CB	Ì 00CC	Í 00CD	Î 00CE	Ï 00CF
D-	Ð 00D0	Ñ 00D1	Ò 00D2	Ó 00D3	Ô 00D4	Õ 00D5	Ö 00D6	× 00D7	Ø 00D8	Ù 00D9	Ú 00DA	Û 00DB	Ü 00DC	Ý 00DD	Þ 00DE	ß 00DF
E-	à 00E0	á 00E1	â 00E2	ã 00E3	ä 00E4	å 00E5	æ 00E6	ç 00E7	è 00E8	é 00E9	ê 00EA	ë 00EB	ì 00EC	í 00ED	î 00EE	ï 00EF
F-	ð 00F0	ñ 00F1	ò 00F2	ó 00F3	ô 00F4	õ 00F5	ö 00F6	÷ 00F7	ø 00F8	ù 00F9	ú 00FA	û 00FB	ü 00FC	ý 00FD	þ 00FE	ÿ 00FF

EBCDIC-Tabelle

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	PF	HT	LC	DEL			SMM	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	TM	RES	NL	BS	IL	CAN	EM	CC	CU1	IFS	IGS	IRS	IUS
2	DS	SOS	FS		BYP	LF	ETB	ESC			SM	CU2		ENQ	ACK	BEL
3			SYN		PN	RS	UC	EOT				CU3	DC4	NAK		SUB
4	SP										¢	.	<	(	+	|
5	&										!	$	*	)	;	¬
6	-	/									¦	,	%	_	>	?
7											:	#	@	'	=	"
8		a	b	c	d	e	f	g	h	i						
9		j	k	l	m	n	o	p	q	r						
A		~	s	t	u	v	w	x	y	z						
B										`						
C		A	B	C	D	E	F	G	H	I						
D		J	K	L	M	N	O	P	Q	R						
E	\		S	T	U	V	W	X	Y	Z						
F	0	1	2	3	4	5	6	7	8	9						

Unicode / UTF-8 / UTF-16

Mit den 128 Möglichkeiten des 7-bit ASCII Codes bzw. den 256 Zeichen von EBCDIC konnte man gerade die gängigsten Zeichen der westlichen Alphabethe und einige Sonderzeichen darstellen, aber nicht die Zeichen aller Sprachen dieser Welt. Deshalb haben Entwickler von Xerox und Apple ca 1987 begonnen, einen **Universal Character Set UCS** mit 16 Bits zu definieren. Es stellte sich aber schnell heraus, dass auch 2^{16} = 65535 Zeichen nicht ausreichen und nachdem die Arbeitsgruppe um Mitarbeiter von Sun, Microsoft, usw. erweitert wurde, entstand 1990 die 1. Version von Unicode.

Unicode wurde ein Standard zur Darstellung und Verarbeitung von Texten und besteht aus 17 Ebenen (Planes), welche jeweils 2^{16} Zeichen umfassen. Jedes Zeichen wird durch die Ebene und die jeweilige Positionsnummer (Codepoint) identifiziert. Die Unicode-Kodierungstabellen haben insgesamt 1.114.112 Codepoints und nur etwa 10% der Codepoints sind aktuell mit Zeichen belegt.

Unicode kann mit verschiedenen Zeichensätzen kodiert werden. Die gebräuchlichsten Kodierungen sind UTF-8 und UTF-16 .
UTF-8 (8-bit Unicode Transformation Format) benutzt 1 bis 4 Bytes zur Abbildung eines Unicode Zeichens.

Beispiele für UTF-8 Kodierungen

Zeichen	Unicode	Unicode binär	UTF-8 binär	UTF-8 hexadezimal
Buchstabe *y*	U+0079	00000000 01111001	01111001	79
Buchstabe *ä*	U+00E4	00000000 11100100	11000011 10100100	C3 A4
Zeichen für eingetragene Marke ®	U+00AE	00000000 10101110	11000010 10101110	C2 AE
Eurozeichen €	U+20AC	00100000 10101100	11100010 10000010 10101100	E2 82 AC
Violinschlüssel 𝄞	U+1D11E	00000001 11010001 00011110	11110000 10011101 10000100 10011110	F0 9D 84 9E

https://de.wikipedia.org/wiki/UTF-8

UTF-16 (16-bit Unicode Transformation Format) ist eine Zeichenkodierung, die alle möglichen Unicode-Characters darstellen kann. Die Kodieriung hat variable Länge und die Codepoints sind entweder mit 1 oder 2 16-bit *code units*.

Beispiele für UTF-16 Kodierungen

Zeichen	Unicode	Unicode binär	UTF-16BE binär	UTF-16BE hexadezimal
Buchstabe *y*	U+0079	00000000 01111001	00000000 01111001	00 79
Buchstabe *ä*	U+00E4	00000000 11100100	00000000 11100100	00 E4
Eurozeichen €	U+20AC	00100000 10101100	00100000 10101100	20 AC
Violinschlüssel 𝄞	U+1D11E	00000001 11010001 00011110	11011000 00110100 11011101 00011110	D8 34 DD 1E
CJK-Ideogramm 妹	U+24F5C	00000010 01001111 01011100	11011000 01010011 11011111 01011100	D8 53 DF 5C

https://de.wikipedia.org/wiki/UTF-16

Big Endian versus Little Endian

Eine Hauptspeicheradresse adressiert ein einziges Byte im Hauptspeicher. Unter der *„Endianess“* eines Rechners versteht man die Reihenfolge, in der eine Gruppe von Bytes im Hauptspeicher abgespeichert wird.

Nehmen wir einen Maschinenbefehl an, der 4 aufeinanderfolgende Bytes in ein Mehrzweckregister der Zentraleinheit lädt. Werden die Bytes in aufsteigender oder in absteigender Reihenfolge geladen?

Wenn Halbworte oder Worte im Hauptspeicher gespeichert sind, dann befindet sich an der adressierten Hauptspeicherstelle:

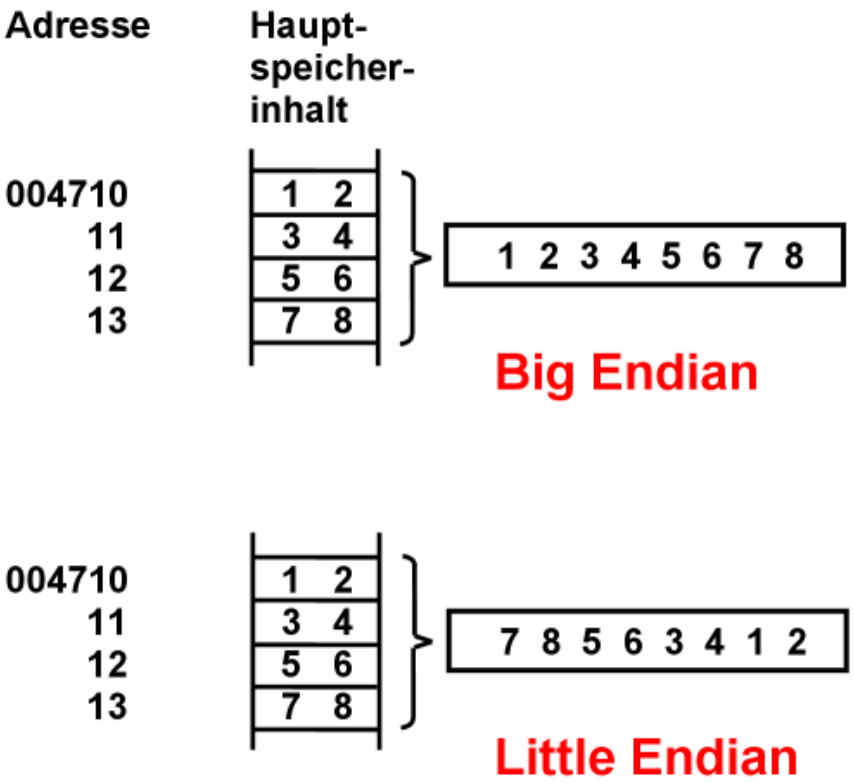
- Das **wertniedrigste** Byte bei **Little** Endian Rechnern
- Das **werthöchste** Byte bei **Big** Endian Rechnern

Die Bytes eines Halbwortes oder Wortes werden bei Little Endian Rechnern in umgekehrter Reihenfolge abgespeichert als bei Big Endian Rechnern.

Anmerkung (http://www.searchnetworking.de/definition/Big-Endian-und-Little-Endian **):**
*Die Begriffe Big-Endian und Little-Endian stammen aus dem Buch „**Gullivers Reisen**“ von **Jonathan Swift**.*
Big-Endian beschreibt eine der politischen Fraktionen, die ihre Eier vom größeren Ende aus schält (der „primitive Weg“). Sie rebellieren gegen den Lilliputanerkönig, da dieser von seinen Untergebenen (die Little Endians) erwartet, dass sie ihre Eier vom kleineren Ende aus öffnen.

Der Unterschied ist in den folgenden Abbildungen dargestellt.

Es enthält die Hauptspeicheradresse ***004710***

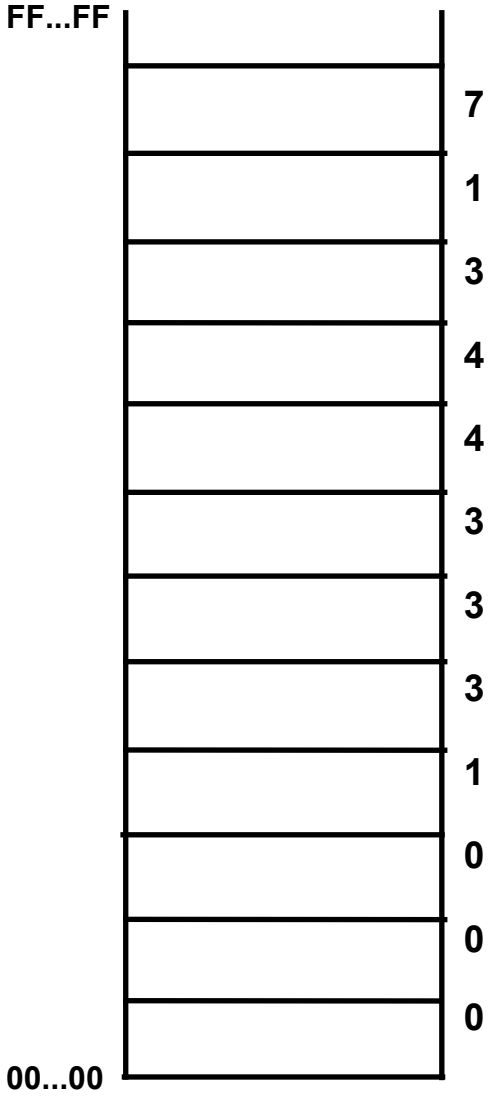


$439041101_D = 1A2B3C4D_H$

	Big Endian		Little Endian	
Adresse	Hex	Dez	Hex	Dez
10000	1A	26	4D	77
10001	2B	43	3C	60
10002	3C	60	2B	43
10003	4D	77	1A	26

Mainframes, die meisten Unix Rechner und das Internet verwenden die Big Endian Architektur.
x86 und Ethernet verwenden die Little Endian Architektur.

Für Konvertierungszwecke enthält die x86 Architektur einen „Byte Swap“ Maschinenbefehl, mit dessen Hilfe die Endianess eines Feldes im Hauptspeicher umgedreht werden kann.
Für Netzwerkkommunikation werden die Funktionen `htonl( )` / `htons( )` bzw. `nothl( )` / `ntohs( )` im BSD-IP Socket API angeboten.

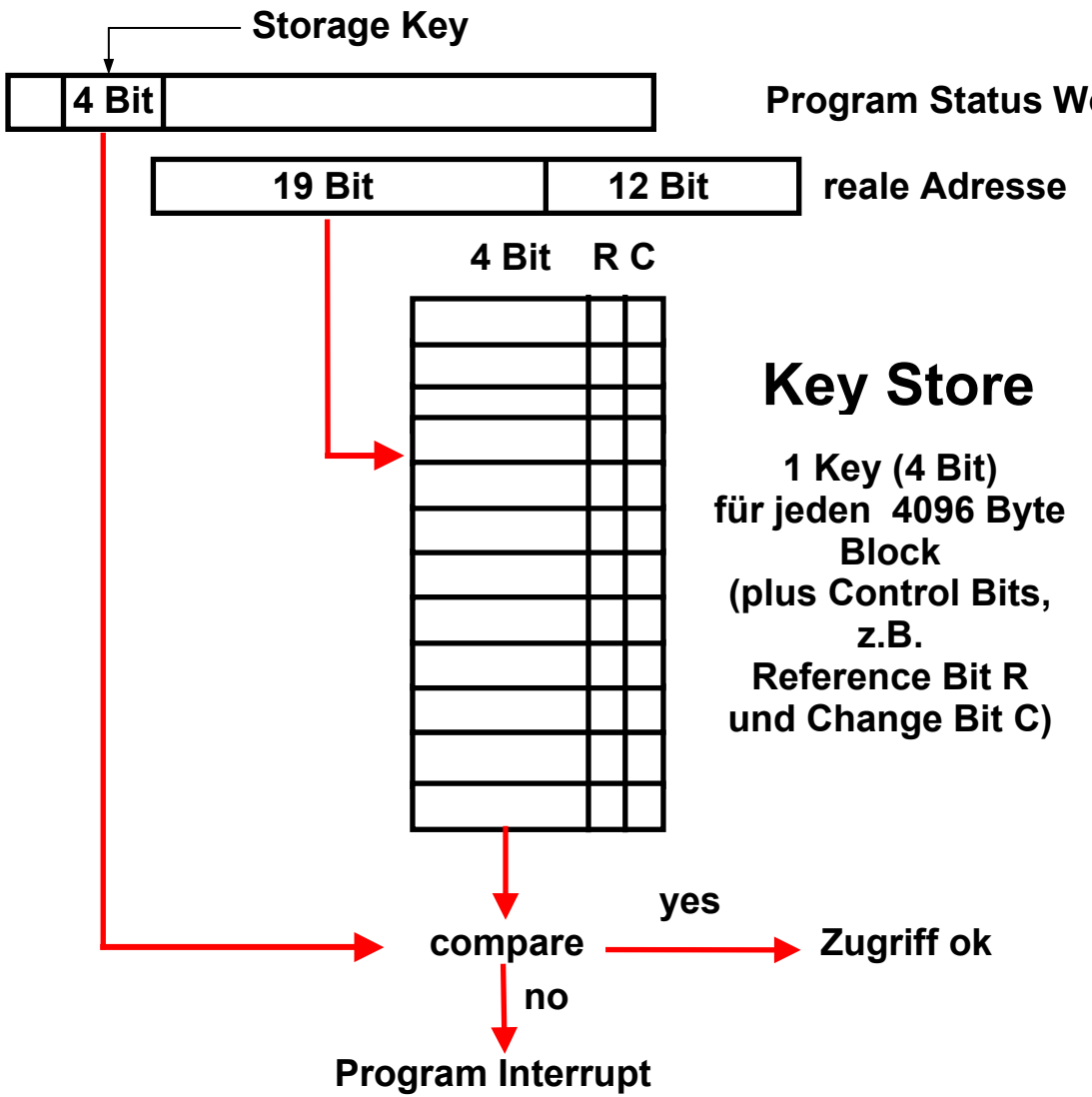


System z Speicherschutz

Hardware-Speicherschutz (**Storage Protection**) ist eine einzigartige Einrichtung der Mainframes. Der Hardware-Speicherschutz teilt den Hauptspeicher in 4096 Byte große Blöcke auf und ordnet jedem dieser Blöcke einen 4 Bit Speicherschutzschlüssel (**Storage Key**) zwischen 0 ... 15 zu, der in einem getrennten Schnellspeicher abgespeichert wird. Der Speicherschutzschlüssel wird in einem 4-Bit-Feld des CPU Status Registers (**Program Status Word**, PSW*)* gespeichert. Bei jedem Speicherzugriff wird aus einem Schnellspeicher dieser Speicherschutzschlüssel ausgelesen und mit dem 4-Bit-Feld im PSW verglichen. Nur wenn dieser Vergleich positiv ausfällt (4-Bit-Felder sind identisch), erfolgt der Zugriff.

Die einzelnen Prozesse haben unterschiedliche Speicherschutzschlüssel. Somit wird verhindert, dass die Programme eines Prozesses auf Speicherbereiche eines anderen Prozesses zugreifen können.

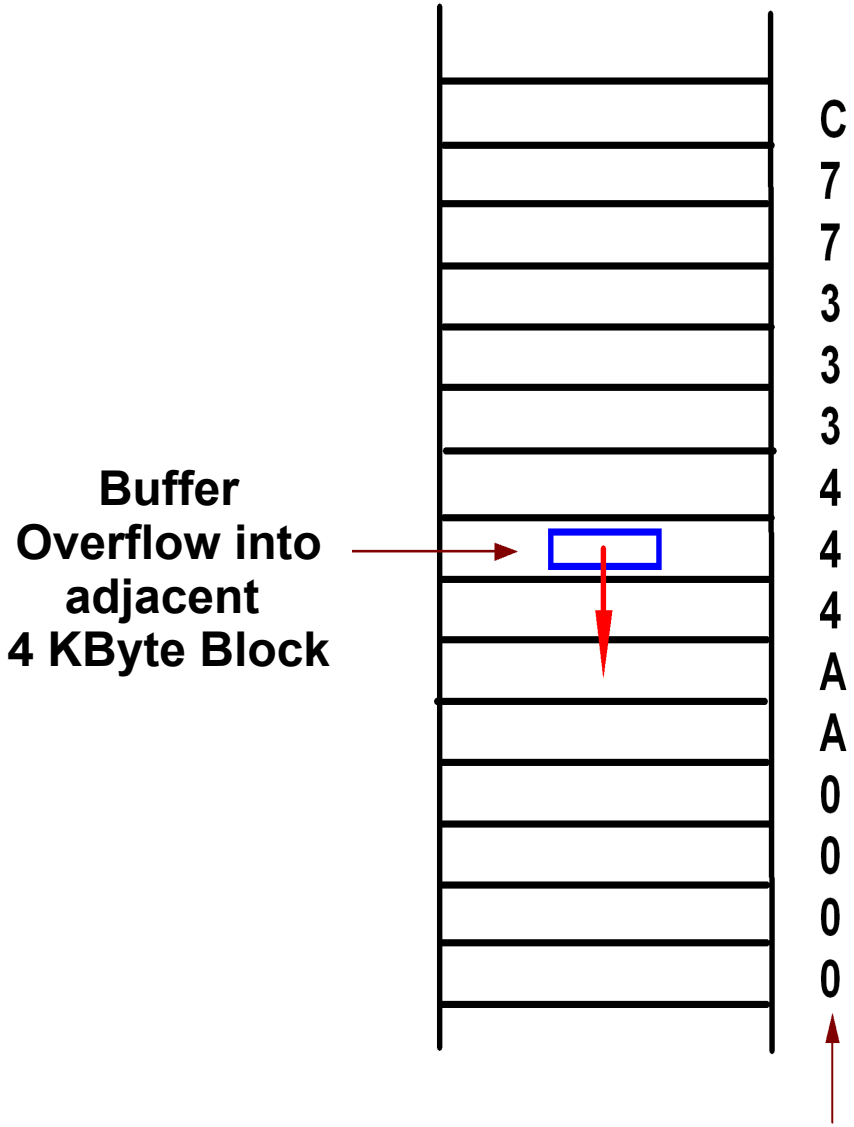
Die hierfür erforderliche Logik ist in der folgenden Abbildung dargestellt.



Storage Protection Keys

Bei jedem Hauptspeicher-zugriff wird der Wert im Key Store für den adressierten 4 KByte Block mit dem Wert im PSW verglichen. Nur bei Übereinstimmung wird der Zugriff durchgeführt.

Literatur: ABCs of z/OS System Programming, Volume 10 http://www.redbooks.ibm.com/redbooks/pdfs/sg246990.pdf
Section 1.24, S. 43



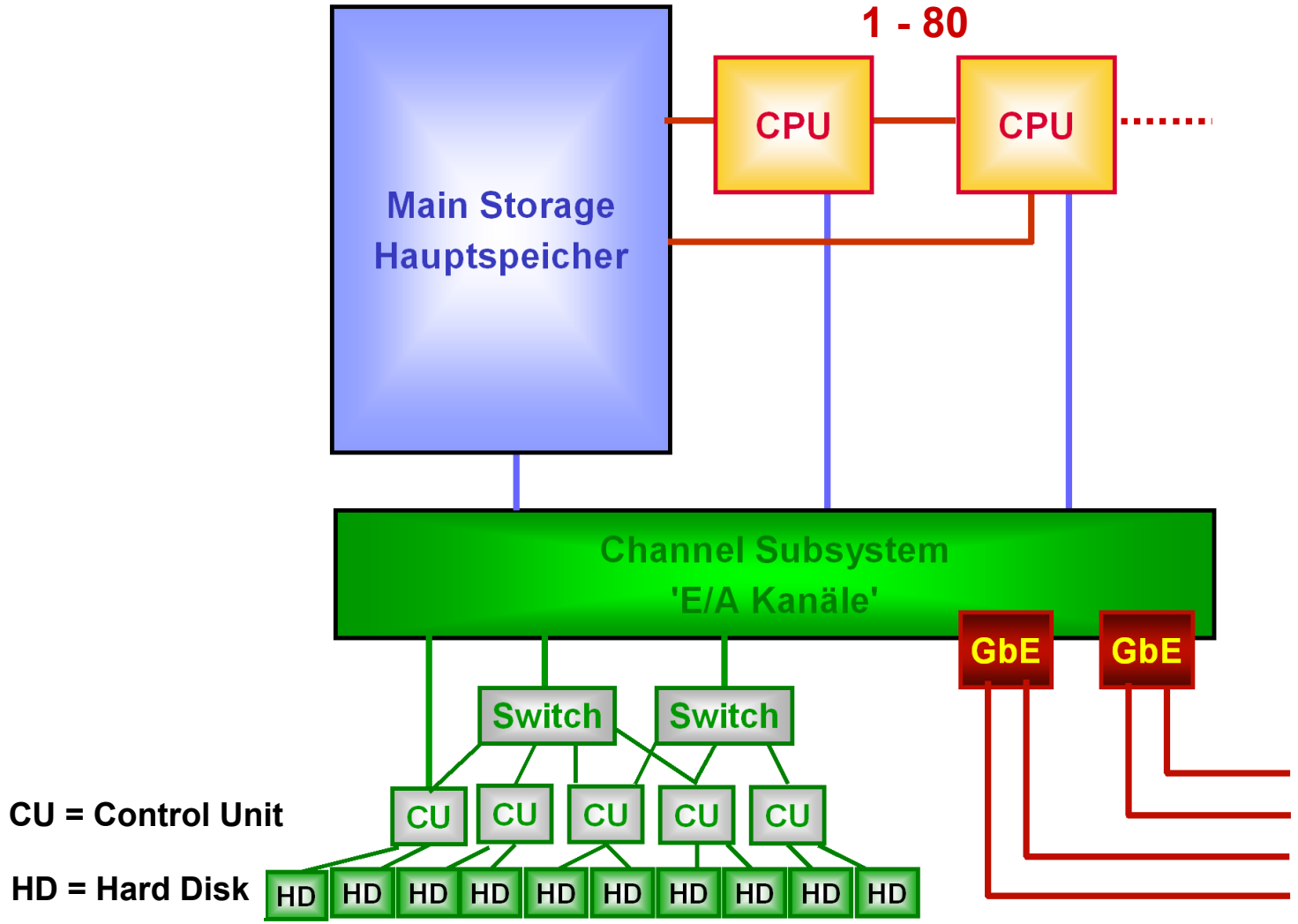
Buffer Overflow Prevention

Der Hauptspeicher ist in 4 KByte große Blöcke mit unterschiedlichen Protection Keys aufgeteilt. Dies verhindert ein bekanntes Sicherheitsproblem, den Buffer-Overflow.

Versucht ein Prozess auf den (realen) Hauptspeicherbereich eines anderen Prozesses zuzugreifen, so verhindert die Storage Protection Einrichtung den Zugriff, wenn die Storage Keys ungleich sind.

Vor allem der Zugriff auf Betriebssystem-Kernel Funktionen wird damit ausgeschlossen.

Mainframe Configuration



Ein Mainframe System besteht zumindest aus einem oder mehreren (bis zu 101) Prozessoren, einem Hauptspeicher, und Anschlüssen für Ein/Ausgabe (E/A) Geräte, besonders Plattenspeicher (Hard Disk) und Netzwerkanschlüsse, häufig Gigabit oder 10 GBit Ethernet (GbE). Plattenspeicher und andere Input/Output Geräte (I/O) werden grundsätzlich über *Control Units* (CU) angeschlossen.

Control Unit

Bei einem PC erfolgt die Ansteuerung eines Plattenspeichers durch eine als I/O Driver bezeichnete Komponente des Betriebssystems. Die Ausführung des I/O Driver Codes benötigt CPU Zeit.

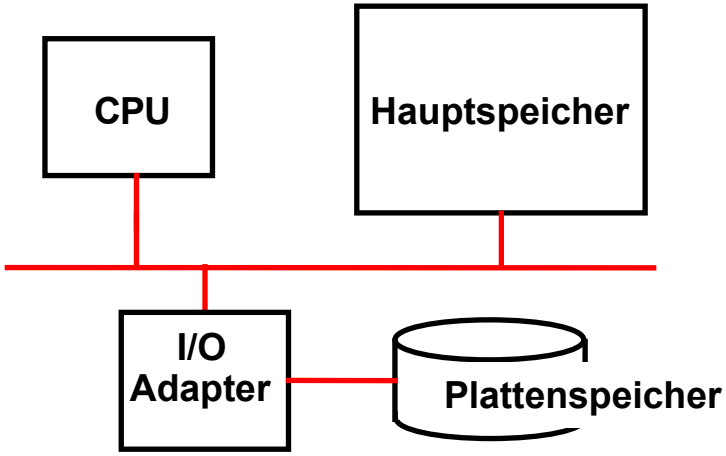
An viele PCs ist nur ein einziger Plattenspeicher angeschlossen. An einen Mainframe können Tausende, zehntausende oder hunderttausende von Plattenspeichern angeschlossen sein. Zur Entlastung der CPU(s) wird die I/O Driver Abarbeitung auf dezentrale Spezialrechner ausgelagert, die als Control Units bezeichnet werden. Ohne diese Auslagerung wäre ein Mainframe Rechner niemals in der Lage, die erforderliche hohe Ein/Ausgabeleistung, besonders Plattenspeicherdurchsatz, zu erbringen.

Zum Anschluss der Control Units und Plattenspeicher wird ein Netzwerk von Verbindungen benötigt, die als Kanäle (Channels) bezeichnet werden. Kanäle werden den einzelnen I/O Anforderungen dynamisch zugeordnet, was ebenfalls sehr verarbeitungsaufwendig ist. Um hiervon die CPU(s) zu entlasten, erfolgt die Ansteuerung der Kanäle durch eine getrennte Komponente, das „*Channel Subsystem*“.

Wir werden uns die Ein/Ausgabe Ansteuerung später noch genauer ansehen.

Verarbeitungsgrundlagen Teil 2

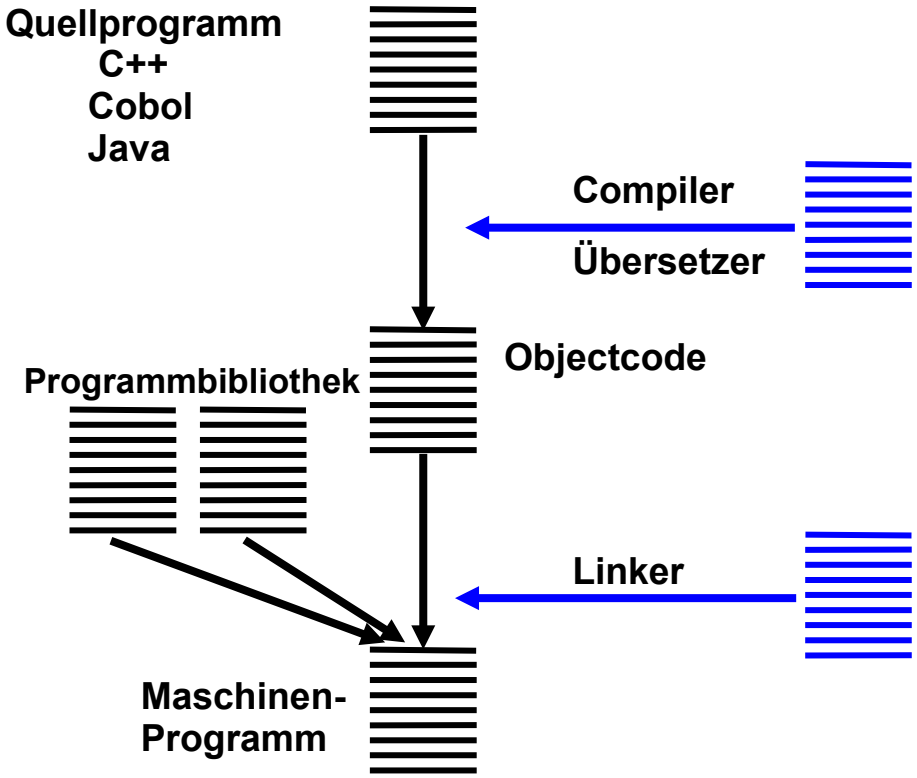
Multiprogramming



Ein moderner Rechner besteht aus einem Prozessor (auch Central Processor Unit – **CPU** genannt), die ein Programm ausführt; einem **Hauptspeicher** (Main Storage, oft auch Arbeitsspeicher genannt), welcher das auszuführende Programm und temporäre Daten enthält, und einem Anschluss (**I/O Adapter**) für einen (oder viele) Plattenspeicher, auf dem ausführbare Programme in Bibliotheken (Libraries) und Daten in Dateien und Datenbanken gespeichert sind.

Ein auszuführendes Programm besteht aus einer Folge von Maschinenbefehlen (Machine Instructions). Das Format der Maschinenbefehle wird durch die Rechnerarchitektur bestimmt. Architekturen wie x86 (Pentium), PowerPC, Sparc, Itanium und System z (Mainframe) haben unterschiedliche Formate der Maschinenbefehle.

Ein Anwendungsprogramm, auch Benutzerprogramm genannt (application program, user program) wird heutzutage fast immer in einer höheren Programmiersprache wie Java, C++, Cobol, PL/1 oder ADA geschrieben, und durch einen Compiler in ein Maschinensprache-Programm übersetzt, welches aus einer Folge von Maschinenbefehlen besteht.



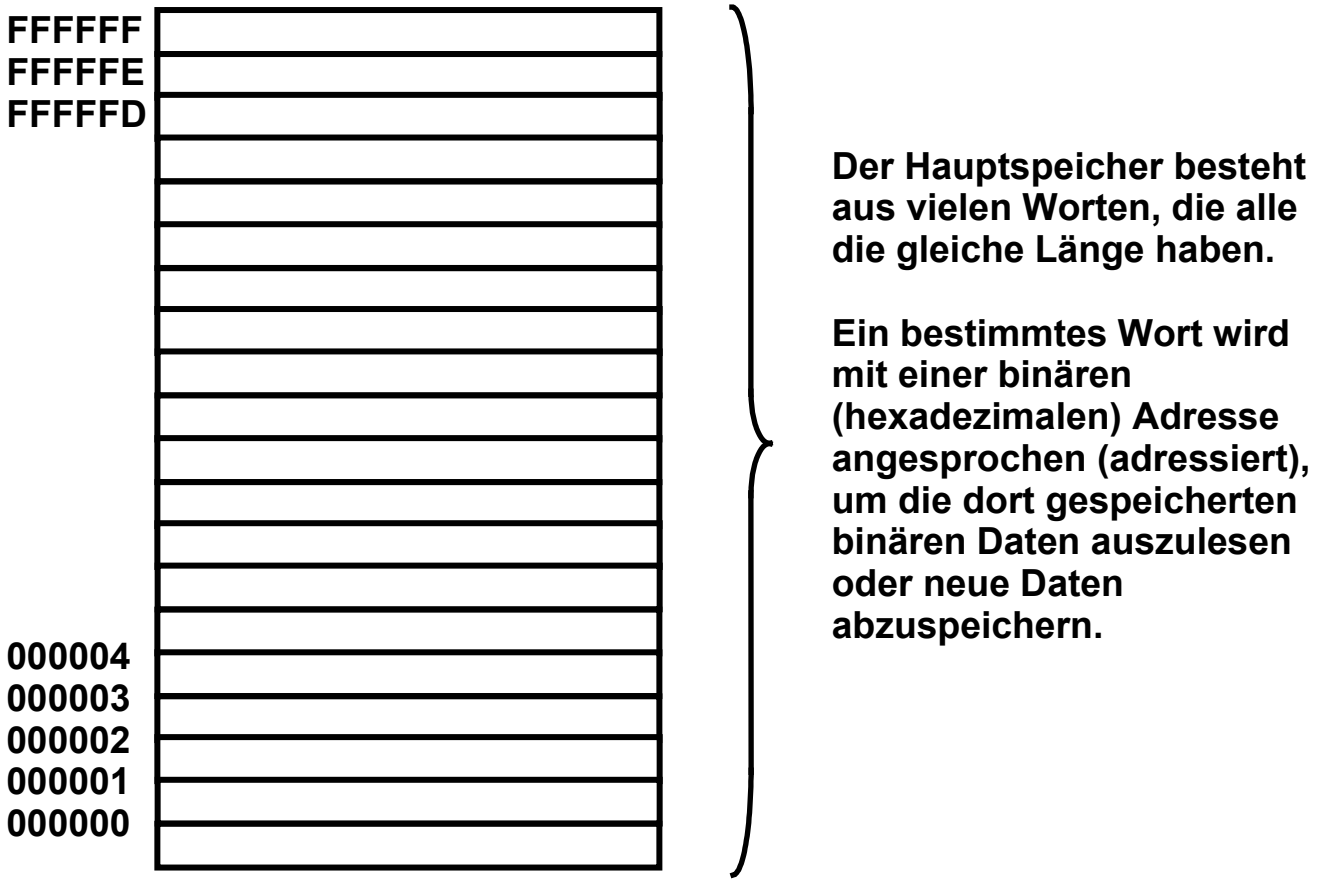
Benutzerprogramme (andere Bezeichnung Anwendungsprogramm, engl.: user programm, application programm) werden in der Regel in einer höheren Programmiersprache wie C++, Cobol, PL/1, Java usw. geschrieben.

Ein Compiler ist ein Programm, dessen Ausführung ein Quellprogramm (Source Program) als Input Daten benutzt, um daraus ein Maschinenprogramm, bestehend aus einzelnen Maschinenbefehlen, zu erstellen. Andere Bezeichnungen: Object Code oder Binaries. Der Aufruf eines Compilers bewirkt eine Übersetzung (Compilation) eines Quellprogramms in ein Maschinenprogramm.

Meistens benutzt man ein weiteres Programm, einen **Linker**, um das übersetzte Maschinenprogrammen mit anderen, bereits früher übersetzten Programmen, zu einem ausführbaren Programm (executable program) zu verknüpfen. Ein Benutzerprogramm kann aus einzelnen Teilen (Modulen) bestehen, die einzeln geschrieben und übersetzt werden. Die übersetzten Maschinenprogramm-Module werden in einem als Programmbibliothek (Library) bezeichneten Verzeichnis auf dem Plattenspeicher untergebracht.

Das Betriebssystem stellt zahlreiche weitere Maschinenprogramm-Module zur Verfügung, die mit Hilfe des Linkers ebenfalls mit einem neu erstellten Maschinenprogramm verknüpft werden. Beispiele sind Ein/Ausgabe (I/O) Routinen oder die Socket Library für Intersystem Kommunikation. Unter dem z/OS Betriebssystem stehen viele solcher Maschinenprogramme in einer mit dem Parameter SYS1 gekennzeichneten Library.

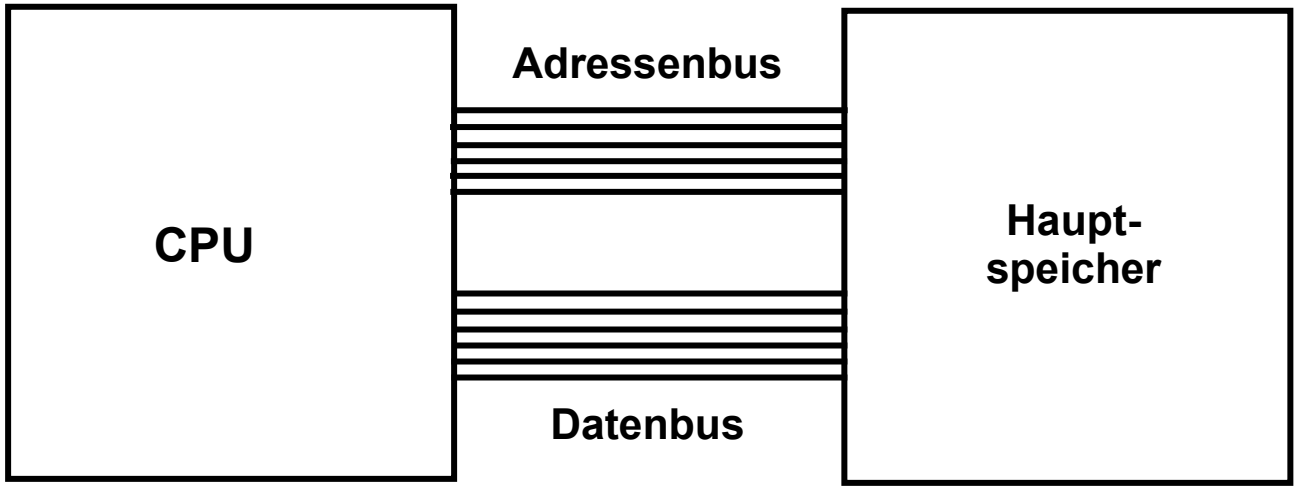
Ein **Loader** ist ein Maschinenprogramm, welches ein ausführbares Programm aus einer Library ausliest und in den Hauptspeicher lädt.



Der Hauptspeicher (main storage) besteht aus lauter gleich langen Wörtern, die mit binären bzw. hexadezimalen Ziffern angesprochen (adressiert), gelesen und gespeichert werden.

Die Wortlänge eines Hauptspeichers ist entweder 8, 12, 16, 14, 32, 36 oder 48 Bit. Moderne Rechner benutzen fast ausschließlich eine Wortlänge von 8 Bit (ein Byte).

Im Hauptspeicher werden sowohl Maschinenprogramme als auch Daten gespeichert.

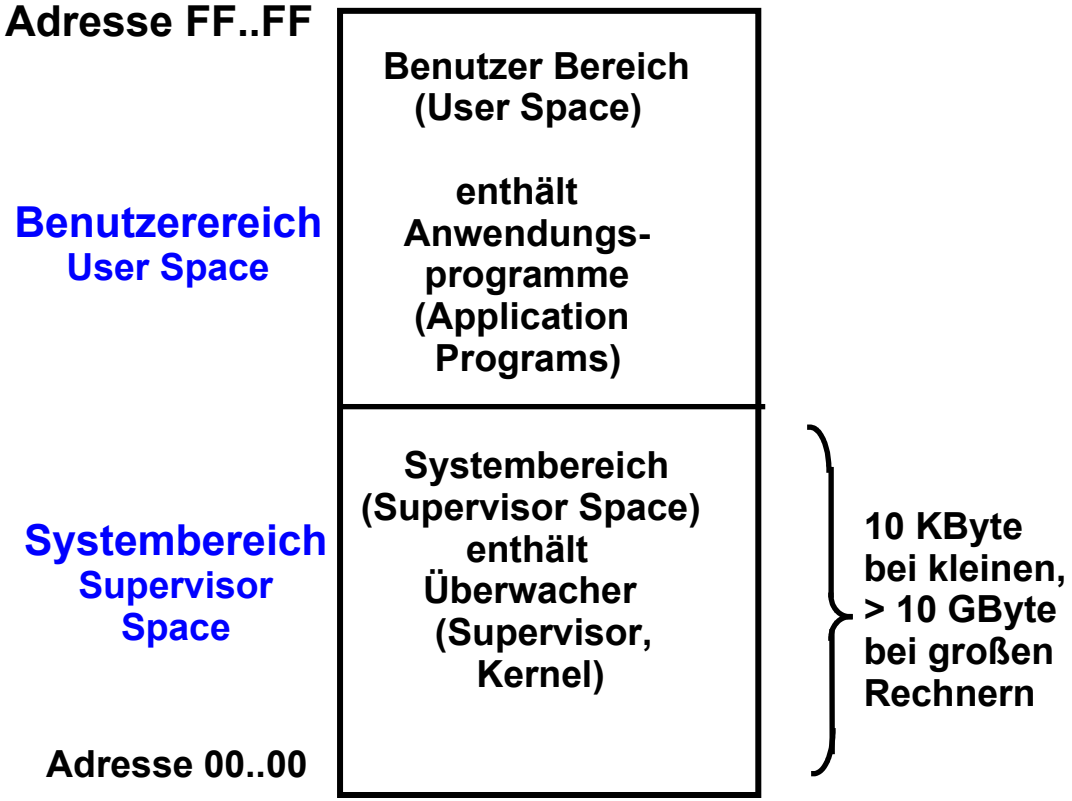


Wir unterscheiden zwischen 32 Bit und 64 Bit Architekturen. Bei einer 32 Bit Architektur ist die CPU mit dem Hauptspeicher mit einem Adressenbus und einem Datenbus verbunden, wobei der **Adressenbus** aus 32 Leitungen besteht, um eine von 2^{32} Adressen an den Hauptspeicher zu übergeben. Damit lassen sich maximal 4 GByte Hauptspeicher adressieren.
Bei einer 64 Bit Architektur sind es 64 Leitungen, um eine von 2^{64} Adressen an den Hauptspeicher zu übergeben.

Der Datenbus besteht aus 8 Leitungen, auf denen jeweils 1 Byte (8 Bit) zwischen Hauptspeicher und CPU übertragen wird.

Die Maschinenbefehle gehen von dieser Struktur aus, die wir als das logische Erscheinungsbild einer CPU bezeichnen. Die physische Implementierung eines Rechners weicht in der Regel von der logischen Sicht ab. So besteht z.B. ein Datenbus häufig aus 256 Leitungen, und es werden gleichzeitig 8 Bytes (256 Bits) zwischen CPU und Hauptspeicher übertragen. Ein moderner Mainframe Rechner (z196) hat einen physischen Hauptspeicher mit einer maximalen Größe von 4 TByte (2^{42} Byte), für dessen Adressierung 42 Leitungen des Adressenbusses ausreichen.

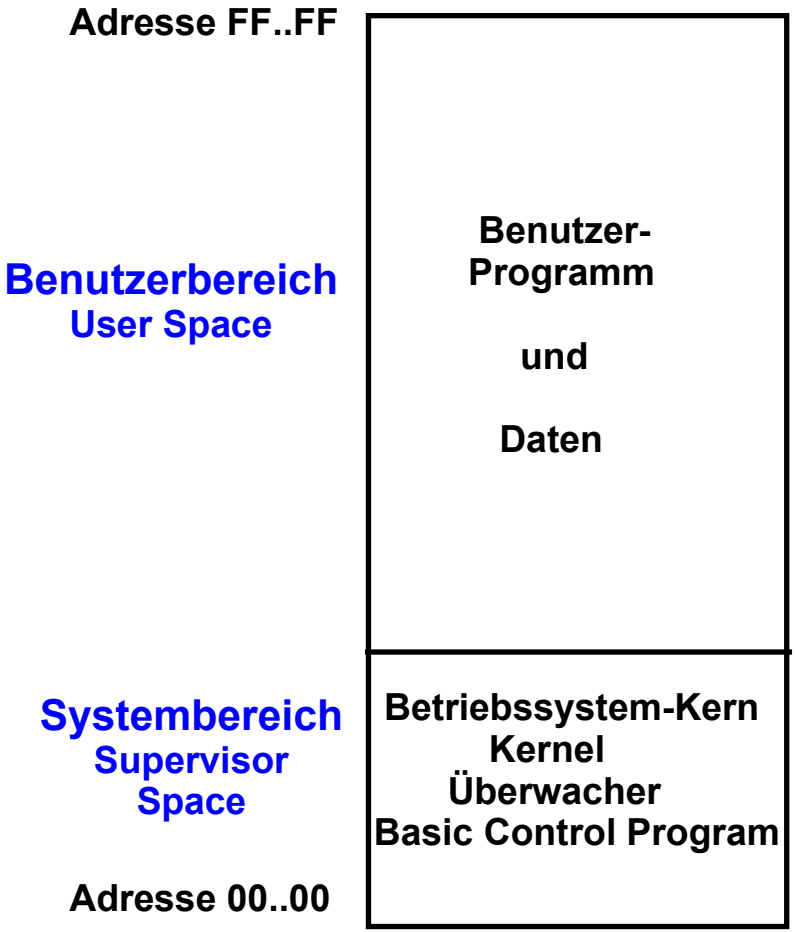
Anmerkung: Bei der Mainframe Architektur verfügt die ehemalige 32 Bit Architektur nur über 31 Bit Adressen, und adressiert damit einen Hauptspeicher mit einer maximalen Größe von 2^{31} Adressen und 2^{31} Byte Speicherkapazität.



Aufteilung des Hauptspeichers in einen System- und einen Benutzerbereich

Ein Betriebssystem enthält Programmroutinen, die von Benutzerprogrammen immer wieder gebraucht werden. Ein Beispiel ist der I/O Driver (Input/Output Driver), der das Lesen oder Schreiben von Daten vom/zum Plattenspeicher bewirkt.

Der Code des Betriebssystems befindet sich auf dem Plattenspeicher. Beim Hochfahren eines Rechners wird ein Teil des Betriebssystems in den Hauptspeicher geladen. Diesen Teil bezeichnet man als Betriebssystem-Kern, Kernel oder Überwacher (Supervisor). z/OS benutzt auch die Begriffe *Nucleus* oder „Basic Control Program“ (BCP).

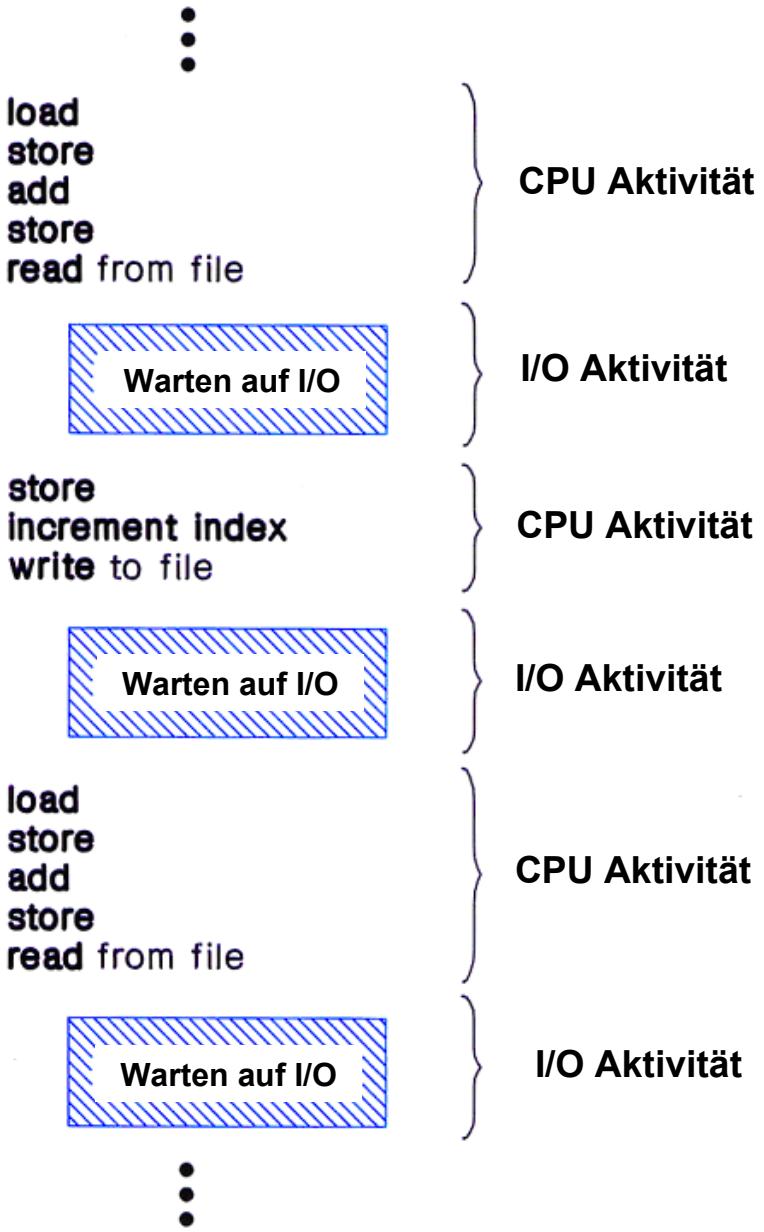


Wir bezeichnen die Menge aller Hauptspeicheradressen als Adressenraum (Address Space). In dem hier dargestellten einfachsten Fall wird der Adressenraum des Hauptspeichers in 2 Teile aufgeteilt: einen Teil für den Überwacher und einen zweiten Teil für das auf dem Rechner laufende Benutzerprogramm (Anwendungsprogramm) und seine Daten. Das ursprüngliche DOS Betriebssystem für den PC hatte diese Struktur.

Der gesamte Adressenraum des Hauptspeichers gliedert sich in einen Überwacher-Adressenraum (Supervisor Address Space) und einem Benutzer-Adressenraum (User Address Space).

Damit ein fehlerhaftes oder böswilliges Benutzerprogramm nicht unbefugt Programme oder Daten im Überwacher-Adressenraum modifizieren kann, läuft der Rechner in jedem Augenblick entweder im Überwacherstatus (Supervisor State) oder im Benutzerstatus (User State, auch als Problem State bezeichnet).

Ein Programm, das im Überwacherstatus läuft, kann auf den gesamten Adressenraum des Hauptspeichers zugreifen. Ein Programm, das im User Modus läuft, hat Zugriffsrechte nur für den User Adressenraum.



Struktur eines Benutzerprogramms

Ein Benutzerprogramm besteht aus einer Folge von

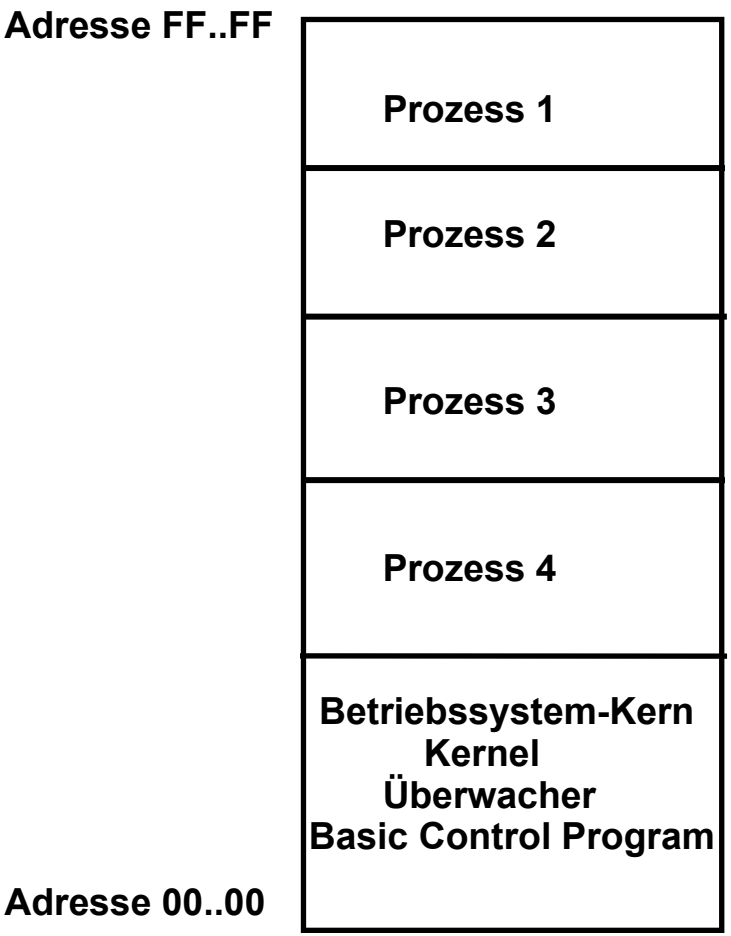
- **Gruppen von Maschinenbefehlen**
- **I/O Operationen (Zugriff auf Plattenspeicher-Daten).**

Die Programmausführung ist eine abwechselnde Folge von CPU und I/O Aktivitäten.

Beim Start einer I/O Operation muss das Anwendungsprogramm in der Regel warten, bis die I/O Operation abgeschlossen ist (z.B. die vom Plattenspeicher gelesenen Daten im Hauptspeicher eingetroffen sind).

Der Zugriff auf einem Plattenspeicher benötigt etwa 10 Millisekunden. Während dieser Zeit könnte die CPU etwa 10 Millionen Maschinenbefehle ausführen. Es wäre unökonomisch, wenn die CPU während der I/O Aktivität warten müsste.

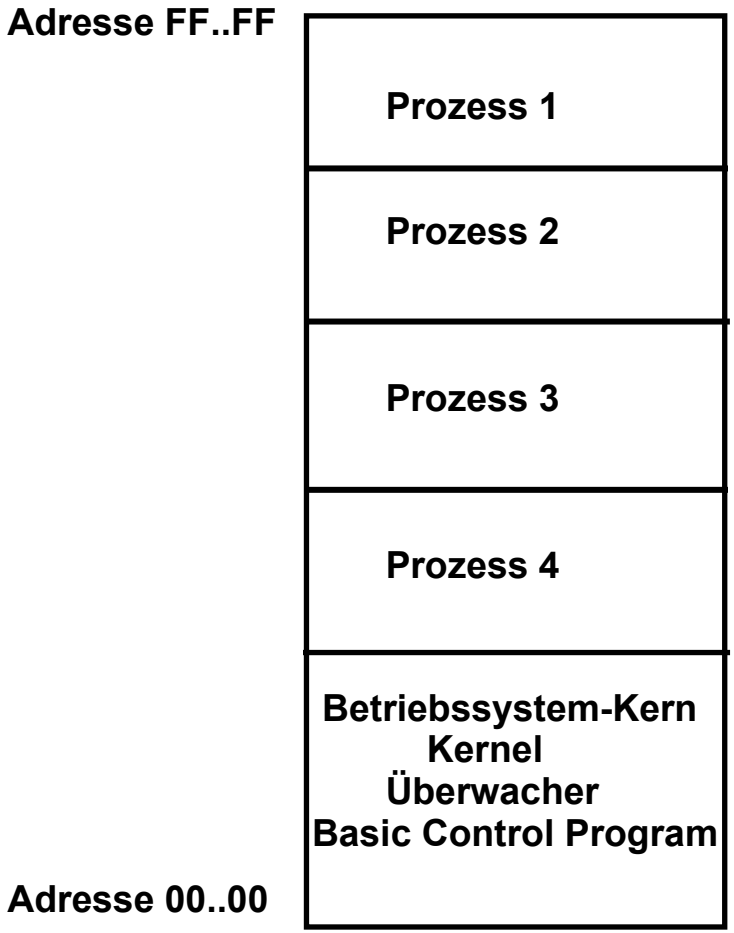
Multiprogrammierung



Hierzu teilt man den Benutzer Adressenraum in mehrere Teilbereiche (*Regions*) auf und ordnet einem Prozess jeweils eine eigene Region zu.

Jetzt befinden sich die Programme (Code) mehrerer Prozesse gleichzeitig im Hauptspeicher. Wir nehmen in diesem Beispiel an, dass der Rechner eine einzige CPU hat. Wenn nun ein Prozess eine I/O Operation ausführt, wird er in einen Wartezustand versetzt. Ein anderer Prozess, dessen Programme und Daten sich ausführungsbereit in seiner Region des Benutzer Adressenraums befinden, wird nun von der CPU ausgeführt, bis er ebenfalls eine I/O Operation ausführen will. Jetzt beginnt die CPU einen dritten Prozess auszuführen usw.

In einem Mainframe Rechner werden auf diese Art typischerweise tausende oder zehntausende von Prozessen parallel ausgeführt. Diese Art der Datenverarbeitung, bei der mehrere Prozesse ineinander geschachtelt parallel verarbeitet werden, wird als „Multiprogrammierung“ (Multiprogramming) bezeichnet.



Die Aufteilung des Benutzer Adressenraums in mehrere Regions für mehrere Prozesse erzeugt mehrere Probleme:

- **Es muss verhindert werden, dass fehlerhafte Programme eines Prozesses auf die Region eines anderen Prozesses zugreift.**

- **Bei Beendigung eines Prozesses wird die benutzte Region des Benutzer Adressenraums freigegeben. In ihr kann jetzt ein neuer Prozess gestartet werden.**

  Die einzelnen Prozesse benötigen aber unterschiedlich viel Speicherplatz. Die Regions haben deshalb eine unterschiedliche Größe. Die Größe der freigewordenen Region entspricht nicht notwendigerweise den Bedürfnissen des neuen Prozesses. Eine komplexe Speicherverwaltungskomponente (als Teil des Überwachers) ist erforderlich.

Zur Lösung dieses Problems führte IBM 1972 die virtuelle Speichertechnik (Virtual Storage) für eine neue Serie von Mainframe Rechnern (System /370) ein.

Prozess

Ein Maschinen-Programm lässt sich mit den Noten eines Musikstückes vergleichen. Ein Konzert ist die Aufführung eines Musikstückes entsprechend den verwendeten Musiknoten.
Ein Prozess ist die Ausführung eines Maschinenprogramms auf einem Computer.

Der Begriff „Prozess“ ist ein zentrales Konzept in der modernen Datenverarbeitung. Ein Prozess ist die Ausführung von einem Programm (oder mehreren Programmen), um ein für einen Benutzer relevantes Problem zu bearbeiten. Beispiele für Prozesse sind die monatliche Lohn- und Gehaltsabrechnung in einem Unternehmen, die Verbuchung bei der Überweisung eines Geldbetrages, eine URL-Abfrage im Internet oder die Stücklistenerstellung beim Bau eines Automobils.

Ein Benutzer Prozess läuft innerhalb des Benutzer Adressenraums. Systemprozesse bearbeiten irgendwelche Teilaufgaben des Betriebssystems.

Die Ausführung eines Prozesses bewirkt in der Regel, dass Daten abgeändert werden, die auf einem Festplattenspeicher permanent gespeichert sind. Ein Beispiel sind Kontodaten bei der Überweisung eines Geldbetrages. Hierzu lädt der Prozess die Daten vom Plattenspeicher in den Hauptspeicher, modifiziert sie und schreibt das Ergebnis auf den Plattenspeicher zurück.

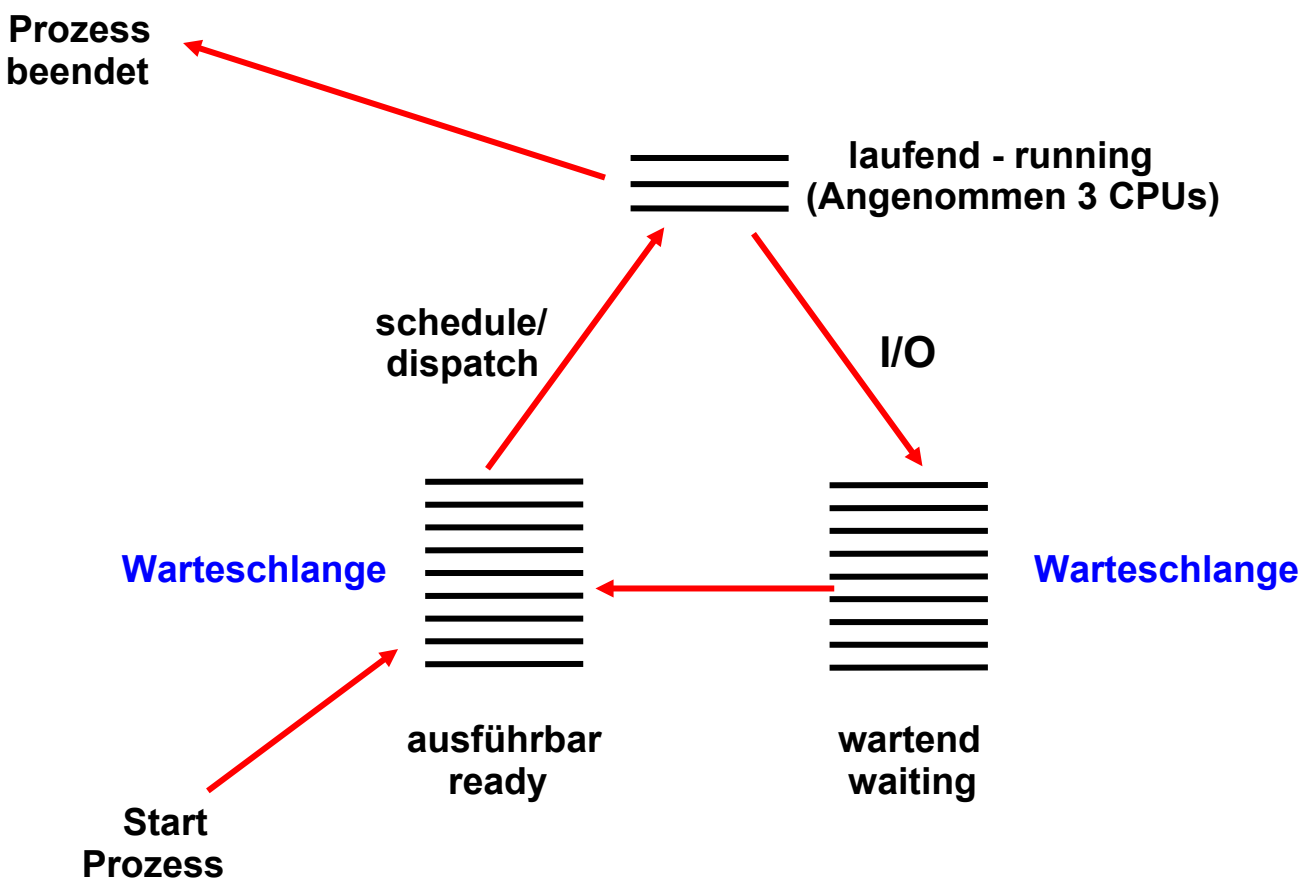
Eine Transaktion ist eine spezielle Art eines Prozesses, bei dem garantiert wird, dass alle Änderungen von Daten auf dem Plattenspeicher entweder vollständig oder gar nicht, nicht aber teilweise, erfolgen.

Zustand von Prozessen

Wenn ein Prozess eine I/O Operation startet, wird er in den Zustand „wartend“ (waiting) versetzt. Wenn die I/O Operation abgeschlossen ist, könnte er wieder von der CPU ausgeführt werden. Da die CPU vermutlich gerade mit einem anderen Prozess beschäftigt ist, wird er stattdessen in den Zustand „ausführbar“ (ready) versetzt. Irgendwann wird die CPU den Prozess weiter verarbeiten. Ein Prozess, der durch die CPU verarbeitet wird, wird als „Laufend“ (running) bezeichnet.

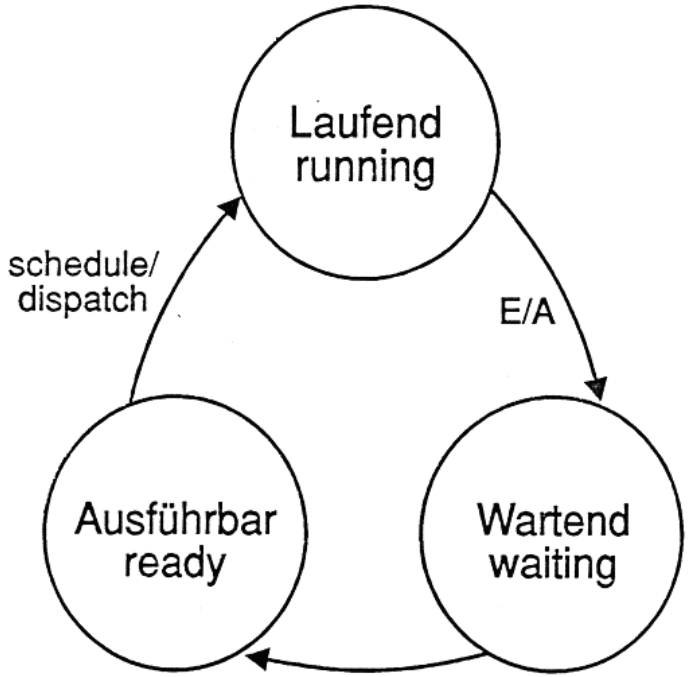
Jeder Prozess rotiert wiederholt durch die Zustände laufend, wartend und ausführbar. Moderne Rechner haben meistens mehr als eine CPU, z.B. 80 bei einem z196 Mainframe. In der Regel verarbeitet jede CPU einen anderen Prozess. In diesem Fall können sich mehrere Prozesse im Zustand „laufend“ befinden.

- Ein Prozess befindet sich im Zustand **laufend**, wenn er von einer der verfügbaren CPUs ausgeführt wird.
- Ein Prozess befindet sich im Zustand **wartend**, wenn er auf den Abschluss einer I/O Operation wartet.
- Ein Prozess befindet sich im Zustand **ausführbar**, wenn er darauf wartet, dass die Scheduler Komponente des Überwachers ihn auf einer frei geworden CPU in den Zustand laufend versetzt.



Jede Linie entspricht einem Prozess. Jeder Prozess wird durch einen Bereich im Hauptspeicher (Task Control Block, TCB) gekennzeichnet und repräsentiert. Die Scheduler Komponente des Überwachers unterhält drei Warteschlangen (Queues) für wartende, ausführbare und laufende Prozesse. In diese Warteschlangen werden die TCBs der Prozesse beim Wechsel des Zustandes eingeordnet.
Angenommen 3 CPUs – 3 Prozesse sind gleichzeitig im Zustand laufend

Die Scheduler Komponente des Überwachers selektiert aus der Warteschlange ausführbarer Prozesse jeweils einen Kandidaten, wenn immer eine CPU frei wird.



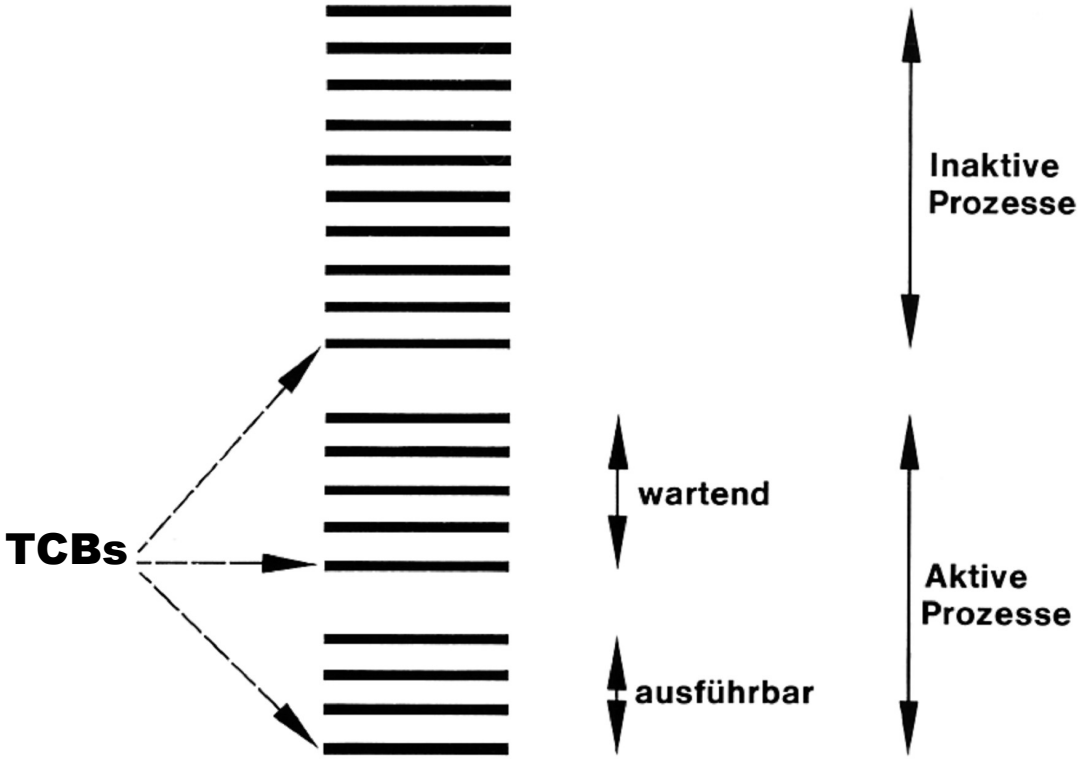
Dies ist eine andere Darstellung der obigen Abbildung. Auf einem Mehrrechnersystem (Rechner mit mehr als einer CPU) können sich mehrere Prozesse im Zustand laufend befinden. Es ist auch möglich, dass ein Prozess mehrere CPUs beschäftigt.

Ausführbare Prozesse verfügen über Ressourcen, besonders über Platz im Hauptspeicher (Programm Code, Daten).

Zeitscheiben Steuerung (Time Slicing)

Bei der Ausführung eines Anwendungsprogramms treten I/O Anforderungen relativ häufig auf. In diesem Fall wird der betroffene Prozess in den Zustand wartend versetzt und die Scheduler Komponente des Überwachers (Supervisor) selektiert aus der Menge (Queue) der ausführbaren Prozesse einen Prozess, der in den Zustand laufend versetzt wird.

In der großen Mehrzahl der Fälle wird dieser Prozess nach einem Zeitraum, der selten 1 ms überschreitet, in den Zustand wartend versetzt, weil er eine I/O Operation ausführt. Es gibt jedoch Situationen, in denen dies nicht der Fall ist. Um zu verhindern, dass ein Prozess eine CPU dauerbeansprucht, enthält der Scheduler eine **Zeitscheiben (Time Slice)** Komponente, die einen laufenden Prozess nach einer gewissen Zeit (typischerweise wenige ms) unterbricht, in den Zustand ausführbar versetzt, und dafür einen anderen ausführbaren Prozess in den Zustand laufend versetzt. Dieser Vorgang wird als „**Preemption**“ (Präemptives Multitasking) bezeichnet.



Task Control Block (TCB) Warteschlangen

Ein Prozess wird durch einen Prozessleitblock beschrieben. Der Prozessleitblock enthält wichtige Funktionen für die Ausführung eines Prozesses.

z/OS verwendet die Bezeichnung Task Control Block (TCB). Andere Architekturen verwenden die Bezeichnung Process Control Block (PCB).

Zu jedem Zeitpunkt beanspruchen viele Prozesse Platz im Hauptspeicher und könnten ausgeführt werden. In einem Rechner mit einer CPU verarbeitet die CPU aber nur einen Prozess. In einem Mehrfachrechner mit n CPUs können gleichzeitig n Prozesse ausgeführt werden.

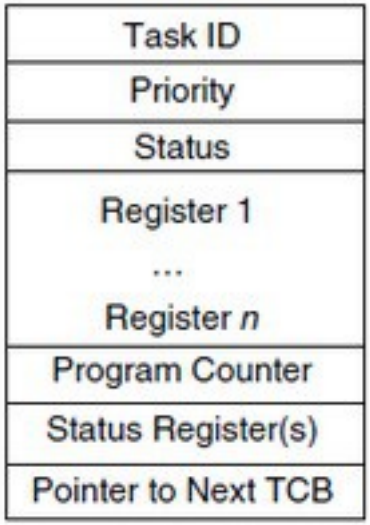
TCBs der wartenden und ausführbaren Prozesse werden vom Scheduler in Warteschlangen eingereiht und verwaltet.

Prozesse können inaktiv sein. Dies bedeutet, ihre TCBs und ihre Seiten sind auf einen Plattenspeicher ausgelagert. Der Scheduler entscheidet, ob und wann ein inaktiver Prozess aktiviert wird.

Inhalt des TCB

Ein Prozess wird durch seinen TCB repräsentiert und dargestellt. Der TCB ist ein Bereich im Hauptspeicher und enthält Informationen über den Prozess.

Mehrzweck Register, Gleitkommaregister usw. sowie das PSW sind in einer CPU nur einmal vorhanden. Wird ein Prozess vom laufenden in den wartenden Zustand versetzt, wird der Inhalt der Register und des PSW in seinem TCB abgespeichert, ebenso alle weitere Information, die den Prozess charakterisiert. Wird der gleiche Prozess später einmal wieder vom ausführbaren in den laufenden Zustand versetzt, wird diese Information benutzt, um den bisherigen Zustand der CPU wieder herzustellen.



Verarbeitungsgrundlagen Teil 3

Virtual Storage

Virtueller Speicher

Die Befehlsadressen und effektiven Adressen der Operanden arbeiten bei einem modernen Rechner mit einer Illusion eines linearen Speichers (virtueller Speicher), der eine andere und einfachere Struktur hat als der reale Hauptspeicher, in dem sich die Befehle und Operanden tatsächlich befinden.

Hierzu wird der virtuelle Speicher in Blöcke aufgeteilt, die alle die gleiche Größe (z.B. 4096 Bytes) haben. Diese Blöcke nennt man **Seiten (pages)**.

Auch der Hauptspeicher wird in Blöcke aufgeteilt, die genauso groß wie die Seiten sind und als Platzhalter für die Aufnahme von Seiten dienen. Diese Blöcke nennt man **Rahmen (frames** oder **pageframes)**.

- Virtuelle Adressen adressieren Maschinenbefehle und Operanden im virtuellen Speicher.
- Reale Adressen adressieren Maschinenbefehle und Operanden im (realen) Hauptspeicher.

Der virtuelle Speicher ist aus der Sicht des Programmierers ein kontinuierlicher, einfach zusammenhängender, linearer Adressenraum.

Die Abbildungsvorschrift für die virtuelle (logische) in die reale (physikalische) Adressumsetzung ist in einer **Seitentabelle** enthalten.

Begriffe

Wir benutzen die folgenden Begriffe:

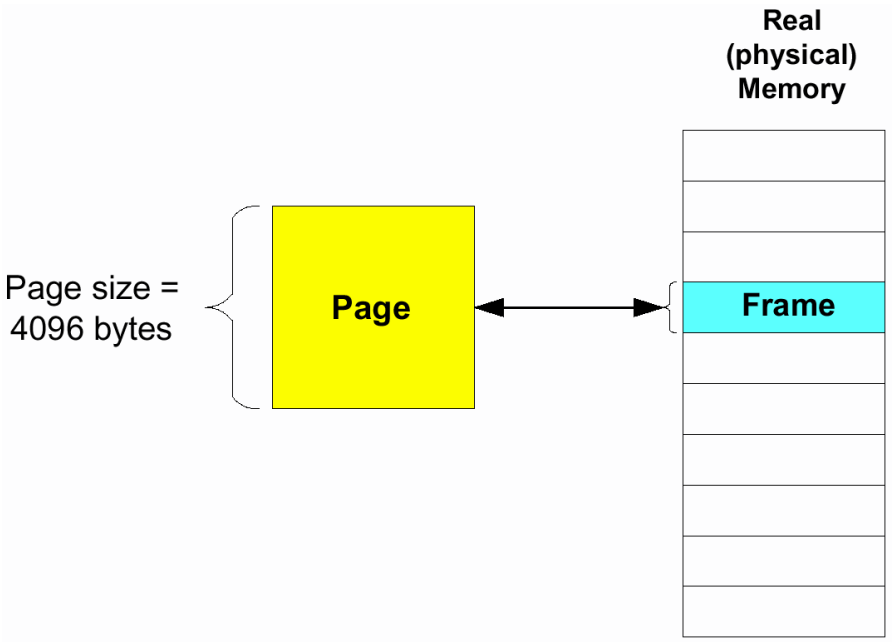
Ein **Adressenraum** (**Address Space**) ist die lineare Folge von Adressen der Bytes (oder anderer adressierbarer Einheiten) eines Speichers.

Ein Virtueller Adressenraum ist die lineare Folge von Adressen der Bytes eines virtuellen Speichers.

Ein Realer Adressenraum ist die lineare Folge von Adressen der Bytes eines realen (tatsächlich existierenden) Hauptspeichers.

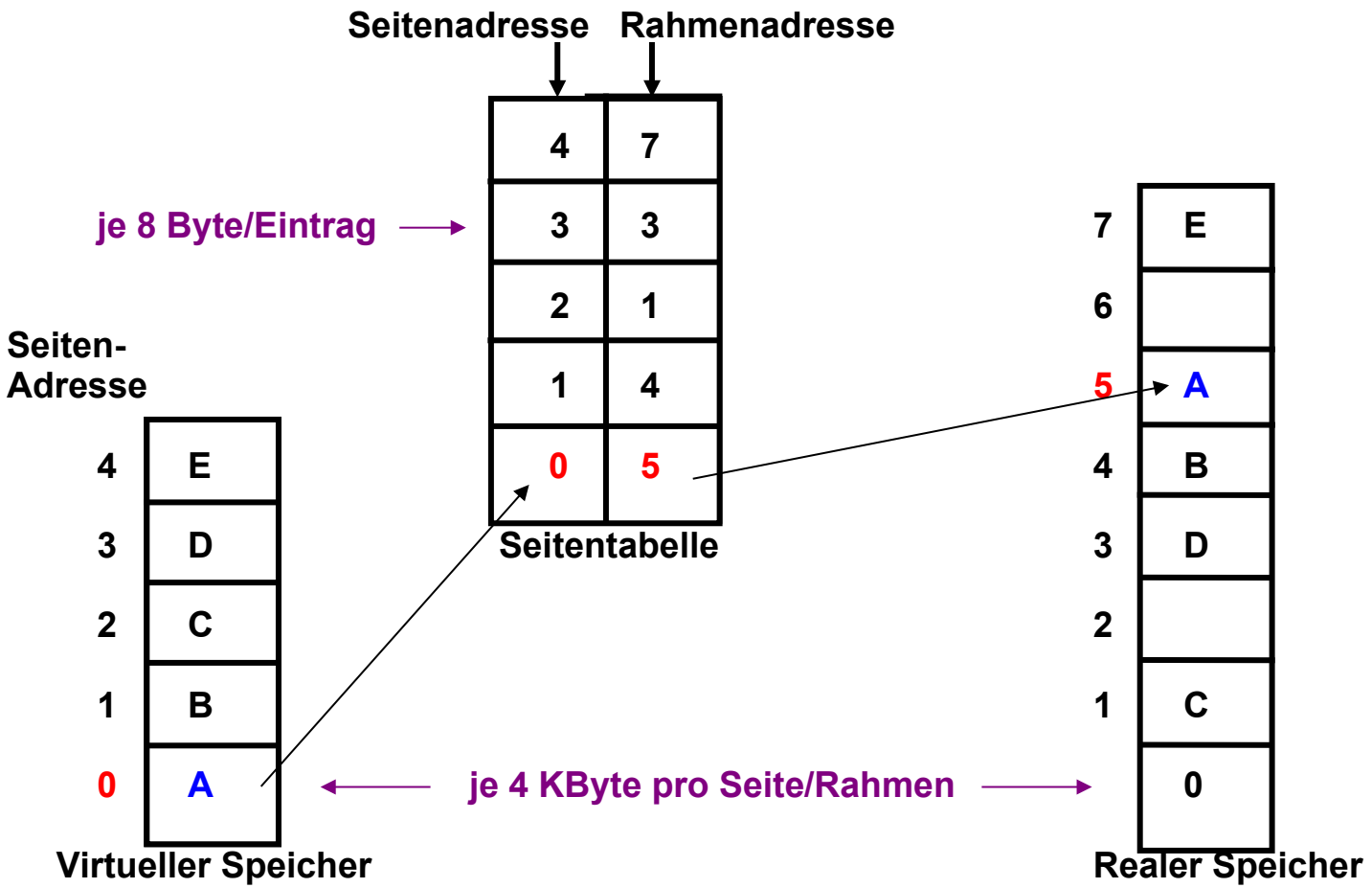
Mittels der Adressumsetzung (Dynamic Address Translation, DAT) wird der virtuelle Adressenraum der Befehle und Operanden eines Benutzerprozesses vom realen Adressenraum des Hauptspeichers getrennt. Die Adressumsetzung bewirkt die Abbildung von virtuellen Adressen auf reale Adressen.

Adressumsetzung



Für die Adressumsetzung werden der virtuelle und der reale Adressenraum jeweils in 4096 Byte große Blöcke aufgeteilt. Die Blöcke des virtuellen Speichers werden als Seiten (Pages) und die Blöcke des realen Speichers als Rahmen (Frames) bezeichnet. Die Anordnung der Bytes innerhalb einer Seite oder eines Rahmens ist identisch und wird bei der Adressumsetzung nicht verändert. Es erfolgt aber eine willkürliche Zuordnung von Seiten- zu Rahmenadressen.

Eine virtuelle bzw. reale Adresse besteht deshalb grundsätzlich aus zwei Feldern: 1. Seiten- bzw. Rahmenadresse und 2. Byteadresse



In dem hier gezeigten Beispiel erstreckt sich der virtuelle Speicher über 5 Seiten, der reale Hauptspeicher über 8 Rahmen. Virtueller und realer Speicher können durchaus unterschiedliche Größen haben.

Die 5 Seiten des virtuellen Speichers speichern die Inhalte A, B, C, D und E. A hat die Seitenadresse 0. Die Adressumsetzung erfolgt mit Hilfe einer Seitentabelle (page table). Jede Seite des virtuellen Speichers hat einen Eintrag in der Seitentabelle. Der Eintrag für Seitenadresse 0 besagt, dass der Seiteninhalt A in dem Rahmen mit der Rahmen-Adresse 5 abgebildet wird.

Größe des virtuellen Speichers richtig wählen

Theoretisch ist es denkbar, dass bei einem Rechner mit 64 Bit Adressen jeder virtuelle Adressenraum eine Gröβe von 2^{64} Byte hat. In der Praxis ist das nicht machbar, unter anderem, weil die Größe der Seitentafel im realen Speicher proportional zu der Größe aller virtuellen Speicher ist.

Seitentabellen erfordern wenige Promille der virtuellen Speichergröße an realem Hauptspeicherplatz. Zwei Promille von 2^{64} Bytes = 16 ExaByte sind 32 TeraByte.

Ein Mainframe Rechner kann 10 000 Prozesse mit getrennten Seitentabellen unterhalten. Die maximale Hauptspeichergröße eines z196 Mainframe beträgt aber „nur“ 3 TByte.

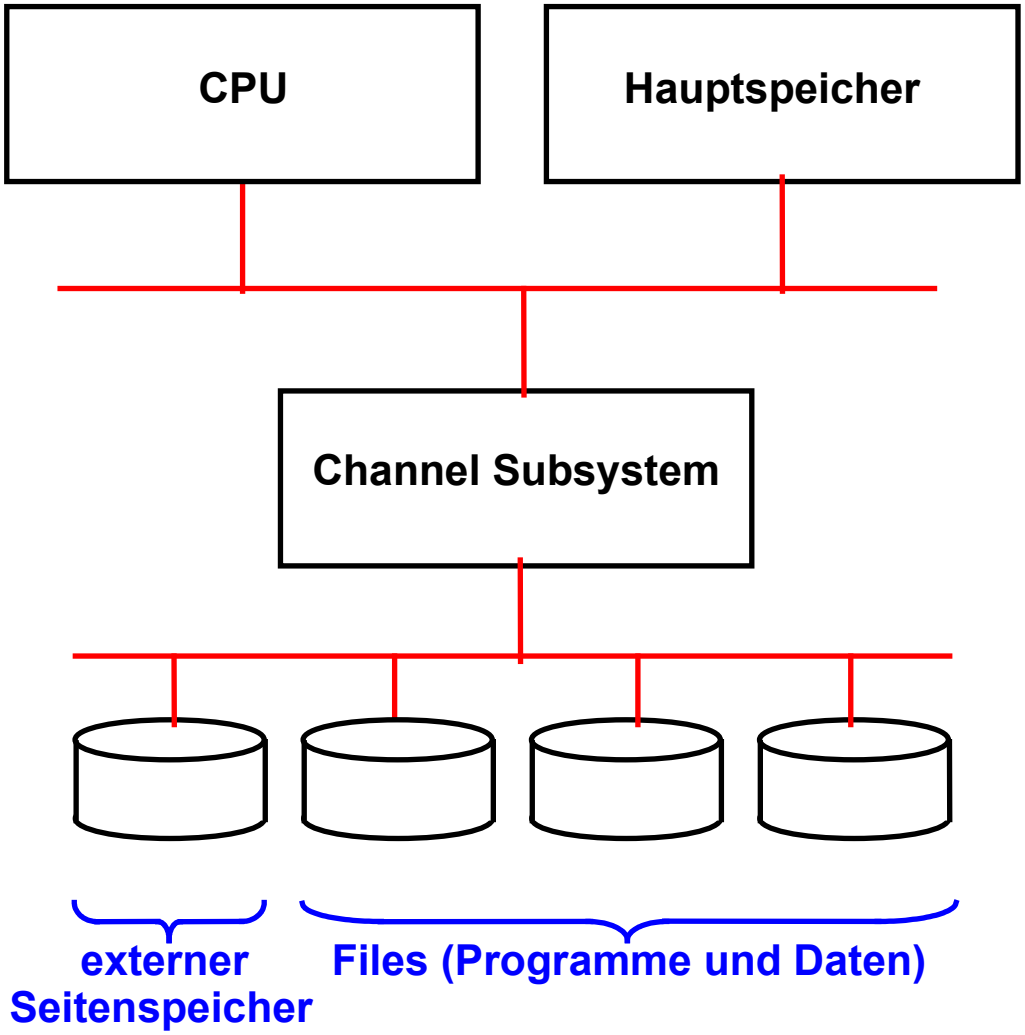
Virtuelle Speicher sollen deshalb nur so groß eingerichtet werden, wie es der Prozess erfordert.

Seitentabellen Hierarchie

Die Seitentabelle befindet sich (entweder teilweise oder ganz) im Überwacherteil des realen Hauptspeichers.

Genau genommen verwendet ein Mainframe (und auch ein x86) Rechner nicht eine einzige Seitentabelle, sondern eine 2 – 5 stufige Hierarchie von Seitentabellen. Im Gegensatz dazu verwendet die PowerPC Architektur nur eine einzige Seitentabelle.

Dies sind Implementierungsdetails, die für ein Verständnis der folgenden Erläuterungen nicht benötigt werden.



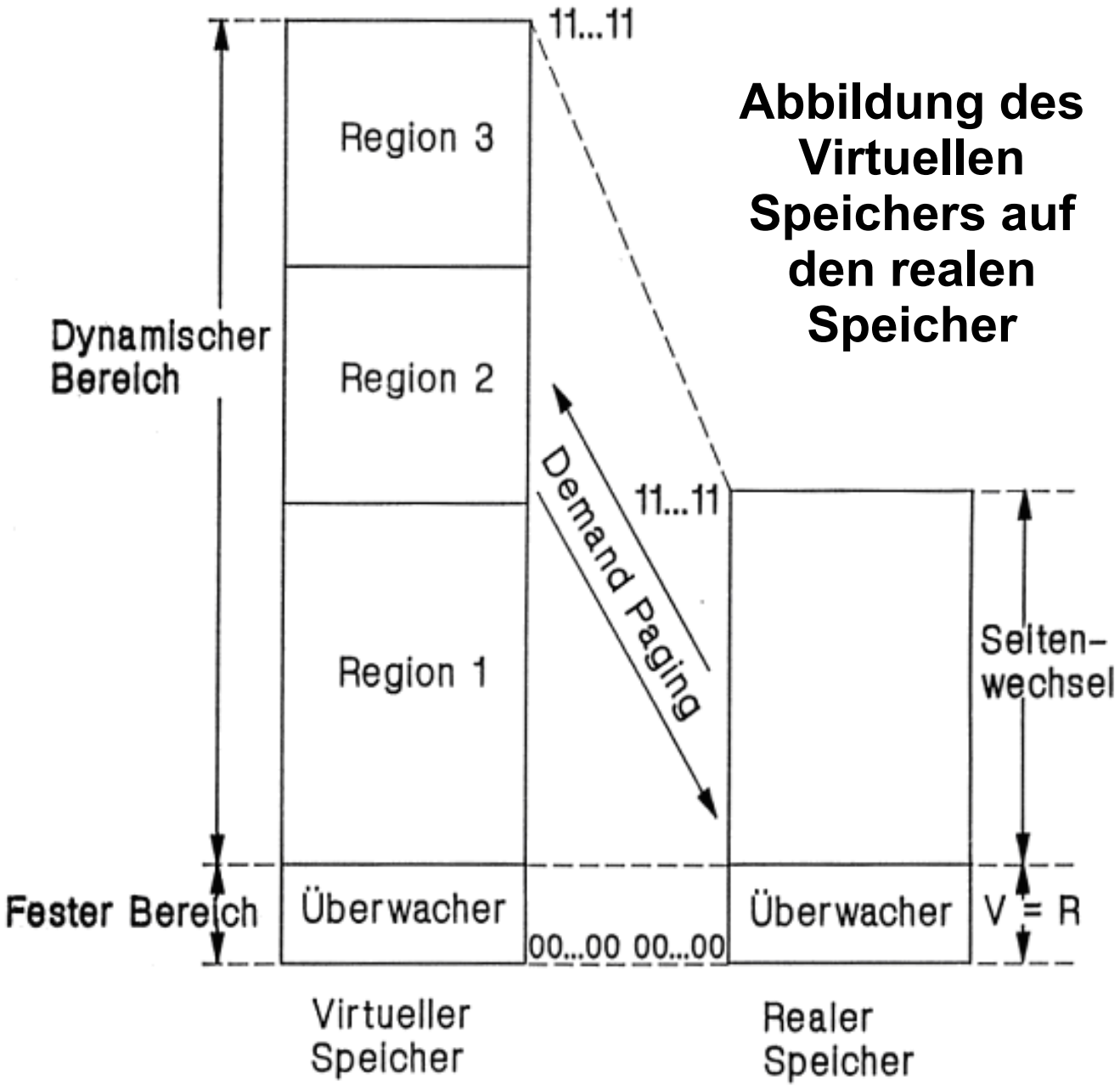
Der virtuelle Speicher kann auch größer als der reale Hauptspeicher sein.

In diesem Fall wird ein Teil des realen Hauptspeichers (Real Storage) auf einen als externer Seitenspeicher (Auxiliary Storage oder Page Data Sets) bezeichneten Plattenspeicher ausgelagert.

Bei einem Windows System ist dies z.B. der Bereich pagefile.sys auf der Partition C:. Bei einem Mainframe ist das in der Regel ein (oder mehrere) nur hierfür benutzter Festplatten-speicher. Der externe Seitenspeicher ist hierfür in 4096 Byte große Rahmen aufgeteilt.

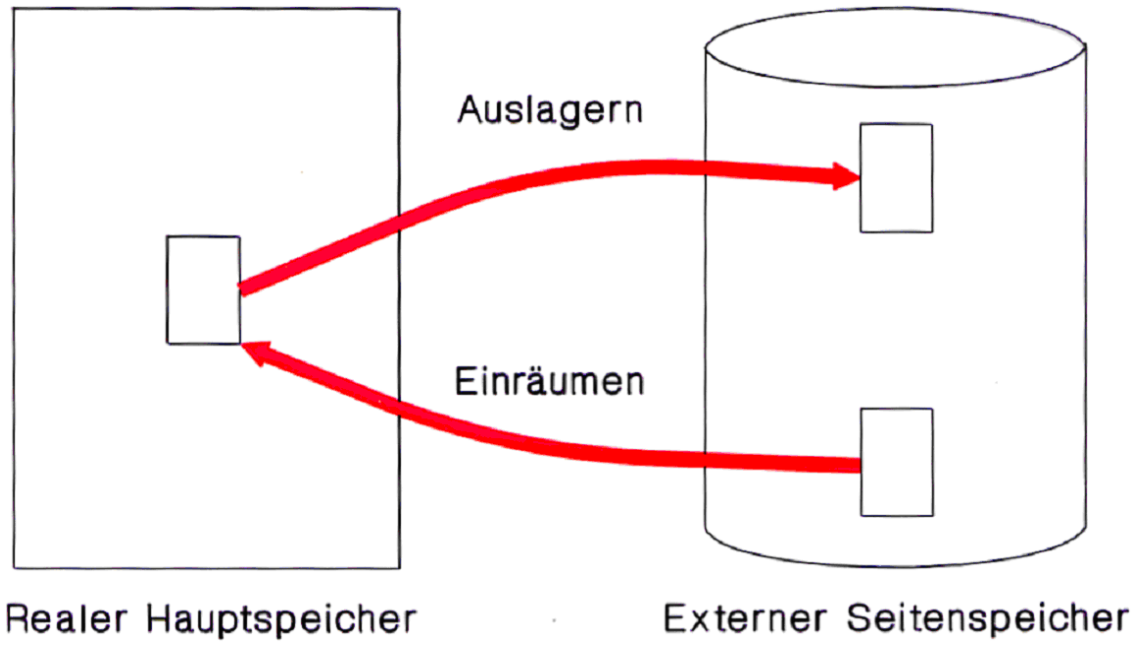
Sehr grob betrachtet hat das Channel Subsystem eines Mainframes die Funktion eines I/O Adapters.

Abbildung des Virtuellen Speichers auf den realen Speicher



Der Benutzer Adressenraum ist in der Regel größer als der reale Hauptspeicher. Ein Teil des Benutzerraums wird deshalb in jedem Augenblick auf den externen Seitenspeicher ausgelagert.

Eine Möglichkeit der Nutzung des virtuellen Speicherkonzeptes besteht darin, den virtuellen Speicher in mehrere Regions aufzuteilen, wobei in jeder Region ein Prozess läuft. Hierbei ist in jedem Augenblick ein Teil der Seiten einer jeden Region im realen Hauptspeicher abgebildet; der Rest befindet sich auf dem externen Seitenspeicher. Der Inhalt der Regionen verändert sich auf Grund des Demand Paging Konzeptes ständig; der Benutzer Adressenraum wird deshalb als der Dynamische Bereich bezeichnet.



Demand Paging

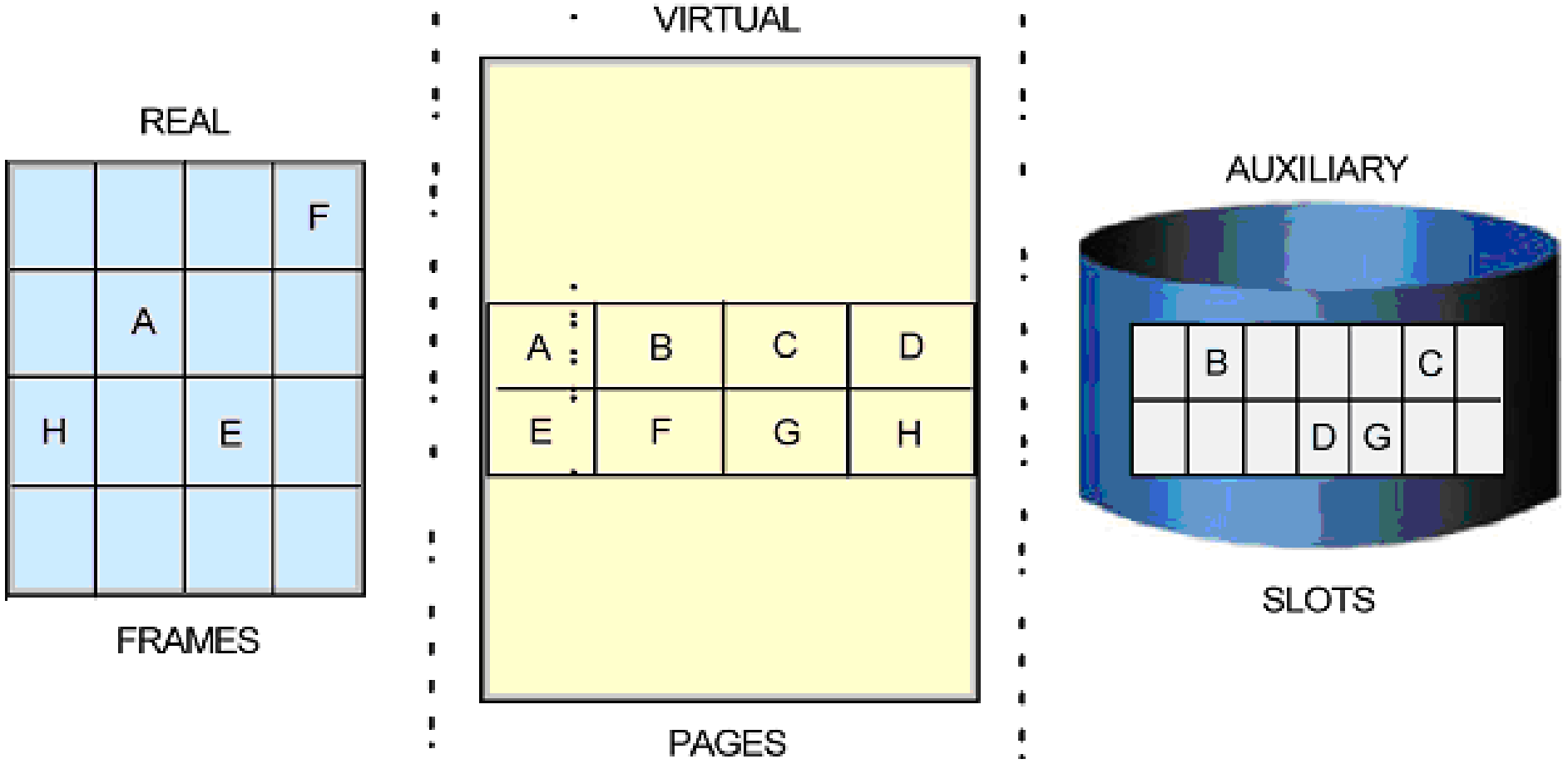
Mehrere Prozesse haben eigene (in der Regel) unabhängige virtuelle Speicher. Jeder virtuelle Speicher kann größer als der reale Hauptspeicher sein.

Der größere Teil der Seiten eines virtuellen Speichers ist in jedem Augenblick auf einem "externen Seitenspeicher ", typischerweise ein Plattenspeicher, ausgelagert. Der reale Speicher besteht somit aus 2 Teilen: dem realen Hauptspeicher und dem externen Seitenspeicher.

Beim Zugriff zu einer ausgelagerten Seite (nicht in einem Rahmen des Hauptspeichers abgebildet) erfolgt eine "**Fehlseitenunterbrechung** " (**Page Fault**).

Diese bewirkt den Aufruf einer Komponente des Überwachers (Seitenüberwacher, Paging Supervisor), der die benötigte Seite aus dem externen Seitenspeicher holt und in den Hauptspeicher einliest. Evtl. muss dafür Platz geschaffen werden, indem eine andere Seite dafür auf den externen Seitenspeicher gelegt wird.

Dieser Vorgang wird als „**Demand Paging**" bezeichnet.



In dem hier gezeigten Beispiel enthält der virtuelle Speicher die Seiten mit den Inhalten A, B, C, D, E, F, G und H.
A, E, F und H sind in Rahmen des realen Hauptspeichers (Real Storage) abgespeichert. B, C, D und G befinden sich auf dem externen Seitenspeicher (Auxiliary Storage). z/OS bezeichnet die Rahmen des externen Seitenspeichers gelegentlich auch als „Slots“.

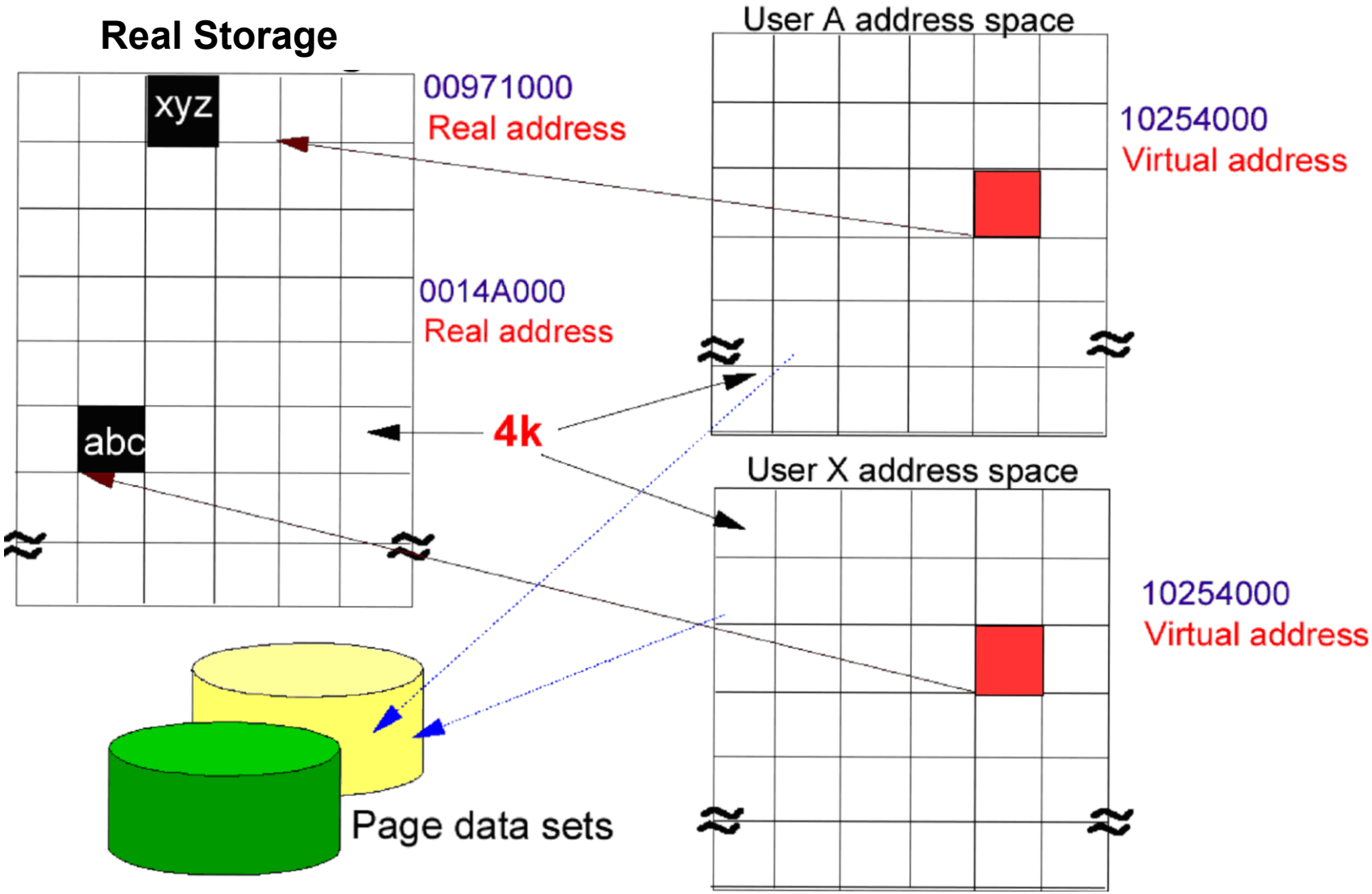
Wenn ein Benutzerprogramm auf B, C, D oder G zugreifen will, muss eine Komponente des Überwachers, der „Seitenüberwacher“ (Paging Supervisor), den Rahmen zuerst vom externen Seitenspeicher in den Hauptspeicher kopieren. Vermutlich muss er, um Platz zu schaffen, vorher den Inhalt eines anderen Rahmen des Hauptspeichers auf den externen Seitenspeicher auslagern.

Abbildung des Virtuellen Speichers auf den realen Speicher

Der Inhalt der Rahmen des realen Hauptspeichers ändert sich ständig, da mittels des Demand Paging ständig Seiten zwischen dem Hauptspeicher und dem externen Seitenspeicher hin- und herbewegt werden.

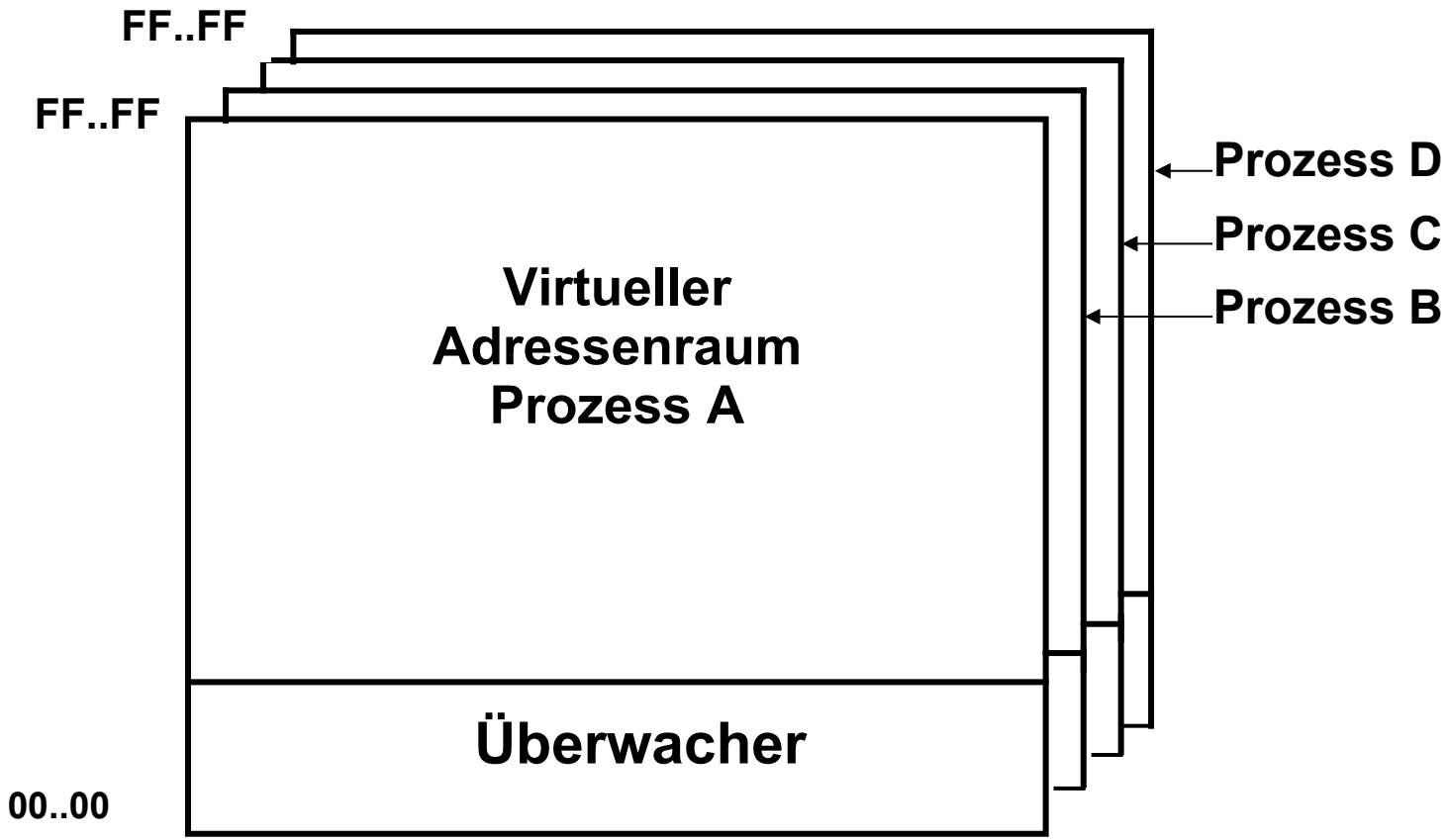
Hierzu gibt es eine Ausnahme: Seiten, die den Überwacher enthalten, werden aus Performance Gründen ständig im Hauptspeicher gehalten. Weiterhin sind die virtuellen Adressen des Überwachers identisch mit den realen Adressen. Man bezeichnet den Überwacher deshalb auch als den Residenten Überwacher. Der Überwacher belegt die untersten Adressen, beginnend mit der hexadezimalen Adresse 000…000.

Streng genommen ist diese Darstellung stark vereinfacht. Tiefer gehende Details werden aber für die folgenden Erläuterungen nicht benötigt.

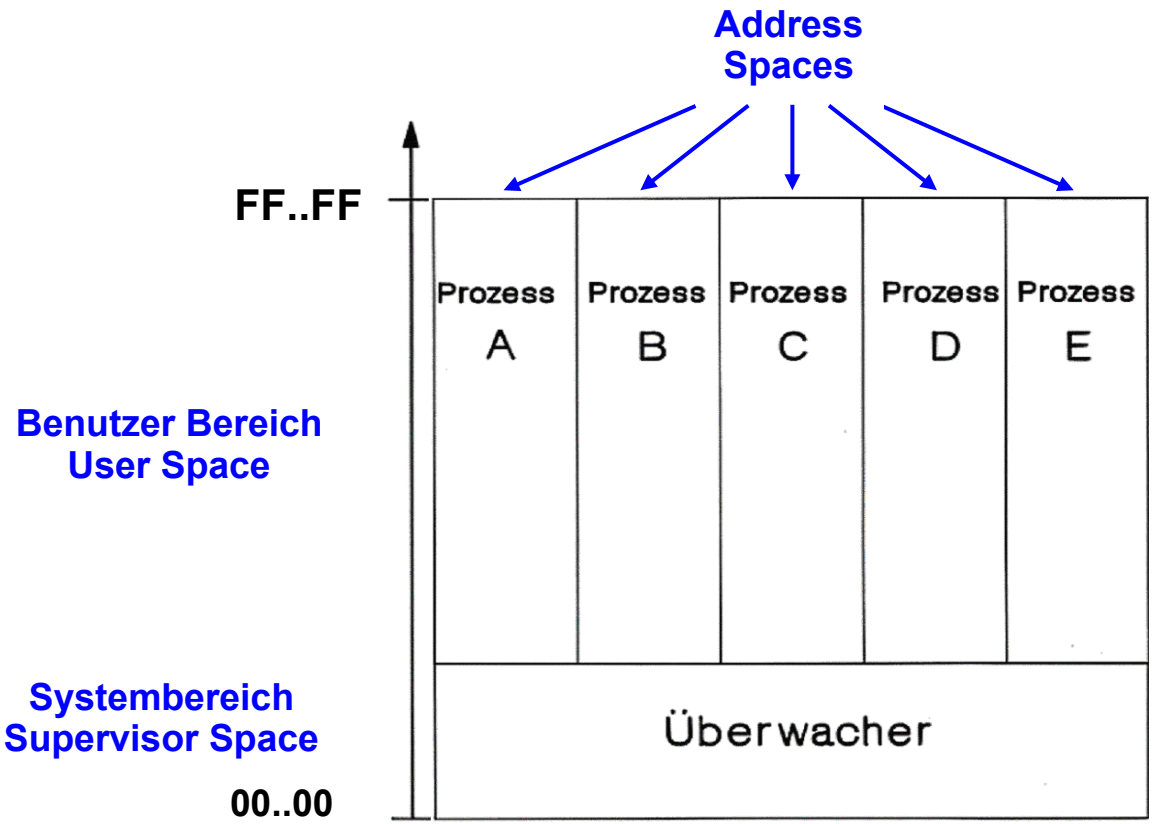


Gezeigt ist, wie zwei Seiten in getrennten virtuellen Adressräumen, aber mit identischen virtuellen Adressen, auf unterschiedliche reale Rahmenadressen abgebildet werden. Seiten werden während der Prozessausführung mehrfach auf den Externen Seitenspeicher ausgelagert und später wieder in den Hauptspeicher eingeräumt. Hierfür werden vermutlich andere Rahmen benutzt; die reale Hauptspeicheradresse ändert sich zwischen Auslagern und Einräumen.

Mehrfache virtuelle Adressenräume



Da der virtuelle Speicher sehr viel größer als der reale Hauptspeicher sein kann, ist es möglich, jedem Prozess einen virtuellen Speicher maximaler Größe zuzuordnen. In diesem Fall benutzen unterschiedliche virtuelle Speicher identische Adressen. Der Adressenbereich der virtuellen Speicher geht von Hex 000…000 bis zu einer maximalen Größe, theoretisch bis zu FFF...FFF, z.B. 2^{31} oder 2^{64} Bytes. Da der reale Hauptspeicher viel kleiner ist, wird ein Großteil der Seiten auf dem externen Seitenspeicher abgespeichert.



Die Abbildung der virtuellen Adressen auf unterschiedliche reale Adressen erfolgt, in dem man jedem Prozess eine eigene Seitentabelle zuordnet.

Der Systembereich (Supervisor Space) mit dem Überwacher ist nur einmal vorhanden, und ist Bestandteil aller virtuellen Adressenräume. Für den Systembereich sind virtuelle und reale Adressen identisch.

z/OS benutzt dieses Verfahren. Der virtuelle Adressenraum eines Prozesses wird als (virtual) Address Space bezeichnet.

Verarbeitungsgrundlagen Teil 4

Betriebssystem Überwacher / Supervisor

Überwacherstatus vs. Problemstatus

Eine CPU läuft in jedem Augenblick entweder im Überwacherstatus (supervisor state) oder im Benutzerstatus (user state).

Der Überwacherstatus bzw. Benutzerstatus wird durch 1 Bit im Flag Bit Register Teil des Programm Status Wortes der Zentraleinheit definiert.

Der Überwacher (Kernel) läuft im **Überwacherstatus (Supervisor State, Kernel State)**.

Benutzerprogramme (Anwendungsprogramme) laufen im **Problemstatus (Problem State, User State)**.

Auswirkungen sind:

- Bestimmte ***privilegierte*** Maschinenbefehle können nur im Überwacherstatus ausgeführt werden
- ***Speicherschutz*** – Im Problemstatus kann nur auf einen Teil des virtuellen und realen Hauptspeicher zugegriffen werden

Unterbrechungen

Unterbrechungen (Interruptions) sind ein zentrales Steuerelement in einem jeden Rechner. Unterbrechungen bewirken eine Änderung des Status der Zentraleinheit als Folge von Bedingungen (Ursachen), die entweder

- außerhalb der Zentraleinheit (CPU), oder
- innerhalb der Zentraleinheit

auftreten.

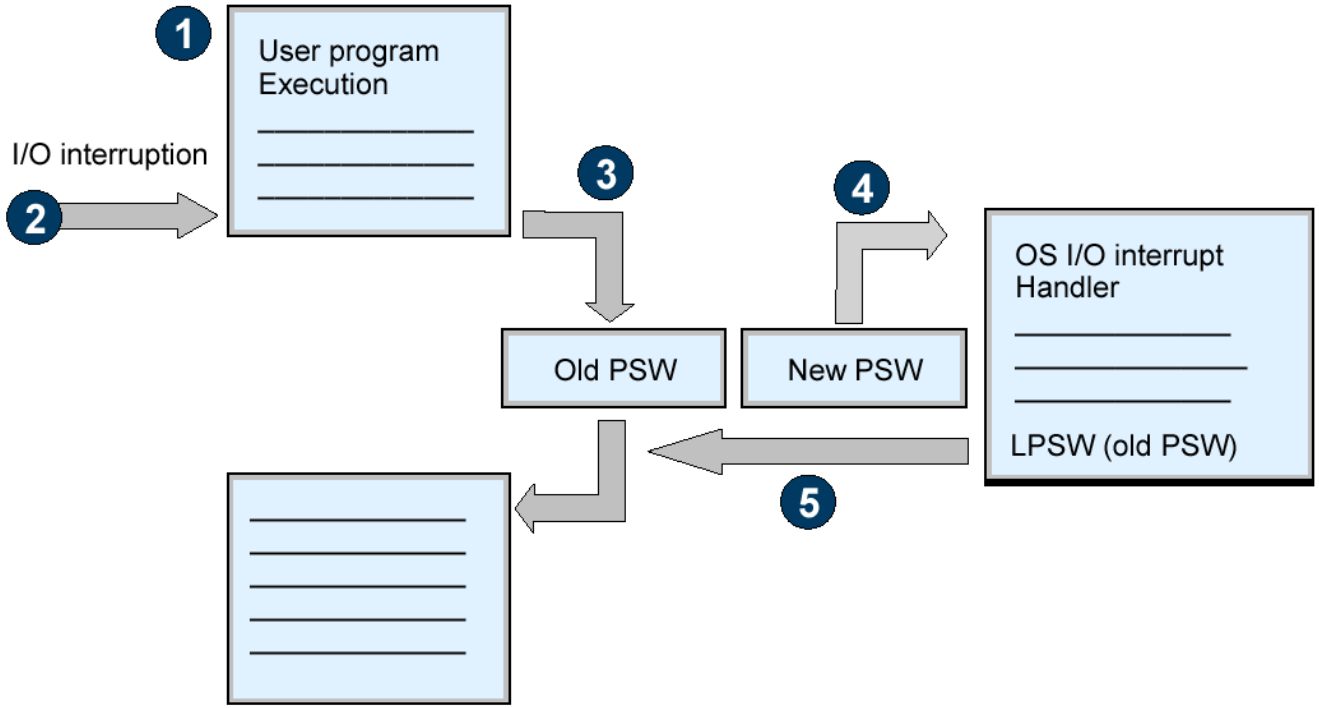
Unterbrechungen bewirken den Aufruf und die Ausführung von speziellen Programmen (Unterbrechungsroutinen) außerhalb des normalen Programmablaufs.

Eine Unterbrechung bewirkt:

1. Abspeichern des PSW (Programm-Status-Wort, Flag Bits und Befehlszählers) im Hauptspeicher.
2. Laden des Befehlszählers mit der Anfangsadresse einer Unterbrechungsroutine (Interrupt Handler).
3. Status-Initialisierung im PSW (z.B. Überwacherstatus setzen).
4. In der Regel: Abspeichern der Mehrzweckregister, evtl. auch Steuerregister, Gleitkommaregister durch die Unterbrechungsroutine.

Typischerweise enthält ein Feld im Hauptspeicher zusätzliche Informationen über die Ursache der Unterbrechung.

Interruption Process Flow



Eine Unterbrechung bewirkt das Abspeichern des derzeitig gültigen PSW (als old PSW bezeichnet) im Hauptspeicher an einer hierfür vorgesehenen Adresse und ein Laden eines neuen PSW (new PSW) in das PSW Register. Der Befehlszählerteil des neuen PSW enthält die Adresse des ersten Befehls der Unterbrechungsroutine. Die CPU befindet sich automatisch im Überwacherstatus.

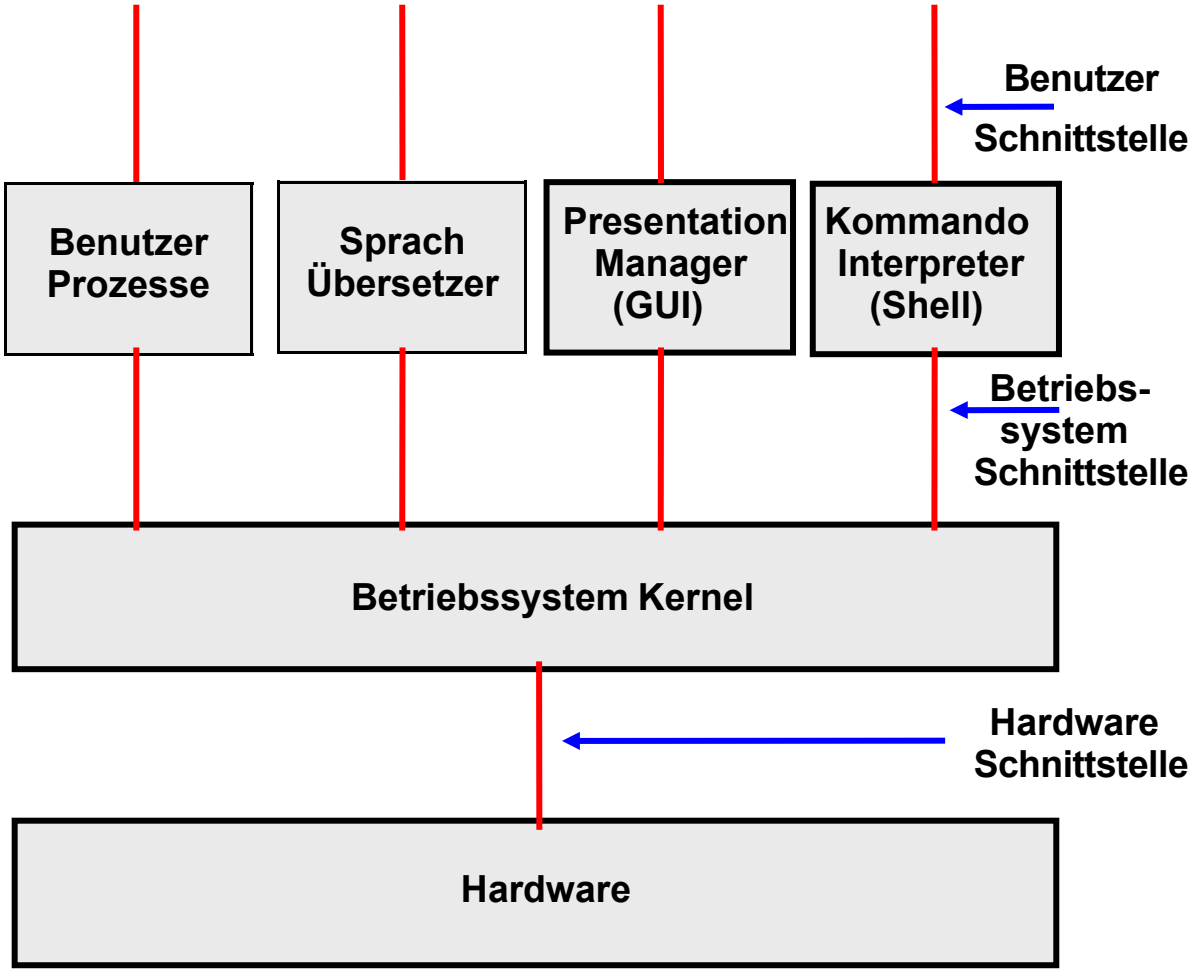
Als allerletzten Befehl führt die Unterbrechungsroutine den Maschinenbefehl „LPSW“ (Load Programm Status) aus. Dieser bewirkt, dass das alte PSW in das PSW Register geladen wird. Der Befehlszählerteil des PSW enthält die Adresse des Maschinenbefehls, den die CPU ohne Eintreten der Unterbrechung als nächstes ausgeführt hätte.

Unterbrechungsklassen

Unterschiedliche Bedingungen können unterschiedliche Klassen von Unterbrechungen auslösen. Es existieren sechs Unterbrechungsklassen mit mehreren Unterbrechungsarten:

Maschinenfehler	Beispiel Paritäts-Fehler (Bus, Hauptspeicher), fehlerhafte Addition
Reset	Beispiel: Setzt Zentraleinheit (und System) in ursprünglichen Zustand
I/O	Beispiel: Signalisiert den Abschluss einer E/A Operation
Extern	Beispiel: System externes Signal, z. B. Ablauf des Zeitgebers.
Programm	Beispiele: Division durch Null, Fehlseitenunterbrechung, illegaler OP Code, illegale Adresse
Systemaufruf	(SVC) Aufruf des Überwachers im Benutzerprogramm (eigentlich ein Maschinenbefehl)

Schichtenmodell der Rechnerarchitektur



Der z/OS Kernel (Supervisor, Überwacher) wird auch als „Basic Control Program“ oder BCP bezeichnet.

Das Betriebssystem besteht aus einer zentralen Steuerungs- und Verwaltungsfunktion, dem Überwacher (Supervisor, Kernel), und weiteren Systemprogrammen oder Komponenten, die unter z/OS als „Subsysteme“ bezeichnet werden. Beispiele für Subsysteme sind der Kommandointerpreter (TSO), das Job Entry Subsystem(JES) oder das DB2 Datenbanksystem. Subsysteme sind Systemprozesse, die parallel zu den Benutzerprozessen laufen. Sie können (müssen aber nicht) permanent Platz im Hauptspeicher belegen.

Überwacher

Der **Überwacher (Supervisor)** besteht aus:

1. Datenbereichen (Control Blocks)

2. Programmteilen, welche Controlblock Daten manipulieren

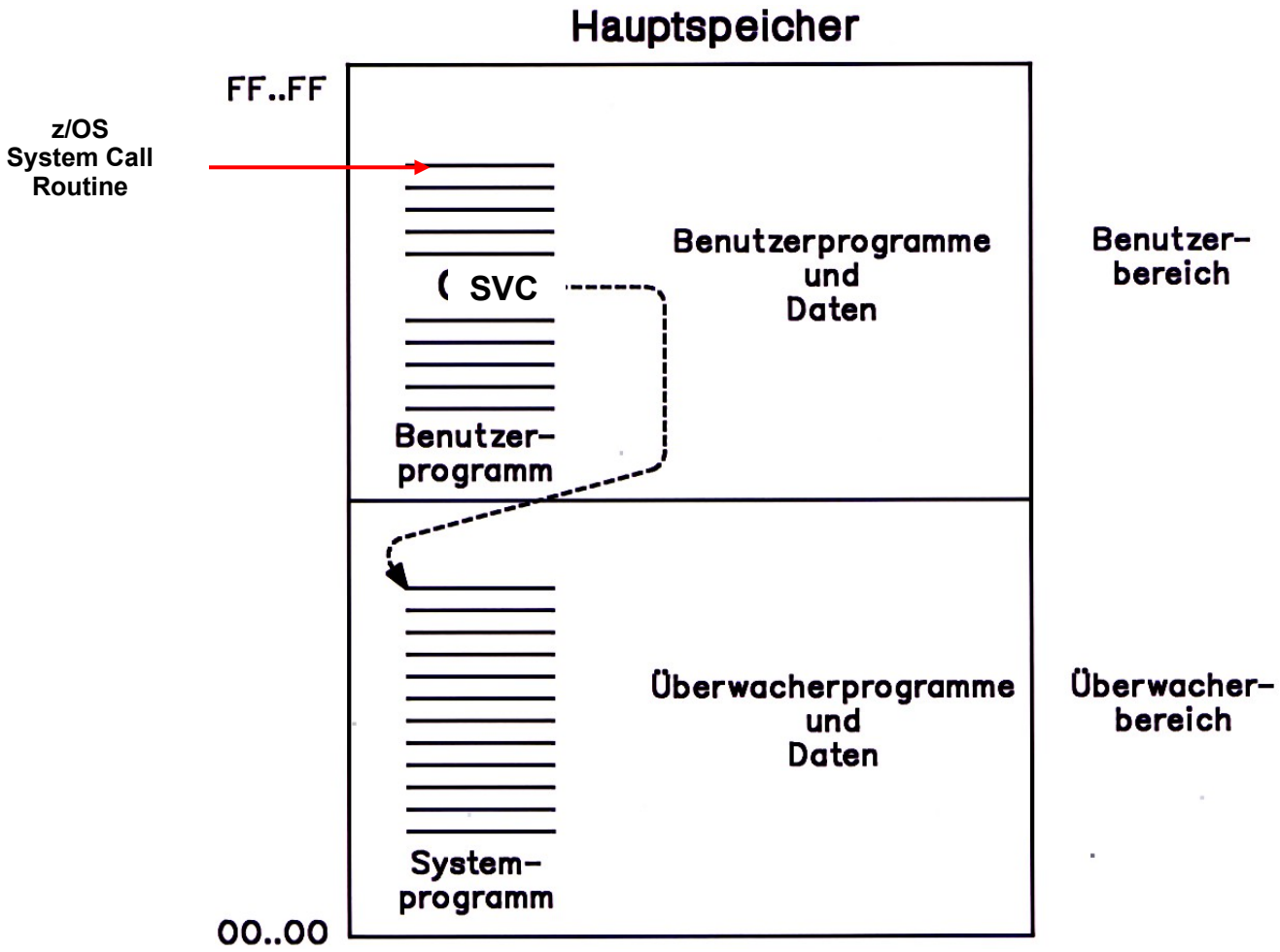
Der Überwacher ist in der Regel nicht strukturiert. In seinem Buch “Modern Operating Systems” published in 1992, pp.18, bezeichnet Prof. Andrew S. Tanenbaum den Überwacher eines Betriebssystems als „The Big Mess“.

Der Überwacher enthält Funktionen, die von vielen Prozessen gemeinsam genutzt werden. Häufig verbringt ein Prozess 50% der Ausführungszeit (Pfadlänge) mit der Ausführung von Überwacherfunktionen (läuft 50% der Zeit im Überwacherstatus).

Die Hardware des Rechners reagiert auf die Eingabe von Maschinenbefehlen und auf Unterbrechungen. Der Überwacher kann nur über Unterbrechungen aufgerufen werden. Er läuft im Überwacherstatus.

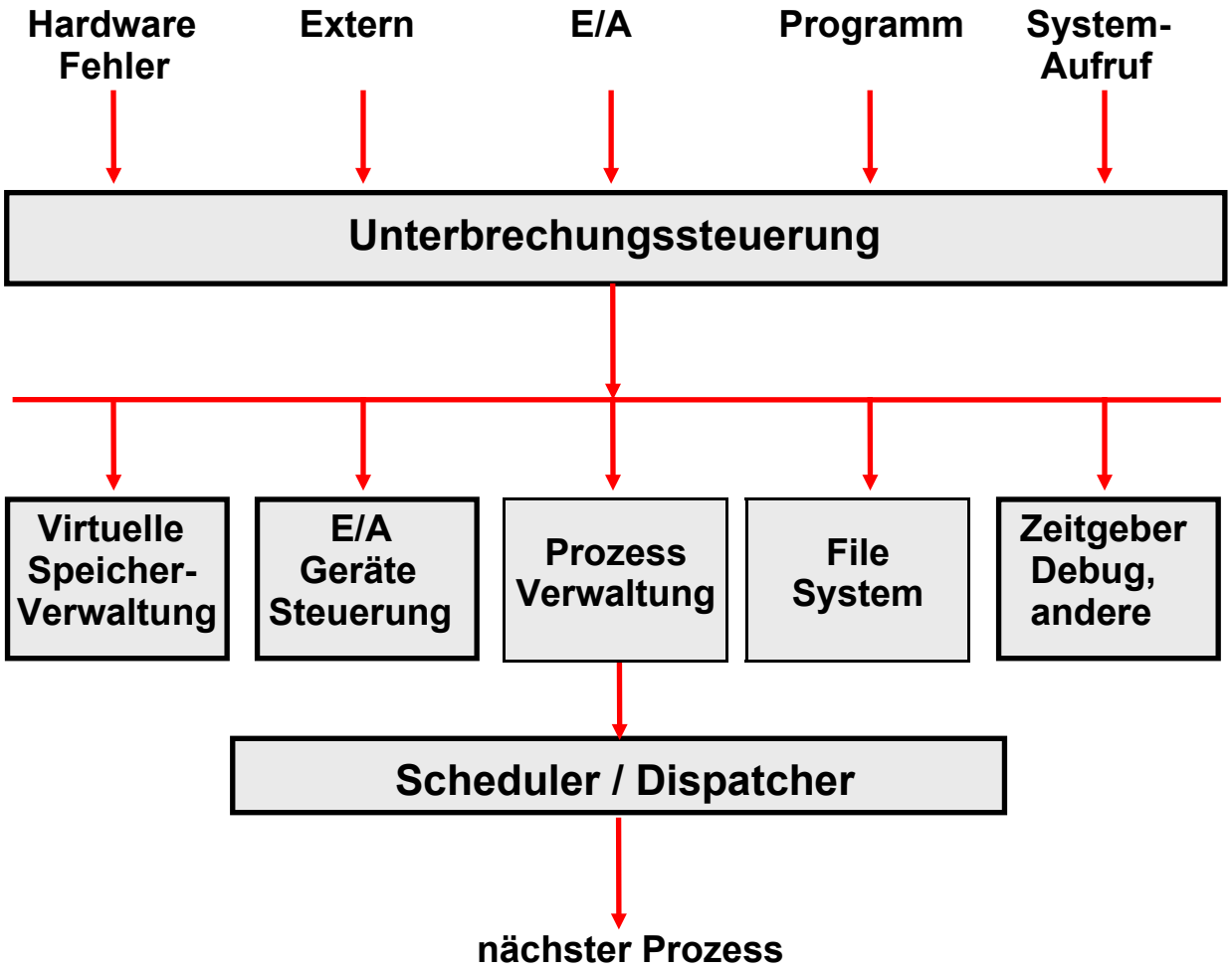
Systemaufrufe (System Calls) sind die einzige Möglichkeit für Benutzerprozesse, mit dem Überwacher zu kommunizieren. Unter z/OS implementiert der SVC Maschinenbefehl die System Call Funktion. Beim x86 hat der INT Maschinenbefehl die gleiche Funktion. Ein System Call ist eine Routine, die im Benutzer Status läuft, u.a. den SVC (oder INT) Maschinenbefehl ausführt, und dadurch den Übergang vom Benutzerstatus in den Überwacherstatus herbeiführt.

Systemaufruf (System Call)



Ein System z Benutzerprogramm kann eine Funktion des Überwachers durch Ausführung des SVC (Supervisor Call) Maschinenbefehls aufrufen. Der SVC Befehl übergibt hierzu einen Parameter, welcher die Art der gewünschten Funktion angibt. Ein Beispiel ist die Durchführung einer WRITE Operation, welche Daten auf den Plattenspeicher schreibt. Die Ausführung des SVC Maschinenbefehls bewirkt eine Unterbrechung. Die Unterbrechungsroutine bewirkt unter anderem den Wechsel von Benutzerstatus in den Kernel Status.

Struktur des Supervisors



Der Aufruf des Überwachers (Supervisor, Kernel) erfolgt grundsätzlich über Unterbrechungen.

Je nach Art der Unterbrechung werden unterschiedliche Komponenten des Überwachers aufgerufen.

Benutzerprozesse nehmen Dienste des Überwachers über eine architekturierte Schnittstelle, den Systemaufruf (System Call, Supervisor Call, SVC) in Anspruch. Diese Begriffe haben alle die gleiche Bedeutung.

Der Scheduler sucht den nächsten auszuführenden Prozess aus.

Struktur des Überwachers

Der Überwacher wird über eine Unterbrechung aufgerufen. Dies ist der einzige Weg, über den man eine Funktion des Überwachers in Anspruch nehmen kann.

Die wichtigsten Überwacherfunktionen sind:

- Die **Unterbrechungssteuerung** untersucht die Art der aufgetretenen Unterbrechung und ruft je nach Art eine andere Komponente des Überwachers auf.
- Die **virtuelle Speicherverwaltung** ordnet virtuellen und realen Speicherplatz zu. Sie bestimmt, welche Seiten in Rahmen des Hauptspeichers abgebildet werden und welche Seiten auf den externen Seitenspeicher (Auxiliary Store) ausgelagert werden. Sie lädt bei Bedarf Seiten vom externen Seitenspeicher in den Hauptspeicher.
- Die **Input/Output Steuerung (I/O Supervisor)** wird aufgerufen, wenn ein Benutzerprogramm eine Lese- oder Schreiboperation auf ein I/O Gerät (z.B. Plattenspeicher) durchführen will. Sie nimmt auch eine I/O Unterbrechung entgegen, die z.B. besagt, dass ein Plattenspeicher eine I/O Operation erfolgreich abgeschlossen hat.
- Die **Prozessverwaltung** aktiviert und deaktiviert Prozesse. Aktive Prozesse verfügen über Ressourcen im Hauptspeicher. Bei deaktivierten Prozessen sind alle Resourcen (vermutlich) auf einen Plattenspeicher ausgelagert.
- Datenstrukturen auf Plattenspeichern werden mit Hilfe des **File Systems** verwaltet. Windows verwendet die NTFS und FAT32 File Systems. VSAM, PDS und zFS sind die wichtigsten z/OS File Systems.
- Der **Scheduler** verwaltet die Warteschlangen der TCBs. Er versetzt Prozesse in den Zustand wartend, ausführbar oder laufend. Er steuert Prioritäten, nach denen entschieden wird, welcher Prozess vom Zustand wartend in den Zustand ausführbar überführt wird.
- Zahlreiche weitere Funktionen wie Zeitscheibensteuerung und Funktionen zum Debuggen von Software sind vorhanden.

Scheduler / Dispatcher

Der Scheduler/Dispatcher ist die Komponente des Überwachers, welcher drei Warteschlangen für die laufenden, wartenden und ausführbaren TCBs verwaltet. Er überführt bei gegebenem Anlass einen TCB von einer Warteschlange in eine andere.

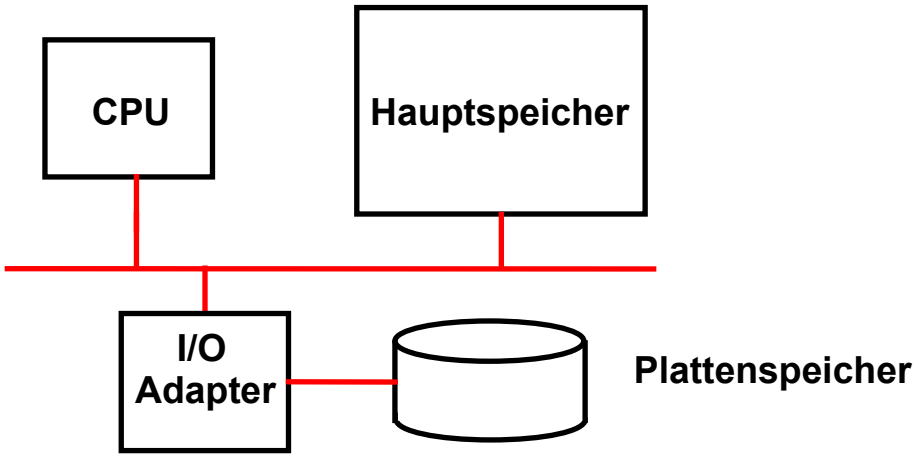
Wenn eine CPU frei wird, selektiert der Scheduler/Dispatcher aus der Warteschlange der ausführbaren TCBs einen Prozess und versetzt ihn in den Zustand laufend. Dies kann über Prioritäten gesteuert werden, und der z/OS Überwacher verfügt hierzu über eine Prioritätssteuerung.

In der Mehrzahl der Fälle bleibt ein Prozess nur kurze Zeit im Zustand laufend. Um zu verhindern, dass ein bestimmter Prozess eine CPU zu lange in Anspruch nimmt, verfügt der Scheduler/Dispatcher über eine Zeitscheibensteuerung. Eine Zeitscheibe ist ein Zeitintervall von typischerweise wenigen Millisekunden. Verbleibt ein Prozess im Zustand laufend über seine Zeitscheibenlänge hinaus, erfolgt eine Unterbrechung durch einen Zeitgeber. Diese bewirkt, dass der laufende Prozess in den Zustand ausführbar versetzt wird, und ein anderer Prozess dafür in den Zustand laufend versetzt wird.

In der Regel laufen unterschiedliche Prozesse auf den CPUs eines Rechners. Es ist jedoch möglich, dass ein Prozess mehr als eine CPU in Anspruch nimmt.

Verarbeitungsgrundlagen Teil 5

Cache



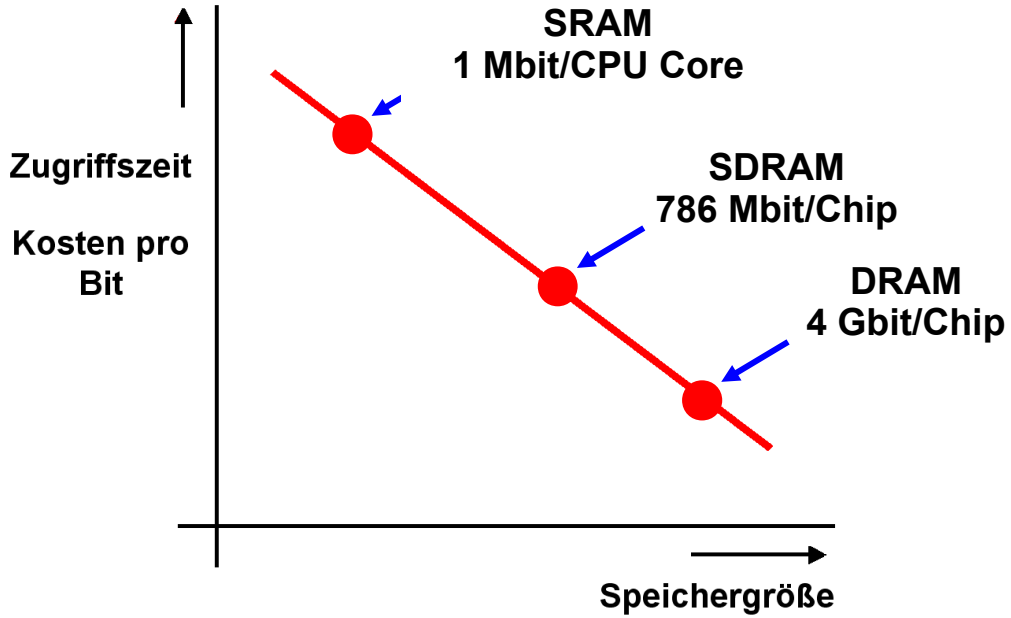
Eine moderne CPU mit einem GHz Prozessor führt größenordnungsmäßig eine Milliarde (10^9) Maschinenbefehle pro Sekunde aus, d.h. eine Instruktion dauert ca. 1 Nanosekunde (10^{-9} s).

Die Zugriffszeit direkt zum Hauptspeicher beträgt ca. 10 - 100 Nanosekunden (10^{-8} s / 10^{-7} s).

Die Zugriffszeit zum Plattenspeicher beträgt größenordnungsmäßig 10 Millisekunden (10^{-2} s).

Diese Unterschiede in der Zugriffszeit haben einen sehr großen Einfluss auf die Struktur moderner Prozessor und Betriebssysteme.

Halbleiter Speicher Chips

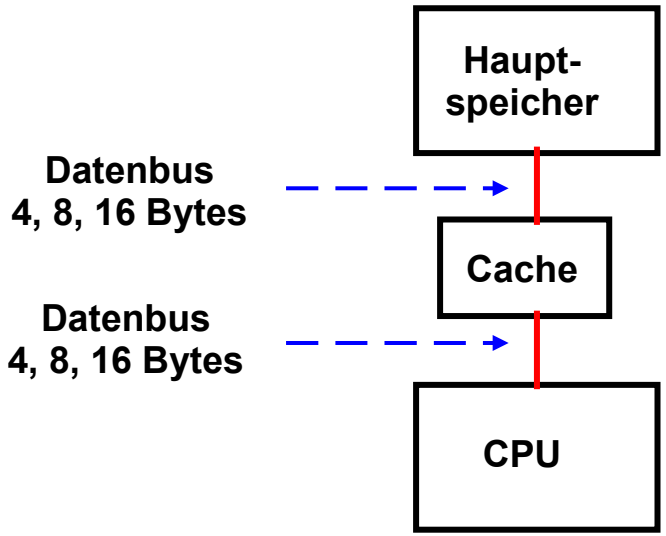


Es existieren eine ganze Reihe unterschiedlicher Technologien, mit denen man einen Speicher auf einem Halbleiterchip realisieren kann. Die Technologien unterscheiden sich in der Zugriffszeit auf den Speicher sowie der Speicherdichte, der Anzahl der Bits, die man pro mm^2 unterbringen kann. Die Speicherdichte beeinflusst die Kosten pro Bit. Allgemein gilt der hier dargestellte Zusammenhang: Kleine Speicher (Schnellspeicher) haben eine schnelle Zugriffszeit, brauchen viel Platz pro Bit und haben hohe Kosten pro Bit. Große Speicher haben eine längere Zugriffszeit, brauchen wenig Platz pro Bit und haben niedrige Kosten pro Bit.

Hauptspeicher Chips verwenden die DRAM (Dynamic Random Access Memory) Technologie. Das speichernde Element ist dabei ein Kondensator, der entweder geladen oder entladen ist. Über einen Schalttransistor wird er zugänglich und entweder ausgelesen oder mit neuem Inhalt beschrieben. Der Speicherinhalt ist flüchtig (volatil), das heißt, die gespeicherte Information geht bei fehlender Stromversorgung oder zu später Wiederauffrischung verloren.
Plattenspeicher sind dagegen statisch. Die gespeicherten Daten bleiben auch im abgeschalteten Zustand erhalten.

Die Adressleitungen eines DRAMs sind üblicherweise **gemultiplext**, d.h., es sind nur etwa halb so viele physisch vorhanden wie insgesamt benötigt werden. (Dagegen wird bei **SRAMs** zwecks höherer Geschwindigkeit meist der komplette Adressbus an Pins geführt, so dass der Zugriff in einer einzigen Operation erfolgen kann.)

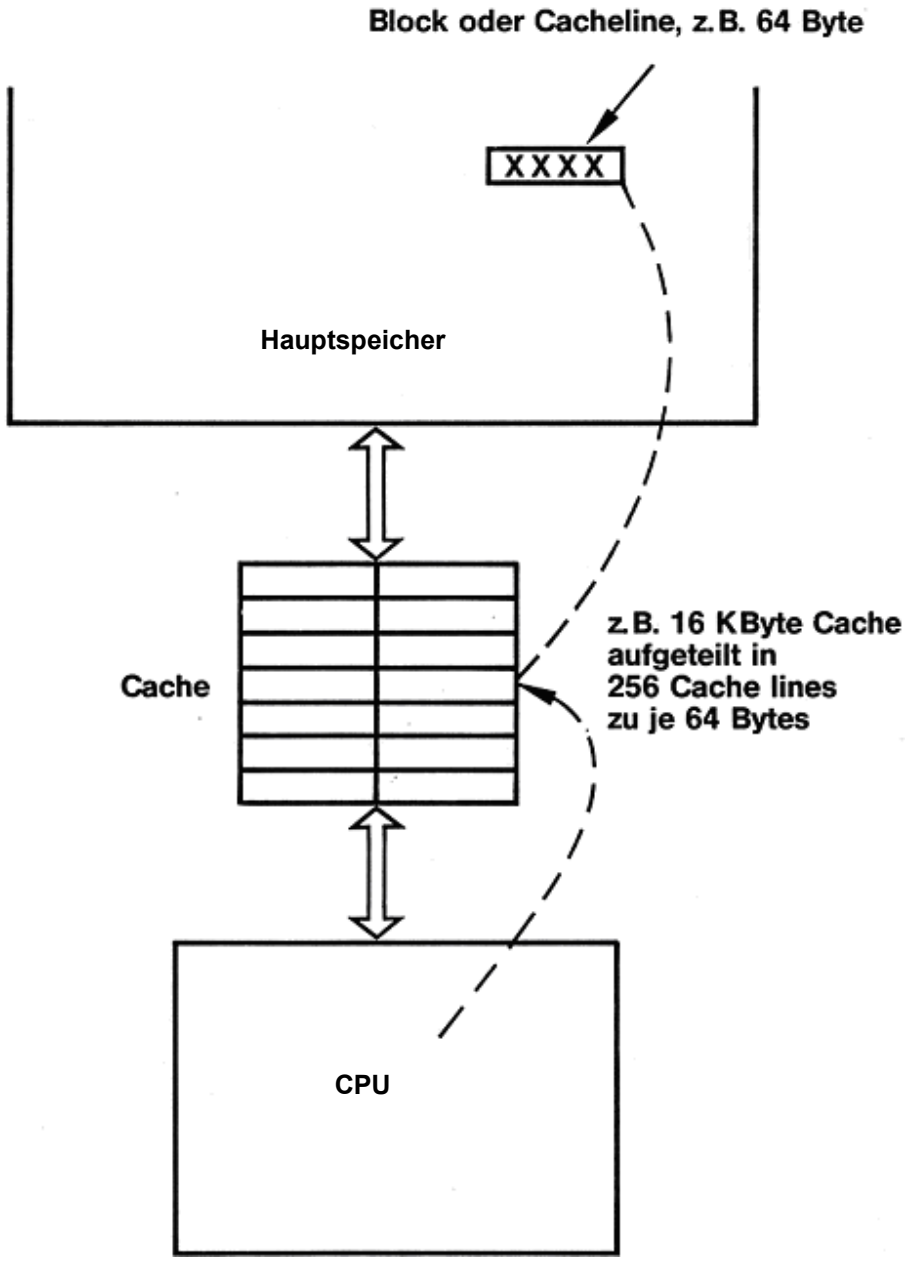
Cache Speicher



Hauptspeicher Zugriffszeiten sind zu langsam, um mit der heute möglichen Verarbeitungsgeschwindigkeit einer CPU mithalten zu können. Deswegen schaltet man zwischen Hauptspeicher und CPU einen Cache Speicher. Der Cache Speicher verwendet eine SRAM (Static Random Access Memory) Technologie. Das speichernde Element ist dabei ein FlipFlop. Es existieren viele unterschiedliche SRAM Technologien, mit unterschiedlichen Zugriffszeiten, Speicherdichten und Kosten. Während mit DRAMs aufgebaute Hauptspeicher eine Zugriffszeit in der Gegend von 100ns aufweisen, haben mit SRAMs aufgebaute Cache Speicher Zugriffszeiten zwischen 100ps und 10ns.

Wenn man von einem Cache redet, meint man meistens einen **Hauptspeicher Cache**. Da es auch Plattenspeicher-Caches und andere Caches gibt, ist die Unterscheidung wichtig.

Jeder moderne Rechner hat einen oder mehrere Caches. Die Existenz des Caches ist für den Benutzer praktisch unsichtbar. Es existieren auch keine Maschinenbefehle, mit denen der Programmierer den Inhalt des Caches manipulieren kann.

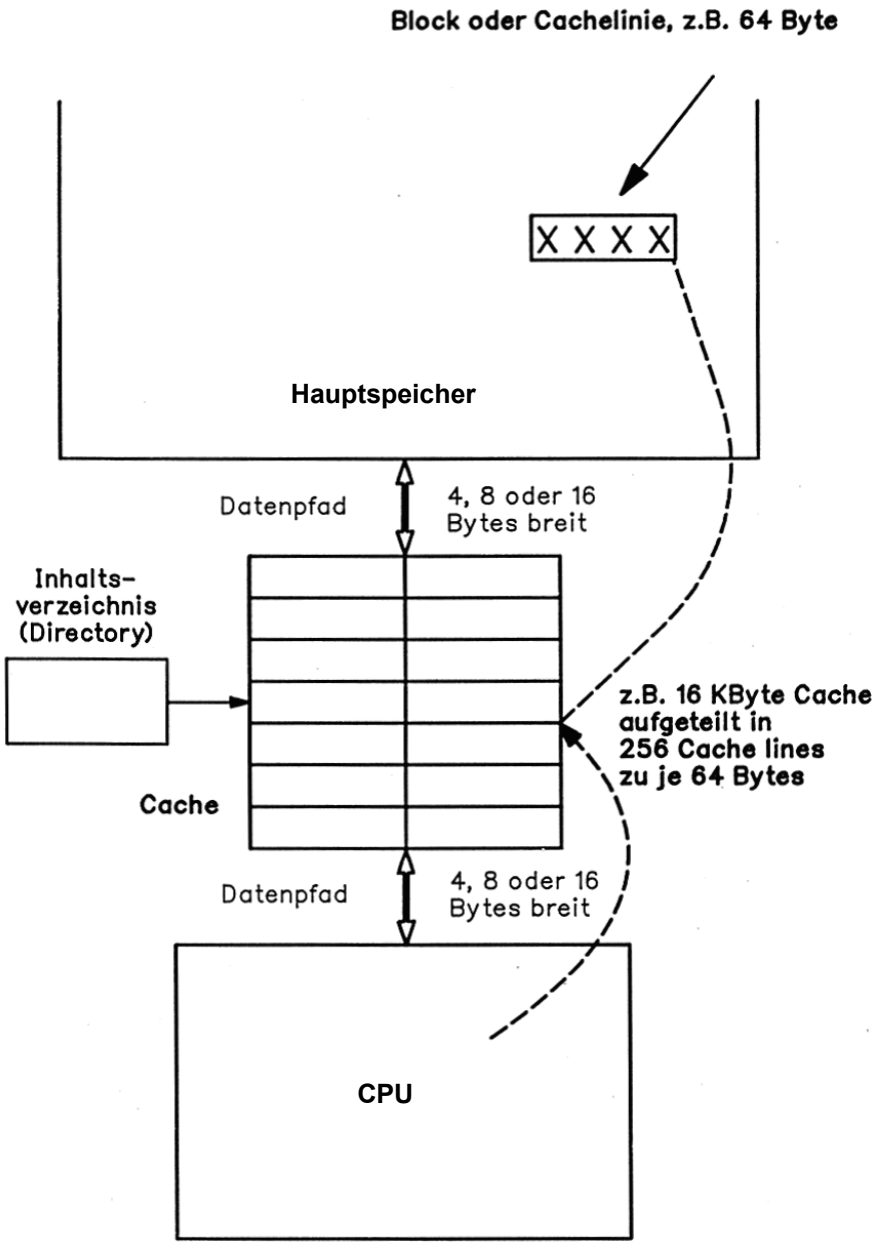


Struktur des Cache Speichers

Der Cache enthält in jedem Augenblick nur die Kopie einer Untermenge der Daten im Hauptspeicher. Diese Untermenge wird ständig ausgewechselt.

Hierzu werden Hauptspeicher und Cache Speicher in Blöcke mit einer identischen Größe aufgeteilt. Diese Blöcke werden als „Cache Lines“ bezeichnet. Die Größe der Cache Lines ist implementierungs-abhängig. Bei den z10 und z196 Mainframes sind es 256 Bytes, bei anderen Rechnern häufig weniger.

Ein Prozessor mit einer Cache Line-Size von 256 Byte wird aus dem Hauptspeicher immer nur Pakete dieser Größe in den Cache transportieren. Bei einem Hauptspeicher-Interface mit z.B. 256 Datenleitungen bedeutet das, dass jeder Cache Miss (wenn also angeforderte Daten nicht im Cache stehen) eine Adressierung und 8 Daten-Transferzyklen nach sich zieht.



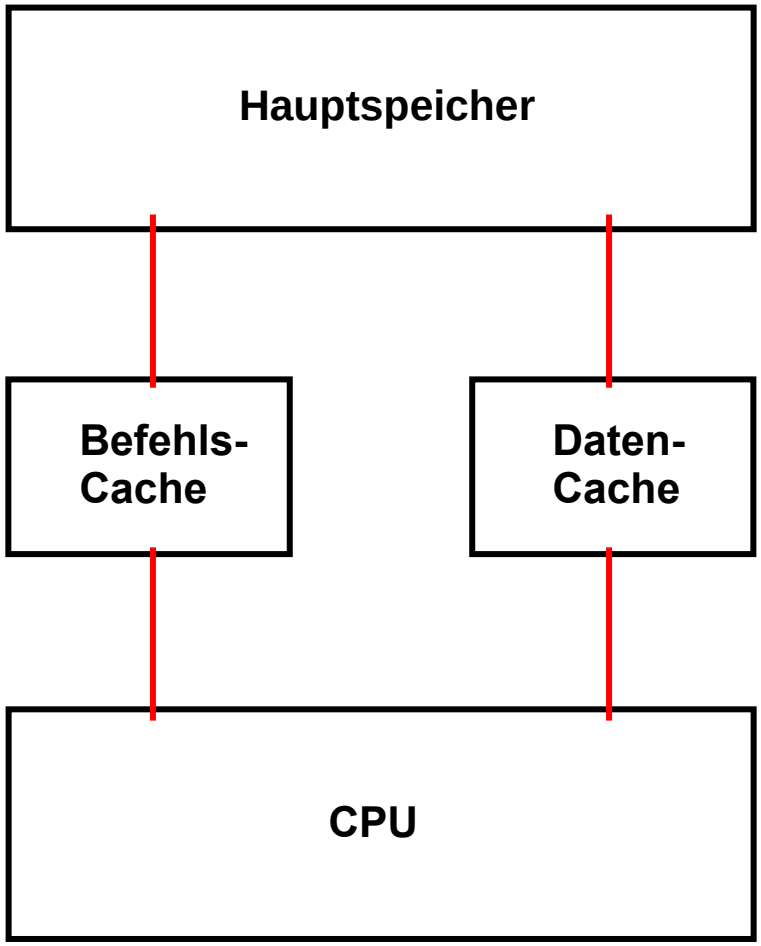
Die Anordnung der Cache Lines ist willkürlich und ändert sich ständig. Die CPU konsultiert ein „Cache Directory“ (Inhaltsverzeichnis) um eine bestimmte Cache Line innerhalb des Caches zu finden.

Das Cache Directory enthält je einen Eintrag für jede im Cache gespeicherte Cache Line. Der Eintrag enthält die Adresse der Cache Line im Hauptspeicher und im Cache.

Bei jedem Hauptspeicherzugriff durchsucht die CPU das Cache Directory in der Hoffnung, dass die benötigte Cache Line sich im Cache befindet. Wenn das nicht der Fall ist, entsteht ein Cache Miss. Dies bewirkt, dass die benötigte Cache Line in ihrer Gesamtheit vom Hauptspeicher in den Cache geladen wird. Vermutlich muss dabei eine andere (nicht mehr benötigte) Cache Line aus dem Cache ausgelagert werden, um Platz zu schaffen.

Einzelheiten hierzu in:

Udo Kebschull, Paul Herrmann, Wilhelm G: Spruth:
Einführung in z/OS und OS/390.
Oldenbourg-Verlag, 2002, ISBN 3-486-27214-4



Split Cache

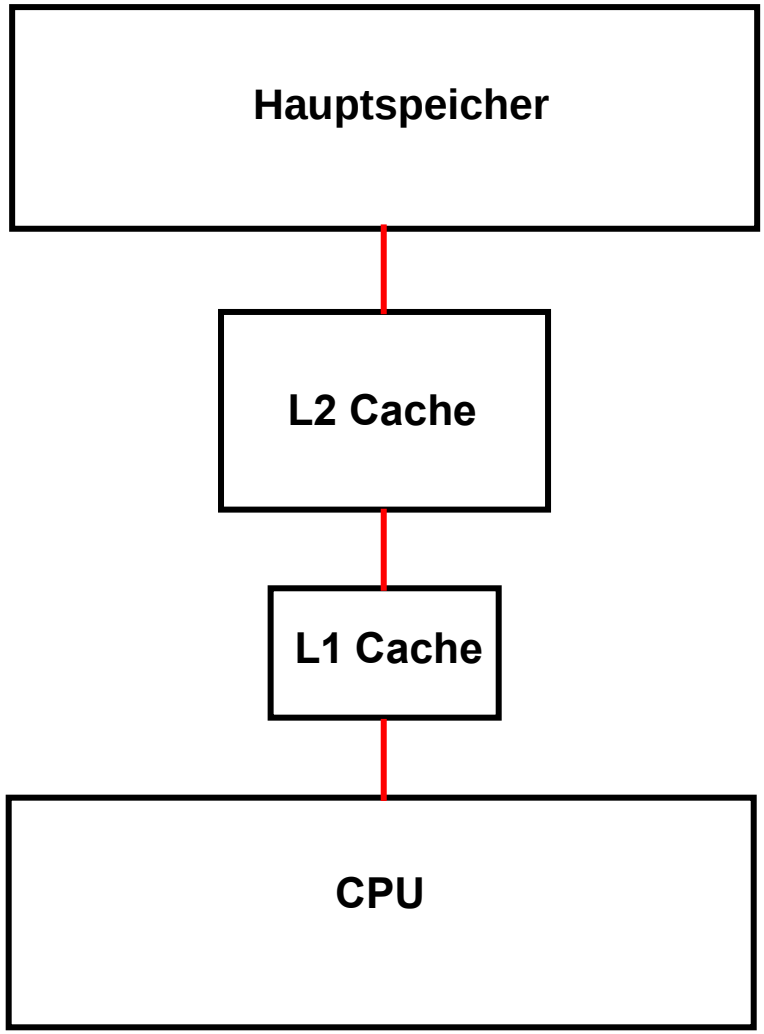
Dual Cache, Harvard Architecture Cache

Im Hauptspeicher sind Programme (bestehend aus Maschinenbefehlen) und Daten gespeichert. Diese befinden sich in unterschiedlichen Hauptspeicherbereichen und damit auch in unterschiedlichen Cache Lines.

Um den Durchsatz zu verbessern, besteht der Cache häufig aus zwei unabhängigen Cache Speichern (Dual Cache): Der Befehlscache enthält nur Maschinenbefehle und der Datencache enthält nur Daten.

Der z9, der z10 und der z196 Cache ist ein Dual Cache.

Der Befehlscache wird häufig als I-Cache (Instruction Cache) und der Datencache als D-Cache bezeichnet.

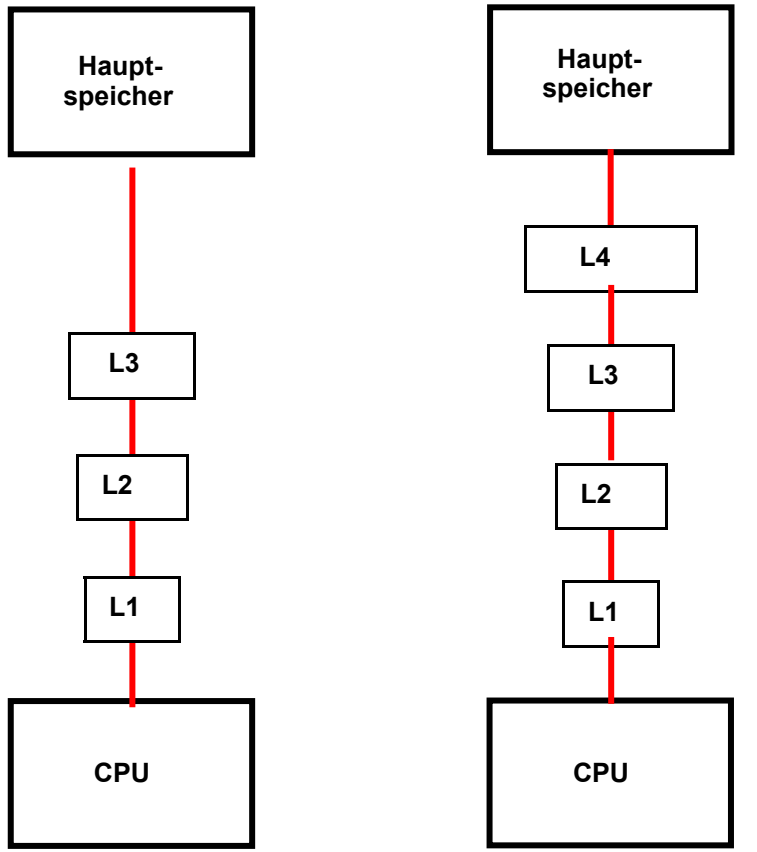


Second Level Cache

Es existieren zahlreiche SRAM Technologien, um Caches zu implementieren. Heute ist es üblich, mit einer Cache Hierarchie zu arbeiten. Hierbei wird ein „Level 1“ Cache (L1) mit besonders schnellen, aber teuren und platzaufwendigen SRAM Speicherzellen implementiert. Ein „Level 2“ Cache (L2) verwendet langsamere, aber dafür kostengünstigere SRAM Zellen.

Hierbei wird häufig, z.B. beim z9 Mainframe, der L1 Cache als Split Cache und der L2 Cache als Uniform Cache implementiert

Mainframe Cache Hierarchien



Moderne Cache Hierarchien sind noch komplizierter. Gezeigt sind die 3-stufige z10 Cache Hierarchie und die 4-stufige z196 Cache Hierarchie. Sinnvoll wurde dies durch die Erfindung einer neuartigen als eDRAM bezeichneten Cache Speicherzellen-Technologie, die im z196 Mainframe die SRAM Zellen in den L3 und L4 Caches ersetzt.

Verarbeitungsgrundlagen Teil 6

Weiterführende Informationen

Das wichtigste Dokument

z/Architecture Principles of Operation:

http://publibz.boulder.ibm.com/epubs/pdf/dz9zr001.pdf

Dies ist die “Bibel” der Mainframe Architektur. Frühere Ausgaben gehen zurück bis in das Jahr 1964.

Sternstunde der Menschheit

Die Entstehung der Mainframes und der S/360 Architektur im Jahre 1964 ist eines der ganz ungewöhnlichen Ereignisse, so unwahrscheinlich, dass es eigentlich nie hätte passieren dürfen.

Die Vorgeschichte, die hierzu führte, ist in einigen Dokumenten beschrieben, die sich teilweise wie ein Kriminalroman lesen, und gut zur Entspannung eignen. Dies sind besonders

1. die Erinnerungen von IBM Vizepräsident Bob O. Evans, der seinerzeit für die Entwicklung zuständig war
   http://www.informatik.uni-leipzig.de/cs/Literature/History/boevans.pdf , und

2. zwei Veröffentlichungen aus der Wirtschafts-Zeitung Fortune:

   I.B.M.'s $ 5,000,000,000 Gamble
   http://www.informatik.uni-leipzig.de/cs/Literature/History/FiveMillGamble1.pdf , und
   The rocky Road to the Marketplace
   http://www.informatik.uni-leipzig.de/cs/Literature/History/RockyRoad1.pdf

Der wissenschaftliche Hintergrund ist beschrieben in

http://www.informatik.uni-leipzig.de/cs/Literature/History/Amdahl.pdf

Einführung in die System z Betriebssystem:

Udo Kebschull, Paul Herrmann, Wilhelm G: Spruth:
Einführung in z/OS und OS/390.
Oldenbourg-Verlag, 2002, ISBN 3-486-27214-4

Einführung in die System Programmierung von z/OS:

ABCs of z/OS System Programming, Volume 10