

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN



Enterprise Computing

z/OS Betriebssystem

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

WS2016/17

System z und S/390 Betriebssysteme

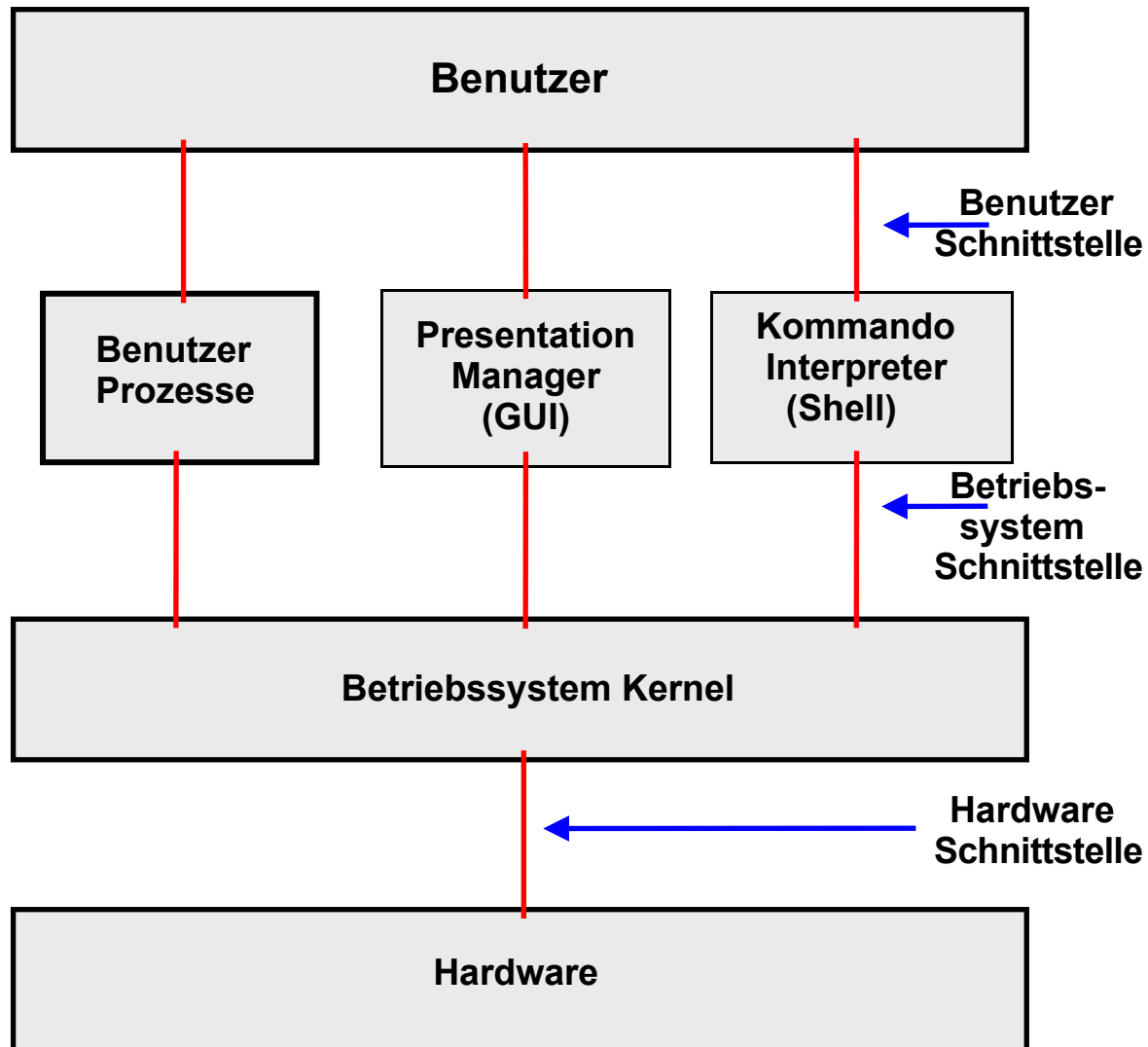
Im Laufe der Jahre sind eine ganze Reihe von Betriebssystemen für die System z Plattform entstanden:

Name	Hersteller	
• z/OS	IBM	große Installationen (früher als OS/390, MVS bezeichnet)
• z/VSE	IBM	mittelgroße Installationen
• z/VM	IBM	Virtualisierung, Software Entwicklung
• z/TPF	IBM	spezialisierte Transaktionsverarbeitung
• zLinux	Public Domain	Normale Suse oder Red Hat Adaption für System z Hardware
• UTS 4	Amdahl	Unix Betriebssystem, based on System V, Release 4 (SVR4)
• Open Solaris	Sun	Open Solaris Adaption für System z Hardware
• BS2000	Siemens/Fujitsu	von Siemens entwickelte Alternative zu OS/390

Alle System z bzw. S/390 Betriebssysteme sind Server Betriebssysteme, optimiert für den Multi-User Betrieb. Eine sehr ungewöhnliche Rolle spielt das z/TPF Betriebssystem.

z/OS Betriebssystem Teil 1

Übersicht



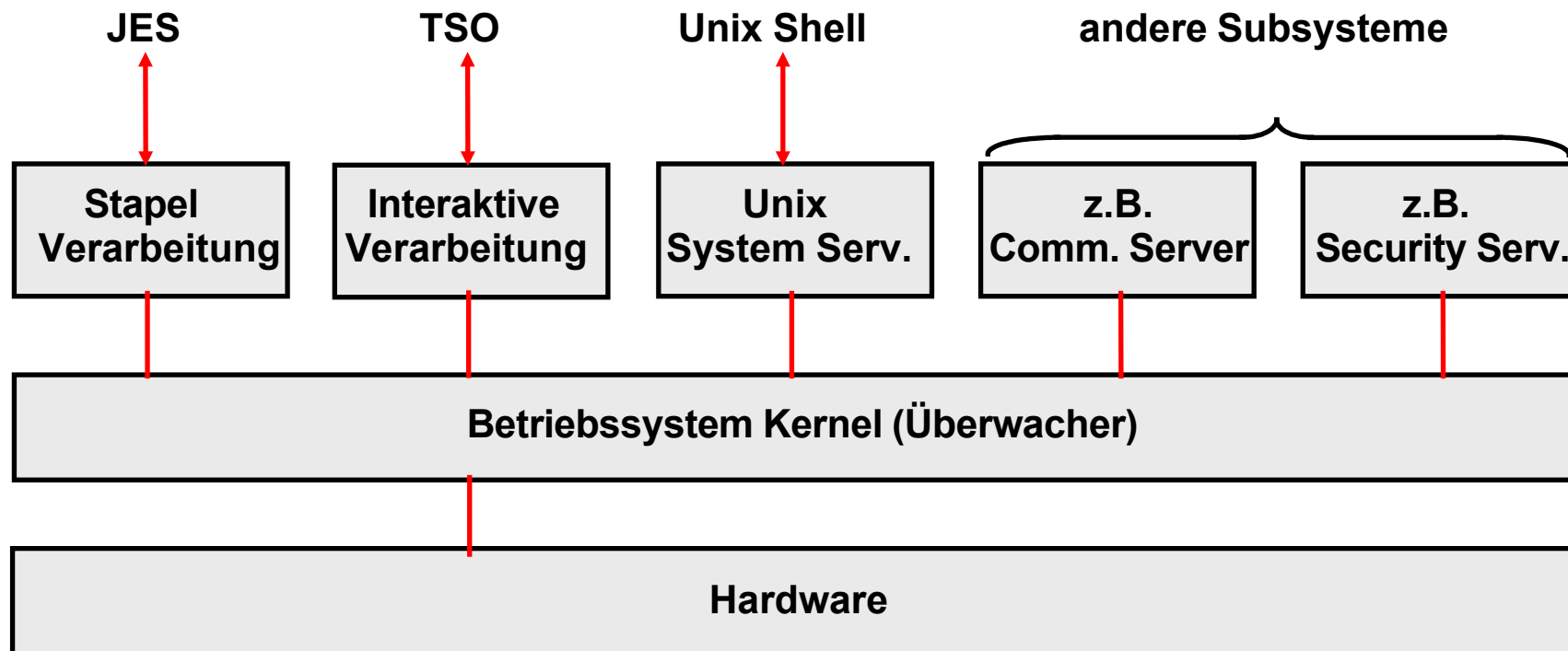
Schichtenmodell der Rechnerarchitektur Windows, Linux, Unix, z/OS

Bei allen Rechnern setzt der Betriebssystem Kernel (Überwacher, Supervisor, Basic Control Programm) direkt auf der Hardware auf. Der Kernel kann nur über wohl definierte Schnittstellen aufgerufen werden; in der Praxis sind das meistens Unterbrechungen.

Auf dem Kernel setzen Benutzerprozesse (von einem Benutzer geschriebenen Anwendungen) oder Systemprozesse auf. Systemprozesse werden auch als Subsysteme bezeichnet. Zwei der wichtigsten Systemprozesse sind der Presentation Manager (Graphical User Interface, GUI, z.B. Windows Desktop, Motiv, KDE) und der Kommando Interpreter (z.B. Windows DOS Shell, Unix Shell, TSO).

z/OS arbeitet fast ausschließlich mit verschiedenen Kommando Interpretern. GUIs werden vor allem für Anwendungen benutzt.

z/OS Grundstruktur



Die drei wichtigsten z/OS Subsysteme (Shells) für die Steuerung des Systems sind:

- JES (Job Entry Subsystem) für die Stapelverarbeitung
- TSO (Time Sharing Option) für die interaktive Verarbeitung (z.B. Programmentwicklung, Administration)
- Unix System Services, Posix kompatibles Unix Subsystem

Daneben existieren viele weitere Subsysteme, die Bestandteil des Betriebssystems sind, z.B. der Security Server, Communication Server, und viele andere

Begriffe

z/OS, wie auch Linux und Windows arbeitet mit dem Konzept eines Prozesses. Ausführbare Programme sind normalerweise in Programmbibliotheken auf einem Plattenspeicher abgespeichert. Ein Prozess ist der Aufruf und die Ausführung eines Programms. Bei z/OS werden die Prozesse auch **Task** genannt.

z/OS, wie auch Linux und Windows, arbeitet mit einer virtuellen Adressumsetzung (Address Translation). Hierbei existieren viele virtuelle Adressenräume, die mit der hexadezimalen Adresse Hex 00 ... 00 beginnen und eine einstellbare maximale Größe haben. z/OS bezeichnet diese virtuellen Adressenräume als (virtual) **Address Spaces**, Access Spaces oder auch als **Regions**. Die Begriffe sind austauschbar.

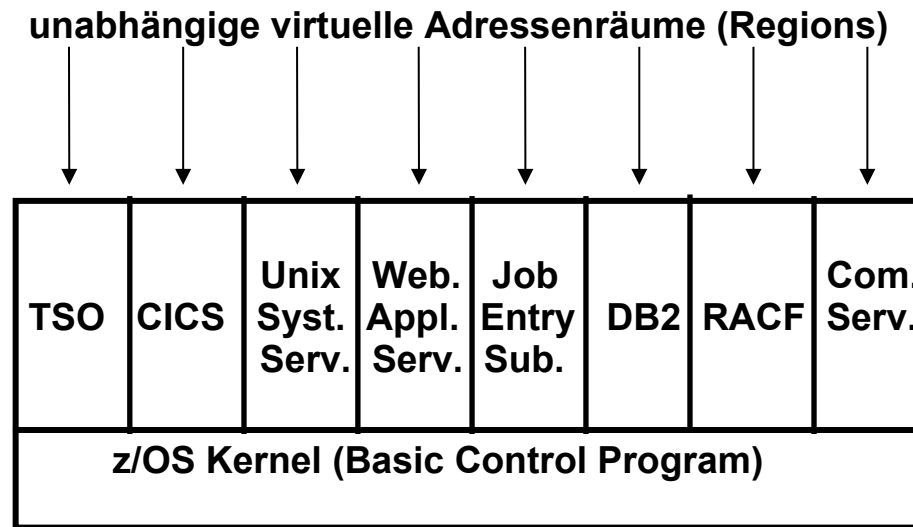
Moderne Rechner arbeiten multiprogrammiert. Auf einem Windows, Linux oder z/OS Rechner laufen in der Regel zahlreiche Prozesse parallel zueinander ab. Jeder der Prozesse läuft in einem eigenen virtuellen Adressenraum (Region, Address Space).

Manche der Prozesse sind Systemprozesse. Systemprozesse laufen als Teil des Betriebssystems. Wenn Sie unter Windows den Task Manager aufrufen (<ctrl><alt>), sehen Sie dort eine lange Liste von aktiven Systemprozessen in entsprechend vielen virtuellen Adressenräumen.

z/OS bezeichnet einige seiner Systemprozesse als **Subsysteme**. Subsysteme sind in sich abgeschlossene Softwareeinheiten. Manche Subsysteme wie JES, TSO, der Security Server oder der Communication Server sind ein Bestandteil des z/OS Betriebssystems. Andere Subsysteme wie die DB2 Datenbank, der WebSphere Application Server oder der CICS Transaktionsmanager sind optional, müssen extra installiert werden, und kosten zusätzliche Lizenzgebühren.

Im Gegensatz zu den Systemprozessen stehen die **Benutzer- (Anwendungs-) Prozesse**. Diese führen Programme (Anwendungen, Applications) aus, die vom Benutzer des z/OS Systems geschrieben wurden.

z/OS Grundstruktur



Systemprozesse und Anwendungsprozesse laufen in getrennten virtuellen Adressenräumen. Der z/OS Kernel unterstützt eine Vielzahl von virtuellen Adressenräumen, die im z/OS Jargon als **Regions** bezeichnet werden.

Einige der Regions beherbergen Subsysteme, die Teil des Betriebssystems sind, aber im Benutzerstatus laufen. Gezeigt sind eine der wichtigsten Subsysteme, die wir uns im Einzelnen noch ansehen werden.

Programmarten

z/OS wurde Anfang 1966 unter dem Namen **OS/360** als reines Stapelverarbeitungssystem eingeführt. Es wurde von Fred Brooks entwickelt, und diente als Vorlage für sein berühmtes Buch „The mythical Manmonth“ (http://en.wikipedia.org/wiki/The_Mythical_Man-Month).

Mit wachsendem Funktionsumfang wurde der Name immer wieder geändert: OS/360 → MFT → MVT → **MVS**. 1996 erfolgte eine Namensänderung von MVS nach **OS/390**. 2000 wurde der Name **z/OS** im Zusammenhang mit der neuen 64 Bit Adressierung eingeführt.

An Stelle von z/OS wird der Name MVS heute noch häufig gebraucht, was nicht ganz korrekt ist.

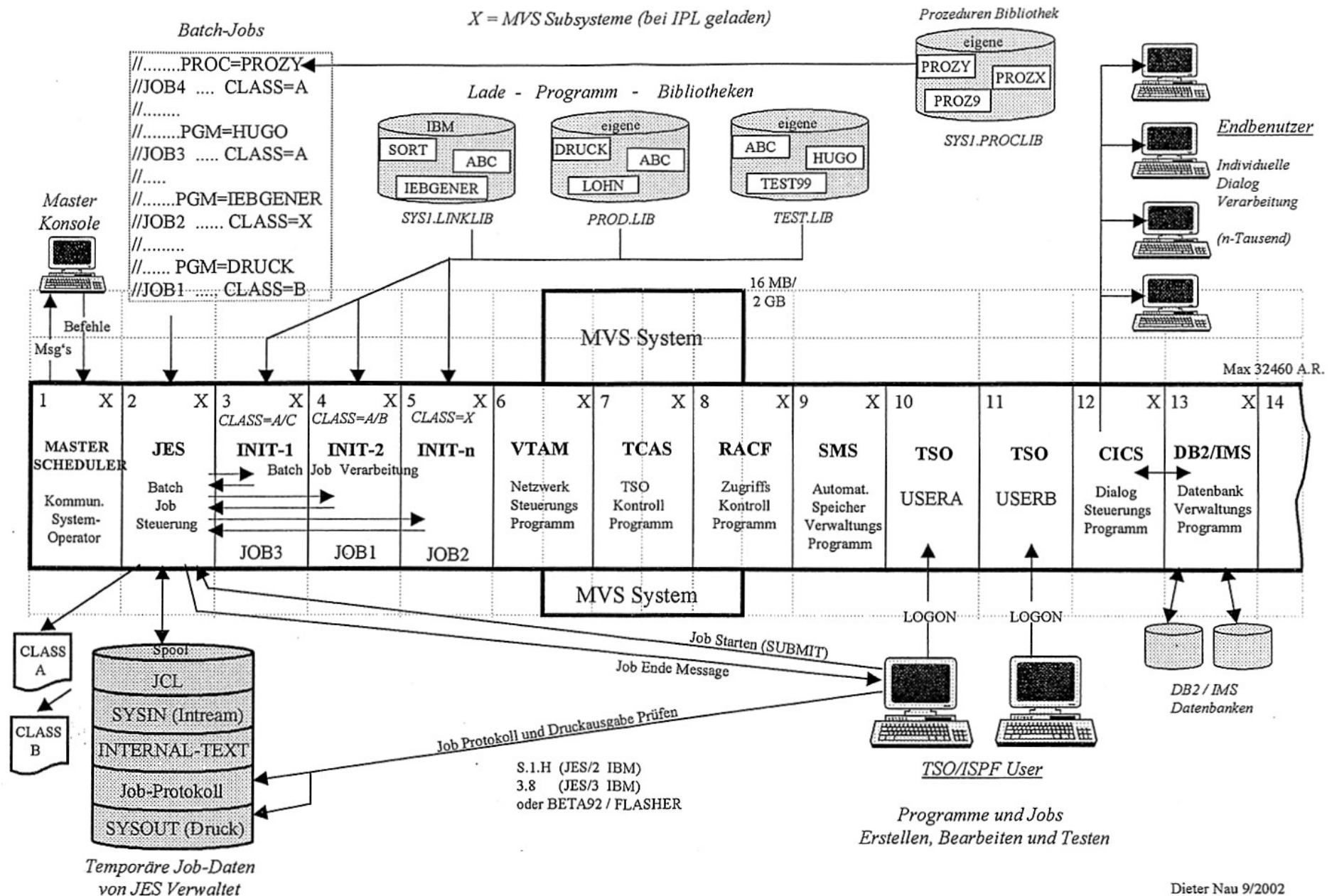
Subsysteme sind Programmprodukte wie Datenbanken und Transaktionsmonitore, die Laufzeitumgebungen für eigentliche Benutzerprogramme zur Verfügung stellen.

Benutzerprogramme können sein:

- klassische z/OS (bzw. OS/390) Hintergrundprogramme (Batch Programs)
- Benutzeranwendungen, die unter der Kontrolle von Subsystemen wie TSO, CICS, IMS oder Websphere ablaufen, oder
- UNIX-Programme, die die UNIX System Services unter z/OS ausnutzen.

System Management Funktionen werden für die Steuerung und Überwachung des Ablaufes benötigt. Es gibt sehr viele solcher Funktionen, sowohl von IBM als auch von Drittanbietern. Sie dienen der Überwachung des Betriebssystems und, da sich das Betriebssystem in weiten Bereichen selbst steuert, zur Überwachung der Subsysteme und der Benutzeranwendungen.

Unübersichtliche, aber lehrreiche Darstellung der z/OS Subsysteme



Dieter Nau 9/2002

Unübersichtliche, aber lehrreiche Darstellung der z/OS Subsysteme

Dargestellt sind die virtuellen Adressenräume für

- Master Scheduler,
- Batch Job Steuerung JES mit 3 Initiators,
- Communication Subsystem VTAM,
- Security Subsystem RACF,
- Dataset Verwaltungssystem SMS,
- Interaktive Benutzer mit TSO,
- Transaction Monitor CICS,
- Datenbank System.

Der Master Scheduler fährt z/OS hoch, steuert den laufenden Betrieb, und ermöglicht es einem Administrator mittels einer Master Konsole den laufenden Betrieb zu beobachten und in ihn einzugreifen. VTAM ist Teil des Communication Subsystems, RACF ist das Security Subsystem, und SMS ist für die Verwaltung von Dateien zuständig.

Alle diese Subsysteme werden im Laufe dieser Vorlesung näher erläutert.

z/OS Betriebssystem Teil 2

Job Control Language (JCL)

Zwei Arten der Datenverarbeitung

Betriebswirtschaftliche Großrechner betreiben Datenverarbeitung auf zwei unterschiedliche Arten:

Bei der **Interaktive Verarbeitung** dient der Mainframe als Server in einer Client/Server Konfiguration. Zahlreiche Klienten (meistens PCs, aber auch Geldausgabeautomaten oder Registrierkassen im Supermarkt) nehmen Dienstleistungen des Servers in Anspruch. In Großunternehmen wie z.B. der Volkswagen AG können das mehrere 100 000 Klienten sein, die in der Fachsprache oft als **Terminals** bezeichnet werden.

Dies ist ein Beispiel für die interaktive Verarbeitung: Sie sitzen an Ihrem PC und haben einen Excel Prozess gestartet. Sie arbeiten mit einer großen Excel Tabelle und stoßen eine Berechnung an, die viele Sekunden oder Minuten dauert.

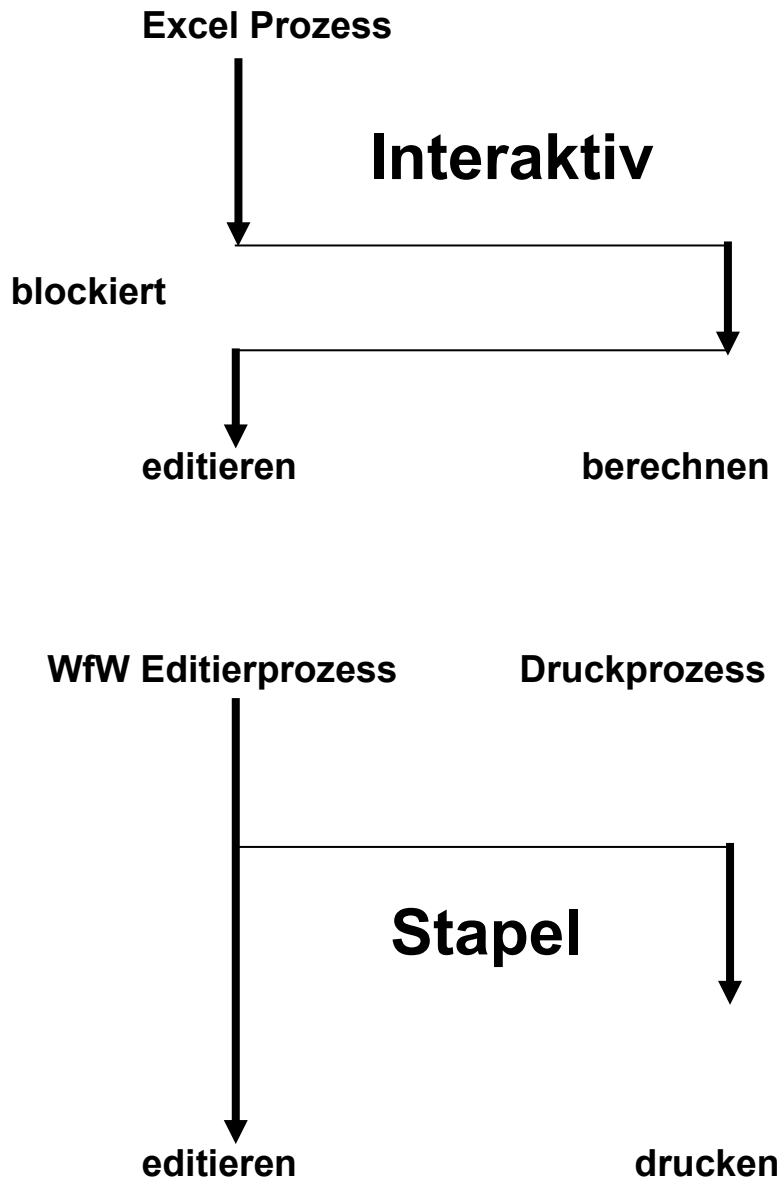
Während der Berechnung blockiert der Excel Prozess, reagiert z.B. nicht auf Tastatur-Eingaben. Der Grund ist, die Bearbeitung wird als Teil des Excel Prozesses durchgeführt; während dieser Zeit kann der Prozess nichts anderes machen.

Der Gegensatz zur interaktiven Verarbeitung ist die **Stapelverarbeitung**. Hier stößt ein Prozess, z.B. für länger laufende Verarbeitungsaufgaben, einen zweiten Prozess an.

Dies ist ein Beispiel für die Stapelverarbeitung: Sie sitzen an Ihrem PC und editieren ihre Masterarbeit mit Word for Windows. Dies ist ein interaktiver Prozess. Sie beschließen, die ganze Arbeit auf Ihrem Tintenstrahldrucker probeweise auszudrucken. Dies dauert viele Sekunden oder Minuten.

Während des Druckens blockiert der Word Prozess nicht. Sie können die Diplomarbeit weiter editieren.

Hierzu setzt Word for Windows neben dem Editierprozess einen getrennten Druckprozess auf, der als Stapelbearbeitungsprozess (batch job oder background Prozess) bezeichnet wird. Der Scheduler/Dispatcher des Betriebssystems stellt zeitscheibengesteuert beiden Prozessen CPU Zeit zur Verfügung.



Interaktive und Stapelverarbeitung

Die nebenstehende Grafik zeigt diesen Zusammenhang im zeitlichen Ablauf.

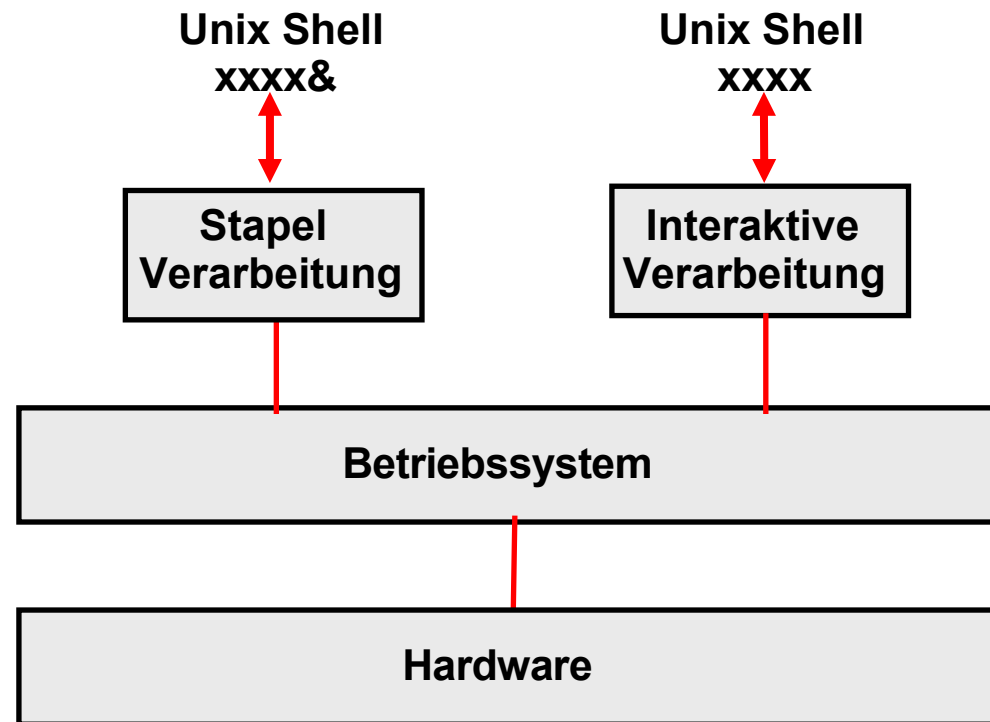
Ein Mainframe kann ohne weiteres 100 000 Benutzer mit 100 000 angeschlossene Terminals bedienen. Hierzu wird für jeden Benutzer ein eigener interaktiver Prozess in einem eigenen virtuellen Adressenraum gestartet.

Parallel dazu laufen auf dem Mainframe zahlreiche Stapelverarbeitungsprozesse, die als **Jobs** bezeichnet werden.

Im Rechenzentrum der Credit Suisse, einer Schweizer Großbank in Zürich, waren es im Frühjahr 2010 durchschnittlich 86 000 Jobs pro Tag. Die Ausführungszeit der einzelnen Jobs konnte Sekunden, Minuten, Stunden, und in Extremfällen auch Tage betragen.

Starten, Überwachung des Ablaufs und Terminierung der einzelnen Jobs ist die Aufgabe eines z/OS Subsystems, des **Job Schedulers**.

Interaktive und Stapelverarbeitung in Unix

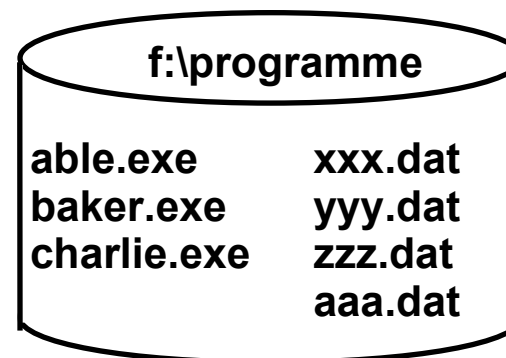


Unter Unix kann die Stapelverarbeitung (Batch Processing) als Sonderfall der interaktiven Verarbeitung betrieben werden. In der Shellsprache werden Batch-Aufträge durch ein nachgestelltes „&“ gekennzeichnet.

Beispiel eines DOS Jobs

auftrag.bat

```
f:
cd programme
able
baker
charlie
cd \
c:
```



able

```
=====
read xxx.dat
=====
write yyy.dat
=====
end
```

baker

```
=====
read yyy.dat
=====
write zzz.dat
=====
end
```

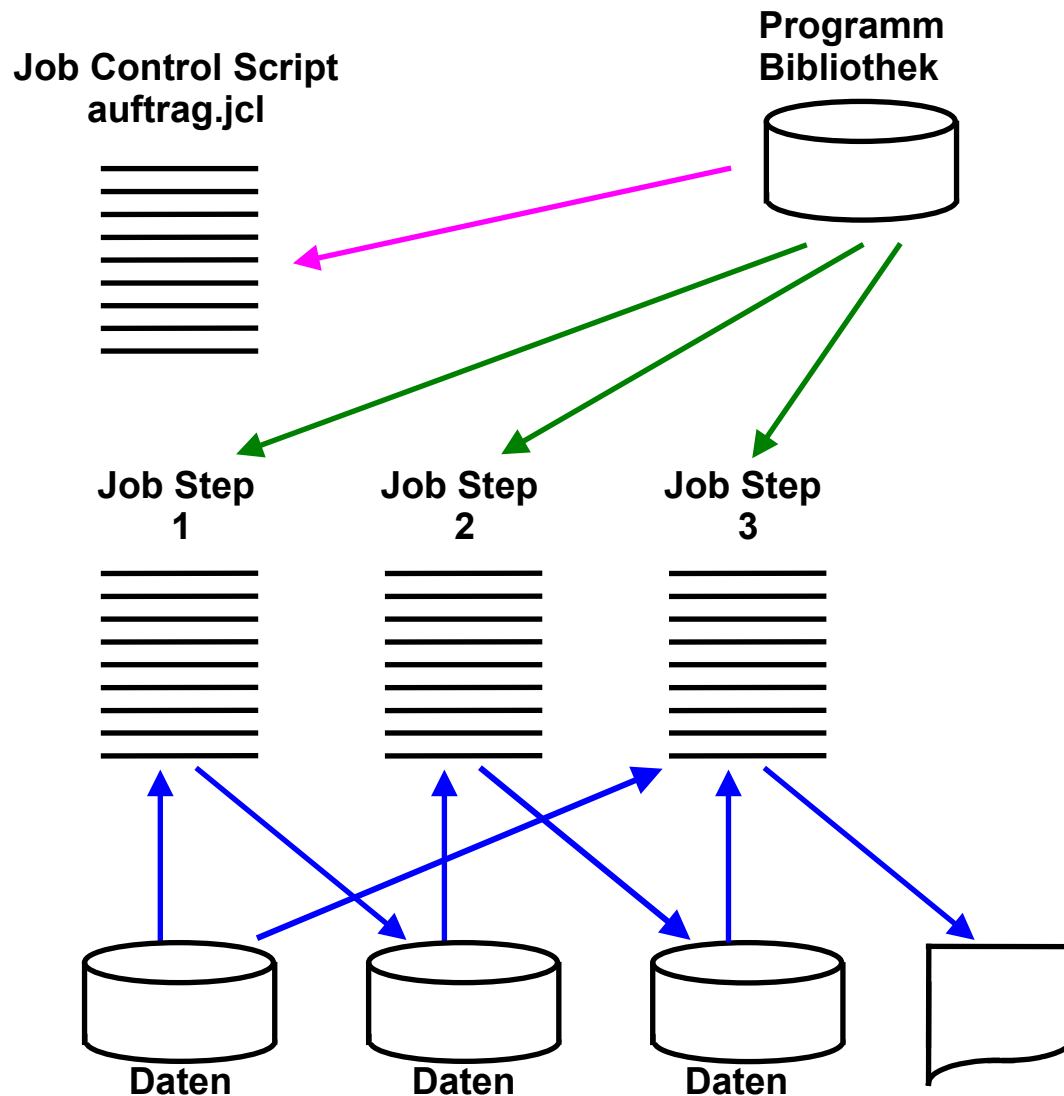
charlie

```
=====
read zzz.dat
=====
read xxx.dat
=====
write aaa.dat
=====
end
```

—

Als Beispiel sei angenommen, dass Sie unter Benutzung Ihrer Windows DOS Eingabeaufforderung drei Programme able, baker und charlie der Reihe nach ausführen wollen. Um dies zu automatisieren können Sie ein .bat Programm mit dem Namen auftrag.bat schreiben. auftrag.bat ist ein Job Script, welches die sequentielle Ausführung der drei Programme automatisch steuert. Es ist in der .bat Language geschrieben.

Konzept eines z/OS Jobs



Die Ausführung von `auftrag.bat` besteht aus drei Schritten, den Job Steps, die jeweils die Ausführung der drei Programme `able`, `baker` und `charlie` bewirken.

Unter z/OS verwendet man für die Job Steuerung eine eigene Scriptsprache, die Job Control Language, **JCL**.

Da das JCL Programm die verwendeten Dateien angibt, ist ein „late Binding“ der verwendeten Dateien an die auszuführenden Programme möglich.

Eine „Cataloged Procedure“ ist ein JCL Programm, welches vom Benutzer für eine spätere Verwendung zwischengespeichert wird (z.B. in einer vom Benutzer erstellten Library JCLLIB) und bei Bedarf mittels eines JCL Befehls aufgerufen wird.

Die Eigenschaften eines Stapelverarbeitungsprozesses sind häufig:

- Lange Laufzeit
- wird häufig periodisch zu festgelegten Zeiten ausgeführt
- Hohes Datenvolumen. Es kann aus Tausenden oder Millionen von Datenbankelementen (z.B. Reihen in einer SQL Tabelle) bestehen
- Für die Daten möglicherweise komplexe Verarbeitungsanforderungen
- Der Verarbeitungsprozess benötigt möglicherweise große Datenmengen von einem anderen Rechner. Diese werden als eine große Datei zu einer bestimmten Zeit angeliefert.
- Der Stapelverarbeitungsprozess läuft asynchron zu irgendwelchen Benutzer-Aktionen. Er ist nicht Teil einer Benutzer Session in einem Online (Client/Server) System.
- Wird typischerweise nicht von einem Benutzer eines Online Systems gestartet, Normalerweise startet ein Online Benutzer keinen Stapelverarbeitungsprozesses, und wartet auch nicht auf eine Antwort.

Script Sprachen

Sie kennen vermutlich bereits eine der gängigen Scriptsprachen wie Pearl, PHP, Java Script, Tcl/Tk, Python, Ruby usw.

Sie haben vielleicht auch schon einen der erbitterten Religionskriege miterlebt, welche Scriptsprache die beste ist, und welche grottenschlecht ist. Glauben Sie mir, es gibt keine „beste“ Scriptsprache.

Eine weitere Scriptsprache ist REXX (*Restructured Extended Executor*). REXX wird vor allem auf Mainframes eingesetzt. REXX Interpreter sind aber auch für Windows, Apple, Linux, alle Unix Dialekte usw. verfügbar, und weiter verbreitet als allgemein angenommen. REXX wird vor allem von Leuten benutzt, die schon auf dem Mainframe damit gearbeitet haben.

Hier ist ein kurzes Programm in REXX

```
/* A short program to greet you */
/* First display a prompt */

say 'Please type your name and then press ENTER: '
parse pull answer /* Get the reply into answer */

/* If nothing was typed, then use a fixed greeting */
/* otherwise echo the name politely */

if answer='' then say ' Hello Stranger! '
              else say ' Hello ' answer '! '
```

Auffallend ist, dass an Stelle des Schlüsselwortes “write” das Schlüsselwort “say” benutzt wird. Daran können sie immer erkennen, ob ein Script Programm in REXX geschrieben ist.

Shell Scripte

Einige Script Sprachen sind sog. shell Scripte. Dazu gehören .bat Files unter Windows, bash unter Unix und Linux und eben JCL unter z/OS. Shell Scripts sind aus Sequenzen von shell Kommandos entstanden. Auch die make file, die Sie als C++ Programmierer für das Übersetzen Ihres C++ Programms benutzen , benutzt eine eigenes Shell Script.

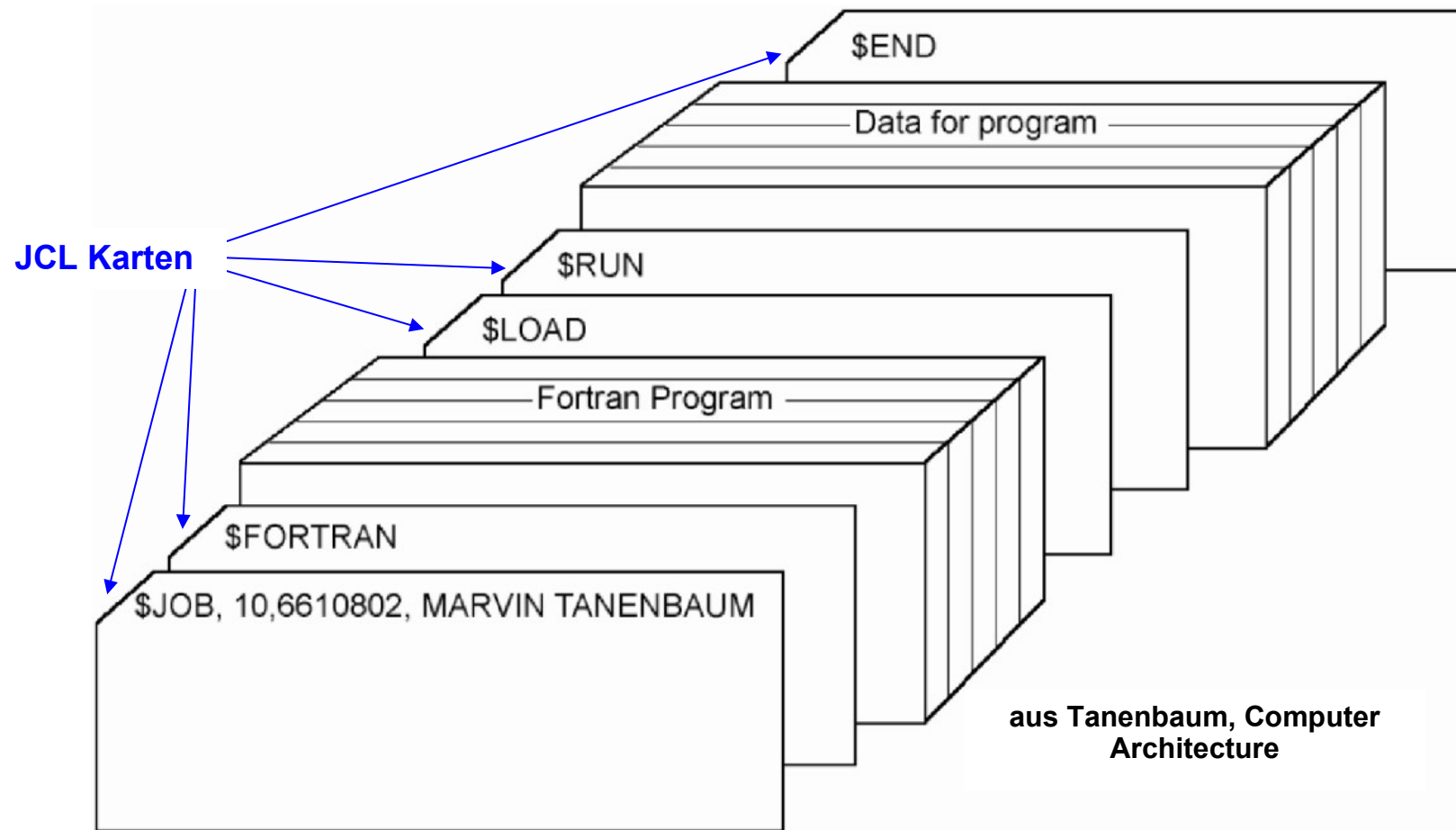
Shell Scripte sind in der Regel für den Anfänger unübersichtlicher und schwieriger zu verstehen als normale Script Sprachen wie Tcl/Tk, REXX oder PHP. Das gilt besonders auch für JCL.

Die JCL Syntax macht einen absolut archaischen Eindruck, und ist für den erstmaligen Benutzer ein Kulturschock. Das wird auch für Sie gelten, wenn sie Ihr erstes JCL Script schreiben, also „be prepared“. JCL hat sicher viel dazu beigetragen, Mainframes zu dem Ruf „obsolete Technology“ zu verhelfen.

Tatsache ist, nachdem Neulinge ihren ersten Schock überwunden haben (das ist nach wenigen hundert Zeilen JCL Code der Fall), beginnen sie JCL heiß und innig zu lieben. Es hat viele Versuche gegeben, JCL durch eine „moderne“ Scriptsprache zu ersetzen – bisher mit wenig Erfolg. Tatsache ist, JCL ist relativ leicht erlernbar, sehr mächtig und sehr produktiv.

JCL wurde zusammen mit OS/360 (der ersten Version des heutigen z/OS) entwickelt und hat Ähnlichkeiten mit Unix Shell Scripts; ein Beispiel ist die Ähnlichkeit des Unix dd und des JCL DD Kommandos.

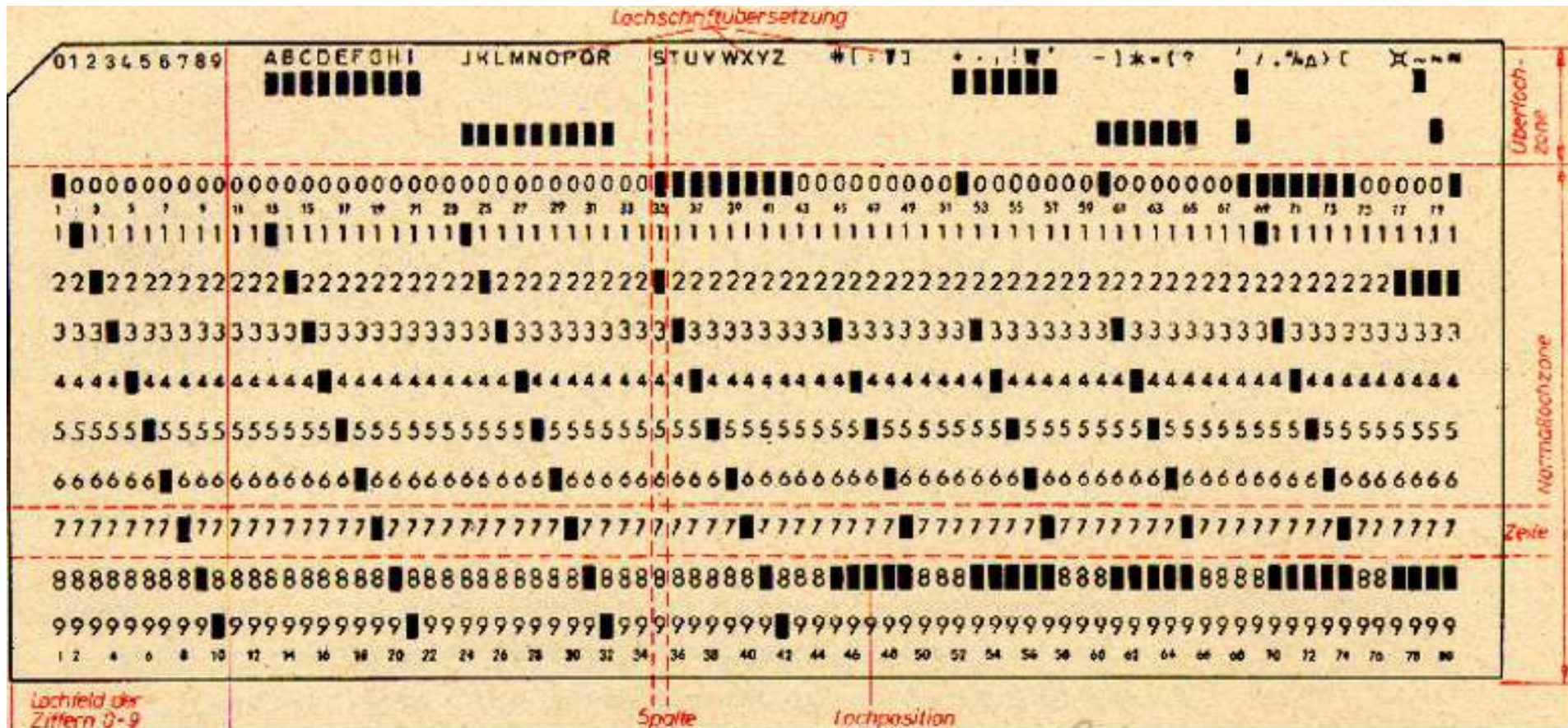
Es ist unmöglich, JCL zu verstehen, ohne sich mit der historischen Entwicklung auseinanderzusetzen. Diese geht auf die frühere Benutzung von **Lochkarten** zurück. Lochkarten waren bis in die 70er Jahre beliebte und kostengünstige Elemente für die Speicherung von Daten.



Gezeigt ist die Implementierung eines FORTRAN Programms etwa Anno 1960. Es besteht aus einer Reihe von JCL Karten, ein JCL Statement pro Karte. Zwischen den JCL Karten befindet sich das FORTRAN Programm Karten Deck, jeweils eine Karte pro Fortran Statement, sowie ein weiteres Karten Deck mit den von dem vom Fortran Programm zu verarbeitenden Daten. Das gesamte Kartendeck wurde von dem Lochkartenleser des Rechners in den Hauptspeicher eingelesen und verarbeitet. Die Ausgabe erfolgte typischerweise mittels eines angeschlossenen Druckers.

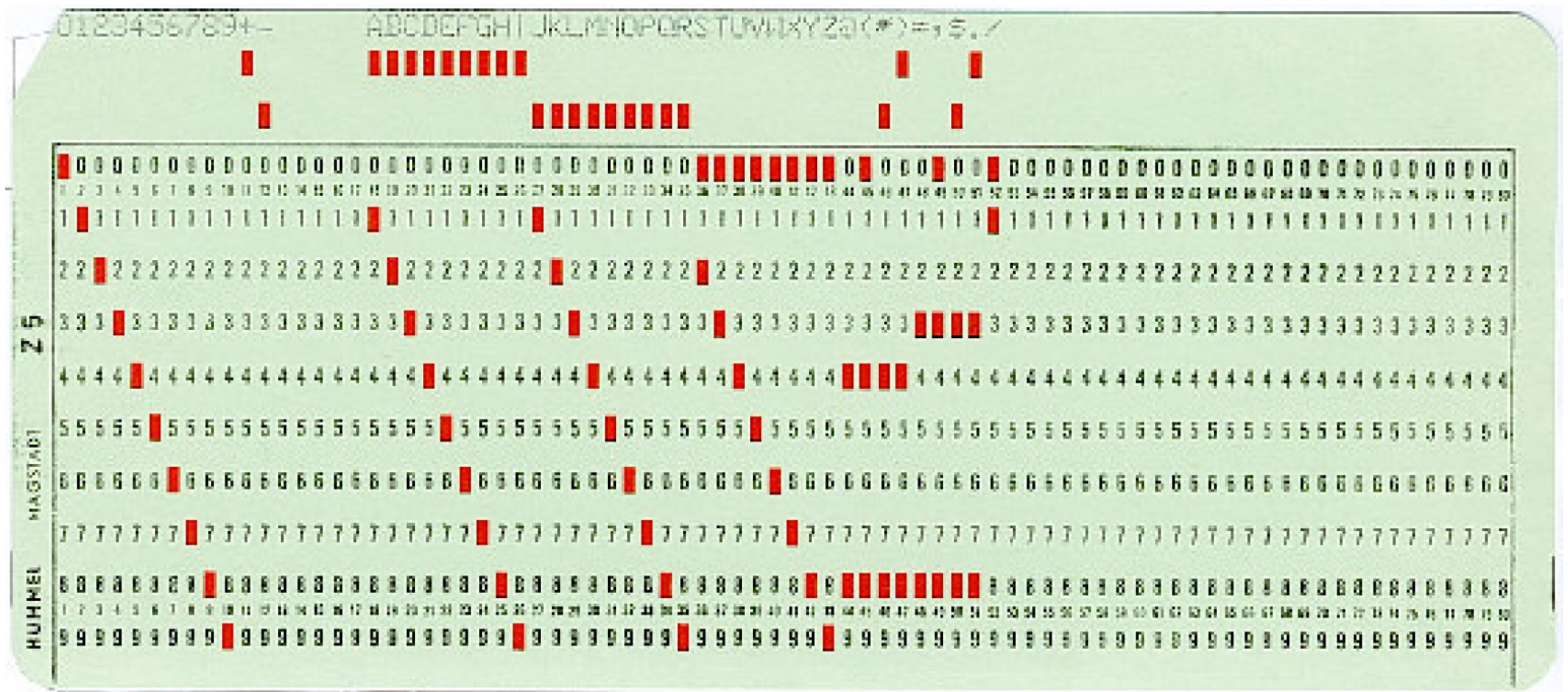
Später wurde es üblich, das Fortran Programm und die Daten auf Magnetband (und noch später auf einem Plattenspeicher) zu speichern, und das Programmdeck und das Datendeck durch zwei Library Call Lochkarten zu ersetzen. Noch später speicherte man dann das JCL Script ebenfalls in einer Library auf dem Plattenspeicher.

IBM Lochkarte



Die IBM Lochkarte, etwa 19 x 8 cm groß, wurde 1928 eingeführt. Vorläufer mit einem etwas anderen Format wurden von Herrmann Hollerith erfunden und 1890 bei der Volkszählung in den USA und wenig später auch in Deutschland eingesetzt.

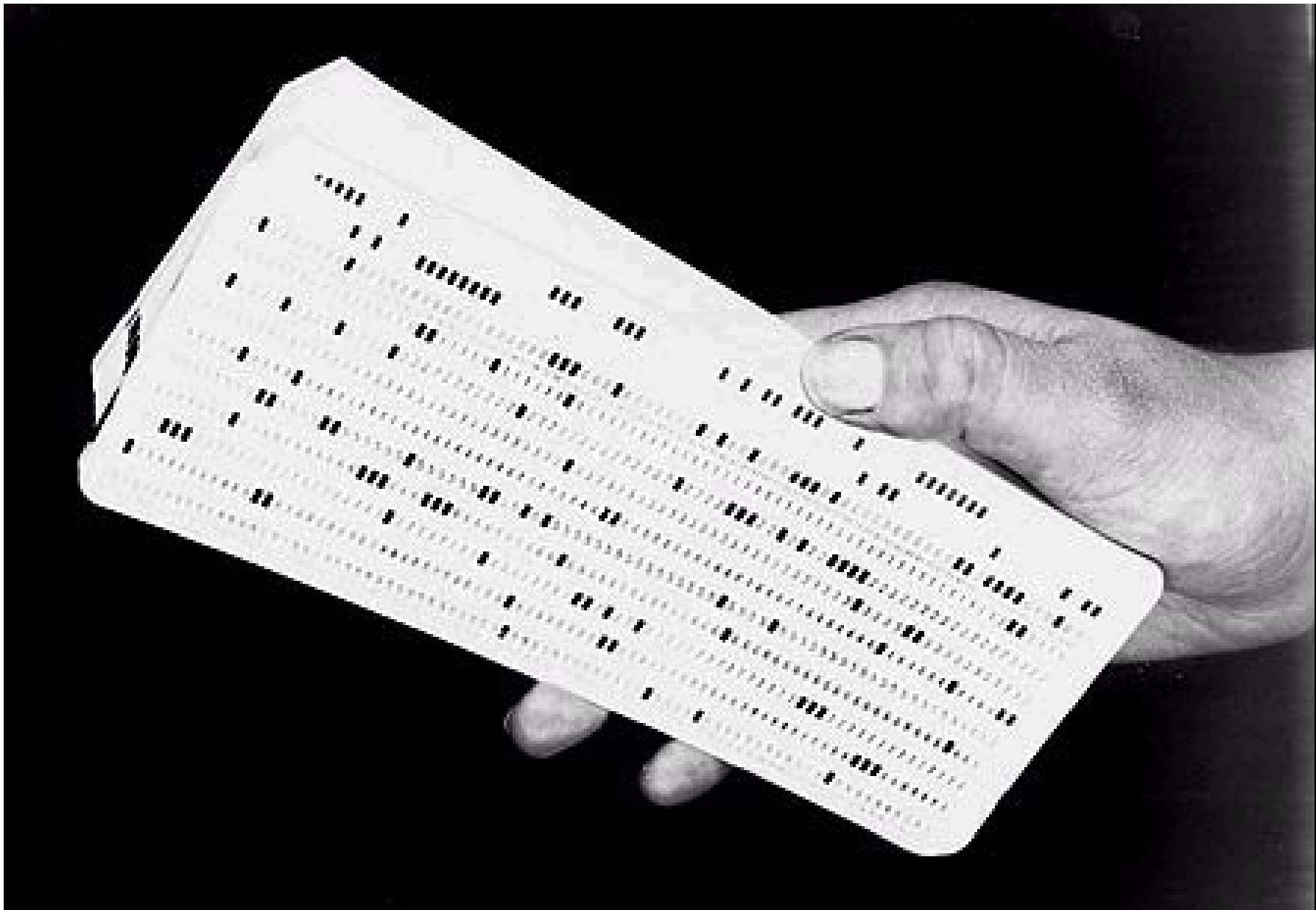
Die IBM Lochkarte besteht aus 80 Spalten. In jede Spalte konnte an einer von 12 Stellen jeweils ein Loch gestanzt werden. Die Position konnte in einem Lochkartenleser abgefühlt werden. Damit war es möglich, mittels der sog. BCD Kodierung (binary coded decimal) in jeder Lochkarte 80 alphanumerische Zeichen zu speichern.



Zahlreiche Darstellungen von Lochkarten sind zu finden unter

[http://www.google.de/search?](http://www.google.de/search?q=punched+card&hl=de&lr=&as_qdr=all&prmd=imvns&tbm=isch&tbo=u&source=univ&sa=X&ei=wiglUMu4FobV4QSk0oHICA&ved=0CCoQsAQ&biw=1516&bih=963)

[q=punched+card&hl=de&lr=&as_qdr=all&prmd=imvns&tbm=isch&tbo=u&source=univ&sa=X&ei=wiglUMu4FobV4QSk0oHICA&ved=0CCoQsAQ&biw=1516&bih=963](http://www.google.de/search?q=punched+card&hl=de&lr=&as_qdr=all&prmd=imvns&tbm=isch&tbo=u&source=univ&sa=X&ei=wiglUMu4FobV4QSk0oHICA&ved=0CCoQsAQ&biw=1516&bih=963)



Ein Stapel Lochkarten

<http://www-03.ibm.com/systems/z/os/zvse/about/history1960s.html>

JCL Format

Ein JCL Script besteht aus einzelnen JCL Statements. Ein JCL Statement passte früher auf eine Lochkarte mit 80 Spalten, und daran hat sich bis heute nichts geändert.

Wie auch bei anderen Programmiersprachen besteht ein JCL Statement aus mehreren (genau fünf) Feldern.

//	Label	Operation	Parameter	Kommentar
----	-------	-----------	-----------	-----------

Um die Logik zum Parsen eines Statements zu vereinfachen, führte man einige Konventionen ein:

- Jedes Feld beginnt immer an einer bestimmten Spalte der Lochkarte. Das vereinfachte damals das Parsen des Statements. Diese Spaltenabhängigkeit existiert auch heute noch !!! Wenn Sie dagegen verstoßen, gibt es eine Fehlermeldung.
Warnung: Dies ist mit großem Abstand der häufigste Programmierfehler beim Bearbeiten unserer ersten Mainframe Tutorials.
- Jedes JCL Statement beginnt mit den beiden Zeichen // (forward slashes) . Damit war es möglich, die JCL Statement leicht von den Lochkarten des Fortran Decks und des Datendecks zu unterscheiden. Die beiden Slashes befinden sich in Spalte 1 und 2 .
- Das Label Feld beginnt in Spalte 3 , die maximale Länge ist 8 .
- Das Operation Feld folgt dem Label Feld, getrennt durch ein Leerzeichen, in Spalte 12 .
- Das Parameter Feld (Operand Feld) folgt dem Operation Feld, getrennt durch ein (oder mehr) Leerzeichen
- Alles was hinter dem Operand Feld folgt, ist ein Kommentar. Zwischen Operand und Kommentar muss sich (mindestens) ein Leerzeichen befinden.

z/OS MVS JCL Reference SA22-7597-10 Eleventh Edition, April 2006 , chapter 3, <http://ulita.ms.mff.cuni.cz/pub/predn/NSWI119/iea2b661.pdf>

Ein einfaches JCL Script

Statement Bezeichnung beginnt in Spalte 12



```
//SPRUTHC JOB (123456) , ' SPRUTH ' , CLASS=A , MSGCLASS=H , MSGLEVEL=(1 , 1) ,  
//          NOTIFY=&SYSUID , TIME=1440  
//PROCLIB JCLLIB ORDER=CBC . SCBCPRC  
//CCL     EXEC PROC=EDCCB ,  
//          INFILE=' SPRUTH . TEST . C (HELLO1) '  
//          OUTFILE=' SPRUTH . TEST . LOAD (HELLO1) ' , DISP=SHR
```

Label, Spalte 3 - 10

JOB Statement markiert den Anfang eines Jobs

EXEC Statement bezeichnet Prozedur, die ausgeführt werden soll

PROC Statement gibt die Adresse einer Prozedur an

DD Statement bezeichnet die zu benutzenden Dateien (hier nicht verwendet)

Achten Sie darauf, dass Ihr JCL Script die richtigen Spalten benutzt !!!

Beispiel

Beispiel für einen JCL Befehl:

// DCB=(RECFM=FB,LRECL=80,BLKSIZE=400)

DCB, RECFM, FB, LRECL und **BLKSIZE** sind Schlüsselwörter der JCL Sprache.

Der Befehl besagt, daß die hiermit angesprochene Datei (bzw. deren Data Control Block, DCB) ein Fixed Block (FB) Record Format (RECFM) hat (alle Datensätze haben die gleiche Länge), dessen Länge (Logical Record Length LRECL) 80 Bytes beträgt, und daß für die Übertragung vom/zum Hauptspeicher jeweils 5 Datensätze zu einem Block von (Blocksize BLKSIZE) 400 Bytes zusammengefaßt werden.

Literatur

M.Winkler: „MVS/ESA JCL“. Oldenbourg, 3. Auflage, 1999

<http://www.mainframezone.com/blog/the-it-skills-shortage-where-will-we-find-mainframe-talent>

A February 2011 study conducted by SHARE entitled, “CLOSING THE IT SKILLS GAP: 2011 SHARE Survey for Guiding University & College IT Agendas”, found that half of the companies surveyed hire new IT employees straight out of school, with relatively little actual work experience. The study also indicates a strong demand for mainframe skills with the finding, “In terms of platform-specific skills, companies seek applicants skilled in running two types of environments—database administration and mainframe administration. Specific mainframe administration skill areas also are in demand by a majority, or close to a majority, of companies in the survey — 55% seek mainframe administrative skills, and half are in need of skills involving JCL, or Job Control Language.”

So archaisch JCL auch sein mag, es ist nach wie vor sehr populär. Nach Absolvierung des Einarbeitungs-Kulturschocks finden die Benutzer JCL sehr produktiv und mächtig.

Was ist die Funktion von „Submit“ bzw. „Sub“

TSO bzw. ISPF sind Command Line Shells, vergleichbar mit den Korn, bash Shells unter Unix/Linux bzw. der DOS Eingabeaufforderung unter Windows. Solche Shells beinhalten eine primitive Ausführungsumgebung, die es ermöglicht, von der Kommandozeile aus ein Programm (z.B. xyz.exe) direkt auszuführen. Dies ist auch unter der TSO Shell möglich.

Professioneller ist die Programmausführung unter einem Anwendungsserver. Viele Anwendungsserver sind Bestandteil einer Client/Server Umgebung (sog. interaktive Server). Ein Beispiel hierfür ist z.B. der Web Application Server Ihrer persönlichen Home Page, auf dem eine CGI oder Java Servlet Anwendung läuft.

Andere Anwendungsserver implementieren Stapelverarbeitung, und das JES Subsystem unter z/OS ist eine führende Implementierung. Ein JES Job wird typischerweise durch die Ausführung eines JCL Scripts gestartet, und genau dies tun Sie bei der Durchführung Ihres ersten Cobol Tutorials, bei dem Sie das unten stehende JCL Script erzeugen.

Das Kommando „SUB“ (Abkürzung für Submit) bewirkt, dass das JCL Script zwecks Ausführung an das JES Subsystem übergeben wird. JES generiert hierfür einen Job mit einer eindeutigen Job Nummer, und übergibt ihn zwecks Ausführung an einen JES „Initiator“ (siehe den folgenden Teil 3 dieses Themas) .

Der wichtigste Befehl in dem JCL Script auf der folgenden Seite ist „EXEC IGYWCL“. Er bewirkt, dass ein weiteres JCL Script mit dem Namen IGYWCL aus einer Programmbibliothek aufgerufen wird. IGYWCL enthält Anweisungen, die bewirken, dass der in der folgenden Zeile angegebene Data Set TEST.COB.(COB02) compiled und linked wird.

File Edit Confirm Menu Utilities Compilers Test Help

```
-----  
EDIT          PRAK085.TEST.CNTL(COBSTA02) - 01.02          Columns 00001 00072  
***** Top of Data *****  
==MSG> -Warning- The UNDO command is not available until you change  
==MSG>          your edit profile using the command RECOVERY ON.  
000100 //PRAK085C JOB (),CLASS=A,MSGCLASS=M,MSGLEVEL=(1,1),NOTIFY=&SYSUID,  
000200 //          REGION=4M  
000300 //STEP1     EXEC IGYWCL  
000400 //COBOL.SYSIN DD DSN=&SYSUID..TEST.COB(COB02),DISP=SHR  
000500 //LKED.SYSLMOD DD DSN=&SYSUID..TEST.LOAD,DISP=SHR  
000600 //LKED.SYSIN DD *  
000700 NAME COB02(R)  
000800 /*  
***** Bottom of Data *****
```

Command ==> **SUB** Scroll ==> PAGE
F1=Help F3=Exit F5=Rfind F6=Rchange F12=Cancel



z/OS Betriebssystem Teil 3

Job Entry Subsystem (JES)

Stapelverarbeitung unter Unix

Der **cron**-Daemon ist eine Jobsteuerung von Unix bzw. Unix-artigen Betriebssystemen wie Linux, BSD oder Mac OS X, die wiederkehrende Aufgaben (cronjobs) automatisch zu einer bestimmten Zeit ausführen kann. cron ist die Software, die dem z/OS Batch Processing Subsystem (Job Entry Subsystem, **JES**) am nächsten kommt. Allerdings geht der JES Funktionsumfang weit über den von cron hinaus.

Cron startet Skripte und Programme zu vorgegebenen Zeiten. Der auszuführende Befehl wird in einer Tabelle, der sog. crontab, gespeichert. Jeder Benutzer des Systems darf eine solche crontab anlegen.

Diese Tabelle besteht aus sechs Spalten; Die ersten fünf dienen der Zeitangabe (Minute, Stunde, Tag, Monat, Wochentage), die letzte enthält den Befehl. Die einzelnen Spalten werden durch Leerzeichen oder Tabulatoren getrennt.

Häufig führt der Cron-Daemon wichtige Programme für die Instandhaltung des Systems aus, wie zum Beispiel Dienste für das regelmäßige Archivieren und Löschen von Logdateien.

Beim Hochfahren eines Linux Systems wird als Erstes cron gestartet. cron wiederum startet einen Shell Prozess. Ähnlich wird beim Hochfahren eines z/OS Systems JES als erstes gestartet. Das Hochfahren selbst bewirkt ein weiterer Prozess, der Master Scheduler.

Griechisch chronos (χρόνος) bedeutet Zeit.

Job Scheduler

Ein reguläres Windows oder Linux Betriebssystem enthält nur primitive Job Scheduling Funktionen wie z.B. cron. Zum Füllen dieser Marktlücke stellen eine ganze Reihe von Software Unternehmen Job Scheduler für Apple, Windows, Unix und Linux Plattformen her. Beispiele sind:

- cosbatch der Firma OSM <http://www.cosbatch.com/>
- Dollar Universe der Firma Orsyp, <http://www.orsyp.com/en.html>
- OpenPBS (Portable Batch System),
open source package ursprünglich von der NASA entwickelt <http://www.openpbs.org>,
- PBS Pro der Firma PBS Grid Works, <http://www.pbsgridworks.com>,
- enhanced commercial version von OpenPBS.
- Global Event Control Server der Firma Vinzant Software, <http://www.vinzantsoftware.com/>
- ActiveBatch der Firma Advanced–System Concepts, <http://www.advsyscon.com/>

All diese Produkte erreichen bei weitem nicht den Funktionsumfang von z/OS JES

Job Entry Subsystem (JES)

In großen Unternehmen ist die Stapelverarbeitung (neben der interaktiven Verarbeitung) eine wichtige Aufgabe. Ein Job Entry Subsystem (JES) hat die Aufgabe, Stapelverarbeitungsvorgänge zu automatisieren. Diese Funktion ist in den meisten Windows- und Unix-Betriebssystemen nur sehr rudimentär vorhanden. Job Control-Subsysteme werden für Windows- und Unix-Betriebssysteme von unabhängigen Herstellern angeboten.

Das z/OS-Betriebssystem stellt eine spezielle Systemkomponente zur Verfügung, die für die Steuerung und Ablaufkontrolle aller Jobs (einschließlich aller TSO-Sitzungen) zuständig ist. Diese Systemkomponente hat den Namen **Job Entry Subsystem (JES)**. Aus historischen Gründen existieren davon zwei Varianten: JES2 und JES3, die sich in ihrer Funktion jedoch sehr ähnlich sind.

Beide Varianten haben die gleiche Hauptaufgabe, und sind in der Lage, die Jobverarbeitung auf einem einzelnen Rechner zu steuern. Der wesentliche Unterschied zwischen JES2 und JES3 liegt in der Steuerung einer System-Konfiguration, die aus mehreren Rechnern besteht. Ein einzelner Rechner wird in der Regel mit JES2 gesteuert.

Werden mehrere Mainframe Rechner in einem (als Sysplex bezeichneten) Verbund (Cluster) betrieben, erfolgt die Steuerung häufig durch JES3. Mehrere Rechner arbeiten gemeinsam, und jeder Job wird von JES3 je nach Auslastung einem der Rechner zur Verarbeitung zugewiesen.

z/OS Stapelverarbeitung

Enterprise Batch Processing erfordert Sophistication und eine umfangreiche Funktionalität, die auf Windows, Unix und Linux Servern nicht verfügbar ist:

- Zeitabhängige Ausführung
- Beziehungen von Jobs bewahren
- Administrative und Controlling Funktionen
- Einrichtungen für die Steuerung von System Ressourcen
- Effektive Multiprogrammierung und Multiprocessing
- Accounting Einrichtungen
- Umgebung für die Stapelverarbeitung (Batch Execution)

Können Sie sich vorstellen, in einem Unix oder Windows System mehr als 10 Batch Jobs gleichzeitig auszuführen und zu steuern ?

z/OS kann in einem (als Parallel Sysplex bezeichneten) Cluster zehntausende von Jobs gleichzeitig ausführen, verwalten und steuern .

Die Übergabe eines neuen Jobs an das Betriebssystem, das Queuing von Jobs, die Prioritäts- und Ablaufsteuerung zahlreicher Jobs untereinander, Startzeit und Wiederholfrequenz, Durchsatzoptimierung sowie die Zuordnung oder Sperrung von Ressourcen ist die Aufgabe eines Job Steuerungssystems.

Aufgaben der z/OS Stapelverarbeitung

z/OS Jobs können eine Verarbeitungsdauer von Sekundenbruchteilen, Minuten, Stunden und evtl. Tagen haben.

Ein Job wird submitted, es erfolgt eine lange Serie von Berechnungen, unterbrochen durch zahlreiche Ein/Ausgabe (I/O) Anforderungen, Ergebnisse der Berechnungen werden in eine Datei geschrieben, und der Job wird beendet (terminated) mit einem Return Code (RC), der die erfolgreiche oder nicht erfolgreiche Beendigung der Verarbeitung anzeigt.

Beispiele für z/OS Jobs sind:

- Complex Data Base Queries
- Cheque and account processing, e.g. debits, credits, loans, credit rating
- Data Backups, Compression, Conversion
- Offline ATM (Automatic Teller Machine) processing
- System and Database maintenance
- Data replication and synchronization
- Processing exchanged B2B (Business to Business) data
- Tape processing
- Printing
- File system maintenance, healthchecks, compression...
- Configuration Jobs (Admin, RACF)

Job Ausführung

Die Übergabe (Submission) eines Stapelverarbeitungs-Jobs an das Betriebssystem erfolgt durch einen Benutzer; bei einem Großrechner durch einen spezifisch hierfür abgestellten Bediener (Operator). Unter z/OS wird die Job Submission mit Hilfe des Job Entry Subsystems (JES) automatisiert.

Voraussetzung für die Ausführung eines Jobs ist, dass benutzte Programme und Dateien (Data Sets, Files) bereits existieren.

Die Erstellung und Eingabe (Submission) eines Jobs erfordert die folgenden Schritte:

1. Zuordnung einer Datei, welche das Job Control Programm (JCL Script) enthalten soll.
2. Editieren und Abspeichern der JCL Datei
3. Submission (JES übernimmt die Ausführung des Jobs).

Viele Jobs werden in einem Unternehmen immer wieder ausgeführt. Ein Beispiel ist die monatliche Lohn- und Gehaltsabrechnung und Geldüberweisung für alle Mitarbeiter eines Unternehmens. Hier wird jeden Monat das gleiche JCL Script ausgeführt. Eine „**Cataloged Procedure**“ ist ein JCL Programm, welches vom Benutzer für eine spätere Verwendung zwischengespeichert wird (z.B. in einer vom Benutzer erstellten Library JCLLIB) und bei Bedarf mittels eines JCL Befehls aufgerufen wird.

Das JCL Programm gibt die verwendeten Dateien an. In dem Beispiel der Gehaltsabrechnung sind die sich ändernden Gehaltslisten zusätzlich zu sich nur selten ändernden Dateien wie Personaldaten. In dem JCL Script werden die verwendeten Dateien mittels von DD Statements angegeben. Damit ist ein „late Binding“ der verwendeten Dateien an die auszuführenden Programme möglich.

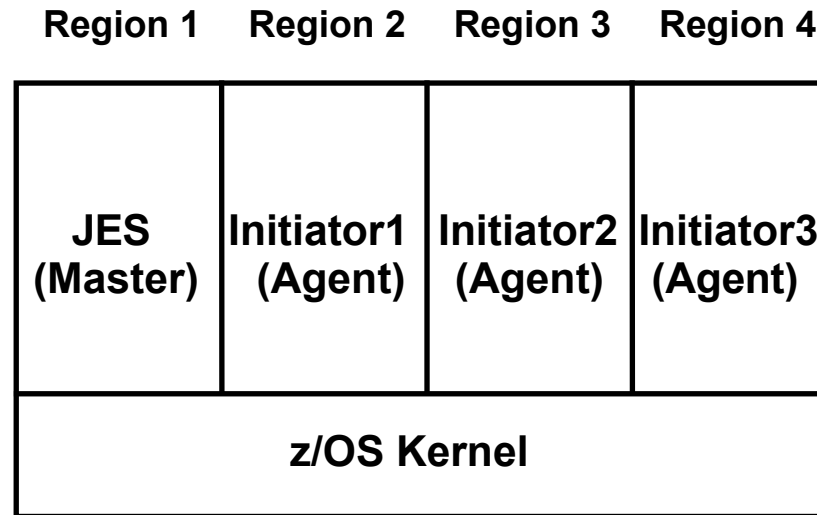
Job Scheduler, Job Entry Subsystem

JES arbeitet nach dem Master/Agent-Konzept. Der Master (entweder JES2 oder JES3)

- initiiert die Jobs,
- ordnet zu Anfang der Bearbeitung die Eingabe und Ausgabe Ressourcen zu
- reiht sie je nach Priorität in Warteschlangen ein
- übernimmt während der Verarbeitung die Zuordnung von Ressourcen wie CPU's und Hauptspeicher und
- kontrolliert den Ablauf der Jobs.
- Nach Abschluß der Bearbeitung übernimmt JES die Freigabe der Ressourcen. Sie werden damit für andere Jobs verfügbar.

Die Agenten werden unter z/OS als Initiator bezeichnet. Dies sind selbständige Prozesse, die getrennt von JES in eigenen virtuellen Adressräumen (Regions) laufen. Sie

- starten die Anwendungsprogramme des Jobs und
- senden Statusinformationen an die JES Steuerungskomponente, und/oder
- fertigen Laufzeitberichte an.



Initiator

Zur Verbesserung der Auslastung unterhält ein z/OS System typischerweise mehrere Initiators. Ein Initiator unterhält eine Input Warteschlange, an die JES einen neuen Job übergibt. Der Initiator fertigt die Jobs in seiner Input Warteschlange unter Prioritätsgesichtspunkten ab. Jeder Initiator verarbeitet in jedem Augenblick nur einen einzigen Job, aber mehrere Initiatoren können mehrere Jobs gleichzeitig verarbeiten.

Jeder Initiator belegt einen eigenen virtuellen Adressenraum, in dem er selbst und ein von ihm gerade bearbeiteter Job untergebracht ist. Das z/OS Stapelverarbeitungs-Subsystem belegt somit eine Reihe von Adressräumen, einen für JES, und weitere für mehrere Initiators.

Wenn ein Job abgeschlossen wird, wird der Initiator des Adressraums aktiv und JES sendet den nächsten Job, der darauf wartet, verarbeitet zu werden.

JES2 kann bis zu 999 Initiators in 999 Address Spaces verwalten.

Batch Processing Beispiel

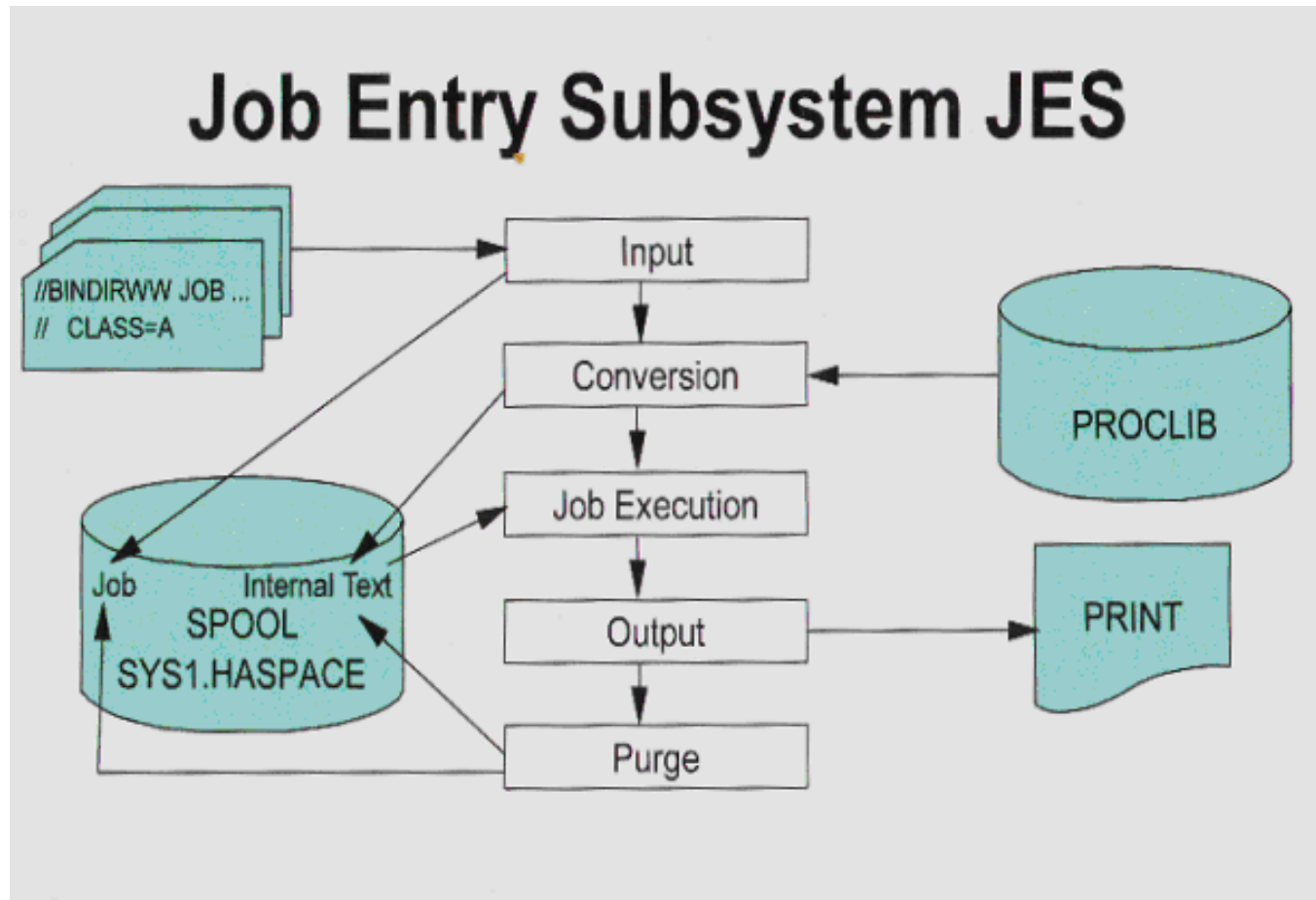
Ein Job Scheduler steuert die zeitliche Ablauffolge einer größeren Anzahl von Jobs. Er dient als “single point of control” für die Definition und das Monitoring der Background Ausführung für einen einzelnen oder eine Gruppe (Cluster) von z/OS Mainframes.

Eine Stapelverarbeitung erfolgt häufig in mehreren Schritten (Job Steps). Der Initiator steuert die Ablauffolge der einzelnen Job Steps.

Beispiel: Monatliche Kreditabrechnung für 300 000 Kundenkonten in einer Bank. Diese könnte z.B. aus den folgenden Schritten (Job-Steps) bestehen:

1. Darlehnskonto abrechnen, Saldo um Tilgungsrate verändern
2. Tilgung und Zinsen im laufenden Konto (Kontokorrent) auf der Sollseite buchen
3. Globales Limit überprüfen
4. Bilanzpositionen (Konten)
5. G+V Positionen (Gewinn- und Verlust Konten)
6. Zinsabgrenzung monatlich für jährliche Zinszahlung
7. Bankmeldewesen (ein Kunde nimmt je € 90 000.- bei 10 Banken auf, läuft am Stichtag)

Verarbeitungsablauf



Die Ausführung eines Jobs durch einen Initiator erfolgt in 5 Schritten: Input, Conversion, Execution, Output und Purge. Spool Files sind temporäre Dateien, die nach Abschluss der Verarbeitung des Jobs nicht mehr gebraucht werden.

INPUT

Ein Job wird über den Reader (Internal Reader) eingelesen und auf dem Spool.Datenträger abgelegt. Er gelangt in die JES Input Queue und erhält eine Jobnummer.

CONVERSION

Die JCL wird im Falle von Prozeduren ergänzt und auf Gültigkeit überprüft. Es wird ein so genannter Internal Text erstellt, der als Kontrollblockstruktur die Batch Verarbeitung gegenüber dem JES repräsentiert.

EXECUTION

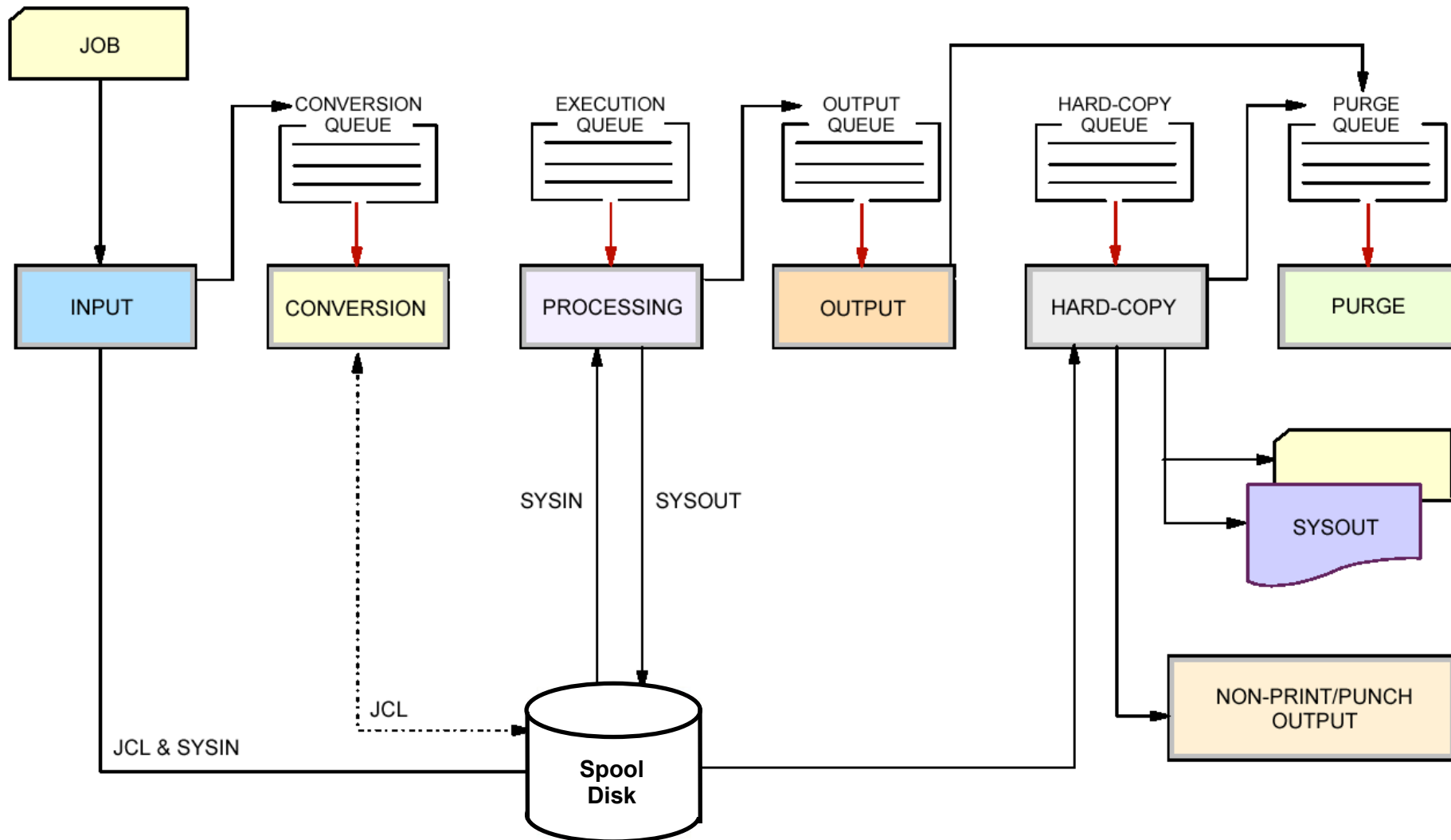
Nach einem bestimmten Algorithmus und auf der Basis von Prioritäten wird ein Job zur Ausführung ausgewählt. Die Kontrollblöcke des ausgewählten Jobs werden dazu einem freien Initiator übergeben.

OUTPUT

Die vom Programm erstellten Listen werden in der JES Output Queue auf dem Spool.Datenträger gehalten, bis ein freier (heute meist virtueller) Drucker bereit ist, die Listen auszudrucken, oder bis der User/Operator den Output löscht (cancelt).

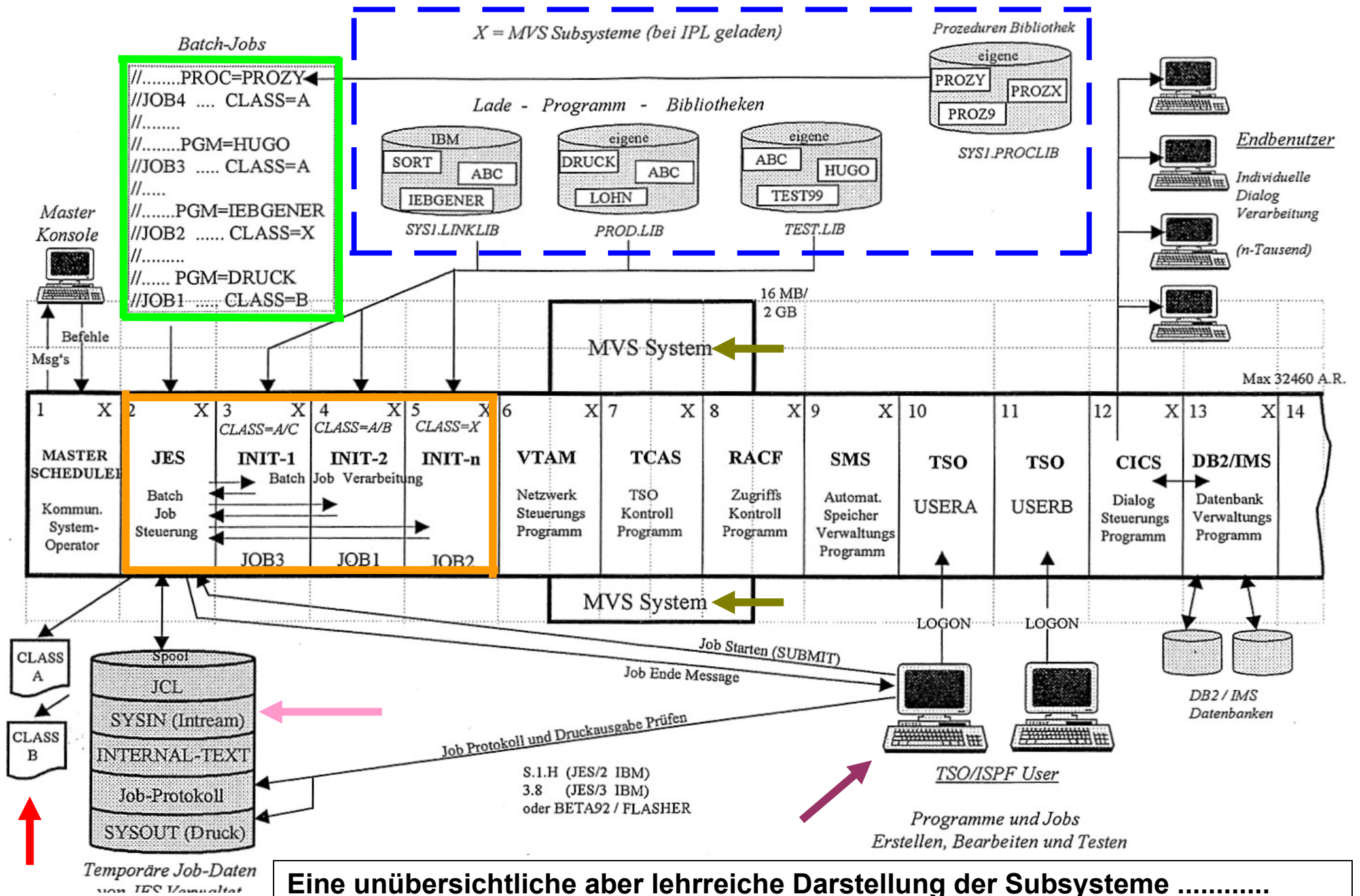
PURGE

Erst wenn der gesamte Output auf einen Drucker ausgegeben wurde, werden die von einem Batchjob belegten Bereiche auf dem Spool.Datenträger gelöscht.



Zur Speicherung aller Informationen über einen Job (z.B. JCL-Records oder Ausgabe-Daten) wird ein dem JES gehörender Dataset, der sogenannte SPOOL-Dataset, benutzt in dem folgende Daten gespeichert werden;

- Original-JCL und ihre Interpretation durch JES,
- Uhrzeit, wann der Job die einzelnen Phasen oder Steps durchlaufen hat,
- Return-Codes für die einzelnen Steps,
- Benutzung aller Systemkomponenten, z.B. Rechenzeit, Speicherplatz, Zahl der Zugriffe auf Platten und Bänder.










Eine unübersichtliche aber lehrreiche Darstellung der Subsysteme

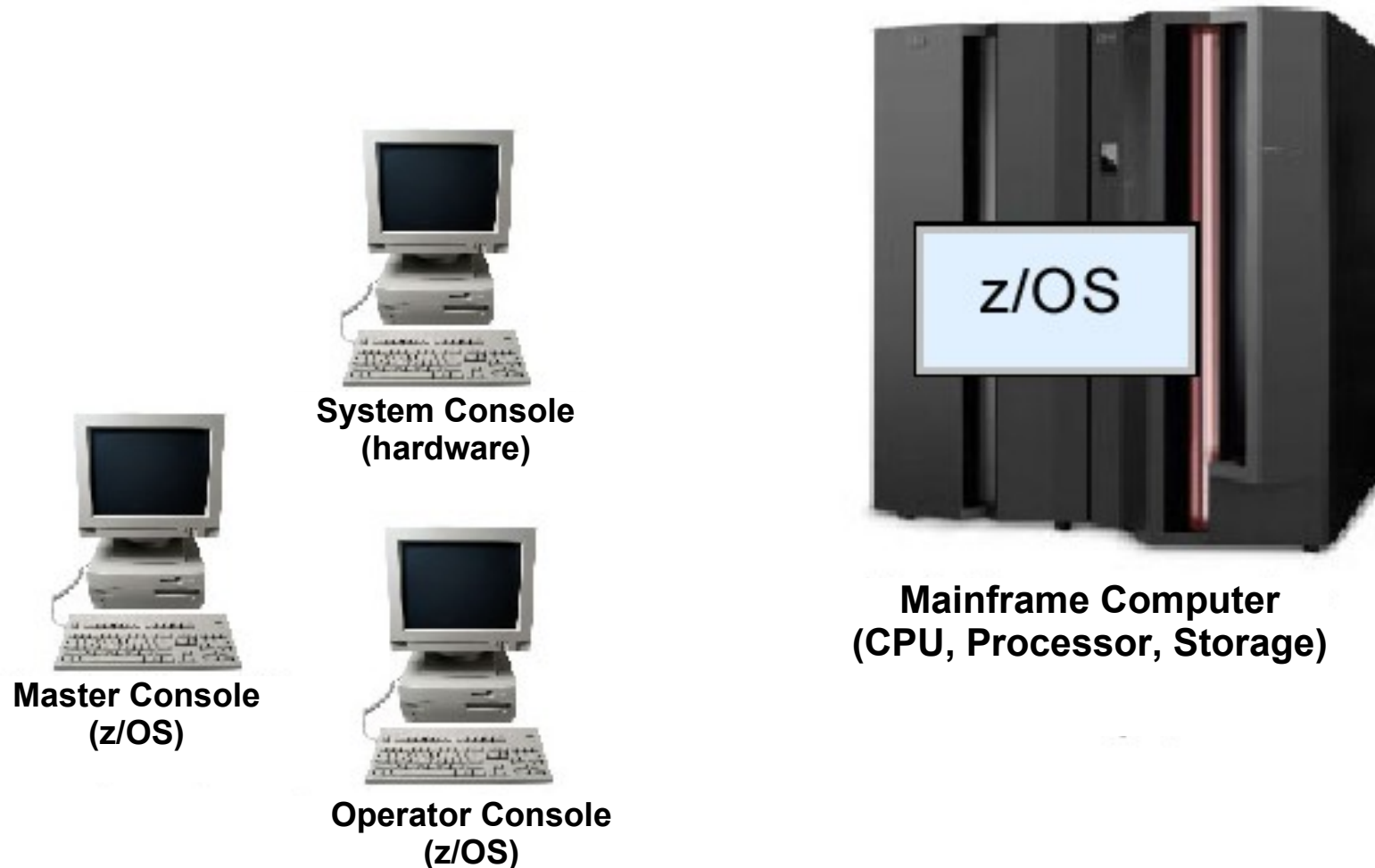
Eine unübersichtliche aber lehrreiche Darstellung der Subsysteme

Die obige Abbildung zeigt als Beispiel ein z/OS System mit einer größeren Anzahl von Subsystemen, die jeweils in eigenen virtuellen Adressräumen (Regions) laufen.

JES holt sich ein Script aus seiner Procedure Bibliothek. Die Initiators laden Programme aus Programmbibliotheken.

-  An Stelle der korrekten Bezeichnung z/OS wird der noch immer geläufige Begriff MVS benutzt.
-  Das Stapelverarbeitungssystem besteht aus einer JES Region und drei Initiator Regions.
-  JES steuert derzeit 4 Batch Jobs mit den Namen PROZY, HUGO, IEBGENER und Druck.
-  Die 4 JOBS werden entsprechend ihrer Prioritätsklassifizierung A, B und X von unterschiedlichen Initiators ausgeführt.
-  Das Programm PROZY ist in einer System-Library SYS1.PROCLIB untergebracht. IEBGENER befindet sich ebenfalls in einer Systemlibrary SYS1.LINKLIB. HUGO befindet sich in einer vom Benutzer angelegten Library. TEST.LIB. DRUCK schließlich befindet sich in einer ebenfalls vom Benutzer angelegten Library PROD.LIB.
-  Eine Spool File enthält temporäre Daten, die nach Abschluss der Verarbeitung wieder gelöscht werden können.
-  Ein Interaktiver Benutzer kann mittels seines TSO Terminals (siehe unten) einen neuen Job starten. Sie werden dies in Ihren praktischen Übungen auf unserem Mainframe Rechner ausprobieren.

Master Konsole und Master Scheduler



Die System Console steuert die Hardware des Rechners. Für die Steuerung des z/OS Betriebssystems sind die Master Console und die Operator Console vorgesehen.

Wie oft starten Sie das Betriebssystem für Ihren PC oder Notebook? Wahrscheinlich mindestens einmal am Tag, vielleicht öfter, wenn Sie Upgrades oder Sicherheits-Patches installieren möchten. Das Starten von z/OS ist ein ähnlicher Prozess, geschieht aber weit weniger häufig, und nur mit viel sorgfältiger vorheriger Planung.

z/OS Initialisierung, als Initial Program Load (IPL) bezeichnet, lädt eine Kopie des Betriebssystems von der Festplatte in den Hauptspeicher, und führt es aus. Dieses Verfahren besteht im wesentlichen aus:

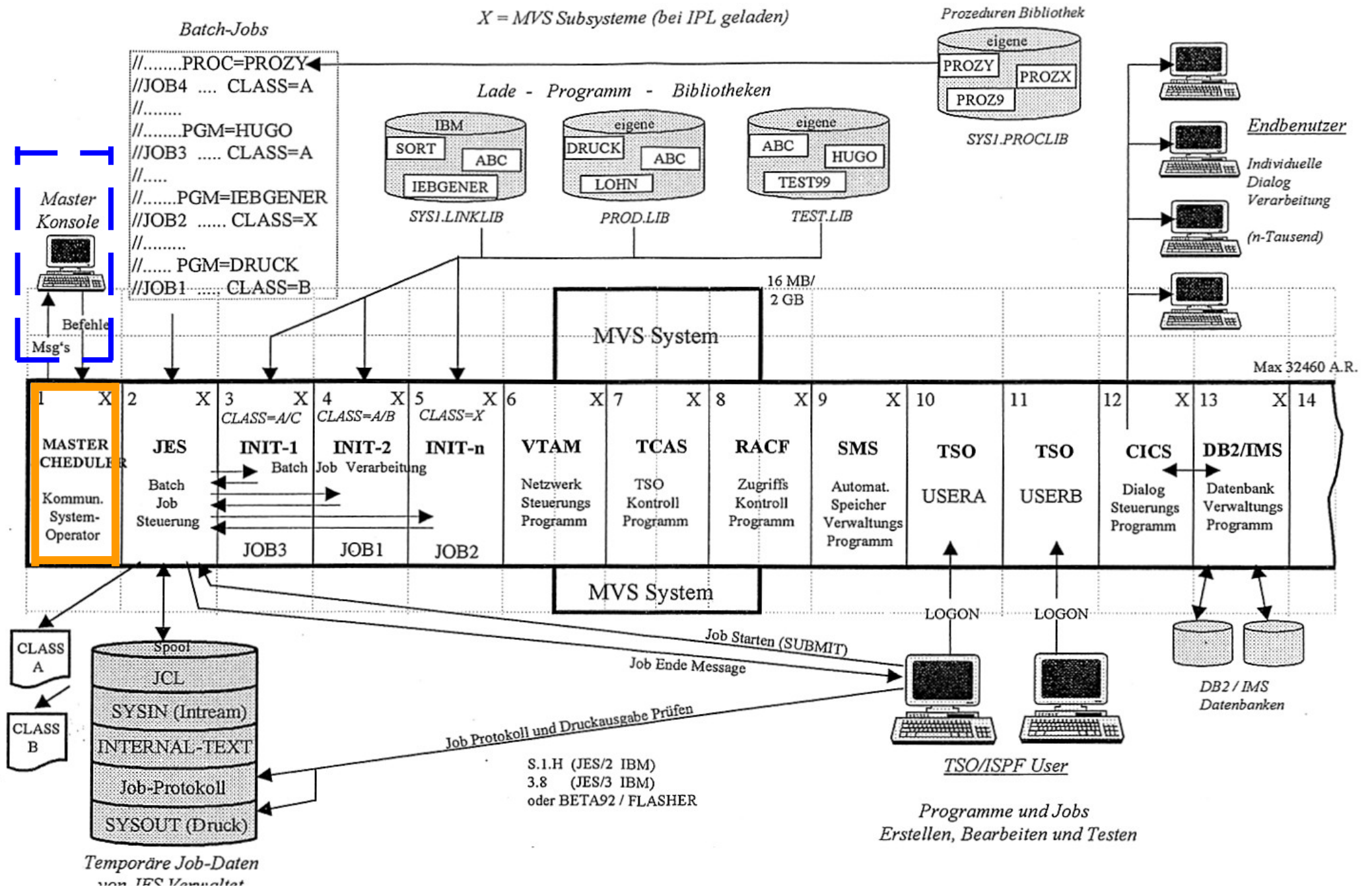
- System-und Storage-Initialisierung, einschließlich der Schaffung von Adressräumen (Regions) für Systemkomponenten
- Master-Scheduler Initialisierung und Subsystem Initialisierung

z / OS-Systeme wurden entwickelt, kontinuierlich zu laufen. Viele Monaten können vergehen, ehe das Betriebssystem neu gestartet wird. Änderungen sind der häufigste Grund, das Betriebssystem neu zu starten, zum Beispiel:

- Ein Testsystem kann täglich oder noch öfter IPLed werden.
- Ein Hochverfügbarkeits-Banking-System darf nur einmal im Jahr neu geladen werden, oder auch weniger häufig, um die Software zu aktualisieren.
- Äußere Einflüsse können oft die Ursache für IPLs sein, z.B. die Notwendigkeit, Stromversorgungs-Systeme in dem Maschinenraum zu testen und zu pflegen.
- Manchmal verbraucht fehlerhafte Software Systemressourcen, die nur durch eine IPL aufgefüllt werden können. Diese Art von Verhalten ist in der Regel Gegenstand einer Untersuchung und Korrektur.

Viele der Änderungen, die in der Vergangenheit ein IPL benötigten, können heute dynamisch erfolgen.

Das Herunterfahren von z/OS passiert so selten wie ein IPL.



Master Konsole und Master Scheduler

Wenn Sie beim Booten Ihres PC eine spezielle Taste drücken, kommen Sie in das BIOS Menue. Hier können Sie Kommandos absetzen, um z. B. ein Booten von einer CD zu ermöglichen.

Unter z/OS ist diese Funktionalität nicht nur beim Hochfahren des Rechners, sondern auch während des laufenden Betriebs in der Form der „Master Konsole“ vorhanden. Im Vergleich zum BIOS sind dabei wesentlich erweiterte Funktionen vorhanden.

- Mit Hilfe der z/OS Master Konsole kann ein System Administrator (Operator) den Systemzustand beobachten, und bei Bedarf in das laufende System eingreifen..
- Die Steuerung der Master Console übernimmt die z/OS Komponente **Master Scheduler**. Dieser ist auch beim Hochfahren des Betriebssystems aktiv. Der Master Scheduler etabliert die Kommunikation zwischen dem Betriebssystem und dem Job Entry Subsystem.

Neben der z/OS Master Konsole existieren noch weitere Konsolen, z.B. eine Hardware Konsole nur für die Steuerung der Hardware. In modernen z/OS Versionen können diese Funktionen auf einer einzigen physischen Konsole vereinheitlicht werden.

z/OS Betriebssystem Teil 4

Time Sharing Option (TSO)

Server Zugriff

Unterschied zwischen Einzelplatzrechner und Client/Server Betriebssystemen.

Wir unterscheiden zwischen Arbeitsplatzrechnern (Klienten) und Servern. Arbeitsplatzrechner arbeiten im **Single-User-Modus** und können mit einem **Single-User-Betriebssystem** betrieben werden. Windows und CMS sind typische Single-User-Betriebssysteme. Windows fehlt die Benutzerverwaltung für den simultanen Multi-User-Betrieb. Unix und z/OS sind traditionelle Multi-User-Betriebssysteme.

Linux und Unix werden sowohl als Arbeitsplatzrechner- als auch als Server-Betriebssysteme eingesetzt. Windows Server 2008 R2 ist ein Server Betriebssystem, welches teilweise die gleichen Komponenten wie die Windows 7 Produktfamilie benutzt. z/OS ist ein reinrassiges Server-Betriebssystem. Andere Beispiele für Server-Betriebssysteme sind i/OS (OS/400), Tandem Non-Stop und DEC OpenMVS. Unix Betriebssysteme wie Solaris, HP-UX (unter Itanium) sowie AIX werden bevorzugt als Server Betriebssysteme eingesetzt

Bei einem Server-Betriebssystem unterscheiden wir zwischen einem interaktiven zeitscheibengesteuerten und einem *Run-to-Completion* Betrieb. Wenn mehrere Benutzer sich gleichzeitig in einen Unix-Server einloggen, geschieht dies typischerweise im interaktiven, zeitscheibengesteuerten Modus. SQL-Aufrufe und Transaktionen arbeiten in vielen Fällen im *Run-to-Completion* Modus. Run-to-Completion bedeutet den Wegfall der Zeitscheibensteuerung, was unter Performance Gesichtspunkten vorteilhaft sein kann. Hierbei ist es die Aufgabe des Entwicklers, seine Programme so zu schreiben, dass nicht ein bestimmtes Programm alle Ressourcen eines Rechners usurpieren kann.

Client Software

Ein Server Zugriff erfordert spezielle Client Software. Hierfür existieren drei Alternativen:

1. Selbstgeschriebene Anwendungen:

Sockets, RPC, Corba, DCOM, RMI

2. Zeilenorientierte Klienten:

Unix Server
z/OS Server
VMS Server

Telnet, Putty Client, FTP
3270 Client (3270 Emulator)
VT 100 Client

3. Klienten mit graphischer Oberfläche:

Windows Server
WWW Server
SAP R/3 Server
Unix Server
z/OS und OS/390 Server

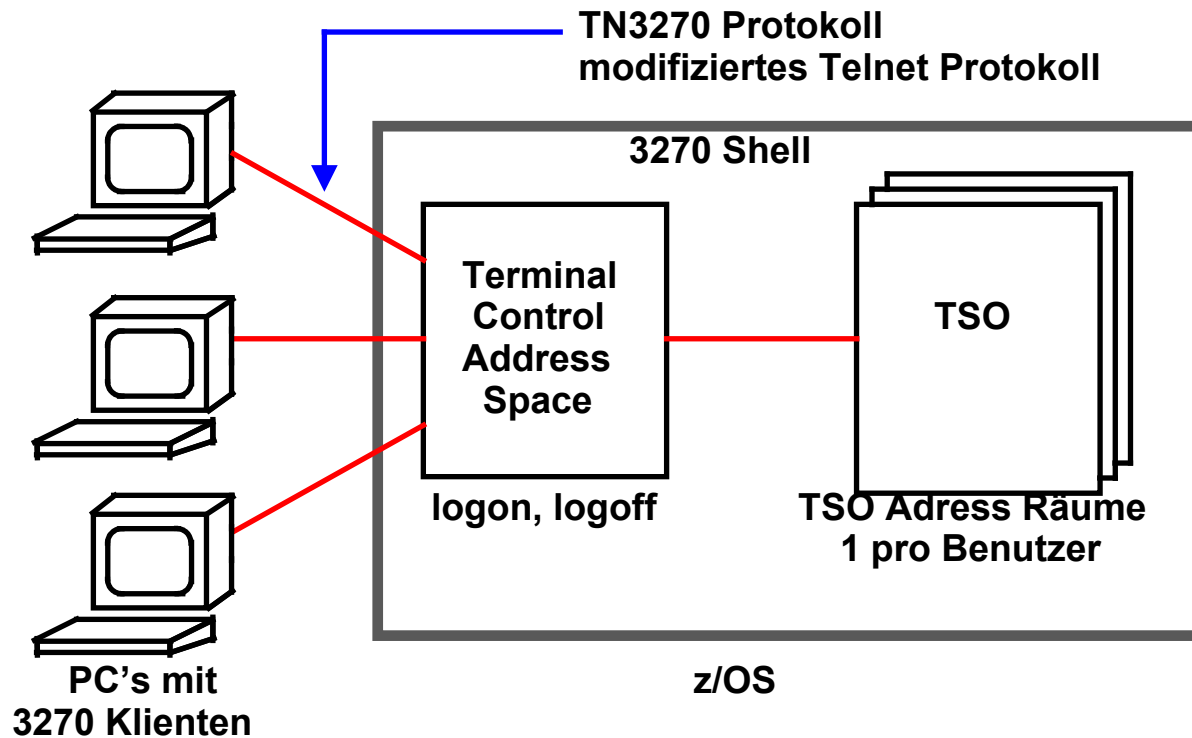
Citrix Client
Browser Client
SAPGUI Client
XWindows, Motif
Servlet, JSP Client



Der 3270 Client, meistens als 3270 Emulator bezeichnet, ist der einfachste Klient für den Zugriff auf einen Mainframe Sever. Unter <http://jedi.informatik.uni-leipzig.de/de/access.html> können Sie einen kostenlosen 3270 Emulator herunterladen.

TSO (Time Sharing Option)

Die TSO Shell ist vergleichbar mit den UNIX Time Sharing Shells (korn, bash,)



Der TSO Terminal Control Prozess (Terminal Control Access Space, TCAS) arbeitet in einem eigenen Adressenraum. Er nimmt Nachrichten von den einzelnen TSO Klienten entgegen und leitet sie an den für den Benutzer eingerichteten separaten TSO Adressenraum weiter.

Für jeden neuen TSO Benutzer wird beim login von TCAS ein eigener (virtueller) Adressenraum eingerichtet.

Es ist eine „Full Screen“ und eine Command Level Schnittstelle verfügbar. Die Full Screen Komponente wird als **Interactive System Productivity Facility (ISPF)** bezeichnet.

TSO (Time Sharing Option)

TSO ist eine z/OS zeilenorientierte Shell, vergleichbar mit der Windows „DOS Eingabeaufforderung“ oder den Unix/Linux Bourne, Korn, Bash oder C-Shells.

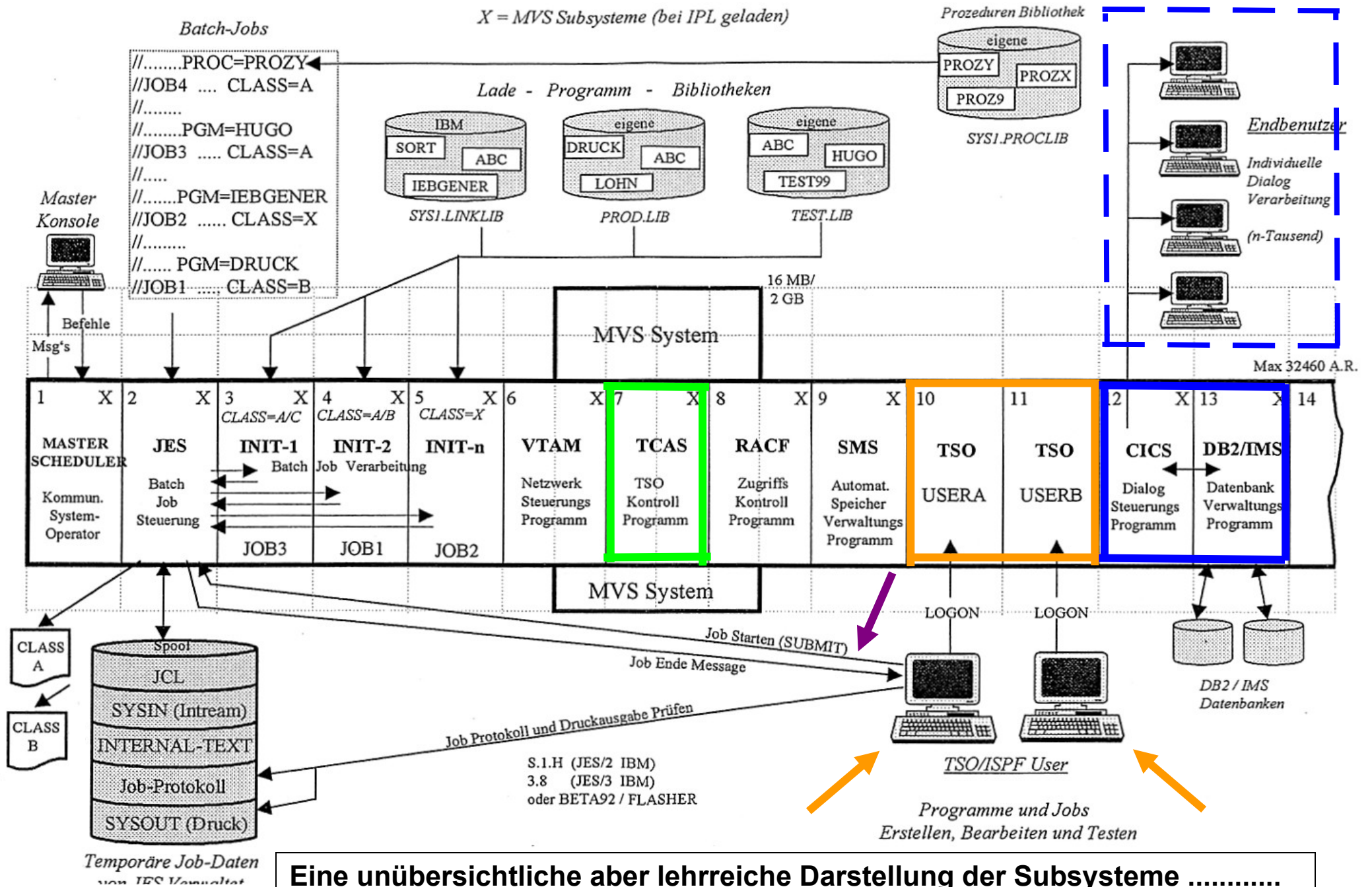
TSO wird hauptsächlich für angewendet für:

- Software Entwicklung und Test,
- System Administration.

Systemadministratoren (Systemprogrammierer) verwenden TSO um Files zu editieren, Steuerungen vorzunehmen, Systemparameter zu setzen und einen Job Status zu überprüfen.







ISPF ist eine wichtige Erweiterung / Ergänzung für die einfache TSO Shell. Hierbei wird auf dem Klienten ein Bildschirm wiedergegeben, der neben der Kommandozeile eine Auflistung der möglichen Kommandos enthält. IBM nennt dies „Full Screen Mode“ und bezeichnet derartige Bildschirminhalte als Panels.

Daneben enthält ISPF einen Bildschirm Editor, den ISPF Editor. Der Editor lässt sich mit dem Unix/Linux vi Editor vergleichen. Beide Editoren sind hoffnungslos inkompatibel, kryptisch, schwer erlernbar, sehr mächtig, und werden von ihren Benutzern heiß und innig geliebt.



Eine unübersichtliche aber lehrreiche Darstellung der Subsysteme

Die obige Abbildung zeigt ein z/OS System mit einer größeren Anzahl von Subsystemen, die jeweils in eigenen virtuellen Adressräumen (Regions) laufen.

-  Der TSO Terminal Control Access Space(TCAS) wird beim Hochfahren von z/OS gestartet.
-  Wenn immer ein Benutzer sich in TSO einlogged, richtet TCAS für ihn einen eigenen TSO Address Space ein.
-  In dem hier gezeigten Beispiel haben sich 2 Benutzer (User A und User B) jeweils mit ihrem eigenen Bildschirm Terminal eingelogged. Jeder der beiden TSO Benutzer hat nur Zugriff zu seinem eigenen Address Space.
-  Der linke der beiden TSO Benutzer übergibt gerade an JES einen Stapelverarbeitungsauftrag.
-  Parallel dazu existieren mehrere Address Spaces für (in diesem Beispiel) eine Kombination der CICS und der DB2 Subsysteme. CICS Programme kommunizieren intern direkt mit DB2 Programmen.
-  Parallel zu den TSO Benutzern haben sich 4 Benutzer mit Ihren Bildschirm Terminals in das CICS Subsystem eingeloggend. Während bei TSO für jeden Benutzer ein eigener Address Space eingerichtet wird, existiert für alle Benutzer (plus dem CICS Subsystem selbst) nur ein einziger Address Space.

z/OS Z18 Level 0609

IP Address = 88.64.138.18

VTAM Terminal = SC0TCP76

Application Developer System

```
          // 0000000 SSSS
        // 00    00 SS
zzzzzz // 00    00 SS
      zz // 00    00 SSSS
     zz // 00    00  SS
    zz  // 00    00  SS
zzzzzz // 0000000 SSSS
```

System Customization - ADCD.Z18.*

==> Enter "LOGON" followed by the TSO userid. Example "LOGON IBMUSER" or

==> Enter L followed by the APPLID

==> Examples: "L TSO", "L CICS", "L IMS3270"

L TSO

Diesen Welcome Screen sehen sie, wenn Sie sich in unseren Mainframe Rechner jedi.informatik.uni-leipzig.de oder 139.18.4.30 einloggen. Von hier aus können Sie sich in eins von mehreren Subsystemen, wie z.B. TSO oder CICS einloggen.

z/OS Z18 Level 0609

IP Address = 92.75.227.134

VTAM Terminal = SC0TCP33

Application Developer System

```
          // 0000000 SSSSS
        //  OO   OO SS
zzzzzz //  OO   OO SS
      zz //  OO   OO SSSS
     zz //  OO   OO  SS
    zz //  OO   OO  SS
zzzzzz //  0000000 SSSS
```

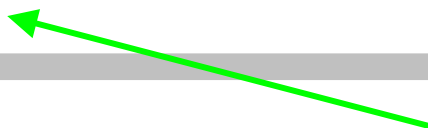
System Customization - ADCD.Z18.*

==> Enter "LOGON" followed by the TSO userid. Example "LOGON IBMUSER" or

==> Enter L followed by the APPLID

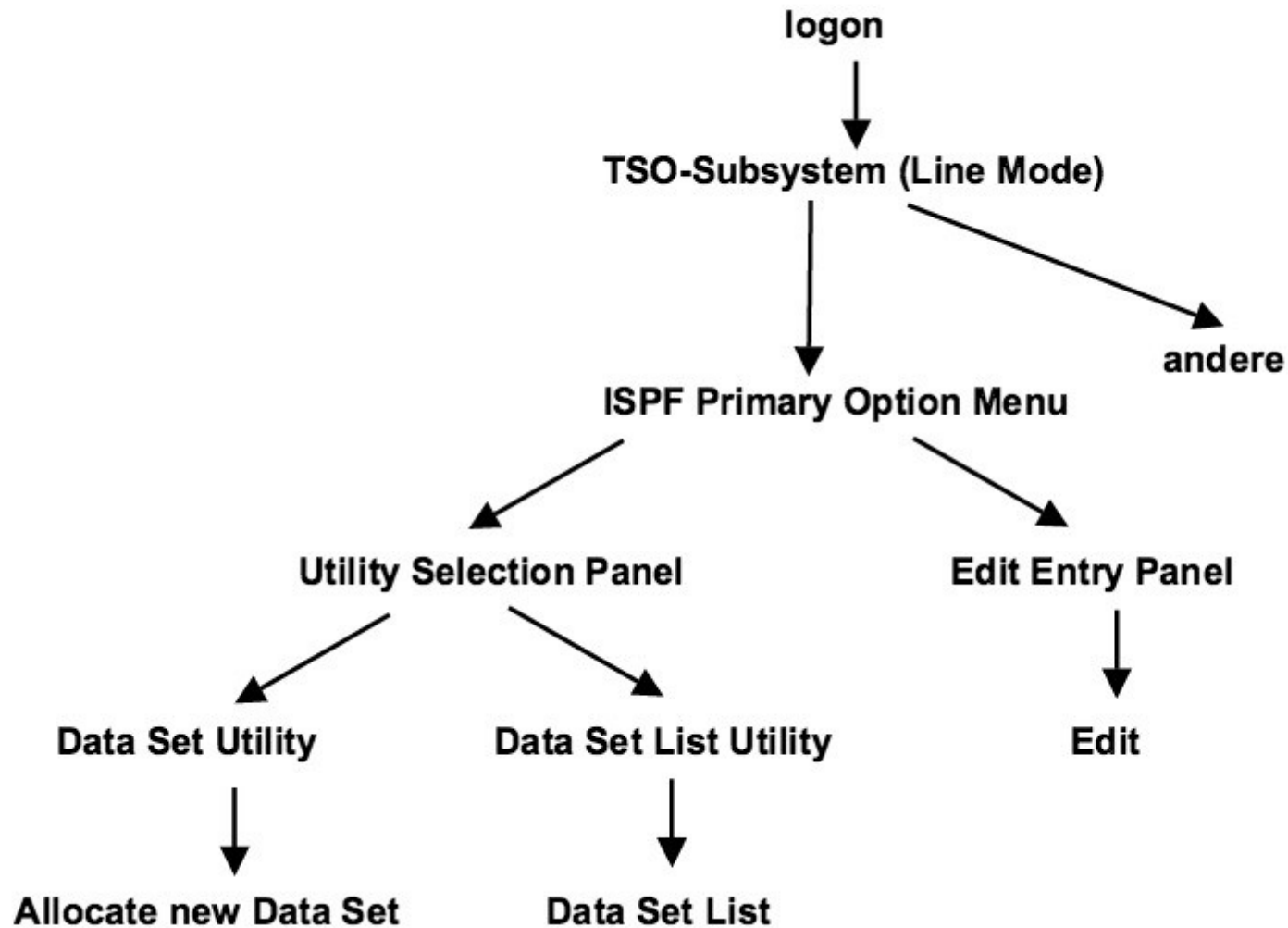
==> Examples: "L TSO", "L CICS", "L IMS3270

L TSO



Hier wird gerade das Logon (abgekürzt L) für das TSO Subsystem eingegeben.

Interactive System Productivity Facility (ISPF)



Menu Utilities Compilers Options Status Help

ISPF Primary Option Menu

End of data

0	Settings	Terminal and user parameters	User ID . : SPRUTH
1	View	Display source data or listings	Time. . . : 18:40
2	Edit	Create or change source data	Terminal. : 3278
3	Utilities	Perform utility functions	Screen. . : 1
4	Foreground	Interactive language processing	Language. : ENGLISH
5	Batch	Submit job for language processing	Appl ID . : ISR
6	Command	Enter TSO or Workstation commands	TSO logon : DBSPROC
7	Dialog Test	Perform dialog testing	TSO prefix: SPRUTH
9	IBM Products	IBM program development products	System ID : ADCD
10	SCLM	SW Configuration Library Manager	MVS acct. : ACCT#
11	Workplace	ISPF Object/Action Workplace	Release . : ISPF 5.8
M	More	Additional IBM Products	

Enter X to Terminate using log/list defaults

Option ==> 2
F1=Help F2=Split F3=Exit F7=Backward F8=Forward F9=Swap
F10=Actions F12=Cancel

==> ist das Cursor Symbol. Eingabe einer „2“ hinter dem Cursor ruft den „ISPF“ Editor auf.

Detaillierte Screen by Screen Tutorials sind zu finden unter <http://jedi.informatik.uni-leipzig.de>

z/OS Betriebssystem Teil 5

Datasets

Dateisystem

Betriebssysteme wie Unix, Linux oder Windows speichern Daten in Files (Dateien). Die Identifizierung von Dateien erfolgt über Dateiverzeichnisse, die selbst wie Dateien aussehen und wie Dateien behandelt werden können.

Ein Dateisystem (File System) verbirgt die Eigenschaften der physischen Datenträger (Plattenspeicher, CD, evtl. Bandlaufwerke) weitestgehend vor dem Anwendungsprogrammierer. Für ein Betriebssystem existieren typischerweise mehrere inkompatible File Systeme, z. B. FAT32 oder NTFS unter Windows, oder extf2, extf3 und Reiser unter Linux.

Unter Unix, Linux und Windows sind Dateien strukturlose Zeichenketten (unstrukturierte Menge von Bytes), welche über Namen identifiziert werden. Hierfür dienen Dateiverzeichnisse, die selbst wie Dateien aussehen und behandelt werden können.

Die meisten z/OS Dateien haben im Gegensatz zu Linux oder Windows Dateien zusätzliche Struktureigenschaften, und werden dann als Data Sets bezeichnet. Viele Data Sets bestehen aus Records. Hier ein Record Beispiel: Angenommen eine Datei, die das Telefonverzeichnis einer Stadt speichert. Die Datei besteht aus vielen Records (Einträgen). Jeder Record besteht aus Feldern, z.B. Nachname, Vorname, Straße, Haus Nr, Tel. Nr. usw.

Im Gegensatz zu einem Unix File speichert eine Unix SQL Datenbank die Daten strukturiert in der Form von Tabellen (Tables), Relationen, Rows, Columns usw. Die Rows innerhalb einer Unix SQL Datenbank Tabelle enthalten Records, im Gegensatz zu Unix Files.

Datenspeicherung unter z/OS

z/OS verwendet gleichzeitig mehrere Filesysteme. Eine z/OS „Access Method“ definiert das benutzte Filesystem und stellt gleichzeitig Routinen für den Zugriff auf die Records des Filesystems zur Verfügung. Das Filesystem wird durch die Formattierung eines physischen Datenträgers definiert. Bei der Formattierung der Festplatte wird die Struktur des Datasets festgelegt. Es gibt unter z/OS unterschiedliche Formattierungen und Zugriffsmethoden für Datasets mit direktem Zugriff, sequentiellen Zugriff oder index-sequentiellen Zugriff.

Genauso wie bei Unix existieren eine ganze Reihe von File Systemen (Access Methods) mit Namen wie BSAM, BDAM, QSAM, ISAM usw. Die wichtigste Access Method (File System) hat den Namen VSAM (Virtual Storage Access Method).

Dataset Zugriffe benutzen an Stelle eines Dateiverzeichnisse „Kontrollblöcke“, welche die Datenbasis für unterschiedliche Betriebssystemfunktionen bilden. Die Kontrollblöcke haben exotische Namen wie VTOC, DSCB, DCB, UCB, deren Inhalt vom Systemprogrammierer oder Anwendungsprogrammierer manipuliert werden können.

Unter z/OS werden Datenbanken ebenfalls in Data Sets abgespeichert. Eine Datenbank ist also eine höhere Abstraktionsebene als ein Data Set. Während DB2 unter z/OS hierfür normale z/OS Data Sets verwendet, benutzen Unix Datenbanken hierfür häufig Files mit proprietären Eigenschaften sowie direkte (raw) Zugriffe auf Dateien.

Blocks, physische und logische Records

In der Anfangszeit von OS/360 wurde bei einem Plattenspeicherzugriff ein einzelner Record zwischen Plattenspeicher und Hauptspeicher transportiert. Dies ist nicht sehr effektiv, weil Plattenspeicherzugriffe relativ lange dauern (häufig mehr als 10ms). Records sind meistens nicht sehr lang; Sie würden überrascht sein, wie populär heute noch eine Recordlänge von 80 Bytes ist (in Anlehnung an die 80 Spalten der IBM Lochkarte).

Man ging dazu über, bei jedem Plattenspeicherzugriff eine Gruppe von nebeneinanderliegenden Records auszulesen, in der Hoffnung, dass das Anwendungsprogramm demnächst einen benachbarten Record brauchen würde, der sich dann schon im Hauptspeicher befand. Besonders bei der sequentiellen Verarbeitung von Datenrecords ist das sehr effektiv.

Man bezeichnete diese Gruppe von gleichzeitig ausgelesenen Records als **Block** oder **physischen Record**, im Gegensatz zum eigentlichen Record, der dann als **logischer Record** bezeichnet wird.

Ein Block (physischer Record) besteht also aus mehreren logischen Records, welche die eigentlichen Verarbeitungseinheiten darstellen. Die Blockungsgröße definiert, wie viele logischen Records in einem physischen Record (Block) untergebracht werden können.

Wenn man heute von Records redet, meint man damit meistens logische Records.

Benutzung einer z/OS Datei

Wenn sie sich das erste Mal unter TSO einloggen ist Ihre allererste Aufgabe die Zuordnung (allocate) von Data Sets.

Ihre Frage sollte sein: Was soll das? „Das habe ich unter Windows noch nie gemacht“.

Dies ist falsch!!! Wenn Sie unter Windows eine neue Datei erstmalig anlegen, müssen Sie z. B. entscheiden, ob sie unter C: oder D: gespeichert wird. Wenn Ihr Rechner über 24 Plattenspeicher verfügt, z.B.

C:, D:, E:, ... Z: ,

wird die Verwaltung schon etwas schwieriger. Bei der Auswahl kann auch sein, dass einige der Platten (welche ?) mit FAT32 und andere mit NTFS formatiert sind, dass sie unterschiedliche Kapazität und/oder Performance Eigenschaften haben, und dass dies für Ihre Entscheidung relevant sein mag. Stellen Sie sich vor, Ihr Rechner hat eine normale Festplatte und zusätzlich eine besonders schnelle, aber kleine, Solid State Disk.

Was machen Sie, wenn Ihr z/OS Rechner über 60 000 Plattenspeicher verfügt, dazu 1500 Solid State Disks ?

Sie verwenden für die Verwaltung eine z/OS Komponente **Storage Management System (SMS)**. Wenn Sie Platz für eine neue Datei brauchen, melden Sie dies bei SMS an. Dieser Vorgang wird als **Allocation** bezeichnet.

Genau genommen besteht das z/OS Storage Management System aus mehreren Komponenten. Die wichtigste dieser Komponenten heißt **Data Facility Storage Management System (DFSMS)**.

Die Menge aller von SMS verwalteten Data Sets wird als „System Managed Storage“ bezeichnet. In der Vergangenheit war es möglich, Data Sets auch ohne Benutzung von SMS zu verwalten. Das ist heute unüblich.

Erstellen einer Datei: Entscheidungen

Benutzer, die Datenverarbeitung betreiben, haben beim Umgang mit den Daten täglich viele Entscheidungen zu treffen. Zusätzlich zum Umgang mit Themen, die nur die Daten oder die Anwendung betreffen, müssen sie die Speicherverwaltungsmaßnahmen der Installationen kennen. Sie müssen sich außerdem mit den Themen auseinandersetzen, die Format, Bearbeitung und Positionierung der Daten betreffen:

- Welchen Wert soll die Blockgröße haben? Wieviel Speicherplatz ist erforderlich?
- Welcher Festplattentyp soll verwendet werden? Sollen die Daten in den Cache geschrieben werden? Soll eine Fehlerbehebung durchgeführt werden?
- Welche Datenträger stehen für die Dateipositionierung zur Verfügung?
- Wie oft soll eine Sicherung oder Migration durchgeführt werden? Soll die Sicherung/Migration erhalten bleiben oder (wann ?) gelöscht werden?

Wenn die Verwendung von Datenverarbeitungsservices vereinfacht werden soll, müssen dem System einfachere Schnittstellen zur Verfügung gestellt werden. Insbesondere JCL ist einer der Bereiche, in denen Vereinfachungen vorgenommen werden.

DFSMS Klassen

Angesichts der unübersichtlich großen Anzahl von Plattenspeichern, File Systemen und Data Sets werden letztere in Klassen eingeteilt. Hierfür ist eine z/OS Komponente zuständig, das **Data Facility Storage Management System (DFSMS)**. Beim Neuanlegen eines Data Sets müssen angegeben werden:

Data Class

Eine Data Class fasst Data Sets mit identischen File System Attributen wie Record Format, Record Länge, Schlüssellänge bei VSAM Dateien, usw zusammen.

Storage Class

Eine Storage Class fasst mehrere Plattenspeicher mit identischen Performance Eigenschaften wie Antwortzeit (ms), Nutzung von Read/Write Caching, Verwendung der Dual Copy Funktion usw. zusammen. Beispielsweise könnte man z.B alle Solid State Disks zu einer Storage Class zusammenfassen

Management Class

Eine Management Class fasst mehrere Plattenspeicher zu einer Gruppe zusammen, wenn diese identisch verwaltet werden. Verwaltungseigenschaften sind z.B. Migration nach x Tagen, Anzahl Backup Versionen, schrittweiser Abbau der Backup Versionen, Löschen der Daten, ...

DFSMS bewirkt die Zuordnung eines Data Sets zu einer Storage Group, einer Gruppe von Plattenspeichern (Volumes) mit der gleichen Data Class, Storage Class und Management Class.

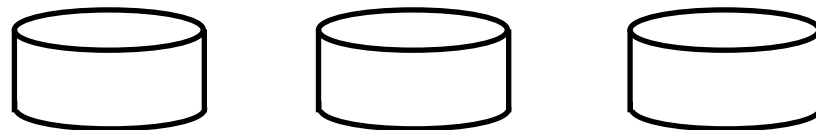
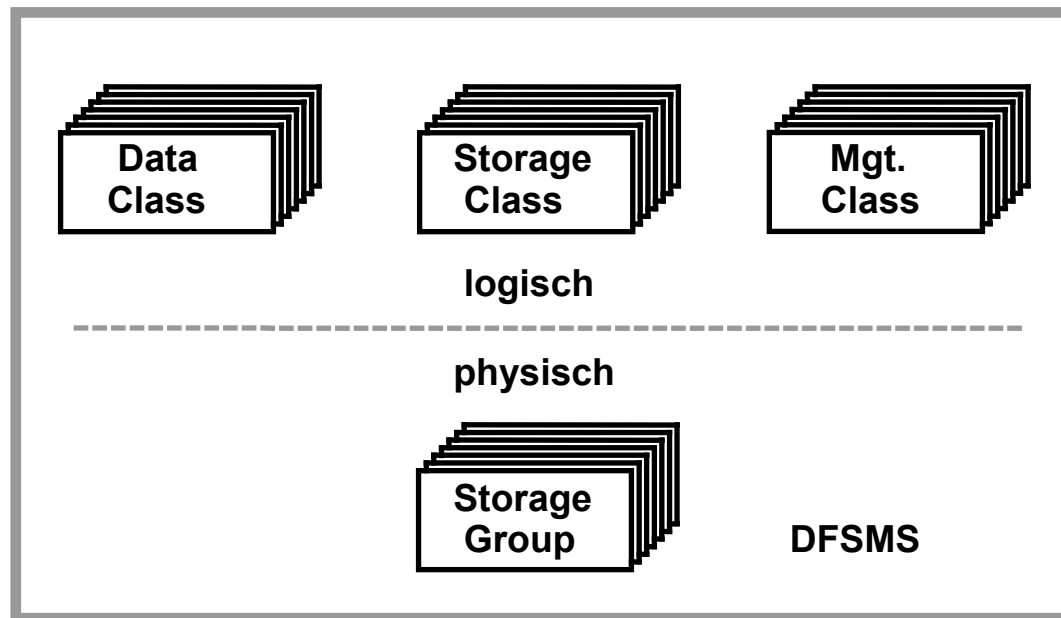
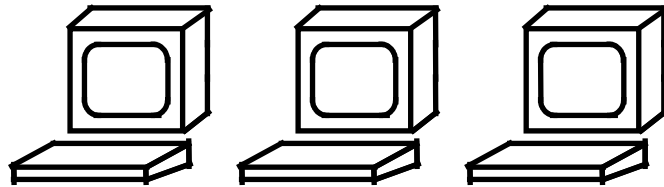
Lebenszyklus eines Data Sets

Alle Dateien und Data Sets in einem Unternehmen haben einen Lebenszyklus: Sie entstehen irgendwann, werden verarbeitet und geändert und werden irgendwann wieder gelöscht. Das Was, Wie und Wo ist in der **Management Class** festgelegt.

Es werden beispielsweise Eigenschaften wie die Folgenden festgelegt:

- Anlegen (allocate) der Datei erfolgt durch den Benutzer
- Benutzung (Schreiben und/oder Lesen der Daten)
- Es werden Sicherungskopien angelegt
- Angaben zur Platzverwaltung (wann wird der Data Set freigegeben / erweitert / komprimiert)
- Auslagern von inaktiven Dateien, sowie Wiederbenutzung
- Ausmustern von Sicherungskopien
- Wiederherstellung von Dateien
- Löschen der Datei

Data Facility Storage Management System



Physische Festplattenspeicher (Volumes)

Der Benutzer sieht nur die logische Sicht der Daten:

- vorbereitete Datenmodelle in den Datenklassen
- in den Storageklassen festgelegte Serviceanforderungen
- Management Kriterien für die Auslagerung der Daten

Festplattenspeicher mit der gleichen Data Class, Storage Class und Management Class werden zu einer Storage Group zusammengefasst.

Anlegen eines neuen Data Set

Die nachfolgende Abbildung zeigt den Bildschirminhalt (Screen), mit dem Sie das Allocating eines Data Sets vornehmen.

In dem gezeigten Beispiel ist die Management Class mit „Default“ vorgegeben, desgleichen die Storage Class mit PRIM90.

Bei der Data Class haben Sie die Auswahl, die gewünschte Größe in Bytes, Kilobytes, Megabytes, Records, Blöcken, Plattenspeicher-Spuren (Tracks) oder Zylindern anzugeben.

Die angegebene Größe ist 16 KByte (Primary quantity) mit einer Reserve von 1 KByte (Secondary Quantity) . Elemente innerhalb des Data Sets können mit 2 Directory Blocks adressiert werden.

Alle Records haben die gleiche Länge (Fixed Block, FB). Die Länge der logischen Records (Record length) beträgt 80 Bytes. Vier logische Records werden zu einem physischen Record zusammengefasst (Block size = 4 x 80 Bytes, = 320 Bytes).

Es existieren eine ganze Reihe von unterschiedlichen File Systemen für z/OS Data Sets. (z/OS bezeichnet die File Systeme als „Access Methods“). Das hier gewählte File System ist PDS (was immer das auch sein mag, wir schauen es uns später an).

Menu RefList Utilities Help

Allocate New Data Set

More: +

Data Set Name . . . : PRAKT20.TEST.DATASET

Management class . . .	DEFAULT	(Blank for default management class)
Storage class . . .	PRIM90	(Blank for default storage class)
Volume serial . . .		(Blank for system default volume) **
Device type . . .		(Generic unit or device address) **
Data class . . .		(Blank for default data class)
Space units . . .	KILOBYTE	(BLKS, TRKS, CYLS, KB, MB, BYTES or RECORDS)
Average record unit		(M, K, or U)
Primary quantity . .	16	(In above units)
Secondary quantity	1	(In above units)
Directory blocks . .	2	(Zero for sequential data set) *
Record format . . .	FB	
Record length . . .	80	
Block size . . .	320	
Data set name type :	PDS	(LIBRARY, HFS, PDS, or blank) * (YY/MM/DD, YYYY/MM/DD)

Command ==>

F1=Help

F3=Exit

F10=Actions

F12=Cancel

z/OS Betriebssystem Teil 6

z/OS Subsysteme

z/OS Subsysteme

Einige der wichtigsten z/OS Subsysteme (häufig auch als Server bezeichnet) sind:

CICS Transaktions Manager

IMS Transaktionsmanager

DB2 Datenbank

IMS Datenbank

JES2 oder JES3

Security Server (RACF, DCE Security, Firewall)

Netview, Systemview

WebSphere

UNIX System Services

Distributed Computing Services (DCE, NFS, DFS, FTP)

Run Time Language Support

C/C++

PL/1

COBOL

Fortran

Object Oriented Cobol

Assembler

C/C++ Open Class Library

Einige dieser Subsysteme werden wir uns näher anschauen.

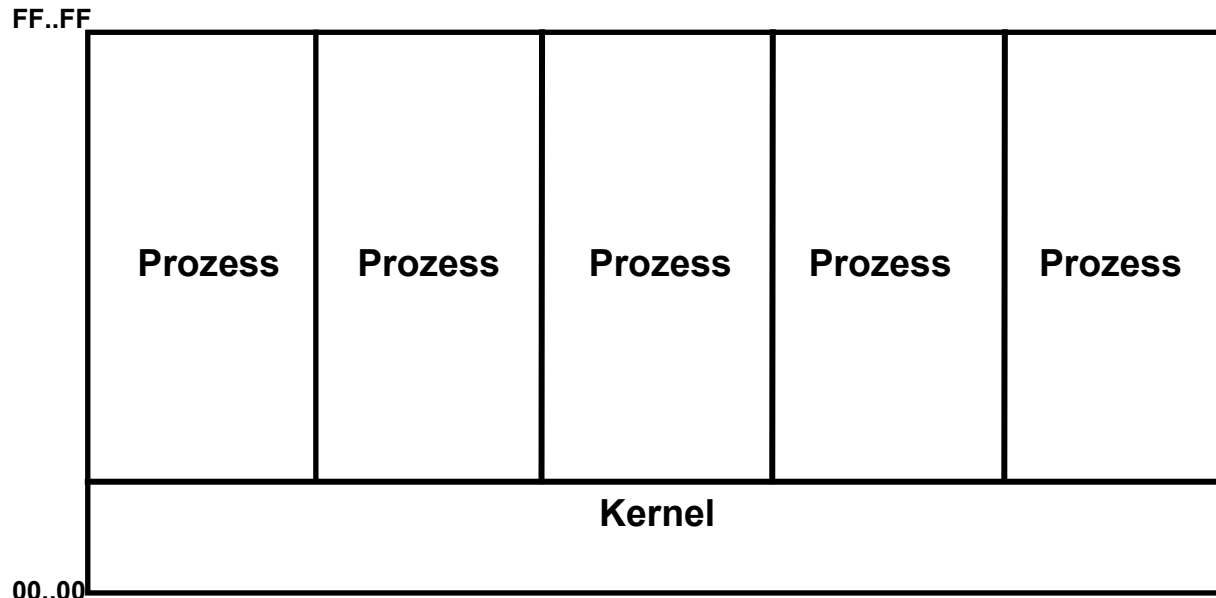
Prozessverwaltung

Datenbanken

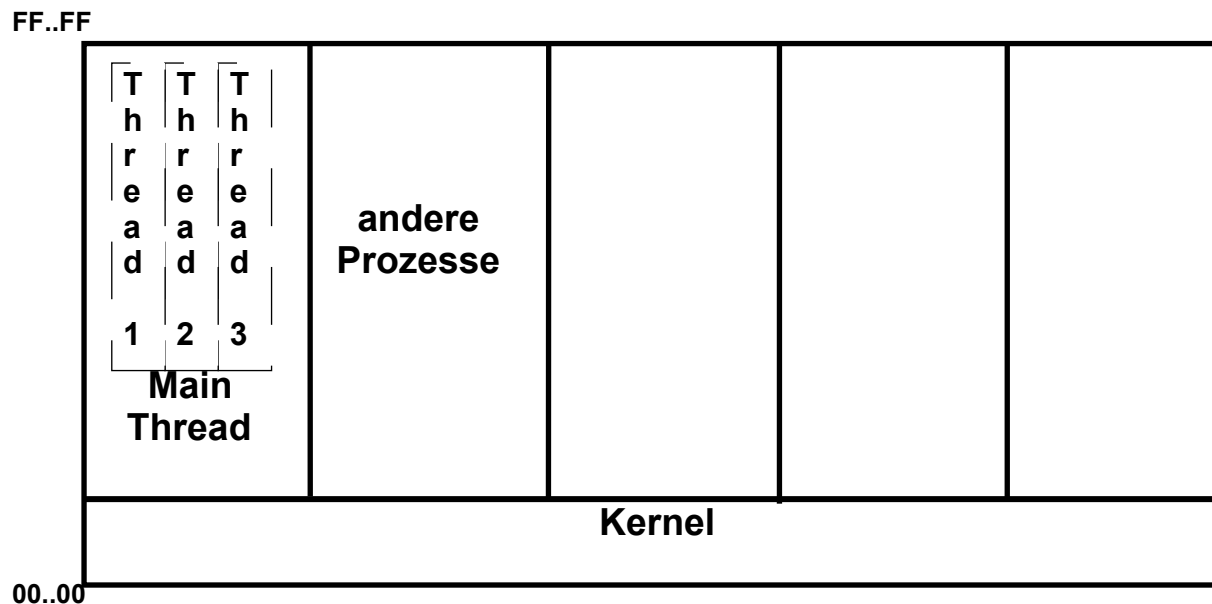
Communication Server

RACF

Language Environment



Prozess Ansatz



Thread Ansatz

z/OS Prozessverwaltung

Bei der Ausführung von Programmen unterscheiden wir zwischen Prozessen und Threads.

Prozesse laufen in getrennten virtuellen Adressenräumen. Threads sind unabhängige Ausführungseinheiten, die innerhalb des gleichen virtuellen Adressenraums ablaufen. Ein Wechsel zwischen Threads erfordert deutlich weniger Aufwand als ein Wechsel zwischen Prozessen.

z/OS Prozessverwaltung

Ein z/OS Prozess besteht aus mehreren Arbeitseinheiten, die als „Tasks“ bezeichnet werden. Eine „**Main Task**“ repräsentiert einen Prozess und typischerweise einen Address Space. Eine Main Task kann Subtasks generieren; eine **Subtask** entspricht in etwa einem Thread in Unix oder Windows. Subtasks laufen im gleichen Address Space wie die Main Task.

Wenn ein Prozess gestartet wird, erstellt z/OS hierfür einen Main **Task Control Block (TCB)**. Ein TCB entspricht in etwa einem Unix **Process Control Block (PCB)**. Das Programm kann weitere Subtasks erstellen mit Hilfe des ATTACH System Aufrufes. Hierbei wird jeweils ein eigener Subtask TCB erzeugt. Da der ATTACH Overhead relativ groß ist, implementieren zeitkritische Subsysteme wie z.B. CICS ihr eigenes Subtasking. Bei der Nutzung der Unix System Services (siehe weiter unten) wird empfohlen, die POSIX Funktion `pthread_create` an Stelle der z/OS ATTACH Makro zu benutzen.

Im Kernelmodus (Supervisor State) existieren zusätzliche Mechanismen, die als **Service Request Blocks (SRB)** bezeichnet werden. SRBs werden benutzt, um Systemroutinen auszuführen. Windows kennt im Kernelmodus einen ähnlichen Mechanismus, der als „**Fibers**“ (light weight threads) bezeichnet wird. Fibers werden durch die Anwendung gescheduled. Die CICS Portierung auf Windows benutzt Fibers.

Task Control Block

Prozesse (und Threads) unter z/OS werden von einem **Task Control Block (TCB)** gesteuert. Unix benutzt die Bezeichnung „Process Control Block“ (PCB).

Ein TCB ist eine Datenstruktur in dem z/OS-Kernel-Adressraum, der die erforderlichen Informationen enthält, um einen bestimmten Prozess zu verwalten. Der TCB ist die Manifestation eines Prozesses in z/OS.

Ein TCB umfasst:

- Den Identifier des Prozesses (Prozess-Identifier, oder PID).
- Registerinhalte für den Prozess (Mehrzweck-, Gleitkomma-, Kontroll-, Steuerregister) einschließlich des Programm-Statuswortes (PSW) für den Prozess.
- Den Adressraum für den Prozess.
- Priorität: Ein Prozess mit höherer Priorität bekommt die erste Präferenz (Gegenstück zur "nice"-Variable in Unix-Betriebssystemen).
- Prozess-Metadaten, wie z.B.: wann wurde der Prozess zuletzt ausgeführt, wie viel CPU-Zeit hat er akkumuliert, usw.
- Zeiger auf den TCB eines Prozesses, den die CPU als nächstes ausführen soll.
- I/O-Information (z.B. I/O-Devices, die dem Prozess zugeordnet wurden, Liste der geöffneten Data Sets, usw.).

TCB and SRB Mode

In einer z/OS-Umgebung kann ein Programm in einem von zwei Modi ausgeführt werden:

- TCB-Modus, in der Regel als Task-Modus bezeichnet
- SRB-Modus (**Service Request-Block** Modus)

Die meisten Programme - und alle im Userstatus laufenden Programme – werden als „Task“ ausgeführt. Jeder Thread der Ausführung wird durch einen „Task Control Block“ (TCB) repräsentiert. Ein Programm kann in mehrere Tasks unterteilt werden, was eine Ausführung auf mehreren Prozessoren ermöglicht. Programme im Task-Modus können I/O ausführen, auf Events warten und alle Arten von Systemdiensten nutzen.

SRBs sind leichtgewichtige und effiziente z/OS-Ausführungs-Threads, **die nur im Supervisor-Status verfügbar sind**. Der SRB-Modus wird von manchen Kernel-Routinen benutzt, um bestimmte Performance-kritische Funktionen auszuführen. Wenn zum Beispiel ein I/O-Interrupt auftritt, sendet z/OS einen SRB zu dem entsprechenden Adressraum, um einen ECB (Event Control Block) zu benachrichtigen, und um möglicherweise andere Verarbeitungsvorgänge auszuführen.

Der SRB-Modus ist außerhalb des Betriebssystems und außerhalb von Middleware-Produkten wie DB2 weitgehend ungenutzt. Ein Grund dafür ist: Der SRB-Modus ist nur im Supervisor-Status verfügbar. Die meisten Programmierer glauben, SRB-Modus-Funktionen sind zu schwierig zu testen und zu debuggen. Als Folge läuft fast aller z/OS-Anwendungs-Code im Task-Modus. Der SRB-Modus wird nur dort eingesetzt, wo er absolut notwendig ist.

Windows hat „Process Control Blocks“ ähnlich zu z/OS TCBs. Das Windows-Äquivalent eines SRB wird als **Fibre** bezeichnet.

Prozessverwaltung

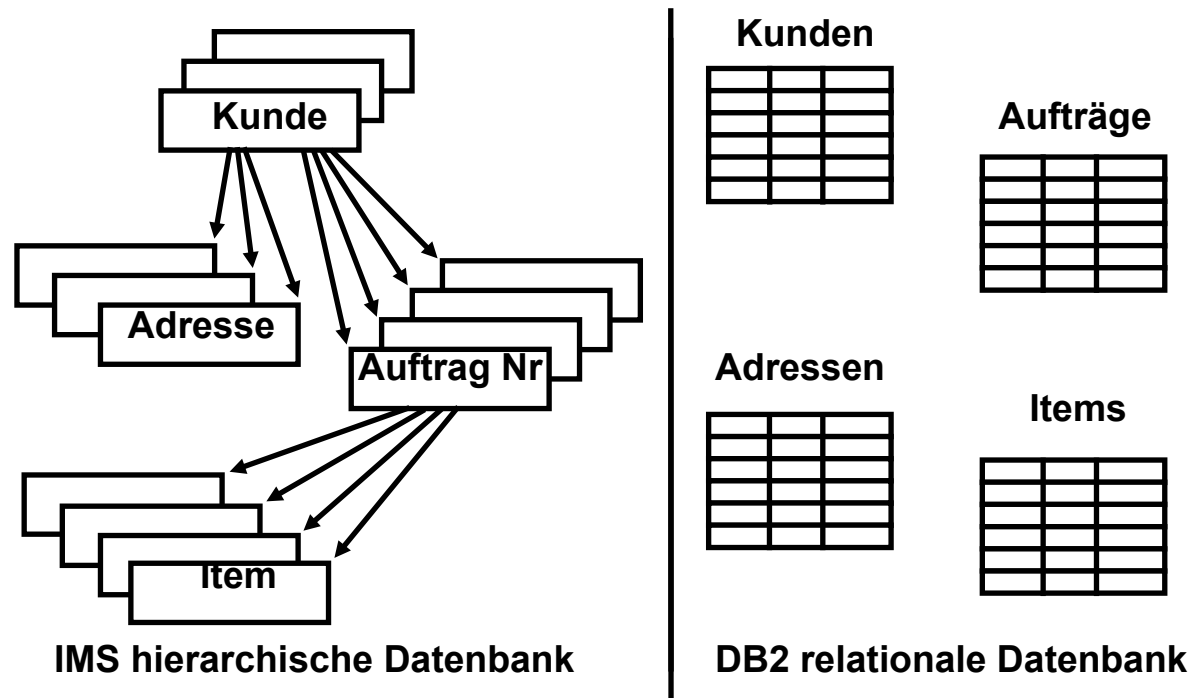
Datenbanken

Communication Server

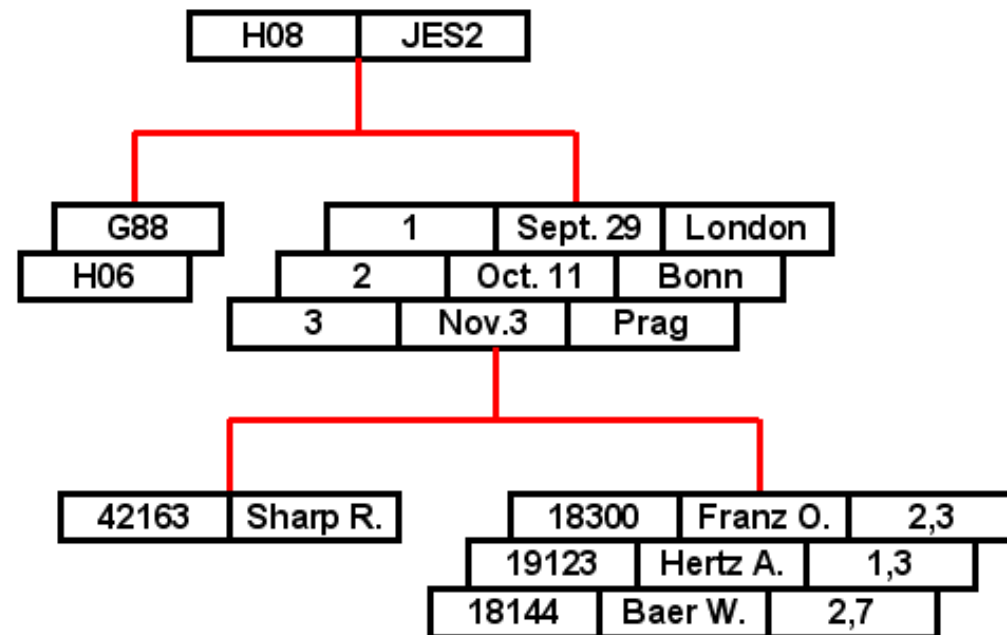
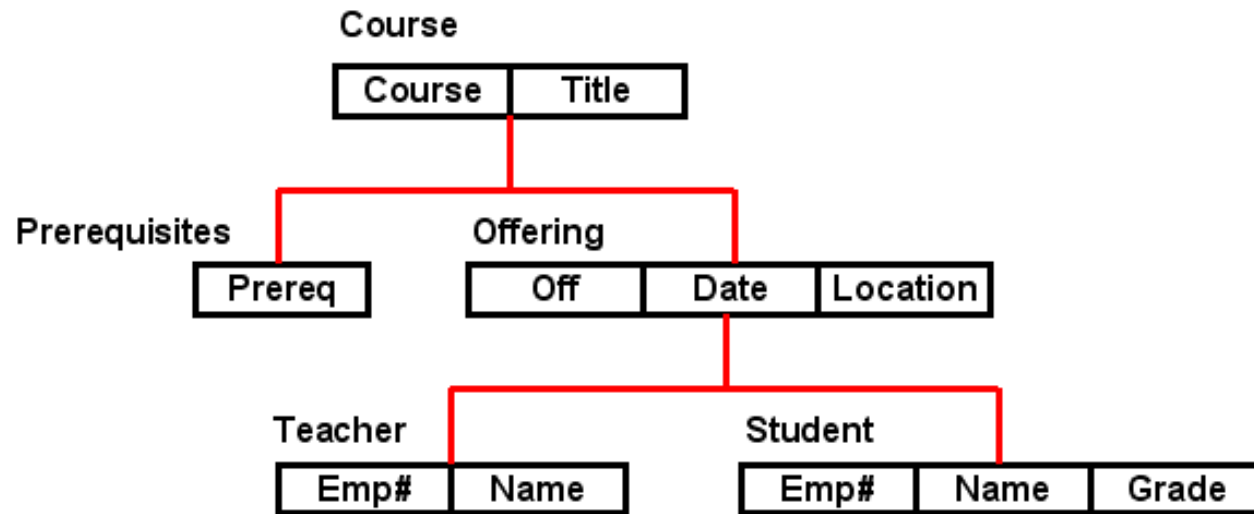
RACF

Language Environment

Datenbanksysteme



Neben DB2 ist IMS ein häufig unter z/OS eingesetztes Datenbanksystem. IMS ist im Gegensatz zu DB2 ein nicht-relationales, hierarchisches Datenbanksystem. Bei einfachen Datenmodellen ermöglicht IMS höhere Transaktionsraten als DB2.



IMS Datenbank-system

IMS benutzt Bäume (Trees) an Stelle der Tabellen in DB2.

Das Beispiel zeigt im oberen Teil einen Baum-Typen und im unteren Teil eine Ausprägung dieses Baum-Typs: Einen Kurs, seine Termine und seine dazugehörigen Buchungen.

IMS Datenbanksystem

Eines der ersten großen DBMS war IMS mit der Sprache DL/I (Data Language One). Die damit verwalteten Datenbanken waren hierarchisch strukturiert.

Das hierarchische Datenmodell des IMS besteht aus einer geordneten Menge von Bäumen, genauer aus Ausprägungen von Bäumen eines bestimmten Typs. Jeder Baum-Typ enthält eine Basis (Root-Segment) und keinen, einen oder mehrere Unterbaum-Typen. Der Unterbaum-Typ seinerseits enthält ggf. wiederum weitere Unterbäume.

Ein IMS Baum besteht aus Segmenten. Ein Segment ist jeweils ein logischer Satz bestehend aus einem oder mehreren Feldern (Fields). Ein Elternsegment kann mehrere Kindsegmente haben. Auch ein Kindsegment kann zur gleichen Zeit, ein Elternsegment sein, das heißt, es kann eigene Kindersegmente haben. Das Segment ohne Elternsegment, das sich an der Spitze des Trees befindet, wird Wurzel Segment (Root Segment) genannt.

Die Sprache **Data Language One (DL/I)** ist das IMS Äquivalent zu der SQL Sprache. Data Language/I (DL/I) Function Calls werden von Anwendungsprogrammen für Query und Update Vorgänge benutzt.

Die logische Struktur der Datenbank ist in der Database Description (DBD) und in den Program Communication Blocks (PCB) definiert. Dabei entspricht ein DBD grob einer Tabelle und ein PCB einer Sicht in relationalen Datenbanken, genauer einem Create-Table- bzw. einem Create-View-Befehl.

Anmerkung: Die korrekte Bezeichnung für die Datenbank ist IMS DB (IMS Data Base). Daneben existiert die IMS DC (IMS Data Communication) Komponente. IMS DC wurde seinerzeit als Transaktionsmonitor spezifisch für IMS DB entwickelt, hat heute aber weniger Bedeutung als der weit verbreitete z/OS CICS Transaktionsmonitor.

DB2 relationale Datenbank

DB2 Universal Database (UDB) ist IBMs relationales Datenbank-Produkt. Es existiert eine identische Implementierung für alle UNIX-, Linux-, OS/2-, und Windows-Betriebssysteme.

Eine zweite getrennte Implementierung mit dem gleichen Namen und erweitertem Funktionsumfang ist für das z/OS-Betriebssystem verfügbar, Obwohl es sich um zwei getrennte Implementierungen handelt, ist die Kompatibilität sehr gut.

Neben Oracle- und Microsoft-SQL ist DB2 eines der drei führenden relationalen Datenbankprodukte. Unter z/OS ist DB2 neben IMS die am häufigsten eingesetzte Datenbank. Andere populäre z/OS-Datenbanksysteme sind IDMS der Fa. Computer Associates sowie Adabas der Fa. Software AG. Oracle unter z/OS wird seit 2009 nicht mehr weiter entwickelt.

DB2 ist eine Server-Anwendung, die grundsätzlich in einem getrennten Adressraum läuft. Wie bei allen Server-Anwendungen ist ein Klient erforderlich, um auf einen DB2-Server zuzugreifen. Der Klient kann ein Anwendungsprogramm auf dem gleichen Rechner sein, oder auf einem getrennten Rechner laufen.

Der Zugriff kann mit Hilfe von SQL-Statements erfolgen, die in einem Anwendungsprogramm eingebettet sind. Alternativ existieren eine Reihe spezifischer SQL-Klienten-Anwendungen.

Prozessverwaltung

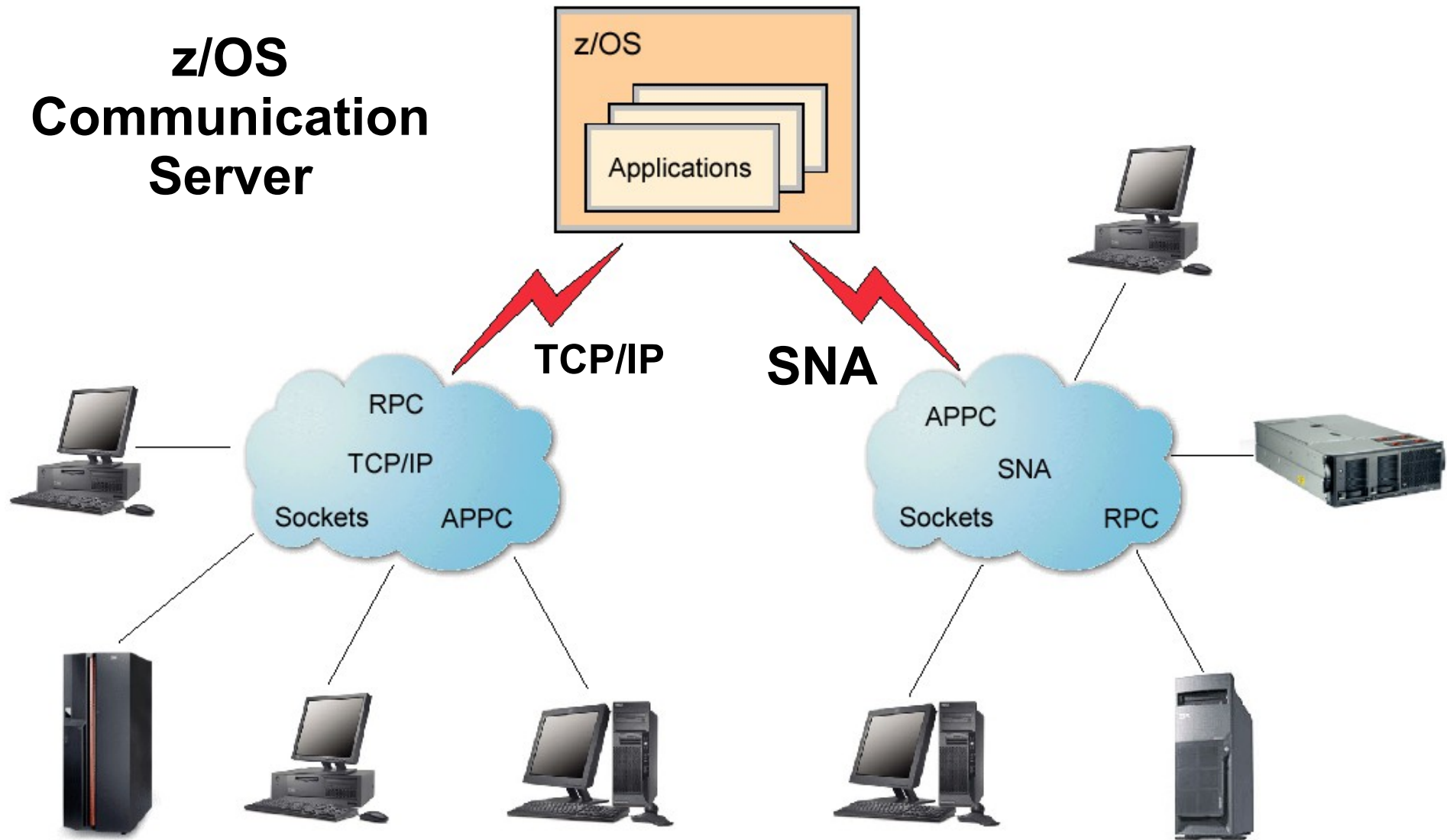
Datenbanken

Communication Server

RACF

Language Environment

z/OS Communication Server



Der **z/OS Communication Server** ist ein eigenständiges Subsystem in einem eigenen virtuellen Adressenraum. Er implementiert die Software (Netzwerk Architektur Stacks) für eine ganze Reihe von Netzwerk Architekturen, besonders für TCP/IP und SNA.

SNA

Die Systems Network Architecture (SNA) wurde von IBM entwickelt und im Jahre 1974 vorgestellt. TCP/IP kam erst wesentlich später.

SNA ist vom Funktionsumfang her immer noch sehr modern und in etwa mit TCP/IP Version 6 vergleichbar. Bis in die 90er Jahre war SNA der de facto Standard in den allermeisten Mainframe Installationen. Danach erfolgte ein gradueller Wechsel zu TCP/IP, getrieben durch die Verbreitung des Internet. Heute sind physische SNA Netze fast überall durch TCP/IP Netze ersetzt worden.

Unter der Decke benutzen viele Systemkomponenten nach wie vor SNA. So kommuniziert Ihr 3270 Emulator über eine SNA Verbindung mit TSO oder CICS. Für den Benutzer ist dies unsichtbar, weil SNA Pakete in TCP/IP Pakete verpackt und über einen TCP/IP Tunnel versandt werden. Für die Kommunikation zwischen einem 3270 Emulator und einem Mainframe verwendet man hierzu ein aufgebohrtes Telnet Protoll, welches als TN3270 bezeichnet wird. Port 23 ist der Standard Telnet Port, und aus diesem Grunde ist Port 23 der Standard Port für Zugriffe auf einen Mainframe Rechner.

Der z/OS Communication Server hat Zusatzeinrichtungen, mit denen getunnelte TCP/IP Pakete entpackt, und der Inhalt an den SNA Stack weitergegeben werden kann.

Prozessverwaltung

Datenbanken

Communication Server

RACF

Language Environment

z/OS “SecureWay” Security Server

Der z/OS Security Server ist ein eigenes Subsystem, welches für sicherheitsrelevante Belange zuständig ist. z/OS ist in Bezug auf Sicherheit allen anderen Betriebssystemen überlegen. Beispiel: Wir betreiben seit 13 Jahren einen Mainframe Server am Lehrstuhl Technische Informatik. Wir haben noch nie ein Security Update installiert und nach unserem Wissen existiert auch kein Virens Scanner für z/OS. Zu diesen hervorragenden z/OS Sicherheitseigenschaften tragen viele Faktoren bei; einer davon ist der z/OS Security Server.

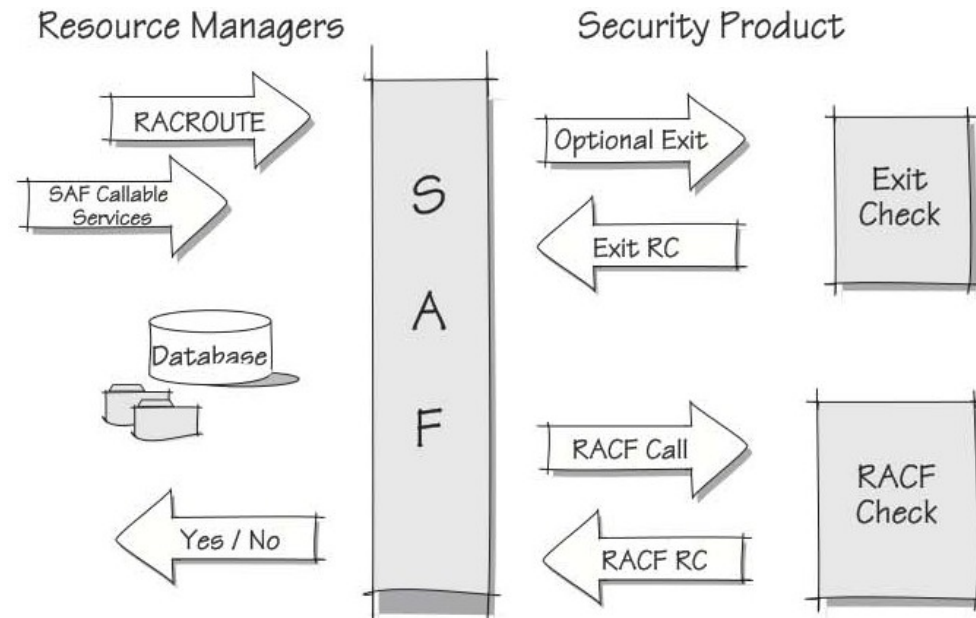
Die wichtigsten Komponenten sind:

- LDAP Server - Secure Directory Server
- Kerberos Network Authentication Service, einschliesslich Kryptographie und Firewall Unterstützung.
Hierfür existiert zusätzliche Hardware Unterstützung:
 - o Zusätzliche Maschinenbefehle für kryptographische Operationen
 - o Kryptographie Assist-Prozessoren
- RACF

LDAP ist ein Verzeichnisdienst (Directory Service). Ein Verzeichnisdienst ist eine spezielle hierarchische Datenbank mit einer eigenen Abfragesprache. Es hat ähnliche Funktionen wie der DNS Namensdienst des Internets, kann aber darüber hinaus zusätzliche Informationen speichern. Verzeichnisdienste werden in der Regel dazu verwendet, Benutzerdaten zentral zu sammeln und Applikationen zur Verfügung zu stellen. Gegenüber einer dezentralen Speicherung z.B. mit einem eigenen Dienst für jedes Programm hat die zentrale Verwaltung den Vorteil, dass Benutzer und Rechte an zentraler Stelle nur einmal geändert werden müssen.

Security Authorisation Facility (SAF)

z/OS spezifiziert kritische Events innerhalb des Betriebssystems als sicherheitssensitive Verzweigungen. Die z/OS **Security Authorisation Facility (SAF)** des z/OS Kernels bewirkt an diesen Stellen den Aufruf einer Security Engine, eines Subsystem Prozesses, der im Benutzer Status läuft.



In z/OS ist diese Security Engine häufig RACF; alternative z/OS Security Engines sind ACF/2 oder TopSecret der Firma Computer Associates.

Zugriffsrechte können von spezifischen granularen Bedingungen abhängig gemacht werden; z.B. der Zugriff auf eine Datenbank kann von der Nutzung eines spezifischen Anwendungsprogramms abhängig gemacht werden, oder auf bestimmte Tageszeiten begrenzt sein.

SAF übergibt der externen Security Engine die pertinente Information. Die Security Engine kann dann auf der Basis von „Profilen“ über die Zugriffsrechte entscheiden.

Resource Access Control Facility (RACF)

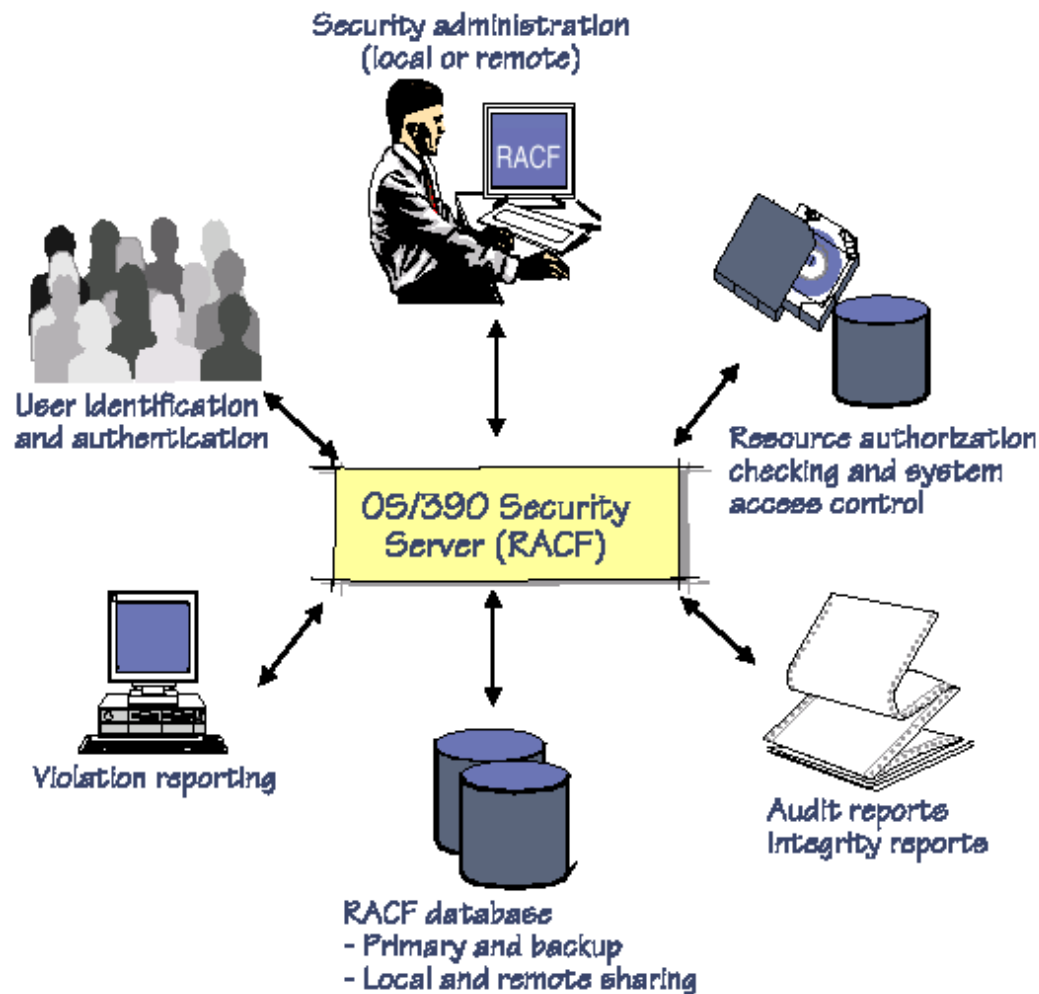
Resource Access Control Facility (RACF) ist ein weiteres z/OS Subsystem. Es ist IBMs Implementierung der Sicherheitschnittstelle SAF. Neben RACF existieren zwei relativ weit verbreitete Subsystem Implementierungen der Firma Computer Associates mit den Namen ACF2 und Top Secret. Letztere können als Alternative zu RACF eingesetzt werden.

Die Hauptfunktionen, die es erfüllt sind:

- Identifikation und Verifikation der Benutzer mittels Benutzerschlüssel und Passwortprüfung (Authentifizierung)
- Schutz von Ressourcen durch die Verwaltung der Zugriffsrechte (Autorisierung)
- Logging der Zugriffe auf geschützte Ressourcen (Auditing).

Die erforderliche Information wird in einer RACF Datenbank in sogenannten Profilen abgelegt. Ein RACF Profil enthält die Benutzerschlüssel (Userids), die zu schützenden Ressourcen (Resources) und Gruppen (Groups).

RACF



RACF bewirkt:

- Identifizierung und Authentifizierung von Benutzern
- Benutzer Authorisierung für Zugriff auf geschützte Ressourcen
- Logging und Berichte über unauthorisierte Zugriffe
- Überwacht die Art, wie auf Ressourcen zugegriffen wird
- Anwendungen können RACF Macros benutzen
- Audit Trail

Literatur : IBM Form No. GC28-1912-06

RACF

RACF benutzt das Konzept von zu schützenden „Ressourcen“. Ressourcen werden in „Klassen“ aufgeteilt. Beispiele für Klassen sind:

- Benutzer
- Dateien
- CICS Transaktionen
- Datenbank Rechte
- Terminals

Jedem Element einer Klasse wird ein „Profil“ zugeordnet. Das Profil besagt, welche sicherheitsrelevanten Berechtigungen vorhanden sind. Die Profile werden in einer Systemdatenbank, der RACF Profildatenbank, abgespeichert.

Vor allem bei der Benutzer-Identifikation entscheidet RACF:

- ob der Benutzer für RACF definiert wird
- ob der Benutzer ein gültiges Password , PassTicket, Operator Identification Card und einen gültigen Gruppen-Namen verwendet.
- ob der Benutzer das System an diesem Tag und zu dieser Tageszeit benutzen darf
- ob der Benutzer autorisiert ist, auf das Terminal (Tag und Zeit) zuzugreifen
- ob der Benutzer autorisiert ist, auf die Anwendung zuzugreifen
- ob der Benutzer autorisiert ist, auf spezifische Daten zuzugreifen

Beispiel:

Ein interaktiver Benutzer logged sich ein. Sein RACF Profil enthält den Benutzerschlüssel (Userid), die zu schützenden Ressourcen (Resources) und Gruppen (Groups). Der Login Prozess überprüft das Profil, ob die Login Berechtigung besteht. Er überprüft weiterhin, ob das Terminal, von dem der Benutzer sich einwählt, eine Zugangsberechtigung hat. Hierzu ruft der Login Prozess RACF auf, welches die entsprechenden Profile in seiner RACF Datenbank konsultiert.

Anschließend ruft der Benutzer ein Programm auf, welches auf eine Datei zugreift. Die OPEN Routine ruft RACF auf, welches das Profil der Datei bezüglich der Zugriffsrechte befragt.

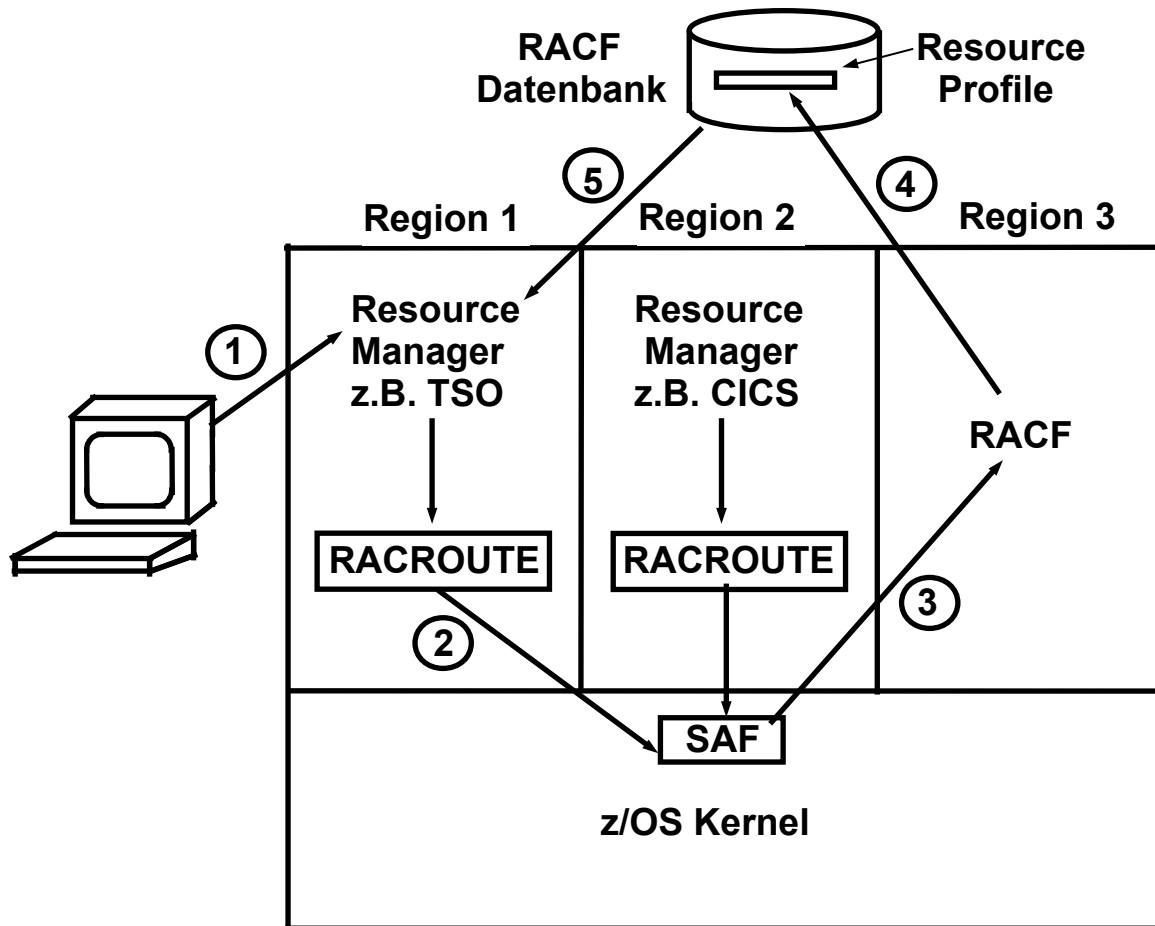
Benutzer Profile enthalten „Capabilities“. Datei Profile enthalten „Access Control Listen“.

Zugriffsrechte können von spezifischen granularen Bedingungen abhängig gemacht werden; z.B. der Zugriff auf eine Datenbank kann von der Nutzung eines spezifischen Anwendungsprogramms abhängig gemacht werden, oder auf bestimmte Tageszeiten begrenzt sein.

Es existieren in einer Installation eine sehr große Anzahl an Profilen, die vom RACF Administrator gewartet werden müssen. Ein Problem ist die Wartung der Profile. Beispiel: Ein Unternehmen hat 100 000 Mitarbeiter und User. Einige wechseln ihren Job innerhalb des Unternehmens, was zu anderen Benutzer Berechtigungen führt. Die RACF Profile müssen geändert werden. Für neu eingestellte Mitarbeiter müssen RACF Profile angelegt werden, für ausgeschiedene Mitarbeiter gelöscht werden. Wenn neue Anwendungen installiert werden, bedingt dies ebenfalls einen Aufwand in der RACF Administration.

Die Sicherheit eines z/OS Rechners hängt davon ab, wie korrekt die Einträge in der RACF Datenbank sind.

RACF Arbeitsweise



1. Ein Benutzer greift auf eine Resource über einen Resource Manager zu, z.B. TSO
2. Der Resource Manager benutzt einen System Call „RACROUTE“ um auf die Security Access Facility (SAF) des z/OS Kernels zuzugreifen. SAF ist eine zentrale Anlaufstelle für alle sicherheitsrelevanten Aufgaben.
3. SAF ruft ein Zugriffskontrolle Subsystem auf. Dies ist normalerweise RACF.
4. RACF greift auf einen „Profile“ Datensatz in seiner eigenen RACF Datenbank zu und überprüft die Zugangsberechtigungen
5. Das Ergebnis teilt RACF dem anfragenden Resource Manager mit.

Prozessverwaltung

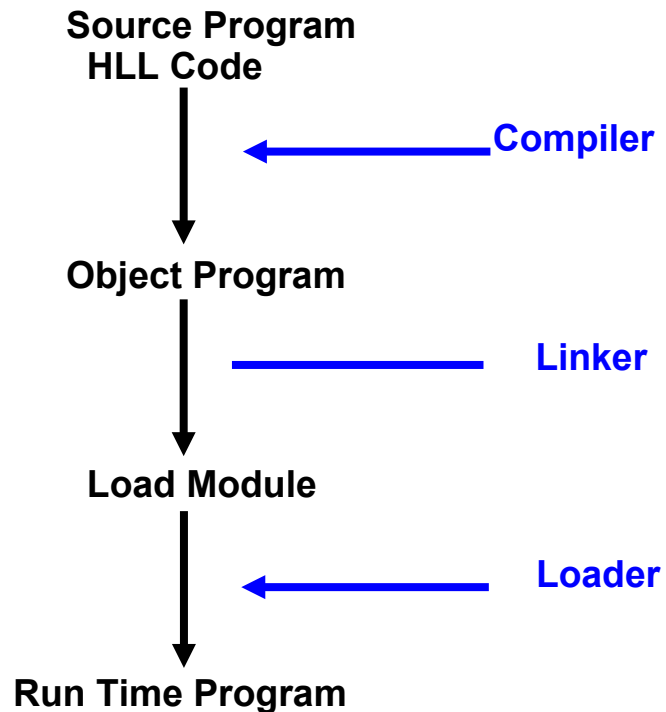
Datenbanken

Communication Server

RACF

Language Environment

Übersetzung eines neu entwickelten Programms



Der Compiler übersetzt ein Quellprogramm (in Cobol, Java, C++, ...) in ein Objektprogramm (Binaries). Das Objektprogramm besteht aus Maschinenbefehlen.

Der Linker verkettet das Objektprogramm, gemeinsam mit einem/mehreren zu einem anderen Zeitpunkt entstandenen Objektprogramm/en, sowie mit Routinen aus einer Programmbibliothek und aus der Run-Time Library zu einem ausführbaren Gesamtprogramm, dem „Load Module“. Die Run-Time Library besteht aus Objektmodulen (object modules).

Alle Komponenten des „Load Modules“ haben eindeutige Adressen; der zusammenhängende Adressbereich beginnt typischerweise mit der Adresse 000...000 .

Der Loader lädt das „Load Module“ vom Plattenspeicher in den Hauptspeicher, typischerweise auf eine Adresse, die nicht mit 000...000 beginnt. Hierzu werden alle Adressen durch den Loader verschoben (relocated).

Language Environment

Das **z/OS Language Environment (LE)** Programm-Management-Modell bietet einen Rahmen, in dem der Code von Anwendungsprogrammen ausgeführt werden kann, die in verschiedenen Sprachen geschrieben wurden.

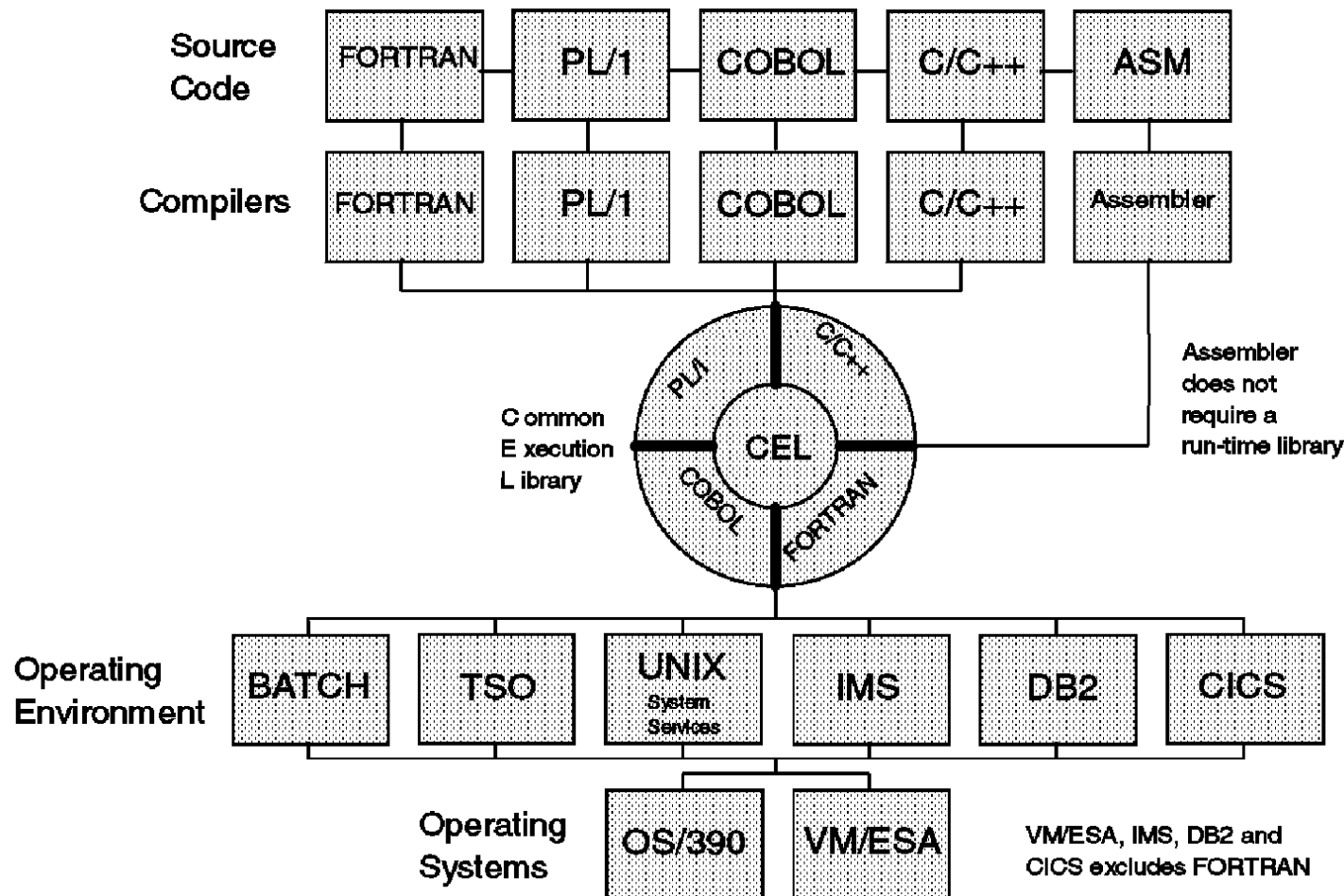
LE bietet eine gemeinsame Laufzeitumgebung für die IBM-Versionen bestimmter Hochsprachen (High Level Language, HLLs) an, nämlich C/C++, COBOL, Fortran, PL/I und Java. LE kann als eine Sammlung von Ressourcen, Bibliotheken und Betriebssystem-spezifischen Diensten und Schnittstellen betrachtet werden.

LE kombiniert häufig verwendete Laufzeitdienste, wie Routinen für

- Mathematische Funktionen (z.B., transzendente Funktionen wie root , arctan , x^π , ...)
- Laufzeit-Message-Handling,
- Condition-Handling,
- Storage-Management, und
- Datum und Uhrzeit.

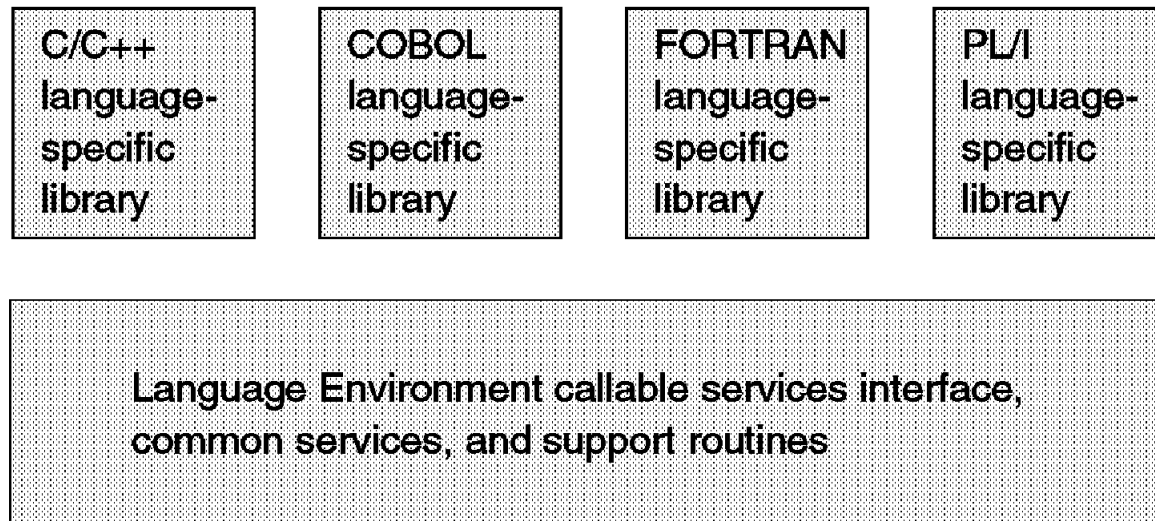
Das bedeutet z.B., dass es nur eine „arctan Object Code Routine“ gibt, die von unterschiedlichen Programmiersprachen genutzt wird. LE stellt dafür eine Reihe von Schnittstellen zur Verfügung, die konsistent für alle unterstützten Programmiersprachen sind.

LANGUAGE ENVIRONMENT FOR OS/390



Jede „High Level Language“ (HLL) hat ihre spezifischen Laufzeit- und SYSLIB-Bibliotheken. Zusätzlich benutzt sie zusammen mit anderen Sprachen eine gemeinsame „Common Execution Library“ (CEL).

Die auf diese Weise hergestellten „Load Modules“ können in verschiedenen Ausführungsumgebungen (Batch, CICS, DB2 usw.) unter z/OS oder z/VM ausgeführt werden.

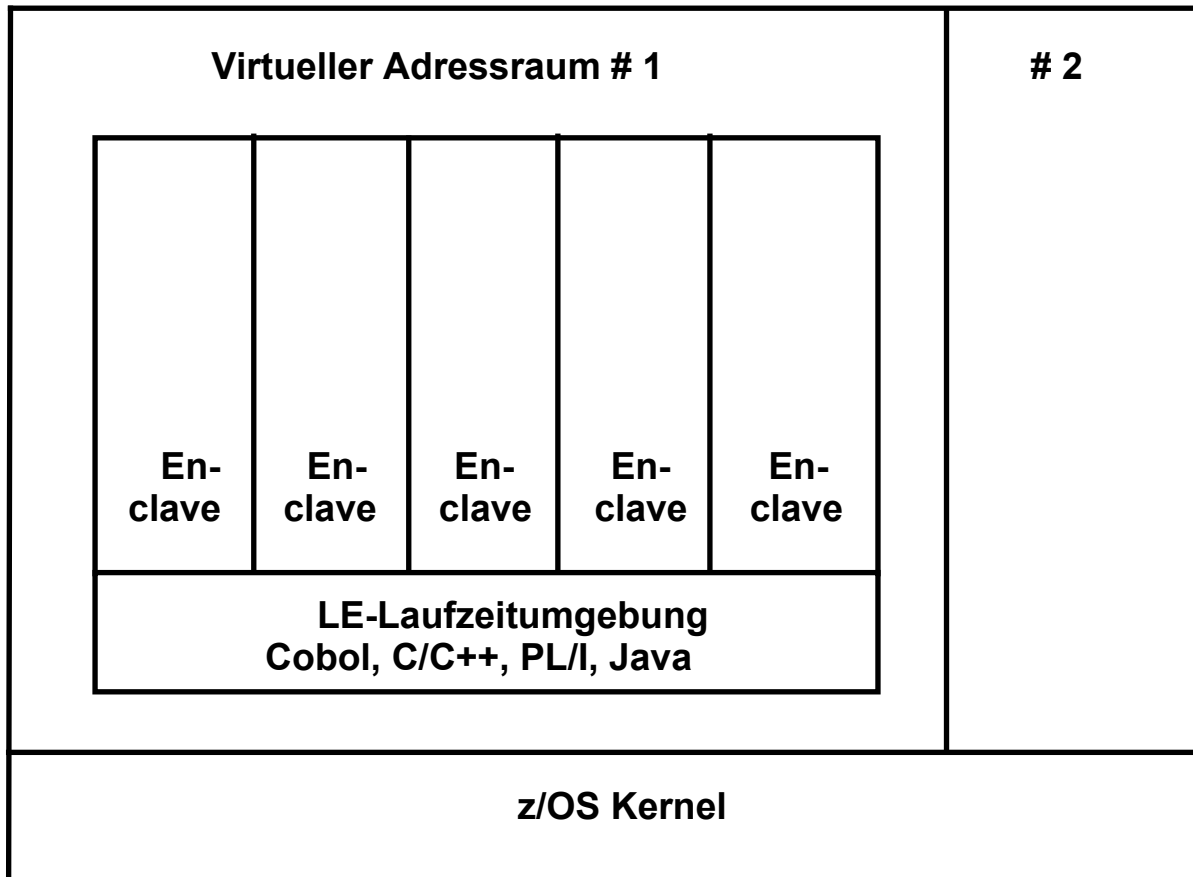


„Language Environment“ (LE) besteht aus:

- Basic-Routinen, die den Start und Stopp von Programmen, Zuweisung von Speicherkapazitäten, sowie Handhabungsbedingungen unterstützen.
- Gemeinsame bibliothekarische Dienstleistungen, wie z. B. mathematische Funktionen sowie Datums- und Uhrzeit-Dienste.
- sprachspezifische Teile der Laufzeitbibliothek, da viele sprachspezifische Aufrufe von LE-Routinen unterstützt werden müssen. Dabei ist das Verhalten konsistent für alle unterstützten Sprachen.
- Einrichtungen für die Kommunikation zwischen Programmen, die in verschiedenen Sprachen geschrieben sind.

POSIX-Unterstützung ist in der LE-Grundausrüstung sowie in der C/C++-spezifischen Bibliothek vorhanden.

z/OS-Enclaves



LE-Enclaves können eingesetzt werden um unterschiedliche Anwendungen innerhalb des gleichen virtuellen Adressraums voneinander zu isolieren. Spezifisch ist es hiermit möglich, mehrere JVMs innerhalb eines Adressraums laufen zu lassen.

LE Program Management

Drei Elemente:

- **Prozess**
- **Enclave**
- **Thread**

sind der Kern des „Language Environmen“-Programm-Management-Modells.

Ein Prozess kann mehrere Enklaven enthalten. Jede Enklave enthält ihren eigenen Programm-Code. Der Code kann in verschiedenen Sprachen geschrieben werden.

Die Threads innerhalb der gleichen Enklave benutzen gemeinsam den gleiche Code, und ebenso den Heap-Storage. Jeder Thread hat seinen eigenen Stack.

Process

Die höchste Ebene des LE-Programm-Modells ist der Prozess. Jeder Prozess hat seinen eigenen Adressraum. Ein Prozess ist eine Sammlung von Ressourcen, sowohl Programmcode als auch Daten.

Ein Prozess besteht in der Regel aus einer Enclave und ist logisch getrennt von anderen Prozessen. (Mehrere Enclaves pro Prozess sind möglich). Jeder Prozess hat seinen eigenen Speicherplatz; sie haben keinen gemeinsamen Speicherplatz. Prozesse sind gleichberechtigt und unabhängig voneinander. Es besteht keine hierarchische Gliederung.

Prozesse können neue Prozesse erstellen. Sie kommunizieren miteinander mit Hilfe einer LE-definierten Kommunikation, z. B. um anzuzeigen, dass ein Prozess beendet wurde.

Enclave

Die **Enclave** ist eine Sammlung von Routinen, die eine Anwendung darstellen. Eine Enclave ist das Äquivalent von einer der folgenden Bezeichnungen:

- Einer „Run Unit“ in COBOL
- Einem Programm, (Main-C-Funktion und seine Sub-Funktionen), in C/C++
- Einer „Main Procedure“ und alle seiner Unterprogramme in PL/I
- Einer JVM und ihrer Klassen in Java

Die Enclave besteht aus einem Hauptprogramm und einer beliebigen Anzahl von Unterprogrammen. Die Main-Routine ist als erstes in einer Enclave auszuführen; alle nachfolgenden Routinen werden als Unterprogramme behandelt.

Externe Daten sind nur innerhalb der Enclave, in der sie sich befinden, verfügbar. Der Geltungsbereich der externen Daten ist die Enclave. Auch wenn externen Daten identische Namen in verschiedenen Enclaves aufweisen, sind sie nur innerhalb ihrer Enclave gültig.

Die Threads in einer Enclave können weitere Enclaves erstellen, diese können mehr Threads erstellen, usw.

Für Sonderfälle existiert ein Mechanismus, mit dem Enclaves Daten gemeinsam nutzen können. Ein Beispiel ist die „PL/I Standard SYSPRINT File“. Sie wird von mehreren Enclaves innerhalb einer Anwendung geteilt (shared).

Es existiert eine LE-Message-File, die von mehreren Enclaves gemeinsam genutzt werden kann. Dies ist ein nützlicher zentraler Speicherort für Nachrichten, die während der Ausführung von Programmen innerhalb der Enclaves generiert werden können.

Threads

Jedes Enclave besteht aus mindestens einem Thread.

Ein **Thread** ist ein Ausführungskonstrukt, das aus synchronen Aufrufen und Terminierungen von Routinen besteht. Ein Thread wird von dem System mit seinem eigenen Runtime-Stack, Befehlszähler und Registern verarbeitet. Threads können gleichzeitig mit anderen Threads innerhalb einer Enclave existieren.

Der Runtime-Stack steuert die Ausführung eines Threads. Er enthält außerdem den Befehlszähler, Register und Condition-Handling-Mechanismen. Jeder Thread stellt eine unabhängige Instanz einer Routine dar, die in einer Enclave unter Benutzung der Enclave-Ressourcen läuft.

Threads benutzen gemeinsam alle Ressourcen einer Enclave. Ein Thread kann den ganzen Speicher innerhalb einer Enclave adressieren. Alle Threads sind gleichberechtigt und unabhängig voneinander und nicht hierarchisch miteinander verbunden.

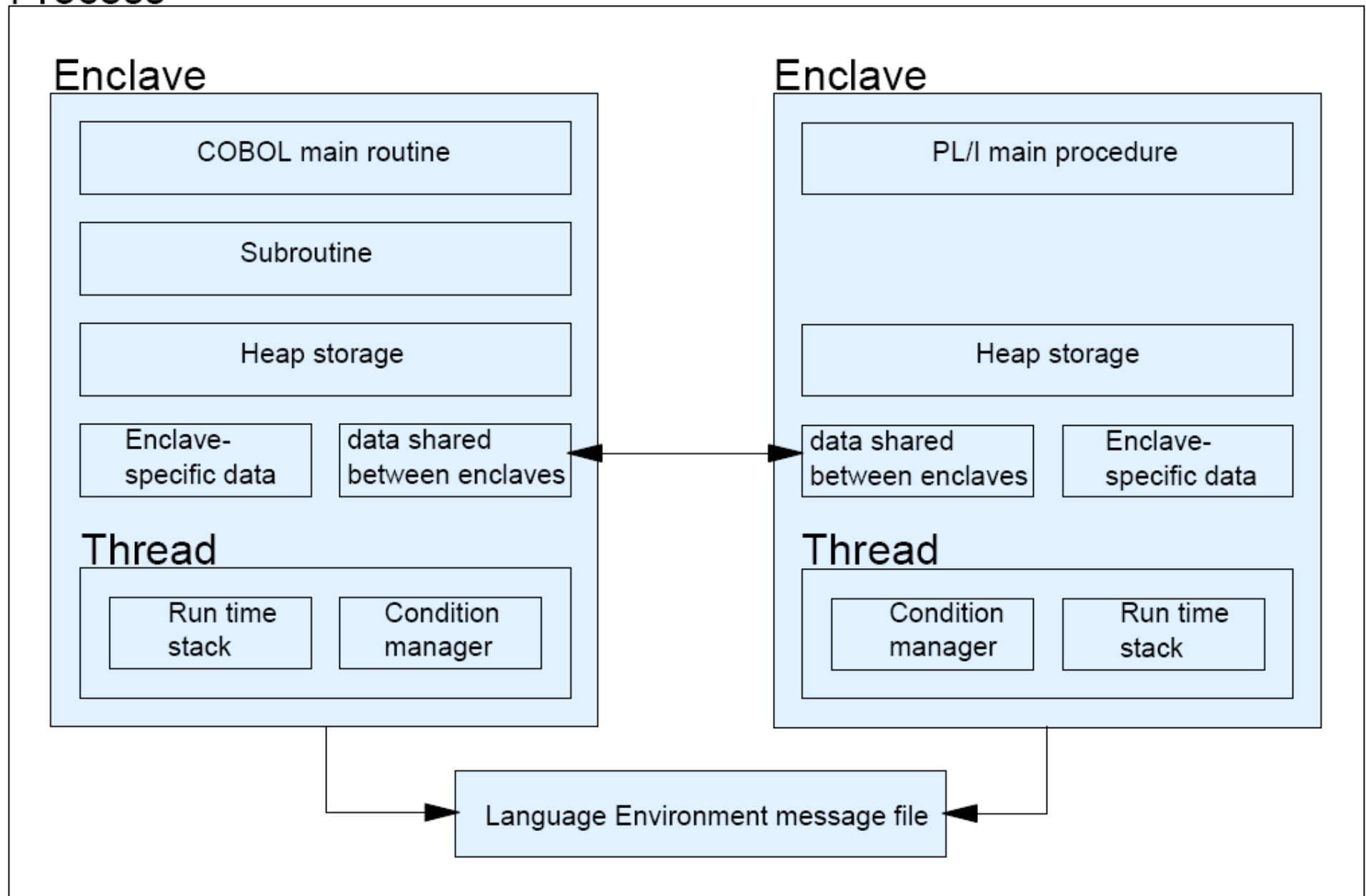
Weil Threads mit ihrem eigenen Run-Time-Stacks arbeiten, können sie gleichzeitig innerhalb einer Enclave laufen und ihren Speicherplatz verwalten. Weil sie gleichzeitig ausführbar sind, können Threads verwendet werden, um eine parallele Verarbeitung von Anwendungen (und von ereignisgesteuerten Anwendungen) zu implementieren.

In einer z/OS-Enclave kann jeder Thread mit unabhängigen eigenen Performance-Zielen ausgeführt werden. Mit z/OS-Workload-Management (WLM) kann man z.B. z/OS-Performance-Ziele für einzelne DB2-Server-Threads etablieren.

Threads innerhalb einer Enclave erfordern eine kritische Handhabung um zu gewährleisten, dass sie sich nicht gegenseitig beeinflussen. Die Isolation muss gewährleistet sein. **Deshalb benutzen CICS-Anwendungen in der Regel keine Threads, um mehrere Transaktionen in einer einzigen Enclave laufen zu lassen.**

Allerdings verwendet der neue CICS-„JVM Server“ Threads! Das OSGi-Framework innerhalb jeder JVM stellt die erforderliche Isolation (teilweise) sicher.

Process



z/OS Betriebssystem Teil 7

Weiterführende Information

Mehr Details zum z/OS Betriebssystem

Unsere Lehrveranstaltung Enterprise Computing hat die Zielrichtung, das Wissen für die Entwicklung neuer z/OS Anwendungen sowie deren Integration in das Internet zu vermitteln. Wir beschränken uns dabei auf die hierfür erforderlichen z/OS Betriebssystem Grundlagen.

Systemadministratoren benötigen weitergehendes Wissen über das z/OS Betriebssystem sowie das Zusammenspiel mit den angeschlossenen Festplatten. Mehr Details zu z/OS sind in einem Vorlesungsscript enthalten, welches Prof. Spruth für eine Lehrveranstaltung 2009 an der IT Akademie Bayern verfasste.

Es ist verfügbar im pdf Format unter <http://jedi.informatik.uni-leipzig.de/de/VorlesMirror/bs/index.html>

Eine fast identische HTML Seite ist verfügbar unter <http://jedi.informatik.uni-leipzig.de/de/VorlesMirror/bs/word.html>

Hier können Sie die gleichen Dokumente im Microsoft Word Format herunterladen.

JCL

Ramesh Krishna Reddy: JCL Tutorial:

<http://www.mainframegurukul.com/srcsinc/drona/programming/languages/jcl/jcl.chapter1.html>

Introduction to JCL

<http://www.informatik.uni-leipzig.de/cs/Literature/Textbooks/jcl/jclintro.pdf>

JCL Lehrbuch

<http://www.informatik.uni-leipzig.de/cs/Literature/Textbooks/jcl/JCLBuch.pdf>

IBM: z/OS JCL Users Guide. SA22-7598-04, July 2004

<http://www.informatik.uni-leipzig.de/cs/Literature/Textbooks/jcl/MvsJclUserGuide.pdf>

IBM: z/OS JCL Reference. GC28-1757-09 September 2000

<http://www.informatik.uni-leipzig.de/cs/Literature/Textbooks/jcl/MvsJclReference.pdf>

Storage Management Subsystem

<http://jedi.informatik.uni-leipzig.de/de/VorlesMirror/es/Vorles/SMS01.pdf>

enthält eine gute Übersicht über das Storage Management Subsystem

TSO/ISPF Benutzeroberfläche

Wenn Ihnen die TSO bzw. ISPF Benutzeroberfläche nicht gefällt, und Sie Eclipse mit RDz auch keinen Geschmack abgewinnen können:

Es existieren unzählige Software Produkte, welche mit „verbesserten“ Oberflächen arbeiten.

Ein Beispiel ist SELCOPYi, siehe

http://www.cbl.com/pdf/SELCOPYi_Brochure_zOS.pdf

oder Video Tutorials unter

(läuft gut mit dem Windows Media Player).

<http://www.cbl.com/cblidem.html>

Wie entwickelt man ein neues Betriebssystem ?

Zum Thema Software Engineering existieren Tonnen von Lehrbüchern und Fachaufsätzen.

Ein Lehrbuch schlägt alle anderen um Meilen.

z/OS begann Anfang 1966 unter dem Namen OS/360, als reines Stapelverarbeitungssystem. Es wurde von Fred Brooks entwickelt, und diente als Vorlage für sein 1975 erscheinendes berühmtes Buch „The mythical Manmonth“ (http://en.wikipedia.org/wiki/The_Mythical_Man-Month).

Fred Brooks ist von der wissenschaftlichen Gemeinschaft auf Grund der von ihm vertretenen Thesen stark angefeindet worden. Heute gibt man ihm recht. Das Buch wird auch gerne als "The Bible of Software Engineering" bezeichnet.

Die heute erhältliche Version des Buches enthält zusätzlich einen in der Zeitschrift IEEE Computer, vol. 20, no. 4, 1987 erschienenen Aufsatz „No Silver Bullet: Essence and Accidents of Software Engineering“, verfügbar unter <http://www.informatik.uni-leipzig.de/cs/Literature/Textbooks/NoSilverBullet.pdf>.

Wie war das noch mit den Lochkarten ?

Ein faszinierender Überblick über die Entwicklung der Lochkartentechnik ist zu finden in dem Buch

Günther Sandner, Hans Spengler:

Die Entwicklung der Datenverarbeitung von Hollerith Lochkartenmaschinen zu IBM Enterprise-Servern

ISBN-10: 3-00-019690-0, ISBN13: 978-3-00-019690-4.

Ein kostenloses Download ist verfügbar unter

<http://www.informatik.uni-leipzig.de/cs/Literature/History/SandnerSpengler.pdf>

Zahlreiche Darstellungen von Lochkarten sind zu finden unter [http://www.google.de/search?](http://www.google.de/search?q=punched+card&hl=de&lr=&as_qdr=all&prmd=imvns&tbm=isch&tbo=u&source=univ&sa=X&ei=wiglUMu4FobV4QSk0oHICA&ved=0CCoQsAQ&biw=1516&bih=963)

[q=punched+card&hl=de&lr=&as_qdr=all&prmd=imvns&tbm=isch&tbo=u&source=univ&sa=X&ei=wiglUMu4FobV4QSk0oHICA&ved=0CCoQsAQ&biw=1516&bih=963](http://www.google.de/search?q=punched+card&hl=de&lr=&as_qdr=all&prmd=imvns&tbm=isch&tbo=u&source=univ&sa=X&ei=wiglUMu4FobV4QSk0oHICA&ved=0CCoQsAQ&biw=1516&bih=963)

IMS

IMS Datenbanken werden nach wie vor sehr häufig eingesetzt. Eine Einführung (IMS Primer) in das IMS Datenbanksystem ist zu finden unter:

<http://www.redbooks.ibm.com/redbooks/pdfs/sg245352.pdf>

Eine andere, etwas kürzere Einführung:

Bemerkung: Es existiert zusätzlich zu der IMS Datenbank (allgemeine Bezeichnung IMS DB) noch ein IMS Transaction Manager (allgemeine Bezeichnung IMS DC), der aber im Vergleich zum CICS Transactionsmanager keine große Bedeutung hat.

Hier zwei Video Tutorials

<http://www.youtube.com/watch?v=Xs8nbkNSqdl><http://www.mainframes360.com/2009/04/imsdb-tutorials.html>

<http://www.youtube.com/watch?v=IE7a0ZXzrow>

IMS Zugriff über ein iPhone

<http://imsmadesimple.tumblr.com/post/967487367/tutorial-access-ims-data-on-your-iphone>