Mainframe Internet Integration

Prof. Dr. Martin Bogdan
Prof. Dr.-Ing. Wilhelm G. Spruth

SS2013

Java Transaction Processing Teil 2

Enclaves

Language Environment

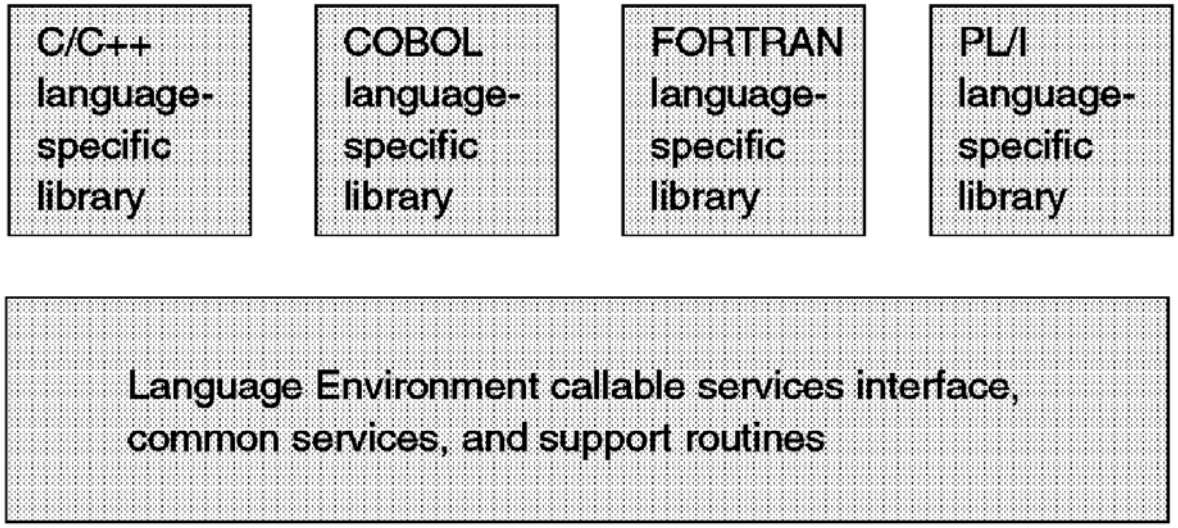
Das z/OS „Language Environment“ (LE) Programm Management Modell bietet einen Rahmen, in dem der Objekt Code von Anwendungsprogrammen ausgeführt werden kann, die in verschiedenen Sprachen geschrieben wurden.

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 häufig verwendete Laufzeit-Dienste, wie Routinen für

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

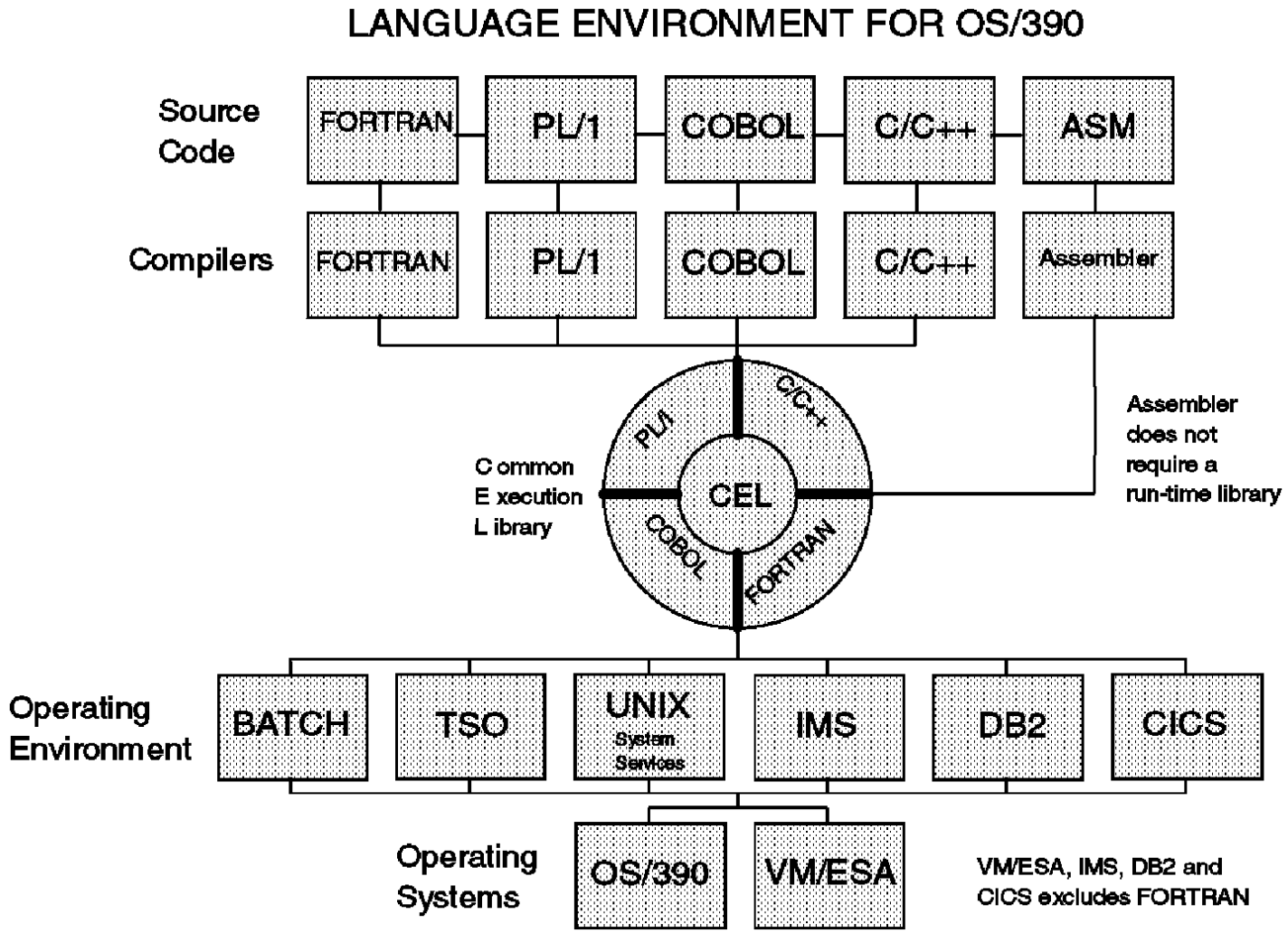
Das bedeutet z.B., dass es nur eine arctan Object Code Routine gibt, die von unterschiedlichen Programmiersprachen genutzt wird. LE stellt dafür eine Reihe von Schnittstellen zur Verfügung, die konsistent für alle unterstützten Programmiersprachen sind.



Language Environment besteht aus:

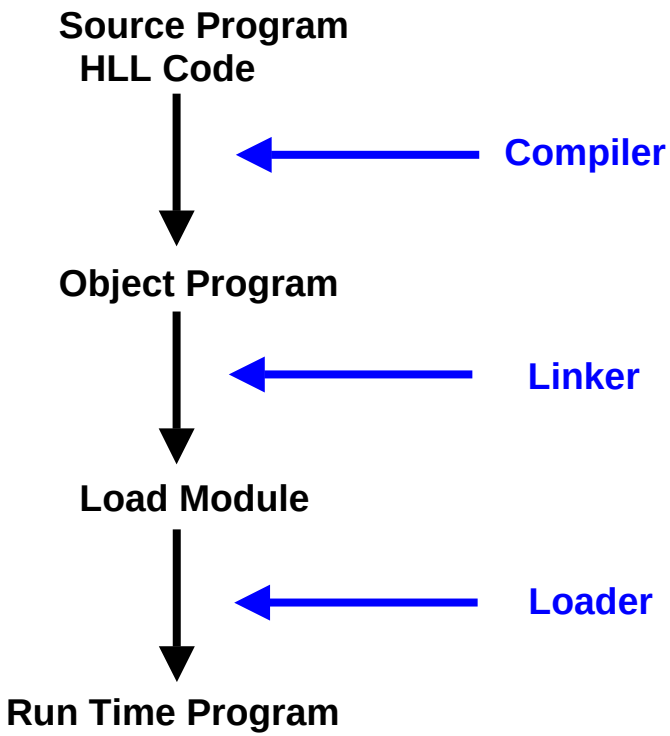
- **Basic-Routinen, die den Start-und Stopp von Programmen, Zuweisung von Speicherkapazitäten, sowie Handhabungsbedingungen unterstützen.**
- **Gemeinsame bibliothekarischen Dienstleistungen, wie z. B. mathematische Funktionen sowie Datum und Uhrzeit Dienste.**
- **Sprach-spezifische Teilen der Laufzeitbibliothek, da viele Sprach-spezifische Aufrufe von Language Environment 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 Language Environment Grundausstattung sowie in der C/C++ spezifischen Library vorhanden.



Jede 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.



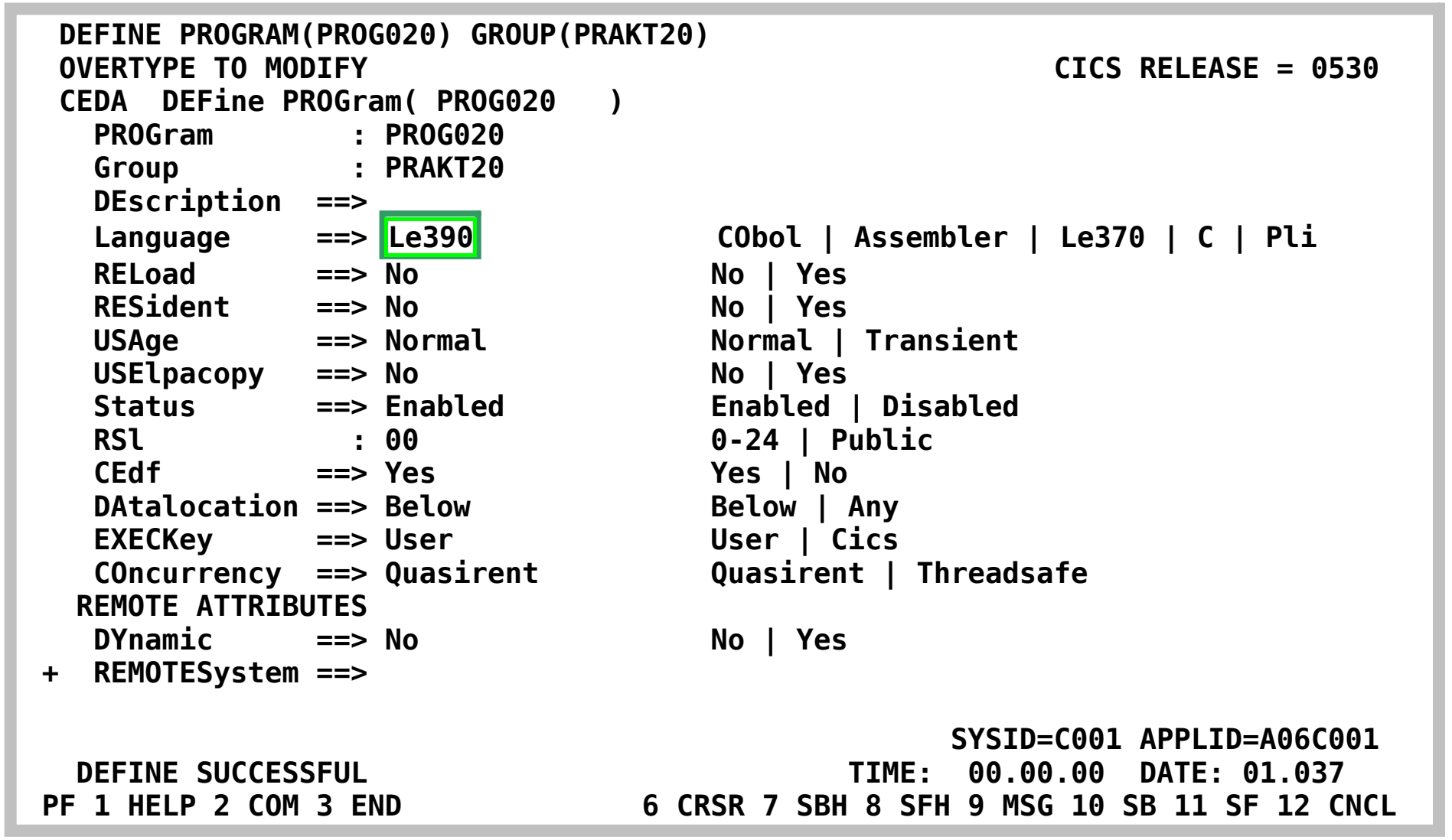
Der Compiler übersetzt ein Source Programm (in Cobol, Java, C++, ...) in ein Object Programm (Binaries). Das Object Programm besteht aus Maschinenbefehlen.

Der Linker verkettet das Objekt Programm, gemeinsam mit einem oder mehreren zu einem anderen Zeitpunkt entstandenen Objekt Programm(en), sowie mit Routinen aus einer Programmbibliothek und aus der Common Execution Library zu einem ausführbaren Gesamtprogramm, dem Load Module. Alle Kompnenten des Load Modules haben eindeutige Adressen; der zusammenhängende Adressenbereich beginnt typischerweise mit der Adresse 000...000 .

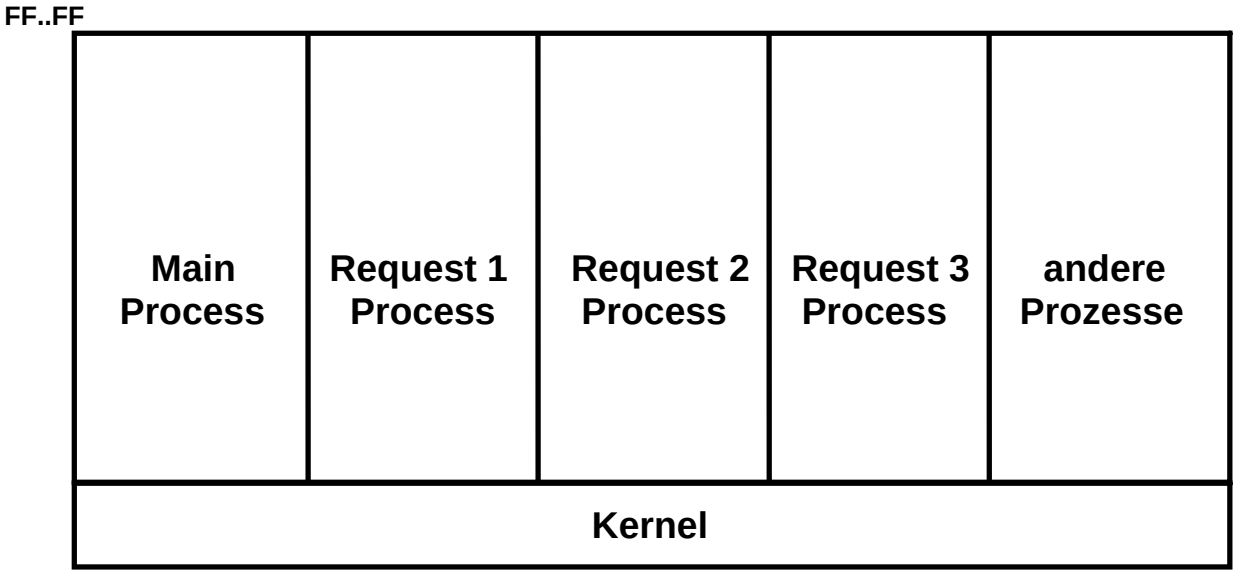
Der Loader lädt das Load Module vom Plattenspeicher in den Hauptspeicher, typischerweise auf eine Adresse, die nicht mit 000...000 beginnt. Hierzu werden alle Adressen durch den Loader verschoben (relocated).

Übersetzung eines neu entwickelten Programms

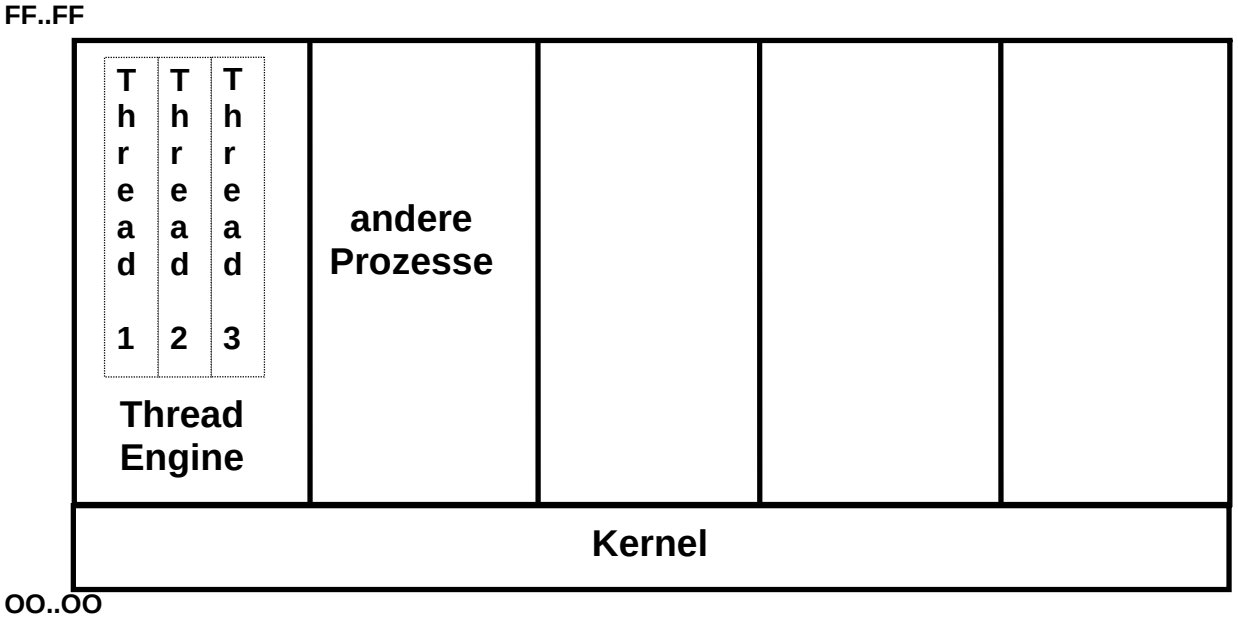
Die Common Execution Library besteht aus Object Modulen.



Als Sie Ihr erstes CICS Tutorial bearbeitet hatten, sahen Sie den oben dargestellten Bildschirm. Sie wurden gebeten, eine Sprache für Ihre CICS-Programm angeben, und die richtige Antwort war Le390 (oder Le370). Le390 ist jedoch keine Sprache, es spezifiziert das z/OS Language Environment als die Ausführungsumgebung für Ihre CICS-Anwendung.



Ansatz mit mehreren Prozessen



Thread Ansatz

Ein Hochleistungs-Transaktionssystem verarbeitet in jedem Augenblick gleichzeitig Hunderte oder Tausende von Transaktionen. Für die Verarbeitung gibt es zwei Alternativen:

- **1 Prozess pro Transaktion**
- **1 Thread pro Transaktion)**

1 Prozess pro Transaktion bedeutet, dass jede Transaktion in einem eigenen virtuellen Adressenraum (z/OS Region) läuft. Vorteilhaft ist, dass die Isolation der Transaktionen untereinander (das i in ACID) optimal gewährleistet ist. Nachteilig ist der höhere Verarbeitungsaufwand im Vergleich zum Thread Ansatz.

Threads haben den Nachteil, dass die Isolation der Threads gegeneinander gewährleistet werden muss.

Bei der z/OS Version von CICS laufen alle Anwendungen unter einem Thread ähnlichen Mechanismus in einem einzigen Adressenraum.

CICS Enclaves

Die grundlegende CICS Konfiguration besteht aus einer einzigen z/OS Region, die den CICS Nucleus beinhaltet, sowie bis zu mehr als 1 000 gleichzeitig ausgeführten Transaktionen beherbergt.

Transaktionen müssen ACID Anforderungen erfüllen: Atomicity, Consistency, Isolation, Durability.

Die Isolation ist oft schwierig zu implementieren. CICS verwendet etwas ähnliches wie Threads, den „Enclave“ Mechanismus.

Enclaves sind ein Teil des z/OS Language Environment (LE). Enclaves sind Thread-ähnliche Einheiten. Sie sind Laufzeit-Einheiten, die das Äquivalent einer multithreaded Verarbeitung durch den CICS Nucleus ermöglichen. Enclaves stellen sicher, dass parallel ausgeführte Transaktionen innerhalb eines einzigen z/OS Adressraums (Region) voneinander isoliert sind.

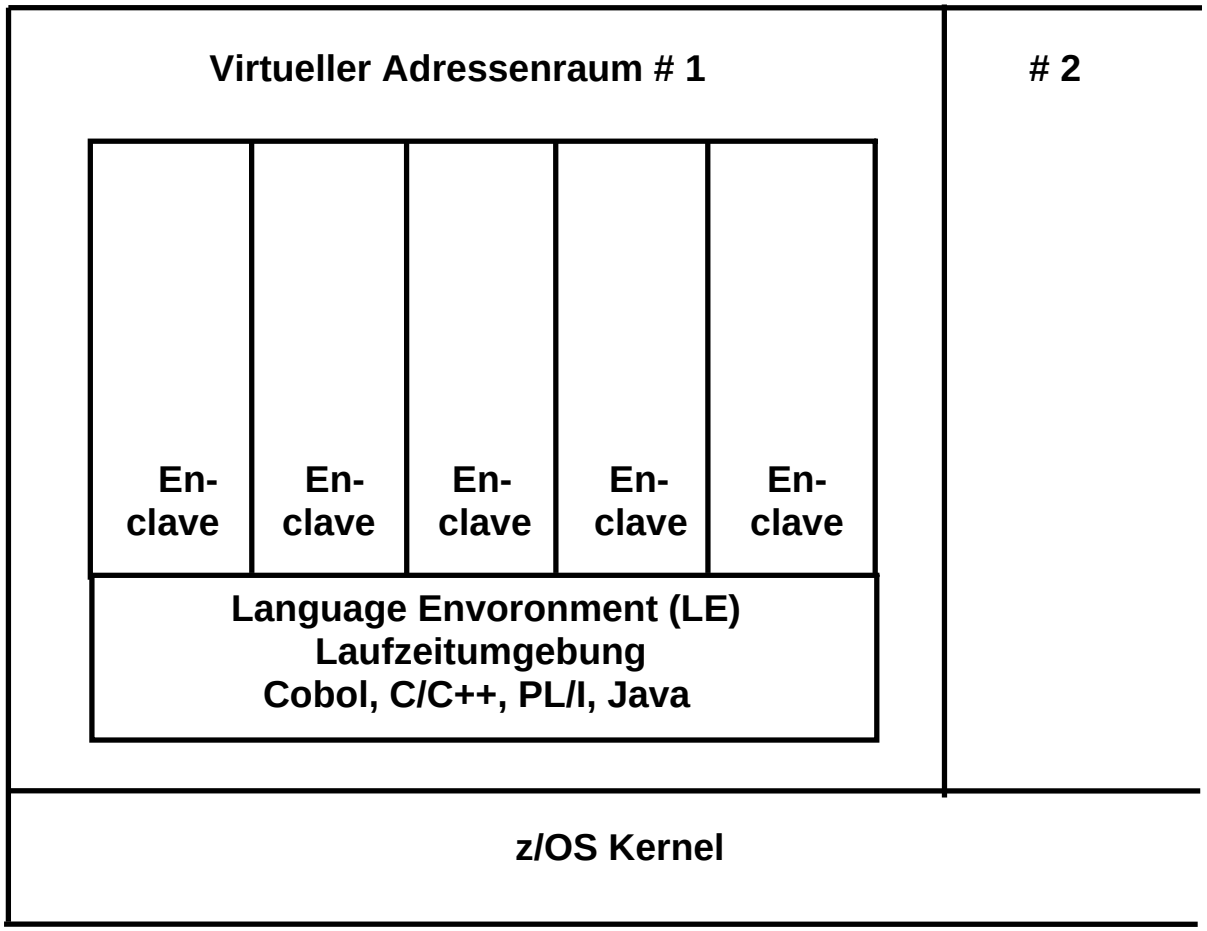
LE Enclaves können eingesetzt werden, um unterschiedliche Anwendungen innerhalb des gleichen virtuellen Adressenraums voneinander zu isolieren. Sie werden u.a. in transaktionalen Subsystemen wie CICS, IMS und DB2 benutzt. Zahlreiche Enclaves laufen in einem einzigen CICS Virtuellen Adressenraum. CICS Anwendungen werden in getrennten Enclaves ausgeführt.

Enclaves benutzen hierfür einen Speicherschutzschlüssel (storage protection key). Der Hardware Speicherschutz arbeitet unabhängig und zusätzlich zu dem üblichen Speicherschutz über Segment- und Seitentafeln und ist eine Einrichtung der z/Series Hardware Architektur, die auf anderen Plattformen nicht verfügbar ist. Der CICS Nucleus verwendet typischerweise Speicherschutzschlüssel Nr. 8, die Anwendungen in ihren Enclaves Speicherschutzschlüssel Nr. 9. Der CICS Nucleus ist somit vor falschen Zugriffen durch fehlerhafte Anwendungen in den Enclaves geschützt. Siehe Vorlesung Wintersemester Thema Einführung Teil 3, oder http://jedi.informatik.uni-leipzig.de/de/VorlesMirror/es/Vorles/HWprotect.pdf

Enclaves und Java

Im Falle von Java läuft in jeder Enclave eine JVM, und in dieser eine Java Transaktion unter einem eigenen Open TCB (JTCB). Spezifisch ist es hiermit möglich, mehrere JVMs innerhalb eines Adressraums laufen zu lassen. Siehe

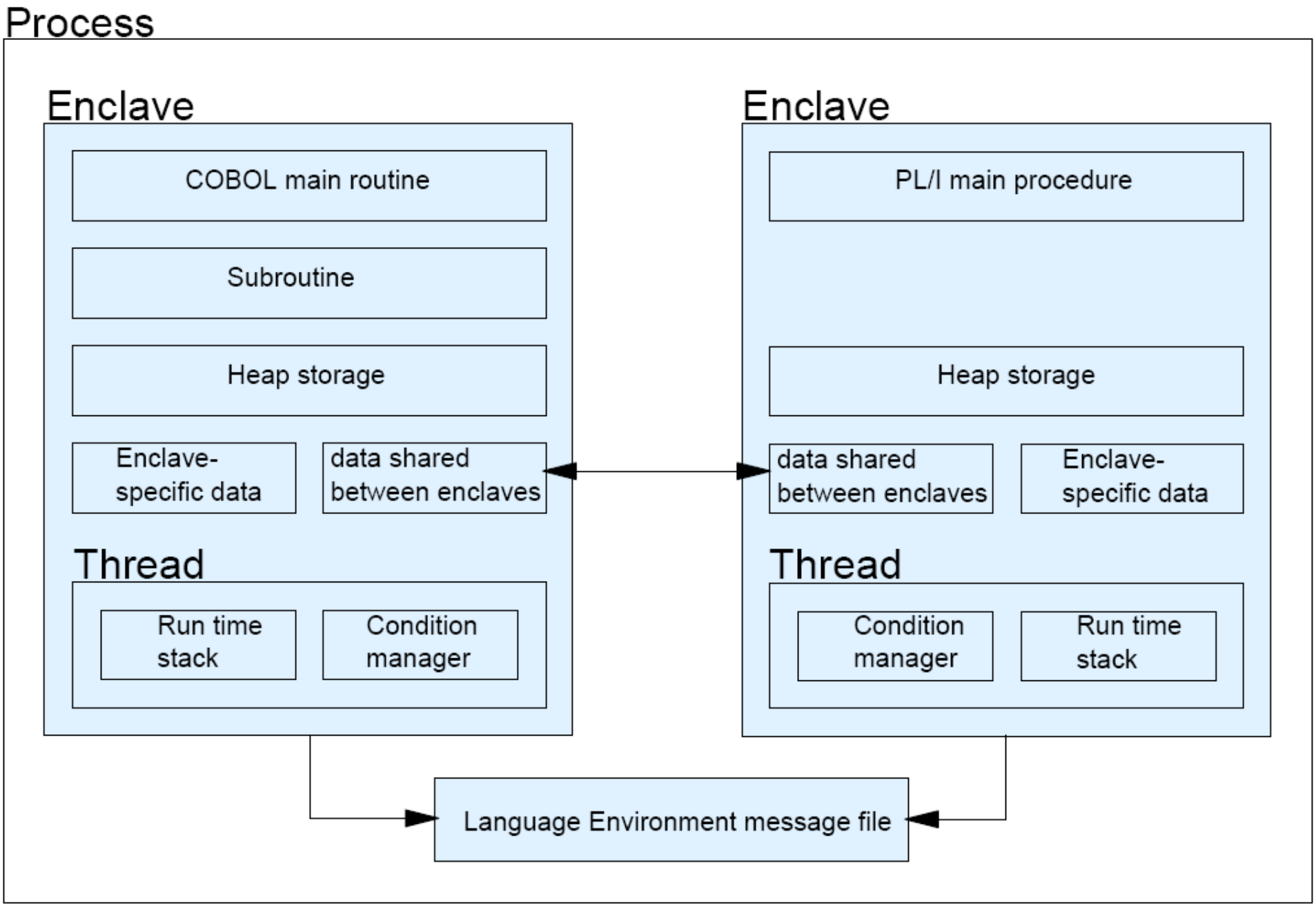
http://jedi.informatik.uni-leipzig.de/de/VorlesMirror/ii/Vorles/OpenTCB.pdf



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

Im Falle von CICS ist die LE Laufzeitumgebung Bestandteil des CICS Nucleus.

z/OS Enclaves


Process
Enclave
COBOL main routine
Subroutine
Heap storage
Enclave-specific data
data shared between enclaves
Thread
Run time stack
Condition manager
Enclave
PL/I main procedure
Heap storage
data shared between enclaves
Enclave-specific data
Thread
Condition manager
Run time stack
Language Environment message file

LE Program Management

Drei Elemente:

- **Prozess,**
- **Enclave und**
- **Thread**

sind der Kern des Language Environment Programm-Management-Modells.

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

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

Process

Die höchste Ebene des Language Environment Programm-Modells ist der Prozess. Jeder Prozess hat seinen eigenen Adressenraum. 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 Language Environment 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**
- **Ein Programm, (Main-C-Funktion und seine Sub-Funktionen), in C/C++**
- **Eine Main Procedure und alle seine Unterprogramme in PL/I**
- **Eine JVM und ihre Klassen in Java**

Die Enclave besteht aus einem Hauptprogramm und einer beliebige 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 identisch Namen in verschiedenen Enclaven 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 ist shared von mehreren Enclaves innerhalb einer Anwendung.

Es existiert eine Language Environment 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.

Enclaves und WLM

Einer der wichtigen Eigenschaften von CICS Enclave Transaktionen ist, dass der z/OS Work Load Manager sie unabhängig voneinander verwalten kann, auch wenn mehrere Enclaves in dem gleichen Adressenraum laufen.

Den einzelnen aktiven Enclaves in einem Adressraum können ihre eigenen unabhängigen Dispatching und I/O-Prioritäten zugeordnet werden, abhängig von den WLM Goals, die der Benutzer definiert hat. Wichtige Enclave Transaktionen können ihre WLM Goals nicht verpassen, weil weniger wichtige Transaktionen im gleichen Adressraum ausgeführt werden.

Threads

Jedes Enclave besteht aus mindestens einem Thread.

Ein Thread ist ein Ausführungs-Konstrukt, das aus synchronen Aufrufen und Terminierungen von Routinen besteht. Ein Thread wird von dem System mit seinem eigenen Runtime-Stack, Befehlszähler und Registern dispatched. 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 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 ausgeführt werden können, 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 einem einzigen Enclave laufen zu lassen.

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