UNIVERSITÄT LEIPZIG

Mainframe Internet Integration

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

SS2013

Java Transaction Processing Teil 3

CICS und Java Standard Edition

TCB

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 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 (Prozessidentifier, oder PID).**
- **Register Inhalte für den Prozess (Mehrzweck, Gleitkomma, Controk, Access Register) einschließlich des Programm-Status Wortes (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 Accounting Informationen, 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:

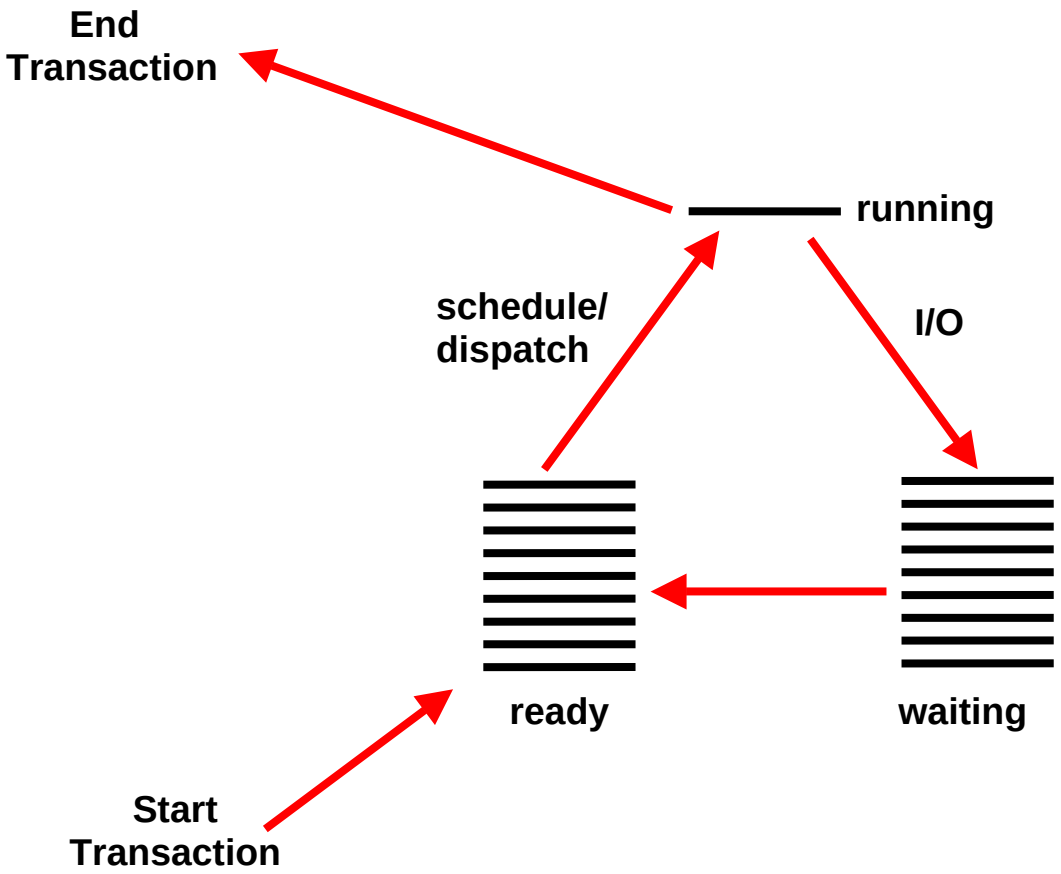
- **TCB-Modus, in der Regel als Task-Modus bezeichnet**
- **SRB-Modus (Service Request-Block Modus)**

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, Dies entspricht einem Unix-oder Windows Process Control Block (PCB). 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 System-Diensten nutzen.

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

Der SRB-Modus ist weitgehend ungenutzt außerhalb des Betriebssystems und außerhalb von Middleware-Produkten wie DB2. 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 Prozesse Control Blöcke ähnlich zu z/OS TCBs. Das Windows Äquivalent eines SRB wird als "Fibre" bezeichnet.



Transaction State

Traditionell laufen alle CICS Transaktionen in einem einzigen Adressraums unter der Steuerung eines einzigen Prozesses (dem CICS Nucleus) und einem einzigen TCB (genannt der QR TCB). Jedes Mal, wenn die Transaktion einen EXEC CICS-Befehl ausführt, wird die Steuerung an den CICS Nucleus transferiert. Der Programmierer muss sicherzustellen, dass die Pfad Länge zwischen EXEC CICS Kommandos kurz genug ist, um eine akzeptable Reaktionszeit (Antwortzeit) zu garantieren.

Ein in Java geschriebenes CICS-Programm benutzt keine EXEC CICS Befehle. Auch kann ein Java-Programm Ressourcen außerhalb der Kontrolle von CICS verwenden. RMI/IIOP ist ein Beispiel. Die Pfad Länge kann nicht gesteuert werden. Daher ist es nicht möglich, dass Java CICS Transaktionen mit einem QR TCB arbeiten.

CICS Quasi-reentrant TCB

Unter z/OS, existieren mehrere Varianten des TCB. Es gibt zwei verschiedene Alternativen für das Betreiben eines CICS Subsystems, wobei jede mit einer anderen Art von TCB arbeitet.

In der traditionellen Alternative laufen alle Anwendungsprogramme unter einer einzigen, CICS-managed, Task Control Block (TCB), dem so genannten „quasi-reentrant“ (QR) TCB. In diesem Fall müssen alle CICS Anwendungsprogramme quasi-reentrant geschrieben sein.

CICS quasi-reentrant Anwendungsprogramme werden durch den CICS Dispatcher unter dem QR TCB aufgerufen. Bei der Ausführung unter diesem TCB kann ein Programm sicher sein, dass keine anderes quasi-reentrant Programm ausgeführt werden kann, bis es die Kontrolle (die Verfügungsgewalt über die einzige CPU) aufgrund eines EXEC CICS-Befehl aufgibt. An diesem Punkt wird die Transaktion unterbrochen (Waiting oder Ready State). Während dieser Warte Zeit kann eine andere Transaktion das gleiche, quasi-reentrant Anwendungsprogramm ausführen. Dies bedeutet, ein quasi-reentrant Anwendungsprogramm kann gleichzeitig von mehr als einer Task benutzt werden, obwohl nur eine Task es in jedem Augenbkick ausführen kann.

Um sicherzustellen, dass Programme sich nicht mit der Nutzung des Arbeitsspeichers stören, untererhält CICS eine getrennte Kopie der Arbeitsspeichers für jede Transaktion. Wenn somit eine Kopie eines Anwendungsprogramms gleichzeitig von 11 Tasks benutzt wird, existieren 11 Kopien des Arbeitsspeichers in dem entsprechenden dynamischen CICS Speicherbereich (Dynamic Storage Area, DSA).

Achten Sie darauf, wenn ein Programm langwierige Berechnungen durchführt, weil ein Anwendungsprogramm die Kontrolle (Verfügungsgewalt über die CPU) von einem EXEC CICS-Befehl bis zum nächsten behält. Die Verarbeitung von anderen Transaktionen unter dem QR TCB ist während dieser Zeit ausgeschlossen. CICS terminiert eine Transaktion, die nicht vor Ablauf eines angegebenen Zeit Intervalls die Kontrolle abgibt..

Der CICS QR TCB bietet Schutz durch die ausschließliche Kontrolle der globalen Ressourcen nur dann, wenn alle Transaktionen, die auf diese Ressourcen zugreifen unter dem QR TCB laufen. Es bietet keine automatischen Schutz vor anderen Prozessen, die gleichzeitig unter einem anderen (offenen) TCB ausgeführt werden.

Open TCB

Wenn alle Transaktionen unter einem QR TCB laufen, impliziert dies, dass ein CICS Region nur mit einer einzigen CPU läuft. Es ist aus diesem Grund, dass mehrere Application Owning Regions (AOR) beliebt sind. Jede AOR läuft auf einer anderen CPU.

Das Open Transaction Environment (OTE) ist ein Umfeld, in dem

- **Eine einzelne CICS-Region mehrere CPUs benutzen kann, und**
- **CICS Anwendungscode nicht-CICS Dienste (Einrichtungen außerhalb des Umfangs der CICS API) innerhalb des CICS Adressraum ausführen kann, ohne Interferenz mit anderen Transaktionen.**

.

Das Problem ist, ruft eine Transaktion einen Service außerhalb der CICS-Adressraums auf, kann diese Transaktion für eine lange Zeit zu sperren, ohne dass der CICS Nucleus dessen bewusst sein muss. Aus diesem Grund läuft jede Anwendung (Transaction), die das Open Transaction Environment (OTE) nutzt, unter einem eigenen open TCB. In der CICS Region können mehrere aktive (parallel laufende) Programme mit eigenen open TCBs existieren anstatt eines einzigen aktiven Programms mit einem QR TCB. Während der QR TCB vom CICS Nucleus dispatched wird, erfolgt das open TCB Dispatching durch den z/OS Kernel. Wenn eine open TCB Transaktion einen non-CICS Service aufruft, die den TCB blockiert, hat dies keine Auswirkungen auf andere open TCB CICS Anwendungen.

Es gibt auch andere Verwendungen für das Open Transaction Environment. Allerdings hat das alles seinem Preis. Die Komplexität wächst, und die Performance kann schlechter werden. So ergänzt der offene TCB den QR TCB, aber ersetzt ihn nicht, und der QR TCB bleibt sehr beliebt. In einer gegebene z/OS-Installation können einige CICS Regionen mit dem QR TCB, und andere mit dem open TCB laufen.

Es existieren mehrere Varianten des open TCB. Einige von ihnen sind speziell für CICS Java-Anwendungen bestimmt.

Details unter http://jedi.informatik.uni-leipzig.de/de/VorlesMirror/ii/Vorles/OTE.pdf.

Programming CICS Applications in Java Enterprise Edition

Die meisten CICS-Anwendungen wurden bisher in Cobol, PL/I oder Assembler programmiert. C/C++ wurde weniger häufig verwendet. Mit der Einführung von Java wurde es möglich, CICS-Anwendungen auch in Java zu programmieren. Viele Experten gehen davon aus, dass Java die dominierende Sprache werden wird, um neue CICS-Anwendungen zu programmieren. Es gibt drei alternative Möglichkeiten, um CICS-Anwendungen in Java zu schreiben:

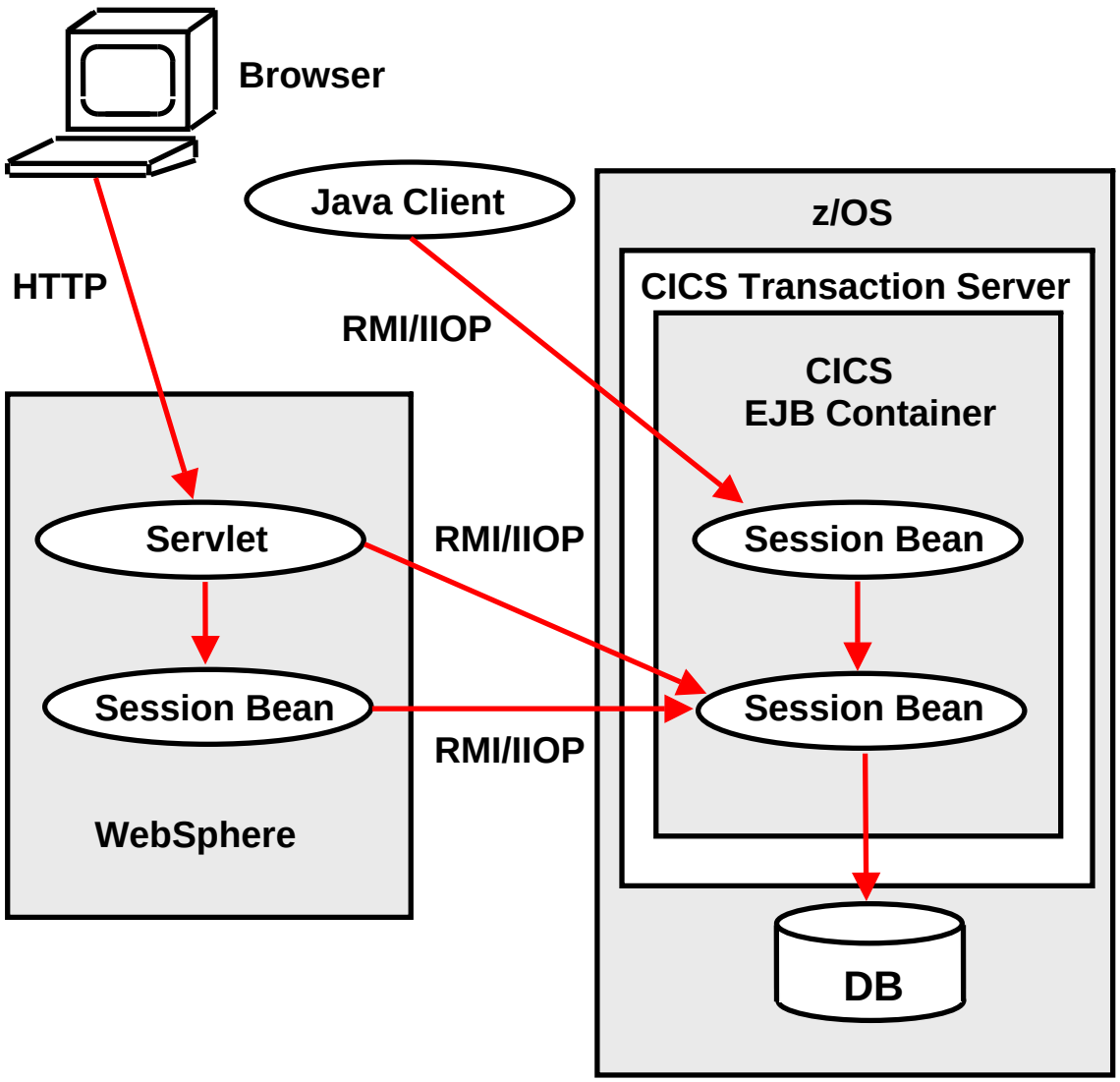
1. **Der z/OS "High Performance Java Compiler" hat eine Option, Objekt-Code wie ein Cobol oder PL/I Compiler zu generieren. Dieser Objekt Code wird gelinked und geladen wie jede andere Code. Der Java Quellcode ist auf andere Plattformen übertragbar, der Objekt-Code aber nicht. Die Java Architektur unterstützt diese Möglichkeit nicht, sie existiert außerhalb des Java Standards.**

2. **Alternativ kann der z/OS High Performance Java Compiler Byte Code generieren. Der Byte-Code wird innerhalb einer JVM, die als Teil des CICS Transaction Server installiert wird, ausgeführt. Es wird die JSE an Stelle der JEE benutzt.**

   Es gibt jedoch ein Problem mit diesem Ansatz. Die Java-Sprache erlaubt nicht den Einsatz von EXEC-Anweisungen. Deshalb wird die EXEC CICS-Schnittstelle durch eine Reihe von Java Bibliothek Aufrufe ersetzt, durch die „JCICS“ Schnittstelle.

3. **Beide bisher genannten Ansätze benutzen keinen EJB-Container. Wenn Sie JEE Einrichtungen mit CICS verwenden möchten, können Sie eine JEE EJB Container (mit einer eigenen JVM) innerhalb des CICS Transaction Servers installieren.**

Die letzten beiden Ansätze erfordern die Verwendung des open TCB, da die Java-Programme Einrichtungen außerhalb von CICS verwenden können.



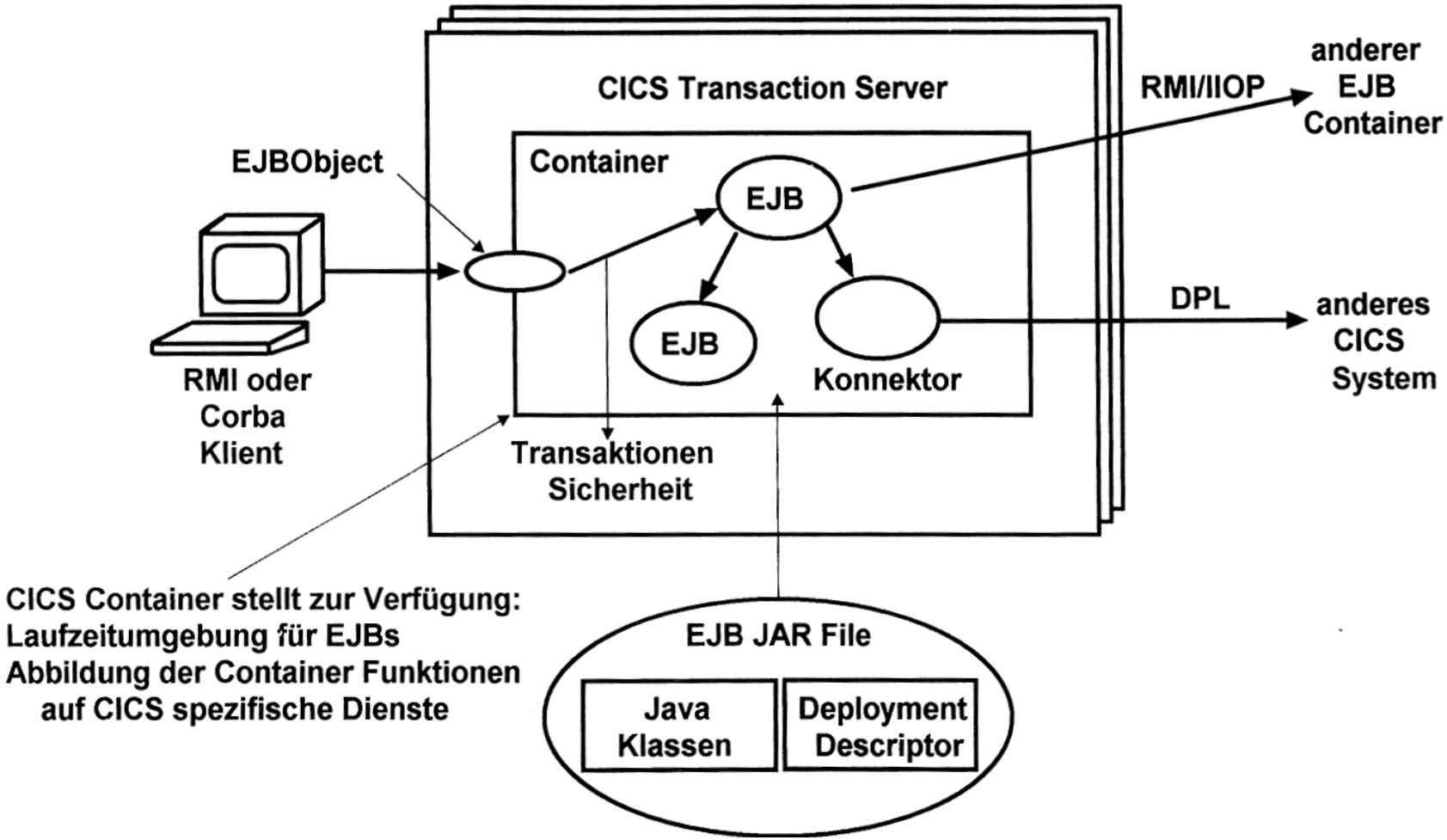
CICS bietet die Möglichkeit, einen EJB-Container zu installieren. Dies ist in Wirklichkeit ein Corba ORB. Es ermöglicht die Verwendung von RMI/IIOP.

CICS unterstützt keine Entity Beans. Es benutzt eigene Verfahren, um die Persistenz zu managen.

CICS Session Beans können über RMI/IIOP aufgerufen werden. Jede Java-Klasse oder jeder Corba Client ist in der Lage, mit RMI/IIOP auf eine CICS Session Bean zuzugreifen.

Der CICS EJB Container benutzt das X/Open Distributed Transaction Processing (DTP) Modell.

Benutzung von Enterprise Java Beans in einer CICS Anwendung



Im Vergleich zu anderen Umgebungen können Enterprise Beans in einem CICS EJB CICS Transaction Management Services wie System Log Management, Performance Optimization, Runaway und Deadlock Detection sowie Monitoring und Statistics nutzen.

Programmieren von CICS Transactionen in der Java Standard Edition

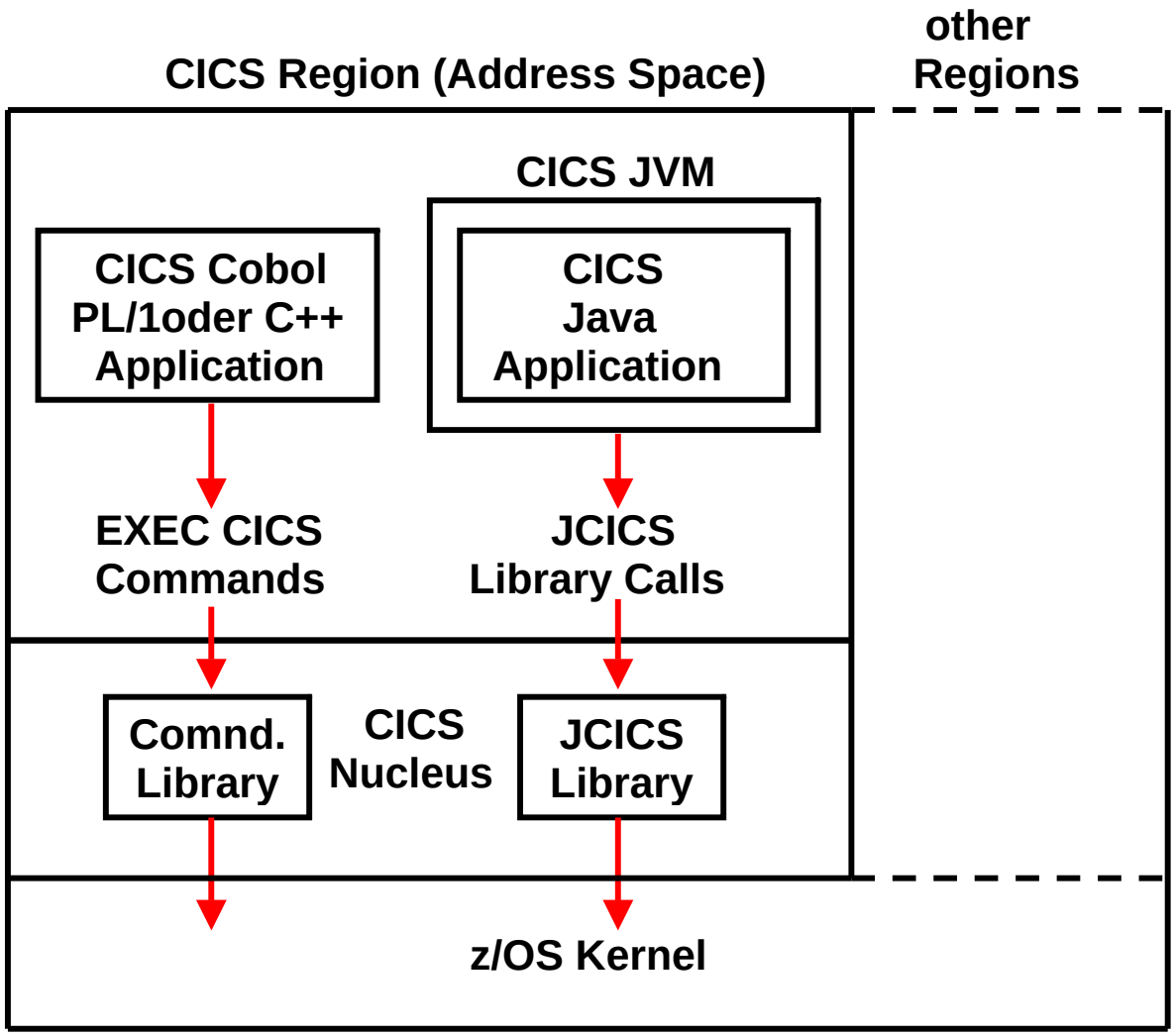
JEE Server wie WebSphere, Web Logic, Netweaver, und Open-Source-Produkte wie Jboss und Geronimo sind sehr beliebte Transaction Processing Produkte. CICS kann ebenfalls einen JEE-Server integrieren. Jedoch ist die JSE (Java Standard Edition) eine überraschend attraktive Alternative für CICS Java-Programme. Dies ist der Grund:

- **Ein JEE Server integriert einen Transaktion Monitor in eine bestehendes JEE Infrastruktur, die viele Funktionen aufweist, die nicht direkt mit der Transaktionsverarbeitung zu tun haben. JNDI und RMI/IIOP sind Beispiele. Viele dieser Funktionen werden von CICS eigentlich nicht benötigt, und stellen einen überflüssigen Overhead dar.**
- **CICS integriert Java in einer vorhandenen Transaktion Processing Infrastruktur. OTE ermöglicht es, JSE Programmen direkt unter CICS laufen zu lassen.**

Wie alle anderen Java-Programme laufen CICS JSE Programme innerhalb einer JVM. Sie können Seite an Seite mit Cobol, PL/I und C/C++ CICS Programmen ausgeführt werden.

Java hat aber keine Einrichtungen um die EXEC CICS Aufrufe zu implementieren. Deshalb benutzen Java-Programme statt dessen die JCiCS Bibliotheksroutinen. Die JCICS Bibilotheksaufrufe ersetzen eins zu eins die EXEC CICS-Befehle und nehmen damit den Platz des EXEC CICS Befehle ein.

Dieser Ansatz wird in der folgenden Abbildung dargestellt.



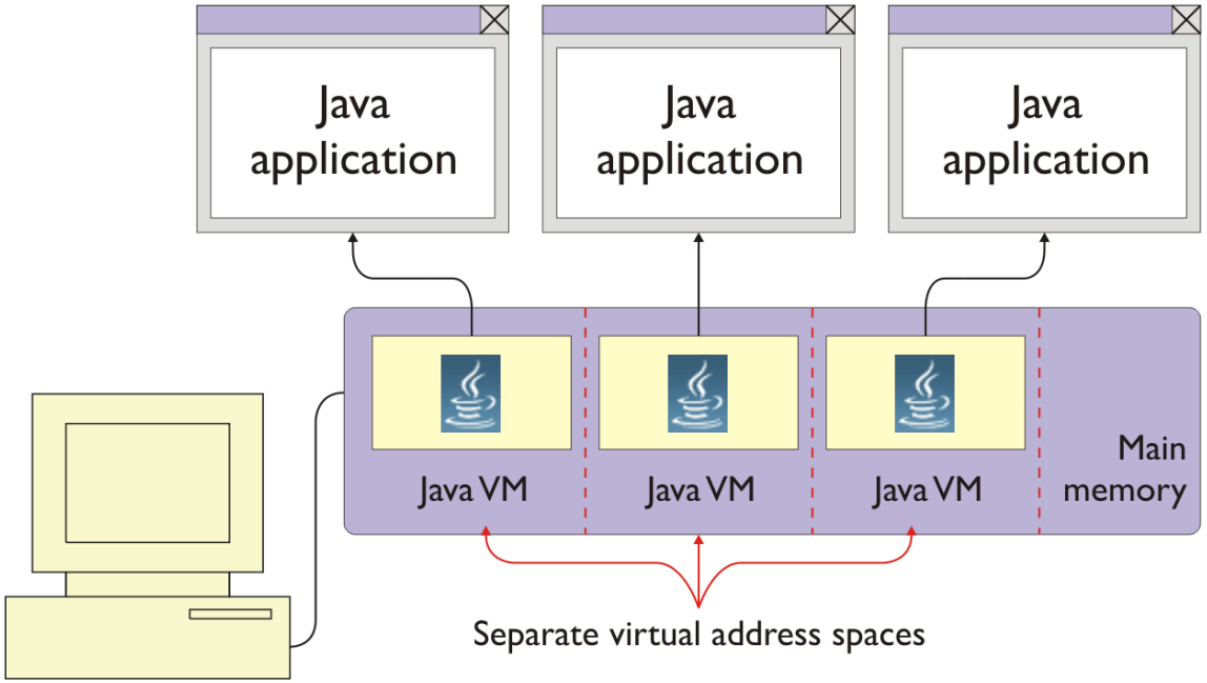
Benutzen Sie Java wie jede andere Programmiersprache. Die JCICS Library Calls ersetzen eins zu eins die EXEC CICS Commands.

Dieser Ansatz benutzt keine EJBs..

Siehe das Redbook
Java Application Development for CICS

http://publib-b.boulder.ibm.com/abstracts/sg245275.html?Open

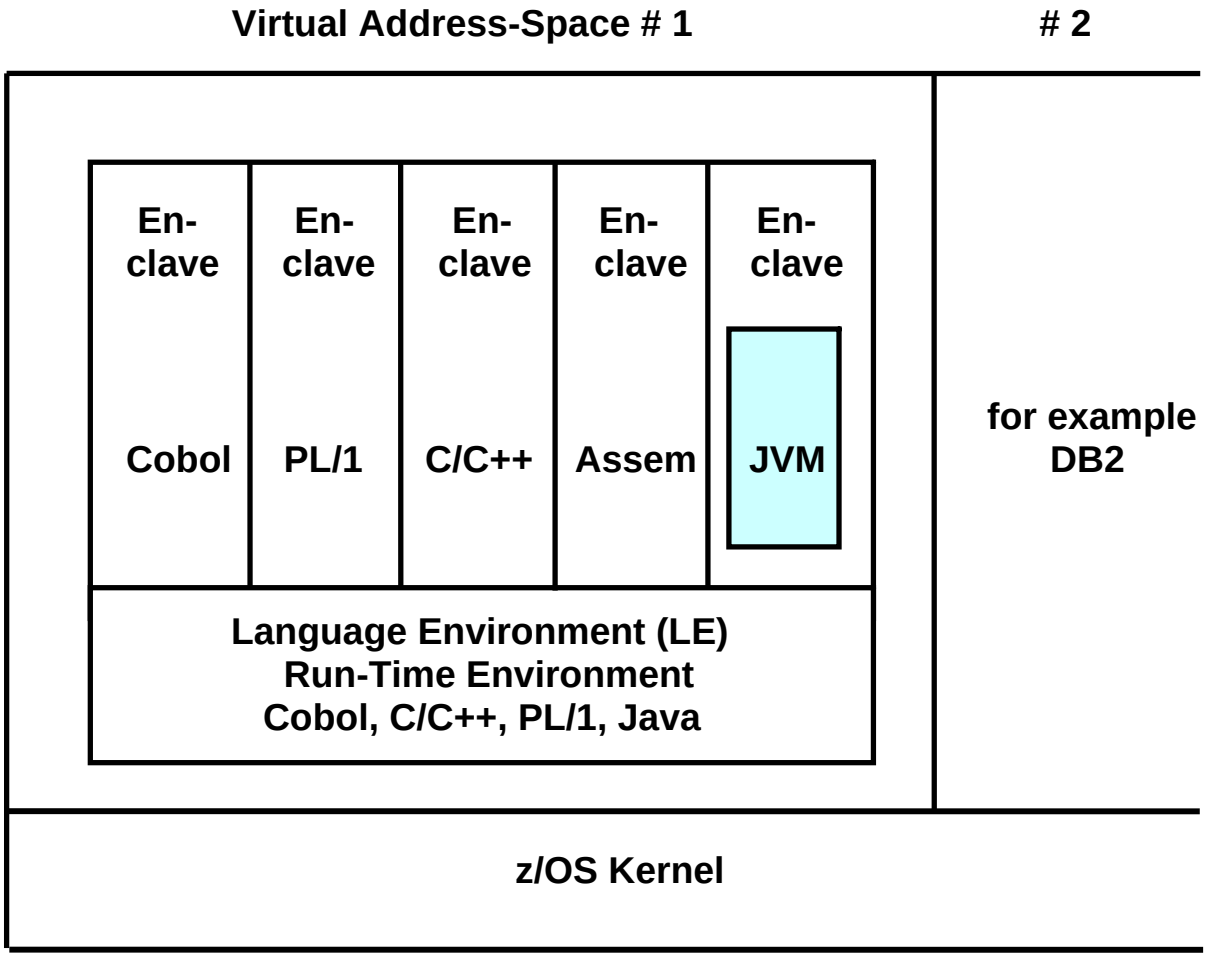
JCICS Java Anwendungsprogrammierung



Parallele Ausführung von JSE Java Transactionen

Der einfachste Weg, JSE Transaktionen parallel auszuführen ist, jeder Transaktion eine separaten JVM zuzuordnen, und eine getrennte z/OS Region jeder JVM. Das ist ziemlich aufwendig in Bezug auf den Verbrauch von CPU-Zyklen. Unter CICS laufen daher alle JSE-Anwendungen wie Cobol und PL/I Anwendungen innerhalb eines einzigen Adressraums.

Es gibt zwei JSE Alternativen, dies zu tun: die JVM Pool Architektur und die JVM Server Architektur. Wir diskutieren die JVM Pool Architektur zuerst.



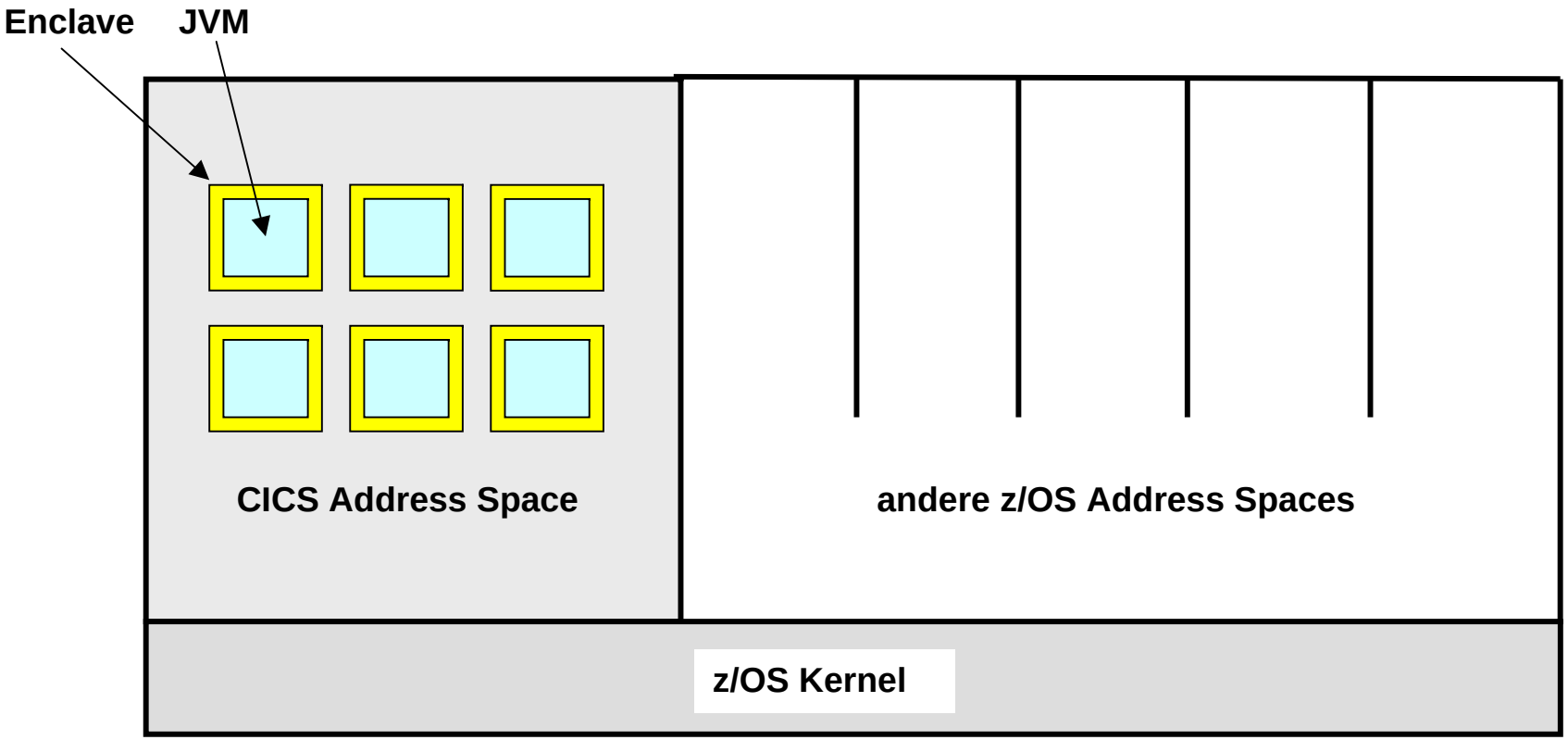
CICS-Transaktionen, die in Cobol, C/C++, PL/I oder Assembler geschrieben sind, laufen in getrennten Enclaves; eine Enclave für jede Transaktion.

CICS Java-Transaktionen werden in getrennten JVMs ausgeführt, jeweils eine JVM pro Enclave. Java Threads werden nicht benutzt.

Cobol, PL/I, C/C++ und Java Enclaves können gleichzeitig innerhalb einer CICS-Region laufen.

CICS Java Standard Edition

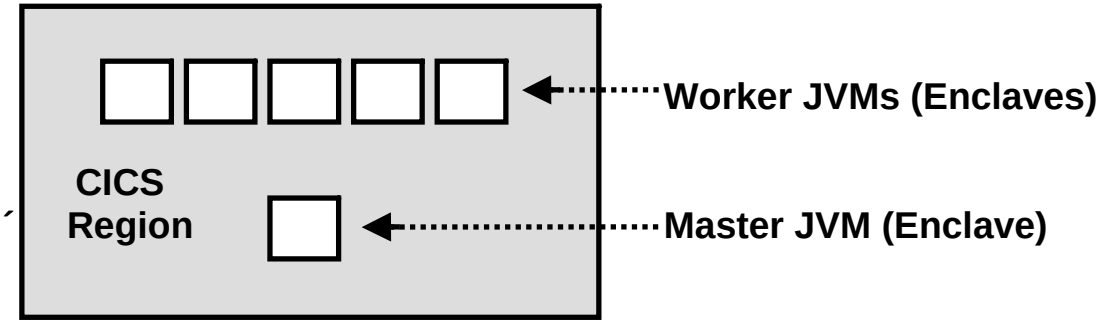
Java Enclaves können in der gleichen CICS Region parallel zu Cobol, PL/I oder C/C++ Enclaves ausgeführt werden.



CICS JVM Pool Architektur

Ein wichtiger Punkt ist die gegenseitige Isolation von mehreren Java Transaktionen (das „I“ in ACID) zu gewährleisten. Die JVM Pool Architektur ist ein gradliniger Ansatz. Hier unterhält CICS mehrere JVMs in seinen Adressraum, als JVM Pool bezeichnet. Einzelne CICS Tasks werden jeweils einer JVM in dem Pool zugeordnet, und nur eine einzige Java Anwendung läuft in jeder JVM. Dies garantiert die Isolation zwischen den Java Anwendungen.

Jede JVM läuft in einem eigenen Enclave. Theoretisch sind > 100 JVMs möglich. Allerdings wird viel Speicherplatz benötigt (~ 20MByte + Heap). Die Anzahl der gleichzeitig laufenden Transaktionen ist durch die Anzahl der JVMs begrenzt, die in eine Region passen. Das Ergebnis ist max. 20 gleichzeitige JVMs in einer 2 GByte Region.



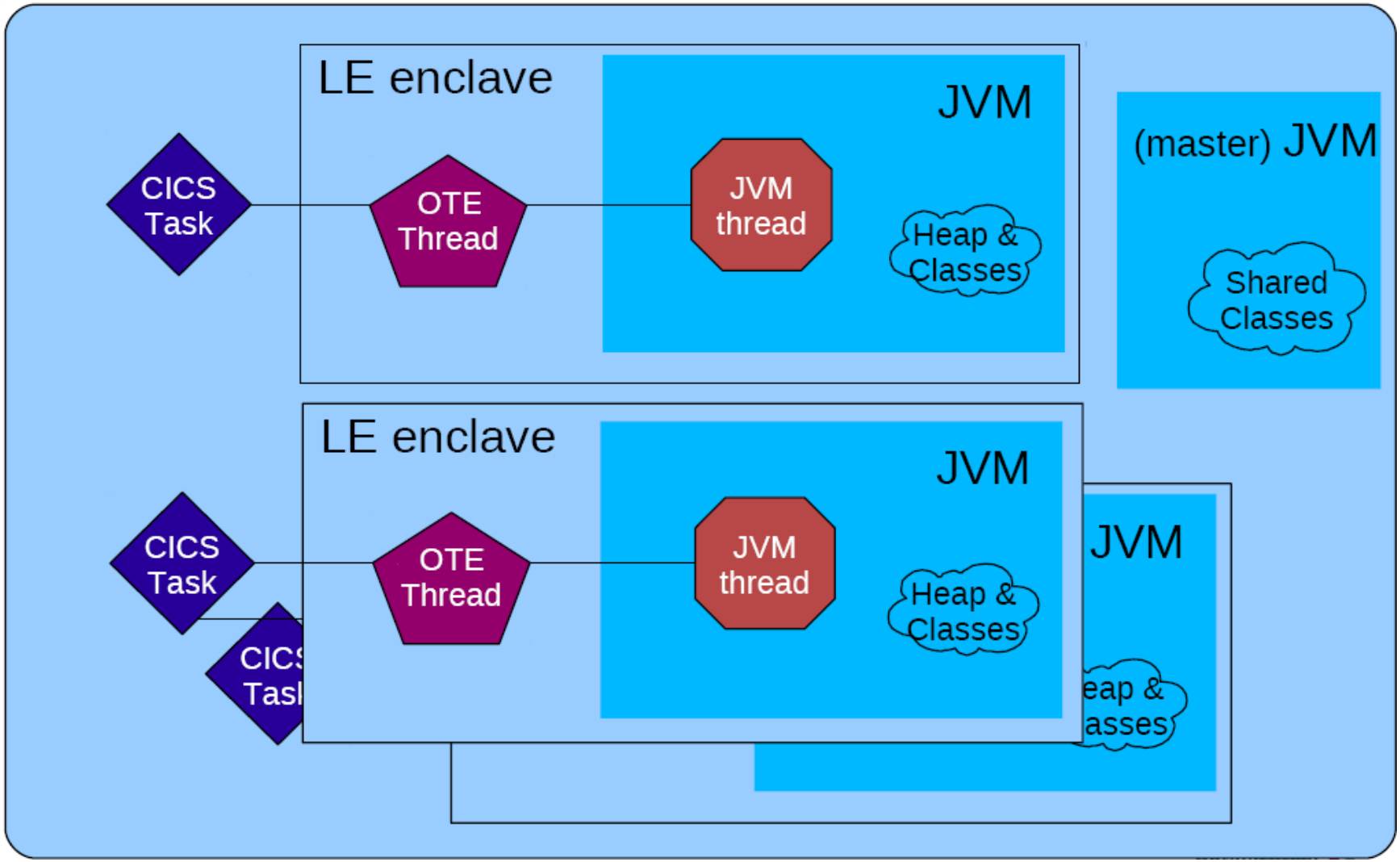
LE Enclaves

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. Spezifisch ist es hiermit möglich, mehrere JVMs innerhalb eines CICS Adressenraums laufen zu lassen.

Es existiert immer eine Enclave für jede JVM. Eine Master JVM steuert mehrere Worker JVMs. Transaktionen werden in den Worker JVMs ausgeführt.

CICS unterhält eine „Shared Class Cache Facility für die JVM. Mehrere JVMs können gemeinsam eine einzige Cache mit Class Files nutzen. Die gemeinsam genutzte Cache ersetzt den System-Heap und die Anwendungsklassen für die JVMs. JVMs, die gemeinsam genutzte Klassen verwenden starten schneller und haben einen geringere Speicherplatzbedarf als JVMs, die dies nicht tun.

Die Master JVM initialisiert u.A. den Shared Class Cache. Die Master JVM wird nicht für die Ausführung von Java Anwendungen benutzt.

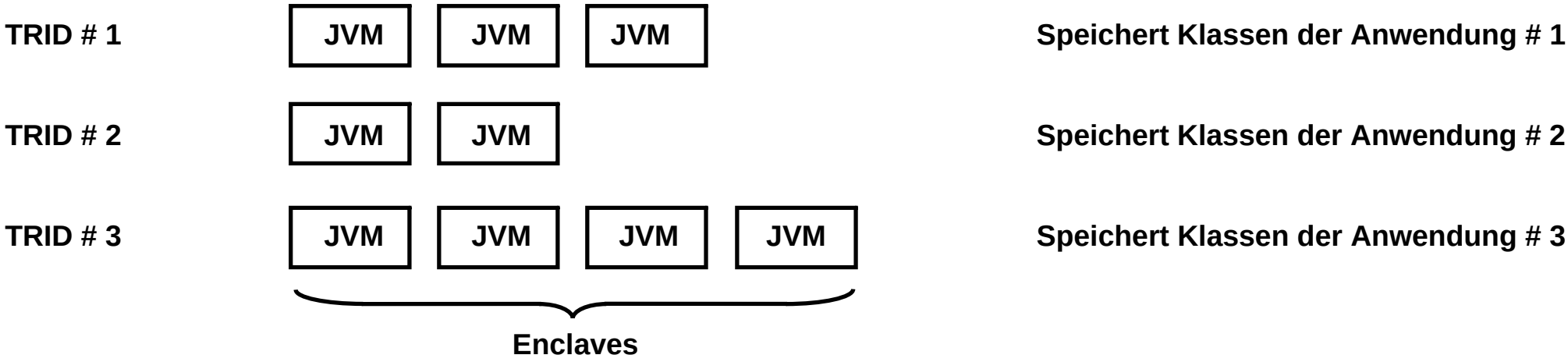


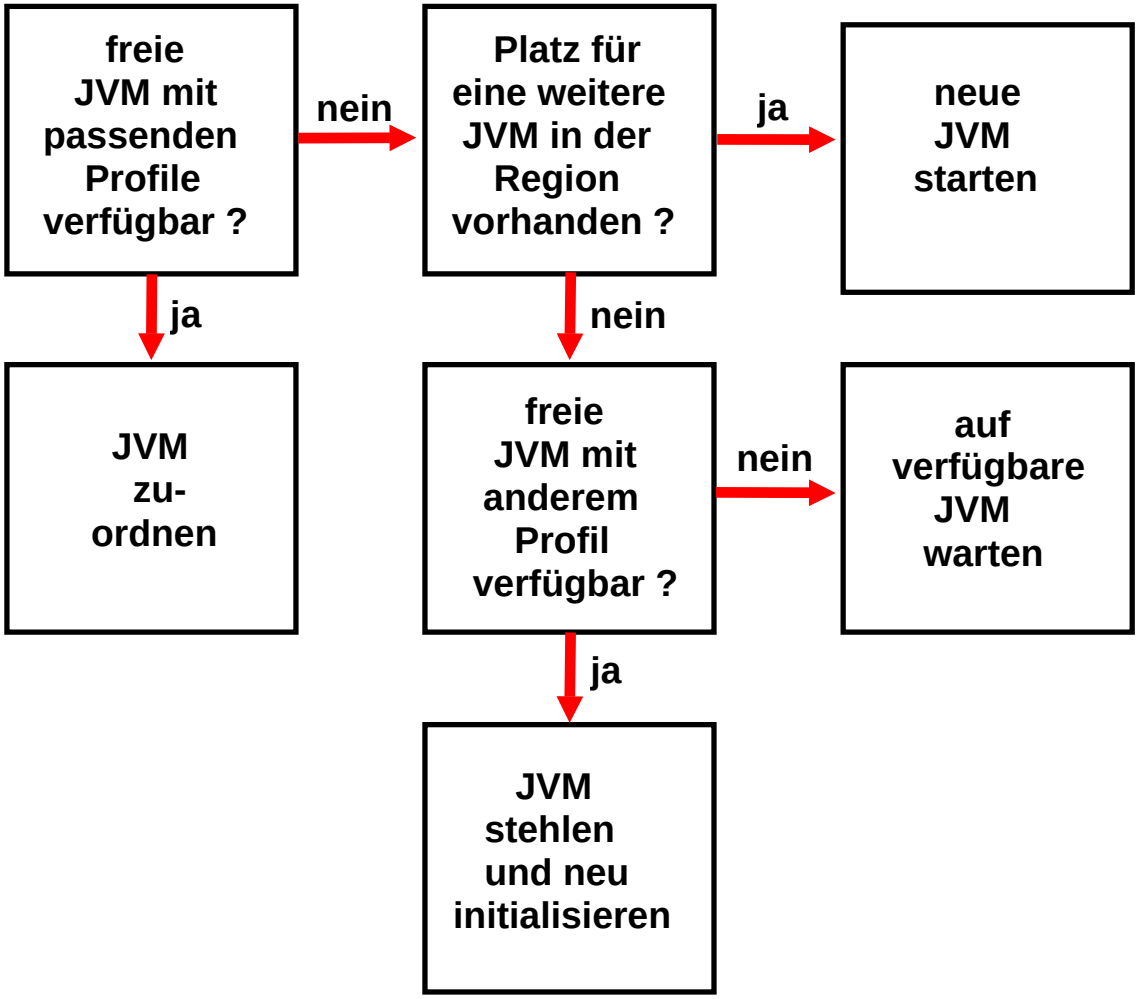
Mehrere JVMs sind in jeweils in getrennten Enclaves untergebracht, und laufen dort als ein Open Transaction Environment (OTE) Thread. Eine Master JVM cloned je nach Bedarf mehrere Worker JVMs.

JVM Zuordnung

Wenn eine neue Transaktion eintrifft, sucht CICS eine freie Enclave mit Ihrer vorinstallierten JVM, um in dieser die Transaktion auszuführen. Wenn möglich, sucht CICS eine JVM aus, in der die gleiche Anwendung, gekennzeichnet durch ihre TRID, bereits gelaufen ist. In diesem Fall müssen die Anwendungsklassen nicht neu geladen werden.

Nehmen wir an, in einer Java CICS Region sind 20 JVMs vorhanden. In jeder JVM sind die Anwendungsklassen einer spezifischen Java CICS Anwendung gespeichert. Häufig sind die Klassen einer Anwendung in mehr als einer JVM gespeichert.

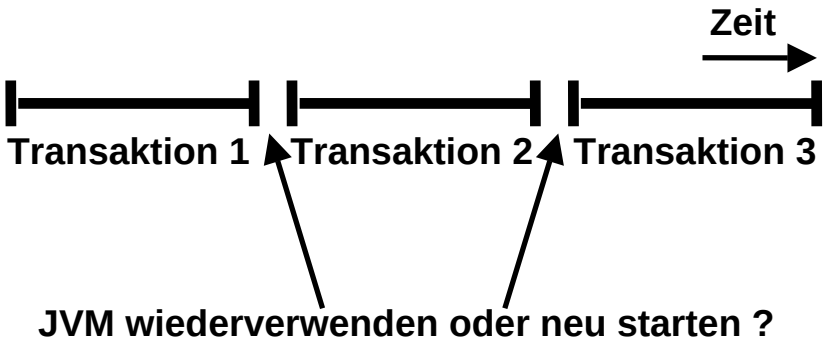




Dargestellt ist der Algorithmus, mit dem beim Starten einer weiteren Transaktion eine passende JVM für deren Bearbeitung ausgewählt wird.

Ein bestimmter Transaktionstyp wird durch seine TRID gekennzeichnet, und kann von mehreren Transaktionen parallel ausgeführt werden. Unterschiedliche Transaktionstypen unterscheiden sich durch unterschiedliche Anwendungsklassen. Wenn eine neue Transaktion gestartet wird, wird nach Möglichkeit eine freie JVM ausgewählt, in der die richtigen Anwendungsklassen bereits geladen sind. Wenn das nicht möglich ist, werden die benötigten Anwendungsklassen in eine verfügbare Worker JVM nachgeladen.

Ausführung einer Folge von Transaktionen



Eine Transaktion kann die Ausführung der Folge-Transaktion beeinflussen, indem sie den Zustand (State) der JVM ändert. Beispiele für eventuelle sicherheitskritische Reste der vorhergehenden Transaktion

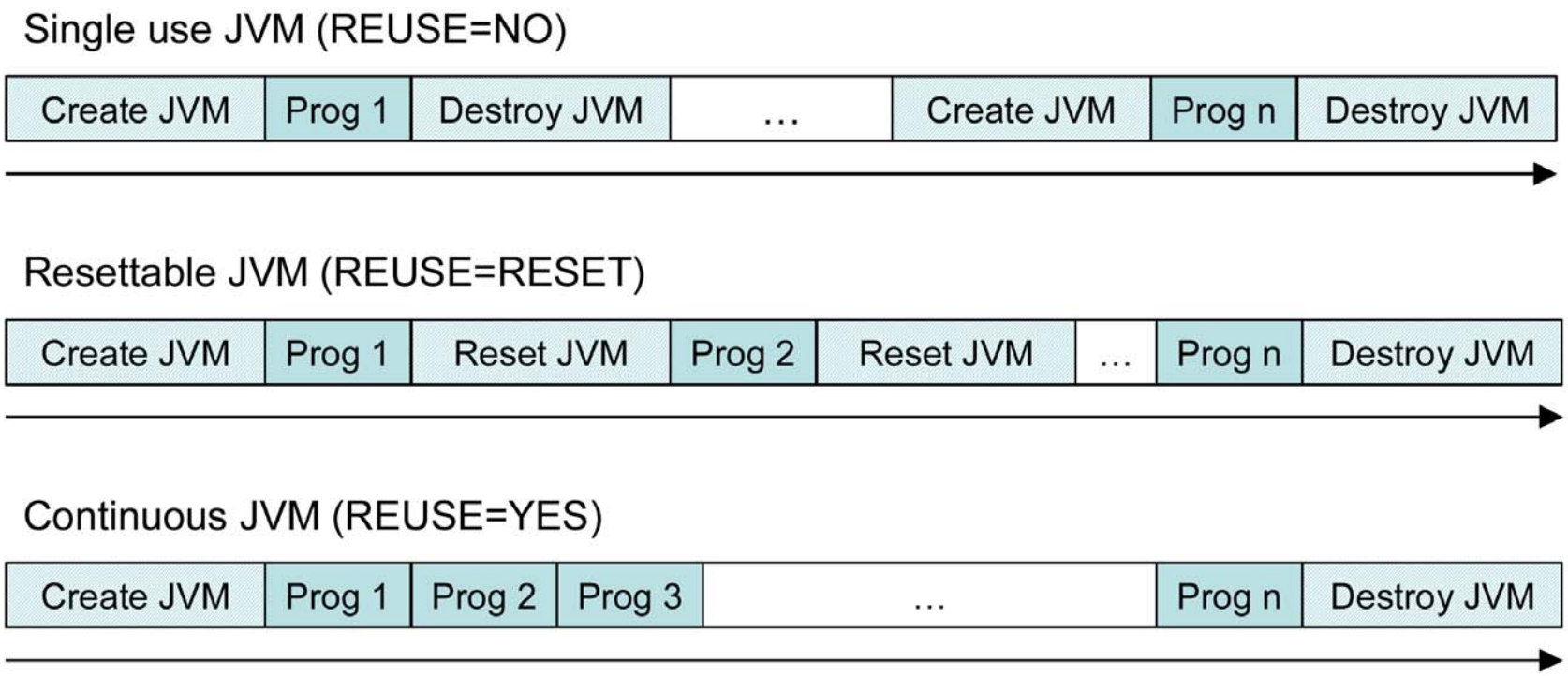
- **überschriebene statische Variablen**
- **Starten von Threads**
- **geladene native Bibliotheken.**

Klassischer Ansatz: Für jede Transaktion wird eine neue JVM gestartet und nach Abschluss der Transaktion wieder beendet.

Das Hoch- und Herunterfahren einer JVM hat jedoch einen erheblichen Zeitaufwand zur Folge. Bei jeder Initialisierung einer JVM werden

- **60 System Klassen geladen,**
- **700 Array-Objekte und**
- **1000 non-Array-Objekte allokiert und angelegt.**

Pfadlänge bis zu 100 Millionen Maschinenbefehle, ca. 0,1 Sekunde Verarbeitungsdauer, sind möglich.



CICS JVM Modus

In the CICS JVM Pool Architecture kann eine JVM can in einem der folgenden Modi laufen:

1. **Single use mode. Die JVM wird zerstört, und für die nächste Transaktion wird eine neue JVM gestartet.**
2. **Resettable mode. Eine Zusatzeinrichtung bewirkt, dass der ursprüngliche Zustand der JVM wiederhergestellt wird.**
3. **Continuous mode. Es ist Aufgabe des Anwendungs-Programmierers, sicherzustellen, den dass der Zustand der JVM nicht verändert wird..**

In welchem Modus die JVM läuft wird durch den REUSE Parameter in dem assoziierten JVM Profile definiert.

Resettable Modus

Der Single Use Modus hat den Nachteil, dass der Aufwand für das Starten einer JVM sehr hoch ist. Dies hat zur Folge, dass der Single Use Modus aus Performance Gründen nur in Ausnahmefällen eingesetzt werden kann.

Der Continuous Use Modus hat das Problem, dass die JVM nach Beendigung einer Transaktion sich im gleichen Zustand wie zu Beginn einer Transaktion befinden muss. Dies hat der Anwendungsprogrammierer zu gewährleisten. Was hierzu zu tun ist, ist jedoch nicht ausreichend dokumentiert, und erfordert sehr tief gehende Java Kenntnisse, über die ein durchnittlicher Java Programmierer nicht notwendigerweise verfügt.

Dieses Problem löst der Resettable Modus, der jedoch nicht Teil des JEE Standards ist, und somit eine unerwünschte IBM proprietäre Erweiterung des Standards darstellt ist.

Dr. Jens Müller hat in seiner Diplomarbeit: " Anwendungs- und Transaktionsisolation unter Java ", Mai 2005, eine detaillierte Untersuchung der JVM Isolationseigenschaften durchgeführt. Viele der Isolationsprobleme können durch eine sorgfältige Programmierung umgangen werden. Im Vergleich zu Cobol CICS Programmen ist dies jedoch eine zusätzliche Herausforderung.

CICS JVM Server Architecture

Der "JVM Server" ist eine neue Implementierung der JVM, die es erlaubt, mehrere CICS-Transaktionen als Threads innerhalb einer einzigen JVM auszuführen. Sie wird möglicherweise die JVM Pool Architektur ablösen.

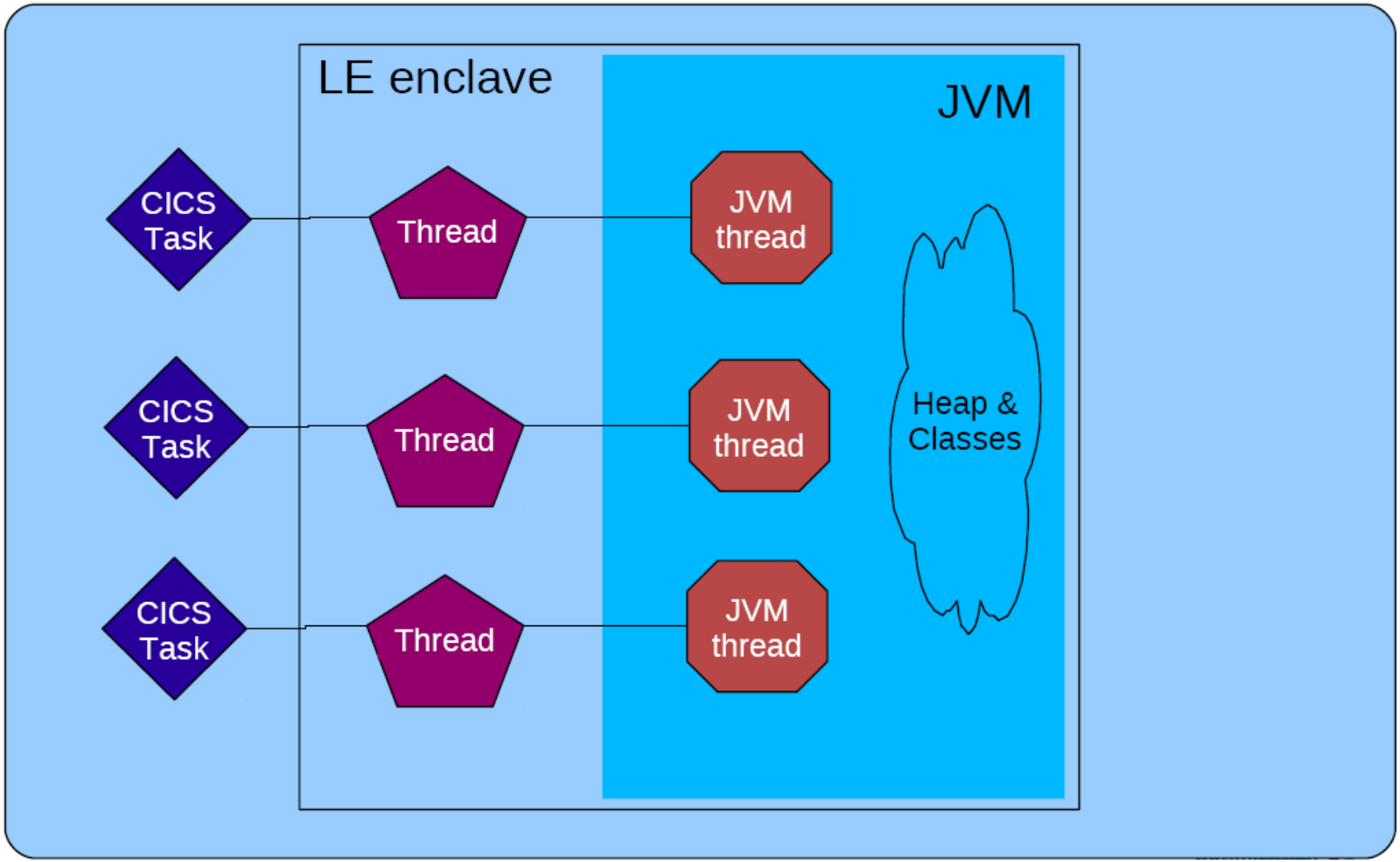
Über viele Jahre benutzten CICS und WebSphere die gleiche, als „Sovereign“ bezeichnete Implementierung der JVM. Unterschiedliche Optimierungspotentiale führten dazu, dass WAS heute die „J9“ JVM verwendet. Diese und die Sun „HotSpot“ JVM gelten heute als die performantesten verfügbaren JVMs. Die J9 wird auch in der JSE und der JME eingesetzt.

CICS entwickelte statt dessen die „JVM Server“ JVM, die spezifisch für die CICS Bedürfnisse optimiert wurde. Die JVM Server JVM ermöglicht es, mehrere Transaktionen als Java Threads innerhalb einer einzigen JVM laufen zu lassen.

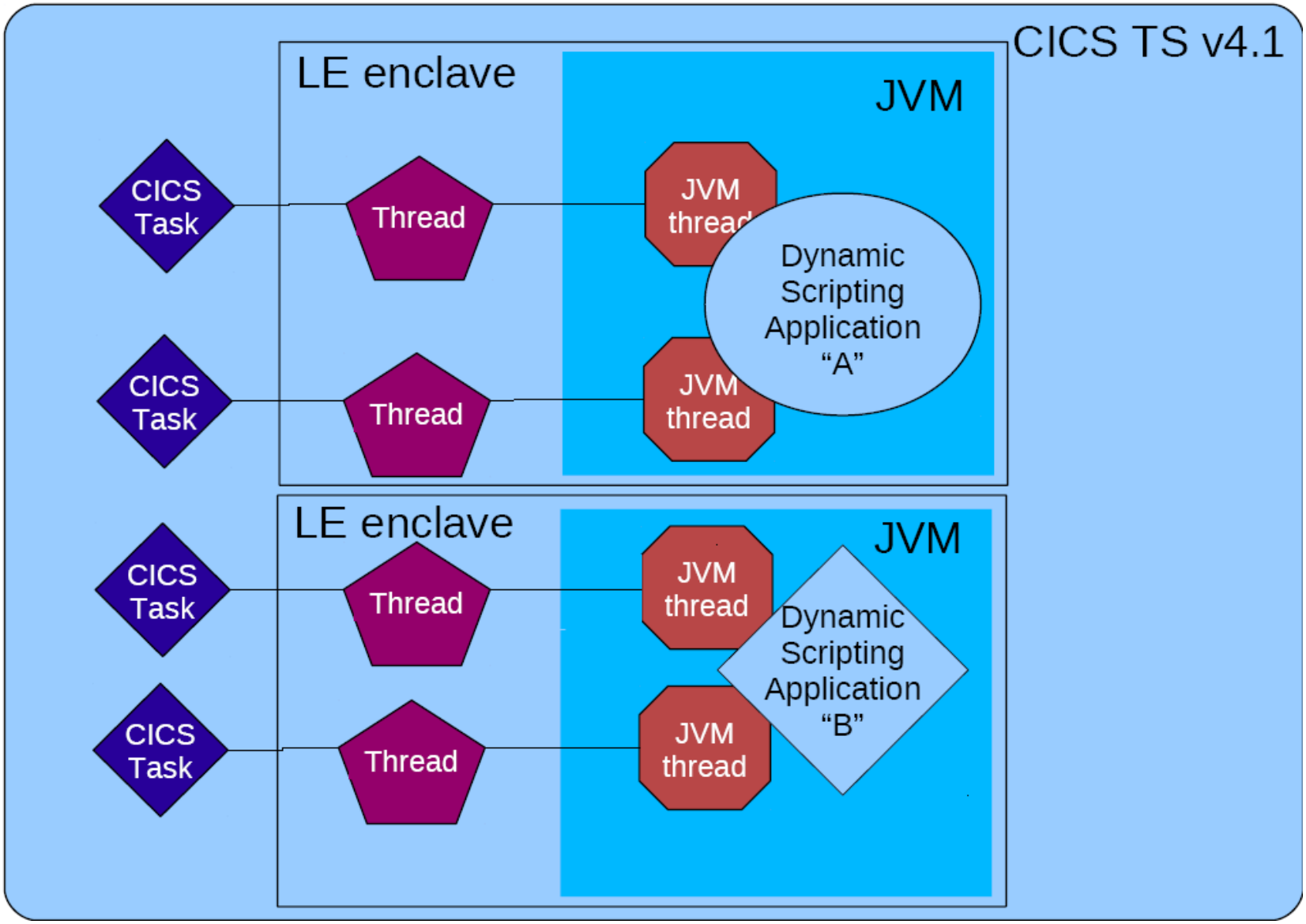
Die JVM Server Umgebung ermöglicht die gegenseitig Isolation von mehreren Java Anwendungen innerhalb einer JVM durch den Einsatz von Industrie-Standard OSGi-Bundles. Das OSGi-Framework innerhalb des JVM Servers bietet erforderliche Quarantäneeinrichtungen für mehrere gleichzeitig ausgeführte CICS Java Programme innerhalb der gleichen JVM. Dazu müssen CICS Java-Anwendungen in einem OSGi-Bundle verpackt werden, und dann als CICS BUNDLE Ressource mit Verweis auf die OSGi-Bundle deployed werden.

Der JVM-Server bietet auch integrierte Statistikfunktion zur Überwachung der JVM Garbage Collection. Dies erleichtert das Performance Tuning von Java CICS Anwendungen.

Der OSGI Ansatz wird auch von dem Android Betriebssystem als Laufzeitumgebung für Java Apps verwendet, siehe http://felix.apache.org/site/presentations.data/OSGi%20on%20Google%20Android%20using%20Apache%20Felix.pdf, oder als Mirror unter http://jedi.informatik.uni-leipzig.de/de/VorlesMirror/ii/Vorles/OSGI.pdf



In der JVM Server Architektur laufeb mehrere CICS Tasks als unabhängige JVM Threads innerhalb einer einzigen JVM. Jeder JVM Thread wird auf einen LE Enclave Thread abgebildet.



Es ist möglich, mehrere JVM Server in einem einzigen CICS Adressenraum auszuführen, genauso wie bei der CICS JVM Pool Architektur . Eine praktische Grenze ist etwa 20 JVMs innerhalb eines CICS Adressenraums.

Facit

Wie vollständig ist die Isolierung der Threads innerhalb des JVM Servers ?

Besser als alles, was es vorher gab, aber nicht so vollständig, wie man es gerne hätte.

Robert Harbach hat in seiner Master Thesis: "CICS JVM Server Application Isolation", März 2012, eine detaillierte Untersuchung der JVM Server Isolationseigenschaften durchgeführt. Während die Einführung des OSGi-Frameworks viele Probleme löst, müssen eine Reihe von Isolation Fragen noch geklärt werden. Eine sorgfältige Programmierung kann diese Probleme umgehen. Das Entwicklungsrisiko wächst jedoch mit der Größe und Komplexität einer neuen CICS Java-Anwendung.

Die Master Thesis kann heruntergeladen werden unter
http://www-ti.informatik.uni-tuebingen.de/~spruth/DiplArb/harbach.pdf

Es ist eine dringende Erfordernis, dass eine zukünftige Version des Java Standards das Thread Isolationsproblem der JVM an der Wurzel anpackt, und eine zufriedenstellende Lösung anbietet. Dieses und verwandte Probleme haben dazu geführt, dass sich Java bisher nur für die Präsentationslogik von neuen Java Anwendungen durchgesetzt hat, während der Business Logik teil häufig immer noch in Cobol oder PL/I programmiert wird.