

## 5. Parallel Sysplex

kapitel52.docph/26.4.01

### 5.1 Coupling Facility (CF)-Architektur

Ein Sysplex-Cluster kann bis zu 32 Systeme (SMPs) integrieren, wobei ein SMP bis zu 16 CPUs enthalten kann (s. Kapitel 2). Cluster-Konfigurationen besitzen im allgemeinen signifikante Nachteile bezüglich des Leistungsverhaltens. Die Implementierung eines Symmetric Multiprocessors mit 16 CPUs führt bereits zu erheblichen Problemen, da die erreichbare Performance zwischen 8 und 10 CPUs infolge des zunehmenden Overheads abnimmt. In der Grobstruktur des Parallel Sysplex (Abbildung 1) kommt zu den Kommunikations-Problemen innerhalb eines SMPs noch diejenigen der SMPs untereinander. In der /390-Cluster-Architektur kommunizieren die einzelnen SMPs über ein Hochgeschwindigkeits-Netzwerk, das als Crossbar-Switch implementiert wird, miteinander.

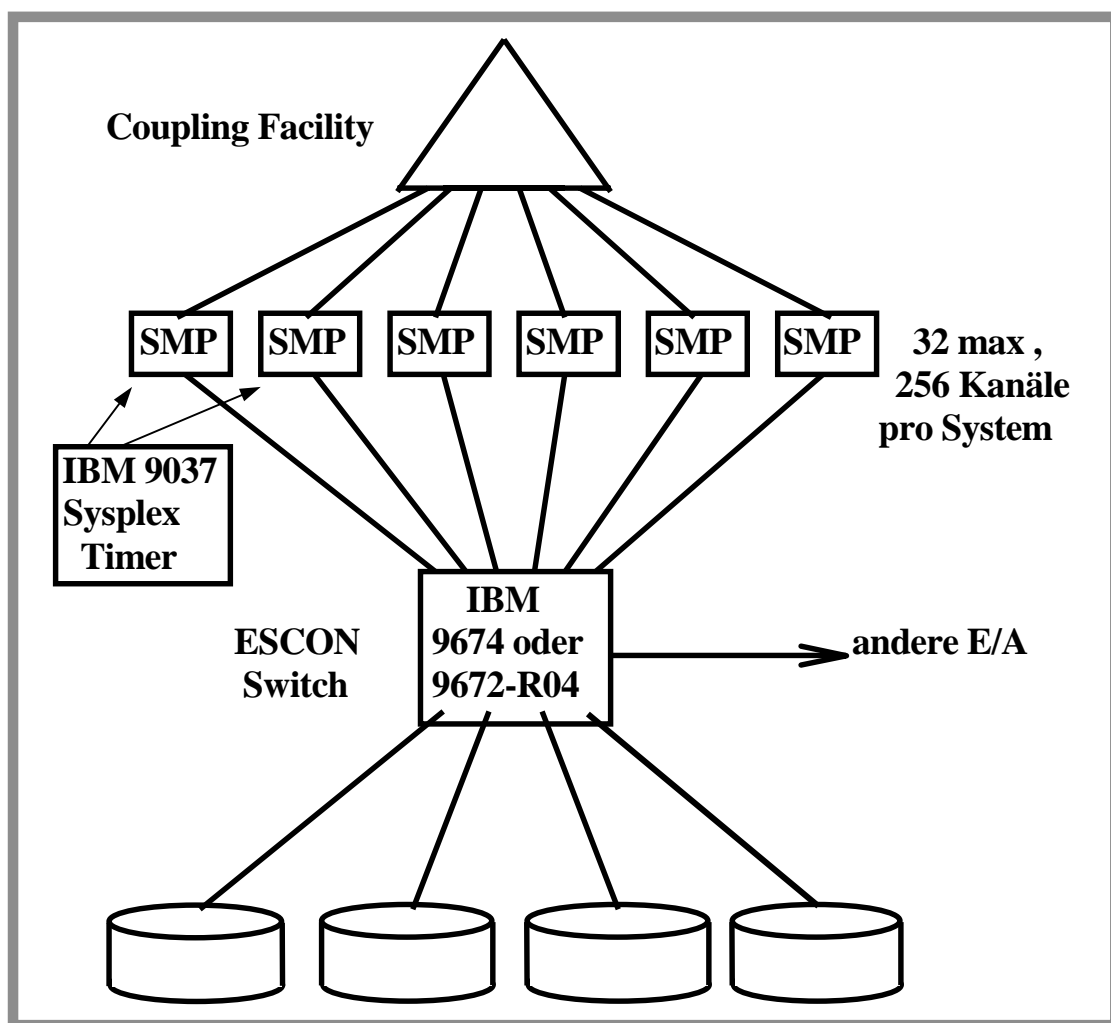


Abbildung 1: Parallel Sysplex

Die Verbindung der SMPs zu den Plattengeräten erfolgt über ein oder mehrere ESCON-Switches. Da jedes System 256 Ein-/Ausgabe-Kanäle enthalten kann, hat diese Verbindungsstruktur die Aufgabe, bei 32 Systemen maximal  $256 * 32 = 2^{13}$  Kanäle zu verwalten.

Den SMPs steht für die Kommunikation ein globaler Speicher innerhalb der CF zur Verfügung. Die CF enthält außerdem einen eigenen Mikroprozessor, der eine Vielzahl von Steuerfunktionen übernimmt. Dazu gehören u.a. Data Sharing und Dynamic Workload Balancing zwischen den CPUs innerhalb eines Systems sowie zwischen den Systemen im Cluster, Dynamic Workload Dispatching sowie die Steuerung eines Hot Standby Systems. Die Verbindung eines Systems mit der CF erfolgt über die Coupling Support Facility, Glasfaser-Kabel (100 MByte/s) und einen Hardware-Assist (Abbildung 2).

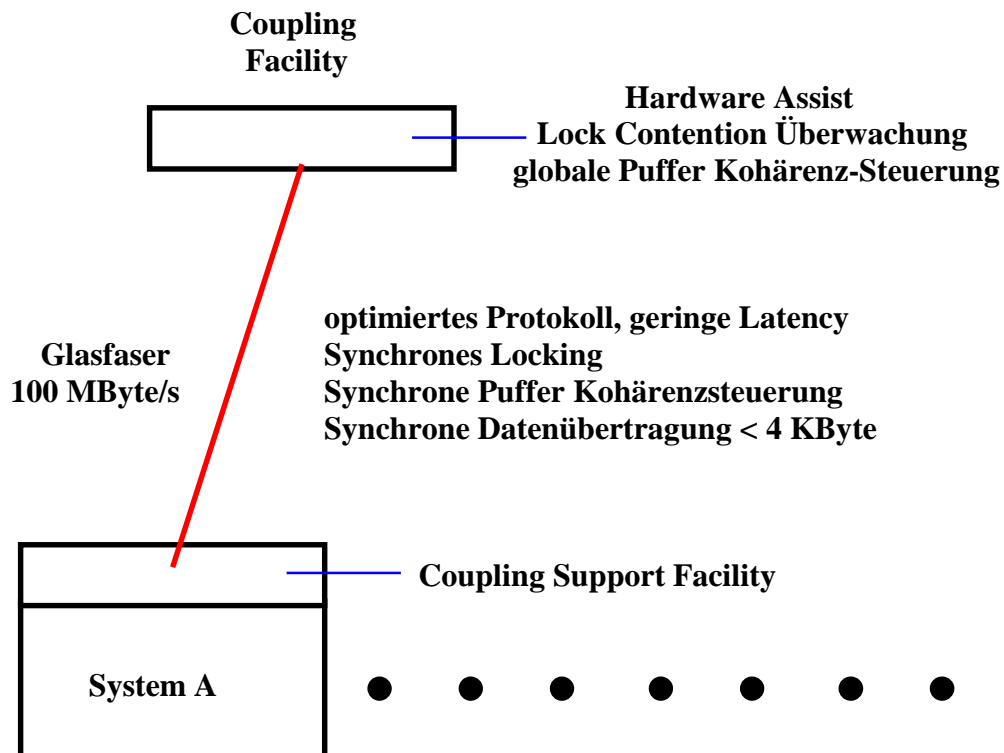


Abbildung 2: Anbindung eines Systems an die Coupling Facility

Die CF implementiert drei unterschiedliche Funktions-Typen: Lock-, Cache-, List-Funktion. Jede Funktion besitzt eine entsprechende Struktur, die im Speicher der Coupling Facility untergebracht ist.

Die Lock-Funktion liefert den Mechanismus, um das Problem der globalen Parallelität mit der Shared Data-Architektur zu lösen. Die Lock-Struktur enthält Lock-Table-Einträge, die durch Software des Datenbank-Managers auf Blöcke oder Records abgebildet werden. Jeder Lock-Table-Eintrag enthält "Shared"- oder "Exklusiv"-Hinweise für jedes System. Wenn ein System ein Lock für einen Daten-Eintrag wünscht, wendet es sich an die CF, die in Abhängigkeit vom augenblicklichen Zustand des Locks dieses dem System übergibt oder nicht. Sollte das angeforderte Lock einen inkompatiblen Zustand generieren, so muss es durch den entsprechenden Lock-Manager bearbeitet werden. In 99% der Fälle wird das Lock unmittelbar dem interessierten System gewährt.

Die Cache-Funktion der CF wird zur Behandlung der Datenbankpuffer-Kohärenz verwendet. Durch diese Kohärenz ist jeder Knoten in dem Sysplex in der Lage, Daten aus der gemeinsamen benutzten globalen Datenbank in seinem lokalen Cache zu referenzieren. Die Cache-Struktur besteht aus zwei Teilen: Directory-Einträge und optionale Daten-Elemente. Ein Directory-Eintrag existiert für jeden Daten-Block, der sich entweder in dem lokalen Puffer-Pool irgendeines Systems oder in einem globalen Pool befindet. Jeder Directory-Eintrag enthält einen Hinweis darauf, welche Systeme momentan Kopien des entsprechenden Daten-Blocks besitzen. In dem geschützten Speicher jedes Systems ist ein Bit-Vektor definiert, wobei ein Bit zu jedem lokalen Daten-Puffer gehört. Beim Lesen eines Daten-Blocks von einer gemeinsam benutzten Datenbank speichert der Datenbank-Manager den Daten-Block-Namen und den damit verbundenen lokalen Bit-Vektor-Offset bezüglich des CF-Directory-Eintrags und setzt den lokalen Bit-Vektor-Wert, um einen gültigen Daten-Block anzuzeigen. Sollte der Datenbank-Manager auf einem anderen System

diesen Datenblock aktualisieren, wird das Update der Coupling Facility mitgeteilt. Die CF fragt ihr Directory ab und sendet "Invalidate Signals" an die Systeme, die im Augenblick Kopien dieses Daten-Blocks haben. Das "Invalidate Signal" bewirkt, dass ein dem Datenblock entsprechendes lokales Bit des Bit-Vektors gesetzt und damit ein ungültiger Zustand signalisiert wird. Immer, wenn ein Datenbank-Manager einen lokal gepufferten Datenblock benutzen möchte, zeigt ein einfacher Bit-Test die Gültigkeit an. In dem Fall, dass dieser Test einen ungültigen Zustand feststellt, nimmt der Datenbank-Manager einen Refresh der lokalen Kopie dieses Datenblocks vor. Die Datenblöcke selber können auch in dem optionalen Datenelement-Teil der CF-Cache-Struktur gespeichert werden. Eine solche Maßnahme hat den Vorteil, dass eine lokale Kopie eines Datenblocks, die sich als ungültig erweist, aus der CF heraus aktualisiert werden kann. Bei vielen Shared-Data-Architekturen wird die Leistung geringer infolge des zunehmenden Overheads der Puffer-Datenungültigkeit. Das CF-Cache-Modell löst dagegen dieses Problem erfolgreich und effizient. Die CF-List-Funktion liefert ein Mehrzweck-Queuing-Konstrukt mit einem breiten Anwendungsbereich. Eine List-Struktur enthält ein oder mehrere List-Header und List-Elemente. Die Elemente können mit Hilfe von LIFO/FIFO oder Key-Sequenced Ordering hinzugefügt oder entfernt werden. Von besonderem Interesse sind Listen-Übergänge z.B. von "empty" zu "nonempty". List-Strukturen werden für Intersystem-Messaging und Workload-Distribution benutzt. Mit Hilfe von List-Strukturen können Instanzen eines Workload-Managers im Parallel Sysplex periodisch Performance-Status-Informationen austauschen. Diese werden wiederum für dynamische Routing-Transaktionen von überbelasteten zu unterbelasteten Systemen ausgenutzt.

## 5.2 Work Load Manager

S/390-Großinstallationen bestehen in der Regel aus mehreren Systemen, auf denen mehrere Instanzen (Kopien) der Betriebssysteme (z.B. OS/390, VM, Unix) installiert sind. In dem Sysplex sind weiterhin eine größere Anzahl von Crossbar (ESCON)-Switche, hunderte von Steuereinheiten und Devices (Plattenspeicher, Magnetbandgeräte, Drucker, Bildschirme) usw. implementiert. Die Anforderungen an dieses Rechnersystem sind sehr unterschiedlicher Natur: Vordergrund-, Hintergrund-Betrieb, Transaktionen verschiedener Typen, Entwicklungs- und Wartungs-Arbeiten, Leistungsmessungen usw. Diese Aufgaben müssen noch in unterschiedlichen Prioritäten abgearbeitet werden unter der Nebenbedingung, dass zu verschiedenen Zeiten sich die Rechnerbelastung bezüglich dieser Komponenten permanent ändert, d.h. tagsüber wird die interaktive Arbeit dominieren und die Stapelverarbeitung dagegen gering ausfallen, nachts ist dieses Verhältnis umgekehrt. Für eine möglichst optimale Aufteilung der Rechnerbelastung zu den vorhandenen Ressourcen bezüglich bestmöglicher Performance ist eine Software-Komponente verantwortlich: Der Work Load Manager (WLM). Diese Komponente läuft unter jedem Betriebssystem und bildet eine der wichtigsten Steuerfunktionen der CF.

Dem WLM ist eine weitere Komponente unterstellt (Abbildung 3), der sogenannte System Ressource Manager (SRM). Dieser stellt eine Menge von Routinen zur Verfügung, die alle angeschlossenen Systeme beobachten. Dazu gehört die Messung der individuellen CPU-Auslastung, d.h. es wird die Zahl der CPU-Zyklen, die tatsächlich ausgenutzt werden, verfolgt. Eine andere Routine bestimmt die Hauptspeicher-Nutzung. Daraus resultiert die Maßnahme, möglicherweise die Anzahl der laufenden Prozesse zu reduzieren, wenn die Nutzung des Hauptspeichers zu hoch wird. Das hätte zur Folge, dass die Seitenwechselrate zu groß wird und die Performance damit abnimmt. Das System-Monitoring erstreckt sich außerdem insbesondere auf die Ein-/Ausgabe-Belastung (Channel Pathes, Control Units), die Kommunikation der Systeme untereinander bzw. den Kommunikations-Overhead sowie die Belastung der Plattenspeicher. Aus diesen Ergebnissen, die der SRM permanent liefert, werden Strategien festgelegt, die laufende Anwendungen auf die vorhandenen Ressourcen verteilen. Eine rein statische Zuordnung erscheint wenig sinnvoll, weil sich die genannten Belastungen dynamisch ändern. Die Komplexität spezifischer Low-Level-Steuerungen zur Verteilung der System-Ressourcen führte dazu, dass der Systemadministrator bestimmte Ziele für die Arbeit im System formuliert und danach Zuordnungs-Algorithmen das Tuning der System-Ressourcen übernehmen. Der WLM implementiert diese Algorithmen anstatt des Systemadministrators und nimmt die optimale Aufteilung der zur Verfügung stehenden Ressourcen vor.

Der Work Load Manager muss dabei zwei unterschiedliche Funktionen ausführen. Die erste besteht darin, die

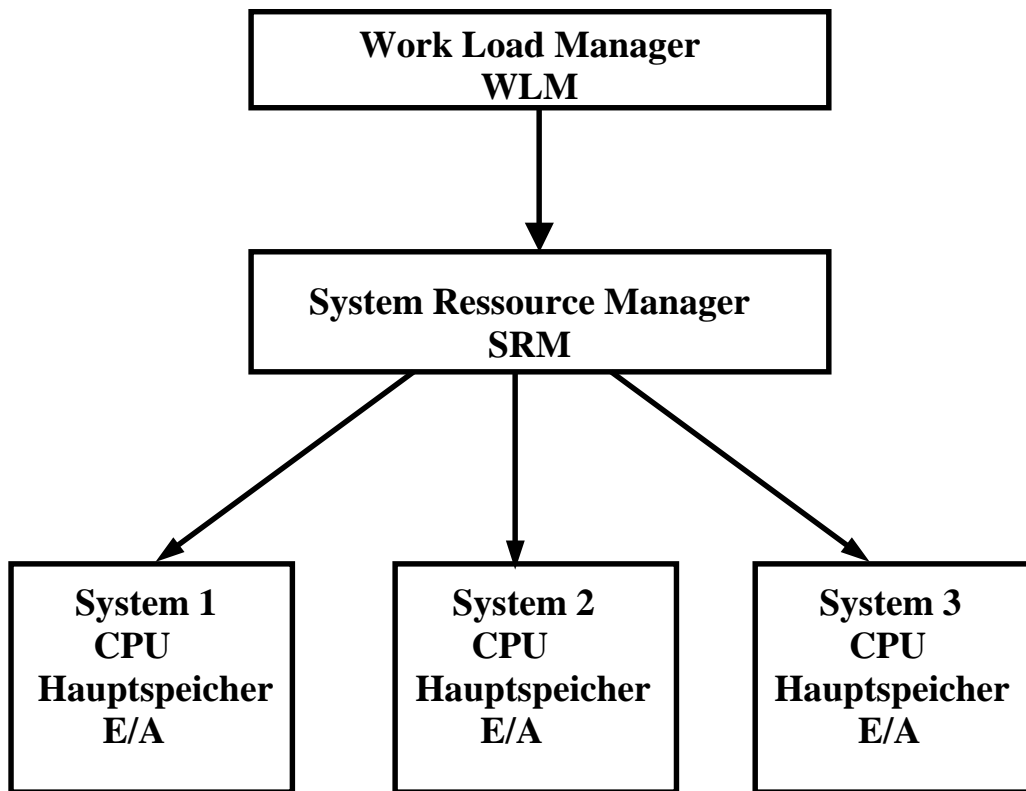


Abbildung3: System Ressource Manager

gesamten Arbeitsanforderungen an das Rechnersystem in voneinander getrennte Klassen aufzuteilen. Diese Klassen heißen Service-Klassen. Die Klassifikation basiert auf den Attributen einer individuellen Arbeitsanforderung. Letztere kann das UserID , das aufgerufene Transaktions-Programm, die Arbeits-Umgebung oder das Subsystem, an das die Anforderung gerichtet ist, usw. einschließen. Der WLM ist mit Hilfe des Multisystem Goal-Driven Performance Controller (MGDPC) in der Lage anzugeben, welche Anwendungsklasse mit welcher Arbeitsanforderung verbunden werden muss. Diese Zuordnung erfolgt durch die angegebenen Attribut-Werte und der entsprechenden Service-Klasse. Jede Service-Klasse repräsentiert die Arbeitsanforderungen mit identischen Business-Performance-Zielen. Das fundamentale Problem besteht darin, dass der Ressourcen-Bedarf der meisten Arbeitsanforderungen zu Beginn der Ausführungszeit unbekannt ist, er kann aber variieren in Abhängigkeit von Parametern, die nur zur Ausführungszeit bekannt sind. In der Abbildung 4 ist der Fall dargestellt, dass 5 verschiedene Arbeitsanforderungen an den Sysplex gestellt werden. Dabei sind unterschiedliche Ziele für die Anforderungen vorgegeben, z.B. sollen bei der CICS-Transaktion 90% der Antwortzeiten kleiner als 0,3 s betragen, beim Stapel 1 werden für 90 % eine Verarbeitungszeit von weniger als 3 Stunden verlangt usw.

## OS/390

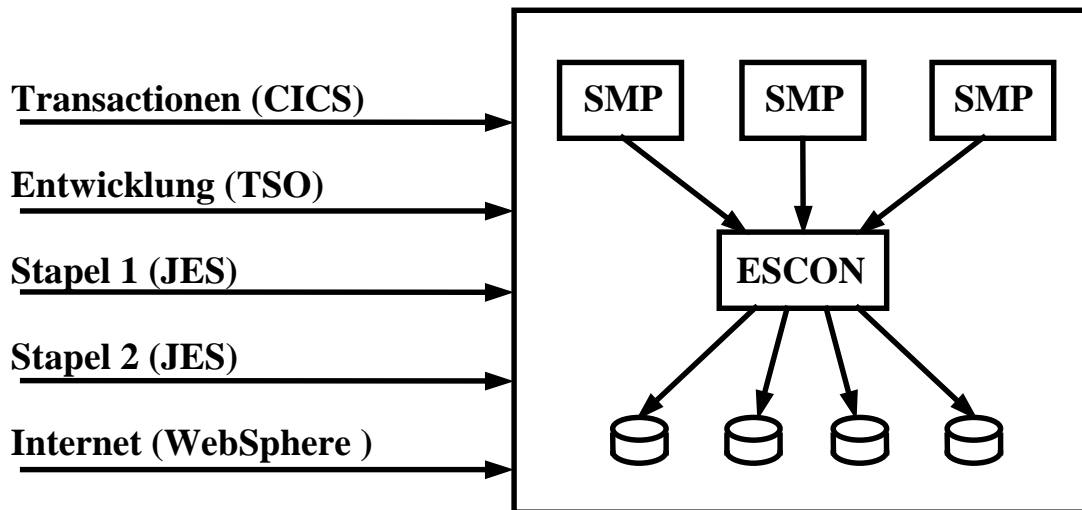
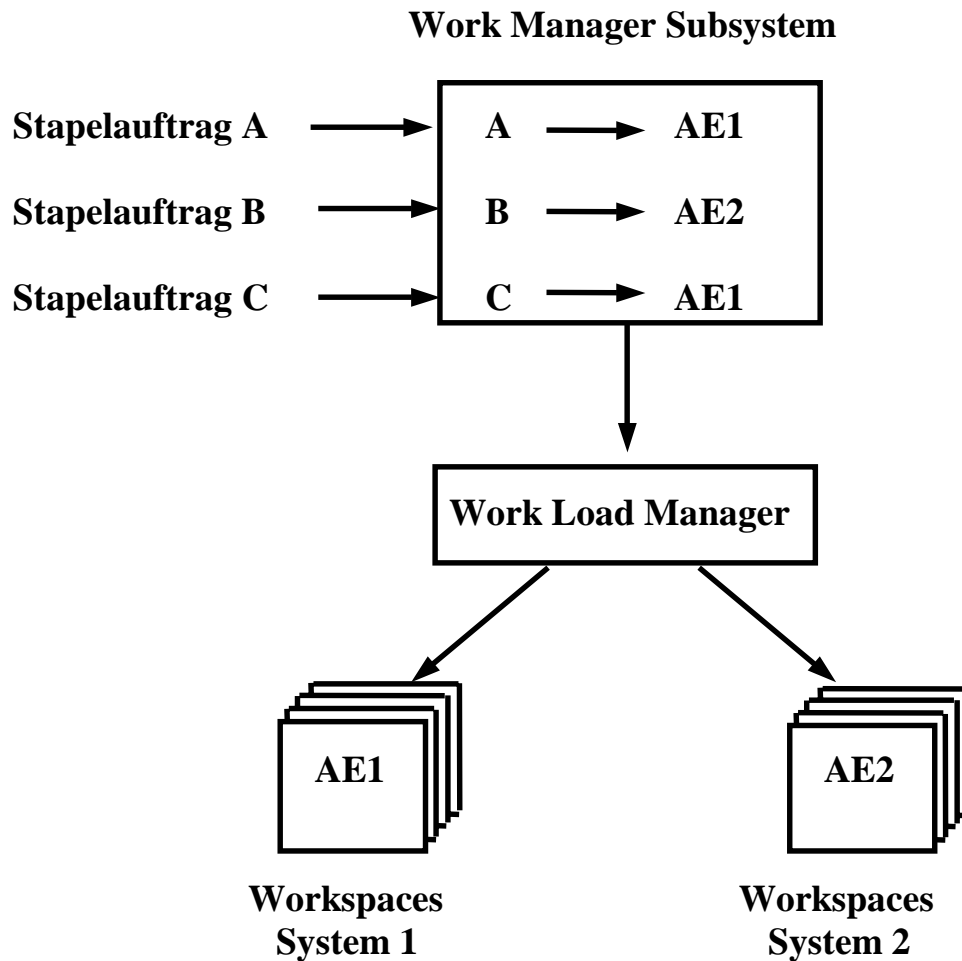


Abbildung 4: Work Load Manager-Klassifikations-Regeln

Eine Service-Klasse enthält eine Folge von Perioden mit einem Installations-abhängigen Wert, der angibt, wie lange eine Arbeitsanforderung zu einer Periode gehört. Jede Arbeitsanforderung beginnt in der Periode 1 und wird solange entsprechend dem Ziel in der 1. Periode behandelt, bis genügend Service in Anspruch genommen und die Perioden-Dauer überschritten wurde. Danach beginnt für die Arbeitsanforderung die 2. Periode, in der sie entsprechend dem Ziel der 2. Periode bearbeitet wird usw. Jede Periode hat ein mit ihr verbundenes Ziel und eine spezielle Bedeutung. Der Perioden-Dauer können unterschiedliche Werte für verschiedene Service-Klassen zugeordnet werden. Gleichfalls können die Ziele für eine gegebene Periode in unterschiedlichen Service-Klassen verschieden sein. Eine WLM-Installation kann 3 Hauptziel-Typen für Arbeitsanforderungen spezifizieren: Antwortzeit, Stellenwert, Geschwindigkeit.

Die zweite Funktion, die der WLM übernehmen muss, besteht in der Auswahl des Stellenwertes. Der Stellenwert bezieht sich auf eine relative Bearbeitungs-Rangfolge, in der die gewählten Algorithmen entscheiden müssen, welche Ziele bei einer System-Ressourcen-Reallokation zuerst verfolgt werden sollen. Die Algorithmen kümmern sich um die Bearbeitungs-Ziele mit dem höchsten Stellenwert, bevor sie sich den weniger wichtigen zuwenden.

Die Verteilung von Arbeitsaufträgen auf die einzelnen Systeme ist in der Abbildung 5 dargestellt. Die Aufträge werden von dem Work-Manager-Subsystem entgegengenommen und anschließend vom WLM den Systemen zugeordnet. Der WLM reduziert einerseits die Notwendigkeit für spezifisches Detailwissen des Administrators und andererseits den Aufwand für die System-Administration insgesamt.



*Abbildung 5: Zuordnung von Aufträgen zu Systemen*

### 5.3 Lock-Architektur

Eine der wichtigsten Funktionen der Coupling Facility (CF) bildet die Steuerung der Kommunikation der einzelnen Systeme, die selbst bis zu 16 CPUs integrieren können. Die CF realisiert einen eigenen Rechner mit einem spezifischen Betriebssystem, das auf die Inter- und Intra-System-Kommunikation optimiert ist. Der Rechner stellt einen Hochgeschwindigkeits-Ein-/Ausgabe-Prozessor mit einem relativ großen Hauptspeicher dar. Das Betriebssystem der CF arbeitet bezüglich der Kommunikation mit einem synchronen Protokoll. Letzteres hat zur Folge, dass ein gleichzeitiger Zugriff von zwei verschiedenen Verarbeitungseinheiten auf dieselbe Ressource mittels einer Locking-Funktion aufgelöst wird und kein Prozess-Wechsel (Benutzer- in Kernel-Status und zurück im Fall der Ein-/Ausgabe) notwendig ist. Die Locking-Architektur der CF bietet optimale Voraussetzungen für hohe Skalierbarkeit und Performance.

Den Hauptanwendungsbereich von Rechner-Großinstallationen bilden momentan Transaktionsverarbeitungs-Systeme. Datenbank-Systeme wie ORACLE, DB/2 oder ADABAS haben die Eigenschaft, direkt auf die Plattenspeicher zuzugreifen. Es sind insbesondere aus Sicherheits- und Performance-Gründen einerseits spezielle Transaktions-Monitore und andererseits effiziente Kommunikations-Mechanismen erforderlich.

Als Beispiel für das Verständnis des Locking-Problems in der Transaktionsverarbeitung sollen 2 Prozesse dienen, die zwei identische Programm-Sequenzen ausführen wollen:

## Locking Problem:

Anfangswerte:  $d1 = 15$ ,  $d2 = 20$ :

### Transaktion 1

```
read d1
if d1 > 10

sub d1, 10

add d2, 10
```

### Transaktion 2

```
read d1
if d1 > 10

sub d1, 10

add d2, 10
```

Ergebnis:  $d1 = -5$ ,  $d2 = 40$

Die Transaktion 1 liest zunächst den Datensatz  $d1$  und stellt fest, dass  $d1 > 10$  erfüllt ist. Die Transaktion 2 verrichtet dieselbe Arbeit und kommt zum identischen Ergebnis bei diesem Schritt. Anschließend subtrahiert Transaktion 1 von  $d1$  die Ziffer 10, d.h. es ergibt sich als Resultat  $d1 = 5$ . Bevor Transaktion 1 zu  $d2$  die Ziffer 10 addiert, subtrahiert Transaktion 2 von  $d1$  die Ziffer 10. Die Variable  $d1$  ist aber von der Transaktion 1 auf das Ergebnis 5 verändert worden, durch Transaktion 2 wird von dem veränderten Wert  $d1 = 5$  die Ziffer 10 subtrahiert, d.h.  $d1$  hat jetzt den Wert -5. Ähnlich ergibt sich für die Variable  $d2$  das Ergebnis  $d2 = 40$ . Beide Ergebnisse für  $d1$  und  $d2$  sind falsch.

Um das richtige Ergebnis zu erhalten, müssen Locks verwendet werden, die verhindern, dass beide Prozesse gleichzeitig auf die Variablen  $d1$  bzw.  $d2$  zugreifen. Transaktion 1 bewirbt sich um ein ReadLock für den Datensatz  $d1$ . Wird dieses ihr von der CF zugewiesen, dann liest Transaktion 1 zunächst  $d1$  und führt anschließend den Test (if  $d1 > 10$ ) durch. Im Anschluss daran bittet Transaktion 1 die CF um Zuteilung eines "Exclusive WriteLock", d.h. sie möchte  $d1$  verändern. Dasgleiche passiert bezüglich  $d2$ . Nachdem die Locks verwendet wurden, werden sie wieder freigegeben und können von der CF weiter vergeben werden. Die beiden Transaktionen laufen unabhängig voneinander ab, und sie liefern das richtige Ergebnis  $d1 = 5$ ,  $d2 = 30$ .

## 5.4 Benutzung von Locks (Sperren)

```
GetReadLock (d1)
read d1
if d1 > 10
```

```
GetWriteLock (d1)
sub d1, 10
```

```
GetWriteLock (d2)
add d2, 10
```

```
GetReadLock (d1)
read d1
if d1 > 10
```

```
GetWriteLock (d1)
sub d1, 10
```

```
GetWriteLock (d2)
add d2, 10
```

Ergebnis:  $d1 = +5$ ,  $d2 = 30$

Grundsätzlich stellt in Transaktions- und Datenbanksystemen das Lock ein Objekt dar, das mindestens über 4 Methoden verfügt:

- GetReadLock
- GetWriteLock
- PromoteReadtoWrite
- Unlock

In einer Two-Phase-Transaktion finden alle Lock-Aktionen zeitlich vor allen Unlock-Aktionen statt. Eine Two-Phase-Transaktion verfügt über eine Wachstums-Phase (Growing) und eine Schrumpf-Phase (Shrink). Während der Wachstums-Phase werden die Locks angefordert und in der Schrumpf-Phase wieder freigegeben.

Aus dem dargestellten Lock-Beispiel folgt sofort das Problem der Lock-Verwaltung. Eine primitive Lösung der Lock-Verwaltung zeigt die Abbildung 6. Es sind 2 Systeme dargestellt und ein Plattenspeicher, auf dem die Lock-Table implementiert ist.

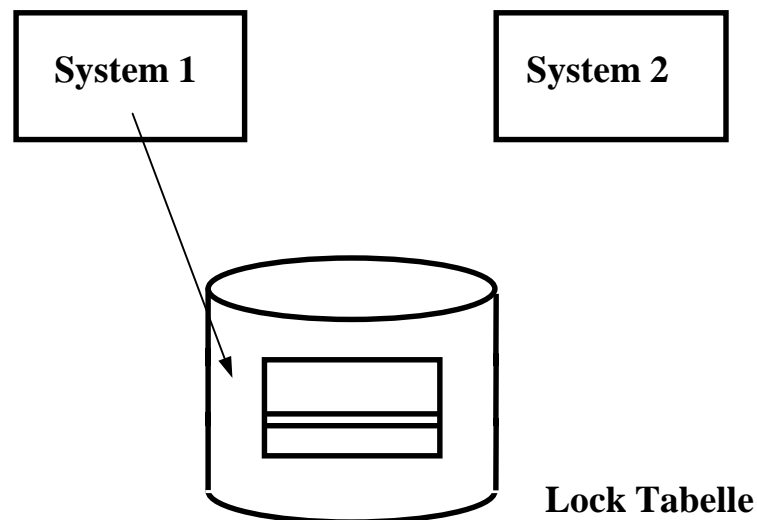


Abbildung 6: Primitive Lock-Verwaltung

Die Tabelle befindet sich in einer relationalen Datenbank derart, dass hinter jedem Datensatz noch ein Feld angeordnet ist. Beim Erwerb bzw. Freigeben des Locks wird dieses Feld geeignet modifiziert. Wenn ein Prozess z.B. auf einen spezifischen Record zugreifen möchte, muss er zunächst den Zustand des angehängten Locks überprüfen, anschließend das Lock erwerben und setzen. Erst dann kann der Datenzugriff durchgeführt werden. Der Nachteil dieser einfachen Lock-Verwaltung besteht darin, dass insgesamt zu viele Zugriffe zum Plattenspeicher vorgenommen werden müssen, die relativ viel Zeit in Anspruch nehmen und sich deshalb negativ auf die Performance auswirken.

Die momentan benutzte, moderne Methode der Lock-Verwaltung ist das sogenannte Shared-Disc-Verfahren. Es implementiert die Lock-Table im Hauptspeicher der Coupling Facility und die States und Queues der Locks in den Hauptspeichern der implementierten Systeme.

Die Funktionen der Lock-Architektur gehören sowohl zur CF als auch zum Operationssystem. Der zentrale Lock-Manager verfügt über einfache Shared (SHR)- und Exclusive (EXC)-Locks. Zusätzlich wird eine Menge von Tables bereit gestellt. Jede Table hat einen Namen und enthält N lockfähige Einträge (0...N-1). Tabelle 1 zeigt die Semantik der Lock-Architektur.

<b>derzeitiger Status</b> <b>Anforderung</b>	<b>kein</b>	<b>Lesen shared</b>	<b>Schreiben exclusive</b>
<b>Lesen Share</b>	<b>bewilligt, share-mode</b>	<b>bewilligt, share-mode</b>	<b>abgelehnt, Mitteilung über Besitzer</b>
<b>Schreiben Exclusive</b>	<b>bewilligt, exclusive mode</b>	<b>bewilligt, Warnung über Besitzer</b>	<b>abgelehnt, Mitteilung über Besitzer</b>

*Tabelle 1: Locking Semantics*

Ein Schlüsselaspekt dieses Modells besteht entgegen konventionellen Lösungen darin, dass die Requests zu der Locking Facility synchron mit Rücksicht auf die Verarbeitungseinheiten, die den Request ausführen, erfolgen. Das bedeutet, wenn eine Lock-Anforderung ausgeführt wird, wartet der anfordernde Prozessor solange, bis der Request komplett ist. Diese Lösung ist aufgrund der Performance-Charakteristik technisch möglich.

## 5.5 Contention-Erkennung

Der Besitz eines Locks kann mit ein oder mehreren Eigentums-Privilegien verbunden sein. Das Lock kann vergeben werden, wenn die Privilegien des Lock-Besitzers mit denen des Requestors kompatibel sind. Das bedeutet z.B. bei der Anforderung eines Locks mit Share- oder Exclusive-Privileg, dass die Contention erkannt wird, wenn ein Share-Privileg angefordert wird und der Lock-Besitzer das Exklusiv-Privileg hat oder ein Exklusiv-Privileg wird angefordert, und es existiert ein Besitzer eines Locks mit diesem Privileg.

Der System-Lock-Manager (SLM) kommuniziert mit den Tailored-Lock-Managers (TLMs) über einen Mechanismus, der in OS/390 Exits genannt wird. Der TLM implementiert eine Funktionalität, die spezifisch an irgendeine Umgebung wie eine Datenbank oder ein Shared-File-System angepasst ist. Ein TLM muss in der Lage sein, seine Performance auf der Grundlage seiner eindeutigen Umgebung zu optimieren. Die TLMs benutzen zwei dieser Exits (Contention, Notify), um die Contention für Shared-Ressourcen aufzulösen. In diesen Exits können die 64 Bytes der Nutzer-Daten geprüft und modifiziert werden. Weiterhin ist die Implementierung komplexer Lock-Protokolle möglich.

Wenn der Lock-Manager ein Lock wegen einer Contention nicht vergeben kann, verwaltet der SLM einen spezifischen Anforderungs-Record auf einer Liste der Waiters. Die Regeln bei der Verarbeitung der Waiter-Queues unterscheiden sich bei den DBM (DataBase Manager)s und TLMs. Das Queueing erfolgt innerhalb des SLMs, die Regeln für die Lock-Kompatibilität werden mit Hilfe der sogenannten Exits durch den TLM festgelegt.

## 5.6 Verfügbarkeit und Recovery

Um den heutigen Anforderungen großer kommerzieller Transaktions-Systeme gerecht zu werden, ist es wichtig, die Transaktions-Verarbeitung während der Behandlung von Hard- und Software-Fehlern bei voller Integrität der Datenbank zu garantieren. Da die CF physikalisch und logisch von den Systemen getrennt ist, kann sie die entsprechende Recovery-Funktionalität bei System- oder Software-Fehlern Performance-optimal zur Verfügung stellen. Dabei liefert die CF-Architektur beides: Locking- und Recovery-Recording-Funktion. Die modifizierten Lock-Namen (Exklusive Locks, benutzt für das Update der Datenbank-Records) werden in den Listen-Elementen der CF-Recovery-Tables aufgenommen. Jeder Instanz des System-Lock-Managers (SLM) wird eine Liste zugewiesen, die ein Teil des global verwalteten Locking-Protokolls darstellt. Die Coupling-Facility benutzt die User-Identifikation (UID), die der SLM für den Zugriff auf die geeignete Liste spezifiziert. Die CF-Architektur liefert auch die atomaren Operationen, mit deren Hilfe Lock-Table-Einträge und Record-Daten-Elemente manipuliert werden. Diese Operationen unterstützen Operationen zum Generieren, Lesen, Ersetzen und Löschen von Elementen. Durch die Bereitstellung von atomaren Operationen stellt die Architektur sicher, dass die Lock- und Recovery-Struktur immer konsistent sind. Beim Auftreten eines Fehlers in der CF gehen keine Daten verloren, weil alle relevanten Informationen in die CF innerhalb der Menge der TLM/SLMs kopiert sind. Da die Architektur mehrere CFs unterstützt, kann eine neue Struktur einer anderen CF zugeordnet werden, wobei die TLM/SLMs den CF-Inhalt regenerieren.

## 5.7 Lock-Modell-Konzeption

Die Modell-Konzeption umfasst prinzipiell zwei unterschiedliche Lock-Manager: Der verteilte und der zentrale Lock-Manager. Die Menge der Operationssystem-Dienste, die dieses Modell unterstützt, bildet den System-Lock-Manager (SLM). Das Operationssystem betrachtet die Locking-Architektur als ein "High-Speed-Lock-Contention-Detektor".

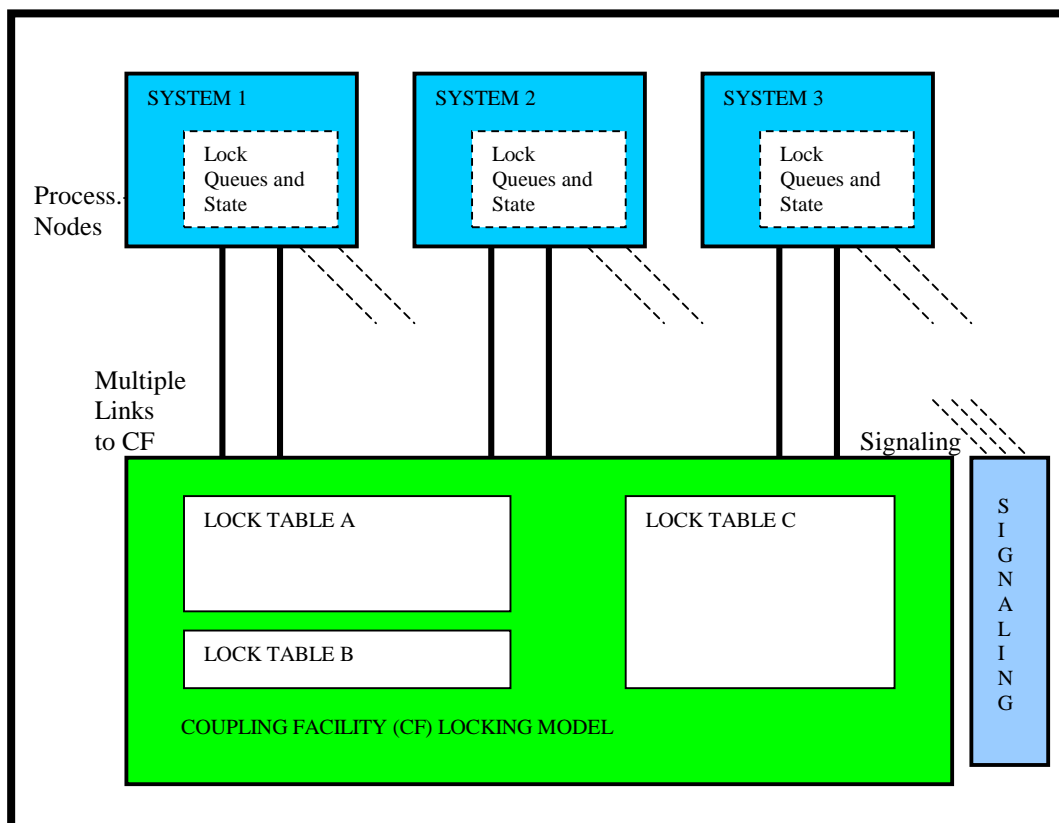


Abbildung 7: Operations-System-Modell

Um den Lock-Mechanismus zu beschreiben, werden verschiedene Beispiele von Lock-Requests benutzt. Die Abbildung 7 zeigt das allgemeine Locking-Modell. Die einzelnen Systeme (Knoten) sind über mehrere Links mit der CF verbunden. Jedes System besitzt seine Lock-Queues, in denen ein Signal-Protokoll zur Lösung der Lock-Konflikte benutzt wird. Die CF enthält den Zustand jedes Eintrags in der Lock-Table, während jedes System noch zusätzliche Informationen besitzt.

Die Lock-Table (Abbildung 8) enthält N-Einträge, d.h. jede Zeile entspricht einem Lock. Der Eintrag beschreibt den Zustand des Locks, der 3 verschiedene Werte annehmen kann: Exclusive (EXC, E), Shared (SHR, S, =1) und Frei (0). Jeder Eintrag setzt sich aus zwei Feldern zusammen: Das erste Feld bildet die System-Kennung für exklusiv oder global verwaltete Locks, während ein Bit-Vektor das zweite Feld einnimmt. Jedes Bit dieses Vektors repräsentiert ein mögliches System, das Interesse an dem zugehörigen Eintrag haben könnte. Die Multisystem-Locking-Struktur umfasst bis zu 32 Knoten, die mit ein oder mehreren CFs verbunden sein können. Jedes System kann mehrere Links zu der CF aus Verfügbarkeits- und Performance-Gründen implementieren, wobei jede CF ein oder mehrere Locking Tables verwaltet.

Die Locking-Struktur wird durch das Betriebssystem unterstützt, d.h. es liefert die Fähigkeit, jeder Lock-Anforderung einen eindeutigen Namen zuzuordnen. Dieser Name besteht aus einer Zeichenkette, die sich auf den Lock-Namen bezieht und einem Integer-Wert für die Hash-Klasse. Da sich die Lock-Namen in der Regel auf Datensätze beziehen, können die Namen relativ lang sein (z.B. in IMS-Datenbanken bis zu 19 Bytes = 152 Bits). Um keine Lock-Table im Hauptspeicher mit  $2^{152}$  Einträgen aufzubauen, werden die Locks in Hash-Klassen zusammengefasst. Bei 262.144 Locks kann man also den Adressraum für die Lock-Einträge durch die Einführung von Hash (Lock)-Klassen von 152 Bit auf 18 Bit ( $2^{18} = 262144$  Einträge) reduzieren. Dabei muss sichergestellt werden, dass jedes System einen identischen Hashing-Algorithmus benutzt.

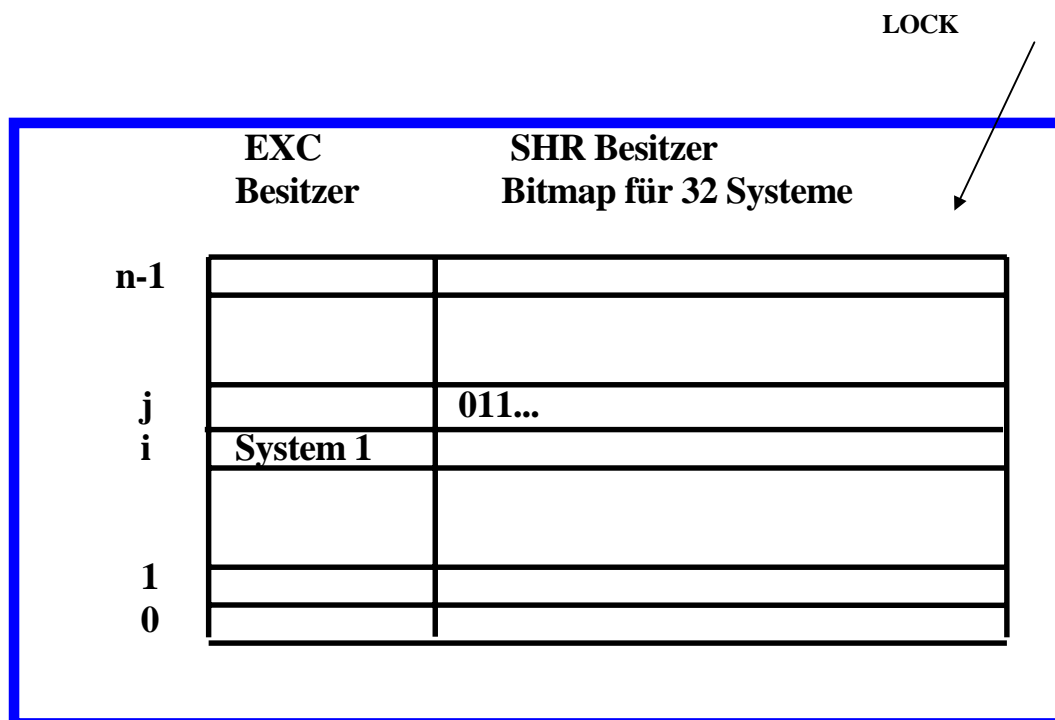


Abbildung 8: Struktur der Lock-Table

In der Abbildung 9 sind die State- und Queue-Informationen dargestellt, die jeweils im Hauptspeicher der einzelnen Systeme abgespeichert sind. Der Speicherbereich jedes Systems wird grundsätzlich in 4 Bereiche aufgeteilt:

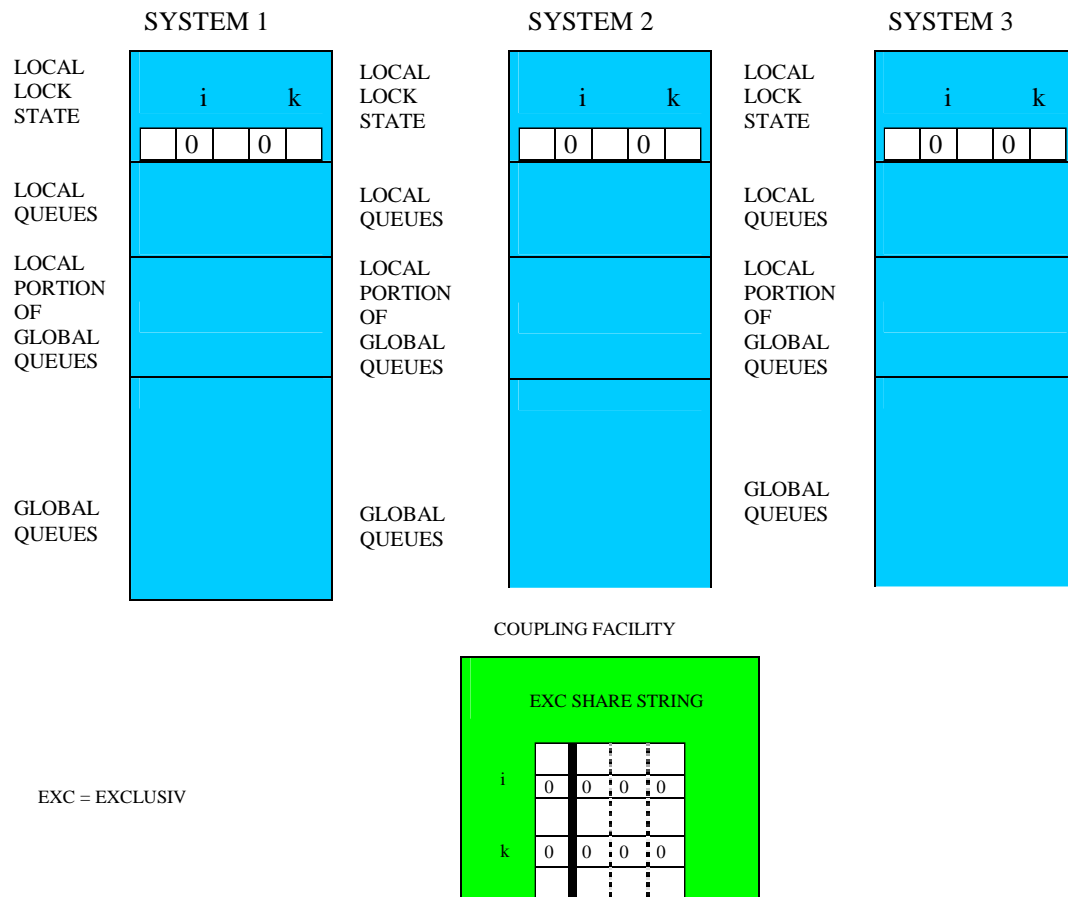


Abbildung 9

Der erste Bereich "Local Lock State" bezeichnet den Speicherplatz, auf dem die Kenntnis vom Zustand jedes Lock-Eintrags in einem speziellen System abgelegt wird. Diese Kenntnis ist entweder "0" (keine Kenntnis), "S" (Shared-Kennntnis), "E" (exklusive Kenntnis) oder "Gx" (ein anderes System besitzt exklusive Kenntnis).

Der zweite Bereich (unter dem ersten in Abbildung 9) in den Zustands- und Queue-Informationen jedes Systems, mit Local Queues bezeichnet, zeigt die Software-Queues, die entstehen, wenn ein Lock ohne Beteiligung anderer Systeme vergeben wird. Er enthält den Lock-Namen und den Hash-Klassen-Wert (z.B. "A,i"), den augenblicklich anfordernden Prozess, Zustand, Eigentümer. Z.B. "P1 (Own, EXC)" heißt, dass der Prozess P1 das Lock A in exklusivem Modus besitzt.

Die Bereiche drei und vier enthalten denselben Informations-Typ wie die Local Queues aber aus unterschiedlicher Perspektive. Die Requests kommen in den dritten Bereich (Local Portion of Global Queues), wenn ein anderes System zum globalen Manager wird, d.h. ein einzelnes System liefert das gesamte Management der Hash-Klassen. Das bedeutet, dieser Bereich zeigt die globale Queue aus der Sicht dieses Systems. Die Requests gelangen in den vierten Bereich (Global Queues), wenn dieses System zum globalen Manager der Queues wird. Demzufolge repräsentieren die Locks in diesem Bereich den kompletten, globalen Zustand.

Die folgenden Beispiele enthalten jeweils drei mit der CF verbundene Systeme. Die State- und Queue-Informationen beziehen sich auf die Einträge i und k in der Lock-Table der CF. Die Lock-Namen werden in die Hash-Klassen abgebildet und alle Lock-Manager, die eine gemeinsame Lock-Table benutzen, müssen auch einen gemeinsamen Hashing-Algorithmus verwenden. In der Lock-Table bedeutet der Eintrag in dem Exclusive-Feld (Spalte "EXC"), das ein bestimmtes System das Lock exklusiv besitzt. Der "Share String" ist eine Bit-Maske, die entsprechend der Position ein System betrifft, das Shared-Interesse an dem Lock-Eintrag besitzt, d.h. wenn die Position gleich "1" gesetzt ist. Ein Beispiel wäre der String "011", der bedeutet, dass die Systeme 2 und 3 Interesse (Shared) an dem Lock-Eintrag anmelden. Aus den folgenden Abbildungen geht hervor, dass jedes System über 4 Zielbereiche von Zustands- und Queue-Information verfügt

Das erste Beispiel in der Abbildung 10 (Figure 6, S. 209) enthält einen einfachen Request vom Prozess P1 für das Lock A in der Hash-Klasse i. Beim Empfang dieses Requests wird der lokale Lock-Zustand für die Hash-Klasse i geprüft und da im System 1 keine Kenntnis vom Zustand vorliegt, löst es einen Lock-Request für die Hash-Klasse i im exklusiven Zustand aus (bezeichnet durch 1 in der Abbildung 10). Die CF empfängt diesen Request und da es keine höheren Requests für diese Hash-Klasse gibt, setzt sie den exklusiven Eintrag von System in der Spalte EXC ein und sendet eine positive Antwort zurück. Die Anforderung wird dem System 1 gewährt, und es legt die geeignete Information in der lokalen Queue ab. System 1 stellt auch ein Zeichen in das Feld "Local Lock State" in der Position i und nimmt den vollen Lock-Namen in den Bereich "Local Queues" auf. Der Prozess P1 ist nun in der Lage, ein globales Lock ohne Wechselwirkung mit anderen Systemen zu erhalten.

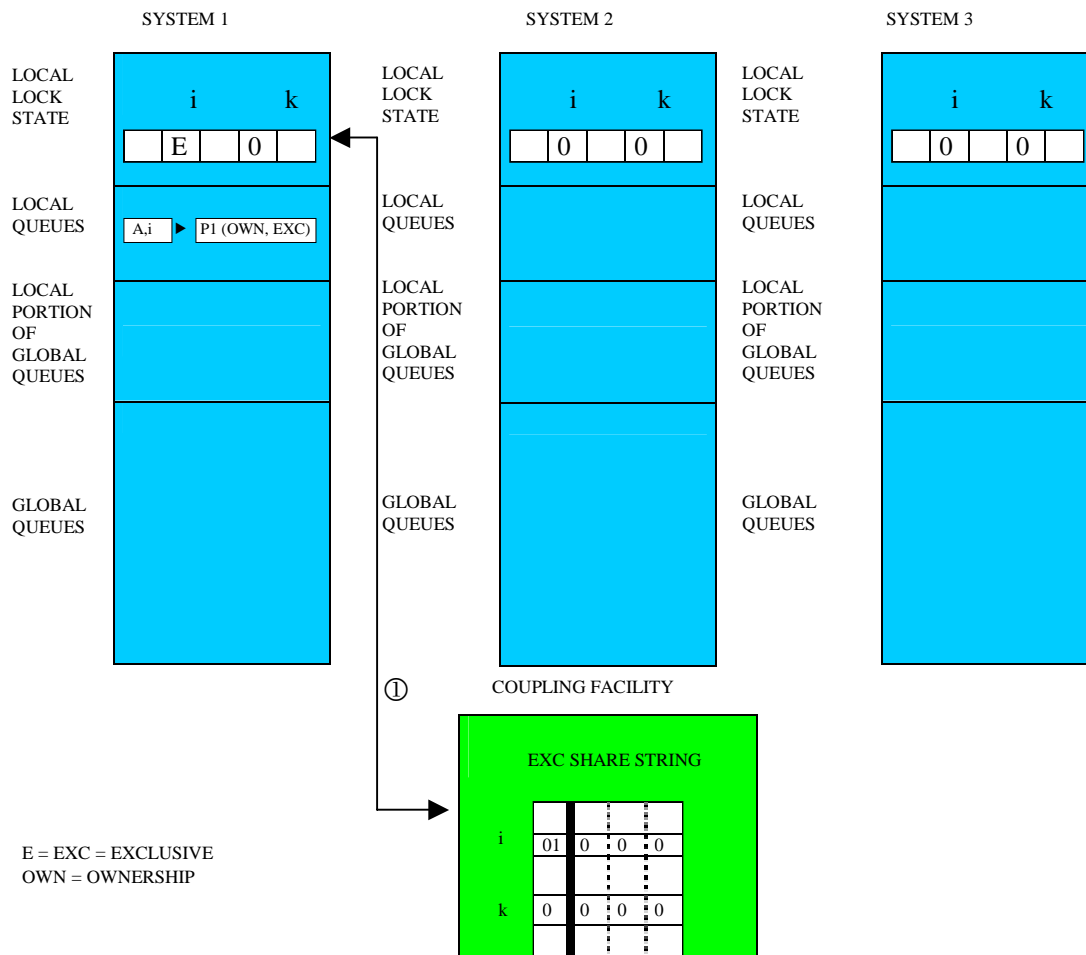


Abbildung 10

In der Abbildung 11 (Figure 7, Seite 210) ist der Zustand dargestellt, nachdem der Prozess P2 auf dem System 2 ein Lock mit dem Namen C in der Hash-Klasse k erhalten hat. Dieses resultiert auch aus einem Request an die CF, wo der Eintrag S für das System 2 gesetzt wird. Der Request wird gewährt und dann in den Bereich "Local Queues" aufgenommen.

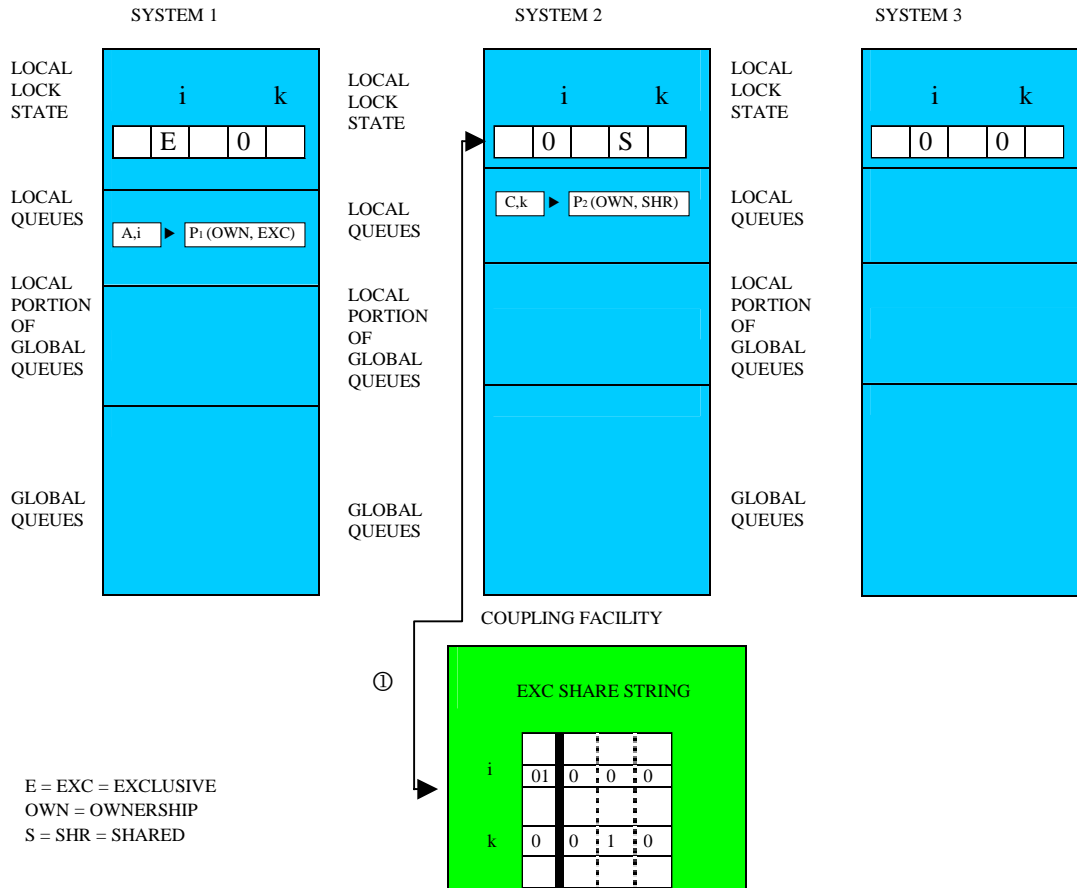


Abbildung 11

Ein weiteres Beispiel zeigt die Abbildung 12 (Figure 8, S. 211). Hier handelt es sich um den Fall, bei dem ein Prozess einen Request innerhalb derselben Hash-Klasse auslöst. In diesem Beispiel wird das globale Lock ohne Kommunikation mit der CF erhalten. Der Prozess P3 auf dem System 1 löst einen Request für ein Lock mit dem Namen B in der Hash-Klasse i aus. Während der lokale Lock-Zustand geprüft wird (Bereich "Local Lock State"), stellt das System fest, dass es schon exklusives Interesse in der Hash-Klasse hat, so dass kein CF-Zugriff notwendig ist. Es kann einfach die "Local Queues" prüfen und festlegen, dass der Request für ein anderes Lock als für B vergeben ist. Der Request kann anschließend gewährt werden.

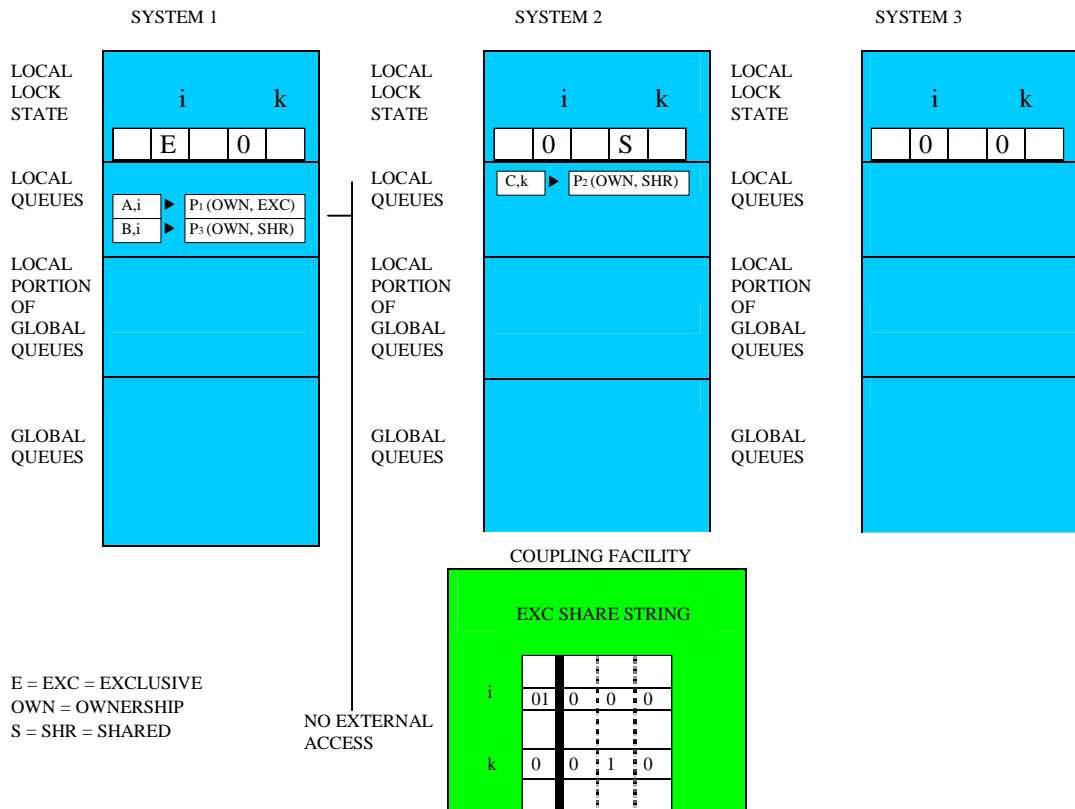


Abbildung 12

Die Abbildung 13 (Figure 9, S. 212) demonstriert, wie Requests in einen anderen "reinbomben" bei demselben Hash-Klassen-Niveau multibler Systeme, die aber noch kompatibel sind. Der Hauptgesichtspunkt in dem Beispiel besteht darin, dass das globale Lock ohne Austausch der Lock-Namen erhalten wird, gerade wenn sich eine Kollision in der CF ereignet. Abbildung 13 enthält ein Beispiel, bei dem ein Prozess P4 auf dem System 3 ein Request für ein Lock D in der Hash-Klasse k auslöst. Der "Local Lock State" zeigt, dass das System kein Exclusive-Interesse in der Hash-Klasse hat, so dass ein Request an die CF erfolgt. Dieses Mal sieht die CF, dass System 2 auch ein Interesse (Shared) an dem Request hat, aber da der augenblickliche Request kompatibel (Shared) ist, setzt es den Share-Eintrag für das System 3 und gibt eine positive Antwort zurück. Das System 3 wird nicht den anderen Systemen (Shared) bekannt gemacht, und der Request kann gewährt werden. Dieses Beispiel zeigt den Fall eines Hashing-Schemas, in dem multiple Lock-Namen in die derselbe Hash-Klasse abgebildet werden.

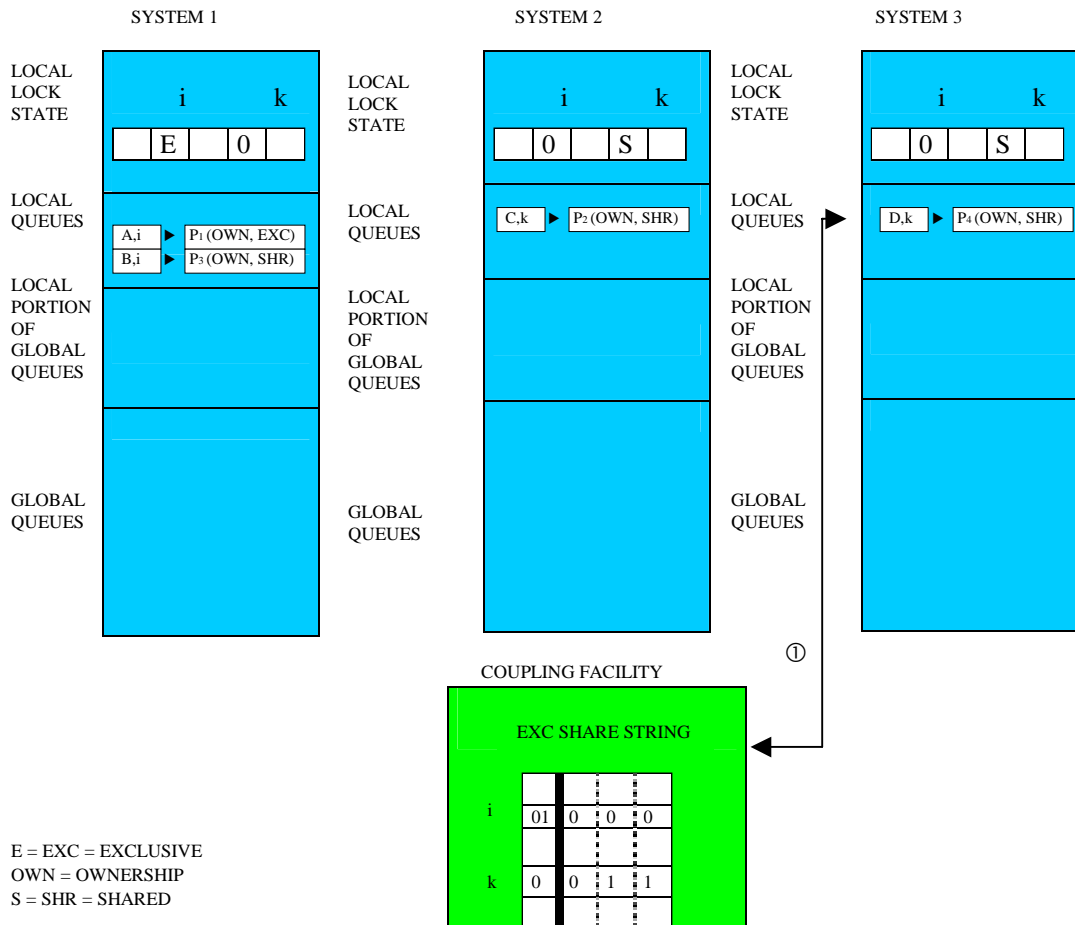


Abbildung 13

Eine Variante dieses Beispiels ist in der Abbildung 14 (Figure 10, S. 213) dargestellt. In diesem Fall werden zwei unterschiedliche Lock-Namen in dieselbe Hash-Klasse abgebildet, aber diese sind inkompatibel auf dem Hash-Klassen-Niveau. Diese Abbildung zeigt ein Beispiel, in dem der Prozess P5 auf dem System 1 einen Request für ein Lock E in der Hash-Klasse k auslöst. Aus dem "Local Lock State" ist erkennbar, dass das System kein Interesse in der Hash-Klasse hat, so dass ein Request zur CF erfolgt. Dieses Mal realisiert die CF, dass verschiedene Systeme ein Shared-Interesse in der Hash-Klasse haben. Die CF setzt den Exclusive-Eintrag und gibt den Shared-String zum System 1 zurück. System 1 hat nun die Verantwortung, den globalen Zustand dieser Hash-Klasse auszusortieren. Es startet dann einen Prozess, genannt Escalation, in dem eine globale Queue für die Hash-Klasse aufgebaut werden muss. Dieser analysiert den Shared-String zuerst syntaktisch, um zu bestimmen, dass die Systeme 2 und 3 Interesse an der Hash-Klasse haben, und signalisiert diesen Systemen, ihre Lock-Informationen zurückzugeben. Es gibt hier zwei wichtige Gesichtspunkte: Zuerst laufen diese Signale parallel über eine Hochgeschwindigkeits-Signalisierungs-Facility. Zweitens werden nur die Systeme benachrichtigt, die momentan Interesse in der Hash-Klasse haben. Dieses ist ein wichtiger Aspekt der Skalierbarkeit des Designs: Wenn es 32 Systeme in dieser Beispiel-Konfiguration gibt,

würden nur 2 Systeme für die Lock-Information angesprochen. Da System 1 die lokale Information von den anderen Systemen empfängt, baut es ein lokales Bild auf und realisiert, dass es keine Contention für irgendeinen Lock-Namen, insbesondere für eine Hash-Klasse, gibt. Diese Situation wird "Falsche Contention" genannt und dem Prozess P5 kann das Lock gewährt werden. Das Beispiel zeigt auch die Bewegung der Queue-Information auf den Systemen 2 und 3 zum lokalen Teil der "Global Queues". Es demonstriert, dass der Prozess nun die Verantwortung hat, um mit dem globalen Manager auf zukünftigen Zustands-Übergängen zu kommunizieren (z.B. Unlocks). Weiterhin wird gezeigt, dass der "Local Lock State" von S nach G1 geändert wurde, d.h. System 1 ist der globale Manager für diese Hash-Klasse.

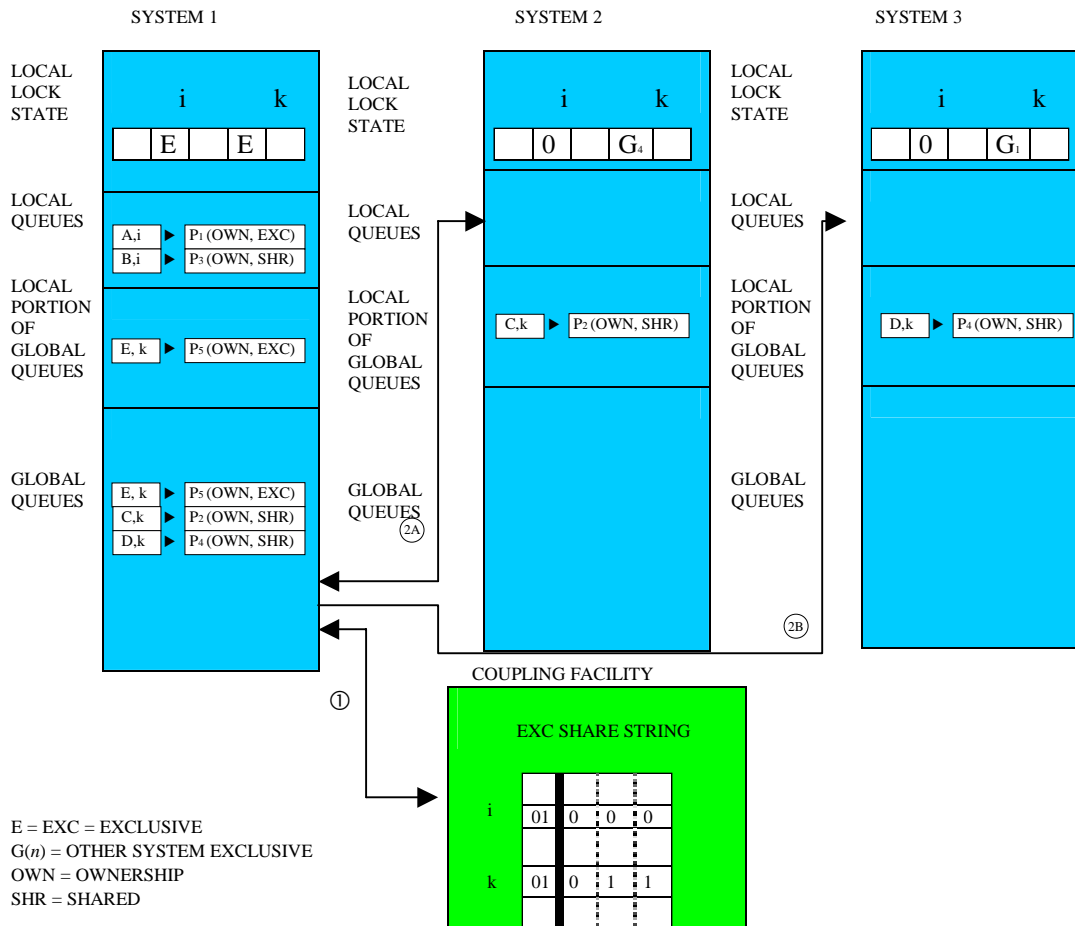


Abbildung 14

Das letzte Beispiel in der Abbildung 15 (Figure 11, Seite 214) stellt eine echte Contention dar. Die Abbildung enthält ein Beispiel des Systems 2, das ein Request für den Prozess P6 bezüglich eines Locks A in der Hash-Klasse i auslöst. Der "Local Lock State" zeigt, dass das System kein Interesse in der Hash-Klasse hat und deshalb ein Request zu der CF vorgenommen wird. Dieses Mal realisiert die CF, dass System 1 exklusives Interesse in dieser Hash-Klasse hat. Die CF signalisiert zurück, dass System 1 der exklusive Besitzer dieser Hash-Klasse ist. Wenn System 1 die Nachricht von System 2 empfängt, baut es eine Menge von globalen Queues auf. Da System 1 der exklusive Besitzer war, muss es nicht die anderen Systeme benachrichtigen. Sobald die Queues aufgebaut sind, legt es fest, dass es sich um eine reale Contention handelt.

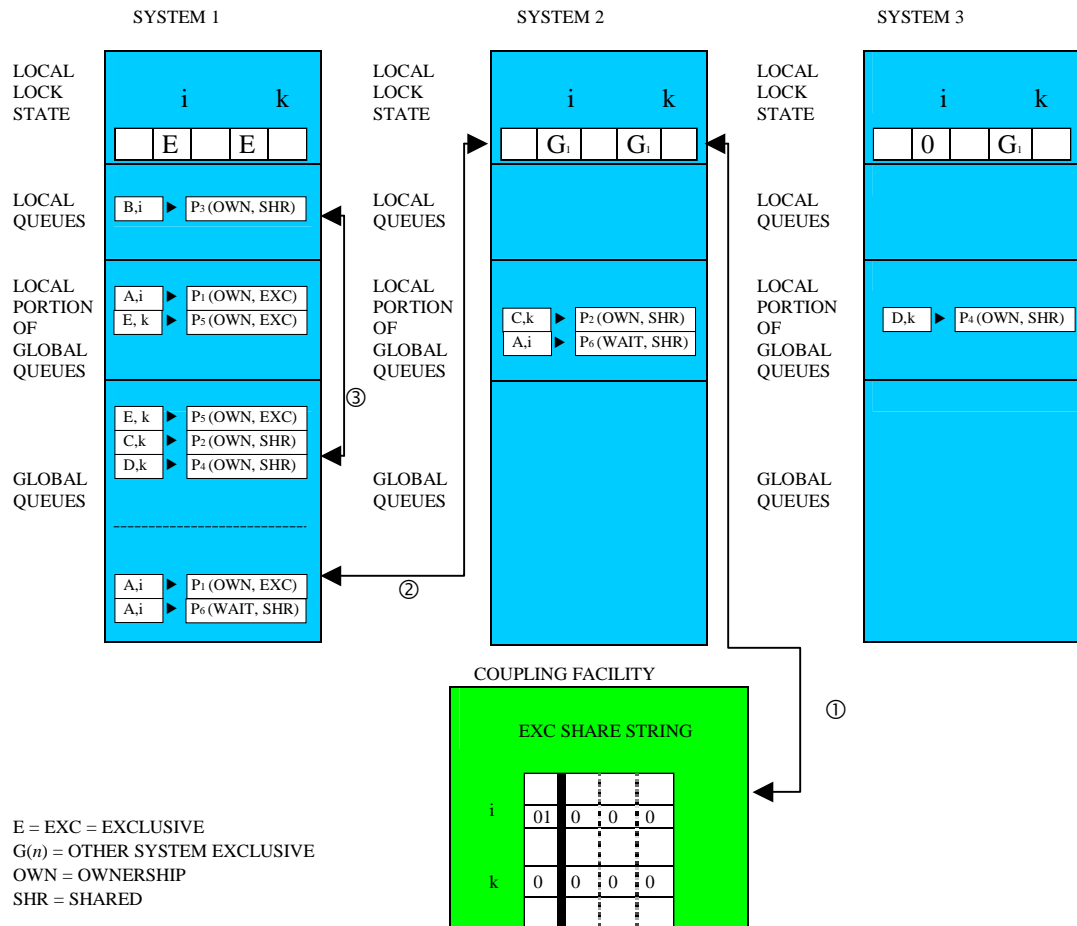


Abbildung 15

## 5.8 Lock-Contention

Die Auflösung einer Contention stellt einen komplexen Prozess dar. Um zunächst die Existenz einer Contention festzustellen, verwenden die CF und der SLM das Lock-Kompatibilitäts-Schema (s. Tabelle 2). In dem Fall, dass der angeforderte Zustand mit dem momentan existierenden Zustand kompatibel ist, wird dem Request entsprochen und das Lock durch den Halter lokal verwaltet. Wenn der angeforderte Zustand mit dem existierenden Zustand nicht kompatibel ist, dann wird das Lock durch eine gewählte TLM/SLM-Kombination global verwaltet und dieser Request mit zukünftigen Requests vom globalen Manager (TLM/SLM) verarbeitet. Der gewählte SLM verwaltet nicht die Contention sondern unterhält eine Queue der Besitzer und Requestoren für die TLM-Nutzung zur Contention-Verwaltung. Ist die Contention erkannt, so übergibt der gewählte SLM die Request-Queue dem TLM, wobei das Contention-Exit benutzt wird. Die Nutzer-Daten-Information spielt eine integrale Rolle in den Nutzer-definierten Lock-Protokollen. Diesbezüglich muss der TLM die Contention durch Verwendung einer der folgenden Aktionen verwalten:

1. Gewähren eines anstehenden Requests, möglicherweise mit einem unterschiedlichen Zustand als angefordert. Daraus ergibt sich, dass der Requestor seine Arbeit wieder aufnimmt oder seinen Exit fertig stellt.
2. Abweisen eines anstehenden Requests. Das führt auch dazu, dass der Requestor seine Arbeit wieder aufnimmt. In diesem Fall wird der Requestor von der Abweisung unterrichtet und ihm entsprechende Daten durch den ablehnenden TLM übergeben. Die Nutzer-Daten können modifiziert werden, wenn der Request abgewiesen wird. Die Nutzer-Daten dienen im Normalfall dazu, dem Requestor den Grund für die Abweisung zu übermitteln.
3. Wiedergewähren eines anstehenden Requests mit einem unterschiedlichen Zustand als ursprünglich gewährt wurde (d.h. Herunterstufen eines Exclusive-gehaltenen Locks zu Shared). In diesem Fall wird der Completion Exit des Halters dadurch eingeleitet, dass der Halter über die Zustandsänderung informiert wird. Zusätzlich können die Nutzer-Daten bei einem Wiedergewähren modifiziert werden.
4. Informieren eines aktuellen Besitzers einer Ressource darüber, dass eine Contention für die Ressource, die er besitzt, existiert. Dieses erfolgt durch Übergabe der Mitteilung durch den SLM an den haltenden TLM mittels seines Notify Exit's.
5. Entfernen eines anstehenden Requests. Der Request wird in der Request-Queue durch den SLM entfernt. Der TLM kann den Request aufgrund einer anschließenden Bitte des Contention Exit gewähren (d.h. wenn das Unlock des augenblicklichen Besitzers dem Contention Exit signalisiert wird).

Das Lock-Contention-Problem tritt dann auf, wenn ein Lock-Request für eine Ressource generiert wird und dieser zu dem augenblicklichen Lock-Zustand inkompatibel ist. Die Lock-Request-Anweisung bezieht sich auf das gegebene Locking-Niveau, wobei jedes Niveau eine unterschiedliche Contention-Sicht besitzt und darauf in verschiedener Art und Weise einwirken kann. Auf dem Architektur-Niveau wird eine Contention auf Hash-Klassen der Locks angezeigt. Die Contention wird dem System-Lock-Manager (SLM) über die Information, die in dem Antwort-Block abgespeichert ist, mitgeteilt. Der SLM versucht, die Contention durch Kommunikation mit seinen anderen Instanzen, den Share-Informationen in den Hash-Klassen und durch Vergleich des aktuellen Lock-Namens mit der Menge der Lock-Namen für die augenblicklichen Besitzer und Wartenden zu lösen. Es kann nun passieren, dass der angeforderte Lock-Name nicht mit irgendeinem dieser anderen Namen übereinstimmt. In diesem Fall existiert auf diesem Niveau keine Contention und das Lock kann ohne Bedenken vergeben werden. Diese Situation heißt "False Contention", d.h. die auf dem Architektur-Niveau angezeigte Contention existiert nicht auf dem SLM-Niveau, auf dem sie aufgelöst wird.

In dem Fall, dass der SLM ein oder mehrere Namen in der Liste der Hash-Klasse findet, die mit dem Namen des angeforderten Locks übereinstimmen, spricht man von einer "Real Contention". Der SLM meldet die reale Contention an den TLM. Letzterer bestimmt, nachdem er die Nachricht vom SLM erhalten hat, die Contention. Wenn der TLM zu der Entscheidung kommt, dass es sich um eine reale Contention handelt, dann wird der anfordernde Prozess in eine Warteschlange eingeordnet. Im anderen Fall kann der TLM dem Request entsprechen. Es kann vorkommen, dass der Lock-Name einen zu großen Granularitäts-Bereich in der Datenbank einnimmt und deshalb eine feinere Granularität der Locks notwendig ist. Das führt zu einem Herabstufen aller relevanten Locks in eine Granularität, in der die Contention nicht existieren kann. Diese hierarchische Locking-Methode wird benutzt, um zu verhindern, dass gesperrte Datenbank-Bereiche entstehen, auf die selten oder nur durch ein einzelnes System zugegriffen wird. Dieses Design kann die gesamte Locking-Rate reduzieren und stellt ein wichtiges Verfahren zur Erhöhung der Performance durch Anpassen der Locking-Rate an das Niveau der Contention im System dar.

Ein Beispiel für das explizite hierarchische Locking in DB2 enthält die Abbildung 16. Die CF-Lock-Struktur kommuniziert über den Cross System Extended Services (XES) jedes Systems mit diesen. Dabei bilden alle XES und die Lock-Struktur in der CF zusammen den globalen Lock-Manager. Ein globales Lock liefert Intra- und Inter-DB2-Concurrency-Control, dagegen implementiert ein lokales Lock nur Intra-DB2-Concurrency-Control. Wird ein globales Lock durch einen Requestor angefordert, überprüft der Local Lock-Manager (LLM), ob dieses Lock lokal, d.h. ohne Zugriff auf die CF, vergeben werden kann. Zur Abwicklung dieser Prozedur dient das Explicit Hierarchical Locking (EHL)-Verfahren. Das Locking in der CF erfolgt mit möglichst grober Granularität. Der LLM verwaltet Daten-Informationen mit feinerer Granularität. Vom LLM werden möglichst viele Lock-Requests ohne Zugriff auf die globale Struktur der CF behandelt. Angenommen, System 2 (s. Abbildung 16) fordert von der CF ein Lock an, das derzeit vom System 1 gehalten wird. In diesem Fall benachrichtigt die CF das System 2 über die Vergabe des globalen Locks. Das System 2 kann nun über die Channel-To-Channel (CTC)-Verbindung mit dem System 1 eine Herabstufung und feinere Granularität vereinbaren. Die CTC-Verbindung erfolgt physikalisch über den Escon-Director. Der Grund für die Lock-Herabstufung besteht in der möglichen Chance, dass auf einer niederen Hierarchie-Stufe kein Lock-Konflikt vorliegt. Die Cross System Coupling Facility (XCF) in jedem System stellt die OS/390 Coupling Services zur Verfügung. Mit Hilfe dieser Dienste ist es möglich, dass Programme unter den entsprechenden

OS/390-Images in einem Sysplex miteinander kommunizieren. Die Grundlage dafür bilden jeweils zwei Pfade PATHIN und PATHOUT zwischen einem Image-Paar im Sysplex.

Wenn die Locking-Granularität seine feinste Stufe erreicht hat, stimmen die reale und die aktuelle Contention überein. Das Workload zeigt, dass dieses Contention-Niveau in den meisten Transaktions-Systemen extrem begrenzt ist, und erfahrungsgemäß wird angenommen, dass es kleiner als 0,5 % aller Lock-Requests ausmacht.

Die "False Contention" ist eine Funktion der Lock-Table-Größe (d.h. der Anzahl der aktiven Locks im System) und des Hashing-Algorithmus. Die Größe der Lock-Table kann vom Systemprogrammierer dynamisch verändert werden, um die Menge der "False Contention" zu minimieren. Der Hashing-Algorithmus ist schwieriger zu manipulieren, und es wurde bisher viel Arbeit in die Entwicklung einheitlicher Hashing-Funktionen investiert. Eine Faustregel besagt, dass die gesamte Contention in dem System nicht größer als 1.0 % sein sollte. Bei einem guten Locking-Design kann die reale Contention bis zu einem extrem kleinen Niveau herabgestuft werden. Im Falle einer relativ einheitlichen Hashing-Funktion ist es möglich, eine falsche Contention durch Steuerung der Lock-Table-Größe zu beseitigen.

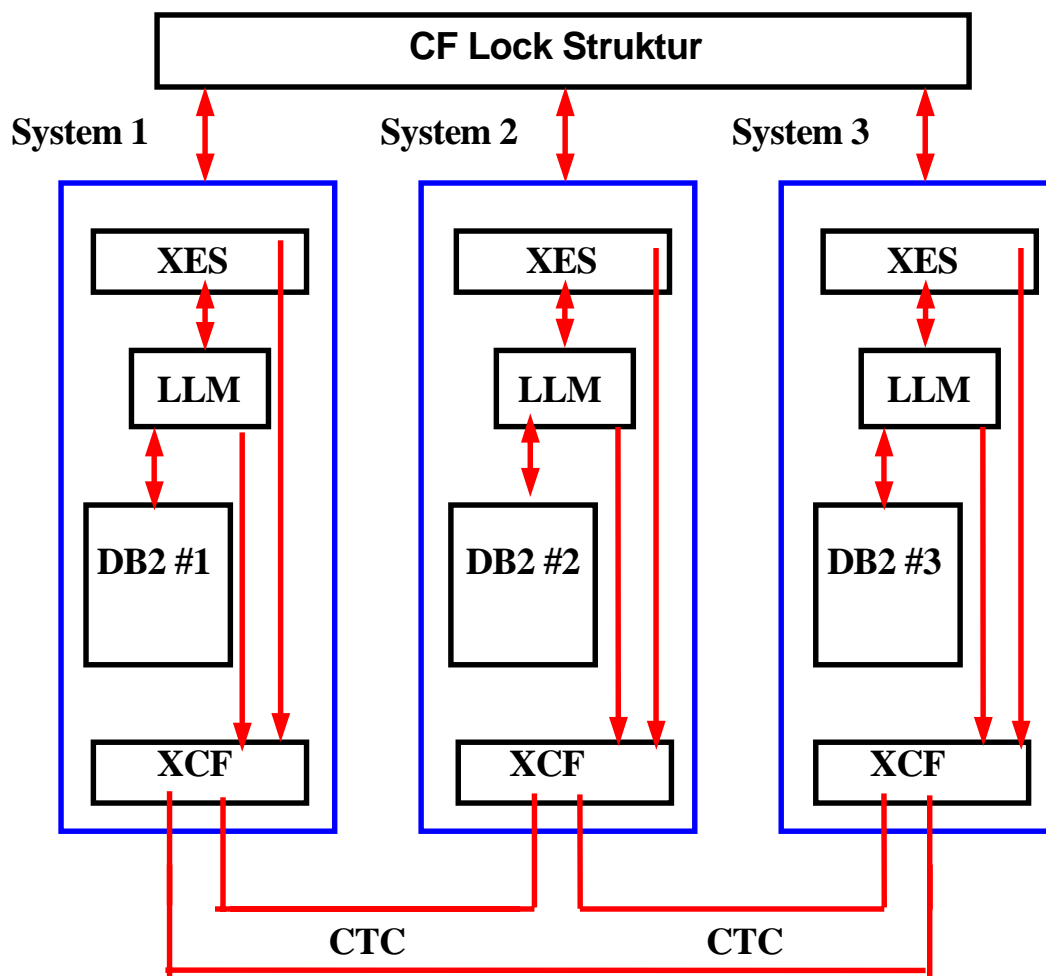


Abbildung 16: DB2 Globales Locking

WLM- und Sysplex-Unterstützung ist für folgende Subsysteme verfügbar:

- CICS
- IMS
- DB2
- USS
- VSAM
- Communication Server

Das bedeutet, dass diese Subsysteme die CF benutzen können. Das CF-Locking und -Caching integrieren den WLM. Dabei erfolgt das Locking auf der Record-Ebene. Der WLM ist Bestandteil von jedem OS/390-Image. Auf einem Sysplex ist eine Version des Transaktions-Monitors CICS (CICSplex) lauffähig. In der Abbildung 17 ist der Sysplex-Overhead dargestellt, wobei die obere Kurve dem linearen Wachstum und die untere dem realen Sysplex-Verhalten entspricht. Der Overhead ergibt sich selbst dann, wenn der Sysplex nur aus einem System besteht.

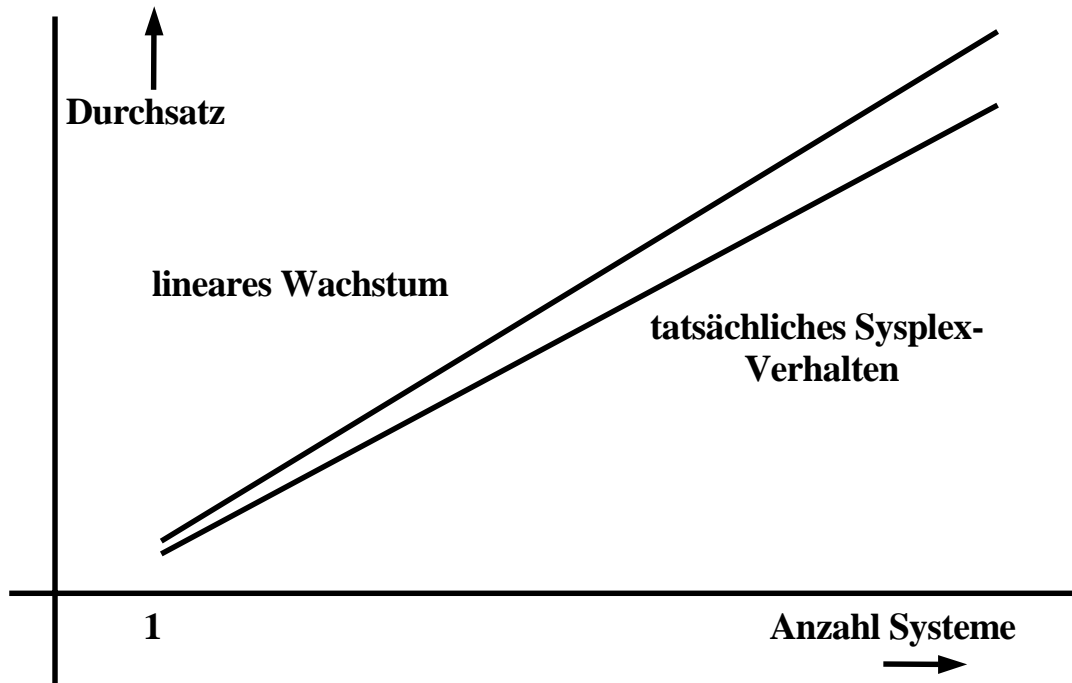


Abbildung 17: Sysplex-Overhead

Eine entsprechende Aufstellung des prozentualen Sysplex-Overheads als Funktion der System-Anzahl in verschiedenen Installationen zeigt die Tabelle 2.

<b>Installation</b>	<b>Anzahl System</b>	<b>% Sysplex-Overhead</b>
<b>A</b>	<b>4</b>	<b>11</b>
<b>B</b>	<b>3</b>	<b>10</b>
<b>C</b>	<b>8</b>	<b>9</b>
<b>D</b>	<b>2</b>	<b>7</b>
<b>E</b>	<b>11</b>	<b>10</b>
<b>Relational Warehouse Workload</b>	<b>2</b>	<b>13,30</b>