UNIVERSITÄT LEIPZIG

Enterprise Computing
Einführung in das Betriebssystem z/OS

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

WS2012/2013

Transaktionsverarbeitung Teil 4

Transaction Processing Monitor

Locking

Ein fundamentales Problem der Transaktionsverarbeitung besteht darin, dass ein Transaktionsmonitor aus Performance Gründen in der Lage sein muss, hunderte oder tausende von Transaktionen gleichzeitig auszuführen.

Um die Isolation (das `i` in ACID) zu gewährleisten muss der Anschein erweckt werden, dass alle Transaktionen der Reihe nach bearbeitet werden, also dass Transaktion Nr. 2 erst dann an die Reihe kommt, wenn die Verarbeitung von Transaktion Nr. 1 abgeschlossen ist. Wenn beide Transaktionen auf unterschiedliche Daten zugreifen, ist es kein Problem sie trotzdem parallel zu verarbeiten. Wenn beide Transaktionen aber auf die gleichen Daten zugreifen, muss sichergestellt werden, dass keine Verletzung der ACID Eigenschaften auftritt. Dies geschieht mit Hilfe von Locks (Sperren).

Locking Problem

Angenommen zwei Transaktionen, die auf die beiden zwei Variablen d1 und d2 zugreifen.

Anfangswerte: d1 = 15, d2 = 20 .

Transaktion 1	Transaktion 2
read d1	
if d1 > 10	
•	
•	read d1
sub d1, 10	if d1 > 10
•	•
add d2, 10	sub d1, 10
•	•
end if	add d2, 10
	•
	end if

Zeit ↓

Die beiden Abhängigkeiten:

Dirty Read
Eine Transaktion erhält veraltete Information

Lost update
Eine Transaktion überschreibt die Änderung einer anderen Transaktion

müssen gesteuert werden.

Das Ergebnis ist: d1 = - 5, d2 = 40, obwohl die if-Bedingung ein negatives Ergebnis verhindern sollte.

Benutzung eines Locks

Um eine Funktion `incr` zu implementieren, die einen Zähler x hochzählt, könnte Ihr erster Versuch sein:

```
void incr()
{
    x++;
}
```

Dies garantiert nicht die richtige Antwort, wenn die Funktion `incr` von mehreren Threads gleichzeitig aufgerufen wird. Das Statement x++ besteht aus drei Schritten: Abrufen des Werts x; Erhöhung um 1, Ergebnis in x speichern. In dem Fall, dass zwei Threads dies im Gleichschritt tun, lesen beide den gleichen Wert, erhöhen um 1, und speichern den um 1 erhöhten Wert. x wird nur um 1 an Stelle von 2 inkrementiert. Ein Aufruf von `incr()` verhält sich nicht atomar; es ist für den Benutzer sichtbar, dass er aus mehreren Schritten besteht. Wir könnten das Problem durch die Verwendung eines Mutex lösen, welcher nur von einem Thread zu einem Zeitpunkt gesperrt werden kann:

```
void incr()
{
    mtx.lock();
    x++;
    mtx.unlock();
}
```

In Java könnte das so aussehen

```
void incr()
{
    synchronized(mtx) {
        x++;
    }
}
```

Hans-J. Boehm, Sarita V. Adve: You Don't Know Jack About Shared Variables or Memory. Models. Communications of the ACM, February 2012, vol. 55, no. 2.

Concurrency Control

Concurrency Control (Synchronisierung) ist zwischen den beiden Transaktionen erforderlich, um dies zu verhindern. Der Begriff, der dies beschreibt ist *Serialisierung (serializability).*

Hierfür wird jedes Datenfeld, auf das zwei Transaktionen gleichzeitig zugreifen können, mit einem Lock (einer Sperre) versehen.

Eine Transaktion, welche auf dieses Datum zugreifen will, erwirbt zunächst das Lock. Dies geschieht dadurch, dass das Lock, welches normalerweise den Wert 0 hat, auf 1 gesetzt wird.

Die Transaktion verrichtet nun ihre Arbeit. Nach Abschluss wird das Lock wieder freigegeben (auf 0 gesetzt).

Eine andere Transaktion, die während dieser Zeit ebenfalls versucht, auf das gleiche Datenfeld zuzugreifen, wird daran gehindert, weil das Lock den Wert 1 hat. Sie muss warten, bis das Lock wieder den Wert 0 hat.

Bei großen Datenbeständen können viele Millionen von Locks für unterschiedliche Datenobjekte vorhanden sein

Fred Brooks erfand Locks in 1964 bei der Entwicklung von OS/360, damals als "exclusive control" bezeichnet.

Read und Write Locks

In der Praxis unterscheidet man zwischen Read und Write Locks. Read Locks verhindern „Dirty Reads“. Write Locks verhindern “Lost Updates”.

Mehrere Transaktionen können gleichzeitig ein Read Lock besitzen. Sie können sich gleichzeitig über den Inhalt eines Datenfeldes informieren. Solange der Inhalt des Datenfeldes nicht geändert wird, ist noch nichts passiert.

Wenn eine Transaktion den Inhalt des Datenfeldes ändern will, konvertiert sie das Read Lock in ein Write Lock. Dies bewirkt, dass eine Nachricht an alle anderen Transaktionen geschickt wird, die ein Read Lock für das gleiche Datenfeld besitzen. Letztere wissen nun, dass der Wert, den sie gelesen haben, nicht mehr gültig ist.

Hierzu ist erforderlich, dass nur eine Transaktion in jedem Augenblick ein Write Lock für ein bestimmtes Datenfeld besitzen darf. Write Locks werden deshalb auch als „Exclusive Locks“ bezeichnet.

Benutzung von Locks (Sperren)

Angenommen zwei Transaktionen, die auf die beiden zwei Variablen d1 und d2 zugreifen.

Anfangswerte: d1 = 15, d2 = 20 .

Transaktion 1	Transaktion 2
GetReadLock (d1)	
read d1	
if d1 > 10	
	GetReadLock (d1)
	read d1
	if d1 > 10
GetWriteLock (d1)	→ *Nachricht an Transaktion 2*
GetWriteLock (d2)	
sub d1, 10	
add d2, 10	
ReleaseLocks	GetReadLock (d1)
	read d1
	if d1 > 10
	GetWriteLock (d1)
	GetWriteLock (d2)
	sub d1, 10
	add d2, 10
end if	end if

Das Senden einer Nachricht informiert Transaktion 2, dass ihr Wissen über den Zustand der beiden Variablen d1 und d2 nicht mehr gültig ist.

Transaktion 2 muss sich neu über den Zustand der Variablen d1 informieren, ehe sie ein Update von d1 vornimmt.

Frage: Woher weiß Transaktion 1, dass Transaktion 2 ein Interesse an der Variablen d1 hat (ein shared Lock besitzt)? Erfordert zusätzlichen Aufwand.

Ergebnis: d1 = + 5, d2 = 30. Transaktion 2 ändert die beiden Variablen d1 und d2 nicht mehr.

Two-Phase Locking - Two-Phase Transaction

In Transaktionssystemen und Datenbanksystemen werden Locks (Sperren) benutzt, um Datenbereiche vor einem unautorisierten Zugriff zu schützen. Jedem zu schützenden Datenbereich ist ein Lock fest zugeordnet. Ein Lock ist ein Objekt welches über 4 Methoden und zwei Zustände S und E verfügt. Die Methode

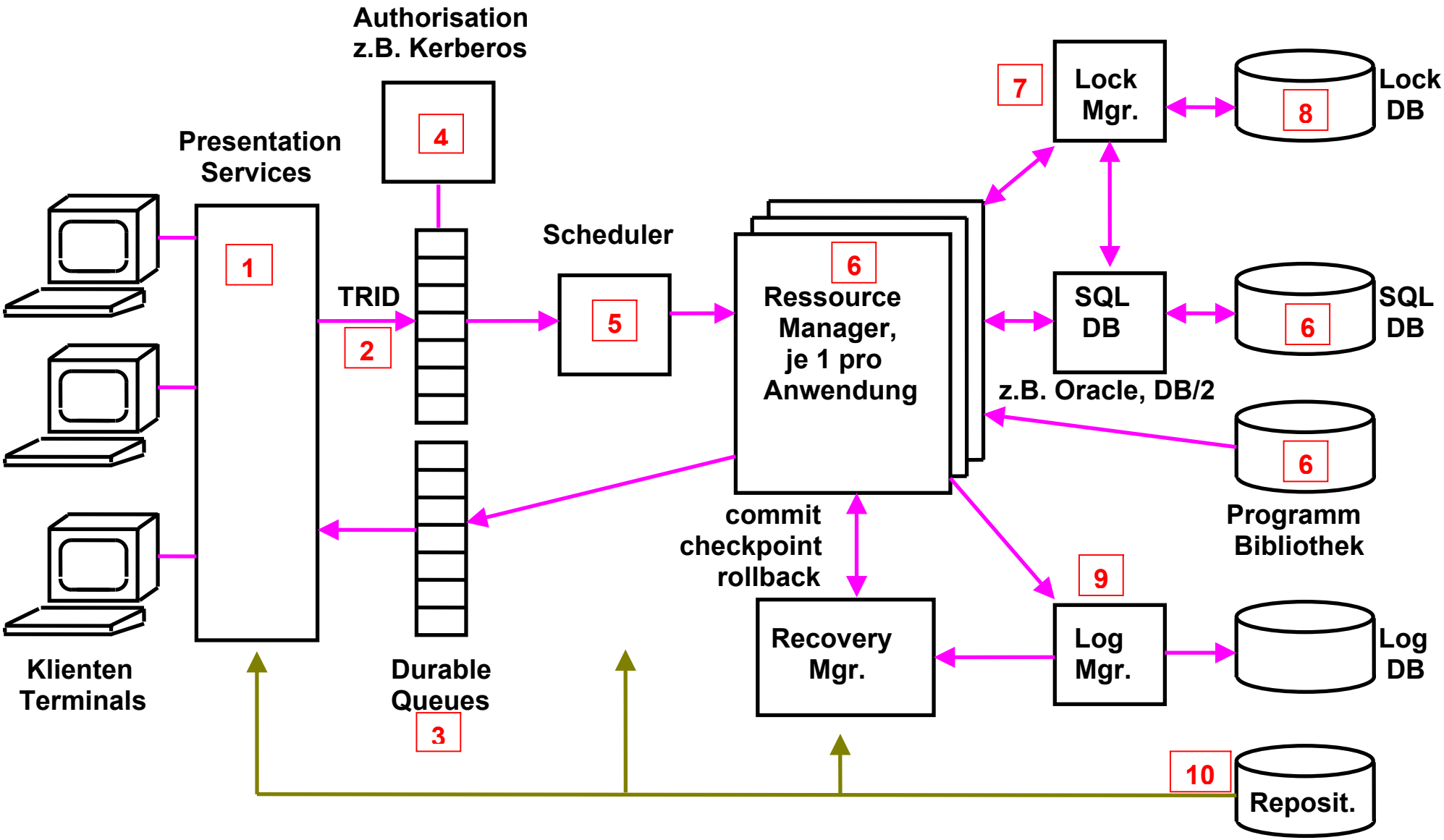
- **GetReadLock** **reserviert S Lock (shared),**
- **GetWriteLock** **reserviert E Lock (exclusive),**
- **PromoteReadtoWrite** **bewirkt Zustandswechsel S $\rightarrow$ E,**
- **Unlock** **gibt Lock frei.**

Mehrere Transaktionen können ein S Lock für den gleichen Datenbereich (z.B. einen Datensatz) besitzen. Nur eine Transaktion kann ein E Lock für einen gegebenen Datenbereich besitzen. Wenn eine Transaktion ein S Lock in ein E Lock umwandelt, müssen alle anderen Besitzer des gleichen S Locks benachrichtigt werden.

Normalerweise besitzt eine Transaktion mehrere Locks.

In einer Two-Phase Locking Transaktion finden alle Lock Aktionen zeitlich vor allen Unlock Aktionen statt. Eine Two-Phase Transaktion hat eine Wachstumsphase (growing), während der die Locks angefordert werden, und eine Schrumpf- (shrink) Phase, in der die Locks wieder freigegeben werden.

Two Phase Locking ist nicht zu verwechseln mit dem 2-Phase Commit Protokoll der Transaktions-verarbeitung



Struktur eines Transaction Processing Monitors (TP Monitor)

Das Zusammenspiel der einzelnen Komponenten wird auf den folgenden Seiten erläutert.

Komponenten eines TP Monitors (1)

Endbenutzer kommunizieren mit dem TP Monitor mit Hilfe von Nachrichten. Klienten (Arbeitsplatzrechner) werden häufig als „Terminals“ bezeichnet.

1. **Presentation Services empfangen Nachrichten und bilden die Datenausgabe auf dem Bildschirm des Benutzers ab.**

2. **Eingabe-Nachrichten werden mit einer TRID (Transaktions ID) versehen und in einer Warteschlange gepuffert.**

3. **Aus Zuverlässigkeitsgründen muß diese einen Systemabsturz überleben und ist persistent auf einem Plattenspeicher gesichert. Eine ähnliche Warteschlange existiert für die Ausgabe von Nachrichten.**

4. **Die Benutzer-Authorisation erfolgt über ein Sicherheitssystem, z.B. Kerberos.**

5. **Der Scheduler verteilt eingehende Bearbeitungsanforderungen auf die einzelnen Server Prozesse.**

Komponenten eines TP Monitors (2)

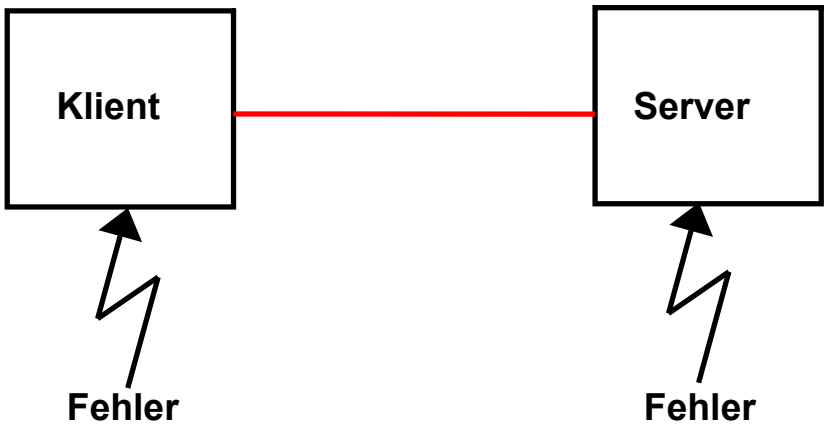
6. Ein TP Monitor bezeichnet seine Server Prozesse als Ressource Manager. Es existiert ein Ressource Manager pro (aktive) Anwendung. Ressource Manager sind multithreaded; ein spezifischer Ressource Manager für eine bestimmte Anwendung ist in der Lage, mehrere Transaktionen gleichzeitig zu verarbeiten.

7. Der Lock Manager blockiert einen Teil einer Datenbanktabelle. Alle in Benutzung befindlichen Locks werden in einer Lock Datei gehalten,

8. Aus Performance Gründen befindet sich diese in der Regel im Hauptspeicher. In Zusammenarbeit mit dem Datenbanksystem stellt der Lock Manager die „Isolation“ der ACID Eigenschaft sicher.

9. Der Log Manager hält alle Änderungen gegen die Datenbank fest. Mit Hilfe der Log Datenbank kann der Recovery Manager im Fehlerfall den ursprünglichen Zustand der Datenbank wiederherstellen (Atomizität der ACID Eigenschaft).

10. In dem Repository verwaltet der TP Monitor Benutzerdaten und -rechte, Screens, Datenbanktabellen, Anwendungen sowie zurückliegende Versionen von Anwendungen und Prozeduren.



Fehlerbehandlung

In einer Client/Server Konfiguration existieren drei grundsätzliche Fehlermöglichkeiten:

1. **Fehler in der Verbindung zwischen Client und Server**
2. **Server kann Prozedur nicht beenden (z.B. Rechnerausfall, SW-Fehler)**
3. **Client-Process wird abgebrochen ehe Antwort vom Server eintrifft, kann die Antwort des Servers nicht entgegennehmen.**

Fehler in der Verbindung zwischen Client und Server werden in der Regel von TCP in Schicht 4 automatisch behoben und sind deshalb unkritisch. Die beiden anderen Fehlermechanismen erfordern Fehlerbehandlungsmaßnahmen.

Fehlerbehandlung - Ausfall des Servers

1. Idempotente Arbeitsvorgänge

"Idempotent" sind Arbeitsvorgänge, die beliebig oft ausgeführt werden können; Beispiel: Lese Block 4 der Datei xyz. Nicht idempotent ist die Überweisung eines Geldbetrages von einem Konto auf ein anderes.

2. Nicht-idempotente Arbeitsvorgänge

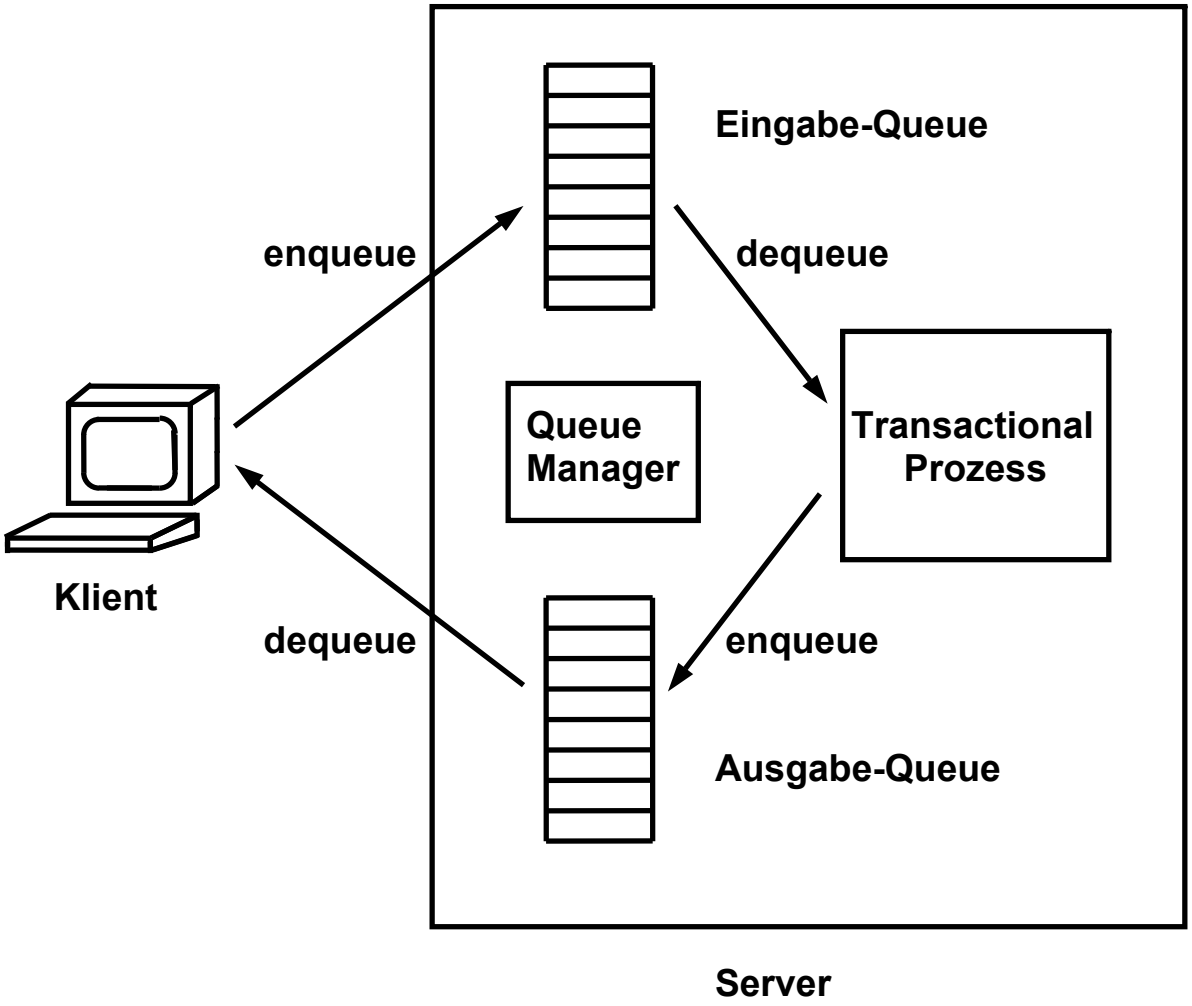
Problem: Wie wird festgestellt, ob Arbeitsgang durchgeführt wurde oder nicht?

"Genau einmal" (exactly once).
Überweisung von Konto x nach y

"Höchstens einmal" (at most once).
Stimmabgabe Bundestagswahl

"Wenigstens einmal" (at least once). Ideal für idempotente Vorgänge.
Update Virenscanner

Transaktionen stellen eine „exactly once“ Semantik sicher.



Getrennte Warteschlangen (Queues) für eingehende und ausgehende Nachrichten lösen viele Probleme. Die Warteschlangen sind persistent (z.B. auf einem Plattenspeicher) gespeichert.

Wenn der Klient abstürzt (Waise), stellt der Server das Ergebnis der Verarbeitung in die Ausgabe Queue. Die Transaktion ist damit vom Standpunkt des Servers abgeschlossen. Das Verarbeitungs-ergebnis kann beim Wiederanlauf des Klienten aus der Ausgabe-Queue abgeholt werden.

Andere Aufgaben der Queues:

- Prioritätensteuerung
- Lastverteilung auf mehrere Server

Fehlerbehandlung - Ausfall des Klienten

Queue Manager des TP Monitors

Jede einzelne Transaktion wird in 3 Subtransaktionen aufgelöst, die alle eigene ACID Eigenschaften haben.

1. **Subtransaktion 1 nimmt die Eingabenachricht entgegen, stellt sie in die Eingabe Queue, und commits.**
2. **Subtransaktion 2 dequeues die Nachricht, verarbeitet sie, stellt das Ergebnis in die Ausgabe-queue, löscht den Eintrag in der Eingabequeue und commits.**
3. **Subtransaktion 3 übernimmt das Ergebnis von der Ausgabequeue, übergibt das Ergebnis an den Klienten, löscht den Eintrag in der Ausgabequeue und commits.**

Ein eigener Queue Manager ist optimiert für diese Aufgabe.

Die Queues sind auf einem RAID Plattenspeicher persistent gespeichert. Für den TP Monitor ist die Transaktion abgeschlossen, wenn die Ergebnisse in der Ausgabe Queue festgehalten wurden, auch wenn der Klient zwischenzeitlich ausgefallen ist.

Sicherheitssystem

Unter z/OS werden die Basis-Sicherheitserfordernisse durch den z/OS Security Server abgedeckt, z.B. mit Hilfe von RACF oder einer äquivalenten Software Komponente, z.B. ACS von der Firma Computer Associates. Diese ist für den Login-Prozess zuständig und regelt weiterhin Zugriffsrechte, z.B. auf bestimmte Daten.

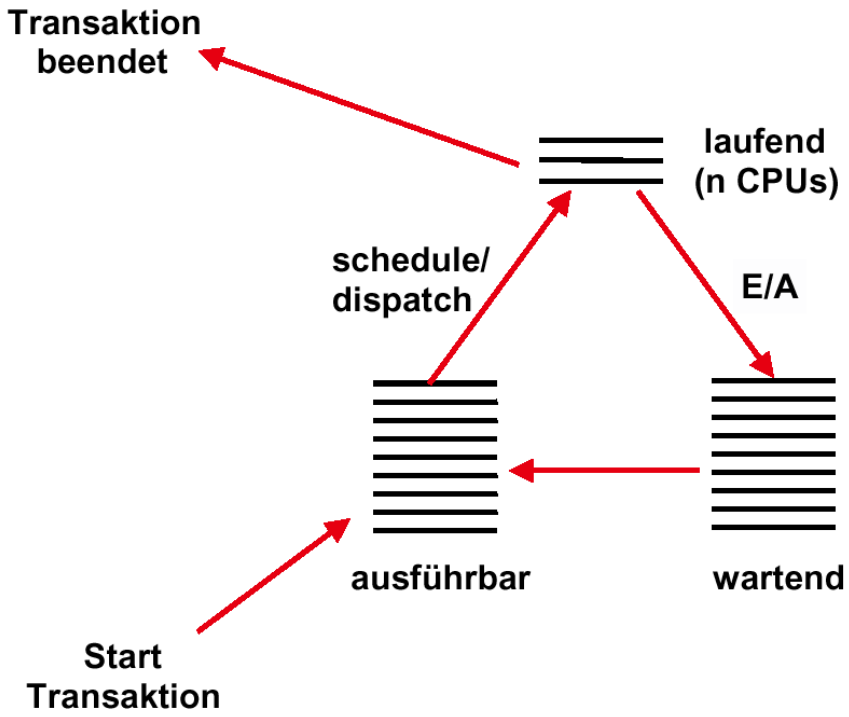
In vielen Fällen ist es zusätzlich erforderlich, eine weitergehende Klienten-Authentifizierung durchzuführen und/oder Daten kryptographisch verschlüsselt zu übertragen. Hierfür werden Sicherheitsprotokolle wie SSL (Secure Socket Layer), TLS (Transport Layer Security) oder Kerberos eingesetzt.

**Kerberos, ursprünglich vom Massachusetts Institute of Technology (MIT) entwickelt, gehört zum z/OS Lieferumfang und ist in Mainframe Installationen weit verbreitet.
Daneben werden SSL (TLS) und IPsec unter z/OS eingesetzt.
Kerberos hat gegenüber SSL den Vorteil, dass ausschließlich symmetrische kryptografische Schlüssel einsetzt werden, was besonders bei kurzen Transaktionen die Performance verbessert.**

Kerberos (Κέρβερος) ist der Name des dreiköpfigen Höllenhundes, der in der antiken griechischen Mythologie den Hades (die Unterwelt) bewacht.

Herkules besiegt Kerberos

Zustand von Prozessen



Jede Transaktion rotiert wiederholt durch die Zustände laufend, wartend und ausführbar. Der Scheduler des TP Monitors selektiert aus der Queue ausführbare Transaktionen jeweils einen Kandidaten wenn immer eine CPU frei wird.

In einem Rechner laufen immer zahlreiche Prozesse gleichzeitig ab. Die Prozesse befinden sich immer in einem von drei Zuständen. Hierfür werden drei Wartschlangen benutzt, in denen sich die TCBs der Prozesse befinden.

Ein Prozess befindet sich im Zustand laufend, wenn er von einer der verfügbaren CPUs ausgeführt wird. (Die allermeisten Mainframes sind Mehrfachrechner und verfügen über mehrere CPUs, bis zu 80 bei einem z196 EC).

Ein Prozess befindet sich im Zustand wartend, wenn er auf den Abschluss einer I/O Operation wartet.

Ein Prozess befindet sich im Zustand ausführbar, wenn er darauf wartet, dass die Scheduler/Dispatcher Komponente ihn auf einer frei geworden CPU in den Zustand laufend versetzt.

TP Monitor Repository

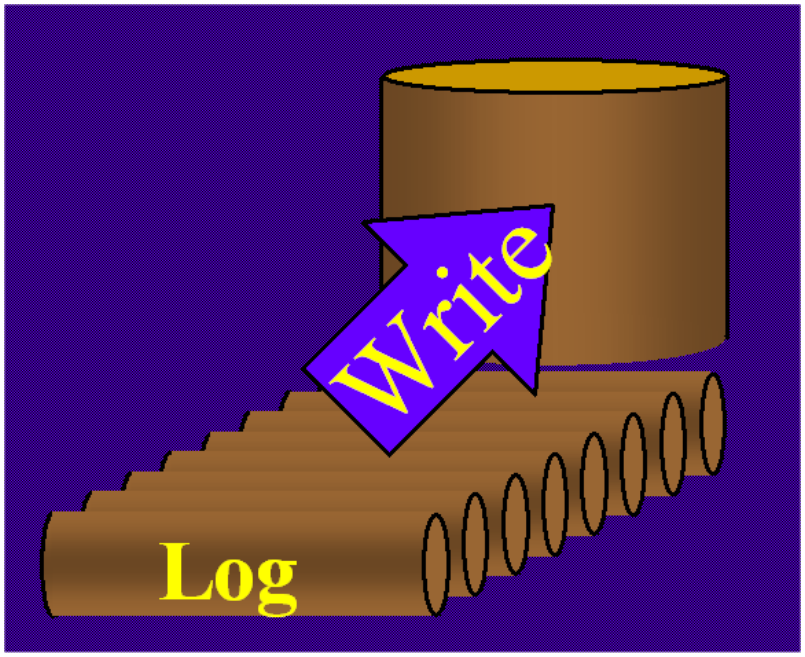
Das Repository des TP Monitors speichert alle Daten, die für die interne Ablaufsteuerung benötigt werden. Andere Bezeichnungen sind Katalog oder Dictionary.

Im Repository werden katalogisiert:

Benutzer Rechte
Workstation IDs (Terminal IDs)
Screens
Anwendungen
Prozeduren
Tabellen
Rollen (Wer kann/darf/soll wann was machen)

- ●
- ●
- ●
- ●

Z.B. Hinzufügen eines Feldes in einen Screen ändert Client Software, Server Schnittstelle, fügt eine weitere Spalte in eine SQL Tabelle ein, ...



Log plus alter Zustand ergibt neuen Zustand.

Beim „Commit“ einer Transaktion wird das Log persistent gespeichert (z.B. auf einer gespiegelten oder einer RAID Festplatte).

Das Log wird zur Wiederherstellung einer Transaktion im Falle eines Fehlers benutzt:

- **System failure: lost in-memory updates**
- **Media failure (lost disk)**

Bewirkt die Dauerhaftigkeit (Durability) einer Transaktion.

Log File Konzept

Das Log ist eine historische Aufzeichnung aller Änderungen des Zustands (state) des TP Systems. Das Log ist eine sequentielle Datei. Das vollständige Log enthält die historische Entwicklung aller Datenbankänderungen.

Es existiert ein statisches Log, welches erlaubt, alle historisch abgelaufenen Transaktionsabläufe wiederherzustellen (reconstruct), sowie ein dynamisches Log, dessen Enträge nach erfolgreicher Durchführung einer Transaktion wieder gelöscht werden können.

Im Fall, dass eine Datenbank zerstört wird, kann mit Hilfe einer (nicht den neuesten Stand enthaltenden) Backup Kopie sowie des statischen Logs die zerstörte Datenbank wieder hergestellt werden, indem man gegen die Backup Kopie alle seither stattgefundene Transaktionen laufen lässt.

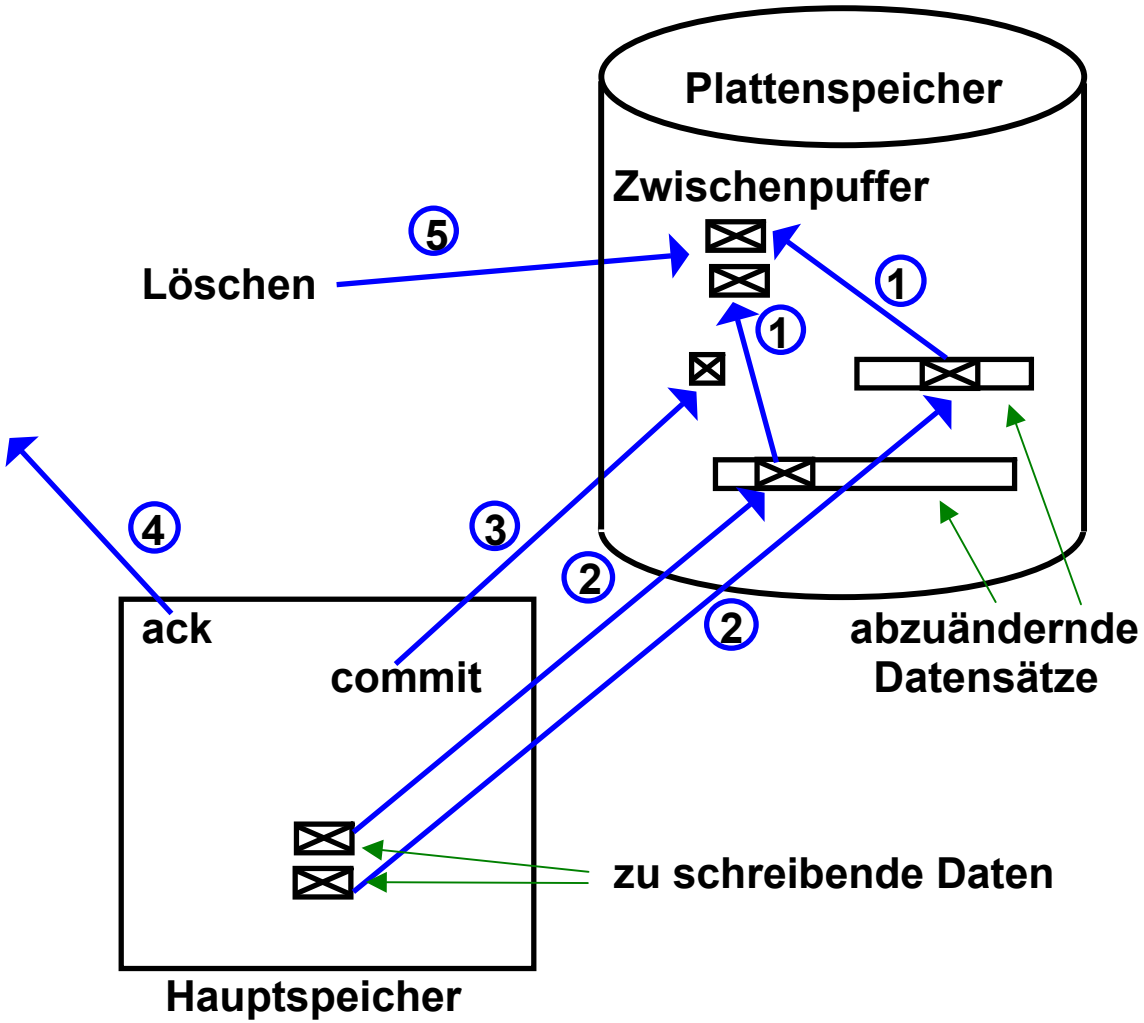
Backward Recovery

Die Backward Recovery Manager Komponente des Transaktionsmonitors stellt sicher, dass im Fehlerfall der teilweise Ablauf einer Transaktion rückgängig gemacht wird, und dass alle abgeänderten Felder einer Datenbank wieder in ihren ursprünglichen Zustand zurückübersetzt werden.

Andere Bezeichnungen für Backward Recovery sind:

- **backout**
- **rollback**
- **abort**

Der Recovery Manager benutzt hierfür Daten, die entweder in temporären Zwischenpuffern persistent auf der Festplatte oder in dem Log file (oder beiden) festgehalten werden.



Andere Bezeichnungen sind: backout, roll back oder abort.

Zur Ermöglichung einer eventuellen Recovery finden bei jedem Write Vorgang die folgenden Schritte statt:

1. abzuändernde Information in Puffer zwischenspeichern
2. Datensätze überschreiben
3. Commit Status festschreiben. Damit ist der ACID-relevante Teil der Transaktion abgeschlossen.
4. erfolgreiches Commit dem Benutzer mitteilen
5. Zwischenpuffer löschen

Der Zwischenpuffer ist Teil eines dynamischen Logs.

Backward Recovery

Recoverable Resources

Gewisse Daten werden von einem Transaction Processing Monitor als *recoverable resource* behandelt. Hierzu gehören:

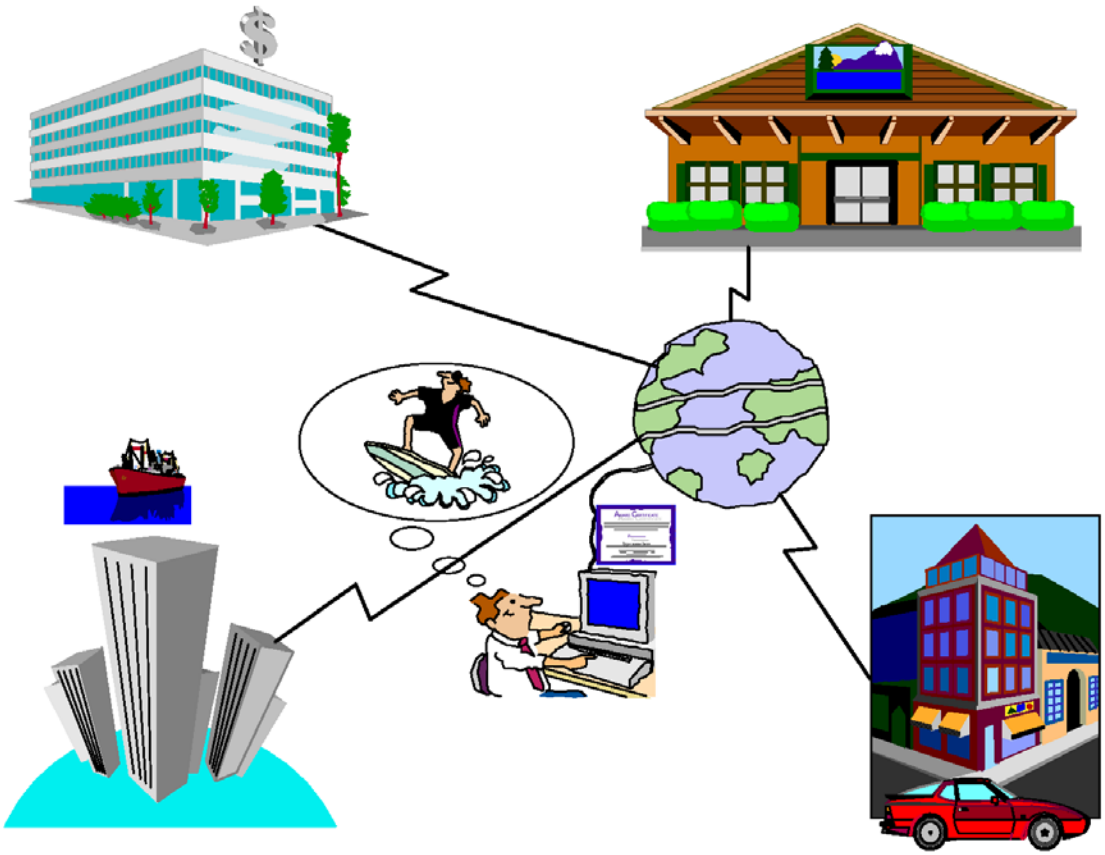
- **Alle von Anwendungsprogrammen verwendeten Dateien und Datenbanken**
- **Gewisse transiente Daten und temporary storage queues**

Für diese Daten wird recovery information vom Transaction Monitor logged oder aufgezeichnet (recorded). Hierfür dient ein dynamisches Log. Wenn eine Transaktion erfolgreich abgeschlossen wird, wird die recovery information in dem dynamischen Log gelöscht. Unabhängig davon existiert ein statisches Log, in dem alle transaktional vorgenommenen Datenänderungen durch erfolgreich durchgeführte Transaktionen festgehalten werden.

Wenn eine Transaktion nicht ausgeführt werden kann (z.B. Fehler im Anwendungsprogramm, im Datenzugriff oder aus anderen Gründen), dann führt der Transaction Monitor ein dynamic transaction backout (DTB) oder Rollback aus.

Dies macht alle bisherigen, vor dem Transaktionsabbruch durchgeführten Updates (Datenänderungen) rückgängig. DTB verhindert, dass corrupted Data von anderen Tasks oder Transaktionen benutzt werden.

Der Abbruch einer Transaktionsverarbeitung wird in der z/OS Terminologie als „abnormal end“, oder abgekürzt “abend“ bezeichnet.



Distributed
Flat
Transaction

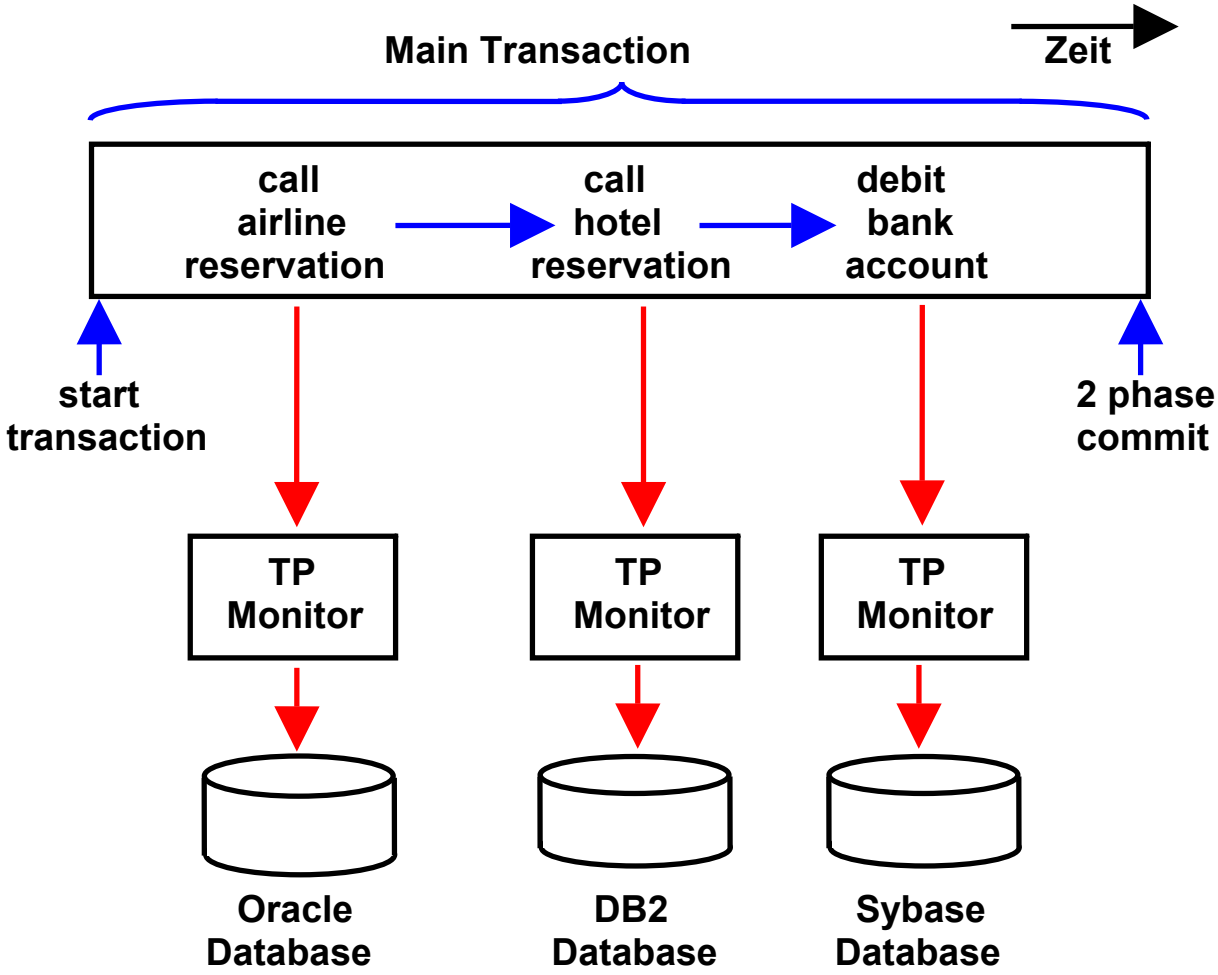
Angenommen, ein Benutzer geht zu einem Reisebüro um dort eine Urlaubsreise zu buchen. Nachdem er den Flug, das Hotel und den Mietwagen selektiert hat, beschließt das Reisebüro, die Reise über das Internet zu buchen.

Hierzu startet das Reisebüro eine Transaktion, welche

- **den Interessenten authentifiziert und in der Datenbank des Reisebüros speichert.,**
- **die Flugreservierung im Datenbanksystem der Fluglinie aufzeichnet,**
- **die Raumreservierung im Computer des Hotels vornimmt,**
- **das Gleiche im Datenbanksystem der Autovermietung vornimmt,**
- **den gesamten Preis vom Bankkonto des Interessenten abbucht,**
- **Überweisungen auf die Bankkonten der anderen Teilnehmer der Transaktion vornimmt,**
- **für den Benutzer eine Multimedia File mit Information über das Paket an Reservierungen erstellt, und**
- **eine Bestätigung über die erfolgreiche Durchführung an den Interessenten sendet.**

Offensichtlich sind hier mehrere TP Monitore auf geographisch verstreuten Systemen involviert.

Wichtig ist, dass die gesamte Datenverarbeitung als eine atomare Transaktion durchgeführt wird. Der Interessent wird nicht zufrieden sein, wenn Hotel und Mietwagen gebucht wurden, das Geld vom Konto abgebucht wurde, aber der Flug wegen Überbuchung nicht verfügbar ist. Deshalb muss jeder Teil der Transaktion erfolgreich durchgeführt werden, oder die Transaktion insgesamt darf nicht ausgeführt werden.



In dem gezeigten Beispiel beinhaltet die Transaktion für die Urlaubsreservierung (Main Transaction) Aufrufe für drei unterschiedliche TP Monitore und drei verschiedene Datenbanken.

Dieser Fall wäre mit einer nested Transaktion lösbar. Nested Transaktionen werden in der Praxis aber nicht eingesetzt.

An deren Stelle tritt das Two-Phase Commit Protokoll, welches Bestandteil fast aller TP Monitore ist, besonders auch der unter z/OS verfügbaren TP Monitore.

Die Two-Phase Commit Management Komponente wird auch als Sync-Point Manager bezeichnet.

Distributed Flat Transaction

Two-phase Commit Protokoll

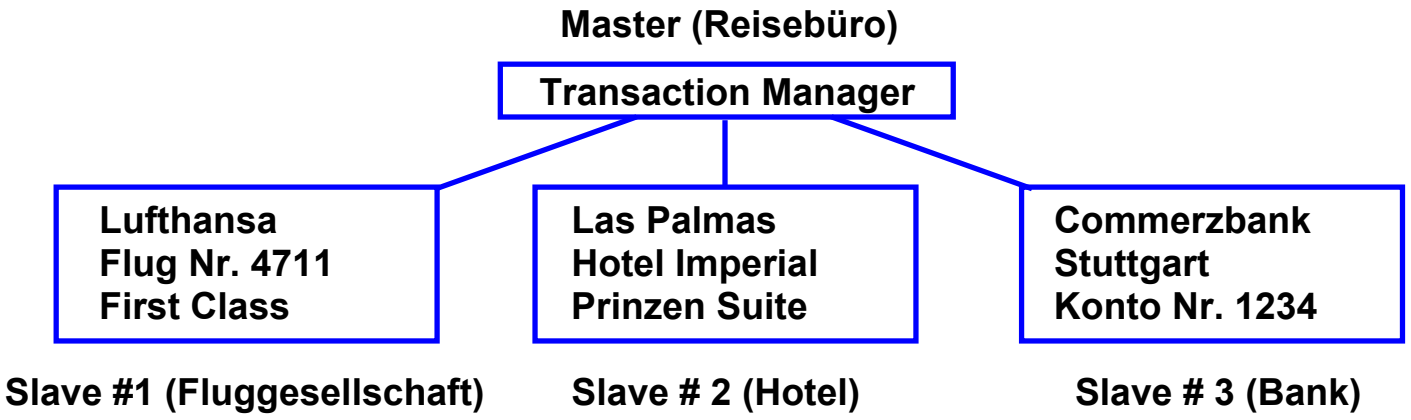
Das 2-Phase Commit Protokoll steuert die gleichzeitige Änderung mehrerer Datenbanken, z.B. bei der Urlaubsreise-Buchung die Änderung der Datenbanken der Fluggesellschaft, des Hotels und des Bankkontos von dem die Bezahlung der Reise abgebucht wird. Das Update der drei Datenbanken erfolgt durch unabhängige TP Monitore.

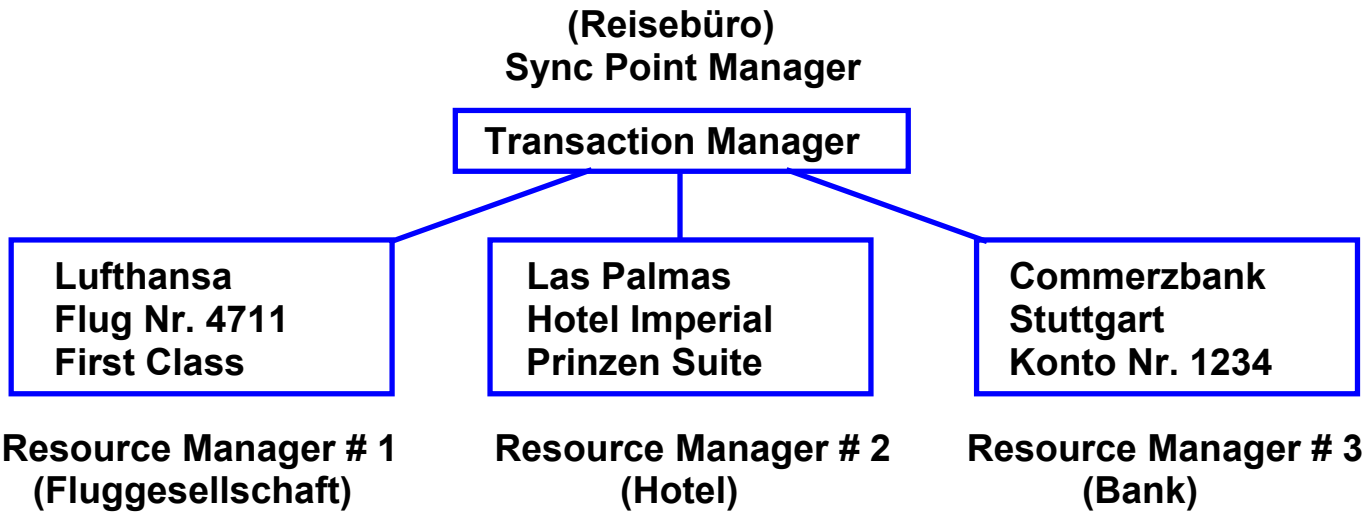
Ein Problem tritt auf, wenn eines der Updates nicht erfolgen kann, z.B. weil das Hotel ausgebucht ist. Daher sind atomare Transaktionen erforderlich

Die Konsistenz wird erreicht durch einen „Master“, der die Arbeit von „Slaves“ überwacht. Der TP Monitor des Reisebüros übernimmt die Rolle des Masters, die TP Monitore der Fluggesellschaft, des Hotels und der Bank sind die Slaves.

Der Master sendet Nachrichten an die drei Slaves. Jeder Slave markiert die Buchung als tentativ und antwortet mit einer Bestätigung (Phase 1).

Wenn alle Slaves ok sagen sendet der Master eine Commit Nachricht, ansonsten eine Rollback Nachricht (Phase 2).





Der X/Open Standard verwendet eine andere Terminologie.

Der Master wird auch als Transaction Manager, Recovery Manager oder Sync Point Manager bezeichnet. Slaves werden als Resource Manager bezeichnet.

IBM bezeichnet den Master häufig als Sync Point Manager.

Master	Slave
Begin atomic action Send Request 1 ..... Send Request n Send „ Prepare to commit“	
	if action can be performed then begin Lock data Store initial state on disk Store requests on disk Send „OK“ end else Send „Failure“
Ende Phase 1	
if all slaves said „OK“ then send „Commit“ else send „Rollback“ Wait for acknowledgements	
	if master said commt then begin Do work Unlock data end Send „Acknowledgement“
Ende Phase 2	

2-Phase Commit Protokoll

Der Master fragt bei allen beteiligten Slaves an, ob die gewünschte Aktion machbar ist. Jeder Slave markiert die Anforderung als tentativ und schickt eine Bestätigung an den Master (Phase 1).

Wenn alle Slaves positiv antworten sendet der Master eine commit Aufforderung, worauf die Slaves die tentative Änderung permanent machen. Wenn einer der Slaves seine Transaktion nicht durchführen kann senst der Master an die Slaves eine Nachricht die tentative Änderung wieder rückgängig zu machen (Phase 2).

Alle Slaves bestätigen den Abschluss der 2-Phase Commit Transaktion.

Ende

In Phase 1 übergibt das Anwendungsprogramm die 2-Phase Aufforderung an den Syncpoint Manager. Dieser sendet einen PREPARE-Befehl an allen Ressource-Manager. In Reaktion auf den Befehl PREPARE, antwortet jeder an der Transaktion beteiligte Ressource-Manager an den Syncpoint Koordinator er ist bereit oder auch nicht.

Wenn der Syncpoint Manager Antworten von allen Resource Managern erhalten hat wird die Phase 2 eingeleitet. Der Syncpoint Manager sendet einen Commit oder Rollback-Befehl auf der Grundlage der vorherigen Antworten. Wenn auch nur einer der Resource Manager eine negativen Antwort gesendet hat, veranlasst der Syncpoint Manager ein Rollback der tentativen Änderungen in allen Resource Managern.

CICS enthält einen eigenen Syncpoint Manager, der mit dem EXEC CICS SYNCPOINT Command aufgerufen wird. Daneben und alternativ kann CICS einen generischen Recovery Manager „z/OS Resource Recovery Services“ (RRS), der von z/OS Resource Managern wie WebSphere, IMS, DB2, aber auch von CICS (an Stelle des CICS Syncpoint Managers) benutzt werden kann.

z/OS Resource Recovery Services
RRS

Ein Prozess, der transaktionale Anwendungen ausführt, wird als Resource Manager bezeichnet. Unter z/OS können mehrere Resource Manager gleichzeitig tätig sein. Beispiele sind die CICS, IMS und WebSphere Transaktionsmonitore, sowie Stored Procedures mehrerer Datenbanken wie IMS und DB2.

z/OS Resource Recovery Services (RRS) stellt einen Recovery Manager zur Verfügung, der das Two-Phase Commit Protokoll beinhaltet, und den jeder Resource Manager innerhalb von z/OS nutzen kann. Es ermöglicht Transaktionen, ein Update geschützter (protected) Resourcen vorzunehmen, die von unterschiedlichen Resource Managern (z.B. unterschiedlichen TP Monitoren) verwaltet werden.

RRS wird von neuen Resource Managern eingesetzt, sowie für die Nutzung neuartiger Funktionen durch existierende Resource Managers. Neu entwickelte Resource Manager Produkte unter z/OS (z.B. WebSphere Enterprise Java Beans) verwenden RRS, an Stelle einer nicht vorhandenen eigenen Two-phase Commit Protocol Komponente.

Existierende Tansaction Managers wie CICS verfügen bereits über viele Funktionen, die in RRS enthalten sind, können stattdessen aber auch RRS benutzen. Vermutlich wird sich RRS in der Zukunft zu einer zentralen z/OS Komponente für die Transaktionssteuerung entwickeln.

Ursprünglich besaß jede Komponente von z/OS, die Transaktionen ausführen konnte (z.B. CICS, IMS/DC, andere) ihren eigenen Two-Phase Commit Recovery Manager. Dieser kann durch RRS ersetzt werden.

http://www.redbooks.ibm.com/abstracts/sg246980.html