

Client/Server-Systeme

Prof. Dr.-Ing. Wilhelm Spruth

SS 2005

Teil 12

SAP System R/3

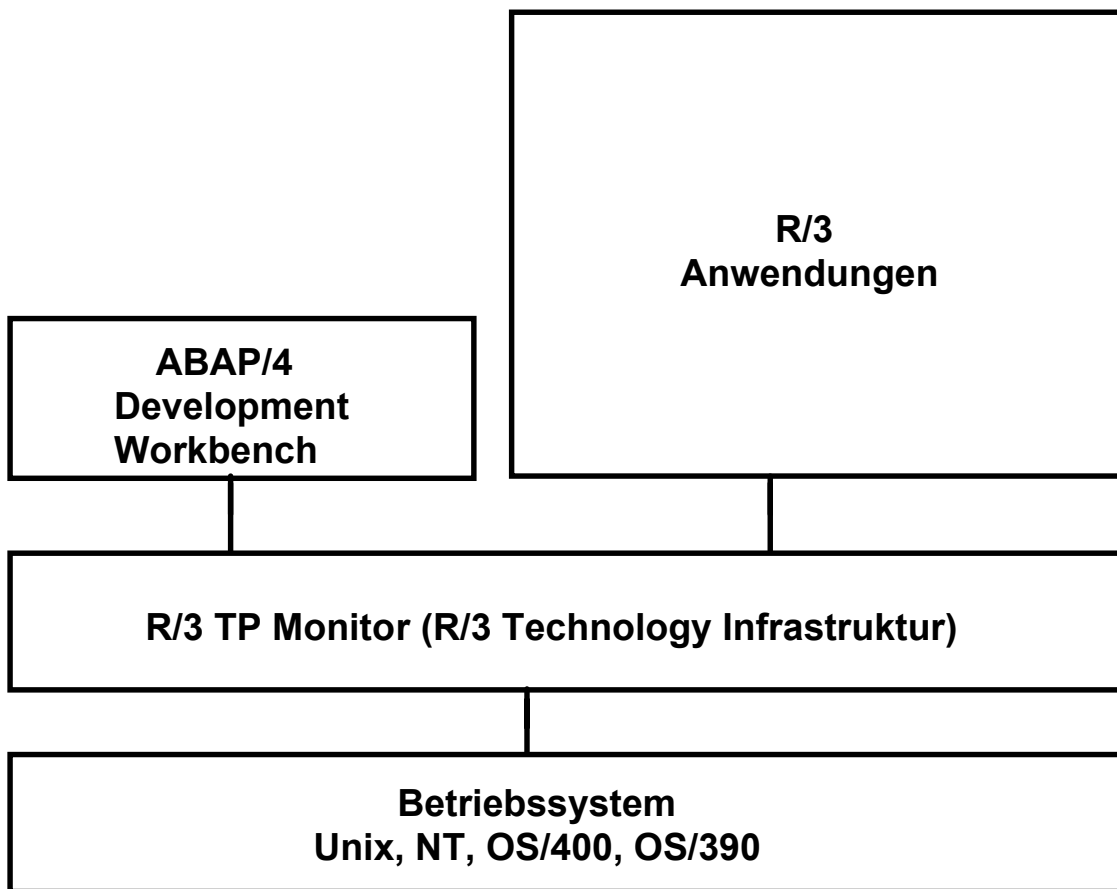
SAP System R/3

Literatur

R. Buck-Emden: „Die Client/Server Technologie des SAP System R/3“. Addison-Wesley 1996.

L. Will: „Administration des SAP System R/3“. Addison-Wesley 1997.

Es existieren zahlreiche Bücher über die System R/3 Programmierung unter ABAP/4 sowie über den betriebswirtschaftlichen Einsatz.



Schichtenarchitektur des SAP Systems R/3

Der R/3 Transaction Processing Monitor ist in C und in C++ geschrieben.

Alle von SAP vorgefertigten Anwendungsprogramme und alle vom Benutzer selbst erstellten Anwendungsprogramme sind in der SAP proprietären 4GL Sprache ABAP/4 geschrieben.

ABAP/4 ist eine interpretative Sprache; die Programme werden interpretativ abgearbeitet.

SAP System R/3

Version R/2 (seit 1972) im Einsatz auf S/370 Rechnern. Verwendet CICS als Transaktionsmonitor.

System R/3 Entwicklung startet 1988. Erste Anwendung wird 1989 auf der CeBIT gezeigt. Im Gegensatz zu System R/2 benutzt System R/3 seinen eigenen Transaktionsmonitor.

1991 erste Kundeninstallation. 1992 graphische Oberfläche SAPGUI verfügbar.

1995 Release 3.0 verfügbar. Sollte ursprünglich System R/2 ablösen. Eine signifikante Anzahl von System R/2 Installationen existiert noch heute, weil System R/3 ursprünglich nur auf Unix Rechnern verfügbar war.

1997 Portierung auf OS/390 Unix System Services.

"There are no plans for R/4 today"

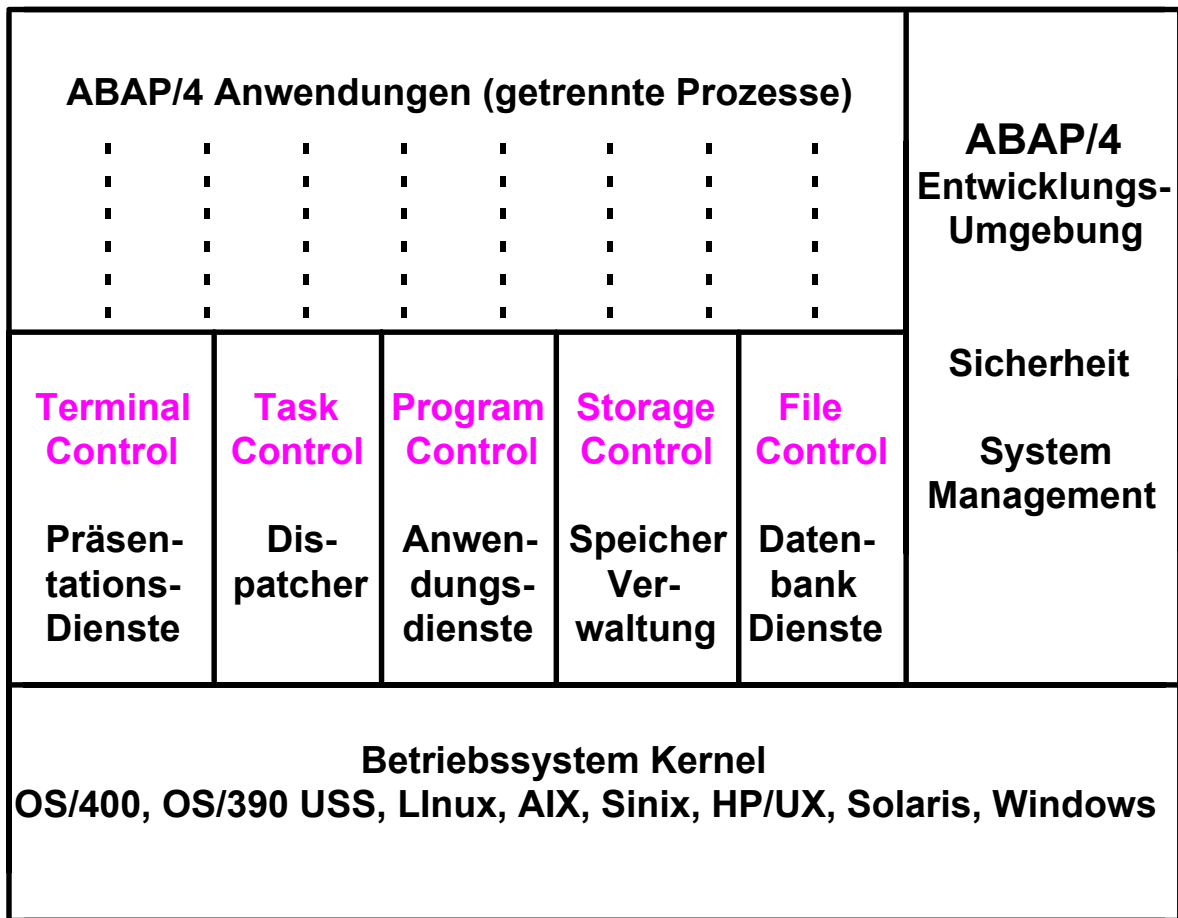
7,500+ SAP Kunden in 90+ Länderncountries

- 13 000+ R/3 Installationen (15 000 CICS)**
- 1 400+ R/2 Installationen**

60 % der Fortune 500 Unternehmen setzen SAP ein (CICS > 90 %)

81% des Umsatzes ausserhalb Deutschlands (IBM, Microsoft > 90%)

FF..FF



00..00

SAP System R/3 Komponenten

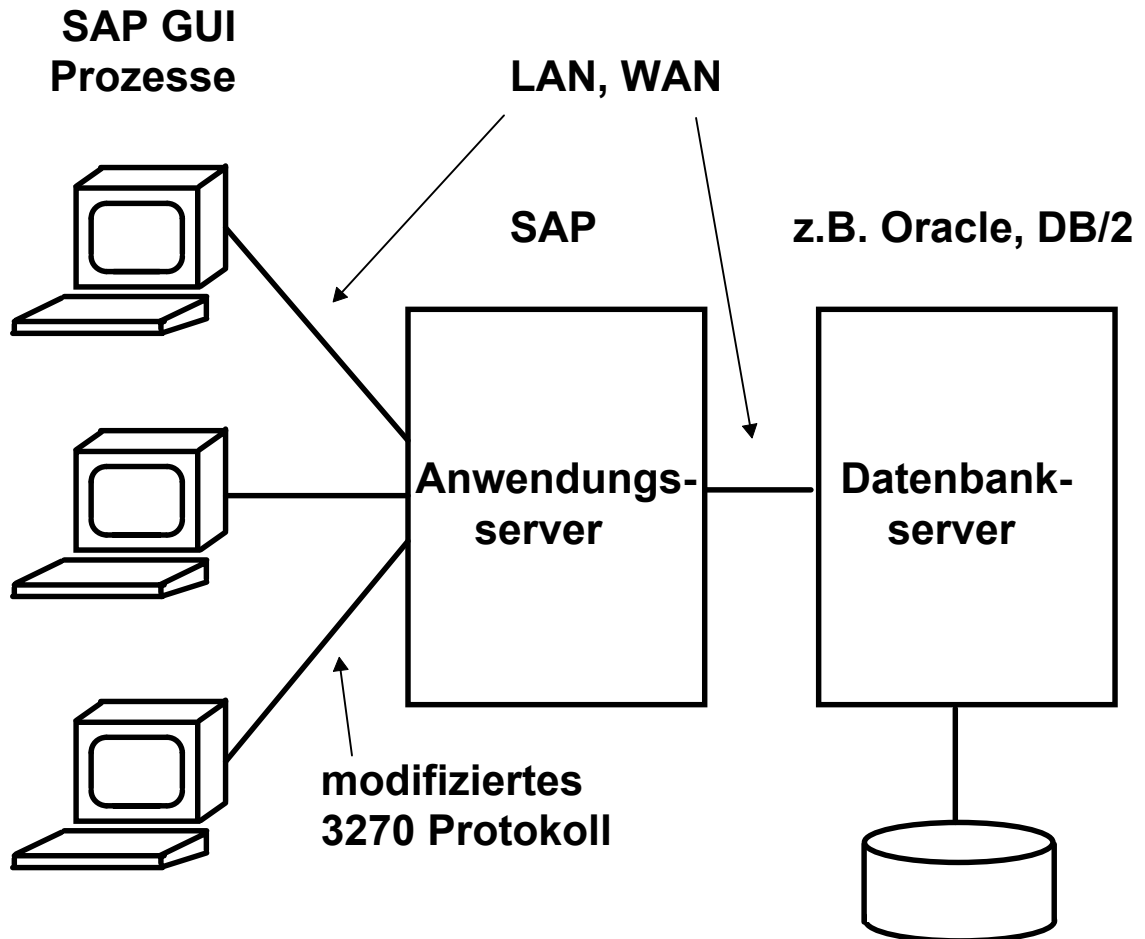
Im Gegensatz zu CICS werden für die ABAP/4 Anwendungen eigene Prozesse (z.B. Dialog Work Prozesse) eingerichtet.

Es existiert unter R/3 eine eigene ABAP/4 Entwicklungsumgebung.

CICS verwendet BMS als „native“ Präsentationsdienst und benutzt BMS zur Darstellung der 3270 „Green Screens“. Die SAP R/3 Präsentationsdienste haben eine vergleichbare Funktion für die SAPGUI. Die Datenübertragung erfolgt mit einem erweiterten 3270 Protokoll.

Im Falle von CICS werden für Sicherheit und System Management die integrierten OS/390 Funktionen benutzt. SAP/ R3 hat hierfür seine eigene Komponenten.

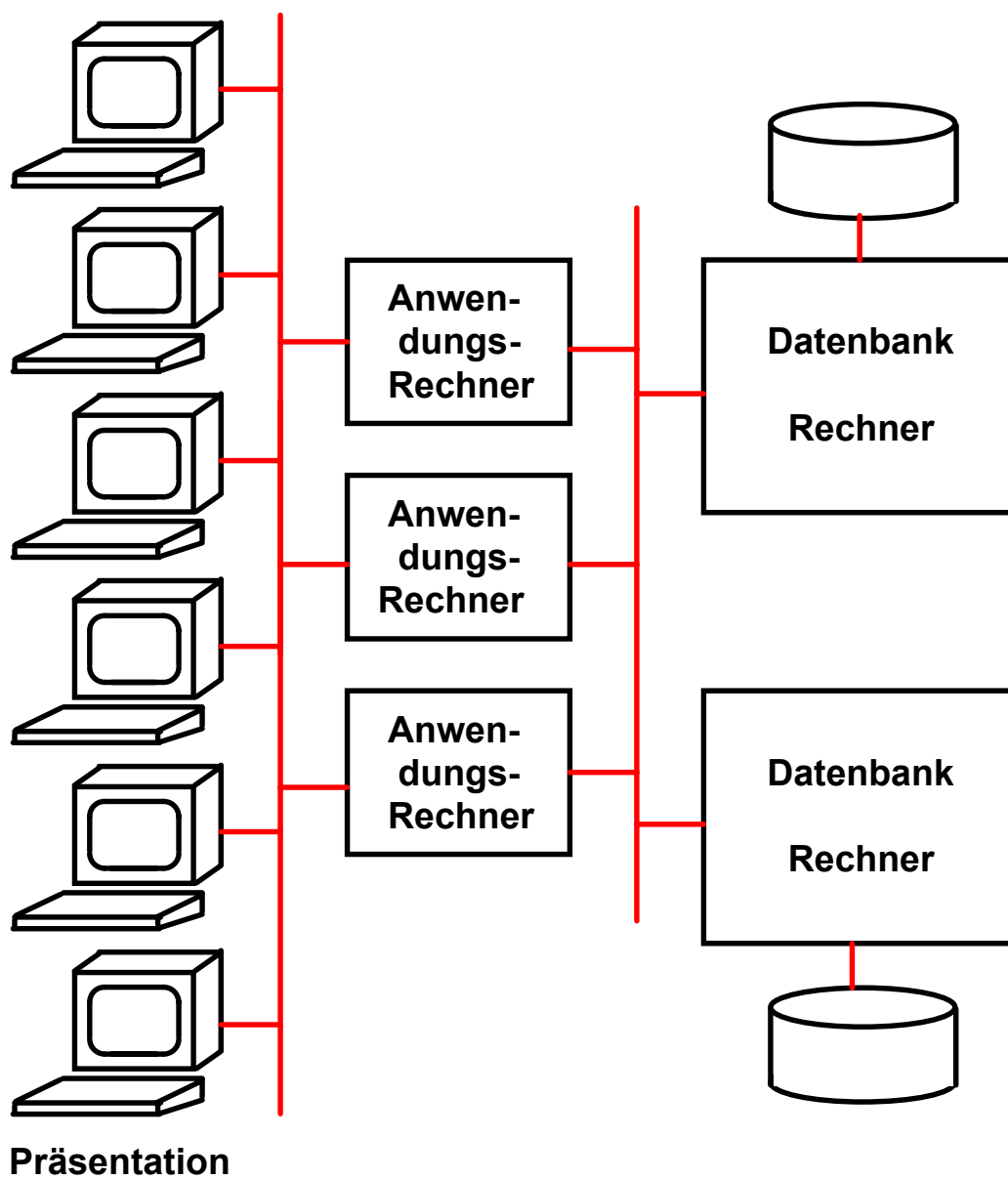
R/3 Präsentationsrechner



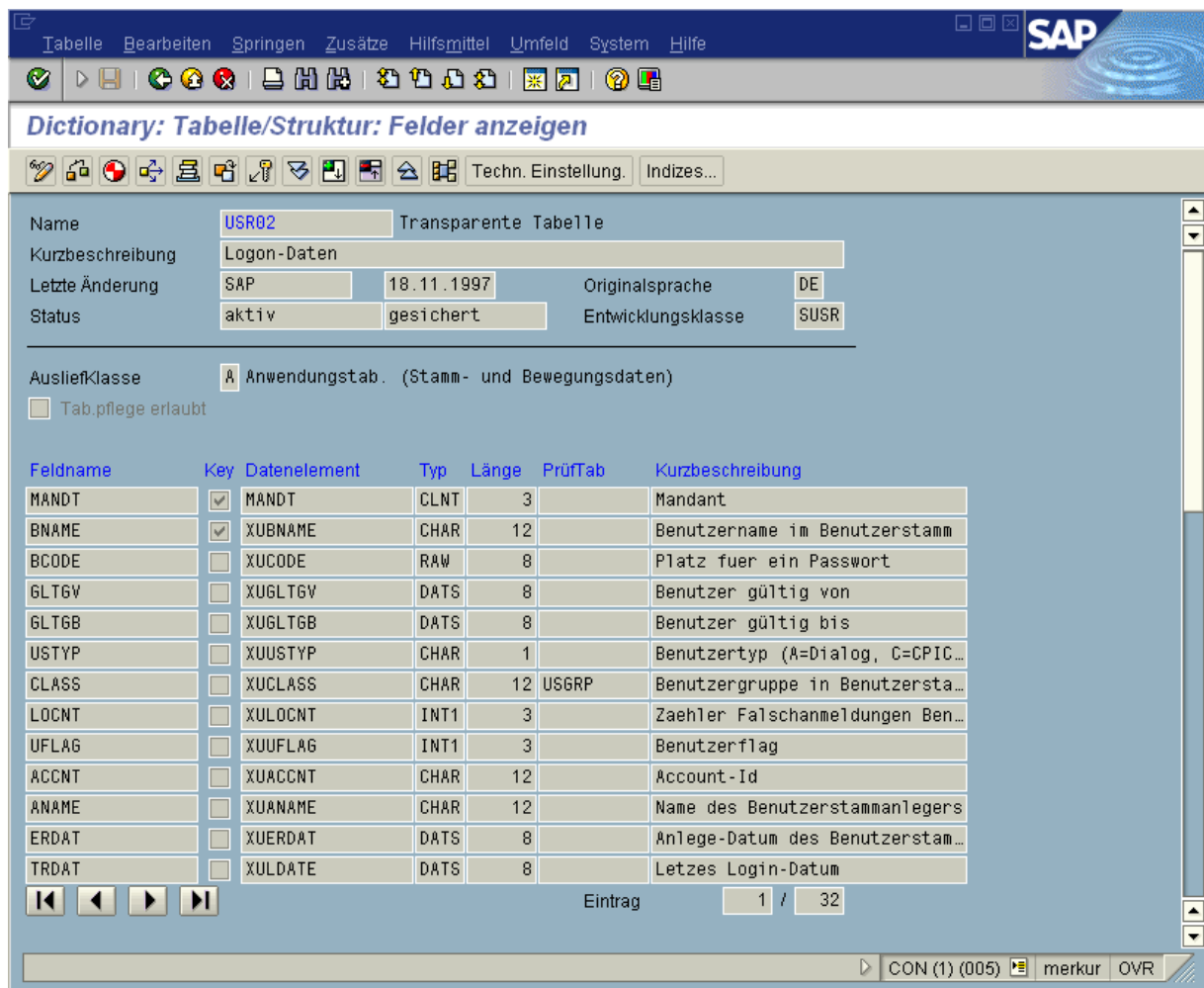
Dreistufige SAP R/3 Konfiguration

Saubere Trennung zwischen den Funktionen

- Präsentation
- Anwendung
- Datenbank



Beispiel der dreistufigen R/3 Client/Server Architektur

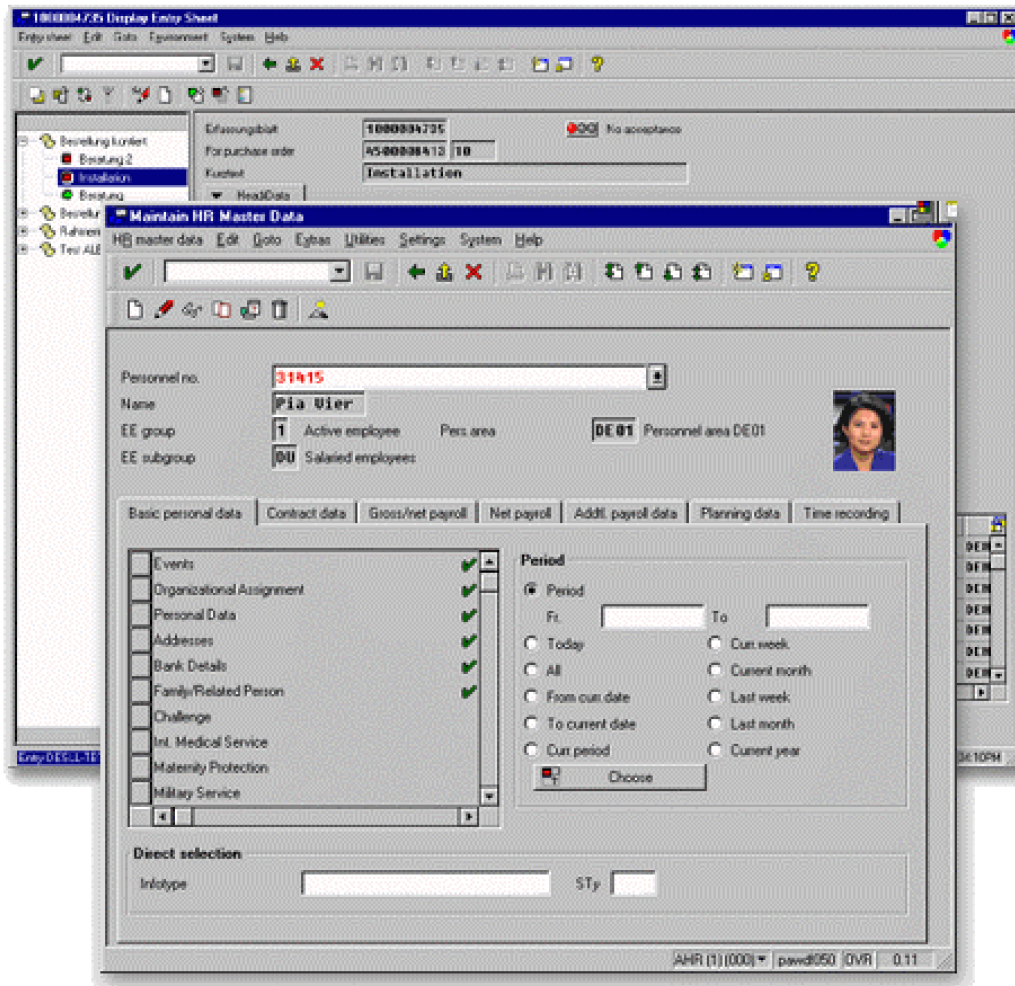


System R/3 SAPGUI

Zwischen dem SAP Applicationsserver und dem Präsentationsrechner werden generische, Plattform unabhängige Beschreibungen der graphischen Darstellung ausgetauscht, nicht aber die bereits komplett aufbereiteten Bildschirmbilder.

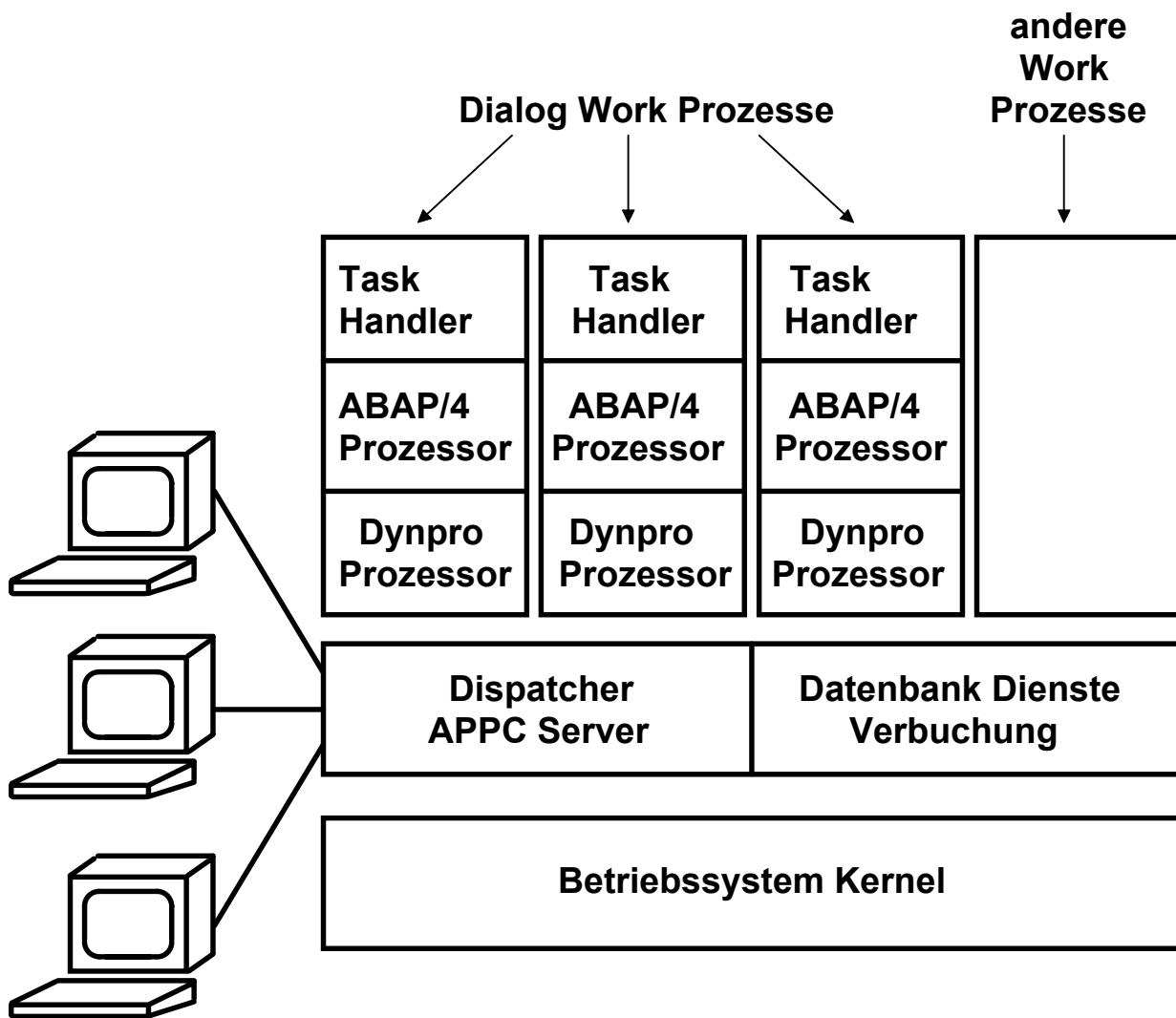
Die bei einem Bildschirmwechsel zu übertragende Datenmenge liegt typischerweise im Bereich von 1 - 2 KByte.

SAPGUI ist der SAP eigene Terminal Prozeß. Er wird für verschiedene Präsentationsumgebungen angeboten, wie z.B. OSF/Motiv oder Microsoft Windows.



SAPGUI

Mehrfache Fenster und eingebettete Bilder



R/3 Dispatcher und Task Handler

Das R/3 Laufzeitsystem ist als Menge paralleler, kooperierender Systemprozesse realisiert. Auf einem Applications-Server gehören hierzu ein zentraler Dispatcher und eine variable Anzahl von „Workprozessen“. Es gibt Workprozesse für die Verarbeitung von Dialogschritten, für die Verbuchung, Sperrverwaltung, Hintergrundverarbeitung, für das Spooling von Druckaufträgen u.A.

Die Aktivitäten innerhalb eines Workprozesses koordiniert ein „Task Handler“. Z.B. aktiviert im Falle eines Dialog-Workprozesses der Task Handler je nach Bedarf den Dynpro-Prozessor für die Verarbeitung der Bildschirm-Ablaufsteuerung und den ABAP/4-Prozessor für die Anwendungslogik.

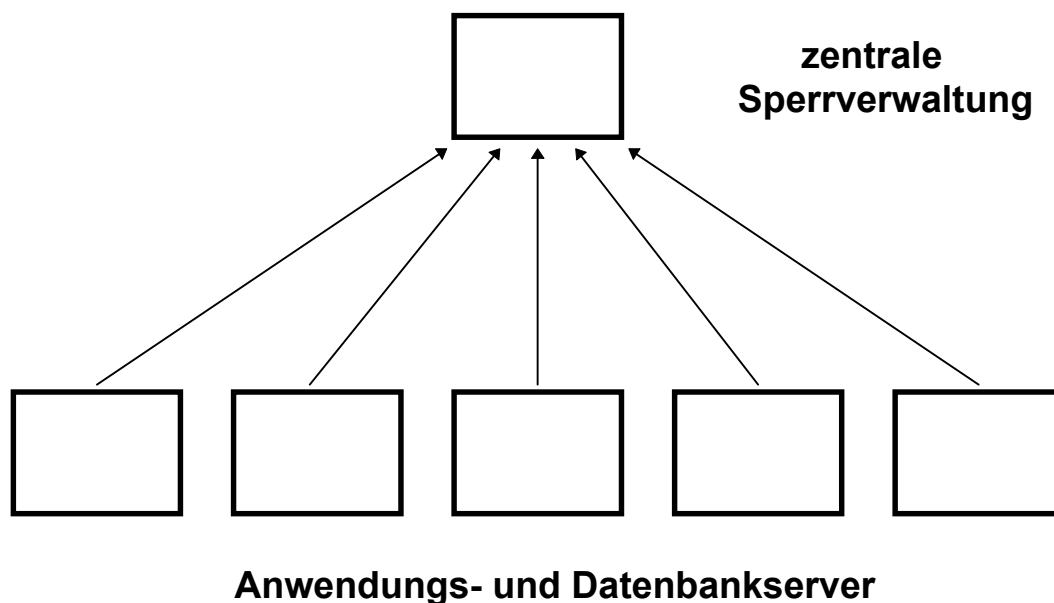
Sperrverwaltung

Da die als Bestandteil relationaler Datenbank Systeme verfügbaren Sperrmechanismen für die Transaktionsverwaltung nicht ausreichend sind, verfügt System R/3 über eine zentrale Sperrverwaltung.

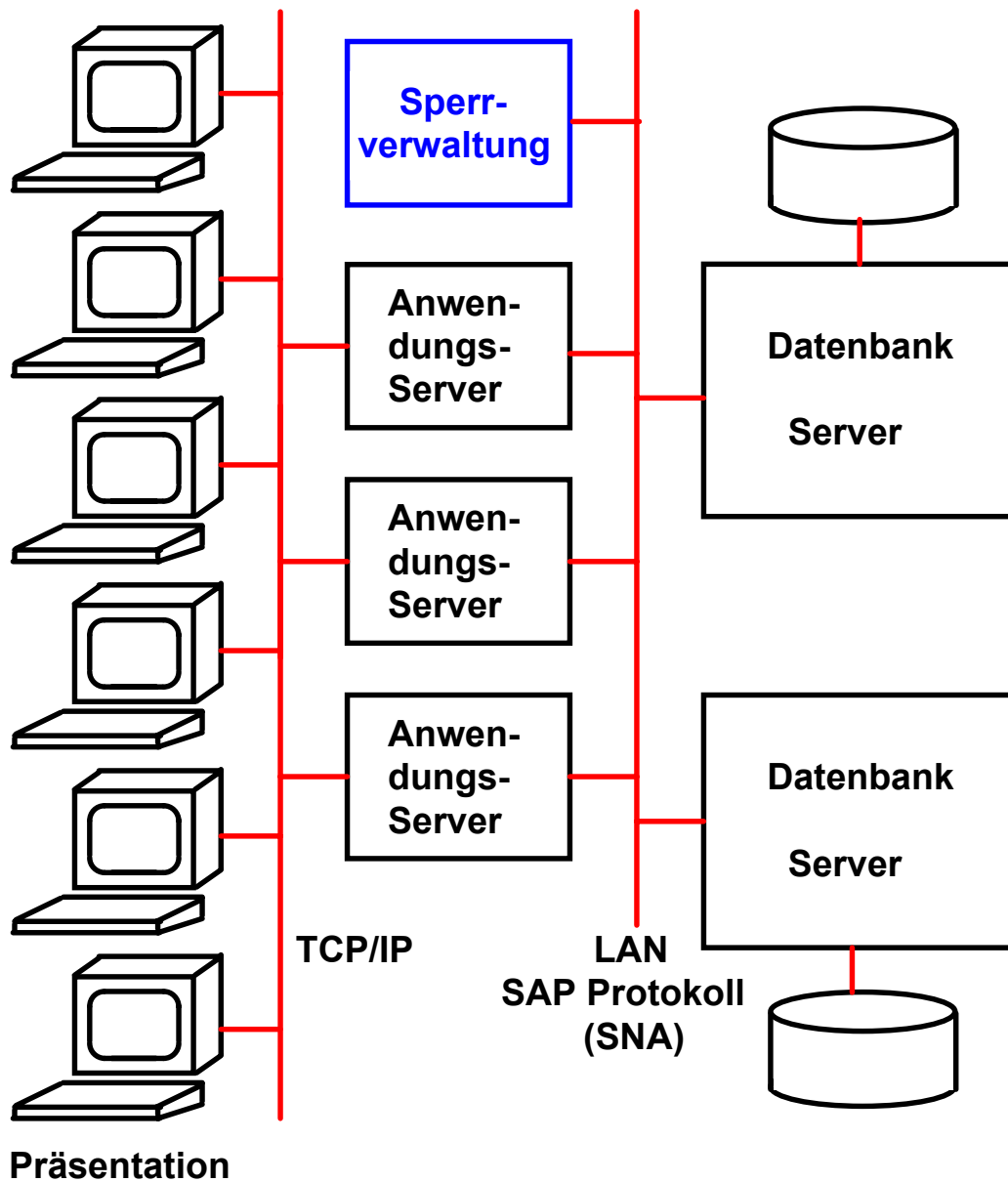
Die Sperrverwaltung stellt sicher, daß Transaktionen, deren Dialogschritte von unterschiedlichen Workprozessen bearbeitet werden, die von ihnen gesetzten Sperrern über Prozesswechsel hinweg behalten. Das Verbuchungsprogramm löscht nach erfolgter Verbuchung des vom Dialogteils der Transaktion erzeugten Protokollsatzes automatisch alle gesetzten Sperrern.

In einer verteilten Systemumgebung muß die Sperre nicht nur für denjenigen Anwendungsserver gelten, der die sperrende Transaktion durchführt, sondern für alle beteiligten Server. Jede R/3 Installation enthält deshalb eine zentrale Sperrverwaltung (Enqueue Service).

Der zentrale Sperrverwalter kann auf einem beliebigen Server untergebracht werden.



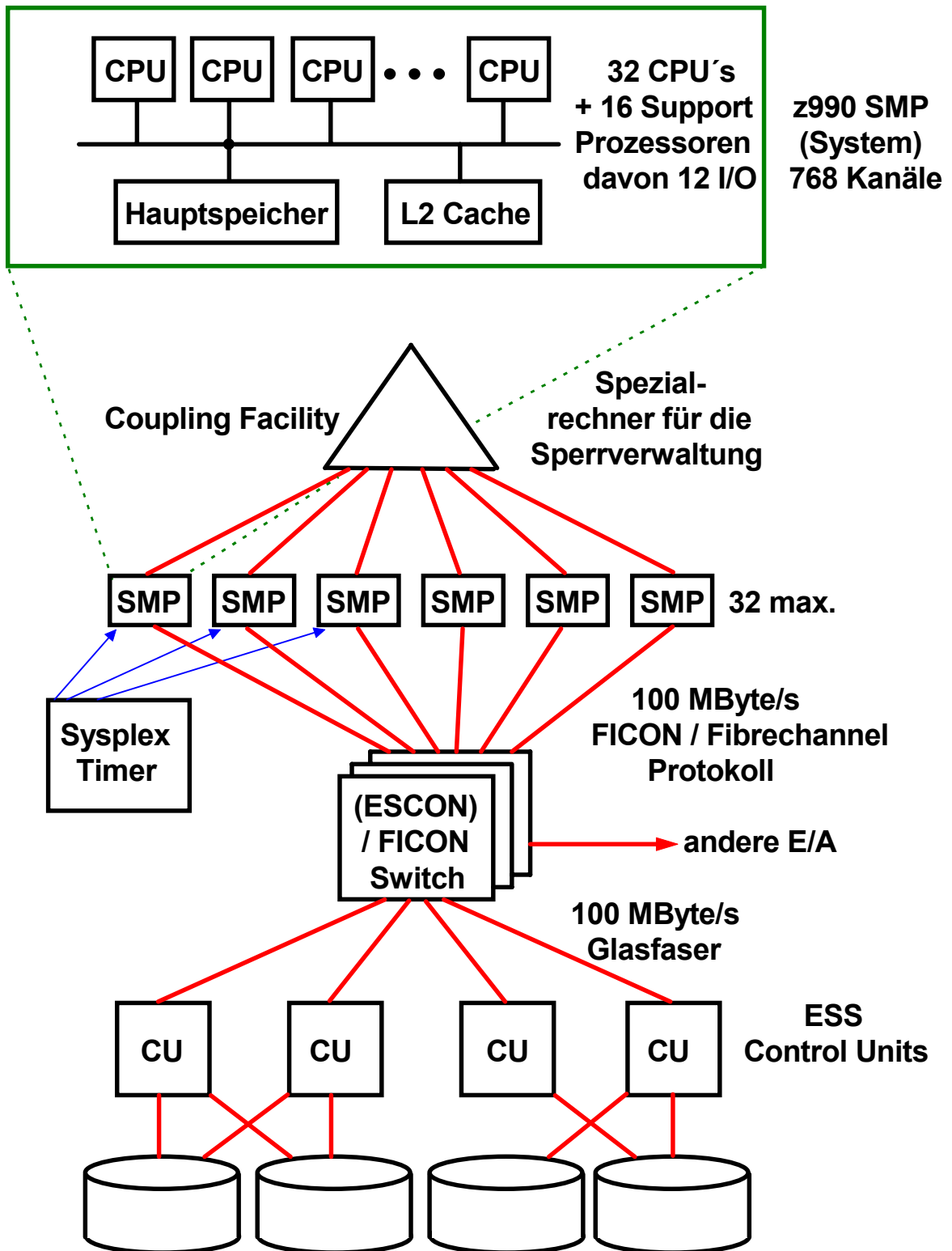
getrennte Rechner, häufig SMP
 Unix, Linux, z/OS Unix System Services



System R/3 Sperrverwaltungs-Server

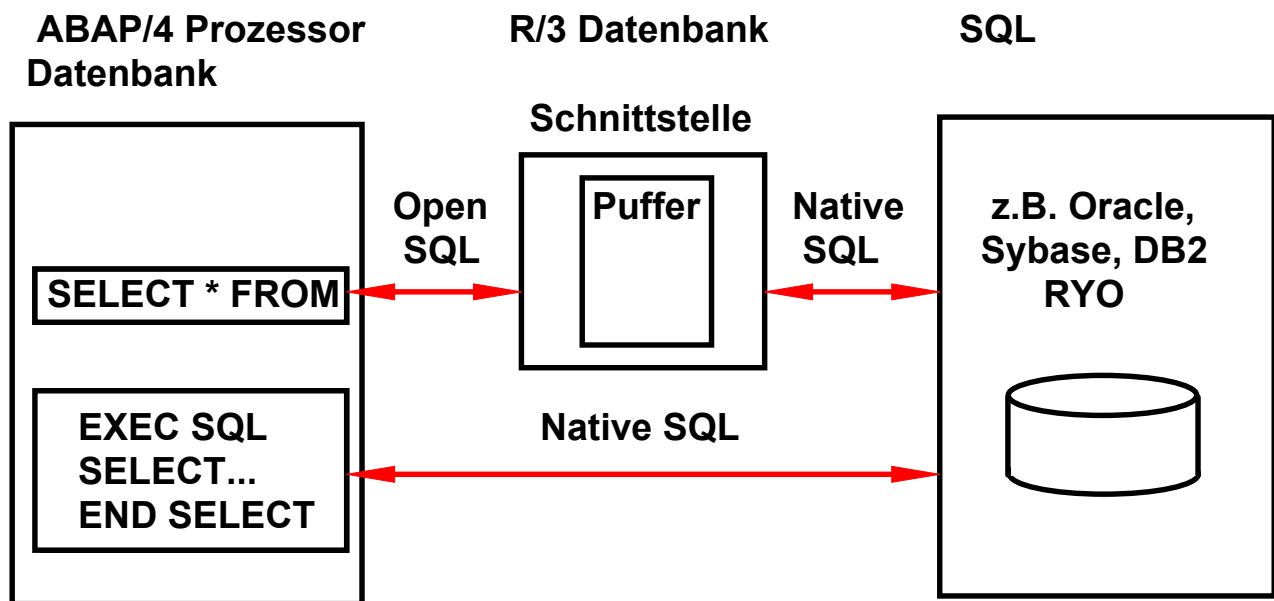
Sperrverwaltungs-Server (Lock Server) ist ein normaler Unix oder Linux Rechner, über normales Netzwerkprotokoll angeschlossen.

z/OS CICS, IMS und DB/2 Stored Procedures verwenden stattdessen „Coupling Facility“, mit speziell für die Transaktionsverwaltung optimierter Hard- und Software. Höhere Skalierung.



z/OS Parallel Sysplex

SMP IBM 990 Symetrischer Multiprocessor, bis zu 32 CPUs
ESS Enterprise Storage Subsystem



Relationale Datenbankzugriffe

Native SQL

Open SQL

optimierte SQL Anweisungen

Client Cache (LRU)

Die Datenbankschnittstelle benutzt keine eigene Netzwerkkomponente, sondern bedient sich der Möglichkeiten des RDBMS, wobei die herstellerspezifischen Verfahren eingesetzt werden.

SAP R/3 Anwendungen

Finanzwesen
Controlling
Anlagenwirtschaft
Materialwirtschaft
Produktionsplanung und -steuerung
Vertriebslogistik
Qualitätsmanagement
Instandhaltung
Projektmanagement
Servicemanagement
Personalwirtschaft
Bürokommunikation
Workflow-Funktionen
Branchenlösungen
Information Warehouse

cs 1336 ww6

wgs 04-98

Industry Strength

Verfügbarkeit
Funktionalität der Anwendungen
Daten Integrität
Vertraulichkeit
Authentifikation
Skalierbarkeit

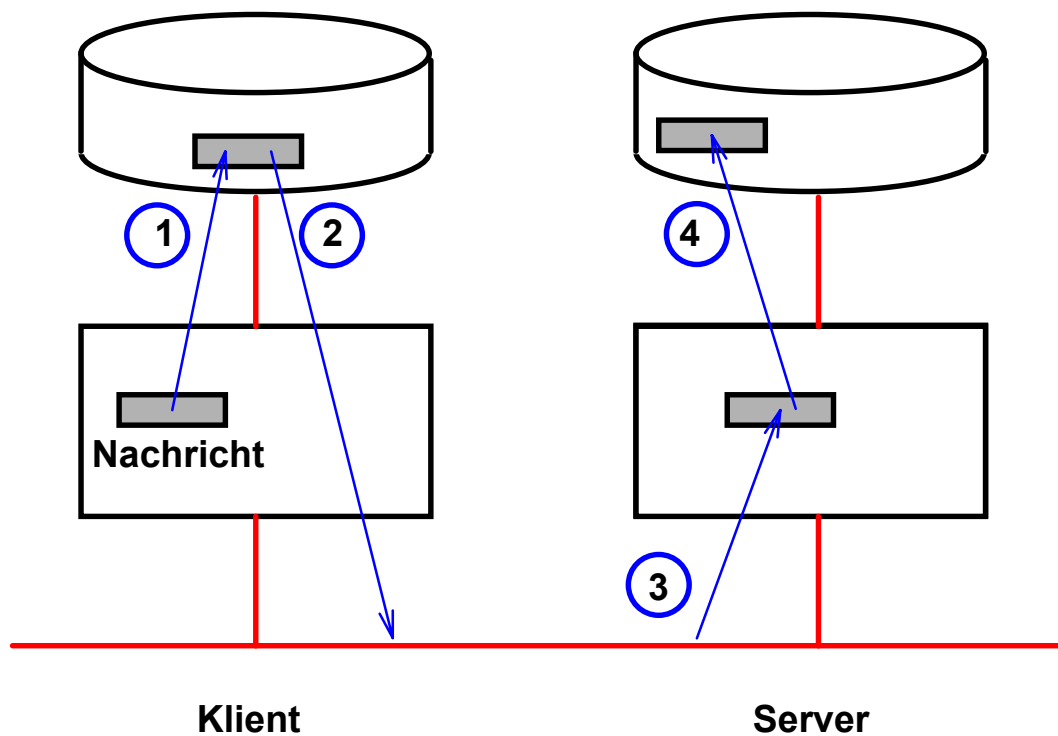
cs 1320 ww6

wgs 04-98

Message oriented Middleware (MOM)

Message based Queuing (MBQ)

MQSeries Implementierungsbeispiel



MBQ - Message Based Queuing, Message Oriented Middleware (MOM)

Alternative zum RPC, Ersatz für Batch file transfers

Eigenschaften

- Nachrichten werden bis zur endgültigen Auslieferung an die Zielanwendung in Warteschlangen zwischengespeichert
- Asynchron (im Gegensatz zum RPC, Store-and-Forward Prinzip)
- Recovery Mechanismen beim Versagen von Knoten oder Verbindungen
- Auslieferung wird garantiert
- Message Tracking (lokale Platte, entfernte Platte, Annahme der Nachricht durch Anwendung)

Die Operation wird durch einen *Queue Manager* auf jedem Rechner gesteuert

Message Based Queuing Message oriented Middleware

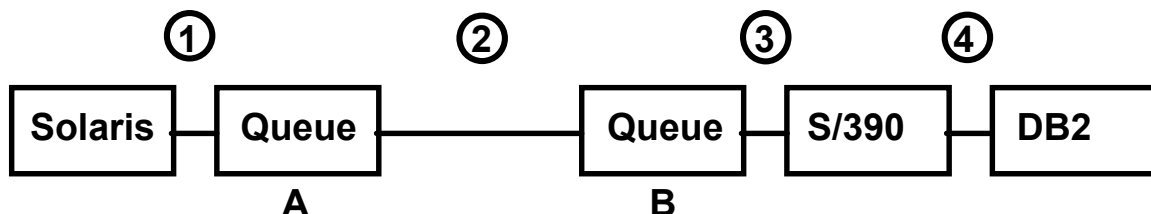


Message besteht aus zwei Teilen:

- Daten, die vom einem Programm zu einem anderen gesendet werden
- Message-Deskriptor oder Message-Header

Der Message-Deskriptor identifiziert die Message (Message-ID) und enthält Steuerinformationen (Attribute), wie z.B. Message-Type, Zeit-Ablauf, Korrelations-ID, Priorität und Namen der Antwort-Queue.

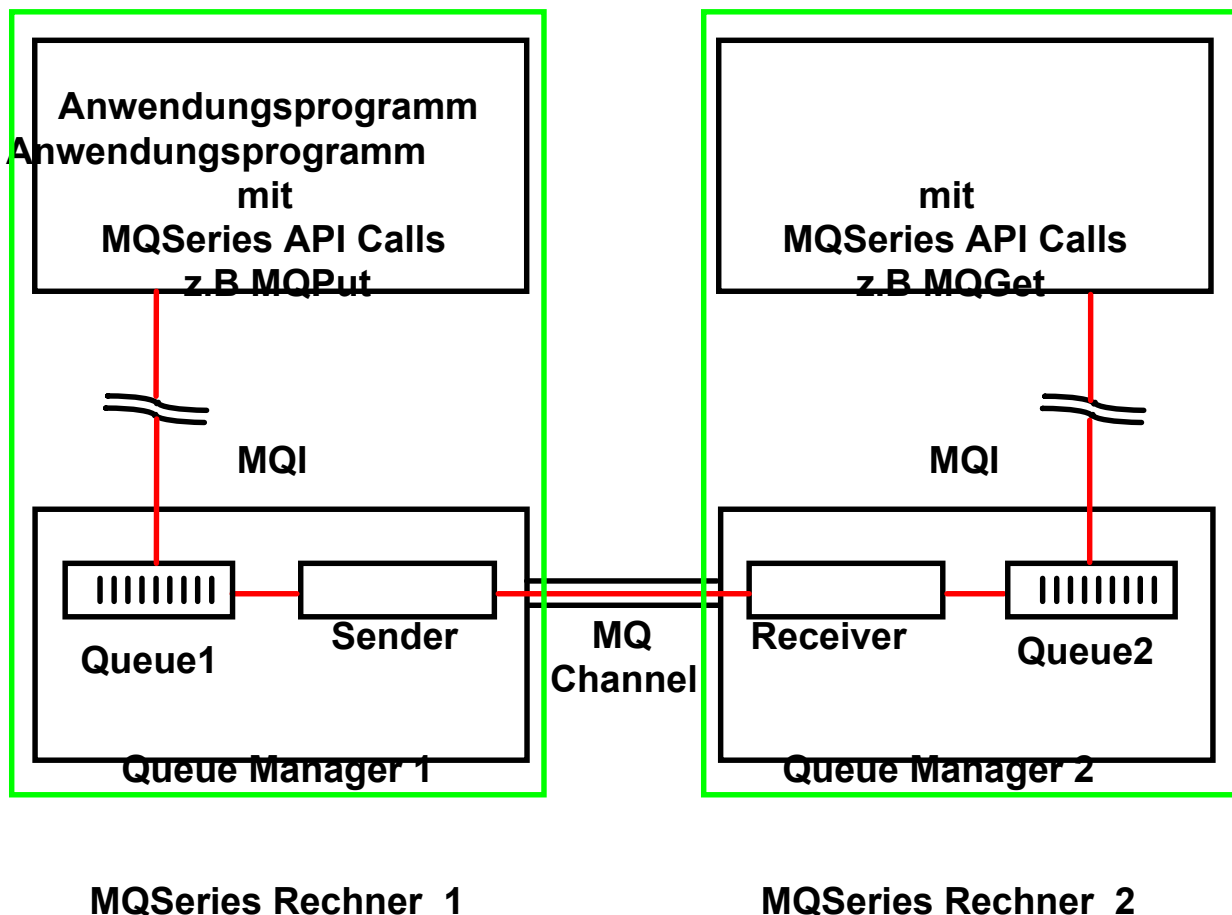
Übertragung von Datenbank Reports, Transaktionen, Audio, Video.



Schritte 1 - 3 : Wenn erfolgreich, Antwort senden

Step 4: Wenn S/390 nach DB2 versagt: roll back nach Queue B, dann recover

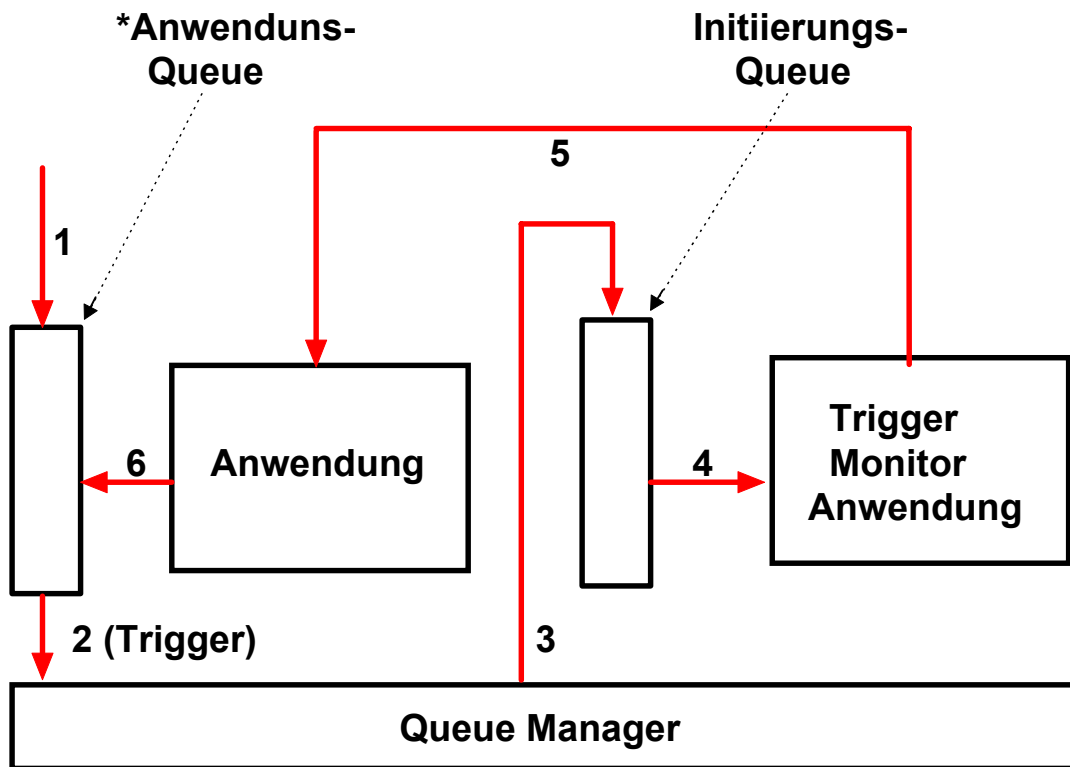
IBM MQSeries ist Marktführer. Für praktisch alle Betriebssysteme verfügbar: AIX, DG/UX, HP-UX, Windows, OS/390, OS/400, Psion Pervasive SOD, Sequent, Sinix, Solaris, Tandem, TPF, TrueUnix, VMS/VAX.



MQSeries Arbeitsweise

Das Anwendungsprogramm auf Rechner 1 stellt mit Hilfe eines *Message Queue Interface (MQI)* API Aufrufs (z.B. MQPut) eine Nachricht in Queue 1 . Die als *Queue Manager (QM)* bezeichnete MQSeries Laufzeitumgebung (Runtime) bewirkt unter ACID Bedingungen den Transport der Nachricht nach Queue 2 auf Rechner 2. Der Transport erfolgt über einen MQ-Channel. Hierfür unterhält Queue Manager 1 einen *Channel Sender* und Queue Manager 2 einen *Channel Receiver*. Queues, Queue Manager, Channel Sender/Receiver und Channels repräsentieren Schicht 5 Middleware. Der Channel kann z.B. mit Hilfe von Sockets in Schicht 4 das TCP/IP Protokoll verwenden.

Das Anwendungsprogramm auf Rechner 2 kann nun asynchron mit Hilfe eines MQI Aufrufs MQGet die Nachricht aus Queue 2 auslesen. Evtl. unterhält Queue Manager 2 eine Trigger Funktion, welche die Anwendung beim Eintreffen einer Nachricht informiert.

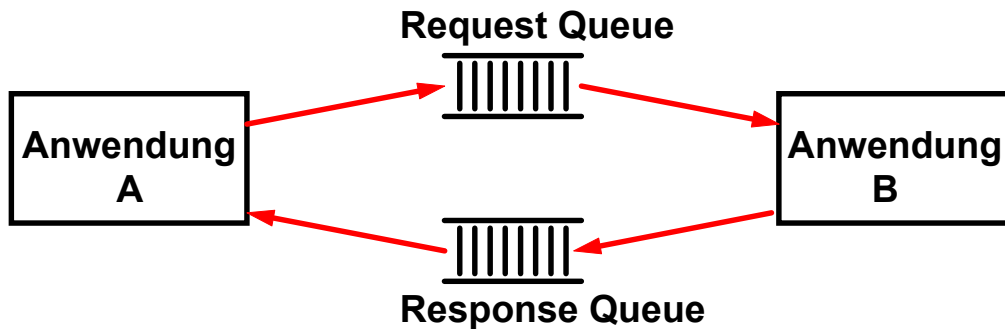


Trigger Nachrichten

Als Triggering wird der Mechanismus bezeichnet, mit Hilfe dessen eine Anwendung bei Bedarf gestartet werden kann. Der Queue Manager erkennt die Ankunft neuer Nachrichten (für die Triggering enabled ist). Er benachrichtigt eine spezielle Anwendung, den Trigger Monitor. Der Trigger Monitor startet die Anwendung, die für die Nachricht zuständig ist.

Da mehrere Nachrichten gleichzeitig eintreffen können, werden die Trigger in einer separaten Queue, der Initiierungsqueue, zwischengespeichert. Die Trigger Nachricht enthält die „Prozess Definition“ - Information, mit deren Hilfe der Trigger Monitor die gewünschte Anwendung starten kann.

MQSeries Betrieb



Nachrichtenkopf enthält: „Reply to“

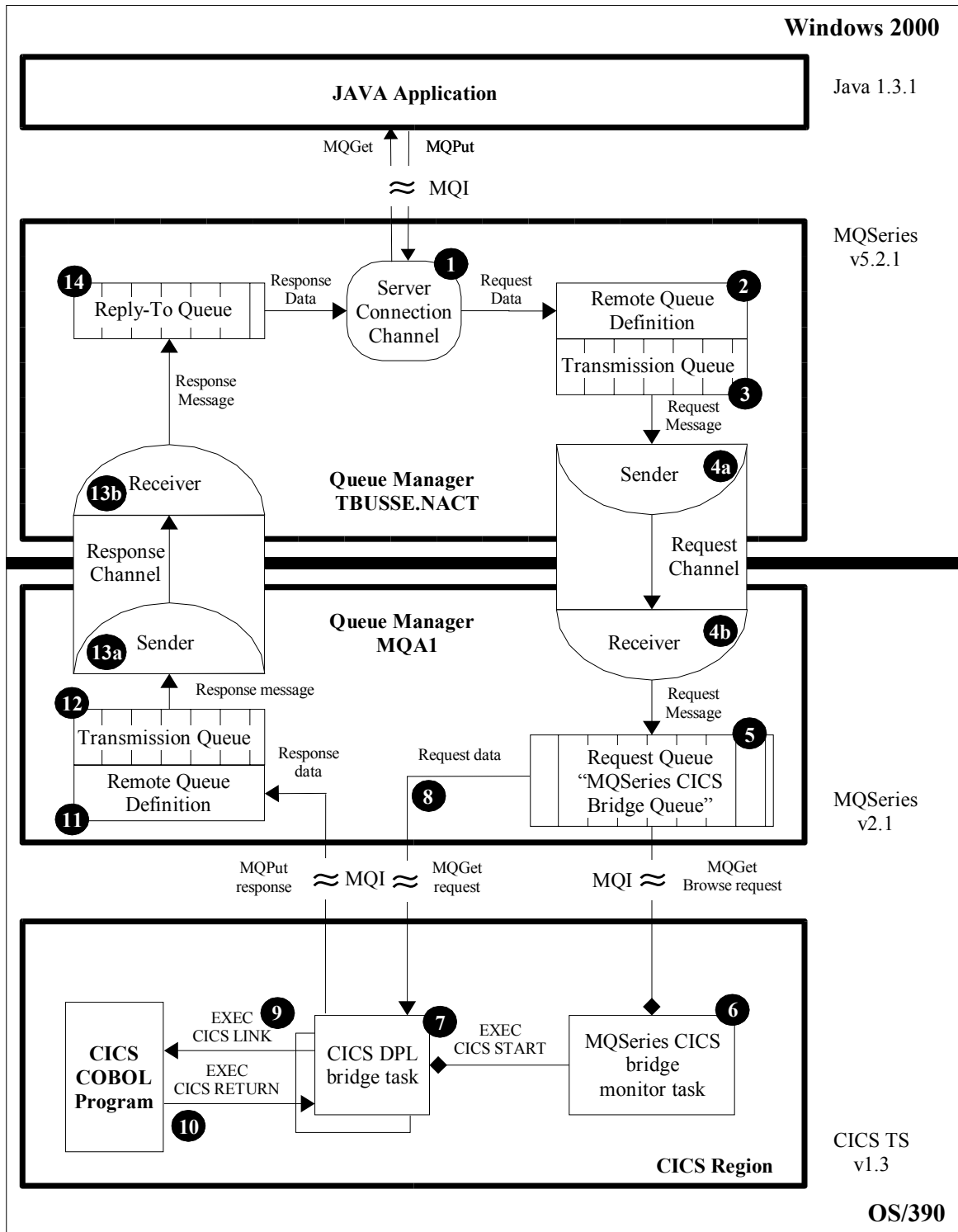
Asynchroner Betrieb: Request/reply Modus kann synchronen Betrieb simulieren. Wird häufig so benutzt.

MQ-CICS Bridge: Receiving queue lädt Nachricht in die CICS Input Queue. CICS behandelt es wie eine 3270 Nachricht, gibt eine 3270 Nachricht zurück. Response Queue B übersetzt Nachricht in das MQSeries Format.

Keine Daten Konvertierung, nur binäre Daten werden übertragen.

Firmen wenden 40 % ihres eigenen Budgets für Software Entwicklung für Integrationsaufgaben.

Wird Software gekauft, kostet die Integration das 9fache des Kaufpreises.



MQSeries Zugriff auf CICS

Java Präsentationslogik greift mit Hilfe von MQSeries auf CICS COMMAREA zu

MQSeries API

MQI

MQCONN stellt eine Verbindung mit einem Queue-Manager mit Hilfe von Standard-Links her.

MQBEGIN startet eine Arbeitseinheit, die durch den Queue-Manager koordiniert wird und externe XA-kompatible Ressource-Manager enthalten kann. Dieses API ist mit MQSeries Version 5 eingeführt worden. Es wird für die Koordinierung der Transaktionen , die Queues (MQPUT und MQGET unter Syncpoint-Bedingung) und Datenbank-Updates (SQL-Kommandos) verwenden.

MQGET

MQPUT

MQPUT1 öffnet eine Queue, legt eine Message darauf ab und schließt die Queue wieder. Dieser API-Call stellt eine Kombination von MQOPEN, MQPUT und MQCLOSE dar.

MQINQ fordert Informationen über den Queue-Manager oder über eines seiner Objekte an, wie z.B. die Anzahl der Nachrichten in einer Queue.

MQSET verändert einige Attribute eines Objekts.

MQCMIT gibt an, dass ein Syncpoint erreicht worden ist.

MQBACK teilt dem Queue-Manager alle zurück gekommenen PUT's- und GET's-Nachrichten seit dem letzten Syncpoint mit.

MQDISC schließt die Übergabe einer Arbeitseinheit ein. Das Beenden eines Programms ohne Unterbrechung der Verbindung zum Queue-Manager verursacht ein "Rollback" (MQBACK).

```

MQHCONN HConn; // Connection handle
MQHOBJ HObj1; // Object handle for queue 1
MQHOBJ HObj2; // Object handle for queue 2
MQLONG CompCode, Reason; // Return codes
MQLONG options;
MQOD od1 = {MQOD_DEFAULT}; // Object descriptor for queue 1
MQOD od2 = {MQOD_DEFAULT}; // Object descriptor for queue 2
MQMD md = {MQMD_DEFAULT}; // Message descriptor
MQPMO pmo = {MQPMO_DEFAULT}; // Put message options
MQGMO gmo = {MQGMO_DEFAULT}; // Get message options
:
// 1 Connect application to a queue manager.
strcpy (QName, "MQMGR");
MQCONN (QName, &HConn, &CompCode, &Reason);

// 2 Open a queue for output
strcpy (od1.ObjectName, "QUEUE1");
MQOPEN (HConn, &od1, MQOO_OUTPUT, &HObj1, &CompCode, &Reason);

// 3 Put a message on the queue
MQPUT (HConn, HObj1, &md, &pmo, 100, &buffer, &CompCode, &Reason);

// 4 Close the output queue
MQCLOSE (HConn, &HObj1, MQCO_NONE, &CompCode, &Reason);

// 5 Open input queue
options = MQOO_INPUT_AS_Q_DEF;
strcpy (od2.ObjectName, "QUEUE2");
MQOPEN (HConn, &od2, options, &HObj2, &CompCode, &Reason);

// 6 Get message
gmo.Options = MQGMO_NO_WAIT;
buflen = sizeof(buffer) - 1;
memcpy (md.MsgId, MQMI_NONE, sizeof(md.MsgId));
memset (md.CorrelId, 0x00, sizeof(MQBYTE24));
MQGET (HConn, HObj2, &md, &gmo, buflen, buffer, 100, &CompCode, &Reason);

// 7 Close the input queue
options = 0;
MQCLOSE (HConn, &HObj2, options, &CompCode, &Reason);

// 8 Disconnect from queue manager
MQDISC (HConn, &CompCode, &Reason);

```

Figure 21. A Code Fragment

MQ Series Code Fragment