Von Transaktionsmonitoren zu Applikationsservern

Vorlesung 7: Software-Engineering für große Informationssysteme

TU-Wien, Sommersemester 2002

Wolfgang Keller

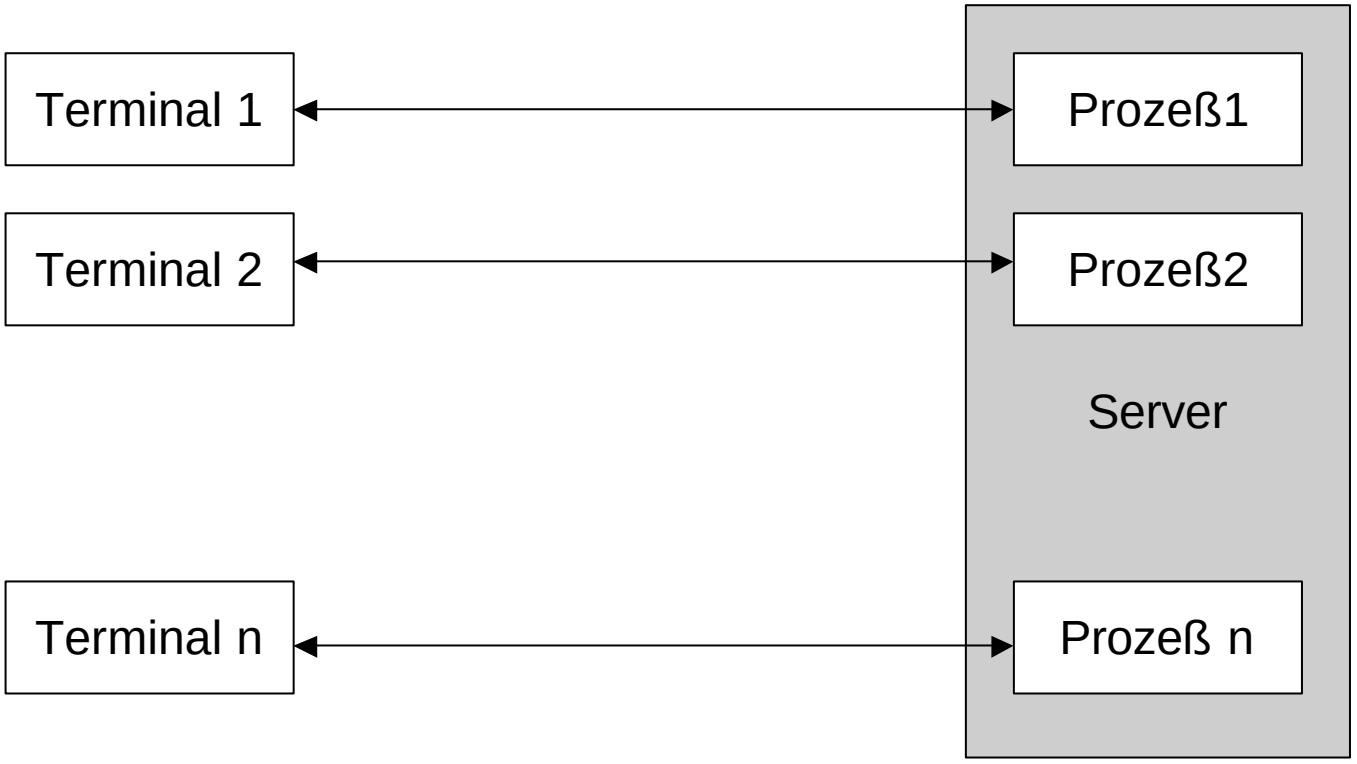
Überblick (1)

- Die Probleme
  - Bedienung vieler Dialogbenutzer
  - MIPS-Entwicklung: Die Hosts der 70er Jahre ... Die PCs heute
  - Gemeinsamer Zugriff auf Ressourcen – Transaktionen
- Prozeß- und Dialogmodelle
  - Conversational vs. Transactional Programming
  - Und die dahinterliegende Prozeßorganisation
- Transaktionen und verteilte Transaktionen: 2PC
  - ACID Eigenschaften
  - X/Open Transaktionsmanager

Überblick (2)

- CICS als Beispiel für einen Transaktionsmonitor
  - Programmiermodell
  - Dienste
  - Zum Vergleich noch IMS
  - Services von IMS und CICS
- Was leistet im Vergleich dazu ein J2EE Applikationscontainer?

Bedienung vieler Dialogbenutzer
Ein Prozess pro Benutzer



Ein Prozess pro Benutzer das Modell hat Grenzen ...

- Das Modell gibt es in verschiedenen Ausprägungen
- Zum Beispiel auf der AS/400 – mit bis zu ca. 400 Benutzern
- Oder aber auch als Windows Terminalserver mit ca. 20 Benutzern für ca. 4 Prozessoren

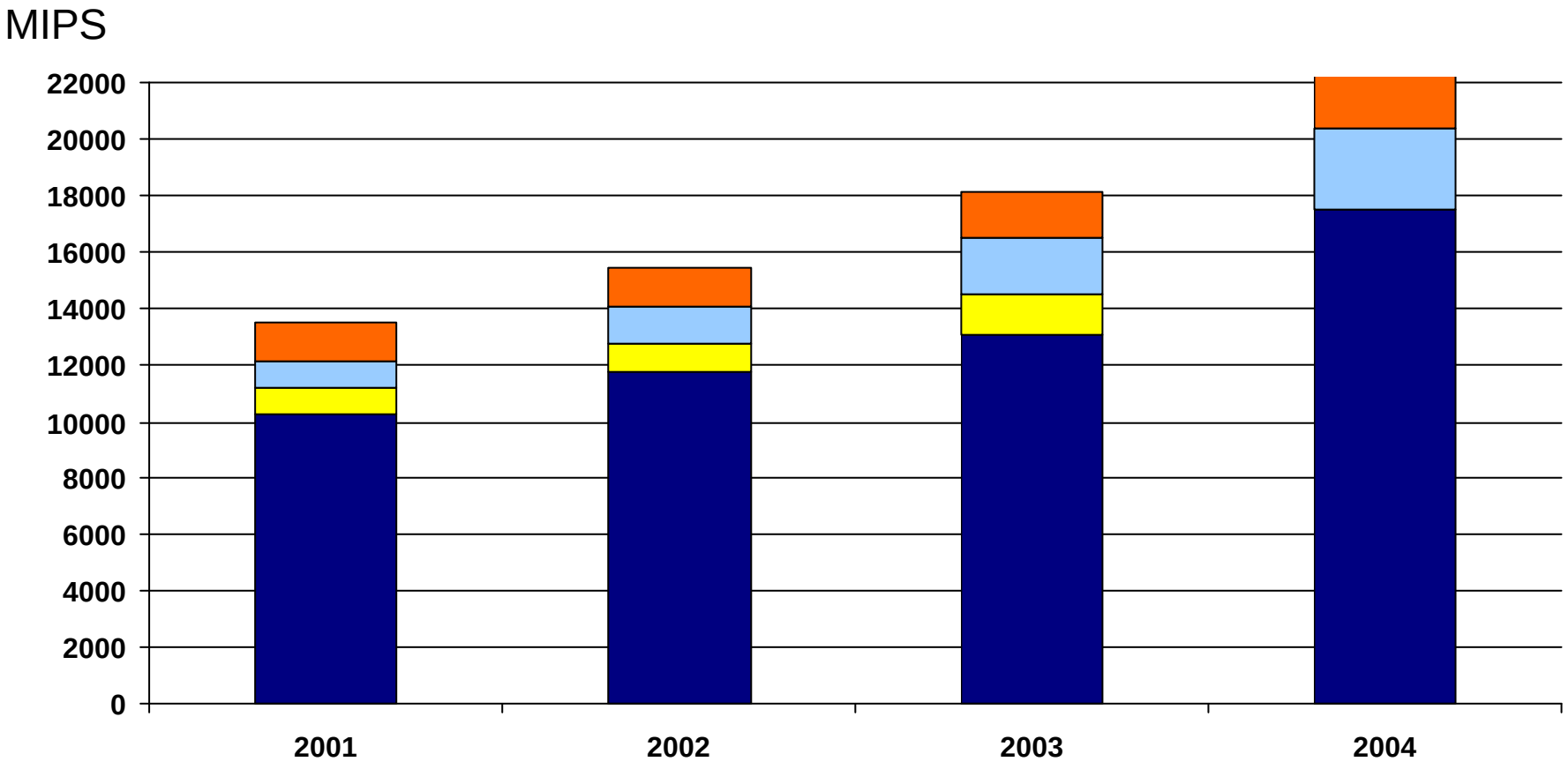
- Problem: Prozeßwechsel der Prozessoren sind aufwändig

Leistungsdaten eines Mainframes der 70er Jahre ...

- Hardware und Leistung „Großrechner“ der 70er
  - 256 KB Hauptspeicher
  - 0,5 Mips
  - Bediente hunderte Benutzer ..
- Im Vergleich dazu ein Pentium PC (200 MHz)
  - Ca. 200 Mips
  - 256 KB Prozessorcache
  - 128 MB++ Hauptspeicher

- Aber Vorsicht – MIPS allein sagen nichts über die Dialog- und IO-Leistung

Nur als Dimension: Projezierte MIPS Entwicklung Großanwender

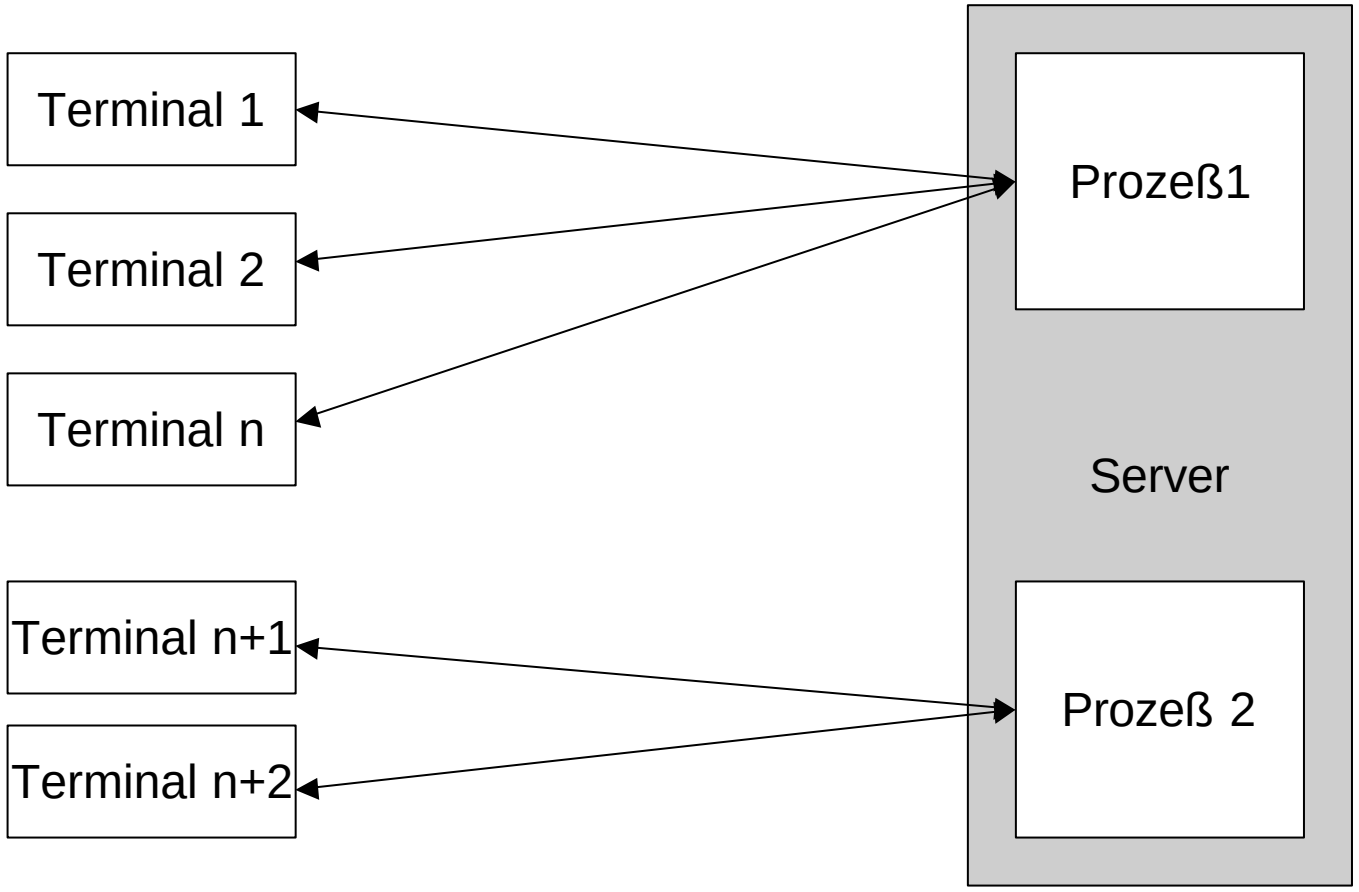


Was war anders und Folgerungen

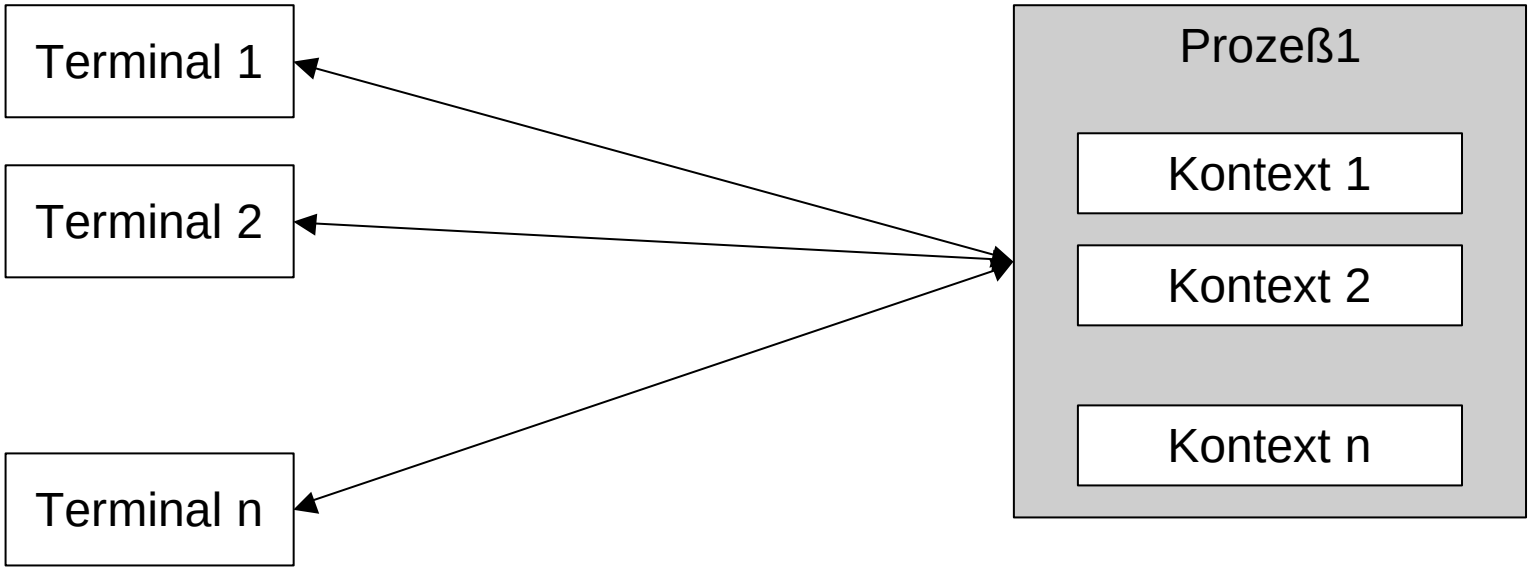
- Anwendungen waren weniger CPU intensiv
- Datenbanken weitgehend nicht existent – VSAM Files und ähnliches
- Weniger Ressourcen pro Benutzer
- Maschinen waren auf Batch- und nicht Dialogbetrieb ausgelegt

- Man brauchte also ein Konzept für leichtgewichtige Prozesse
- Threads waren damals noch nicht bekannt

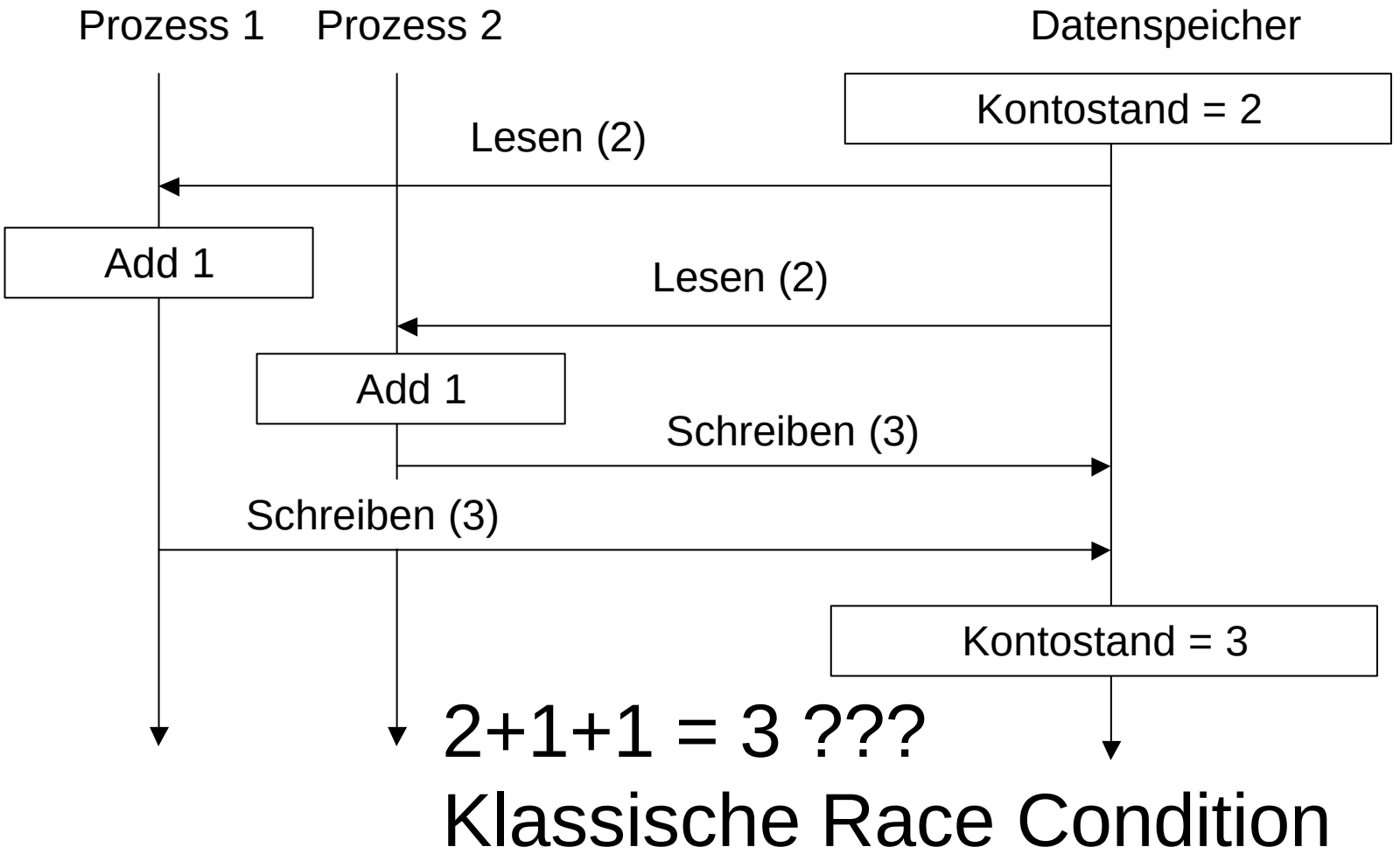
Bedienung vieler Dialogbenutzer
Ein Prozess pro Benutzer



Pro Terminal muß es einen Kontext im Prozeß geben ...

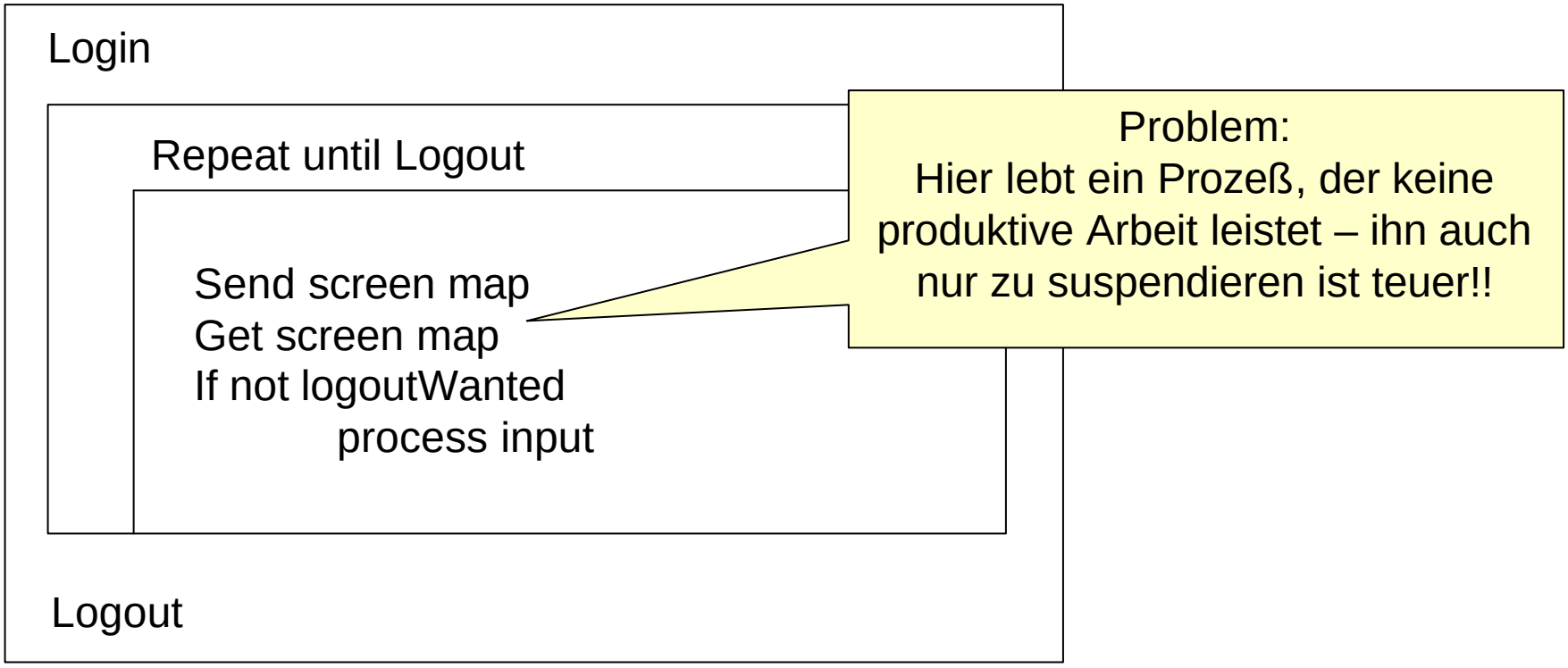


Problem 2: Gemeinsamer Zugriff auf Datenbanken



Prozeß- und Dialogmodelle

Prozeß- und Dialogmodelle Conversational Programming



Zu einem Prozeßkontext bei symetrischem Multiprocessing gehören u.a. ...

- Name des Prozesses
- Bezeichnung der gerade bearbeiteten Aufträge
- Stellung in der Prozeßhierarchie
- Rechnerkernzustand (Register, ...)
- Alarmzustand
- Arbeitszustand
- Rechte
- Betriebsmittelkonten
- Beschreibung zugeordneter Objekte und Betriebsmittel (IO, Controller, Screens, ...)
- Beschreibung des Programmadreßraumes

Quelle: Siegert, Betriebssysteme, Eine Einführung, Oldenbourg Verlag 1988

Und bei einem Prozeßwechsel passiert folgendes ...

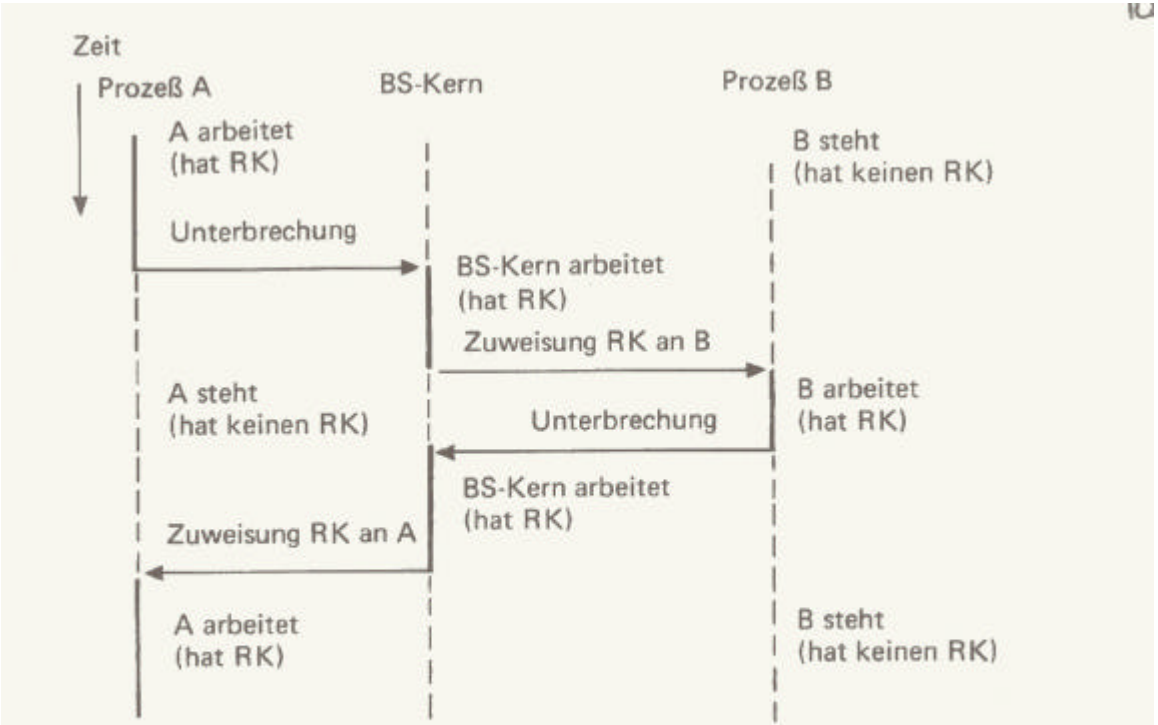
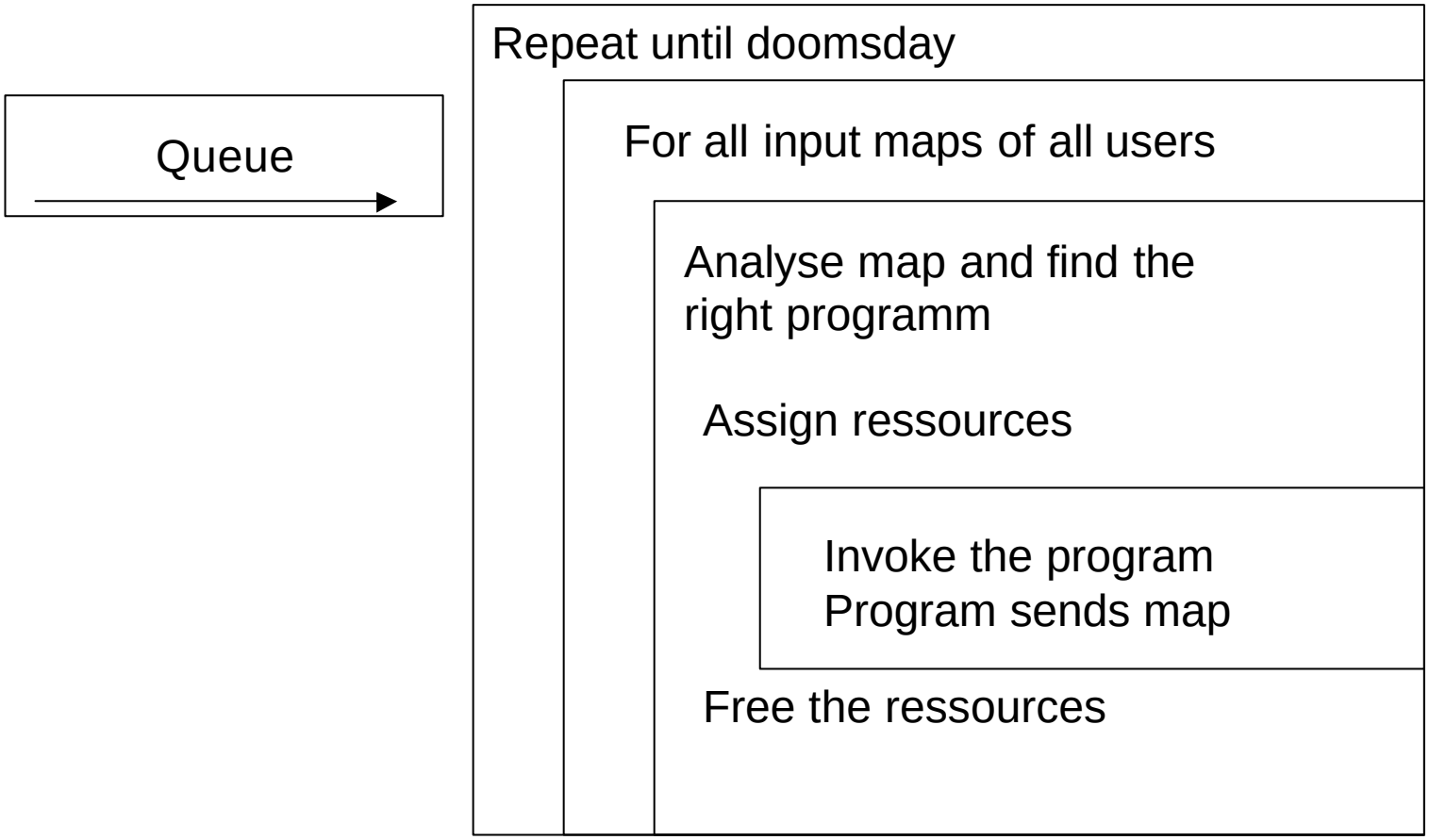


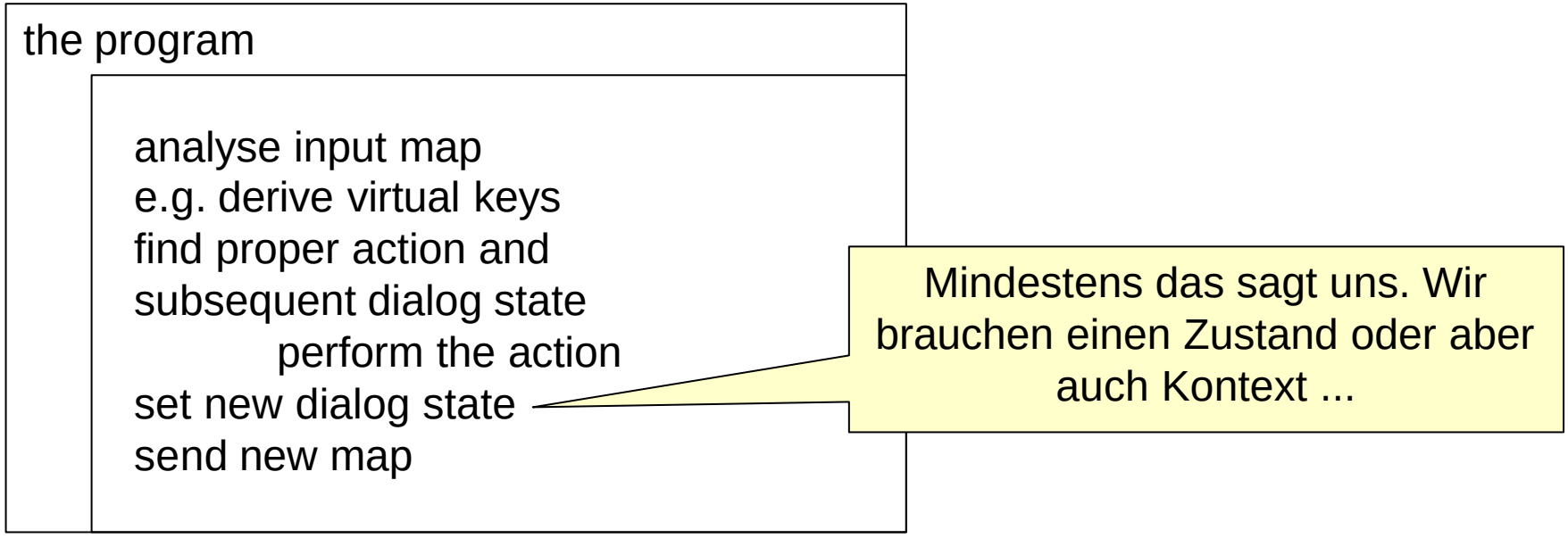
Abb. 3-2: Zeitliche Verschränkung der Prozeßbearbeitung in einem Einprozessorsystem

Quelle: Siegert, Betriebssysteme, Eine Einführung, Oldenbourg Verlag 1988

Transactional Programming



Transactional Programming



Vorsicht also: Der Begriff Transaktion bedeutet ..

- Transaktion im Sinne von „transactional programming“
  - Man programmiert „eine Transaktion“
- Transaktion im Sinne der Datenbankprogrammierung
  - Ich muß eine Transaktion gegen eine Datenbank fahren

Transaktionen und verteilte Transaktionen: 2PC

- ACID Eigenschaften
- 2PC

ACID Eigenschaften (1)

- Atomicity (Atomarität)
  eine Transaktion wird entweder komplett ausgeführt (commit) oder gar nicht – sie erscheint dann so, als wäre sie nie gestartet worden (rollback).

- Consistency (Konsistenz)
  eine Transaktion führt eine Datenbank von einem konsistenten Zustand in den nächsten über
  (Hinweis: konsistent = technisch konsistent)

ACID Eigenschaften(2)

- Isolation:
  Transaktionen, die zeitlich parallel ausgeführt werden erscheinen so, als wären sie isoliert voneinander (hintereinander) ausgeführt worden
  - -> siehe Serialisierbarkeitstheorie, Sperrprotokolle
- Durability:
  Nach dem Commit werden die Effekte dauerhaft, also persistent gespeichert

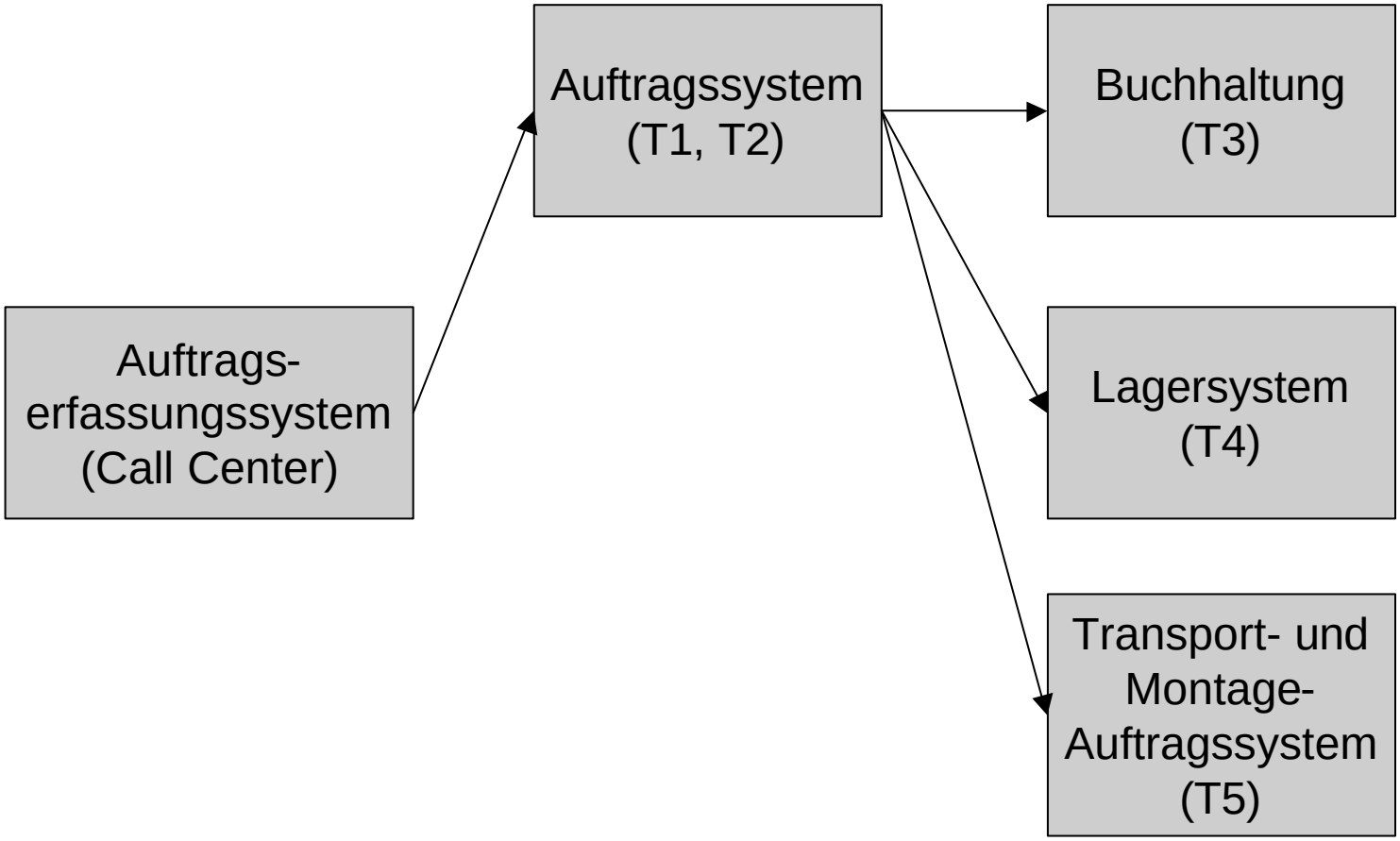
Beispiel für verteilte TAs oder TAs mit >1 Ressource ..

- Wenn ein Callcenter-Mitarbeiter von einem Kunden einen Auftrag für eine Waschmaschine bekommen hat, muss Folgendes passieren:
  - (OP1) Auftrag, Rechnung: Es muss ein Auftrag und eine Rechnung erstellt werden. Die Rechnung muss in der Buchhaltung verbucht werden.
  - (OP2) Auslagerung, Lieferschein: Die Verfügbarkeit der Waschmaschine im Lager muss geprüft werden und falls vorhanden, muss die Waschmaschine ausgelagert werden. Der Einfachheit halber nehmen wir an, dass die Waschmaschinen immer lagernd verfügbar sind. Wir können deshalb immer einen Lieferschein erstellen. Die Auslagerung und der Lieferschein benötigen als Daten:
    - die Rechnungsnummer und die Auftragsnummer,
    - die Adresse des Kunden.
  - (OP3) Montageauftrag: Es muss ein Montageauftrag erstellt werden. Der Auftrag weist einen Mitarbeiter an, die Waschmaschine aus dem Lager abzuholen, zum Kunden zu bringen und dort zu montieren.

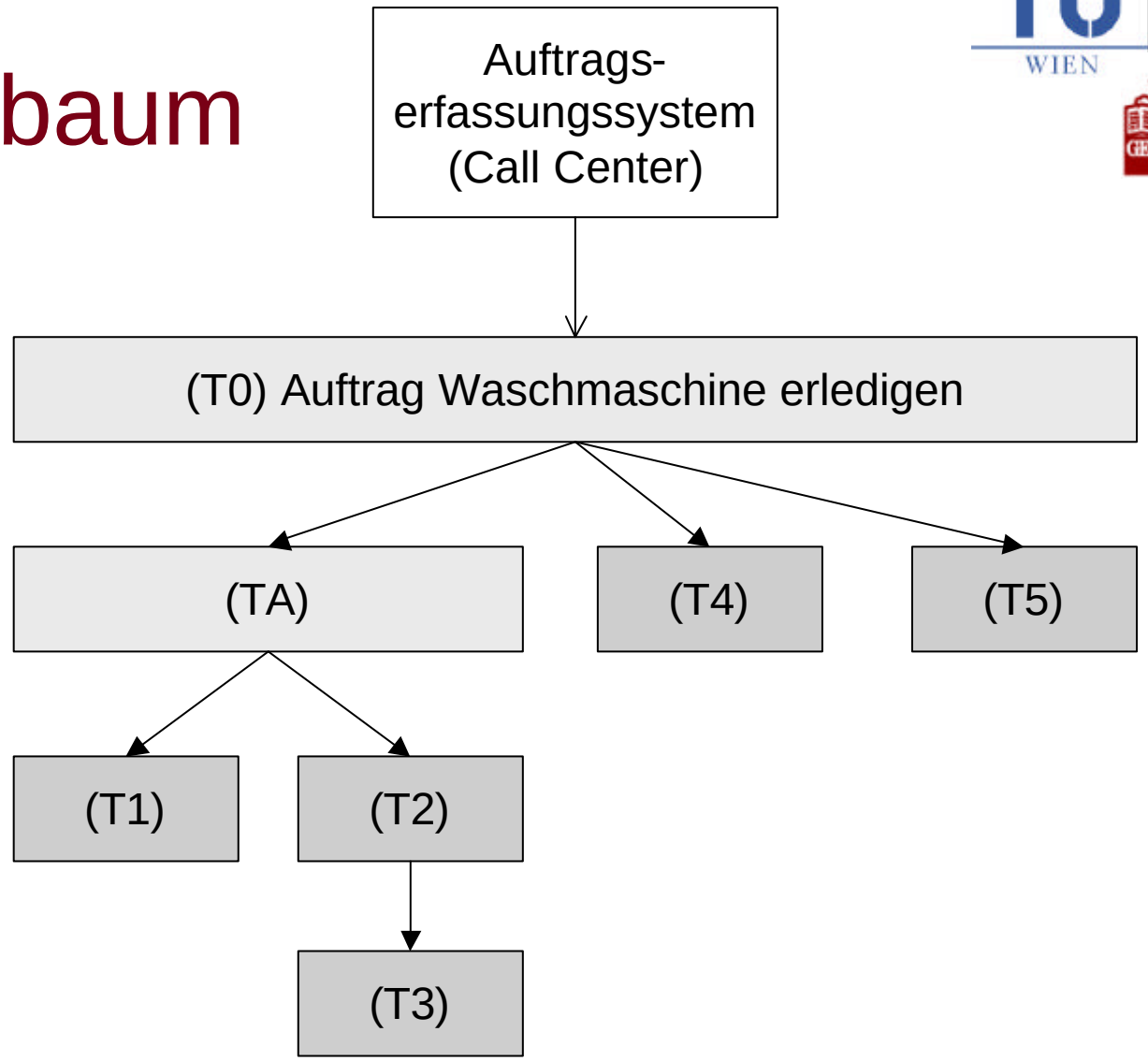
Beispiel für verteilte TAs oder TAs mit >1 Ressource ..

- In einem Unternehmen können an einem solchen Vorgang zum Beispiel vier DV-Systeme auf verschiedenen Plattformen beteiligt sein:
  - ein Auftragserfassungssystem,
  - ein Buchhaltungssystem,
  - ein Lager- und Bestandsverwaltungssystem,
  - ein System für Transport- und Montageaufträge.

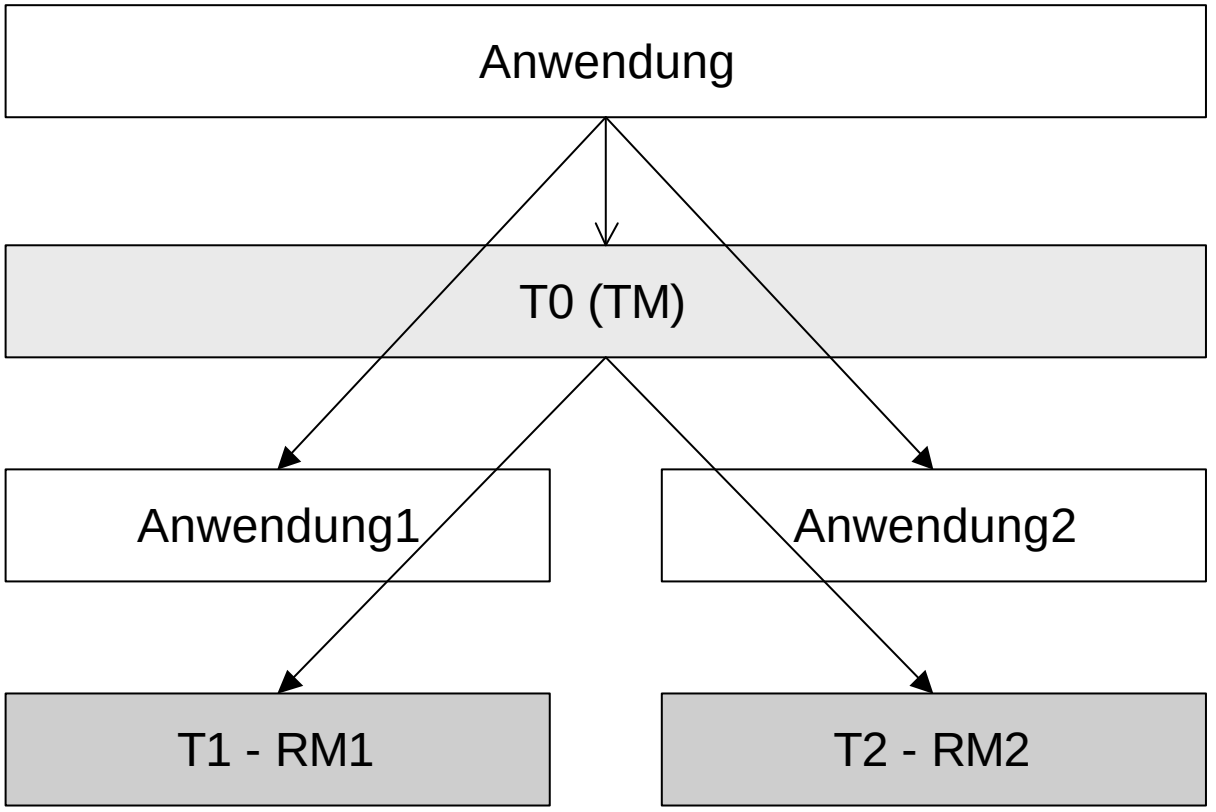
Verteilte Transaktion (Beispiel)



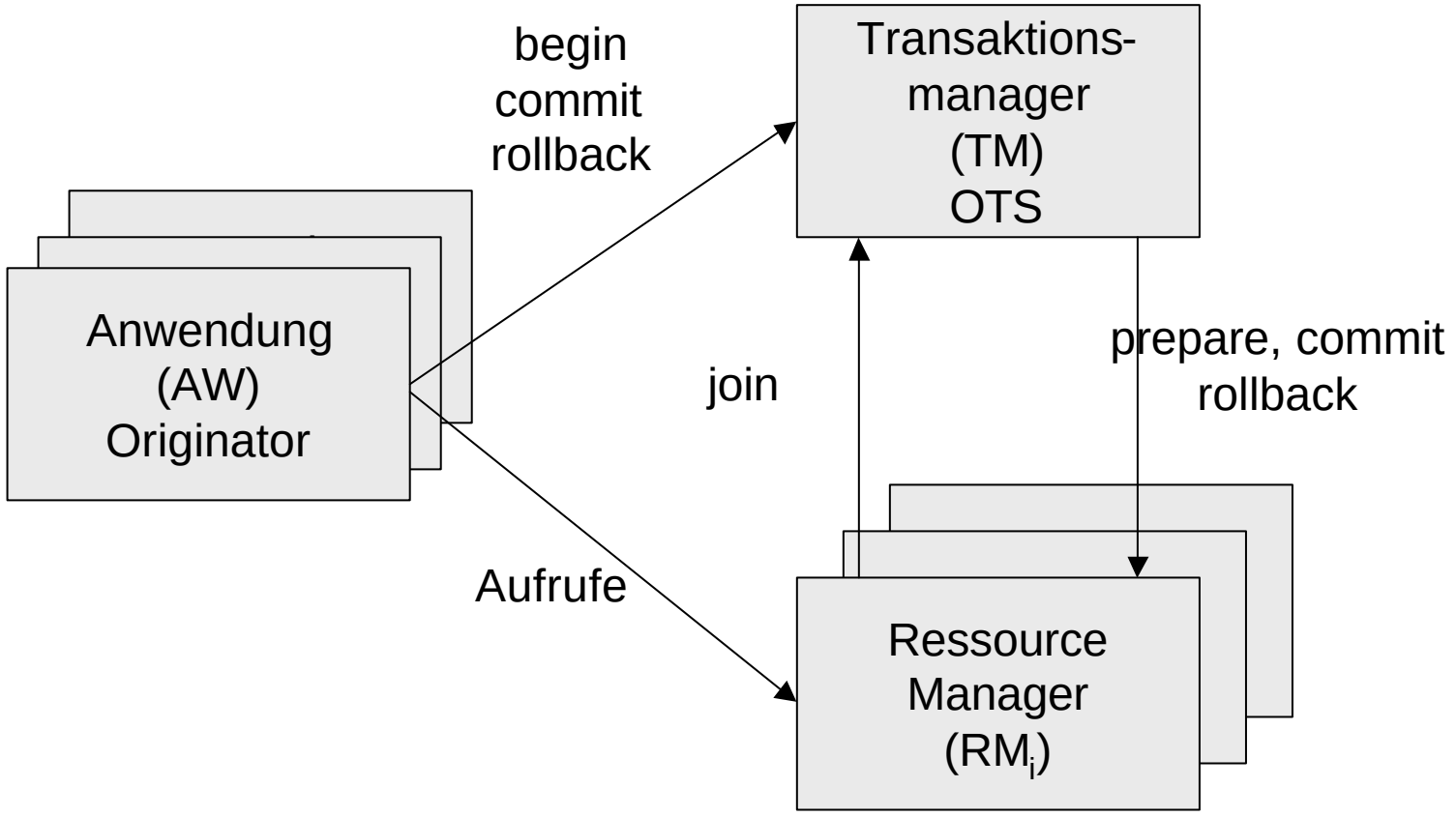
Transaktionsbaum



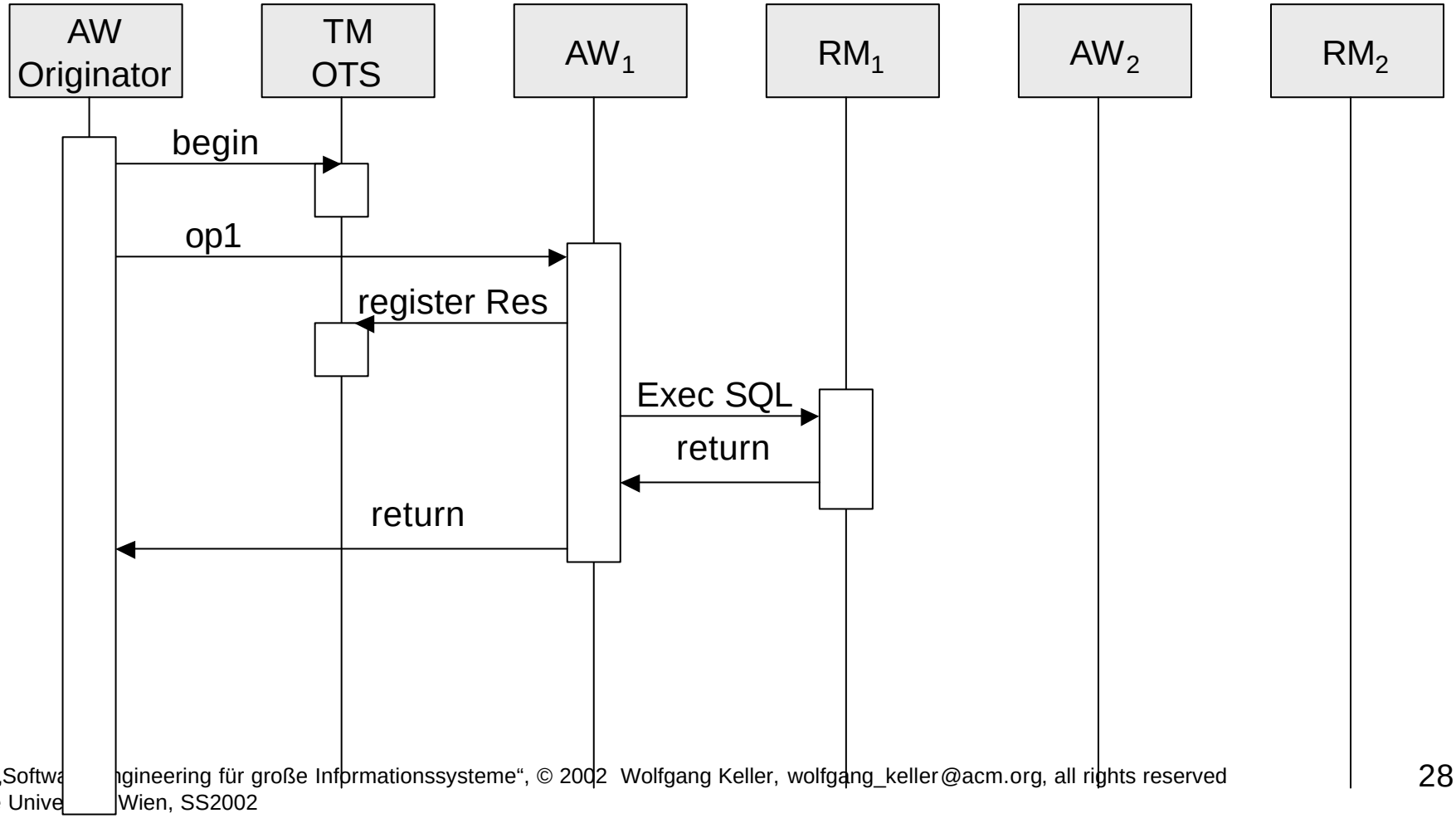
Transaktionsbaum
Transaktions- und Ressourcemanager



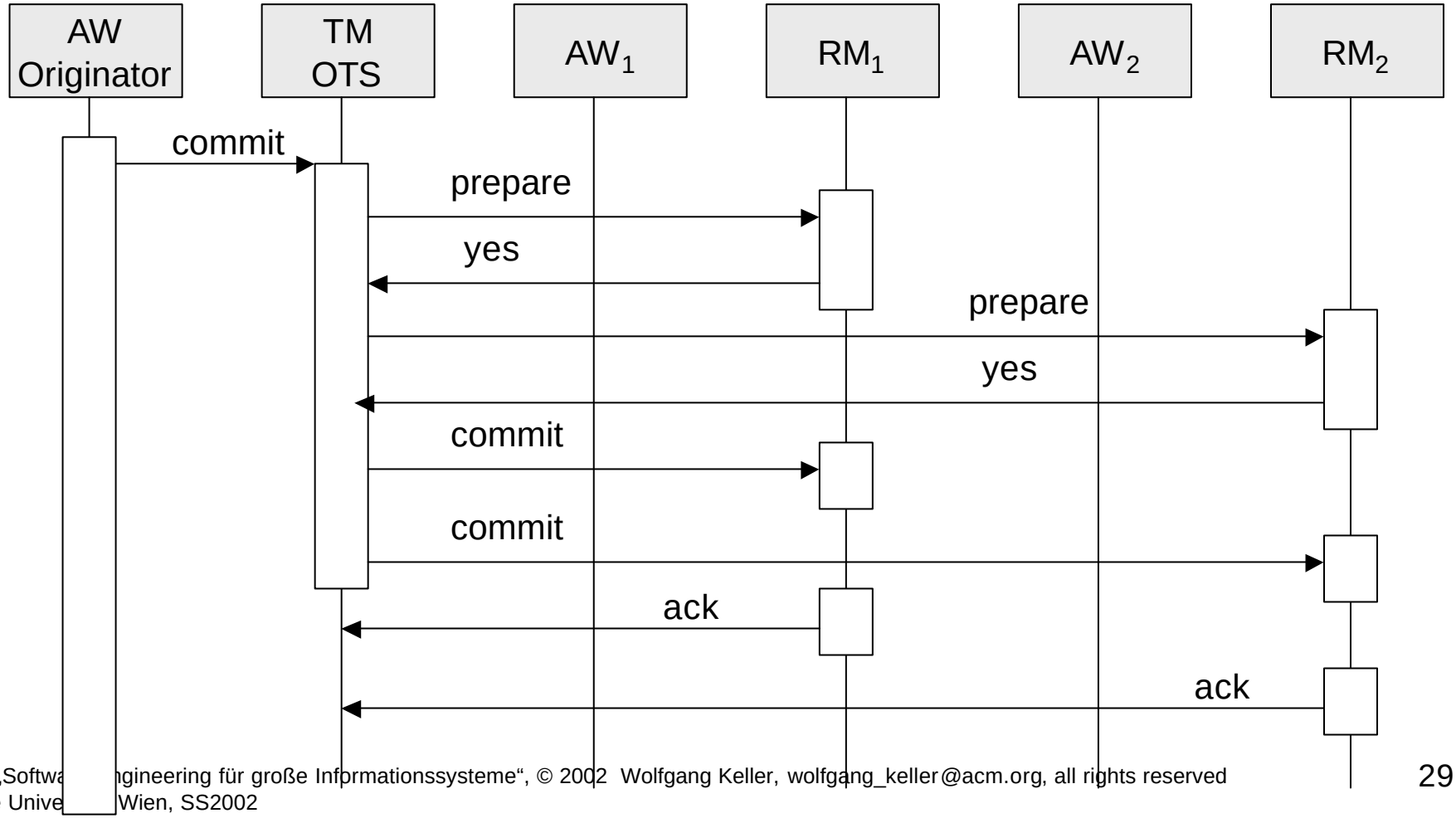
2PC Teilnehmer



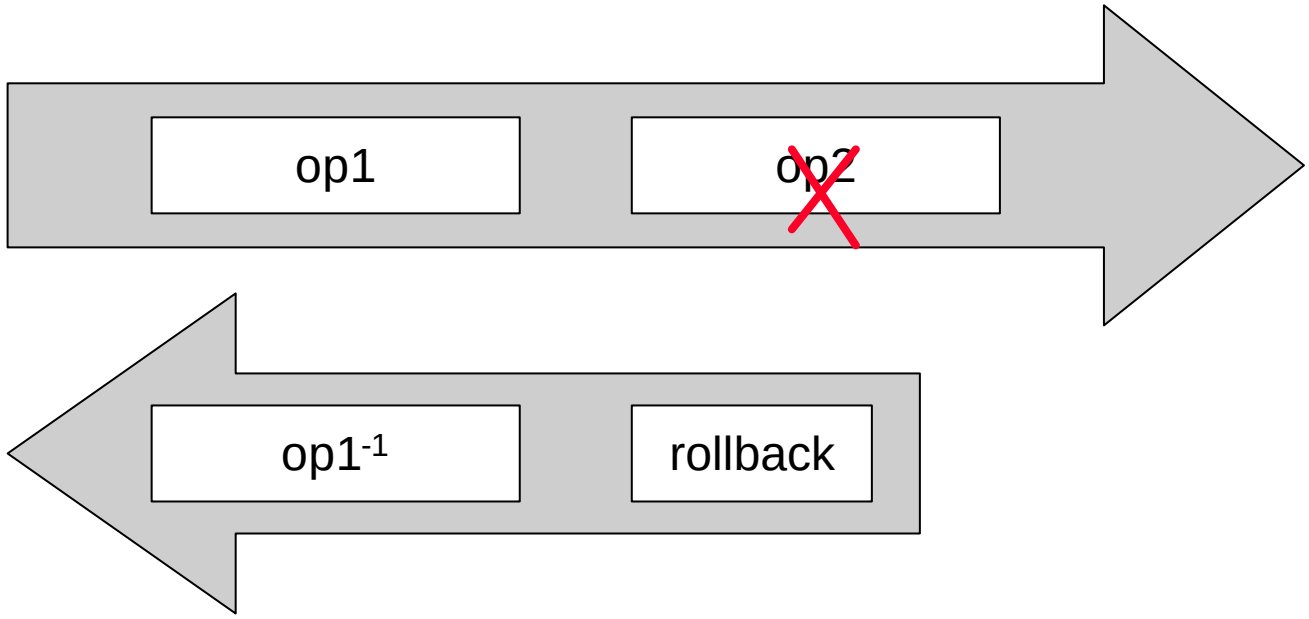
2PC Beispiel



2PC Beispiel



Als anderes Modell gibt es auch noch „Soft Rollback“



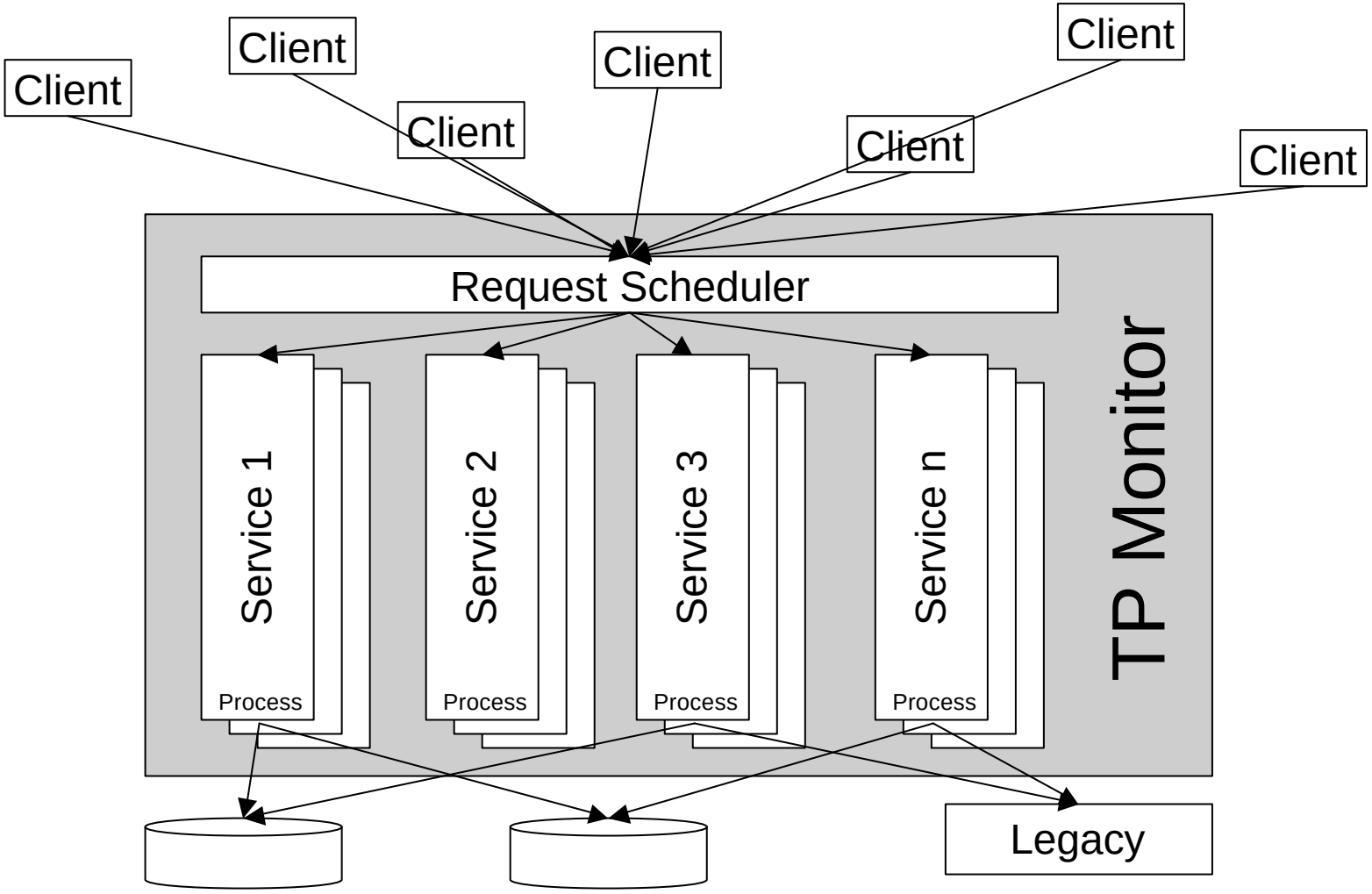
TP-Monitore – ganz allgemein (1)

- Ein „transaction processing monitor“ ist eine Art von Middleware.
- Er nimmt einen Strom von Request entgegen von vielen Clients
- Er bietet „failover“ Management, wenn einer der Server versagt und steuert Transaktionen im Auftrag des Clients
- Er bietet Übersetzung von Kommunkationsprotokollen an

TP-Monitore – ganz allgemein (2)

- Er konsolidiert Requests und Antworten darauf (Responses) ziwschen Clients und vielen heterogenen Servern
- Er bietet darüber hinaus umfangreiche Funktionen zu seiner eigenen Steuerung an (Service Monitoring)

TP Monitor - allgemein

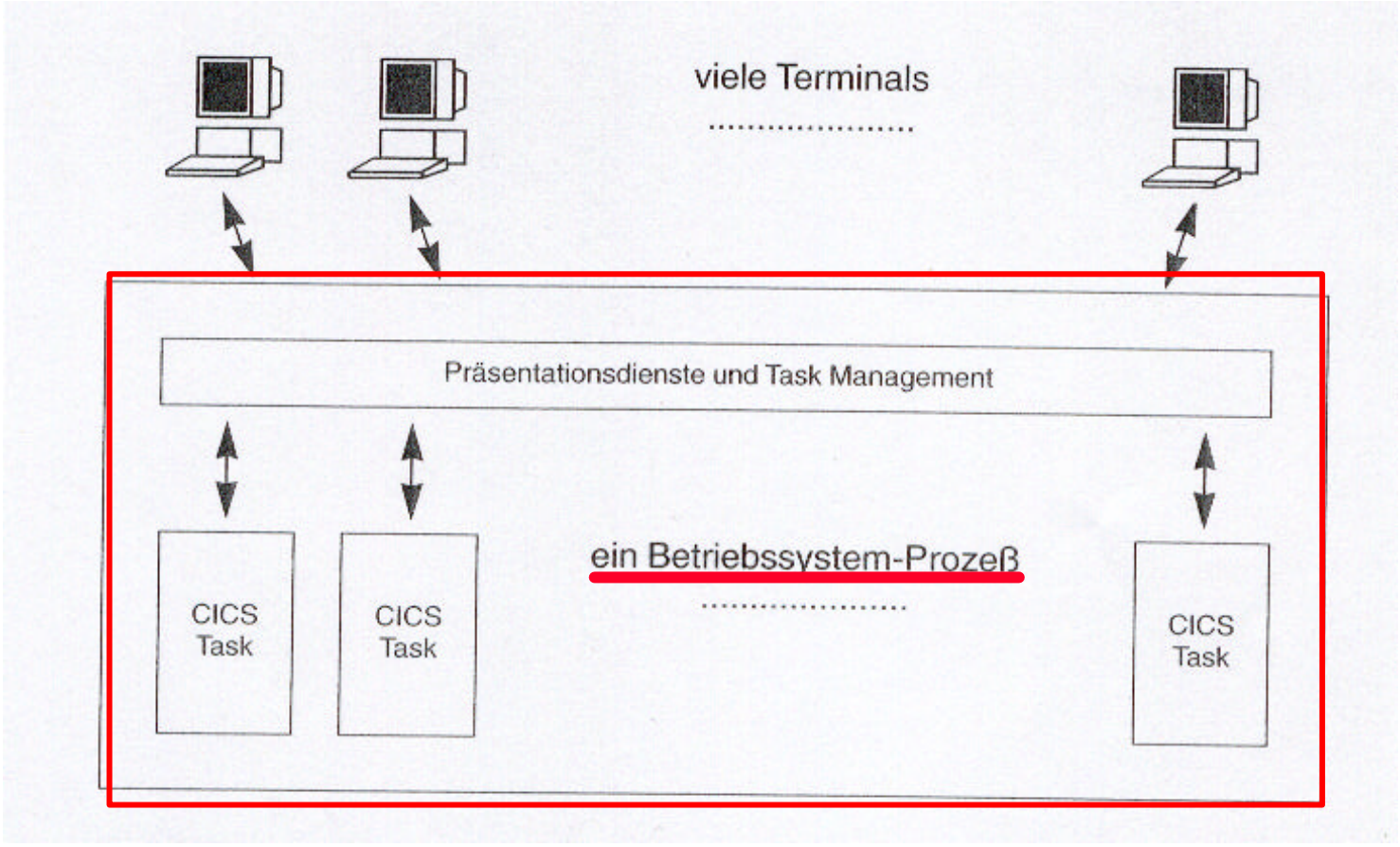


CICS als Beispiel für einen Transaktionsmonitor

CICS heißt

- **C**ustomer **I**nformation **C**ontrol **S**ystem
  - Beinhaltet Unterstützung für „transactional programming“
  - Und für 2PC – also Unterstützung für Transaktionen mit mehr als einer „managed ressource“

CICS Prozessorganisation



Quelle: Brössler et al.; Softwaretechnik, Hanser 2000

Unter CICS Programmierern bekannte Eigenschaften

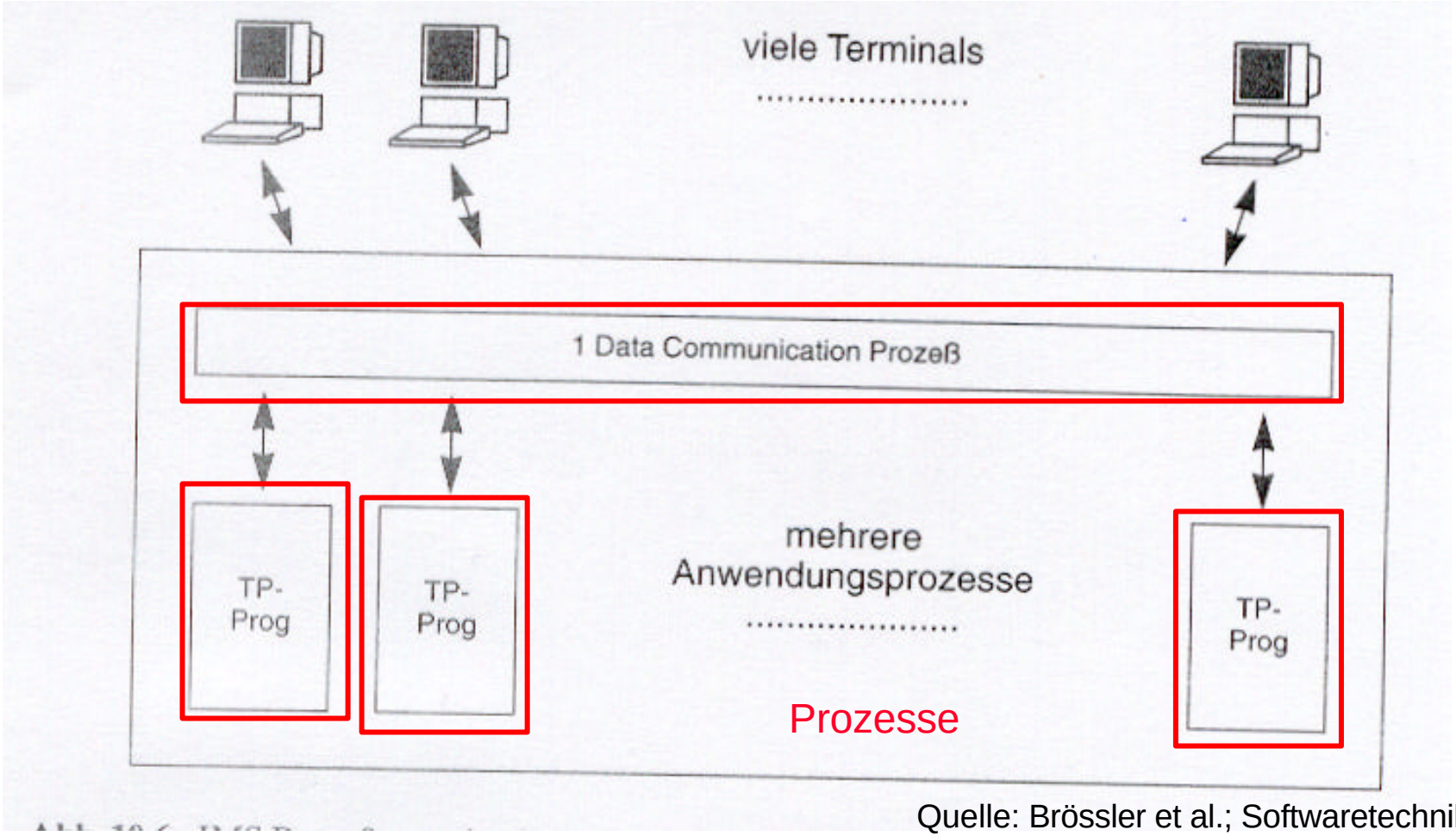
- Alle CICS Tasks eines CICS laufen im selben Prozeß
- Damit sind sie nicht gegeneinander geschützt
  - Man kann durch eine Endlosschleife eine ganze CICS Instanz aufhängen
  - Man kann durch Programmierfehler die Daten anderer Tasks zerstören oder das komplette CICS abschießen

Aufbau eines CICS „Anwendungsprogrammes“

```
CICS-DIALOGPROGRAMM.
    EXEC CICS RECEIVE
        MAP(Maskenname)
        INTO(Masken-Datenstruktur)
    END-EXEC
    PERFORM Zustand-Info-herstellen
    PERFORM Eingabe-Verarbeiten-Ausgabe-Aufbereiten
    PERFORM Zustand-Info-sichern
    EXEC CICS SEND
        MAP(Maskenname)
        FROM(Masken-Datenstruktur)
    END-EXEC
    EXEC CICS RETURN
    END-EXEC.
```

Quelle: Brössler et al.; Softwaretechnik, Hanser 2000

Zum Vergleich: IMS Prozeßmodell



Quelle: Brössler et al.; Softwaretechnik, Hanser 2000

Eigenschaften von IMS

- Jedes „Anwendungsprogramm“ läuft bei IMS in einem oder mehreren Prozessen
- Damit besteht Speicherschutz
  - Ein fehlerhaftes Programm kann das IMS-System nicht abschießen
  - Ein fehlerhaftes Programm kann aber sehr wohl die Daten einer anderen Session des selben Programmes „zerschießen“
  - Aber nicht die anderer „Programme“

Und zum Vergleich die IMS „Programmschleife“

```
IMS-DIALOGPROGRAMM.
  PERFORM VERARBEITUNG UNTIL END-OF-MESSAGES.
  GOBACK.
VERARBEITUNG.
  PERFORM GET-INPUT-MESSAGE
  IF NOT END-OF-MESSAGES
     PERFORM Zustand-Info-herstellen
     PERFORM Eingabe-Verarbeiten-Ausgabe-Aufbereiten
     PERFORM Zustand-Info-sichern
     PERFORM INSERT-OUTPUT-MESSAGE
  END-IF.
GET-INPUT-MESSAGE.
  CALL 'CBLTDLI' USING  DLI-GU IO-PCB
                        INPUT-MESSAGE-IO-AREA
*   keine Nachrichten in der Eingabe-Queue mehr vorhanden ?
*   -> Flag für Programmabbruch setzen.
  IF STATUS-CODE OF IO-PCB = 'QC'
     SET END-OF-MESSAGES TO TRUE
  END-IF.
```

Quelle: Brössler et al.; Softwaretechnik, Hanser 2000

Welche Services bieten IMS und CICS also (1)

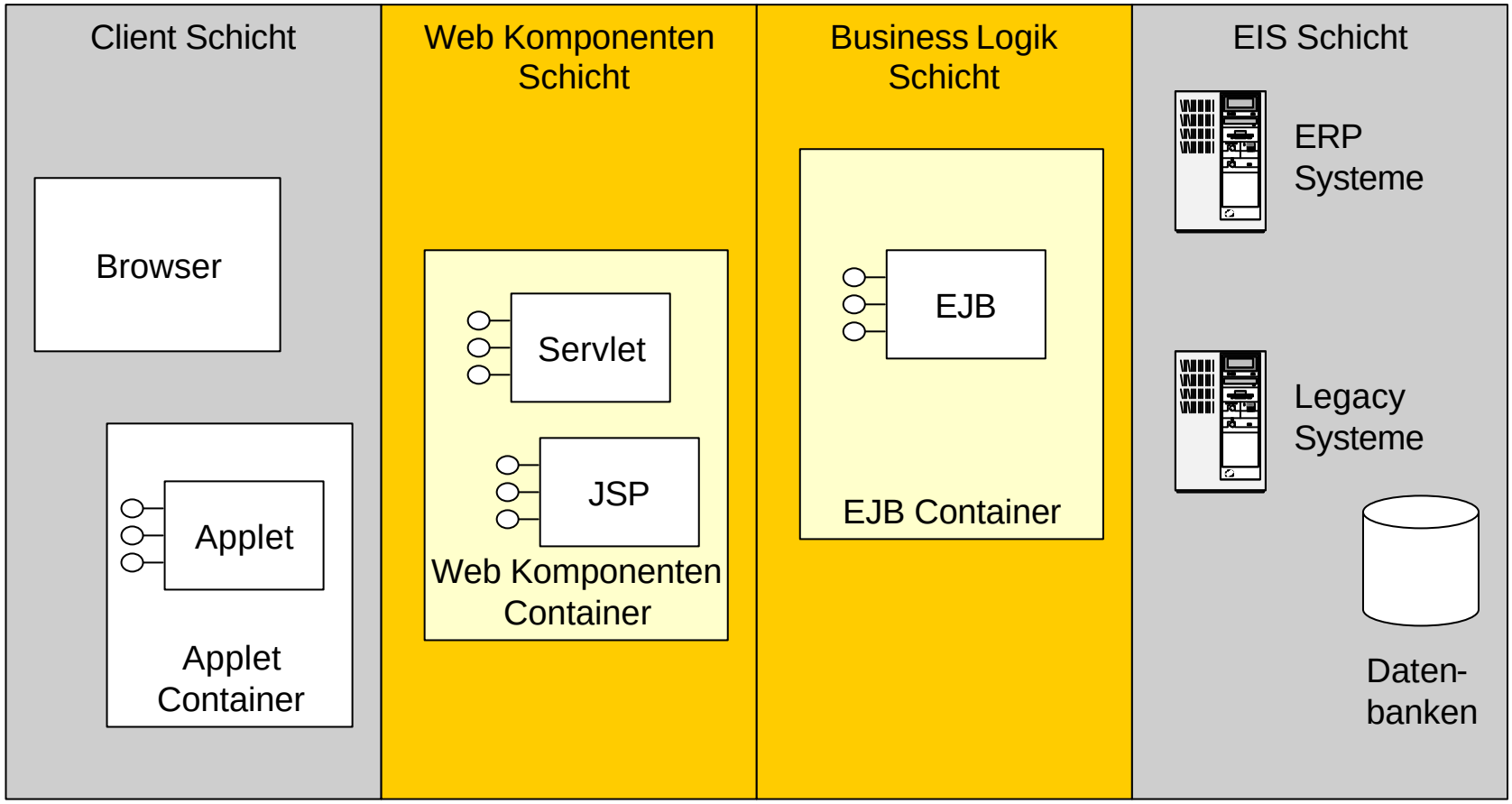
- Unterstützung für performantes „transaktionales“ Programmieren auf Maschinen, die für Batch-Processing ausgelegt sind
  - Zuordnung von Terminalsessions zu Programmen
- Unterstützung für leichtgewichtige Prozesse
- Halten von Kontext über Dialogschritte (CICS: COMMON AREA)
- DLL-artiges, performantes Nachladen von Unterpogrammen

Welche Services bieten IMS und CICS also (2)

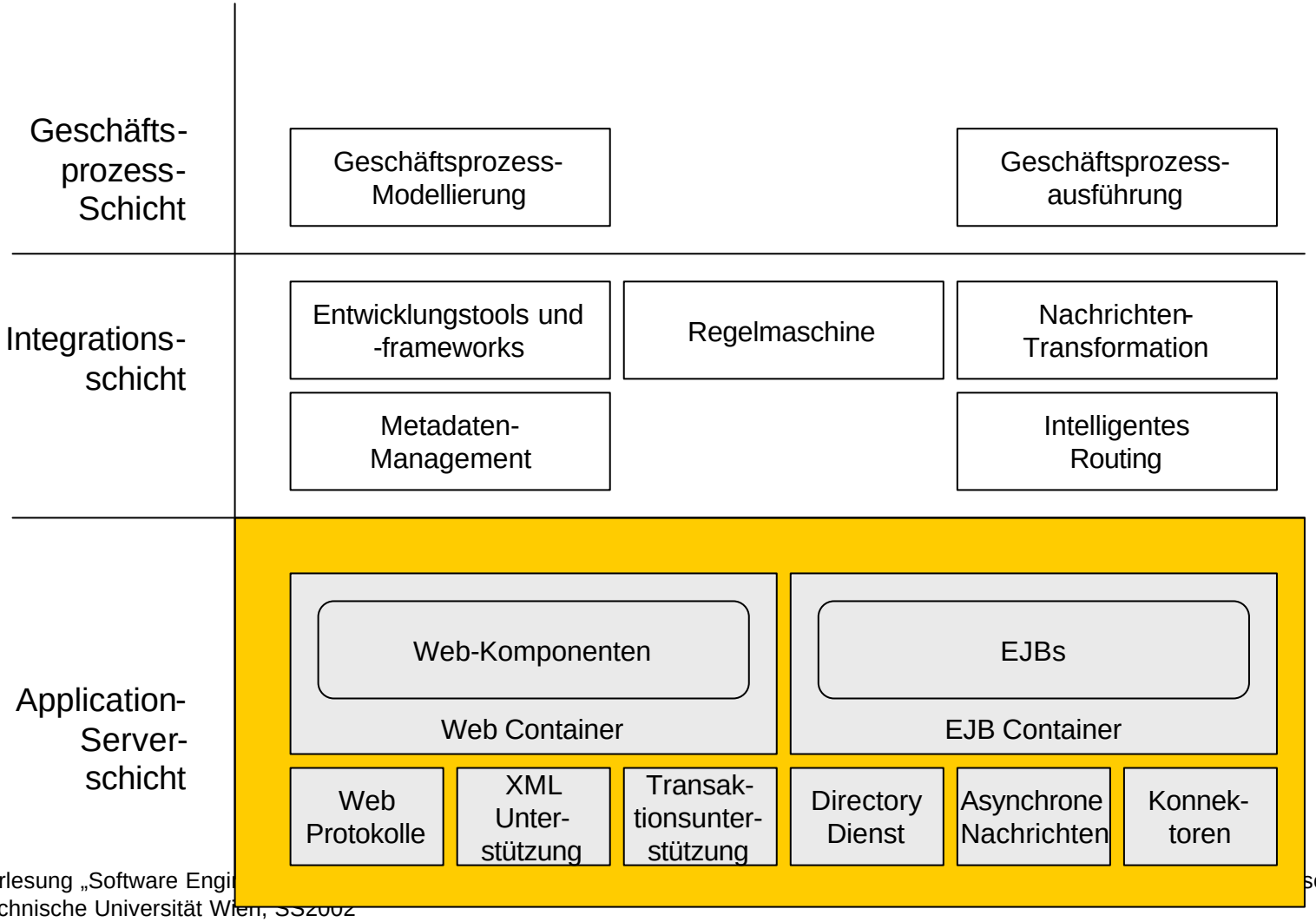
- Transaktionsklammer (Ressorce Manager) über mehr als eine transaktionsgesicherte Ressource .. Also zum Beispiel
  - DB2
  - IMS-DB
  - Und von jedem auch mehr als eine Instanz

Was leistet im Vergleich dazu ein J2EE Applikationsserver?

J2EE Applikationsserver
Wo finde ich den?



J2EE Applikationsserver
Wo finde ich den?

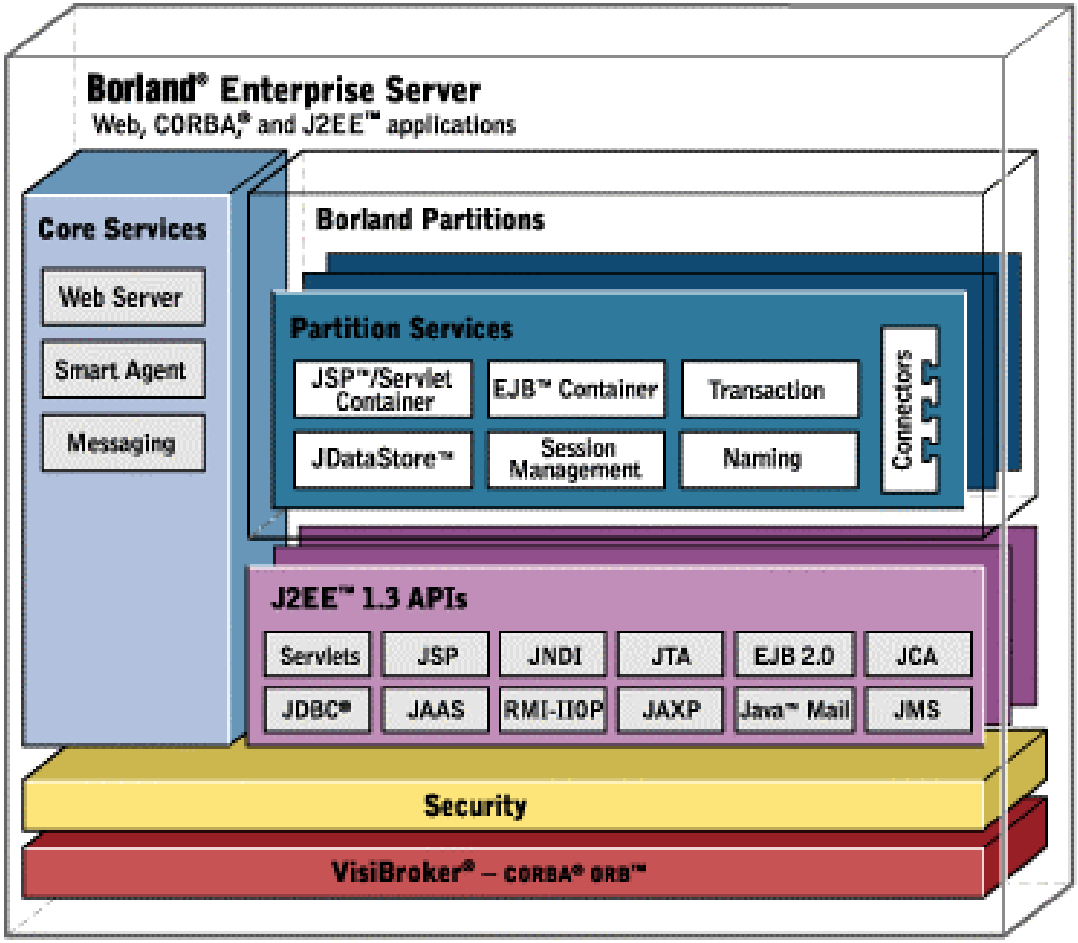


J2EE Applikationsserver Wesentliche Dienste

- Ressource Manager für TA – Steuerung
- Container für Programme
  - JSP, EJB, Servlets
- Persistenzdienste (CMP und BMP)
- Load Balancing
- Security
- Messaging
- Name Services
- Backend Connectivity (Konnektoren)

Bei CICS/IMS so nicht direkt vorhanden ...

J2EE Applikationsserver
Selbstbeschreibung eines Repräsentaten ...



Quelle:
Borland

Zusammenfassung

- Namen und Programmiersprachen haben sich geändert
- Produkte sind besser geworden – die Frameworks sind umfangreicher
- Herausforderungen sind ähnlich geblieben
  - Perfomantes Bedienen von Requests vieler Clients
  - Transaktionssteuerung
  - Weitere Anwendungsdienste
- Kenntnis „der alten Eisen“ immer nocht ganz nützlich – weil man sie immer wieder integrieren muß