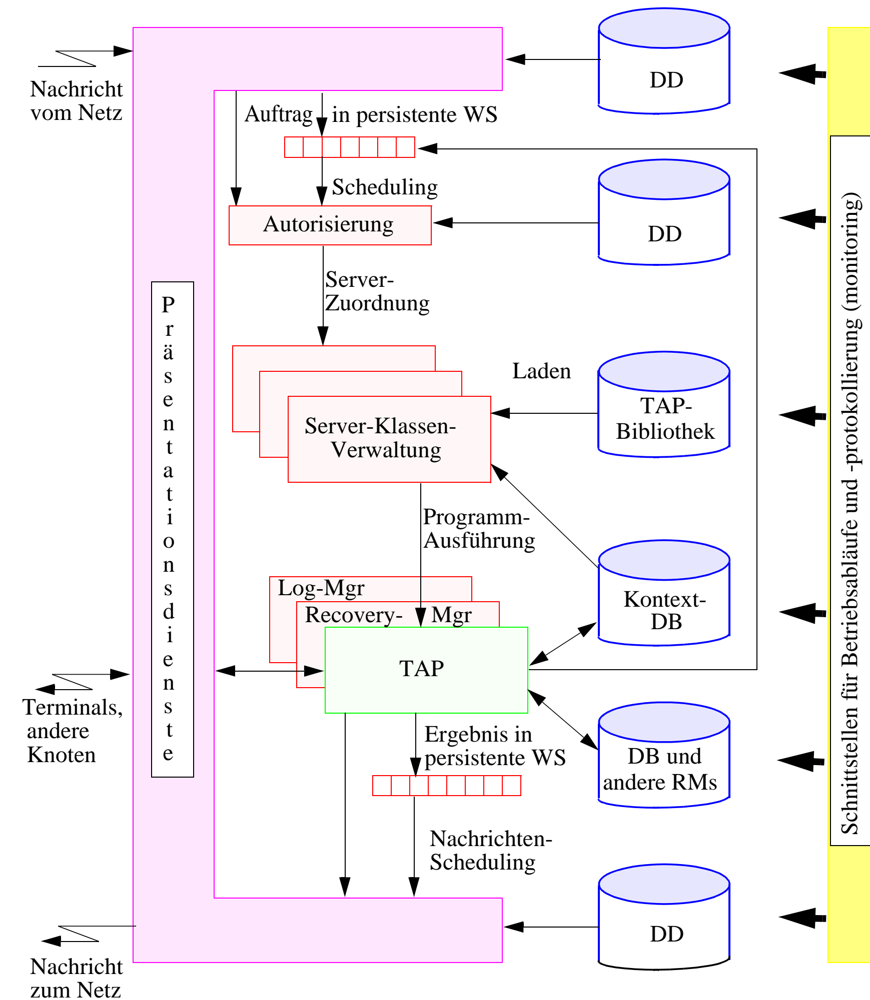
Was sind TP-Monitore? (4)

- **Was ist nun genau ein TP-Monitor?**
  - Er lässt sich als BS für transaktionsgeschützte Applikationen auffassen
  - Er ist ein Framework für Applikations-Server
  - **Er erledigt drei Aufgaben extrem gut:**
    - **Prozessverwaltung:**
      Sie schließt das Starten von Server-Prozessen,
      das Initiieren von TAPs,
      das Kontrollieren ihres Ablaufs und
      die Lastbalancierung ein
    - **Transaktionsverwaltung**[10]**:**
      Sie garantiert die ACID-Eigenschaften von allen Programmen,
      die unter ihrem „Schutz“ ablaufen.
      Dazu muss sie im Normalbetrieb Logging (z. B. Protokollieren von Nachrichten) durchführen, um im Fehlerfall Recovery-Maßnahmen ergreifen zu können
    - **C/S-Kommunikationsverwaltung**:
      Es ist Client/Server- und Server/Server-Kommunikation zu unterstützen.
      Sie erlaubt den Client- und Server-Programmen, die an einer Anwendung beteiligten Dienste und Komponenten auf verschiedene Weise aufzurufen: RPCs, Konversationen, asynchrone Nachrichten über persistente Warteschlangen (MOM: *message-oriented middleware*), Broadcasts, ...

10. The Standish Group estimates that the world electronically processes 68 million transactions every second. 53 million of the 68 million use a TP Monitor (Jim Johnson, Oct. 1998)

Struktur eines TP-Monitors

- **Kontrollfluss durch einen TP-Monitor** (vereinfacht)



➥ Diagramm repräsentiert die wichtigsten Grundfunktionen für TAP-Verwaltung und -Ausführung

Struktur eines TP-Monitors (2)

- **Präsentationsdienste**
  - bilden die Schnittstellen zwischen dem TAP und den E/A-Geräten
  - bieten Geräteunabhängigkeit (Terminaltyp, Formatkontrolle, *Scrolling*) und Kommunikationsprotokollunabhängigkeit

    ➥ wegen vieler verschiedener Präsentationsdienste (Window-Protokolle, Graphikstandards) implementieren oft eigene RM diese Dienste

- **Warteschlangenverwaltung**
  - WS-Dienste müssen transaktionsorientiert sein: Auftrag in WS wird genau einmal ausgeführt (‘exactly once’)
  - Nur bei Commit der Server-TA kommt Ergebnis in Ausgabe-WS
  - Abhängig von Anwendung und Inhalt der Nachricht können für Auslieferung verschiedene Zusicherungen vereinbart werden: ‘at least once’, ‘at most once’ oder ‘exactly once’

    ➥ WS-Mgr ist oft als RM implementiert, der zum TP-Monitor gehört

- **Server-Klassen-Verwaltung**
  - Zuständig dafür, dass für jedes TAP eine Server-Klasse definiert und aktiv ist
  - Aktivierung der Server entweder ‘by default’ oder ‘on demand’
  - Aufgaben: Erzeugen von Prozessen und WS, Laden der TAP-Codes, Besorgen der Zugriffsrechte für die WS-Zugriffe, Festlegen der Server-Prioritäten u.a.

    ➥ Aufgaben fallen auch in die Verantwortlichkeit der Lastbalancierung. Sie sind deshalb in enger Kooperation zu lösen.

    ➥ Funktionen wie Prozesserzeugung und TAP-Ausführung werden vom BS zur Verfügung gestellt.

Struktur eines TP-Monitors (3)

- **Scheduling von Aufträgen**
  - die am **häufigsten angeforderte Funktion** des TP-Monitors
  - Bestimmung des angeforderten Dienstes: lokal oder entfernter Knoten.
  - Weiterleitung des Auftrags an den Server.

- **Autorisierung der Aufträge**
  - Teil der systemweiten Sicherheitsvorkehrungen
  - Spektrum der Lösungen: von einfacher, statischer Autorisierung bis zu wertabhängiger, dynamischer Autorisierung

    ➥ Wegen spezifischer Betriebscharakteristika von TA-Systemen ist dynamische Autorisierung für individuelle Aufträge sehr wichtig

- **Kontextverwaltung**
  - 1. Speicherung von Verarbeitungskontexten, die **über TA-Grenzen** gehalten werden sollen (Mehr-TA-Vorgänge)

    ➥ Kontext-DB hat alle ACID-Eigenschaften und könnte durch ein SQL-DBS implementiert werden

  - 2. Unterstützung der Kontextnutzung **innerhalb einer TA**: Weitergabe von Zwischenergebnissen einer TA über Nachricht beim Server-Aufruf zu aufwendig; Kontextverwaltung speichert die Daten zwischen und erlaubt nachfolgende Server Zugriff auf die Daten

    ➥ Diese Zugriffe erfolgen innerhalb einer TA; deshalb sind keine persistenten Kontexte erforderlich

Struktur eines TP-Monitors (4)

- **Metadatenverwaltung**

  Annahme: globales Repository (DD, Data Dictionary)

  ➥ Je mehr der TP-Monitor tun soll, desto genauer/umfangreicher müssen die Metadaten sein

- **Metadaten-Übersicht:** Beschreibung der
  - zum verteilten TA-System gehörigen Knoten: Namen, Adressen, ...
  - lokalen Komponenten der TA-Dienste wie Log-Mgr, Recovery-Mgr, Kommunikations-Mgr, ...
  - Geräte und HW-Komponenten, die der TP-Monitor kennen muss, wie Terminals, Controller, physische Verbindungen, ...
  - TAPs und RMs, die am betreffenden Knoten installiert sind
  - Autorisierungsinformation
    - Zugriffskontrollisten für TAPs und RMs
    - über die dem System bekannten Benutzer
    - Autorisierungs-Codes (Passwörter), Sicherheitsprofile, ...
  - Konfigurationsdaten
    - für Server-Klassen über Prozesse, Tasks, Prioritäten
    - über Betriebs- und Administrations-Schnittstellen
    - für Wiederanlauf und spezielle Abläufe (Restart-Reihenfolge der RMs)

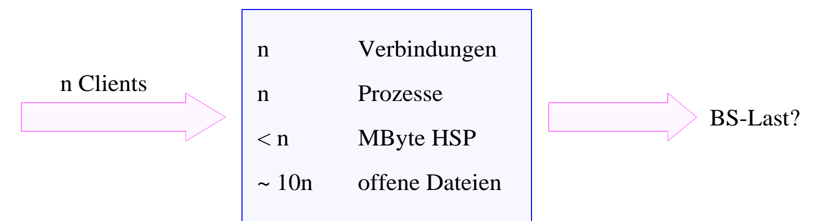
  ➥ Inhalt diese Kataloge wird gewartet und aktualisiert über System-TAs. Beim Restart wird die hier hinterlegte Konfiguration „hochgefahren”

Ablauf in dreistufigen C/S-Architekturen

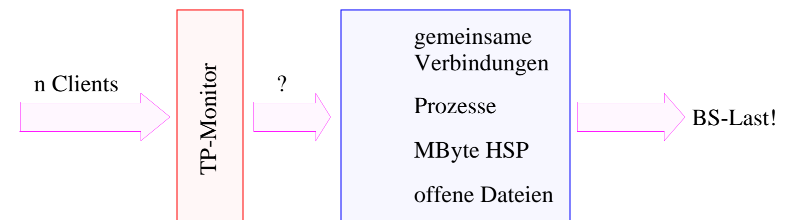
- **Typischer BM-Bedarf pro Client auf einem Server**
  - eine Kommunikationsverbindung
  - 0.5 — 1 MByte Hauptspeicher (RAM)
  - 1 oder 2 Prozesse
  - ~ 10 offene Dateikontrollblöcke (file handle)

  ➥ Soll jeder Client seine eigenen BM statisch zugeordnet bekommen (virtueller Prozessor)?

- **Herkömmliche BS-Abbildung**



- **Einsatz eines TP-Monitors**[11]

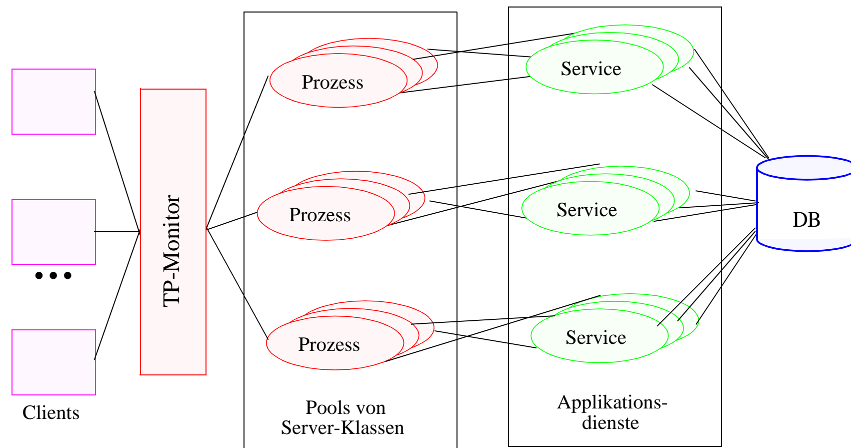


11. "TP-Monitor: The 3-Tier Workhorse" (Jeri Edwards)

Ablauf in dreistufigen C/S-Architekturen (2)

- **BM-Zuteilung erfolgt auftragsbezogen**
  - OLTP-Applikation besteht aus einer Menge von Diensten (TAPs)
  - Server-Klasse besteht aus Pool von (statisch erzeugten) Prozessen (mit Tasks/Threads), welche vorab geladene TAPs abwickeln können. TP-Monitor kann dynamisch neue Prozesse starten

    ➥ Lastbalancierung
  - Applikation kann eine oder mehrere Server-Klassen besitzen
  - TP-Monitor weist Server-Klasse ankommenden Auftrag (TAC) zu; nach Abwicklung Freigabe der auftragsbezogenen BM. TP-Monitor leitet die Antwort an den Client weiter



- **Bildung von Server-Klassen**
  - nach Prioritätsaspekten für die TAP-Ausführung
  - nach Applikationstypen
  - nach Antwortzeitvorgaben
  - nach Fehlertoleranzanforderungen, ...

Aufbau des DB-Servers

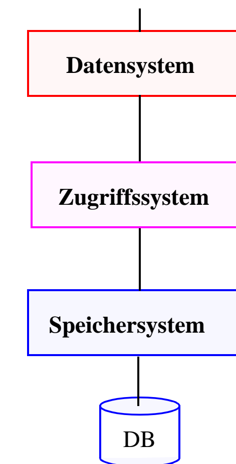
- **Wichtigstes Entwurfsziel: datenunabhängiges DBS**
- **Vereinfachtes Schichtenmodell**

Aufgaben | **Art der Operationen**

Übersetzung und Optimierung von Anfragen

Verwaltung von physischen Sätzen und Zugriffspfaden

DB-Puffer- und Externspeicher-Verwaltung



deskriptive Anfragen
Zugriff auf Satzmengen

Satzzugriffe

Seitenzugriffe

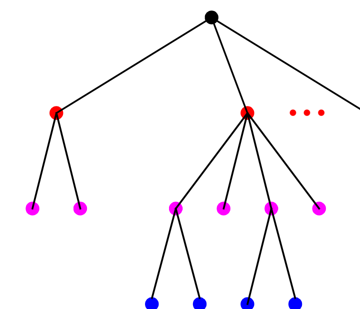
Achtung: Schichtung verkörpert „benutzt"-Hierarchie!
Warum sind das keine C/S-Beziehungen?

- **Dynamischer Kontrollfluss einer Operation im DBS**

Datensystem

Zugriffssystem

Speichersystem



DBS-Operationen (API)

Füge Satz ein
Modifiziere Zugriffspfad

Stelle Seite bereit
Gib Seite frei

Lies / Schreibe Seite

Ablaufbeispiel

- **Transaktionsprogramm „Auszahlung“:**

  Read message (kontonr, schalternr, zweigstelle, betrag) from Terminal;

  BEGIN TRANSACTION

  UPDATE Konto

  SET kontostand = kontostand - *betrag*
  WHERE konto_nr = *kontonr* and kontostand >= *betrag*

  . . .

  UPDATE Schalter

  SET kontostand = kontostand - *betrag*
  WHERE schalter_nr = *schalternr*

  UPDATE Zweigstelle

  SET kontostand = kontostand - *betrag*
  WHERE zweig_stelle = *zweigstelle*

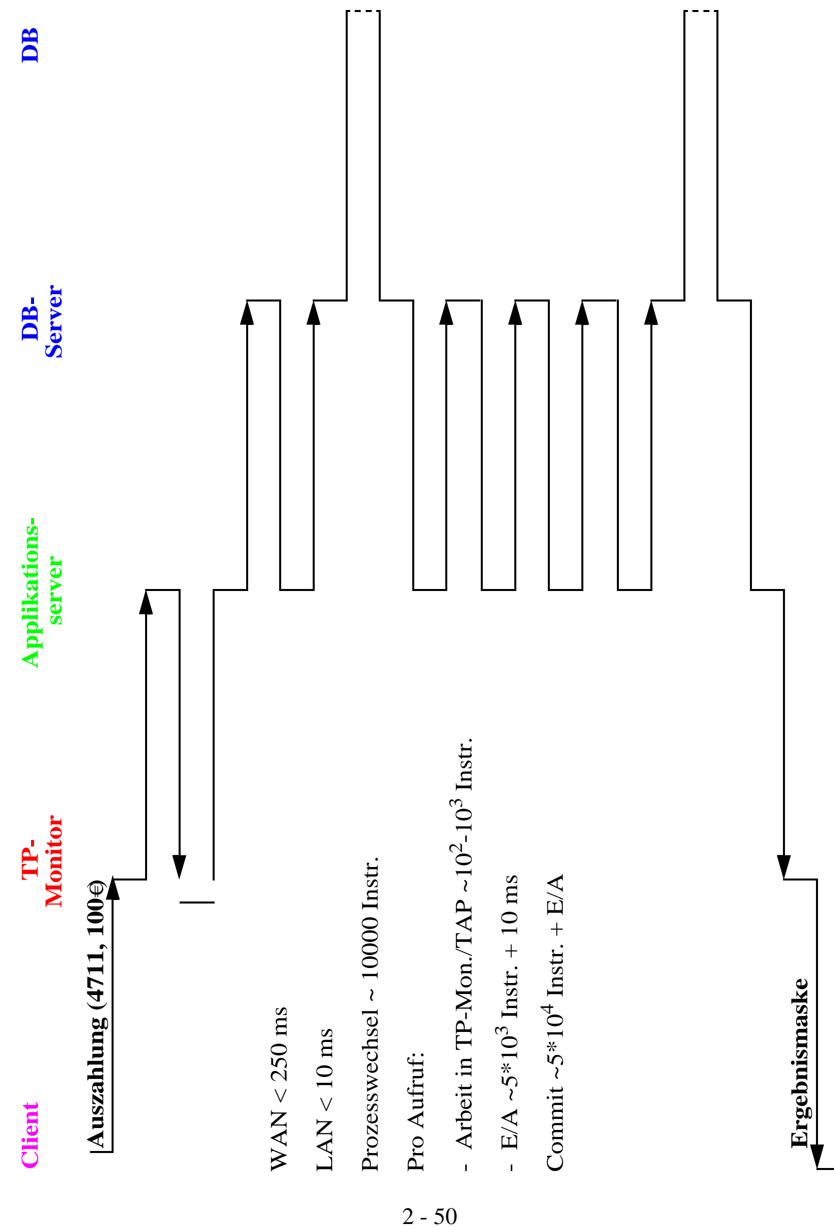
  INSERT INTO Ablage (zeitstempel, werte)

  COMMIT TRANSACTION ;

  Write message (kontonr, kontostand, . . .) to Terminal

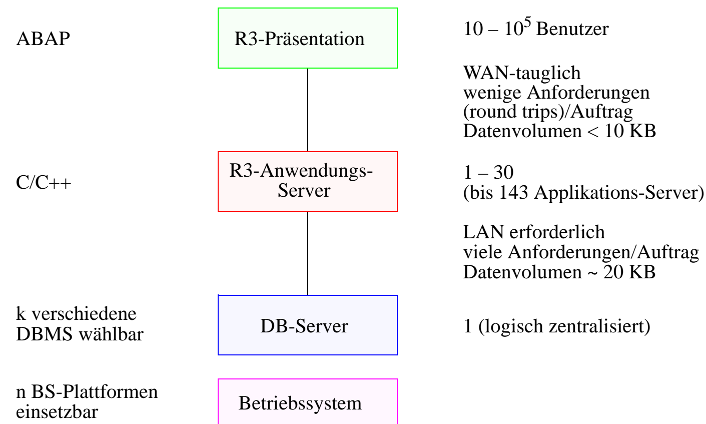
- **Annahmen**

  - Alle Indexstrukturen (B*-Bäume) können im DB-Puffer gehalten werden
  - Es gibt sehr viele Objekte vom Typ **Konto**. Sie müssen immer vom Externspeicher geholt werden
  - Alle Objekte der Typen **Schalter** und **Zweigstelle** (kleine Mengen) sind bereits im DB-Puffer
  - Alle Änderungen (UPDATE) werden „später” verdrängt
  - Alle Log-Daten werden bei Commit auf einmal geschrieben



Dreistufige C/S-Architektur

- **SAP/R3-Realisierung in**



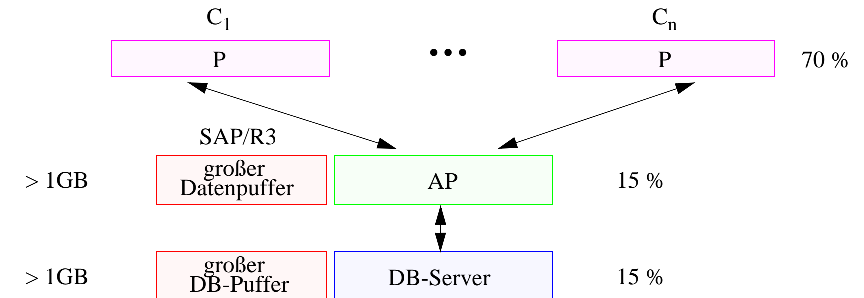
- **Funktionen in der mittleren Schicht (M)**
  - ABAP-Interpreter
  - TP-Monitor
  - AW-Systeme
  - Caching von DB-Daten
  - DBMS-Schnittstelle

➥ Unternehmensintegration über eine einzige DB, Skalierbarkeit sehr wichtig

- **Zahlen (SAP ERP Data Model (Part of Business Suite '05)**
  - DB-Schema: > 67 000 Tabellen (Relationen), 10 000 Sichten
    700 000 Spalten, 13 000 Indexe, 10^8 Sätze (Anfangsgröße)
  - > 270 M LOCs Anwendungsprogramme
  - Wachstumsrate pro größerer Version: + 30%
  - Systemgröße (leer) auf Platte: 57 GB (*disk foot print, initial size*)
  - Unterstützung von mehr als 20 Sprachen

Dreistufige C/S-Architektur (2)

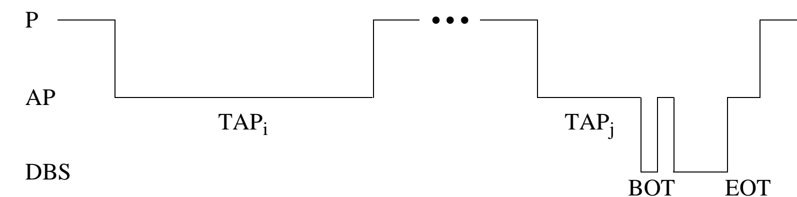
- **Einsatz von großen Puffern**



- **Ziele**
  - Anwendungsdaten sollen in der Nähe der Anwendung gehalten werden (lokal im AP-Puffer)
  - Ausnutzung von Anwendungswissen bei der Zugriffssynchronisation auf Ebene des AP-Servers

➥ AP-Server enthält TP-Monitor-Funktionalität und muss viel DBS-Funktionalität reimplementieren

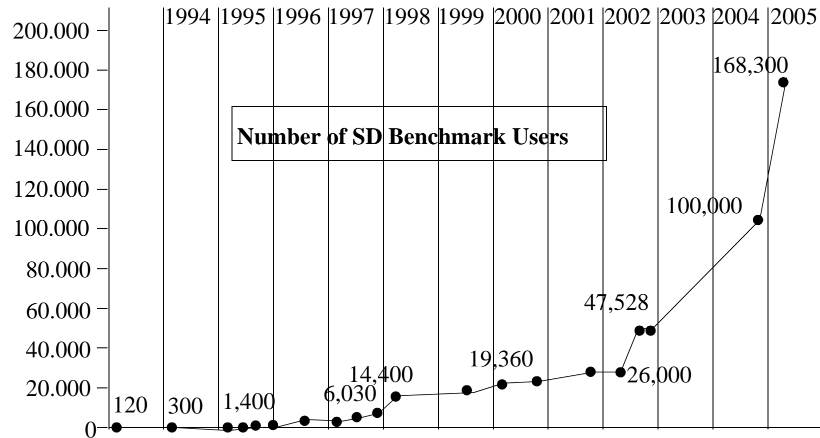
- **Zeitlicher Ablauf einer Transaktion**



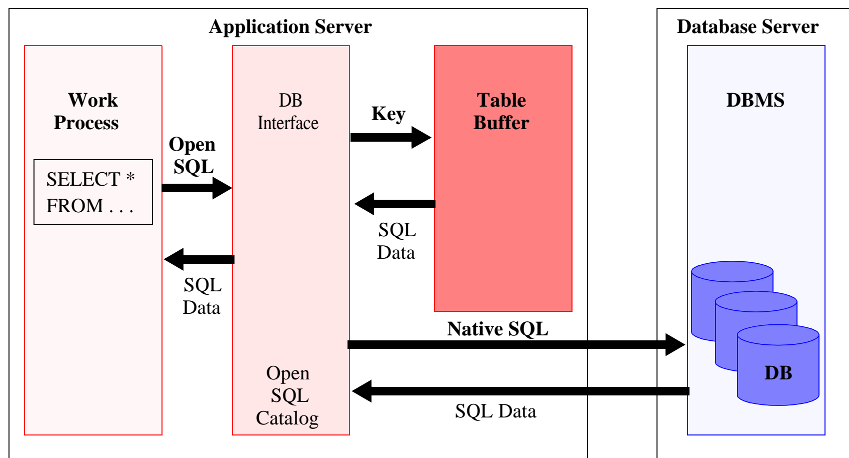
- **Eigenschaften**
  - Präsentation mit GUI ist sehr aufwendig; Präsentationslogik (Benutzerinteraktion) sollte nicht im TA-Pfad liegen!
  - Mengenorientierter Zugriff (SQL) auf AP-Puffer ist sehr restriktiv
  - Hauptsächlich genutzte DBS-Funktionalität: **Datenspeicher und Logging/Recovery**

SAP/R3 – Results and Observations

- **Published Results for SD Benchmark (three-tier)**

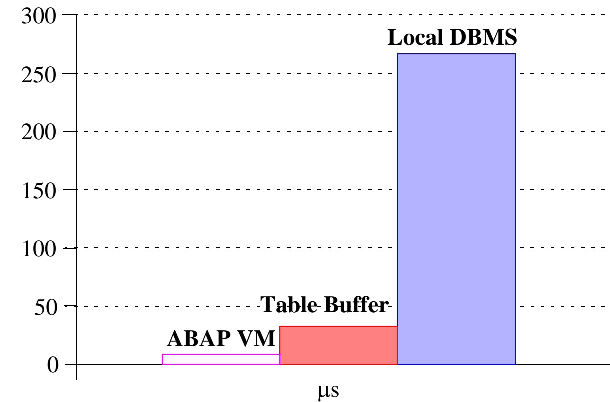


- **Table Buffer in Application Server (caching close to the application)**

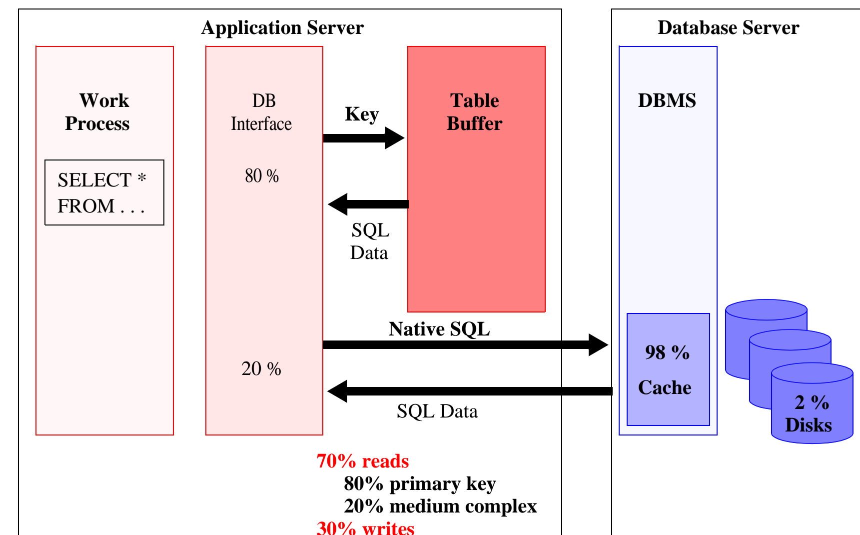


SAP/R3 – Results and Observations (2)

- **Performance of Table Buffer vs. DBMS (Primary Key Access)**



- **Typical OLTP Traffic Distribution**



TA-Verarbeitung in offenen Systemen

- **Idee der Client/Server-Verarbeitung korrespondiert mit dem Konzept von Offenen Systemen.**

  Definition von IEEE im Rahmen der POSIX-Aktivität:

  Ein **offenes System** ist

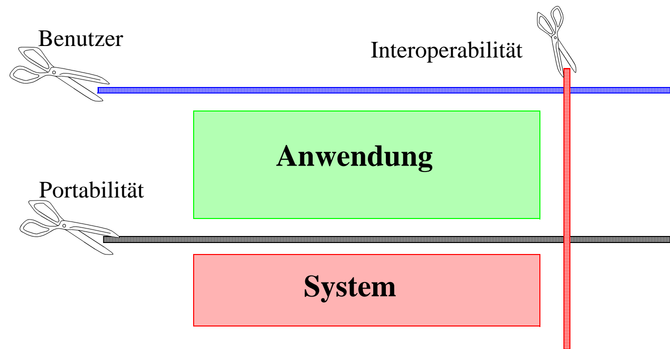
  „ein System, das in ausreichendem Maße offengelegte Spezifikationen für Schnittstellen und dazugehörige Formate implementiert, damit entsprechend gestaltete Anwendungssoftware

  - auf eine **Vielzahl verschiedener Systeme portiert** werden kann (mit Anpassungen),
  - mit anderen **Anwendungen lokal und entfernt** interoperabel ist,
  - mit Benutzern in einer Art interagiert, die das **Wechseln der Benutzer zwischen Systemen erleichtert**".

  ➥ Diese Anforderungen spiegeln sich z. T. auch in den Zielvorstellungen von Client/Server-Systemen wider.

- **Offenheit impliziert Standardisierung**

  - Definition und Veröffentlichung von Schnittstellen



  - **Systemschnittstellen** gewährleisten die Portabilität der AW-Software
  - **Aufrufschnittstellen** erlauben die Interoperabilität zwischen AWs und zwischen Systemen
  - **Benutzerschnittstellen** regeln einen einheitlichen Benutzerzugang

TA-Verarbeitung in offenen Systemen (2)

- **Standards zur TA-Verwaltung**[12]

  TP-Monitore benötigen Standards, weil sie letztendlich den „Klebstoff" verkörpern, der die unterschiedlichen und heterogenen SW-Komponenten zusammenfügt.

  - Ihre AW laufen auf verschiedenartigen Plattformen und
  - haben Zugriff zu verschiedenen DBs und RMs
  - Die AW haben absolut kein Wissen voneinander

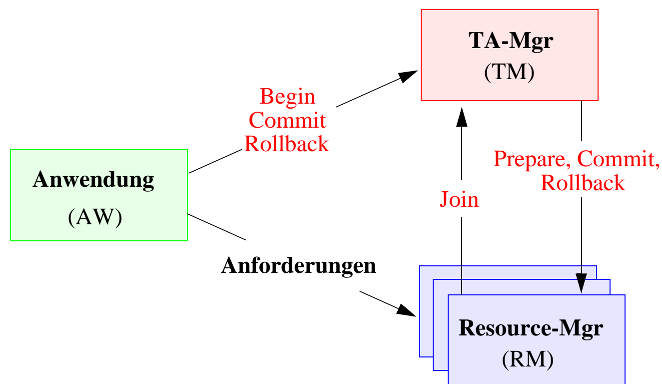
  ➥ Einziger Weg zur Verknüpfung: **„Offene Standards"**
  Bereitstellung von Schnittstellen zwischen TP-Monitor und RMs, TP-Monitor untereinander und TP-Monitor und AW

- **Standards kommen aus zwei Quellen**

  - **ISO:**
    OSI-TP spezifiziert die Nachrichtenprotokolle zur Kooperation von TP-Monitoren
  - **X/OPEN** definiert den allgemeinen Rahmen für TA-Verarbeitung
    X/Open DTP (distributed transaction processing) stellt eine SW-Architektur dar, die mehreren AWP erlaubt, gemeinsam Betriebsmittel mehrerer RMs zu nutzen und die Arbeit der beteiligten RMs durch globale Transaktionen zusammenzufassen.

12. „May all your transactions commit and never leave you in doubt!" (Jim Gray)

TA-Verarbeitung in offenen Systemen (3)



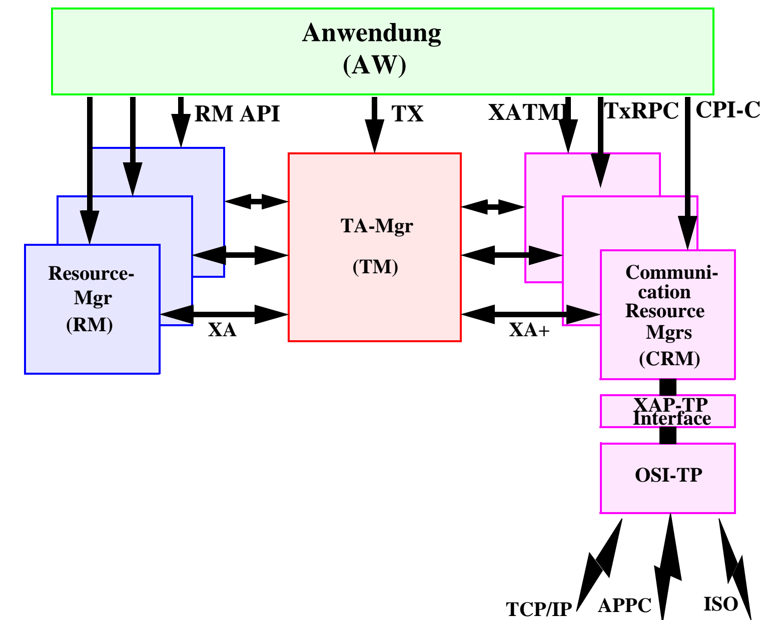
- **X/Open DTP (1991)** für die lokale Zusammenarbeit von Anwendung (AW), TA-Mgr (TM) und Resource-Mgrs (RMs)

- **Standardisierung**
  - Schnittstelle zur Transaktionsverwaltung (TX: AW — TM)
  - Schnittstelle zwischen TM und RMs
    (XA: zur TA-Verwaltung und ACID-Kontrolle)
  - zusätzlich: Anforderungsschnittstellen (API, z. B. SQL)

- **TA-Ablauf**
  - Die AW startet eine TA, die vom lokalen TA-Mgr verwaltet wird
  - RMs melden sich bei erster Dienst-Anforderung beim lokalen TA-Mgr an (Join)
  - Wenn AW Commit oder Rollback ausführt (oder zwangsweise zurückgesetzt wird), schickt TA-Mgr Ergebnis zu den RMs (über Broadcast)

➥ **hier sorgen die RMs für Synchronisation und Logging**
in ihrem Bereich (private Lock-Mgr und Log-Mgr) — Warum?

TA-Verarbeitung in offenen Systemen (4)



- **X/Open DTP (1993)** standardisiert die verteilte TA-Verarbeitung.
  - Es kommt der „Communication Resource Manager“ (CRM) hinzu.
  - XA+ erweitert Schnittstelle XA für den verteilten Fall.

- **Definition von Schnittstellen auf der AW-Ebene**
  - TxRPC definiert transaktionalen RPC. Seine TA-Eigenschaften können ausgeschaltet werden (optional)
  - CPI-C V2 ist Konversationsschnittstelle (peer-to-peer), welche die OSI-TP-Semantik unterstützen soll
  - XATMI ist Konversationsschnittstelle für Client/Server-Beziehungen

➥ Sind drei Schnittstellen-Standards erforderlich? Kompromisslösung für drei existierende Protokolle (DCE, CiCS, Tuxedo)

Transaktionsverarbeitung in offenen Systemen (5)

- **TA-Ablauf**
  - AW startet TA, die vom lokalen TA-Mgr verwaltet wird
  - Wenn die AW oder der RM, der für die AW eine Anforderung bearbeitet, eine entfernte Anforderung durchführen, informieren die CRMs an jedem Knoten ihre lokalen TA-Mgr über die ankommende oder ausgehende TA
  - TA-Mgr verwalten an jedem Knoten jeweils die TA-Arbeit am betreffenden Knoten
  - Wenn die AW COMMIT oder ROLLBACK durchführt oder scheitert, kooperieren alle beteiligten TA-Mgr, um ein atomares und dauerhaftes Commit zu erzielen.

Zentralisiertes Zweiphasen-Commit

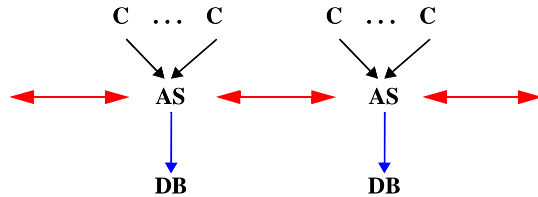
* synchrone Log-Ausgabe (log record is force-written)
„End“ ist wichtig für Garbage Collection im Log von C

- **Protokoll (Basic 2PC)**
  - Folge von Zustandsänderungen für Koordinator und für Agenten
    - Protokollzustand auch nach Crash eindeutig: synchrones Logging
    - Sobald C ein NO VOTE (FAILED) erhält, entscheidet er ABORT
    - Sobald C die ACK-Nachricht von allen Agenten bekommen hat, weiß er, dass alle Agenten den Ausgang der TA kennen

  ➥ C kann diese TA vergessen, d. h. ihre Log-Einträge im Log löschen!
  - Warum ist das 2PC-Protokoll blockierend?
- **Aufwand im Erfolgsfall:**
  - Nachrichten:
  - Log-Ausgaben (forced log writes):

2PC-Protokoll in TA-Bäumen

- **Typischer AW-Kontext: C/S-Umgebung**
  - Beteiligte: Clients (C), Applikations-Server (AS), DB-Server (DB)
  - unterschiedliche *Quality of Service* bei
    Zuverlässigkeit, Erreichbarkeit, Leistung, ...



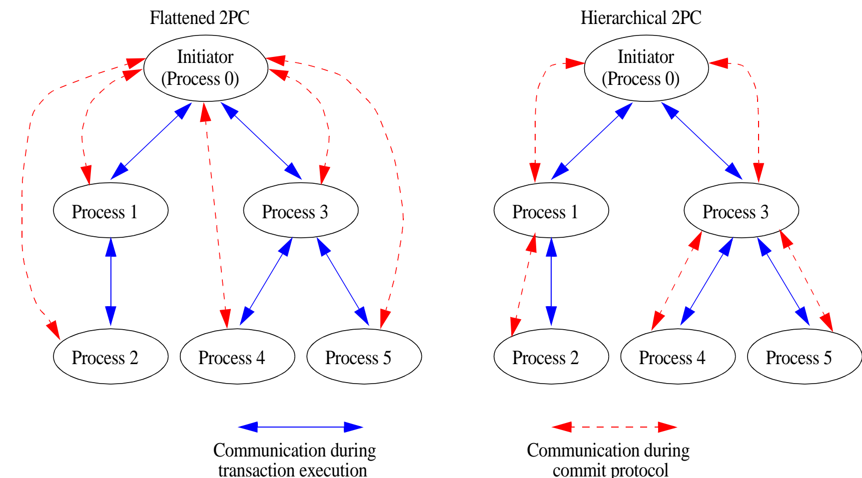
- **Wer übernimmt die Koordinatorrolle? – wichtige Aspekte**
  - TA-Initiator
  - Zuverlässigkeit und Geschwindigkeit der Teilnehmer
    - Anzahl der Teilnehmer
    - momentane Last
    - Geschwindigkeit der Netzwerkverbindung
  - Kommunikationstopologie und -protokoll
    - sichere/schnelle Erreichbarkeit
    - im LAN: Netzwerkadressen aller Server sind bekannt; jeder kann jeden mit Datagrammen oder neu eingerichteten Sitzungen erreichen
    - im WAN: „transitive" Erreichbarkeit; mehrere Hops erforderlich;
      Initiator kennt nicht unbedingt alle (dynamisch hinzukommenden) Teilnehmer (z. B. im Internet)
  - Im einfachsten Fall: TA-Initiator übernimmt Koordinatorrolle

➥ sinnvoll, wenn Initiator ein
zuverlässiger, schneller und gut angebundener Applikations-Server ist!

2PC-Protokoll in TA-Bäumen

- **Drei wichtige Beobachtungen**
  - Während der TA-Verarbeitung formen die involvierten Prozesse einen (unbalancierten) Baum mit dem Initiator als Wurzel. Jede Kante entspricht einer dynamisch eingerichteten Kommunikationsverbindung
  - Für die Commit-Verarbeitung kann der Baum „flachgemacht" werden
    - Der Initiator kennt die Netzwerkadressen aller Teilnehmerprozesse bei Commit (durch „piggybacking" bei vorherigen Aufrufen)
    - Nicht möglich, wenn die Server die Information, welche Server sie aufgerufen haben, kapseln

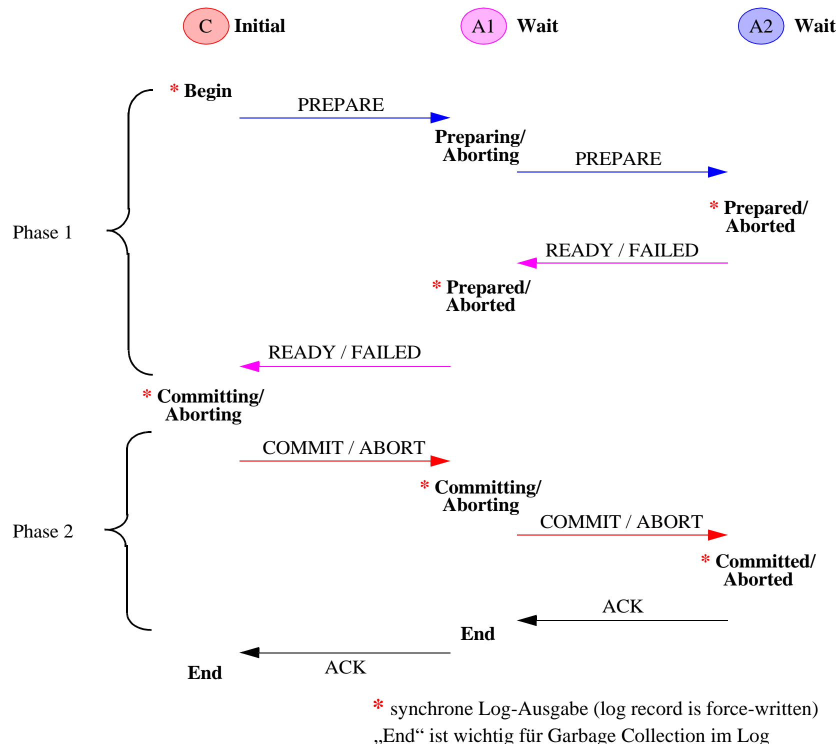
Initiator = Coordinator



  - „Flachmachen" kann als spezieller Fall der Restrukturierung der C/S-Aufrufhierarchie gesehen werden
    - Es könnte auch ein zuverlässiger innerer Knoten als Koordinator ausgewählt werden
    - „Rotation" der Aufrufhierarchie um den neu gewählten Koordinatorknoten

Hierarchisches 2PC

- **Allgemeineres Ausführungsmodell**
  - beliebige Schachtelungstiefe, angepasst an C/S-Modell
  - Modifikation des Protokolls für "Zwischenknoten"



* synchrone Log-Ausgabe (log record is force-written)
„End" ist wichtig für Garbage Collection im Log

- **Aufwand im Erfolgsfall:**
  - Nachrichten:
  - Log-Ausgaben (forced log writes):
  - Antwortzeit steigt mit Schachtelungstiefe
- **Problem Koordinator-/Zwischenknotenausfall** ➥ **Blockierung möglich!**

Optimierte Protokolle für 2PC

- **Bisherige Optimierungsüberlegungen**
  - erfordern Änderungen der API oder sind unpraktisch
  - „belästigen" C eine sehr lange Zeit
  - sind – bis auf die Leseroptimierung – nicht allgemein einsetzbar

- **Allgemeine Optimierungsdimensionen**

1. Weniger Nachrichten und (synchrone) Log-Ausgaben
   - Keine absolute Notwendigkeit, den „Begin"-Eintrag synchron in den Log von C zu schreiben:
     a) C könnte vorher „Prepare" für sich durchführen und diese Log-information als 2PC-Beginn interpretieren (nach Crash-Recovery) oder
     b) nach einem Crash diese TA einfach vergessen (ABORT)
   - Einführung spezifischer Konventionen (Default-Reaktionen), falls zur Behandlung einer Situation keine explizite Information mehr gefunden wird[13]
   - so genanntes „Presumed"-Verhalten für T_i, falls beispielsweise C seine Log-Einträge für T_i schon gelöscht hat

2. Kürzerer kritischer Pfad, bis die Sperren freigegeben werden können
   - 1) dient ganz allgemein diesem Ziel
   - Leser-Optimierung für Teilbäume
   - Wahl eines zuverlässigen oder schnellen Servers als Koordinator (Koordinator-Transfer)

3. Reduzierung des Blockierungspotenzials
   - alle Maßnahmen von 1) und 2) helfen implizit, die Blockierungs-Wahrscheinlichkeit zu reduzieren
   - Verkürzung der Protokolldauer verkleinert das Zeitfenster für eine mögliche Blockierung (Crash oder Nicht-Erreichbarkeit von C)

13. Basic 2PC = PN-Protokoll (presumed nothing), da keine Default-Reaktionen spezifiziert sind

PA-Protokoll

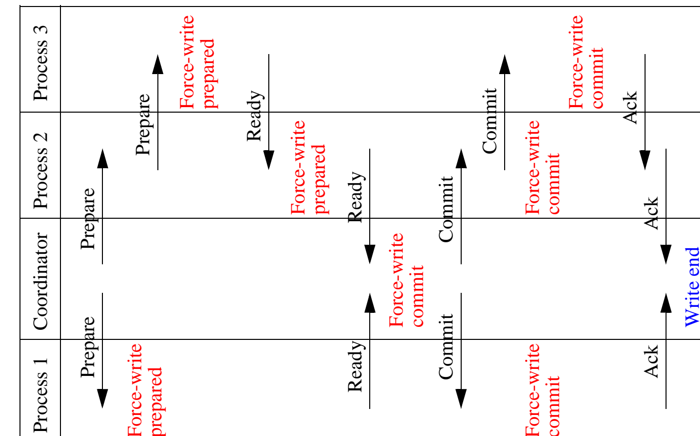
- **Kandidaten für die Protokoll-Optimierung**
  Abschwächung bzw. Weglassen von
  - synchronem Logging von „Begin“ durch C
  - synchronem Logging der Commit/Abort-Einträge durch die A_i
  - ACK-Nachrichten der A_i

- **„Presumed Abort“ erlaubt die Nutzung aller drei Möglichkeiten für Verlierer-TA**
  - „Begin“ stellt keine Probleme dar
  - Falls der Commit/Abort-Eintrag eines Agenten A für T_i verlorengeht, kann er diese Information von C erfragen (nach Restart von A)
  - Weglassen von ACK führt keine neuen Probleme ein, wenn C ein „unendlich langes Gedächtnis“ hat
  - Wenn **C die Log-Einträge von T_i löscht** (Garbage Collection), kann später der korrekte Ausgang von T_i durch **„Presumed Abort“ gefolgert** werden
  - Da der Abort-Fall keine „persistente“ Information mehr benötigt, kann **auch C bei Abort-Entscheidungen auf ein Force-Write** verzichten

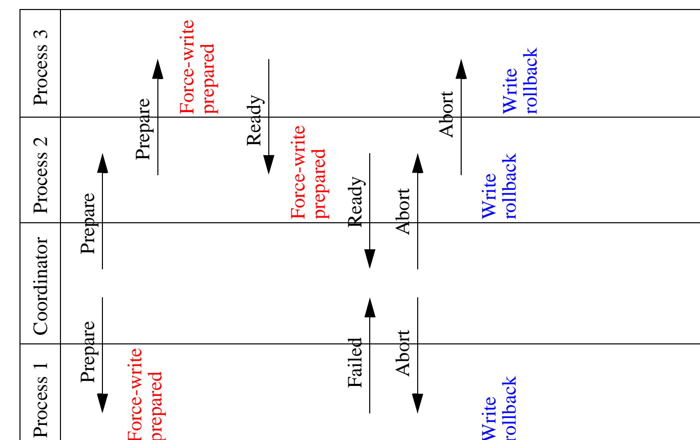
- **PA ist nicht „symmetrisch“ für Gewinner-TA**
  - Gleiche Optimierung würde für Gewinner viel mehr einsparen
  - Abort einer TA ist ein destruktiver Akt, Commit jedoch ein konstruktiver!

  ➥ **fundamentale Asymmetrie zwischen Abort und Commit!**
  - deshalb:
    - alle Teilnehmer müssen bei Commit-Entscheidung **synchrones Logging** durchführen
    - und **ACK an C schicken**, um ihren Commit-Abschluss zu bestätigen

PA-Protokoll (2)



Case 2: Transaction commit



Case 1: Transaction abort

PC-Protokoll

- **Wie muss Presumed Commit gestaltet werden?**

  Annahme: PC-Protokoll schreibt die Commit-Log-Einträge nicht synchron aus und enthält keine ACK-Nachrichten:

  - Wenn A_j nach seinem Restart feststellt, dass er im Prepare-Zustand ist und bei C den Ausgang von T_i erfragt, kann C in einer von drei Phasen (Zuständen) sein:
    - in der 1. Phase, bevor der Commit-Log-Eintrag von C geschrieben ist
    - in der 2. Phase, in der ein persistenter Log-Eintrag von C vorhanden ist
    - in der 3. Phase, in der die Log-Einträge von T_i bereits gelöscht sind
  - In diesem Fall kann C nicht zwischen 1. und 3. Phase unterscheiden. Die erforderliche Entscheidungen (Phase 1: Abort und Phase 3: Commit) sind jedoch miteinander unverträglich
  - Als Entscheidungshilfe: synchrones Logging vom Begin-Eintrag ist jetzt ein Muss für die Protokoll-Korrektheit

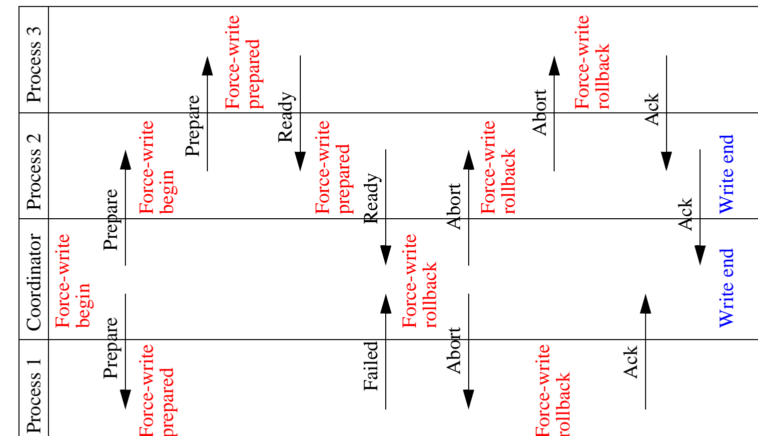
- **Charakteristische Eigenschaften von PC**

  - **synchrones** Logging der Begin- und Commit/Abort-Log-Einträge für C
  - keine ACK-Nachrichten für Gewinner-TA und damit auch kein synchrones Logging der lokalen Commit-Einträge für A_i

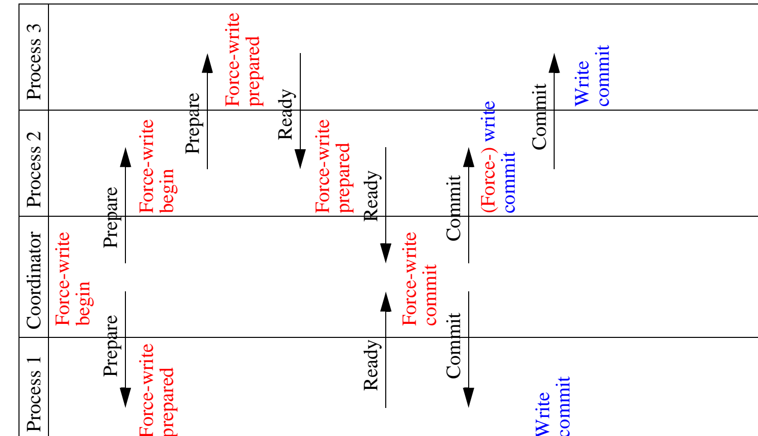
    ➥ **signifikante Einsparung von Logging-Kosten!**

  - Force-Write der Commit-Einträge bei Zwischenknoten kann eingespart werden, da Information bei C nachgefragt werden kann
  - jedoch explizite ACK-Nachrichten für Verlierer-TA und damit vorheriges synchrones Logging der Abort-Einträge für A_i

PC-Protokoll (2)



Case 1: Transaction abort



Case 2: Transaction commit

PC- und PA-Protokolle – Einsparungen

- **PA-Variante von 2PC spart**
  - für Verlierer-TAs
    - N Nachrichten (ACKs)
    - N+2 synchrone Log-Ausgaben (einschließlich „Begin“ von C)
  - für Gewinner-TAs : Nichts!
- **PC-Variante von 2PC spart**
  - für Gewinner-TAs
    - N Nachrichten (ACKs)
    - N synchrone Log-Ausgaben (für „Commit“ von A_i)
  - für Verlierer-TAs : Nichts!

➥ **geringfügig höhere Einsparungen bei PA:**
Es optimiert jedoch den weit weniger häufigen Fall!

- **Beide Protokoll-Varianten können weiter optimiert werden**
  - insbesondere mit der häufig möglichen Leser-Optimierung für Teilbäume
  - Behandlung der Zwischenknoten bringt wesentlichen Vorteil für die PA-Variante[14]

    ➥ **PA wurde für den ISO/OSI XA-Standard ausgewählt:**
    Hierarchisches PA-2PC
  - PC ist jedoch die attraktivere Variante für „flattened 2PC“
- **Presumed-Any-Protokoll**
  PA, PC und PN können in derselben Föderation von Servern koexistieren:
  Falls einzelne Server jedoch nicht alle Varianten unterstützen, müssen
  PA, PC und PN in einer TA zusammenwirken

14. implementiert in: R* von IBM, TMF von Tandem, VAX/VMS von DEC, Transarc von Encina, Tuxedo usw.

Zusammenfassung

- **Wichtige Server-Eigenschaften**
  - Auftrag — Berechnung — Antwort (Ergebnis)
  - Verschiedenste Kommunikationsprotokolle
  - verbindungsorientiert oder verbindungslos
  - Ein wichtiges zusätzliches Konzept: **„Session“**
- **Bewertung des Client/Server-Ansatzes**
  im Vergleich zu zentralen Systemstrukturen
  - Er bietet eine Reihe von Vorteilen insbesondere bei Kosten, Wachstum, Flexibilität, Offenheit, Dezentralisierungsmöglichkeit usw.
  - Prinzipielle Nachteile liegen vor allem in der **höheren Komplexität** bei **der Wartung und Verwaltung** großer, heterogener Client/Server-Systeme sowie im **Fehlen eines „single system image“**
- **Zusammenfassung der Vor- und Nachteile**

Pro	Contra
Kosten von Hard- und Software	höhere Komplexität durch Verteilung und Heterogenität
Herstellerunabhängigkeit	Gesamtkosten schwer zu überschauen
attraktive Funktionalität	komplexe Netzinfrastrukturen
Ergonomie der Benutzeroberfläche von PCs und Workstations	unausgereifte Managementlösungen
Flexibilität und Skalierbarkeit	unausgereifte Sicherheitsmechanismen
Entkopplung durch Dezentralisierung	Trainingsaufwand
Entsprechung organisatorischer Strukturen	wichtige Anwendungen nicht verfügbar

Zusammenfassung (2)

- **C/S-Architekturen**
  - 2-stufig: fette Clients vs. fette Server, keine Skalierung
  - 3-stufig: gute Skalierbarkeit, usw.
  - n-stufig: meist Einbezug von Web-Komponenten
- **Server-Strukturen und -Techniken**
  - Zur Steigerung der Effizienz: Einsatz von Programmverkettung, Multi-Tasking, Multi-Threading, ablaufinvarianten Programmen
  - Programmierung dennoch so einfach wie möglich:
    Einen Auftrag bearbeiten, Kontext = lokale Variablen, Isolierung, Code-Ablaufinvarianz durch Compiler
  - Aufgabe der Middleware insbesondere: Strategien änderbar (optimierbar)
- **TP-Monitore**
  - liefern den „Klebstoff" in komplexen C/S-Umgebungen
  - erlauben die Umsetzung des Transaktionskonzeptes als Grundlage zur Realisierung zuverlässiger verteilter Systeme
  - sind Spezialisten für **Prozess-, Transaktions- und C/S-Kommunikations-Verwaltung**
- **X/OPEN DTP**
  - Interoperabilität über standardisierte Schnittstellen und Protokolle
  - für die lokale Zusammenarbeit von Anwendung, TA-Mgr und RMs
  - und für die verteilte TA-Verarbeitung mit hierarchischen 2PC
- **Kandidaten für die 2PC-Protokoll-Optimierung**
  Abschwächung bzw. Weglassen von
  - synchronem Logging von „Begin" durch C
  - synchronem Logging der Commit/Abort-Einträge durch die A_i
  - ACK-Nachrichten der A_i
  - **„Presumed Abort"** erlaubt Nutzung aller drei Möglichkeiten für Verlierer-TA

C/S-Architekturen – Vergleich

- **Vergleich der Anwendungscharakteristika**
  Intergalaktisches C/S vs. abteilungsbezogenes C/S

Anwendungscharakteristika	abteilungsbezogenes C/S	intergalaktisches C/S
Anzahl von Clients pro AW	< 100	> 10^6
Anzahl von Servern pro AW	1 oder 2 homogene Server	10^5 ++ (viele heterogene Server in verschiedenen Rollen)
Geographie	Campus	global
Interaktionen zwischen Servern	nein	ja
Middleware (Verteilungsplattform)	SQL und stored procedures	Komponenten im Internet und in Intranets
C/S-Architektur	2-stufig	3-stufig
transaktionsgeschützte Änderungen	selten	sehr häufig
Multimedia-Inhalt	gering	hoch
Mobile Agenten	keine	zunehmend mehr
Client-Front-Ends	fette Clients (GUIs)	On-demand-Clients, Web-Tops, komplexe Dokumente
Zeitrahmen	1985 — heute	1997 — 2007 und danach

Zusammenfassung - Vielfalt an Bezeichnungen

- **RPC-basierte Systeme:**
  Ein RPC ermöglicht die synchrone Kommunikation zwischen einem Client und einem Server, der sich auf einem anderen Rechner befindet. Aus der Sicht eines aufrufenden Programms (Client) sieht der RPC wie ein gewöhnlicher Prozeduraufruf aus. **Synchrone Kommunikation** bedeutet, dass der Client beim Aufruf des RPC den Programmfluss unterbricht und auf die Ergebnisse der Prozedur wartet. Der RPC verbirgt also die Etablierung eines Kommunikationskanals durch die Bereitstellung einer **standardisierten Programmierschnittstelle**. RPC-basierte Systeme bilden heute die Grundlage für fast jede Form von Middleware. So stellt beispielsweise auch das SOAP-Protokoll (Simple Object Access Protocol), das eine Interaktion zwischen Web Services unterstützt, im Wesentlichen eine Möglichkeit dar, RPC-Aufrufe auf XML-Nachrichten abzubilden.

- **TP-Monitore:**
  TP-Monitore stellen die älteste Form von Middleware dar und gleichzeitig die stabilste und zuverlässigste. Ein TP-Monitor ist ein Steuerungsprogramm zur transaktionsorientierten Verwaltung von Anwendungsprogrammen, insbesondere OLTP-Anwendungen (Online Transaction Processing). Letztlich basiert auch ein TP-Monitor auf dem RPC, der hier jedoch **transaktionsgeschützt** abläuft. Ein so genannter **Transaktionaler RPC (TRPC)** gewährleistet die Einhaltung der ACID-Eigenschaften von Transaktionen, die entfernte Prozeduraufrufe tätigen.[15]

- **Objekt-Broker:**
  Rein implementierungstechnisch unterscheiden sich Objekt-Broker kaum von anderen RPC-basierten Systemen. Der wesentliche Fortschritt gegenüber dem traditionellen RPC, der auf den Programmierparadigmen imperativer Programmiersprachen beruht, ist der **Bezug zur objektorientierten Programmierung**. Objekt-Broker unterstützen den entfernten Zugriff auf Objekte. Als wichtigste Vertreter dieser Form der Middleware sind die auf COBRA (Common Object Request Broker Architecture) basierenden Objekt-Broker zu nennen.

15. Die hier beschriebene Middleware des TP-Monitors stellt eine 3-tier-Architektur dar, die auch als „TP-heavy" bezeichnet wird. Unter „TP-lite" versteht man eine 2-tier-Architektur, bei der die Anwendungslogik (zumindest zum Teil) in Form so genannter „stored procedures" auf der Seite der Ressourcen-Mgr angesiedelt wird.

Zusammenfassung - Vielfalt an Bezeichnungen (2)

- **Objekt-Monitore:**
  Der Trend hin zur objektorientierten Programmierung hat dazu geführt, dass die gut erprobte Funktionalität von TP-Monitoren, die zunächst für imperative Programmierumgebungen ausgelegt war, auch im Kontext verteilter objektorientierter Systeme gewünscht wurde. Objekt-Monitore sind somit das Resultat der **Konvergenz zwischen TP-Monitoren und Objekt-Brokern**. Im Wesentlichen handelt es sich dabei um **TP-Monitore mit objektorientierten Schnittstellen**.

- **Nachrichtenorientierte Middleware – Message-Oriented Middleware (MOM):**
  Eine nachrichtenorientierte Middleware ist speziell für die Unterstützung einer **asynchronen Kommunikation auf der Basis persistenter Warteschlangen** ausgerichtet. Viele Anwendungen erfordern keine synchrone Kommunikation, für diese wäre ein RPC also ein unnötiger Aufwand. Im Rahmen der umfassenden Funktionalität von TP-Monitoren wurden daher bereits asynchrone Kommunikationsmechanismen, wie der asynchrone RPC und persistente Warteschlagen, zur Verfügung gestellt. Durch die persistente Zwischenspeicherung der Nachrichten wird eine bessere Entkopplung von Sender und Empfänger erreicht, da die Nachricht auch dann noch sicher zugestellt werden kann, wenn der Empfänger zwischenzeitlich nicht erreichbar war. Eine nachrichtenorientierte Middleware stellt typischerweise verschiedene Operationen zum Umgang mit persistenten Warteschlangen zur Verfügung. Dazu gehört **insbesondere ein transaktionsgeschützter Zugriff auf Warteschlangen**.

  Eine besondere Form der asynchronen Kommunikation, die mit Hilfe persistenter Warteschlangen und MOM umgesetzt werden kann, stellen so genannte **Publish/Subscribe-Systeme** dar. Auch hierdurch wird eine Entkopplung von Sender und Empfänger erreicht: Ein Sender stellt Informationen ohne Angabe des Empfängers zur Verfügung (Publish), die dann von interessierten Abonnenten bezogen werden können (Subscribe).

Zusammenfassung - Vielfalt an Bezeichnungen (3)

- **Message Broker**

  stellen eine spezielle Form der MOM dar. Die Besonderheit hierbei ist, dass Nachrichten nicht nur einfach vermittelt werden, sondern dass der **Message Broker zusätzlich die Möglichkeit bietet, Nachrichten zu filtern und gemäß vorgegebener Regeln zu transformieren**. Auf diese Weise können beispielsweise die Empfänger einer Nachricht dynamisch aufgrund des Inhalts der Nachricht bestimmt werden. Die Möglichkeit zur Transformation ist hilfreich zur Abbildung verschiedener Datenformate auf Sender und Empfängerseite. Diese Form der Middleware lässt sich im Zusammenhang mit der Integration heterogener IT-Systeme hilfreich einsetzen.

- **Application Server**

  unterscheiden sich in ihrer Kernfunktionalität nicht wesentlich von den bereits vorgestellten Middleware-Ansätzen. Der **Hauptunterschied besteht darin, dass Application Server darauf ausgerichtet sind, das Web als Zugriffskanal für die implementierten Middleware-Dienste nutzbar** zu machen. Damit verbunden ist vor allem eine Ausrichtung auf den dokumentenorientierten Informationsaustausch mittels HTTP (HyperText Transfer Protocol). Zur Ergänzung herkömmlicher Middleware-Plattformen benötigen Application Server daher vor allem Mechanismen zur **dynamischen Erzeugung und Verwaltung von Hypertext-Dokumenten**. Damit wird die nunmehr Web-orientierte Präsentationsebene wieder mit der Anwendungsebene verknüpft. Als Programmierschnittstellen (API) für Application Server haben sich im Wesentlichen zwei Ansätze herausgebildet: der von SUN propagierte Java-basierte Ansatz J2EE und der von Microsoft propagierte Ansatz .Net. Über diese APIs werden vielfältige Middleware-Dienste, die von TP-Monitoren, Objekt-Brokern und nachrichtenbasierter Middleware bereits bekannt sind, im Rahmen einer einheitlichen Schnittstelle für Anwendungsprogrammierer verfügbar gemacht.

Zusammenfassung - Vielfalt an Bezeichnungen (4)

- **Workflow-Management-Systeme (WfMS)**

  haben ihren Ursprung in der Büroautomation, wo es darum ging, gut strukturierbare Abläufe, die bisher auf Papierbasis stattfanden, durch den Einsatz der IT zu automatisieren. Viele der administrativen Workflows, die zunächst im Fokus waren, werden heute im Rahmen von Dokumenten-Management-Systemen (DMS) umgesetzt. DMS bilden damit letztlich eine spezielle Form von WfMS. Ein wesentliches Ziel der WfMS ist die Durchgängigkeit von der Geschäftsprozessmodellierung bis hin zur Ausführung von Workflows. Verschiedene Methoden zur Spezifikation von Workflows lassen sich unterscheiden: **Kommunikationsbasierte Methoden** sind in erster Linie für die Spezifikation von Dialogen mit menschlichen Akteuren geeignet. **Aktivitätsorientierte Methoden** betrachten die im Rahmen eines Geschäftsprozesses zu leistende Arbeit, indem sie sich auf die Aktivitäten und die Abhängigkeiten zwischen ihnen konzentrieren (z. B. Petri-Netze, Ereignis-Prozessketten (EPK)). **Entitätenbasierte Methoden** fokussieren auf die Modellierung von Objekten, die während des Anwendungsprozesses erzeugt, bearbeitet und verbraucht werden (z. B. UML). Die meisten WfMS basieren allerdings – zumindest hinsichtlich ihrer Benutzerschnittstelle – auf gerichteten Graphen (aktivitätsorientierten Modellierungsmethoden). Der Unterschied zu herkömmlichen Programmiersprachen besteht vor allem darin, dass WfMS langlaufende Abläufe mit vielen verschiedenen Beteiligten in einer verteilten Ausführungsumgebung unterstützen müssen. Das hat vor allem zur Folge, das WfMS komplexe Techniken zur Fehlerbehandlung vorzusehen haben, die für die Abwicklung kurzer Transaktionen nicht nötig wären. Für die Ausführung von Workflows ist eine „Workflow Engine“ zuständig. Diese verwaltet Instanzen von Workflows, die im Sinne eines Schedulers schrittweise abgearbeitet werden: Die Workflow Engine bestimmt, welche ausführenden Komponenten im Rahmen des Workflows wann aufgerufen werden. Mit zunehmender Reife der WfMS und mit dem wachsenden Bedarf nach durchgängig IT-unterstützten Geschäftsprozessen wurden WfMS immer mehr auch als Werkzeuge zur Integration heterogener Anwendungssysteme angesehen. WfMS bilden damit Werkzeuge zur Unterstützung der „**Programmierung im Großen**“. Die Idee besteht darin, große Software-Module als Komponenten in die Spezifikation einer übergreifenden Geschäftsprozesslogik einzubinden. Dies erfordert allerdings eine Kombination mit herkömmlichen Middleware-Plattformen, die notwendig sind, um einen transparenten Zugriff auf heterogene Komponenten zu ermöglichen.