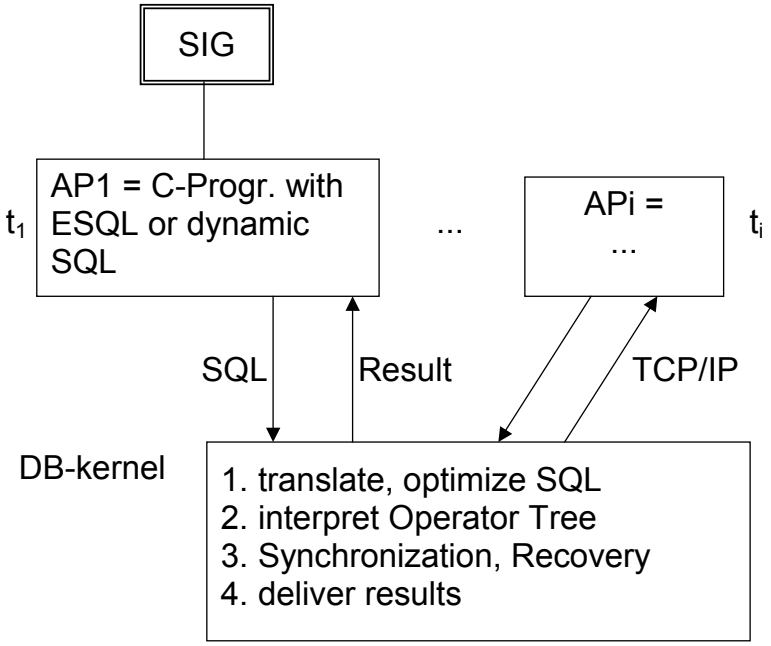
Chapter 7 Transaction-Processing Monitor

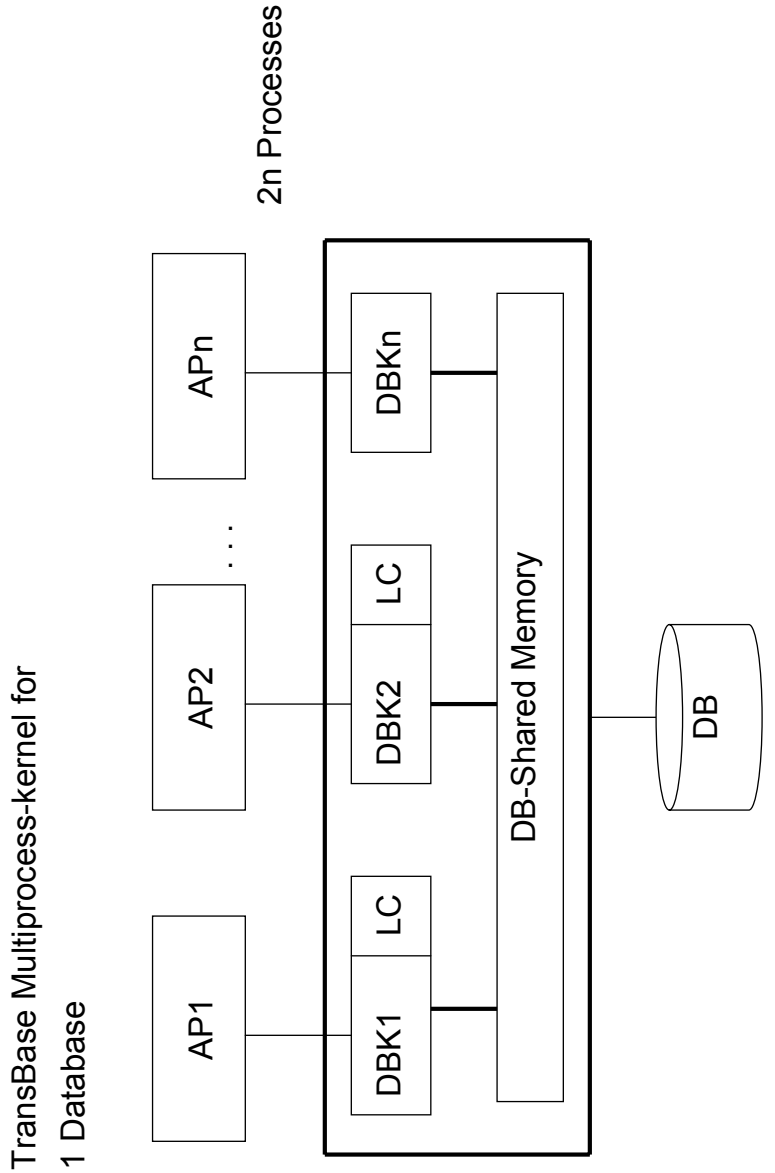
Chapter 7.1 Process-Architecture

typical for UNIX-Workstations



UNIX-world: 1 DB-kernel per APi

Mainframe: 1 DB-kernel per CPU or
per DB



Adaptions

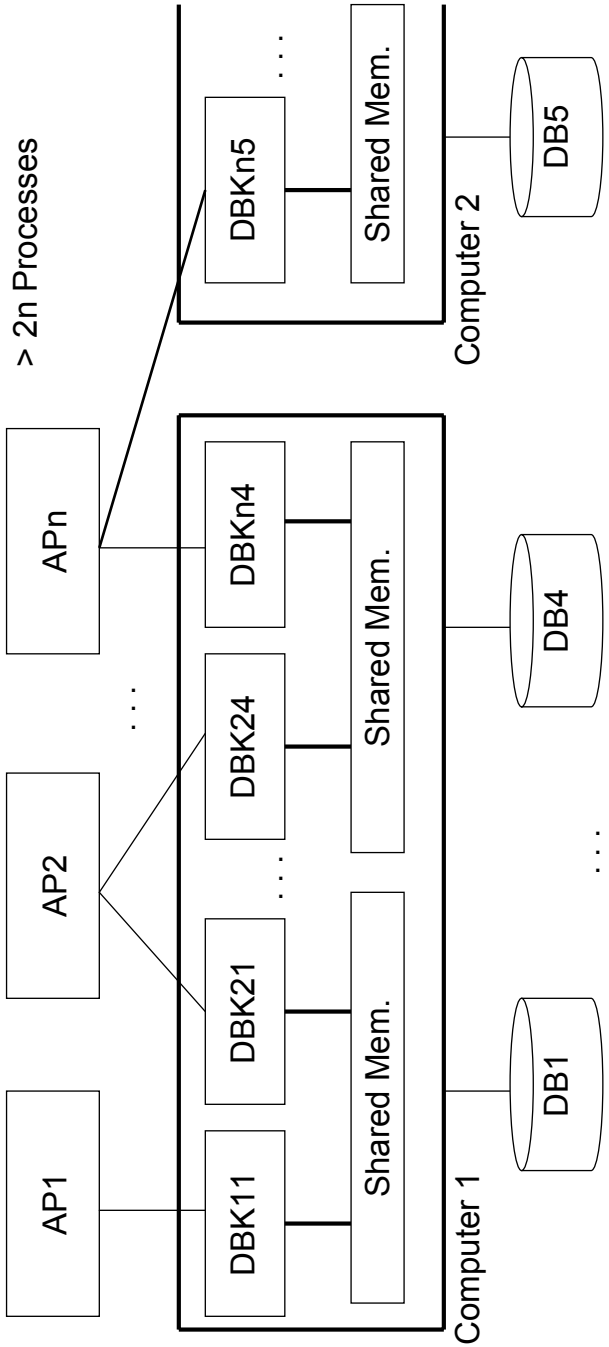
1. with Shared Memory:

   - Tuning: page size, cache size
   - Cache-management
   - Cache-performance

2. without Shared Memory:

   - message-system
   - providing data for DB-kernels

TransBase Multiprocess-kernels for Multi-Databases



Summary: For every APi, user, transaction, SIG
2 processes

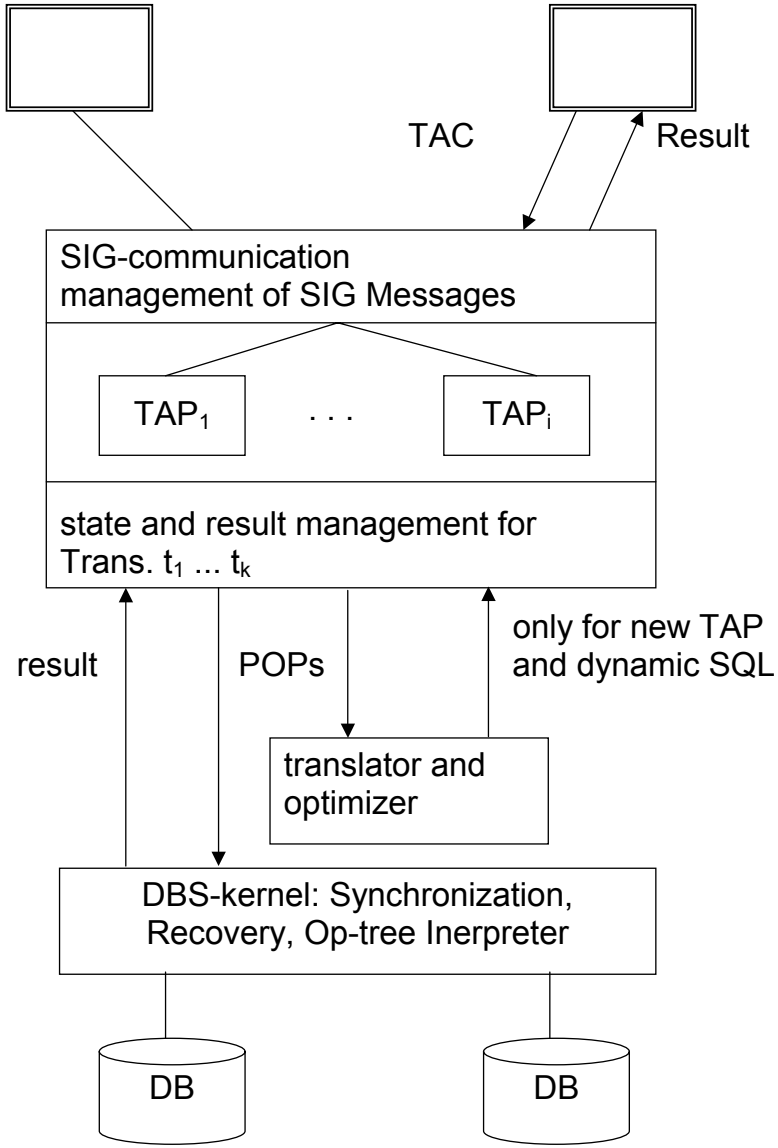
Operating Systems today:

1. Few processors, i. e. little true parallelism
2. Precess generation expensive: several 10^3 operations
3. Code-Sharing?
4. APi with much overhead, e. g.
   VTAM linked
   Process communication linked, etc.
   message handling: SIG $\leftrightarrow$ API
   Generation and management of interaction masks
   ...

⇨ All APi require a lot of similar overhead

Goal: 1 TP-Monitor for common, similar work.
APi trimmed down to TAPi

Architecture or a TP-Monitor



Tasks of a TP-Monitor

1. Select and start TAP_i based on Transaction-Code TAC_i
2. Output mask M_i on SIG_j for input message mes_{ij}
3. check and delivery of mes_{ij} to TAP_i
4. manage storage S_{ij} for data D_{ij} for communication between SIG_j and TAP_i
5. manage run time context for transaction t_i
6. save messages between TAP_i and SIG_j (e.g. = ATM)
   automatic teller machine
   "secure message queues"
7. ect. user and rights management

Summary: TP-Monitor ~ special BS for TAP

Variants of TP-Monitors

Terminology: dialogue step: from SIG-Input to SIG-output (program for 1 dialog step DS)

Simple TAPs: TAP is exactly 1 DS (e.g. money transfer), but i.g. several SQL-statements E-TAP

Complex TAPs: sequence of DS, e.g. booking a trip: flights, hotels, ... C-TAP

Server: incarnation of a TP-M

Task: execution of a TAP

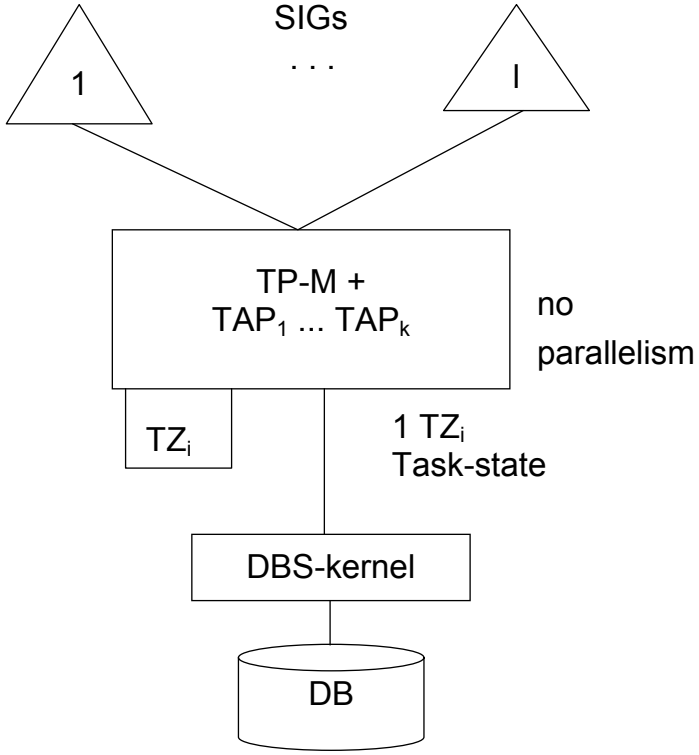
Variants:	# Server,	#Tasks
	1	1
	n	1
	1	n
	n	m

1. Single Server, Single Task

1 Server + 1 acitve Task (started)

- no synchronization
- message-management TP-M ↔ SIG
  asynchronously and in parallel
  for several messages, but only 1 active task

Question: commit and issueing messages?
e.g. payment at ATM?

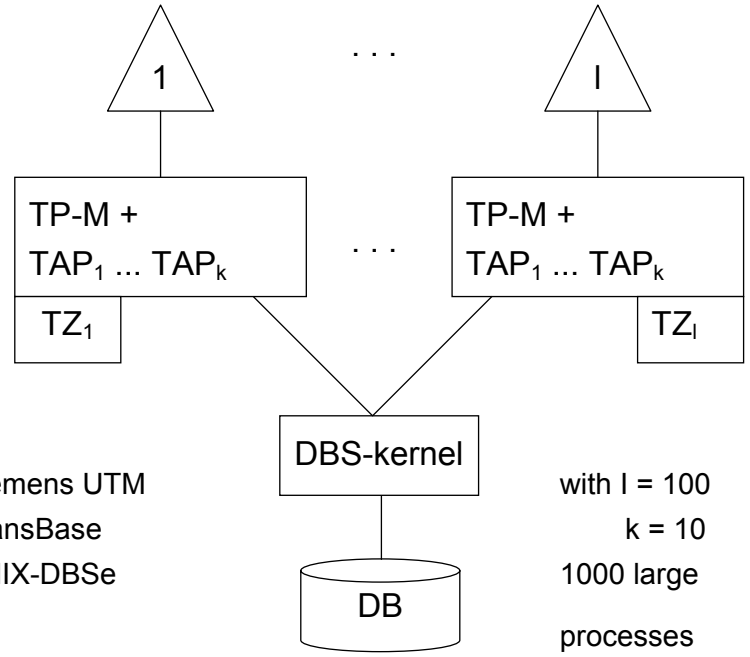


2. Multi-Server, Single Task

Several incarnations ot the TP-M, per TP-M one Task,
e.g. fixed assignment of SIG_i to TP-M_j.
Transaction t_i started from SIG_i, executed and managed via TP-M_j.

⇨ - every TP-M knows all TAPs
- scheduling by BS
- many process switches

better: dyn. assignment of SIG_i to TP-M_j depending on TAP-type, i.e. TP-M_j knows only some TAPs.
Number of TP-M depends on load!



Siemens UTM
TransBase
UNIX-DBSe

with I = 100
k = 10
1000 large
processes

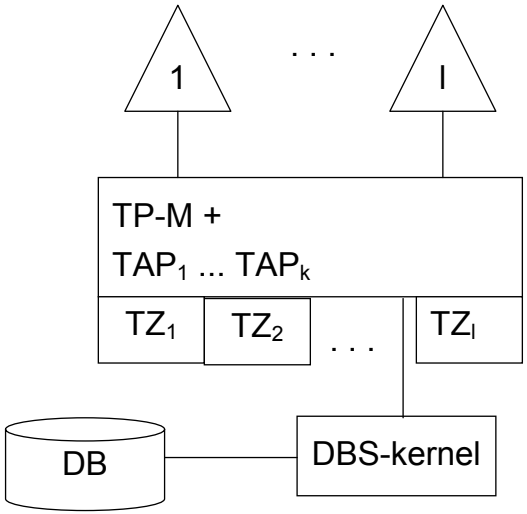
3. Single Server, Multi-Task

very similar to light weight processes, threads
1 TP-M manages many Tasks, TZ_i

⇨ - Task-Management
- TP-M schedules Tasks
- storage management for TZ_i
- "Interrupt"-treatment because of asynchronous cooperation with DBS and SIGs

⇨ TP-M becomes BS' within BS

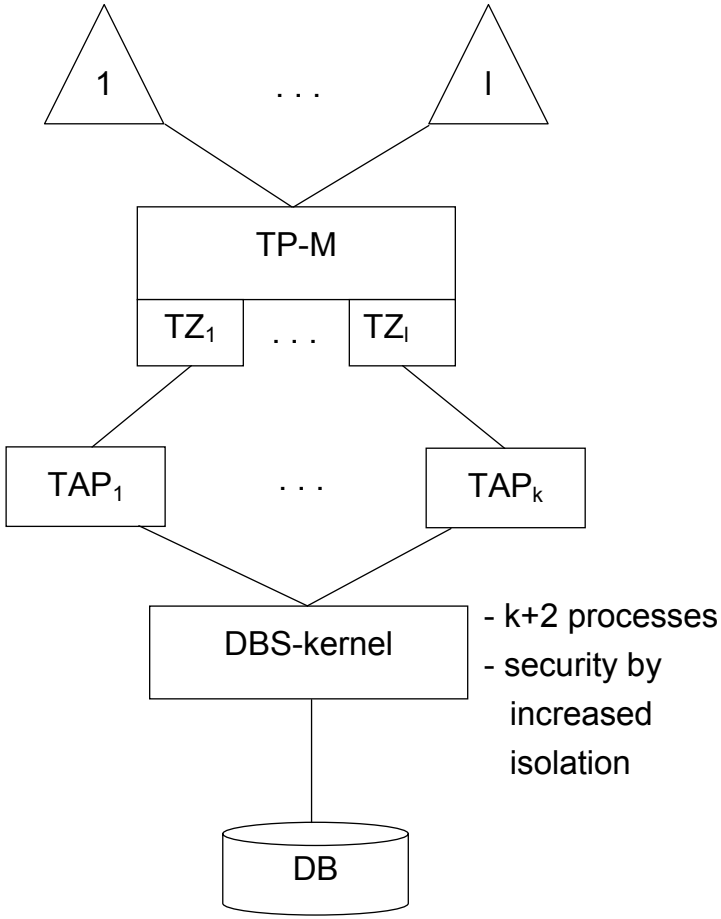
Advantages: few processes
few process switches
typical for main frames!



suitable for
1-processor-
systems

IBM: CICS/IMS

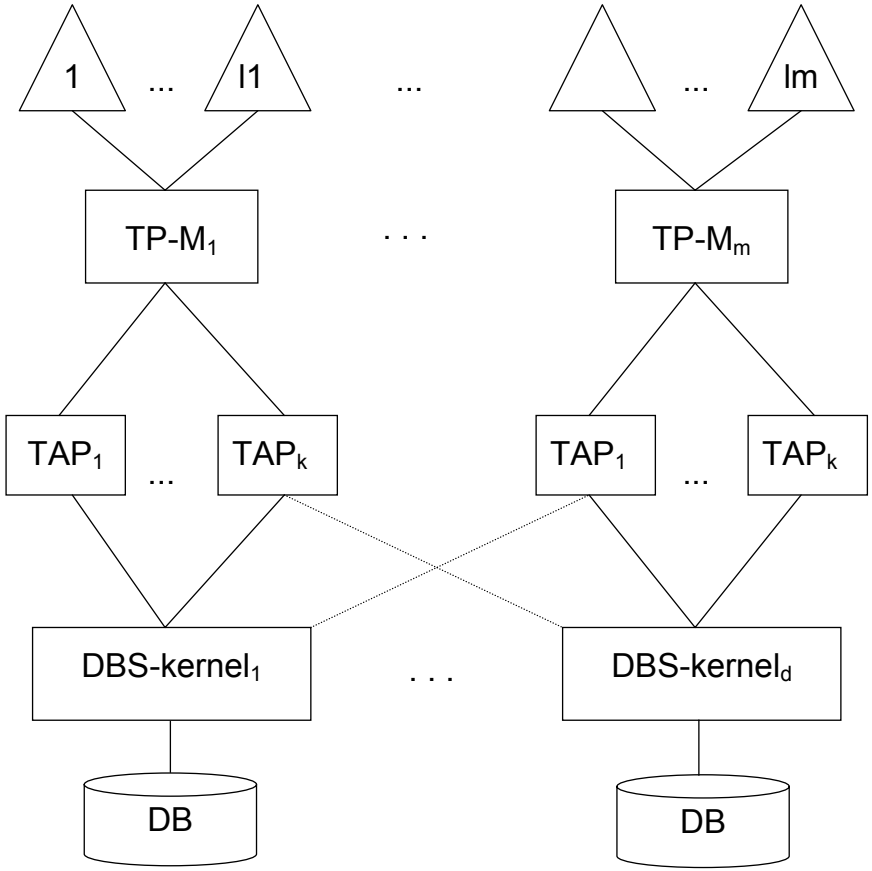
with TASs factored out:



- k+2 processes
- security by
increased
isolation

- performance improvement: time-slicing by BS
- suitable for multiprocessors
- TZ_i as shared-memory or task queues
- several incarnations of TAP_i, depending on load

4. Mulit-Server, Multi-Task



- several TP-M via front end processor
- free map Processes → Processors
- Client / Server Architectures
- TZ_i implemented via shared-memory or communication
- for multiprocessor- and multicomputer-systems

7.1.15

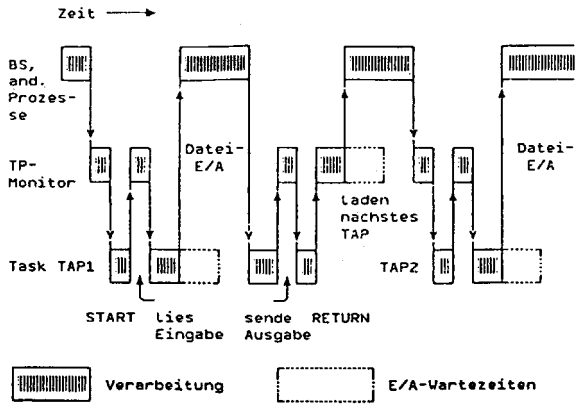


Abb. 9. Single-Process/Single-Tasking: Kontrollfluß und Aufteilung der Rechenzeit

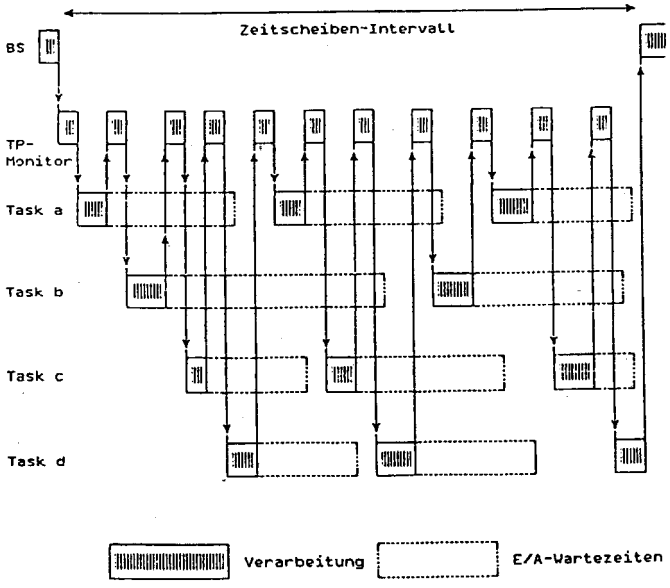
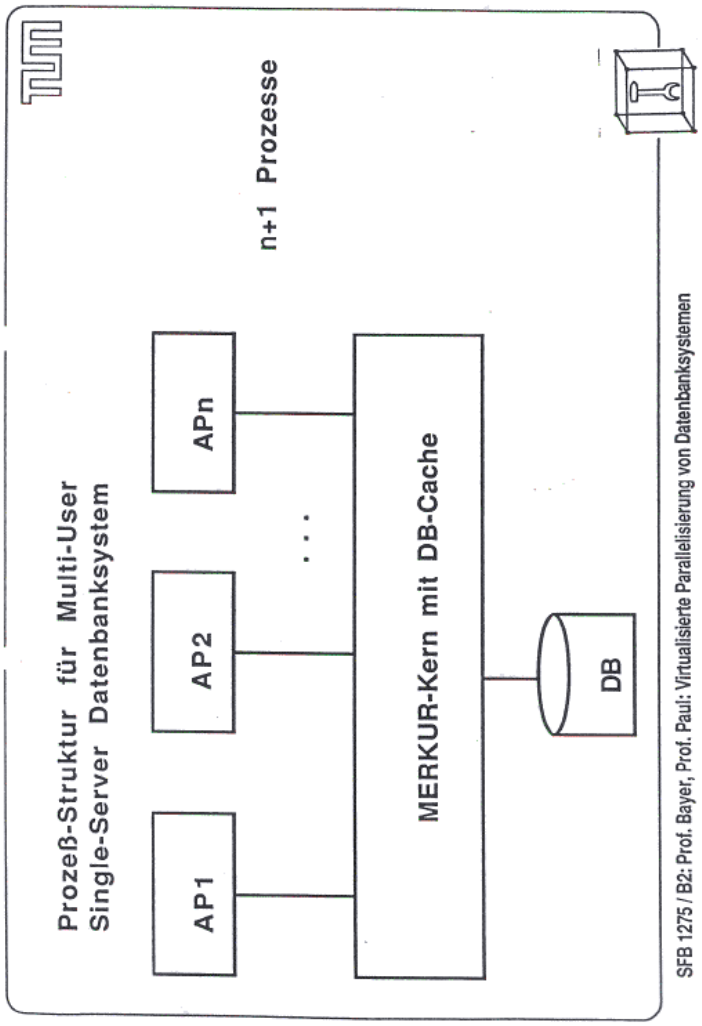


Abb. 10. Single-Process/Multi-Tasking: Kontrollfluß und Aufteilung der Rechenzeit

3a



Chapter 7.2 TP-monitor Internals

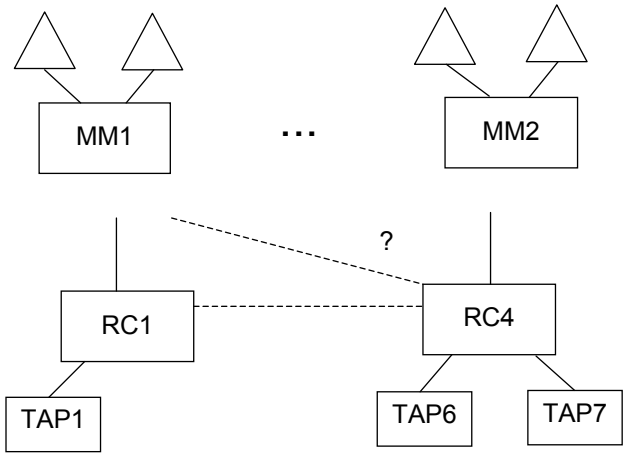
7.2.1 Message Manager

1. support different SIGs
2. Masks for every TAP_i
   today generic, stored in DB
   adaptive for special DB-schema
   e.g. OMNIS-Archiver
3. Plausibility checks for field inputs
   (in Client)
4. transform filled masks into message –
   Formats with Header: e.g. (TAP-Id, Method, list of parameters)
5. Standardized message → TAP via Requ.-Control (asynchromous)
6. Reply from TAP → response mask
7. Output mask → SIG, Client
8. Security: user rights, modified masks

7.2.2 Request Control

Tasks: TAP-Code (= TAP – ID) = TAC

Table: TAC → TAP-incarnation
i.g. several incarnations for fault-tolerance and load-performance (TANDEM!)



- message routing
- global name-service, e.g. NIS
  for TAC → RC → TAP ?

1. Connection oriented message exchange:

- establish connection:
  check process - ID
  buffer for messages
- sequence of messages
- disconnect

e.g. IBM/CICS with SNA LU 6.2
and half duplex
WWW connectionless, i.e. t consisting of several dialogue-steps is difficult
Example: 1 Session per day and bank teller (counter)
Problem: for Application-Programmer:
- procedure-call within process
- message exchange between processes
⇒ RPC

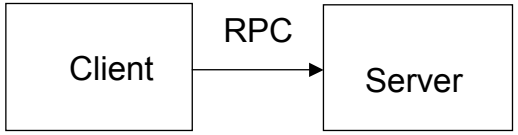
2. Remote Procedure Call

Messages : synchronous or asynchronous

depending on sender

RPC : sysnchronous like procedure-call

asynchronous message exchange can be simulated by RPC



- wait
- new process
- new thread

⇒ i.e. RPC is an abstraction from single- task and multi-task servers

- hides locality of server (fault-tolerance, load balancing)
- hides differences in languages, „Stub“ converts parameters into expected format, buffers calls, etc.

7.2.2

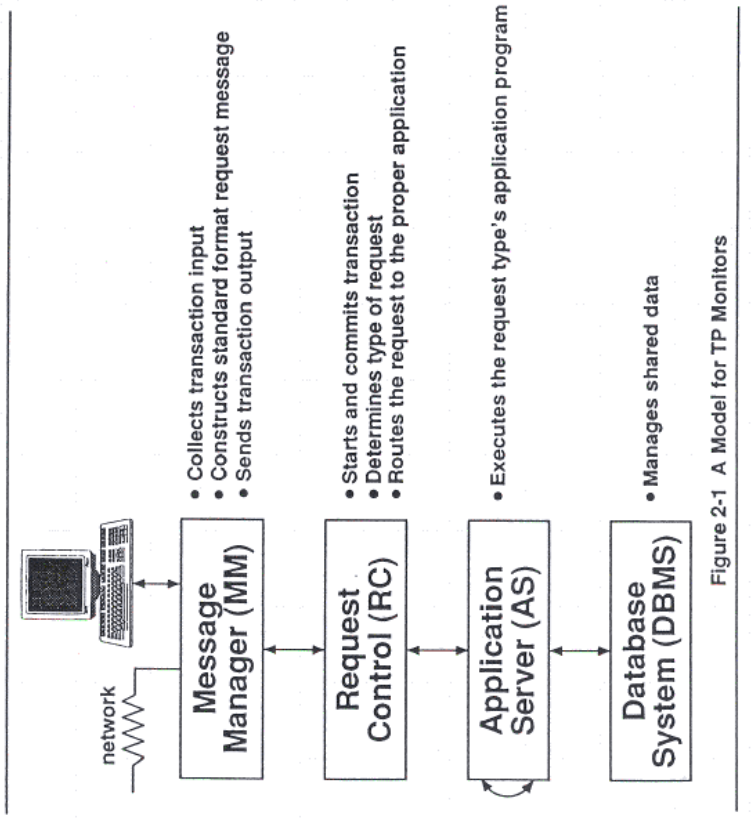


Figure 2-1 A Model for TP Monitors

7.2.3 Further Aspects

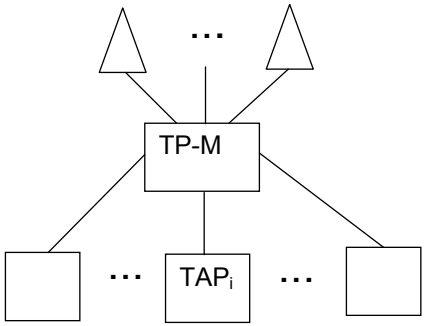
Transmission of large result sets:

- complete file
- by packets

Efficiency:

- local procedure call : ~ 50 instructions
- RPC : 1.000 – 10.000 instructions
- Research topic : cheap RPC ?

TAP-Classes:



Multitasking: very complex

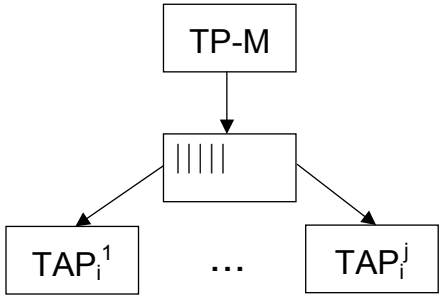
Should be single-tasking
⇒ synchronous calls

⇒TAP_i may block, several TAP_i-incarnations?

solution: TAP-classes with

- common request-queue
- TAPi-incarnations according to demand

⇒ i.e. single TAP as single-task server, fetches requests from queue
TP-M starts and terminates TAP according to request load



Request queue in common store

- Mangement of queues
- e.g. persistent queues via DBS
- Failure of a single TAP_i not critical