4. High-Performance Transaction Systems (HTPS)

Theo Härder
www.haerder.de

Shared Nothing vs. Shared Disk

Performance, availability, extensibility

Realization problems

Replication

Query optimization

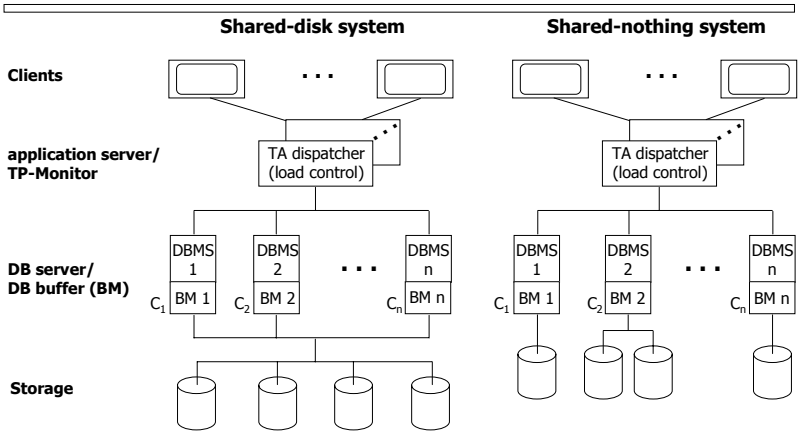
Measures of transaction processing

The Benchmark Success

Main reference:
Erhard Rahm: Hochleistungs-Transaktionssysteme, Vieweg, 1993

Trends in DBS

Overall Architectures of MC-DBMS

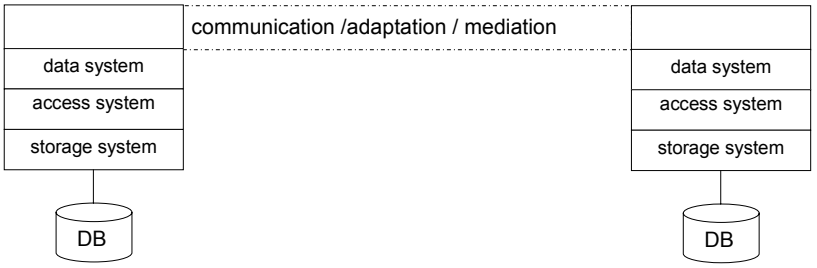


Configuration:

- number of computers ?
- broadband communication / local placement
- no "single point of failure"
- (copy of) DB system on each computer

Layer Model for Distributed/Parallel DBMS Processing

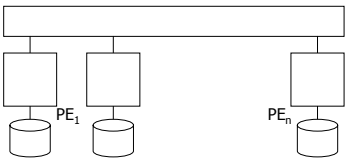
- Each node
  - runs a copy of the DBMS
    - typically, a process is unit of scheduling, addressing, and protection
    - single/multi process
    - singe/multi tasking within a process
  - various models of data access
    - shared nothing, shared disk, shared everything
    - here: SN and SD architectures



SN vs. SD: Potential of Performance

- **Strong dependencies to**
  - application characteristics and
  - realization of single system functions

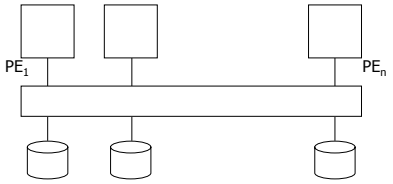
- **Shared Nothing**



  - static data partitioning determines execution of DB-Ops
  - little opportunities for load balancing or saving of communication
  - **problematic**: "dominating" transaction types and DB partitions

SN vs. SD: Potential of Performance

- **Shared Disk**



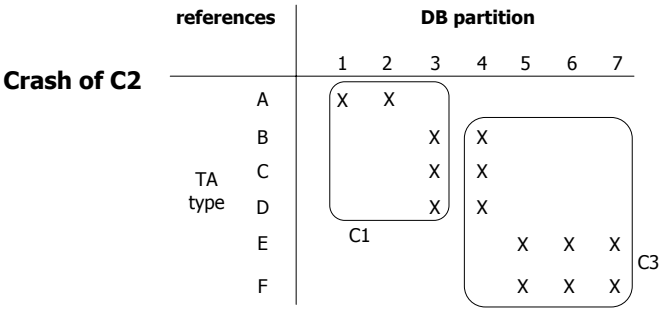
  - local accessibility of all data:
    ➔ more flexible means for load balancing
  - communication for concurrency control and coherency control (dealing with buffer invalidations)
  - close coupling can be utilized for performance improvement
  - higher flexibility to be used for increased parallelism

The "Ideal" Workload

references		DB partition						
		1	2	3	4	5	6	7
TA type	A	X	X					
	B	C1		X	X	C2		
	C			X	X			
	D			X	X			
	E					X	X	X
	F					X	X	X
								C3

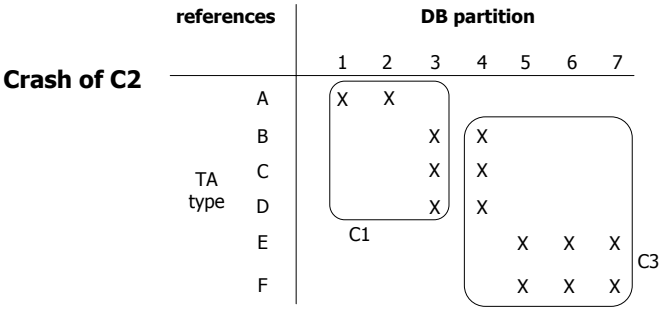
- **Partitioning of workload and data**
  - allocation of TA types to disjoint data areas
  - static workload
  - uniform distribution of workload (amount of processing, time)
- **Delightful transactions**
  - local processing of all transactions (e.g. DebitCredit)
  - few problems of concurrency control

SN vs. SD: Availability



- **Shared nothing**
  - partition of a failed computer is not accessible anymore
  - preparation of a take-over of the affected partition by another computer (danger of overloading)
  - recovery by the adopting computer
  - replication enables fast recovery in case of a catastrophe

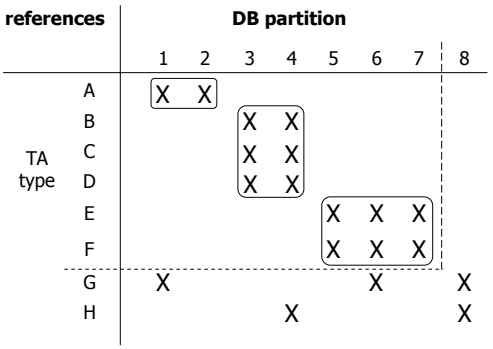
SN vs. SD: Availability (2)



- **Shared disk**
  - entire DB remains accessible after crash of a single computer; each single computer or all together can take over the workload of the failed computer
  - complex crash recovery
  - creation of a global log file

SN vs. SD: Extensibility

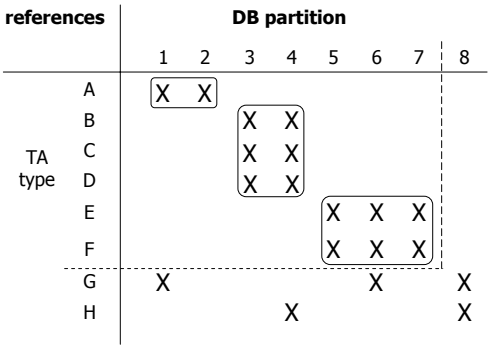
growth of TA types and DB partitions



- **Shared nothing**
  - new engine requires physical repartitioning of the DB ($N \rightarrow N+1$)
  - simple attachment of storage (disks)
  - especially problematic for non-relational DBMS

SN vs. SD: Extensibility (2)

growth of TA types and DB partitions

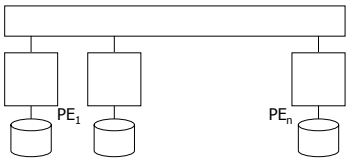


- **Shared disk**
  - no physical (re-) partitioning of the DB
  - adding engines does not affect DB schema and programs
  - direct attachment of disks can limit the number of engines

    ➔ "message-based" I/O interface

SN vs. SD: Realization Problems

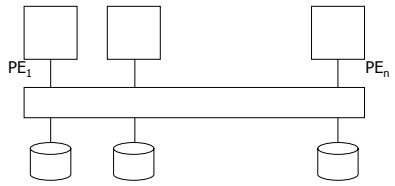
- **Shared nothing**



  - determination of the physical DB partitioning (fragmentation + allocation)
  - **distributed query processing (optimization)**
  - **management of replicated databases**
  - distributed Commit protocol
  - global deadlocks (detection + resolution)
  - load distribution, -balancing
  - administration
  - special problems in remotely distributed systems (network partitioning, node autonomy, ...)

SN vs. SD: Realization Problems (2)

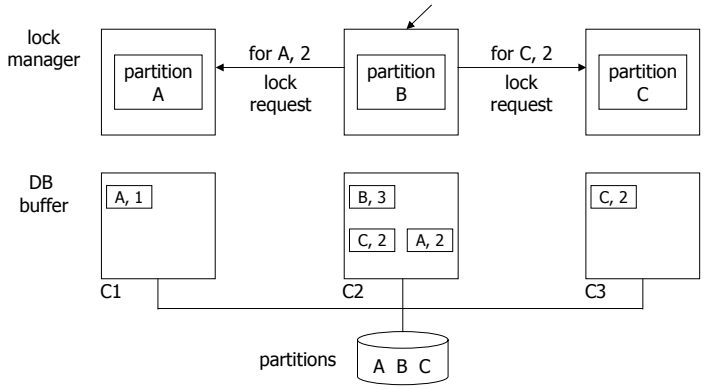
- **Shared disk**



  - **concurrency control**
  - global deadlocks (detection + resolution)
  - **DB buffer management, coherency control**
  - **logging, recovery**
  - load distribution, -balancing
  - parallel query processing
  - administration

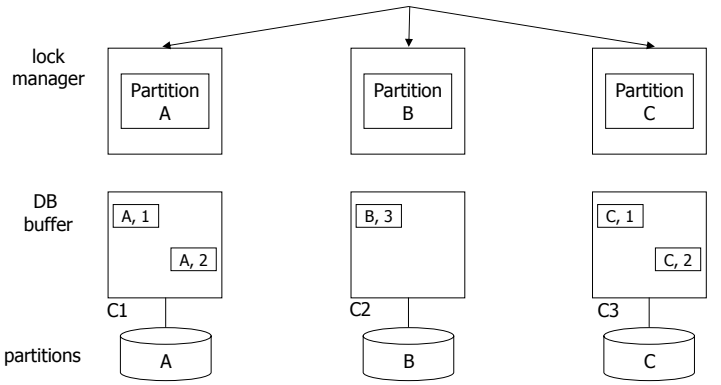
Concurrency Control

- **Shared disk**
  - logical and therefore dynamic allocation of DB partitions and responsibility for concurrency control
  - allocation of example TA to C2



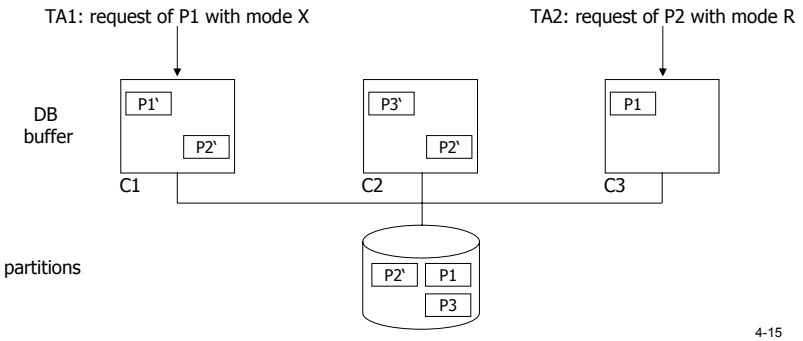
Concurrency Control (2)

- **Shared nothing**
  - static allocation of DB partitions and responsibility for concurrency control
  - shipping of operations
  - partition-wise locking responsibility (as in the centralized case)



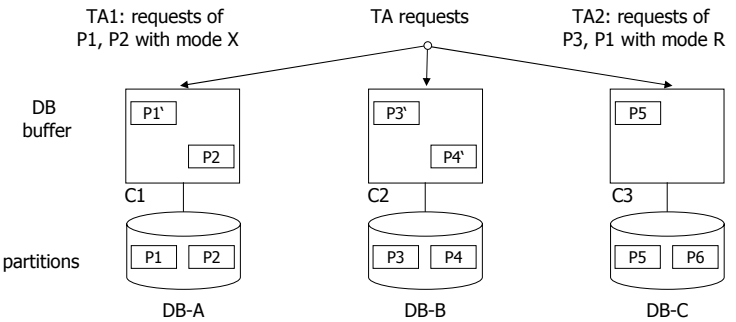
DB Buffer Management

- **Shared disk**
  - assignment of new transactions: affinity problem
  - local processing of DML operations
  - processing principle : **data shipping to the executing engine**
  - coherency control required



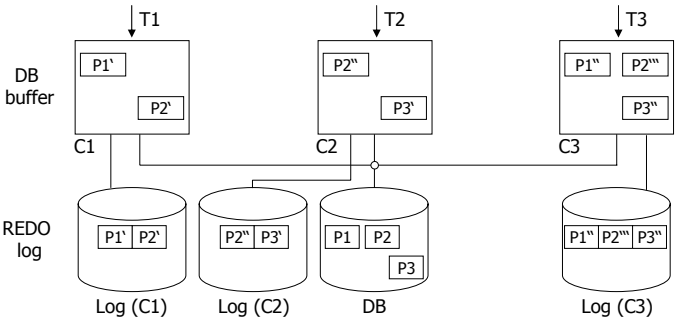
DB Buffer Management (2)

- **Shared nothing**
  - as in the centralized DBMS → no copies
  - processing principle: **the workload follows the data**
  - **partition-wise processing – optimal workload allocation critical! assume:**
    - TA1 is dispatched to C1
    - TA2 is dispatched to C3



Logging

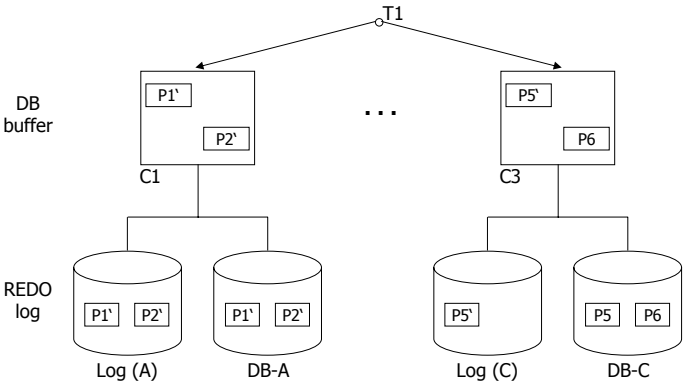
- **Shared disk**



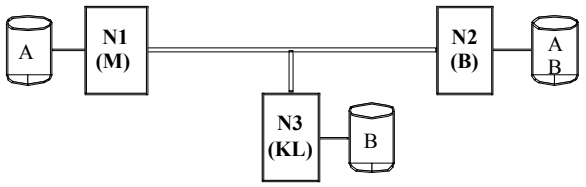
- T1 < T2 < T3
- creation of a global log (merging) possibly at restart because of media failures

Logging

- **Shared nothing**
  - partition-wise (as in the centralized DBMS)
  - log files of a transaction can be distributed

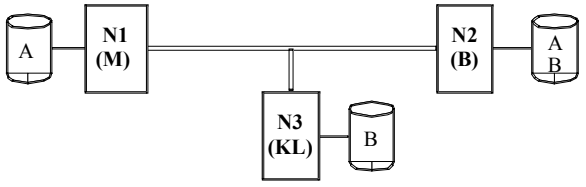


Replication



- **Data allocation**
  - partitioned
  - partial/full replication
  - distinction: caching vs. replication!
- **Units of replication**
  - tables, indexes
  - metadata
  - fragments (vertical/horizontal decomposition of tables, etc.)
    - by predicates
    - user-guided decomposition
  - DB procedures/operations
- **Most important application: distributed and parallel DBs**

Replication



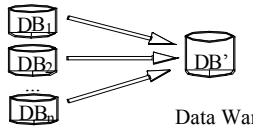
- **Pros**
  - increased **availability** of data (masking and tolerating failures)
  - **acceleration** of read accesses
    (improved response times, saving of messages)
  - enhanced potential of **load balancing** and query optimization
- **Cons**
  - high **update overhead**
  - larger **storage consumption**
  - increased system **complexity**
    - flexible update algorithm
    - extended requirements of concurrency control (1-copy serializability)
    - recovery problems, especially in case of network partitions
    - query optimization

Replication – Application Areas

- **Replicated catalog data** (metadata)
- **Catastrophe recovery**

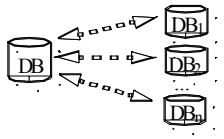
not decay-related to enable take-over of entire processing load

- **Data Warehousing:**
  storage of transformed data in an own DB for decision support



Data Warehouse

- **Mobile Computing and Internet:**
  DB partitions, files ... on notebook, PDA, mirrored sites, indexing of search engines, etc.



Mobile Computing

Replication and Serializability

- **Serializability in case of a replica-free DB**
  - a schedule S is called serializable, if there exists at least one serial schedule of the same set of TAs obtaining the same DB state and creating the same output as S.
- **Definition: 1-copy serializability**
  - a schedule S of a replicated DB is called 1-copy serializable, if there exists at least one serial schedule of the same set of TAs on a replica-free DB creating the same output and obtaining the same DB state as S on the replicated DB.

  → **synchronous update** of all copies
- **Weaker consistency criterion**
  - is sometimes desired to achieve higher availability or throughput
  - it is not necessary to (synchronously) update all copies at a time
  - but the DB again must converge to a consistent state

  → **epsilon serializability** (ε-serializability)

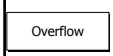
Replication and Serializability (2)

- **Epsilon serializability**
  - a schedule S on a replicated DB is called ε-serializable, if
    1. the schedule S_U = S \ {reader TAs} is serializable.
    2. for every reader TA holds:
       as long as the values read deviate less than ε from the up-to-date (consistent) values, the actions of a reader TA may arbitrarily overlap with the actions of other conflicting TAs.
- The overlap limit is sometimes not defined via a value interval (as weighted overlap parameter ε), but via the number of updates applied.
- e**-serializability enables limited compatibility (0)** between the accesses of a reader TA (R_Q) and a writer TA W_U:

	R_Q^j	R_U^j	W_U^j
R_Q^i	+	+	0
R_U^i	+	+	-
W_U^i	0	-	-

standard compatibility

0: conflicts with writer TAs are ignored until the ε-limit is reached



Replication and Serializability (3)

- **Start state with accounts k_1 and k_2**

A: k_1=1000, k_2=1000 B: k_1=1000

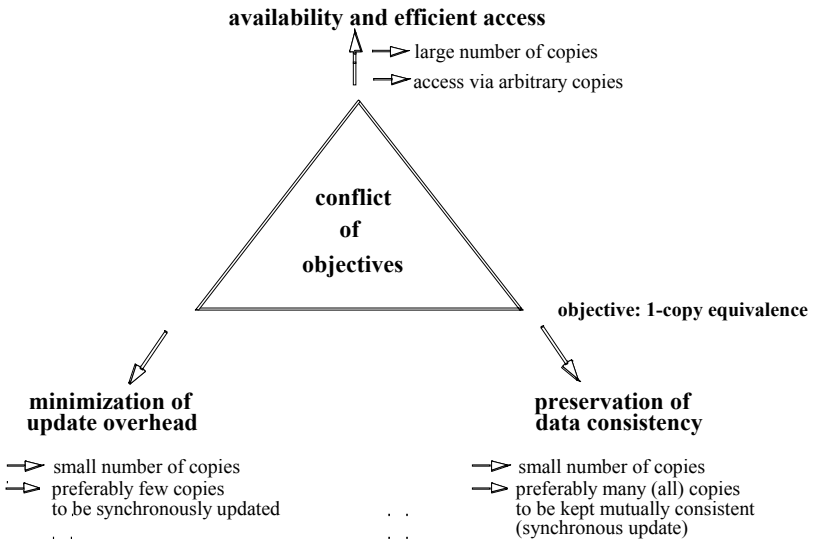
- **Transactions**
  1. T_U transfers 200 Euro from k_1 to k_2:
     $w^A_{TU}(k_1, -200)$, $w^B_{TU}(k_1, -200)$, $w^A_{TU}(k_2, +200)$
  2. T_Q is initiated at node B at the same time to compute the sum of all accounts (for statistical purpose):
     $r^B_{TQ}(k_1)$, $r^A_{TQ}(k_2)$
- **Possible schedules at nodes A and B**
  - $S^A = (w^A_{TU}(k_1, -200), w^A_{TU}(k_2, +200), r^A_{TQ}(k_2))$
  - $S^B = (r^B_{TQ}(k_1), w^B_{TU}(k_1, -200))$

  → global schedule S = (S^A, S^B) is not 1-copy serializable!
- Final state

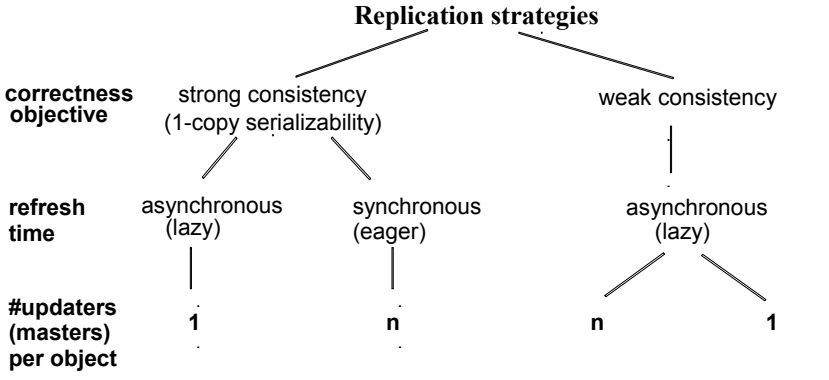
A: k_1= 800, k_2=1200 B: k_1=800

→ if a weighted ε-overlap parameter > 200 Euro is chosen (or an ε-overlap of more than 1 TA), then S = (S^A, S^B) is a correct ε-serializable schedule

Replication Control

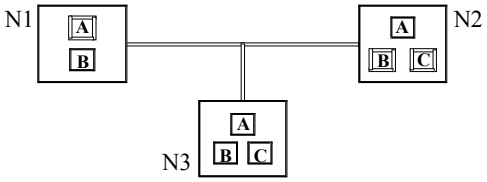


Classification of Replication Strategies



Primary Copy Method

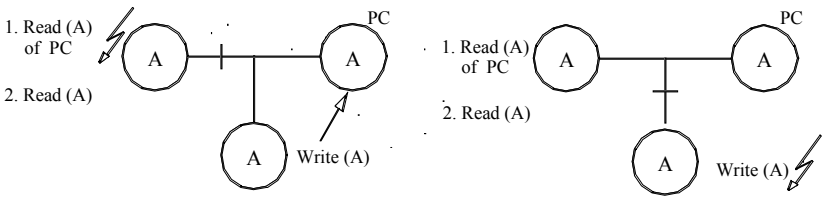
- **Asymmetric method**
  - 1 primary copy, n secondary copies per object
  - object owns version number



- **Synchronous update only for primary copy**
  - management of write and read locks at the primary copy node (publisher)
  - **delayed/asynchronous update** of secondary copies (subscriber) through primary copy node
  - read: primary copy node **communicates version number** of up-to-date copy when granting the read lock; read of a local (up-to-date) copy
- **Alternatives for read accesses**
  - read of local copy without lock request at the primary copy node (potentially outdated)
  - read of the primary copy (replication is not used!)

Primary Copy Method (2)

- **Network partitioning**
  updates can be performed only in the partition containing the primary copy



- **Failure of the primary copy node**
  - static approach: no further write operations possible until primary copy node active again
  - dynamic approach:
    determination of a new primary copy node
    - if network partitioning, DB processing can be continued only in one partition (e.g. where majority of copies exists)
    - new primary copy node must bring its copy up to scratch, if necessary ("pending updates")

Read-One-Copy

- **Synchronous update**
  - readers always refer to the up-to-date object state
  - methods of this class can be seen as special cases of voting (n=1)

1. Write-All / Read-Any method (Read-One / Write-All, ROWA)

- preferred processing of read accesses
  - read of the closest (local) replica
  - enhanced availability for readers
- very high cost for writers
  - write locks have to be acquired from all nodes
  - propagation of updates by 2PC protocol
- availability problem
  - updates only when all nodes carrying copies are accessible
  - node failure/partitioning can affect write availability

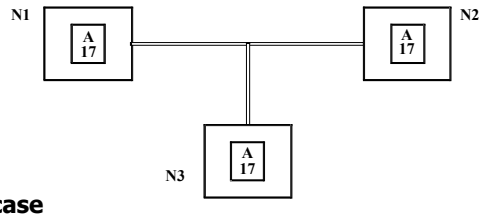
2. Write-All-Available variant

- only available replicas are updated
  - failed nodes have to integrate updates at restart
  - does not work in case of network partitioning

3. Write-One / Read-All ?

Voting

- **"Democratic" approach**
  - synchronization of accesses requires voting
  - base method: Majority Consensus
- **Properties**
  - read or write of an object requires access to the majority of replicas
  - each replica can be simultaneously read by several TAs, however, updated only by a single TA
  - determination of the up-to-date object state using version numbers



- **Failure case**
  continuation of processing as long as majority of replicas reachable
- **Problems**
  - high communication costs for read and write accesses
  - unsuitable for n=2 ("Read All, Write All")

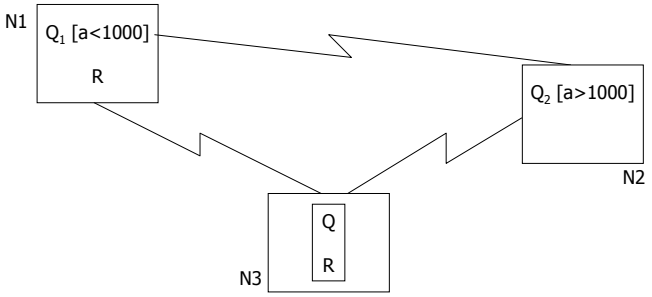
Voting – Weighted Voting, Quorum Consensus (2)

- **Each copy obtains a certain number of votes**
- **Protocol**
  - read requires R votes (read quorum)
  - write requires W votes (write quorum)
  - write/read overlap rule: R + W > V (= sum of all votes)
  - write/write overlap rule: W > V / 2
- **Properties**
  - simultaneous read and write not possible
  - each access to R (W) copies contains at least **one up-to-date copy**
  - determination of V, R and W allows trade-off between read and write costs as well as between performance and availability
  - other methods result as special cases
- **Example 1**
  - 5 copies with 1 vote each
  - R=1, W=5 → Write All, Read Any
  - R=3, W=3 → Majority Consensus
- **Example 2**
  - 5 copies
  - 4 copies without vote, 1 copy with 1 vote
  - R=1, W=1 → primary copy method

Publish/Subscribe Replication Models

- **Snapshot**
  - creation of a materialized view
  - **not up-to-date**, in general
  - typically only read access; no synchronization conflicts
  - replication usually not transparent for the user
- **Merge Replication: several updaters per replica**
  - use of triggers to propagate changes
  - conflict detection at record or attribute level (Merge Agent)
  - user-defined conflict resolution strategies possible
- **Master/Slave approaches with reading subscribers**
  - Snapshot Replication:
    data is distributed to subscribers at certain points in time
    (push- vs. pull-subscribers)
  - Transactional Replication:
    on a transaction basis, log data for updates are transferred to replica locations and applied to the copies
    special cases:
    - 1 publisher / n subscribers
    - n publishers / 1 subscriber (field service employees – central office)

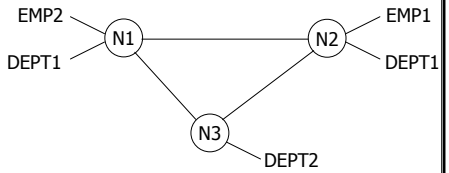
Query Optimization: Problem Statement



- **Query in N3:**
  SELECT * FROM Q WHERE a IS IN (482, 517, 763);
  - local execution in N3 or
  - execution in N1 with (smaller) fragment Q_1
- **Query in N2:**
  SELECT x, y, z FROM Q, R WHERE Q.j = R.k;
  - ship query for execution to N3 or
  - ship fragment Q_2 for join computation to N1

Query Optimization – Example

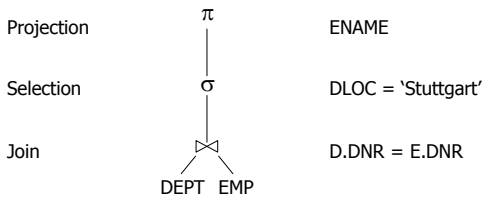
- DEPT (DNR, DLOC,...)
  EMP (ENR, ENAME, DNR, .. . **)**



➔ Fragmentation by predicate

DEPT1:	DLOC	=	'Stuttgart' OR DLOC = 'München'
DEPT2:	DLOC	=	'Frankfurt'
EMP1:	DNR ≤		'K50'
EMP2:	DNR >		'K50'

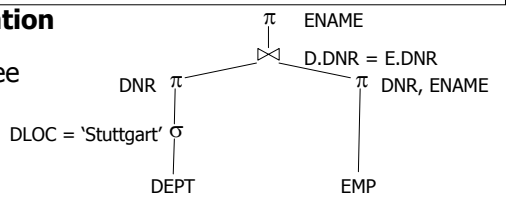
- Query in N3:
  Find the names of all employees working for departments in Stuttgart
- Operator tree (relational algebra)



Query Optimization – Example (2)

- **Algebraic optimization**

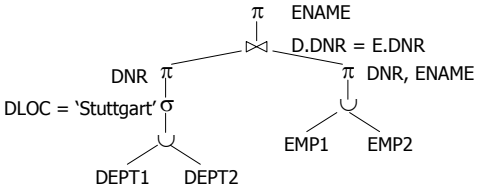
➔ Optimal operator tree for a centralized DBS



- **In the centralized as in the distributed case:** fatal assumptions
  - uniform distribution of all attribute values of an attribute
  - independence of attribute values
  - further assumption: uniform network load

➔ computation of expected #msgs dependent on these assumptions

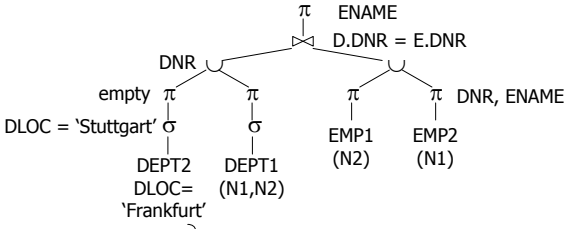
- **Fragmentation transformation**



DEPT and EMP are fragmented by means of predicates

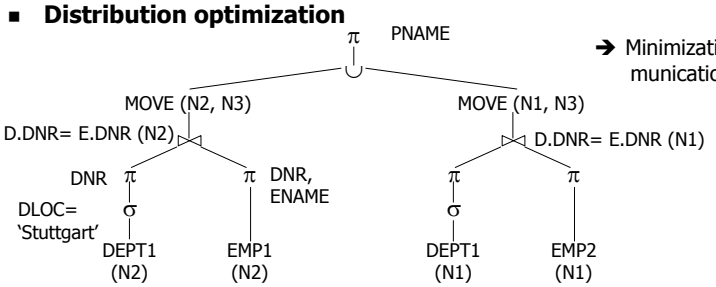
Query Optimization – Example (3)

- **Optimization and distribution transformation**



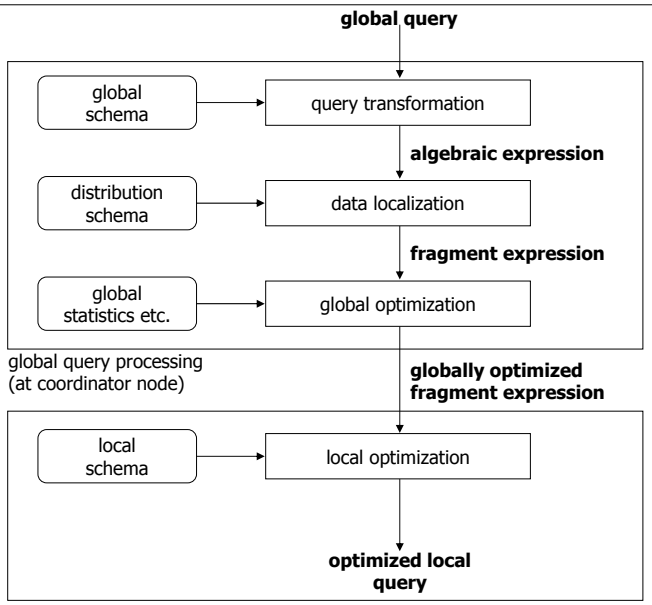
➔ EMP1 and DEPT1 in N2
EMP2 and DEPT1 in N1
} first ⋈ , then ∪

- **Distribution optimization**



➔ Minimization of communication cost

Phases of Distributed Query Processing



Global Optimization

- **Goal: execution plan with minimal global costs**
  - designation of **processing nodes**
  - fixing of execution sequence (sequential, parallel)
  - assessment of alternative strategies for join computation (e.g. semi-join)
  - refined data allocation leads to selection and projection ops on fragments

  → **separate optimization of global and local query execution may lead to suboptimal plans**

- **Cost model**
  - consideration of costs for CPU, I/O and communication

$$\text{total cost} = W_{CPU} * \#\text{instructions} + W_{I/O} * \#\text{I/O} + W_{msg} * \#\text{messages} + W_{byt} * \#\text{bytes}$$

- **Needed statistics**
  - cardinalities of tables and fragments
  - sizes of records and attributes
  - frequency distributions of attribute values, • • •

Parallel Selection / Projection

- **Effective Parallelism for horizontal fragmentation:** R = ∪ (R1, R2, ..., Rn)

  Selection: σ (R) → ∪ (σ (R1), σ (R2), ... , σ (Rn))
  Projection: π (R) → ∪ (π (R1), π (R2), ... , π (Rn))

  - data distribution enables parallel computation of local (partial) selections/projections
  - union of partial results
  - projection: (double) duplicate elimination if necessary

- **Shared Nothing**
  - operator execution at data nodes
  - nodes and degree of parallelism (n) determined by the **data distribution** (exception: specific queries on distribution attributes)
  - in general, index scans have to be executed on all n nodes, too

- **Shared Disk / Shared Everything**
  - data distribution on disk only determines maximal degree of parallelism
  - selective queries can be restricted to a single node (→ min. communication overhead)
  - table scans can be operated by n processors
  - selection of nodes can be performed at runtime (→ dynamic load balancing)
  - degree of parallelism depending on query type and current workload

Parallel Computation of Aggregate Functions

- Q(R) be an attribute of R to which built-in functions can be applied
- **Parallel computation of MIN, MAX is always possible**

  MIN (Q(R)) → MIN (MIN (Q(R1)), ... , MIN (Q(Rn))
  MAX (Q(R)) → MAX (MAX (Q(R1)), ... , MAX (Q(Rn))

  - parallel computation of local minima/maxima → result selection

- **SUM, COUNT, AVG only allow parallel computation, if no *duplicate elimination* required**

  SUM (Q(R)) → Σ SUM (Q(Ri))

  COUNT (Q(R)) → Σ COUNT (Q(Ri))

  AVG (Q(R)) → SUM (Q(R)) / COUNT (Q(R))

Join – Most Important Operator

- **Implementation alternatives in centralized DBMS**
  - **Nested Loops:**
    - each records of the outer (larger) table is compared to each record of the inner (smaller) table
    - applicable for all kinds of joins
  - **Sort-Merge**:
    - input tables are sorted according to join attribute or have index on join attribute (→ optimization: clustered index)
    - equi-join via table scans (other join types are more complicated)
  - **Hash Join:**
    - inner table is stored in main-memory hash table (HT)
    - checking for each record of the outer table if value of the join attribute is in HT
    - only applicable for **equi-join**
- **Optimization goals in DDBS/PDBS**
  - reduction of communication costs and use of parallelism
  - multi-way joins:
    - execution sequence determined by the form of operator trees
    - intra- and inter-operator parallelism
    - data- and pipeline parallelism

Trends in DBS

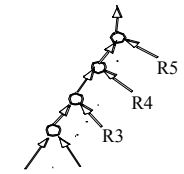
Shared nothing vs. shared disk
Technical problems
Replication
Query optimization
SN vs. SD comparison
Measures of TA processing
The benchmark success
Overflow

Joins in Distributed DBMS

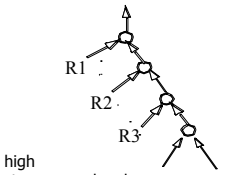
- **Query at node K requiring join between**
  (partial) table R at node K_R and (partial) table S at K_S
  - designation of processing node: K, K_R or K_S
  - **Selection of evaluation strategy**
    a) **Ship Whole:** ship tables completely to a node and execute local join
    - minimal number of msgs, very high transfer volumes

    b) **Fetch as Needed:** request for each join value in first table related records in second table
    - high number of msgs, but only relevant records are considered

    c) trade-off solution: *semi-join* resp. extensions (*hash-filter join*)
- **Multi-way joins: determination of execution sequence!**

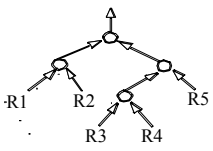


left-deep join tree



high storage overhead

right-deep join tree



greatest optimization potential

unrestricted (bushy tree)

Each join in the join tree (for hash joins) has its building table to the left and its probing table to the right.
A left-deep tree is a deep tree whose probing tables are restricted to base tables.

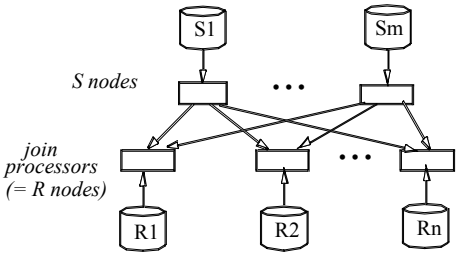
Parallel Join – Dynamic Replication of the Inner Table

- **Join between R and S**

  R = ∪ (R1, R2, ... , Rn) and S = ∪ (S1, S2, ... , Sm), S be smaller than R

- **Algorithm**

  1. *Coordinator: initiates join at all Ri (i = 1 ... n) and all Sj (j = 1 .. m)*
  2. *Scan phase: execute in parallel at each S node:*
     read local partition Sj and transmit it to each node Ri (i = 1.. n)
  3. *Join phase: executes in parallel at each R node having partition Ri:*
     - *S := ∪ Sj (j=1..m)*
     - computes Ti := Ri ⋈ S (implies read of Ri)
     - transmits Ti to the coordinator
  4. *Coordinator: receives and merges all Ti*



properties:
- applicable for **all** join predicates
- local join computation can use an arbitrary method

Trends in DBS

Shared nothing vs. shared disk

Technical problems

Replication

Query optimization

SN vs. SD comparison

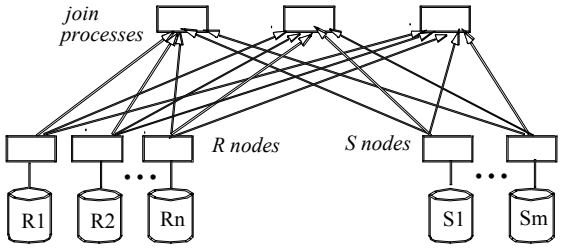
Measures of TA processing

The benchmark success

Overflow

Parallel Join – Dynamic Partitioning

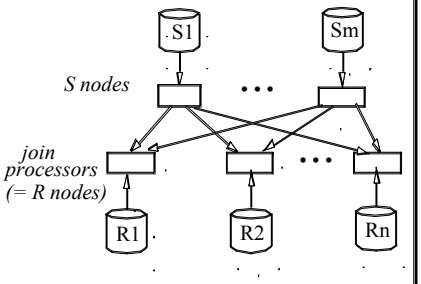
- **Prerequisite: equi-join!**
- General case:
  - **redistribution** of both tables under p join processors
  - **distribution function** (hash- or range partitioning) for the join attribute



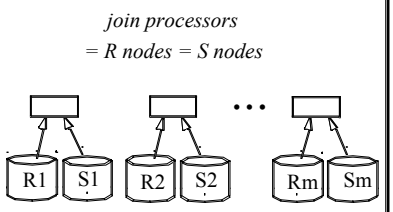
- **Assessment**
  - every **local join algorithm** applicable
  - reduced join overhead as compared to dynamic replication
  - high flexibility for dynamic load balancing
    (degree of parallelism p and join processors can be freely chosen)
  - high communication overhead

Parallel Join – Dynamic Partitioning (2)

- **Special case 1**
  distribution attribute = join attribute for one table (e.g. R)
  - only one table needs to be redistributed
  - no potential for dynamic load balancing anymore

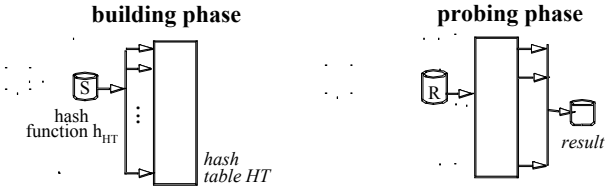


- **Special case 2**
  both tables carry join attribute as distribution attribute and identical distribution function (m=n, Ri and Si at the same nodes)
  - can be characterized as dependent horizontal fragmentation
  - no redistribution required!



Hash Join (Centralized Case)

- **Ideal case: inner table S completely fits into memory**
  - building phase
    reading of S from disk and storing it in HT in memory using a hash function $\mathbf{h_{HT}}$ on the join attribute
  - probing phase
    reading of R from disk and checking each record if, for the join attribute value, related S records are in HT
    (if so, records are added to join result)

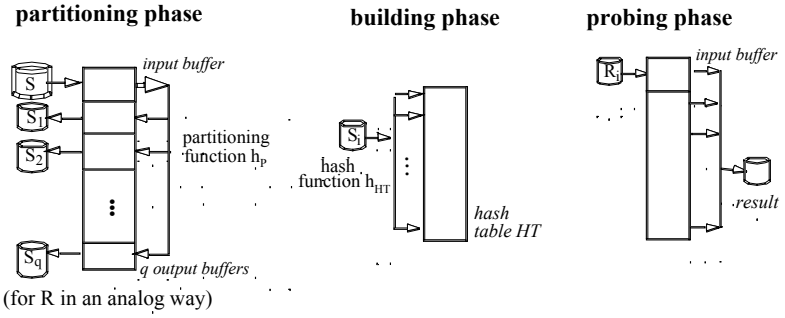


- **Pros**
  - linear cost O(N)
  - hashing reduces search for join partners to the records of a hash class (partitioning of search space)
  - use of large memories
  - well applicable for joins on intermediate results, too

Hash Join (2)

- **General case**
  inner table does not completely fit into memory
  → **overflow handling required**

- **Solution: partioning of input tables**
  - partitioning of S and R in q partitions using (hash) function $\mathbf{h_P}$ on join attribute such that each S partition fits into HT
  - q-fold application of base algorithm for each of related partitions

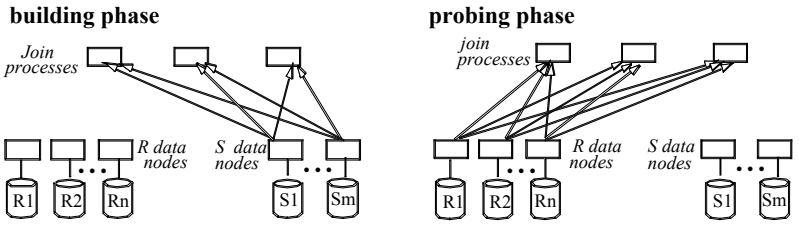


(for R in an analog way)

- Approx. 3-fold I/O overhead compared to base method without overflow

Parallel Hash Join

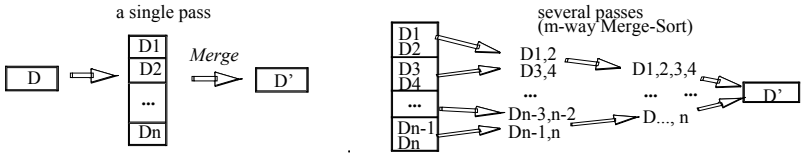
- **Principle**
  - partitioning and redistribution of smaller table S via hash function $\mathbf{h_P}$ on join attribute
  - S records transmitted to join processors are stored in HT (hash fct. $\mathbf{h_{HT}}$)
  - redistribution of second table R onto join processors using $\mathbf{h_P}$
  - probing: for incoming R records, the join partners are determined in HT



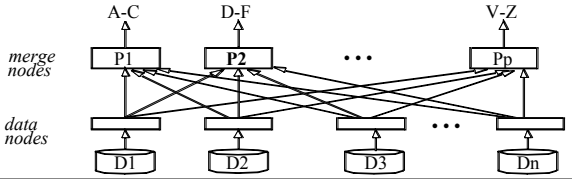
- **Properties**
  - sequential processing of building- and probing phases!
  - advantage: reduction of redistribution overhead for R possible using a so-called bit-vector filter (recorded in the building phase on S nodes)
  - pipeline parallelism applicable in building- and probing phase
  - overflow handling required, if S partitions do not completely fit into HT (→ three-stage partitioning)

Parallel Sorting

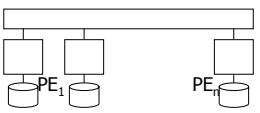
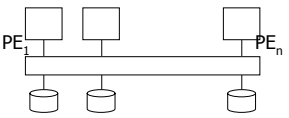
- **DBMS: external sorting**
  - decomposition of the input into several runs, sorting and merging them



- **Requirements**
  - parallel input (multiple input) and parallel sort phases
  - parallel merge and partitioning of sorted output (multiple output)
- **Approach**
  - local sorting of partitions, dynamic redistribution of sorted runs over p merge nodes
  - redistribution is governed via dynamic range fragmentation of sort attribute
  - parallel merge in p merge nodes and partitioned output



Shared-Nothing vs. Shared-Disk

	Shared nothing	Shared disk
Criterion		
Performance	- static data partitioning determines execution location of DB operations - fewer opportunities for load balancing or saving of communication requests	- local accessibility of all data facilitates load balancing - close coupling can be used for performance enhancements - higher flexibility to be used for parallelism
Extensibility	- new computer requires physical repartitioning of the DB ($N \rightarrow N+1$) - simple attachment of disks	- no physical (re-)partition of the DB - direct attachment of disks may limit number of computers ($\rightarrow$ msg-based I/O interface)

Shared-Nothing vs. Shared-Disk (2)

Criterion	Shared-Nothing	Shared-Disk
Availability	- takeover/recovery of the affected partition by other computer should be provided (danger of overload possible) - geographically distributed replication enables fast catastrophe recovery	- entire DB is accessible after node crash - complex crash recovery - creation of a global log file
Realization problems	- physical DB partitioning - distributed query processing - handling of replicated DBs - distributed commit protocol - global deadlock handling - load distribution, -balancing - administration - special problems in geographically distributed systems (network partitioning, node autonomy, …)	- concurrency control - global deadlock handling - coherency control - logging - recovery - load distribution, -balancing - parallel query processing - administration

Shared Nothing vs. Shared Disk

criterion	Shared nothing	Shared disk
performance (ideal load)	++	++
load balancing	-	++
availability	-	+
using replication	+	?
extensibility	o	+
location transparency	++	++
heterogeneous databases	-	-
node autonomy	-	-
geographic distribution	++	--
cost effectiveness	+	+/o (disk attachm.)
administration	--	o

Benchmarks[1] – Reproduction of TA Workloads

- Characterization of performance of complex SW systems is very difficult
  - TP-, DW-, or DSS system
  - performance behavior strongly varying for different workloads!
  - Response time is critical factor for interactive TA

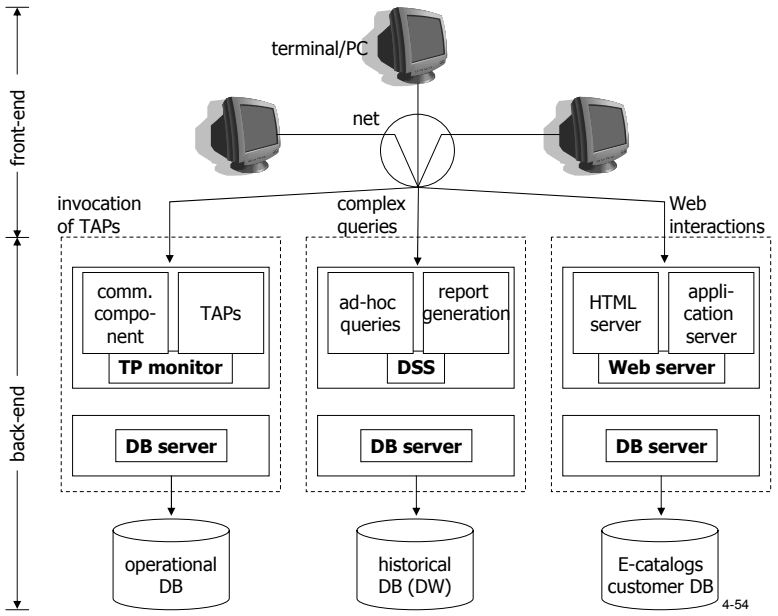
➔ How can a system be evaluated and compared with others w.r.t. performance?

- **Needed:** test to evaluate the essential performance features

- Requirements specified as application-related functionality (TA types)
  - prediction of the entire system cost
  - performance behavior under growth of the TA workload and/or of the data volumes (scalability)
  - objective performance measures such as throughput and response time to enable easy comparability (in case of system growth or competing systems)

1 Benchmark: Vergleichspunkt, Bezugswert, Maßstab

Benchmarks for Transaction Systems – Test Environment

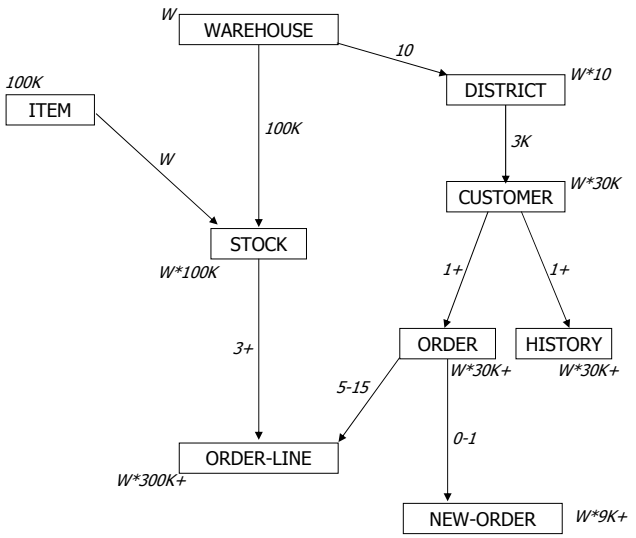


Benchmark TPC-C

- **Modeling/processing of activities at wholesale**
  - representative for complex OLTP application environments
  - management, sale, or distribution of products or services
  - enterprise owns geographically distributed business districts and corresponding stores
  - more realistic transaction workload consisting of several transaction types of varying complexity and update frequency
- **Application: order processing at wholesale**
  - business comprises W warehouses, per warehouse 10 districts, per districts 3000 customers
  - 100.000 articles; number of existing articles is kept per warehouse
  - 1% of all orders are requested from a non-local warehouse

Benchmark TPC-C (2)

- **9 record types**



TPC-C (3)

BEGIN WORK {Begin of Transaction}
SELECT … FROM **CUSTOMER**
WHERE c_w_id = :w_no AND c_d_id = :d_no AND c_id = :cust_no
SELECT … FROM **WAREHOUSE** WHERE w_id = :w_no
SELECT … FROM **DISTRICT** (* → next_o_id*)
WHERE d_w_id = :w_no AND d_id = :d_no
UPDATE DISTRICT SET d_next_o_id := :next_o_id +1
WHERE d_w_id = :w_no AND d_id = :d_no
INSERT INTO NEW_ORDER …
INSERT INTO ORDERS …

per article (on avg. 10) the following statements are executed:
SELECT … FROM **ITEM** WHERE …
SELECT … FROM **STOCK** WHERE …
UPDATE STOCK …
INSERT INTO ORDER-LINE …
COMMIT WORK {End of Transaction}

- on avg. 48 SQL statements
  (BOT, 23 SELECT, 11 UPDATE, 12 INSERT, EOT)
- throughput measure for New-Order transactions in tpmC
  (transactions per minute)

TPC-H and TPC-R

- **Benchmarks for the evaluation of complex queries on large DBs**
  - originally introduced as TPC-D (valid until 4/99)
  - reproduce typical activities at wholesale using queries
  - focus on data analysis
    - computation of trends
    - support of the decision process
  - use schema, scaling factors and queries from TPC-D
  - extend the queries to 22 query types and 2 update functions
    (DB refresh)

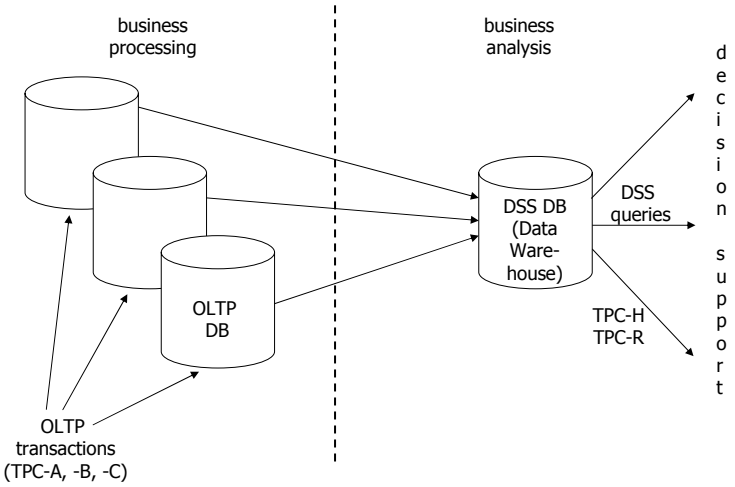
- **Availability of DB:** 24*7*52 h (incl. maintenance breaks)

- **TPC-H** (ad-Hoc, decision support)
  does not exploit pre-knowledge (long execution times for queries)

- **TPC-R** (business Reporting, decision support)
  exploits pre-knowledge (in contrast to TPC-H):
  DBS can be specially optimized w.r.t. standard queries!

TPC-H und TPC-R (2)

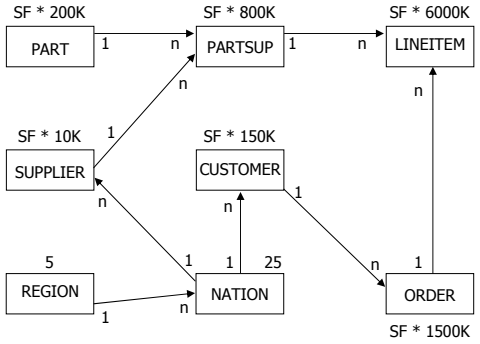
- **Usage environment**



TPC-H (3)

- Schema

SF: scale factor



Query example Q9**:**

Product Type Profit Measure Query

The query finds, for each nation and each year, the profit for all parts ordered in that year which contain a specified substring in their names and which were filled by a supplier in that nation. The profit is defined as the sum of [(L_EXTENDEDPRICE*(1-L_DISCOUNT)) - (PS_SUPPLYCOST * L_QUANTITY)] for all line items describing parts in the specified line. The query lists the nations in ascending alphabetical order and, for each nation, the year and profit in descending order by year (most recent first).

TPC Benchmarks: Key Data[1]

- **Many benchmark measurements from 33 corporations**
  - TPC-A (TPC-B): >300 (>130) published results from 115 (73) different systems
  - two "golden figures": tps and $/tps
  - TPC-A and TPC-B were not used anymore after 6/1995
- **Performance values for TPC-A**
  - ca. 100 - 200 KInstr. / TA (initially up to 1 Mill. Instr. per TA)
  - 2 I/O request per TA (initially up to 20)
  - 1990: 33 tpsA to 25,500 $/tpsA
  - 1995: 3692 tpsA to 4,873 $/tpsA

    ➔ 111 and 5 as improvement factors
- **Performance values for TPC-B**
  - ca. 75 KInstr. / TA
  - 1991: 103 tpsB zu 4,167 $/tpsB
  - 1994: 2,025 tpsB zu 254 $/tpsB

    ➔ 19 und 16 as improvement factors
- **Why were these TPC benchmarks so successful?**
  - first benchmark measurements without special optimization
  - real performance improvements through HW- and SW-products
  - system improvement to eliminate the performance bottlenecks revealed by the benchmark
  - effective use of the "benchmark games": manufacturer learned from each other how to optimally run the benchmark

1 http://www.tpc.org/

TPC Benchmarks: Key Data (2)

- **Performance values for TPC-C**
  - 1992: 54 tpmC to 188,562 $/tpmC
  - 1998: 52,871 tpmC to 135 $/tpmC
  - 2001: 709,220 tpmC to 14. $/tpmC (TPC-C version 5)
  - 2006: 3,210,540 tpmC to 5.07 US

    ➔ **59,455** and **37,192** as improvement factors
- **Performance values for TPC-C (Version 5) using Price/Performance**
  - 2002: 16,756 tpmC to 2.78 $/tpmC
  - 2003: 82,226 tpmC to 2.76 $/tpmC
  - 2006: 38,622 tpmC to 0.99 $/tpmC
- **Performance values for TPC-D (at 100 GB)**
  - 1995: 84 QthD and 52,170 $/QphD
  - 1998: 1,205 QthD and 1,877 $/QphD

  ➔ 14 and 28 as improvement factors (until 1998)
  - since 1999: TPC-H and TPC-R; they are extended from 17 to 22 queries compared to TPC-D
- **Performance values for TPC-R**
  - 2000: 21,254 QphR and 607 $/QphR at 1000GB
  - 2003: 4,442 QphR and 35 $/QphR at 100 GB

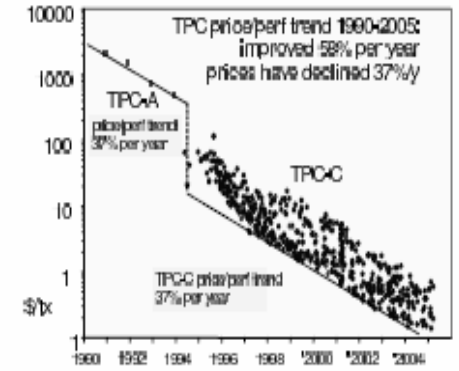
TPC Benchmarks: Key Data (3)

- **Performance values for TPC-H[1]**
  - 2002: 5,578 QphH and 358 $/QphH at 100 GB
  - 2006: 12,600 QphH and 7.67 $/QphH at 100 GB
  - 2002: 25,805 QphH and 203 $/QphH at 1,000 GB
  - 2006: 68,100 QphH and 59 $/QphH at 1,000 GB
  - 2002: 81,501 QphH and 243 $/QphH at 10,000 GB
  - 2006: 108,099 QphH and 53.80 $/QphH at 10,000 GB
- **Performance values for TPC-W**
  - 2000: 1,262 WIPS and 277 $/WIPS
  - 2002: 21,139 WIPS and 32.62 $/WIPS (Item Count 10,000)
  - 2002: 10,439 WIPS and 106.73 $/WIPS (Item Count 100,000)
    (2002 are the latest reported results, TPC-W obsolete as of 4/28/05)
- **Remark:**
  for "transactions" (better mouse clicks) via the Internet, people jokingly coined the cost measure „m$/tps"

1 Note: The TPC believes that comparisons of TPC-H results measured against different database sizes are misleading and discourages such comparisons. The TPC-H results shown below are grouped by database size to emphasize that only results within each group are comparable.

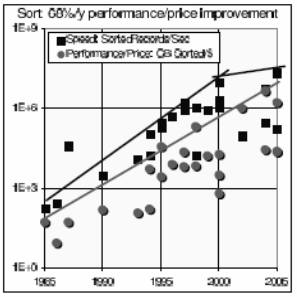
A "Measure of Transaction Processing" 20 Years Later

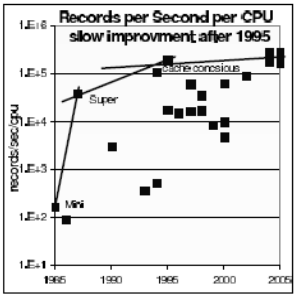
- **Jim Gray et al., 1985 defined three performance benchmarks**
  - DebitCredit, a test of DBS and TA system
  - Sort, a test of the OS and I/O system
  - Copy, a test of the file system
- **DebitCredit**
  - morphed into TPC-A and then TPC-C
  - 100 TPS → 100,000 TPS
  - TPC price/performance trend 1990 – 2005
    - improved 58%/y
    - prices have declined 37%/y



A "Measure of Transaction Processing" 20 Years Later (2)

- **sort benchmarks**
  - sort 1M records (now a fraction of a second)
  - PennySort (sort as much as you can for a penny)
  - MinuteSort (sort as much as you can in a minute)
  - TerabyteSort (sort a trillion records)
- **results**
  - sort speed doubled every year from 1985 to 2000
  - but only 20% since then
  - price-performance has steadily improved at 68%/y





Summary

- **MRDBS enable**
  - the management of very large DBs
  - the processing of very high transaction workloads
  - interactive operations on very large data volumes, especially with parallel DBS (full-text search, multimedia operations, new data types, ...)
- **Main architectures: SD and SN**
  - **virtualization is a big issue at the OS level**
  - **clusters of mainframes for high-level requirements in performance, availability, extensibility, ...**
- **Practical performance evaluation of HPTS**
  - TPC-A and -B are too simple; TPC-C no challenge anymore
  - TPC-H and –R address decision support environments
  - TPC-App is currently introduced

➔ complex benchmark specifications; interpretation of results becomes more difficult

Further References

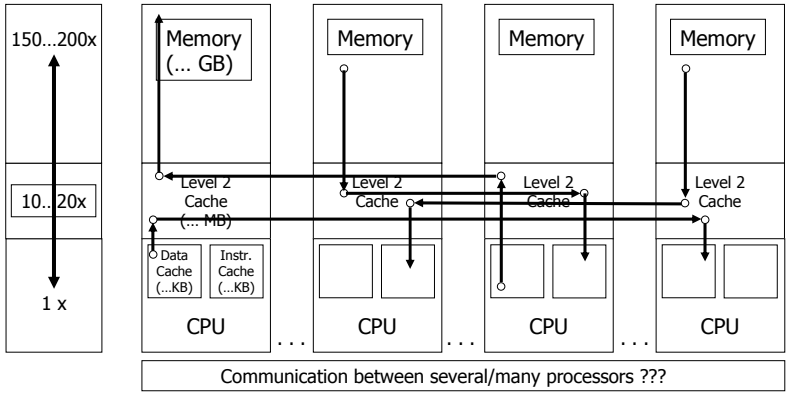
- *Gray, J.:* A "Measure of Transaction Processing" 20 Years Later, IEEE Data Engineering Bulletin 28:2, 3-4, 2005
- *Özsu, T., Valduriez, P.:* Distributed and Parallel Database Systems. The Computer Science and Engineering Handbook 1997: 1093-1111
- *Özsu, T., Valduriez, P.:* Principles of Distributed Database Systems. Second Edition, Prentice-Hall, 1999
- *Rahm, E.:* Hochleistungs-Transaktionssysteme. Konzepte und Entwicklungen moderner Datenbankarchitekturen. Vieweg 1993
- *Rahm, E.:* Mehrrechner-Datenbanksysteme, Addison-Wesley, 1994
- *Spruth, W., Rahm, E.:* Sysplex-Cluster-Technologien für Hochleistungs-Datenbanken. Datenbank-Spektrum 3, 16-26, 2002

Mainframes: 5 9` Guaranteed by the Platform

- (Only) mainframes provide **sufficient availability, reliability, scalability, performance, etc.** for HPTSs
- They embody the fundament: the hardware- and firmware-basis (IBM example: zSeries)
  - performance & scalable capacity of all resources
    - processors, data flow, memory,
    - I/O bandwidth, I/O connectivity, Sysplex
  - extreme availability of the entire infrastructure
    - processors, books, I/O, firmware, system
  - flexibility of operation
    - dynamic optimization of all system resources anchored in the architecture
    - Logical and physical capacity adaptation at runtime
  - security/safety of operation
    - Certified LPAR security, highly secure cryptographic hardware

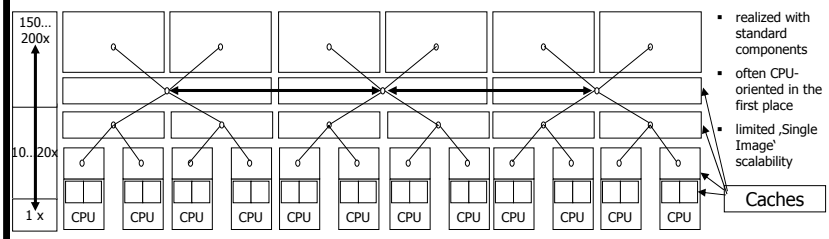
Communication between several/many Processors

Dilemma of current microprocessor systems:
memory access does not scale with CPU frequency !



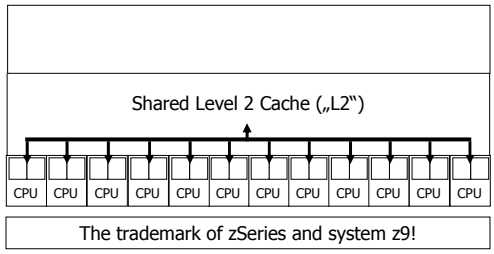
Multi-Processor Structures

Distributed hierarchically ordered switches with ‚Non-Uniform Memory Access' characteristics:



- realized with standard components
- often CPU-oriented in the first place
- limited ‚Single Image' scalability

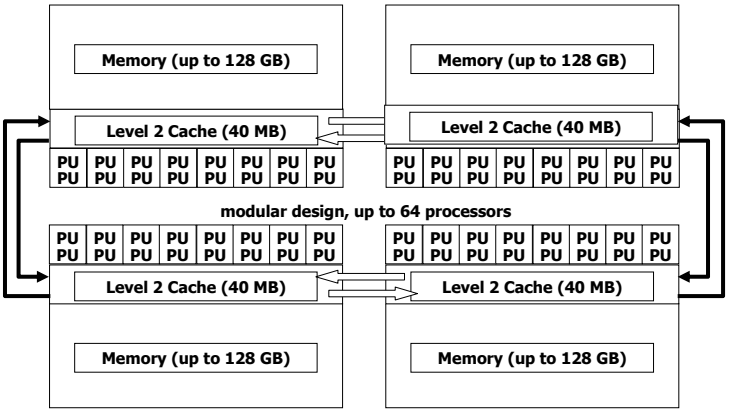
. . . or a central switch with parallel access of all CPUs



- highly optimizing L2-design
- requires extremely dense packaging
- data oriented
- extremely high scalable
  - extremely flexible
  - self-optimizing
  - extremely high available (2-8 spare processors are included for free)

Extended Multi-Book Structures

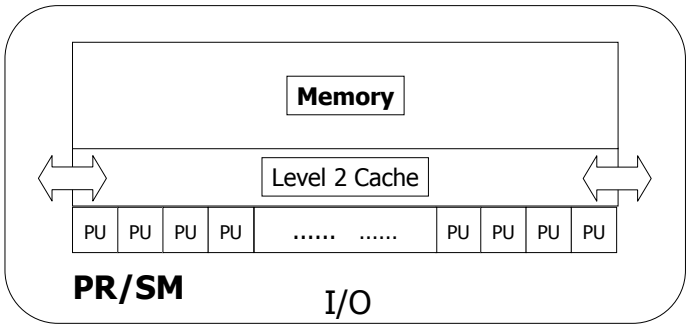
System roadmap:
- zSeries z900 (2003): 2 books, 1 proc./chip, 16 MB L2 cache/book, 32 GB memory/book
- system z990: 4 books, 2 proc./chip, 32 MB L2 cache/book, 64 GB memory/book
- system z9-109 (2005): 4 books, 2 proc./chip, 64 MB L2 cache/book, 128 GB memory/book



Dynamic replacement of books: everything (processes, memory structures, I/O connections) are automatically migrated from one book to another (may last for hours!)

Multi-Book Structures (Logical View)

zSeries and system z9



- Mode of operation: "Logical Partition Mode"
- A single pool of physical resources (CPU's, memory, I/O) in modular implementation (1/2/3/4 nodes/'books')
- Usage by virtual servers: up to 60 LPARs … 100 + … (VM)
- Multiple channel subsystems: up to 4 x 256 "I/O channels" (Channel Path IDs)