DIS
2014

1. Architecture of Database Systems

Norbert Ritter

Towards a relational DBS architecture

Evolution of the layer model

Optimization issues

Provisions for ACID properties

Architectural extensions for DBS

DBS use in CA* environments

Main reference:
Theo Härder: DBMS Architecture – The Layer Model and its Evolution,
in: Datenbank-Spektrum, dpunkt-Verlag, Heft 13, May 2005, pp. 45-57.

Special thanks to **Theo Härder** for his support w.r.t the slides of this chapter!

UHH

The "Religious War" – Which Abstraction Level Wins?

- Network model
  - The more complex the data structure, the better
  - But: very simple operations

⇒ **use of pointers, navigation, record orientation**

- Relational model
  - "Data structure of spartan simplicity" (E. F. Codd)
  - Operations with closure property
  - Each embellishment needs additional operations
  - Therefore: simplicity is the secret for data independence

⇒ **Use of values, declarative queries, set orientation**

Structured Programming and Data Independence

- E. W. Dijkstra's ideas
  - "Notes on Structured Programming" (EWD249, 1969)
  - "The Goto Statement Considered Harmful" (Comm. ACM, 1968)

Battle cry translated into the world of data bases:

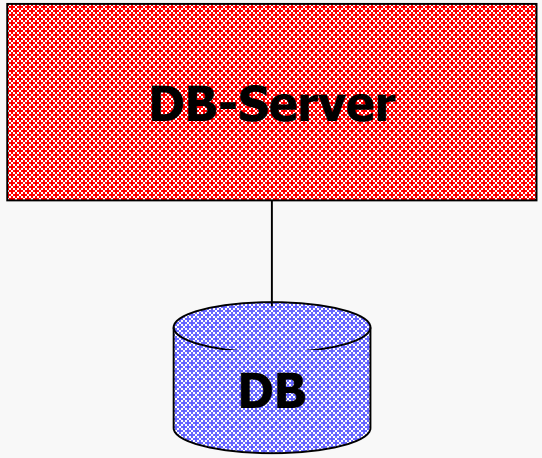
pointers are the evil!

- Since 1971: D. L. Parnas publishes his ideas concerning
  - "Information Hiding" at the IFIP conference
  - Hierarchical structuring of software systems, …

- The concept of "information hiding" as a software design principle is widely accepted in academic circles. Many successful designs can be seen as successful applications of abstraction and information hiding. On the other hand, most industrial software developers do not apply the idea and many consider it unrealistic …

30 Years Ago: Towards a Layered DBMS Architecture

- Relational model
  - Declarative set-oriented access and data independence
  - How can the postulated independence at the language and implementation levels be transformed into a system?
- Monolithic approach?
  - **Q1**: Select B.Title, P.Year, A.Name
    From Books B, Authors A
    Where B.Author = A.Author
    And A.Name = „S*" And B.Topic = „DBMS"



  - Interface to external memory: read and write of pages (DB is a very long bit string!)

Layered DBMS Architecture – System Evolution!

- Permanent evolution expected
  - Long lifetime of a DBMS > 30 years
  - Growing information needs:
    object types, integrity constraints, ...
  - New storage structures and access methods, ...
  - Rapid changes in technology, storage media, ...
- But no revolution!
  - Space and time, data streams?
  - Unstructured and semi-structured documents? …
- Important challenges
  - Logical and physical data independence
    - Separation of applic. programs and data as strong as possible
    - Isolation of the interface from all changes in the DBMS
  - Encapsulation and hierarchical structuring in the system

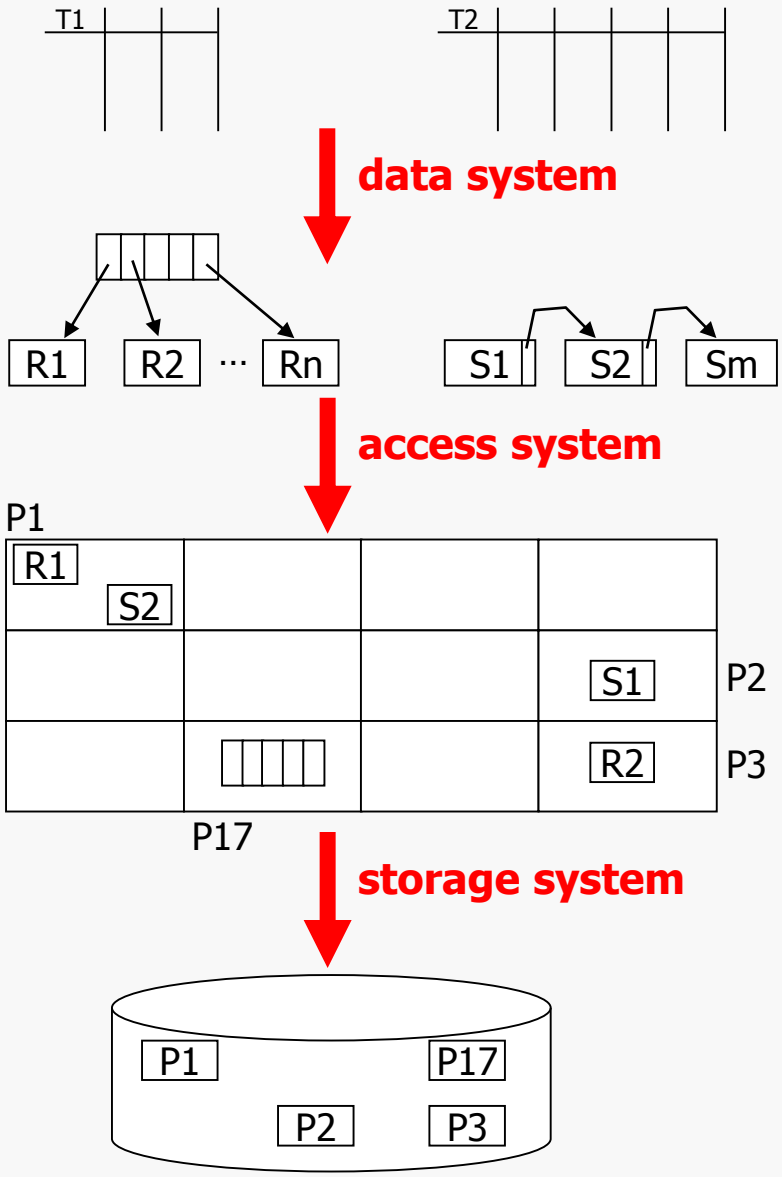
Abstraction is Geared to the Mapping Hierarchy

Tables/views with set-oriented operations

Variety of record types and access paths with record-oriented operations

DB buffer in memory providing page access

External storage with files of different types

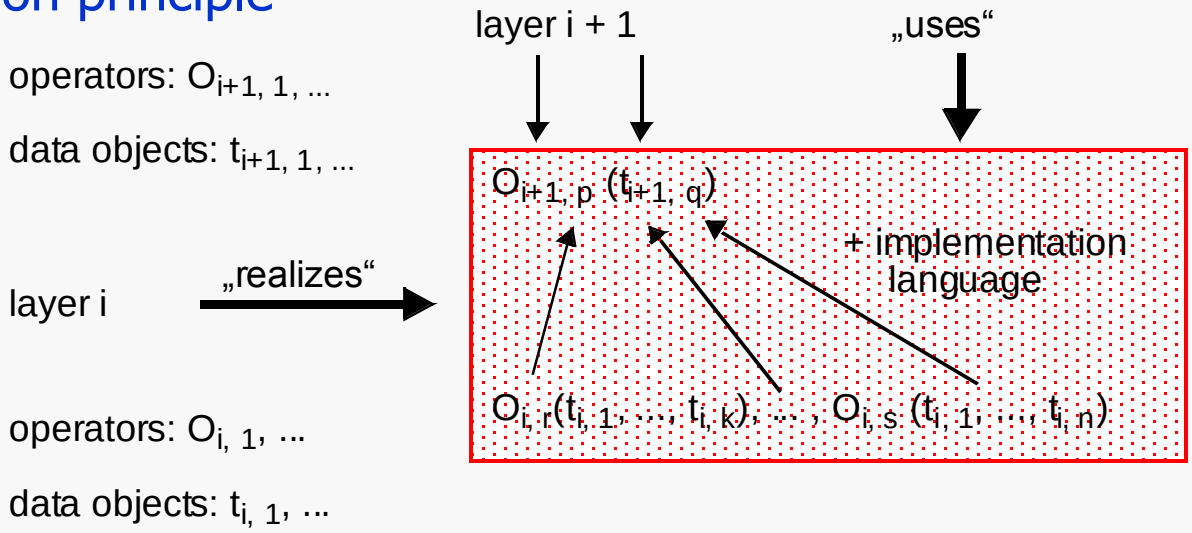


Layer Building in the Layer Model

- Goal: Architecture of a data-independent DBS
- How many layers are needed?

⇒ There is no architectural theory for the construction of large SW systems

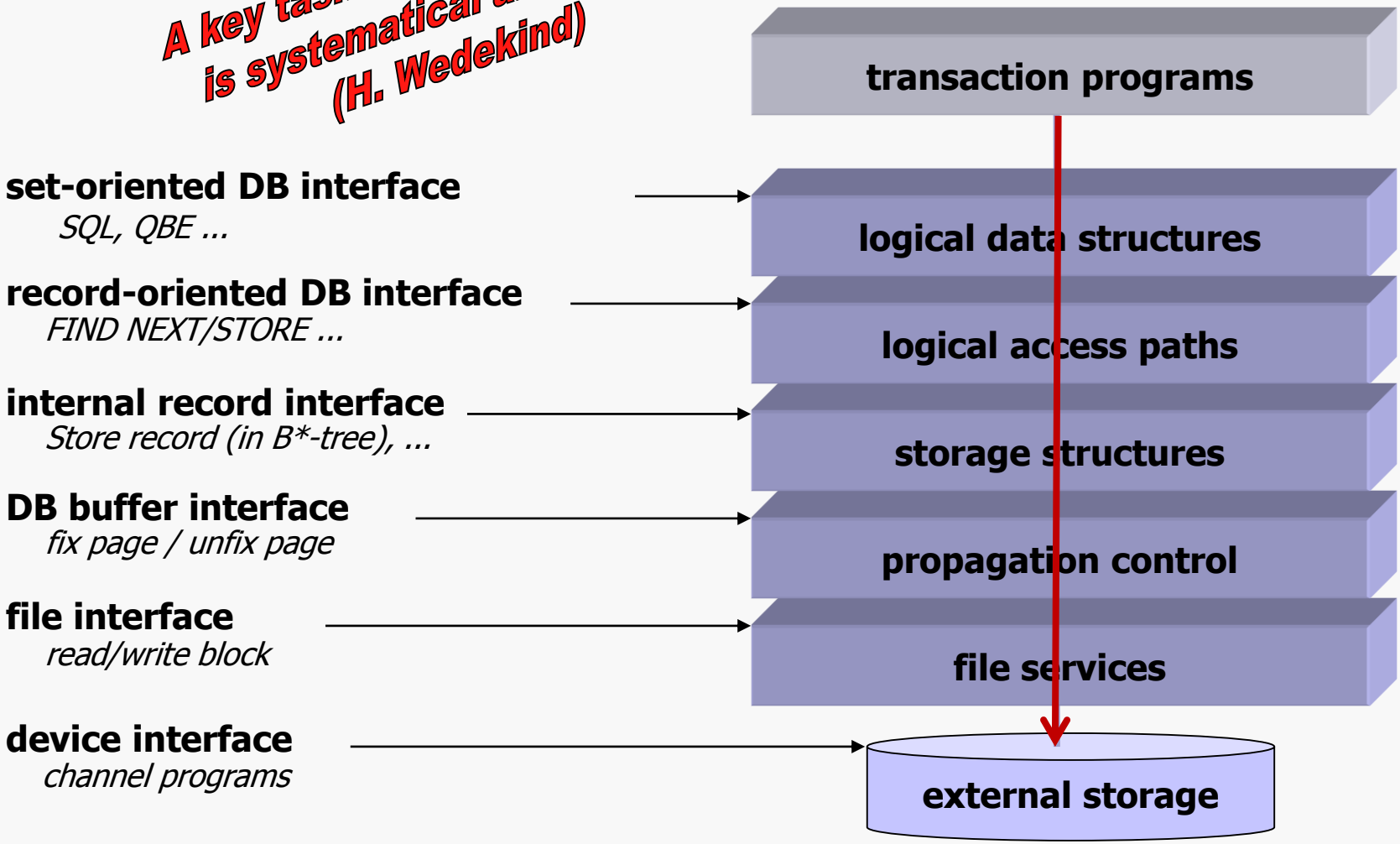
- **Recommended concepts**
  - Information hiding
  - Hierarchical structuring
- Construction principle



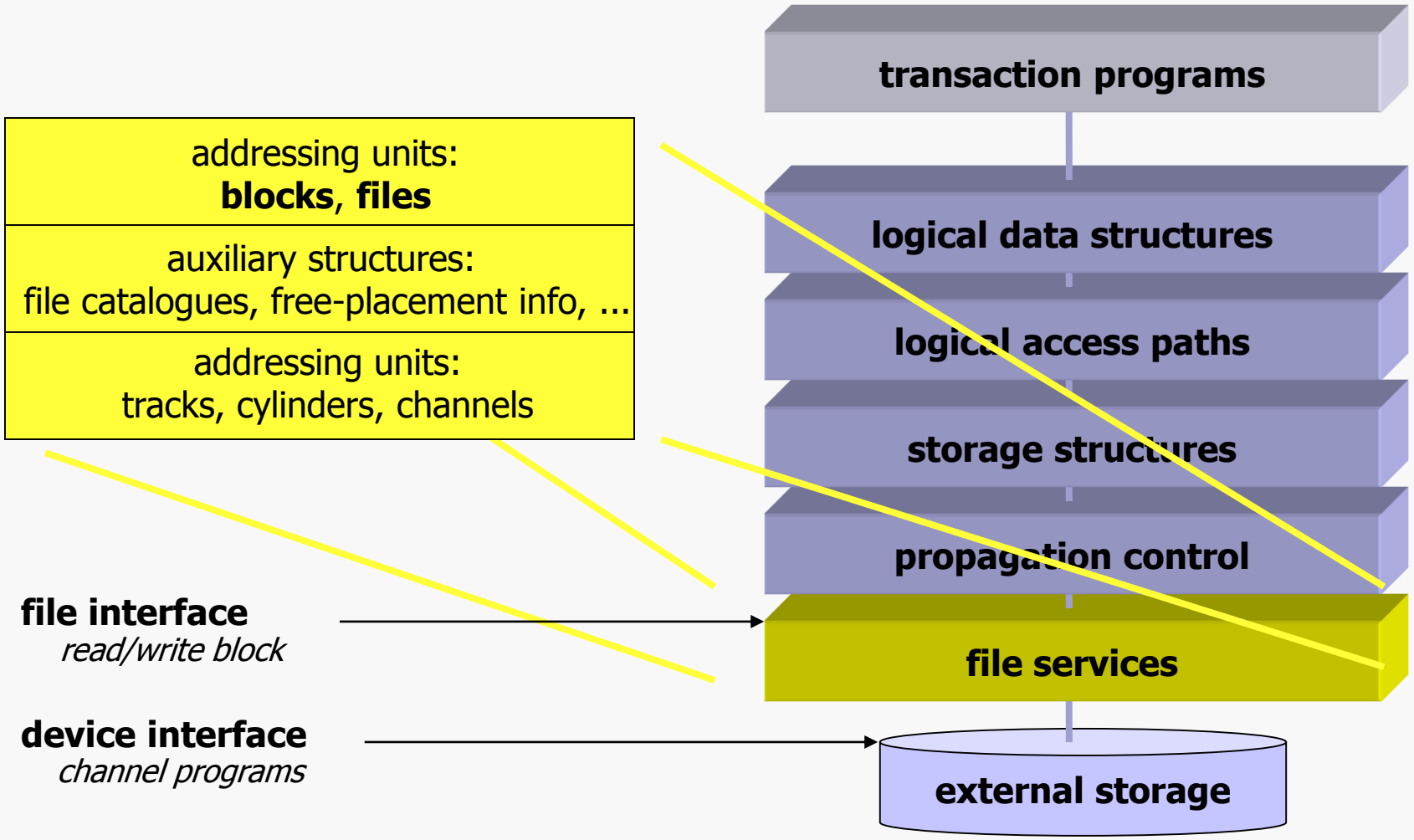
- "Uses" relation
  - A **uses** B, if A calls B and the correct execution of B is necessary for the complete execution of A

5-Layer Model (1)

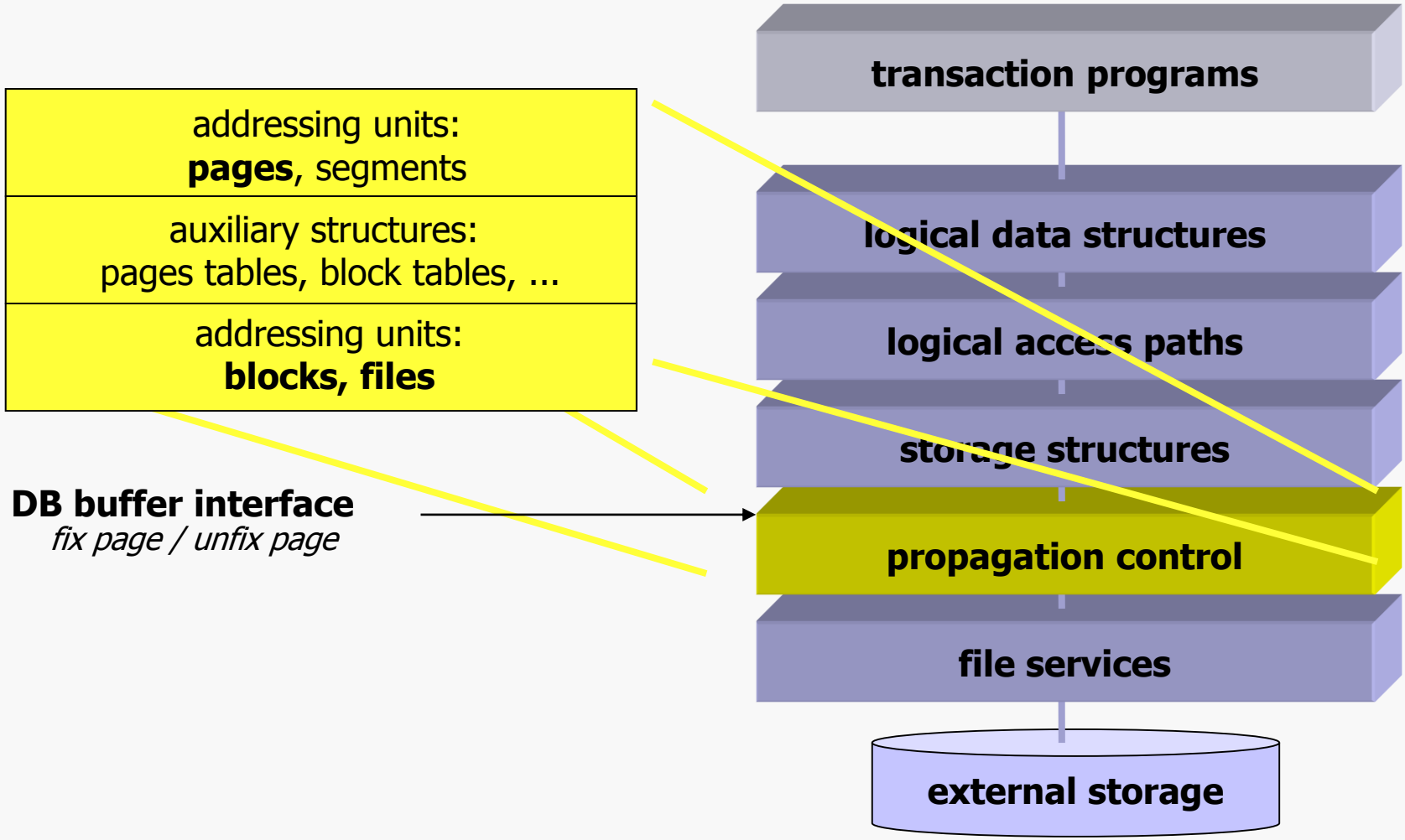
A key task of computer science is systematical abstraction. (H. Wedekind)



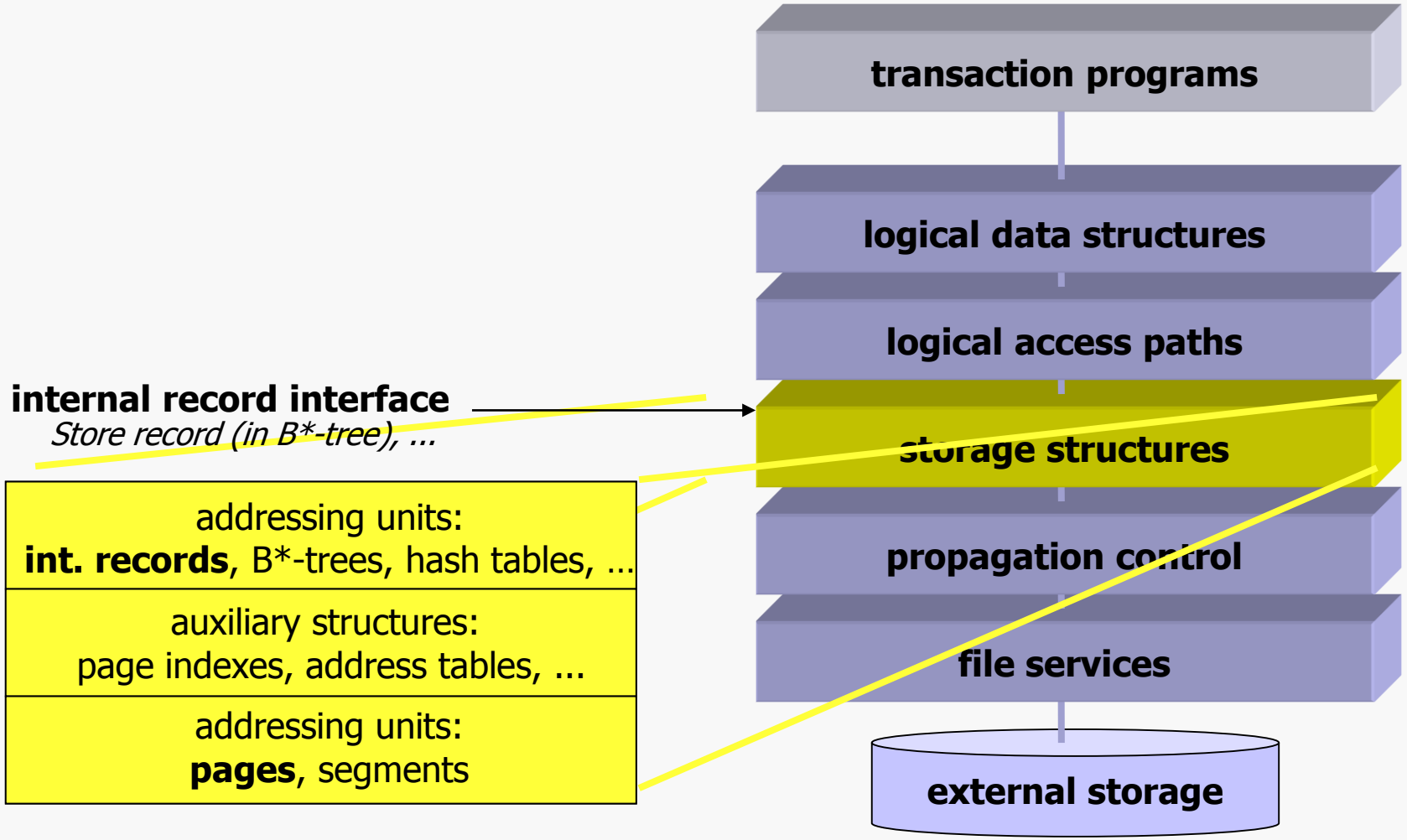
5-Layer Model (2)



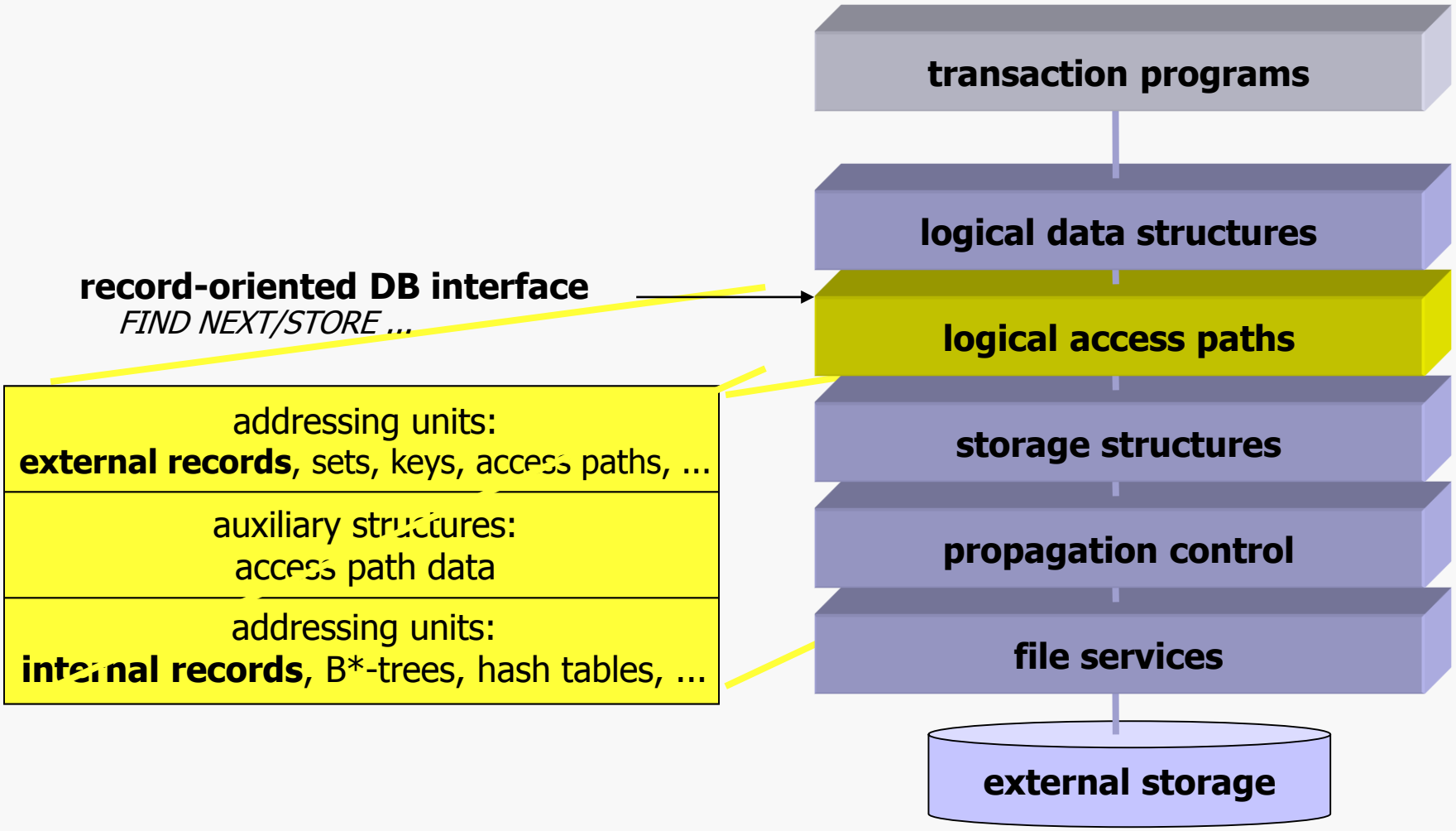
5-Layer Model (3)



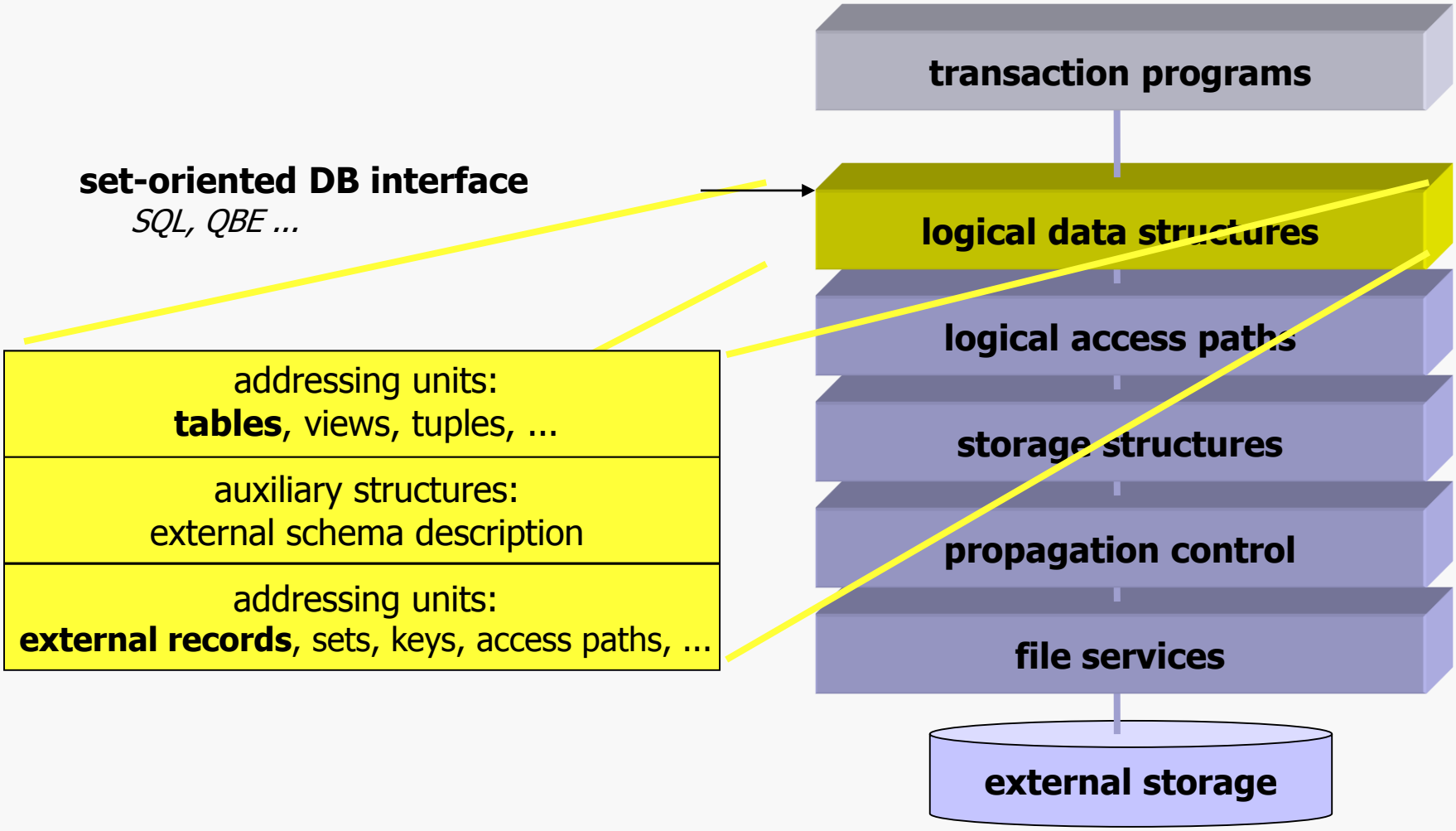
5-Layer Model (4)



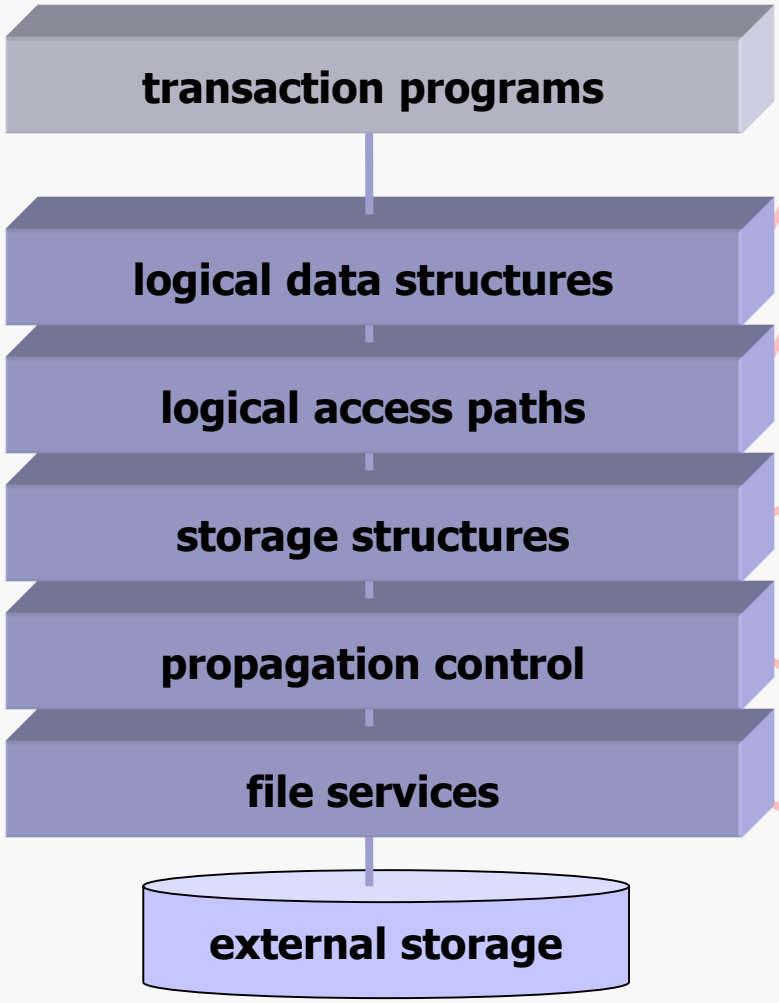
5-Layer Model (5)

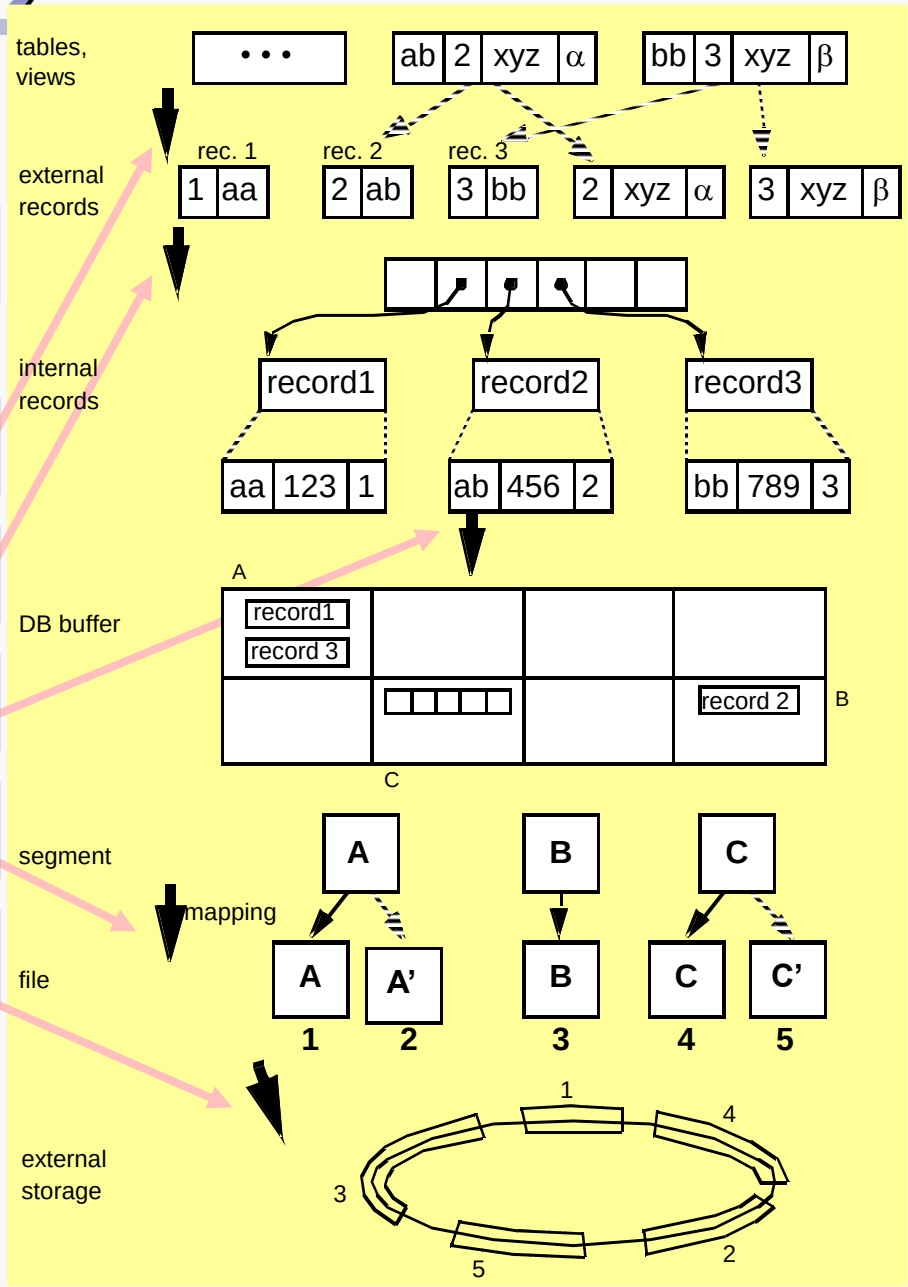


5-Layer Model (6)



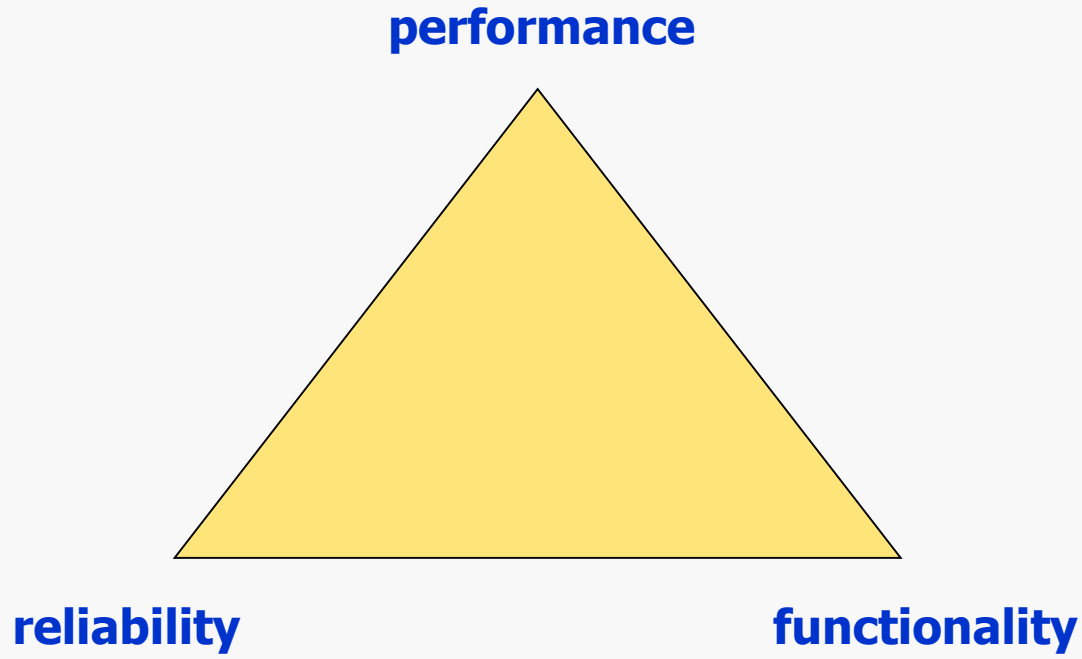
5-Layer Model (7)





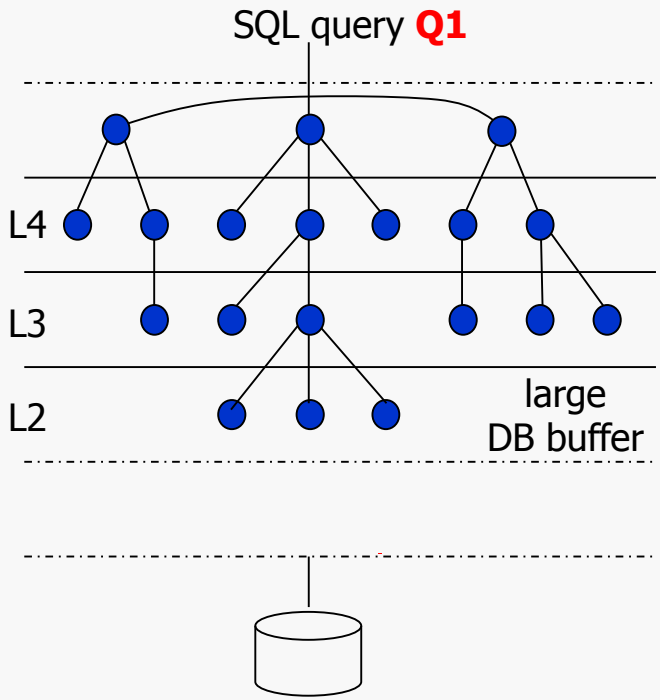
The Magic Triangle

- "Classical" key requirements for a DBMS



Layer Model – Role of n?

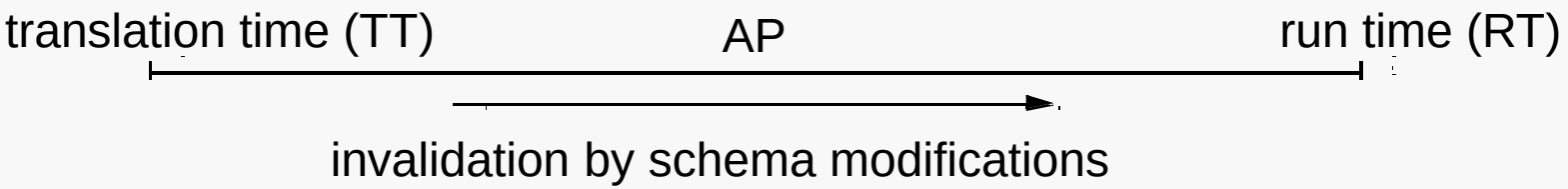
- For DBMSs, it is especially true:
  - "Performance is not everything, but without performance everything is worth nothing!"
  - What is the appropriate number of layers?
- Complexity / reliability of the mapping layer vs. optimization potential of the layer
- Optimized layer model at run time




```
Open Scan(I_B(Topic),Topic='DBMS',Topic>'DBMS')/*SCB1*/
Sort Access (SCB1) ASC AuthId Into T1 (AuthID, …)
Close Scan (SCB1)
Open Scan (I_A(Name), Name>='S', Name>'S')          /*SCB2*/
Sort Access (SCB2) ASC AuthID Into T2 (AuthID, …)
Close Scan (SCB2)
Open Scan (T1, BOF, EOF)                            /*SCB3*/
Open Scan (T2, BOF, EOF)                            /*SCB4*/
While Not Finished
Do
      Fetch Tuple (SCB3, Next, None)
      Fetch Tuple (SCB4, Next, None)
      . . .
End
```

DBMS – Paradigm of a Generic System

- **Pre-compilation and early binding are good ideas**
  - What – table-, attribute-descriptions, indexes, access rights, assertions, ...
  - When – spectrum of points in time



accesses (at TT):

efficient

data dependent!

{ **balance required!** }

accesses (at RT):

expensive

data independent!

- Layer-crossing optimization
  - Avoidance of thrashing: L1 – L4/L5
  - Reservation of buffer space (Hot Set Model)
  - Optimization of lock management (Lock Contention)

What can go wrong, will go wrong ...

May all your transactions commit and never leave you in doubt. (J. Gray)

- **Goal of Development**
  Build a system used by millions of people that is always available – out less than 1 second per 100 years = 8 9's of availability!
  (J. Gray: **1998 Turing Lecture**)
- Availability today (optimistically):
  - For Web sites: 99%
  - For well-administrated systems: 99,99%
  - zSeries OS only: 99,9999%
  - For an entire information system – realistic goal today: 99,999%
- There are some 9`missing (to be achieved until ~2050?)
- ACID transactions are introduced to guarantee far-reaching assertions concerning the quality of data

Jim Gray: We have added three 9s in 45 years (starting with 90%), or about 15 years per order-of-magnitude improvement in availability. We should aim for five more 9s: an expectation of one second outage in a century. This is an extreme goal, but it seems achievable if hardware is very cheap and bandwidth is very high. One can replicate the services in many places, use transactions to manage the data consistency, use design diversity to avoid common mode failures, and quickly repair nodes when they fail. Again, this is not something you will be able to test: so achieving this goal will require careful analysis and proof.

Transaction as Dynamic Control Structure

- **Atomicity**
  "All or nothing" property of any DBMS action

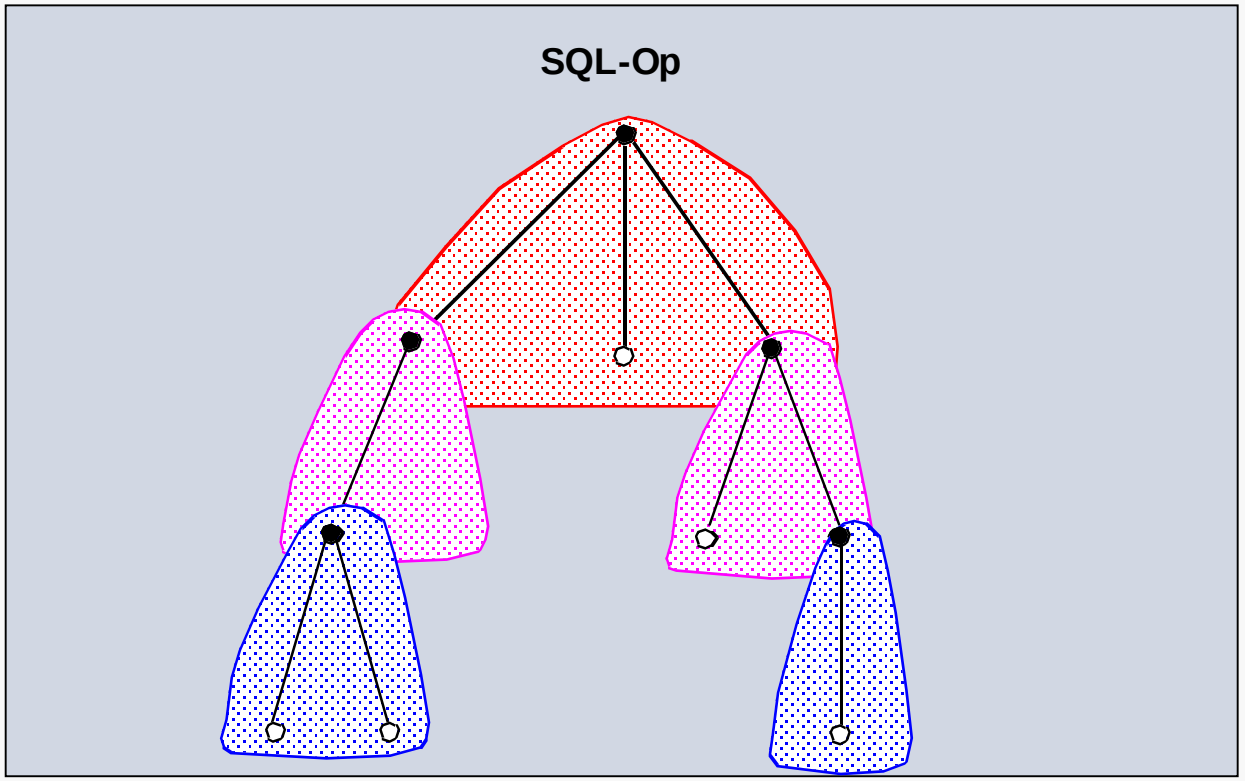
- **Consistency and semantic integrity**
  of the DB is endangered by erroneous data and operations of an application program

- **Isolated execution**
  means "logical single-user mode"

- **Durability**
  requires that modified data of successful transactions must survive any type of failure

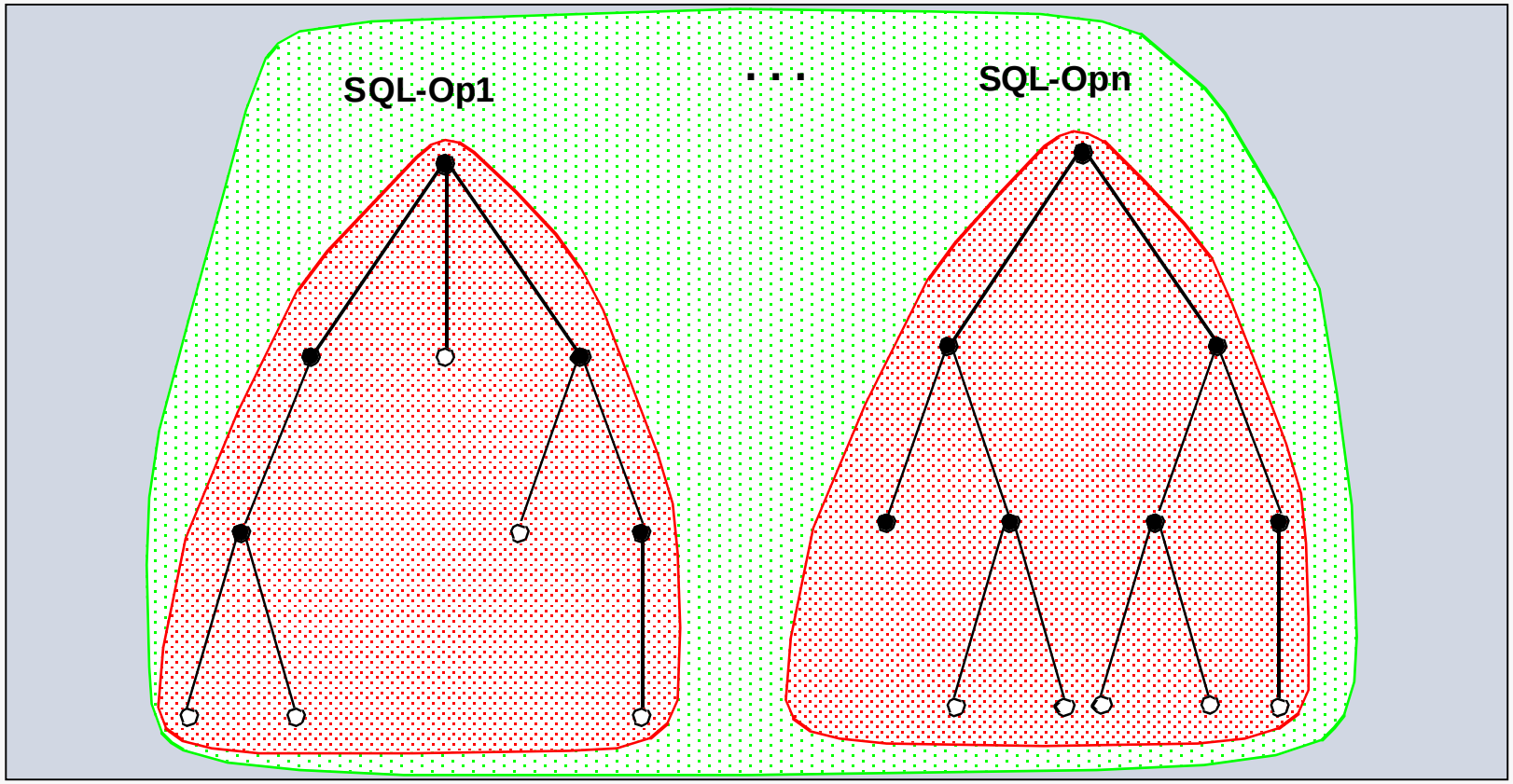
Atomicity is not a Natural Property of Computers

- Transactions are abstractions
  - Building blocks: atomic actions (AAs)
  - AAs are abstractions, too
  - Even if AAs would be implementable in an atomic way, hierarchy of AAs would not!



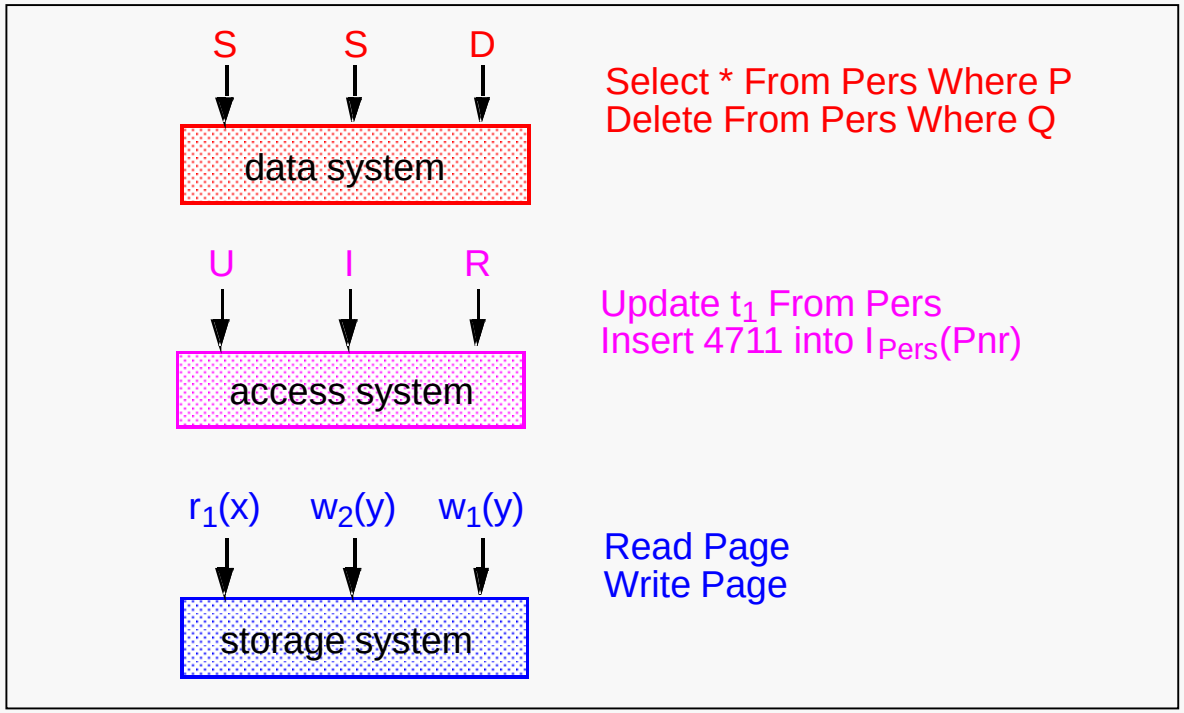
Isolation Requirements of a Transaction

- SQL guarantees "statement atomicity"
  - "Do it twice" or "Do it once"
- And, of course, transaction atomicity
- Key mechanisms:
  - **Concurrency control** and **logging & recovery**



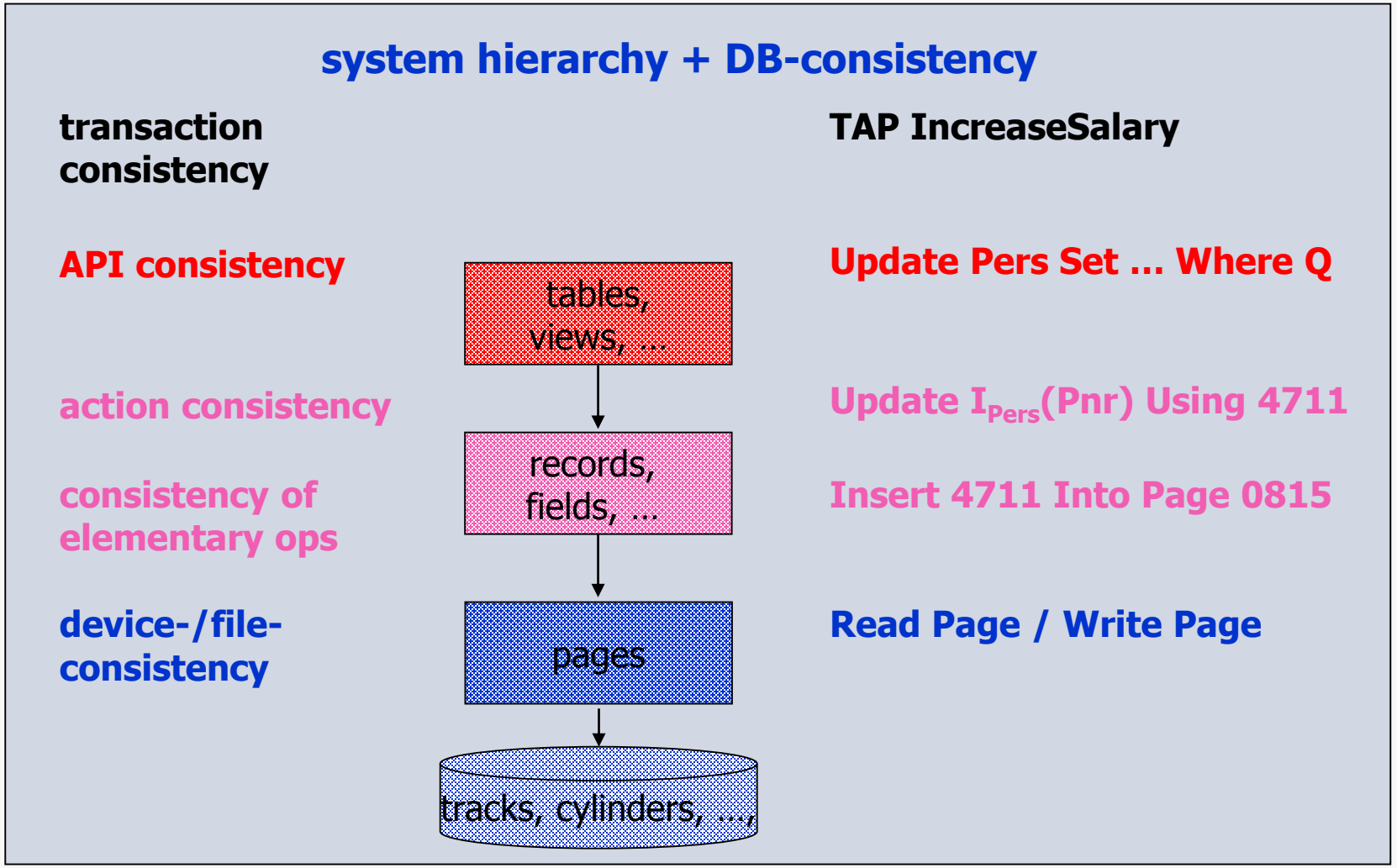
Modeling for Concurrency Control

- Correctness criterion: conflict serializability
- History writer
  - Read/write model (Page Model)
  - Record operations
  - Logical predicates

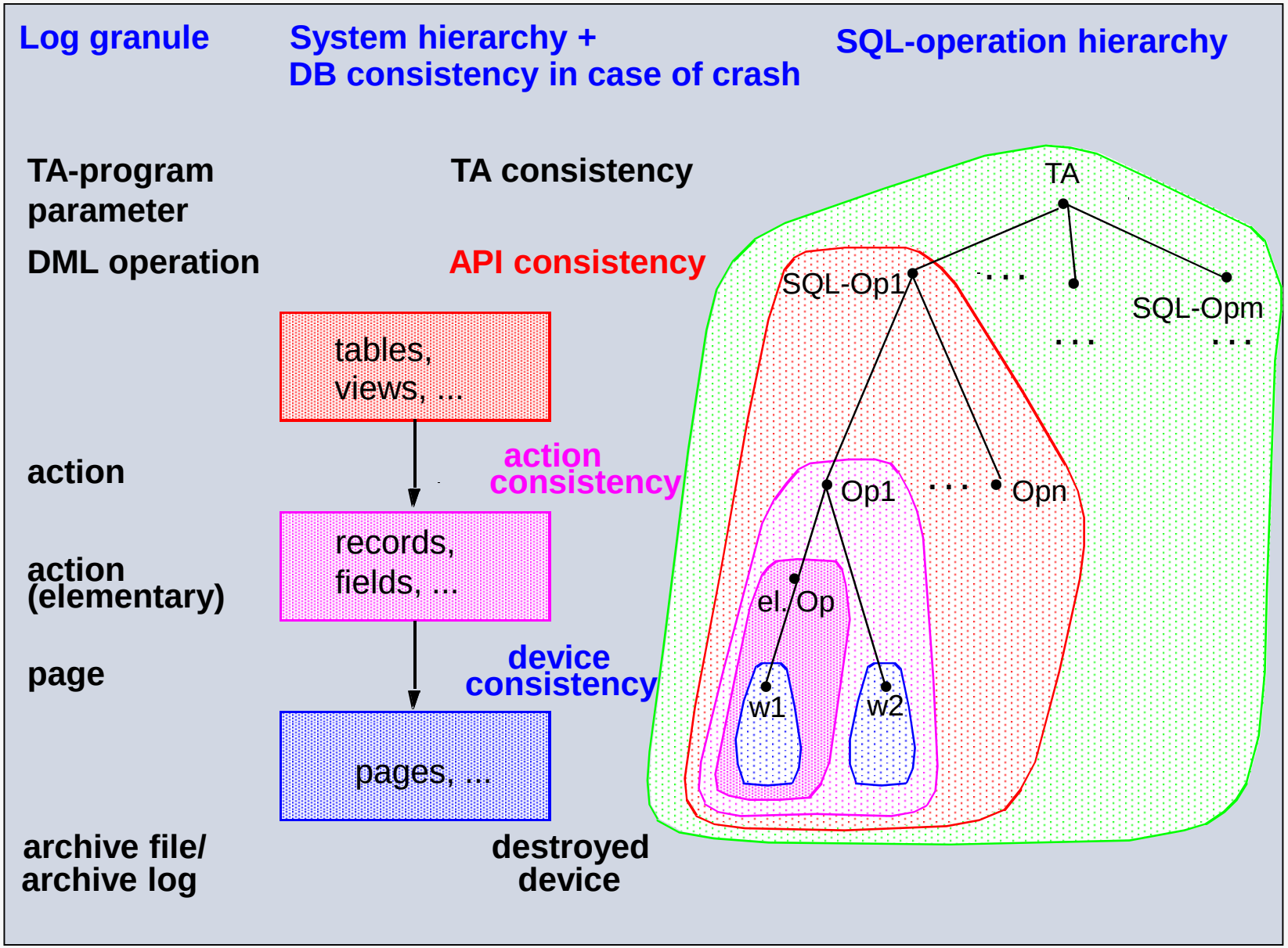


Aspects of DB-Recovery

- Consistency of a layer requires layer-specific consistency of all layers below

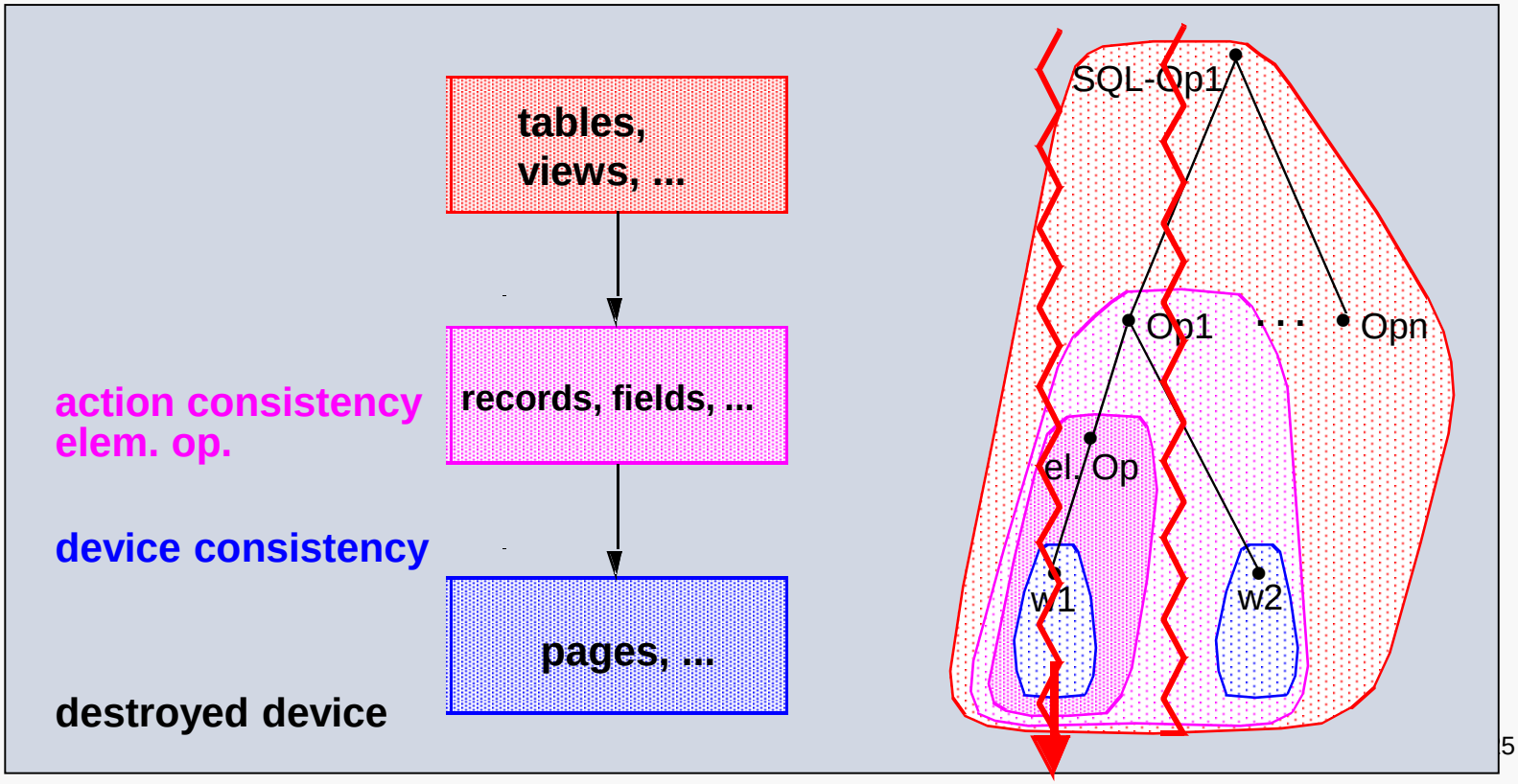


DB Consistency and Logging



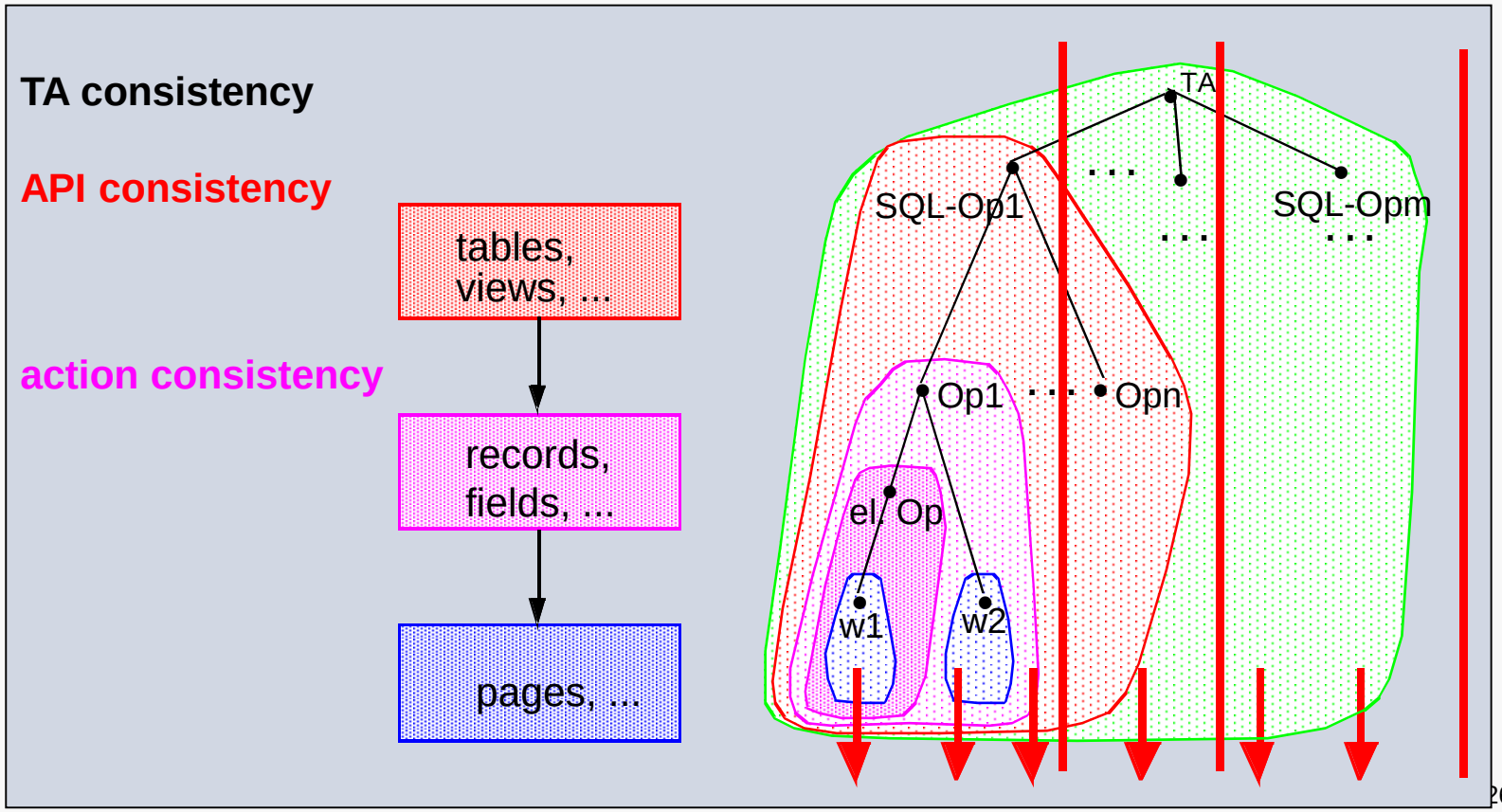
Non-Atomic Propagation Methods

- DB is "chaos consistent" after a crash
- If blocks are destroyed: device recovery
- Otherwise: device consistency as minimal condition, using
  - Page logging: entire pages for Redo/Undo (**extremely expensive**!)
  - Trick: physiological logging using LSNs



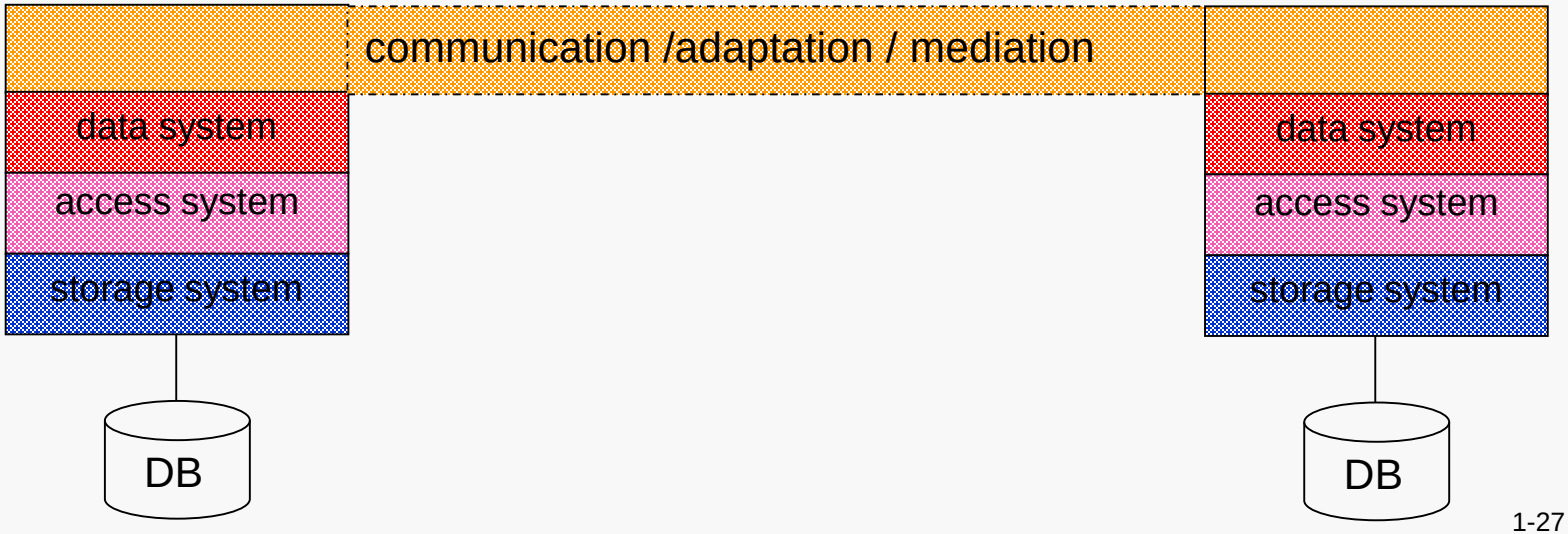
Atomic Propagation Methods

- Operations of higher layers span several pages
  - Log unit must perform Undo/Redo in several pages
  - Pages must be **completely or not at all** in the DB
- Role of checkpoints (limitation of Redo recovery)



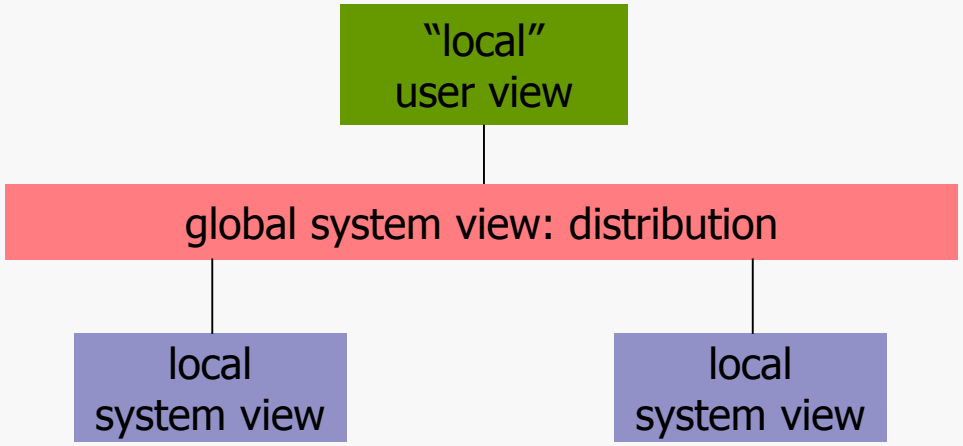
How far does this Architecture Model Reach?

- The **invariants of DBMS processing** determine the mapping steps in the DBMS architecture
  - Layer model aims at the navigational or set-oriented processing of record-structured data
  - Horizontal distribution of DBMS processing in
    - Distributed DBMS (SN/SD)
    - Federated DBMS, Multi-DBS
    - ...

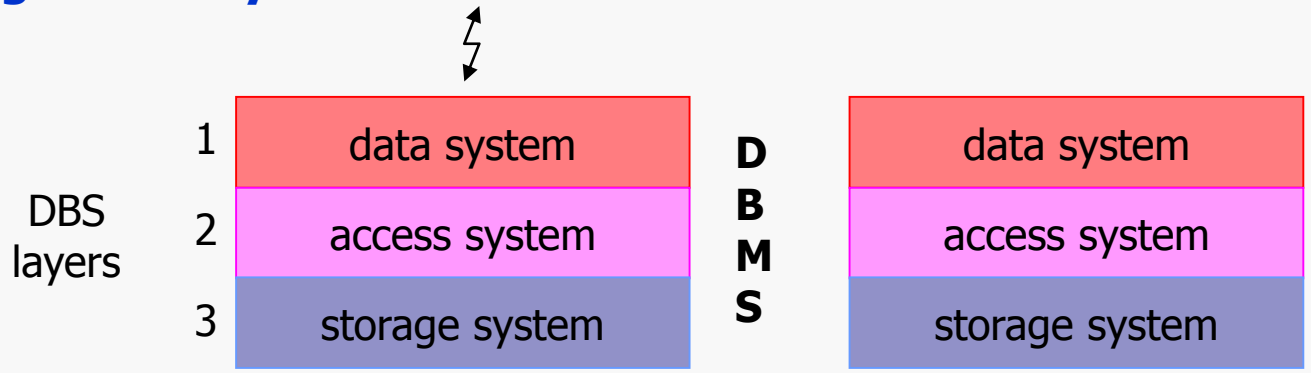


Classification of Distributed DBS

- **Approach to distribution**
  - Partitioning of data (possibly including replication)
  - Communication among processes to access distant data
- **Desired visibility**

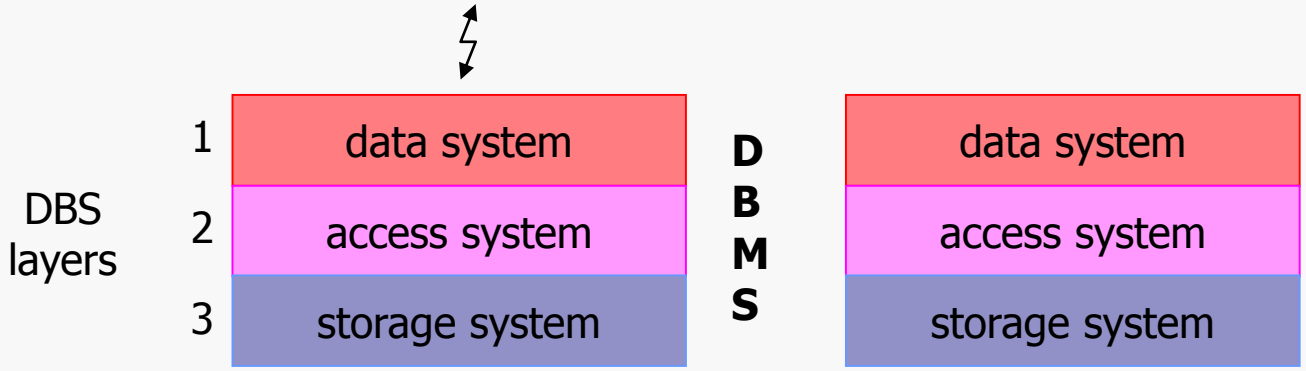


- **Homogeneous system model**



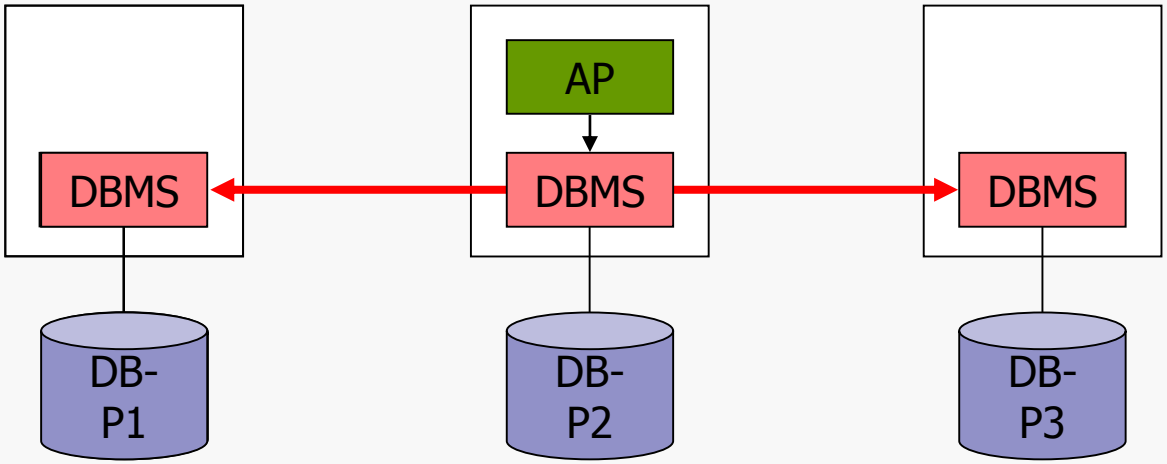
Classification of Distributed DBS (2)

- **Homogeneous system model**



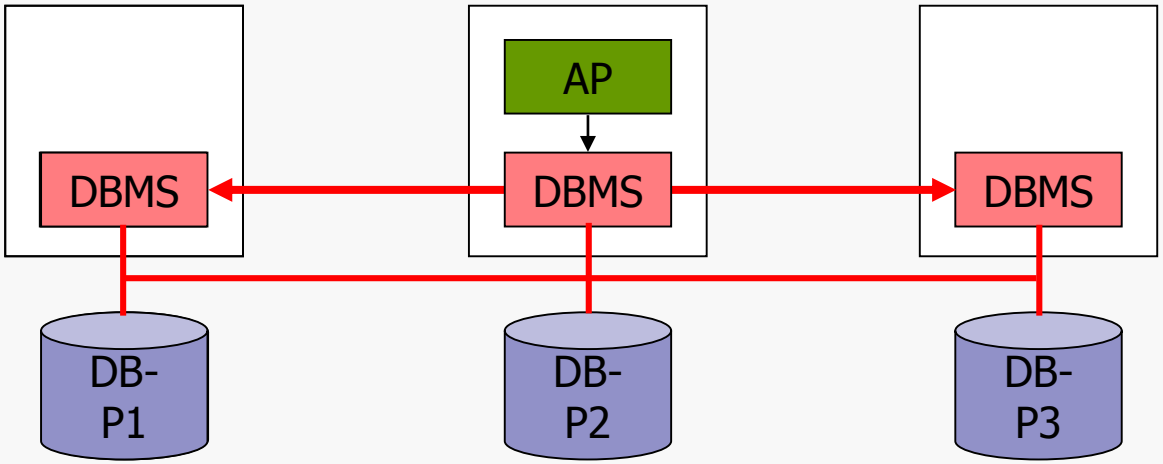
- **Several (homogeneous) realization alternatives**
  - Global system view is obtained by an (additional) DBS layer
  - Distribution of entire DML operations resp. of sub-operations (layer 1)
  - Distribution of internal record operations (layer 2)
  - Distribution of page accesses (layer 3)

Multi-Computer DBS



- **Distribution under control of the DBS**
  - Cooperation among (homogeneous) DBS
  - Location transparency for AP (a single DB schema)*:* *Single system image*
  - *Flexibility for the distribution of data and load*

Multi-Computer DBS (2)



- **Architecture classes**

  1. DB partitioning (*Shared Nothing*, DDBS)

  - Each node owns full functionality and manages one DB partition
  - Data replication increases availability and reliability
    - Minimization of effects of "remote" failures,
    - Failure masking by redundancy
  - Processing principle: **The load follows the data**

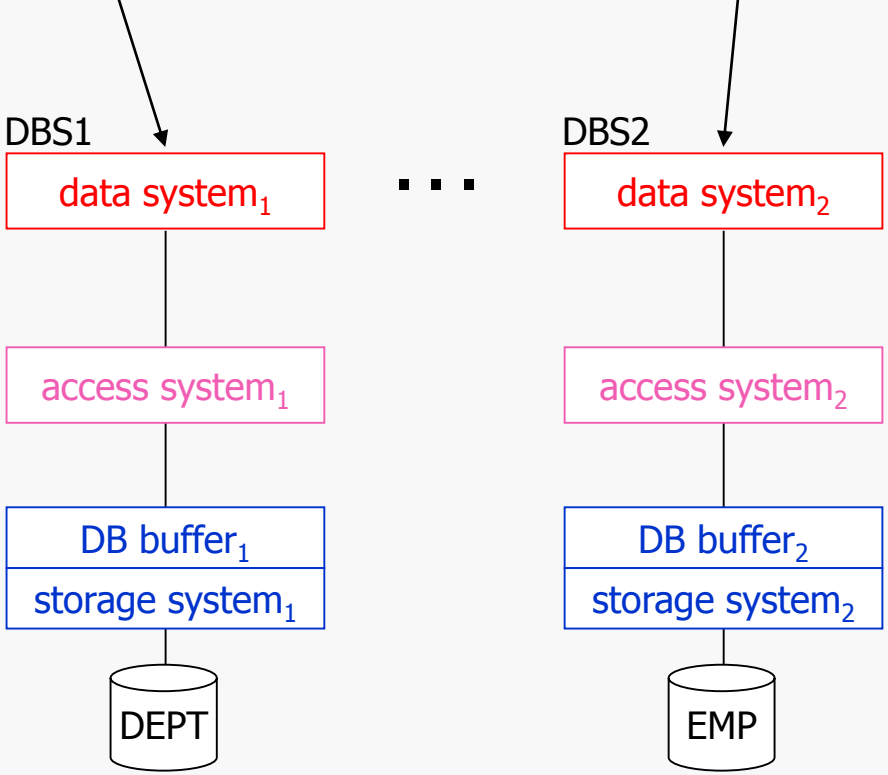
  2. DB sharing (*Shared Disk*)

  - Local execution of DML operations
  - Processing principle: **Data transport to the computer processing the TA**
  - Local reachability of disks (external storage)

Processing Overhead in DDBS

Q1: SELECT ...
FROM EMP E
WHERE E.DNO = 'K55'
AND E.SAL < E.PROV

Q2: SELECT ...
FROM EMP E, DEPT D
WHERE E.DNO = D.DNO
AND E.DNO = 'K55'



Function Request Shipping

	to DBS2	to DBS1
	→ entire Q1 ← Q1 result	→ D.DNO = 'K55' to DBS1 ← result 1 'K55' record
	Scan for all records with E.DNO = 'K55' → request next record ← transmit record	**Scan for all records with D.DNO = ‚K55'** → request next record ← transmit record
	for all pages of EMP file: → request next page ← transmit page	**for all pages of DEPT file:** → request next page ← transmit page

Assumptions:

- EMP file has 10^5 pages
- E.DNO = 'K55': 100 records in EMP
- DEPT file has 10^3 pages
- E.DNO = 'K55' AND E.SAL < E.PROV: 5 records in EMP

Layer Models for Client/Server DBS

- **Critical for the DB-related performance capabilities**
  - Kind and complexity of DB operations (set-oriented, navigational)
  - Exploitation of reference locality when accessing data

- **Processing concept so far**
  - In all architectural proposals, "shipping of the queries" to the (server) DBS was assumed (query shipping approach):
  - DML statement is sent to the server
  - After its processing, the result is made available to the application (client)
  - **Important property:** there is no replication of DB data

  ↪ No exploitation of inherent reference locality in the client

Layer Models for Client/Server DBS (2)

- **Further processing concept:**

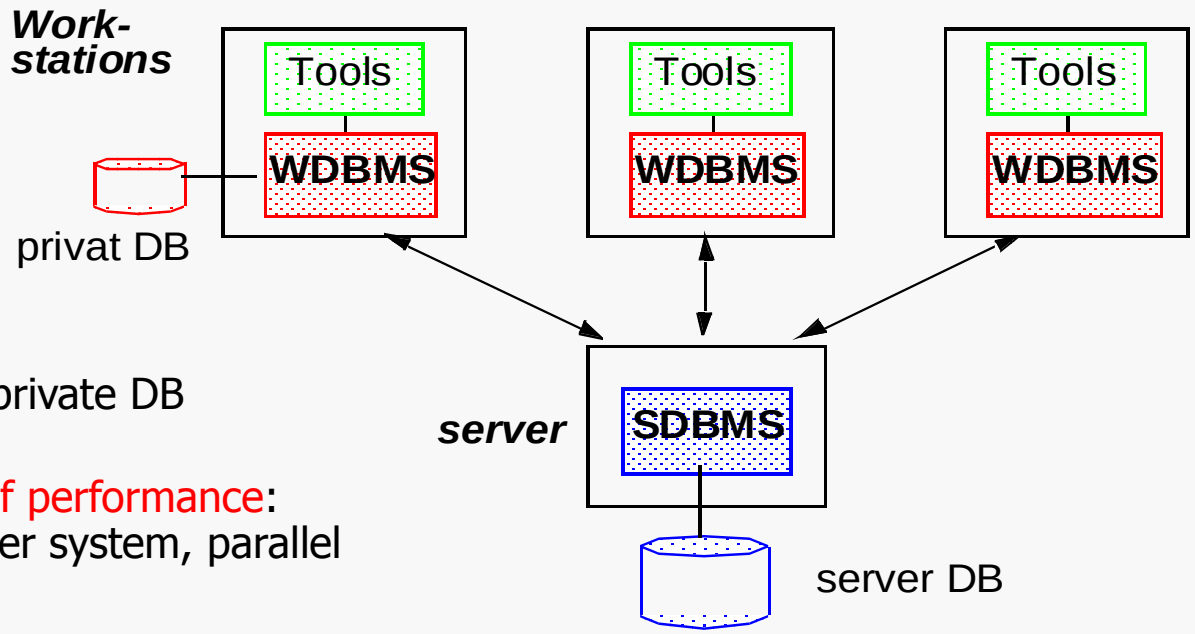
  The following approach is applied by most of the object-oriented DBS (OODBS) (data shipping approach):

  - All data needed for the evaluation of a query, are transported to the client and are buffered there
  - Then, the query (or a sequence of queries) is evaluated in the client buffer
  - **Note:** The query complexity is severely restricted
    - Typically only navigation operations
    - Queries with simple search arguments (SSA) are sometimes supported, but no join operations, aggregations, ...

  ↪ This concept is primarily advantageous in case of high reference locality. It is utilized by the Checkout/Checkin concept

Layer Models for Client/Server DBS (3)

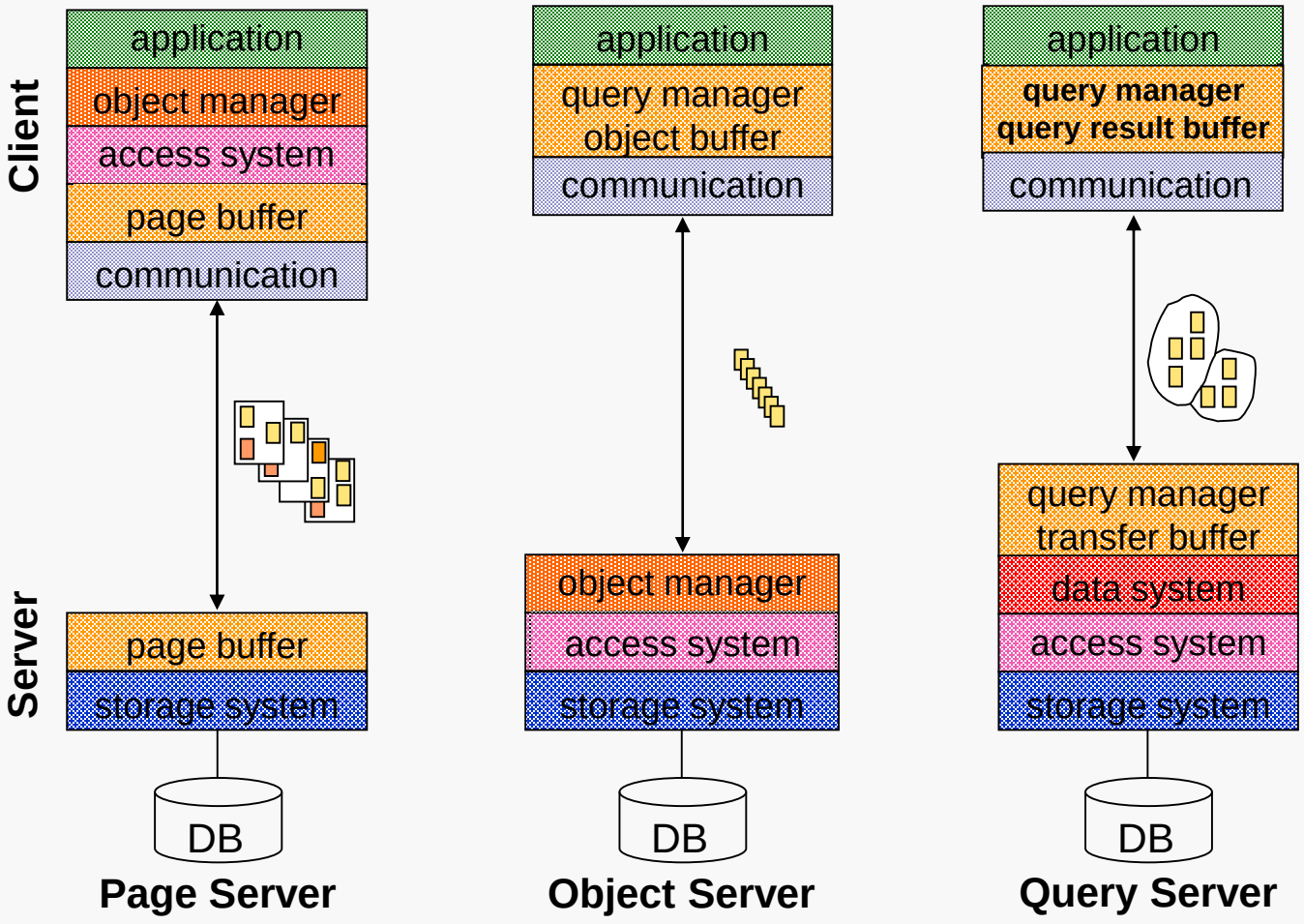
- **Advantages of data shipping**
  - Local data buffering reduces interactions between client and server (reduction of network load; avoidance of repeated server queries)
  - Aggregated computer power and totally available storage capacity of the system are increased
  - Server load is reduced
  - System scalability is improved
- **Client/Server architecture: principle**



  - Local disks for private DB and log data
  - Enhancement of performance: distributed server system, parallel DB processing

Layer Models for Client/Server DBS (4)

- **Client/Server architectures for object-oriented DBS**



- Vertical distribution of layers: DB functionality in Server and Client
- Seamless connection with the application program, but many clients

↪ In object buffers, typically navigational processing is performed!

DBS Use in Design Environments

- Engineering applications (CA*) need local data management
  - Exploitation of reference locality during processing
  - Long-term storage (object buffer) of large data sets in main memory
  - Local representation of objects at the client window (screen)
  - Fast transformation and processing of complex objects

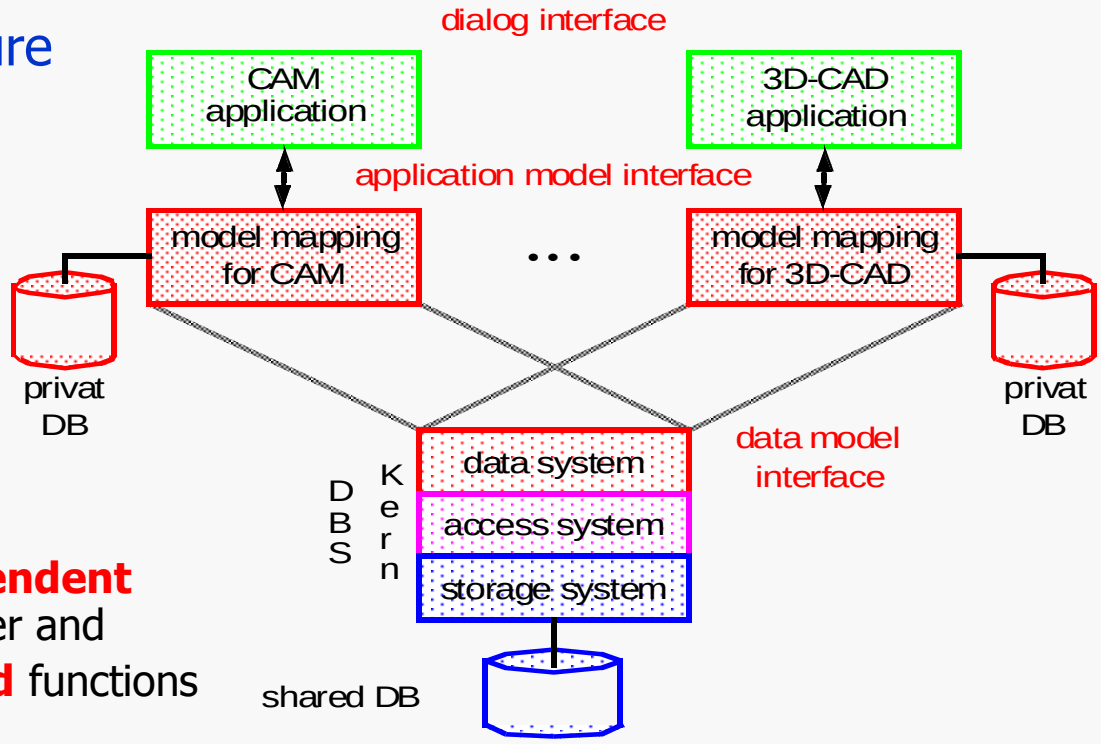
  ➥ DBS which should support design applications, need appropriate functionality for data management / transaction support at client and server side

- New requirements
  - Design tasks (CA*), knowledge-based applications, ...
  - DB-supported processing of large data sets in client
  - Long transactions
  - Large data sets and reference locality

  Consequences:
  - Data management in client and server
  - Replication of data in DB buffer and object buffer
  - Increased overhead for the integrity control

DBS Use in Design Environments (2)

- **Workstation-oriented or client/server DB architectures**
  - They should support so-called Non-Standard applications
  - Mapping of the DB processing onto client/server architectures
- DB kernel architecture



- **Separation of**
  - **Application-independent** functions in the server and
  - **Application-related** functions in the client
- **Key concepts**
  - Data model for complex objects at the server interface
  - Tailor-made data model at the application interface
  - Object buffer to exploit reference locality and to save communication requests

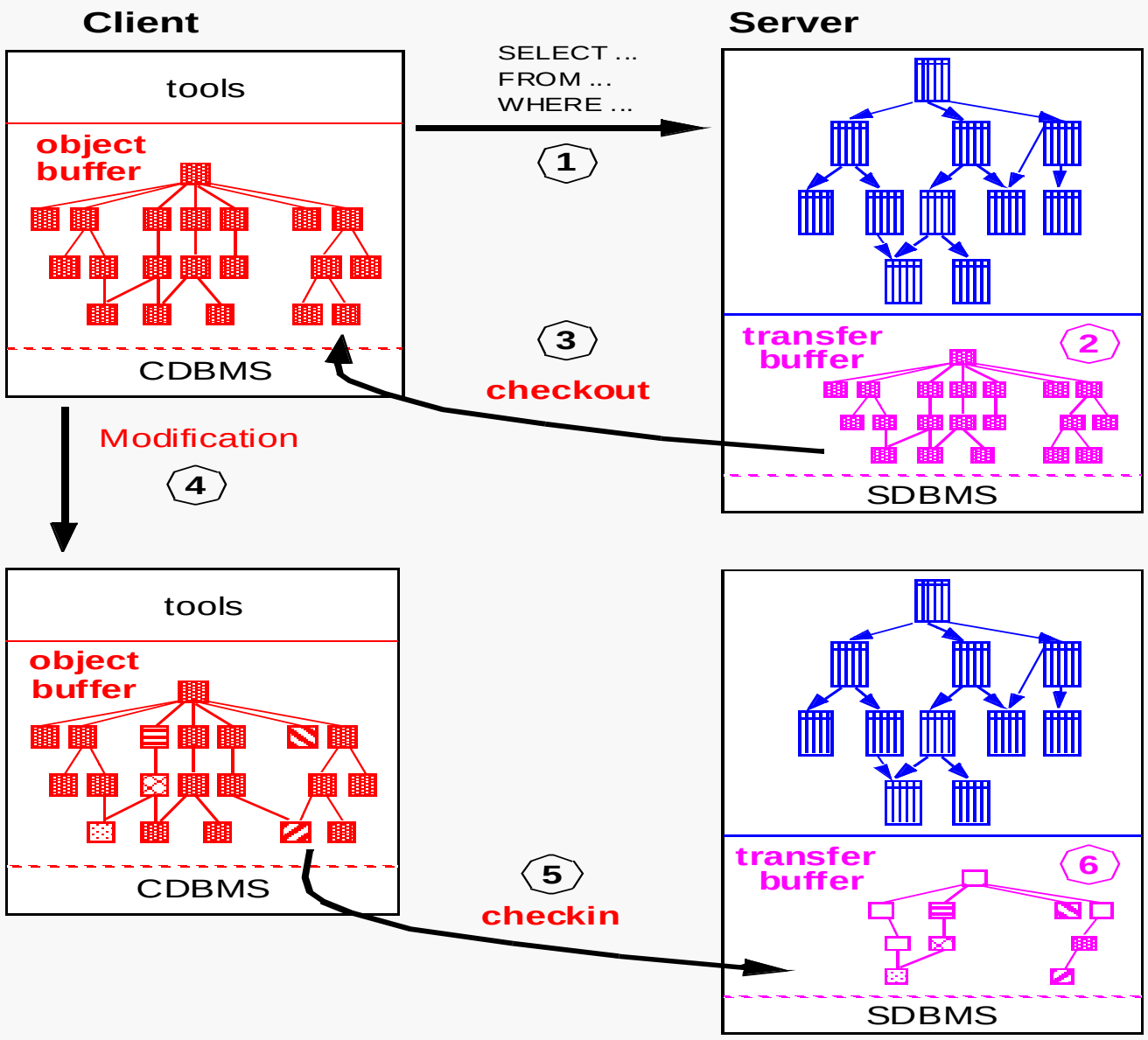
Processing Principle: Load, Operate, Integrate

- **Data requirements for a design transaction**
  - Data loading by one or several Checkout requests
  - Design objects are stored into the object buffer
  - Use of special storage structures and access paths
- **Data manipulation**
  - In case of complex-structured objects: use of hierarchical cursors
  - Entire object processing is local: accumulation of updates
  - Application model: set of ADTs
- **Processing control and recovery measures**
  - Client DBS propagates persistent intermediate state in a private DB
  - Local rollback measures by Savepoints (SAVE, RESTORE)
  - Interruption of processing (SUSPEND, RESUME)

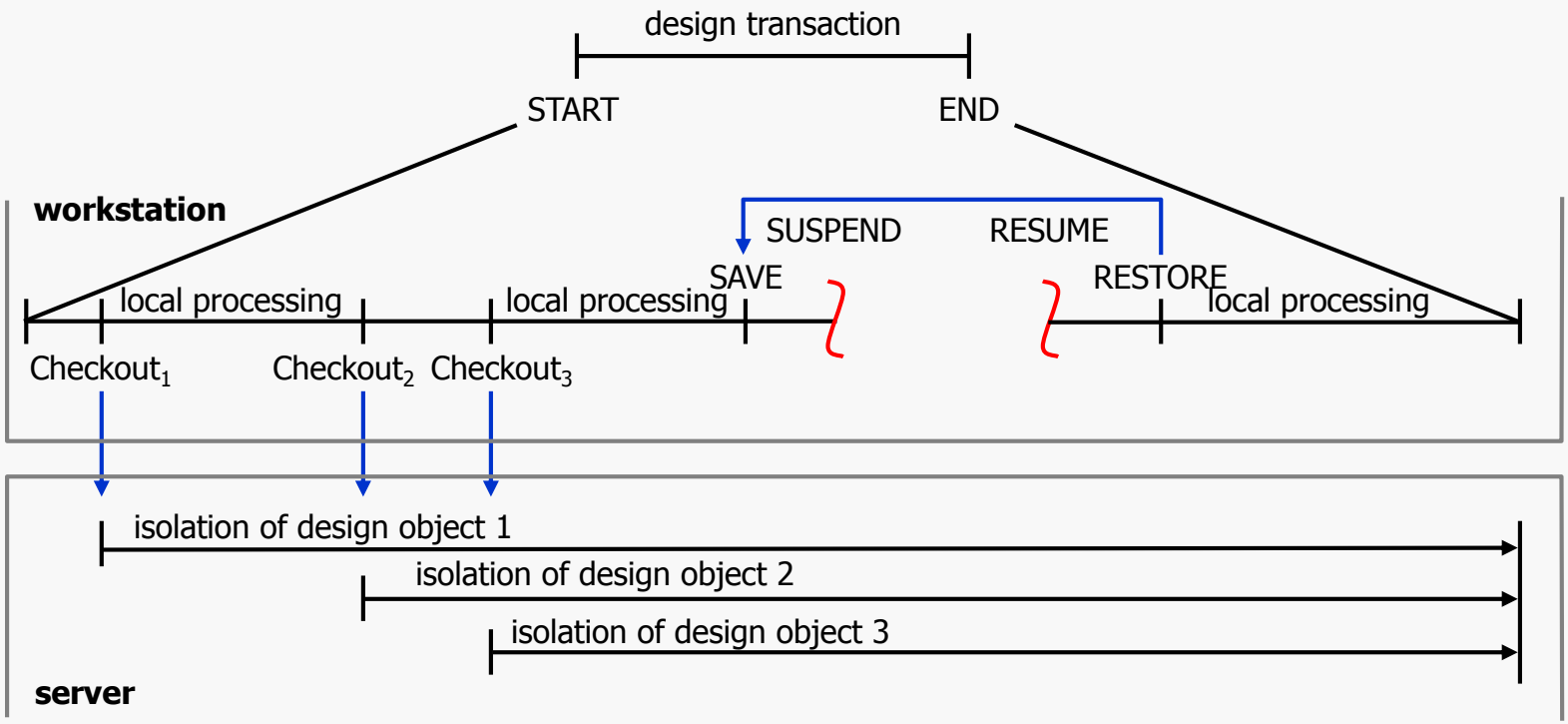
  ➥ What about the ACID properties in case of design transactions?
- **Data propagation**
  - By Checkin at the end of the design transaction,
  - Then, (deferred) checking of integrity conditions is performed in the server DB

  ➥ What happens when a violation of integrity is detected?

Load, Operate, Integrate



Model of a Design Transaction



Characteristics:

0 .. n	Checkout
0 .. 1	Checkin
long duration	

Storage of intermediate states of a design transaction to:

- To interrupt processing (SUSPEND, RESUME)
- Rollback to earlier processing states (SAVE, RESTORE)

Summary

- **General aspects of the layer model**
  - Layer model is general explanation model for the DBS implementation
  - Layer mapping can be refined/coarsened when needed: applicability for distributed DBS, client/server DBS, ...
  - Design of suitable interfaces requires great implementation experience
  - Concrete implementations sometimes violate the layer isolation for performance reasons
    ($\rightarrow$ critical functions have "reach-through")
- **Implementation concepts**
  - The fundamental implementation concepts of centralized DBS can be also found in each node of a distributed DBS
  - Knowledge of implementation concepts enables the objective evaluation and comparison of existing DBS
  - Their accurate knowledge is a prerequisite for performance analyses resp. estimation of the system behavior
  - By identifying few, fundamental concepts, one obtains a deepened understanding of what is meant by **"data independence"**

Summary (2)

- **DB caching**
  - Caching vs. replication for the support of Web-based DB-Apps
  - Caching is the proven technique to improve scalability and performance in large, distributed systems!
  - Columns of tables can be attributed in a DB cache with the property "column complete"
  - Cache Groups support join operations in case of DB caching
- **Classification of distributed DBS**
  - Single System Image
  - Flexibility for data- and load distribution
  - Multi-computer DBS as Shared Nothing or Shared Disk Systems
- **Client/server systems**
  - Data Shipping vs. Query Shipping
  - Data Shipping is primarily advantageous in case of high reference locality. It is employed by the Checkout/Checkin concept
  - Object-oriented DBS as Page, Object, or Query server
  - Typical form of processing in the client is navigational (for embedded systems and design systems)
  - Updates of DB data required in client and server

Further References

- *Chamberlin, D. D.; Astrahan, M. M.; Blasgen, M. W., Gray, J. et al.:* A History and Evaluation of System R. Commun. ACM 24(10): 632-646 (1981)
- *Härder, T.; Reuter, A.*: Principles of Transaction-Oriented Database Recovery. ACM Comput. Surv. 15(4): 287-317 (1983)
- *Härder, T.; Reuter, A.*: Concepts for Implementing a Centralized Database Management System. Proc. Int. Computing Symp. on Application Systems Development, 1983, Nürnberg, B.G. Teubner-Verlag, 28-60
- *Härder, T.; Reuter, A.*: Database Systems for Non-Standard Applications. Proc. Int. Computing Symp. on Application Systems Development, 1983, Nürnberg, B.G. Teubner-Verlag, 452-466
- *Härder, T.; Reuter, A.*: Architektur von Datenbanksystemen für Non-Standard-Anwendungen. BTW 1985: 253-286
- ***Härder, T.:* DBMS Architecture – New Challenges Ahead. Datenbank-Spektrum, dpunkt-Verlag, Heft 14, Aug. 2005, pp. 38–48.**
- *Härder, T., Rahm, E.*: Datenbanksysteme: Konzepte und Techniken der Implementierung, 2nd edition, Springer 2001, Kap. 1