Outline

- Introduction
- Background
- Distributed DBMS Architecture
- Distributed Database Design
- Semantic Data Control
- Distributed Query Processing
- Distributed Transaction Management
- Distributed Database Operating Systems
- Open Systems and Interoperability
  - Client-server architectures
  - Multidatabase Systems
  - Standards
- Parallel Database Systems
- Distributed Object Management
- Concluding Remarks

Changing Applications

- Traditional applications (business)
  - simple data types
  - structured data
  - efficient data access
  - data sharing and security
- well supported by relational systems
- New applications (OIS, CAD/CAM, AI, Medical imaging, etc.)
  - richer database programming languages (types)
  - complex type structures
  - general rules and assertions
  - others: dynamic schema, multimedia, triggers, alerters

not well supported by relational systems

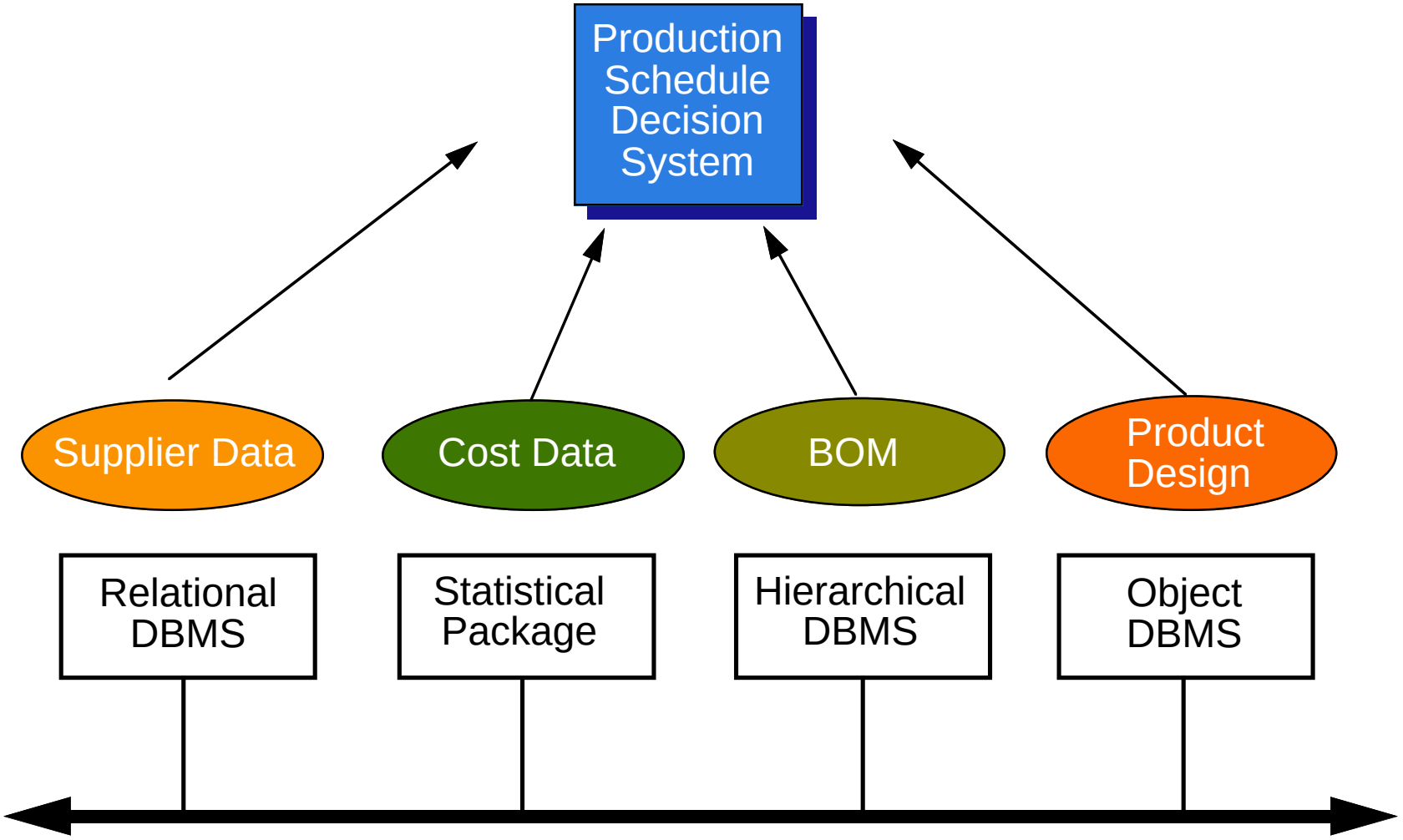
Technological Developments

- Improvements in computer networking technology
  - local area networks becoming much faster
    - FDDI - 100 Mbps
  - wide area networks becoming faster, reliable, and far reaching
    - Internet, T1 lines, digital phone lines, ATM technology
- Developments in workstation technology
  - much better price/performance
  - increased capabilities – parallel workstations
- Maturation of distributed data management technology
  - early multiple client/single server systems abundant
  - multiple client/multiple server systems emerging
  - we have started to understand the tradeoffs better

Likely Future Scenario

- We will still need to use relational systems for typical data processing applications
- We will need to use a more sophisticated data management system for advanced applications
- The problem of "legacy systems" will continue to exist
  - Remember – 45% of mainframe DBMSs are still IMS.
  - A large number of applications still use file systems.
- We will be required to incorporate non-data management systems (e.g., imaging systems, word processors, etc) into our environment

Likely Future Environment



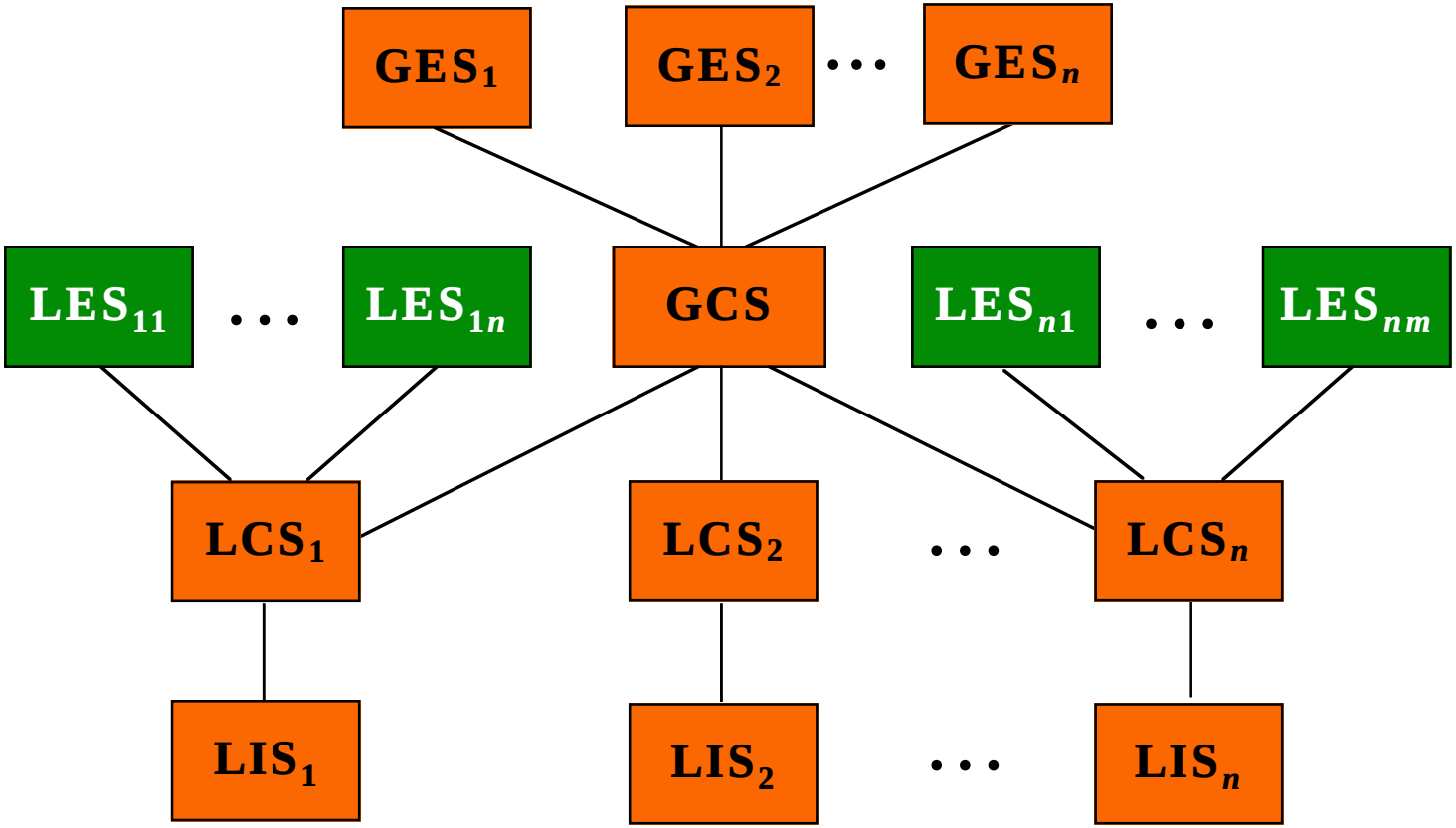
The Demand for Interoperable DBMSs

- Consequences of trends in distributed systems, networking and DBS
  - independent development of DB by different user groups
  - existing DB will still survive
  - DB accessed by various tools
- Difficulties:
  - hard to know which data is useful and its location
  - hard to integrate rapidly heterogeneous data
  - hard to combine the activity of heterogeneous tools
- Interoperability:
  - intelligent interaction between heterogeneous systems (e.g., DBS, spreadsheet)
  - requires the combination of databases, distributed systems, window editors, etc.

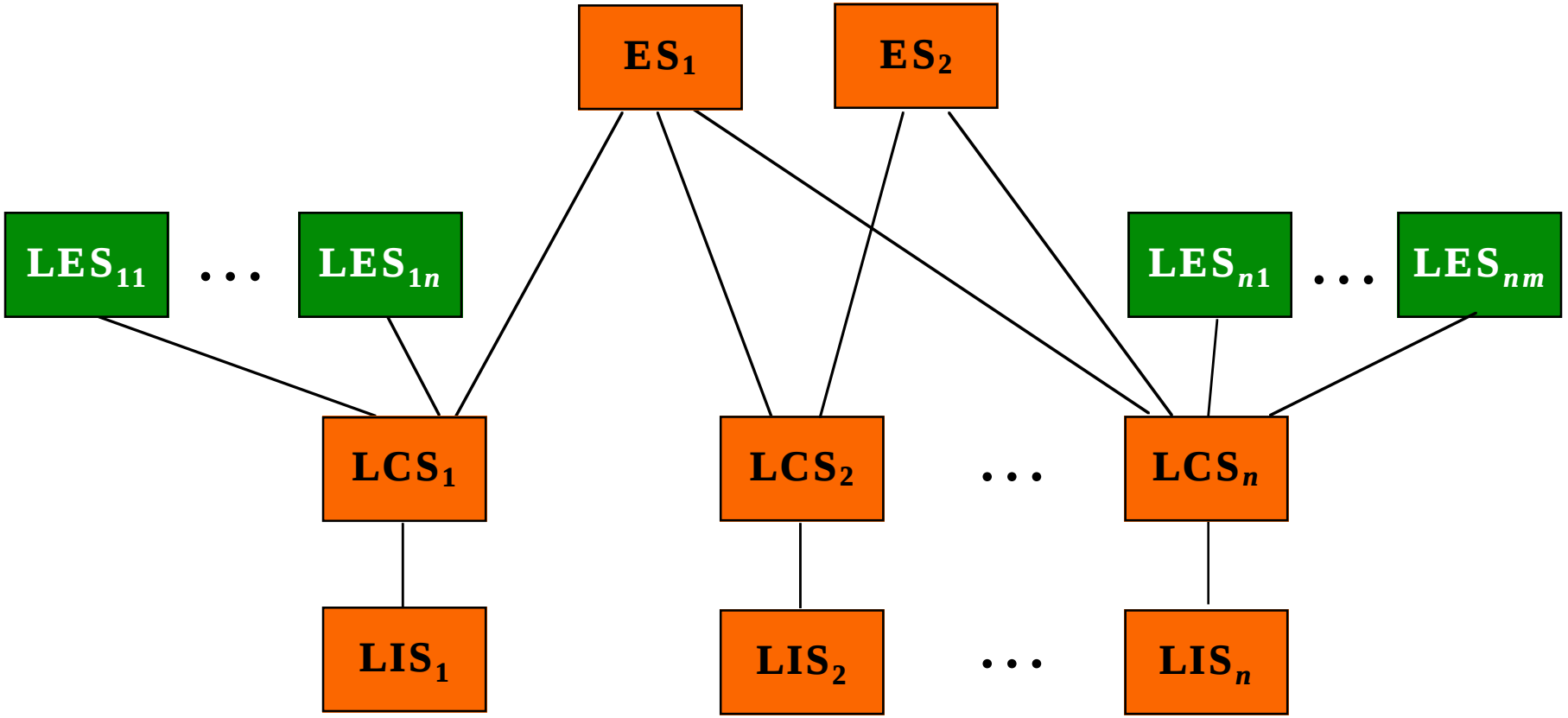
Objectives of Interoperable DBMSs

- Interoperability of existing, heterogeneous and distributed databases, as in distributed MDBS
- Integration of both conventional and unconventional data types, as in object-oriented DBMS
- Interoperability of existing applications
- Ability of using existing sofware components in defining object methods
- Ability to invoke arbitrary application programs, and to create arbitrary combinations of invoked application functions

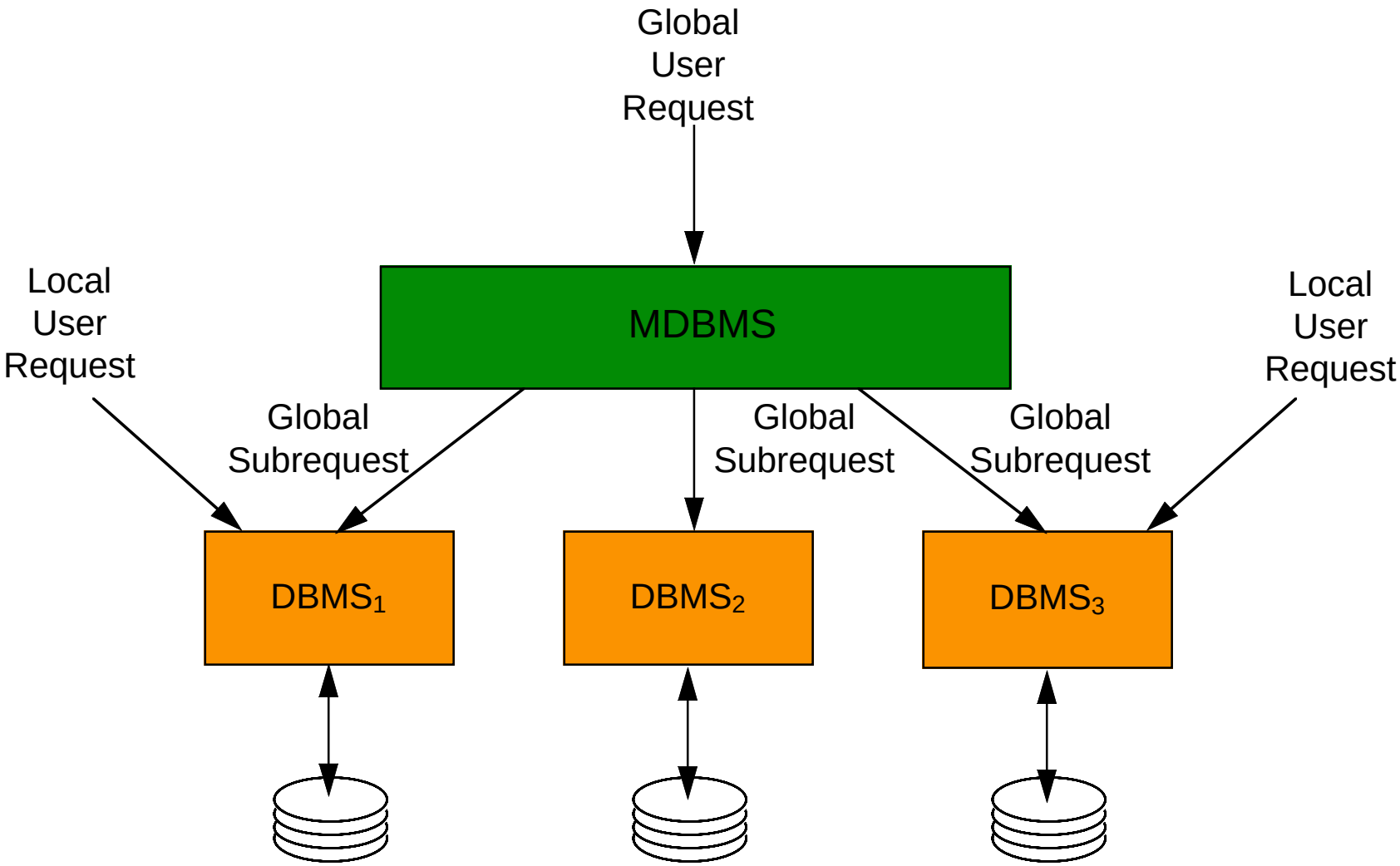
Datalogical Multi-DBMS Architecture



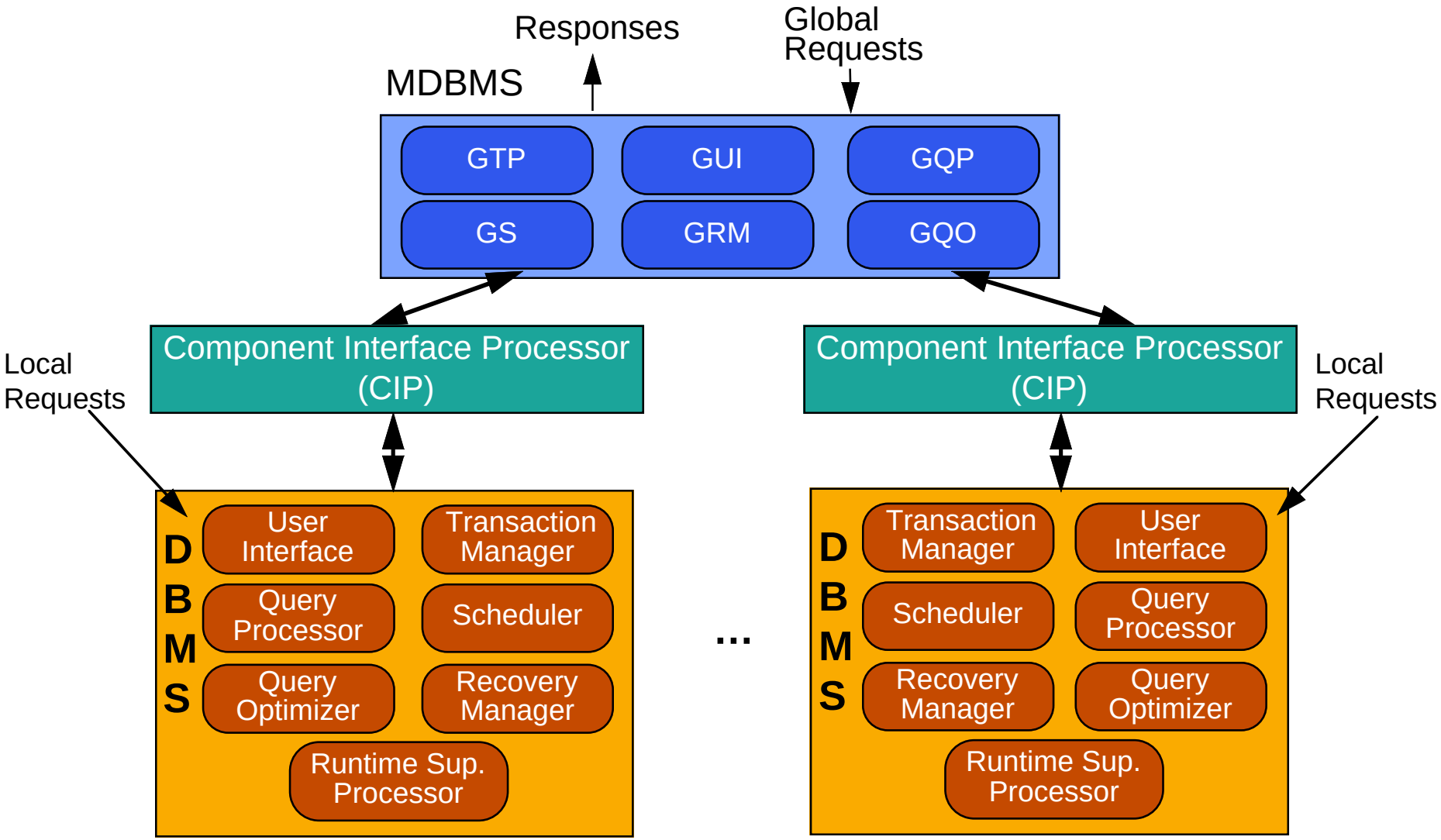
Datalogical Multi-DBMS Architecture – No GCS



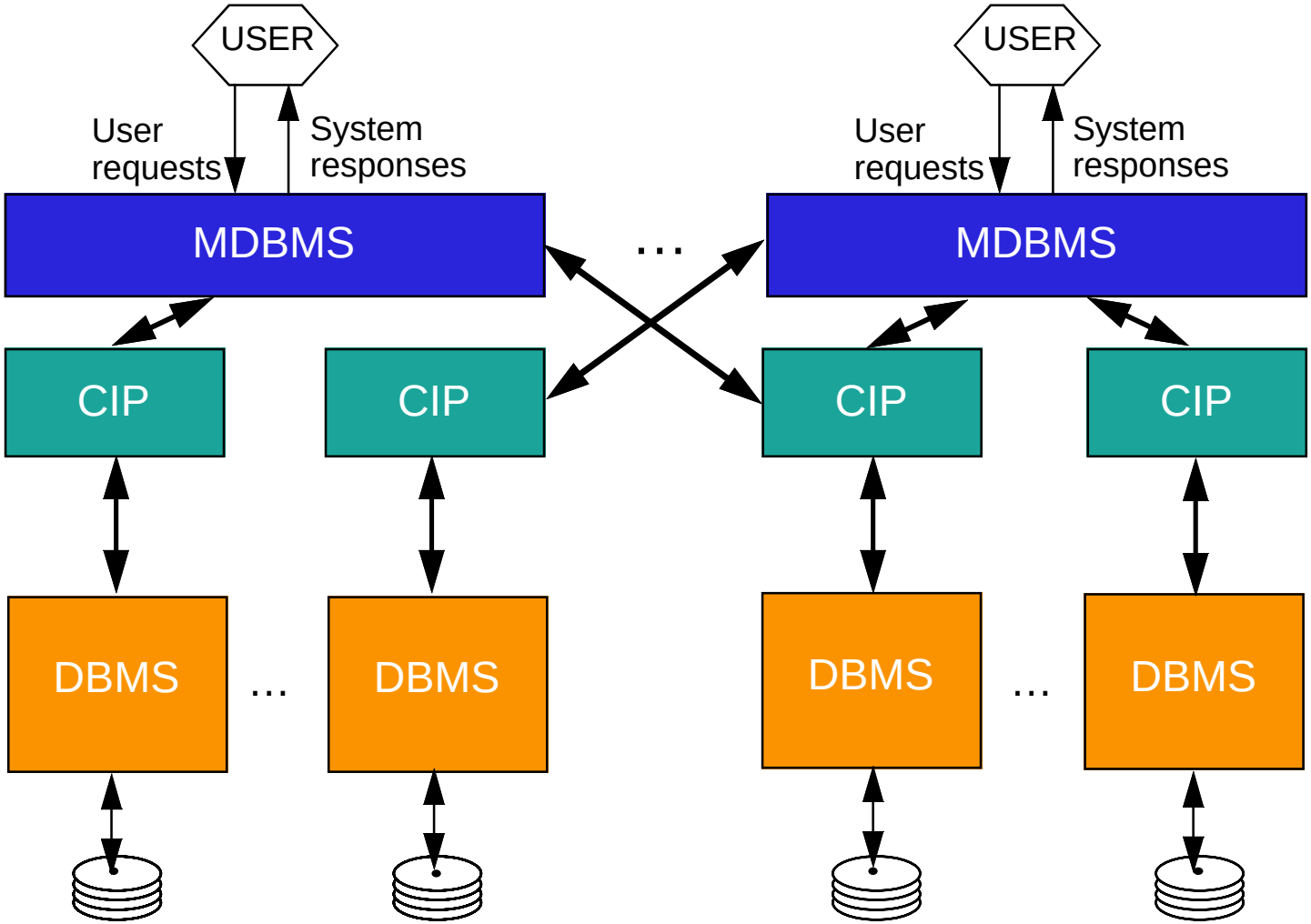
Multi-DBMS Execution Model



Components of a Multi-DBMS



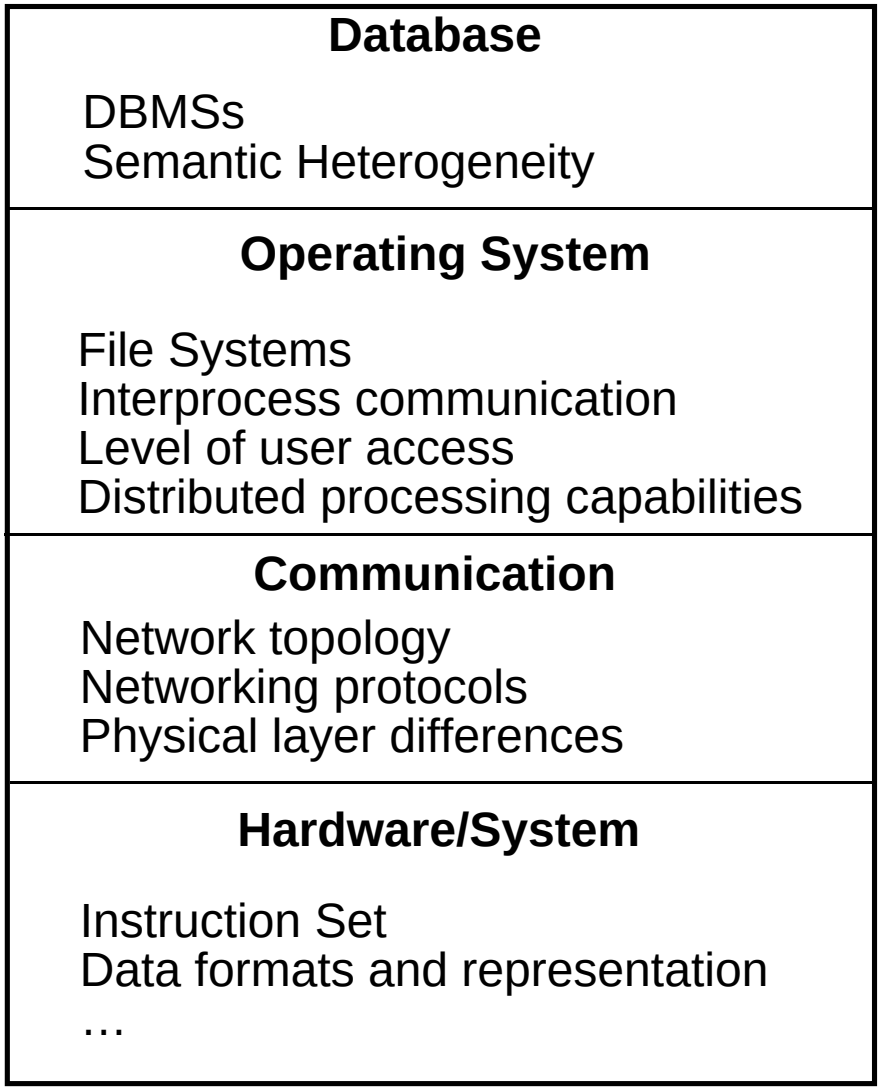
Distributed Multi-DBMS Architecture



Dimensions of the Problem

- Distribution
  - We have dealt with this
- Heterogeneity
  - Various levels
  - DBMS Important one
- Autonomy
  - Not well understood
  - Various versions
  - Most difficult

Heterogeneity



Autonomy

- Design autonomy
  - Ability of a component DBS to decide on issues related to its own design.
  - Data, representation, interpretation, implementation, etc.
  - Also called physical autonomy.
- Communication autonomy
  - Ability of a component DBMS to decide whether and how to communicate with other DBMSs.
  - Our definition of a multi-DBMS says it doesn't communicate directly.
- Execution autonomy
  - Ability of a component DBMS to execute local operations in any manner it wants to.
  - Also called site autonomy.

Issues

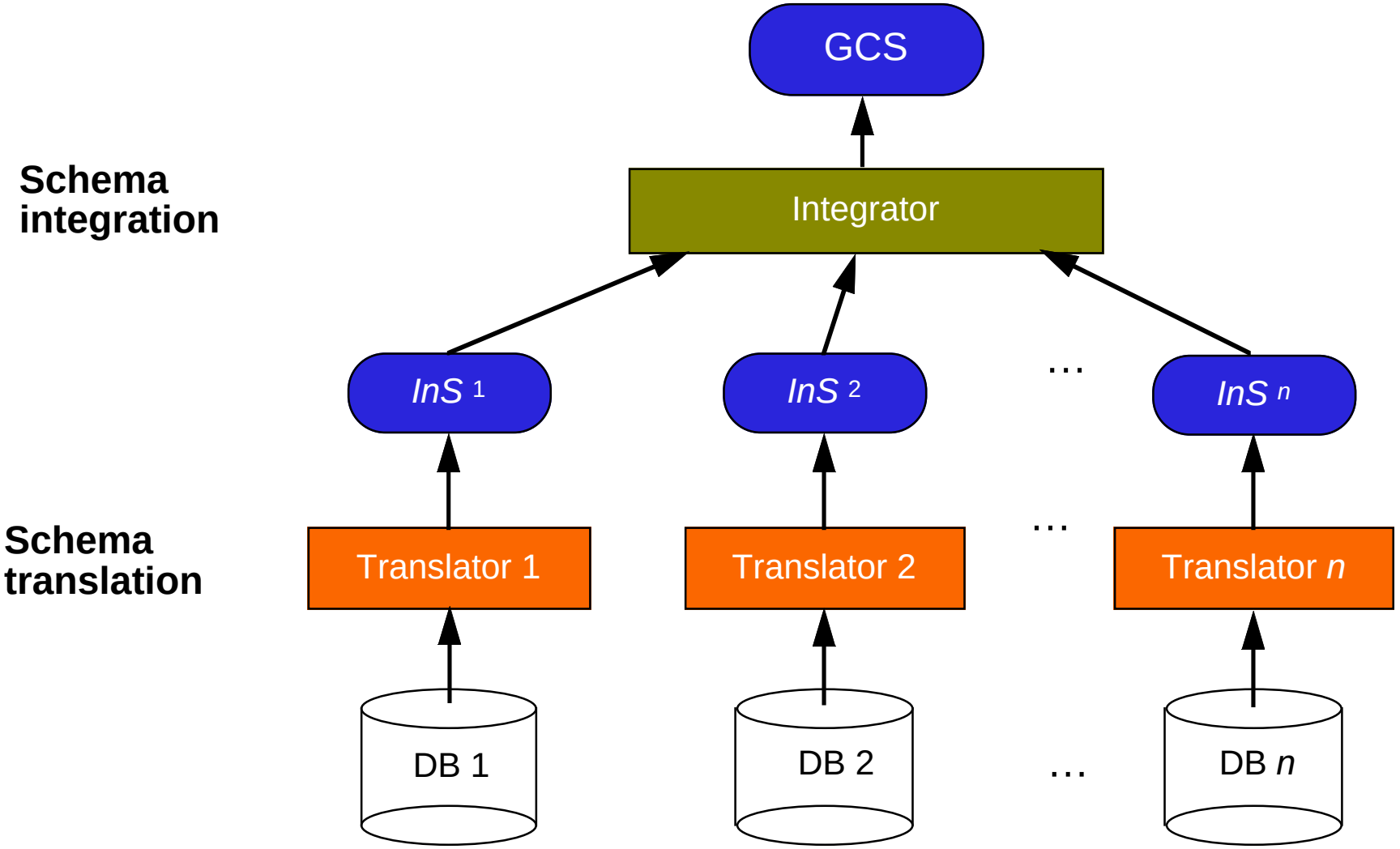
- Database integration
- Query processing
- Transaction management

Database Integration...

- the process by which information from participating databases can be conceptually integrated to form a single cohesive definition of the multidatabase.

- It is the process of defining the global conceptual schema.

Integration Process



Multidatabase Query Processing

- The complexity of query processing is higher due to:
  - Capabilities of individual DBMSs may be different
  - Cost of processing queries may be different
  - May be difficult to move data between DBMSs
  - Local optimization capability may be different

Heuristic-Based Query Optimization

- Decompose global query into smallest possible subqueries each of which is executed by one DBMS. Multiple subqueries may be submitted to a given DBMS. Global query processor collects partial results.
- Global processor does more work, too many messages, simple.
- Decompose global query into largest possible subqueries each of which is executed by one DBMS. Single subquery to each DBMS. Global query processor collects partial results.
- Global processor does less work, fewer messages, more sophisticated component interface processor.
- Component interface processors and the component optimizers are more involved in processing and optimization. CIP's can exchange messages among each other, can store and delete temporary files, and operate on them.

Types of Multidatabases

- Proprietary DBMS
  - Component DBMSs are from the same vendor and make their cost functions and database statistics available to the global optimizer.
  - Similar to the distributed DBMS query optimization.
- Conforming DBMS
  - Components from a different vendor; components can divulge some database statistics, but not the cost function.
- Non-conforming DBMS
  - The DBMS is a black box, incapable of making available any information about its query optimizer.

Types of Components

- Primary components
  - are relational DBMSs
    - support an SQL interface
    - some of them can store temporary relations
- Secondary components
  - None of these are true

Cost-Based Optimization

- Solution space
  - The set of annotated algebra expressions (query trees).
  - The algebra has to be extended with a *move* operator indicating the shipment of data
  - Some of the alternatives are invalid since data cannot be shipped to secondary components
  - Consider bushy trees: should consider as much local work and as much parallelism as possible
- Cost function (in terms of time)
  - Most difficult issue
  - Cost for level 1 (from the bottom) have to come from component DBMSs
  - Cost for higher levels can be calculated recursively
- Search algorithm
  - Similar considerations to distributed DBMSs

Determination of Component Costs

Pegasus project at HP Labs

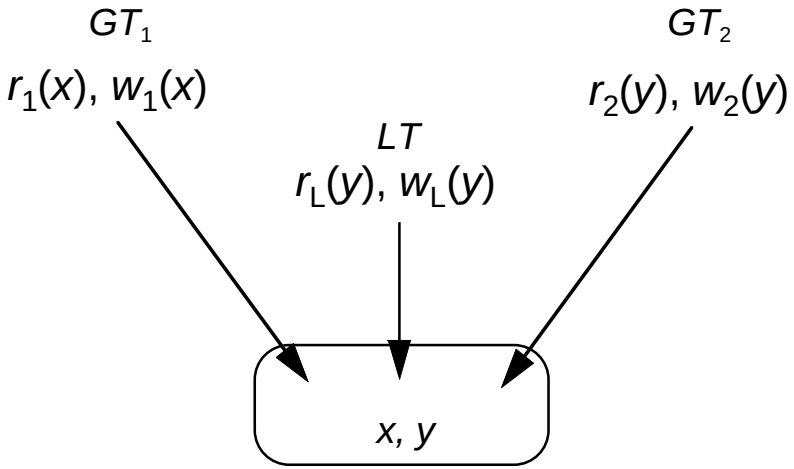
- Cost Function
  - Express cost functions logically (e.g., aggregate CPU and I/O costs, selectivity factors), rather than on the basis of physical characteristics (e.g., relation cardinalities, number of pages, number of distinct values for each column).
  - Cost = initialization cost + cost to find the qualifying tuples + cost to process selected tuples
- Calibrating database
  - To determine the coefficients of this cost formula, create a statistically "correct" calibrating database (consisting of one relation), run queries against it (in isolation) and measure the elapsed time.

Transaction Management Conditions

- Individual DBMSs guarantee local synchronization atomicity (serializability)
- Local transaction managers maintain the relative execution order of the subtransactions determined by the global transaction manager
- Somehow guarantee that commit/abort decisions at different DBMSs are identical
- Each global transaction can have at most one global subtransaction submitted to each local transaction manager

Additional Problems

- Indirect conflicts



- Inability to control execution order
  - Serialization order ≠ execution order
  - This is because the interface to the component DBMSs is not operational, but transactional
- May not have a visible ready-to-commit state
  - How to implement atomic commitment

Alternatives for Concurrency Control

- Ignore indirect conflicts
- Relax the local autonomy requirement
  - Each component DBMS can provide its execution order to the GTM.
- Prevent indirect conflicts by controlling execution order of global transactions
  - Either by serial execution or by “altruistic locking”
- Be a pessimist and assume conflicts will occur between global transactions if they cannot be ruled out

Alternatives for Concurrency Control

- Relax the correctness requirement
  - quasi-serializability or multidatabase serializability
- Use a semantics-based correctness criterion
  - accept non-serializable executions when they are acceptable according to application semantics

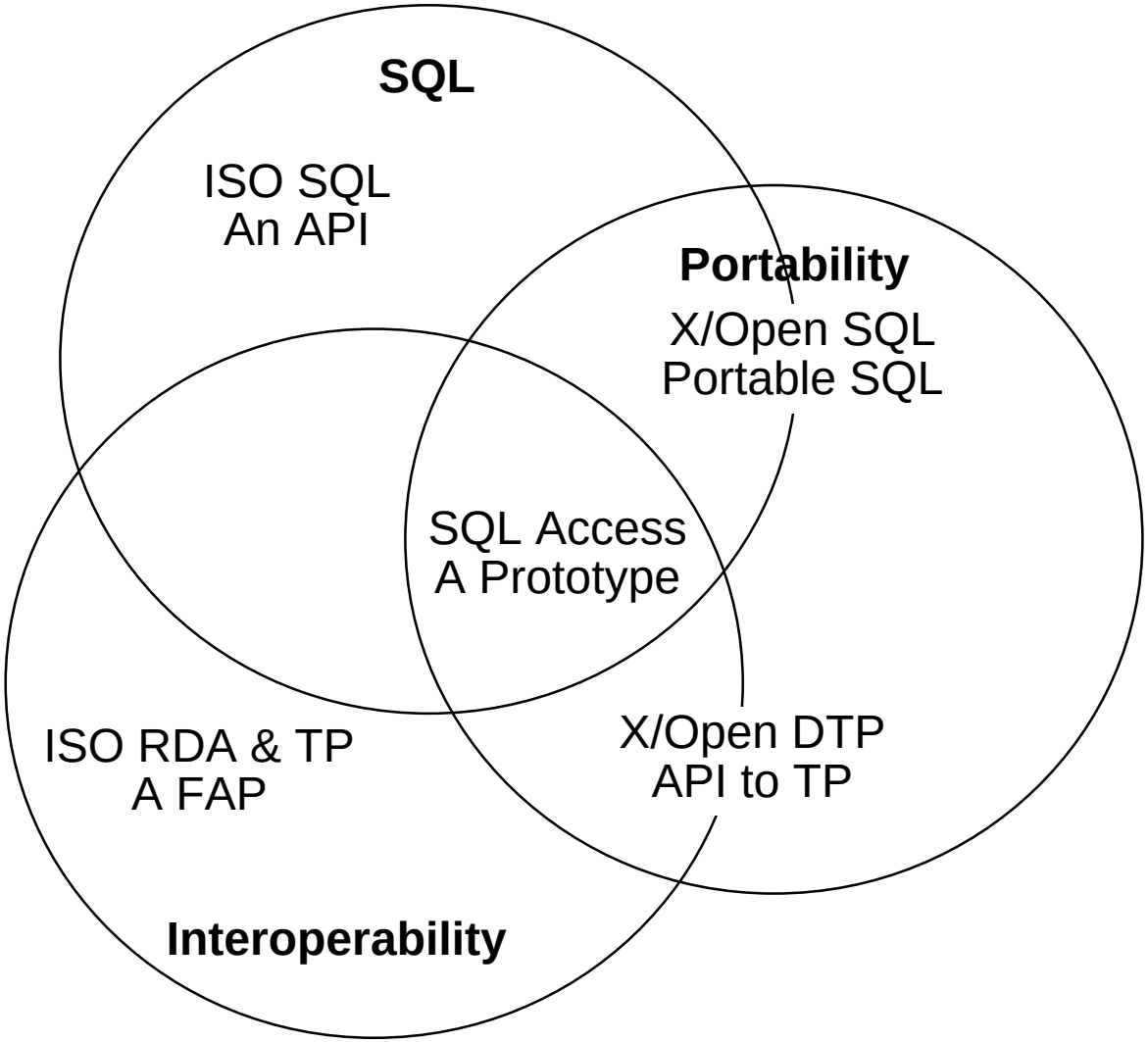
Alternatives for Commit

- Emulate ready-to-commit state by changing the transactions before they are submitted to the component DBMSs
- Compensation
  - will it always be possible?

DBMS Standards Activities

- ANSI and ISO SQL standardization activities
- X/Open SQL activities
- ISO and SQL AccessGroup RDA activities

Database Standards Players



[Gray, 1991]

SQL Related Standards

Year	Standard	Common Name	Name	Primary Orientation
1986	ANSI X3.135-1986	SQL-86	Database Language SQL	Least common denominator of SQL products
1987	ISO 9075-1987	SQL-86 or SQL-87	Database Language SQL	Identical to ANSI X3.135-1986
1989	ANSI X3.135-1989	SQL-89	Database Language SQL with integrity enhancement	Added basic referential integrity
1989	ISO/IEC 9075:1989	SQL-89	Database Language SQL	Identical to ANSi X3.135-1989
1992	ANSI X3.135-1992	SQL-92 or SQL2	Database Language SQL	Major enhancements based on implementations, requirements, and public review comments
1992	ISO/IEC 9075:1992	SQL-92 or SQL2	Database Language SQL	Identical to ANSI X3.135-1992

[Melton, 1992]

SQL-86/89

- Level 1
  - minimal language to allow all existing implementations to conform
- Level 2
  - extensions
- Addendum 1 (SQL-89)
  - added referential integrity

SQL-2

- Entry Level
  - almost same as SQL-89 with some small improvements (DDL)
- Intermediate Level
  - half of the remaining language
- Full language

SQL-2 Extensions

- Richer type system
  - BIT, DATE, TIME, TIMESTAMP, INTERVAL
- Internalization
  - Collating sequences for comparing characters in different character sets
- String operations
  - Concatenation, extraction of substrings, conversions
- Domains
  - User-defined domains using built-in data types or other domains
- Isolation levels
  - For transactions: levels 0 - 3. Level 3 provides serializability
- Sessions
  - User-defined sessions (connect/disconnect)

SQL-3

- Still being worked on
- Combine relational and object-oriented DBMSs
- Companies involved: DEC, HP, Oracle, IBM
- Date: around 1995 (?)
- Some new features:
  - object-oriented extensions
  - ADT
  - object identity
  - encapsulation
  - inheritance
  - polymorphism
  - subtyping
  - parameterized types
  - dynamic binding of operators to objects

RDA

- File Access Protocol
- Generic Model
  - ISO/IEC 9589-1-DIS
  - Defines a common database access protocol and transfer syntax
- Specializations
  - data models
  - languages
    - SQL specialization (ISO/IEC 9589-2-DIS)

RDA Service Definitions

R-Initialize: establish a session between client & server

R-Terminate

R-Open: Open a data resource on server for the client

R-Close

R-BeginTransaction

R-Commit

R-Rollback

R-Status: Return status of pending operations

R-Cancel: Cancel a pending operation

R-ExecuteDBL: Execute an associated DBL statement

R-DefineDBL: Store a DBL statement for later execution

R-InvokeDBL: Execute a DBL statement stored previously

R-DropDBL: Remove a previously stored DBL statement

RDA Example

Client Application	Client Request	Server Response
CONNECT TO server **AS** tamer **USER** ozsu	A-Assoc-Request	A-Assoc-Response
	R-Initialize-Req	R-Initialize-Rsp
	R-Open-Req	R-Open-Rsp
SELECT ENAME **INTO** :temp **FROM** E, S		
WHERE E.TITLE = S.TITLE		
AND S.SAL>30000	R-BeginTransaction-Req	
	R-ExecuteDBL-Req	R-ExecuteDBL-Rsp
COMMIT WORK	R-Commit-Req	R-Commit-Rsp
DISCONNECT tamer	R-Close-Req	R-Close-Rsp
	R-Terminate-Req	R-Terminate-Rsp

[Arnold et al., 1991]

SQL Access Group

- 38 companies
  - Apple, Ashton-Tate, Ask-Ingres, Boeing Computer Services, Corland, British Telecom, Bull, Cincom, DB Access, DEC, Du Pont, Fujitsu, Fulcrum, Gupta, HP, Infocentre, Informix, Locus, MCC, Metaphore, Micro Decisionware, Microsoft, Mimer, NCR, Novel, Oracle, Progress, Retix, Software AG, Sterling Software, Sun Microsystems, Sybase, Tandem, Teradata, Uniface, Unify, Unisys, X/Open Company Ltd.
- NIST & OSF observers
- Tool vendors : want *portability*
  - negotiated extension sets
- Server vendors: want *interoperability*
  - minimal gateway implementation cost
- Users: want both

SQL Access Group Prototype Demonstration