# CGS 2545: Database Concepts
# Summer 2007

## Chapter 9 – The Client/Server Database Environment


Instructor : Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 823-2790
http://www.cs.ucf.edu/courses/cgs2545/sum2007

School of Electrical Engineering and Computer Science
University of Central Florida


# Objectives

- Definition of terms.
- List advantages of client/server architecture.
- Explain three application components: presentation, processing, and storage.
- Suggest partitioning possibilities.
- Distinguish between file server, database server, 2-tier, 3-tier, and n-tier approaches.
- Describe and discuss middleware.
- Explain query-by-example (QBE).
- Explain database linking via ODBC and JDBC.
- Explain VBA and Microsoft Access.



# Client/Server Systems

- Client/server systems operate in network environments, splitting the processing of an application between a front-end client and a back-end processor.
- Generally speaking, the client requires some sort of service or resource which the server provides to the client.
- Client – Workstation (usually a PC) that requests and uses a service.
- Server – Computer (PC/mini/mainframe) that provides a service.
- Typically, the client and server are on different computers, although this is not a requirement.
- In a database system, the server is a database server that processes the DBMS.



# Client/Server Architectures

- Client/server environments typically use a LAN (local area network) to support a network of PCs, each with its own storage, that are also able to share common devices, (such as hard drives, printers, etc.) and software (such as a DBMS) attached to the LAN.
- At least one of the PCs on the LAN is designated as a file server, on which the database is stored.
- The LAN modules of a DBMS add concurrent access control, possibly extra security features, and query- or translation-queuing management to support concurrent access from multiple users of a shared database.



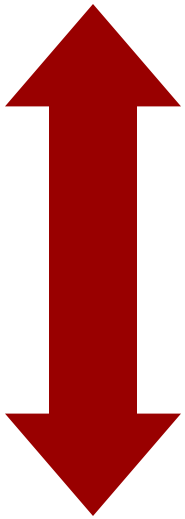
# Application Logic in C/S Systems

| | |
|---|---|
| Presentation Logic<br>– Input – keyboard/mouse<br>– Output – monitor/printer | **GUI Interface** |
| Processing Logic<br>– I/O processing<br>– Business rules<br>– Data management | **Procedures, functions, programs** |
| Storage Logic<br>– Data storage/retrieval | **DBMS activities** |

# Client/Server Architectures

- File Server Architecture
- Database Server Architecture
- Three-tier Architecture

**Client does extensive processing**



**Client does little processing**

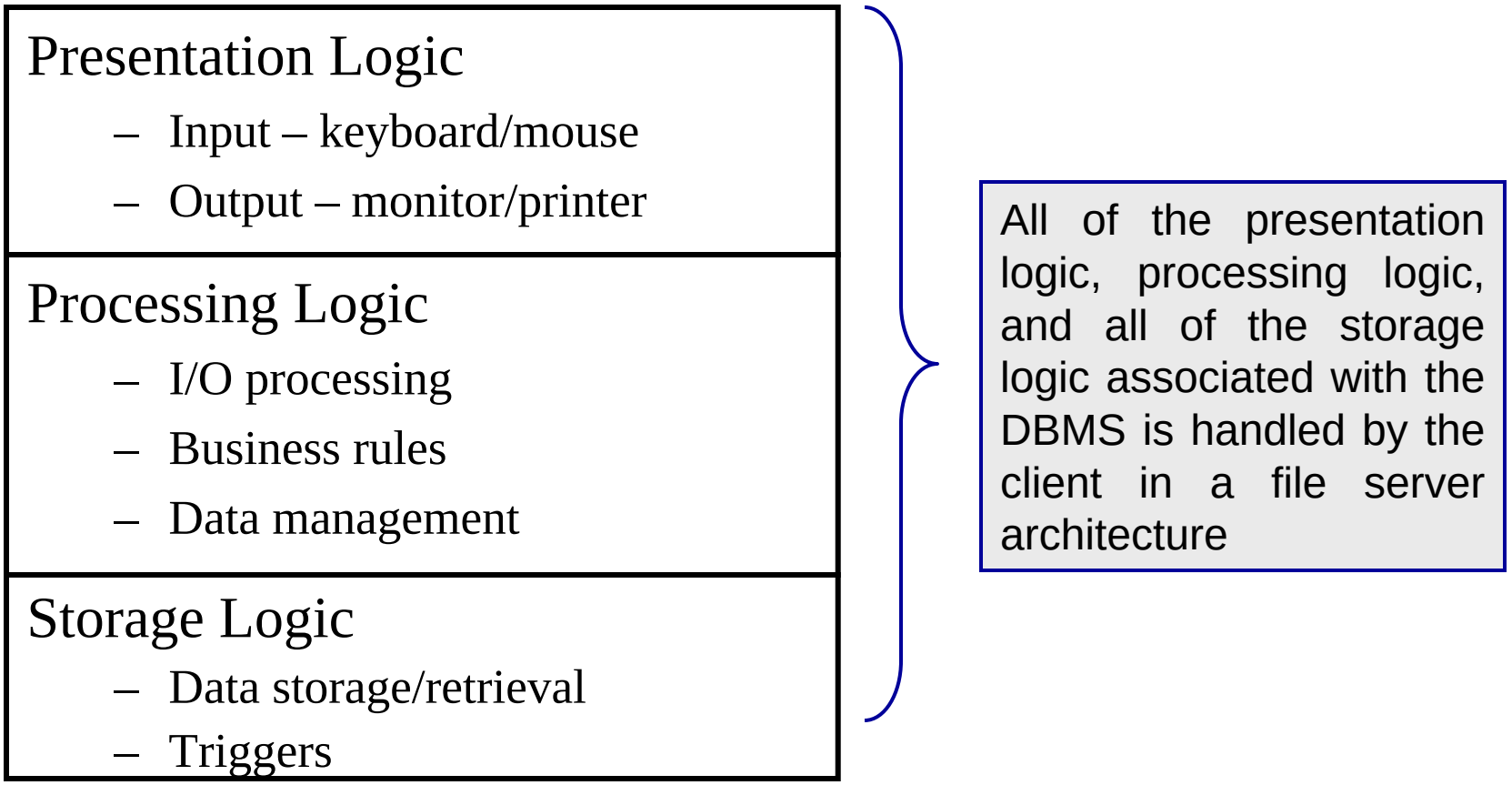


# File Server Architecture: FAT CLIENT

- All processing is done at the PC that requested the data.
- Entire files are transferred from the server to the client for processing.
- Problems:
  - Huge amount of data transfer on the network.
  - Each client must contain full DBMS.
    - Heavy resource demand on clients.
    - Client DBMSs must recognize shared locks, integrity checks, etc.

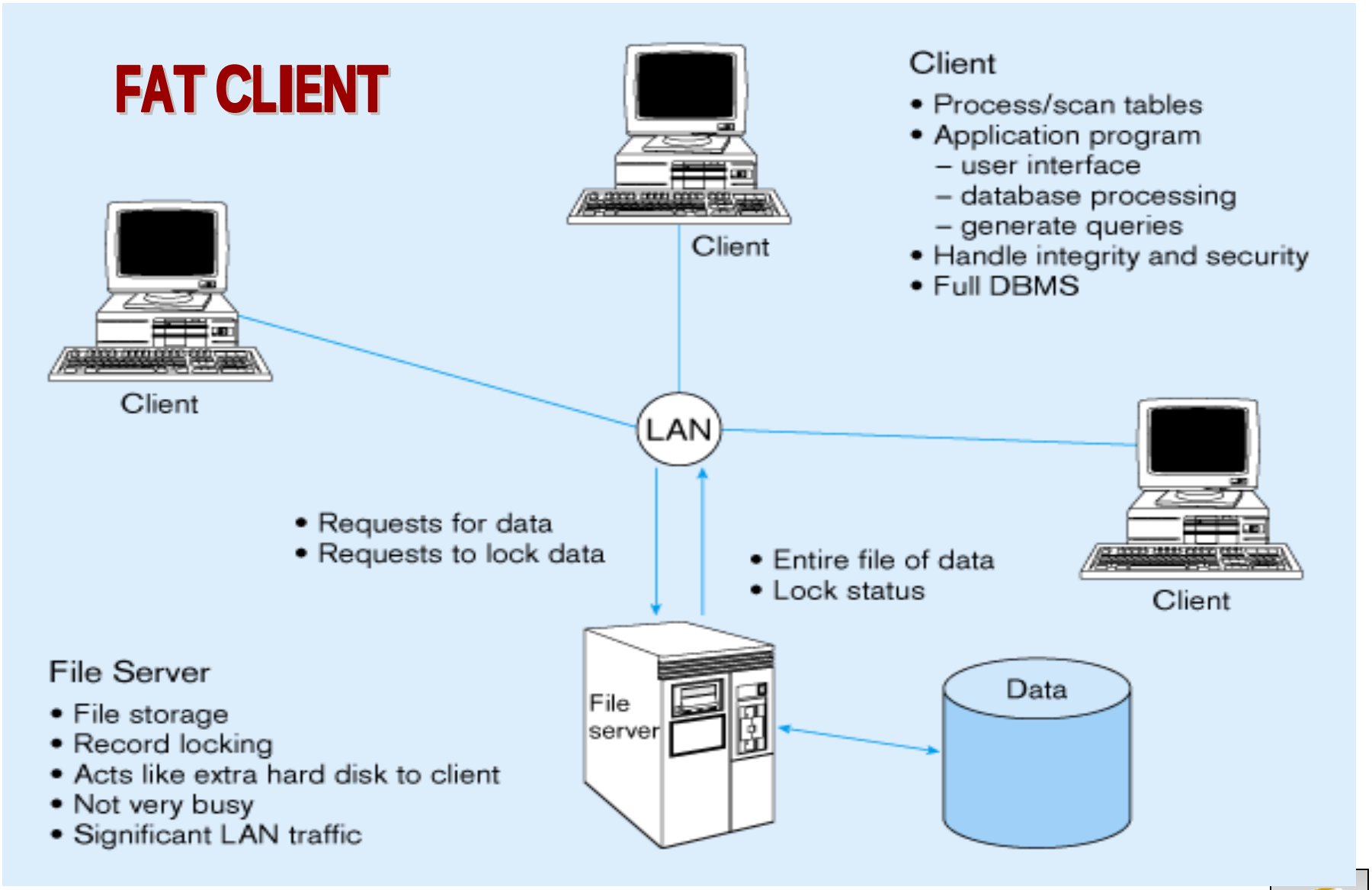


# Application Logic In File Server Systems






# Figure 9-2: File Server Architecture






# Limitations of File Server Architectures

- There are three primary limitations of the file server architecture.
  1. Considerable amounts of data must be moved around the network in response to client requests. Using our supplier/parts/jobs/shipments database as an example, a query such as list the part numbers of all red parts would require sending the entire Parts table to the client submitting the query. If we assume that only 10% of all the parts are red, then 90% of the data transferred to the client is useless for their processing.
  2. Each client must maintain a full copy of the DBMS, thereby limiting local resources.
  3. The DBMS copy in each client must manage the shared database integrity.

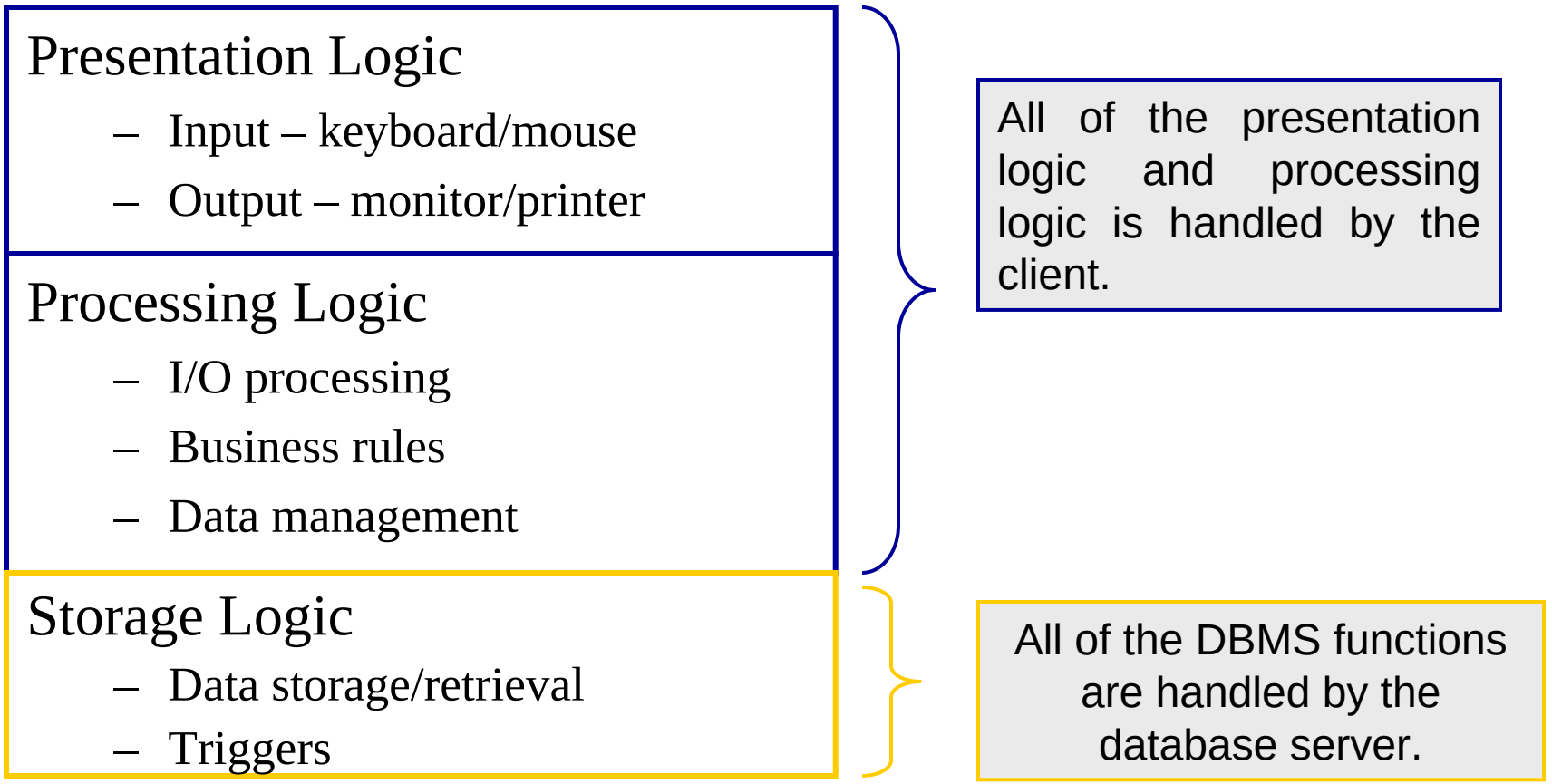


# Two-Tier Database Server Architectures

- Two-tiered approaches to client/server architectures are an extension of the file server architecture in which the database server (rather than a file server) is responsible for all database storage, access, and processing.
- In the two-tiered approach the client is responsible only for:
  - I/O processing logic.
  - Some business rules logic (those not incorporated into the DBMS).
- There is only one copy of the DBMS located on the DBMS server rather than one copy on each PC as was the case in the file server architecture.



# Application Logic In Database Server Systems

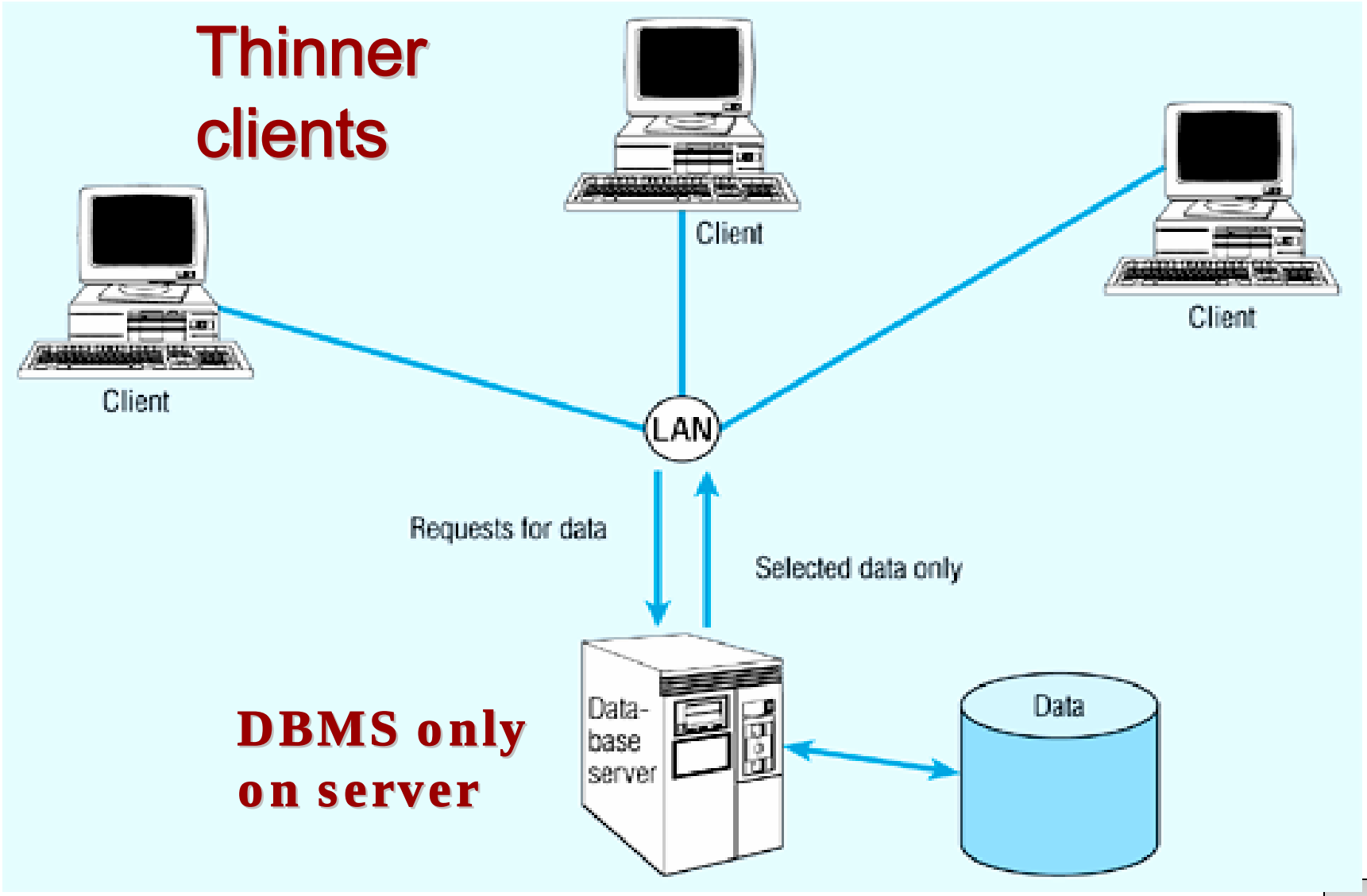





# Advantages of Two-Tier Approach

- Clients do not have to be as powerful. They do not contain any DBMS logic.
- Greatly reduces data traffic on the network. Only "results" are sent to the clients not data to be processed.
- Improves data integrity since it is all processed centrally.
- **Stored procedures**, which are modules of code that implement application logic and are included on the database server, allow the database server to handle more critical business application by implementing some business rules on the database server.

Figure 9-3: Two-tier database server architecture






# Three-Tier Architectures

- In general, a three-tier architecture adds a mid-level server which handles processing logic.
  - In reality, the exact roles of the two servers maybe defined somewhat differently, but this is basically the idea.
  - If application logic resides on this mid-level server, it will be referred to as an application server.
  - Alternatively, the mid-level server may hold a local database, while the another server holds the enterprise database.
  - Any of these, as well as a host of other configurations is likely to be called a three-tier architecture even though the functions of the various servers may differ dramatically.



# Application Logic In Three-Tier Architectures

**Presentation Logic**
- Input – keyboard/mouse
- Output – monitor/printer

The client handles only the presentation logic.

**Processing Logic**
- I/O processing
- Business rules
- Data management

The middle-tier server handles only the processing logic.

**Storage Logic**
- Data storage/retrieval
- Triggers

All of the DBMS functions are handled by the database server.

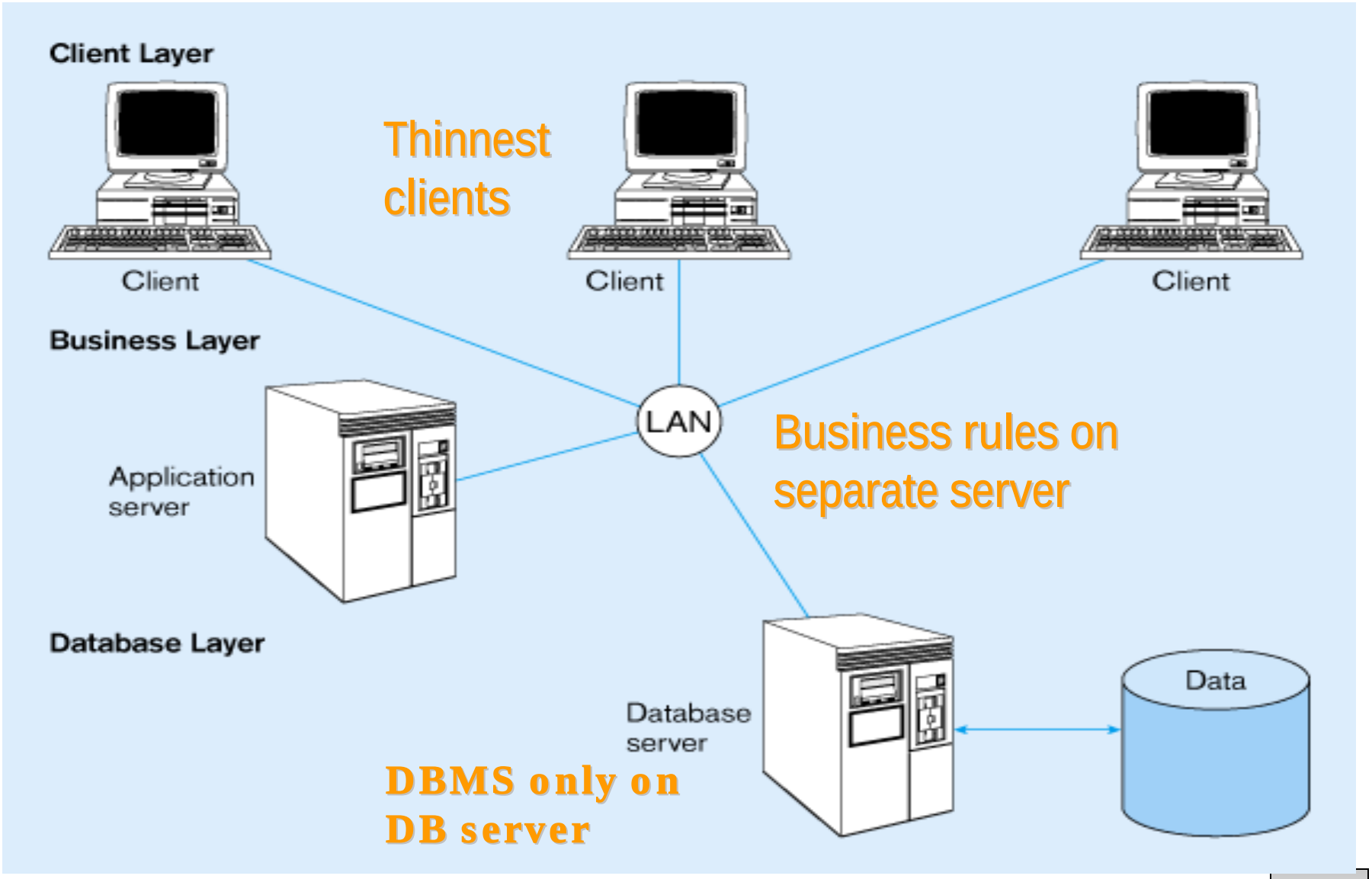
# Three-Tier Architectures

| | | |
|---|---|---|
| Client | **GUI interface (I/O processing)** | ***Browser*** |
| Application server | **Business rules** | ***Web Server*** |
| Database server | **Data storage** | ***DBMS*** |

## *Thin Client*

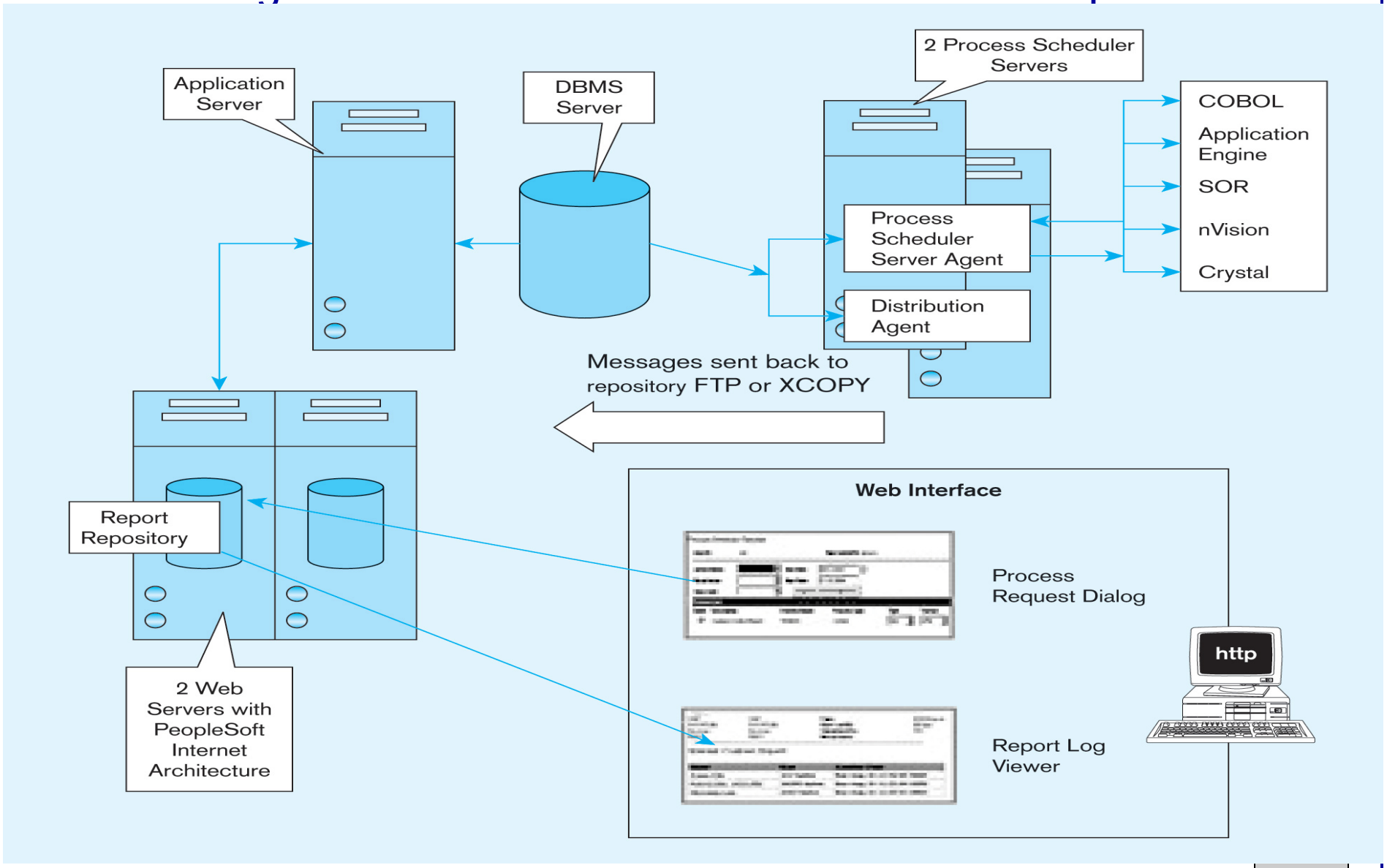
- PC just for user interface and a little application processing. Limited or no data storage (sometimes no hard drive, e.g., a dumb terminal.)

# Figure 9-4a: Three-tier architecture






# Figure 9-4b: Three-tier architecture: Example



# Advantages of Three-Tier Architectures

- Scalability.
- Technological flexibility.
- Long-term cost reduction.
- Better match of systems to business needs.
- Improved customer service.
- Competitive advantage.
- Reduced risk.



# Challenges of Three-tier Architectures

- High short-term costs.
- Tools and training.
- Experience.
- Incompatible standards.
- Lack of compatible end-user tools.



# Application Partitioning

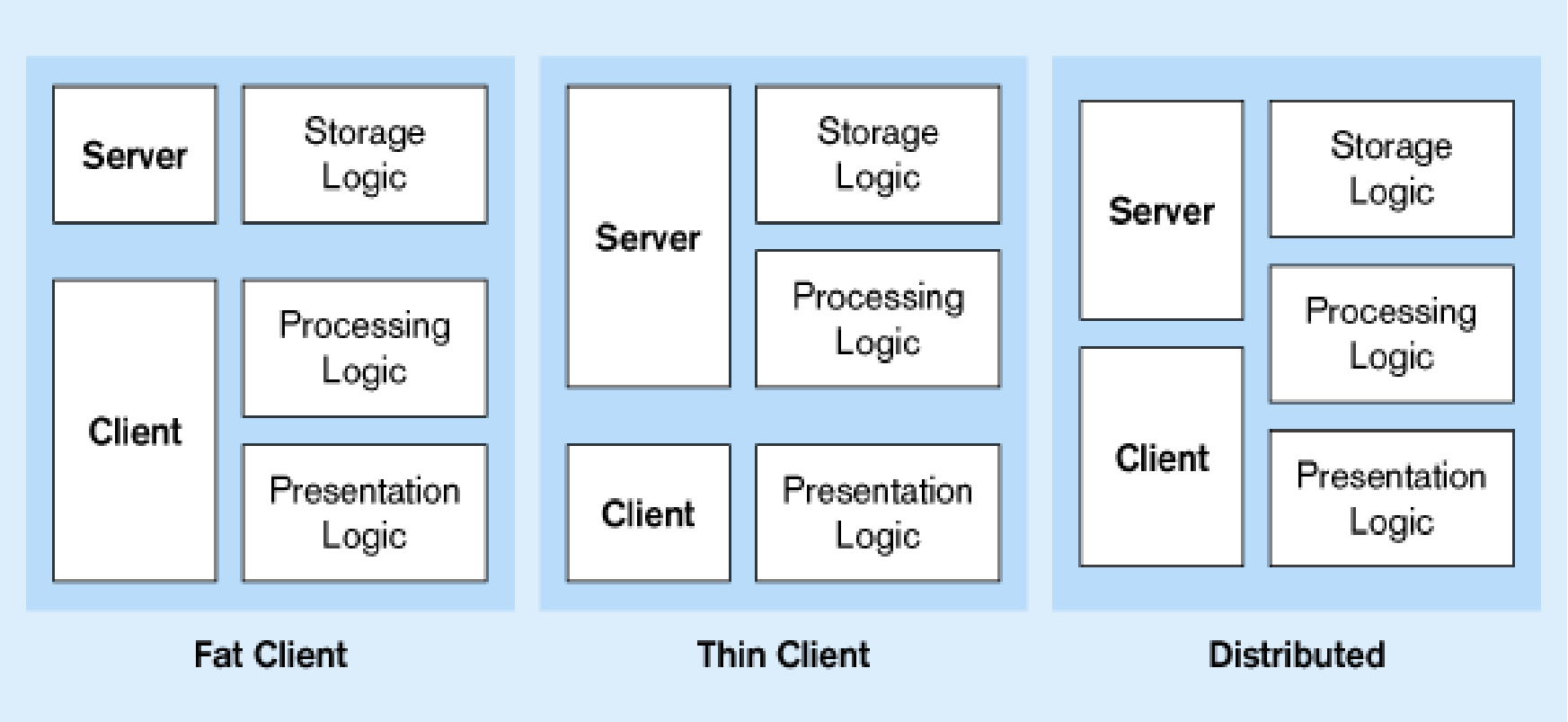
- Placing portions of the application code in different locations (client vs. server) AFTER it is written.
- Advantages
  - Improved performance
  - Improved interoperability
  - Balanced workloads



# Processing Logic Distributions

Two-tier Distributions

Processing logic could be at client, server, or both.

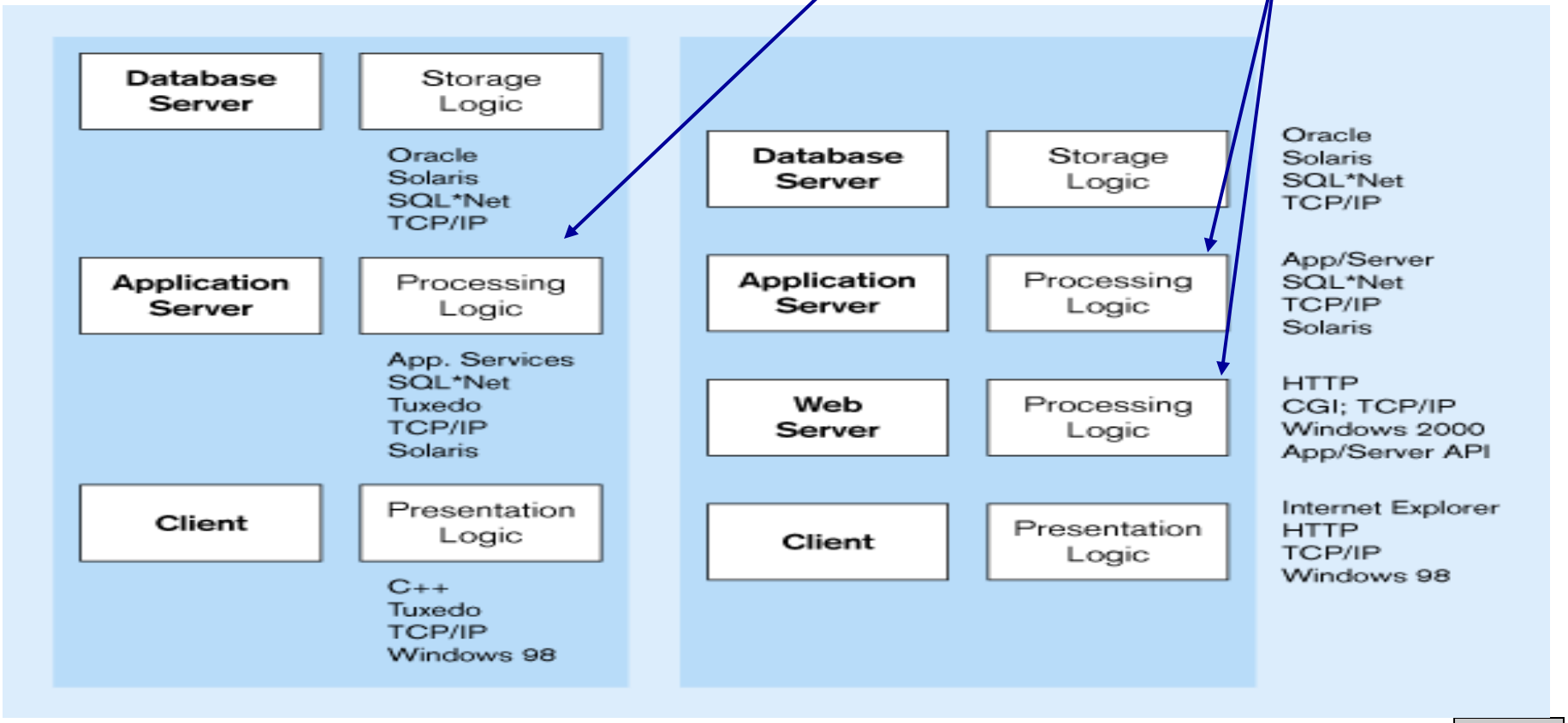





# Processing Logic Distributions

## *N*-tier Distributions

Processing logic will be at application server or Web server.






# Middleware

- Software which allows an application to ***interoperate*** with other software.
- No need for programmer/user to understand internal processing.
- Accomplished via ***Application Program Interface*** (API).
- Middleware is the term commonly used to describe any software component between the client and the database in n-tier architectures.

  **The *"glue"* that holds client/server applications together.**



# Middleware

- While middleware has existed in one form or another for decades, the advent of client/server technologies and web-oriented development, has stimulated new development of commercially available middleware.
- Universal middleware, one magical package of software that could integrate and connect every type of system would be ideal, however, no such package currently exists (and is unlikely to ever do so).
- Currently most organizations will use several different middleware packages, sometimes even within one application.



# Middleware

- The development of e-business requires that computers be able to communicate with each other over the Web.
- The middleware necessary to achieve this communication has come to be known as Web services, and we'll deal with this special area later.



# Synchronous vs. Asynchronous Communication

- Synchronous communication
  - The requesting system waits for a response to the request in real time.
  - An example might be an online banking system where the teller checks the balance of an account before cashing a check.
- Asynchronous communication
  - The requesting system sends a request but does not wait for a response in real time, rather the response is accepted whenever it is subsequently received.
  - An example would be email systems.



# Classification of Middleware

- Asynchronous Remote Procedure Calls (RPC)
  - The client makes calls to procedures running on remote computers (i.e., requests a service) but does not wait for a response. The client typically establishes a point-to-point connection with the server and performs other processing while it waits for the response.
  - If the connection is lost, the client must reestablish the connection and resend the request.
  - This type of middleware has high scalability but low recoverability, and has largely been replaced by synchronous RPC since 1998.

- Synchronous Remote Procedure Calls (RPC)
  - A distributed program using synchronous RPC may call services available on different servers simultaneously.
  - Examples include Microsoft's Transaction Server and IBM/s CICS. The Java equivalent of an RPC is a Remote Method Invocation (RMI).



# Classification of Middleware

- Message-Oriented Middleware (MOM)
  - Asynchronous calls between the client and server via message queues. The client collects the messages from the server and acts upon them at a later time.
  - Workflow applications such as insurance policy applications, which often involve several processing steps are among the types of applications that can benefit from MOM. The queue where requests are stored is often journalized in order to provide some recoverability.

- Publish/Subscribe
  - This is an asynchronous push technology that monitors activity and "pushes" information to subscriber clients when available. The clients perform other activities in between notifications from the server.
  - This type of middleware is most useful for monitoring situations where actions need to be taken when particular events occur.



# Classification of Middleware

- Object Request Broker (ORB)
  - This type of middleware makes it possible for applications to send objects and request services in an object-oriented system.
  - The ORB tracks the location of each object and routes the requests to each object.
  - Current ORBs are synchronous, but asynchronous ORBs are being developed.

- SQL-oriented Data Access
  - Connecting applications to database over networks is achieved using SQL-oriented data access middleware.
  - This middleware also has the capability to translate generic SQL into the SQL-specific to the database, e.g., generic SQL into Access SQL.
  - Database vendors and companies that have developed multidatabase access middleware dominate this segment of middleware.



# Database Middleware

- In client/server systems, database-oriented middleware provides some sort of application program interface (API) access to the database.
- APIs are sets of routines that an application program uses to direct the performance of procedures by the computer's operating system.
  - For example, in achieving access to a database, an API calls library routines that transparently route SQL commands from the front-end client application to the database server.
- ODBC – Open Database Connectivity
  - Most DB vendors support this.
  - Similar to API, but is specifically for Windows-based client/server applications.
  - Most useful for accessing relational data, and not well suited for accessing other types of data such as ISAM files.
  - Although difficult to program and implement, it has been widely accepted because it allows programmers to make connections to almost any vendor's database without learning proprietary code specific to that database.



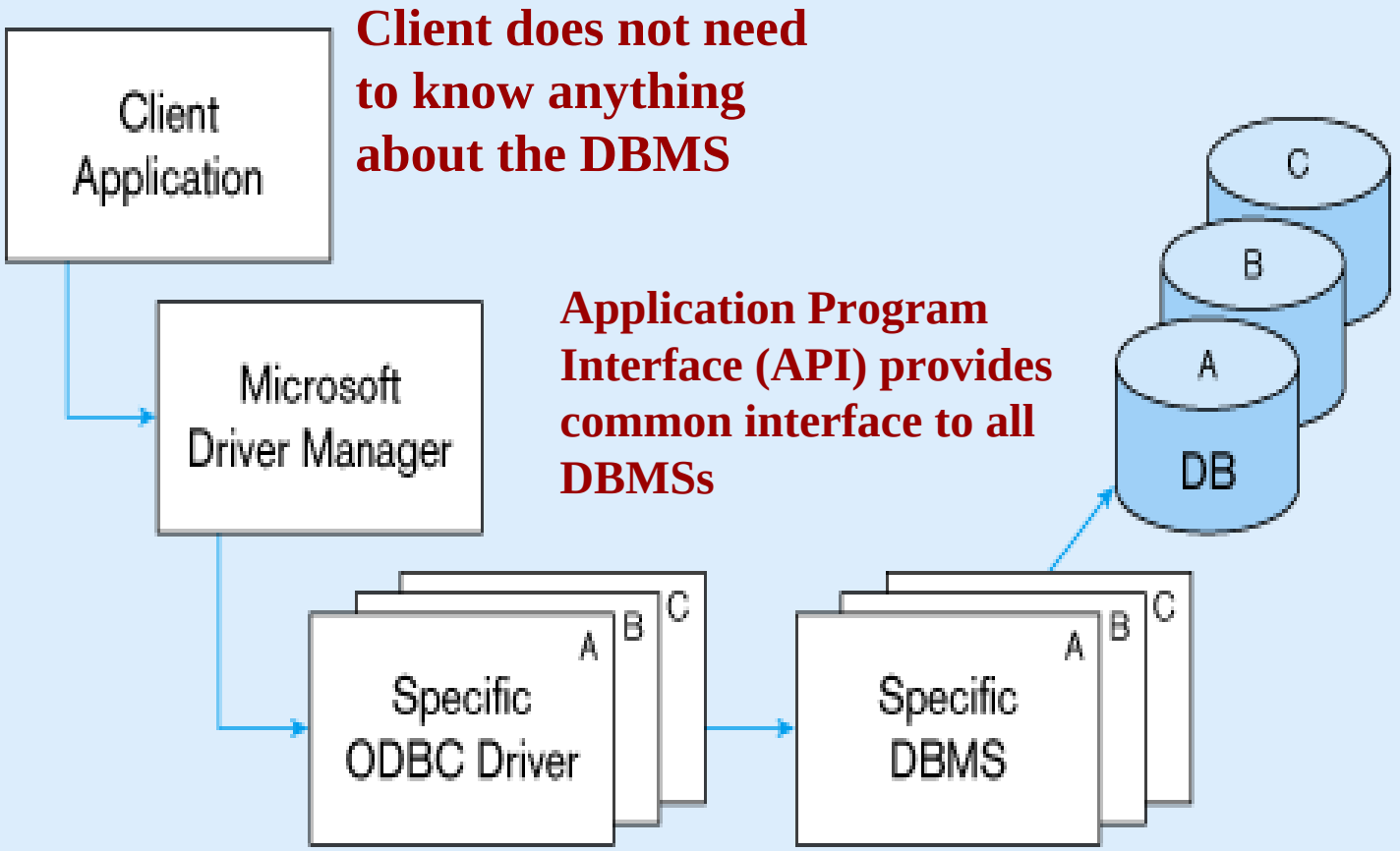
# Using ODBC to Link External Databases Stored on a Database Server

- Open Database Connectivity (ODBC)
  - API that provides a common language for application programs to access and process SQL databases independent of the particular RDBMS that is accessed
- Required parameters:
  - ODBC driver
  - Back-end server name
  - Database name
  - User id and password
- Additional information:
  - Data source name (DSN)
  - Windows client computer name
  - Client application program's executable name

Java Database Connectivity (JDBC) is similar to ODBC – built specifically for Java applications



# ODBC Architecture






# Database Middleware

- OLE-DB
  - Microsoft enhancement of ODBC.
  - Plans are to make this a universal access standard.
  - Plans are also in the works to develop an OLE-DB for data mining applications and one for OLAP.
- JDBC – Java Database Connectivity
  - Special Java classes that allow Java applications/applets to connect to databases.
- The Object Management Group (OMG), established in 1989, is an industry coalition that has produced the Common Object Request Broker Architecture (CORBA), which sets the specification of object-oriented universal middleware.
  - Microsoft has developed a competing model, Distributed Component Object Model (DCOM), but CORBA is a more robust specification because it has been developed to handle many different platforms. Interoperability between the two standards is slowly emerging.



# Issues In Client/Server Systems

- To succeed, a client/server project should address a specific business problem with well-defined technology and cost parameters.
- To develop a successful client/server application, several key areas must be addressed, including:
  - Accurate business problem analysis.
  - Detailed architecture analysis.
  - Avoidance of tool-driven architectures.
  - Achieving appropriate scalability.
  - Appropriate placement of services.
  - Network analysis.
  - Determination of hidden costs.
  - Establishing client/server security.



# Accurate Business Problem Analysis

- Don't fit the technology to the problem.
- Instead first define the scope of the problem accurately, determine the requirements, and then use that information to select the appropriate technology.
- Too often developer's will tend to pick a technology because it is a "hot" item and then fit the application to the technology rather than the other way around.



# Detailed Architecture Analysis

- It is important to specify the details of the client/server architecture.
- Building a client/server solution involves connecting many components, which may not work together easily.
- Analysts should specify the client workstations, server(s), network, DBMS, as well as the network infrastructure, middleware layer, and the application development tools to be used.
- At each step, care should be taken to ensure that the selected tools will connect with the middleware, database, network, and so forth.



# Avoidance of Tool-Driven Architectures

- Determine the project requirements before choosing the software tools, and not the reverse.
- Choosing the tool first and then applying it to the problem risks having a poor fit between the problem and the tool.
- Don't select the tool because it's the trendy thing to do, do it because it fits the problem the best.
  - Example - Modula



# Achieving Appropriate Scalability

- A multi-tier solution allows client/server systems to scale to any number of users and handle divers processing loads.
- However, multi-tier solutions are significantly more expensive and difficult to build and maintain.
- Avoid multi-tier solutions where they are not really needed. Typically, this would imply more than 100 concurrent users, high-volume transaction processing systems, or real-time processing.
- Smaller, less intense environments can frequently run more efficiently on the more traditional two-tier systems, especially if triggers (and procedures) are used to manage processing.



# Appropriate Placement of Services

- Again, a careful analysis of the business problem being addressed is important when making decisions about the placement of processing services.
- The move toward thin clients and fat servers is not always the most appropriate scenario.
  - Moving the application logic to a server, thus creating a fat server, can affect capacity, as clients all attempt to use the application now located on the server.
- Sometimes it is possible to achieve better scaling by moving the application processing to the client.
- Fat servers tend to reduce network load since processing takes place close to the data, and fat servers do lessen the need for powerful clients.
- Thus, understanding the business problem intimately will help the architect to distribute the logic appropriately.



# Network Analysis

- The most common bottleneck in distributed systems is still the network.
- Architects ignore at their own peril the bandwidth capabilities of the network that the system must utilize.
- If the network is insufficient to handle the amount of information that must pass between the client and server, response time will suffer badly, and the system is likely to fail.



# Determination of Hidden Costs

- Client/server implementation problems go beyond the analysis, development, and architecture problems we've already covered.
- For example, systems that are intended to use existing hardware, networks, operating systems, and DBMSs are often plagued by the complexities of integrating these heterogeneous components to build the client/server system.
- Training is a significant and recurring expense that is often overlooked or under estimated.
- The complexities of working in a multi-vendor environment can be very costly.



# Client/Server Security

- The distributed nature of client/server database computing implies that security issues are more complex than those encountered in a centralized environment.
- Both server security and network security must be established.
- The Web-enabled database environment raises additional security issues.
- Recent invasions of organizational databases and the resulting loss of sensitive customer information has made people much more aware of the potential threats that exist.



# Client/Server Benefits

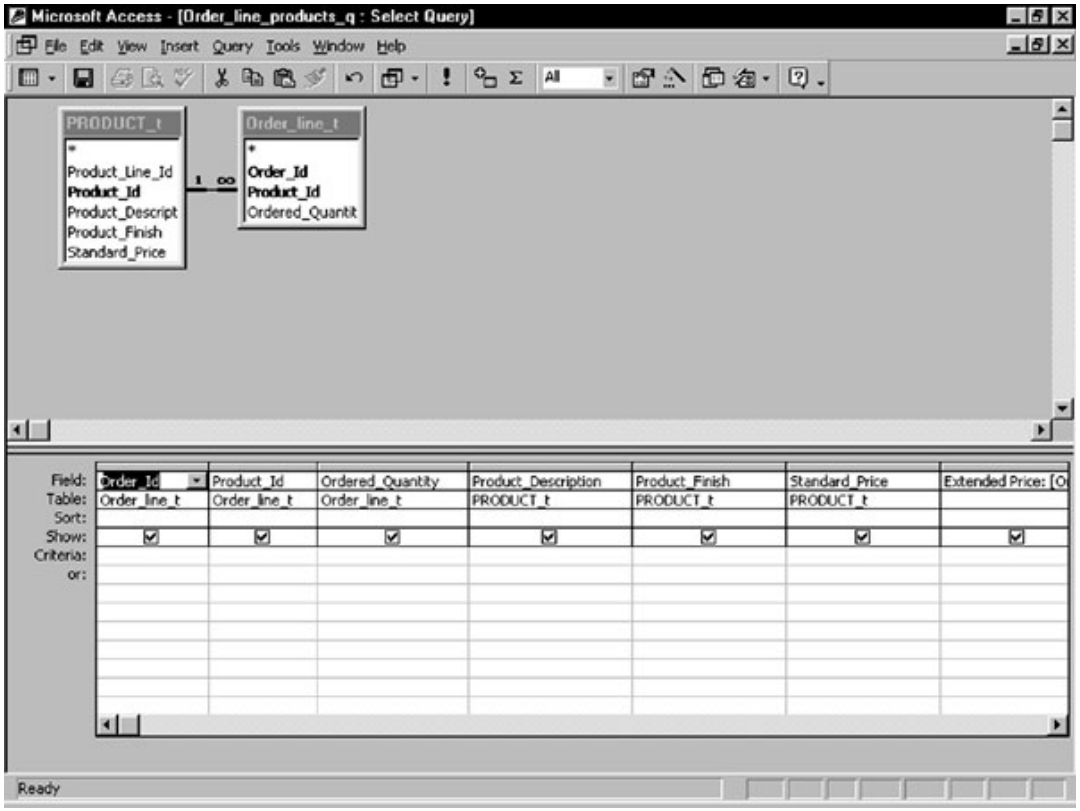
- If the issues we've just covered are addressed appropriately, there are many benefits to be obtained from moving to client/server architectures including:
  - Functionality can be delivered in stages to the clients. Thus, it begins to arrive more quickly as the first pieces of the project are deployed.
  - The GUIs common in client/server environments encourage users to use the application's functionality.
  - The flexibility and scalability of client/server solutions facilitate business process reengineering.
  - More processing can be performed close to the source of the data being processed, thereby improving response times and reducing network traffic.
  - Client/server architectures allow the development of Web-enabled applications, facilitating the ability of organizations to communicate effectively internally and to conduct external business over the Internet.



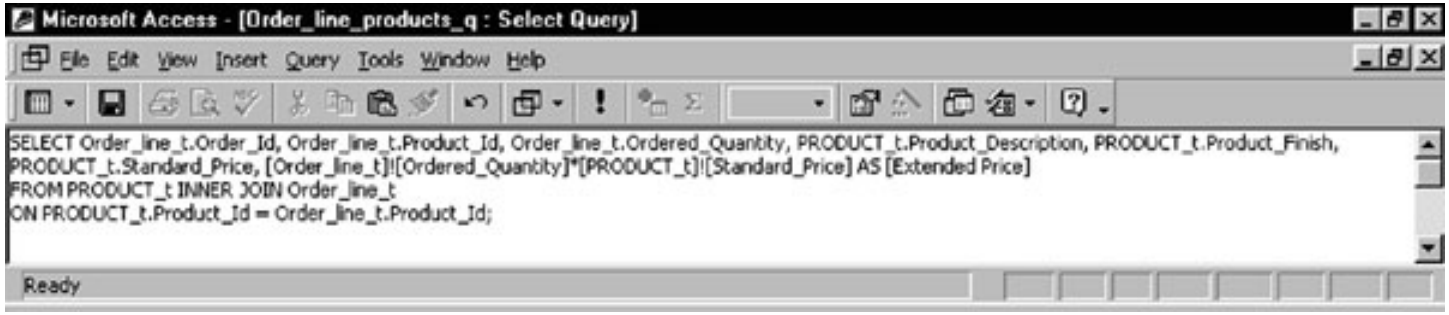
# Query-by-Example (QBE)

- Direct-manipulation database language
- Graphical approach
- Available in MS Access
- MS Access translates QBE to SQL and vice versa
- Useful for end-user database programming
- Good for ad hoc processing and prototyping





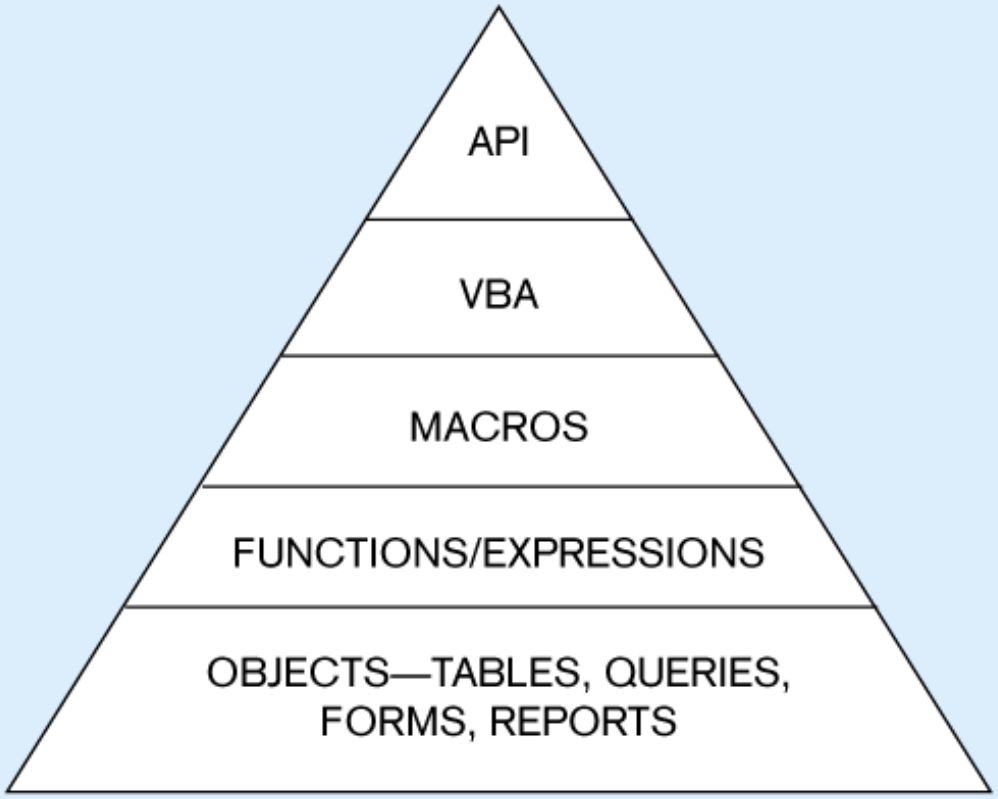

QBE view of a multiple-table join query




Equivalent query in SQL



# Access usability hierarchy




**API to call functions in DLLs external to MS Access**

**Visual Basic for Applications…language for customizing the application**

**Stored modules of pre-existing VBA code**

**Simple processes**

**Foundation of MS Access**



# Visual Basic for Applications

- VBA is the programming language that accompanies Access 2003.
- VBA provides these features:
  - Ability to perform complex functionality
  - Error handling
  - Faster execution than macros
  - Easier maintenance
  - OLE automation
  - Programmatic control
  - Ease of reading for programmers
- Event-driven – nonprocedural programming that detects events and generates appropriate responses