# Enterprise Data Analysis and Design

Lecture 2:
Enterprise Architectures

Johannes Gehrke
johannes@cs.cornell.edu
http://www.cs.cornell.edu/johannes

(Some of the slides are courtesy of Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju)

## Course Outline

- 1/26 Database Management Systems
- 1/28 Enterprise Information Architectures
- 2/2, 2/4, and 2/9: Data Modeling
- 2/11, 2/16, 2/18, 2/23, and 2/25: Data Mining
- 3/2 OLAP
- 3/4 Web Services
- 3/9 Future Trends



## Why Store Data in a DBMS?

- Benefits
  - Transactions (concurrent data access, recovery from system crashes)
  - High-level abstractions for data access, manipulation, and administration
  - Data integrity and security
  - Performance and scalability

## A Digress – What Is a Transaction?

The execution of a program that performs a function by accessing a database.

Examples:

- Reserve an airline seat. Buy an airline ticket.
- Withdraw money from an ATM.
- Verify a credit card sale.
- Order an item from an Internet retailer.
- Download a video clip and pay for it.
- Play a bid at an on-line auction.



## Transactions

- A transaction is an atomic sequence of actions
- Each transaction must leave the system in a consistent state (if system is consistent when the transaction starts).
- The ACID Properties:
  - Atomicity
  - Consistency
  - Isolation
  - Durability



## Concurrency Control for Isolation

(Start: A=$100; B=$100)

Consider two transactions:

- T1: START, A=A+100, B=B-100, COMMIT
- T2: START, A=1.06*A, B=1.06*B, COMMIT

The first transaction is transferring $100 from B's account to A's account. The second transaction is crediting both accounts with a 6% interest payment.

Database systems try to do as many operations concurrently as possible, to increase performance.

## Example (Contd.)

(Start: A=$100; B=$100)

- Consider a possible interleaving (schedule):

T1: A=A+$100, B=B-$100 COMMIT
T2: A=1.06*A, B=1.06*B COMMIT
End result: A=$106; B=$0

- Another possible interleaving:

T1: A=A+100, B=B-100 COMMIT
T2: A=1.06*A, B=1.06*B COMMIT
End result: A=$112; B=$6

The second interleaving is incorrect! Concurrency control of a database system makes sure that the second schedule does not happen.



## Ensuring Atomicity

- DBMS ensures atomicity (all-or-nothing property) even if the system crashes in the middle of a transaction.
- Idea: Keep a log (history) of all actions carried out by the DBMS while executing :
  - Before a change is made to the database, the corresponding log entry is forced to a safe location.
  - After a crash, the effects of partially executed transactions are undone using the log.



## Recovery

- A DBMS logs all elementary events on stable storage. This data is called the log.
- The log contains everything that changes data: Inserts, updates, and deletes.
- Reasons for logging:
  - Need to UNDO transactions
  - Recover from a systems crash

## Recovery: Example

(Simplified process)

- Insert customer data into the database
- Check order availability
- Insert order data into the database
- Write recovery data (the log) to stable storage
- Return order confirmation number to the customer



## Why Store Data in a DBMS?

- Benefits
  - Transactions (concurrent data access, recovery from system crashes)
  - High-level abstractions for data access, manipulation, and administration
  - Data integrity and security
  - Performance and scalability



## Data Model

- A data model is a collection of concepts for describing data.
- Examples:
  - ER model (used for conceptual modeling)
  - Relational model, object-oriented model, object-relational model (actually implemented in current DBMS)

## The Relational Data Model

A relational database is a set of relations. Turing Award (Nobel Price in CS) for Codd in 1980 for his work on the relational model

- Example relation:
  Customers(cid: integer, name: string, byear: integer, state: string)

| cid | name | byear | state |
|---|---|---|---|
| 1 | Jones | 1960 | NY |
| 2 | Smith | 1974 | CA |
| 3 | Smith | 1950 | NY |



## The Relational Model: Terminology

- Relation instance and schema
- Field (column)
- Record or tuple (row)
- Cardinality

| cid | name | byear | state |
|---|---|---|---|
| 1 | Jones | 1960 | NY |
| 2 | Smith | 1974 | CA |
| 3 | Smith | 1950 | NY |



## Customer Relation (Contd.)

- In your enterprise, you are more likely to have a schema similar to the following:

  Customers(cid, identifier, nameType, salutation, firstName, middleNames, lastName, culturalGreetingStyle, gender, customerType, degrees, ethnicity, companyName, departmentName, jobTitle, primaryPhone, primaryFax, email, website, building, floor, mailstop, addressType, streetNumber, streetName, streetDirection, POBox, city, state, zipCode, region, country, assembledAddressBlock, currency, maritalStatus, bYear, profession)

## Product Relation

- Relation schema:
  Products(pid: integer, pname: string, price: float, category: string)
- Relation instance:

| pid | pname | price | category |
|---|---|---|---|
| 1 | Intel PIII-700 | 300.00 | hardware |
| 2 | MS Office Pro | 500.00 | software |
| 3 | IBM DB2 | 5000.00 | software |
| 4 | Thinkpad 600E | 5000.00 | hardware |



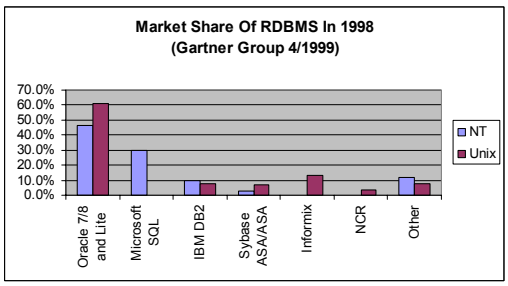
## Transaction Relation

- Relation schema:
  Transactions(
  tid: integer,
  tdate: date,
  cid: integer,
  pid: integer)
- Relation instance:

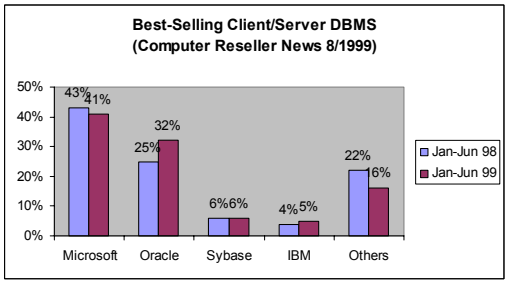
| tid | tdate | cid | pid |
|---|---|---|---|
| 1 | 1/1/2000 | 1 | 1 |
| 1 | 1/1/2000 | 1 | 2 |
| 2 | 1/1/2000 | 1 | 4 |
| 3 | 2/1/2000 | 2 | 3 |
| 3 | 2/1/2000 | 2 | 4 |



## The Relational DBMS Market

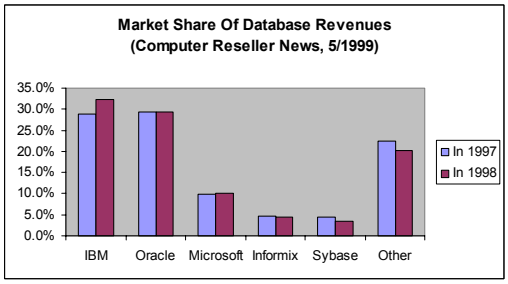


## The Relational DBMS Market (Contd.)






## The Relational DBMS Market (Contd.)






## The Object-Oriented Data Model

- Richer data model. Goal: Bridge impedance mismatch between programming languages and the database system.
- Example components of the data model: Relationships between objects directly as pointers.
- Result: Can store abstract data types directly in the DBMS
  - Pictures
  - Geographic coordinates
  - Movies
  - CAD objects

## Object-Oriented DBMS

- Advantages: Engineering applications (CAD and CAM and CASE computer aided software engineering), multimedia applications.
- Disadvantages:
  - Technology not as mature as relational DMBS
  - Not suitable for decision support, weak security
  - Vendors are much smaller companies and their financial stability is questionable.



## Object-Oriented DBMS (Contd.)

Vendors:

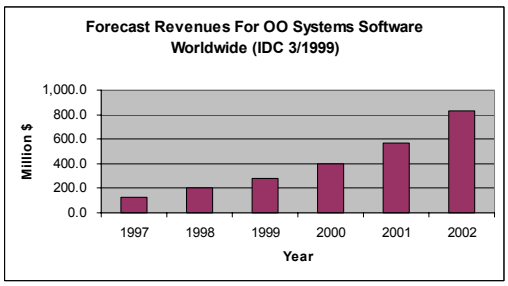
- Gemstone (www.gemstone.com)
- Objectivity (www.objy.com)
- ObjectStore (www.objectstore.net)
- POET (www.poet.com)
- Versant (www.versant.com, merged with POET)

Organizations:

- OMG: Object Management Group (www.omg.org)



## The OO DBMS Market



## Object-Relational DBMS

- Mixture between the object-oriented and the object-relational data model
  - Combines ease of querying with ability to store abstract data types
  - Conceptually, the relational model, but every field
- All major relational vendors are currently extending their relational DBMS to the object-relational model



## Query Languages

We need a high-level language to describe and manipulate the data

Requirements:

- Precise semantics
- Easy integration into applications written in C++/Java/Visual Basic/etc.
- Easy to learn
- DBMS needs to be able to efficiently evaluate queries written in the language



## Relational Query Languages

- The relational model supports simple, powerful querying of data.
  - Precise semantics for relational queries
  - Efficient execution of queries by the DBMS
  - Independent of physical storage

## SQL: Structured Query Language

- Developed by IBM (System R) in the 1970s
- ANSI standard since 1986:
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision, current standard)
  - SQL-99 (major extensions)
- More about SQL in the next lecture



## Example Query

- Example Schema:
  Customers(
  cid: integer,
  name: string,
  byear: integer,
  state: string)

| cid | name | byear | state |
|---|---|---|---|
| 1 | Jones | 1960 | NY |
| 2 | Smith | 1974 | CA |
| 3 | Smith | 1950 | NY |

- Query:

```
SELECT
    Customers.cid,
    Customers.name,
    Customers.byear,
    Customers.state
FROM Customers
WHERE Customers.cid = 3
```

| cid | name | byear | state |
|---|---|---|---|
| 3 | Smith | 1950 | NY |



## Example Query

```
SELECT
    Customers.cid,
    Customers.name,
    Customers.byear,
    Customers.state
FROM Customers
WHERE
    Customers.cid = 1
```

| cid | name | byear | state |
|---|---|---|---|
| 1 | Jones | 1960 | NY |
| 2 | Smith | 1974 | CA |
| 3 | Smith | 1950 | NY |

| cid | name | byear | state |
|---|---|---|---|
| 1 | Jones | 1960 | NY |

## Why Store Data in a DBMS?

- Benefits
  - Transactions (concurrent data access, recovery from system crashes)
  - High-level abstractions for data access, manipulation, and administration
  - Data integrity and security
  - Performance and scalability



## Integrity Constraints

- Integrity Constraints (ICs): Condition that must be true for any instance of the database.
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.
  - A legal instance of a relation is one that satisfies all specified ICs.
  - DBMS should only allow legal instances.
- Example: Domain constraints.



## Primary Key Constraints

- A set of fields is a superkey for a relation if no two distinct tuples can have same values in all key fields.
- A set of fields is a key if the set is a superkey, and none of its subsets is a superkey.
- Example:
  - {cid, name} is a superkey for Customers
  - {cid} is a key for Customers
- Where do primary key constraints come from?

## Primary Key Constraints (Contd.)

- Can there be more than one key for a relation?
- What is the maximum number of superkeys for a relation?
- What is the primary key of the Products relation? How about the Transactions relation?



## Enforcing Referential Integrity

- What should be done if a Transaction tuple with a non-existent Customer id is inserted? (Reject it!)
- What should be done if a Customer tuple is deleted?
  - Also delete all Transaction tuples that refer to it.
  - Disallow deletion of a Customer tuple that has associated Transactions.
  - Set cid in Transactions tuples to a default or special cid.
- SQL supports all three choices



## Where do ICs Come From?

- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations.
- We can check a database instance to see if an IC is violated, but we can NEVER infer that an IC is true by looking at an instance.
  - An IC is a statement about all possible instances!
  - From example, we know state cannot be a key, but the assertion that cid is a key is given to us.
- Key and foreign key ICs are very common; a DBMS supports more general ICs.

## Security

- Secrecy: Users should not be able to see things they are not supposed to.
  - E.g., A student can't see other students' grades.
- Integrity: Users should not be able to modify things they are not supposed to.
  - E.g., Only instructors can assign grades.
- Availability: Users should be able to see and modify things they are allowed to.



## Discretionary Access Control

- Based on the concept of access rights or privileges for objects (tables and views), and mechanisms for giving users privileges (and revoking privileges).
- Creator of data automatically gets all privileges on it.
  - DMBS keeps track of who subsequently gains and loses privileges, and ensures that only requests from users who have the necessary privileges (at the time the request is issued) are allowed.
- Users can grant and revoke privileges



## Role-Based Authorization

- In SQL-92, privileges are actually assigned to authorization ids, which can denote a single user or a group of users.
- In SQL:1999 (and in many current systems), privileges are assigned to roles.
  - Roles can then be granted to users and to other roles.
  - Reflects how real organizations work.
  - Illustrates how standards often catch up with "de facto" standards embodied in popular systems.

## Security Is Important

Financial estimate of database losses, by activity in 1998
(ENT/CSI/FBI Computer Crime Survey, 5/1999)

| *Activity* | *Loss in million $* |
|---|---|
| Theft of Proprietary Information | 28.51 |
| System Penetration by Outsiders | 13.39 |
| Financial Fraud | 11.24 |
| Unauthorized Insider Access | 50.57 |
| Laptop Theft | 0.16 |
| Total | 103.87 |



## Why Store Data in a DBMS?

- Benefits
  - Transactions (concurrent data access, recovery from system crashes)
  - High-level abstractions for data access, manipulation, and administration
  - Data integrity and security
  - Performance and scalability



## DBMS and Performance

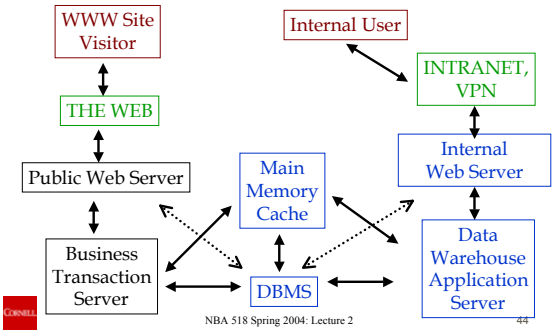
- Efficient implementation of all database operations
- Indexes: Auxiliary structures that allow fast access to the portion of data that a query is about
- Smart buffer management
- Query optimization: Finds the best way to execute a query
- Automatic high-performance concurrent query execution, query parallelization

## Summary Of DBMS Benefits

- Transactions
  - ACID properties, concurrency control, recovery
- High-level abstractions for data access
  - Data models
- Data integrity and security
  - Key constraints, foreign key constraints, access control
- Performance and scalability
  - Parallel DBMS, distributed DBMS, performance tuning

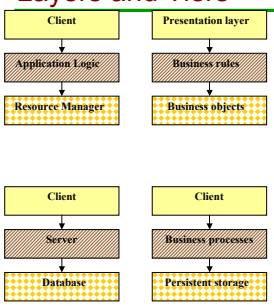


## The Big Picture (Revisited)






## Layers and Tiers

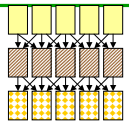



Client is any user or program that wants to perform an operation over the system. Clients interact with the system through a presentation layer

The application logic determines what the system actually does. It takes care of enforcing the business rules and establish the business processes. The application logic can take many forms: programs, constraints, business processes, etc.

The resource manager deals with the organization (storage, indexing, and retrieval) of the data necessary to support the application logic. This is typically a database but it can also be a text retrieval system or any other data management system providing querying capabilities and persistence.

## A Game of Boxes and Arrows

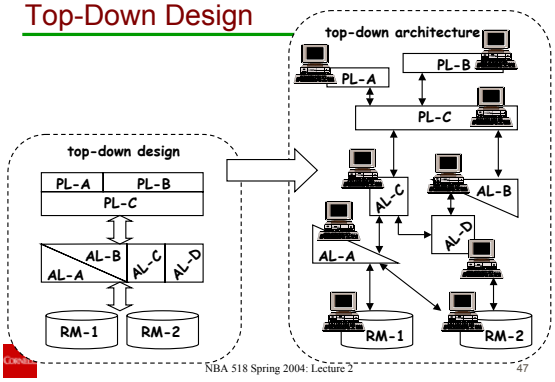


There is no problem in system design that cannot be solved by adding a level of indirection. There is no performance problem that cannot be solved by removing a level of indirection.

- Each box represents a part of the system.
- Each arrow represents a connection between two parts of the system.
- The more boxes, the more modular the system: more opportunities for distribution and parallelism. This allows encapsulation, component based design, reuse.
- The more boxes, the more arrows: more sessions (connections) need to be maintained, more coordination is necessary. The system becomes more complex to monitor and manage.
- The more boxes, the greater the number of context switches and intermediate steps to go through before one gets to the data. Performance suffers considerably.
- System designers try to balance the flexibility of modular design with the performance demands of real applications. Once a layer is established, it tends to migrate down and merge with lower layers.

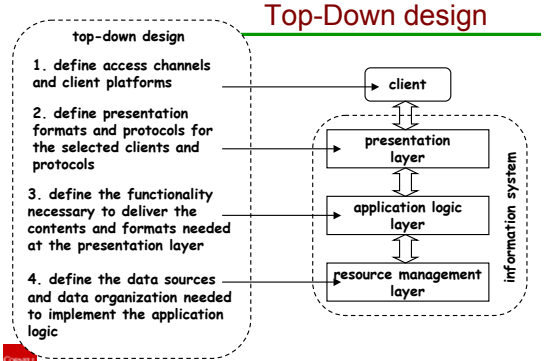


## Top-Down Design

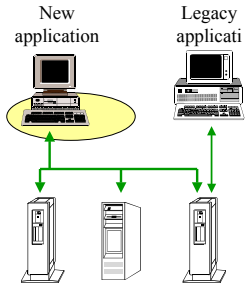





## Top-Down design



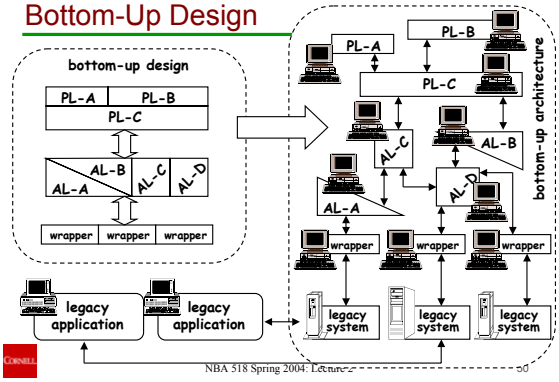
## Bottom-Up Design




- In a bottom up design, many of the basic components already exist. These are stand alone systems which need to be integrated into new systems.
- The components do not necessarily cease to work as stand alone components. Often old applications continue running at the same time as new applications.
- This approach has a wide application because the underlying systems already exist and cannot be easily replaced.
- Much of the work and products in this area are related to middleware, the intermediate layer used to provide a common interface, bridge heterogeneity, and cope with distribution.

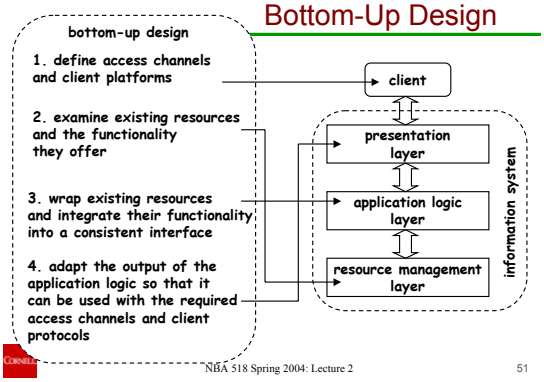


## Bottom-Up Design

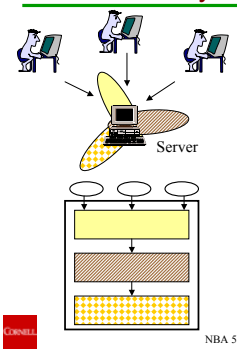





## Bottom-Up Design



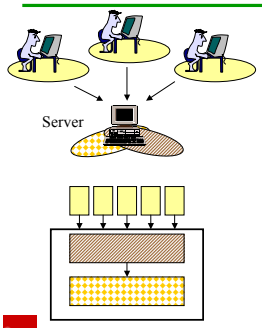
## One Tier: Fully Centralized




- The presentation layer, application logic and resource manager are built as a monolithic entity.
- Access through dumb terminals
- This was the typical architecture of mainframes, offering several advantages:
  - no forced context switches in the control flow (everything happens within the system),
  - all is centralized, managing and controlling resources is easier,
  - the design can be highly optimized by blurring the separation between layers.



## Two Tier: Client/Server

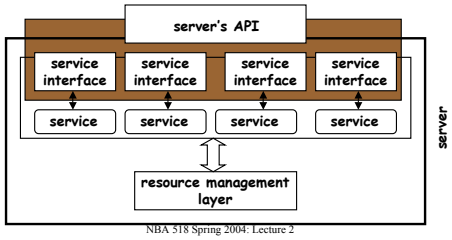



- As computers became more powerful, it was possible to move the presentation layer to the client. This has several advantages:
  - Clients are independent.
  - Computing power at clients.
  - It introduces the concept of API (Application Program Interface). An interface to invoke the system from the outside. It also allows designers to think about federating the systems into a single system.
  - The resource manager only sees one client: the application logic. This greatly helps with performance since there are no client connections/sessions to maintain.



## APIs in Client/Server

- Introduced notion of a service
- Introduced notion of an interface (how the client can invoke a given service)
- Many standardization efforts due to need for common APIs

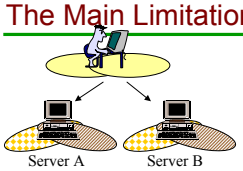


## Technical Aspects Of Two Tier

- Advantages to Single Tier:
  - Take advantage of client capacity to off-load work to the clients
  - Work within the server takes place within one scope (almost as in 1 tier),
  - The server design is still tightly coupled and can be optimized by ignoring presentation issues
  - Still relatively easy to manage and control from a software engineering point of view
- Disadvantages:
  - Connection management
  - Clients are "tied" to the system (no standard presentation layer). Connect to two systems, a client needs two presentation layers.
  - No failure or load encapsulation. If the server fails, nobody can work.
  - The load created by one client will directly affect the work of others since they are all competing for the same resources.



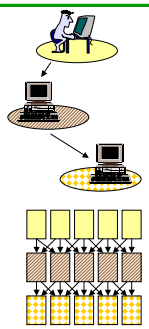
## The Main Limitation of Client/Server




- Accessing more than two servers:
  - The underlying systems don't know about each other
  - No common business logic
  - Client is the point of integration (increasingly fat clients)
  - The responsibility of dealing with heterogeneous systems is shifted to the client.
  - The client becomes responsible for knowing where things are, how to get to them, and how to ensure consistency
- Very inefficient (software design, portability, code reuse, performance since the client capacity is limited, etc.).
- These issues cannot be solved with 2-tier

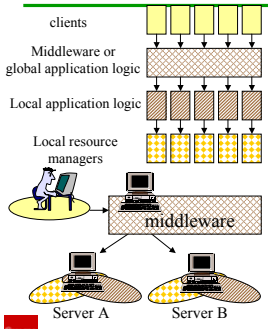


## Three Tier: Middleware



- Three layers are fully separated.
- The layers are also typically distributed taking advantage of the complete modularity of the design

## Middleware




- Middleware is just a level of indirection between clients and other layers of the system.
- Introduces an additional layer of business logic encompassing all underlying systems.
- By doing this, a middleware system:
  - simplifies the design of the clients by reducing the number of interfaces,
  - provides transparent access to the underlying systems,
  - acts as the platform for inter-system functionality and high level application logic, and
  - takes care of locating resources, accessing them, and gathering results.

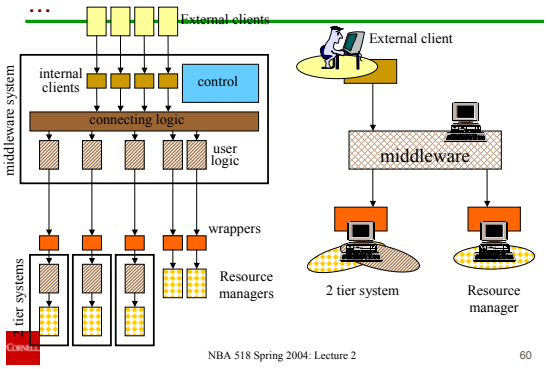


## Technical Aspects of Middleware

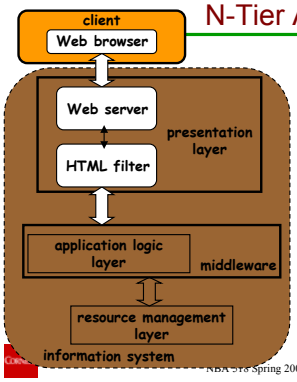
- The introduction of a middleware layer helps in that:
  - the number of necessary interfaces is greatly reduced:
    - clients see only one system (the middleware),
    - local applications see only one system (the middleware),
  - it centralizes control (middleware systems themselves are usually 2 tier),
  - it makes necessary functionality widely available to all clients,
  - it allows to implement functionality that otherwise would be very difficult to provide, and
  - it is a first step towards dealing with application heterogeneity (some forms of it).
- The middleware layer does not help in that:
  - it is another indirection level,
  - it is complex software,
  - it is a development platform, not a complete system



## A three tier middleware based system



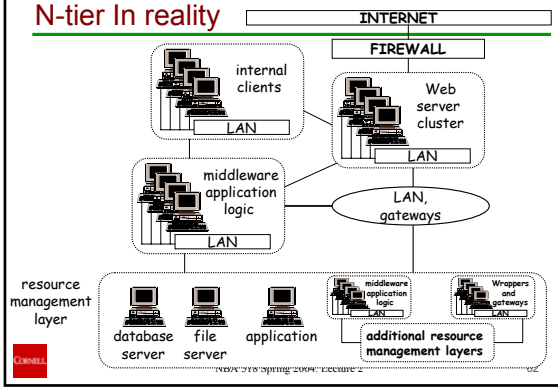
## N-Tier Architectures




- N-tier architectures result from connecting several three tier systems to each other
- The addition of the Web layer led to the notion of "application servers", which was used to refer to middleware platforms supporting access through the Web



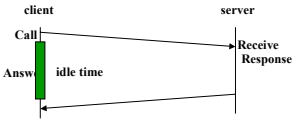
## N-tier In reality






## Blocking or Synchronous Interaction

- Traditionally, information systems use blocking calls Synchronous interaction requires both parties to be "on-line": the caller makes a request, the receiver gets the request, processes the request, sends a response, the caller receives the response.
- The caller must wait until the response comes back. but the interaction requires both client and server to be "alive" at the same time

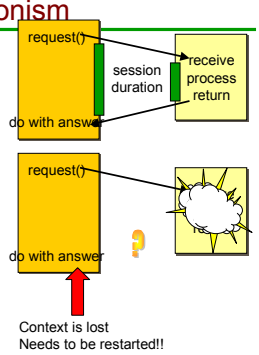



Disadvantages due to synchronization:

- Connection overhead
- Higher probability of failures
- Difficult to identify and react to failures
- It is not really practical for complex interactions

## Overhead of Synchronism

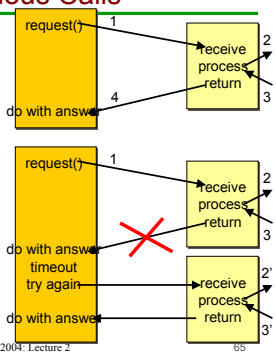
- Need to maintain a session between the caller and the receiver.
- Maintaining sessions is expensive. There is also a limit on how many sessions can be active at the same time
- For this reason, client/server systems often resort to connection pooling to optimize resource utilization
  - Have a pool of open connections
  - Allocate connections as needed
- Synchronous interaction requires a context for each call and a context management system for all incoming calls.






## Failures In Synchronous Calls

- If the client or the server fail, the context is lost.
  - If the failure occurred before 1, nothing has happened
  - If the failure occurs after 1 but before 2 (receiver crashes), then the request is lost
  - If the failure happens after 2 but before 3, side effects may cause inconsistencies
  - If the failure occurs after 3 but before 4, the response is lost but the action has been performed (do it again?)
- Who is responsible for finding out what happened?
- Finding out when the failure took place may not be easy. If there is a chain of invocations the failure can occur anywhere along the chain.






## Two Solutions

ENHANCED SUPPORT

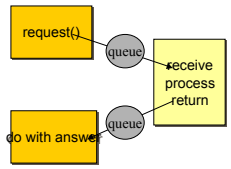
- Client/Server systems and middleware platforms provide a number of mechanisms to deal with the problems created by synchronous interaction:
  - Transactional interaction
  - Service replication and load balancing

ASYNCHRONOUS INTERACTION

- Using asynchronous interaction, the caller sends a message that gets stored somewhere until the receiver reads it and sends a response. The response is sent in a similar manner
- Asynchronous interaction can take place in two forms:
  - Non-blocking invocation
  - Persistent queues

## Message Queuing

- Reliable queuing is an excellent complement to synchronous interactions:
  - Suitable to modular design: the code for making a request can be in a different module (even a different machine!) than the code for dealing with the response
  - Easier to design sophisticated distribution modes and it also helps to handle communication sessions in a more abstract way
  - More natural way to implement complex interactions between heterogeneous systems






## Summary

- Functionality of database systems
- Three-tier architectures