Database and Data Center Security

- We now look at the **unique security issues** that relate to databases
- The focus is on **relational database management systems** (RDBMS), as the relational approach dominates industry, government, and research sectors and is likely to do so for the foreseeable future

Index

- Database Management Systems
- The Need for Database Security
- Relational Databases
- SQL Injection Attacks
- Database Access Control
- Inference
- Database Encryption
- Data Center Security

Learning Objectives

- Present an overview of the **basic elements** of a database system and of a **relational** database management system
- Understand the unique **need for database security**, separate from ordinary computer security measures
- Define and explain **SQL injection** attacks
- Describe approaches to **database access control**
- Explain security threats to database systems posed by **inference**
- Discuss the use of **encryption** in a database system
- Discuss security issues related to **data centers**

Index

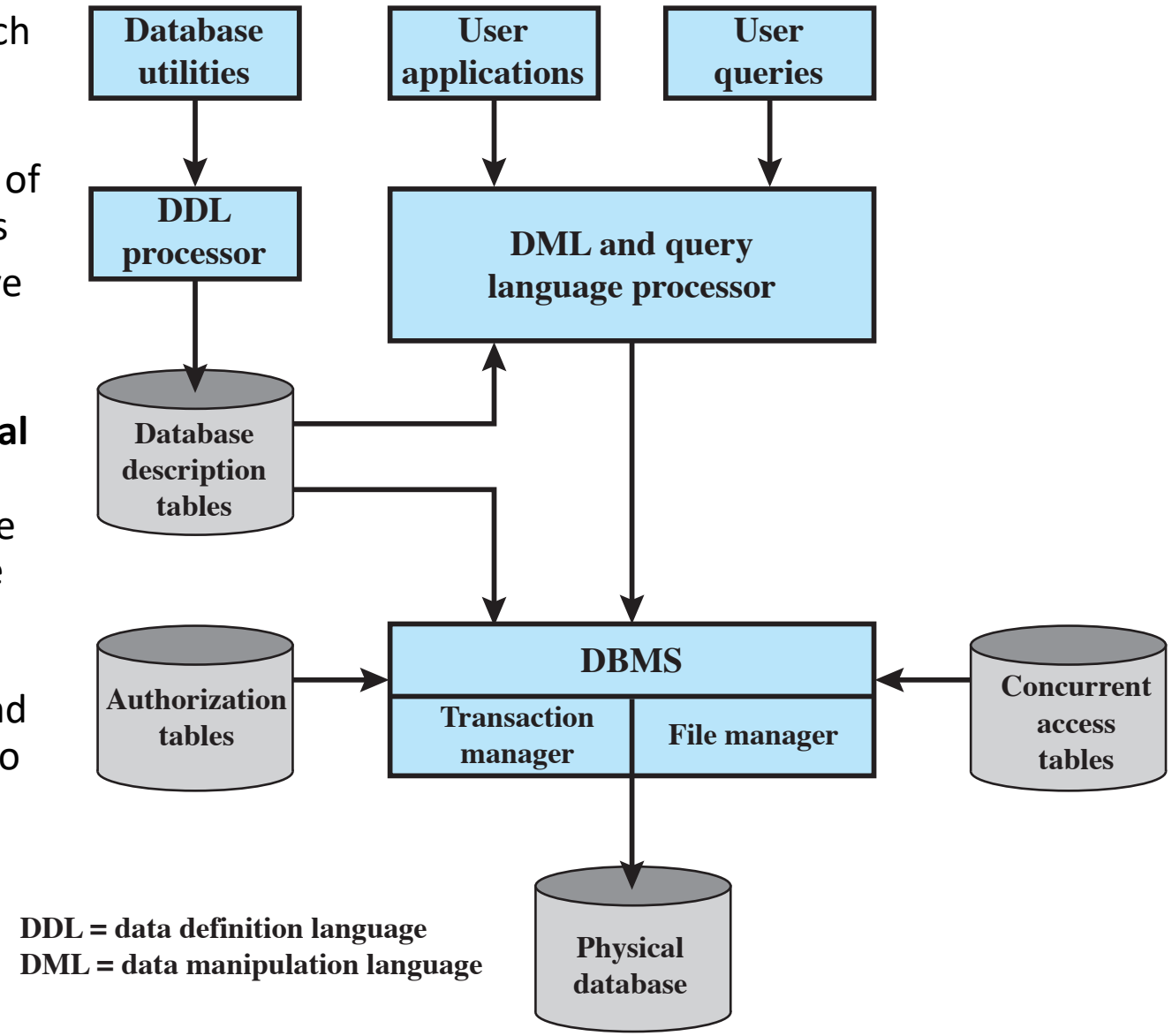
- Database Management Systems
- The Need for Database Security
- Relational Databases
- SQL Injection Attacks
- Database Access Control
- Inference
- Database Encryption
- Data Center Security

Database Management System

- A database is a **structured collection of data** stored for use by one or more applications
- In addition to data, contains the **relationships** between single data items and groups of data items
- A database management system **(DBMS)** is a suite of programs for
  - constructing and maintaining the database
  - offering ad hoc **query** facilities to multiple users and applications
    - A **query language** provides a uniform interface to the database for users and applications

DBMS Architecture

- **DB designers/administrators** make use of a DDL to define the database logical structure, which is represented by a set of **database description tables**
- A **DML** provides a powerful set of tools for application developers
- **Query languages** are declarative languages designed to support end users
- The **DBMS** manages the **physical database** by relying on the database description tables, the **authorization tables** (to ensure the user has permission to execute the query language statement on the database), and the **concurrent access tables** (to prevent conflicts when simultaneous, conflicting commands are executed)
- The interface to the physical database is through a **file manager module** and a **transaction manager module**



Index

- Database Management Systems
- The Need for Database Security
- Relational Databases
- SQL Injection Attacks
- Database Access Control
- Inference
- Database Encryption
- Data Center Security

The Need For Database Security

- Databases can sometimes contain *sensitive data* that needs to be secured
  - Corporate financial data
  - Customer and employee information
  - Proprietary product information
  - Health care information and medical records
- It is important to be able to provide customers, partners, and employees with access to this information
- But such information can be targeted by internal and external threats of misuse or of unauthorized change
- Thus security specifically tailored to databases is an increasingly important component of an overall organizational security strategy

The Need For Database Security

- Database security has **not** kept pace with the increased reliance on databases
  - The **increasing complexity** of the DBMS has brought a number of new vulnerabilities and the potential for misuse
  - Databases have a **sophisticated interaction protocol** called the Structured Query Language (SQL)
  - Typical organization **lacks full-time database security personnel**
  - Most enterprise environments consist of a **heterogeneous mixture** of database platforms, enterprise platforms, and OS platforms
- An additional *recent challenge for organizations* is their increasing reliance on **cloud technology** to host part or all of the corporate database

The Need For Database Security

- DB systems generate security requirements that are beyond the capability of typical OS-based security mechanisms or stand-alone security packages
- **OS security mechanisms** typically control read/write access to entire files
  - But they could not be used to limit access to specific records or fields in that file
- A DBMS typically **allows** this type of more detailed access control to be specified
  - It also usually enables access controls to be specified over a **wider range of commands**, such as to select, insert, update, or delete specified items in the database
- Thus, security services and mechanisms are needed that are **designed specifically** for, and integrated with, database systems

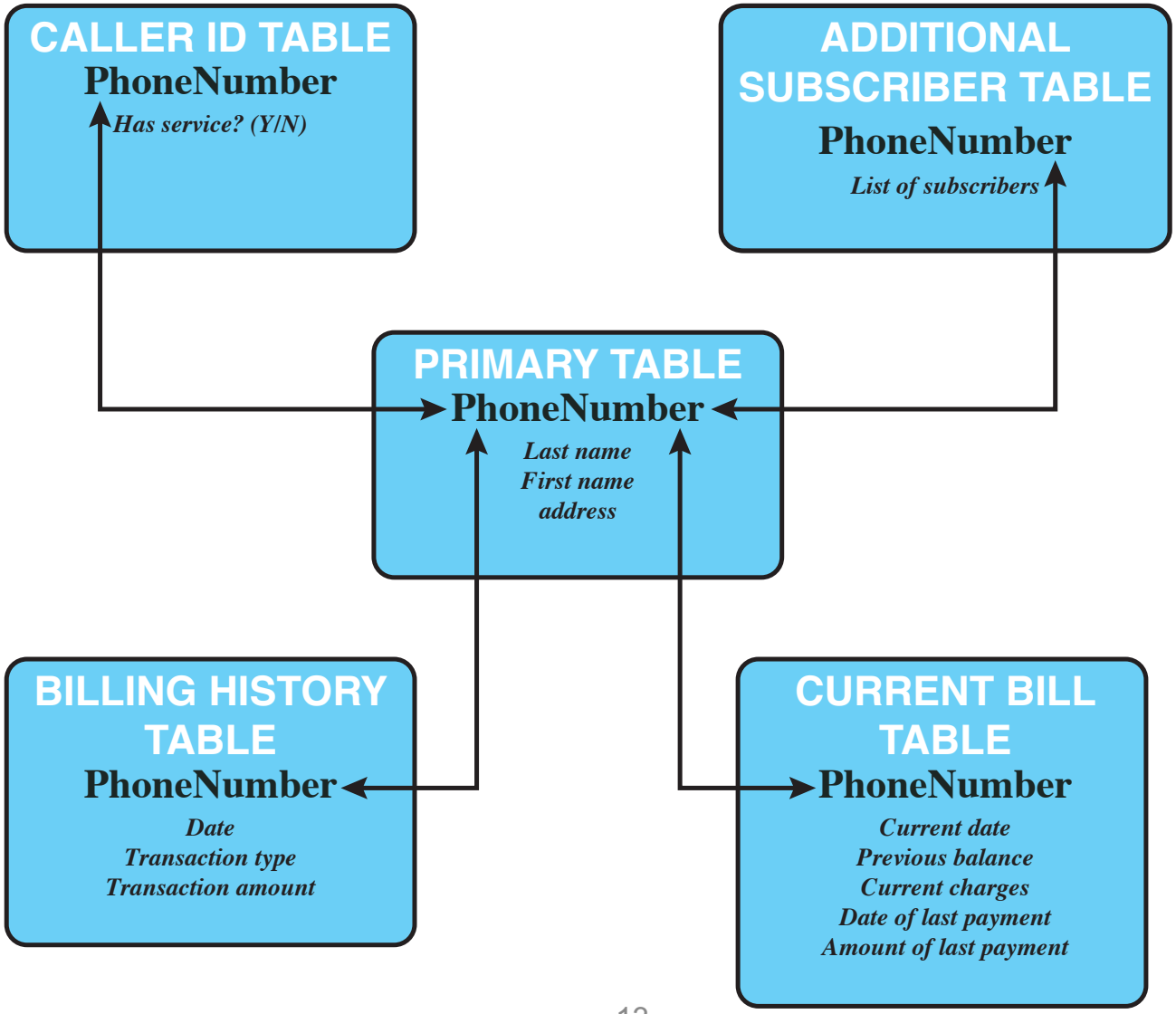
Index

Relational Databases

- The basic building block of a relational database is a table of data consisting of rows and columns
  - Each column holds a particular type of data
  - Each row (entry, record) contains a specific value for each column
  - Ideally, there is at least one column where all values are unique, serving as an identifier/key for a given entry
  - Such a table is called a *flat file* because all of the data are stored in a single two-dimensional (rows and columns) table
- **Drawbacks** of using a single table
  - Some of the column positions for a given row may be blank (not used)
  - Every time a new service or new type of information is incorporated in the database, more columns must be added and the database and accompanying software must be redesigned and rebuilt
- To overcome these drawbacks multiple tables can be linked together by a unique identifier that is present in all tables

Example Relational Database

A relational database uses multiple tables related to one another by a designated **key**; in this case the key is the **PhoneNumber** field



Relational Databases

- Users and applications use a relational query language to access the database
  - The query language uses **declarative statements** rather than the procedural instructions of a programming language
  - In essence, the query language allows the user to request selected items of data from **all records** that fit a given set of conditions
  - The DBMS takes care of extracting the requested data from one or more tables

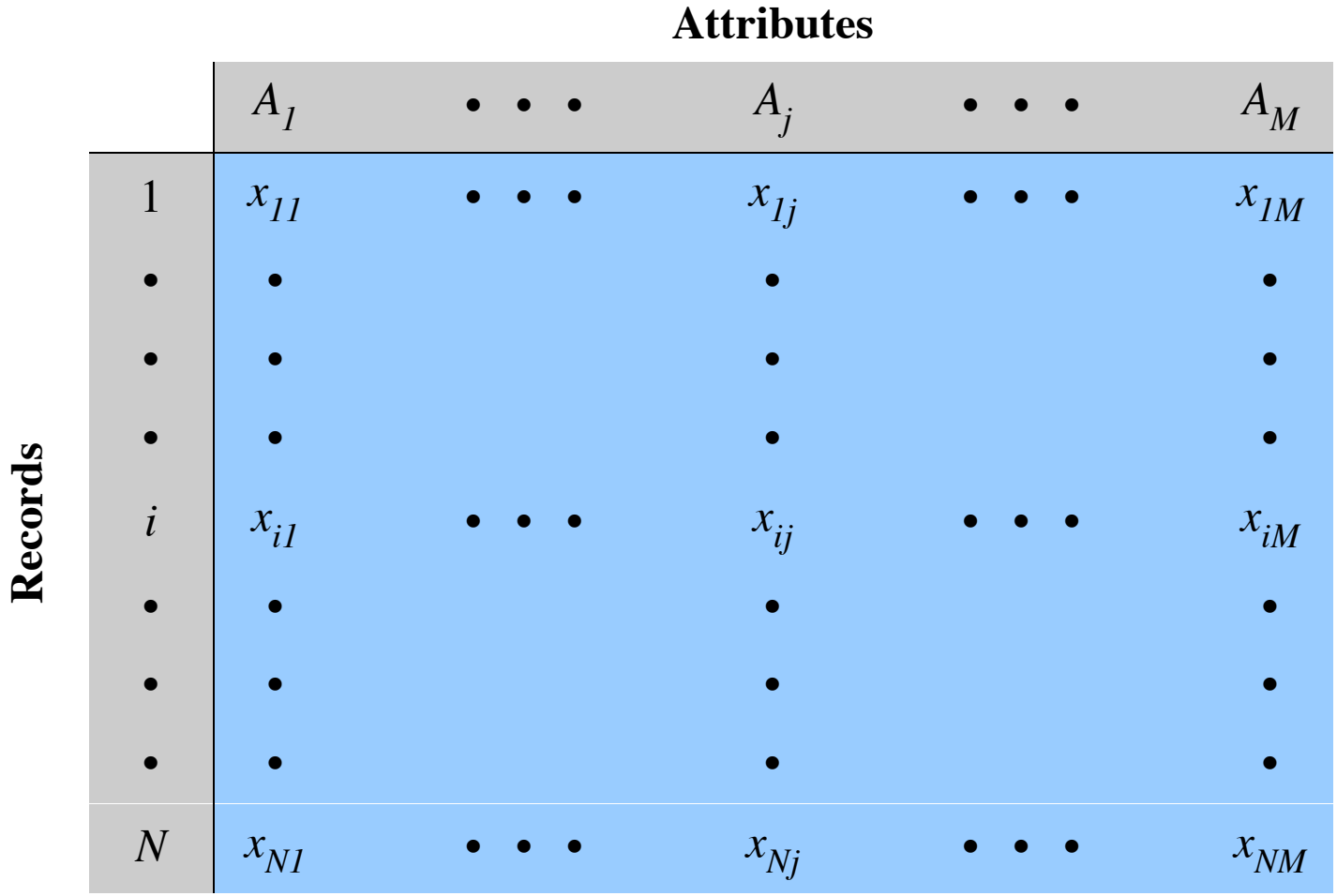
Basic Terminology for Relational Databases

Formal Name	Common Name	Also Known As
Relation	Table	File
Tuple	Row	Record
Attribute	Column	Field

In relational database parlance

- the basic building block is a **relation**, which is a flat table
- rows are referred to as **tuples**, and
- columns are referred to as **attributes**

Abstract Model of a Relational Database Table



There are *N tuples* (or *entities* or *individuals*) and *M attributes*

Each attribute A_j has $|A_j|$ possible values, with x_{ij} denoting the value of attribute j for entity i

Relational Database Elements

Primary key

- A portion of a row that uniquely identifies the row in a table
- Consists of one or more column names

Foreign key

- Attributes defining the primary key in one table that occur as attributes in another table
- Links one table to another one

View (or virtual table)

- The result of a query that returns selected rows and columns from one or more tables
- Often used for security purposes to provide restricted access to a relational database to a user or application

Relational Database Example

Department Table

Did	Dname	Dacctno
4	human resources	528221
8	education	202035
9	accounts	709257
13	public relations	755827
15	services	223945

primary key

Relational Database Example

Department Table

Did	Dname	Dacctno
4	human resources	528221
8	education	202035
9	accounts	709257
13	public relations	755827
15	services	223945

primary key

Employee Table

Ename	Did	Salarycode	Eid	Ephone
Robin	15	23	2345	6127092485
Neil	13	12	5088	6127092246
Jasmine	4	26	7712	6127099348
Cody	15	22	9664	6127093148
Holly	8	23	3054	6127092729
Robin	8	24	2976	6127091945
Smith	9	21	4490	6127099380

foreign key

primary key

Two tables in a relational database

N.B. a foreign key value can appear *multiple* times in a table

Relational Database Example

Department Table

Did	Dname	Dacctno
4	human resources	528221
8	education	202035
9	accounts	709257
13	public relations	755827
15	services	223945

primary key

Employee Table

Ename	Did	Salarycode	Eid	Ephone
Robin	15	23	2345	6127092485
Neil	13	12	5088	6127092246
Jasmine	4	26	7712	6127099348
Cody	15	22	9664	6127093148
Holly	8	23	3054	6127092729
Robin	8	24	2976	6127091945
Smith	9	21	4490	6127099380

foreign key

primary key

Two tables in a relational database

N.B. a foreign key value can appear *multiple* times in a table

Dname	Ename	Eid	Ephone
human resources	Jasmine	7712	6127099348
education	Holly	3054	6127092729
education	Robin	2976	6127091945
accounts	Smith	4490	6127099380
public relations	Neil	5088	6127092246
services	Robin	2345	6127092485
services	Cody	9664	6127093148

A **view** that includes the employee name, ID, and phone number from the Employee table and the corresponding department name from the Department table

Structured Query Language (SQL)

- A **standardized language** (ANSI 1986/ISO 1987) to define schema, and manipulate and query data in relational databases
- Several similar versions of the standard, all follow the same basic syntax and semantics

SQL statements can be used to:

- Create tables
- Insert and delete data in tables
- Create views
- Retrieve data with query statements

- Despite the name, it is **not just a query language**, but some of its subsets deal with the creation, management and administration of the database

SQL Example

Department Table

Did	Dname	Dacctno
4	human resources	528221
8	education	202035
9	accounts	709257
13	public relations	755827
15	services	223945

primary key

Employee Table

Ename	Did	Salarycode	Eid	Ephone
Robin	15	23	2345	6127092485
Neil	13	12	5088	6127092246
Jasmine	4	26	7712	6127099348
Cody	15	22	9664	6127093148
Holly	8	23	3054	6127092729
Robin	8	24	2976	6127091945
Smith	9	21	4490	6127099380

foreign key

primary key

The two previous **tables** are defined as follows:

```
CREATE TABLE department (
    Did INTEGER PRIMARY KEY,
    Dname CHAR (30),
    Dacctno CHAR (6) )
CREATE TABLE employee (
    Ename CHAR (30),
    Did INTEGER,
    SalaryCode INTEGER,
    Eid INTEGER PRIMARY KEY,
    Ephone CHAR (10),
    FOREIGN KEY (Did) REFERENCES department (Did) )
```

SQL Example

Department Table

Did	Dname	Dacctno
4	human resources	528221
8	education	202035
9	accounts	709257
13	public relations	755827
15	services	223945

primary key

Employee Table

Ename	Did	Salarycode	Eid	Ephone
Robin	15	23	2345	6127092485
Neil	13	12	5088	6127092246
Jasmine	4	26	7712	6127099348
Cody	15	22	9664	6127093148
Holly	8	23	3054	6127092729
Robin	8	24	2976	6127091945
Smith	9	21	4490	6127099380

foreign key

primary key

The **basic command** for retrieving information is the `SELECT` statement:

```
SELECT Ename, Eid, Ephone
    FROM Employee
    WHERE Did = 15
```

This query returns the `Ename`, `Eid`, and `Ephone` fields from the Employee table for all employees assigned to department 15

SQL Example

Department Table

Did	Dname	Dacctno
4	human resources	528221
8	education	202035
9	accounts	709257
13	public relations	755827
15	services	223945

primary key

Employee Table

Ename	Did	Salarycode	Eid	Ephone
Robin	15	23	2345	6127092485
Neil	13	12	5088	6127092246
Jasmine	4	26	7712	6127099348
Cody	15	22	9664	6127093148
Holly	8	23	3054	6127092729
Robin	8	24	2976	6127091945
Smith	9	21	4490	6127099380

foreign key

primary key

Dname	Ename	Eid	Ephone
human resources	Jasmine	7712	6127099348
education	Holly	3054	6127092729
education	Robin	2976	6127091945
accounts	Smith	4490	6127099380
public relations	Neil	5088	6127092246
services	Robin	2345	6127092485
services	Cody	9664	6127093148

The **view** above is created using the following SQL statement:

```
CREATE VIEW newtable (Dname, Ename, Eid, Ephone)
AS SELECT D.Dname, E.Ename, E.Eid, E.Ephone
FROM Department D Employee E
WHERE E.Did = D.Did
```

Index

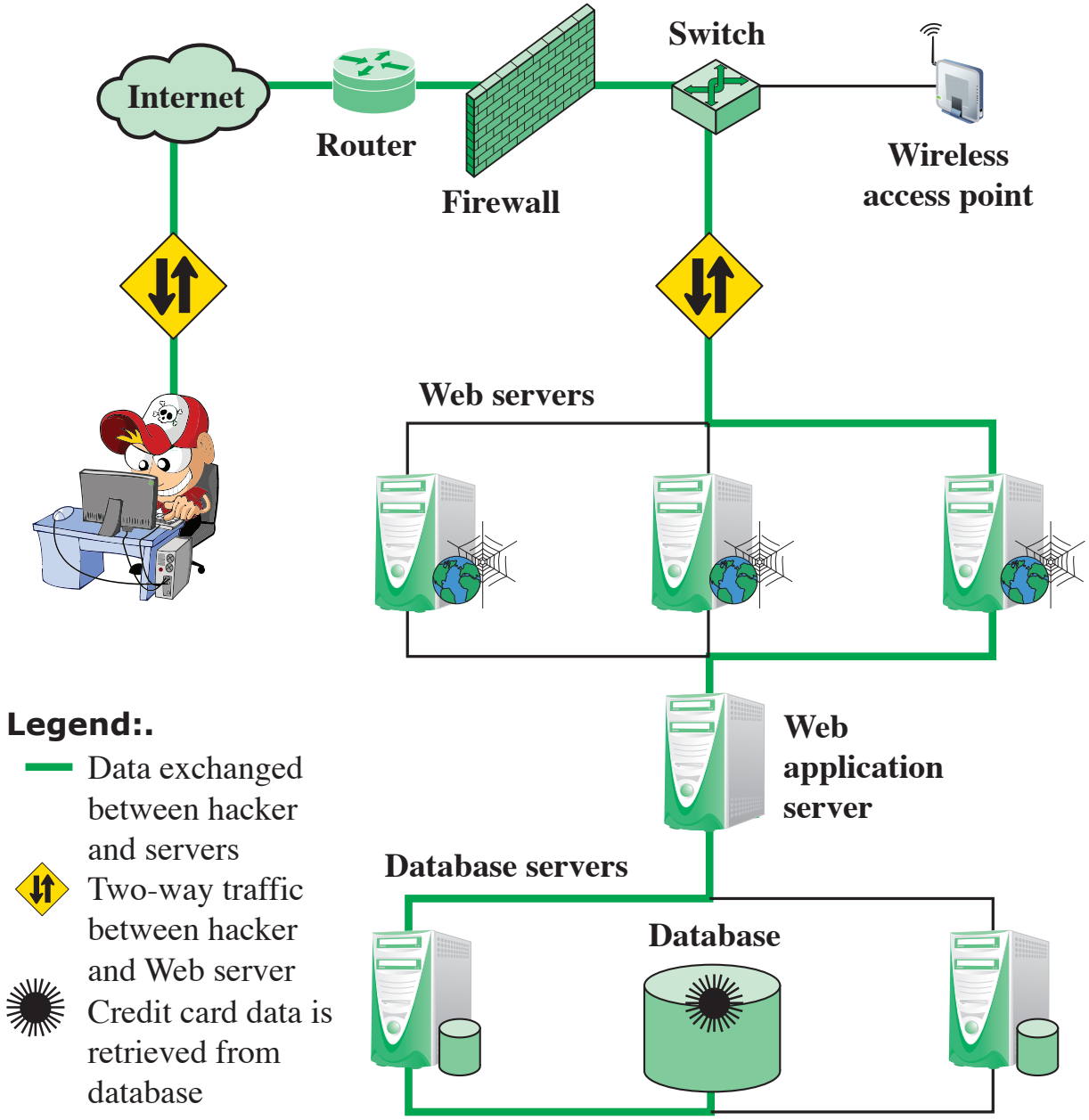
SQL Injection Attacks (SQLi)

- One of the most prevalent and dangerous network-based security threats
- Designed to exploit the nature of current Web application pages
  - Most current Web sites have *dynamic components* and *content*
  - This dynamic content is usually transferred to and from back-end databases that contain volumes of information
- A SQLi attack sends **malicious SQL** commands to the database server
- Most common attack goal is **bulk extraction** of data
- Depending on the environment, SQL injection can **also** be exploited to:
  - Modify or delete data
  - Execute arbitrary OS commands
  - Launch DoS attacks

A Typical SQLi Attack

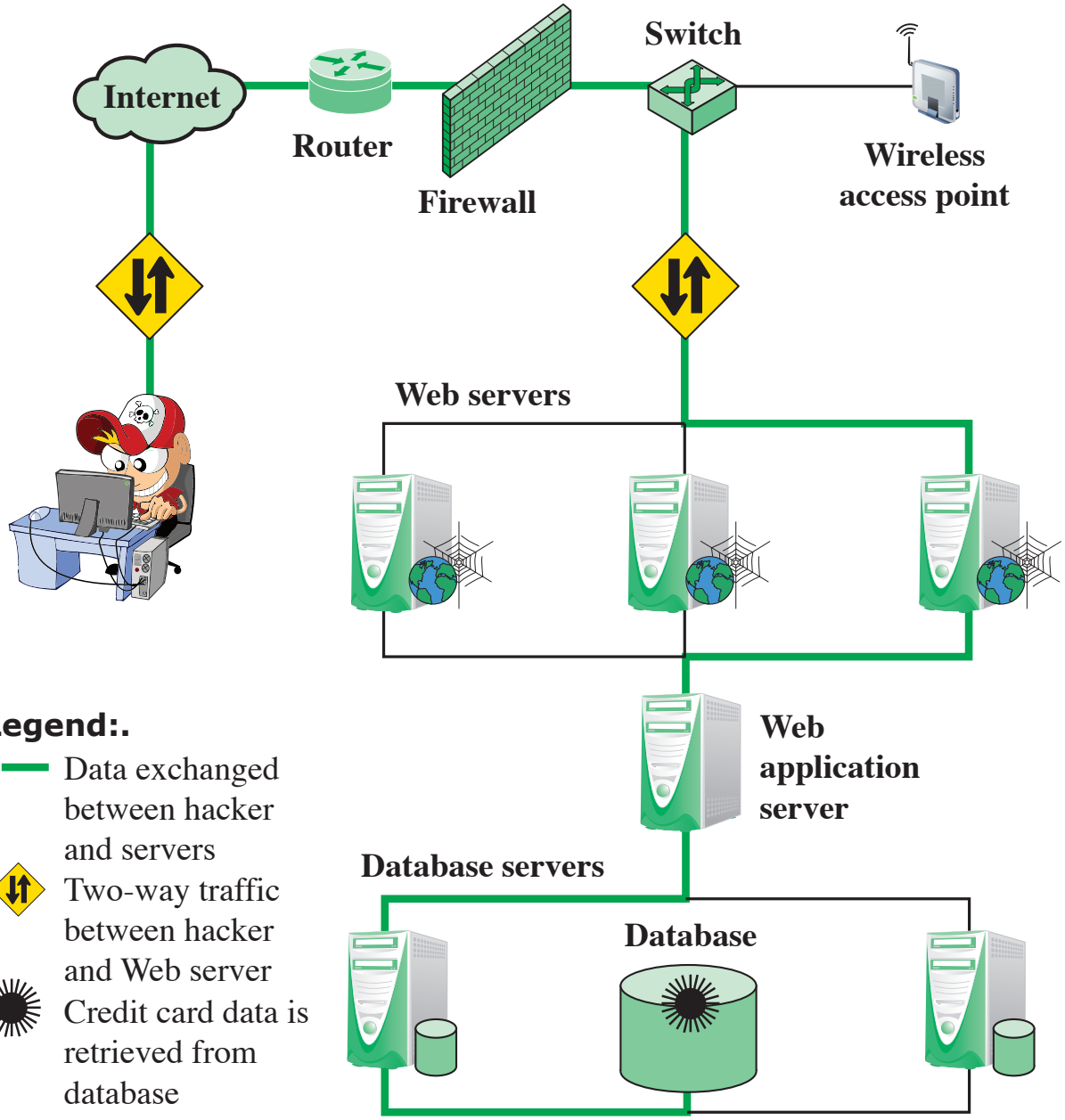
- Exploits a security vulnerability occurring in the **database layer** of an application
- The attack is viable when **user input** is
  - *incorrectly filtered* for string literal escape characters embedded in SQL statements
  - *not strongly typed*

  and thereby unexpectedly executed



A Typical SQLi Attack: involved steps

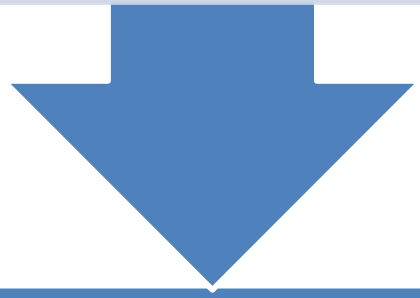
1. Hacker finds a vulnerability in a custom Web application and injects an SQL command to a database by sending the command to the Web server
2. The command is injected into traffic that will be accepted by the firewall, since client browsers communicate with web servers using HTTP
3. The Web server receives the malicious code and sends it to the Web application server
4. The Web application server receives the malicious code from the Web server and sends it to the database server using databases languages, such as SQL
5. The database server executes the malicious code on the database
6. The database returns data from credit cards table
7. The Web application server dynamically generates a page with data including credit card details from the database
8. The Web server sends the credit card details to the hacker



Injection Technique

The SQLi attack typically works by prematurely terminating a text string and appending a new command

Because the SQL command may have additional strings appended to it before it is executed, the attacker terminates the injected string with a comment mark "-- " (or "#")



Subsequent text is ignored at execution time

SQLi attack Example

Consider a script that builds an SQL query by combining predefined strings with text entered by a user:

```
var ShipCity;
ShipCity = Request.form ("ShipCity");
var sql = "SELECT * FROM OrdersTable WHERE ShipCity = '" +
          ShipCity + "'";
```

The intention of the script's designer is that when the script is executed, the user is prompted to enter the name of a city which is then assigned to `ShipCity`

Whatever data are typed in the form, they will eventually become a part of the string, representing an SQL command which will be executed by the database, assigned to the variable `sql`

Thus, although users do not directly interact with the database, there exists a channel between user and database which creates a new attack surface for the database

If the user enters Redmond, then the following SQL query is generated:

```
SELECT * FROM OrdersTable WHERE ShipCity = 'Redmond'
```

that selects all records in `OrdersTable` where `ShipCity` is `Redmond`

SQLi attack Example

Consider a script that build an SQL query by combining predefined strings with text entered by a user:

```
var ShipCity;
ShipCity = Request.form ("ShipCity");
var sql = "SELECT * FROM OrdersTable WHERE ShipCity = '" +
          ShipCity + "'";
```

The intention of the script's designer is that when the script is executed, the user is prompted to enter the name of a city which is then assigned to `ShipCity`

Suppose, however, the user enters the following:

```
Redmond'; DROP table OrdersTable--
```

This results in the following SQL query:

```
SELECT * FROM OrdersTable WHERE ShipCity = 'Redmond';
DROP table OrdersTable--
```

The semicolon is an indicator that separates two commands, and the double dash is an indicator that the remaining text of the current line is a comment and not to be executed

When the SQL server processes this statement

- first it selects all records in `OrdersTable` where `ShipCity` is `Redmond`
- then, it executes the `DROP` request, which deletes the entire table!

SQLi attack Example

Consider a script that build an SQL query by combining predefined strings with text entered by a user:

```
var ShipCity;
ShipCity = Request.form ("ShipCity");
var sql = "SELECT * FROM OrdersTable WHERE ShipCity = '" +
          ShipCity + "'";
```

The intention of the script's designer is that when the script is executed, the user is prompted to enter the name of a city which is then assigned to `ShipCity`

- By typing some special characters, such as apostrophe (') and double dash (--, followed by at least a withespace or control character), the user can thus change the meaning fo the SQL statement
- This is a security breach!
- Thus, the channel between the user and the database must be properly protected, otherwise malicious users can launch attacks to the database through the channel

SQLi Attack Avenues

User input

- Inject SQL commands by providing suitable crafted user input (e.g. through form submission)

Server variables

- Forge the values that are placed in HTTP and network headers and exploit this vulnerability by placing data directly into the headers

Second-order injection

- Rely on data already present in the system or database to trigger an SQL injection attack, so when the attack occurs, the input that modifies the query to cause an attack does not come from the user, but from within the system itself

Cookies

- Alter cookies such that when the application server builds an SQL query based on the cookie's content, the structure and function of the query is modified

Physical user input

- Supply input, e.g. conventional barcodes, RFID tags, paper forms which are scanned using optical character recognition, that constructs an attack outside the realm of web requests

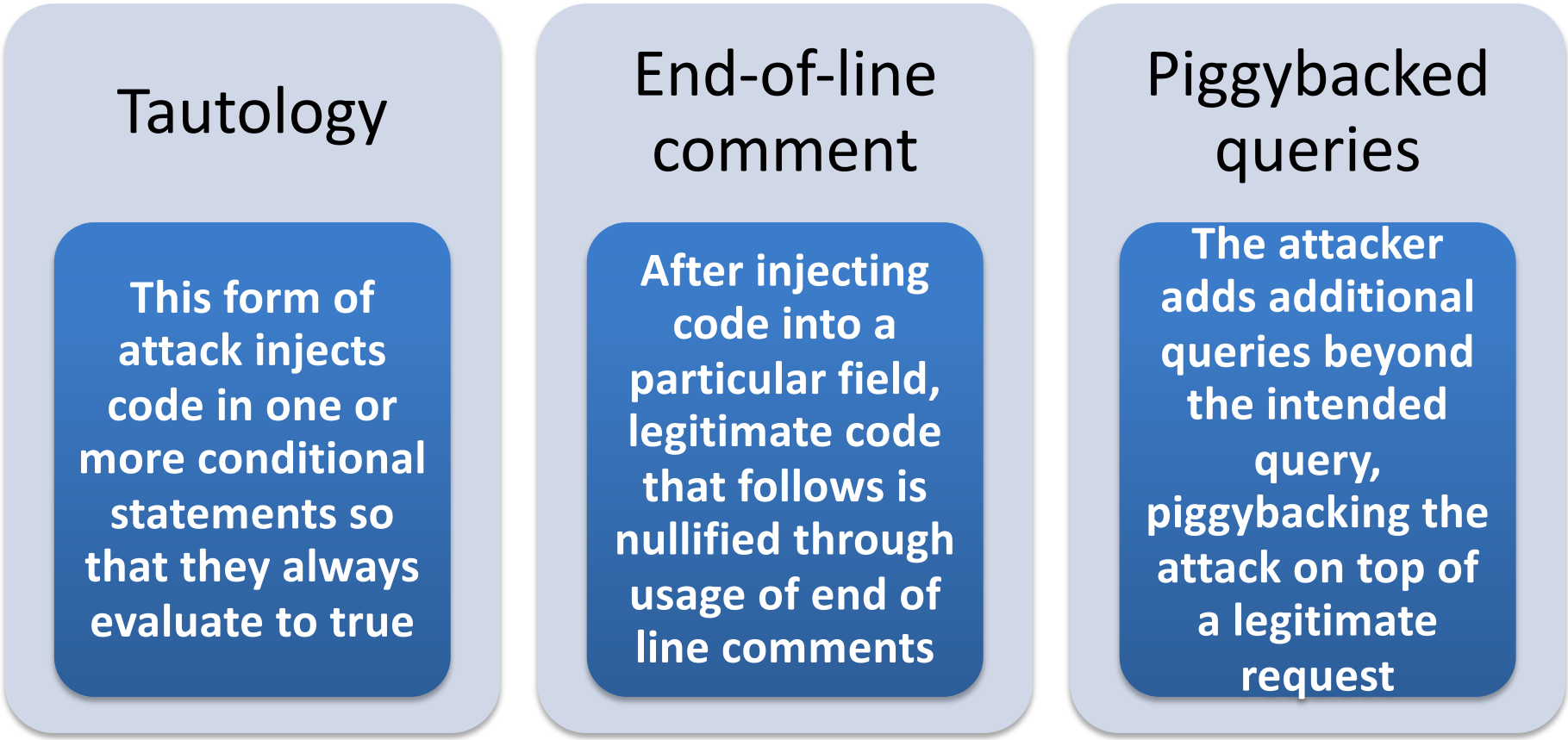
SQLi Attack Types

Attack types can be grouped into three main categories:

- inband
- inferential
- out-of-band

Inband Attacks

- Use the **same** communication channel for injecting SQL code and retrieving results
  - The retrieved data are presented directly in the application Web page
  - May include the following aspects:



Example of Tautology Attacks

Consider this script, whose intent is to require the user to enter a valid name and password

```
$query = "SELECT info FROM user WHERE name =
'$_GET["name"]' AND pwd = '$_GET["pwd"]'";
```

Suppose the attacker submits "' OR 1=1 -- " for the name field
The resulting query would look like this

```
$query = "SELECT info FROM user WHERE name =
'' OR 1=1 -- ' AND pwd = '$_GET["pwd"]'";
```

The injected code effectively disables the password check (because of the **end-of-line comment** indicator "-- ") and turns the entire WHERE clause into a **tautology**

The database uses the conditional as the basis for evaluating each row and deciding which ones to return to the application

Hence, the query evaluates to true for each row in the table and returns all of them!

Inferential Attacks

- The attacker is able to reconstruct some information by sending specific requests and observing the resulting behavior of the Web/database server
  - There is no actual transfer of data
- Include:
  - **Illegal/logically incorrect queries**
    - The vulnerability leveraged by this attack is that the default error page returned by application servers is often overly descriptive and can reveal vulnerable/injectable parameters to an attacker
    - This is often a preliminary, information-gathering step for other attacks
  - **Blind SQL injection**
    - Allows attackers to infer the data present in a DB even when the system is sufficiently secure to not display any erroneous information back to the attacker
    - The attacker asks the server true/false questions
      - If the injected statement evaluates to true, the site continues to function normally
      - Otherwise, although there is no descriptive error message, the page differs significantly from the normally functioning page

Out-of-Band Attacks

- Data are retrieved using a different channel (e.g., an email with the results of the query)

- This can be used when there are limitations on information retrieval, but outbound connectivity from the database server is lax

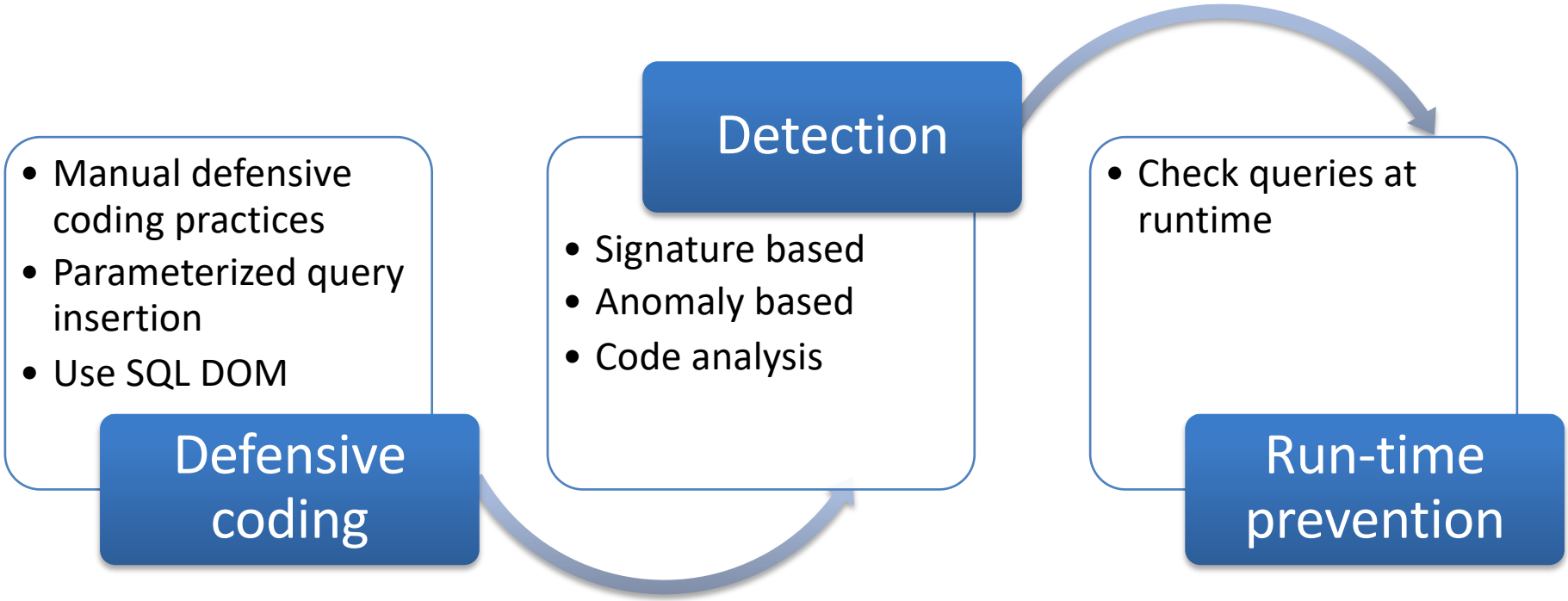
The Foundamental Cause of Vulnerability

Mixing data and code together is the cause of several types of vulnerabilities and attacks, including the *cross-site scripting attack* and the *format string attack*

- They all mix two pieces of information together:
  - one piece comes from users, which are untrusted
  - the other piece is typically provided by the program, which is usually trusted
- Initially the developer knows the boundaries between them, once they are mixed the boundaries disappear
  - In the SQL injection cases, if the user data input contains some special characters reserved for code, they alter the original boundaries between code and data
- After the two pieces of information are mixed, the result is passed to a parser
  - For SQL, it is database's SQL parser
  - For the case of cross-site scripting it is the HTML parser
- The parser needs to separate data and code, so the code can be executed
- If data contain keywords or special characters, they will be interpreted as code, even though originally they are part of data, since the parser does not know the original boundaries between code and data
- This is how attackers can inject code into a vulnerable program via data channel

SQLi Countermeasures

- Because SQLi attacks are so prevalent, damaging, and varied, both by attack venue and type, an **integrated set** of techniques is necessary
- The countermeasures can be classified into **three types**



SQLi Countermeasures

Defensive coding

- **Manual input validation**, e.g. input type checking (inputs that are supposed to be numeric contain no characters other than digits)
- Use **prepared statements** to send an SQL statement template to the database, with certain values left unsepcified (they are called parameters); next bind values to the parameters and ask the database to execute the statement
- Use of **SQL DOM** API to encapsulate database queries to systematically apply coding best practices, e.g. input filtering and rigorous type checking of user input

Detection

- **Signature** based: attempts to match specific attack patterns, may not work against self-modifying attacks
- **Anomaly** based: attempts to define normal behavior (training phase) and then detect behavior patterns outside the normal range (detection phase)
- **Code analysis**: use of a test suite to detect SQLi vulnerabilities by generating a wide range of SQLi attacks and assessing the system response

Run-time prevention

- Check queries at runtime to see if they conform to a model of expected queries
- Various automated tools are available for this purpose

Index

- Database Management Systems
- The Need for Database Security
- Relational Databases
- SQL Injection Attacks
- Database Access Control
- Inference
- Database Encryption
- Data Center Security

Database Access Control

Database access control system determines:

If the user has access to the **entire** database or just **portions** of it

What **access rights** the user has (create, insert, delete, update, read, write, …)

Granularity of access rights ranges from the entire database to selected rows or columns within a table

Access rights **can depend on the contents** of a table entry

Can support a range of administrative policies

Centralized administration

- Small number of privileged users may grant and revoke access rights

Ownership-based administration

- The owner/creator of a table may grant and revoke access rights to the table

Decentralized administration

- The owner/creator of the table may grant and revoke authorization rights to other users, allowing them to grant and revoke access rights to the table

SQL Access Controls

- SQL provides two commands for managing access rights
  - `GRANT`
    - Used to **grant** one or more access rights or to assign a user to a role
    - Enables an access right to **cascade** through a number of users
  - `REVOKE`
    - Used to **revoke** access rights

- Typical access rights are:
  - `SELECT`: grantee may read entire database, individual tables, or specific columns in a table
  - `INSERT`: grantee may insert rows in a table, or insert rows with values for specific columns in a table
  - `UPDATE`: semantics is similar to `INSERT`
  - `DELETE`: grantee may delete rows from a table
  - `REFERENCES`: grantee is allowed to define foreign keys in another table that refer to the specified columns

SQL Access Controls Examples

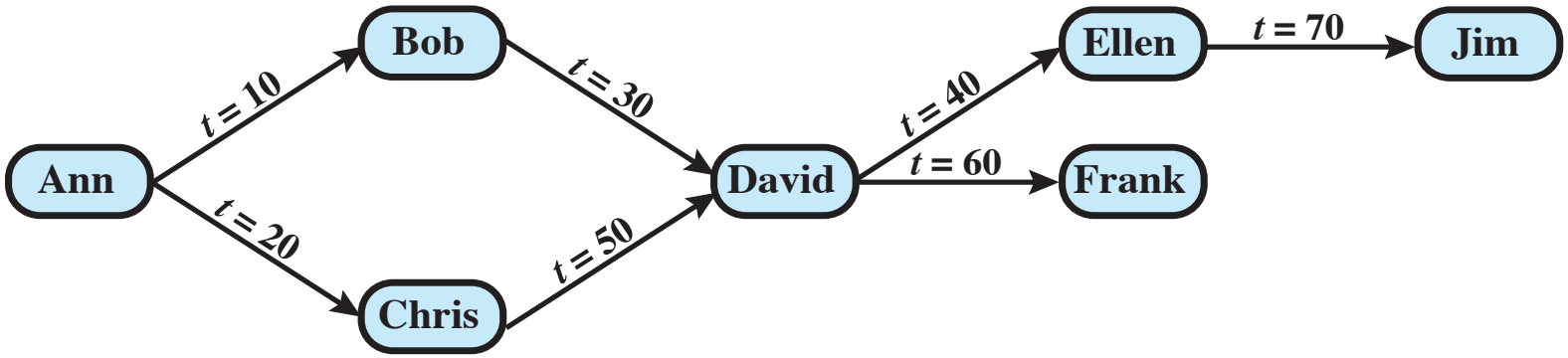
```
GRANT SELECT ON ANY TABLE TO ricflair
```

This statement enables user `ricflair` to query any table in the database

```
REVOKE SELECT ON ANY TABLE FROM ricflair
```

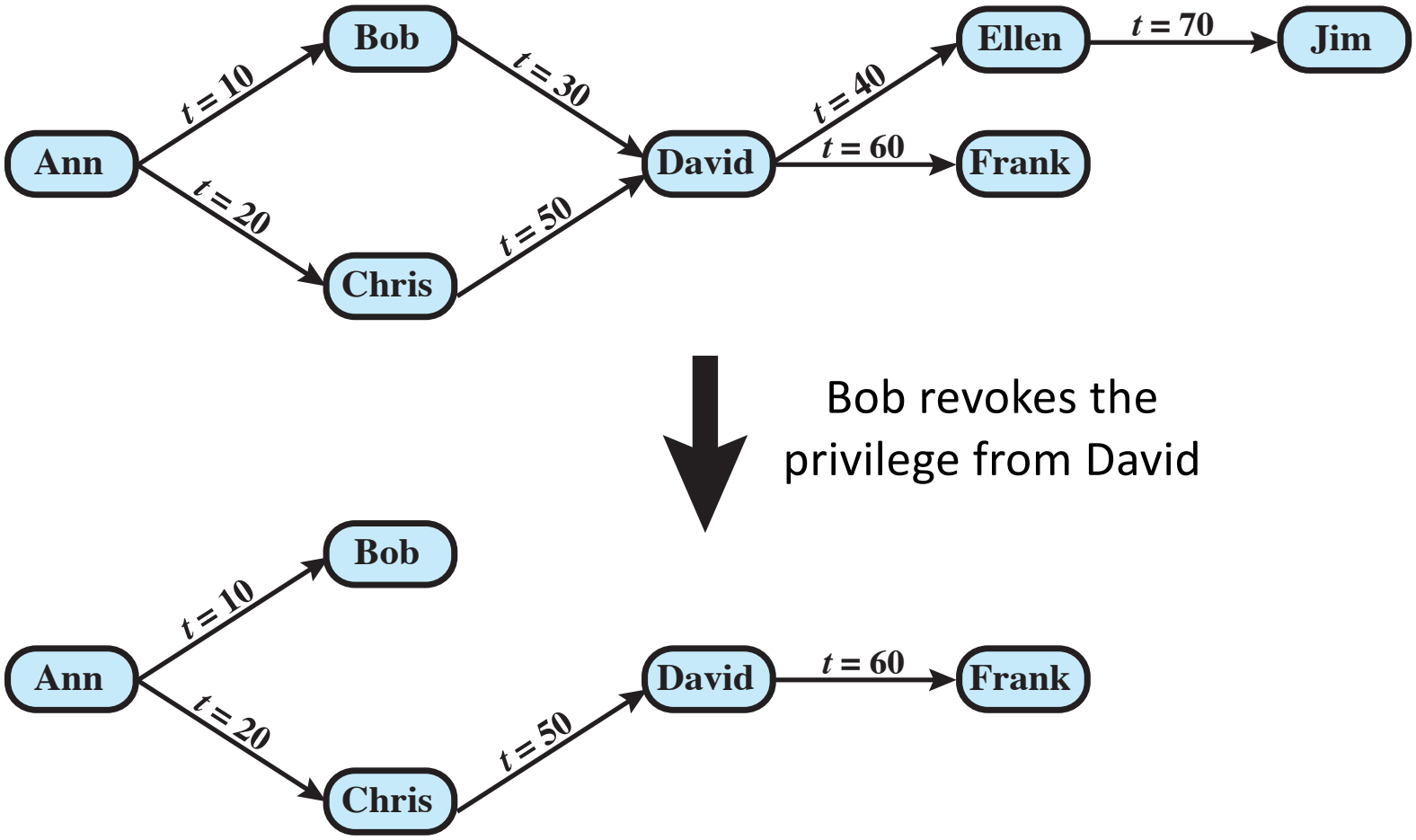
This statement revokes the access rights of the preceding example

Cascading Authorizations



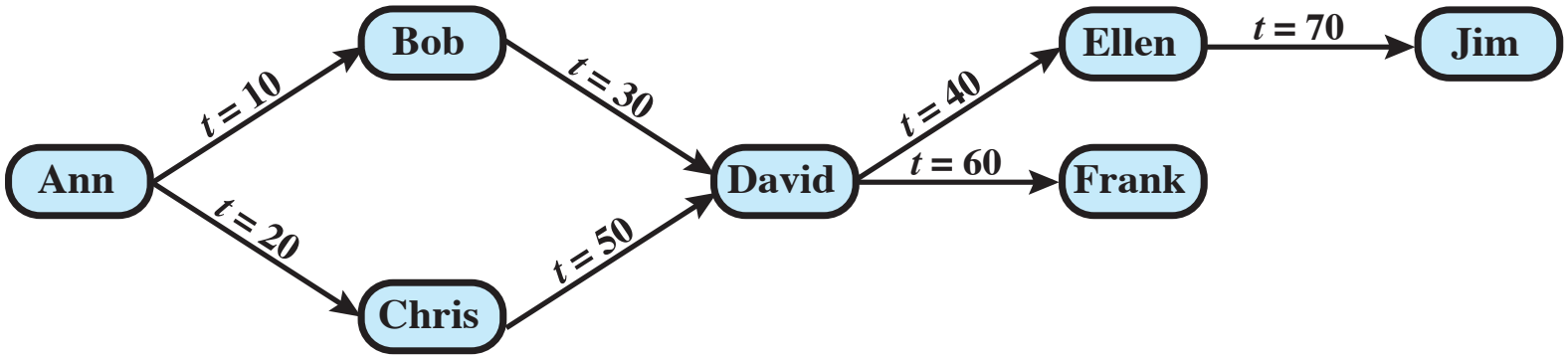
- Ann grants the access right to Bob at time *t = 10* and to Chris at time t = 20
- Assume that *the grant option is always used*
- Thus, Bob is able to grant the access right to David at t = 30
- Chris redundantly grants the access right to David at t = 50
- Meanwhile, David grants the right to Ellen, who in turn grants it to Jim; and subsequently David grants the right to Frank

Cascading Authorizations:
Bob revokes a specific privilege from David

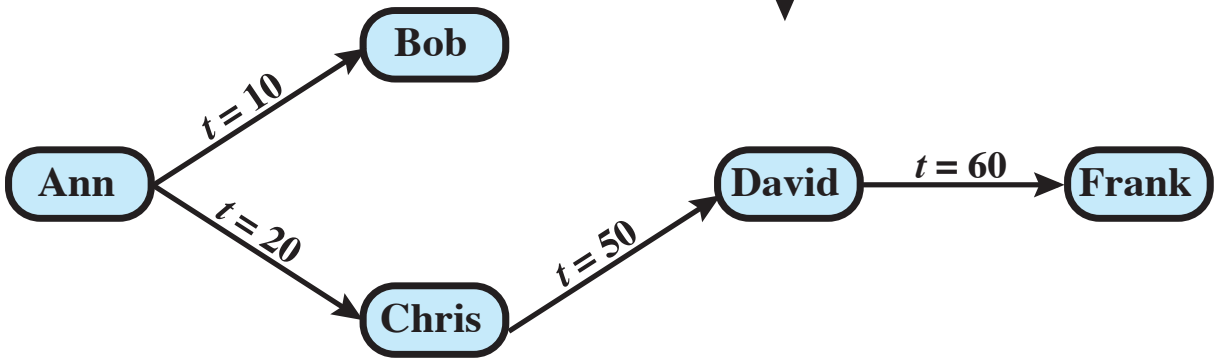


Convention followed by most implementations: When user A revokes an access right, any cascaded access right is also revoked, unless that access right would exist even if the original grant from A had never occurred

Cascading Authorizations:
Bob revokes a specific privilege from David



Bob revokes the privilege from David



- David still has the access right because it was granted by Chris at *t = 50*
- However, David granted the access right to Ellen after receiving the right, with grant option, from Bob but prior to receiving it from Chris
- Hence, the access right to Ellen, and therefore Jim, is revoked

Role-Based Access Control

Categories of typical database users

Application owner	End user	Administrator
• An end user who owns database objects as part of an application	• An end user who operates on database objects via a particular application but does not own any of the database objects	• User who has administrative responsibility for part or all of the database

- An **application** has associated with it a number of **tasks**, with each task requiring specific access rights to portions of the database
  - For each **task**, one or more roles can be defined that specify the needed access rights
- The **application owner** may assign roles to **end users**
- **Administrators** are responsible for more sensitive or general roles, including those having to do with managing physical and logical database components, such as data files, users, and security mechanisms
  - **Administrators** in turn can assign **users** to administrative-related roles
- The **system** needs to be set up to give certain administrators certain privileges

Role-Based Access Control

- RBAC eases administrative burden and improves security
  - A database system often supports dozens of applications
  - An individual user may use a variety of applications to perform a variety of tasks, each of which requires its own set of privileges
  - RBAC provides a means of granting users only the access rights they need for each task they perform
- A database RBAC **facility** must provide at least the following capabilities:
  - Create and delete roles
  - Define permissions for a role
  - Assign and cancel assignment of users to roles
- A good example of the use of roles in database security is the RBAC facility provided by *Microsoft SQL Server*

Microsoft SQL Server

- Microsoft SQL Server supports **three types of roles**: server roles, database roles, and user-defined roles
  - The first two types of roles are fixed, preconfigured with specific access rights
  - The administrator or user cannot add, delete, or modify fixed roles; it is only possible to add and remove users as members of a fixed role
- **Server roles** are defined at the server level and exist independently of any user database
  - They are designed to ease the administrative task
  - Database administrators can use these roles to assign different administrative tasks to personnel and give them only the rights they absolutely need
- **Database roles** operate at the level of an individual database
- **User-defined roles** can be assigned access rights to portions of the database
  - A user with proper authorization (e.g. with `db_securityadmin` role) may define a new role and associate access rights with the role

Fixed Roles in Microsoft SQL Server

Role	Permissions
Fixed Server Roles	
sysadmin	Can perform any activity in SQL Server and have complete control over all database functions
serveradmin	Can set server-wide configuration options, shut down the server
setupadmin	Can manage linked servers and startup procedures
securityadmin	Can manage logins and CREATE DATABASE permissions, also read error logs and change passwords
processadmin	Can manage processes running in SQL Server
dbcreator	Can create, alter, and drop databases
diskadmin	Can manage disk files
bulkadmin	Can execute BULK INSERT statements
Fixed Database Roles	
db_owner	Has all permissions in the database
db_accessadmin	Can add or remove user IDs
db_datareader	Can select all data from any user table in the database
db_datawriter	Can modify any data in any user table in the database
db_ddladmin	Can issue all Data Definition Language (DDL) statements
db_securityadmin	Can manage all permissions, object ownerships, roles and role memberships
db_backupoperator	Can issue DBCC, CHECKPOINT, and BACKUP statements
db_denydatareader	Can deny permission to select data in the database
db_denydatawriter	Can deny permission to change data in the database

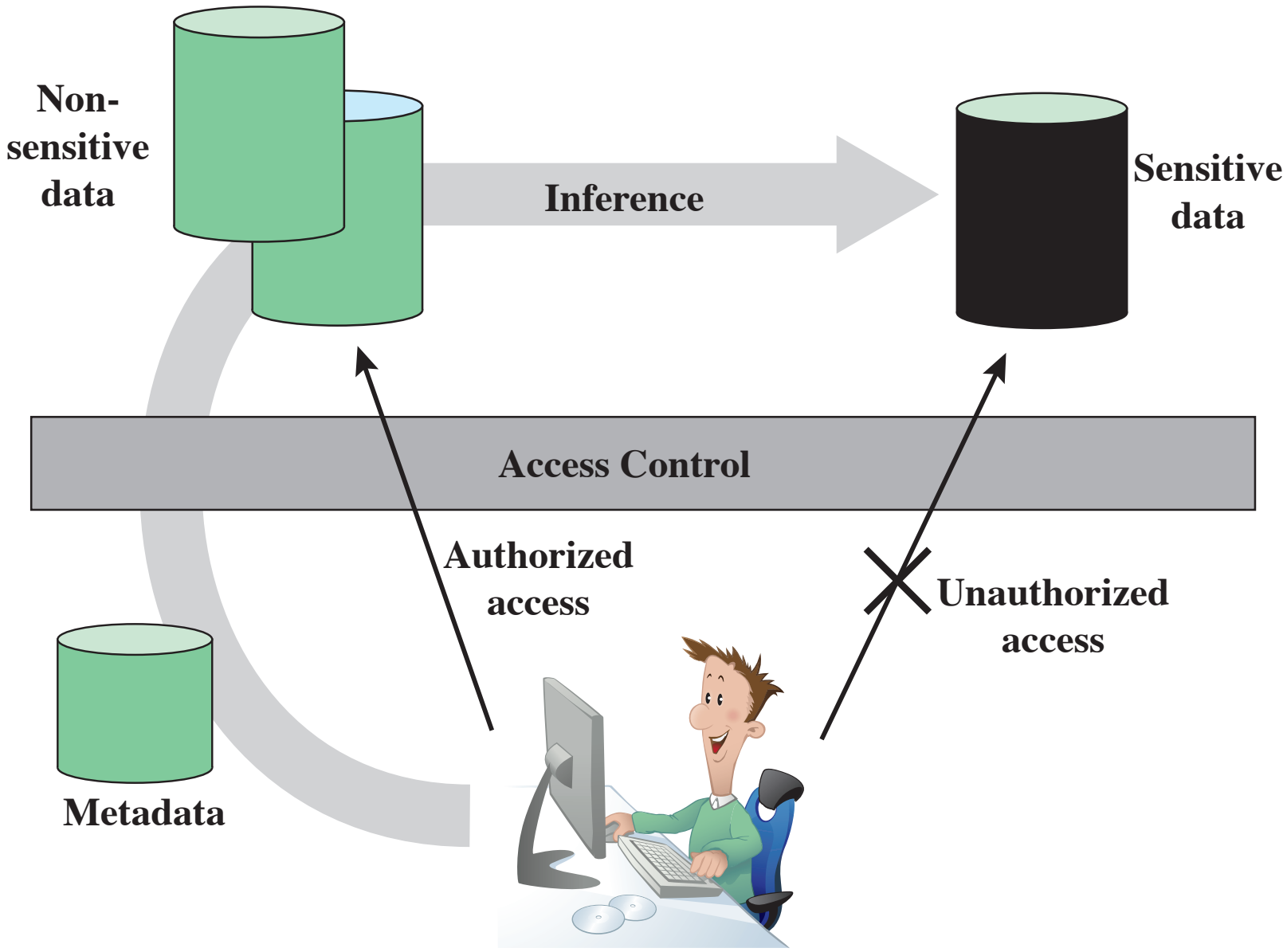
Index

- Database Management Systems
- The Need for Database Security
- Relational Databases
- SQL Injection Attacks
- Database Access Control
- Inference
- Database Encryption
- Data Center Security

Inference

- It is the process of **performing authorized queries** and **deducing unauthorized information** from the legitimate responses received
- The inference problem arises when the **combination** of a number of data items
  - is **more sensitive** than the individual items, or
  - can be used to infer data of a **higher** sensitivity
- The attacker may use **nonsensitive data** as well as **metadata** (i.e. knowledge about correlations or dependencies among data items)
- The information transfer path by which unauthorized data is obtained is referred to as an inference channel

Indirect Information Access via Inference Channel



Inference techniques

Two inference techniques can be used to derive additional information:

- **analyzing functional dependencies** between attributes within a table or across tables
- **merging views** with the same constraints

Inference Example: merging views with the same constraints

- Table (a) is an Inventory table

Item	Availability	Cost ($)	Department
Shelf support	in-store/online	7.99	hardware
Lid support	online only	5.49	hardware
Decorative chain	in-store/online	104.99	hardware
Cake pan	online only	12.99	housewares
Shower/tub cleaner	in-store/online	11.99	housewares
Rolling pin	in-store/online	10.99	housewares

(a) Inventory table

Inference Example: merging views with the same constraints

- The two views (b) can be created by means of the following SQL commands:

```
CREATE view V1 AS SELECT
Availability, Cost
FROM Inventory
WHERE Department =
”hardware”

CREATE view V2 AS SELECT
Item, Department
FROM Inventory
WHERE Department =
”hardware”
```

Item	Availability	Cost ($)	Department
Shelf support	in-store/online	7.99	hardware
Lid support	online only	5.49	hardware
Decorative chain	in-store/online	104.99	hardware
Cake pan	online only	12.99	housewares
Shower/tub cleaner	in-store/online	11.99	housewares
Rolling pin	in-store/online	10.99	housewares

(a) Inventory table

Availability	Cost ($)
in-store/online	7.99
online only	5.49
in-store/online	104.99

Item	Department
Shelf support	hardware
Lid support	hardware
Decorative chain	hardware

(b) Two views

Inference Example: merging views with the same constraints

- Suppose users of views (b) are **not authorized** to access the relationship between Item & Cost
- A user who has access to either or both views (b) cannot infer the relationship by functional dependencies
- However, a user who knows the structure of the Inventory table and who knows that the view tables maintain the same row order as the Inventory table is then able to **merge the two views** to construct the table (c)
- This violates the access policy that the relationship of Item and Cost must not be disclosed

Item	Availability	Cost ($)	Department
Shelf support	in-store/online	7.99	hardware
Lid support	online only	5.49	hardware
Decorative chain	in-store/online	104.99	hardware
Cake pan	online only	12.99	housewares
Shower/tub cleaner	in-store/online	11.99	housewares
Rolling pin	in-store/online	10.99	housewares

(a) Inventory table

Availability	Cost ($)
in-store/online	7.99
online only	5.49
in-store/online	104.99

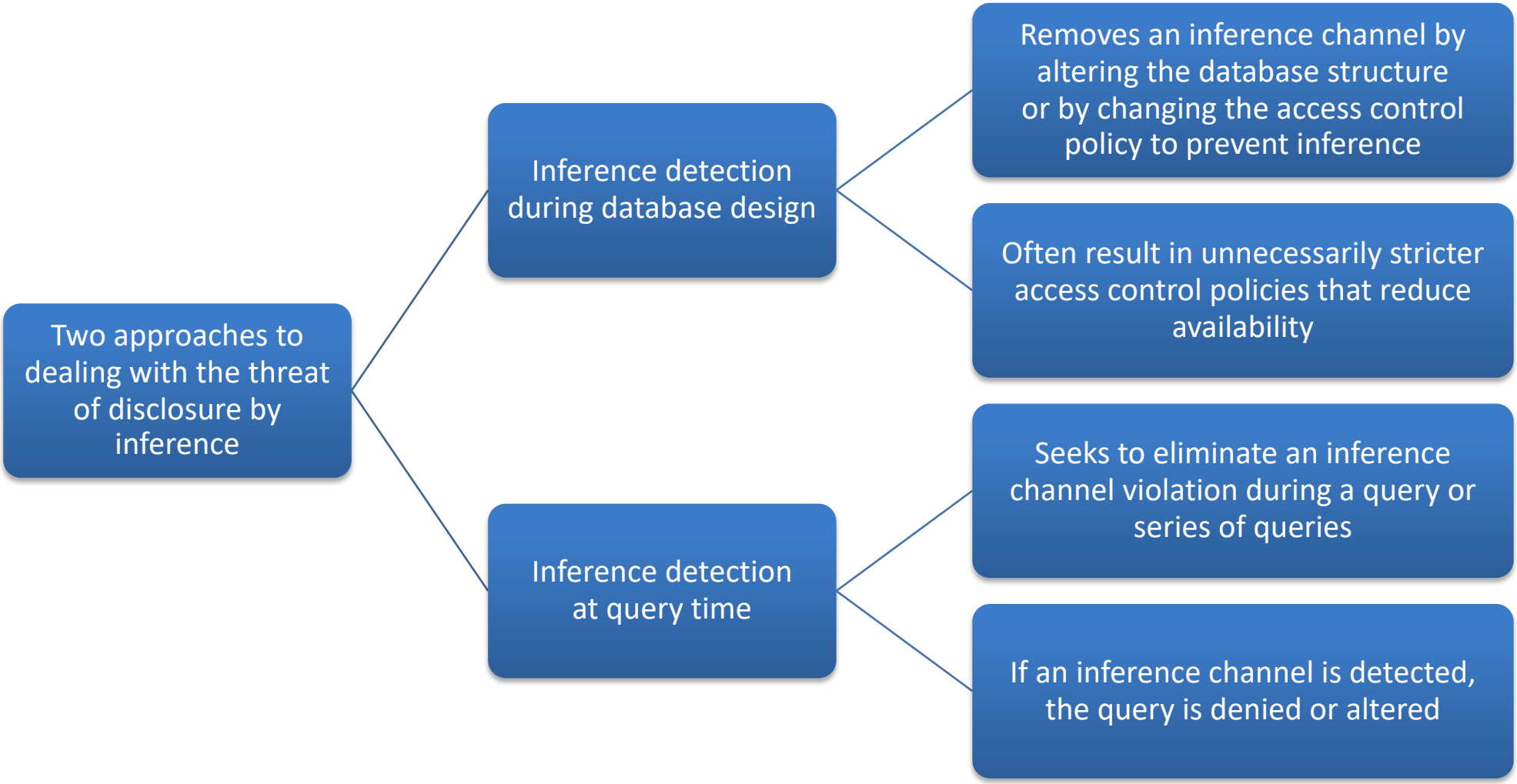
Item	Department
Shelf support	hardware
Lid support	hardware
Decorative chain	hardware

(b) Two views

Item	Availability	Cost ($)	Department
Shelf support	in-store/online	7.99	hardware
Lid support	online only	5.49	hardware
Decorative chain	in-store/online	104.99	hardware

(c) Table derived from combining query answers

Approaches to Inference Detection



- This is a difficult problem and the subject of ongoing research
- To give some appreciation of the difficulty, we present an example

Another Example of Disclosure by Inference

- Consider a database containing personnel information, including names, addresses, and salaries of employees
- Individually, the name, address, and salary information is available to a subordinate role, such as *Clerk*, but the association of names and salaries is restricted to a superior role, such as *Administrator*

Another Example of Disclosure by Inference

- Consider a database containing personnel information, including names, addresses, and salaries of employees
- Individually, the name, address, and salary information is available to a subordinate role, such as *Clerk*, but the association of names and salaries is restricted to a superior role, such as *Administrator*
- One solution to this problem is to construct three tables, which include the following information:

```
Employees (Emp#, Name, Address)
Salaries (S#, Salary)
Emp-Salary (Emp#, S#)
```

where each line consists of the table name followed by a list of column names for that table

- Each employee is assigned a unique employee number (`Emp#`) and a unique salary number (`S#`)
- Tables `Employees` and `Salaries` are accessible to the *Clerk* role, but the `Emp-Salary` table is only accessible to the *Administrator* role
- The sensitive relationship between employees and salaries is protected from users assigned the *Clerk* role

Another Example of Disclosure by Inference

- Suppose that we want to add a new, not sensitive attribute, *employee start date*
- This could be added to the `Salaries` table as follows:

```
Employees (Emp#, Name, Address)
Salaries (S#, Salary, Start-Date)
Emp-Salary (Emp#, S#)
```

- However, an employee's start date is an easily observable or discoverable attribute of an employee
- Thus a user in the *Clerk* role could be able to infer (or partially infer) the employee's name
- This would compromise the relationship between employee and salary
- A straightforward way to remove the inference channel is to add the `Start-Date` column to the `Employees` table rather than to the `Salaries` table

Inference Detection approaches: Considerations

- The *first security problem* (it is possible to infer the relationship between employee and salary) can be detected through analysis of the data structures and security constraints that are available to the DBMS
- The *second security problem* (it is possible to infer the relationship between employee and start-date) cannot be detected using only the information stored in the database (since the database does not indicate that the employee name can be inferred from the start date)
- Inference detection **algorithms** are needed for both approaches to inference detection
  - Progress has been made in devising specific inference detection techniques for **multilevel secure databases** (where data have multiple levels of classification) and **statistical databases** (which provide summary or aggregate information)

Index

- Database Management Systems
- The Need for Database Security
- Relational Databases
- SQL Injection Attacks
- Database Access Control
- Inference
- Database Encryption
- Data Center Security

Database Encryption

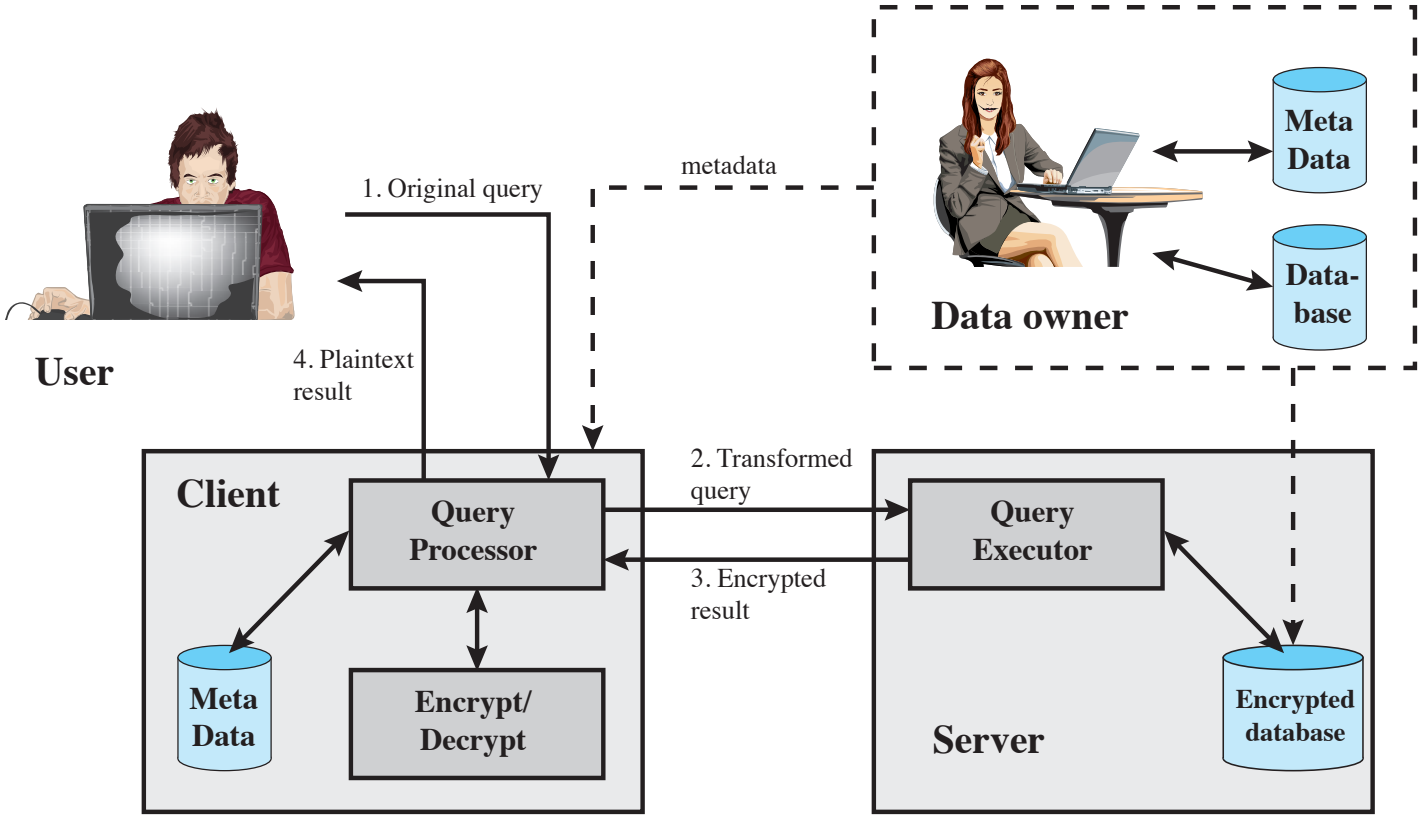
- The database is typically the most valuable information resource for any organization
  - Must be protected by multiple layers of security: firewalls, authentication mechanisms, general access control systems, DB access control systems, database encryption
- Encryption becomes the **last line of defense** in database security
  - Can be applied to the entire database, at the record level, the attribute level, or at the level of individual entries
- *Disadvantages* to encryption:
  - **Key management**: authorized users must have access to the decryption key for the data for which they have access
    - Because a database is typically accessible to a wide range of users and applications, providing secure keys to selected parts of the database to authorized users and applications is a complex task
  - **Inflexibility**: when part or all of the database is encrypted it becomes more difficult to perform record searching
    - The user would have to download entire tables from the database, decrypt the tables, and work with the results

Database Encryption

- A DBMS is a **complex** collection of hardware and software
  - It requires a large storage capacity and skilled personnel
- An often used **solution** is to outsource the DBMS and the database to a service provider
  - The **service provider** maintains the database and provide high availability, disaster prevention, and efficient access and update
- The main concern with such a solution is data confidentiality
  - A straightforward solution to the security problem is to **encrypt the entire database** and not provide the encryption/decryption keys to the service provider
  - However, to improve **flexibility**, it must be possible to work with the database in its encrypted form

A Database Encryption Scheme: simplest arrangement

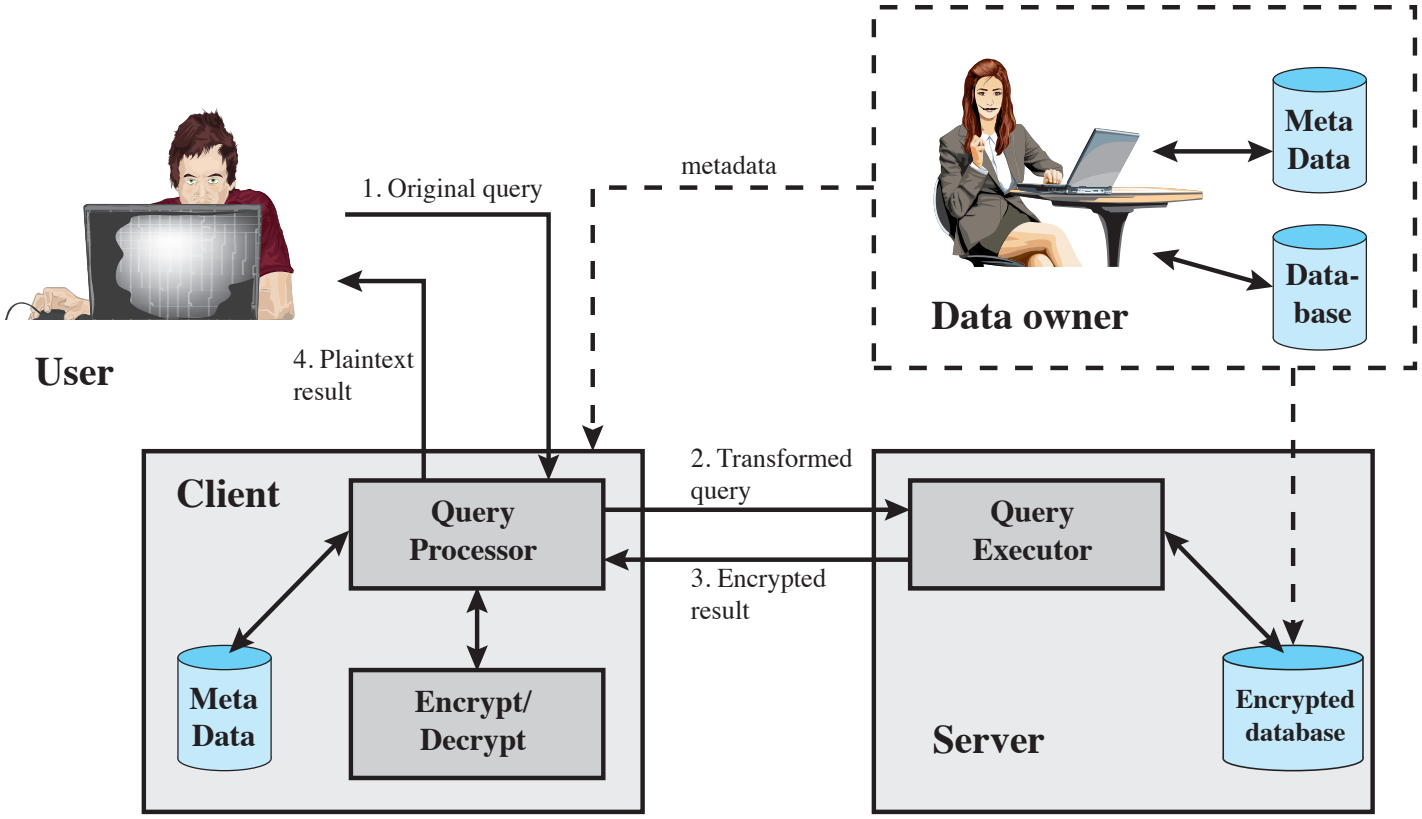
- **Data owner**: organization that produces data whose access has to be controlled
- **User**: human entity that presents queries to the system
- **Client**: front end that *transforms user queries into queries on the encrypted data* stored on the server
- **Server**: organization that receives the encrypted data from a data owner and makes them available for distribution to clients



- Suppose that each individual item in the database is **encrypted separately**, all using the same encryption key
- The encrypted database is stored at the server, but the server does **not** have the key, so that the data are secure at the server
- Even if someone were able to hack into the server's system, all he or she would have access to is encrypted data
- The client system does **have** a copy of the encryption key

A Database Encryption Scheme: simplest arrangement

- **Data owner**: organization that produces data whose access has to be controlled
- **User**: human entity that presents queries to the system
- **Client**: front end that transforms user queries into queries on the encrypted data stored on the server
- **Server**: organization that receives the encrypted data from a data owner and makes them available for distribution to clients

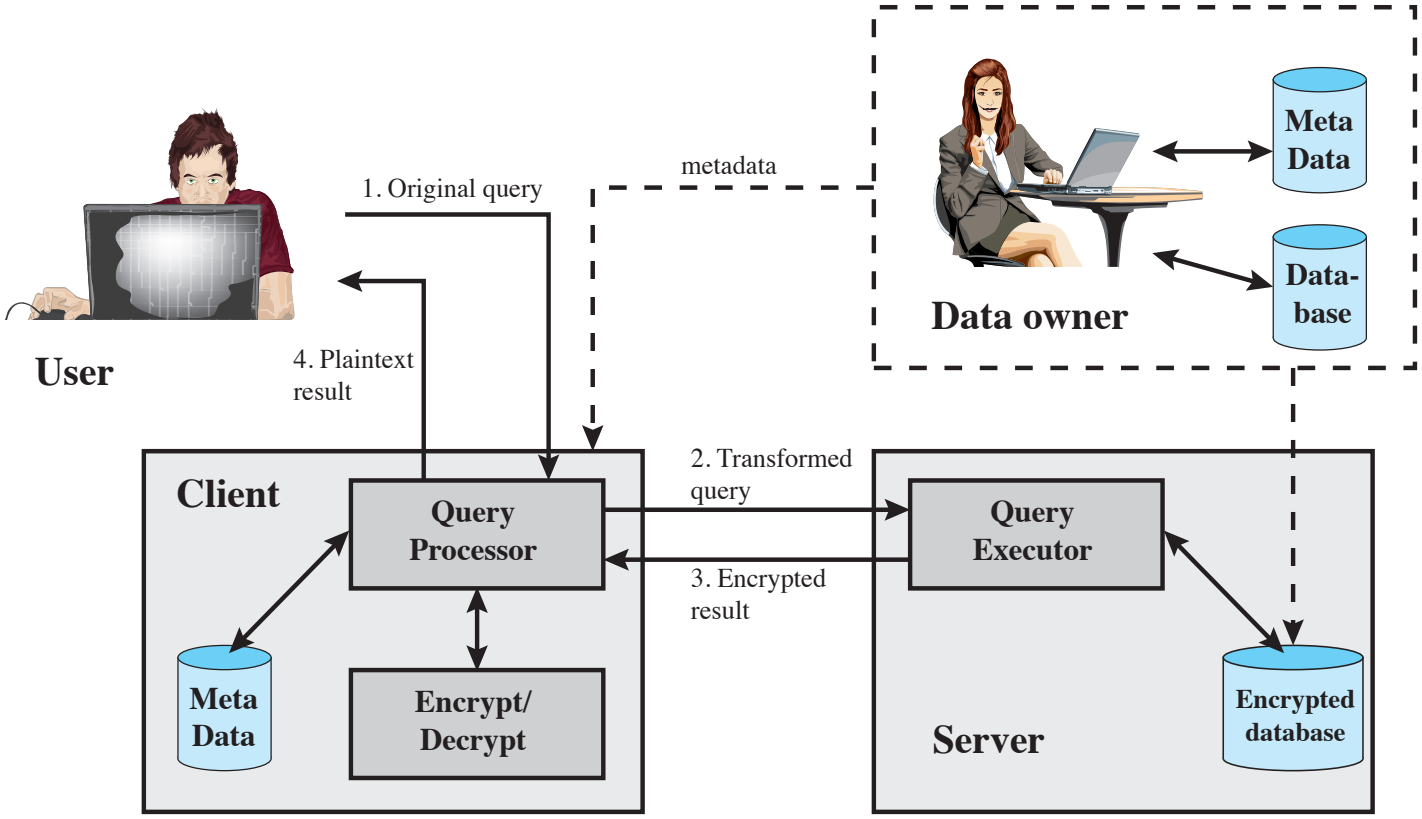


Steps for a user to **retrieve a record** from the database:

1. The user issues an SQL query for fields from one or more records with a specific value of the primary key
2. The query processor at the client encrypts the primary key, modifies the SQL query accordingly, and transmits the query to the server
3. The server processes the query using the encrypted value of the primary key and returns the appropriate record or records to the client
4. The query processor decrypts the data and returns the results

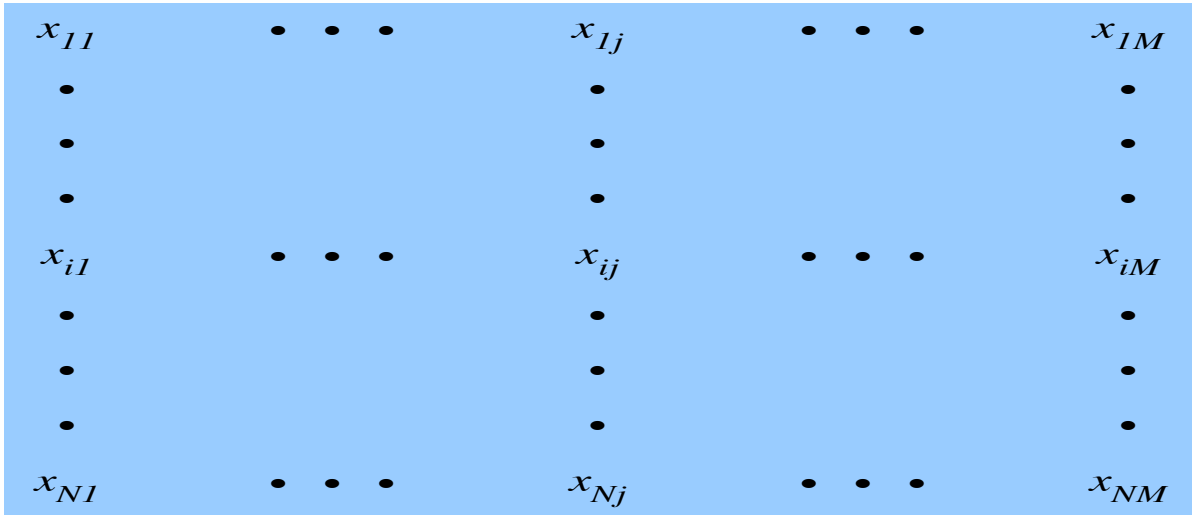
A Database Encryption Scheme: simplest arrangement

- **Data owner**: organization that produces data whose access has to be controlled
- **User**: human entity that presents queries to the system
- **Client**: front end that transforms user queries into queries on the encrypted data stored on the server
- **Server**: organization that receives the encrypted data from a data owner and makes them available for distribution to clients



- This method is certainly straightforward but **lacks flexibility**
- For example, suppose there is an `Employee` table containing a `salary` attribute and the user wishes to retrieve all records for salaries less than $70K
- There is no obvious way to do this, because the attribute value for salary in each record is encrypted and encrypted values do not preserve the ordering of values in the original attribute

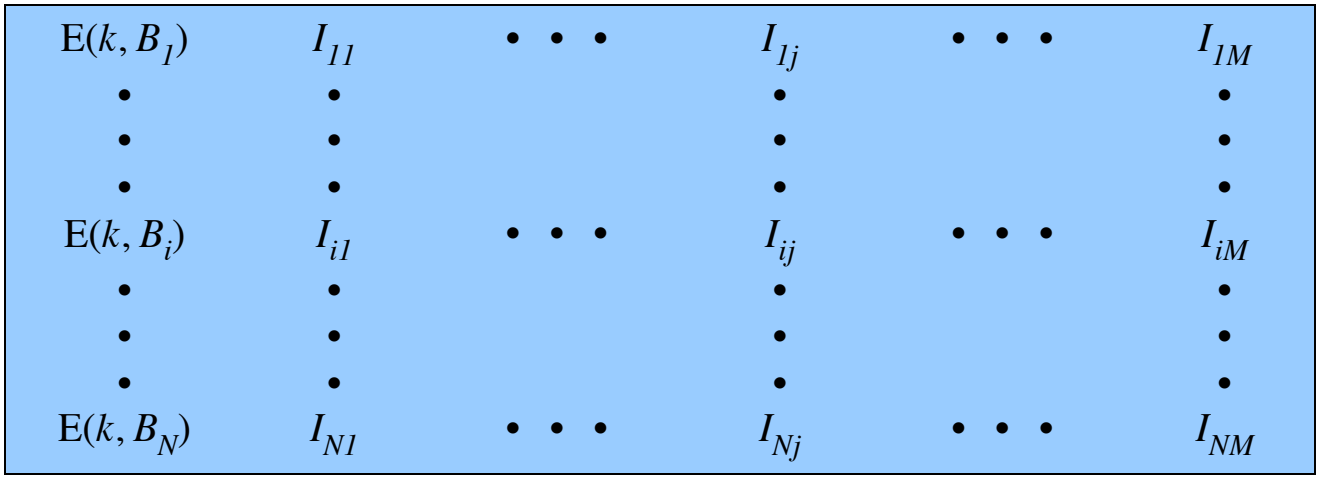
A more flexible Database Encryption Scheme



- Each record (row) of a table in the database is **encrypted as a block**
  - Each row R_i is treated as a contiguous block $B_i = (x_{i1}||x_{i2}||...||x_{iM})$
  - Each attribute value in R_i, regardless of whether it is text or numeric, is treated as a sequence of bits, and all of the attribute values for that row are concatenated together to form a single binary block
  - The entire row is encrypted, expressed as $E(k, B_i) = E(k, (x_{i1}||x_{i2}||...||x_{iM}))$
- To assist in data retrieval, **attribute indexes** are associated with each table
  - For some or all of the attributes an index value is created
  - For any attribute, the range of all possible attribute values is partitioned and an index value is assigned to each partition element
- For each row in the original database, there is one row in the encrypted database

$$(x_{i1}, x_{i2}, \ldots, x_{iM}) \rightarrow [E(k, B_i), I_{i1}, I_{i2}, \ldots, I_{iM}]$$

A more flexible Database Encryption Scheme



- Each record (row) of a table in the database is **encrypted as a block**
  - Each row R_i is treated as a contiguous block $B_i = (x_{i1}||x_{i2}||...||x_{iM})$
  - Each attribute value in R_i, regardless of whether it is text or numeric, is treated as a sequence of bits, and all of the attribute values for that row are concatenated together to form a single binary block
  - The entire row is encrypted, expressed as $E(k, B_i) = E(k, (x_{i1}||x_{i2}||...||x_{iM}))$
- To assist in data retrieval, **attribute indexes** are associated with each table
  - For some or all of the attributes an index value is created
  - For any attribute, the range of all possible attribute values is partitioned and an index value is assigned to each partition element
- For each row in the original database, there is one row in the encrypted database

$$(x_{i1}, x_{i2}, \ldots, x_{iM}) \rightarrow [E(k, B_i), I_{i1}, I_{i2}, \ldots, I_{iM}]$$

A more flexible Database Encryption Scheme

E.g. eid values lie in the range [1, 1000].
We divide these values into five partitions: [1, 200], [201, 400], [401, 600], [601, 800], and [801, 1000]
and assign index values 1, 2, 3, 4, and 5, resp.

Employee Table

eid	ename	salary	addr	did
23	Tom	70K	Maple	45
860	Mary	60K	Main	83
320	John	50K	River	50
875	Jerry	55K	Hopewell	92

E.g. for *ename*, assign index 1 to values starting with A or B, index 2 to values starting with C or D, and so on

The mapping functions between attribute values and index values constitute **metadata**, stored at the client and data owner locations but not at the server

E(*k*, *B*)	I(eid)	I(ename)	I(salary)	I(addr)	I(did)
1100110011001011…	1	10	3	7	4
0111000111001010…	5	7	2	7	8
1100010010001101…	2	5	1	9	5
0011010011111101…	5	5	2	4	9

Encrypted Employee Table with Indexes

A more flexible Database Encryption Scheme

Provides for **more efficient data retrieval**, e.g.

- A user requests records for all employees with *eid* < 300
- The query processor requests all records with I(*eid*) ≤ 2
- These are returned by the server
- The query processor decrypts all rows returned, discards those that do not match the original query, and returns the requested unencrypted data to the user

A more flexible Database Encryption Scheme

Provides for **more efficient data retrieval**, e.g.

- A user requests records for all employees with *eid* < 300
- The query processor requests all records with I(*eid*) ≤ 2
- These are returned by the server
- The query processor decrypts all rows returned, discards those that do not match the original query, and returns the requested unencrypted data to the user

Provides **a certain amount of information to an attacker**, namely a rough relative ordering of rows by a given attribute

- To obscure such information, **the ordering of indexes can be randomized**
  - E.g. the *eid* values could be partitioned by mapping [1, 200], [201, 400], [401, 600], [601, 800], and [801, 1000] into 2, 3, 5, 1, and 4, respectively
- Because the metadata are not stored at the server, an attacker could not gain this information from the server

A more flexible Database Encryption Scheme

Provides for **more efficient data retrieval**, e.g.

- A user requests records for all employees with *eid* < 300
- The query processor requests all records with I(*eid*) ≤ 2
- These are returned by the server
- The query processor decrypts all rows returned, discards those that do not match the original query, and returns the requested unencrypted data to the user

Provides **a certain amount of information to an attacker**, namely a rough relative ordering of rows by a given attribute

- To obscure such information, **the ordering of indexes can be randomized**
  - E.g. the *eid* values could be partitioned by mapping [1, 200], [201, 400], [401, 600], [601, 800], and [801, 1000] into 2, 3, 5, 1, and 4, respectively
- Because the metadata are not stored at the server, an attacker could not gain this information from the server

Other features may be added

- To increase the **efficiency** of accessing records by means of the primary key, the system could use the encrypted value of the primary key attribute values, or a hash value
  - The row corresponding to the primary key value could be retrieved individually
- Different portions of the database could be encrypted with **different** keys, so that users would only have access to that portion for which they had the decryption key
  - This latter scheme could be incorporated into a RBAC system

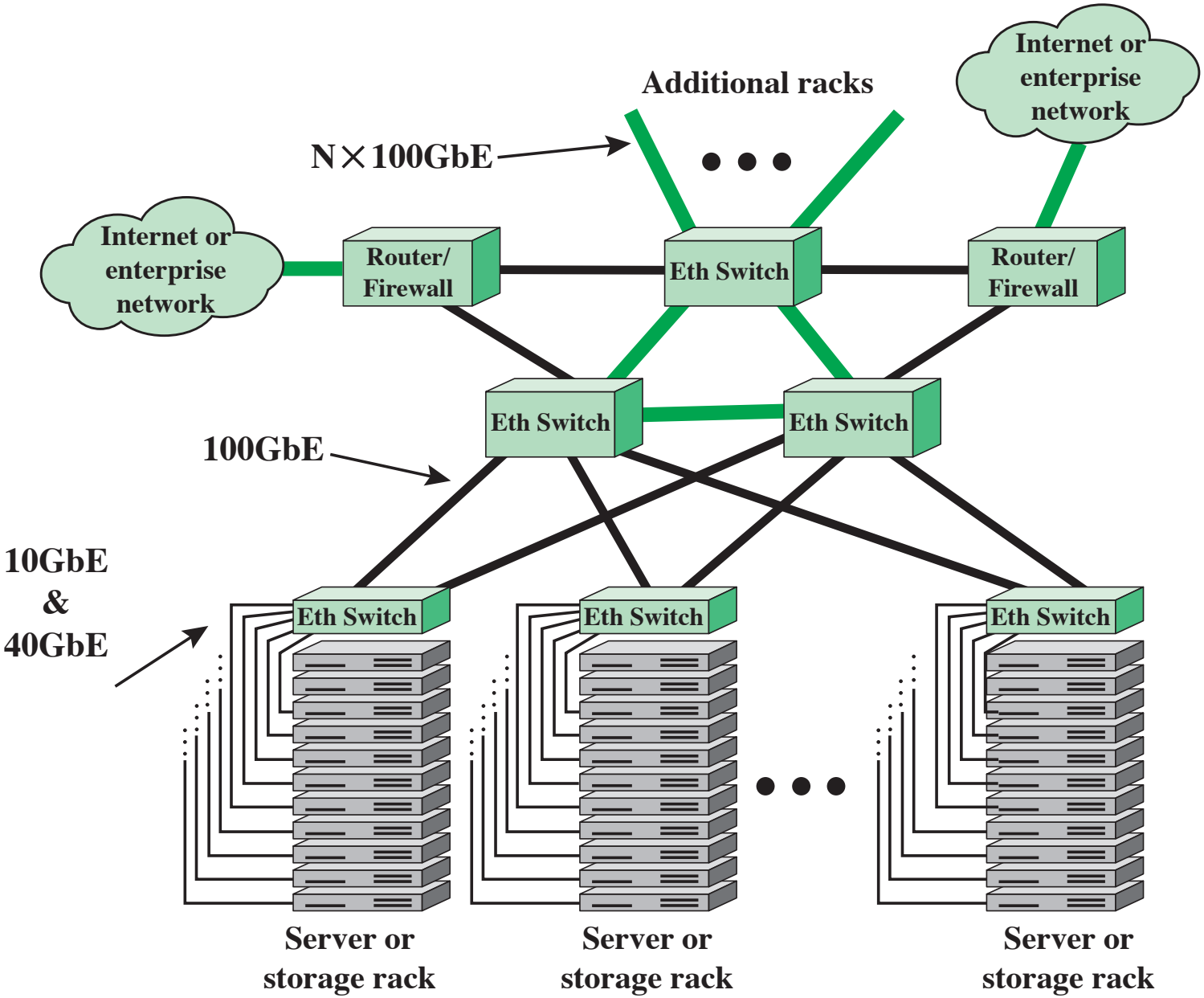
Index

- Database Management Systems
- The Need for Database Security
- Relational Databases
- SQL Injection Attacks
- Database Access Control
- Inference
- Database Encryption
- Data Center Security

Data Center Security

- Data center
  - An enterprise facility that houses a large number of servers, storage devices, and network switches and equipment
  - The number of servers and storage devices can run into the tens of thousands in one facility
  - Generally includes redundant or backup power supplies, redundant network connections, environmental controls (e.g., air conditioning and fire suppression), and various security devices
  - Can occupy one room of a building, one or more floors, or an entire building
  - Large data centers are industrial-scale operations using as much electricity as a small town
- *Examples of uses* include
  - Cloud service providers
  - Search engines
  - Large scientific research facilities
  - IT facilities for large enterprises

Key Data Center Elements



Key Data Center Elements

- Most of the equipment in a large data center is in the form of **stacks** of **servers** and **storage modules** mounted in open racks or closed cabinets, which are usually placed in single rows forming corridors between them
  - This allows access to the front and rear of each rack or cabinet
- Each **rack** has one or two 10, 40 or 100-Gbps **Ethernet switches** to inter-connect all the servers and provide connectivity to the rest of the facility
  - The *individual modules* are equipped with 10-Gbps or 40-Gbps Ethernet ports to handle the massive traffic to and from these servers
  - The switches are often mounted in the rack and referred to as *top-of-rack (ToR) switches*
- Very large data centers, such as **cloud providers**, require switches operating at 100-Gbps to support the interconnection of server racks and to provide adequate capacity for connecting off-site through network interface controllers (NICs) on routers or firewalls

Data Center: Security Considerations

- The data center houses massive amounts of data that are:
  - located in a confined physical space
  - interconnected with direct-connect cabling
  - accessible through external network connections, so once past the boundary, a threat is posed to the entire complex
  - typically representative of the greatest single asset of the enterprise
- Thus, data center security is a **top priority** for any enterprise with a large data center
  - All of the security threats and countermeasures we study are relevant
- Some of the important **threats** to consider include the following:
  - Denial of service
  - Advanced persistent threats from targeted attacks
  - Privacy breaches
  - Application exploits such as SQL injection
  - Malware
  - Physical security threats

Data Center Security four-layer Model

Important aspects of data center security can be represented as a four-layer model

- **Site security** refers primarily to the *physical security of the entire site* including the building that houses the data center, as well as the use of redundant utilities, buffer zones, crash barriers, entry points, …
- **Physical security** *of the data center* itself includes barriers to entry, such as a mantrap (a double-door single-person access control space) coupled with two/three factor authentication techniques for gaining physical access, security personnel, surveillance systems, …
- **Network security** is extremely important in a facility in which such a large collection of assets are concentrated in a single place and accessible by external network connections, it includes firewalls, anti-virus systems, intrusion prevention/detection, authentication, …
- **Data security** involves techniques like encryption, password policy, Data protection, Data masking, Data retention, …

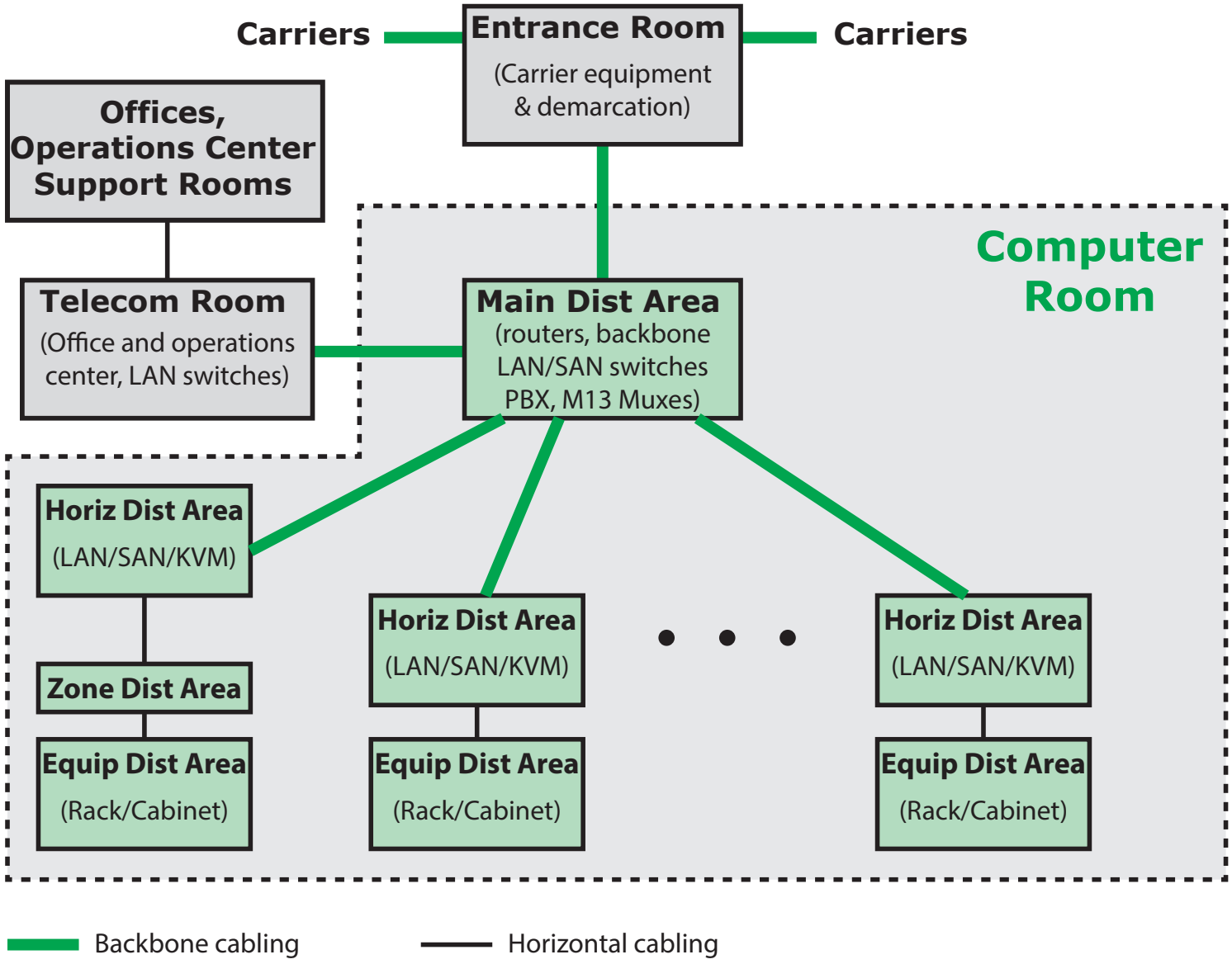
TIA-942

- The Telecommunications Industry Association (TIA) standard TIA-942 (Telecommunications Infrastructure Standard for Data Centers) specifies the **minimum requirements** for telecommunications infrastructure of data centers
- Topics covered include the following:
  - Network architecture
  - Electrical design
  - File storage, backup, and archiving
  - System redundancy
  - Network access control and security
  - Database management
  - Web hosting
  - Application hosting
  - Content distribution
  - Environmental control
  - Protection against physical hazards (fire, flood, and windstorm)
  - Power management

TIA-942

- The standard specifies **functional areas** that a TIA-942 compliant data center should include
- This helps to define **equipment placement** based on the standard hierarchical design
- This architecture anticipates growth and helps create an environment where applications and servers can be added and upgraded with **minimal downtime**
- This standardized approach supports high availability and a uniform environment for implementing security measures

Key Functional Areas of a TIA-942 Compliant Data Center



TIA-942 Functional Areas

- **Computer room:** Portion of the data center that houses data processing equipment
- **Entrance room:** One or more entrance rooms house external network access provider equipment, plus provide the interface between the computer room equipment and the enterprise cabling systems
  - Physical separation of the entrance room from the computer room provides better security
- **Main distribution area:** A centrally located area that houses the main cross-connect as well as routers and switches for LAN and SAN (storage area network) infrastructures
- **Horizontal distribution area (HDA)**: Serves as the distribution point for horizontal cabling and houses cross-connects and active equipment for distributing cable to the EDA
- **Equipment distribution area (EDA):** The location of equipment cabinets and racks, with horizontal cables terminating with patch panels
- **Zone distribution area (ZDA):** An optional interconnection point in the horizontal cabling between the HDA and EDA, which can house freestanding equipment such as mainframes

TIA-942 Tiered Reliability

- An important part of TIA-942, especially relevant for computer security
- The standard defines four tiers
- For each of the four tiers, TIA-942 describes detailed architectural, security, electrical, mechanical, and telecommunications recommendations such that the higher the tier is, the higher will be the availability

- Tier 1 – basic data center, no redundancy
- Tier 2 – redundant components
  - Single power and cooling distribution path with redundant components
- Tier 3 – concurrently maintainable
  - Multiple power and cooling distribution paths with only one active
- Tier 4 – fault tolerant
  - Multiple active power and cooling distribution paths

TIA-942 Data Center Tiers

Tier	System design	Availability /Annual Downtime
1	• Susceptible to disruptions from both planned and unplanned activity • Single path for power and cooling distribution, no redundant components • May or may not have raised floor, UPS, or generator • Takes 3 months to implement • Must be shut down completely to perform preventive maintenance	99.671%/ 28.8 hours
2	• Less susceptible to disruptions from both planned and unplanned activity • Single path for power and cooling distribution, includes redundant components • Includes raised floor, UPS, and generator • Takes 3 to 6 months to implement • Maintenance of power path and other parts of the infrastructure require a processing shutdown	99.741%/ 22.0 hours
3	• Enables planned activity without disrupting computer hardware operation but unplanned events will still cause disruption • Multiple power and cooling distribution paths but with only one path active, includes redundant components • Takes 15 to 20 months to implement • Includes raised floor and sufficient capacity and distribution to carry load on one path while performing maintenance on the other	99.982%/ 1.6 hours
4	• Planned activity does not disrupt critical load and data center can sustain at least one worst-case unplanned event with no critical load impact • Multiple active power and cooling distribution paths, includes redundant components • Takes 15 to 20 months to implement	99.995%/ 0.4 hours

Summary

- Database Management Systems
- The Need For Database Security
- Relational Databases
  - Elements of a Relational Database System
  - Structured Query Language
- SQL Injection Attacks
  - A Typical SQLi Attack
  - The Injection Technique
  - SQLi Attack Avenues and Types
  - SQLi Countermeasures
- Database Access Control
  - SQL-Based Access Definition
  - Cascading Authorizations
  - Role-Based Access Control
- Inference
- Database Encryption
- Data Center Security
  - Data center elements
  - Data center security considerations
  - TIA-942