https://brown-csci1660.github.io

# CS1660: Intro to Computer Systems Security
# Spring 2026

## Lecture 14: OS Security I

Instructor: **Nikos Triandopoulos**

March 17, 2026

BROWN

# CS1660: Announcements

- Course updates
  - Project 1, project 2, HW 1, HW2, midterm
  - Project 3 is out, due March 31 (likely to be extended)
  - Two lectures this week, then Spring break!

# Last class

- Cryptography
- Authentication
- Web security
  - The Dyn DDOS attack
  - Web security model
    - Background, web-application security, browser security, cookies
  - Cross-site references and risks
    - Recap, Javascript/iframes, CRFS attacks, CRFS defenses
    - SQL injection + Cross-site scripting (XSS) + Lecture Material Review

# Today

- Cryptography
- Authentication
- Web security
  - Cross-site references and risks
    - SQL injection + Cross-site scripting (XSS)
    - Database security + Buffer overflow attacks
  - Operating system (OS) security
    - Authorization and Access Control (AC)

# 14.1 Database security

# Database (DB)

Organized collection of structured data

- high-level data representation
  - relationships among data elements
  - semantics and logical interpretation
- set of rules for fine-grained data management
  - data retrieval and analysis
  - selective & user-specific data access

cf. unstructured/“flat”

- low-level representation
  - e.g., file
- coarse-grained
  - e.g., name, location
  - e.g., size, format

# Database management system (DBMS)

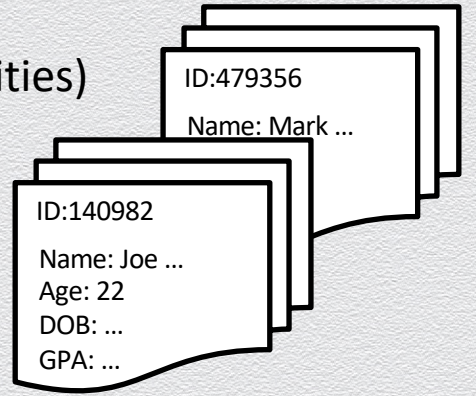
System through which users interact with a database

- provides **data-management** functions
- **data definition**
  - creation, modification and removal of data relationships and organization specs
- **update**
  - insertion, modification, and deletion of the actual data
- **retrieval**
  - derivation and presentation of information in forms directly usable by apps
- **administration**
  - definition and enforcement of rules related to reliable data management
    - e.g., user registration, performance monitoring, concurrency control, data recovery

# Relational databases

Predominant model for databases

- **collection** of records and **relations** among them
- **record/tuple**
  - one related group of data elements (representing specific entities)
  - e.g., a student, department, customer or product record
- **attributes/fields/elements**
  - elementary data items (related to entities)
  - e.g., name, ID, major, GPA, address, city, school, …
- **relations**
  - “inter-connections” of interest among records (e.g., faculty of same department)



# Table representation of relational DBs

Data is organized in tables

- entity-type tables
  - rows are individual records
  - columns are attributes of an entity
- relation-type table
  - rows are “inter-connected” records
  - columns are relevant attributes

a record or row

Table: CS-306 students

| First_Name | Last_Name | ID | … |
|---|---|---|---|
| John | Myers | 123459 | |
| Maria | Palm | 222235 | |
| Alex | Klein | 211123 | |
| ….. | …. | …. | |

Table: CS-579 students

| First_Name | Last_Name | ID | … |
|---|---|---|---|
| John | Myers | 123459 | |
| Olga | Johnson | 227800 | |
| Alex | Klein | 211123 | |
| ….. | …. | …. | |

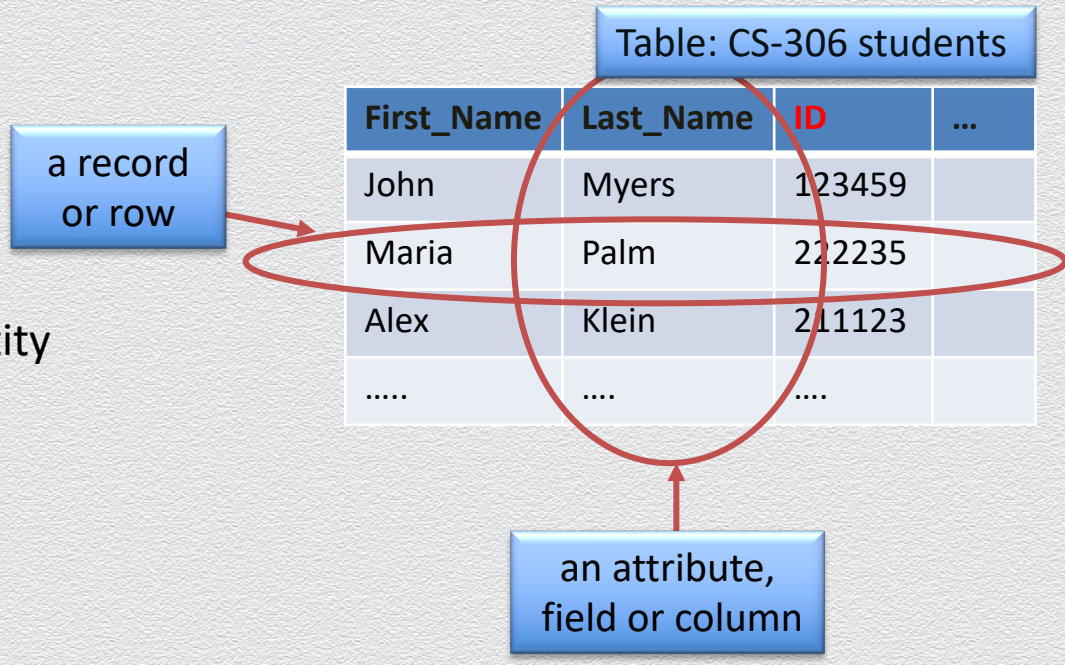
Table: CS-579 & CS-306 students

| First_Name | Last_Name | ID | … |
|---|---|---|---|
| John | Myers | 123459 | |
| Alex | Klein | 211123 | |

# Table representation of relational DBs

Data is organized in tables

- entity-type tables
  - rows are individual records
  - columns are attributes of an entity



# Table representation of relational DBs

Data is organized in tables

- entity-type tables
- relation-type table
  - rows are “inter-connected” records
  - columns are relevant attributes

Table: CS-306 students

| First_Name | Last_Name | ID | … |
|---|---|---|---|
| John | Myers | 123459 | |
| Maria | Palm | 222235 | |
| Alex | Klein | 211123 | |
| ….. | …. | …. | |

Table: CS-579 students

| First_Name | Last_Name | ID | … |
|---|---|---|---|
| John | Myers | 123459 | |
| Olga | Johnson | 227800 | |
| Alex | Klein | 211123 | |
| ….. | …. | …. | |

Table: CS-579 & CS-306 students

| First_Name | Last_Name | ID | … |
|---|---|---|---|
| John | Myers | 123459 | |
| Alex | Klein | 211123 | |
| ….. | …. | …. | |

# A entity-type table example

Table: Home_Address

| Name | First | Address | City | State | Zip | Airport |
|---|---|---|---|---|---|---|
| ADAMS | Charles | 212 Market St. | Columbus | OH | 43210 | CMH |
| ADAMS | Edward | 212 Market St. | Columbus | OH | 43210 | CMH |
| BENCHLY | Zeke | 501 Union St. | Chicago | IL | 60603 | ORD |
| CARTER | Marlene | 411 Elm St. | Columbus | OH | 43210 | CMH |
| CARTER | Beth | 411 Elm St. | Columbus | OH | 43210 | CMH |
| CARTER | Ben | 411 Elm St. | Columbus | OH | 43210 | CMH |
| CARTER | Lisabeth | 411 Elm St. | Columbus | OH | 43210 | CMH |
| CARTER | Mary | 411 Elm St. | Columbus | OH | 43210 | CMH |

# More technically…

A **relational database** is a database perceived as a collection of **tables**

- a relation $R$ is a subset of $D_1 \times \cdots \times D_n$
  - $D_1, \ldots, D_n$ are the domains on $n$ attributes
  - elements in the relation are $n$-tuples $(v_1, \ldots, v_n)$ with $v_i \in D_i$
  - the value of the $i$-th attribute has to be an element from $D_i$
  - a special null value indicates that a field does not contain any value

# Types of relations

- **Base** (or real) relations
  - named, autonomous relations comprising entity-type tables
  - exist in their own right and have ‘their own’ stored data
- **Views**
  - named, derived relations, defined in terms of other named relations
  - they **do not store** data of their own
- **Snapshots**
  - named, derived relations, defined by other named relations
  - **store** data of their own
- **Query results**
  - may or may not have a name; no persistent existence in the database per se

# Database keys

Tuples in a relation must be uniquely identifiable

- **primary keys (PKs)**
  - subset of attributes uniquely identifying records (tuples)
- every relation R must have a primary key K that is
  - **unique**: at any time, no tuples of R have the same value for K
  - **minimal**: no component of K can be omitted without destroying uniqueness
- **foreign keys**
  - a primary key of one relation that is an attribute in some other

# Schema of relational DBs

- schema
  - logical structure of a database
- subschema
  - portion of a database
  - e.g., a given user has access to

Table: Cyber Security students

| First_Name | Last_Name | ID |
|---|---|---|
| ….. | …. | …. |

Table: CS-306 students

| First_Name | Last_Name | ID | … |
|---|---|---|---|
| ….. | …. | …. | |

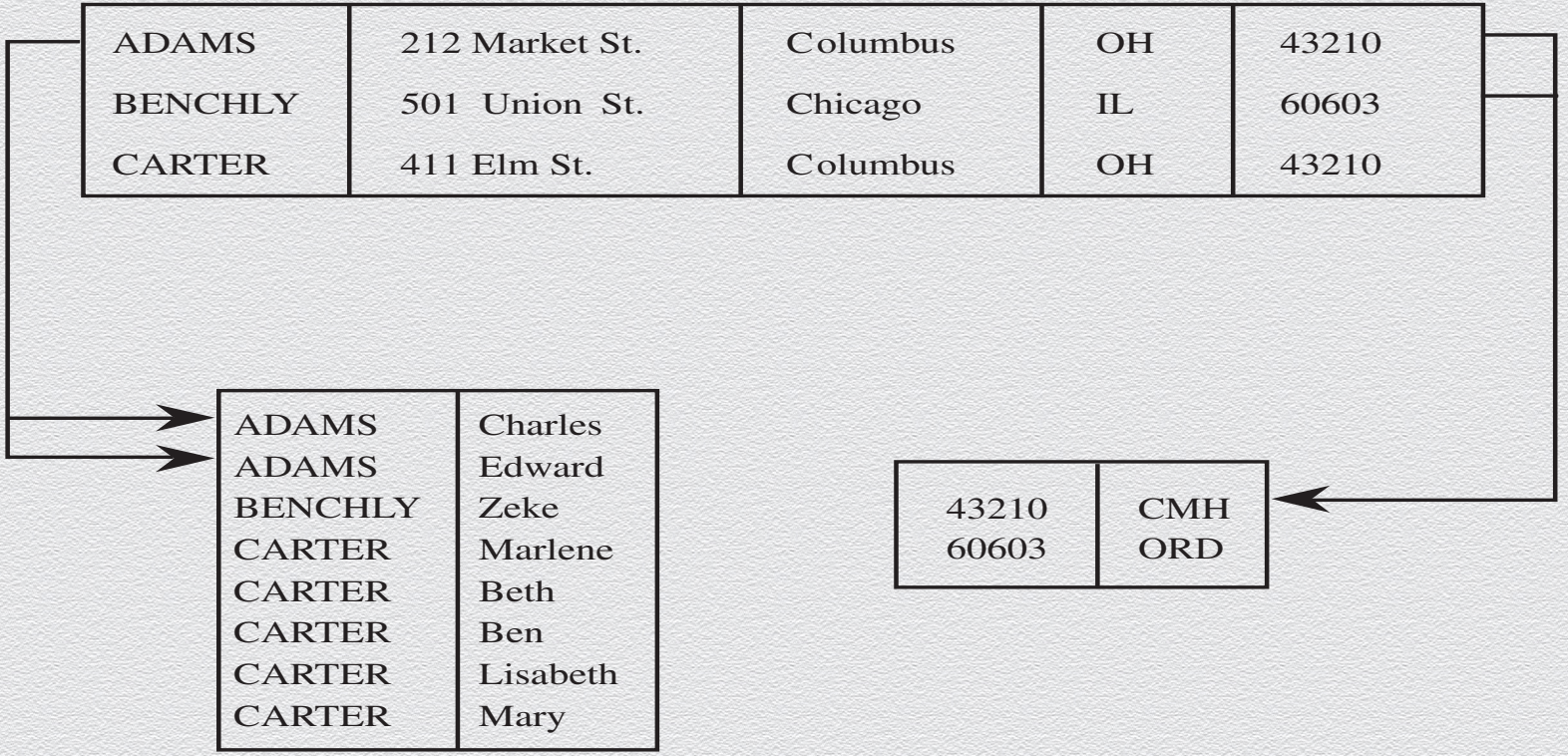
Table: CS-579 students

| First_Name | Last_Name | ID | … |
|---|---|---|---|
| ….. | …. | …. | |

Table: CS579 & CS-306 students

| First_Name | Last_Name | ID | Average Grade | … |
|---|---|---|---|---|
| ….. | …. | …. | | |

# A database example



# Database queries

Commands for accessing databases

- how information in a relational DBs can be retrieved and updated
  - specify how to retrieve, modify, add, or delete fields or records
  - specify how to derive information from database contents

The most common database query language is SQL

- Structured Query Language (SQL)
  - very widely used in practice: successful, solid technology
    - runs in banks, hospitals, governments, businesses, …
    - offered in cloud platforms (e.g., Azure SQL, AWS RDB)

# SQL – general features

Rich set of operations

- data manipulation, retrieval, presentation
- nested queries, operators, pattern matching

Main operations

- SELECT: retrieves data from a relation
- UPDATE: update fields in a relation
- DELETE: deletes tuples from a relation
- INSERT: adds tuples to a relation

# Example SQL Query

- SELECT *
  FROM HOME_ADDRESS
  WHERE ZIP='43210'

| Name | First | Address | City | State | Zip | Airport |
|---|---|---|---|---|---|---|
| ADAMS | Charles | 212 Market St. | Columbus | OH | 43210 | CMH |
| ADAMS | Edward | 212 Market St. | Columbus | OH | 43210 | CMH |
| CARTER | Marlene | 411 Elm St. | Columbus | OH | 43210 | CMH |
| CARTER | Beth | 411 Elm St. | Columbus | OH | 43210 | CMH |
| CARTER | Ben | 411 Elm St. | Columbus | OH | 43210 | CMH |
| CARTER | Lisabeth | 411 Elm St. | Columbus | OH | 43210 | CMH |
| CARTER | Mary | 411 Elm St. | Columbus | OH | 43210 | CMH |

Table: Home_Address

| Name | First | Address | City | State | Zip | Airport |
|---|---|---|---|---|---|---|
| ADAMS | Charles | 212 Market St. | Columbus | OH | 43210 | CMH |
| ADAMS | Edward | 212 Market St. | Columbus | OH | 43210 | CMH |
| BENCHLY | Zeke | 501 Union St. | Chicago | IL | 60603 | ORD |
| CARTER | Marlene | 411 Elm St. | Columbus | OH | 43210 | CMH |
| CARTER | Beth | 411 Elm St. | Columbus | OH | 43210 | CMH |
| CARTER | Ben | 411 Elm St. | Columbus | OH | 43210 | CMH |
| CARTER | Lisabeth | 411 Elm St. | Columbus | OH | 43210 | CMH |
| CARTER | Mary | 411 Elm St. | Columbus | OH | 43210 | CMH |

# SELECT operation

`SELECT [FROM WHERE]`

- projections, range restrictions, aggregation, etc.
- JOIN sub-query related to set operations

Table: CS-579 students

| First_Name | Last_Name | ID | Age | … |
|---|---|---|---|---|
| John | Myers | 123459 | 20 | |
| Olga | Johnson | 227800 | 21 | |
| Alex | Klein | 211123 | 22 | |
| ….. | …. | …. | | |

Table: CS-306 students

| First_Name | Last_Name | ID | Final_Grade | … |
|---|---|---|---|---|
| John | Myers | 123459 | A+ | |
| Maria | Palm | 222235 | A+ | |
| Alex | Klein | 211123 | A- | |
| ….. | …. | …. | | |

# SQL syntax example 1

```
SELECT  First_Name
FROM CS-306
WHERE Final_Grade = A+
```

Table: CS-306 students

| First_Name | Last_Name | ID | Final_Grade | … |
|---|---|---|---|---|
| John | Myers | 123459 | A+ | |
| Maria | Palm | 222235 | A+ | |
| Alex | Klein | 211123 | A- | |
| ….. | …. | …. | | |

- SELECT statement
  - used to select data FROM one or more tables in a database
- result-set is stored in a result table
- WHERE clause is used to filter records in terms of attribute contents

# SQL syntax example 2

```
SELECT  Last_Name
FROM CS-579
WHERE age=21
ORDER BY First_Name ASC
LIMIT 3
```

Table: CS-579 students

| First_Name | Last_Name | ID | Age | … |
|---|---|---|---|---|
| John | Myers | 123459 | 20 | |
| Olga | Johnson | 227800 | 21 | |
| Alex | Klein | 211123 | 22 | |
| ….. | …. | …. | | |

- ORDER BY
  - used to order data following one or more fields (columns)
- LIMIT
  - allows to retrieve just a certain numbers of records (rows)

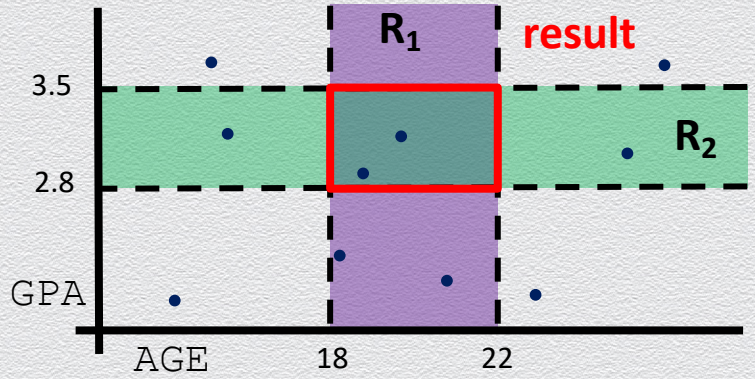
# SQL syntax example 3

```
SELECT * FROM STUDENT
   WHERE 18  < AGE < 22
   AND 2.8 < GPA < 3.5
```

Table: CS-579 students

| First_Name | Last_Name | ID | Age | GPA | … |
|---|---|---|---|---|---|
| John | Myers | 123459 | 20 | 3.5 | |
| Olga | Johnson | 227800 | 21 | 4.0 | |
| Alex | Klein | 211123 | 22 | 2.9 | |
| ….. | …. | …. | | | |

- range searching




intersection of partial results

# Database security

- DBs store **data** and provide **information** to their users
- DB security
  - ensure users update or retrieve information in **a reliable and controlled manner**
- **CIA – confidentiality, integrity, availability**
  - protect sensitive data **& disallow unauthorized leakage of information**
  - ensure data integrity **& guarantee correctness/consistency of authorized operations**
  - allow DB access **& ensure authorized access at all times**

# Confidentiality & integrity requirements

- **Physical / logical / element integrity**
  - e.g., ensure reliability (i.e., running for long times without interruptions)
  - e.g., protect database as a whole against catastrophic failures
  - e.g., updates do not change the DB schema
  - e.g., elementary data are inserted with correct / accurate values by authorized data “owners”
- **Data / privacy protection**
  - e.g., protect against unauthorized **direct or indirect** disclosure of information
  - e.g., protect against server breaches

# Additional DB security requirements

- **Auditability**
  - e.g., DB accessed are recorded and can be traced any time in the future
- **Access control**
  - e.g., different users get different DB views and can update only their “own” data
- **User authentication**
  - e.g., positively identify users (both for auditability and access control)

# Database security in the man-machine scale…

Difference to operating-system security

- DB security controls **access to information** more than access to data

# Integrity rules

- **entity integrity rule**
  - no PK component of a base relation is allowed to accept nulls
- **referential integrity rule**
  - the database must not contain unmatched foreign key values
- **application specific integrity rules**
  - field checks: correct data entry
  - scope checks: queries over statistical DBs of large support
  - consistency checks: guarantee users get the same DB view

# Concurrency via locked query-update cycles

Controls for DB consistency (when multiple users access DB concurrently)

- solves the “double-booking” or “full-flight” problems
- due to concurrent reads & writes
  - e.g., two distinct agencies reserve at the same time the same airplane seat which appears to be empty for a given flight
  - e.g., an agency cancels a previous reservations but another agency cannot reserve it as the flight still appears to be full

**Solutions**

- treat a (seat availability) query and (seat reservation) update as one **single atomic operation**
- use **locks** to block read (seat availability) requests while a write (seat cancelation) operation is still processed

# Consistency via two-phase updates

Control for DB consistency (when failures result in partial data updates)

- solves the “inconsistent inventory” problem

Phase 1: **Intent**

- DBMS does everything it can to **prepare** for the update
  - collects records, opens files, locks out users, makes calculations
  - but it makes **no changes** to the database
- DBMS **commits** by writing a commit flag to the database

Phase 2: **Write**

- DBMS **completes** all update operations and **removes** the commit flag

If either phase fails, it is repeated without causing any harm to the DBMS!

# Other DB security mechanisms for integrity

- Error detection and correction codes to protect data integrity
- For recovery purposes, a database can maintain a change log, allowing it to repeat changes as necessary when recovering from failure
- Databases use locks and atomic operations to maintain consistency
  - writes are treated as atomic operations
  - records are locked during write so they cannot be read in a partially updated state

# SQL security model for access control

Discretionary access control using privileges and views, based on:

- **users**: authenticated during logon
- **actions**: include SELECT, UPDATE, DELETE, and INSERT
- **objects**: tables, views, columns (attributes) of tables and views

Users invoke actions on objects permitted or denied by DBMS

- when an object is created, it is assigned an owner
- initially only the owner has access to the object
- other users have to be issued with a **privilege**
  - (grantor, grantee, object, action, grantable)

# Sensitive data

- Inherently sensitive
  - passwords, locations of weapons
- From a sensitive source
  - confidential informant
- Declared sensitive
  - classified document, name of an anonymous donor
- Part of a sensitive attribute or record
  - salary attribute in an employment database
- Sensitive in relation to previously disclosed information
  - an encrypted file combined with the decryption key to open it

# Types of disclosures

- Exact data
  - e.g., finding the exact value of a field
- Bounds
  - e.g., finding a range in which a field value is contained
- Negative result
  - e.g., finding whether one has been convicted 0 times
- Existence
  - e.g., finding whether a person is in a black list
- Probable value
  - e.g., knowing that half of the students have outstanding loans

# Means of disclosure

- Direct inference
  - e.g., through a SQL query
- Inference by arithmetic
  - e.g., via computation of sums, counts, means, medians, etc.
  - e.g., via tracker attacks, e.g., count(a & b & c) = count(a) – count(a & ~(b & c))
  - e.g., by solving a linear system
- Aggregation
  - e.g., data mining
  - e.g., by correlating with data from other users, other sources, or prior knowledge
- Hidden data attributes/meta-data
  - e.g., file tags, geo-tags, device tracking / fingerprinting

# Disclosure-prevention techniques

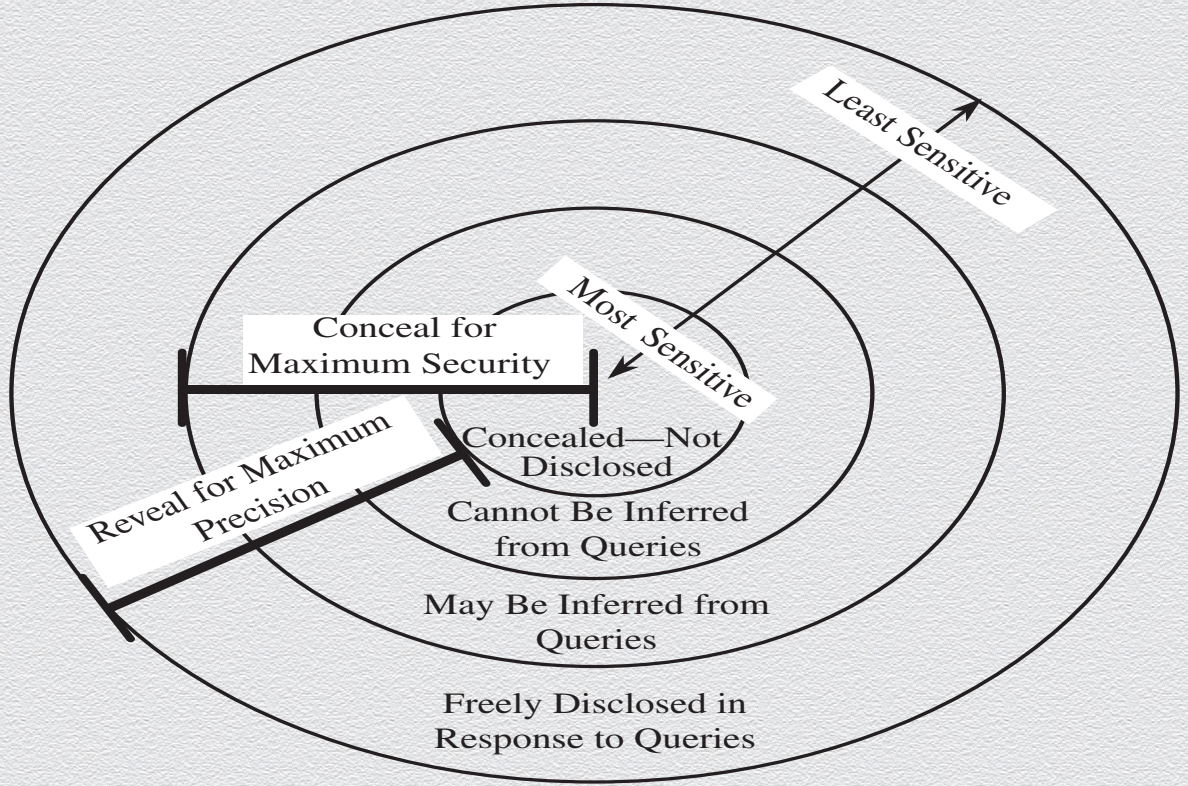
- Suppress obviously sensitive information
  - e.g., never return the SSN number of a customer or the disease of a patient
- Keep track of what each user knows based on past queries, e.g.,
  - use audit logs for the entire query history of a user or a group of users
  - compare new queries against possibly leaked information given past query history
- Disguise the data
  - e.g., perturb data by adding some “zero-mean” random noise
  - e.g., use of differential privacy techniques
- Cryptographically protect database
  - e.g., use of “structured-preserving” encryption

# Suppression techniques

- Limited response suppression
  - eliminate certain low-frequency elements from being displayed
- Combined results
  - use ranges, rounding, sums, averages
- Random samples and blocking small sample sizes
- Random data perturbation
  - randomly add/subtract a small error value to/from actual values
- Swapping
  - randomly swap values for individual records while keeping statistical results the same

# Security vs. precision

Precise, complete & consistent responses to queries against sensitive information make it more likely that the sensitive information will be disclosed



# Cryptographic means

Encrypting data records protects against leakage due to server breaches

- but it reduces utility/usability to zero…

Solution concept: “Compute over encrypted data”

- Multi-party computation
  - parties compute (reliably) only a specific result and nothing not implied by this!
- Fully-homomorphic encryption
  - encryption schemes that allow to compute any function over ciphertext data!
- Structure/Order-preserving encryption
  - encryption schemes that preserver a property over plaintext data (e.g., order)

# Take-home messages

Data & privacy protection

- way beyond data record/field suppression (of simple data contents)
  - e.g., keeping data from being dumped out of DB is insufficient to prevent disclosure
- all possible ways of maliciously deducing DB contents must be considered
  - e.g., by taking into account the possible ranges of data fields
  - e.g., by understanding what a priori information potential attackers may possess
- existing disclosure-prevention techniques induce inconvenient trade-offs
  - e.g., between utility and privacy (loss of precision/completeness makes DB unusable)
  - e.g., computing over encrypted data is still impractical

# Data mining

- Data mining uses statistics, machine learning, mathematical models, pattern recognition, and other techniques to discover patterns and relations on large datasets
- The size and value of the datasets present an important security and privacy challenge, as the consequences of disclosure are naturally high

# Data mining challenges

- Correcting mistakes in data
- Preserving privacy
- Granular access control
- Secure data storage
- Transaction logs
- Real-time security monitoring

# SQL injection (or SQLI) attack

- many web applications take user input from a form
- often a user's input is used literally in the construction of a SQL query submitted to a database
  - e.g.,

```
SELECT user FROM table WHERE name = 'user_input';
```

- an SQL injection attack involves placing SQL statements in the user input

# Login authentication query

- Standard query to authenticate users
  - select * from users where user='$usern' AND pwd='$password'
- Classic SQL injection attacks
  - Server side code sets variables $username and $passwd from user input to web form
  - Variables passed to SQL query
  - select * from users where user='$username' AND pwd='$passwd'
- Special strings can be entered by attacker
  - select * from users where user='M' OR '1=1' AND pwd='M' OR '1=1'
- Result: access obtained without password
- Solution: Careful with single quote characters
  - filter them out!

# 14.2 Buffer overflow

# Memory basics

Stack

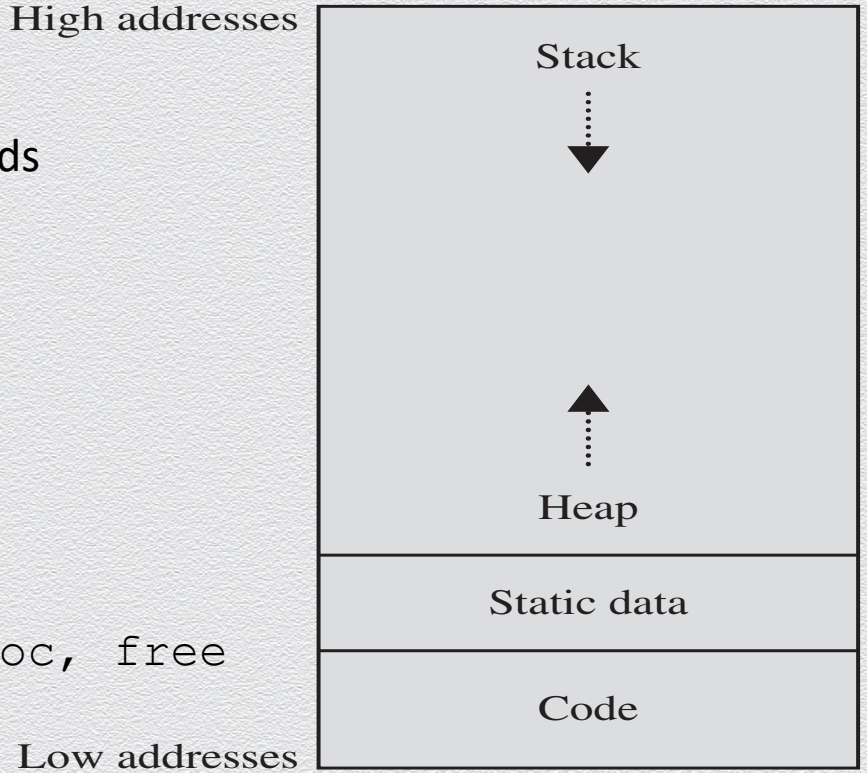
- used whenever a function call is made
- typically higher addresses growing downwards

Static data area

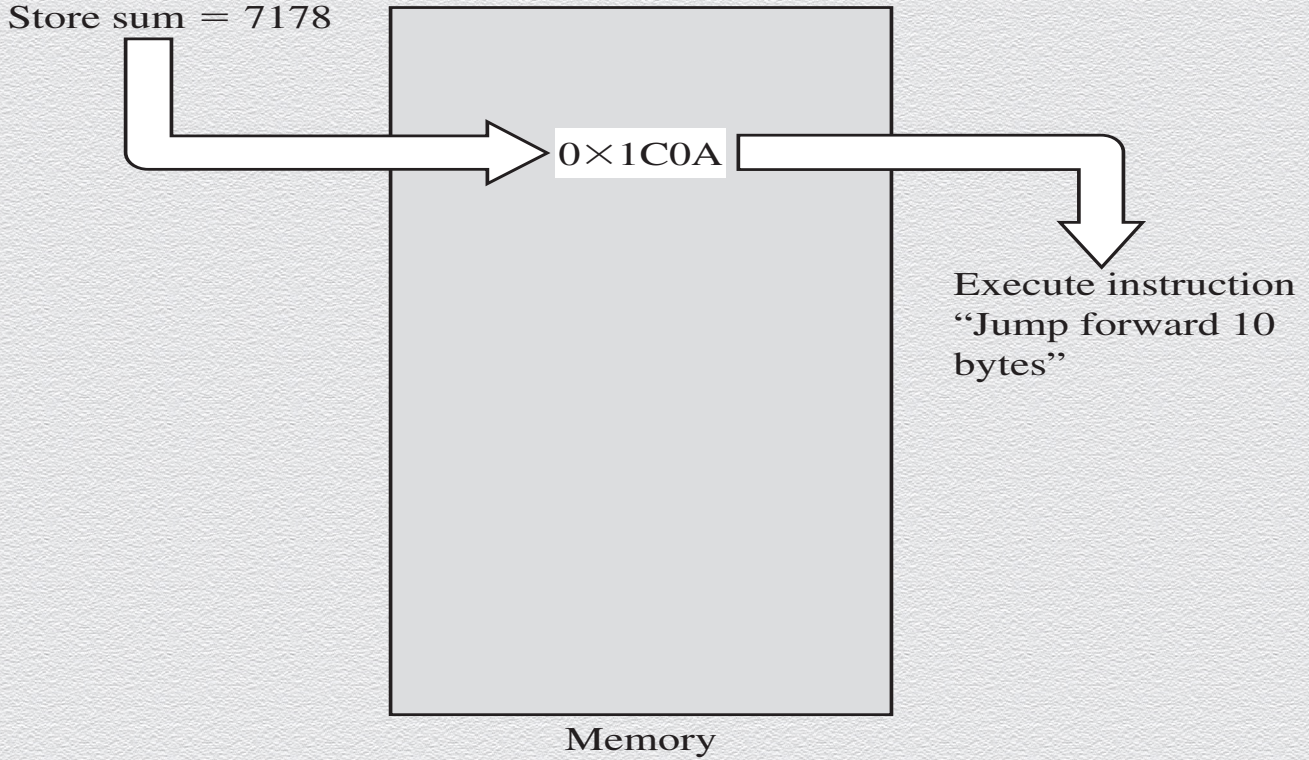
- global variables used by programs (not initialized with zero)
- e.g., `char s[] = "hello world"`

Heap

- begins after data area, growing upwards
- dynamically managed by `malloc, realloc, free`



# Data vs. Instructions



# Buffer overflows

Based on programmers' oversights (or programming languages **vulnerabilities**)

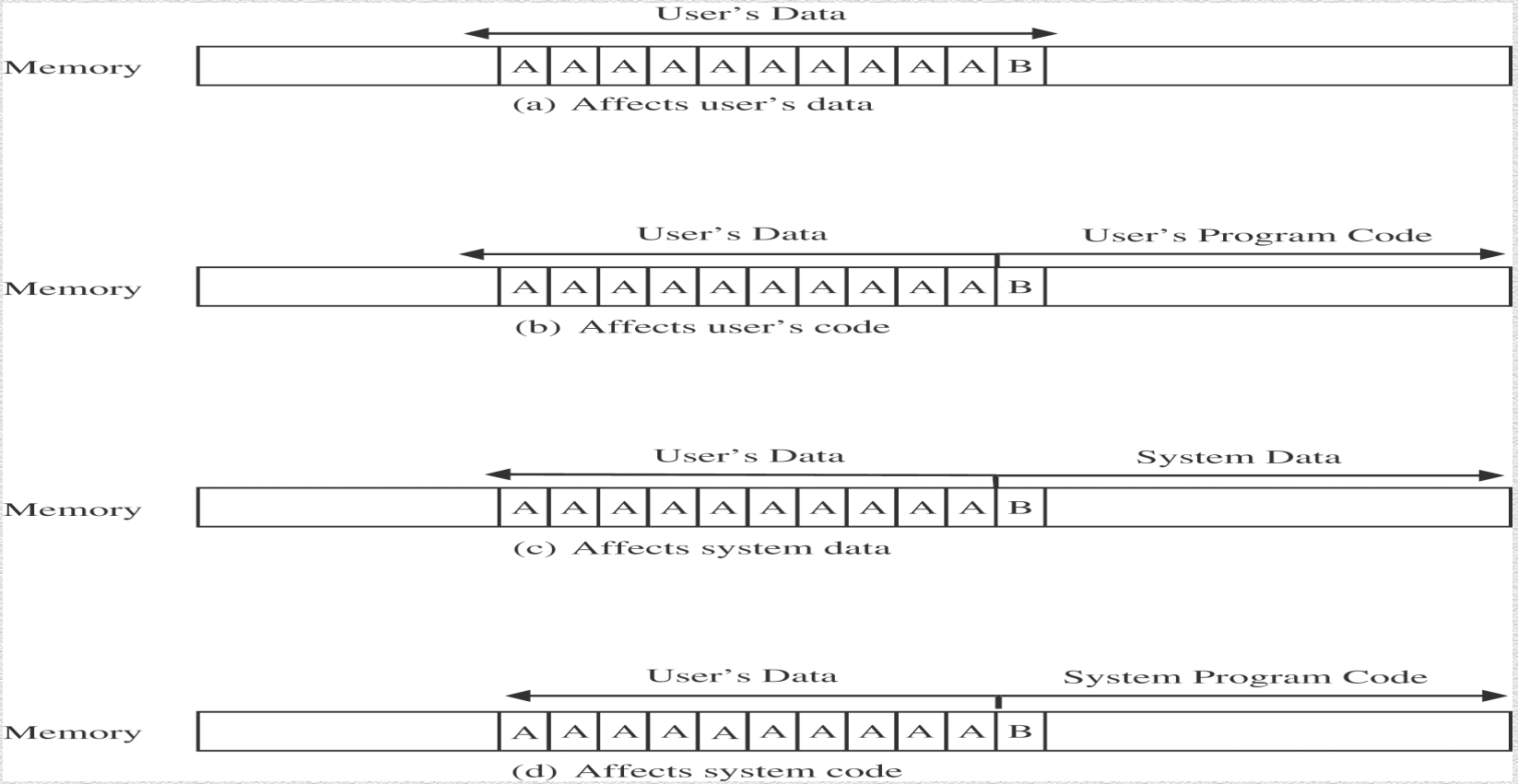
- exploited by attackers by **inputting more data than expected**
  - attacker's data that is written beyond the space allocated for it
  - e.g., a $10^{th}$ byte in a 9-byte array
- typical exploitable buffer overflow
  - users' inputs are expected to go into regions of memory allocated for data; instead
  - attacker's inputs **are allowed to overwrite** memory holding executable code
- attacker's challenge is to discover buffer-overflow vulnerabilities
  - find opportunities **leading to overwritten memory being executed**
  - **find the right code to input** (that inflicts some specific harm)

# Example: How buffer overflows happen

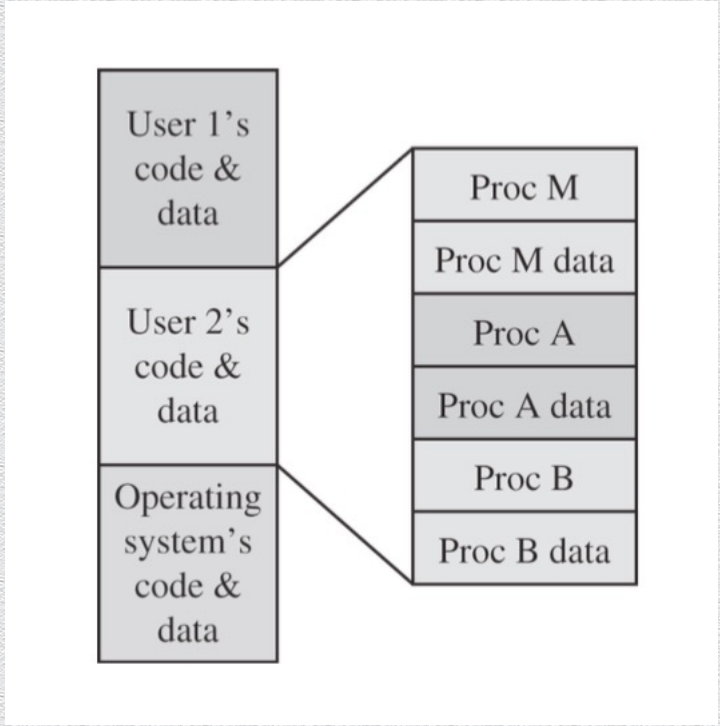
```
char sample[10];

int i;

for (i=0; i<=9; i++)
  sample[i] = 'A';

sample[10] = 'B';
```

(or `sample[i] = 'B';`)

# Overflows can affect data or code, or even the OS



# Overflows can affect other users

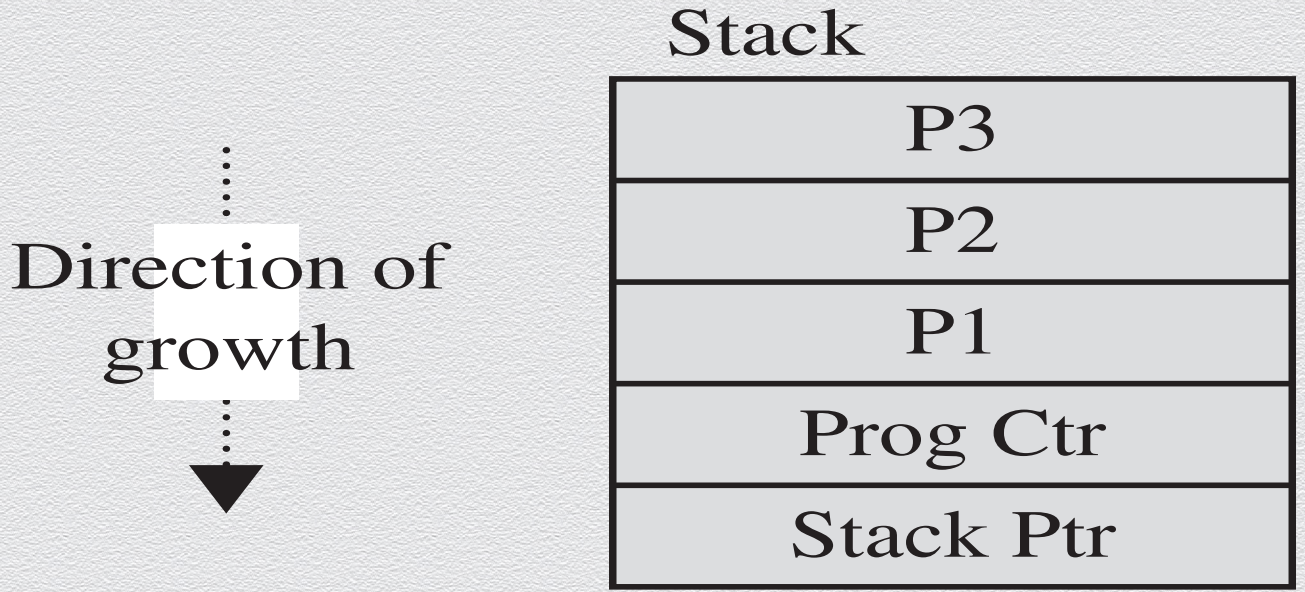


# Harm from buffer overflows

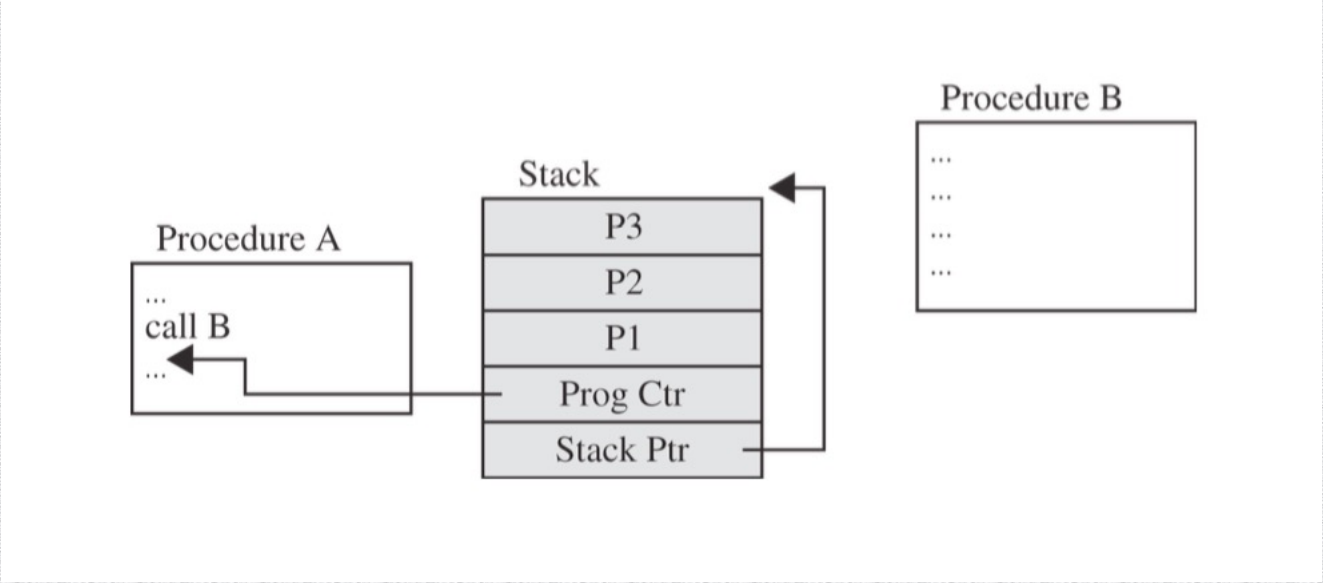
- overwrite
  - an instruction or data item of same program's data
    - e.g., PC and data in the stack so that PC points to the stack
  - data or code belonging to another program or the OS
    - e.g., part of the code in low memory, substituting new instructions
    - gives to attacker that program's execution privileges or root privileges
- results in
  - **unauthorized access**
  - **privilege escalation**

When successfully completed, attacker runs **maliciously written code** at **higher privilege levels**!

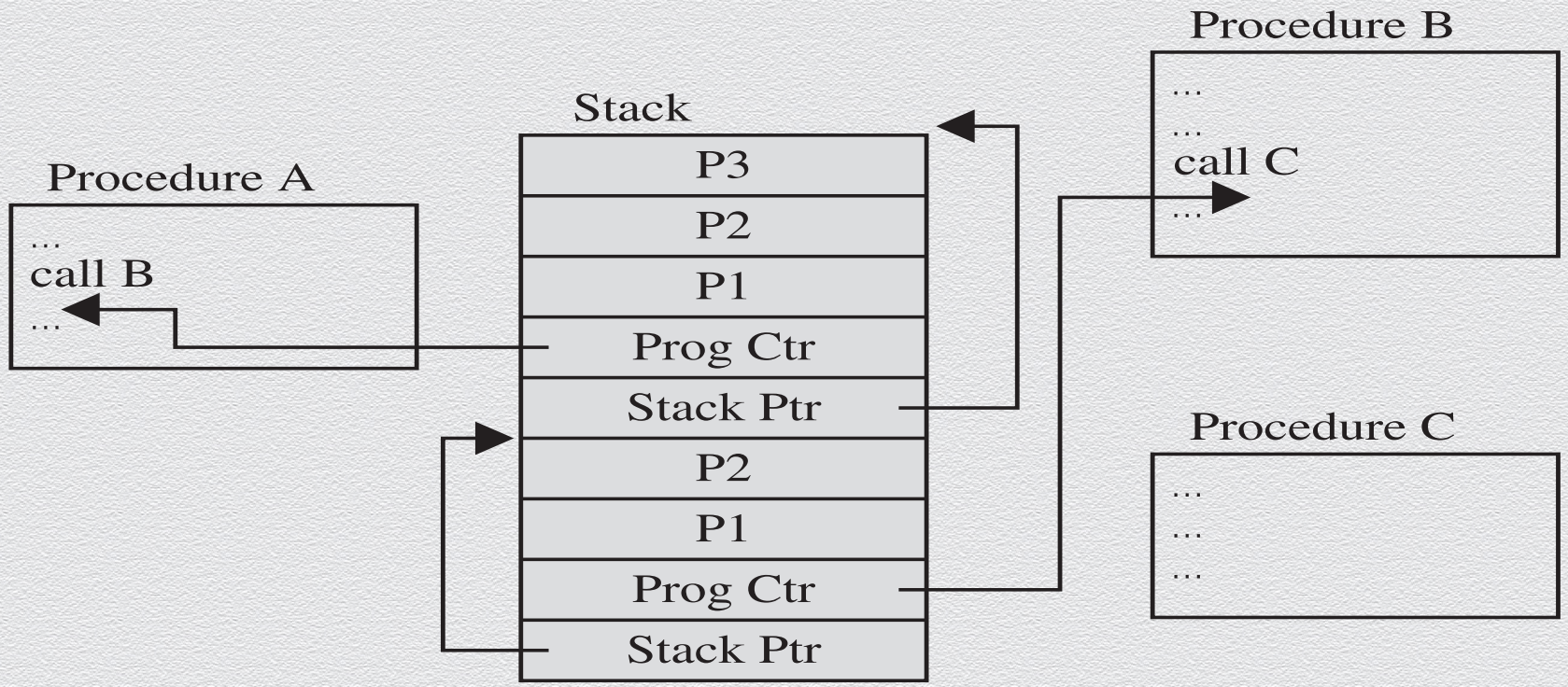
# The stack



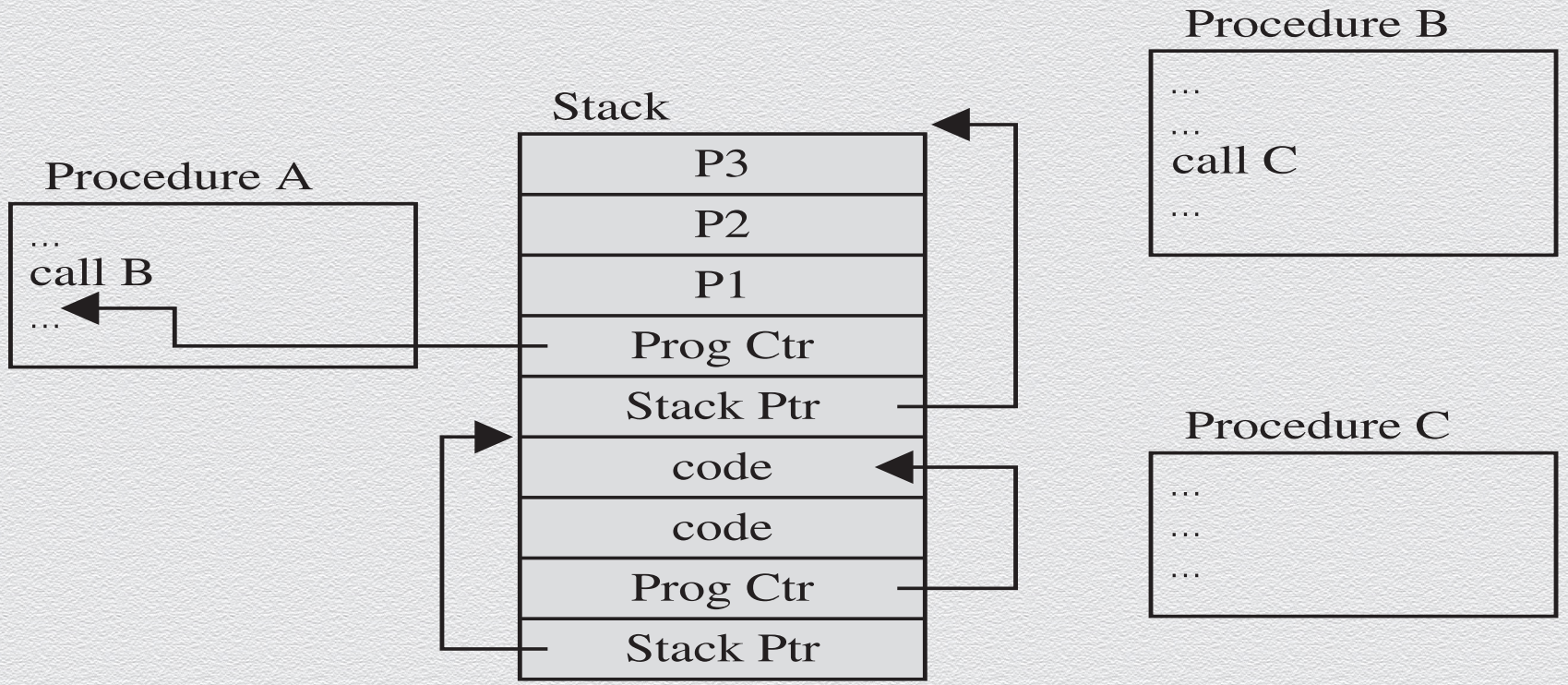
# The stack after procedure calls: A calls B



# The stack after nested procedure calls: A calls B, B calls C



# Compromised stack



# Attack structure

To exploit a buffer overflow vulnerability the attacker must address some challenges

**[1]** write malicious code (that does some harm)

- not trivial task (depends on next steps/challenges)
- e.g., a special type of malicious code called **shellcode** can be written

**[2]** inject the malicious code into the memory of the target program (TP)

- control the contents of the buffer in TP
- e.g., in following example, **by storing the malicious code in the input file**

**[3]** jump to (and execute) the malicious code

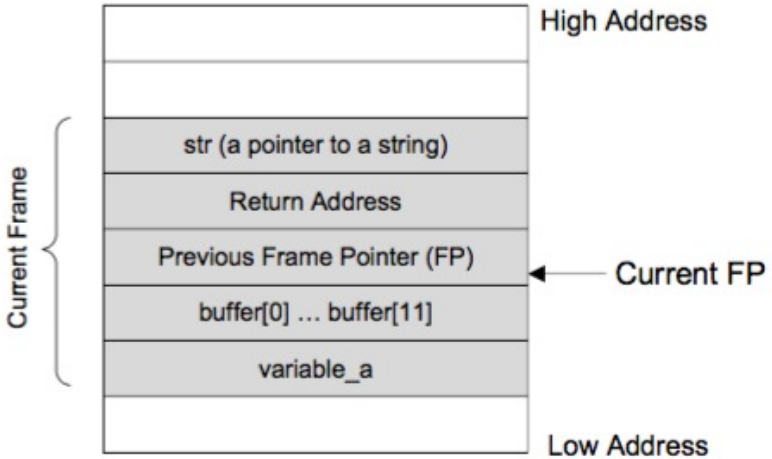
- control the execution of TP and execute injected malicious code
- e.g., in following example, **by pointing the program counter to the right position in the stack**

# Buffer-overflow vulnerability: A specific example

- layout of stack after the program execution has entered function func()
- grows from-high-to-low addresses (but `buffer` grows from-low-to-high)

```
void func (char *str) {
  char buffer[12];
  int  variable_a;
  strcpy (buffer, str);
}

Int main() {
  char *str = "I am greater than 12 bytes";
  func (str);
}
```

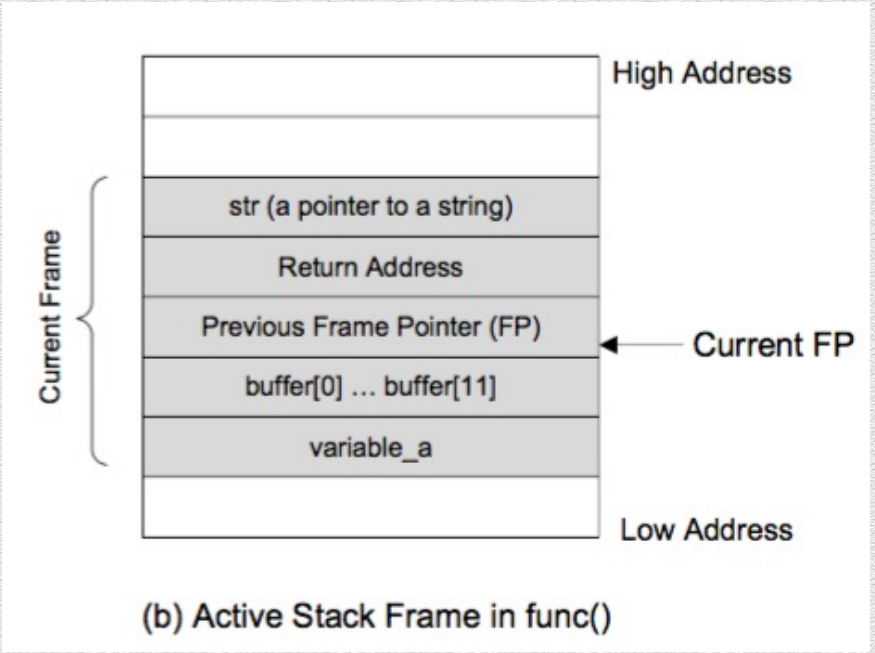
(a) A code example




(b) Active Stack Frame in func()

# Data stored in current frame

- local data: `buffer, variable_a`
- function parameter: `str`
- **return address**
  - what to execute after function ends
  - command after function call
- frame pointer (FP)
  - pointer on current frame that is used to reference local data & function parameters
  - e.g., `variable_a` is referred to as FP-16, `buffer` as FP-12, `str` as FP+8
- previous frame pointer
  - pointer to previous frame (corresponding to function that called `func()` )

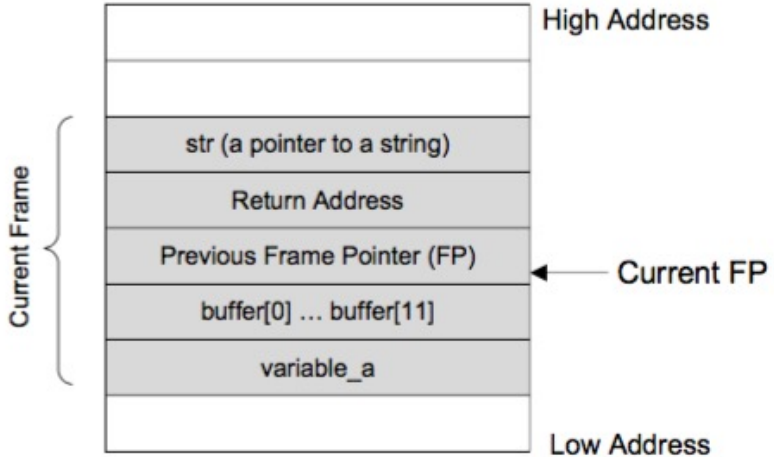


(b) Active Stack Frame in func()

# Buffer overflow: [2] Inject malicious code

- `strcpy(buffer, str)` copies the contents from `str` to `buffer[]`
- the string pointed by `str` has more than 12 chars, while the size of `buffer[]` is only 12
- `strcpy()` does not check whether the boundary of `buffer[]` has reached
  - it only stops when seeing the end-of-string character '\0'
- contents in the memory above `buffer[]` will be overwritten by the characters at the end of `str`

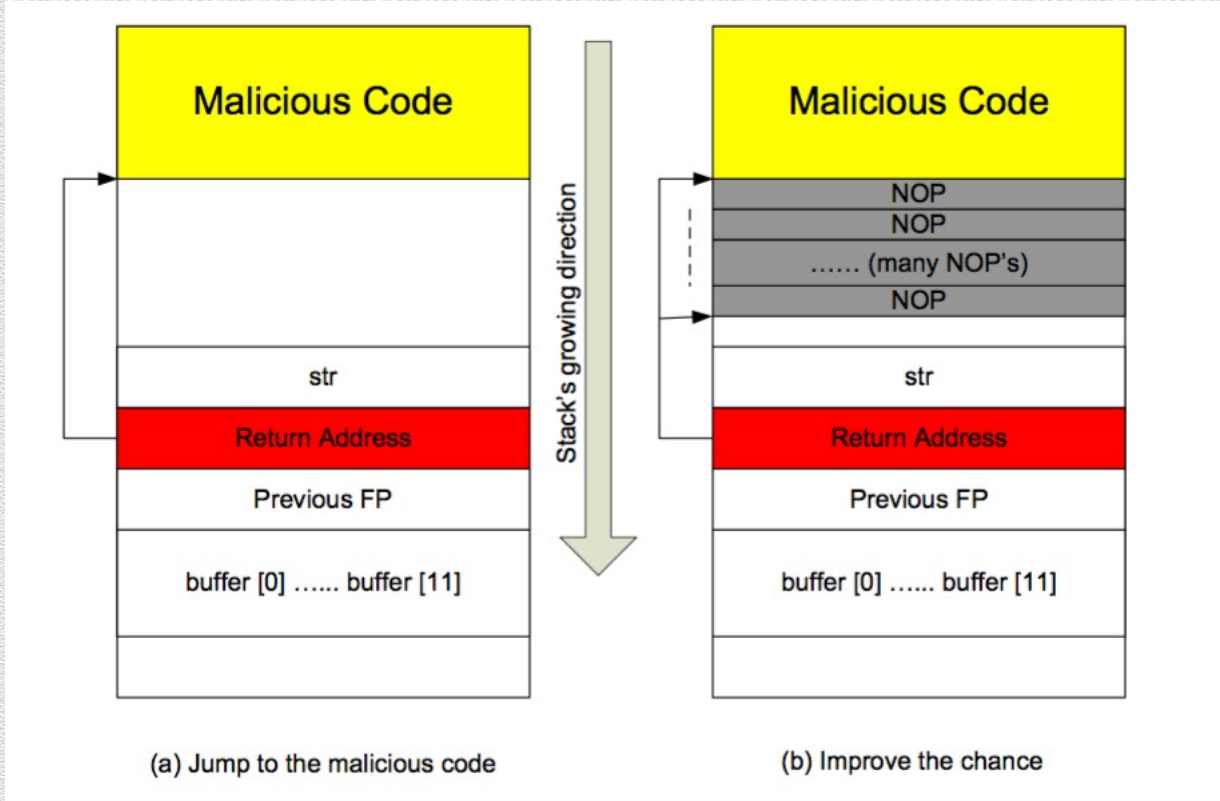
```
void func (char *str) {
  char buffer[12];
  int  variable_a;
  strcpy (buffer, str);
}

Int main() {
  char *str = "I am greater than 12 bytes";
  func (str);
}
```



# [2] Inject malicious code: A more interesting example

```
/* stack.c */ /* This program has a buffer overflow vulnerability. */

#include <stdlib.h> #include <stdio.h> #include <string.h>

int func (char *str) {
char buffer[12];
strcpy(buffer, str); /* This statement has a buffer overflow problem */
return 1; }

int main(int argc, char **argv) {
char str[517];
FILE *badfile;
badfile = fopen("badfile", "r");
fread(str, sizeof(char), 517, badfile);
func (str);
printf("Returned Properly\n");
return 1; }
```

# [3] Jump to the malicious code



# [3] Jump to the malicious code

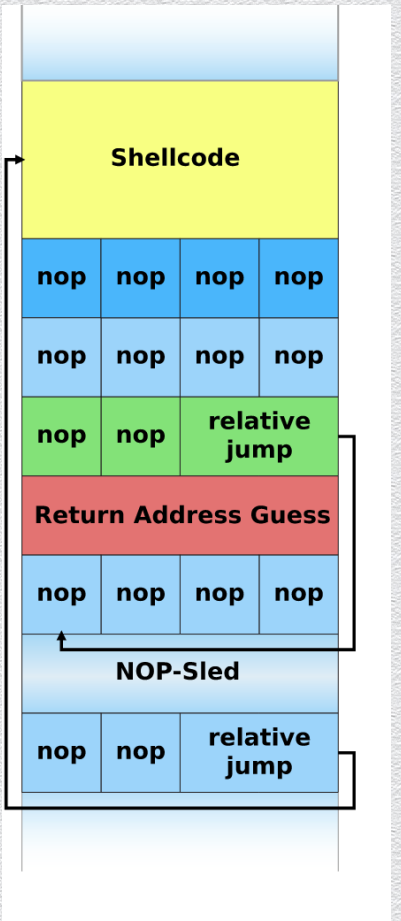
To run the malicious code (already injected in TP's stack)

- need to **know the absolute address of the malicious code**
  - overflow the buffer so that this address overwrites the return address
  - when function returns, the malicious code will run
- strategies to find where the malicious code starts
  - make a copy of the TP and find (approximate) the start of malicious code by debugging
  - set-UID TP: allows to run an executable with the privileges of the executable's owner
- if the TP runs remotely, you can always guess
  - stack usually starts at the same address and is not very deep
  - range of addresses to guess is actually quite small

# [3] Jump to the malicious code: Nopsled

To improve the chance of success

- add many NOP operations to the beginning of the malicious code
- NOP (no operation) is a special instruction
  - does nothing other than advancing to the next instruction
  - therefore, as long as the guessed address points to one of the NOPs, the attack will be successful!
  - with NOPs, the chance of guessing the correct entry point to the malicious code is significantly improved!



# [1] Write malicious code: Shellcode

Powerful code that invokes a shell

- attacker can run any command in that shell!
- if TP has root privileges, then any command runs also at root level!
- e.g., C program that simply launches a shell:

```
#include <stdio.h>
int main( ) {
char *name[2];
name[0] = ‘‘/bin/sh’’;
name[1] = NULL;
execve(name[0], name, NULL); }
```

# [1] Write malicious code: Further challenges

Directly compiling the previous program into binary code is not enough

- (1) to invoke system call `execve()`, need to know the address of the string "`/bin/sh`"
  - storing and deriving the address of this argument is not easy
- (2) function `strcpy()` will stop in the first occurrence of a `NULL` (i.e., 0) value

- e.g., C program that simply launches a shell:

```
#include <stdio.h>
int main( ) {
char *name[2];
name[0] = ``/bin/sh'';
name[1] = NULL;
execve(name[0], name, NULL); }
```

# [1] Write malicious code: Solutions

Directly compiling the previous program into binary code is not enough

- (1) to invoke system call `execve()`, need to know the address of the string "`/bin/sh`"
  - **push string "`/bin/sh`" onto stack and use the stack pointer `esp` to get its location**
- (2) function `strcpy()` will stop in the first occurrence of a `NULL` (i.e., 0) value
  - **convert instructions containing 0 into equivalent instructions not containing 0**
  - **e.g., to store 0 to a register, use XOR operation, instead of directly assigning 0**
- e.g., C program that simply launches a shell:

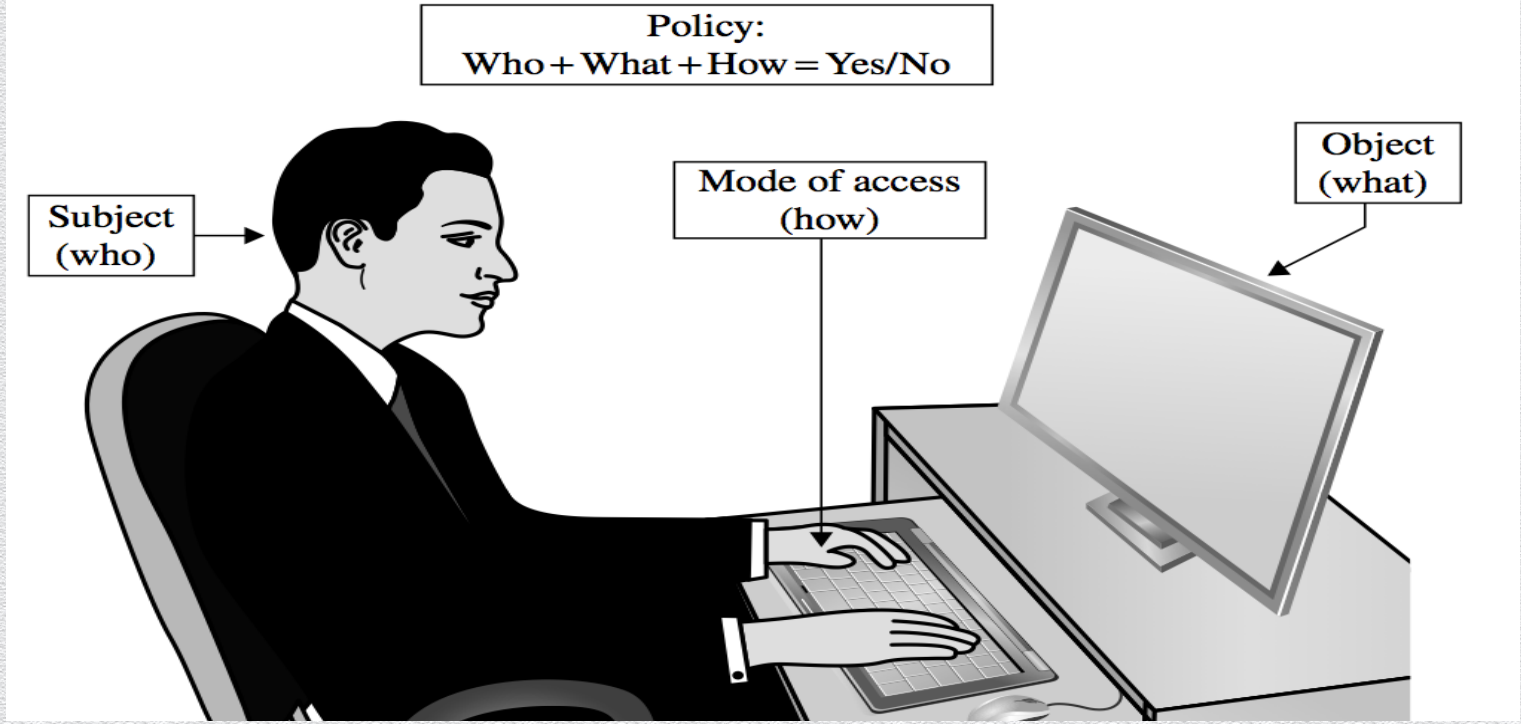
```
#include <stdio.h>
int main( ) {
char *name[2];
name[0] = ''/bin/sh'';
name[1] = NULL;
execve(name[0], name, NULL); }
```

# [1] Write malicious code: Final malicious code

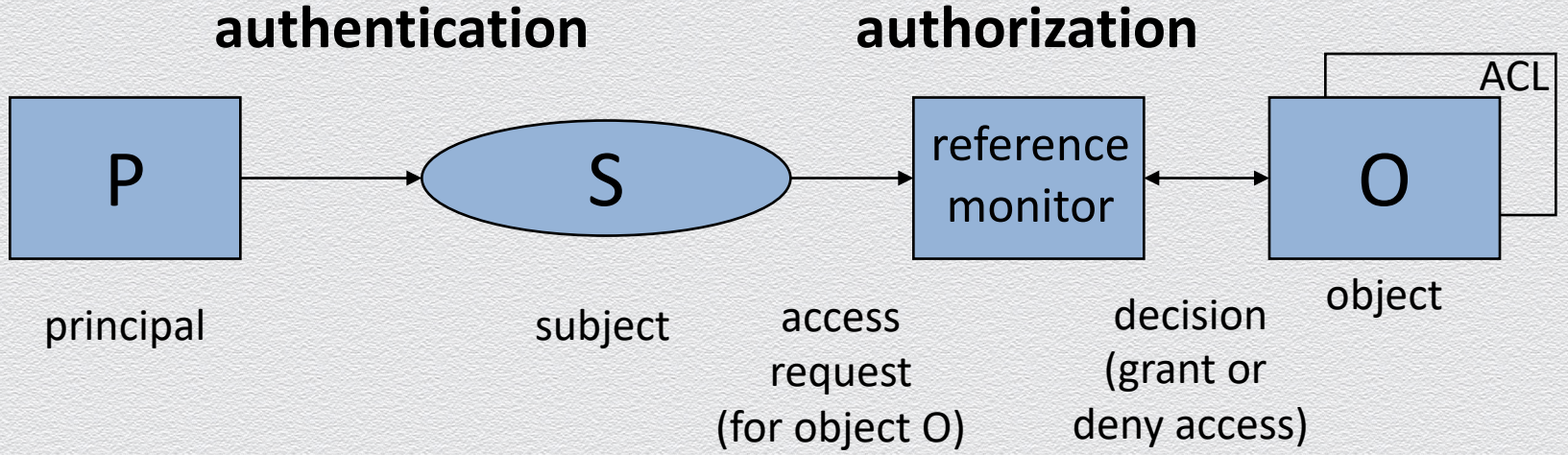
```
xorl %eax,%eax
pushl %eax              # push 0 into stack (end of string)
pushl $0x68732f2f       # push "//sh" into stack
pushl $0x6e69622f       # push "/bin" into stack
movl %esp,%ebx          # %ebx = name[0]
pushl %eax              # name[1]
pushl %ebx              # name[0]
movl %esp,%ecx          # %ecx = name
cdq                     # %edx=0
movb $0x0b,%al
int $0x80               # invoke execve(name[0], name, 0)
```

# 14.3 Access control

# Access control (AC)



# General structure of access control mechanism



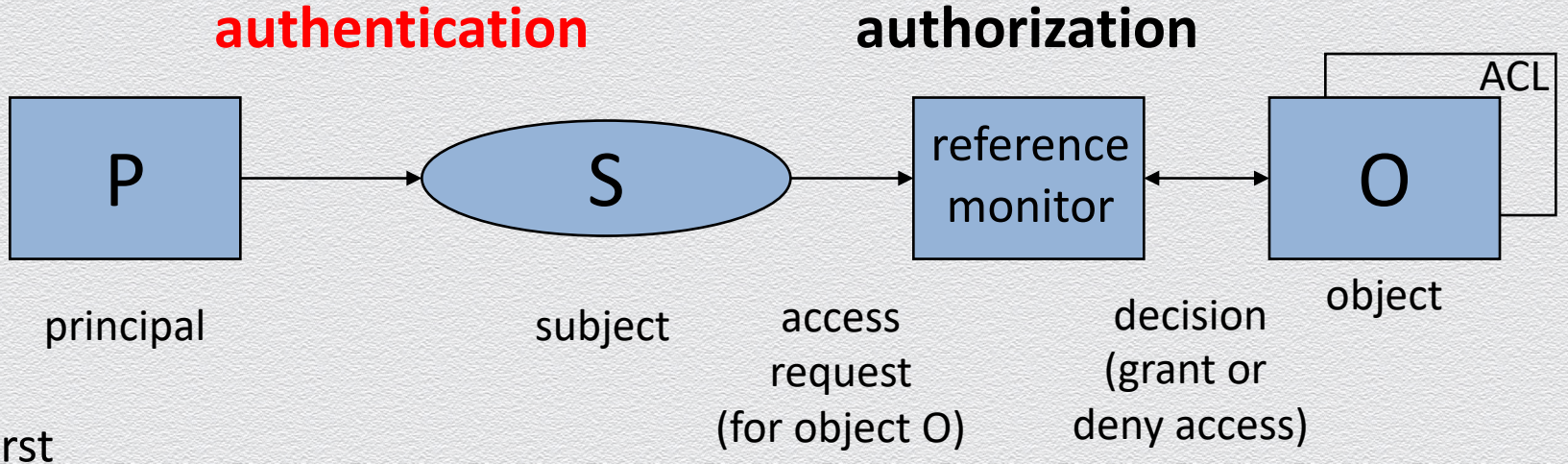
# Basic terminology

- Subject/Principal
  - active entity – user or process
- Object
  - passive entity – file or resource
- Access operations
  - vary from basic memory access (read, write) to method calls in object-oriented systems
  - comparable systems may use different access operations or attach different meanings to operations which appear to be the same

# Access operation

- Access right
  - right to perform an (access) operation
- Permission
  - typically a synonym for access right
- Privilege
  - typically a set of access rights given directly to roles like administrator, operator, ...

# Authentication




First

- reference monitor verifies the identity of the principal making the request
  - a user identity is one example for a principal
  - cf. authentication Vs. identification

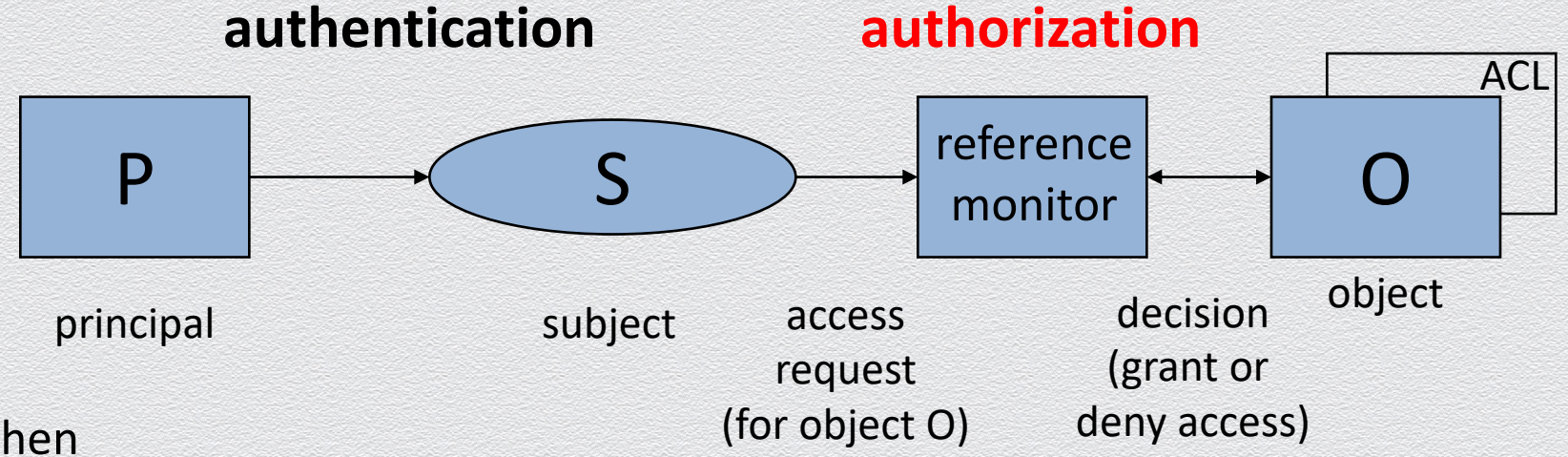
# Authentication

- user enters username and password
- if the values entered are correct, the user is “authenticated”
- we could say: “The machine now runs on behalf of the user”
  - this might be intuitive, but it is imprecise
- log on creates a process that “runs with access rights” assigned to the user
  - the process runs under the user identity of the user who has logged on

# Users & user identities

- requests to reference monitor do not come directly from a user or a user identity, but from a process
- in the language of access control, the process “speaks for” the user (identity)
- the active entity making a request within the system is called the subject
- must distinguish between three concepts
  - user: person
  - principal: identity (e.g., user name) used in the system, possibly associated with a user
  - subject: process running under a given user identity

# Authorization



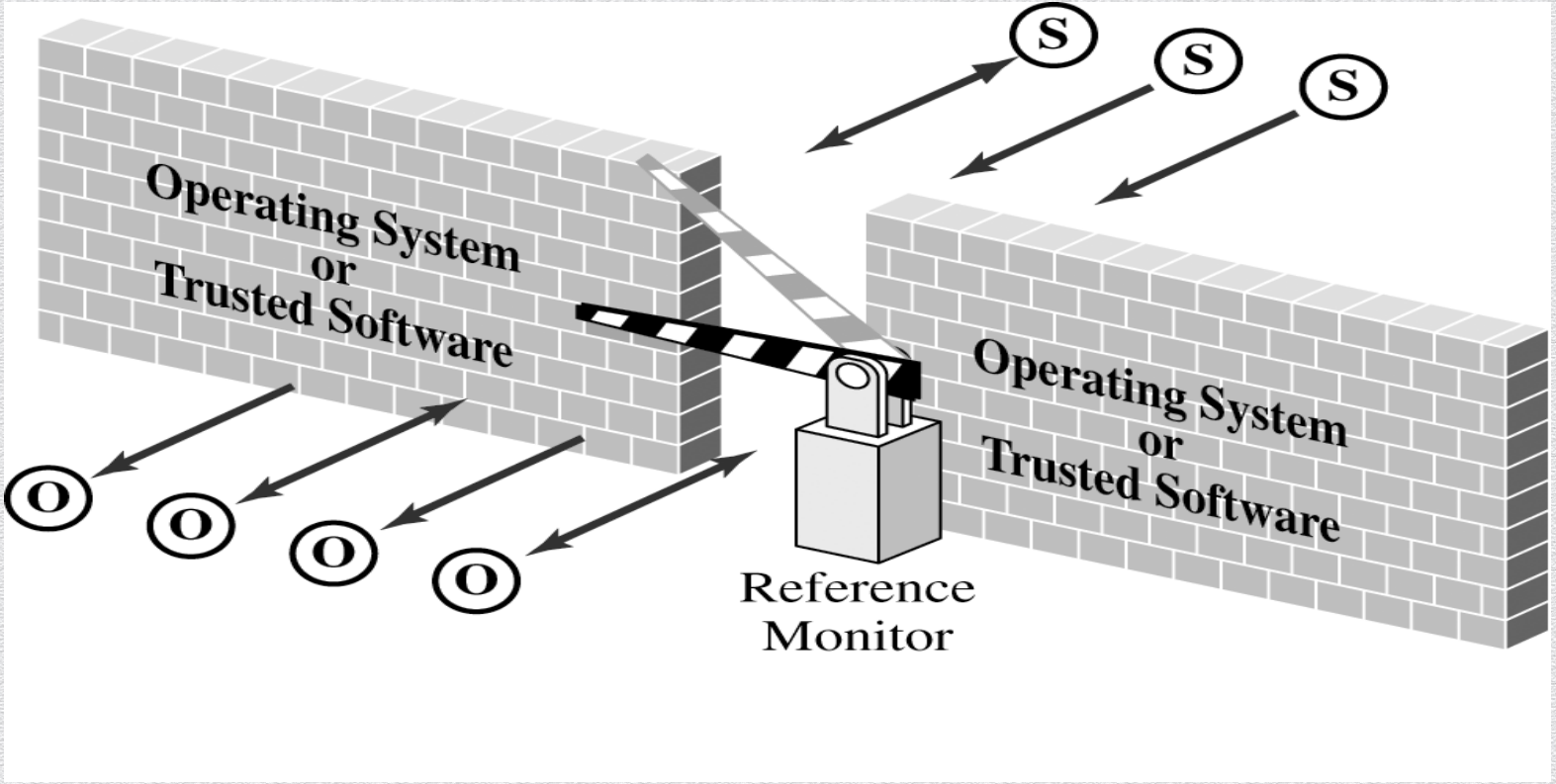

Then

- reference monitor decides whether access is granted or denied
- has to find and evaluate the security policy relevant for the given request
- “easy” in centralized systems; in distributed systems,
  - how to find all relevant policies? how to make decisions if policies may be missing?

# Principals & subjects

- a principal is an entity that can be granted access to objects or can make statements affecting access control decisions
  - example: user ID
- subjects operate on behalf of (human users we call) principals
- access is based on the principal's name bound to the subject in some unforgeable manner at authentication time
  - example: process (running under a user ID)

# Reference monitor



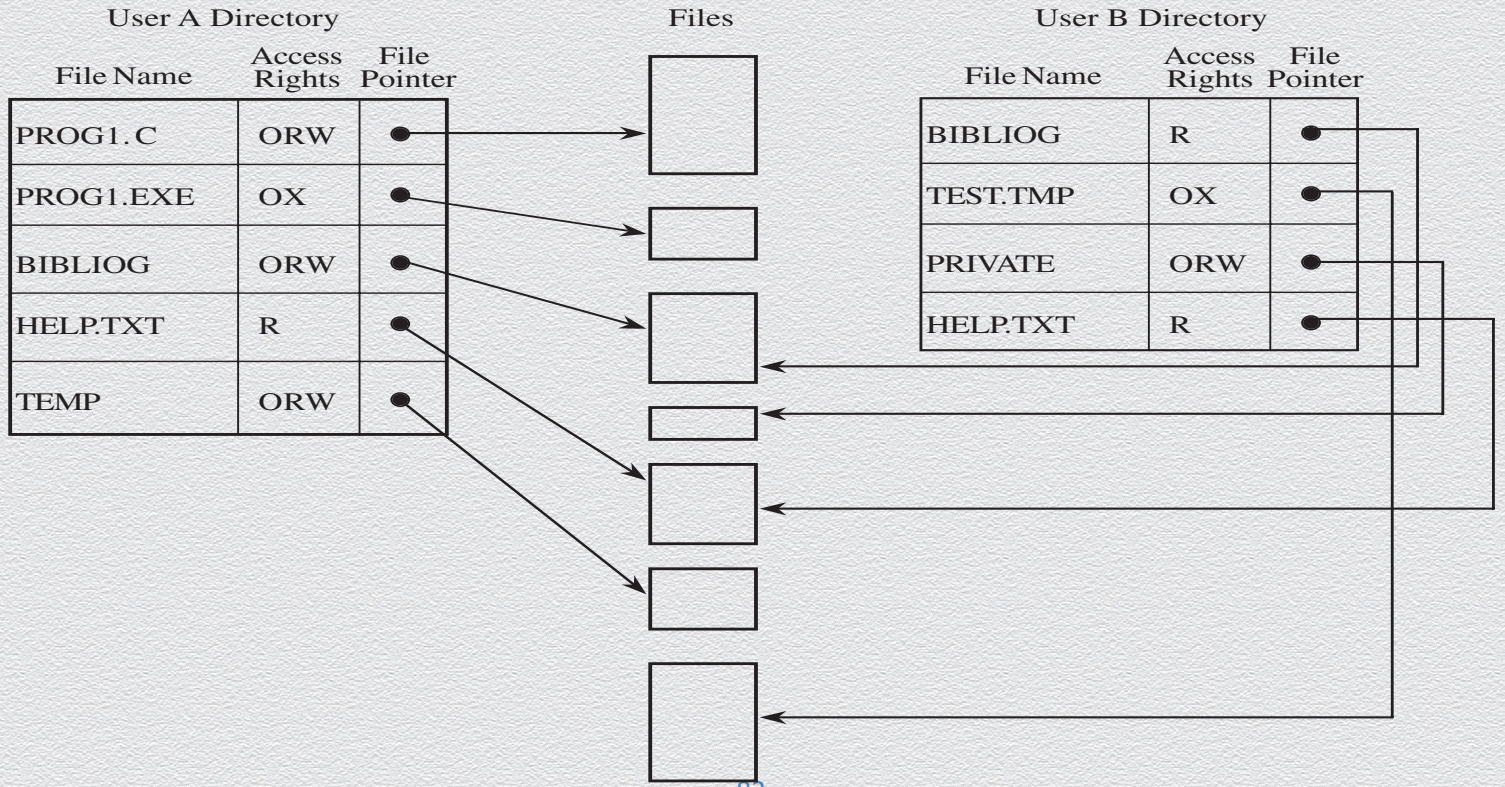
# AC policies

- Goals
  - Check every access
  - Enforce least privilege
  - Verify acceptable usage
- Track users' access
- Enforce at appropriate granularity
- Use audit logging to track accesses

# Implementing AC policies

- Reference monitor
- Access control directory
- Access control matrix
- Access control list
- Privilege list
- Capability
- Procedure-oriented access control
- Role-based access control

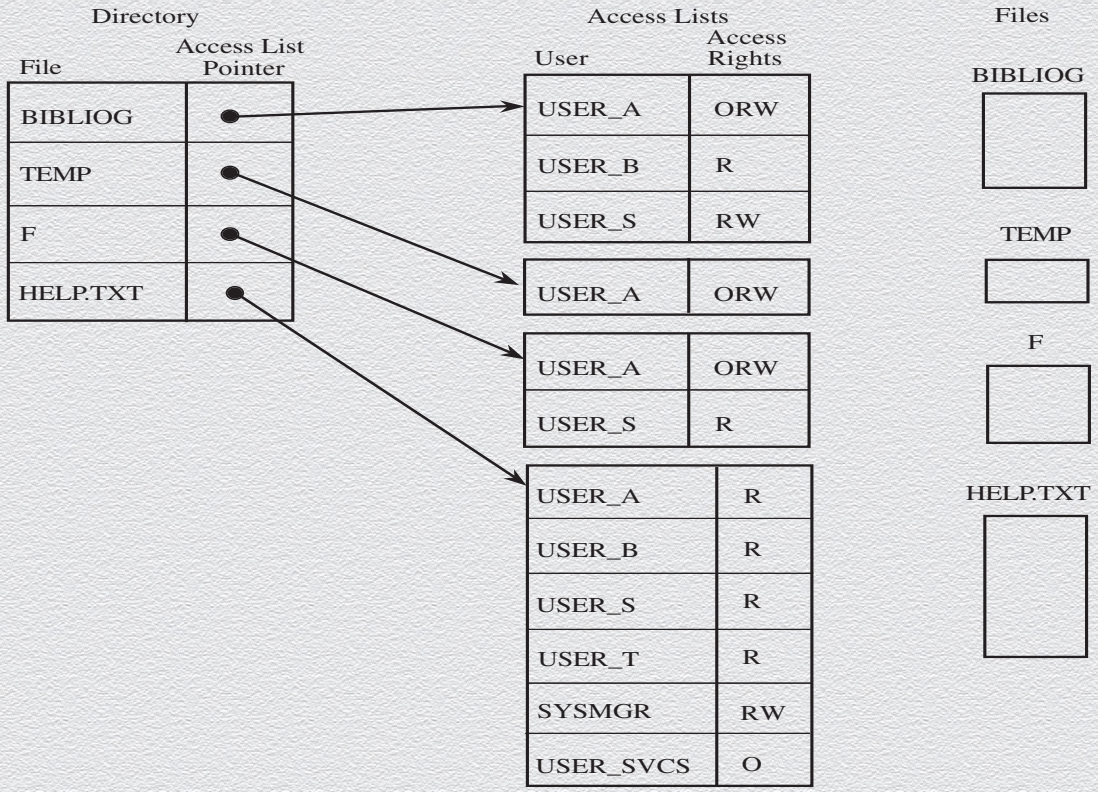
# Access control directory



# Access control matrix

| | BIBLIOG | TEMP | F | HELP.TXT | C_COMP | LINKER | SYS_CLOCK | PRINTER |
|---|---|---|---|---|---|---|---|---|
| **USER A** | ORW | ORW | ORW | R | X | X | R | W |
| **USER B** | R | - | - | R | X | X | R | W |
| **USER S** | RW | - | R | R | X | X | R | W |
| **USER T** | - | - | - | R | X | X | R | W |
| **SYS_MGR** | - | - | - | RW | OX | OX | ORW | O |
| **USER_SVCS** | - | - | - | O | X | X | R | W |

# Access control list



# Basic access control and information flow models

- Discretionary access control (DAC)
  - owner determines access rights
  - typically identity-based access control: access rights are assigned to users based on their identity
  - e.g., ACM
- Mandatory access control (MAC)
  - system enforce system-wide rules for access control
  - e.g., law allows a court to access driving records without the owners' permission

# DAC

- In DAC the user (e.g., owner of resources/files) is responsible for deciding how information is accessed
- Local access decisions of users might conflict with each other
- Basic terms
  - Access control matrix
  - Security policy (specifying who has the access rights to what)
  - Security mechanism (enforce security policies)

# DAC and MAC

- When is DAC insufficient?
  - when owner cannot be trusted for the discretion of the data and external protection of the data is necessary
  - e.g., DAC has the danger of right propagation
    - A can read X and write Y
    - B can read Y, but no access to X
    - A reads X, write the content of X to Y, B got access to X
- MAC
  - non-discretionary
  - labels are assigned to subjects and objects
  - owner has no special privileges
  - e.g., Bell-LaPadula, lattices models, SELinux by NSA

# Traditional models for MAC

- Bell-LaPadula (BLP)
  - About confidentiality
- Biba
  - About integrity with static/dynamic levels

# Bell-LaPadula security model

- The Bell-LaPadula (BLP) model is about information confidentiality
- It was developed to formalize the US Department of Defense multilevel security policy

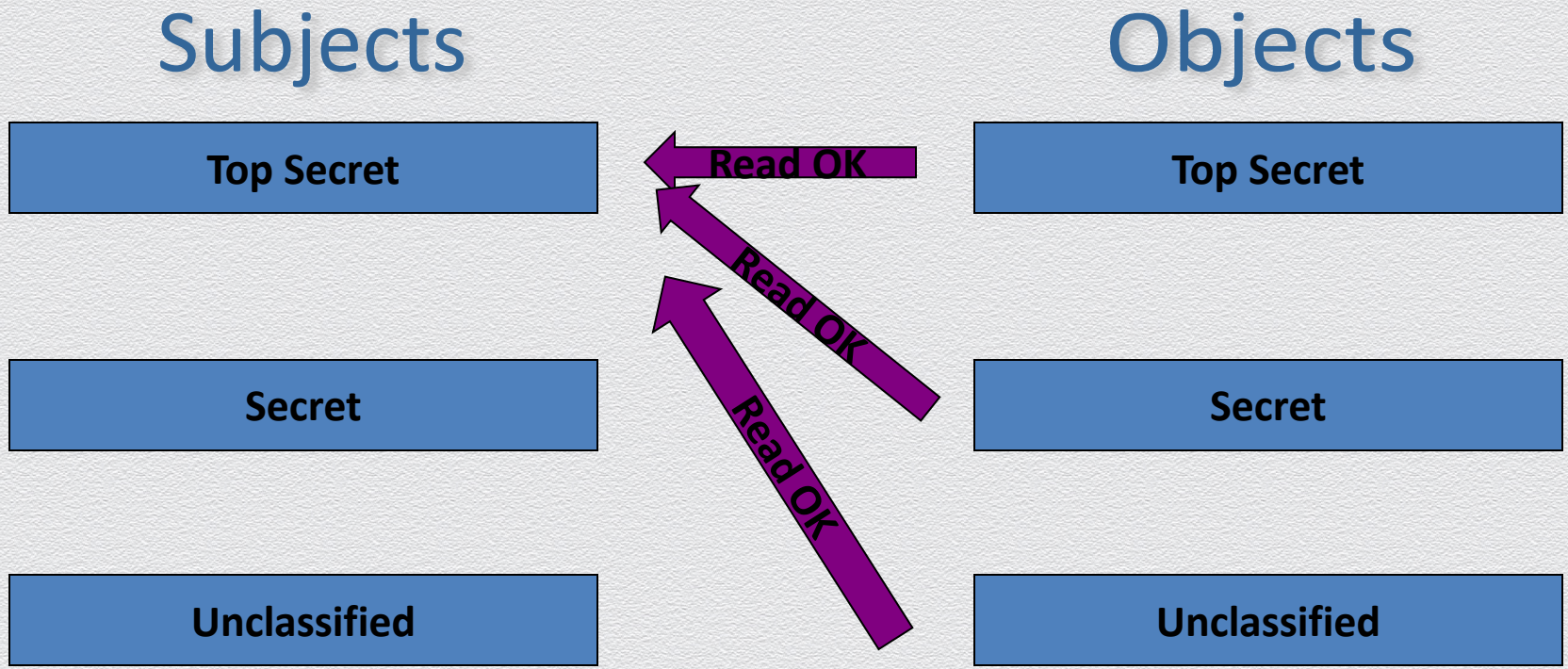
# Bell – LaPadula - details

- Each user subject and information object has a fixed security class – labels
- Use the notation **≤** to indicate **dominance**
- Simple Security (ss) property:

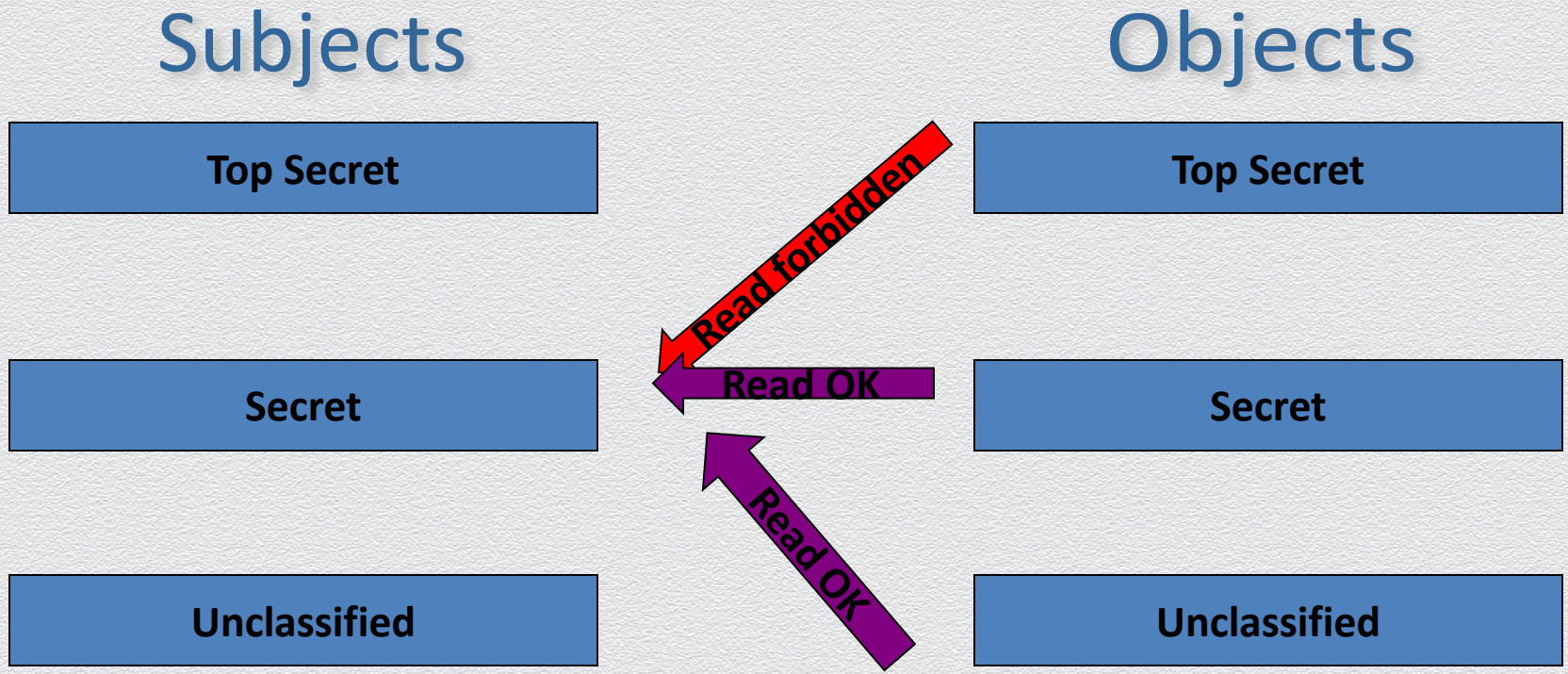
  **no read-up property**

  - a subject s has read access to an object o iff the class of the subject C(s) is greater than or equal to the class of the object C(o)
  - i.e. subjects s can read objects o iff C(o) ≤ C(s)

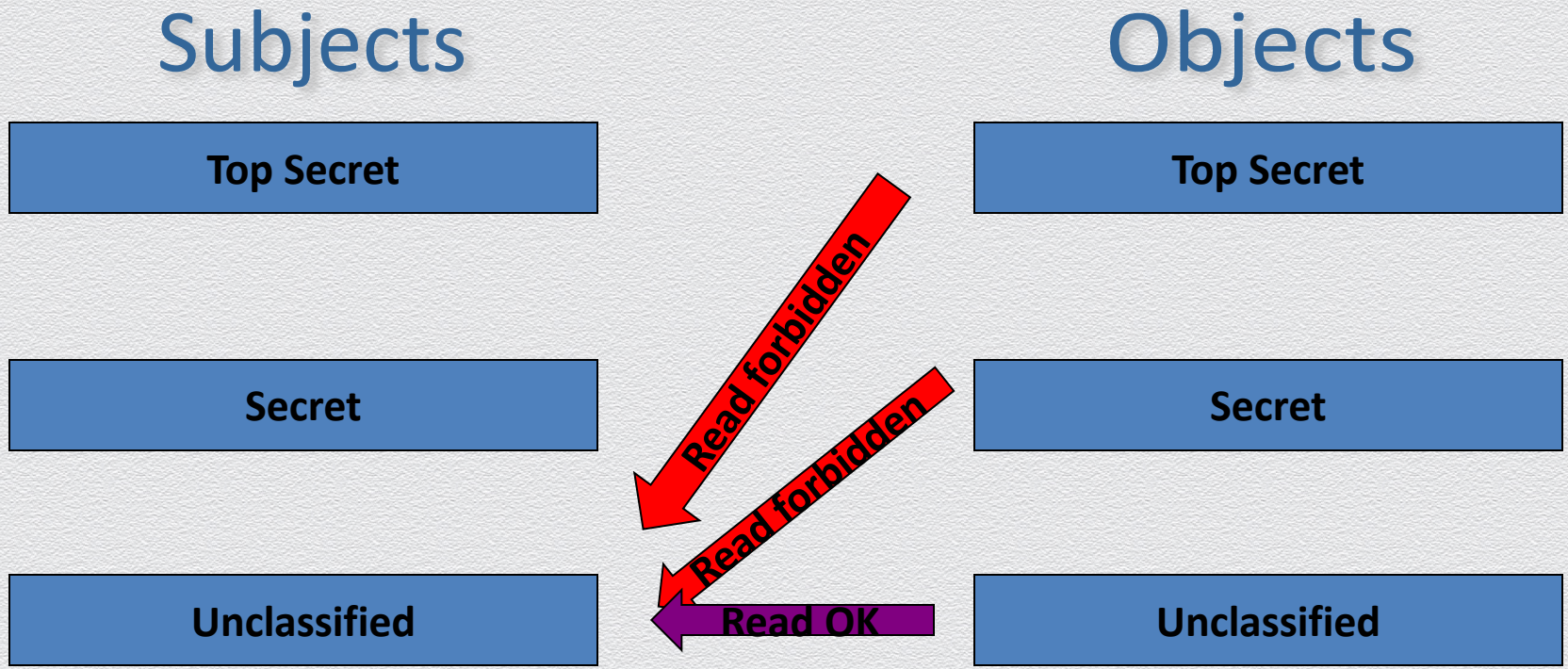
# Access control: Bell-LaPadula



# Access control: Bell-LaPadula



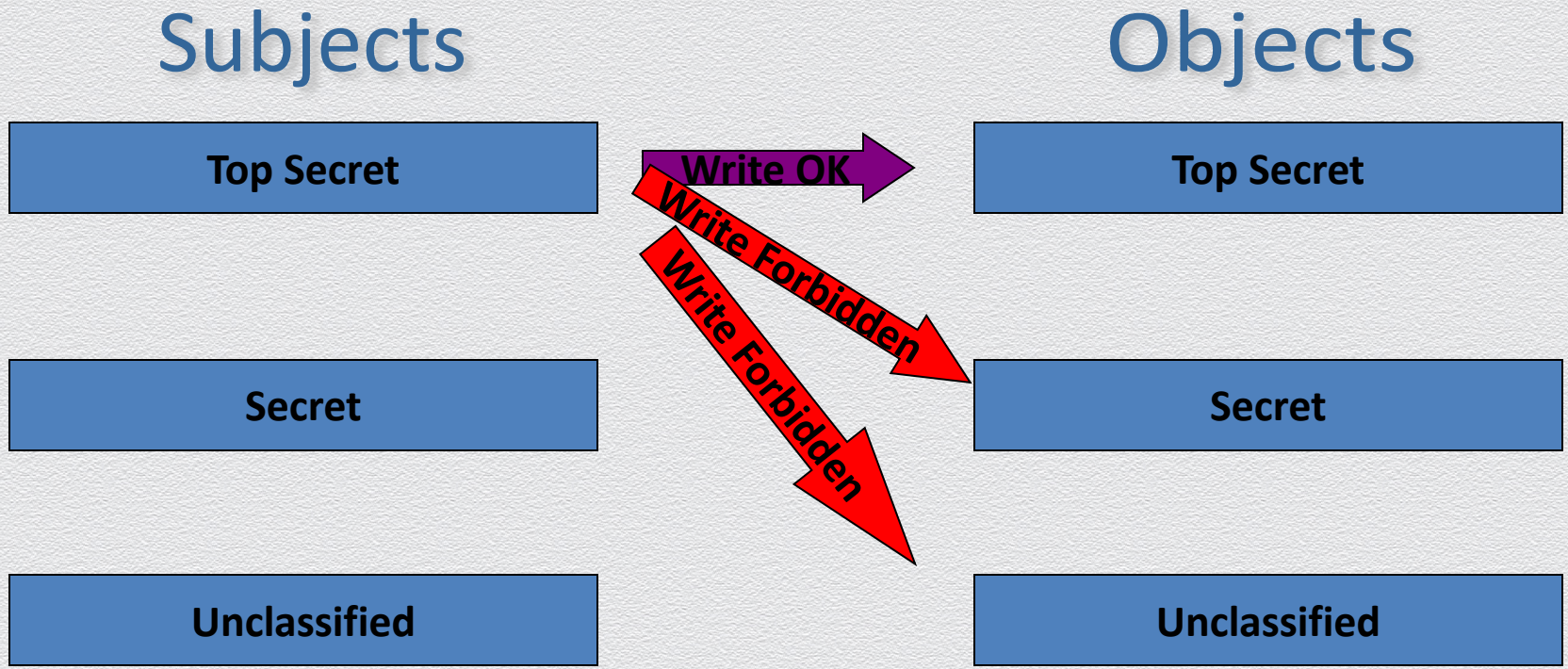
# Access control: Bell-LaPadula



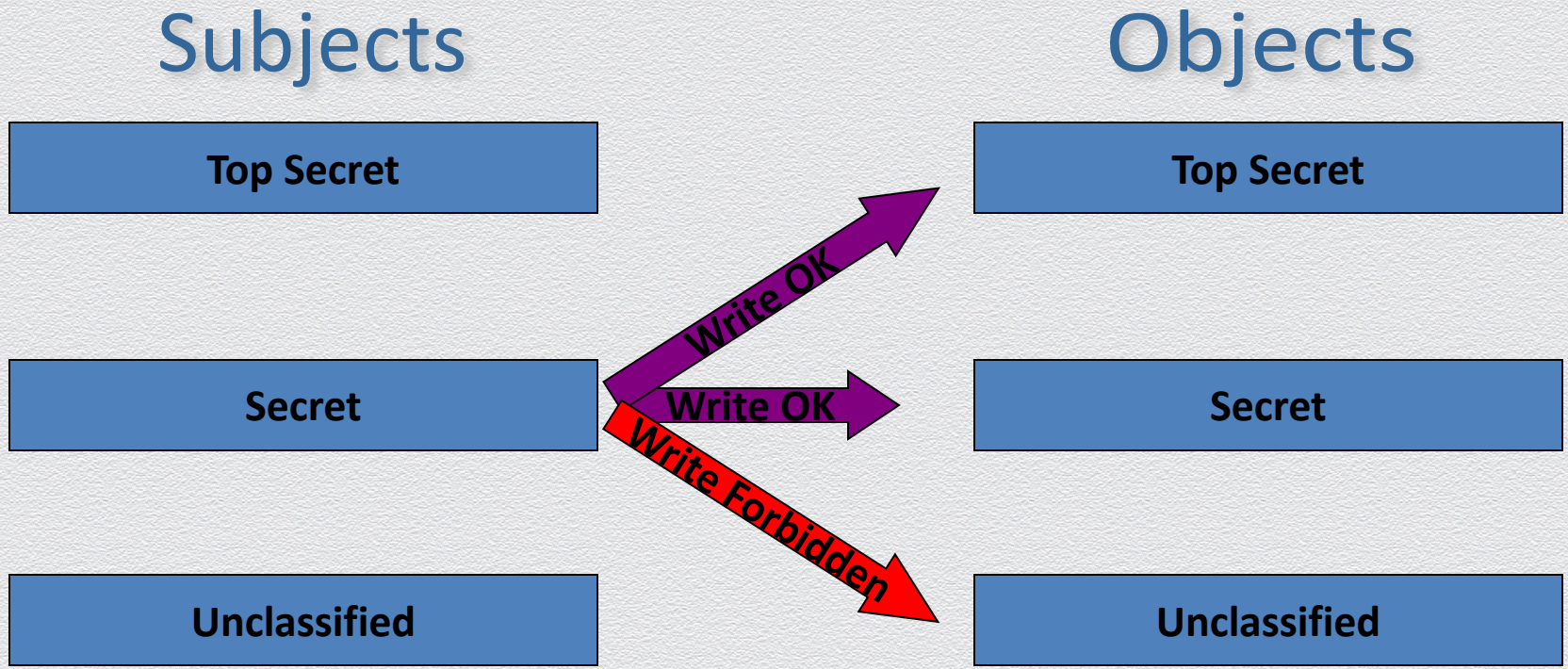
# Bell - LaPadula (2)

- * property (star):
  the no write-down property
  - A subject s can write to object *p* if C(s) ≤ C (p)

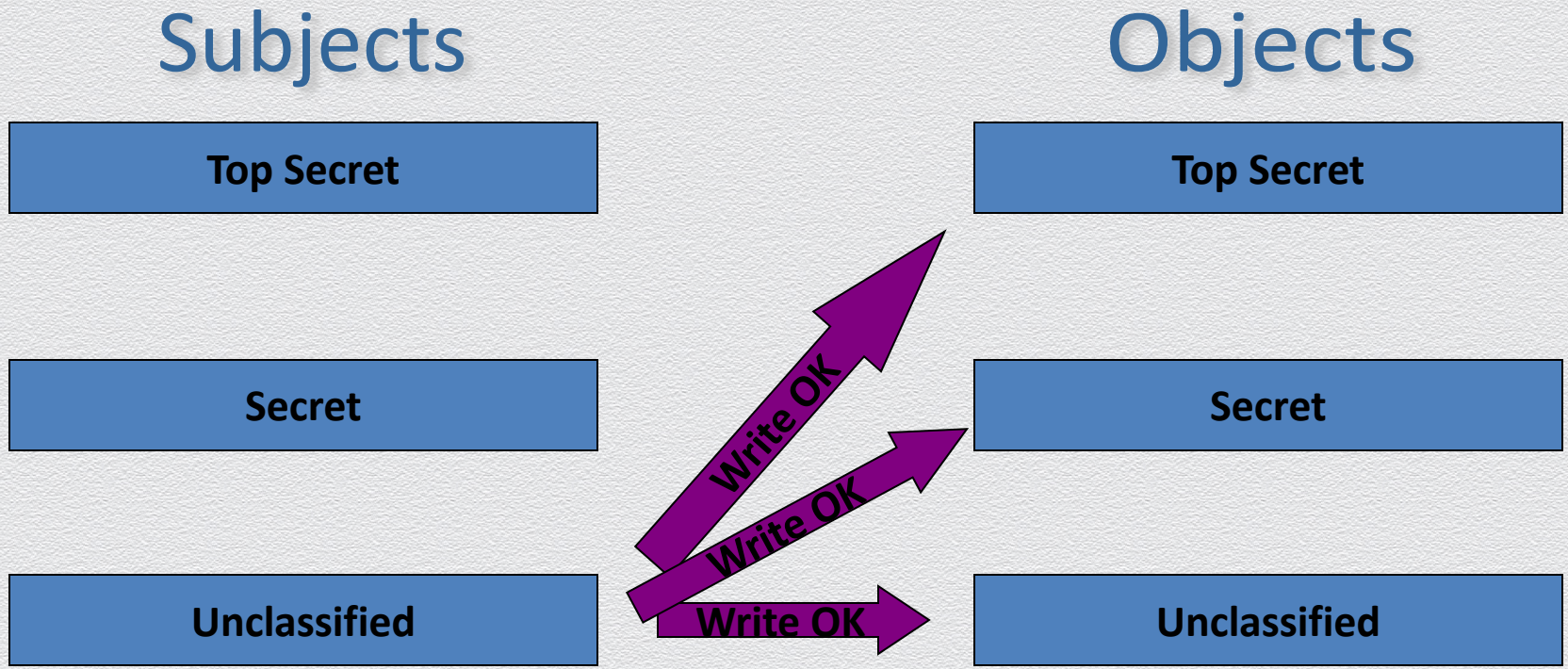
# Access control: Bell-LaPadula



# Access control: Bell-LaPadula



# Access control: Bell-LaPadula



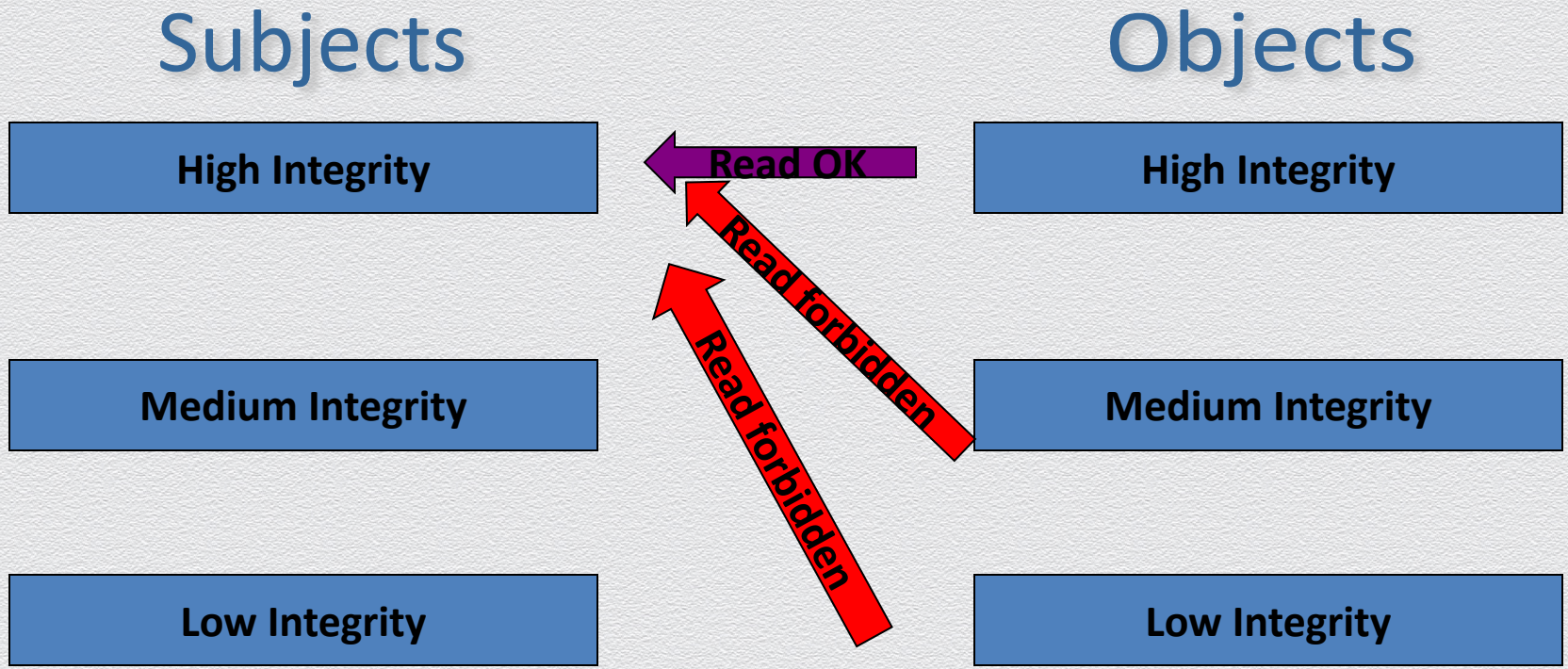
# Security models - Biba

- Based on the Cold War experiences, information *integrity* is also important, and the Biba model, complementary to Bell-LaPadula, is based on the flow of information where preserving integrity is critical.
- The “dual” of Bell-LaPadula

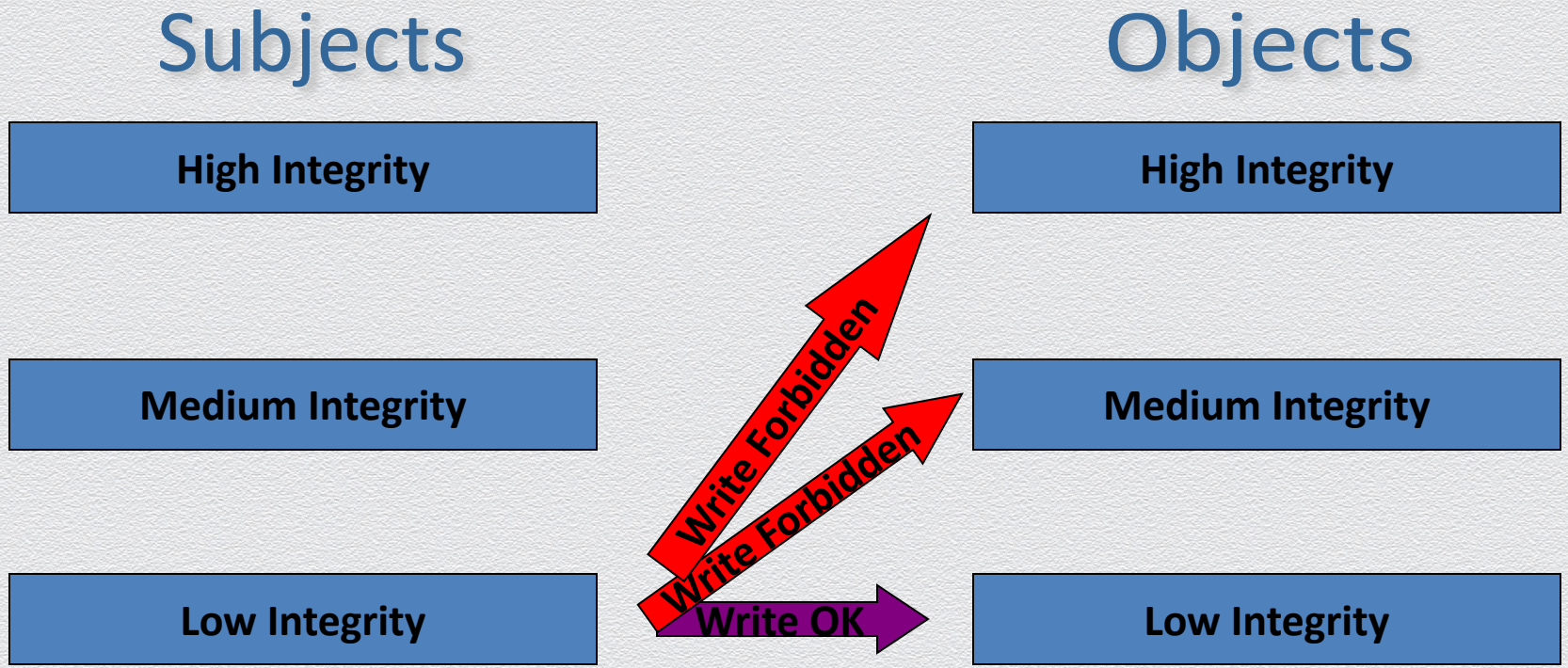
# Integrity control: Biba

- Designed to preserve integrity, not limit access
- Three fundamental concepts:
  - Simple Integrity Property – no read down
  - Star Integrity Property (*) – no write up
  - No execute up

# Integrity control: Biba



# Integrity control: Biba

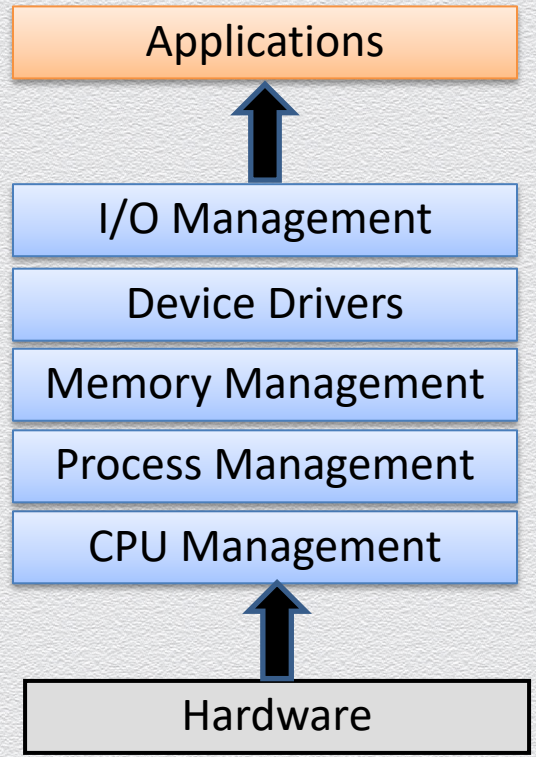


# 14.4 Operating system

# Operating System Layers

Many layers of abstraction

- Kernel  core of the OS, controls hardware, resource access
  - Various subsystems (memory management, networking, storage, …)
- Execution modes
  - user mode: access to resources mediated by the kernel
  - kernel mode: full and direct access to resources



# Processes

The kernel manages applications as processes (or threads)

Every process has

- Process ID (PID), virtual memory, effective user

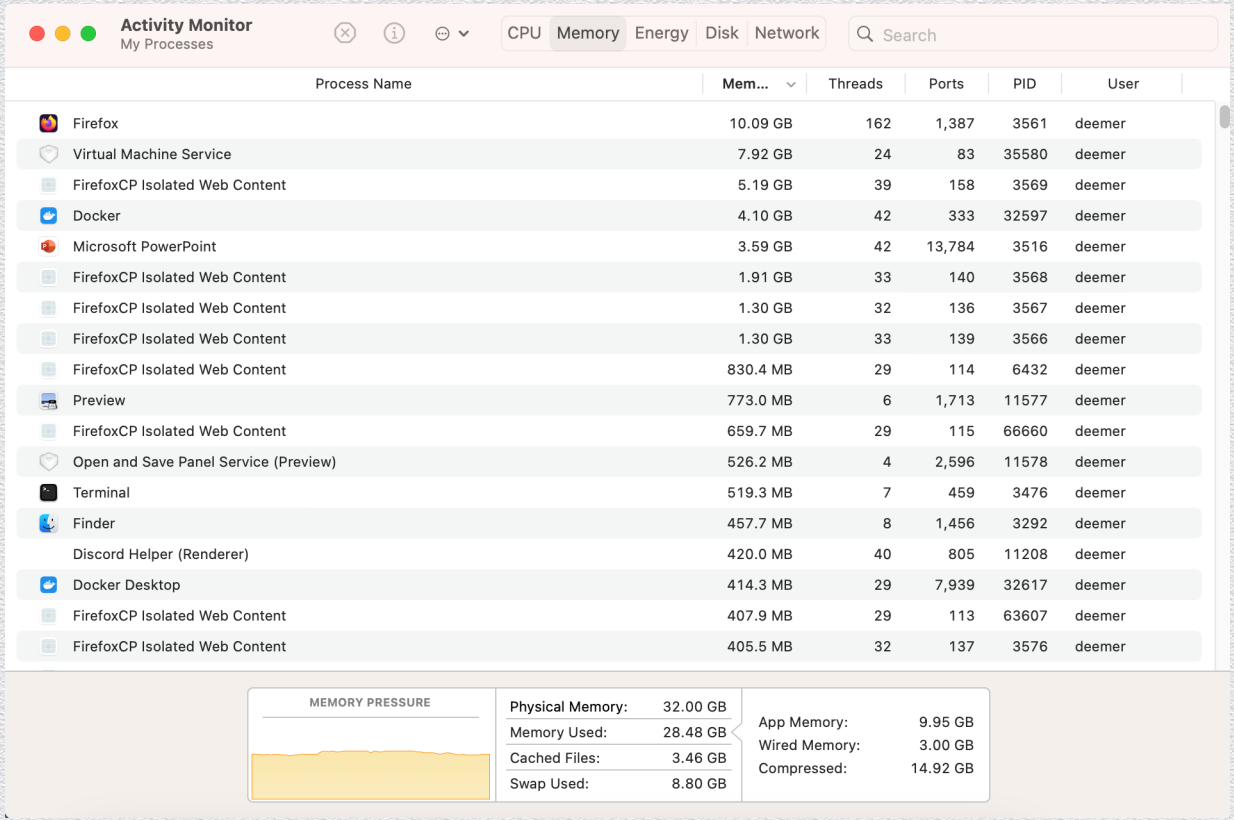
Kernel provides

- Separate address space from other process
- Time/resource sharing
- Access control

# Processes (cont.)

```
emplisi@ubuntu:~$ pstree
systemd─┬─ModemManager───2*[{ModemManager}]
        ├─NetworkManager───2*[{NetworkManager}]
        ├─VGAuthService
        ├─accounts-daemon───2*[{accounts-daemon}]
        ├─acpid
        ├─anacron───sh───run-parts───mlocate───flock───updated
        ├─avahi-daemon───avahi-daemon
        ├─bluetoothd
        ├─boltd───2*[{boltd}]
        ├─colord───2*[{colord}]
        ├─cron
        ├─cups-browsed───2*[{cups-browsed}]
        ├─cupsd
        ├─dbus-daemon
        ├─firefox─┬─3*[Web Content───18*[{Web Content}]]
        │         ├─Web Content───19*[{Web Content}]
        │         ├─WebExtensions───18*[{WebExtensions}]
        │         ├─file:// Content───18*[{file:// Content}]
        │         └─59*[{firefox}]
        ├─fwupd───4*[{fwupd}]
        ├─gdm3─┬─gdm-session-wor─┬─gdm-wayland-ses─┬─gnome-ses
```

# Processes (cont.)



# Processes (cont.)

- ps: displays snapshot of running processes
  - ps -ef : show all processes
  - ps -u <username>: show processes for a user
- top, htop: fancier list of processes
  - top -u <username>: filter by username
- kill  <pid>: terminates a process

```
metcalfe /u/lmeyerov % ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0  2005 ?        00:00:01 init [2]
root         2     1  0  2005 ?        00:00:00 [ksoftirqd/0]
root         3     1  0  2005 ?        00:00:00 [events/0]
root         4     3  0  2005 ?        00:00:00 [khelper]
root        31     3  0  2005 ?        00:00:00 [kblockd/0]
root       104     3  0  2005 ?        00:00:01 [pdflush]
root       105     3  0  2005 ?        00:00:00 [pdflush]
root       107     3  0  2005 ?        00:00:00 [aio/0]
root       106     1  0  2005 ?        00:00:03 [kswapd0]
root       694     1  0  2005 ?        00:00:00 [kseriod]
root       745     3  0  2005 ?        00:00:00 [ata/0]
root       750     1  0  2005 ?        00:00:00 [scsi_eh_2]
root       751     1  0  2005 ?        00:00:00 [scsi_eh_3]
root       769     1  0  2005 ?        00:00:02 [kjournald]
root      1157     1  0  2005 ?        00:00:00 [khubd]
root      1263     1  0  2005 ?        00:00:00 [kjournald]
root      1264     1  0  2005 ?        00:00:00 [kjournald]
root      1265     1  0  2005 ?        00:00:00 [kjournald]
root      1266     1  0  2005 ?        00:00:06 [kjournald]
root      1521     1  0  2005 ?        00:00:00 [khpsbpkt]
root      1584     1  0  2005 ?        00:00:00 [knodemgrd_0]
root      2813     1  0  2005 ?        00:00:00 dhclient -e -pf /var/run/dhclien
```

```
top - 14:23:25 up 41 days, 20:45, 10 users,  load average: 0,02, 0,12, 0,19
Tasks: 142 total,   1 running, 140 sleeping,   0 stopped,   1 zombie
Cpu(s):  3,7% us,  0,7% sy,  0,0% ni, 95,7% id,  0,0% wa,  0,0% hi,  0,0% si
Mem:   1034436k total,   910324k used,   124112k free,   183132k buffers
Swap:  1048816k total,     4892k used,  1043924k free,   222260k cached
Which user (blank for all):
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 4104 root       5 -10  180m  50m 6724 S  1,0  5,0   2:42,53 XFree86
13091 lmeyerov  15   0  132m  72m  26m S  1,0  7,2  17:22,59 mozilla-bin
21114 lmeyerov  15   0 36876  17m  13m S  1,0  1,7   0:01,38 ksnapshot
24555 lmeyerov  15   0 21128 7508 5460 S  0,7  0,7   0:00,79 artsd
21270 lmeyerov  17   0  2036 1116  860 R  0,7  0,1   0:00,02 top
    1 root      16   0  1504  528  468 S  0,0  0,1   0:01,32 init
    2 root      35  19     0    0    0 S  0,0  0,0   0:00,28 ksoftirqd/0
    3 root       5 -10     0    0    0 S  0,0  0,0   0:00,85 events/0
    4 root       6 -10     0    0    0 S  0,0  0,0   0:00,00 khelper
   31 root       5 -10     0    0    0 S  0,0  0,0   0:00,81 kblockd/0
  104 root      15   0     0    0    0 S  0,0  0,0   0:01,81 pdflush
  105 root      15   0     0    0    0 S  0,0  0,0   0:00,32 pdflush
  107 root      15 -10     0    0    0 S  0,0  0,0   0:00,00 aio/0
  106 root      15   0     0    0    0 S  0,0  0,0   0:03,65 kswapd0
  694 root      25   0     0    0    0 S  0,0  0,0   0:00,00 kseriod
  745 root       6 -10     0    0    0 S  0,0  0,0   0:00,00 ata/0
  750 root      18   0     0    0    0 S  0,0  0,0   0:00,00 scsi_eh_2
```

# Processes management

- Each process has a context, which includes the user, parent process, and address space

- Kernel enforces policies to decide which resources each processes can use

# System calls (syscalls)

- Primary way processes interact with kernel
- OS provides a “library” of syscalls for nearly all OS functions
  - Files:  read, write, open, close, chmod, …
  - Process management:  fork, clone, kill, …
  - Networking:  socket, bind, connect

On syscall, process “yields” to kernel, executes in privileged kernel mode

# System services (daemons)

- Background process that performs common tasks
- Started at boot time
- Could run with higher permissions than users

Typical services:

- Remote SSH connections
- Web servers
- Logging

# Identification and Authentication (recap)

- A subject should provide a unique identifier
- Authentication is the act of confirming the truth of an attribute of a datum or entity
- There are three authentication factors:
  - Knowledge: Something you know
  - Ownership: Something you have
  - Inherence: Something you are

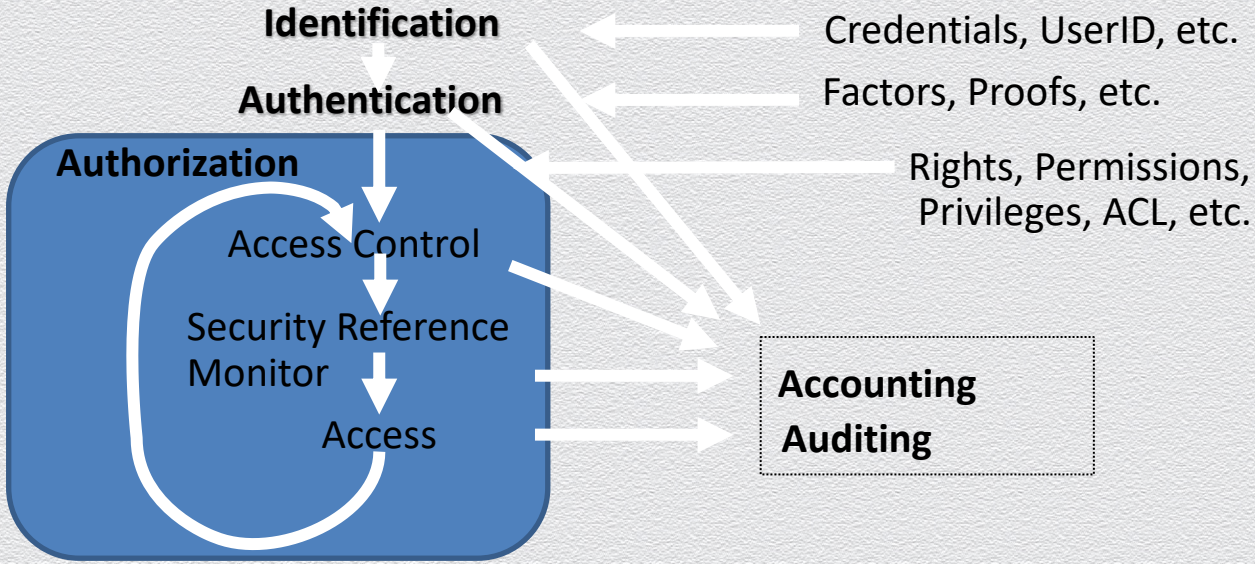
# Authorization

- Once a  subject is Authenticated, access should be authorized
- Authorization is the function of specifying access rights to resources (access control)
- More formally, "to authorize" is to define access policy: permissions, rights, etc.

# AAA

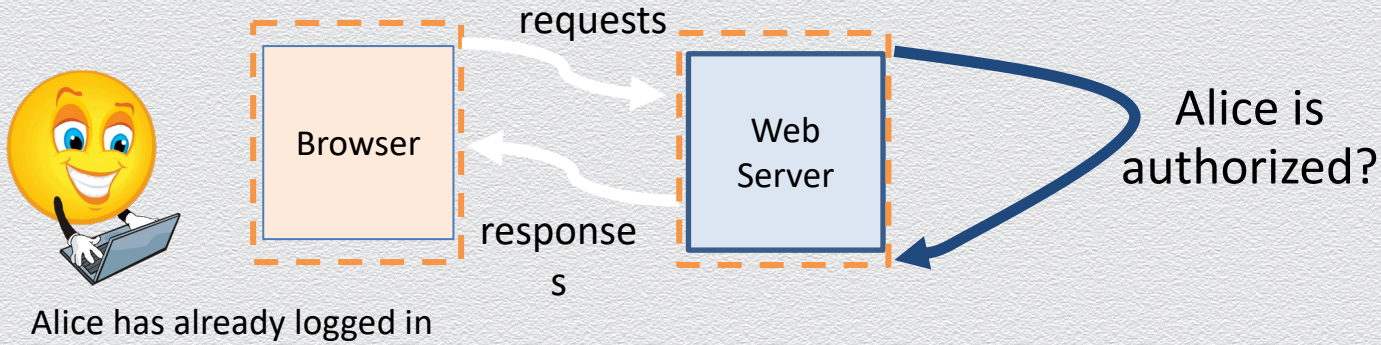
Identification, **Authentication**, **Authorization**, **Accounting**, Auditing

- AAA Working Group, IETF



# Authorization on the Web

- Alice logs in (i.e. authenticates); the web server is now aware of who is logged in
- Alice attempts to access a course
- The application checks to see if Alice has the authorization for the course…
  - If so, Alices receives the requested information; If not, Alice has a denied access response
- Authorization could be just for reading or writing or execute  (more in the future lectures)



# AAA: Authorization

Authorization:  how to specify access rights to resources

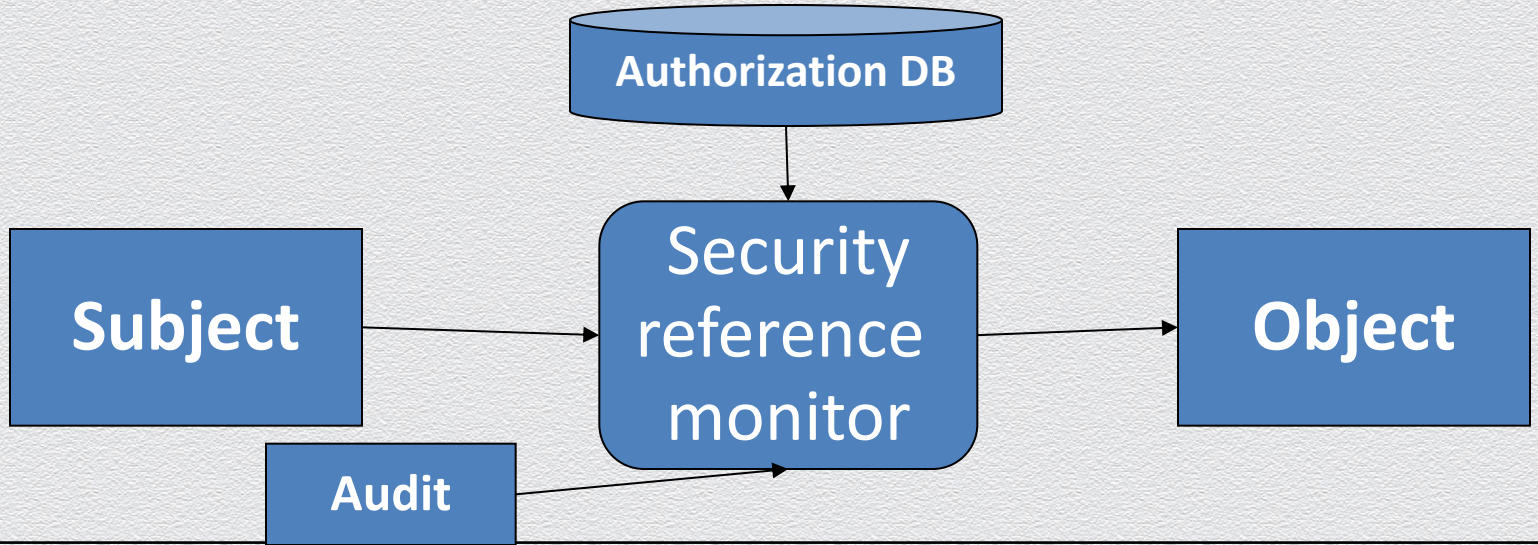
- To authorize => to define access policy

# Users

- Each process is associated with a user
- Specific users can have more privileges than regular users
  - Install or remove programs
  - Change rights of other users
  - Modify the configuration of the system
- Unix: root is a “super-user” with no restrictions

# Security Reference Monitor (SRM) (recap)

- Checks for proper authorization before granting access to objects.
- Object manager asks SRM if a Subject has the proper rights to execute a certain type of action on an Object.
- Implements auditing functions to keep track of attempts to access an object.



# Discretionary Access Control (DAC)

- Users can protect what they own
  - The owner may grant access to others
  - The owner may define the type of access (read/write/execute) given to others
- DAC is the standard model used in operating systems
- Mandatory Access Control (MAC)
  - Multiple levels of security for users and documents (i.e. confidential, restricted, secret, top secret)
  - A user can create documents with just his level of security

# General Principles

- Files and folders are managed by the operating system
- Applications, including shells, access files through an API
- Access control entry (ACE)
  - Allow/deny a certain type of access to a file/folder by user/group
- Access control list (ACL)
  - Collection of ACEs for a file/folder
- A file handle provides an opaque identifier for a file/folder
- File operations
  - Open file: returns file handle
  - Read/write/execute file
  - Close file: invalidates file handle
- Hierarchical file organization
  - Tree (Windows)
  - DAG (Linux)

# Access Control Entries and Lists

- An Access Control List (ACL) for a resource (e.g., a file or folder) is a sorted list of zero or more Access Control Entries (ACEs)
- An ACE refers specifies that a certain set of accesses (e.g., read, execute and write) to the resources is allowed or denied for a user or group
- Examples of ACEs for folder “Bob’s CS166 Grades”
  - Bob; Read; Allow
  - TAs; Read; Allow
  - TWD; Read, Write; Allow
  - Bob; Write; Deny
  - TAs; Write; Allow

# Closed vs. Open Policy

## Closed policy

- Also called “default secure”
- Give Tom read access to “foo”
- Give Bob r/w access to “bar
- Tom: I would like to read “foo”
  - Access allowed
- Tom: I would like to read “bar”
  - Access denied

## Open Policy

- Deny Tom read access to “foo”
- Deny Bob r/w access to “bar”
- Tom: I would like to read “foo”
  - Access denied
- Tom: I would like to read “bar”
  - Access allowes

# Closed Policy with Negative Authorizations and Deny Priority

Give Tom r/w access to “bar”

Deny Tom write access to “bar”

Tom: I would like to read “bar”

Access allowed

Tom: I would like to write “bar”

Access denied

Policy is used by Windows to manage access control to the file system

# Role-Based Access Control

- Within an organization roles are created for various job functions
- The permissions to perform certain operations are assigned to specific roles
- Users are assigned particular role, with which they acquire the computer authorizations
- Users are not assigned permissions directly, but only acquire them through their role

U.S. Navy image in the public domain.