Information Security

Chapter 2: Software & OS Security

Nguyễn Đăng Quang

Goals

01

Understand the important role an operating system plays in Computer Security,

02

Understand Trusted Computing Base (TCB),

03

Learn about the need for hardware support for isolating OS from untrusted user/application code,

04

Understand key trusted computing base concepts

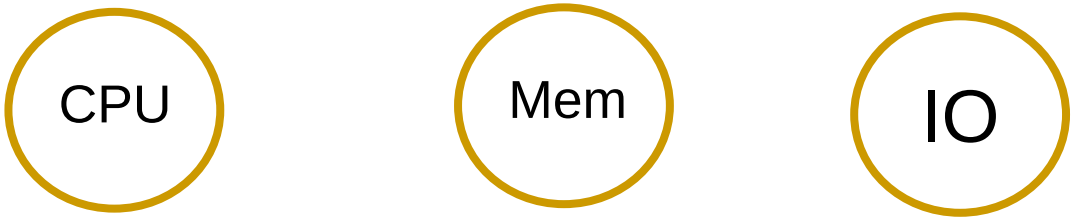
Top 13 Oses by total number of Distinct Vulnerabilities (cvedetails.com)

No.	OS name	2019	2021
1	Android	414	3782
2	Debian linux	360	5118
3	Windows server 2016	357	2133
4	Windows 10	357	2361
5	Windows server 2019	351	1567
6	Windows 7	250	1901
7	Windows server 2008	248	2030
8	Windows server 2012	246	1799
9	Windows 8.1	242	1695
10	Ubuntu Linux	190	2986
11	Linux kernel	170	2703
12	Iphone OS	156	2299
13	Mac OS X	117	2759

Operating System (OS)

Applications

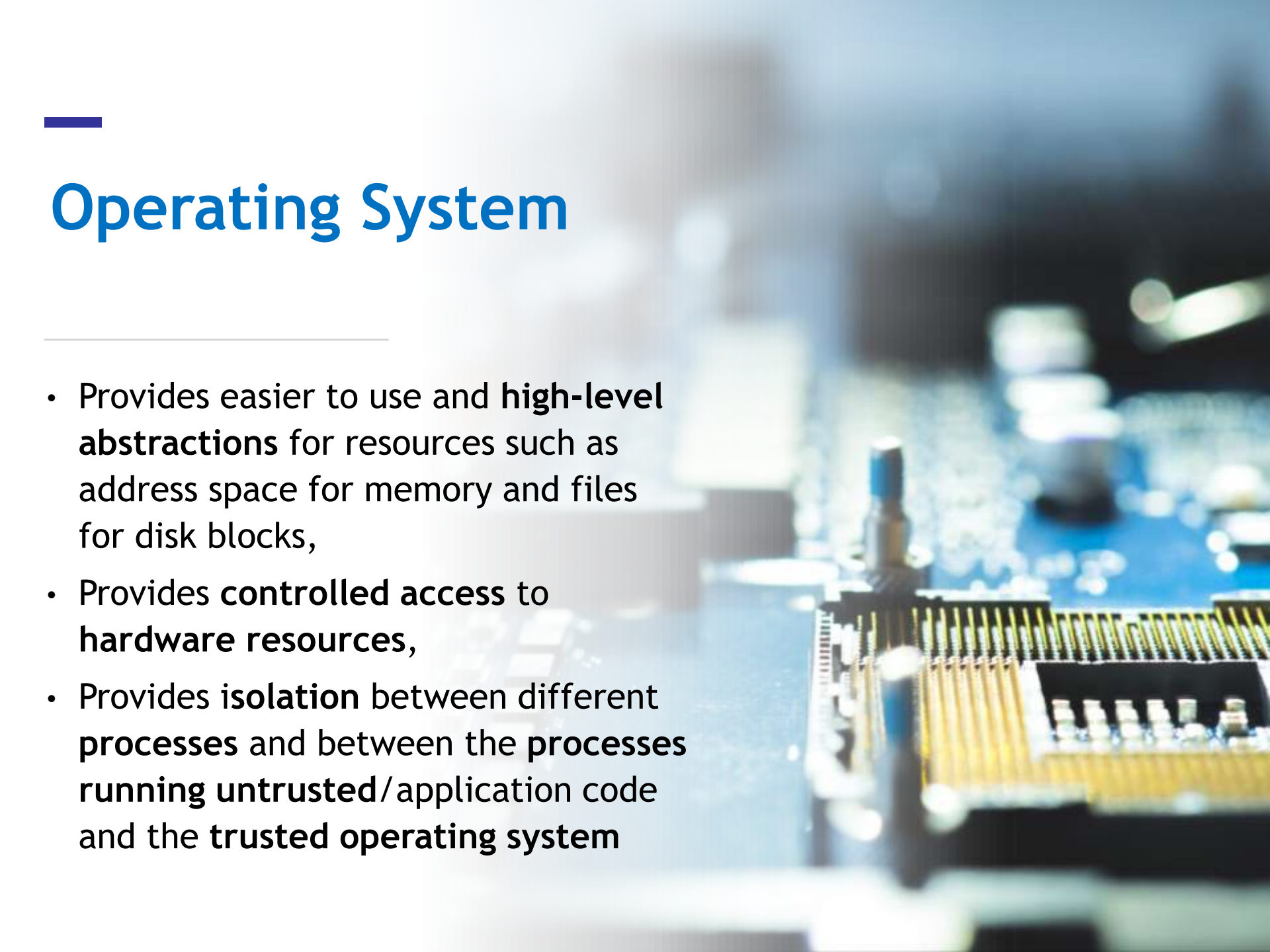
OS:
Windows/Linux/Android



Hardware

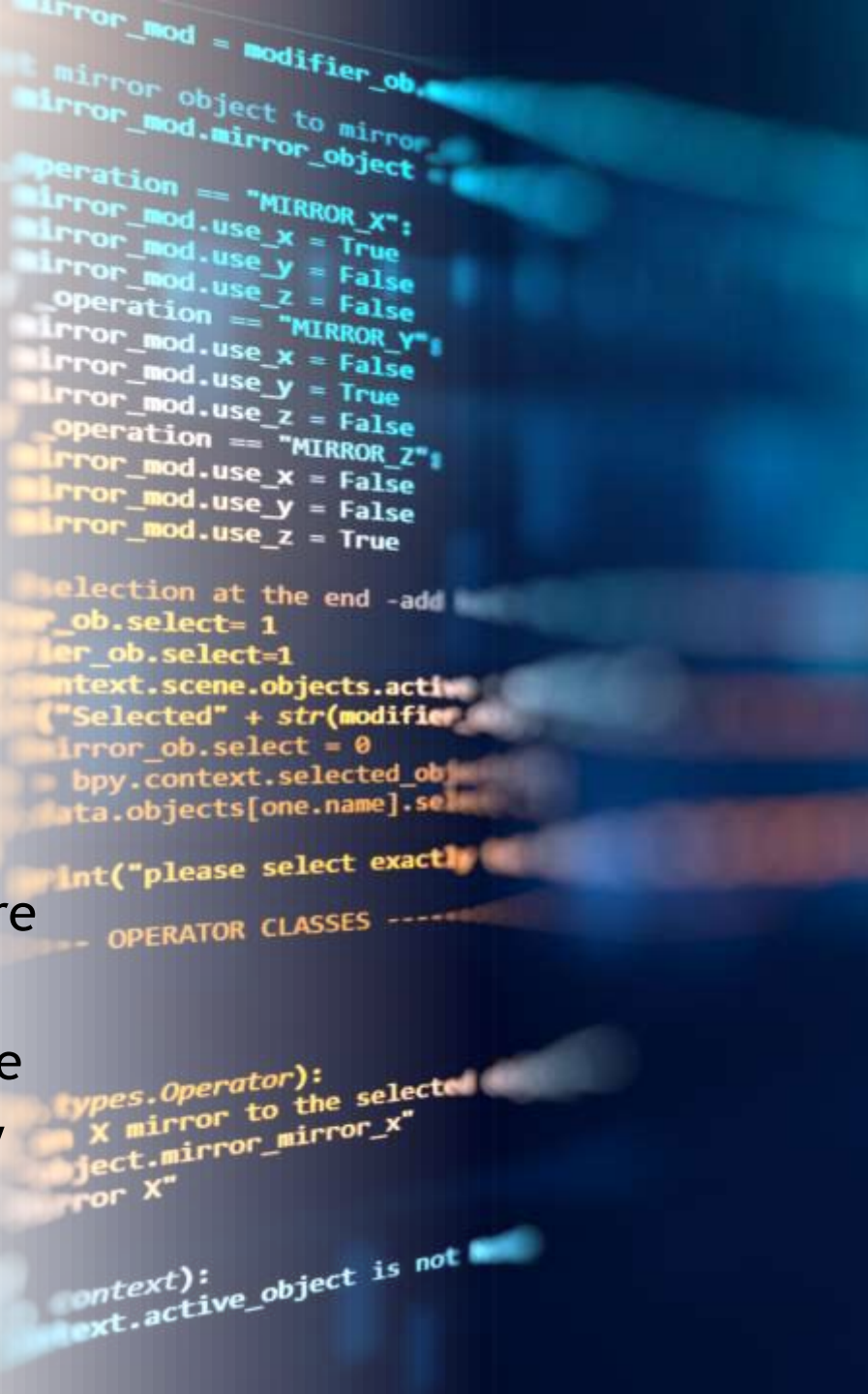
Operating System

- Provides easier to use and **high-level abstractions** for resources such as address space for memory and files for disk blocks,
- Provides **controlled access** to **hardware resources**,
- Provides i**solation** between different **processes** and between the **processes running untrusted**/application code and the **trusted operating system**



Trusted Computing Base

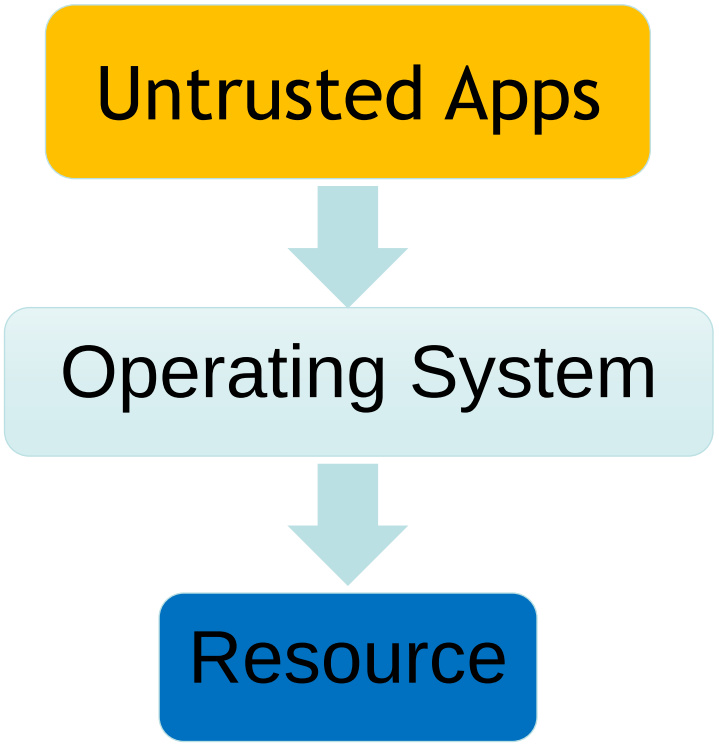
- is a combination of hardware, software, and controls that work together to form a trusted base to enforce your security policy.
- A given piece of hardware or software is a part of the TCB if and only if it has been designed to be a part of the mechanism that provides its security to the computer system.



Need for Trusting an Operating System

- Is a Trusted Computing Base (TCB)
- TCB requirements:
  1. Complete mediation
  2. Tamper-proof

  and
  3. Correctness



Complete mediation examples

Example 1:

File access

Example 2:

DNS caches mapping

OS and Resource Protection

OS controls access to protected resources

Must establish the **source of a request** for a resource (authentication),

Access control check or **authorization**

How can we trust an OS ?

How does OS meet the requirements of a TCB ?

Tamper-proof/Isolation

1. **Hardware support** for memory protection.
2. Processor **execution modes** (system/user modes, execution rings)
3. **Privileged instructions** which can only be executed in system mode.

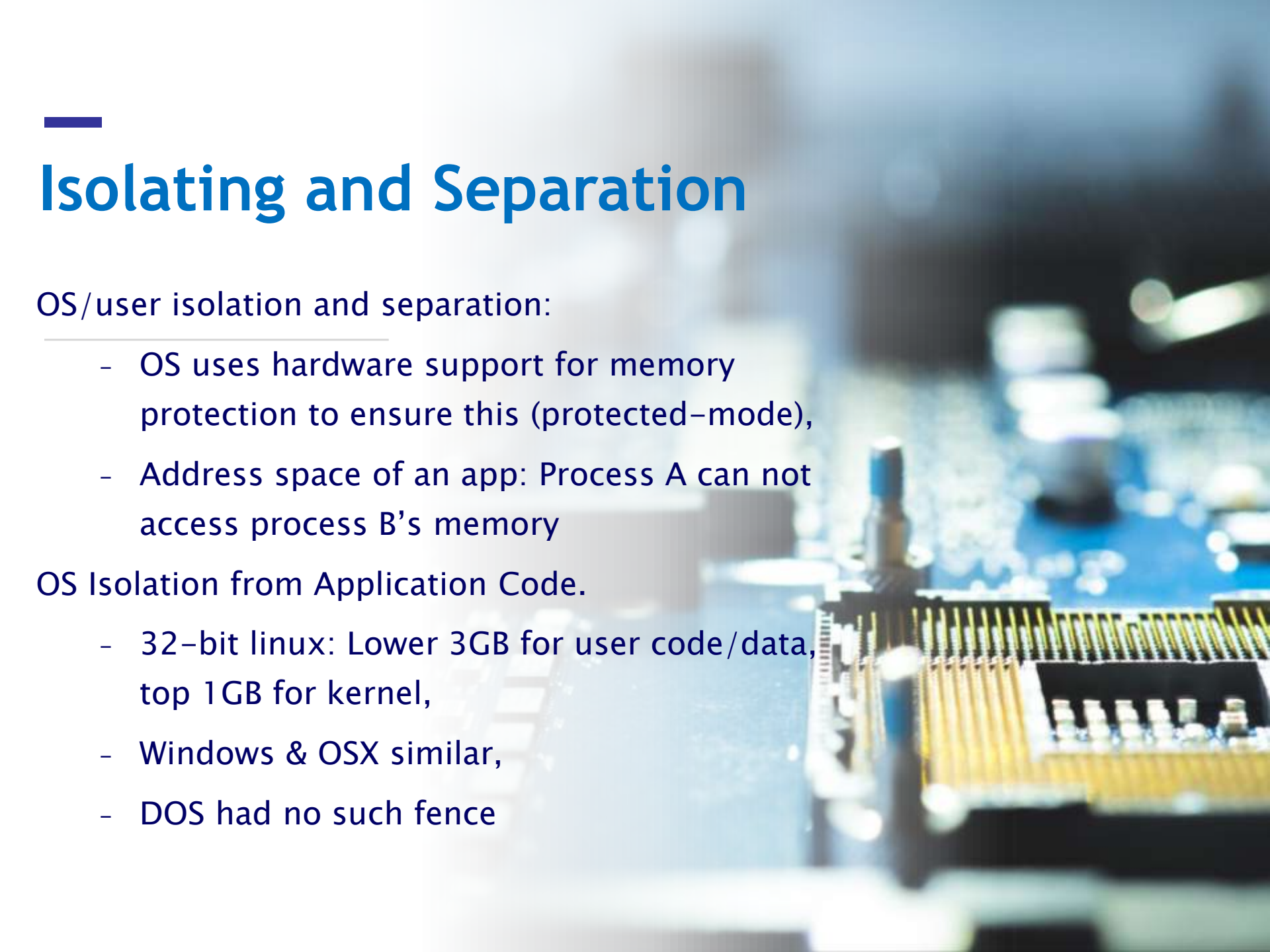
Isolating and Separation

OS/user isolation and separation:

- OS uses hardware support for memory protection to ensure this (protected-mode),
- Address space of an app: Process A can not access process B's memory

OS Isolation from Application Code.

- 32-bit linux: Lower 3GB for user code/data, top 1GB for kernel,
- Windows & OSX similar,
- DOS had no such fence



How can we trust an OS?

How do we meet the requirements of a TCB ?

Complete mediation

- Not be able to bypass the operating system and go directly to a protected resource.
- Make sure that no protected resource, whether it's a memory page or a file, could be accessed without going through the Trusted Computing Base (kernel)

How can we trust an OS?

How do we meet the requirements of a TCB ?

Complete mediation

- User code cannot access OS part of address space w/o changing to system mode.
- User code cannot access physical resource because they require privileged instructions which can only be executed in system mode.

How can we trust an OS?

How do we meet the requirements of a TCB ?

Complete mediation

- OS virtualizes physical resources and provide an API for virtualized resources
  - File abstraction: virtual resource for storing persistence data. OS does disk IO, interact through disk controller; user can only ask for access to files opening/reading/ writing to them. User don’t have ways to talk directly to the disk controller.
  - File buffers are used to translate virtual resource to physical one which can only be done by OS (complete mediation)

How can we trust an OS?

How do we meet the requirements of a TCB ?

Correctness

- Compromise of OS (TCB) means an attacker has access to everything ➔ Getting TCB right is extremely important.
- Secure coding is really important when writing OS which typically written in languages that are not type safe.

TCB Design Principles

- Least privilege for users & programs
- Economy
  - Keep trusted code small as possible, easier to analyze & test
- Open design
  - Security by obscurity does not work
- Complete mediation
  - Every access check, attempt to bypass must be be prevented
- Fail-safe default
  - Default deny
- Ease of use

Support key security features of TCB

- Object reuse protection
- Disk blocks, memory frames reused
- Process can allocate disk or memory, then look to see what left behind
- Trusted OS should zero out objects before reuse
- Secure file deletion: overwrite with varying patterns of zeros and ones
- Secure disk destruction: physical destruction

Support key security features of TCB

- Complete mediation of accesses
  - Trusted path from user to secure system
  - Prevents programs from spoofing interface of secure components
  - Prevents programs from tapping paths (e.g. keylogger)
- Audit log showing object access
  - Detect unusual use of the system

Some Linux OS known vulnerabilities

Environment variables

env command, environ variable, system() and execve()

Set-uid

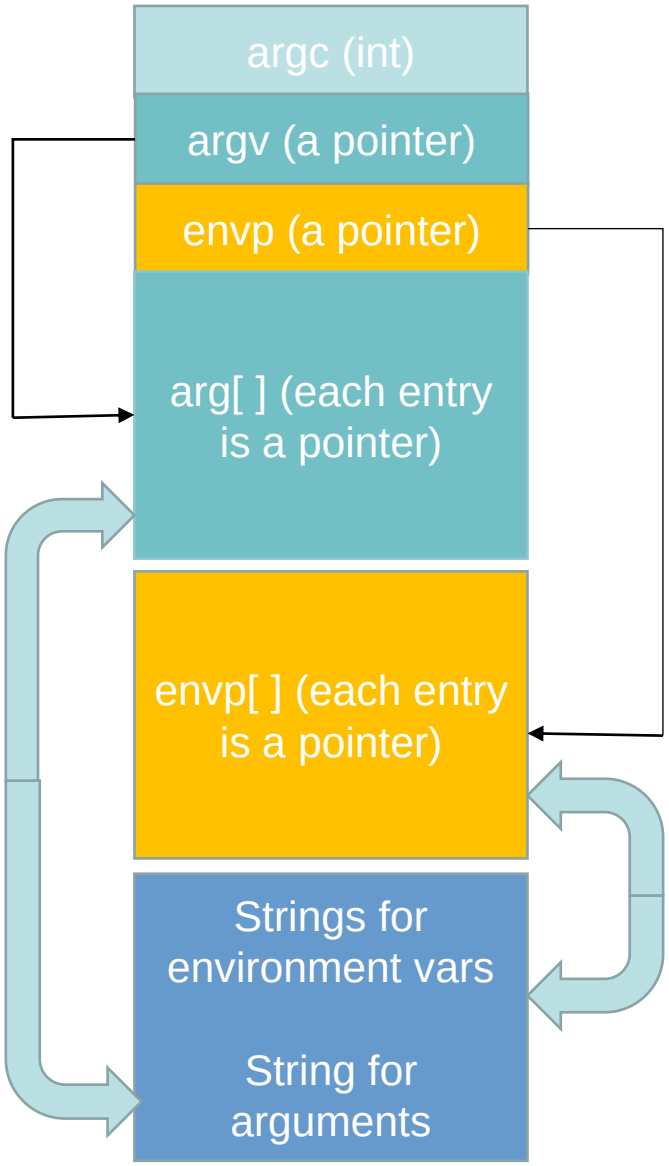
Privilege Escalation

Modify Path

Environment variables

- Are set of dynamic name-value pairs stored inside a process, they affects the way the process behaves
- To print out Environment variables: env/printenv
- To access Environment variables: envp[]/environ

Memory location



Shell vars and Environment vars

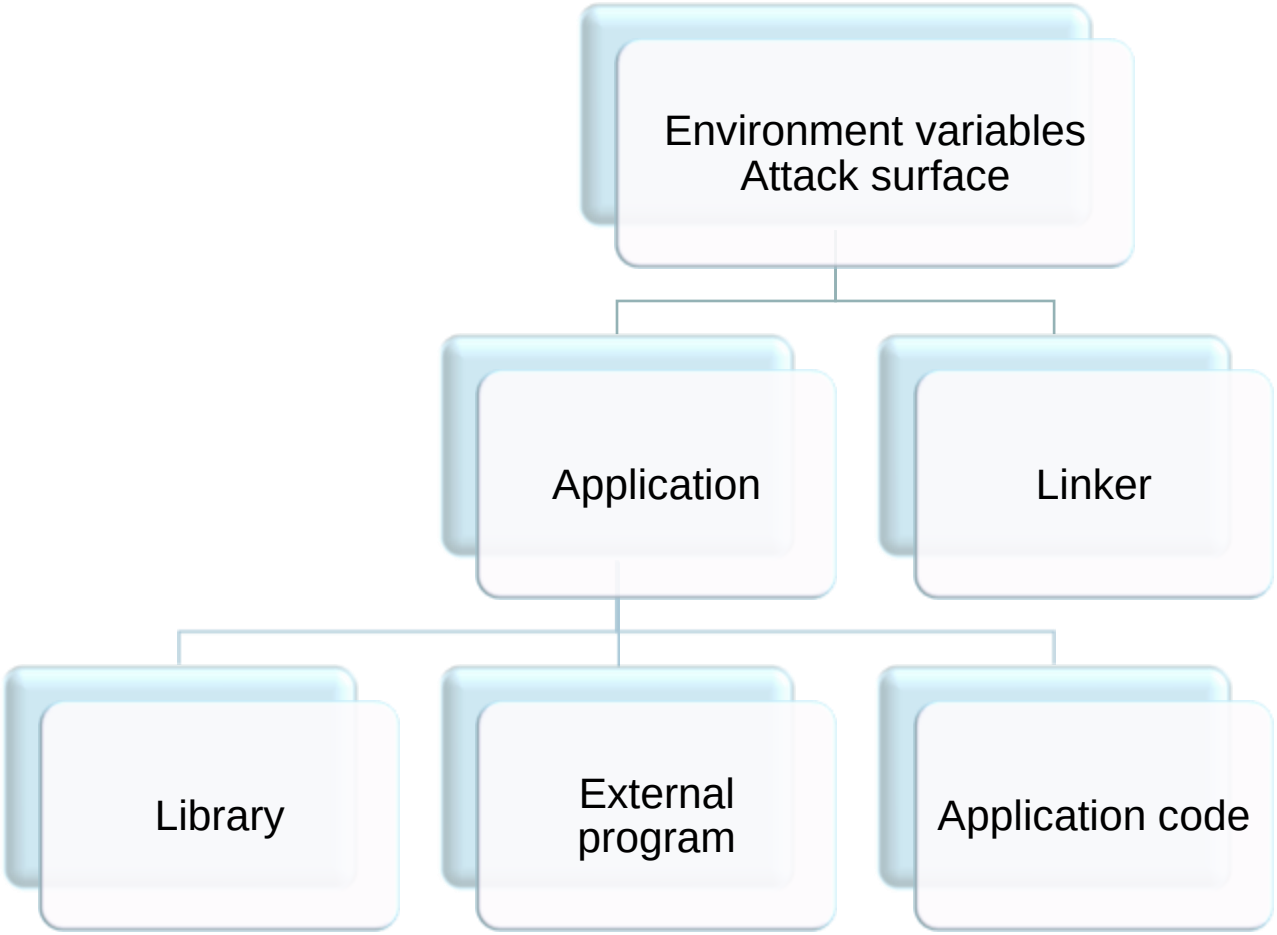
- Shell variables are internal variables maintained by a shell program.
- Affect the shell's behaviors
- Can be used in shell scripts

```
1   $ FOO=/bin/sh    ; declare a new shell variable
2   $ echo $FOO      ; print out
3   /bin/sh
4   $ unset FOO      ; delete it
5   $ echo $FOO
6
7   $
```

They are different

- When a shell program starts, it defines a shell variable for each of the environment vars using the same names and copying their values. Whatever changes made to the shell variable will not affect the environment variable of the same name, and vice versa.
- When we type a program name in the shell prompt, shell will start the program in a child process. **Only shell vars marked for export will be copied to the child process**

Attack surface caused by environment variables



Example of attack via linker

- A linker is used to find the external library funtion used by a program.
- Linkers in most Oses use environment variables to find where the libraires are → opportunity for attacker to attack.