Cloud Computing Concepts

CS3132

Dr. Anand Kumar Mishra

NIIT University

API and ABI

Application Programming Interface

- The highest level of abstraction is represented by the application programming interface (API), which interfaces applications to libraries and/or the underlying operating system
- For any operation to be performed in the application level API, ABI and ISA are responsible for making it happen

API - Application Programming Interface

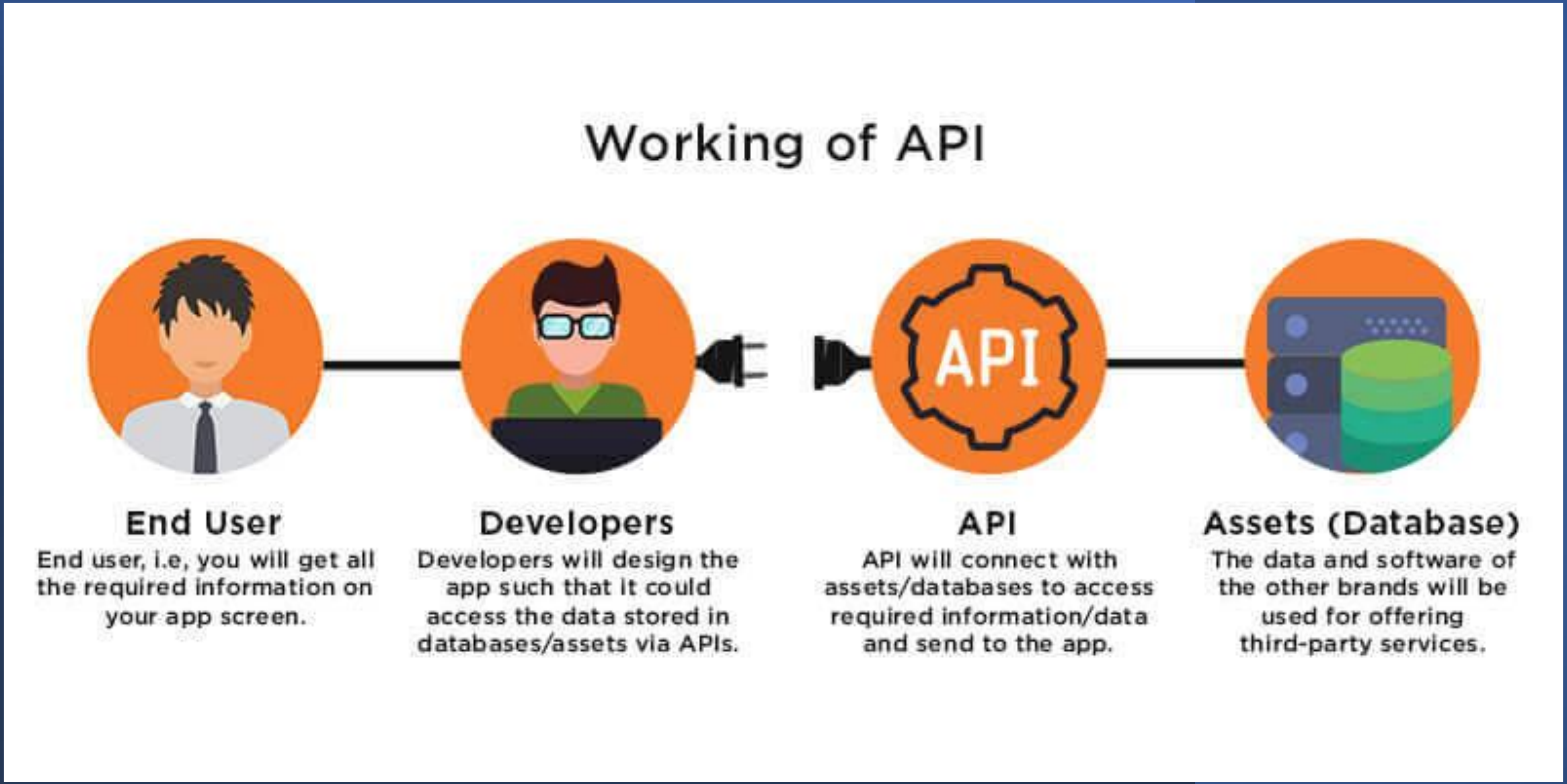
- An application programming interface (API) is a set of rules and specifications that define how computers communicate with each other
- It is a way for two or more software programs to interact with each other and exchange data
- APIs are used in a wide variety of applications, including websites, mobile apps, and enterprise software

How APIs work?

- APIs work by providing a way for software programs **to make requests and receive responses** from each other
    - The **request** typically includes a set of parameters, such as the type of data being requested or the action that needs to be performed
    - The **response** includes the requested data or the results of the requested action

How APIs work?

- APIs can be implemented in a variety of ways, but they typically use a standard communication protocol – HTTP, RPC, etc.
- These protocols provide a common way for software programs to communicate with each other, regardless of the programming language or operating system they are using



Working of API

End User

End user, i.e, you will get all the required information on your app screen.

Developers

Developers will design the app such that it could access the data stored in databases/assets via APIs.

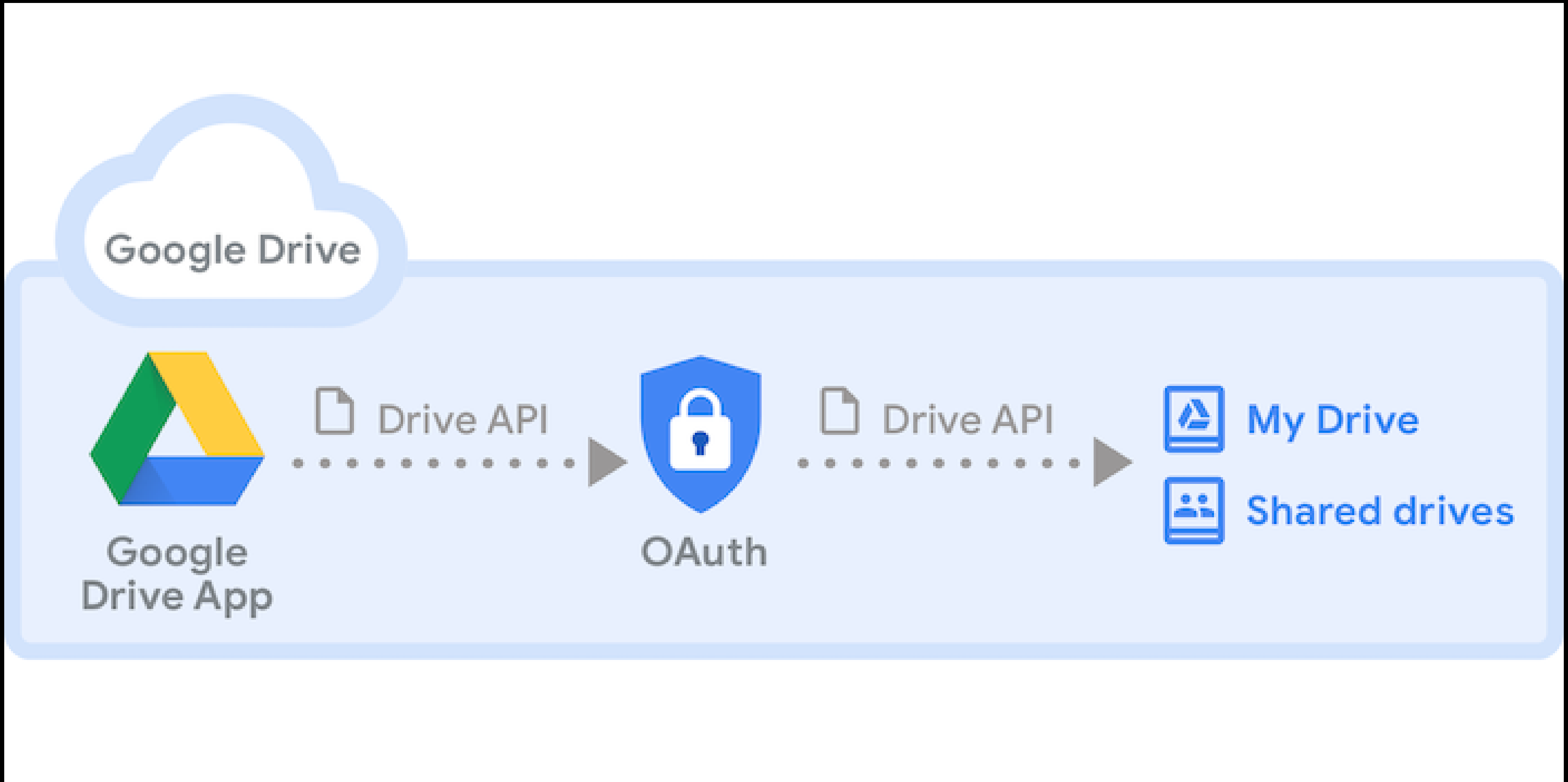
API

API will connect with assets/databases to access required information/data and send to the app.

Assets (Database)

The data and software of the other brands will be used for offering third-party services.

Example: Google Drive API



Examples of APIs

Google Maps API

- This API allows developers to embed Google Maps into their own websites and applications.

Twitter API

- This API allows developers to access Twitter data and functionality, such as posting tweets, getting user information, and searching for tweets.

Facebook API

- This API allows developers to access Facebook data and functionality, such as posting to user walls, getting friend information, and sending messages.

Payment processing APIs

- These APIs allow developers to accept payments through their websites and applications.

Weather APIs

- These APIs allow developers to get weather data for specific locations.

Practical examples of API implementations and uses

- When you use a weather app on your phone,
  - the app uses an API to get the weather data from a weather service
- When you book a flight online,
  - the website uses an API to communicate with the airline's reservation system
- When you use a social media app to log in with your Google account,
  - the app uses an API to authenticate your login with Google
- When you use a ride-sharing app to request a ride,
  - the app uses an API to communicate with the ride-sharing company's dispatch system

Some common types of APIs

- Public APIs
  - These APIs are available to anyone who wants to use them
  - They are often used by developers to build new products and services
    - Examples include the Google Maps API, the Twitter API, and the Facebook API.
- Partner APIs
  - These APIs are only available to a select group of people, such as employees of a company or its partners
  - They are often used to integrate different systems within a company or to provide access to data and functionality to partners
    - Examples include the Salesforce API and the HubSpot API

Some common types of APIs

- Internal APIs
  - These APIs are only available to employees of a company
  - They are often used to integrate different systems within a company or to provide access to data and functionality to employees
    - Examples include the Netflix API and the Amazon internal APIs

APIs can also be classified based on their architecture

- Monolithic APIs
  - These APIs are designed as a single, coherent codebase that provides access to a complex data source
- Microservices APIs
  - These APIs are designed as a set of loosely coupled services that each perform a specific task
- Composite APIs
  - These APIs combine multiple APIs into a single interface

APIs can also be classified based on the communication protocol they use

- REST APIs [representational state transfer architectural style]
- SOAP APIs [Simple Object Access Protocol]
- RPC APIs [Remote Procedure Call]

APIs can also be classified based on the communication protocol they use

- REST APIs [representational state transfer architectural style]
  - These APIs use the HTTP protocol to communicate with a server
  - REST APIs are designed to be simple, flexible, and scalable.
- SOAP APIs [Simple Object Access Protocol]
  - These APIs use SOAP to communicate with a server
  - SOAP APIs are more complex than REST APIs, but they offer more features, such as support for transactions and security
  - SOAP is often used for enterprise applications

APIs can also be classified based on the communication protocol they use

- RPC APIs [Remote Procedure Call]
    - These APIs use RPC protocol to communicate with a server
    - RPC APIs are designed to be efficient, but they are less flexible than REST APIs
    - RPC is often used for high-performance applications

Benefits of using APIs

- Abstraction
  - APIs provide a layer of abstraction between applications and libraries
  - This means that applications do not need to know the underlying implementation details of the libraries they are using
  - This can make applications more robust and easier to maintain
- Modularity
  - APIs allow applications to be built in a modular wa
  - This means that applications can be composed of different modules that each perform a specific task
  - This can make applications more scalable and easier to develop

Benefits of using APIs

- Interoperability
  - APIs allow applications to interoperate with each other
  - This means that applications can communicate with each other and share data
  - This can make it possible to build complex applications that are composed of multiple different applications

Libraries

- A library in computer science is a collection of pre-written code that can be used by programmers to develop software
- Libraries typically contain functions, classes, and other data structures that can be used to perform common tasks, such as reading and writing files, manipulating data, and communicating with networks
- A library is a collection of code that is shared between different programs
- Libraries can be defined in the source code of a program, or they can be compiled into separate files that can be linked to a program at runtime

Examples of libraries

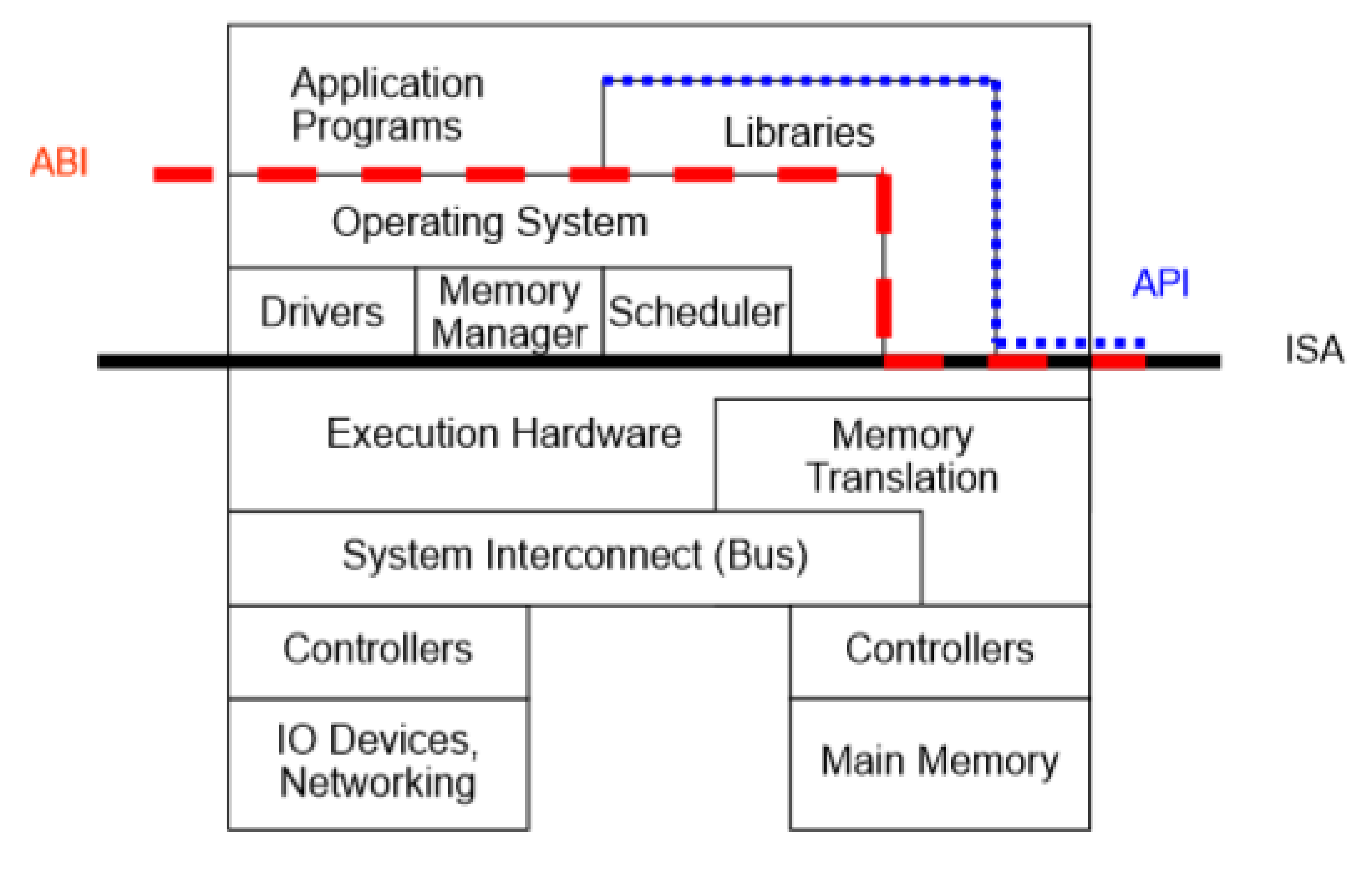
- The C standard library
  - provides functions for performing common tasks such as memory management, input/output, and mathematical operations.
- The Java standard library
  - provides a wide variety of functions and classes for developing Java applications
- The Python standard library
  - provides a wide variety of functions and modules for developing Python applications

Application Binary Interface (ABI)

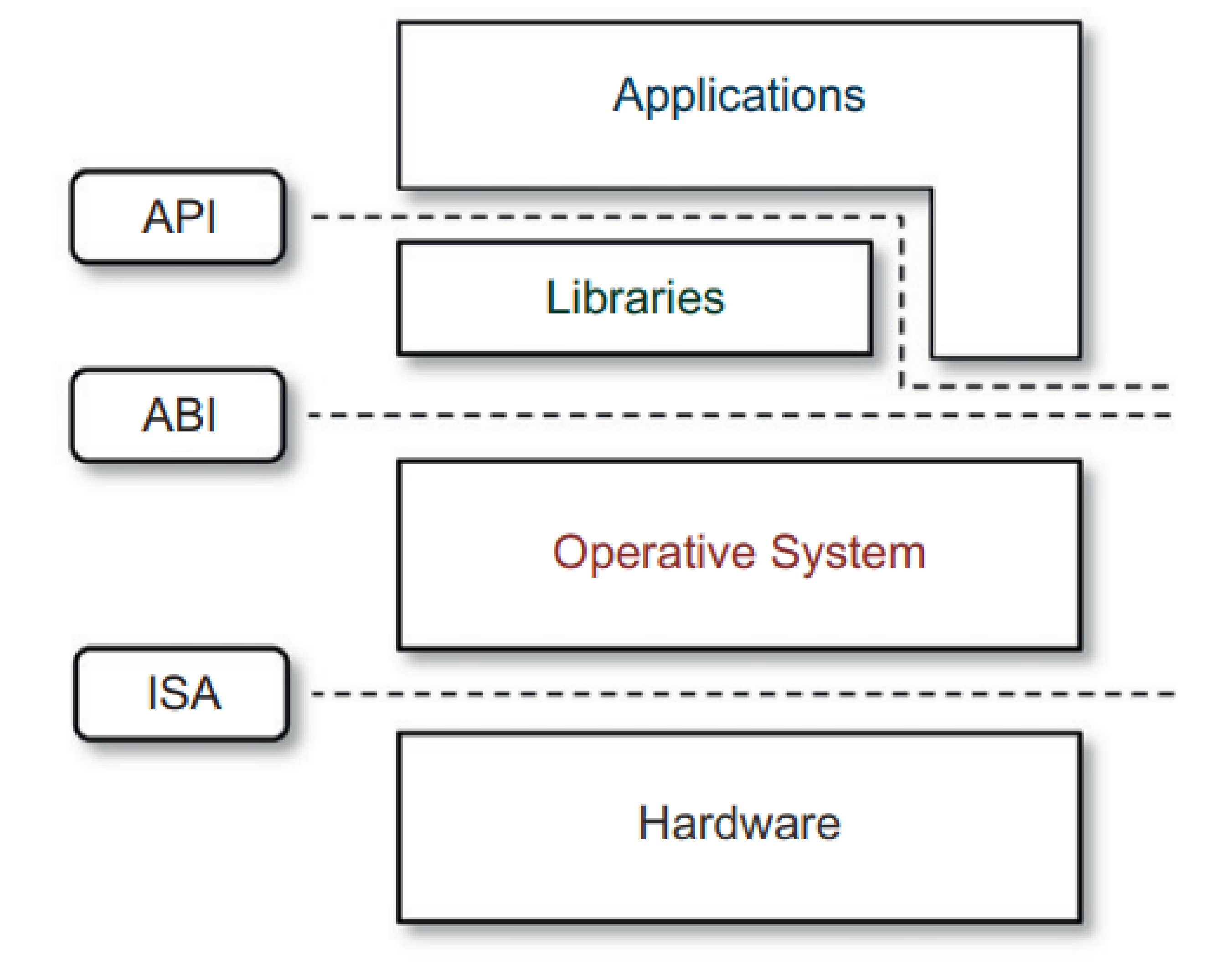
- ABI separates the operating system layer from the applications and libraries, which are managed by the OS
- ABI covers details such as low-level data types, alignment, and call conventions and defines a format for executable programs
  - System calls are defined at this level
- API: *"Here are all the functions you may call."*
- ABI: *"This is how to call a function."*

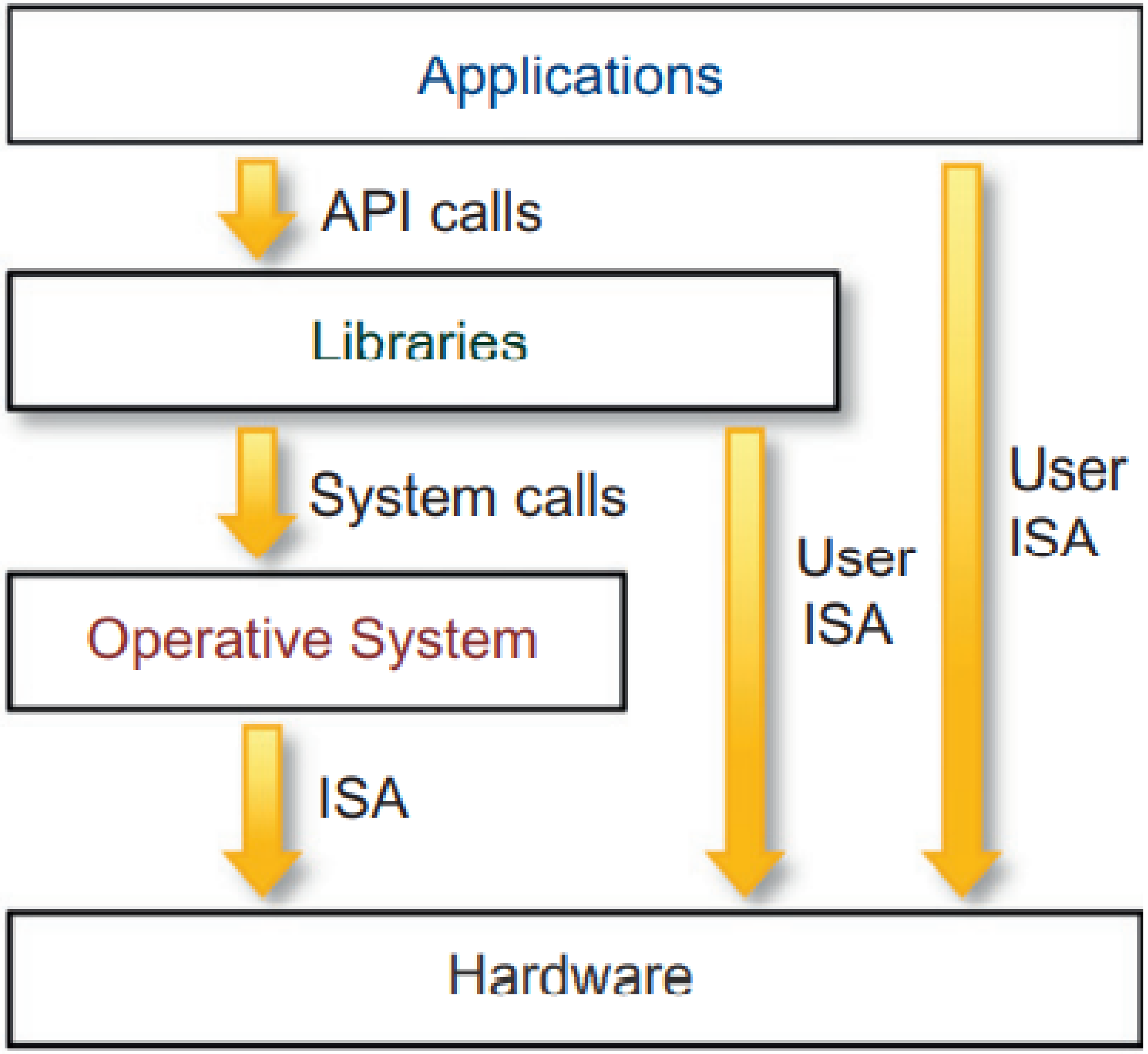
Application Binary Interface (ABI)

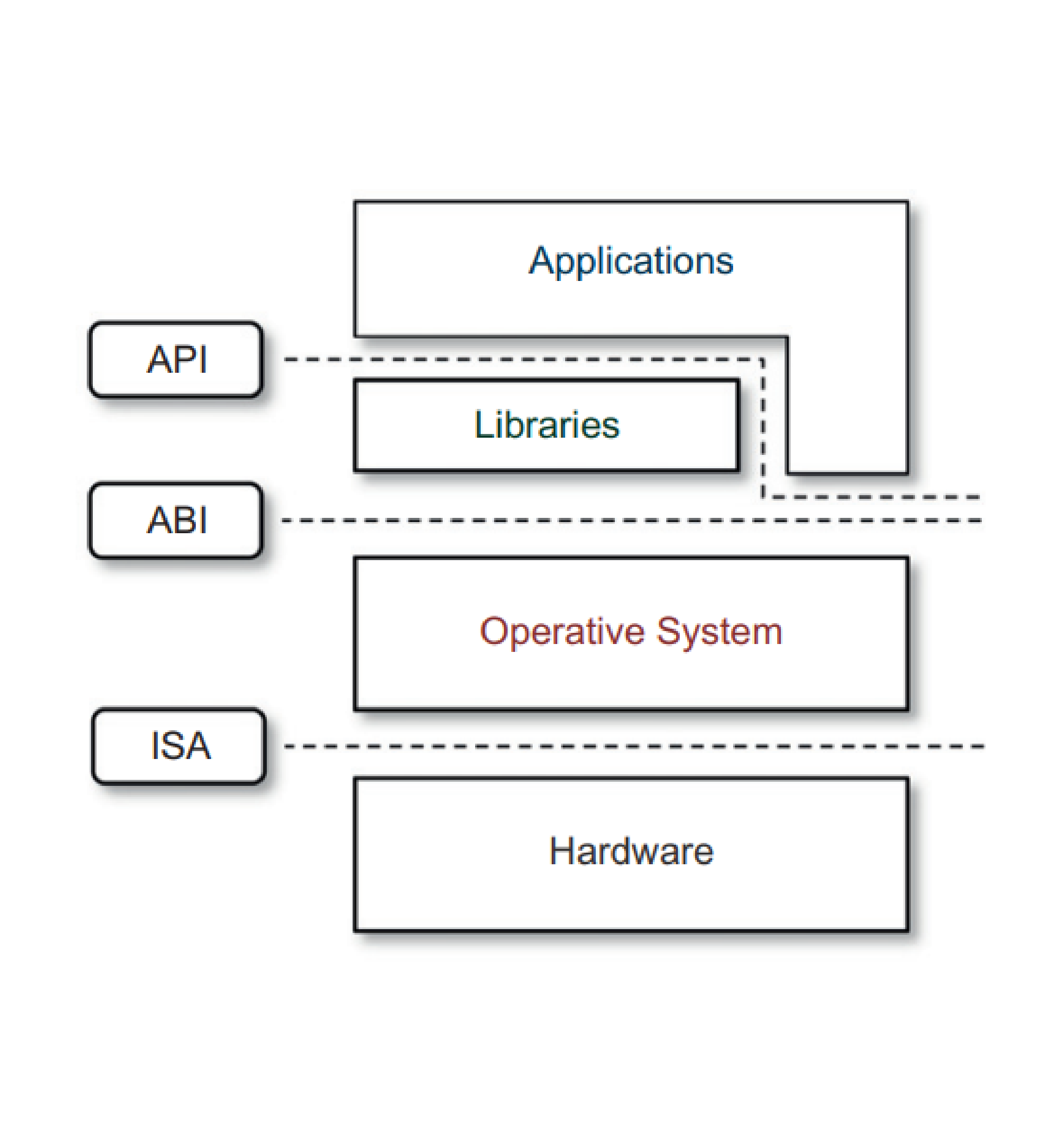
- ABI defines **how** to access data structures or computational routines in a low-level, hardware-dependent format, machine code
- ABIs are typically used to define the interaction between different programs, or between programs and the operating system
- ABIs typically define things like:
  - The calling conventions for functions, such as the order of arguments and the return value
  - The layout of data structures in memory
  - The way that programs interact with the operating system, such as how to request system resources or handle interrupts

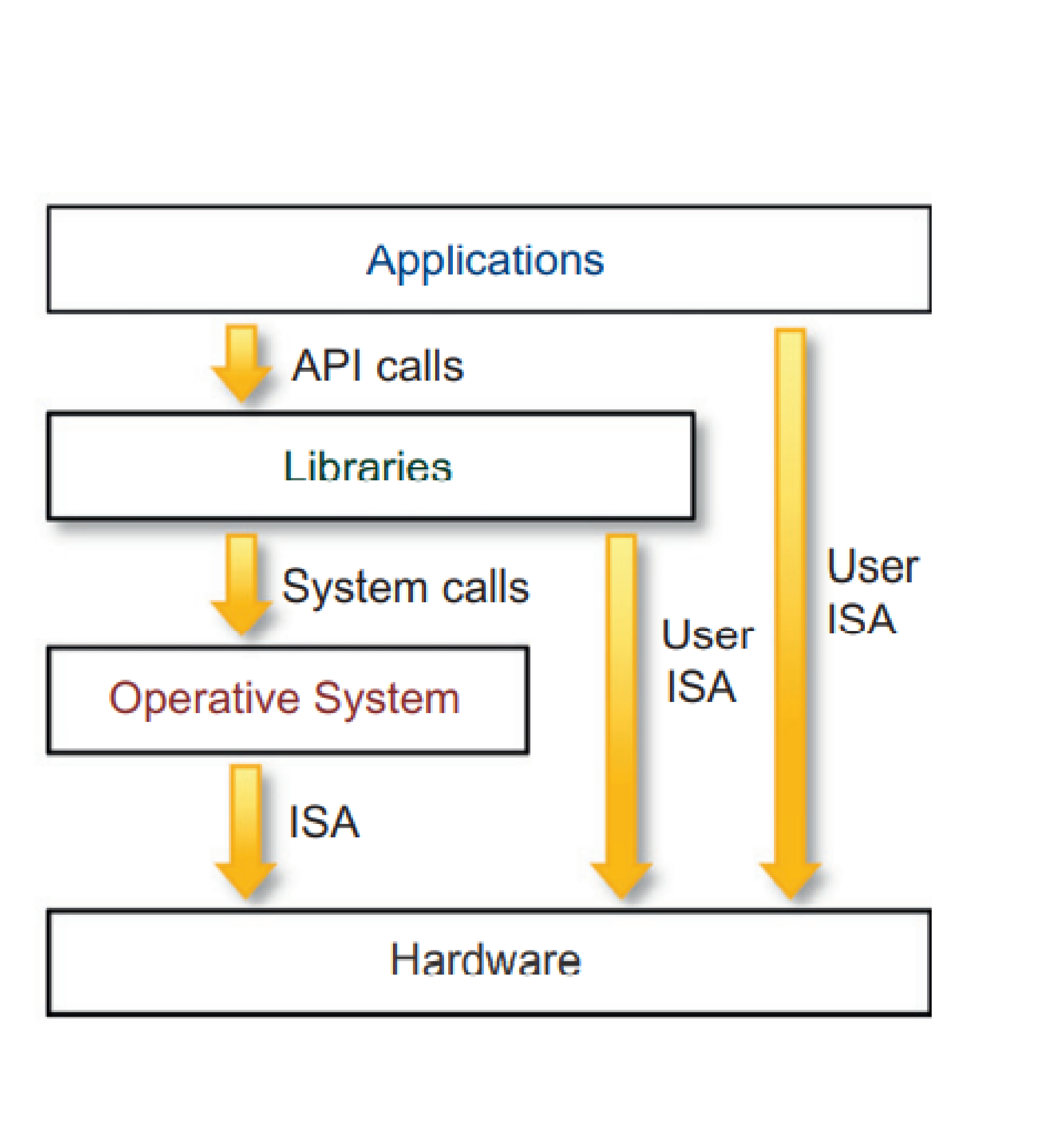

Application Programs
Libraries
ABI
Operating System
Drivers
Memory Manager
Scheduler
API
ISA
Execution Hardware
Memory Translation
System Interconnect (Bus)
Controllers
Controllers
IO Devices, Networking
Main Memory

Application Binary Interface (ABI)




Applications
API calls
Libraries
System calls
Operative System
ISA
User ISA
User ISA
Hardware


Applications
API
Libraries
ABI
Operative System
ISA
Hardware


Applications
API calls
Libraries
System calls
User ISA
User ISA
Operative System
ISA
Hardware

Examples of ABIs:

- The x86-64 ABI is a standard ABI for the x86-64 architecture
  - It is used by most operating systems that run on x86-64 processors, such as Linux, macOS, and Windows
- The ARM ABI is a standard ABI for the ARM architecture
  - It is used by most operating systems that run on ARM processors, such as Android and iOS
- The Java Virtual Machine (JVM) ABI is a standard ABI for Java applications
  - It allows Java applications to run on a variety of different platforms, such as Linux, macOS, and Windows

ABI conventions depend on two main things

1. The Instruction Set Architecture (ISA) — mainly for calling conventions
2. The operating system (OS) being used — system calls, runtime libraries, etc.

- This dual dependency is why code compiled for Windows won’t work on an OS X machine, even though they might use the same CPU with the same ISA.

Examples of how the ABI is used in operating system software

- When a program makes a system call, the **ABI defines how the operating system should handle the system call**.
- When a program accesses a library function, the ABI defines how the program should call the function and how the function should return its results.
- When a program communicates with another program over a network, the ABI defines how the program should format the messages and how it should interpret the messages received from the other program.

API and ABI

API

API defines the order in which we pass arguments to a function

API defines which functions are part of your library.

ABI

- ABI defines the mechanics of how these arguments are passed (registers, stack, etc.).
- ABI defines how your code is stored inside the library file, so that any program using the library can locate the desired function and execute it

System Call and ABI

- **System calls** are a way for programs to request services from the operating system
  - They allow programs to perform tasks such as
    - reading and writing files,
    - creating and managing processes, and
    - communicating with other devices
- ABI defines how programs should make system calls
- ABI specifies the **calling conventions** for system calls, such as the number and order of arguments, and the return value

System Call and ABI

- ABI specifies how the operating system should handle system calls, such as
  - how to validate the arguments
  - how to return the results

System Call and ABI - Example

1. A program calls the open() system call to open a file.
2. The ABI specifies the calling conventions for the open() system call, such as the number and order of arguments.
3. The operating system validates the arguments to the open() system call.
4. The operating system opens the file and returns a file descriptor to the program.
5. The ABI specifies how the operating system should return the file descriptor.

❖The program can then use the file descriptor to read and write data to the file.

ABI calling conventions

- ABI calling conventions are a set of rules that define:
  - how functions are called and
  - how arguments are passed and returned
- They are important for ensuring compatibility between different programs and libraries
- There are many different ABI calling conventions, but they all share some common features. For example, most ABIs define the following:
  - The order in which arguments are passed to a function
  - Whether arguments are passed on the stack, in registers, or a mix of both
  - Which registers are used to return the results of a function
  - Who is responsible for cleaning up the stack after a function call

Common ABI calling conventions

- **Cdecl**
  - used by most C and C++ compilers
  - It passes arguments on the stack & returns the result in the **EAX register**
- **Stdcall**
  - used by most Windows APIs
  - It passes arguments on the stack & returns the result in the EAX register
  - The caller is responsible for cleaning up the stack after the function call
- **Fastcall**
  - This is a calling convention that is designed for performance
  - It passes the first few arguments in registers & the remaining arguments on the stack
  - The caller is also responsible for cleaning up the stack after the function call

ABI calling conventions

- ABI calling conventions used by a particular program or library are typically **defined in the documentation for that program or library**
- It is important to use the correct calling conventions when calling functions, as **otherwise the program may crash or produce unexpected results**

Examples of how ABI calling conventions are used in practice:

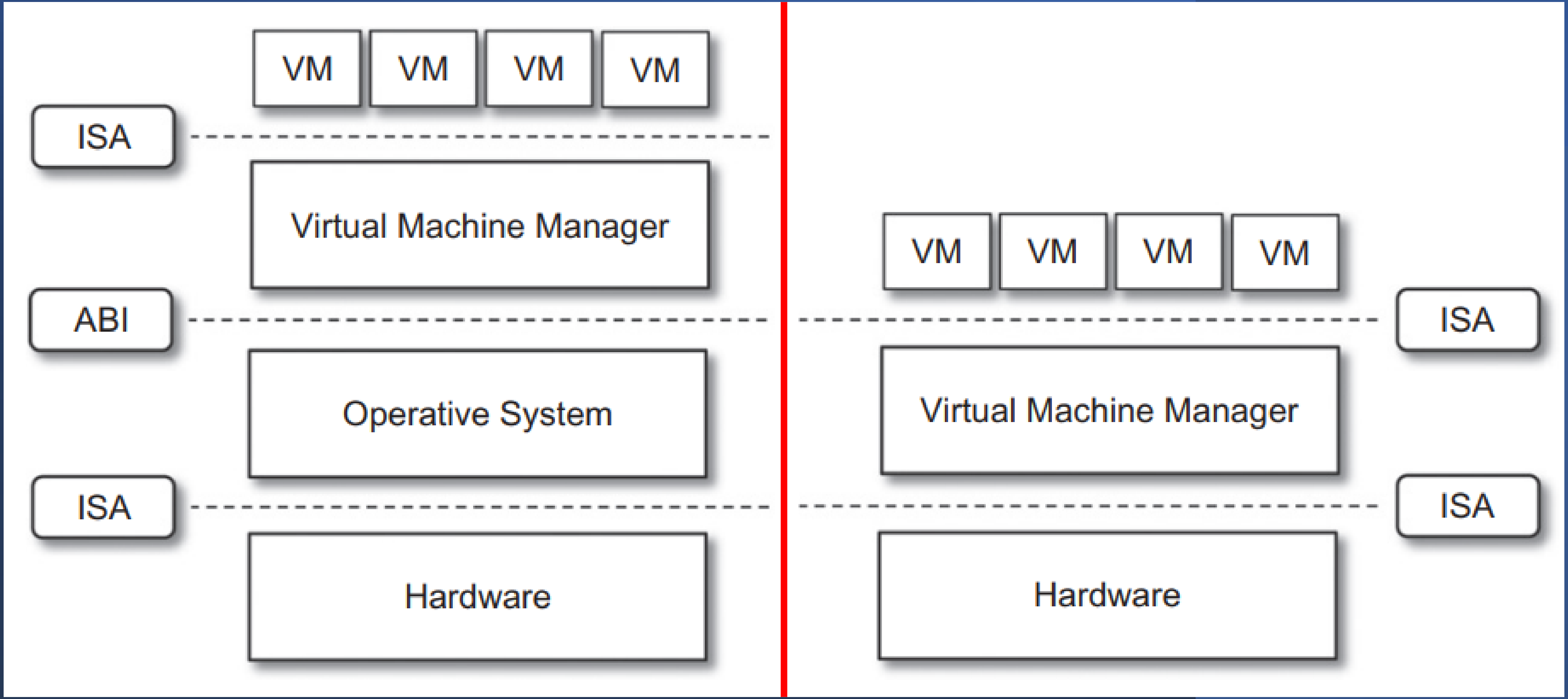
- When a program calls a library function,
  - the ABI calling conventions define how the program should pass the arguments to the function and how the function should return the result
- When a program makes a system call,
  - the ABI calling conventions define how the program should call the system call and how the operating system should handle the system call
- When a program communicates with another program over a network,
  - the ABI calling conventions may be used to define how the programs should format the messages and how they should interpret the messages received from the other program

Hypervisor – Type I

- Type I hypervisors run directly on top of the hardware
- Interact directly with the ISA interface exposed by the underlying hardware, and they emulate this interface in order to allow the management of guest operating systems.
- A native virtual machine since it runs natively on hardware

Hypervisor – Type II

- Type II hypervisors require the support of an operating system to provide virtualization services
- This means that they are programs managed by the operating system, which interact with it through the ABI and emulate the ISA of virtual hardware for guest operating systems
- This type of hypervisor is also called a hosted virtual machine since it is hosted within an operating system


VM
VM
VM
VM
ISA
Virtual Machine Manager
ABI
Operative System
ISA
Hardware
VM
VM
VM
VM
ISA
Virtual Machine Manager
ISA
Hardware

Virtualization Techniques

- Process-level techniques are implemented on top of an existing operating system, which has full control of the hardware
- System-level techniques are implemented directly on hardware and do not require (or require a minimum of support from) an existing operating system
    - Hardware virtualization techniques
        - Hardware assisted Virtualization (known as System Virtualization)
        - Full Virtualization
        - Paravirtualization
        - Partial Virtualization