

•
5 0 . 0 0 5 C S E

Natalie Agus
Information Systems Technology and Design
SUTD

OS SERVICES

What can the OS do?



Execute

Programs



I/O

Operations



File System

Operations and manipulation



Comms

Of processes via shared memory



Security & Error Detection

Handles debugging facilities



Resource

Allocations



Accounting

Keeps track of processes

The OS acts as an intermediary between these two:

Hardware (regs, caches, RAM, I/O devices, CPU, etc)

User programs (browser, Spotify, Matlab, anything installed in your com)

User & System Processes

GUI

Batch file

Terminal

UI of Operating System

SYSTEM CALLS

SERVICES:

Protection

Program exec

Accounting

Security

Error detection File system

Resource alloc

Comms

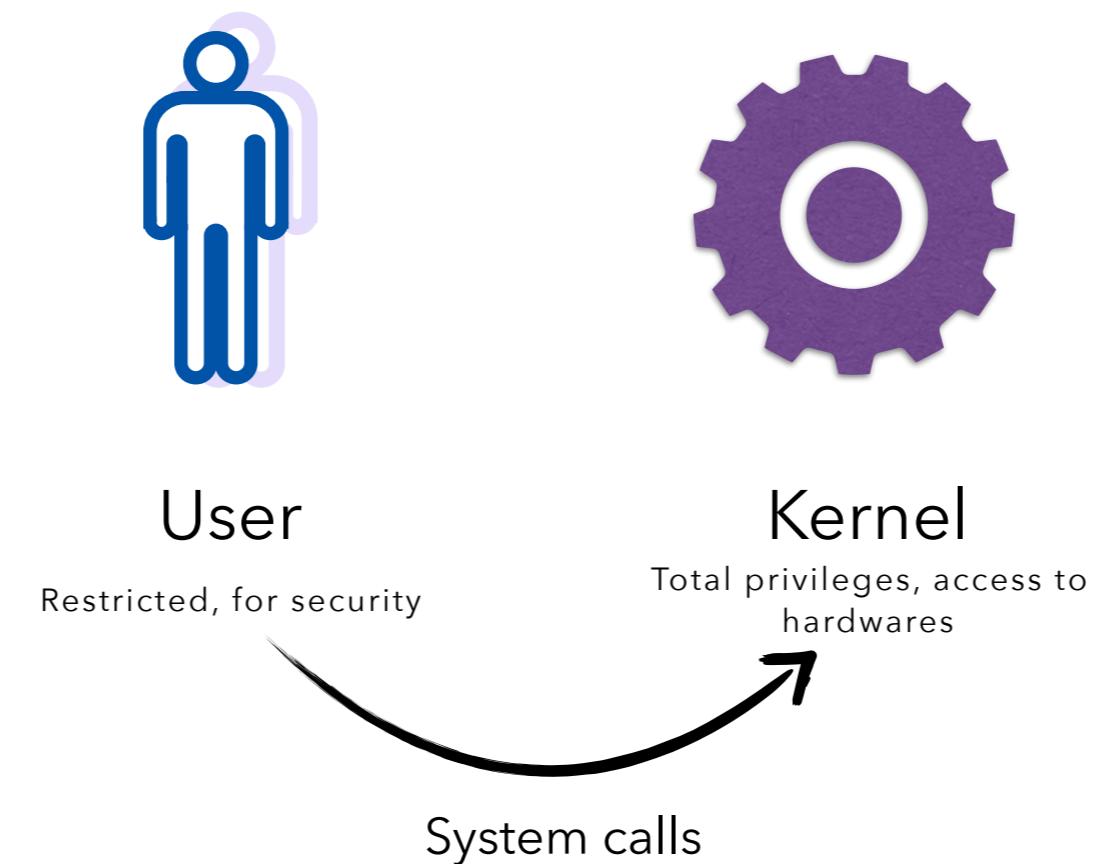
I/O Operations

OPERATING SYSTEM

HARDWARE

SYSTEM CALLS

Programming interface provided by the OS Kernel for user to access services

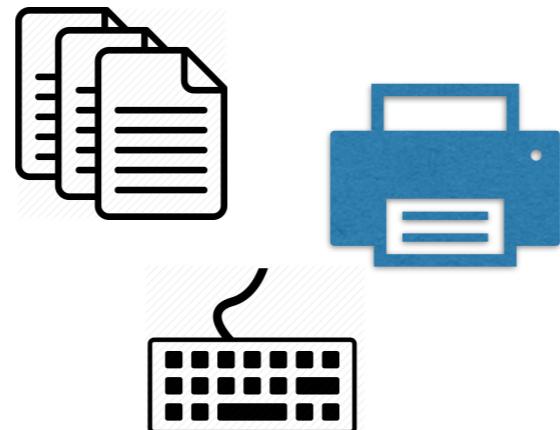


TYPES OF SYSTEM CALLS



PROCESS SCHEDULING

Decides which process executes, sleep, terminates, queue, priority



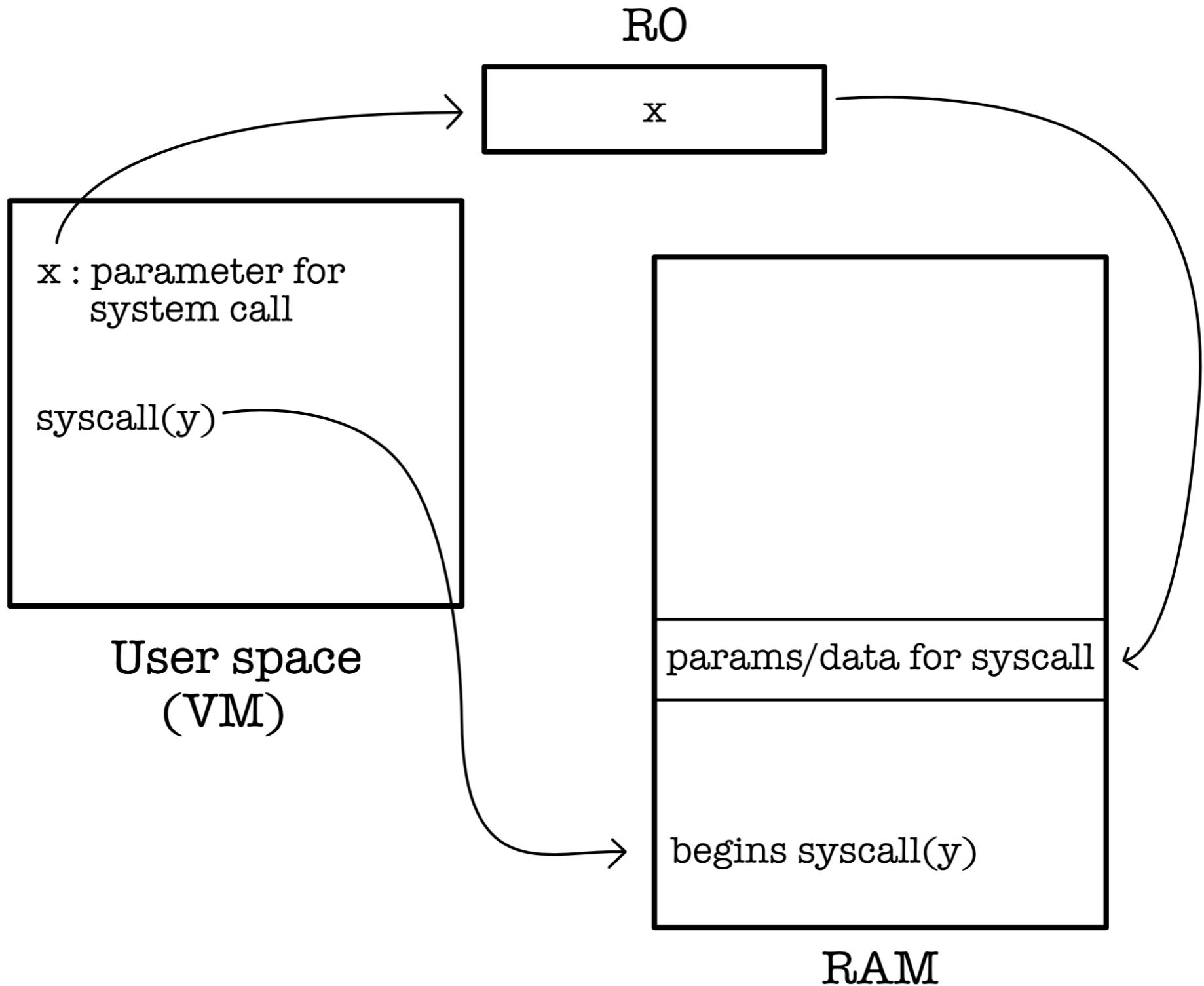
File and device I/O Management



Information maintenance & Security

SYSTEM COMMUNICATION





```
#include <unistd.h>
#include <sys/syscall.h>
int main(void) {
    syscall(SYS_write, 1, "hello, world!\n", 14);
    return 0;
}
```

Using System Call
SYS_write is an int, OS-specific

PRINT TO STDOUT

```
#include <stdio.h>
int main(void) {
    printf("hello, world!\n");
    return 0;
}
```

Using Function Call

syscall() is assembly

```
#include <sysdep.h>

/* Please consult the file sysdeps/unix/sysv/linux/x86-64/sysdep.h for
   more information about the value -4095 used below. */

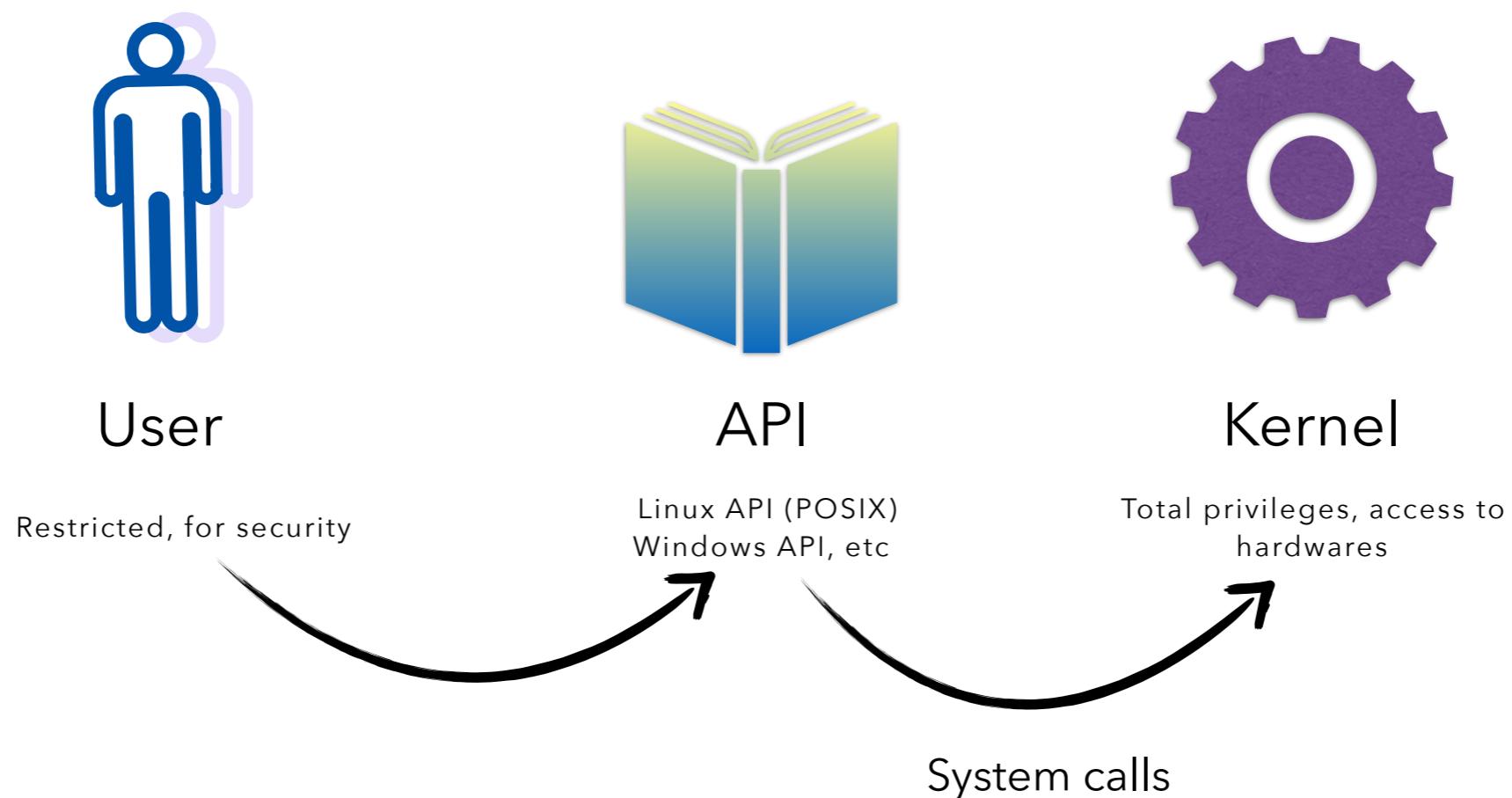
/* Usage: long syscall (syscall_number, arg1, arg2, arg3, arg4, arg5, arg6)
   We need to do some arg shifting, the syscall_number will be in
   rax. */

.text
ENTRY (syscall)
    movq %rdi, %rax           /* Syscall number -> rax. */
    movq %rsi, %rdi            /* shift arg1 - arg5. */
    movq %rdx, %rsi
    movq %rcx, %rdx
    movq %r8, %r10
    movq %r9, %r8
    movq 8(%rsp),%r9          /* arg6 is on the stack. */
    syscall                   /* Do the system call. */
    cmpq $-4095, %rax          /* Check %rax for error. */
    jae SYSCALL_ERROR_LABEL   /* Jump to error handler if error. */
    ret                      /* Return to caller. */

PSEUDO_END (syscall)
```

A BETTER CHOICE: API

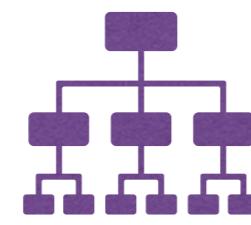
Application programming interface adds layer of abstraction than direct system calls



MORE ABOUT API

**A good API makes it easier for developer to develop a computer program
Just like a good GUI makes it easier for user to use a computer**

Software building tools



Subroutine definitions



Communication protocols

```
void test_time(){
    // to store execution time of code
    double time_spent = 0.0;

    clock_t begin = clock();

    // do some stuff here
    sleep(3);

    clock_t end = clock();

    // calculate elapsed time by finding difference (end - begin)
    // divide by CLOCKS_PER_SEC to convert to seconds
    time_spent += (double)(end - begin) / CLOCKS_PER_SEC;

    printf("Time elapsed is %f seconds", time_spent);
}
```

And many other **function calls** to access OS services in C:

- read()
- write()
- open()
- close()
- fork()
- execvp()

SYSTEM PROGRAMS

A convenient environment (tools) for user applications to perform system calls and use the computer hardware

- Simple: simply UI to system calls
- Complex: perform complex series of system calls to provide services to user apps

Most users' view of an OS is defined by system programs, not system call itself



SYSTEM PROGRAMS

Used for operating computer hardware and very common purposes

Installed on the computer when OS is installed

User doesn't typically interact with system software because they run in the background

System programs run independently, provides platform for running app software

More example: compiler, assembler, debugger, driver, antivirus, network softwares

USER PROGRAMS

Used to perform a specific task as requested by user

Installed according user requirements

User interacts mostly with user programs (also called application softwares)

Cannot run independently, it requires help from various system softwares to run

Some example: media player, photos editing softwares, games

• S Y S T E M P R O G R A M C A T E G O R I E S

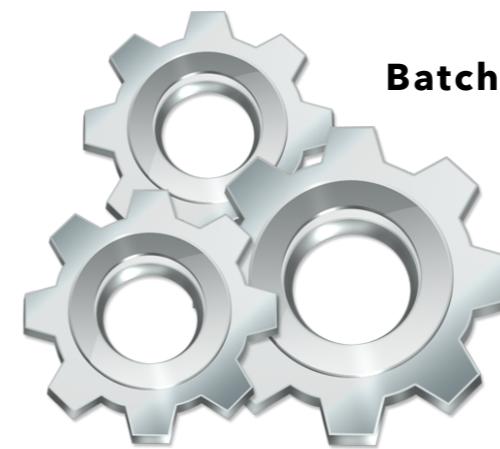
1. File manipulation
2. Status information
3. File modification
4. Programming language support
5. Program loading and execution
6. Communication
7. Application programs

UI OF OS SERVICES



Graphical User Interface

- E.g: desktop
- Click icons to open program
- Change settings
- Close programs
- Switch between programs



Batch file

- A series of commands: telling the computer exactly what to do
- Commands are OS-specific
- To be executed by command line interpreter
- Stored as a plaintext file

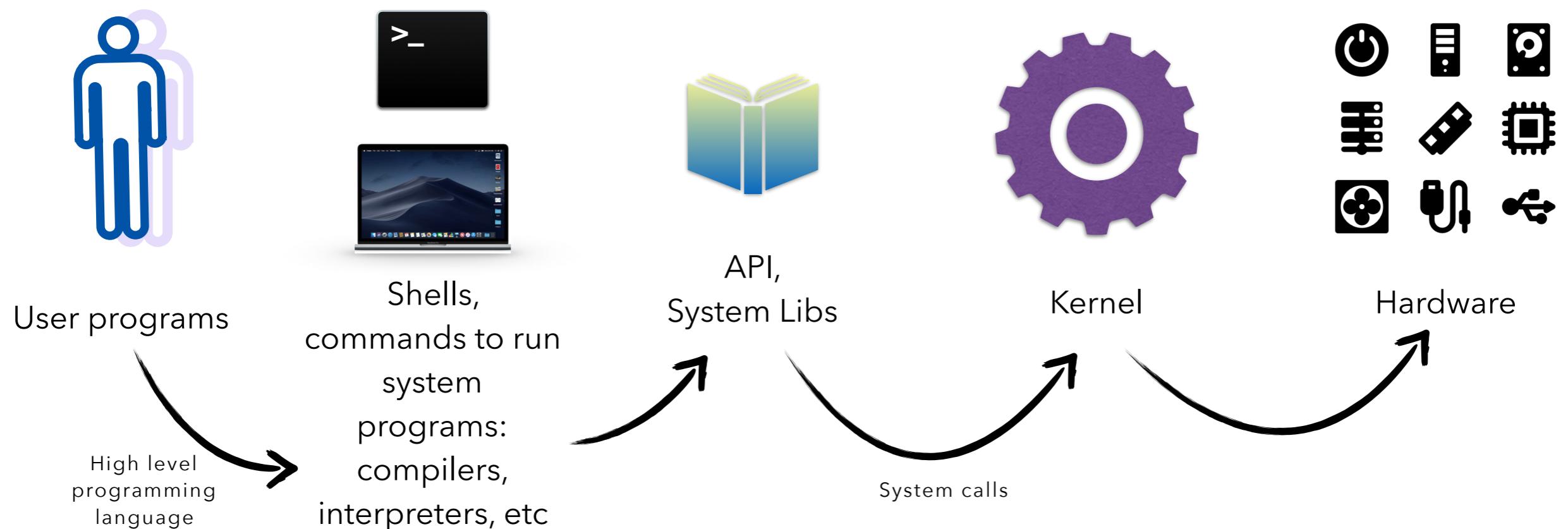


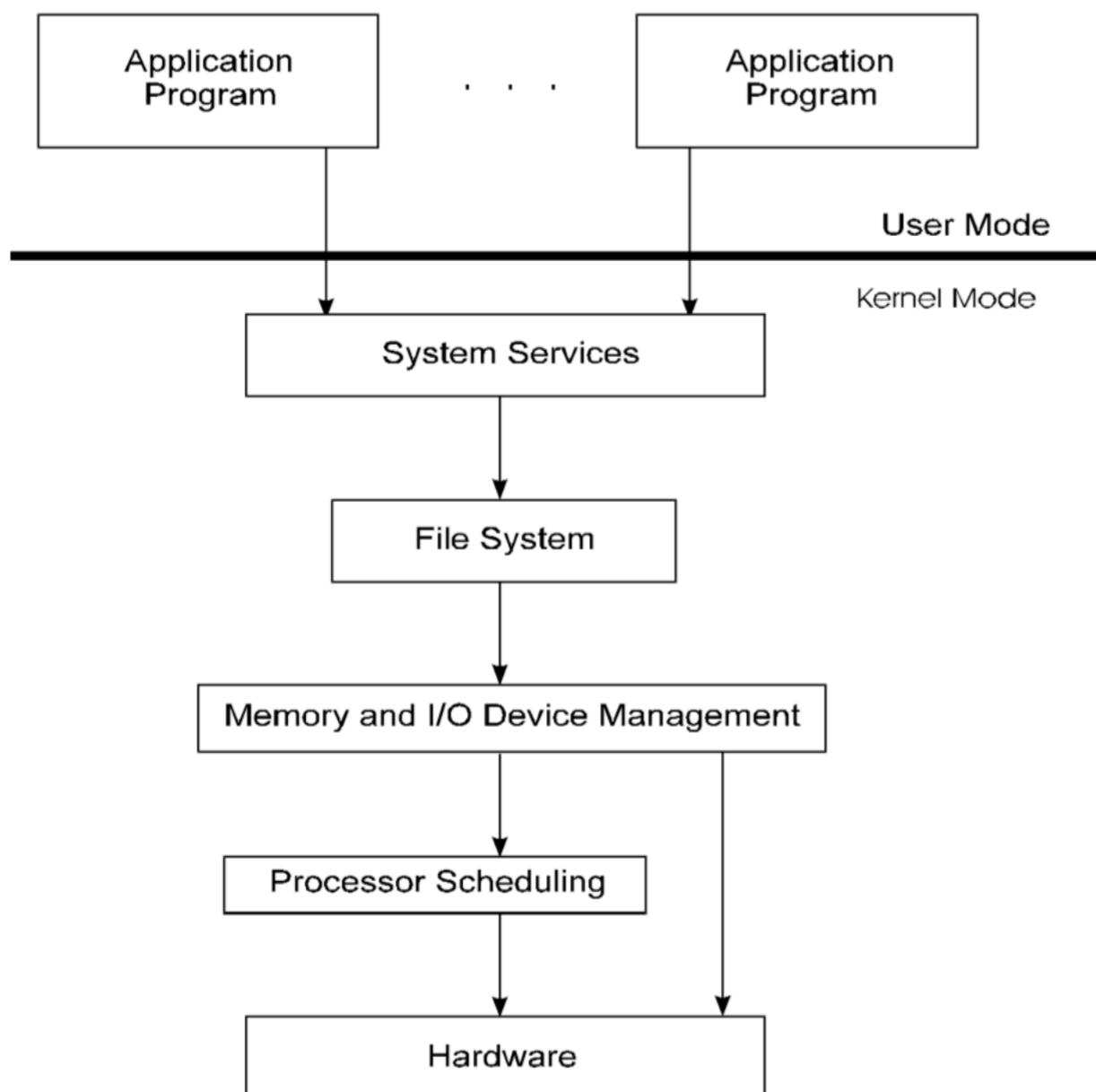
Terminal (command line interface)

- Uses shell
- A software that allows user interaction with computer via terminal
- UNIX shell: **Bash**
- Interprets commands and executes as individual processes

LAYERED APPROACH: MODERN UNIX

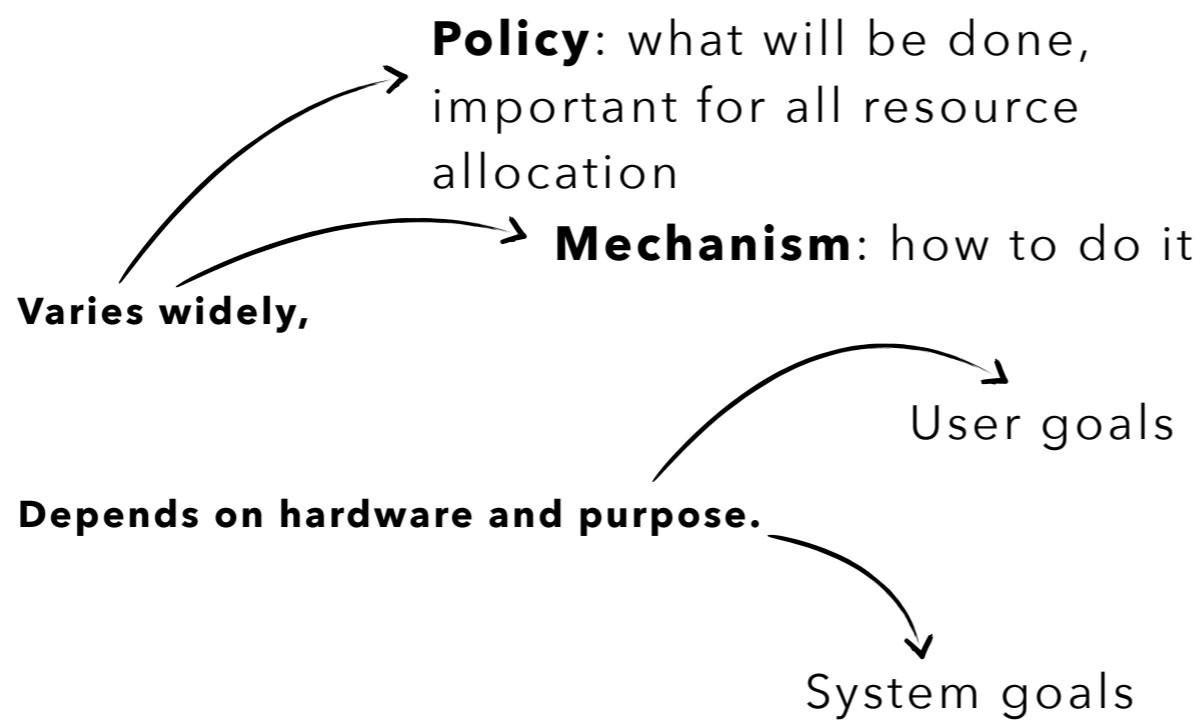
Each layer uses services provided by the layer beneath for
modularity and **ease of debugging**





Generic overview of
layered approach

GENERAL IDEA OF OS DESIGN



OS EXAMPLES

Most OS is written in C + assembly.

1. Simple and layered structure (monolithic): MS-DOS, UNIX

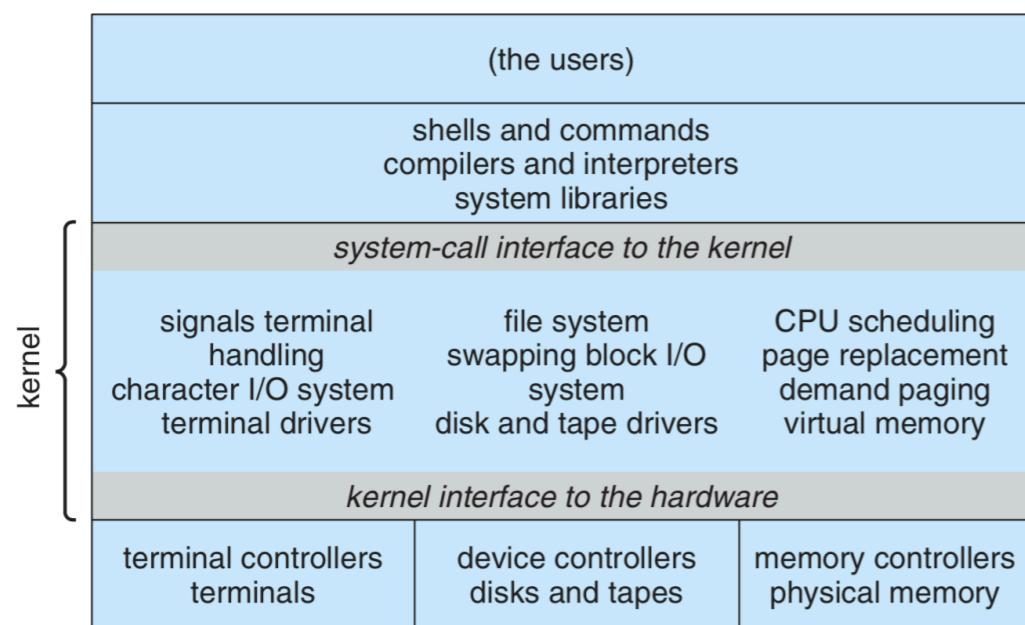


Figure 2.13 Traditional UNIX system structure.

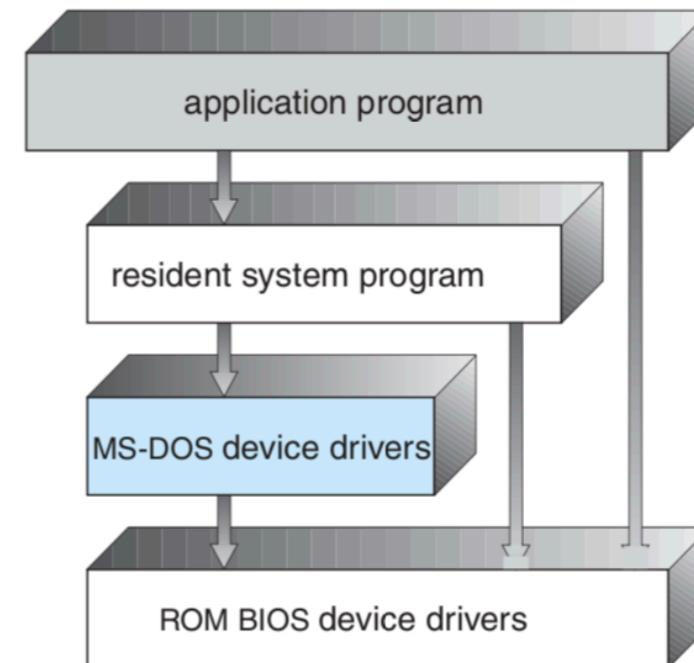
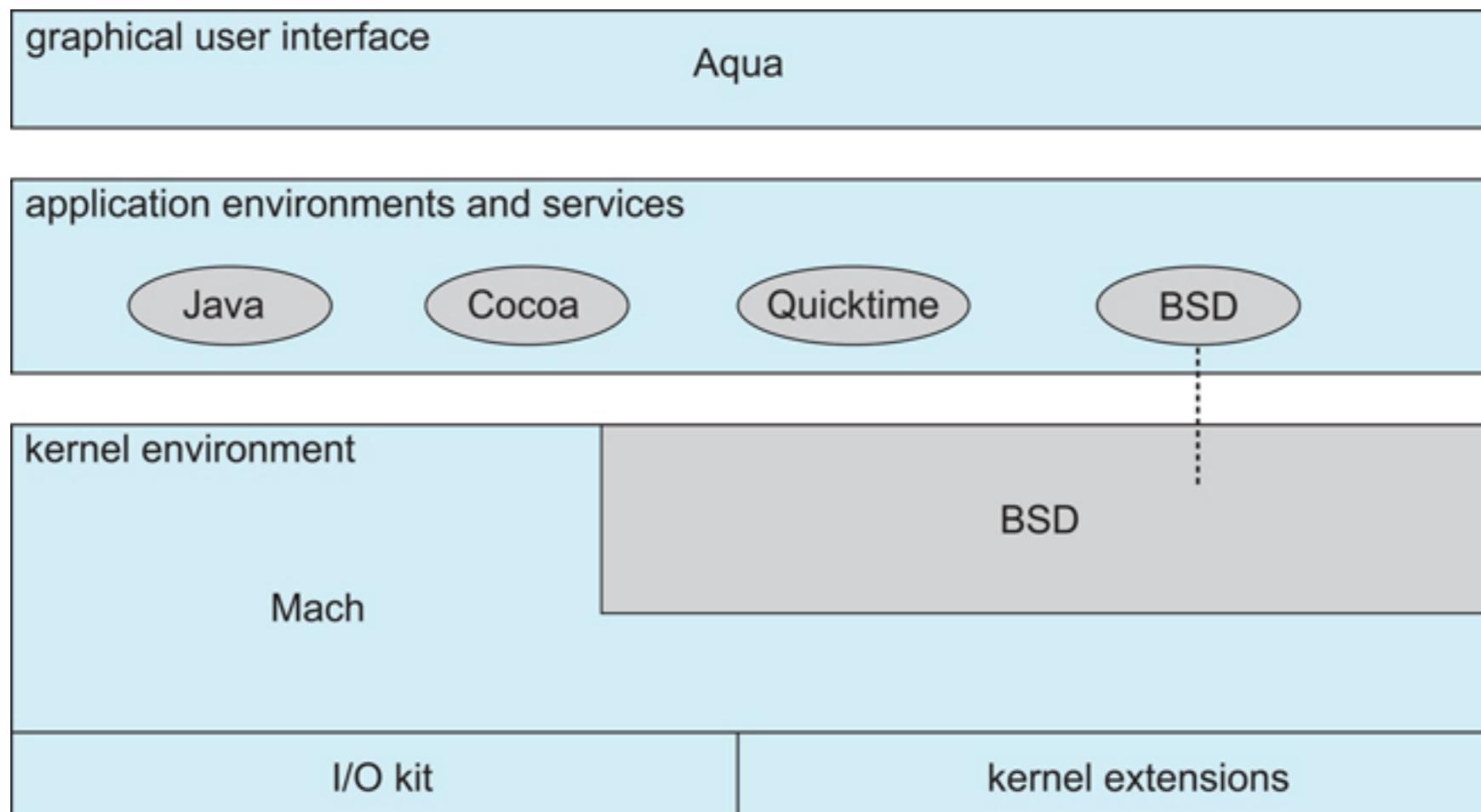


Figure 2.12 MS-DOS layer structure.

OS EXAMPLES

2. Microkernels: the original Mach

Microkernel provides minimal process and memory management, and communication facility. Everything else is moved to system program and user program.



OS EXAMPLES

3. Object oriented: JX - single address space system, no MMU

JX OS is mostly in Java except domainZero which is also in C + assembly. Each domain is an independent JVM.

An instance of JVM is made whenever a Java applet is run.

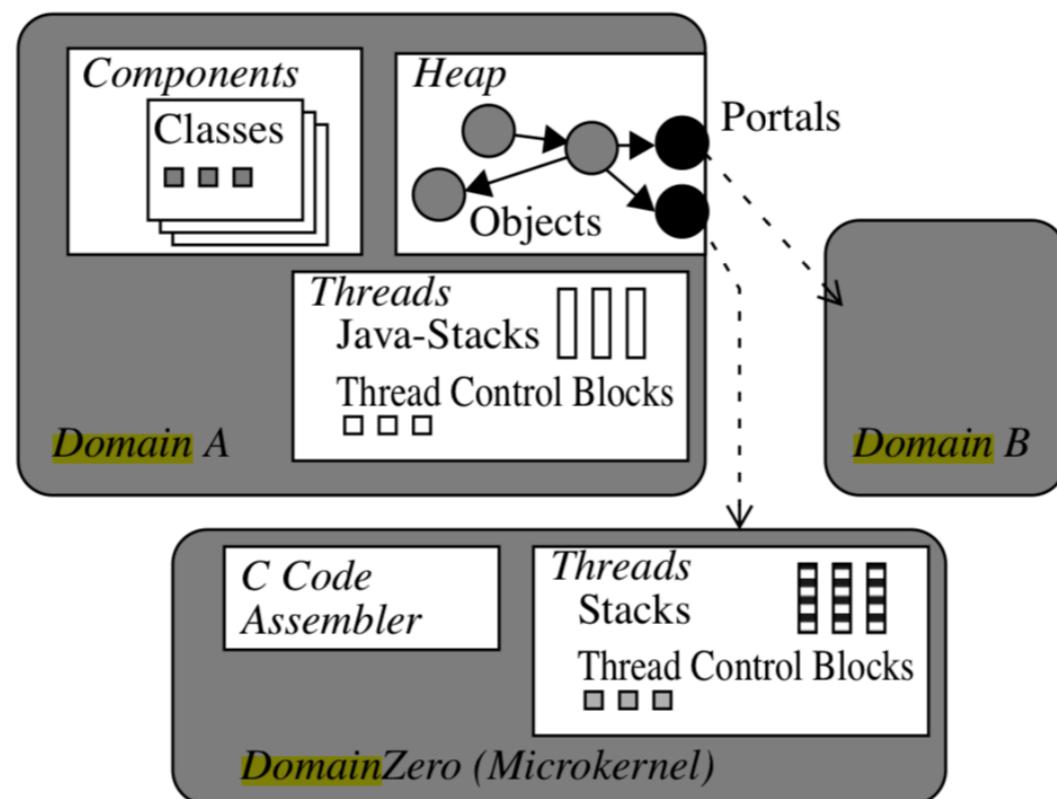
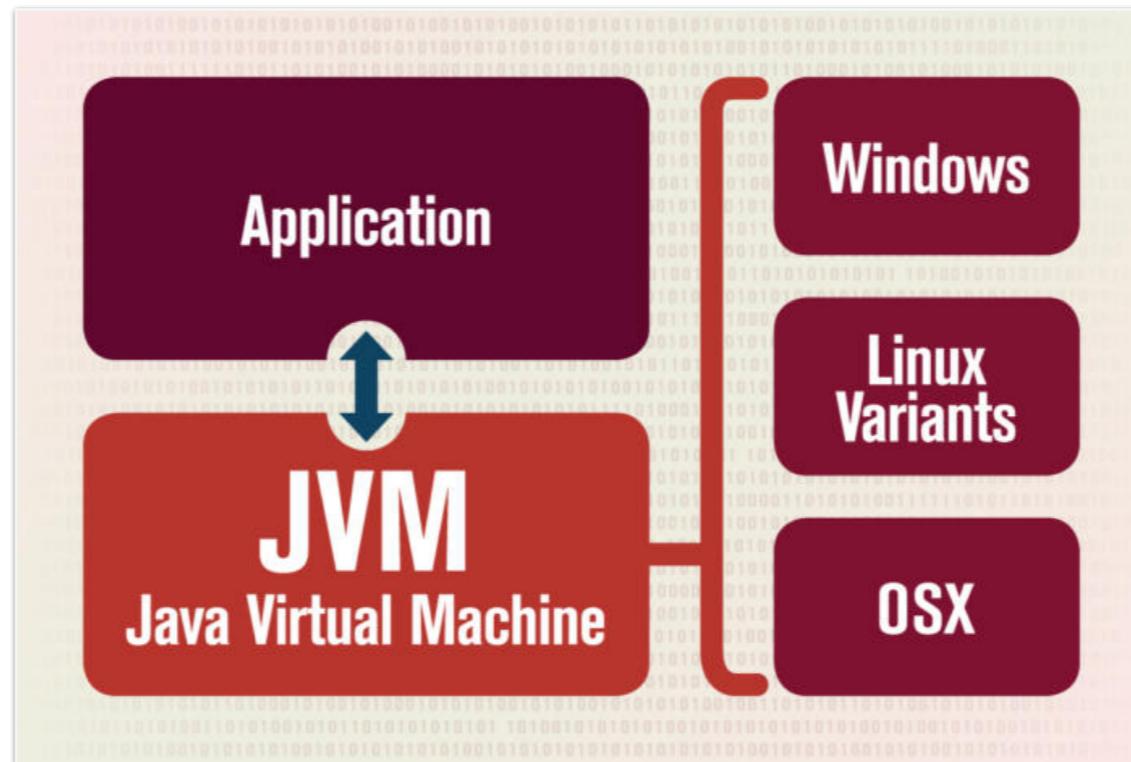


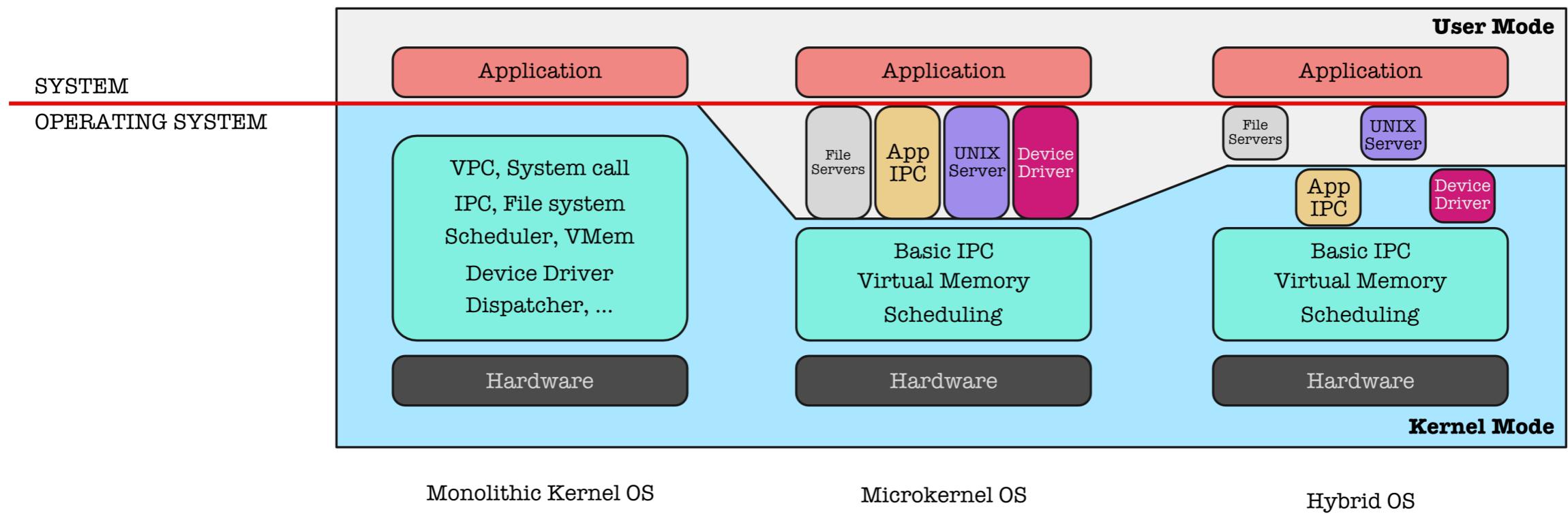
Figure 1: Structure of the JX system

MORE ABOUT JVM

JVM: An abstract machine that can run on any host OS. Takes care of its own memory management (allocation and garbage collection).



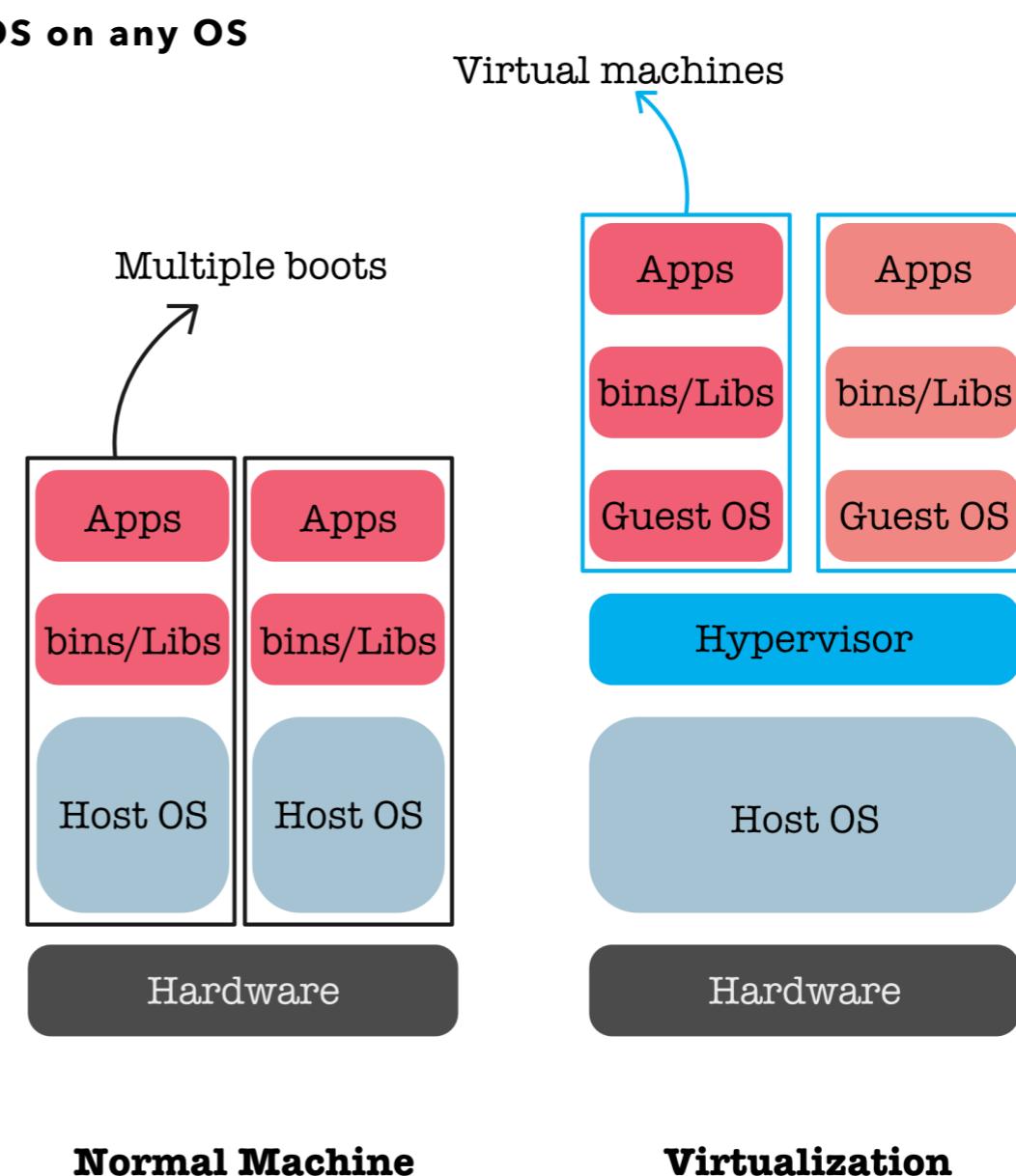
The JVM provides a **portable execution environment** for Java-based applications



OS STRUCTURES SUMMARY

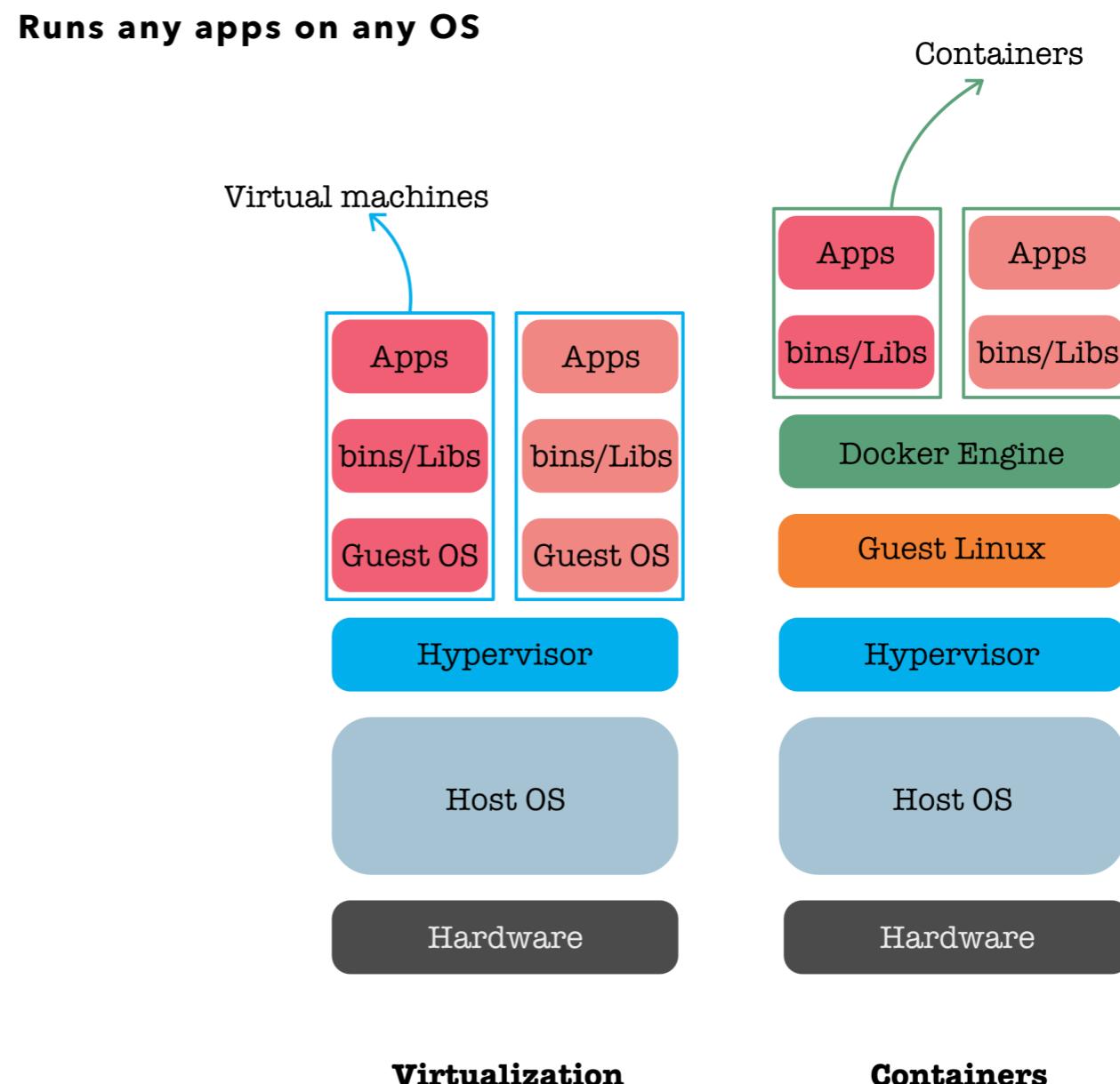
LAYERED APPROACH EXTENSION: VIRTUALIZATION

- Normal machine (computers) nowadays can run any OS on its hardware, e.g: **Dual boot**
- An extension to that is **virtualization**, where you can run any OS on any OS
- Note on **hypervisor**: a computer software / firmware / hardware that runs virtual machines.
- Examples of hypervisors: VMWare, VMWorkstation, VirtualBox, Parallel Desktop for Mac, etc



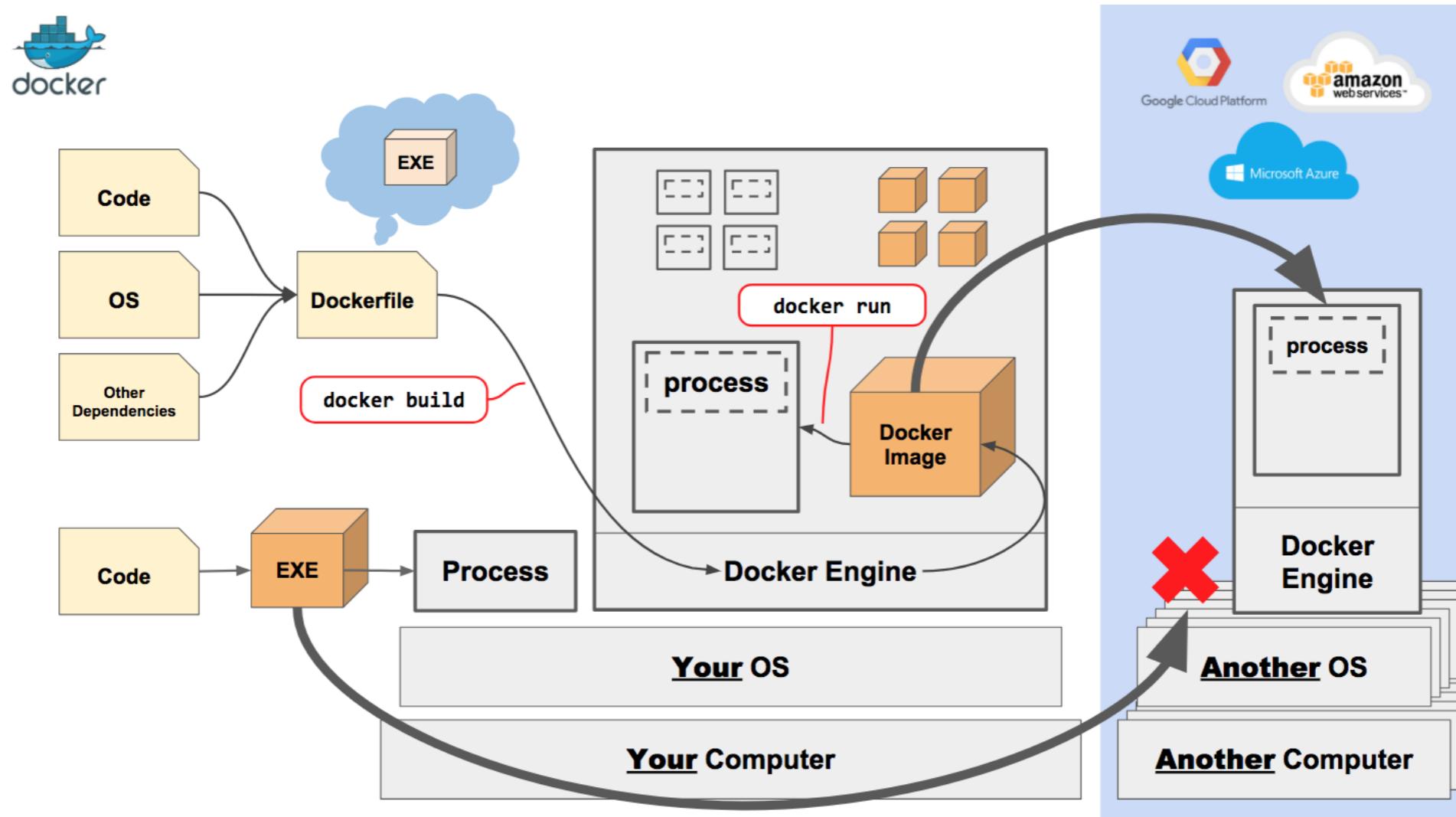
LAYERED APPROACH EXTENSION: CONTAINERIZATION

- A further extension to this is **containerisation**
- **Containers** allow a developer to package up an application with **all** of the parts it **needs**, such as libraries and other dependencies, and ship it all out as one package
- Example of softwares that support containerisation:
Docker



DOCKER

<https://medium.freecodecamp.org/docker-quick-start-video-tutorials-1dfc575522a0>



LAYERED APPROACH ALLOWS ABSTRACTION

