Operating System

Tassadaq Hussain

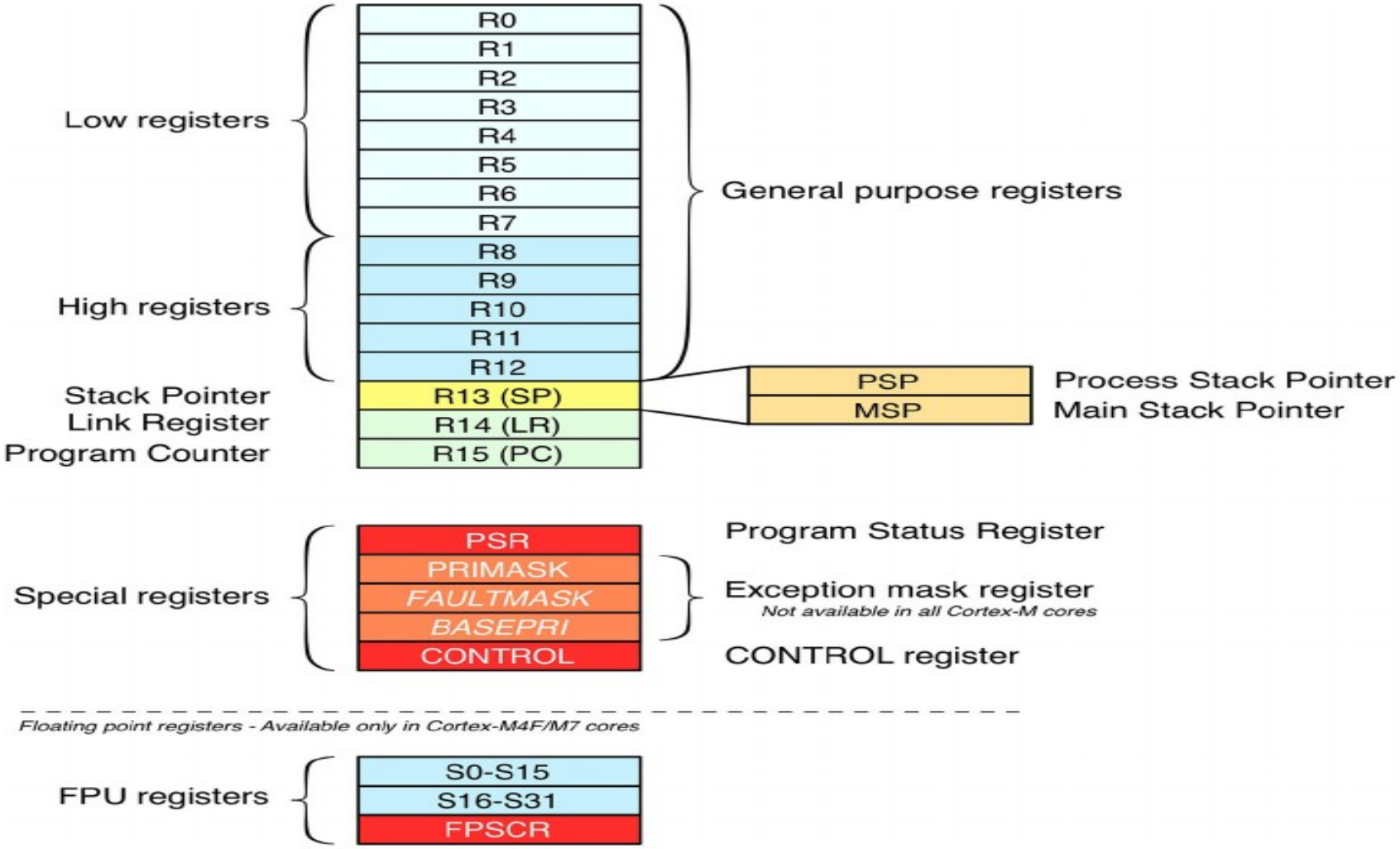
Assistant Professor: Riphah International University

Research Collaborations: Microsoft Barcelona Supercomputing Center

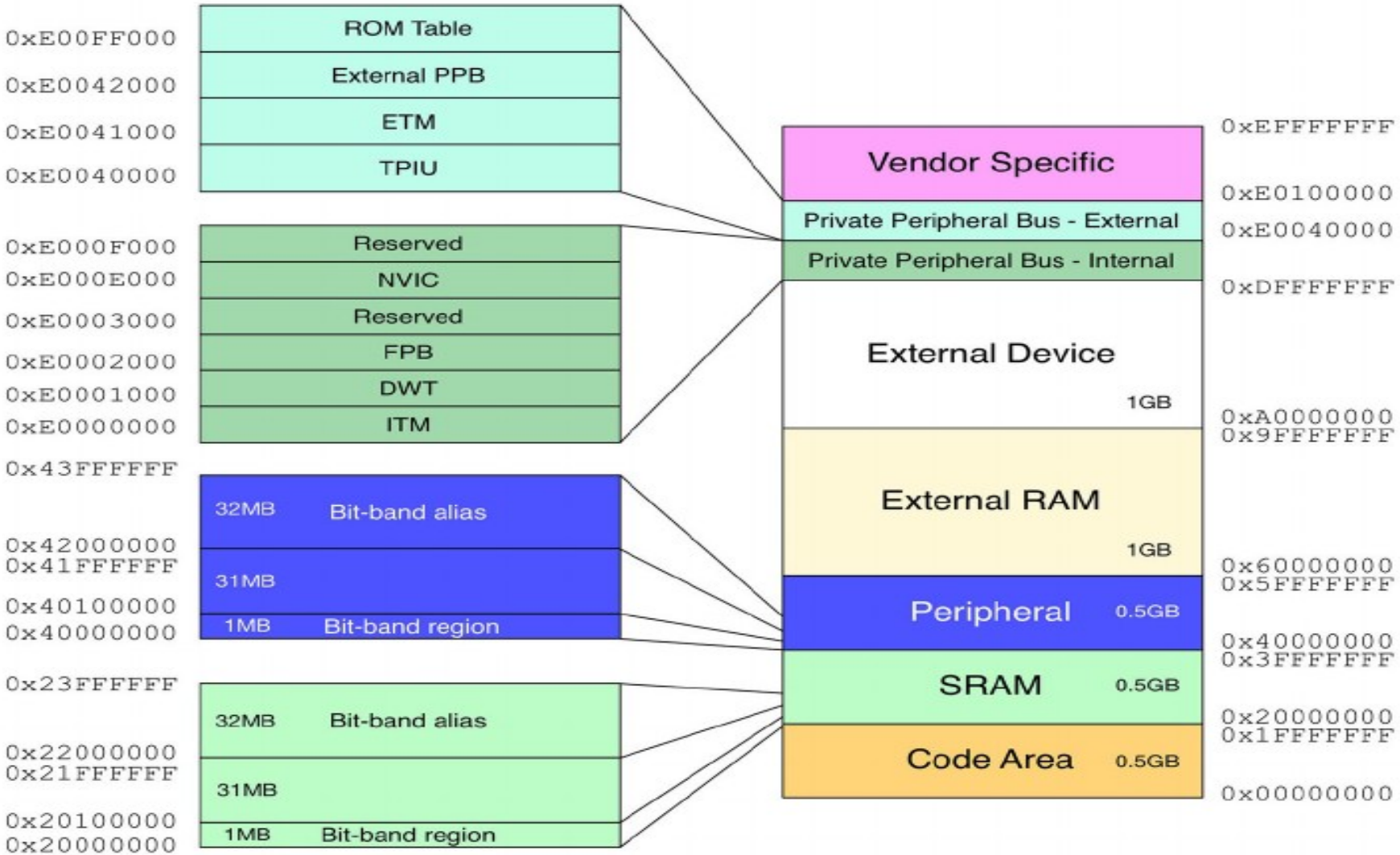
University of Valenciennes, France (CNRS UMR)

UCERD Pvt Ltd Pakistan

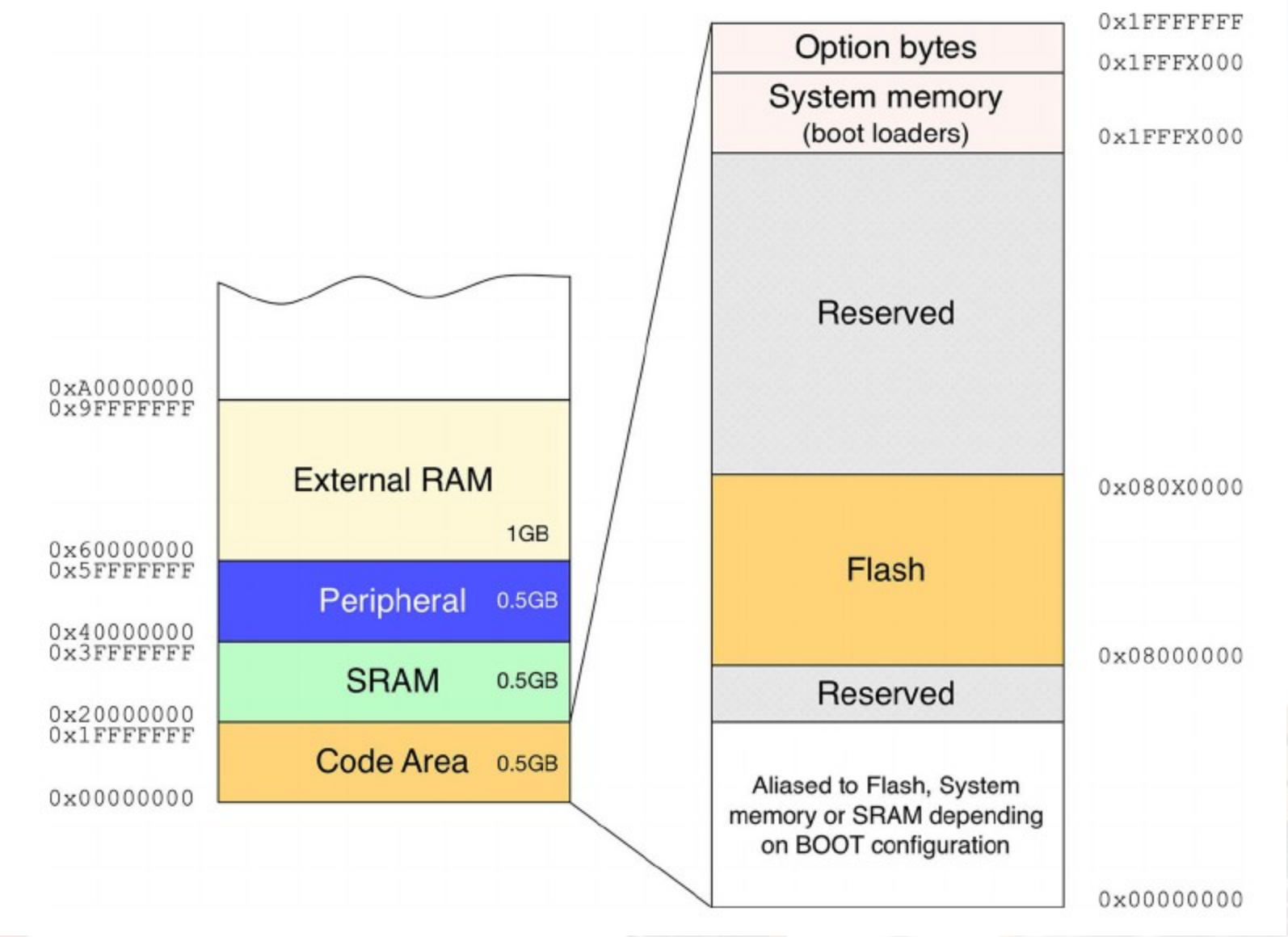
Core Registers



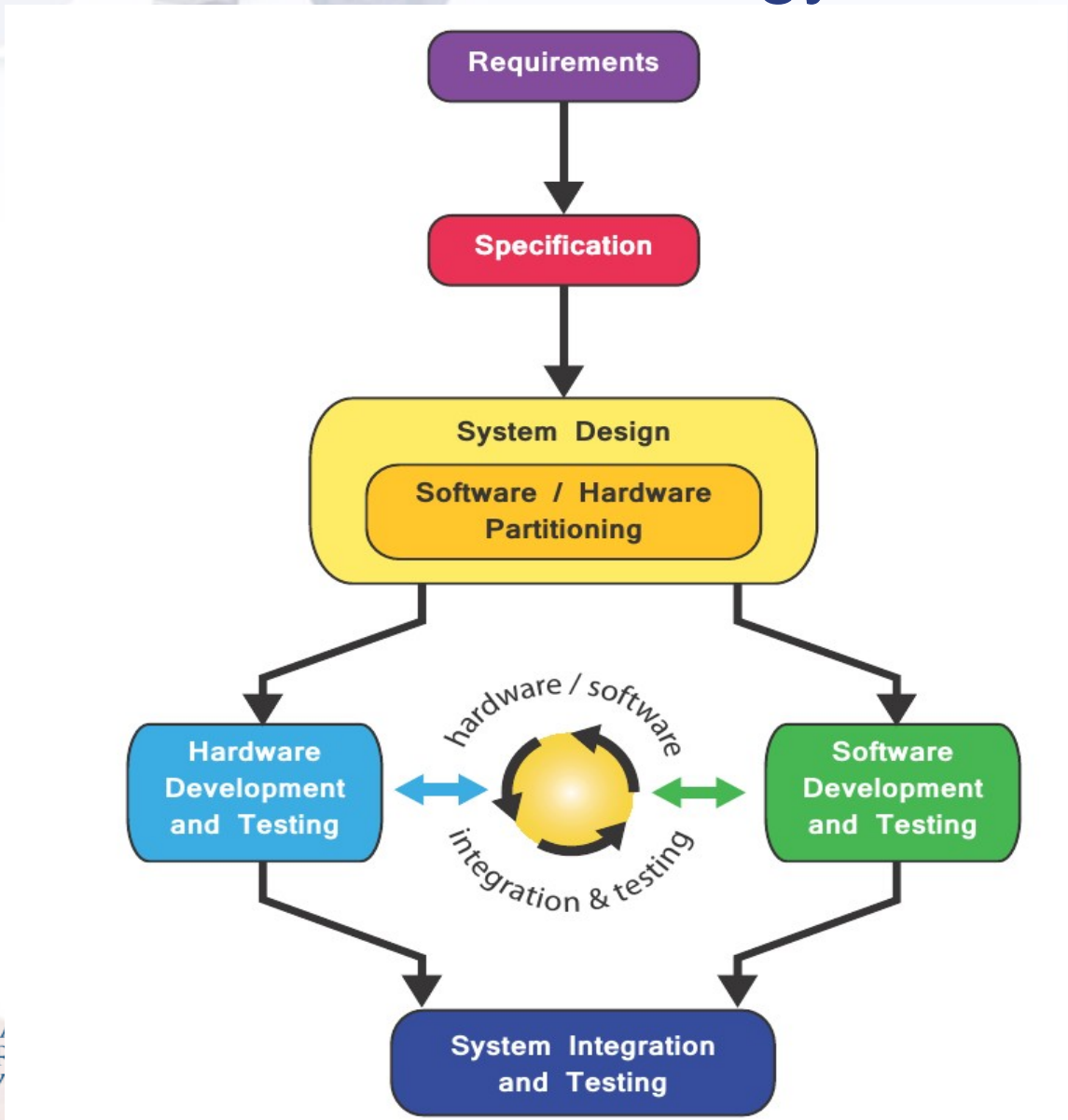
Cortex-M Fixed Memory Map



Code Area STM32 MCUs



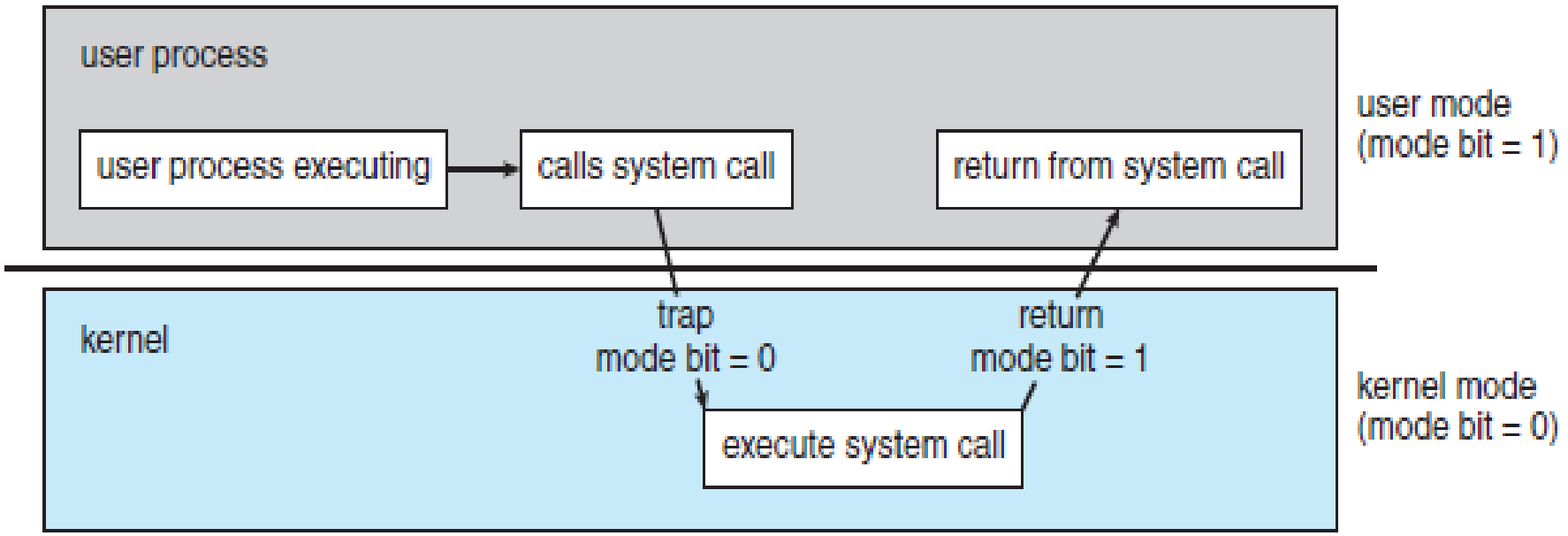
A general hardware-software codesign methodology



Hardware Abstraction Layer

Hardware Abstraction Layer (HAL) provides function API-based service to the higher-level layers (eg: Application Framework, customer application,Et cetera) that allows them to perform hardware-oriented operations independent of actual hardware details.

User, Kernel modes



- At the very least, we need two separate ***modes*** of operation: **user mode** and **kernel mode** (supervisor mode, protected mode, privileged mode)
- **Kernel mode – privileged instructions**
- At system **boot** time, the hardware starts in **kernel** mode.
- The operating system is then loaded and starts **user applications in user mode**.
- user application **requests a service** from the operating system (via a system call), the system must perform transition **from user to kernel mode** to fulfill the request.

Privileged instructions

* The hardware allows privileged instructions to be executed **only in kernel mode.**
* If an **attempt** is made to **execute a privileged instruction in user mode**, the hardware does not execute the instruction but rather treats it as **illegal and traps** it to the operating system.

  * I/O control
  * Interrupt management
  * Set value of timer.
  * Clear memory.
  * Turn off interrupts.
  * Access I/O device.

Privileged

Non Privileged
Non Privileged

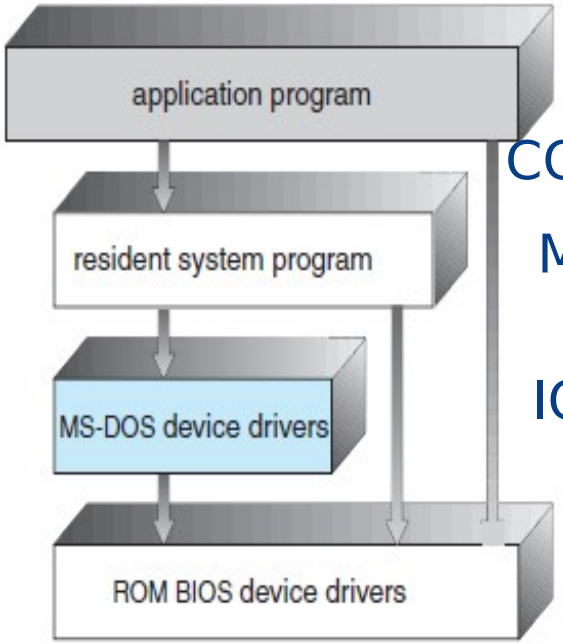
Timer

- We must **ensure** that the operating system **maintains control** over the CPU.
  - We cannot allow a **user program** to get stuck in an **infinite loop**
  - or to fail to call system services and **never return control** to the operating system.
- To accomplish this goal, we can use a **timer**.
- A timer can be set to **interrupt** the computer after a **specified period**.
  - The period may be **fixed** (for example, 1/60 second)
  - or **variable** (for example, from 1 millisecond to 1 second).
- The operating system **sets the counter**.
  - Every time the clock ticks, the counter is **decremented**.
  - When the counter **reaches 0**, an **interrupt** occurs.

Operating System Structure MSDOS

Modular or Monolithic

- MS-DOS could be considered as monolithic ?
  - Only drivers could be added to the system
  - OS itself was one big static program (never changes)
- No protection mode
  - In software
  - In hardware also (at that time)



COMMAND.COM - shell

MSDOS.SYS - kernel

IO.SYS - drivers

Monolithic kernel

Benefits:

Fast

All modules are in the same address space. All calls are in the same address space. That is fast.

Disadvantages:

Large code – with the time it becomes larger and larger

Error prone (not reliable)

Difficult to manage

Difficult to update, not flexible

Each change needs the program (kernel) to rebuild.

Difficult to develop

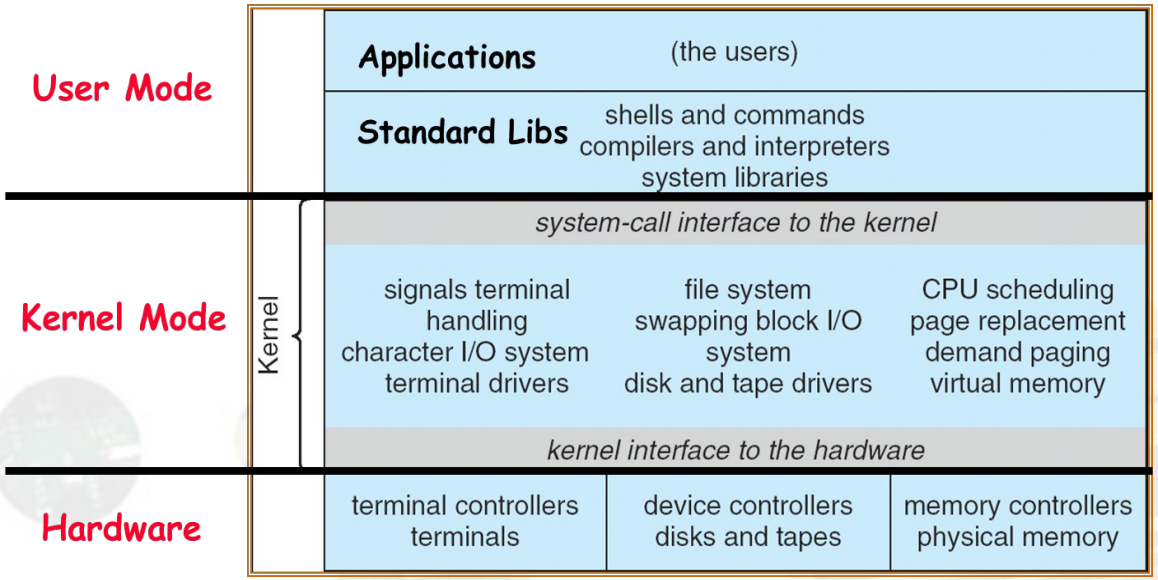
Solution Make OS modular

Layering the OS

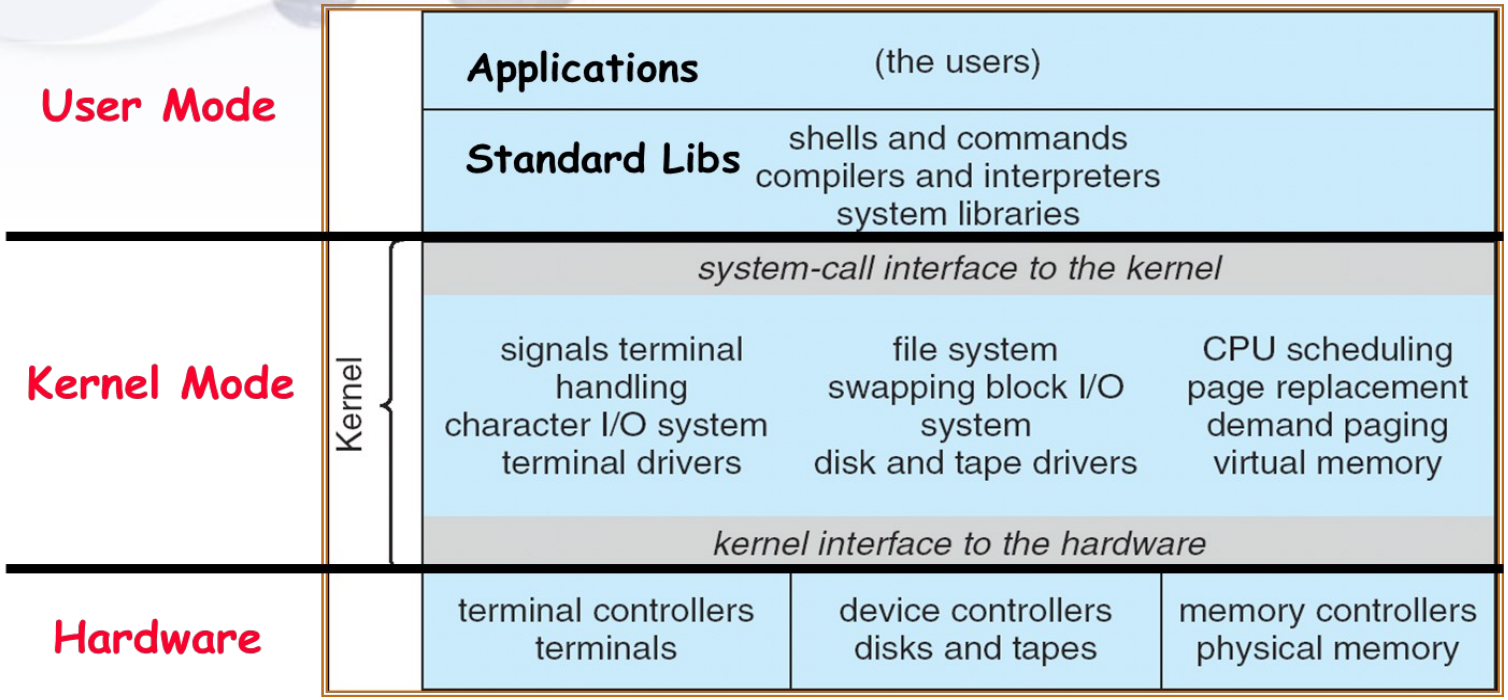
Microkernel

Loadable Kernel modules

UNIX System Structure



UNIX System Structure



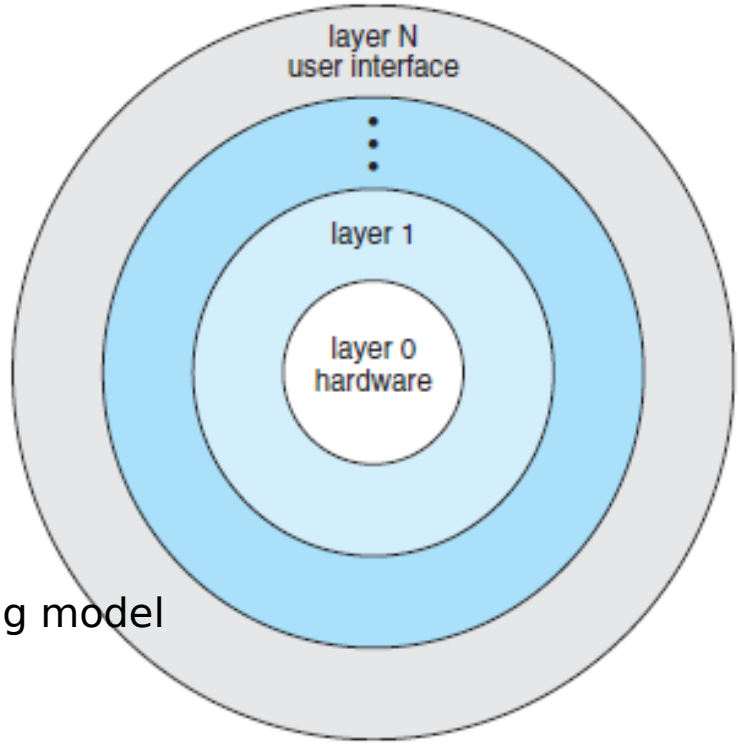
- The modular kernel provides
  - The file system
  - CPU scheduling
  - Memory management
  - and other operating-system functions through system calls.

A layered operating system

Modular or Monolithic

- A system can be made modular in many ways.
- One method is the **layered approach**
  - the operating system is broken into a number of layers (levels).
  - The **bottom** layer (layer 0) is the **hardware**
  - the **highest** (layer *N*) is the **user interface**.
  - Each layer knows only its neighbors

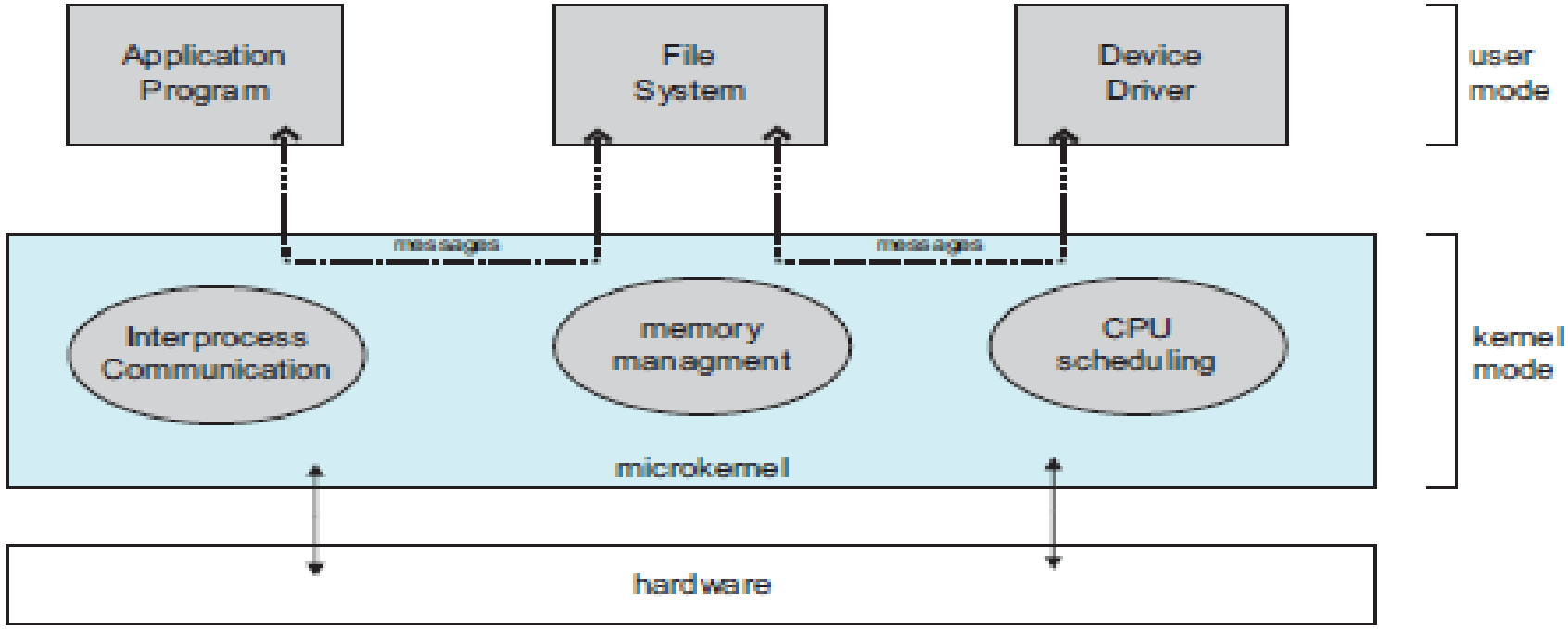
- **advantage** - simplicity of construction and debugging
- **major difficulty** - appropriately define the various layers
- **Disadvantage -** Each layer adds overhead to the system call



Examples
- Hardware abstraction layer (HAL) – Windows
- Open system interconnection (OSI) networking model

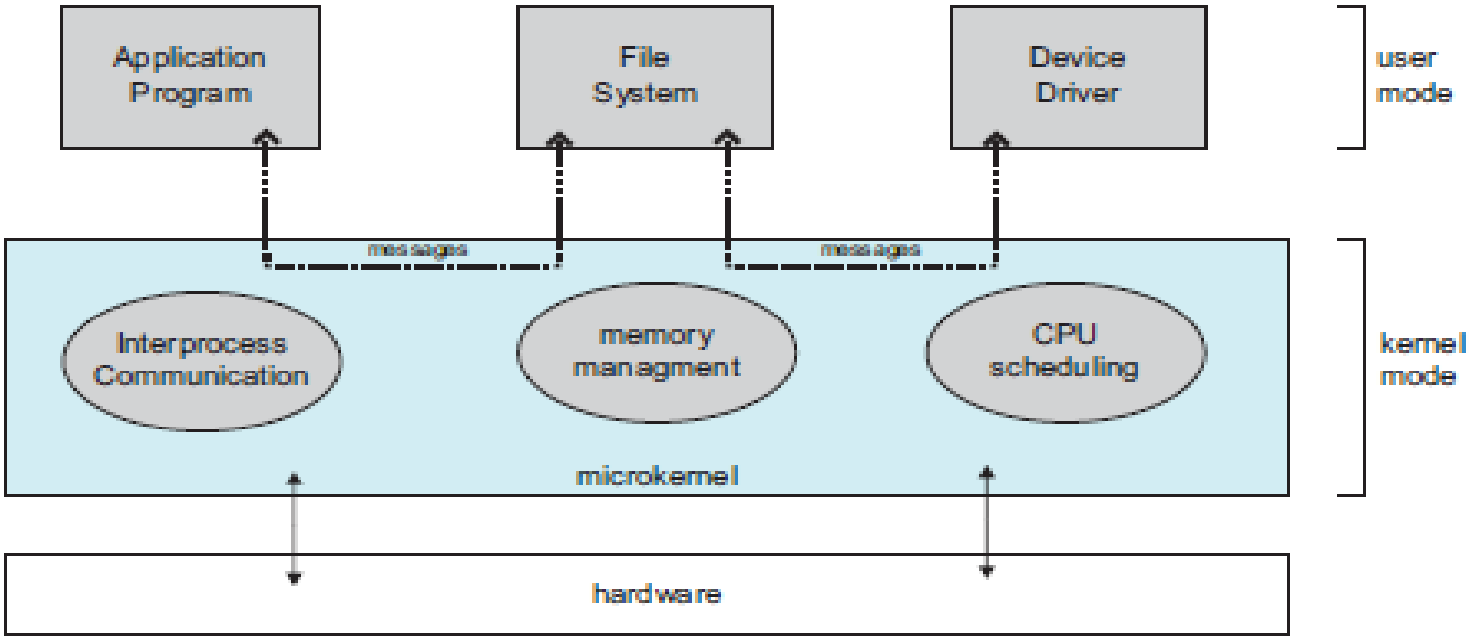
First Microkernel

* UNIX expanded, the kernel became large and difficult to manage.
* In the mid-1980s, researchers at Carnegie Mellon University developed an operating system called **Mach** that modularized the kernel using the **microkernel** approach.



Microkernel structure

- remove all nonessential components from the kernel and implement them as system and user-level programs.
- The result is a smaller kernel.
- Typically **microkernels provide minimal process and memory management, in addition to a communication facility**.

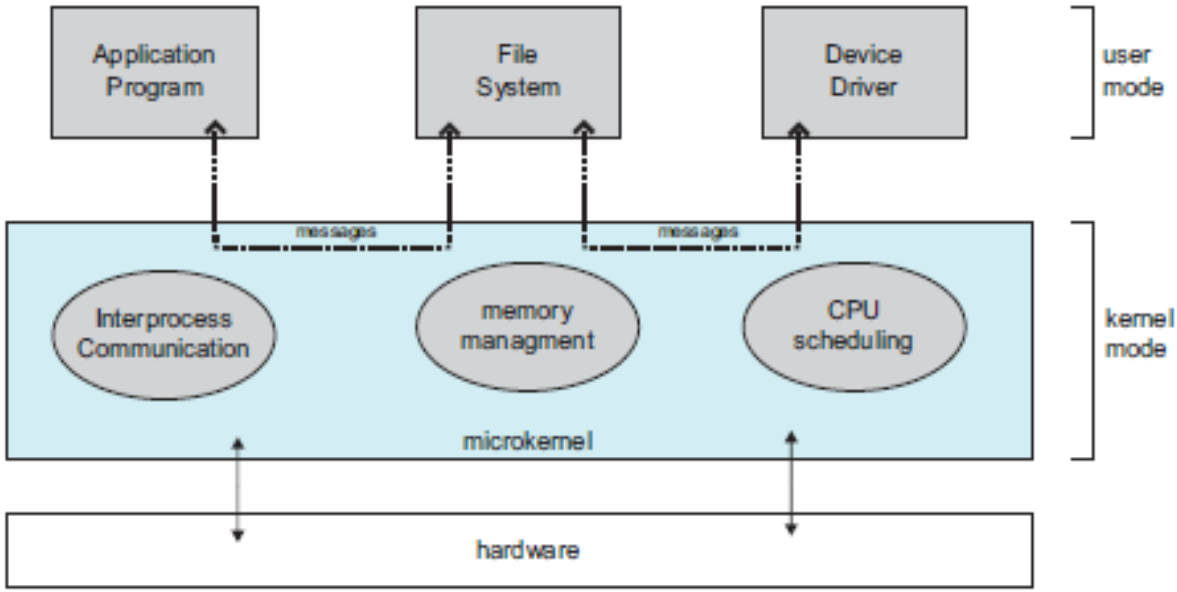


Microkernel tradeoff

Advantages

* Extending the operating system easier
  * All new services are added to user space and consequently do not require modification of the kernel
* Less error prone code, more reliable kernel, fault isolation
  * Small code less errors. Module error does not affect the other parts of OS
* Secure kernel
  * Less code in kernel space, less security breaches.
* Easier to port to different hardware platforms.
  * Less depends on hardware
* Small size

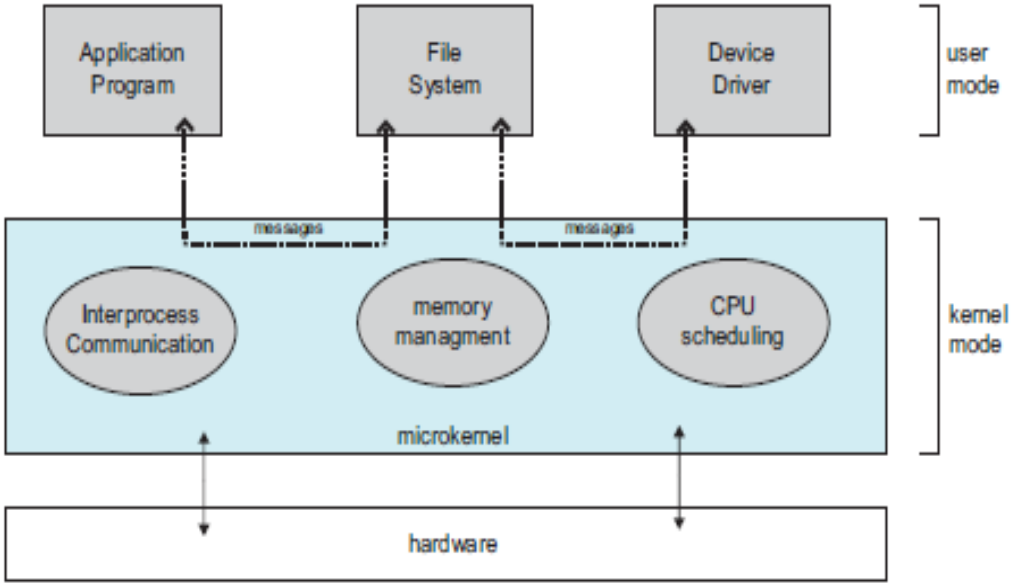
- Mac OS X kernel (also known as **Darwin**)
- QNX, a real-time operating system for embedded systems
  - **Performance ???**



Microkernel Performance

- the **performance of microkernels can suffer** due to increased system-function overhead
  - The main function of the microkernel is to provide communication between the client program and the various services that are also running in user space.

- Communication is provided through **message passing,**
- For example, if the client program wishes to access a file
  - it must interact with the file server.
  - The client program and service never interact directly.
  - Rather, they communicate indirectly by exchanging messages with the microkernel.



Monolithic vs. Microkernel

Benefits:

Fast

All modules are in the same address space

Detriments:

- Large code
- Error prone (not reliable)
- Difficult to manage
- Difficult to update
- Difficult to develop

Moves as much from the kernel into "*user*" space

- Small core OS running at kernel level
- OS Services built from many independent user-level processes
- Communication between modules with message passing

Benefits:

- **Easier to extend OS**
- **Easier to port OS to new architectures**
- **More reliable** (less code is running in kernel mode)
- **Fault Isolation** (kernel and other OS parts are protected from other parts)
- **More secure**

Detriments:

- **Performance overhead severe for naive implementation**

Microkernel Structure

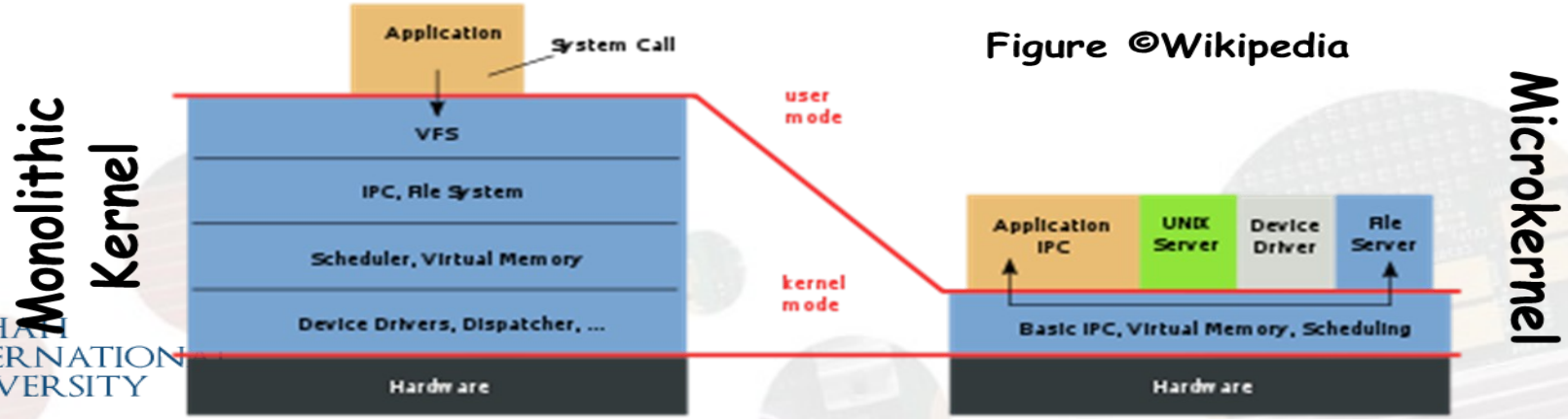


Figure ©Wikipedia

Kernel Modules dynamically linked libraries

* Perhaps the **best current methodology** for operating-system design involves using **loadable kernel modules**.
* The idea of the design is
  * for the **kernel to provide core services**
  * while other services are implemented **dynamically**, as the kernel is running.
* This type of design is common in modern implementations of UNIX, such as Solaris, Linux, FreeBSD and Mac OS X, as well as Windows.

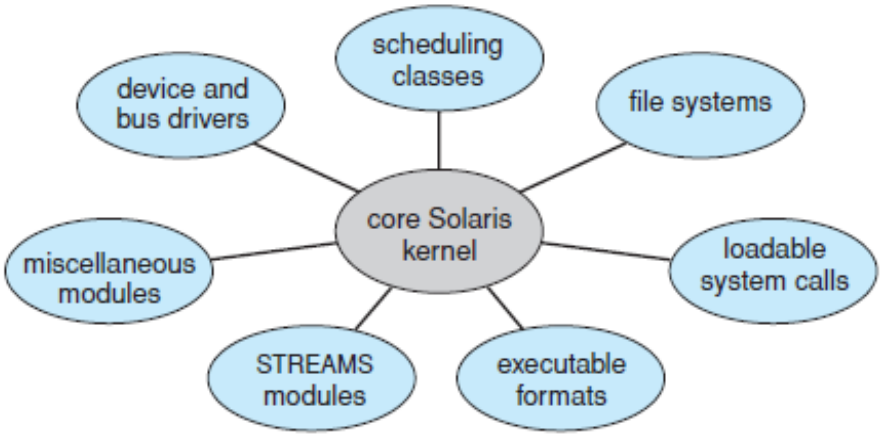
- *.dll instead of *.lib in windows
- *.so instead of *.a in Unix type systems.

Advantages:

- No kernel rebuilding for updates
- Much easier to create new modules rather than new layers

Disadvantages:

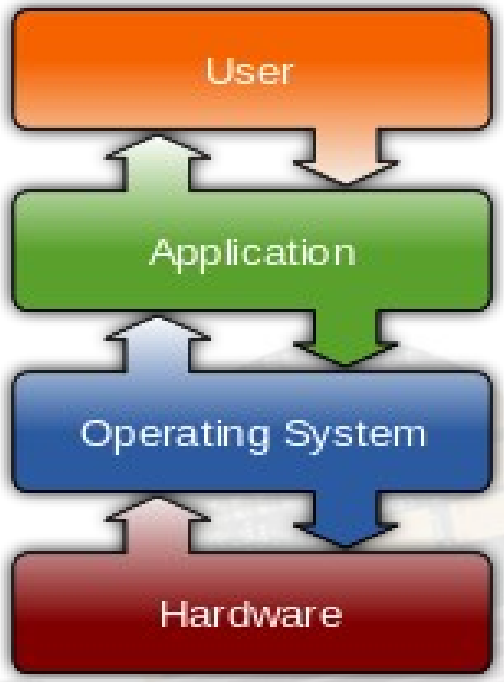
Run Time linking takes "Run Time"



Operating System

An operating system (OS) manages computer hardware and software resources. It includes:

1) Task manager
2) Memory manager
3) Device manager
4) File manager
5) Network manager



Operating System: Blocks