CS 450: Operating Systems
Lecture 2: OS Structures

Spring 2014, J. Sasaki
Dept of Computer Science
Illinois Institute of Technology

What is an OS?

*“**Operating System**: The software that supports a computer's basic functions, such as scheduling tasks, executing applications, and controlling peripherals.”*

— New Oxford American Dictionary

“An operating system (OS) is a collection of software that manages computer hardware resources and provides common services for computer programs.”

— Wikipedia

*"An **operating system** acts as an intermediary between the user of a computer and the computer hardware. The purpose of an operating system is to provide an environment in which a user can execute programs in a **convenient** and **efficient** manner."*

— Dinosaur book, page 1

"A Unix-like operating system is a software collection of applications, libraries, and developer tools, plus a program to allocate resources and talk to the hardware, known as a kernel...

“… GNU is typically used today with a kernel called Linux. This combination is the GNU/Linux operating system. GNU/Linux is used by millions, though many call it "Linux" by mistake.

— gnu.org homepage

“Why do you call it GNU/Linux and not Linux?

“Most operating system distributions based on Linux as kernel are basically modified versions of the GNU operating system. We began developing GNU in 1984, years before Linus Torvalds started to write his kernel.…

— gnu.org

“Should we always say “GNU/Linux” instead of “Linux”?

“Not always—only when you're talking about the whole system. When you're referring specifically to the kernel, you should call it “Linux”, the name its developer chose.

— gnu.org

Kernel vs Non-Kernel

- Why have a kernel?
  - Manageability
  - Separation of functions
  - Protection and security
    - Supervisor (“kernel”) mode vs user mode
    - The kernel is everything that runs in kernel mode?

- Kernel provides
  - Basic interface between software and hardware
  - Lowest level of abstraction for resources
    - Processes, virtual memory, device access, communication
- Access kernel via system calls

Process Management

- Process = Program + its memory + control info
  - Address space, process's runtime stack
  - Schedule CPU usage by process
    - Multitasking (*cf*. uni-tasking/batch)
    - Cooperative vs Preemptive
    - Scheduler — policy vs mechanism

Process Usage

- Process operations
  - fork, exec, wait, exit
- Process communication

Process Communication

- IPC — inter-process communication
  - Shared memory
    - Fast, easy to use; cache coherence?
  - Message passing
    - Communicate, synchronize; cost?
  - Remote procedure calls

Remote Procedure Calls

- Call client stub, locally
- Marshal parameters into a message, send message to server
- Server stub un-marshals message into actual arguments, calls server procedure
- … and reply

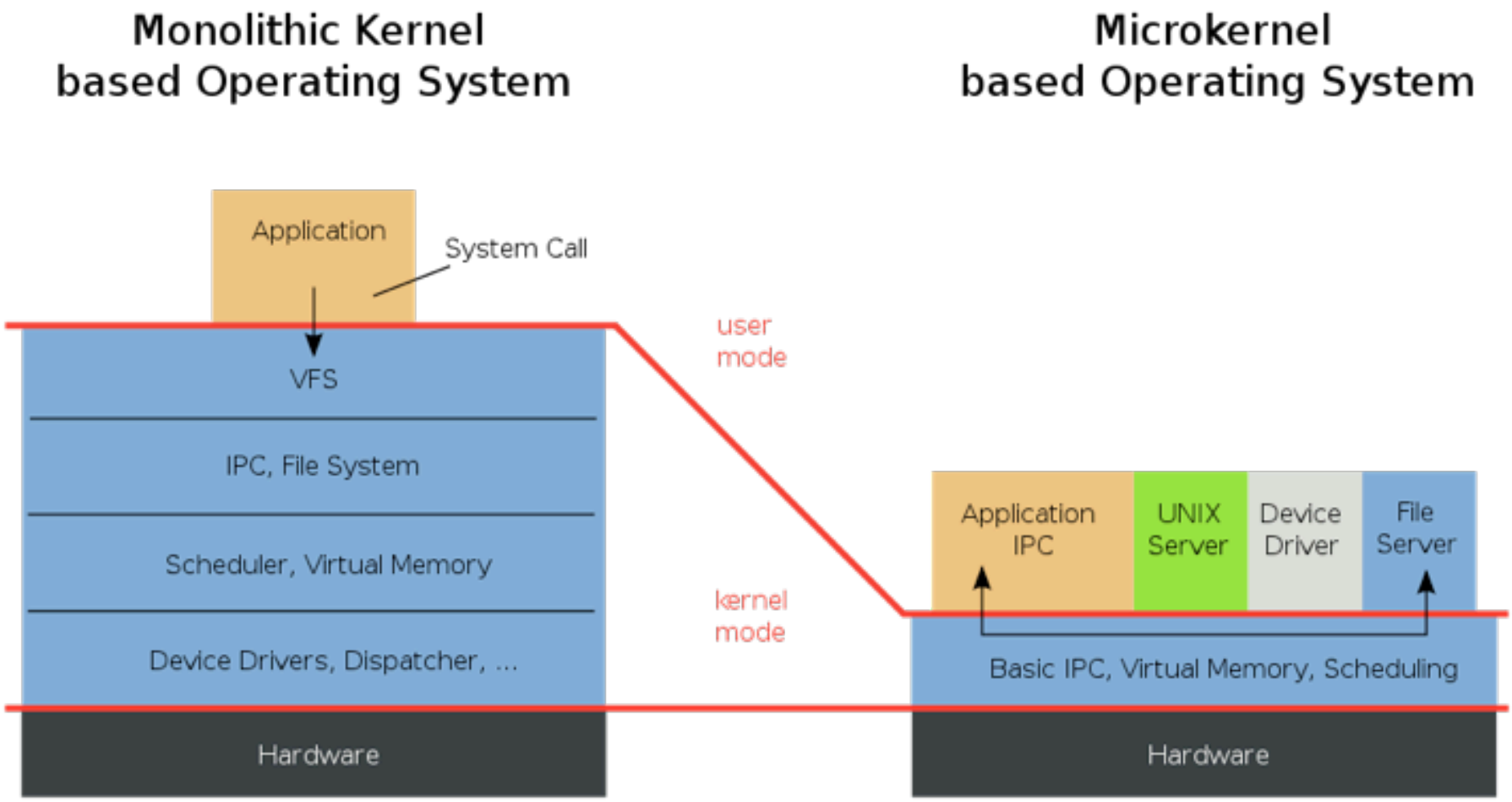
Kernel Organization

- Monolithic kernel
  - Kernel resides in one address space (basically)
  - Intra-kernel operations via function call
  - Fast; few system calls; direct access to shared kernel data
  - Drawbacks: Less robust? Buggier? Large memory footprint?

Microkernel

- Restrict kernel to basic minimum
  - Process support: Address spaces, thread and process scheduling, minimal IPC
- Turn device drivers, file system, higher-level IPC, shell into user-level code
- Nice separation of mechanism & policy, makes protection easier
- Lots of message passing

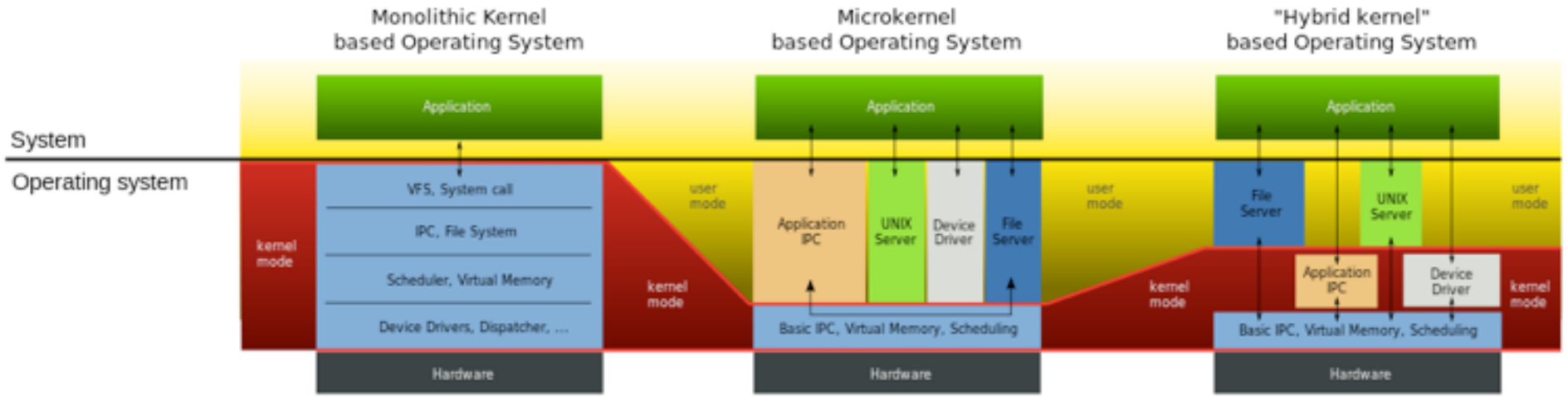
http://en.wikipedia.org/wiki/File:OS-structure.svg



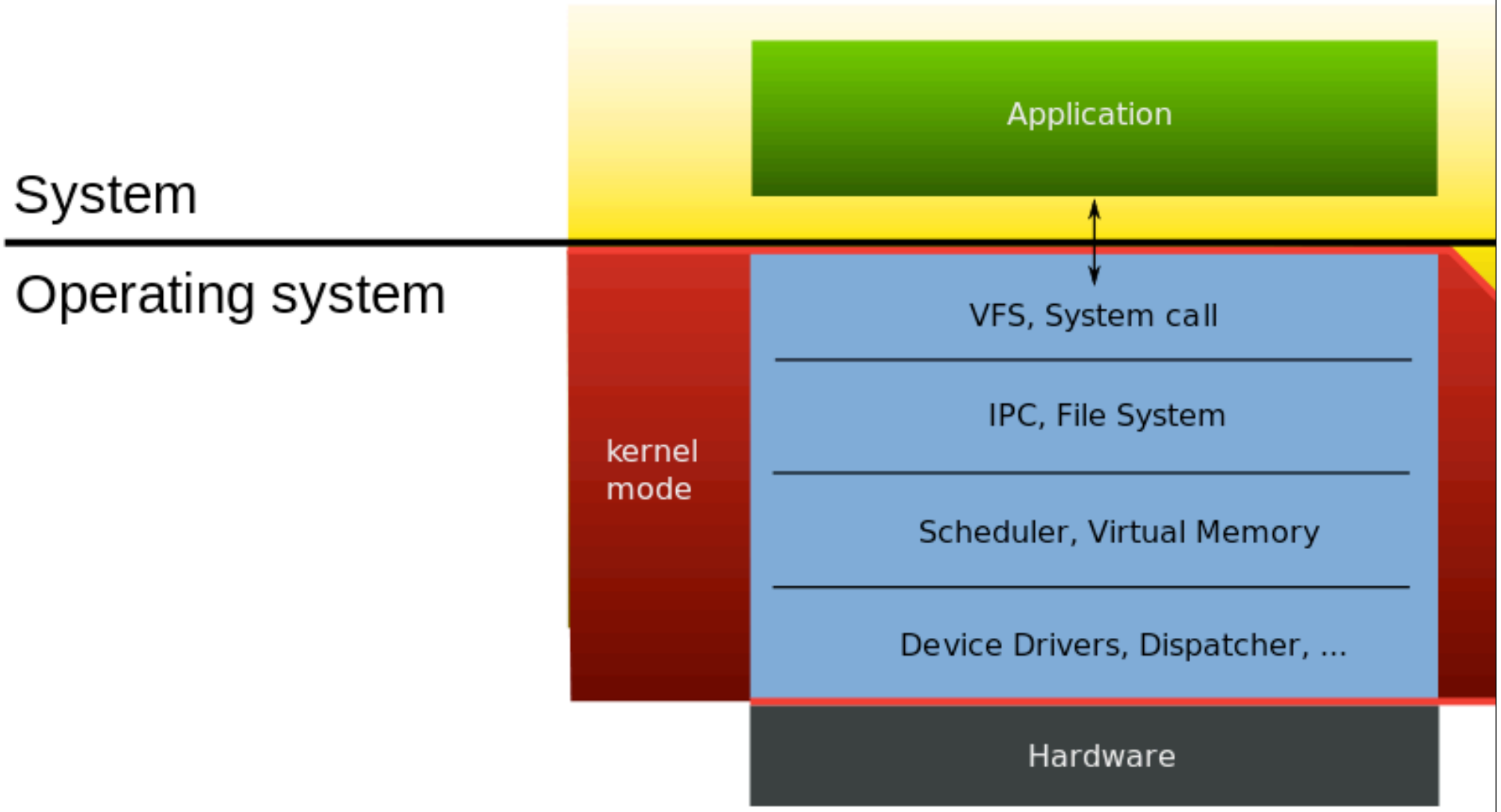
Hybrid Kernel

- Run more OS services in kernel mode than in a microkernel, fewer than in a monolithic kernel
  - More efficiency, give up some reliability benefits

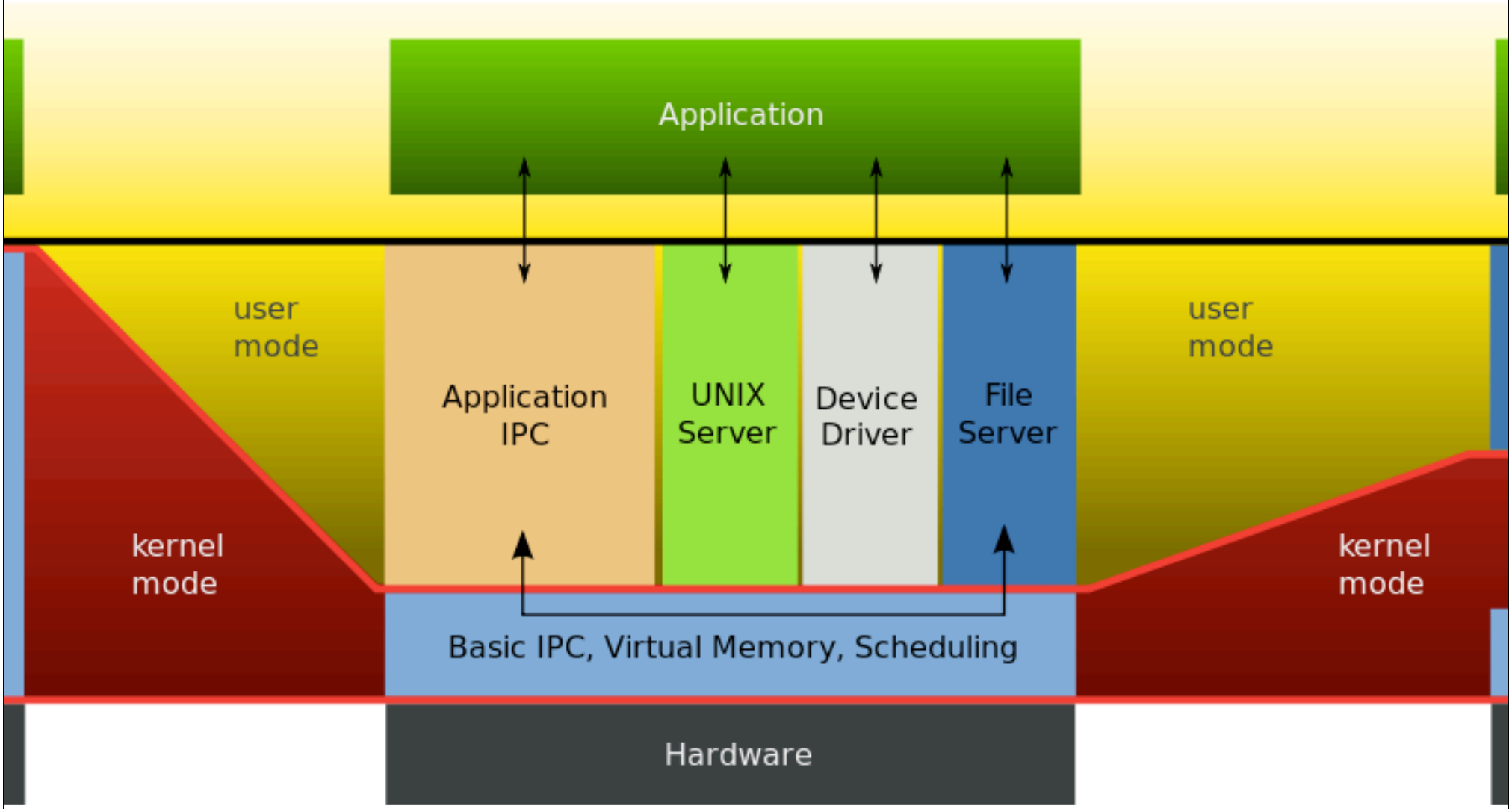
http://en.wikipedia.org/wiki/File:OS-structure2.svg



Monolithic Kernel based Operating System



Microkernel based Operating System



stem

"Hybrid kernel"
based Operating System