Jan.13.2019

1.1 What Operating Systems Do

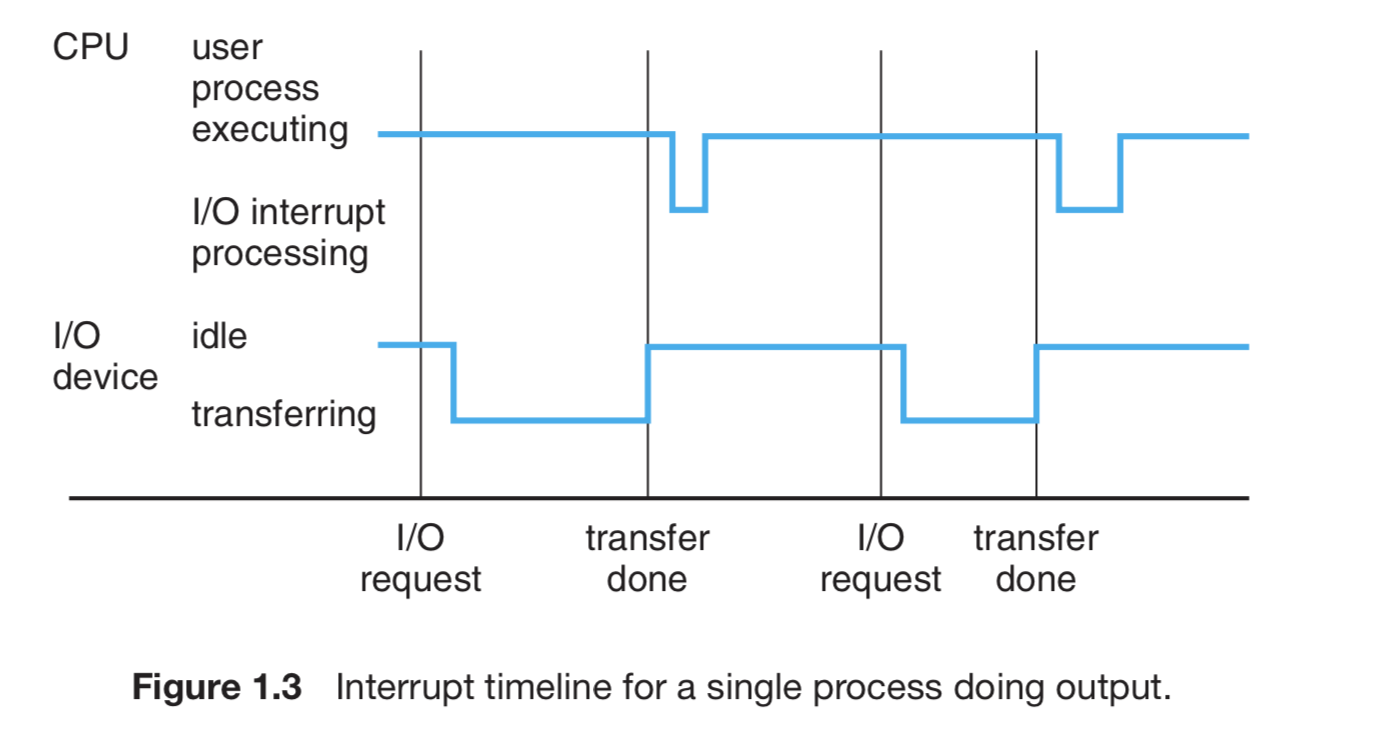
A more common definition, and the one that we usually follow, is that the operating system is the one program running at all times on the computer—usually called the **kernel**. (Along with the kernel, there are two other types of programs: **system programs**, which are associated with the operating system but are not necessarily part of the kernel, and application programs, which include all programs not associated with the operation of the system.)

1.2 Computer-System Organization

For a computer to start running—for instance, when it is powered up or rebooted—it needs to have an initial program to run. This initial program, or **bootstrap program**, tends to be simple. Typically, it is stored within the computer hardware in read-only memory (**ROM**) or electrically erasable programmable read-only memory (**EEPROM**), known by the general term **firmware**. It initializes all aspects of the system, from CPU registers to device controllers to memory contents.

Once the kernel is loaded and executing, it can start providing services to the system and its users. Some services are provided outside of the kernel, by system programs that are loaded into memory at boot time to become **system processes**, or **system daemons** that run the entire time the kernel is running. Once this phase is complete, the system is fully booted, and the system waits for some event to occur.

The occurrence of an event is usually signaled by an **interrupt** from either the hardware or the software(executing a special operation called a **system call** (also called a **monitor call**) ).



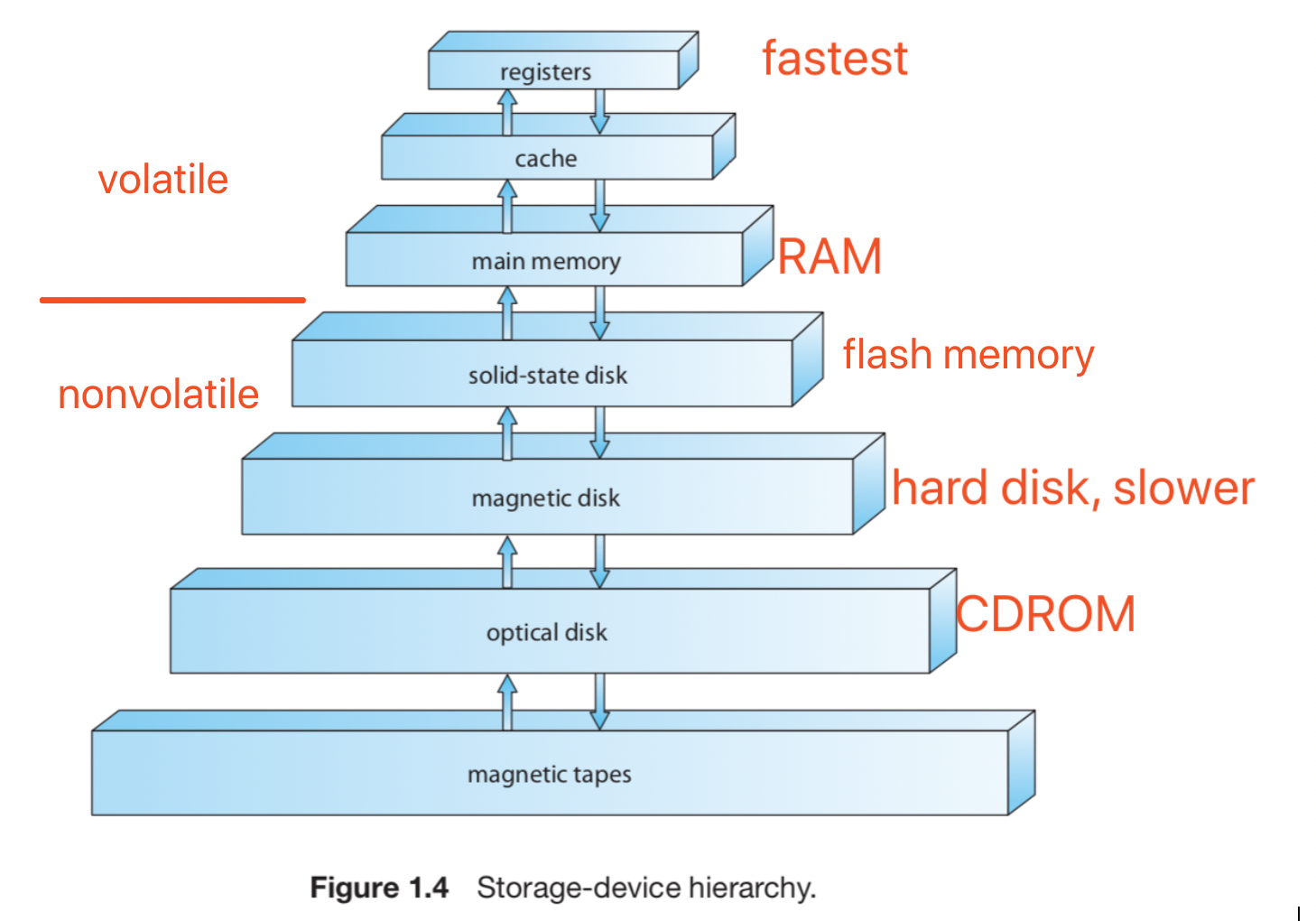
Interrupts must be handled quickly. Since only a predefined number of interrupts is possible, a table of pointers to interrupt routines can be used instead to provide the necessary speed. The interrupt routine is called indirectly through the table, with no intermediate routine needed. Generally, the table of pointers is stored in low memory (the first hundred or so locations). These locations hold the addresses of the interrupt service routines for the various devices. This array, or **interrupt vector**, of addresses is then indexed by a unique device number, given with the interrupt request, to provide the address of the interrupt service routine for the interrupting device.

General-purpose computers run most of their programs from rewritable memory, called main memory (also called **random-access memory**, or **RAM**). Main memory commonly is implemented in a semiconductor technology called **dynamic random-access memory (DRAM)**.

Interaction is achieved through a sequence of load or store instructions to specific memory addresses. The load instruction moves a byte or word from main memory to an internal register within the CPU, whereas the store instruction moves the content of a register to main memory. Aside from explicit loads and stores, the CPU automatically loads instructions from main memory for execution.

A typical instruction–execution cycle, as executed on a system with a **von Neumann architecture**, first fetches an instruction from memory and stores that instruction in the **instruction register**. The instruction is then decoded and may cause operands to be fetched from memory and stored in some internal register. After the instruction on the operands has been executed, the result may be stored back in memory.

The main requirement for secondary storage is that it be able to hold large quantities of data permanently.



Flash memory is slower than DRAM but needs no power to retain its contents.

**NVRAM**: DRAM with battery backup power. This memory can be as fast as DRAM and (as long as the battery lasts) is nonvolatile.

1.2.3 I/O Structure

I/O operation:

Device driver: to start an I/O operation, loads the appropriate registers within the device controller.

Device controller: in turn, examines the contents of these registers to determine what action to take (such as “read a character from the keyboard”). The controller starts the transfer of data from the device to its local buffer.

Once the transfer of data is complete, the device controller: informs the device driver via an interrupt that it has finished its operation.

Device driver: then returns control to the operating system, possibly returning the data or a pointer to the data if the operation was a read. For other operations, the device driver returns status information.

**direct memory access (DMA)** is used for bulk data movement such as disk I/O.

1.3 Computer-System Architecture

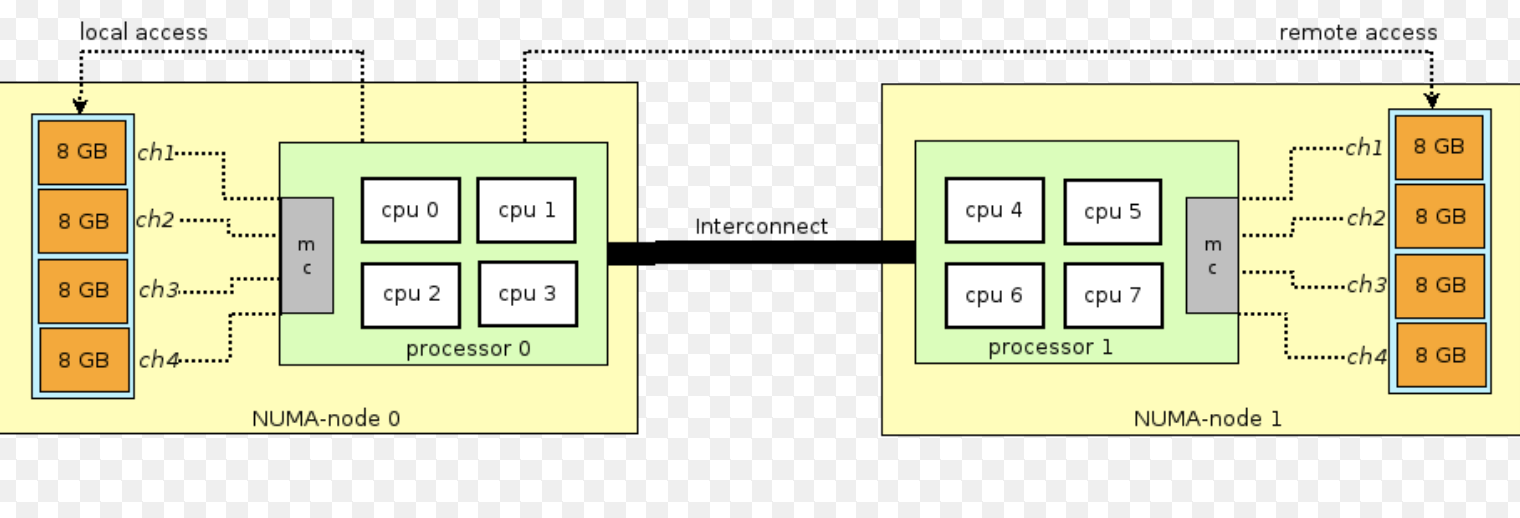
Timer

Multiple cpu in a chip (share a memory) benefits: faster, cheaper, more accurate

Asymmetric multiprocessing v.s. **symmetric multiprocessing**

Blade servers

Multicore: faster than multiple cpu



NUMA

**distributed lock manager (DLM)**

trap

mode

longterm schedule

Jan.20.2019

Operating system: Process

File

Device

Info maintain

Communication

Protection

Simple(graph)

Layered

Micro kernels

Modules

Hybrid

PCB:

Process state

Program counter(which line is executing)

CPU scheduling info(priority)

Memory management

Accounting info(how much cpu)

I/o info

State for cpu

New -> ready(go into ready queue) -> running -> terminated

Running –(i/o request)-> waiting(go into wait queue, store pcb somewhere) -> ready

Running –(cpu call , aka interrupt)-> ready

Long term scheduler

Inter \*\* processing

**Sharing memory(sushi belt)**

Bounded buffer / unbounded buffer

Lock

unbounded buffer:



Multiple consumer & producers

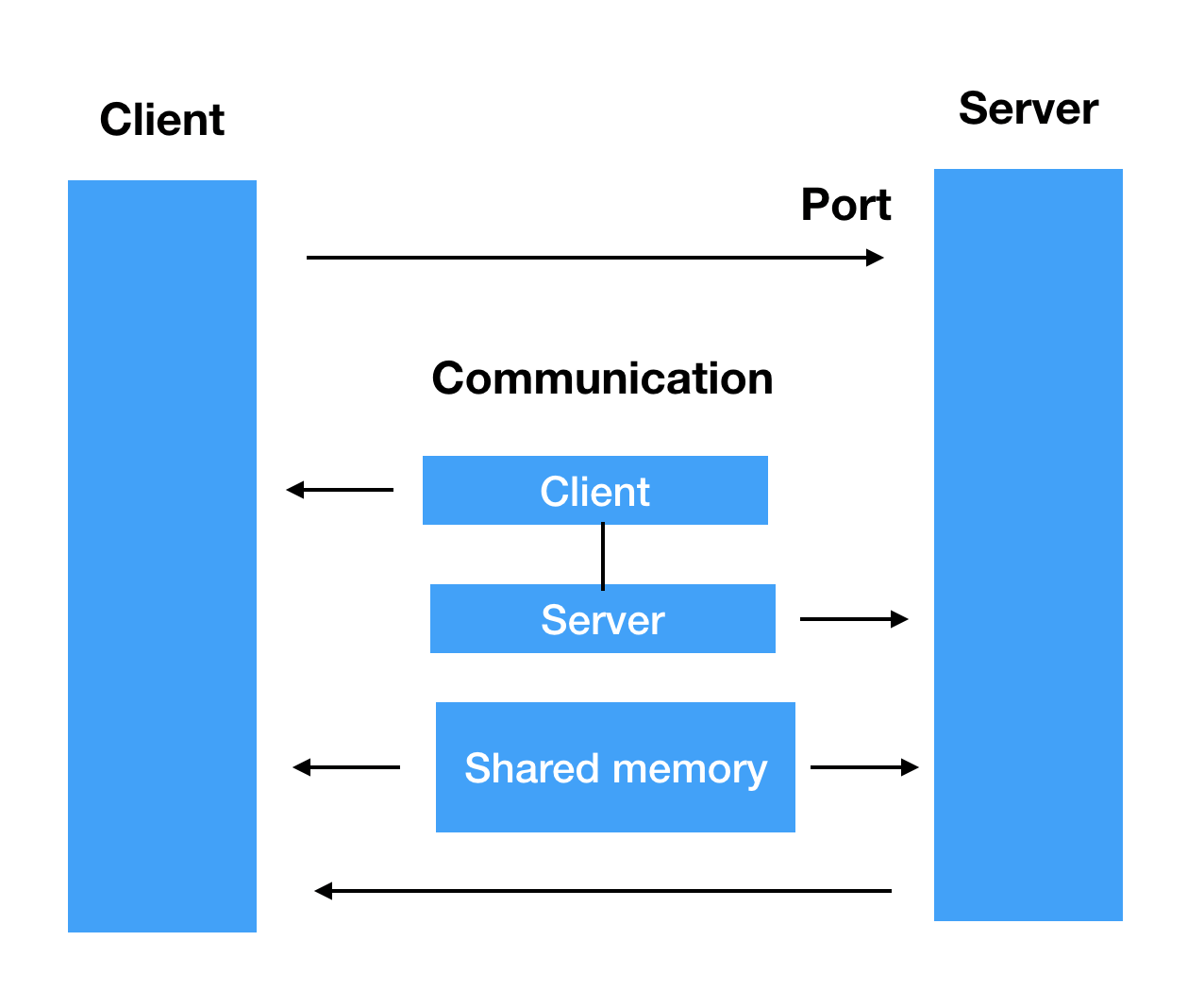


Sending message: direct or indirect

Direct: send(p1, msg1). Request(p2, , msg);

Indirect: mailbox

Synchronous / asynchronous(e.g. posting on blackboard)



Cpu utilization

Throughput 

Turn around time

Waiting time

Response time

First come first serve v.s. Shortest job first