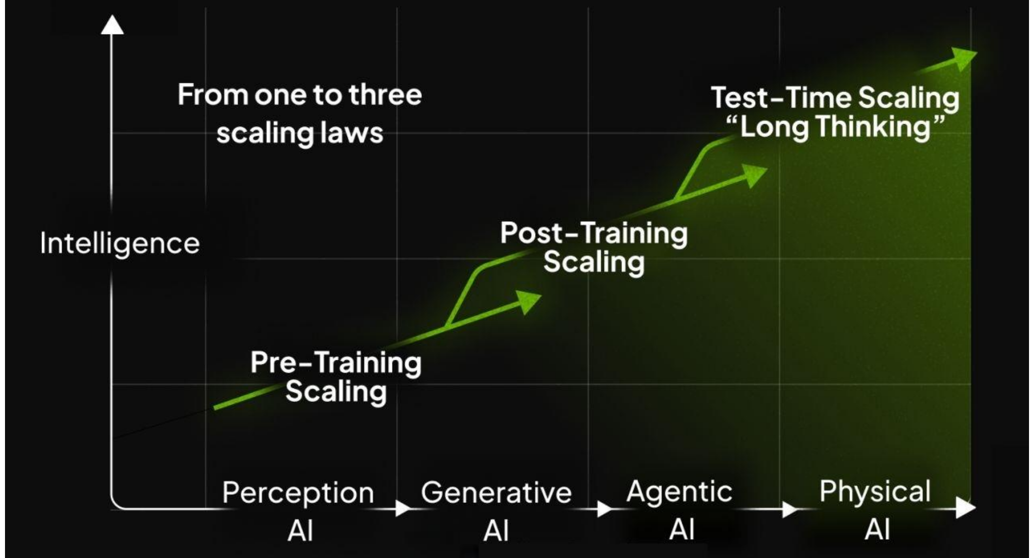
Part 6

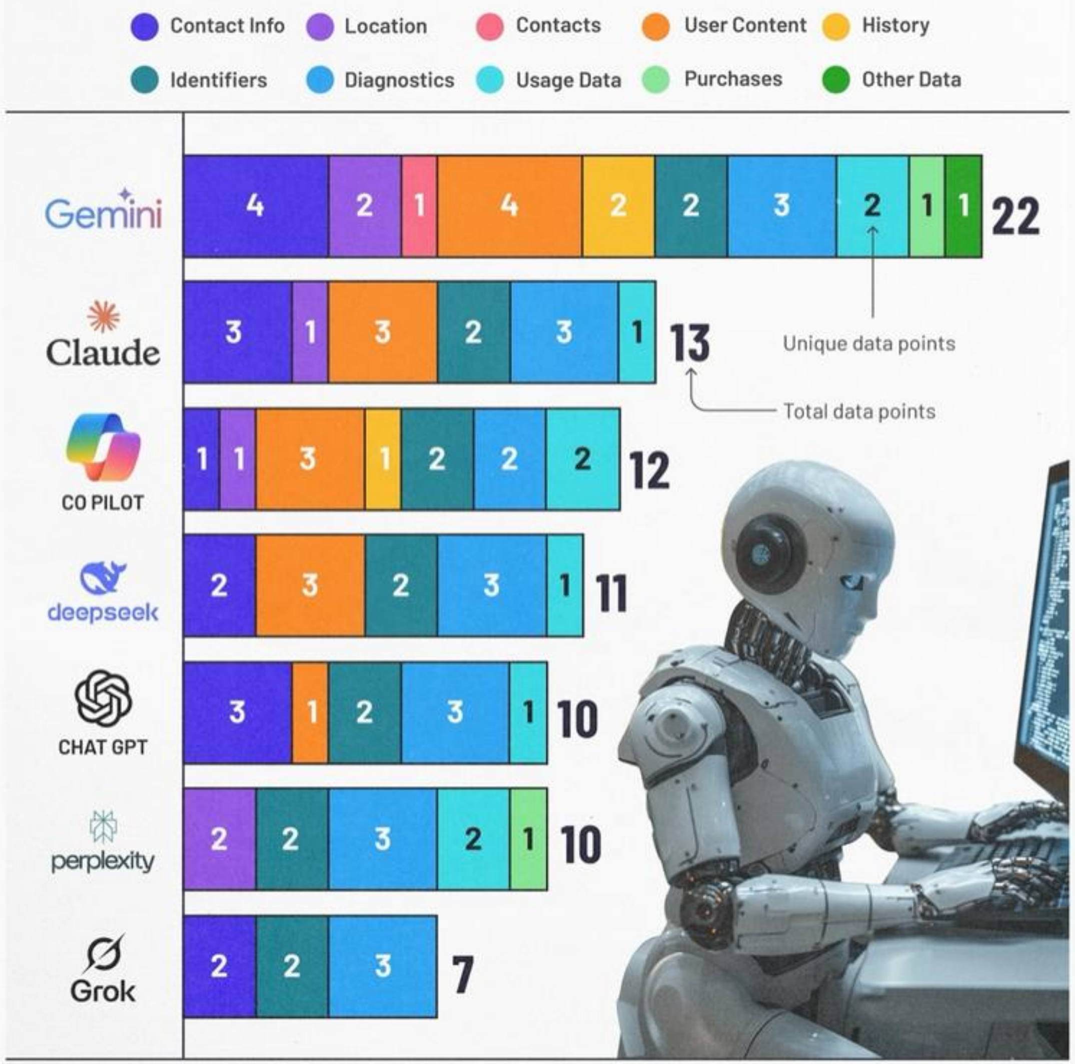
NVIDIA Scaling Principles



Autonomy	Humans: 80% AI: 20%	Humans: 60% AI: 40%	Humans: 30% AI: 70%	Humans: 10% AI: 90%
Compute Need	Lower	Medium	Higher	Highest
Data Demand	Labeled, Moderate	Unstructured, High	Diverse, Higher	Sensory, Highest
Key Players	:) Affectiva IIElevenLabs	OpenAI DeepMind ANTHROP\C	manus CURSOR Harvey	FIGURE 1X Robotics Waymo

WHAT KIND OF DATA

DO AI CHATBOTS COLLECT?



As of Feb 2025. **Source:** Surfshark

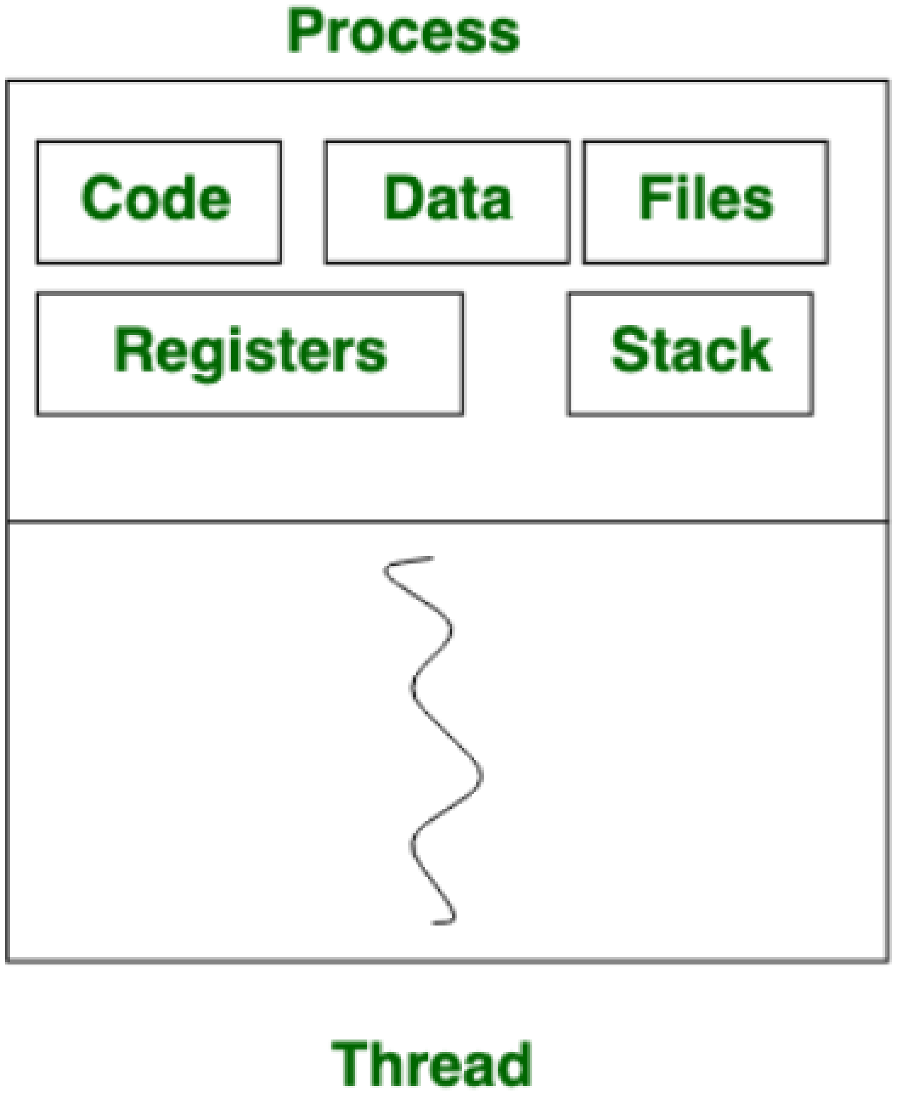
VISUAL CAPITALIST

Overview of Processes and Threads

Thread:

Thread is the segment of a process means a **process** can have **multiple** threads and these multiple threads are contained within a process.

- A thread have **3 states**: **running, ready, and blocked**.
- Thread takes less time to terminate as compared to process



Process

Code

Data

Files

Registers

Stack

Thread

□ What is a Thread?

A thread **is a path of execution within a process**. A process can contain **multiple threads**.

- **Why Multithreading?**
  A thread is also known as ***lightweight process***.
- The idea is to achieve parallelism by dividing a process into multiple threads.
- For example, **in a browser, multiple tabs can be different threads.**
- MS Word uses multiple threads: **one thread to format the text, another thread to process inputs, etc.**

- **Process vs Thread?**
  The primary difference is that threads within the same process run in a **shared** memory space, while processes run in **separate** memory spaces.

- Threads are **not independent** of one another like processes are, and as a result threads share with other threads their code section, data section, and OS resources (like open files and signals).
- But, like process, a thread has its own program counter (PC), register set, and stack space.

Advantages of Thread over Process

1. Responsiveness: If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.

2. Faster context switch: Context switch time between threads is lower compared to process context switch. Process context switching requires more overhead from the CPU.

3. Effective utilization of multiprocessor system: If we have multiple threads in a single process, then we can schedule multiple threads on multiple processor. This will make process execution faster.

4. Resource sharing: Resources like **code, data, and files** can be shared among all threads within a process.
Note: **stack and registers** can’t be shared among the threads. Each thread has its own stack and registers.

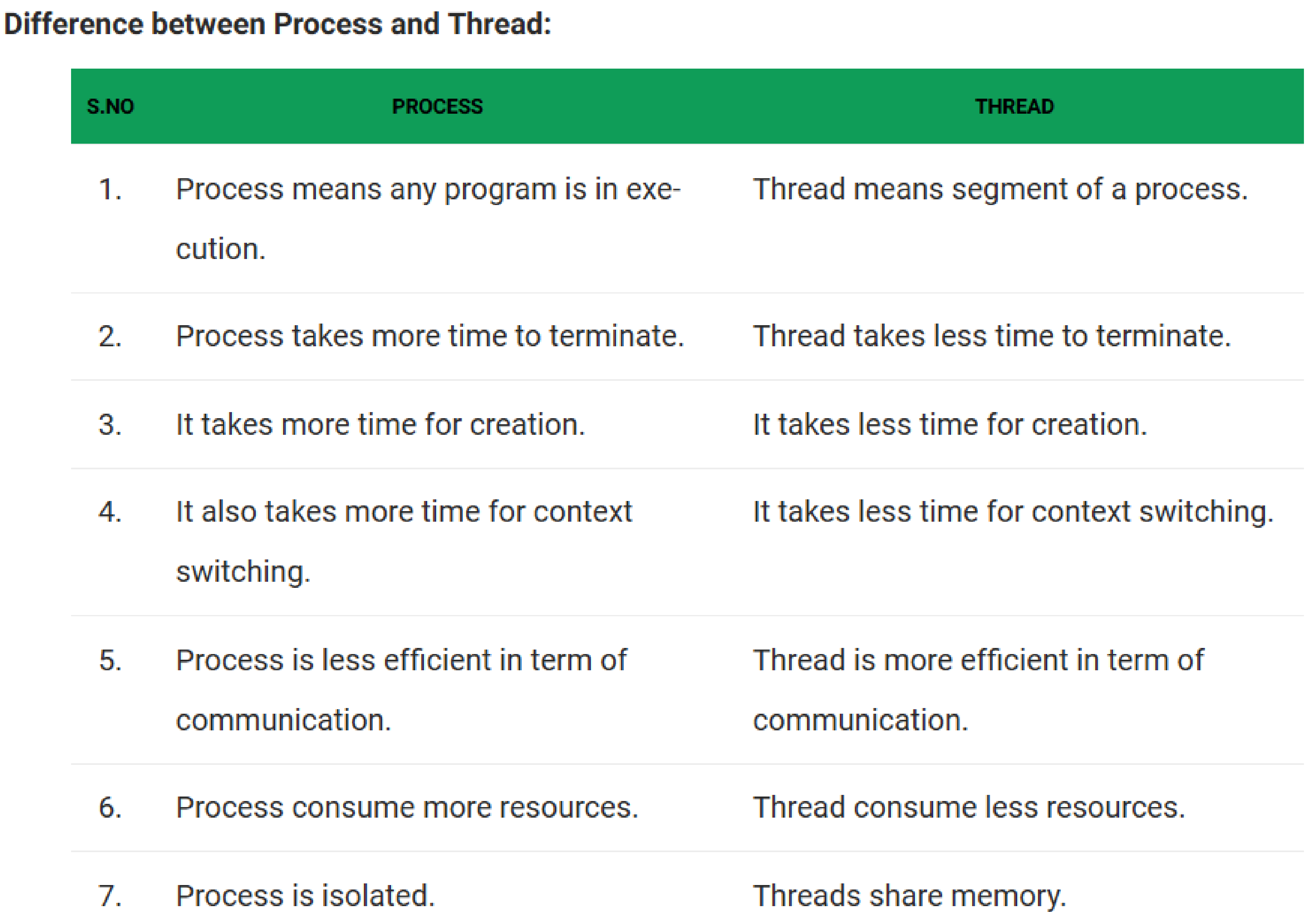
5. Communication: Communication between multiple threads is easier, as the threads **shares common address space**. while in process we have to follow some specific communication technique for communication between two process.

6. Enhanced throughput of the system: If a process is divided into multiple threads, and each **thread function is considered as one job**, then the number of jobs completed per unit of time is increased, thus increasing the throughput of the system.

Types of Threads

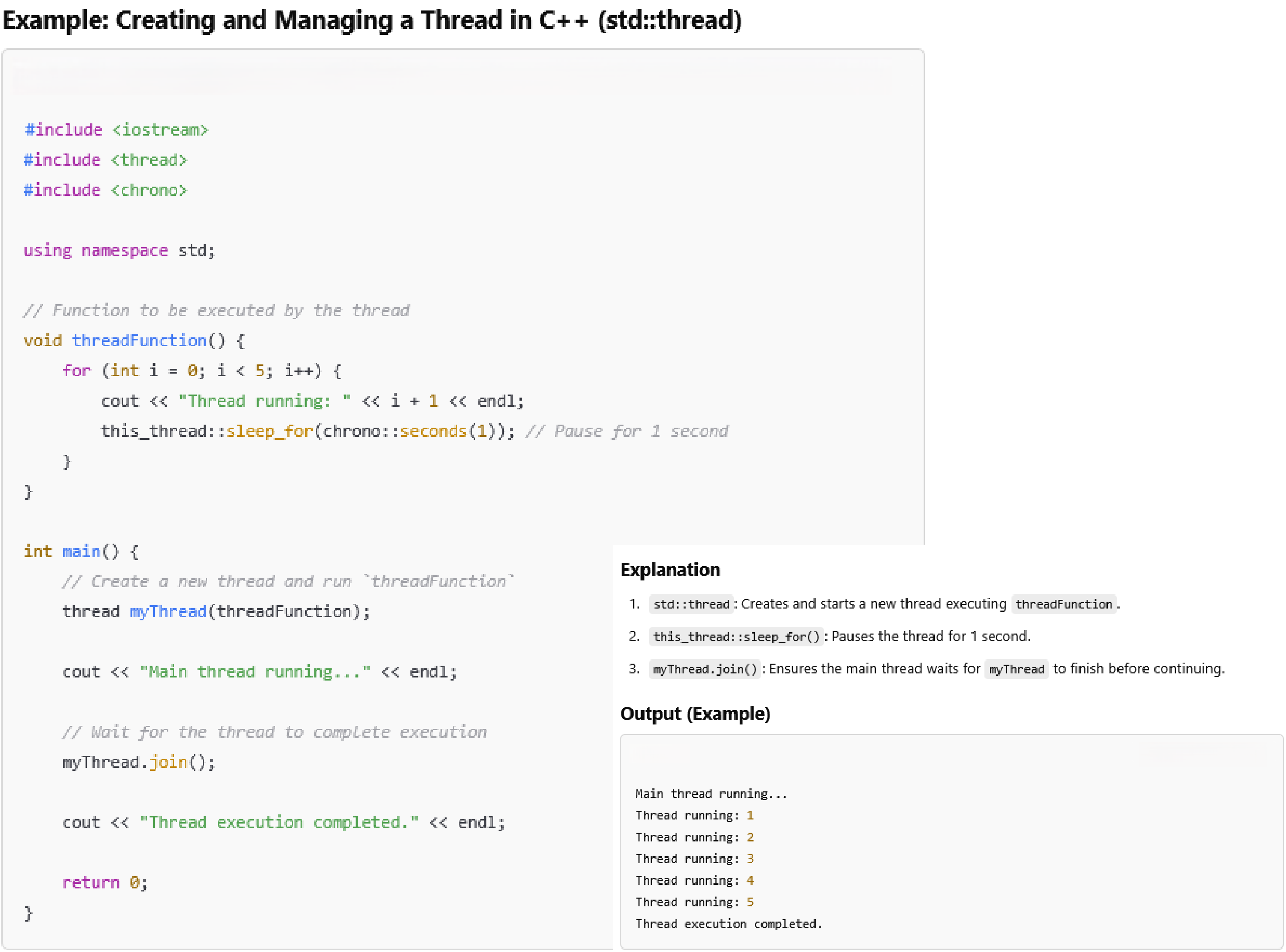
There are **two types** of threads.

- **User Level Thread**
- **Kernel Level Thread**



Difference between Process and Thread:

S.NO	PROCESS	THREAD
1.	Process means any program is in execution.	Thread means segment of a process.
2.	Process takes more time to terminate.	Thread takes less time to terminate.
3.	It takes more time for creation.	It takes less time for creation.
4.	It also takes more time for context switching.	It takes less time for context switching.
5.	Process is less efficient in term of communication.	Thread is more efficient in term of communication.
6.	Process consume more resources.	Thread consume less resources.
7.	Process is isolated.	Threads share memory.



Example: Creating and Managing a Thread in C++ (std::thread)

```cpp
#include <iostream>
#include <thread>
#include <chrono>

using namespace std;

// Function to be executed by the thread
void threadFunction() {
    for (int i = 0; i < 5; i++) {
        cout << "Thread running: " << i + 1 << endl;
        this_thread::sleep_for(chrono::seconds(1)); // Pause for 1 second
    }
}

int main() {
    // Create a new thread and run `threadFunction`
    thread myThread(threadFunction);

    cout << "Main thread running..." << endl;

    // Wait for the thread to complete execution
    myThread.join();

    cout << "Thread execution completed." << endl;

    return 0;
}
```

Explanation

1. `std::thread`: Creates and starts a new thread executing `threadFunction`.
2. `this_thread::sleep_for()`: Pauses the thread for 1 second.
3. `myThread.join()`: Ensures the main thread waits for `myThread` to finish before continuing.

Output (Example)

```
Main thread running...
Thread running: 1
Thread running: 2
Thread running: 3
Thread running: 4
Thread running: 5
Thread execution completed.
```

Difference between Process and Thread

Process:
Process means ***any program is in execution***.

- **Process control block (PCB)** controls the operation of any process.
- **Process control block (PCB)** contains the information about processes for example: Process priority, process id, process state, CPU, register etc.
- A process can creates other processes which are known as **Child Processes**.
- Process takes more time to terminate and it is isolated means **it does not share memory with any other process**.

Process Concept

- An operating system executes a variety of programs that run as a process.
- **Process** – a program in execution; process execution must progress in **sequential fashion**
- Multiple parts
  - The program code, also called **text section**
  - Current activity including **program counter**, processor registers
  - **Stack** containing temporary data
    - Function parameters, return addresses, local variables
  - **Data section** containing global variables
  - **Heap** containing memory dynamically allocated during run time

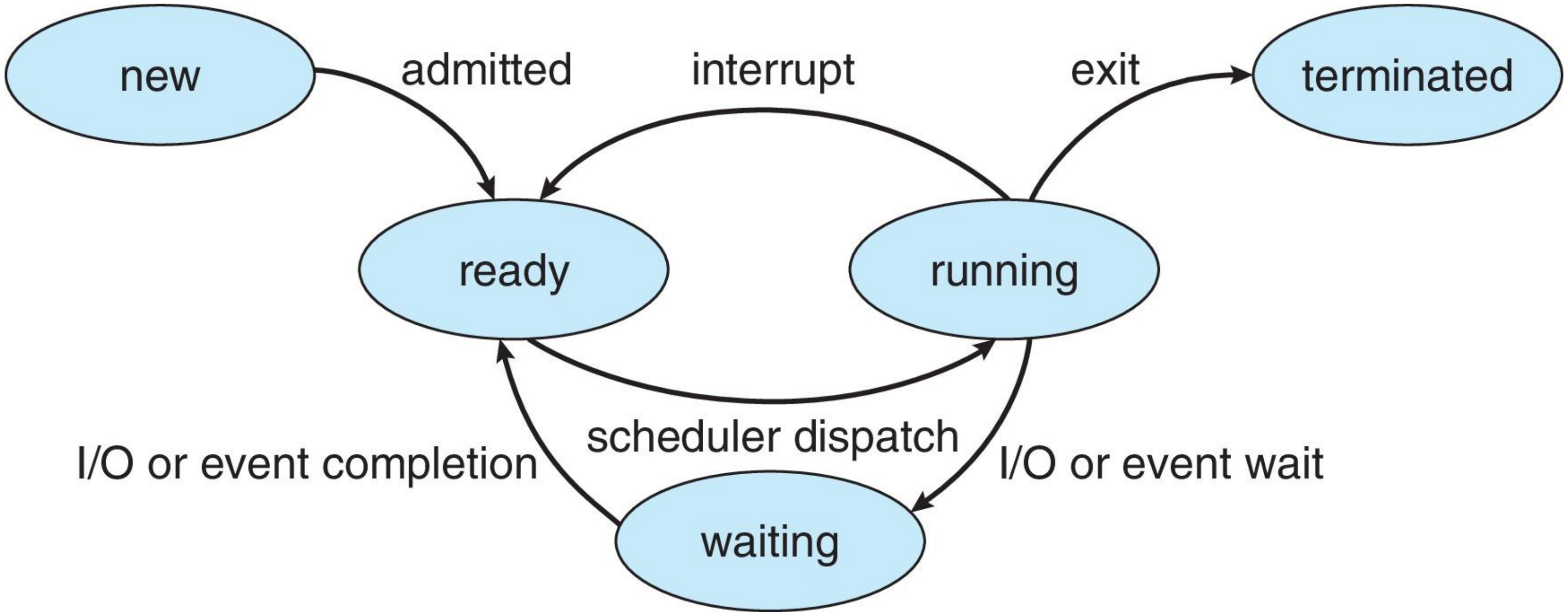
Process Concept (Cont.)

- Program is ***passive*** entity stored on disk (**executable file**); process is ***active***
  - **Program becomes process when executable file loaded into memory**
- **Execution of program started via GUI mouse clicks, command line entry of its name, etc**
- One program can be **several processes**
  - **Consider multiple users executing the same program**

Process State

- As a process executes, it changes **state**
  - **New**: The process is being created
  - **Running**: Instructions are being executed
  - **Waiting**: The process is waiting for some event to occur
  - **Ready**: The process is waiting to be assigned to a processor
  - **Terminated**: The process has finished execution

Diagram of Process State

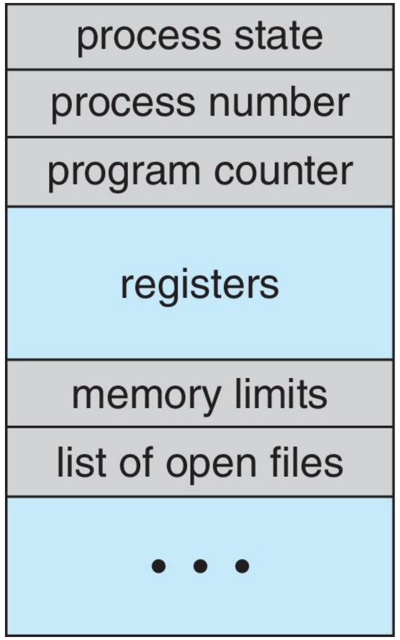


Process Control Block (PCB)

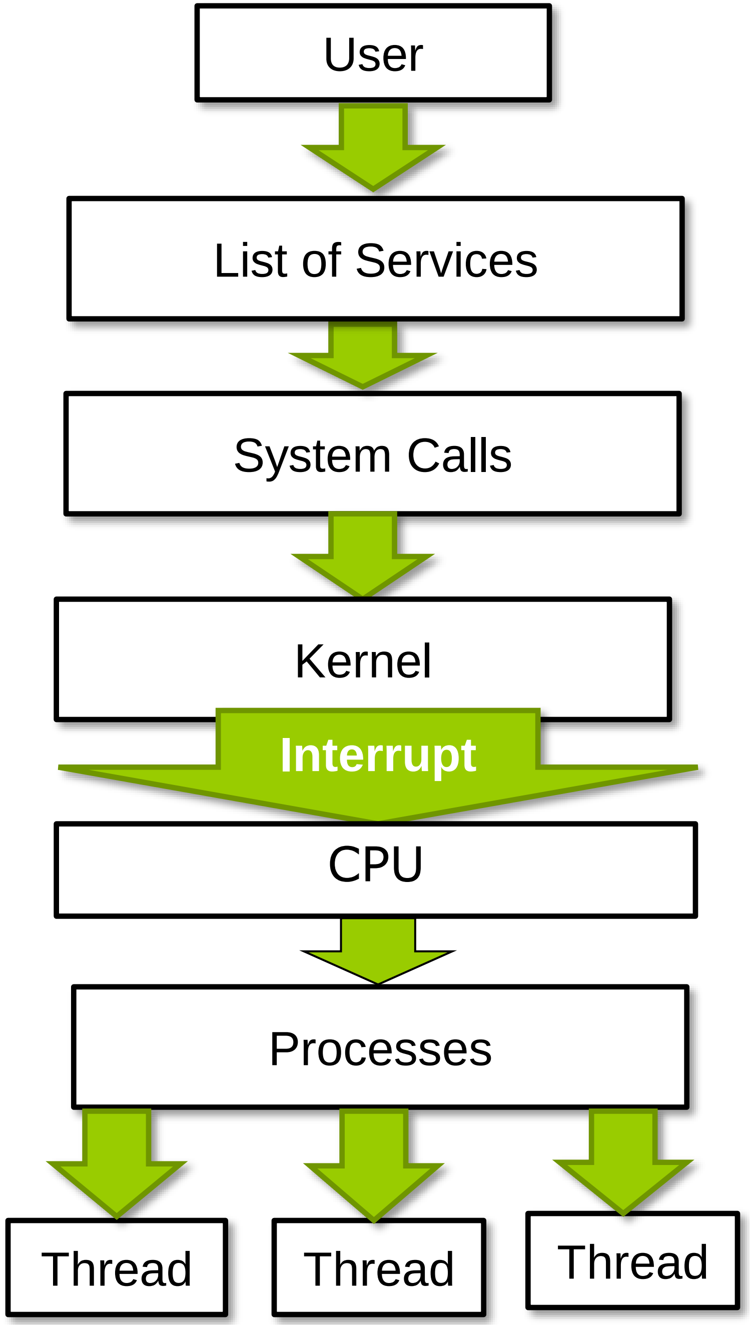
Information associated with each process

(also called **task control block**)

- Process state – running, waiting, etc.
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files



The Big Pic of OS


User

List of Services

System Calls

Kernel

Interrupt

CPU

Processes

Thread

Thread

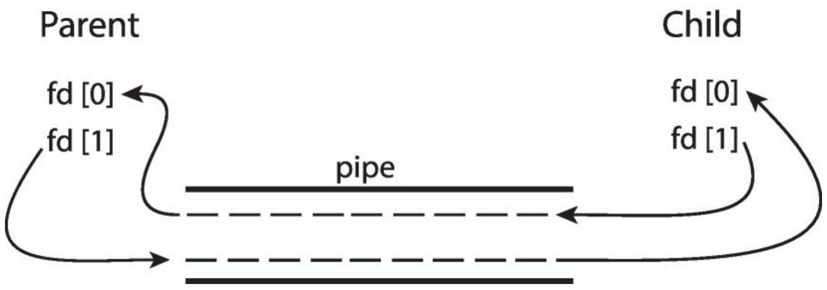
Thread

Pipes

- Acts as a conduit allowing two processes to communicate
- **Ordinary pipes** – cannot be accessed from outside the process that created it. Typically, a parent process creates a pipe and uses it to communicate with a child process that it created.
- **Named pipes** – can be accessed without a parent-child relationship.

Ordinary Pipes

- Ordinary Pipes allow communication in standard producer-consumer style
- Producer writes to one end (the **write-end** of the pipe)
- Consumer reads from the other end (the **read-end** of the pipe)
- Ordinary pipes are therefore unidirectional
- Require parent-child relationship between communicating processes



- Windows calls these **anonymous pipes**

Named Pipes

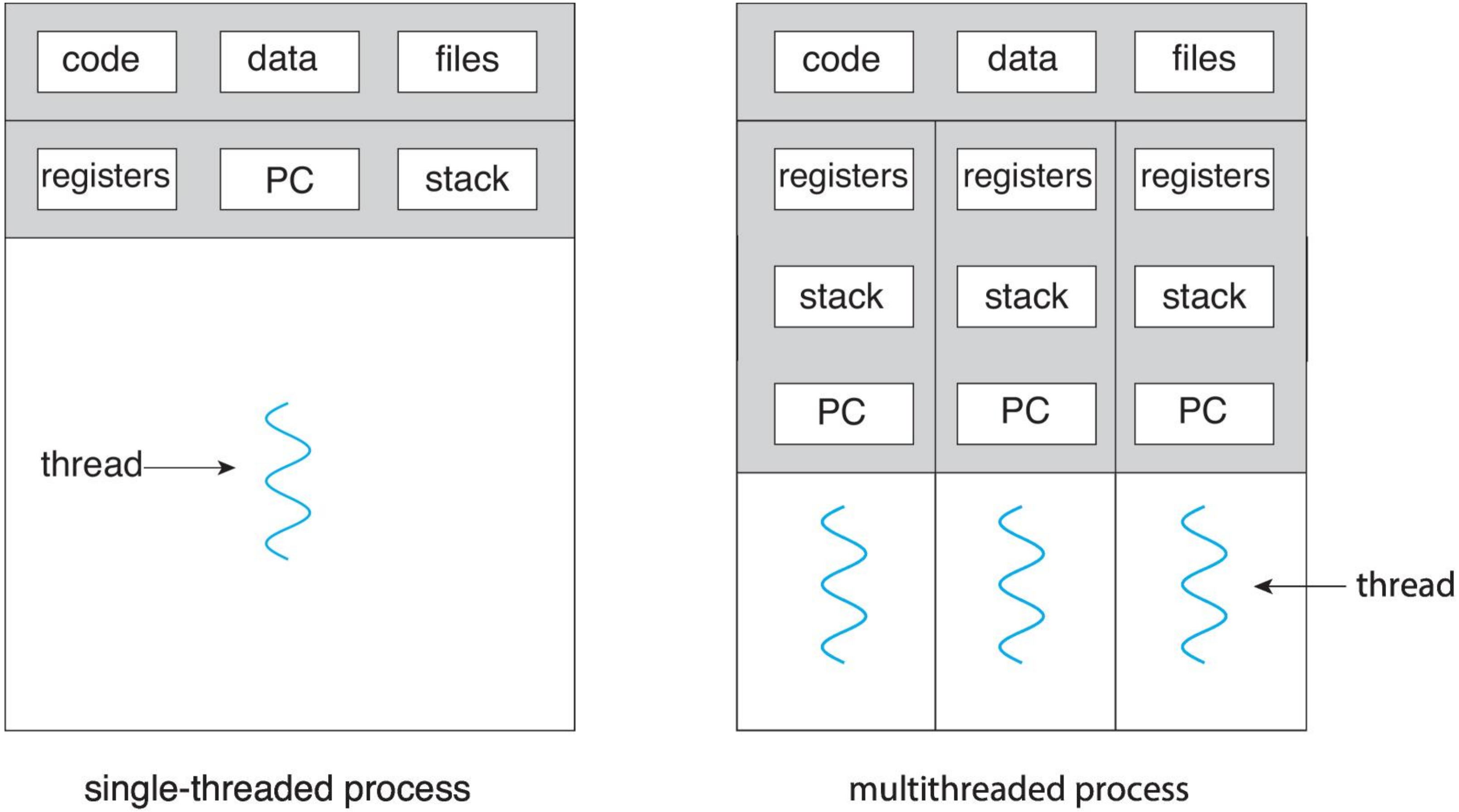
- Named Pipes are more powerful than ordinary pipes
- Communication is bidirectional
- No parent-child relationship is necessary between the communicating processes
- Several processes can use the named pipe for communication
- Provided on both UNIX and Windows systems

Multithreading

Motivation

- **Most modern applications** are multithreaded
- Threads run within **application**
- **Multiple tasks** with the application can be implemented by separate threads
  - Update display
  - Fetch data
  - Spell checking
  - Answer a network request
- Process creation is heavy-weight while thread creation is light-weight
- **Can simplify code, increase efficiency**
- **Kernels** are generally multithreaded

Single and Multithreaded Processes



Benefits

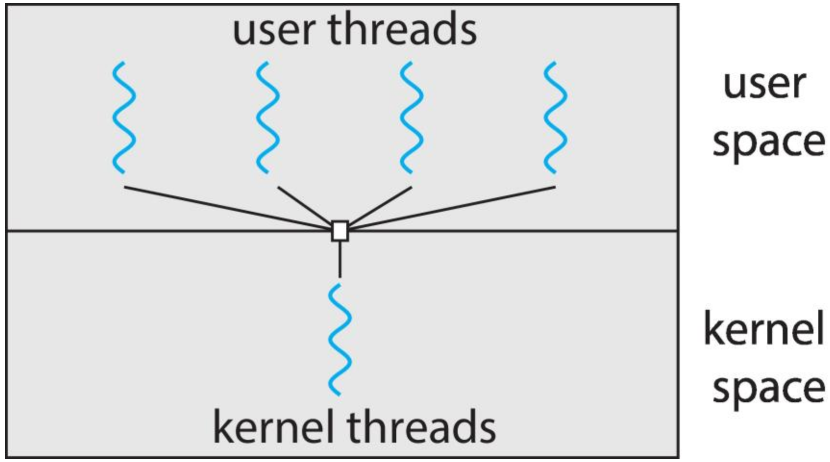
- **Responsiveness –** may allow continued execution if part of process is blocked, especially important for user interfaces
- **Resource Sharing –** threads share resources of process, easier than shared memory or message passing
- **Economy –** cheaper than process creation, thread switching lower overhead than context switching
- **Scalability –** process can take advantage of multicore architectures

Multithreading Models

- Many-to-One
- One-to-One
- Many-to-Many

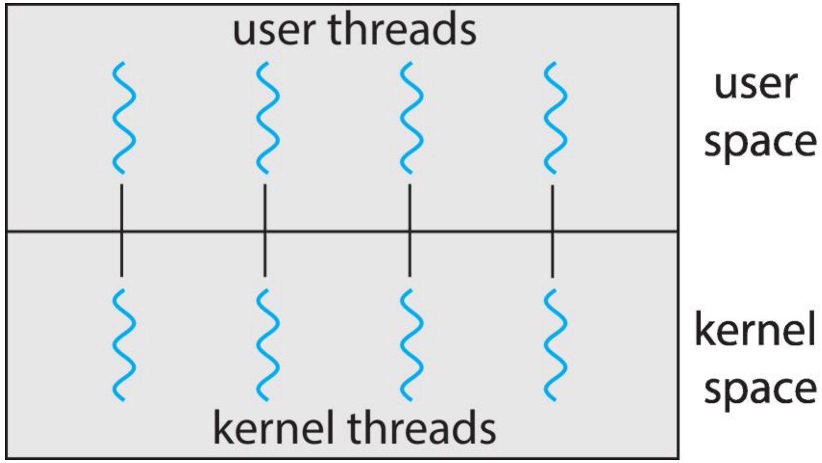
Many-to-One

- Many user-level threads mapped to single kernel thread
- One thread blocking causes all to block
- Multiple threads may not run in parallel on muticore system because only one may be in kernel at a time
- Few systems currently use this model
- Examples:
  - **Solaris Green Threads**
  - **GNU Portable Threads**



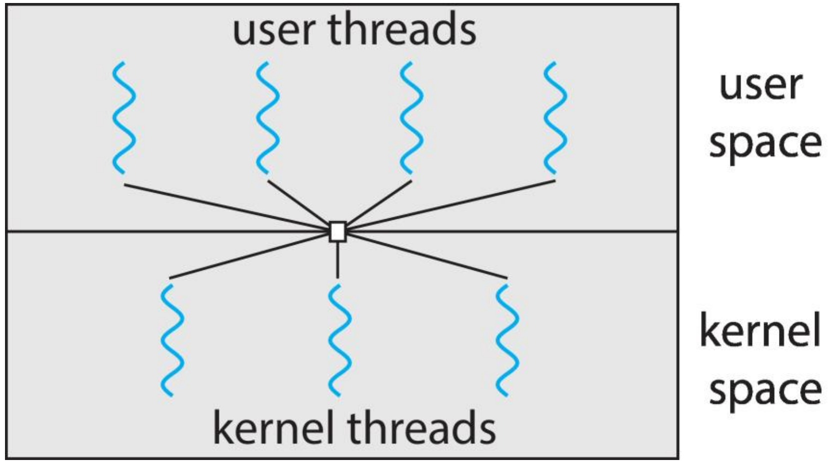
One-to-One

- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead
- Examples
  - Windows
  - Linux

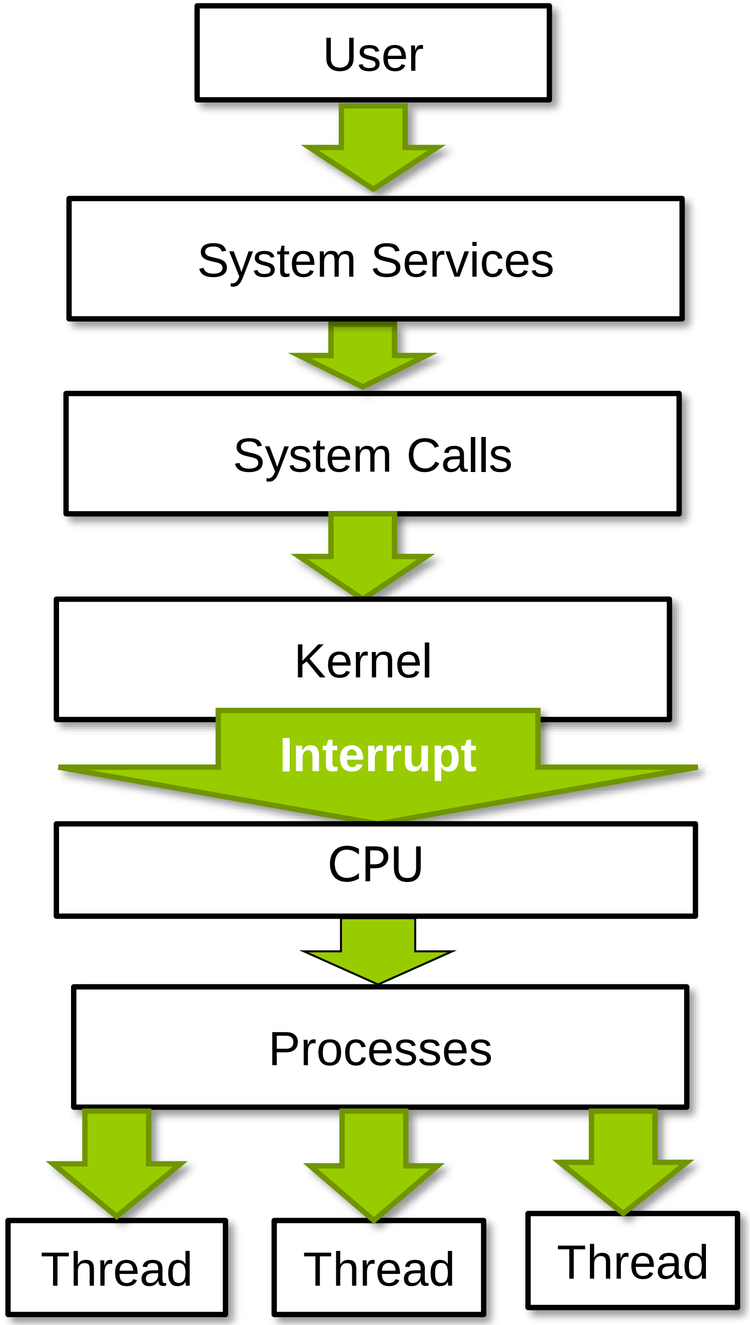


Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Windows with the *ThreadFiber* package
- Otherwise not very common

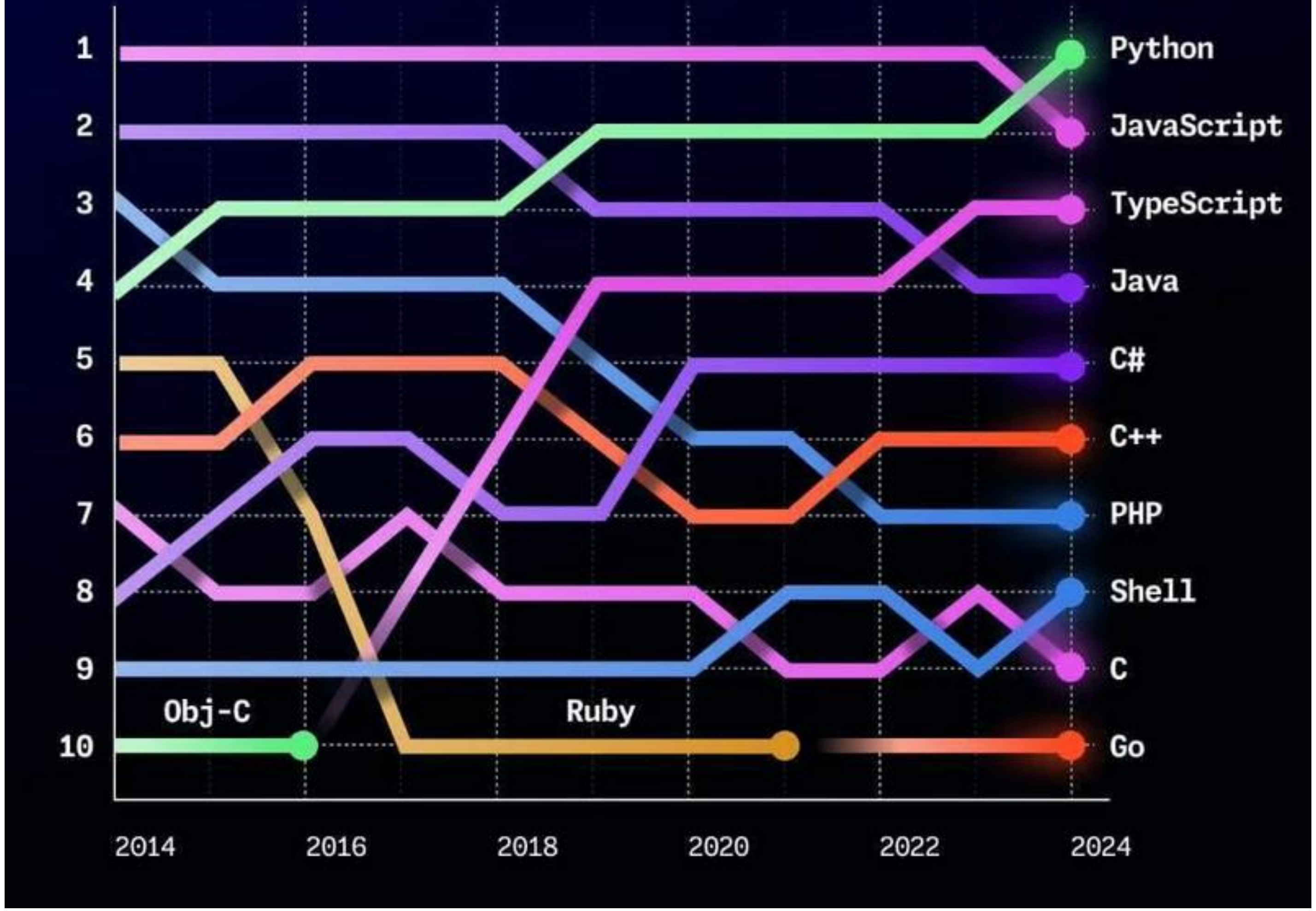


The Big Pic of OS

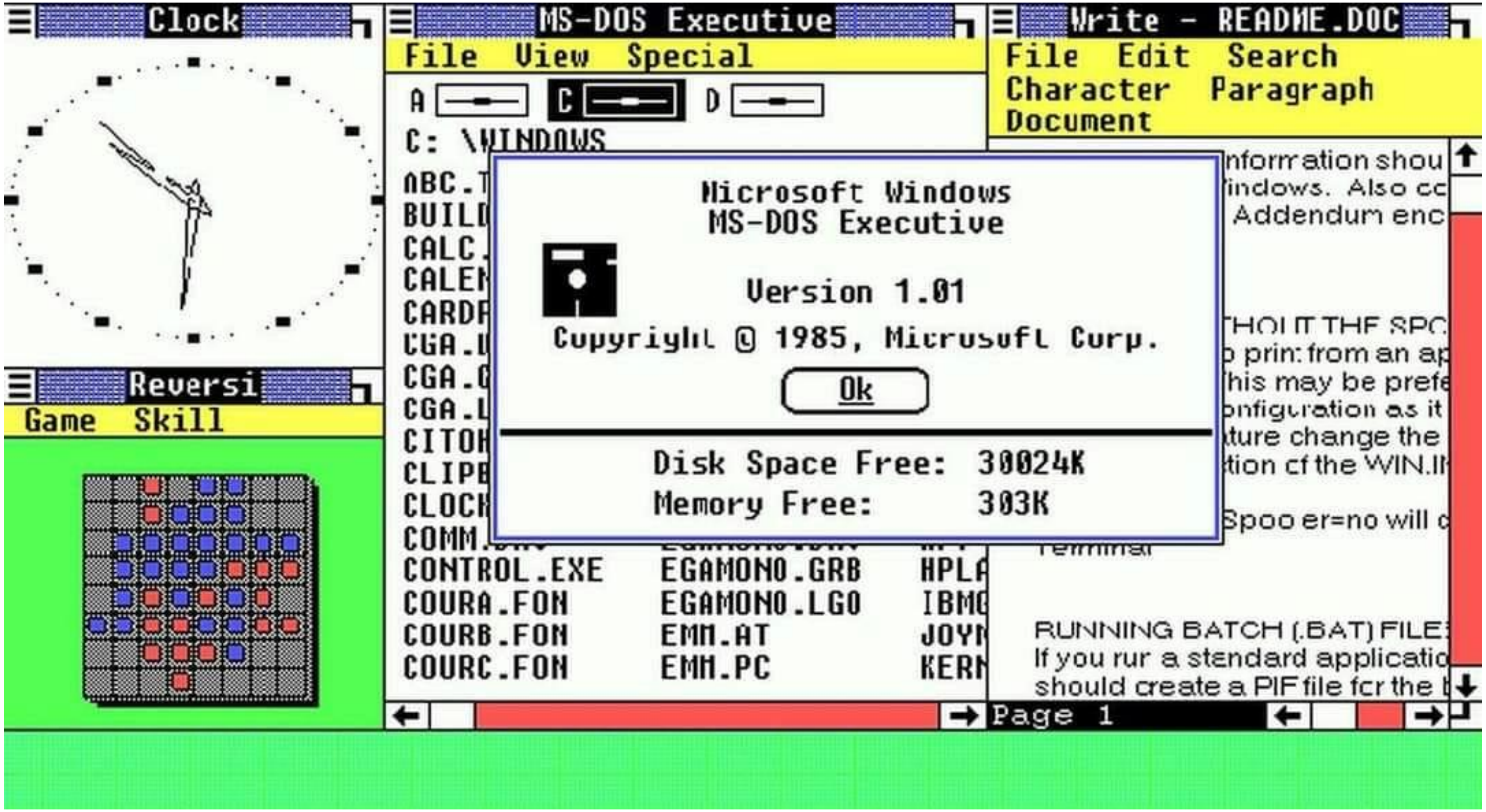


Top programming languages on GitHub

RANKED BY COUNT OF DISTINCT USERS CONTRIBUTING TO PROJECTS OF EACH LANGUAGE.



Windows 1.0 Released 20th November 1985



Microsoft Windows
MS-DOS Executive

Version 1.01

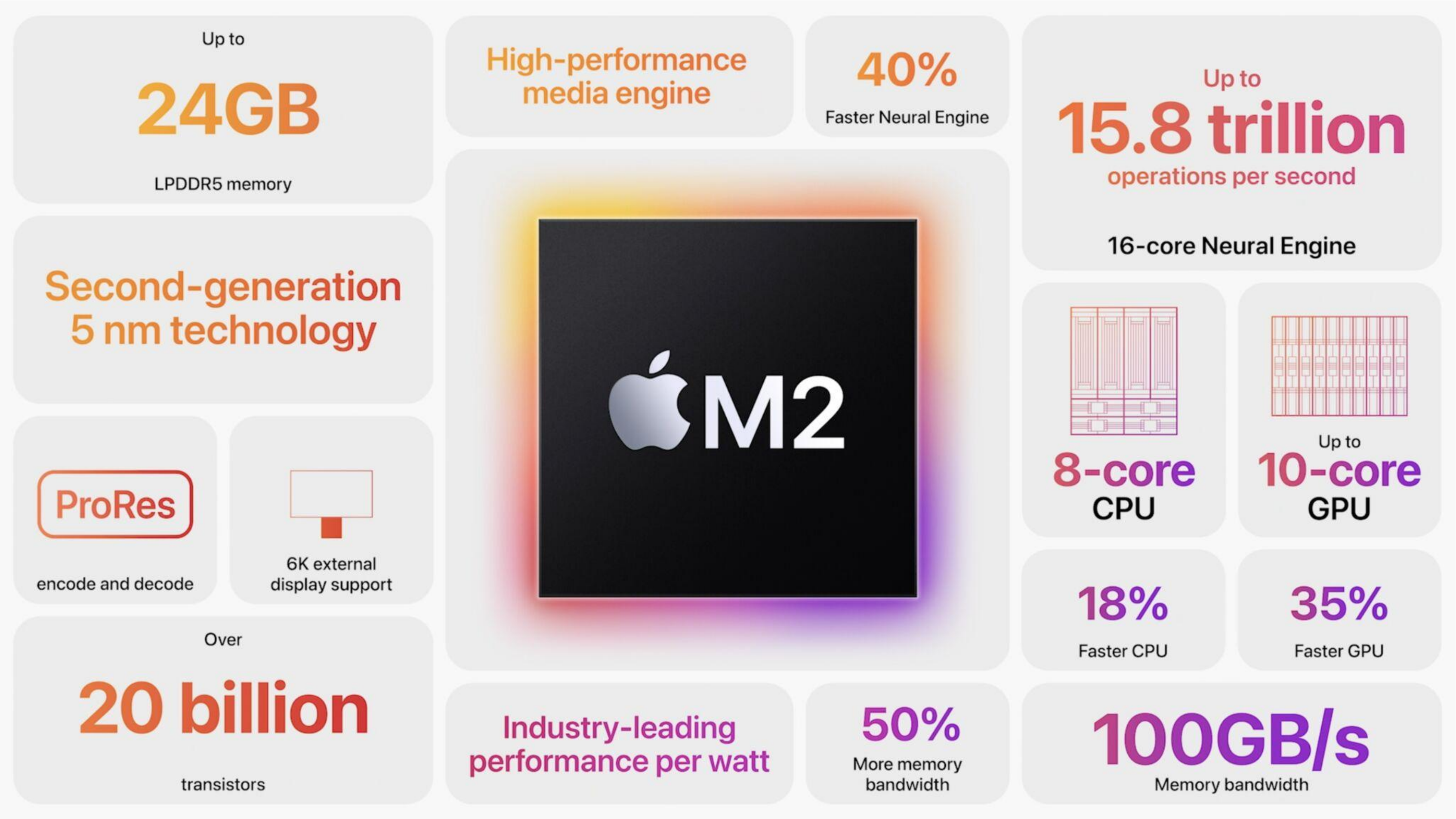
Cupyrighl © 1985, Micrusufl Curp.

Ok

Disk Space Free: 30024K
Memory Free: 303K

Microsoft® Windows
Operating Environment
MICROSOFT®
050-050-007
Setup Disk
For IBM® and COMPAQ®
Personal Computers
■ Disk 1 of 4
050050.101
MICROSOFT®
The High Performance Software™

The Central Processing Unit (CPU)



Up to

24GB

LPDDR5 memory

Second-generation 5 nm technology

ProRes

encode and decode

6K external display support

Over

20 billion

transistors

High-performance media engine

40%

Faster Neural Engine

M2

Industry-leading performance per watt

50%

More memory bandwidth

Up to

15.8 trillion

operations per second

16-core Neural Engine

8-core CPU

Up to

10-core GPU

18%

Faster CPU

35%

Faster GPU

100GB/s

Memory bandwidth

Thunderbolt 4

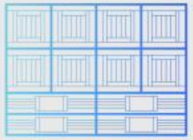
Up to **20%** faster CPU

Up to **30%** faster GPU

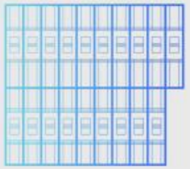
Industry-leading performance per watt

Up to **32GB** LPDDR5 memory

Over **40 billion** transistors



12-core CPU



Up to **19-core** GPU

16-core **Neural Engine** 15.8 trillion ops/s

40% Faster Neural Engine

200GB/s Memory bandwidth

High-performance media engine with ProRes

Second-generation **5 nm technology**

⚡

Thunderbolt 4

Up to

20%

faster CPU

Up to

30%

faster GPU

Industry-leading performance per watt

Up to

96GB

LPDDR5 memory

Over

67 billion

transistors

12-core

CPU

Up to

38-core

GPU

16-core

Neural Engine

15.8 trillion ops/s

40%

Faster Neural Engine

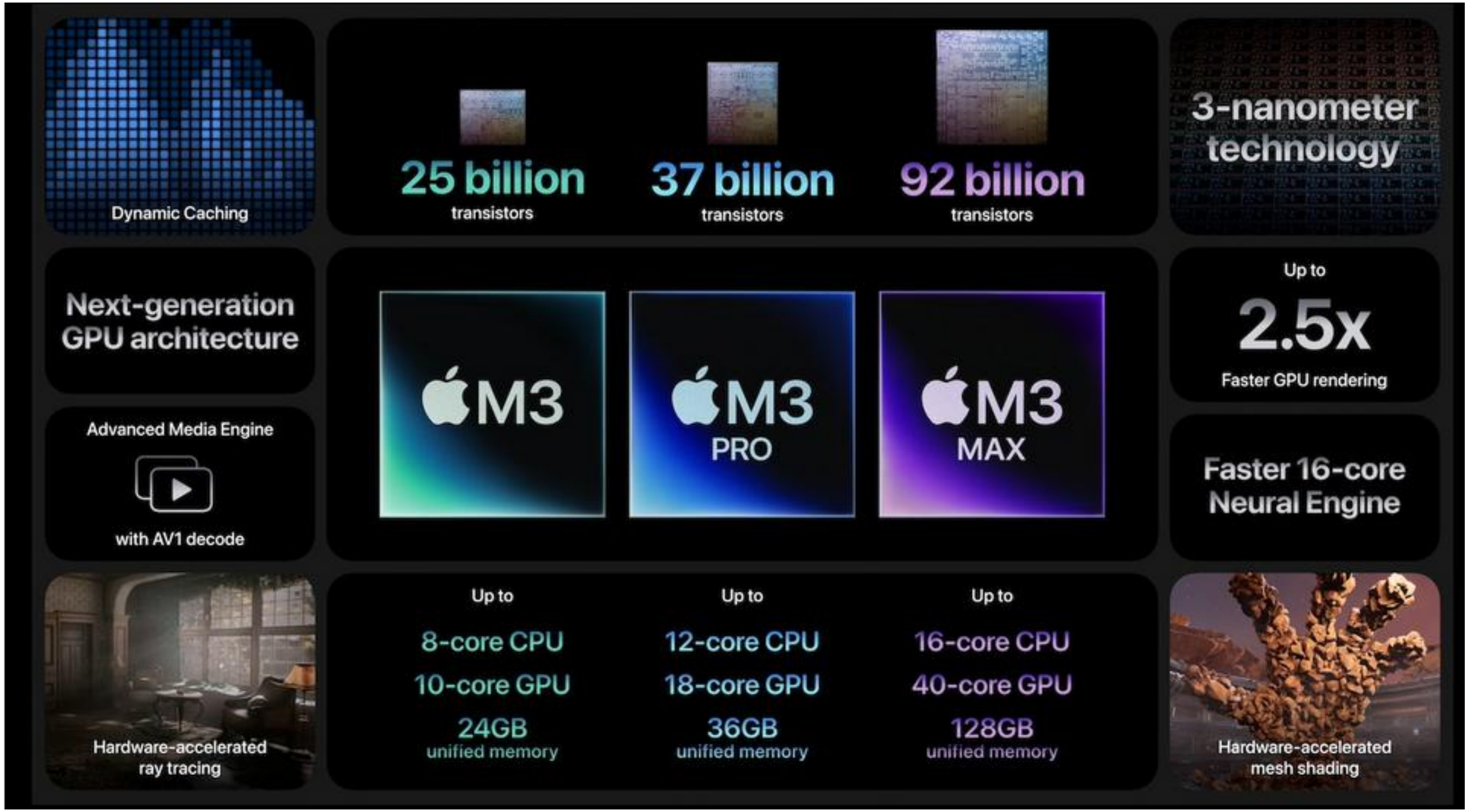
400GB/s

Memory bandwidth

High-performance media engine with ProRes

Second-generation

5 nm technology



Dynamic Caching

M3	M3 PRO	M3 MAX
25 billion transistors	37 billion transistors	92 billion transistors
Up to	Up to	Up to
8-core CPU	12-core CPU	16-core CPU
10-core GPU	18-core GPU	40-core GPU
24GB unified memory	36GB unified memory	128GB unified memory

3-nanometer technology

Next-generation GPU architecture

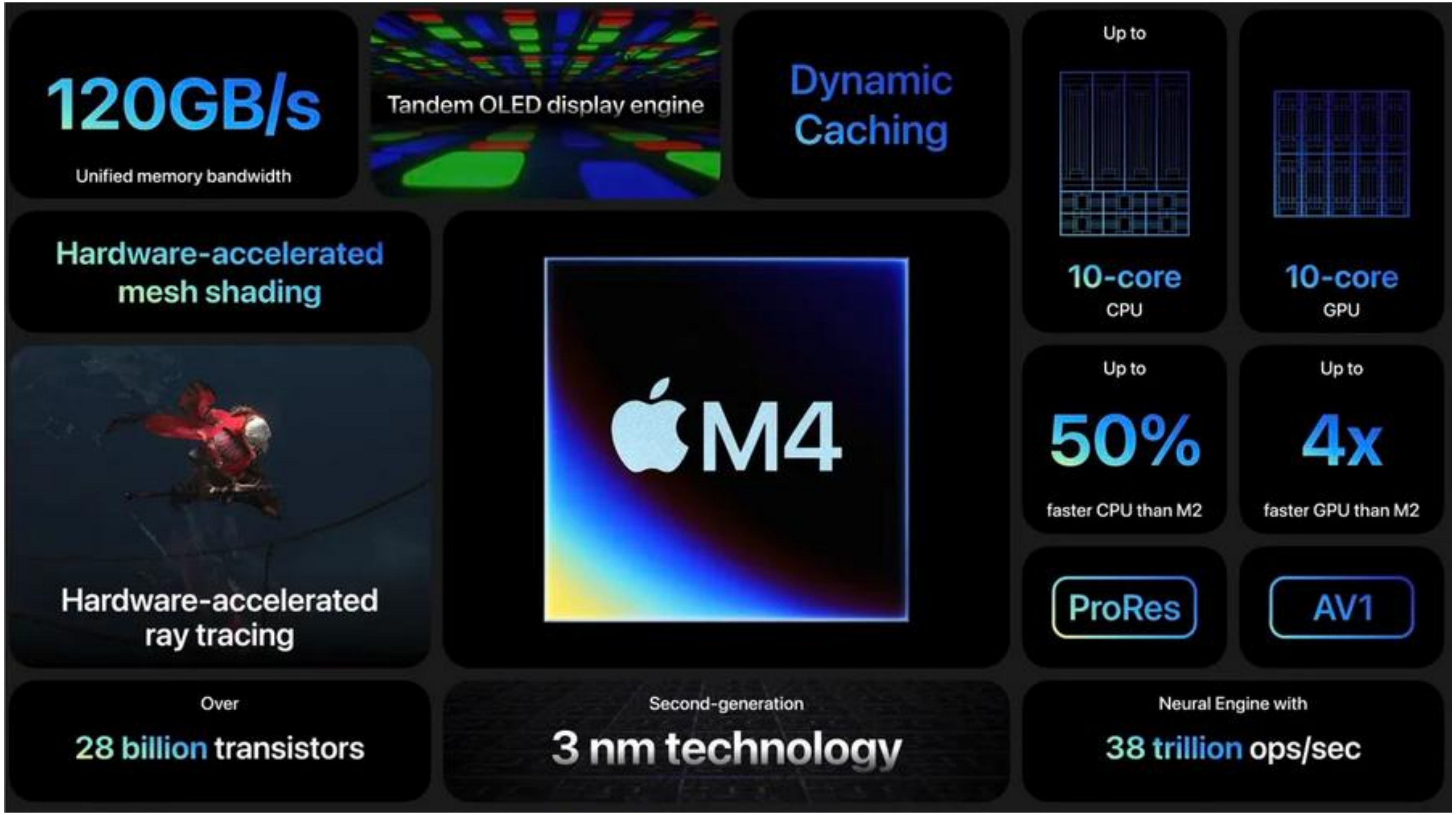
Advanced Media Engine with AV1 decode

Up to 2.5x Faster GPU rendering

Faster 16-core Neural Engine

Hardware-accelerated ray tracing

Hardware-accelerated mesh shading



120GB/s

Unified memory bandwidth

Tandem OLED display engine

Dynamic Caching

Up to

10-core

CPU

10-core

GPU

Hardware-accelerated mesh shading

M4

Hardware-accelerated ray tracing

Up to

50%

faster CPU than M2

Up to

4x

faster GPU than M2

ProRes

AV1

Over

28 billion transistors

Second-generation

3 nm technology

Neural Engine with

38 trillion ops/sec

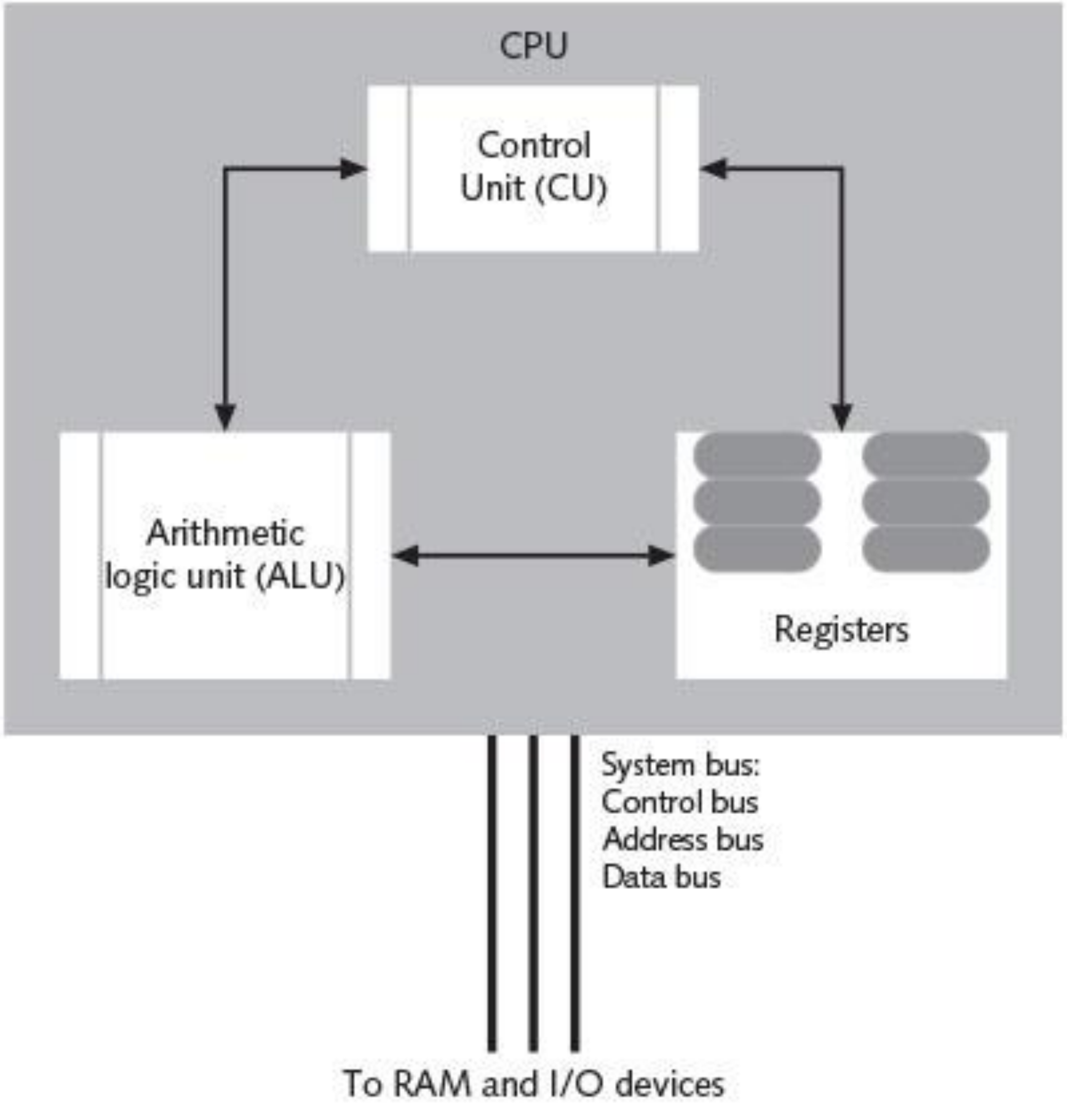
Understanding CPUs

- The **system architecture** of the computer is built **around** the CPU
  - Includes the number and type of CPUs in the hardware, and the communications routes (buses) between CPUs and other hardware components
- **CPU** – chip that performs the actual computational and logic work
- **Core** – section of the processor that actually does the **reading and execution of instructions**
  - Multicore processor has two or more cores
- **Multiprocessor computers** have multiple physical CPU chips

Basic CPU Architecture

- Most CPUs are composed of the following elements:
  - **Control unit** – provides timing and coordination between other parts of the CPU
  - **Arithmetic logic unit (ALU)** – performs the primary task of executing instructions
  - **Register** – temporary holding location where data must be placed before the CPU can use it
  - **System bus** – series of lanes used to communicate between the CPU and other major parts of the computer

Basic CPU Architecture



Basic CPU Architecture

- There are three types of buses:
  - **Control bus** – carries status signals between the CPU and other devices
  - **Address bus** – carries address signals to indicate where data should be read or written to in the system’s memory
  - **Data bus** – carries the actual data that is being read from or written to system memory

Basic CPU Architecture

CPUs can be **classified** by hardware elements:

- **Design type**
- **Speed**
- **Cache**
- **Address bus**
- **Data bus**
- **Control bus**
- **CPU scheduling**

Design Type

- Two general CPU designs are used today:
  - **Complex Instruction Set Computing (CISC)**
  - **Reduced Instruction Set Computing (RISC)**
- Main difference between the two design types is **the number of different instructions** the chip can process
- CPUs can process as many as **20 million** (low-end) to **several billion** (high-end) operations per second
- **Instruction set** – **list of commands** the CPU can understand and carry out

Design Type

- CISC and RISC CPUs differ in the following ways:
  - **Complex** versus **simple** instructions
    - CISC CPUs are generally more complex
  - **Clock cycles**
  - **Pipelining**
    - The ability of the CPU to perform more than one task on a single clock cycle
  - **Hardware versus microcode**
    - Small program inside the chip that interprets and executes each instruction

Design Type

- CISC and RISC CPUs differ in the following ways:
  - **Compiler**
    - A computer program that takes a high-level language and turns it into assembly code that is executed by the CPU
  - **Number and usage of registers**
    - CISC CPUs have **fewer registers** than a RISC CPU

Speed

- The speed of a CPU **defines how fast it can perform operations**
- Most obvious indicator is the **internal clock speed**
  - Clock provides a rigid schedule to make sure all the chips know what to expect at what time
  - The faster the clock, the faster the CPU
- As more components are needed to make a CPU, the chip uses more energy, which is converted to heat.
  - CPUs require fans to keep cool

Speed

- CPU must be able to communicate with other chips in the computer
    - Uses an **external clock speed** to communicate with the rest of the computer
    - External clock speed runs **slower** than the internal clock speed
        - Typically **one-half, one-third, one-fourth, or one-eighth** the speed of the internal CPU clock

Cache

- Since internal clock *is **faster** than* the external clock
  - The CPU would have to wait on information to arrive from other parts of the computer
- Most modern CPUs have cache memory built into the chip
- While CPU is executing program code
  - Instructions or data that are most likely to be used next are fetched from main memory and placed in cache memory

Cache

- There are different levels of cache
  - **Level 1 (L1) cache** is the fastest and usually runs at the same speed as the CPU
  - **Level 2 (L2) cache** is slower but much larger
  - **Level 3 (L3) cache,** until the last several years, was not part of the CPU chip, but part of motherboard
  - **Level 4 (L4) cache** will usually be found on motherboard (if it exists)
- **Cache controller** – predicts what data will be needed and makes the data available in cache before it is needed

Address Bus

- **Address Bus** – internal communications pathway that specifies the source and target addresses for memory reads and writes
  - Typically **runs at the external clock spe**ed of CPU
  - Width of the address is the number of bits that can be used to address memory
    - Wider bus means the computer can address more memory and store more data
  - Modern processors use a **64-bit** address bus
    - Allows them to address **16 terabytes (TB) of memory**

Data Bus

- The data bus allows computer components, such as CPU, display adapter, and main memory, to **share information**
- The number of bits in the data bus indicates how many bits of data can be transferred **from memory to the CPU in one clock tick**
  - **A CPU with an external clock speed of 1 GHz and a 64-bit data bus could transfer as much 8 GB per second**
- A CPU with a 64-bit data bus typically can perform operations on 64 bits of data at a time

Control Bus

- Information is transported on the **control bus** to keep the CPU **informed about the status of resources and devices connected to the computer**
- Memory read and write status is transported on this bus, as well as interrupt requests
  - **Interrupt request (IRQ)** – a request to the processor to “interrupt” whatever it is doing to take care of a process, which in turn might be interrupted by another process

CPU Scheduling

- CPU Scheduling – **determines which process to start given the multiple processes waiting to run**
- Beginning with Windows NT, use of CPU scheduling began to evolve to allow multithreading
  - **Multithreading** is the ability to run two or more processes (known as threads) at the same time
  - A **thread** is the smallest piece of computer code that can be independently scheduled for execution
  - **Hyper-Threading** allows two threads to run on each CPU core simultaneously

Summary

- One of the **main functions of the operating system is to provide the interface between the various application programs running on a computer and the hardware inside**
- Most CPUs are composed of **a control unit, arithmetic logic unit, registers, and a system bus, which is composed of a control bus, address bus, and data bus**

Summary

- CPUs can be **classified by** several elements, including **design type, speed, cache, address bus, data bus, control bus, and CPU scheduling**
- The **amount of cache is cri**tical to CPU’s overall **speed** because it is much faster than RAM

Summary

- CPU scheduling allows an operating system to schedule multiple processes or threads
- Intel processors are the most popular CPUs today, but AMD processors are frequently used
- Other processors include the Motorola, PowerPC, the SPARC, and the Alpha