Concurrent Programming

This presentation introduces concurrent programming. We will explore its concepts and applications. We will also examine the history and key differences. Let's delve into the world of concurrency.

Defining Concurrency

Parallel Execution

Tasks run simultaneously on multi-core systems.

Interleaved Execution

Tasks appear to run at the same time.

Why Study Concurrency?

1 **Real-World Systems**

It models complex systems effectively.

2 **Various Architectures**

It is useful across computer architectures.

3 **Application Necessity**

It's vital for certain applications.

Concurrent Application Examples

Web servers handle multiple client requests.

Artificial Intelligence Agents

Data Science processing

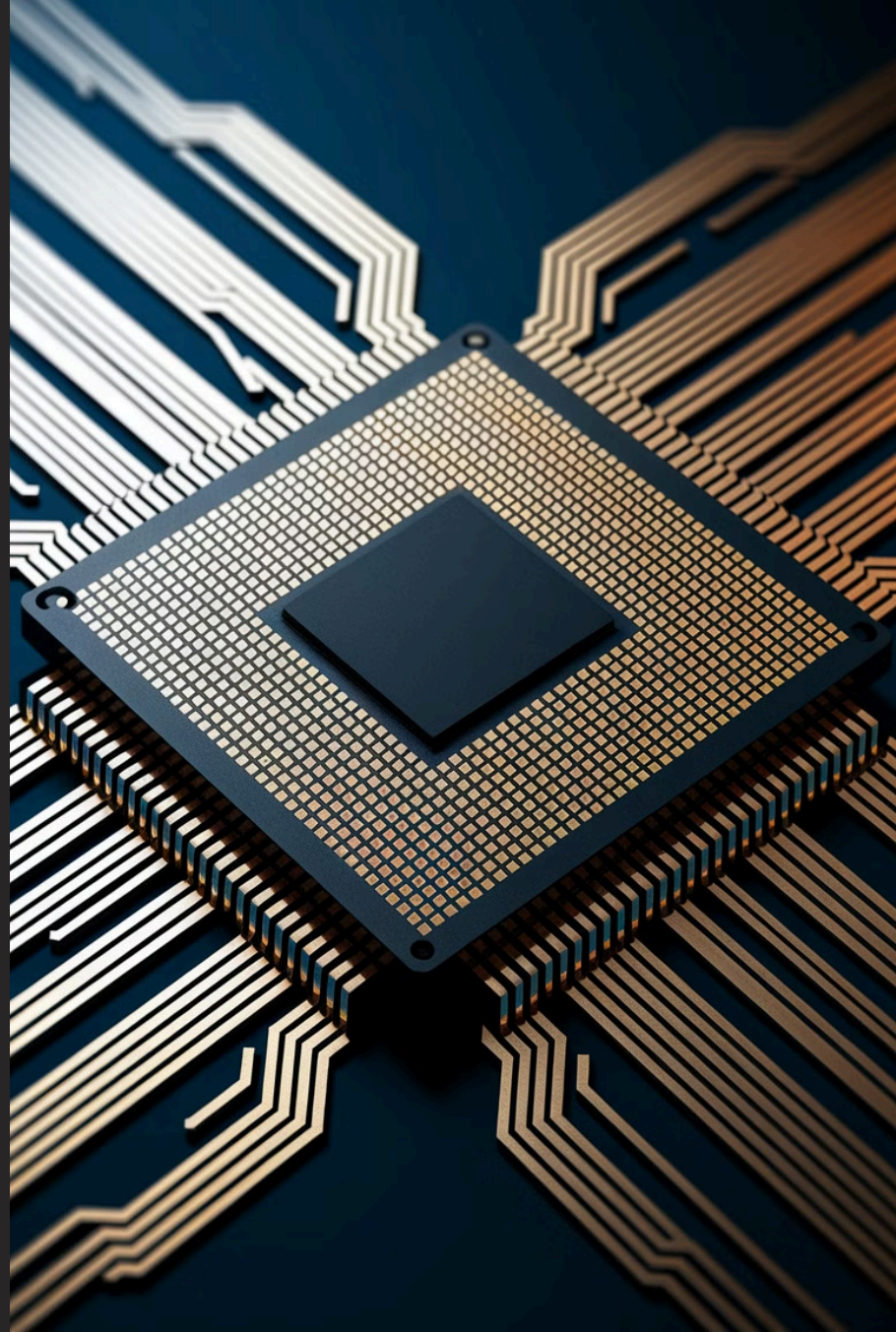
Simulations forecast weather or simulate flights.

Games manage actions from multiple players.

Operating systems running multiple applications.

Introduction to Moore's Law

Moore's Law predicts processing power growth. It has driven innovation for decades. Let's look at its origin and impact.



Moore's Law: Definition

Transistors

The number of transistors doubles every two years.

Cost

The cost of computers is halved.

Origin

Coined by Gordon Moore in 1965.

Moore's Law: Impact

1. **Exponential Growth**

   Computing power and efficiency increase.

2. **Innovation Acceleration**

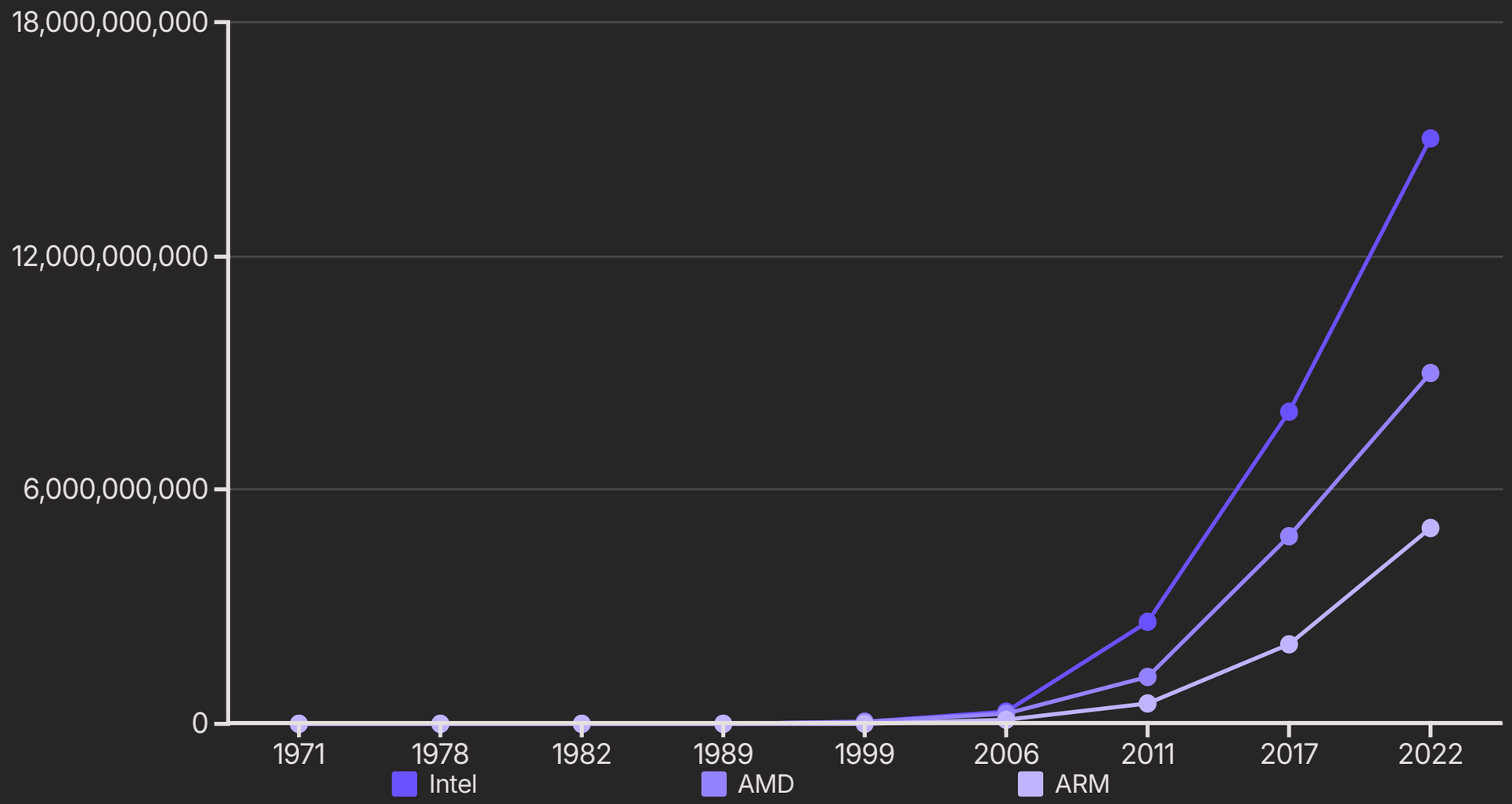
   More powerful and compact devices emerge.

3. **Economic Effects**

   The tech industry emphasizes rapid development.

Transistors Over Time

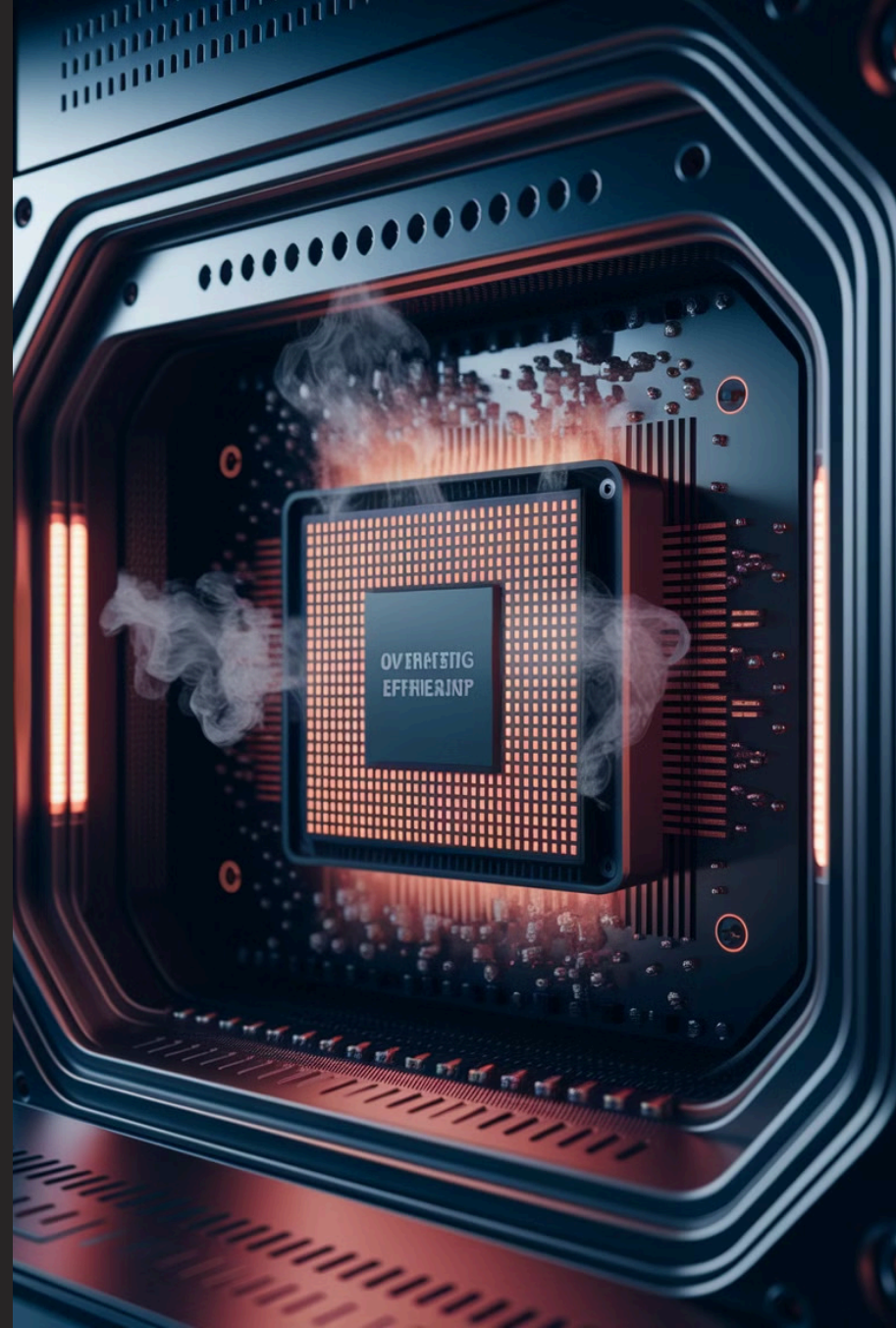
Transistor count has grown exponentially across processor generations, as predicted by Moore's Law.



Key milestones include: Intel 4004 (1971, 2,300 transistors), Intel 8086 (1978, 29,000), Intel 80386 (1982, 134,000), Intel Pentium (1993, 3.1 million), and modern processors with billions of transistors. This exponential growth has powered decades of computing advancement.

Limits in Processor Speed

Speed improvements began to slow in the mid-2000s. The number of transistors continued to grow.



Heat and Power Wall

Heat Generation

Transistor density increases heat generation.

Power Consumption

Higher clock speeds increase power use and heat.

Diminishing Returns

Beyond certain point Clock speed gains do not translate to performance gains.

Innovations Beyond Clock Speed

1

Architectural Improvements

Advances in CPU architecture, larger caches.

2

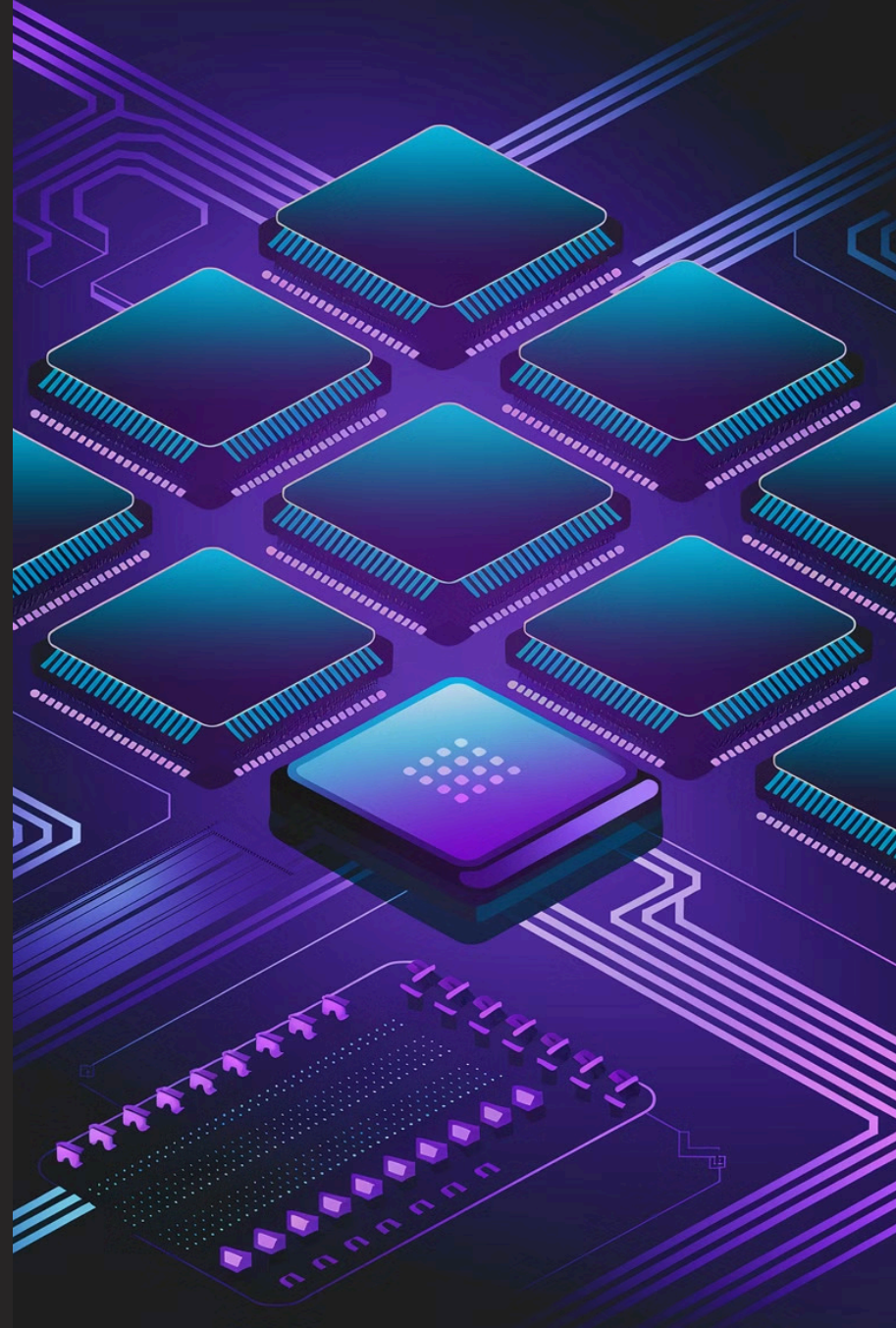
Multi-core Processors

Adding more processing cores for parallel execution.

3

Specialized Units

Using specialized computing units, like GPUs.



Parallelism and Hardware

- Multicore Processors
- Multiple Processors
- GPUs
- Grid Computing on Networks

History of Concurrency

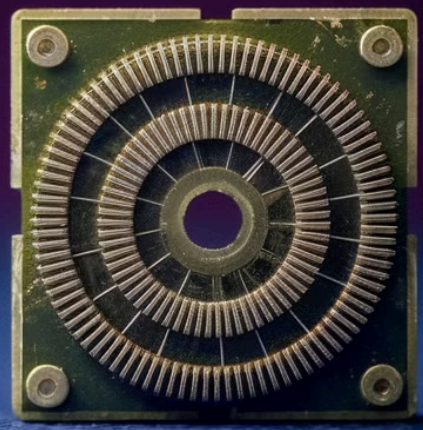
Let's look at the early days of computing. We will discuss multiprogramming and time-sharing. This will give us context.

Early Days of Computing

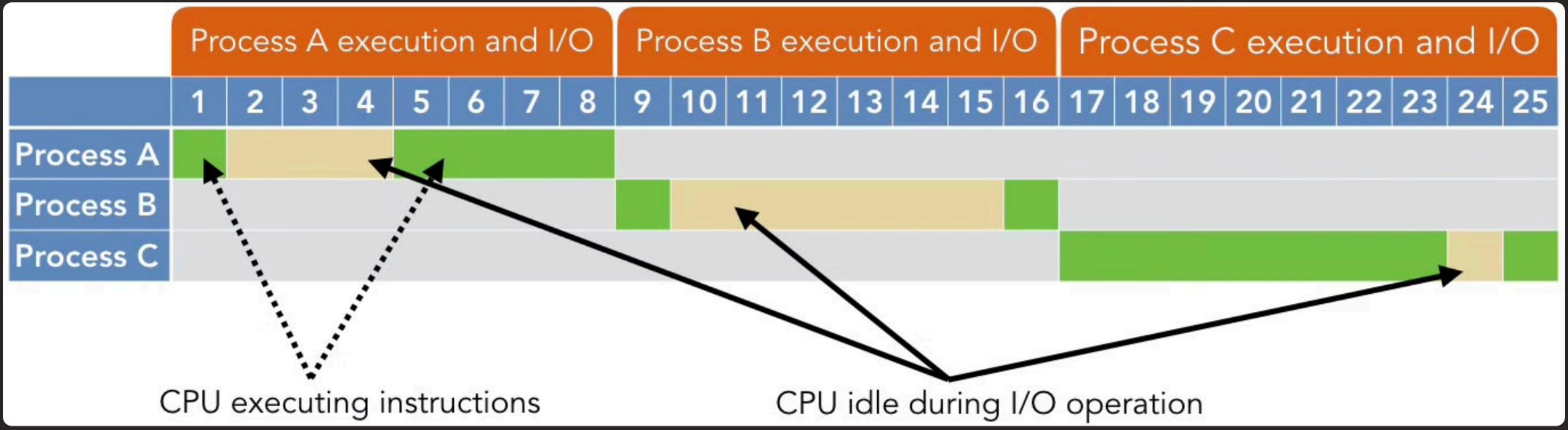
1. Computers ran a single program.
2. Direct access to all machine resources.
3. Inefficient resource utilization.

Introduction to Multi-programming

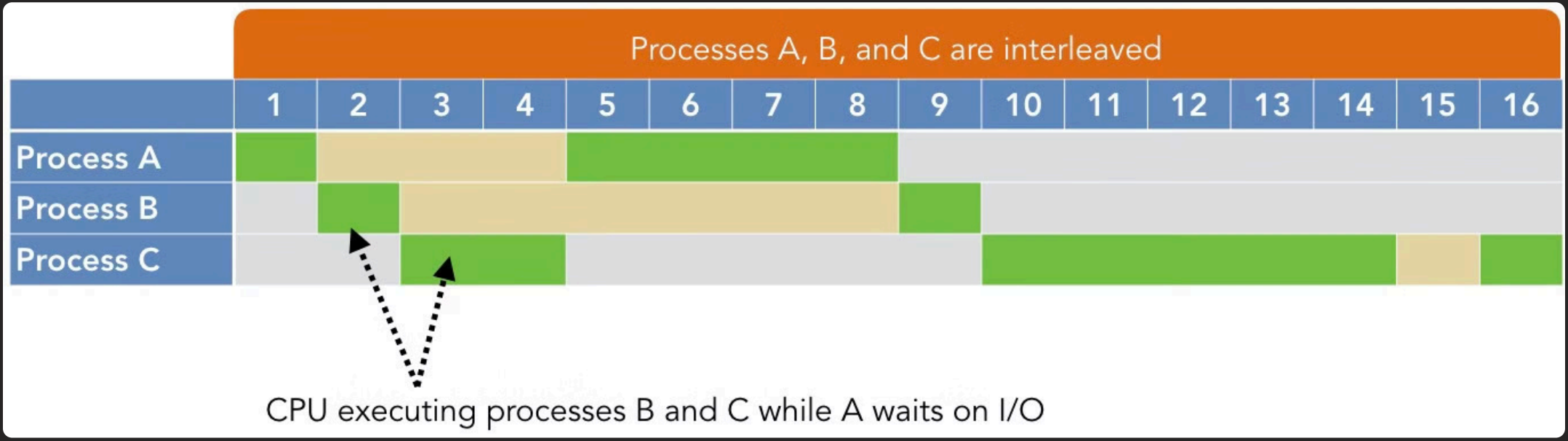
1. Shift from single to multiple program execution.
2. Loading several jobs into memory to be executed concurrently
3. Enhancing CPU utilization. by organizing jobs so CPU is always busy



Uni-programming: sequential jobs



Multiprogramming: parallel jobs



Operating Systems and Time-Sharing

Time-Sharing

Time-sharing is a logical extension of multiprogramming.

Processor Sharing

Processor's time is shared among multiple users.

User Experience

Creates the illusion of a dedicated machine for each user.

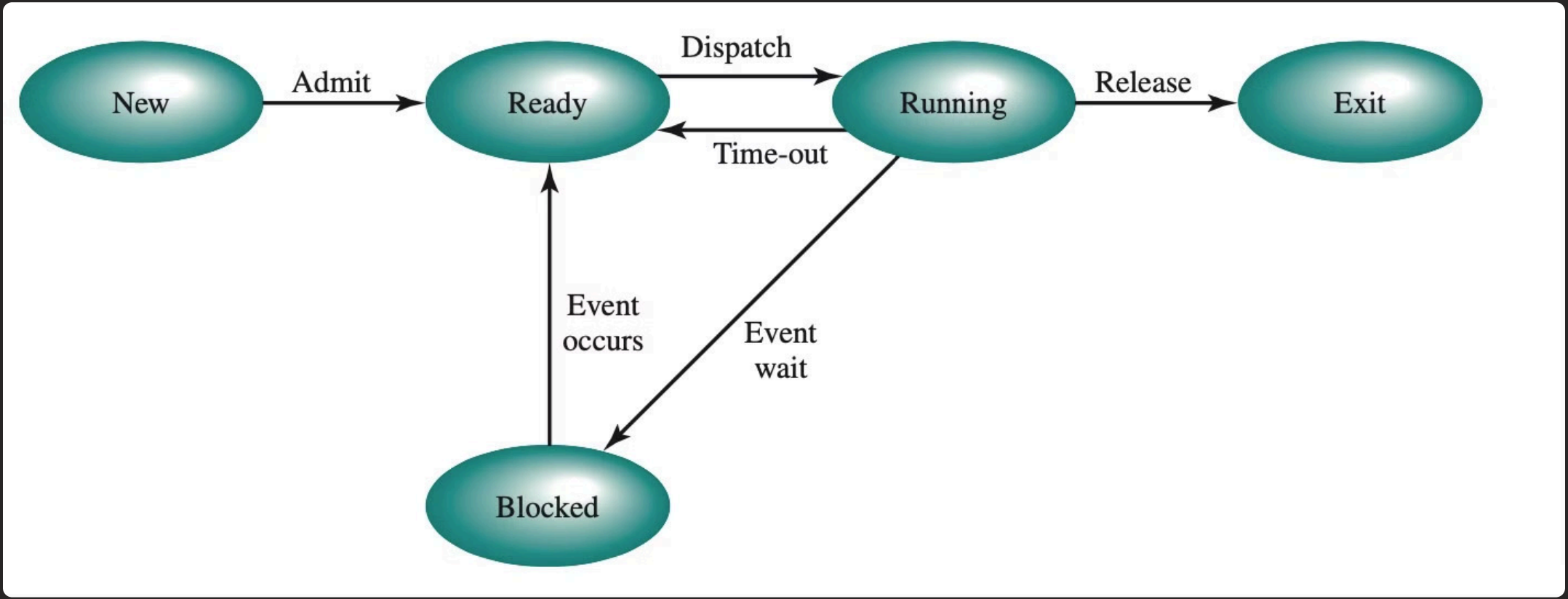
Why Concurrent Execution?

Efficient resource utilization.

Fairness: Equal resource sharing.

Convenience: Easier task management.

Process States



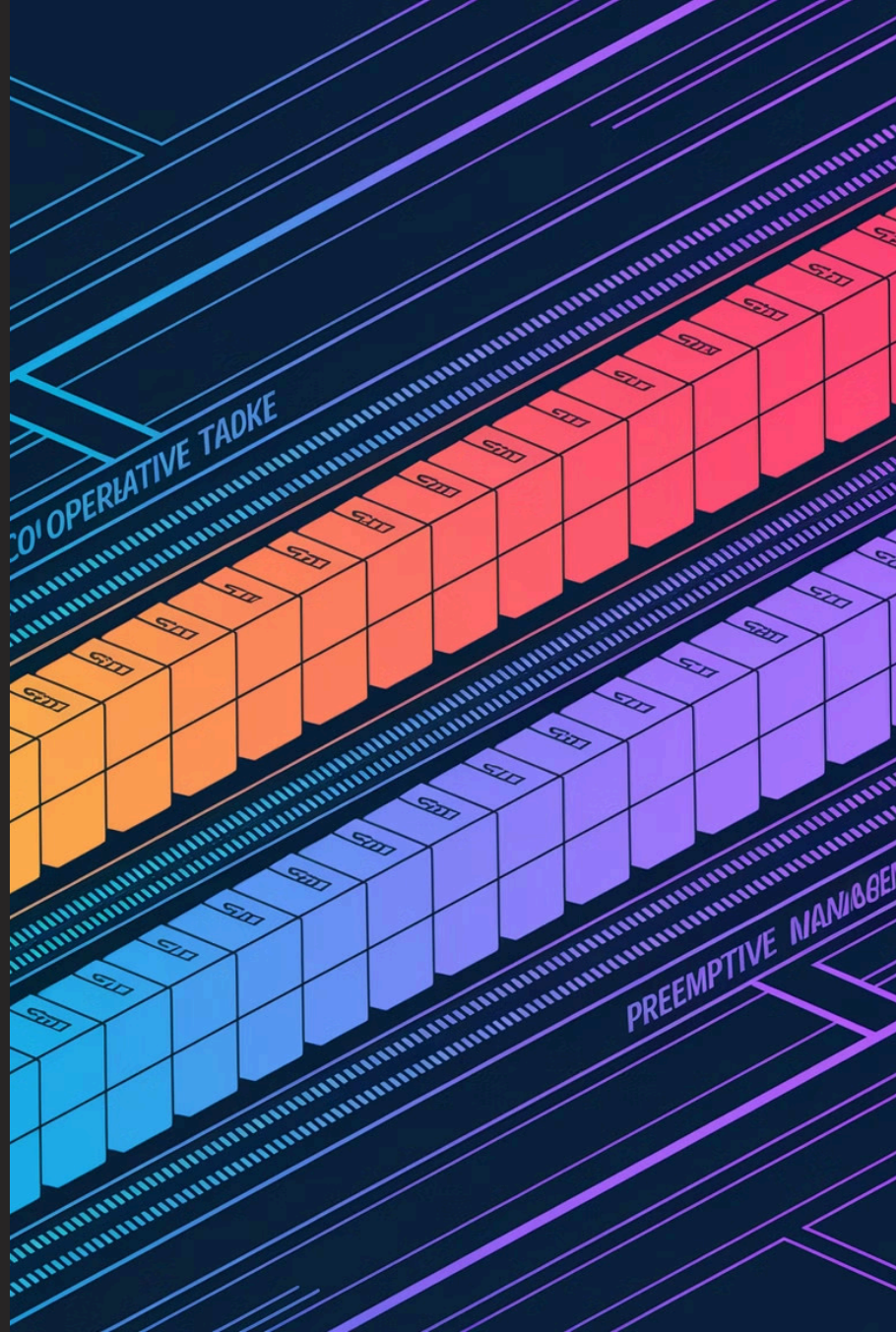
Scheduling

Cooperative

Tasks voluntarily yield control of the CPU.

Preemptive

OS controls task execution, forcing interruptions and resuming them as needed



Cooperative Scheduling

1 **Task Control**

Tasks control relinquishing the CPU.

2 **Yielding**

Tasks yield when idle or decide to allow others.

3 **Advantages**

Simplicity, **low overhead**, predictable.

4 **Challenges**

Vulnerable to misbehaving processes, potential for CPU monopolization.

Preemptive Scheduling

1. **Controlled by OS**

   OS determines when a task relinquishes CPU.

2. **Time Slicing**

   Tasks have CPU time slices.

3. **Advantages**

   Improved responsiveness and fairness.

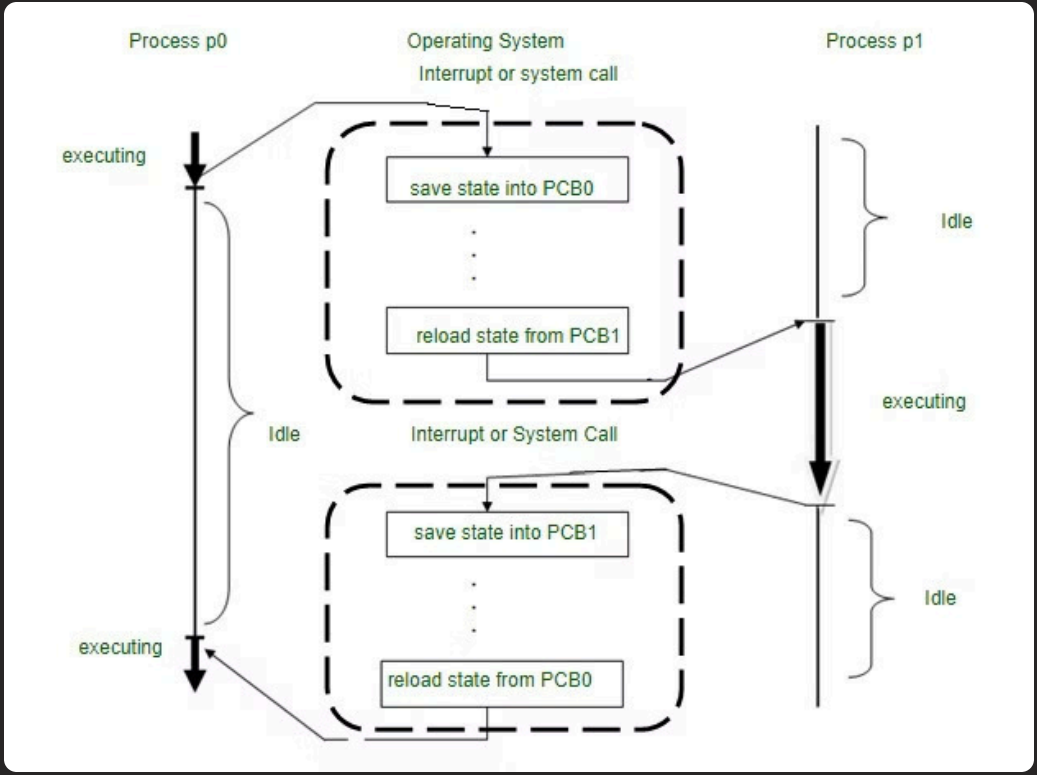
4. **Challenges**

   Potential for resource contention, overhead from frequent context switching, and complexity in implementation.

Context Switch

A context switch stores and restores the CPU state. It pauses and resumes processes.

Context Switch Process



A context switch involves saving and restoring the execution state when switching between processes.

Program Counter

Address of the next instruction.

CPU Registers

Values in processor registers.

Stack

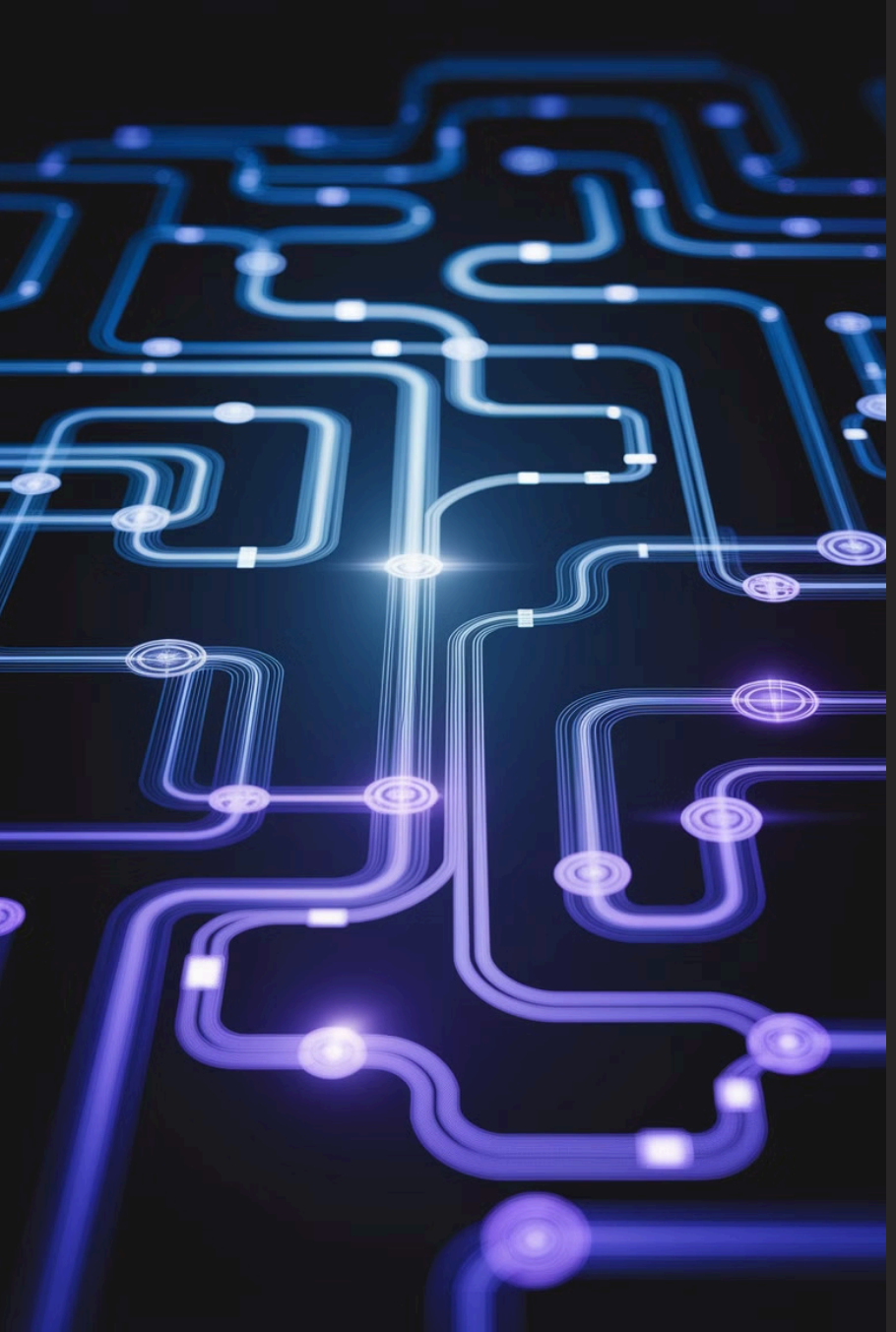
Local variables, parameters.

Memory

Management Information.

Concurrency vs Parallelism

Concurrency manages multiple tasks. Parallelism executes multiple tasks simultaneously. Interleaving alternates between tasks.



Concurrency Defined

1. The ability of a system to Manage multiple tasks at the same time.
2. Operations are not necessarily executing at the same time.
3. Different tasks can be in progress at the same time. No necessarily at the some moment.

Concurrency is about structure

Parallelism Defined

1. Executing multiple tasks **simultaneously**.
2. Requires hardware with **multiple** processing units.
3. Tasks are literally executed at the same time.

Parallelism is about execution

Interleaving

A technique to alternate between tasks rapidly. It executes them in small slices.

Interleaving Characteristics

Simulates Parallelism — 1

2 — **Time-sharing**

CPU time is divided among multiple processes.

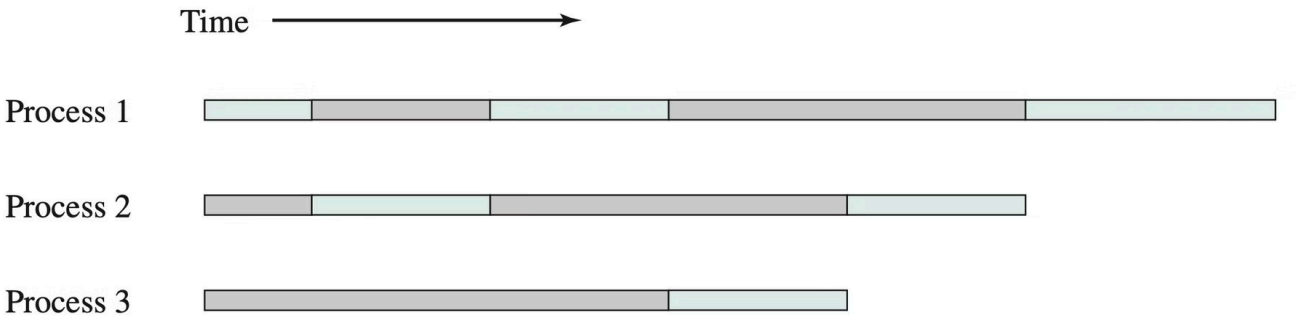
Efficiency — 3

Provides efficient CPU utilization.

A chef preparing multiple dishes by switching between them.

Rather than cooking each one start to finish.

Interleaving and Overlapping



(a) Interleaving (multiprogramming; one processor)

(b) Interleaving and overlapping (multiprocessing; two processors)

Processes vs Threads

Program

A collection of instructions and data that can be executed by a computer to perform a specific task or function.

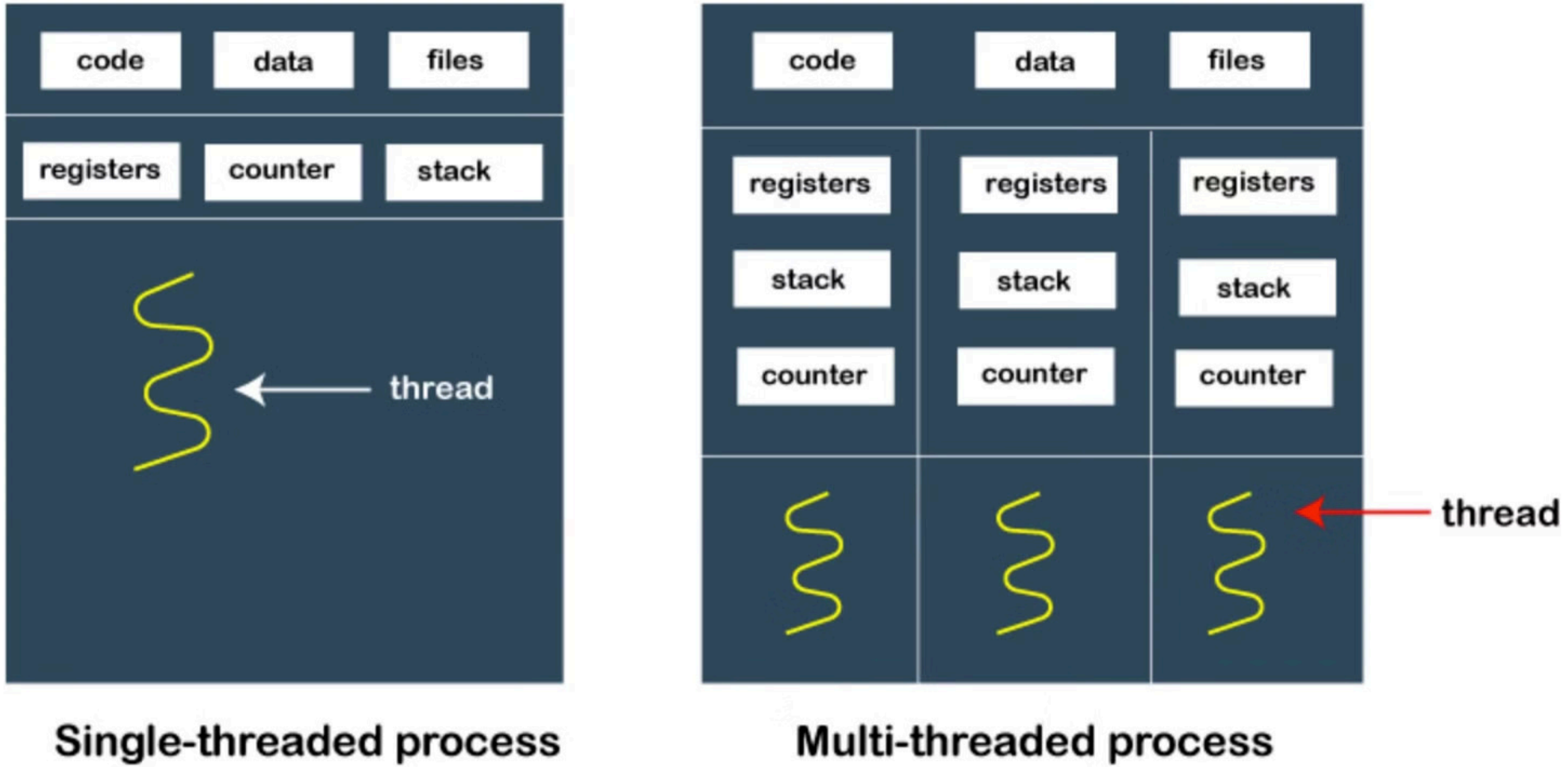
Process

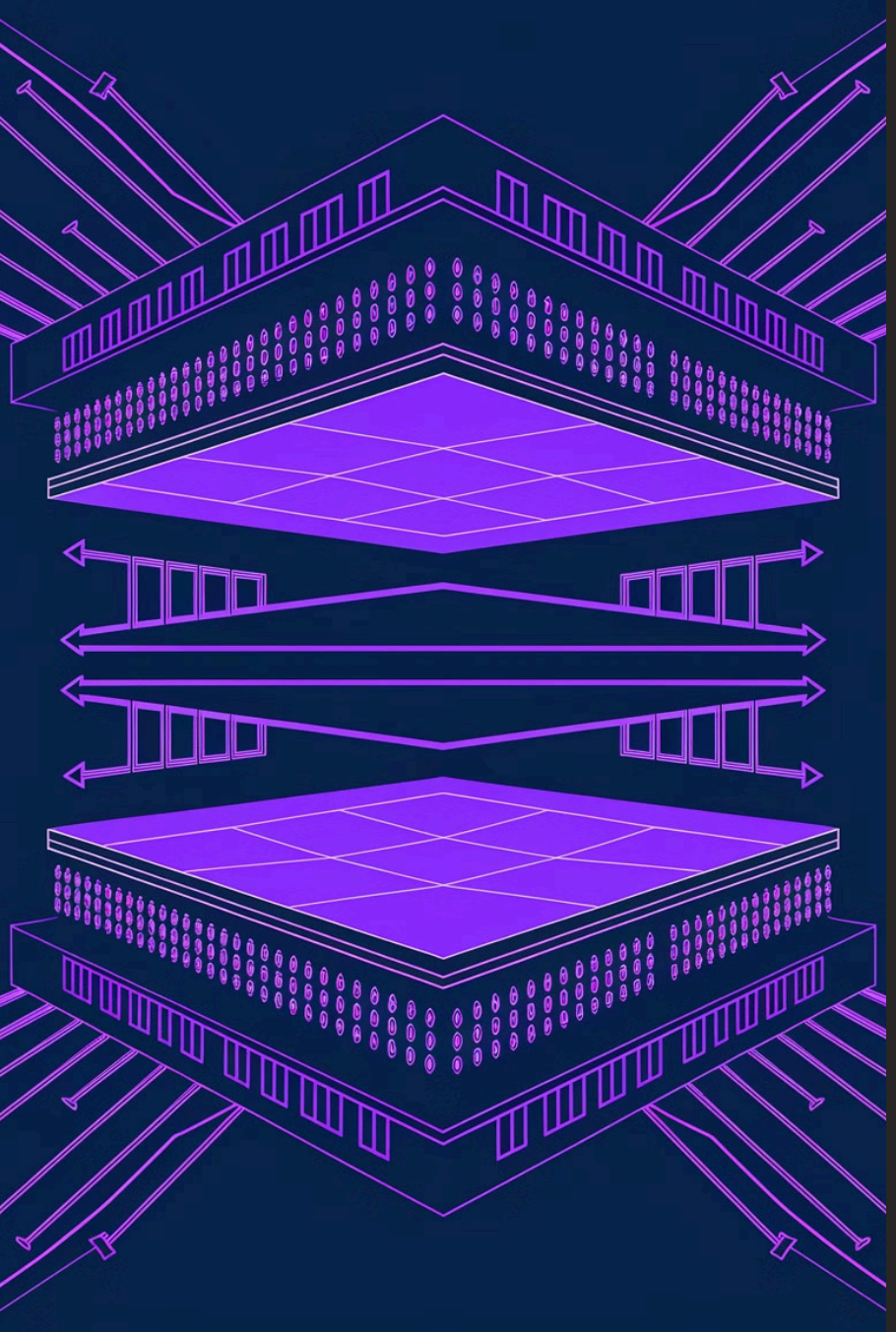
An independent running instance of a program with its own address space, memory, and data.

Thread Defined

1. Smaller unit of execution within a process.
2. A single sequence of instructions, that can be scheduled by the system's scheduler
3. Multiple threads within a process **share** the same memory space. (Not the stack!)

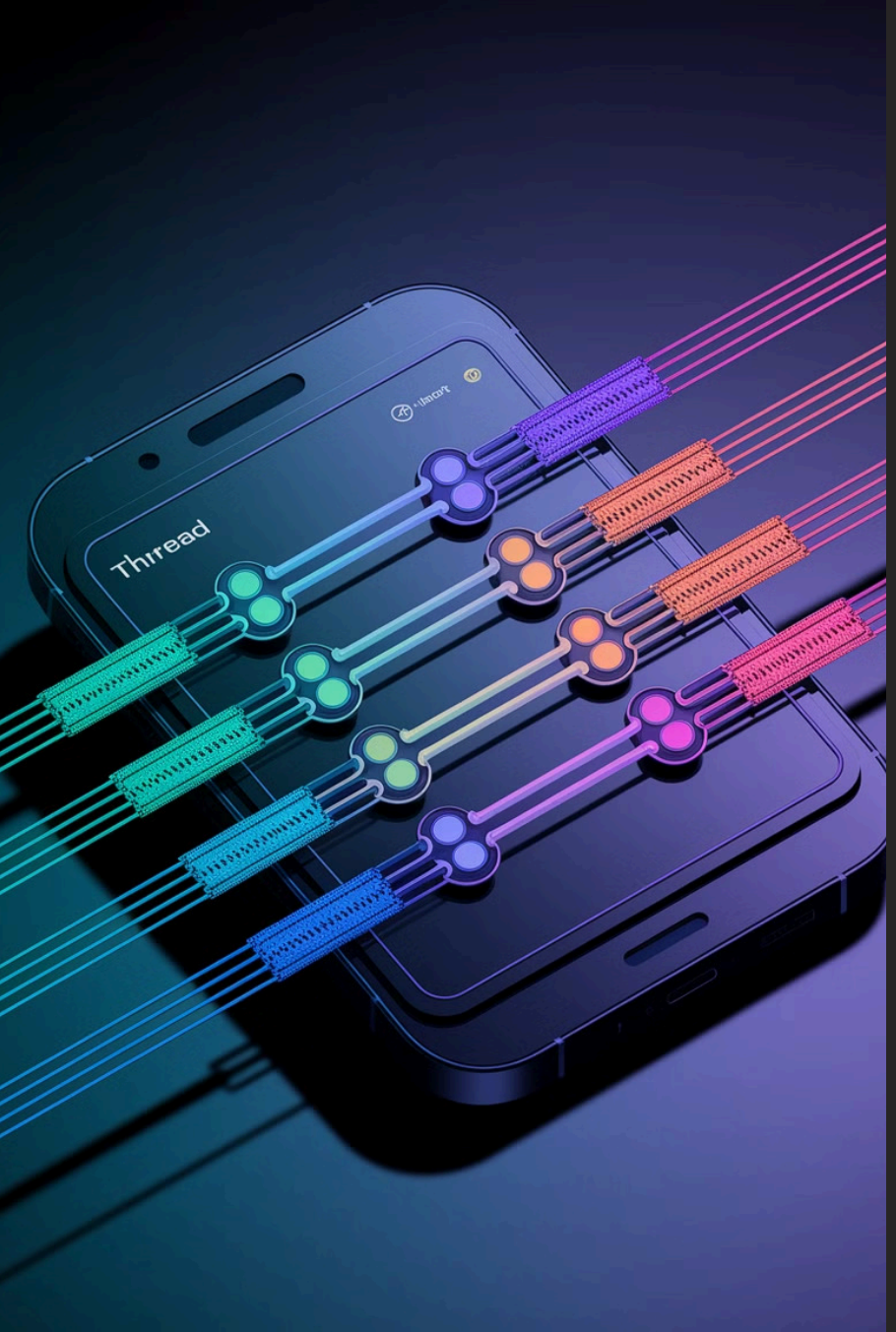
Processes vs Threads





Types of Threads

Threads can be kernel-level or user-level. Each has different characteristics.



User-Level Threads

Thread management is done by the application. The kernel is not aware of threads.

Also called virtual or green threads

Kernel-Level Threads

All Thread management is done by the kernel. The application uses an API to interact to the kernel

Kernel vs User Level Threads

Feature	User Level Threads	Kernel-Level Threads
Implementation	In User Space	In Kernel Space
Context Switch Time	**Fast** 😄	Slow
Memory Consumption	**Low** 😄	Higher
Scheduling	By User-Library	**By OS** 😄
Blocking	One Thread Can Block All	**Independent** 😄
Multi-CPU Use	None or Limited	**Full** 😄

Summary

1. Concurrent programming history.
2. Scheduling types.
3. Processes and context switch.
4. Concurrency vs Parallelism.
5. Threads. Kernel vs user level threads.