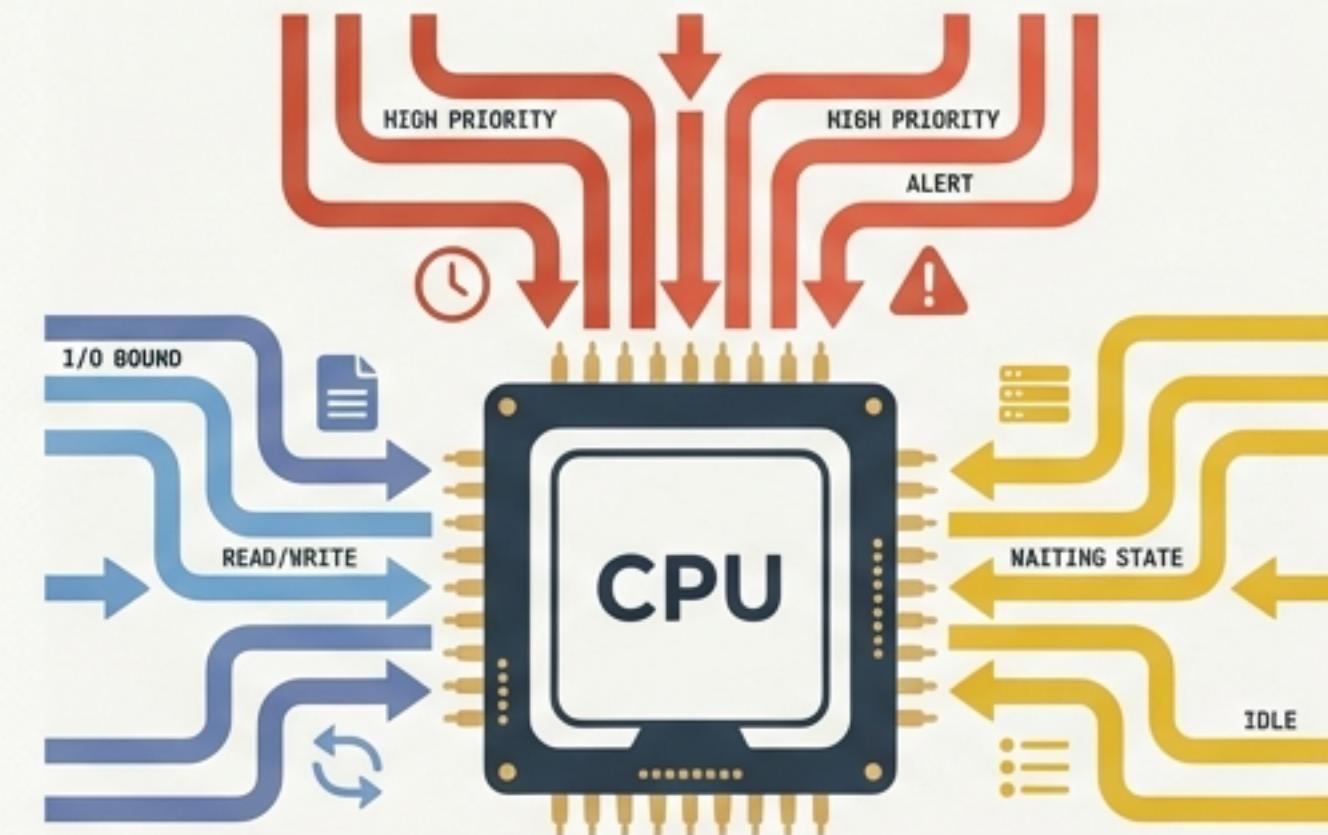


CPU 스케줄링 개요

운영체제가 자원을 공정하고 합리적으로 배분하는 기술



시스템 성능 결정



CPU 스케줄링은 컴퓨터 시스템의 전반적인 성능을 좌우하는 핵심 요소입니다.

자원의 효율성



합리적인 기준 없이 자원을 배분하면, 중요한 작업이 지연되고 시스템 효율이 저하됩니다.

핵심 목표



한정된 CPU 자원을 어떻게 배분해야 공정하고 효율적일까요?

단순한 순서대로 처리하면 안 되는 이유

먼저 요청한 프로세스에게 무조건 CPU를 주는 것이 합리적일까요?

Timeline Comparison

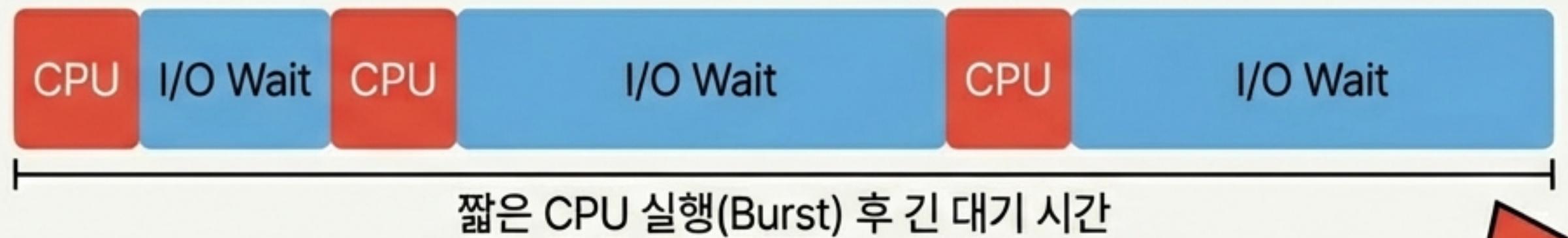


문제점 (The Conflict)

- 프로세스마다 긴급성과 자원 소모량이 다릅니다.
- 단순한 선착순 처리는 급한 작업이 덜 중요한 긴 작업에 밀려 시스템 전체를 전체를 느리게 만듭니다.
- 해결책: 도착 순서가 아닌 **우선순위(Priority)**가 필요합니다.

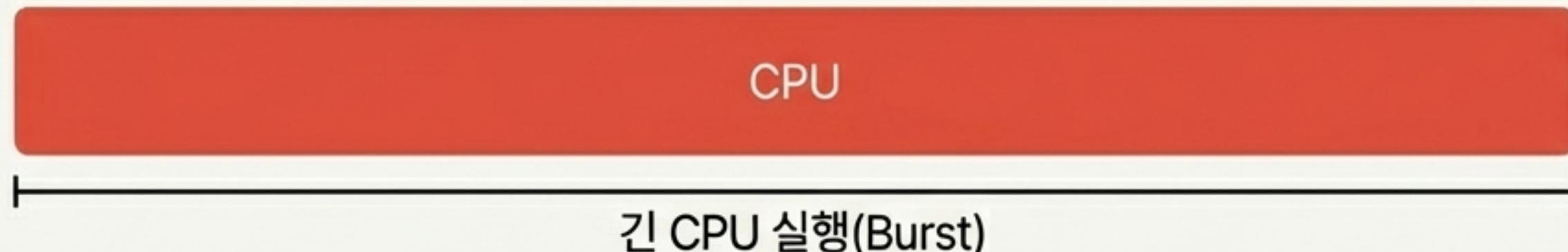
우선순위의 기준: I/O 집중 vs CPU 집중

입출력 집중 프로세스 (I/O Bound Process)



비디오 재생, 디스크 백업 등
입출력 장치를 주로 사용하며
CPU는 잠깐씩만 사용합니다.

CPU 집중 프로세스 (CPU Bound Process)

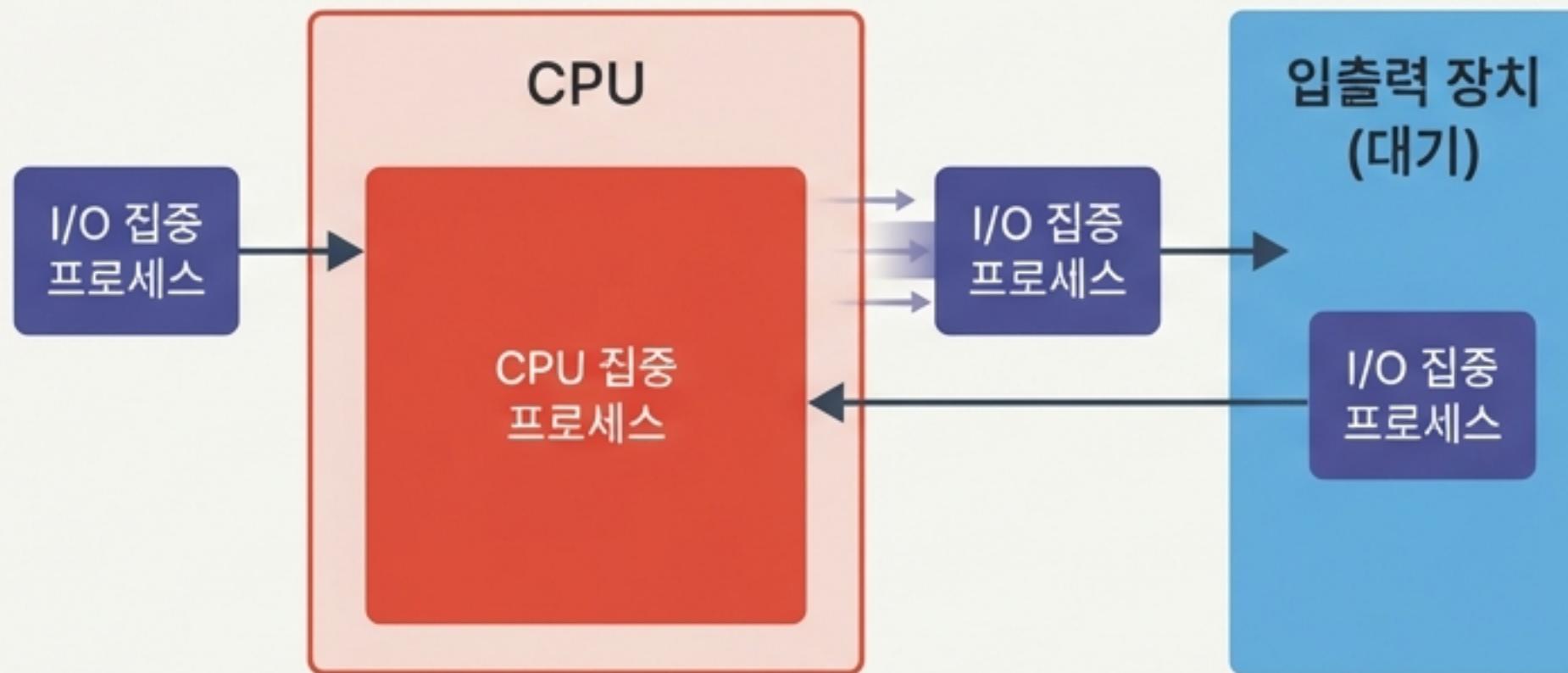


더 높은 우선순위 부여

복잡한 수학 연산, 컴파일,
그래픽 렌더링 등 CPU 연산이
주된 작업입니다.

왜 입출력 집중 프로세스에게 높은 우선순위를 줄까요?

자원 활용의 최적화



논리 (The Logic)

- 입출력 집중 프로세스는 CPU를 아주 잠깐만 사용하고 곧바로 대기 상태로 빠집니다.
- 이들을 먼저 처리해버리면, 그들이 입출력 장치에서 대기하는 동안 다른 프로세스가 CPU를 사용할 수 있습니다.

핵심: 입출력 집중 프로세스를 빨리 처리하면, 다른 프로세스가 CPU를 사용할 시간을 더 많이 확보할 수 있어 시스템 전체 처리량이 증가합니다.

우선순위의 저장과 확인: PCB

PCB (Process Control Block)

PID: 3412

Process State: Ready

Priority: High (10)

Program Counter: 0x4F...

Registers...

우선순위 확인 (Mechanism)

- 모든 프로세스는 자신의 정보를 담은 **PCB(프로세스 제어 블록)**를 가집니다.
- 운영체제는 스케줄링 시 PCB 내부의 '우선순위(Priority)' 필드를 확인하여 순서를 결정합니다.

사용자 제어 (User Control)

- 대부분 자동이지만, 사용자가 직접 조정할 수도 있습니다.
- 예: Linux의 `nice` 명령어, Windows 작업 관리자의 우선순위 설정.

비효율성을 해결하는 방법

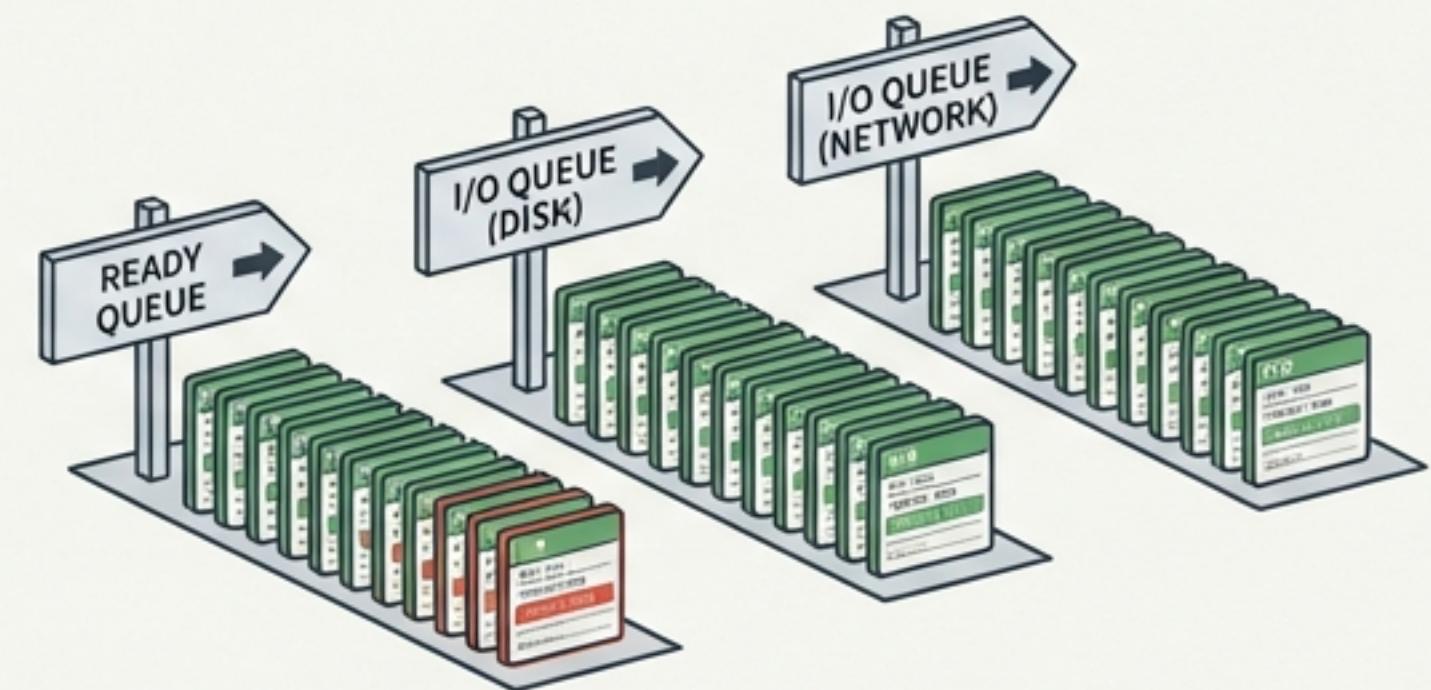
Chaotic Scanning vs. Organized Queues

비효율적 방식: 전체 탐색



운영체제가 수많은 PCB를 매번 전부 뒤져서
우선순위가 높은 것을 찾는 것은 매우 느립니다.

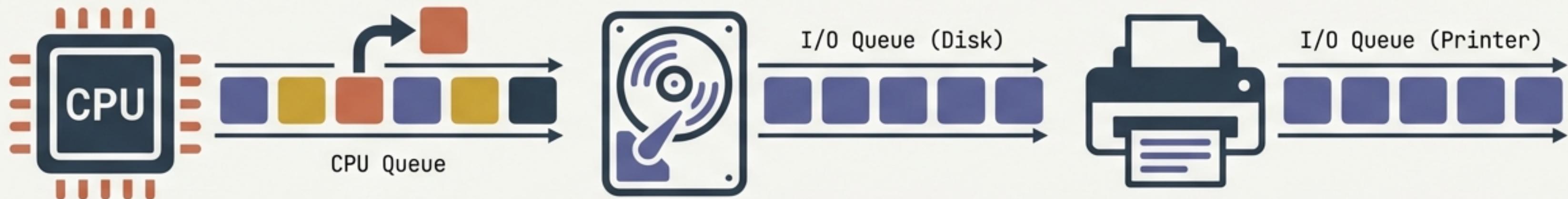
해결책: 스케줄링 큐



자원별로 줄(Queue)을 세워 관리합니다.

운영체제는 모든 PCB를 검사하는 대신, **스케줄링 큐(Scheduling Queue)**를 도입하여 프로세스를 체계적으로 관리합니다.

스케줄링 큐란 무엇인가?



정의 (Definition)

- 특정 자원(CPU, 메모리, 입출력 장치 등)을 사용하기 위해 프로세스들이 대기하는 줄입니다.

특징 (Distinction)

- 일반적인 큐(Queue) 자료구조는 선입선출(FIFO)이지만,
- **스케줄링 큐**는 우선순위가 높은 프로세스가 먼저 처리될 수 있습니다.

핵심 큐: 준비 큐와 대기 큐



1. 준비 큐 (Ready Queue)

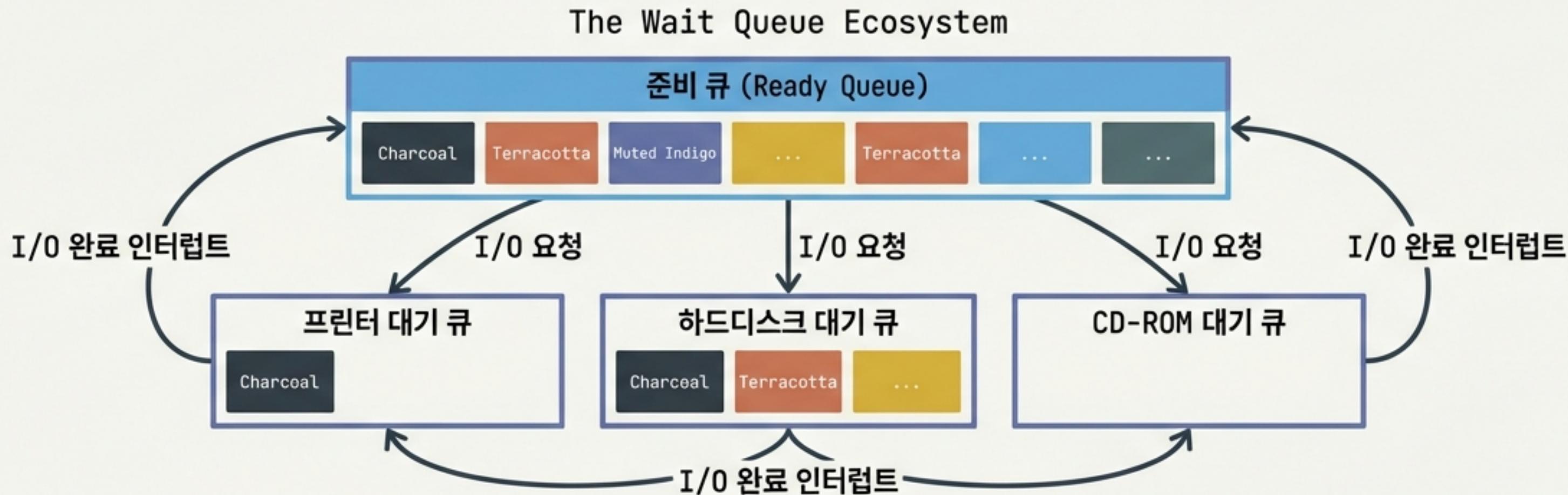
- 준비 상태 (Ready State)의 프로세스들이 모여 있습니다.
- CPU를 할당받기 위해 기다리는 줄입니다.
- 차례가 오면 즉시 실행(Dispatch)됩니다.



2. 대기 큐 (Wait Queue)

- 대기 상태 (Wait State)의 프로세스들이 모여 있습니다.
- 입출력(I/O) 작업이 완료되기를 기다리는 줄입니다.
- 입출력이 끝나면 다시 준비 큐로 이동합니다.

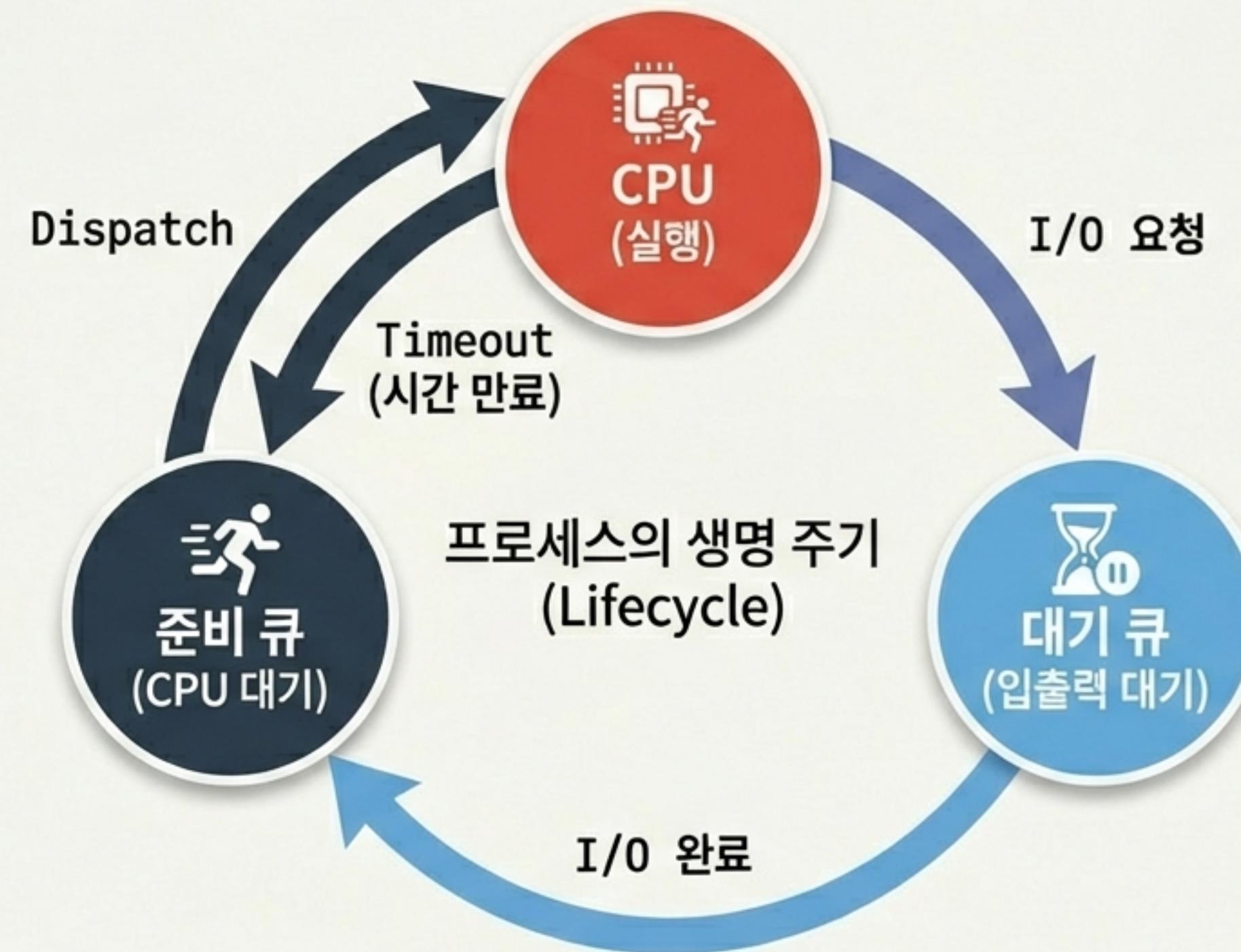
장치별로 분리된 대기 큐



대기 큐는 하나가 아닙니다.

- 프린터, 하드디스크 등 각 장치마다 별도의 큐가 존재합니다.
- 프로세스는 특정 장치의 작업이 끝나면 인터럽트를 통해 다시 준비 큐로 복귀합니다.

프로세스 상태와 큐의 흐름

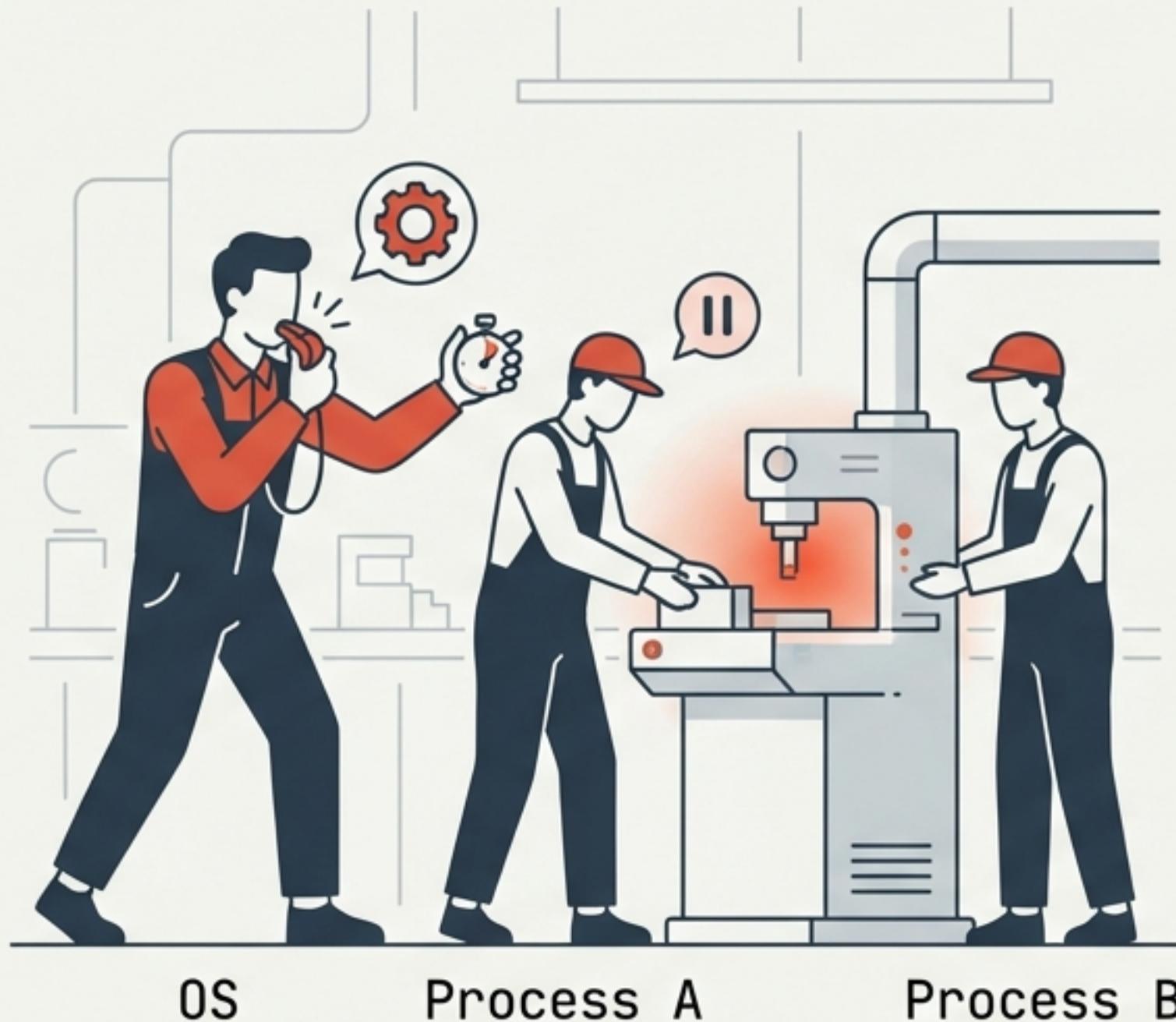


인터럽트와 복귀



운영체제는 하드웨어 인터럽트를 감지하여, 작업이 끝난 프로세스를 즉시 대기 큐에서 꺼내어 준비 큐로 옮깁니다.

선점형 스케줄링 (Preemptive Scheduling)



정의 (Definition)

- 프로세스가 CPU를 사용 중이더라도, 운영체제가 강제로 CPU 자원을 뺏을 수 있는 방식입니다.
- 트리거: **타이머 인터럽트**(시간 종료), **더 높은 우선순위 프로세스** 등장.

장점과 단점

- 장점: 특정 프로세스의 자원 독점을 방지하고 골고루 자원을 배분합니다.
- 단점: 잦은 **문맥 교환(Context Switch)**으로 인한 오버헤드가 발생합니다.

비선점형 스케줄링 (Non-preemptive Scheduling)



정의 (Definition)

- 프로세스가 한 번 CPU를 잡으면, 작업이 끝나거나 스스로 대기 상태에 들어갈 때까지 운영체제가 개입하지 못합니다.

장점과 단점

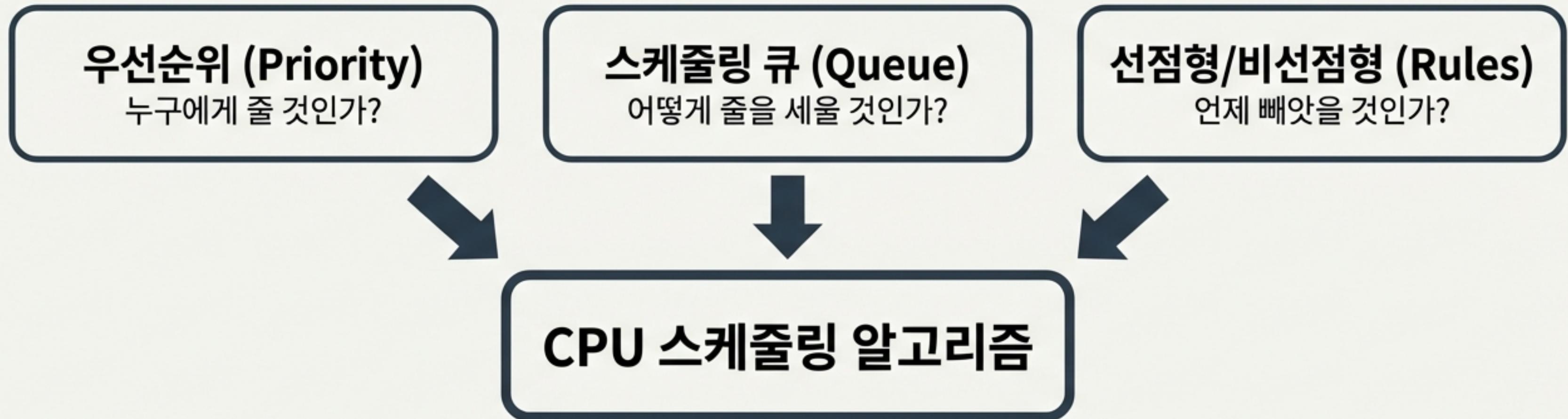
- 장점: 문맥 교환 횟수가 적어 오버헤드가 낮습니다.
- 단점: 하나의 프로세스가 자원을 독점하면 시스템 전체가 멈출(Freeze) 수 있습니다. 응답성이 낮습니다.

선점형 vs 비선점형 비교

구분	선점형 (Preemptive)	비선점형 (Non-preemptive)
제어권 (Control)	운영체제가 강제 회수 가능	프로세스가 자발적으로 반납
오버헤드 (Overhead)	높음 (잦은 문맥 교환)	낮음
응답성 (Responsiveness)	높음 (모든 프로세스가 골고루 실행)	낮음 (독점 가능성 있음)
활용 (Usage)	대부분의 현대 운영체제	일괄 처리 시스템 등 특수 목적

현대의 범용 운영체제는 멀티태스킹의 유연성을 위해 대부분 **선점형 스케줄링**을 채택하고 있습니다.

요약 및 다음 단계



지금까지 스케줄링의 구조와 규칙을 알아보았습니다.
다음 단계에서는 실제로 어떤 수학적 알고리즘으로 프로세스를 선택하는지 알아봅니다.

- FCFS (First Come, First Served)
- SJF (Shortest Job First)
- Round Robin