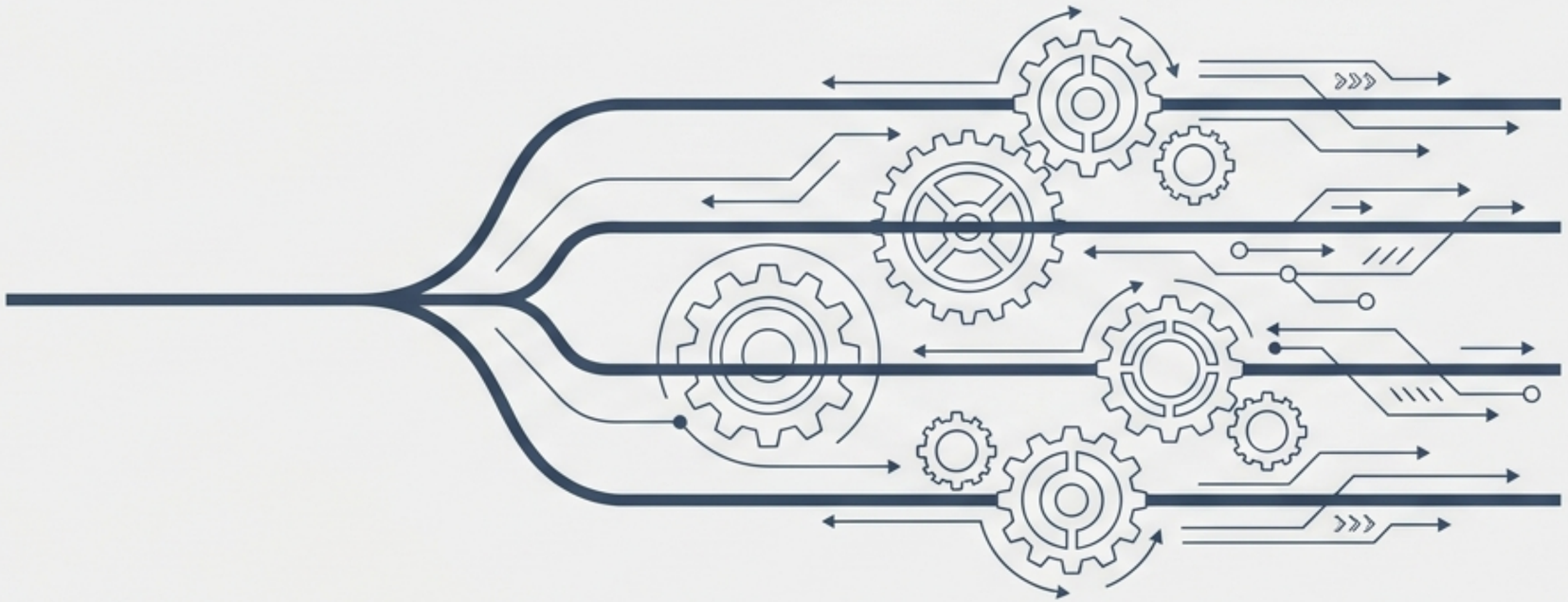
같은 시간에 여러 일을 한다는 것

CS 스터디 핵심 요약: 동시성과 병렬성의 진짜 의미



Multi-process

Thread

Coroutine

Multi-poess

Isolate

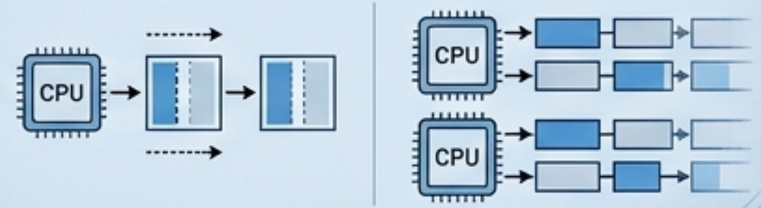
asyncio

Fscnas

goroutine

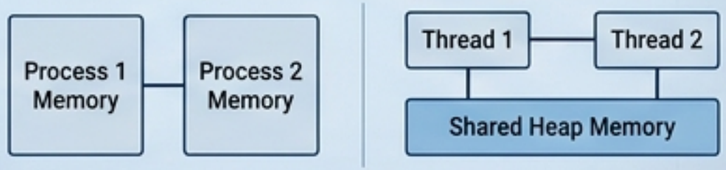
1. 진짜로 동시에 도는가?

(단일 코어 번갈아 실행 vs 물리적 멀티코어 병렬)



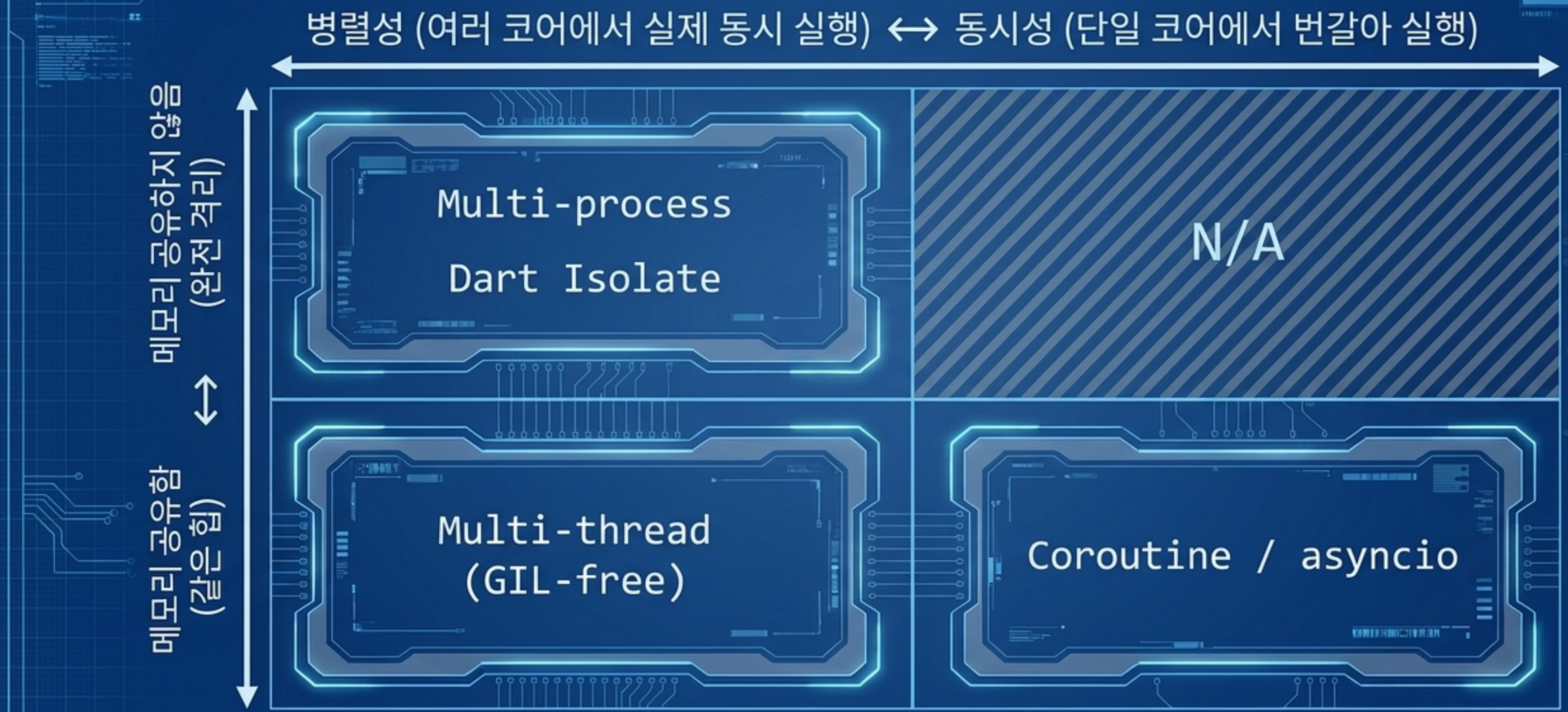
2. 메모리를 공유하는가?

(독립된 주소 공간 vs 힙 공유)



멀티프로세스, 스레드, 코루틴을 뭉뚱그려 이해하면 실전에서 반드시 한계에 부딪힙니다.
이들을 명확히 구분하는 기준은 단 두 가지입니다.

동시성과 병렬성의 2x2 멘탈 모델



Multi-process: 완벽한 격리와 묵직한 비용

안전성 & 격리	높은 비용
한 프로세스가 죽어도 다른 프로세스 생존. Race condition 구조적 원천 차단.	OS 스케줄러 단위. 주소공간/파일 디스크립터 복제로 인해 생성 및 통신 비용이 매우 비쌈.

`C/C++ (OS API)` `Python (multiprocessing)` `Rust (std::process)`

Dart Isolate: 메시지로만 대화하는 경량화된 격리망

스레드처럼 가볍지만 힘을 전혀 공유하지 않는(share-nothing) 하이브리드 구조.
Flutter에서 UI 스레드 멈춤(Jank)을 완벽히 방지하기 위한 언어적 설계 철학.

Execution process/mini-map

Multi-process	Dart Isolate
Multi-thread	

Multi-thread: 빠르고 강력한, 그러나 위험한 공유 힙

프로세스보다 생성과 통신(메모리 직접 참조)이 훨씬 빠르지만,
동기화(Lock) 설계가 어긋나면 치명적인 버그 발생.

스레드를 대하는 언어의 철학: Python vs Rust

Python

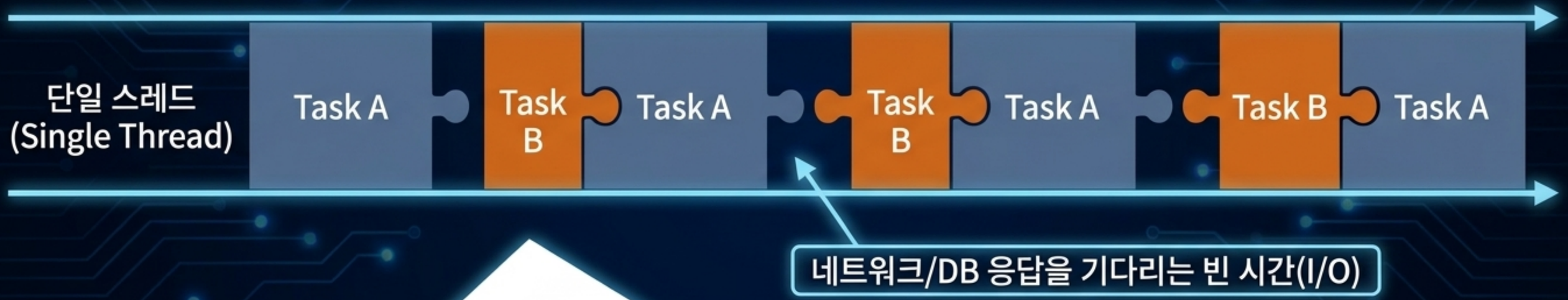
CPython 메모리 관리를 위한 트레이드오프.
(최근 3.13+에서 실험적 free-threaded 도입 중)

Rust

런타임 Lock에 의존하지 않고,
컴파일러(Ownership/Borrow Checker)가
Data Race를 원천 차단.

Execution Blueprint

Coroutine & asyncio: 진짜 동시성의 마법



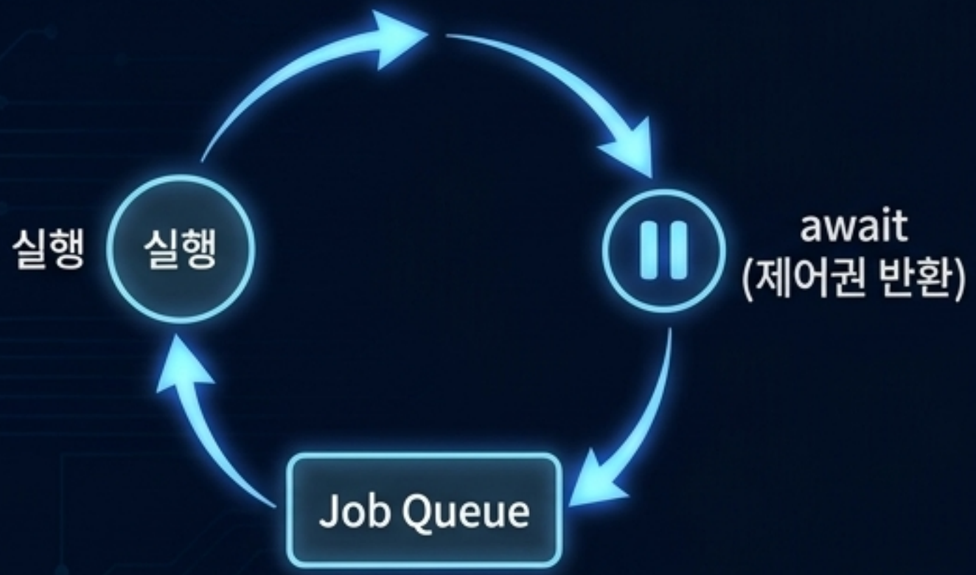
두 색이 겹쳐서 실행되는 구간은 전혀 없다.
물리적 병렬 연산이 아닌, 치밀한 작업 스위칭.

Execution Blueprint

Coroutine & asyncio: 진짜 동시성의 마법

메커니즘: 수만 개의 연결을 버티는 이유

빠르고 가벼운 상태 기계(State Machine)

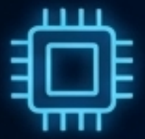


Context Switching Cost

CPU를 쪼개 쓰는 것이 아니라 대기 시간을 쪼개 쓴다. 비용이 거의 0에 가까워 채팅 서버 등에 최적.

실전 가이드: 언제 무엇을 써야 하는가?

작업의 가장 큰 병목이 무엇인가?



무거운 수학 연산, 이미지 처리

↓

병렬성(Parallelism) 필요

↓

Multi-process / Multi-thread 권장

(암달의 법칙 주의: 코어 수만큼 무조건 빨라지진 않음)

네트워크 요청, DB 쿼리, 파일 읽기

↓

동시성(Concurrency) 최적화

↓

Coroutine / asyncio 권장

초보자의 흔한 함정: 코루틴에 무거운 연산(CPU-bound)을 던지면 단일 스레드가 블로킹되어 전체 시스템이 멈춥니다.

5개 언어별 동시성/병렬성 지원 마스터 매트릭스

언어	Multi-process	Dart Isolate	Multi-thread	Coroutine/asyncio
C	✔ O (표준 fork/exec)	- - - - - - — - - - - - -	✔ O (pthread)	— (표준 없음)
C++	✔ O (OS API)	—	✔ O (std::thread)	⚠ 부분 지원 (co_await + 별도 실행기)
Python	✔ O (multiprocessing)	—	⚠ 부분 지원 (GIL / 3.13+ 실험적 free-threaded)	✔ O (asyncio)
Dart	✔ O (Process.start)	✔ O (고유 지원)	— (비노출, isolate로 추상화)	✔ O (async/await)
Rust	✔ O (std::process)	—	✔ O (std::thread + 컴파일타임 안전)	⚠ 부분 지원 (async/await + tokio 등 별도 실행기)

CS 전공생이 기억해야 할 3가지 절대 원칙

동시성은 구조, 병렬성은 자원이다.

동시성은 작업을 어떻게 엮을지의 문제이고,
병렬성은 하드웨어를 어떻게 나눌지의 문제다.

공유는 빠르지만 위험하고, 격리는 안전하지만 비싸다.

상태를 공유하는 스레드는 Lock이 필수이며, 메모리를 격리하는
프로세스/Isolate는 통신(IPC/메시지) 비용을 치러야 한다.

작업의 성격(CPU vs I/O)이 도구를 결정한다.

I/O 병목에 스레드를 낭비하거나, 연산 병목에 코루틴을 집어넣는 실수를 피하라.

The Execution Blueprint — End of presentation