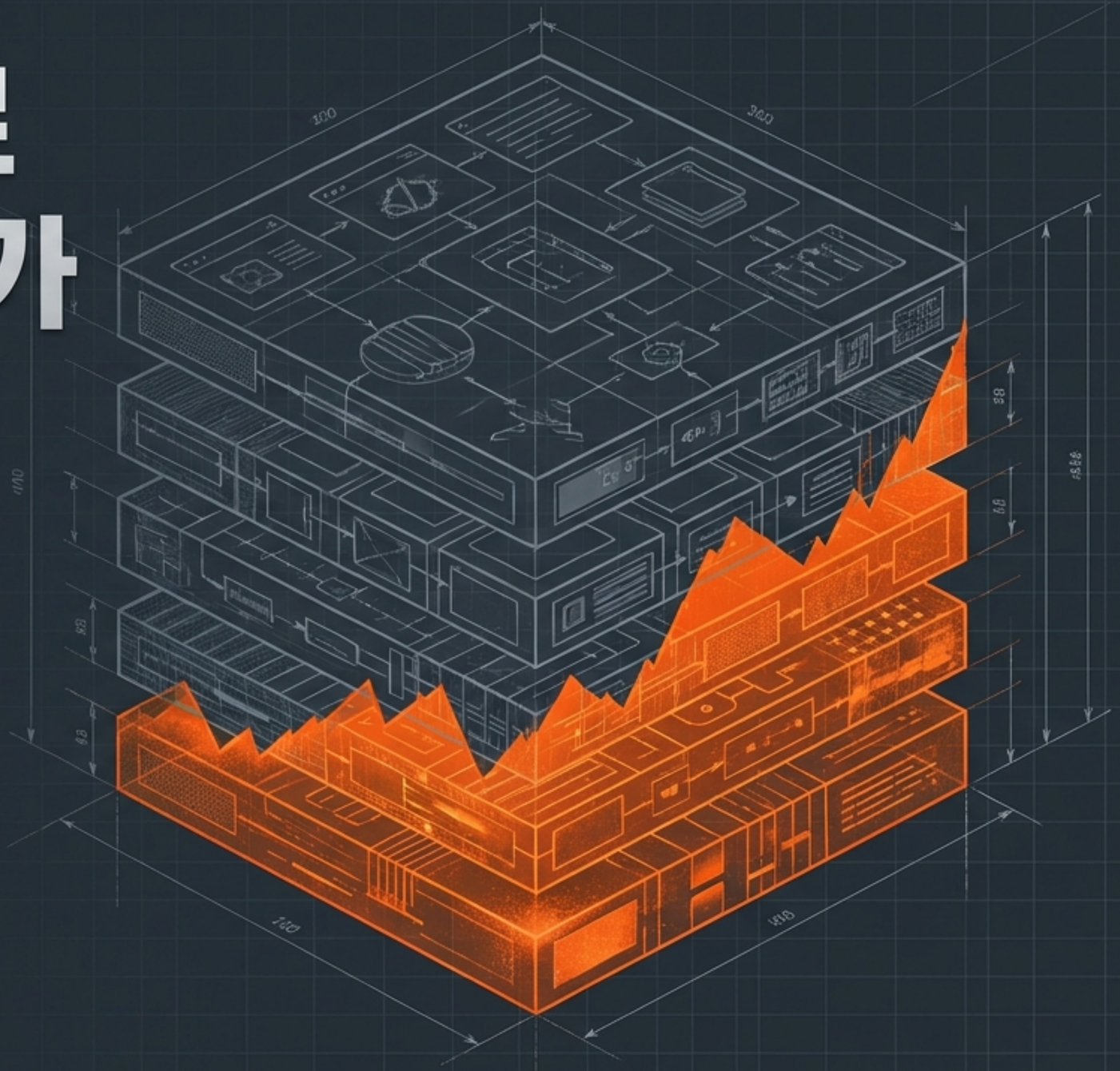
Rust는 실제로 어디에 쓰이는가

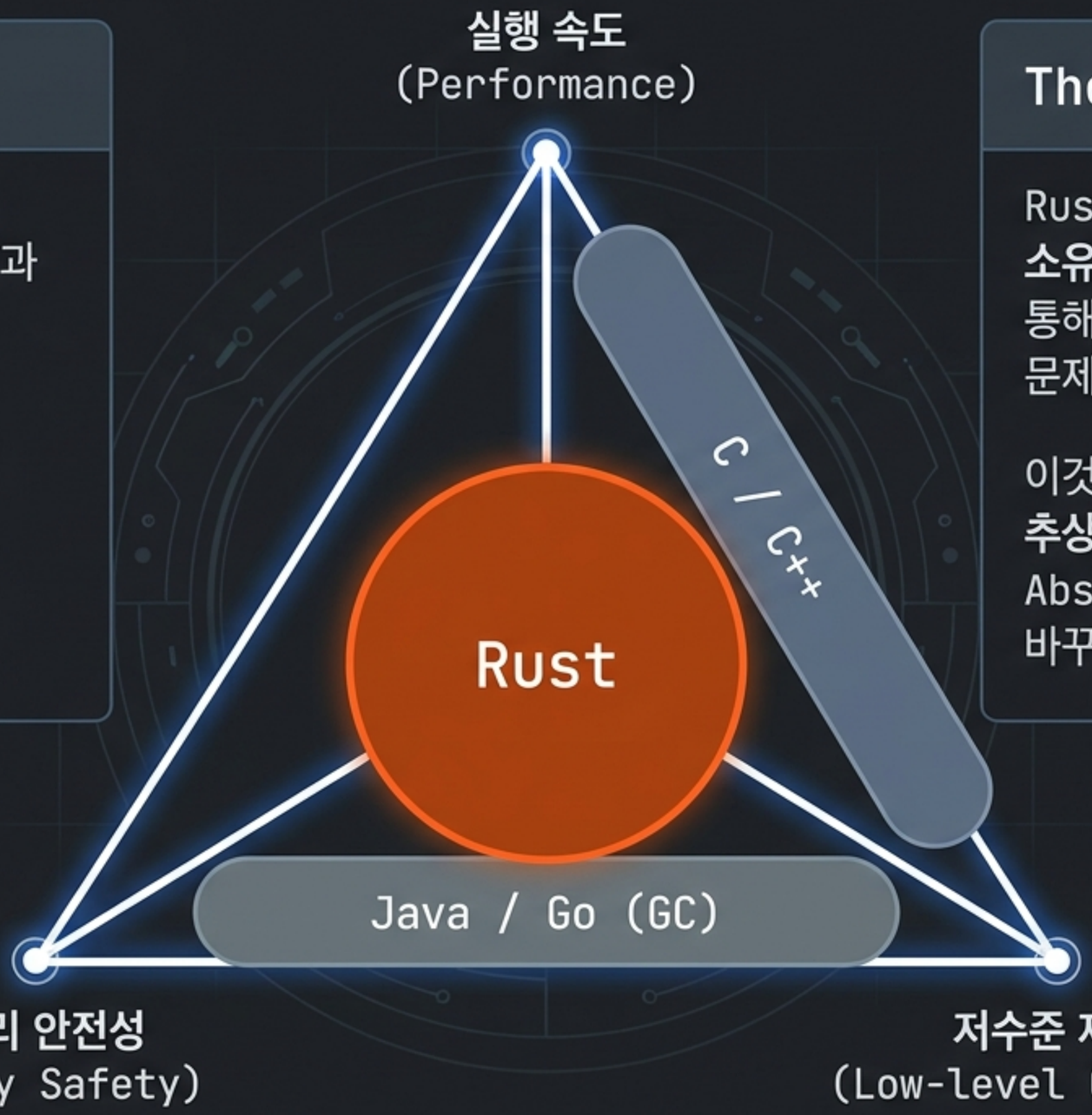
CS 학부생을 위한
산업 현장 리포트



언어 문법이 아닌, 시스템 아키텍처와
산업 표준이 교체되는 과정을
6가지 실사례로 추적합니다.

The Problem

50년 동안 커널 및 시스템 프로그래밍은 **안전성**(GC)과 **속도**(C/C++) 중 하나를 포기해야 했습니다.



The Catalyst

Rust는 런타임 오버헤드 없이 **소유권**(Ownership) **규칙**을 통해 컴파일 타임에 메모리 문제를 해결합니다.

이것이 **'제로 비용 추상화**(Zero-cost Abstraction)'가 산업을 바꾸는 방식입니다.

The Target Stack: 스택 침투 경로

성능이 필요한 곳에는 늘 C/C++가 있었습니다.
이제 그 자리는 점진적으로 덮어씌워지고 있습니다.

Layer 1: OS Kernel (Linux & Windows)

2022
Rust for Linux 프로젝트 시작
(실험적 단계)

2025.12
커널 메인테이너, ‘Rust 지원은
더 이상 실험적이지 않다’ 선언

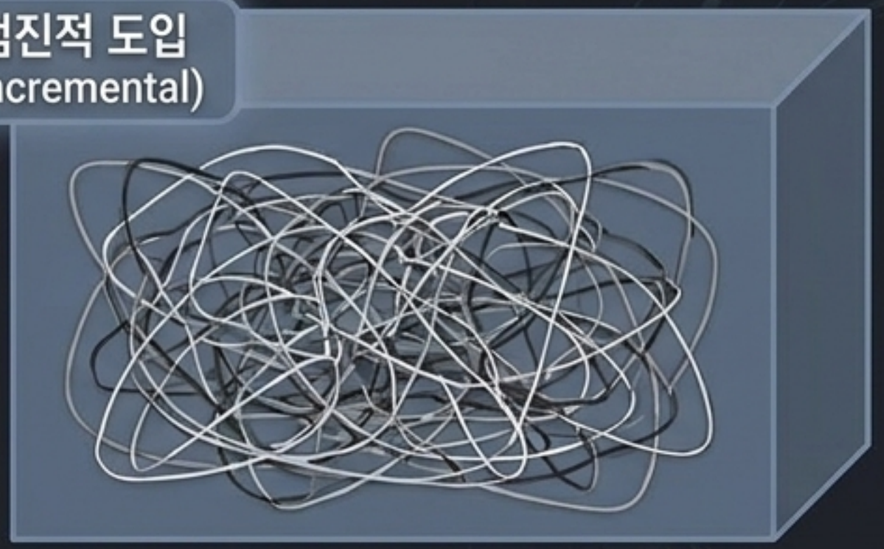
2026.02
Linux 7.0 출시
(‘Rust 실험 종료’ 공식 명시)

2030
Microsoft, Windows
코드베이스에서 C/C++ 제거 목표

Layer 2: Experimental OS (Redox OS)

Monolithic Kernel (수천만 줄의 C 코드)

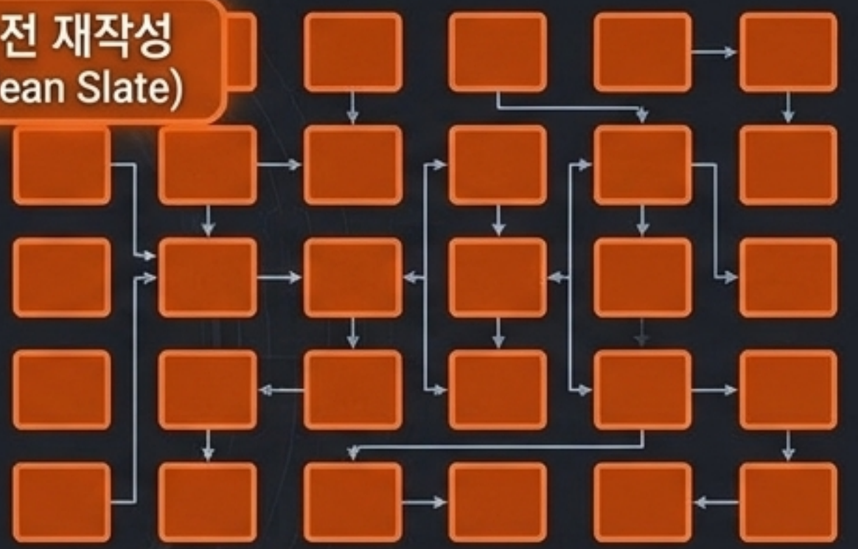
점진적 도입
(Incremental)



레거시 호환성을 유지하며 격리된
드라이버 영역부터 Rust를 침투시킵니다.

Microkernel Architecture

완전 재작성
(Clean Slate)



레거시 없이 처음부터 Rust의
설계 원칙 위에 세워진 실험적 OS.

학부생을 위한 인사이트

방대한 레거시 코드를 읽는 것보다, Redox OS의 마이크로커널 코드를 분석하는 것이
최신 시스템 아키텍처를 학습하는 훨씬 효과적인 방법일 수 있습니다.

Layer 3: Core Utilities (The Security Rewrite)

Corporate Integration

- Prossimo 이니셔티브에 의한 완전 재작성.
- 2025.10: Ubuntu 25.10 기본 탑재.
- 2026.04: Ubuntu 26.04 LTS 기본 탑재 확정 (메모리 안전 sudo 최초 LTS 적용).

Layer 3: Core Utilities (The Speed Rewrite)

1970~80년대 C 코드가 지배하던 영역의 세대 교체.
이 영역에서 Rust의 도입 동기는 '보안' 이전에 '압도적인 검색 및 실행 속도'입니다.

Layer 4: Python Toolchain (Fast Infrastructure for Slow Languages)

1. ruff — Flake8 등 수십 개 툴 대체 (수십~수백 배 속도 향상).
2. uv — 패키지 매니저 통합. (일반 캐시 환경 최대 100배, 노캐시 10배 빠른 설치 - Astral 공식 벤치마크).
3. pydantic-core / polars — 데이터 연산 병목 구간 재작성.

비즈니스 제품 전략: 느린 생태계의 인프라를 Rust로 재작성하는 것은 단순한 리팩토링이 리팩토링이 아니라, 스타트업(Astral)이 시장을 장악하는 핵심 전략으로 작용했습니다.

Layer 5: WebAssembly (The Dual Arena)

WASM은 GC가 없는 저수준 바이트코드입니다.
GC가 없는 Rust는 WASM 생태계의 사실상 '기본 언어(Default)'가 되었습니다.

브라우저 환경

`wasm-bindgen, wasm-pack`

Use Case: 순수 JS로 성능이 부족한 이미지/오디오 처리, 게임 엔진 연산.

Adoption: Figma, 1Password 웹 클라이언트.

서버 및 엣지 환경

`WASI, Wasmtime, Wasmer`

Use Case: 컨테이너를 대체하는 경량 컴퓨팅.
Catalyst: 컨테이너 대비 압도적으로 빠른 콜드 스타트(Cold Start).

Adoption: Cloudflare Workers, Fastly Compute.

C/C++가 지키던 마지막 요새, GPU 메모리가 열리다.

2026.09.08 — NVIDIA, ‘CUDA Rust’ 공식 발표 (커뮤니티 비공식 실험에서 제작사 공식 투자로 전환)

Track 1: cuda-oxide (SIMT 모델)

기존 CUDA C++와 동일한 스레드 단위 프로그래밍.

MIR → LLVM IR → PTX로 직접 컴파일.

초기 Alpha 단계

Track 2: cutile-rs (Tile 모델)

- 데이터 묶음(타일) 단위 연산 기술 (최신 모델).
- Hugging Face 추론 엔진(Grout) 채택.

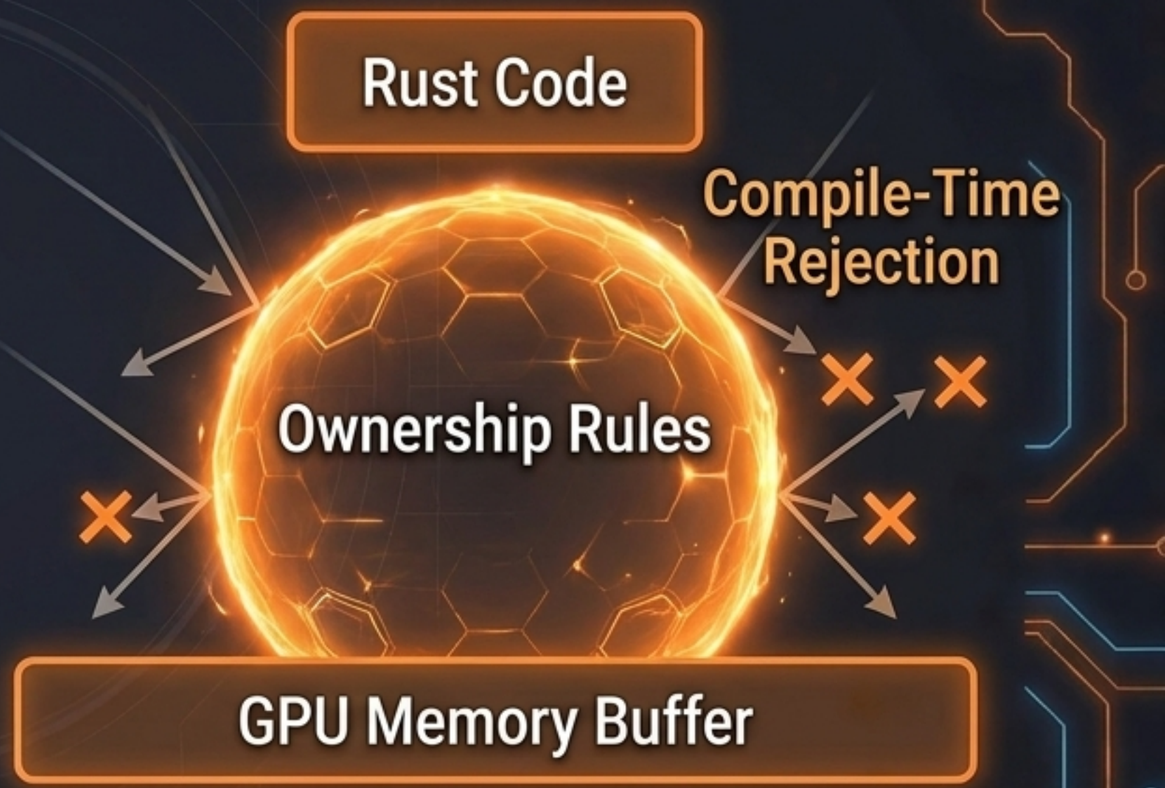
Rust 1.89+ (Stable)

The GPU Compile-Time Advantage

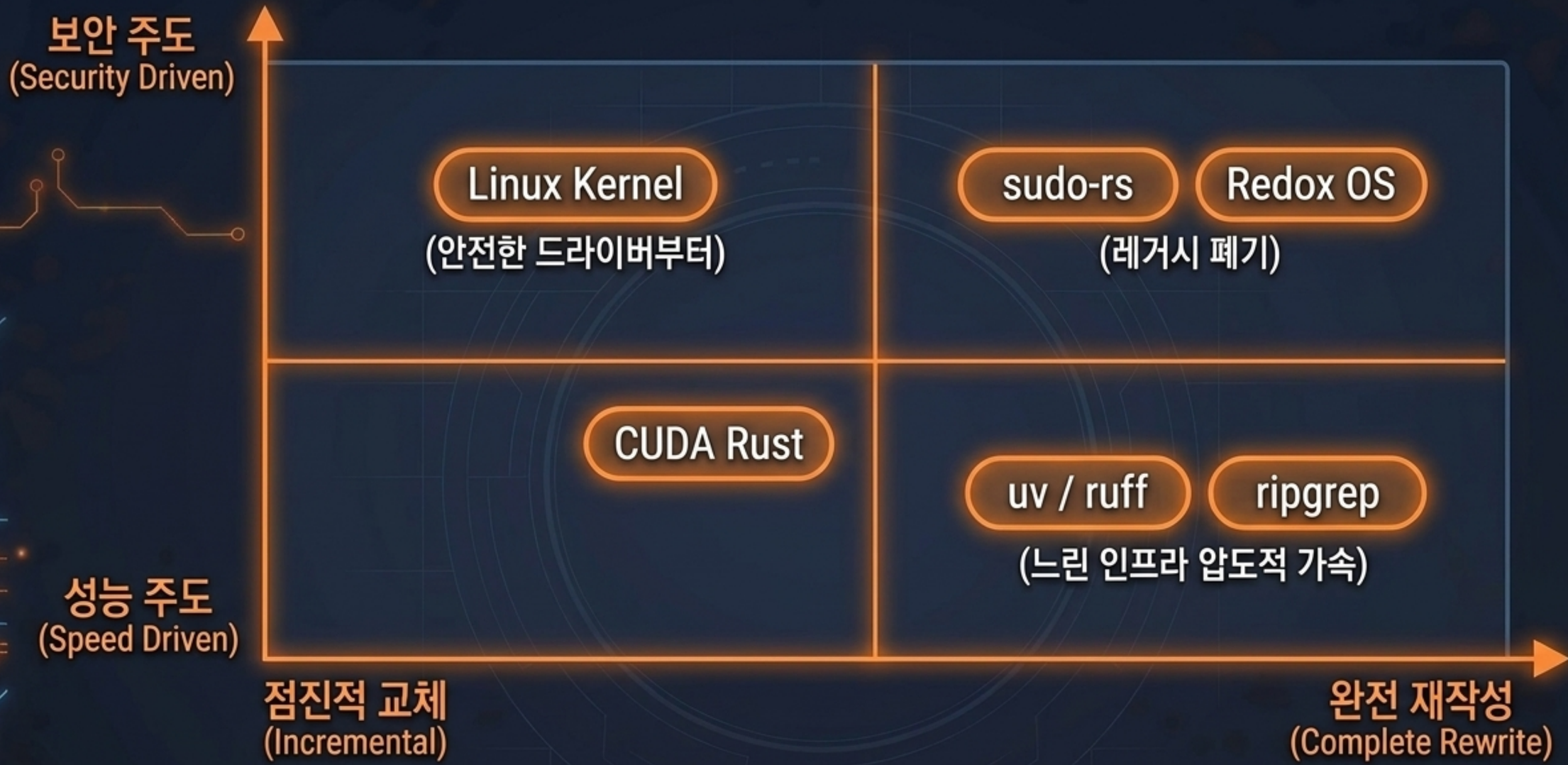
C++ CUDA - The Runtime Explosion

출력 버퍼를 입력으로 잘못 넘겨도
실행 후에야 충돌을 발견.

Rust CUDA - The Compile-Time Shield



수천 개의 스레드 메모리 접근 충돌을
컴파일 단계에서 원천 차단.



산업 내 Rust 도입은 단일한 형태가 아닙니다. 레거시의 민감도(보안)와 생태계의 병목(속도)에 따라 도입의 형태(점진/완전)가 결정됩니다.

2025-2026: 커뮤니티 실험에서 글로벌 기업 표준으로

더 이상 “Rust가 트렌드인가?”를 묻는 시기는 지났습니다.
하드웨어, OS, 시스템 인프라 전반에서 글로벌 빅테크의
공식 보증(Corporate Endorsement) 단계가 시작되었습니다.

다음 세대 엔지니어를 위한 질문

1

Architecture

왜 어떤 시스템(Linux)은 **점진적 교체**를 선택하고,
어떤 시스템(sudo, Redox)은 **완전 재작성**을 선택했을까요?

2

Catalyst

sudo-rs처럼, **속도가 아닌 '보안'**이 언어 교체의 직접적 계기가 될
다음 인프라는 어디일까요?

3

Market Dynamics

엔비디아의 발표처럼, **한 회사의 투자가 전체 생태계의 표준**을 순식간에 바꿉니다.
이 리포트를 읽는 지금, **또 어떤 생태계가 덮어씌워지고** 있을까요?

[SYS_LOG] Source Verification

> Rust for Linux 로드맵: rust-lang.github.io/goals
> Linux 7.0 보도: Phoronix, 2026.02
> State of Rust 2026 (Android 취약점, MS 2030): devnewsletter.com
> CVE-2021-3156 & sudo-rs: memorysafety.org, The Register, The New Stack
> Astral Tools Benchmark: uv 공식 문서, ripgrep GitHub README
> CUDA Rust 공식 발표: NVIDIA Technical Blog (introducing-cuda-rust-two-tracks), 2026.09.08

[END OF LOG]