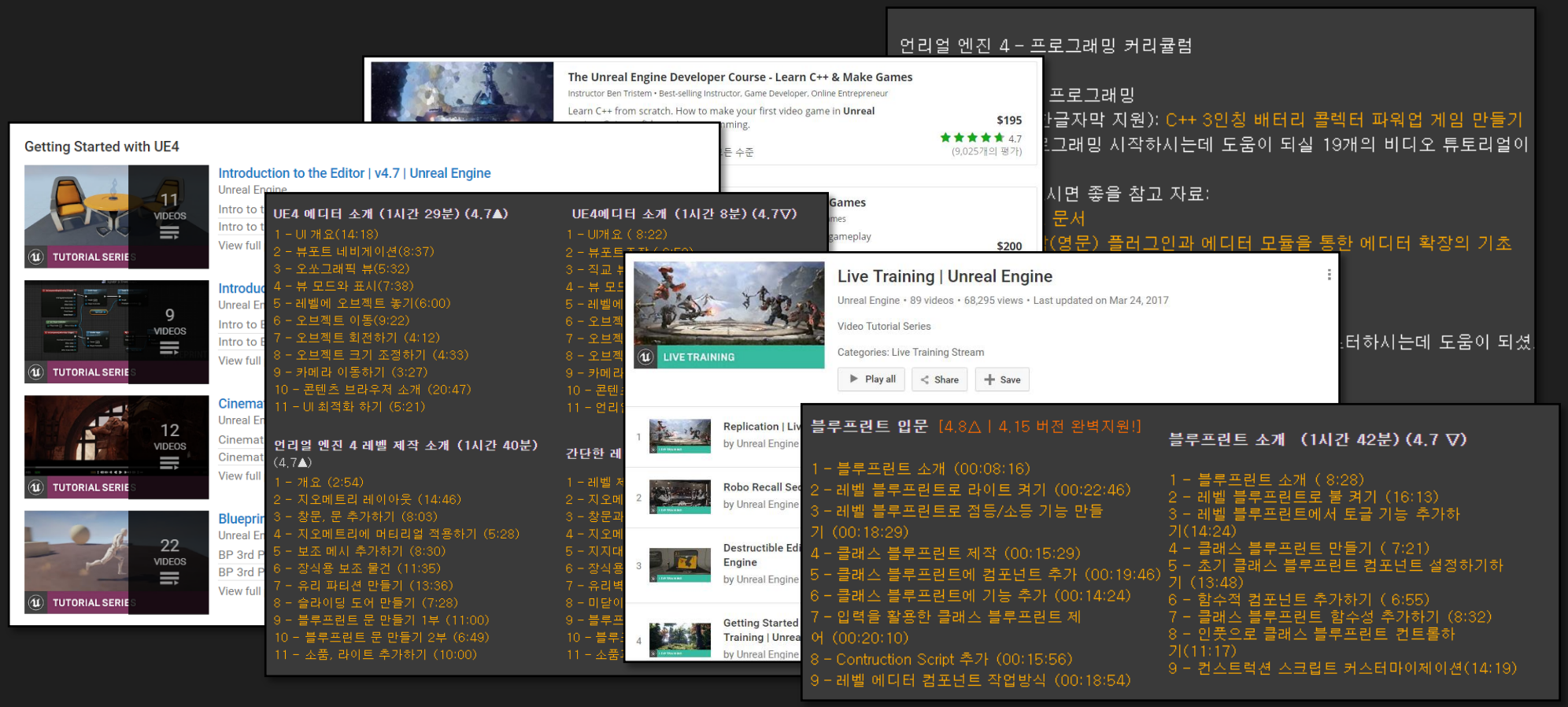
UNREAL ENGINE

언리얼 튜토리얼만 쌓여가는 유니티 개발자를 위한 조언

튜토리얼이 이렇게 많은데 무엇이 문제인가?



본 세션은 심도 깊은 최신 기술에 대한 내용은 없습니다.

(언리얼 처음 다루는 분들이 듣기 좋습니다.)

언리얼 엔진을 다루기 전에 제품의 구조와 철학을

이해하자는 관점에서 발표하는 세션입니다.

게다가 발표 내용은 모두 발표자의 주관적인 의견입니다.

나의 언리얼 튜토리얼은 왜 쌓여만 가는가?

- 바빠서 살펴볼 시간이 없다. 늘어만 가는 내 북마크 리스트.
- 사실 튜토리얼을 따라해봤는데, 다 했는데, 이제 뭐가 뭔지 잘 모르겠다.

나는 과연 잘 하고 있는가?

혼자서 잘 할 수 있을까?

왜 튜토리얼만 쌓여가는지 근본적인 이야기를 해봅시다.

대부분 튜토리얼 내용은 컨텐츠 제작에 집중되어 있습니다.

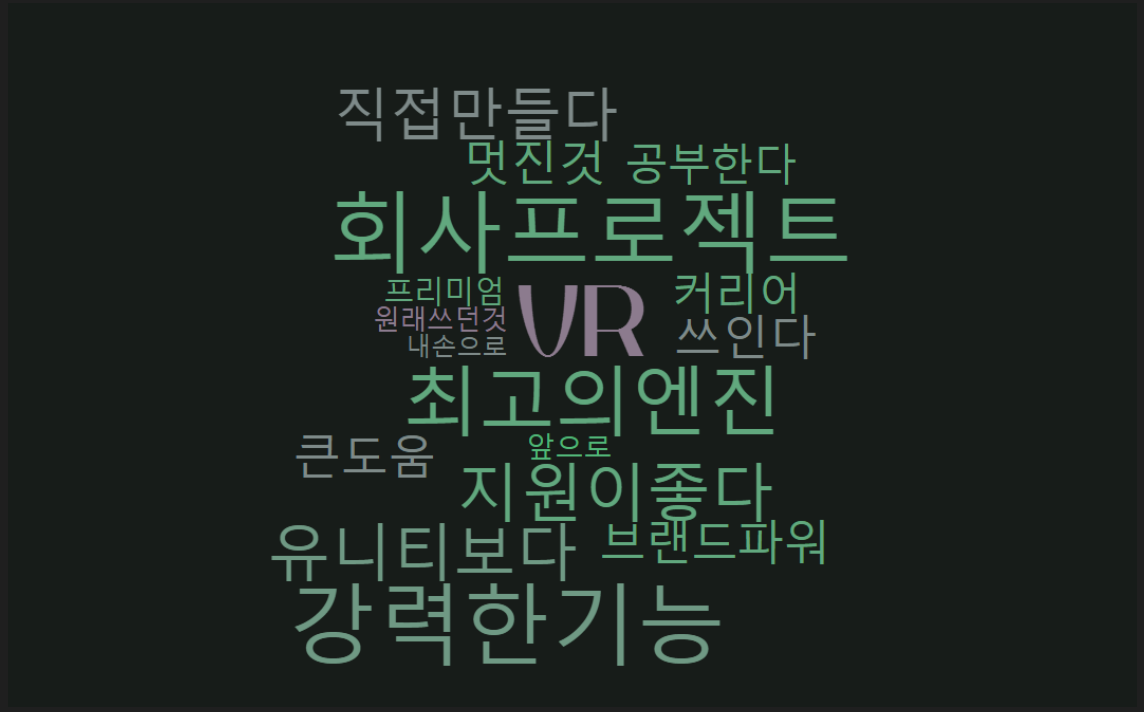
언리얼 엔진의 원리나 구조에 대해서 공식 홈페이지에

문서가 있지만 너무 방대하고 어렵습니다.

언리얼 엔진은 (초) 대형 게임 제작이 커버 가능한 엔진입니다. 왜 언리얼 엔진을 쓰는지 먼저 생각해봅시다.

나는 왜 언리얼 엔진을 쓰는가?

페이스북 언리얼 엔진 4 개발자 커뮤니티 설문



1. 내 손으로 직접 멋진 것을 만들고 싶어서
2. 앞으로도 계속 쓰일 엔진
3. 좀 더 공부를 많이 하고 싶어서

나는 왜 언리얼 엔진을 쓰는가?

게임 엔진을 더욱 깊숙히 알고 싶은 자유

프로그래머 직군을 위한 조언

언리얼 엔진의 철학 #1

모든 직군이 프로그래밍을 몰라도 컨텐츠를 제작할 수 있도록 설계한다.

Modern Game Engine Architecture

최신 게임 엔진은 세 개의 계층으로 구분할 수 있다.

Modern Game Engine Architecture - Runtime

- 단단하고 빠른 게임 엔진의 심장
- 일반적으로 개발 언어는 C++를 많이 사용
- 사소한 오류에도 결과는 크리티컬하기 때문에 코딩에 엄격한 룰을 적용
- 컨텐츠 제작자에게 노출하지 않는다.

Modern Game Engine Architecture – Developer

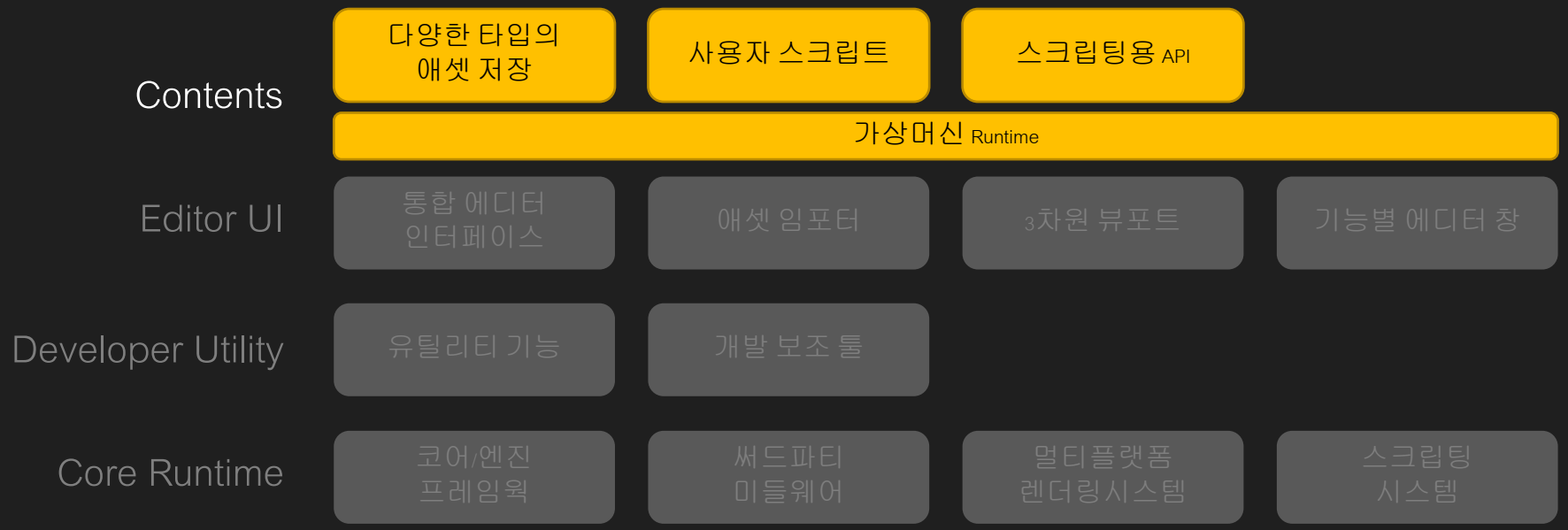
- 컨텐츠 개발에 보조로 사용되는 개발 도구.
- 방대한 엔진 기능 관리를 위해 다양한 도구의 개발이 필요.
- 사용 프레임웍에 따라 구현. C++ 을 그대로 쓰거나 C#로 변경하거나.
- 컨텐츠 제작자에게 노출하지 않음.

Modern Game Engine Architecture – Editor

- 게임 컨텐츠 개발을 위한 에디터 UI의 제공
- 다양한 외부 파일을 엔진 파일로 변환시켜주는 변환 기능
- 3차원 공간을 다룰 수 있는 뷰포트 제공
- 컨텐츠 제작자가 직접 활용하고 결과물을 디스크에 저장

Modern Game Engine Architecture – Contents 제작

- 컨텐츠를 개발하고 결과물을 디스크에 저장
- 생산성 : 빠른 컴파일, 스크립트와 전문 개발 도구 연동
- 안정성 : 실행 영역의 분리 : 안전한 메모리관리



Modern Game Engine Architecture – C# 스크립팅

- 이미 산업적으로 널리 쓰이는 완성도 높은 언어
- 업계 최강의 개발도구 Visual Studio
- MS 표준 Runtime에서 안전하게 실행
- 전문가의 눈높이를 맞출 만큼 탄탄한 설계가 가능
- Native 라이브러리와 연동(Interop) 및 확장 가능한 설계
- Assembly Component의 공유 (GAC) – 코드 재사용, 컴파일 타임을 단축
- Native 못지 않는 빠른 성능 (JIT / AOT 컴파일)

초급/고급 프로그래머 모두를 만족시키는 프로그래밍 환경

Modern Game Engine Architecture – 블루프린트

- 에픽 게임스가 만든 시각적 프로그래밍 언어 (Visual Programming Language)
- 자체 개발 도구를 제공 (Blueprint Editor)
- 자체 제작한 Runtime에서 안전하게 실행
- 빠른 컴파일 속도
- 가독성 높은 디자인

프로그래밍이라는 진입 장벽이 없어 모든 직군이 사용 가능

Modern Game Engine Architecture – 블루프린트

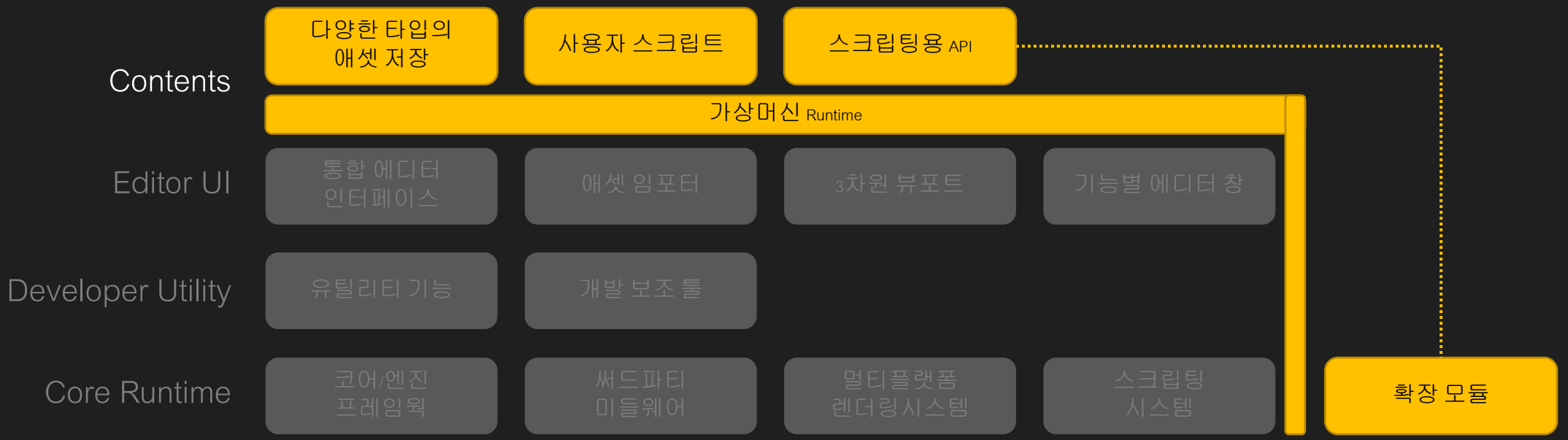
- 전문가의 눈높이를 맞출 만큼 설계 구조가 탄탄하지 못함.
- 블루프린트 시스템 밖으로는 확장할 수 없다.
- 컴파일은 빠르나 실행 속도는 느리다.(그렇다고 게임을 못 만들 성능은 아님)
- 복잡도가 증가할 수록 사용하는 리소스가 늘어나고 가독성이 떨어짐.

블루프린트는 만능이 아니다.

대신 프로그래머를 위해 C++을 제공한다.

Modern Game Engine Architecture – Unity

블랙박스 존 : 외부 개발자가 볼 수 없도록 철저하게 격리, 유니티에서 관리 배포



Modern Game Engine Architecture – Unreal

개발자가 모든 엔진 소스를 활용해 추가 개발할 자유를 부여

나는 왜 언리얼 엔진을 쓰는가?

게임 엔진 코드를 마음껏 쓸 수 있는 자유 (SET IT FREE)

그런데..

자유에는 (굉장한) 대가가 따릅니다.

프로그래머가 언리얼 엔진을 사용하기 위해서는
앞으로 어떤 대가를 치룰지 마음의 준비를 해야 합니다. ㅋ

저는 마음의 준비를 안해 고생했지만,
이 세션을 통해 여러분들은 안 그랬으면 합니다.

Modern Game Engine Architecture – Unity

격리된 환경의 장점 : 프로그래머는 하나만 신경쓰면 된다.

Modern Game Engine Architecture – Unreal

무한한 자유를 얻기 위해서는 C++을 사용해야 한다.

소스 코드를 얻게 되면 생기는 문제점

- 소스 코드 양이 너무 어마어마하다.
- 개발 할 때 이를 함께 가지고 가야 한다.
- 부수적으로 따르는 부작용
  - 느린 컴파일 속도
  - 느린 인텔리센스 기능
  - 엄청난 용량의 결과물

이런 문제들은 빠른 C++ 언어의 자체적인 문제

에픽 게임스 C++ 환경 개선 캠페인 #1

소스 코드의 모듈화

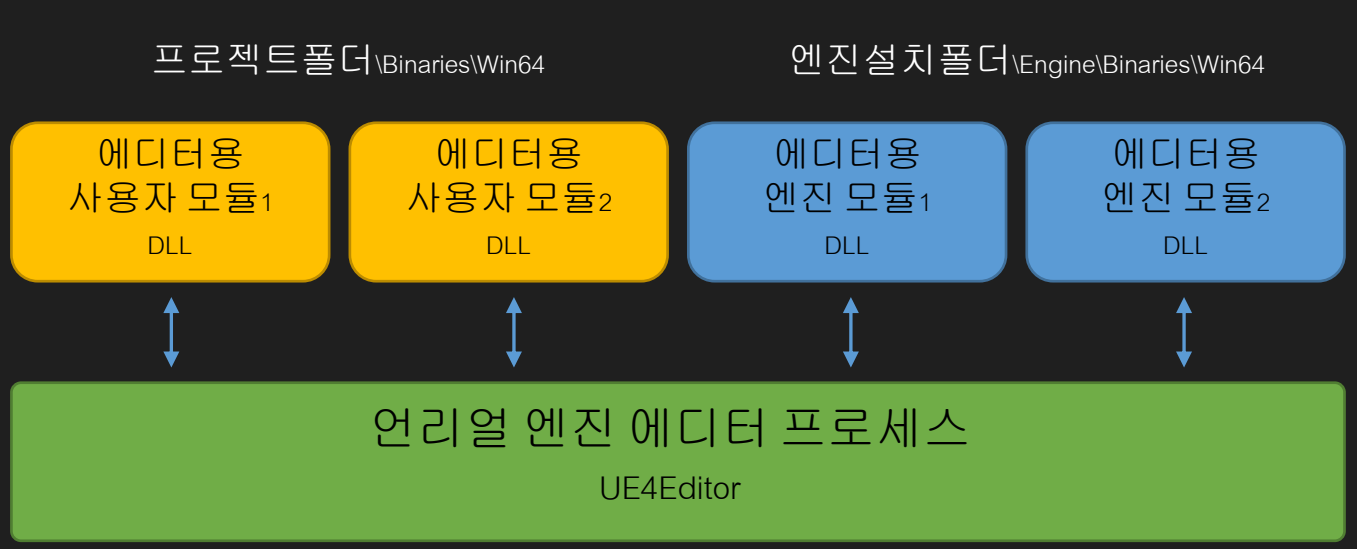
에디터에서 사용할 컨텐츠를 처음부터 컴파일 한다면?

- 에디터에서 사용할 Runtime 라이브러리를 매번 컴파일한다면?
- 생각만 해도 개발하기 싫어진다.
- 모듈이라는 엔진이 제공하는 기능을 소분화시키자.
- 컨텐츠 제작시에는 모듈 별로 DLL로 분리하고
  에디터 실행 중에 붙였다 떼었다 하게 만들자. 이것이 HotReload
- 실제 게임 패키징에는 한꺼번에 모아 실행 파일로 생성

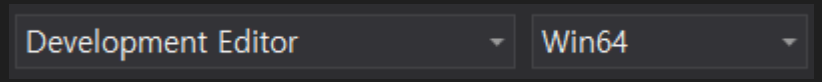
왜 소스코드를 모듈 단위로 나누는지 이해해봅시다.

언리얼 엔진 C++ 프로그래밍의 시작은 모듈

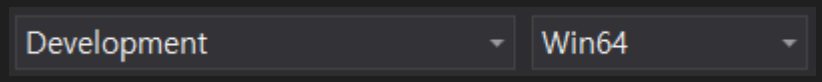
쪼개놓은 후에 상황에 따라 헤쳐 모여 가능하도록 설계



에디터 빌드 타입 (Modular)



게임 빌드 타입 (Monolithic)



자동으로 완성해주는 마법의 빌드 툴 - Unreal Build Tool

- 멀티플랫폼 빌드 툴
- Private 영역과 Public 영역을 구분해 빌드 목록 생성
- 의존성을 파악하고 빌드 순서 지정
- Modular (에디터) / Monolithic (게임) 방식의 빌드 지원
- 프리컴파일드헤더 컴파일 지원
- Static Library 지원 (플러그인)
- 유니티 빌드 방식 지원 (대용량 컴파일을 묶어서 빠르게)

프로젝트의 Target.cs와 모듈의 Build.cs 를 통해 설정 가능

모듈을 직접 만들어보고 싶다면?

네이버 언리얼 엔진 공식 카페 커뮤니티

http://cafe.naver.com/unrealenginekr/

UE 튜토리얼 및 팁 섹션 프로그래밍 게시판

[C++기초 2] 언리얼 빌드 시스템

[C++기초 3] 모듈의 제작

http://cafe.naver.com/unrealenginekr/13881

http://cafe.naver.com/unrealenginekr/13882

에픽 게임스 C++ 생산성 개선 캠페인 #2

언리얼 오브젝트 프레임웍

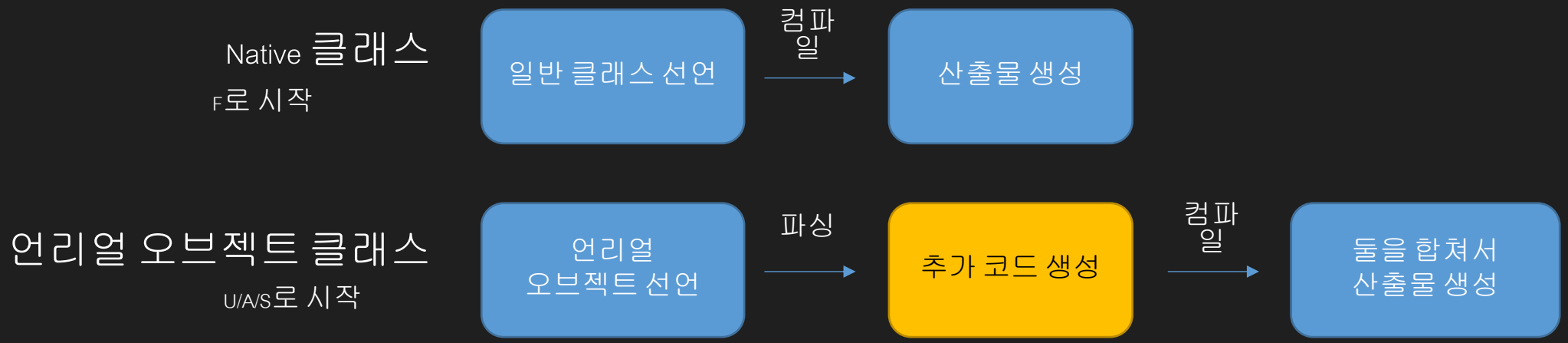
C++ 언어의 생산성을 높이자

- 객체의 초기 설정 값을 손쉽게 관리
- 런타임에서 클래스와 인스턴스 정보를 검색
- 객체의 저장과 로딩을 간편하게
- 메모리 관리를 편하게
- 함수를 묶어서 한번에 호출
- 모든 플랫폼에 안정적으로 동작하는 API

C#과 같은 C++ 프레임웍의 제공

관리받는 코드와 Native 코드의 혼합

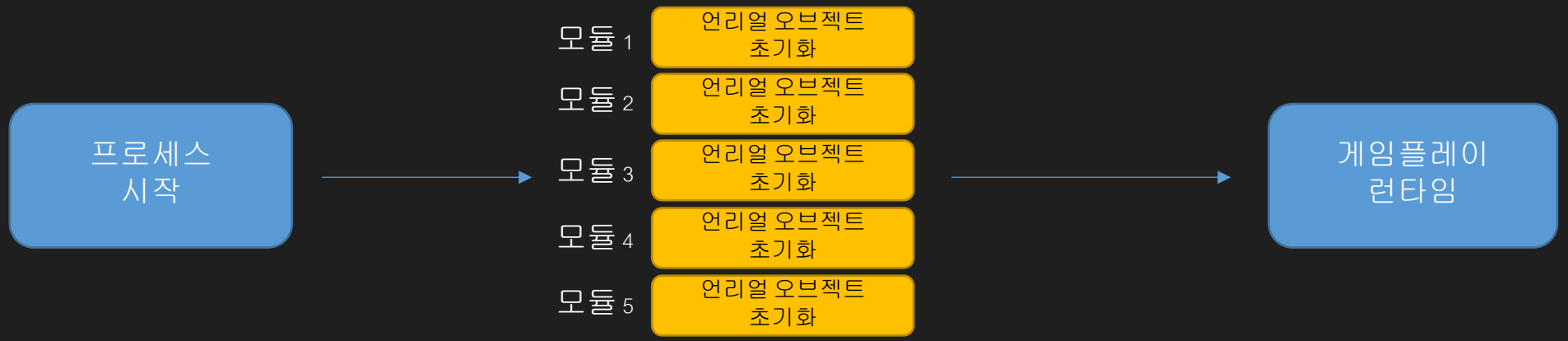
생산성을 높이기 위해 관리받는 C++ 클래스 프레임웍을 구축



소스코드에 특별한 매크로가 있으면 파싱을 시도하고 규칙에 맞으면 엔진에서 관리

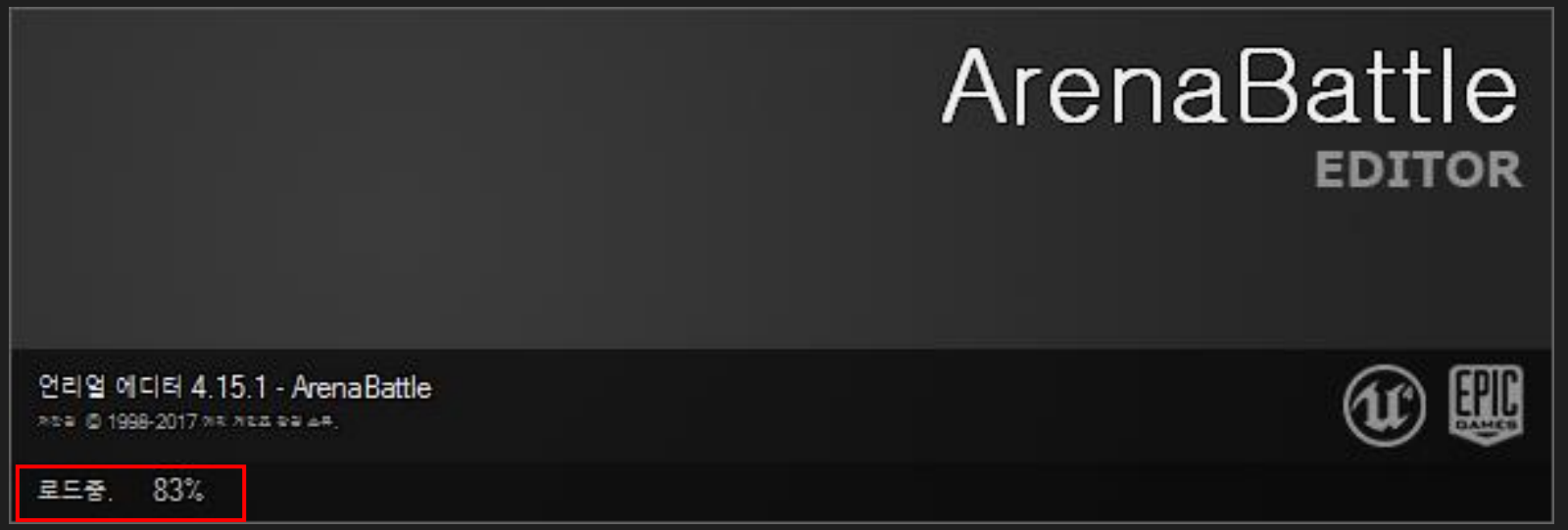
관리받는 코드와 Native 코드의 혼합 – 모듈 초기화

하지만, 편리한 기능으로 인해 고려해야 할 것이 늘어난다.



언리얼 엔진의 실행의 처음에는 항상 모듈 단위로, 언리얼 오브젝트 초기화 과정이 들어간다.

관리받는 코드와 Native 코드의 혼합 – 모듈 초기화



DLL 단위로 모듈 내 모든 언리얼 오브젝트 초기화가 완성되면 %가 올라간다.

관리받는 코드와 Native 코드의 혼합 – UClass

하나의 언리얼 오브젝트에는 상응하는 UClass가 존재한다.

초기화 단계에서 모듈 별로 자신이 속한 언리얼 오브젝트의 UClass DB를 구축한다.

언리얼 엔진에서는 "/Script/모듈이름.클래스이름" 형식으로 고유 주소를 부여한다.

관리받는 코드와 Native 코드의 혼합 – CDO

CDO – Class Default Object (클래스 기본 객체)

언리얼 오브젝트는 CDO를 복제해 인스턴스를 생성하도록 설계되어 있다.

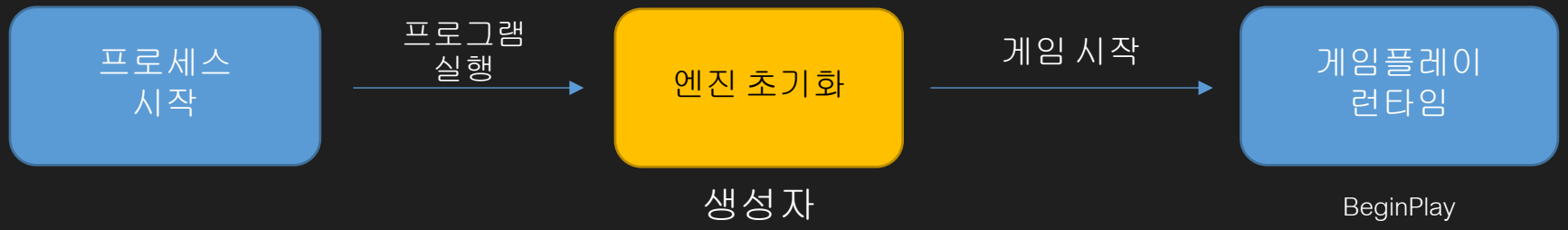
관리받는 코드와 Native 코드의 혼합 – 모듈 초기화

초기화 단계에서 복제하기 쉽게 미리 CDO를 만들어준다.

초기화가 끝나면 대략 위와 같은 형태로 메모리가 완성된다.

관리받는 코드와 Native 코드의 혼합 – CDO

CDO는 생성자 코드를 실행하면서 완성된다.
즉, 생성자 코드는 엔진 초기화 단계에서 실행된다.



생성자 코드에서는 게임에 대한 내용을 전혀 알 수가 없다!

언리얼 오브젝트에 대해 궁금하다면?

네이버 언리얼 엔진 공식 카페 커뮤니티

http://cafe.naver.com/unrealenginekr/

UE 튜토리얼 및 팁 섹션 프로그래밍 게시판

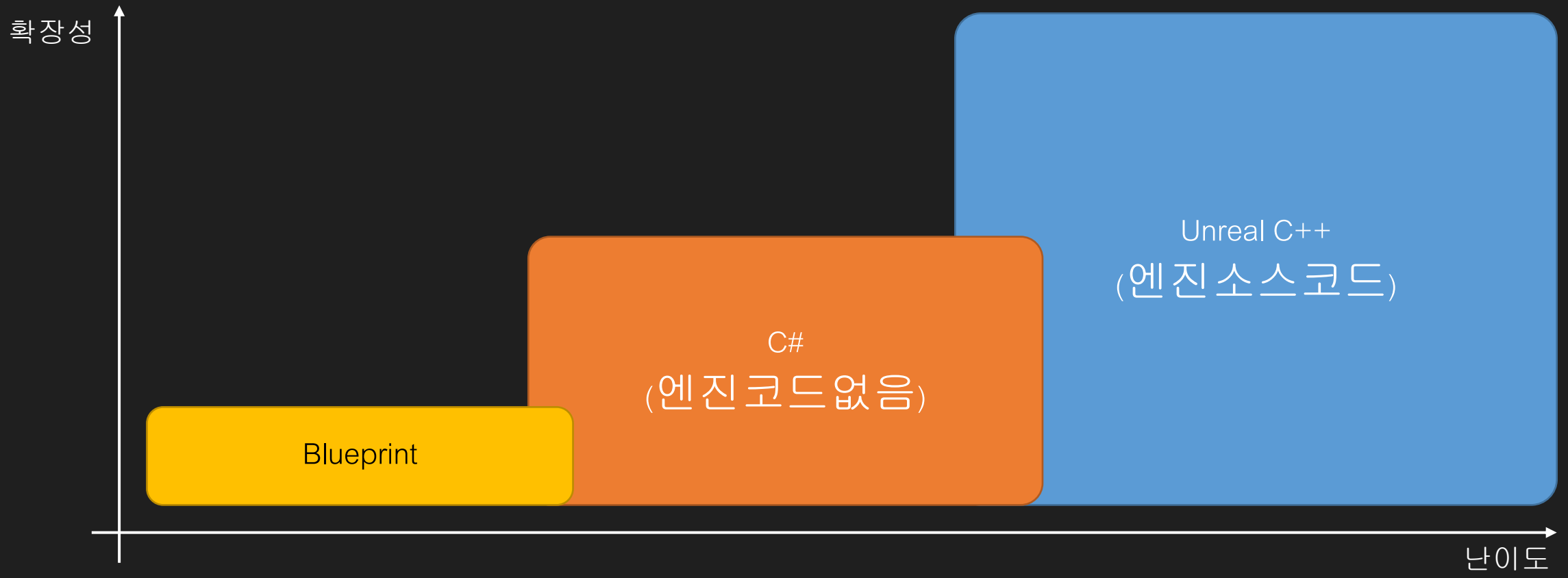
[C++기초 5] Class Default Object
[C++기초 6] UClass 와 리플렉션

http://cafe.naver.com/unrealenginekr/13885

http://cafe.naver.com/unrealenginekr/13921

게시판에도 그 외, 언리얼 엔진 프레임웍의 기초를 정리한 튜토리얼을 올렸습니다.

각 언어 별, 난이도/확장성 분석



이런 내용에 대한 이해 없이
바로 튜토리얼만 보고 컨텐츠 제작에 들어간다면
응용이 안되 튜토리얼이 쌓일 수 밖에 없습니다.

유니티 프로그래머를 위한 조언

- 빠르게 프로토타입 컨텐츠를 만들려면 (방식이) 맘에 안들어도 블루프린트가 맘 편하다.
- 블루프린트는 확장성에 한계가 있으니, 어디까지 할 수 있는지 미리 조사하자.
- 제대로 엔진을 다루고 싶다면 C++로 가야 한다.
- 자유에는 (엄청난) 대가가 따른다! (*컴파일 타임, 인텔리센스 ㅜㅜ*)
- C# 개발환경보다 (많이) 불편할지라도 익숙해지려고 노력하자.
- 하나씩 차근차근 따라오면 세계 최고 엔진이 내 것이 된다.

블루프린트로 꽤 많이 구현할 수 있고, C++을 한다면 시작하기 전 기간을 넉넉히 잡고

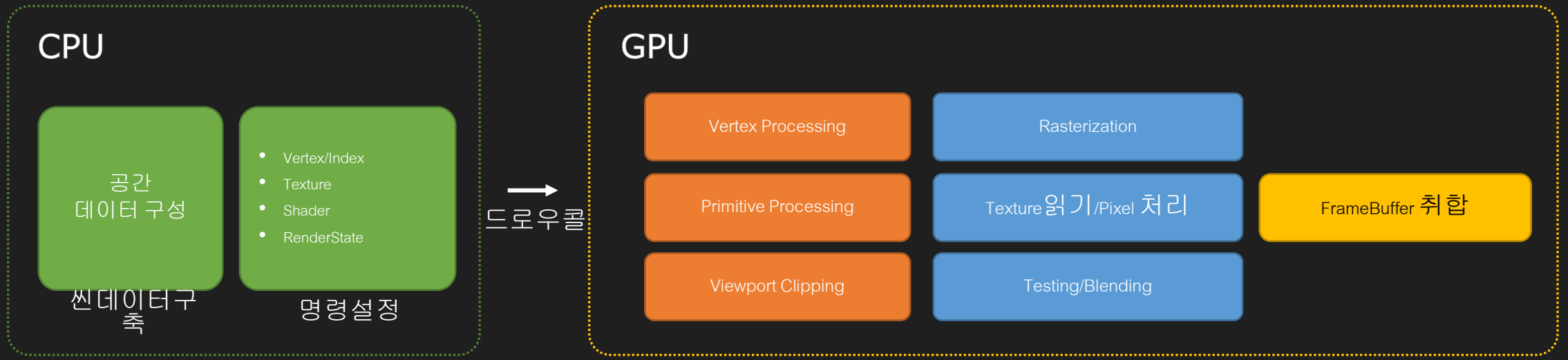
언리얼 빌드 시스템 > 언리얼 오브젝트 순서대로 기초를 탄탄하게 익히는 것을 권장합니다.

아티스트 직군을 위한 조언

언리얼 엔진의 철학 #2

게임을 제작한 경험을 바탕으로, 최대한 완성된 프레임웍을 만들어 제공한다.

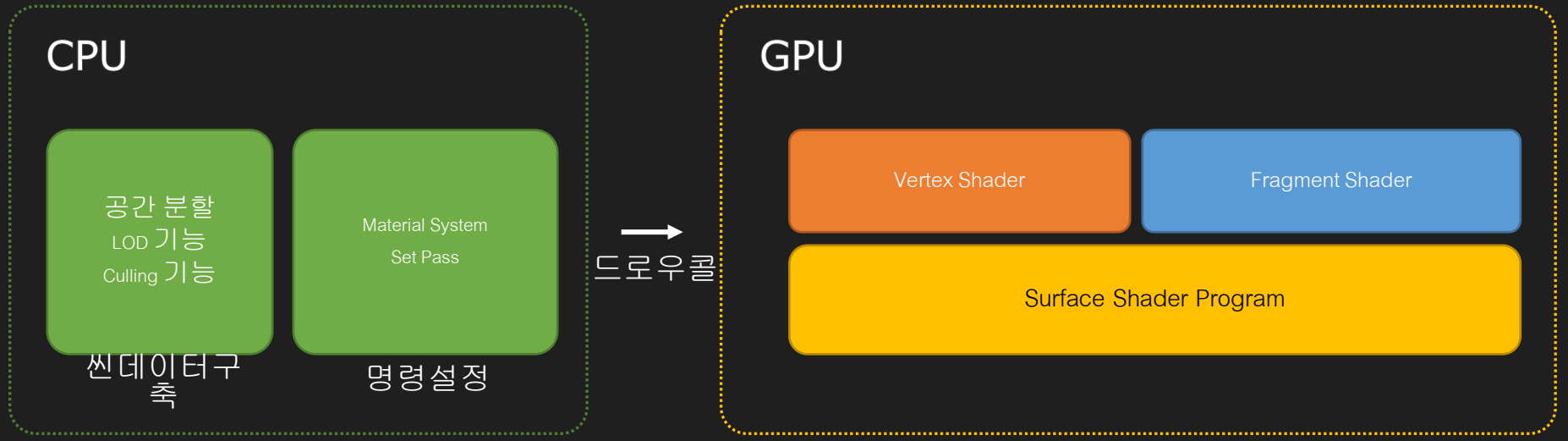
Rendering Pipeline 개요



주어진 파이프라인을 활용하는 방법은 굉장히 다양하다.
(자체 엔진이 필요한 이유)

Rendering Pipeline 개요 – Unity

게임엔진은 생산성을 위해 작업자들이 일관성있게 작품을 제작하도록
일정한 규격을 제공해준다.



모델이 심플할수록 생산성은 올라가지만
파이프라인을 다양하게 활용하는 유연성이 떨어진다.

생산성을 위해 어느 정도까지 모델을 설계할 것인가?

= 어느 정도까지 유연성을 포기할 것인가?

Rendering Pipeline 특징 – Unity

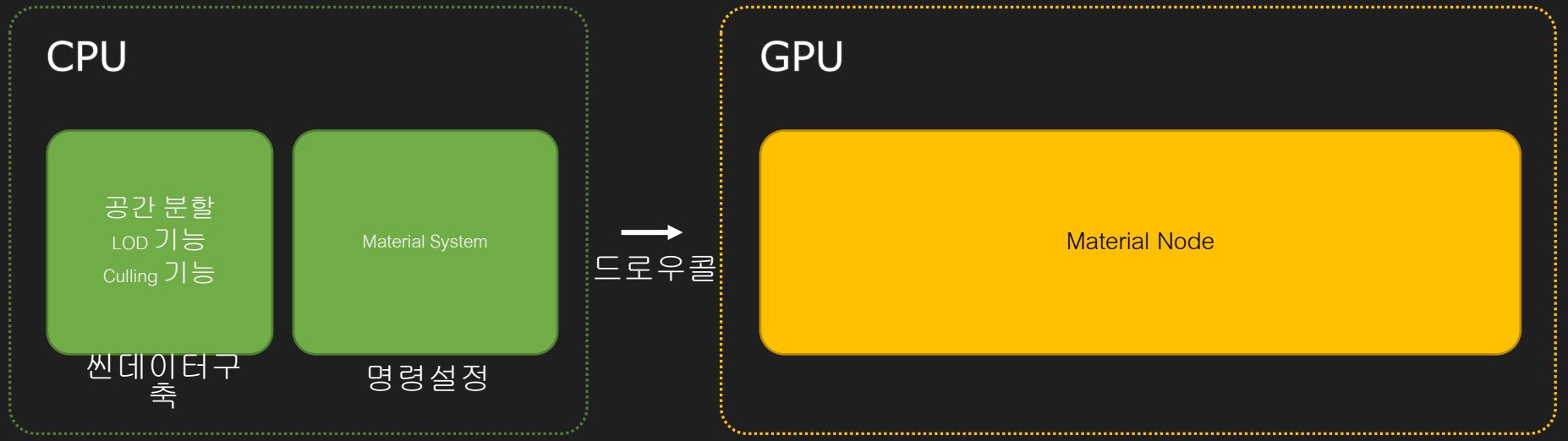
- 유니티는 낮은 수준의 모델을 제공한다. = 엔진의 유연성이 높다.
- 아티스트가 편하게 작업할 수 있도록 머티리얼 시스템이 설계되어 있다.
- 셰이더 프로그래밍도 어느 정도 파고들 수 있다.
- 셰이더 프로그래밍도 두 단계로 나눈다. (서피스 셰이더 / 일반)

유니티 렌더링 모델의 추상도 :(중하)

유니티 렌더링 기능의 유연성 :(중상)

Rendering Pipeline 개요 – Unreal Engine

언리얼 엔진은 유연성보다 최대한 기능과 완성도를 높인 프레임웍을 제작해 모델을 제공



유연함에 있어서 언리얼보다 유니티가 더 큰 장점을 가진다.

Rendering Pipeline 특징 – Unreal Engine

- 언리얼은 높은 수준의 모델을 제공한다. = 엔진의 유연성이 떨어진다.
- 머티리얼 시스템에서 모든 것이 끝난다. (이 이상은 전문 프로그래머에게.)
- 옵션 체크만으로 하이엔드와 모바일 기능이 (자동으로) 처리된다.

언리얼 렌더링 모델의 추상도 : (상)

언리얼 렌더링 기능의 유연성 : (하)

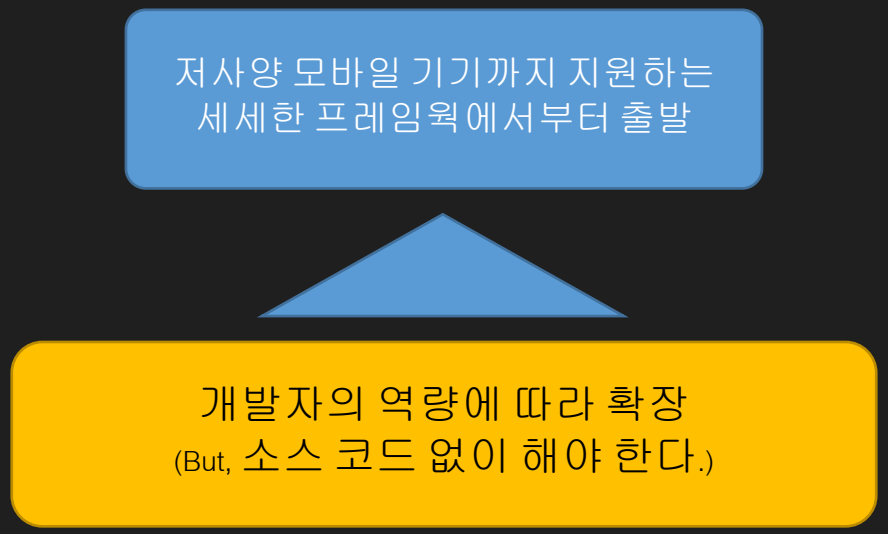
언리얼 엔진의 철학 #3

프레임웍의 설계는 (초) 대형 게임을 제작할 수 있는 스케일로 기획한다.

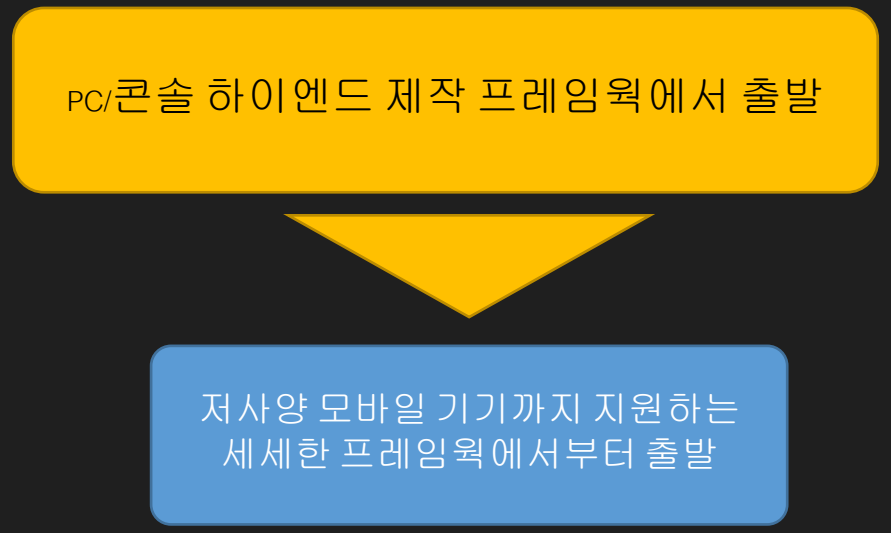
언리얼 엔진의 철학 정리

1. 모든 직군이 프로그래밍을 몰라도 컨텐츠를 제작할 수 있도록 설계한다.
2. 게임을 제작한 경험을 바탕으로, 최대한 완성된 프레임웍을 만들어 제공한다.
3. 프레임웍의 설계는 (초) 대형 게임을 제작할 수 있는 스케일로 기획한다.

두 엔진간 렌더링 시스템 철학의 비교



유니티는 최소 기능에서부터
수동으로 확장하는 방식



언리얼은 최대 기능에서부터
호환성을 유지해 자동으로 줄여주는 방식

두 엔진간 렌더링 시스템 철학의 비교

Unity

모바일 — Forward Rendering

PC/콘솔 — Deferred Rendering

각 파이프라인 마다 다르게 작업 환경을 설계한다.

각 렌더링 파이프라인별로 독립적으로 기능을 개발한다.

편리하게 셰이더 프로그래밍을 다룰 수 있다.
이로 인한 다양한 효과가 가능하다.

Unreal

PC/콘솔 — Deferred Rendering (PBR only)

모바일 — Forward Rendering (PBR only)

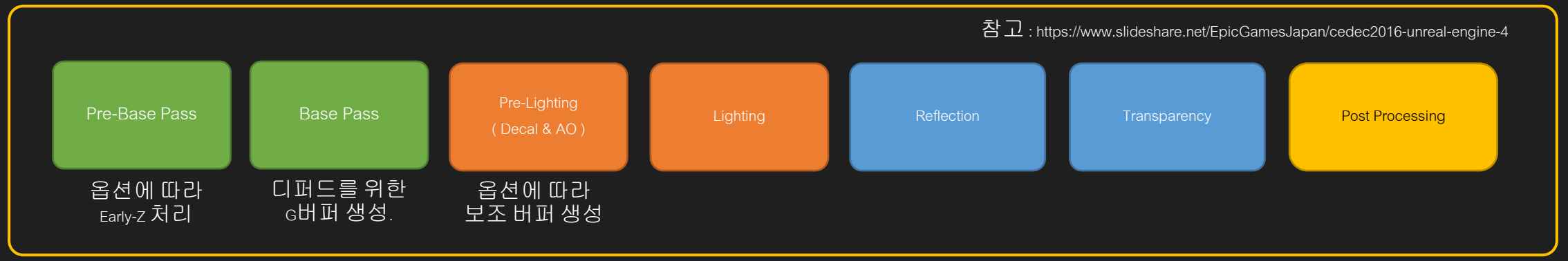
자동으로 변환해준다.

모든 작업 환경이 디퍼드 렌더링 스펙으로 맞춰져 있다.

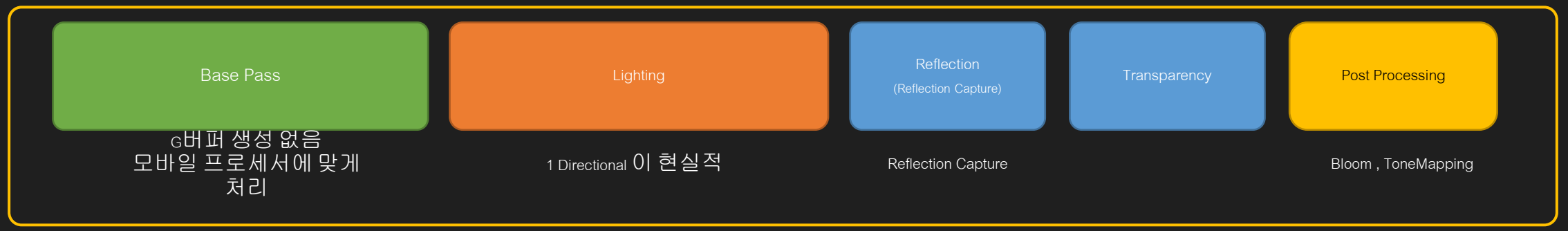
.usf로 모델을 확장할 수 있지만, 전문 프로그래머 영역

언리얼 엔진의 렌더링 파이프라인

참고 : https://www.slideshare.net/EpicGamesJapan/cedec2016-unreal-engine-4



Deferred Rendering Pipeline (PC / Console)



Forward Rendering Pipeline (Mobile)

유니티 아티스트를 위한 조언

- 언리얼 엔진에서 제대로 셰이더 프로그래밍을 하려면 엔진 소스를 이해해야하고 그러면 다시 앞단의 프로그래밍을...
- 언리얼 엔진이 만든 틀 안에서 움직일 수 밖에 없습니다.
- 구조상 어쩔 수 없이 긴 셰이더 컴파일/쿠킹 타임을 감내해야 합니다.
- 에디터와 기기에 나오는 룩이 다른 건 어쩔 수 없습니다.
- PBR에 이미 많은 셰이더 기능이 들어가 있습니다. 텍스쳐로 최대한 역량을..

전통적 방식의 멀티패스 렌더링이 안 된다고 너무 당황하지 마십시오.

떨어지는 유연성은 확장성, 생산성과 성능으로 돌아옵니다.(포스트프로세싱 짱!)