Microsoft

tech·days Korea 2014
개발자 컨퍼런스

The Future of C# and .NET Framework

김명신
Principle Technical Evangelist
DX, Microsoft

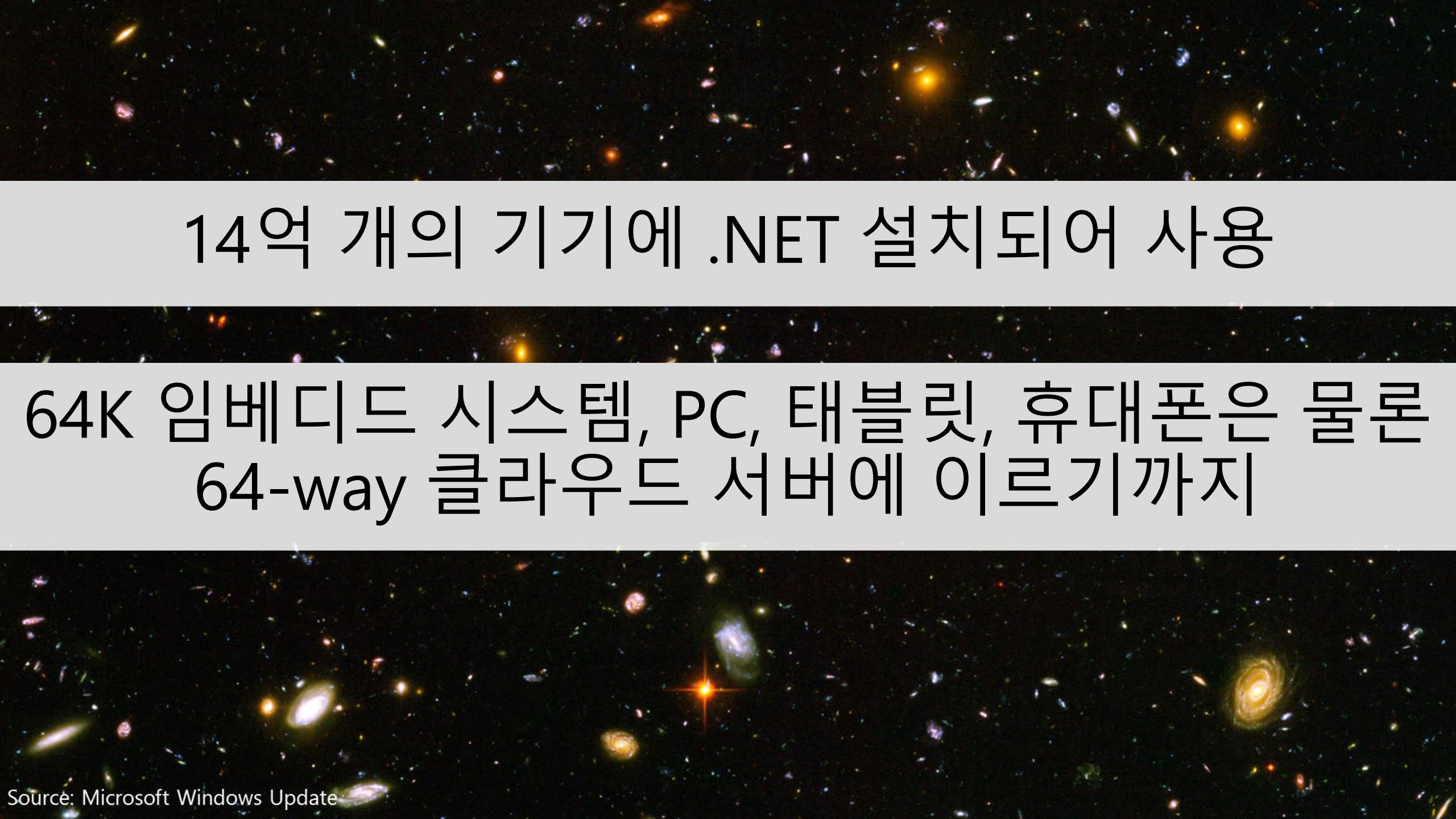
@himskim

C#은 Java의 카피캣에 지나지 않는다.

.NET은 사용하지 않는 플랫폼이다.

C#와 .NET은 더 이상 발전이 없다.

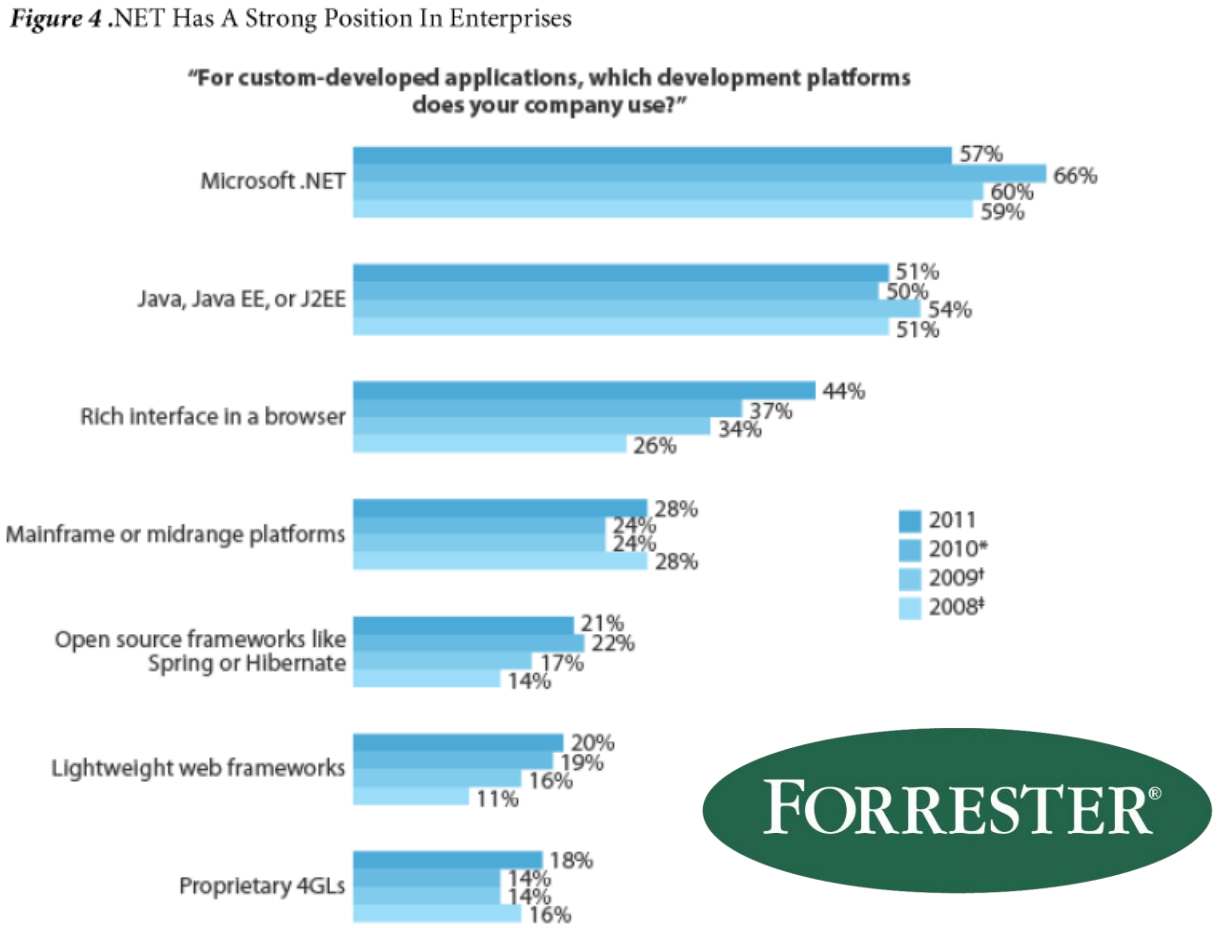
Reality Check


14억 개의 기기에 .NET 설치되어 사용
64K 임베디드 시스템, PC, 태블릿, 휴대폰은 물론
64-way 클라우드 서버에 이르기까지
Source: Microsoft Windows Update

비즈니스 응용 프로그램의 .NET 사용현황

57%의 기업이 자사의 응용 프로그램을 개발하기 위해서 .NET을 사용

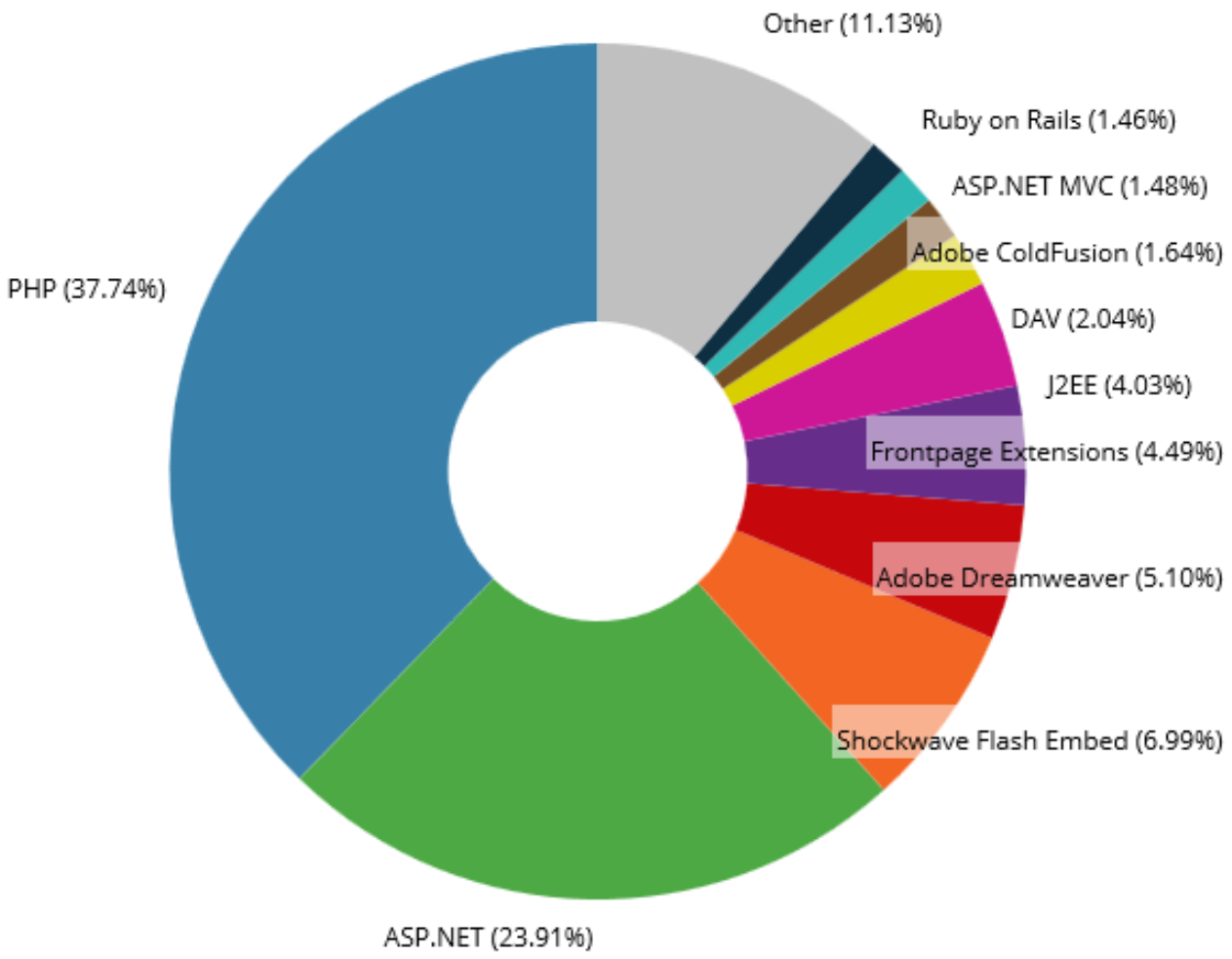
(Forrester 2012 report)



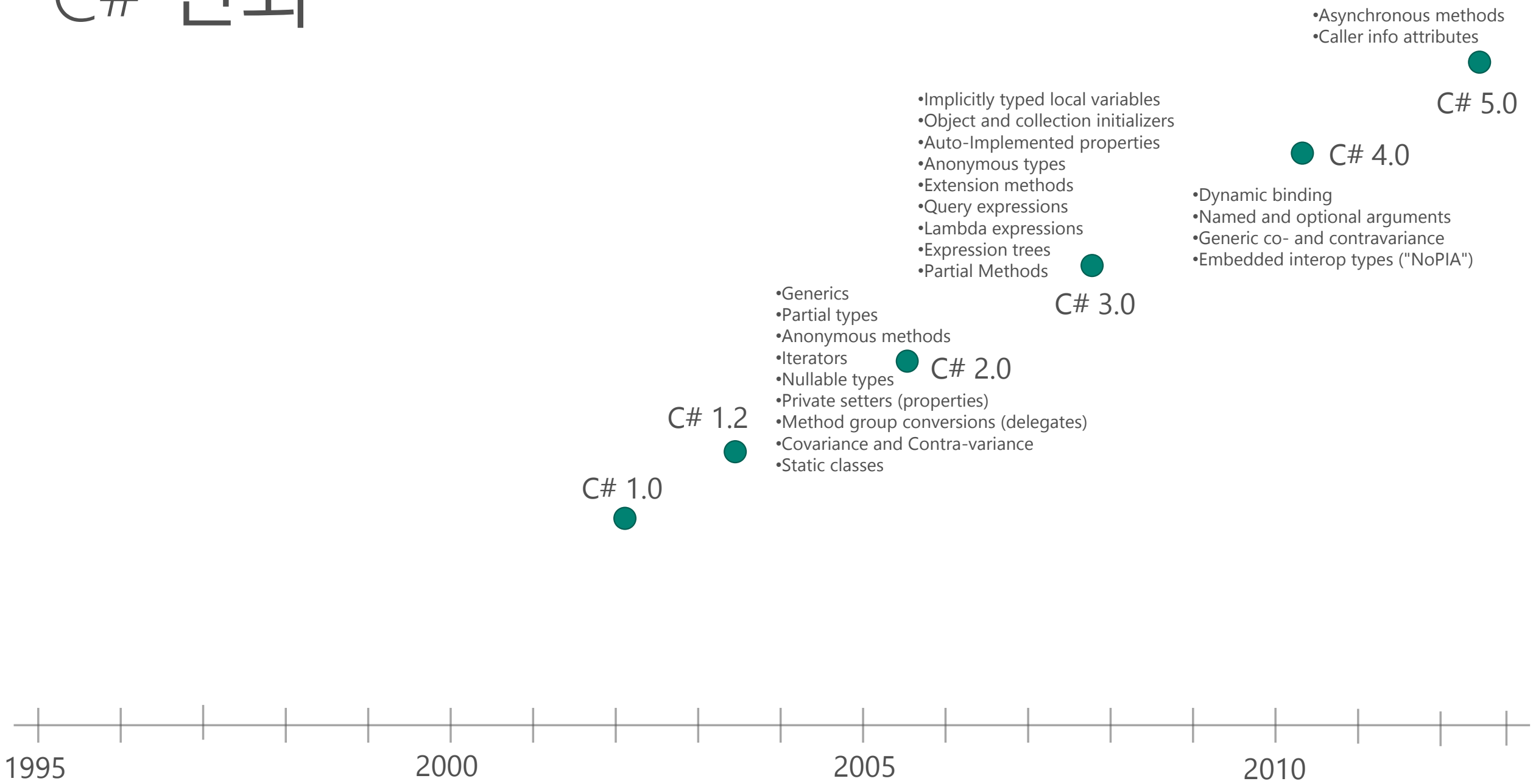
웹 사이트 개발에 사용하는 프레임워크

상위 100,000개의 웹사이트 중 25.38%가 ASP.NET 혹은 ASP.NET MVC를 사용

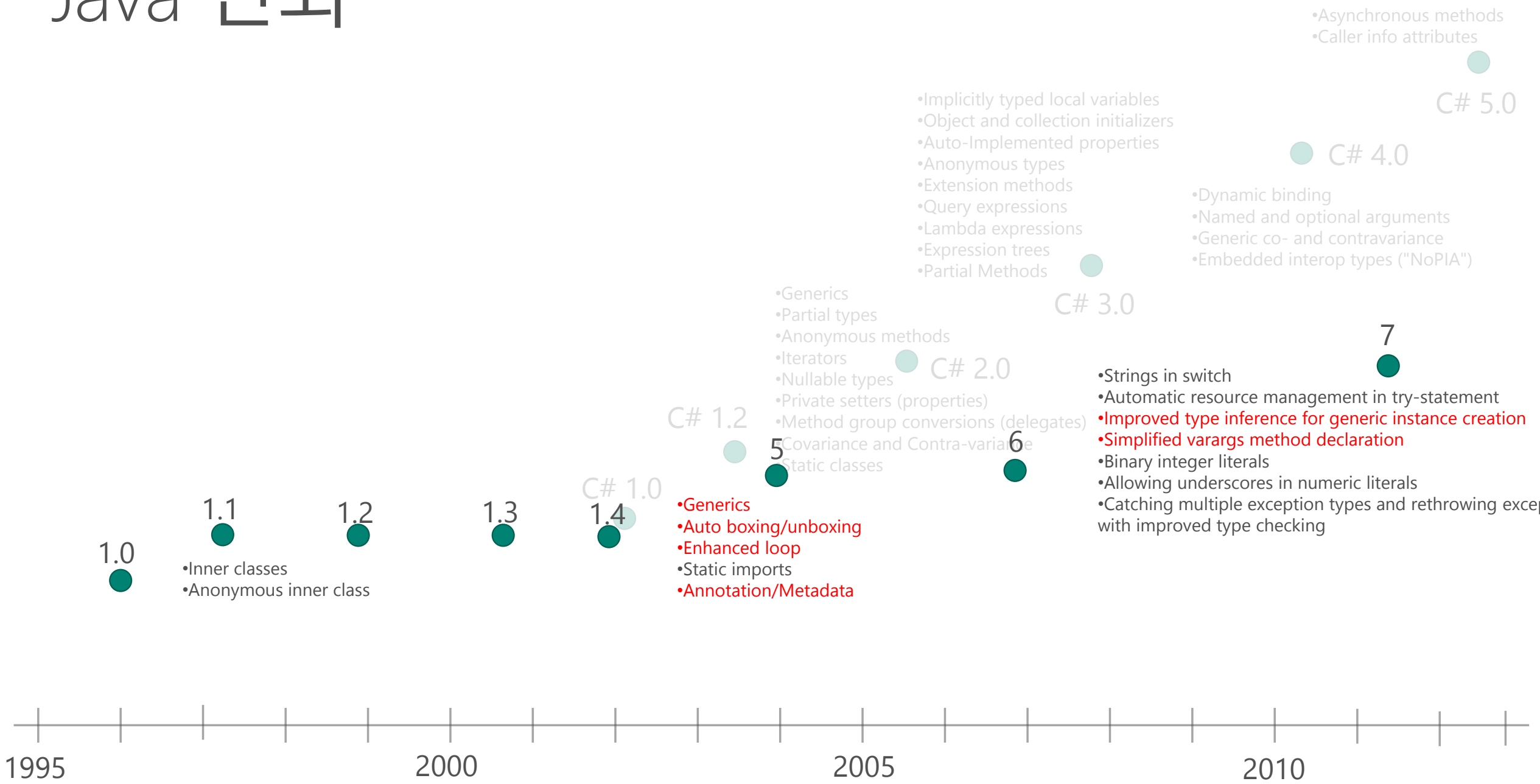
http://trends.builtwith.com/framework



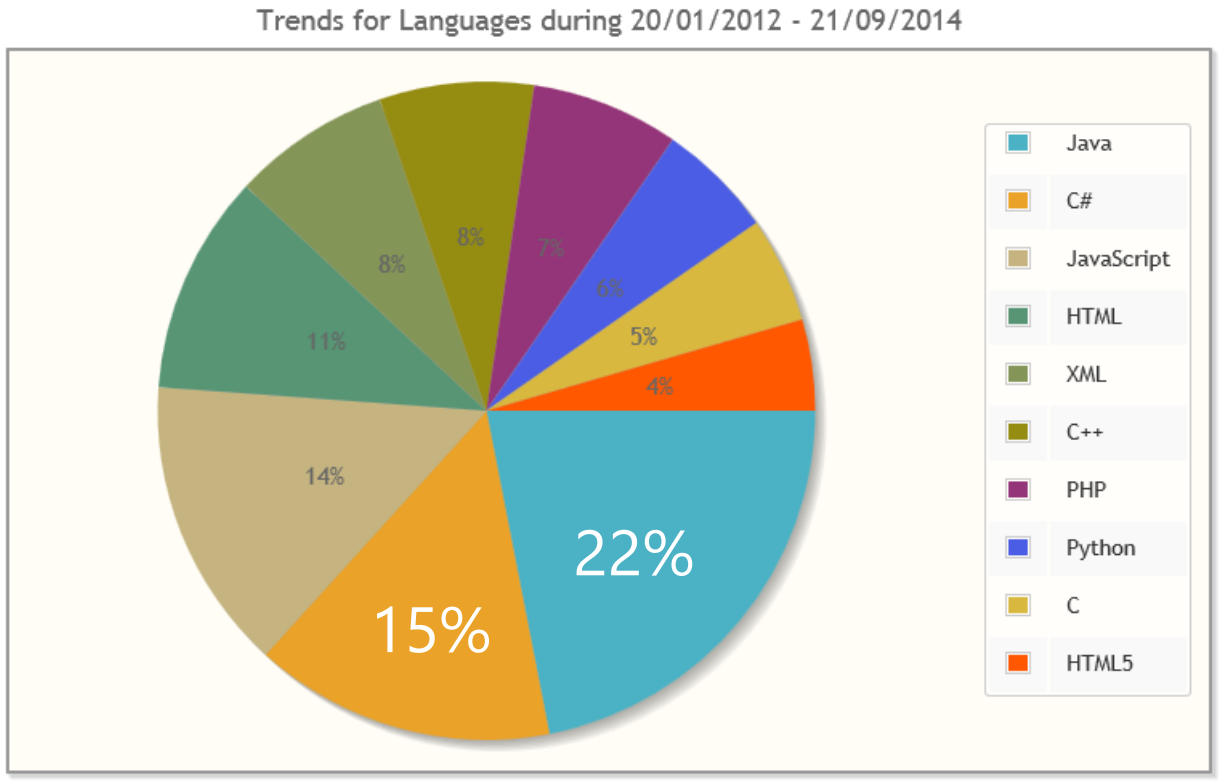
C# 진화

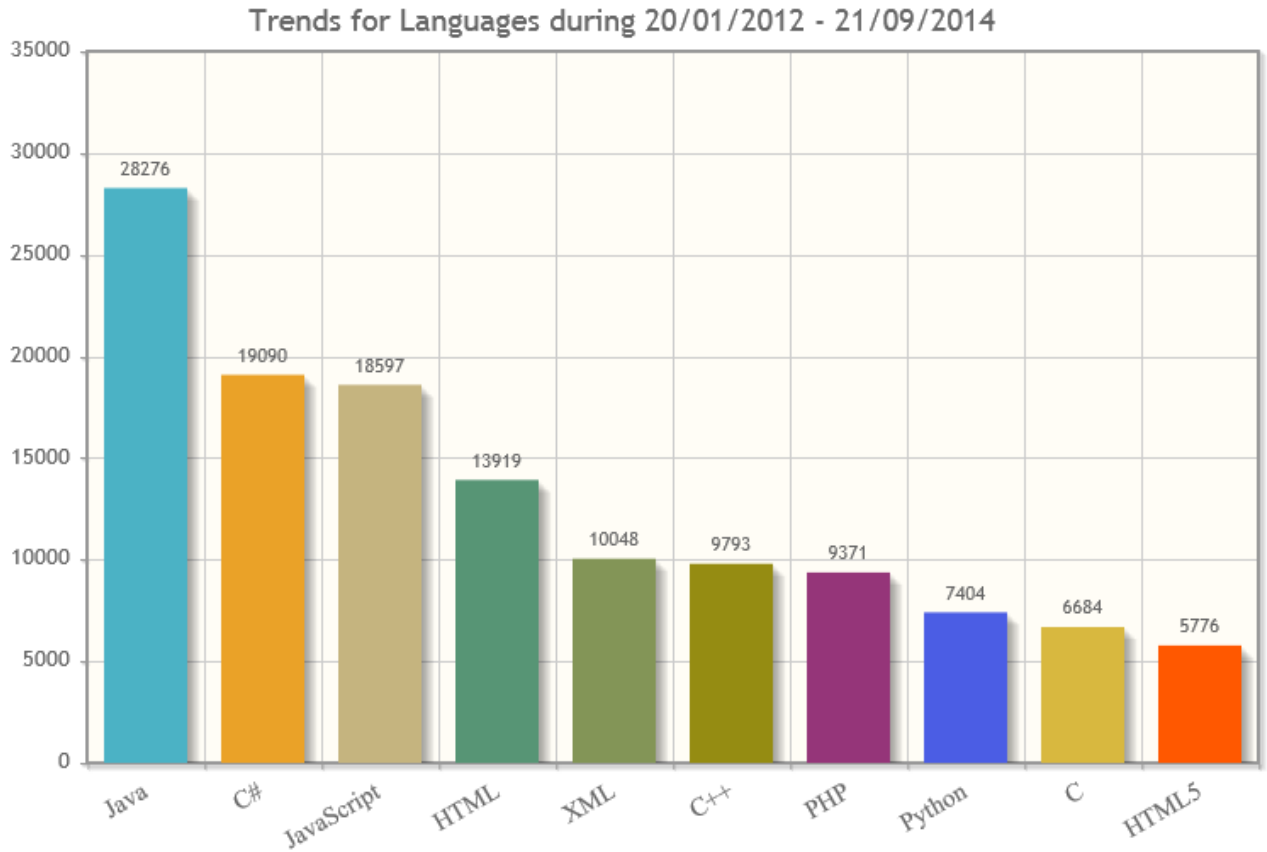

•Asynchronous methods
•Caller info attributes
C# 5.0
C# 4.0
•Dynamic binding
•Named and optional arguments
•Generic co- and contravariance
•Embedded interop types ("NoPIA")
•Implicitly typed local variables
•Object and collection initializers
•Auto-Implemented properties
•Anonymous types
•Extension methods
•Query expressions
•Lambda expressions
•Expression trees
•Partial Methods
C# 3.0
•Generics
•Partial types
•Anonymous methods
•Iterators
•Nullable types
•Private setters (properties)
•Method group conversions (delegates)
•Covariance and Contra-variance
•Static classes
C# 2.0
C# 1.2
C# 1.0
1995
2000
2005
2010

Java 진화


•Asynchronous methods
•Caller info attributes
C# 5.0
C# 4.0
•Implicitly typed local variables
•Object and collection initializers
•Auto-Implemented properties
•Anonymous types
•Extension methods
•Query expressions
•Lambda expressions
•Expression trees
•Partial Methods
•Dynamic binding
•Named and optional arguments
•Generic co- and contravariance
•Embedded interop types ("NoPIA")
C# 3.0
•Generics
•Partial types
•Anonymous methods
•Iterators
•Nullable types
•Private setters (properties)
•Method group conversions (delegates)
•Covariance and Contra-variance
•Static classes
C# 2.0
C# 1.2
C# 1.0
7
•Strings in switch
•Automatic resource management in try-statement
•Improved type inference for generic instance creation
•Simplified varargs method declaration
•Binary integer literals
•Allowing underscores in numeric literals
•Catching multiple exception types and rethrowing excep
with improved type checking
6
5
•Generics
•Auto boxing/unboxing
•Enhanced loop
•Static imports
•Annotation/Metadata
1.4
1.3
1.2
1.1
1.0
•Inner classes
•Anonymous inner class
1995
2000
2005
2010

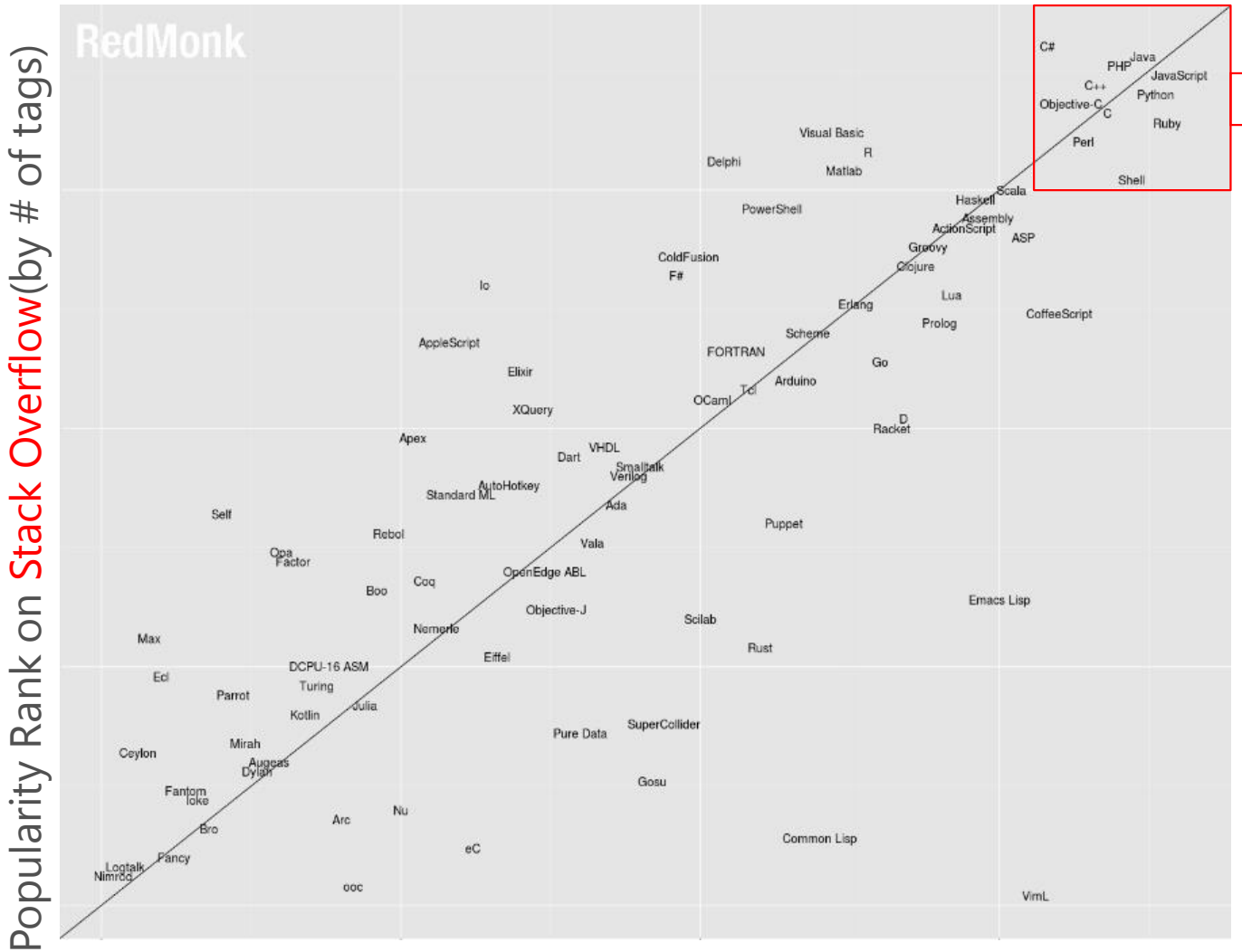
Trendy Skills 인덱스





www.trendyskills.com

RedMonk 인기 순위



Sogrady-media.redmonk.com

.NET: One platform, a world of opportunity

Microsoft .NET

.NET in devices and services

Cloud Services
Azure Mobile Services
Azure Web Sites

Windows Desktop
Async Programming
Frequent releases

Web apps
ASP.NET vNext
ASP.NET updates

Windows Store
.NET Native
Universal Windows apps

Cross-devices
Xamarin partnership
Portable Class Libraries

.NET foundation

Azure and Windows Server | Windows Desktop | Windows Store | iOS and Android

Core .NET

Runtime
Next gen JIT ("RyuJIT")
SIMD CoreCLR

Compilers
.NET Compiler Platform ("Roslyn")
Languages innovation .NET Native

오늘은 딱 3가지만

.NET Compiler Platform
(Project Roslyn)

새로운 C#,VB 컴파일러
풍부한 코드 분석 API

.NET Native

C#을 네이티브 머신
코드로 컴파일

RyuJIT

차세대 JIT 컴파일러

.NET Compiler Platform

What

Project Roslyn

Visual Studio “14” 기본 탑재 예정

Build 2014 이후 CodePlex를 통해 Open Source 형태로 공개

C#, VB Compiler in C# and VB
with Rich Public APIs
on CodePlex

Anders hejlsberg

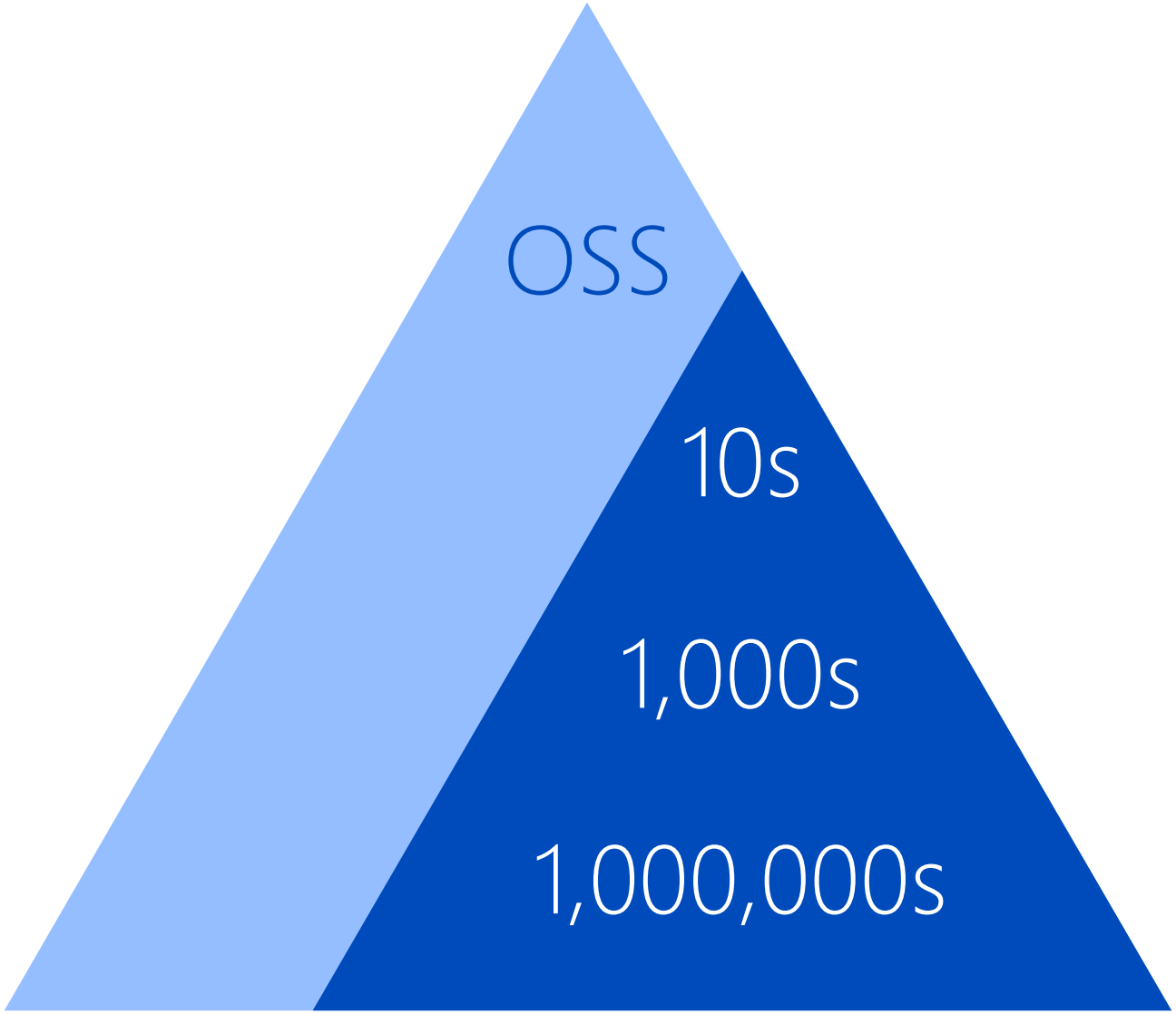
Why

이제 Open Source로

Team
혁신을 이어갈 수 있는 깔끔한 아키텍쳐

Partners
소스 기반 도구나 확장 도구 개발

Developers
더욱 더 강력한 C# IDE 기능 제공

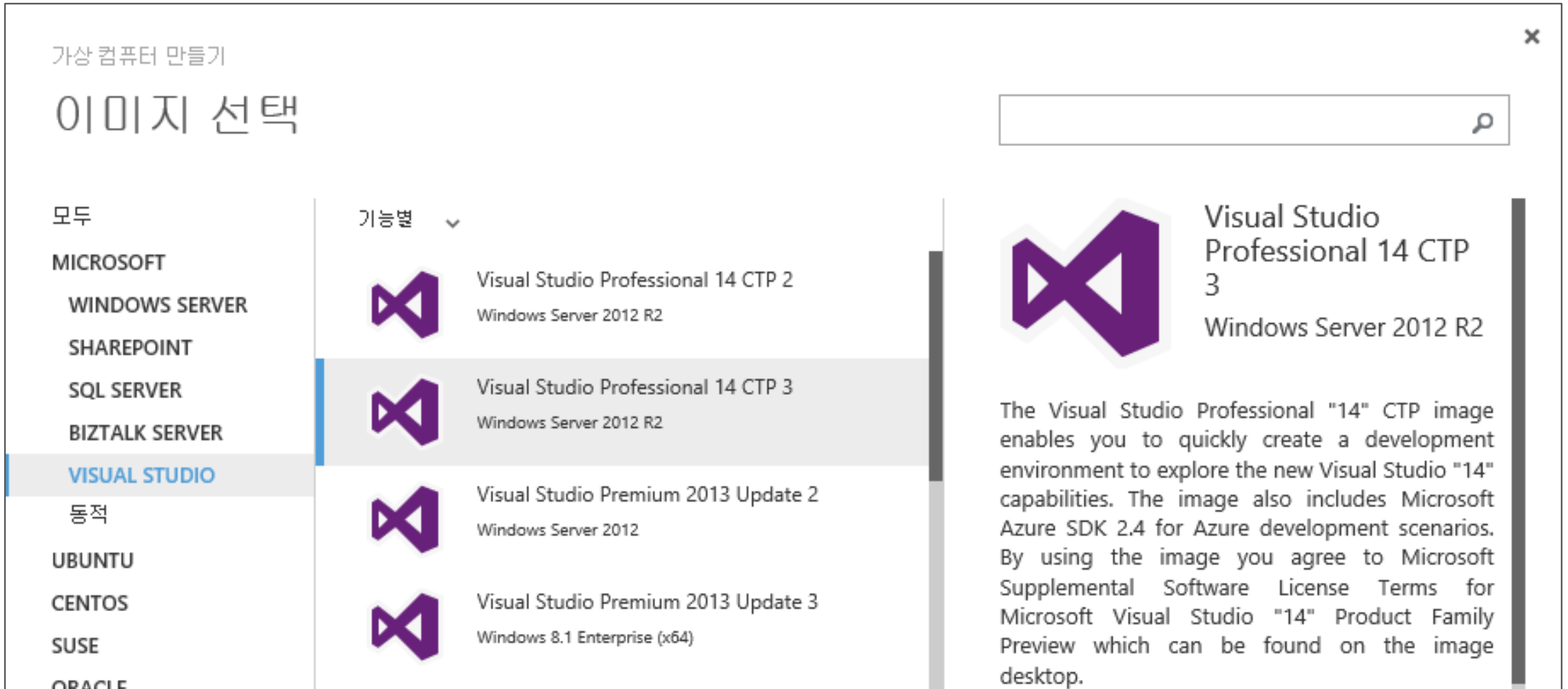


Demo

더욱더 강력한 C# IDE 기능
C#의 새로운 언어 기능
사용자 정의 분석

빠르게 살펴보기

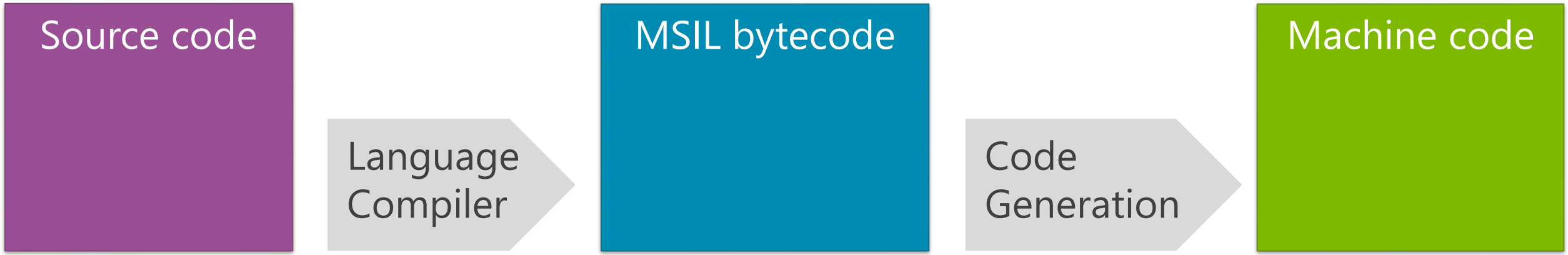
Azure(azure.com) → 가상 컴퓨터



.NET Native

What

- 스토어앱을 위해 클라우드에서 수행되는 차세대 클라우드 컴파일
- C#을 머신 코드로 컴파일
- 정적 링크
- 극도로 작아진 런타임을 사용하고, VC++의 Optimizer를 이용

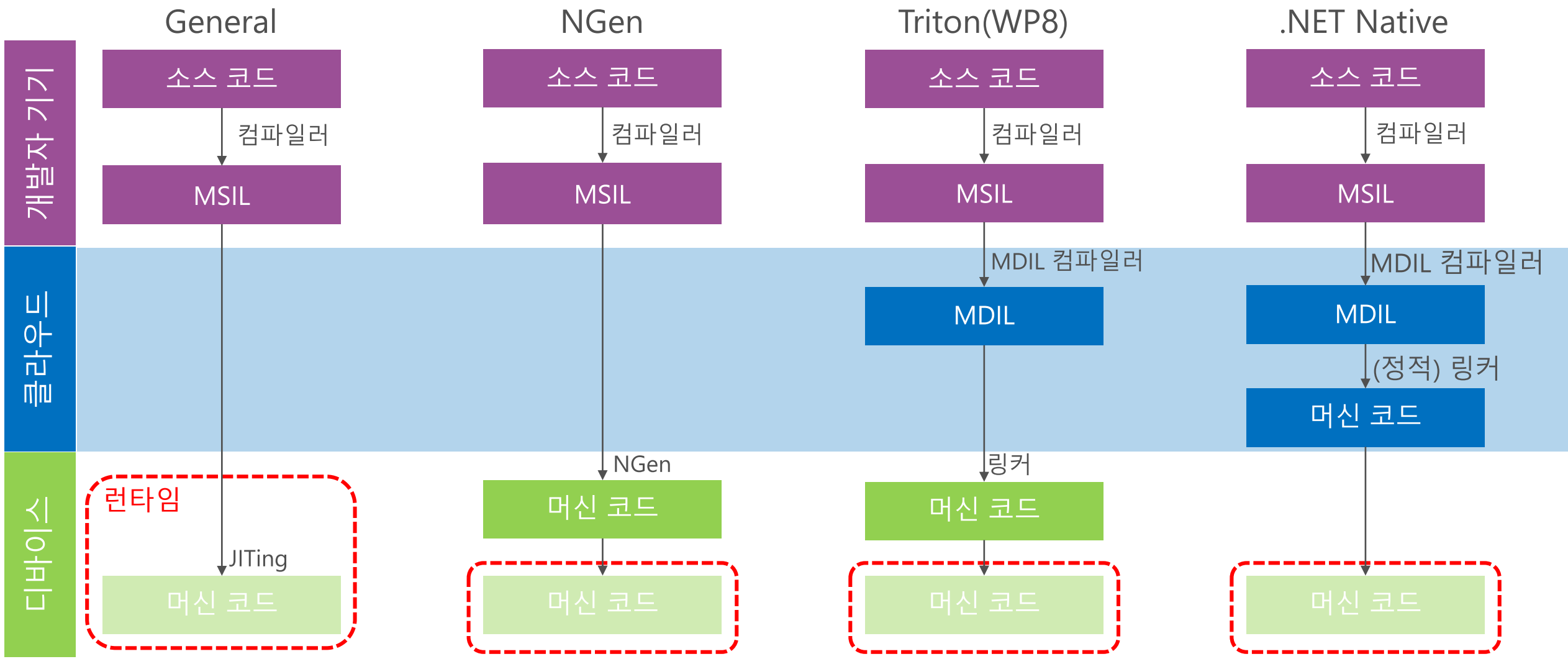


컴파일 모델

컴파일 모델


개발자 기기
디바이스
General
소스 코드
컴파일러
MSIL
런타임
JITing
머신 코드
NGen
소스 코드
컴파일러
MSIL
NGen
머신 코드
머신 코드

컴파일 모델


General
NGen
Triton(WP8)
.NET Native
개발자 기기
클라우드
디바이스
소스 코드
컴파일러
MSIL
MDIL 컴파일러
MDIL
(정적) 링커
머신 코드
NGen
링커
JITing
런타임

.NET Native and Desktop CLR

	.NET Native	Desktop CLR
Visual Studio	개발자는 Visual Studio를 이용하여 동일하게 개발	
라이브러리 Interop Serialization	• 리펙토링 된 라이브러리 사용 • 기기들의 특성을 고려하여 최적화됨 • Interop와 serialization 코드들을 빌드 타임에 생성	• 데스크탑 클라이언트, 서버, 엔터프라이즈 시나리오 등에서 광범위 하게 사용될 수 있는 라이브러리 • Interop와 serialization 코드는 런타임에 생성
정적 링킹	• 앱코드와 라이브러리를 정적 링크 • 필요한 메타정보만 선택적으로 포함시킬 수 있어서 앱의 크기를 줄이고 성능 향상	• 정적 링크 불가. 프레임워크 라이브러리는 모든 앱에서 공유 함
컴파일타임 최적화	• Visual C++ 컴파일러 Optimizer를 사용	• JIT 컴파일러는 신속히 작업을 완료해야 함. • 시간을 오래 소요하는 최적화는 수행하지 못함
런타임	• 더 가볍고 네이티브로 개발 되었으며, 메모리 안정적으로 가비지 수집을 수행	• 다양한 시나리에 대응하기 위해서 풍부한 기능을 가진 런타임

How

런타임 최적화

.NET Native Runtime(MRT)은 정적컴파일을 위해 CLR을 튜닝하고 최적화

라이브러리 최적화 및 단일화

..NET Native와 함께 배포되는 프레임워크 라이브러리 또한 정적 컴파일을 위해 리펙토링되고 계층화

미사용 코드 제거

.NET Native 컴파일러는 여러 단계를 통해 전체 코드를 분석하여 미사용 코드를 제거

전체 프로그램을 대상으로 최적화

C++ optimizer를 이용하여 전체 프로그램을 분석하고 프로그램을 하나의 묶음으로 최적화

개선되고 더 빨라진 Windows Runtime Interop 코드를 생성

응용 프로그램의 Interop 코드를 컴파일 시점에 생성하고, 링크하여 응용 프로그램 내부에 포함

Serializer 사전 생성

런타임이 아닌 컴파일 시점에 serialization 코드 생성

필요시 리플렉션 가능

프로그램에 어떤 메타데이터를 포함시킬지를 개발자가 제어

최소화 되고 최적화된 .NET 라이브러리와 런타임

C#을 이용하여 머신코드로 구성된 네이티브 앱 개발

Visual Studio의 최상의 개발 환경을 이용

Backend에 VC++ optimizer 컴파일러를 이용

앱의 기동 시간(60%)과 수행 속도는 빨라지고

메모리 사용량은 감소(15~20%)

Standalone app package(앱과 Runtime을 모두 포함)

기동 시간 개선 결과

Cold Startup

50% wins on average

App	Desktop Time (ms)	.NET Native Time (ms)	Improvement
Adobe Reader	2,935	1,576	46.3%
Audible	5,403	2,735	49.4%
Fresh Paint	2,714	1,411	48.0%
Hulu	4,415	2,451	44.5%
Khan	2,905	1,234	57.5%
Netflix	3,117	1,681	46.1%
Nick	3,044	1,254	58.8%
Nook	3,179	1,139	64.2%
Twitter	2,873	1,274	55.7%
		Average	**52.3%**

Warm Startup

30% wins on average

App	Desktop Time (ms)	.NET Native Time (ms)	Improvement
Adobe Reader	1,160	815	29.7%
Audible	2,367	1,750	26.1%
Fresh Paint	1,039	769	26.0%
Hulu	1,868	1,381	26.1%
Khan	917	615	32.9%
Netflix	1,543	1,017	34.1%
Nick	977	686	29.8%
Nook	1,319	508	61.5%
Twitter	1,039	658	36.7%
		Average	**33.7%**

Reference Set(Memory)

	Managed Module RS (MB)			Module RS (MB)			Total RS (MB)		
App	Desktop	.NET Native	Improvement	Desktop	.NET Native	Improvement	Desktop	.NET Native	Improvement
Adobe Reader	9.4	3	68%	29.4	22.3	24%	49.3	38.5	22%
Audible	16.3	4.5	72%	43.4	31.6	27%	95.1	77.6	18%
Fresh Paint	24.2	9	63%	60.6	46.7	23%	190.2	177.9	6%
Hulu	16.7	5.3	68%	43.8	31.2	29%	169.6	148.9	12%
Khan	17.1	6	65%	108	96.7	10%	215.8	201.8	6%
Netflix	25.9	8.1	69%	63.3	45.4	28%	244.6	206.2	16%
Nick	18.8	5.2	72%	44.6	31.3	30%	124.4	104.1	16%
Nook	19	5.2	73%	54	39.4	27%	126.3	116.7	8%
Twitter	24.2	7.5	69%	55.9	38.7	31%	97.5	73.5	25%
Average			**69%**			**25%**			**14%**

Demo

Visual Studio를 이용한 .NET Native 컴파일
로드된 모듈 살펴보기

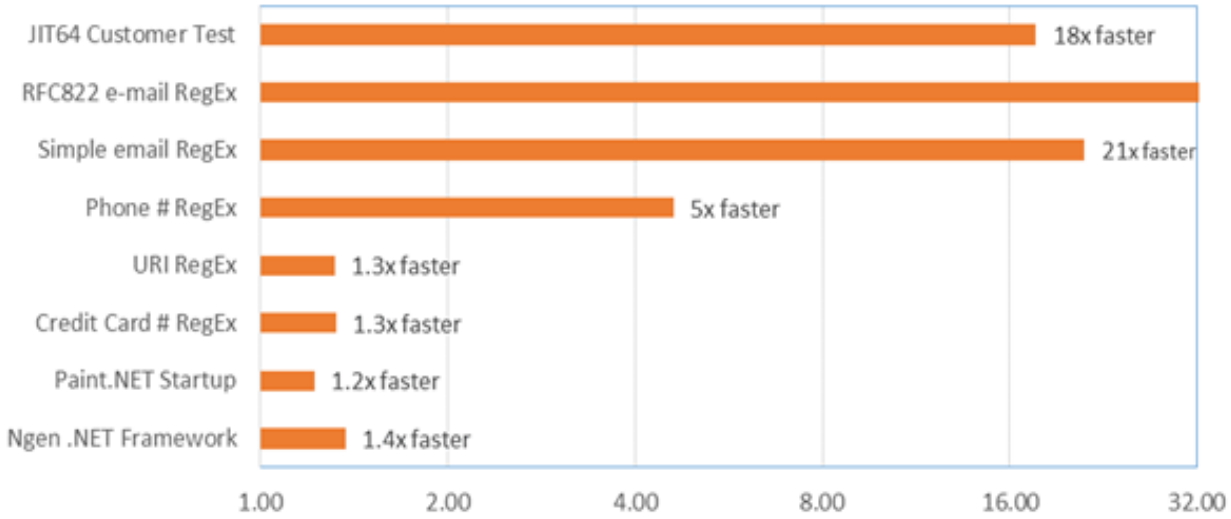
RyuJIT

What?

- .NET을 위한 차세대 64비트 JIT 컴파일러
- 코드 생성 속도 개선, 코드 품질 개선
- 고급 최적화 기능 포함(e.g. SIMD, …)

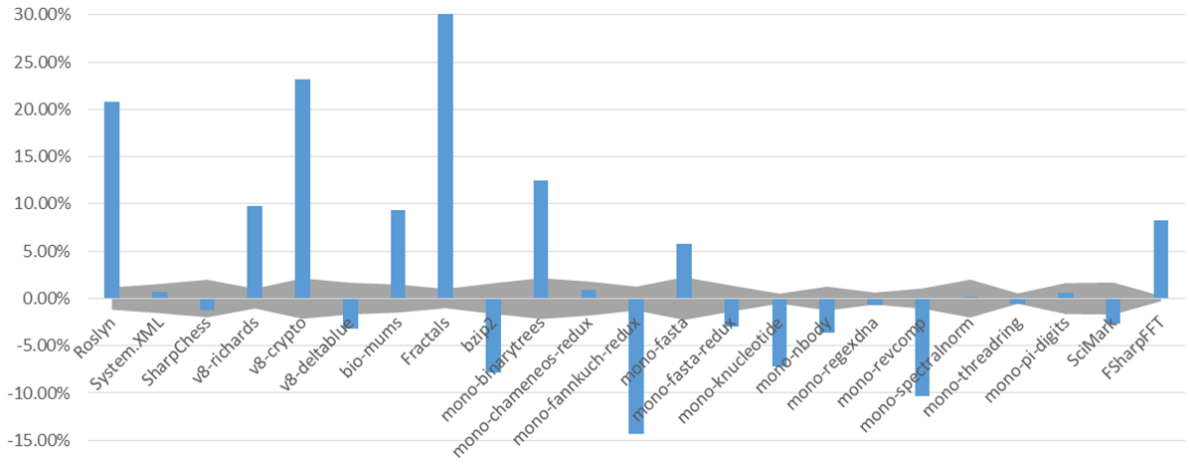
코드 생성 속도

How fast JIT compiler generate app code



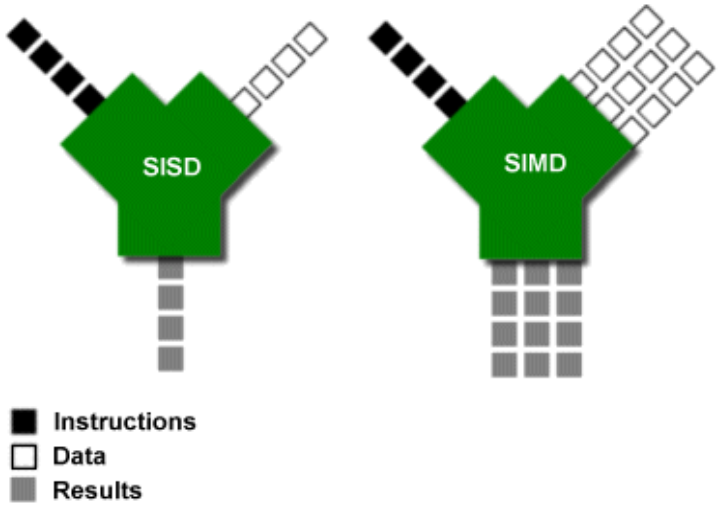
코드 품질

How fast generated code runs



SIMD(Single Instruction Multiple Data)

- .NET에서 data parallelism을 사용할 수 있도록 함
- 게임, 수식 연산, 이미지 처리 등의 응용 프로그램의 성능을 개선
- Nuget을 이용하여 .NET 라이브러리 형태로 이용 가능



Each compute node performs the same task on multiple “streams” of data

Designed to enable auto-scaling

고급 최적화를 통한 플랫폼의 혁신적 개선

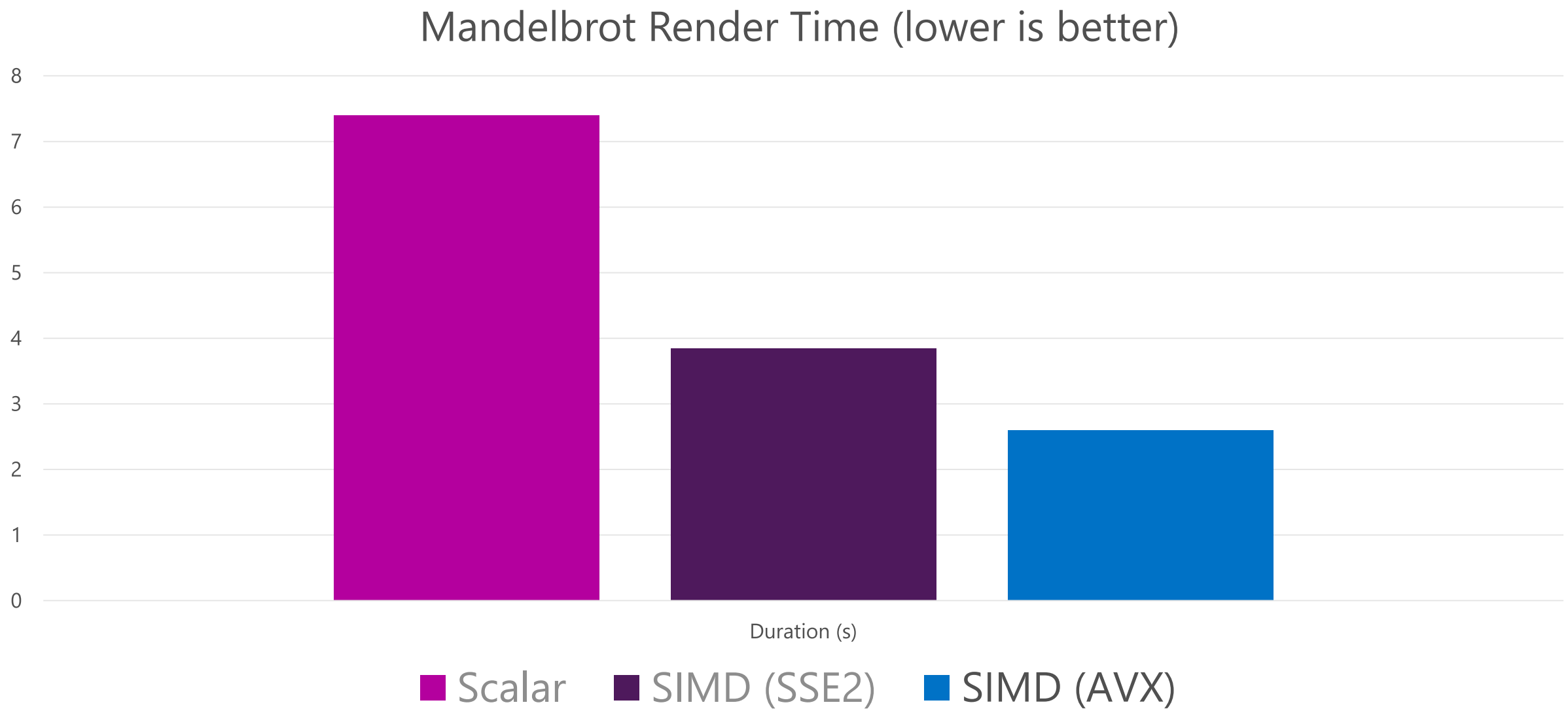
- SIMD, Loop invariant code motion
- Floating point performance

RyuJIT은 기존 JIT32/JIT64 코드베이스를 대체

빠른 코드 생성 속도를 위해 JIT32를 기반으로 함

훌륭한 코드 품질을 위해 JIT64의 최적화를 이용

성능 비교


Mandelbrot Render Time (lower is better)
8
7
6
5
4
3
2
1
0
Duration (s)
Scalar
SIMD (SSE2)
SIMD (AVX)

.NET 컴파일: 로드맵

Native

NGEN: 디바이스에 수행

MDIL-NGEN (Hybrid):
클라우드+디바이스에서 수행

.NET Native: 클라우드에서 수행
(정적 컴파일을 위해 최적화)

Dynamic

JIT 최적화

Codename “RyuJIT”:
차세대 JIT 컴파일러

Demo

RyuJIT 성능
SIMD 지원

C#은 Java의 카피캣에 지나지 않는다.

.NET은 사용하지 않는 플랫폼이다.

C#와 .NET은 더 이상 발전이 없다.

결론

변화와 혁신을 거듭하고 있는 언어/프레임워크

.NET Compiler Platform (Project Roslyn)	.NET Native	RyuJit
New C# & VB Compiler Rich code analysis APIs	Compile C# to native machine code	Next Generation JIT Compiler

tech·days Korea 2014
개발자 컨퍼런스