AI编译器系列

LLVM架构和原理

ZOMI

Talk Overview

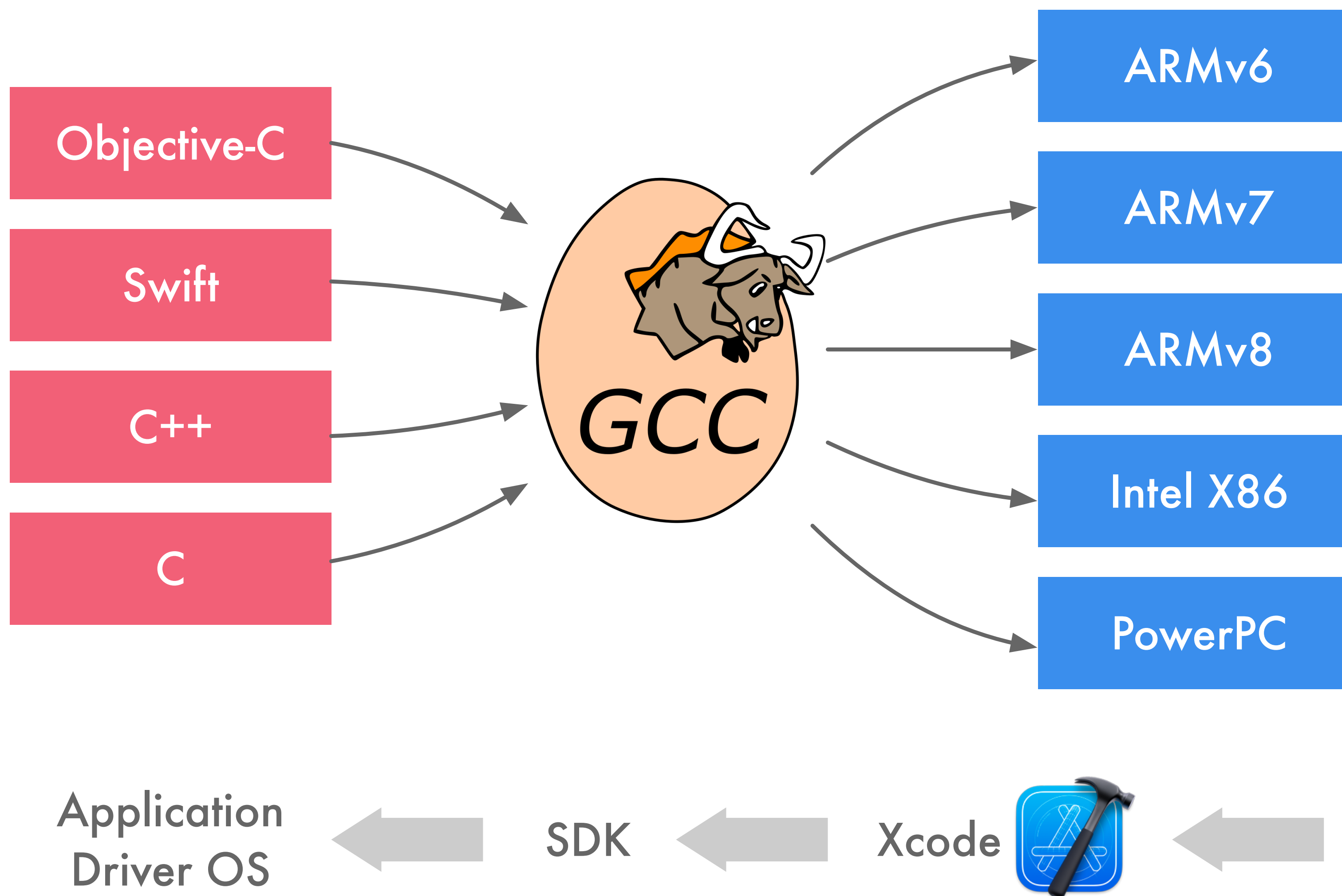
1. **传统编译器**
   - History of Compiler - 编译器的发展
   - GCC process and principle – GCC 编译过程和原理
   - LLVM/Clang process and principle – LLVM 架构和原理
2. **AI编译器**
   - History of AI Compiler – AI编译器的发展
   - Base Common architecture – AI编译器的通用架构
   - Different and challenge of the future – 与传统编译器的区别，未来的挑战与思考

Talk Overview

LLVM/Clang process and principle – LLVM 架构和原理

- LLVM 项目发展历史
- LLVM 基本设计原则和架构
- LLVM 中间表示 LLVM IR
- LLVM 前端过程
- LLVM 中间优化
- LLVM 后端生成
- 基于 LLVM 项目

Complicate Ecosystem



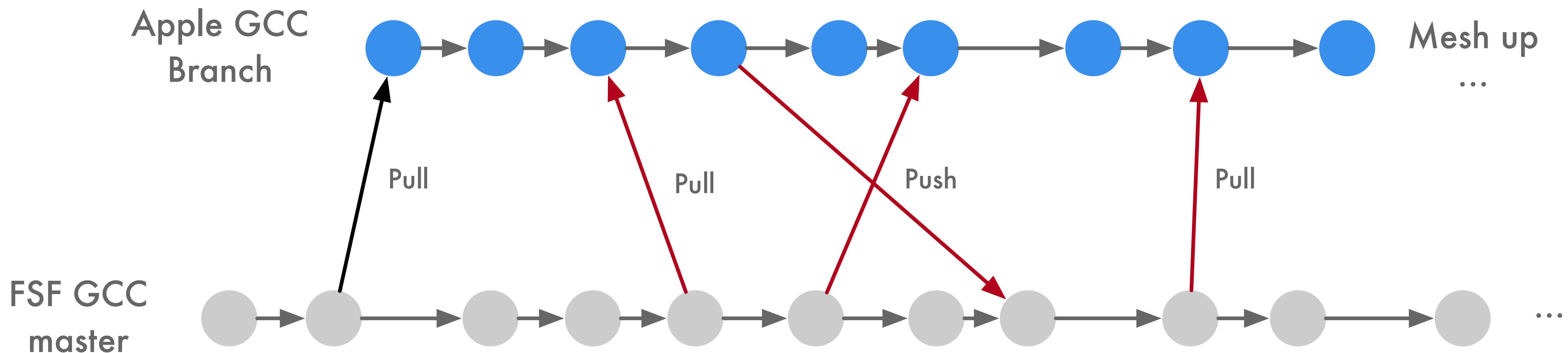
Apple needs find a way out

NeXTSTEP

1997 1997 2001

1998 Power Macintosh G3
1999 Power Macintosh G4
2000 PowerBook
2001 iPod
2002 iPod2
2003 iPod3
2004 iPod4 & Mini & Photo
2005 iPod5
iPod Shuffle
iPod Nano
Power Macintosh G5 (Intel)
2006 MacBook Pro
2007 Apple TV
iPhone
2008 MacBook Air
iPod Touch
iPhone 3G

Apple needs find a way out



GCC is developed for solving real problems,
it has no time to make a good everything perfect.

Apple met LLVM

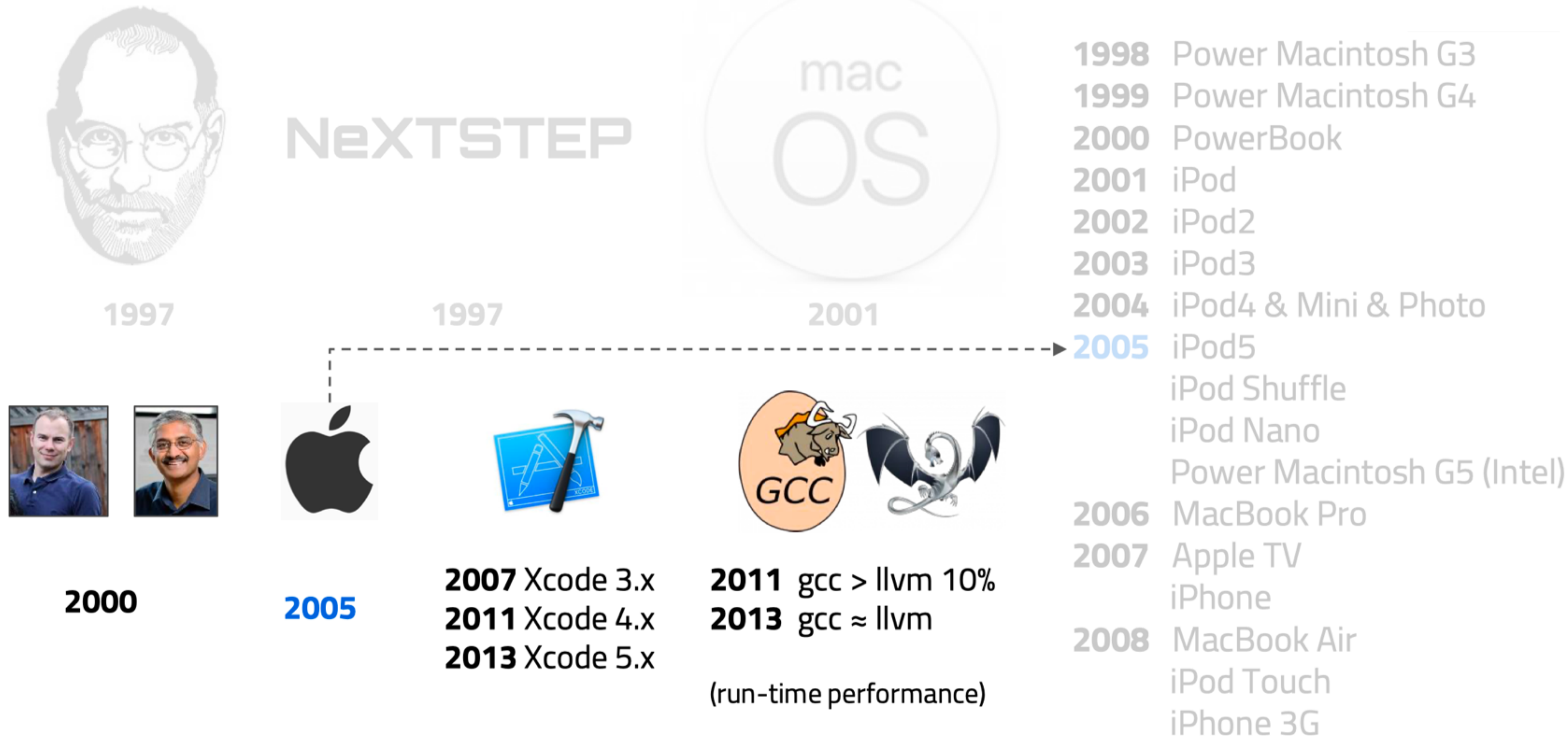
Chris Lattner

Twitter: https://twitter.com/clattner_llvm

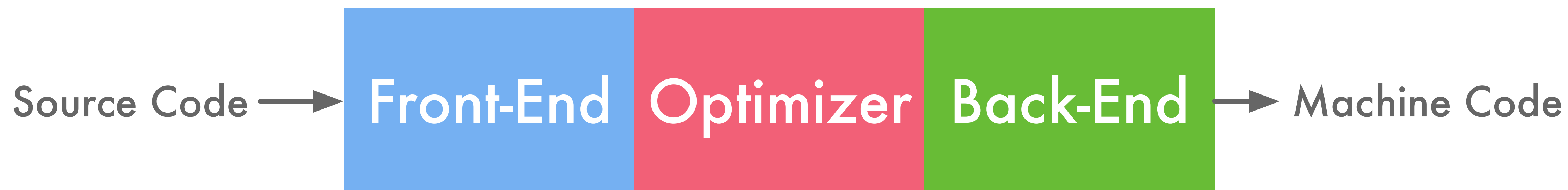
Website: http://nondot.org/sabre

www.hiascend.com
www.mindspore.cn

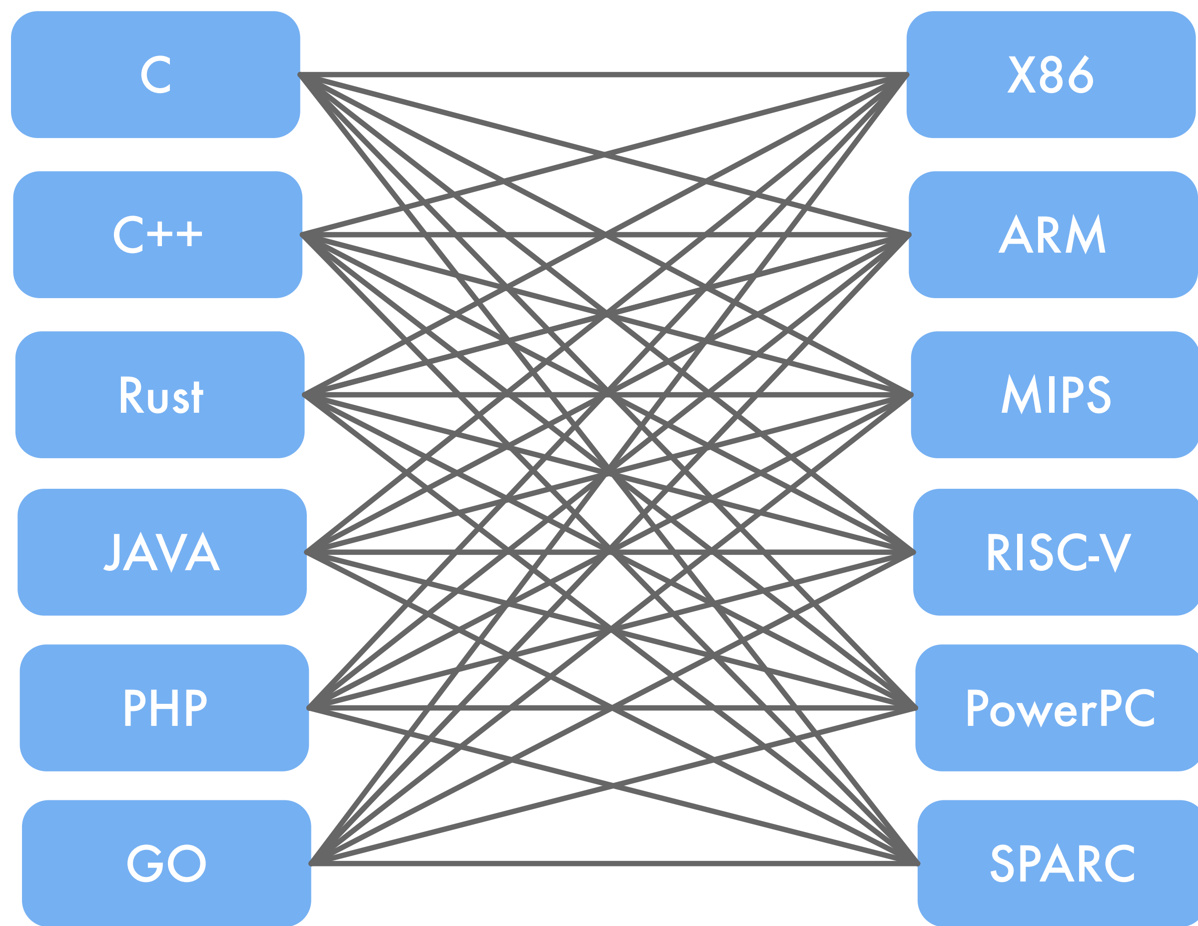
Apple met LLVM



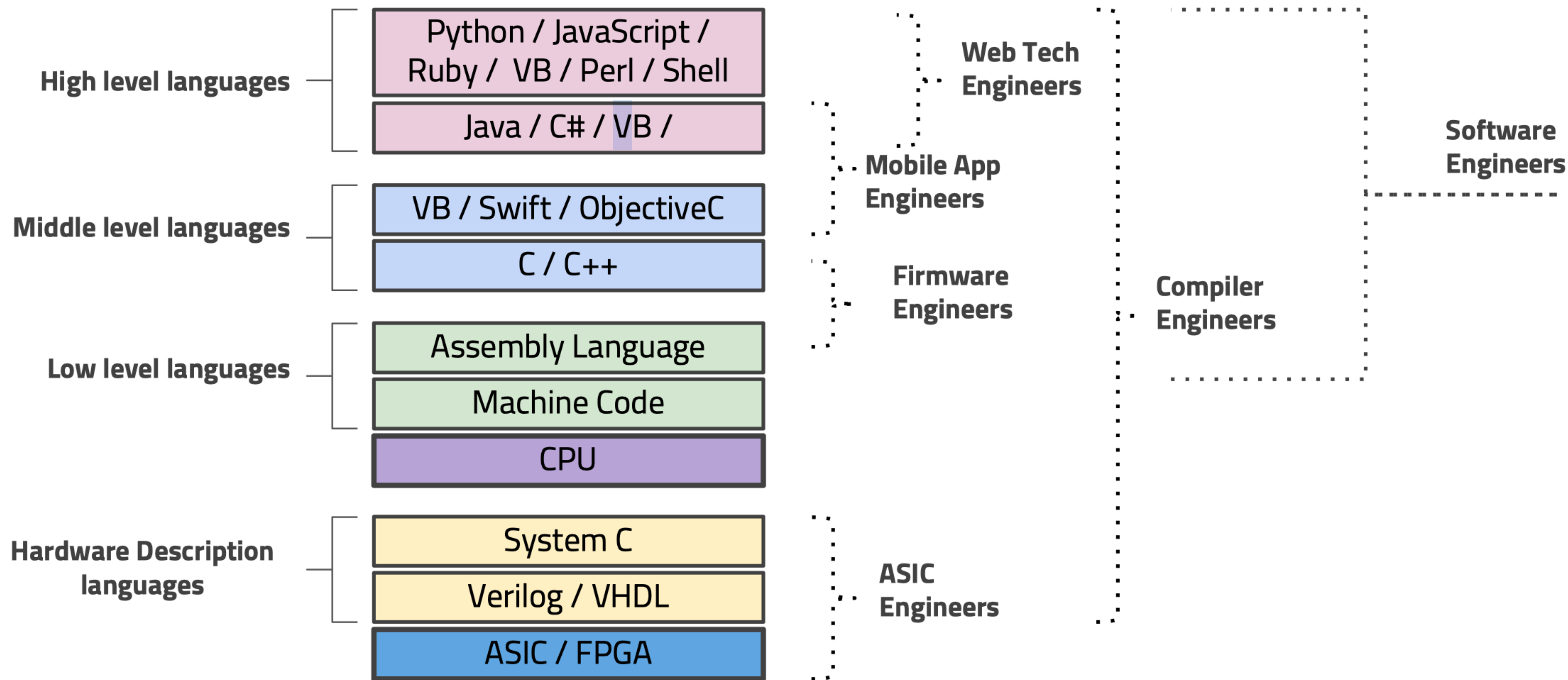
Compiler basic constitution



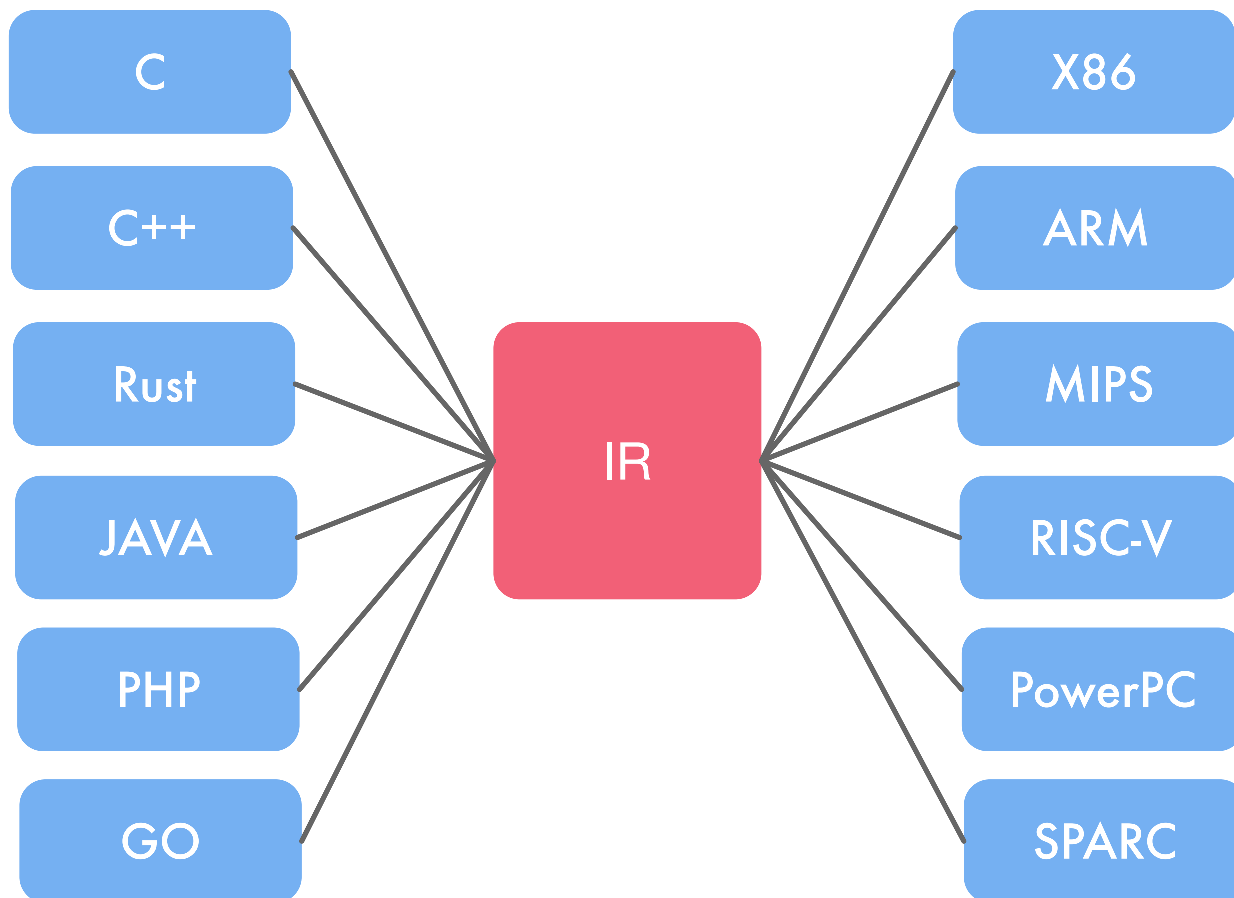
Traditional Compiler vs Modern Compiler



Computer Language stacks



Traditional Compiler vs Modern Compiler



www.hiascend.com
www.mindspore.cn

What is LLVM

- LLVM is a Compiler
- LLVM is a Compiler Infrastructure
- LLVM is a series of Compiler Tools
- LLVM is a Compiler Toolchain
- LLVM is an open source C++ implementation
- LLVM 项目发展为一个巨大的编译器相关的工具集合

Lib base LLVM

Lib base LLVM

LLVM Core	即 LLVM 的核心库，主要是围绕 LLVM 中间代码的一些工具，它提供了一个“源”和“目标”无关的优化器和几乎所有主流 CPU 类型的代码（机器码）生成器。
Clang	是 LLVM 项目中的一个子项目。它是基于 LLVM 架构的轻量级编译器，诞生之初是为了替代 GCC，提供更快的编译速度。它是负责编译C、C++、Objecte-C 语言的编译器，它属于整个 LLVM 架构中的，编译器前端。
Compiler-RT	项目用于为硬件不支持的低级功能提供特定于目标的支持。例如，32位目标通常缺少支持64位的除法指令。Compier-RT通过提供特定于目标并经过优化的功能来解决这个问题，该功能在使用32位指令的同时实现了64位除法。为代码生成器提供了一些中间代码指令的实现，这些指令通常是目标机器没有直接对应的，例如在32位机器上将 double 转换为 unsigned integer 类型。此外该库还为一些动态测试工具提供了运行时实现，例如 AddressSanitinzer、ThreadSanitizer、MemorySanitizer 和 DataFlowSanitizer 等。
LLDB	LLDB是一个LLVM的原生调试器项目，最初是XCode的调试器，用以取代GDB。LLDB提供丰富的流程控制和数据检测的调试功能。
LLD	clang/llvm内置的链接器。
Dragonegg	GCC插件，可将GCC的优化和代码生成器替换为LLVM的相应工具。
libc	C标准库实现。
libcxx/libcxxabi	C++标准库实现。
libclc	OpenCL标准库的实现。
OpenMP	提供一个OpenMP运行时，用于Clang中的OpenMP实现。
polly	支持高级别的循环和数据本地化优化支持的LLVM框架，使用多面体模型实现一组缓存局部优化以及自动并行和矢量化。
vmkit	基于LLVM的Java和.Net虚拟机实现。
klee	基于LLVM编译基础设施的符号化虚拟机。它使用一个定理证明器来尝试评估程序中的所有动态路径，以发现错误并证明函数的属性。klee的一个主要特征是它可以在检测到错误时生成测试用例。
SAFECode	用于C/C++程序的内存安全编译器。它通过运行时检查来检测代码，以便在运行时检测内存安全错误（如缓冲区溢出）。它可以用户保护软件免受安全攻击，也可用作Valgrind等内存安全错误调试工具。

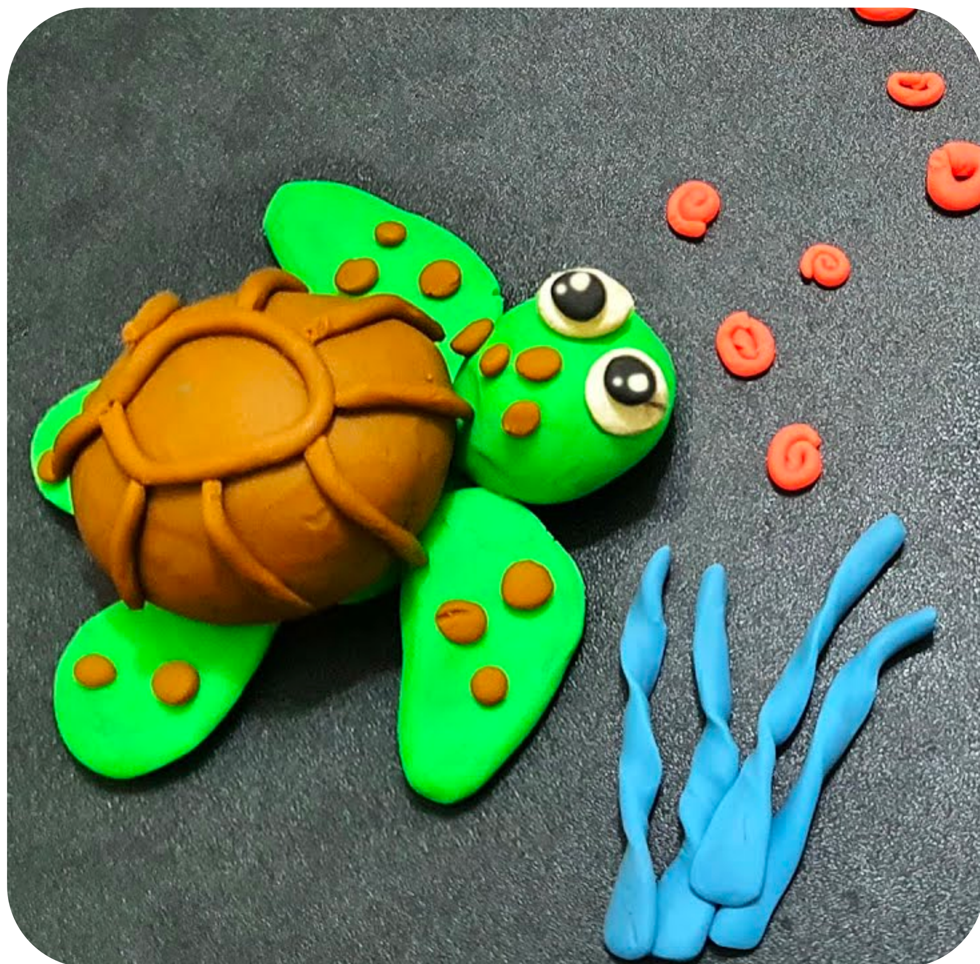
LLVM vs GCC

- 把编译器移植给新的语言只需要实现一个新的编译前端，已有的优化和后端都能实现复用；
- 如果前后端和解析器没有相互解耦，新语言编译器需要支持 N 个目标机和 M 种语言(N*M)；

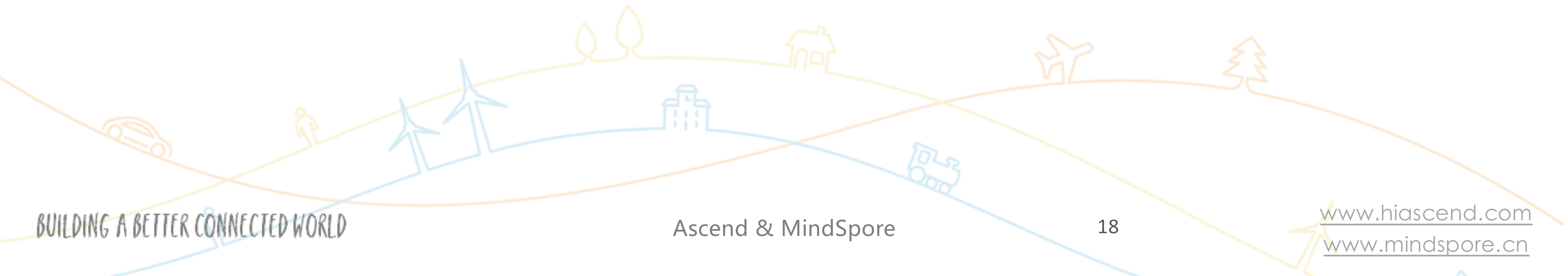
- LLVM 组件之间交互发生在高层次抽象，不同组件隔离为单独程序库，易于在整个编译流水线中集成转换和优化 Pass。现在被作为实现各种静态和运行时编译语言的通用基础结构；
- GCC 饱受分层和抽象漏洞困扰：编译前端生成编译后端数据的结构，编译后端遍历前端抽象语法树（AST）来生成调试信息，整个编译器依赖命令行设置的全局数据结构；

www.hiascend.com
www.mindspore.cn

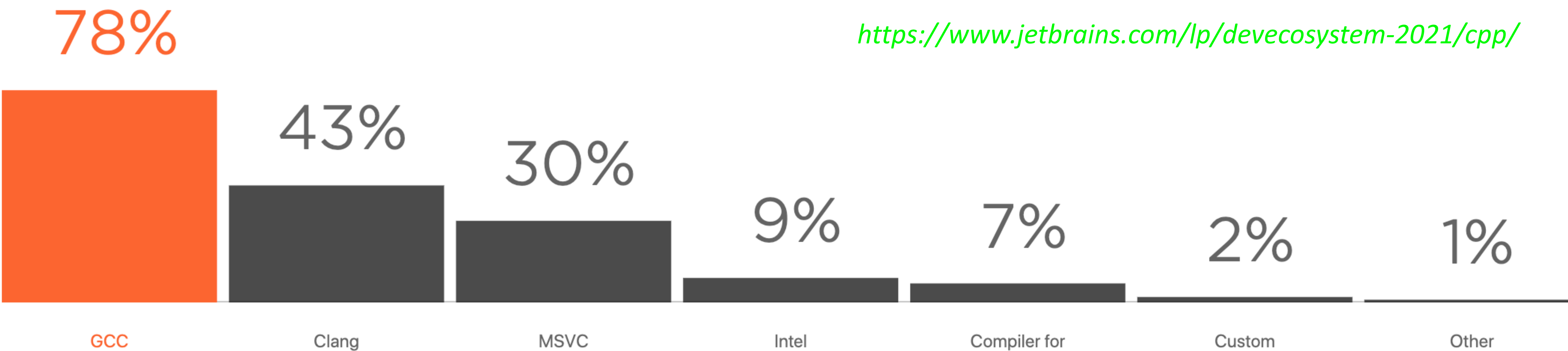
What I see the different like this ...



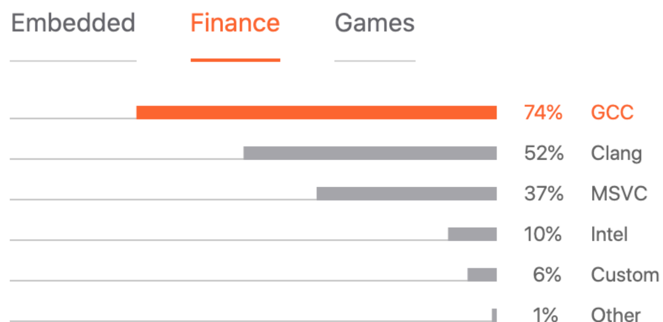
CLANG vs GCC

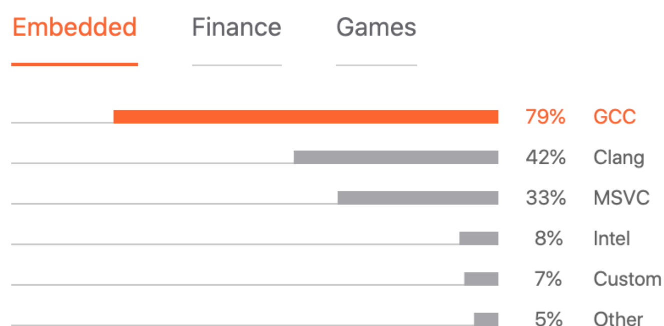


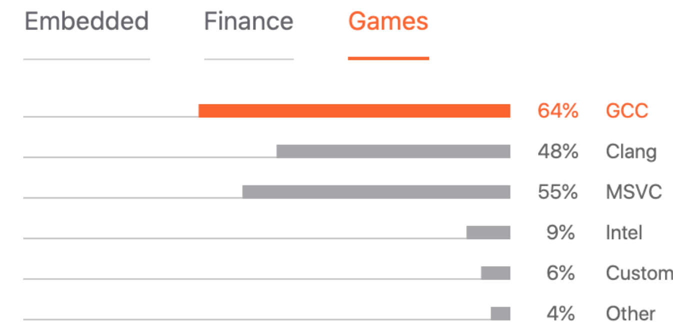
Which compilers do your regularly use?



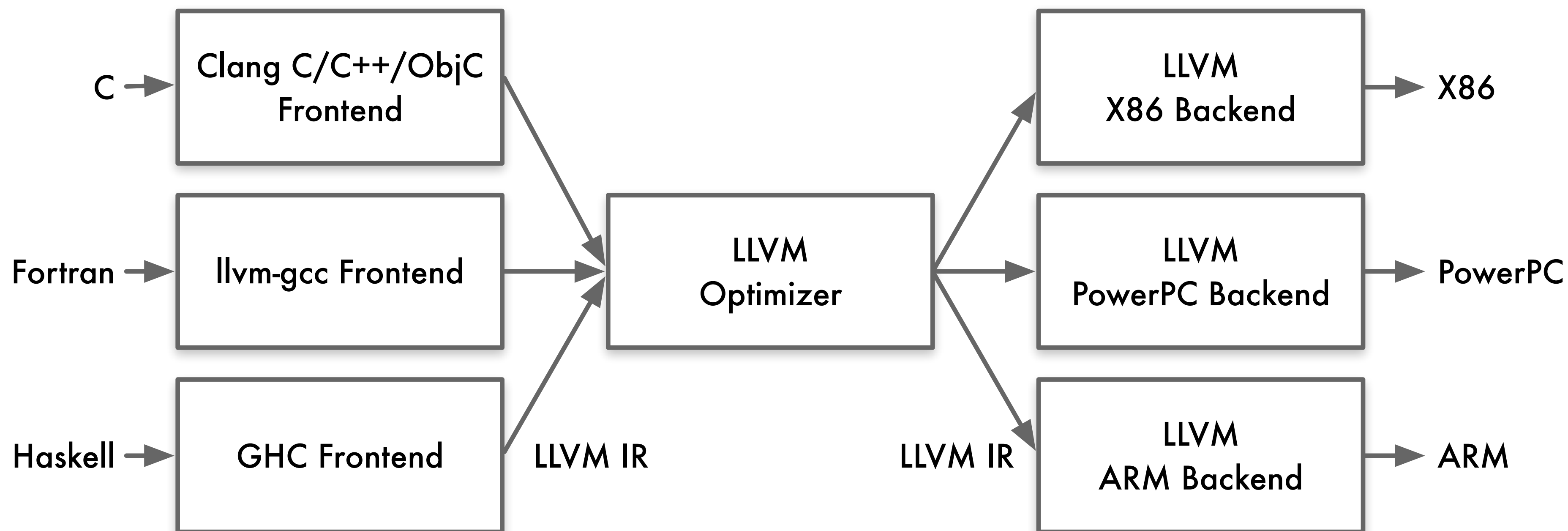
https://www.jetbrains.com/lp/devecosystem-2021/cpp/





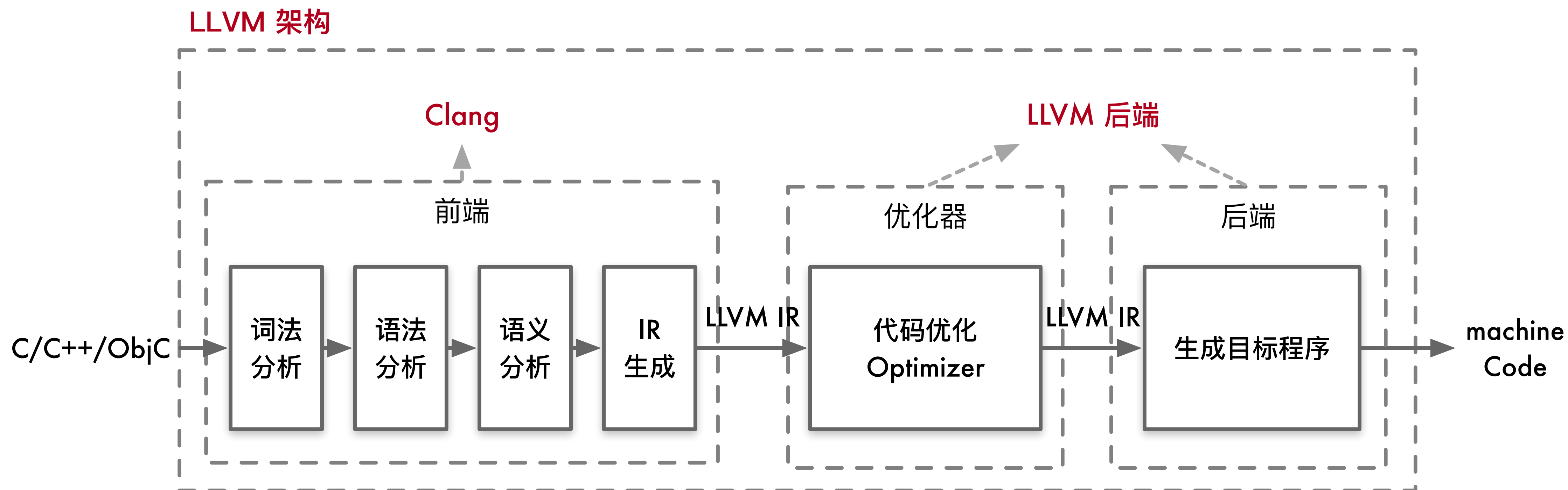


LLVM Architecture



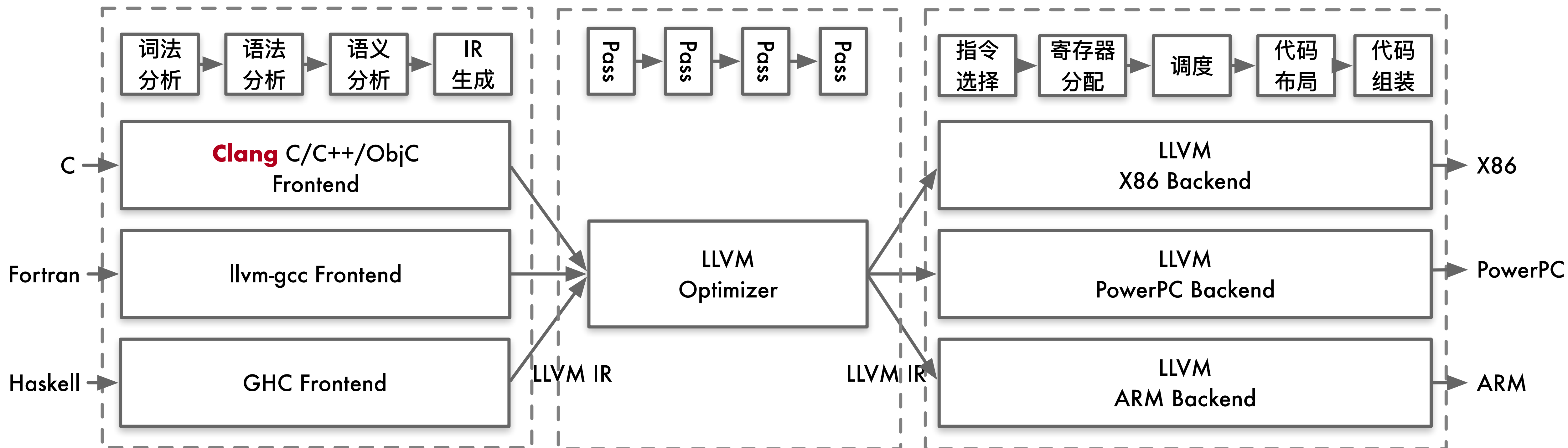
www.hiascend.com
www.mindspore.cn

LLVM Architecture



www.hiascend.com
www.mindspore.cn

LLVM Architecture



Huawei Confidential. Ascend & MindSpore

Pro's of GCC vs Clang

- GCC supports languages that Clang does not aim to, such as Java, Fortran, AN, Go, etc.
- GCC supports more targets than LLVM.
- GCC supports many language extensions.

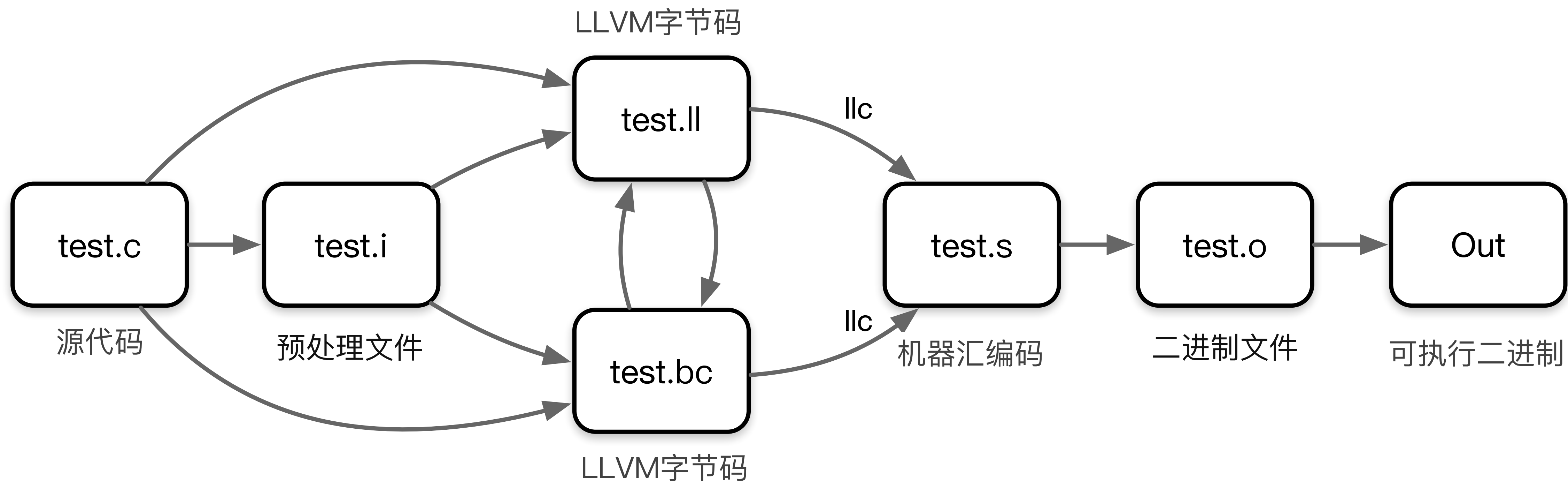
www.hiascend.com
www.mindspore.cn

Pro's of Clang vs GCC

- The Clang ASTs and design(Modular design) are intended to be easily understandable by anyone.
- Clang is designed as an API from its inception, allowing it to be reused by source analysis tools, refactoring, IDEs as well as for code generation. GCC is built as a monolithic static compiler.
- Various GCC design decisions make it very difficult to reuse , Clang has none of these problems.
- Fast compilation speed. In Debug mode, OC compilation speed is three times faster than GCC. Small memory consumption.

www.hiascend.com
www.mindspore.cn

How to run...



How to Use LLVM

How to run…

- clang -E -c hello.c -o hello.i
- clang -emit-llvm hello.i -c -o hello.bc
- clang -emit-llvm hello.i -S -o hello.ll
- llvm-dis hello.bc -o hello.ll
- llvm-as hello.ll -o hello.bc

- clang -O2 -S hello.c -emit-llvm -o hello.ll

- llc hello.ll -o hello.s
- clang hello.s -o hello

How to run…

- clang -ccc-print-phases hello.c
- clang -Xclang -ast-dump -c hello.c

www.hiascend.com
www.mindspore.cn

THANK YOU

Copyright©2014 Huawei Technologies Co., Ltd. All Rights Reserved.
The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.