Accelerating or Complicating PHP Execution by LLVM Compiler Infrastructure

Jim Huang (黃敬群)

Developer, 0xlab

jserv@0xlab.org

PHPConf.TW / Nov 12, 2011

到底要講什麼?


如果產能是我們砍掉重練
那我就讓你們看見直譯的驕傲
賽德克巴萊
3.26 OSDC
11.12 PHPConf
WWW.SEEDIQBALETHEMOVIE.COM


如果產能是我們砍掉重練
那我就讓你們看見直譯的驕傲
如果產能是我們砍掉重練
那我就讓你們看見直譯的驕傲
PHP 快來
3.26 OSDC
11.12 PHPConf
WWW.SEEDIQBALETHEMOVIE.COM

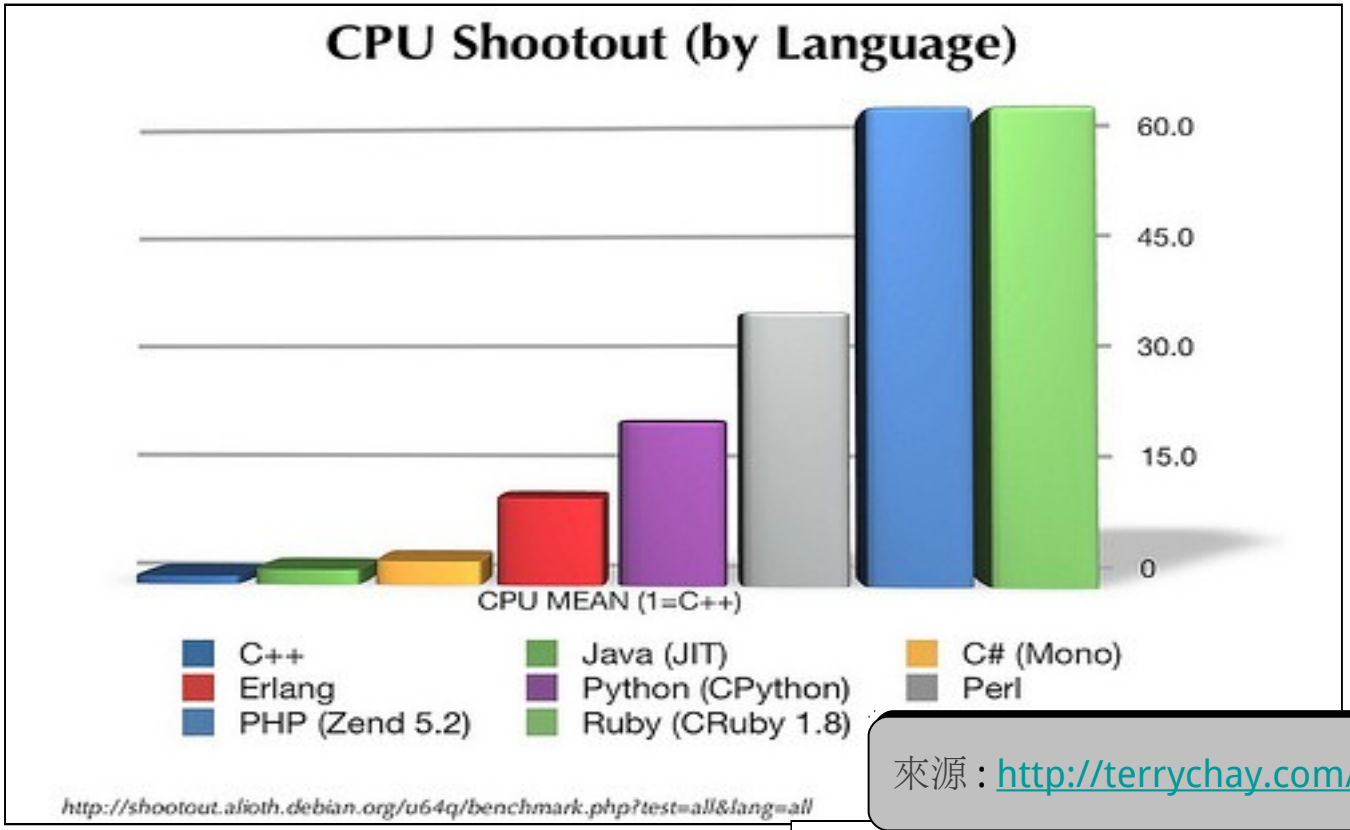
PHP 巴菜 (眞實的 PHP)

- PHP 5 具備物件導向設計的語言特徵
  - PDO = PHP Data Objects
- 自 PHP 4 ， PHP 已是編譯型語言
  - 在 2000 發布 PHP4 前， PHP 的解析與執行都是透過 PHP 直譯器
  - PHP4 引進 Zend engine
- Zend engine/VM 內部處理 bytecode ，並允許特定的快取擴充套件，或統稱「 PHP 加速器」，來進一步執行 PHP
  - PHP → bytecode
  - bytecode → 加速器執行

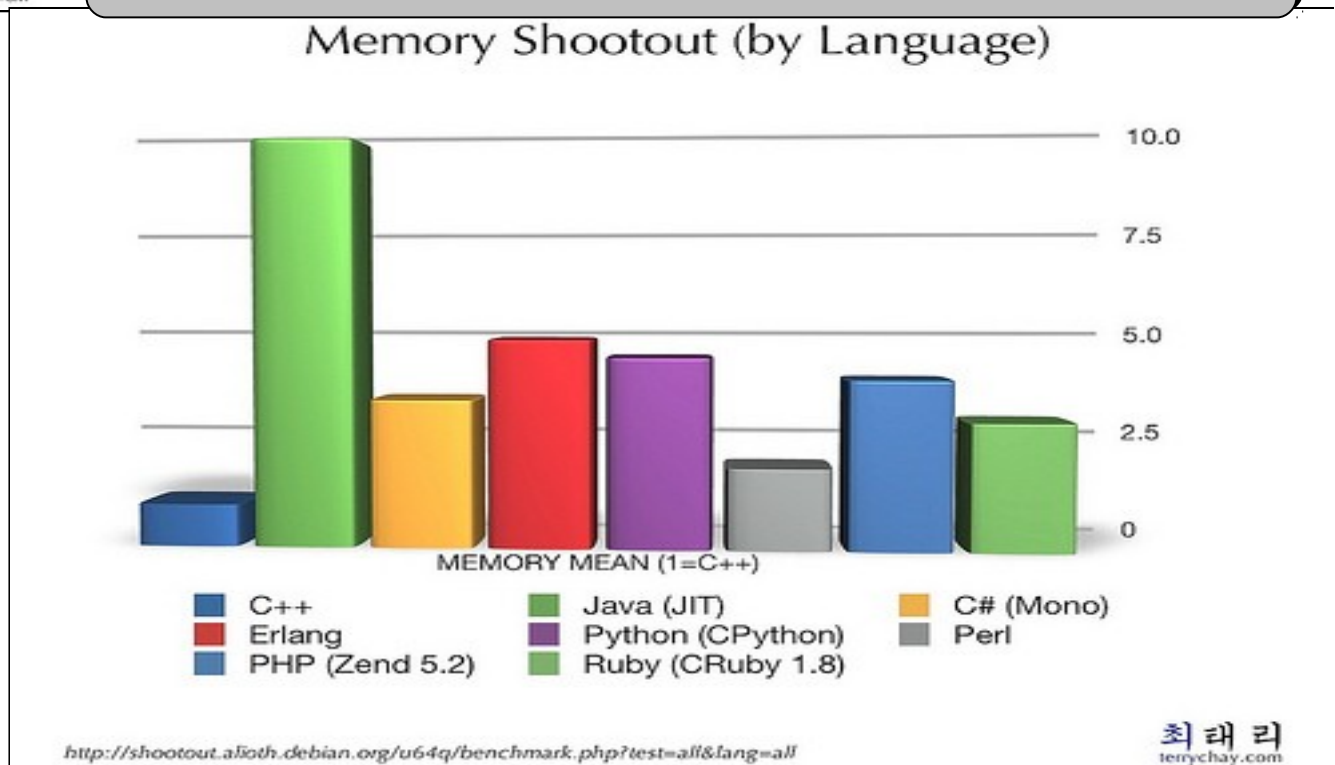
PHP 快（加速）來（廣泛貼近）

現有的 PHP 加速器

- PHC (PHP Open Source Compiler)
  http://www.phpcompiler.org/
- HipHop for PHP, Facebook
  http://wiki.github.com/facebook/hiphop-php/
- Roadsend PHP
  - pcc
    http://code.roadsend.com/pcc
  - Raven (以 LLVM 重寫)
    http://code.roadsend.com/rphp
- 以及其他眾多商業解決方案



來源：http://terrychay.com/article/hiphop-for-faster-php.shtml/4



相關議程：

〈窮得只剩下 Compiler〉

OSDC.tw 2009

http://www.slideshare.net/jserv/what-can-compilers-do-for-us

相關議程：

〈身騎 LLVM ，過三關：
淺談編譯器技術的嶄新應用〉

TOSSUG 2009

http://www.slideshare.net/jserv/llvm-introduction

相關議程：

〈 Applied Computer Science Concepts in Android 〉

台大資訊系 2010

http://www.slideshare.net/jserv/applied-computer-science-concepts-in-android

相關議程：

〈 Build Programming Runtime with LLVM 〉

OSDC.tw 2011

http://www.slideshare.net/jserv/build-programming-language-runtime-with-llvm

提綱

(1) 因應需求的 LLVM 架構

(2) 結合 PHP 到 LLVM

(3) 應用型態

因應需求的 LLVM 編譯器架構

Compiler 領導技術的時代

- 運算模式已大幅改觀
- Framework-driven
- SIMD/vectorization, SMP/multi-core
- 虛擬化 (Virtualization) 技術的時代

  → 更多元、更安全、更有效率地使用硬體
- 非對稱運算環境
- LLVM 的大一統宏願

案例:
Portable Native Client, OpenCL (GPGPU)

到處都有 VM

Java Virtual Machine (JVM)

.NET Common Language Runtime (CLR)

Smalltalk

Squeak

Parrot (Perl 6)

Python

YARV (Ruby 1.9)

Rubinius

Tamarin (ActionScript)

Valgrind (C++)

Lua

TrueType

Dalvik

Adobe Flash (AVM2)

p-code (USCD Pascal)

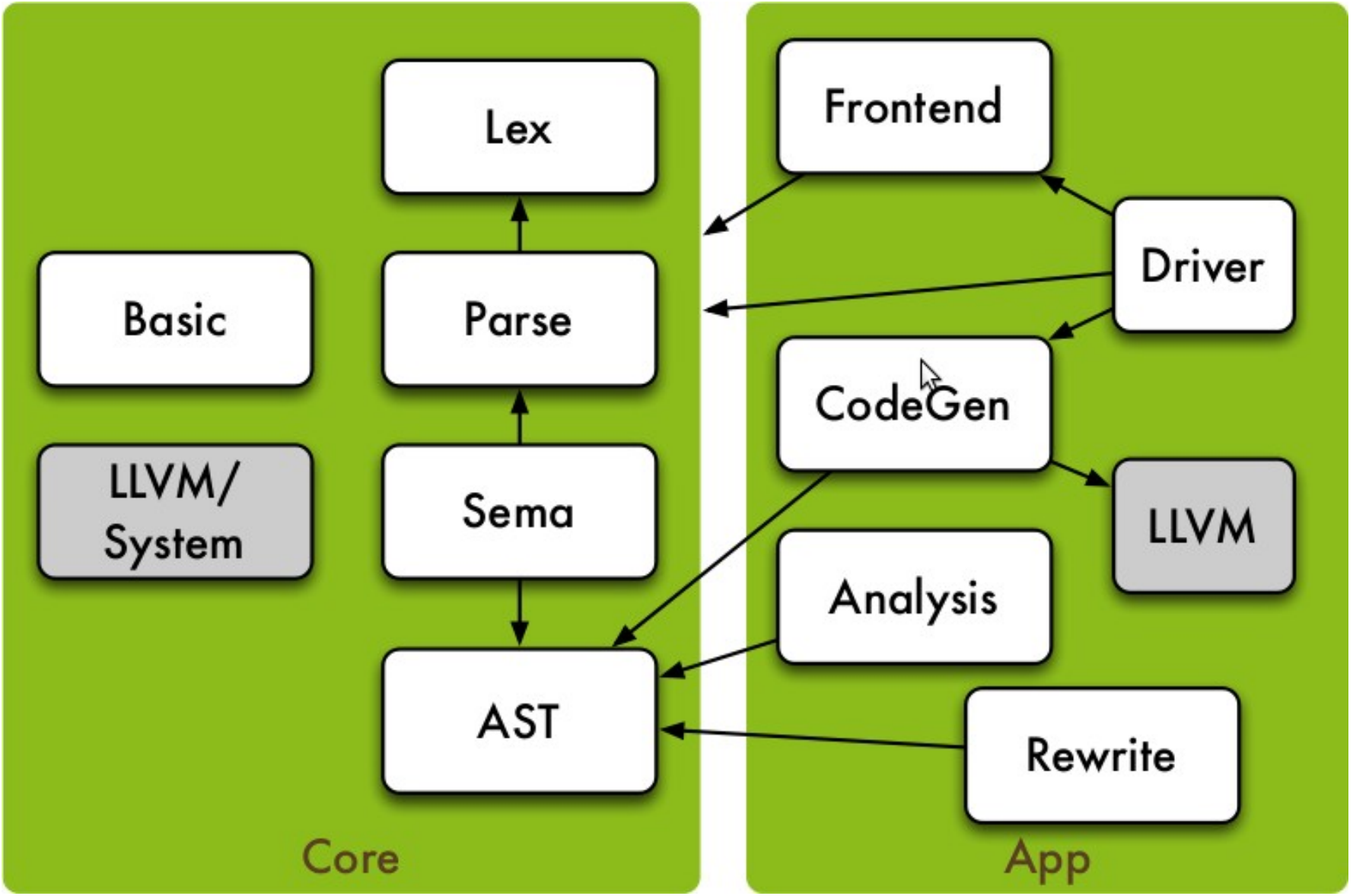
Zend

RenderScript (Android)

LLVM 可作為上述的編譯器應用的根基

LLVM

- Low-Level VM → bit-code
- 完整的編譯器基礎建設
  - 可重用的、用以建構編譯器的軟體元件
  - 允許更快更完整的打造新的編譯器
  - static compiler, JIT, trace-based optimizer, ...
- 開放的編譯器框架
  - 多種程式語言支援
  - 高彈性的自由軟體授權模式 (BSD License)
  - 活躍的開發 (Apple + Google + Qualcomm)
  - 豐富的編譯輸出：C, ARM, x86, PowerPC, ...


Lex
Frontend
Basic
Parse
Driver
CodeGen
LLVM/
System
Sema
LLVM
Analysis
AST
Rewrite
Core
App

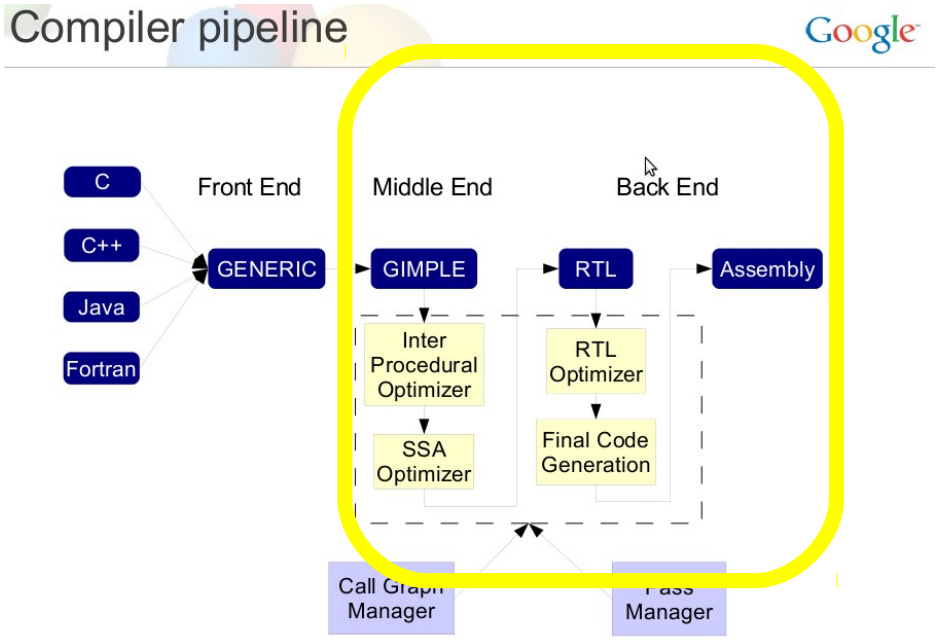
GCC vs. LLVM

GCC

C, C++, Obj-C, Fortran, Java, Ada, ...

x86, ARM, MIPS, PowerPC, ...

binutils (ld as)



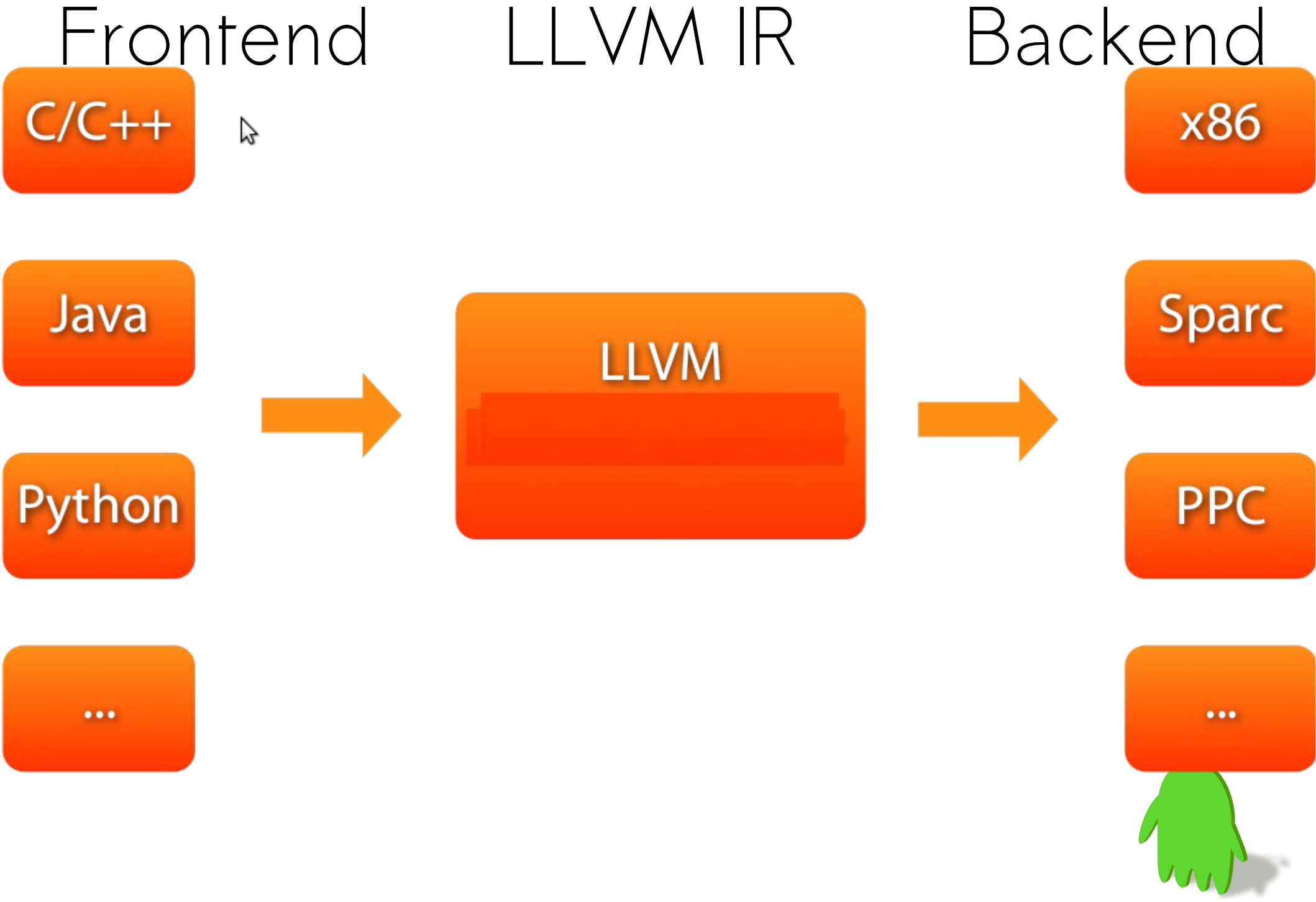
Compiler Driver

LLVM

C, C++, Obj-C

BSD-Style License

JIT/Interpreter


Frontend
LLVM IR
Backend
C/C++
Java
Python
...
LLVM
x86
Sparc
PPC
...

Frontend LLVM IR Backend

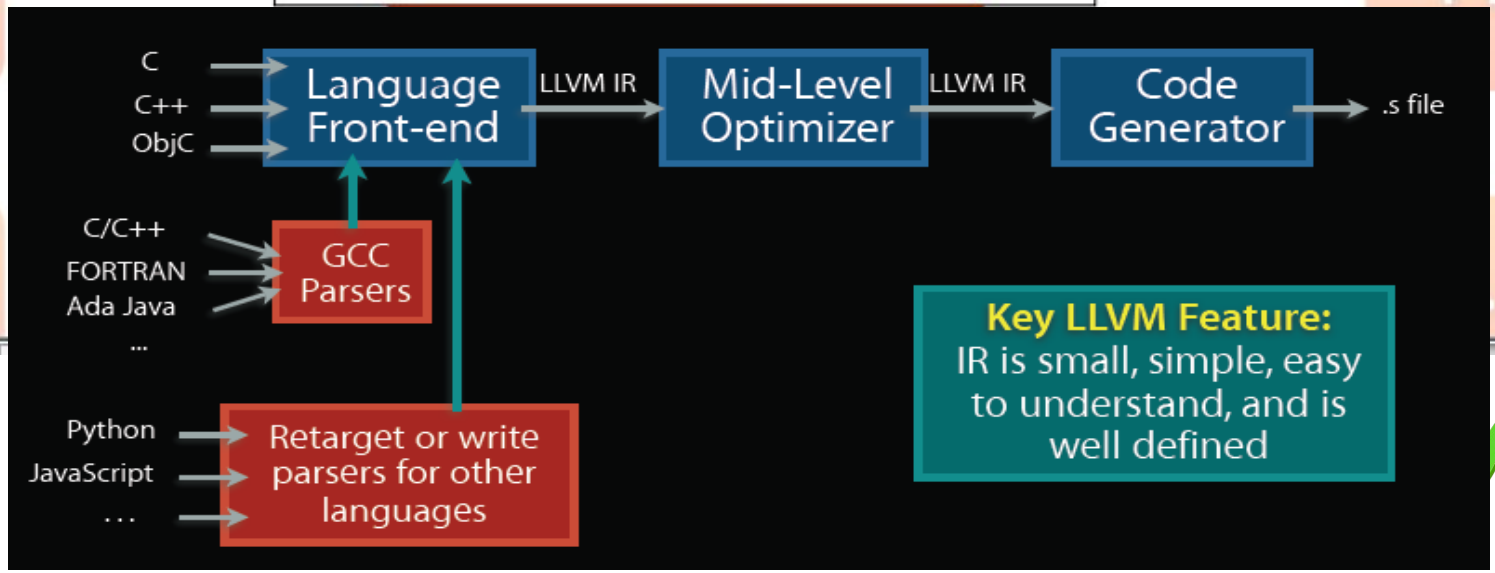
C/C++

x86

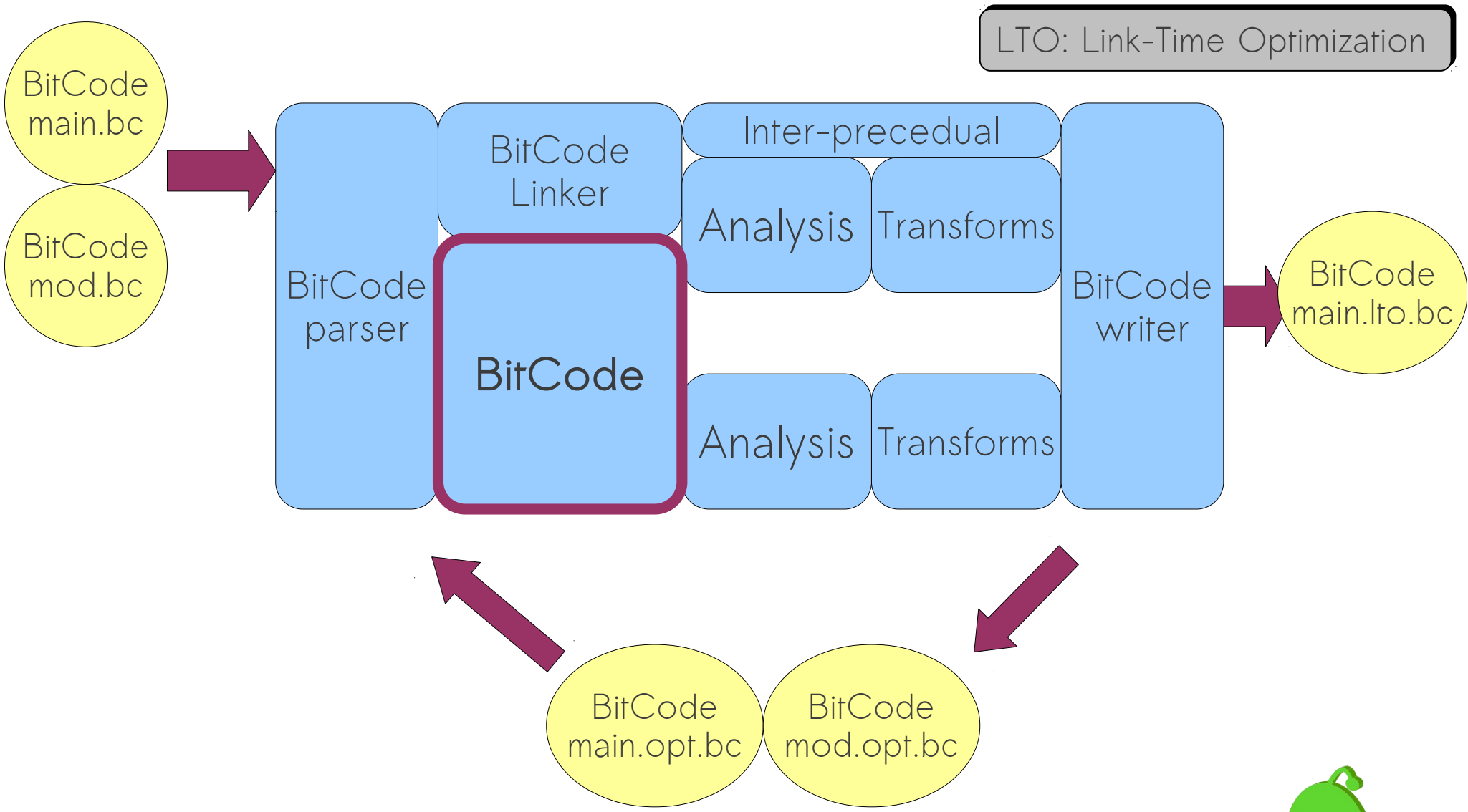
```
int add_func(
  int a, int b)
{
  return a + b;
}
```

```
define i32 @add_func(i32 %a, i32 %b) {
entry:
     %tmp3 = add i32 %b, %a
     ret i32 %tmp3
}
```

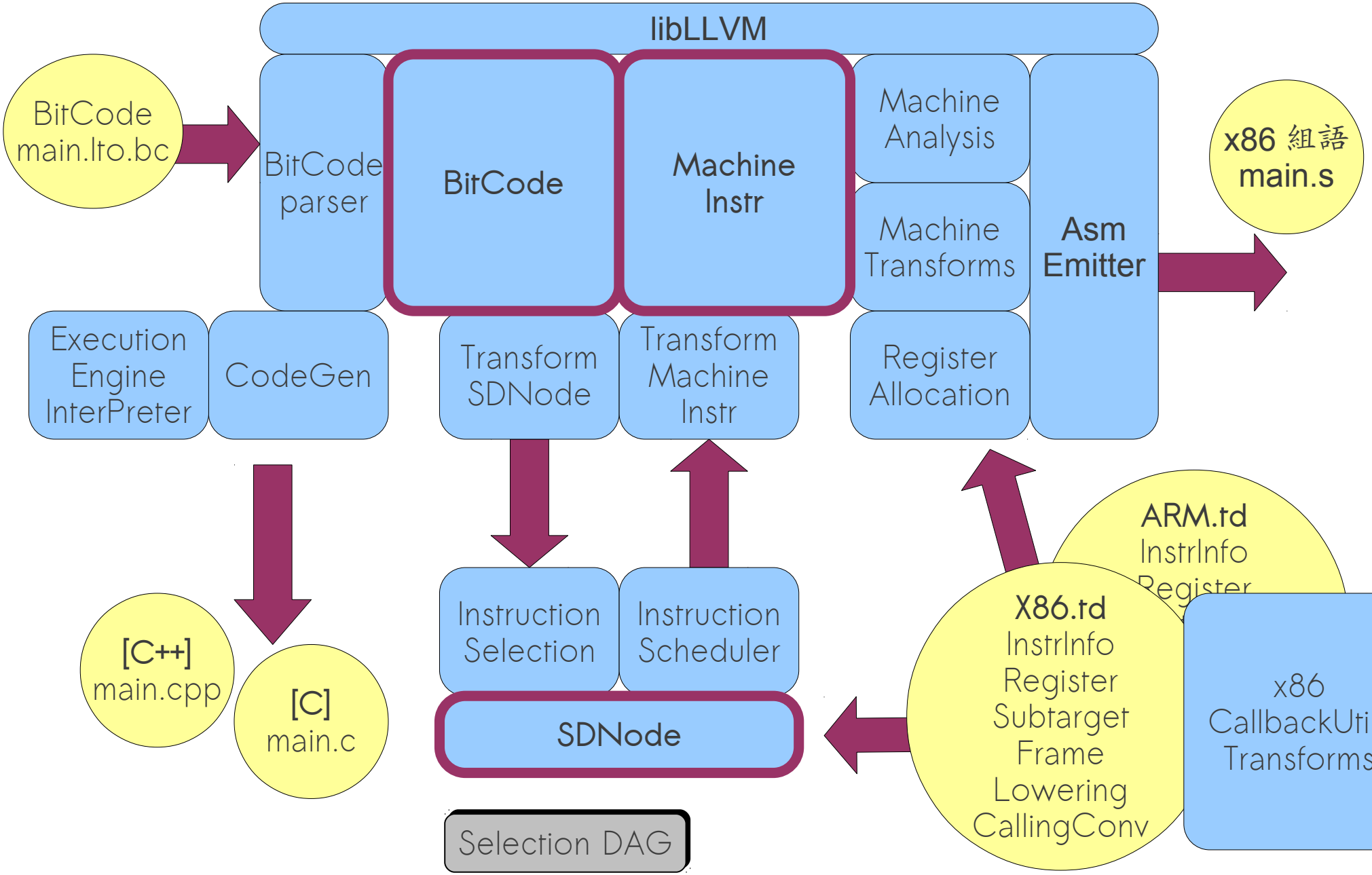
```
_add_func:
  movl  8(%esp), %eax
  addl  4(%esp), %eax
    ret
```



BitCode + Optimizer



LLVM Code Generation



先從 Hello World 開始

- 完整的 Compiler Driver

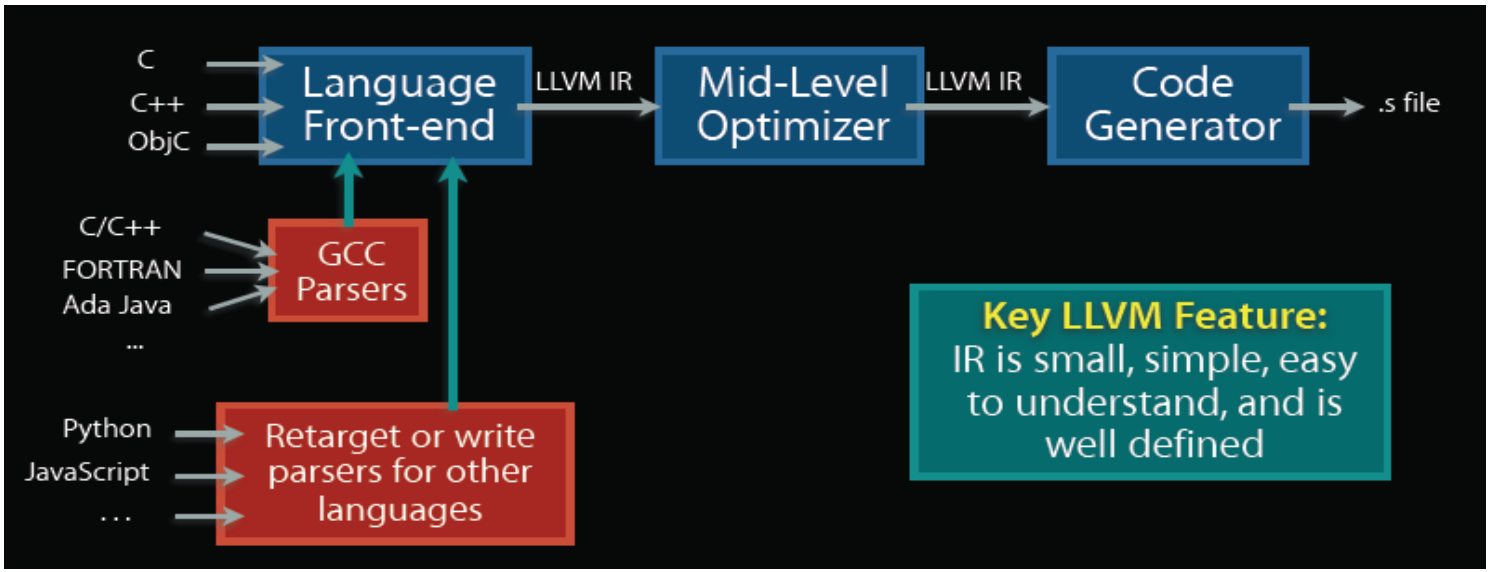
```
$ clang hello.c -o hello
```

- 生成 IR

```
$ clang -O3 -emit-llvm hello.c -c -o hello.bc
```

- 以 Just-In-Time compiler 模式執行 BitCode

```
$ lli hello.bc
```



Getting Started with the LLVM System
http://llvm.org/docs/GettingStarted.html

```c
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("Hello world!\n");
    return 0;
}
```

函式 printf() 後方僅有一個字串參數，前端預設將其轉換爲 puts()

- 反組譯 BitCode

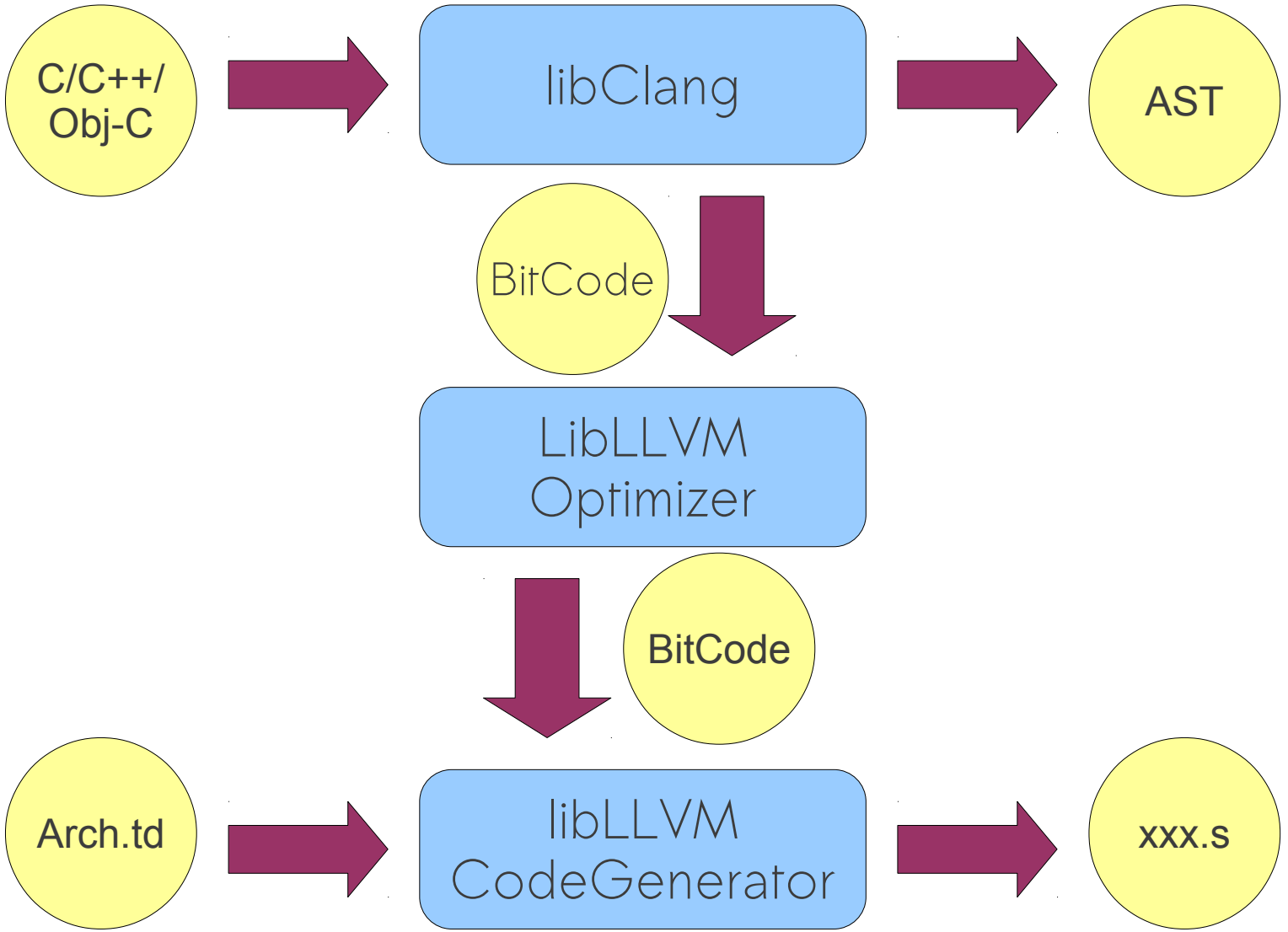
```
$ llvm-dis < hello.bc
```

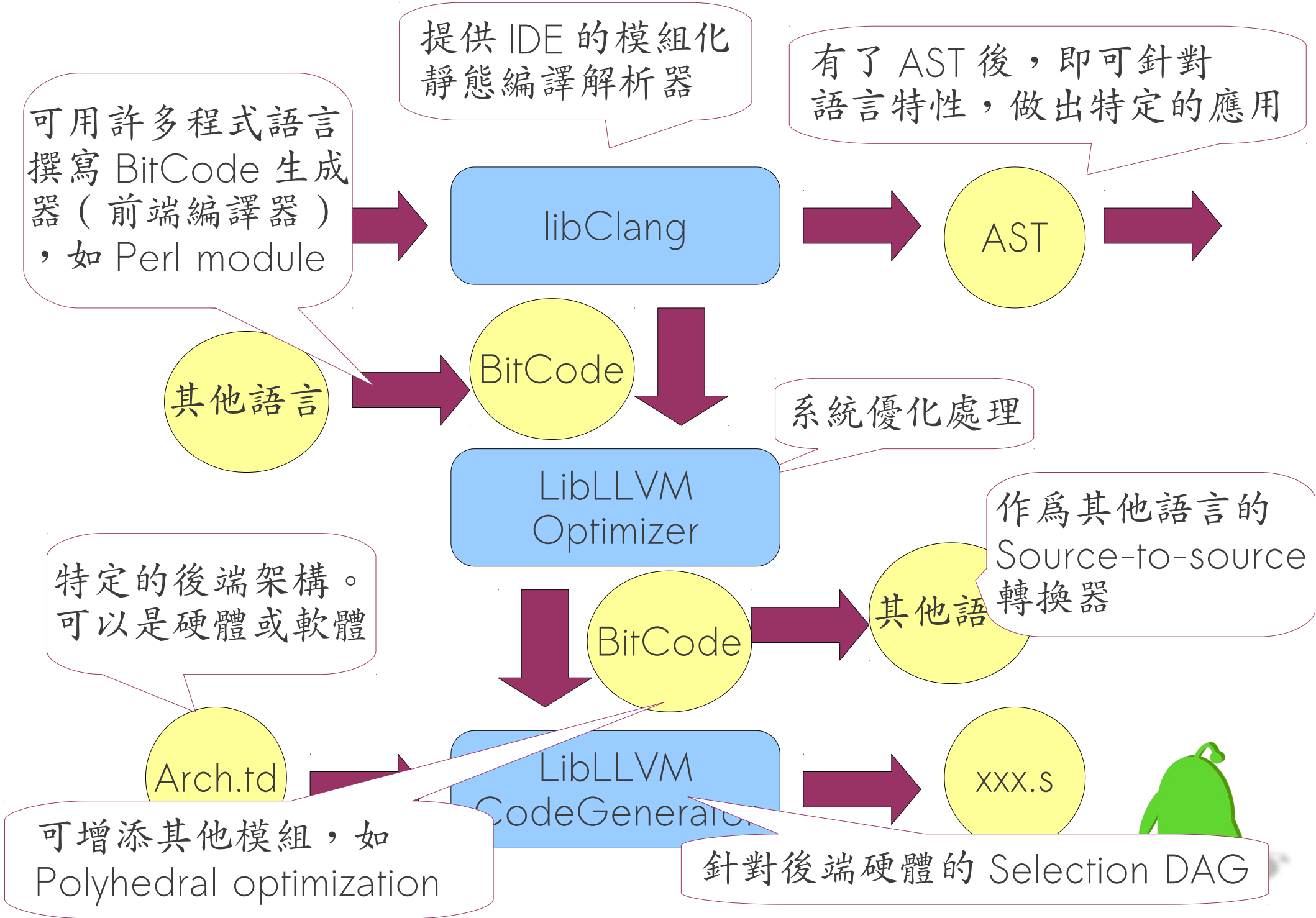
```
; ModuleID = '<stdin>'
target datalayout = "e-p:32:32:32-...
target triple = "i386-pc-linux-gnu"

@str = internal constant [13 x i8] c"Hello world!\00"

define i32 @main(i32 %argc, i8** nocapture %argv) nounwind {
entry:
  %puts = tail call i32 @puts(i8* getelementptr inbounds ([13 x i8]* @str, i32 0, i32 0))
  ret i32 0
}

Declare i32 @puts(i8* nocapture) nounwind
```

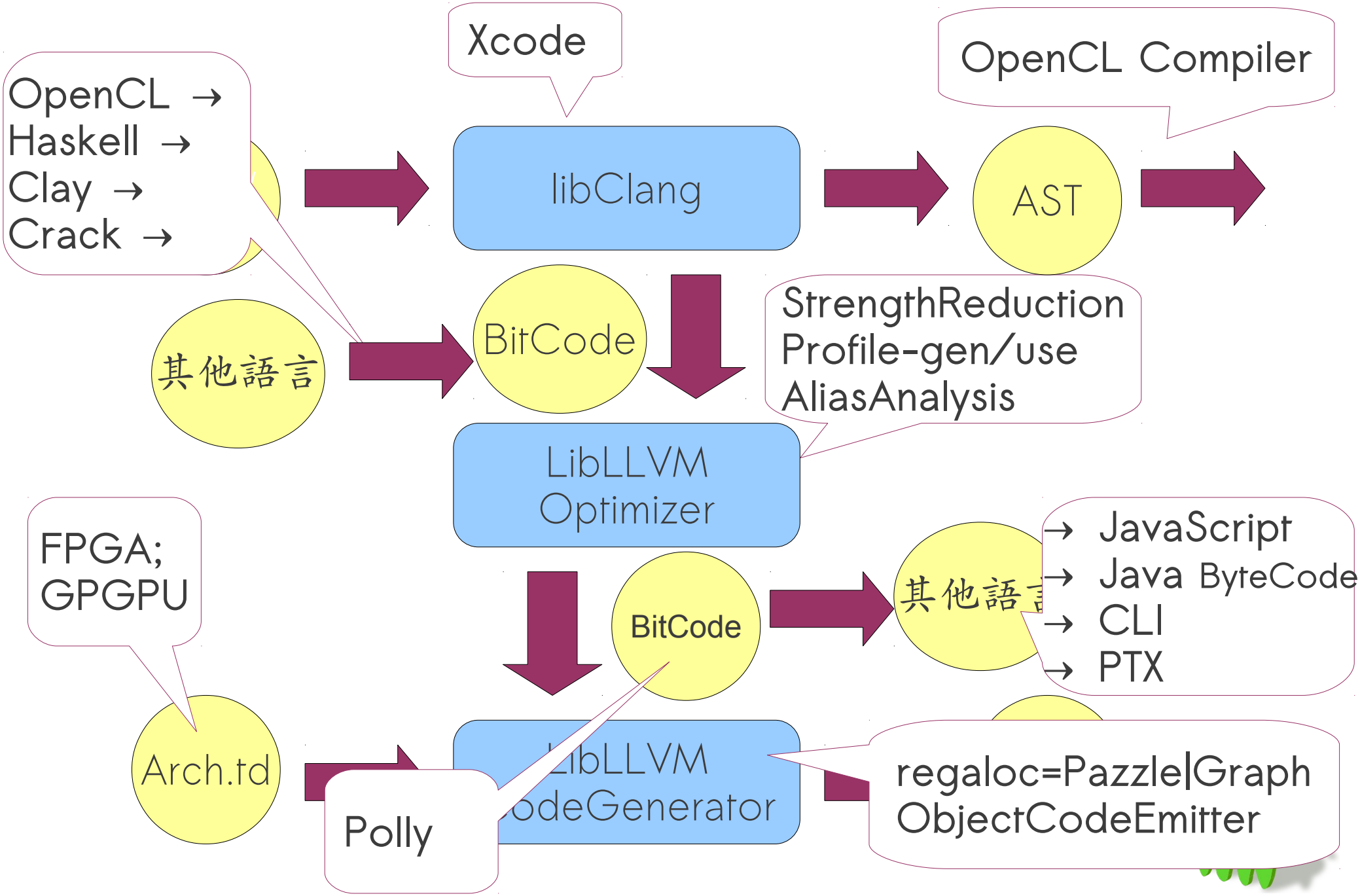
- 輸出 x86 後端組合語言

```
$ llc hello.bc -o hello.s
```

LLVM 給予無限可能

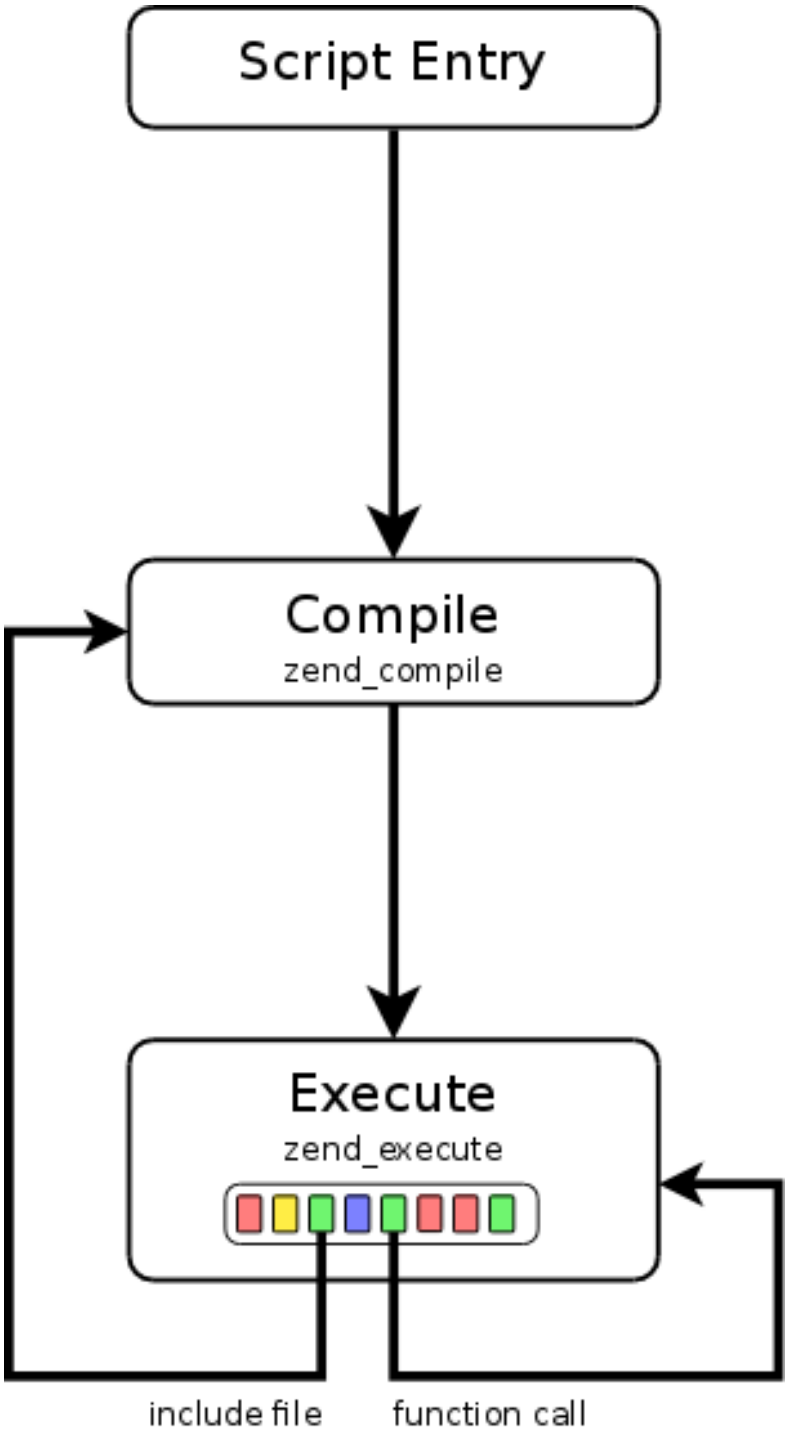

C/C++/
Obj-C
libClang
AST
BitCode
LibLLVM
Optimizer
BitCode
Arch.td
libLLVM
CodeGenerator
xxx.s


提供 IDE 的模組化
靜態編譯解析器
有了 AST 後，即可針對
語言特性，做出特定的應用
可用許多程式語言
撰寫 BitCode 生成
器（前端編譯器）
，如 Perl module
libClang
AST
其他語言
BitCode
系統優化處理
LibLLVM
Optimizer
作爲其他語言的
Source-to-source
轉換器
特定的後端架構。
可以是硬體或軟體
BitCode
其他語
Arch.td
LibLLVM
CodeGenerator
xxx.s
可增添其他模組，如
Polyhedral optimization
針對後端硬體的 Selection DAG


Xcode
OpenCL Compiler
OpenCL →
Haskell →
Clay →
Crack →
libClang
AST
其他語言
BitCode
StrengthReduction
Profile-gen/use
AliasAnalysis
LibLLVM
Optimizer
FPGA;
GPGPU
BitCode
→ JavaScript
→ Java ByteCode
→ CLI
→ PTX
Arch.td
Polly
regaloc=Pazzle|Graph
ObjectCodeEmitter

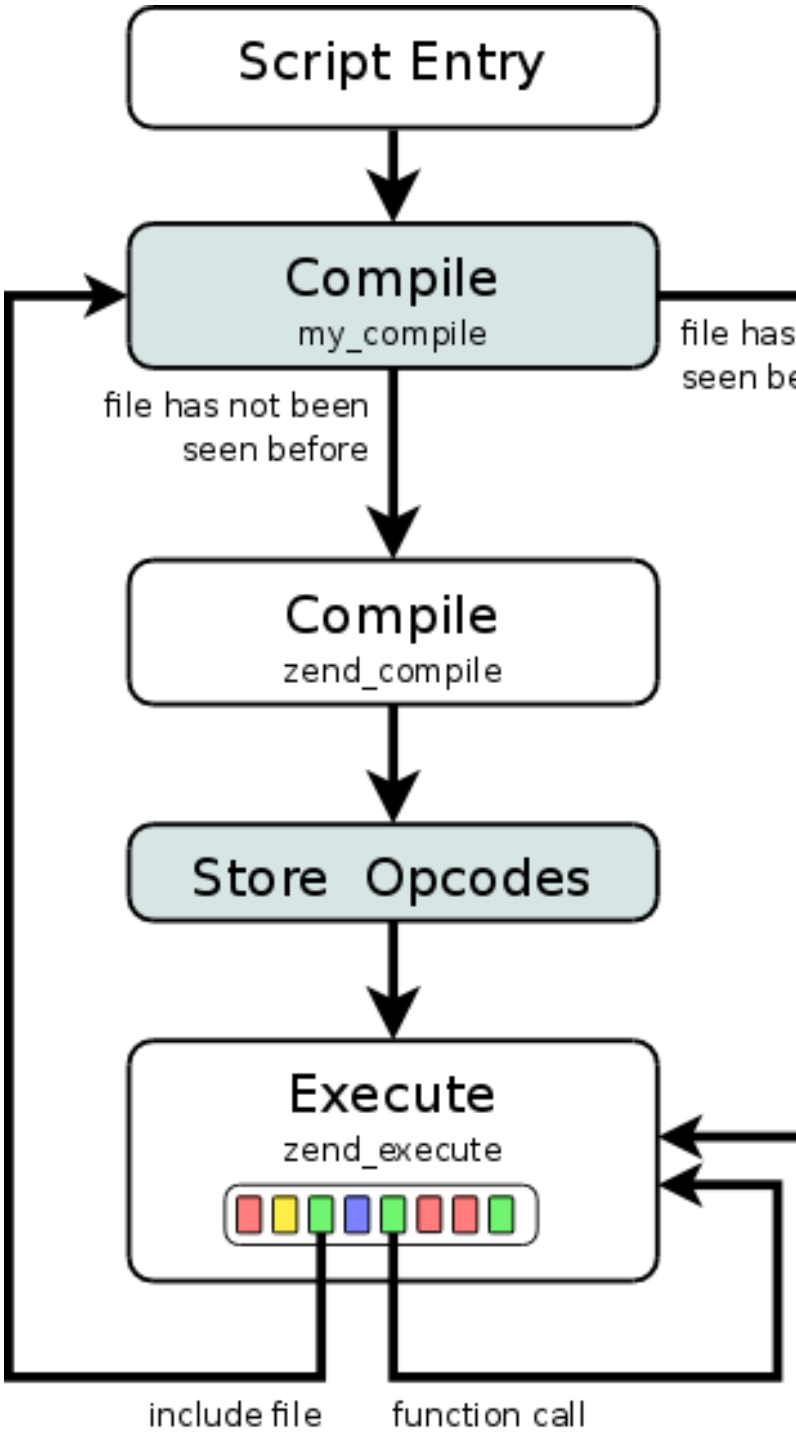
整合 PHP 到 LLVM 架構

Zend VM 概況 (1)



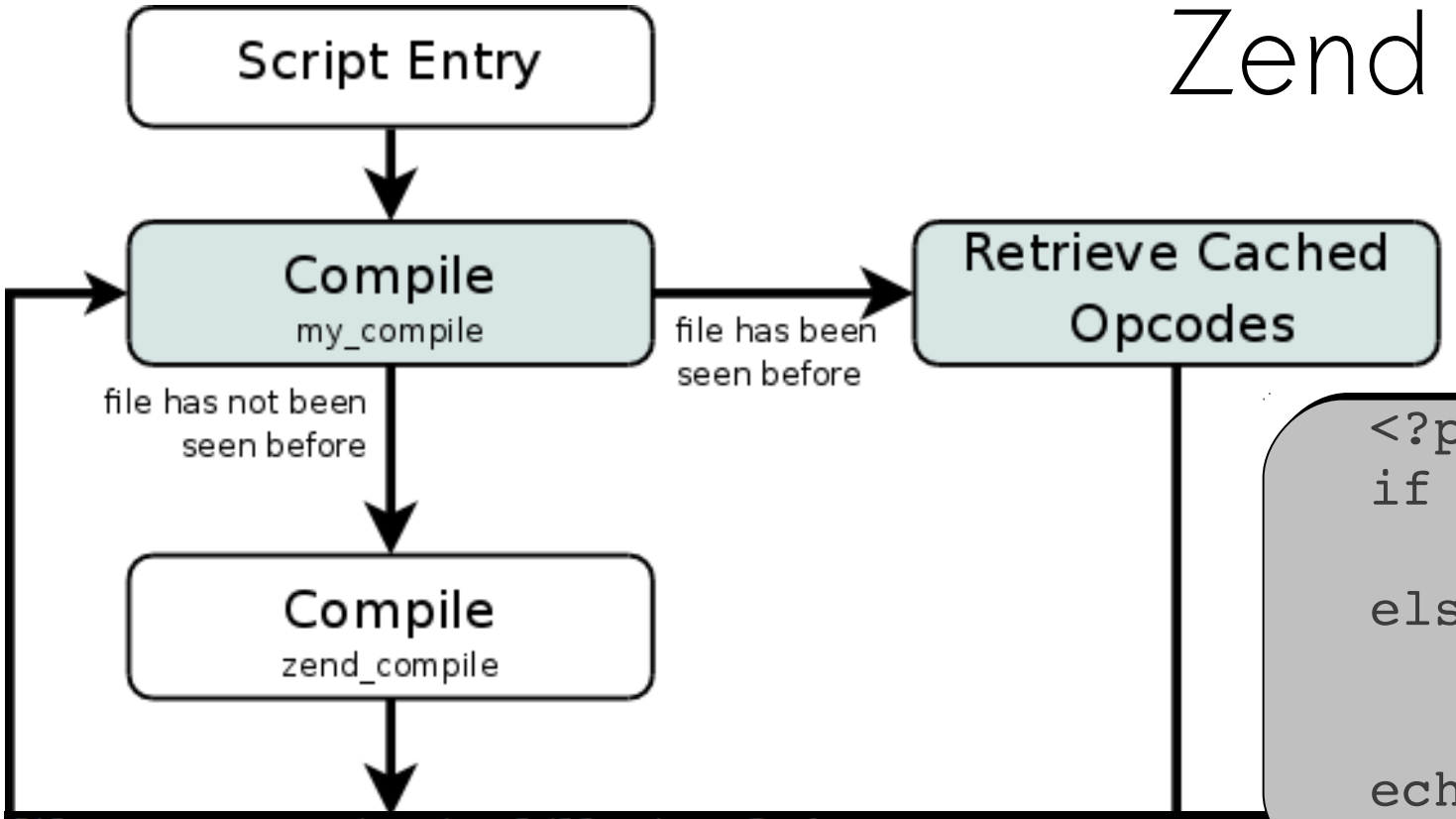
- 依循傳統 SDT (Syntax-Directed Translation) 編譯器設計方法建構
- 解譯 bytecode
- 欠缺進階的編譯器優化機制

Zend VM 概況 (2)



- PHP bytecode 使用單純的表示法，無強制的標準
- 部份資訊如 class 定義，並未包含在 bytecode

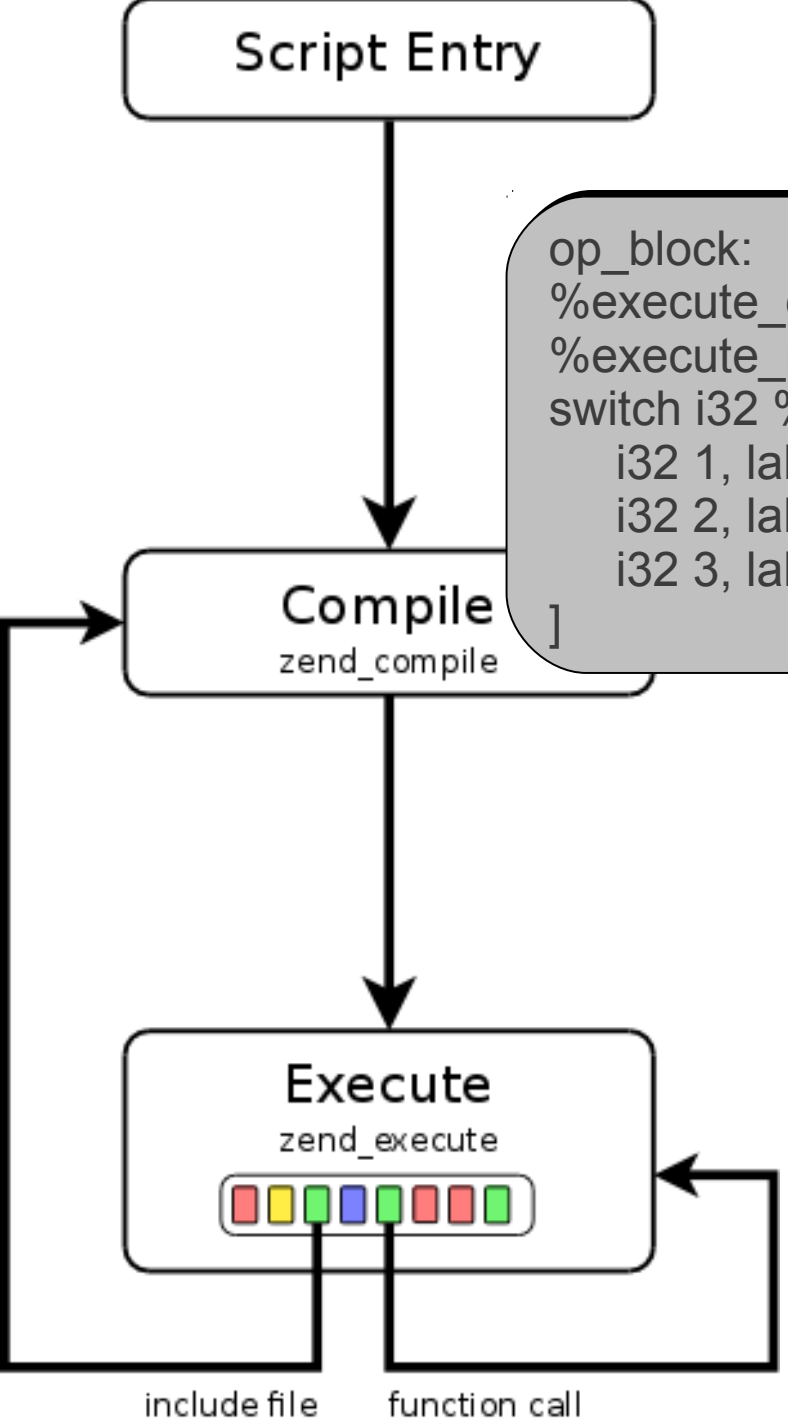
Zend VM 概況 (3)




```
<?php
if (1 > 2)
        $a = 2 * 3;
else
        $a = 2 * 4;

echo $a;
?>
```

```
filename:       /cvs/pecl/llvm/test2.php
function name:  (null)
number of ops:  9
compiled vars:  !0 = $a
line     #  op                           fetch          ext  return  operands
---------------------------------------------------------------------------
   3     0  IS_SMALLER                                       ~0      2, 1
         1  JMPZ                                                    ~0, ->5
   4     2  MUL                                               ~1      2, 3
         3  ASSIGN                                                  !0, ~1
   5     4  JMP                                                     ->7
   6     5  MUL                                               ~3      2, 4
         6  ASSIGN                                                  !0, ~3
   9     7  ECHO                                                    !0
  12     8  RETURN                                                  1
```

Zend + LLVM




```
op_block:
%execute_data = call @phpllvm_get_execute_data(%1)
%execute_result = call @ZEND_IS_SMALLER_HANDLER(%execute_data)
switch i32 %execute_result, label %op_block1 [
    i32 1, label %pre_vm_return
    i32 2, label %pre_vm_enter
    i32 3, label %pre_vm_leave
]
```

```
<?php
if (1 > 2)
        $a = 2 * 3;
else
        $a = 2 * 4;

echo $a;
?>
```

```
filename:       /cvs/pecl/llvm/test2
function name:  (null)
number of ops:  9
compiled vars:  !0 = $a
line     #  op                           fetch          ext  return  operands
-------------------------------------------------------------------------------
   3     0  IS_SMALLER                                        ~0      2, 1
         1  JMPZ                                                 ~0, ->5
   4     2  MUL                                                  ~1      2, 3
         3  ASSIGN                                               !0, ~1
   5     4  JMP                                                  ->7
   6     5  MUL                                                  ~3      2, 4
         6  ASSIGN                                               !0, ~3
   9     7  ECHO                                                 !0
  12     8  RETURN                                               1
```

PHP + LLVM

- PECL::LLVM → Zend bytecode to LLVM assembly compiler, unmaintained

  http://pecl.php.net/package/llvm

  Compiles Zend bytecode to LLVM assembly and then into optimized machine code.
- Roadsend PHP / Raven
  - Static analyzer which dumps (in XML)
  - Tokens / AST / Generated IR
  - Various analysis passes
  - Port of phc optimizer passes

PHP + LLVM

- HipHop for PHP 僅能執行小部份的 PHP 程式，而且無法有效佈署 (deployment; 輸出爲 POSIX C++ code)
- PHP 不該侷限於伺服器端的應用
- 開發工具的整合
- 透過 LLVM 的 Polly 與 LTO (Link-Time Optimization)，可進一步優化 Zend bytecode → LLVM bitcode 的執行
- memcache / MP / GPGPU

架構於 LLVM 的程式語言實做 (1)

- **Unladen Swallow** (Google): faster Python

  $ ./perf.py -r -b call_simple --args "-j always," \
  ../q2/python ../q3/python

  - Min: 1.618273 -> 0.908331: 78.16% faster
  - Avg: 1.632256 -> 0.924890: 76.48% faster

  http://code.google.com/p/unladen-swallow

- **GHC**/Haskell's LLVM codegen
  - 3x faster in some cases
    http://donsbot.wordpress.com/2010/02/21/smoking-fast-haskell-code-using-ghcs-new-llvm-codegen/
- **LLVM-Lua** : JIT/static Lua compiler
  - http://code.google.com/p/llvm-lua/

架構於 LLVM 的程式語言實做 (2)

- IcedTea Version of Sun's OpenJDK (RedHat)
  - Zero: processor-independent layer that allows OpenJDK to build and run using any processor
  - **Shark**: Zero's JIT compiler: uses LLVM to provide native code generation without introducing processor-dependent code.

    http://icedtea.classpath.org
- **Emscripten**
  - LLVM-to-JavaScript compiler
  - It takes LLVM bitcode and compiles that into JavaScript, which can be run on the web (or anywhere else JavaScript can run).

    http://code.google.com/p/emscripten/

http://0xlab.org