Dyncom Powered By **LLVM**

Fast ISS
native code generator

模拟器解决的问题

- 系统软件有开发周期长，开发成本高，系统复杂而缺少有效调试方法的特点。
- 基于仿真平台的开发具有如下优点：
  - 在硬件没有完成之前，可以基于仿真软件进行开发
  - 仿真软件相比于硬件系统具有可维护性，相对的低成本。
  - 仿真软件可以任意时刻查看任意外设的任意状态。在仿真平台上可复现的bug，可以稳定复现。
- 风河公司宣称：未来的电子产品开发大部分都会基于虚拟平台进行开发

模拟器种类

开源软件

- Qemu：目前最流行的开源的仿真平台
- Bochs：较流行的x86仿真平台
- 国外的商业硬件仿真产品：
  - 风河公司的Simics产品
  - Coware公司的仿真软件
  - Imperas公司的ovp仿真软件。

常见的指令模拟方式

- 解释执行：

  对每条指令进行实时的解释执行，并不缓存解释过的指令.比如：Bochs

- 动态翻译：

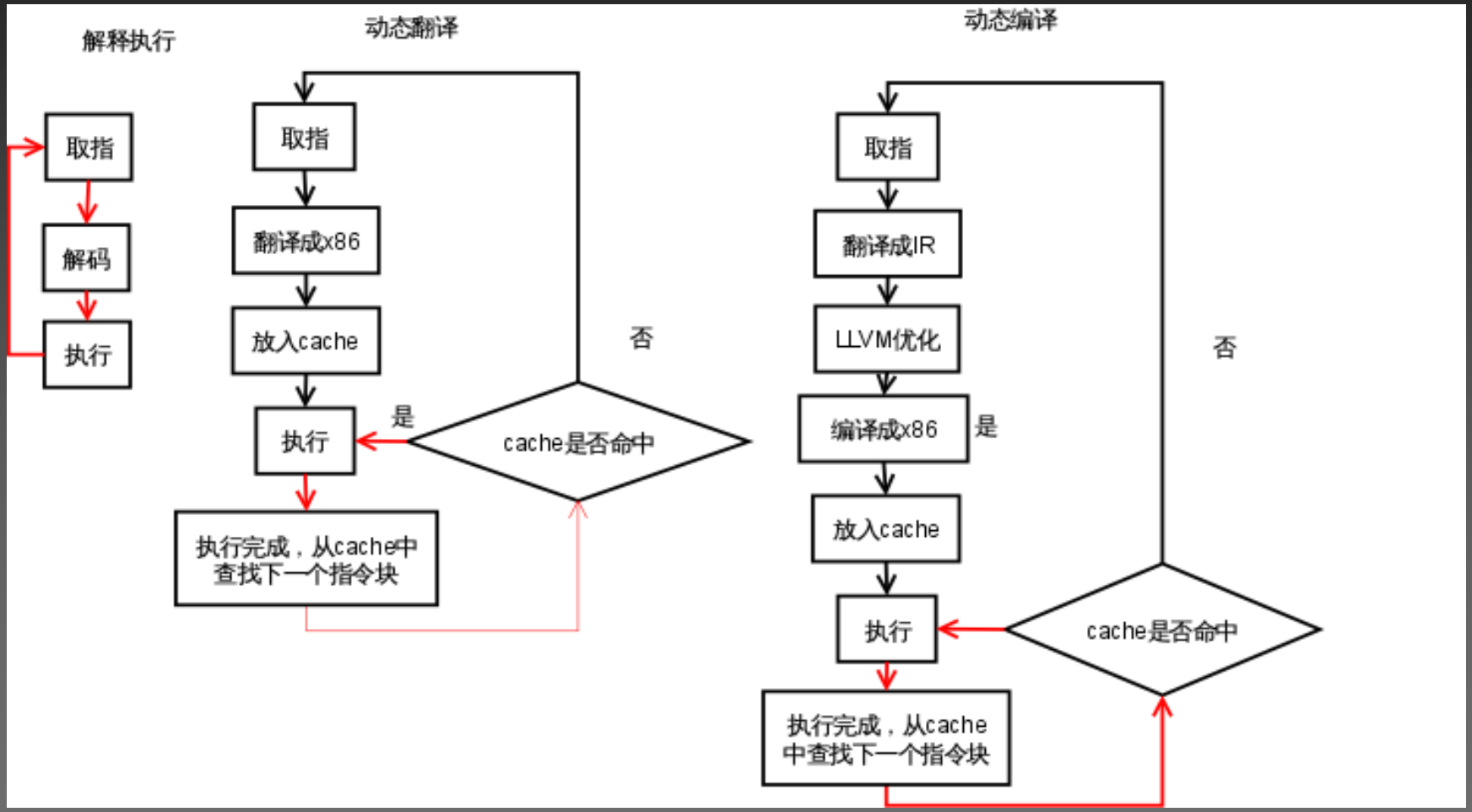
  把每条Guest机器指令直接翻译成Host机器代码

  比如：qemu的dyngen, skyeye的dbct

- 动态编译：

  在运行过程中先把Guest指令翻译成中间表示，然后翻译成Host指令。比如TCG，Dyncom

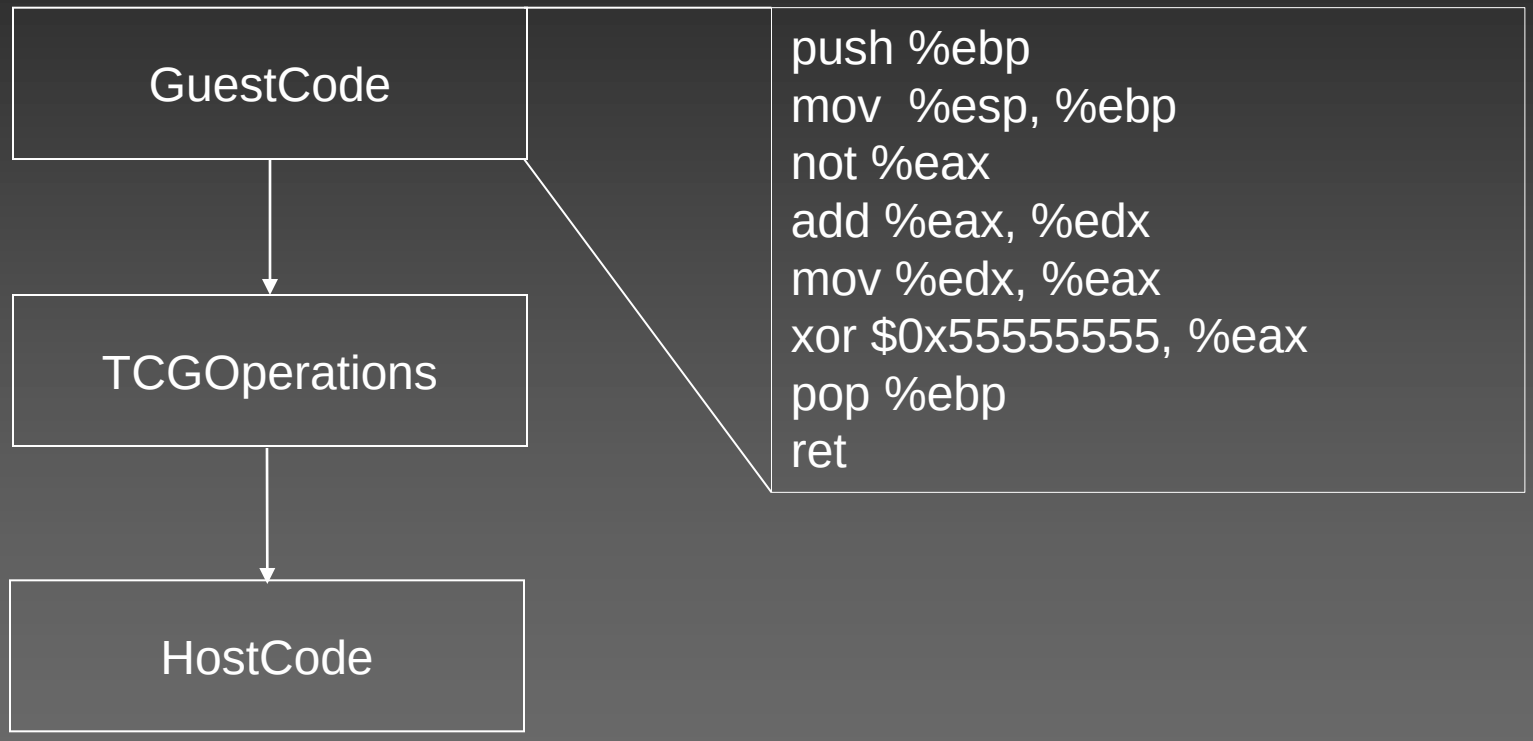
指令翻译的过程


解释执行
取指
解码
执行
动态翻译
取指
翻译成x86
放入cache
执行
是
cache是否命中
否
执行完成，从cache中查找下一个指令块
动态编译
取指
翻译成IR
LLVM优化
编译成x86
是
放入cache
执行
cache是否命中
否
执行完成，从cache中查找下一个指令块

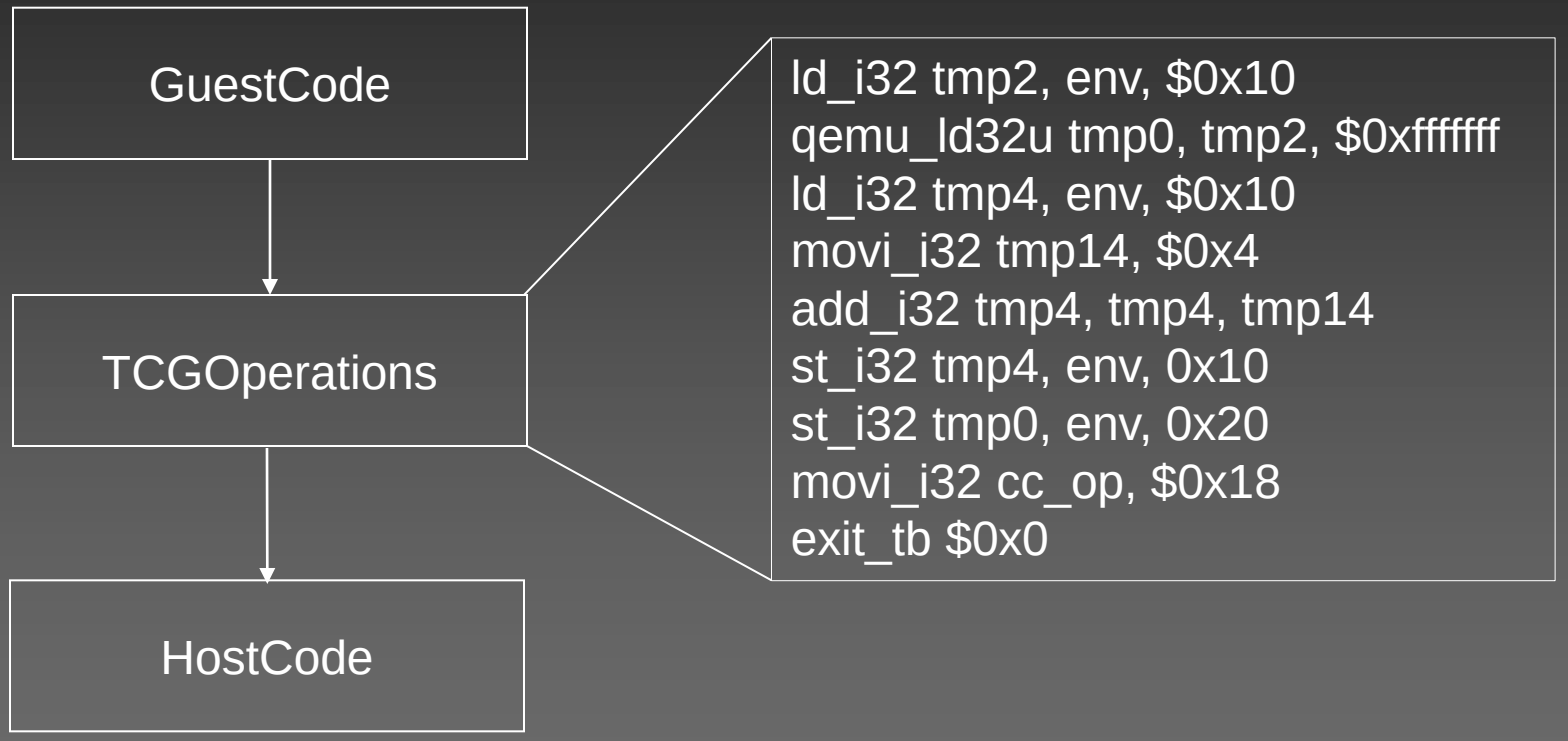
TCG

- TCG IR
- 轻量级的代码生成引擎，翻译速度快
- 寄存器生存期分析只是基本块级别
- 没有做更多代码上的优化
- 一次翻译以TranslationBlock为单位，一般是一个Guest Code基本块

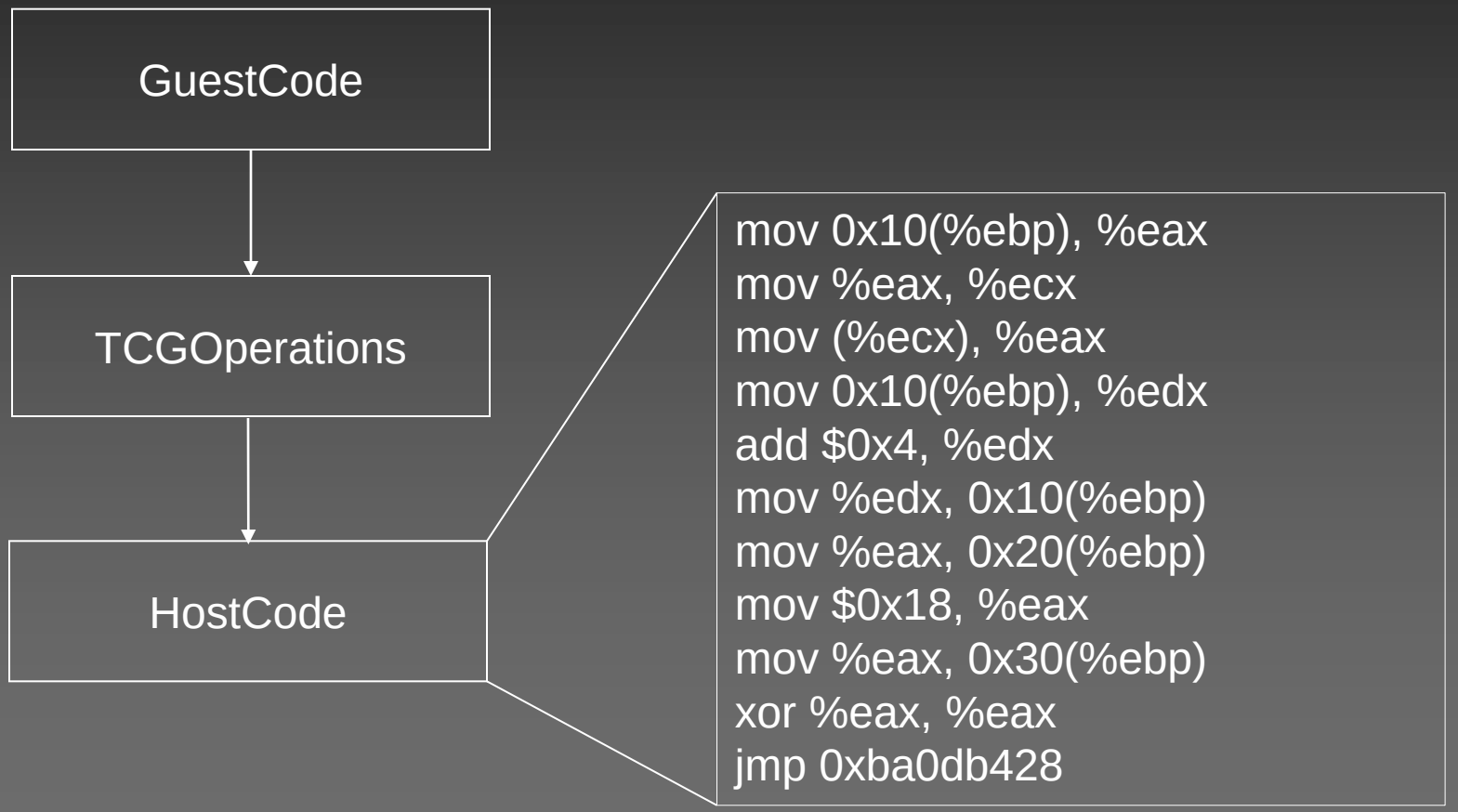
TCG


GuestCode
TCGOperations
HostCode
push %ebp
mov %esp, %ebp
not %eax
add %eax, %edx
mov %edx, %eax
xor $0x55555555, %eax
pop %ebp
ret

TCG


GuestCode
TCGOperations
HostCode
ld_i32 tmp2, env, $0x10
qemu_ld32u tmp0, tmp2, $0xffffffff
ld_i32 tmp4, env, $0x10
movi_i32 tmp14, $0x4
add_i32 tmp4, tmp4, tmp14
st_i32 tmp4, env, 0x10
st_i32 tmp0, env, 0x20
movi_i32 cc_op, $0x18
exit_tb $0x0

TCG


GuestCode
TCGOperations
HostCode
mov 0x10(%ebp), %eax
mov %eax, %ecx
mov (%ecx), %eax
mov 0x10(%ebp), %edx
add $0x4, %edx
mov %edx, 0x10(%ebp)
mov %eax, 0x20(%ebp)
mov $0x18, %eax
mov %eax, 0x30(%ebp)
xor %eax, %eax
jmp 0xba0db428

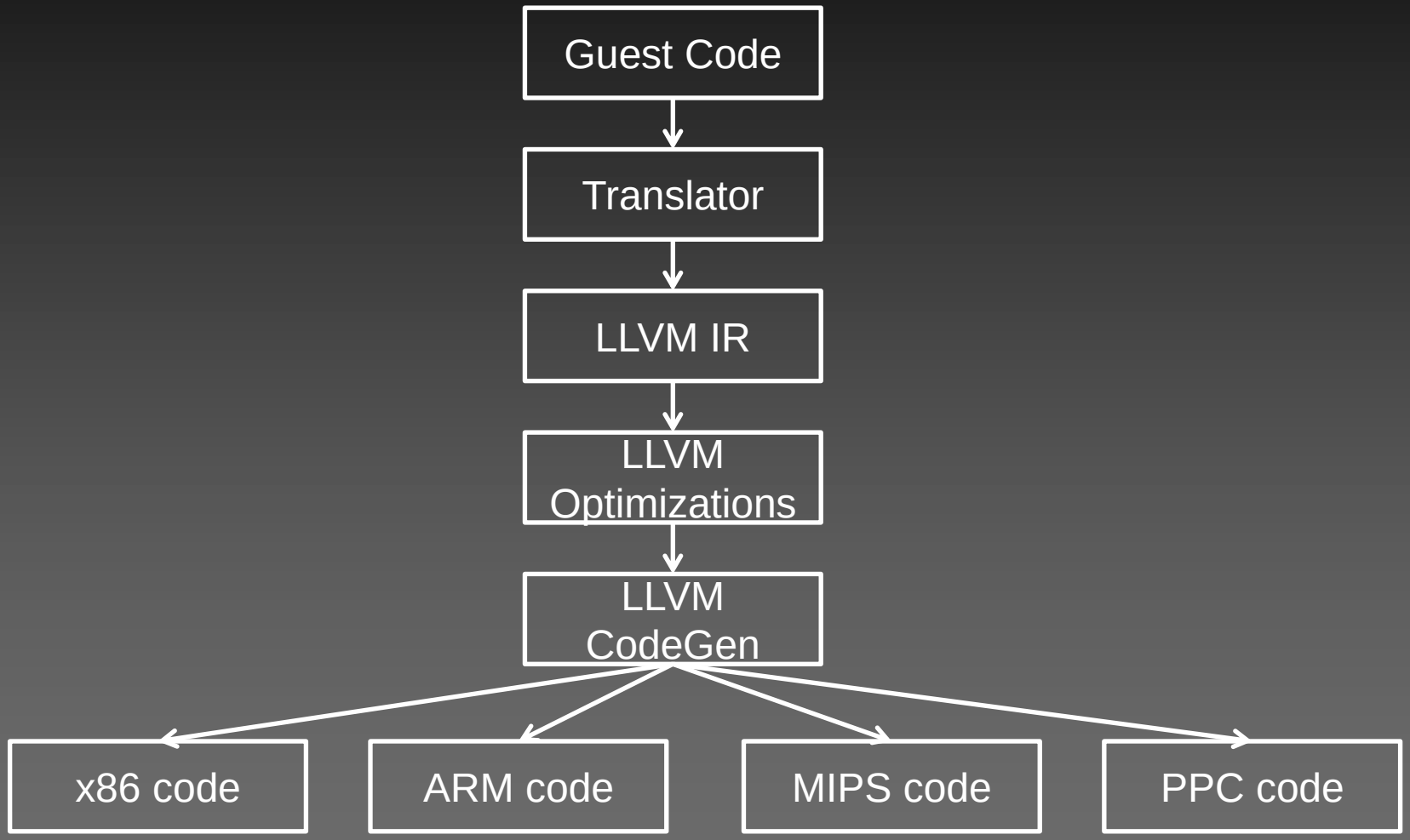
LLVM

- Low Level Virtual Machine
- 主要实现了中间语言和后端代码生成部分
  可以生成多种体系结构的二进制代码
- 很多高级语言都基于LLVM做了前端部分，
  性能得到了很大的提高

  Python, JavaScript, Haskell…etc.

Dyncom

- LLVM IR
- 重量级的代码生成引擎，翻译速度稍慢
- 利用LLVM已有的优化生成高质量的Host代码
- 可以为自己模拟的Arch写单独的优化Pass
- 一次翻译以Function为单位，包含若干个Guest Code的基本块

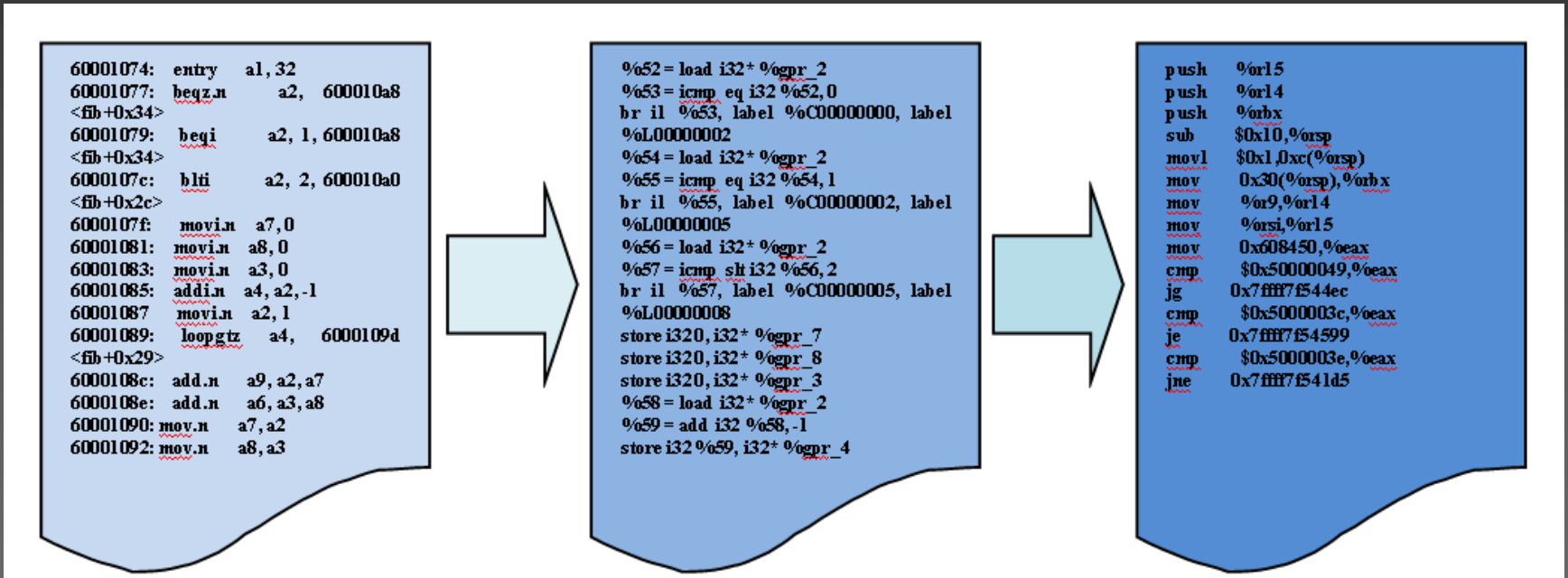

Guest Code
Translator
LLVM IR
LLVM Optimizations
LLVM CodeGen
x86 code
ARM code
MIPS code
PPC code

Dyncom

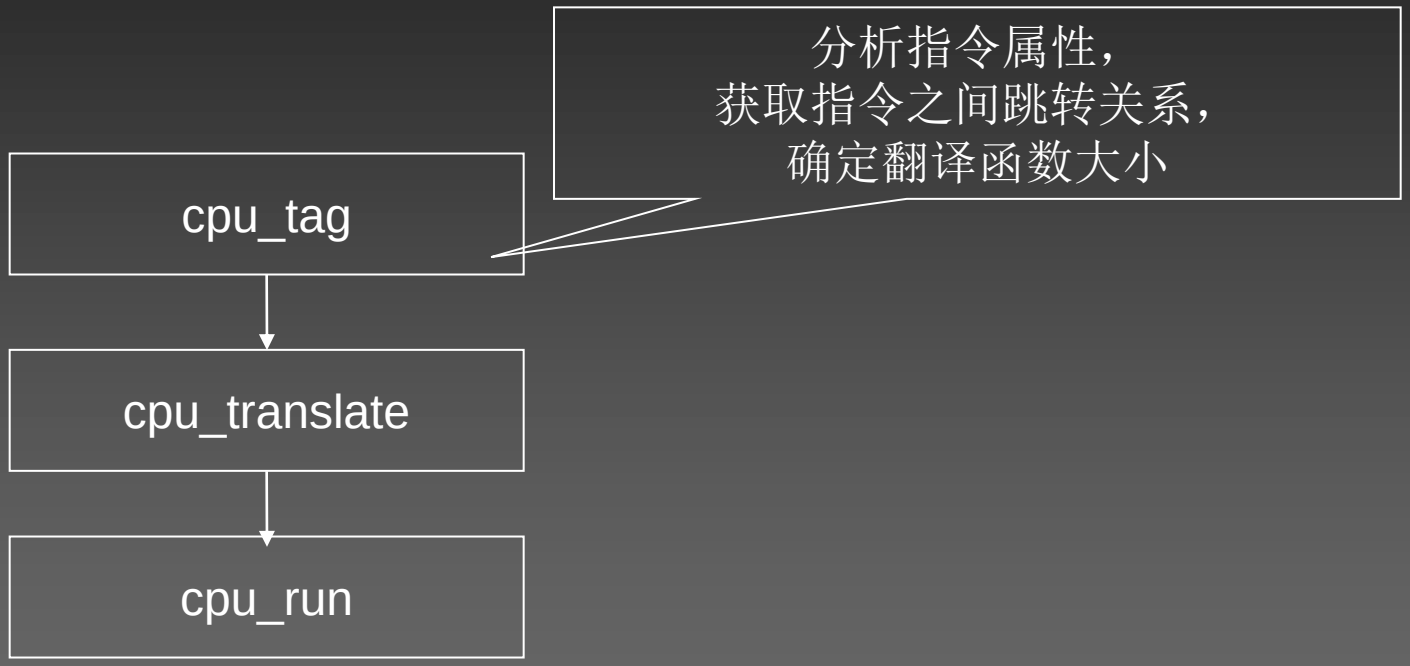
GuestCode

LLVM IR

HostCode

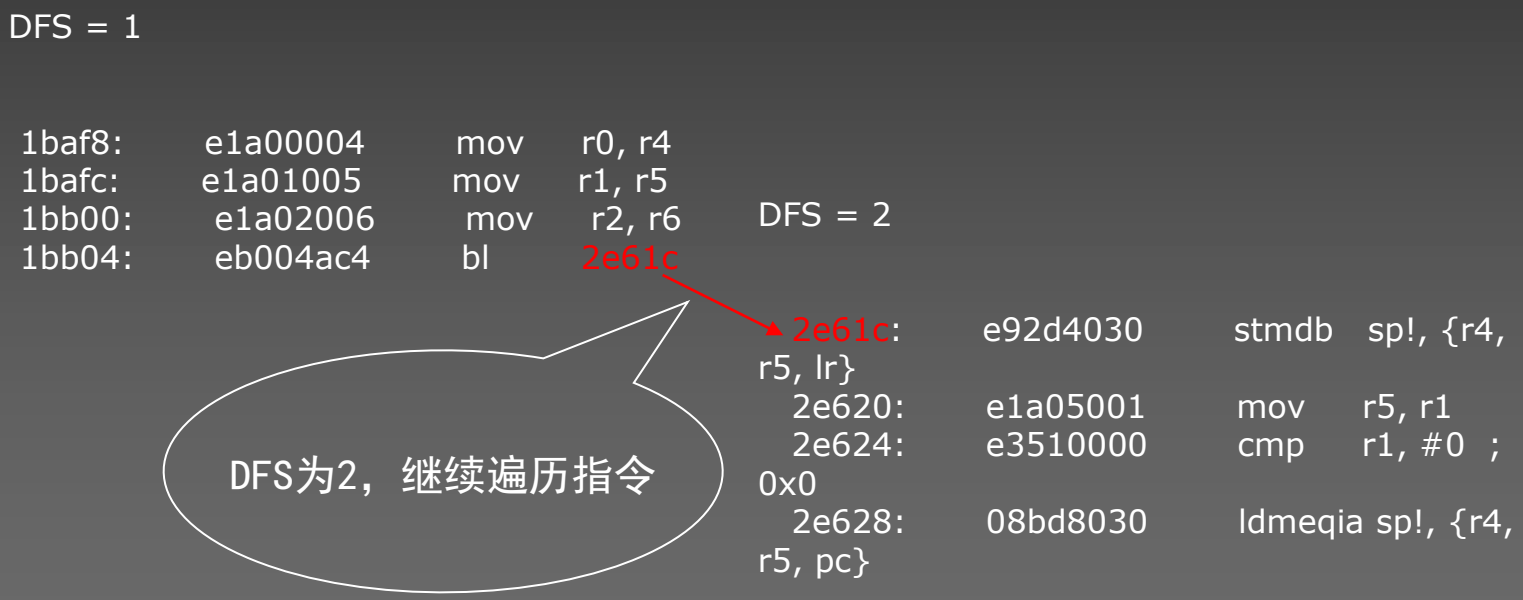


Dyncom运行流程

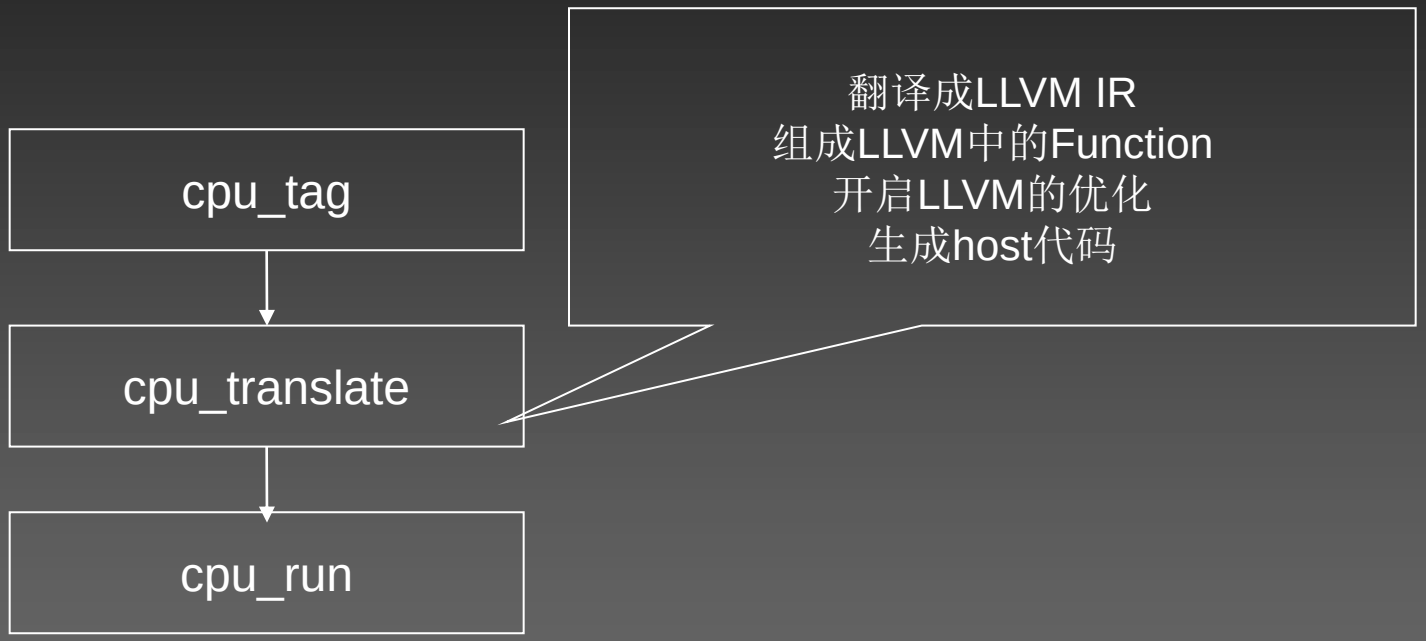


Dyncom翻译函数大小

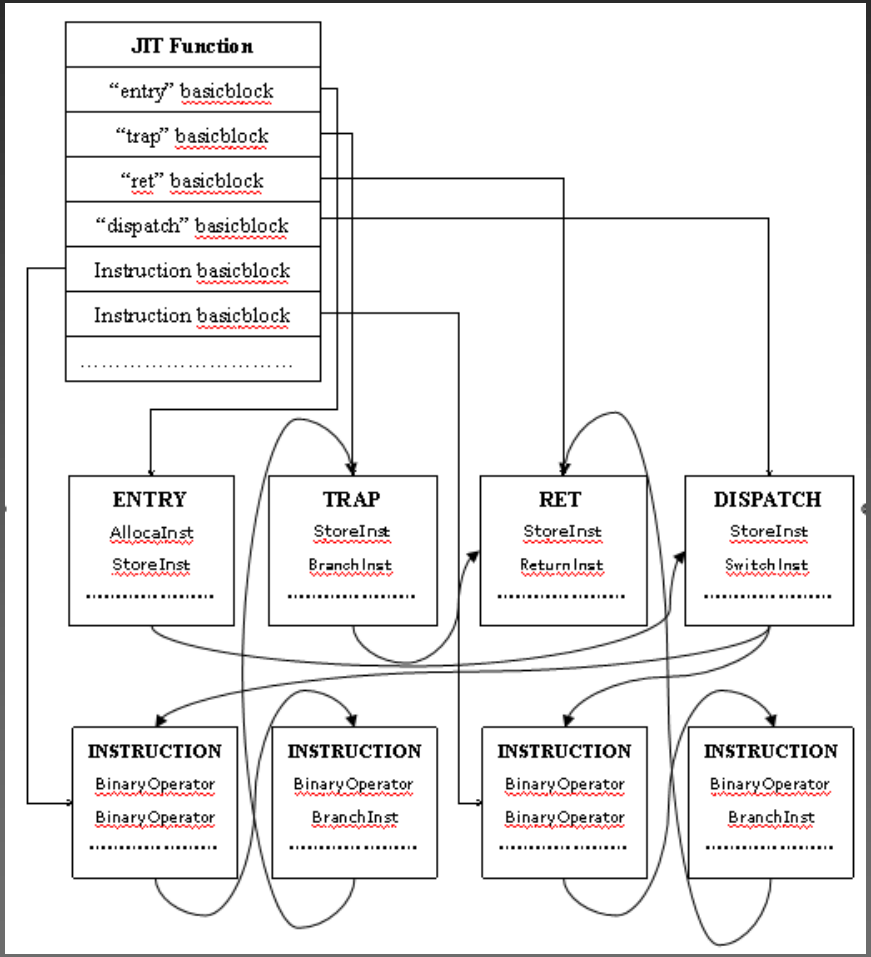
- 利用深度优先遍历的参数来决定函数大小



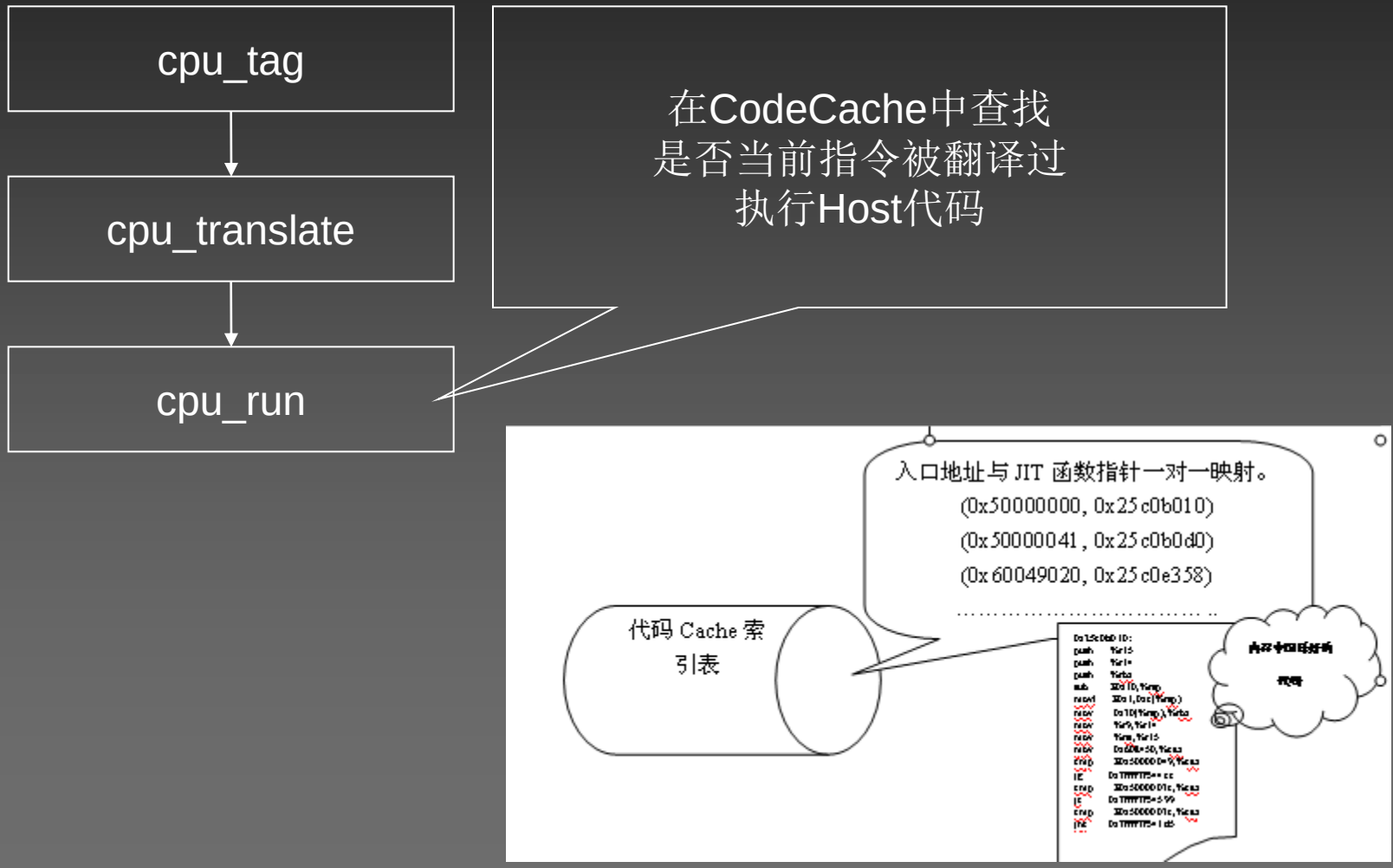
Dyncom运行流程


翻译成LLVM IR
组成LLVM中的Function
开启LLVM的优化
生成host代码
cpu_tag
cpu_translate
cpu_run

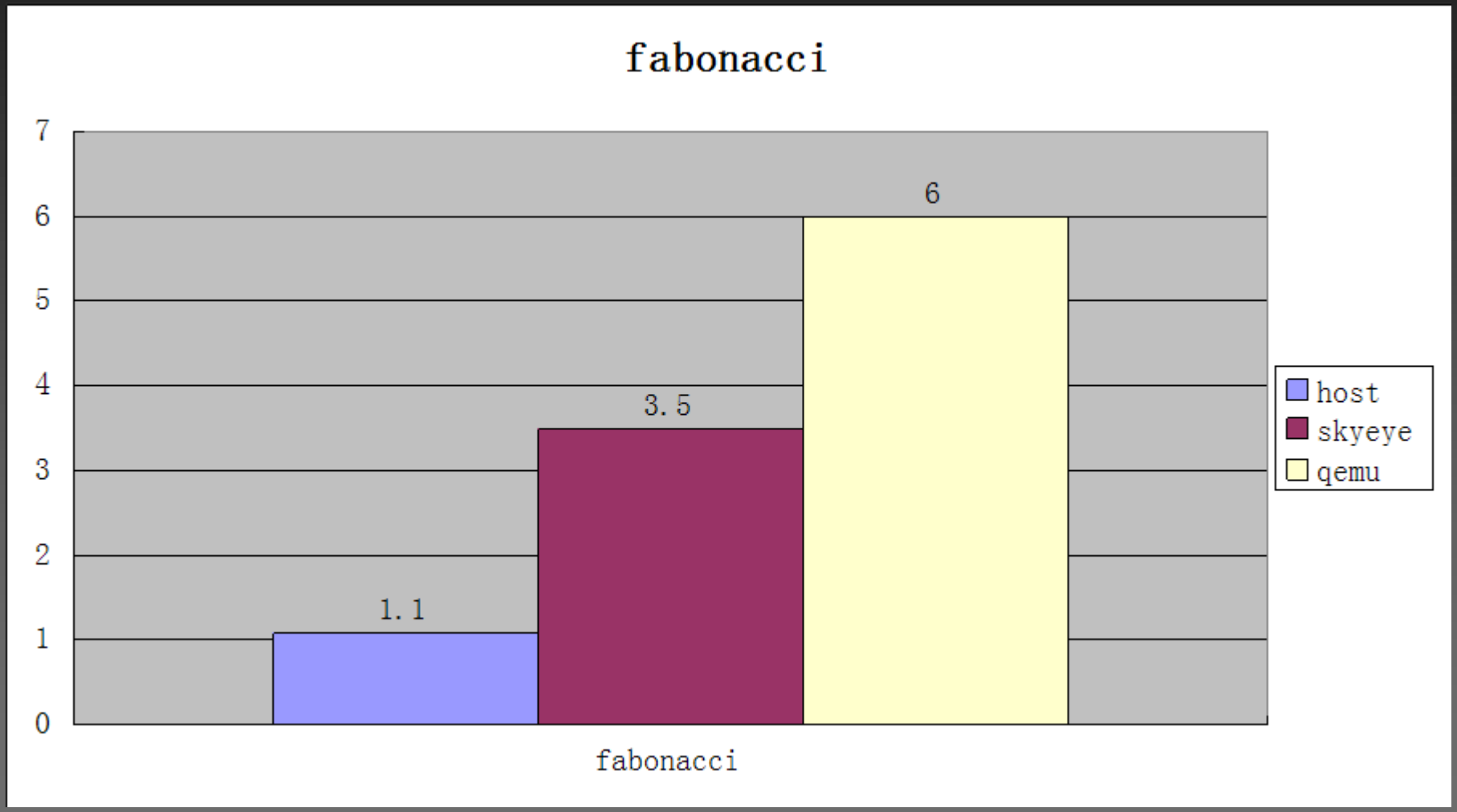
JIT Function结构



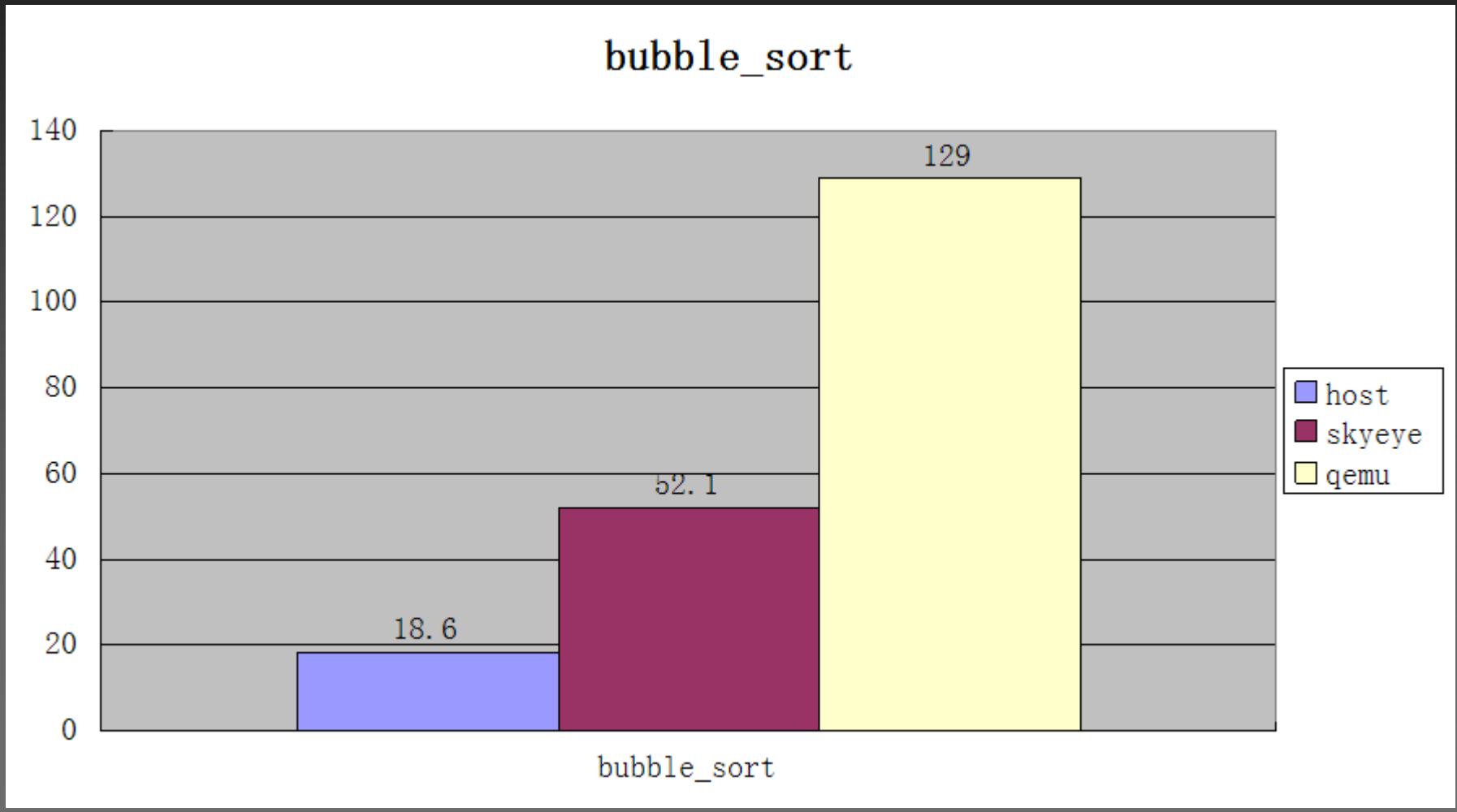
DYNCOM运行流程


cpu_tag
cpu_translate
cpu_run
在CodeCache中查找
是否当前指令被翻译过
执行Host代码
入口地址与 JIT 函数指针一对一映射。
(0x50000000, 0x25c0b010)
(0x50000041, 0x25c0b0d0)
(0x60049020, 0x25c0e358)
代码 Cache 索
引表

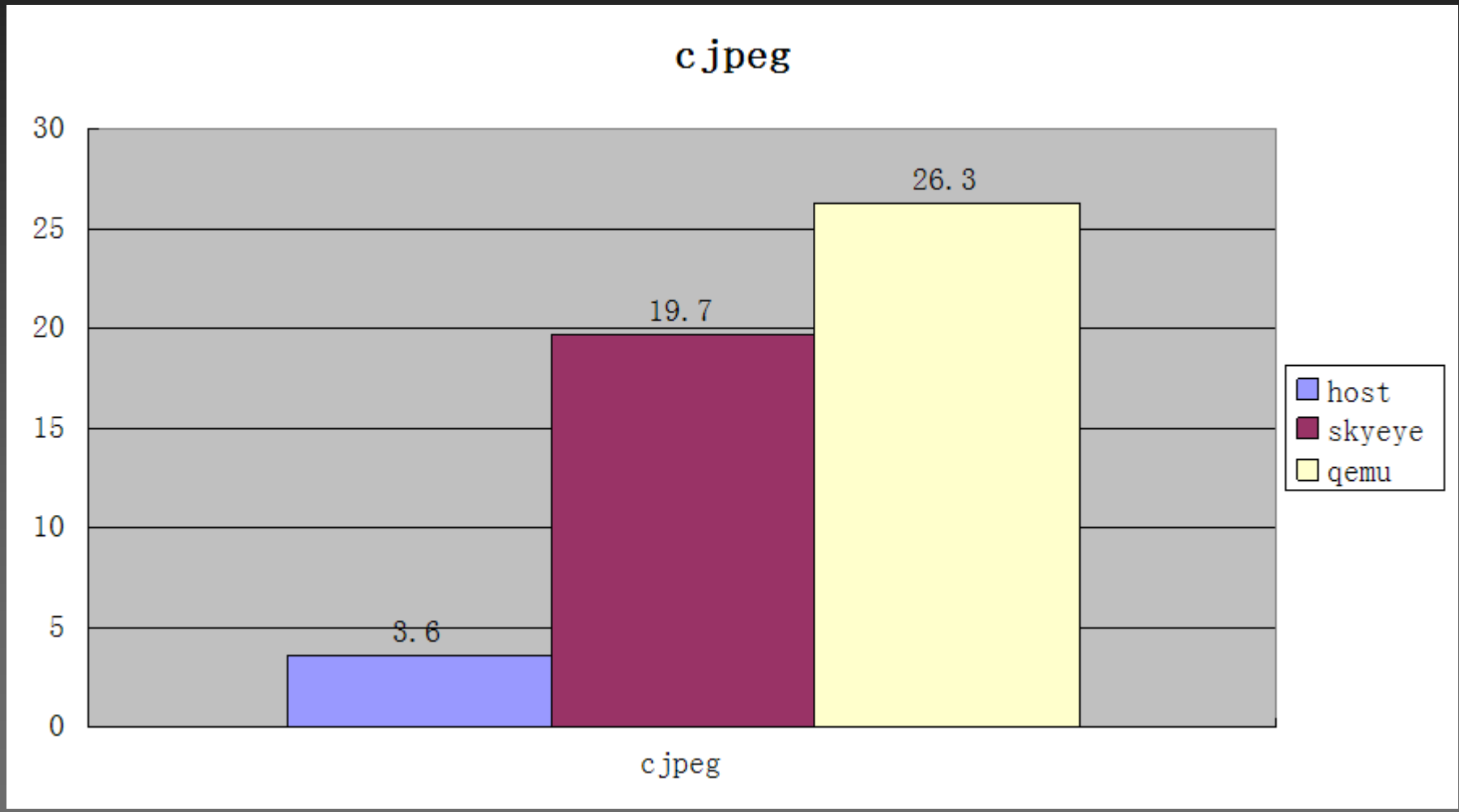
Benchmark



Benchmark


bubble_sort
140
120
100
80
60
40
20
0
129
52.1
18.6
host
skyeye
qemu
bubble_sort

Benchmark



- 多平台性
- 难度低
- 开发工作量少
- 内存占用多
- 编译时间长
- 代码生成质量高

Question?