CHINA LINUX KERNEL
中国Linux内核开发者大会

vivo

基于龙架构的系统级二进制翻译器

牛根

2018 - 2024：中科院计算所直博

2024 - ：龙芯中科工程师

CHINA LINUX KERNEL
中国Linux内核开发者大会

vivo

目录

CONTENTS

01 二进制翻译简介

什么是二进制翻译？

- 将一种架构的二进制代码翻译成另一种架构的二进制代码
- 从而实现在一种架构平台上运行各种架构的程序

二进制翻译能用来做什么？

- 指令集模拟器、二进制分析工具等等
- 跨架构运行各种软件，助力软件生态的过渡发展
  - 重新编译：适配周期长，且无法适配闭源软件
  - 困难挑战：性能和稳定性
    - 翻译运行也需要能够正常、稳定的运行
    - 翻译运行也不能过于卡顿

华为 ExaGear
苹果 Rosetta 2
龙芯 LAT

01

二进制翻译简介

案例：一条加法指令的翻译方法

软件模拟CPU
就是一块数据了


```
struct CPUX86 {
    reg_t  regs[16];
    reg_t  rip;
    seg_t  segs[6];
    ....
} cpu
```

x86

```
add     %rax,   %rbx
```

x86

```
mov     %rax,   &cpu.regs[0]
mov     %rbx,   &cpu.regs[3]
add     %rax,   %rbx
mov     &cpu.regs[0], %rax
# EFLAGS CALCULATION
```

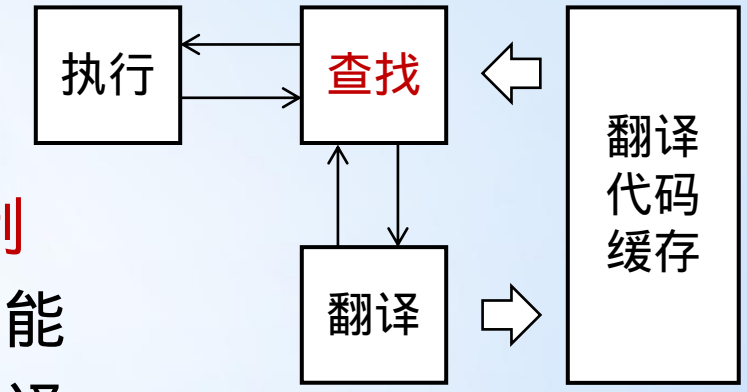
LoongArch

```
ld.d    $t0,    &cpu.regs[0]
ld.d    $t1,    &cpu.regs[3]
add.d   $t0,    $t1
st.d    $t0,    &cpu.regs[0]
# EFLAGS CALCULATION
```

读寄存器
读寄存器
计算
写寄存器
其他

01

二进制翻译简介



二进制翻译的基本原理

- 翻译：以代码块为单位，输入/输出都是二进制
- 执行：执行翻译后的二进制代码，实现程序功能
- 查找：搜索已有的翻译结果，找不到则进行翻译

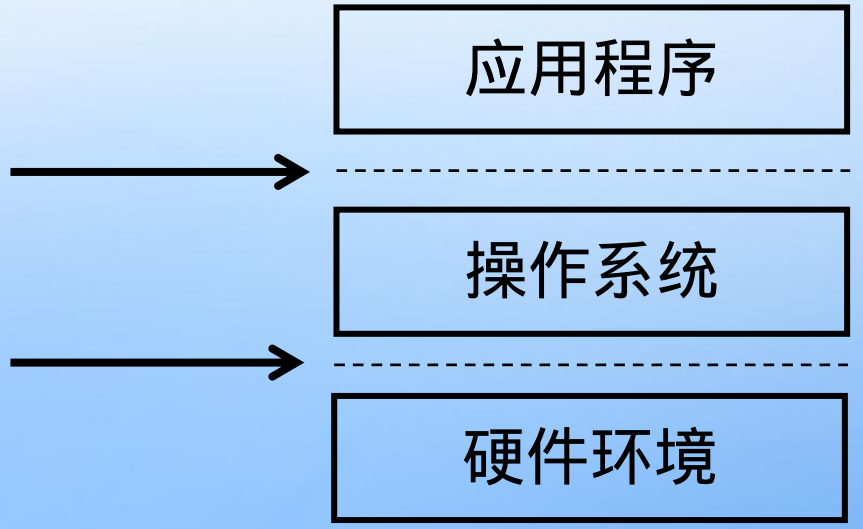
https://github.com/lat-opensource

用户级二进制翻译（运行客户机用户程序）

× 需求繁杂，适配困难 √ 性能更好，运行流畅

系统级二进制翻译（运行客户机操作系统）

√ 适配系统，一劳永逸 × 性能较差，运行卡顿



02 LATS 整体框架

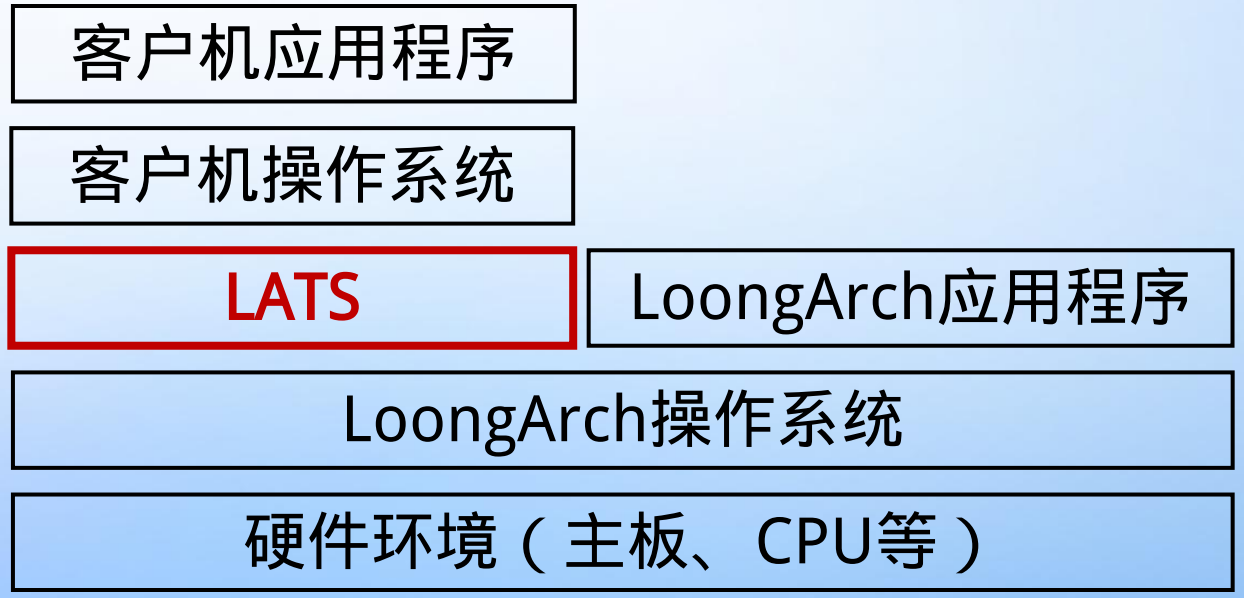
LATS (Loongson Architecture Translator System)

基于QEMU开发

- 开源二进制翻译框架
- 学术界、工业界广泛应用
- 丰富的硬件环境支持

LAT二进制翻译引擎

- 基于硬件的指令扩展
- 丰富的指令优化技术



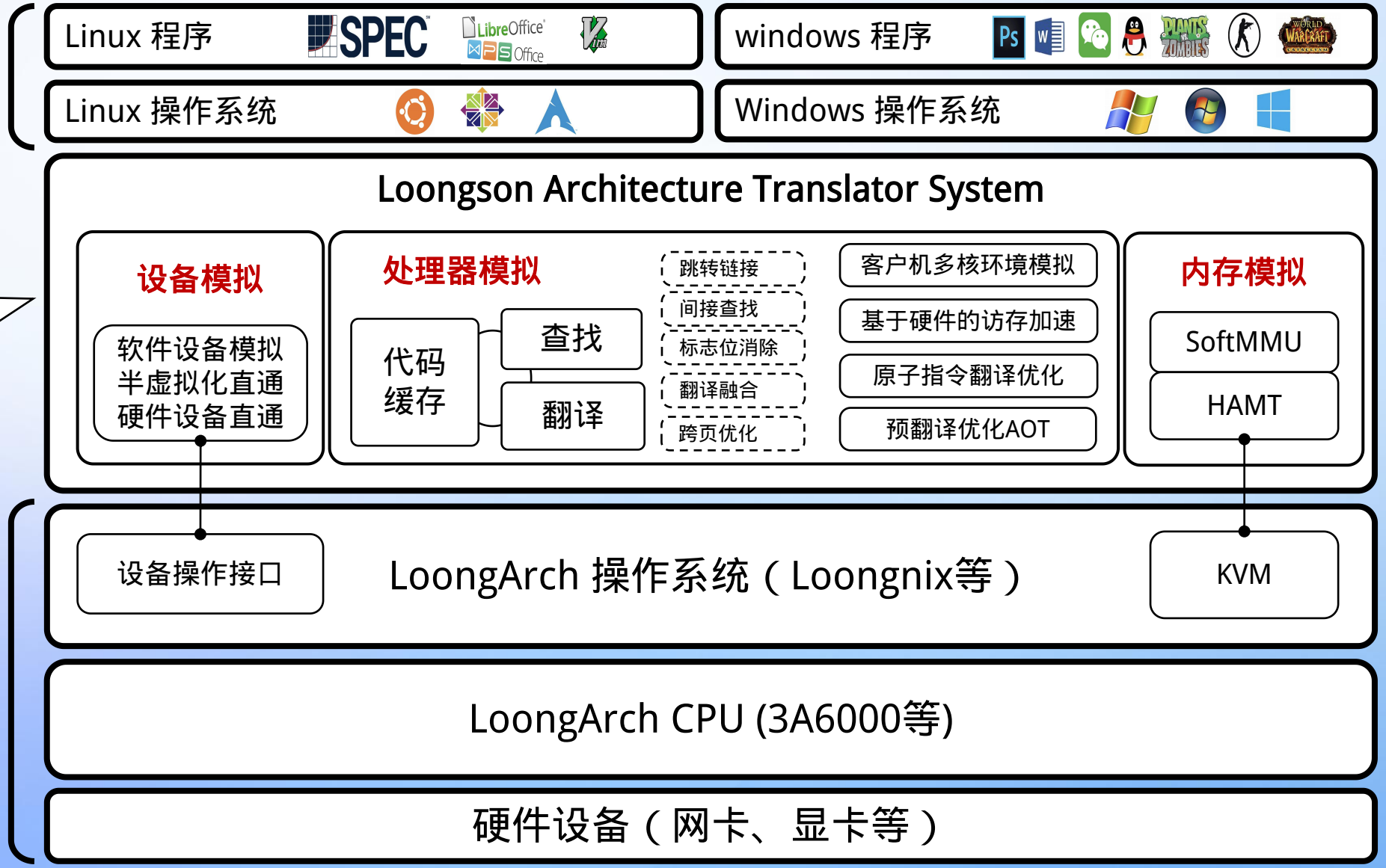
02

LATS 整体框架

客户机
运行操作系统
以支持各种应用

LATS
龙架构应用程序
模拟客户机硬件
环境以支持操作
系统的运行

龙架构平台
包括操作系统、
处理器CPU如
3A6000以及各类
真实硬件设备



02 LATS 整体框架

系统级模拟的目标和方法

- 优化目标：**高效**、**稳定**地运行客户机操作系统及应用程序
- 三大方面：处理器、内存、设备

处理器CPU

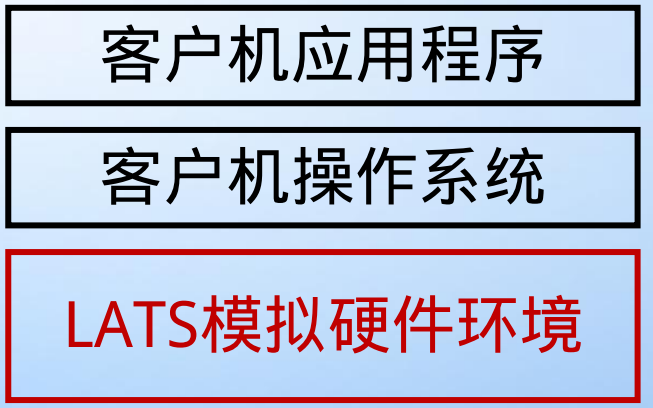
- 通过二进制翻译实现指令的跨架构执行
- 需提供特权资源的模拟，支持特权指令执行

内存MEM

- 客户机物理地址空间的模拟
- 支持客户机访存地址转换

设备I/O

- 模拟完整的硬件环境，主板、处理器、音频、显示等



03 跨架构系统模拟：处理器CPU

系统级模拟的目标和方法

- 优化目标：**高效**、稳定地运行客户机操作系统及应用程序
- 三大方面：处理器、内存、设备

处理器CPU

- 效率问题：翻译带来的**指令膨胀**：一条指令翻译出多条指令
- 经典的代码**翻译优化**技术
  - □ 标志位运算消除
  - □ 翻译模式匹配
  - □ 直接跳转链接
  - □ 返回地址栈
- 系统级模拟的特殊问题
  - □ 跨页直接跳转问题
  - □ 原子指令翻译问题

x86

```
add     %rax,  %rbx
```

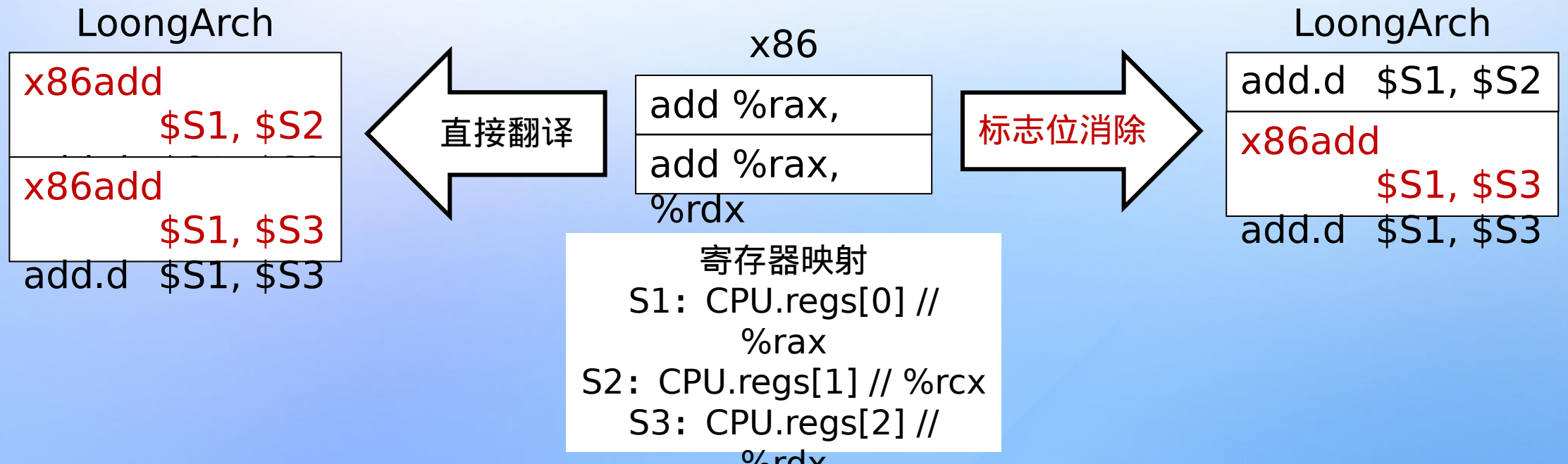
LoongArch

```
ld.d    $t0,    &cpu.regs[0]    读寄存器
ld.d    $t1,    &cpu.regs[3]    读寄存器
add.d   $t0,    $t1             计算
st.d    $t0,    &cpu.regs[0]    写寄存器
# EFLAGS CALCULATION            其他
```

03 跨架构系统模拟：处理器CPU

经典的代码翻译优化技术——标志位运算消除

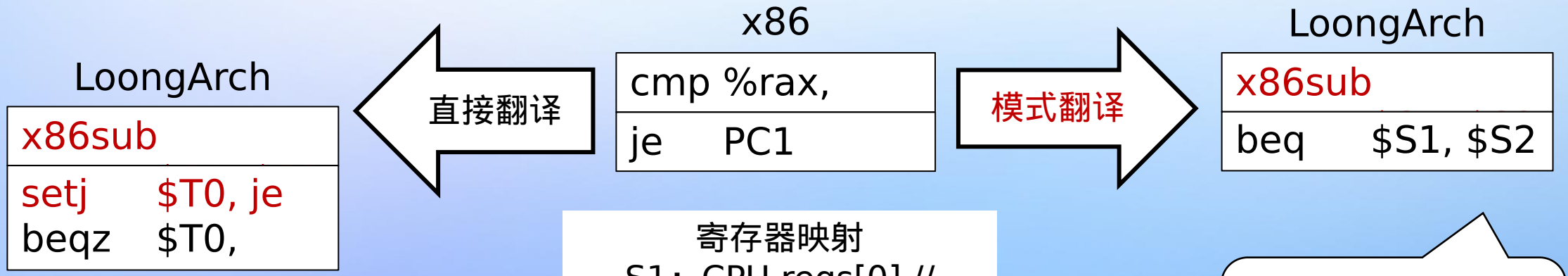
- LoongArch架构的**LBT**指令扩展，支持一条指令计算x86的EFLAGS
- 但仍有开销：分析指令流，消除无用的EFLAGS计算指令



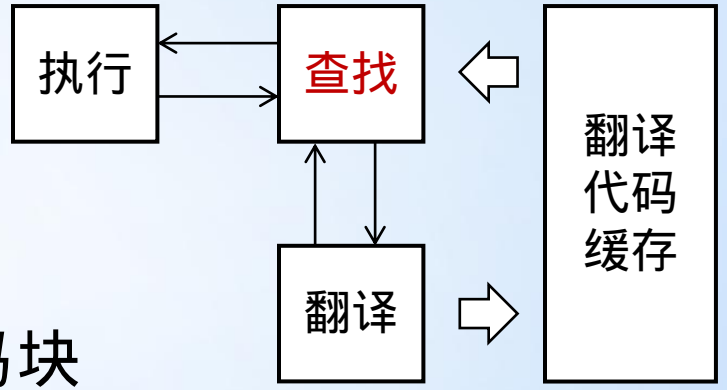
03 跨架构系统模拟：处理器CPU

经典的代码翻译优化技术——翻译模式匹配

- 语义关联的相邻指令进行联合翻译，降低膨胀率，例如cmp/jcc
- 不相邻时：分析内部指令流，保留寄存器实现联合翻译
  - 无访存（避免例外）、无寄存器占用（保证效率）

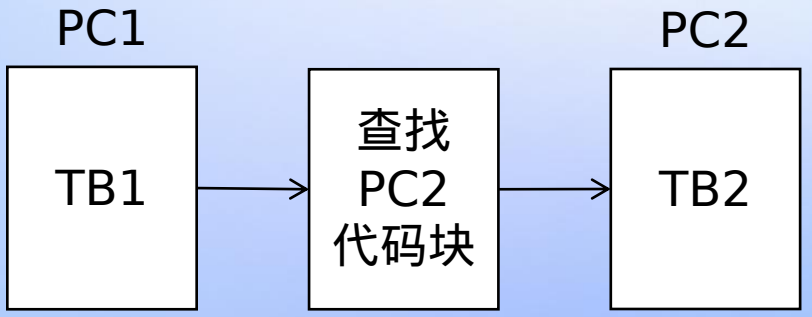


03 跨架构系统模拟：处理器CPU

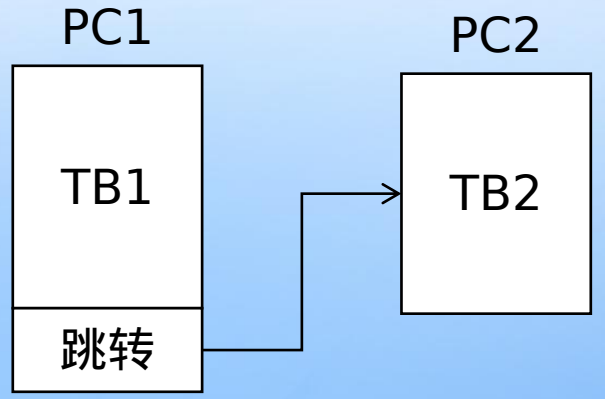


经典的代码翻译优化技术——直接跳转链接技术

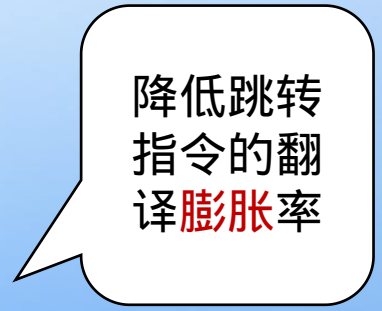
- 跳转指令的语义：客户机处理器的PC的改变
- 如何模拟跳转指令：**查找**到目标PC所在的翻译代码块
- **直接跳转链接**：已知跳转目标地址时，不经过查找而是直接跳转



查找过程实现跳转指令的语义



已知目标代码块，直接跳转



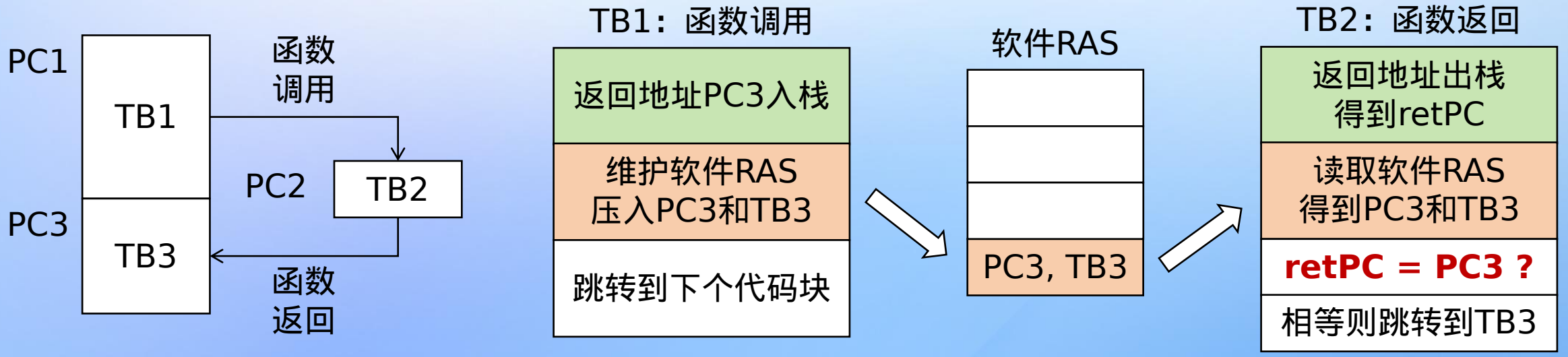
03 跨架构系统模拟：处理器CPU

经典的代码翻译优化技术——返回地址栈

间接跳转的目标地址是运行时变化的，需要**动态查找**目标翻译代码块

- 返回指令的目标地址具备很高的**可预测性**，如硬件RAS

LATS使用软件RAS技术实现高效的返回目标地址查找操作



03 跨架构系统模拟：处理器CPU

系统级模拟的目标和方法

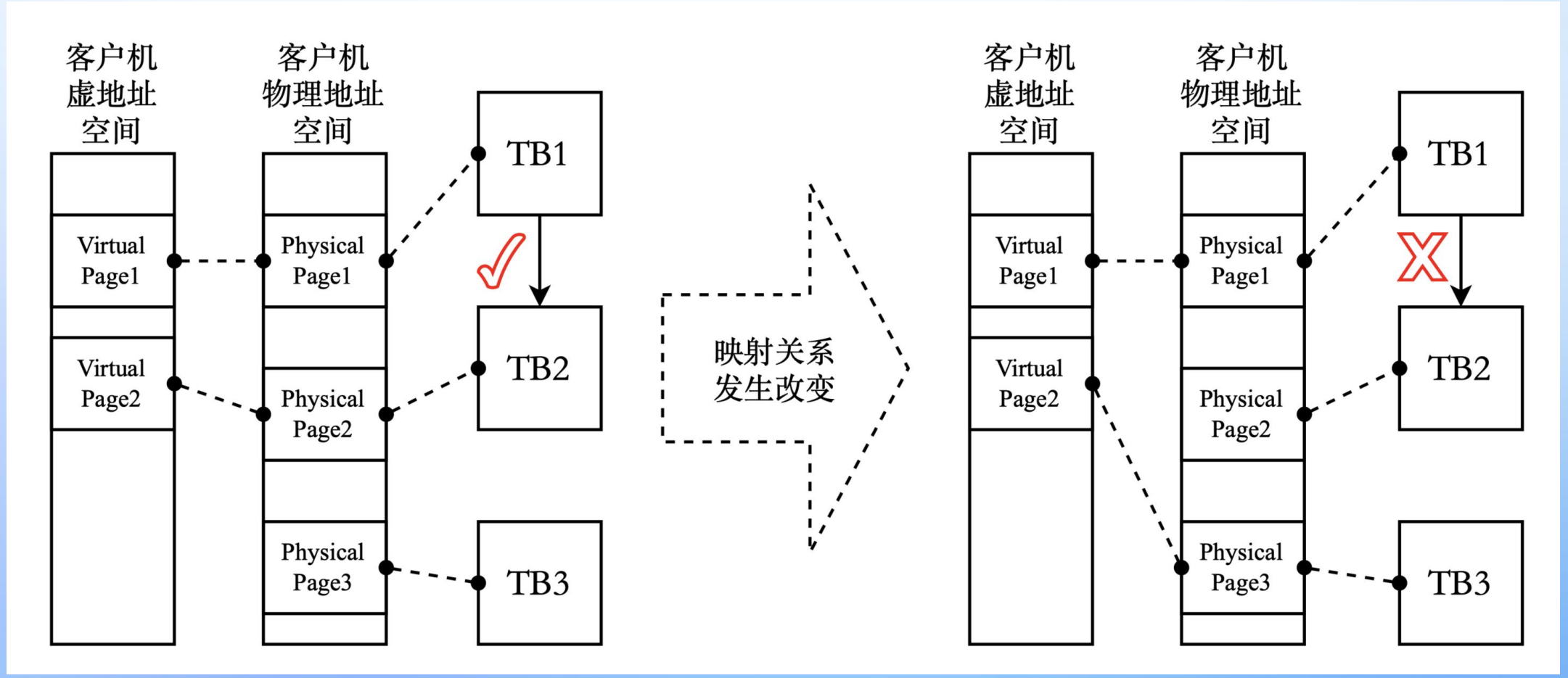
- 优化目标：**高效**、稳定地运行客户机操作系统及应用程序
- 三大方面：处理器、内存、设备

处理器CPU

- 效率问题：翻译带来的指令膨胀：一条指令翻译出多条指令
- 经典的代码翻译优化技术
  - √ 标志位运算消除…… 如x86的add指令需要额外计算EFLAGS
  - √ 翻译模式匹配…… 特定指令组合的语义实现更高效的翻译
  - √ 直接跳转链接…… 直接跳转指令的优化方法
  - √ 返回地址栈…… 返回目标地址的预测技术
- **系统级模拟**的特殊问题
  - □ 跨页直接跳转问题
  - □ 原子指令翻译问题

03 跨架构系统模拟：处理器CPU

系统级模拟——直接跳转跨页问题：客户机操作系统的地址映射发生改变



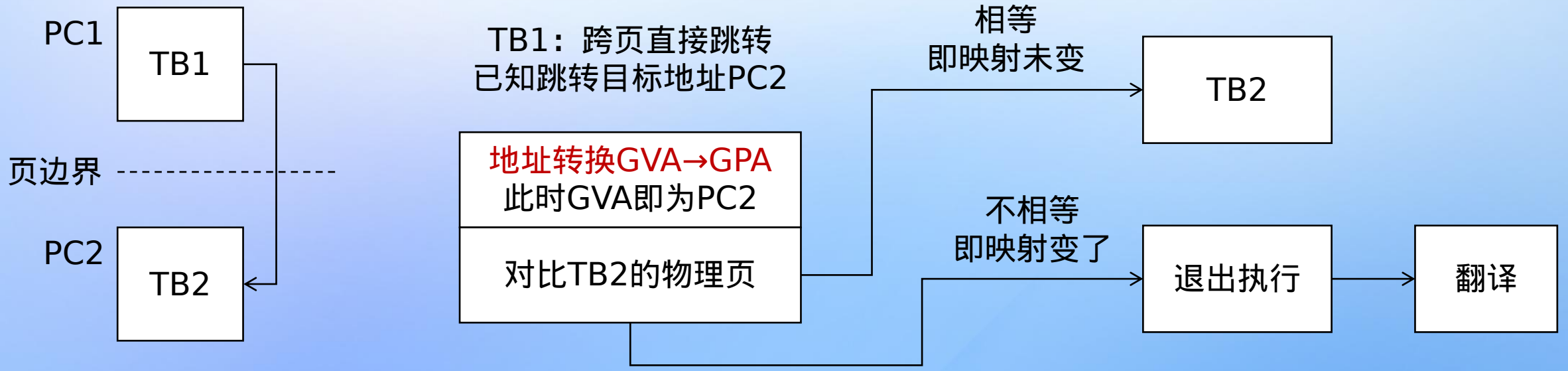
03 跨架构系统模拟：处理器CPU

系统级模拟——直接跳转跨页问题

最简单的处理方式：不允许跨页直接跳转进行代码块的直接链接

- 效率损失严重：程序中跨页直接跳转是普遍存在的，如函数call、页边界的循环

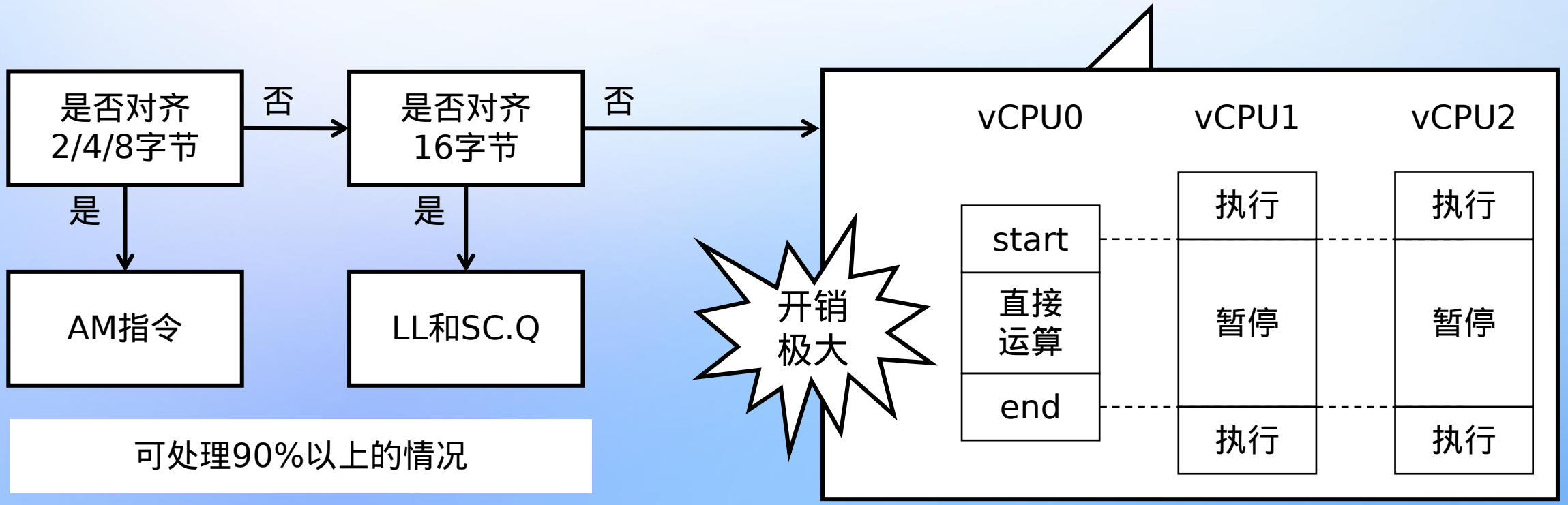
LATS在跨页直接跳转时进行动态的页映射检测



03 跨架构系统模拟：处理器CPU

系统级模拟——原子指令翻译

- 原子指令翻译可以直接使用LoongArch的原子指令，如**AMADD**
- 非对齐：最大16字节内可用**SC.Q**实现，否则退回到保留机制（暂停其他核）



03 跨架构系统模拟：处理器CPU

总结：处理器模拟

二进制翻译技术实现跨架构的指令执行

- **稳定**：反汇编、翻译、执行全过程保证正确，适配多种客户系统和应用
- **高效**：提高基础翻译效率，针对特定场景下瓶颈问题进行优化
  - 优化指令翻译，优化间接目标查找效率
  - 优化跨页问题的处理效率，优化原子指令的执行效率

其他：中断异常的模拟，自修改代码行为，多核架构，AOT预翻译技术

03 跨架构系统模拟：内存MEM

程序中**访存**操作是**广泛存在**的

跨架构系统级访存模拟的挑战：多级的地址转换

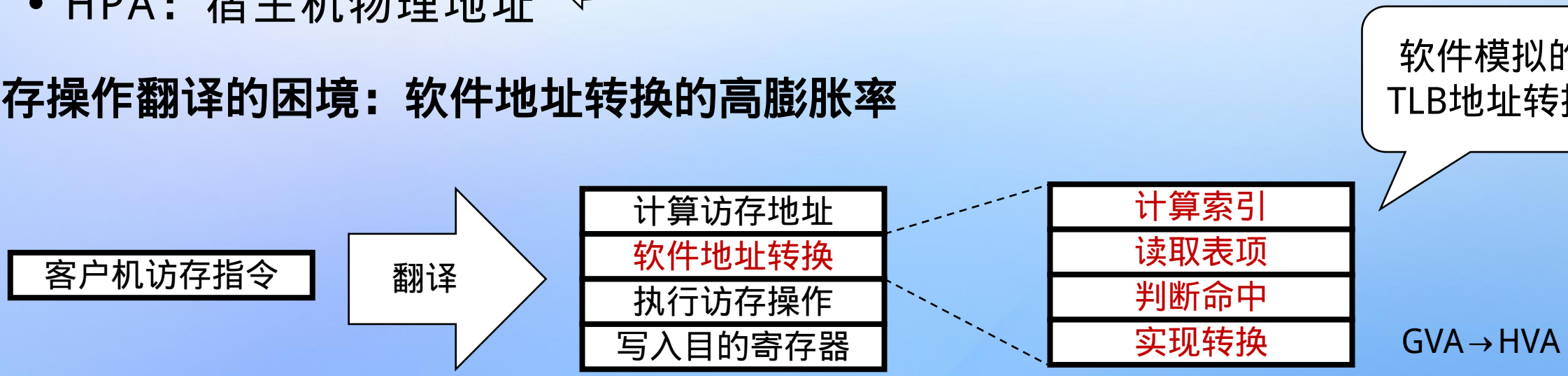
- GVA：客户机虚拟地址
- GPA：客户机物理地址
- HVA：宿主机虚拟地址
- HPA：宿主机物理地址

客户机页表：持有GVA到GPA的映射关系

翻译器维护：使用HVA空间模拟GPA空间，通常直接映射

宿主机页表：持有HVA到HPA的映射关系

访存操作翻译的困境：软件地址转换的高膨胀率



03 跨架构系统模拟：内存MEM

Dune: Safe User-level Access to Privileged CPU Features OSDI’12

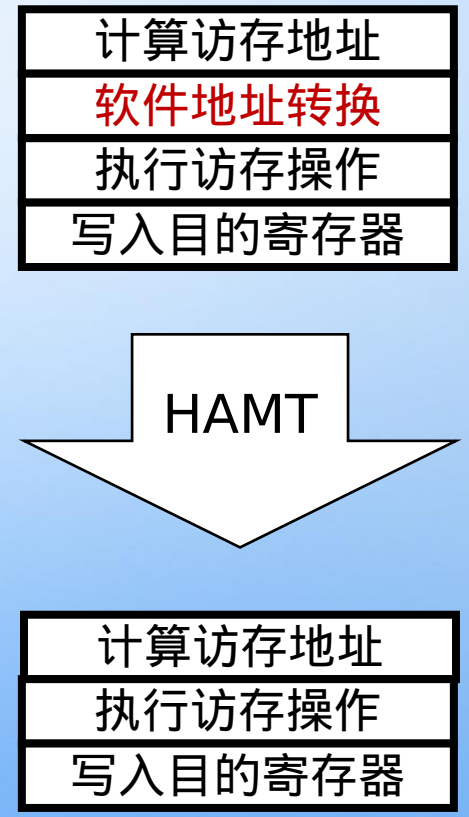
- https://github.com/project-dune/dune
- 2012年由斯坦福大学的研究团队提出
- 一个可以为**用户程序**提供安全、可靠地操作处理器**特权资源**的技术

移植到龙架构的 Loongson-Dune

- https://github.com/Martins3/loongson-dune

HAMT（Hardware Accelerated Memory Translation）

- 基于Dune框架将vCPU线程运行在虚拟机中
- 地址映射（GVA→HVA）直接填入硬件二级TLB
- **直接使用GVA完成访存！**

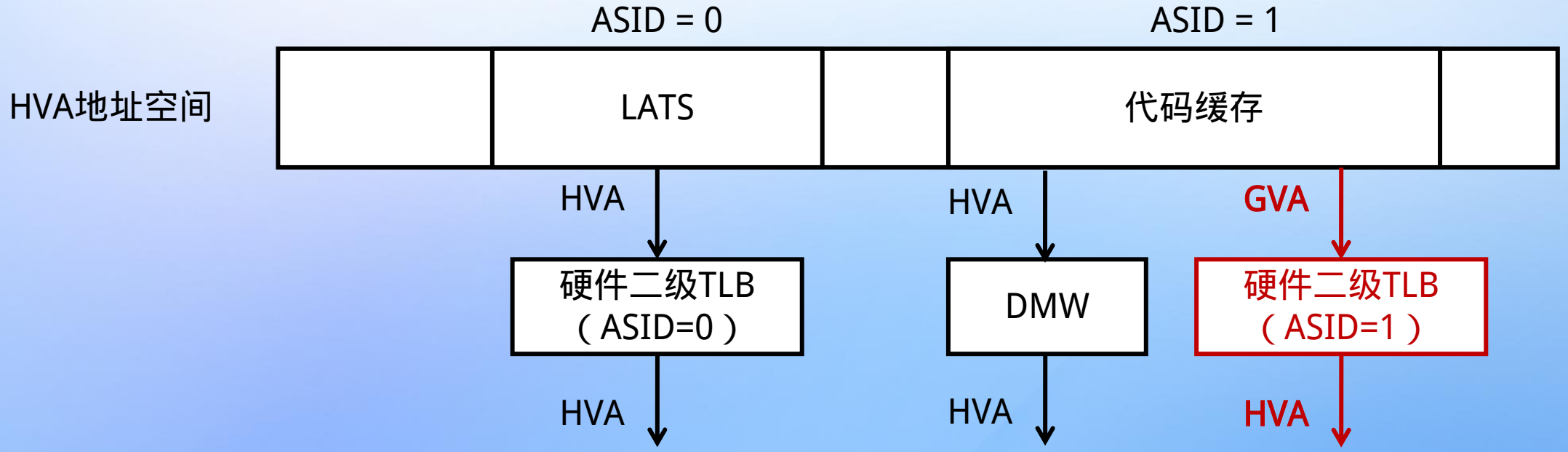


03 跨架构系统模拟：内存MEM

LATS作为用户程序，访存地址只能是 HVA

HAMT技术的关键：识别不同来源的访存请求

- 基于地址空间标识（ASID）区分代码缓存内部和外部
- 基于直接映射窗口（DMW）区分代码缓存内的访存来源



03 跨架构系统模拟：性能测试

二进制翻译性能

- 基于SPEC CPU2000
- 通过LATS翻译运行客户机的测试分数与原生宿主机的测试分数的比值

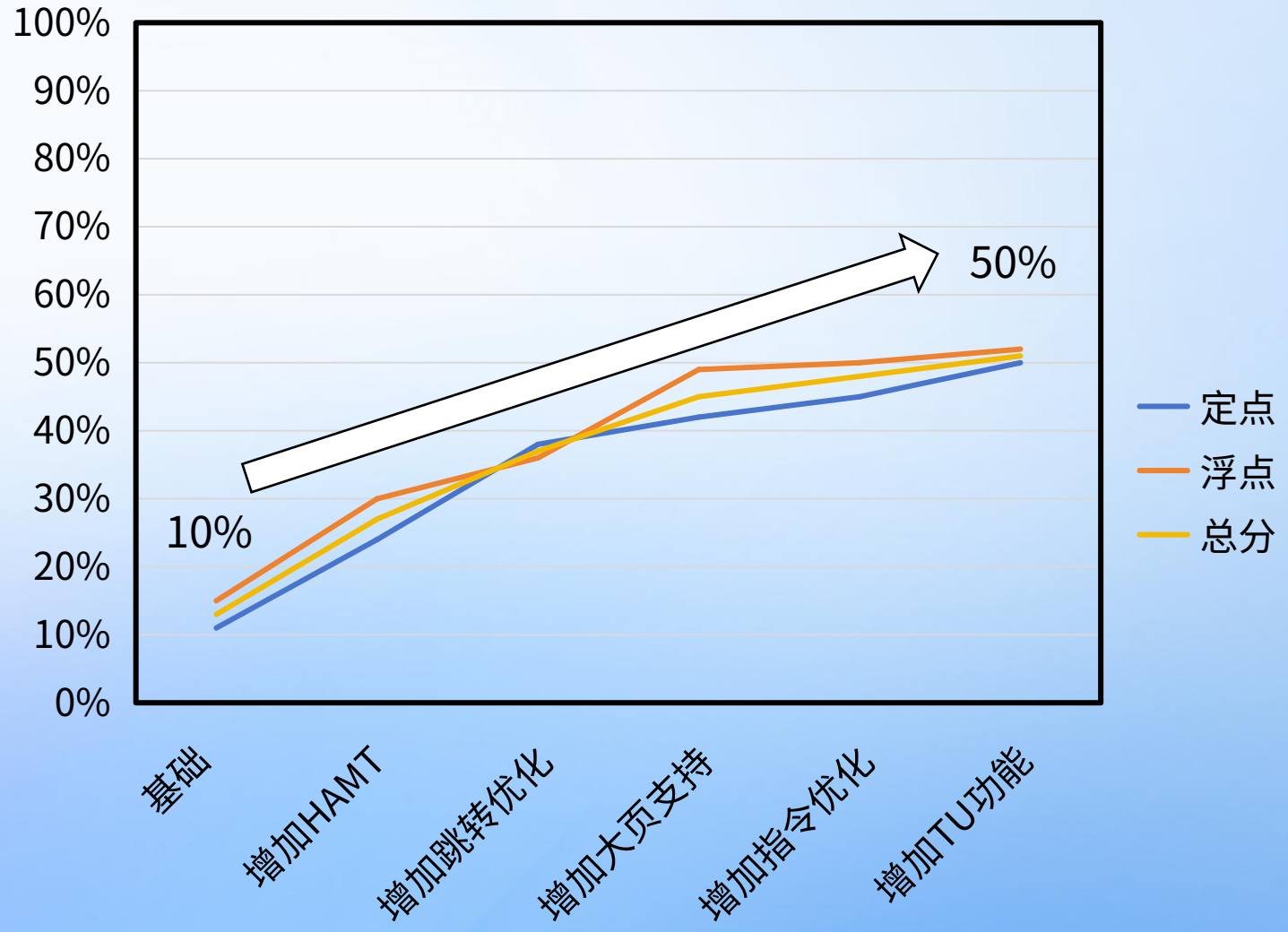
基础性能：10+%

访存优化：20+%

跳转优化：30+%

大页优化：40+%

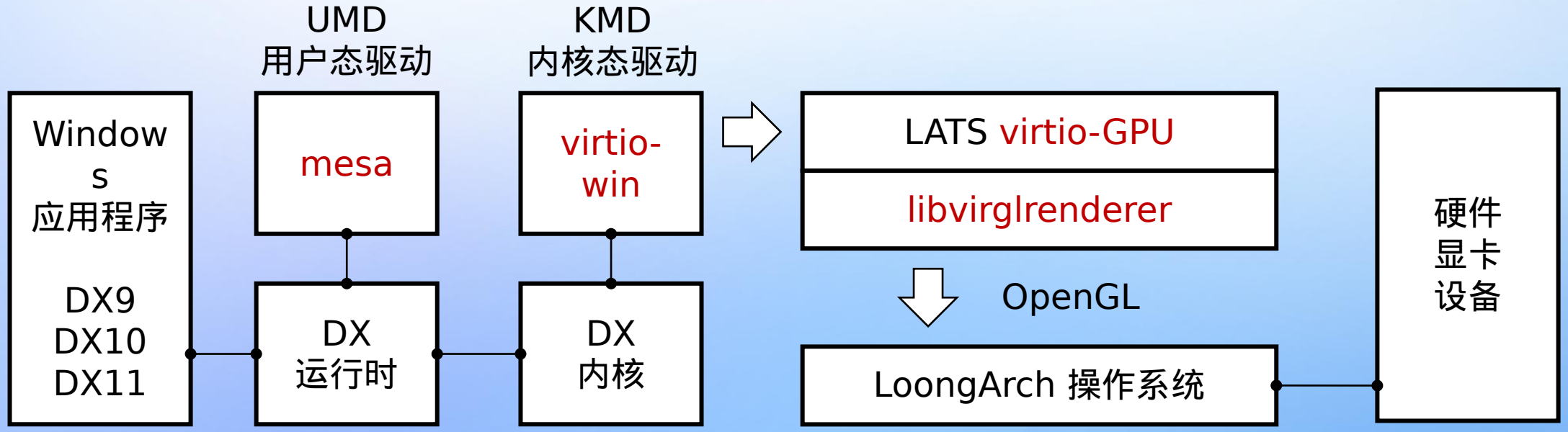
更多优化：50+%



03 跨架构系统模拟：设备I/O

- 软件模拟设备的效率较低：作为模拟器使用尚可，但不具备实用性
- 许多硬件设备通常与指令集**架构无关**，如显卡

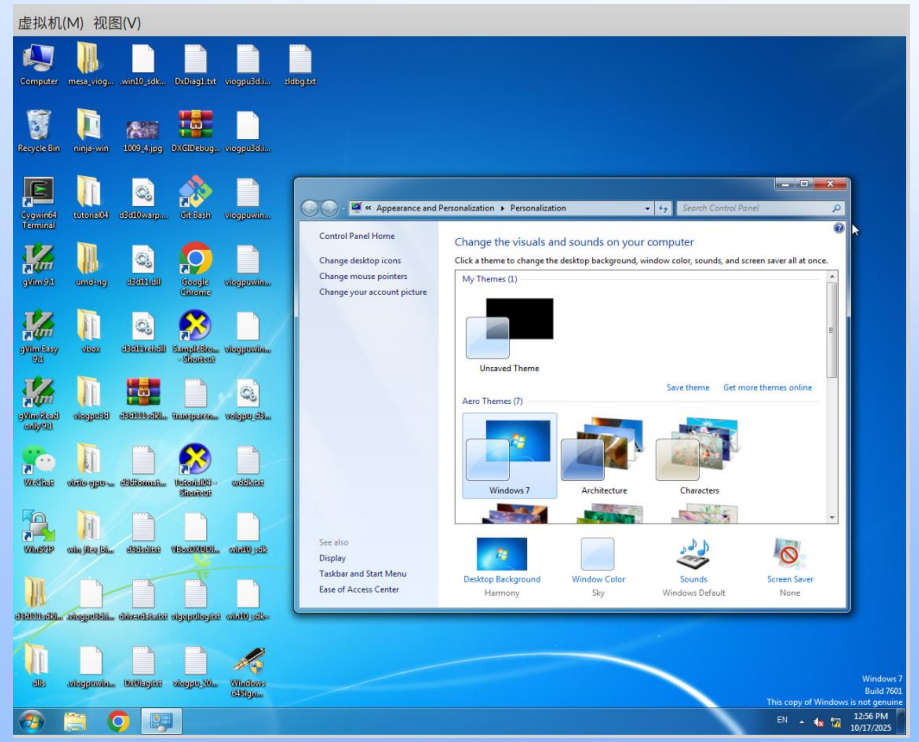
LATS基于virtio-GPU框架支持windows平台的显卡设备



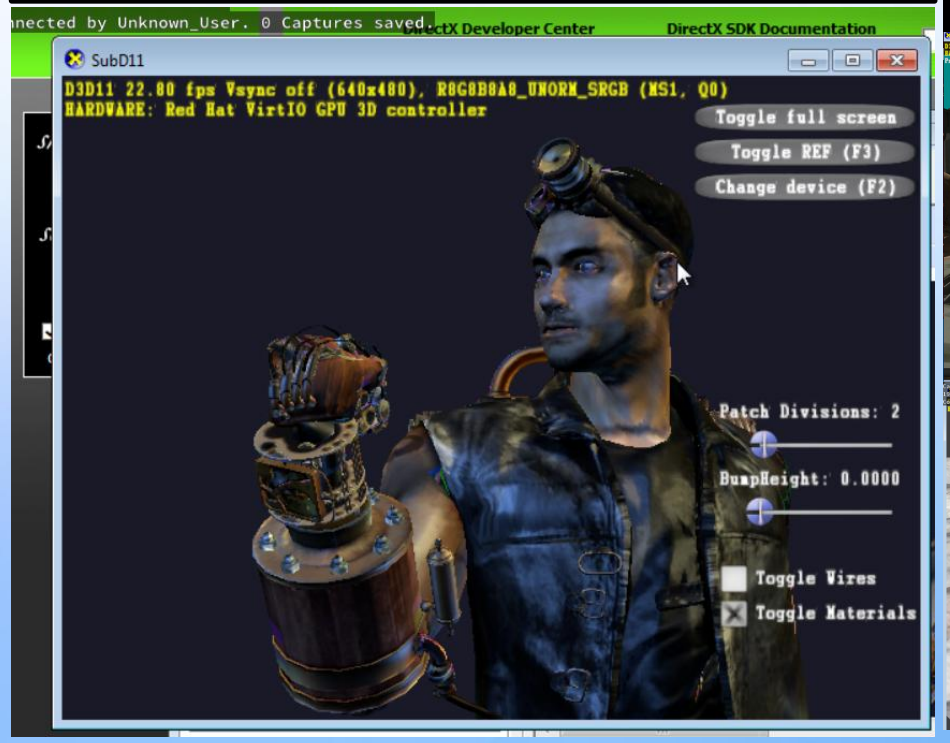
03 跨架构系统模拟：设备I/O

- 软件：LATS 运行 Windows 7 客户机系统
- 硬件：LoongArch 3A6000 + AMD RX580

Win 7 桌面 3D 效果



Dx11 测试 demo



04 总结

系统级二进制翻译：一种跨架构的系统虚拟化技术

处理器CPU

- 二进制翻译，面向所有指令，模拟特权资源和中断异常过程
- 针对翻译器特有问题的处理，如自修改代码行为
- 多种手段提高模拟性能：翻译代码优化降低膨胀率，提升代码局部性，硬件辅助

内存MEM

- 程序执行时的访存地址转换是影响运行效率的关键因素

设备I/O

- 架构无关设备的直通可以是实现高性能的手段：半虚拟化virtio、IOMMU设备直通
- 架构相关设备仍然采用软件模拟，如x86架构的APIC（也许参考KVM的发展？）

最后

LATS 在 Github 上**开源**

- 基于 QEMU 开发，遵循 GPLv2 协议
- 目前支持 x86 架构客户机
- 可以运行 Linux 系统、Windows 系统
- 支持在 LoongArch 的新/旧世界上运行

感谢
聆听

欢迎大家交流讨论！