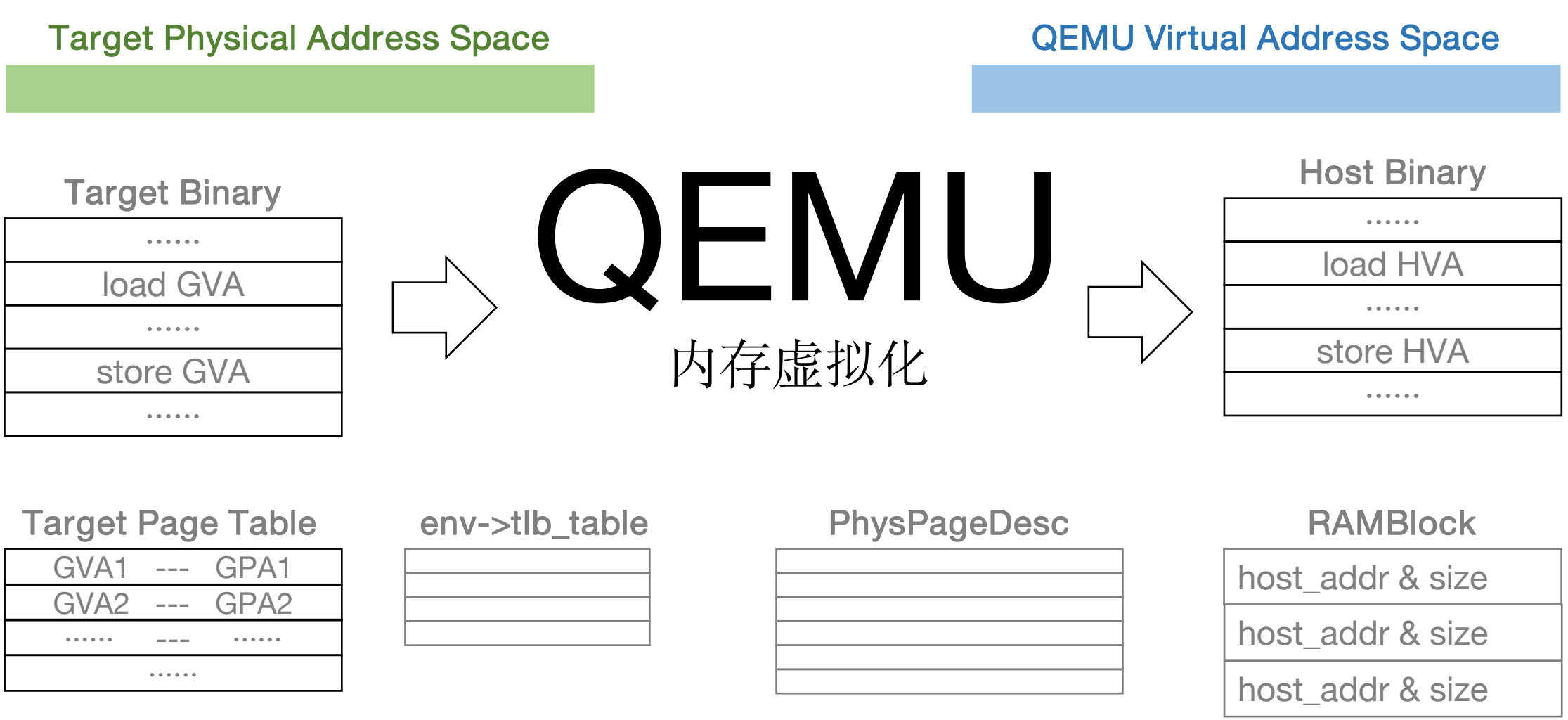
QEMU

内存虚拟化

20190919

Loongson Lab - Binary Translation


Target Physical Address Space
QEMU Virtual Address Space
Target Binary
.....
load GVA
.....
store GVA
.....
QEMU
内存虚拟化
Host Binary
.....
load HVA
.....
store HVA
.....
Target Page Table
GVA1 --- GPA1
GVA2 --- GPA2
..... --- .....
.....
env->tlb_table
PhysPageDesc
RAMBlock
host_addr & size
host_addr & size
host_addr & size

内存虚拟化

- 什么是内存虚拟化？到底在虚拟什么？
  - 所有的访存操作：指令、数据
- 如何管理地址映射
  - 管理 GVA 到 GPA 的映射：　模拟 Target 架构的页表、MMU
  - 管理 GPA 到 HVA 的映射：　通过 PhysPageDesc
  - 分配给 Target 的 HVA 空间：通过 RAMBlock
- Target 需要多大的地址空间：如 -m 512 即 512 MB 的内存空间
- Target 如何使用这地址空间：如 RAM、ROM、BIOS

RAMBlock

PhysPageDesc

主要内容

- 内存初始化
  - 如何建立 RAMBlock 和 PhysPageDesc
  - 不同视角下的内存空间
  - QEMU 管理下 PhysPageDesc 的层次化结构
- 地址转换 / 案例分析
  - 从 GVA 到 GPA：基于 Target 的页表
  - 从 GPA 到 HVA：基于 PhysPageDesc 和 RAMBlock
  - QEMU 内置的 TLB 管理：从 GVA 直接到 HVA
- 其他问题说明

内存初始化

- ./qemu -L pc-bios **-m 32** ~/dos.img
- int main(argc, argv, envp) { // vl.c:1987
  - QEMUMachine *machine;
  - machine = find_default_machine(); // 遍历全局变量 first_machine 链表
  - machine->**init**(**ram_size**, boot_devices,
    kernel_filename, kernel_cmdline,
    initrd_filename, cpu_model);
- struct QEMUMachine { // hw/boards.h:15
  - **QEMUMachineInitFunc *init;**
- **void pc_init_pci(ram_size, ...)** // hw/pc_piix.c:189

内存初始化

- static void pc_init_pci(ram_size, ...) // hw/pc_piix.c:189
  - pc_init1(ramsize, ...)
- static void pc_init1(ram_size, ...) // hw/pc_piix.c:63
  - pc_memory_init(ram_size, ...)
- void pc_memory_init(ram_size, ...) // hw/pc.c:959
  - ram_addr = **qemu_ram_alloc(...** // QEMU 申请动态内存空间
  - **cpu_register_physical_memory(...** // 设置 GPA 到 HVA 的映射

RAMBlock

PhysPageDesc

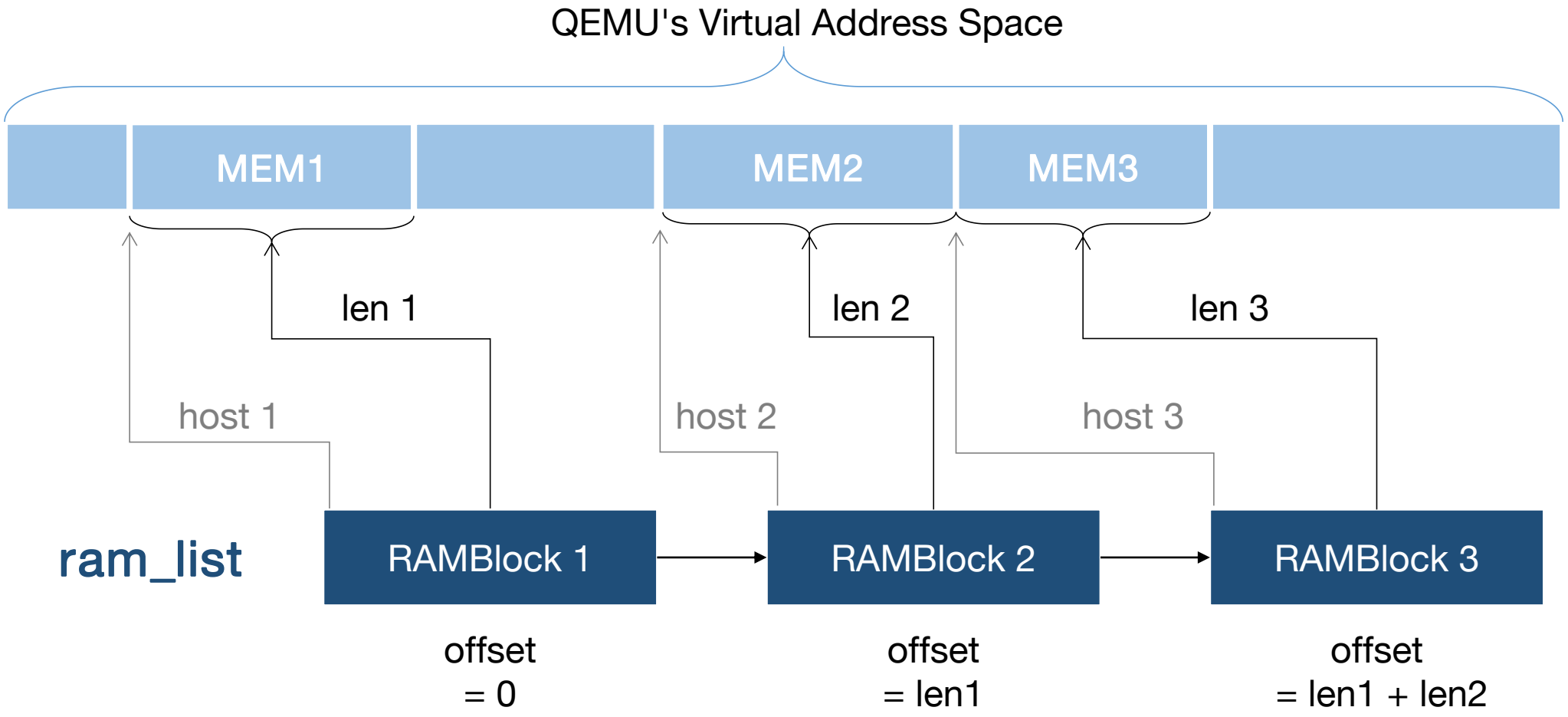
- ram_addr_t qemu_ram_alloc(...) // define in cpu-common.h:51

内存初始化: RAMBlock

- ram_addr_t qemu_ram_alloc(*dev, *name, size); // cpu-common.h:51
- qemu_ram_alloc_from_ptr(*dev, *name, size, *host) // exec.c:3189
  - RAMBlock *new_block, *block;
  - new_block = qemu_mallocz(sizeof(*new_block))
  - **new_block->host = qemu_vmalloc(size);**
  - qemu_madvise(new_block->host, size, ...);
  - **new_block->offset = find_ram_offset(size);**
  - new_block->length = size;

  - return new_block->offset;

内存初始化: RAMBlock



RAMLIST地址空间：所有 RAMBlock 组成的地址空间为 [0, SUM - 1]

内存初始化: PhysPageDesc

- cpu_register_physical_memory (start, size, phys, 0) // hw/pc_piix.c:41
- cpu_register_physical_memory_offset (// exec.c:3014

```
target_phys_addr_t start_addr,
ram_addr_t         size,
ram_addr_t         phys_offset
ram_addr_t         region_offset )
```

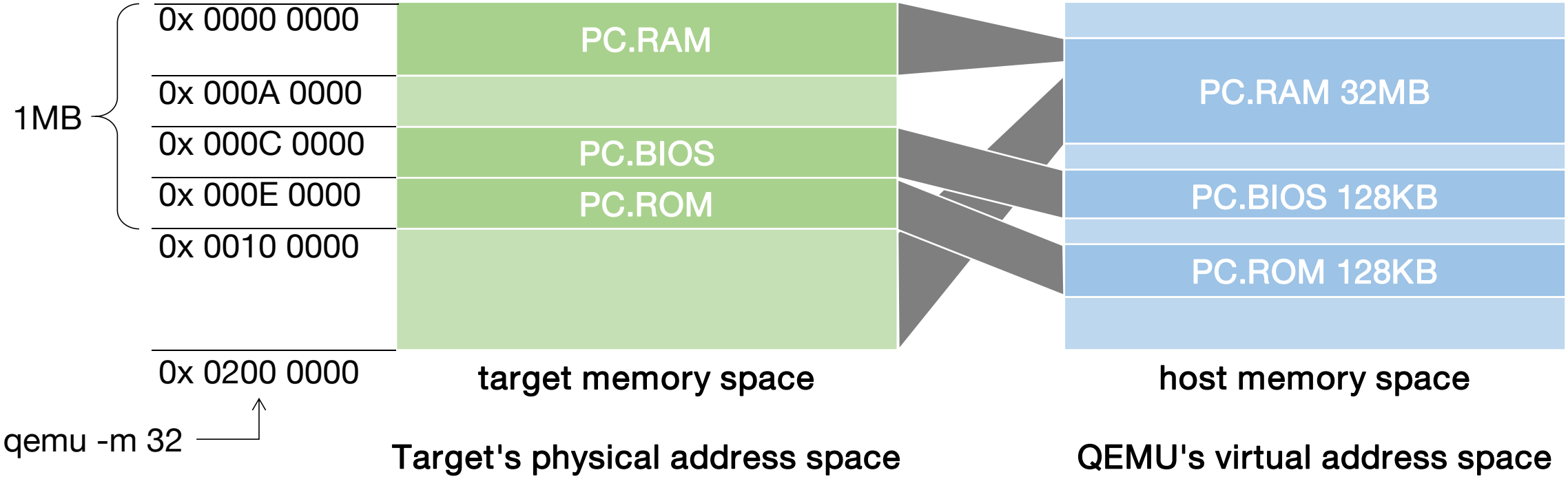
GPA

HVA ?

  - end_addr = start_addr + size;
  - for(addr = start_addr ; ... ; addr += TARGET_PAGE_SIZE)
    - p = phys_page_find_alloc(addr >> TARGET_PAGE_BITS, 1) // exec.c:3061
    - p->phys_offset = phys_offset;
    - p->region_offset = region_offset;

内存初始化

- ram_addr = qemu_ram_alloc(NULL, “pc.ram”, 0x0200 0000) // 32 MB
  - cpu_register_physical_memory(0x0000 0000, 0x000A 0000, ram_addr)
  - cpu_register_physical_memory(0x0010 0000, 0x01F0 0000, ram_addr + 0x0010 0000)
- ram_addr = qemu_ram_alloc(NULL, “pc.bios”, 0x0002 0000) // 128KB
  - cpu_register_physical_memory(0x000E 0000, 0x0002 0000, ram_addr)
- ram_addr = qemu_ram_alloc(NULL, “pc.rom”, 0x0002 0000) // 128KB
  - cpu_register_physical_memory(0x000C 0000, 0x0002 0000, ram_addr)



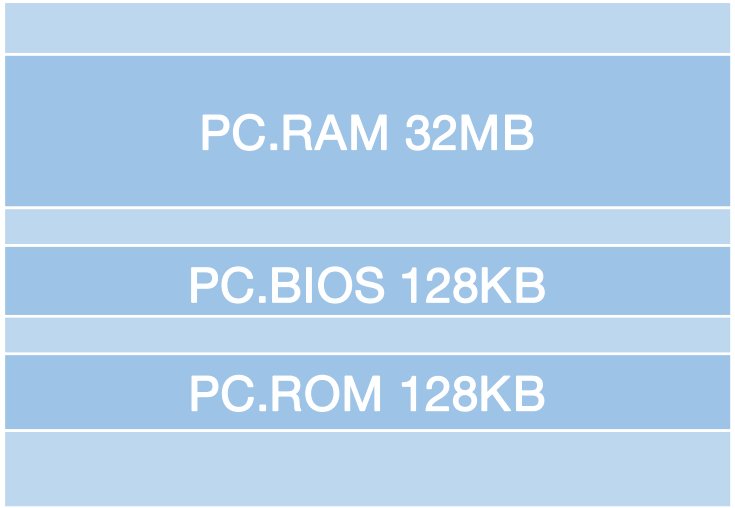
从 RAMBlock 看内存虚拟化

- ram_addr = qemu_ram_alloc(NULL, “pc.ram”, 0x0200 0000) // 32 MB
- ram_addr = qemu_ram_alloc(NULL, “pc.bios”, 0x0002 0000) // 128KB
- ram_addr = qemu_ram_alloc(NULL, “pc.rom”, 0x0002 0000) // 128KB

可以看到 QEMU 拥有的这几块动态内存空间

用于 Target 的内存虚拟化

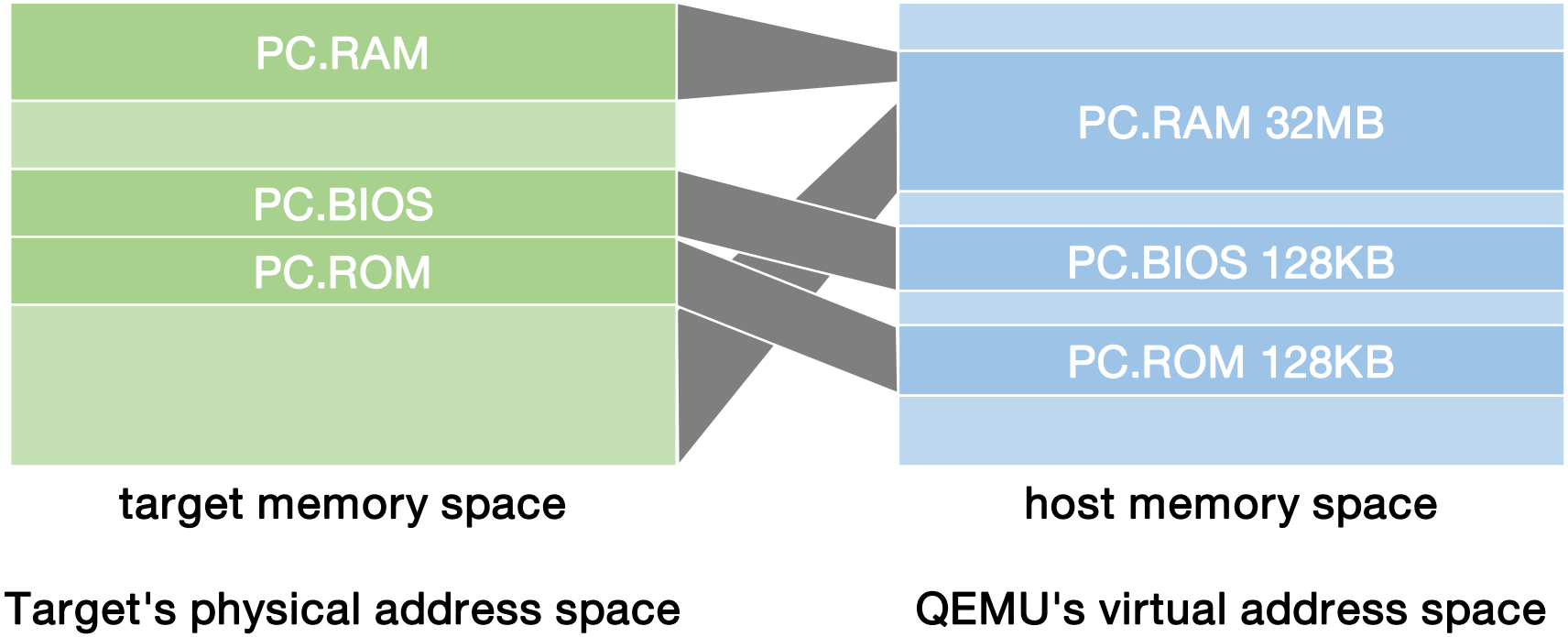
即这几块空间是当作Target的内存空间



从 PhysPageDesc 看内存虚拟化

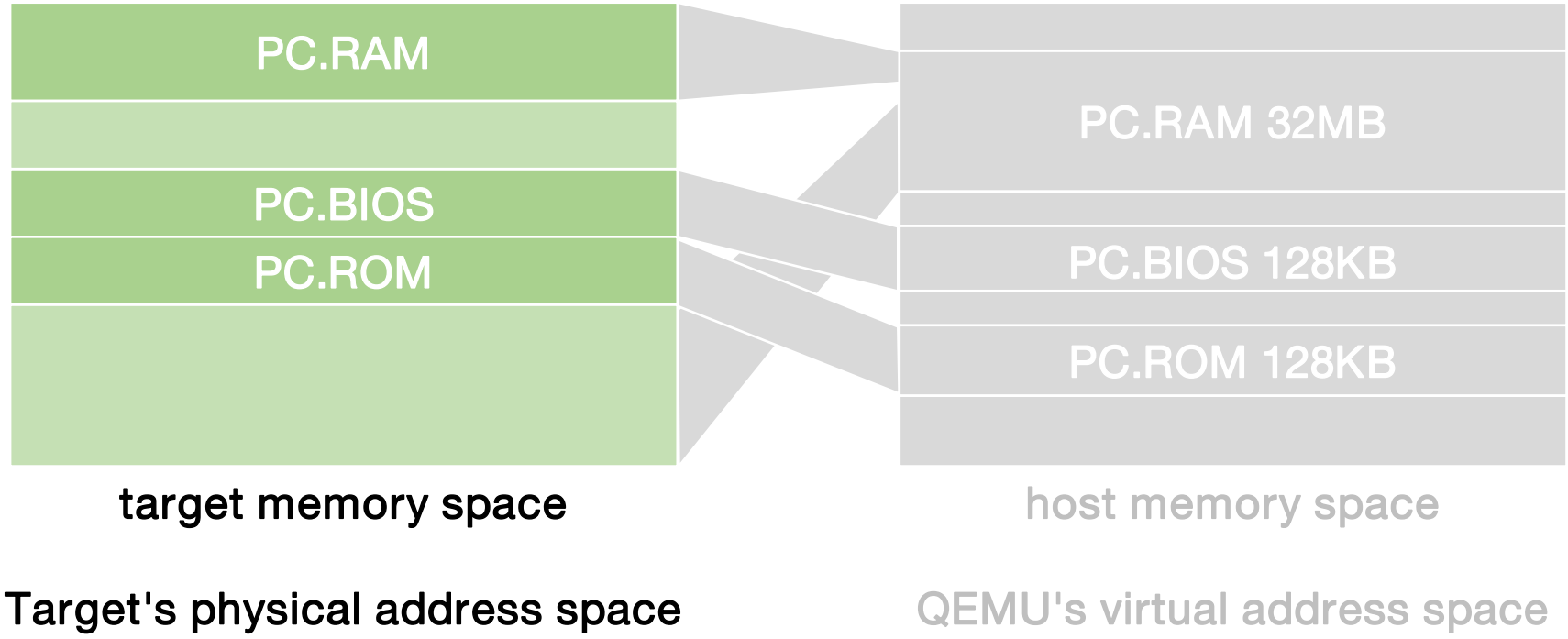
- cpu_register_physical_memory(0x0000 0000, 0x000A 0000, ram_addr)
- cpu_register_physical_memory(0x0010 0000, 0x01F0 0000, ram_addr + 0x0010 0000)
- cpu_register_physical_memory(0x000E 0000, 0x0002 0000, ram_addr)
- cpu_register_physical_memory(0x000C 0000, 0x0002 0000, ram_addr)

可以看到 GPA 和 HVA 之间的映射关系



从 Target 看内存虚拟化

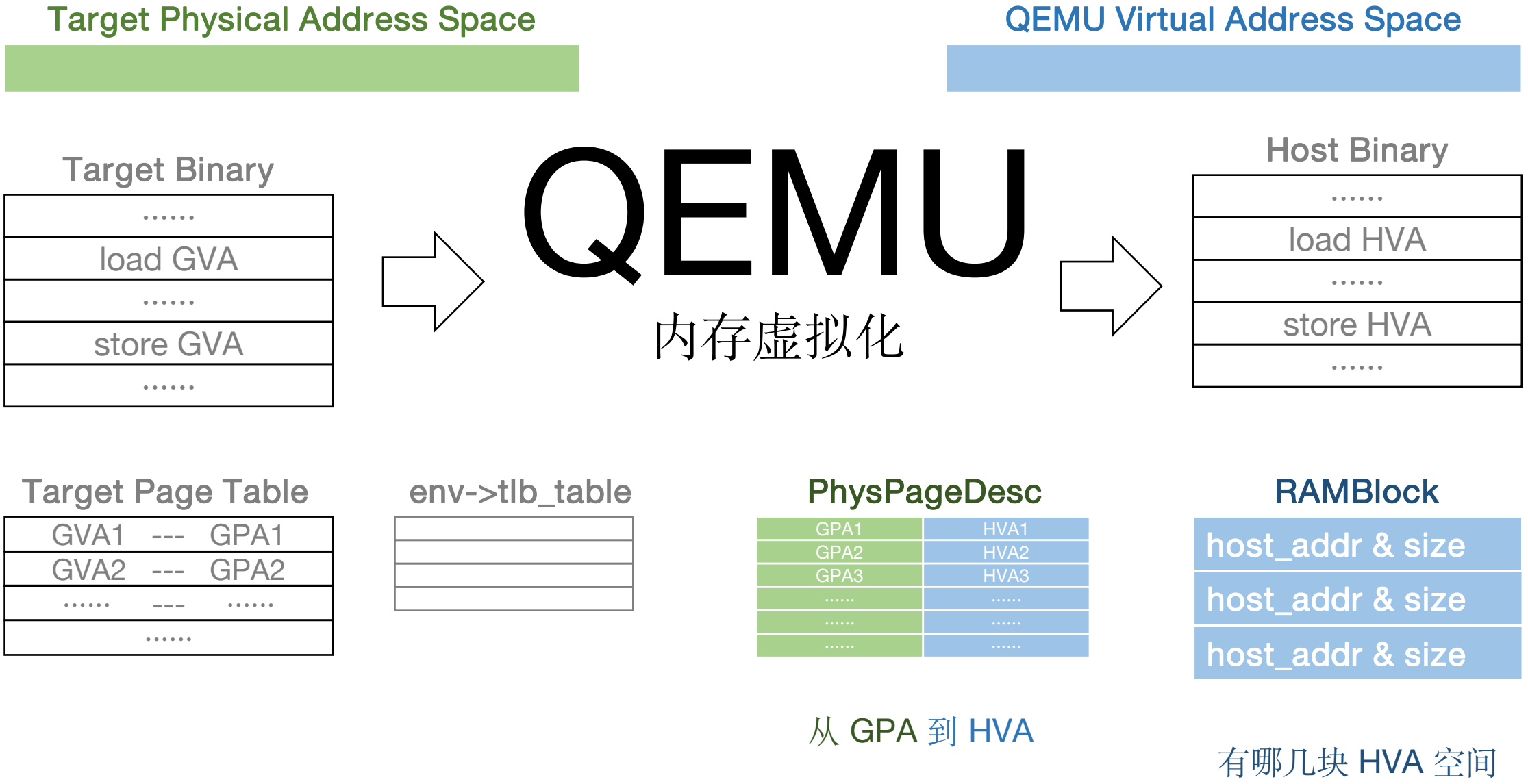
并不能看到。对 Target 而言是透明的。



QEMU 的内存虚拟化

- RAMBlock
  - 管理 QEMU 用于内存虚拟化所申请的动态内存空间
  - 组织成链表 ram_list，形成 RAMLIST 地址空间
  - 在 RAMLIST 地址空间中的地址成为 ram_addr_t
- PhysPageDesc
  - 管理 GPA 到 HVA 的映射关系
  - 页式管理，每个 PhysPageDesc 负责一个 GPA 物理页
  - 层次化组织

如何理解

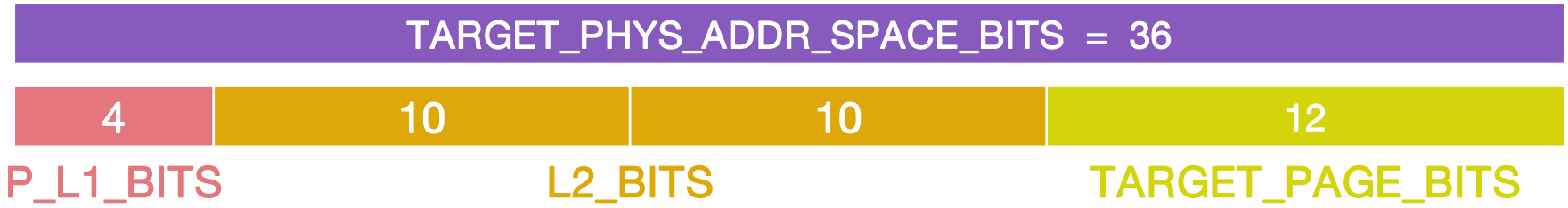

Target Physical Address Space
QEMU Virtual Address Space
Target Binary
.....
load GVA
.....
store GVA
.....
QEMU
内存虚拟化
Host Binary
.....
load HVA
.....
store HVA
.....
Target Page Table
GVA1 --- GPA1
GVA2 --- GPA2
..... --- .....
.....
env->tlb_table
PhysPageDesc
GPA1 HVA1
GPA2 HVA2
GPA3 HVA3
RAMBlock
host_addr & size
host_addr & size
host_addr & size
从 GPA 到 HVA
有哪几块 HVA 空间

PhysPageDesc 的层次结构

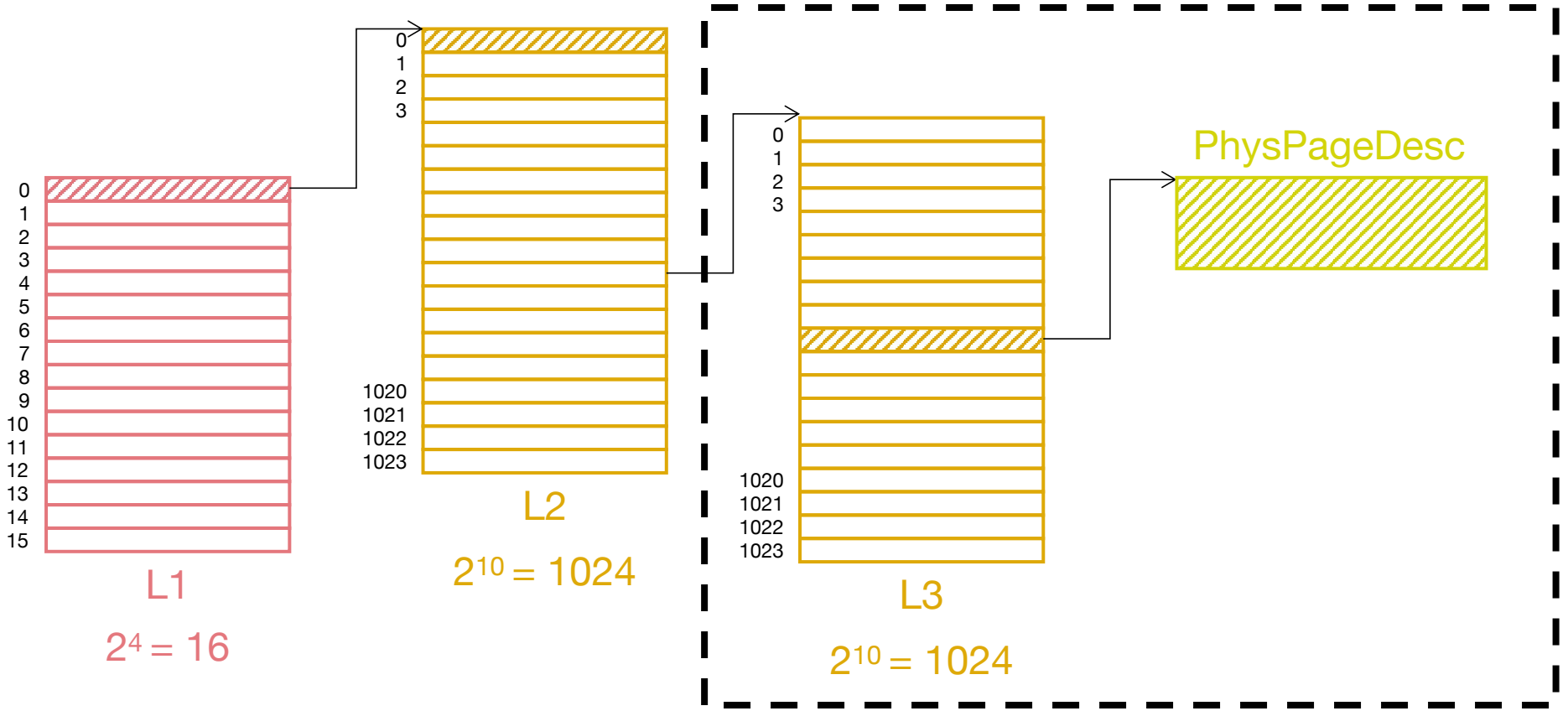
- PhysPageDesc *phys_page_find_alloc(index, alloc) // exec.c:434
  - lp = **l1_phys_map** + ((index >> P_L1_SHIFT) & (P_L1_SIZE - 1));
  - for (i = P_L1_SHIFT / L2_BITS - 1; i > 0; i--) {
    - void **p = *lp;
    - if (p == NULL) ***lp = p = qemu_mallocz(sizeof(void *) * L2_SIZE);**
    - lp = p + ((index >> (i * L2_BITS)) & (L2_SIZE - 1));
  - pd = *lp;
  - if (pd == NULL)
    - *lp = pd = **qemu_malloc(sizeof(PhysPageDesc) * L2_SIZE);**
    - for (i = 0; i < L2_SIZE; i++) {
    - pd[i].phys_offset = IO_MEM_UNASSIGNED;
    - pd[i].region_offset = (index + i) << TARGET_PAGE_BITS;
  - **return pd + (index & (L2_SIZE - 1));**

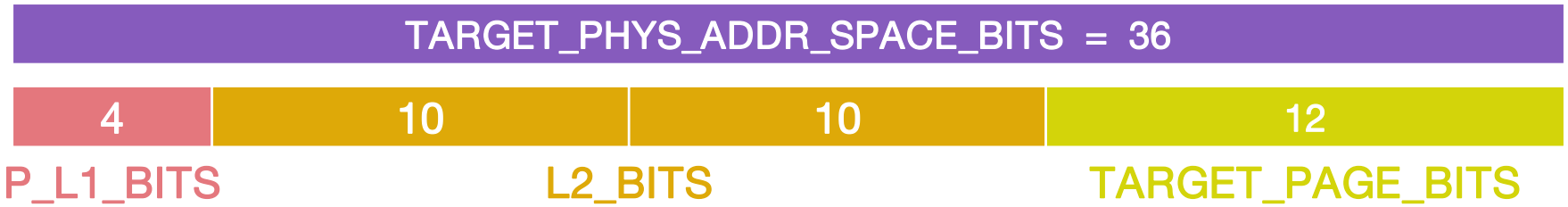
PhysPageDesc 的层次结构

- static void *l1_phys_map[P_L1_SIZE]; // exec.c:224
- #define TARGET_PHYS_ADDR_SPACE_BITS 36 // target-i386/cpu.h:945
- #define TARGET_VIRT_ADDR_SPACE_BITS 32 // target-i386/cpu.h:946
- #define TARGET_PAGE_BITS 12 // target-i386/cpu.h:936
- #define L2_BITS 10 // exec.c:178
- #define L2_SIZE (1 << L2_BITS) // exec.c:179
- #define P_L1_BITS_REM \
- ((TARGET_PHYS_ADDR_SPACE_BITS - TARGET_PAGE_BITS) % L2_BITS) // exe.c:182
- #define P_L1_BITS P_L1_BITS_REM // exec.c:191
- #define P_L1_SIZE ((target_phys_addr_t)1 << P_L1_BITS) // exec.c:200
- #define V_L1_SIZE ((target_ulong)1 << V_L1_BITS) // exec.c:201
- #define P_L1_SHIFT (TARGET_PHYS_ADDR_SPACE_BITS - TARGET_PAGE_BITS - P_L1_BITS) // exec.c:203
- #define V_L1_SHIFT (L1_MAP_ADDR_SPACE_BITS - TARGET_PAGE_BITS - V_L1_BITS) // exec.c:204

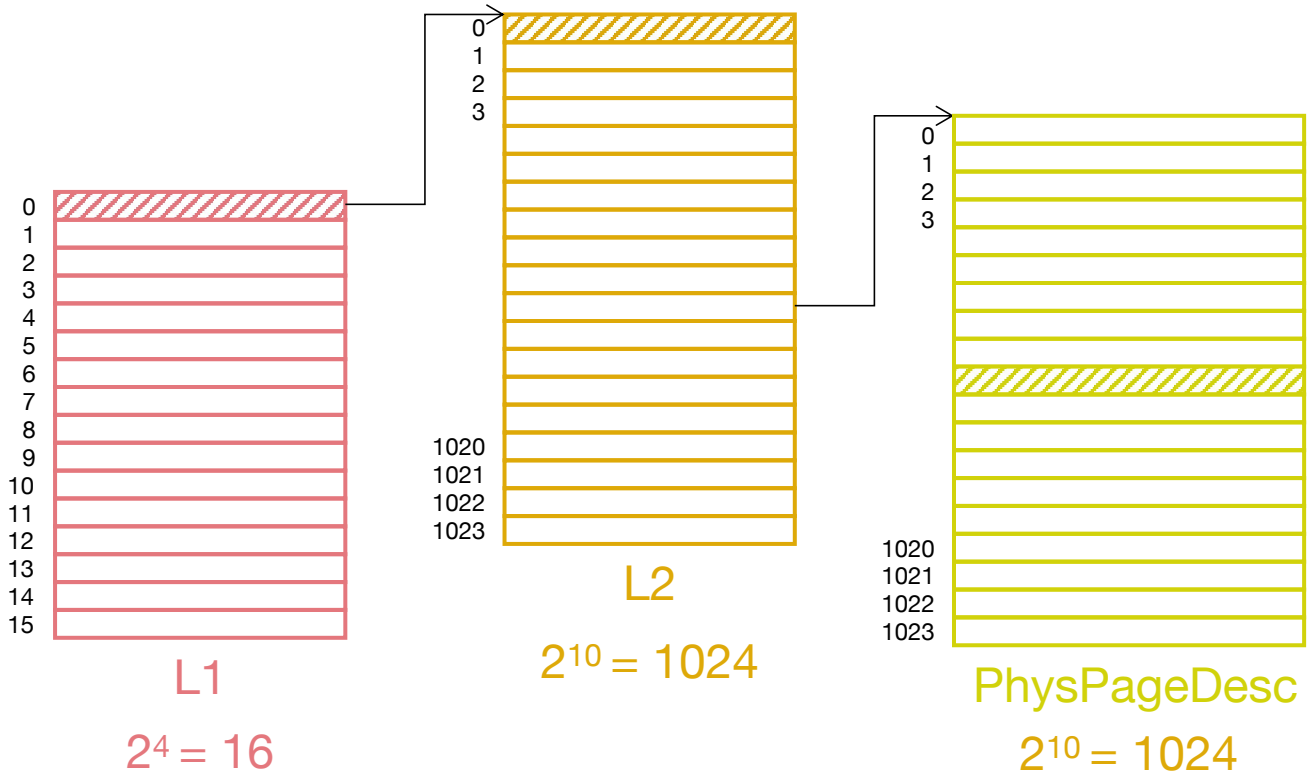


PhysPageDesc 的层次结构

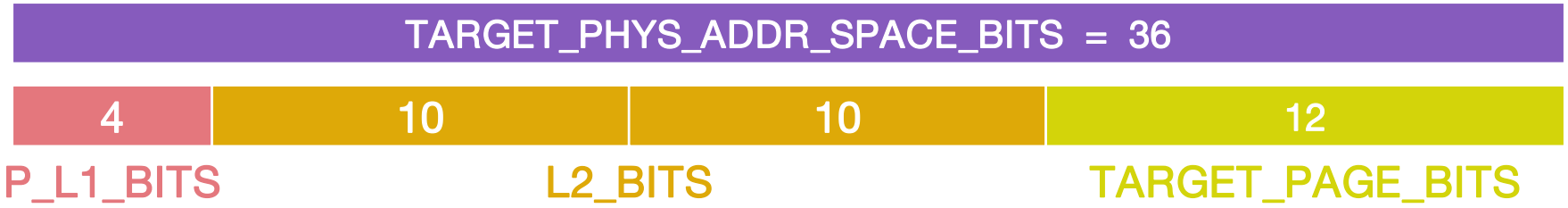




PhysPageDesc 的层次结构




```
typedef struct PhysPageDesc {
    ram_addr_t phys_offset;
    ram_addr_t region_offset;
} PhysPageDesc;
```



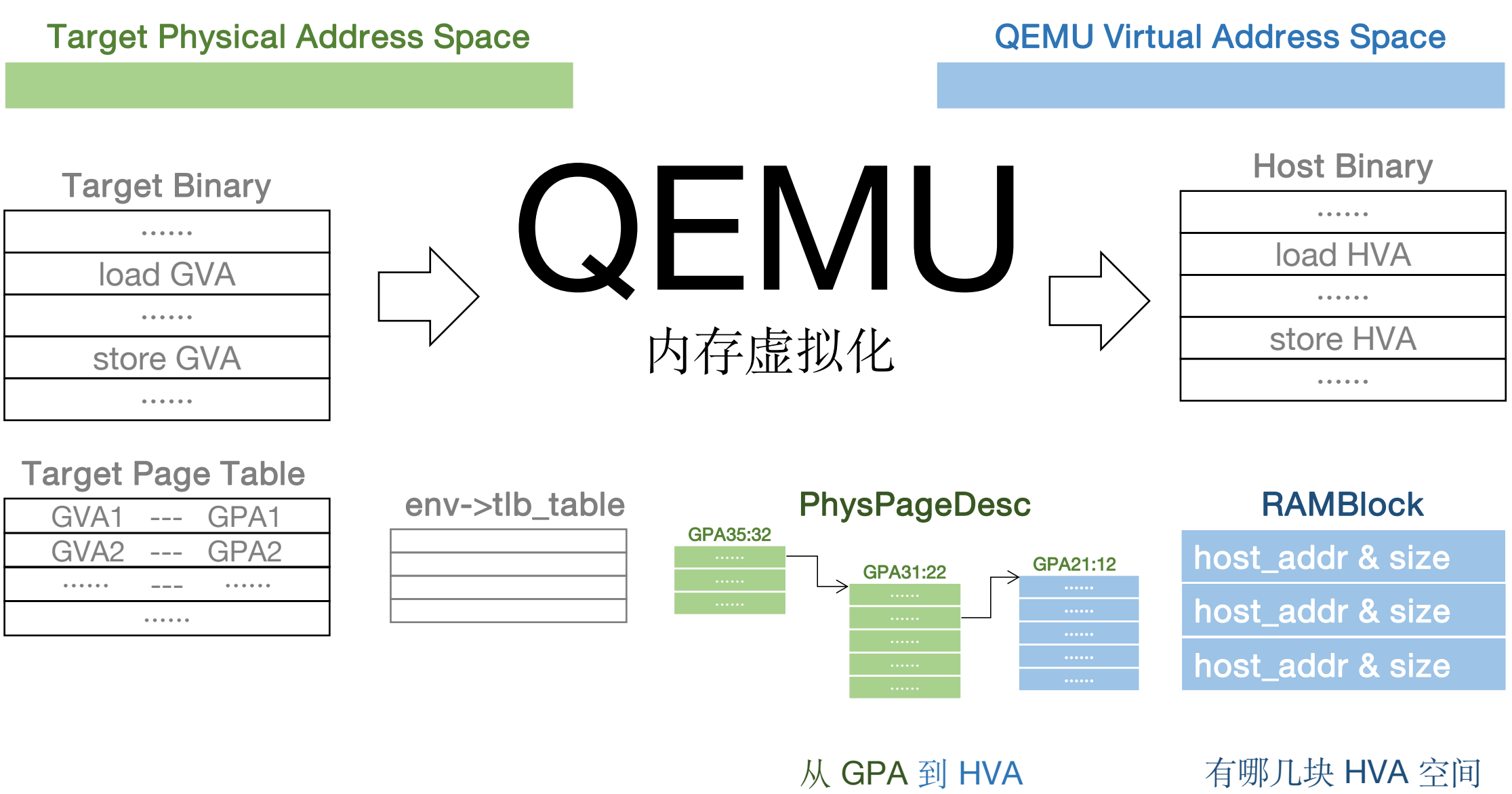
PhysPageDesc 的层次结构

- PhysPageDesc *phys_page_find_alloc(index, 1) // exec.c:434
  - lp = l1_phys_map + ((index >> P_L1_SHIFT) & (P_L1_SIZE - 1));
  - for (i = P_L1_SHIFT / L2_BITS - 1; i > 0; i--) {
    - void **p = *lp;
    - if (p == NULL) *lp = p = qemu_mallocz(sizeof(void *) * L2_SIZE);
    - lp = p + ((index >> (i * L2_BITS)) & (L2_SIZE - 1));
  - pd = *lp;
  - if (pd == NULL)
    - *lp = pd = qemu_malloc(sizeof(PhysPageDesc) * L2_SIZE);
    - for (i = 0; i < L2_SIZE; i++) {
    - pd[i].phys_offset = IO_MEM_UNASSIGNED;
    - pd[i].region_offset = (index + i) << TARGET_PAGE_BITS;
  - return pd + (index & (L2_SIZE - 1));

L1

L2 L3 ……

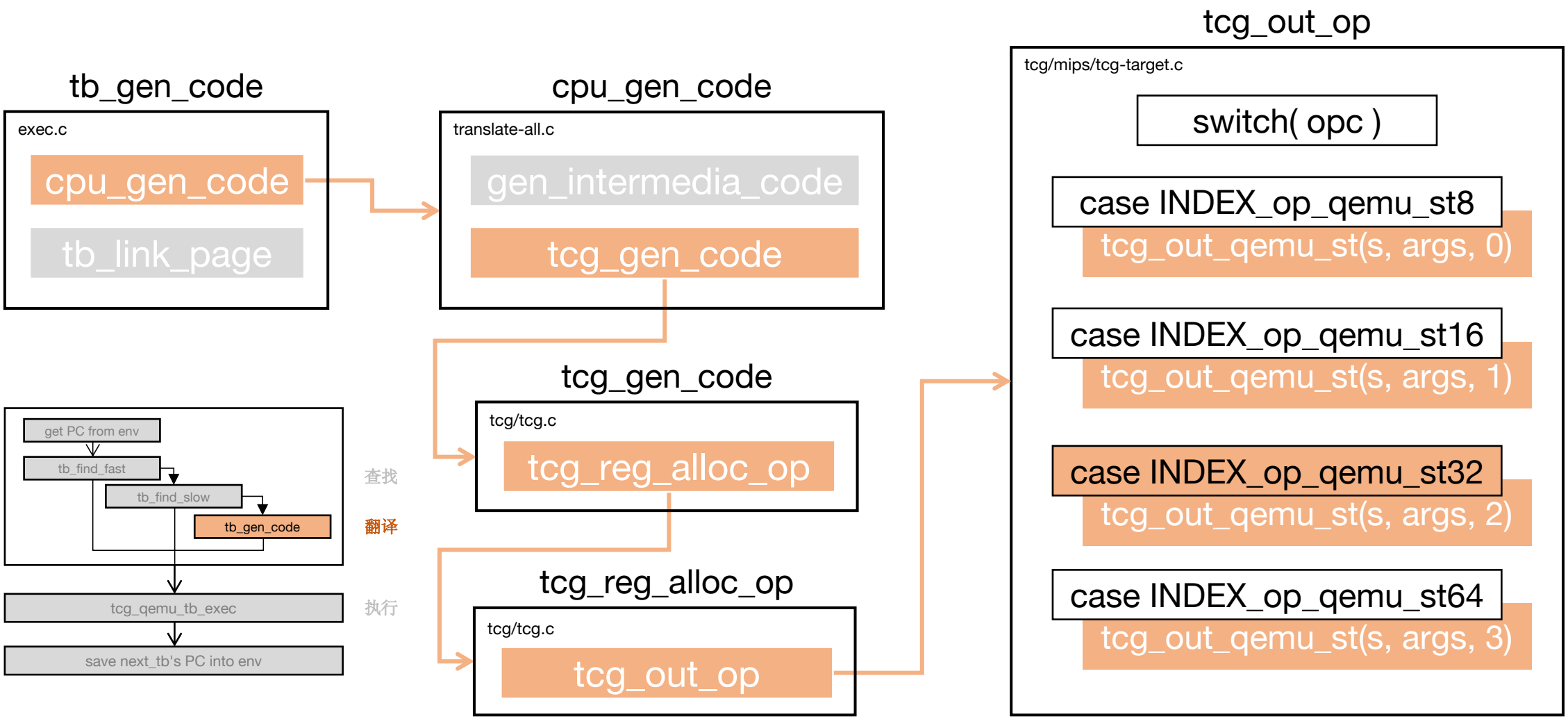
PhysPageDesc


Target Physical Address Space
QEMU Virtual Address Space
Target Binary
.....
load GVA
.....
store GVA
.....
QEMU
内存虚拟化
Host Binary
.....
load HVA
.....
store HVA
.....
Target Page Table
GVA1 --- GPA1
GVA2 --- GPA2
..... --- .....
.....
env->tlb_table
PhysPageDesc
GPA35:32
GPA31:22
GPA21:12
RAMBlock
host_addr & size
host_addr & size
host_addr & size
从 GPA 到 HVA
有哪几块 HVA 空间

地址转换

- 从 GVA 到 GPA
  - 基于 Target 的页表
- 从 GPA 到 HVA
  - 基于 PhysPageDesc 和 RAMBlock
- QEMU 内置的 TLB 管理：从 GVA 直接到 HVA
  - env->tlb_table[mmu_idx][page_index].addend

访存指令：翻译


tb_gen_code
exec.c
cpu_gen_code
tb_link_page
cpu_gen_code
translate-all.c
gen_intermedia_code
tcg_gen_code
tcg_gen_code
tcg/tcg.c
tcg_reg_alloc_op
tcg_reg_alloc_op
tcg/tcg.c
tcg_out_op
tcg_out_op
tcg/mips/tcg-target.c
switch(opc)
case INDEX_op_qemu_st8
tcg_out_qemu_st(s, args, 0)
case INDEX_op_qemu_st16
tcg_out_qemu_st(s, args, 1)
case INDEX_op_qemu_st32
tcg_out_qemu_st(s, args, 2)
case INDEX_op_qemu_st64
tcg_out_qemu_st(s, args, 3)
get PC from env
tb_find_fast
tb_find_slow
tb_gen_code
查找
翻译
tcg_qemu_tb_exec
执行
save next_tb's PC into env

访存指令： 翻译 tcg_out_qemu_st

sw data_reg, 0(addr_reg)

- void tcg_out_qemu_st(*s, *args, opc) // tcg/mips/tcg-atrget.c:1761
  #ifdef CONFIG_SOFTMMU
  #else
  - mov $a0, GUEST_BASE
  - addu $a0, $a0, addr_reg

  #endif
  - sw data_reg, $a0

addr + base 直接得到 HVA

直接执行 sw 指令即可

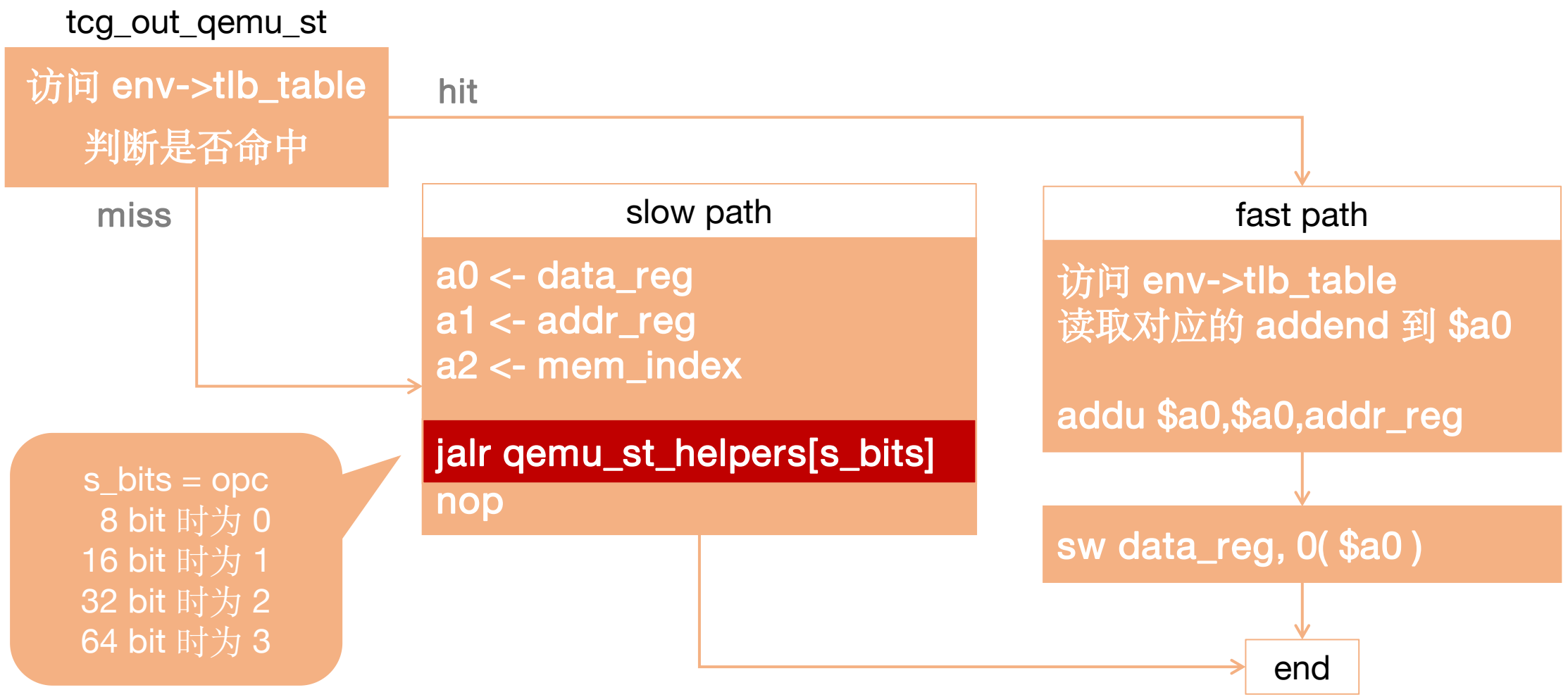
访存指令：翻译 tcg_out_qemu_st

sw data_reg, 0(addr_reg)

- void tcg_out_qemu_st(*s, *args, opc) // tcg/mips/tcg-atrget.c:1761
  - srl $a0, addr_reg, TARGET_PAGE_BITS - TLB_ENTRY_BITS
  - andi $a0, $a0, (CPU_TLB_SIZE - 1) << CPU_TLB_ENTRY_BITS
  - addu $a0, $a0, $s0 // $s0 always stores the address of env
  - lw$at, $a0, OFFSET(CPUState, tlb_table[mmu_idx][0].addr_write)
  - mov $t0, TARGET_PAGE_MASK | ((1 << s_bits) - 1)
  - beq $t0, $at, fast_path
  - nop
  - /* slow path */
  - beq $zero, $zero, end
  - /* fast path */
  - sw addr_reg, 0($a0)
  - /* end */

访存指令： 翻译 tcg_out_qemu_st

sw data_reg, 0(addr_reg)



访存指令：执行 __stl_mmu(addr,val,idx)

- static void *qemu_st_helpers[4] = { // tcg/mips/tcg-target.c:948
  - __stb_mmu, // [0] 8 bit
  - __stw_mmu, // [1] 16 bit
  - __stl_mmu, // [2] 32 bit
  - __stq_mmu, // [3] 64 bit

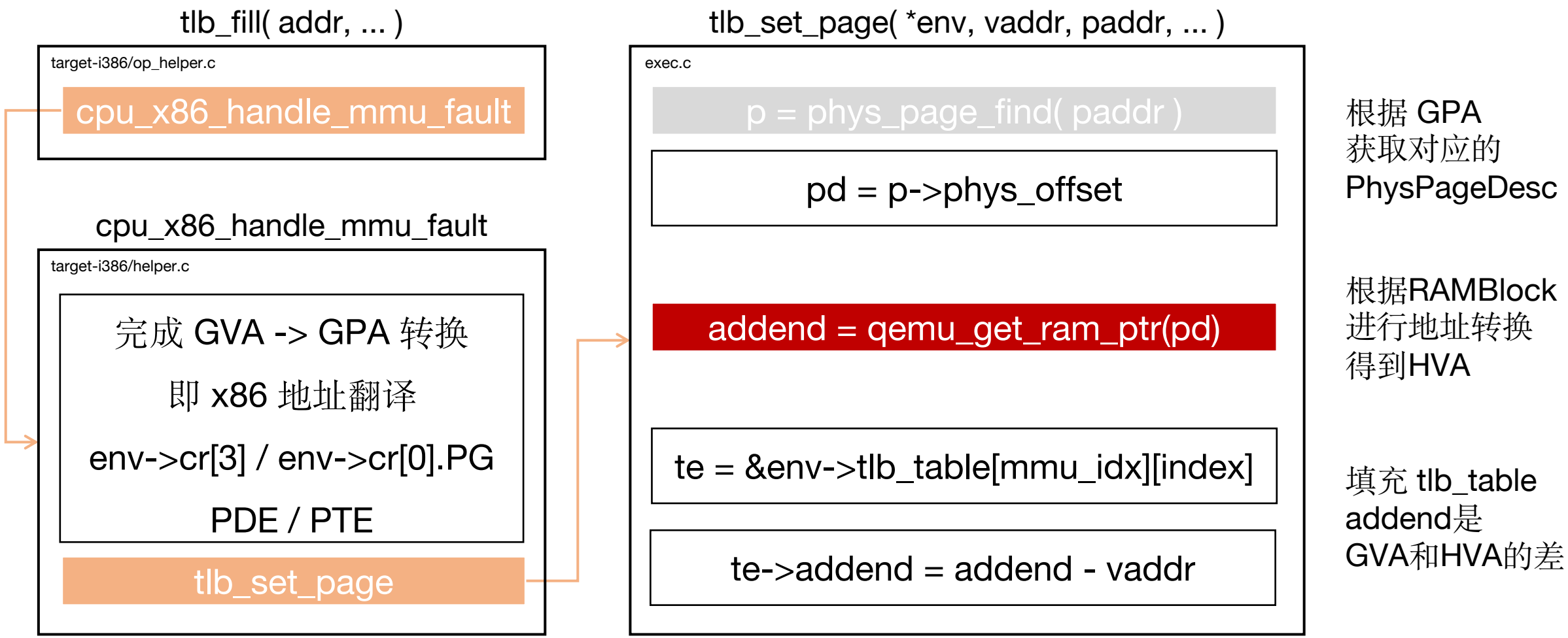
- void REGPARM glue(glue(__st, SUFFIX), MMUSUFFIX)(addr, val, mmu_idx) // softmmu_template.h
  - 胶水代码，通过重复定义宏 SUFFIX 来分别生成针对 8/16/32/64 bit 的操作
  - 主要功能
    1. 访问 env->tlb_table
    2. 若命中，根据访问类型，分别进行 IO 访存、非对齐访问、普通访存操作
    3. 若未命中，则执行 tlb_fill 来填充 env->tlb_table，然后从第 1 步重新开始

```
#define SUFFIX      l
#define MMUSUFFIX   _mu
```

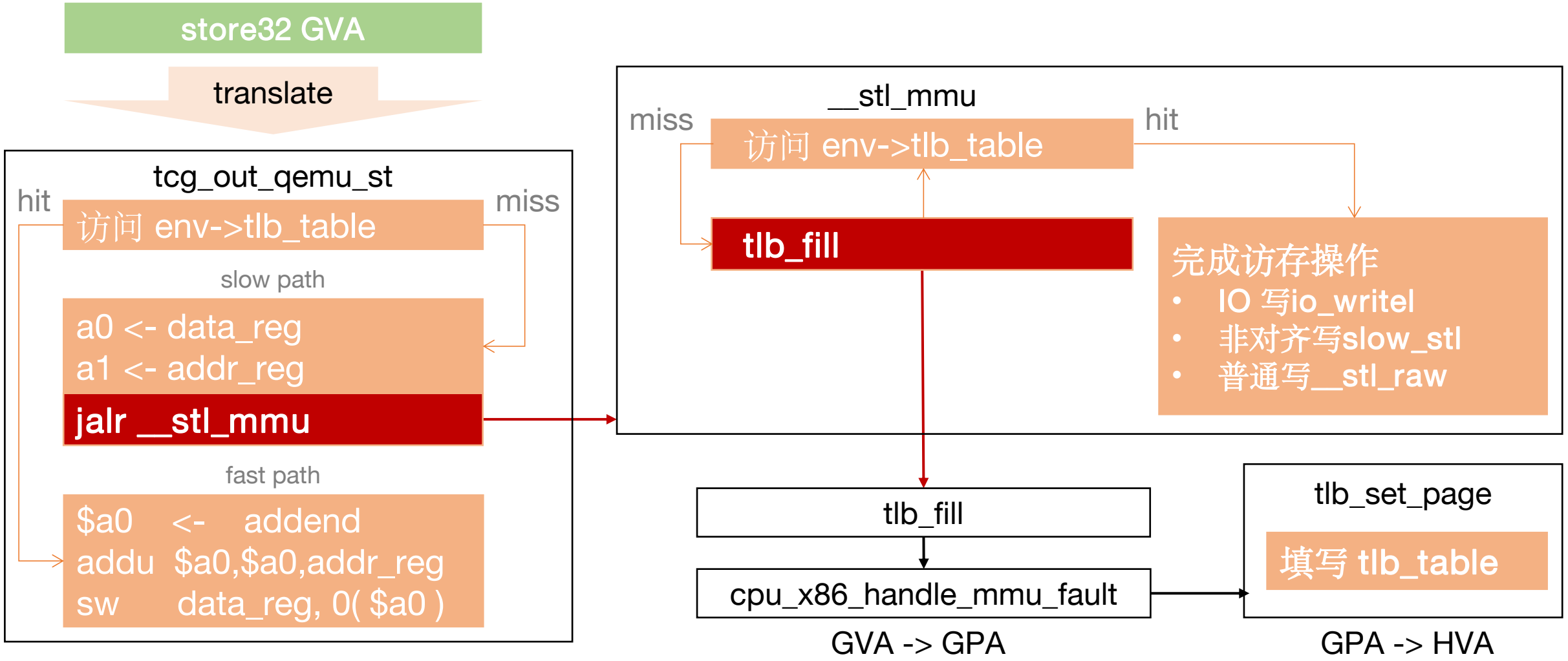

```
void __st_mmu( addr, val, mmu_idx )
```

访存指令：执行 __stl_mmu(addr,val,idx)

访存指令：执行 tlb_fill



访存指令：翻译与执行


store32 GVA
translate
tcg_out_qemu_st
hit
miss
访问 env->tlb_table
slow path
a0 <- data_reg
a1 <- addr_reg
jalr __stl_mmu
fast path
$a0 <- addend
addu $a0,$a0,addr_reg
sw data_reg, 0($a0)
__stl_mmu
miss
hit
访问 env->tlb_table
tlb_fill
完成访存操作
• IO 写io_writel
• 非对齐写slow_stl
• 普通写__stl_raw
tlb_fill
cpu_x86_handle_mmu_fault
GVA -> GPA
tlb_set_page
填写 tlb_table
GPA -> HVA

Target Physical Address Space

QEMU Virtual Address Space

Target Binary

.....
load GVA
.....
store GVA
.....

QEMU

内存虚拟化

Host Binary

.....
tlb_table / __ldl_mmu
.....
tlb_table / __stl_mmu
.....

Target Page Table

GVA1 --- GPA1
GVA2 --- GPA2
..... --- .....
.....

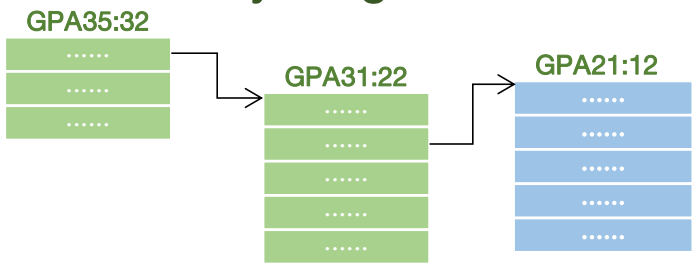
从 GVA 到 GPA

env->tlb_table

GVA	HVA
.....	.....
.....	.....
.....	.....
.....	.....

从 GVA 到 HVA

PhysPageDesc



从 GPA 到 HVA

RAMBlock

host_addr & size
host_addr & size
host_addr & size

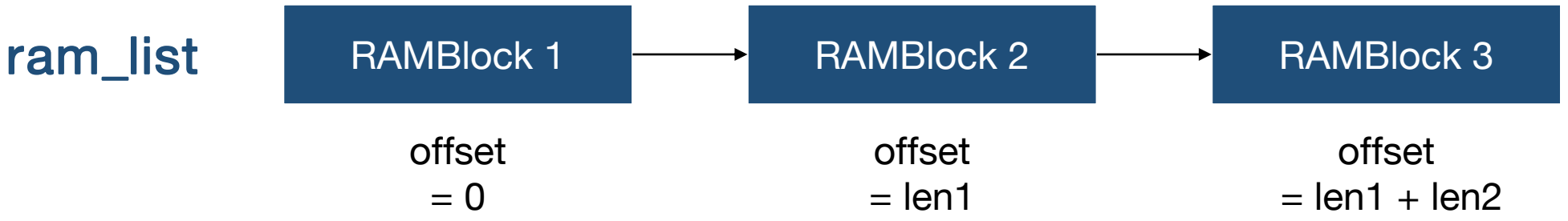
有哪几块 HVA 空间

其他问题说明

- 如何理解 RAMLIST 地址空间
- TranslationBlock 中的 page_addr 域有什么作用
- PhysPageDesc 和 PageDesc 有何异同

如何理解 RAMLIST 地址空间

- 相当于多了一层地址空间
  - GVA 地址空间 ： Target 的虚拟地址空间
  - GPA 地址空间 ： Target 的物理地址空间
  - RAMLIST 地址空间 ： QEMU 模拟的 RAM 地址空间，ram_addr_t
  - HVA 地址空间 ： QEMU 的虚拟地址空间



RAMLIST地址空间：所有 RAMBlock 组成的地址空间为 [0, SUM - 1]

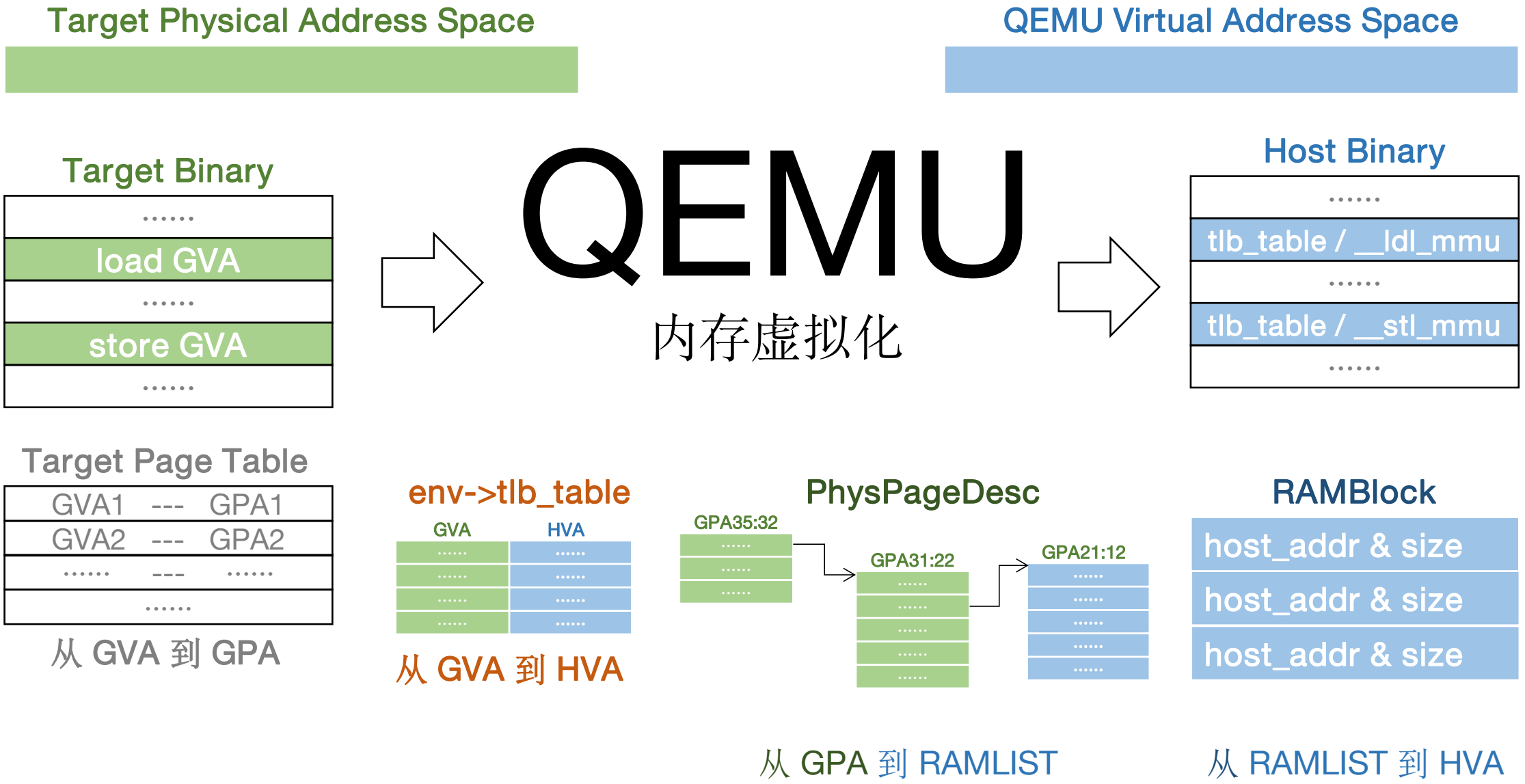
TranslationBlock 中的 page_addr 域

target_ulong	pc
target_ulong	cs_base
uint16_t	size
uint8_t	*tc_ptr
tb_page_addr_t	page_addr[2]
struct TranslationBlock	*phys_hash_next
uint16_t	tb_next_offset[2]
uint16_t	tb_jmp_offset[2]
unsigned long	tb_next[2]
struct TranslationBlock	*tb_loopup_cache[N]

存放的是 ram_addr_t

PhysPageDesc 的兄弟 PageDesc

- 类似的层次结构进行索引
- PhysPageDesc 以 GPA 为索引
  - 用于查找到对应的 ram_addr，进而得到 HVA
- PageDesc 以 ram_addr 为索引
  - 用于查找到该页对应的所有 TB


Target Physical Address Space
QEMU Virtual Address Space
Target Binary
.....
load GVA
.....
store GVA
.....
QEMU
内存虚拟化
Host Binary
.....
tlb_table / __ldl_mmu
.....
tlb_table / __stl_mmu
.....
Target Page Table
GVA1 --- GPA1
GVA2 --- GPA2
..... --- .....
.....
从 GVA 到 GPA
env->tlb_table
GVA
HVA
从 GVA 到 HVA
PhysPageDesc
GPA35:32
GPA31:22
GPA21:12
从 GPA 到 RAMLIST
RAMBlock
host_addr & size
host_addr & size
host_addr & size
从 RAMLIST 到 HVA