ISCAS 中国科学院软件研究所 Institute of Software Chinese Academy of Sciences
智能软件研究中心 Intelligent Software Research Center

基于QEMU 的模拟器扩展与实践

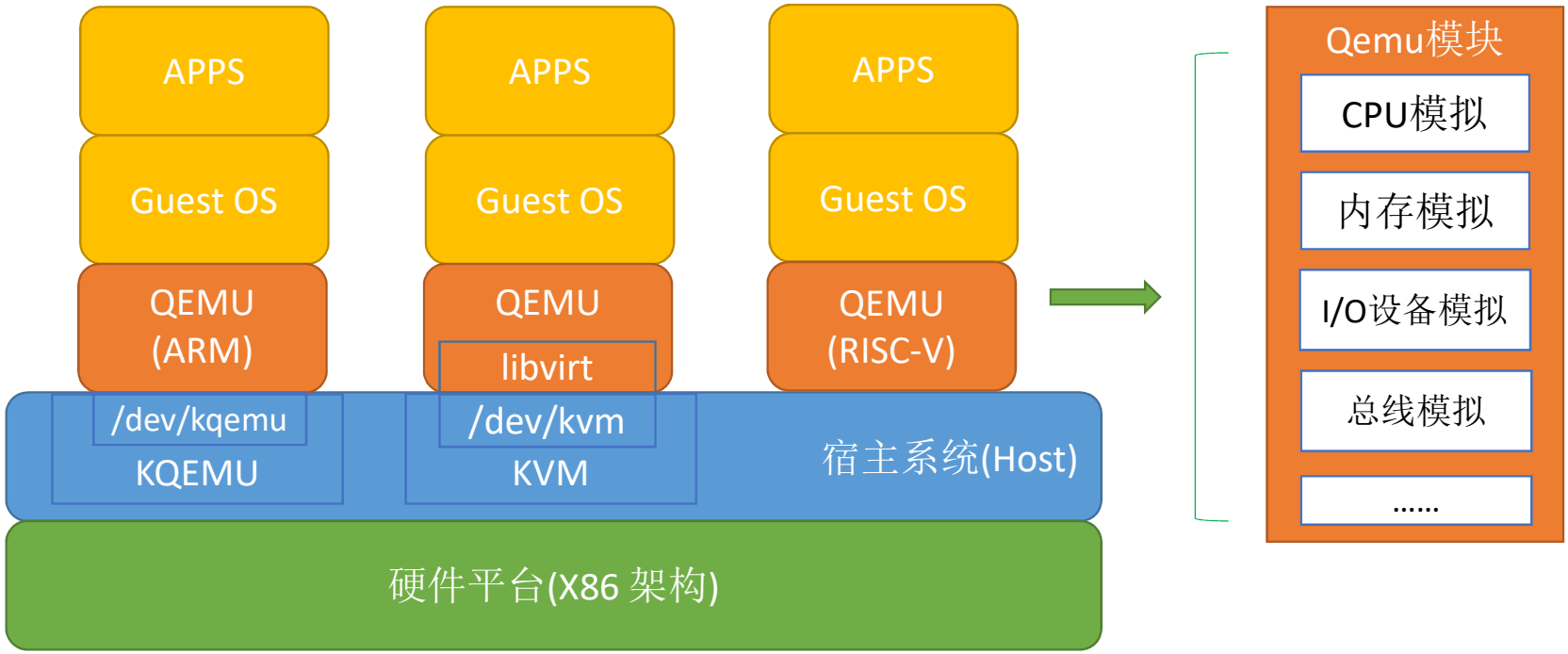
程序语言与编译技术实验室(PLCT) 王俊强

目录

QEMU简介

QEMU是一款开源的模拟器及虚拟机监管器(Virtual Machine Monitor, VMM)，通过动态二进制翻译来模拟CPU，并提供一系列的硬件模型，使guest os认为自己和硬件直接打交道，其实是同QEMU模拟出来的硬件打交道，QEMU再将这些指令翻译给真正硬件进行操作。

QEMU简介


APPS
APPS
APPS
Guest OS
Guest OS
Guest OS
QEMU
(ARM)
QEMU
libvirt
QEMU
(RISC-V)
/dev/kqemu
KQEMU
/dev/kvm
KVM
宿主系统(Host)
硬件平台(X86 架构)
Qemu模块
CPU模拟
内存模拟
I/O设备模拟
总线模拟
……

QEMU Mode

- 用户态仿真模拟器(User mode emulation)

  QEMU可以在当前CPU上执行被编译为支持其他CPU的程序（例如：QEMU可以在x86机器上执行一个ARM二进制可执行程序）。

QEMU Mode

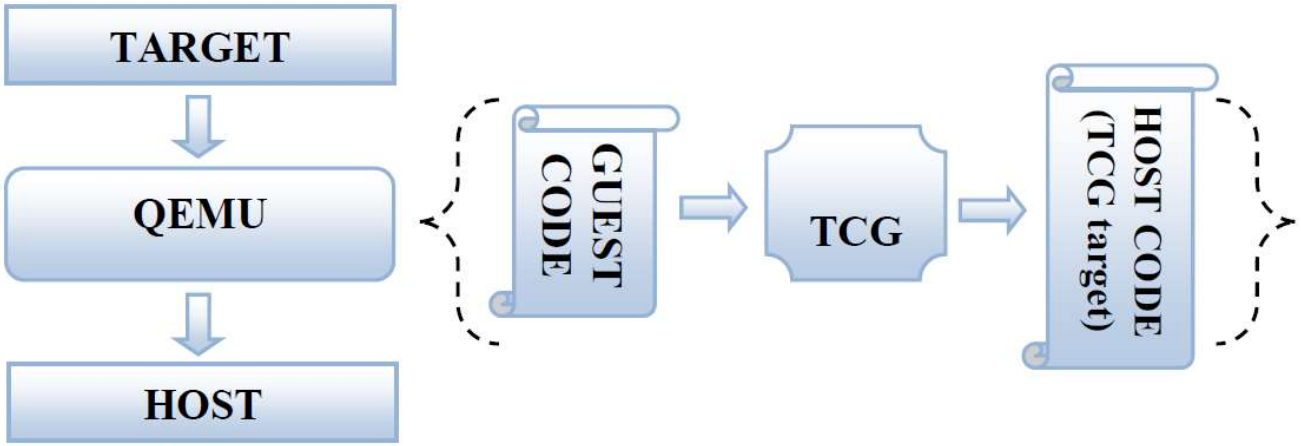
- 全系统仿真模拟器(Full system emulation)

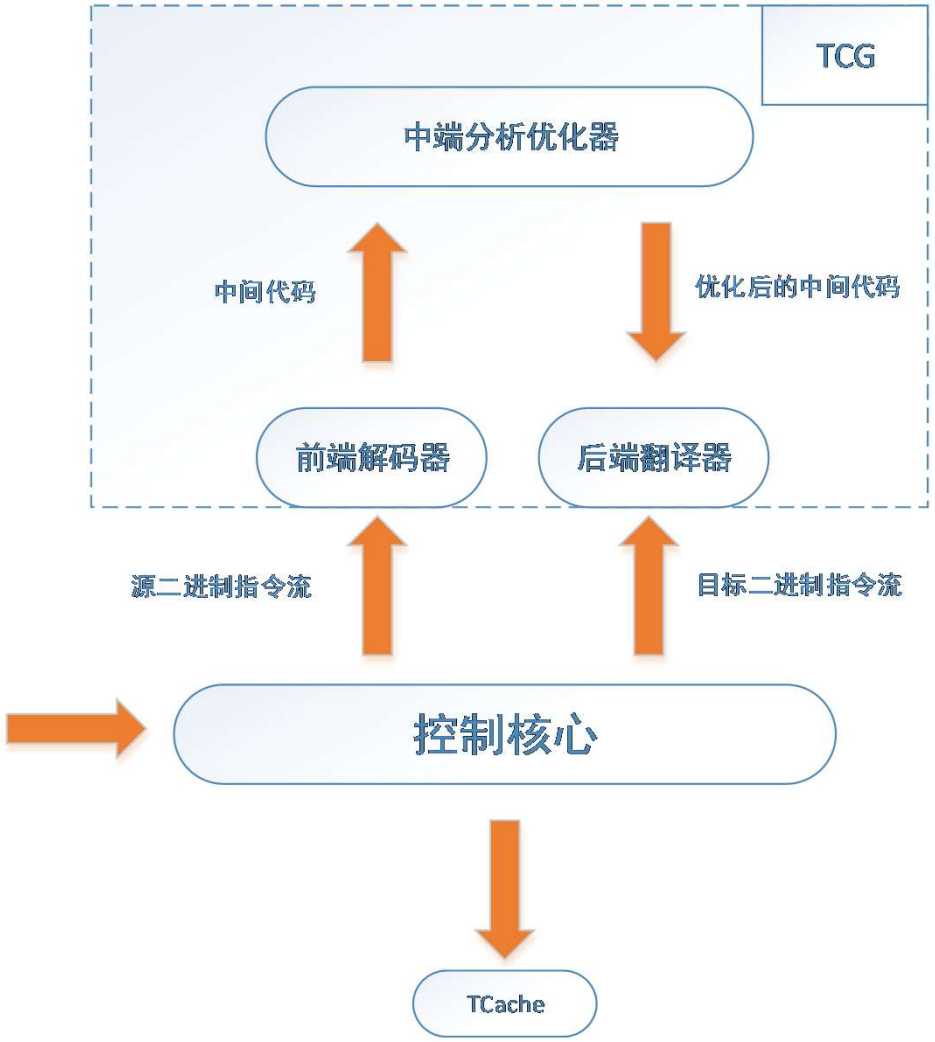
在这种模式下，qemu完整的仿真目标平台，此时，qemu就相当于一台完整的pc机，例如包括一个或多个处理器以及各种外围设备。

QEMU 工作原理

微代码生成器（TCG）

Tiny Code Generator（TCG）将源处理器机器代码转换为虚拟机运行所需的机器代码块(如x86机器代码块)




TCG
中端分析优化器
中间代码
优化后的中间代码
前端解码器
后端翻译器
源二进制指令流
目标二进制指令流
控制核心
TCache

TCG IR

```
typedef enum TCGOpcode {
#define DEF(name, oargs, iargs, cargs, flags) INDEX_op_ ## name,
#include "tcg-opc.h"
#undef DEF
    NB_OPS,
} TCGOpcode;
```

name: 指令名称
oargs: output 参数个数
iargs: input 参数个数
cargs: const 参数个数
flags: flag

源码IR示例：

```
/* arith */
DEF(add_i32, 1, 2, 0, 0)
DEF(sub_i32, 1, 2, 0, 0)
DEF(mul_i32, 1, 2, 0, 0)
DEF(div_i32, 1, 2, 0, IMPL(TCG_TARGET_HAS_div_i32))
```

IR opcode：

```
INDEX_op_add_i32
INDEX_op_sub_i32
INDEX_op_mul_i32
INDEX_op_div_i32
```

TCG IR

TCGOp 结构体：

```
typedef struct TCGOp {
    TCGOpcode opc   : 8;        /*  8 */
    /* Parameters for this opcode.  See below.  */
    unsigned param1 : 4;        /* 12 */
    unsigned param2 : 4;        /* 16 */
    /* Lifetime data of the operands.  */
    unsigned life   : 16;       /* 32 */
    /* Next and previous opcodes.  */
    QTAILQ_ENTRY(TCGOp) link;
    /* Arguments for the opcode.  */
    TCGArg args[MAX_OPC_PARAM];
} TCGOp;
```

TCG IR

Qemu源码中，对IR类别的七种定义:

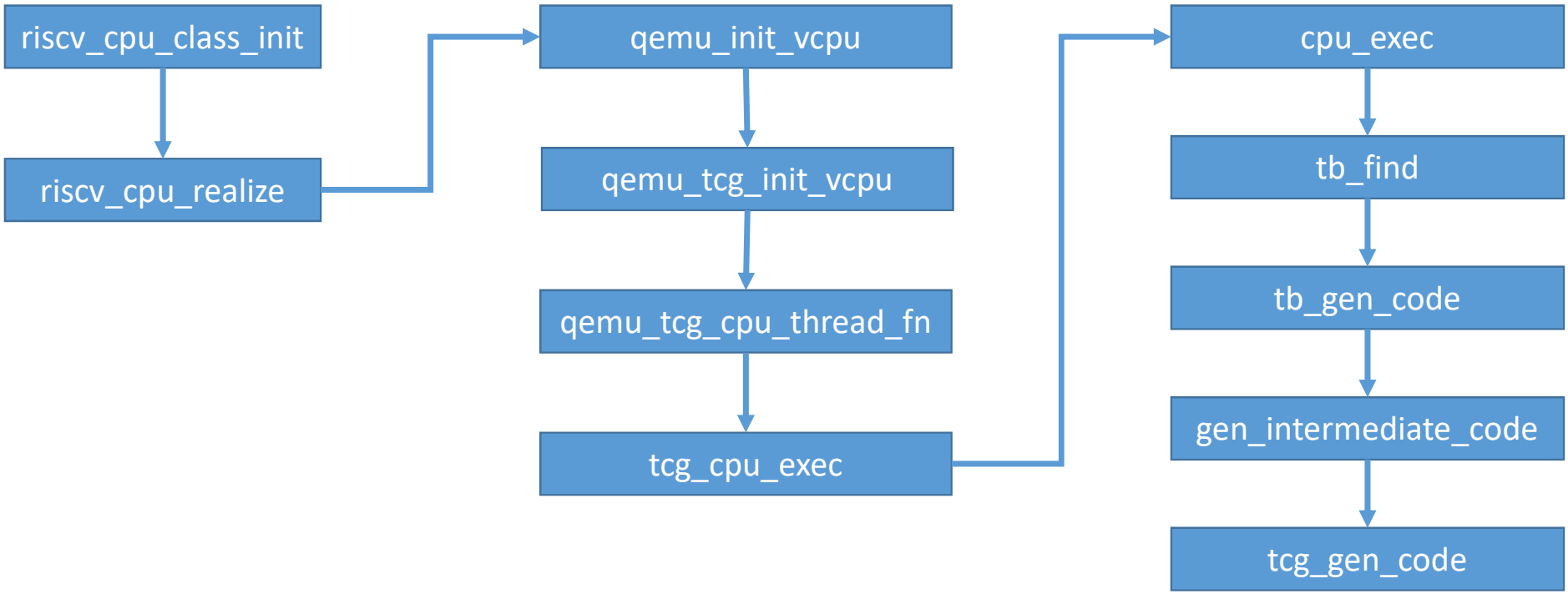
IR类别	示例(32/64)
predefined ops	discard,set_label, call, br,etc.
load/store	ld,st ,etc.
arith	add, sub, mul, div,etc.
shifts/rotates	shl, shr, sar,bswap ,etc.
size changing ops	ext_i32_i64, extrl_i64_i32,extrh_i64_i32 ,etc.
QEMU specific	insn_start, exit_tb, goto_tb, qemu_ld, qemu_st ,etc.
Host vector support	mov_vec, ld_vec, st_vec,add_vec,not_vec ,etc.

TCG转换流程

```
typedef struct TranslationBlock TranslationBlock;
typedef struct TBContext TBContext;

struct TBContext {
    struct qht htable;
    /* statistics */
    unsigned tb_flush_count;
};

typedef struct TCGTemp {
        ……
} TCGTemp;

typedef struct TCGContext TCGContext;
```

TranslationBlock主要结构
pc
cs_base
tb_tc.ptr
orig_tb
jmp_lock
jmp_list_head
jmp_list_next[2]
jmp_dest[2]

TCG转换流程



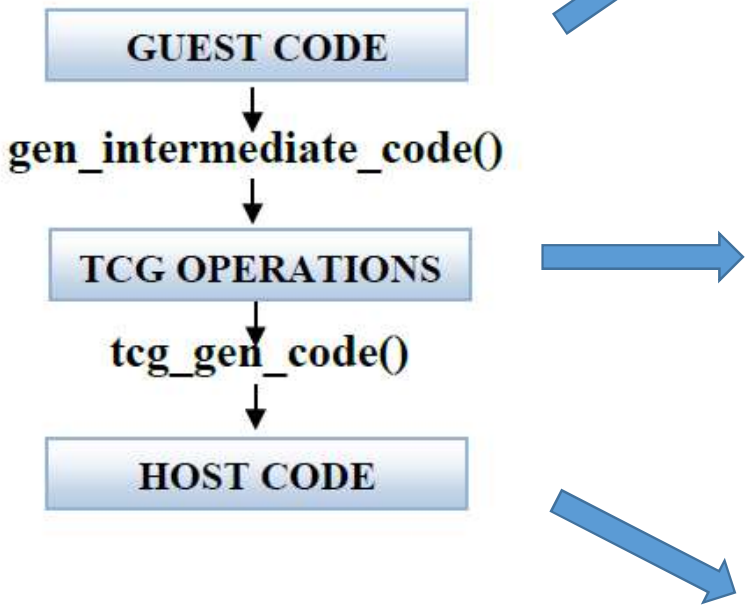
指定CPU设备初始化

tcg翻译和执行的线程

TB块的索引翻译

TCG转换流程

核心过程:



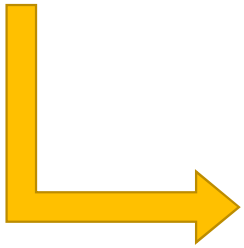

```
push %ebp
mov %esp,%ebp
not %eax
add %eax,%edx
mov %edx,%eax
xor $0x55555555, %eax
pop %ebp
ret
```

```
ld_i 32 tmp2,env,$0x1 0
qemu_ld32u
tmp0,tmp2,$0xffffffff
ld_i 32 tmp4,env,$0x10
movi_I 32 tmp14,$0x4
add_i 32 tmp4,tmp4,tmp14
st_i 32 tmp4 ,env,$0x10
st_i32 tmp0,env,$0x20
movi_i 32 cc_op,$0x18
exit_tb $0x0
```

```
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
```

TCG转换流程(生成IR)

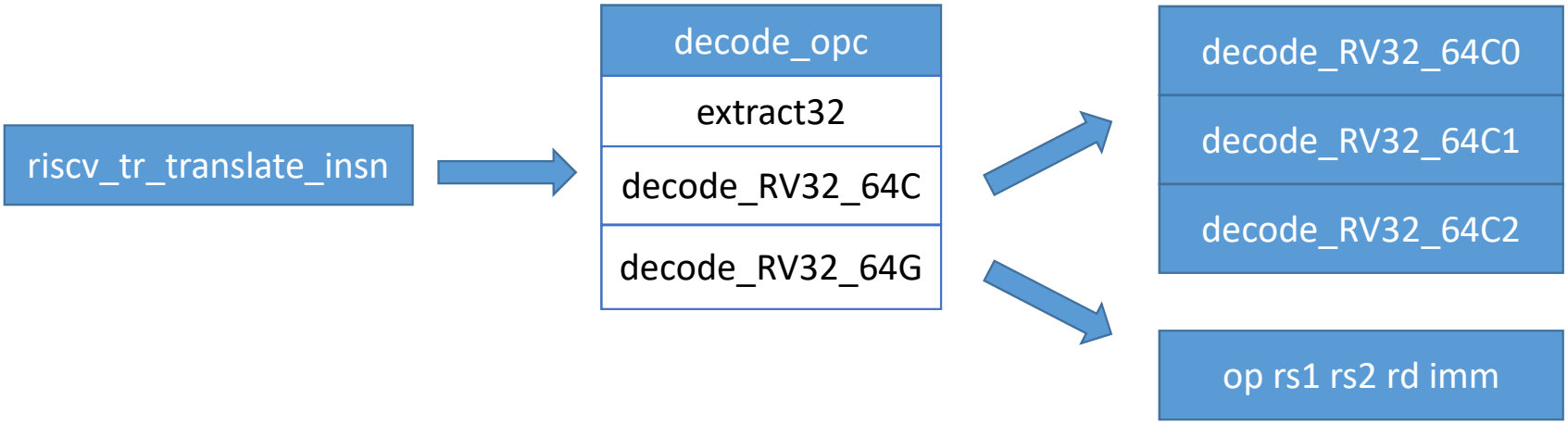
gen_intermediate_code



```
typedef struct TranslatorOps {
    void (*init_disas_context)(DisasContextBase *db, CPUState *cpu);
    void (*tb_start)(DisasContextBase *db, CPUState *cpu);
    void (*insn_start)(DisasContextBase *db, CPUState *cpu);
    bool (*breakpoint_check)(DisasContextBase *db, CPUState *cpu,
                             const CPUBreakpoint *bp);
    void (*translate_insn)(DisasContextBase *db, CPUState *cpu);
    void (*tb_stop)(DisasContextBase *db, CPUState *cpu);
    void (*disas_log)(const DisasContextBase *db, CPUState *cpu);
} TranslatorOps;
```

translate_insn  riscv_tr_translate_insn

TCG转换流程(生成IR)



示例：ADD

funct7(25-31)	rs2(20-24)	rs1(15-19)	funct3(12-14)	rd(7-11)	opcode(0-6)
0000000	rs2	rs1	000	rd	0110011

```
enum {
    /* rv32i, rv64i, rv32m */
    ……
    OPC_RISC_ARITH_IMM  = (0x13),
    OPC_RISC_ARITH      = (0x33),
    OPC_RISC_FENCE      = (0x0F),
    OPC_RISC_SYSTEM     = (0x73),
}
```

TCG转换流程(生成IR)

```
static void gen_arith(DisasContext *ctx, uint32_t opc, int rd, int rs1,
        int rs2)
{
    TCGv source1, source2, cond1, cond2, zeroreg, resultopt1;
    source1 = tcg_temp_new();
    source2 = tcg_temp_new();
    gen_get_gpr(source1, rs1);
    gen_get_gpr(source2, rs2);

    switch (opc) {
    CASE_OP_32_64(OPC_RISC_ADD):
        tcg_gen_add_tl(source1, source1, source2);
        break;
    CASE_OP_32_64(OPC_RISC_SUB):
        tcg_gen_sub_tl(source1, source1, source2);
        break;
```

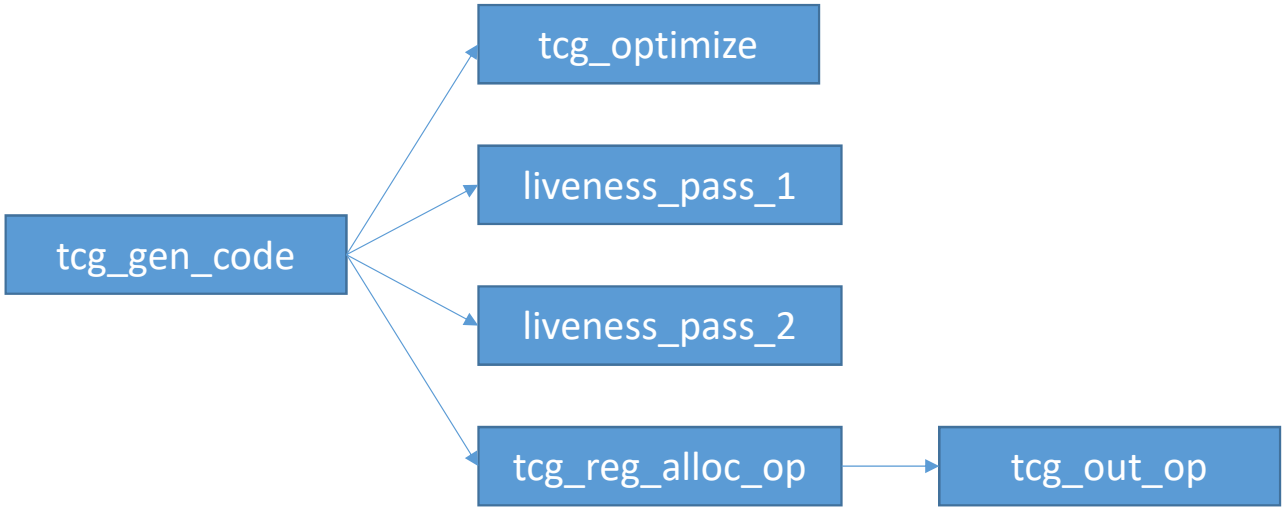
TCG转换流程(生成IR)

```
#define tcg_gen_add_tl tcg_gen_add_i32
#define tcg_gen_addi_tl tcg_gen_addi_i32
#define tcg_gen_sub_tl tcg_gen_sub_i32
```

```
static inline void tcg_gen_add_i32(TCGv_i32 ret, TCGv_i32 arg1, TCGv_i32 arg2)
{
    tcg_gen_op3_i32(INDEX_op_add_i32, ret, arg1, arg2);
}
```

```
TCGOp *op = tcg_emit_op(opc);
op->args[0] = a1;
op->args[1] = a2;
op->args[2] = a3;
```

TCG转换流程(生成Host Code)



i386 Opcode示例：

```
#define OPC_PADDB        (0xfc | P_EXT | P_DATA16)
#define OPC_PADDW        (0xfd | P_EXT | P_DATA16)
#define OPC_PADDD        (0xfe | P_EXT | P_DATA16)
#define OPC_PADDQ        (0xd4 | P_EXT | P_DATA16)
```

TCG执行

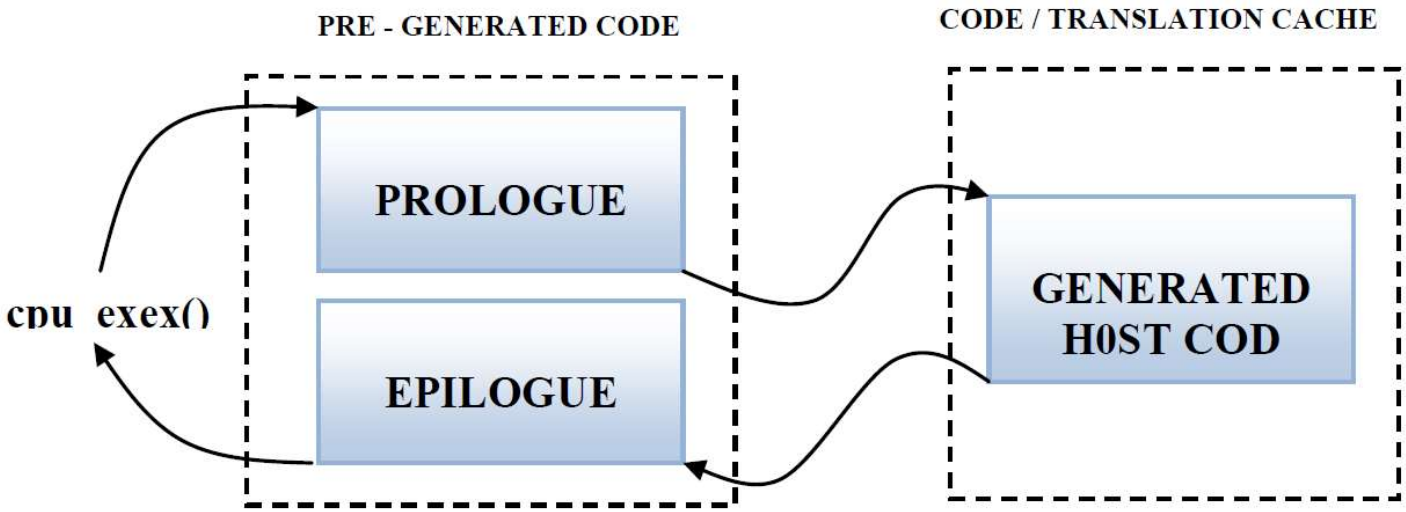
cpu_tb_exec

↓

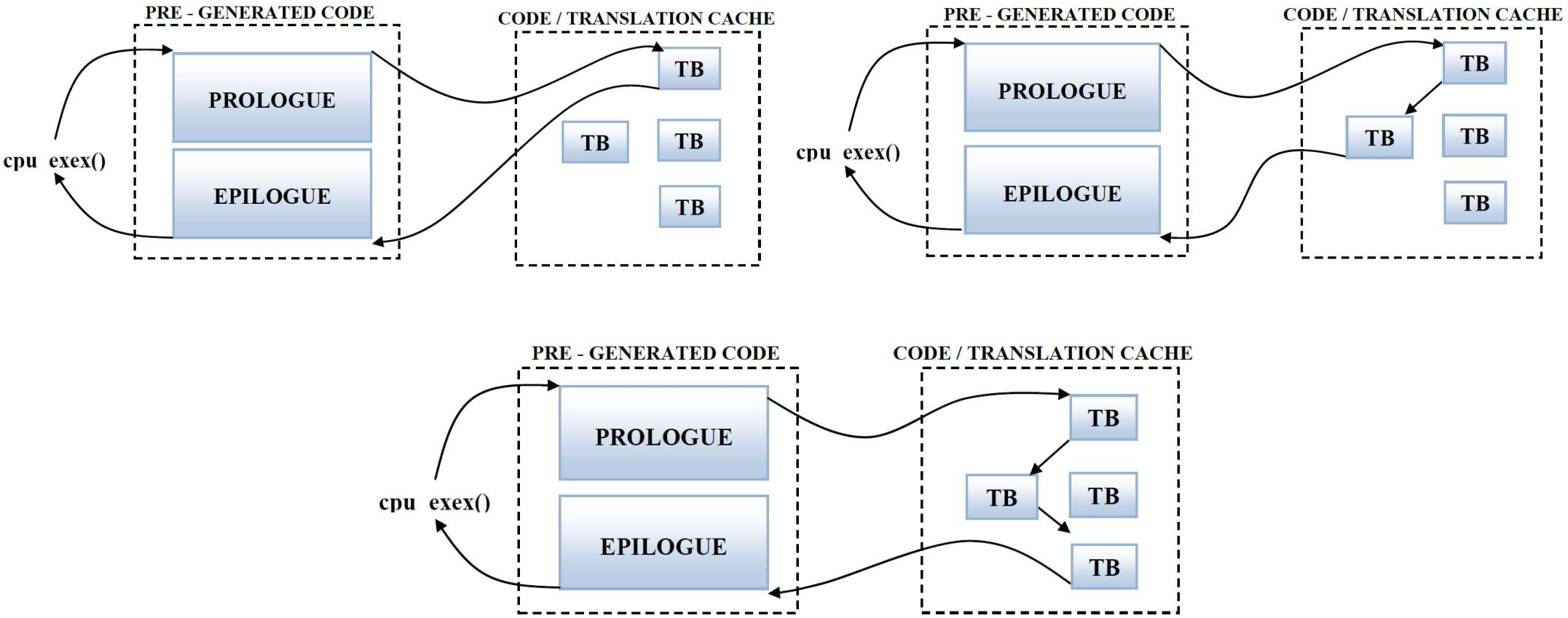
tcg_qemu_tb_exec

函数原型

```
uintptr_t tcg_qemu_tb_exec(CPUArchState *env, uint8_t *tb_ptr);
```



TCG执行


PRE - GENERATED CODE
CODE / TRANSLATION CACHE
PROLOGUE
EPILOGUE
cpu exex()
TB
TB
TB
TB
PRE - GENERATED CODE
CODE / TRANSLATION CACHE
PROLOGUE
EPILOGUE
cpu exex()
TB
TB
TB
TB
PRE - GENERATED CODE
CODE / TRANSLATION CACHE
PROLOGUE
EPILOGUE
cpu exex()
TB
TB
TB
TB

QEMU与QOM

QOM(QEMU Object Module)是用C语言实现的一种面向对象的编程模型。

- 各种CPU架构和SOC的模拟和实现
  - CPU架构模拟　CPU通用属性　和 特有的属性
- 模拟device与bus的关系
  - device➔bus➔device
  - device➔不同的bus➔device
  - bus➔多个device

```
class MyClass {
public:
    int a;
    void set_A(int a);
}
```

```
struct MyClass {
    int a;
    void (*set_A)(MyClass *this, int a);
};
```

QEMU与QOM

主要结构体：

- Object(**The base for all objects**)
- ObjectClass(**The base for all classes**)
- TypeInfo(工具)
- TypeImpl
- InterfaceInfo
- InterfaceClass

```
struct Object
{
    /*< private >*/
    ObjectClass *class;
    ObjectFree *free;
    GHashTable *properties;
    uint32_t ref;
    Object *parent;
};
```

```
struct ObjectClass
{
    /*< private >*/
    Type type;
    GSList *interfaces;
    const char *object_cast_cache[OBJECT_CLASS_CAST_CACHE];
    const char *class_cast_cache[OBJECT_CLASS_CAST_CACHE];
    ObjectUnparent *unparent;
    GHashTable *properties;
};
```

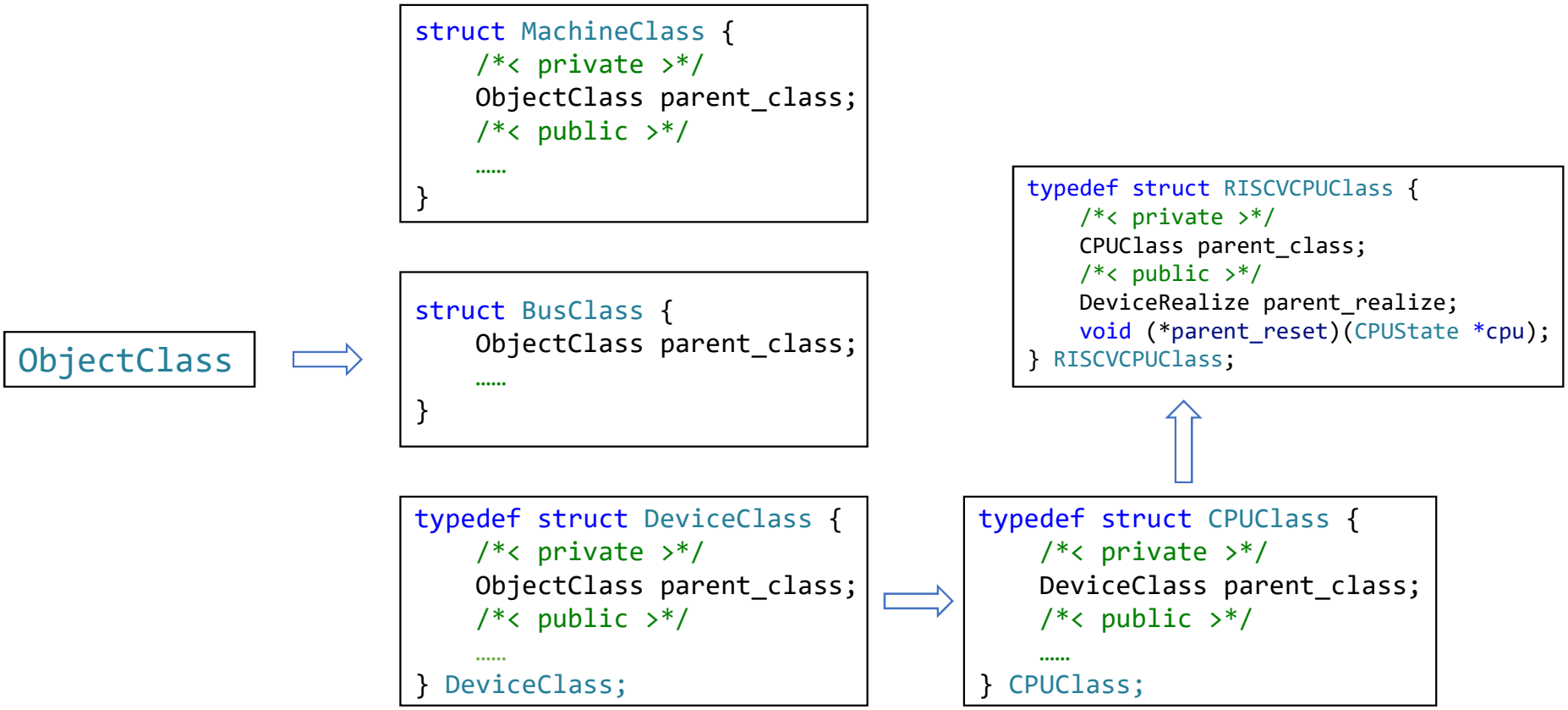
QEMU与QOM

主要结构体：

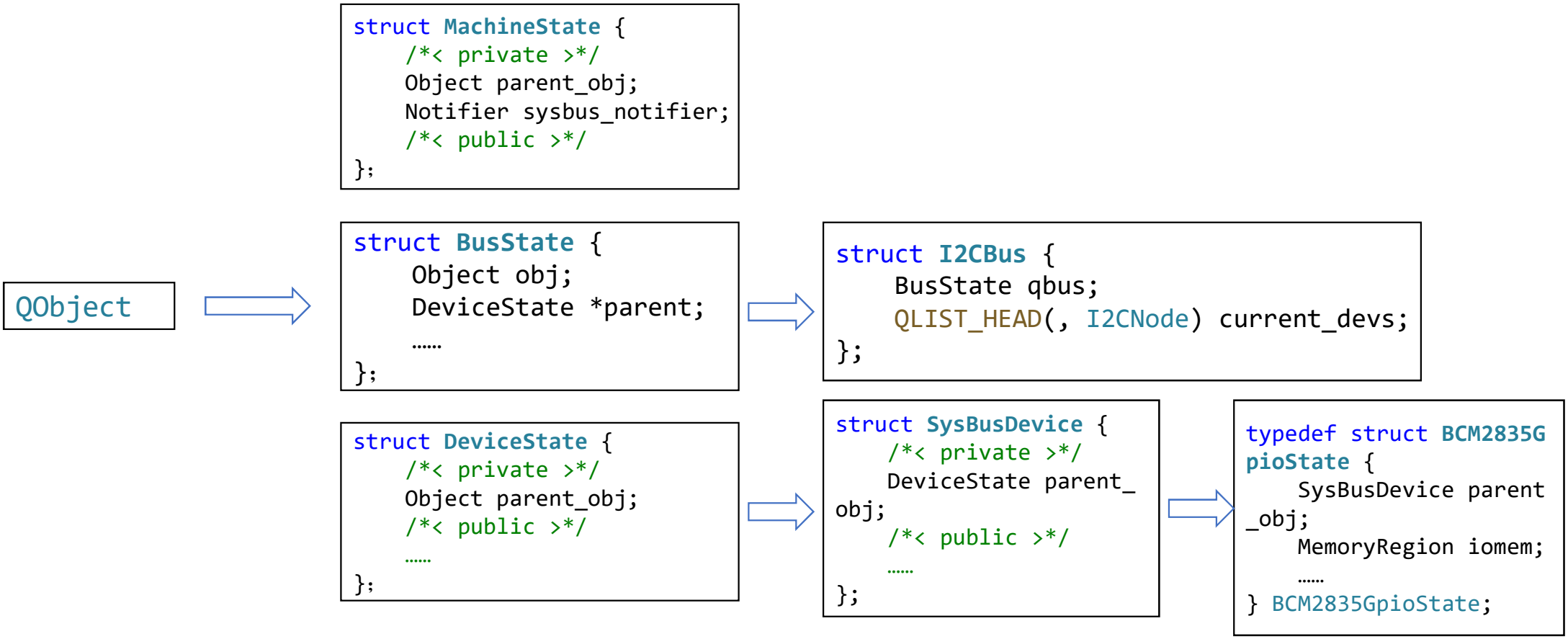
- Object(**The base for all objects**)
- ObjectClass(**The base for all classes**)
- TypeInfo(工具)
- TypeImpl
- InterfaceInfo
- InterfaceClass

```
struct TypeInfo
{
    const char *name;
    const char *parent;

    size_t instance_size;
    void (*instance_init)(Object *obj);
    void (*instance_post_init)(Object *obj);
    void (*instance_finalize)(Object *obj);

    bool abstract;
    size_t class_size;

    void (*class_init)(ObjectClass *klass, void *data);
    void (*class_base_init)(ObjectClass *klass, void *data);
    void (*class_finalize)(ObjectClass *klass, void *data);
    void *class_data;

    InterfaceInfo *interfaces;
};
```

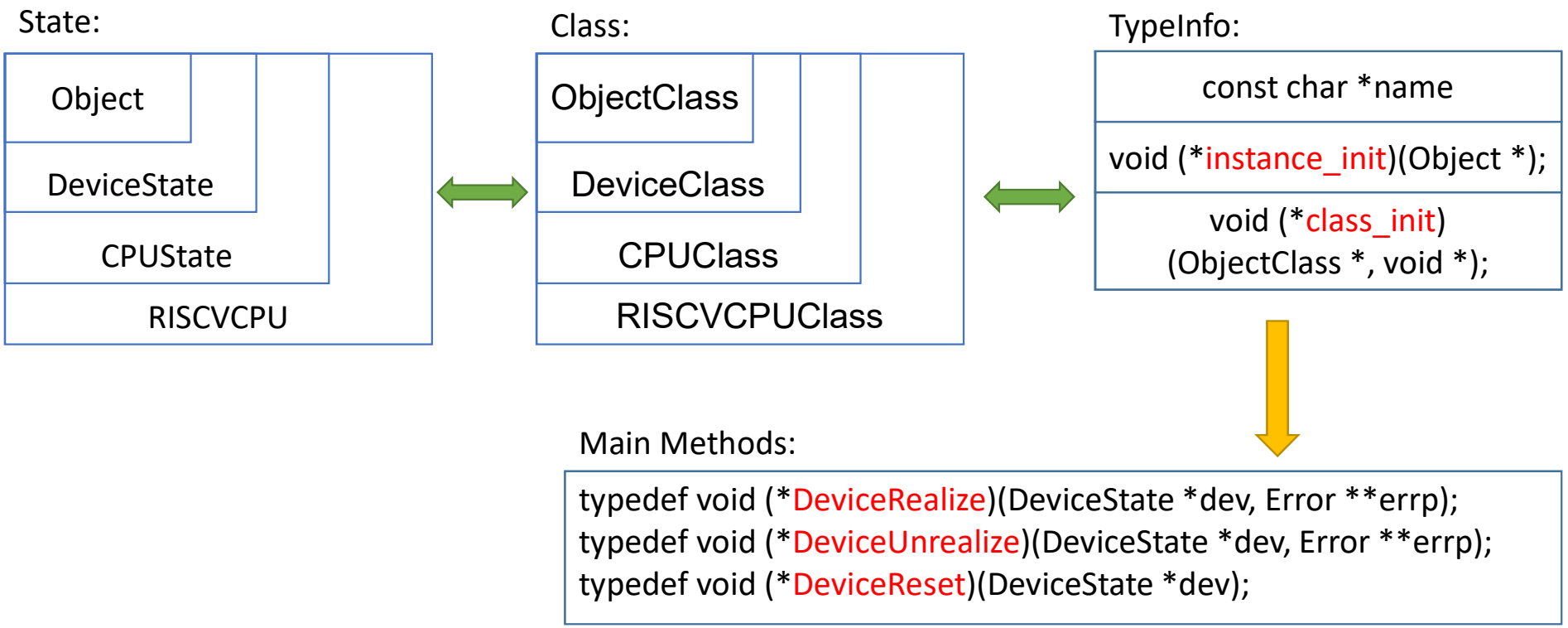
QEMU与QOM


ObjectClass
struct MachineClass {
/*< private >*/
ObjectClass parent_class;
/*< public >*/
……
}
struct BusClass {
ObjectClass parent_class;
……
}
typedef struct DeviceClass {
/*< private >*/
ObjectClass parent_class;
/*< public >*/
……
} DeviceClass;
typedef struct CPUClass {
/*< private >*/
DeviceClass parent_class;
/*< public >*/
……
} CPUClass;
typedef struct RISCVCPUClass {
/*< private >*/
CPUClass parent_class;
/*< public >*/
DeviceRealize parent_realize;
void (*parent_reset)(CPUState *cpu);
} RISCVCPUClass;

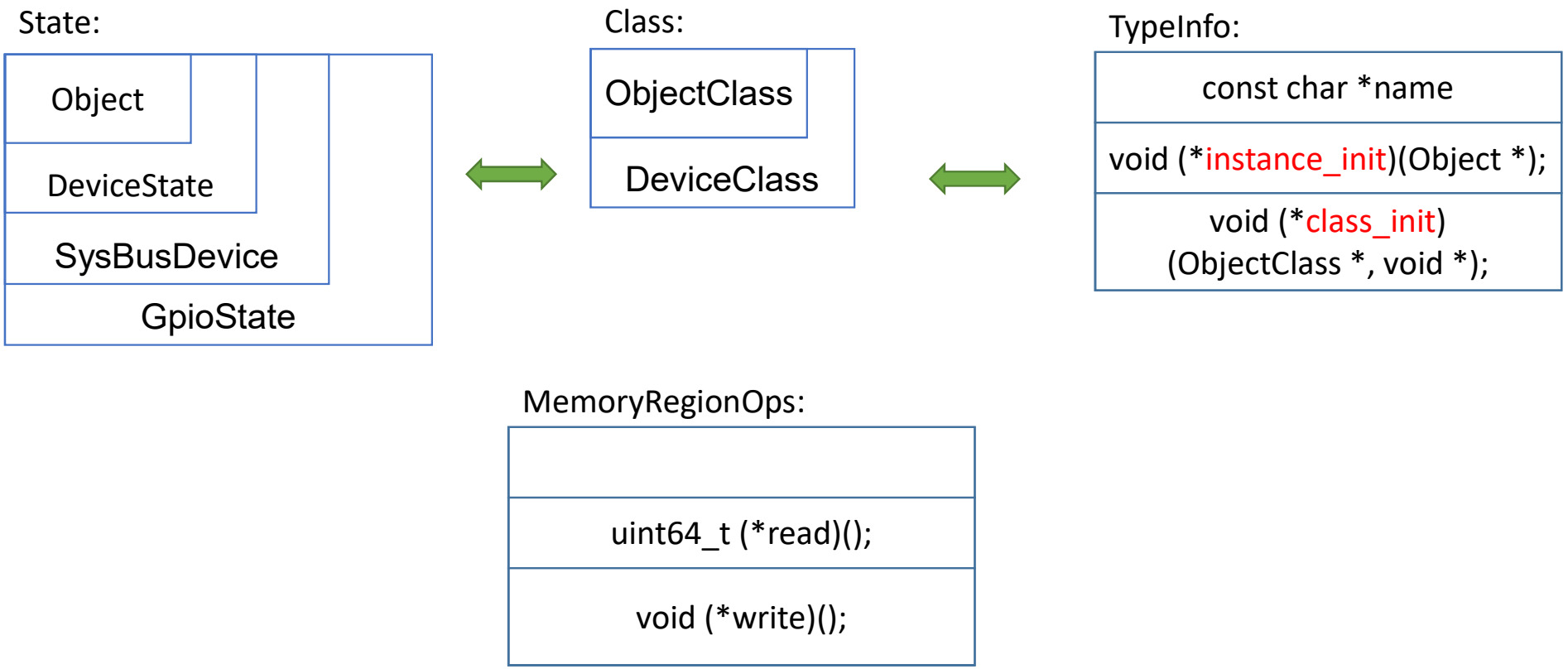
QEMU与QOM


QObject
struct MachineState {
/*< private >*/
Object parent_obj;
Notifier sysbus_notifier;
/*< public >*/
};
struct BusState {
Object obj;
DeviceState *parent;
……
};
struct I2CBus {
BusState qbus;
QLIST_HEAD(, I2CNode) current_devs;
};
struct DeviceState {
/*< private >*/
Object parent_obj;
/*< public >*/
……
};
struct SysBusDevice {
/*< private >*/
DeviceState parent_obj;
/*< public >*/
……
};
typedef struct BCM2835GpioState {
SysBusDevice parent_obj;
MemoryRegion iomem;
……
} BCM2835GpioState;

QEMU与CPU


State:
Object
DeviceState
CPUState
RISCVCPU
Class:
ObjectClass
DeviceClass
CPUClass
RISCVCPUClass
TypeInfo:
const char *name
void (*instance_init)(Object *);
void (*class_init)
(ObjectClass *, void *);
Main Methods:
typedef void (*DeviceRealize)(DeviceState *dev, Error **errp);
typedef void (*DeviceUnrealize)(DeviceState *dev, Error **errp);
typedef void (*DeviceReset)(DeviceState *dev);

QEMU与外设


State:
Object
DeviceState
SysBusDevice
GpioState
Class:
ObjectClass
DeviceClass
TypeInfo:
const char *name
void (*instance_init)(Object *);
void (*class_init)
(ObjectClass *, void *);
MemoryRegionOps:
uint64_t (*read)();
void (*write)();

QEMU外设示例

第一步：定义设备

第二步：添加设备注册

第三步：类初始函数

第四步：初始化函数

第五步：功能函数

```
typedef struct PLCTWatchdogState {
    PCIDevice dev;
    uint8_t activated;
    uint8_t triggered;
    uint32_t missed_ticks;
    QEMUTimer *watchdog_timer;
    uint32_t expiration_ticks;
    MemoryRegion io;
} PLCTWatchdogState;

#define TYPE_PLCT_WATCHDOG "plct-watchdog"
#define PLCT_WATCHDOG(obj) \
        OBJECT_CHECK(PLCTWatchdogState, (obj), TY
PE_PLCT_WATCHDOG)
```

QEMU外设示例

第一步：
定义设备

第二步：
添加设备注册

第三步:
类初始函数

第四步:
初始化函数

第五步:
功能函数

注册：

type_init(function)
module_init(function, type)
register_module_init(function, type);
QTAILQ_INSERT_TAIL(l, e, node);

调用：

module_call_init(module_init_type type)
QTAILQ_FOREACH(e, l, node)
type_register_static(TypeInfo)

```
static TypeInfo plct_watchdog_info = {
    .name          = TYPE_PLCT_WATCHDOG,
    .parent        = TYPE_PCI_DEVICE,
    .instance_init = wdt_initfn,
    .instance_size = sizeof(PLCTWatchdogState),
    .class_init    = wdt_class_init,
};
static void register_types(void)
{
    type_register_static(&plct_watchdog_info);
}
type_init(register_types)
```

QEMU外设示例

第一步:
定义设备

第二步:
添加设备注册

第三步:
类初始函数

第四步:
初始化函数

第五步:
功能函数

```
static void wdt_class_init(ObjectClass *klass, void *data)
{
    DeviceClass *dc = DEVICE_CLASS(klass);
    PCIDeviceClass *k = PCI_DEVICE_CLASS(klass);
    k->init = wdt_realize;
    k->exit = wdt_unrealize;
    k->vendor_id = PCI_VENDOR_ID_REDHAT_QUMRANET;
    k->device_id = 0x0101;
    k->revision = 0x01;
    k->class_id = PCI_CLASS_SYSTEM_OTHER;
    dc->reset = wdt_reset;
    dc->vmsd = &vmstate_wdt;
    dc->props = wdt_properties;
}
```

VMStateDescription vmstate_wdt: 用于热启动/快照状态保存

Property wdt_properties: 命令接口导出和属性配置

QEMU外设示例

第一步：
定义设备

第二步：
添加设备注册

第三步：
类初始函数

**第四步：
初始化函数**

第五步：
功能函数

```
static void wdt_initfn(Object *obj)
{
    PLCTWatchdogState *s = PLCT_WATCHDOG(obj);
    memory_region_init_io(
&s->io, OBJECT(s), &wdt_io_ops, s, "plct-watchdog-io", 16);
    s->watchdog_timer =
    timer_new_ms(QEMU_CLOCK_REALTIME, wdt_timer_event, s);
}

static const MemoryRegionOps wdt_io_ops = {
    .read = wdt_io_read,
    .write = wdt_io_write,
    .endianness = DEVICE_LITTLE_ENDIAN,
};
```

QEMU外设示例

第一步：
定义设备

第二步：
添加设备注册

第三步：
类初始函数

第四步：
初始化函数

第五步：
功能函数

```
static uint64_t wdt_io_read(void *opaque, hwaddr addr,
 unsigned size)
{
    PLCTWatchdogState *s = PLCT_WATCHDOG(opaque);
    ……
    return 0;
}

static void wdt_io_write(void *opaque, hwaddr addr,uin
t64_t val, unsigned size)
{
    PLCTWatchdogState *s = PLCT_WATCHDOG(opaque);
     ……
}
```

谢谢