

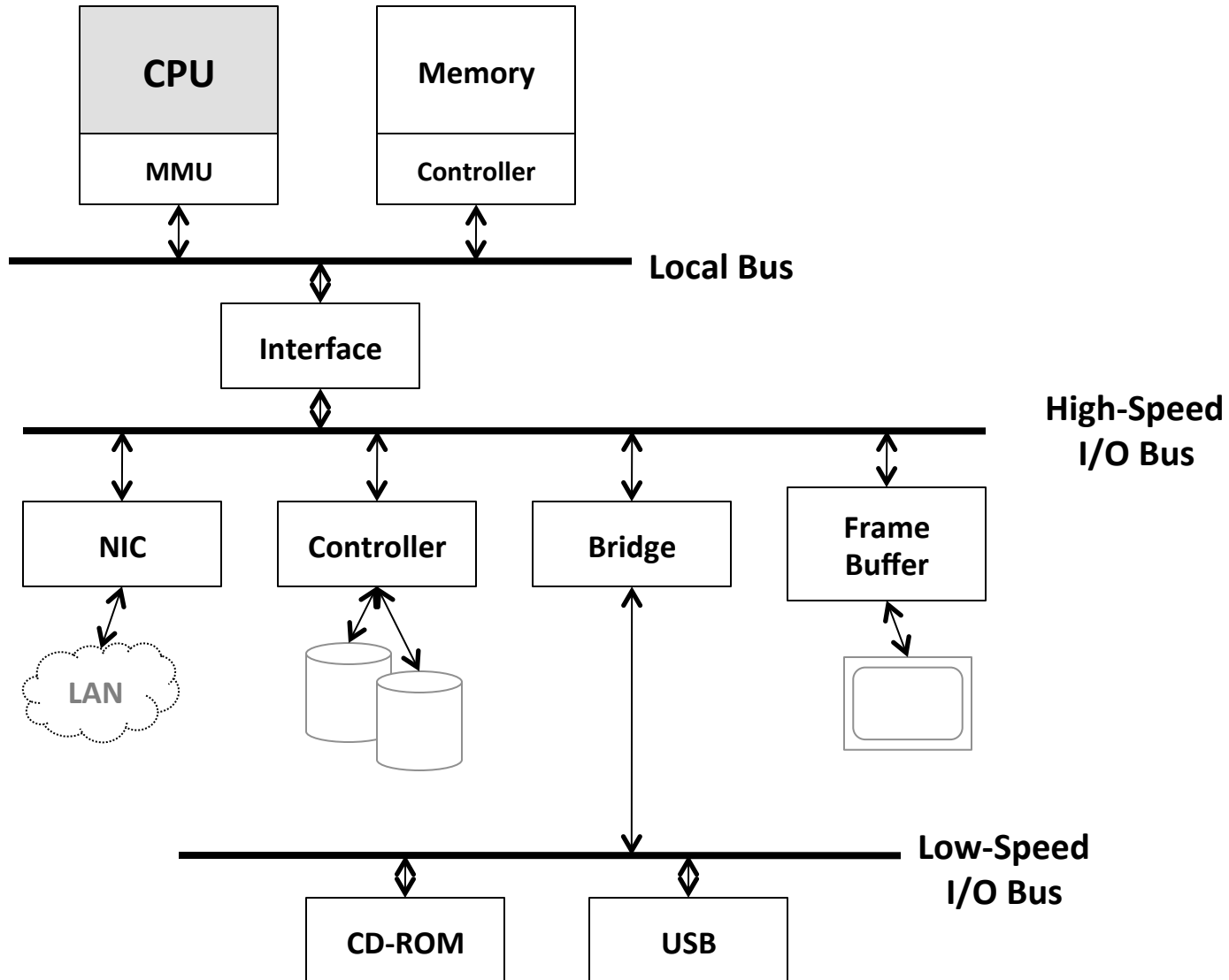
E6998 - Virtual Machines
Lecture 2
CPU Virtualization

Scott Devine
VMware, Inc.

Outline

- **CPU Background**
- **Virtualization Techniques**
 - System ISA Virtualization
 - Instruction Interpretation
 - Trap and Emulate
 - Binary Translation
 - Hybrid Models

Computer System Organization



CPU Organization

- **Instruction Set Architecture (ISA)**

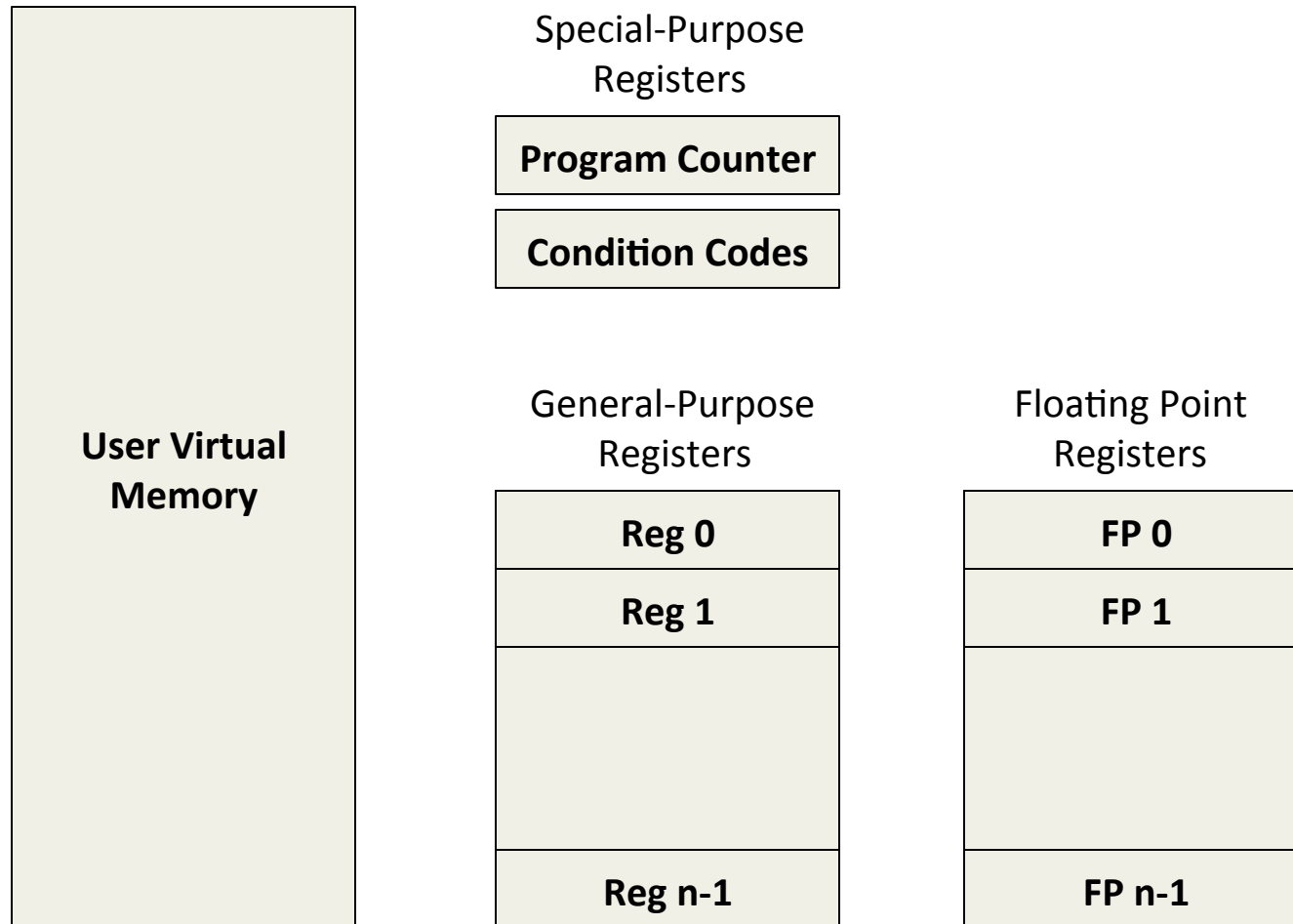
Defines:

- the state visible to the programmer
 - registers and memory
- the instruction that operate on the state

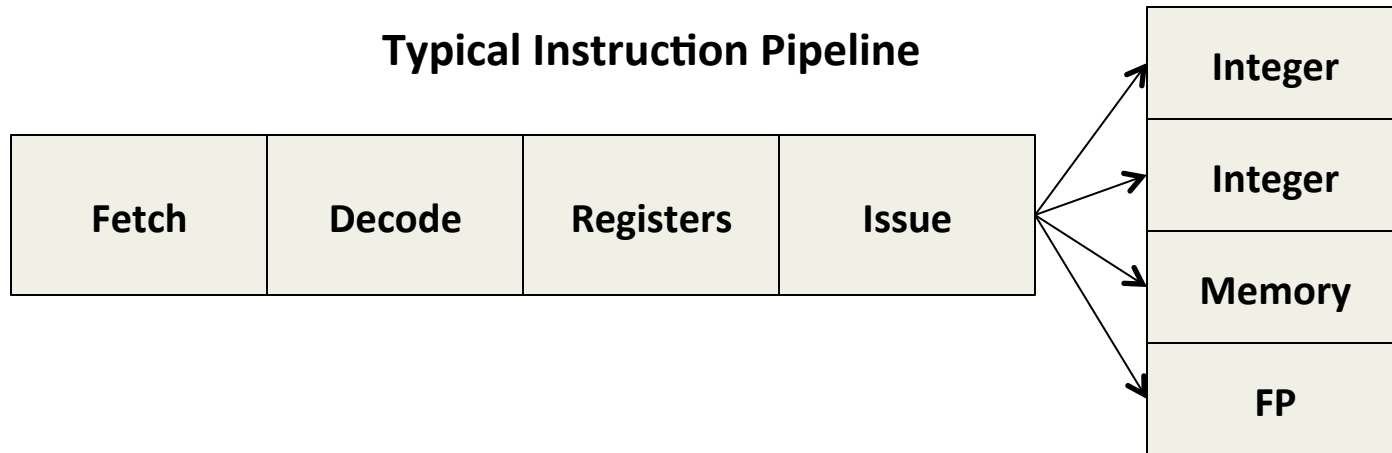
- **ISA typically divided into 2 parts**

- User ISA
 - Primarily for computation
- System ISA
 - Primarily for system resource management

User ISA - State



User ISA – Instructions

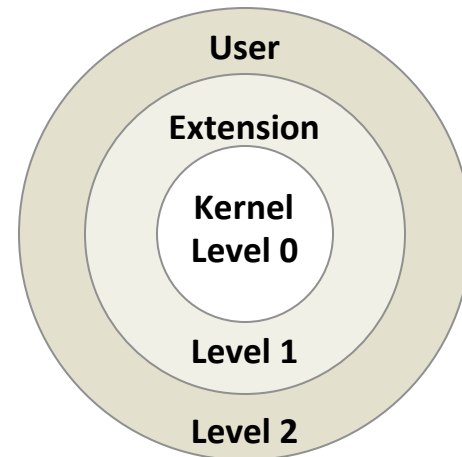
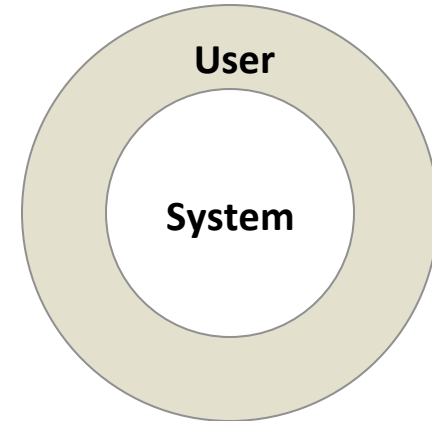


Integer	Memory	Control Flow	Floating Point
Add Sub And Compare ...	Load byte Load Word Store Multiple Push ...	Jump Jump equal Call Return ...	Add single Mult. double Sqrt double ...

Instruction Groupings

System ISA

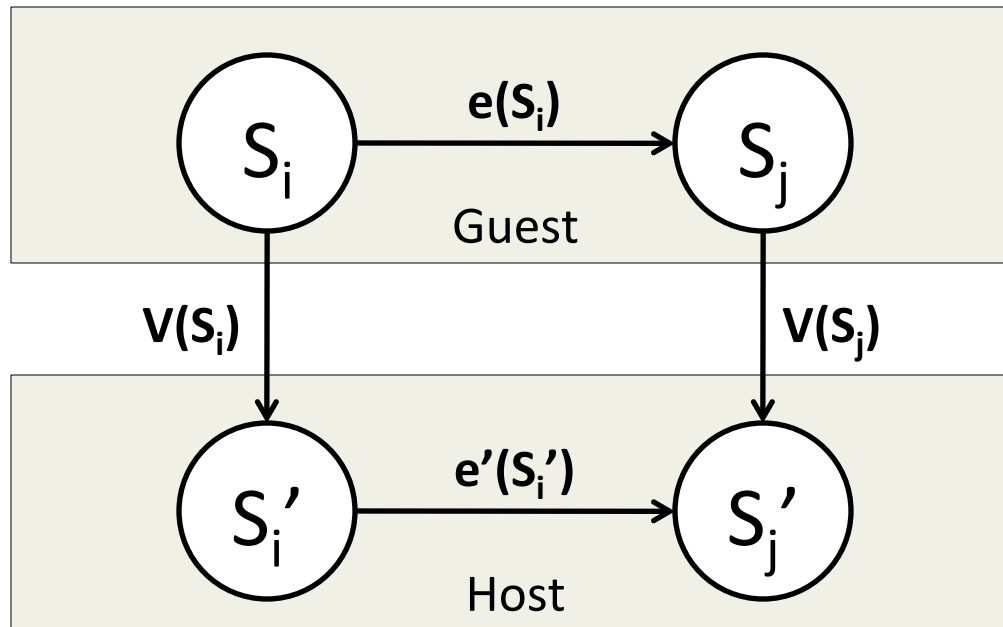
- **Privilege Levels**
- **Control Registers**
- **Traps and Interrupts**
 - Hardcoded Vectors
 - Dispatch Table
- **System Clock**
- **MMU**
 - Page Tables
 - TLB
- **I/O Device Access**



Outline

- **CPU Background**
- **Virtualization Techniques**
 - System ISA Virtualization
 - Instruction Interpretation
 - Trap and Emulate
 - Binary Translation
 - Hybrid Models

Isomorphism



Formally, virtualization involves the construction of an isomorphism from guest state to host state.

Virtualizing the System ISA

- **Hardware needed by monitor**
 - Ex: monitor must control real hardware interrupts
- **Access to hardware would allow VM to compromise isolation boundaries**
 - Ex: access to MMU would allow VM to write any page
- **So...**
 - All access to the virtual System ISA by the guest must be emulated by the monitor in software.
 - System state kept in memory.
 - System instructions are implemented as functions in the monitor.

Example: CPUState

```
static struct {
    uint32  GPR[16];
    uint32  LR;
    uint32  PC;
    int     IE;
    int     IRQ;
} CPUState;

void CPU_CLI(void)
{
    CPUState.IE = 0;
}

void CPU_STI(void)
{
    CPUState.IE = 1;
}
```

- **Goal for CPU virtualization techniques**
 - Process normal instructions as fast as possible
 - Forward privileged instructions to emulation routines

Instruction Interpretation

- **Emulate Fetch/Decode/Execute pipeline in software**
- **Positives**
 - Easy to implement
 - Minimal complexity
- **Negatives**
 - Slow!

Example: Virtualizing the Interrupt Flag w/ Instruction Interpreter

```
void CPU_Run(void)
{
    while (1) {
        inst = Fetch(CPUState.PC);

        CPUState.PC += 4;

        switch (inst) {
        case ADD:
            CPUState.GPR[rd]
                = GPR[rn] + GPR[rm];
            break;
        ...
        case CLI:
            CPU_CLI();
            break;
        case STI:
            CPU_STI();
            break;
        }

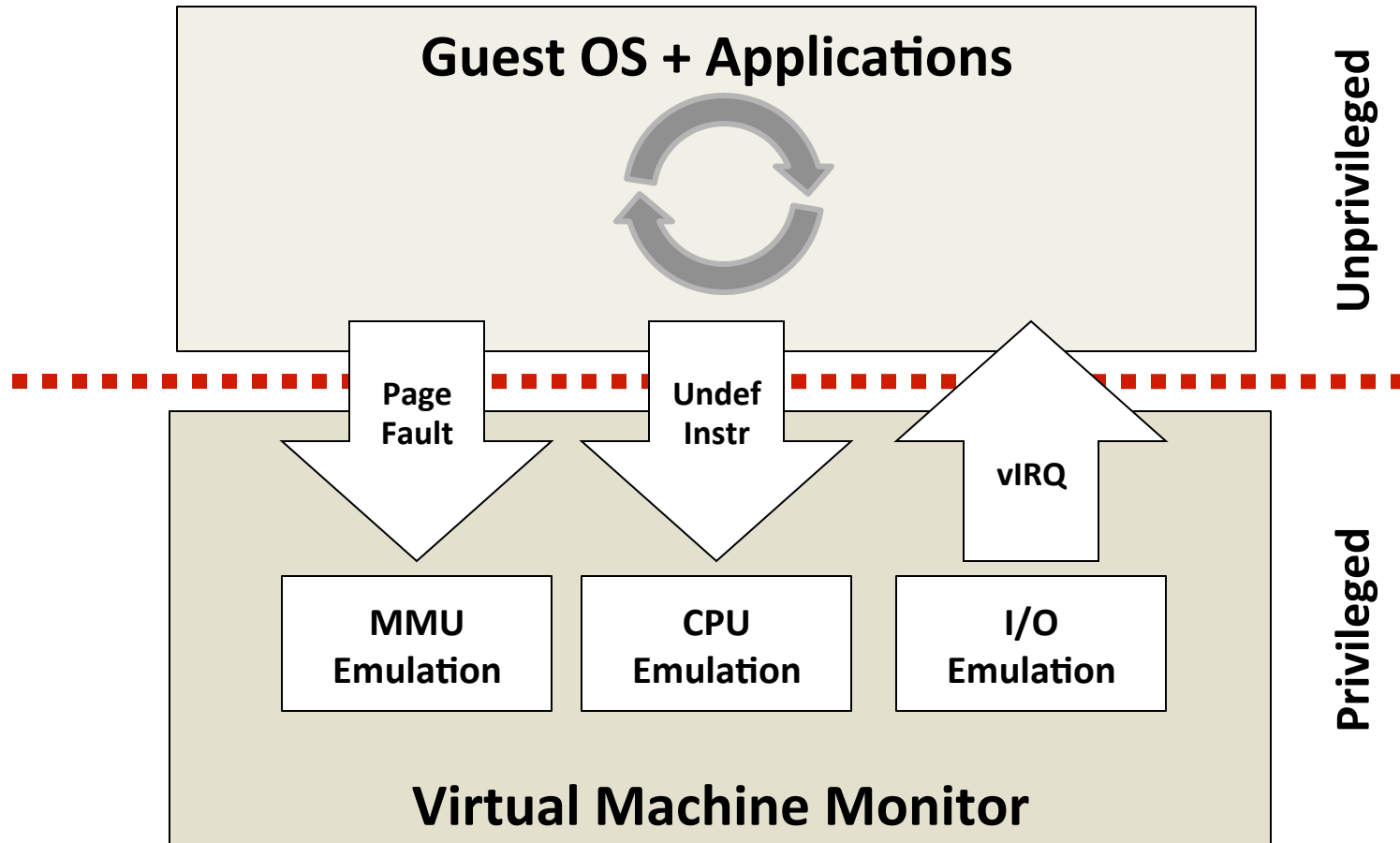
        if (CPUState.IRQ
            && CPUState.IE) {
            CPUState.IE = 0;
            CPU_Vector(EXC_INT);
        }
    }
}

void CPU_CLI(void)
{
    CPUState.IE = 0;
}

void CPU_STI(void)
{
    CPUState.IE = 1;
}

void CPU_Vector(int exc)
{
    CPUState.LR = CPUState.PC;
    CPUState.PC = disTab[exc];
}
```

Trap and Emulate



“Strictly Virtualizable”

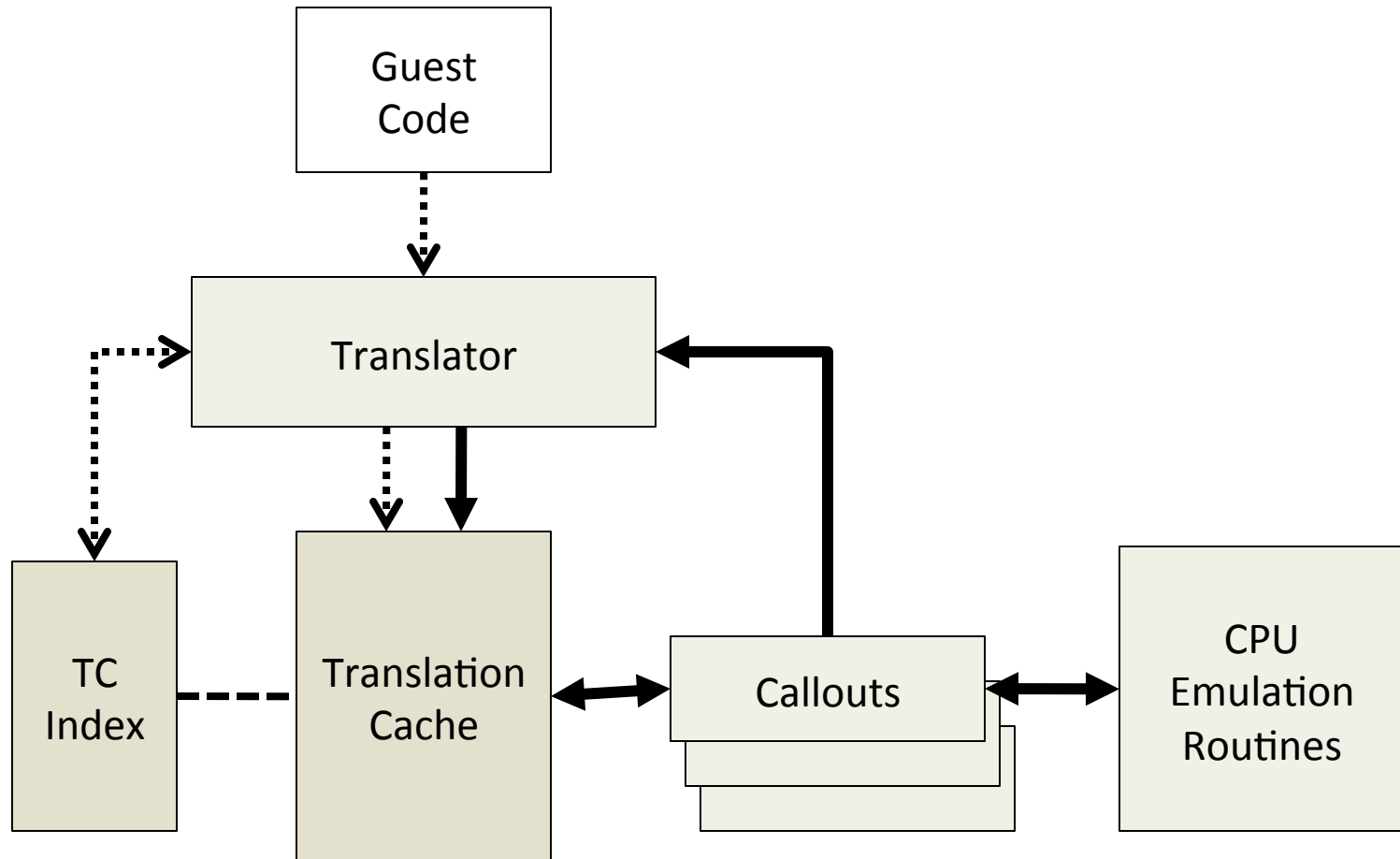
A processor or mode of a processor is strictly virtualizable if, when executed in a lesser privileged mode:

- **all instructions that access privileged state trap**
- **all instructions either trap or execute identically**
- **...**

Issues with Trap and Emulate

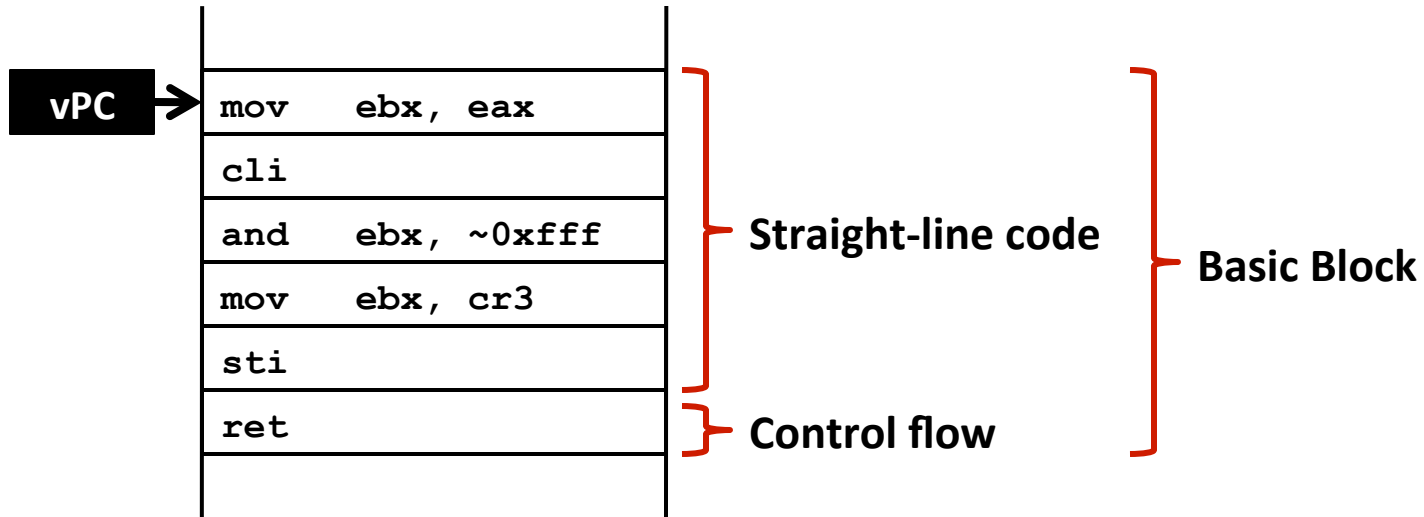
- **Not all architectures support it**
- **Trap costs may be high**
- **Monitor uses a privilege level**
 - Need to virtualize the protection levels

Binary Translator

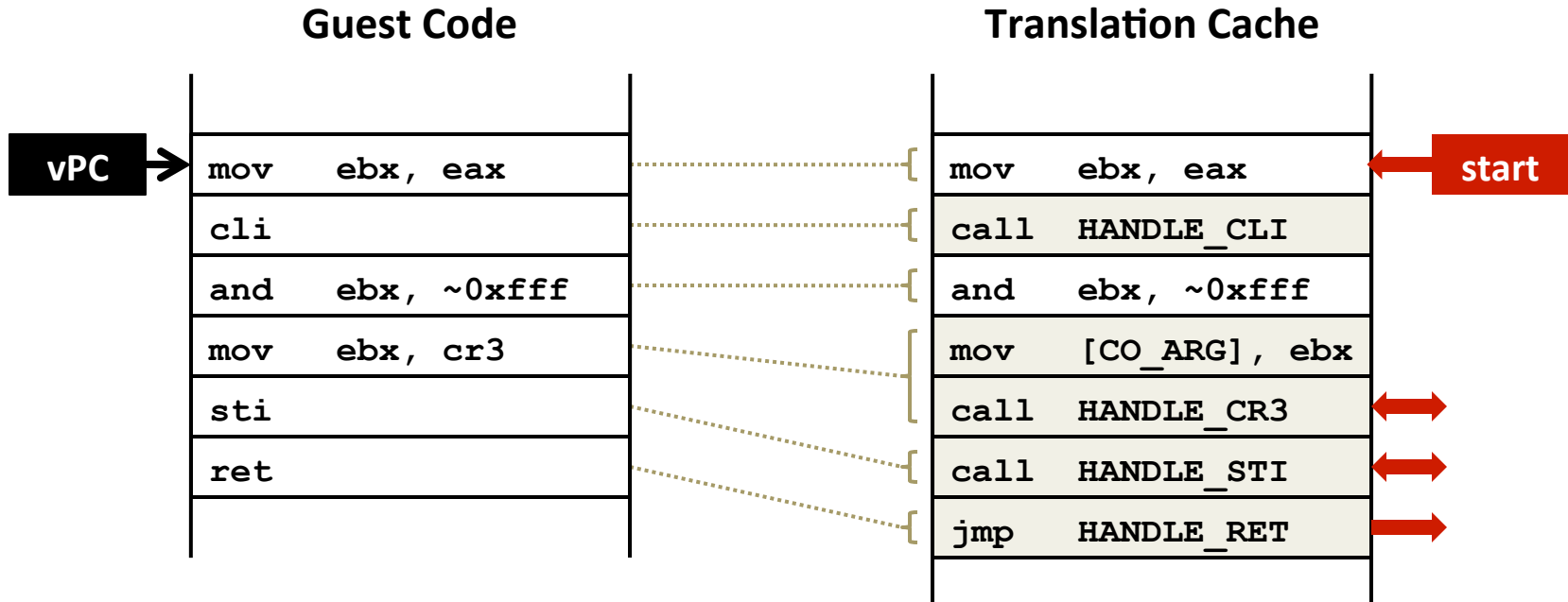


Basic Blocks

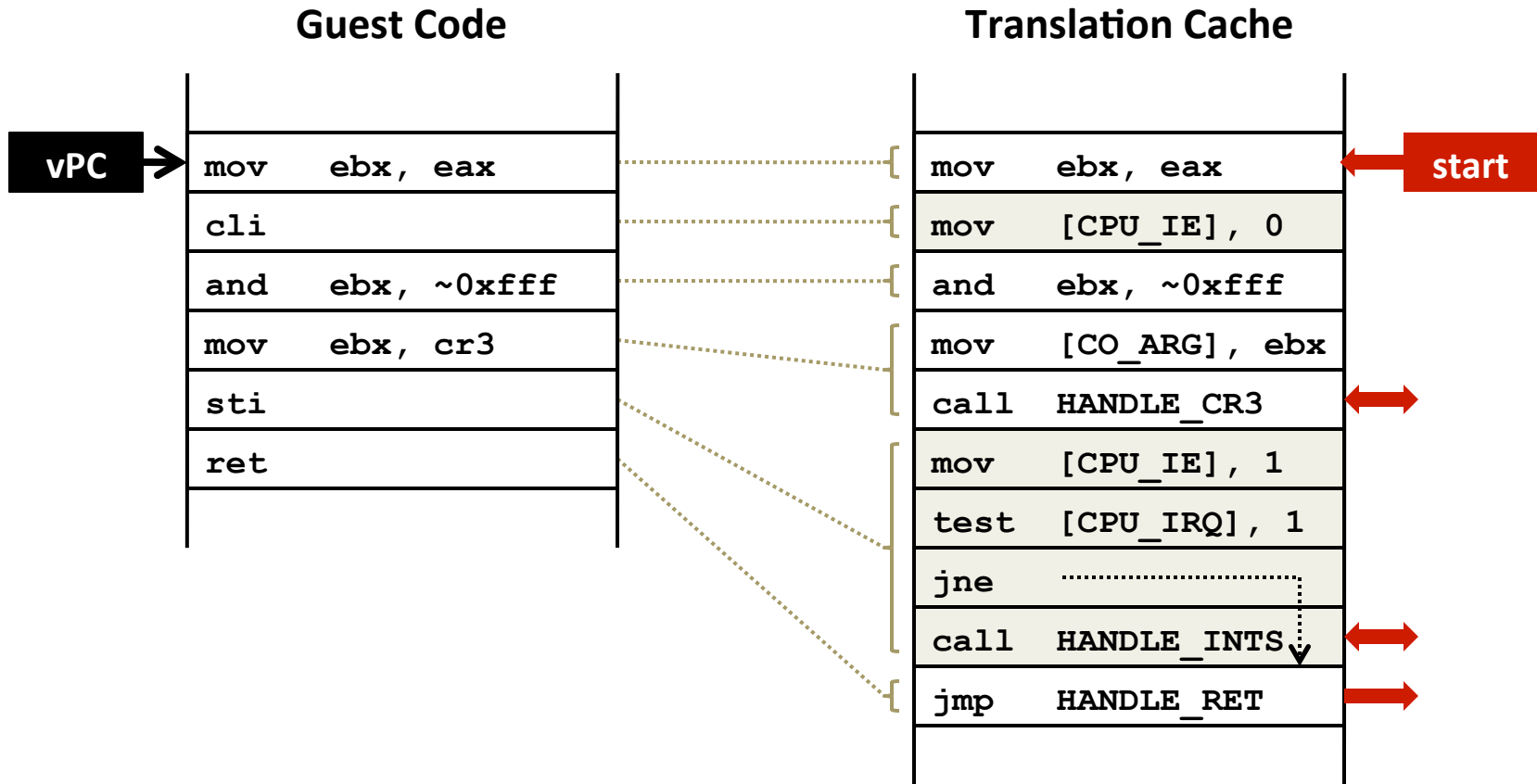
Guest Code



Binary Translation



Binary Translation



Basic Binary Translator

```
void BT_Run(void)
{
    CPUState.PC = _start;
    BT_Continue();
}

void BT_Continue(void)
{
    void *tcpc;

    tcpc = BTFindBB(CPUState.PC);

    if (!tcpc) {
        tcpc = BTTranslate(CPUState.PC);
    }

    RestoreRegsAndJump(tcpc);
}
```

```
void *BTTranslate(uint32 pc)
{
    void *start = TCTop;
    uint32 TCPC = pc;

    while (1) {
        inst = Fetch(TCPC);
        TCPC += 4;

        if (IsPrivileged(inst)) {
            EmitCallout();
        } else if (IsControlFlow(inst)) {
            EmitEndBB();
            break;
        } else {
            /* ident translation */
            EmitInst(inst);
        }
    }

    return start;
}
```

Basic Binary Translator – Part 2

```
void BT_CalloutSTI(BTSavedRegs regs)
{
    CPUState.PC = BTFindPC(regs.tcpc);
    CPUState.GPR[] = regs.GPR[];

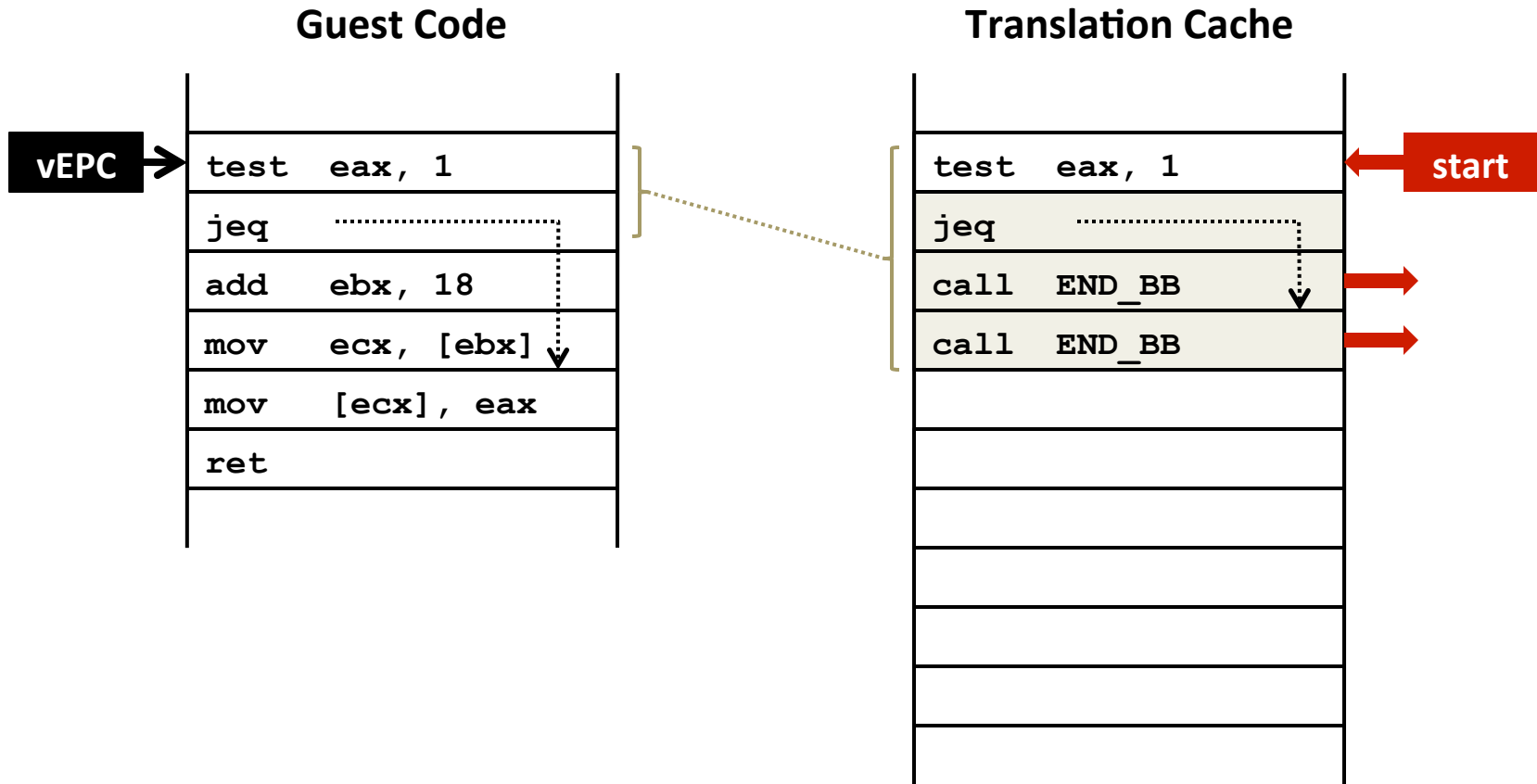
    CPU_STI();

    CPUState.PC += 4;

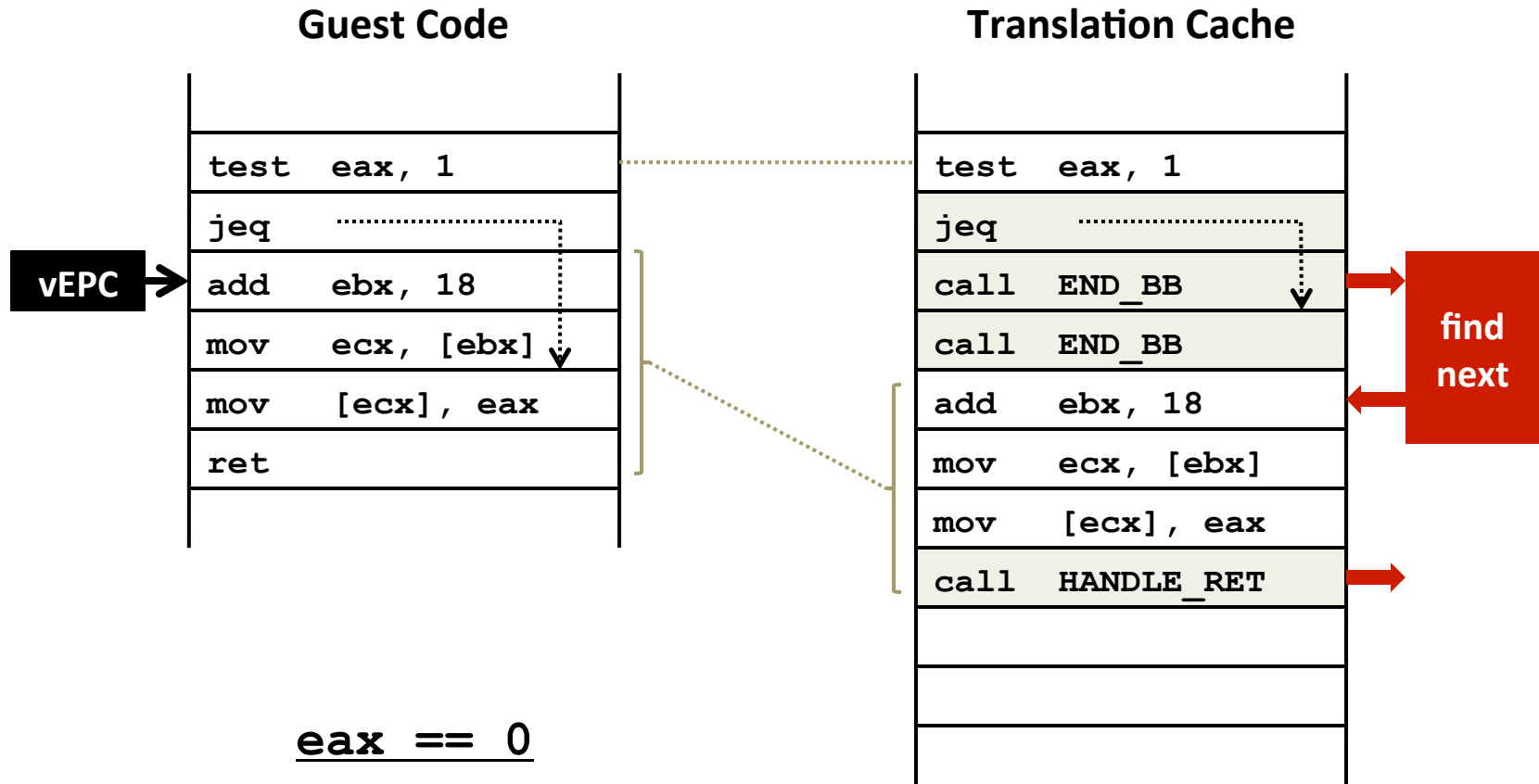
    if (CPUState.IRQ
        && CPUState.IE) {
        CPUVector();
        BT_Continue();
        /* NOT_REACHED */
    }

    return;
}
```

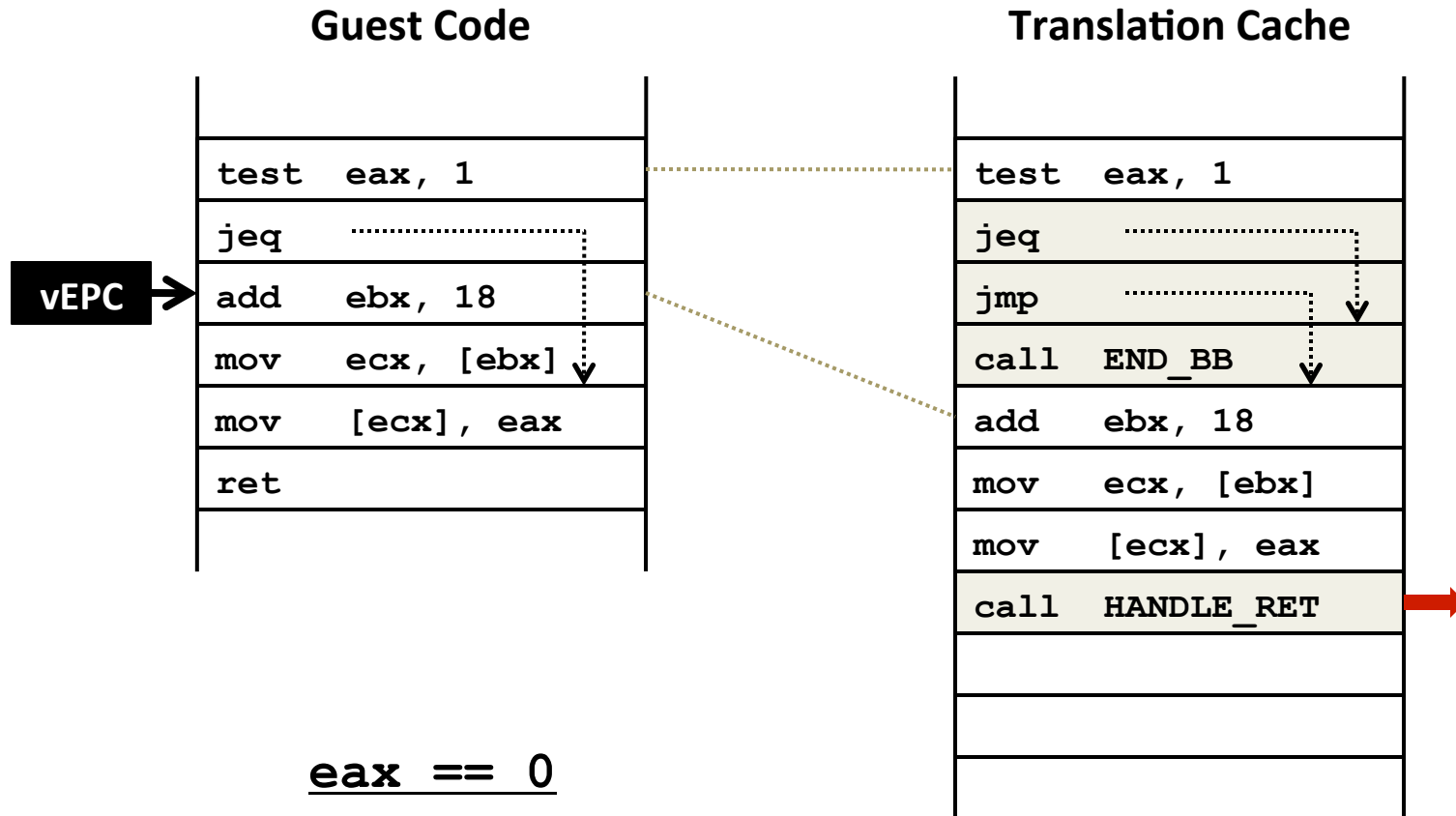
Controlling Control Flow



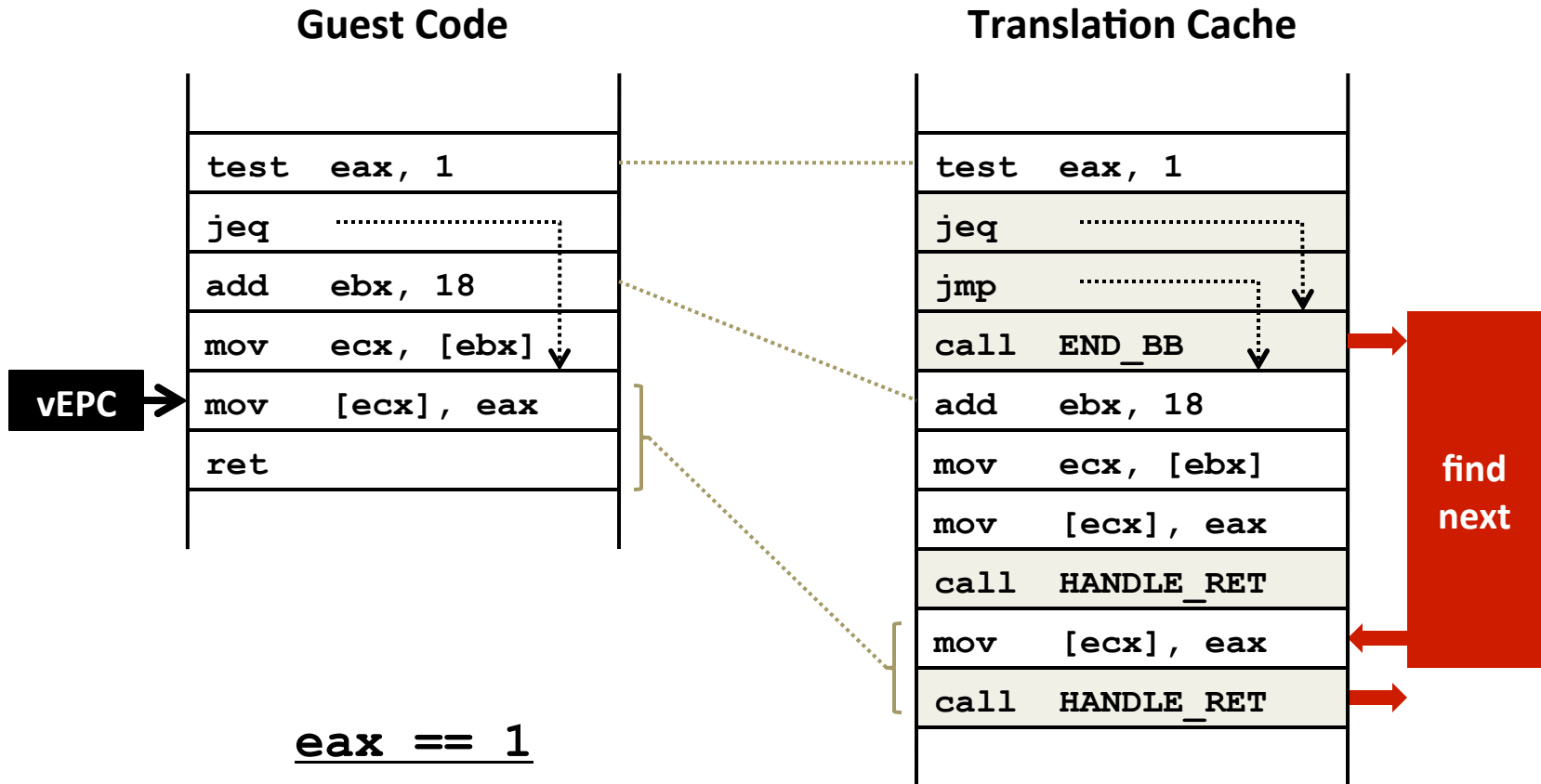
Controlling Control Flow



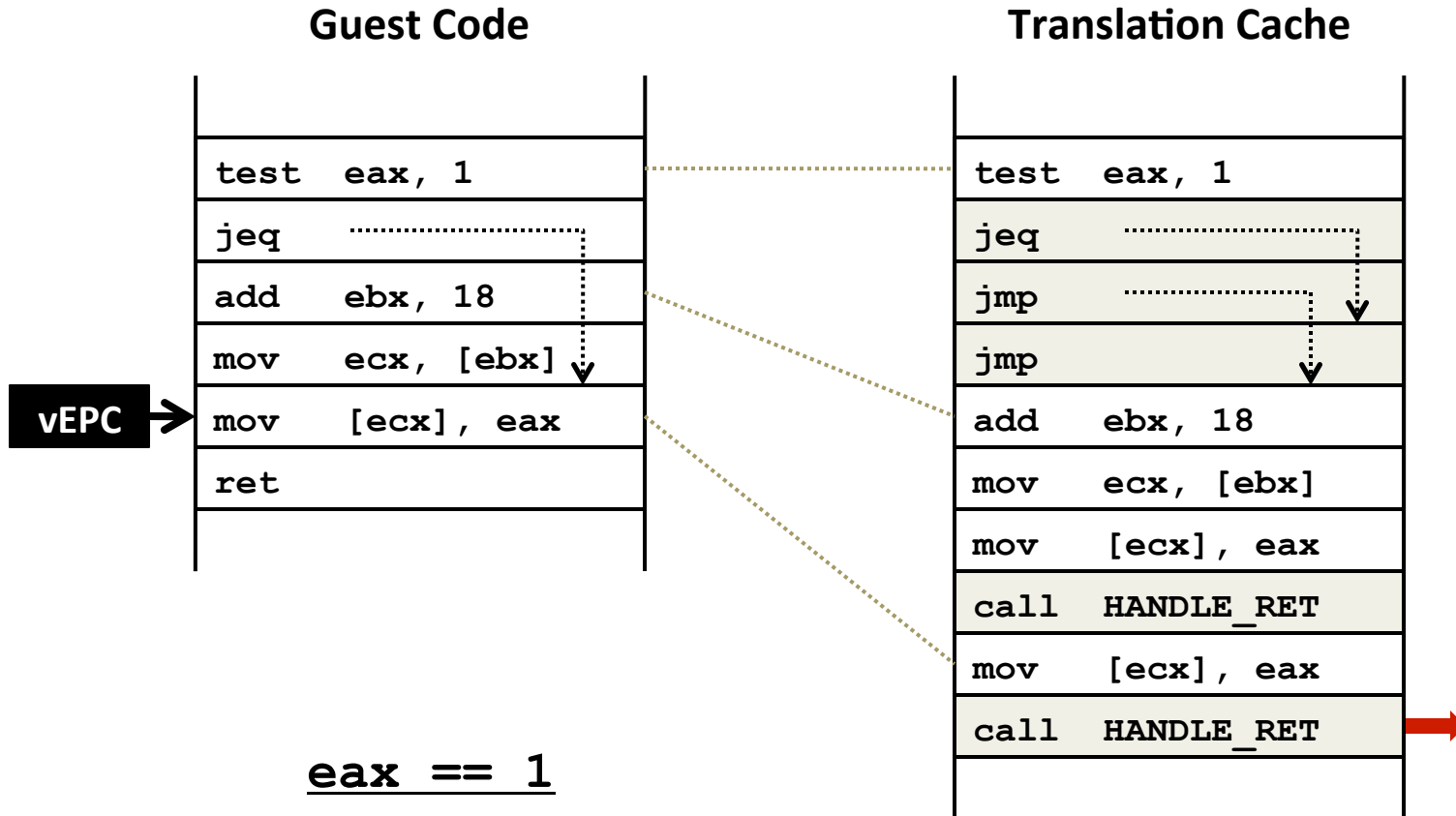
Controlling Control Flow



Controlling Control Flow



Controlling Control Flow



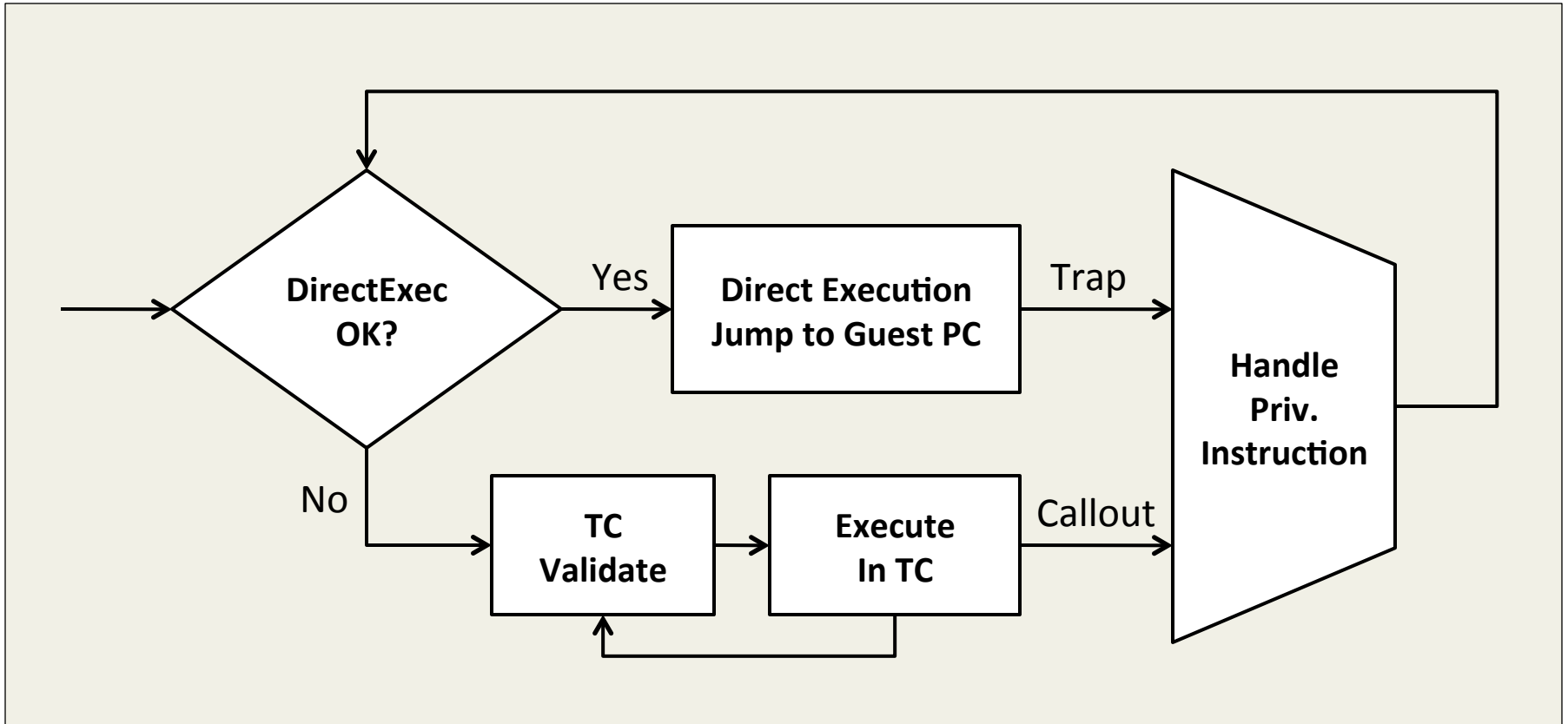
Issues with Binary Translation

- **Translation cache index data structure**
- **PC Synchronization on interrupts**
- **Self-modifying code**
 - Notified on writes to translated guest code

Other Uses for Binary Translation

- **Cross ISA translators**
 - Digital FX!32
- **Optimizing translators**
 - H.P. Dynamo
- **High level language byte code translators**
 - Java
 - .NET/CLI

Hybrid Approach



- **Binary Translation for the Kernel**
- **Direct Execution (Trap-and-emulate) for the User**
- **U.S. Patent 6,397,242**

Homework 1

Binary Patching

- **Binary Patching for profiling and code coverage**
 - Process Virtualization
 - Patching code be compiled into program
- **Follow execution by patching control flow instructions**
- **Patch instructions in-place**
 - No need for translation or copying
- **Instruction decoding need only determine**
 - Length of instruction
 - Control flow points
- **Callouts call through asm linkage**
 - Saves EFLAGS
 - Saves registers
 - Calls C code