Virtual Machines

Sorav Bansal

Virtualization

Providing a hardware-like view to each process

or

Running an OS inside another OS

or

Running multiple OSes on single physical hardware

Emulating a physical machine in software

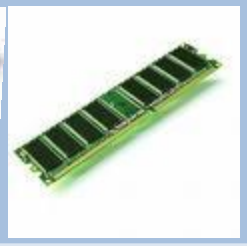
Traditional Picture

Application 1

Application 2

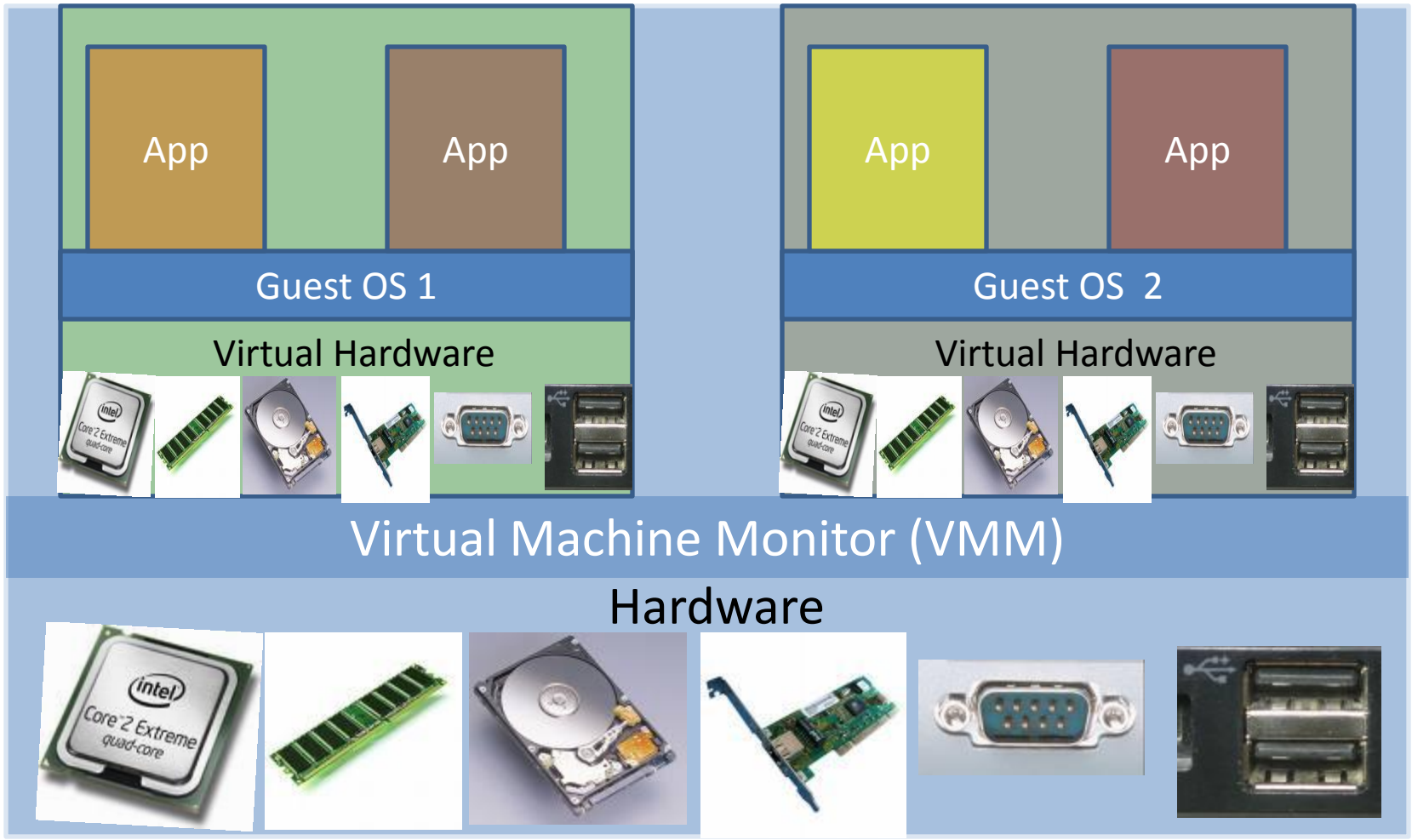
OS

Hardware





Virtualized Picture



This Lecture

- Why?
  - Applications of Virtualization
- How?
  - Binary translation
  - Memory virtualization
  - Device emulation (Disk, NIC, …)
- Looking ahead…

Advantages of Virtualization

- Server consolidation
- Best of all worlds
  - e.g., run Windows and Linux simultaneously
- Complete isolation between applications
  - e.g., Internet VM and development VM (desktop)
  - e.g., Mail server VM and print server VM (server)
- Encapsulation (a VM is just a file)
  - e.g., snapshotting
- New Applications: Security, Reproducibility, Monitoring, Migration, Legacy systems, …

How it works?

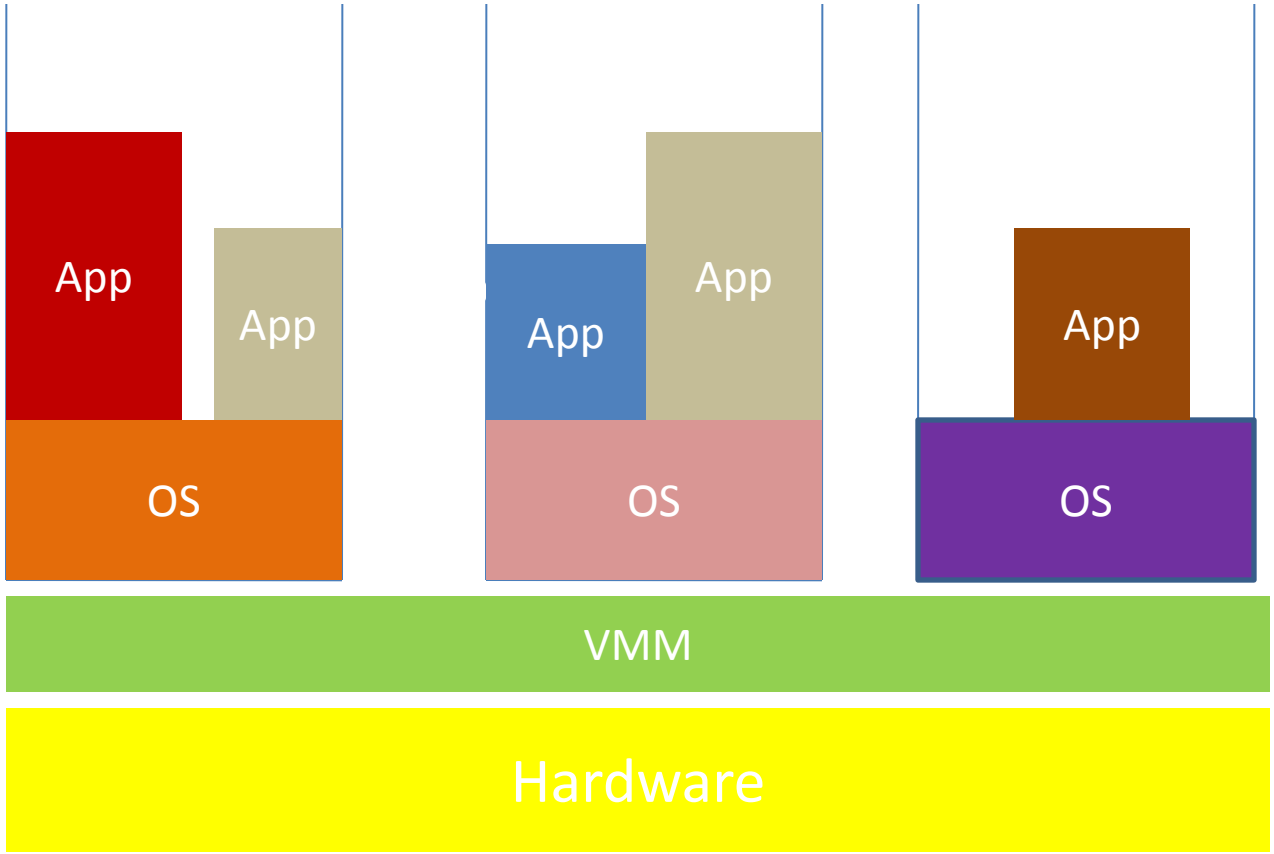
1. Interpretation (e.g., bochs)
   - Interpret each instruction and emulate it
     - e.g., Each instruction is implemented by a C function
       - incl (%eax):
         - » r = regs[EAX];
         - » tmp = read_mem(r);
         - » tmp++;
         - » write_mem(r, tmp);
   - Slowdown? 50x

2. Binary Translation (e.g., qemu)
   - Translate each guest instruction to the minimal set of host instructions required to emulate it
     - e.g.:
       - incl (%eax)
         - » leal mem0(%eax), %esi
         - » incl (%esi)
   - Advantages
     - Avoid function-call overhead of interpreter-based approach
     - Can re-use translations by maintaining a translation cache
   - Slowdown? 5-10x

How it works? (..contd)

- VMM: Direct execution whenever possible, binary translate otherwise
  - reg-reg instructions. e.g., movl %eax, %ecx
    - always possible
  - reg-mem instructions. e.g., movl (%eax), %ecx
    - Need “Memory Virtualization”! (next slide)
  - I/O instructions. e.g., in %eax
    - No! Need binary translation
    - In most cases, the instruction is trying to access a device. Need to emulated the device in software
    - DMA requests handled similarly
  - Traps?
    - Trap in the VMM, take control of the situation and trap to guest OS if needed
  - Interrupts?
    - Deliver interrupts to guest OS at safe instruction boundaries
  - Slowdown? 0-50%, typically 20%... good!

Virtual Machine Monitor (VMM)



Old Idea (1960s) : IBM Mainframes

Was a good idea for expensive hardware at that time

Virtual Machine Monitors

- [Popek, Goldberg 1974]
  - An architecture is virtualizable if the set of instructions that could affect the correct functioning of the VMM are a subset of the privileged instructions
    - i.e., all sensitive instructions must always pass control to the VMM
- x86 was not designed to be virtualizable
  - VMware Solution
    - Binary translate sensitive instructions to force them to trap into VMM
    - Most instructions execute identically
- Intel VT and AMD-V (2008)
  - Support for virtualization in hardware for x86
  - Obey the principles required to make hardware virtualizable
  - Hence, on modern machines, we no longer require binary translation

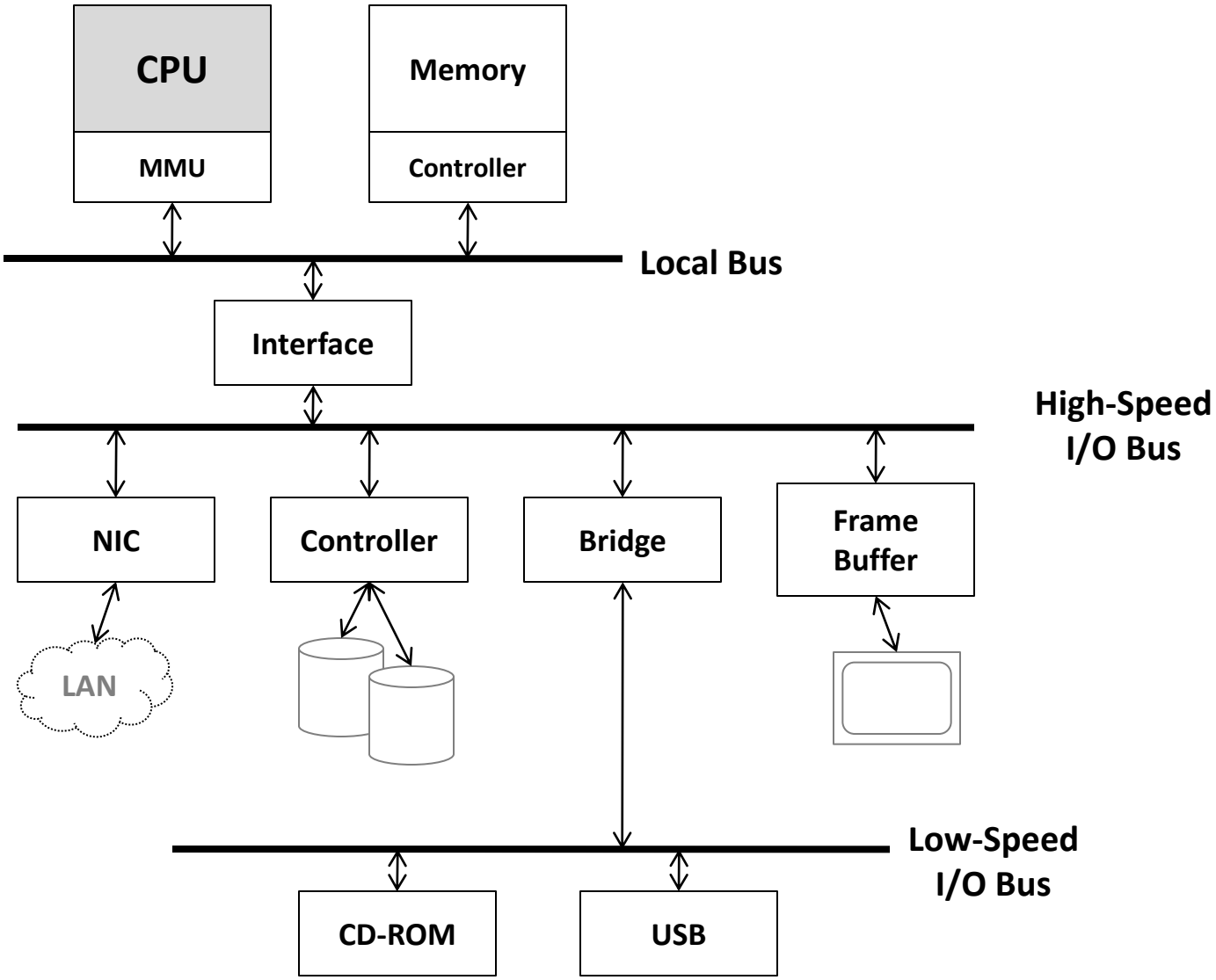
Virtual Machine Monitor

- Hardware Support (IBM Mainframes 1960s, Intel VT/AMD-V 2006)
  - Simple and fast to develop
  - Expected to be faster
- Binary Translation (VMware 1998)
  - More flexible
  - Often faster
- ParaVirtualization (Xen 2003)
  - Much more efficient
  - But… can only run a particular kernel (modified version of Linux) on it

Outline

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

Computer System Organization



Slide Author: Scott Devine

CPU Organization

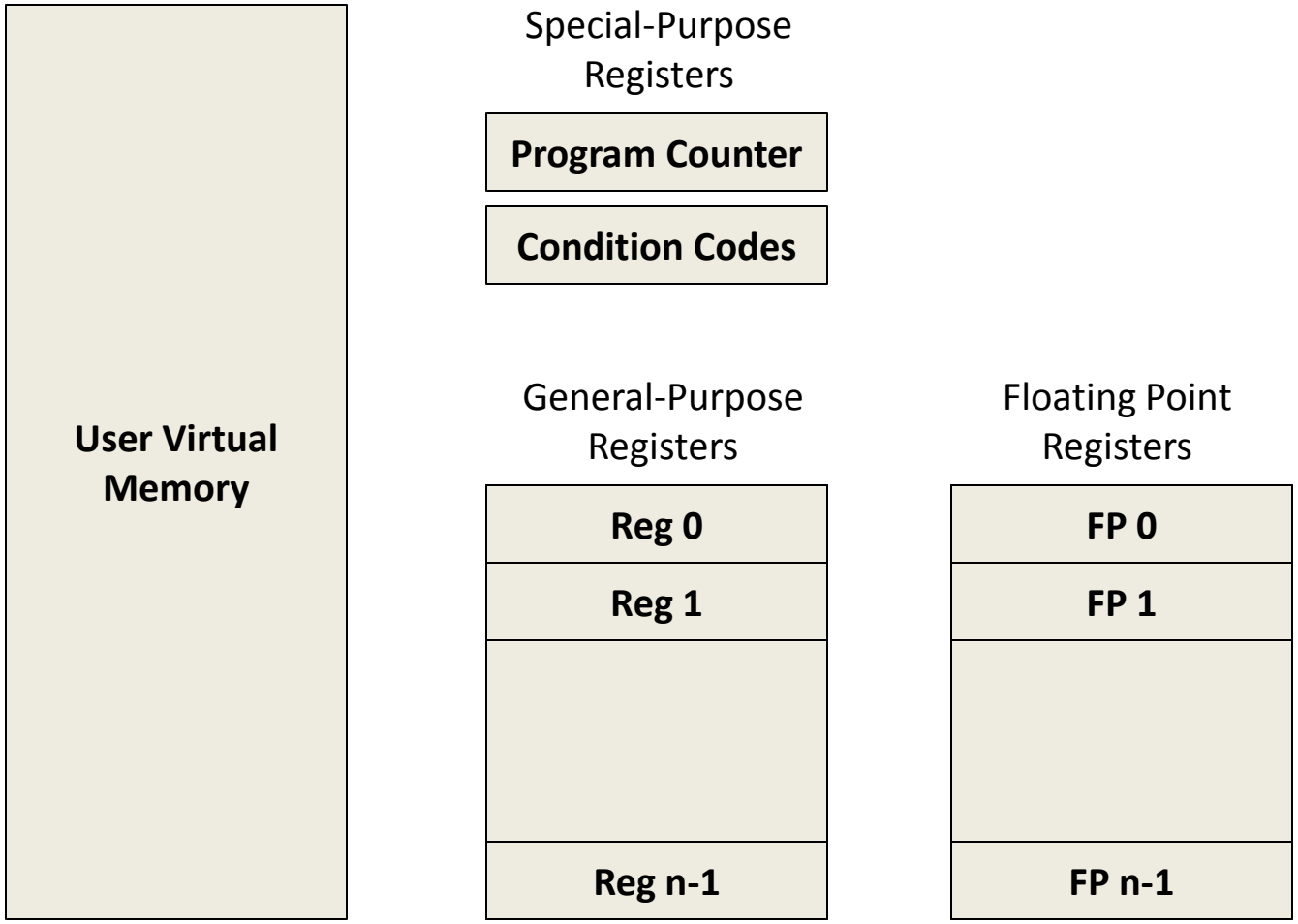
- Instruction Set Architecture (ISA)

  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

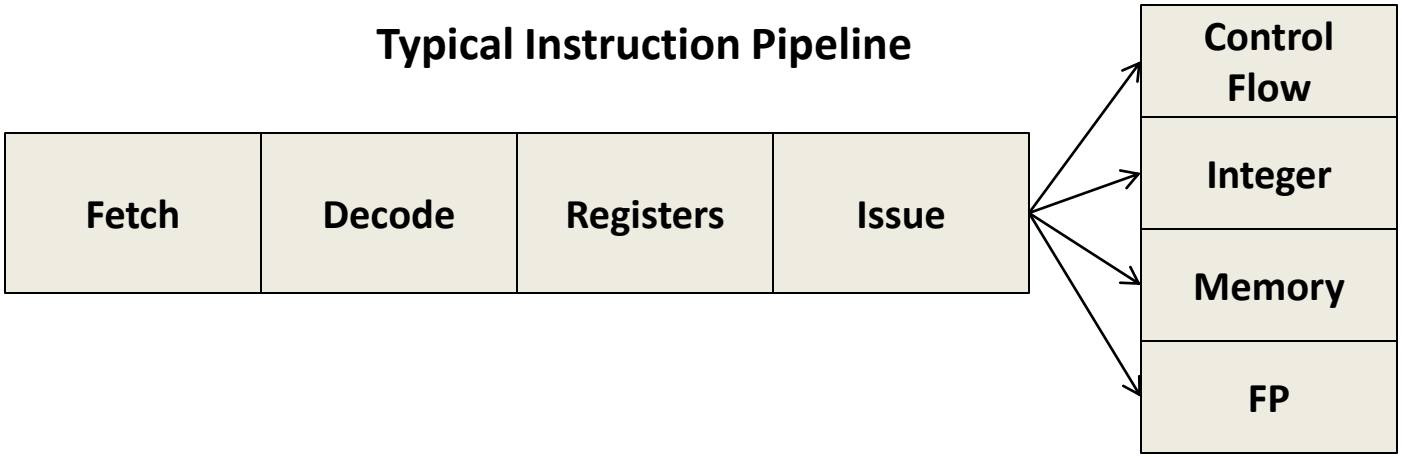
Slide Author: Scott Devine

User ISA - State



Slide Author: Scott Devine

User ISA – Instructions



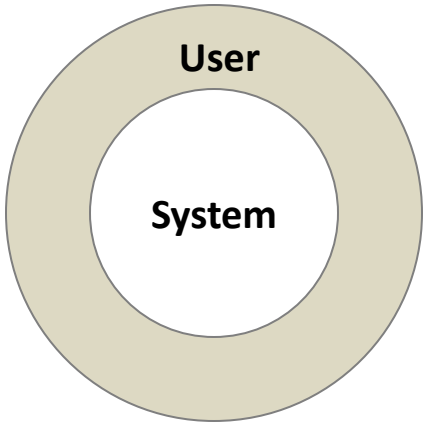
Integer	Memory	Control Flow	Floating Point
Add Sub And Compare …	Load byte Load Word Store Multiple Push …	Jump Jump equal Call Return …	Add single Mult. double Sqrt double …

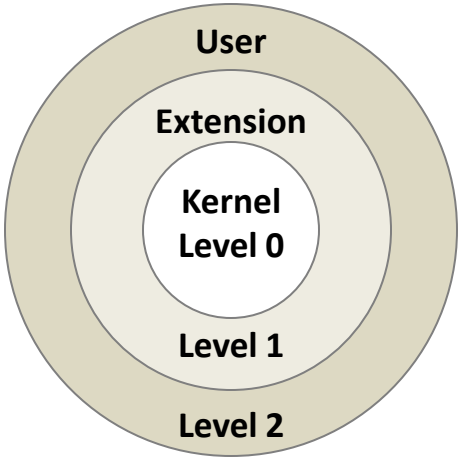
Instruction Groupings

Slide Author: Scott Devine

System ISA

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





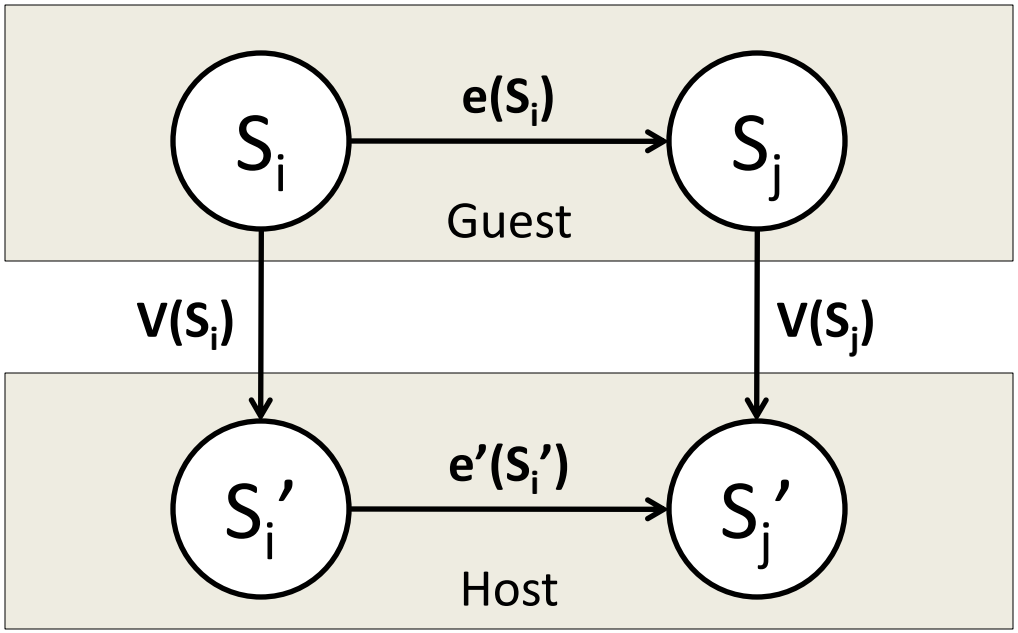
Slide Author: Scott Devine

Outline

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

Slide Author: Scott Devine

Isomorphism



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

Slide Author: Scott Devine

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.

Slide Author: Scott Devine

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

Slide Author: Scott Devine

Instruction Interpretation

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

Slide Author: Scott Devine

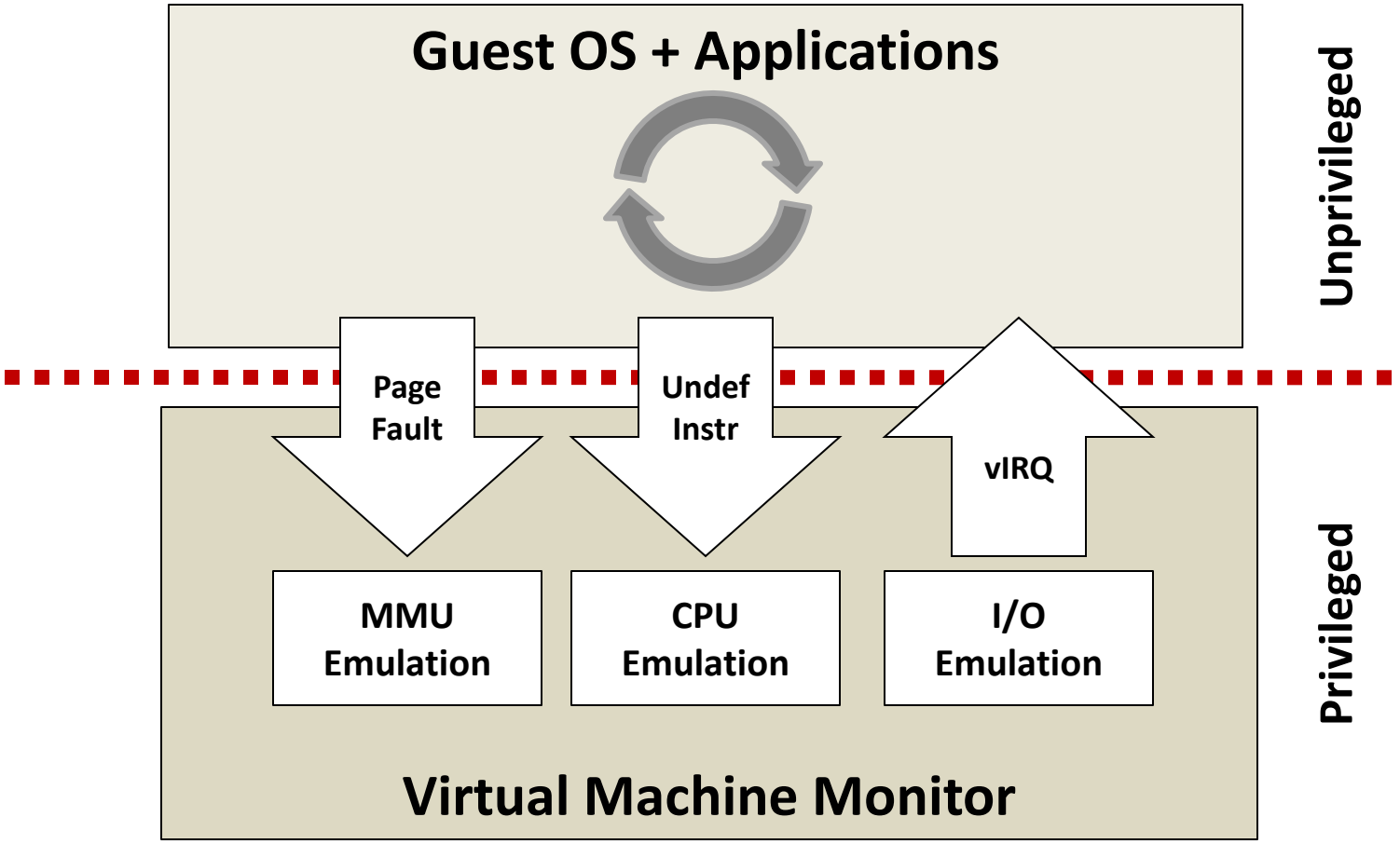
Example: Virtualizing the Interrupt Flag w/ Instruction Interpreter

```
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];
}
```

Slide Author: Scott Devine

Trap and Emulate



Slide Author: Scott Devine

“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
- …

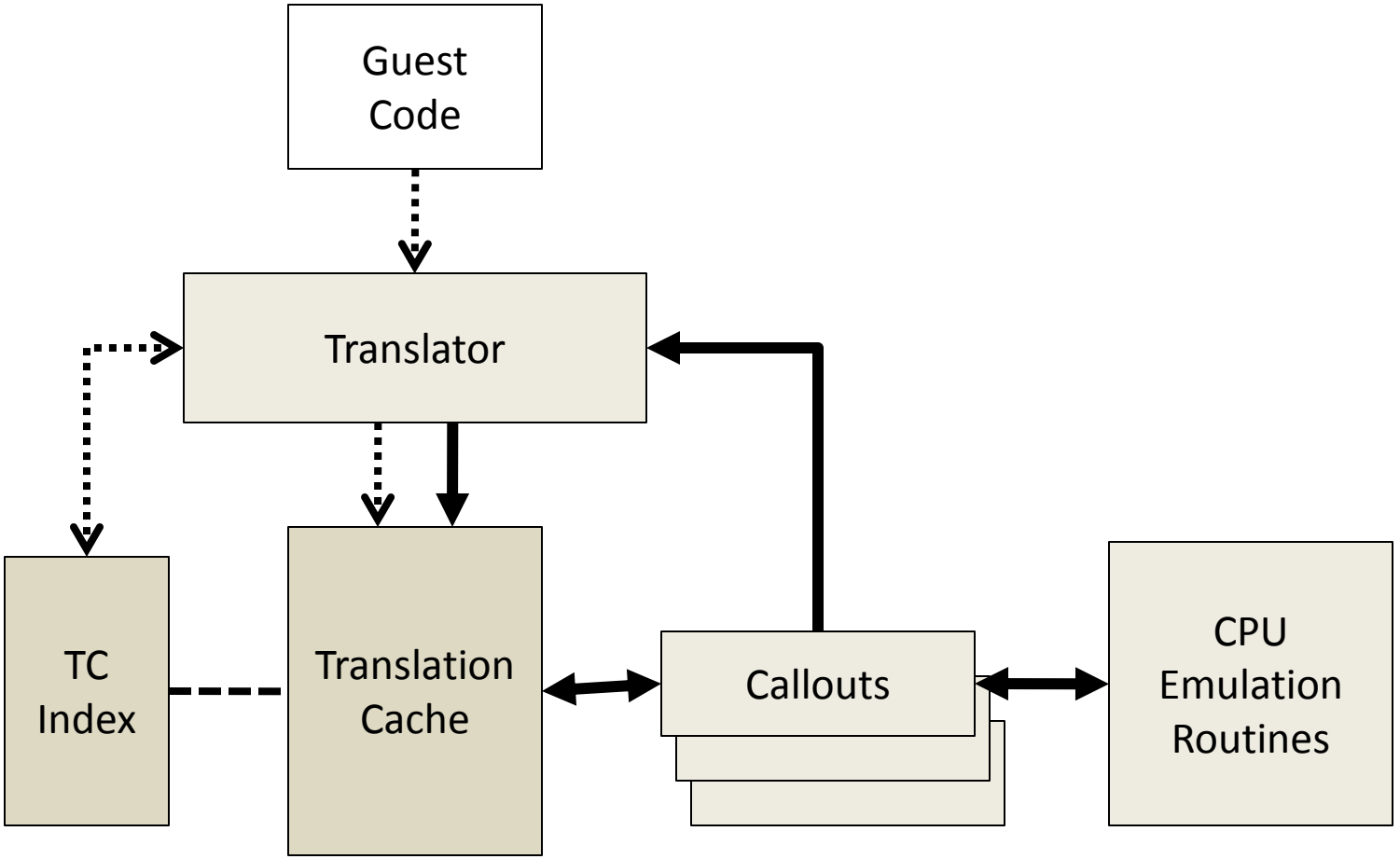
Slide Author: Scott Devine

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

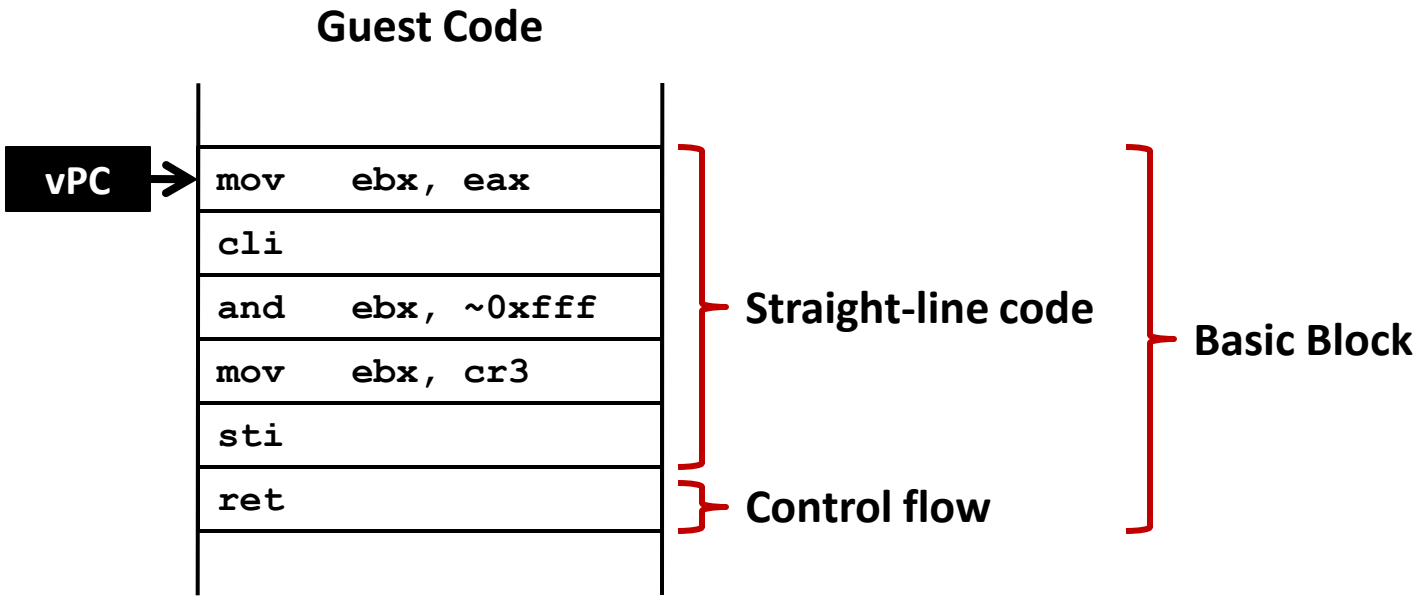
Slide Author: Scott Devine

Binary Translator



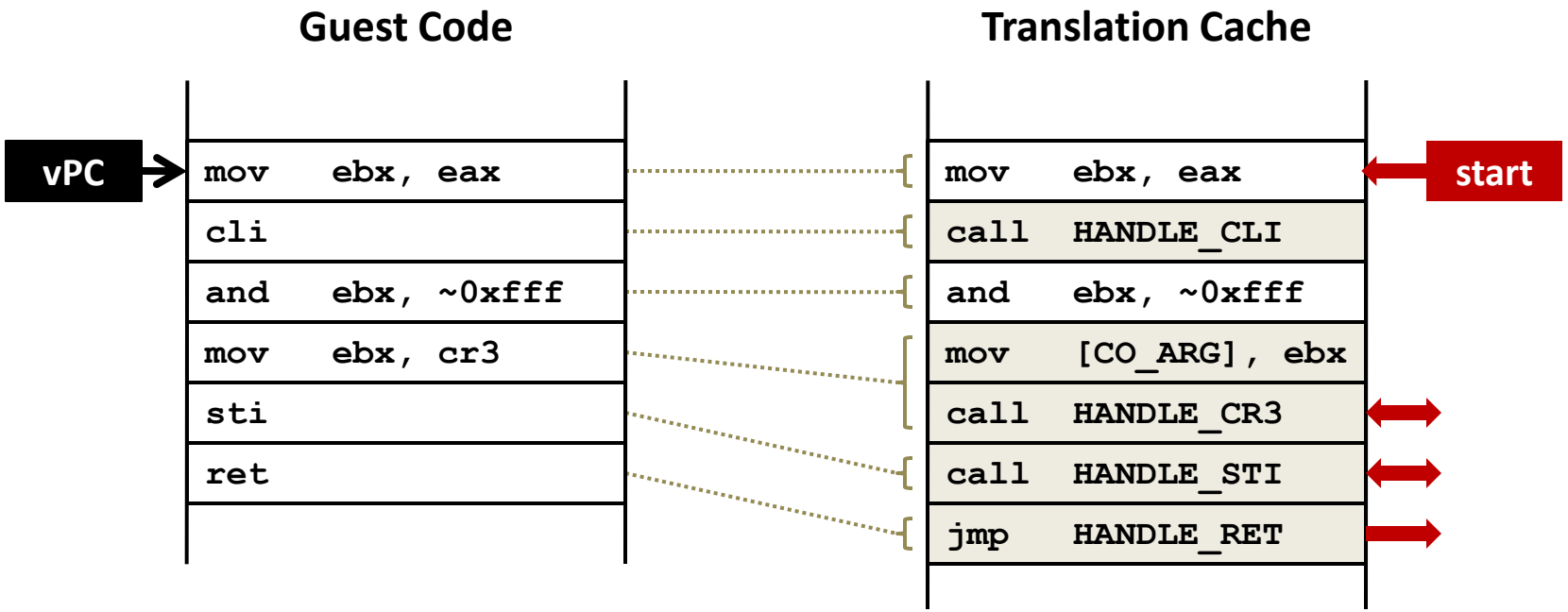
Slide Author: Scott Devine

Basic Blocks



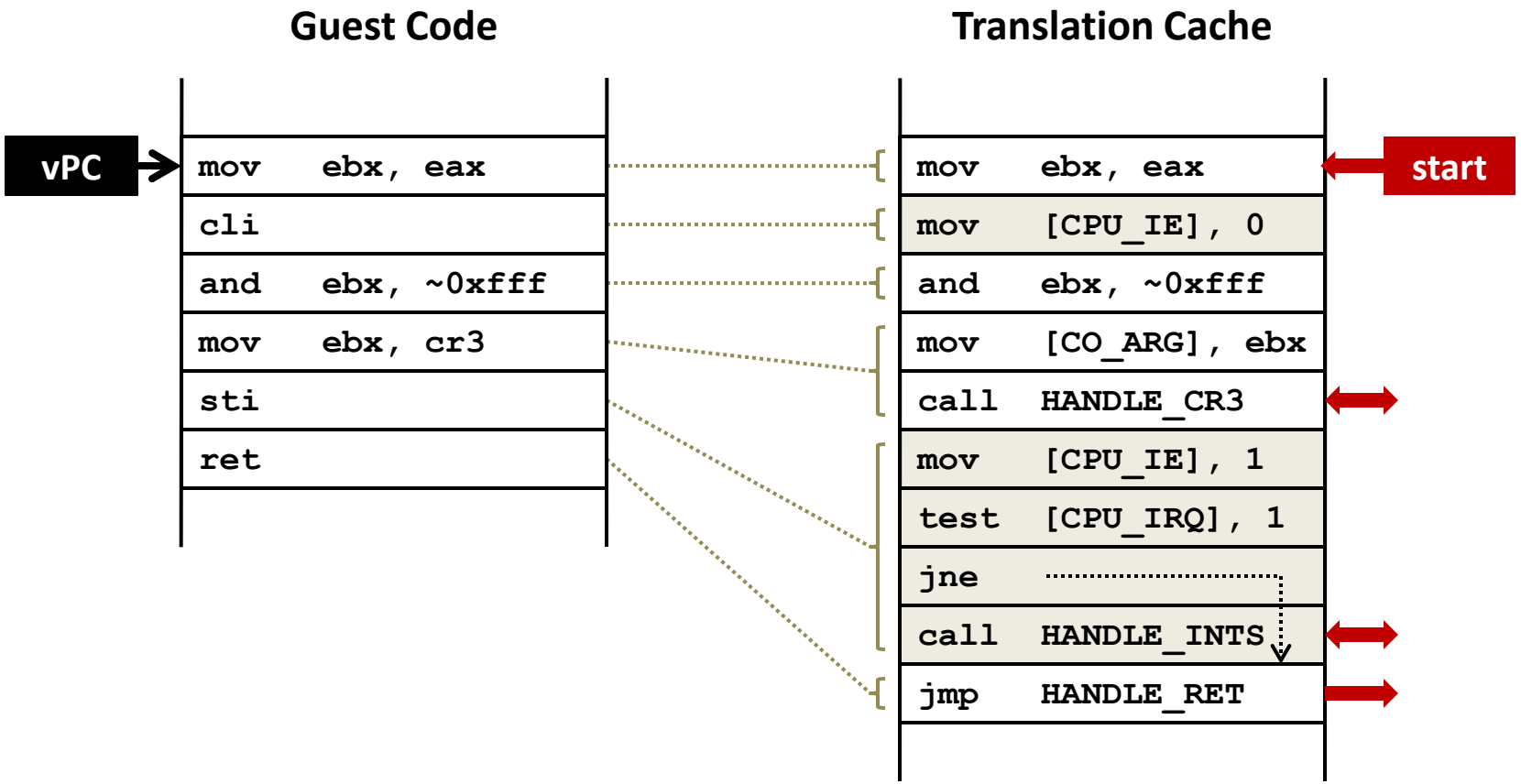
Slide Author: Scott Devine

Binary Translation



Slide Author: Scott Devine

Binary Translation



Slide Author: Scott Devine

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;
}
```

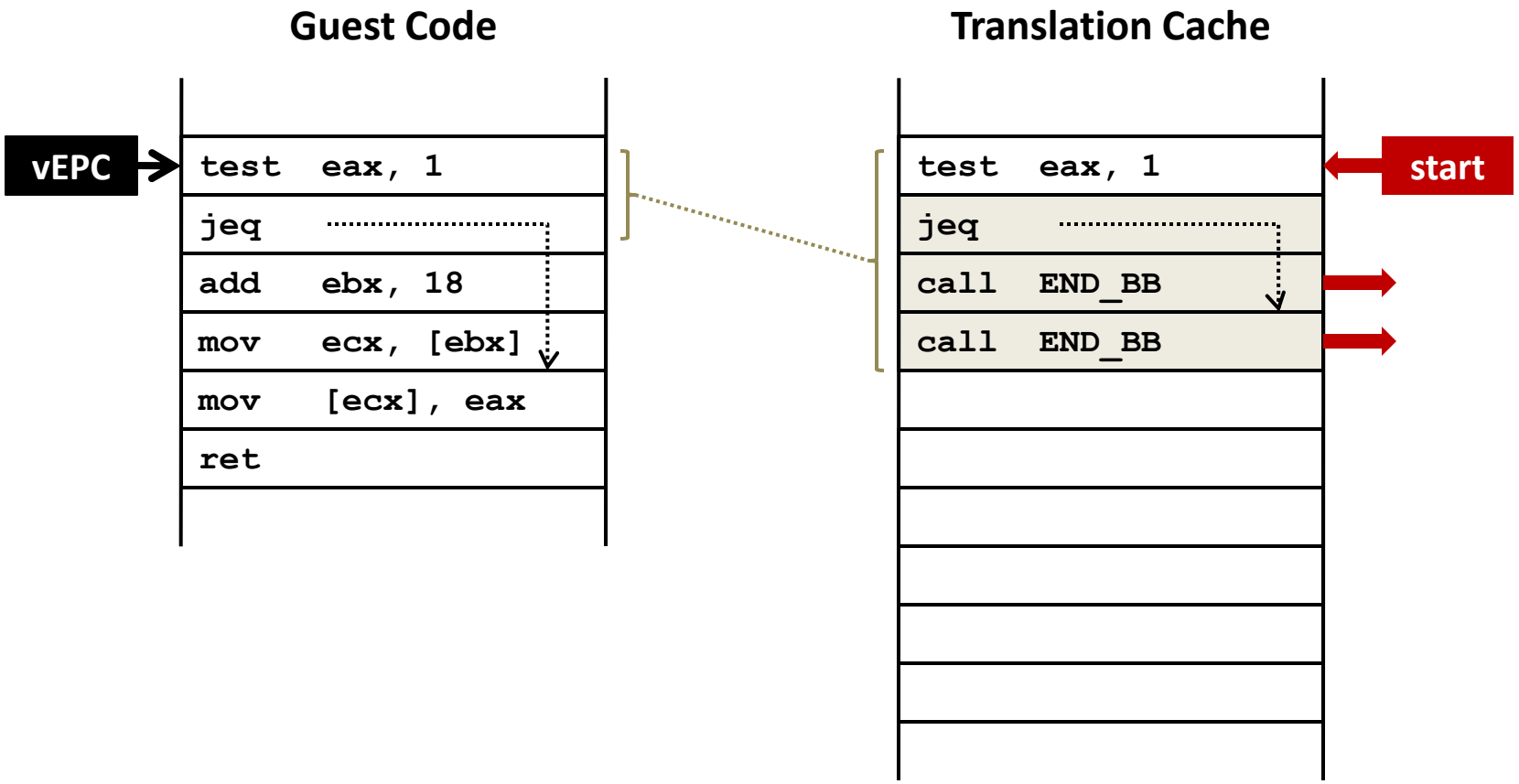
Slide Author: Scott Devine

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;
}
```

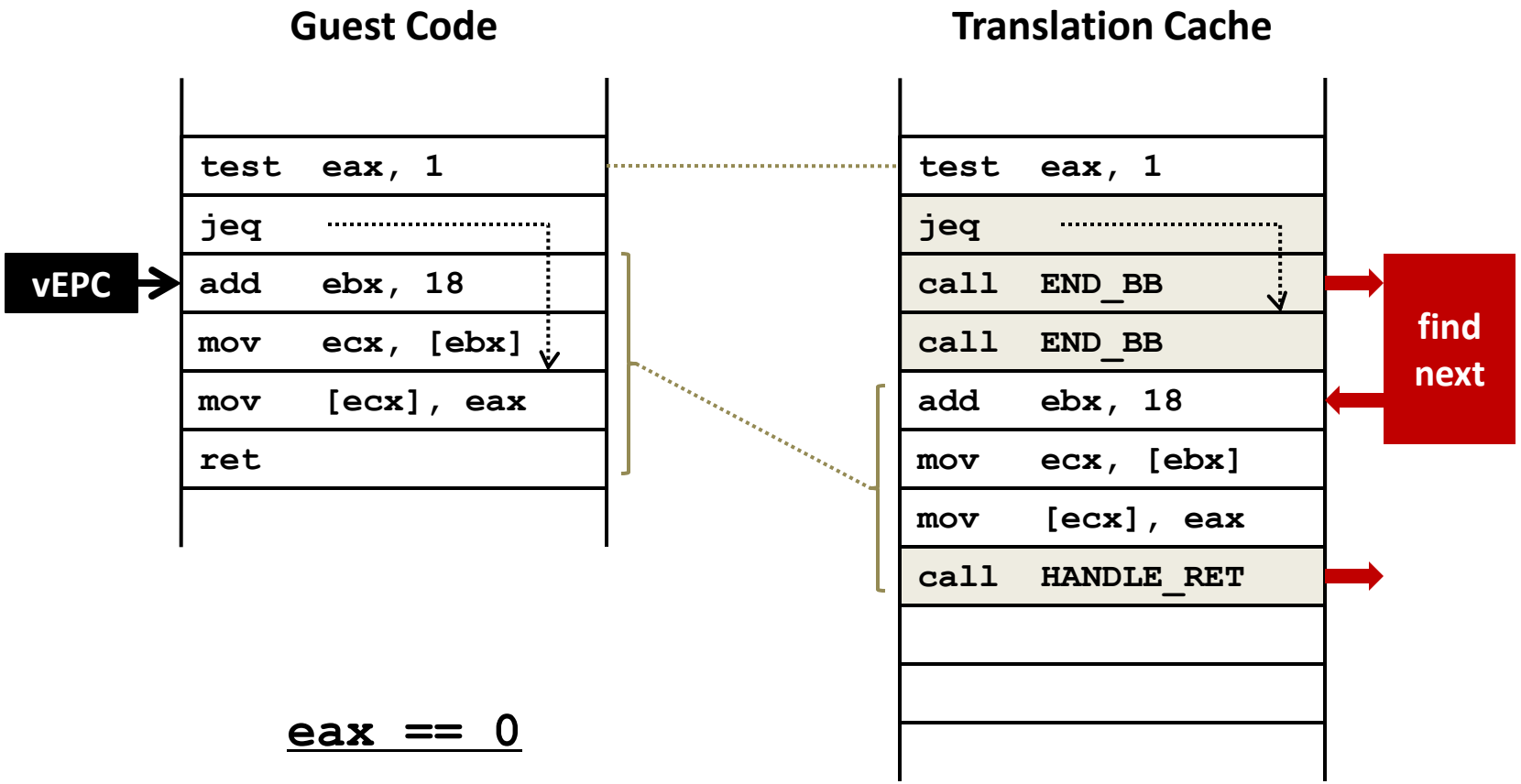
Slide Author: Scott Devine

Controlling Control Flow



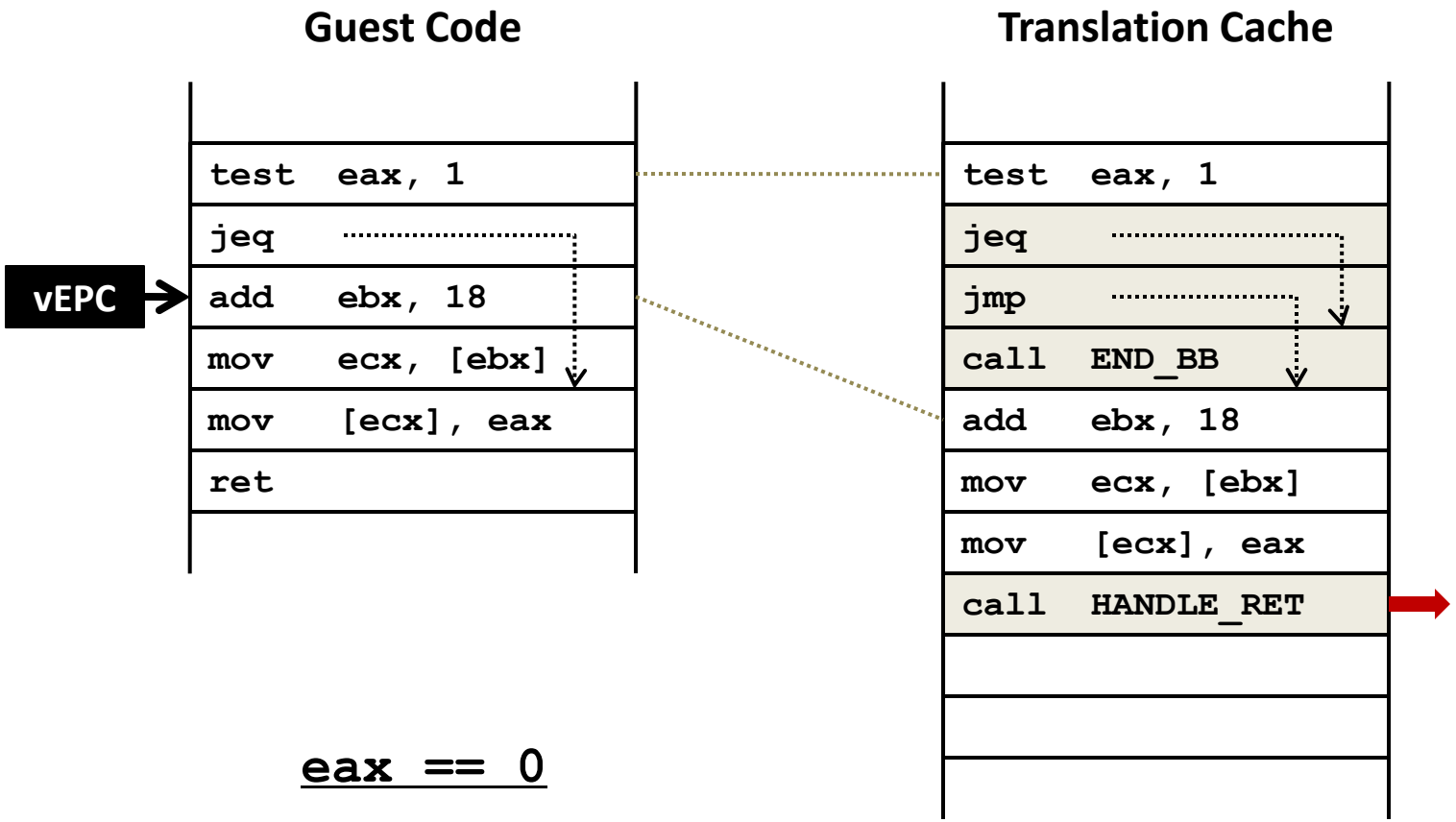
Slide Author: Scott Devine

Controlling Control Flow



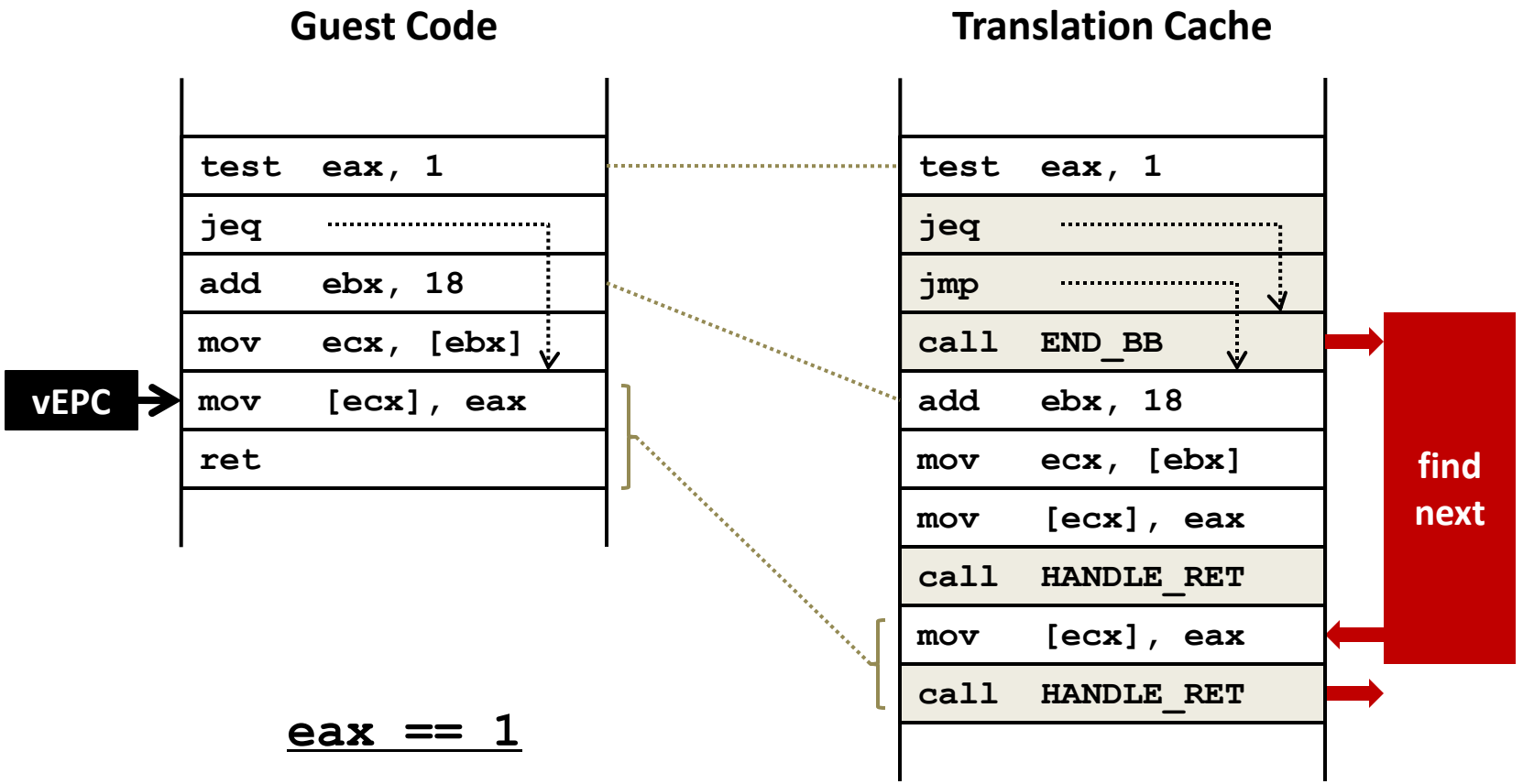
Slide Author: Scott Devine

Controlling Control Flow



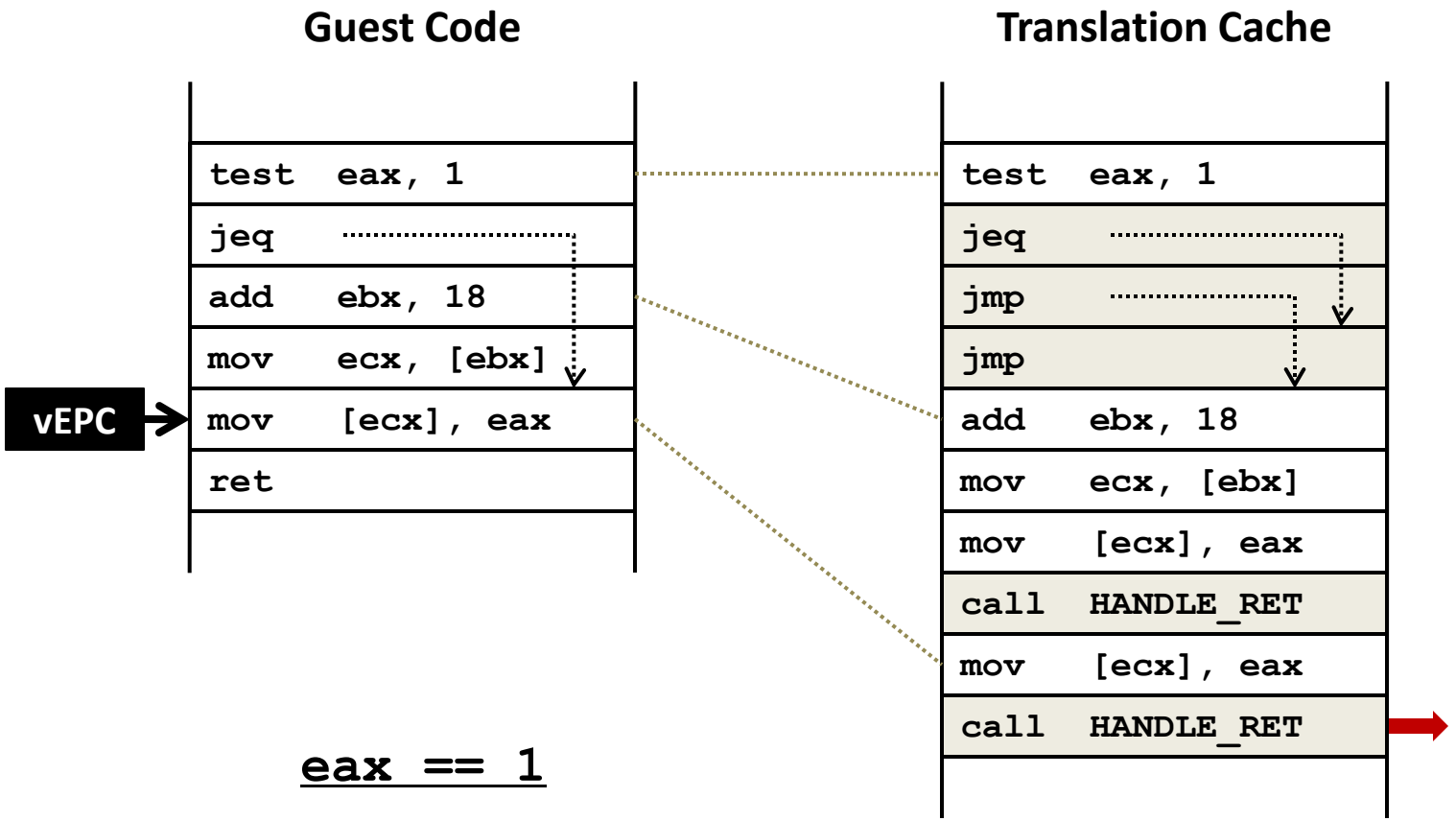
Slide Author: Scott Devine

Controlling Control Flow



Slide Author: Scott Devine

Controlling Control Flow



Slide Author: Scott Devine

Issues with Binary Translation

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

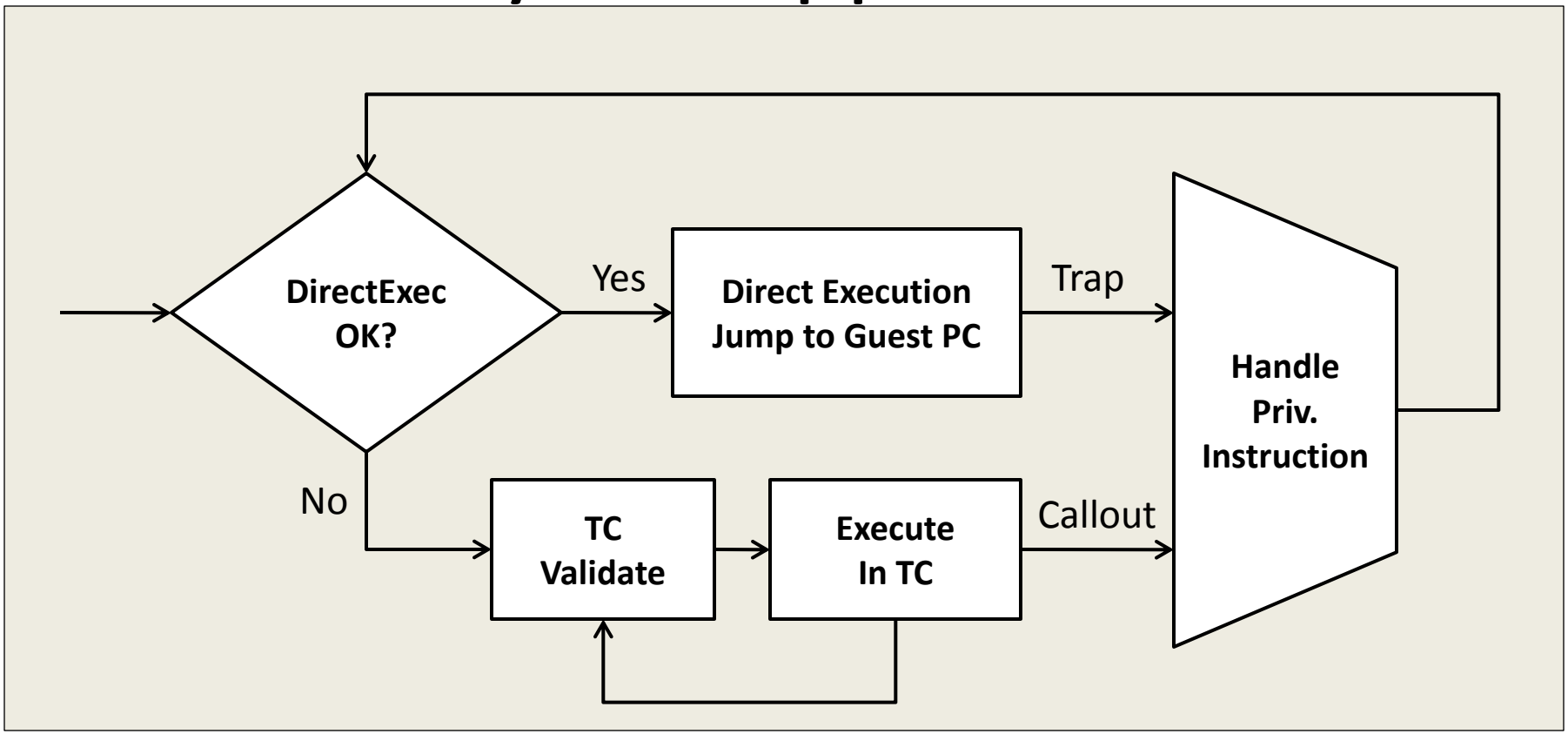
Slide Author: Scott Devine

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

Slide Author: Scott Devine

Hybrid Approach



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

Slide Author: Scott Devine

Binary Translation solution

- Run the guest OS in ring-3
- Is it always possible?
  - When is it possible?
- BT a solution to get-around architectures where this is not possible.
  - Before executing “any” code, study it… if necessary, translate it so that it runs safely.

Binary Translation

- Maintain a software vcpu state.

```
struct vcpu {
        register   regs[8];
        cregister cregs[3];
        …
}
```

- But maintain “most of this state” in actual hardware registers “most of the time”
- For example, all regular registers can be maintained in actual hardware. But control registers will always need to be emulated using vcpu struct fields.

Binary Translation : what do we need

- A method to translate back and forth between the hardware state and the vcpu state

- A way to access the vcpu state from native execution (how: reserve one register to point to vcpu struct, and use it to access it's members)
- e.g.:
  - "mov eax, cr0" translates to "mov eax, vcpu.cr0"

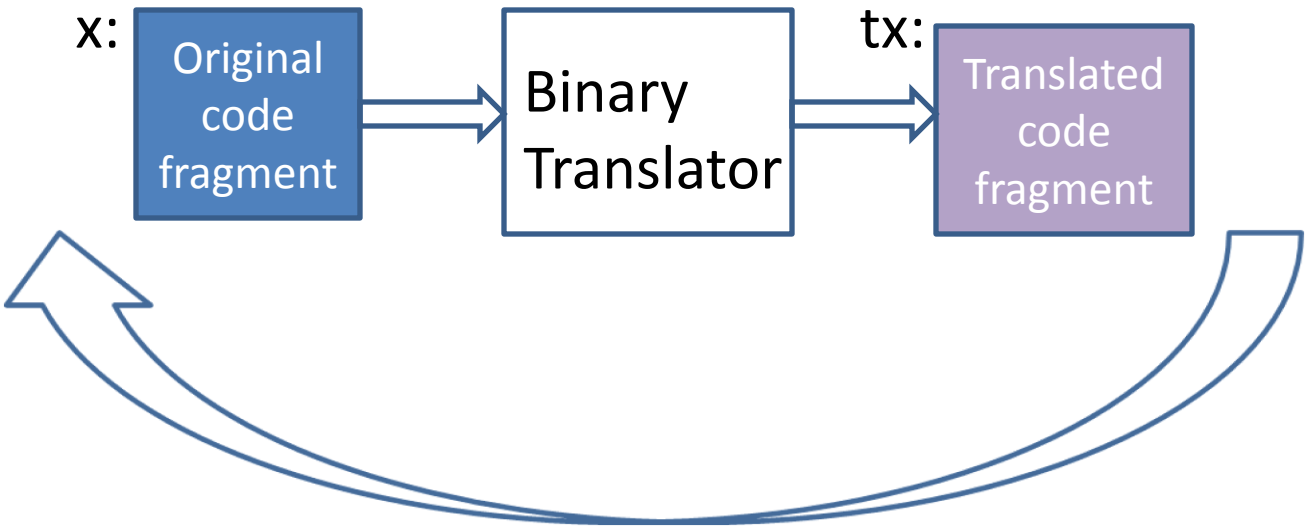
Binary Translation

- Some instructions might need to take the “slow path” and get emulated.
- e.g., mov eax, cr3
  - This instruction specifies that we now have a new page table. We need to update many data structures inside the VMM to reflect this change, so such an instruction is best emulated:
    - Translate hardware state to vcpu
    - Emulate the instruction in software (similar to bochs)
    - Translate the vcpu state back to hardware.

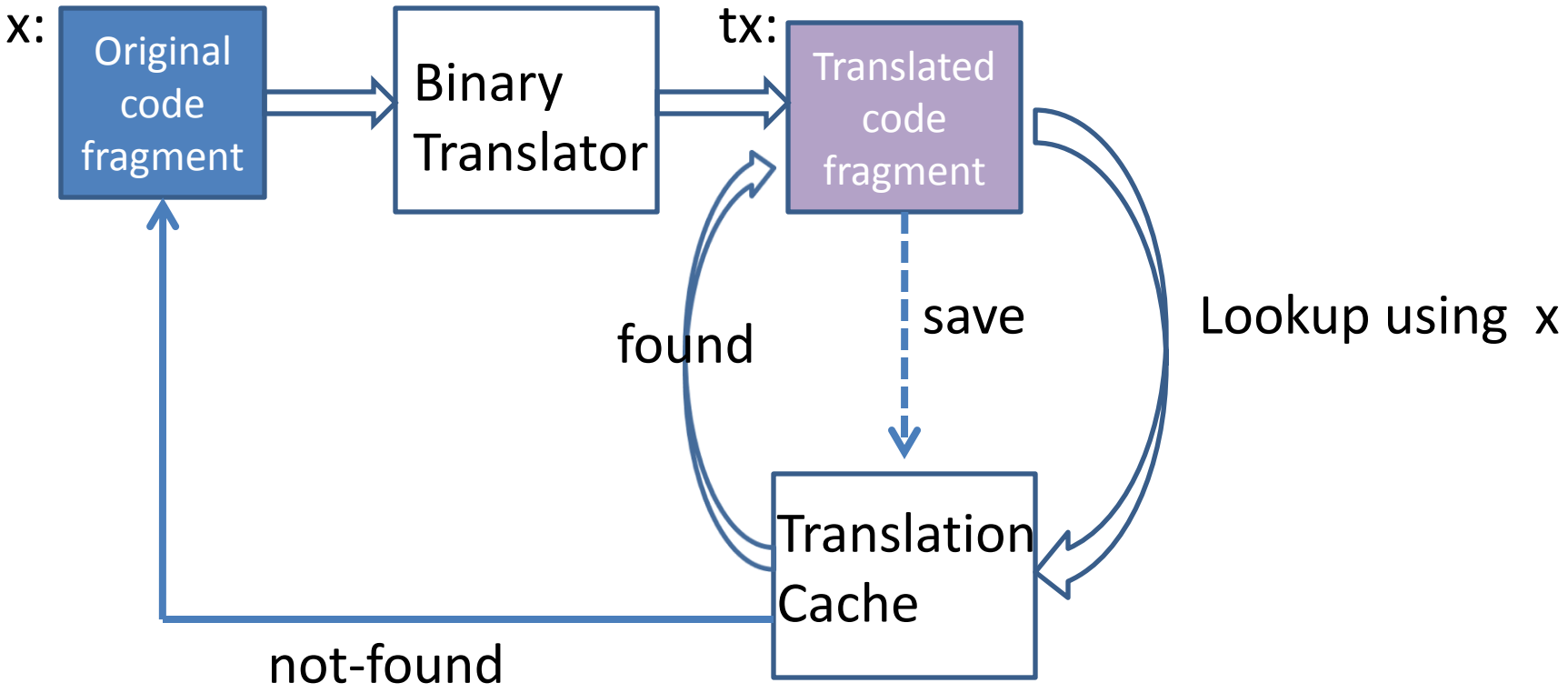
Translation Blocks

- Divide code into “translation blocks”
  - A translation block ends if
    - Reach a control-flow instruction
    - Or, MAX_INSNS instructions have been translated

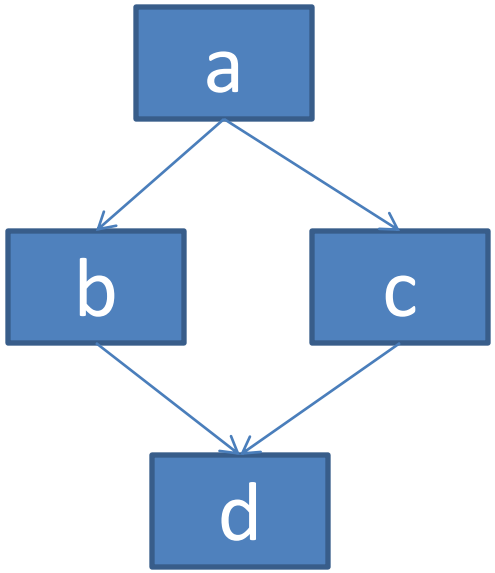
A Simple Scheme

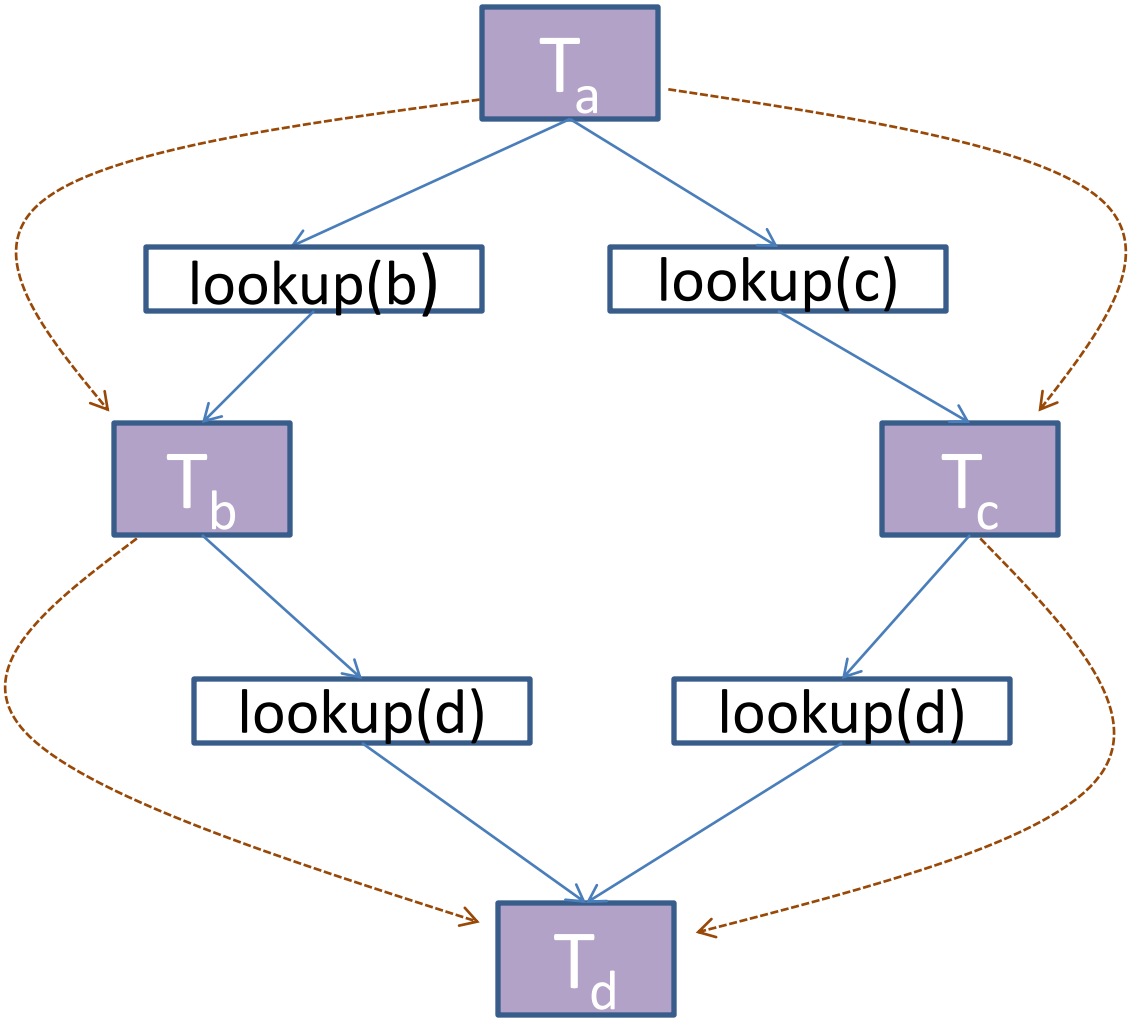


Use a Cache

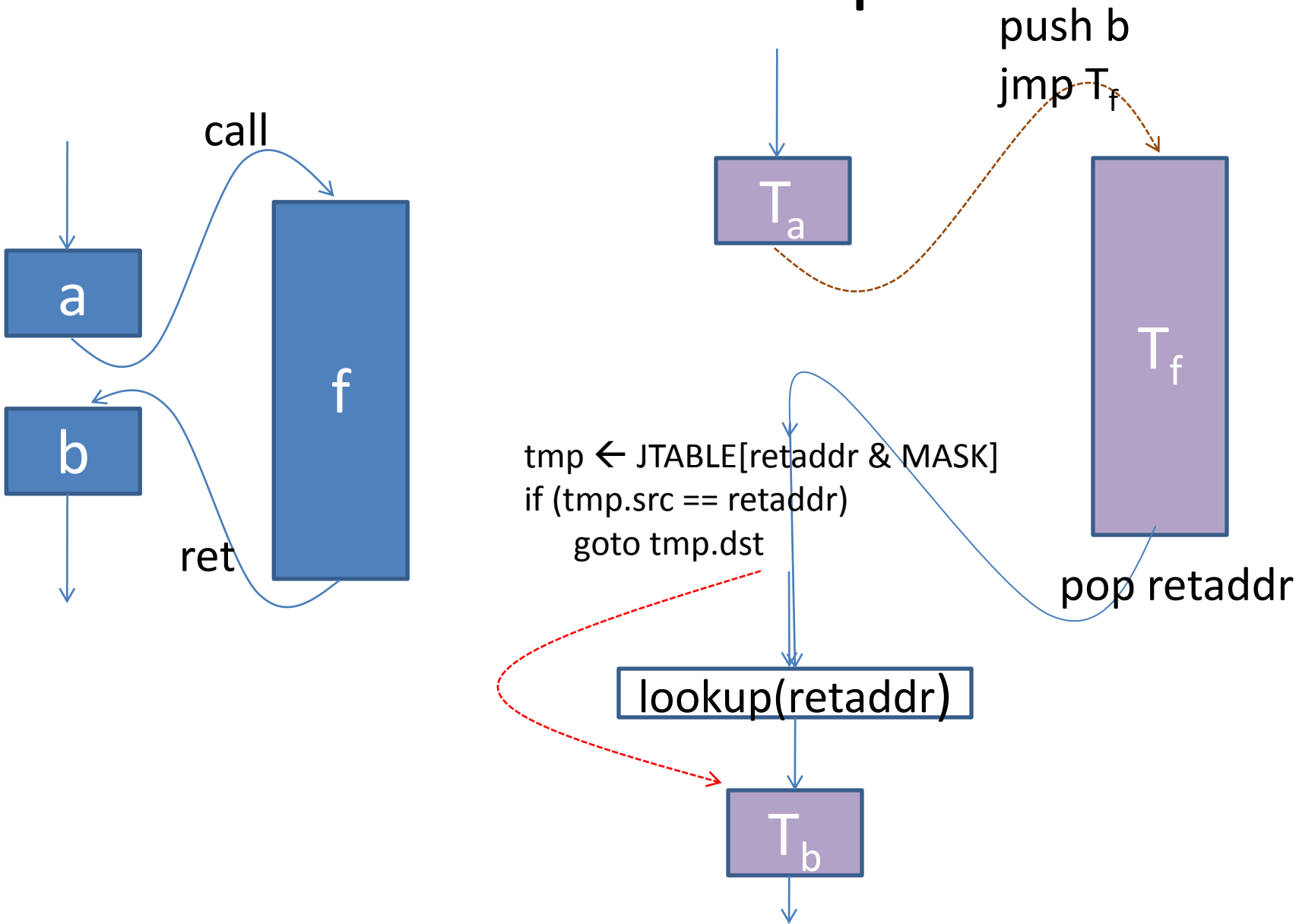

x:
Original code fragment
Binary Translator
tx:
Translated code fragment
save
Lookup using x
found
Translation Cache
not-found

Direct Jump Chaining





Indirect Jumps



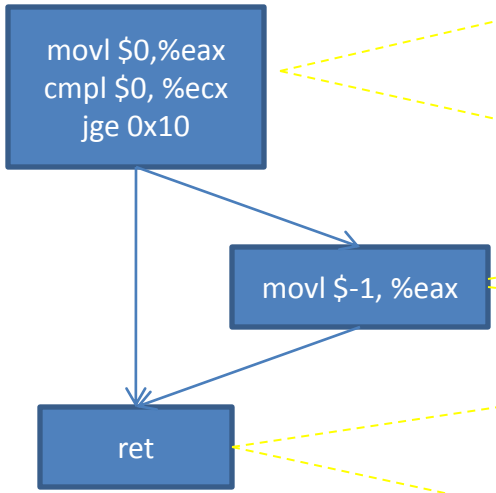
Binary Translation Example

f(x) = -1 if x < 0

0 ow

Original:

```
0x0:   movl   $0, %eax
0x5:   cmpl   $0,  %ecx
0x8:   jge    0x10
0xa:   movl   $-1,  %eax
0x10:  ret
```



Translated:

```
0x800:   movl   $0, %eax
0x800:   cmpl   $0,  %ecx
0x803:   jge    [0x10-trans]
0x807:   jmp    [0xa-trans]
...   :  ...
0x90a:   movl   $-1,  %eax
...   :  ...
0xa10:   movl   (%esp), %esi
0xa15:   jmp    lookup
0xa17:   movl   %esi, (%esp)
0xa1c:   ret
```

NOTES:

- [0xabc-trans] represents the address of the translated code for the basic block starting at 0xabc in original code.
- lookup() is a function that converts 0xabc to [0xabc-trans]. Both input and output of this function are assumed to be in the register %esi. The function must not modify memory or stack in any way
- If a basic block starting at 0xabc has not been previously translated, a control transfer to [0xabc-trans] transfers control to the binary translator. Once the translation has been done, all subsequent translations jump directly to the translated code
- It is possible to optimize away some “jmp” instructions by more intelligent code placement

Binary Translation

- Perform translation in “translation blocks”
  - A translation block is a straight-line code fragment starting at a particular address
  - A translation block is identified by the starting address
- For direct jumps and function calls
  - Transfer control to the translated address
- For indirect jumps and function returns
  - Use the lookup() function
- For privileged instructions and I/O requests
  - Binary translate to emulate in software
  - May need to execute a few times to identify instructions that can trap

Hardware Virtualization

- Making x86 virtualizable
  - Host mode and guest mode.
    - Any privileged instruction executing in guest mode must trap to the VMM running in host mode.
  - Optimization: VMCBs (virtual machine control blocks)
    - Shadow state for guest maintained by hardware
    - Many privileged instructions update/read only VMCB rather than reading the actual hardware
    - VMCB maintained and read by VMM.
- Pros: Writing a VMM much easier
- Cons: Only solves the many-on-one problem. What about security, reproducibility, monitoring? Nesting?

Hardware vs Software Virtualization

Even with all the extra logic in hardware to implement virtualization, software virtualization seems to perform better than hardware virtualization.

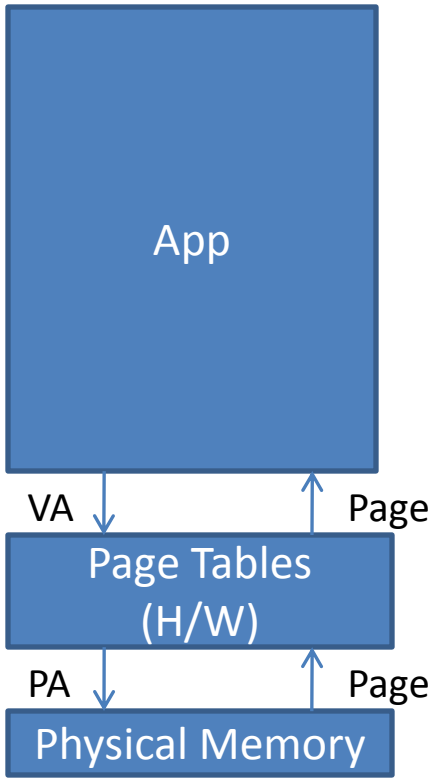
In hardware virtualization, many privileged instructions trap (100s of cycles)

A binary translation system instead replaces the trapping instruction with a few 10s of instructions

K. Adams, O. Agesen. **A Comparison of Software and Hardware Techniques for x86 Virtualization**, *ASPLOS 2006*.

Memory Virtualization

Memory architecture review:



- OS is responsible for setting up and switching between page tables for different processes
- Hardware implements fast table lookup using Translation Lookaside Buffers (TLB)
- Without TLB support, each memory read/write would require 3-5 extra memory accesses to lookup page tables!

Memory Virtualization

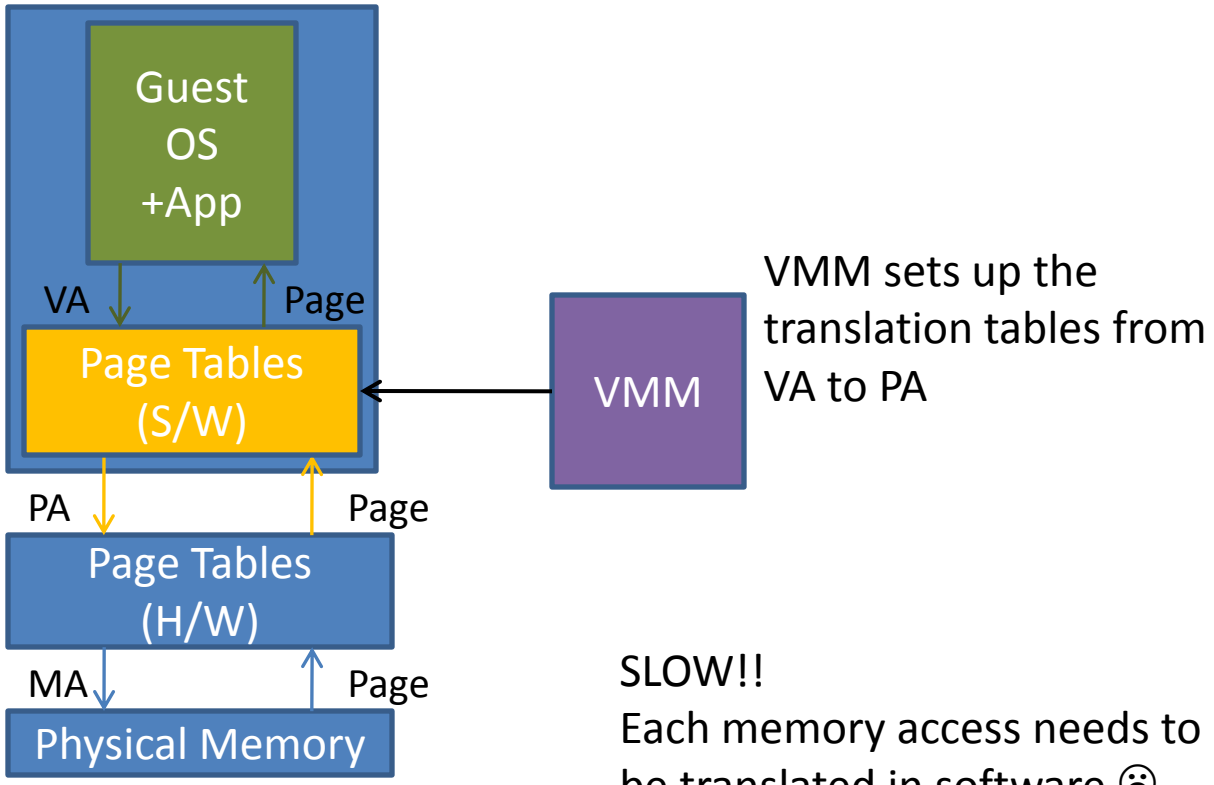
- Now, we need two-level translations:

Guest VA → Guest PA → Host PA

- New terminology in virtual environment:
  - Guest VA = Virtual Address (VA)
  - Guest PA = Physical Address (PA)
  - Host PA = Machine Address (MA)

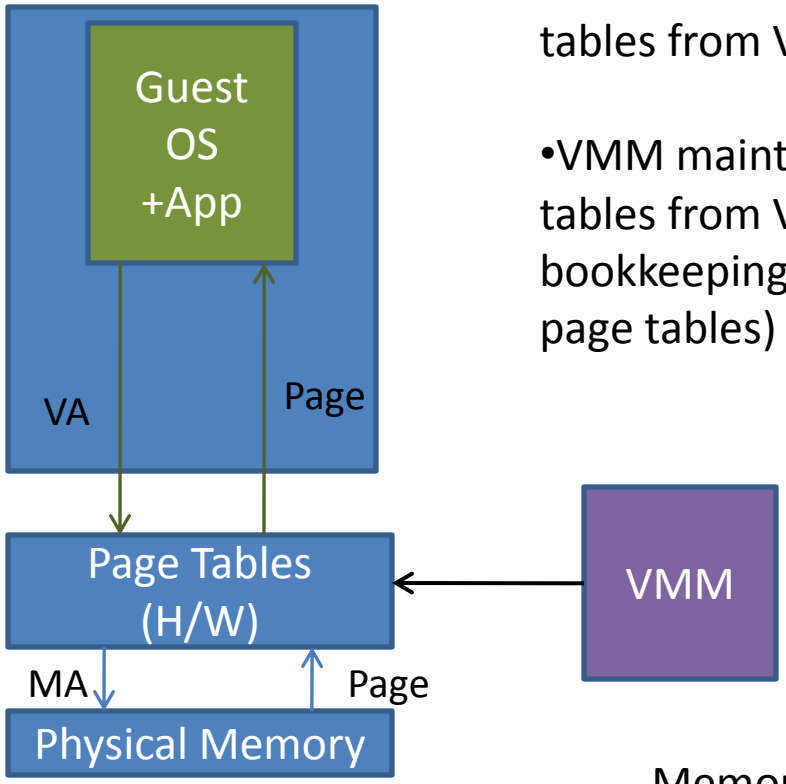
Memory Virtualization

- One option:



Memory Virtualization

- Better Option



•VMM sets up the translation tables from VA → MA directly.

•VMM maintains separate page tables from VA→PA to perform bookkeeping (also called shadow page tables)

Memory access is at near-native speed ☺

Memory Virtualization

- Many interesting issues:
  - How to reclaim memory from a Guest OS?
  - Memory overcommitment
  - Page sharing
  - Performance

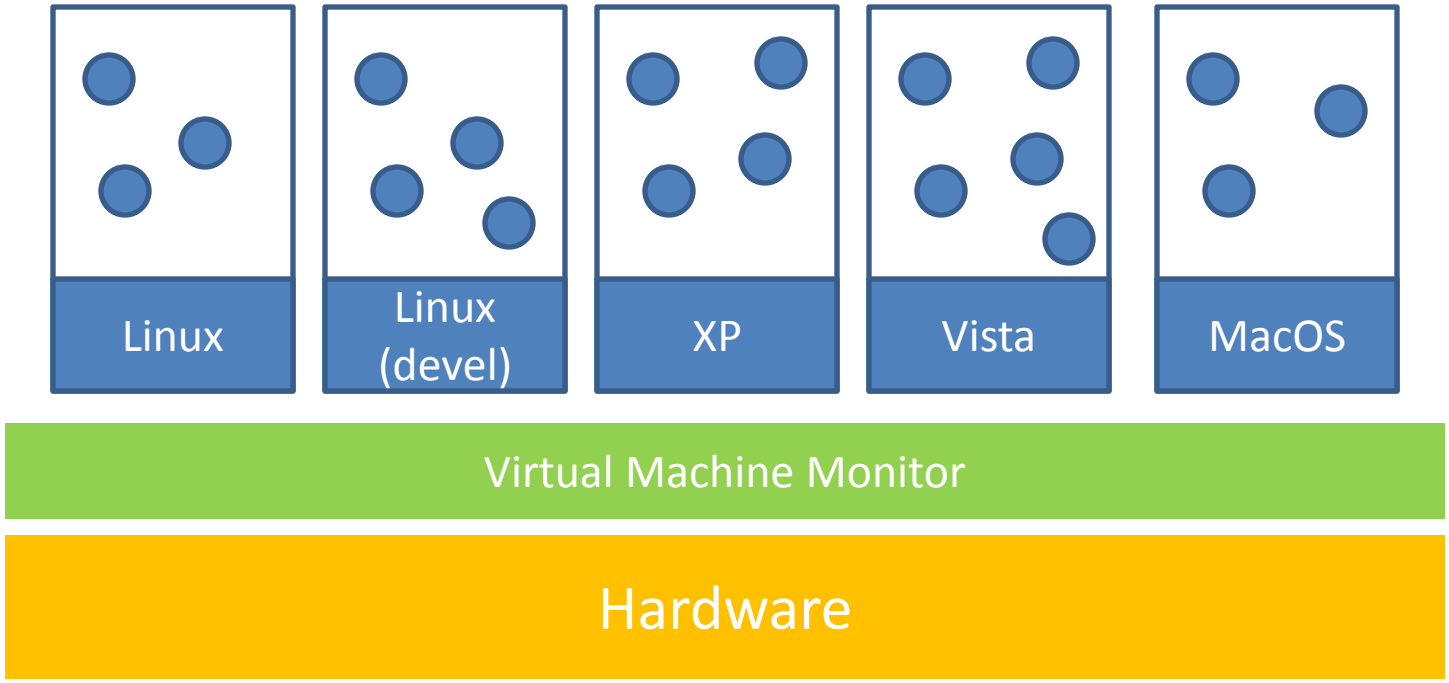
Carl A. Waldspurger. **Memory Resource Management in VMware ESX Server**, *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI '02)*, Boston, Massachusetts, December 2002.

I/O Devices

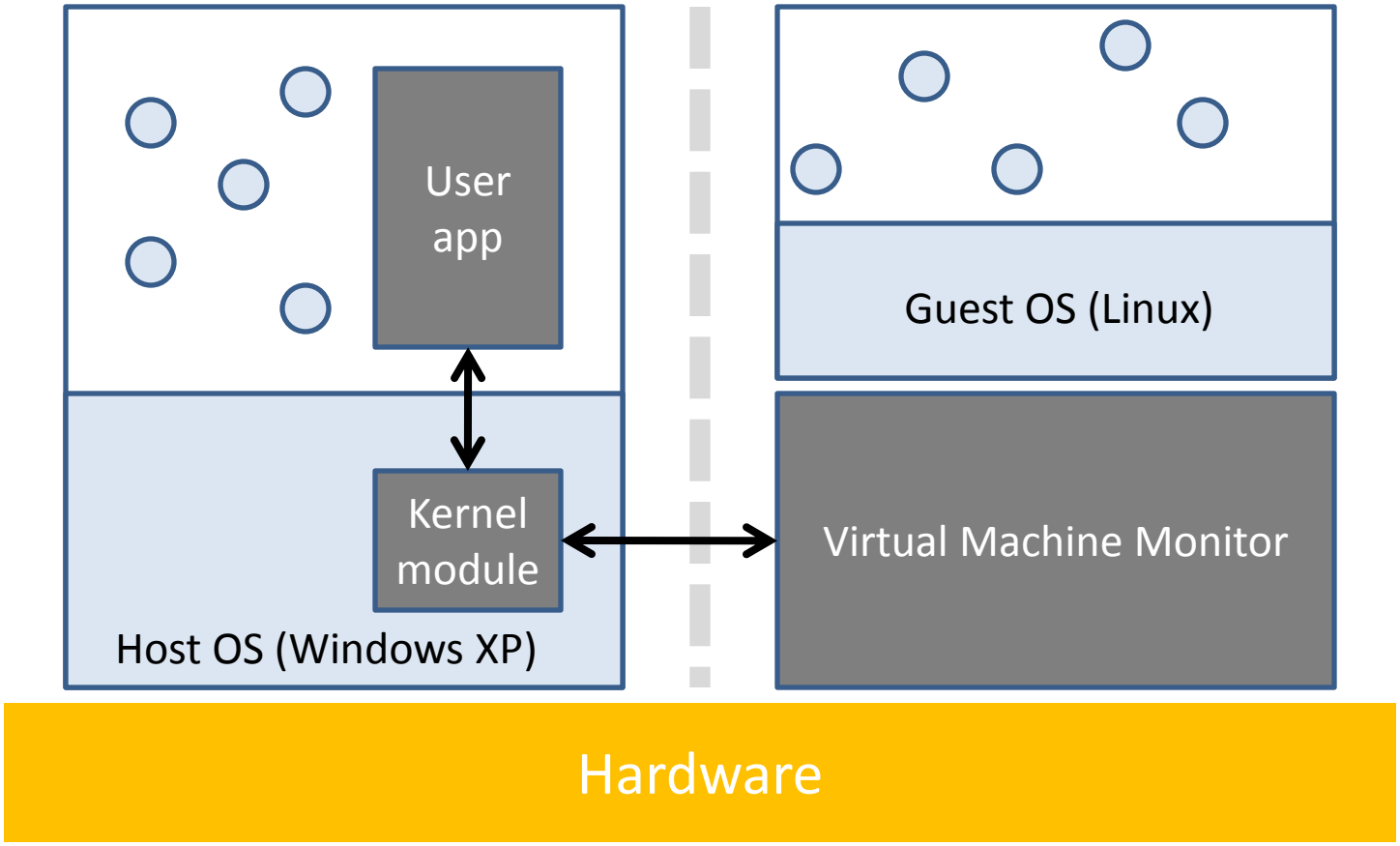
- Simple approach
  - Emulate a common I/O device in software
    - Examples:
      - SCSI Disk: Expose a well-known storage adapter. Store disk data in a file. Each operation decoded and state updated appropriately
      - Network card: Buffer outgoing packets and send them over the network. For incoming packets, generate interrupts into the Guest OS

- J. Sugerman, G. Venkitachalam, B. Lim. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. *USENIX Technical Conference 2002*

Traditional Architecture



Hosted Monitor Architecture



VMware ESX 2.0

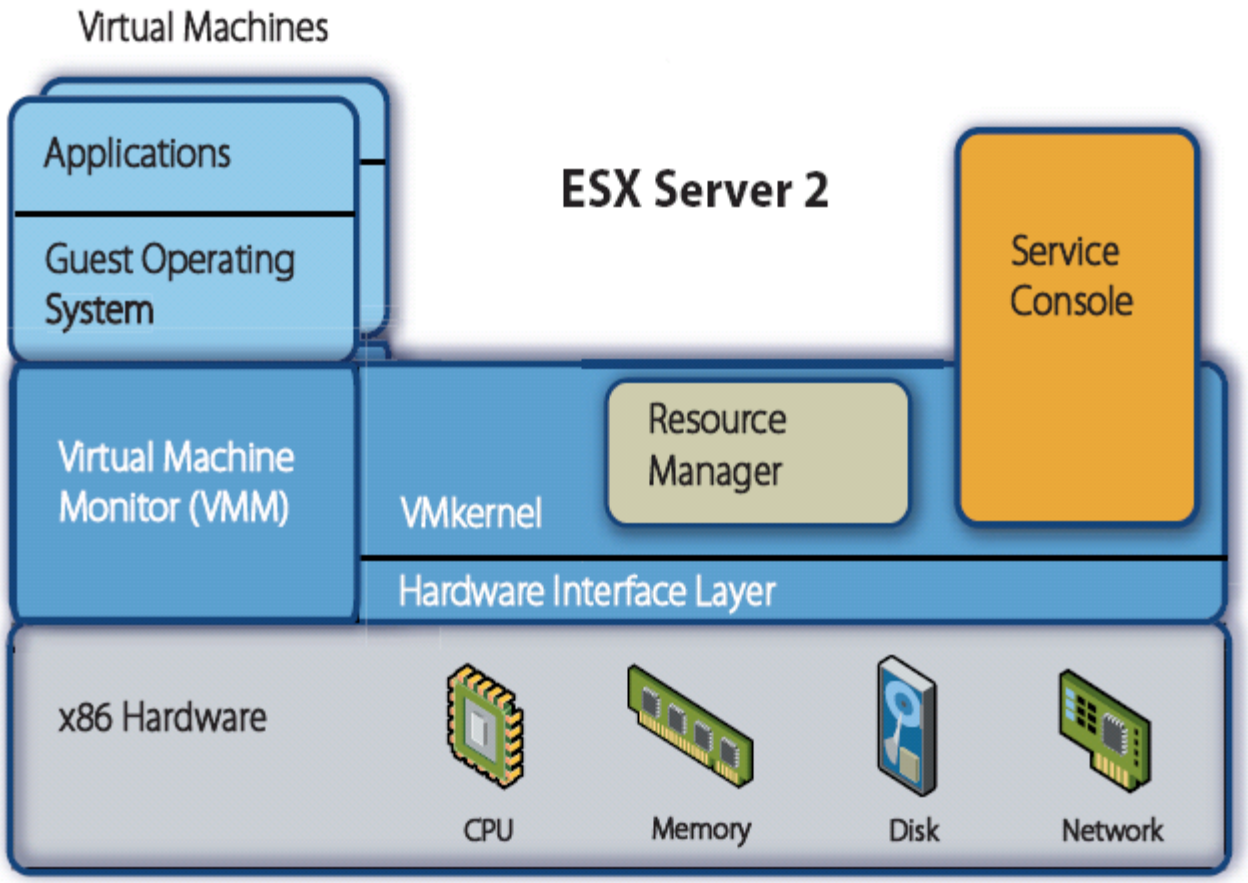
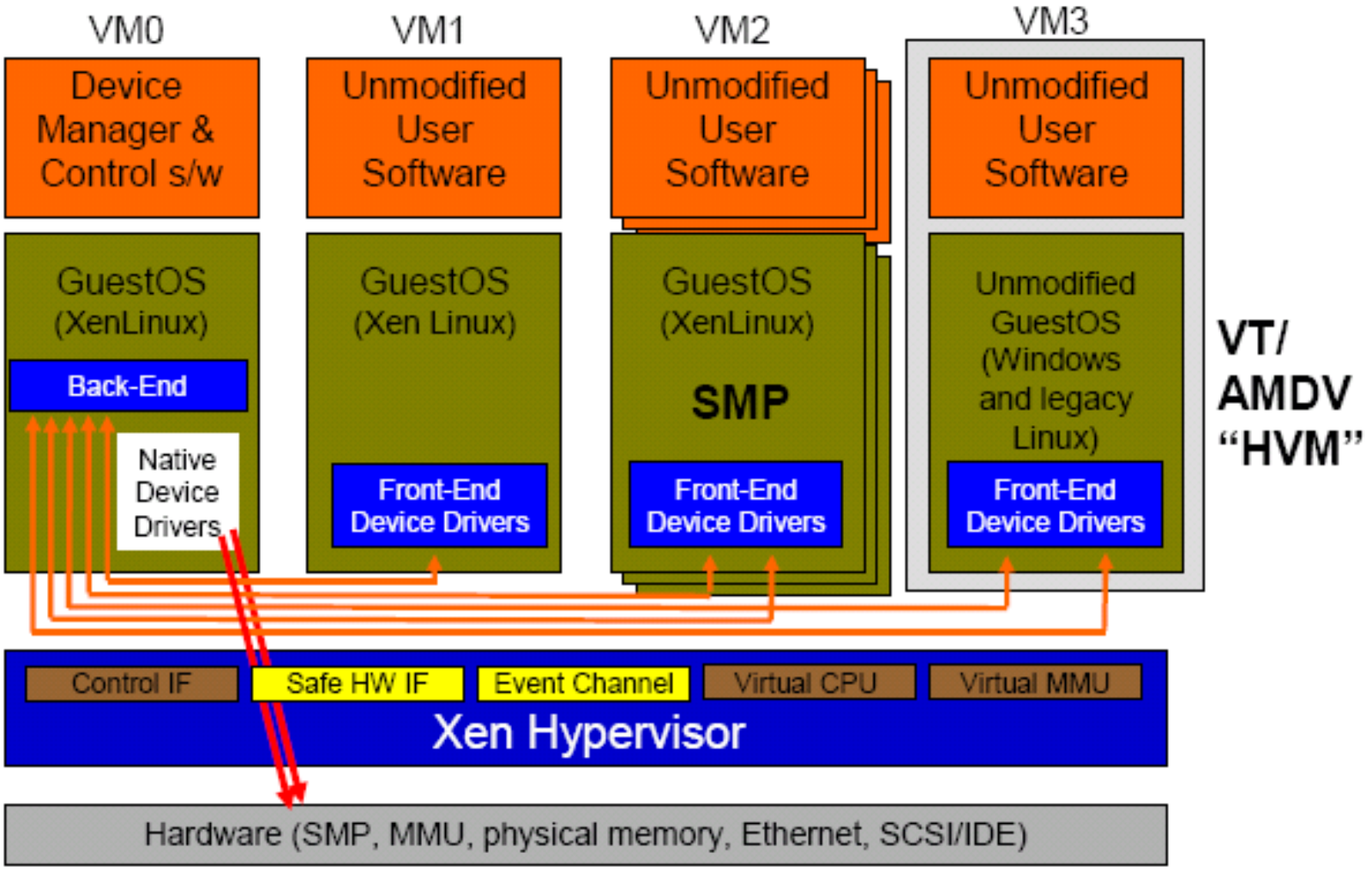


Figure 1: ESX Server architecture

Source: http://www.vmware.com/pdf/esx2_performance_implications.pdf

Xen 3.0



Source: Ottawa Linux Symposium 2006 presentation.
http://www.cl.cam.ac.uk/netos/papers/

Client Virtualization

- VMM on client computers
  - Management (Software Version Control)
  - Homogeneity (Gold image)
  - Security (e.g., VMware ACE)
  - Mobility (e.g., Moka5)
  - Desktop-as-a-service?

- Not as successful (yet)

Looking ahead …

- Ability to give 0.1 of a machine's resources
  - Cloud Computing (e.g., Amazon's EC2)
    - 10 cents per compute hour
    - 10 cents per GB-month
    - 10 cents per GB of data transfer
- Ability to record and replay a machine execution deterministically
  - Software debugging
  - High availability
- Mobile desktops
  - USB stick (Moka5)
  - Leverage the cloud to de-couple data from devices
- Enterprise IT infrastructure
  - Use secured/restricted-use company VMs on employee laptops

Cloud Computing Synonyms

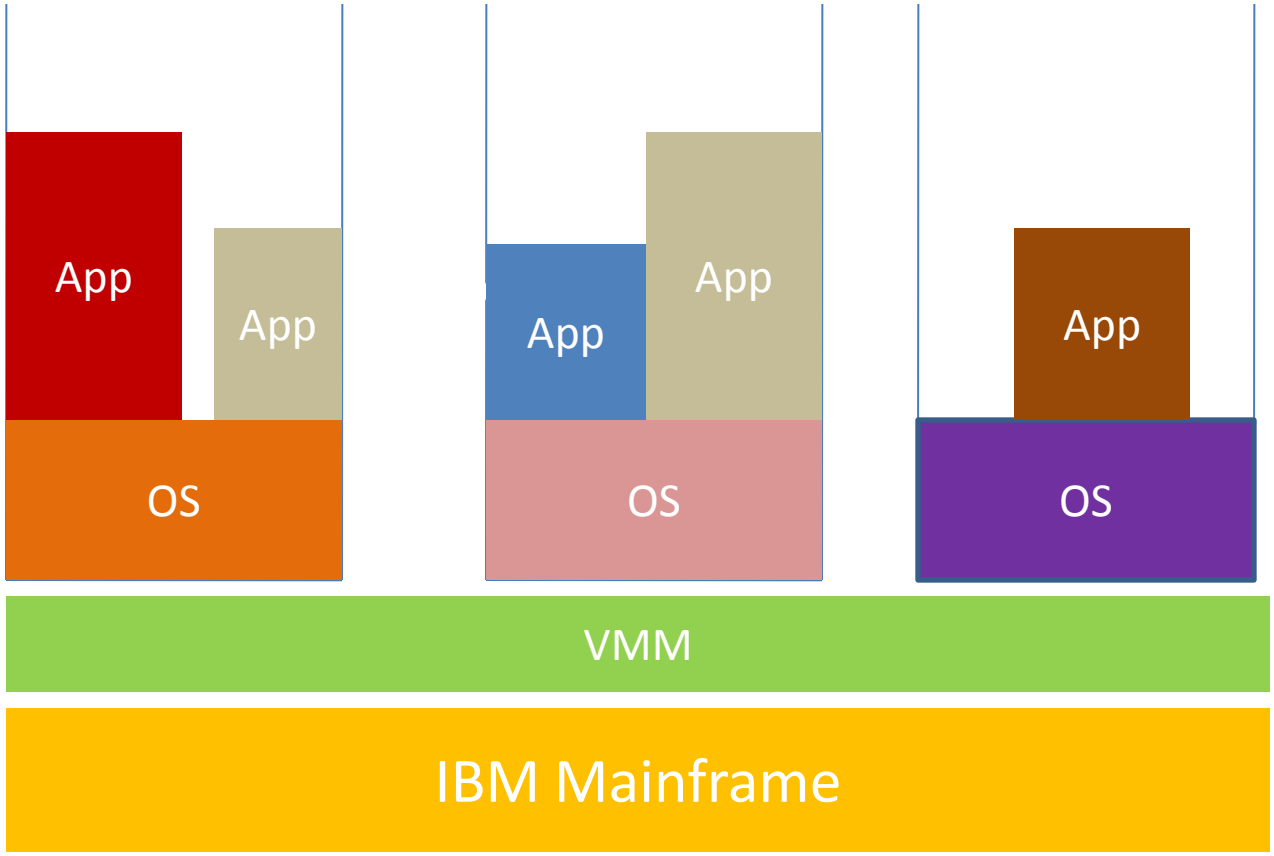
- Platform-as-a-service
- Software-as-a-service
- Grid Computing (Sun)
- Utility Computing (IBM)
- Ubiquitous computing
- IBM Mainframes

What has changed?

- Connectivity
- Many small-to-medium sized users
- Maintenance cost
- Device Variety
- Cost model
- Pricing model

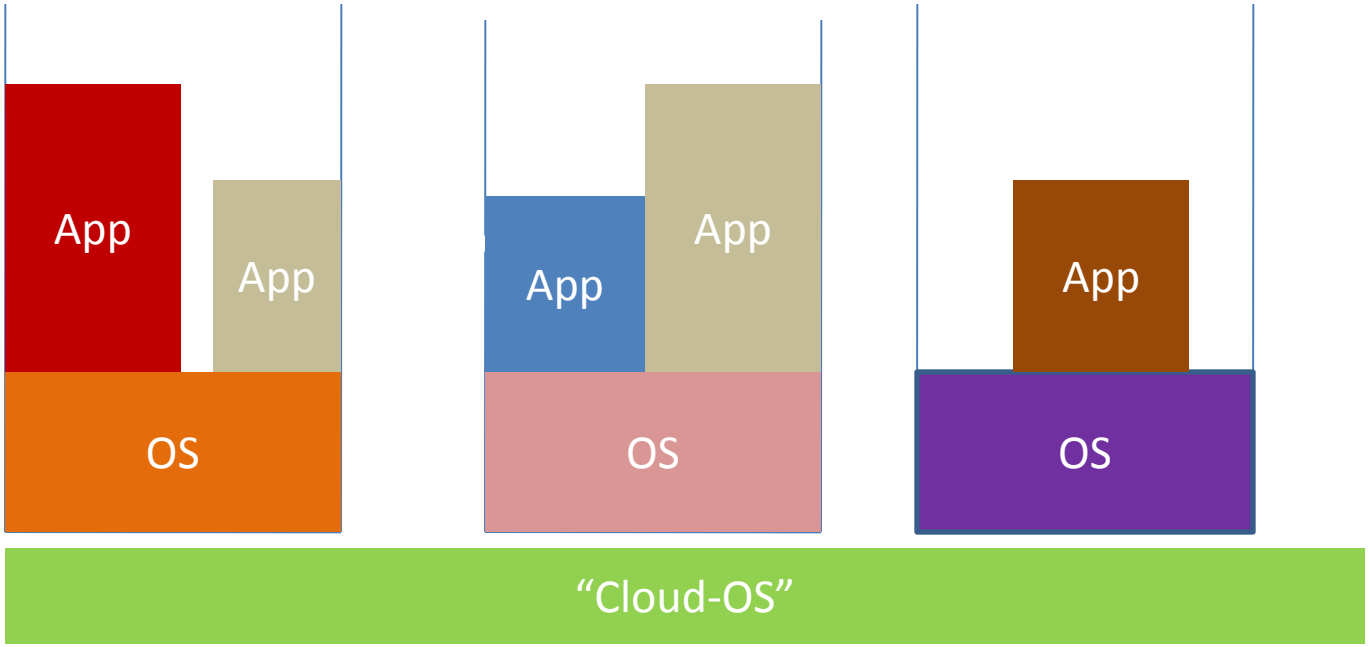
So, are we going back to the mainframes?

IBM Mainframes (circa 1960)



Was a good idea because hardware was expensive

Modern Cloud Environments (2010)



“Cloud-OS”

- Infrastructure Layer (slave) + Management layer (master)
- Unit of abstraction = VM
- Divide hardware into resource pools
- Efficient
- Effective Isolation
- Dynamic
- Fault-Tolerant

Virtualization “Add-ons”

Addons:

- Record/Replay
- Monitoring
- Security
- Software Version Control
- Virtual Appliances