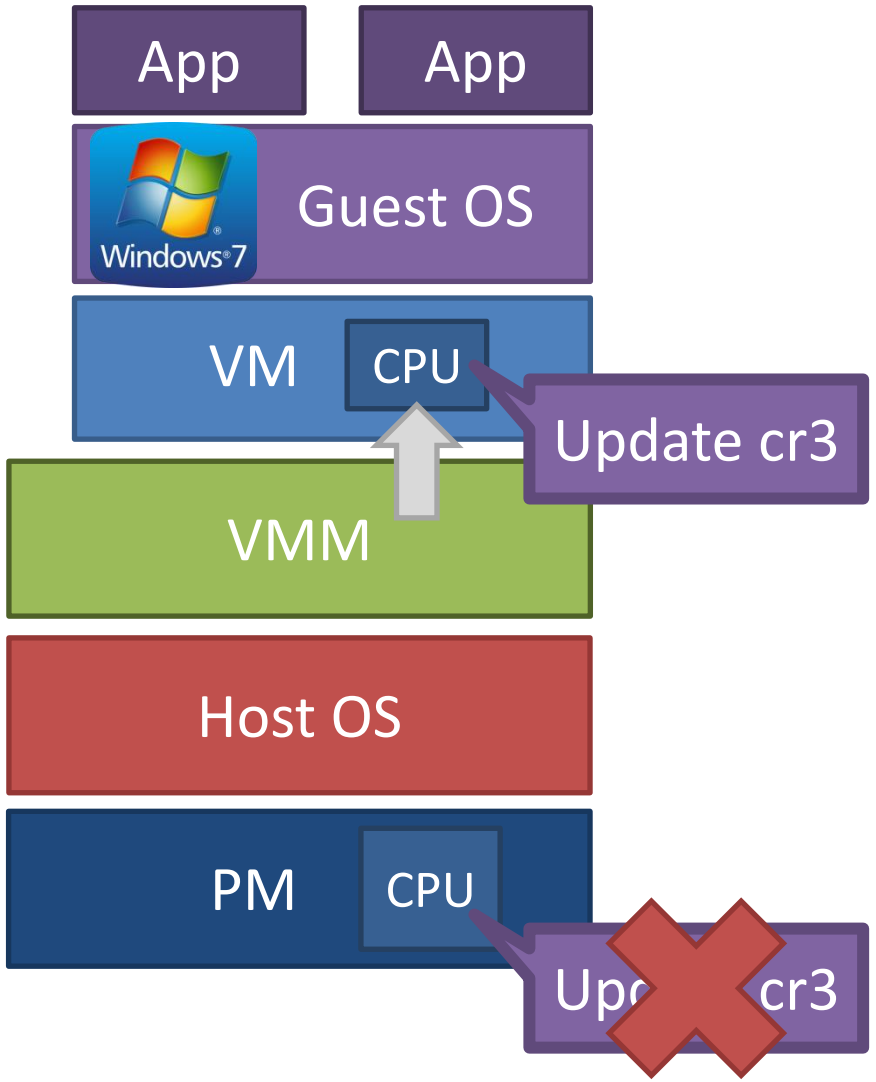
x86 CPU Virtualization

Dr. Amit Praseed

Virtual and Physical CPU

- Each guest has a virtual CPU created by the VMM
- However, the virtual CPU is only used to store state
  - E.g. if a guest updates *cr3* or *eflags*, the new value is stored in the virtual CPU
- Guest code executes on the physical CPU
  - Keeps guest performance high
  - Guests run in userland, so security is maintained



Binary Translation

- **Issue**:
  - Several instructions in the x86 architecture do not generate a trap when executed in user mode
  - Due to this, the VMM cannot emulate these instructions
- Translate the unsafe assembly from the guest to safe assembly
  - Known as binary translation
  - Performed by the VMM
  - Privileged instructions are changed to function calls to code in VMM

A Simple Example

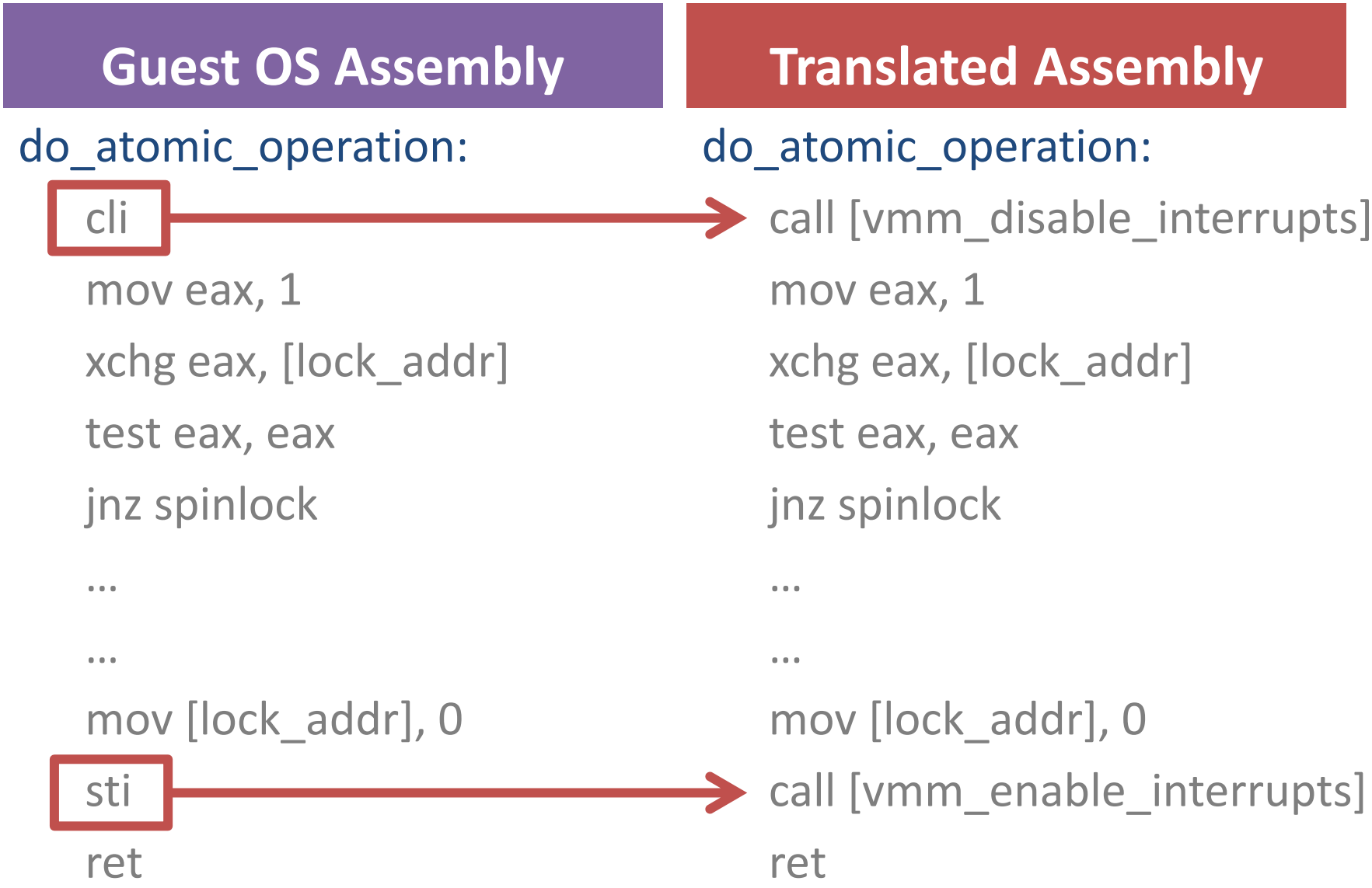
```
push %ebx
mov %eax , %edx
cli
mov $1 , %ecx
xor %ebx , %ebx
jmp doTest
```

The **cli** instruction is meant to clear the interrupt flag in the CPU flag register
But in this case, **it should not perform the action on the physical CPU, but on the virtual CPU**
Hence this instruction should be translated

A Simple Example

```
push %ebx
mov %eax , %edx
and $0xfd, %gs:vcpu.flags
mov $1 , %ecx
xor %ebx , %ebx
jmp doTest
```

Binary Translation Example



Pros and Cons

- Advantages of binary translation
  - It makes it safe to virtualize x86 assembly code
  - Translation occurs dynamically, on demand
    - No need to translate the entire guest OS
  - App code running in the guest does not need to be translated
- Disadvantages
  - Translation is slow
  - Wastes memory (duplicate copies of code in memory)
  - Translation may cause code to be expanded or shortened
    - Thus, jmp and call addresses may also need to be patched

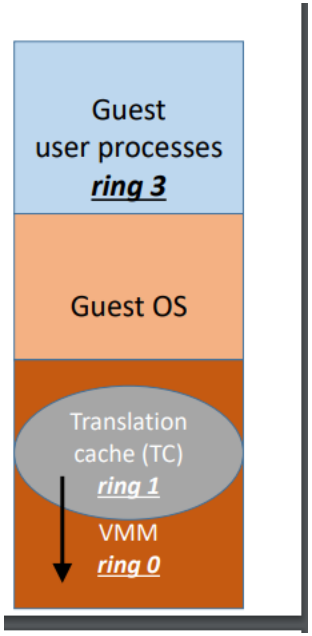
Caching Translated Code

- Typically, VMMs maintain a cache of translated code blocks
  - LRU replacement
- Thus, frequently used code will only be translated once
  - The first execution of this code will be slow
  - Other invocations occur at native speed

The Paravirtualized Solution

- No privileged instructions are run by the guest OS
  - OS is modified – virtualization aware
  - Executes hypercalls to the hypervisor instead of system calls
  - Hypervisor executes the required privileged instruction safely

- Binary translation
  - Guest OS binary is translated instruction-by instruction and stored in translation cache (TC)
  - • Part of VMM memory
  - • Most code stays same, unmodified
  - • OS code modified to work correctly in ring 1
  - • Sensitive but unprivileged instructions modified to trap
  - • Privileged OS code traps to VMM
  - • E.g., I/O, set IDT, set CR3, other privileged ops
  - • Emulated in VMM context or by switching to host • VMM sets sensitive data structures like IDT etc. (maintains shadow copies)

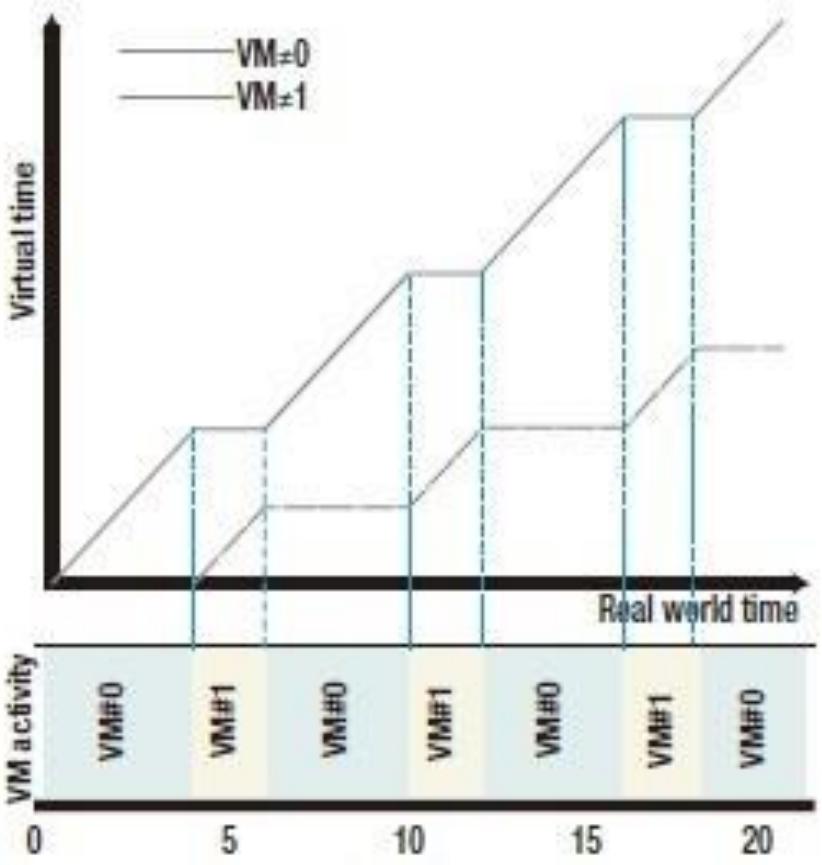


Timing in OS

- Maintaining time is crucial for an OS
  - Job deadlines
  - Task switching
- Real time operating systems especially have strict deadlines for tasks
- In a non-virtualized scenario, this is not an issue, as the OS only needs to maintain a single time value – the wallclock time (or real time)

Real vs Virtual Time

- In a virtualized scenario, there are two concepts of time
  - Real time (or wall-clock time)
  - Virtual Time for different OS
- *A virtual machine's virtual time advances only while the VM (here VM#0 or VM#1) is active*



VM Scheduling

- A scheduling algorithm should be efficient, fair, and starvation-free.
  - The objectives of a scheduler for a batch system are to maximize the throughput (the number of jobs completed in one unit of time, e.g., in one hour) and to minimize the turnaround time (the time between job submission and its completion).
  - The objectives of a real-time system scheduler are to meet the deadlines and to be predictable.

Scheduler Features

- Fair Share vs Proportional Share
  - Proportional Share schedulers provide instantaneous forms of sharing among active clients
  - Fair share schedulers provide a time averaged form of proportional sharing based on usage over a period of time

Scheduler Features

- Work Conserving vs Non-work conserving
  - In WC mode, shares are merely guarantees
  - In NWC mode, shares are caps

Fairness in Scheduling

- **Max-Min Fairness Criterion**: Consider a resource with bandwidth *B* shared among *n* users who have equal rights. Each user requests an amount *bi* and receives *Bi* . Then, according to the max-min criterion, the following conditions must be satisfied by a fair allocation:
  - The amount received by any user is not larger than the amount requested, $Bi \leq bi$.
  - If the minimum allocation of any user is *Bmin* no allocation satisfying condition *1* has a higher *Bmin* than the current allocation.
  - When we remove the user receiving the minimum allocation *Bmin* and then reduce the total amount of the resource available from *B* to *(B − Bmin)*, the condition 2 remains recursively true.

Borrowed Virtual Time (BVT) Scheduling

- Objectives of BVT
  - Support low-latency dispatching of real-time applications
  - weighted sharing of the CPU among several classes of applications
  - supports scheduling of a mix of applications, some with hard, some with soft real-time constraints, and applications demanding only a best effort.

Basics of BVT Scheduling

- Thread *i* has
  - *effective virtual time*, E_i
  - *actual virtual time*, A_i
  - *virtual time warp*, W_i
- The scheduler thread maintains its own *scheduler virtual time (SVT)*, defined as the minimum actual virtual time A_j of any thread.
- The threads are dispatched in the order of their effective virtual time, E_i
  - policy called the earliest virtual time (EVT).

Basics of BVT Scheduling

- The virtual time warp allows a thread to acquire an earlier effective virtual time
  - borrow virtual time from its future CPU allocation.
- The EVT of any thread is calculated as

  $EVT = A_i - (warp? W_i : 0)$
- The thread with the lowest value of EVT runs
  - The time warp allows real time processes to execute faster

Basics of BVT Scheduling

- Each thread also has a warp time limit L_i and an unwarp time requirement U_i
  - Thread i is allowed to run warped for at most L_i
  - If thread i attempts to warp after having previously warped within U_i , the scheduler runs it unwarped until at least time U_i has passed.
- This prevents the starvation of other processes
- BVT Scheduler is one of the schedulers that was used in Xen hypervisor

Simple Earliest Deadline First (sEDF)

- Each domain is set to run for *n ms* every *m ms*
- Can work in both WC and NWC mode

Credit Scheduler

- Every domain has a weight and a cap
  - Weight → share of CPU the domain gets
  - Cap → Maximum share of CPU
- Operates in WC or NWC mode
- Can execute tasks from other physical CPUs