IOWA STATE UNIVERSITY
Department of Electrical and Computer Engineering

Lecture 38: Virtualization & Cloud I

Agenda

- Recap
- Virtual Machines (VMs)

Recap: Buffer Overflow

parse.c

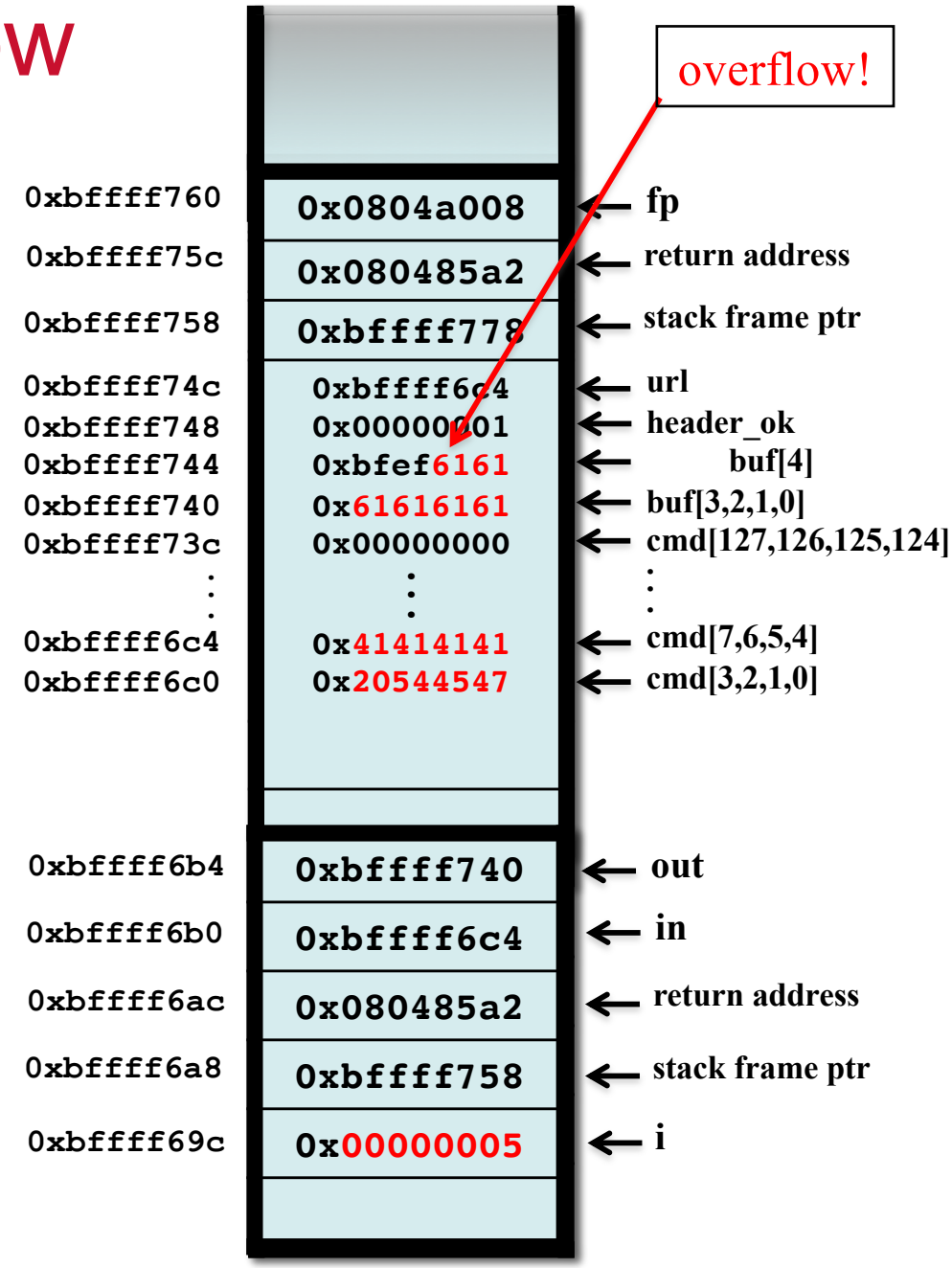
```
 1:void copy_lower (char* in, char* out) {
 2:   int i = 0;
 3:   while (in[i]!='\0' && in[i]!='\n') {
 4:     out[i] = tolower(in[i]);
 5:     i++;
 6:   }
 7:   out[i] = '\0';
 8:}

 9:int parse(FILE *fp) {
10:   char buf[5], *url, cmd[128];
11:   fread(cmd, 1, 128, fp);
12:   int header_ok = 0;
.
.
.
19:   url = cmd + 4;
20:   copy_lower(url, buf);
21:   printf("Location is %s\n", buf);
22:   return 0; }

23: /** main to load a file and run parse */
```

input file (read to cmd[128])

```
GET AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```



Recap: Buffer Overflow

parse.c

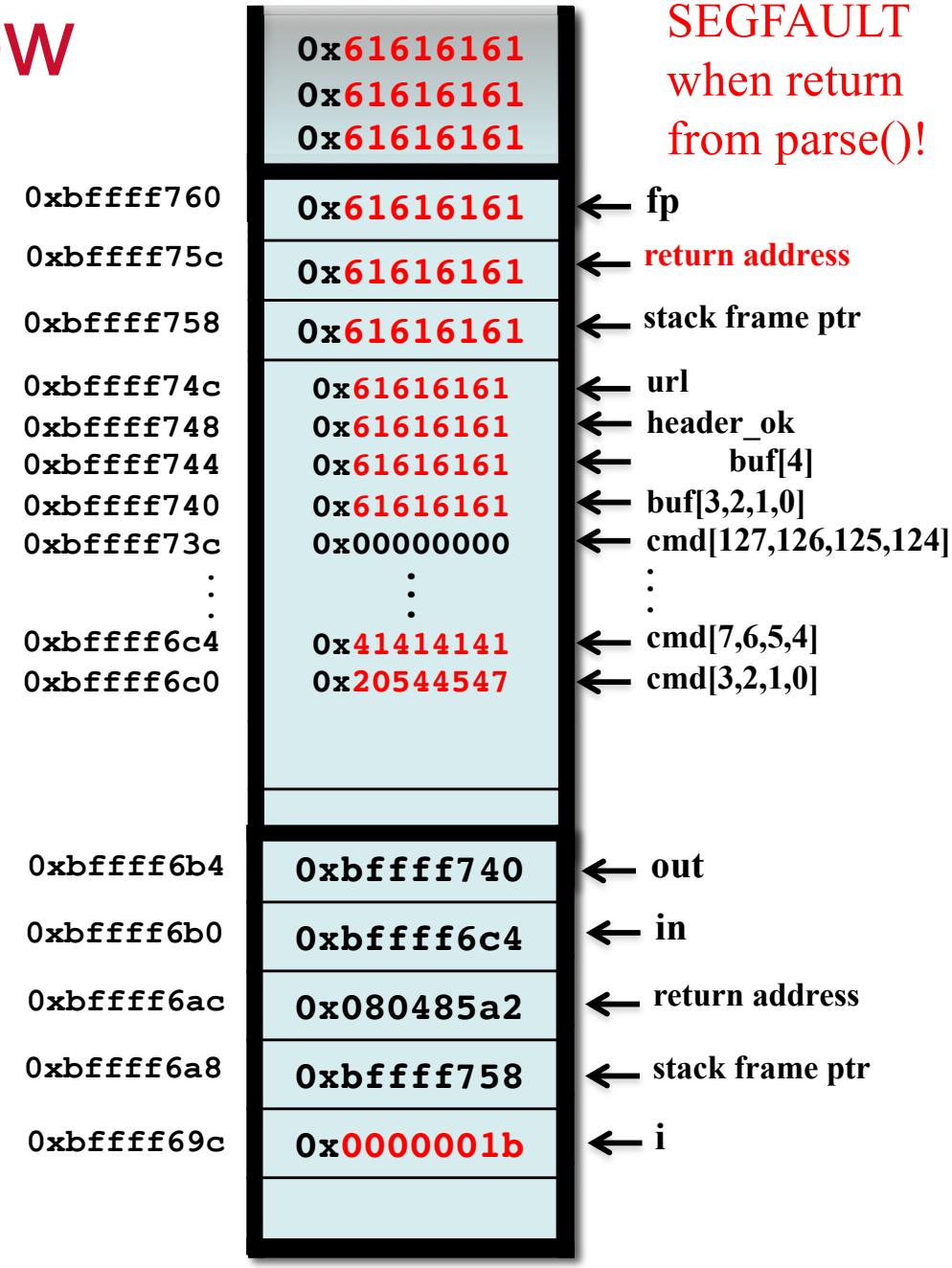
```
 1:void copy_lower (char* in, char* out) {
 2:   int i = 0;
 3:   while (in[i]!='\0' && in[i]!='\n') {
 4:     out[i] = tolower(in[i]);
 5:     i++;
 6:   }
 7:   out[i] = '\0';
 8:}

 9:int parse(FILE *fp) {
10:   char buf[5], *url, cmd[128];
11:   fread(cmd, 1, 128, fp);
12:   int header_ok = 0;
.
.
.
19:   url = cmd + 4;
20:   copy_lower(url, buf);
21:   printf("Location is %s\n", buf);
22:   return 0; }

23: /** main to load a file and run parse */
```

input file (read to cmd[128])

```
GET AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```



Recap: Buffer Overflow

parse.c

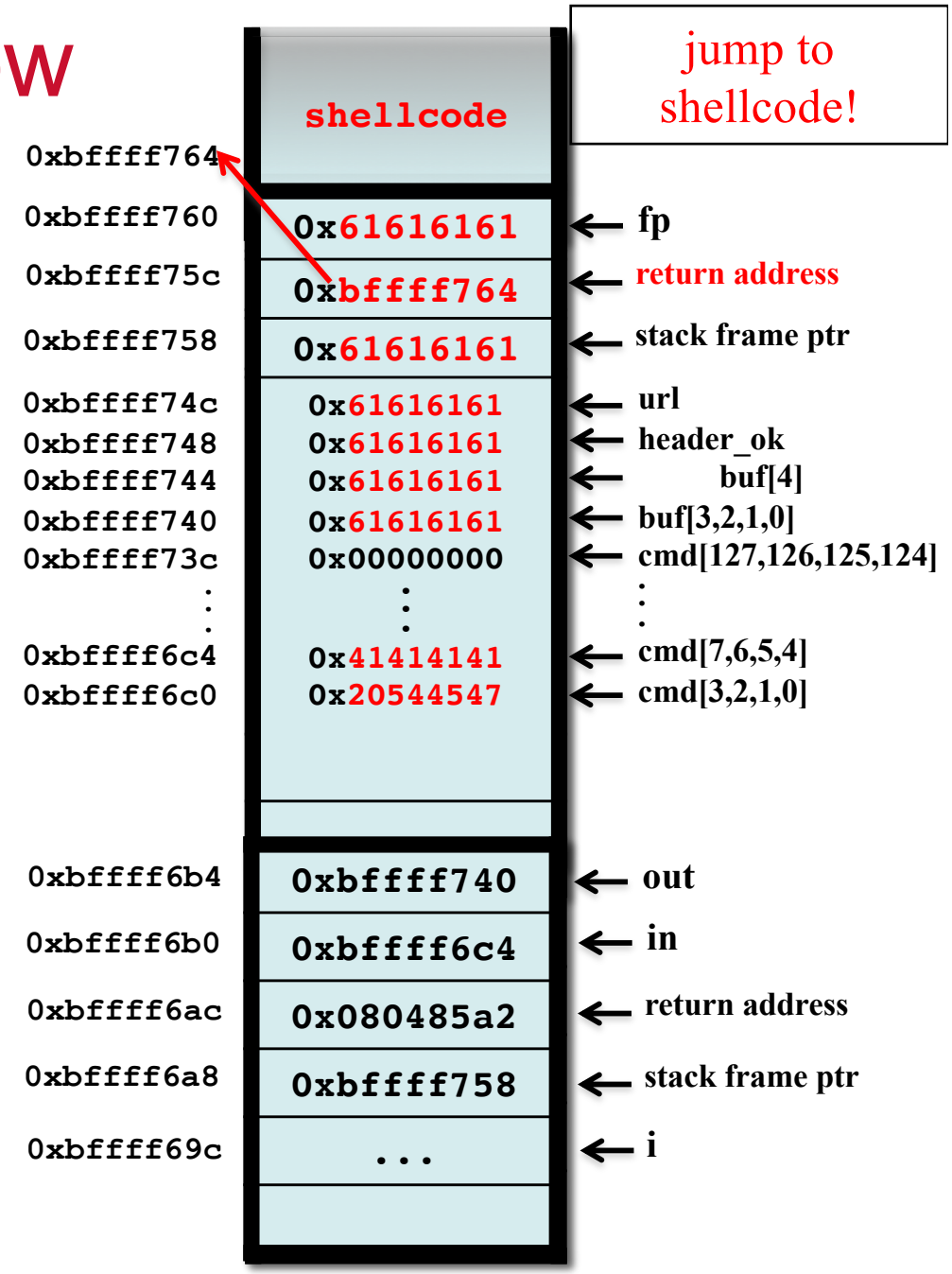
```
 1:void copy_lower (char* in, char* out) {
 2:   int i = 0;
 3:   while (in[i]!='\0' && in[i]!='\n') {
 4:     out[i] = tolower(in[i]);
 5:     i++;
 6:   }
 7:   out[i] = '\0';
 8:}

 9:int parse(FILE *fp) {
10:   char buf[5], *url, cmd[128];
11:   fread(cmd, 1, 128, fp);
12:   int header_ok = 0;
.
.
.
19:   url = cmd + 4;
20:   copy_lower(url, buf);
21:   printf("Location is %s\n", buf);
22:   return 0; }

23: /** main to load a file and run parse */
```

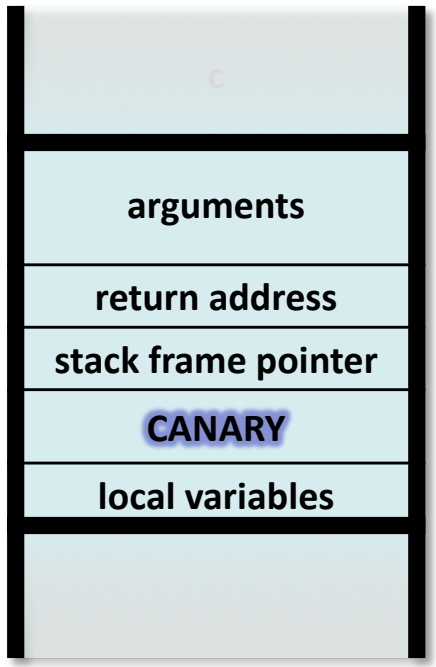
input file (read to cmd[128])

```
GET AAAAAAAAAAAAAAAAAAAA\x64\xf7\xff\xbfAAAA
\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\.....\xff\xff/bin/sh
```



Recap: Defense

- Non-execute
  - Prevent attack code execution by marking stack and heap as **non-executable**
- Address Space Layout Randomization (ASLR)
  - Start stack/heap at a random location
  - Map shared libraries to random location
    - Attacker cannot jump directly to exec function
- StackGuard
  - Embed “canary” in stack frames and verify their integrity prior to function return

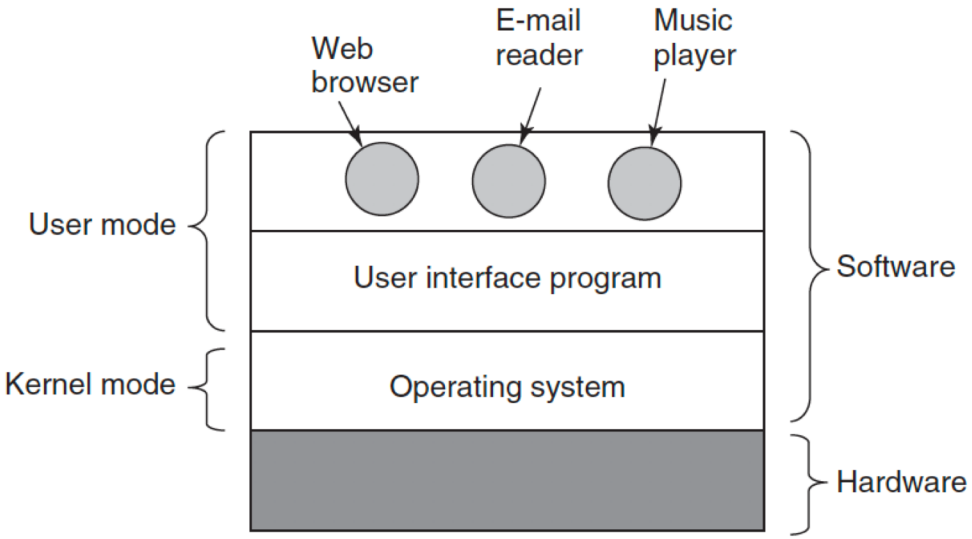


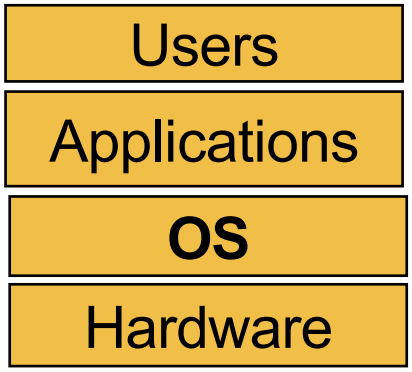
Agenda

- ~~Recap~~
- Virtual Machines (VMs)

Regular Machine V.S. Virtual Machine

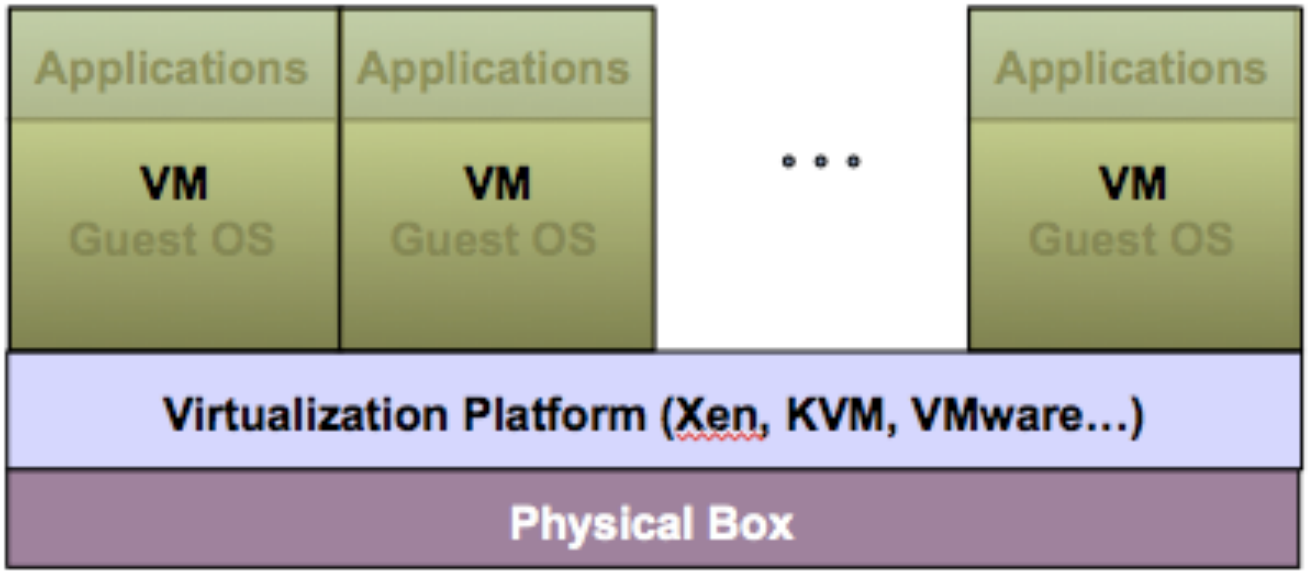
- A regular machine with a regular OS





Regular Machine V.S. Virtual Machine

- Virtual Machines (VMs)
  - A VM is an isolated runtime environment (guest OS + applications)
  - Multiple VMs can run on a single physical system



Why VM

- Manage big machines
  - Multiplex CPUs/memory/devices (e.g., Amazon EC2)
- Multiple OS on one machine
  - E.g., use Windows on Linux OS
- Isolate faults/break-ins
  - One VM is compromised/crashes, others OK
- Kernel development
- ...

Common Goals of VM

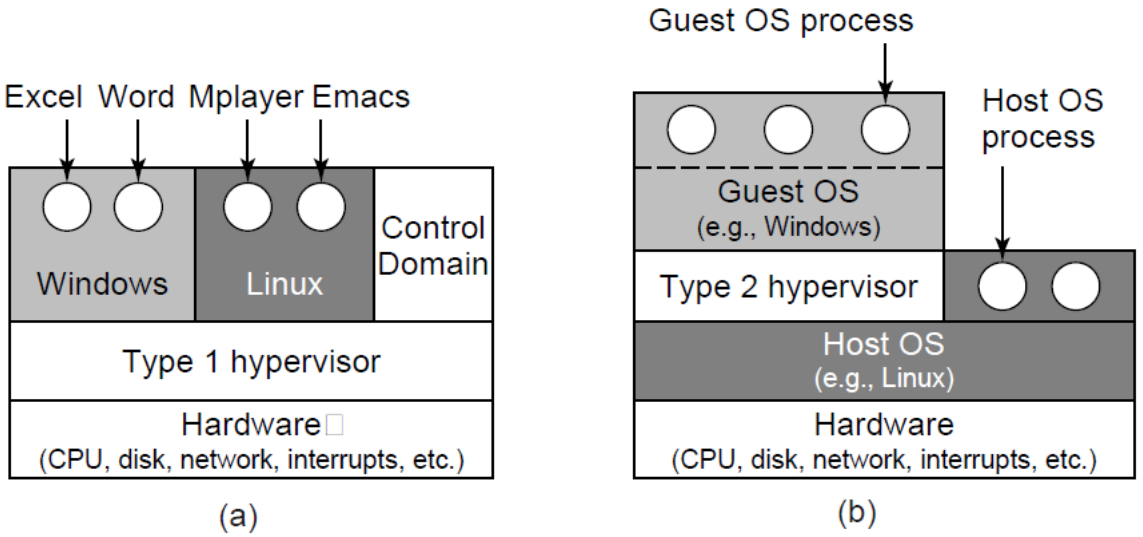
- Accurate
  - Guest cannot distinguish VM from real computer
- Isolated
  - Guest cannot escape VM
- Fast
  - Guest cannot be slowed down (too much)
- ...

Lineage of VM

- 1960s: IBM used VMs to share mainframe
  - VM/370, today's z/VM
  - Still in use!
- 1990s: VMWare re-popularized VMs for x86
  - VMWare ESX servers, VMWare workstation, …
- Today:
  - VMWare, Xen, VirtualBox, KVM, QEMU, ...

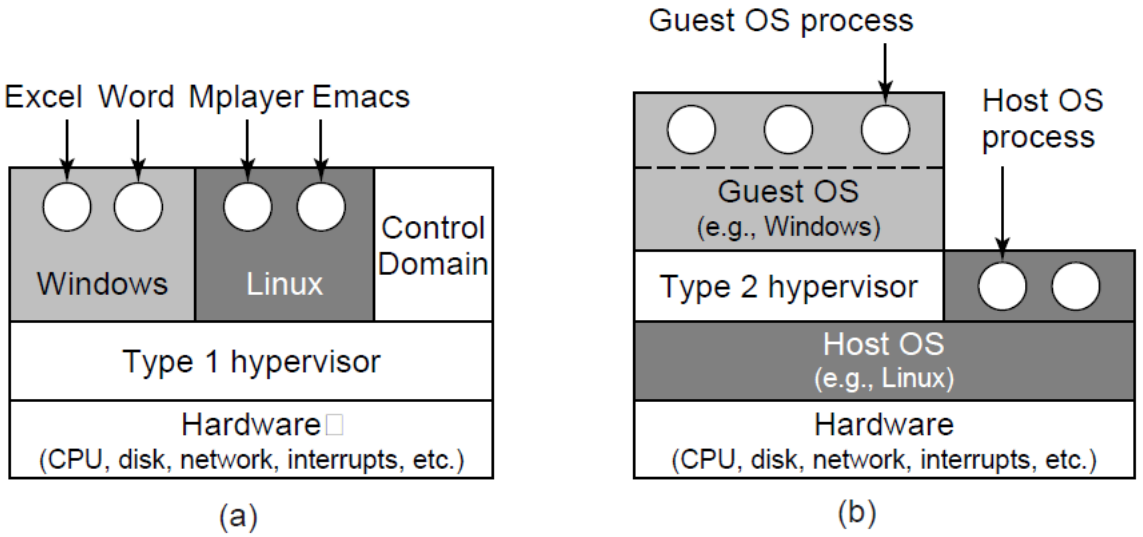
Virtual Machine Monitor (VMM)

- Also called “Hypervisor”
  - Two types
    - Type 1: run on bare metal
    - Type 2: run on a host OS



Virtual Machine Monitor (VMM)

- Responsibilities
  - Time-share CPU among guests
  - Space-share memory among guests
  - Simulate disk, network, and other devices
    - Often multiplex on host devices



Naive Approach: Simulation

```
int32_t regs[8];
#define REG_EAX 1;
#define REG_EBX 2;
#define REG_ECX 3;
...
int32_t eip;
int16_t segregs[4];
...
```

```
for (;;) {
        read_instruction();
        switch (decode_instruction_opcode()) {
        case OPCODE_ADD:
                int src = decode_src_reg();
                int dst = decode_dst_reg();
                regs[dst] = regs[dst] + regs[src];
                break;
        case OPCODE_SUB:
                int src = decode_src_reg();
                int dst = decode_dst_reg();
                regs[dst] = regs[dst] - regs[src];
                break;
        ...
        }
        eip += instruction_length;
}
```

- Interpret each guest instruction
- Maintain each VM state purely in software
- Problem: too slow

2nd Approach: Trap-and-Emulate

- Execute guest instructions on real CPU when possible
  - E.g., addl %eax, %ebx
- Run guest OS in unprivileged mode
- Privileged instructions trap, and VMM emulates
  - E.g., instructions for modifying page table registers
- VMM hides real machine state from guests
  - E.g., virtual registers set by guest, real regesters set by VMM
    - page table, privilege level, interrupt flag, …

2nd Approach: Trap-and-Emulate

- Tricky on x86
  - Not all instructions that should be emulated cause traps
  - Instructions have different effects depending on privilege mode
  - Instructions reading privileged state don't trap
  - Page table modifications don't trap
  - Trap them all: too slow!

3rd Approach: Binary Translation

- Basic Idea
  - Replace non-trapping instructions that read/write sensitive state with trap instruction
  - Keep track of original instruction
  - VMM emulate original instruction in trap
- Dynamic Binary Translation
  - VMWare workstation in 1990s
  - disassemble code only as executed
  - Keep track of what we've translated to avoid retranslate
    - Store translated code in code cache (original -> translated mapping)

4th Approach: Hardware Support

- Virtualization support in CPUs (~2005)
  - Intel VT (Virtualization Technology)
  - AMD SVM (Secure Virtual Machine)
- Basic Idea
  - HW implements a simplified version of VMM
    - HW maintains per-guest virtual state (registers)
    - HW knows it is in “guest mode”
      - Instructions directly modify virtual state
      - Avoids many traps to VMM

Paravirtualization

- Do not aim to present a VM that looks just like the actual underlying hardware
  - Present a machine-like software interface that explicitly exposes the fact that it is a virtualized environment
    - e.g., a set of *hypercalls*, which allow the guest to send explicit requests to the hypervisor for executing privileged sensitive operations
  - Guest has to be aware of the hypervisor API
    - explicitly modified for the hypervisor
    - in cooperation with the hypervisor
- "Xen and the Art of Virtualization" [SOSP'03]

Agenda

- ~~Recap~~
- ~~Virtual Machines (VMs)~~

Questions?

*acknowledgement: slides include content from “Modern Operating Systems” by A. Tanenbaum, “Operating Systems Concepts” by A. Silberschatz etc., “Operating Systems: Three Easy Pieces” by R. Arpaci-Dusseau etc., and anonymous pictures from internet.