E6998 - Virtual Machines
Lecture 6
Topics in Virtual Machine Management

Scott Devine

VMware, Inc.

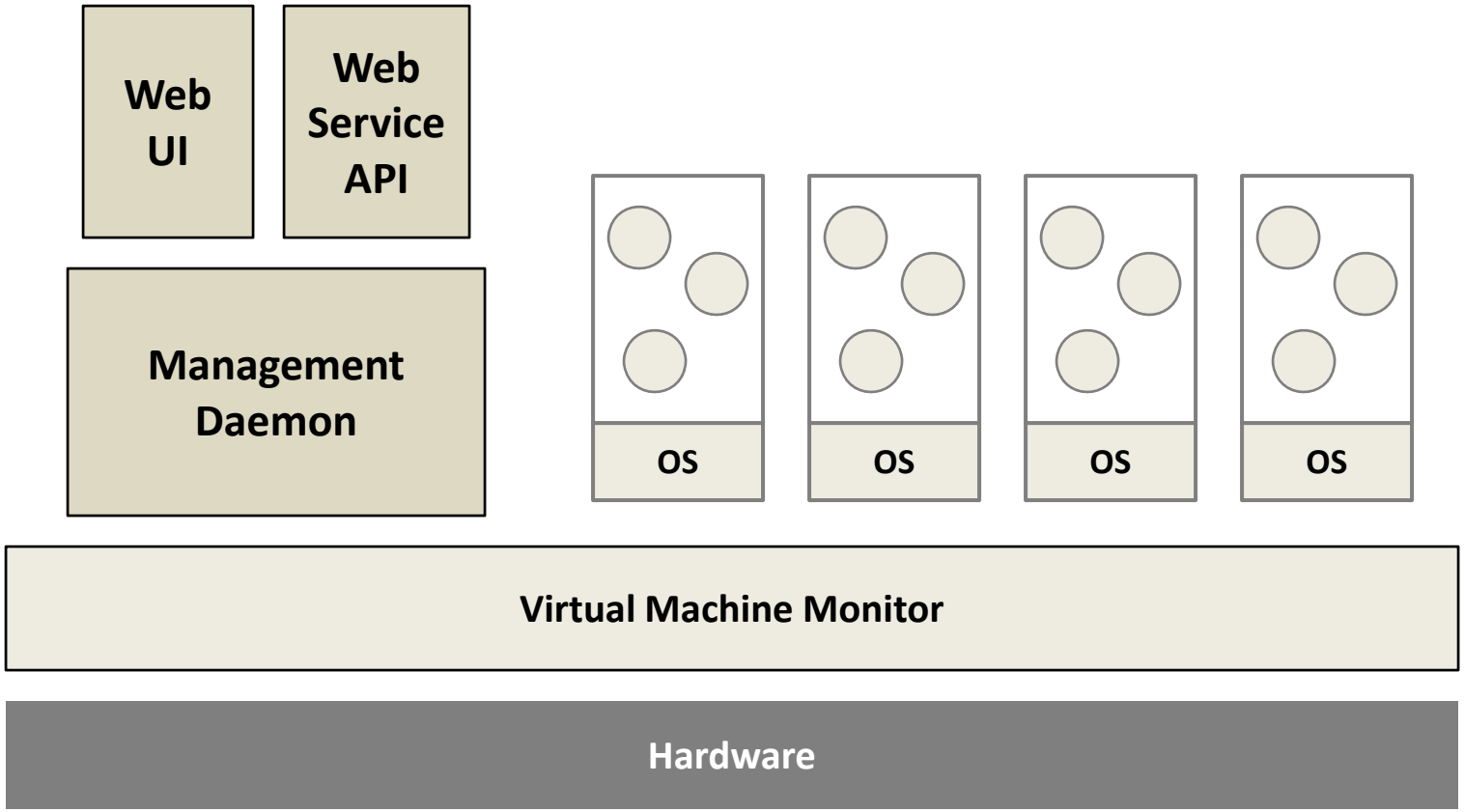
Outline

- **Management Overview**
- **Live Migration**
- **Deterministic Execution**

Covered in Later Lectures

- **Virtual Appliances**
- **Resource Management**
- **Security**

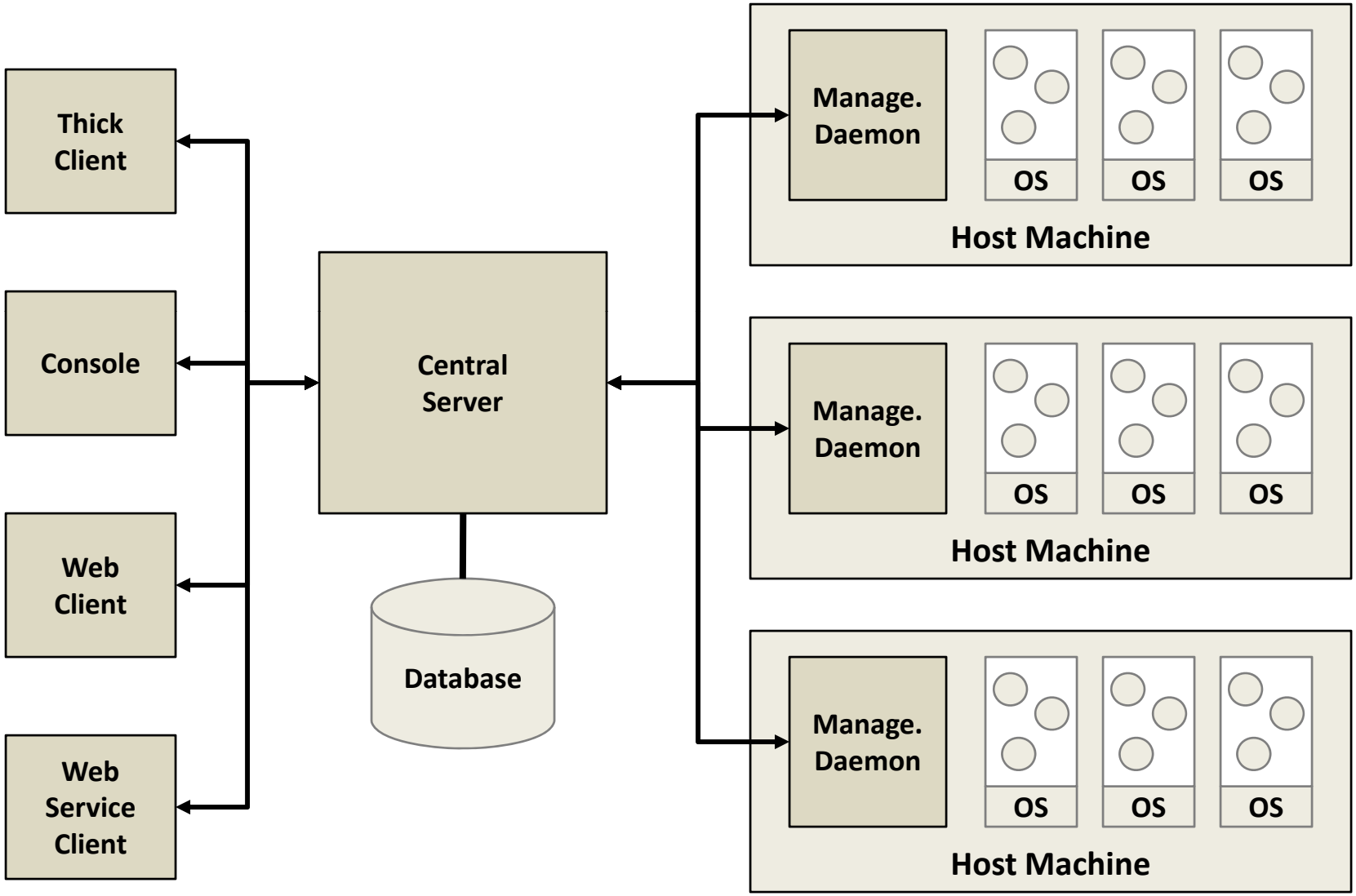
Simple VM Management Architecture



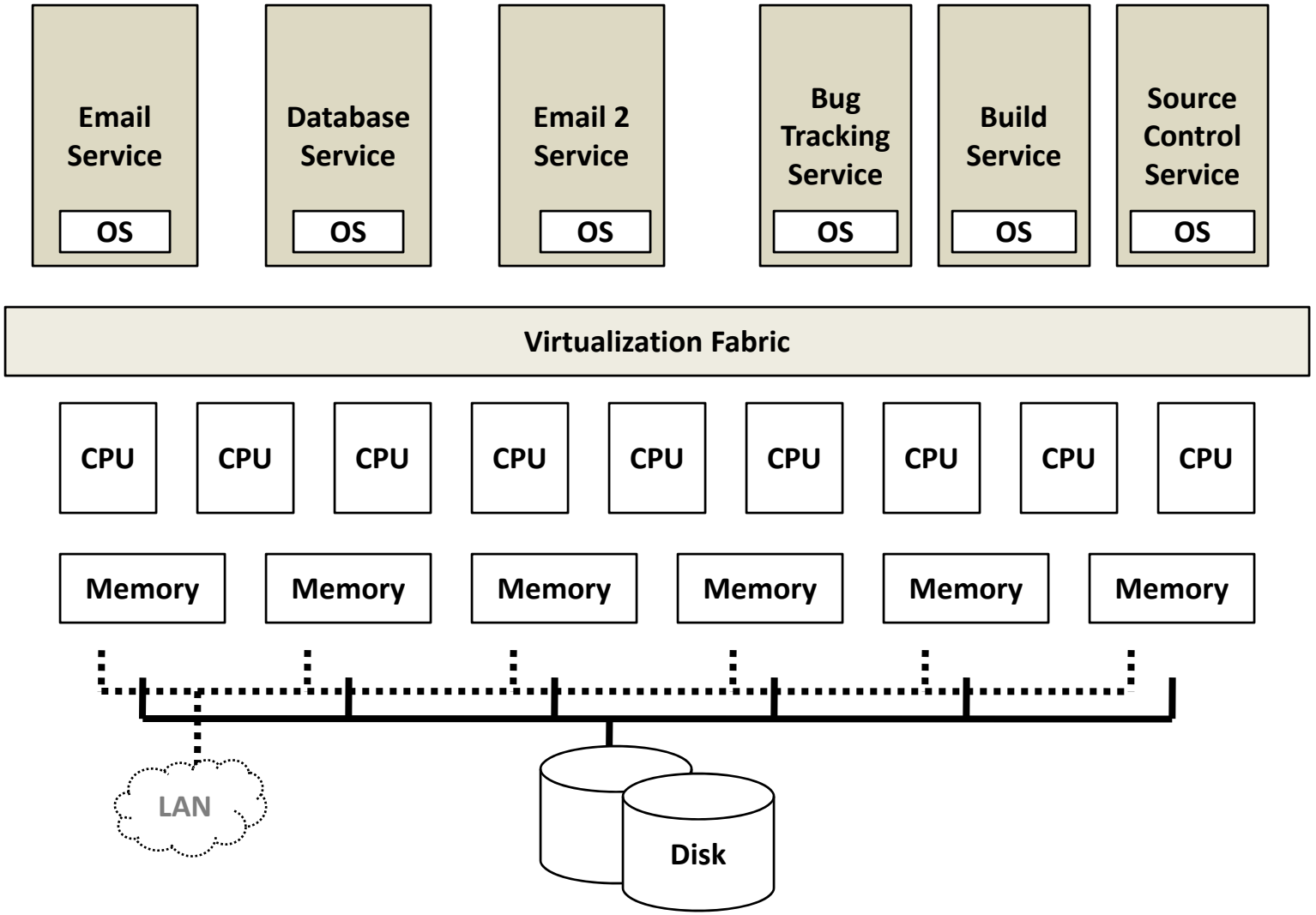
VMware VI SDK Example

```
# login
Util::connect();

# get VirtualMachineviews for all powered on VM's
my $vm_views= Vim::find_entity_views(view_type=> 'VirtualMachine',
                filter => { 'runtime.powerState' => 'poweredOn' });

# snapshot each VM
foreach(@$vm_views) {
   $_->CreateSnapshot(name=> 'snapshot sample',
                description => 'Snapshot created from workshop sample',
                memory => 0,
                quiesce => 0);
   print "Snapshot complete for VM: " . $_->name . "\n";
}

# logout
Util::disconnect();
```

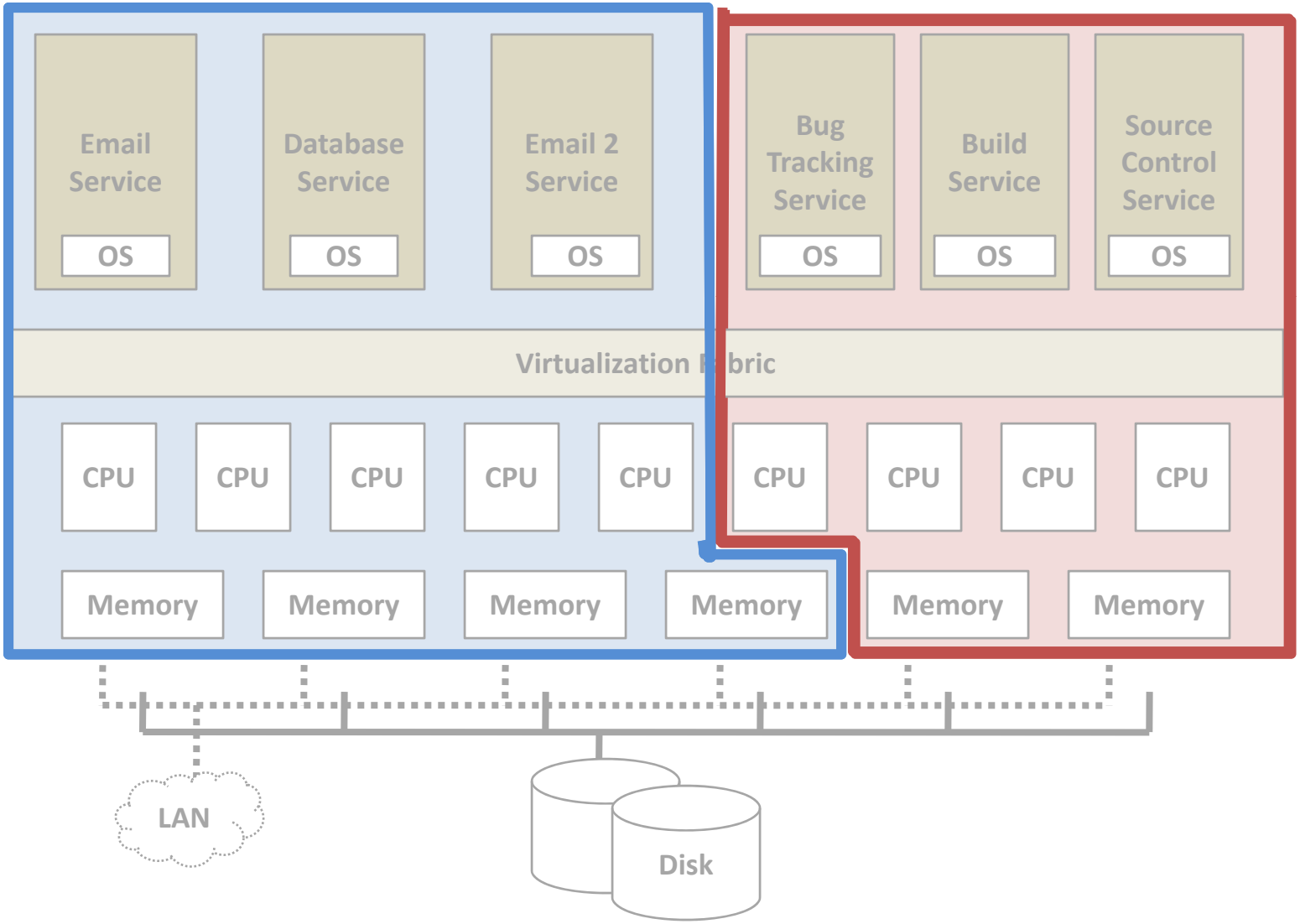
Multi-Host VM Management Architecture



Virtualization Fabric



Resource Pools



Datacenter Enumeration Example

```
my $datacenter = Opts::get_option('datacenter');
my $datacenter_view = Vim::find_entity_view(view_type => 'Datacenter',
                                    filter => { name => $datacenter });

if (!$datacenter_view) {
   die "Datacenter '" . $datacenter . "' not found\n";
}

# get all hosts under this datacenter
my $host_views = Vim::find_entity_views(view_type => 'HostSystem',
                                    begin_entity => $datacenter_view);

# print hosts
my $counter = 1;
print "Hosts found:\n";

foreach (@$host_views) {
   print "$counter: " . $_->name . "\n";
   $counter++;
}
```

Datacenter Enumeration Example (cont.)

```
# print vm's
$counter = 1;
print "\nVM's found:\n";

# get all VM's under this datacenter
my $vm_views = Vim::find_entity_views(view_type => 'VirtualMachine',
                                      begin_entity => $datacenter_view);

foreach (@$vm_views) {
   print "$counter: " . $_->name . "\n";
   $counter++;
}

# disconnect from the server
Util::disconnect();
```

Outline

- Management Overview
- **Live Migration**
- Deterministic Execution

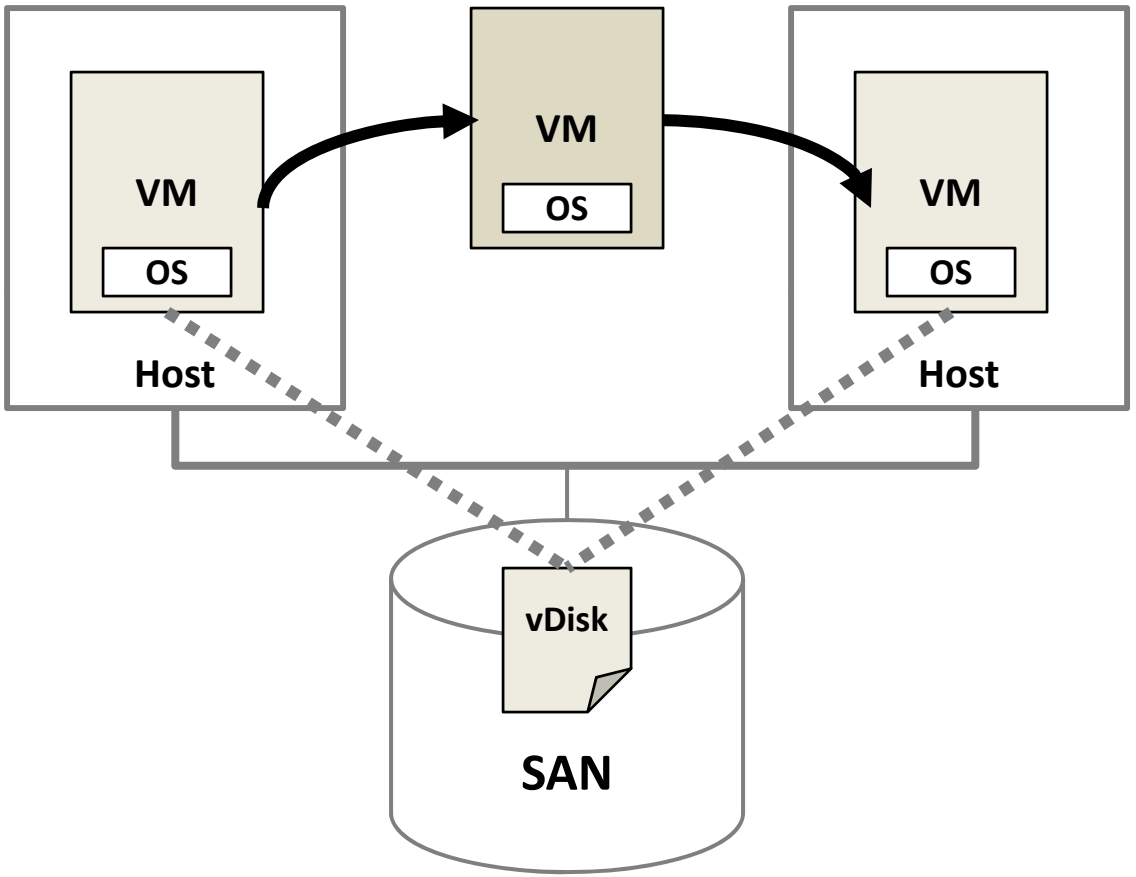
Naive Live Migration

- **Pause execution**
- **Take snapshot, record**
  - Registers
  - Memory
  - Device State, Disk
- **Move snapshot**
  - Memory
  - Disk
- **Restore snapshot**
- **Restart execution**

Naive Live Migration Speed

- **Size**
  - Memory == 4 GBs
  - Disk == 100 GBs
- **Network**
  - 1 Gb/s Dedicated Network
- **Downtime**
  - (104 Gb) ÷ 1 Gb/s ≈ **15 minutes**

Live Migration - Storage Architecture



Live Migration Requires Shared Storage

Live Migration - Memory

- **Iterative Approach**
  - Mark all memory invalid
  - Run VM on source while copying page to destination
  - VM will "fault-in" pages
    - These pages will need to be recopied
  - Once all memory has been copied

    If remaining memory is small enough

        Snapshot and copy remaining VM

    else

        GOTO "Mark"

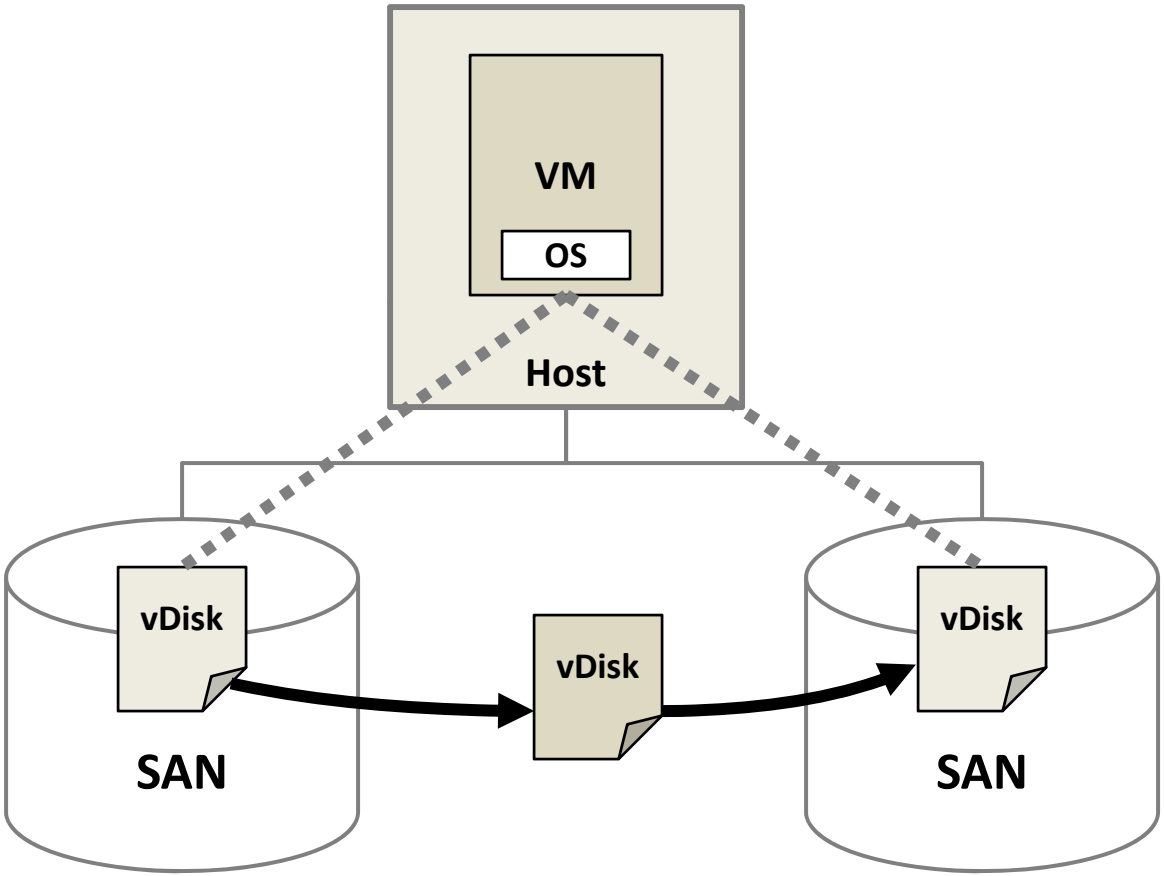
Live Migration – Memory

- **Hopefully converges**
  - Based on memory footprint
  - If remaining pages remain too large
    - Tolerate longer downtime
    - Don't move VM

Migration Management

- **CPU matching**
- **ARP network**
- **Failure during copy**

Storage VMotion



Outline

- Management Overview
- Live Migration
- **Deterministic Execution**

Deterministic Execution

- **Deterministic execution**
  - Same results each time
  - Execution path is identical
- **Lock-Step**
  - Two or more runs
  - Follow same execution path
  - Stay close in execution timing
    - Synchronized

Sources Non-Determinism

- **Initial State**
  - Memory
  - Disk
- **Externally supplied data**
- **Interrupts**
  - Timer
  - I/O
- **Multiple-processor memory interleaving**
- **Exceptions?**

Determinism in Single Threaded Process

- **Execution must take the same path on control flow**
  - Control flow is dependent on machine state
  - Started in same state
  - Run deterministically until this point
  - Control flow will be deterministic
- **So... Initial state must be the same**
  - Either start from scratch or checkpoint
- **Stop at precise point**
  - Count instructions

How to Count Instructions

- **Binary Translation**
  - Instrument code to count each basic block
  - Counter per basic block
  - Multiply by block length
  - SLOW!
- **Hardware Performance Counters**
  - Count instruction, branches, etc
  - Need to interrupt at certain count
  - Counters must be precise
  - Often not available, complete, or precise

Determinism with System Calls

- **System Calls are a form a External Data**
- **Some syscalls can be handled with Inital Data**
  - Time, PID, etc can be checkpointed
  - Need to intercept calls are return checkpointed values
  - Files can be copied so process sees initial state
- **Some syscalls may not be allowed**
  - Ex: sockets are a form of multi-threaded execution
- **What other factors can influence execution?**

Determinism in System Virtualization

- **I/O Data must be same**
  - General approach
    - Record all data input into virtual machine in master run
    - Replay data
  - DMA must be handled specially
    - Generally made atomic
- **Interrupts must occur at precise points**
  - I/O interrupts must be based on master
  - Record instruction count for all interrupts
  - Replay interrupt at precise cycle counts

Lock Step and Fault Tolerance

- **Two VMs start off same initial state**
  - One is master
  - Other is slave
- **Master runs as normal**
  - All interrupt timings, data inputs are shipped to slave
  - Slave replays execution upto next event
- **When master dies**
  - Slave can take over without any downtime
  - Transactions continue
  - Just need to worry about network routing

Multi-Threaded Determinism

- **Messages already handled interrupt delivery**
- **Memory is the Problem**

```
/* process 1 */

count = 0;
flag = 0;

while (!flag) {
   count++;
}

printf("%d\n", count);
```

```
/* process 2 */

while (!count) {
   ;
}

flag = 1;
```

Possible Solution

- **Manage all memory sharing**
  - Writeable pages are only given exclusively to one process
  - When other process touches page
    - Take fault
    - Steal page from owning process
    - Record time
    - Give page to faulting process
  - On replay force page steal at given time
    - Even if other process doesn't request it yet