Imperial College London

Operating Systems

Virtual Machines

Imperial College London

OS Processes

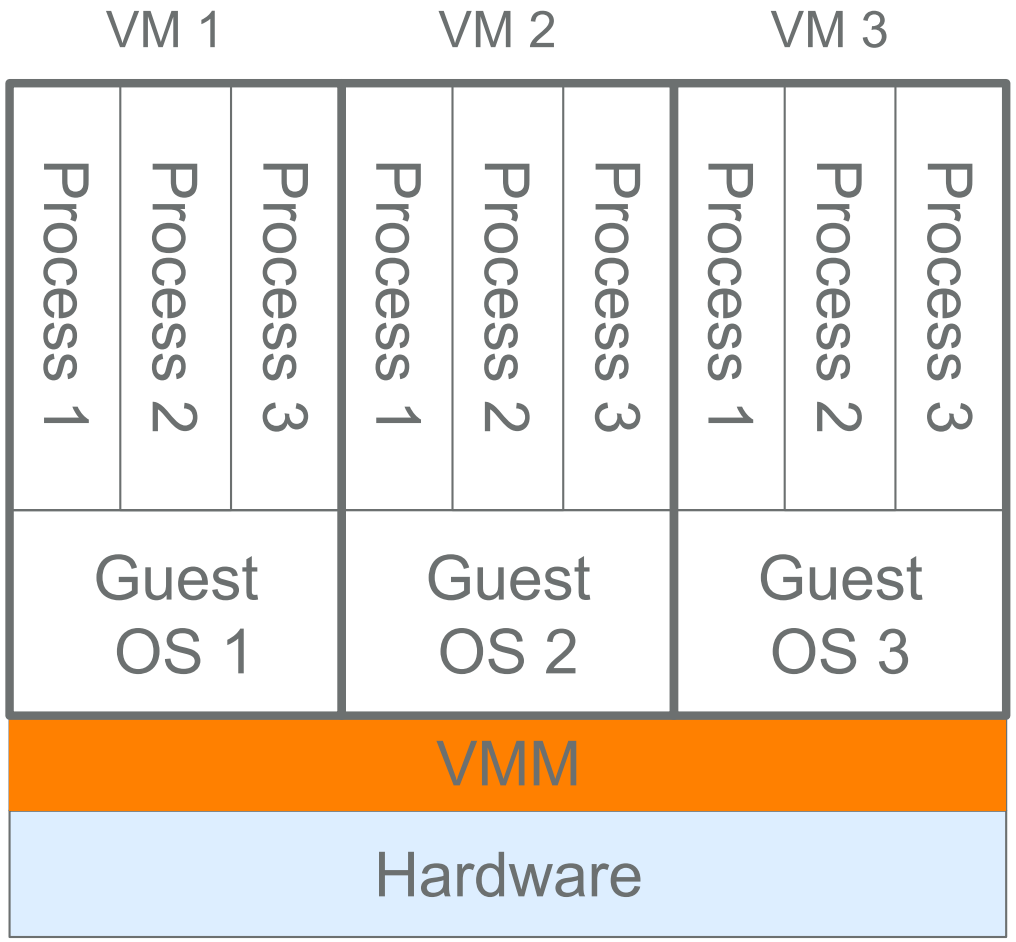
- What is the abstraction provided by the OS to a process?
  - CPU: non-privileged instructions and registers
  - Virtual memory
  - File system
  - System calls for I/O etc.
  - Signals
  - **A subset of underlying machine instructions**
- What if the abstraction looked just like *hardware*?
  - CPU: all instructions and registers
  - Both physical and virtual memory
  - Raw disk access
  - Full access to I/O devices
  - Interrupts

Virtual Machines

- Each process could run a **full OS**
  - Better security, support for legacy software, simplified software management, etc.
- Each OS given its own virtual machine
- Use **Virtual Machine Monitor (VMM)** to partition hardware among different OSs
  - CPU, memory, storage, etc.

Imperial College London

VMM and Guest OSs



- **VMM**: a.k.a. **hypervisor**
- **Guest OS**: OS running inside VM

History of Virtualization

- VMs first proposed in the 70s by IBM (VM/370)
  - Only a few expensive machines, important to multiplex them!
- Idea died out in the 80s and 90s
  - PCs for everyone!
- Revived during past decade
  - Rosenblum et al., Stanford

Imperial College London

Benefits of Virtualization

- End-users running multiple OSs on the same machine (w/o rebooting)

Benefits of Virtualization

- Server consolidation
  - Dedicated machines for email server, web server, etc.
  - Most servers only need a fraction of machine resources
  - Run email server, web server, DB server as separate VMs
    - Fewer physical machines saves money
  - Change resource allocation of VMs on demand

Benefits of Virtualization (cont.)

- Legacy Software
  - Some software only runs in original OS
  - E.g., despite huge efforts from MS, Windows Vista, 7, 8 not truly backward compatible with XP
- Software development/testing
  - Test multiple OSs on the same physical machine
  - Easily test/debug errors that bring machine down

Benefits of Virtualization (cont.)

- Security
  - Better isolation
    - Vulnerable app does not compromise/take down whole system
  - Smaller trusted computing base
    - Only VMM runs in kernel mode
    - VMMs orders of magnitude fewer LOCs than OSs
  - Intrusion detection via introspection
    - Scan memory of guest OS to detect suspicious patterns

Benefits of Virtualization (cont.)

- Software management
  - Snapshots and roll-back
  - **Migration** to different physical hosts
    - Live migration for load balancing
  - Simplified system administration
  - **Virtual appliances**
    - Apps depend on specific OSs, library versions, compilers, etc.
    - Software distribution can be problematic
    - Construct VM with all required dependencies, distribute it!

Implementing VMM

- Intercept all instructions and simulate their execution on actual hardware:
  - Simulate each instruction on virtual machine state
    - E.g., represent memory as an array
  - Typical slowdown:
    - ~100x for CPU-bound processes
    - ~2x for I/O-bound processes

- Modern VMs:
  - Typical slowdown:
    - ~5% for CPU-bound processes
    - ~30% for I/O-bound processes

Imperial College London

Implementing VMM

- Insight:
  - Most CPU instructions can be executed directly on actual CPU
  - E.g., **`mov, add, sub, inc.`**
- Need to intercept only sensitive instructions
  - **Trap and emulate** approach
- Example: **`cli/sli`**

Guest OS k:

`cli` —trap→

```
1) IF[k] = 0;
   (Remember Guest OS k
    disabled interrupts)
2) Stop delivering
   interrupts to Guest OS k
```

Obstacles to Virtualizations

- On some architectures, some sensitive instructions don't trap!
- Some instructions behave differently in user and kernel mode
  - `popf` on i386: used to replace flags, including `IF` bit
  - `IF` bit is simply not set in user mode!
- Visibility of privilege level
  - Read `%cs` (code segment selector) to get current privilege level
  - Guest OS should not find out it runs in a VM!

Virtualizable vs. Non-virtualizable CPUS

- A CPU is virtualizable if all sensitive instructions trap

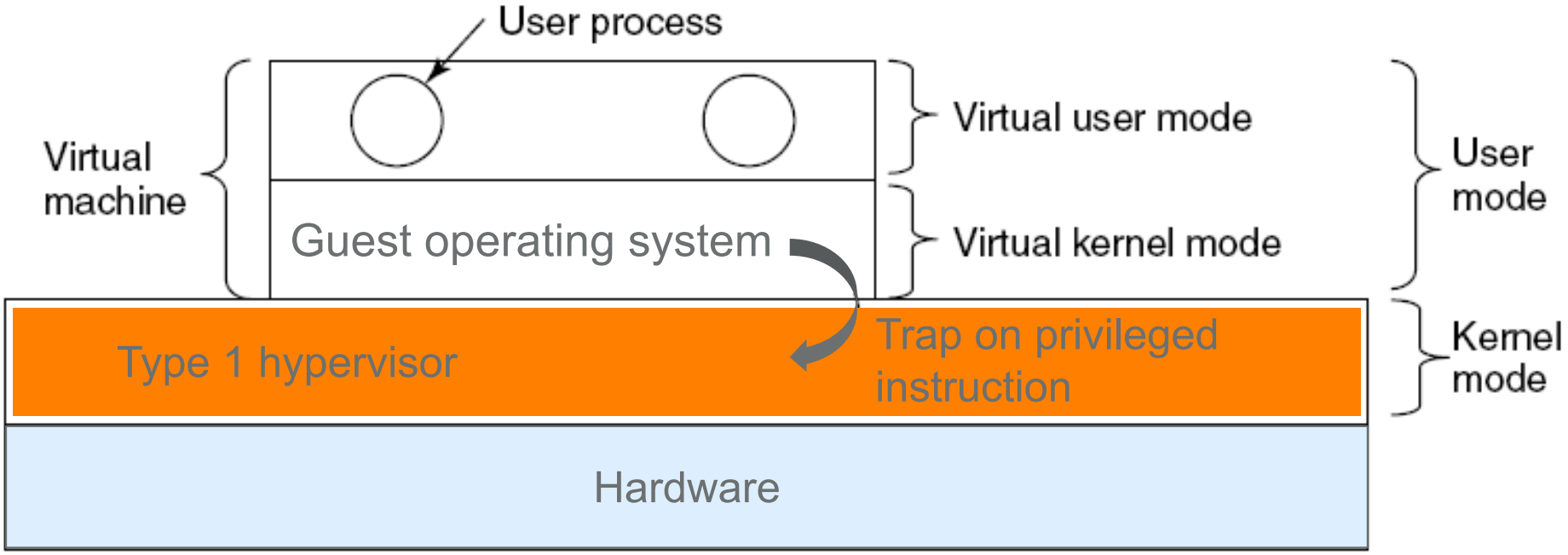
- x86 has become virtualizable in 2005
  - Intel's **Virtualization Technology (VT)**
  - AMD's **Secure Virtual Machine (SVM)**

Hypervisor Types

- Type 1 Hypervisor
  - VMM runs on **bare metal**
  - Better performance
- Type 2 Virtualization
  - VMM runs inside **host OS**

Type 1 Hypervisor

- Type 1 Hypervisor runs on bare metal
  - Need H/W support to trap all sensitive instructions (e.g. I/O, change MMU settings, determine kernel/user mode)



Type 1 Hypervisor Operation

1. Guest OS executes sensitive `out` instruction:

```
mov dx, os_ioport
out dx, al
```

2. CPU traps sensitive instruction

3. Hypervisor checks instructions
   - e.g., verifies I/O port allocated to guest OS

4. Hypervisor executes `out` instruction on real port:

```
mov dx, real_ioport
out dx, al
```

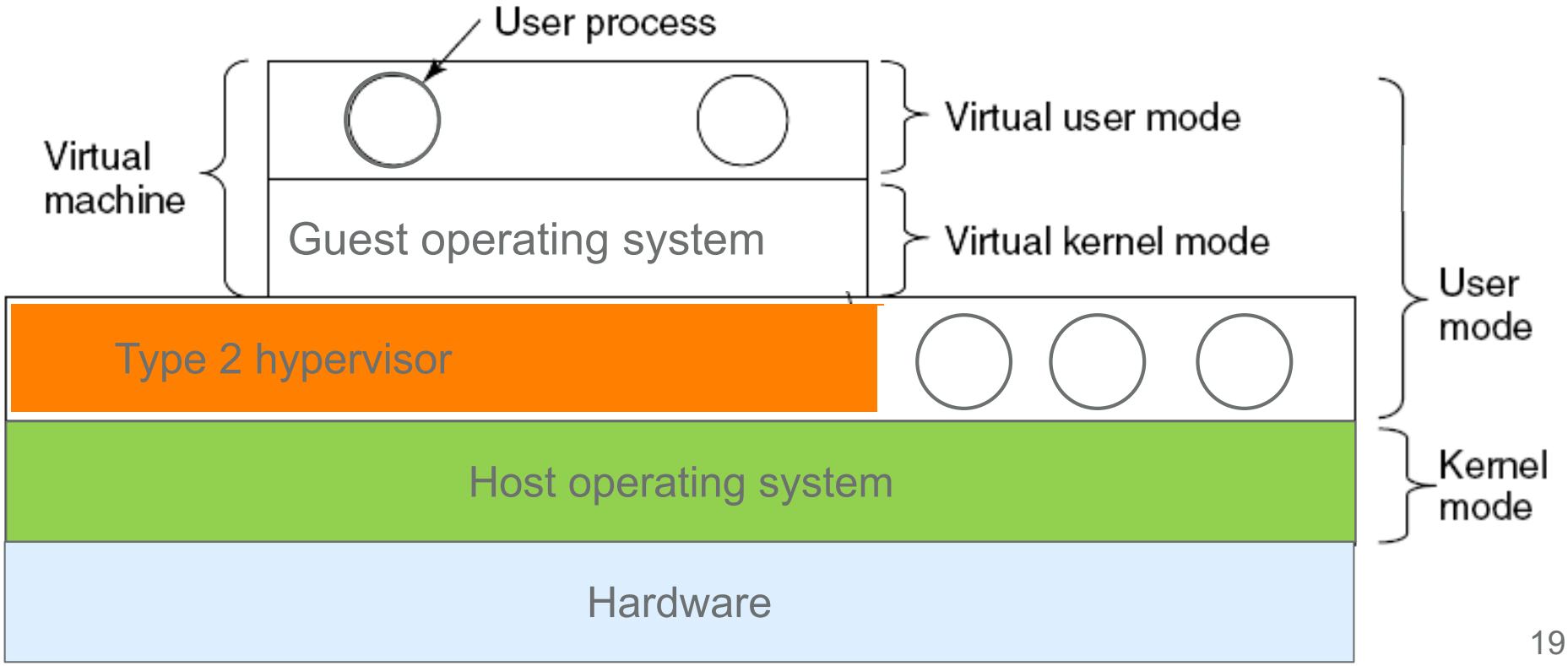
5. Control returns to guest OS

Question

- Consider a Type 1 hypervisor that can support up to n virtual machines at the same time. PCs usually have a maximum of four disk primary partitions. Can n be larger than 4? If so, where can the data be stored?

Type 2 Hypervisor

- Run hypervisor inside host OS
  - Great for end-users: easy to install (like a regular app!)
  - Can use I/O drivers from host OS
  - Can use services (e.g., scheduling) from host OS



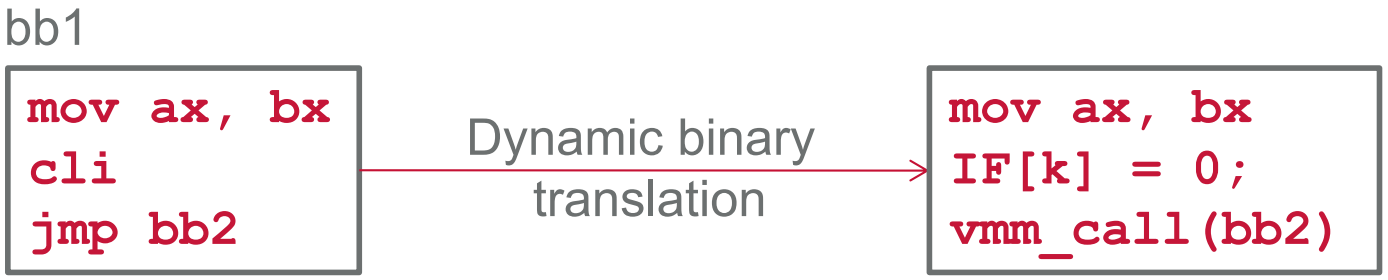
Binary Translation

- Frequent traps can incur significant overhead
  - Must clear TLBs, caches, branch predictions tables, ...
- Optimization idea:
  - Eliminate some of the traps via dynamic **binary translation**
  - Instead of emulating code (slow!), dynamically translate it so that it can run natively
- Introduced by the VMWare Workstation (Type 2 hypervisor)
  - Main goal was to handle unvirtualizable architectures
  - But can also be used by Type 1 hypervisors to improve performance

Binary Translation

Instead of emulating code (slow!), dynamically translate it so that it can run natively:

1. Scan each **basic block** just before it executes
2. If it contains privileged instructions, replace them with hypervisor calls
3. Replace last instruction with hypervisor call
4. Execute basic block natively



Binary Translation Optimizations

- Cache translated basic blocks for fast execution!
- Once successor blocks have been translated, replace final hypervisor call with direct jmp

- No need to translate user code

Question

- VMWare does binary translation one basic block at a time, then it executes the block and starts translating the next one. Could it translate the entire program in advance and then execute it? If so, what are the advantages and disadvantages of each technique?

Question

- Does binary translation slow down user-level function calls?

Paravirtualization

- Change OS source code to replace sensitive instructions with hypervisor calls

- Two benefits:
  - Can handle unvirtualizable architectures
  - Can achieve better performance

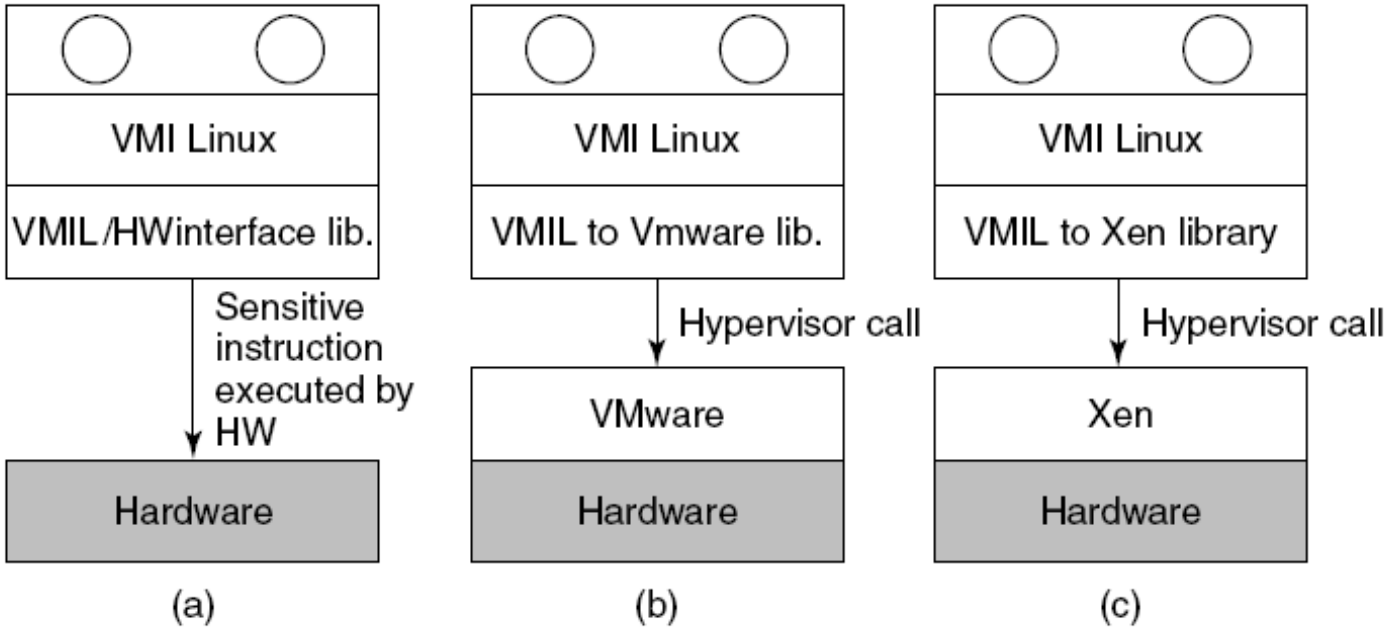
Disadvantages of Paravirtualization

- Need to make changes to OS source code
  - Need access!

- How can a modified OS still be run on native hardware?
- What about using multiple different hypervisors?

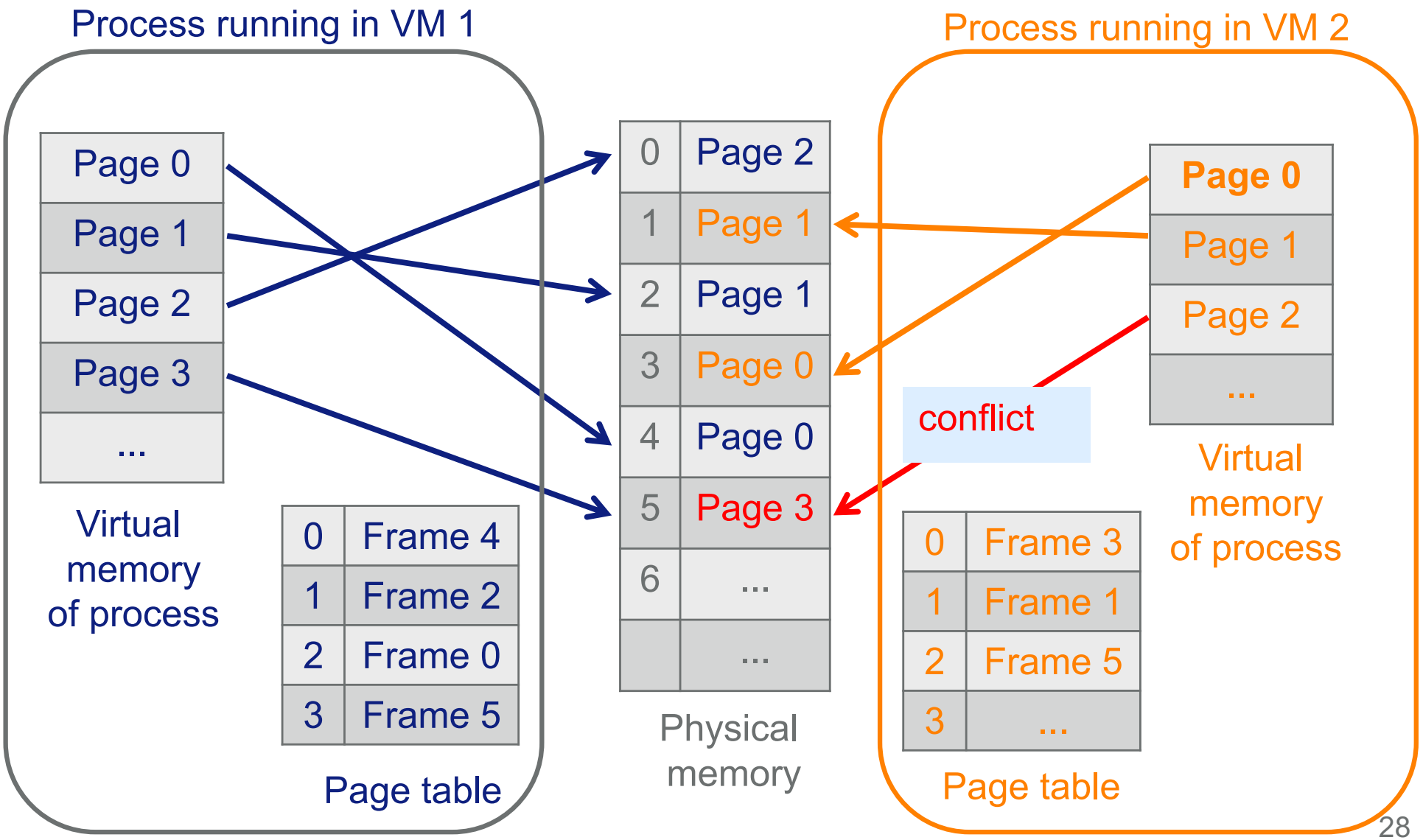
Virtual Machine Interface (VMI)

- Need to agree on a hypervisor API!
- Virtual Machine Interface (VMI)
  - Interfaces with both hardware and hypervisors



Imperial College London

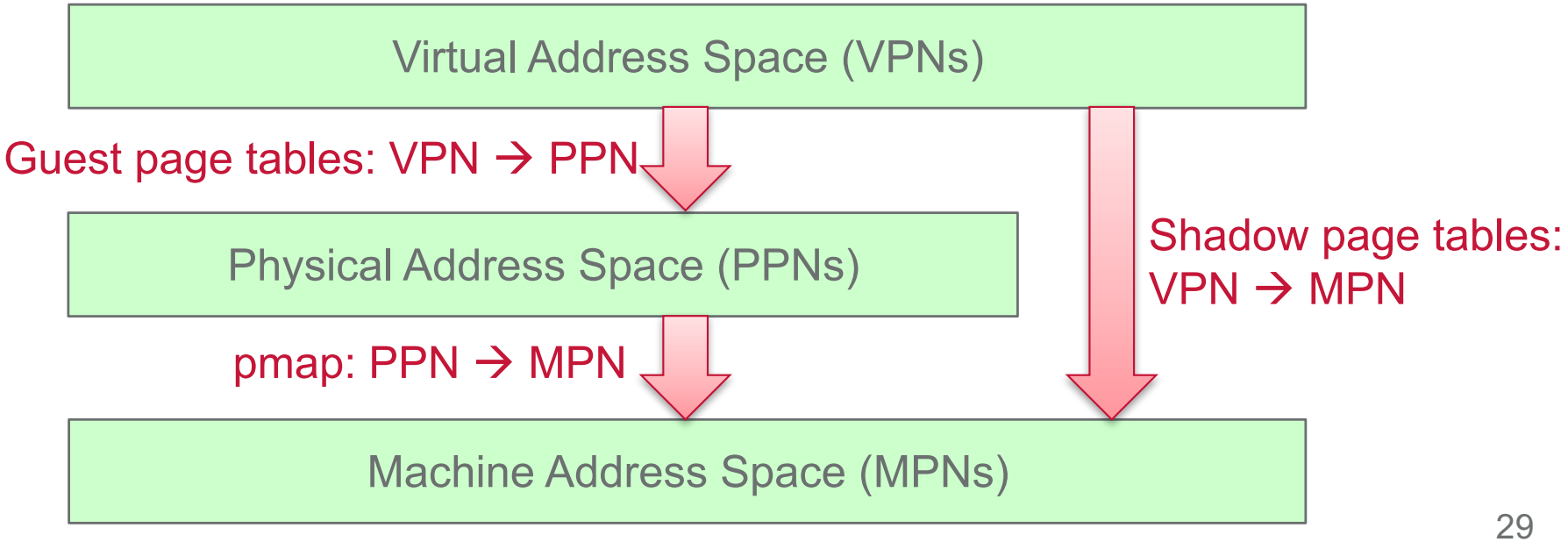
Memory Virtualization



Imperial College London

Memory Virtualization

- Add another level of indirection!
- Keep a **physical map (pmap)** structure for each VM
  - Maps physical addresses to actual machine addresses
- Keep separate **shadow page tables**
  - Map virtual addresses to machine addresses



Imperial College London

Shadow Page Tables

Process running in VM

0	Frame 3
1	Frame 1
2	Frame 5
3	...

Guest page table:
VPN → PPN

Frame 1	M-Frame 1
Frame 3	M-Frame 3
Frame 5	**M-Frame 6**
...	...

pmap: PPN → MPN

0	M-Frame 3
1	M-Frame 1
2	**M-Frame 6**
3	...

Shadow page table:
VPN → MPN

0	Page 2
1	Page 1
2	Page 1
3	Page 0
4	Page 0
5	Page 3
6	**Page 2**
7	
	...

Machine memory

Why shadow PTs?

Imperial College London

Shadow Page Tables

- Hardware MMU uses shadow PTs
- Hardware TLB maps VPN → MPN

- What happens on TLB miss?
  - MMU searches for the mapping in shadow PT
  - What happens if the mapping found?
    - TLB updated, instruction restarted
  - What happens if the mapping not found?
    - Page fault needs to be handled by VMM
    - VMM tries to find the VPN → PPN mapping in the guest OS PT
      - What happens if not found?
        - **True page fault**, forwarded to guest OS
      - What happens if found?
        - **Hidden page fault** (guest OS not aware and does nothing)
      - VMM updates pmap and shadow PT

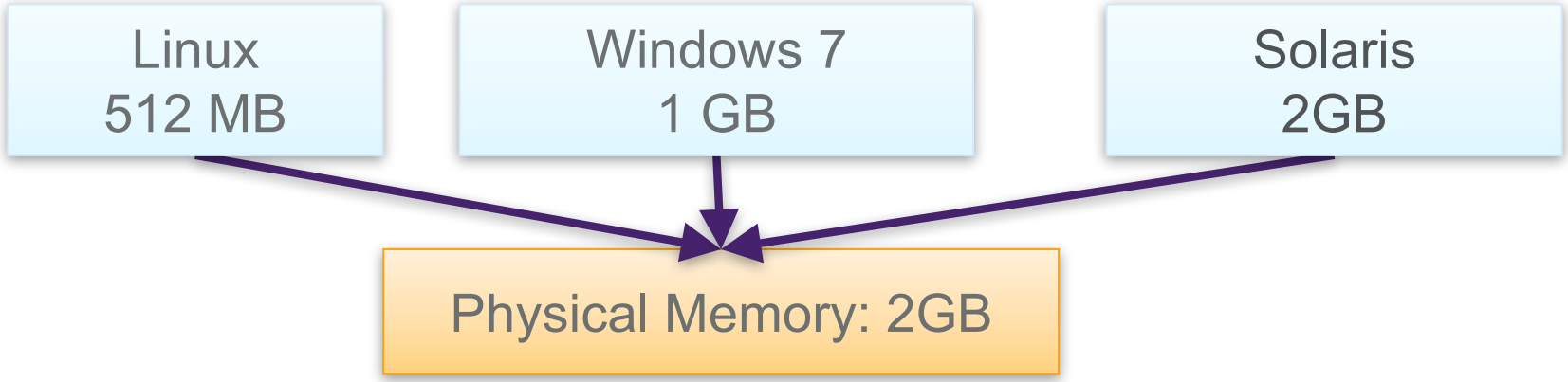
Keeping Guest and Shadow PTs in Sync

- VMM must keep guest and shadow PTs in sync
  - Trap into VMM when hardware changes PTs
  - Mark PT as read-only
- New CPUs have hardware support
  - AMD's Nested Page Tables (NPT)
  - Intel's Extended Page Tables (EPT)
  - Hardware aware of guest PTs, does a second lookup on TLB miss
  - No shadow PTs required

Hardware Support for Memory Virtualization

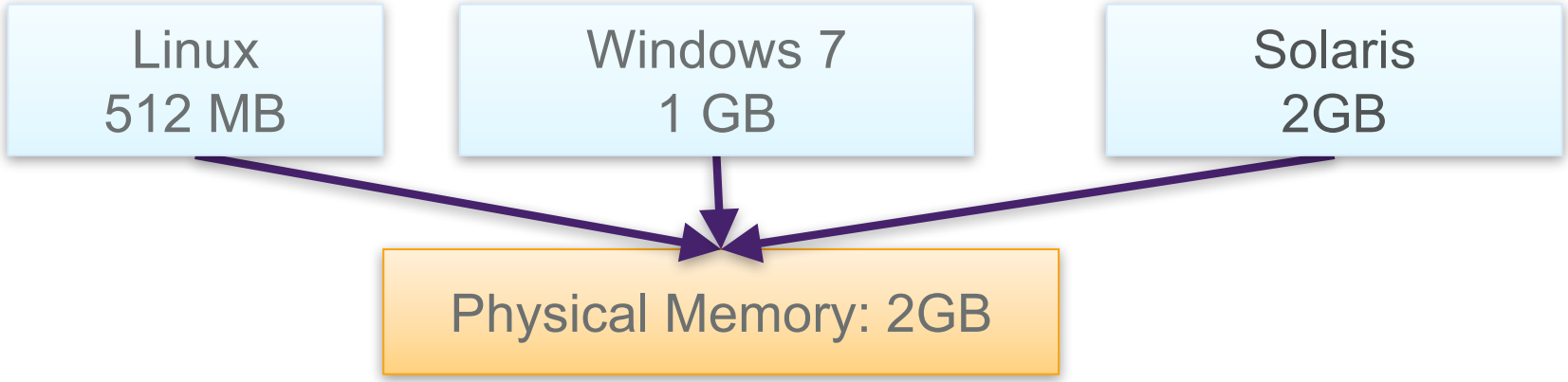
- What happens on TLB miss?
  - MMU searches for the mapping in guest PT
  - What happens if the mapping not found?
    - True page fault, forwarded to guest OS by VMM
  - What happens if the mapping found?
    - MMU searches for a mapping in pmap
    - What happens if found?
      - TLB updated, instruction restarted
    - What happens if not found?
      - Hidden page fault, VMM handles it

VM Memory Management



- How does a hypervisor reclaim memory from a VM?
- Swap out entire VM?
  - Does not provide much flexibility
- Use paging (on top of OS paging)?
  - What pages should be reclaimed from a VM?

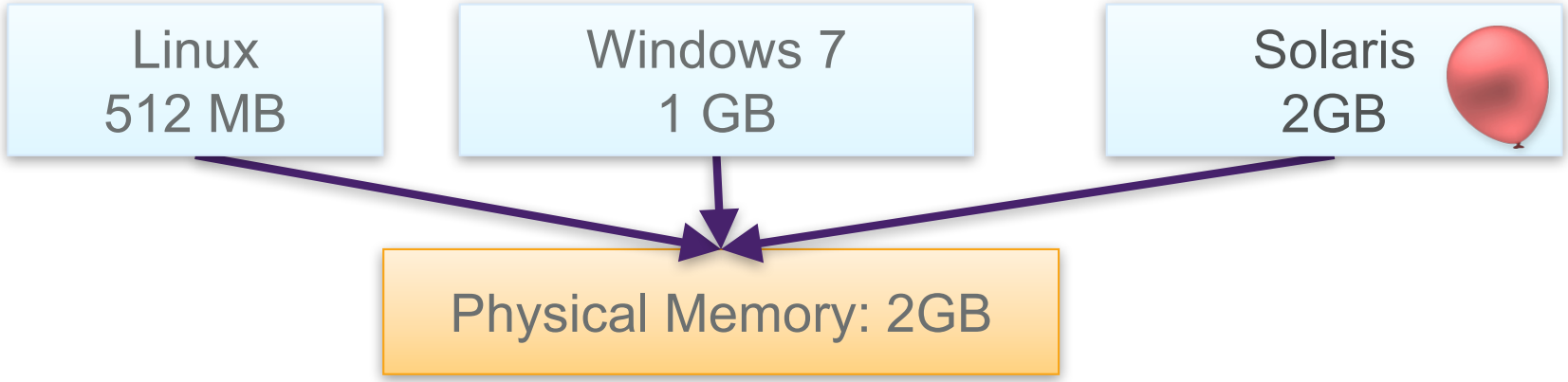
VM Memory Management



- Hypervisor doesn't know which pages are being used
- Guest OS has better information about active pages
- **Double paging problem**: pages being swapped out twice!
  - VMM under memory pressure, selects page P to be paged out
  - Guest OS under memory pressure, also selects P to be paged out to virtual disk
  - Page P will be brought back from disk, only to be written out to virtual disk!

Imperial College London

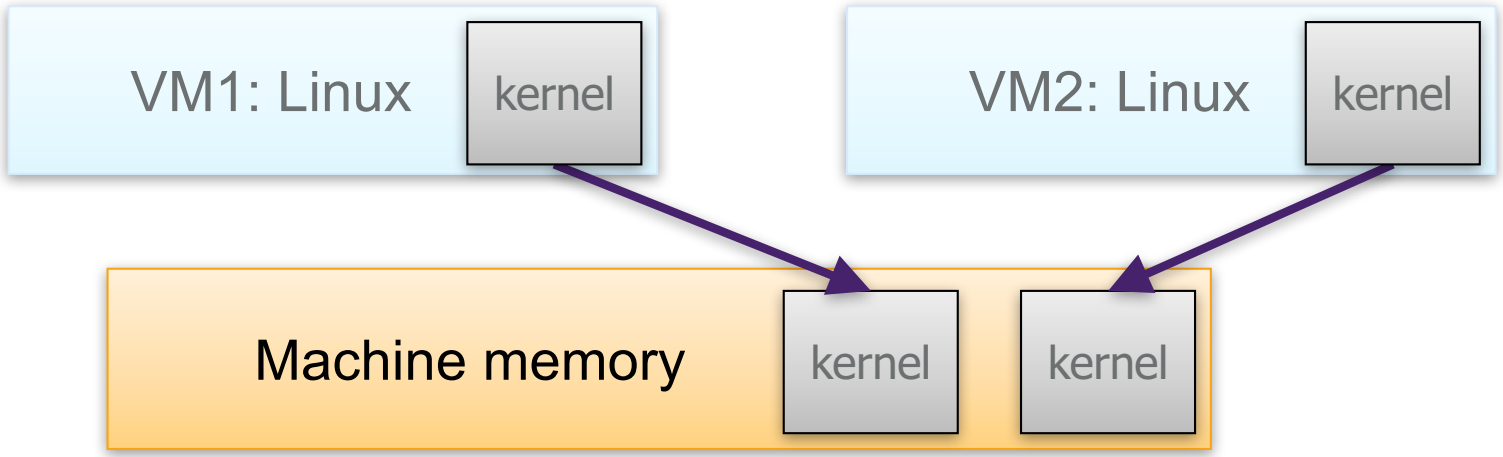
Memory Ballooning [Waldspurger 2002, ESX Server]



- Idea: force guest OS to swap out pages to disk
- Install **balloon driver** into the guest OS
  - Driver **inflates the balloon**: allocates pages (and "uses" them)
  - Guest OS swaps out pages to handle low memory condition
  - Hypervisor reclaims physical memory allocated to balloon
  - Driver **deflates the balloon** to give back memory to VM

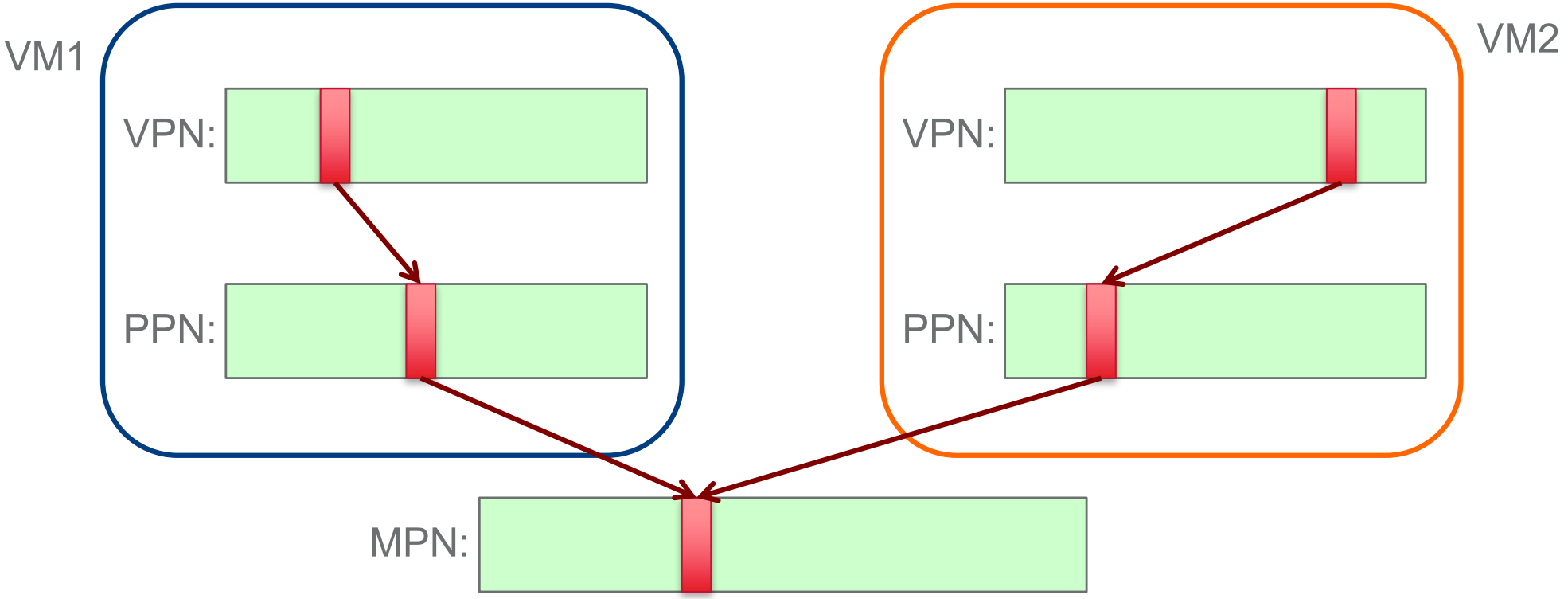
Sharing Memory

- Can we share memory between VMs?
  - Lots of opportunities with multiple VMs running the same OS
  - But hypervisor does not know about specific page usage



Sharing Memory

- Identify and identical pages and share them across VMs
- Copy-on-write (COW) semantics when modified



Content-Based Page Sharing [Waldspurger 2002]

- Comparing each possible pairs of pages expensive
  - Use hashing to identify pages that are potentially identical

1. Compute hash of page content
2. Index into a hash table to find existing page w/ same hash
3. Do full comparison to confirm pages are identical
4. Share!

- When and how is the page sharing algorithm run?
  - Different policies are possible
  - ESX Server implements the following effective policy:
    - Attempt to share a page before paging it out to disk
    - Scan guest pages randomly at a pre-defined rate

Summary

- Benefits of virtualization
  - Server consolidation
  - Better security
  - Simplified software management
  - Support for legacy applications
  - … and many more
- Virtualization techniques
  - Type 1 vs. Type 2 hypervisors
  - Binary translation
  - Paravirtualization
- Memory virtualization
  - Shadow page tables, hardware support
  - Memory ballooning, content-based page sharing

Summary

- Benefits of virtualization
  - Server consolidation
  - Better security
  - Simplified software management
  - Support for legacy applications
  - … and many more
- Virtualization techniques
  - Type 1 vs. Type 2 hypervisors
  - Binary translation
  - Paravirtualization
- Memory virtualization
  - Shadow page tables, hardware support
  - Content-based sharing