TA68

Performance Aspects of x86 Virtualization

Ole Agesen

Principal Engineer

VMware

Talk Outline

- Part 1. Problem statement
- Part 2. Instruction set virtualization techniques
- Part 3. Memory virtualization techniques
- Part 4. Practicalities
- Conclusions

Understanding Performance in a Complex Space

- **Life used to be simple (1999)**
  - All guests were 32 bit
  - All CPUs were 32 bit
  - All VMMs used binary translation (BT)
- **Now we have diversity (2007)**
  - 32 and 64 bit guests (and CPUs)
  - 64 bit architecture divergence (AMD/Intel)
  - CPUs with/without hardware assist
    - 1st generation hw assist: instruction set virtualization
    - 2nd generation hw assist: memory virtualization
  - Para-virtualized guests (not covered in this talk)

What Can You do with this Understanding?

- **Buying systems?**
  - I don't think my analysis will help you
- **Writing software?**
  - Can help you get better performance in virtual machine
- **Tuning and adjusting existing workloads?**
  - Ditto
- **Virtualizing workloads?**
  - Better understand what can be successfully virtualized today, and in the future as new hardware/OSes/workloads appear

Caveats

- **Performance numbers are very rounded**
  - *Indicative* of numbers one might measure, not actual numbers
    - Avoid “apples” vs. “oranges” (no one CPU can run all modes)
    - Ignoring MHz, pipelines, cache sizes
  - Big picture has longer life than details
    - Tomorrow’s CPUs
    - Tomorrow’s guests
  - Using micro-benchmarks that exaggerate properties
- **Discussing just CPU performance; ignoring I/O, resource mgmt, power consumption, etc.**
  - Any use of virtualization must consider all these factors

What's the First Thing You Do with a New Computer?

Benchmark it!

sum.c:

```
uint64_t i, s = 0;
for (i = 0; i < 1000000000; i++) {
  s = s + i;
}
printf("s = %ld, c = %ld\n",
       s, i * (i – 1) / 2);
```

```
phobos 100> cc –o sum sum.c
phobos 101> time sum
s = 499999999500000000, c = 499999999500000000
3.276u 0.003s 0:03.28 99.6% ...
```

...And if You get a New Virtual Computer?

- **Benchmark virtual against native**
  - Native performance generally is upper bound for virtual
    - Ideal performance of VM relative to native: 100%
    - Native(sum): 3.28s
    - Virtual(sum): 3.47s
    - Ratio: 95%
  - Probably happy with the new virtual computer!

What's the Next Thing You Do?

- **Tell your friend about the new computer!**
  - Victor to Bob: "my virtual computer runs at 95% of native!"
  - Bob to Victor: "my virtual computer runs at 25% of native – help!"

What's the Next Thing You Do?

- **Tell your friend about the new computer!**
  - Victor to Bob: "my virtual computer runs at 95% of native!"

sum.c:

```
uint64_t i, s = 0;
for (i = 0; i < 1000000000; i++) {
  s = s + i;
}
```

  - Bob to Victor: "my virtual computer runs at 25% of native – help!"

getppid.c:

```
for (int i = 0; i < 10000000; i++) {
  getppid();
}
```

Hoping to Understand It, We Swap Benchmarks

Percent of native performance:

	Bob	Victor
sum	95%	95%
getppid	25%	95%

...And Call Nat Who Also has a New Virtual Computer

and a couple of benchmarks that have him scratching his head...

forkwait.c:

```
for (int i = 0; i < 1000000; i++) {
  if (fork() == 0) return;
}
```

85%

memsweep.c:

```
#define S  (8192 * 4096)
volatile char large[S];
for (unsigned i = 0; i < 10 * S; i++) {
  large[(4096 * i + i) % S] = 1 + large[i % S];
}
```

40%

A Complex Picture Emerges

	Bob Trellis	Victor Thomson	Nat Petersen
sum	95%	95%	95%
getppid	25%	95%	95%
forkwait	15%	5%	85%
memsweep	85%	85%	40%

- No system handles all benchmarks well
- No two systems suffer same overheads!

A Complex Picture Emerges

	Bob Trellis	Victor Thomson	Nat Petersen
sum	95%	95%	95%
getppid	25%	95%	95%
forkwait	15%	5%	85%
memsweep	85%	85%	40%

- **3 different execution modes:**
  - Binary Translation
  - 1'st generation hardware assist for instructions (Intel VT-x, AMD-V)
  - 2'nd generation hardware assist for memory (AMD-V Nested Paging[1,2])

[1]Results generated on pre-release AMD 'Barcelona' processors. Results are unaudited and are intended to demonstrate relative NP/BT performance only.
[2] Intel has announced a similar feature in a future processor: EPT.

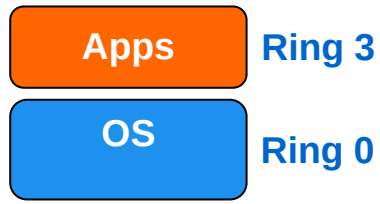
Talk Outline

- **Part 1. Problem statement**
- **Part 2. Instruction set virtualization techniques**
  - Background: classical trap-and-emulate
  - Software: binary translation (BT)
  - Hardware: Intel VT-x and AMD-V
  - Qualitative performance comparison
- **Part 3. Memory virtualization techniques**
- **Part 4. Practicalities**
- **Conclusions**

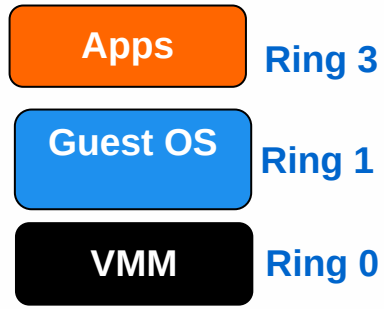
Classical Instruction Virtualization

Trap-and-emulate

- **Nonvirtualized ("native") system**
  - OS runs in privileged mode
  - OS "owns" the hardware
  - Application code has less privilege



- **Virtualized**
  - VMM most privileged (for isolation)
  - Classical "ring compression" or "de-privileging"
    - Run guest OS kernel in Ring 1
    - Privileged instructions trap; emulated by VMM
  - But: does not work for x86 (lack of traps)



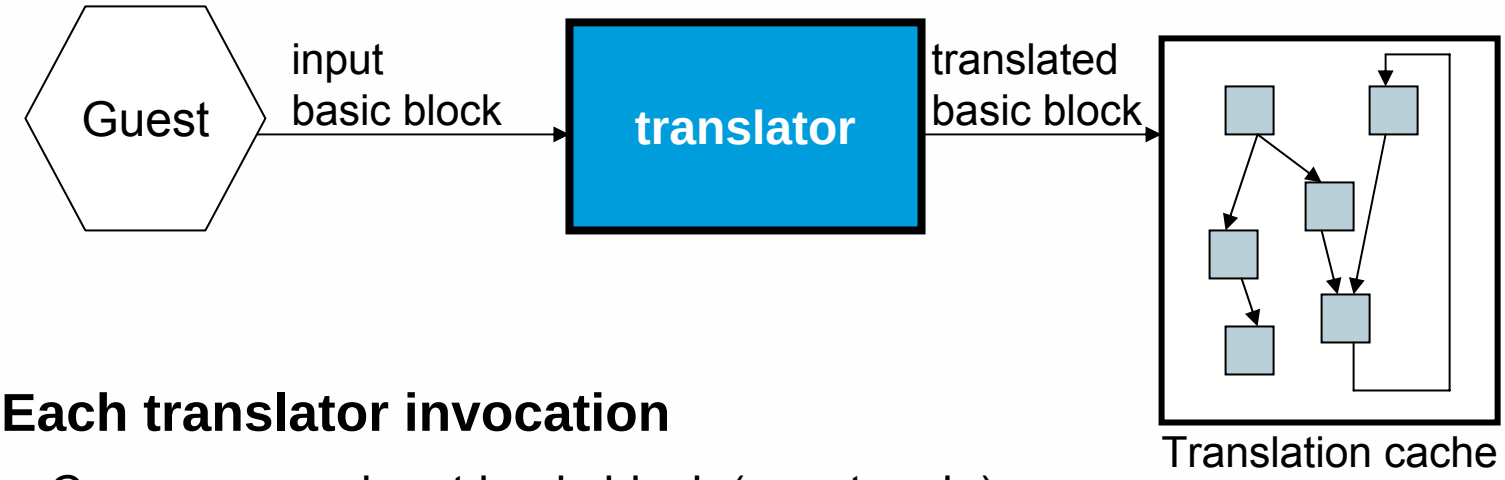
Classical VM performance

- **Native speed except for traps**
  - Overhead = trap frequency * average trap cost
- **Trap sources:**
  - Privileged instructions
  - Page table updates (to support memory virtualization)
  - Memory-mapped devices
- **Back-of-the-envelope numbers:**
  - Trap cost is high on deeply pipelined CPUs: ~1000 cycles
  - Trap frequency is high for “tough” workloads: 50 kHz or greater
  - Bottom line: substantial overhead

Binary Translation of Guest Code

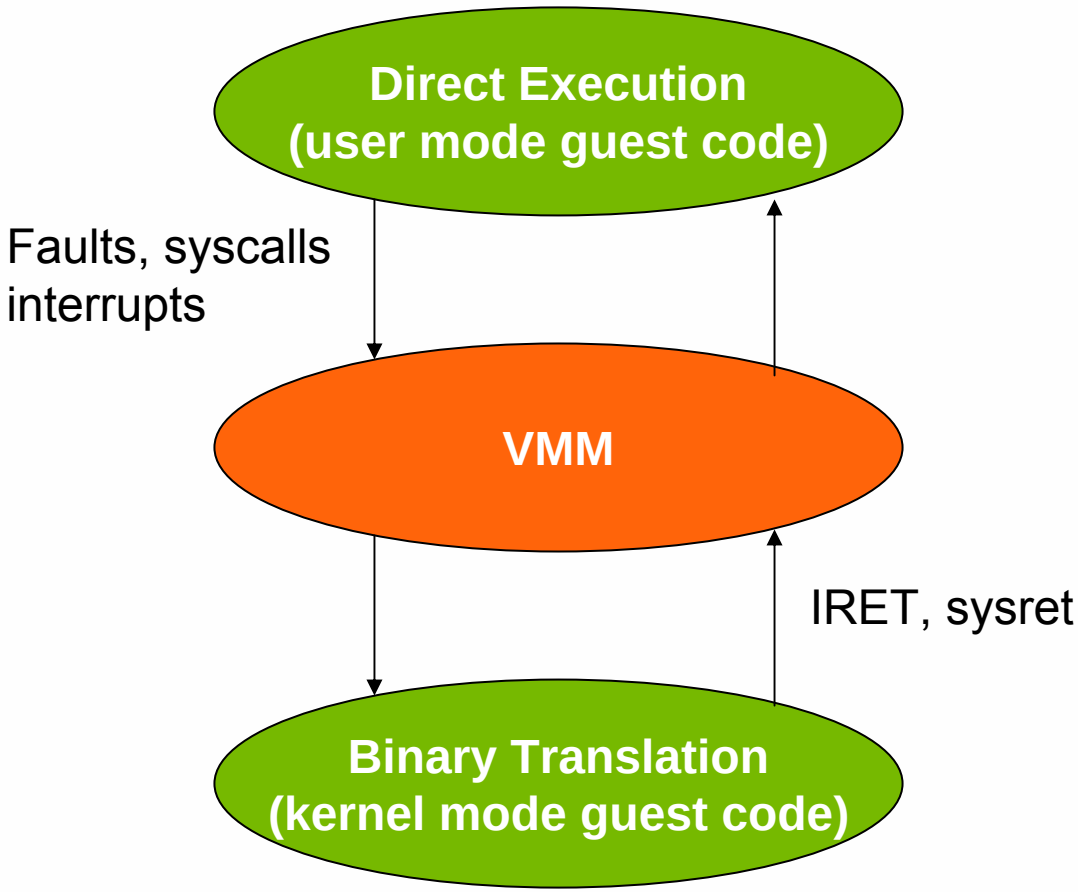
- **Translate guest kernel code**
- **Replace privileged instrs with safe “equivalent” instruction sequences**
- **No need for traps**
- **BT is an extremely powerful technology**
  - Permits *any* unmodified x86 OS to run in a VM
  - Can virtualize *any* instruction set

BT Mechanics



- **Each translator invocation**
  - Consume one input basic block (guest code)
  - Produce one output basic block
- **Store output in translation cache**
  - Future reuse
  - Amortize translation costs
  - Guest-transparent: no patching “in place”

Combining BT and Direct Execution

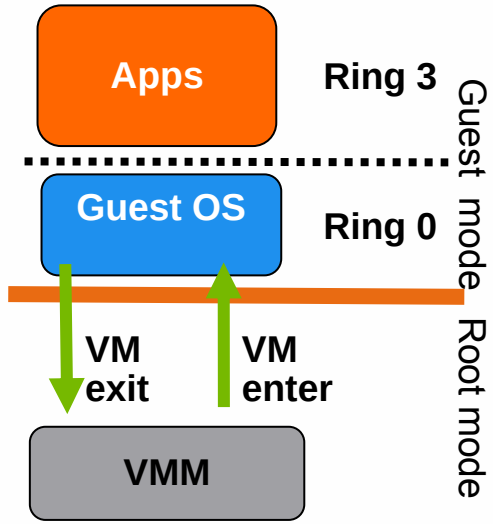


Performance of a BT-based VMM

- **Costs**
  - Running the translator
  - Path lengthening: output is sometimes longer than input
  - System call overheads: DE/BT transition
- **Benefits**
  - Avoid costly traps
  - Most instructions need no change (“identical” translation)
  - Adaptation: adjust translation in response to guest behavior
    - Online profile-guided optimization
  - User-mode code runs at full speed (“direct execution”)

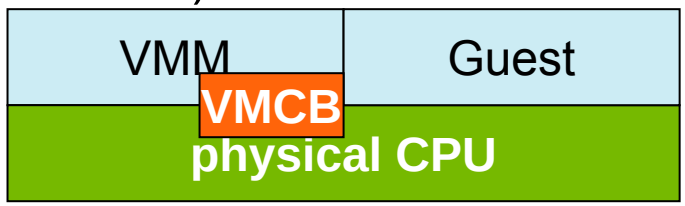
Intel VT-x / AMD-V: 1st Generation HW Support

- **Key feature: root vs. guest CPU mode**
  - VMM executes in root mode
  - Guest (OS, apps) execute in guest mode
- **VMM and Guest run as "co-routines"**
  - VM enter
  - Guest runs
  - A while later: VM exit
  - VMM runs
  - ...



How VMM Controls Guest Execution

- **Hardware-defined structure**
  - Intel: VMCS (virtual machine control structure)
  - AMD: VMCB (virtual machine control block)
- **VMCB/VMCS contains**
  - Guest state
  - Control bits that define conditions for exit
    - Exit on IN, OUT, CPUID, ...
    - Exit on write to control register CR3
    - Exit on page fault, pending interrupt, ...
  - VMM uses control bits to “confine” and observe guest



Performance of a VT-x/AMD-V Based VMM

- **VMM only intervenes to handle exits**
- **Same performance equation as classical trap-and-emulate:**
  - overhead = exit frequency * average exit cost
- **VMCB/VMCS can avoid simple exits (e.g., enable/disable interrupts), but many exits remain**
  - Page table updates
  - Context switches
  - In/out
  - Interrupts

Qualitative Comparison of BT and VT-x/AMD-V

- **BT loses on:**
  - system calls
  - translator overheads
  - path lengthening
  - indirect control flow
- **BT wins on:**
  - page table updates (adaptation)
  - memory-mapped I/O (adapt.)
  - IN/OUT instructions
  - no traps for priv. instructions
- **VT-x/AMD-V loses on:**
  - exits (costlier than “callouts”)
  - no adaptation (cannot elim. exits)
  - page table updates
  - memory-mapped I/O
  - IN/OUT instructions
- **VT-x/AMD-V wins on:**
  - system calls
  - almost all code runs “directly”

Qualitative Comparison of BT and VT-x/AMD-V

- **BT loses on:**
  - system calls
  - translator overheads
  - path lengthening
  - indirect control flow
- **BT wins on:**
  - page table updates (adaptation)
  - memory-mapped I/O (adapt.)
  - IN/OUT instructions
  - no traps for priv. instructions

Explains Bob's slow getppid

- **VT-x/AMD-V loses on:**
  - exits (costlier than
  - no adaptation (ca
  - page table updates
  - memory-mapped I/O
  - IN/OUT instructions
- **VT-x/AMD-V wins on:**
  - system calls
  - almost all code runs "directly"

Explains Victor's very slow forkwait

Qualitative Comparison of BT and VT-x/AMD-V

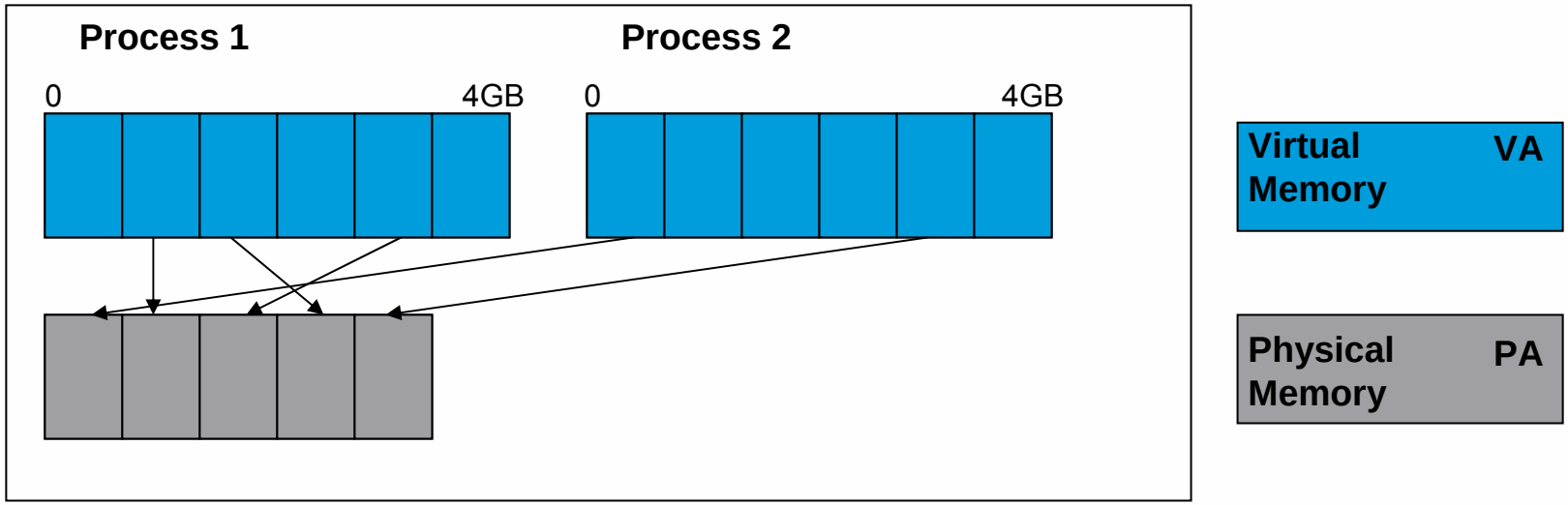
- **BT loses on:**
  - system calls
  - translator overheads
  - path lengthening
  - indirect control flow
- **BT wins on:**
  - page table updates (adaptation)
  - memory-mapped I/O (adapt.)
  - IN/OUT instructions
  - no traps for priv. instructions

- **VT-x/AMD-V loses on:**
  - exits (costlier than “callouts”)
  - no adaptation (cannot elim. exits)
  - page table updates
  - memory-mapped I/O
  - IN/OUT instructions
- **VT-x/AMD-V wins on:**
  - system calls
  - almost all code runs “directly”

Talk outline

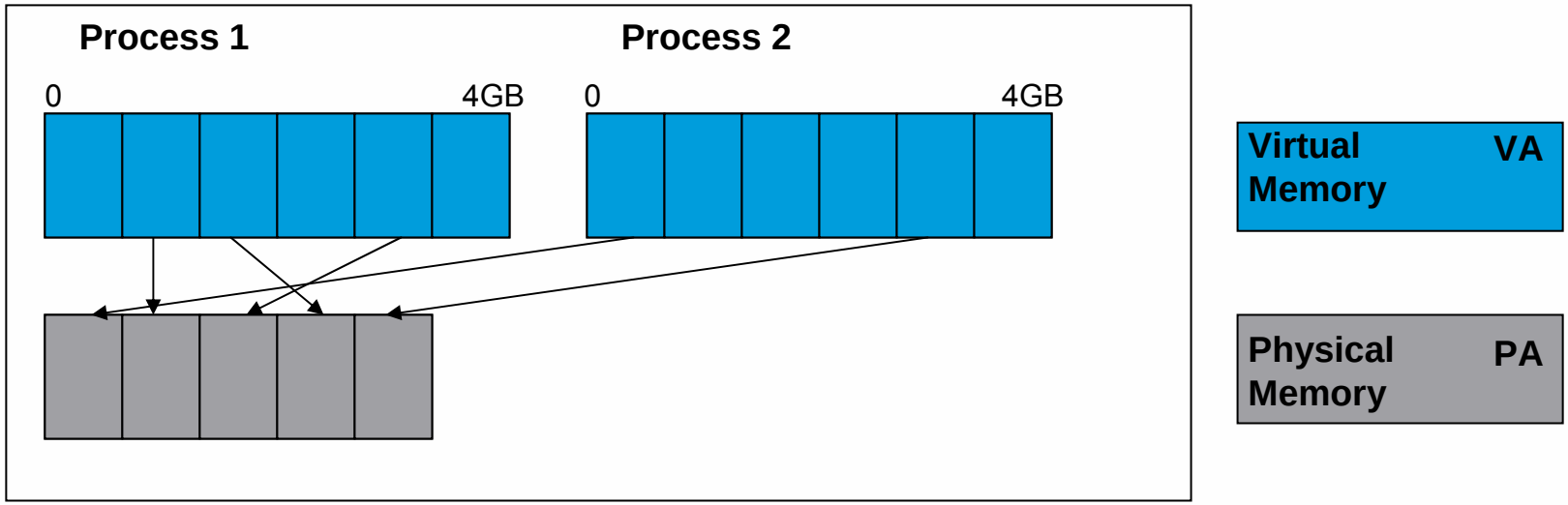
- **Part 1. Problem statement**
- **Part 2. Instruction set virtualization techniques**
- **Part 3. Memory virtualization techniques**
  - Software: shadow page tables
  - Hardware: nested page tables (NPT), extended page tables (EPT)
  - Qualitative performance comparison
- **Part 4. Practicalities**
- **Conclusions**

Virtual Memory

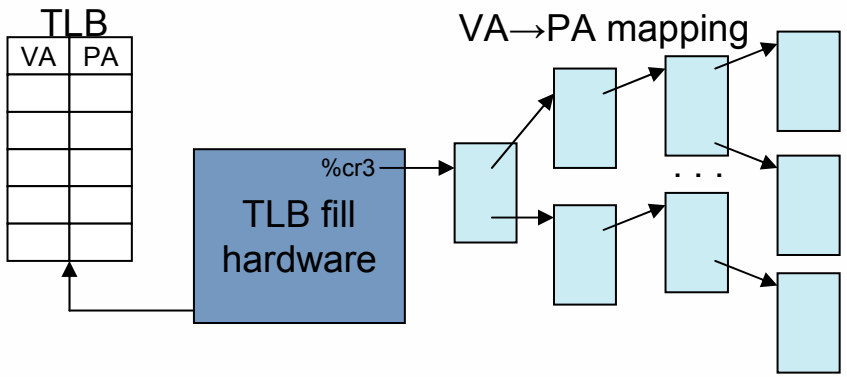


- **Applications see contiguous virtual address space, not physical memory**
- **OS defines VA -> PA mapping**
  - Usually at 4 KB granularity: a *page* at a time
  - Mappings are stored in page tables

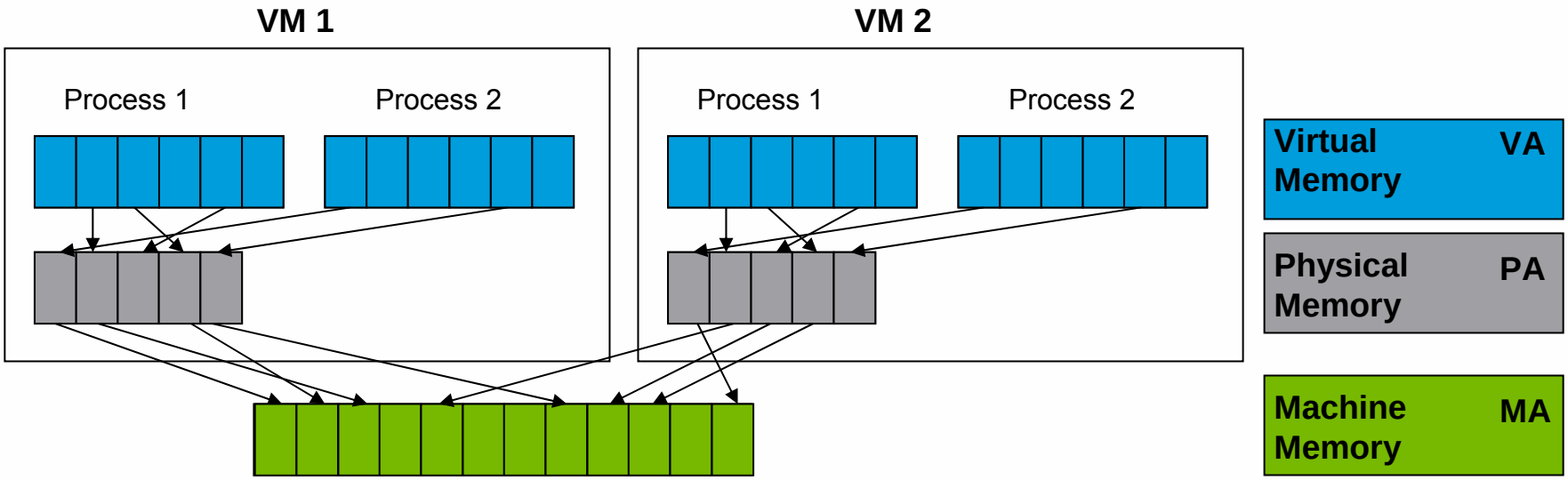
Virtual Memory



- **Applications see contiguous virtual address space, not physical memory**
- **OS defines VA -> PA mapping**
  - Usually at 4 KB granularity
  - Mappings are stored in page tables
- **HW memory management unit (MMU)**
  - Page table walker
  - TLB (translation look-aside buffer)



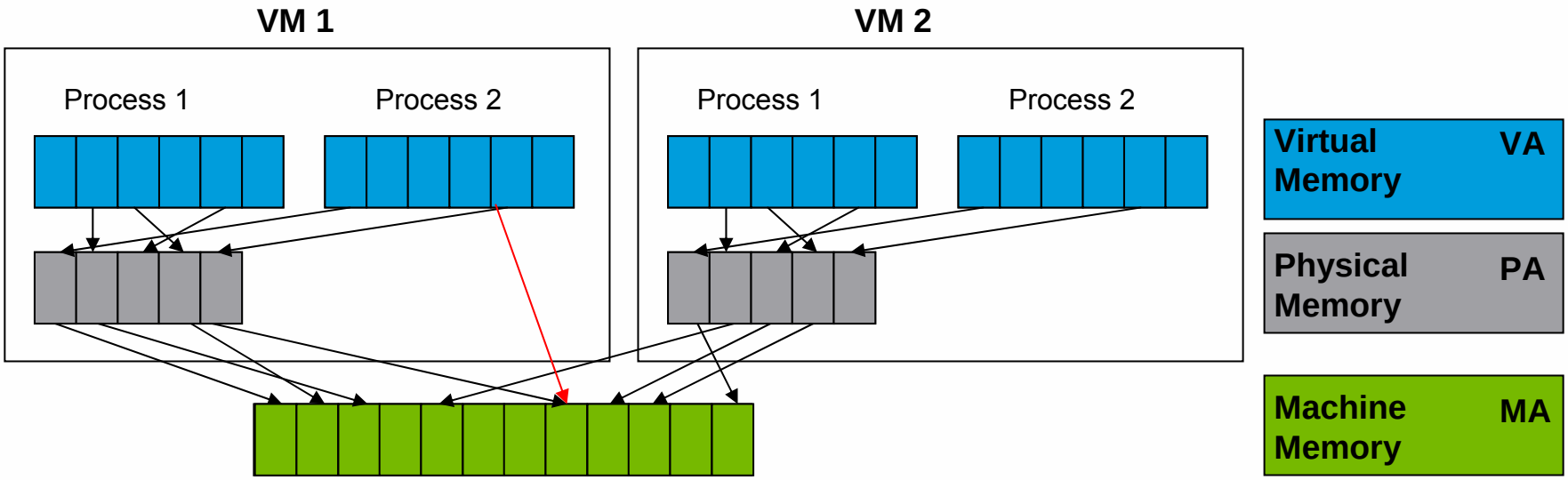
Virtualizing Virtual Memory



- **To run multiple VMs on a single system, another level of memory virtualization must be done**
  - Guest OS still controls virtual to physical mapping: VA -> PA
  - Guest OS has no direct access to machine memory (to enforce isolation)
- **VMM maps guest physical memory to actual machine memory: PA -> MA**

Virtualizing Virtual Memory

Shadow Page Tables



- **VMM builds "shadow page tables" to accelerate the mappings**
  - Shadow directly maps VA -> MA
  - Can avoid doing two levels of translation on every access
  - TLB caches VA->MA mapping
  - Leverage hardware walker for TLB fills (walking shadows)
  - When guest changes VA -> PA, the VMM updates shadow page tables

3-way Performance Trade-off in Shadow Page Tables

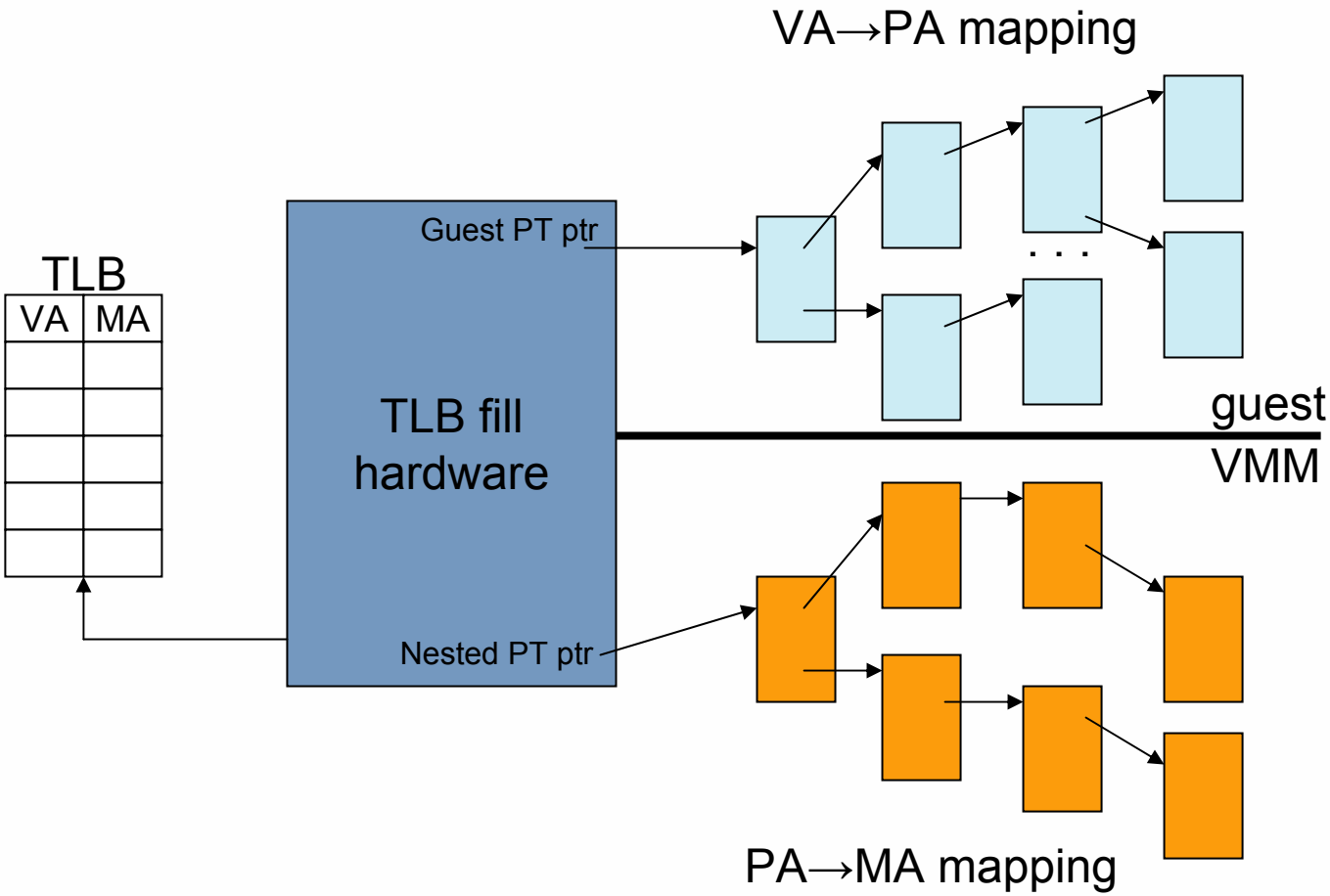
- **1. Trace costs**
  - VMM must intercept Guest writes to primary page tables
  - Propagate change into shadow page table (or invalidate)
- **2. Page fault costs**
  - VMM must intercept page faults
  - Validate shadow page table entry (hidden page fault), or forward fault to Guest (true page fault)
- **3. Context switch costs**
  - VMM must intercept CR3 writes
  - Activate new set of shadow page tables
- **Finding good trade-off is crucial for performance**
- **VMware has 9 years of experience here**

Shadow Page Tables and Scaling to Wide vSMP

- **VMware currently supports up to 4-way vSMP**
- **Problems lurk in scaling to higher numbers of vCPUs**
  - Per-vcpu shadow page tables
    - High memory overhead
    - Process migration costs (cold shadows/lack of shadows)
    - Remote trace events costlier than local events
  - vcpu-shared shadow page tables
    - Higher synchronization costs in VMM
- **Can already see this in extreme cases**
  - forkwait is slower on vSMP than a uniprocessor VM

Hardware Support

Nested/Extended Page Tables



Analysis of NPT

- **MMU composes VA->PA and PA->MA mappings *on the fly* at TLB fill time**
- **Benefits**
  - Significant reduction in “exit frequency”
    - No trace faults (primary page table modifications as fast as native)
    - Page faults require no exits
    - Context switches require no exits
  - No shadow page table memory overhead
  - Better scalability to wider vSMP
    - Aligns with multi-core: performance through parallelism
- **Costs**
  - More expensive TLB misses: $O(n^2)$ cost for page table walk, where n is the depth of the page table tree

Analysis of NPT

- **MMU composes VA->PA and PA-[illegible]gs *on the fly* at TLB fill time**
- **Benefits**
  - Significant reduction in "exit frequency"
    - No trace faults (primary page table modifications as fast as native)
    - Page faults require no exits
    - Context switches require no exits
  - No shadow page table memory overhead
  - Better scalability to wider vSMP
    - Aligns with multi-core: performance through parallelism
- **Costs**
  - More expensive TLB misses: $O(n^2)$ cost for page table walk, where n is the depth of the page table tree

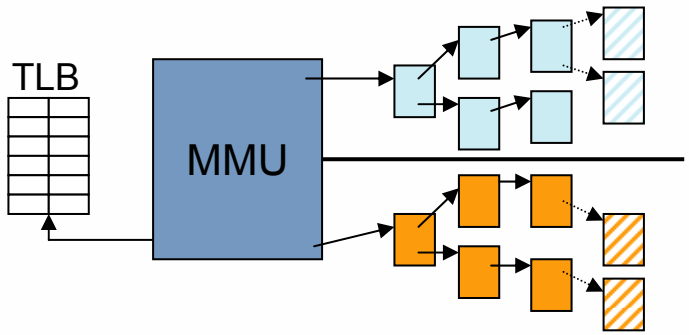
Explains Nat's near-native speed forkwait

Explains Nat's low performance memsweep

Improving NPT Performance

Large pages

- **2 MB today, 1 GB in the future**
  - In part guest's responsibility: "inner" page tables
    - For most guests/workloads this *requires explicit setup*
  - In part VMM's responsibility: "outer" page tables
    - ESX will take care of it
- **1st benefit: faster page walks (fewer levels to traverse)**
- **2nd benefit: fewer page walks (increased TLB capacity)**



Talk Outline

- **Part 1. Problem statement**
- **Part 2. Instruction set virtualization techniques**
- **Part 3. Memory virtualization techniques**
- **Part 4. Practicalities**
  - Which execution mode does my VM use?
  - How do I specify modes (when there is a choice)?
  - Performance suggestions
- **Conclusions**

Which Execution Mode Does my 32 bit VM Use?

- **General 32 bit rule: use Binary Translation**
- **Intel CPUs:**
  - No use of 1st generation VT-x: BT usually performs better
- **AMD CPUs:**
  - No use of 1st generation AMD-V: BT usually performs better
  - Option to use 2nd generation AMD-V NPT[1]

[1]Under consideration for future ESX releases.

Which Execution Mode Does my 64 Bit VM Use?

- **No general rule (AMD/Intel architecture divergence)**
- **AMD CPUs:**
  - Opteron Rev E and F: use BT
    - Rev F has 1st generation hardware support, but we prefer BT (perf)
  - 'Barcelona': use AMD-V with NPT[1]
    - 2nd generation hardware support usually performs better than BT
    - Retain BT option for workloads that don't benefit from NPT
- **Intel CPUs:**
  - Always use VT-x 1st generation hardware support
    - BT not possible (no segment limit checks)
  - EPT supported subject to hardware availability[1]

[1]Under consideration for future ESX releases.

User-visible Choice

To use or not to use NPT[1]

- **monitor.virtual_mmu = software (shadow page table, BT)**
- **monitor.virtual_mmu = hardware (NPT)**
- **monitor.virtual_mmu = automatic**
  - System tries to make the best choice
  - One day perhaps based on online profiling
  - Initially just based on guest OS type:
    - 64 bit VM: “automatic” selects NPT (and AMD-V)
    - 32 bit VM: “automatic” selects shadow page tables (and BT)

[1]Under consideration for future ESX releases.

When Should I Override “automatic”?

- **Depends on**
  - Guest OS
  - Workload
  - Memory size
  - Number of virtual CPUs
- **Bake-off between “software” and “hardware” MMU is best way to decide**

Factors Favoring “Hardware” MMU (NPT)

- **High rate of:**
  - page table updates (e.g., memmap/unmap)
  - context switches (process/process)
  - page faults
  - process creation
- **Higher numbers of virtual CPUs**
- **A need to reduce overhead memory**
- **Win2003 SP2 more likely to work well with NPT than other versions of (SMP) 32 bit Windows**
  - Reason: Microsoft eliminated most APIC TPR accesses whose overheads would otherwise defeat any NPT gains

General Performance Recommendations

- **TLB perf: configure workload to use 2 MB pages**
  - Especially important for NPT but also helps other modes
  - Not a guaranteed win (measure and compare)
    - GOS and system-level bottlenecks may arise
    - May increase memory usage (by reducing page-sharing)
- **I/O perf: vmxnet often more efficient than e1000**
  - Fewer “touches” per packet transmitted
  - Most important for hardware-assisted execution (high exit costs)
  - Adaptive BT tolerates higher rate of device touches

Conclusions
CPU performance

- **Complex and evolving space**
  - CPUs changing
  - Software improving
- **Diversity of execution modes**
  - Workload performance may be strongly affected by mode
  - No one mode is universally best
  - Ability to flexibly leverage different modes
    - Choice of mode partly locked down by CPU/VMM
    - Software/hardware MMU choice exposed to user

Conclusions
Benchmarks

- **Interpret results with reference to execution mode**
- **Micro-benchmarks**
  - Exaggerate certain properties
  - Understand degree to which these properties matter for real workload
- **“Real” workloads mix aspects of all microbenchmarks**
  - Less black and white
  - Usually performs closer to native (poison is much diluted)
  - Examples:
    `http://www.vmware.com/resources/techresources/cat/91,96`
- **No benchmark is as good a test as the real workload**

Questions?

TA68

Performance Aspects of x86 Virtualization

Ole Agesen

VMware

VMWORLD 2007

EMBRACING YOUR VIRTUAL WORLD

BREAKOUT SESSION