A Tale of Ten Bugs

Guest Lecture: Engineering Robust Server Software

Ravi Soundararajan

Performance Team, VMware

4/12/18

The Key to Good Performance

“Make the common case fast…

…but make sure it is correct…

…and make sure uncommon cases are correct, too…”

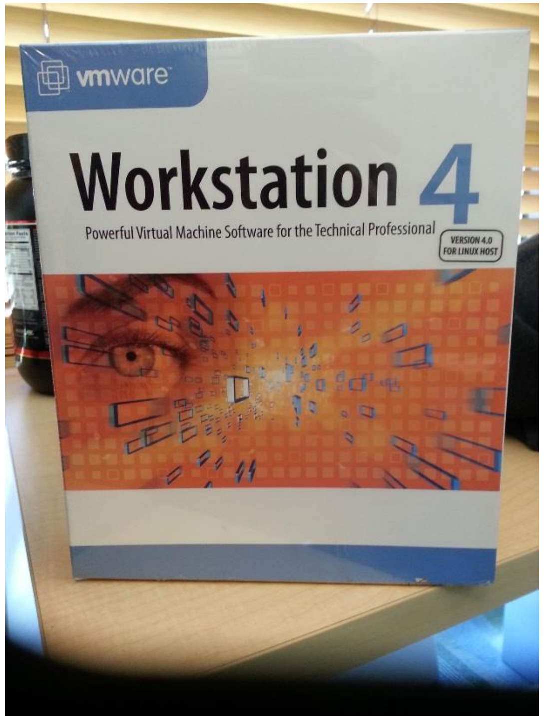
(By the way, make sure it really is the common case)

Good Examples

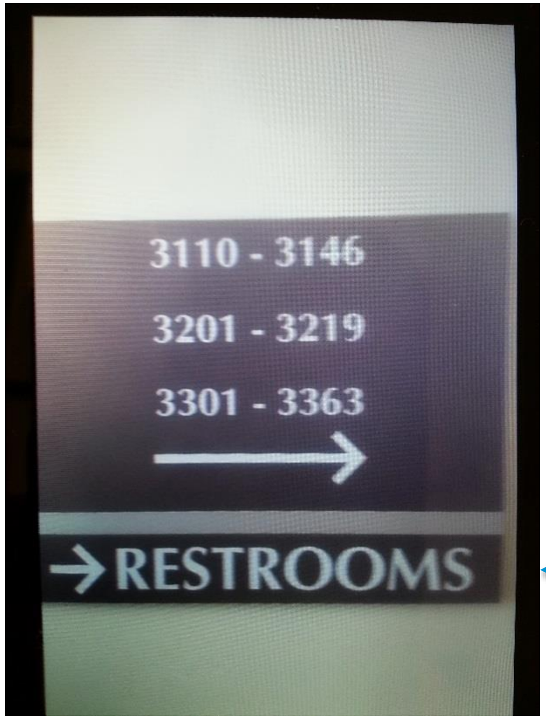
Clean interface, fast answers

Making the Common Case Fast: VMware

Common case:
User-level code



Making the Common Case Fast: Building Design

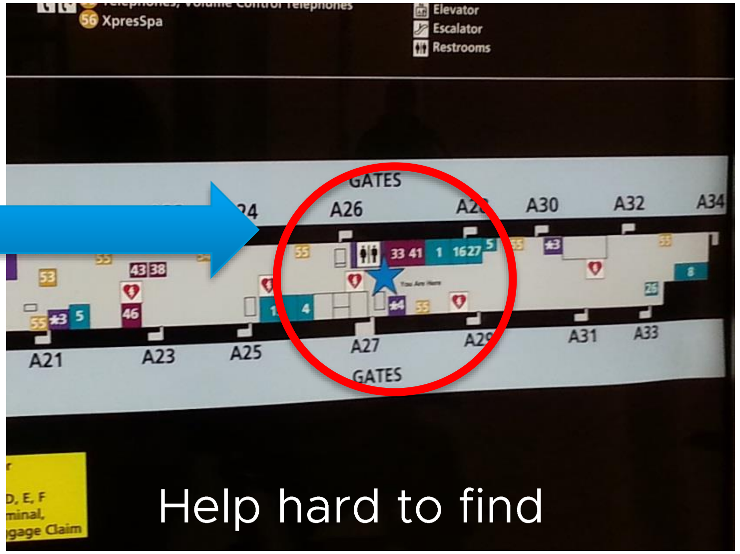


Most Likely Destination
(if you need the map)

Missing the Common Case: Taking the Train from Suburbs in Washington, DC

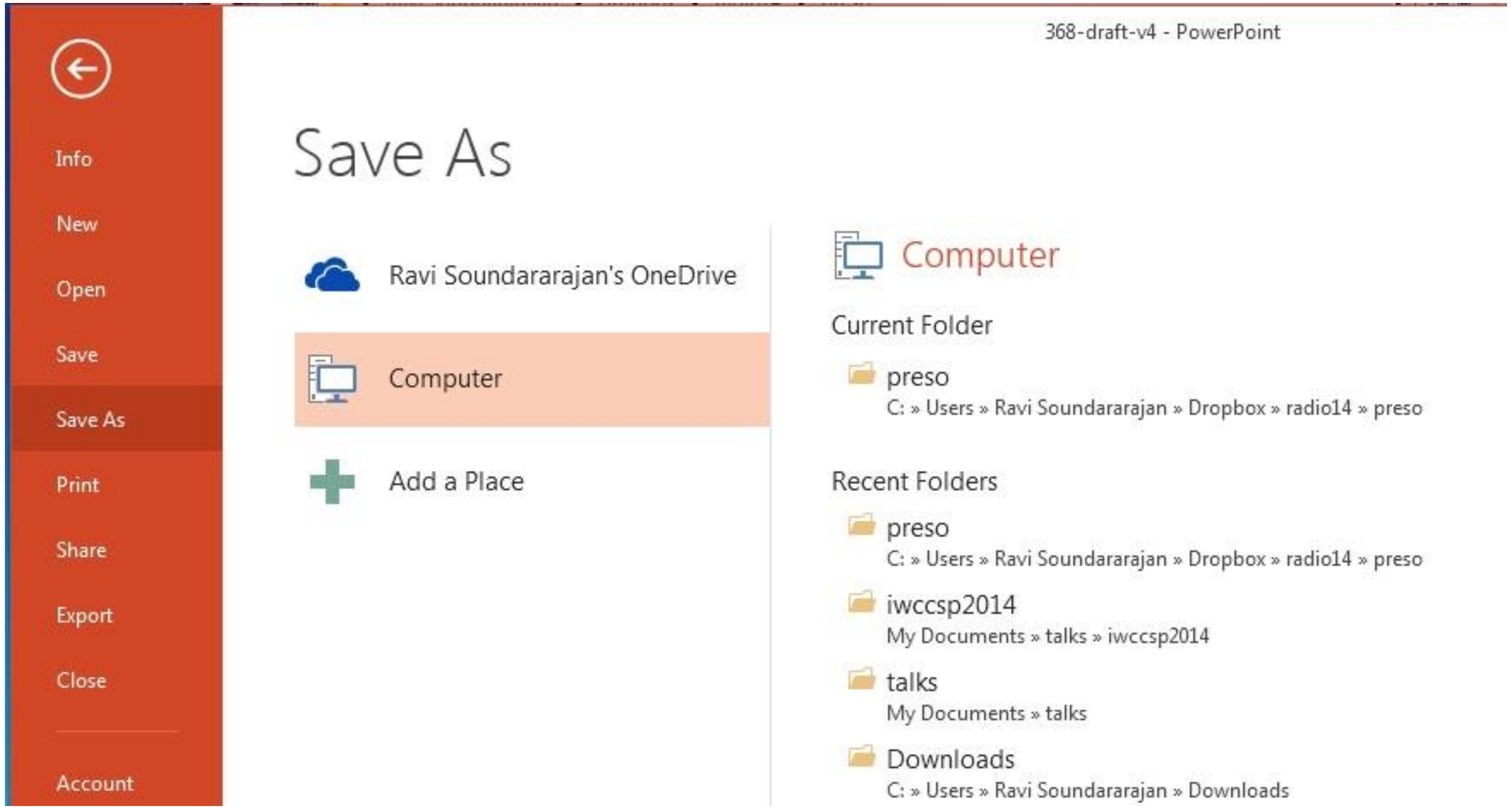
- Complex
- Doesn't leverage DC as likely destination

Missing the Common Case: Where Am I?



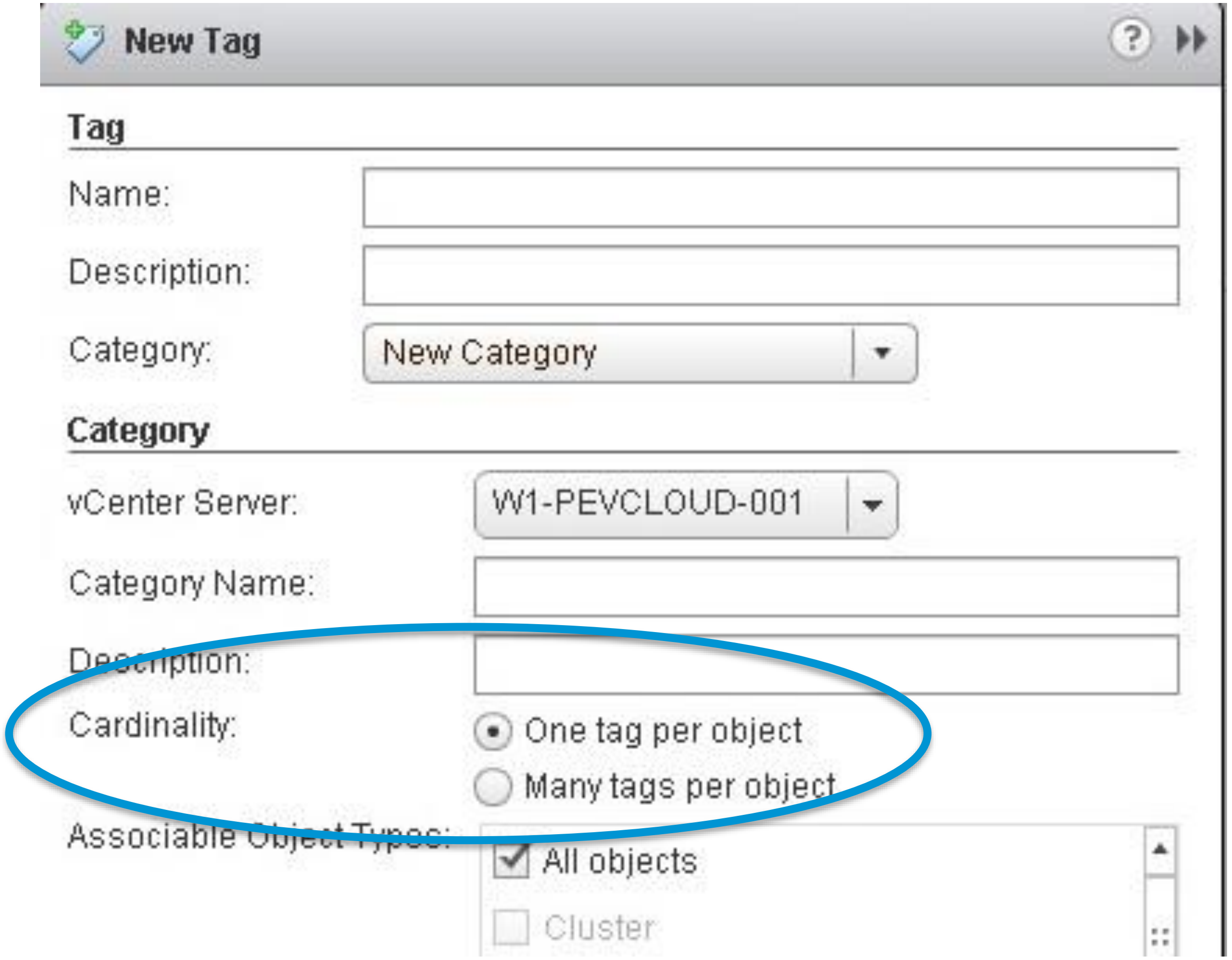
Help hard to find

Save As: Confusing Options



Where did my simple XP dialogue box go?

Cardinality?

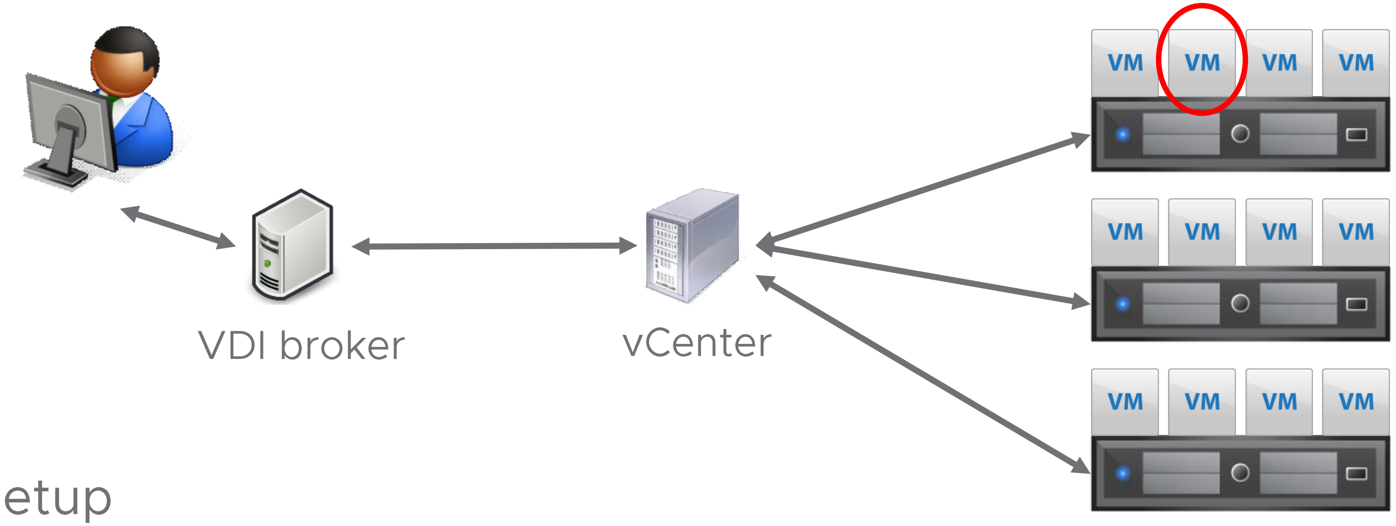


- Common case is 1:1
- Why not an advanced option?

Why is this tough to do?

A convoluted common case example

A Performance Problem



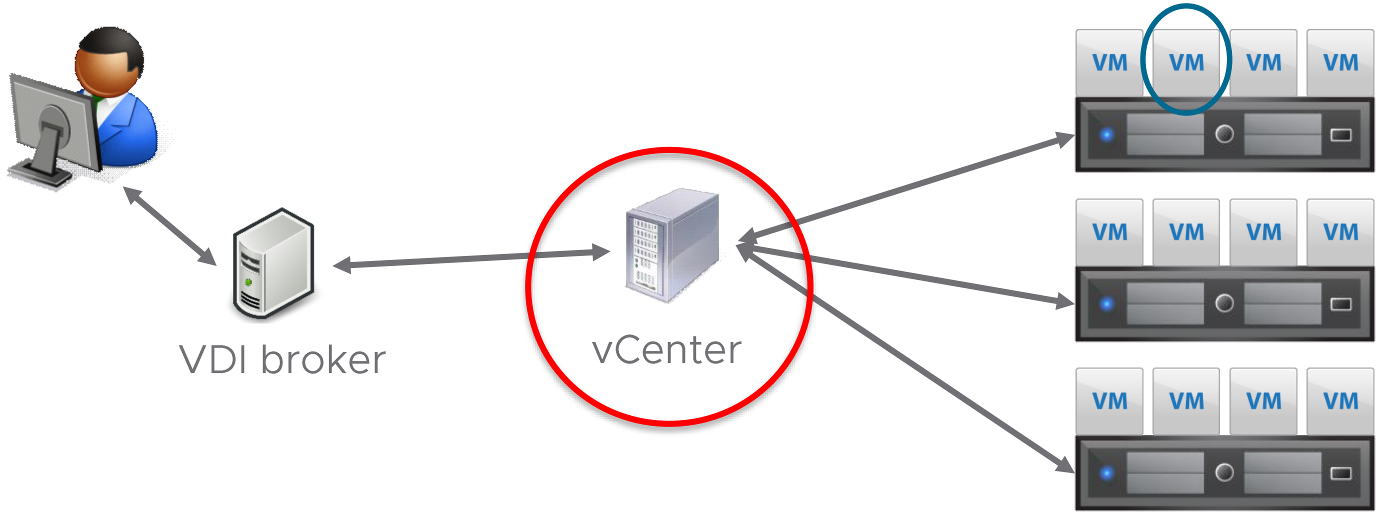
2 Changes in VDI setup

1. Upgraded vCenter
2. Added a few new hosts

vSphere hosts

Suddenly, getting desktop (VM) is slow

Initial Analysis



Symptom: High CPU usage on vCenter

Why?

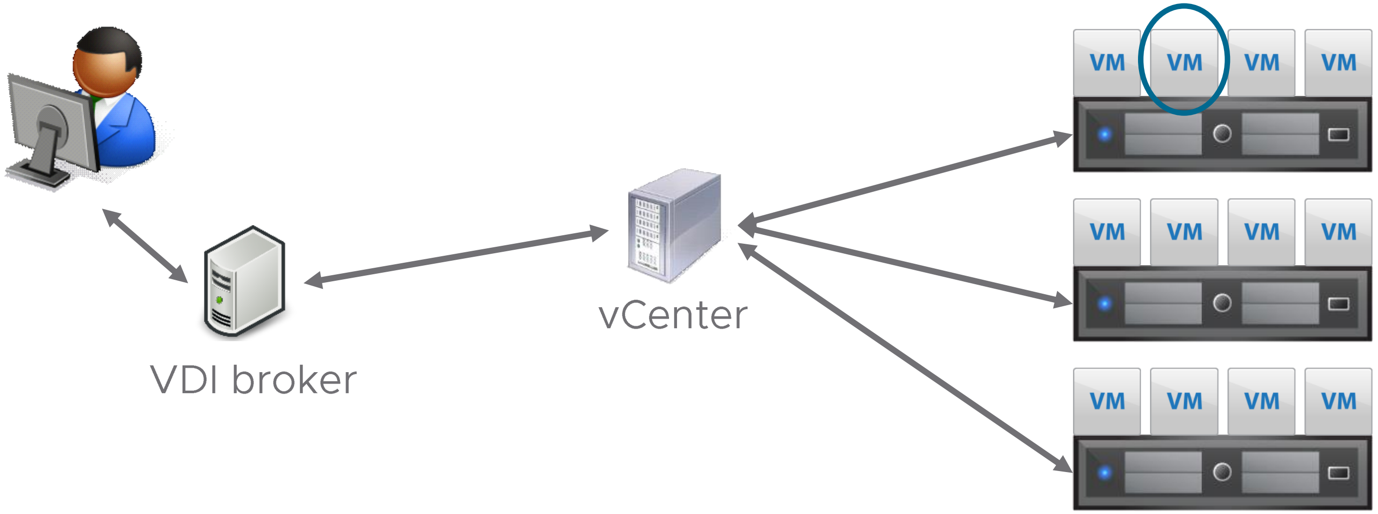
vCenter processing updates from vSphere hosts

(Observation: *fewer* updates in *newer* hosts ➔ Virtualization HW support)

Updates ultimately cause *license check* ➔ *high CPU*

vmware

Licensing: a red-herring?

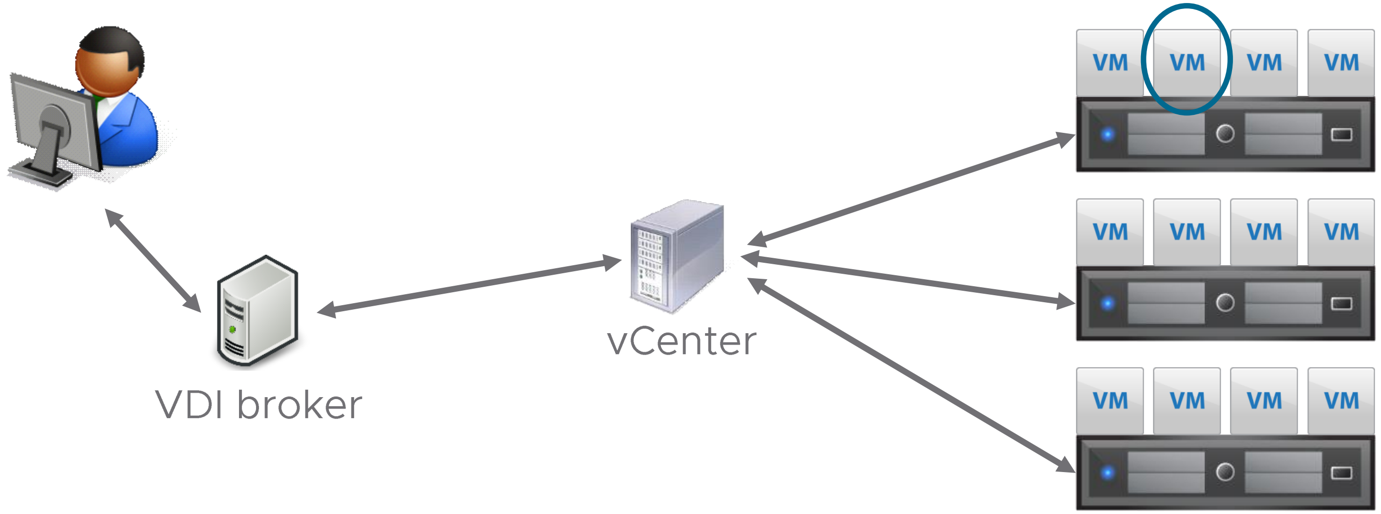


Why is licensing expensive?

Usually not, but miss in vCenter cache ➔ expensive string comparison

Weird…License checks should not miss in vCenter cache ☹

Cache Misses?

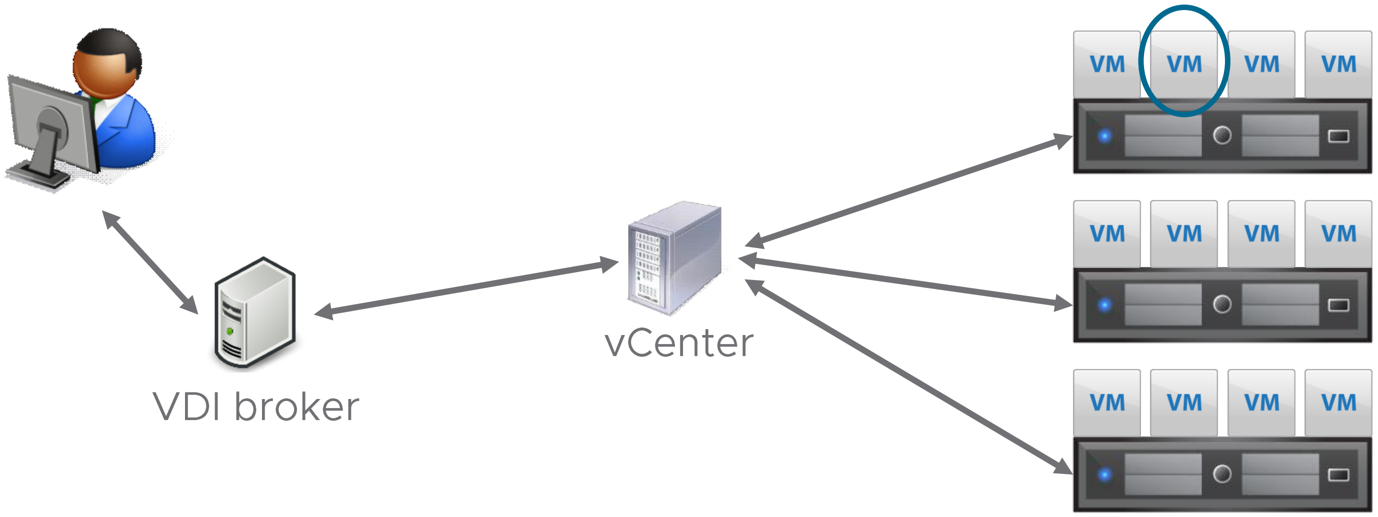


Why the sudden license misses?

➔ Added hosts caused vCenter cache overflow

➔ But…vCenter cache *much* bigger than previous release

Resolution



Good: we anticipated cache increase in vCenter

Bad: Bug in upgrade meant *OLD* cache size was used ☹

3 big customers were impacted in the same week

Common Case Scorecard

Make the common case fast?

- Yes: Cache prevents expensive license checks

Make sure it is correct?

- Yes

Make sure it is the common case?

- Yes: License checks are the common case. **BUT WHY???????**

Make the uncommon cases correct?

- No! Upgrade uncommon and wrong

Why did I show you this example?

Illustrates Complexity of Products and Debugging

- Touches entire stack from VMM all the way to VDI

Highlights Scalability

- Problem exacerbated by adding more hosts

Interesting Plot Twist: what HW do you design for?

- Problem may not have occurred if all hosts were new

Remainder of talk

9 more bugs

- Some annoying (networking)
- Some about languages (Java, C)
- Some about platforms (Linux, Windows)
- Some about hypervisors (CPU/Memory issues)

The Right Tool for the Job

A Simple Networking Performance Problem

Networking and ssh (1/4)

Basic problem: ssh is slow

loginSshSlow.avi

20s from connection attempt to asking for password

Why?

Networking and ssh (2/4)

Verbose logging on server and client

- Client: `ssh -vvv root@10.135.193.1 -p 1026`
- Server (n.b., my sshd running on 1026): `/usr/sbin/sshd -p 1026 -ddd`

loginSshSlowWithVerboseServer.avi

- avi file shows just verbose server logging

Seems to be a server-side issue (duh!)

Networking and ssh (3/4): strace and system calls

```
% strace -tt /usr/sbin/sshd -p 1026 -ddd

01:20:50.069828 stat("/etc/resolv.conf", {st_mode=S_IFREG|0644, st_size=347, ...}) = 0

01:20:50.069915 open("/etc/resolv.conf", O_RDONLY|O_CLOEXEC) = 4

…
```

REVERSE DNS LOOKUP

```
01:20:50.070167 read(4, "# Dynamic resolv.conf(5) file fo"..., 4096) = 347

…
```

WRONG DNS SERVER

```
01:20:50.070682 connect(4, {sa_family=AF_INET, sin_port=htons(53),
sin_addr=inet_addr("10.0.2.3")}, 16) = 0

…

01:20:50.070947 poll([{fd=4, events=POLLIN}], 1, 5000) = 0 (Timeout)

<!!! 5 SECOND GAP !!!>

01:20:55.076323 poll([{fd=4, events=POLLOUT}], 1, 0) = 1 ([{fd=4, revents=POLLOUT}])
```

Networking and ssh (4/4)

Problem:

- Reverse DNS lookup to wrong DNS server
- Two 5s timeouts before proceeding

2 solutions:

1. Ignore DNS in sshd (ok in lab, not production)

```
% /usr/sbin/sshd -p 1026 -ddd -o "UseDNS no"
```

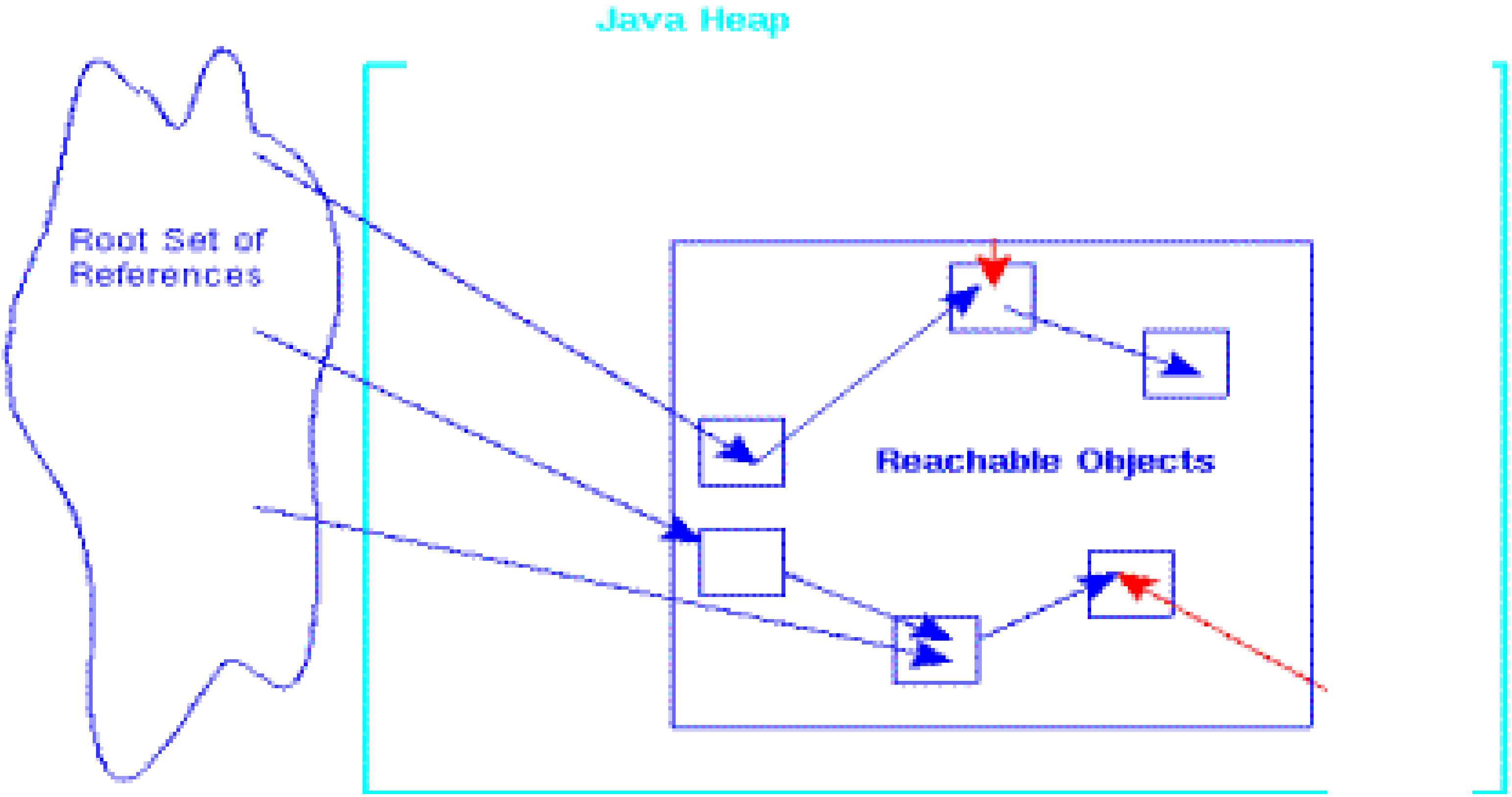
2. Fix DNS server setting (better!)

Trying #1 validated issue, and #2 fixed it for real

loginSshFast.avi

Java Memory Management

Java Memory Management Basics



Java memory management is done by the Java virtual machine
Garbage Collection: Find ‘unreachable objects’ and delete them

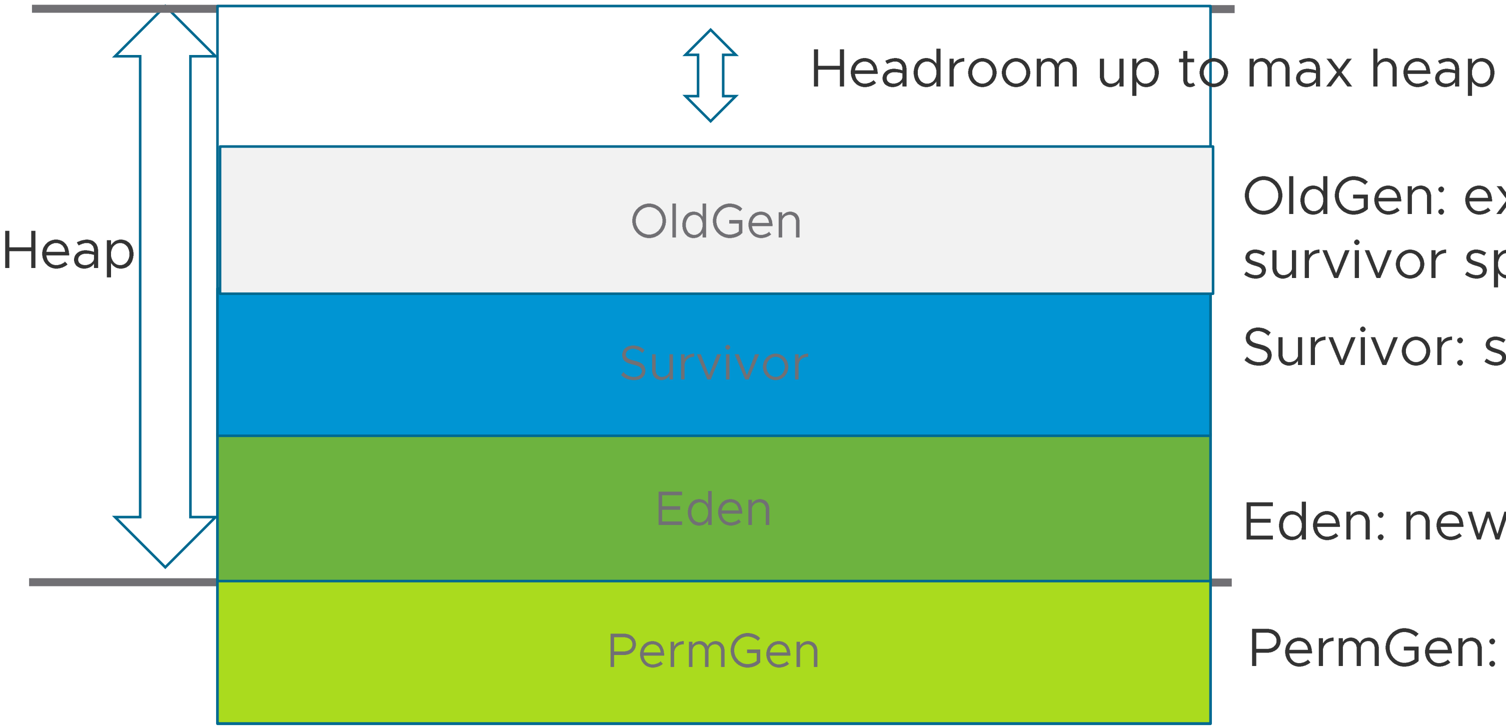
Diagram courtesy of http://java.dzone.com/articles/java-performance-tuning

Java Garbage Collection

"Mark, sweep, and compact" garbage collector:

- Mark: identify garbage
- Sweep: find garbage on heap, de-allocate it
- Compact: collect all live memory together

Java Memory (not including code cache)



OldGen: existed some time in survivor space

Survivor: survive GC of Eden

Eden: newly-created

PermGen: class definitions, etc.

Java GC and Tuning Notes

GC for Eden is frequent and hopefully low overhead

GC for “Oldgen” is less frequent and more CPU-intensive than Eden

Rule of thumb: most (80%?) of memory is short-lived

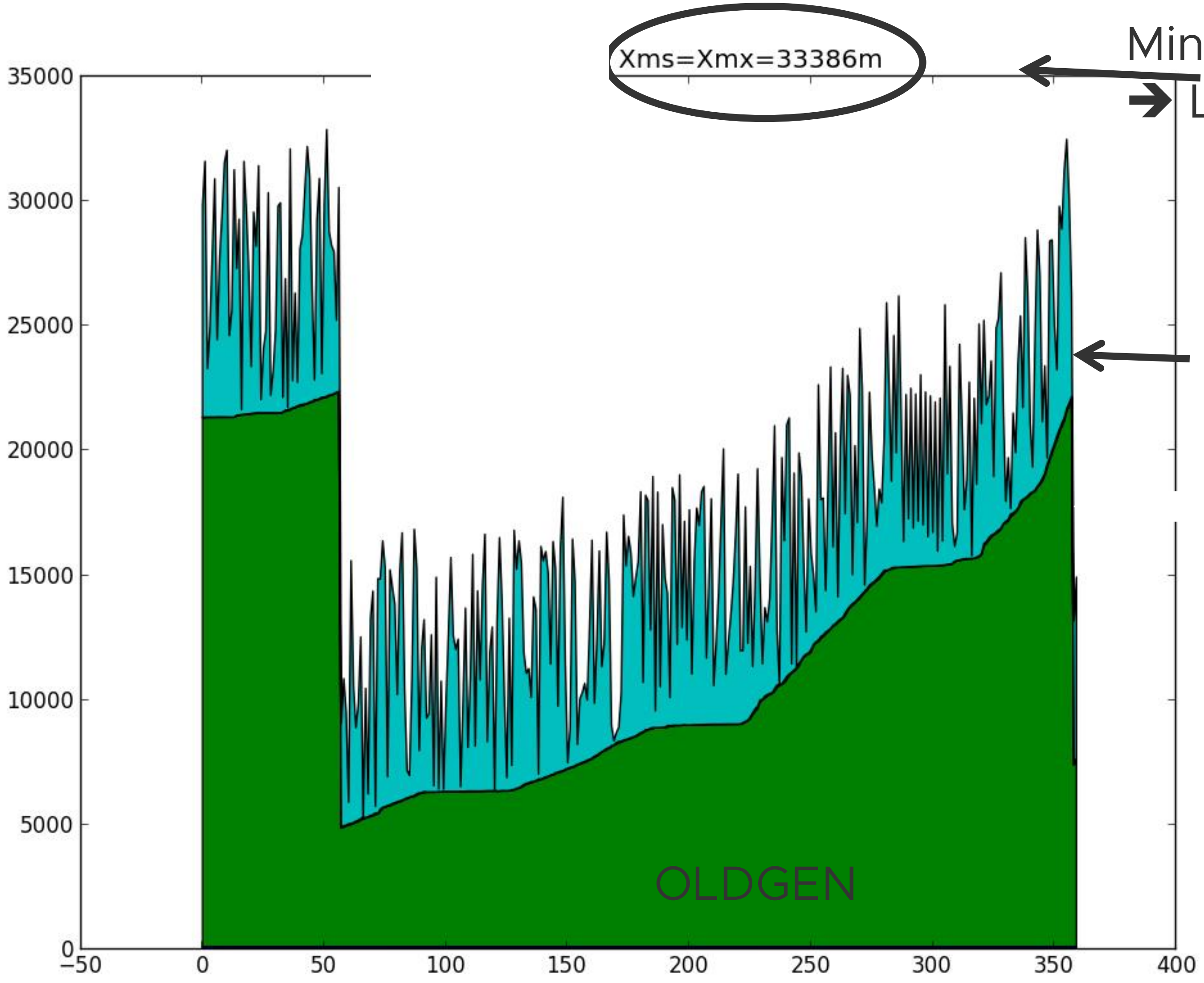
Many tunables in Java:

- Heap sizes (-Xms, -Xmx)
- Desirable ‘free heap’ ratio
- Survivor-to-Eden ratio
- Type of GC (serial, concurrent, mark/sweep, etc.)
- Number of GC threads
- Stack size (thread stacks NOT part of heap memory)
- Permgen size (not part of heap)

Profiling tools

- Yourkit, VisualVM, JMX counters, etc.

Pathological Memory Usage for a Java Process



Min heap = max heap
→ Less incentive to GC

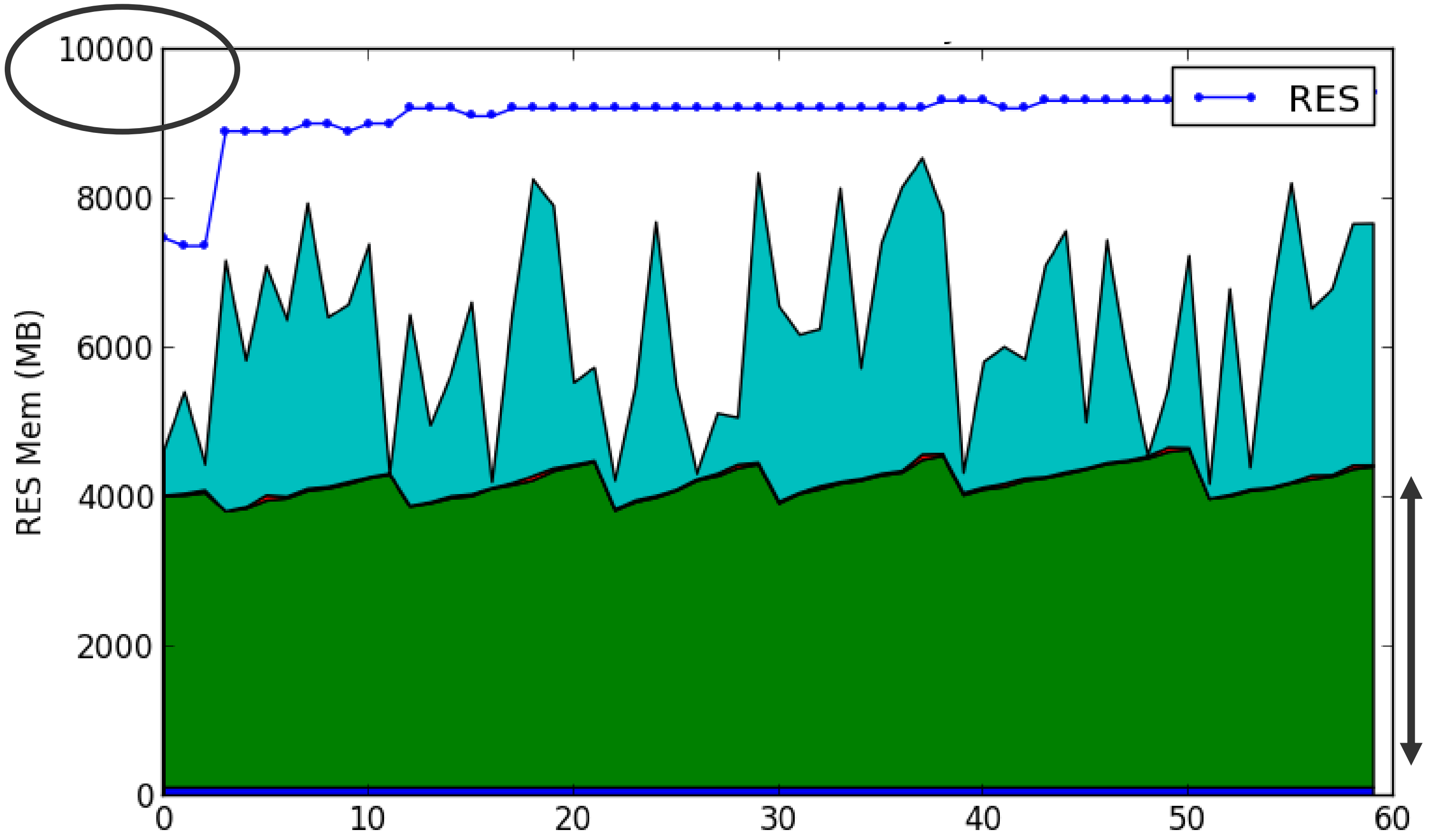
Eden
(Survivor negligible)

OldGen Growth: 5GB to 20GB!
Few OldGen GCs.
Why?

Min = max? Usually good only if you know what you need

Fixing the pathology

10GB



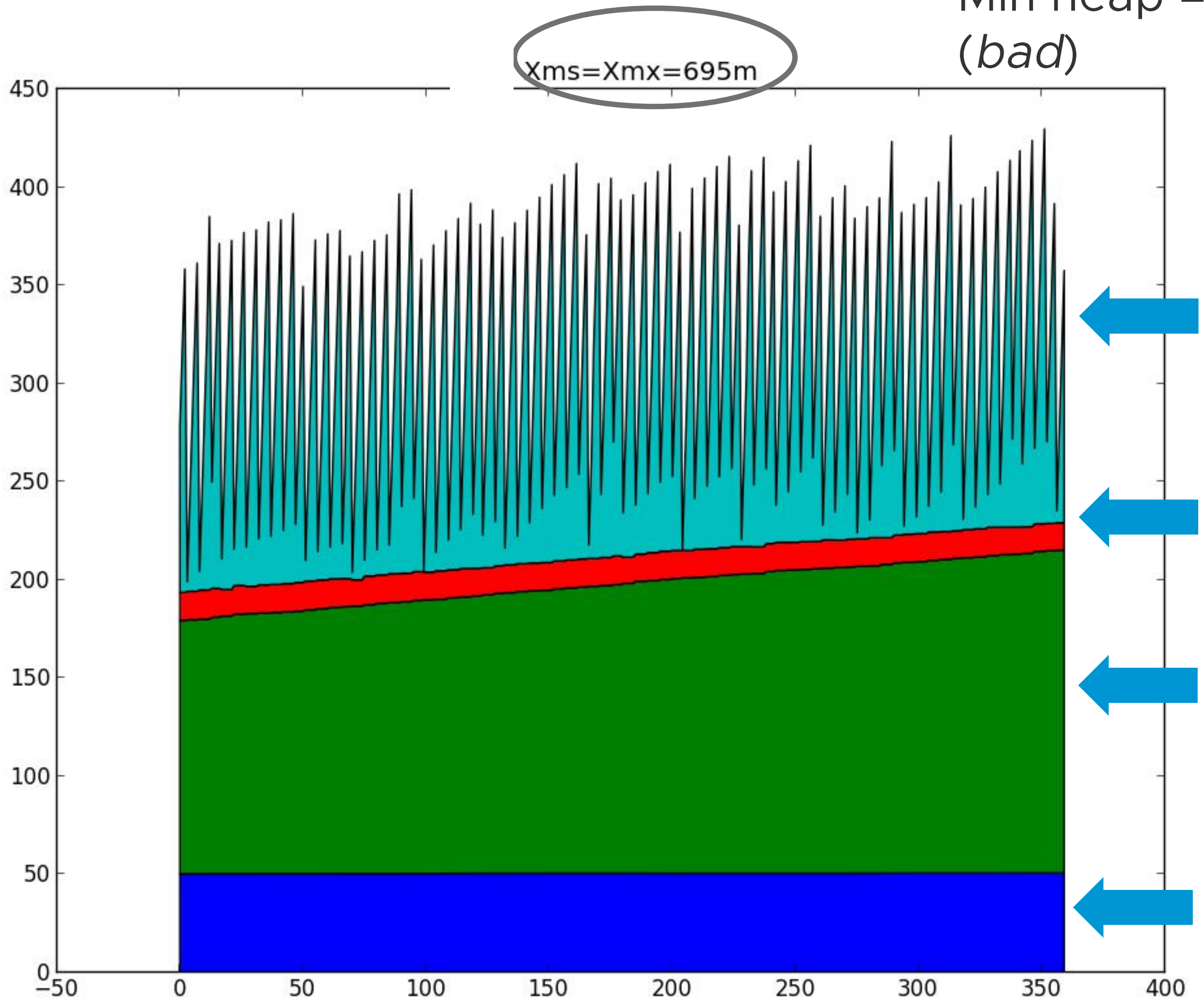
4GB Oldgen (vs. 20GB), More Frequent GCs.

--Shrink max heap setting

--Do not set Xms (initial heap). Do not set initial Permgen

Another example with min heap = max heap

Min heap = max heap
(*bad*)



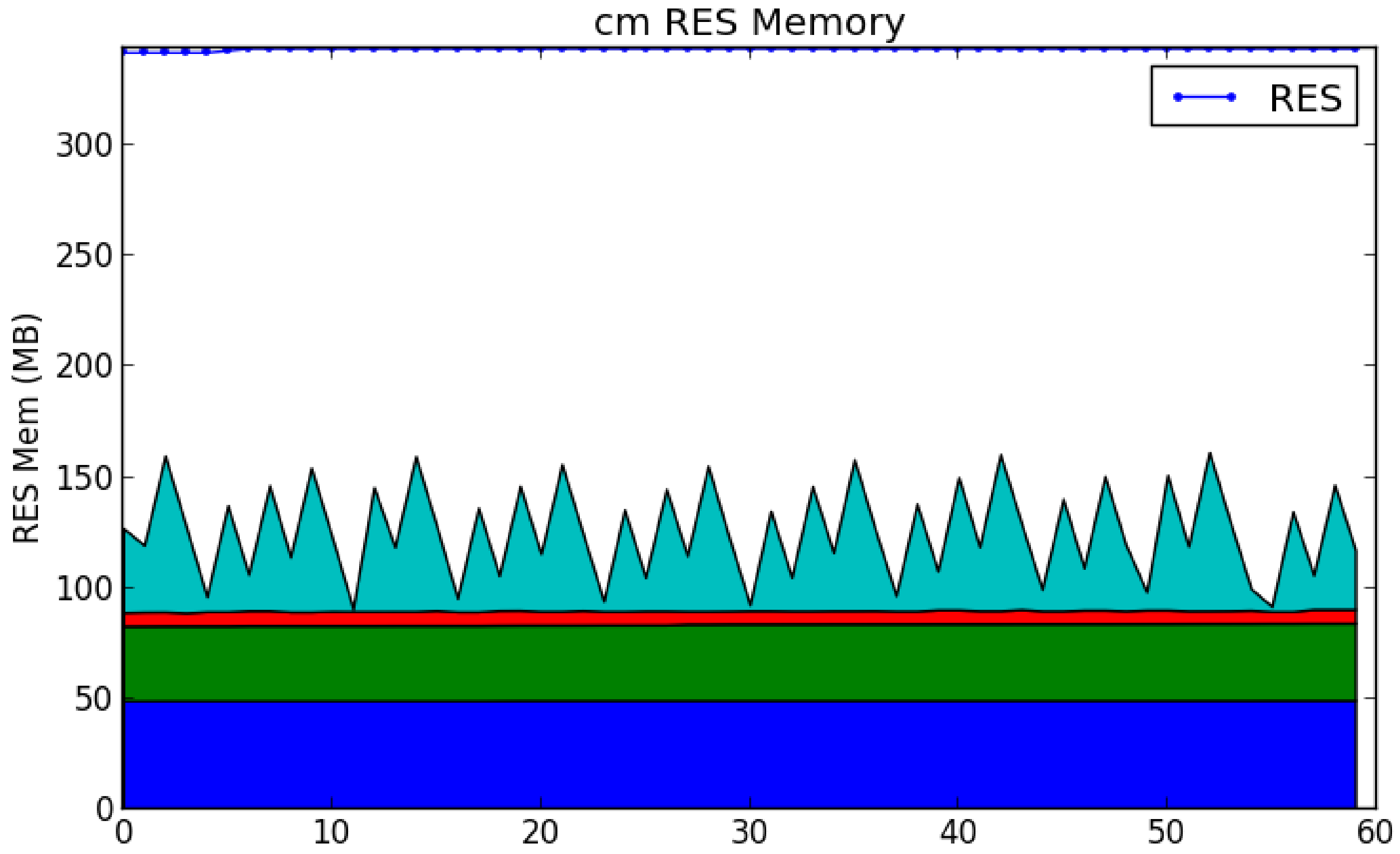
Eden
generation
Frequent GC
ok

Survivor
generation

Old generation:
Growing = may
be bad

Non-heap
(permgen, etc.)

Fixing the JVM settings: no permgen, no min heap



Lower max heap setting
Do not set min heap and do not set permgen: overall mem goes from ~400MB ➔ ~150MB

CPU profiling and diamond patterns

32-bit vs. 64-bit (Thanks, R. M.!)

Benchmark run

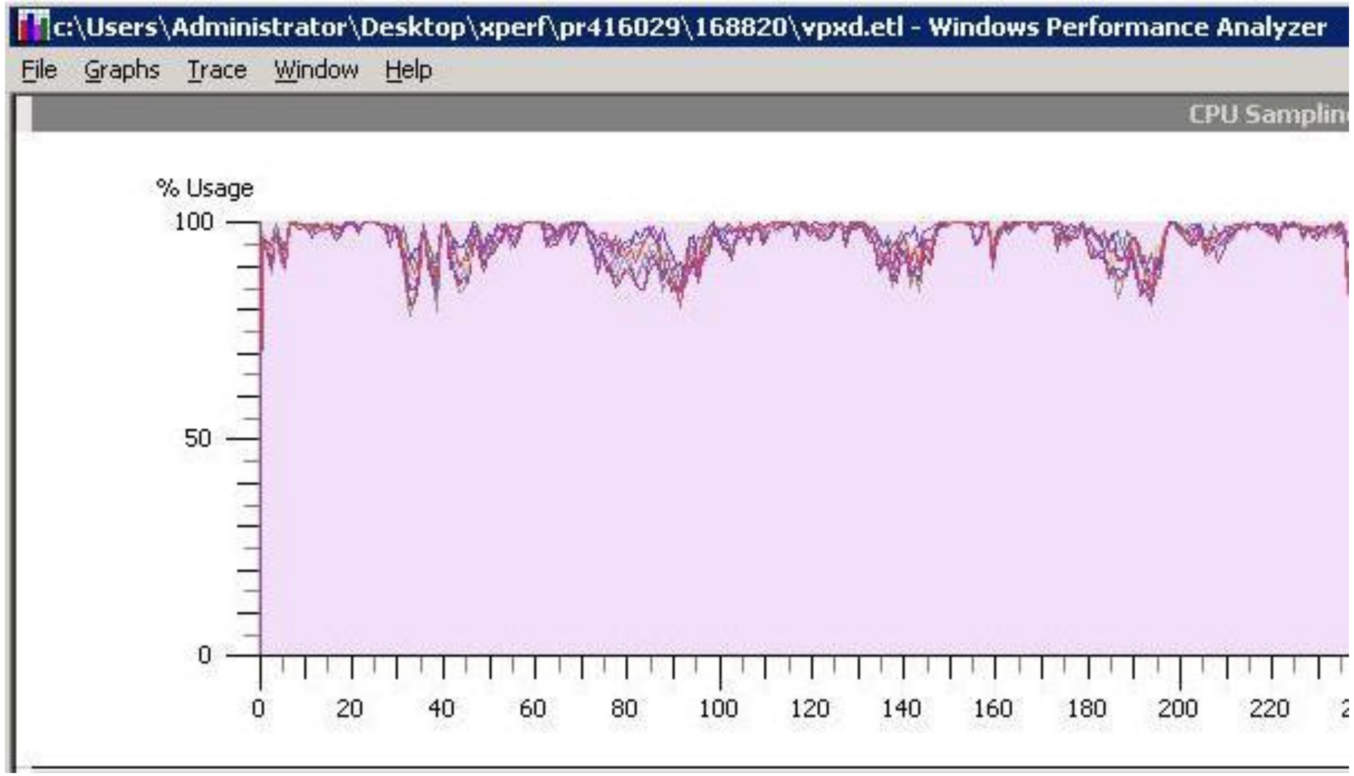
- Build A: 100 ops/min.
- Build B: 50 ops/min.

What was the difference?

- Build A: 32-bit executable on 64-bit hardware
- Build B: 64-bit executable on 64-bit hardware

Huh?

CPU Saturation in 64-bit case



CPU is mostly saturated (in 32-bit case, CPU is not saturated)
CPU Saturated → GOOD USE CASE FOR SAMPLING PROFILER

What _is_ xPerf?

Runs on Windows 2008-

Sampling profiler (with other cool attributes)

Records stack traces

Give caller/callee information

Look at Sampling Profile

Stack	Weight	% Weight
	3,532,612.638 ...	87.39
[Root]	3,532,455.583 ...	87.38
|- ntdll.dll!RtlUserThreadStart	3,508,983.437 ...	86.80
|- ntdll.dll!ZwQueryVirtualMemory	13,762.103 969	0.34
|- ntkrnlmp.exe!KiSystemServiceCopyEnd	5,903.542 252	0.15
|- ntkrnlmp.exe!KiChainedDispatch	901.654 720	0.02
|- ntkrnlmp.exe!KiDpcInterrupt	321.063 214	0.01
|- ntdll.dll! ?? ::FNODOBFM::`string'	258.007 162	0.01
|- ntdll.dll!RtlImageNtHeaderEx	226.071 622	0.01
|- MSVCR80.DLL!_RTtypeid	105.035 557	0.00

Shows stacks originating from root
Shows 87% CPU used from 1 process
But this is just the thread start routine, where threads originate

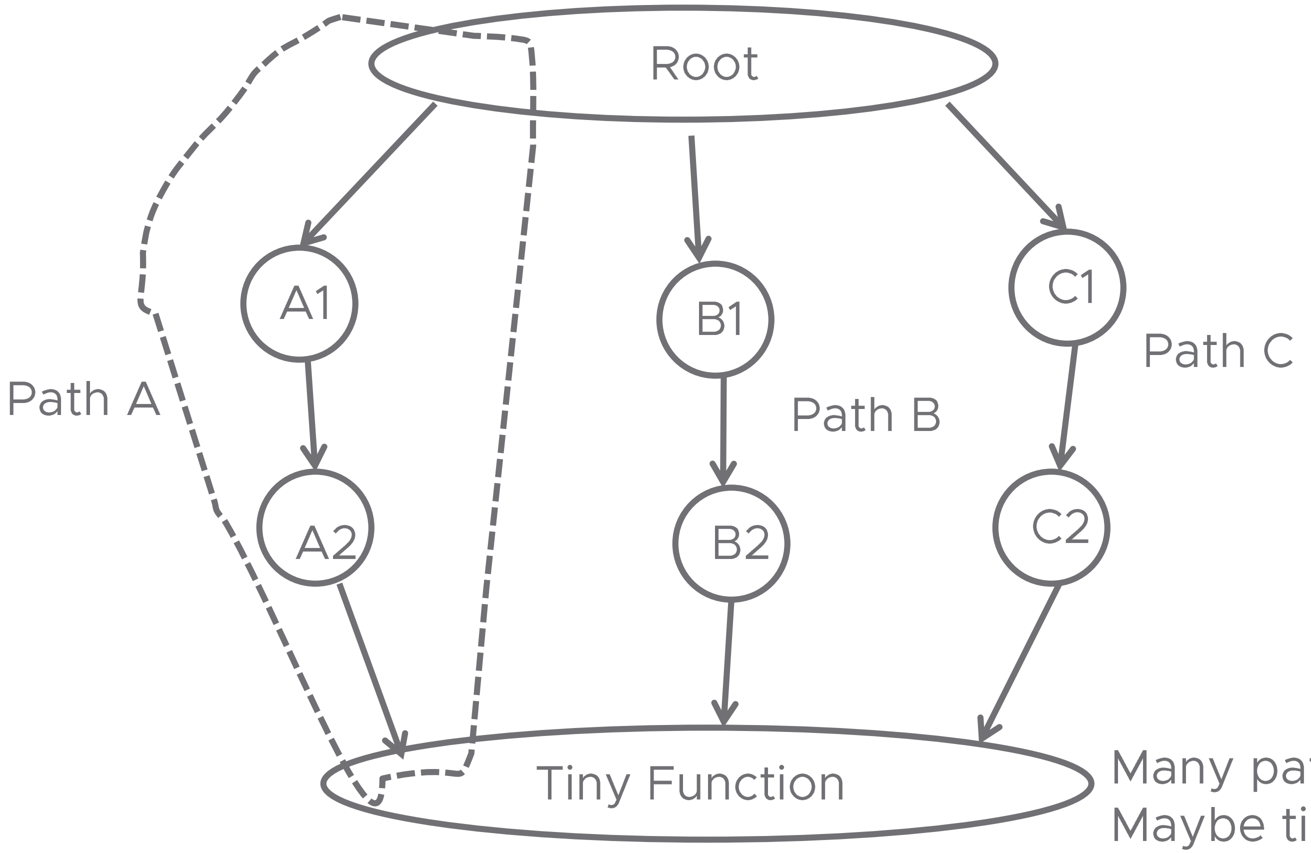
The Perils of Sampling Profilers

Stack	Weight	% Weight
	3,532,612.638 ...	87.39
[Root]	3,532,455.583 ...	87.38
ntdll.dll!RtlUserThreadStart	3,508,983.437 ...	86.80
kernel32.dll!BaseThreadInitThunk	3,508,983.437 ...	86.80
vpxd.exe!Win32ThreadMain	2,270,619.910 ...	56.17
vpxd.exe!VpxdThread::ThreadFunc	2,270,619.910 ...	56.17
vpxd.exe!VpxLroList::ThreadMainEntry	2,045,997.133 ...	50.61
vpxd.exe!VpxLRO::LroMain	2,041,093.903 ...	50.49
vpxd.exe!VpxActivationLRO::InvokeA...	1,571,975.867 ...	38.89
vmomi.dll!Vmomi::ManagedMethod...	1,518,822.158 ...	37.57
vmomi.dll!sVmodlQueryPropert...	770,053.167 788	19.05
types.dll!sVimVirtualMachineDi...	727,687.116 523	18.00
vpxd.exe!VpxdMoVm::Po...	672,630.405 844	16.64

From Root, most of the samples are from this call stack
Most popular stack, but is this the problem?

Perils of Sampling Profilers, Part 2

Most-common trace: not necessarily where time is spent



Many paths to “Tiny Function”
Maybe time spent here?

The Caller View

Look at Callers for various routines in stacks

Callers	Weight	% Weight
⊟ ntdll.dll!ZwQueryVirtualMemory	3,123,003.752 ...	77.26
⊟ |- ntdll.dll! ?? ::FNODOBFM::`string'	3,109,241.648 ...	76.92
⊞ | |- MSVCR80.DLL!_RTDynamicCast	1,579,519.929 ...	39.07
⊞ | |- MSVCR80.DLL!_RTtypeid	1,529,451.706 ...	37.84
⊞ | |- [Root]	257.005 987	0.01
⊞ | |- MSVCR80.DLL!CxxThrowException	13.006 609	0.00
⊞ |- [Root]	13,762.103 969	0.34
⊞ ntdll.dll!ZwQueryVirtualMemory	2.004 444	0.00

Not called a lot from root, however...
Called from few places and takes 77% CPU!
RTtypeid?

RTtypeid?

Callers	Weight	% Weight
⊟ MSVCR80.DLL!_RTtypeid	1,583,683.660 ...	39.18
⊞ |- vmacore.dll!Vmacore::ObjectImpl::IncRef	828,866.361 590	20.50
⊞ |- vmacore.dll!Vmacore::ObjectImpl::DecRef	725,473.308 729	17.95
⊞ |- MSVCR80.DLL!_RTDynamicCast	28,925.004 534	0.72
⊞ |- vpxd.exe!ManagedObjectMapper::operator()	283.942 793	0.01
⊞ |- [Root]	105.035 557	0.00
⊞ |- vpxd.exe!VpxLRO::GetStatsContext	18.005 567	0.00
⊞ |- vpxd.exe!DrmModule::SnapshotDomain	10.999 304	0.00
⊞ |- vmacore.dll!Vmacore::PrintFormatter::FormatException	1.002 432	0.00

Hmm. RTtypeid is used in figuring out C++ type
39% of overall CPU?
IncRef and DecRef are main callers

The Offending Code

```
void
ObjectImpl::IncRef()
{
   if (_refCount.ReadInc() == 0) {
      const type_info& tinfo = typeid(*this);
      FirstIncRef(tinfo);
   }

    …
}
```

typeid(): needs run-time type info (RTTI)

RTTI has pointers in it

But why is 64-bit slower than 32-bit?

Runtime type info (RTTI) has a bunch of pointers

- 32-bit: pointers are raw 32-bit pointers

- 64-bit
  - Pointers are 32-bit offsets
  - Offsets must be added to base addr of DLL/EXE in which RTTI resides
  - Result is a true 64-bit pointer

But wait…why is addition slow?

Why Is Addition Slow? Well, it isn't…

Addition isn't slow, but…

Determining module base address can be slow

- To find base address, RTtypeid calls RtlPcToFileHeader
- RtlPcToFileHeader grabs loader lock, walks list of loaded modules to find RTTI data
- This can be slow
- N.B.: This is why we see calls to zwQueryVirtualMemory

For more info: http://blogs.msdn.com/junfeng/archive/2006/10/17/dynamic-cast-is-slow-in-x64.aspx

What Did We Learn?

RtTypeId is called from a bunch of places

RtTypeId is not, however, called from Root too often

RtTypeId is small and fast: not main contributor in most stacks (*except IncRef and DecRef*)

Lots of little calls add up

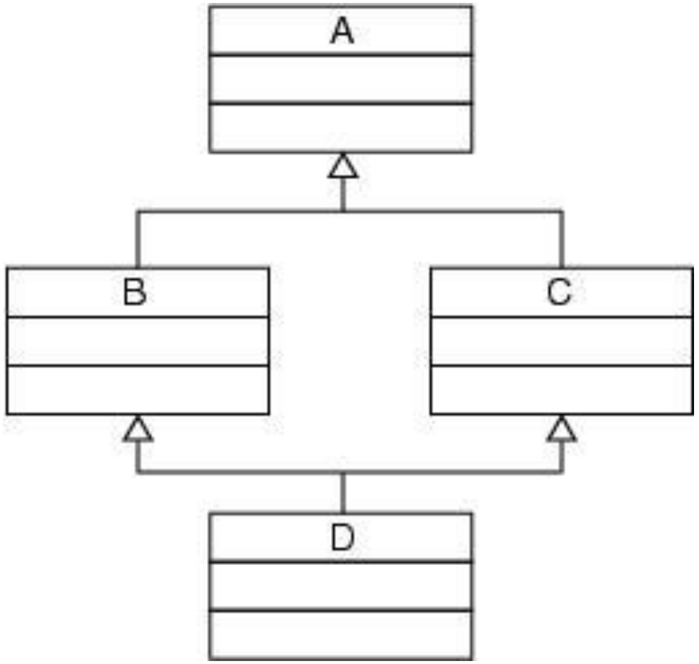
Caller view was important here!

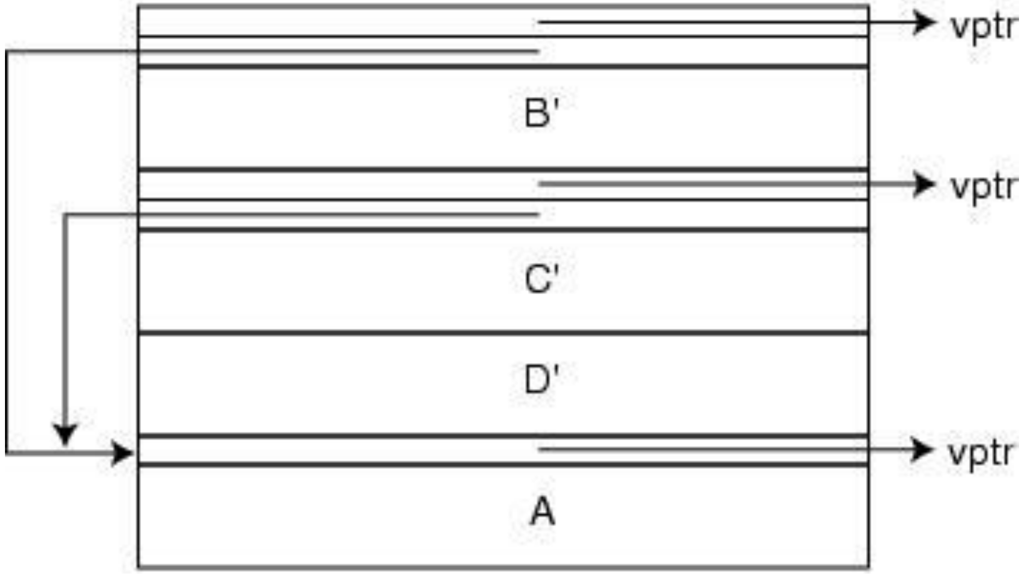
(btw: 2 solutions:

- 1. Statically compute base addr and cache
- 2. Use latest runtime library, which avoids RtlToPcFileHeader)

Of course, maybe we should reconsider design

Do we need multiple inheritance and dynamic_cast?






```
class D : public B, public C {
 public:
   virtual ~D();
   virtual void foo();

};
```

Multiple inheritance and dynamic_casts

```
class B : public A {
 public:
   virtual ~B();
   virtual void foo();

};

class C : public A {
 public:
   virtual ~C();
   virtual void foo();

};
```

```
class D : public B, public C {
 public:
   virtual ~D();
   virtual void foo();

};


D* ptrD = dynamic_cast<D*>(ptrA);
```

Why do we use multiple inheritance?

- Store data as Object *
- Upon retrieval, do dynamic_cast
- Many objects need to inherit from various parents

Nice Url: *http://www.drdobbs.com/cpp/multiple-inheritance-considered-useful/184402074*

Mallocs, Strings, and Ints

Microbenchmarks and macro conclusions

Question: How efficient is your software?

VMware software spans many layers:

- Virtual Machine monitor
  - Needs small footprint for best performance
  - Any CPU cost becomes virtualization overhead: slower guests
- Kernel
- Higher-level application software

➔For best performance, apply ‘monitor’ techniques to higher-level software

RDTSC: read timestamp counter

Lets you see the number of cycles for a section of code

```
#if defined(__x86_64__)


static __inline__ unsigned long long rdtsc(void)

{

    unsigned hi, lo;

    __asm__ __volatile__ ("rdtsc" : "=a"(lo), "=d"(hi));

    return ( (unsigned long long)lo)|( ((unsigned long long)hi)<<32 );

}


#endif
```

Rdtsc malloc/free test

```
t1 = rdtsc();

for (i = 0; i < num_iters; i++) {

     testFoo = (foo_t *)malloc(sizeof(foo_t));

     free(testFoo);

}

t2 = rdtsc();


printf("malloc/free loop 1st time average latency: %llu\n", (t2-t1)/num_iters);
```

➔ On average, about 50 cycles per malloc, 50 cycles per free

➔ Variance: occasional memory issues ➔ 500 cycles per iteration

➔ Is 50 cycles per malloc ok for you?

Malloc from glibc (a subset) (1/2)

```
static void *
_int_malloc (mstate av, size_t bytes)
{
  INTERNAL_SIZE_T nb;               /* normalized request size */
  unsigned int idx;                 /* associated b          */
  mbinptr bin;                                               /
  mchunkptr victim;                    inspected/selected chunk */
  INTERNAL_SIZE_T s                 /* its size */
  int victim_index;                 /* its bin index */
  mchunkptr remainder;              /* remainder from a split */
  unsigned long remainder_size;     /* its size */
  unsigned int block;               /* bit map traverser */
…
```

Lots of Code

Malloc from glibc (2/2)

```
/*
   Convert request size to internal form by adding SIZE_SZ bytes
   overhead plus possibly more to obtain necessary alignment and/or
   to obtain a size of at least MINSIZE, the smallest allocatable
   size. Also, checked_request2size traps (returning [illegible]quest sizes
   that are so large that they wrap a[illegible]d and
   aligned.
 */
checked_request2size (bytes, nb);
```

Lots More Code ☺

… (lots more code) …

The point is that malloc isn't free.

Other options: Different malloc libraries? Custom memory management?

vmware

Rdtsc string vs. integer compare

```
t1 = rdtsc();

for (i = 0; i < num_iters; i++) {

     equal = strncmp(s1,s2,strlen(s1));

}

t2 = rdtsc();
```

String comparison:
81 cycles per loop

```
t1 = rdtsc();

for (i = 0; i < num_iters; i++) {

     equal = (num1 == num2);

}

t2 = rdtsc();
```

Integer comparison:
6 cycles per loop

Strncmp: 81 cycles

```
% objdump -S -l -C test

➔ equal = strncmp(s1,s2,strlen(s1));

400aa7:    48 8b 45 f0             mov    0xfffffffffffffff0(%rbp),%rax
400aab:    48 c7 c1 ff ff ff ff    mov    $0xffffffffffffffff,%rcx
400ab2:    48 89 85 20 ff ff ff    mov    %rax,0xffffffffffffff20(%rbp)
400ab9:    b8 00 00 00 00          mov    $0x0,%eax
400abe:    fc                      cld
400abf:    48 8b bd 20 ff ff ff    mov    0xffffffffffffff20(%rbp),%rdi
400ac6:    f2 ae                   repnz scas %es:(%rdi),%al
400ac8:    48 89 c8                mov    %rcx,%rax
400acb:    48 f7 d0                not    %rax
400ace:    48 8d 50 ff             lea    0xffffffffffffffff(%rax),%rdx
400ad2:    48 8d b5 50 ff ff ff    lea    0xffffffffffffff50(%rbp),%rsi
400ad9:    48 8b 7d f0             mov    0xfffffffffffffff0(%rbp),%rdi
400add:    e8 76 fb ff ff          callq  400658 <strncmp@plt>
```

Integer compare: 6 cycles

```
    equal = (red_apple_six == inputNum);
400d8b:     8b 45 b0          mov    0xffffffffffffffb0(%rbp),%eax
400d8e:     83 f8 06          cmp    $0x6,%eax
400d91:     0f 94 c0          sete   %al
400d94:     0f b6 c0          movzbl %al,%eax
400d97:     89 45 ac          mov    %eax,0xffffffffffffffac(%rbp)
```

Straight-line code, no function calls.

➔ For performance, prefer ints over strings if possible

Strings and Things

Memory allocation differences between Linux and Windows

Memory differences: Linux vs. Windows

Motivation: runtime memory was 2x in Windows vs. Linux

Why?

Offsets in Windows (from windbg)

```
0:000> dt vpxd!VmMo -v

…

+0x7b8 _configId          : std::basic_string<…>

+0x7e0 _layoutId          : std::basic_string<…>

+0x808 _layoutExId        : std::basic_string<…>

…
```

➔ at least 40B between strings no matter what

Offsets in Linux (from gdb)

> gdb vpxd vpxd.core

```
(gdb) printf "0x%x\n", &((('VmMo' *) 0)->_configId)

0x5f0

(gdb) printf "0x%x\n", &((('VmMo' *) 0)->_layoutId)

0x5f8

(gdb) printf "0x%x\n", &((('VmMo' *) 0)->_layoutExId)

0x600
```

Only 8B between strings. Why?

Strings in Windows

```
{
  std::_Container_base_12   # ptr 8B
  _Bx (union) {             # 16B
       _Buf                 # The string, if it fits
       _Ptr                 # ptr to string, if not
       _Alias
  }
  _Mysize                   # 8B
  _Myres                    # 8B (reserved space)
}
```

- Note: 40B minimum for each instance of the string

Windows Strings

Example from Visual Studio

[Raw View]	0x000000000026f850 {...}
std::_String_alloc<0,std	{...}
std::_String_val<std	{_Bx={_Buf=0x000000000026f858 "red apple4" _Ptr=0x6c70706120646
std::_Container_I	{_Myproxy=0x000000000039e730 {_Mycont=0x000000000026f850 {_M
_Myproxy	0x000000000039e730 {_Mycont=0x000000000026f850 {_Myproxy=0x0
_Bx	{_Buf=0x000000000026f858 "red apple4" _Ptr=0x6c70706120646572 <
_Buf	0x000000000026f858 "red apple4"
_Ptr	0x6c70706120646572 <Error reading characters of string.>
_Alias	0x000000000026f858 "red apple4"
_Mysize	0x000000000000000a
_Myres	0x000000000000000f

Strings in Linux (from glibc documentation)

```
_M_dataplus      # Default cost of any string

{

  _M_p           # 8B Ptr to char[] of string body

}

# String body ➔ 24B + sizeof(char [])

{

   _M_length     # 8B

   _M_capacity   # 8B

   _M_refcount   # 8B Reference Count

   char []       # the string (shared among instances)

}
```

- Important: _M_refcount allows string body sharing!
- 20 instances of string: 19 are 8B, 1 is 32B + sizeof(char [])

A Sample String Body with a High Reference Count

```
(gdb) x/32a 0x7f05c0038fb0

0x7f05c0038fb0: 0xf       0xf

0x7f05c0038fc0: 0x41f     0x726774726f707664

0x7f05c0038fd0: 0x3830332d70756f        0x55

0x7f05c0038fe0: 0x7f05c0000158  0x7f05c0000158
```

_M_refcount: 0x41f = 1055 instances shared

Windows: 1055 x 40B = ~40KB

Linux: 1*40B + 1054 * 8B = ~8KB

If we had 1M objects: Windows 40MB, Linux 8MB ➔ 32MB delta

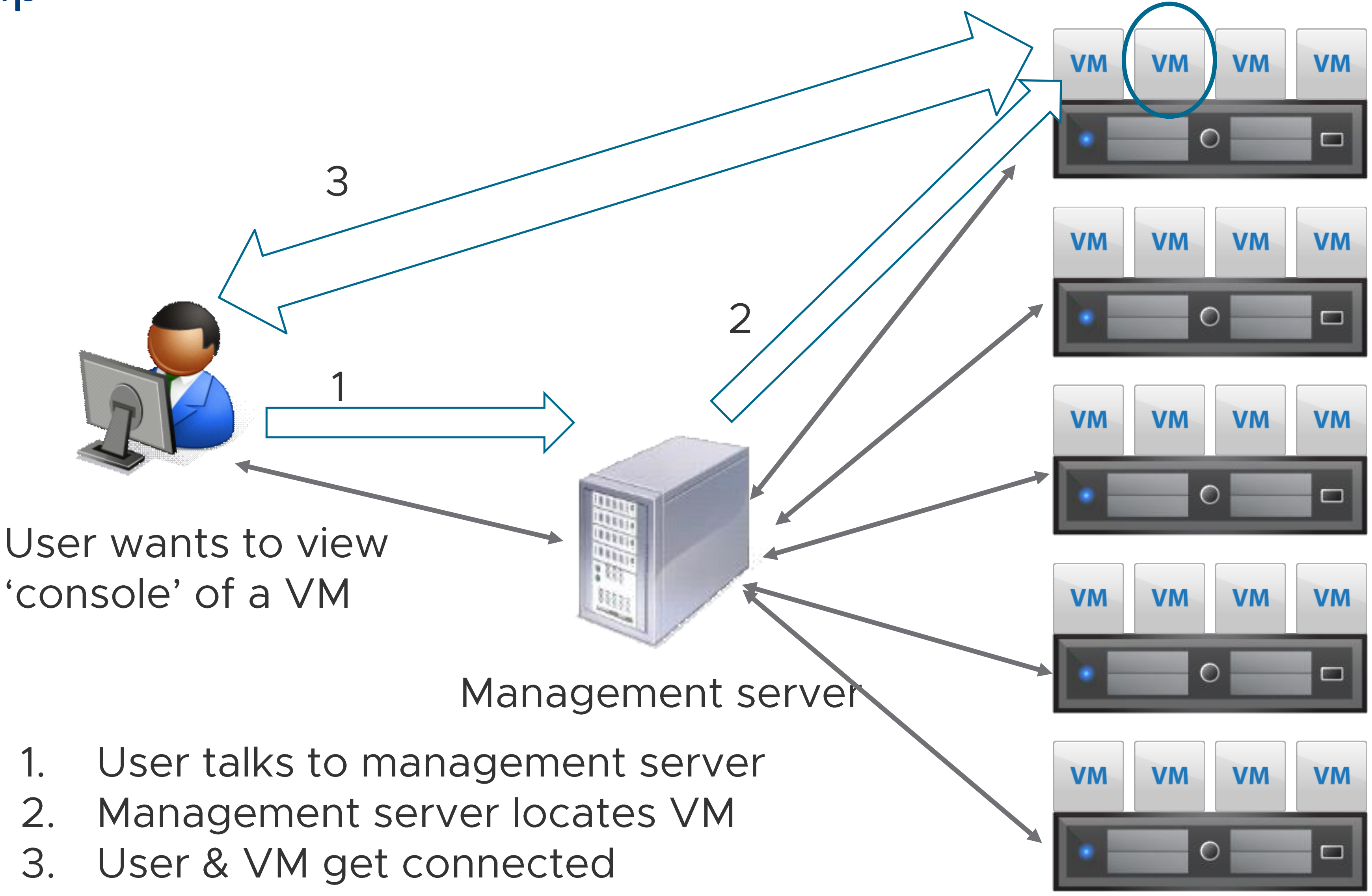
➔ Different platforms utilize memory differently

➔ Be careful which libraries you use (or roll your own)

vmware

Connecting the Dots: A Remote Console story

The setup



1. User talks to management server
2. Management server locates VM
3. User & VM get connected

The Problem: Remote Console Doesn't Show Up

- Problem: could not start VM remote console in large environment

- Sequence of debugging
  - Client folks: it's a server problem
  - Server folks: it's a client problem
  - Client folks: it's a 'vmrc' problem (vmrc = VMware Remote Console)
  - VMRC folks: authentication? MKS tickets?
  - I got curious…

- More Information: Start remote console for a single VM
  - 50 Hosts, no problem
  - 500 Hosts, no problem
  - 1001 Hosts, PROBLEM!

No Console: Examining the Cases that Actually Work

- Debugging observations
  - With < 1000 hosts…
    - Management server CPU and memory goes very high when client invoked
    - Console is dark until CPU and memory go down, then appears

  - Look at server log file
    - Data retrieval call occurs before console appears (WHY???)
    - In failure cases, exception in serializer code

  - Attach debugger
    - Exception is an out-of-memory exception
    - Exception is silently ignored (never returns to client)

No Console: Isolating the Problem

- Problem
  - VMRC creates a request to monitor host information (e.g., is CD-ROM attached)
  - Request gets info on ALL hosts
  - At 1001 hosts, we exceed 200MB buffer on server
  - 200MB restriction only for old-style API clients
- Solution
  - VMRC folks: do NOT create big request
  - Server folks: fail correctly and emit better errors

Nice lessons learned

1. *Create APIs that are difficult to abuse, rather than easy to abuse*
2. *Teach clients how to use APIs*
3. *Make sure (internal) users have input about API design*
4. *Be data-driven in your analysis ☺*

Understanding and using metrics

Memory

Windows-Dev limits/shares example

Windows VM is really slow.

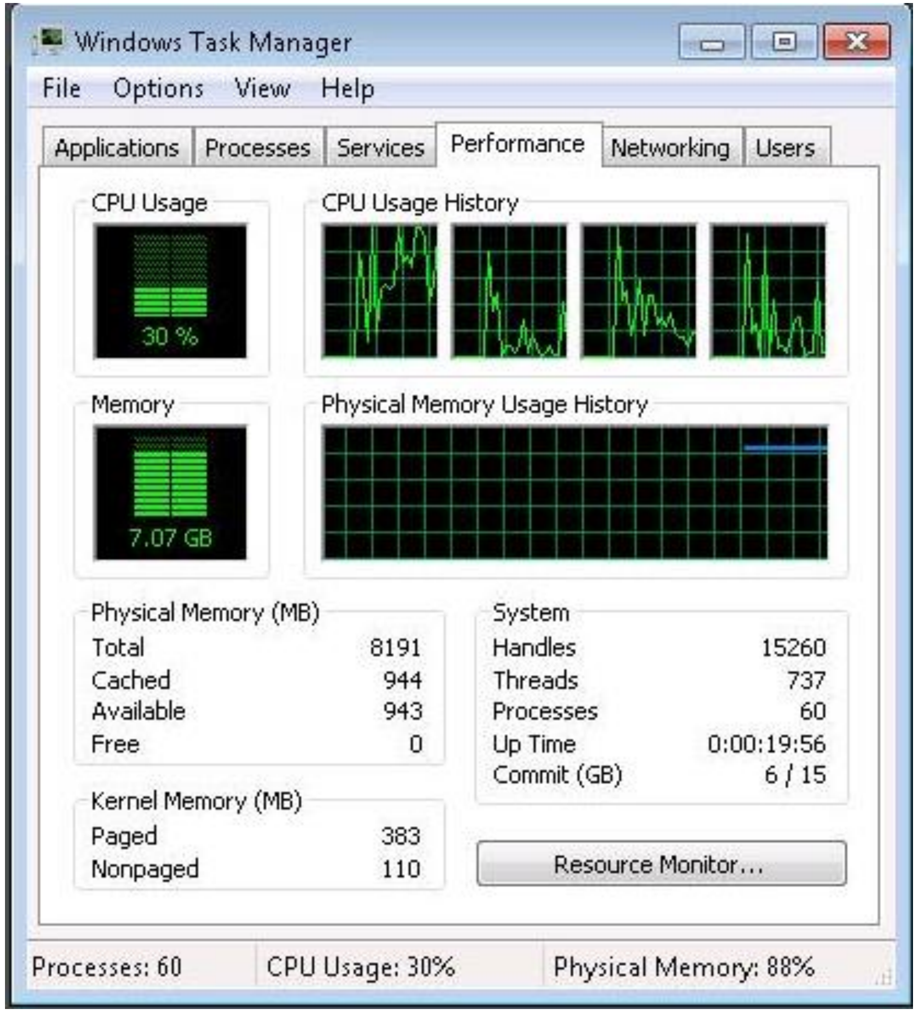
Examples:

- Bootup and login extremely slow.
- Starting up profiling tools (xperf) extremely slow

Starting point in Windows: TaskManager

In-guest metrics

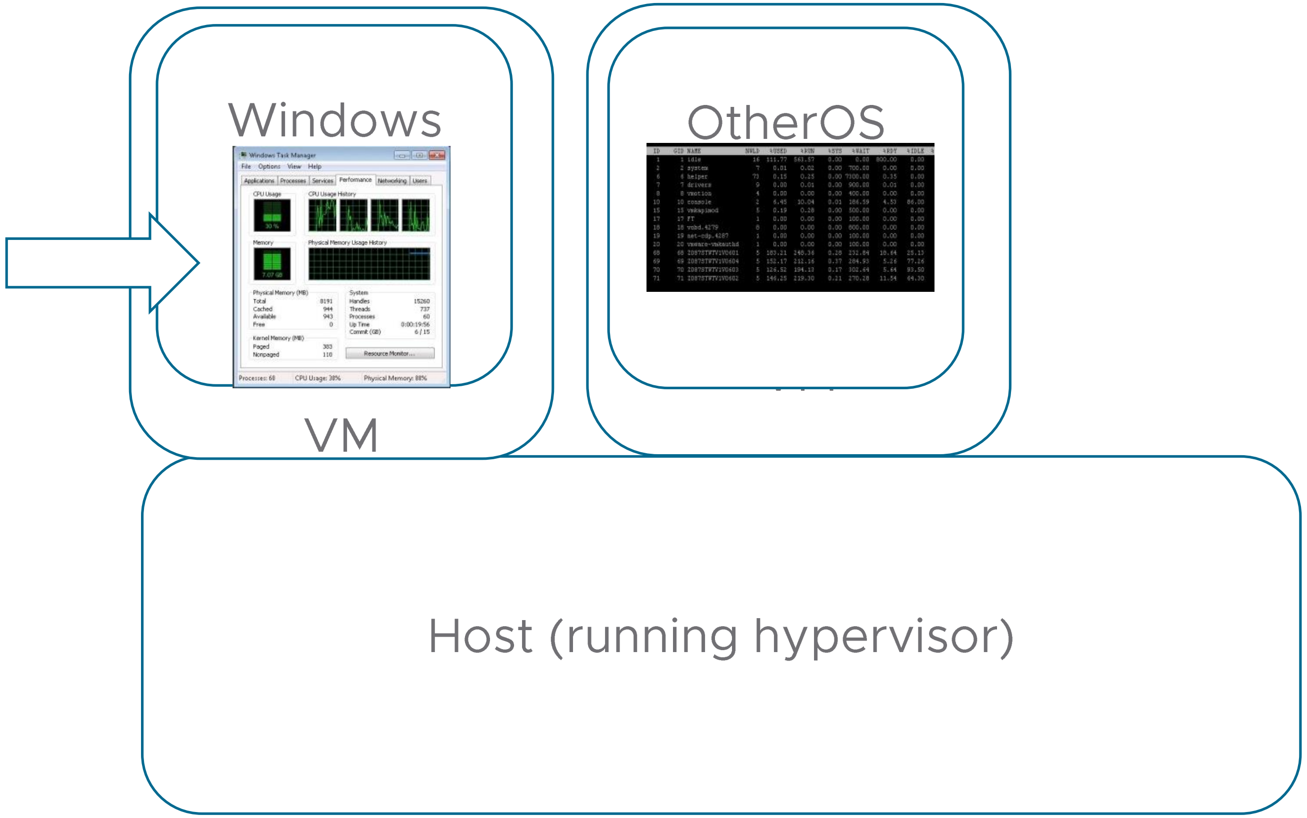
In-guest: memory usage high, but CPU is fine



Going beyond guest-level metrics

We looked in-guest.

What about interaction of this VM with other VMs?



Memory Primer

VMware ESX hypervisor balances memory of VMs, etc.

- **Page sharing** to reduce memory footprint of Virtual Machines
- **Ballooning** to relieve memory pressure in a graceful way
- **Host swapping** to relieve memory pressure when ballooning insufficient

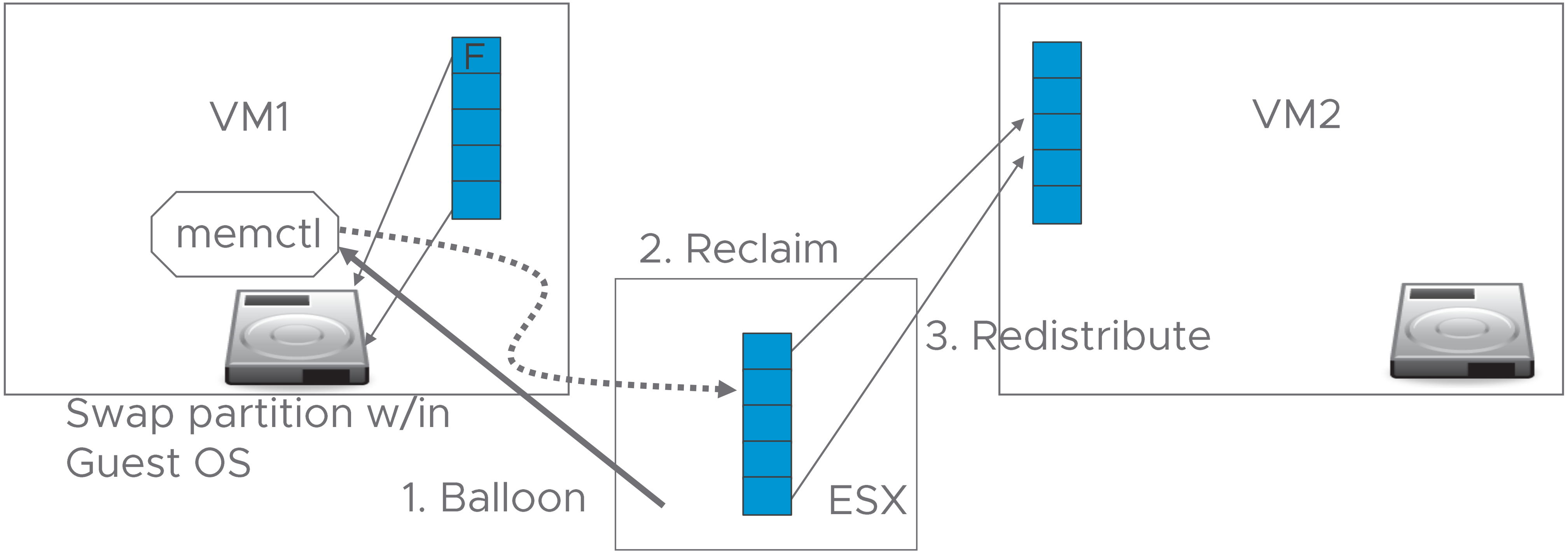
ESX allows overcommitment of memory

- Sum of configured memory sizes of virtual machines can be greater than physical memory if working sets fit

Ballooning vs. Swapping (1)

Ballooning: Memctl driver grabs pages and gives to ESX

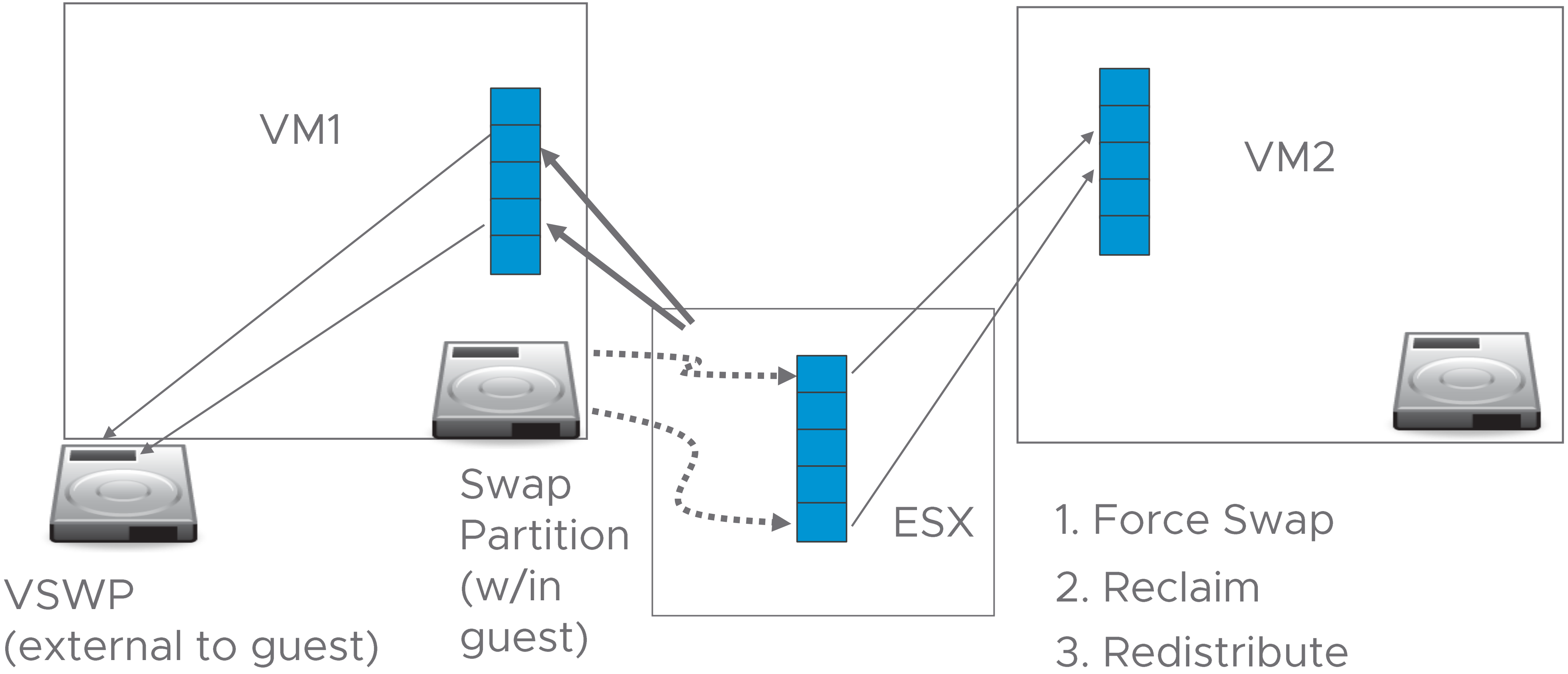
- Guest OS choose pages to give to memctl (**avoids "hot" pages if possible**): either free pages or pages to swap
  - Unused pages are given directly to memctl
  - Pages to be swapped are first written to swap partition within guest OS and then given to memctl



Ballooning vs. Swapping (2)

Swapping: ESX reclaims pages forcibly

- Guest doesn't pick pages…ESX may inadvertently pick "hot" pages (→possible VM performance implications)
- Pages written to VM swap file



Ballooning vs. Swapping: Bottom Line

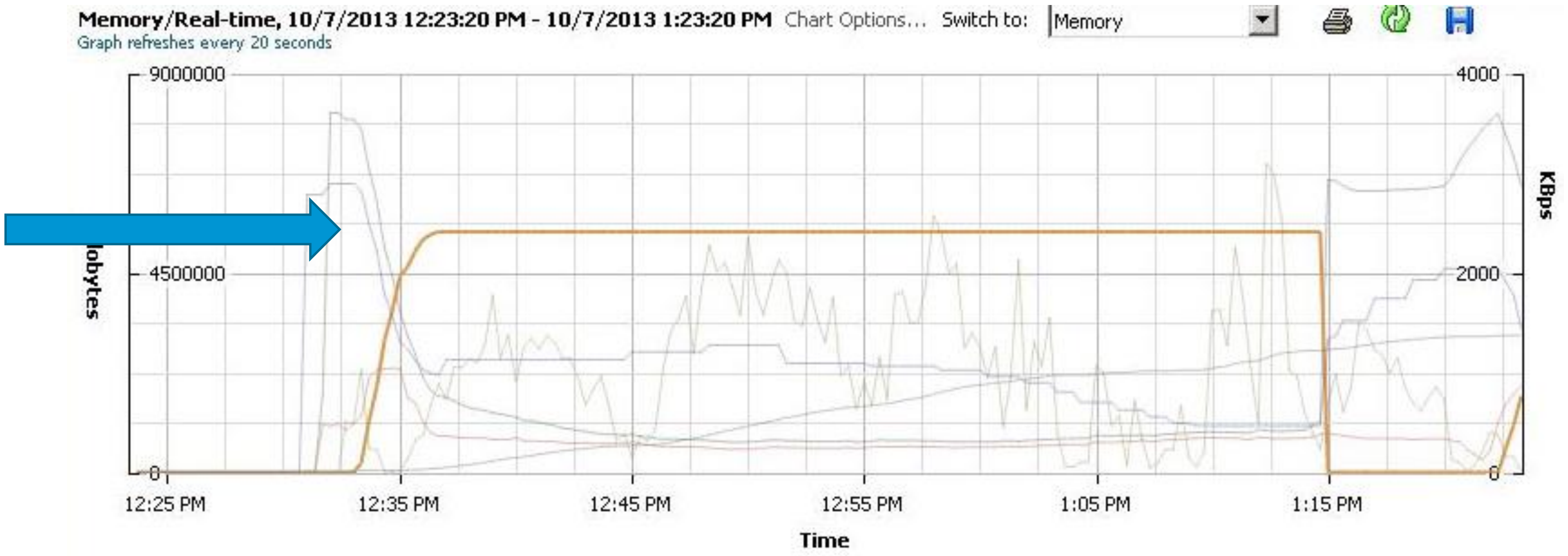
Ballooning may occur even when no memory pressure just to keep memory proportions under control

Ballooning is vastly preferably to swapping

- Guest can surrender unused/free pages
  - With host swapping, ESX cannot tell which pages are unused or free and may accidentally pick “hot” pages
- Even if balloon driver has to swap to satisfy the balloon request, guest chooses what to swap
  - Can avoid swapping “hot” pages within guest

Back to my VM: Let’s look at ballooning

VM is ballooning! It reaches its threshold…

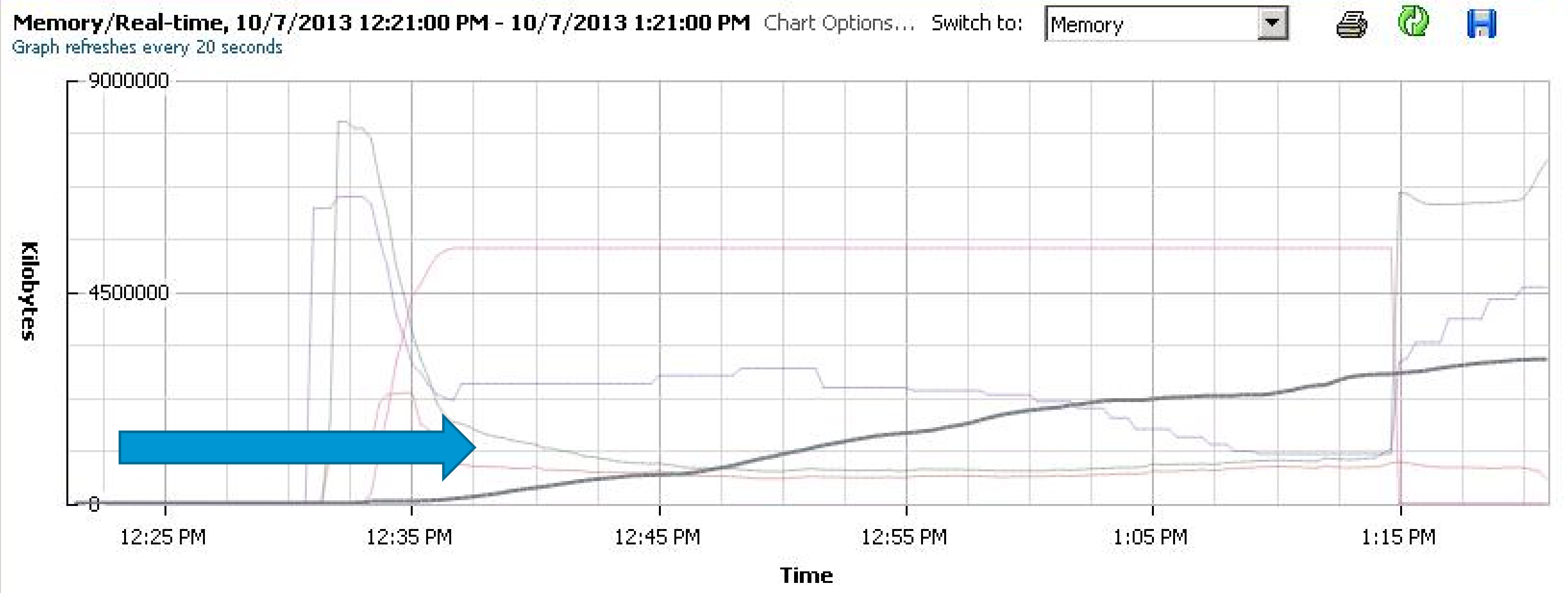


Performance Chart Legend

Key	Object	Measurement	Rollup	Units	Latest	Maximum	Minimum	Average
	Windows7-dev	Granted	Average	Kilobytes	6487340	8148800	0	2248651.6
	Windows7-dev	Consumed	Average	Kilobytes	1961788	2363108	0	765293.72
	Windows7-dev	Balloon	Average	Kilobytes	1730272	5452296	0	4209805.9
	Windows7-dev	Swap in	Average	Kilobytes	3103804	3103804	0	1578085.8
	Windows7-dev	Active	Average	Kilobytes	3262584	6543112	0	2743647.0
	Windows7-dev	Swap in rate	Average	KBps	17	3111	0	988.057

Swap-in

And then the VM starts to do host-level swap



Performance Chart Legend

Key	Object	Measurement	Rollup	Units	Latest	Maximum	Minimum	Average
	Windows7-dev	Granted	Average	Kilobytes	7319140	8148800	0	2004035.9
	Windows7-dev	Consumed	Average	Kilobytes	516100	2363108	0	750484.02
	Windows7-dev	Balloon	Average	Kilobytes	0	5452296	0	4382770.1
	Windows7-dev	Swap in	Average	Kilobytes	3075528	3075528	0	1507849.2
	Windows7-dev	Active	Average	Kilobytes	4613732	6543112	0	2671805.8

Host-level swap impacts performance…

Fine-grained metrics

Check if other VMs are encountering same issue

GID	NAME	MEMSZ	GRANT	SZTGT	TCHD	TCHD_W	SWCUR	SWTGT
8283283	VCVA-5.5-bld236	16384.00	14412.00	15374.60	491.52	491.52	0.00	0.00
8169268	win2k8-r2-prod2	16384.00	16384.00	16501.00	491.52	491.52	0.00	0.00
11021155	vzSim5.1-for-VC	16384.00	16376.00	16061.40	163.84	0.00	0.00	0.00
1961387	SLES11	8192.00	6388.00	7097.34	0.00	0.00	0.00	0.00
10967525	Windows7-dev	8192.00	1142.39	1117.20	745.55	430.12	1590.44	1671.02

(esxtop, a top-like utility specifically for ESX hosts)

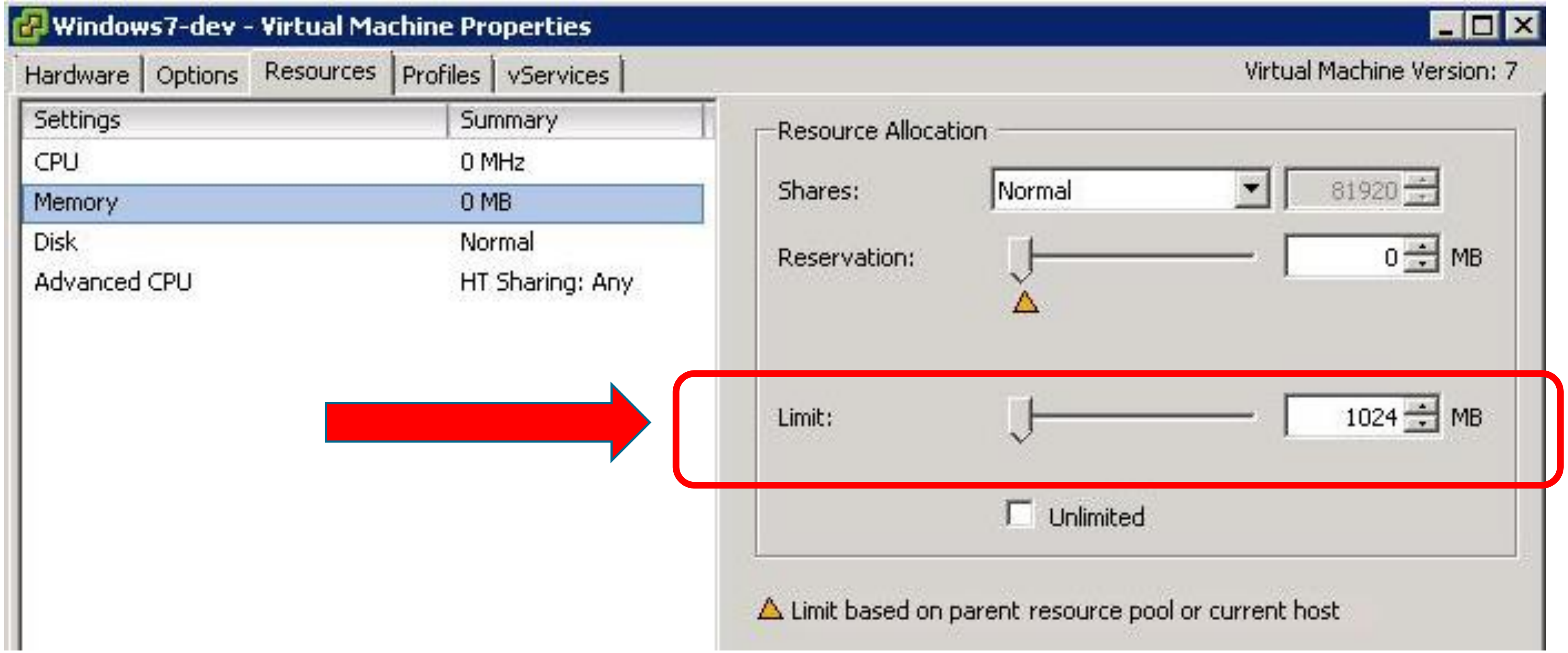
No other VMs are hitting host-level swapping…

Hmm.

Oh, wait!

Accidentally set limit on VM!

If you set a limit on a VM, it cannot exceed the limit



➔ We configured the VM with 8GB RAM, *but set a limit of 1GB!*

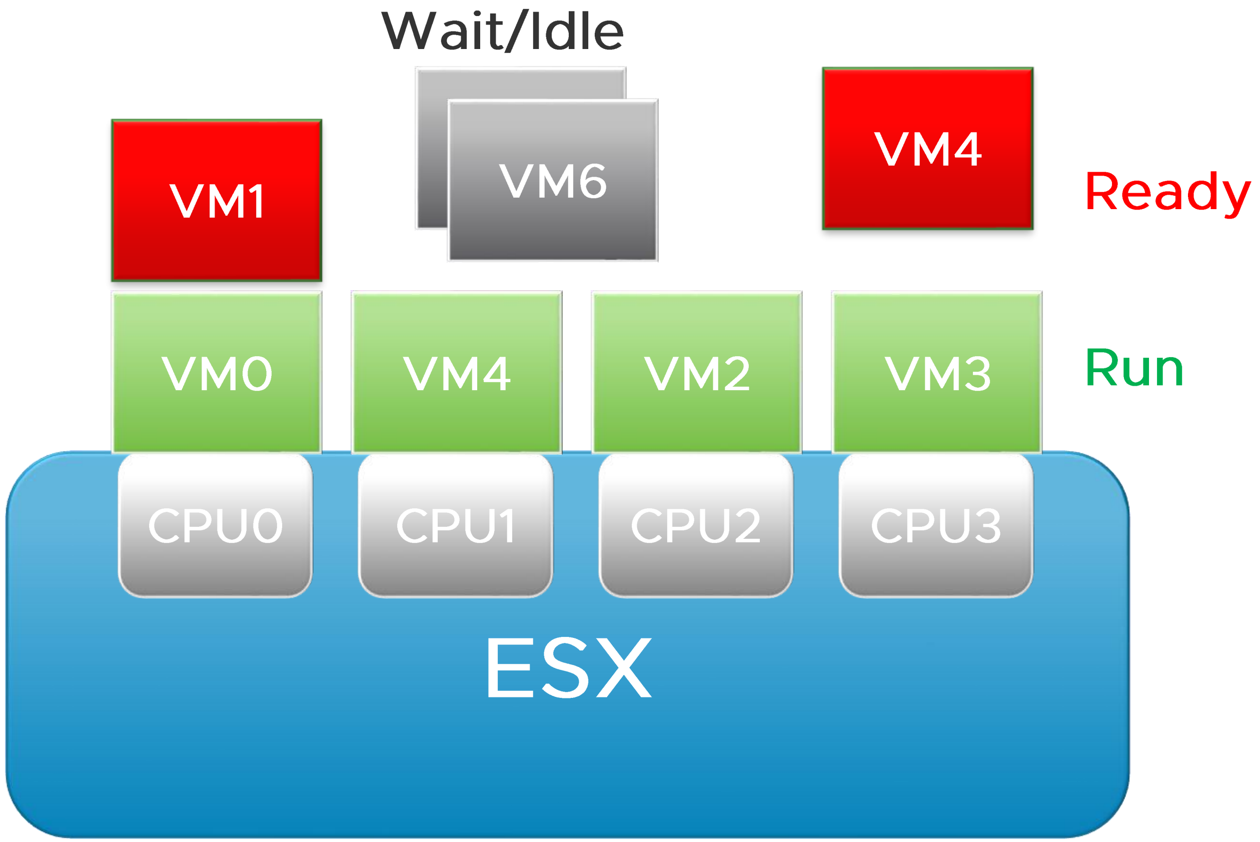
(btw., this was because I accidentally cloned a VM with a limit…)

(note: our tools track LIMIT, but I didn’t show it on previous slides)

Understanding metrics, part 2

CPU

Hypervisor CPU Scheduling



Run (accumulating used time)
Ready (wants to run, no physical CPU available)
Wait: blocked on I/O or voluntarily descheduled

A customer problem…

Problem

- Customer Performs a Load Test: keeps attaching clients to a server
- At some point, CPU is NOT saturated, but latency starts to degrade
- At some point, client is unusable
- Why?

“Oh yeah, it’s a disk problem…”

CPU Usage Increases…

Uh-oh! Disk Latencies go over a cliff!

Hmm. Not So Fast!!!

Problem:

Yes, Disk Latency gets worse at 4pm. (btw…due to swapping)

However, Application latency gets worse at 3:30pm!

What’s going on from 3:30pm to 4pm?

Looking at a different chart…

ID	GID	NAME	NWLD	%USED	%RUN	%SYS	%WAIT	%RDY	%IDLE	%
1	1	idle	16	111.77	563.57	0.00	0.00	800.00	0.00	
2	2	system	7	0.01	0.02	0.00	700.00	0.00	0.00	
6	6	helper	73	0.15	0.25	0.00	7300.00	0.35	0.00	
7	7	drivers	9	0.00	0.01	0.00	900.00	0.01	0.00	
8	8	vmotion	4	0.00	0.00	0.00	400.00	0.00	0.00	
10	10	console	2	6.45	10.04	0.01	186.59	4.53	86.00	
15	15	vmkapimod	5	0.19	0.28	0.00	500.00	0.00	0.00	
17	17	FT	1	0.00	0.00	0.00	100.00	0.00	0.00	
18	18	vobd.4279	8	0.00	0.00	0.00	800.00	0.00	0.00	
19	19	net-cdp.4287	1	0.00	0.00	0.00	100.00	0.00	0.00	
20	20	vmware-vmkauthd	1	0.00	0.00	0.00	100.00	0.00	0.00	
68	68	vm1	5	183.21	248.36	0.28	232.84	18.64	25.13	
69	69	vm2	5	152.17	212.16	0.37	284.93	5.26	77.26	
70	70	vm3	5	126.52	194.13	0.17	302.64	5.64	93.50	
71	71	vm4	5	146.25	219.30	0.21	270.28	11.54	64.30	

%Used? %Run? What's the difference?

%used: normalized to base clock frequency

%run: normalized to clock frequency *while VM is running…*

%run > %used: Power Management is kicking in…

In this case, turn off power management→latency problems go away

End-to-End Performance

Interactions between hypervisors and guests

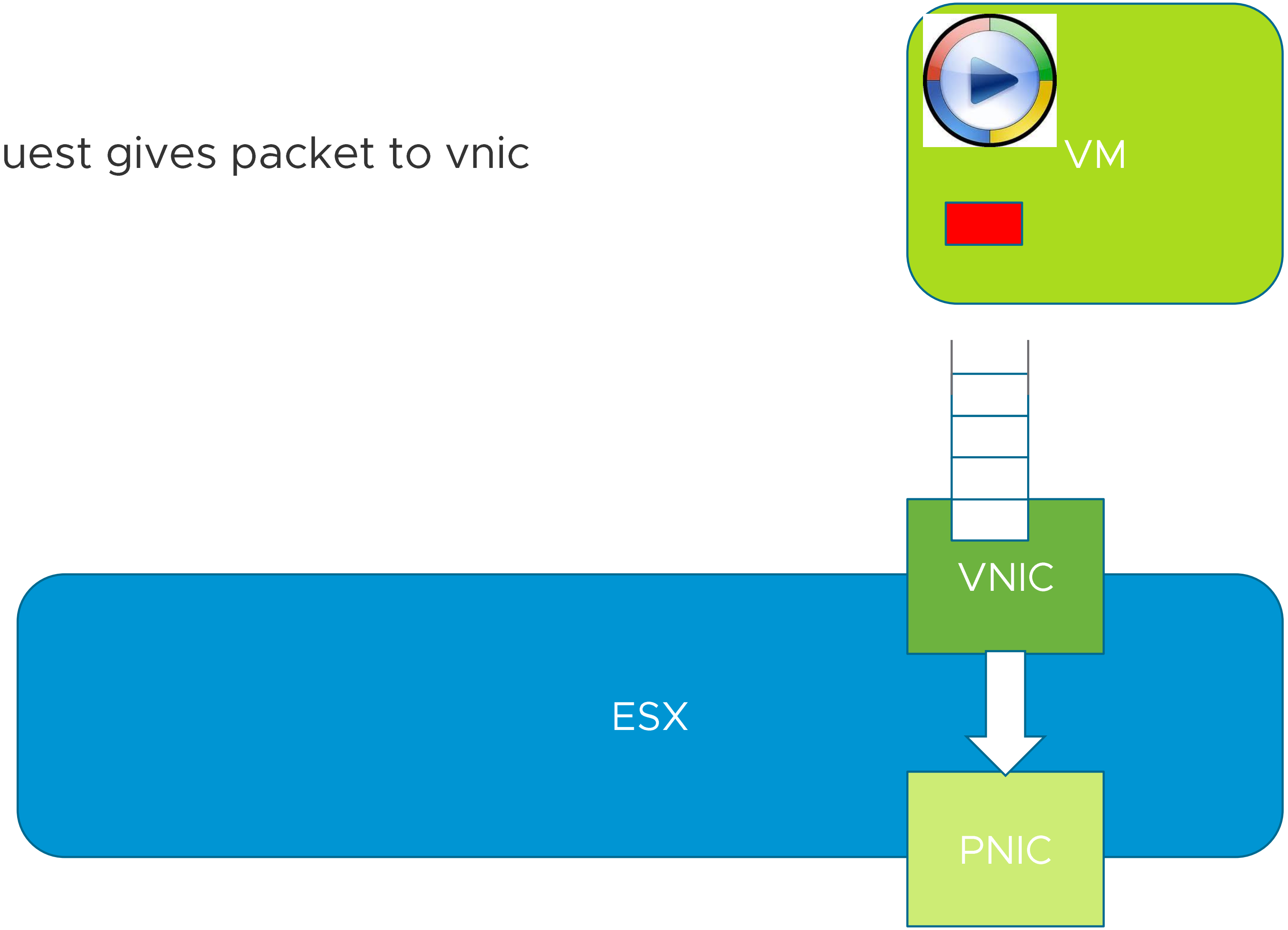
Viewing a Video Remotely: Jittery Experience

- interrupt-coalescing-nofix.mov

- Observation: bimodal latencies in 3D graphics workload
  - Needs 80Mbps at peak
  - When it reached 80Mbps at peak, dropped down to 30Mbps
  - Went back up to 80Mbps
  - Dropped to 30Mbps
  - Repeat…

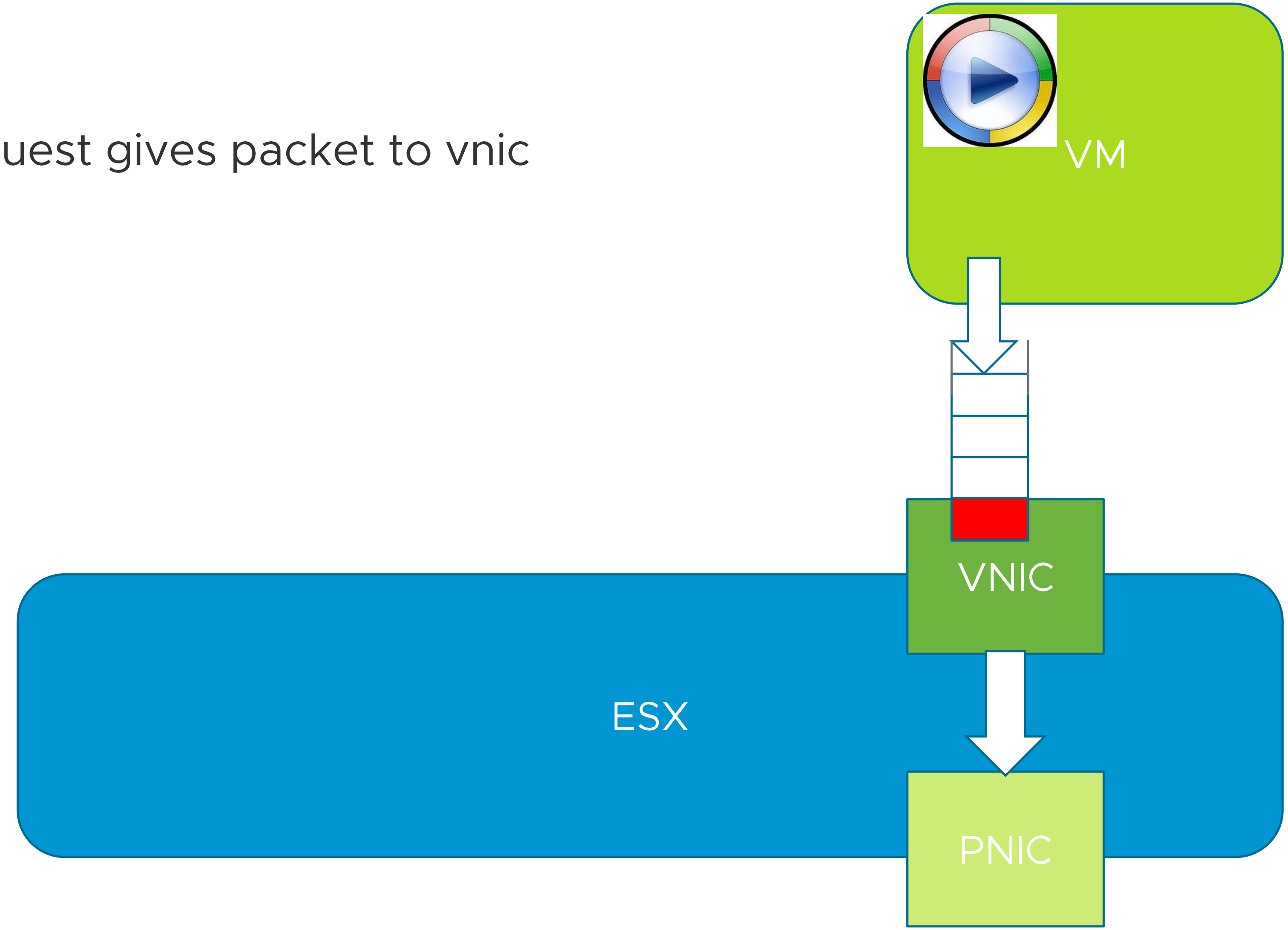
Packet transmission in virtualized environment

Guest gives packet to vnic



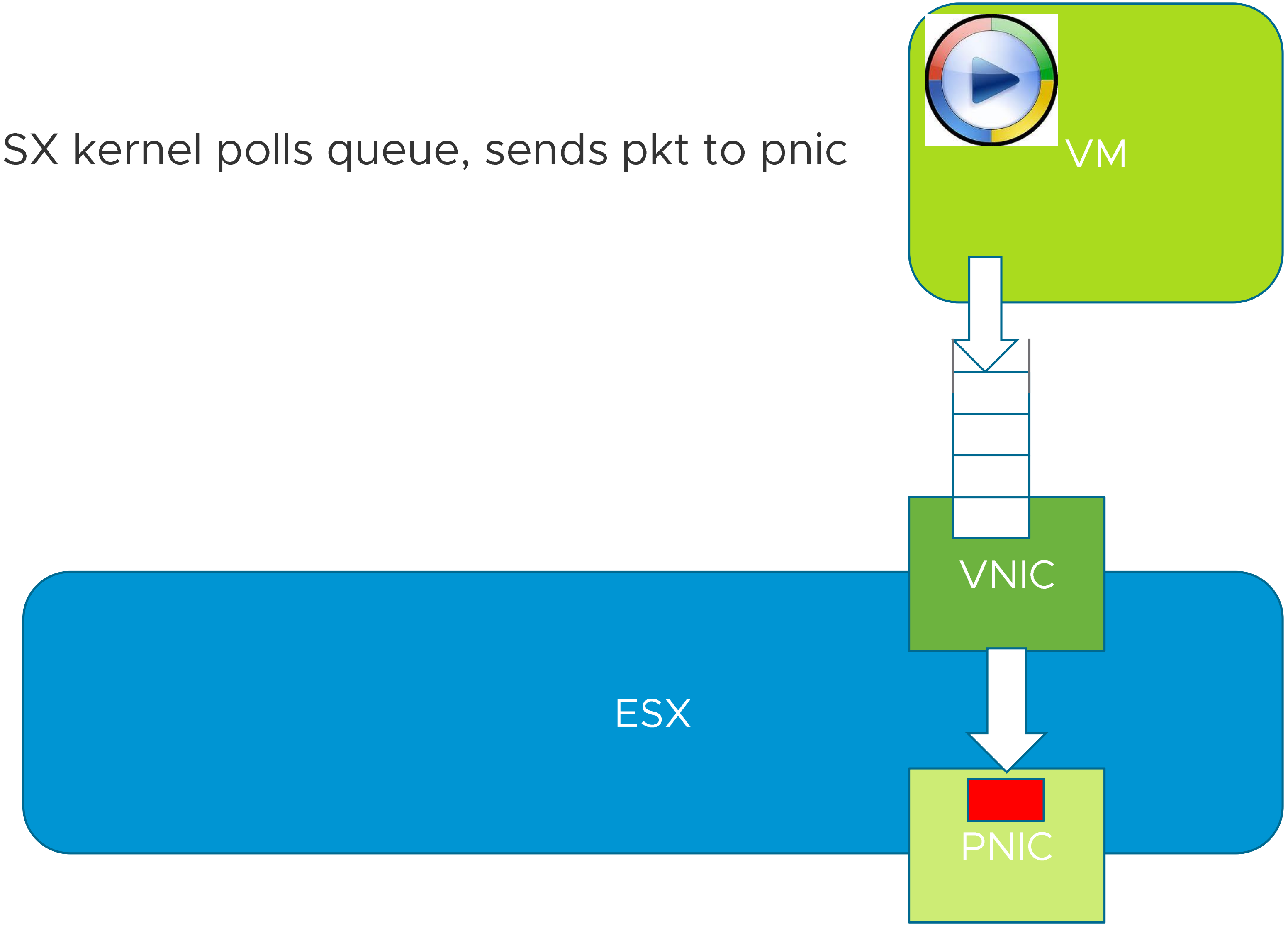
Packet transmission in virtualized environment

Guest gives packet to vnic



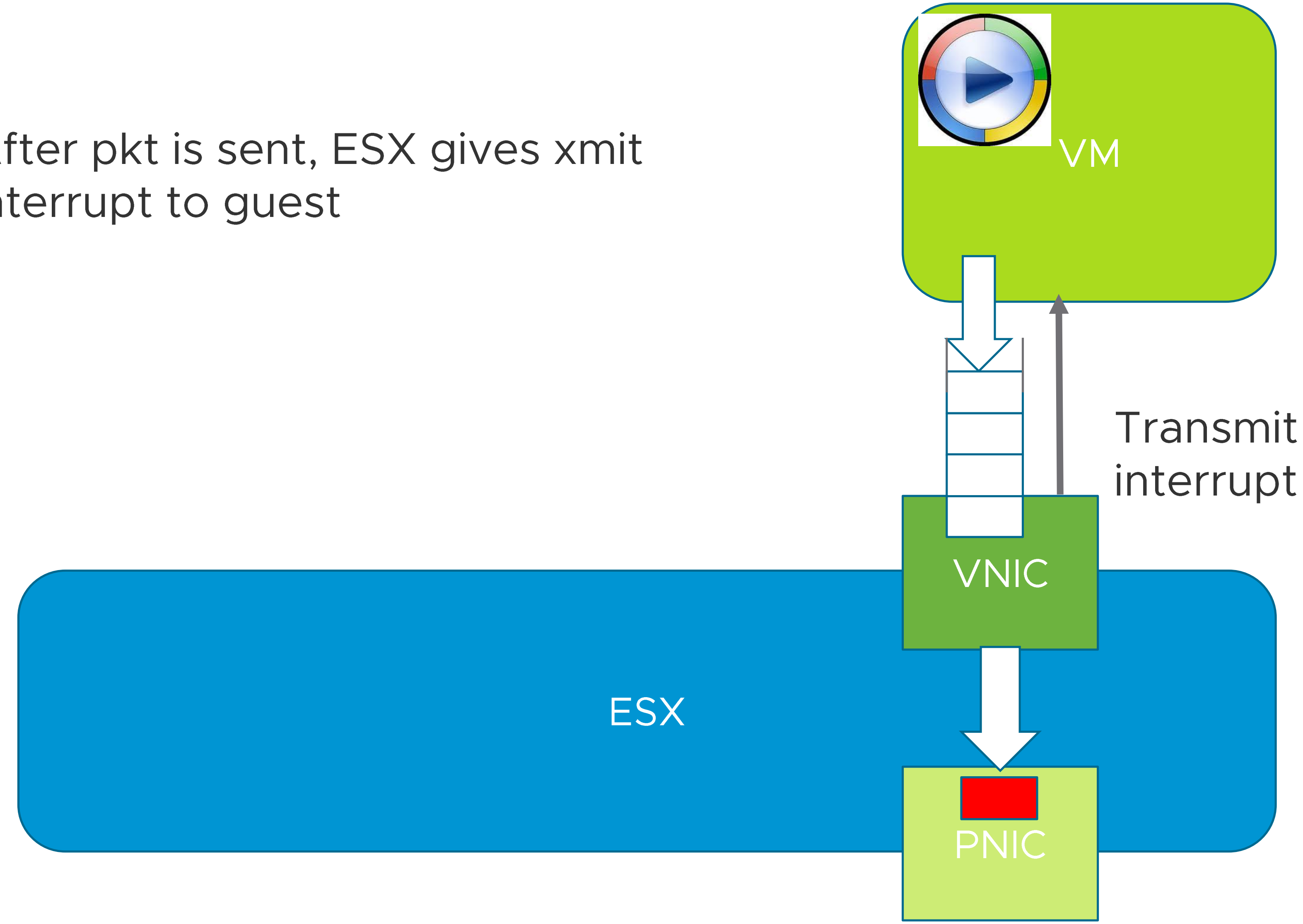
Packet transmission in virtualized environment

ESX kernel polls queue, sends pkt to pnic



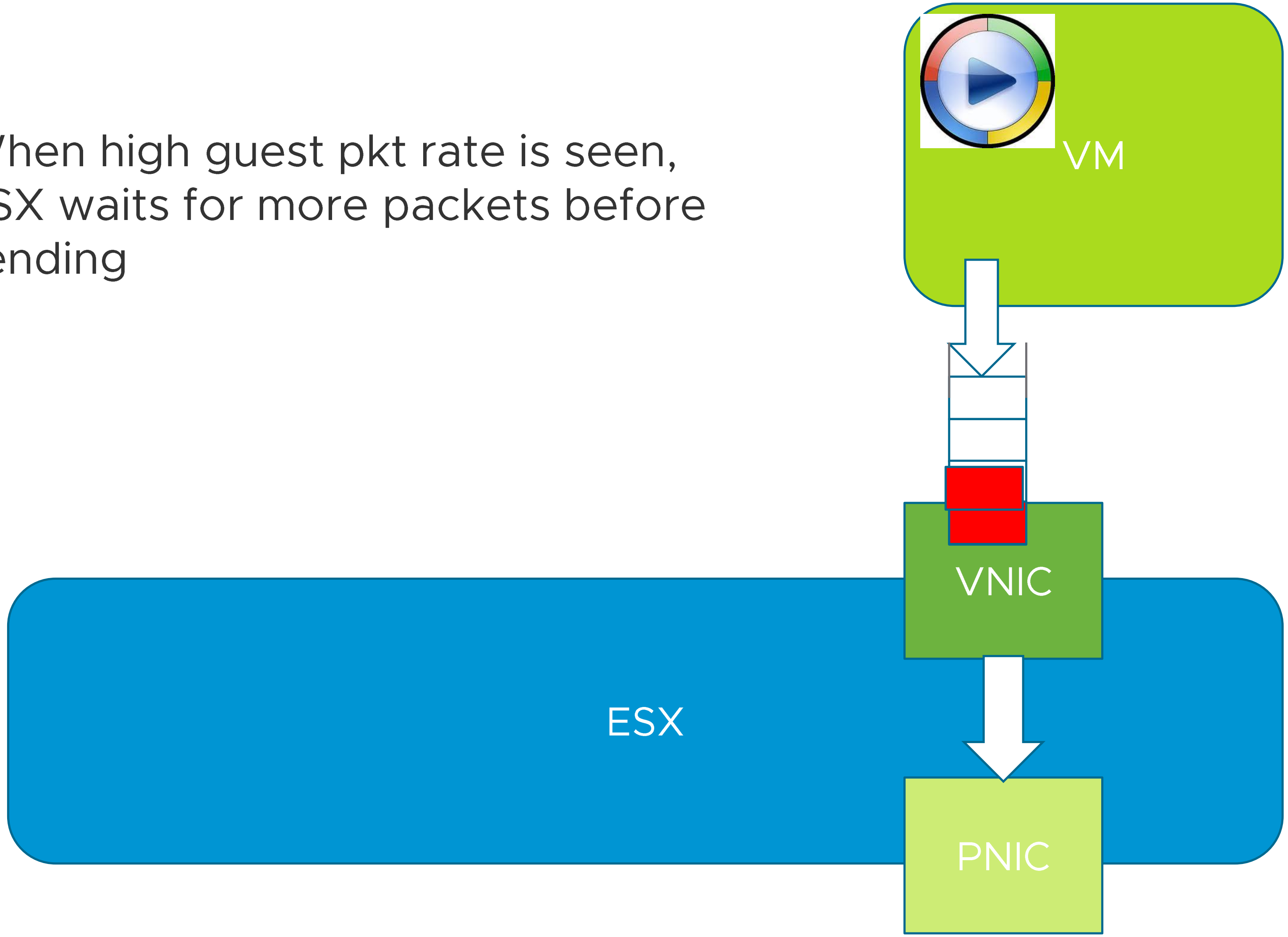
Packet transmission in virtualized environment

After pkt is sent, ESX gives xmit interrupt to guest



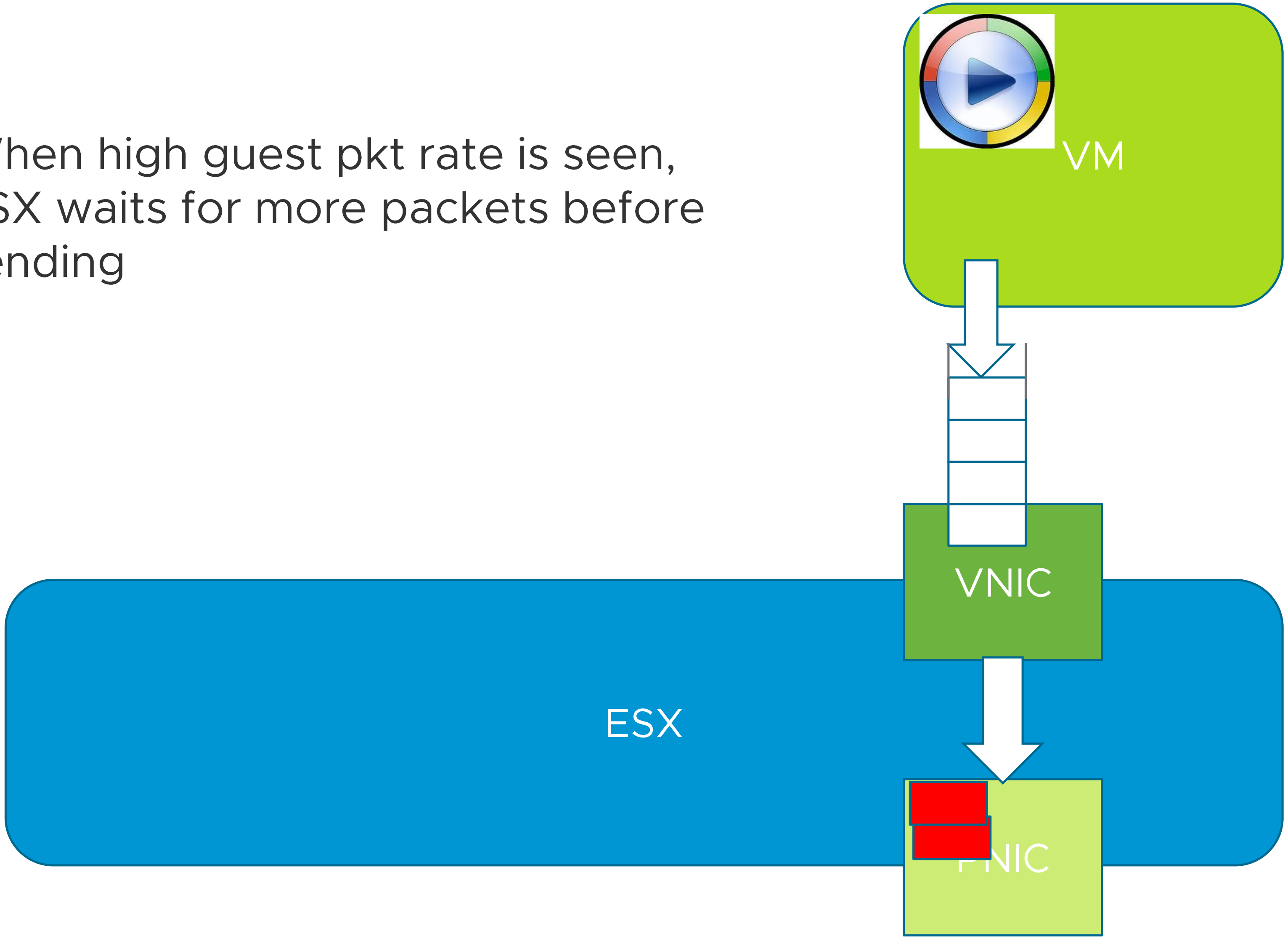
Coalescing

When high guest pkt rate is seen, ESX waits for more packets before sending



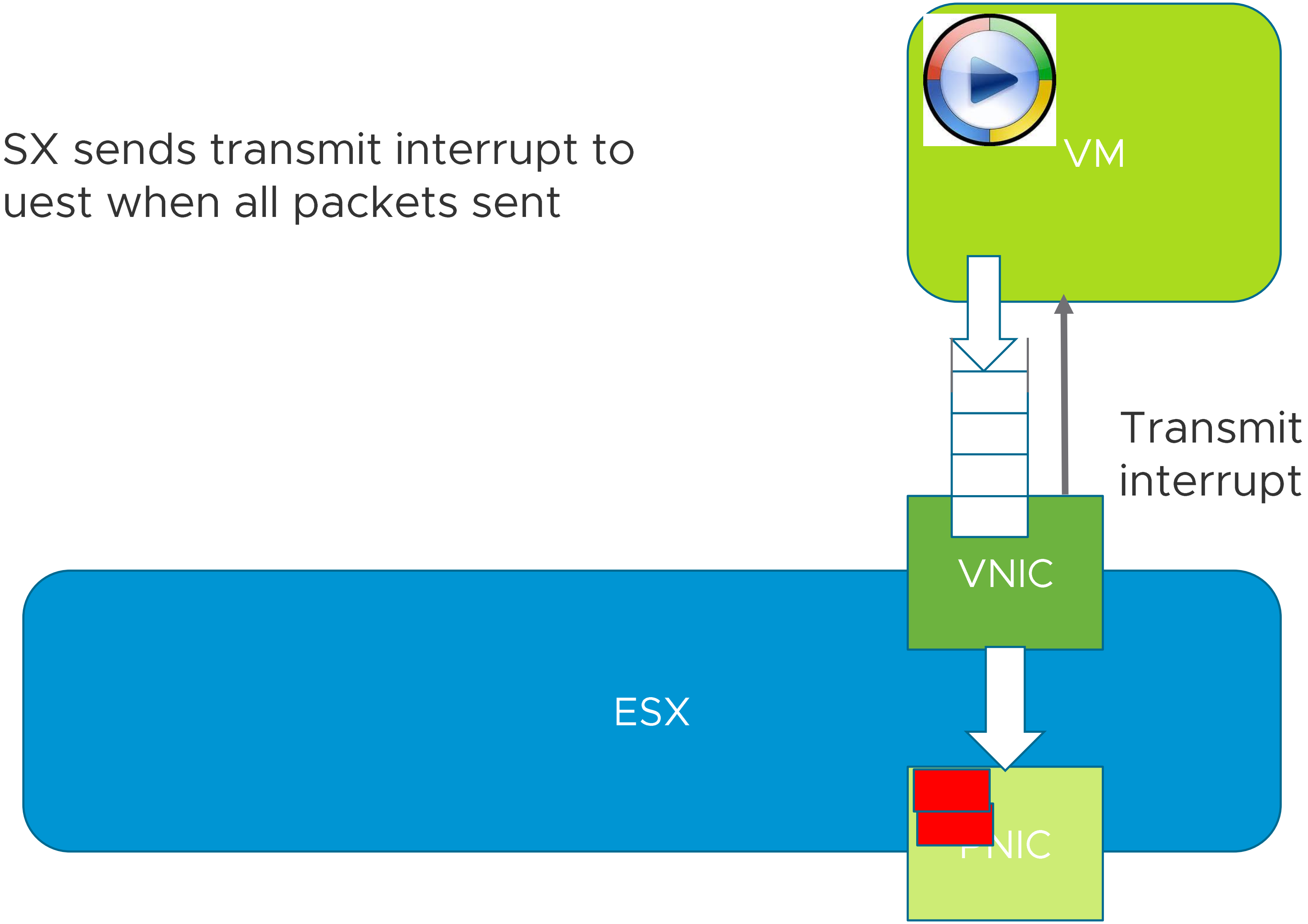
Coalescing

When high guest pkt rate is seen, ESX waits for more packets before sending



Coalescing

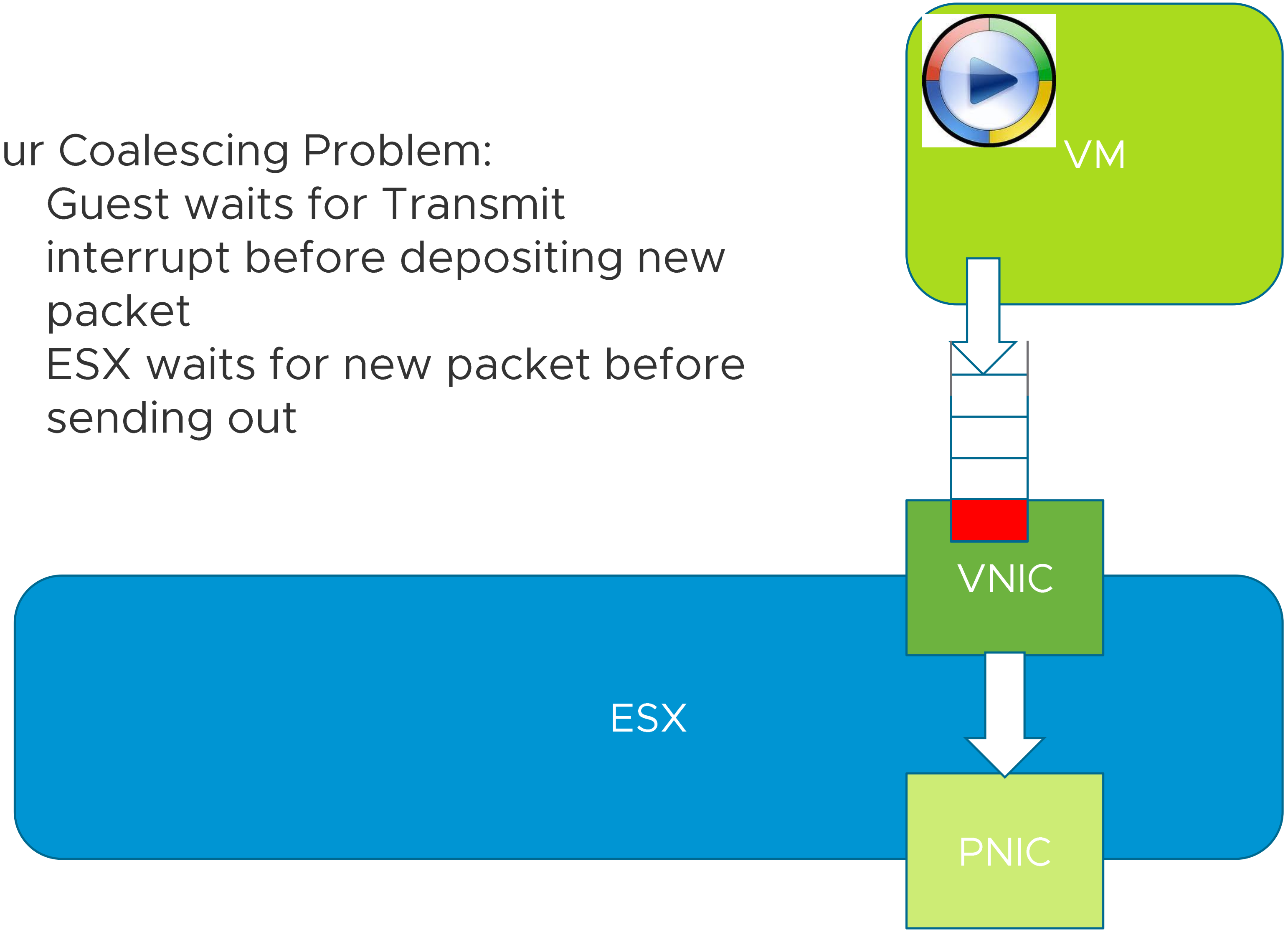
ESX sends transmit interrupt to guest when all packets sent



Coalescing in guest causing issues

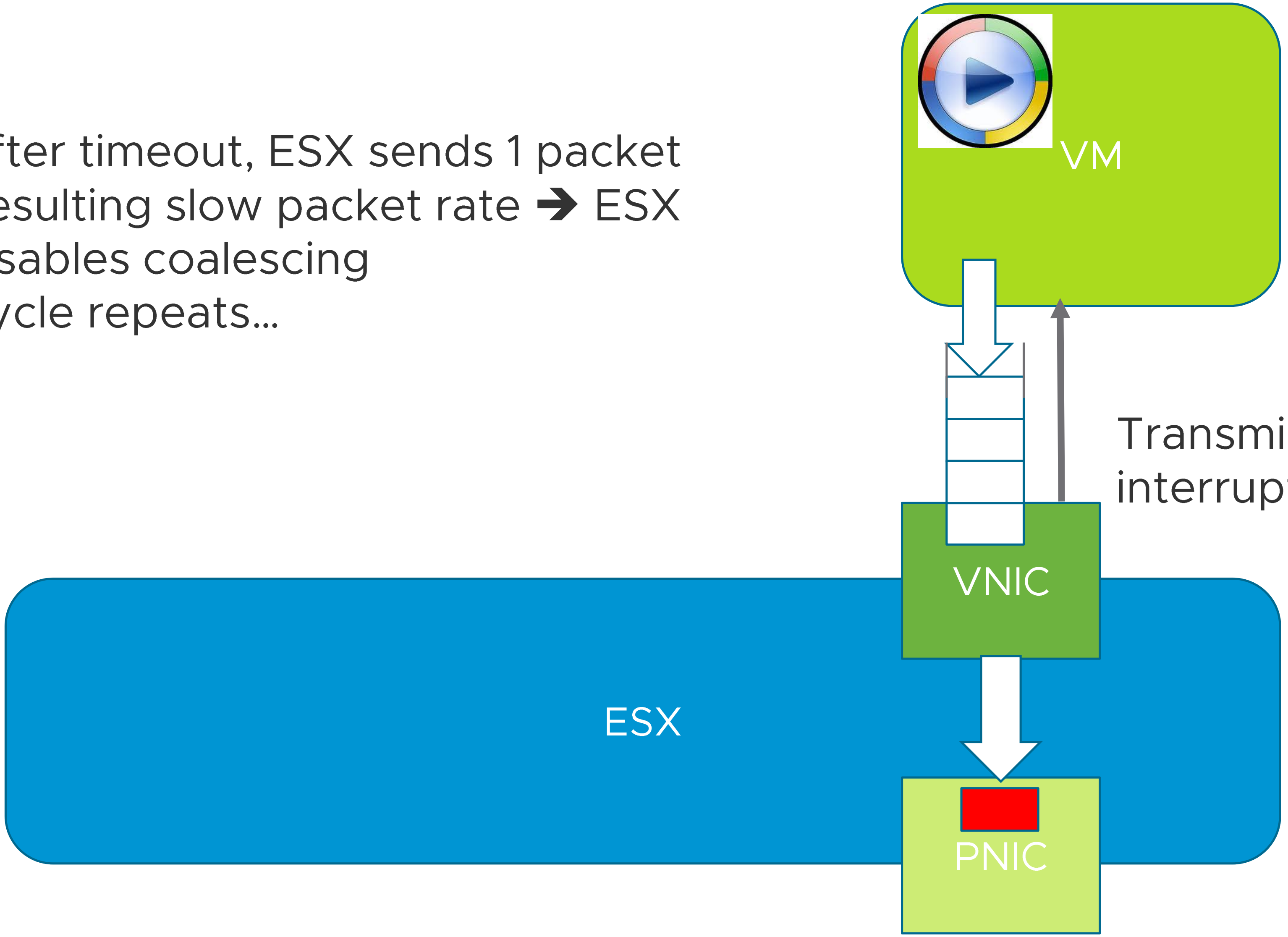
Our Coalescing Problem:

1. Guest waits for Transmit interrupt before depositing new packet
2. ESX waits for new packet before sending out



Coalescing and Windows

After timeout, ESX sends 1 packet
Resulting slow packet rate ➔ ESX disables coalescing
Cycle repeats…



Video playback in Windows: Why Oscillation in Latency?

- Desired Behavior
  - Guest sends packet by giving data to vmnic
  - Hypervisor polls receive queue
  - When packet detected, hypervisor sends packet
  - Hypervisor sends transmit interrupt to guest (packet has been delivered)

- Actual Behavior
  - Hypervisor interrupt coalescing kicks in at high packet rate
  - Guest would not send packet until it received transmit interrupt
  - Both sides wait, timeout in hypervisor, interrupts get sent ➔ drop to 30Mbps
  - Packet rate drops, interrupt coalescing disabled ➔ achieve 80Mbps

Fix for Oscillation in Latency

Fix:

- Known issue in Windows for certain packet sizes
- Disable Windows registry to avoid waiting for transmit interrupt

interrupt-coalescing-withfix.mov

Microsoft KB article:

- http://support.microsoft.com/kb/235257

VMware KB article:

- http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=2040065

Bringing it all together

An interesting link

Performance anti-patterns

http://queue.acm.org/detail.cfm?id=1117403

Some examples:

- Fixing Performance at the end of the project
- Algorithmic antipathy: O(k) vs. O(n)
- Focusing on what you can see rather than the problem
  - Disk IO is high
    - Option 1 (BAD) Workload needs IO: tell customer to add more spindles
    - Option 2 (BETTER) Find source of IO and eliminate it if possible
- Not optimizing for the common case

Parting Thoughts

- Performance debugging is a system-wide exercise
- Don’t blindly optimize resources: take a broader view of architecture as well
- Don’t take down fences unless you know why they were put up
- Make the common case fast (but make sure it is also correct!)