Dynamic Adaptive Optimization: Recovering from Hardware Errors and Software Crashes in a Distributed Virtual Machine

Ike Nassi

(Technical collaborator: David P. Reed)

Adjunct Prof UCSC/CSE

formerly CTO and Founder, TidalScale
(TidalScale now lives at HPE)

TTI / Vanguard
Resilience and Reliability Conference
Chapel Hill, NC

March 5-7, 2024

Motivations for this talk

What if:

Recovery from ~90% of all hardware errors could be automatically and non-disruptively handled?

What if:

When the OS crashes *(and it will)* it can reboot itself automatically not in many minutes, but in seconds?

What if:

This can be accomplished without any special hardware?

What if:

No changes to ANY software were required?

Goals for this talk

Surprise:

All of this is possible

This talk address many failures, *both* hardware and software

Not Brand New, But to Review:

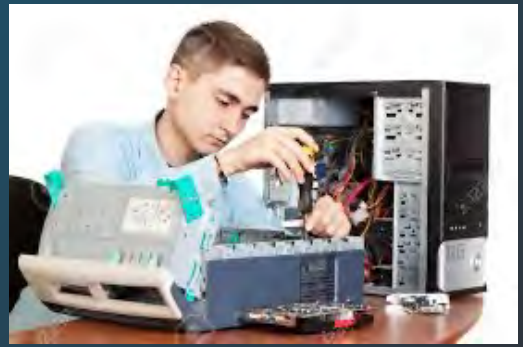
Hardware can fail. We all know this. But,

Most hardware failures can be mitigated using redundant hardware. We (also) know this. But,

We can now do this transparently with respect to application software that's running. We (sort of) know this, but people forget. But,

We can now do this transparently to systems software!

(You may have heard this from me before)



Operating Systems can FAIL!

```
[    0.590229]  ret_from_fork+0x35/0x40
[    0.590229] Modules linked in:
[    0.590229] ---[ end trace 23e2fda7cad73758 ]---
[    0.590229] RIP: 0010:__list_del_entry_valid.cold.1+0x34/0x4c
[    0.590229] Code: 84 4d bc e8 b8 bc ce ff 0f 0b 48 c7 c7 d0 84 4d bc e8 aa bc
 ce ff 0f 0b 48 89 f2 48 89 fe 48 c7 c7 90 84 4d bc e8 96 bc ce ff <0f> 0b 48 89
 fe 48 c7 c7 58 84 4d bc e8 85 bc ce ff 0f 0b 90 90 90
[    0.590229] RSP: 0018:ffffb2fe819fbdc8 EFLAGS: 00010046
[    0.590229] RAX: 0000000000000054 RBX: 0000000000000002 RCX: ffffffffbc65b188
[    0.590229] RDX: 0000000000000000 RSI: 0000000000000092 RDI: 0000000000000046
[    0.590229] RBP: ffffb2fe819fbe78 R08: 00000000000003fb R09: ffff9fde400b8280
[    0.590229] R10: ffffffffbb88c2c0 R11: 2e6e6f6974707572 R12: 0000000000000000
[    0.590229] R13: ffff9fe076d96168 R14: 0000000000000001 R15: ffff9fe076d96160
[    0.590229] FS:  0000000000000000(0000) GS:ffff9fe077a80000(0000) knlGS:00000
00000000000
[    0.590229] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[    0.590229] CR2: 00007f424769f000 CR3: 0000000056c0a001 CR4: 00000000007606e0
[    0.590229] DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
[    0.590229] DR3: 0000000000000000 DR6: 00000000fffe0ff0 DR7: 0000000000000400
[    0.590229] PKRU: 55555554
[    0.590229] Kernel panic - not syncing: Fatal exception
[    0.590229] Kernel Offset: 0x3a400000 from 0xffffffff81000000 (relocation ran
ge: 0xffffffff80000000-0xffffffffbfffffff)
[    0.590229] ---[ end Kernel panic - not syncing: Fatal exception ]---
```

Operating Systems can FAIL!

Thankfully, it's rare. We know this. But,

When an OS fails, it takes quite a lot of time to bring it back online.

Because the OS tests and clears memory, and

Reloads a lot of data from storage. But,

How much of do we really need to do, and how quickly can we do it? Can we do it safely?

Answers: Not much, very quickly, and yes, safely.

The Key Insight: Use A Distributed Virtual Machine

Create a highly reliable and resilient distributed virtual machine architecture (DVM)

Some said it couldn’t be done.

Alan Kay (strong believer in software virtual machines):

> “Ike: You’re only realizing this *NOW?*”

Why use a DVM?

Dynamically Scale, up or down

Simplify – No software modifications

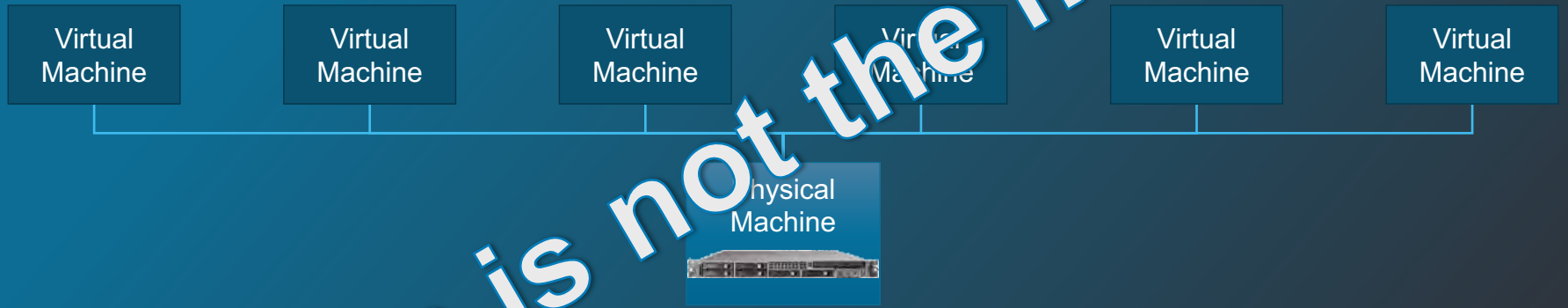
Dynamically Self Optimizing using Machine Learning

No New Infrastructure

Highly Reliable

Traditional virtualization…

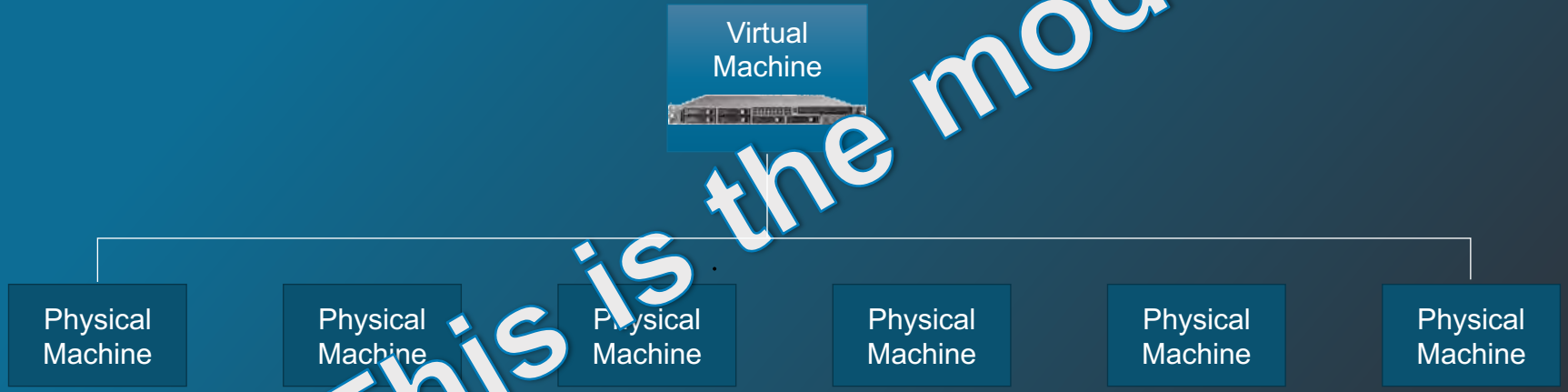
Virtual Machines sharing a server



This is not the model!

Distributed Virtualization Machine

Virtual Machine sharing multiple servers

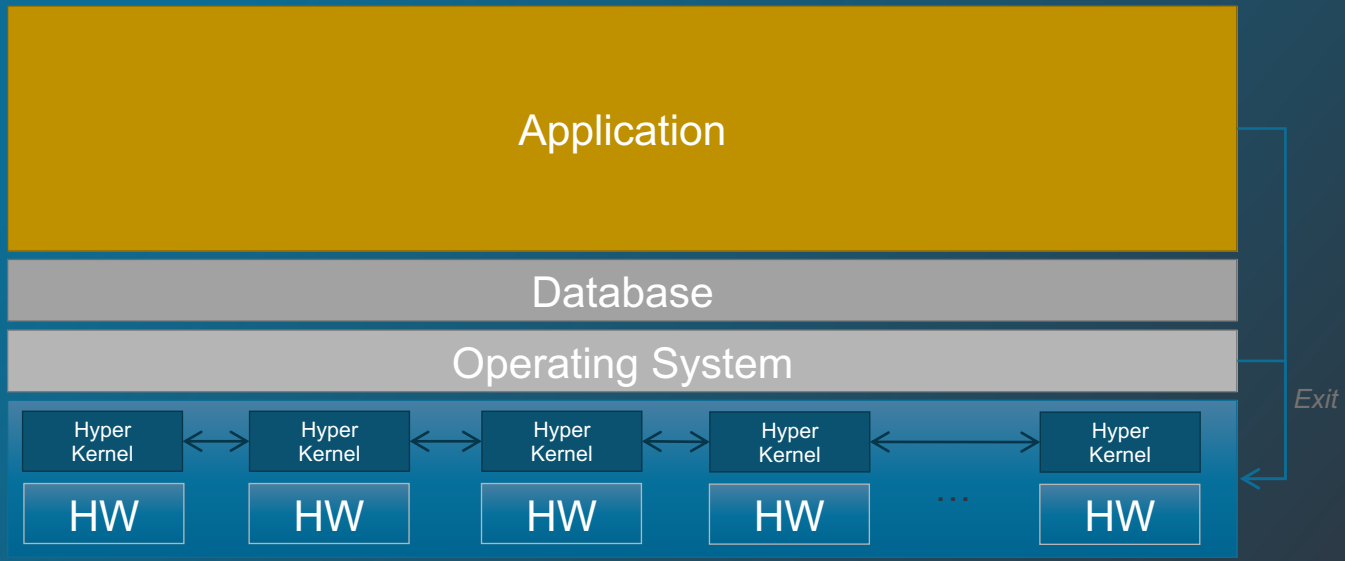


This is the model

What is a DVM?

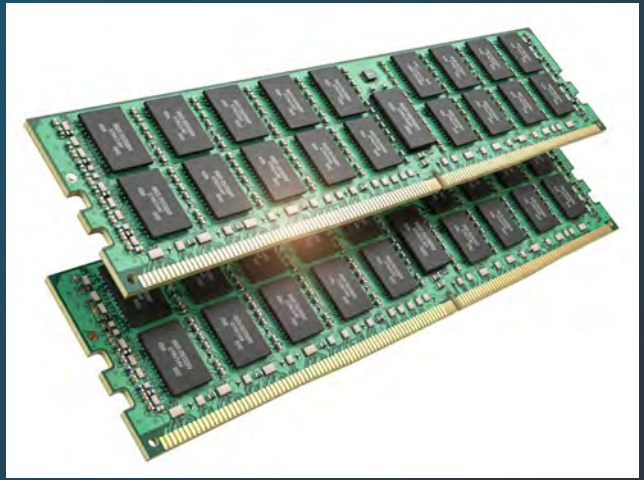
- A DVM aggregates an entire cluster, i.e. all the processors, all the memory, all the disks, and all the local storage and presents the entire set as a single virtual server, e.g., a very large motherboard
- This virtual machine runs across a cluster of closely cooperating, dedicated, homogeneous, off-the-shelf (commodity) servers, connected by standard LAN
- Virtual resources (processors, memory, I/O) can *migrate from server to server and are then automatically rebound to physical resources.*
- The DVM boots a single standard operating system (RedHat, SUSE, Oracle Linux, Centos, etc.)
- *Implication: No changes to any OS, compiler, libraries or applications are required*

What it looks like



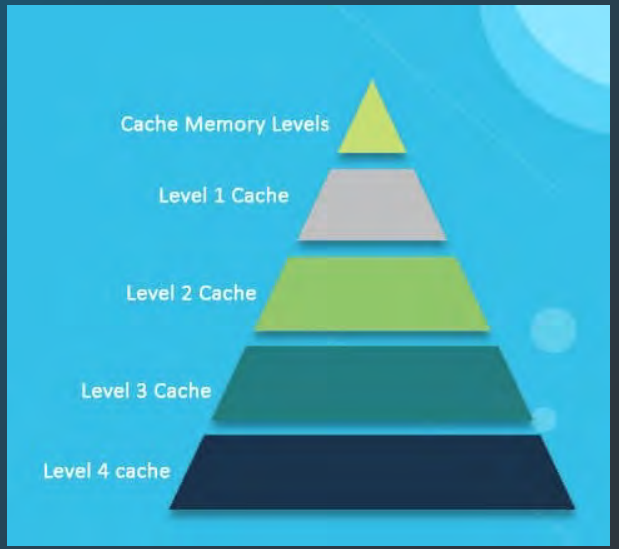
Thanks For the Memories – but what exactly do we mean by *memory*?

- An application process sees *virtual memory,* **NOT** physical memory.
- *Real physical memory* (RPM) is packaged on memory chips, often on dual inline memory modules (DIMMs). This is what you buy when you buy memory.
- RPM DIMMS are installed on physical server motherboards.
- Is this what the OS sees? *Not necessarily.*
- The OS sees what the DVM want it to see, and the DVM can tightly control this. We call the memory that's visible to the OS *Guest Physical Memory (GPM).*
- The DVM controls the mapping between GPM and RPM.



The DVM Becomes an *All-Cache Architecture*

- Modern caches are hierarchical: L1, L2, L3. DVM technology effectively introduces a software-based L4.
- All the memory on the motherboard is treated like any other cache (L4) except for one thing: the L4 cache is all the memory on the motherboards and is therefore HUGE, generally ~0.5TB - 2TB *per server times ~5 servers = ~10TB! It becomes a cache-only system.*

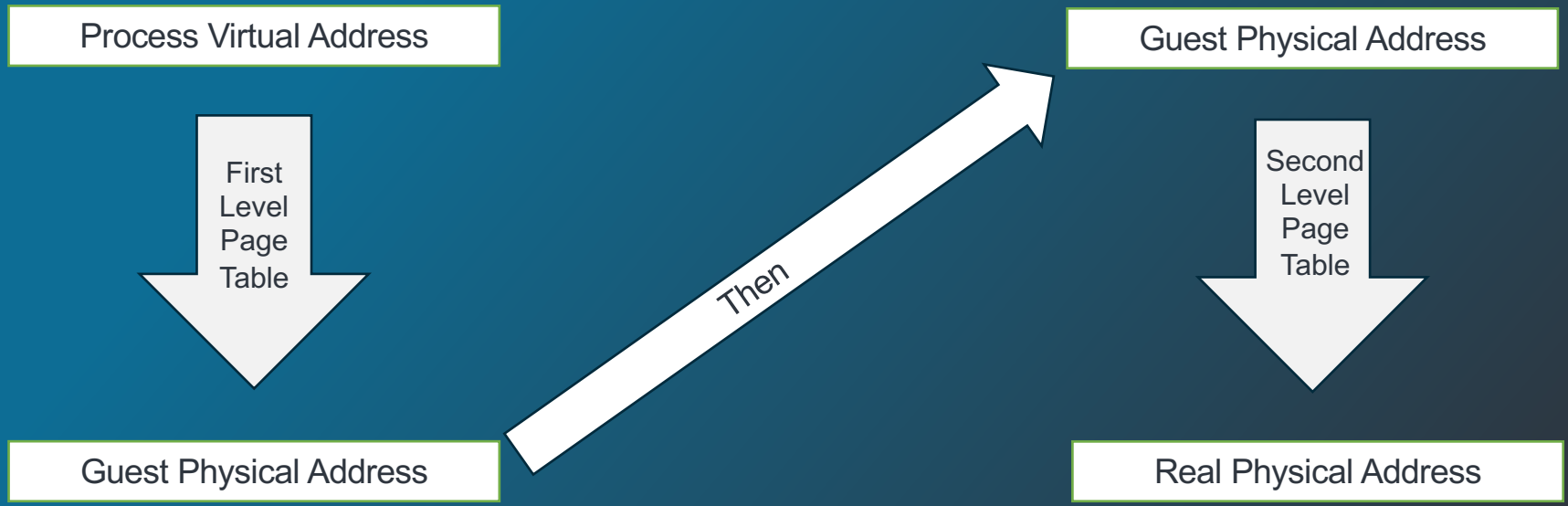


Page Mapping

By setting up the page tables in advance, the OS handles *the first level page table mapping* invisibly to the process as it does today

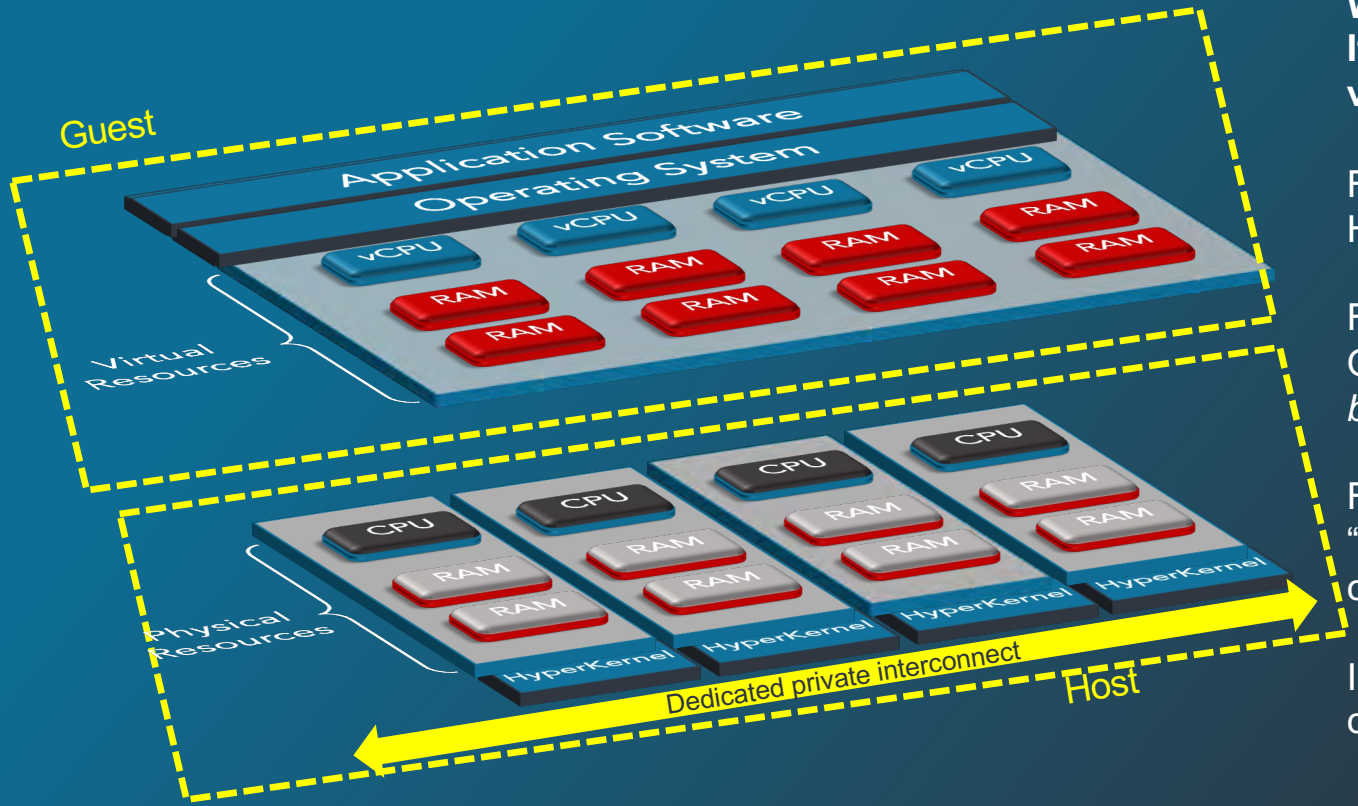
By setting up the page tables in advance, the DVM handles *the second level page table mapping* invisibly to the OS

Multi-level Dynamic Address Translation



Therefore, page address mapping is handled by using two levels of dynamic address translation supported in most modern microprocessors. A process's VM maps to GPM, which in turn maps to RPM *at hardware speed.*

Rethinking DRAM from the SW Point of View



Where is the DRAM? Answer: It depends on your point of view:

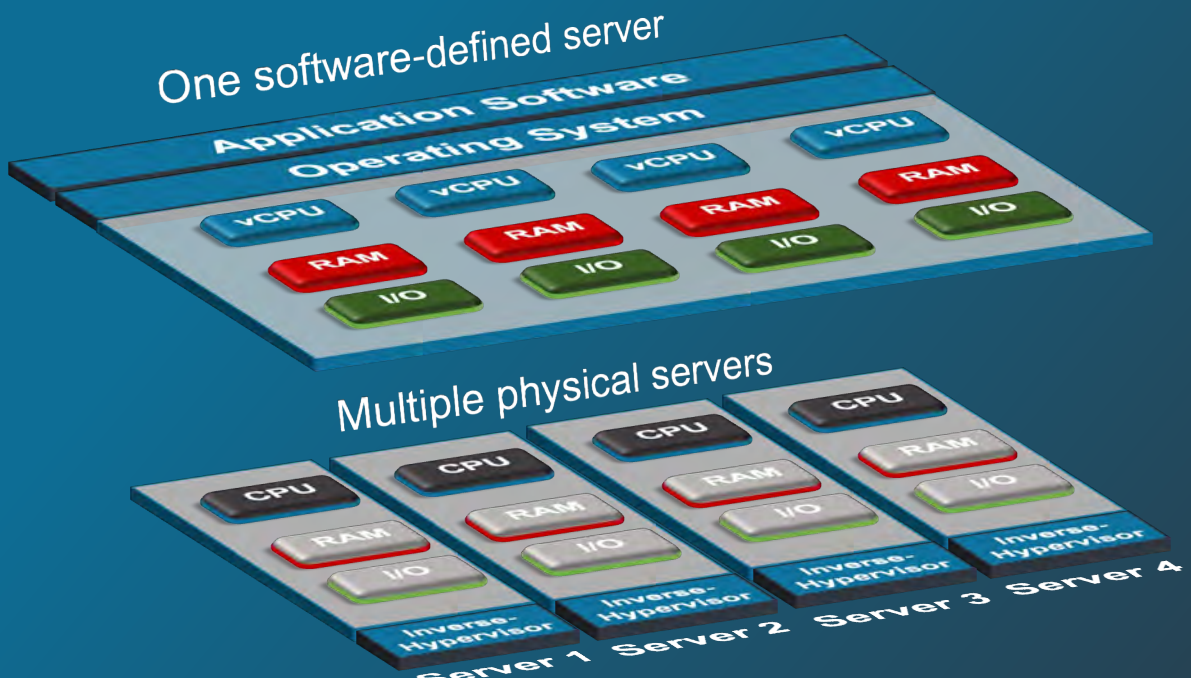
From the host point of view (i.e. HW), the physical motherboard.

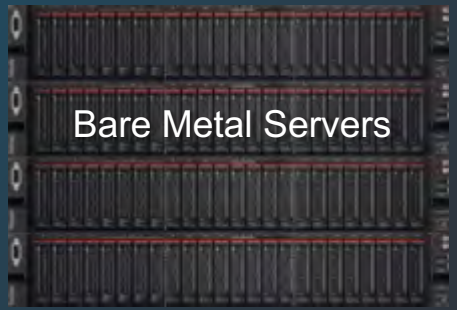
From the guest point of view (i.e. OS), it's also on a motherboard, *but that motherboard is virtual.*

From the guest point of view, "physical" DRAM is actually a cache of host memory.

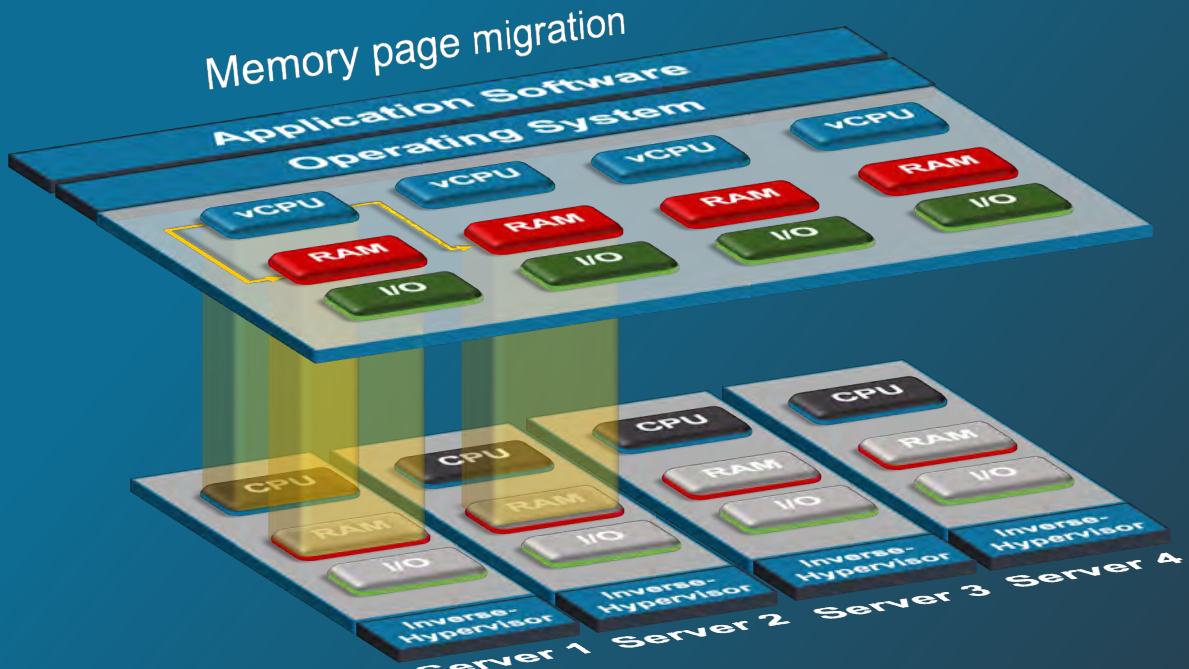
In other words, guest DRAM is a cache. It's an *all-cache design.*

Under the Hood



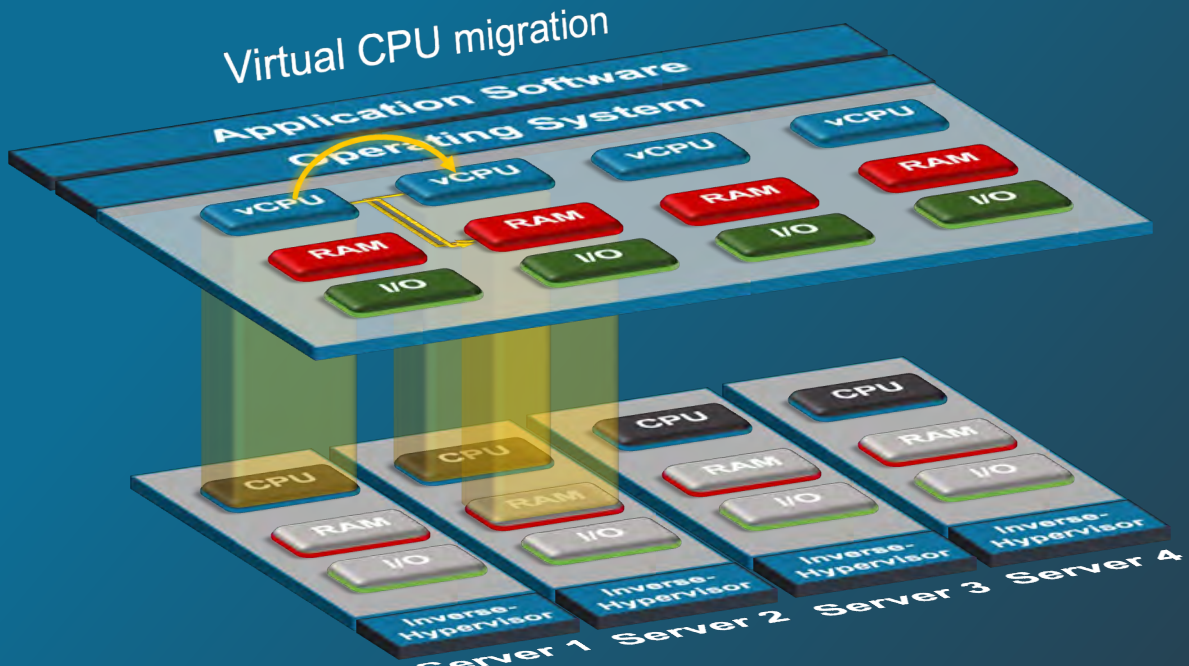


Under The Hood (move the definition of memory?)



- Uses 2 levels of hardware based dynamic address translation
- Migrates vCPUs and memory transparently to the guest OS
- Dynamically and automatically load balances
- Self-optimizing using ML
- No central control

Under The Hood (or move the processor?)



- Uses 2 levels of hardware based dynamic address translation
- Migrates vCPUs and memory transparently to the guest OS
- Dynamically and automatically load balances
- Self-optimizing using ML
- No central control

Scale Up or Out? Best of Both Worlds

	Scale Up	Scale Out	DVM
Software Simplicity	✓	✗	✓
Hardware Cost	✗	✓	✓

Summary of DVM Technology

The virtual machine aggregates the processors, all the memory, all the storage, all the Ethernets (except for a private interconnect that the DVM treats more as a system bus than a network).

The DVM treats the resources as *virtual and mobile*. The DVM hyperkernel binds the virtual resources to physical resources, dynamically, as needed, on demand.

There is *no master* node and *no shared state* in the hyperkernel. DVM managemant is distributed. Therefore, no single point of failure.

The DVM uses hardware virtualization extensions. Modern hardware has two levels of page tables to translate virtual to physical address: the OS uses the first level, the hyperkernel manage the second, invisibly to the OS.

Processor migration is fast!

Memory is always strongly cache coherent. There can be multiple copies of read only pages across the cluster. On a page write, the hyperkernel does the necessary page invalidation, TLB management, etc. Copy on write is supported. Intel store order is preserved.

The DVM hyperkernel sits between the OS and the hardware.

The DVM is completely agnostic. It looks like hardware to the OS.

Reliability

What happens if there is a hardware failure on a single node?

Does it bring down the entire cluster?

Insights

Virtual resources can migrate; therefore, we might be able to exploit redundancy using migration

Motherboards have extensive telemetry (they track temperature, DRAM errors, network errors, and more)

Existing OS's rarely make use of this data.

The DVM can track error trends (e.g. is the DRAM error rate increasing on some DIMM? Is the temperature rising indicating a potential fan failure?)

The DVM can proactively respond when thresholds are exceeded

Optimization Philosophy Based on ML

Guiding Principles:

- Machines operate faster than humans. So let them.
- Performance is dominated by the (largely unpredictable) pattern of access between processors and memory
- The hyperkernel observes, adjusts, analyzes and adapts
- Our goal was to build-in intelligence and machine-learning into the hyperkernel to automatically and dynamically monitor and optimize performance through *introspection*
- This leads us to a model in which the DVM can track *threads and working set locality*

Fast Recovery

First, some background

At boot time, Linux, as an example, clears all memory, reloads the OS, and loads application programs from disk (or over the network).

It can take 30 minutes or more to boot a 1 TB DRAM server to recover, not counting getting the applications back online.

But is it *really* necessary?

If you guessed “No” you’re correct!

As we just saw, a DVM uses two levels of dynamic address translation. We’ll now use this.

Second, be lazy

- Mark the second level page tables with a special code that indicates they must contain zeros, even if they currently don’t.
- Only when you need a MBZ page does it need to contain zeros. In other words, zero it on demand.

Storage Replica Table is stored in RPM

	Storage Replica Table	
Stable Storage Addresses	Physical Memory Address	Guest Physical Memory Address
(14, 0x3274)	0x123456	0x235698
(12, 0x1456)	0x743297	0x189543
…	…	…
…	…	…
…	…	…

The SRT must always be consistent

- The SRT is not addressable by the OS
- Modern HW has a mode to interrupt on all I/O and certain page table updates.
- The DVM marks pages as dirty when updated
- Page table updates also update the SRT
- All disk writes from memory update the SRT
- All disk reads to RPM and GPM update the SRT

How this helps

When the OS panics, it reboots. But the DVM does not need to.

The SRT is preserved in memory. (The Guest OS doesn’t even know it exists and can’t address it.)

The contents of RPM are preserved.

On boot, the OS starts reading from disk, but (guess what?) in most cases the pages are already present in RPM, so just re-map. Fake the reads from disk.

The system only reads from disk those pages not already represented in the SRT

Therefore:

Many fewer pages need to be read from stable storage in the normal case

Often, memory pages do not need to be zero-ed

So, system restart happens very quickly

Summary: benefits of a DVM

Highly scalable
No customization is required
Uses standard commodity hardware and software
Self-optimizing (not covered here)
Highly reliable
Often optimizes SW licensing costs (just pay for what you need, when you need it).
Recover quickly from OS panics
Can be deployed on-premise or in the cloud

Summary of what's new

It's important to rethink some fundamental assumptions (counter-factual thinking)

Complete OS and application compatibility is possible

Performance (trust me, it's good; sometimes better than bare HW)

Reliability/High availability is definitely achievable

Supported many deployment options (it ran at AWS, Oracle Cloud, IBM Cloud)

But wait: There's more!

Thought Experiment

What if you had a server with no processors, only memory?

Might be cheap, slow, static memory rather than DRAM ($$)

Might even be an array of SSDs that is made to look like memory

Anticipates Disaggregated Memory

(aka Memory Appliances)!!

(This was done over a weekend, with performance testing on an industry standard benchmark.)

References

- Gordon Bell: “This is the way all servers will be built in the future”
- Programming an Application When Memory Size Is No Longer A Constraint, IEEE Computer, August 2017
- Revisiting Scalable Coherent Shared Memory, Bell and Nassi, IEEE Computer, Special Issue: Outlook 2018, January 2018
- Scalable Computing Systems for Future Smart Cities, IET Smart Cities 2022;1–2