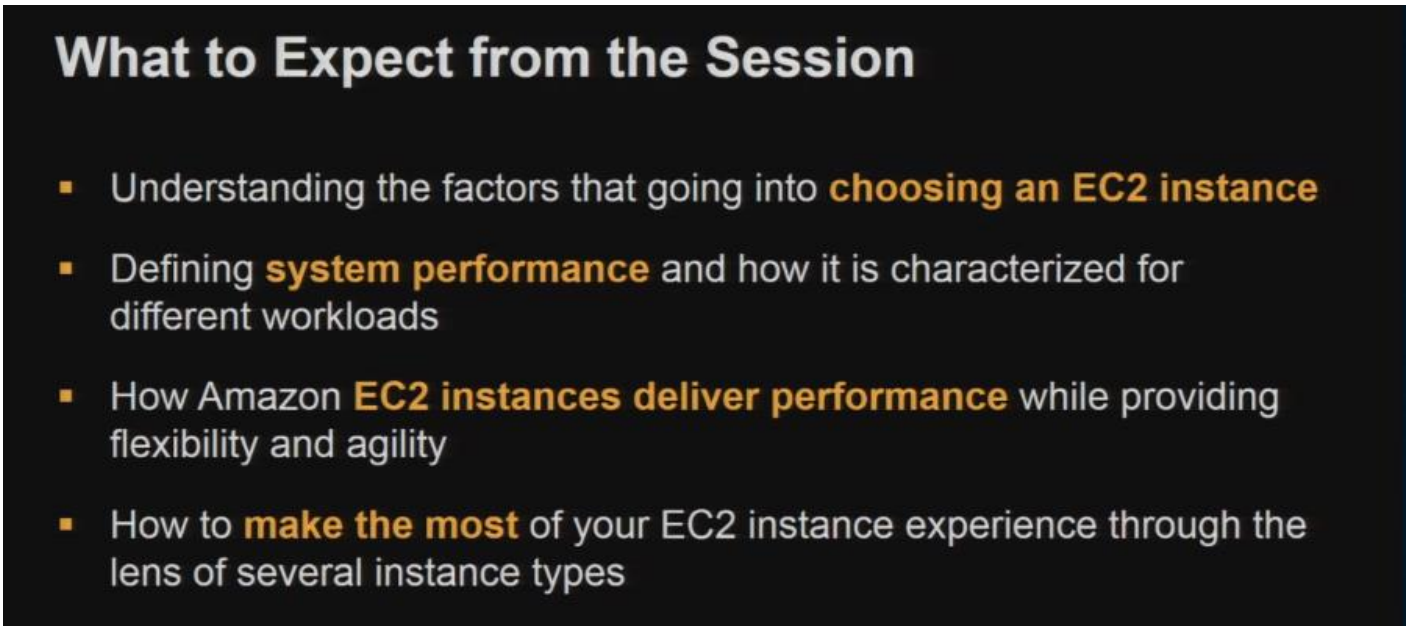
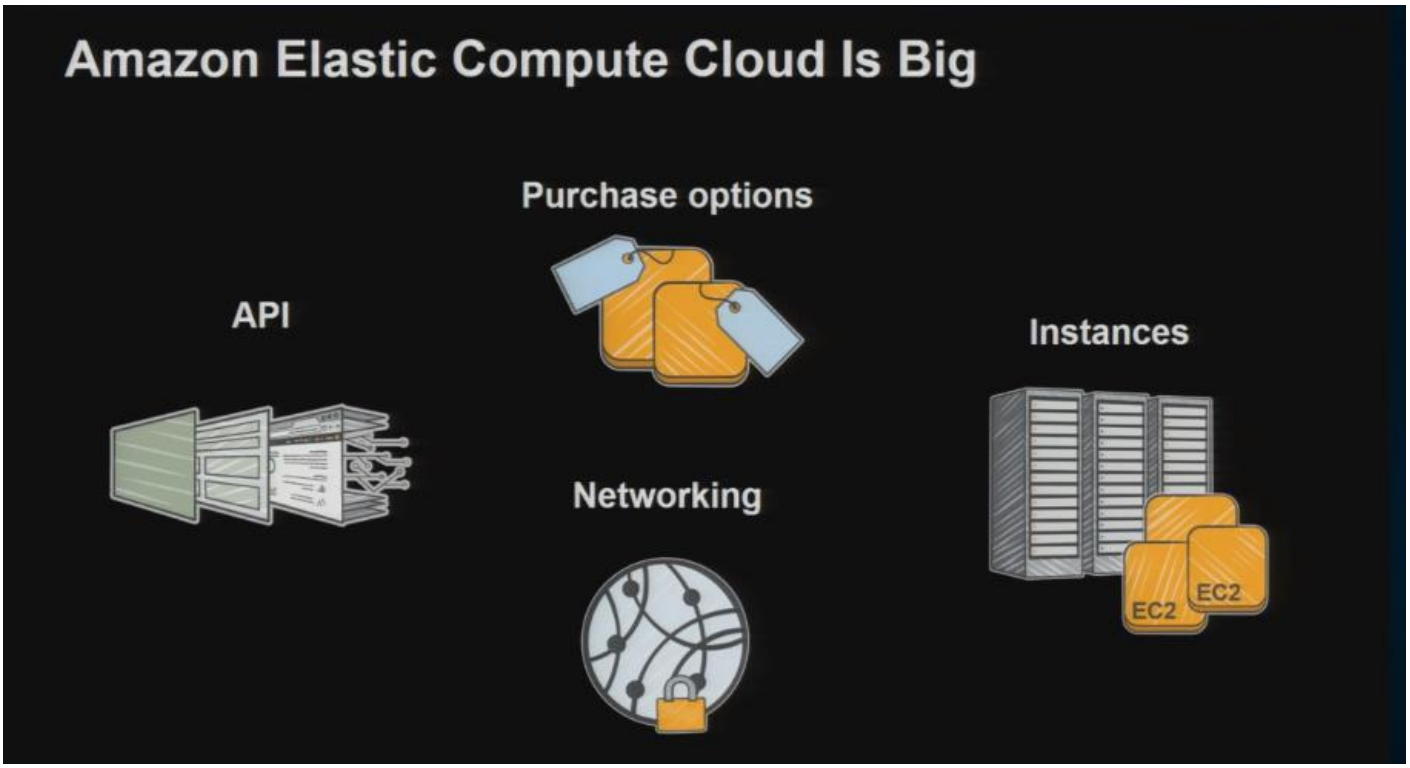
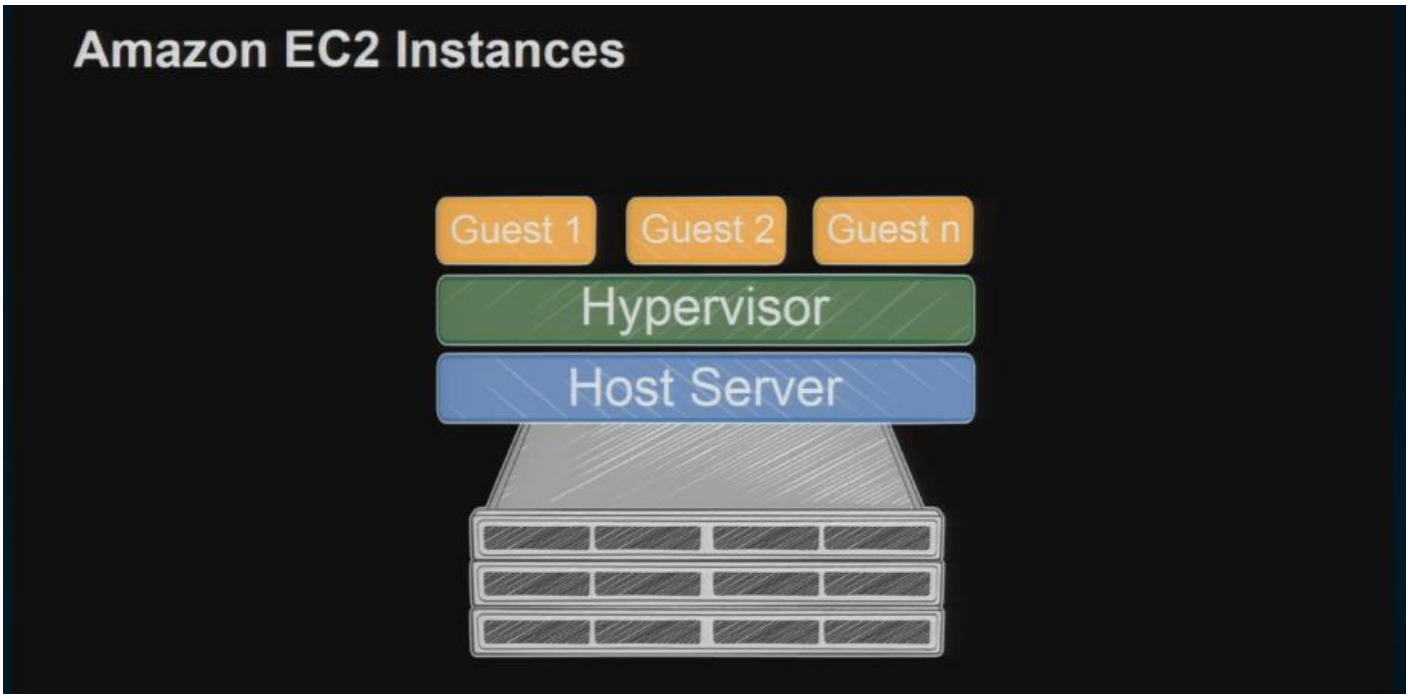
Amazon EC2 provides a broad selection of instance types to accommodate a diverse mix of workloads. In this session, we provide an overview of the Amazon EC2 instance platform, key platform features, and the concept of instance generations. We dive into the current generation design choices of the different instance families, including the General Purpose, Compute Optimized, Storage Optimized, Memory Optimized, and GPU instance families. We also detail best practices and share performance tips for getting the most out of your Amazon EC2 instances.

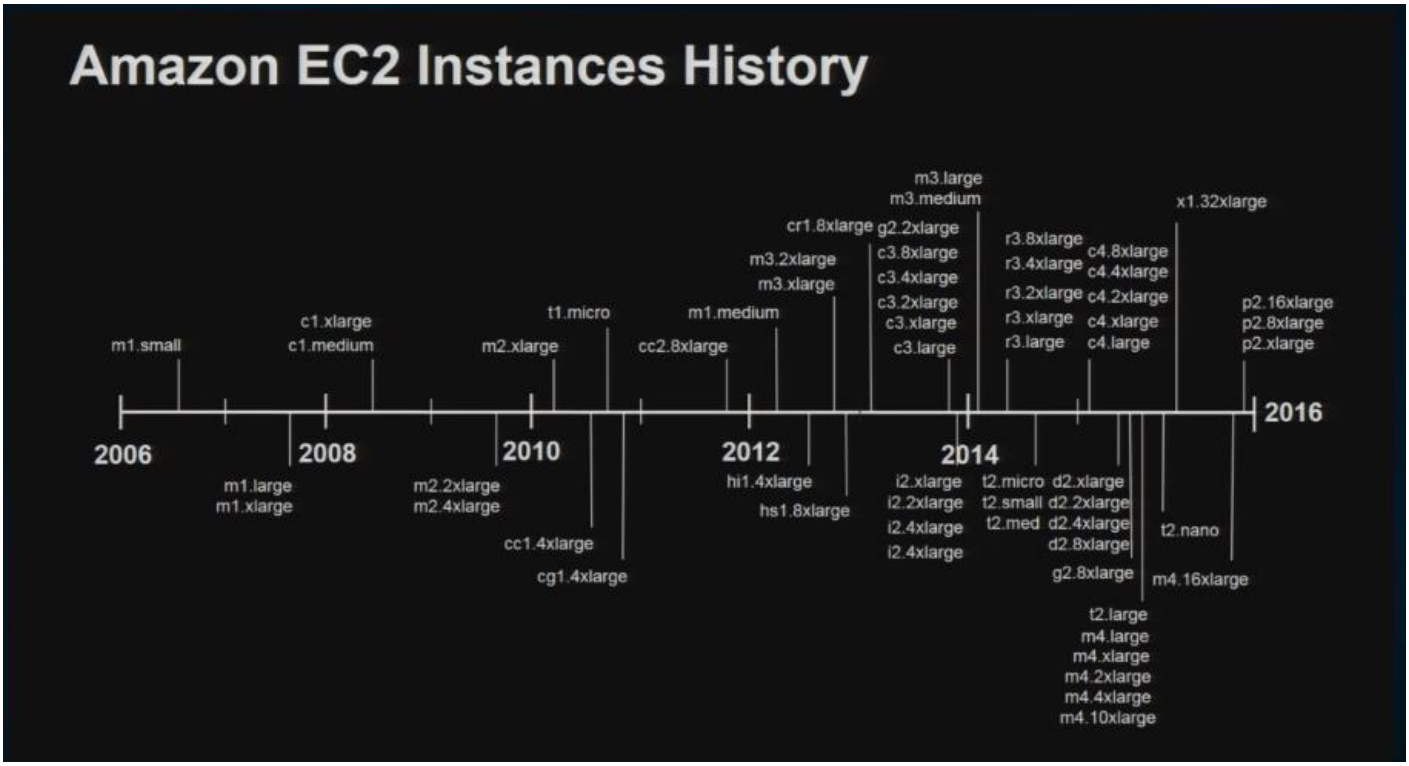


Customers for HPC using Ec2 instances are doing things like CFD, Gene Sequencing and Semiconductor design and performance is really important to them.

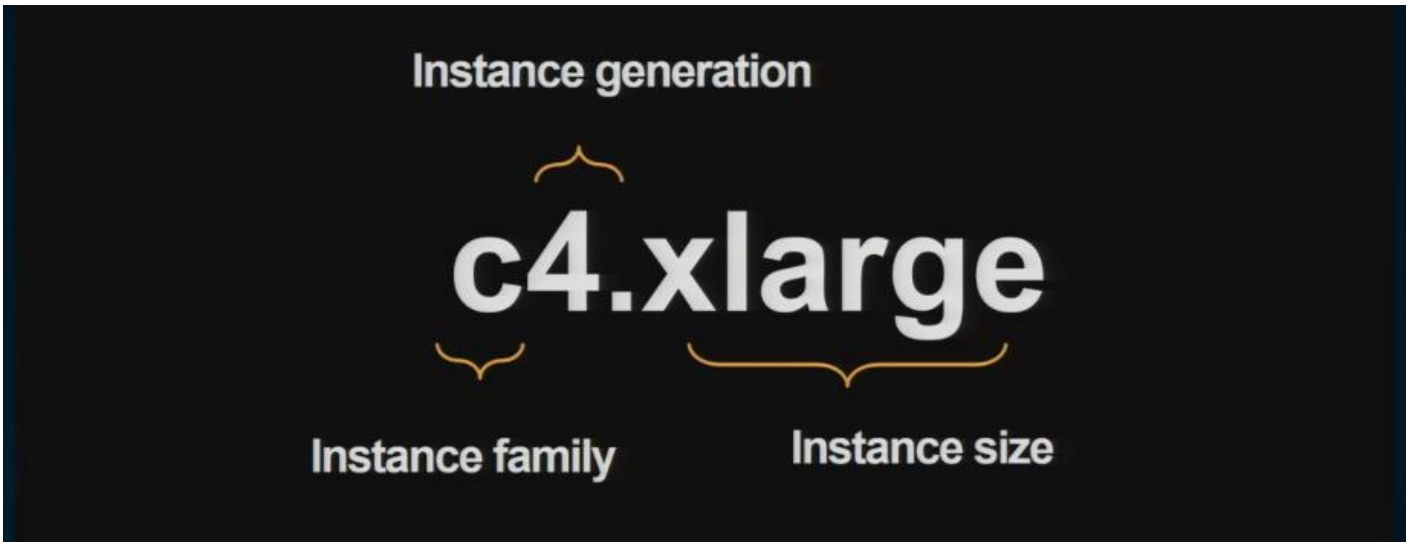




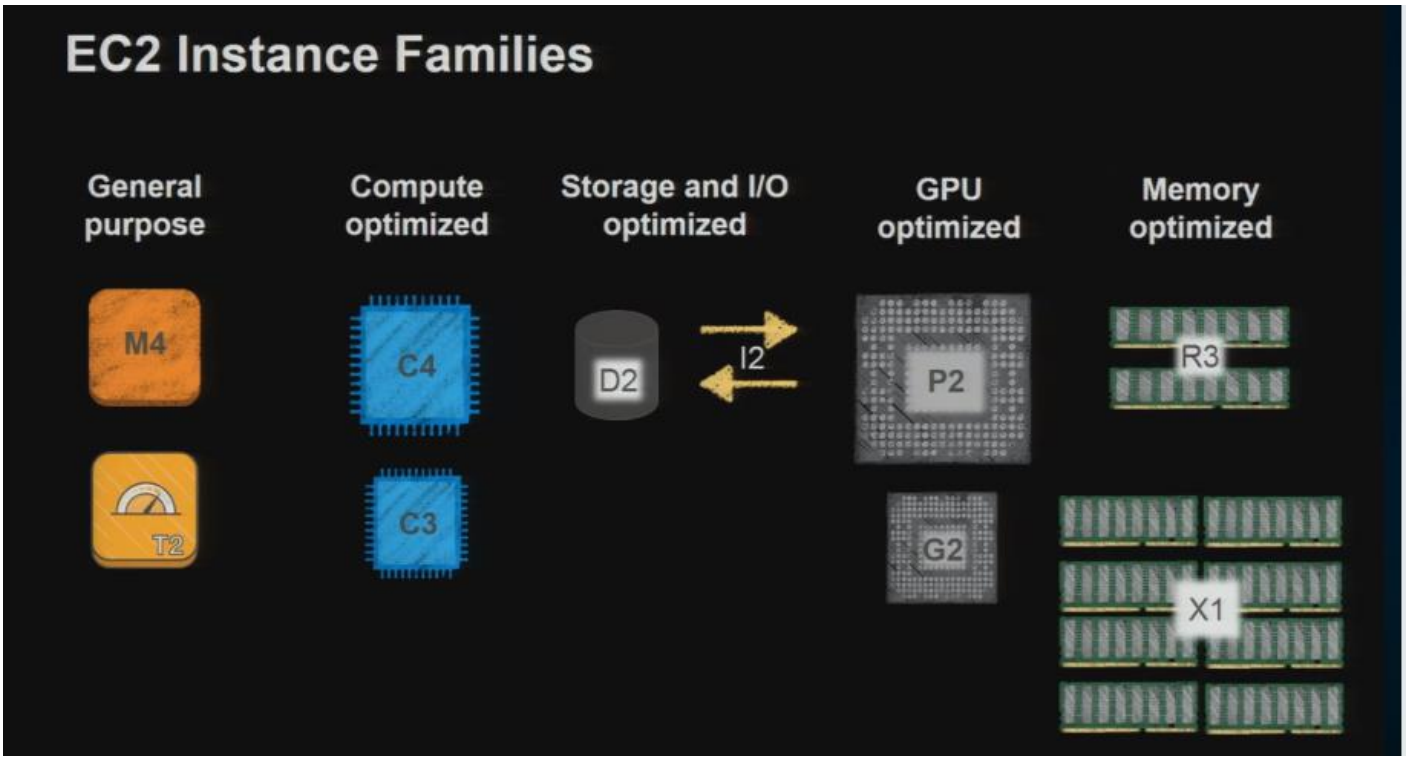
An EC2 instance is a VM, they are the guests sitting on a hypervisor on a host server.



The introduction of the cc2 instance types came with the features that gives you the ability to define the physical topology of your EC2 instances using ***Placement Groups***, where you can now define that your instances be located physically close together for the best possible bandwidth and lowest latency. This is also when we introduce Hardware Assisted Virtualization ***HVM***, that allows you to use much of the underlying hardware and not spend so much time talking to the hypervisor.



C for compute, R for RAM, I for IOPS, etc.



You have so many instance options to choose from when trying to launch your EC2 instance. Find the right instance family, for an application needed lots of memory then start with something like the R3 instance, if you need a lot of compute then start with a C4 instance, if you have an application that is pretty well balanced then you should start with a general-purpose family like a M4 or T2 instance. You can then do some testing to find the correct instance size to use within the chosen family. Check the EC2 documentation for a list of workload and types to use.

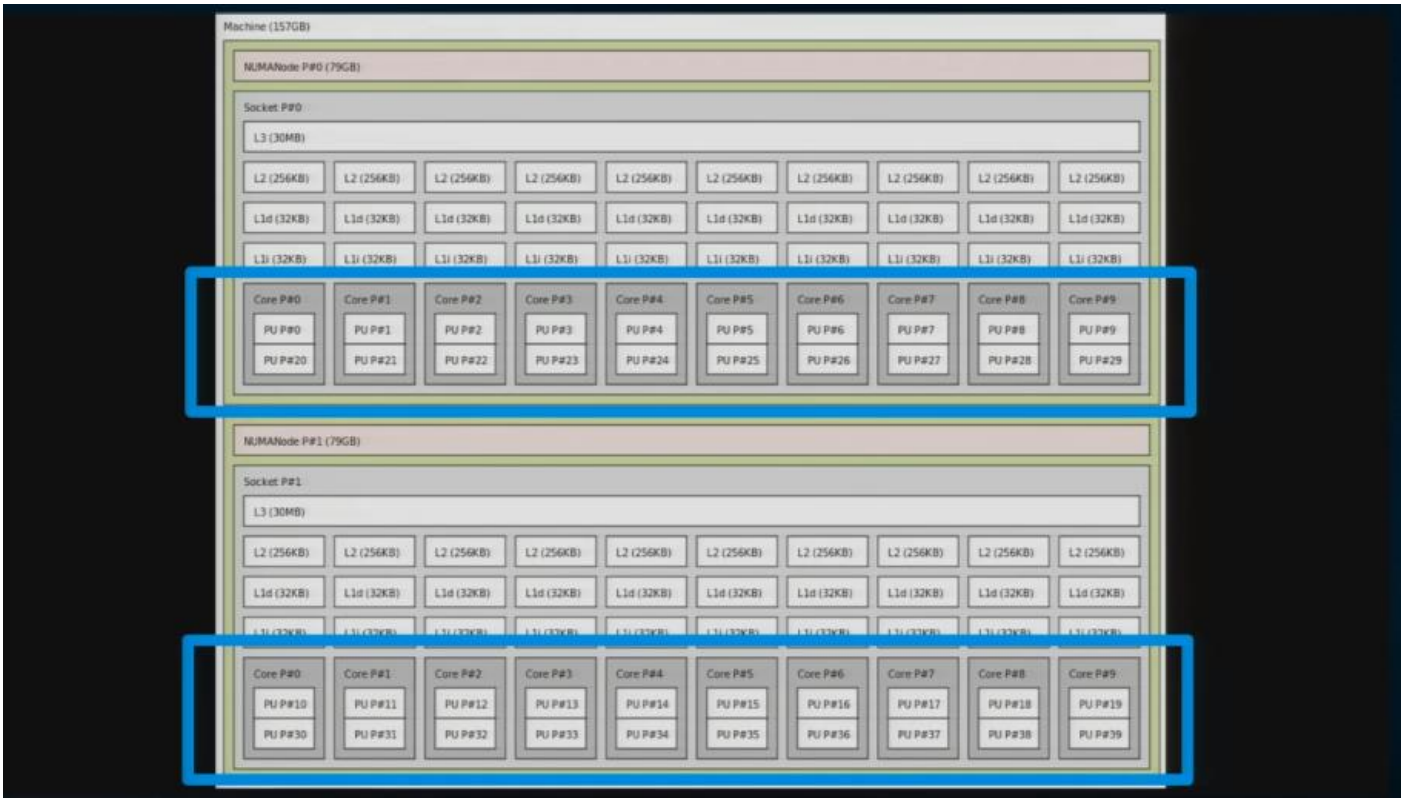
What's a Virtual CPU? (vCPU)

- A vCPU is typically a hyper-threaded physical core*
- On Linux, "A" threads enumerated before "B" threads
- On Windows, threads are interleaved
- Divide vCPU count by 2 to get core count

- Cores by EC2 & RDS DB Instance type: https://aws.amazon.com/ec2/virtualcores/

* The "t" family is special

Hyper-threading is a technology that lets you get more out of your CPU, it lets your CPU do almost 2 things at once such as a process that can get blocked on I/O.



To visualize what a vCPU looks like, here is the output of a program called LS-Topo that gives you a visual representation of the underlying hardware of your instance. Above is the output for a ***m4.10xlarge*** instance, you can see how many sockets it has and how much memory is attached to each socket, the L1 to L3 cache that it has, the physical core count and the threads attached to each core.

Disable Hyper-Threading If You Need To

- Useful for FPU heavy applications
- Use 'lscpu' to validate layout
- Hot offline the "B" threads

```
for i in `seq 64 127`; do
    echo 0 > /sys/devices/system/cpu/cpu${i}/online
done
```

- Set grub to only initialize the first half of all threads

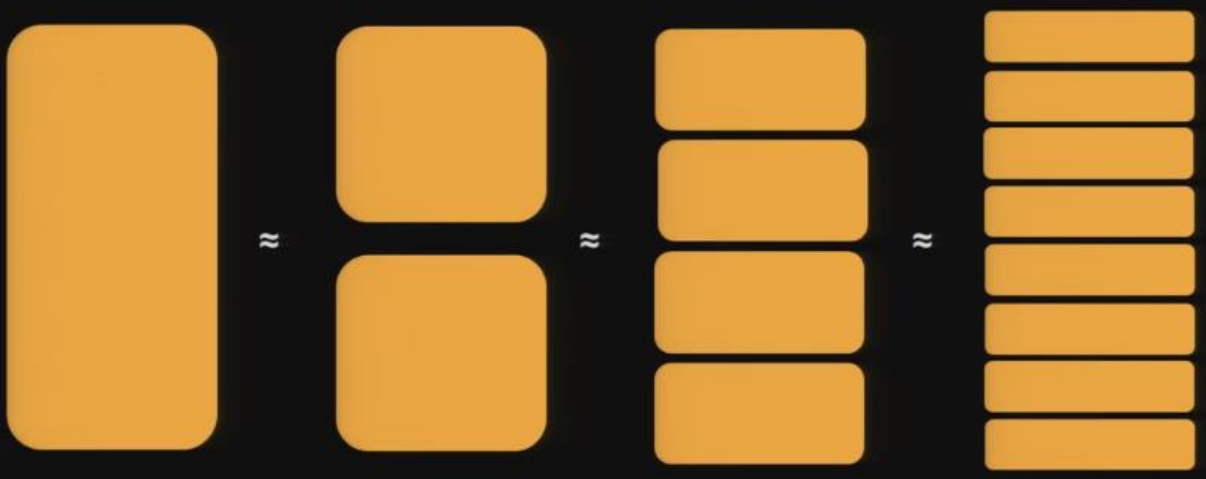
```
maxcpus=63
```

```
[ec2-user@ip-172-31-7-218 ~]$ lscpu
CPU(s):                128
On-line CPU(s) list:   0-127
Thread(s) per core:    2
Core(s) per socket:    16
Socket(s):             4
NUMA node(s):          4
Model name:            Intel(R) Xeon(R) CPU
Hypervisor vendor:     Xen
Virtualization type:   full
NUMA node0 CPU(s):     0-15,64-79
NUMA node1 CPU(s):     16-31,80-95
NUMA node2 CPU(s):     32-47,96-111
NUMA node3 CPU(s):     48-63,112-127
```

Compute heavy applications like financial risk and engineering simulations do not benefit from hyper-threading due to the very many contexts switching they do. They normally disable hyper-threading in these cases using the commands above by turning off the 'B' threads or set a grub boot parameter on the instances.

This is with hyper-threading disabled

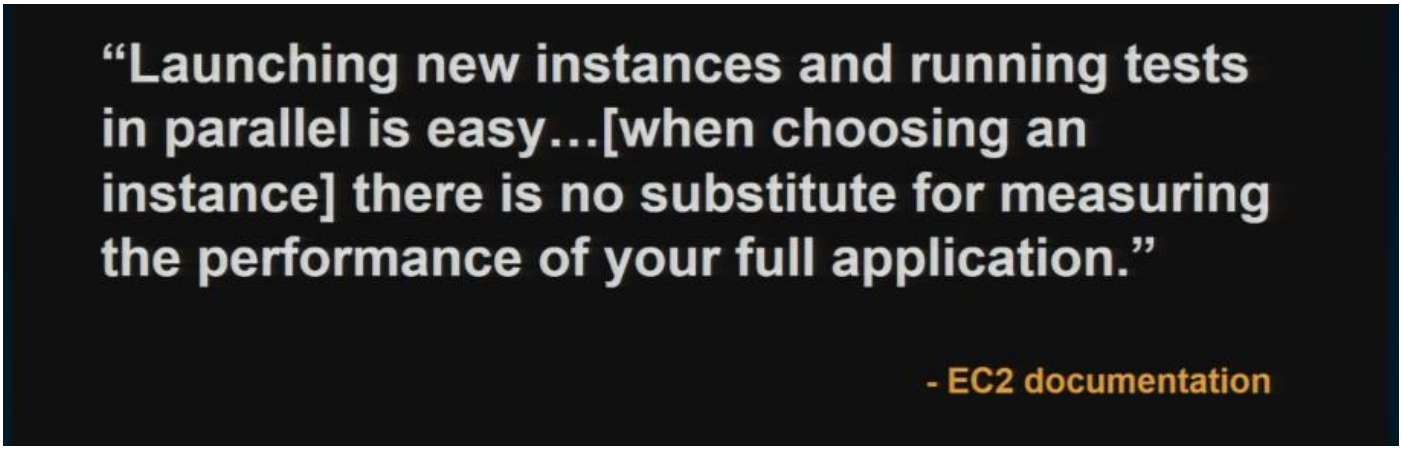
Instance sizing

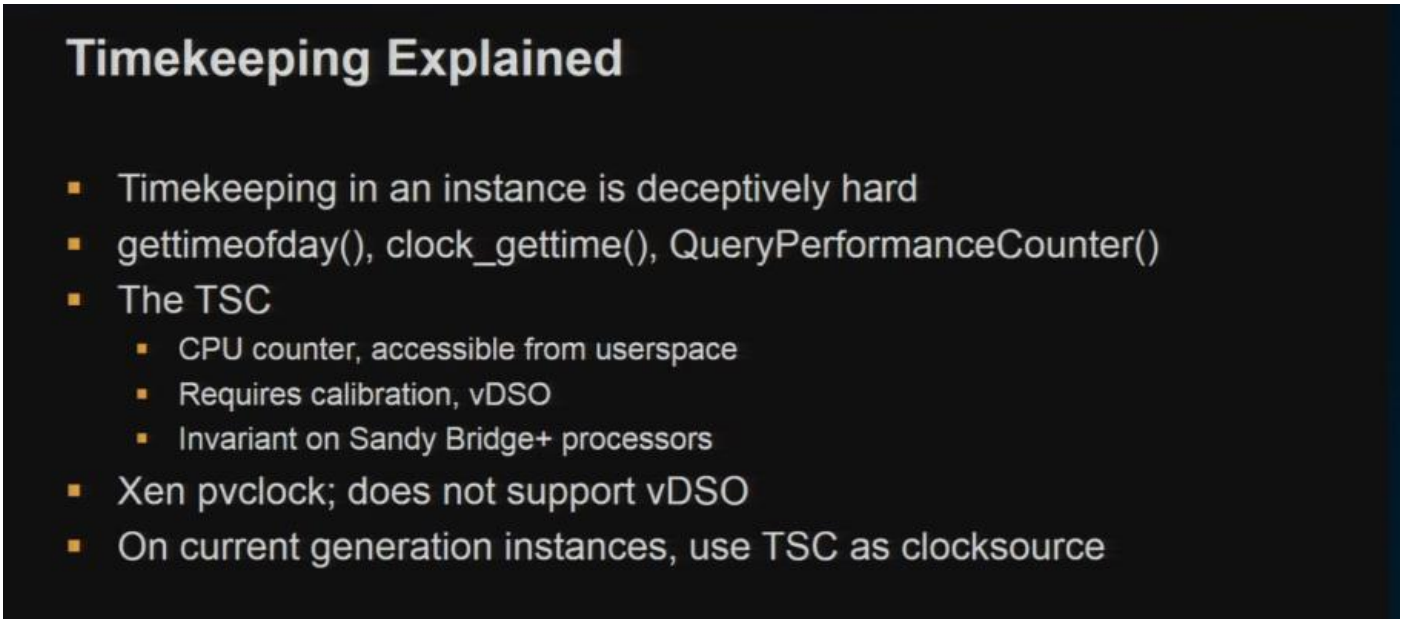


Resource Allocation

- All resources assigned to you are dedicated to your instance with no over commitment*
  - All vCPUs are dedicated to you
  - Memory allocated is assigned only to your instance
  - Network resources are partitioned to avoid “noisy neighbors”
- Curious about the number of instances per host? Use “Dedicated Hosts” as a guide.

*Again, the “T” family is special





Most instances will use the Xen clock source by default because this is compatible with every single instance that is available from AWS.

Benchmarking - Time Intensive Application

```
#include <sys/time.h>
#include <time.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
time_t start,end;
time (&start);
        for ( int x = 0; x < 100000000; x++ ) {
                float f;
                float g;
                float h;
                f = 123456789.0f;
                g = 123456789.0f;
                h = f * g;
                struct timeval tv;
                gettimeofday(&tv, NULL);
        }
time (&end);
double dif = difftime (end,start);
printf ("Elasped time is %.2lf seconds.\n", dif );
return 0;
}
```

This demo app performs a lot of getTimeOfDay() calls and a little bit of floating point math.

Using the Xen Clock Source

```
[centos@ip-192-168-1-77 testbench]$ strace -c ./test
Elasped time is 12.00 seconds.
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 99.99    3.322956           2   2001862           gettimeofday
  0.00    0.000096           6        16           mmap
  0.00    0.000050           5        10           mprotect
  0.00    0.000038           8         5           open
  0.00    0.000026           5         5           fstat
  0.00    0.000025           5         5           close
  0.00    0.000023           6         4           read
  0.00    0.000008           8         1         1 access
  0.00    0.000006           6         1           brk
  0.00    0.000006           6         1           execve
  0.00    0.000005           5         1           arch_prctl
  0.00    0.000000           0         1           munmap
------ ----------- ----------- --------- --------- ----------------
100.00    3.323239              2001912         1 total
```

This is the result with the ***Xen clock source*** using the ***strace*** profiling tool

Using the TSC Clock Source

```
[centos@ip-192-168-1-77 testbench]$ strace -c ./test
Elasped time is 2.00 seconds.
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 32.97    0.000121           7        17           mmap
 20.98    0.000077           8        10           mprotect
 11.72    0.000043           9         5           open
 10.08    0.000037           7         5           close
  7.36    0.000027           5         6           fstat
  6.81    0.000025           6         4           read
  2.72    0.000010          10         1           munmap
  2.18    0.000008           8         1         1 access
  1.91    0.000007           7         1           execve
  1.63    0.000006           6         1           brk
  1.63    0.000006           6         1           arch_prctl
  0.00    0.000000           0         1           write
------ ----------- ----------- --------- --------- ----------------
100.00    0.000367                    53         1 total
```

This is the result of using the ***TSC clock source***, this is so fast just by switching the clock source.

Tip: Use TSC as clocksource

```
# cat /sys/devices/system/cl*/cl*/available_clocksource
xen tsc hpet acpi_pm

# cat /sys/devices/system/cl*/cl*/current_clocksource
xen
```

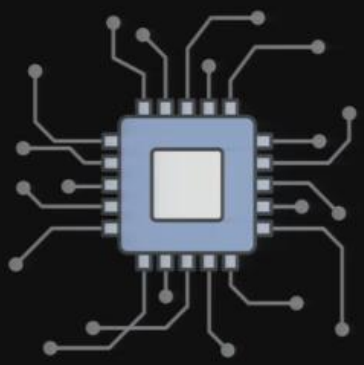
Change with:

```
# echo tsc > /sys/devices/system/cl*/cl*/current_clocksource
```

The change is easy to make on Linux, choose TSC if you have it to get up to 40% better performance for an app that does a lot of time keeping calls

P-state and C-state Control

- c4.8xlarge, d2.8xlarge, m4.10xlarge, m4.16xlarge, p2.16xlarge, x1.16xlarge, x1.32xlarge
- By entering deeper idle states, non-idle cores can achieve up to 300MHz higher clock frequencies
- But… deeper idle states require more time to exit, may not be appropriate for latency-sensitive workloads
- Limit c-state by adding "intel_idle.max_cstate=1" to grub



Tip: P-state Control for AVX2

- If an application makes heavy use of AVX2 on all cores, the processor may attempt to draw more power than it should
- Processor will transparently reduce frequency
- Frequent changes of CPU frequency can slow an application

```
sudo sh -c "echo 1 > /sys/devices/system/cpu/intel_pstate/no_turbo"
```

See also: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/processor_state_control.html

Review: T2 Instances

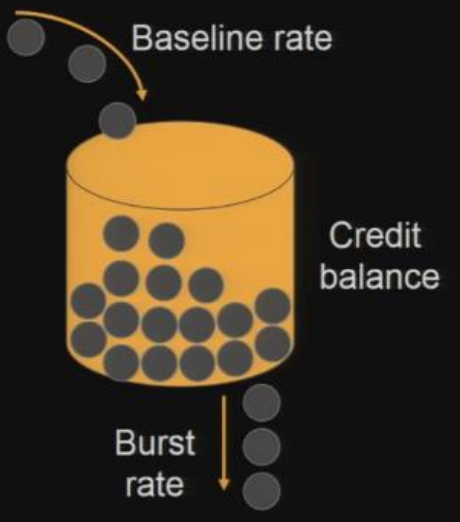
- Lowest cost EC2 instance at $0.0065 per hour
- Burstable performance
- Fixed allocation enforced with CPU credits

Model	vCPU	Baseline	CPU Credits / Hour	Memory (GiB)	Storage
t2.nano	1	5%	3	.5	EBS Only
t2.micro	1	10%	6	1	EBS Only
t2.small	1	20%	12	2	EBS Only
t2.medium	2	40%**	24	4	EBS Only
t2.large	2	60%**	36	8	EBS Only

General purpose, web serving, developer environments, small databases

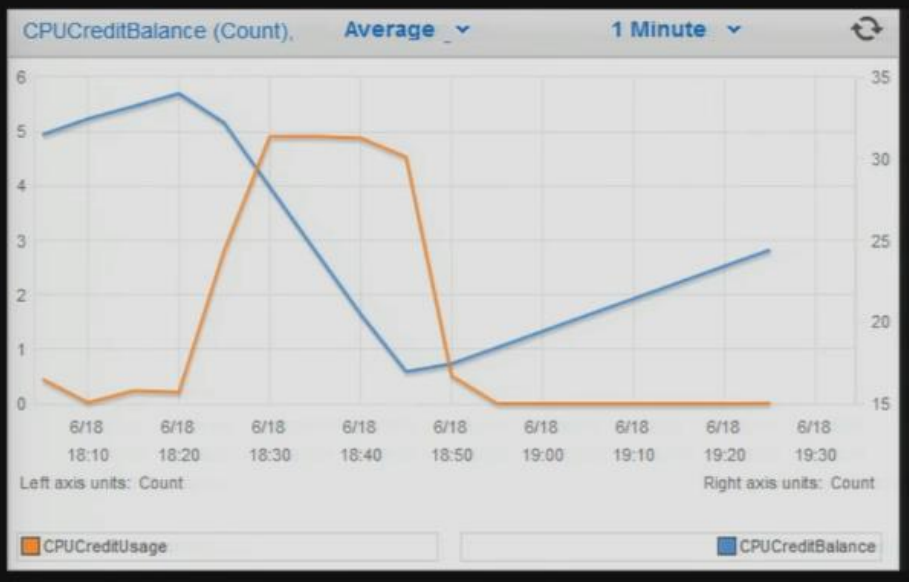
These are great for workloads that have burstable CPU performance like database servers, websites, and dev environments.

How Credits Work



- A CPU credit provides the performance of a full CPU core for one minute
- An instance earns CPU credits at a steady rate
- An instance consumes credits when active
- Credits expire (leak) after 24 hours

Tip: Monitor CPU Credit Balance



There are 2 different CloudWatch metrics that you can monitor to see your available credit for boosting in your EC2 t2 instances.

Review: X1 Instances

- Largest memory instance with 2 TB of DRAM
- Quad socket, Intel E7 processors with 128 vCPUs

Model	vCPU	Memory (GiB)	Local Storage	Network
x1.16xlarge	64	976	1x 1920GB SSD	10Gbps
x1.32xlarge	128	1952	2x 1920GB SSD	20Gbps

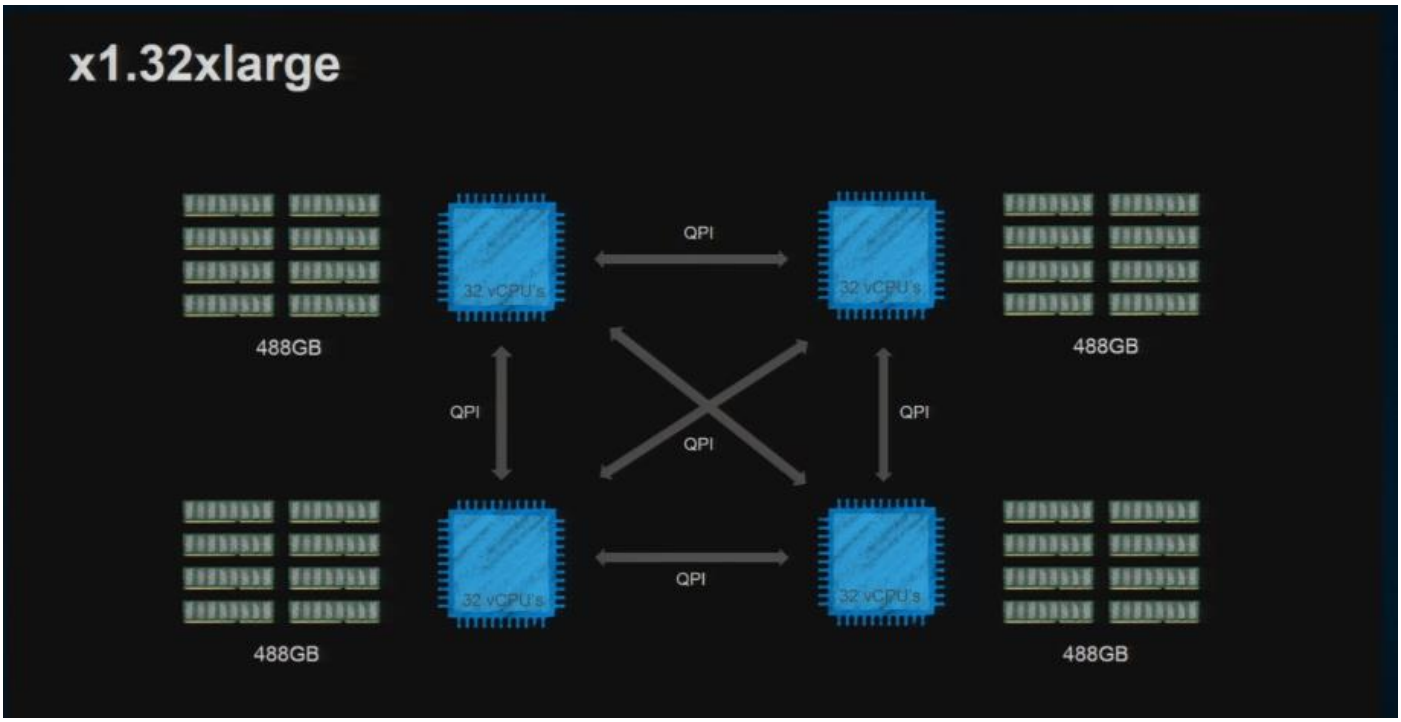
In-memory databases, big data processing, HPC workloads

NUMA

- Non-uniform memory access
- Each processor in a multi-CPU system has local memory that is accessible through a fast interconnect
- Each processor can also access memory from other CPUs, but local memory access is a lot faster than remote memory
- Performance is related to the number of CPU sockets and how they are connected - Intel QuickPath Interconnect (QPI)

r3.8xlarge

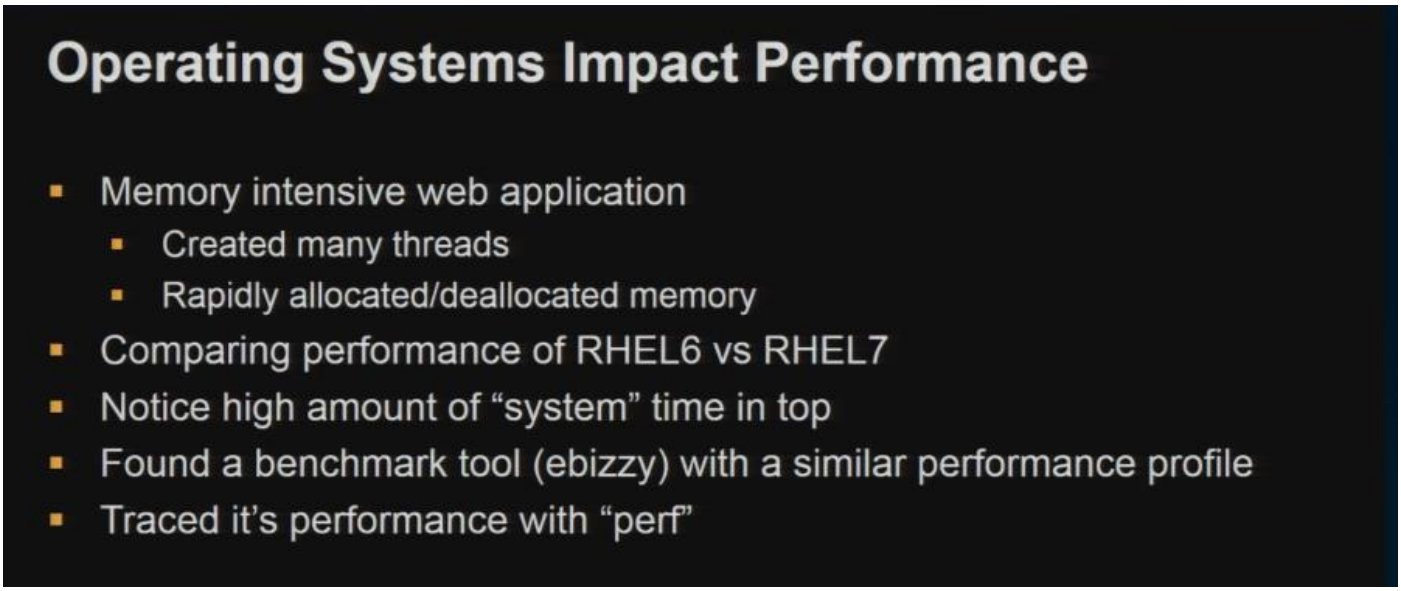
This is a 2-socket box with 122GB RAM attached to each socket to give 16 vCPUs, the sockets can communicate over the QPI paths. So, if you have an application that is running on the socket on the left that is copying data from the memory on the socket on the right, its going to send/get that data over those QPI paths which while fast, are never going to be as fast as accessing memory that is local to that socket.

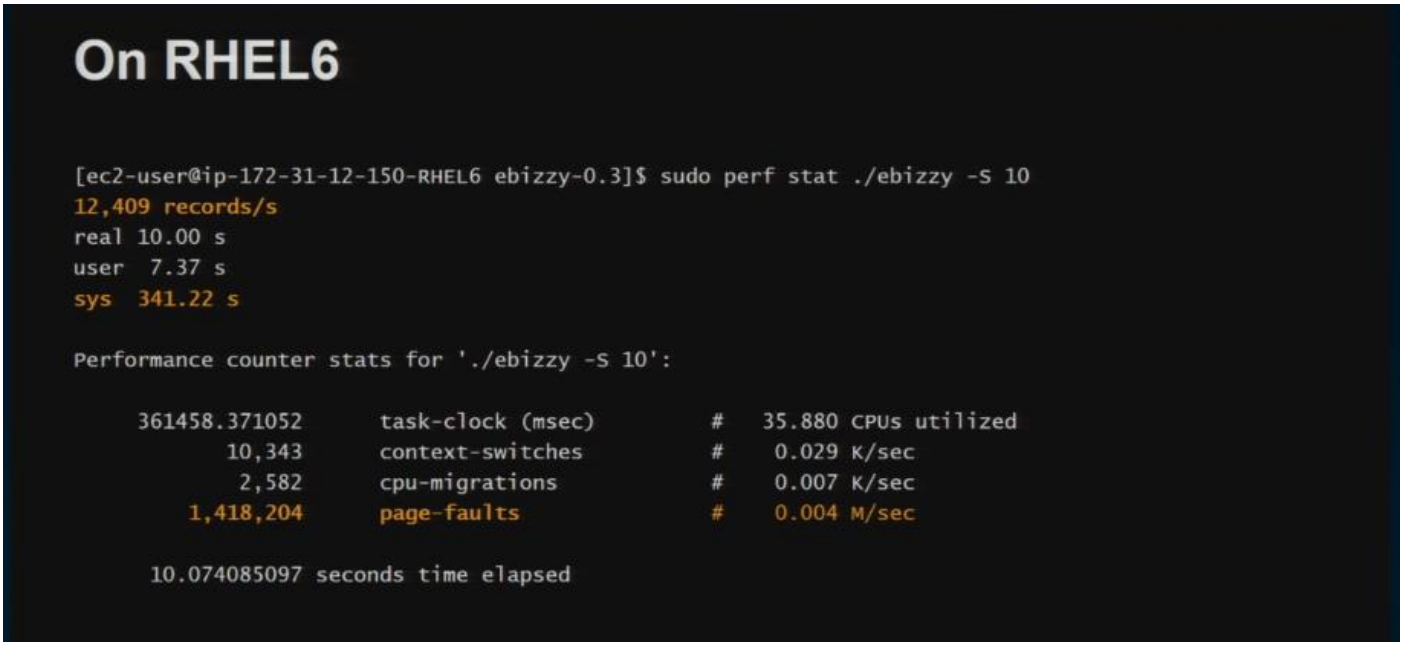


There are 4 sockets in the x1 instance and things get more complex and NUMA is even more important. We now have far more memory per socket and just 1 QPI connecting each socket to any other socket. This means that memory transfer from one NUMA zone to another NUMA zone are going to take longer here on the x1 instance than they would take on the r3 instance. What can you do about this?

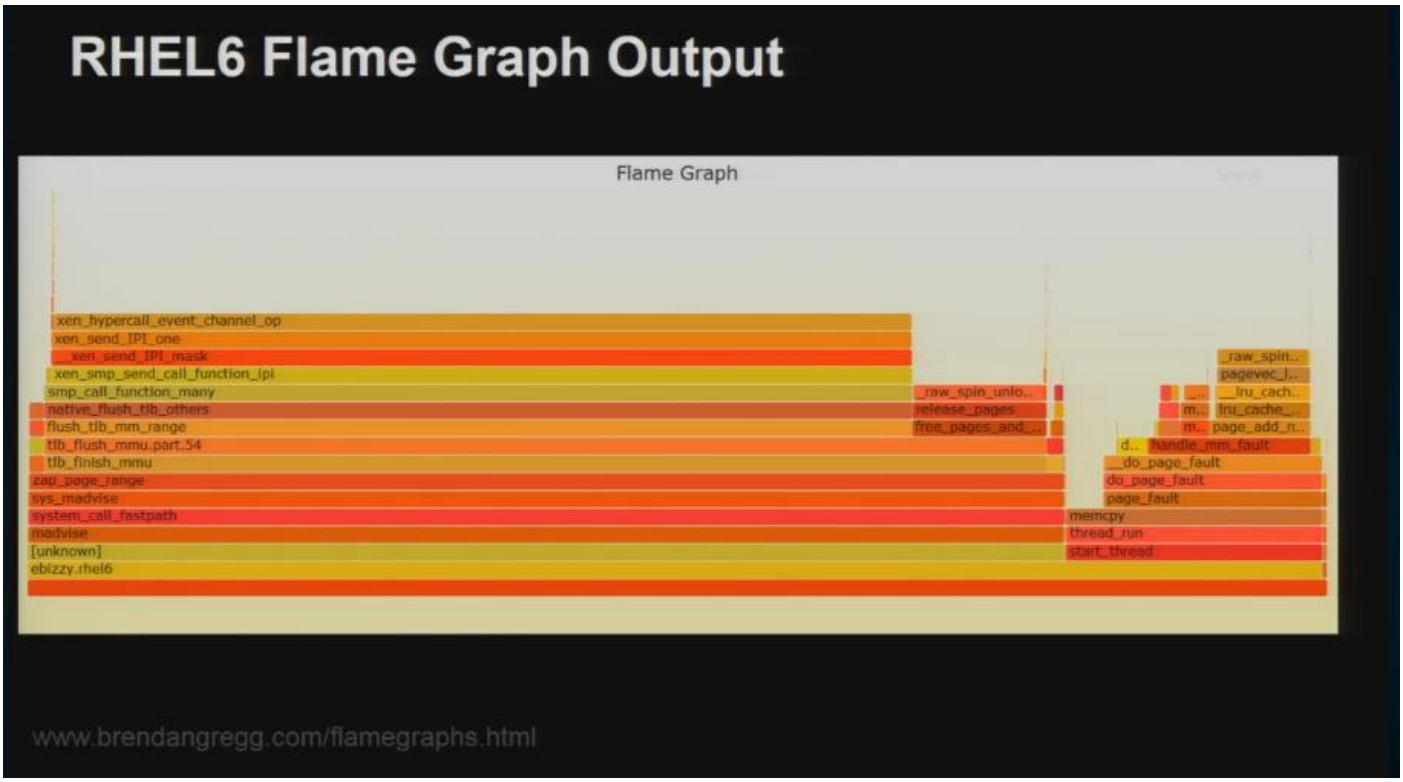
Tip: Kernel Support for NUMA Balancing

- An application will perform best when the threads of its processes are accessing memory on the same NUMA node.
- NUMA balancing moves tasks closer to the memory they are accessing.
- This is all done automatically by the Linux kernel when automatic NUMA balancing is active: version 3.8+ of the Linux kernel.
- Windows support for NUMA first appeared in the Enterprise and Data Center SKUs of Windows Server 2003.
- Set “numa=off” or use numactl to reduce NUMA paging if your application uses more memory than will fit on a single socket or has threads that move between sockets

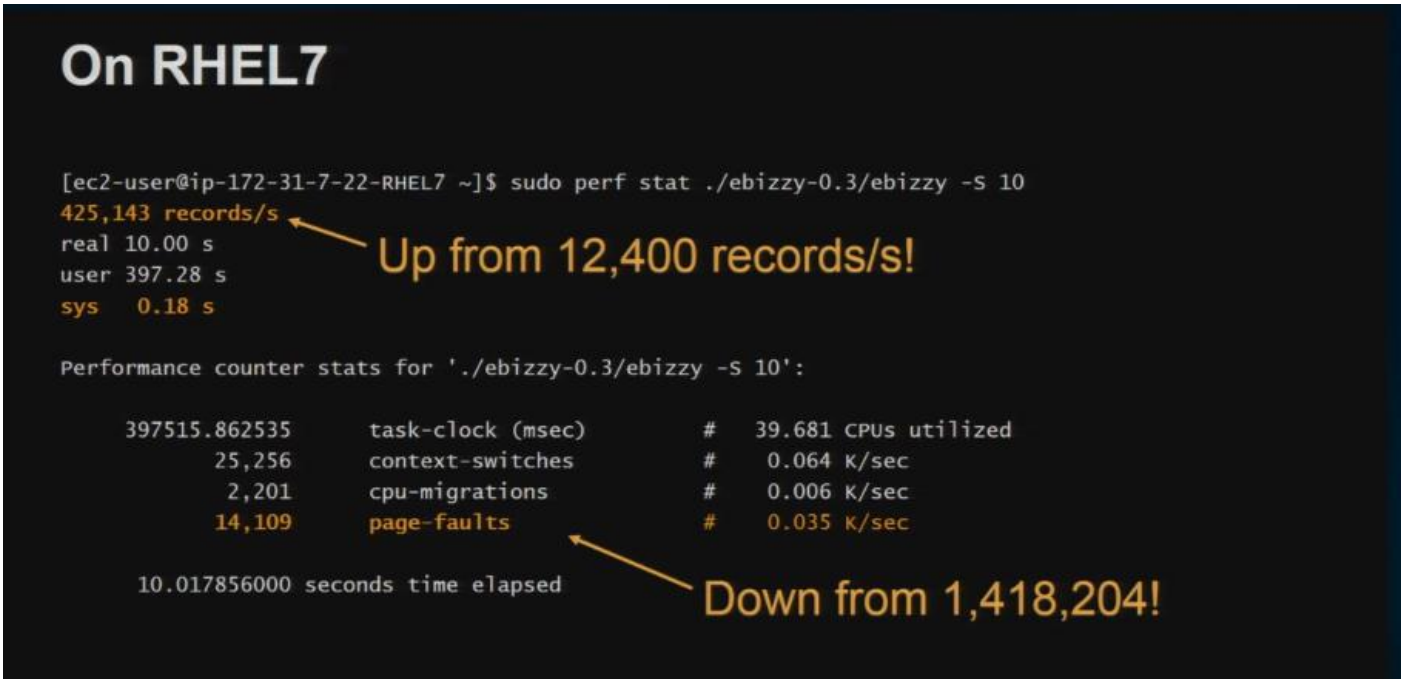




We used a performance testing tool called ***ebizzy*** that we used to test the web application on the RHEL 6 operating system, we used the ***$ sudo perf stat ./ebizzy -s 10*** command for a 10 second run. We also profiled the same application using the ***perf*** tool that can help us understand what is going on at the system level on the web application also as seen above. We are getting a lot of context switches for this 10 sec run.

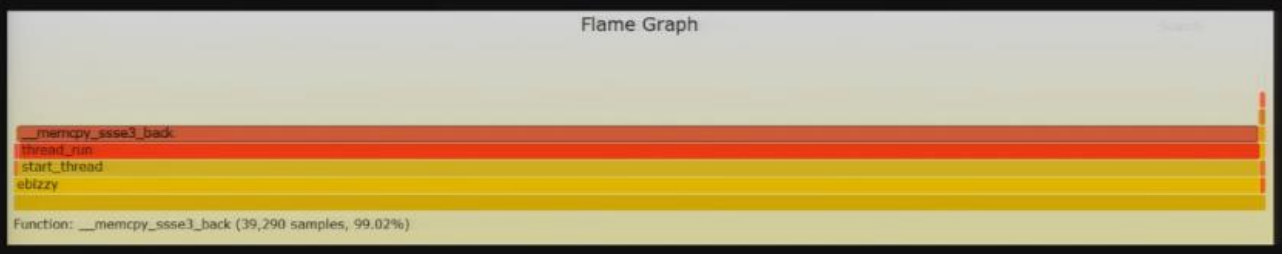


We also used the Flame graph tool to understand where the time is being spent in the web app and the code paths.



We then took this same exact code and recompiled it for RHEL 7 and ran the exact same test, our performance went from 12,000 records to 425,000 records

RHEL7 Flame Graph Output



Again, we see the flame graph for this case, in RHEL 7, glibc changed the way some things were being done. Always try to run your application on the latest AWS images available

Hugepages

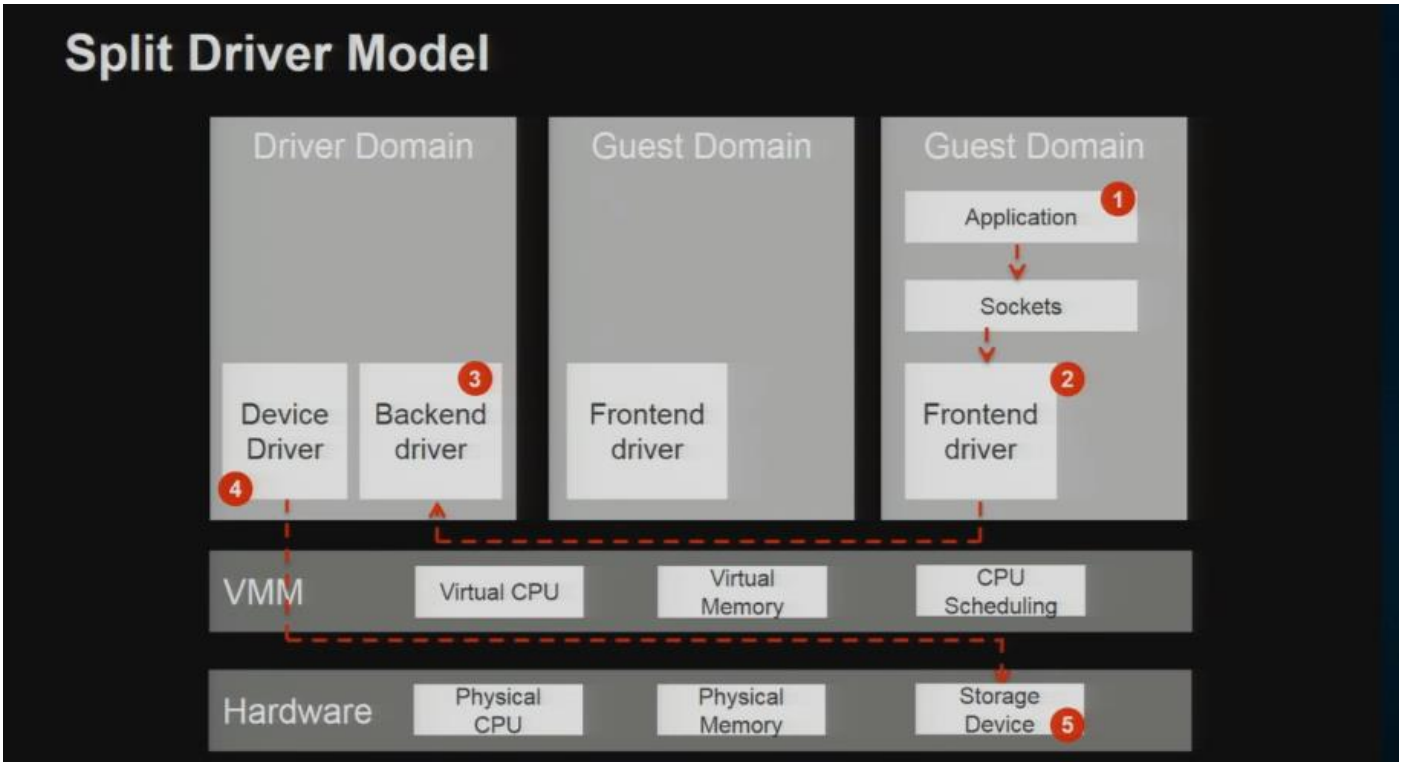
- Disable Transparent Hugepages

```
# echo never > /sys/kernel/mm/redhat_transparent_hugepage/enabled
# echo never > /sys/kernel/mm/redhat_transparent_hugepage/defrag
```

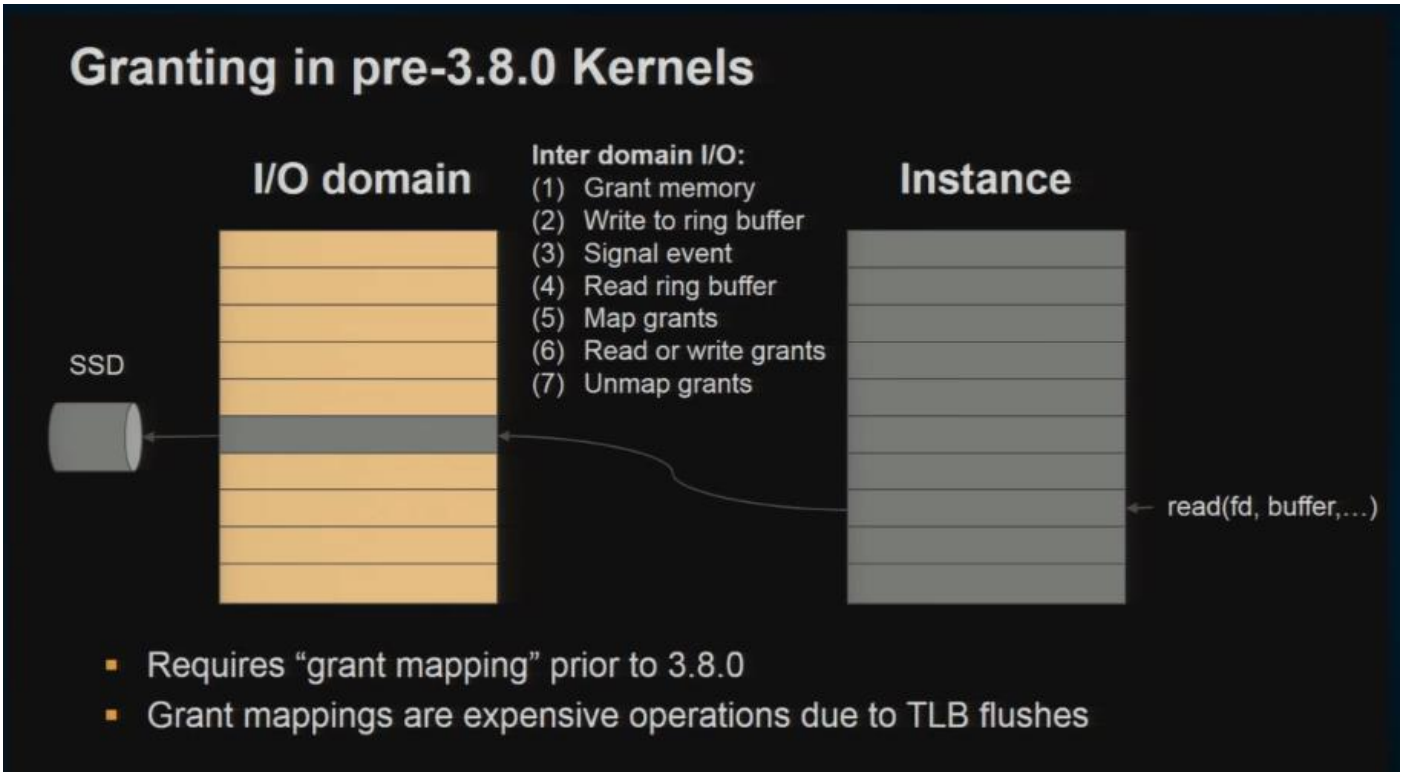
- Use Explicit Huge Pages

```
$ sudo mkdir /dev/hugetlbfs
$ sudo mount -t hugetlbfs none /dev/hugetlbfs
$ sudo sysctl -w vm.nr_hugepages=10000
$ HUGETLB_MORECORE=yes LD_PRELOAD=libhugetlbfs.so numactl --cpunodebind=0 \
     --membind=0 /path/to/application
```

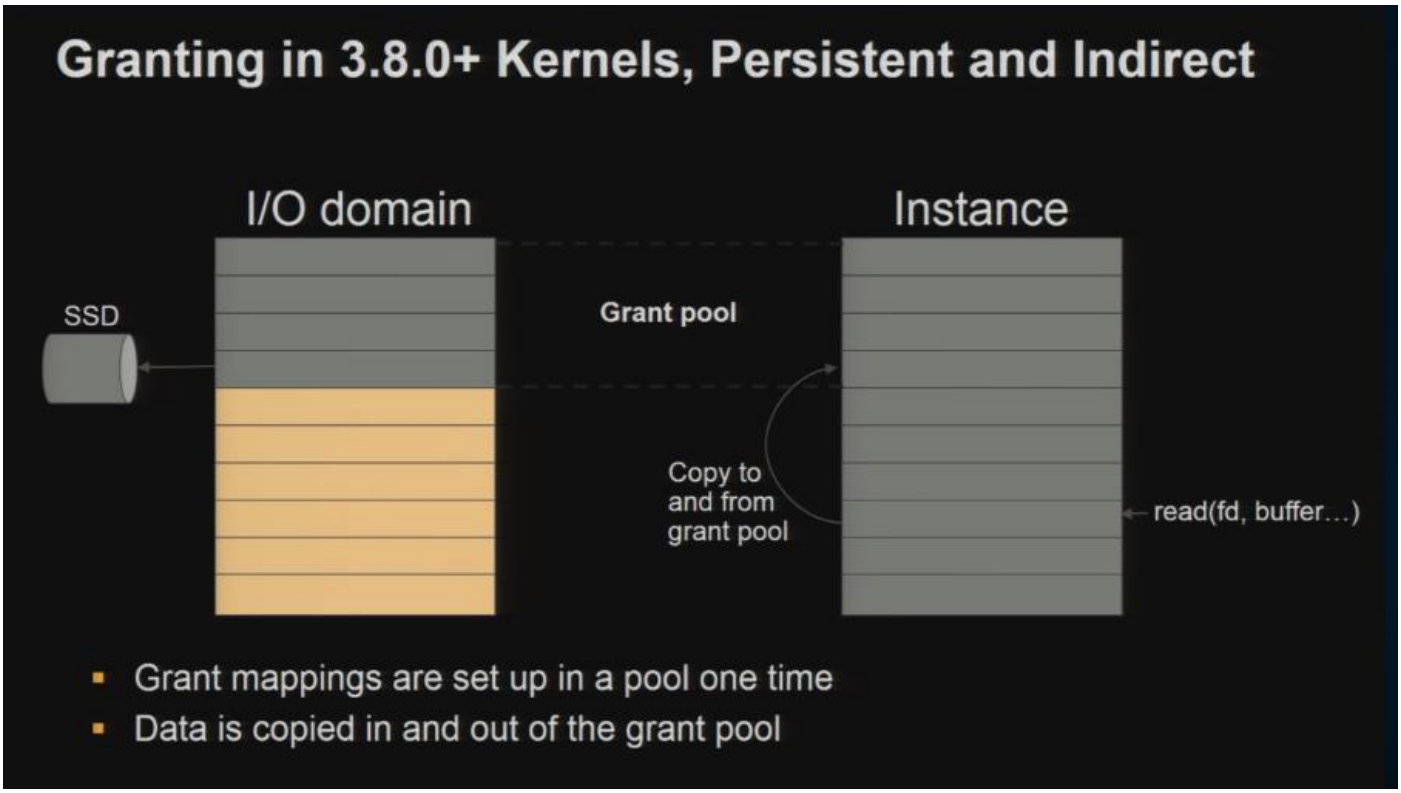
See also: https://lwn.net/Articles/375096/



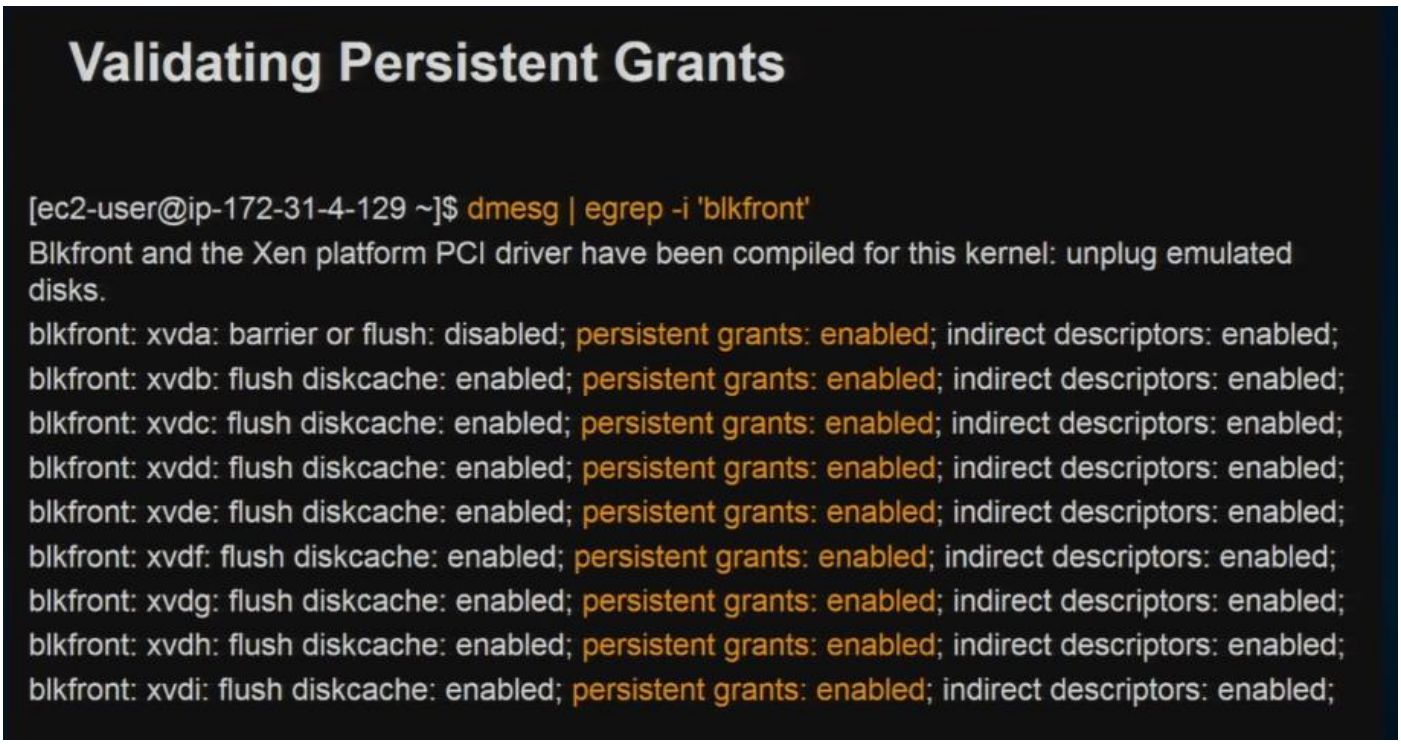
There are a few I instance families that are optimized for I/O usage, they have a bunch of SSD drives. The D2 instances have dense storage that use magnetic drives. To get the best performance from these instances you need to be running a modern Linux kernel on a modern OS, this is because of the split driver model that Xen uses.



This is very cumbersome and not easy to do on your own



This approach will help you solve that problem



2009 – Longer ago than you think

- Avatar was the top movie in the theaters
- Facebook overtook MySpace in active users
- President Obama was sworn into office
- **The 2.6.32 Linux kernel was released**

Tip: Use 3.10+ kernel

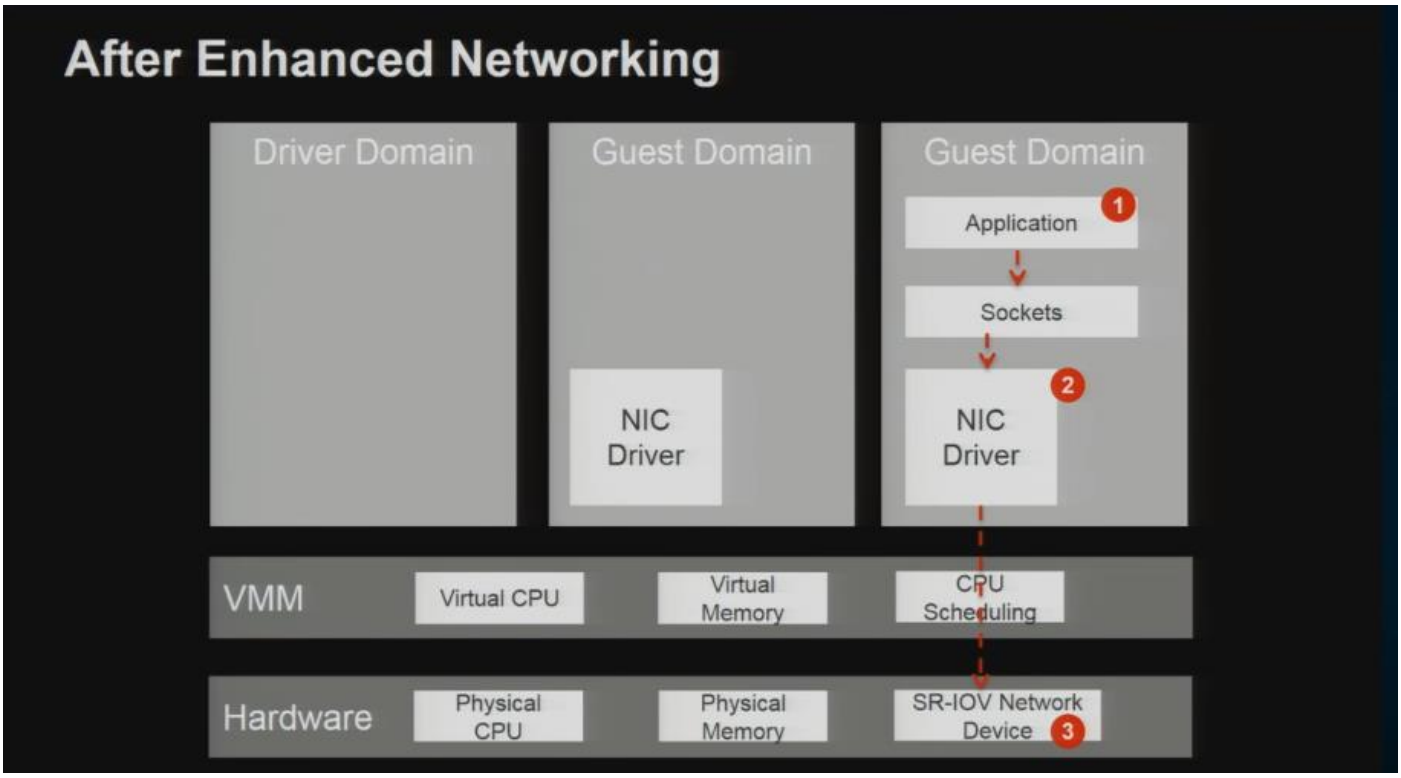
- Amazon Linux 13.09 or later
- Ubuntu 14.04 or later
- RHEL/Centos 7 or later
- Etc.

Please use a modern OS with a modern kernel

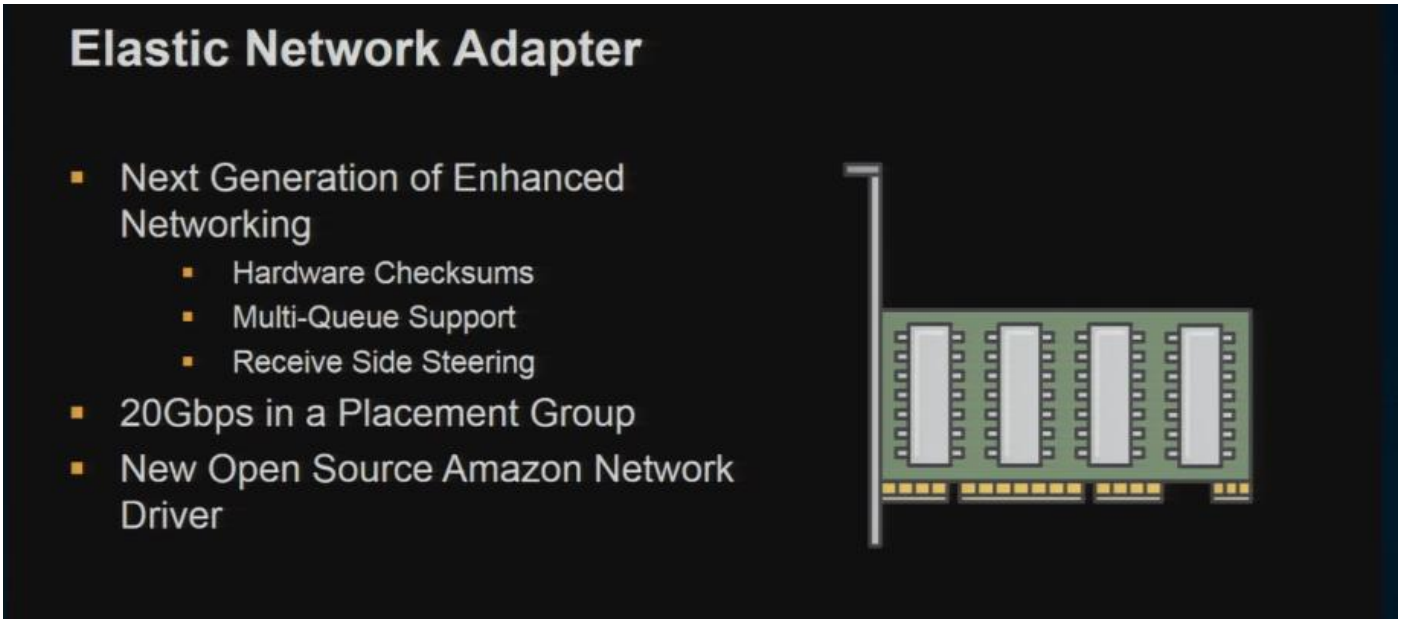
Device Pass Through: Enhanced Networking

- SR-IOV eliminates need for driver domain
- Physical network device exposes virtual function to instance
- Requires a specialized driver, which means:
  - Your instance OS needs to know about it
  - EC2 needs to be told your instance can use it

Enhanced Networking is a new way to talk to networked devices, it uses the technology called ***Single Root I/O Virtualization SR-IOV*** to allow the physical network device be exposed directly to your OS so that your calls do not need to go through the hypervisor. You need a special driver to be installed in your OS and EC2 needs to be told to expose the networked device in the correct way



The networked path is much simpler and packets don't need to go through the hypervisor anymore. Also, because you are now talking to bare metal, you will get a higher rate of packets per second and decreased jitter because the CPU is no longer involved in the process. Enhanced Networking is free on all supported instances and is enabled by default in most AMIs, it is highly to have enhanced networking feature turned on if you are doing anything that touches the network on an EC2 instance.



The Elastic Network Adapter ENA is on some bigger instances, it offers you up to 20 Gbps as opposed to the 10Gbps that you get with enhanced networking communication between instances in the same placement groups. It also has some other features that help you get more packets per second moved.

Network Performance

- 20 Gigabit & 10 Gigabit
  - Measured one-way, double that for bi-directional (full duplex)
- High, Moderate, Low – A function of the instance size and EBS optimization
  - Not all created equal – Test with iperf if it's important!
- Use placement groups when you need high and consistent instance to instance bandwidth
- All traffic limited to 5 Gb/s when exiting EC2

When talking to S3 from an EC2 instance, you will be capped at 5 Gbps

EBS Performance

- Instance size affects throughput
- Match your volume size and type to your instance
- Use EBS optimization if EBS performance is important

Instance type	EBS-optimized by default	Max. bandwidth (MiB/s)*	Max. IOPS (16 KiB I/O size)*	Throughput (Mbps)**
c3.xlarge		62.5	4,000	500
c3.2xlarge		125	8,000	1,000
c3.4xlarge		250	16,000	2,000
c4.large	Yes	62.5	4,000	500
c4.xlarge	Yes	93.75	6,000	750
c4.4xlarge	Yes	250	16,000	2,000
c4.8xlarge	Yes	500	32,000	4,000
d2.xlarge	Yes	93.75	6,000	750
d2.2xlarge	Yes	125	8,000	1,000
d2.4xlarge	Yes	250	16,000	2,000
d2.8xlarge	Yes	500	32,000	4,000
g2.2xlarge		125	8,000	1,000
i2.xlarge		62.5	4,000	500
i2.2xlarge		125	8,000	1,000
i2.4xlarge		250	16,000	2,000
m3.xlarge		62.5	4,000	500
m3.2xlarge		125	8,000	1,000
m4.large	Yes	56.25	3,600	450
m4.xlarge	Yes	93.75	6,000	750
m4.2xlarge	Yes	125	8,000	1,000
m4.4xlarge	Yes	250	16,000	2,000
m4.10xlarge	Yes	500	32,000	4,000

EBS optimization is fast because it uses a separate connection that that used to communicate with other EC2 instances, this is good when you are doing a lot of data storage heavily. It is enabled by default on a lot of instances.

Summary: Getting the Most Out of EC2 Instances

- Choose HVM AMIs
- Timekeeping: use TSC
- C state and P state controls
- Monitor T2 CPU credits
- Use a modern Linux OS
- NUMA balancing
- Persistent grants for I/O performance
- Enhanced networking
- Profile your application

Virtualization Themes

- Bare metal performance goal, and in many scenarios already there
- History of eliminating hypervisor intermediation and driver domains
  - Hardware assisted virtualization
  - Scheduling and granting efficiencies
  - Device pass through

Next Steps

- Visit the Amazon EC2 documentation
- Launch an instance and try your app!