Performance Tuning

Red Hat Enterprise Linux 6

Jeremy Eder, Sr. Software Engineer

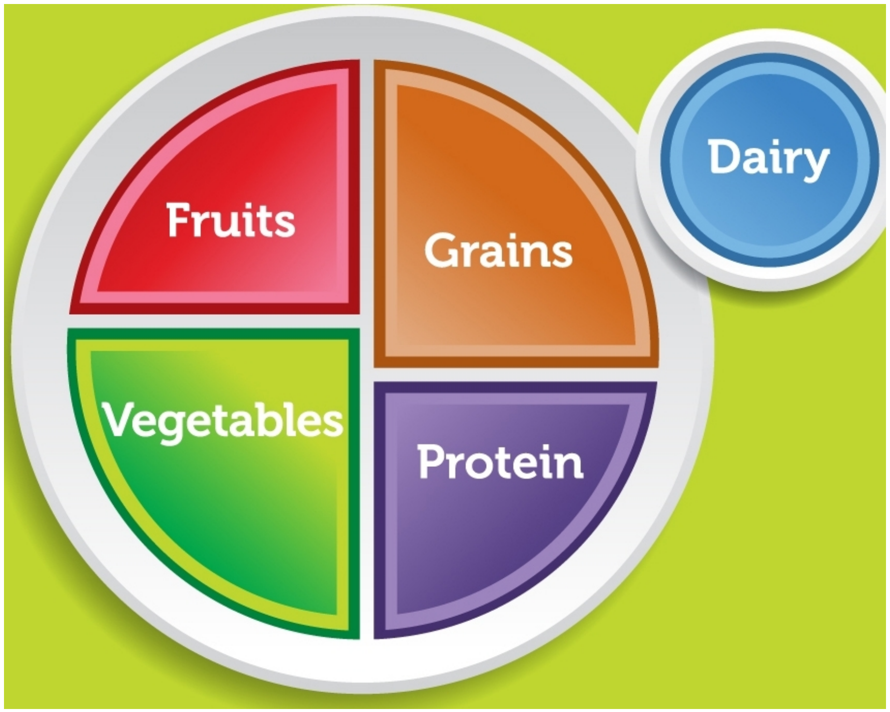
Performance Engineering

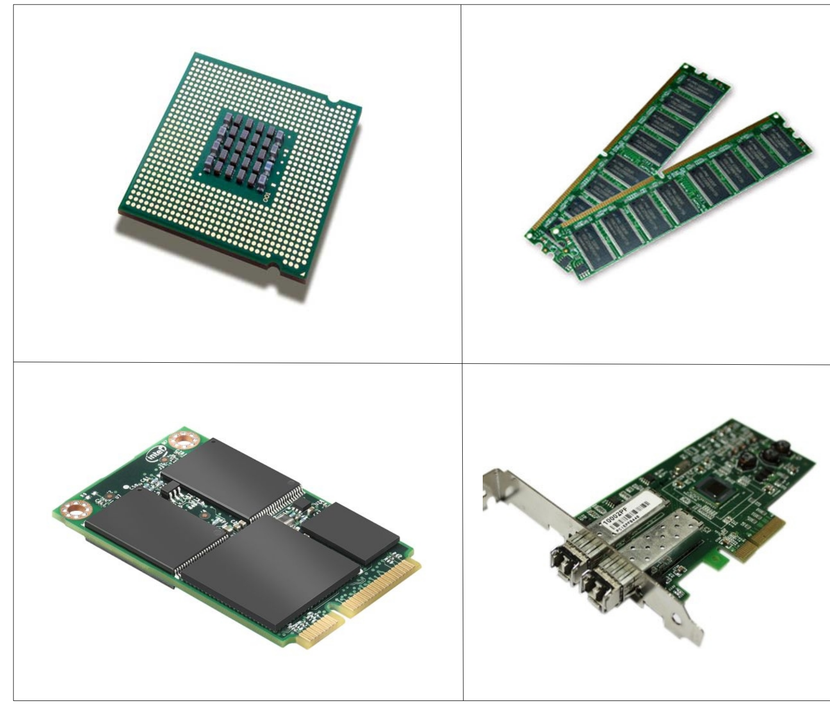
Agenda

(Real) Agenda...

- Performance Tuning Theory
- RHEL6 Performance Improvements
- tuned
- cgroups Architecture and Deep-dive
- CPU Performance Tuning/Power Management
- Memory/NUMA Performance Tuning
- Network Performance Tuning

Performance Tuning Food Groups





Best Practices for Tuning

- Be patient, accurate and methodical...it's iterative
  - Include measurement plumbing into your application
- Know your hardware (`hwloc`)
  - This cannot be stressed enough...
- Be aware of the latency vs throughput balancing act
  - The enemy of extreme low-latency: batching/coalescing
- Avoid disk when you can...
- Use tools such as systemtap and perf

Basic OS Setup

- Disable unnecessary services and use runlevel 3
- Avoid disk access in the critical path
  - Consider disabling filesystem journaling, {a,dir}time
  - Ever consider running swapless ? (vm.swappiness)
- Be aware of BIOS making Power Management decisions
  - processor.max_cstate=1
  - tuned: latency-performance profile (/dev/cpu_dma_latency)

(Real) Agenda...

- Performance Tuning Theory
- RHEL6 Performance Improvements
- tuned
- cgroups Architecture and Deep-dive
- CPU Performance Tuning/Power Management
- Memory/NUMA Performance Tuning
- Network Performance Tuning

Performance Improvements in RHEL6

Performance enhancements in every component

Component	Feature
CPU/Kernel	NUMA – Ticketed spinlocks; Completely Fair Scheduler; Extensive use of Read Copy Update (RCU) Scales up to 64 VCPUs per guest
Memory	Large memory optimizations: Transparent Huge Pages is ideal for virtualization
Networking	vhost-net – a kernel based virtio w/ better throughput and latency. SRIOV for ~native performance, RFS/XPS
Block	AIO, MSI, scatter gather.

RHEL6 “tuned-adm” profiles

 # tuned-adm list

Available profiles:

- **default**
- **latency-performance**
- **throughput-performance**
- **enterprise-storage**
- **virtual-host, virtual-guest ***

Example

```
# tuned-adm profile enterprise-storage
```

Recommend “virtual-host” w/ KVM, others workload-specific.

** New for RHEL6.3*

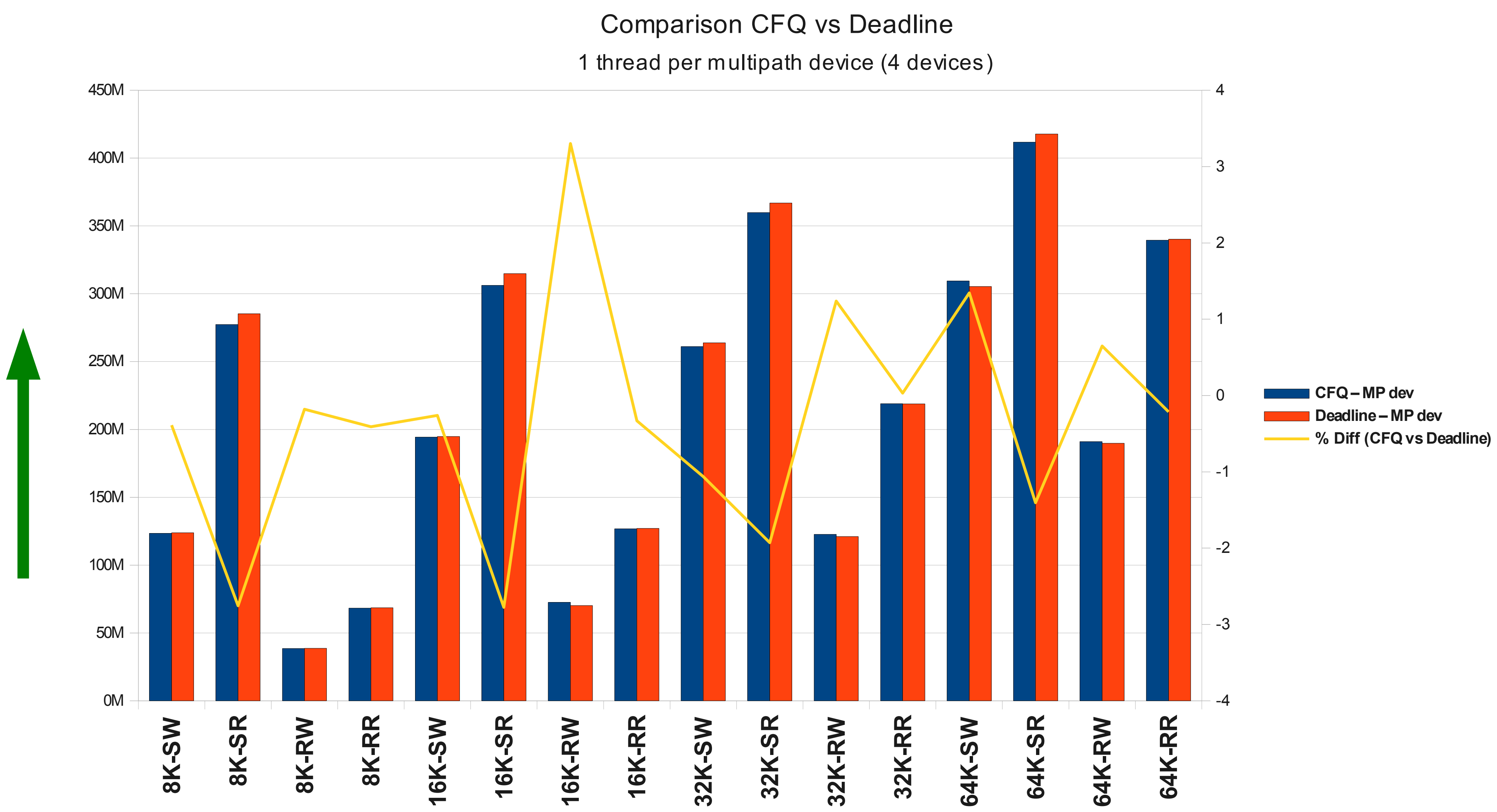
tuned profile summary...

Tunable	default	enterprise-storage	virtual-host	virtual-guest	latency-performance	throughput-performance
kernel.sched_min_granularity_ns	4ms	10ms	10ms	10ms		10ms
kernel.sched_wakeup_granularity_ns	4ms	15ms	15ms	15ms		15ms
vm.dirty_ratio	20% RAM	40%	10%	40%		40%
vm.dirty_background_ratio	10% RAM		5%			
vm.swappiness	60		10	30		
I/O Scheduler (Elevator)	CFQ	deadline	deadline	deadline	deadline	deadline
Filesystem Barriers	On	Off	Off	Off		
CPU Governor	ondemand	performance			performance	performance
Disk Read-ahead		4x				

RHEL6 “enterprise-storage”

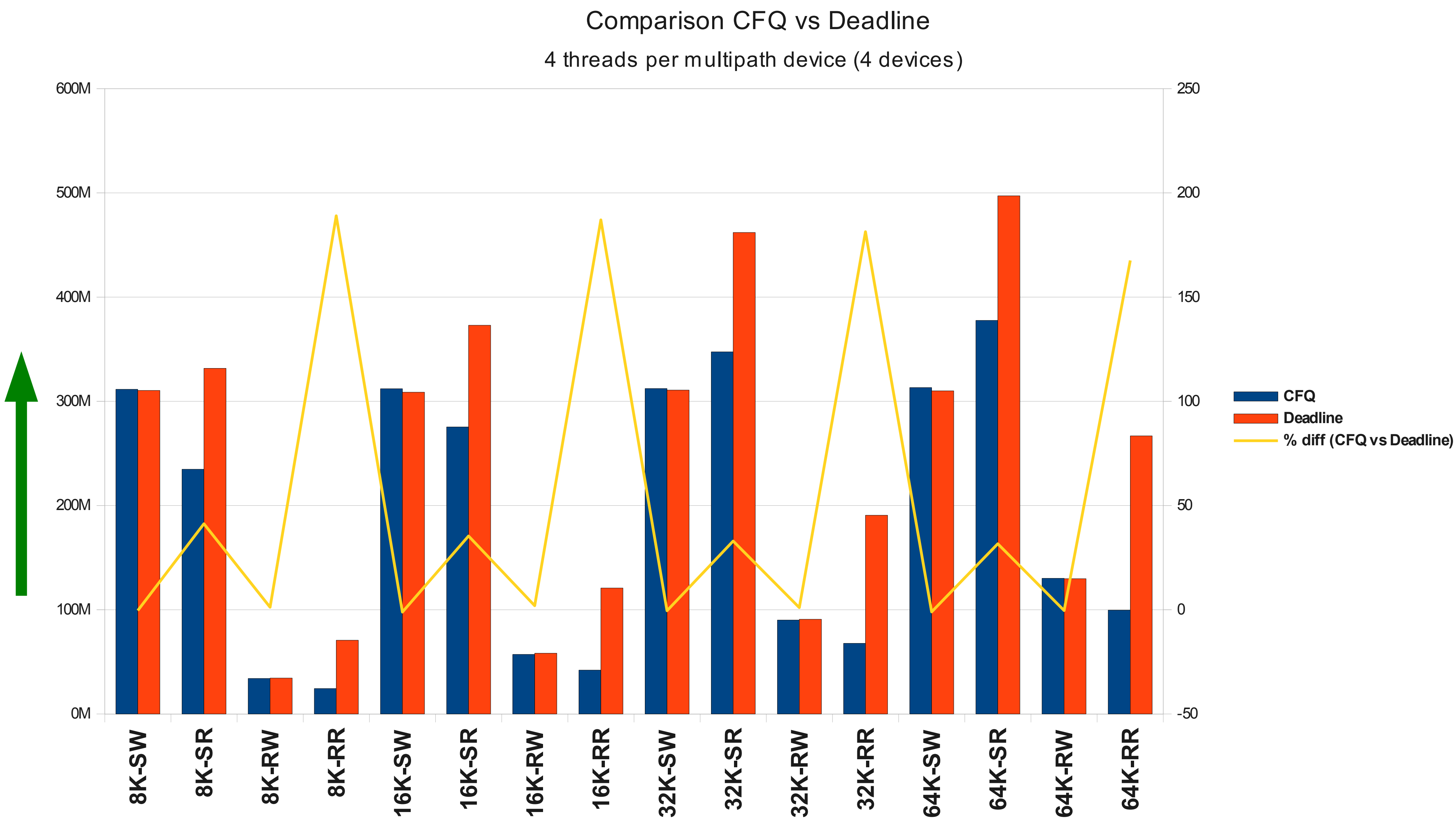
```
# tuned-adm profile enterprise-storage

kernel.sched_min_granularity_ns = 10000000
kernel.sched_wakeup_granularity_ns = 15000000
vm.dirty_ratio = 40
ELEVATOR="deadline"
BARRIERS=off (for mounts other than root/boot vols)
```

CFQ vs Deadline – 1 thread per device (4 devices)

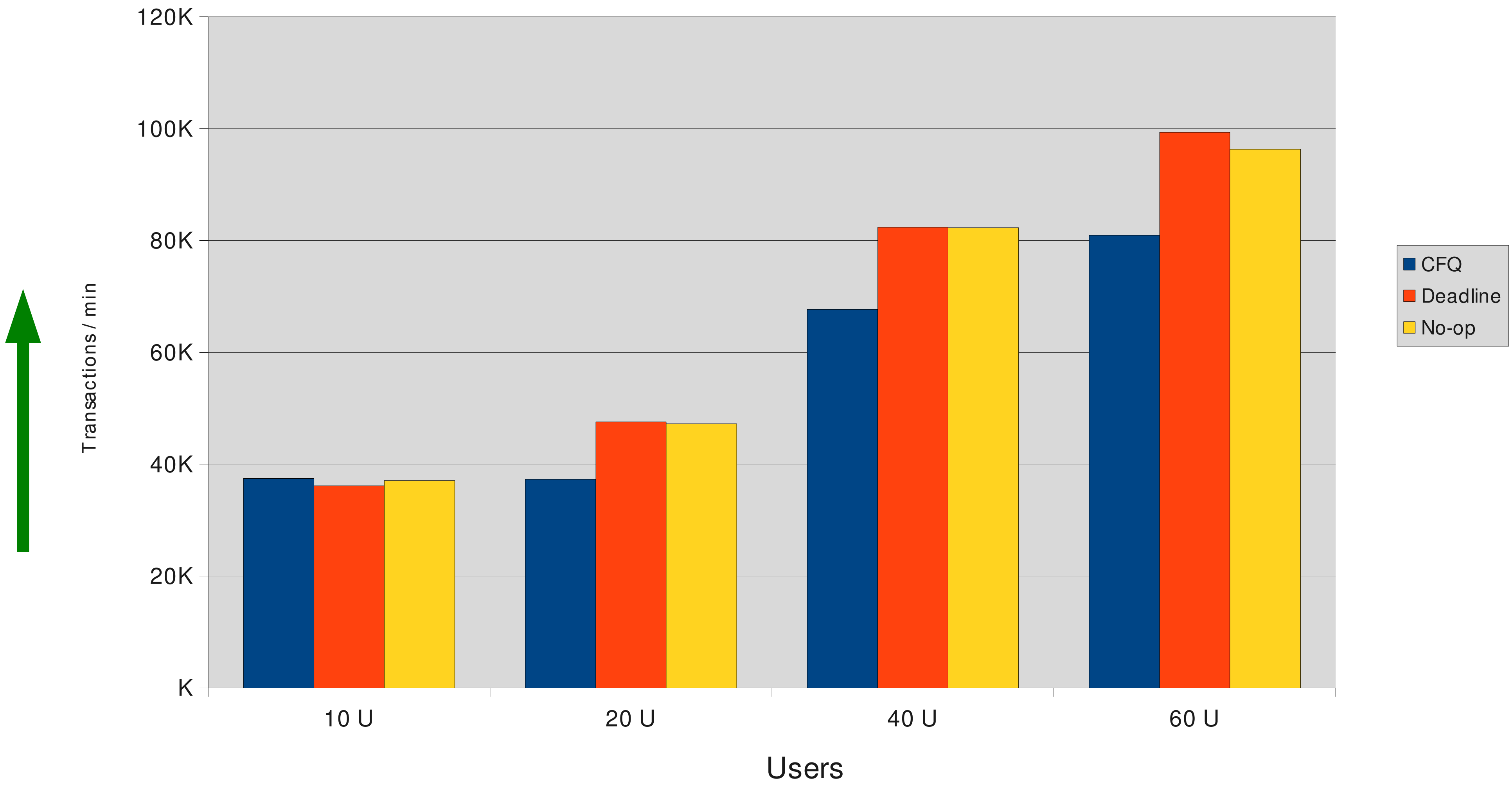


Jeremy Eder

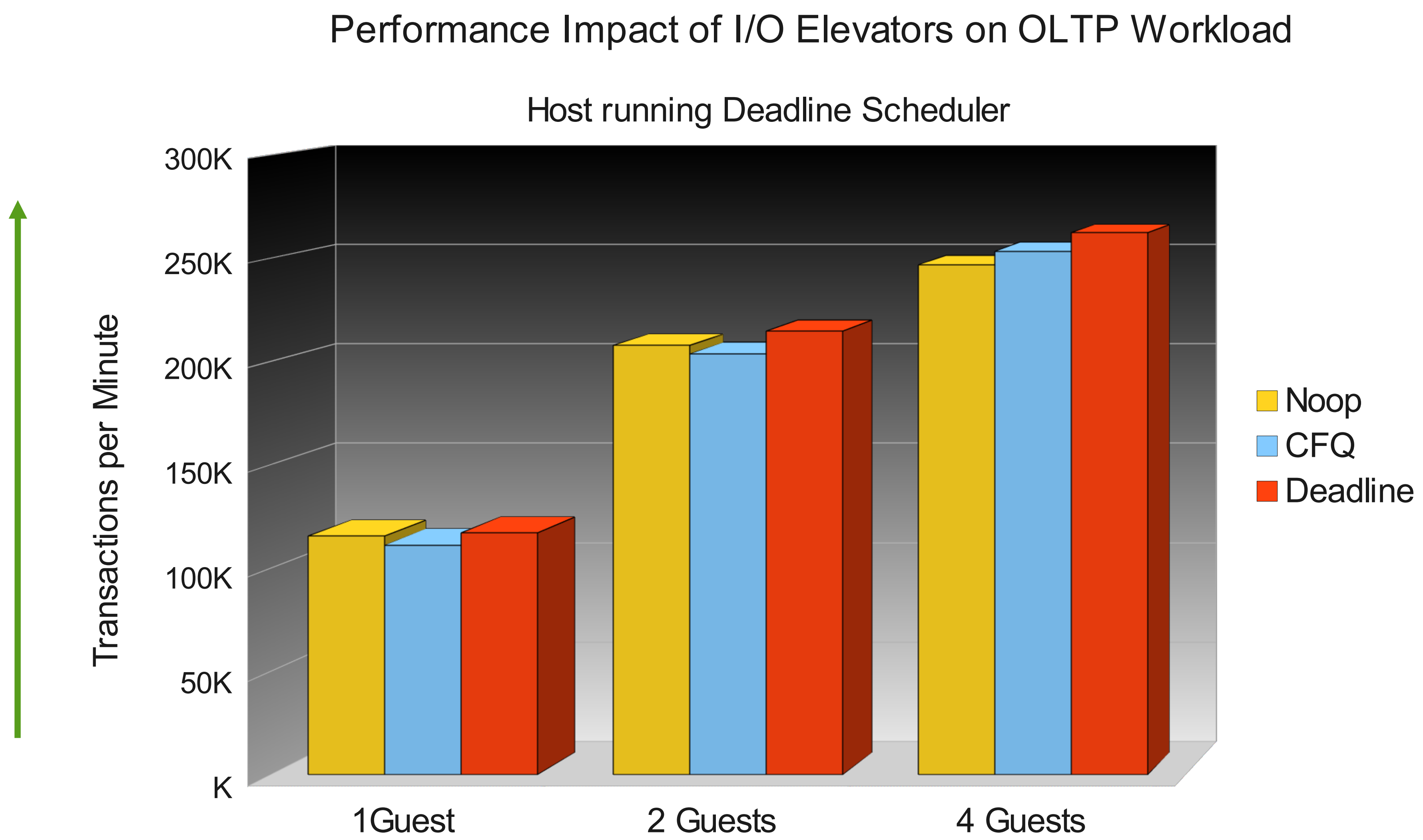
CFQ vs Deadline – 4 threads per device (4 devices)



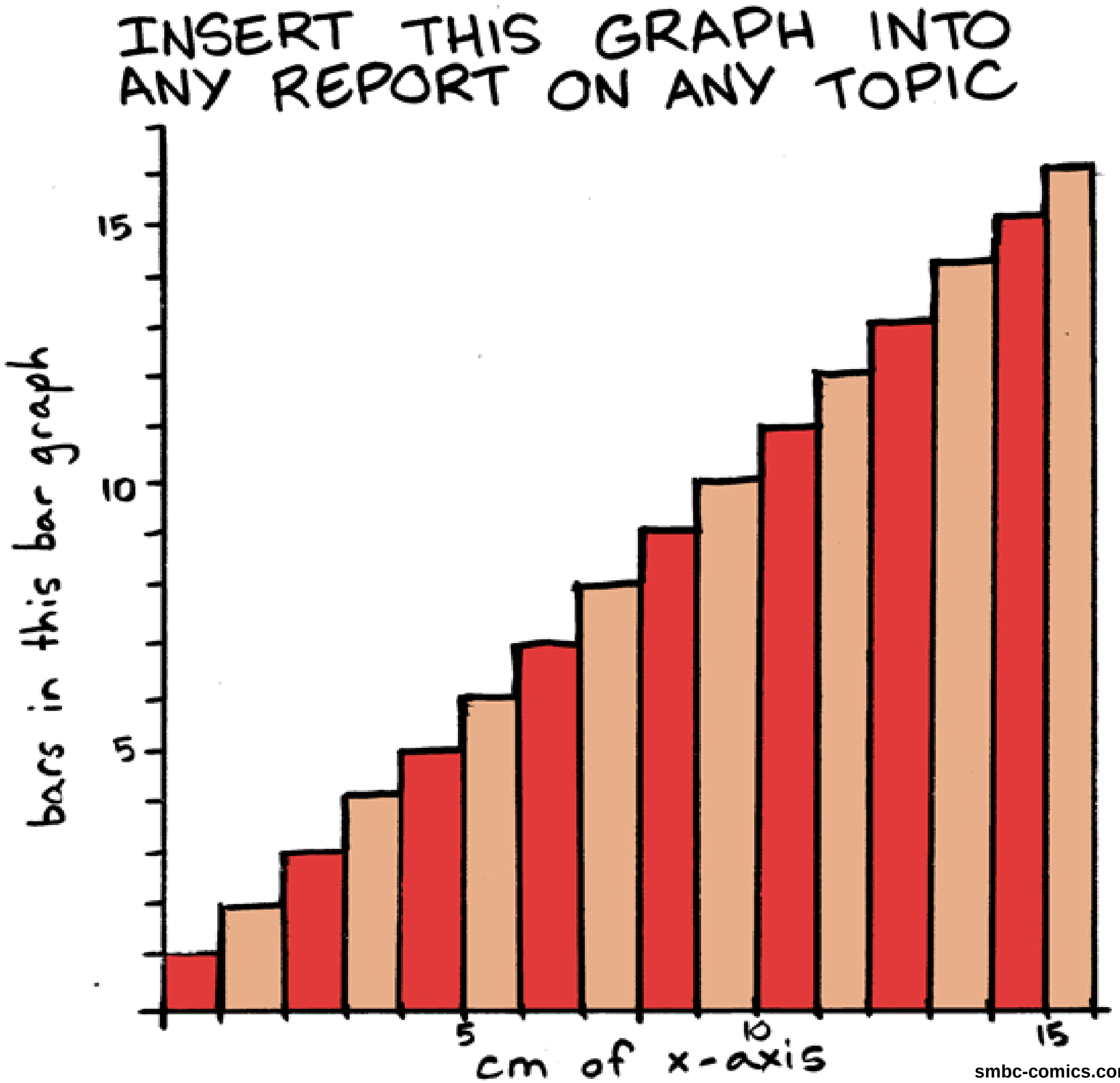
Impact of I/O Elevator - OLTP Workload



Virtualization Tuning I/O elevators - OLTP



Jeremy Eder



smbc-comics.com

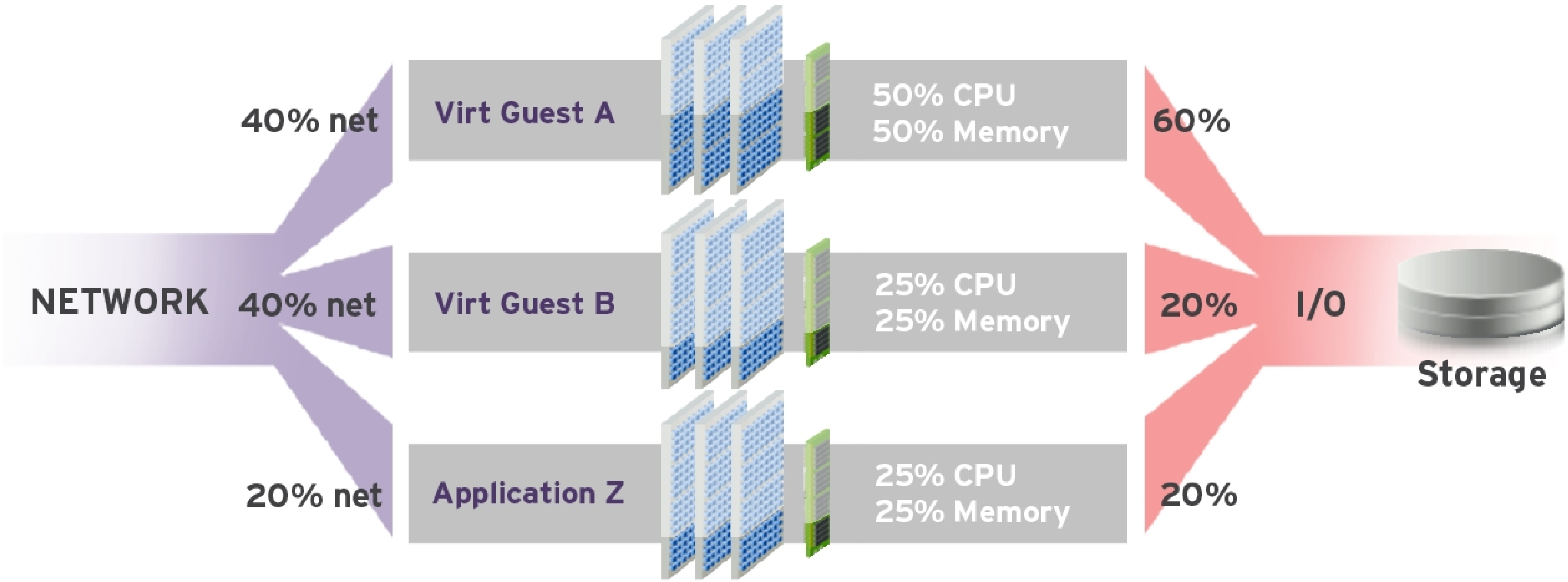
(Real) Agenda...

- Performance Tuning Theory
- RHEL6 Performance Improvements
- tuned
- cgroups Architecture and Deep-dive
- CPU Performance Tuning/Power Management
- Memory/NUMA Performance Tuning
- Network Performance Tuning

Resource Management

Ability to manage large system resources effectively

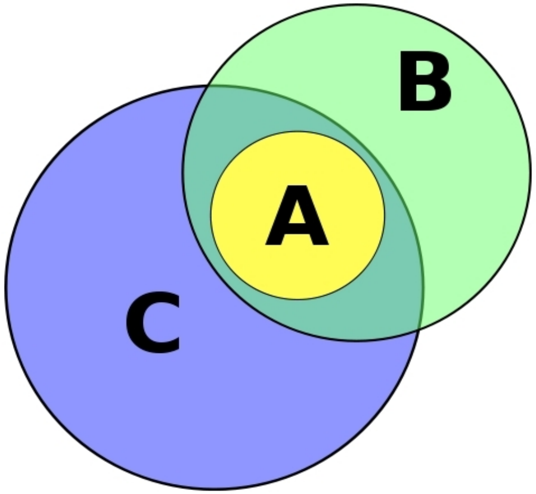
- Control Group (Cgroups) for CPU/Memory/Network/Disk
- Benefit: guarantee Quality of Service & dynamic resource allocation
- Ideal for managing any multi-application environment
  - From back-ups to the Cloud



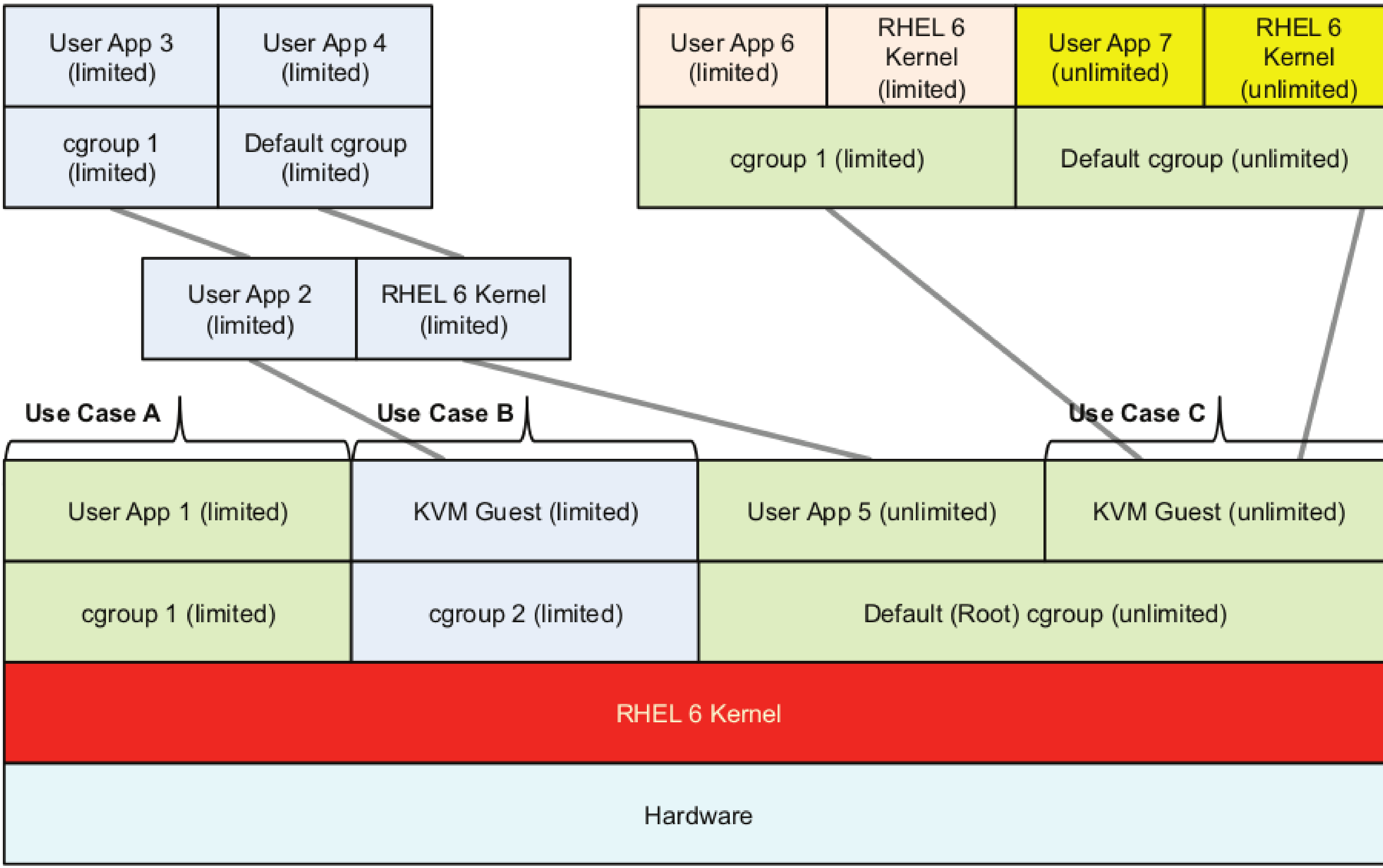
Jeremy Eder

Control Groups in RHEL6

- Mechanism for partitioning sets of tasks into hierarchical groups
  - Applying business policy to these groups
- Introduce new VFS mounted at /cgroup
  - Sub-directories define a new groups, arbitrarily nested
- cgroup “Resource Controllers” interact with the kernel to manage tunables within cgroups
- Allow the finite resources of a physical system to be “partitioned” at the process level, maintaining a defined quality of service for each cgroup

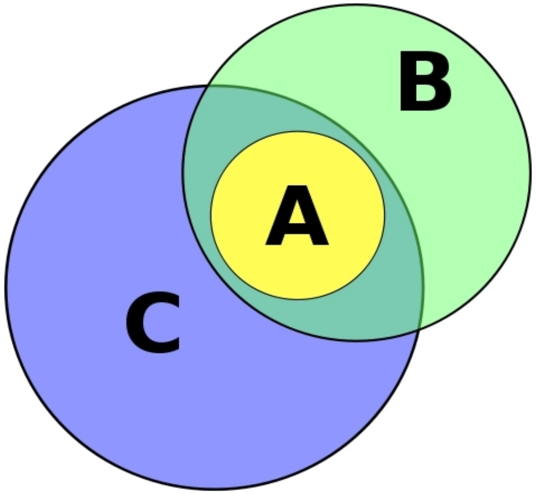


cgroups Architecture



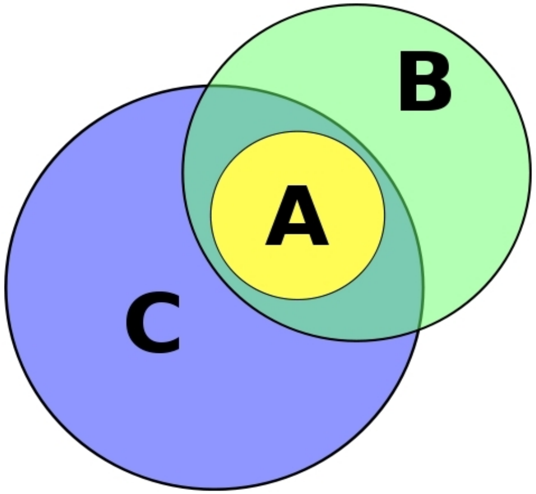
Control Groups in RHEL6 – Utilities 1

- **libcgroup** – parent RPM containing userspace utilities for managing cgroups.
  - `lscgroup` – list all mounted cgroups
  - `lssubsys` – list all known controllers
  - `cgget` – print parameters of a given cgroup



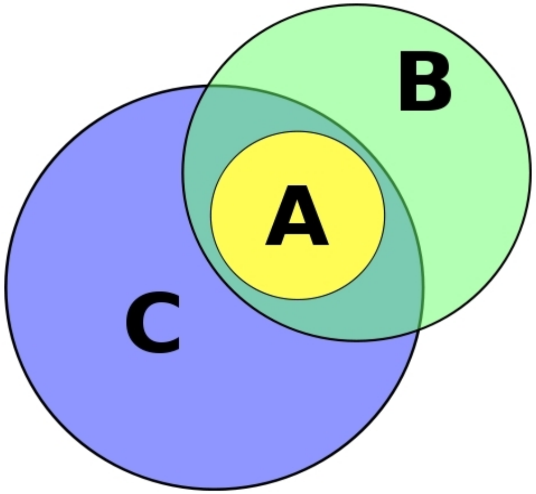
Control Groups in RHEL6 – Utilities 2

- **libcgroup** – parent RPM containing userspace utilities for managing cgroups.
  - `cgcreate` – create a new cgroup
  - `cgdelete` – delete a cgroup
  - `cgset` – set parameters of a cgroup
  - `cgexec` – execute a command in a given cgroup



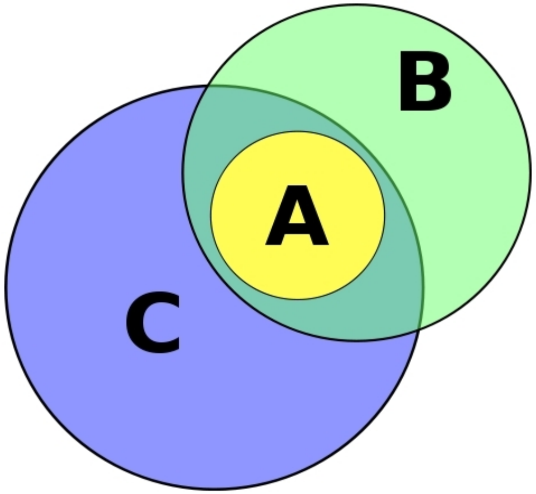
Control Groups in RHEL6 – Utilities 3

- **libcgroup** – parent RPM containing userspace utilities for managing cgroups.
  - `cgclassify` – move a PID into an existing cgroup
  - `cgsnapshot` – dump running cgroup heirarchy/parameters out to a file



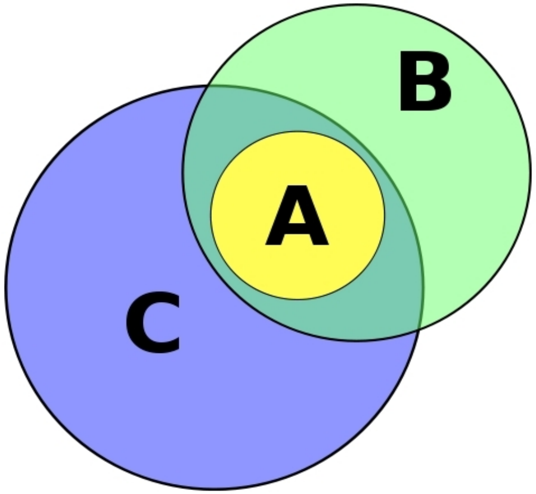
Control Groups in RHEL6 - Controllers

- blkio
- cpu, cpuacct
- cpuset
- devices
- freezer
- memory
- net_cls
- net_prio (new for RHEL6.3, for DCB)



Control Groups in RHEL6 - blkio

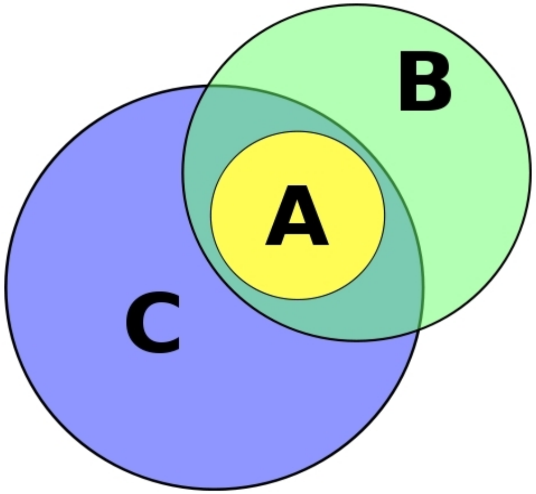
- blkio – applies policy to block I/O. “per cgroup, per-device” queues.
  - Proportional weight, similar to cpu controller “shares”
  - I/O throttling
  - Proportional weight requires CFQ
  - group_isolation=1 in RHEL6.2 (better fairness at cost of throughput.



- Examples:
  - `cgset -r blkio.weight=1000 demo1`
  - `cgset -r \`
    `blkio.throttle.write_bps_device=”8:0 10485760”`

Control Groups in RHEL6 - cpu

- cpu – schedules CPU access
  - Implemented via concept of “shares”
  - Shares are relative to each other
  - 3:1 same as 3000:1000
- Example:
  - 2 versions of postgres Production/Archival
  - Production instance should have priority access to CPU resources
  - `cgset -r cpuset.shares=1000 demo1`
- Controller scalability improvements in RHEL6.2 for large SMP systems

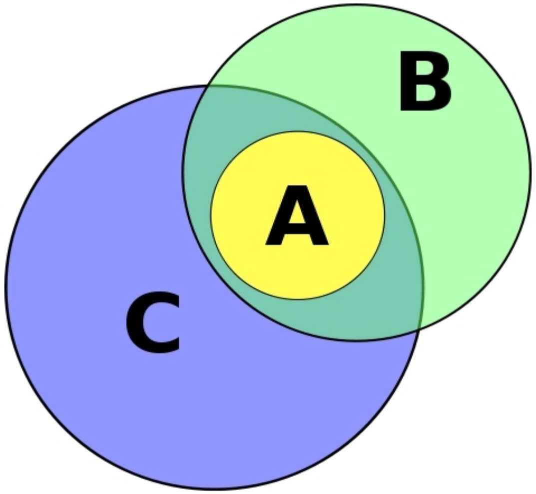


Control Groups in RHEL6 - cpuacct

- cpuacct – handles accounting of CPU usage by tasks in cgroups
  - Publishes user vs system time consumed by a cgroup
- Example:
  - `cat /cgroup/cpuacct/cpuacct.usage_percpu`

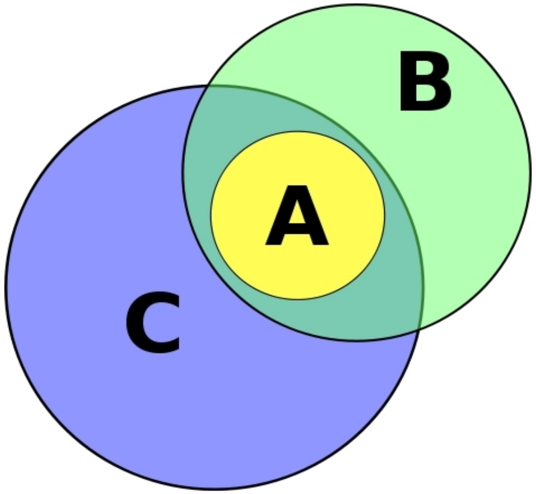
```
65557307889 64937863570 56062767350
49901532710 67833661127 46246490661
23286576086 23175803123
```

  - 8 Cores on this system...above expressed in nanoseconds, and units of USER_HZ=1000.
  - Time used in seconds = `65557307889/10^9`
    - ~65 seconds used on core 0, by this cgroup

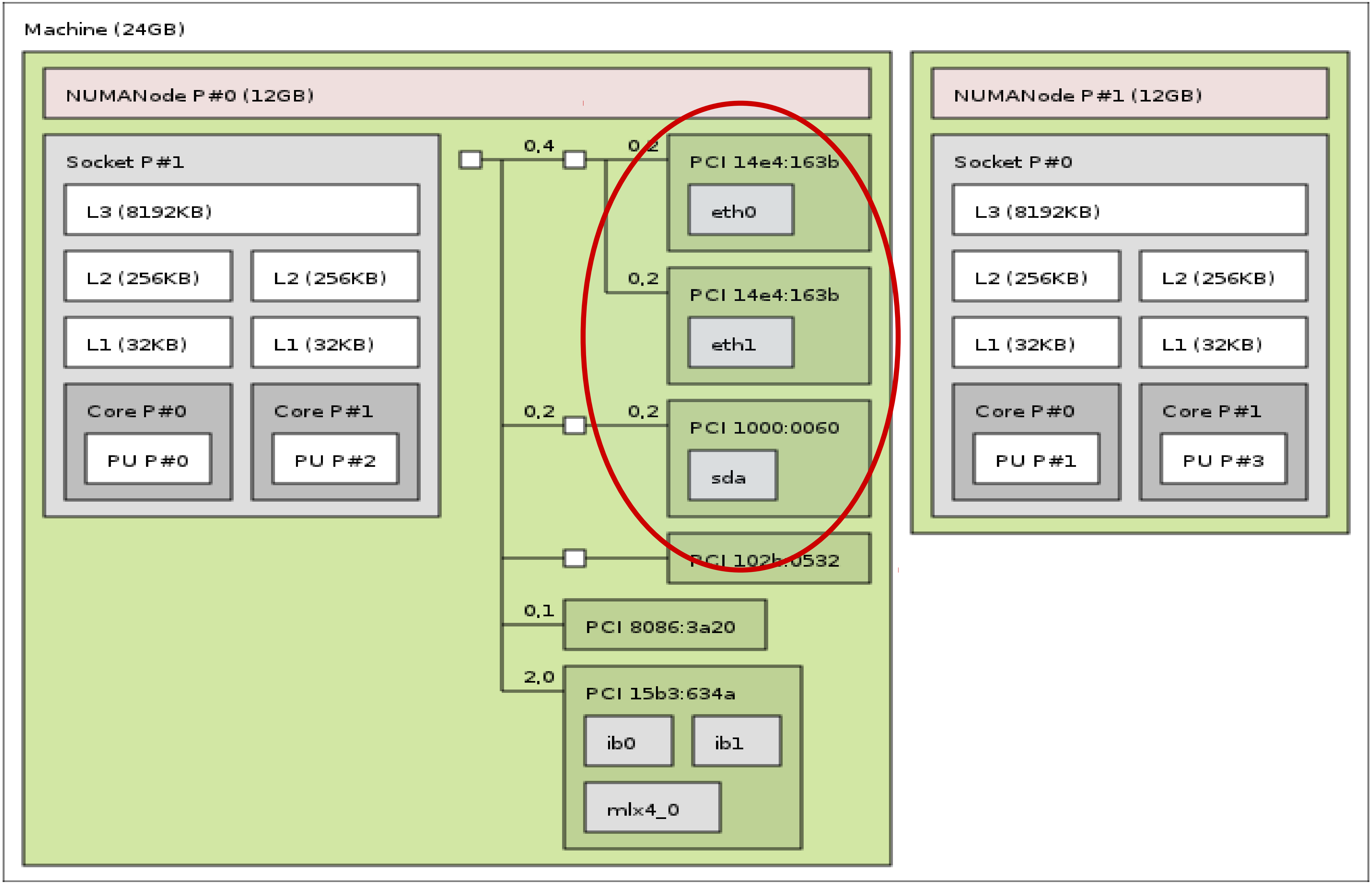


Control Groups in RHEL6 - cpuset

- cpuset – assigns CPUs/Memory nodes to cgroups
  - Refer to “Know your Hardware” slide (next)
- Example:
  - `cgset -r \`
    `cpuset.cpus=1,3,5,7,9,11,13,15,17,19,21,23`
    `demo1`
  - `cgset -r cpuset.mems=1 demo1`

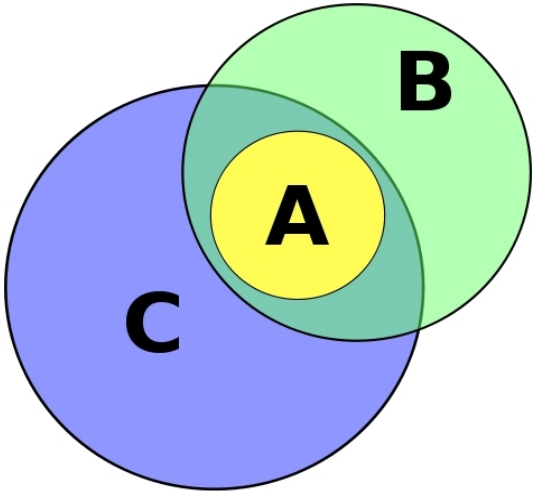


Know Your Hardware (hwloc/lstopo)



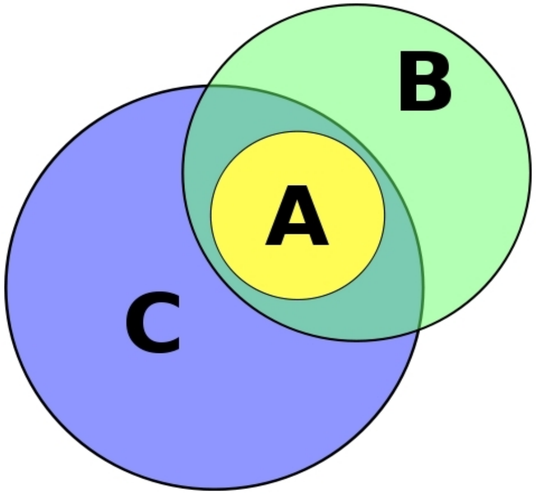
Control Groups in RHEL6 - memory

- memory - applies policy to memory allocation requests
  - Set upper-bounds
  - Maintain statistics
- memory controller overhead significantly reduced by over 37%
  - RHEL6.2 requires 8MB per 1GB, or 1GB for every 128GB physical RAM)
  - Further optimization underway
- Example:
  - `cgset -r memory.limit_in_bytes=2G demo1`



Control Groups in RHEL6 – net_cls

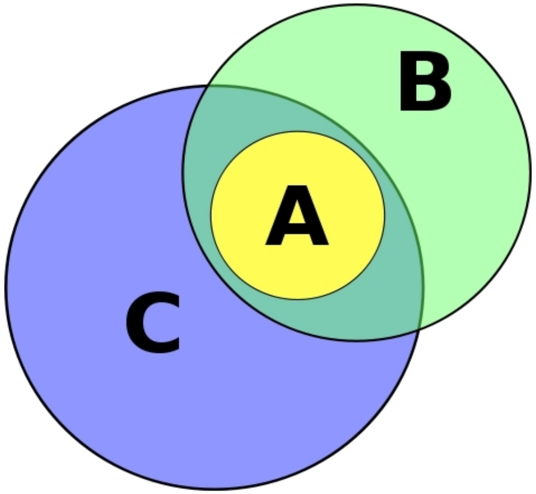
- memory – associate a cgroup with a classid that 'tc' utility creates/manages
  - Set upper-bounds
- Example:
  - `tc qdisc add dev eth1 root handle 10: htb default 10`
  - `tc class add dev eth1 parent 10:10 classid \ 10:10 htb rate 9gbit ceil 9gbit`
  - `tc filter add dev eth1 parent 10:0 protocol all prio 1 handle 1 cgroup`
  - `echo 0x100010 > /cgroup/net_cls/net_cls.classid`
- This sets a 9gbit throughput limit on the cgroup



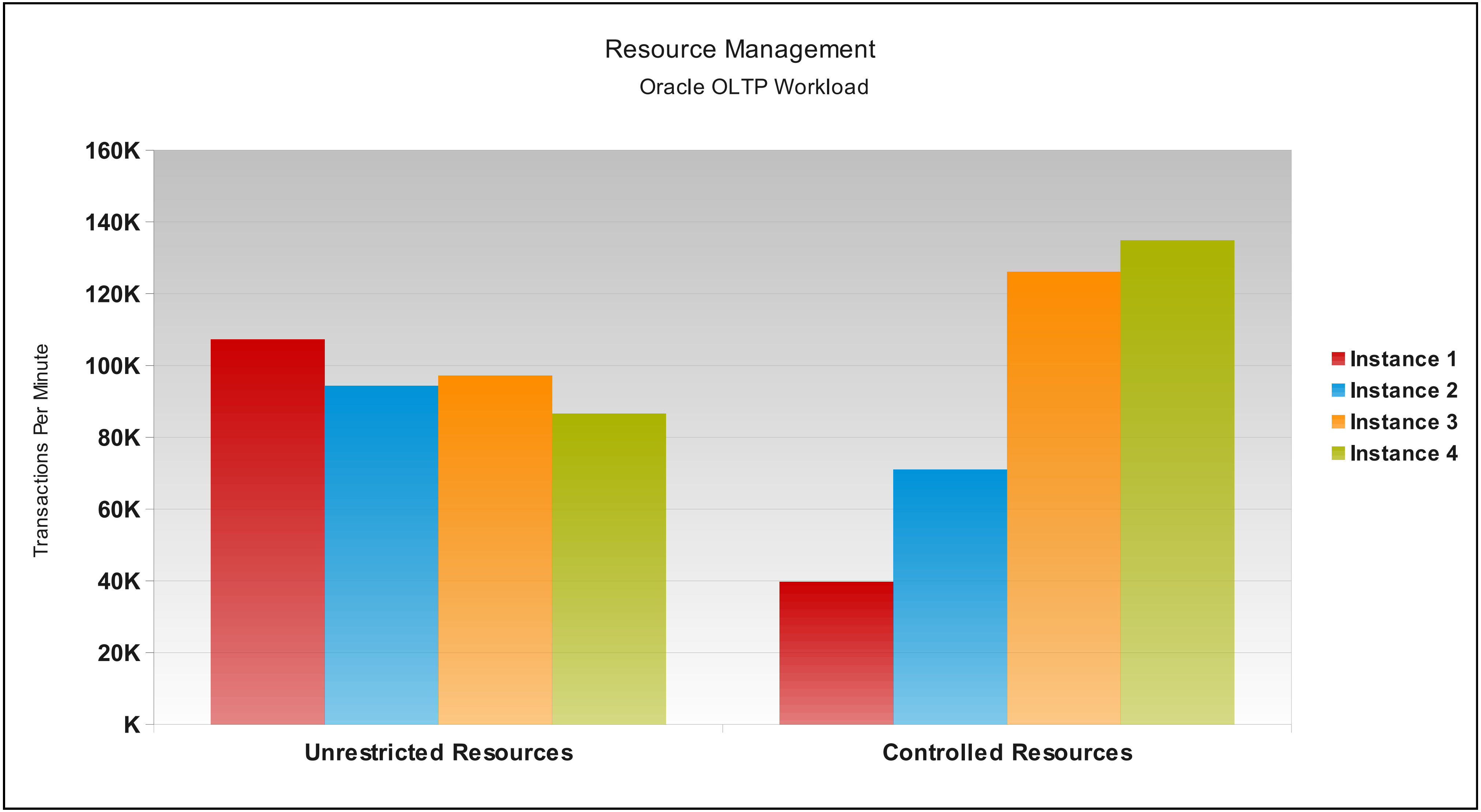
Putting it **all** together...

cgconfig.conf

```
group demo1 {
    cpu {
        cpu.shares = 1000;
    }
    cpuset {
        cpuset.cpus = 1,3,5,7,9,11,13,15,17,19,21,23;
        cpuset.mems = 1;
    }
    blkio {
        blkio.weight = 1000;
    }
    memory {
        memory.limit_in_bytes = 2G;
    }
}
```

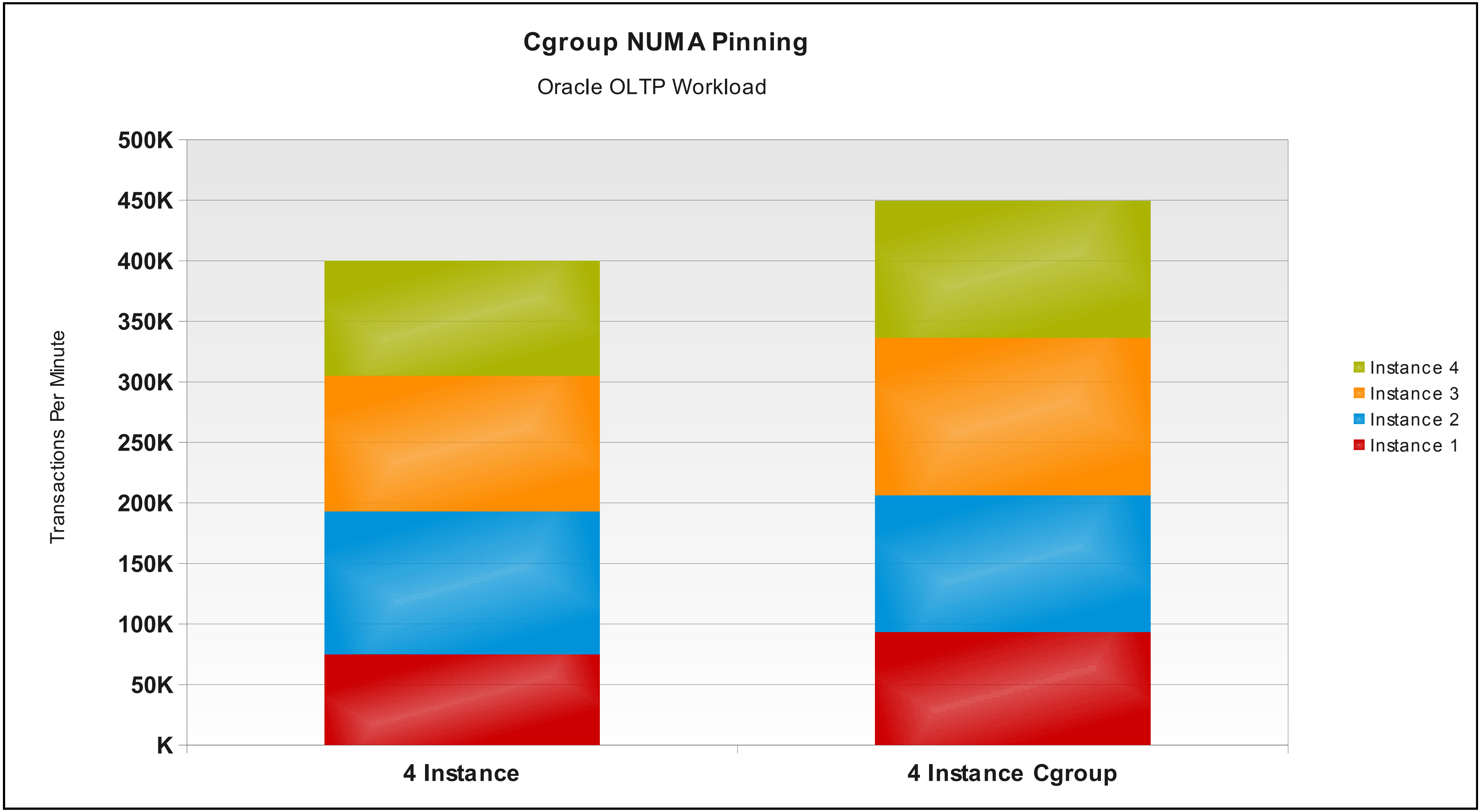


Cgroup – Resource management



Jeremy Eder

Cgroup – NUMA pinning



(Real) Agenda...

- Performance Tuning Theory
- RHEL6 Performance Improvements
- tuned
- cgroups Architecture and Deep-dive
- CPU Performance Tuning/Power Management
- Memory/NUMA Performance Tuning
- Network Performance Tuning

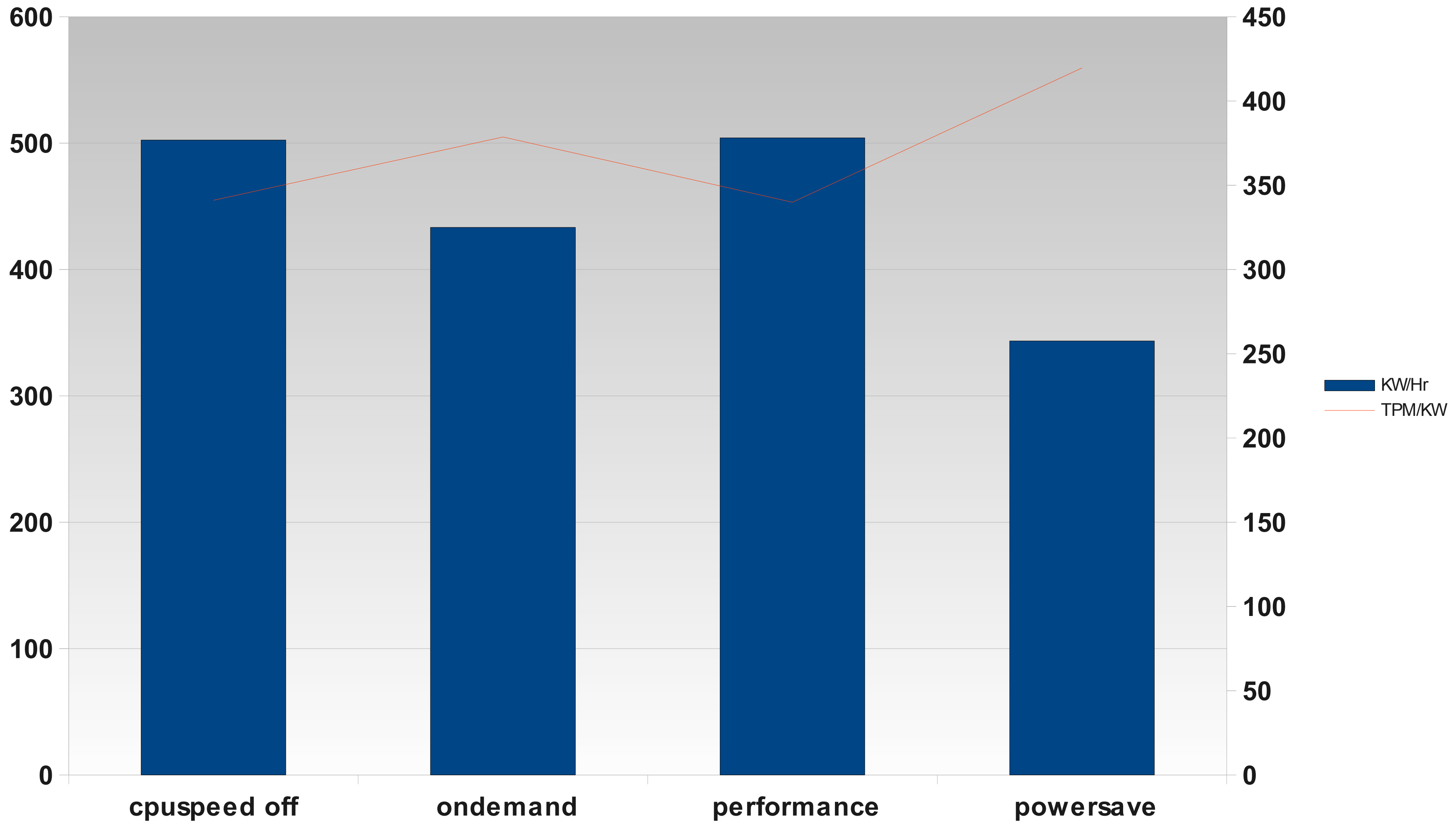
A word about CPU Power Mgmt... C&P-states

- "You *probably* don't *always* need what you paid for..."
  - Recent chips from major vendors slow themselves down
    - Called P-states
  - Or lower voltages/disable portions of the core like timers
    - Called C-states
  - And spin them back up on-demand.
- Examples:
  - Use powertop, or turbostat from kernel source

CPU Tuning

- Variable frequencies
  - Multiple cores
  - Power saving modes (cpuspeed governors)
    - performance
    - ondemand
    - Powersave
- Examples:
  - `echo "performance" > \`
    `/sys/devices/system/cpu/cpu0/cpufreq/scaling_governor`
  - Best of both worlds – cron jobs to configure the governor mode using tuned-adm
  - `tuned-adm profile {default,latency-performance}`

Performance / Power consumption



8 core system, normal conditions...

pk	cr	CPU	%c0	GHz	TSC	%c1	%c3	%c6	%pc3	%pc6
			0.03	2.35	2.67	0.05	0.03	99.88	0.27	90.07
0	0	1	0.38	2.83	2.67	0.08	0.40	99.14	0.27	90.08
0	1	3	0.01	2.30	2.67	0.04	0.00	99.95	0.27	90.08
0	2	5	0.01	2.64	2.67	0.02	0.00	99.97	0.27	90.08
0	8	7	0.01	2.28	2.67	0.02	0.00	99.97	0.27	90.08
1	0	0	0.03	1.62	2.67	0.06	0.00	99.91	0.27	90.07
1	1	2	0.04	1.60	2.67	0.05	0.00	99.91	0.27	90.07
1	2	4	0.02	1.61	2.67	0.03	0.00	99.95	0.27	90.07
1	8	6	0.02	1.60	2.67	0.04	0.00	99.94	0.27	90.07
1	9	8	0.02	1.57	2.67	0.03	0.00	99.95	0.27	90.07

Same system, with C-states locked @ C0

pk	cr	CPU	%c0	GHz	TSC	%c1	%c3	%c6	%pc3	%pc6
			100.00	2.93	2.67	0.00	0.00	0.00	0.00	0.00
0	0	1	100.00	2.93	2.67	0.00	0.00	0.00	0.00	0.00
0	1	3	100.00	2.93	2.67	0.00	0.00	0.00	0.00	0.00
0	2	5	100.00	2.93	2.67	0.00	0.00	0.00	0.00	0.00
0	8	7	100.00	2.93	2.67	0.00	0.00	0.00	0.00	0.00
1	0	0	100.00	2.93	2.67	0.00	0.00	0.00	0.00	0.00
1	1	2	100.00	2.93	2.67	0.00	0.00	0.00	0.00	0.00
1	2	4	100.00	2.93	2.67	0.00	0.00	0.00	0.00	0.00
1	8	6	100.00	2.93	2.67	0.00	0.00	0.00	0.00	0.00

Sources of CPU interference in core Linux code

- Any case where kernel management functions will take CPU time from a pure CPU bound user task running pinned to an isolated CPU with no other contending user tasks.

- Global Inter-processor Interrupts (IPIs)
- Global work queue scheduling
- Global kthread scheduling

```
# ps -emo pid,pcpu,psr,nice,cmd,rtprio,policy
```

- Global Timers

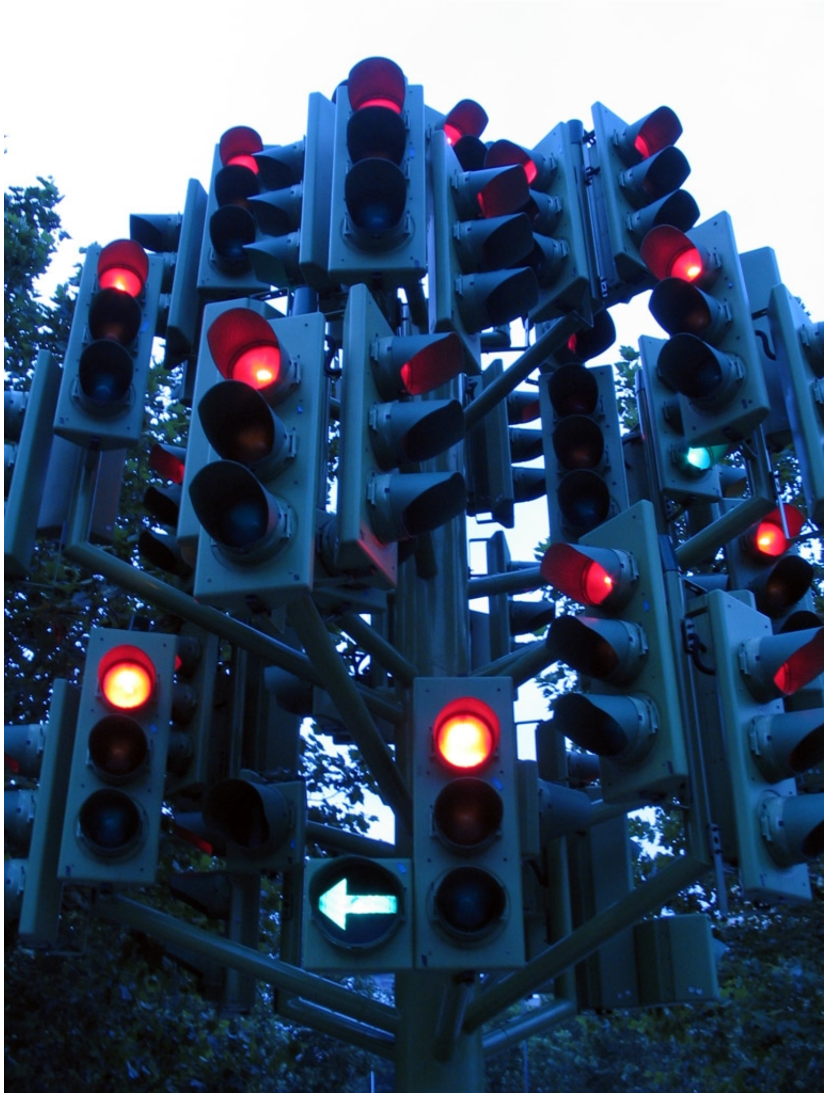
- `https://github.com/gby/linux/wiki`

Completely Fair Scheduler - Overview

- New in RHEL6, replaced O(1) scheduler in RHEL5
- What does it do...
  - Decides what runs, when and for how long.
  - The scheduler is what makes multitasking possible (Virtual Runtime)
  - CFS has to make decisions given imperfect/partial information... “likely”
    - CPUs do the same thing “branch prediction”

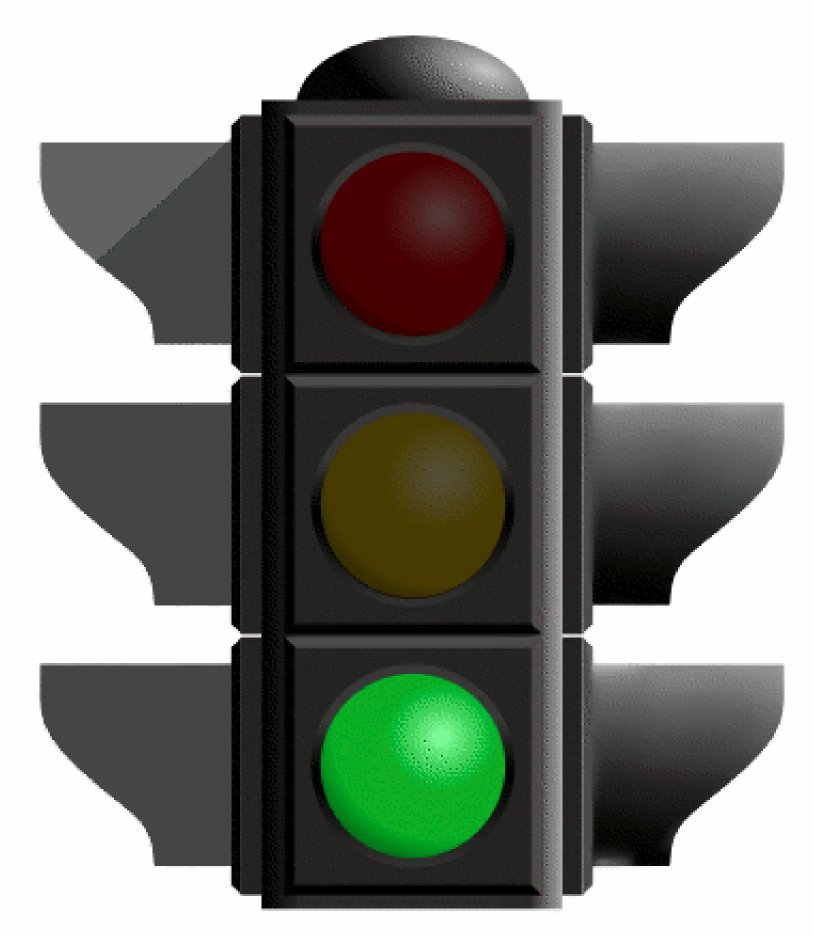
Completely Fair Scheduler – Conflicting Goals...

- Latency vs Throughput Balancing Act...
  - A running process means another has to wait. The waiter incurs this latency
  - But the running process gets a bunch of work (throughput) done
- CFS decision making process is a Red-Black tree of runnable tasks, taking into account Process Priority



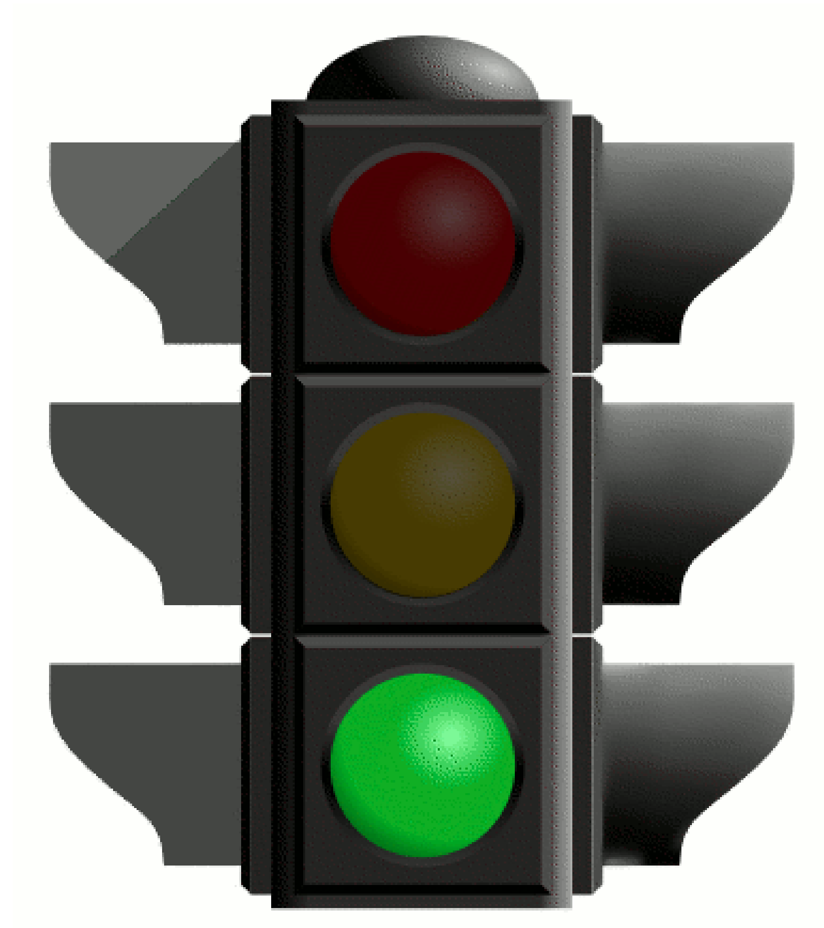
Completely Fair Scheduler – No Timeslices

- Well, more correctly...Timeslices not pre-defined/hardset like in O(1)...
- Based on load of system (number of runnable tasks)
  - 100 runnable tasks of equal priority, each gets something like 1/100th of the available CPU time
  - CFS takes into account each process's priority. Higher priority tasks get more CPU time
  - Certain scheduler policies (like realtime) take precidence over non-realtime tasks, regardless of their priority.



Completely Fair Scheduler – Red Black Tree

- The scheduler builds a future timeline of what will run
- It can do this because the next task to run is the one with the least virtual runtime
- Tasks move from right-to-left along this rbtree. Next task to run is in the left-most position. New tasks are inserted according to their vruntime.
- Relevant function are
  - sched.c pick_next_task

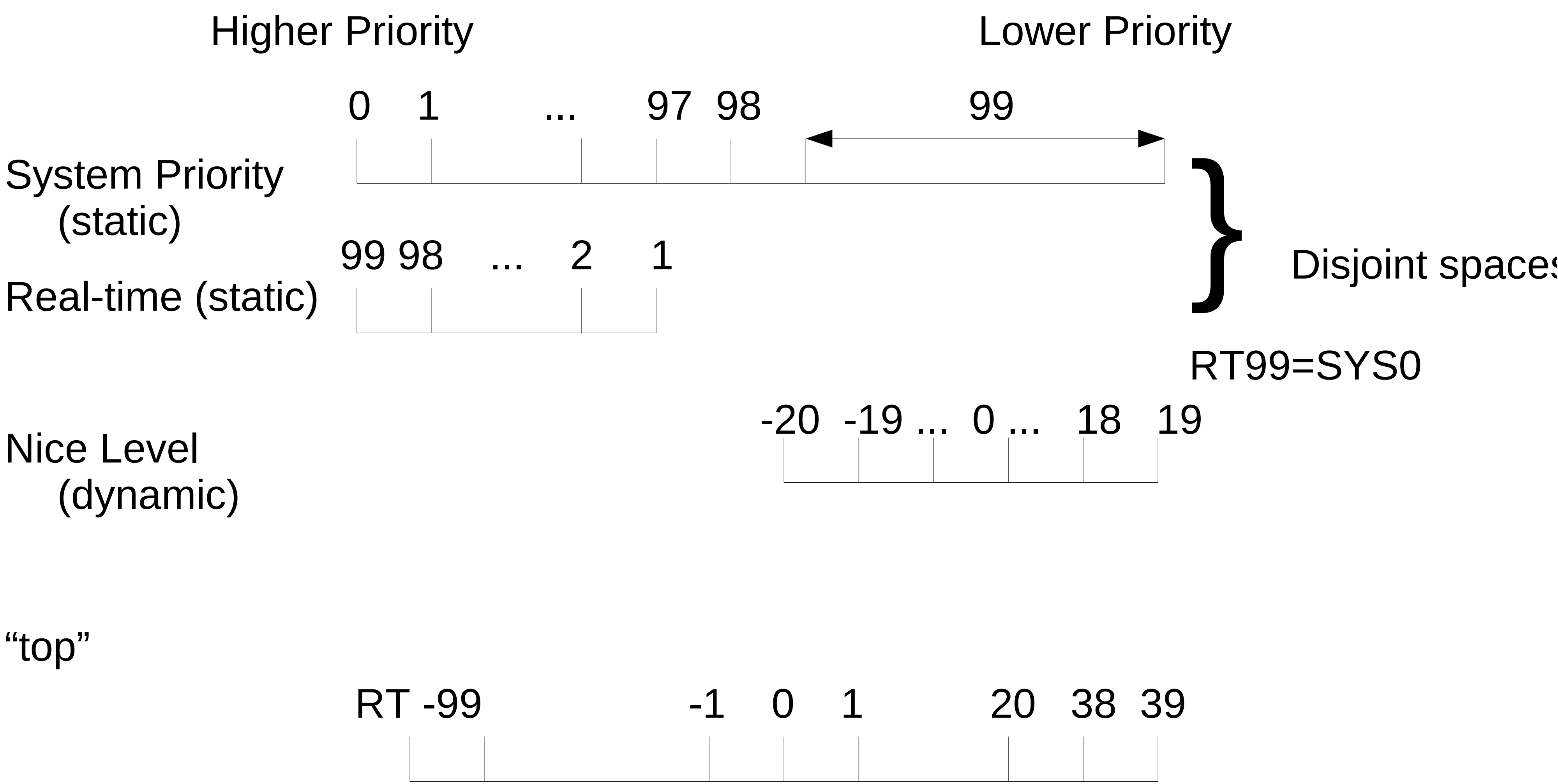


Scheduler Policies

- TS SCHED_NORMAL (aka SCHED_OTHER, the default policy)
- FF SCHED_FIFO (realtime policy, first-in-first-out)
  - Don't set your RTPRIO to 99. This will starve out kernel threads that need to run sometimes.
  - There is no way to fully isolate a core for 100% userspace processing. Recent study in a previous slide...
- RR SCHED_RR, same as FIFO but with a defined quantum
  - SCHED_RR only useful with > 1 tasks of same priority.
- B   SCHED_BATCH, ISO SCHED_ISO, IDL SCHED_IDLE
- Change programmatically, or with `chrt`

```
# ps -emo pid,pcpu,psr,nice,cmd,rtprio,policy
```

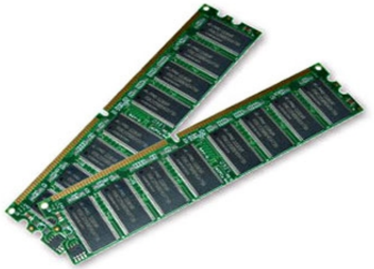
Completely Fair Scheduler
Decisions, Decisions...



(Real) Agenda...

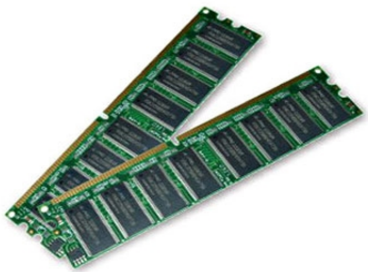
- Performance Tuning Theory
- RHEL6 Performance Improvements
- tuned
- cgroups Architecture and Deep-dive
- CPU Performance Tuning/Power Management
- Memory/NUMA Performance Tuning
- Network Performance Tuning

Memory Tuning - Hugepages



- Typical Linux memory pages are 4KB. Hugepages are 512x larger, 2MB
  - This makes more effective use of limited transaction lookaside buffer (TLB) space, increases likelihood of TLB misses
  - Traditional Hugepages are not swappable
  - Examples:
    - `hugepages=4096 on kernel cmdline`
    - `echo "vm.nr_hugepages=4096" >> /etc/sysctl.conf`
    - `echo 4096 > /proc/sys/vm/nr_hugepages`

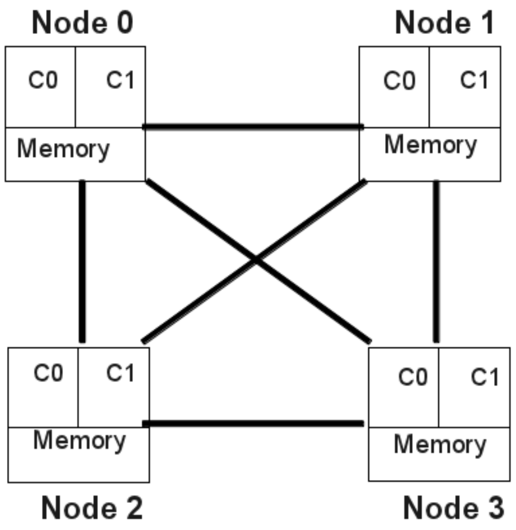
Memory Tuning – Transparent Hugepages



- Introduced in RHEL6.0
  - Anonymous memory only (swappable, can be disabled)
  - Can coexist with traditional hugepages
  - Does not require application support (anon memory).
  - Examples:
    - “It's in there...”

```
# grep -i huge /proc/meminfo
AnonHugePages:    1046528 kB
HugePages_Total:     4096
HugePages_Free:      4096
HugePages_Rsvd:         0
HugePages_Surp:         0
Hugepagesize:        2048 kB
```

Understanding NUMA



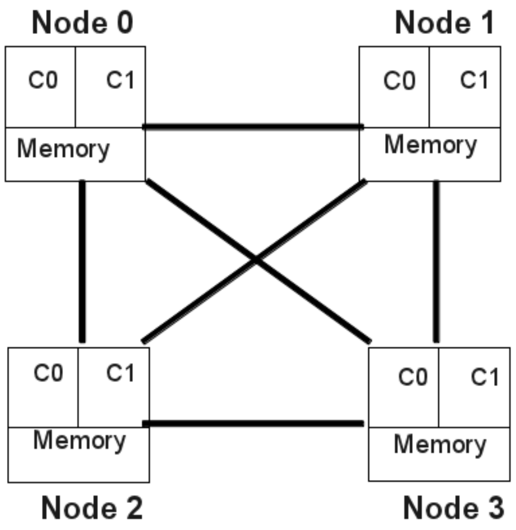
- Multi-socket/Multi-core Architecture used for scaling
  - RHEL5/6 Completely NUMA Aware
  - Additional, significant performance gains by

    enforcing NUMA locality.
- How do you enforce NUMA locality ?
  - `numactl -c1 -m1 ./command`
    - Command executes on CPUs in socket 1
    - And memory allocations are served out of memory node 1.
    - hwloc told me that socket 1 is “local” to memory node 1. Your hardware may vary.
    - Also demonstrated earlier in cgconfig.conf...

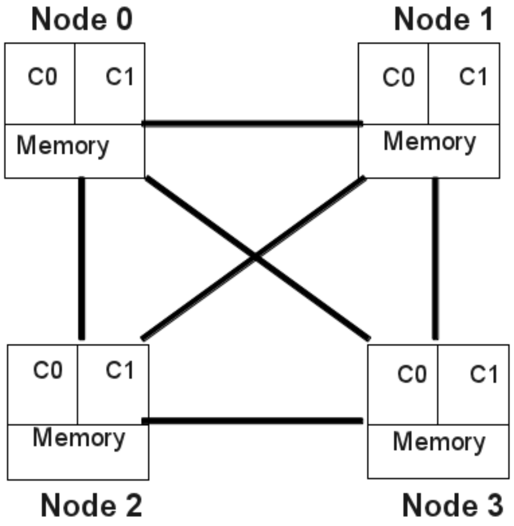
  - ***NUMA automation is an area of significant research and investment by both Red Hat and the community.***
    - ***AutoNUMA, schedNUMA, numad***

Understanding NUMA - nmstat

```
# ./nmstat

Per-node system memory usage (in MBs):

                          N0          N1
                  ----------  ----------
        MemTotal    24576.00    24563.09
         MemUsed    24497.63    24399.35
         MemFree       78.37      163.74
          Active       52.70       30.46
        Inactive    23750.83    22141.46
       FilePages    23794.39    22164.05
    Active(file)       45.44       23.91
  Inactive(file)    23748.84    22139.96
       AnonPages        9.13        7.93
etc...
```



Understanding NUMA - numad

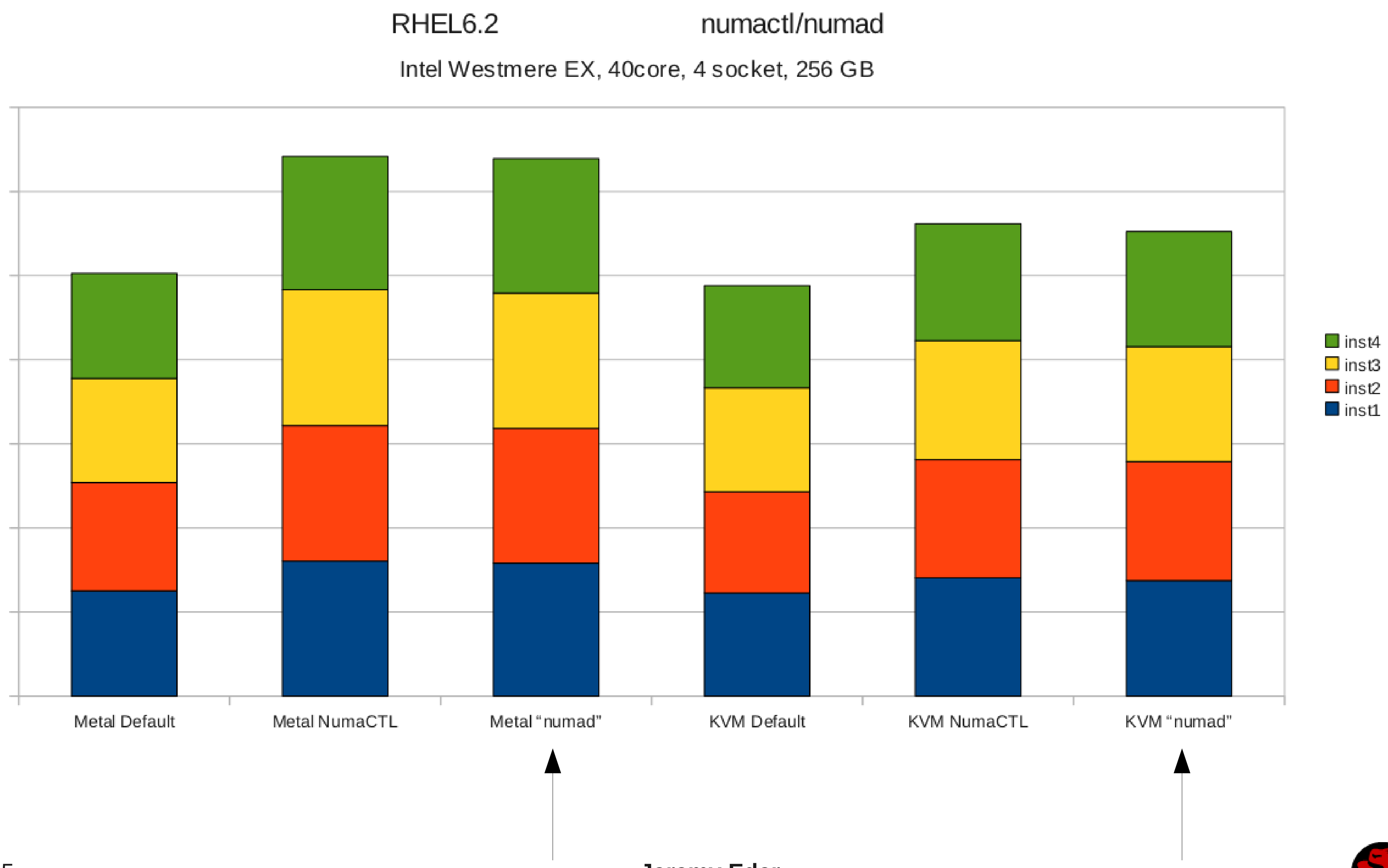


- New daemon that handles NUMA placement for long-running tasks...attempts to co-locate a process and it's mapped memory
  - Tech Preview in RHEL6.3, disabled by default
  - Shows significant performance improvements over scheduler placement
  - Often within 5% of hand-tuning/placement
  - Upstream kernel community still fleshing out NUMA placement algorithms
  - Can also be queried for best manual placement
- Example:
  - `# ./numad -w 1:100 (need 1 CPU and 100MB RAM)`

    `1`
  - `# ./numad -w 14:100 (box has 12 cores/socket)`

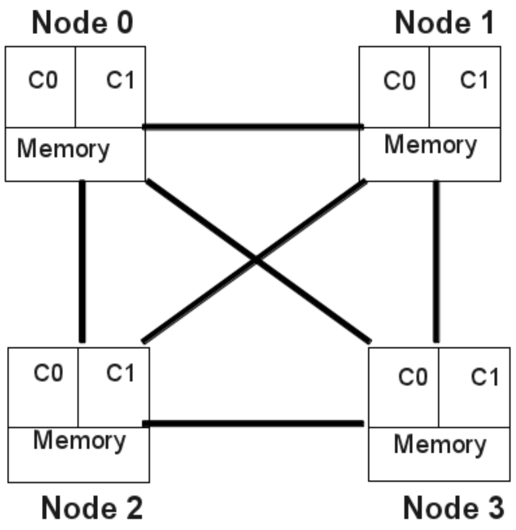
    `0-1`

numad results...



Jeremy Eder

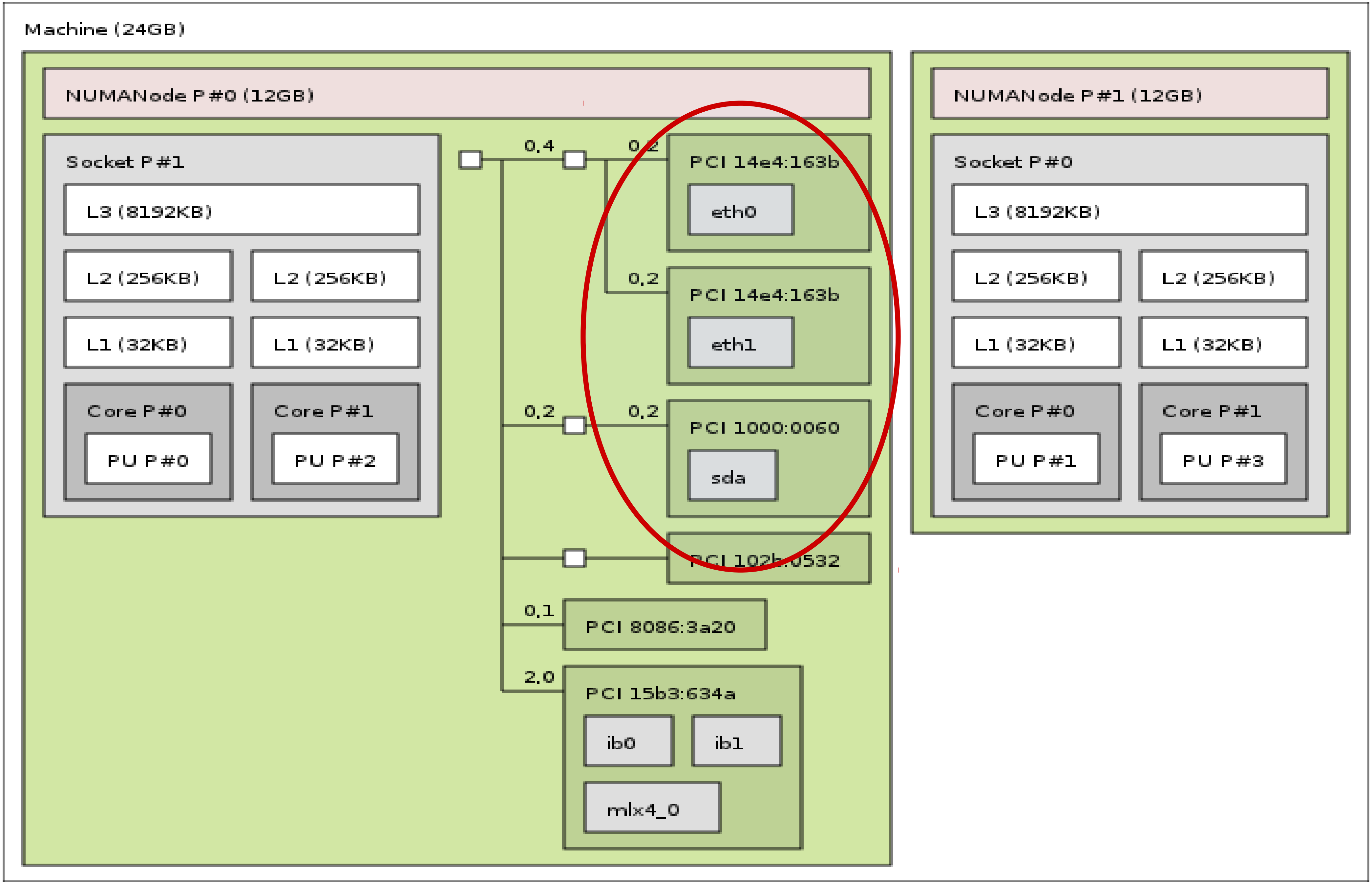
NUMA Topology and PCI Bus



- Server may have more than 1 PCI bus.
  - Optimal performance reduces/eliminates inter-node cross-talk. Install NIC in slot local to node that your application will run on. Use systemtap numa_faults.stp. irqbalance will learn this “soon”.
  - In the below case, the NIC is in PCI bus 0001. CPUs 1,3,5,7 are local to that PCI slot.

```
lspci output:

0001:06:00.0 Ethernet controller: Solarflare Communications
SFC9020 [Solarstorm]

# cat /sys/devices/pci0000\:00/0000\:00\:00.0/local_cpulist

  0,2,4,6

# cat /sys/devices/pci0001\:40/0001\:40\:00.0/local_cpulist

  1,3,5,7

# dmesg|grep "NUMA node"

  pci_bus 0000:00: on NUMA node 0 (pxm 0)

  pci_bus 0001:40: on NUMA node 1 (pxm 1)
```

Know Your Hardware (hwloc/lstopo)



(Real) Agenda...

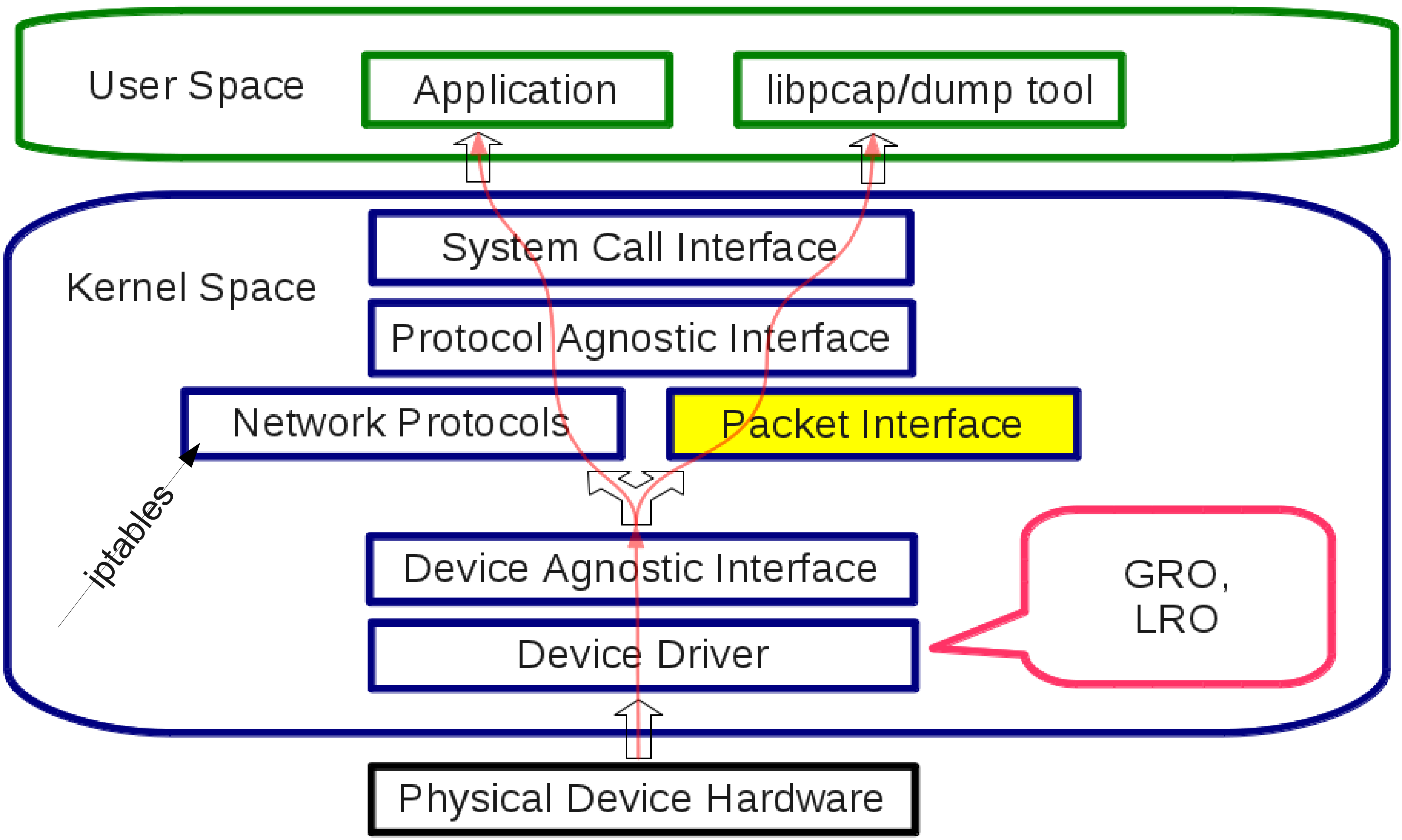
- Performance Tuning Theory
- RHEL6 Performance Improvements
- tuned
- cgroups Architecture and Deep-dive
- CPU Performance Tuning/Power Management
- Memory/NUMA Performance Tuning
- Network Performance Tuning

Network Determinism

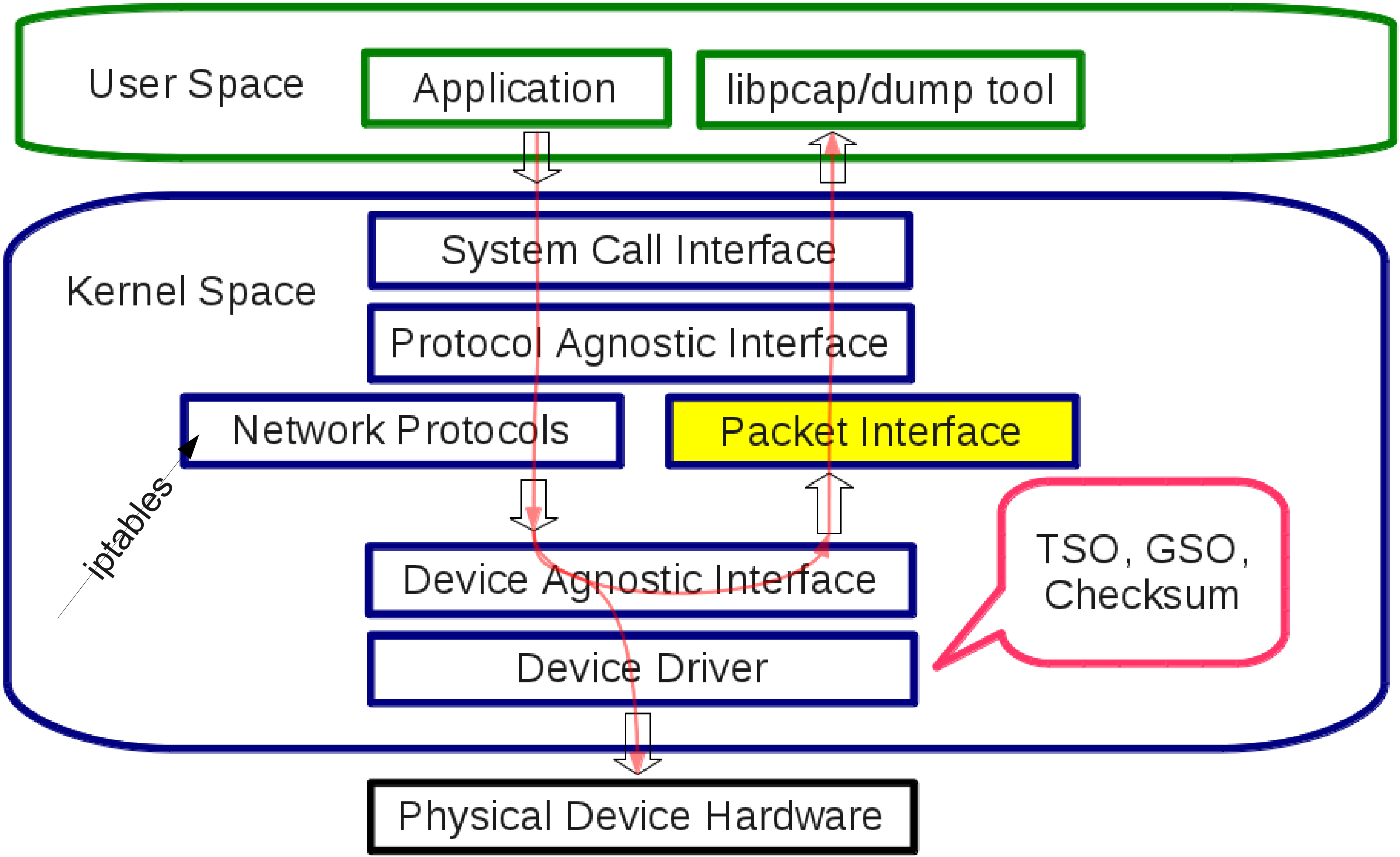
- Ensure you really need to use TCP
  - If so, experiment with TCP_NODELAY socket option (Nagle)
  - From Wikipedia:

```
if there is new data to send
  if the window size >= MSS and available data is >= MSS
    send complete MSS segment now
  else
    if there is unconfirmed data still in the pipe
      enqueue data in the buffer until an acknowledge is received
      ^^^ latency ^^^ more noticeable in high RTT/WAN environments.
    else
      send data immediately
```

Network Stack – Receive Side



Network Stack – Transmit Side



Bandwidth Delay Product/Long Fat Pipe

- Be careful/methodical with NIC offload options (ethtool -k)
  - Bandwidth Delay Product
    - Bandwidth * RTT
    - Get RTT from ping...
- Experiments with tc/netem
  - Add a static RTT delay

```
tc qdisc add dev eth0 root netem delay 100ms
```

  - Add a random delay (WAN variations)

```
tc qdisc change dev eth0 root netem delay 100ms 10ms
```

  - Simulate packet loss (again, usually a WAN)

```
tc qdisc change dev eth0 root netem loss 0.1%
```

  - View tc rules

```
tc qdisc show dev eth0
```

  - Remove netem rules

```
tc qdisc del dev eth0 root
```

Test Case (100Mbit pipe, 100ms RTT, 5GB)	Avg Time	Avg Tput
Untuned	245s	21Mbit
Sys buffer = BDP	90s	58Mbit
Sys buffer = 1.33*BDP	80s	67Mbit
Sys & app buffer = 1.33 * BDP	60s	95Mbit

My experiments with tc/netem and CBQ

- Add a static RTT delay

```
tc qdisc add dev eth0 root netem delay 100ms
```

- Add a random delay (WAN variations)

```
tc qdisc change dev eth0 root netem delay 100ms 10ms
```

- Simulate packet loss (again, usually a WAN)

```
tc qdisc change dev eth0 root netem loss 0.1%
```

- View tc rules

```
tc qdisc show dev eth0
tc filter show dev eth0
```

- Remove netem rules

```
tc qdisc del dev eth0 root
```

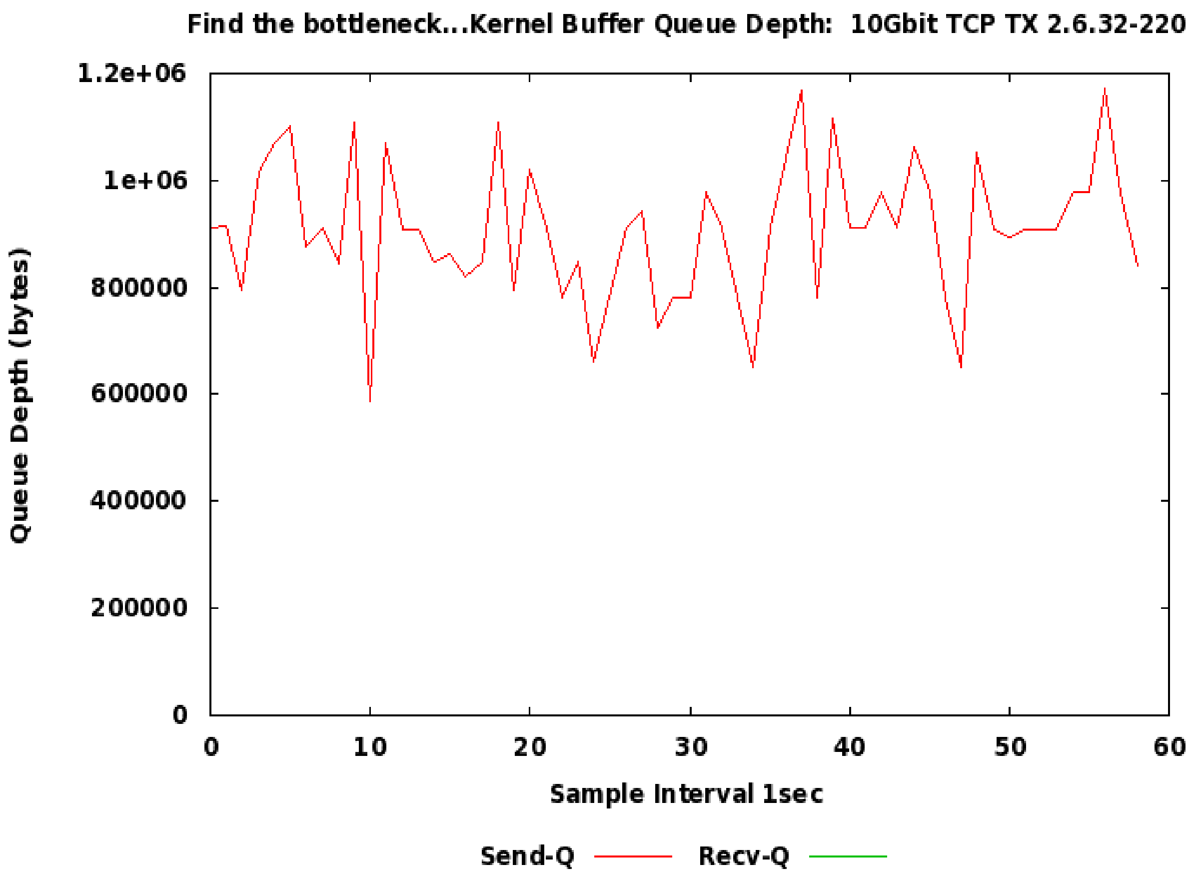
Note: incompatible with certain NIC driver offload implementations (e1000)

Buffer Bloat

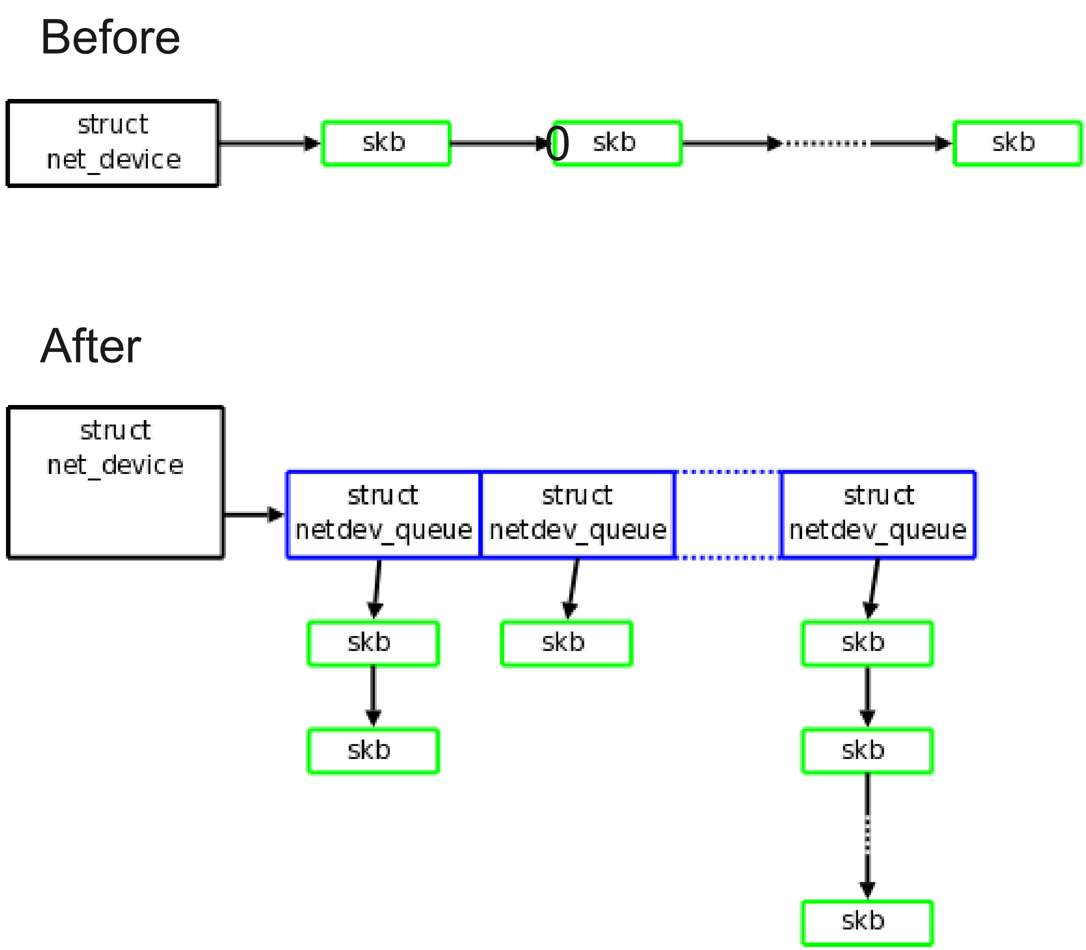
- Buffers are everywhere... www.bufferbloat.net
- What is buffer bloat ?
  - Latency caused by excessive buffering
  - Side-effect if ignoring latency in the race for greater throughput.
    - http://www.bufferbloat.net/projects/bloat/wiki/Introduction
  - Find out about your buffers.
    - Use 'ss -e' or 'netstat -anp'
    - NIC ring buffers and new Byte Queue Limits
      - http://linuxplumbersconf.org/2011/ocw/sessions/171
      - https://lwn.net/Articles/454390/

BDP/Buffer Bloat

- Why are they filling...is there a bottleneck ?
  - Do you care ? Maybe not (yet).



Multiqueue Networking (aka RSS)



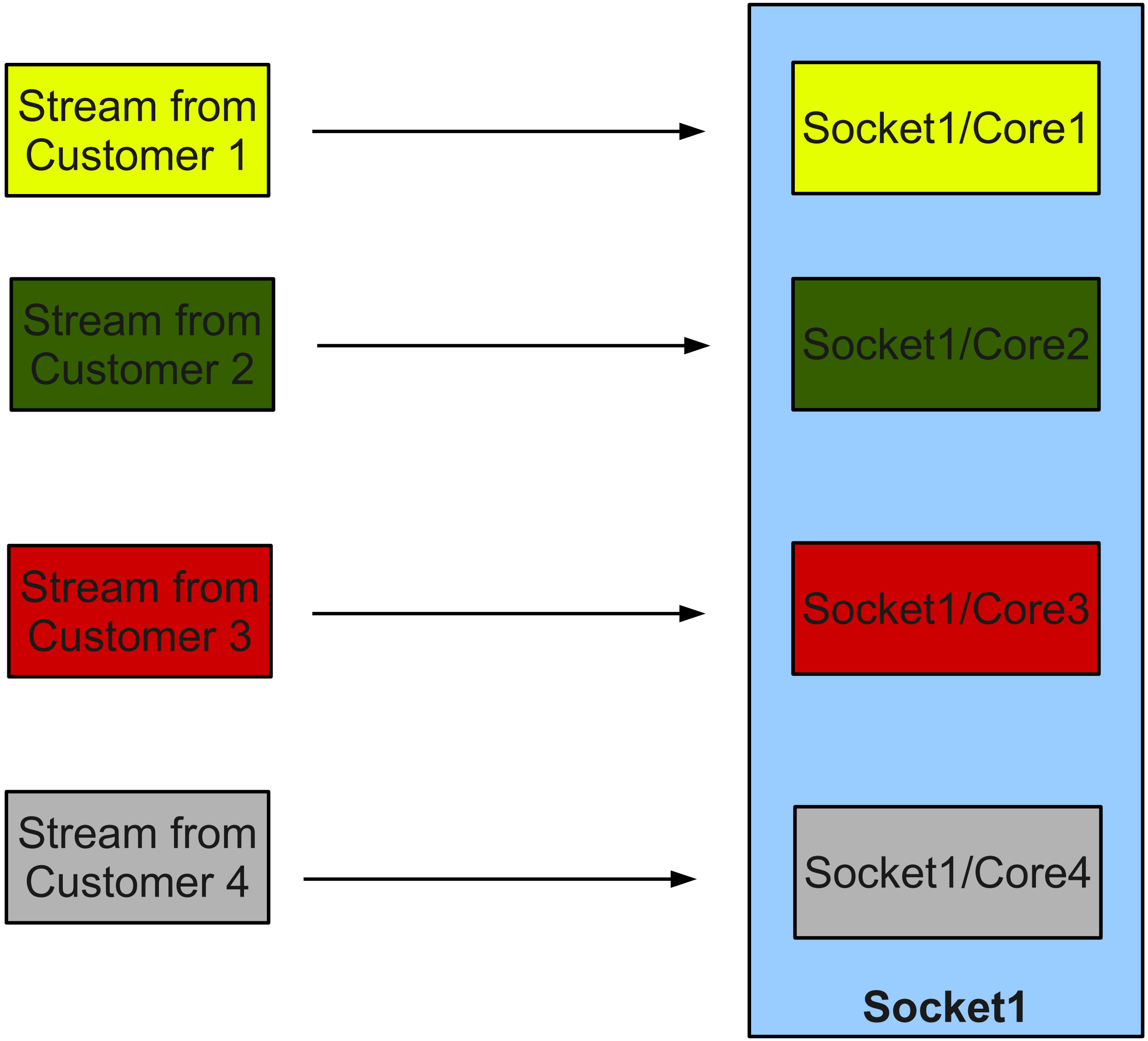
- Invented to allow Linux networking to scale along with hardware
- 2 socket/8 cores extremely common, optimize for this use-case
- Hash of src/dst IP:PORT determines receiving CPU

IRQ Affinity – Why Bother?

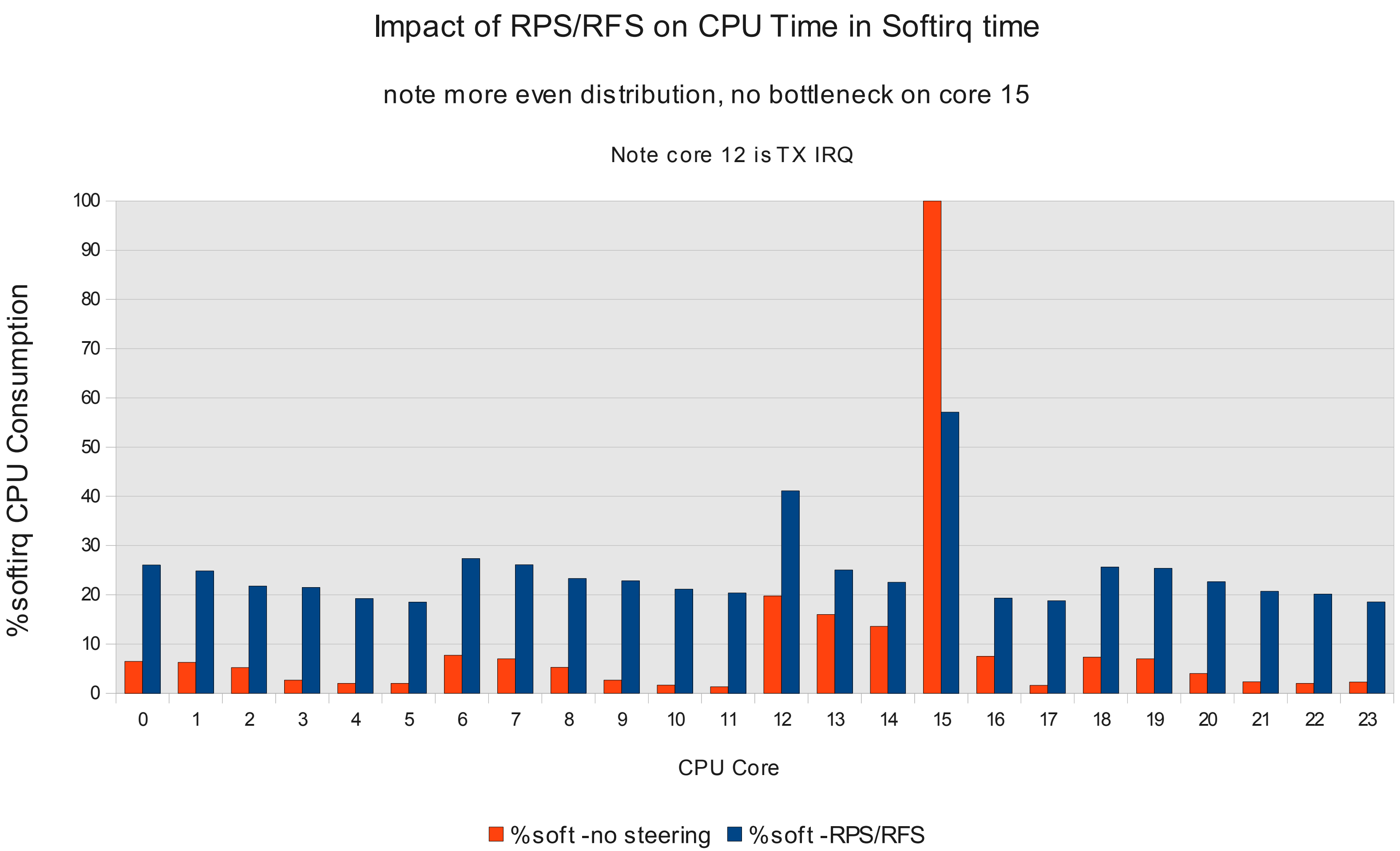
- Premature optimization is...
- Perfect is the enemy of good enough...
- When not to use irqbalance? It's good...improving rapidly.
  - Constantly re-evaluating IRQ placement based on load.
  - Knows the difference between IRQs related to storage, 1G NICs, 10G NICs etc...applies costs to NUMA nodes using sophisticated algorithms.
- So many moving pieces; different drivers allocate IRQs differently. Some drivers allow you to specify the number of TX/RX queues. Some statically define queues per core. Some do their own flow-steering.
  - Diverse hardware is quite a bit for a human to manage. MSI means there are tons of interrupt lines...A daemon's job.
- Manual affinity tuning always an option...great documentation here:

  https://access.redhat.com/knowledge/techbriefs/optimizing-red-hat-enterprise-linux-performance-tuning-irq-affinity

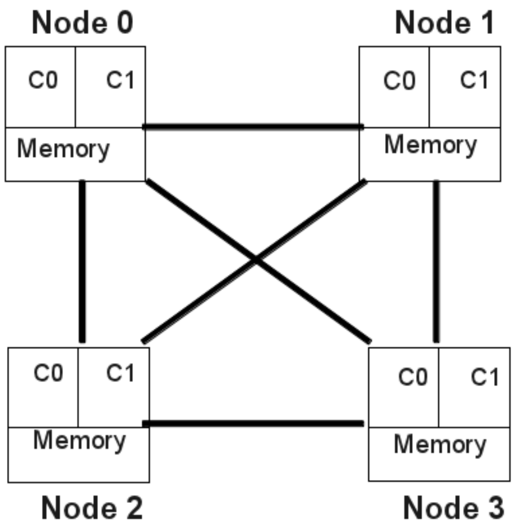
Locality of Packets (IRQ processing)



Network packet / flow steering
- Spreads RX Packet handling across cores (multiple senders, ab)



NUMA Topology and PCI Bus



- Server may have more than 1 PCI bus.
  - Optimal performance reduces/eliminates inter-node cross-talk. Install NIC in slot local to node that your application will run on.
  - Use systemtap numa_faults.stp.
  - irqbalance will learn this “soon”.
  - Kernel already knows.
- Certain network adapters implement Accelerated RFS, or programmable flow-steering.
- It's also possible to configure the max byte read-count from the PCI bus (of the network card or whatever else...)

```
lspci|grep 1234

grep 1234 /proc/bus/pci/devices

setpci -v -d 1234:5678 e6.b=2e (512byte default,
max 4KB)
```

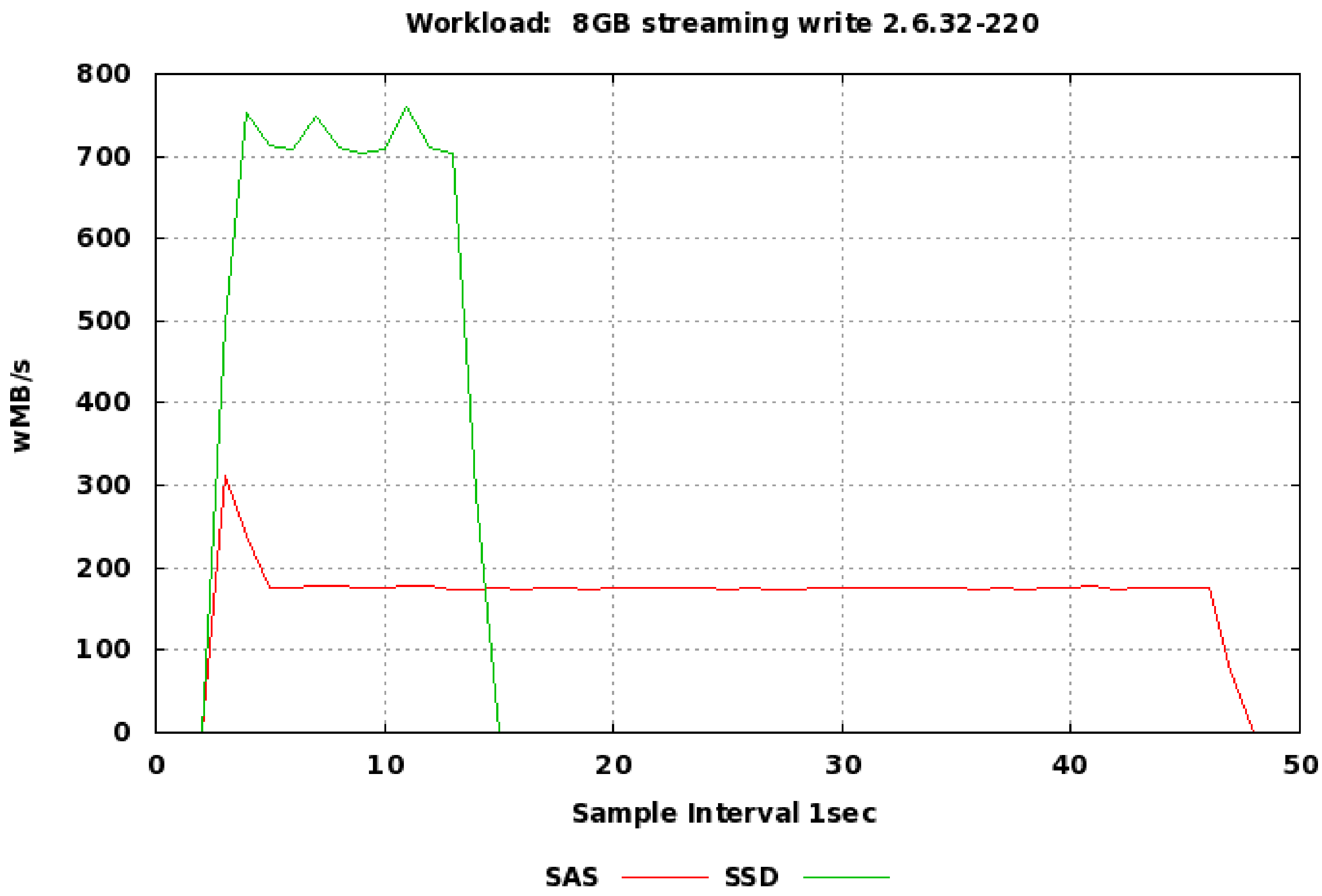
But the OS should handle all of this...

- Performance Group couldn't agree more!
  - Out of the box performance experience highest priority.
    - Defaults work for the majority of use-cases
    - Auto-tuning where it doesn't
    - Hand tuning as a last resort
  - irqbalance being taught about PCI bus locality (local_cpulist)
    - Kernel already knows, RHEL6.3+ will set it for you.
  - numad can automatically balance NUMA node utilization to avoid NUMA faults.

(Real) Agenda...

- Performance Tuning Theory
- RHEL6 Performance Improvements
- tuned
- cgroups Architecture and Deep-dive
- CPU Performance Tuning/Power Management
- Memory/NUMA Performance Tuning
- Network Performance Tuning
- Bonus

Other interesting tid-bits...PCI-e Storage



Other interesting tid-bits...Intel DDIO

- Intel® Data Direct I/O Technology
- Intel DDIO makes the processor cache the primary destination and source of I/O data rather than main memory
- http://www.intel.com/content/www/us/en/io/direct-data-i-o.html

THANK YOU!

Questions