Linux Performance Tuning

Larry Woodman
Joe Mario

October 19, 2017

Agenda

- Low Latency
- CPU cacheline contention
- A few compiler & tools tips
- Fundamental kernel internals:
  - Function wrt to performance
  - Tuning
  - Interactions between file systems, memory, & devices
  - Throughput vs latency tuning

Brief background
Tuned basics

What is “tuned” ?

- Tuning profile delivery mechanism
- Many linux distros ship tuned profiles that improve performance for many workloads…
- Customize your own profile.

Tuned (cont)

```
# tuned-adm list
 Available profiles:
   - balanced
   - cpu-partitioning                    << New in 7.4
   - desktop
   - latency-performance
   - network-latency
   - network-throughput
   - powersave
   - throughput-performance
   - virtual-guest
   - virtual-host
  Current active profile: balanced
```

Setting tuned

What is my system currently tuned to?

```
# tuned-adm active
    Current active profile: balanced
```

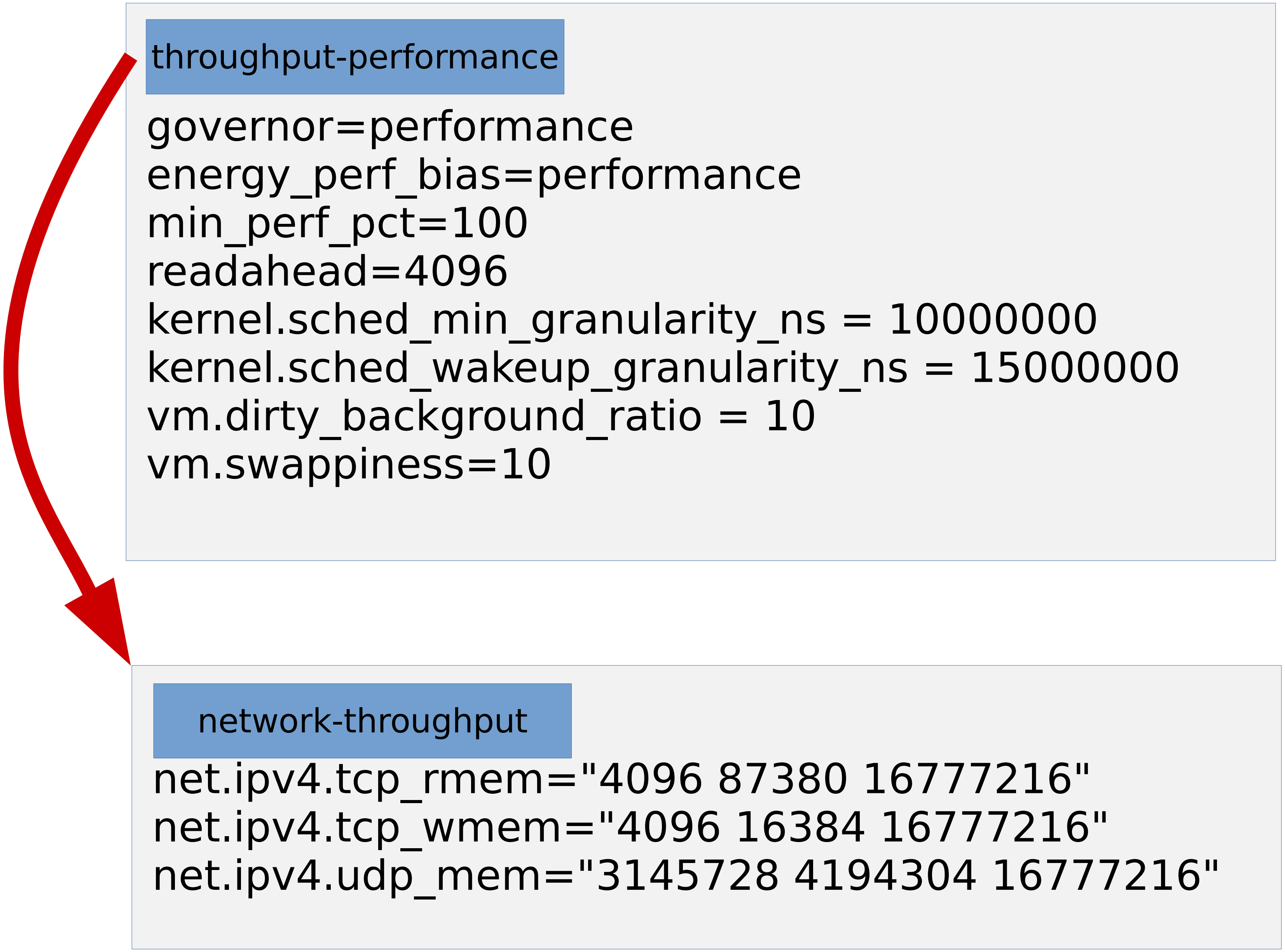
How do I change my current tuning setting?

```
# tuned-adm profile network-latency
```

throughput-performance (RHEL7 default)

- governor=**performance**
- energy_perf_bias=**performance**
- min_perf_pct=**100**
- readahead=**4096**
- kernel.sched_min_granularity_ns = **10000000**
- kernel.sched_wakeup_granularity_ns = **15000000**
- vm.dirty_background_ratio = **10**
- vm.swappiness=**10**

Tuned: Profile Inheritance (throughput)



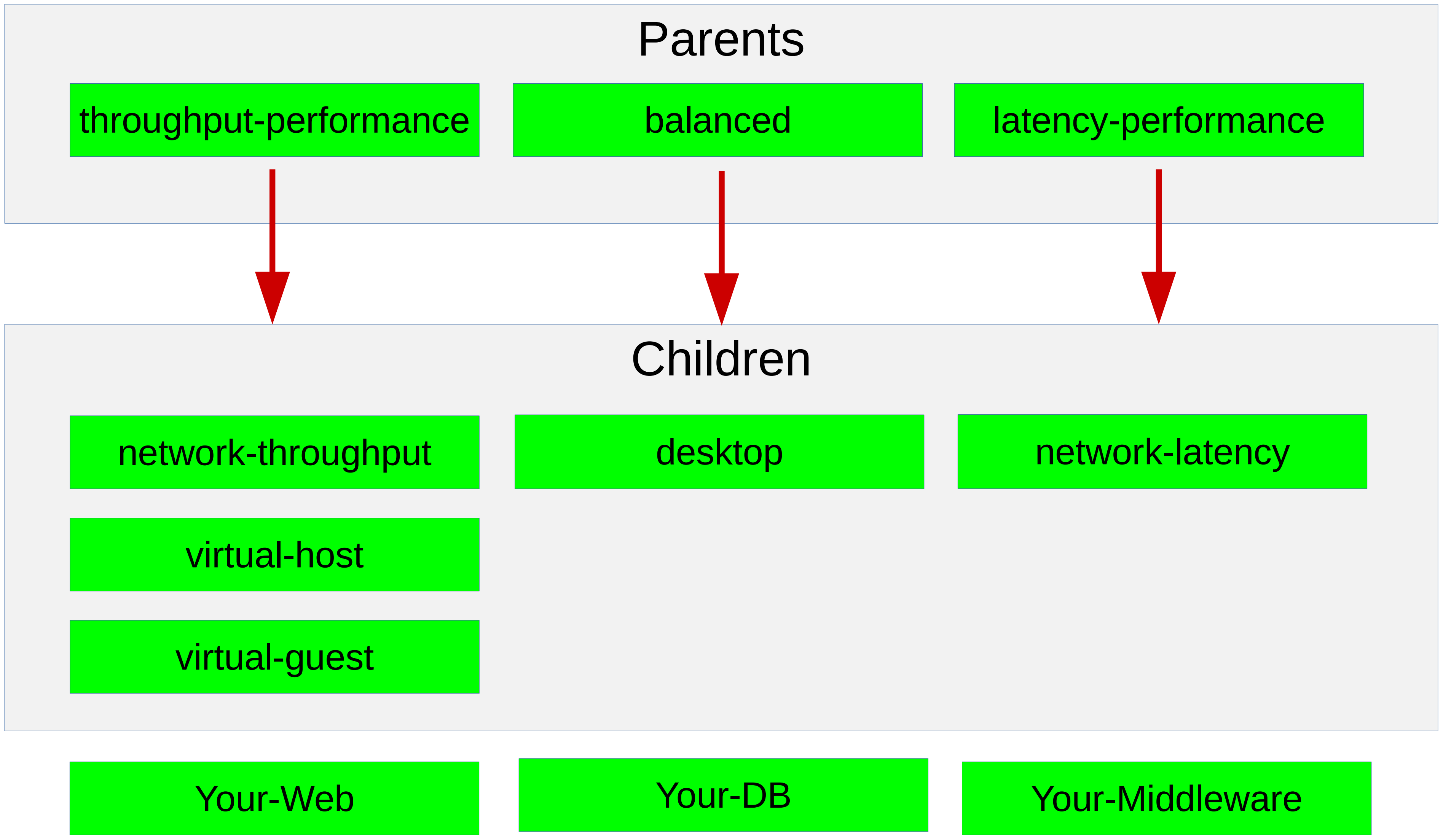
throughput-performance

```
governor=performance
energy_perf_bias=performance
min_perf_pct=100
readahead=4096
kernel.sched_min_granularity_ns = 10000000
kernel.sched_wakeup_granularity_ns = 15000000
vm.dirty_background_ratio = 10
vm.swappiness=10
```

network-throughput

```
net.ipv4.tcp_rmem="4096 87380 16777216"
net.ipv4.tcp_wmem="4096 16384 16777216"
net.ipv4.udp_mem="3145728 4194304 16777216"
```

Tuned: Profile Inheritance

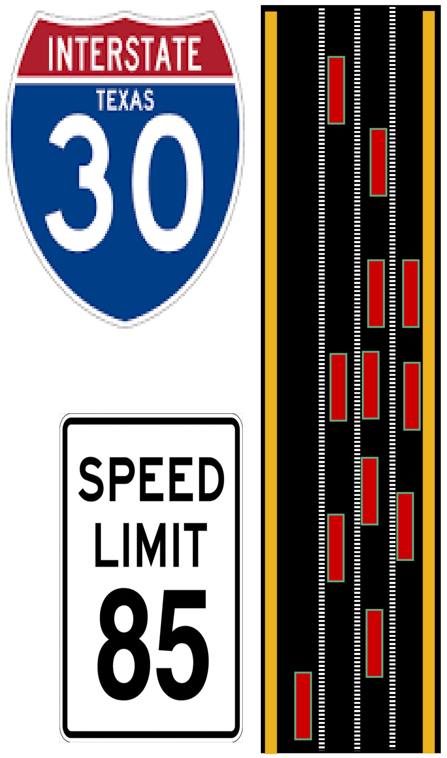


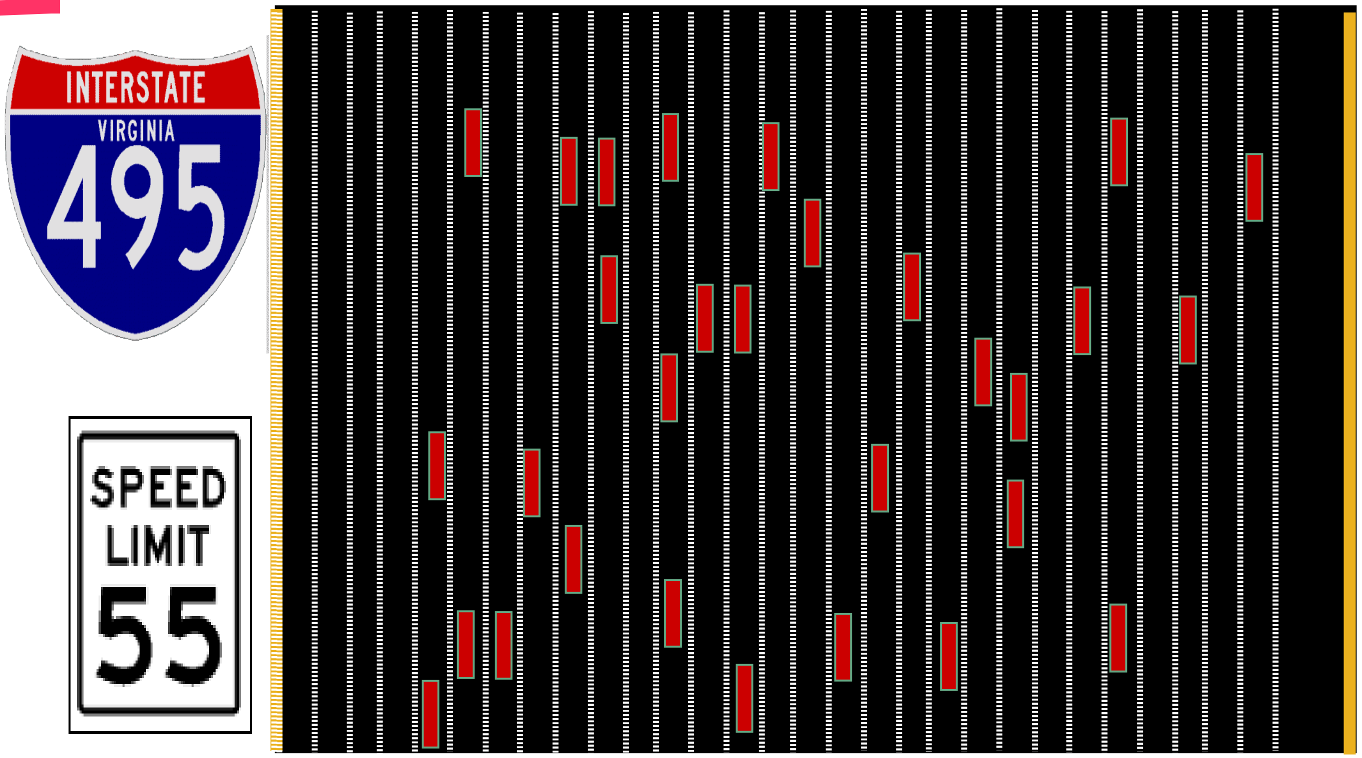
Low Latency Considerations

Performance Metrics

Latency==Speed

Throughput==Bandwidth

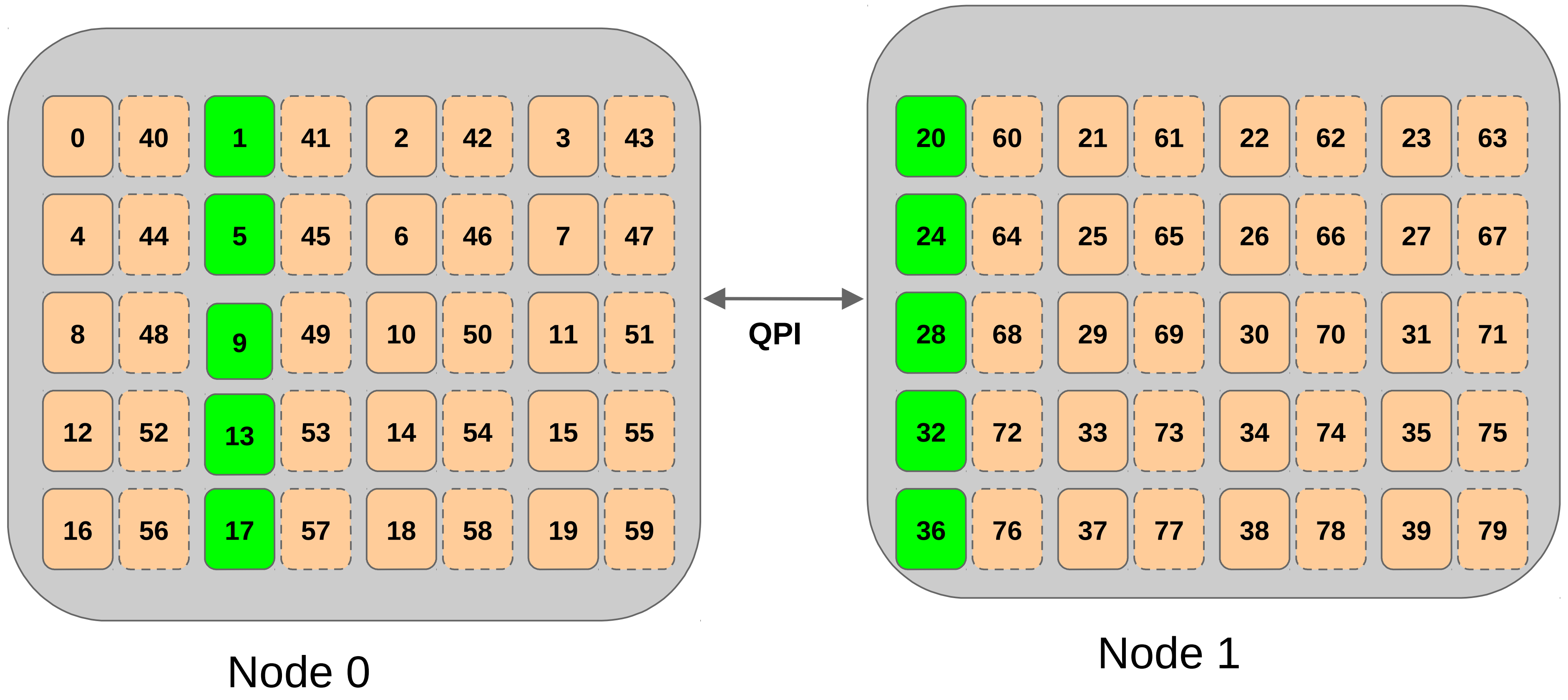




Latency – Speed Limit
- Ghz of CPU, Memory PCI
- Small transfers, disable aggregation – TCP nodelay
- Dataplane optimization DPDK

Throughput: Bandwidth: # lanes in Highway
- Width of data path / cachelines
- Bus Bandwidth, QPI links, PCI 1-2-3
- Network 1 / 10 / 40 Gb – aggregation, NAPI
- Fiberchannel 4/8/16, SSD, NVME Drivers

Isolcpus – widely used



Boot with “isolcpus=1,5,9,13,17,20,24,28,32,36”

Run your application(s) that pins individual threads to the isolated cores.

Life is good.

Isolcpus – no scheduler load balancing

Boot your system with “isolcpus=1,2,3,4”

Then run:

```
taskset -c 1,2,3,4  yes  > /dev/null &
taskset -c 1,2,3,4  yes  > /dev/null &
taskset -c 1,2,3,4  yes  > /dev/null &
taskset -c 1,2,3,4  yes  > /dev/null &
```

Result: All four “yes” processes will run **only** on cpu 1.
CPUs 2,3 and 4 will be idle.

Isolcpus – no scheduler load balancing

Then try:

```
taskset -c 1  yes > /dev/null &
taskset -c 2  yes > /dev/null &
taskset -c 3  yes > /dev/null &
taskset -c 4  yes > /dev/null &
```

Or:

```
taskset -c 1,2,3,4  chrt --rr 1 yes > /dev/null &
taskset -c 1,2,3,4  chrt --rr 1 yes > /dev/null &
taskset -c 1,2,3,4  chrt --rr 1 yes > /dev/null &
taskset -c 1,2,3,4  chrt --rr 1 yes > /dev/null &
```

Result: All four “yes” processes will be spread across cpus 1,2,3,4

With kernel isolcpus,

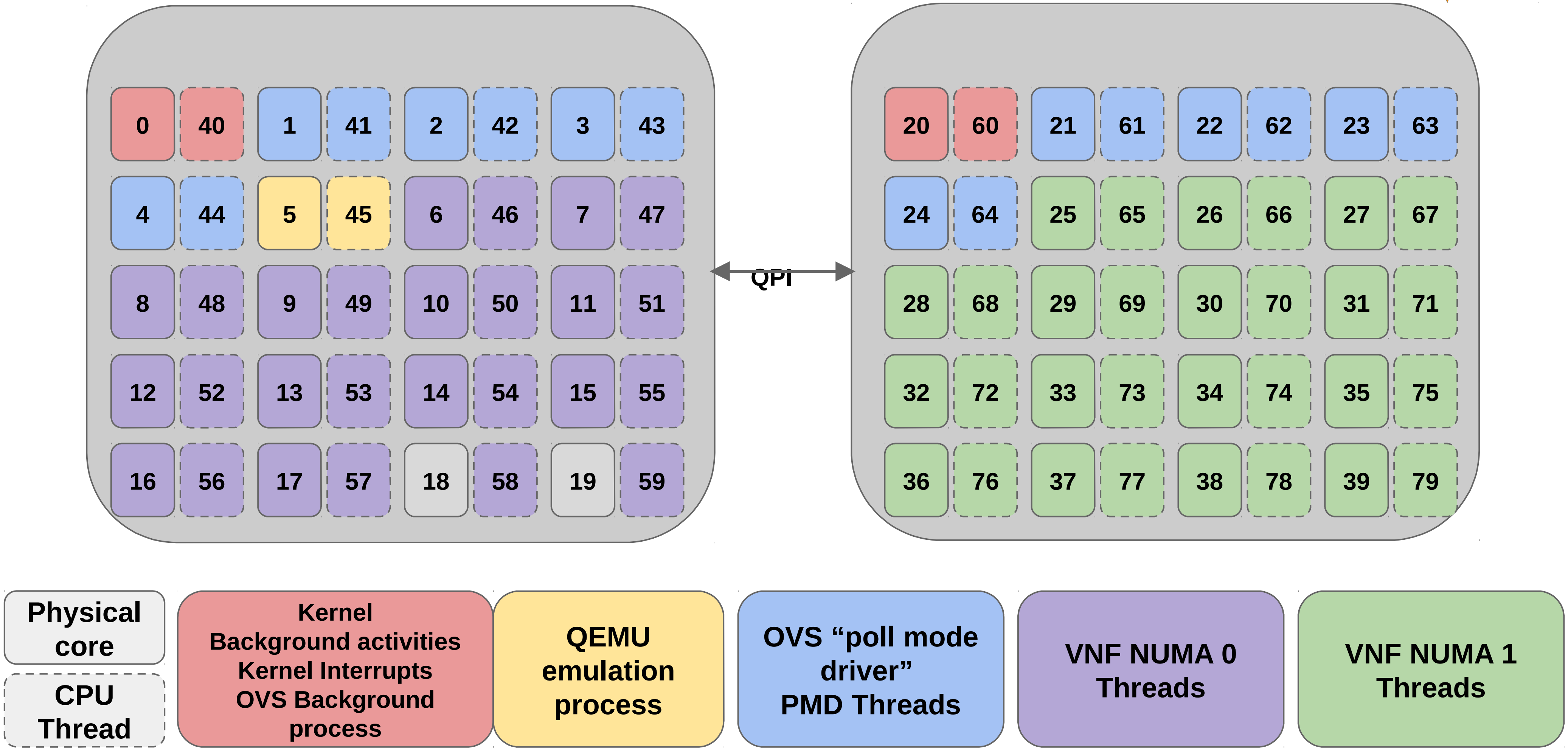
- must manually pin processes or individual threads,
- or use realtime scheduling (chrt)

Isolcpus doesn’t work for larger applications

VNF Mobile Network - Graphical CPU Partitioning

New in7.4

: isolcpus=1-19,21-39,41-59,61-79



New “cpu-partitioning” tuned profile

- For latency sensitive applications needing kernel scheduler load balancing:
  - Decide which cpus you want to allocated to it.
  - Add those cpus to a tuned configuration file.

    /etc/tuned/cpu-partitioning-variables.conf
  - Set the cpu-partitioning tuned profile.

    # tuned-adm profile cpu-partitioning
  - Then reboot!

Cpu-partitioning – after reboot you have:

Sets cpu affinity mask away from isolated cpus for:

- All processes spawned by systemd, IRQs, RCU callbacks
- Kernel thread issuing dirty page writebacks
- Kworker workqueues for interrupts, timers, I/O, etc.

Sets nohz_full on isolated cpus

Disables intel idle driver to prevent frequency scaling

Sets nosoftlockup and disables KSM (kernel same page merging)

Sets mce=ignore_ce (preventing periodic polling of machine check banks)

Sets pause loop exit and ple_gap KVM options set to minimize VM exits

Uses tuna to move all user processes away from isolated cpus.

For example: `# tuna -c 4,5,6,7 -i`

Cpu-partitioning – after reboot (continued):

- kernel.hung_task_timeout_secs = 600
- kernel.nmi_watchdog = 0
- vm.stat_interval = 10
- kernel.timer_migration = 1
- net.core.busy_read = 50 and net.core.busy_poll = 50
- kernel.numa_balancing = 0
- kernel.sched_min_granularity_ns = 10000000
- vm.dirty_ratio = 10
- vm.dirty_background_ratio = 3
- vm.swappiness = 10
- kernel.sched_migration_cost_ns = 5000000
- Disable Transparent Hugepages

Cpu-partitioning – after reboot (continued):

Also sets these tuned parameters

- force_latency = 1
- governor = performance
- energy_perf_bias = performance
- min_perf_pct = 100

Does not set the “isolcpus=” kernel cpu flag

- isolcpus= disables the load balancer
- We’ve measured load balancer hit to be about 40 usec
- Disabling load balancer will soon be an option.
  “*no_rebalance_cores=*” coming soon.

Also does not set the “skew_tick=1” flag (yet)

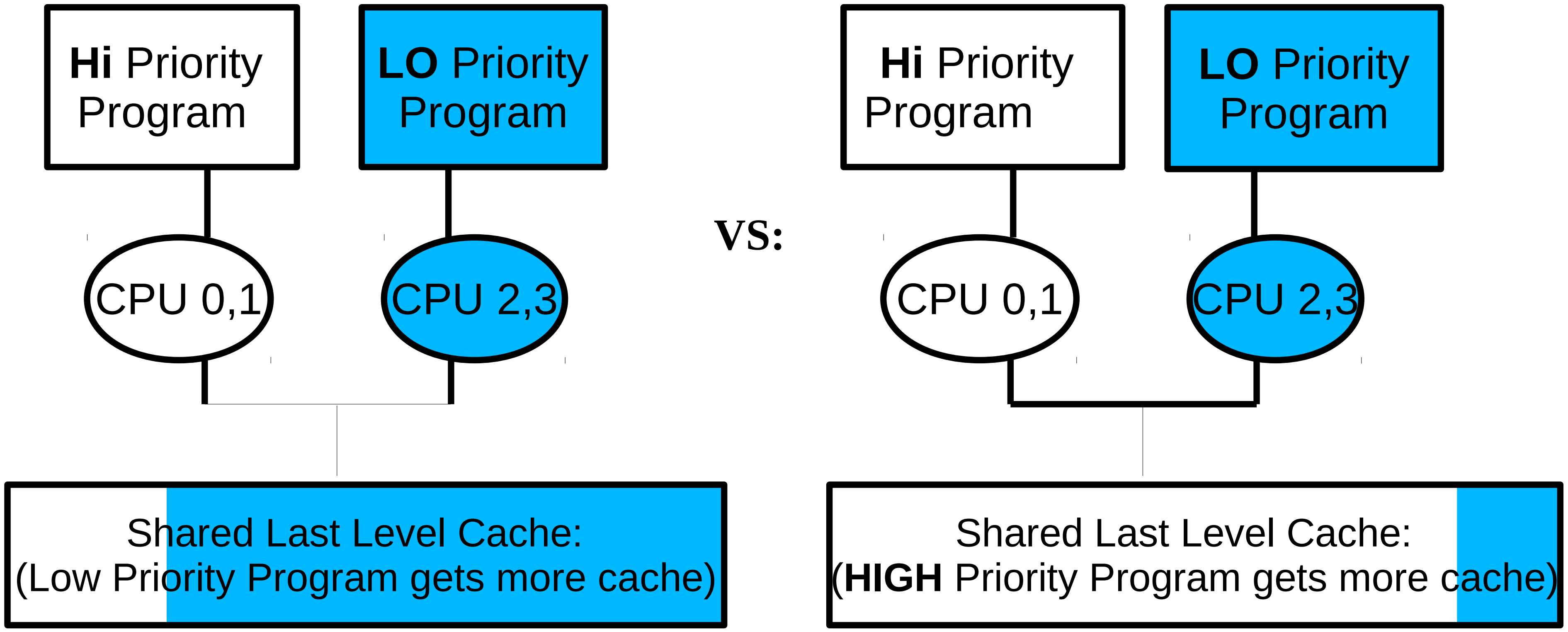
Cstate considerations

- C0
  - Most responsive – cpus fully turned on
  - Prevents turbo mode.
- C1
  - Most common.
  - Red Hat uses C1 in its tuned profiles.
- C3
  - We're seeing **a few** applications run faster with C3.
  - Lowering all cpus to C3 allows for more turbo headroom for the cpus that need it.
  - Depends on your application

Cache Allocation & Cache Monitoring Technology

Noisy Cacheline Neighbor

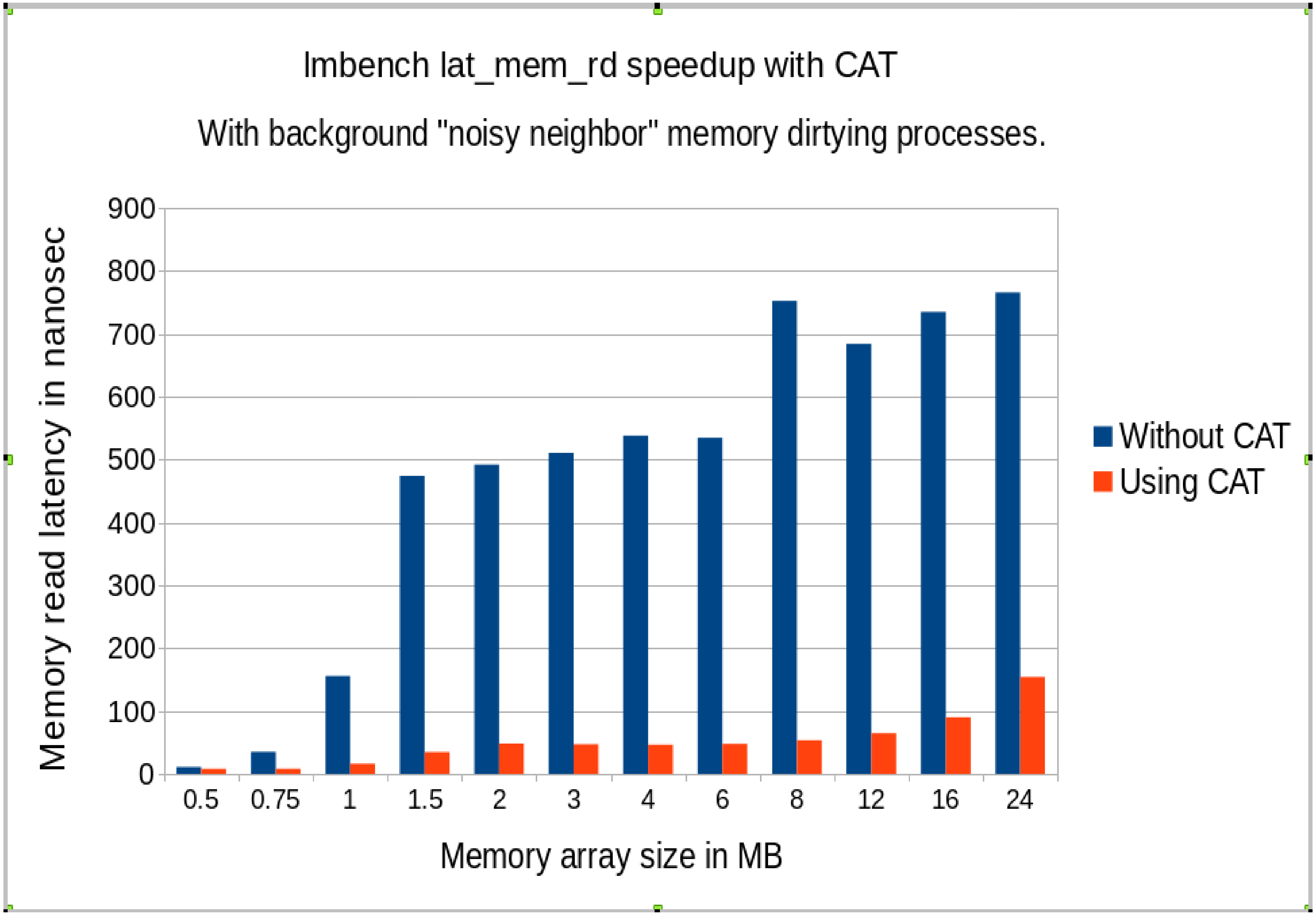
New in7.4



Hi Priority Program
LO Priority Program
CPU 0,1
CPU 2,3
Shared Last Level Cache: (Low Priority Program gets more cache)
VS:
Hi Priority Program
LO Priority Program
CPU 0,1
CPU 2,3
Shared Last Level Cache: (**HIGH** Priority Program gets more cache)

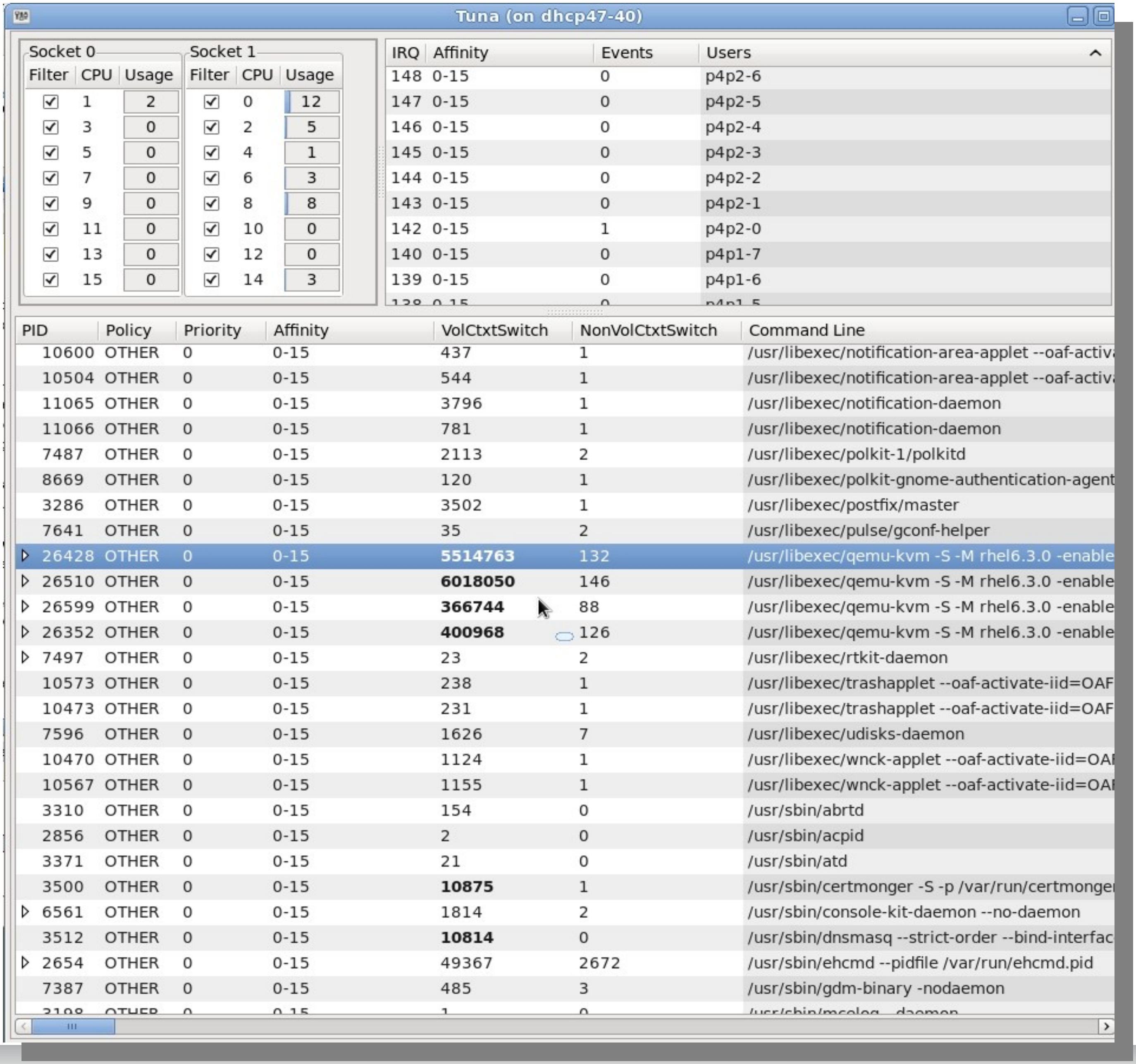
Available in RHEL 7.4 via the intel-cmt-cat-*.el7 package.
See ‘man pqos’
Intel only. (Some Haswells, all Broadwells, Skylake errata)

Memory latency testing using CAT



Process Tuning Tool - tuna

- Fine grained control
- Display applications & processes
- CPU enumeration
- Socket (useful for NUMA tuning)
- Dynamic control of:
  - Process affinity
  - Parent & threads
  - Scheduling policy
  - Device IRQ priorities, etc



Tuna (on dhcp47-40)

Socket 0

Filter	CPU	Usage
✓	1	2
✓	3	0
✓	5	0
✓	7	0
✓	9	0
✓	11	0
✓	13	0
✓	15	0

Socket 1

Filter	CPU	Usage
✓	0	12
✓	2	5
✓	4	1
✓	6	3
✓	8	8
✓	10	0
✓	12	0
✓	14	3

IRQ	Affinity	Events	Users
148	0-15	0	p4p2-6
147	0-15	0	p4p2-5
146	0-15	0	p4p2-4
145	0-15	0	p4p2-3
144	0-15	0	p4p2-2
143	0-15	0	p4p2-1
142	0-15	1	p4p2-0
140	0-15	0	p4p1-7
139	0-15	0	p4p1-6

PID	Policy	Priority	Affinity	VolCtxtSwitch	NonVolCtxtSwitch	Command Line
10600	OTHER	0	0-15	437	1	/usr/libexec/notification-area-applet --oaf-activ
10504	OTHER	0	0-15	544	1	/usr/libexec/notification-area-applet --oaf-activ
11065	OTHER	0	0-15	3796	1	/usr/libexec/notification-daemon
11066	OTHER	0	0-15	781	1	/usr/libexec/notification-daemon
7487	OTHER	0	0-15	2113	2	/usr/libexec/polkit-1/polkitd
8669	OTHER	0	0-15	120	1	/usr/libexec/polkit-gnome-authentication-agent
3286	OTHER	0	0-15	3502	1	/usr/libexec/postfix/master
7641	OTHER	0	0-15	35	2	/usr/libexec/pulse/gconf-helper
▷ 26428	OTHER	0	0-15	**5514763**	132	/usr/libexec/qemu-kvm -S -M rhel6.3.0 -enable
▷ 26510	OTHER	0	0-15	**6018050**	146	/usr/libexec/qemu-kvm -S -M rhel6.3.0 -enable
▷ 26599	OTHER	0	0-15	**366744**	88	/usr/libexec/qemu-kvm -S -M rhel6.3.0 -enable
▷ 26352	OTHER	0	0-15	**400968**	126	/usr/libexec/qemu-kvm -S -M rhel6.3.0 -enable
▷ 7497	OTHER	0	0-15	23	2	/usr/libexec/rtkit-daemon
10573	OTHER	0	0-15	238	1	/usr/libexec/trashapplet --oaf-activate-iid=OAF
10473	OTHER	0	0-15	231	1	/usr/libexec/trashapplet --oaf-activate-iid=OAF
7596	OTHER	0	0-15	1626	7	/usr/libexec/udisks-daemon
10470	OTHER	0	0-15	1124	1	/usr/libexec/wnck-applet --oaf-activate-iid=OA
10567	OTHER	0	0-15	1155	1	/usr/libexec/wnck-applet --oaf-activate-iid=OA
3310	OTHER	0	0-15	154	0	/usr/sbin/abrtd
2856	OTHER	0	0-15	2	0	/usr/sbin/acpid
3371	OTHER	0	0-15	21	0	/usr/sbin/atd
3500	OTHER	0	0-15	**10875**	1	/usr/sbin/certmonger -S -p /var/run/certmonge
▷ 6561	OTHER	0	0-15	1814	2	/usr/sbin/console-kit-daemon --no-daemon
3512	OTHER	0	0-15	**10814**	0	/usr/sbin/dnsmasq --strict-order --bind-interfac
▷ 2654	OTHER	0	0-15	49367	2672	/usr/sbin/ehcmd --pidfile /var/run/ehcmd.pid
7387	OTHER	0	0-15	485	3	/usr/sbin/gdm-binary -nodaemon

tuna command line

```
# tuna --help
   -h, --help                          Give this help list
   -a, --config_file_apply=profilename Apply changes described in profile
   -l, --config_file_list              List preloaded profiles
   -g, --gui                           Start the GUI
   -G, --cgroup                        Display the processes with the type of cgroups they
                                       are in
   -c, --cpus=CPU-LIST                 CPU-LIST affected by commands
   -C, --affect_children               Operation will affect children threads
   -f, --filter                        Display filter the selected entities
   -i, --isolate                       Move all threads away from CPU-LIST
   -I, --include                       Allow all threads to run on CPU-LIST
   -K, --no_kthreads                   Operations will not affect kernel threads
   -m, --move                          Move selected entities to CPU-LIST
   -N, --nohz_full                     CPUs in nohz_full= kernel command line will be
                                       affected by operations
   -p, --priority=[POLICY:]RTPRIO      Set thread scheduler tunables: POLICY and RTPRIO
   -P, --show_threads                  Show thread list
   -Q, --show_irqs                     Show IRQ list
   -q, --irqs=IRQ-LIST                 IRQ-LIST affected by commands
   -r, --run=COMMAND                   fork a new process and run the COMMAND
   -s, --save=FILENAME                 Save kthreads sched tunables to FILENAME
   -S, --sockets=CPU-SOCKET-LIST       CPU-SOCKET-LIST affected by commands
   -t, --threads=THREAD-LIST           THREAD-LIST affected by commands
   -U, --no_uthreads                   Operations will not affect user threads
   -v, --version                       Show version
   -W, --what_is                       Provides help about selected entities
   -x, --spread                        Spread selected entities over CPU-LIST
```

Tuna – command line examples

Move an irq to cpu 5

- tuna -c5 -q eth4-rx-4 –move

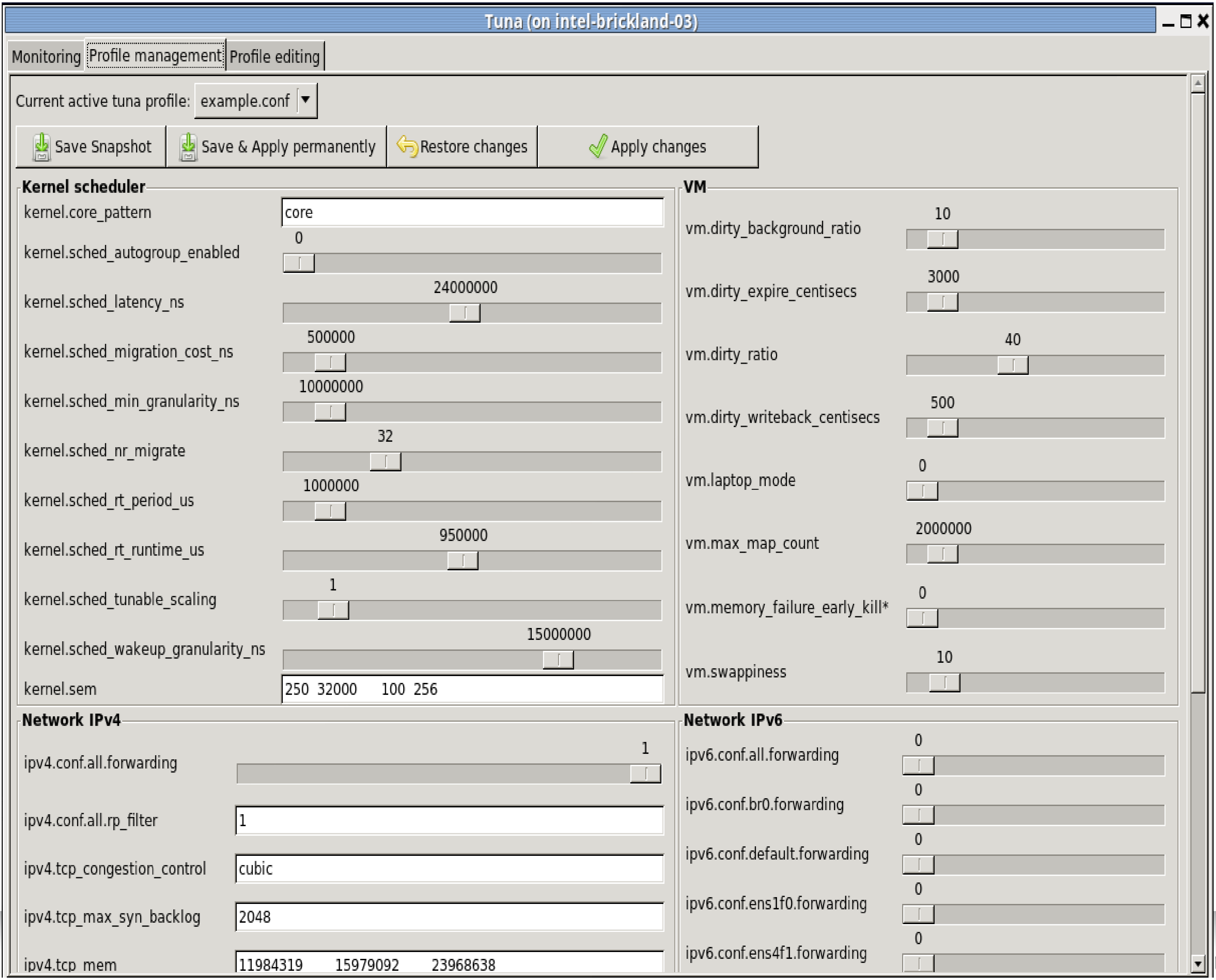
Move all irqs named “eth4*****” away from numa node 1

- tuna -S 1 -i -q ‘eth4*’

Move all rcu kernel threads to cpus 1 and 3

- tuna -c1,3 -t ‘*rcu*’ --move

Tuna GUI Capabilities Updated for RHEL7



Current active tuna profile: example.conf

Kernel scheduler

kernel.core_pattern	core
kernel.sched_autogroup_enabled	0
kernel.sched_latency_ns	24000000
kernel.sched_migration_cost_ns	500000
kernel.sched_min_granularity_ns	10000000
kernel.sched_nr_migrate	32
kernel.sched_rt_period_us	1000000
kernel.sched_rt_runtime_us	950000
kernel.sched_tunable_scaling	1
kernel.sched_wakeup_granularity_ns	15000000
kernel.sem	250 32000 100 256

VM

vm.dirty_background_ratio	10
vm.dirty_expire_centisecs	3000
vm.dirty_ratio	40
vm.dirty_writeback_centisecs	500
vm.laptop_mode	0
vm.max_map_count	2000000
vm.memory_failure_early_kill*	0
vm.swappiness	10

Network IPv4

ipv4.conf.all.forwarding	1
ipv4.conf.all.rp_filter	1
ipv4.tcp_congestion_control	cubic
ipv4.tcp_max_syn_backlog	2048
ipv4.tcp_mem	11984319 15979092 23968638

Network IPv6

ipv6.conf.all.forwarding	0
ipv6.conf.br0.forwarding	0
ipv6.conf.default.forwarding	0
ipv6.conf.ens1f0.forwarding	0
ipv6.conf.ens4f1.forwarding	0

Questions

Identifying cpu cacheline contention

Look at a simple example false sharing example:

Two scenarios of a basic data structure

```
struct false_sharing_buf {                    // Reader & writer
    long writer;                              // fields together
    long reader;
} buf ;


struct uncontended_buf {                      // Writer fields
    long writer;                              // separated from
    long pad[7];                              // writer field
    long reader1;
    long pad2[7];
} buf;
```

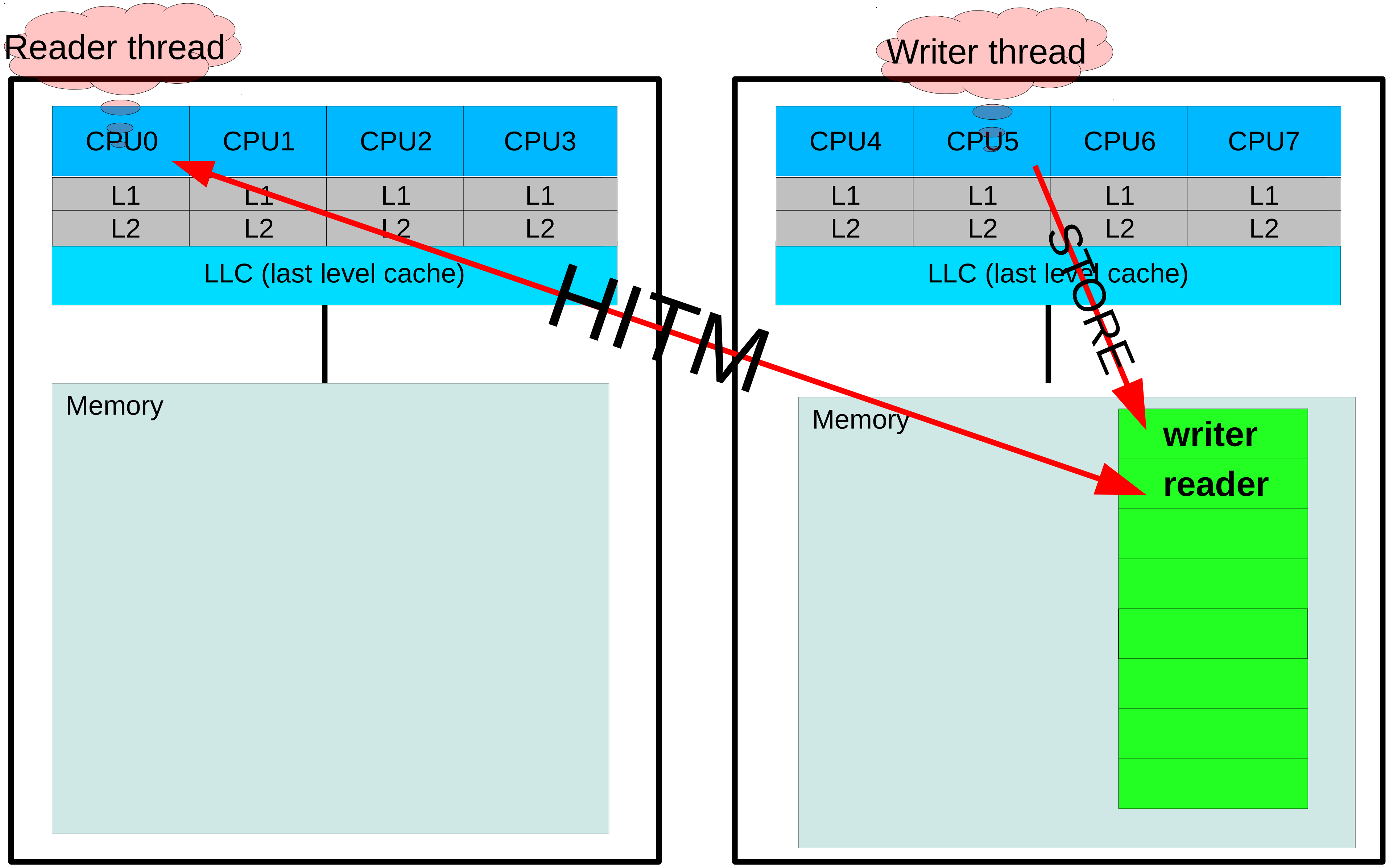
Run it through a simple loop:

- Two threads running in parallel.
- Assume buf struct aligned on 64-byte boundary.
- loop-cnt = 500,000,000

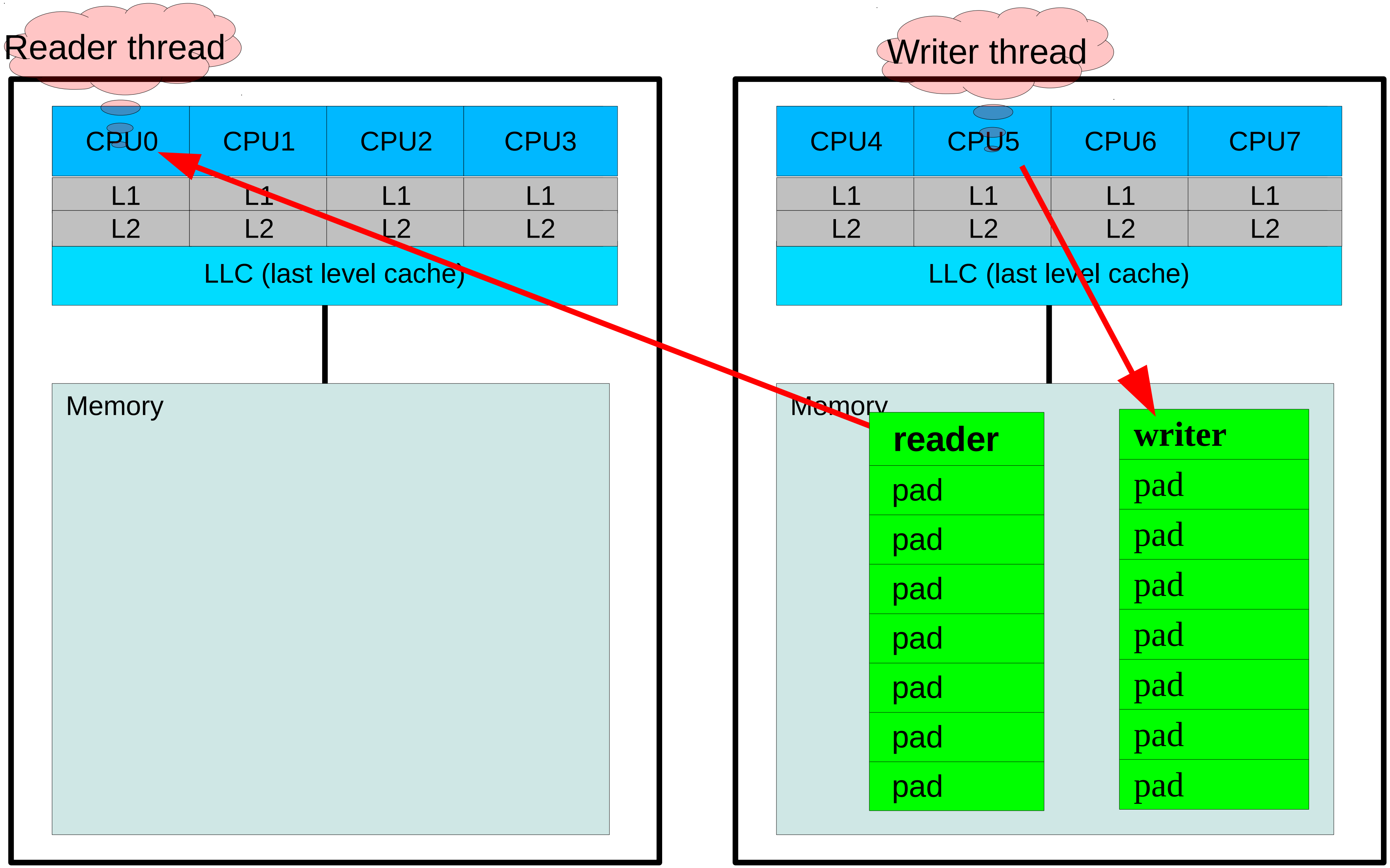
```
/* Writer thread on node 0 */
    for (i = 0; i < loop-cnt; ++i) {
            buf.writer += 1;
            asm volatile("rep; nop")
    }

/* Reader thread on node 1 */
    for (i = 0; i < loop-cnt; ++i) {
            var = buf.reader;
            asm volatile("rep; nop")
    }
```

Question:
How fast can the reader thread complete the loop?

Scenario 1: Both fields in one data struct



Scenario 2: Each field in own cacheline:



Run it through a simple loop:

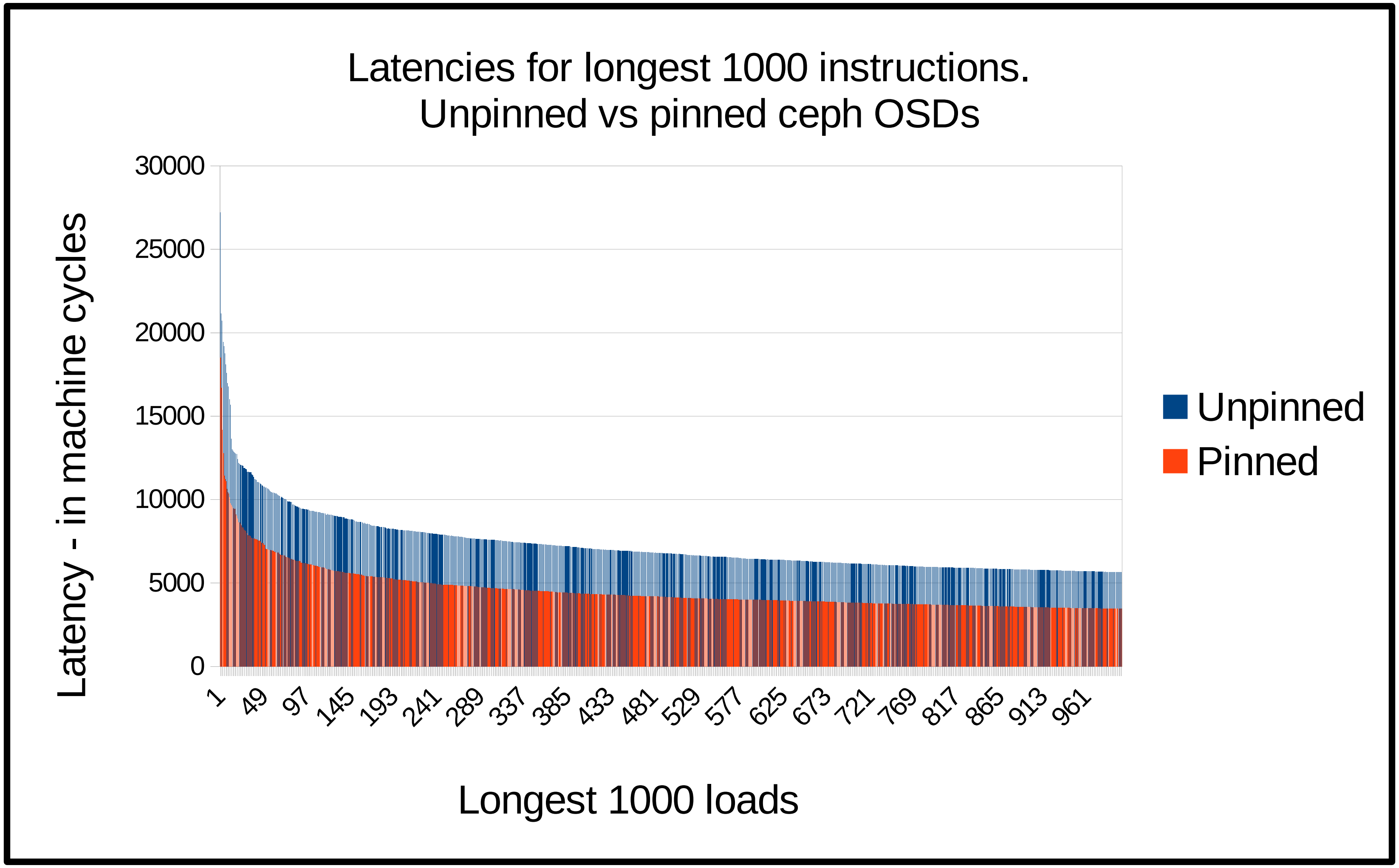
- Two threads running in parallel.
- Assume buf struct aligned on 64-byte boundary.
- loop-cnt = 500,000,000

```
/* Writer thread on node 0 */
    for (i = 0; i < loop-cnt; ++i) {
            buf.writer += 1;
            asm volatile("rep; nop")
    }

/* Reader thread on node 1 */
    for (i = 0; i < loop-cnt; ++i) {
            var = buf.reader;
            asm volatile("rep; nop")
    }
```

Question:
How fast can the reader thread complete the loop?

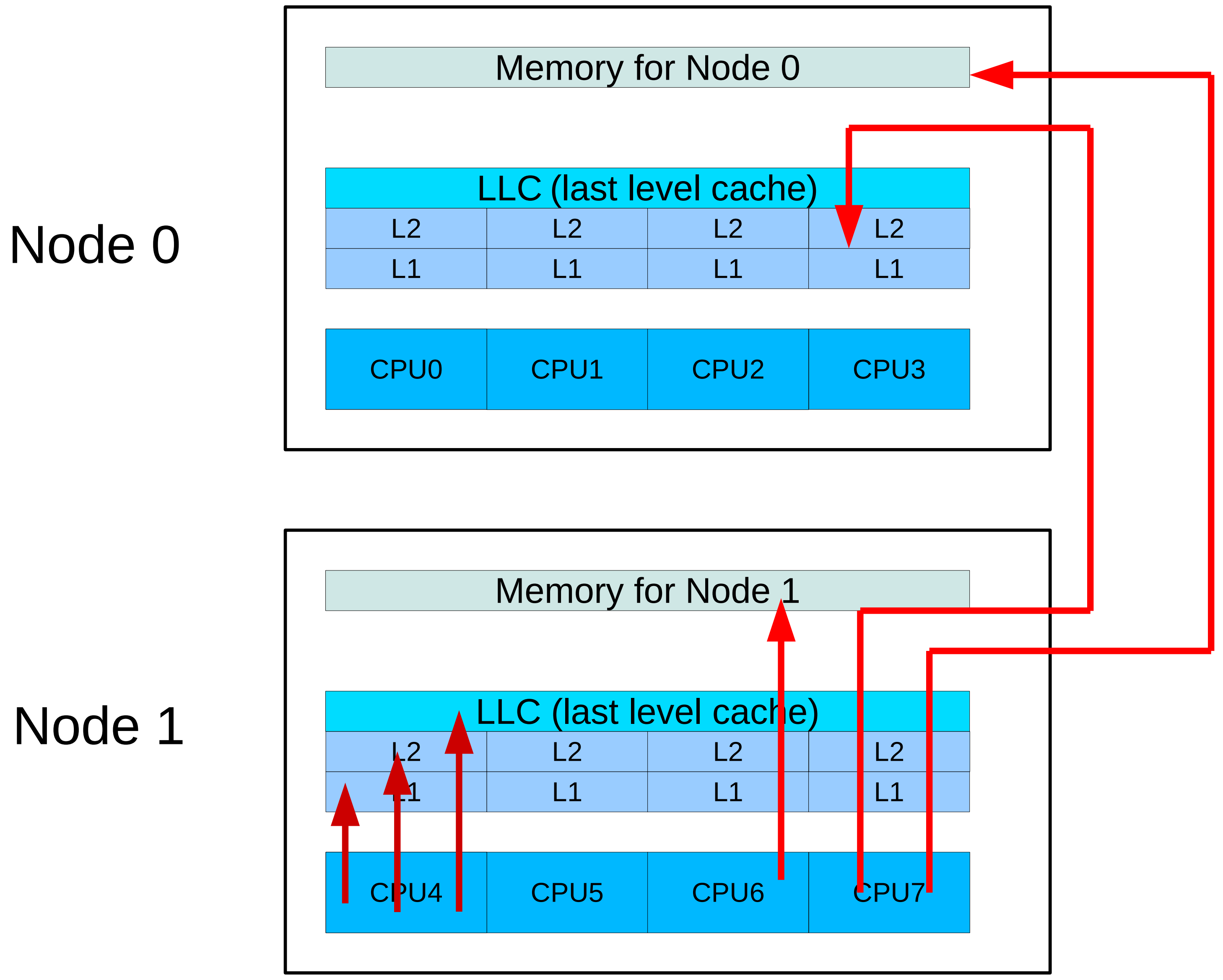
Answer:
When "buf.writer" shares a cacheline, the reader thread finishes loop:
2-4X slower on 2 node system,
20X slower on 4 node system.

Ceph OSDs simultaneously accessing struct and locks, across two numa nodes vs pinned to one numa node.

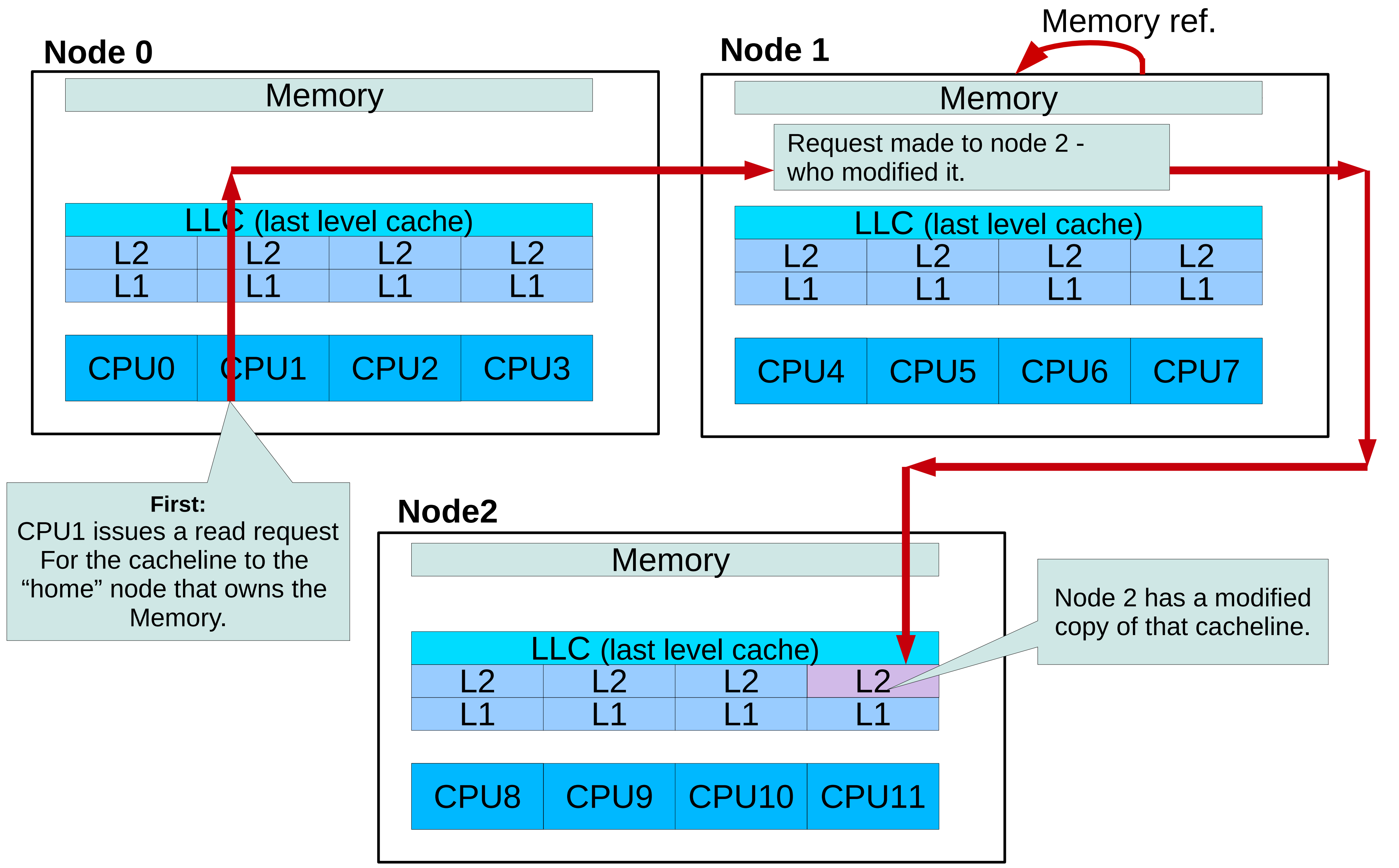


- Where does my program get its memory from?
- When does it hurt your performance the most?
  - CPU cacheline false sharing
- How to find out where it’s happening?
- How to resolve it?

Background Basics
Resolving a memory access



Resolving a memory access – more expensive case.



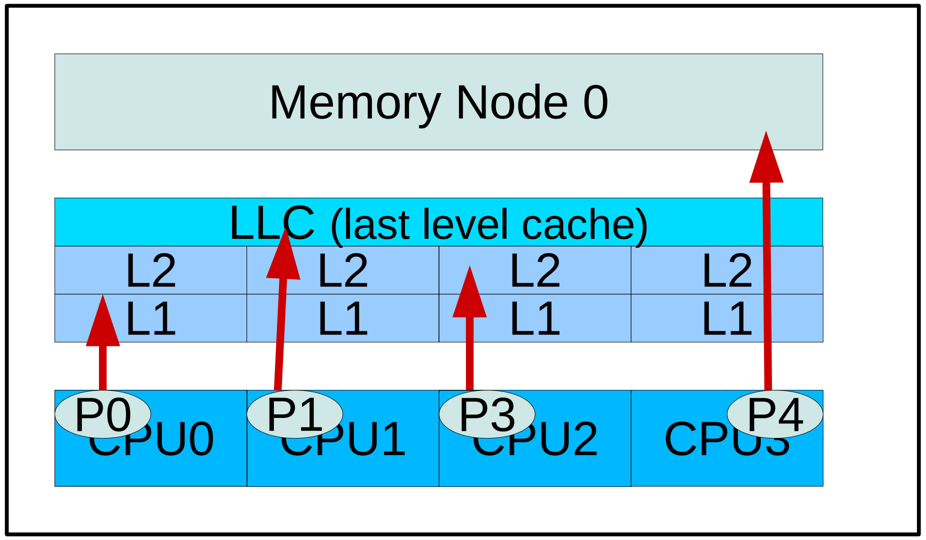
First:
CPU1 issues a read request
For the cacheline to the
"home" node that owns the
Memory.

Node 2 has a modified copy of that cacheline.

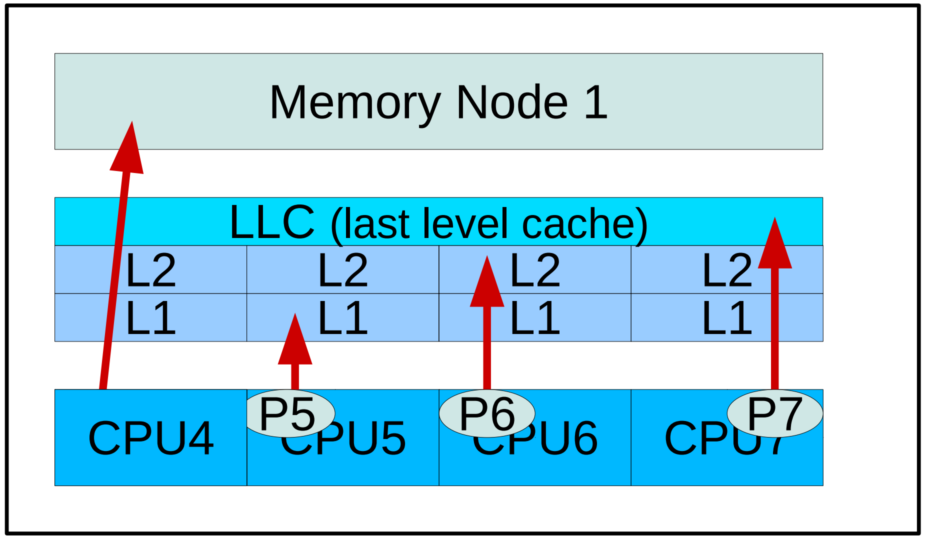
In the ideal world:

All processes and memory are isolated to their own NUMA nodes.

Node 0



Node 1



In the “slightly less than” ideal world

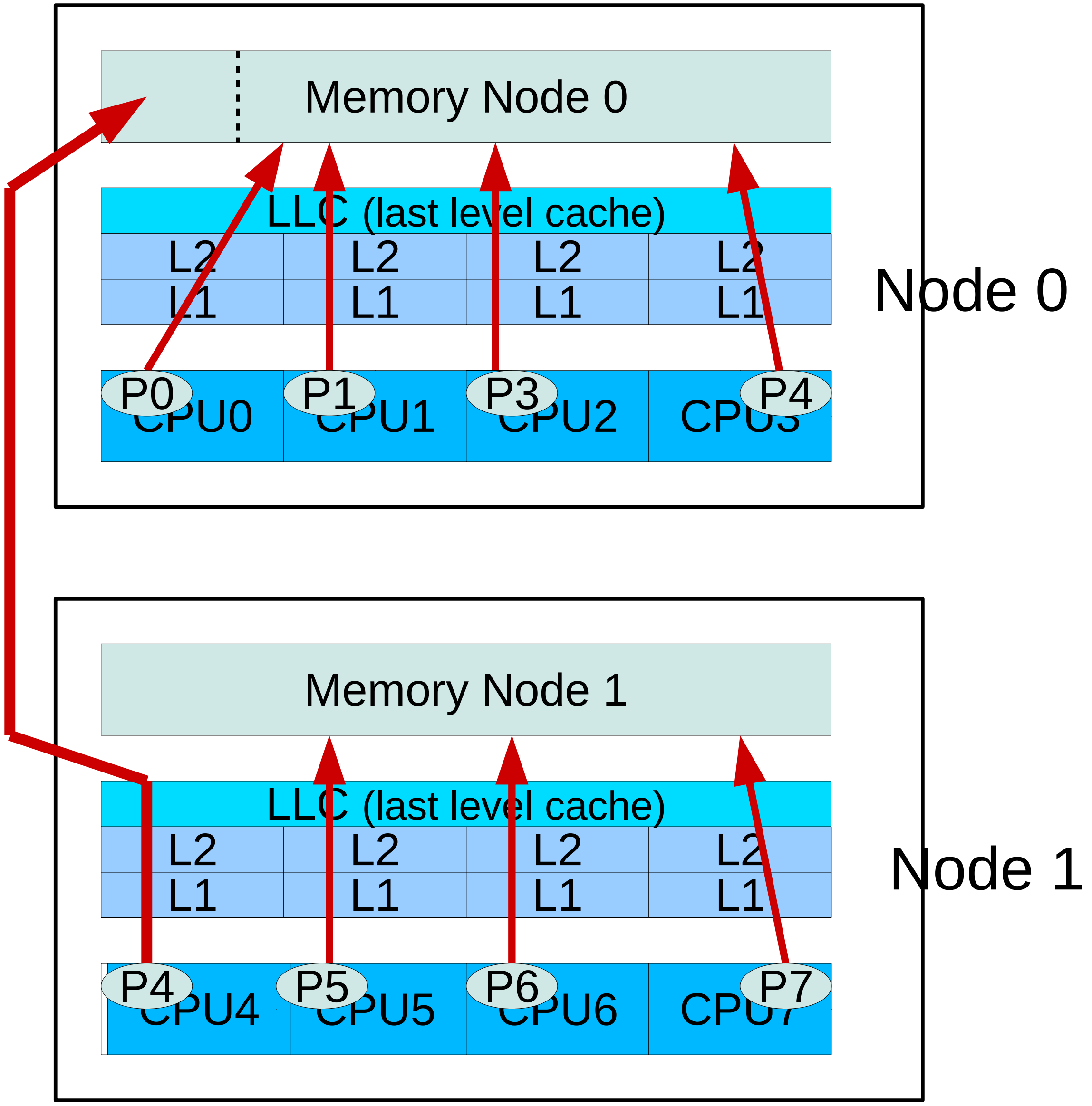
“Sole user” of remote memory.

Not too bad if:

1. It fits in local node 1 cache

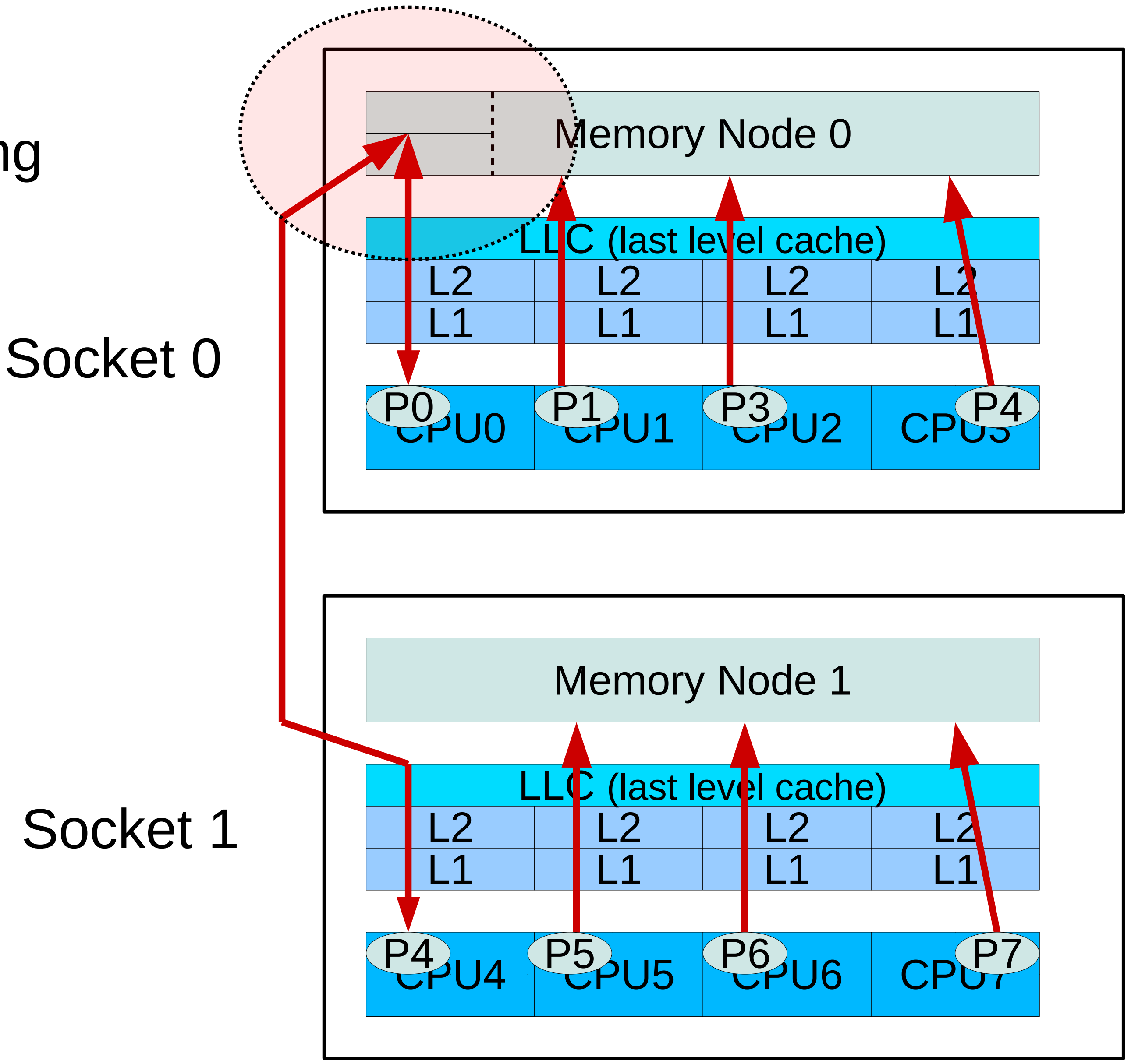
2. It stays in local node 1 cache

3. Your node is the only node accessing that memory.



False Sharing - Where it can hurt the most

Multiple NUMA nodes accessing same memory cacheline.



Basic triage steps

What does my system layout look like?

- lstopo

Where is my program’s memory located?

- numastat

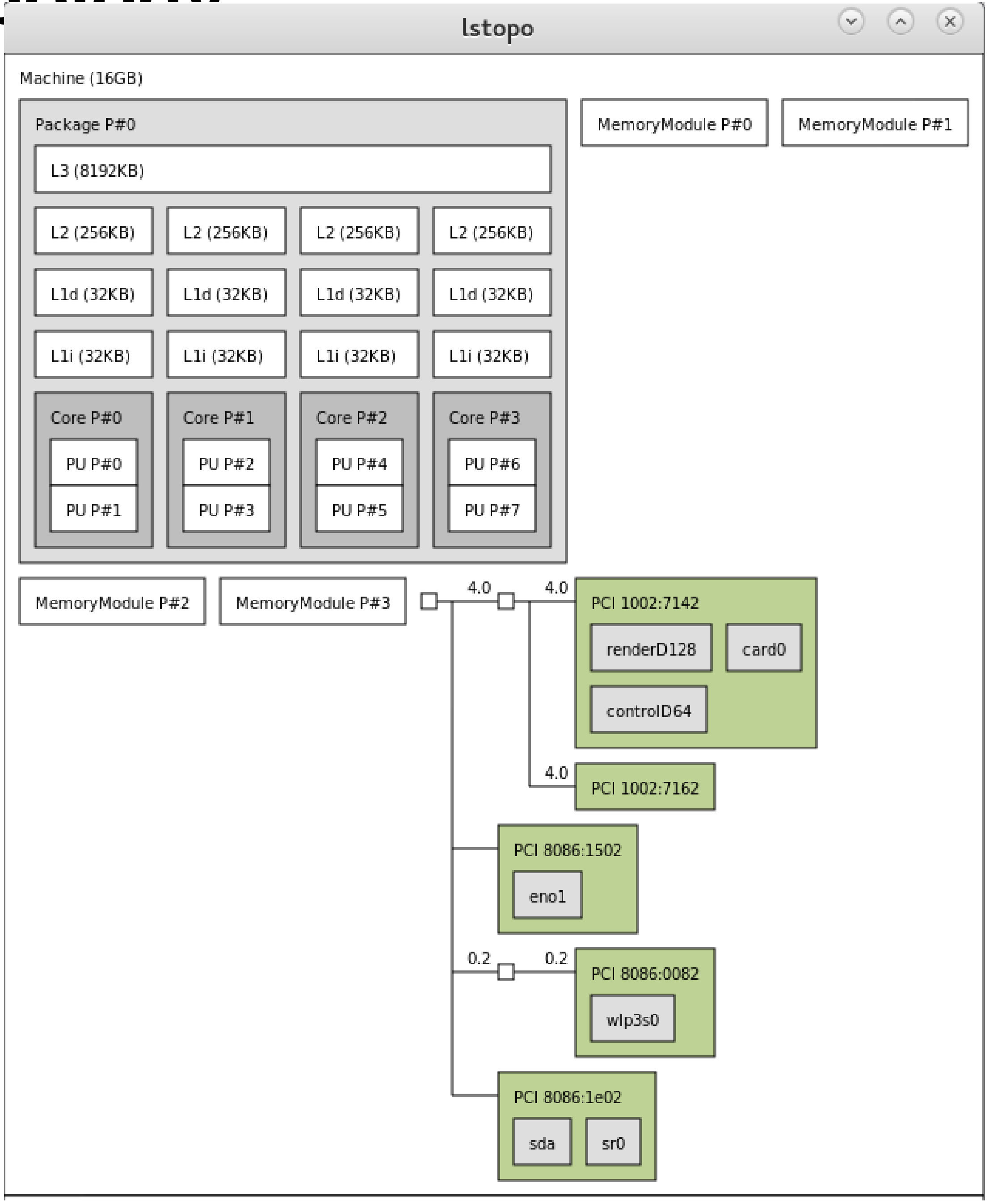
Where are my program’s threads executing?

- *ps -T -o pid,tid,psr,comm <pid>*
- Run “*top*”, then enter “*f*”, then select “*Last use cpu*” field.
- trace-cmd

Where is the memory my program is accessing?

- perf mem
- numatop [Intel]
- perf c2c

lstopo – to see system topology



Numastat
Where is my program’s memory?

Example:
Look at two unpinned instances of SPECjbb2005.

```
# numastat -c java

    Per-node process memory usage (in MBs)
    PID              Node 0 Node 1 Total
    ---------------  ------ ------ -----
    31855 (java)       3160   6206  9366
    31856 (java)       4891   4481  9372
    ---------------  ------ ------ -----
    Total              8051  10687 18738
```

The memory for each pid is scattered across both numa nodes.

Where is my program's memory? (continued)

Invoke it again, but with numactl pinning:

```
    # numactl -m 0 -N 0 java <...>
    # numactl -m 1 -N 1 java <...>

    # numastat -c java

    Per-node process memory usage (in MBs)
    PID                Node 0 Node 1 Total
    ------------       ------ ------ -----
    30707 (java)         9359     11  9370
    30708 (java)            2   9374  9375
    ------------       ------ ------ -----
    Total                9361   9385 18745
```

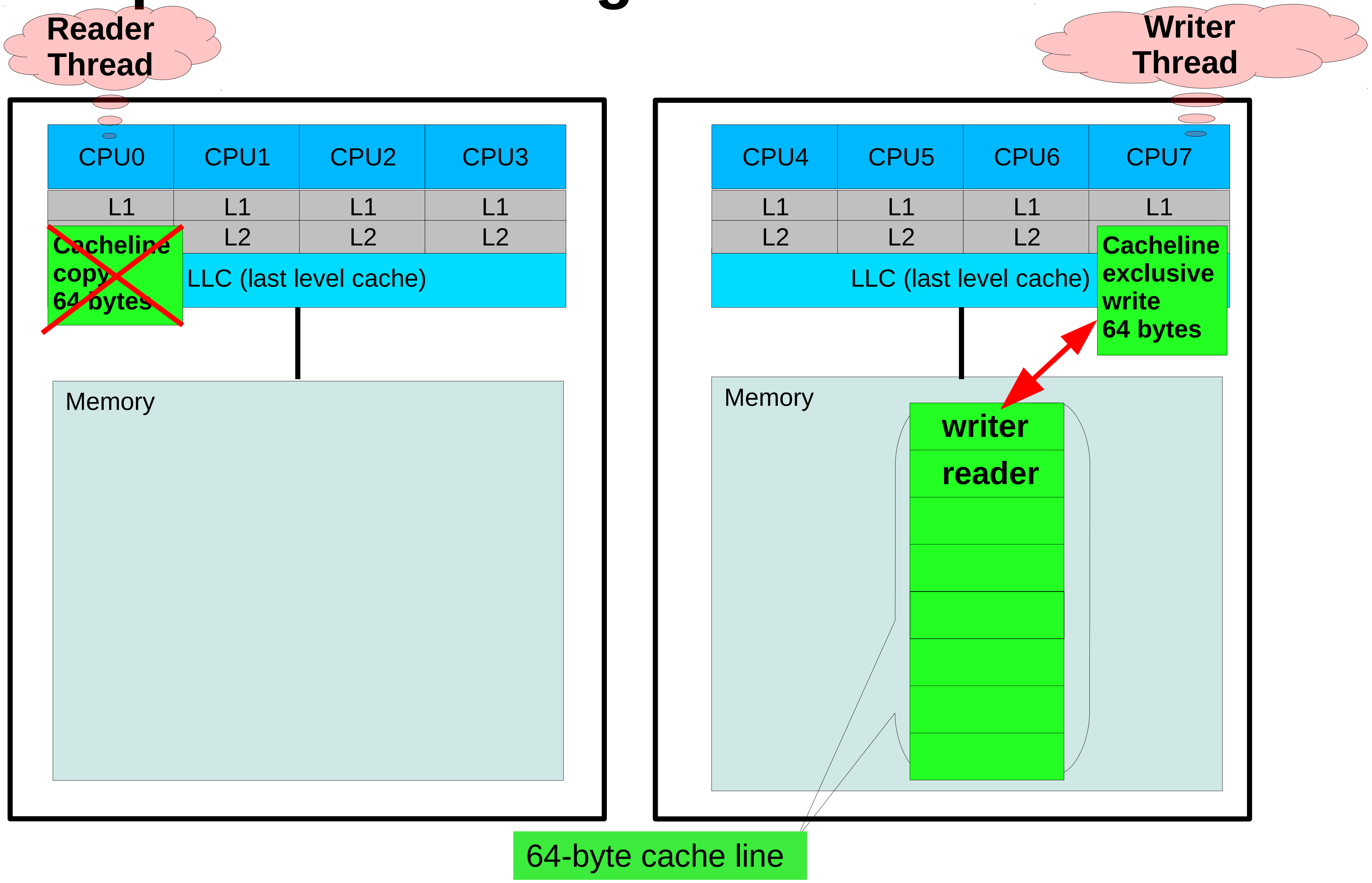
The memory for each pid is confined to a numa node.

Memory location is only part of it.

- numastat shows program's memory location, but not threads.
- The key question: Where are my threads executing <u>and are they contending for the same memory/cachelines</u>?
- Remote HITMs:

  If your program spans multiple numa nodes:
  - Are my threads accessing memory on remote nodes?
  - Are they contending for same memory locations with other threads?
  - Can happen w/ multi-threaded or shared memory programs
  - Just takes contention on one contended memory location to impact performance.
- Local HITMs: Contention between cpu caches on same numa node impacting more – as motherboards pack more cores.

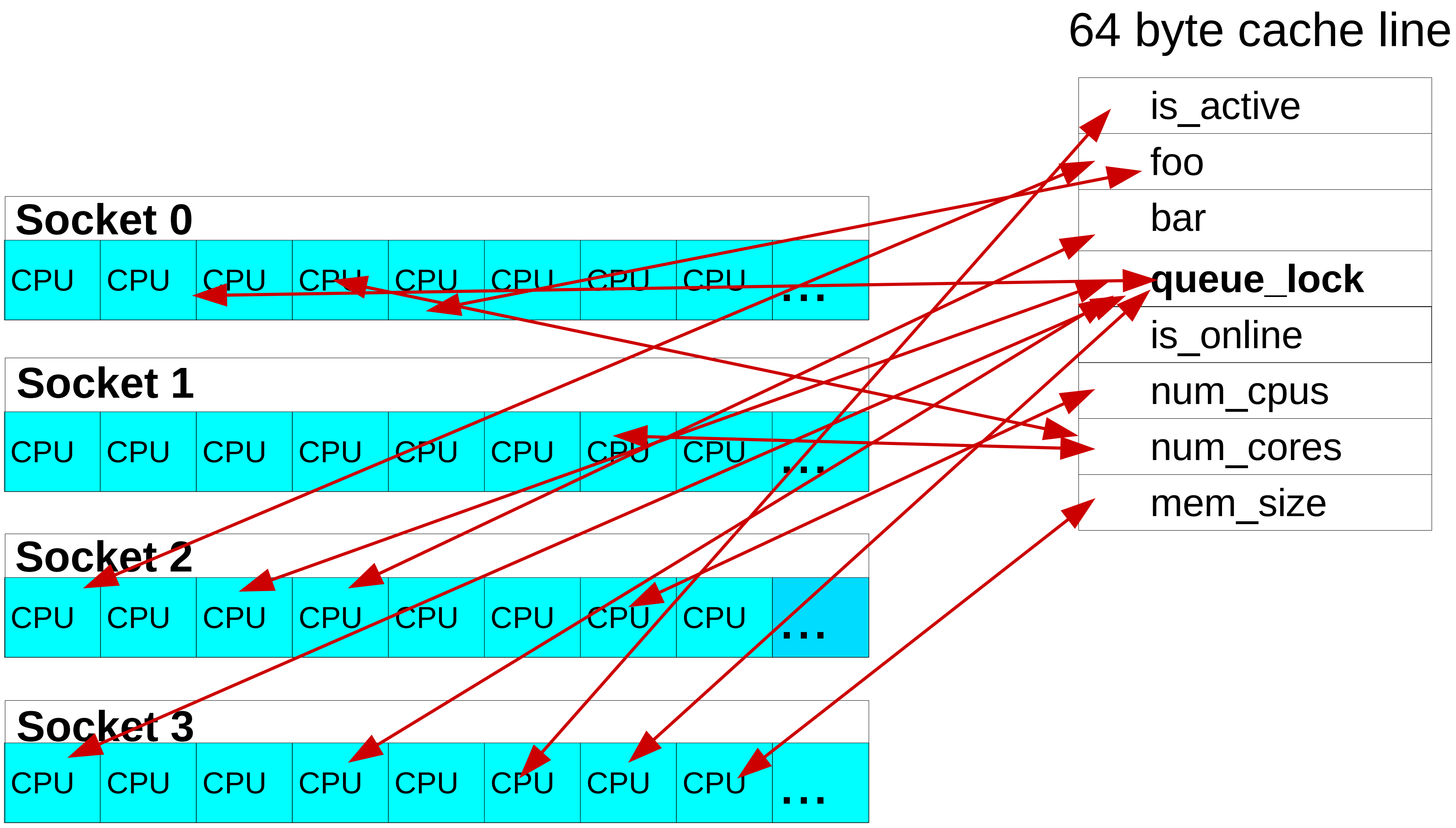
Simple false sharing



Looking a little closer:

- Every time buf.writer is modified:
  - The reader thread’s cacheline copy is disguarded.
  - Must go back for an updated cacheline copy.
  - Or get back in line if other threads are contending for the cacheline.
- With lots of threads and/or large systems:
  - It takes longer for any one of them to access the cacheline.
  - Often lots longer

As your application gets larger... Lots of contention.



CPU cacheline false sharing

- Multiple threads accessing/modifying same cacheline.
- Multiple processes to same cacheline in shared memory.
- Sharing cachelines across numa nodes costly.
- Atomic memory operations make it worse – locked instructions
- Larger systems (8 and 16 numa nodes)

CPU cacheline false sharing (continued)

Approximate latencies for accessing memory.

- L1→L3 caches: **5-30 cycles**
- Local memory: **50-100 cycles**
- Remote memory: **~2x that of local memory**

- On busy systems, (>= 4 sockets), load latencies caused by heavy false sharing often peak over 60,000 clock cycles

How to detect and find this?

New addition to the Linux perf tool:
perf c2c

"*c2c*" stands for "*cache to cache*"

Developed at Red Hat

Merged upstream into 4.9-rc2

Available in RHEL 7.4

Use on Intel IVB or newer cpus

At a high level, “perf c2c” provides:

1) The cacheline’s virtual addr where false sharing was detected.
2) All the readers and writers to those cachelines.
3) The offsets into the cachelines for those accesses.
4) The pid, tid, instruction addr, function name, image filename.
5) The source file and line numbers.

At a high level, “perf c2c” provides:

1) The node & cpu numbers where the accesses are occurring.
2) The average load latency for the loads.
3) Ability to see when hot variables are sharing a cacheline.
4) Ability to see unaligned hot data structs spilling into multiple cachelines.

Extensive usage info in blog at: https://joemario.github.io/

Questions

Compiler & tools tips

Get the latest bits:

Red Hat Developer's Toolset

Developer Toolset 7 beta adds a major update of GCC 7.2 and supporting toolchain

- Addition of Clang/LLVM 4.0.1 compiler set – Technology Preview*
- Addition of Go 1.8.3 compiler – Technology Preview*
- Addition of Rust 1.20 compiler – Technology Preview*

Get the latest bits: Red Hat Sofware Collections

Languages and frameworks

- Node.js v4.4, v6
- Perl 5.20, 5.24
- PHP 5.6, 7.0
- Python 3.5
- Ruby 2.3, 2.4
- Ruby on Rails 5.0

Databases

- MariaDB 10.1
- MongoDB 3.2
- MySQL 5.7
- PostgreSQL 9.5

Web and application servers and HTTP accelerators

- Apache httpd 2.4
- nginx 1.10
- Phusion Passenger 4.0
- Varnish 4.0

Java development tools

- Maven 3.3
- Thermostat 1.6

IDE

- Eclipse IDE 4.6.2 (Neon)

Build steps to minimize contention:

1) Pack read-only/read-mostly variables together.

2) Place the hottest written variables in their own cacheline.

3) Pad cachelines as a small tradeoff for reducing contention.

4) Align data/buffers/structs/c++classes on cacheline boundaries.

5) Lower the granularity of locks (lock smaller chunks of data to reduce contention).

6) Use compile-time asserts to guarantee struct member alignment.

```
_Static_assert( offsetof(struct foo ,bar) %64 == 0,    \
   "struct member bar is not at cacheline aligned offset:");
```

Aligning C++ classes

To align dynamically allocated C++ classes on a cacheline boundary:

Change:

```
    foo *ctx = new Foo(this, tid);
```

to:

```
    void *p  = aligned_alloc (64, sizeof(Foo));
    foo *ctx = new (p) Foo(this, tid);
```

- The above allocates the class on a 64-byte boundary.
- Then use

  "__attribute__((aligned (64)))"

  on individual class members needing 64-byte alignment, (for items allocated in the class).

Long load latency instructions. Are they in your critical path?

Start your application.
Then run perf to collect samples:

```
# perf mem record -U -- -a sleep 3
or
# perf c2c record --all-user -- -a sleep 3
```

Then run:

```
# perf mem report --stdio
or
# perf script
```

Then post-process the output (awk, sed, sort, etc) to find your longest loads, or to get a glimpse for where you’re getting your data.

(e.g.: local or remote cache, local or remote mem, modified local or remote cache).

Tools I use to find who is interrupting a thread

stap cycle_thief.stp

at https://sourceware.org/systemtap/examples/

trace-cmd

which is a wrapper built on ftrace

hwlatdetect

to find SMIs or anything causing the hdwe to get in the way.

Questions