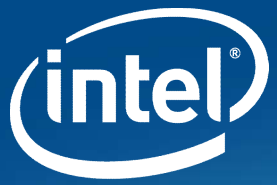


Software

CAN WE SHARE NICELY? HOW TO DEAL WITH NOISY NEIGHBORS

Tony Luck tony.luck@intel.com

October 21st 2017

Agenda

Problem statement

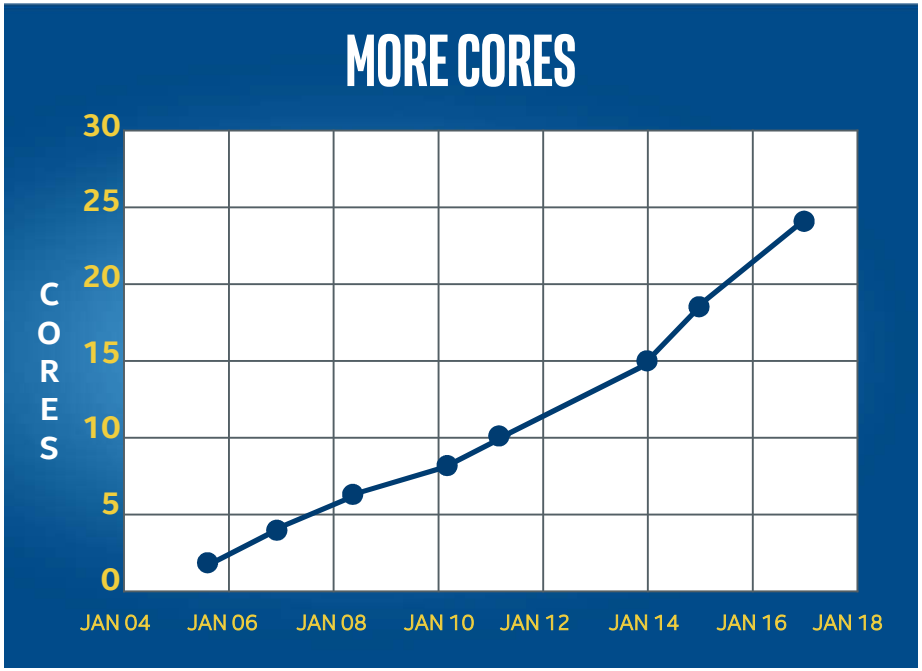
Measuring

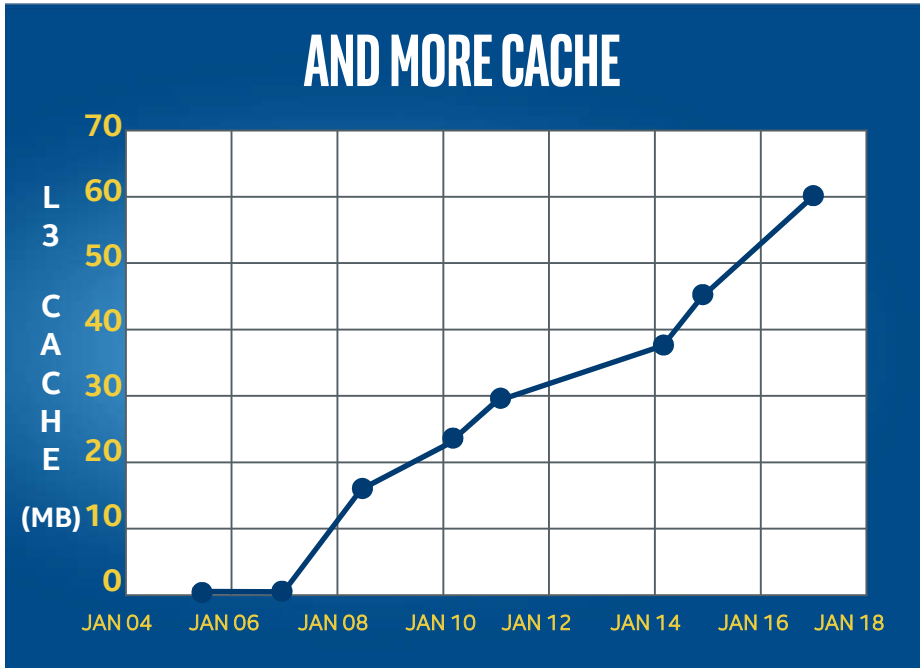
Control

Linux implementation

Results

Moore's law means…

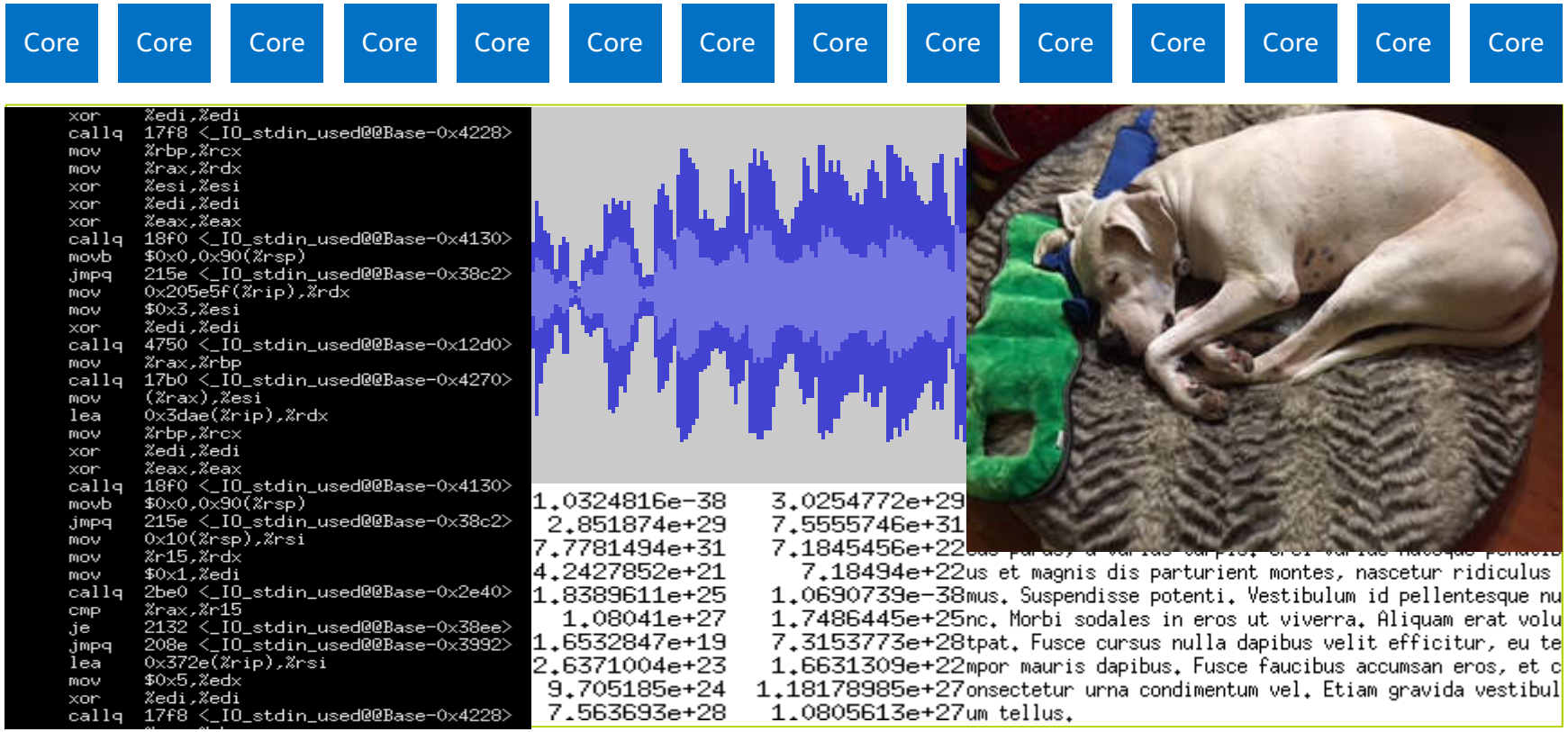




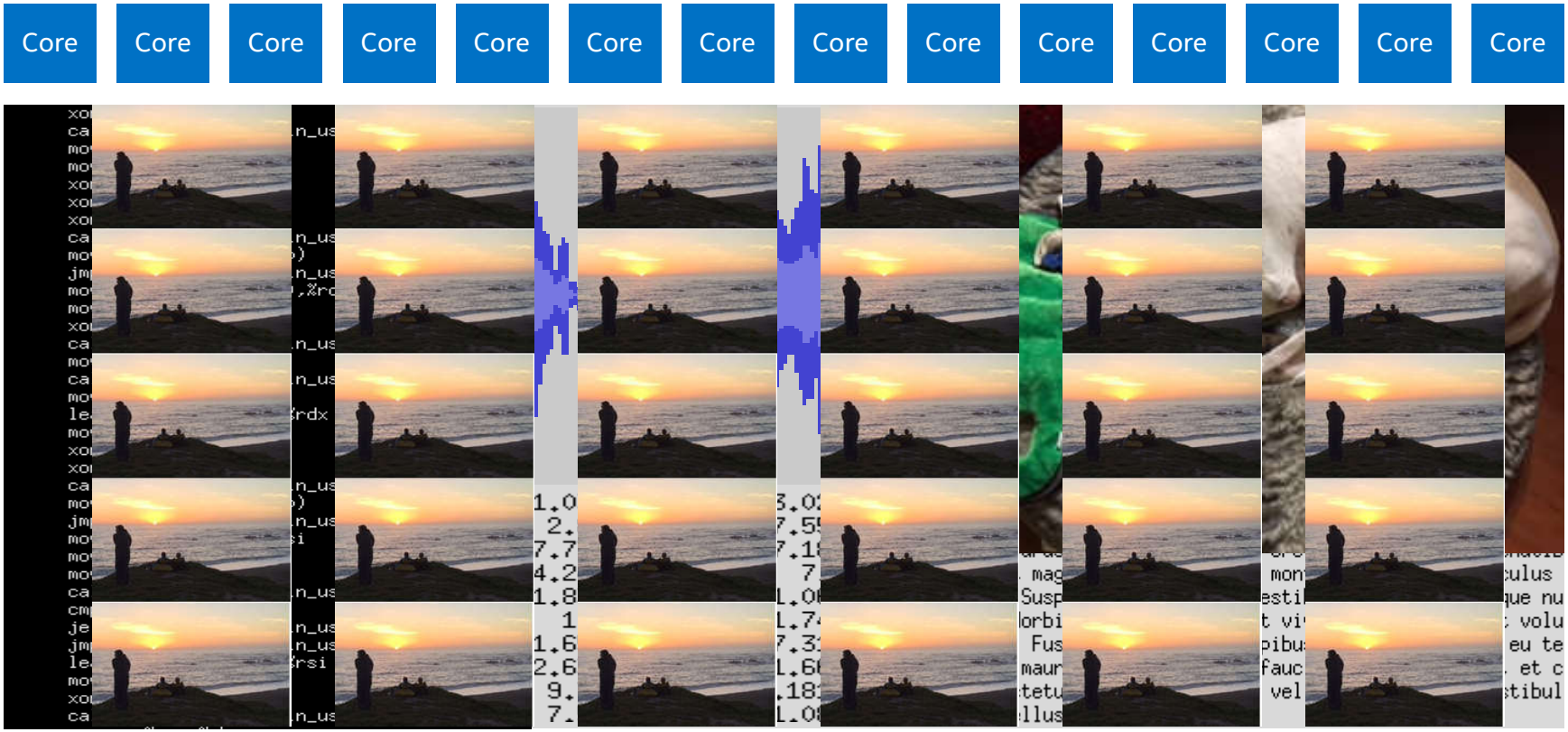
But even though cache has grown as cores have been added, we sometimes see inconsistent performance


Let’s peek inside a cache

When things are going well



When a “noisy neighbor” starts running



Measuring

Conventional performance counters

Hardware increments a counter every time an event happens (e.g. cache miss)

When the OS switches to run a task, software reads the counter

12150615

Before switching to another task, software re-reads the counter

17760704

Subtraction tells us how many events happened during the interval:

17760704 minus 12150615 = 5,610,089 events

Why conventional counters don't help here

If we want to measure how much cache is being used by a process, counting "cache hits", "cache misses", or "cache fills" doesn't help us.

While a process is running	It may load a new cache line, which may evict another line owned by the same process. So cache footprint is not increased.
When a process isn't running	Cache lines will be evicted by other processes reducing the cache footprint.

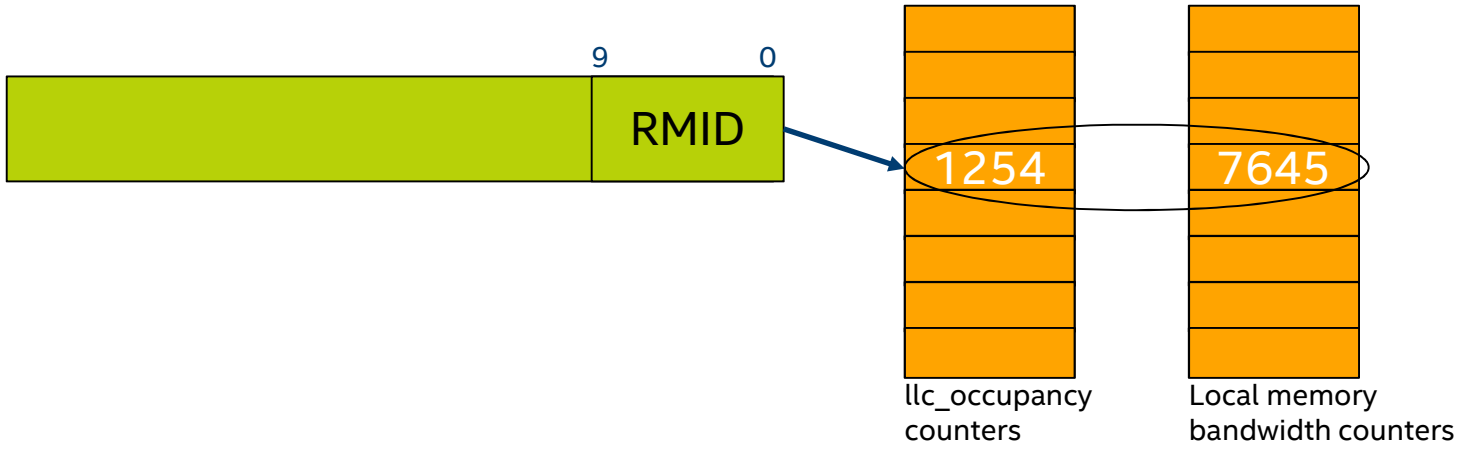
RDT first concept: Resource Monitor ID: “RMID”

For each RDT event, hardware maintains an array of counters

- Number of counters is model specific, enumerated by CPUID

OS assigns an RMID to a task (likely multiple tasks)

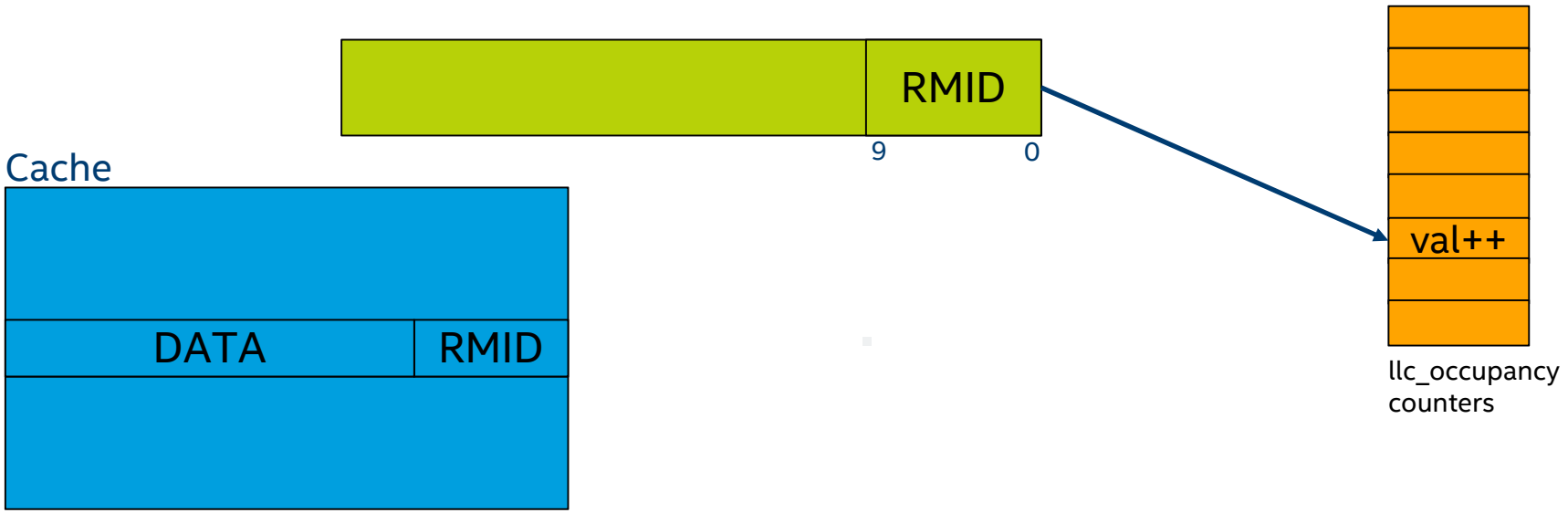
- On each context switch the OS writes that RMID to the IA32_PQR_ASSOC MSR so hardware knows which counter to use.



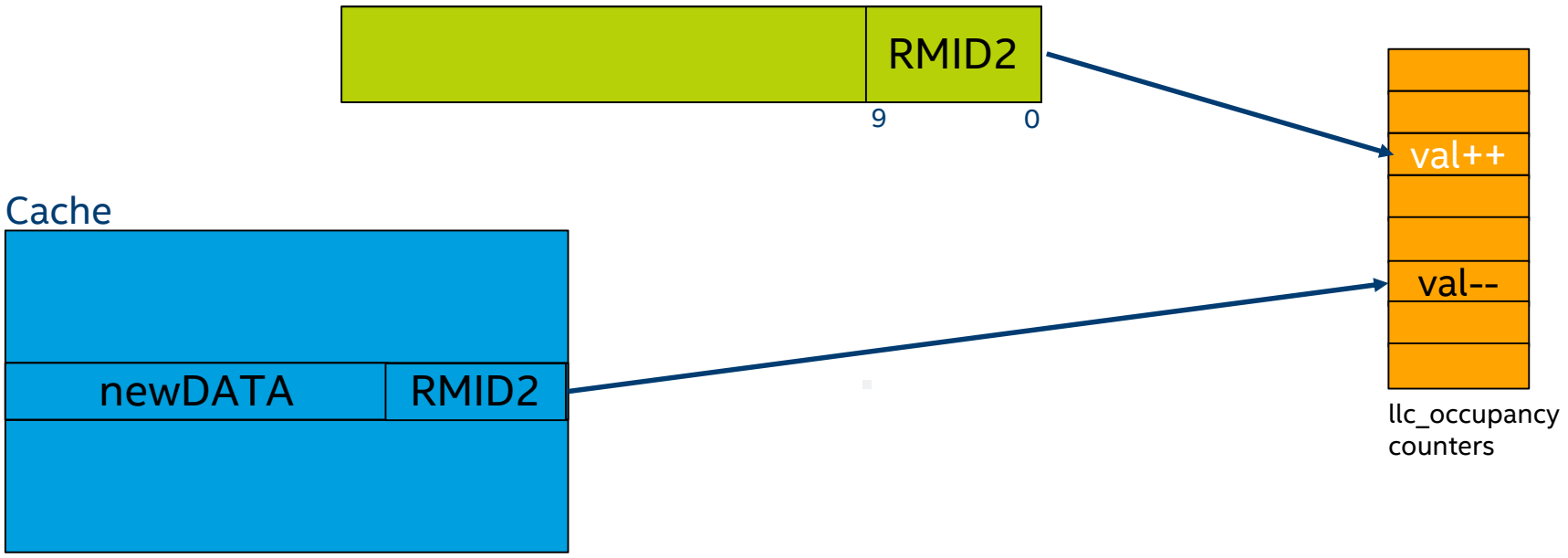
RMID in action – cache allocation

When we allocate a cache line, we increment the counter indexed by IA32_PQR_ASSOC.RMID.

- Also save the RMID along with the data in the cache.



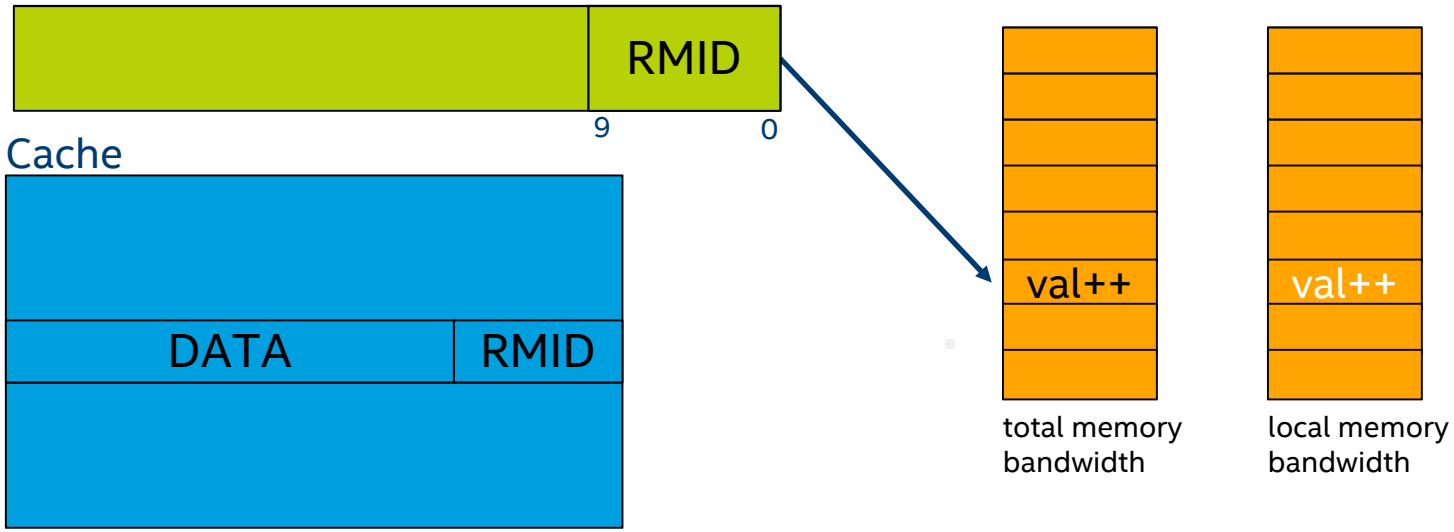
RMID in action – cache eviction/allocation



RMID in action – memory bandwidth

As data is brought into or evicted from the cache, we increment a counter for total memory bandwidth.

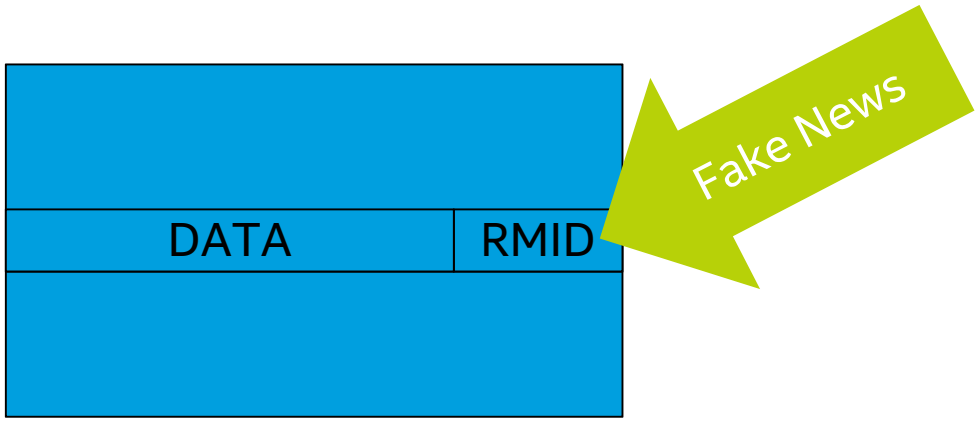
- If data comes from or goes to memory local to the current socket, we also increment the counter for local bandwidth.



Implementation note

This is a useful “mental model” for how this works.

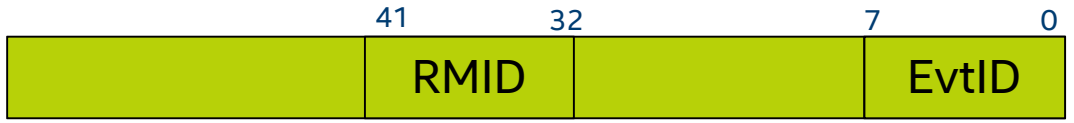
- Hardware doesn’t really store the whole RMID with every cache line … but to a reasonable approximation everything acts as if it did.



Reading RMID event counters

Reading the RMID counters is a two step process.

First software must specify which event and which RMID it wants to read in the IA32_QM_EVTSEL MSR

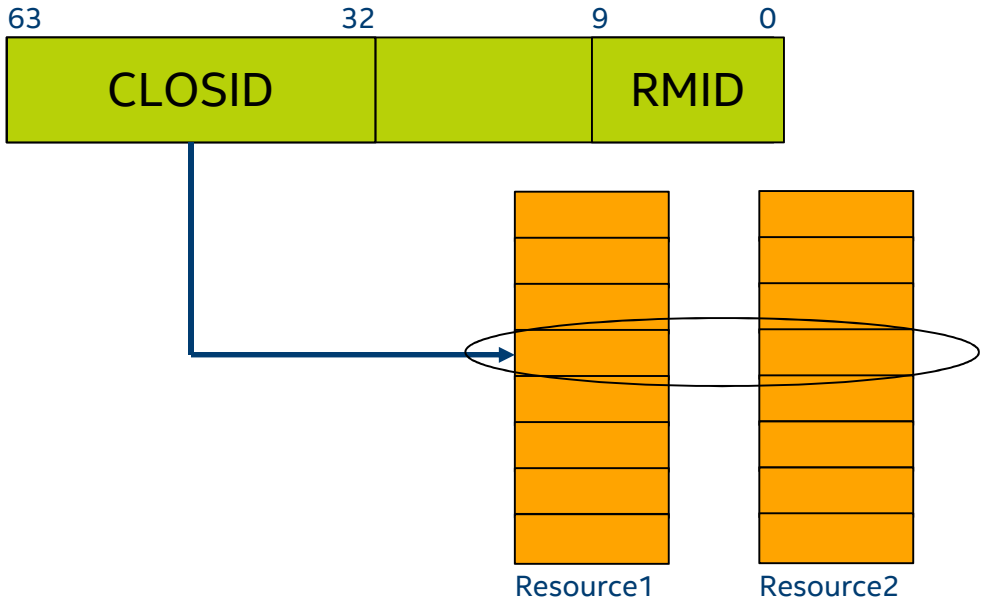


Then it can read the counter from the IA32_QM_CTR MSR


VOLUME
LOCK
TUNE
MANUAL TUNING
TIME ADJ
REBOIL
DISPENSE
SLOW
UNLOCK
208
195
175
TEMP SET
195
140
TIMER
Vol
OK
Ch
BAKE
BROIL
OFF
Refill Rinse Agent
Cancel / Reset
0
CONTROL
Cycle Countdown
Half Load
Power Scrub
Auto Wash
Regular Wash
Quick Wash

RDT second concept: CLass Of Service ID: "CLOSID"

Another field in the same IA32_PQR_ASSOC MSR

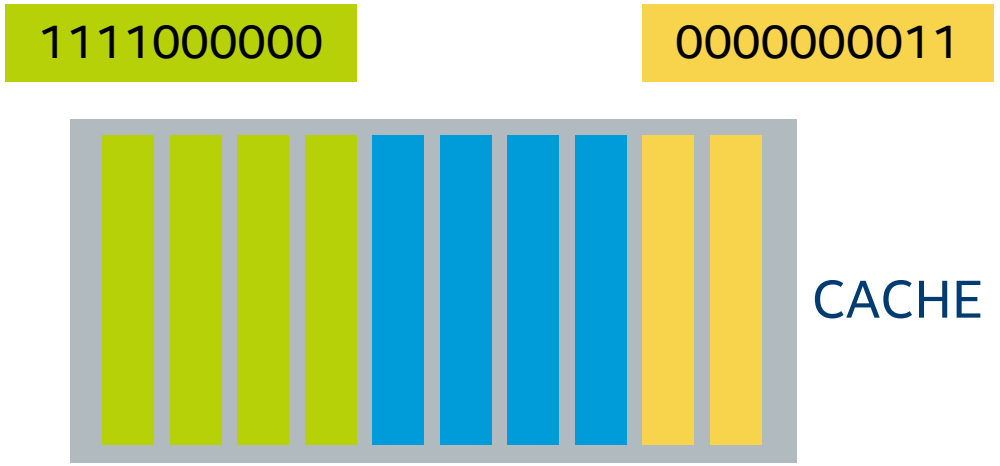


OS also assigns CLOSID to tasks and context switch code updates the MSR. Hardware uses the CLOSID to identify which control to apply to each resource

Cache bitmasks

To control allocations within a cache, the control registers are bitmasks.

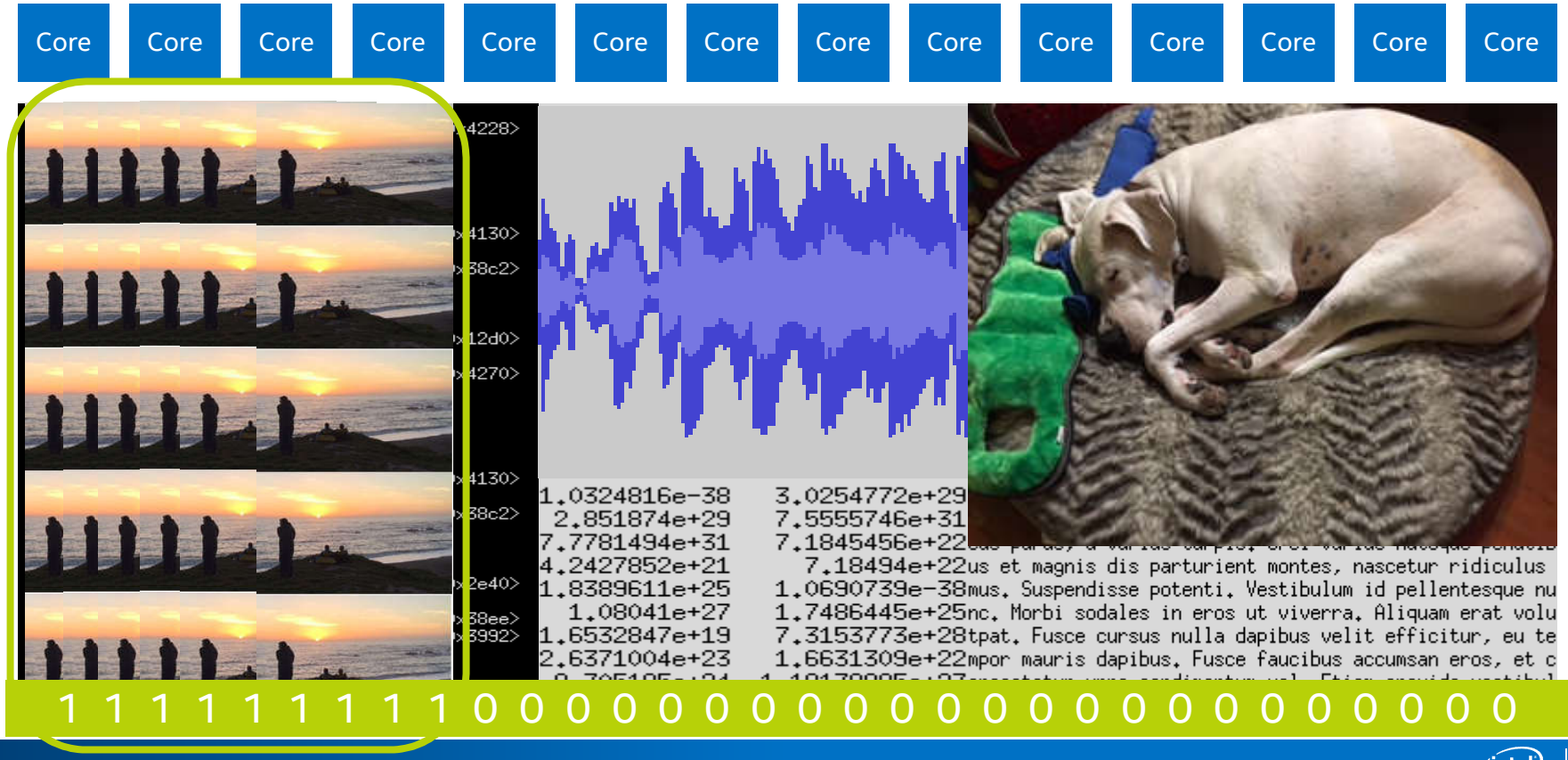
- Granularity is model specific (see CPUID). With 10 bits the cache might be divided like this: 40% to one CLOSID. 20% to another. 40% unassigned.



NOTE

The mask only controls new allocations. A process may get a cache hit on data already in any location in the cache

Use RDT to limit noisy neighbor to part of cache



Control also applies to memory bandwidth

Instead of using bitmasks like we did with cache, we specify a percentage of maximum bandwidth that can be used.

may be allowed to use all available bandwidth

can be throttled down to 30%, 20% or 10%
(model specific – see CPUID)

LINUX IMPLEMENTATION

First attempt at monitoring

In early 2015 (v4.1) patches using RDT via “perf” mechanism were merged.

To report how much cache a process was using Users could run:

```
# perf stat –I 1000 –e intel_cqm/llc_occupancy/ ...
```

In v4.6 more patches merged to add support to measure memory bandwidth.

But there were clues in the code that this wasn’t a good fit

```
/*
 * Ignore the SDM, this thing is _NOTHING_ like a regular perfcnt,
 * it just says that to increase confusion.
 */
```

After a couple of fix attempts, in v4.14 this code was removed and replaced.

First attempt at control

The story for the control parts of RDT was even sadder.

Initially almost everyone thought that it would be a logical progression to build on top of the “cgroups” infrastructure.

Everyone except the cgroups maintainer.

Eventually, after a lot of code was written, and scrapped, and re-written, and scrapped again, this approach was abandoned.

What is in Linux today: /sys/fs/resctrl

First merged into v4.10 this initially only supported the control side of RDT. Some parts of the interface are similar to “cgroups”

- Use mkdir(1) to create new groups.
- Assign tasks to groups by writing the task-id to a “tasks” file.

But others are different.

- Groups are not hierarchical.
- Can control kernel threads as well as user tasks.

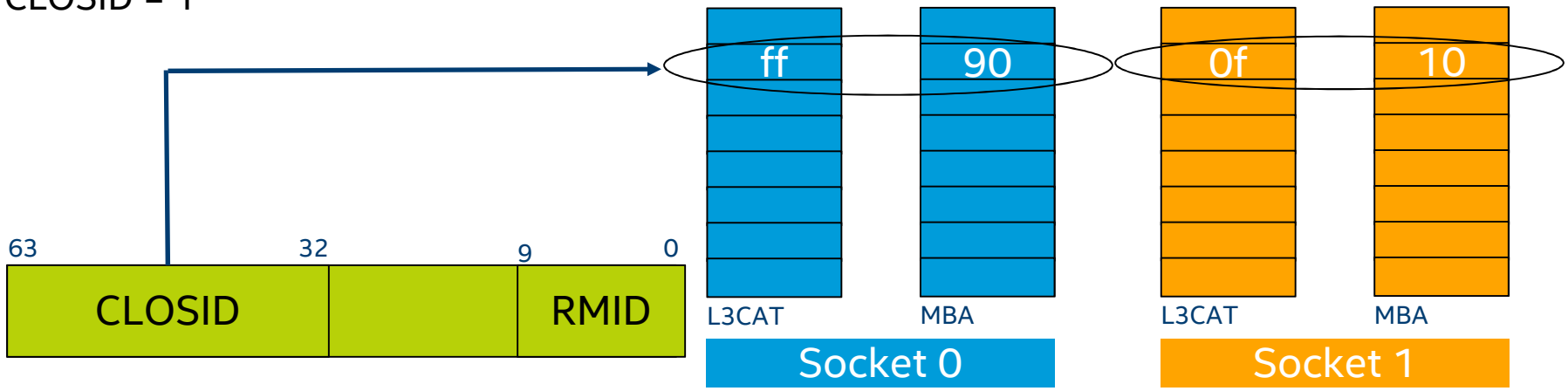
We start with all tasks in the root directory, using CLOSID == 0. We allocate a new CLOSID to each directory, and use a “schemata” file in each to set the MSR registers for each RDT resource.

How “schemata” file maps to h/w registers

Here are the contents of an example schemata file on a two socket machine that supports both cache allocation and memory bandwidth allocation:

```
L3: 0=ff;1=0f
MB: 0=90;1=10
```

Assuming this was in the first subdirectory created, these values would be associated with CLOSID = 1



Using RDT on Linux

The number of CLOSIDs is usually fairly small (16 on some recent Intel® Xeon® processors).

- You can bundle tasks sharing an RDT group. Often this happens naturally as you want to apply the same class of service to an entire container or VMM guest.

On a multi-socket system you may be able to use sched_setaffinity(2) to bind tasks in a group to only run on a specific socket.

- That can help avoid running out of CLOSIDs.

Monitoring (new implementation)

Once we had the /sys/fs/resctrl implementation for RDT control, people started asking questions like:

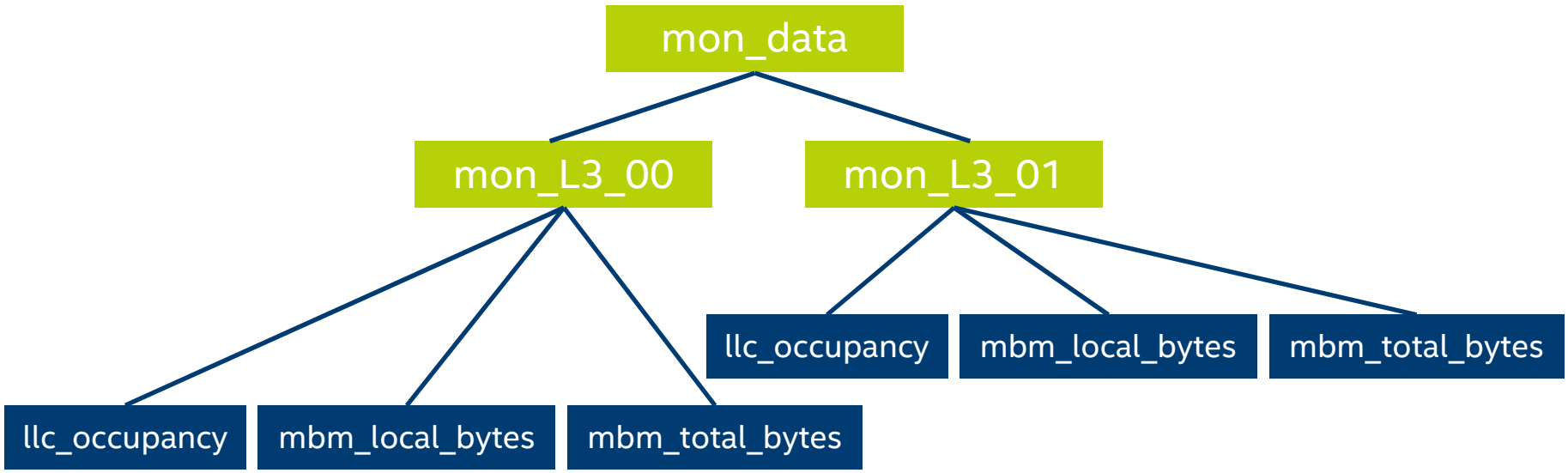
- "How can I measure the aggregate cache occupancy of all the tasks in a group (to see if they are using all of the cache I allocated to the group)?"

This turned out to be very difficult using the "perf" implementation. You would need to attach a perf instance to each task in the group and sum the results:

- You'd need to keep track in case some of the processes forked new tasks
- You might run out of RMIDs
- Since all the perf instances were sampling independently, the results would be very noisy.

Monitor files

The answer. Allocate an RMID along with the CLOSID for each directory. Add some files in each directory so the user can read the counters:



Drilling down inside a group

The next question people ask:

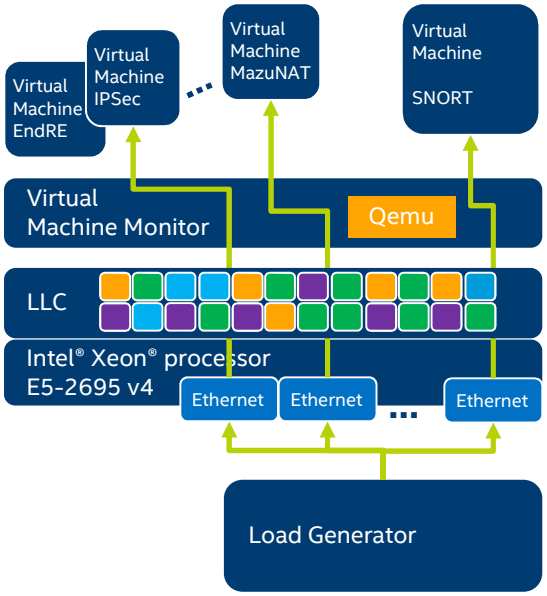
> *"I have hundreds of processes in a group and can see the sum of their resource usage. How can I see which tasks within a group are using more than their share?"*

You can create monitor groups below the "mon_groups" directory in each control directory and move one (or many) processes to that group.

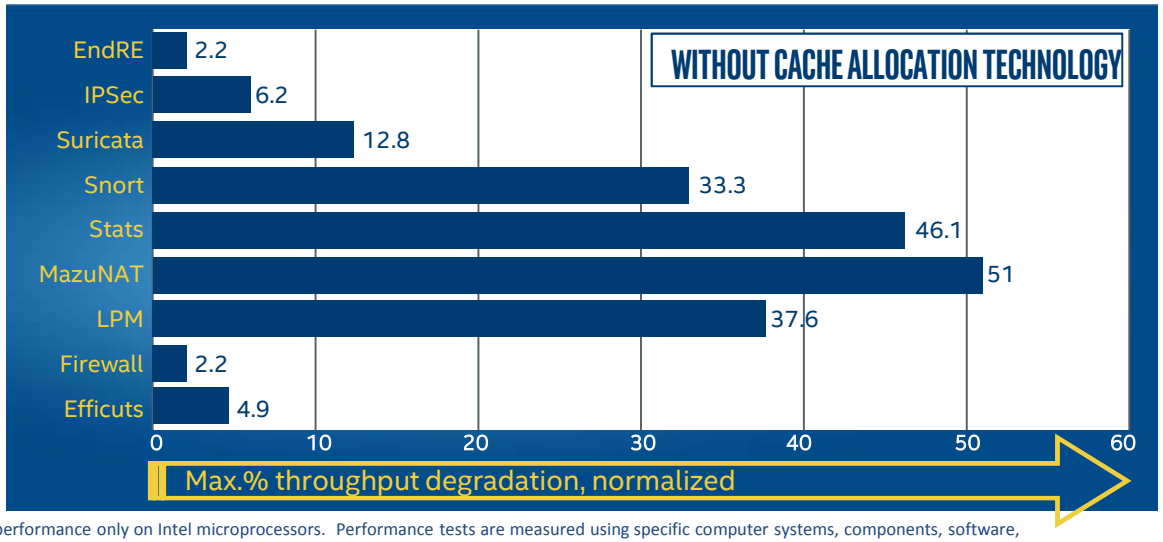
- It is still part of the same control group. But now you have a set of monitor files to look at a subset of the total.
- Since systems generally have many more RMIDs than CLOSIDs, you collect data at a finer granularity.

RESULTS

Intel® Resource Director Technology (Intel® RDT) - University of California, Berkeley

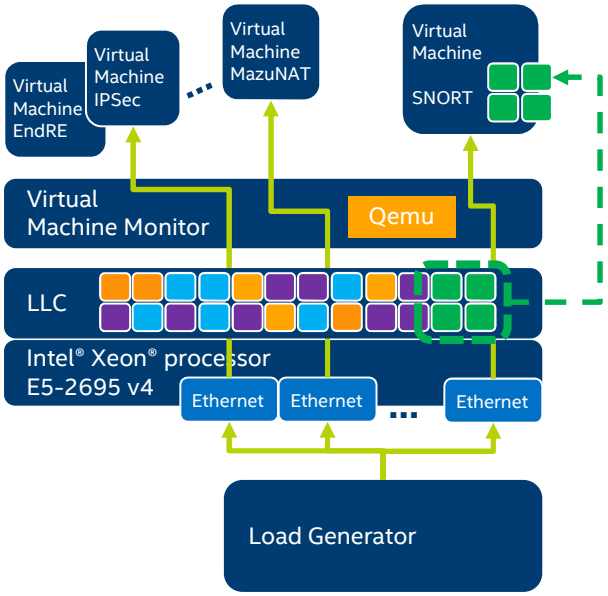


- Network functions are executing simultaneously on isolated cores, throughput of each Virtual Machines is measured
- Min packet size (64 bytes), 100K flows, uniformly distributed
- LLC contention causes up to 51% performance degradation in throughput

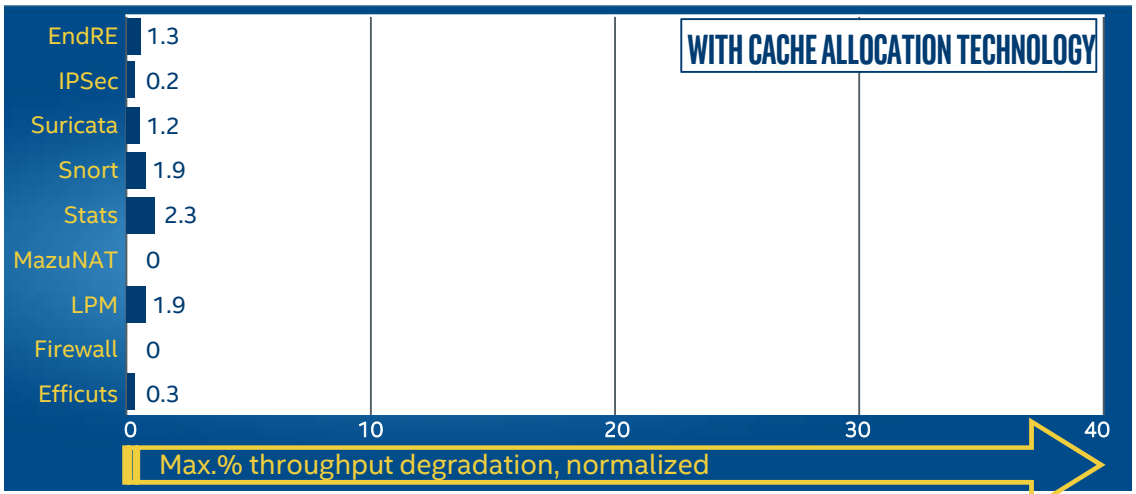


http://span.cs.berkeley.edu

Intel® Resource Director Technology (Intel® RDT) - University of California, Berkeley



- Network functions are executing simultaneously on isolated cores, throughput of each Virtual Machines is measured
- Min packet size (64 bytes), 100K flows, uniformly distributed
- VM under test is isolated utilizing CAT, 2 Ways of LLC are associated with the Network function. Isolation only causes ~2% variation



http://span.cs.berkeley.edu

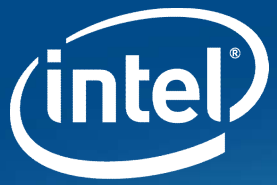
Conclusion

Performance of tasks on a large core count processor can be affected by other unrelated tasks competing for shared resources.

Intel® Resource Director Technology provides mechanisms to:

- Measure the amount of shared resources (L3 cache and memory bandwidth) in use at the granularity of a single task
- Assign classes of service to tasks to limit the amount of shared resources they can consume

Linux v4.14 has interfaces to these hardware features to measure and control tasks



Software

Q&A


intel
Software