Linux QoS framework usage report for containers and cloud and challenges ahead

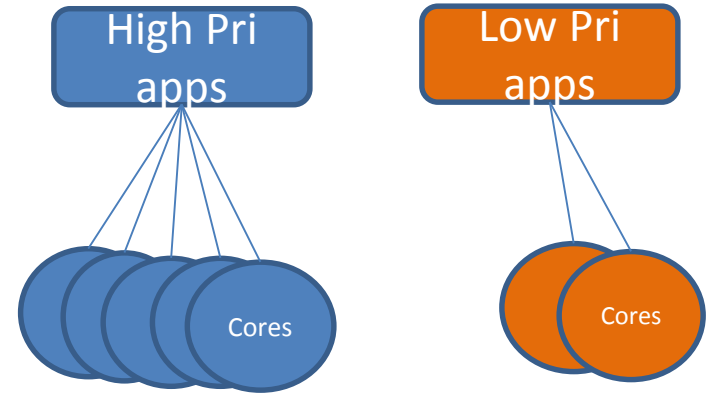
- Vikas Shivappa, Intel

Acknowledgements: Tony Luck, Matt Fleming, CSIG-Intel

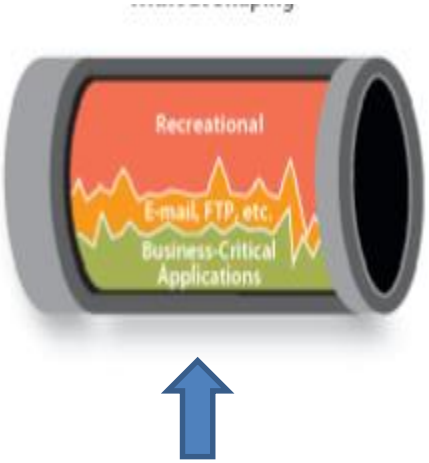
Agenda

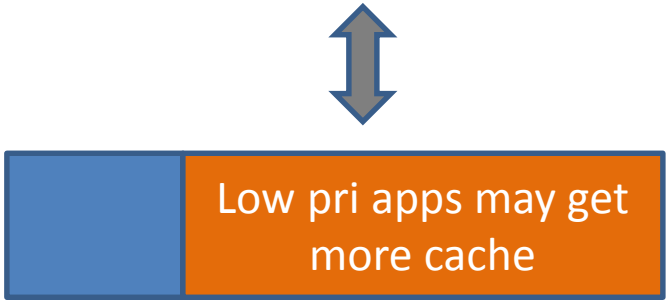
- **Problem definition**
- Why use Kernel QOS framework
- Intel Cache/memory qos support
- Kernel implementation
- Openstack and Container support
- Performance improvement
- Future Work

Without Cache/Memory QoS framework(quality of service)



Increasing cores =>Multithreading





Shared Processor Cache

L3 contention

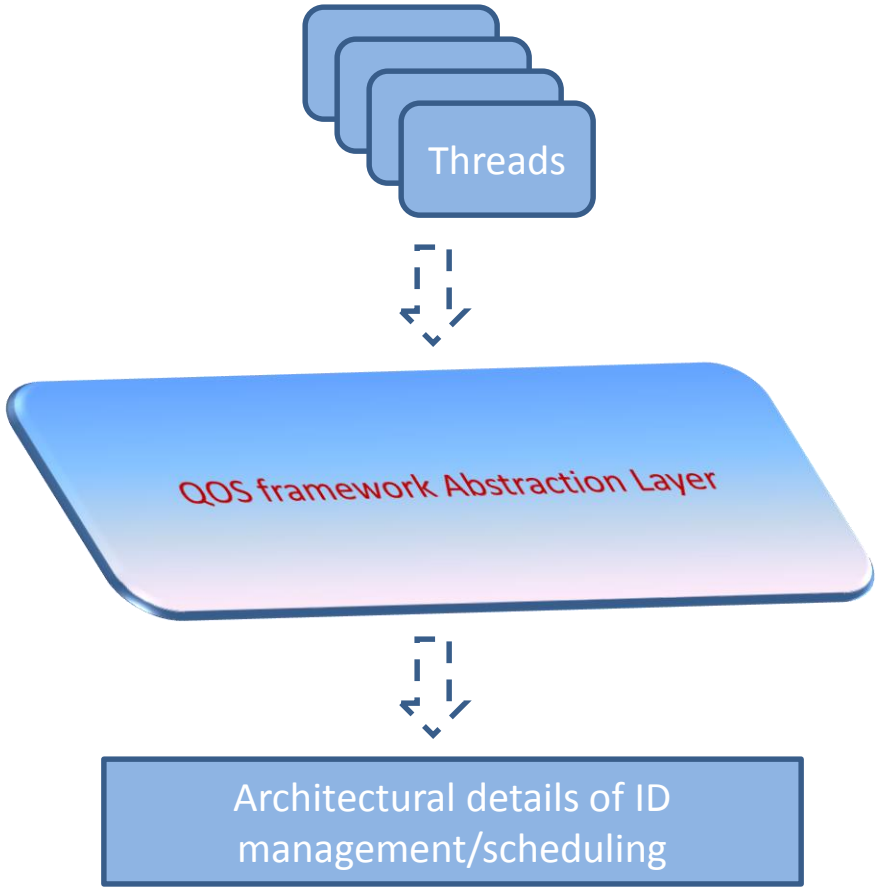
- **Noisy neighbour** => Degrade/inconsistency in response => QOS difficulties
- HPC

Agenda

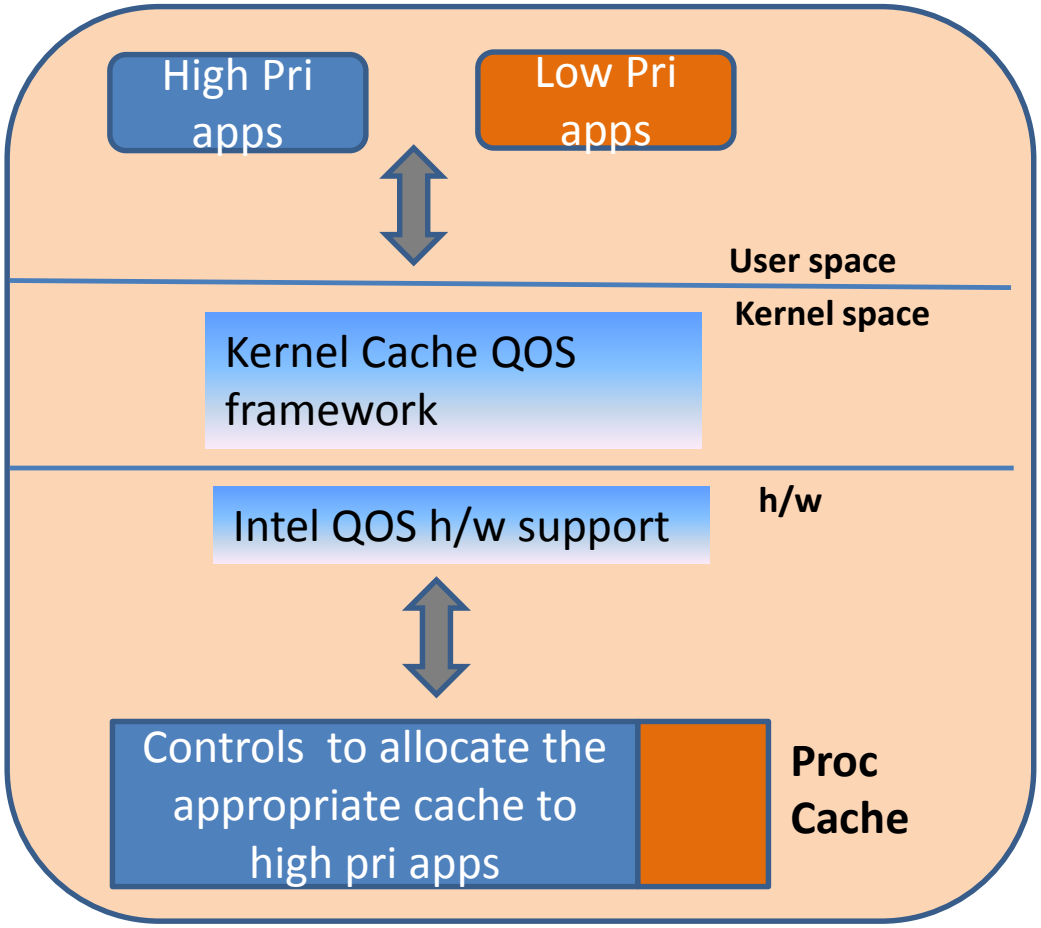
- Problem definition
- **Why use Kernel QOS framework**
- Intel cache/memory qos support
- Kernel implementation
- Openstack and Container support
- Performance improvement
- Future Work

Why use the Cache/Memory QOS framework?

- User friendly interfaces : Perf/cgroup
- Abstracts a lot of architectural/System level details



With Cache QoS



- Help monitor and control shared resources => achieve consistent response => better QoS
  - **Cloud or Server Clusters**
  - **Containers**
  - **HPC**

Agenda

- Problem definition
- Why use Kernel QoS framework
- **Intel Cache/Memory QoS support**
- Kernel implementation
- Openstack support
- Container support
- Performance improvement
- Future Work

What is Cache/Mem QoS ?

- Cache/Memory b/w Monitoring
  - cache occupancy/mem b/w per thread
  - **perf** interface
- Cache Allocation
  - user can allocate overlapping subsets of cache to applications
  - **cgroup** interface (out of tree only, new interface coming up)

Intel QoS Terminologies

- RDT – Resource director technology
  - is basically **“Processor QoS”** under which the cmt/cat/mbm etc are all sub-features
- CMT – Cache Monitoring Technology or also called CQM
- CAT – Cache Allocation Technology
- MBM – Memory b/w monitoring

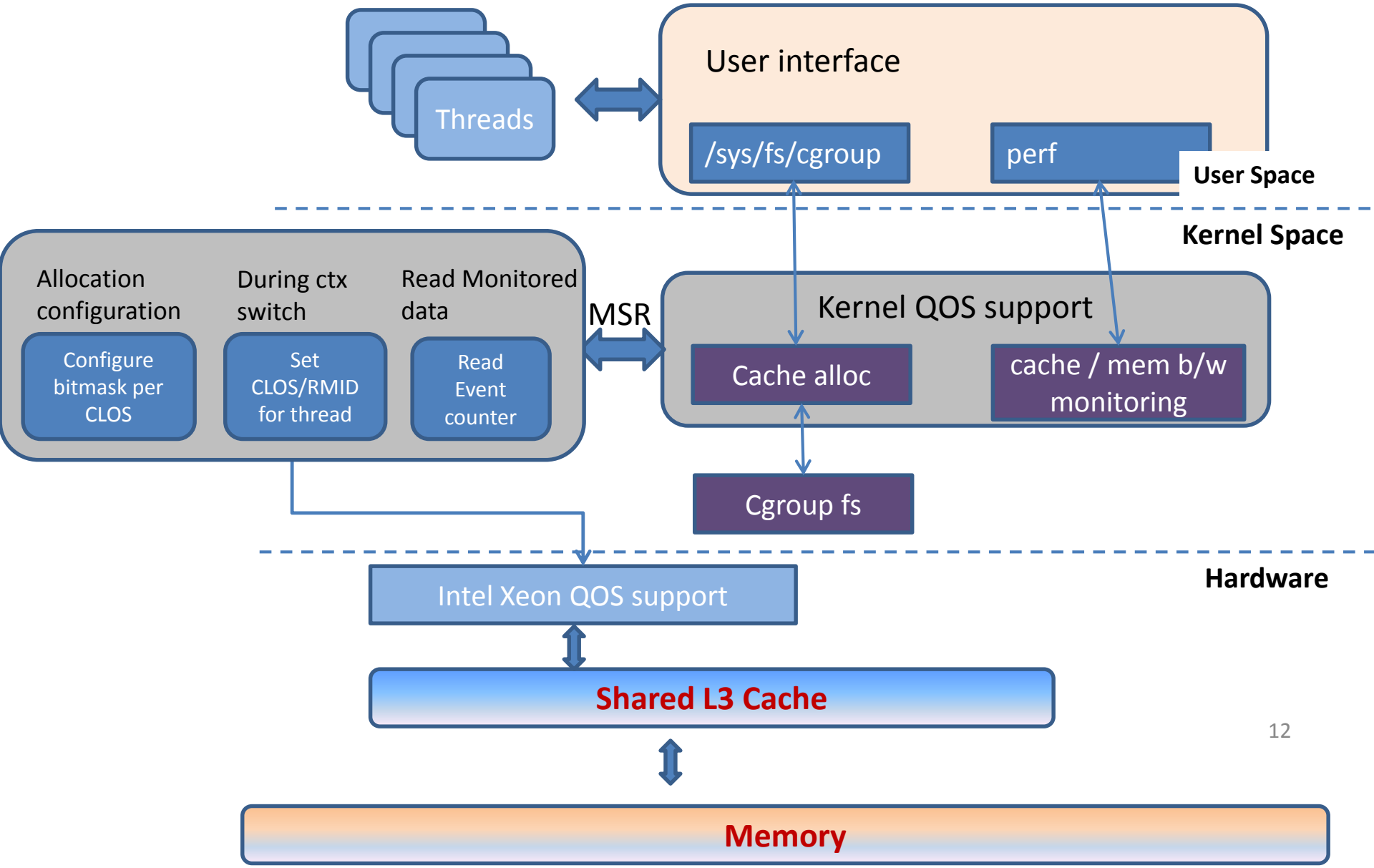
Cache lines ⇔ Thread ID (Identification)

- Cache Monitoring
  - RMID (Resource Monitoring ID) ⇔ PID.
  - RMID tagged to *cache lines allocated*
- Cache Allocation
  - CLOSid (Class of service ID)
  - Restrict *when Cache is filled*
- Memory b/w
  - RMID <=> Total L3 external b/w

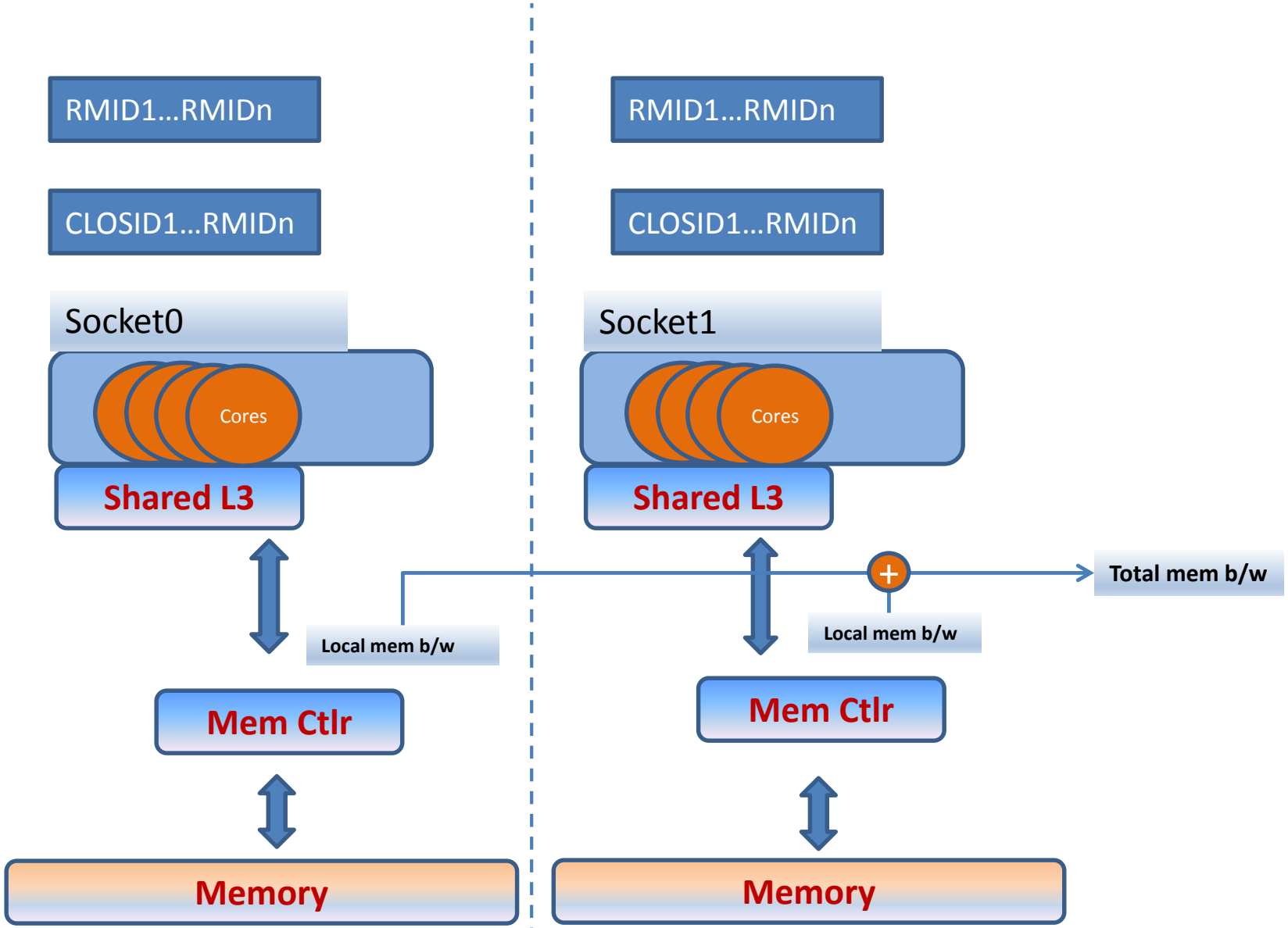
Agenda

- Problem definition
- Existing techniques
- Why use Kernel QOS framework
- Intel Cache qos support
- **Kernel implementation**
- Openstack and Container support
- Performance improvement
- Future Work

Kernel Implementation



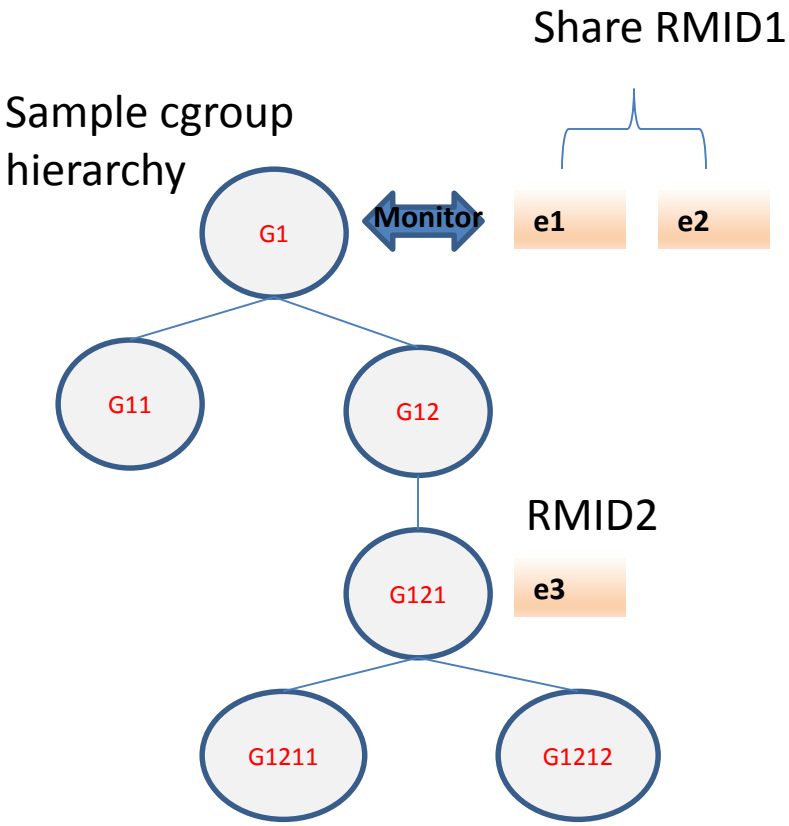
Memory b/w Monitoring



MBM implementation continued

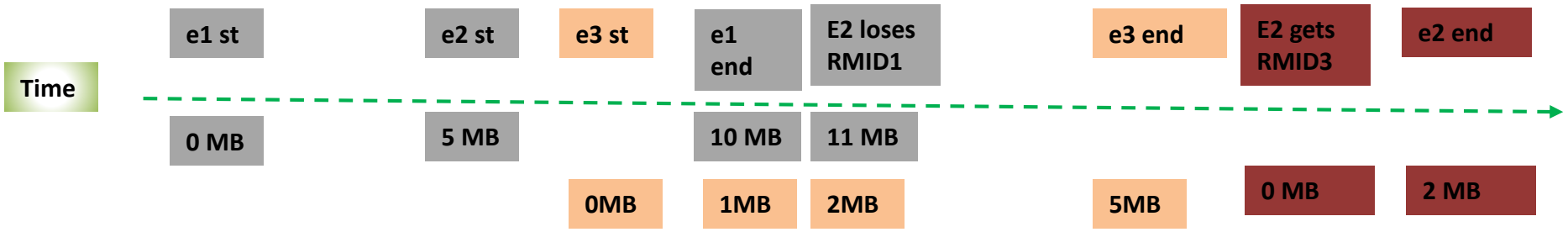
- Typically
  - sched_in
    - prev_count = read_hw_count();
  - sched_out
    - c = read_hw_count();
    - count += c – prev_count;
- Wont work for MBM as we have per package RMIDs
  - Doing the above on 2 core siblings for a PID with same RMID would result in duplicate count.

MBM hierarchy monitoring



- Other considerations
  - Movement of tasks between cgroups
  - MBM counters overflow

- e1 : should read 10MB
- e2 : should read 13MB
- e3 : should read 5MB



MBM hierarchy monitoring

- Implement using periodic updates of the ‘per-RMID count’ as well a ‘per event count’
- This helps take care of all the scenarios
  - Task movement between cgroups
  - RMID recycling
  - Events start counting the same cgroup at different times (they only need to read the current event count)

Usage

Basic monitoring per thread cache occupancy/ Mem b/w

```
labuser@otc-grange-bdw-02:~/src_4.6.6$ ./tools/perf/perf stat -e intel_cqm/llc_occupancy/ -e
intel_cqm/local_bytes/ -e intel_cqm/total_bytes/ -p 2553
^C
 Performance counter stats for process id '2553':

     8,773,632.00 Bytes intel_cqm/llc_occupancy/
        71,114.49 MB    intel_cqm/local_bytes/
        71,114.61 MB    intel_cqm/total_bytes/

      7.022443694 seconds time elapsed
```

- Basic usage example.
- Results display the total cache occupancy and total mem b/w for the thread.

Other Usage modes

- Monitor cgroup
- Per socket monitoring
  - --per-socket does not work as we are not cpu event
  - --per-cpu doesn’t work either
  - Use –C <cpu in the socketN>
- Systemwide
  - Fail if (–a && –t) option (system wide task mode)

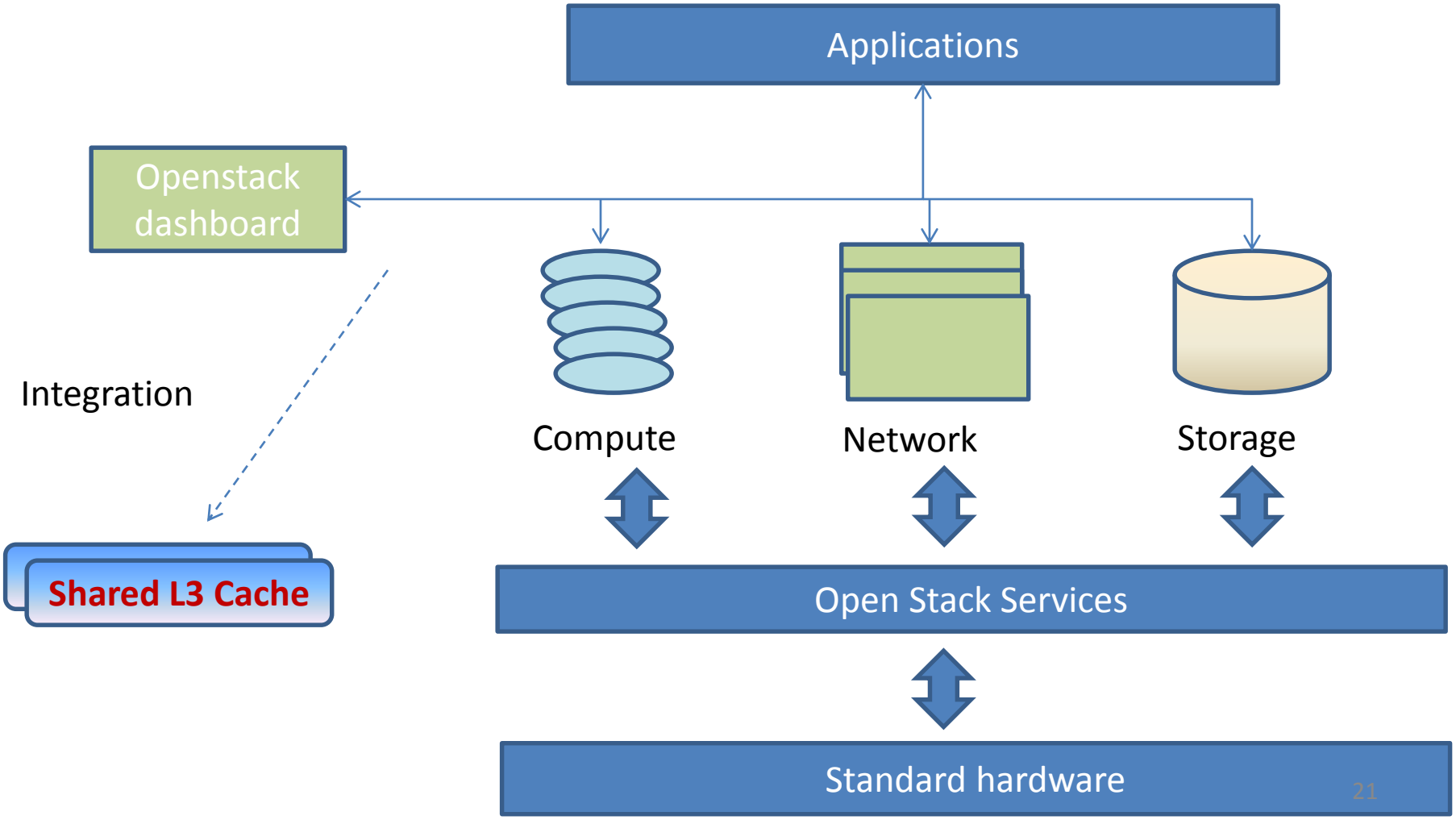
Usage Scenarios

- Units that can be monitored for cache/memory b/w
  - Process/tasks
  - Virtual machines and cloud (transfer all PIDs of VM to one cgroup)
  - Containers (put the entire container into one cgroup)
- Restrict the noisy neighbour
- Fair cache allocation to resolve cache contention

Agenda

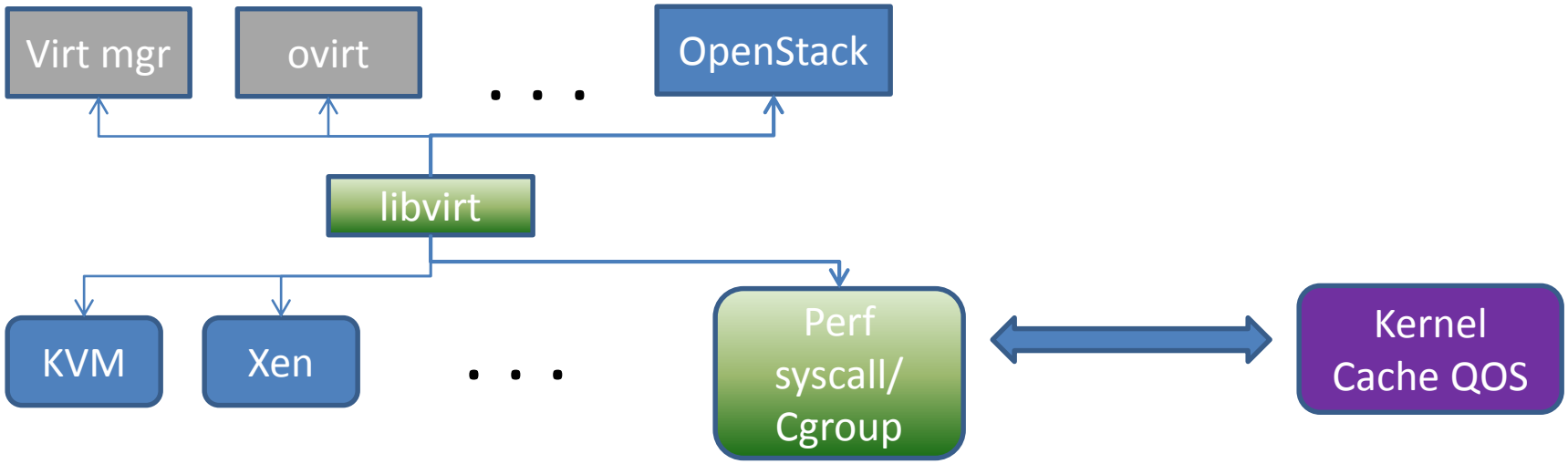
- Problem definition
- Existing techniques
- Why use Kernel QOS framework
- Intel Cache qos support
- Kernel implementation
- **OpenStack / Container support**
- Challenges
- Performance improvement
- Future Work

Openstack usage



Openstack usage …

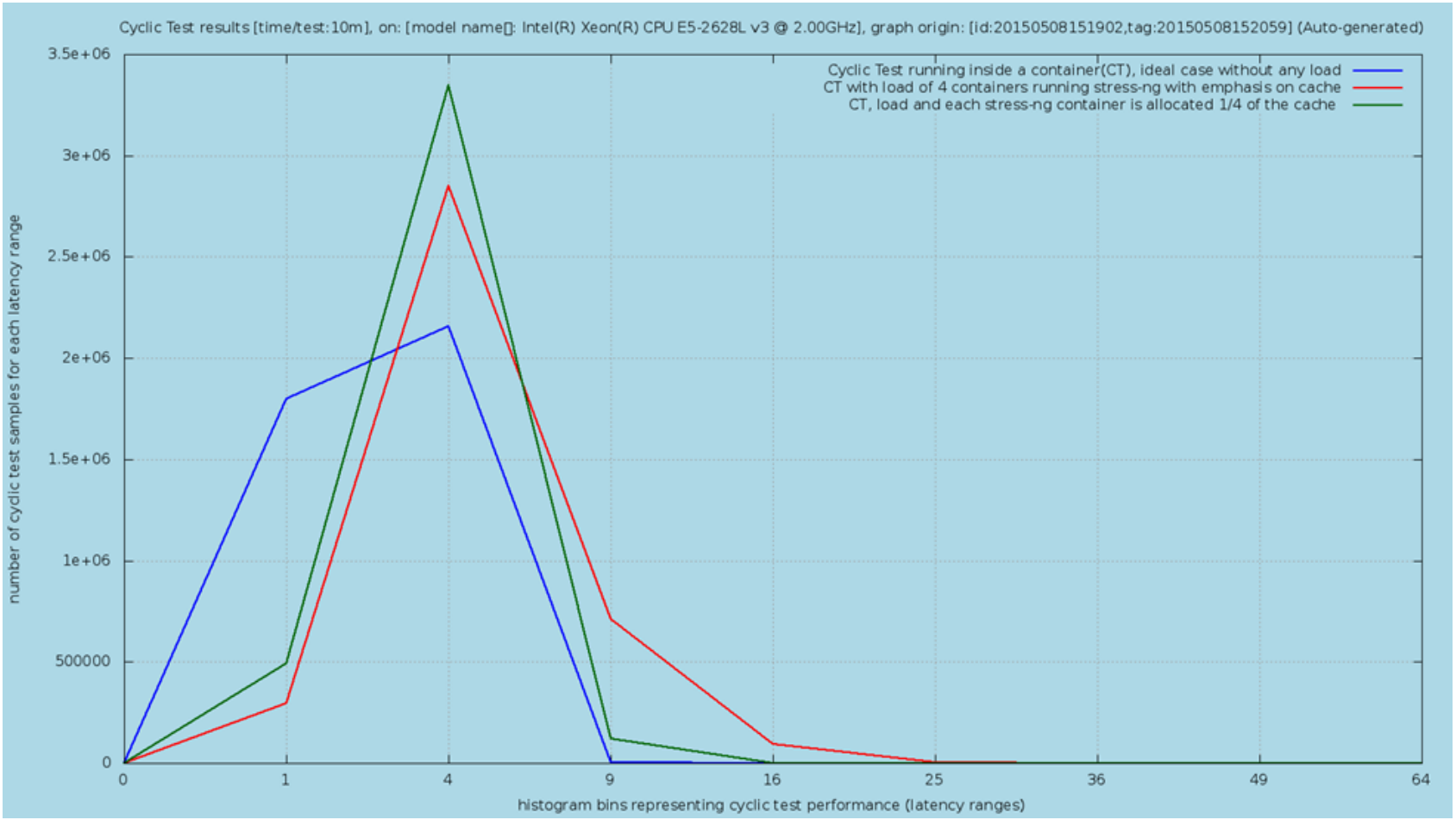
- Libvirt patches submitted (Qiaowei qiaowei.ren@intel.com) – based on kernel QOS framework
- CAT/CMT/MBM was demoed in openstack forums/ conference



Containers support

- Dockers support patch was built to use the new CAT cgroup
- Was simpler change as dockers and systemd already have all the plumbing to use cgroups

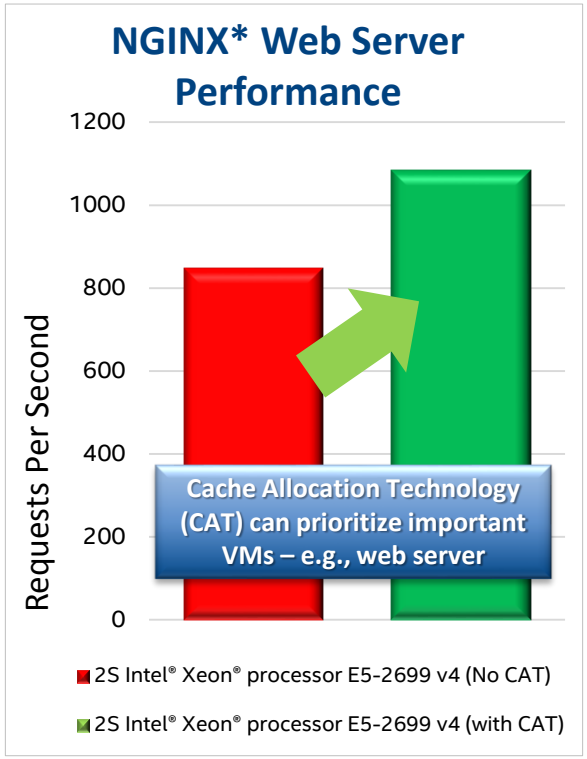
Cyclic tests using docker

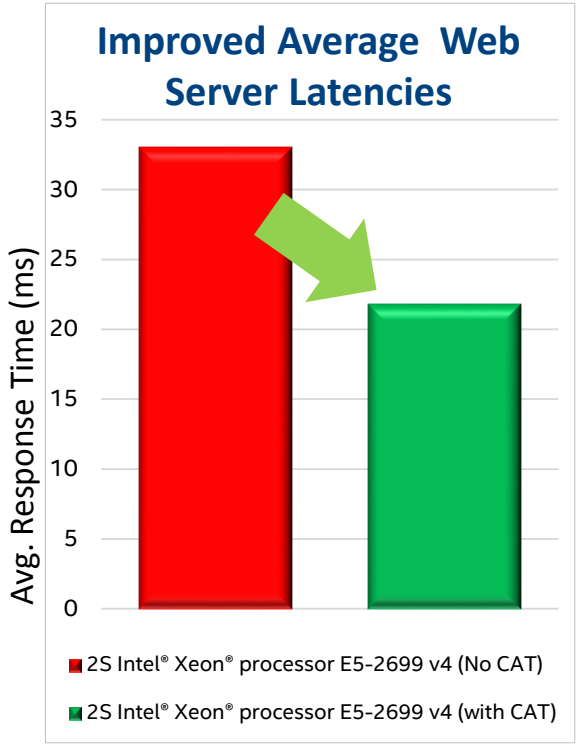


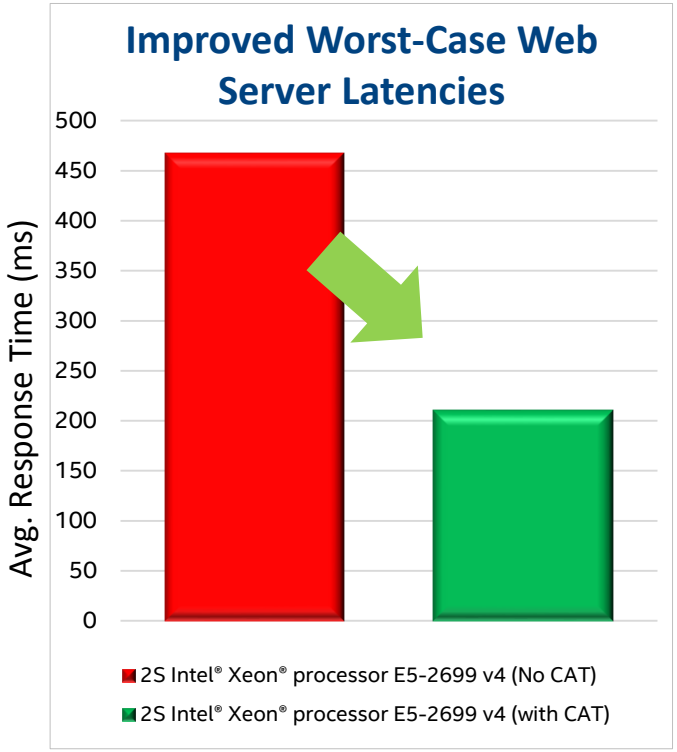
- With CAT(green curve) has a more consistent response latency range comparable to the no-noise scenario (0-16)
- Most of the samples falling the 1-9.

AppFormix* – Orchestration with Containers (Kubernetes)

Workload: NGINX based webserver on Intel Xeon processor E5 v4, 100KB request size

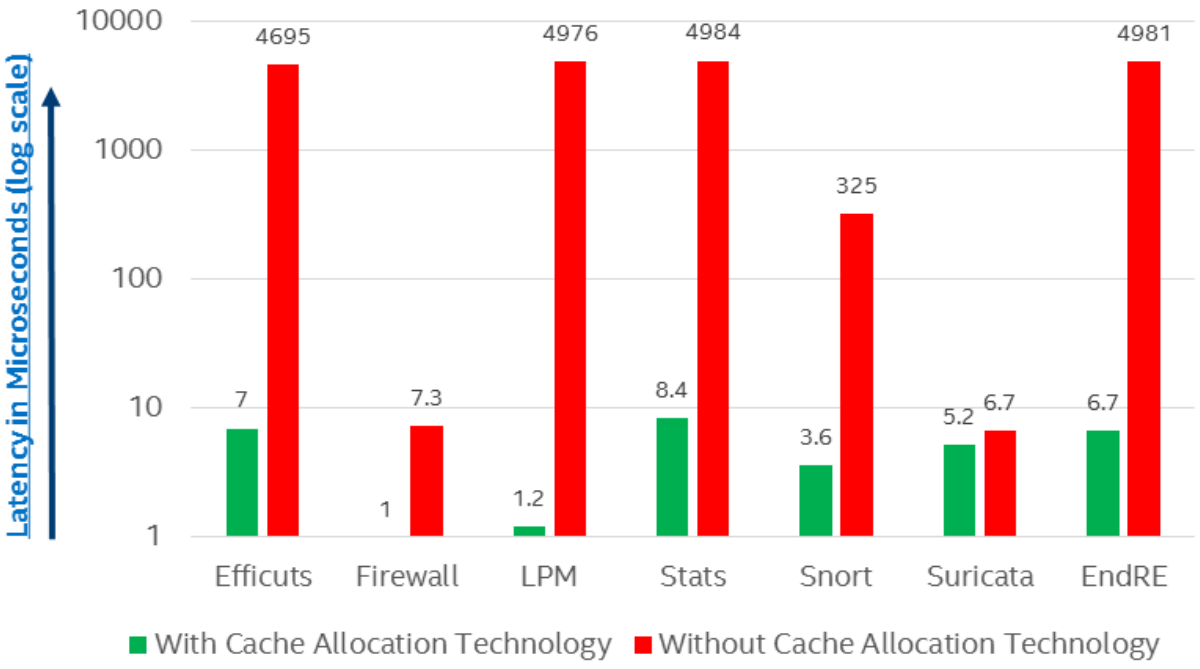






Baseline : NGINX web server, ext. load generation system, 2x Intel® Xeon® processor E5-2699 v4, 2.2GHz, 22c, 64GB DDR4-2133, 10Gb X540-AT2 NICs. Ubuntu14.04, Kernel v4.4 + RDT Patches. C1E / turbo disabled. CAT: Restrict "noisy neighbors" : CAT mask 0x00003. "Noisy neighbor" apps: 11 processes /skt of stream, array size 100e6. Ext Load generation system: wg/WRK running 22 thrds, Ubuntu* 14.04, 2x Intel Xeon processor L5520@ 2.27GHz CPUs, 24GB DDR3-1067 with 10Gb Intel® X540-AT2 NICs. Data Source: Appformix, March 2016

UC , Berkley CA RDT usage



- Network functions are executing simultaneously on isolated core's, throughput of each Virtual Machines is measured
- Min packet size (64 bytes), 100K flows, uniformly distributed

OSV adaption status

- Intel RDT support status for OSVs
  - ✓ CMT:
    - RHEL 7.2 (3.10): merged
    - Ubuntu 15.10 (4.2): merged
    - SLES12 SP2 Beta (4.4): finished backporting and test, will merge
    - Alibaba, Baidu: Backported and in Testbed
  - ✓ MBM:
    - RHEL 7.3 RC (3.10): finished backporting and test, will merge
    - Ubuntu 16.04 (4.4): merged
    - SLES12 SP3 Beta (4.4): will submit request
    - Alibaba, Baidu: Backported and in Testbed
  - ✓ CAT, CDP :
    - Currently all using out of tree patches. Waiting for upstream patches
    - Google : using currently in testbed
    - Alibaba, Baidu: Backported and in Testbed

Challenges

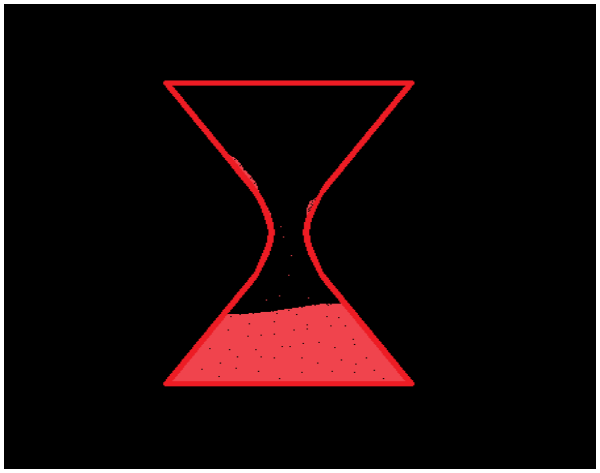
- Openstack, Container next steps
- What if we run out of IDs ?
- What about Scheduling overhead
- Doing monitoring and allocation together

Openstack/container next steps for CAT/CDP

- kernel CAT ***cgroup support*** **will remain out of tree**
  - cgroup Pros
    - openstack/dockers other enterprise users like Google could use the feature on test bed and are ready to adapt
    - Was supported by much of community (Peterz/HPA/dockers/google) for quite sometime.
    - Issues like hierarchy/kernel thread issue was related to cgroup.
  - Cons
    - Thomas rejected cgroup interface eventually.
    - Quickly run out of CLOSIds with *cgroup hierarchy*, more in v2 – However reuse had mitigated some of the issues.
    - *Could not do per socket Closid* due to atomic update issue
- Openstack and Dockers **CAT support needs a rewrite to use the new CAT (resctl) interface.**

What if we run out of IDs ?

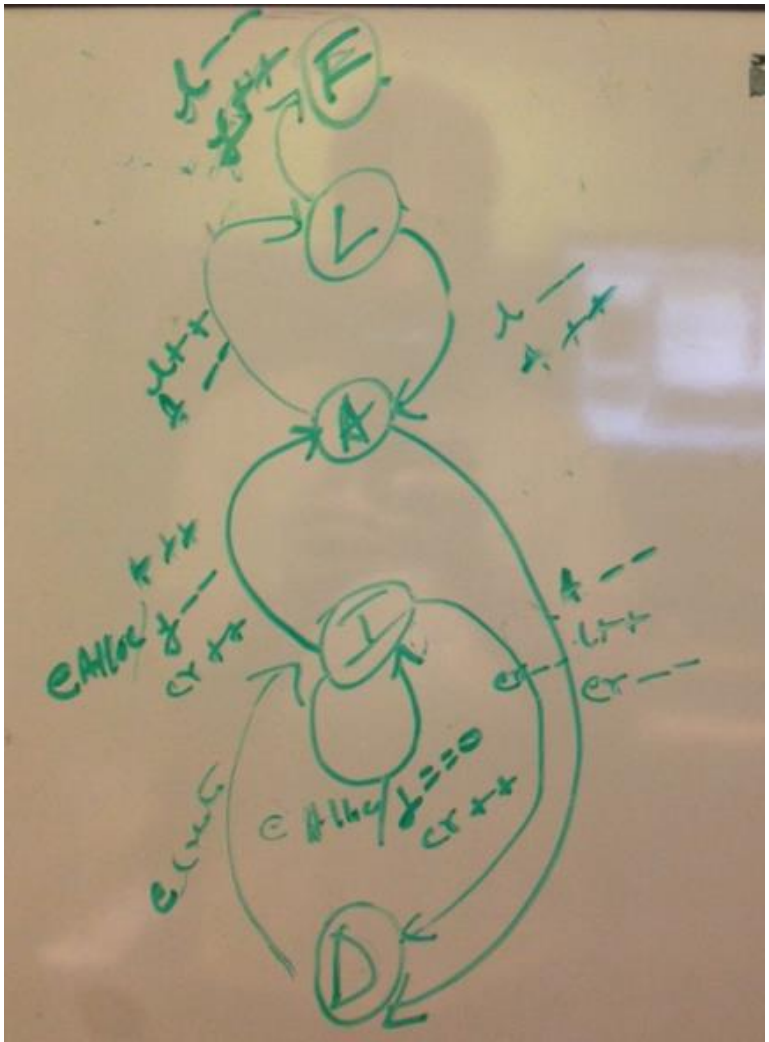
- Group tasks together (by process?)
- Group cgroups together with same mask
- return –ENOSPC
- Postpone/ Recycle



RMID recycling

- Not really ‘virtual RMIDs’ currently as we don’t switch RMIDs at context switch.
- For cqm, cache occupancy is still tied to the RMID after we ‘free’ an RMID -> it goes to limbo list.
- However for MBM , the RMIDs can be used immediately without waiting for zero occupancy.

RMID recycling



F – Free state (f- free count)
L – Limbo
A - Allocated
e – event (er- # of required RMIDs)

RMID recycling accuracy

- Current scheme eg:
- The counting time is proportional to the max RMID to required RMID ratio
- Ex: 80 RMIDs max , 100 required RMIDs
  - on average an event is counted for 80% of time and missed for 20% of the time

Scheduling performance

- msrread/write costs 250-300 cycles
- Keep a cache. Grouping helps !

Monitor and Allocate

- RMID(Monitoring) CLOSid(allocation) different
- Monitoring and allocate same set of tasks easily
  - perf cannot monitor the cache alloc cgroup/ now resctl(?)

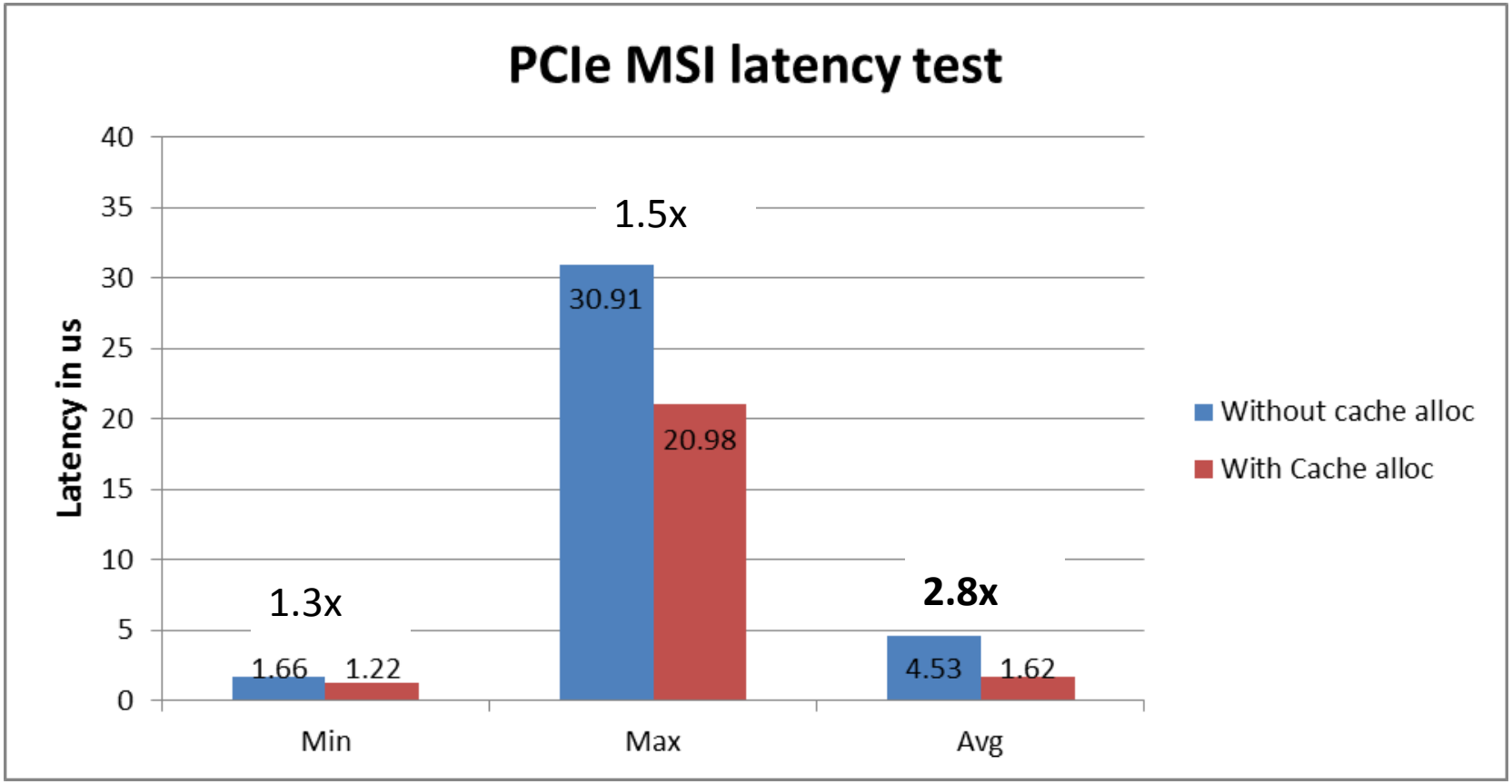
Agenda

- Problem definition
- Existing techniques
- Why use Kernel QOS framework
- Intel Cache qos support
- Kernel implementation
- Challenges
- **Performance improvement and Future Work**

Performance Measurement

- Intel Xeon based server, 16GB RAM
- 30MB L3 , 24 LPs
- RHEL 6.3
- With and without cache allocation comparison
- Controlled experiment
  - PCIe generating MSI interrupt and measure time for response
  - Also run memory traffic generating workloads (noisy neighbour)
- ***Experiment Not using current cache alloc patch***

Performance Measurement[1]



- Minimum latency : 1.3x improvement , Max latency : 1.5x improvement , Avg latency : 2.8x improvement
- **Better consistency** in response times and **less jitter and latency** with the noisy neighbour

Patch status

Cache Monitoring (CMT)	Upstream 4.1.
Cache Allocation(CAT)/CDP for L3	Framework (global clos/cbm management, hotcpu, hsw, sched support) good but ***Cgroup Interface rejected.*** (Vikas, Shivappa) New resctl interface and per-socket closid support in progress (Fenghua, Yu)
Memory b/w Monitoring	Upstream 4.6 (Vikas, Shivappa).
Open stack integration (libvirt update)	Support built for CMT/MBM and CAT cgroup interface (Qiaowei qiaowei.ren@intel.com)
Container support (Dockers)	Support built for CAT cgroup interface(Intel)

Future Work

- Perf overhead during CQM/MBM
- Support data per-process
- Improve and unify ID management for RMID/CLOSID

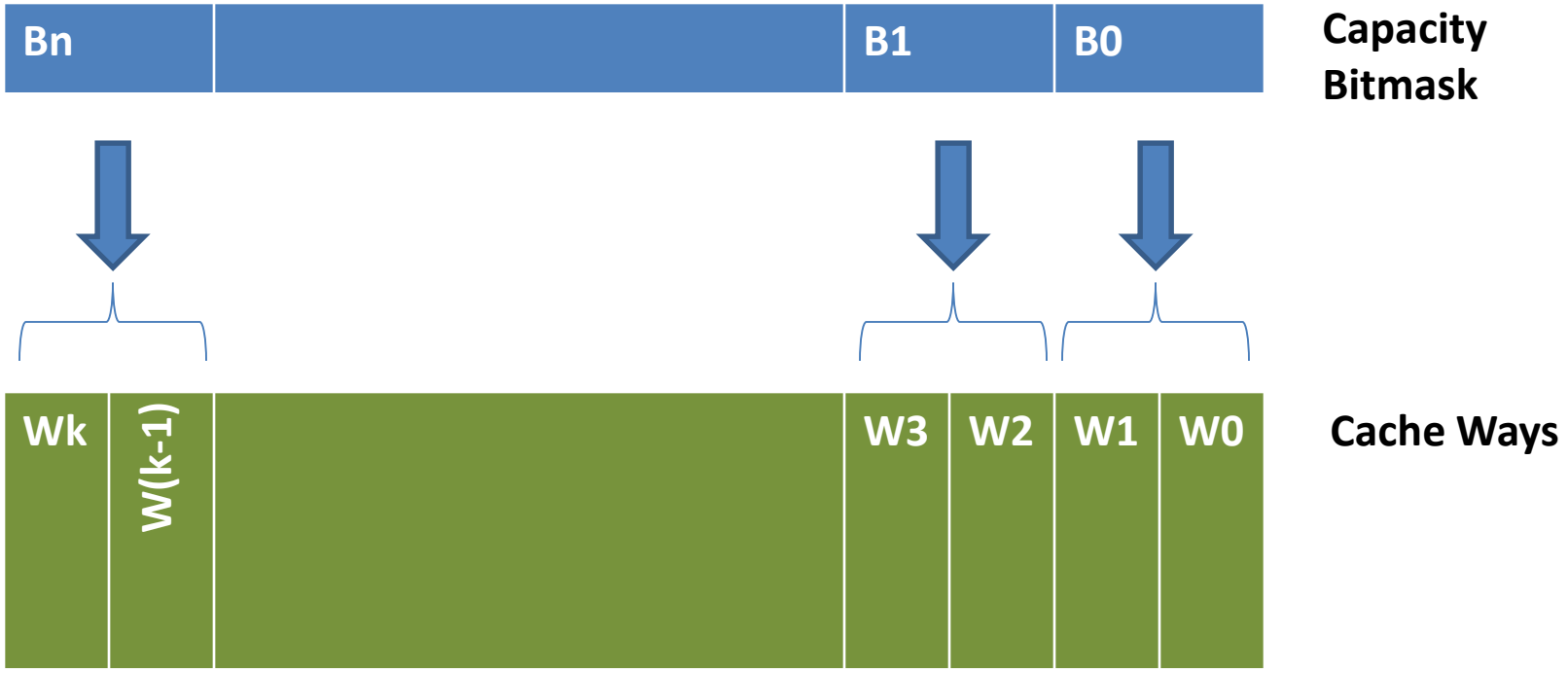
References

- [1] http://www.intel.com/content/www/us/en/communications/cache-allocation-technology-white-paper.html

Questions ?

Backup

Representing cache capacity in Cache Allocation(example)



- Cache capacity represented using **'Cache bitmask'**
- However mappings are hardware implementation specific

Bitmask ⇔ Class of service IDs (CLOS)

Default Bitmask – All CLOS ids have all cache

	B7	B6	B5	B4	B3	B2	B1	B0
CLOS0	A	A	A	A	A	A	A	A
CLOS1	A	A	A	A	A	A	A	A
CLOS2	A	A	A	A	A	A	A	A
CLOS3	A	A	A	A	A	A	A	A

Overlapping Bitmask (only contiguous bits)

	B7	B6	B5	B4	B3	B2	B1	B0
CLOS0	A	A	A	A	A	A	A	A
CLOS1					A	A	A	A
CLOS2							A	A
CLOS3					A	A		