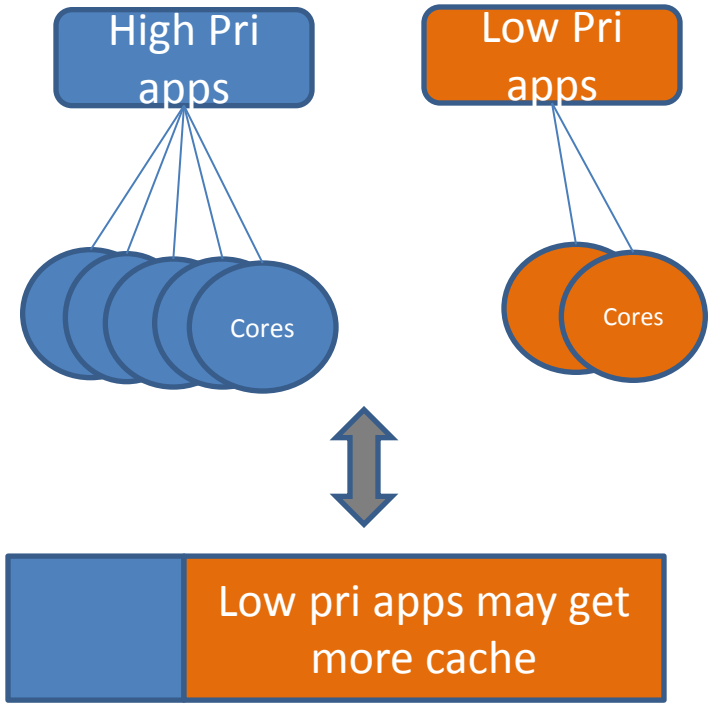
Introduction to Cache Quality of service in Linux Kernel

Vikas Shivappa
(vikas.shivappa@linux.intel.com)

Agenda

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

Without Cache QoS



Shared Processor Cache

- **Noisy neighbour** => Degrade/inconsistency in response => QOS difficulties
- Cache Contention with multi threading

Agenda

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

Existing techniques

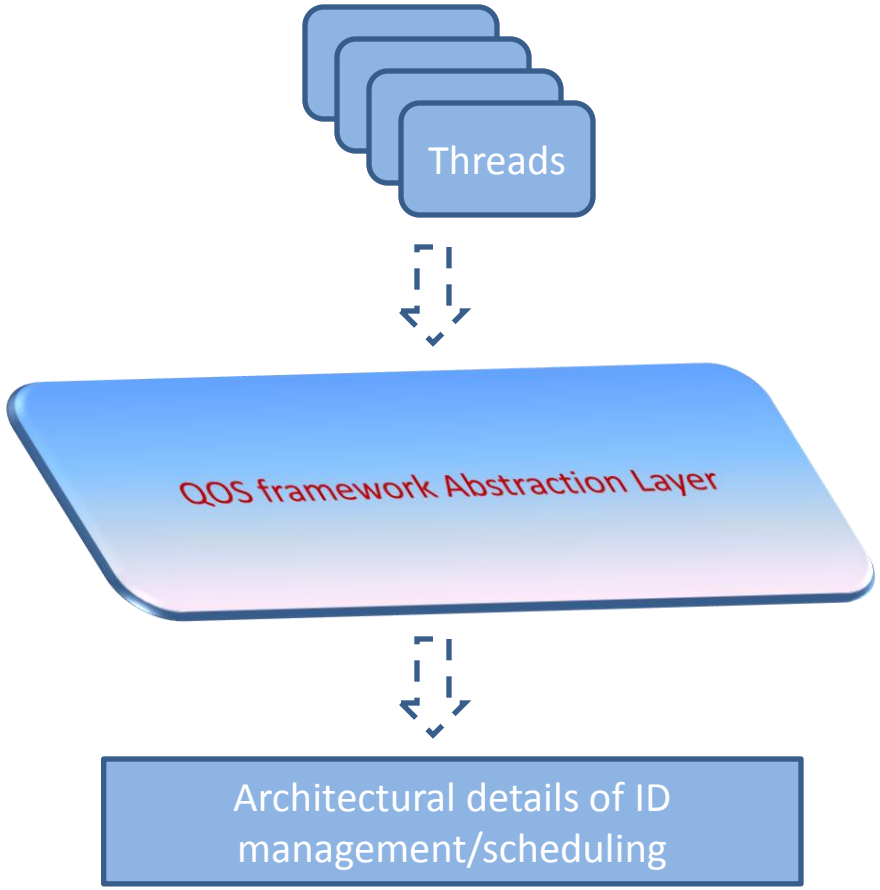
- Mostly heuristics on real systems
- **No methodology to identify cache lines** belonging to a particular thread
- Lacks configurability by OS

Agenda

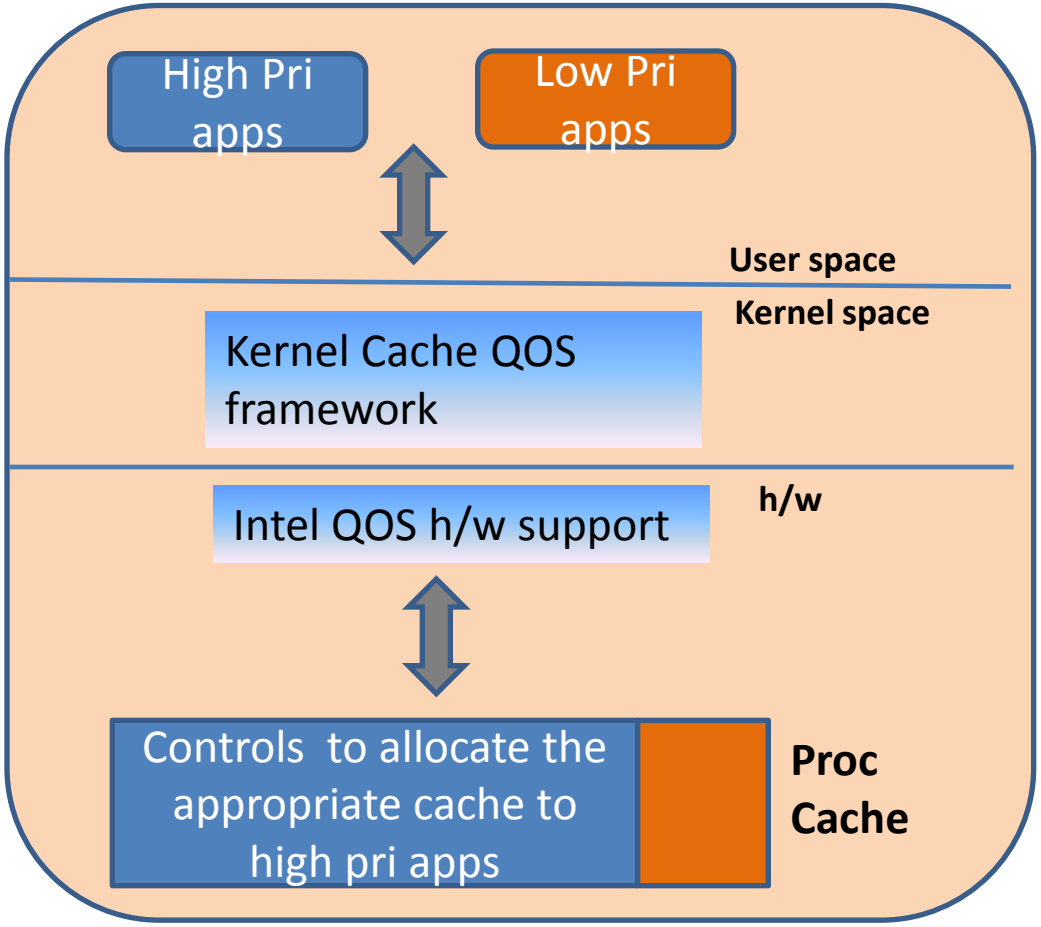
- Problem definition
- Existing techniques
- **Why use Kernel QOS framework**
- Intel Cache qos support
- Kernel implementation
- Challenges
- Performance improvement
- Future Work

Why use the QOS framework?

- Lightweight powerful tool to manage cache
- Without a lot of architectural details



With Cache QoS



- Help maximize performance and meet QoS requirements
  - **In Cloud or Server Clusters**
  - **Mitigate jitter/inconsistent response times due to 'Noisy neighbour'**

Agenda

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

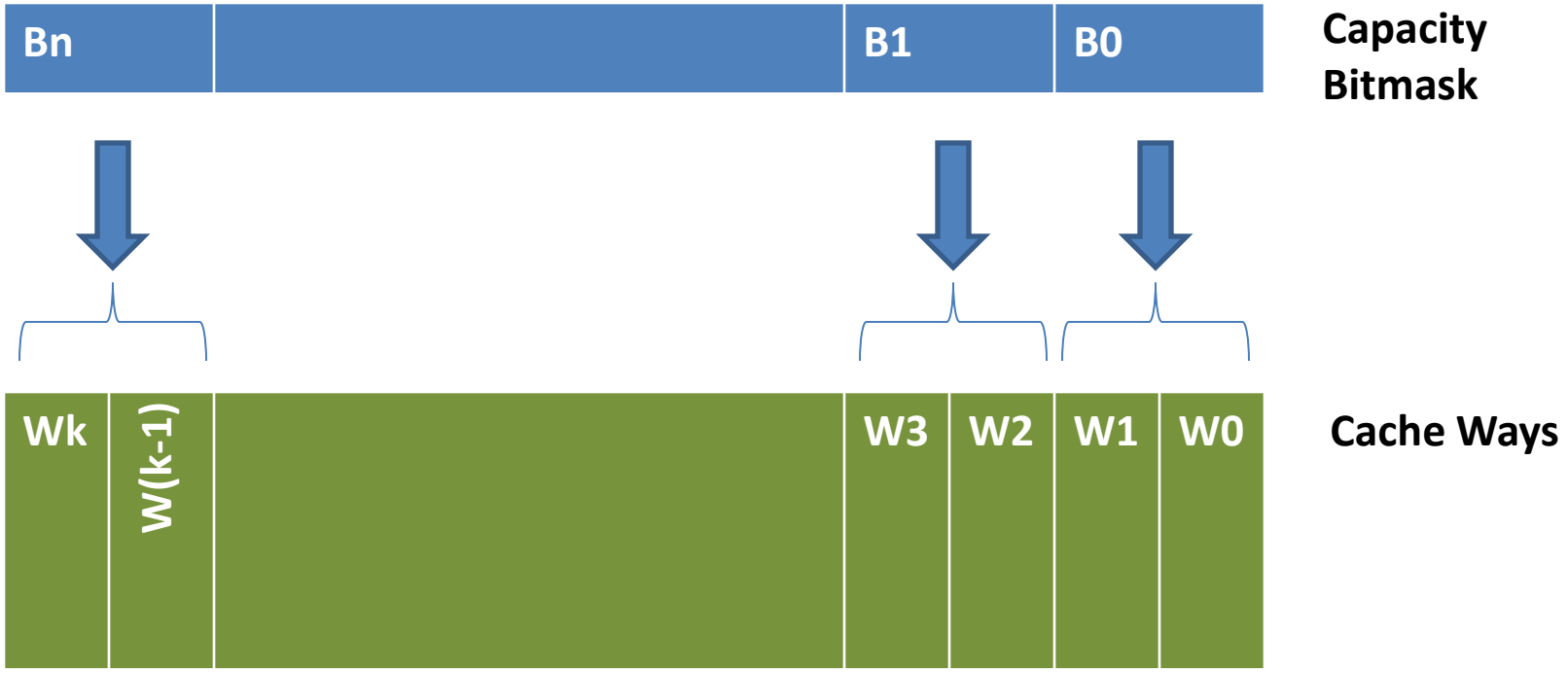
What is Cache QoS ?

- Cache Monitoring
  - cache occupancy per thread
  - **perf** interface
- Cache Allocation
  - user can allocate overlapping subsets of cache to applications
  - **cgroup** interface

Cache lines ⇔ Thread ID (Identification)

- Cache Monitoring
  - RMID (Resource Monitoring ID)
- Cache Allocation
  - CLOSid (Class of service ID)

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

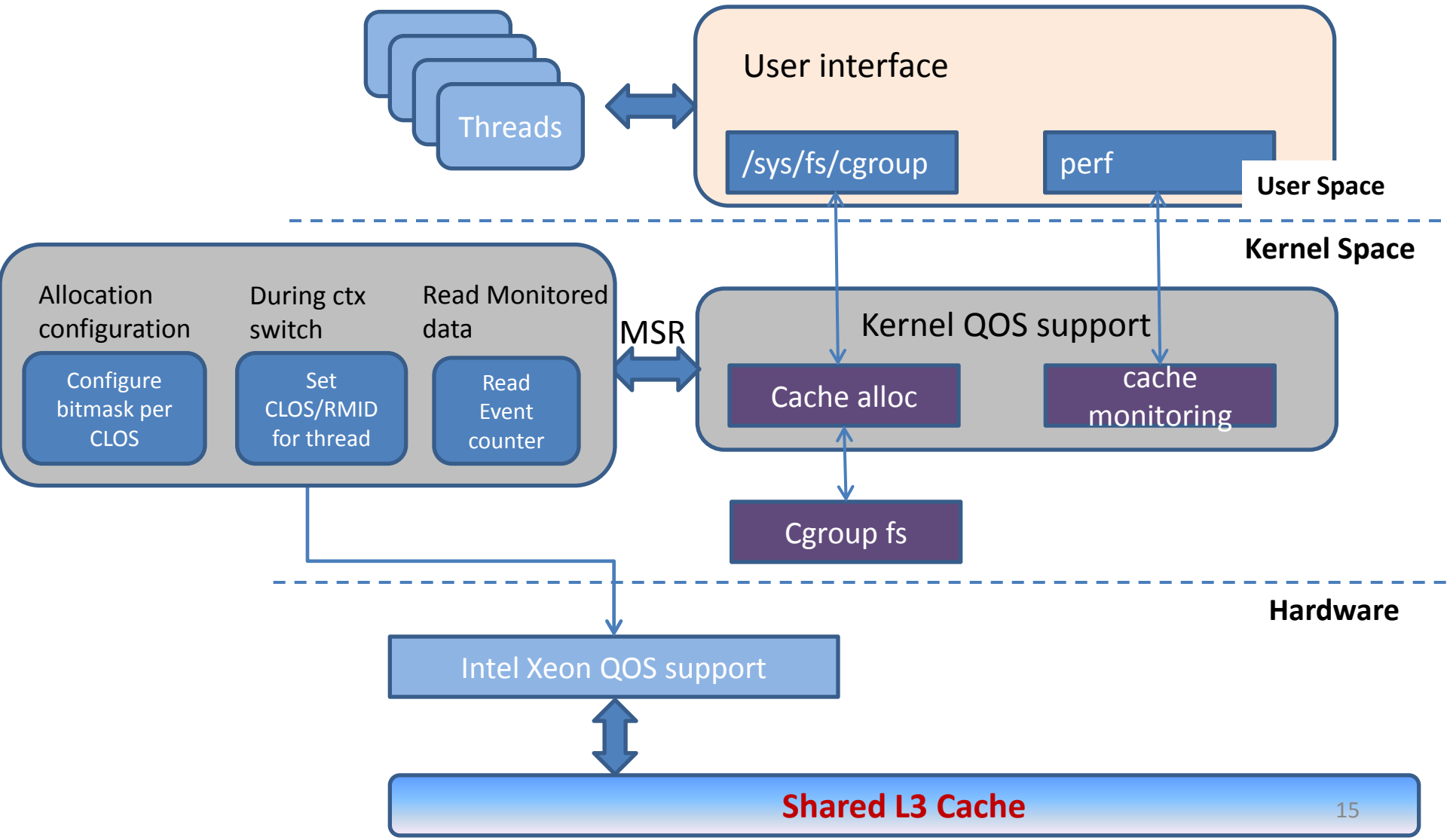
Overlapping Bitmask (only contiguous bits)

	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		

Agenda

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

Kernel Implementation



Usage

Monitoring per thread cache occupancy in bytes

```
./tools/perf/perf stat -e intel_cqm/llc_occupancy/ <cmd or tid>
```

```
Performance counter stats for thread id '5236':

        638976.00 Bytes intel_cqm/llc_occupancy/

     16.140199267 seconds time elapsed
```

Allocating Cache per thread through cache bitmask

```
/bin/echo 4938 > /sys/fs/cgroup/rdt/group1/tasks
```

Cgroup
Clos : Parent.Clos
Exposed to user land
bitmask : Parent.bitmask
Tasks : Empty

```
/bin/echo 0xf > /sys/fs/cgroup/rdt/group1/intel_rdt.cache_mask
```

Scenarios

- Units that can be allocated cache
  - Process/tasks
  - Virtual machines (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

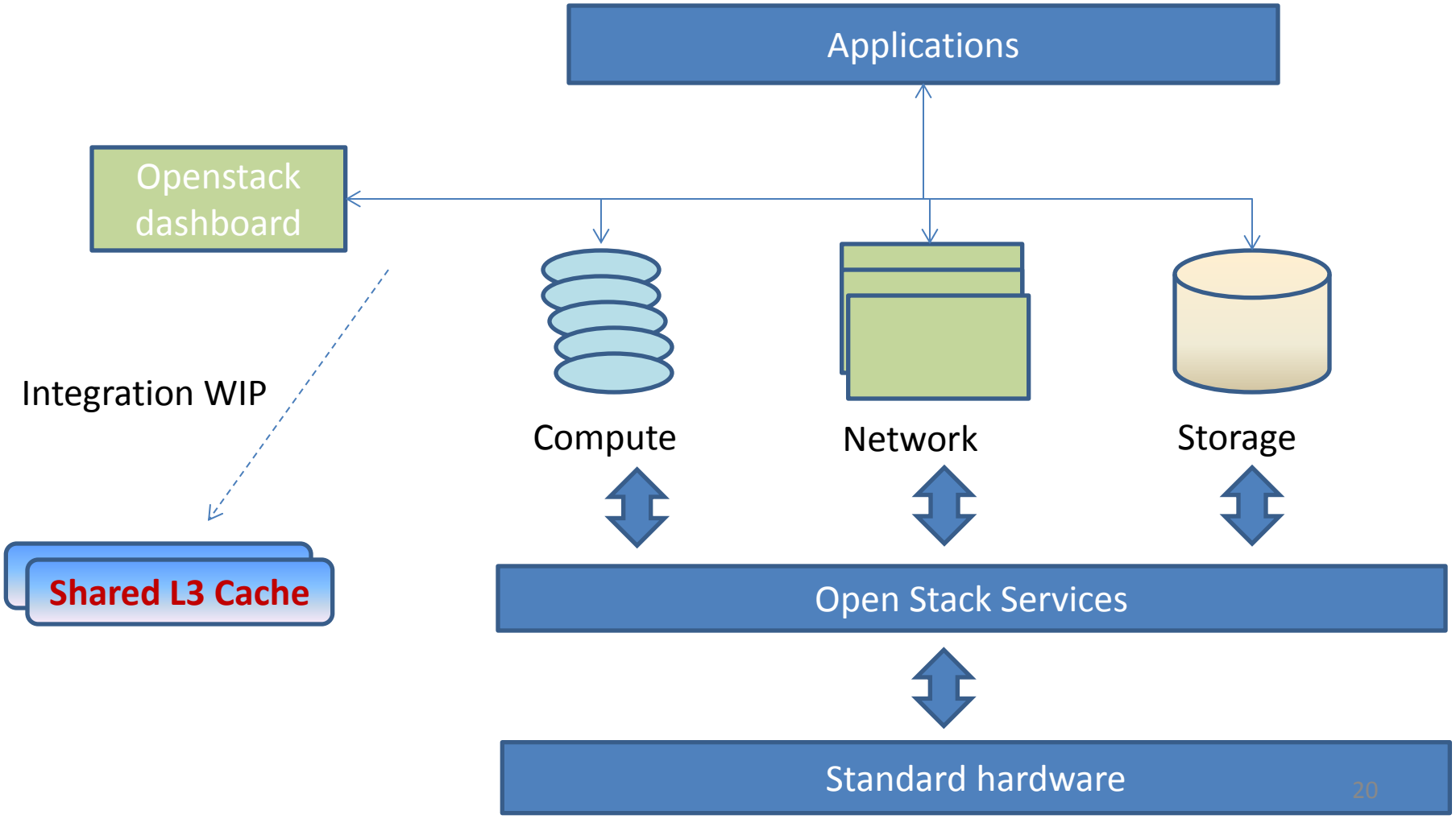
Agenda

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

Challenges

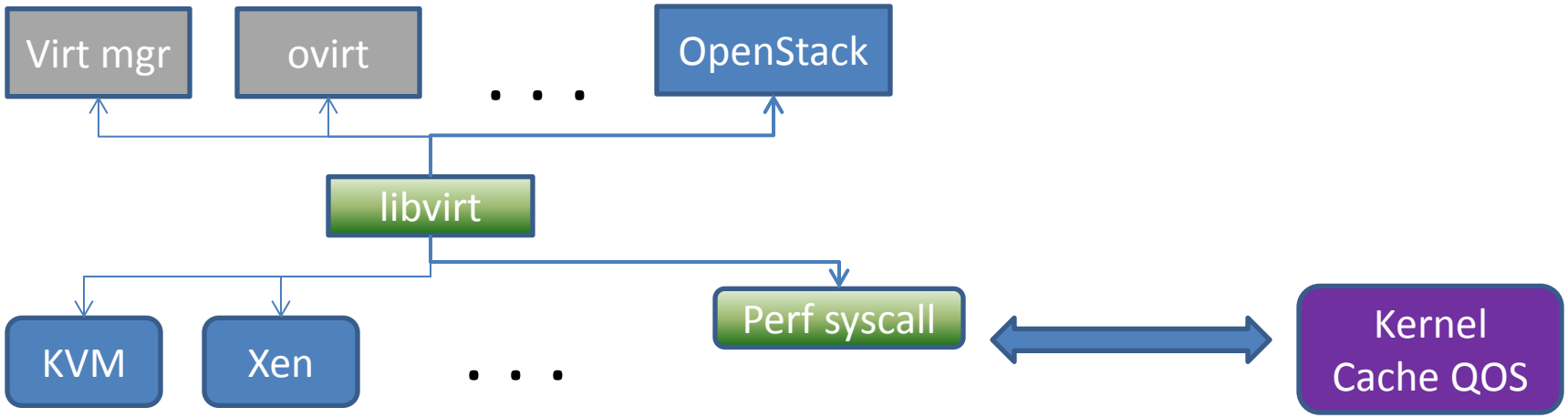
- Openstack usage
- What if we run out of IDs ?
- What about Scheduling overhead
- Doing monitoring and allocation together

Openstack usage



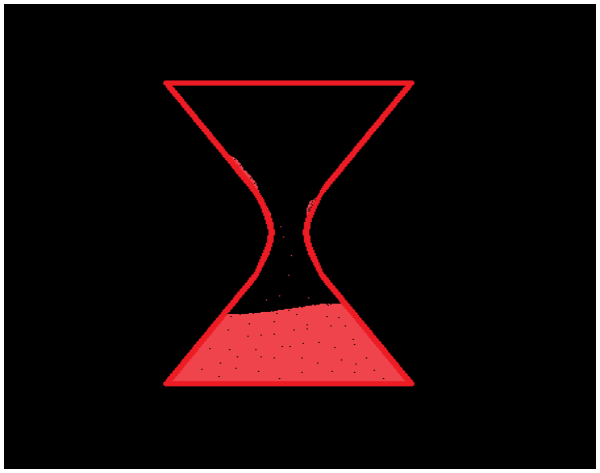
Openstack usage …

Work beginning, not stable yet to add changes to Ceilometer (With Qiaowei qiaowei.ren@intel.com)



What if we run out of IDs ?

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



Scheduling performance

- msrread/write costs 250-300 cycles
- Keep a cache. Grouping helps !
- Don’t use till user actually creates a new cache mask

Monitor and Allocate

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

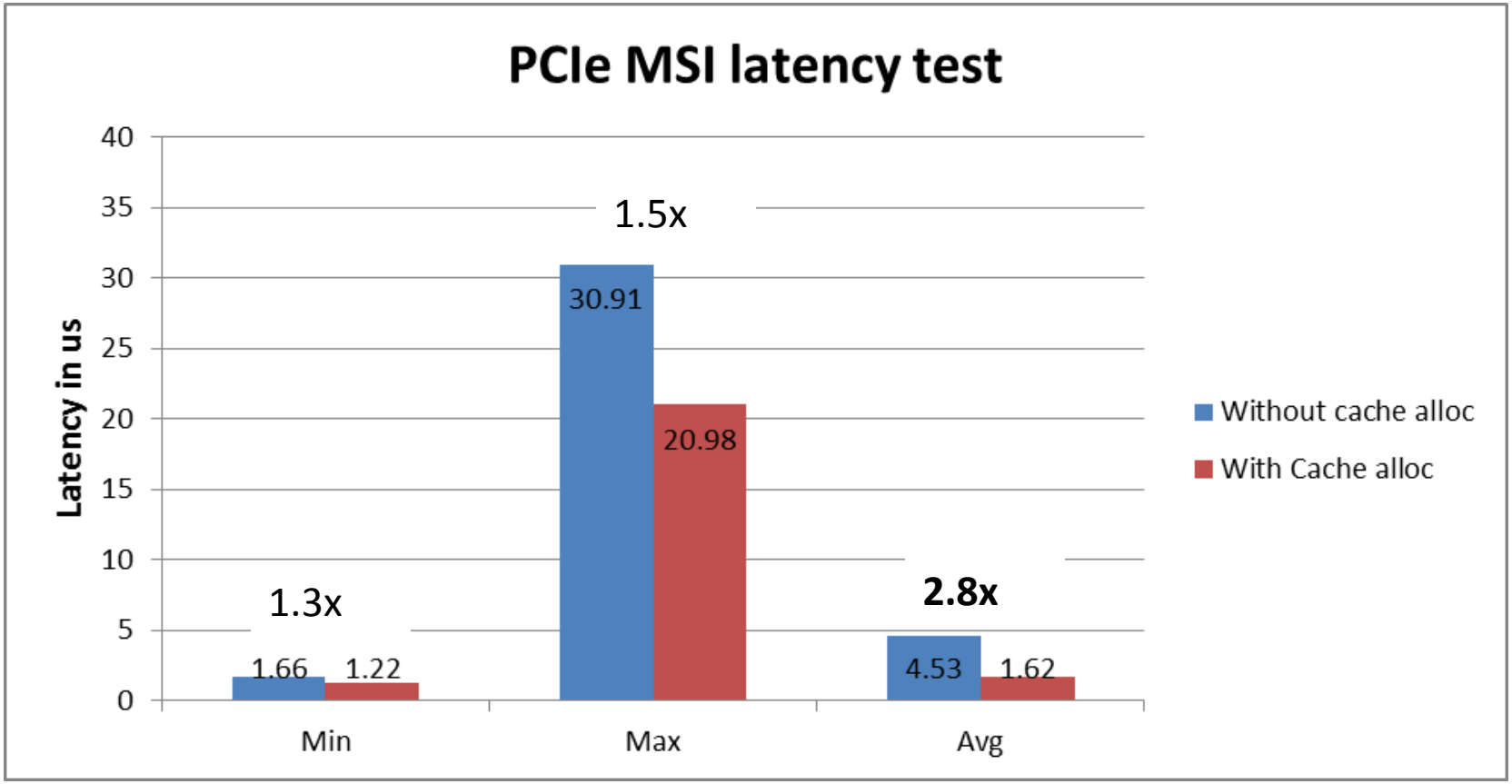
Agenda

- 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	Upstream 4.1 (Matt Fleming ,matt.fleming@intel.com)
Cache Allocation	Under review. (Vikas Shivappa , vikas.shivappa@intel.com)
Code Data prioritization	Under review. (Vikas Shivappa , vikas.shivappa@intel.com)
Open stack integration (libvirt update)	Work started (Qiaowei qiaowei.ren@intel.com)

Future Work

- Performance improvement measurement
- Code and data allocation separately
  - First patches shared on lkml
- Monitor and allocate same unit
- Openstack integration
- Container usage

Acknowledgements

- Matt Fleming (cache monitoring support, Intel SSG)
- Will Auld (Architect and Principal engineer, Intel SSG)
- CSIG, Intel

References

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

Questions ?