Low latency & Mechanical Sympathy: Issues and solutions

Jean-Philippe BEMPEL
Performance Architect

@jpbempel
http://jpbempel.blogspot.com

Ullink
CONNECT. TRADE.

Low latency order router

- pure Java SE application
- FIX protocol / Direct Market Access connectivity (TCP)
- transform to an intermediate form (UL Message)

Performance target

process & route orders in

< 100us

Low latency

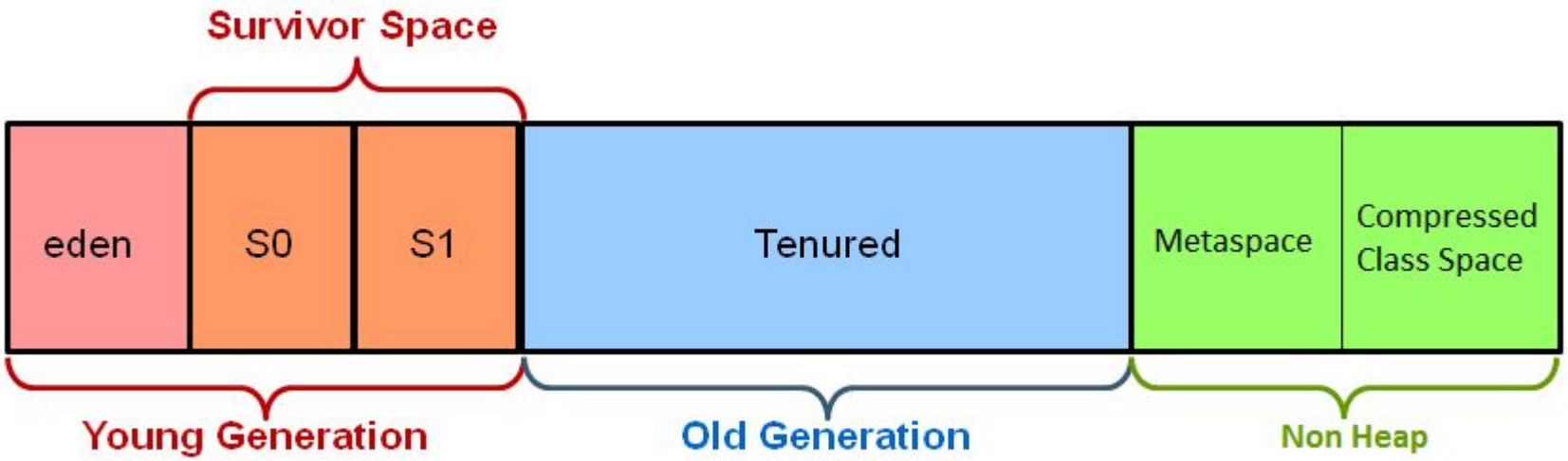
- Process under 1ms
- No (Disk) I/O involved
- Memory is the new disk
- Hardware architecture has a major impact on performance
- Mechanical Sympathy

Agenda

- GC pauses
- Power Management
- NUMA
- Cache pollution

GC pauses

Generations



GC pauses

- Minor GC & Full GC are Stop the World (STW)
  => All app threads are stopped
- Minor GC pauses: 30-100ms
- Full GC pauses: couple of seconds to minutes

Full GC

- None during trading hours
- Outside of trading hours, maintenance time allowed
- Java heap sized accordingly to avoid Full GC

Minor GC

Some tricks to reduce both

- frequency
- pause time

Minor GC frequency

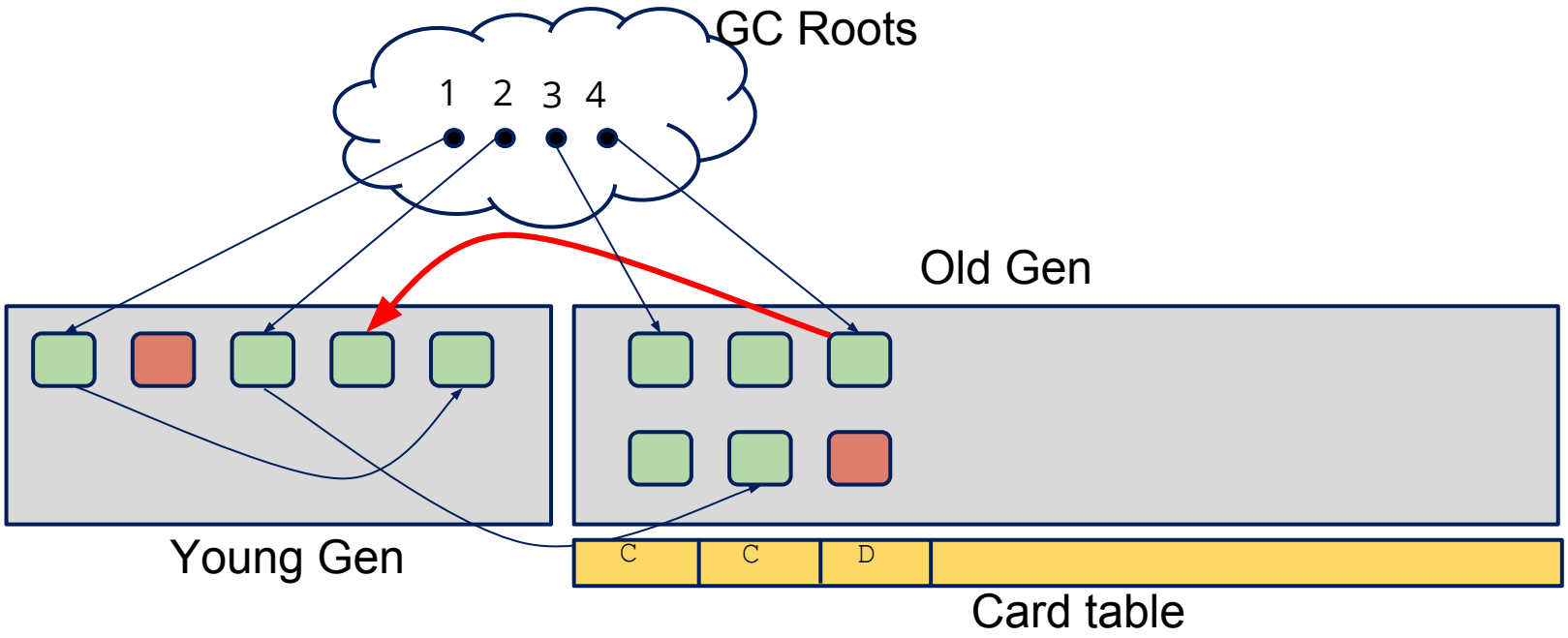
- Increase Young Gen (512MB)
- But increasing YG can increase pause time
- Increase likelihood of objects die

Minor GC pause time

pause time factors:

- Refs traversing
- Card scanning
- Object Copy (survivor or tenured)

Card Scanning



Object Copy

- Need to keep orders during app lifetime
- Orders are relatively large objects
- Orders will be promoted to old generation
  => copy to survivor spaces and then to old gen
- Increase significantly the pause time for minor GC

Object Copy

- Access pattern: 90% of the time accessed immediately after creation
- Weak references: avoid copying orders object
- persistence cache to allow reloading orders without too much impact

Allocation

- Allocation should be controlled carefully
- Reduce frequency of minor gc occurrence
- JDK API is not always friendly with GC/allocation

Allocation

need to rewrite some part:

conversion int -> char[]

```
int intToString(int i, char[] buffer, int offset)
```

avoid intermediate/temporary char[]

ullink

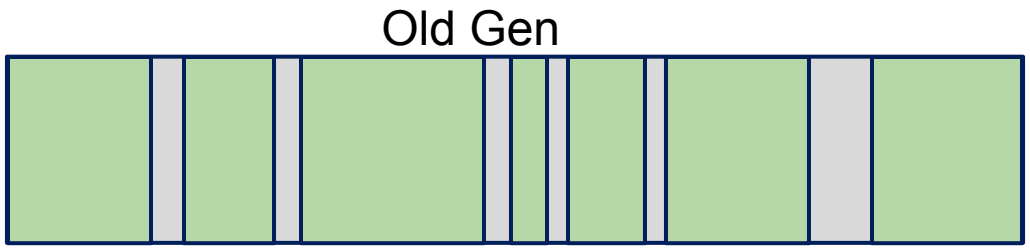
Usage of empty collections & pre-sizing:

```
Collection<String> process() {
   Collection<String> results = Collections.emptyList();
   for (String s : list) {
      if (results == Collections.<String>emptyList())
          results = new ArrayList(list.size());
       results.add(s);
   }
   return results;
}
```

CMS

CMS

- Promotion allocation use Free Lists (like malloc)
- No Compaction of Old Gen



- Increase minor pause time

CMS

Full GC fallback unpredictable (fragmentation, promotion failure, …)

```
176710.366: [GC 176710.366: [ParNew (promotion failed):
471872K->455002K(471872K), 2.4424250 secs]176712.809: [CMS:
6998032K->5328833K(7864320K), 53.8558640 secs] 7457624K->5328833K(8336192K),
[CMS Perm : 199377K->198212K(524288K)], 56.2987730 secs] [Times: user=56.60
sys=0.78, real=56.29 secs]
```

Same problem for G1

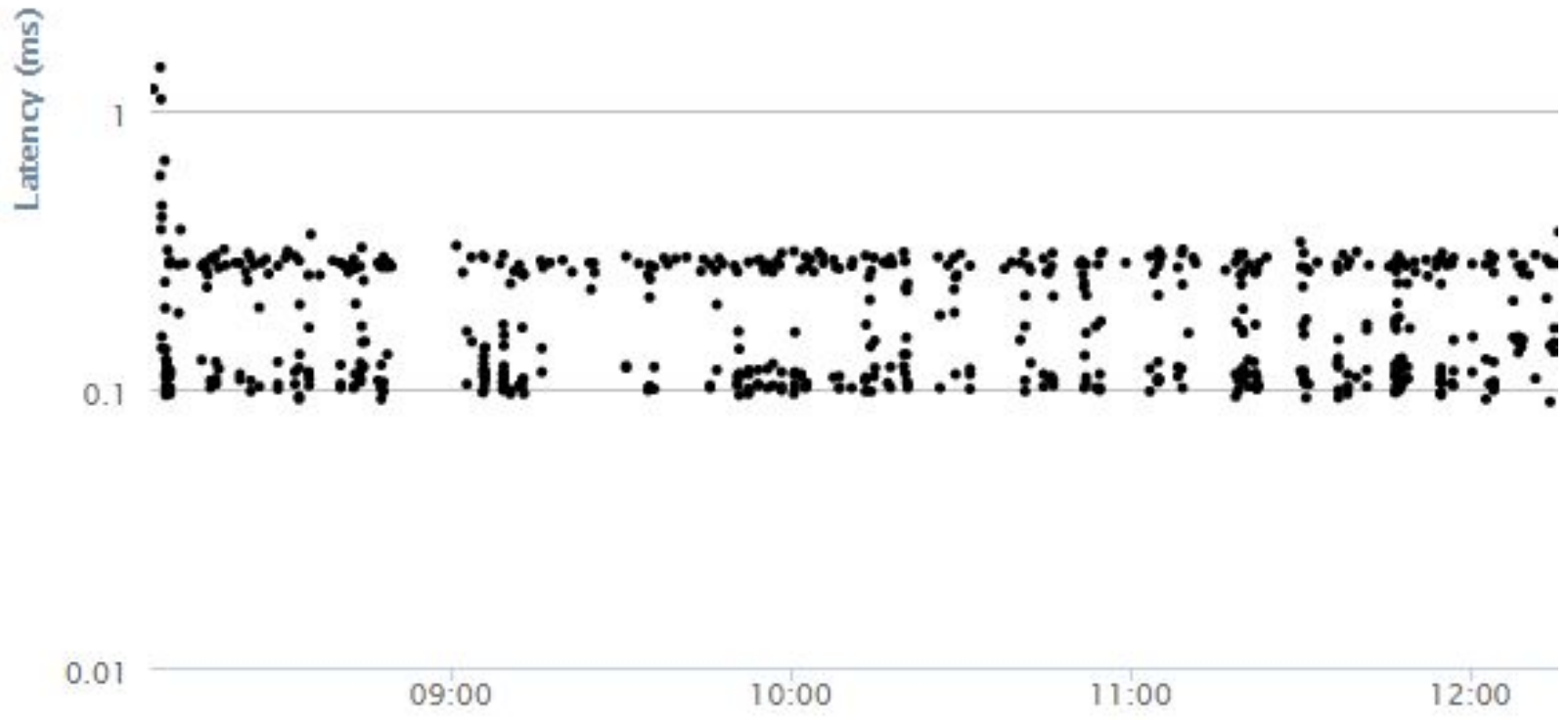
Zing JVM from Azul?

- We are partners
- Best GC algorithm in the world
- Our tests show maximum pause time of 2ms
- Some overhead compared to HotSpot (8%)
- False issue syndrom

Power Management

Why power management matters ?

- few orders per day, most of the time in idle

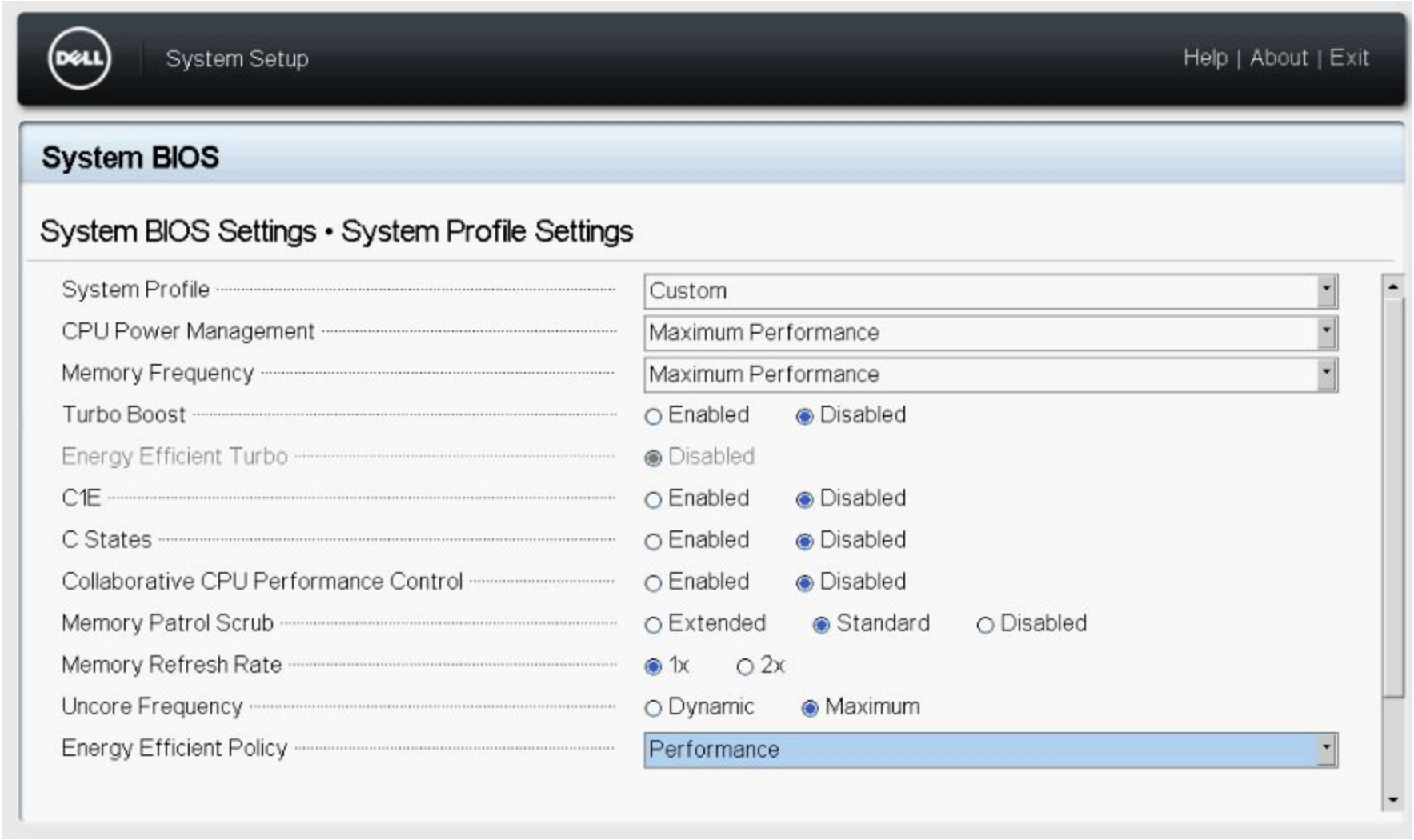


Power management technologies

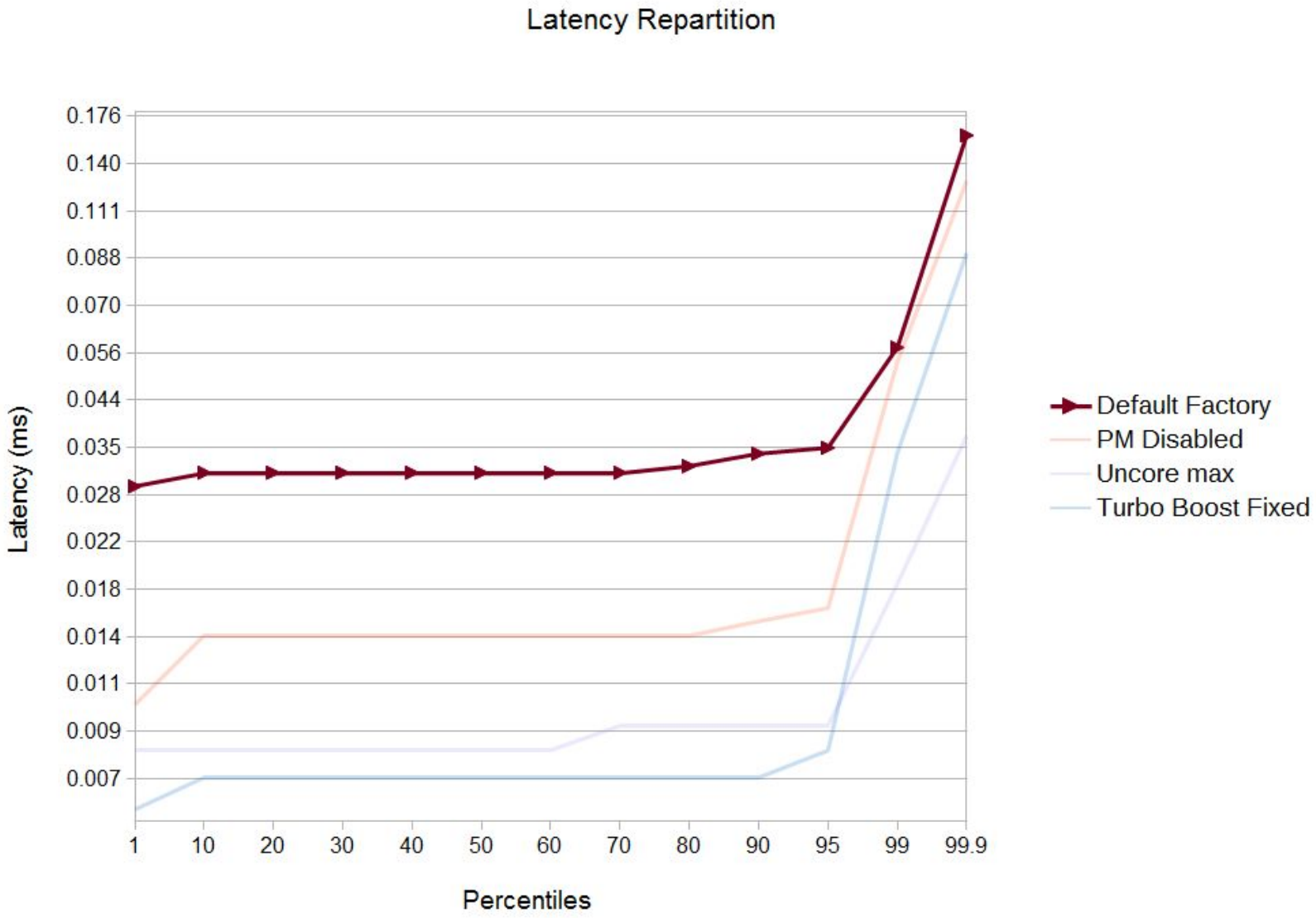
- CPU embeds technologies to reduce power usage
- Frequency scaling during execution (P states)
  P0 = nominal frequency
- Deep sleep modes during idle phases (C states)
  C0 = Running, C1 = idle
- latency to wakeup

```
cat /sys/devices/system/cpu/cpu0/cpuidle/state3/latency
200 (us)
```

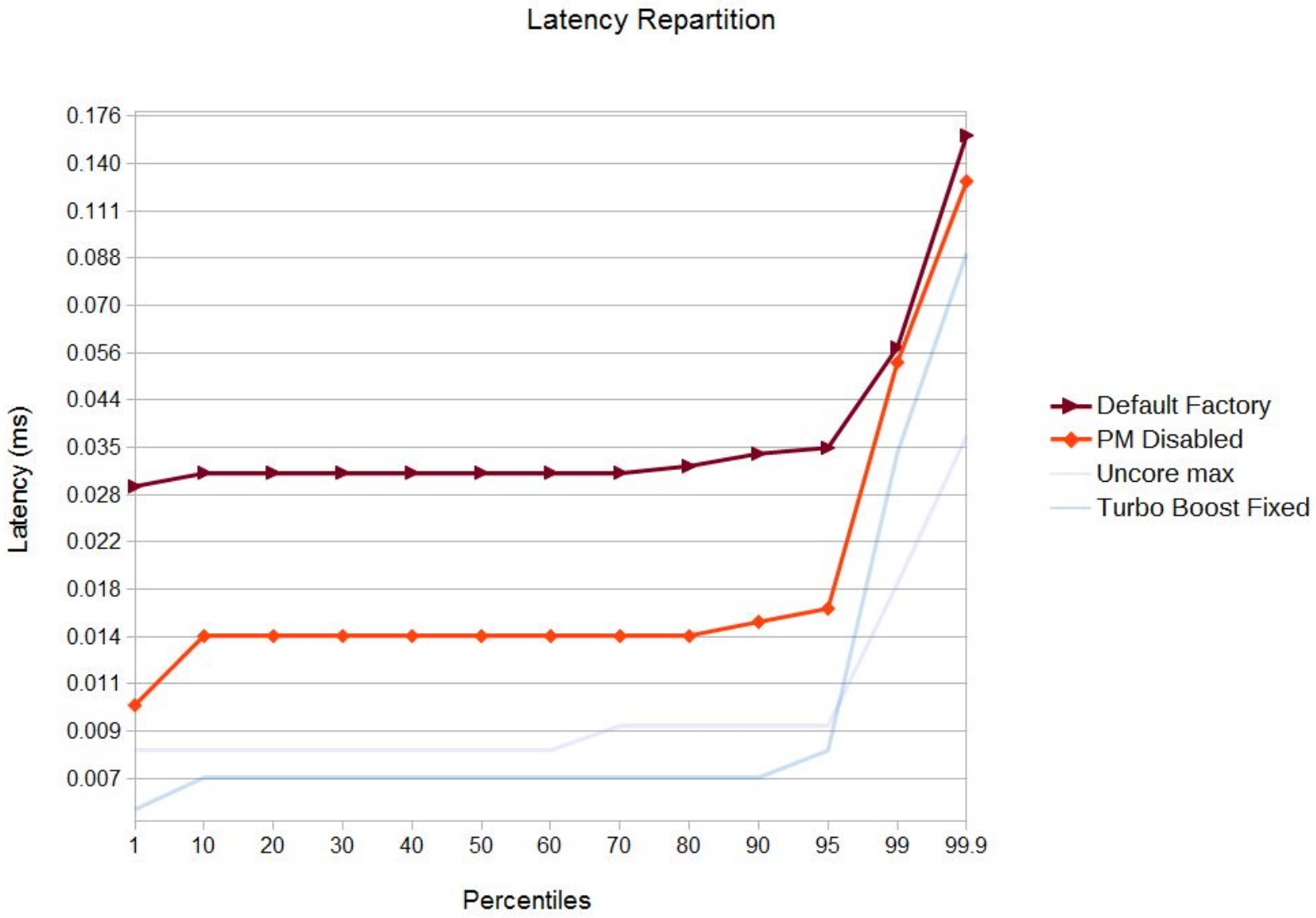
BIOS Settings



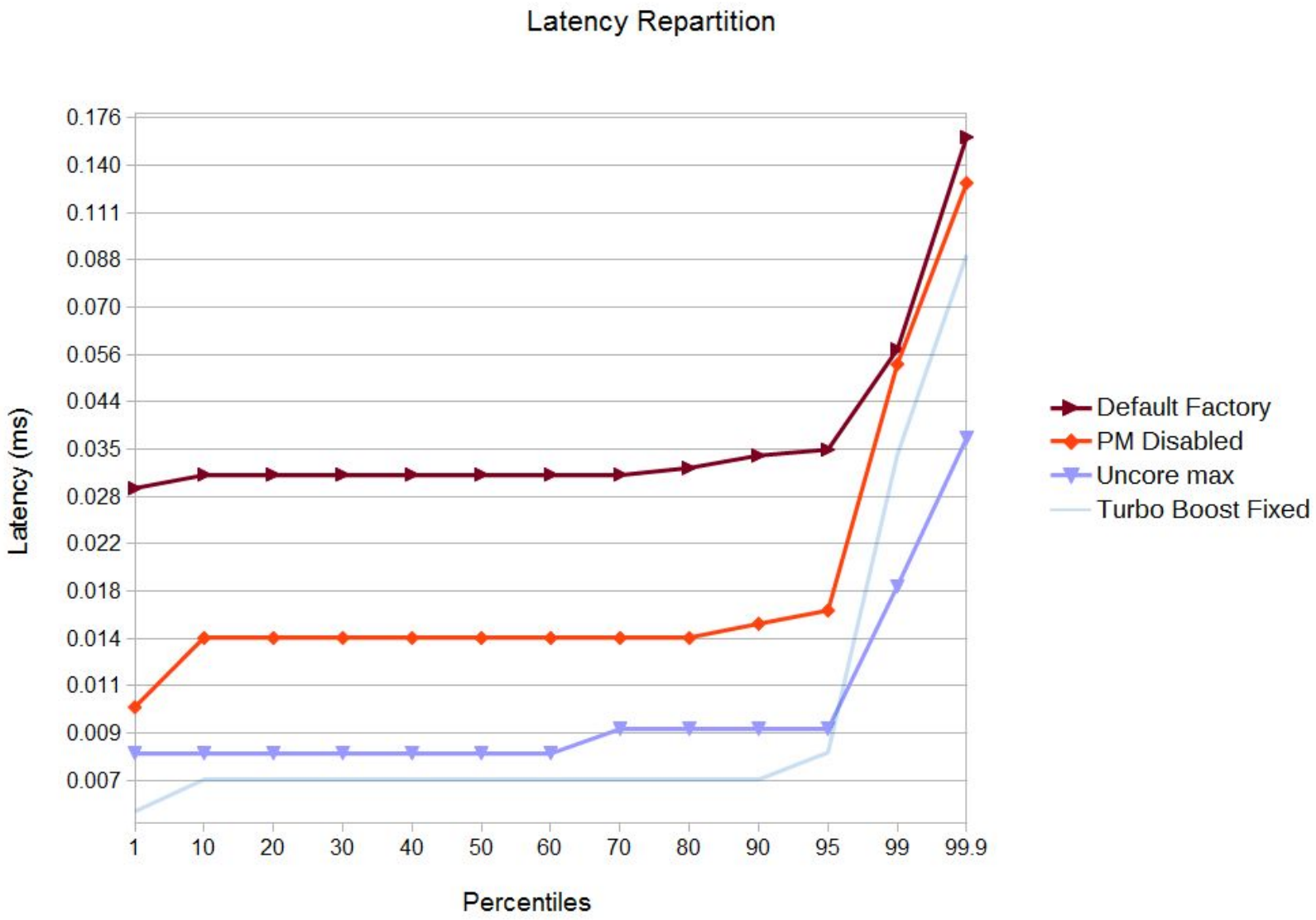
BIOS settings impact



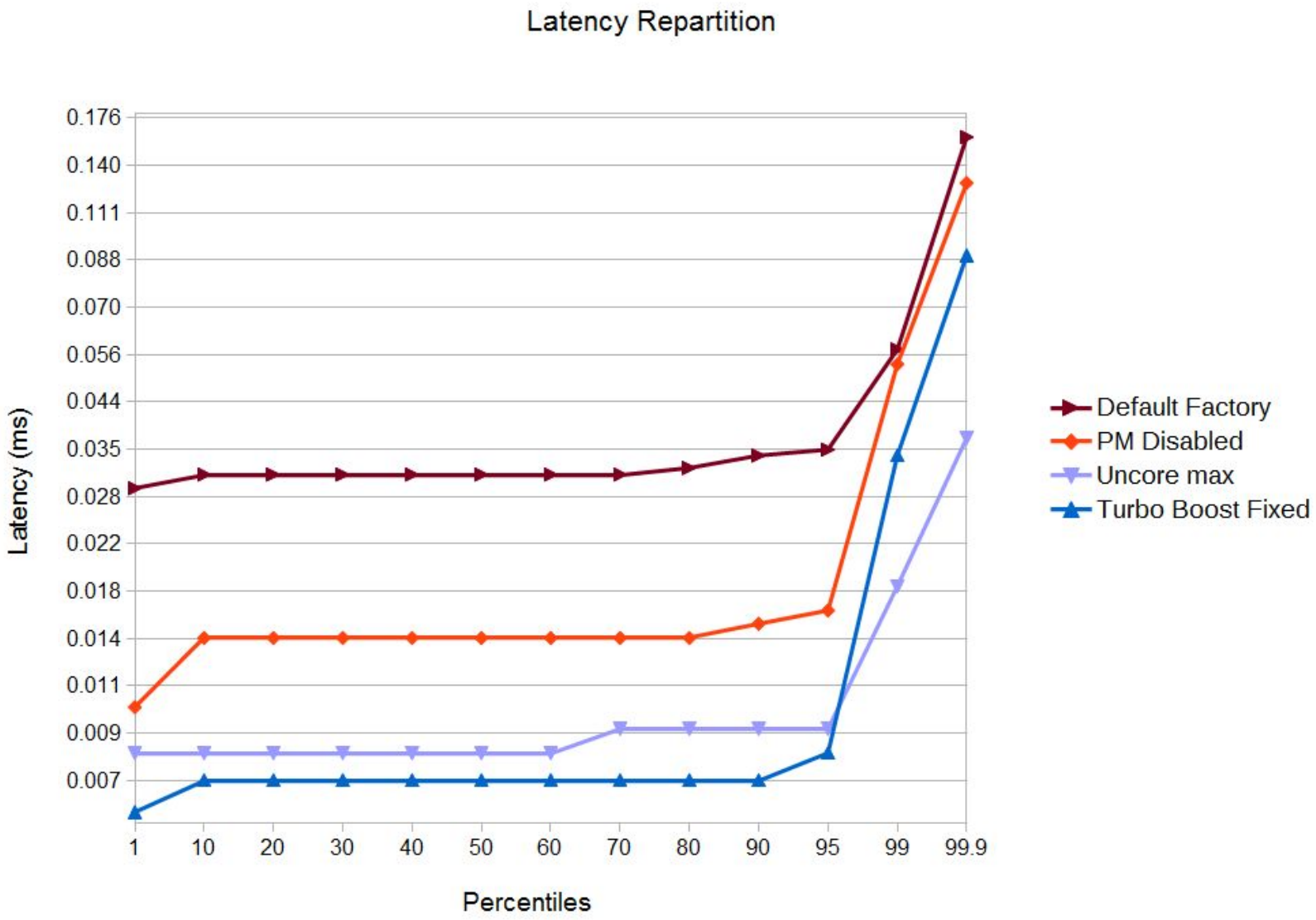
BIOS settings impact



BIOS settings impact



BIOS settings impact



Cores reduction/Fixed Turbo Boost

- Turbo boost is dynamic frequency scaling
- Depends on thermal envelop
- Reducing cores reduce thermal envelop => higher freq
- Some vendors can fix this higher frequency
- Example:
  intel E5 v3 14 cores => 2 cores, 2.6Ghz => 3.6GHz

Power management at OS level

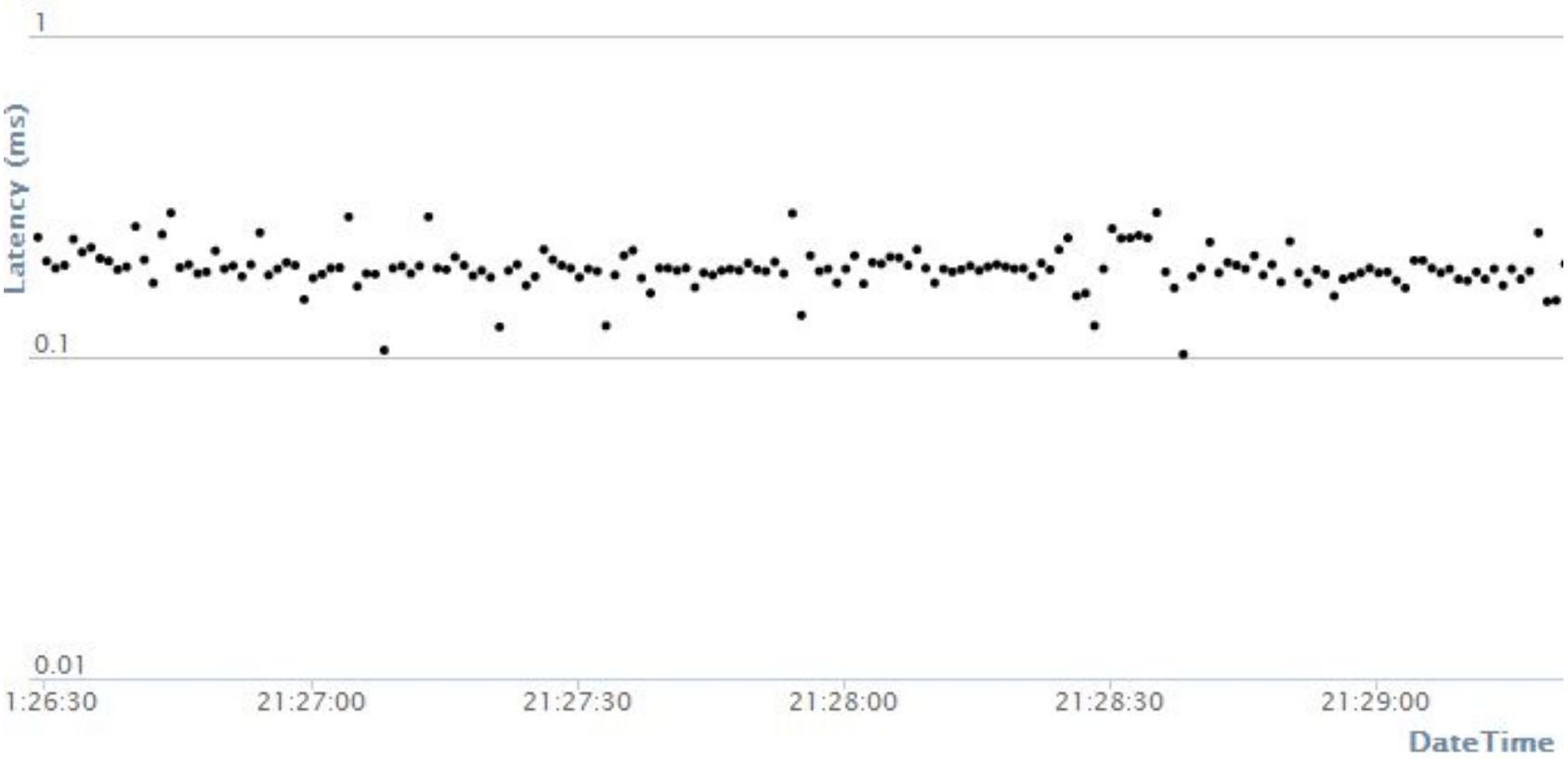
- Some OS drivers are also very aggressive

```
# cat /sys/devices/system/cpu/cpuidle/current_driver
intel_idle
```

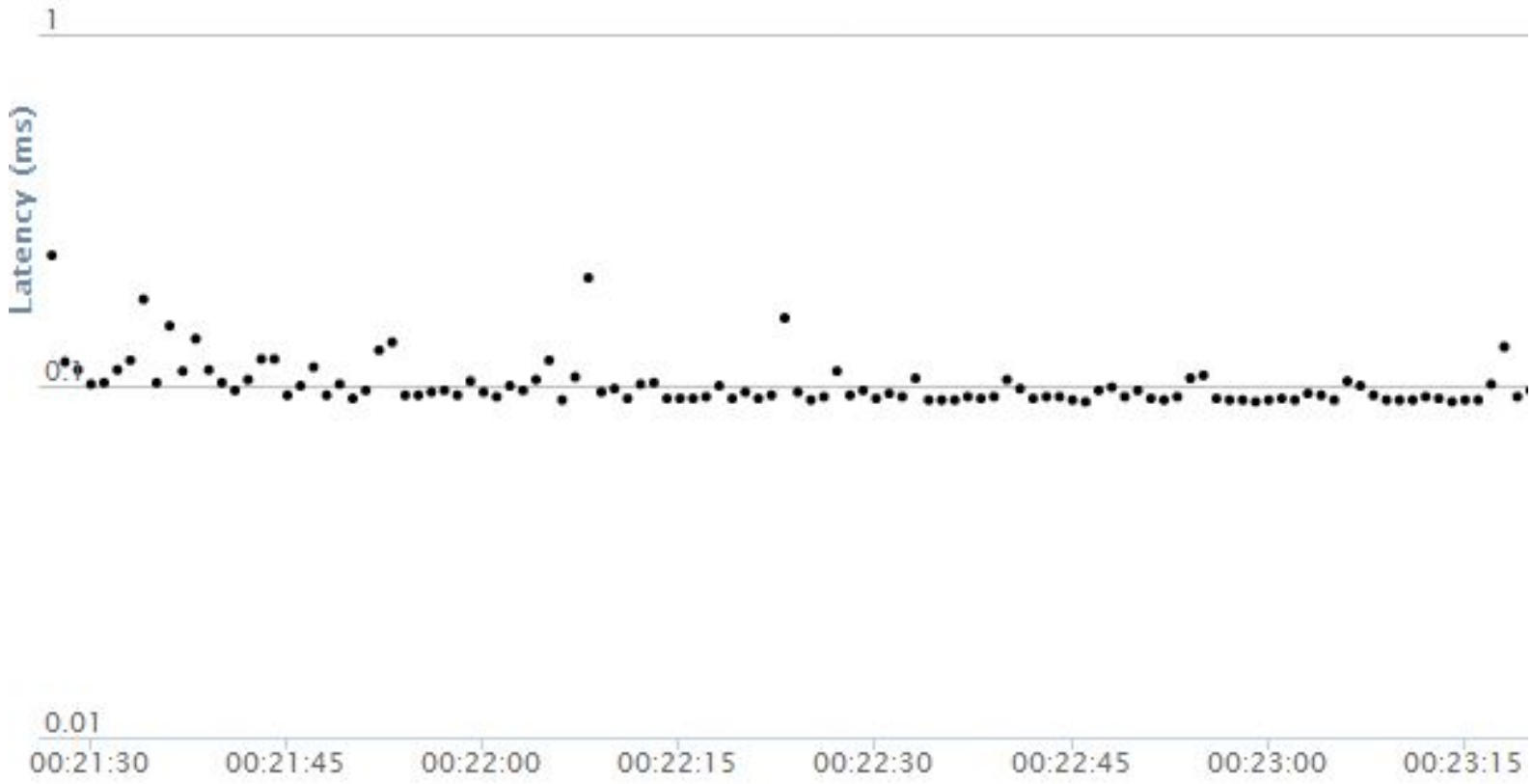
- Disabled in boot kernel parameters:

```
intel_idle.max_cstate=0
```

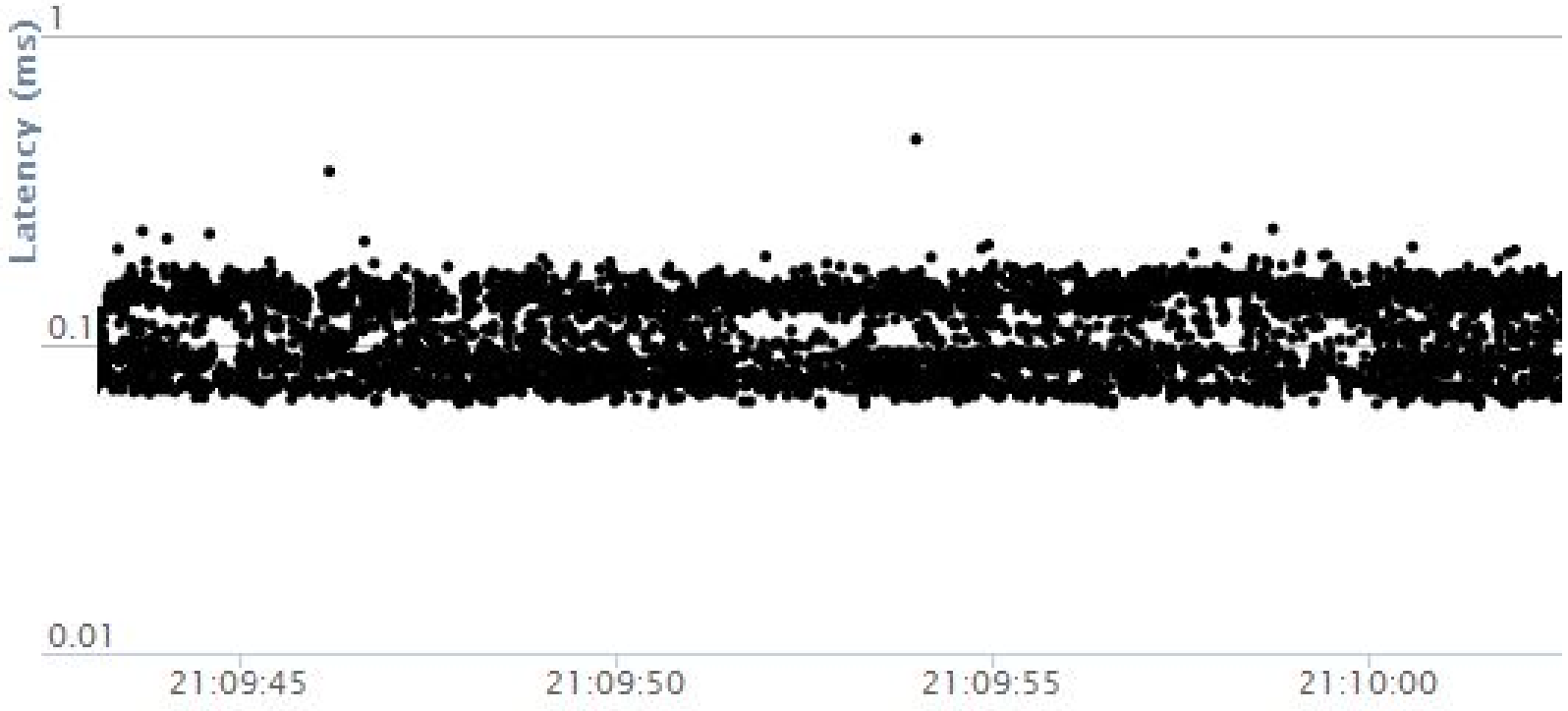
With intel_idle driver, 1 msg/s



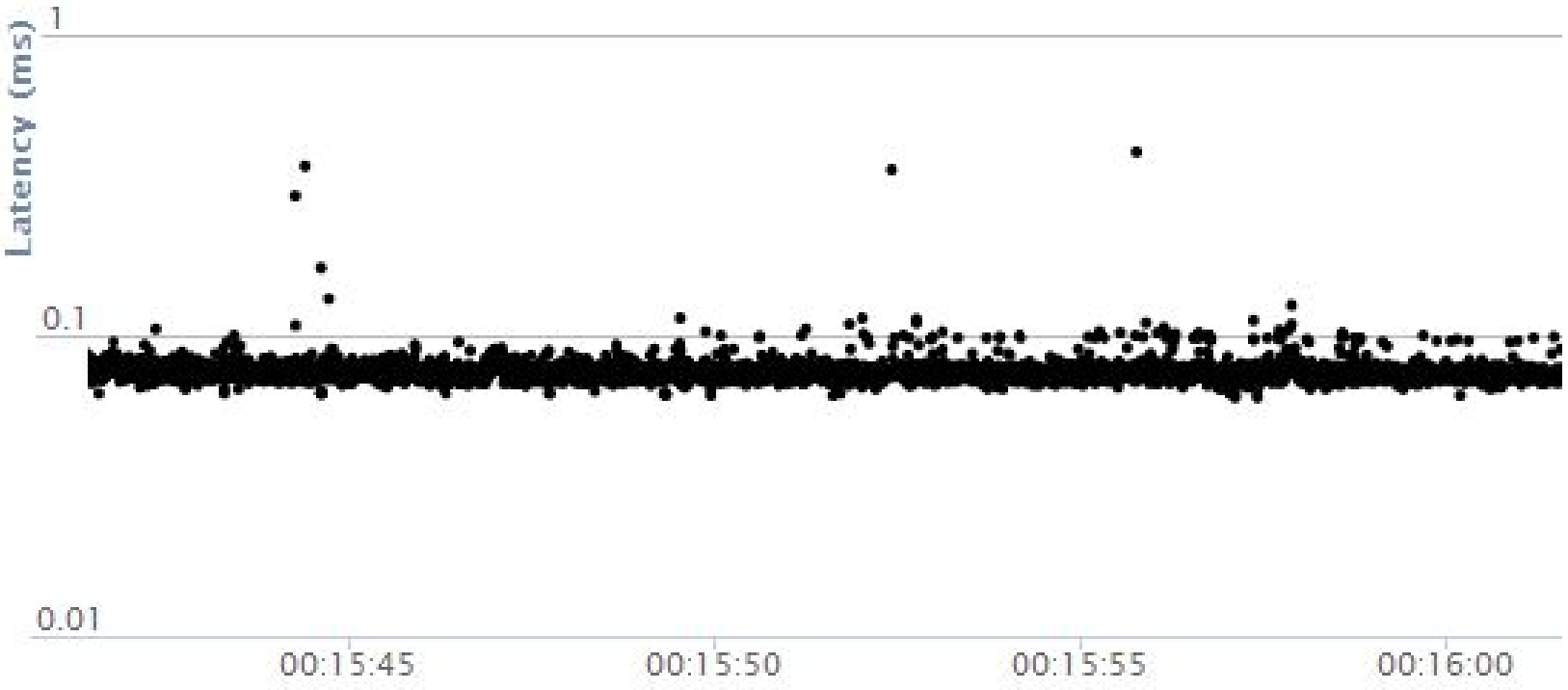
Without intel_idle driver, 1 msg/s



With intel_idle driver, 100 msg/s

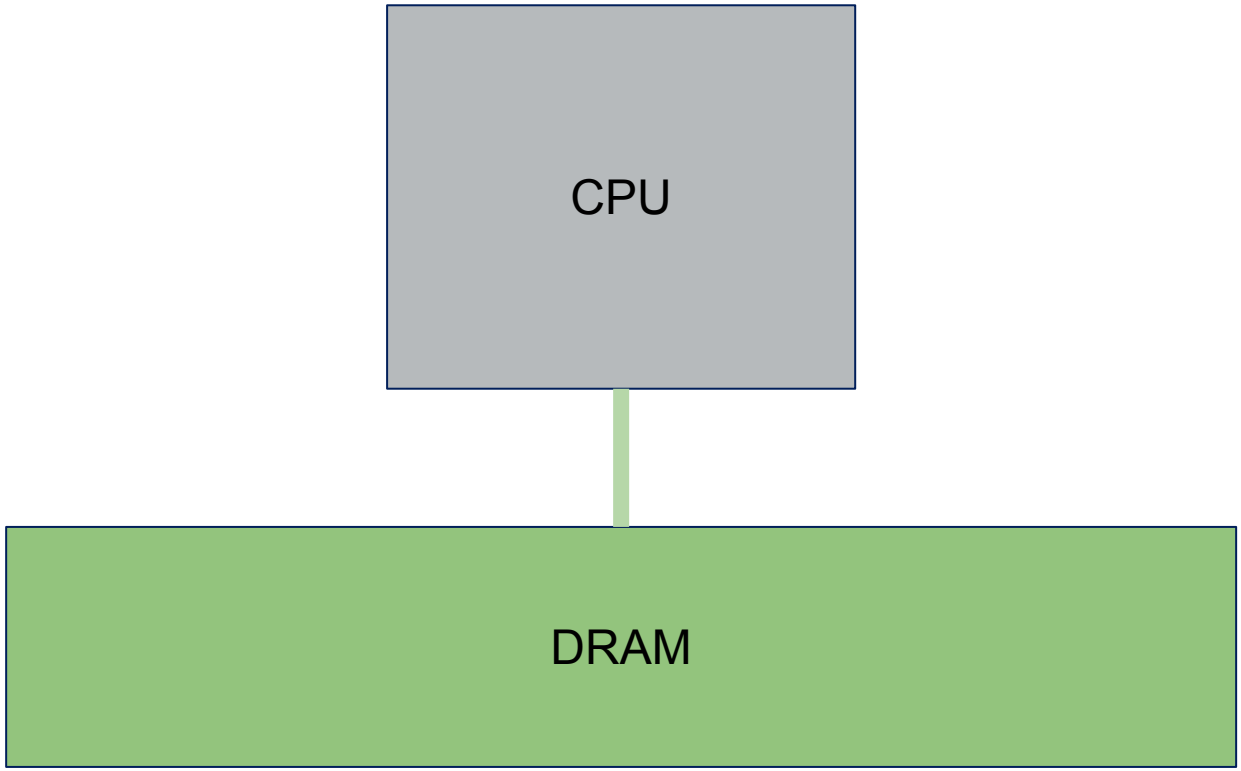


Without intel_idle driver, 100 msg/s

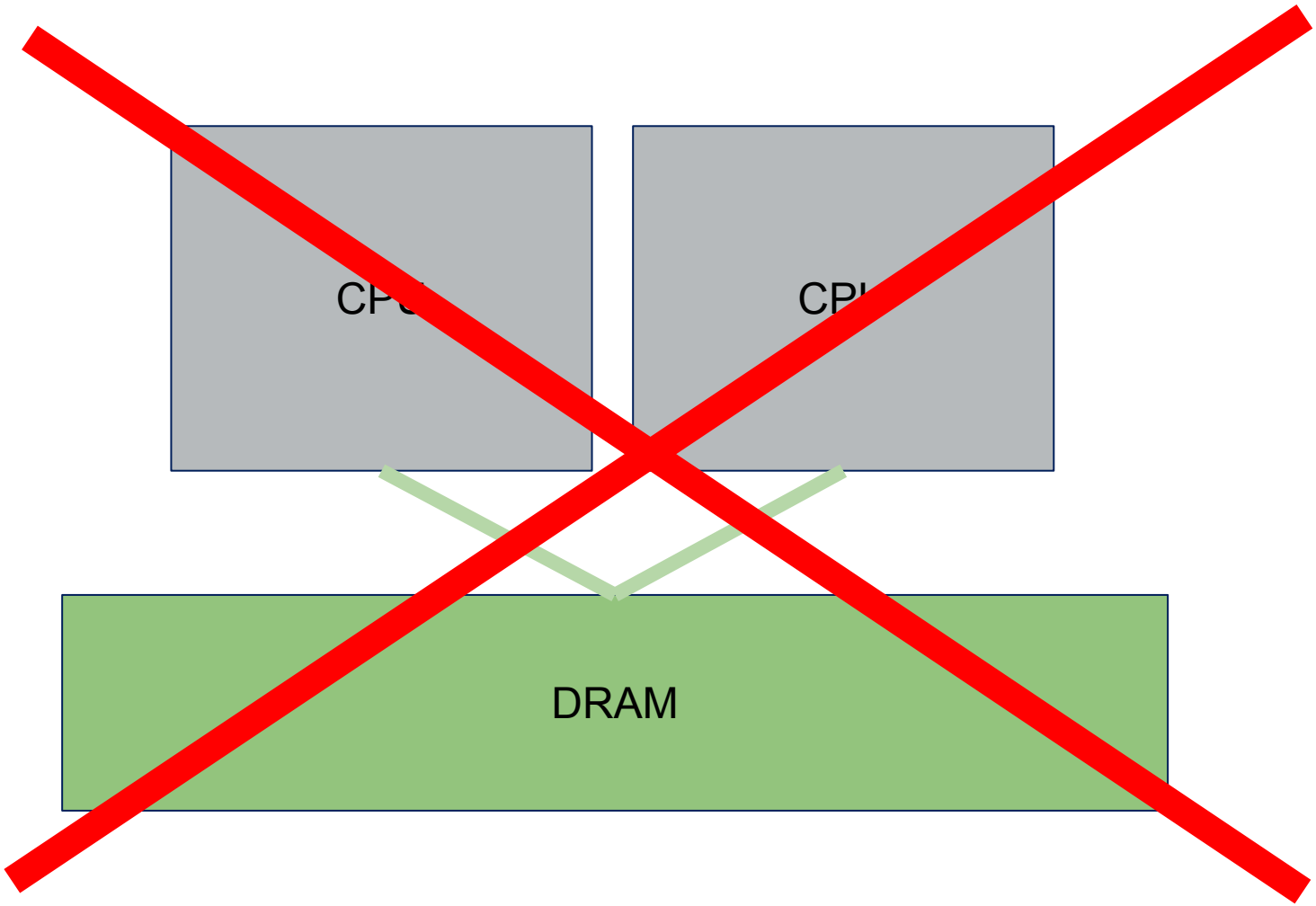


NUMA

Uniform Memory Access?



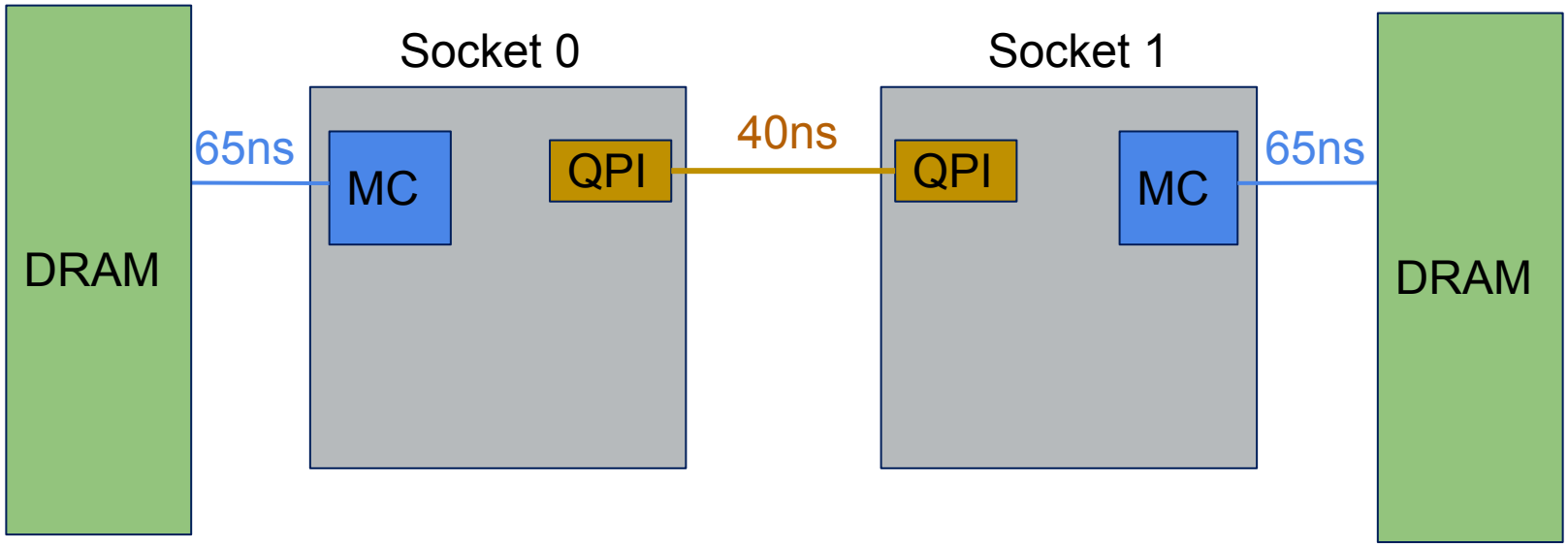
Uniform Memory Access?



NUMA

- Non-Uniform Memory Access
- Starts from 2 cpus (sockets)
- 1 Memory controller per socket (Node)
- Local Memory vs remote memory

NUMA architecture



How to deal with NUMA ?

- Avoid remote memory access
- Avoid scheduling and allocation on different node
- Bind process on one CPU only (numactl)
  Restrictive but effective

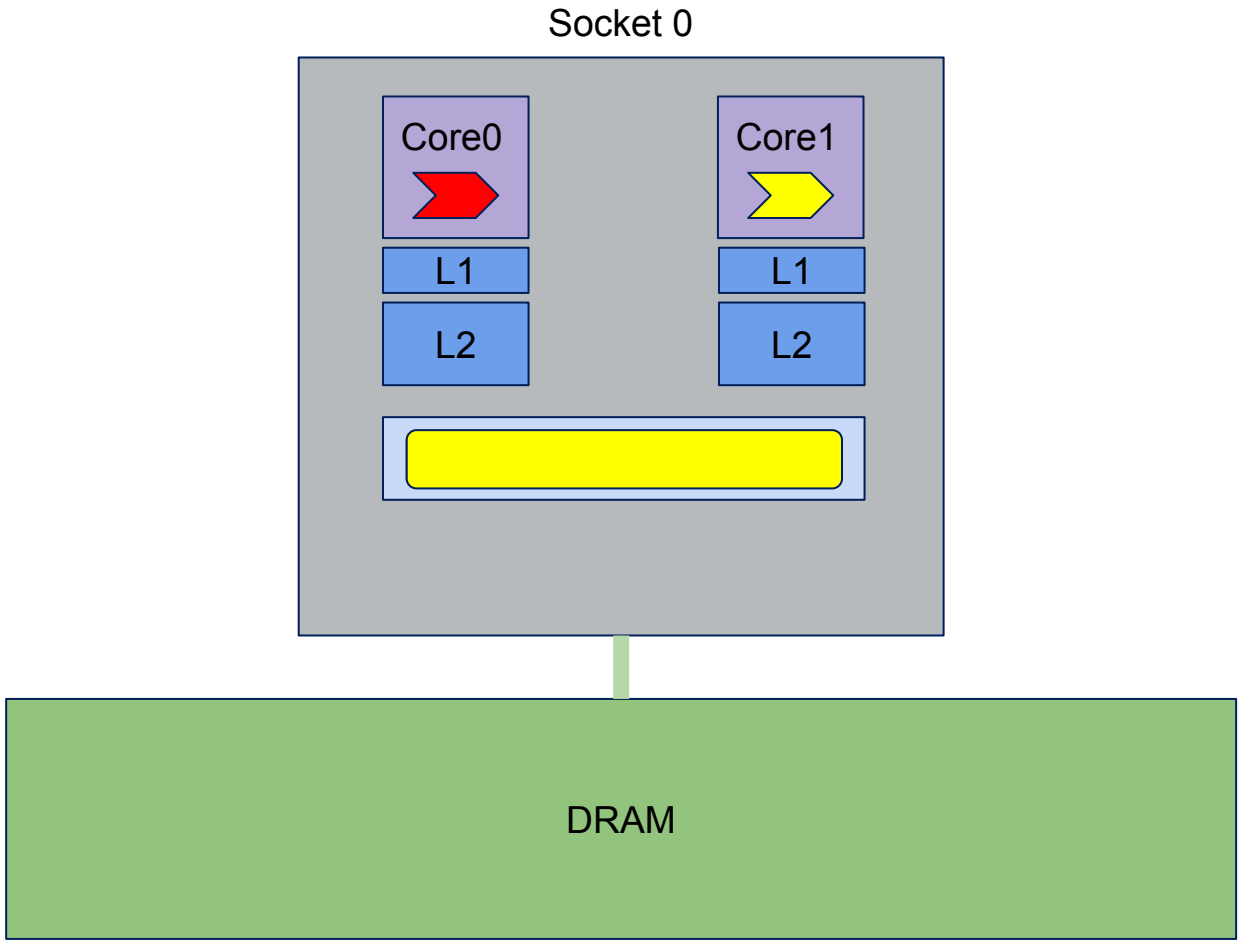
  Binding on node 0:

```
numactl --cpunodebind=0 --membind=0 java …
```

- BIOS: node interleaving disabled

Cache pollution

Cache hierarchy architecture



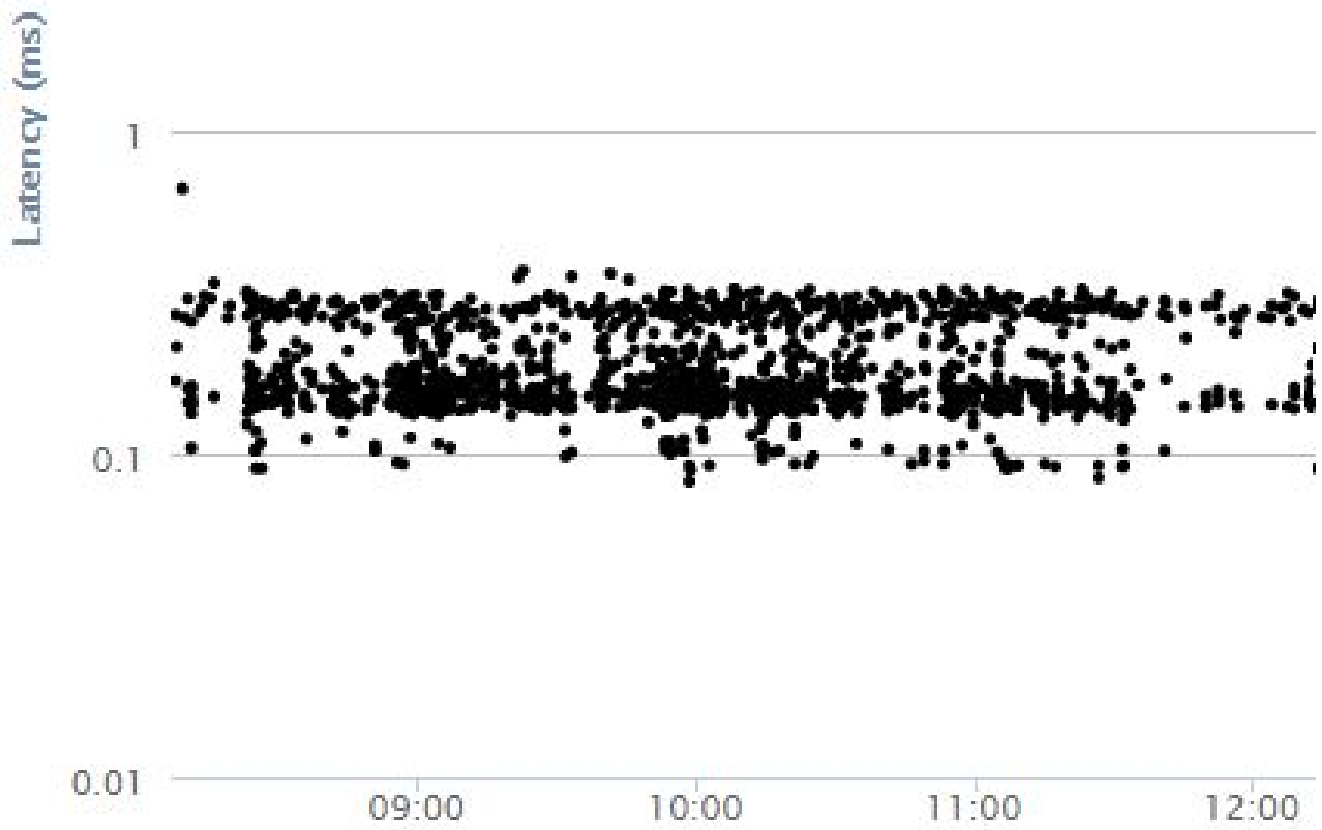
Cache pollution on L3

- CPU L3 shared cache across cores
- Non critical threads can “pollute” cache
  - eviction of data required by critical threads
  - => adds latency to re-access those data
- Need to isolate critical threads to non-critical ones

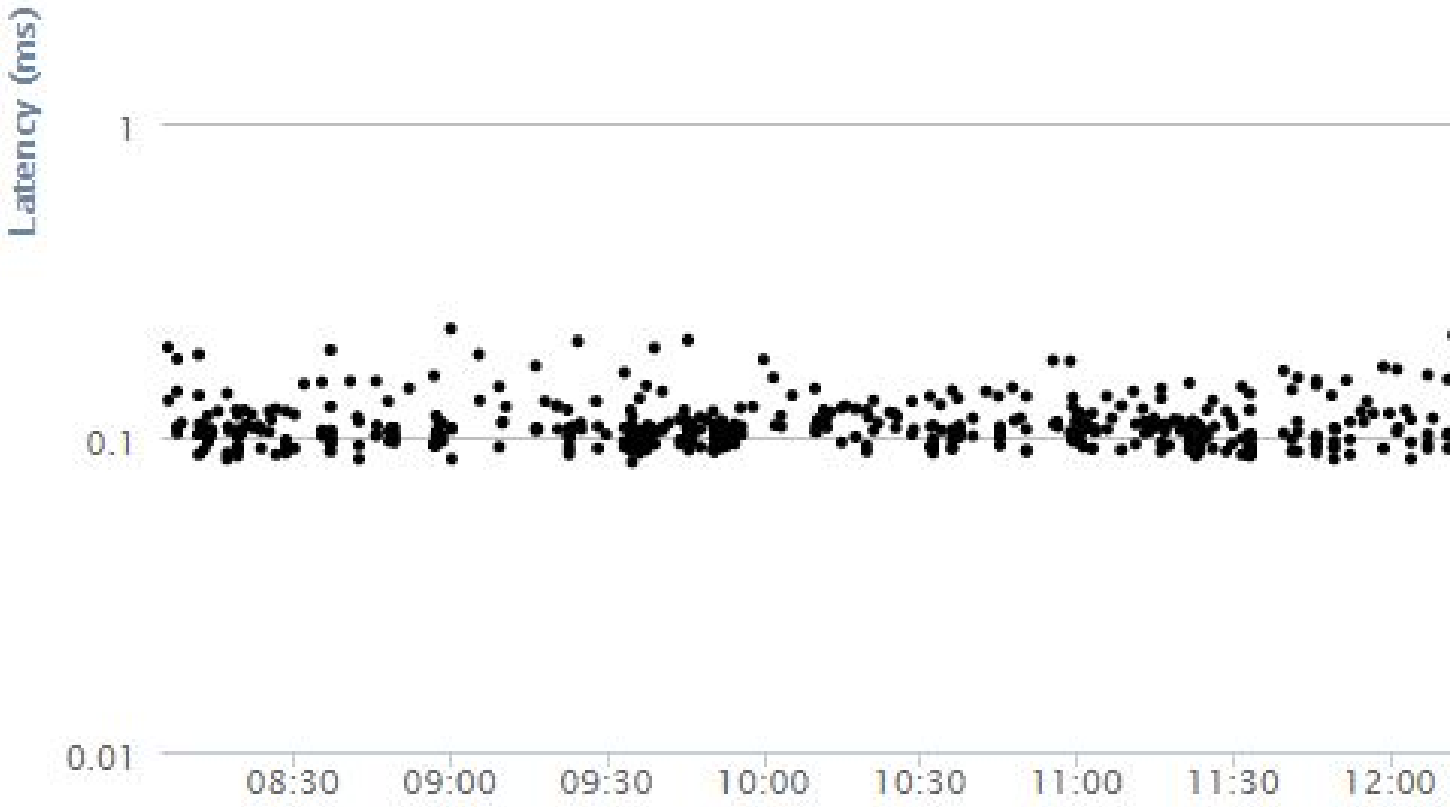
Thread affinity

- OS Scheduling cannot work in this case
- Need to pin manually threads on cores
- Thread affinity allow us to do this
- Identify our critical and non-critical threads in application

No thread affinity



With thread affinity

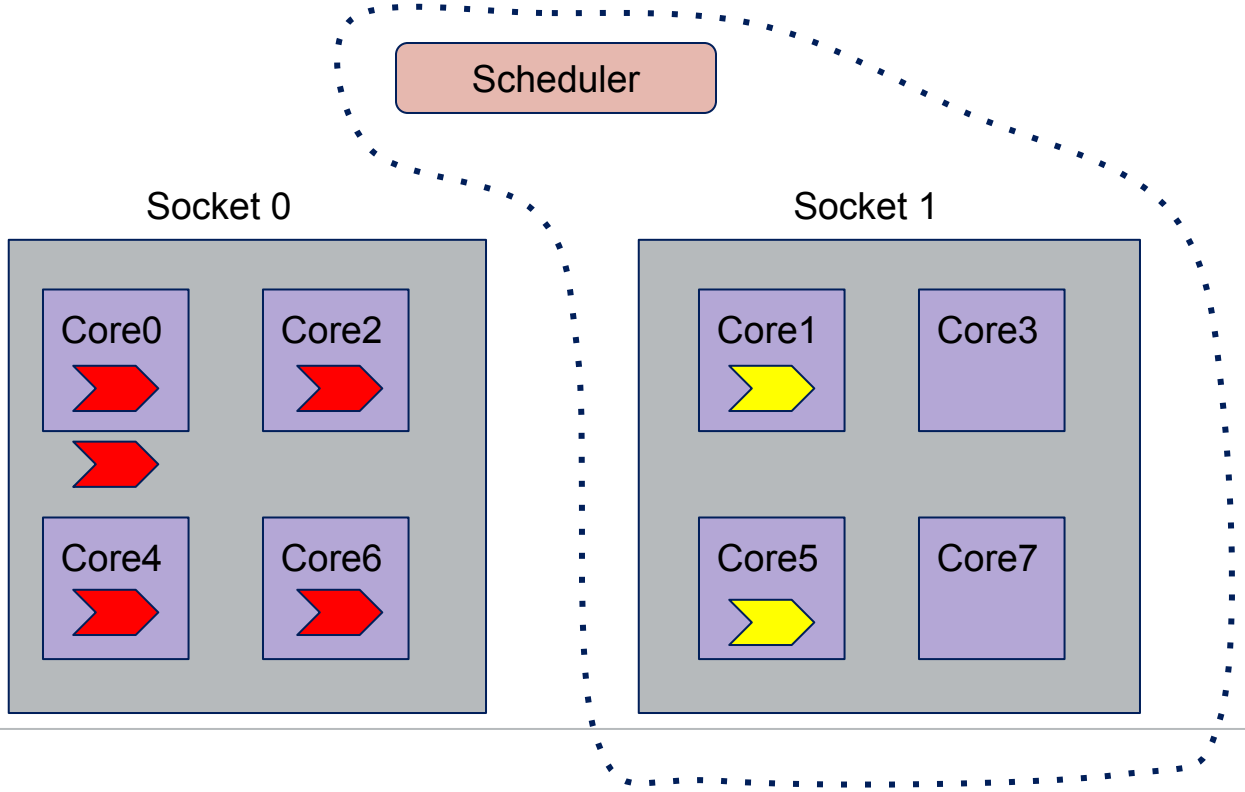


System processes & cache pollution

- Other processes can also pollute L3 cache
- Need to isolate all processes away from critical threads
- The whole CPU need to be dedicated

Isolating cores

- No other processes should be scheduled on the same socket (isolcpus)
- But if more than one thread per core need to use cpuset



Recommendations

- Make GC pauses predictable
- Tune BIOS/OS for maximum performance
- Control NUMA effect by binding on one node
- Avoid cache pollution with Thread Affinity and CPU isolation

Conclusion

- Can optimize a process without changing your code
- Know your hardware and your OS
- This is Mechanical Sympathy

References

- Mechanical Sympathy blog
  http://mechanical-sympathy.blogspot.com/
- Mechanical Sympathy forum
  https://groups.google.com/forum/#!forum/mechanical-sympathy
- Java Thread Affinity library
  https://github.com/peter-lawrey/Java-Thread-Affinity

Q&A

Jean-Philippe BEMPEL
Performance Architect

@jpbempel
http://jpbempel.blogspot.com

Ullink
CONNECT. TRADE.