Running Oracle on a 32 socket server with 24T of memory

Frits Hoogland

`whoami`

- Frits Hoogland
- Working with Oracle products since 1996

- Blog: http://fritshoogland.wordpress.com
- Twitter: @fritshoogland
- Email: frits.hoogland@accenture.com
- Oracle ACE Director
- OakTable Member

Goals & prerequisites

- Goal: Learn about characteristics of a huge system.
- Prerequisites:
  - Basic understand of hardware architecture.
  - Basic understanding of C and Linux.
  - Basic understanding of Oracle running on Linux.

SGI UV300

- CPUs: Intel(R) Xeon(R) CPU E7-8890 v2 @ 2.80GHz
  - (Ivy Bridge EX)
- 32 sockets
- 480 cores (15 cores/socket)
- 960 threads (intel hyper threading)

- 32s480c960t

Memory

- Total memory size: 24TB
- Memory is local to a socket

- (24*1024)/32 = 768 GB / socket

```
# numactl --hardware | grep size
node 0 size: 753624 MB
...
node 31 size: 753648 MB
```

Memory

- Memory is DDR3 @ 1333Mhz

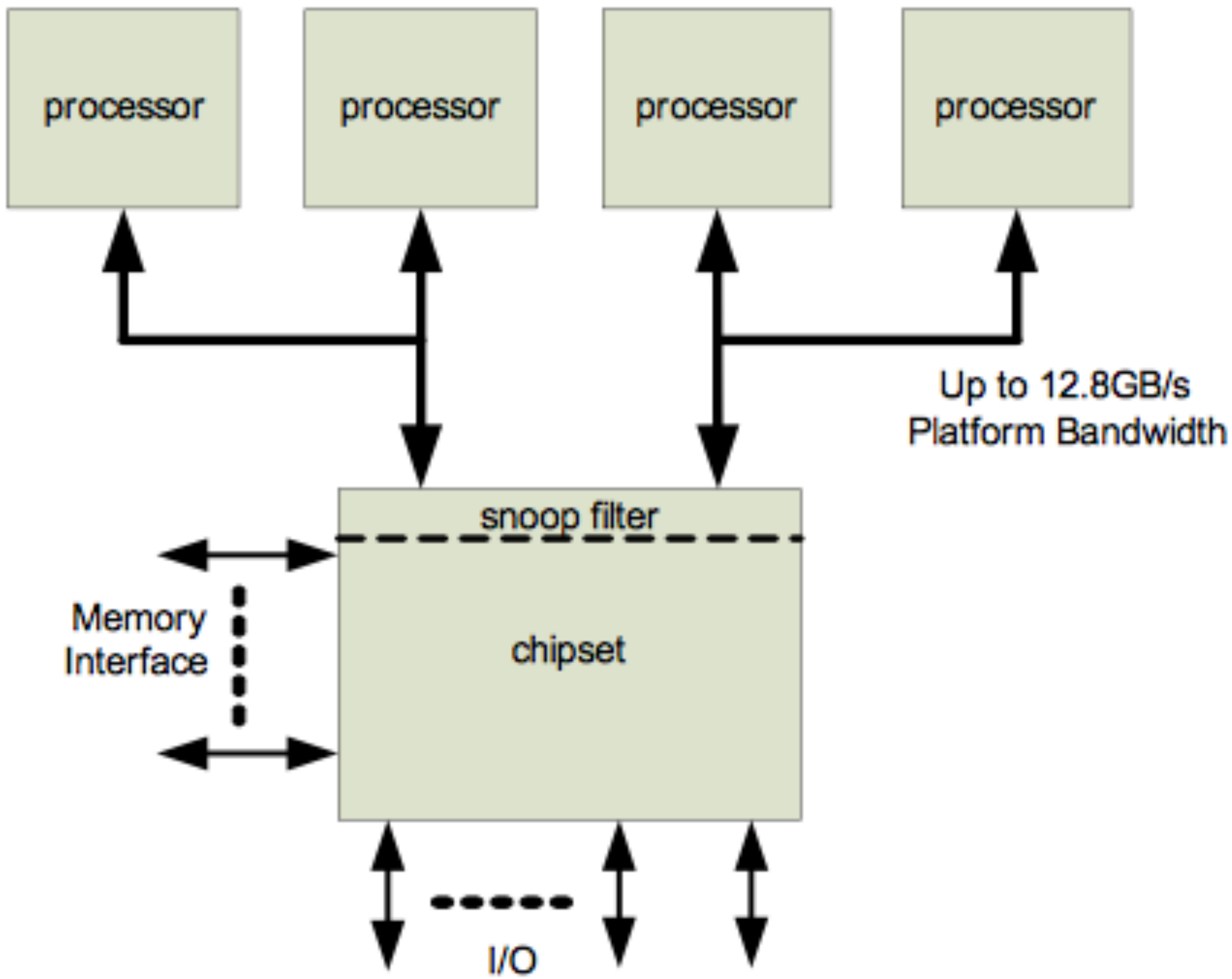
```
# dmidecode | grep -A13 'Memory Device'
Memory Device
    Array Handle: 0x0001
    Error Information Handle: Not Provided
    Total Width: 72 bits
    Data Width: 64 bits
    Size: 32 GB
    Form Factor: DIMM
    Set: 8
    Locator: DIMMD2
    Bank Locator: MEM8
    Type: DDR3
    Type Detail: Synchronous
    Speed: 1333 MHz
    Manufacturer: Samsung
```

History: UMA

- Uniform Memory Access

- SMP in the 90s.
- Intel bus architecture: FSB.
  - Pentium Pro & Pentium II
  - Northbridge (memory controller hub)
  - Southbridge (I/O controller hub)
  - Architecture provided limited scalability.

History: UMA

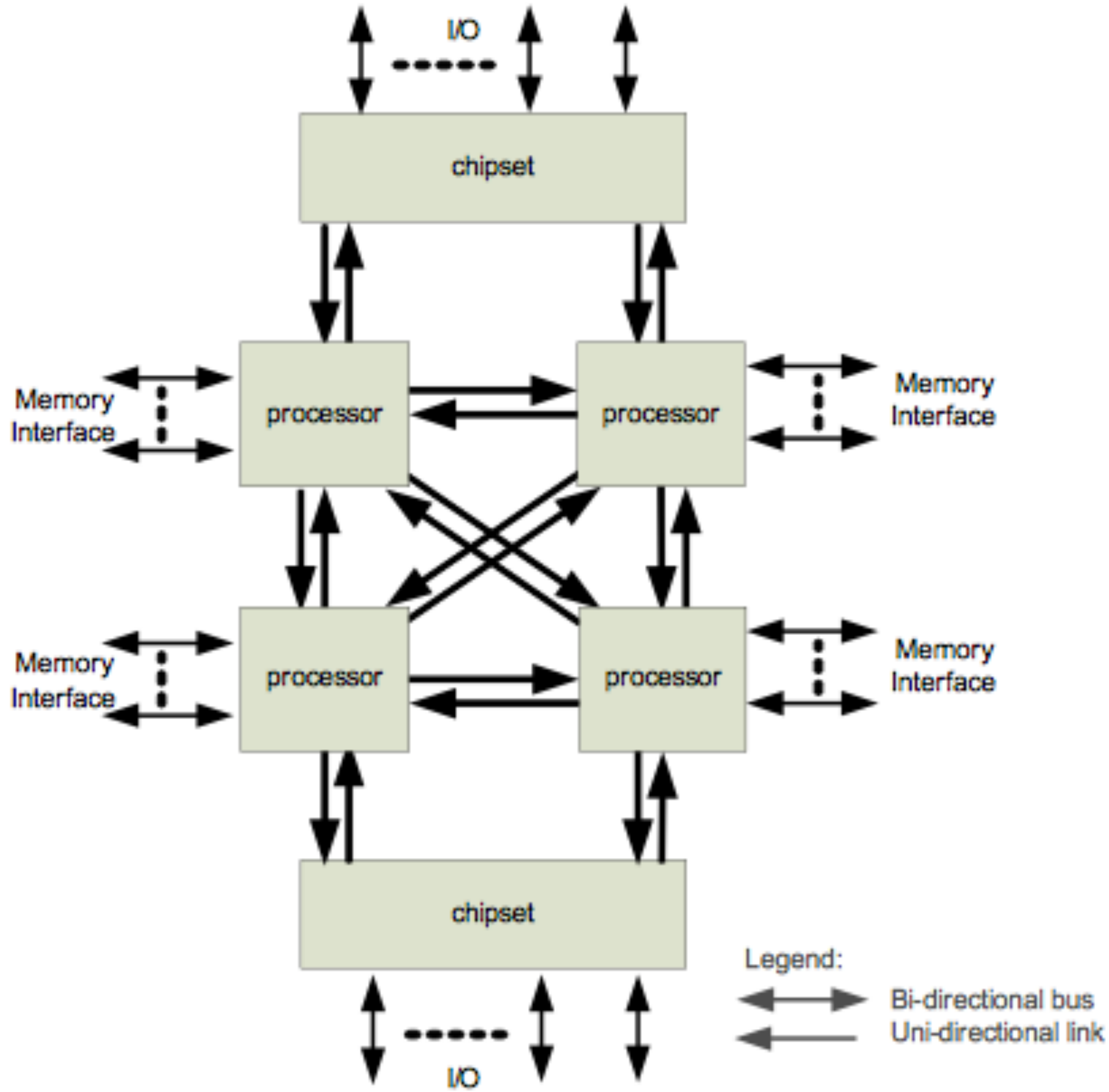

processor
processor
processor
processor
Up to 12.8GB/s
Platform Bandwidth
snoop filter
Memory
Interface
chipset
I/O

NUMA

- Non Uniform Memory Access

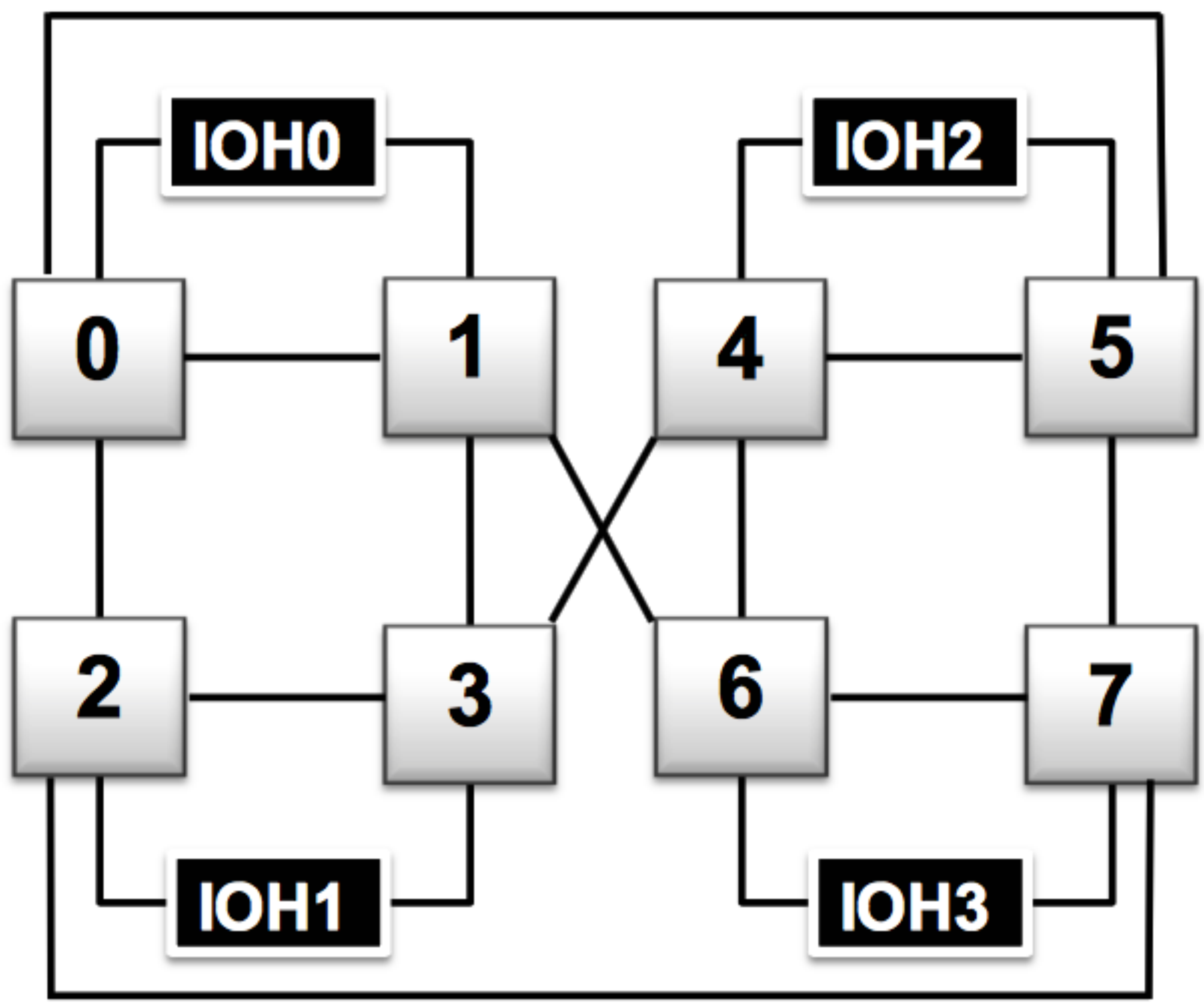
- Memory local to Socket.
  - Allowing much more memory in a server.
- Each socket can also have its own IO channel.
  - Allowing higher IO rates.
- Sockets interconnected using QPI.
  - For Intel based system starting from Nehalem.

NUMA



I/O
chipset
Memory
Interface
processor
processor
Memory
Interface
Memory
Interface
processor
processor
Memory
Interface
chipset
I/O
Legend:
Bi-directional bus
Uni-directional link

NUMA - scaling up beyond 4 s.

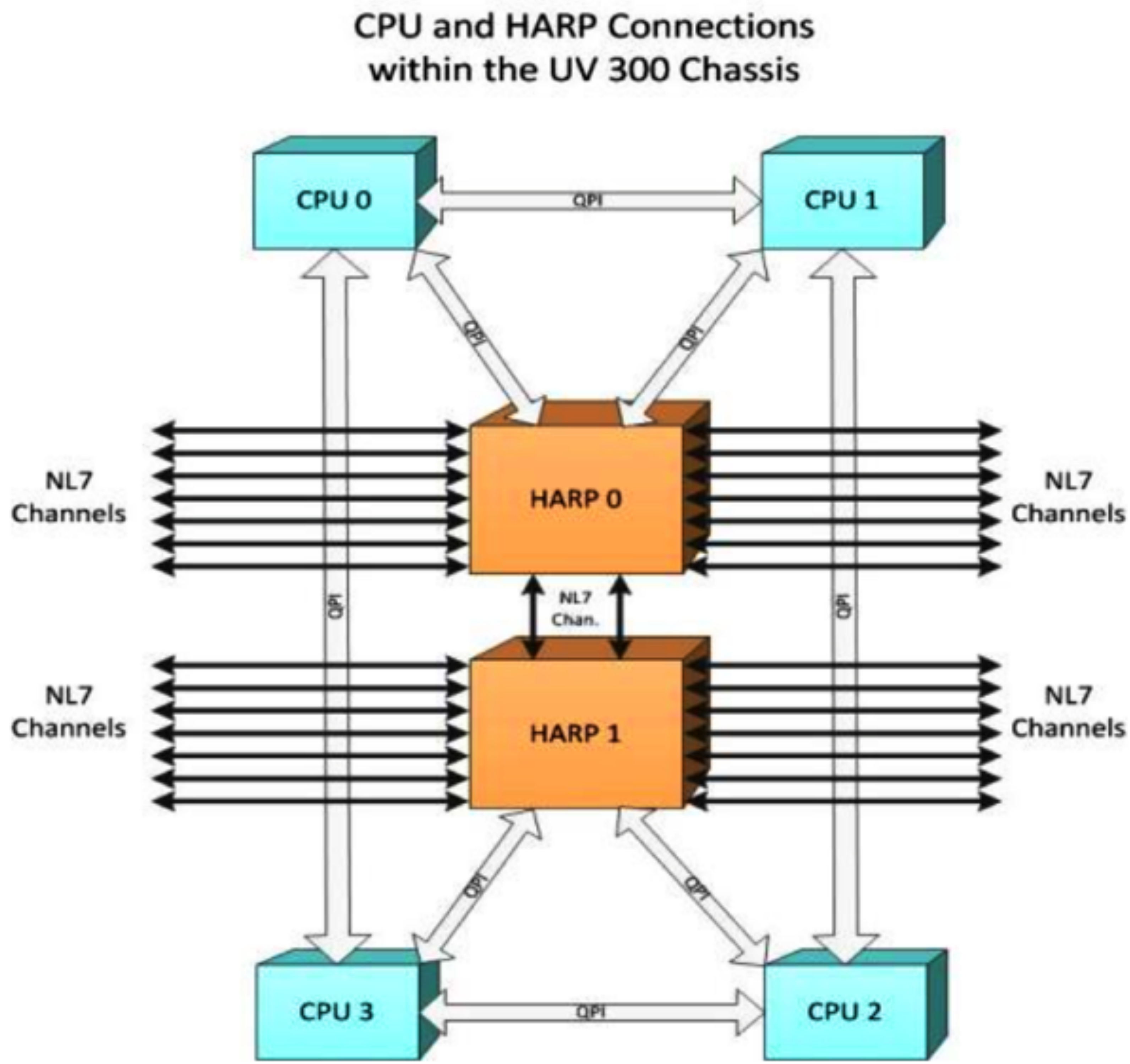


NUMA - SGI UV 300

- So how can the UV300 have 32 sockets?

- The sockets are grouped by 4.
- And include two “HARPs”: CPU interconnects.
- HARPs use SGIs NumaLink7 interconnect.

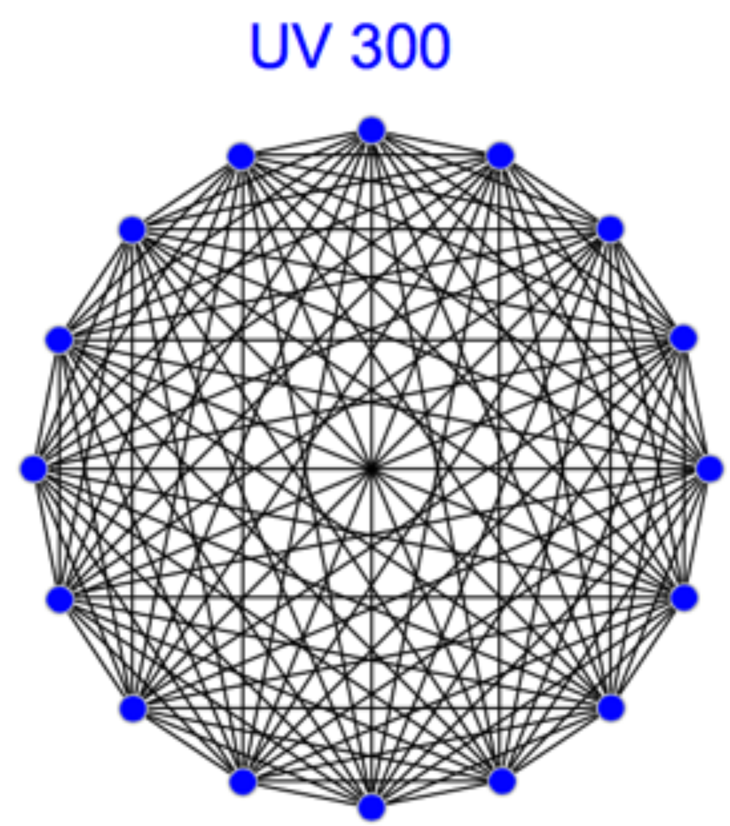
NUMA - SGI UV 300



enkitec

NUMA - SGI UV 300

- The HARPs are connected to every other HARP.
- This means a socket is local, 1 or 2 hops away.



Performance

- Use ‘numactl --hardware' to learn the ‘distance’

```
# numactl --hardware
node distances:
node   0   1   2   3   4   5   6   7   8   9  10  11  12   …
  0:  10  16  19  16  50  50  50  50  50  50  50  50  50   …
  1:  16  10  16  19  50  50  50  50  50  50  50  50  50   …
  2:  19  16  10  16  50  50  50  50  50  50  50  50  50
  3:  16  19  16  10  50  50  50
  4:  50  50  50  50  10  16  19
  5:  50  5   50  50  16  10  16
  …                       …   …
```

Next socket

Local sockets

Same distance for numalink remote sockets.

1 hop

Systems performance

- How does such a system behave?
- Test memory read performance with SLB!
- http://kevinclosson.net/2010/11/17/reintroducing-slb-the-silly-little-benchmark/
- Reads anonymous memory into CPU register

Systems performance

make read go into cpu register

- What SLB does (snippets of memhammer.c)

```
#define A_RANDOM_LINE  ((mybuffer *) (base_addr + pick_buff()))->buffer[(ops %
LINES_PER_BUFFER) * INTS_PER_LINE ]
volatile register char tmp;

  memset ((void *) base_addr, (int) 0, ((sizeof (mybuffer)) * num_buffers));

  for (i = 0, cur_buffer = base_addr; i < num_buffers; i++, cur_buffer++) {
      for (j = 0; j < INTS_PER_BUFFER; j++)  {
          cur_buffer->buffer[j] = (int) i;
      }
  }

  for (ops=0LL ; ops < 3000000000LL; ops+=1LL) {
      tmp = (char) A_RANDOM_LINE;
  }
```

randomizer

allocate memory and initialise it

read data from main memory

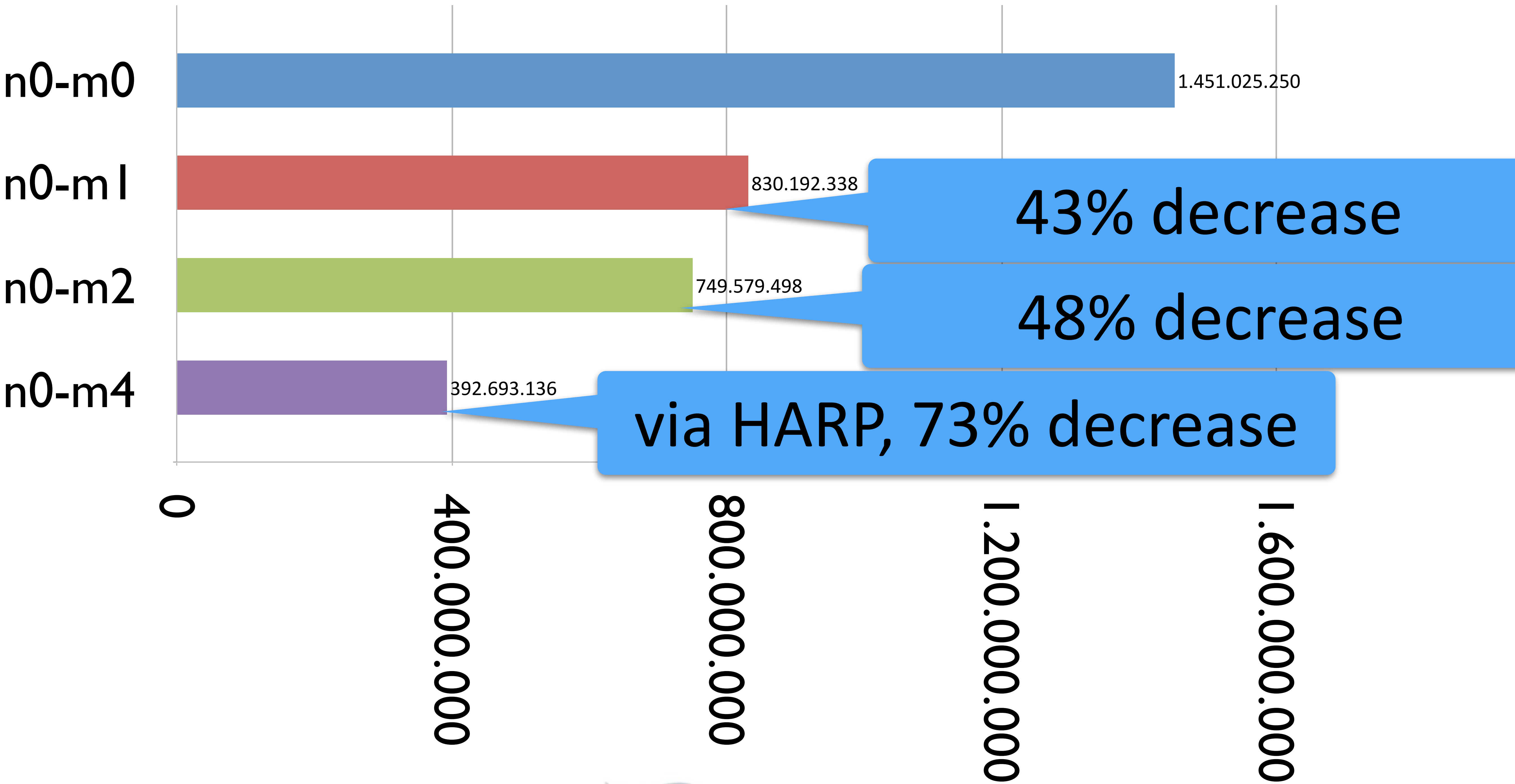
Systems performance

- Why describe this detailed?

- This has nothing to do with Oracle!
- Pure flow from main memory to CPU, 8 bits.

- Reveals memory latency very well!

SLB - 1 reader throughput



enkitec

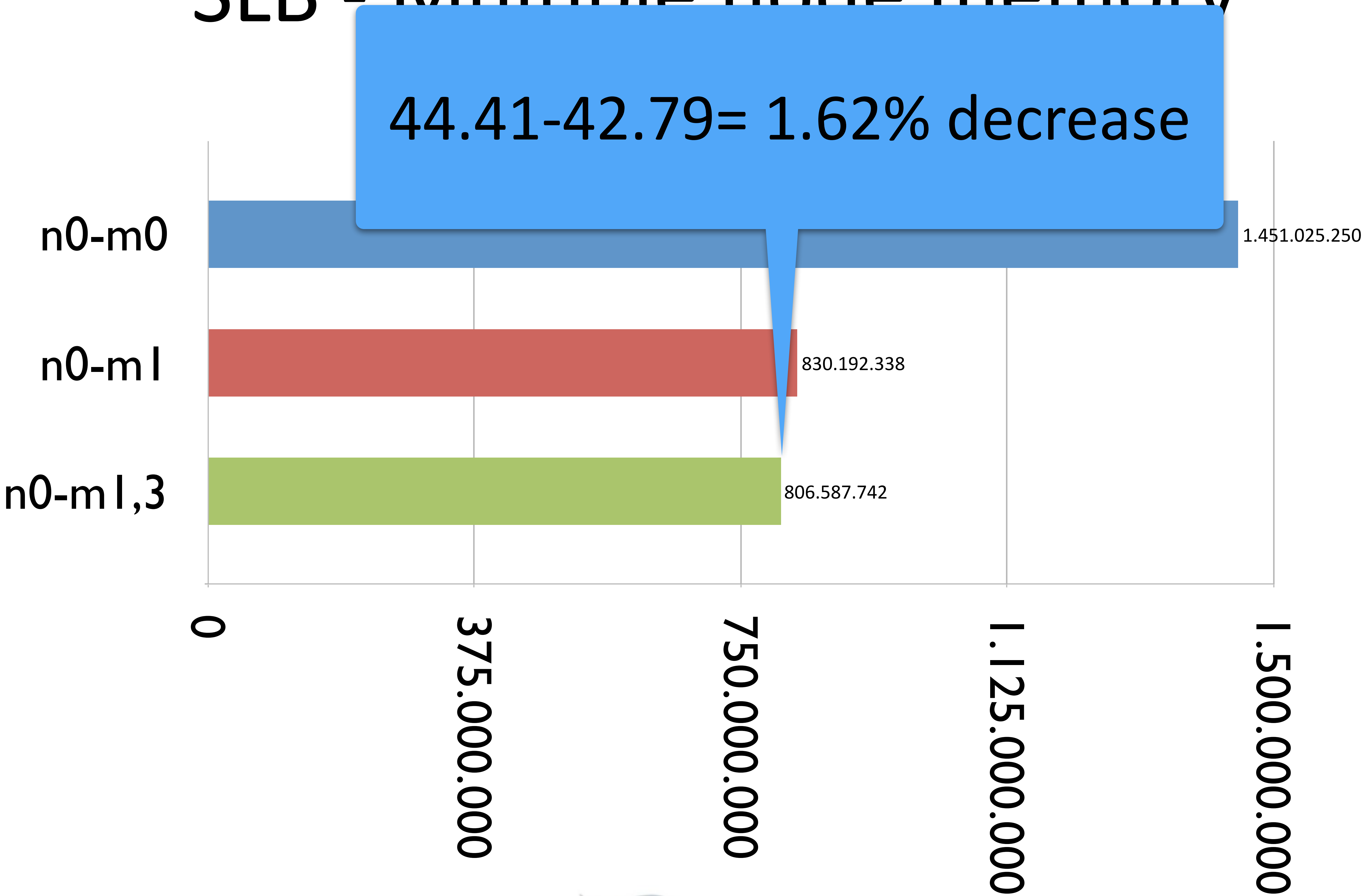
Systems performance

- Is this a problem/flaw?

- No: fact of life.
  - Further away resources means increase in latency.

enkitec

SLB - Multiple node memory



enkitec

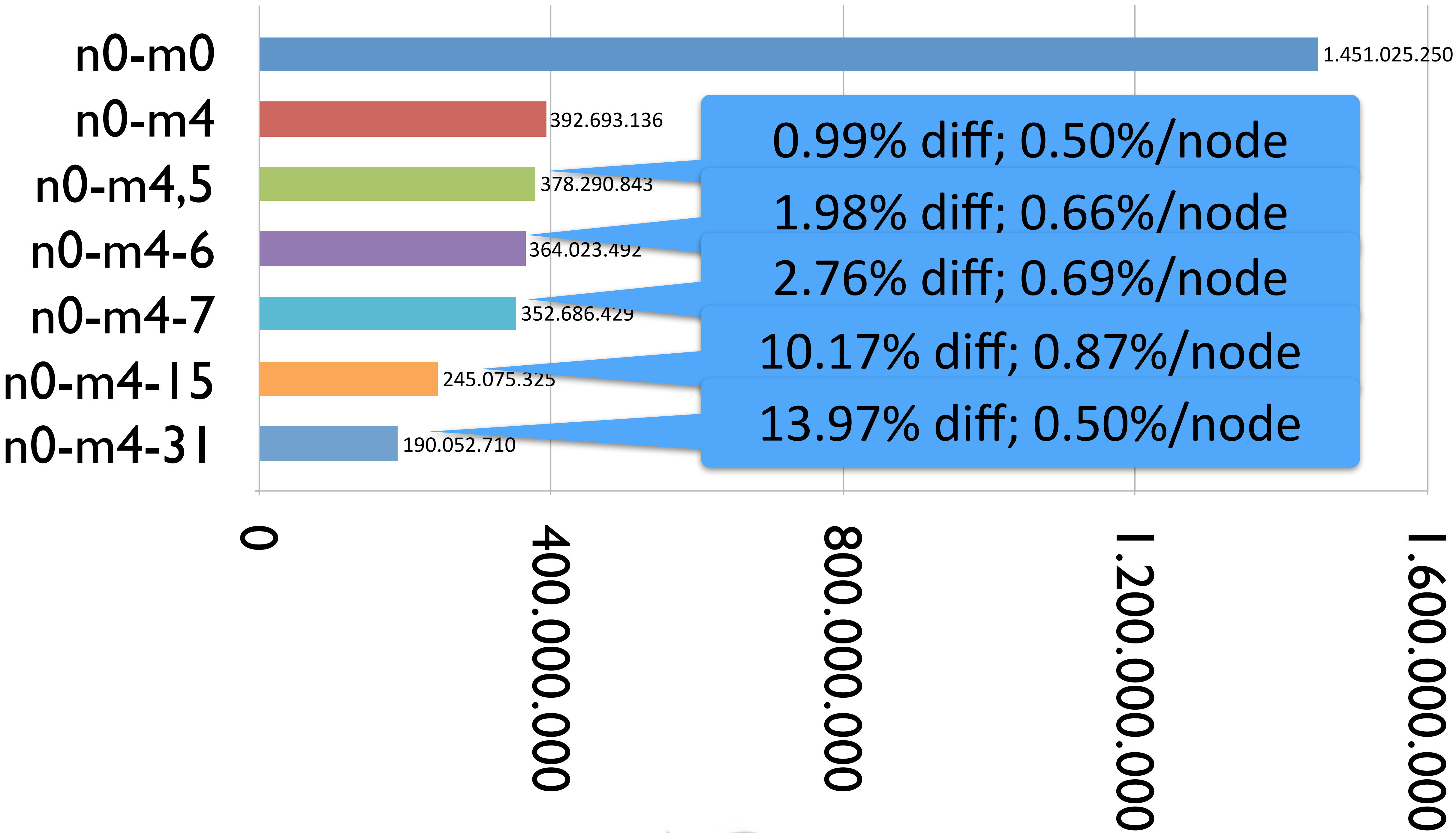
Systems performance

- By increasing the number of sockets/nodes
  - Latency increases.
  - There’s only two nodes at distance 16.

- Let’s look at accessing memory via the HARP!

enkitec

SLB - Multiple node memory



enkitec

Systems performance

- Conclusions so far:
  - Accessing remote memory increases latency.
  - Further away memory gives higher latency.
  - Accessing multiple sockets' memory increases latency to:
    - Local sockets: 0.81%/socket (1.62/2)
    - Via HARP: between 0.50-0.87%/socket.

enkitec

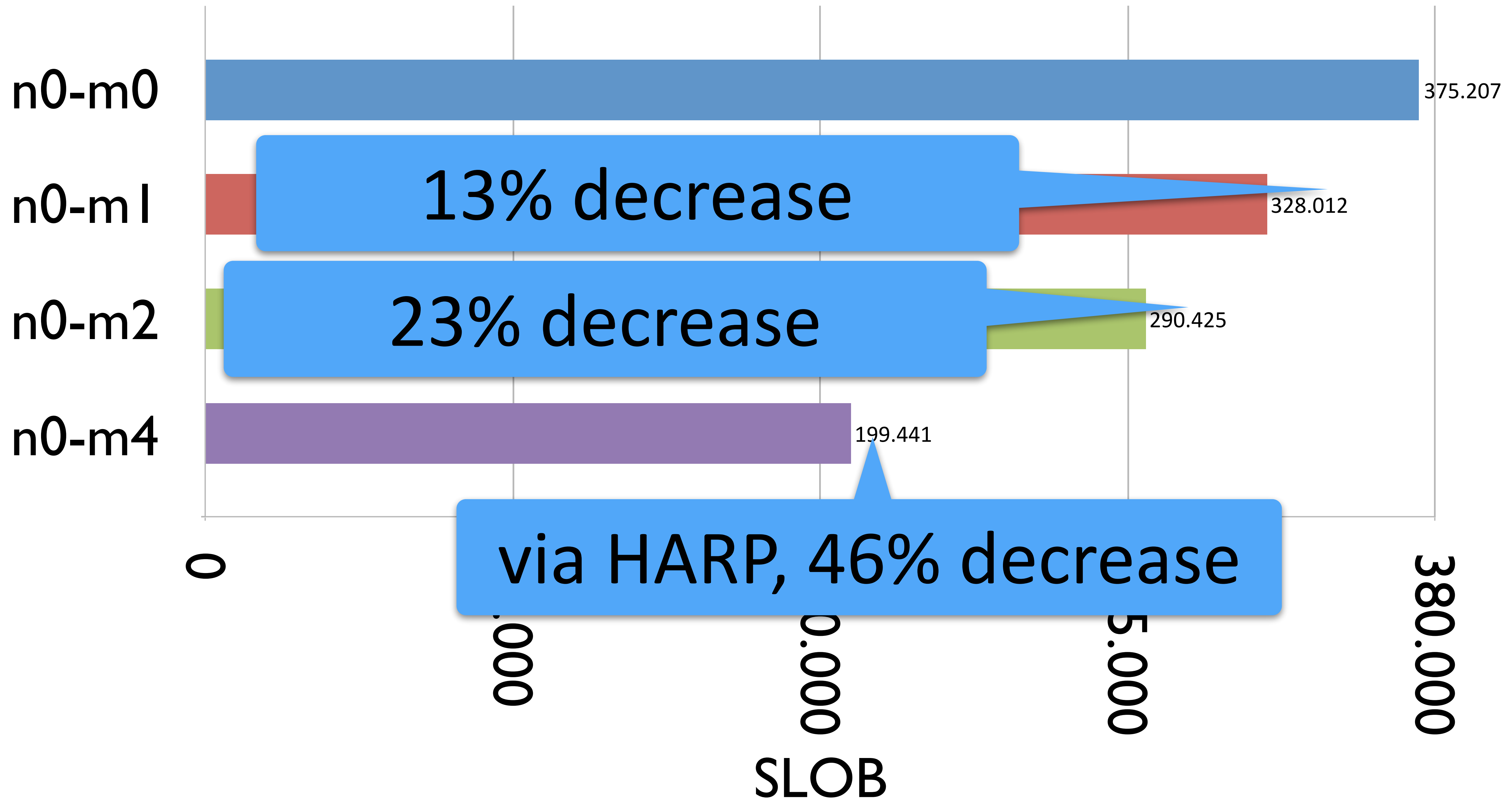
Oracle performance

- Now let’s measure running the Oracle database!

- http://kevinclosson.net/slob

- Test single block access in memory.
- Also known as ‘LIO benchmark’.

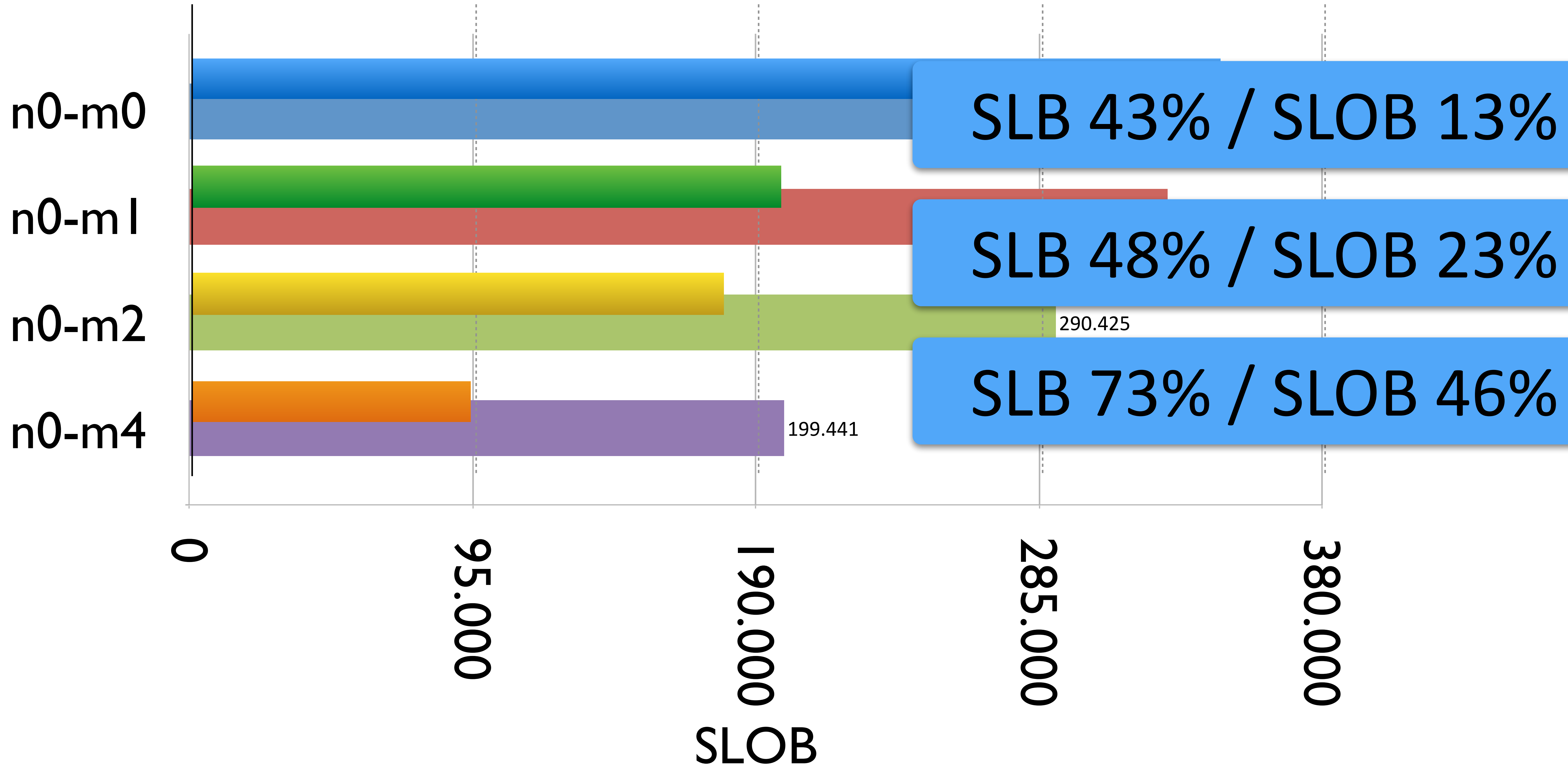
SLOB - 1 reader throughput



SLOB - 1 reader throughput

- This looks different than the memory benchmark
- Let's overlay the SLOB results with the SLB results.

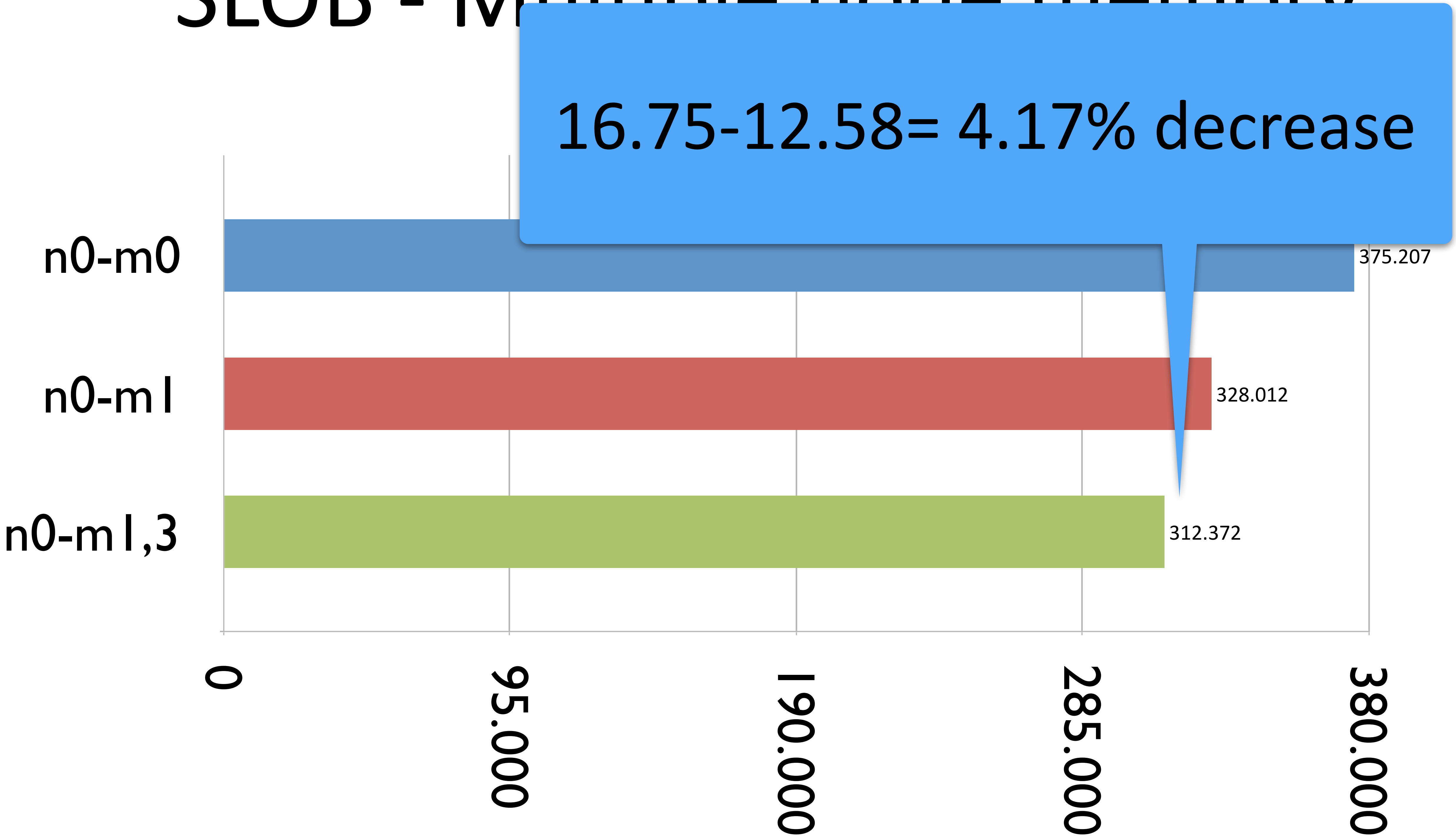
SLOB - 1 reader throughput



SLOB - 1 reader throughput

- The Oracle throughput does NOT decline as fast as the SLB one.
  - (this specific) Oracle load is not only accessing memory.
  - This probably means:
    - It is doing processing using L1/2/3 caches!
    - Probably a result of many years of tuning.

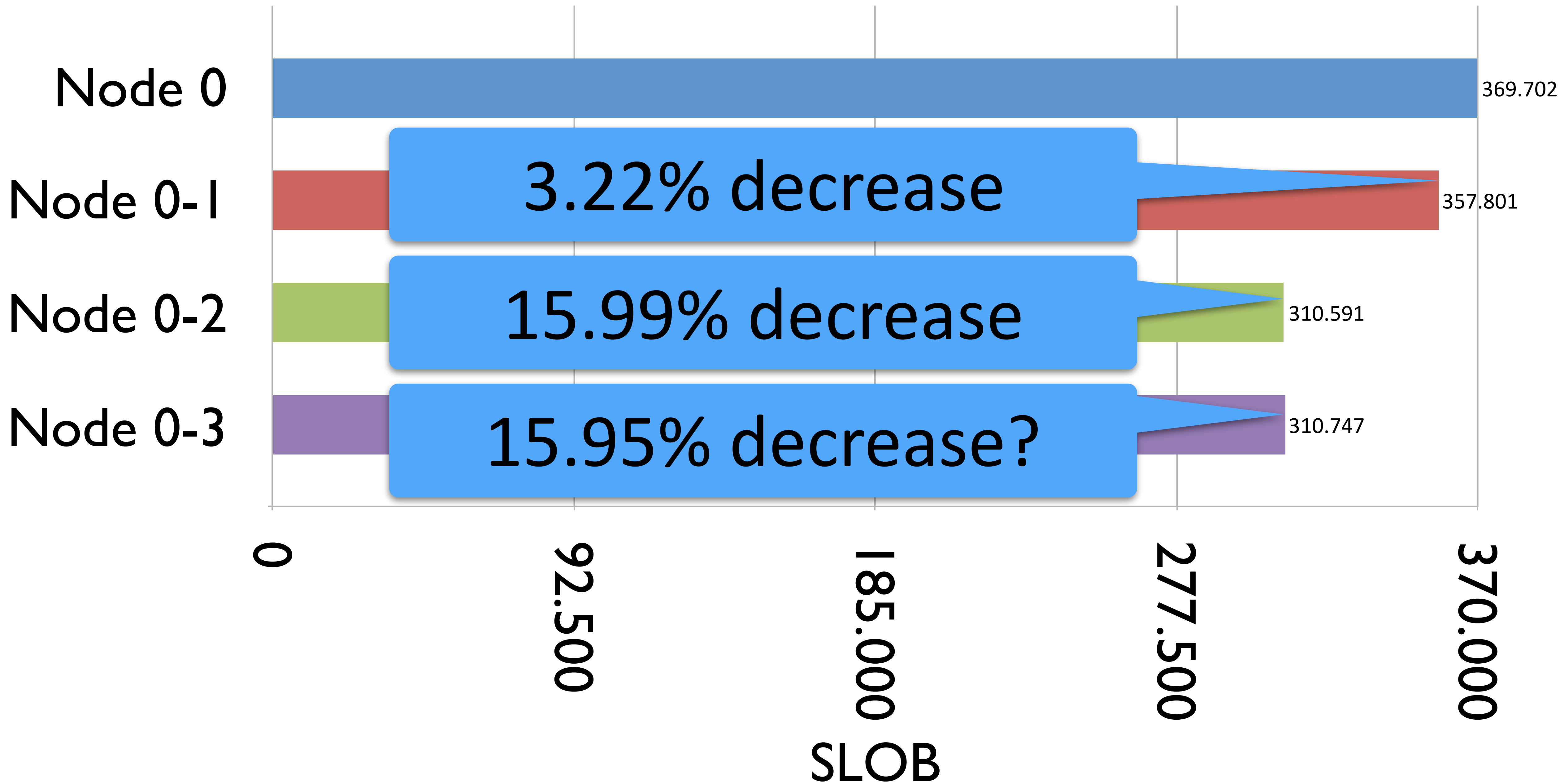
SLOB - Multiple node memory



SLOB - Local NUMA nodes

- Previous measurements are done to measure memory latencies.
- These **do not** show real life usage.
- The next slide is an overview of running on one to four NUMA nodes.

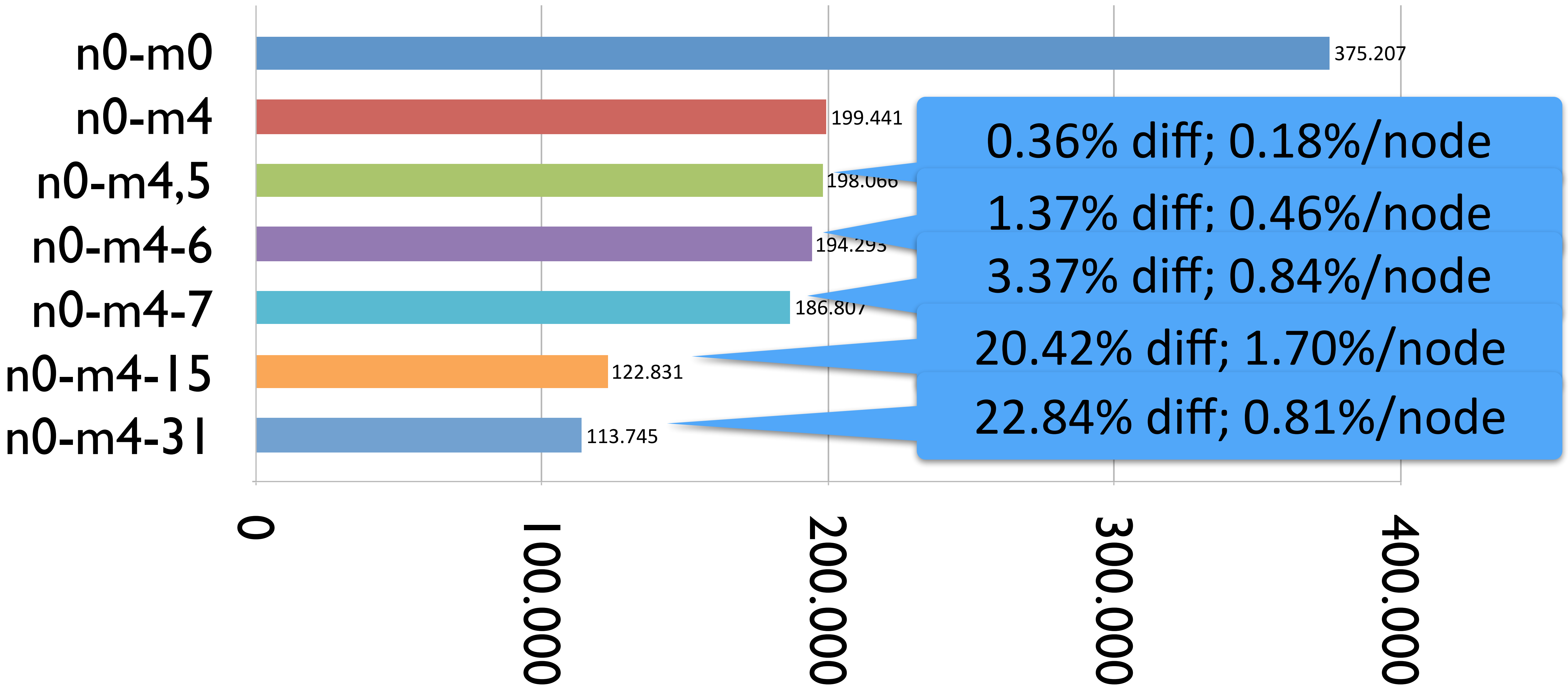
SLOB - 1 to 4 nodes / 1 reader



SLOB - Local NUMA nodes

- Conclusion for Oracle for up to 4 sockets:
  - Latency increases moderately.
    - Way less than pure memory access.
  - For two node servers, don’t enable NUMA.
  - For up to four nodes.
    - Milage varies. Probably not worth the effort.
    - Test your own load.

SLOB - Multiple node with HARP



Oracle/SLOB performance

- Conclusions so far:
  - Accessing multiple sockets' memory increases latency to:
    - Local sockets: 2.09%/socket (4.17/2)
    - Via HARP: between 0.18-1.70%/socket.
      - Reason for the difference Oracle <> SLB:
        - Probably L1/2/3 cache influence.

enkitec

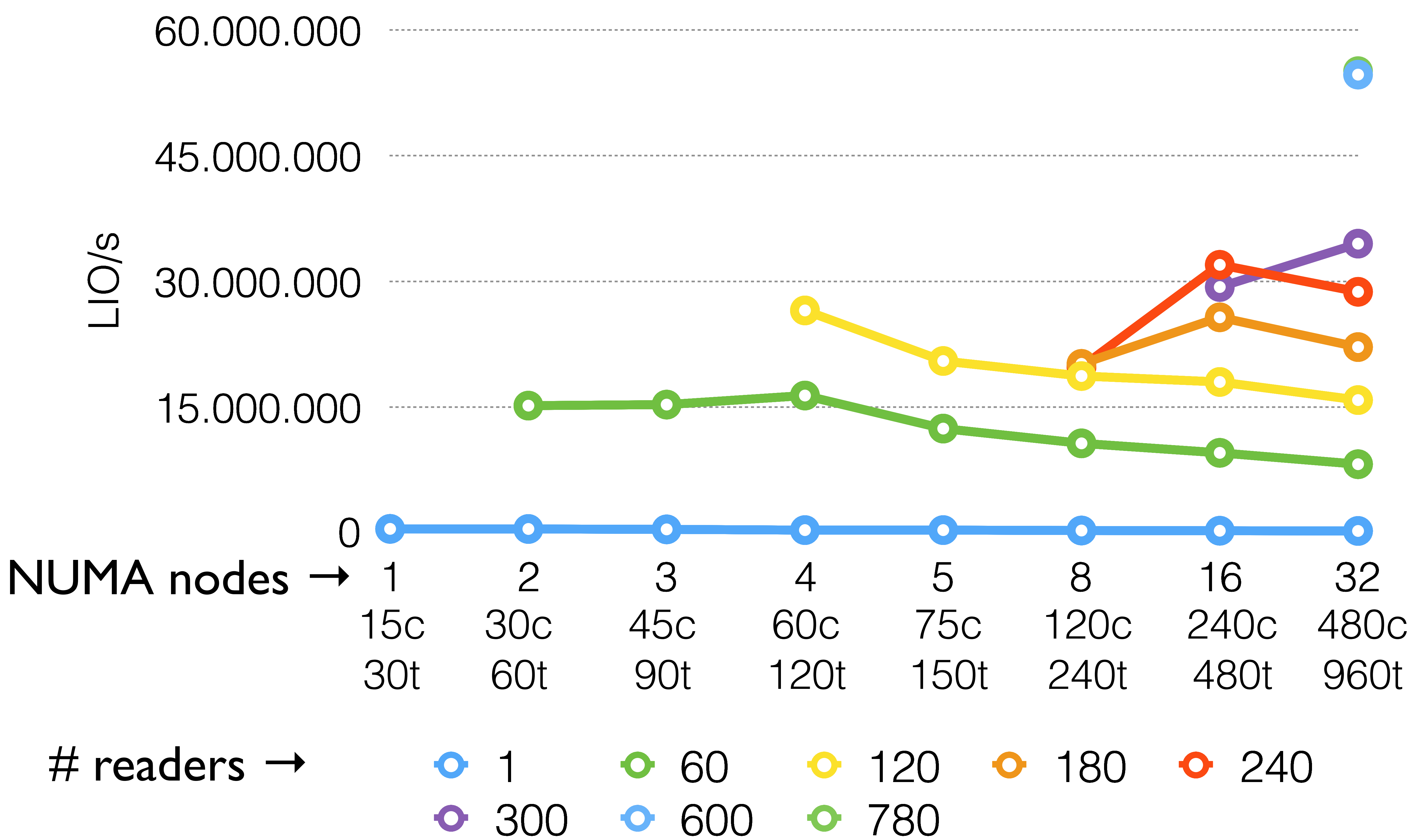
Oracle performance

- Now let's focus on bandwidth with LIO:
  - Start independent sessions without affinity.
  - Using SLOB.
  - Every reader reads its own schema.
  - Index range scans.

- Run SLOB until PIO vanishes from AWR.
- Then measure SLOB run.

enkitec

SLOB readers throughput



enkitec

Scan a big table with PQ

- Created a set of tables with the TPCH kit.
  - Table H_LINEITEM is the biggest one.
  - Size: 739G / 96’864’152 blocks.
  - 5’999’989’709 rows. 6 billion rows!

  - Created with SCALE=1000

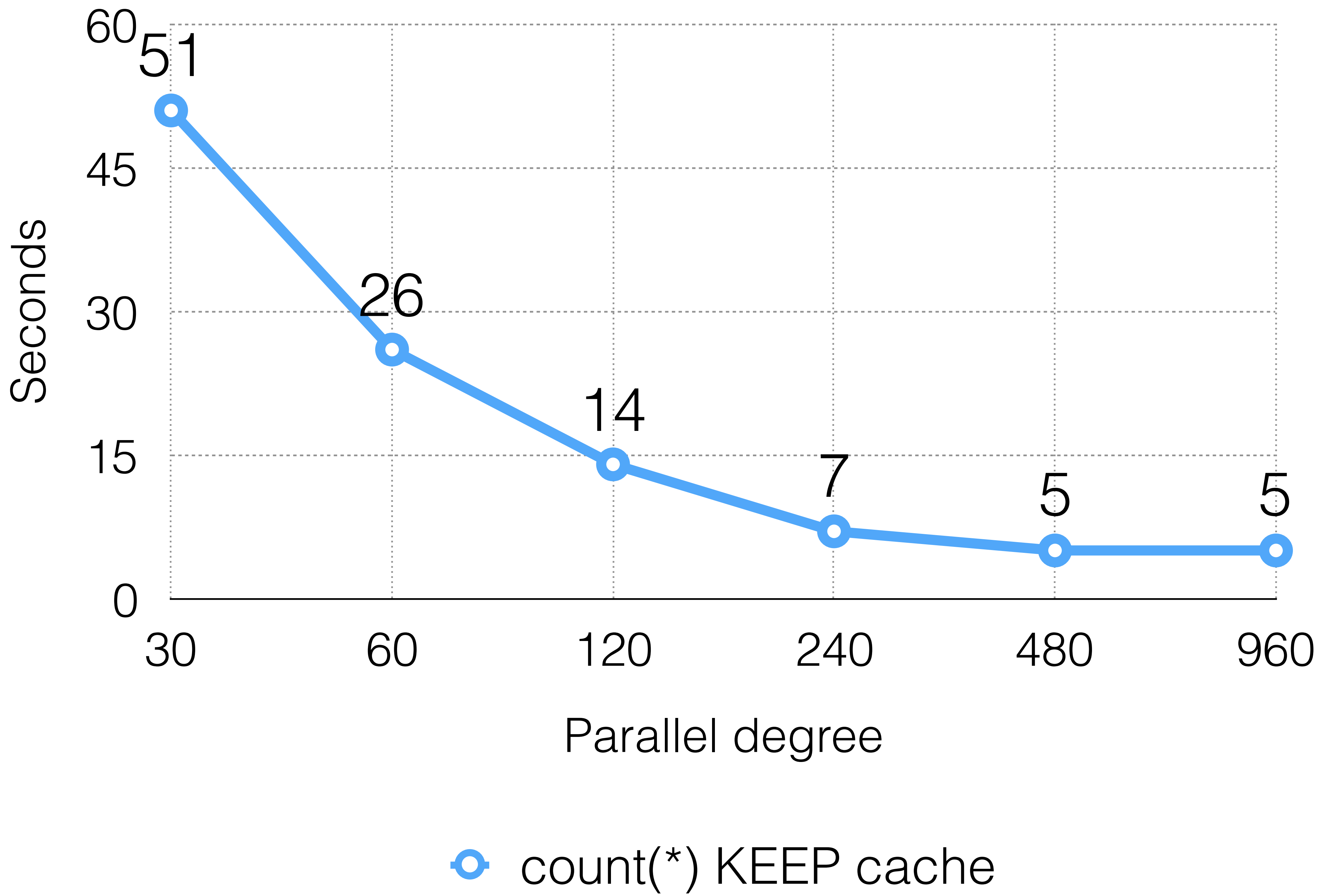
Scan a big table with PQ

- Set my buffercache to be 10T.
- Scanned table with in memory PQ option
  - alter session set parallel_degree_policy=auto;

- Normal scanning only read 1/3rd in the buffer cache.

- Set my KEEP pool to 1T.
- Altered the table H_LINEITEM to the KEEP pool

Scan a big table with PQ

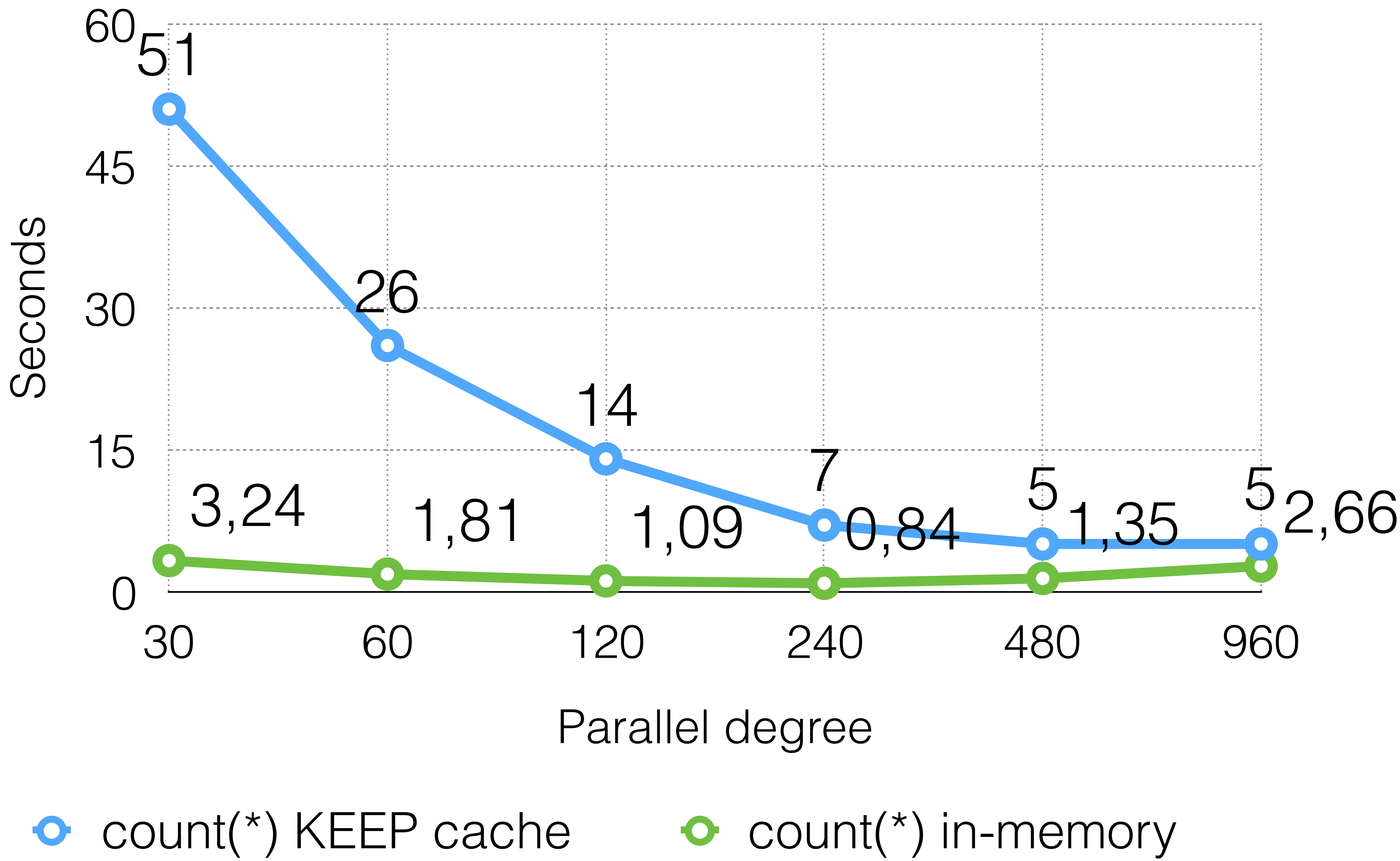


Scan a big table with PQ

- Near linear scaling up to core count.
  - All slaves busy, no producer/consumer model.

- Let's see what the in-memory option can do!
  - Added 1T in-memory pool.
  - Copied table for in-memory (query high).

Scan a big table with PQ



enkitec

Scan a big table with PQ

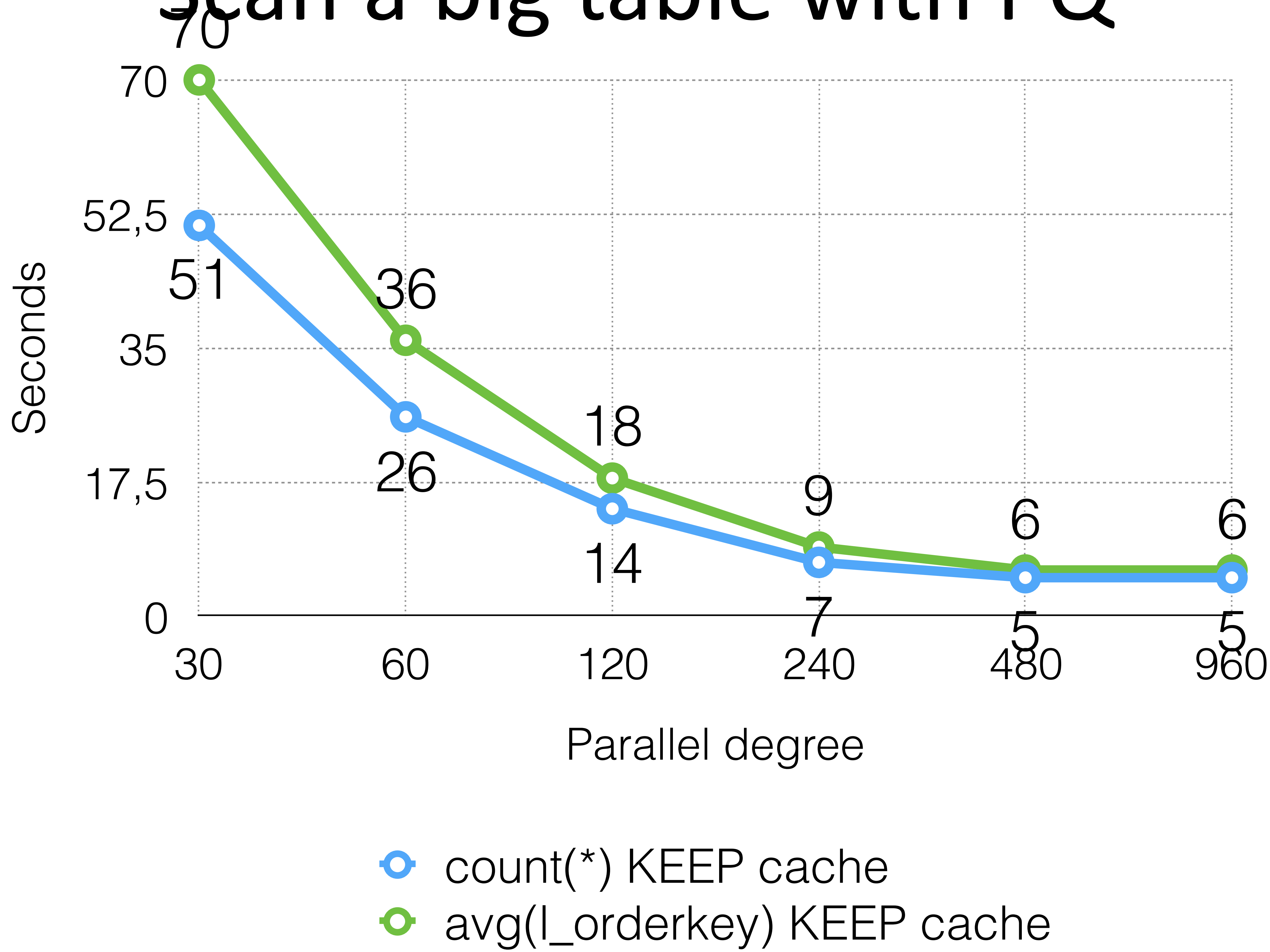
- Unbelievable performance with in-memory option.
- No such thing as magic.

Scan a big table with PQ

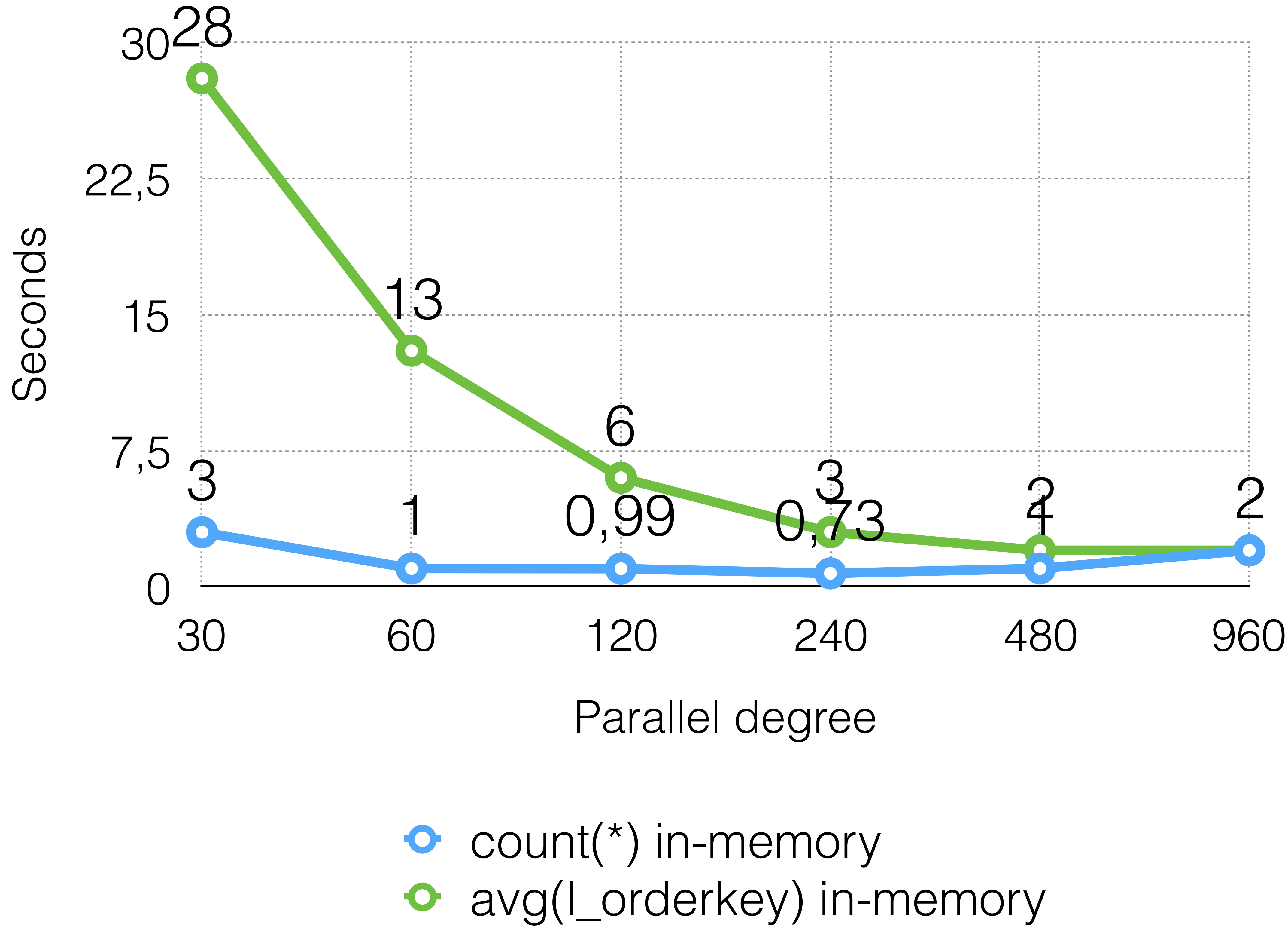
- in-memory:
  - Suspicion of HCC count(*) optimization.

- To mitigate potential pre-computed result:
  - Try different function: avg()

Scan a big table with PQ

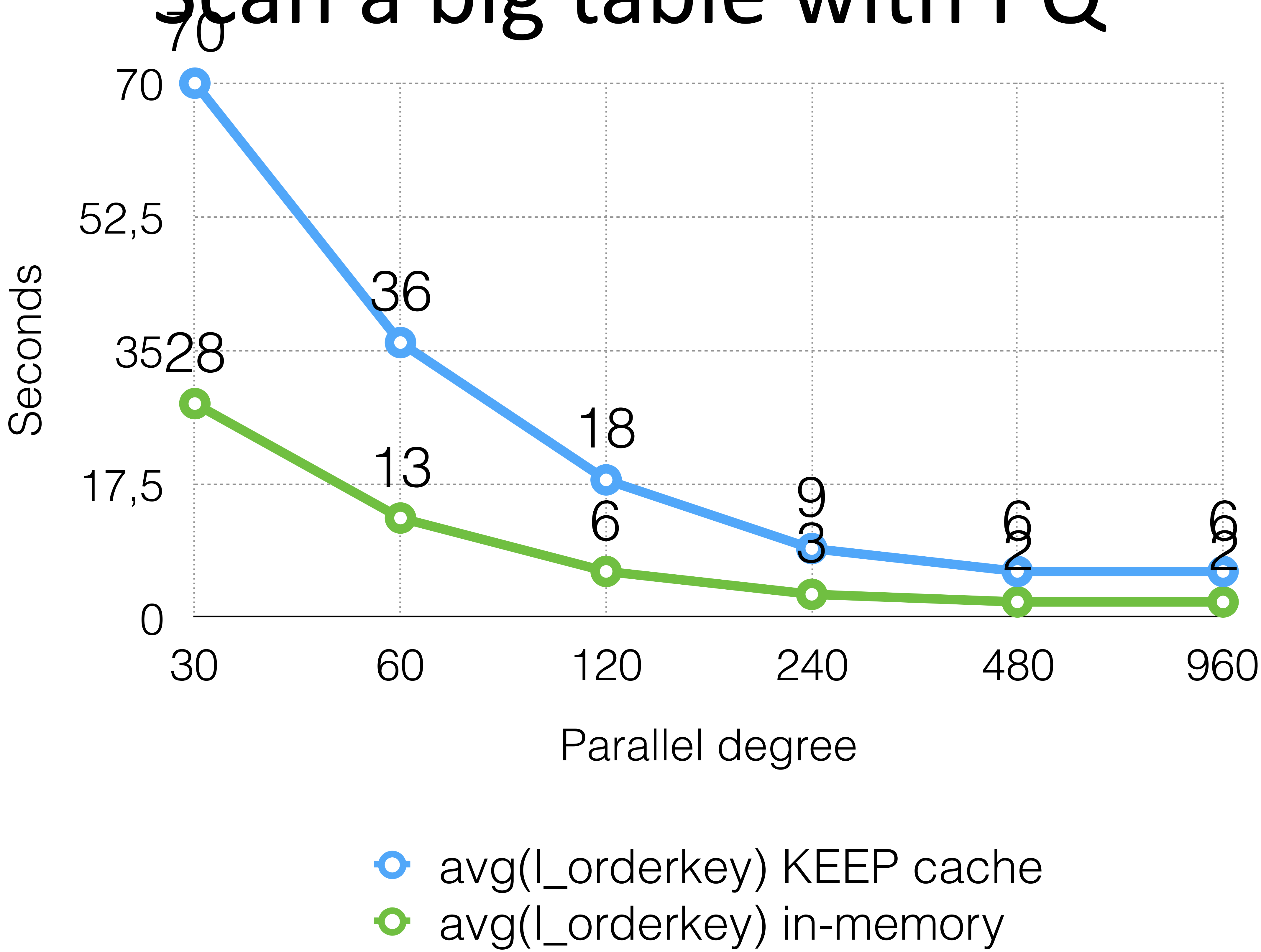


Scan a big table with PQ



enkitec

Scan a big table with PQ



enkitec

Scan a big table with PQ

- in-memory option reduces 66% of time(!!)
- No vector (SIMD) processing tested.

Conclusion

- Increase in NUMA nodes increases random memory access latency.
- However, core count increases processing capacity too.

- Memory placement is important with NUMA count > 4 - 8.

Conclusion

- UV 300 has constant distance via HARP.
  - This means adding NUMA nodes scales linearly.
- Oracle OLTP processing takes very efficient usage of on-die caches (L1/2/3).
- Disable NUMA on low socket count (>=4) servers for Oracle, unless you can prove it benefits.

Conclusion

- PQ can be turned to cached reads by:
  - Setting the NOCACHE attribute to CACHE*.
  - Moving a table in the KEEP pool.
- With no cache fixation, Oracle might restrict blocks in cache to 1/3rd of the total.

- The in-memory parallel query option scans almost all blocks into cache.

Conclusion

- UV 300 processing works well together with PQ.
- In-memory compression has pre-computed count(*) optimization.
- There is overhead involved in PQ processing.

Oracle Findings

- The SGA huge pages are initialised by a single process.
  - Initialising a 10T SGA takes a significant time.
- Shutdown normal/immediate never finishes.
  - PMON failed to acquire latch error.
  - Process shutting down the instance continuously running through /proc/stat.

enkitec

Oracle Findings

- Heavy parallel scan on 750G table in memory.
  - Took ~ 5 seconds right after startup.
  - Took ~ 20 seconds after 14 days uptime.
  - 5 seconds was restored after bounce.
  - Uncertain what the cause is…

Q & A