RED HAT SUMMIT

10 YEARS *and counting*

SAN FRANCISCO | APRIL 14-17, 2014

Automatic NUMA Balancing

Rik van Riel, Principal Software Engineer, Red Hat
Vinod Chegu, Master Technologist, HP

Automatic NUMA Balancing Agenda

- What is NUMA, anyway?
- Automatic NUMA balancing internals
- Automatic NUMA balancing performance
  - What workloads benefit from manual NUMA tuning
- NUMA tools
- Future developments
- Conclusions

Introduction to NUMA

What is NUMA, anyway?

What is NUMA, anyway?

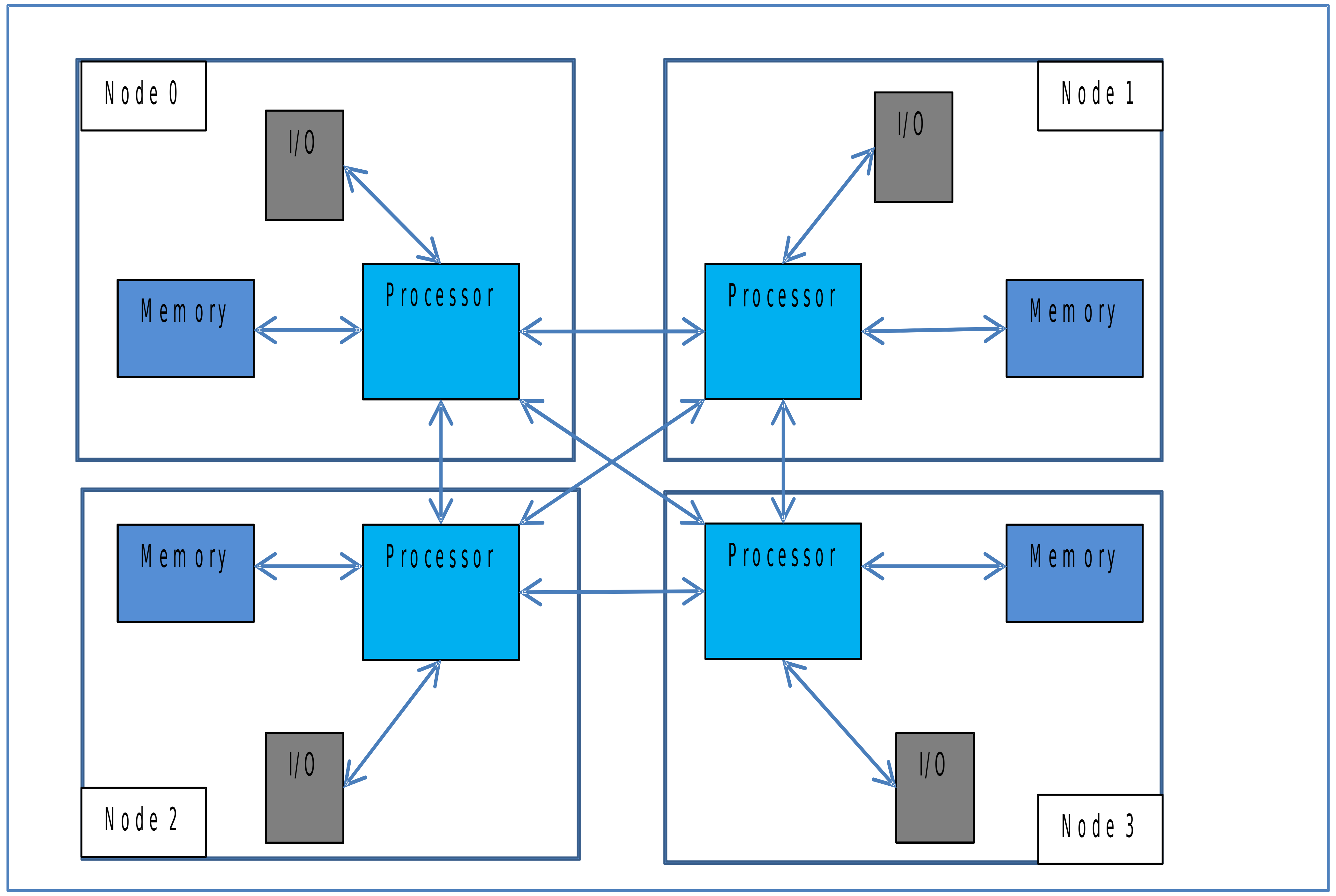
- Non Uniform Memory Access
- Multiple physical CPUs in a system
- Each CPU has memory attached to it
  - Local memory, fast
- Each CPU can access other CPU's memory, too
  - Remote memory, slower

NUMA terminology

- Node
  - A physical CPU and attached memory
  - Could be multiple CPUs (with off-chip memory controller)
- Interconnect
  - Bus connecting the various nodes together
  - Generally faster than memory bandwidth of a single node
  - Can get overwhelmed by traffic from many nodes

HP Proliant DL580 Gen8 – NUMA topology

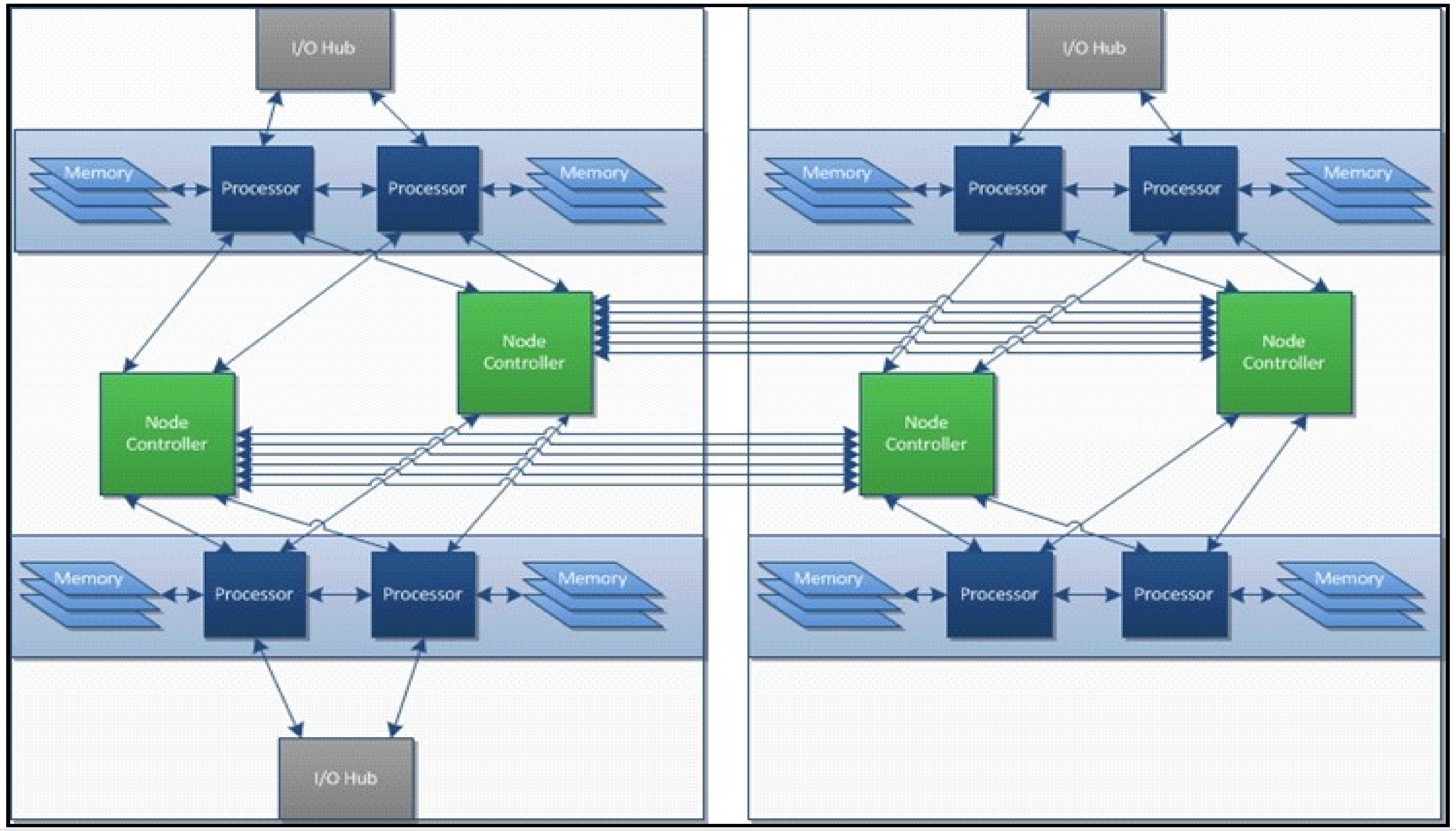
4-socket Ivy Bridge EX processor



```
# numactl -H
available: 4 nodes (0-3)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
node 0 size: 262040 MB
node 0 free: 249261 MB
node 1 cpus: 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
node 1 size: 262144 MB
node 1 free: 252060 MB
node 2 cpus: 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
node 2 size: 262144 MB
node 2 free: 250441 MB
node 3 cpus: 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
node 3 size: 262144 MB
node 3 free: 250080 MB
node distances:
node   0   1   2   3
  0:  10  21  21  21
  1:  21  10  21  21
  2:  21  21  10  21
  3:  21  21  21  10
```

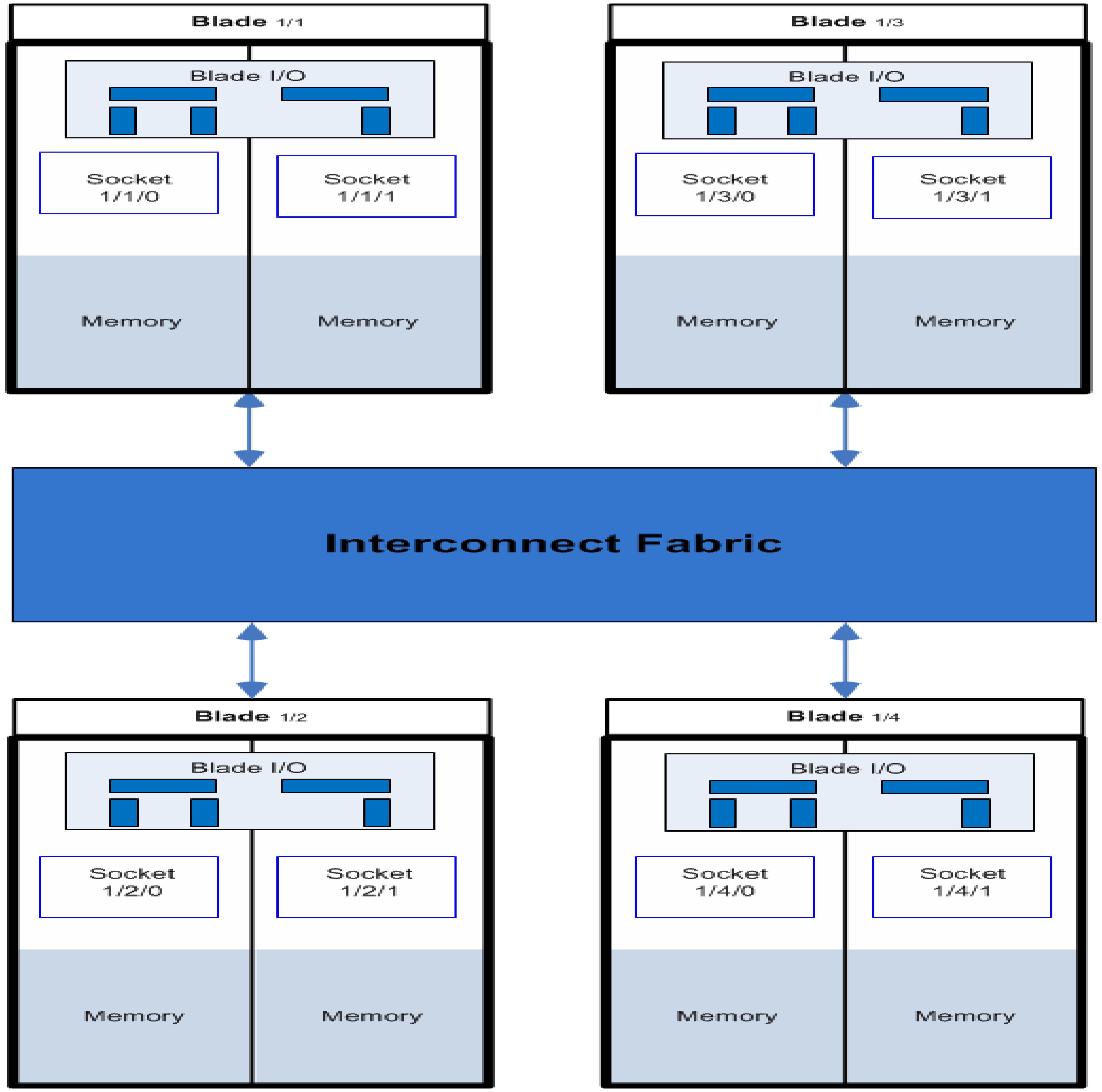
HP Proliant DL980 G7 – NUMA topology

8-socket Westmere EX processor



```
# numactl -H
available: 8 nodes (0-7)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9
node 0 size: 262133 MB
node 0 free: 250463 MB
node 1 cpus: 10 11 12 13 14 15 16 17 18 19
node 1 size: 262144 MB
node 1 free: 256316 MB
node 2 cpus: 20 21 22 23 24 25 26 27 28 29
node 2 size: 262144 MB
node 2 free: 256439 MB
node 3 cpus: 30 31 32 33 34 35 36 37 38 39
node 3 size: 262144 MB
node 3 free: 255403 MB
node 4 cpus: 40 41 42 43 44 45 46 47 48 49
node 4 size: 262144 MB
node 4 free: 256546 MB
node 5 cpus: 50 51 52 53 54 55 56 57 58 59
node 5 size: 262144 MB
node 5 free: 256036 MB
node 6 cpus: 60 61 62 63 64 65 66 67 68 69
node 6 size: 262144 MB
node 6 free: 256468 MB
node 7 cpus: 70 71 72 73 74 75 76 77 78 79
node 7 size: 262144 MB
node 7 free: 255232 MB
node distances:
node   0   1   2   3   4   5   6   7
  0:  10  12  17  17  19  19  19  19
  1:  12  10  17  17  19  19  19  19
  2:  17  17  10  12  19  19  19  19
  3:  17  17  12  10  19  19  19  19
  4:  19  19  19  19  10  12  17  17
  5:  19  19  19  19  12  10  17  17
  6:  19  19  19  19  17  17  10  12
  7:  19  19  19  19  17  17  12  10
```

8 (or 16) socket Ivy Bridge EX *prototype* server – NUMA topology



```
# numactl -H
available: 8 nodes (0-7)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
node 0 size: 130956 MB
node 0 free: 125414 MB
node 1 cpus: 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
node 1 size: 131071 MB
node 1 free: 126712 MB
node 2 cpus: 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
node 2 size: 131072 MB
node 2 free: 126612 MB
node 3 cpus: 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
node 3 size: 131072 MB
node 3 free: 125383 MB
node 4 cpus: 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
node 4 size: 131072 MB
node 4 free: 126479 MB
node 5 cpus: 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89
node 5 size: 131072 MB
node 5 free: 125298 MB
node 6 cpus: 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104
node 6 size: 131072 MB
node 6 free: 126913 MB
node 7 cpus: 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
node 7 size: 131072 MB
node 7 free: 124509 MB
node distances:
node   0   1   2   3   4   5   6   7
  0:  10  16  30  30  30  30  30  30
  1:  16  10  30  30  30  30  30  30
  2:  30  30  10  16  30  30  30  30
  3:  30  30  16  10  30  30  30  30
  4:  30  30  30  30  10  16  30  30
  5:  30  30  30  30  16  10  30  30
  6:  30  30  30  30  30  30  10  16
  7:  30  30  30  30  30  30  16  10
```

NUMA performance considerations

- NUMA performance penalties from two main sources
  - Higher latency of accessing remote memory
  - Interconnect contention
- Processor threads and cores share resources
  - Execution units (between HT threads)
  - Cache (between threads and cores)

Automatic NUMA balancing strategies

- CPU follows memory
  - Reschedule tasks on same nodes as memory
- Memory follows CPU
  - Copy memory pages to same nodes as tasks/threads
- Both strategies are used by automatic NUMA balancing
  - Various mechanisms involved
  - Lots of interesting corner cases...

Automatic NUMA Balancing

Internals

Automatic NUMA balancing internals

- NUMA hinting page faults
- NUMA page migration
- Task grouping
- Fault statistics
- Task placement
- Pseudo-interleaving

NUMA hinting page faults

- Periodically, each task's memory is unmapped
  - Period based on run time, and NUMA locality
  - Unmapped “a little bit” at a time (chunks of 256MB)
  - Page table set to “no access permission” marked as NUMA pte
- Page faults generated as task accesses memory
  - Used to track the location of memory a task uses
    - Task may also have unused memory “just sitting around”
  - NUMA faults also drive NUMA page migration

NUMA page migration

- NUMA page faults are relatively cheap
- Page migration is much more expensive
  - ... but so is having task memory on the “wrong node”
- Quadratic filter: only migrate if page is accessed twice
  - From same NUMA node, or
  - By the same task
  - CPU number & low bits of pid in page struct
- Page is migrated to where the task is running

Fault statistics

- Fault statistics are used to place tasks (cpu-follows-memory)
- Statistics kept per task
- “Where is the memory this task is accessing?”
  - NUMA page faults counter per NUMA node
  - After a NUMA fault, account the page location
    - If the page was migrated, account the new location
  - Kept as a floating average

Types of NUMA faults

- Locality
  - “Local fault” - memory on same node as CPU
  - “Remote fault” - memory on different node than CPU
- Private vs shared
  - “Private fault” - memory accessed by same task twice in a row
  - “Shared fault” - memory accessed by different task than last time

Fault statistics example

numa_faults	Task A	Task B
Node 0	0	1027
Node 1	83	29
Node 2	915	17
Node 3	4	31

Task placement

- Best place to run a task
  - Where most of its memory accesses happen

Task placement

- Best place to run a task
  - Where most of its memory accesses happen
- It is not that simple
  - Tasks may share memory
    - Some private accesses, some shared accesses
    - 60% private, 40% shared is possible – group tasks together for best performance
  - Tasks with memory on the node may have more threads than can run in one node's CPU cores
  - Load balancer may have spread threads across more physical CPUs
    - Take advantage of more CPU cache

Task placement constraints

- NUMA task placement may not create a load imbalance
  - The load balancer would move something else
  - Conflict can lead to tasks “bouncing around the system”
    - Bad locality
    - Lots of NUMA page migrations
- NUMA task placement may
  - Exchange tasks between nodes
  - Move a task to an idle CPU if no imbalance is created

Task placement algorithm

- For task a, check each NUMA node N
  - Check whether node N is better than task a's current node (C)
    - Task A has a larger fraction of memory accesses on node N, than on current node C
    - Score is the difference of fractions
  - If so, check all CPUs on node N
    - Is the current task (t) on CPU better off on node C?
    - Is the CPU idle and can we move task a to the CPU?
    - Is the benefit of moving task a to node N larger than the downside of moving task t to node C?
  - For the CPU with the best score, move task a (and task t, to node C).

Task placement examples

NODE	CPU	TASK
0	0	a
0	1	t
1	2	(idle)
1	3	(idle)

Fault statistics	Task a	Task t
NODE 0	30% (*)	60% (*)
NODE 1	70%	40%

- Moving task a to node 1: 40% improvement
- Moving task a to node 1 removes a load imbalance
- Moving task a to an idle CPU on node 1 is desirable

Task placement examples

NODE	CPU	TASK
0	0	a
0	1	(idle)
1	2	t
1	3	(idle)

Fault statistics	Task a	Task t
NODE 0	30% (*)	60%
NODE 1	70%	40% (*)

- Moving task a to node 1: 40% improvement
- Moving task t to node 0: 20% improvement
- Exchanging tasks a & t is desirable

Task placement examples

NODE	CPU	TASK
0	0	a
0	1	(idle)
1	2	t
1	3	(idle)

Fault statistics	Task a	Task t
NODE 0	30% (*)	40%
NODE 1	70%	60% (*)

- Moving task a to node 1: 40% improvement
- Moving task t to node 0: 20% worse
- Exchanging tasks a & t: overall a 20% improvement ==> do it

Task placement examples

NODE	CPU	TASK
0	0	a
0	1	(idle)
1	2	t
1	3	(idle)

Fault statistics	Task a	Task t
NODE 0	30% (*)	20%
NODE 1	70%	80% (*)

- Moving task a to node 1: 40% improvement
- Moving task t to node 0: 60% worse
- Exchanging tasks a & t: overall 20% worse ==> leave things alone

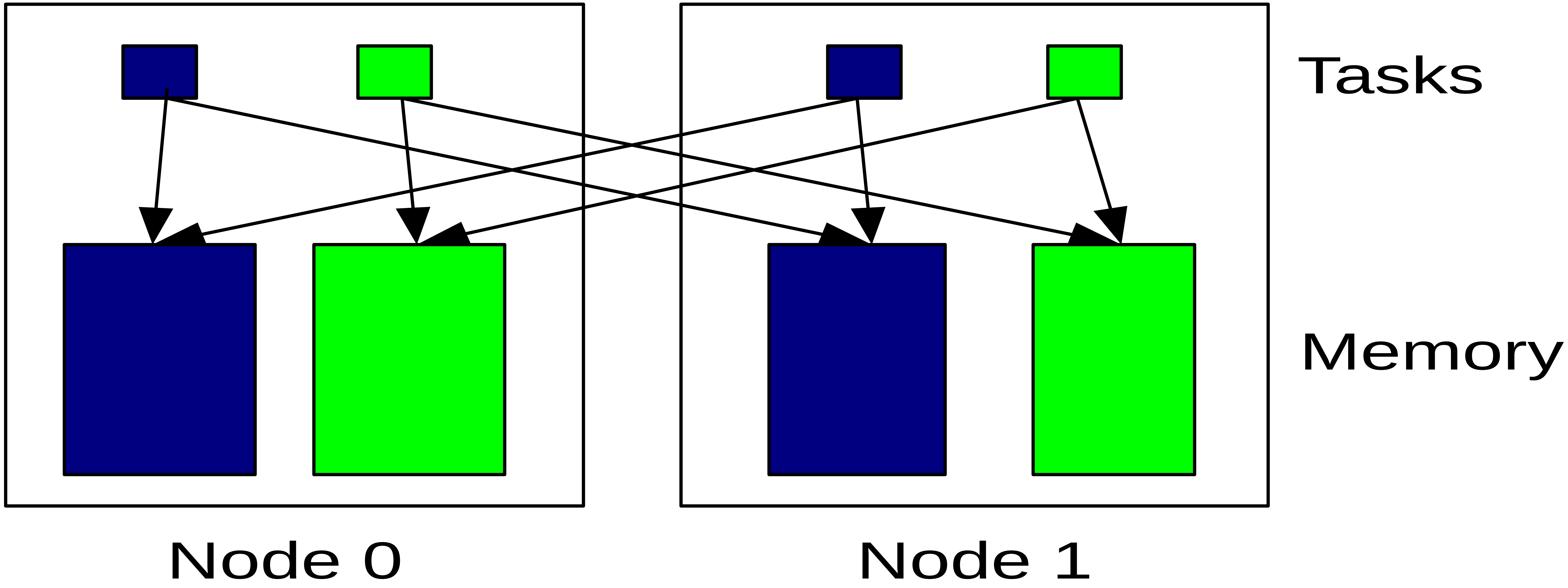
Task grouping

- Multiple tasks can access the same memory
  - Threads in a large multi-threaded process (JVM, virtual machine, ...)
  - Processes using shared memory segment (eg. Database)
- Use CPU num & pid in struct page to detect shared memory
  - At NUMA fault time, check CPU where page was last faulted
  - Group tasks together in numa_group, if PID matches
- Grouping related tasks improves NUMA task placement
  - Only group truly related tasks
  - Only group on write faults, ignore shared libraries like libc.so

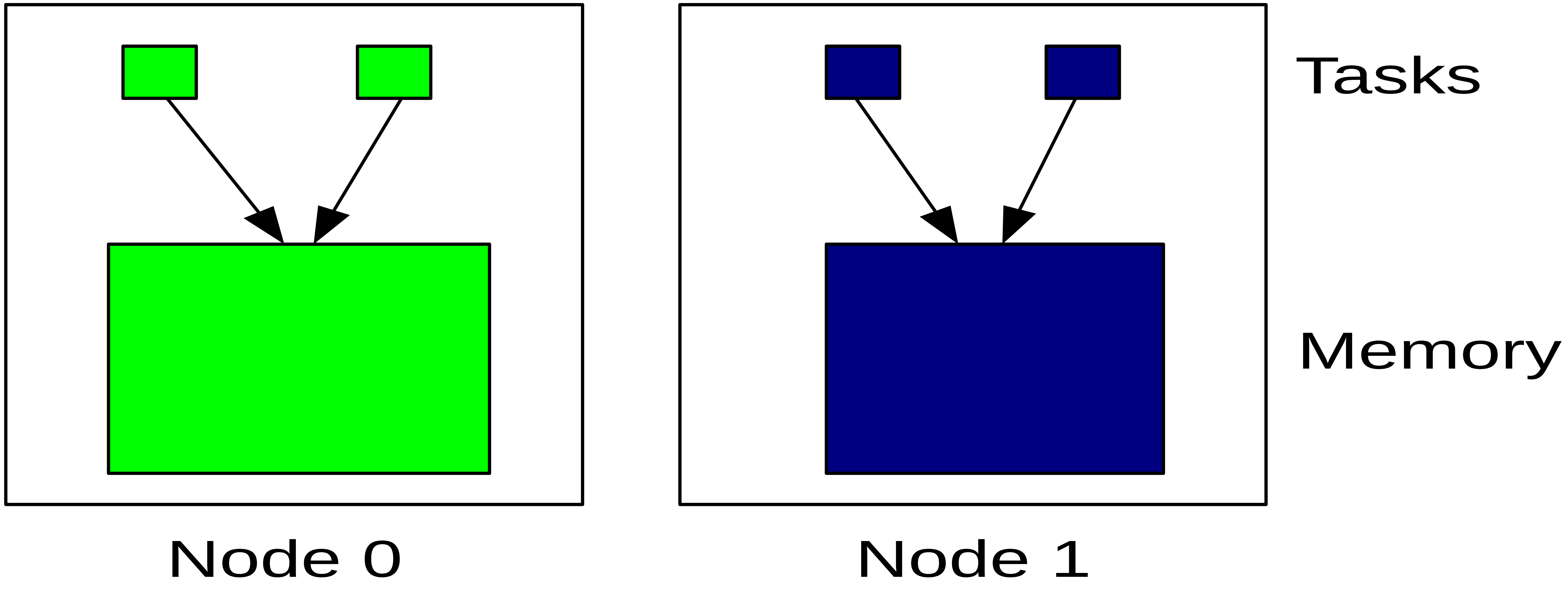
Task grouping & task placement

- Group stats are the sum of the NUMA fault stats for tasks in group
- Task placement code similar to before
- If a task belongs to a numa_group, use the numa_group stats for comparison instead of the task stats
  - Pulls groups together, for more efficient access to shared memory
- When both compared tasks belong to the same numa_group
  - Use task stats, since group numbers are the same
  - Efficient placement of tasks within a group

Task grouping & placement example



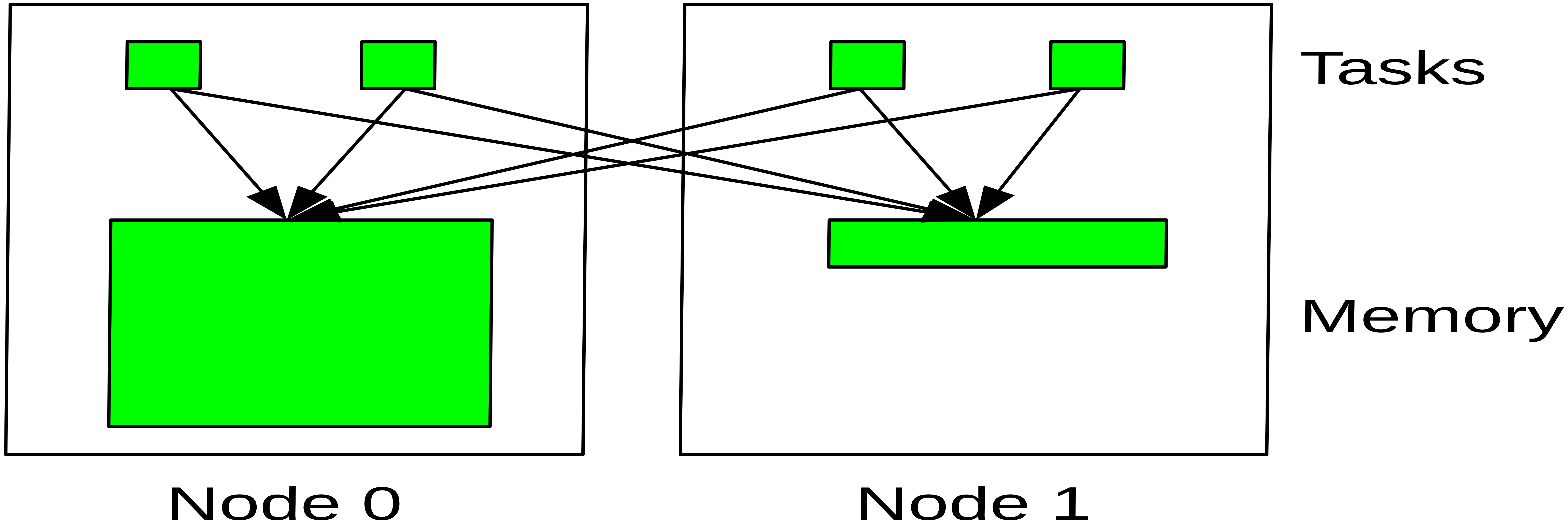
Task grouping & placement example



Pseudo-interleaving

- Sometimes one workload spans multiple nodes
  - More threads running than one node has CPU cores
  - Spread out by the load balancer
- Goals
  - Maximize memory bandwidth available to workload
  - Keep private memory local to tasks using it
  - Minimize number of page migrations

Pseudo-interleaving problem

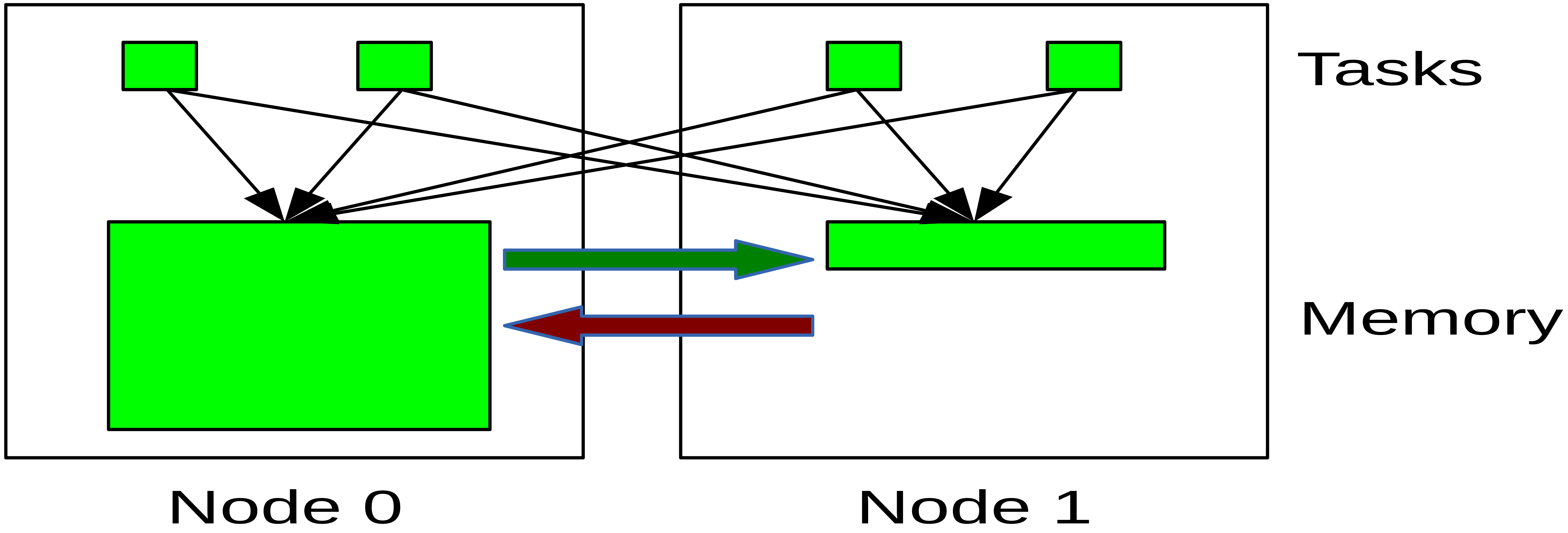


- Most memory on node 0, sub-optimal use of memory bandwidth
- How to fix? Spread out the memory more evenly...

Pseudo-interleaving algorithm

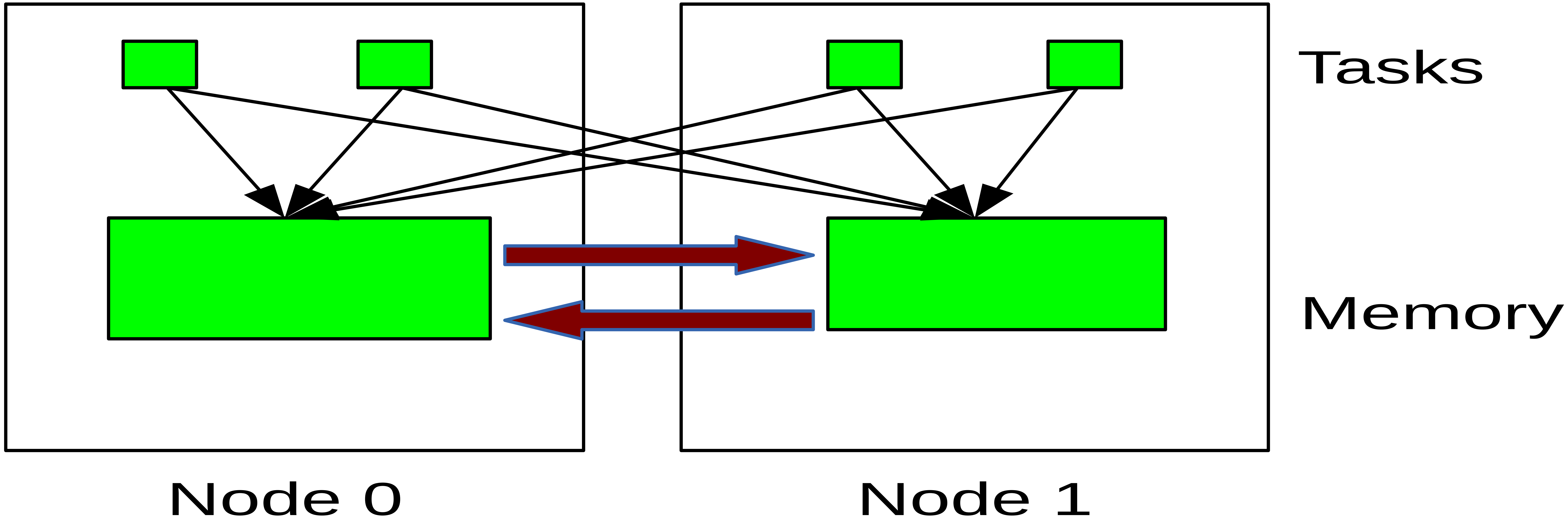
- Determine nodes where workload is actively running
  - CPU time used & NUMA faults
- Always allow private faults (same task) to migrate pages
- Allow shared faults to migrate pages only from a more heavily used node, to a less heavily used node
- Block NUMA page migration on shared faults from one node to another node that is equally or more heavily used

Pseudo-interleaving solution



- Allow NUMA migration on private faults
- Allow NUMA migration from more used, to lesser used node

Pseudo-interleaving converged state



- Nodes are equally used, maximizing memory bandwidth
- NUMA page migration only on private faults
- NUMA page migration on shared faults is avoided

Automatic NUMA Placement Performance

Show me the numbers!

Evaluation of Automatic NUMA balancing – *Status update*

Goal : Study the impact of Automatic NUMA Balancing on out-of-the-box performance compared to no NUMA tuning and manual NUMA pinning

- On bare-metal and KVM guests
- Using a variety of synthetic workloads*:
    - 2 Java workloads
        - SPECjbb2005 used as a workload
        - Multi-JVM server workload
    - Database
        - A *synthetic* DSS workload (using tmpfs)
        - A *synthetic* OLTP workload in KVM (using virtio)

*** Note: These sample workloads were used for relative performance comparisons. This is not an official benchmarking exercise!***

Experiments with bare-metal

- Platforms used :
  - HP Proliant DL580 Gen 8 - 4-socket Ivy Bridge EX server
  - 8-socket Ivy Bridge EX *prototype* server.
- Misc. settings:
  - Hyper-threading off, THP enabled & cstate set to 1
- Configurations :
  - Baseline : No manual pinning of the workload, No Automatic NUMA balancing
  - Pinned : Manual (numactl) pinning of the workload
  - Automatic NUMA balancing : default, out-of-the box setting.

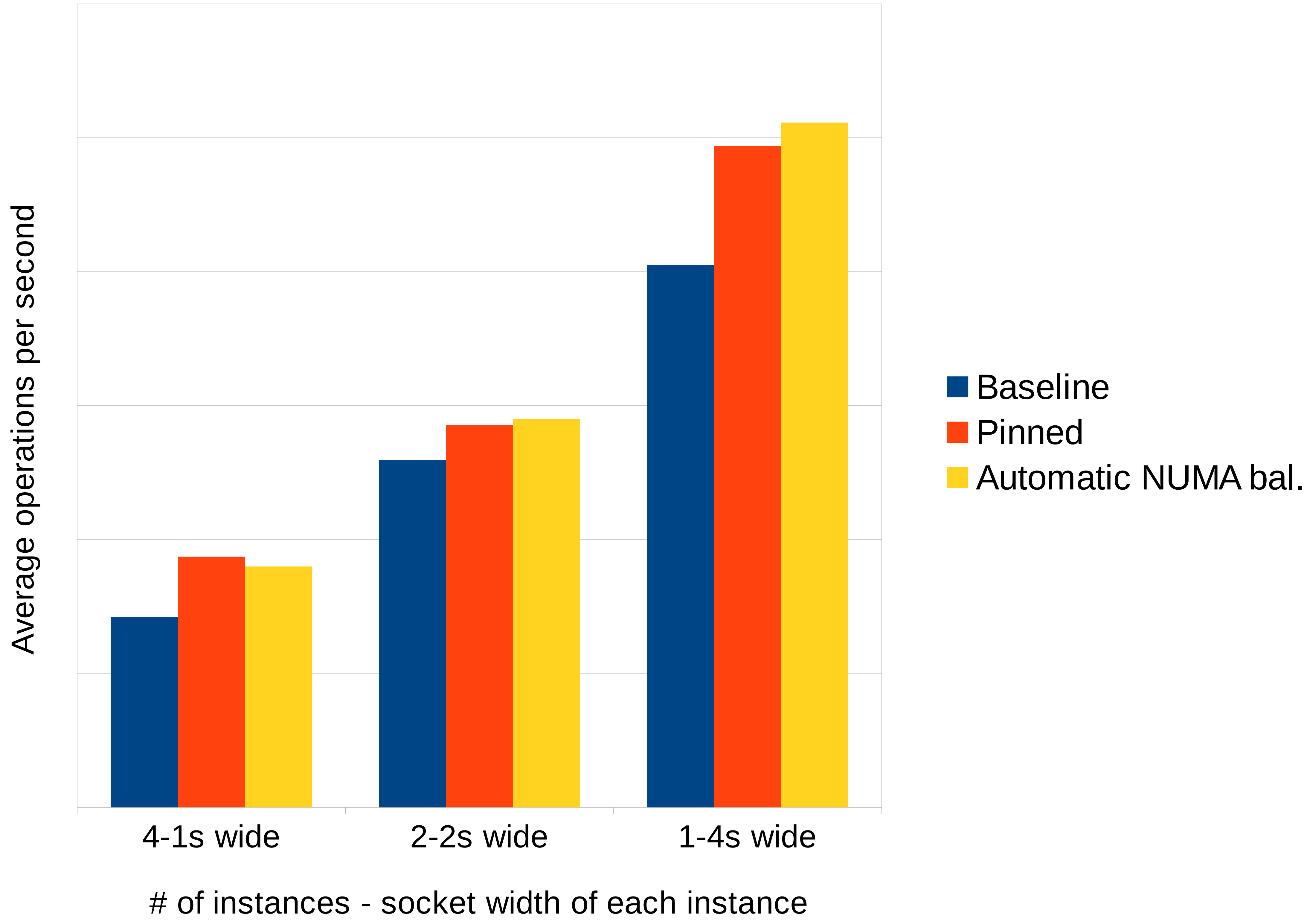
Tools

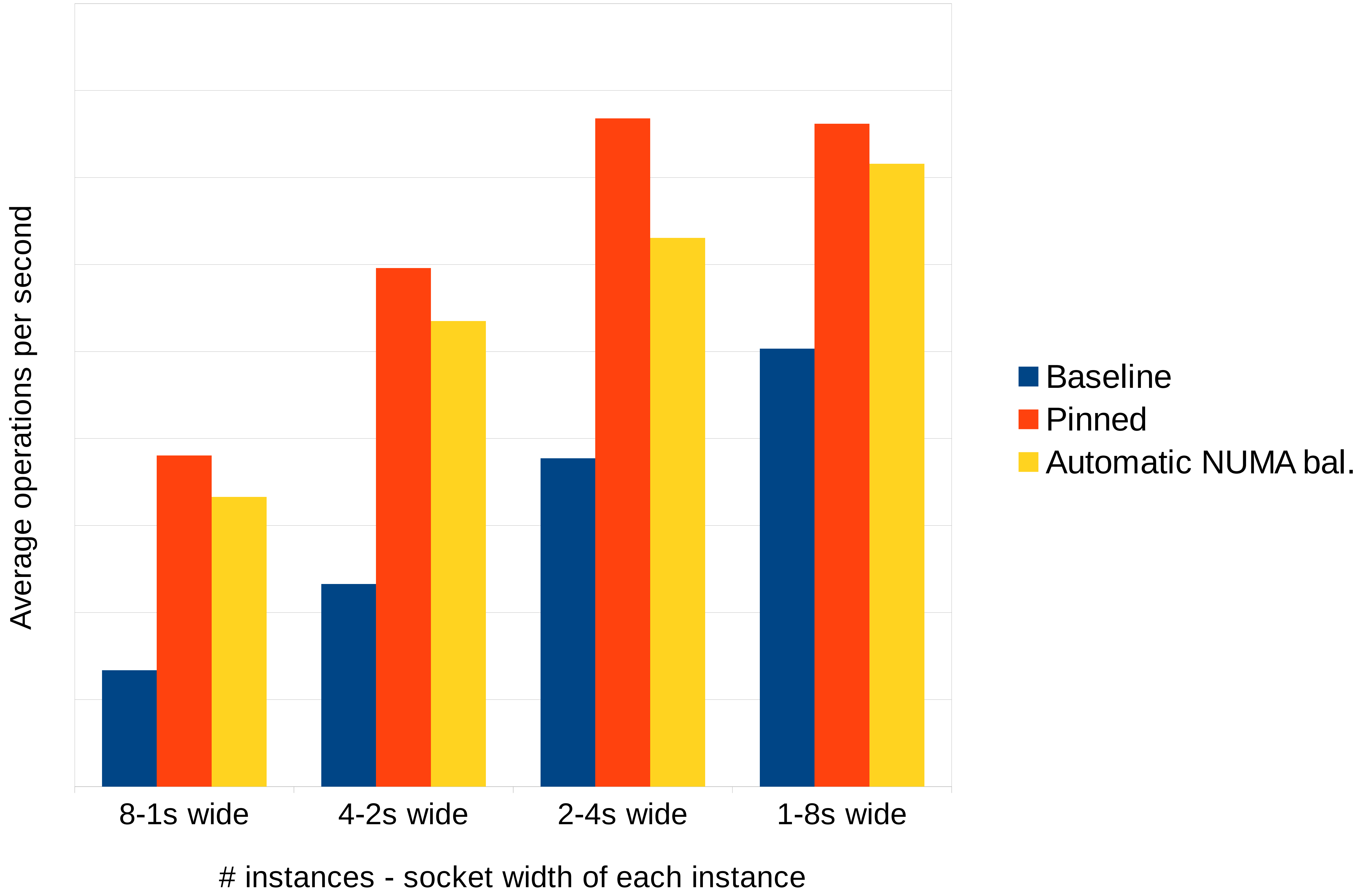
- Status of Automatic NUMA balancing
  - Use sysctl to check/disable/enable “kernel.numa_balancing”
  - Default is set to enabled.
- /proc/vmstat
  - Indication of # of pages migrated & # of pages that failed to migrate
- /proc/zoneinfo
  - Indication of remote vs. local NUMA accesses
- numastat
  - Indication of which nodes are contributing to the running tasks.
- Miscellaneous upstream tools : e.g. numatop

SPECjbb2005 - bare-metal

(4-socket IVY-EX server vs. 8-socket IVY-EX *prototype* server)

1s wide = 15 warehouse threads, 2s wide = 30 warehouse threads; 4s wide = 60 warehouse threads, 8s wide = 120 warehouse threads





Pinned case was ~10-25% better than Baseline case Automatic NUMA balancing case & the Pinned case were pretty close (+/- 4%).

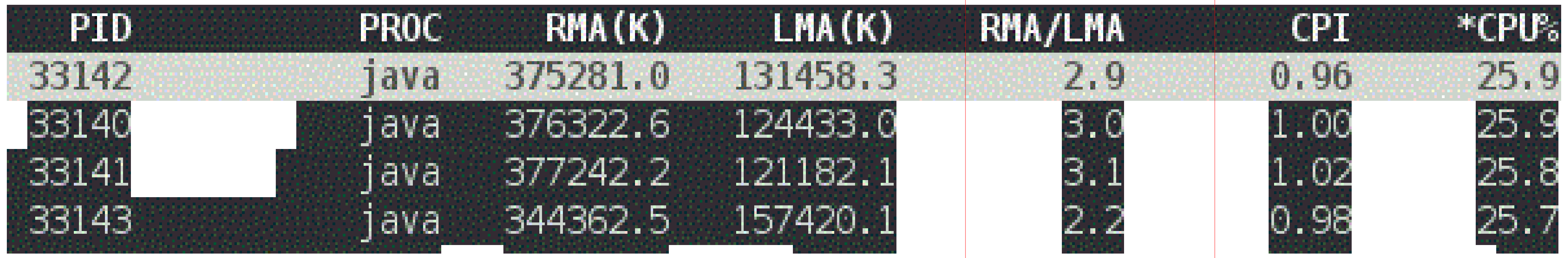
Pinned case was ~ 34-65% better than the Baseline case. Delta between Automatic NUMA balancing case & the Pinned case was as high as ~18+%

Remote vs. local memory access (RMA/LMA samples)*

(Workload : Multiple 1 socket-wide instances of SPECjbb2005)

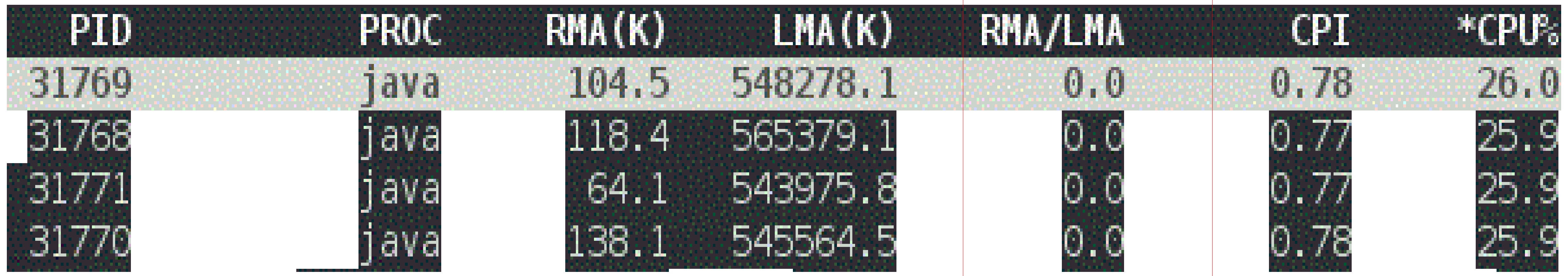
4-socket IVY-EX server

Baseline



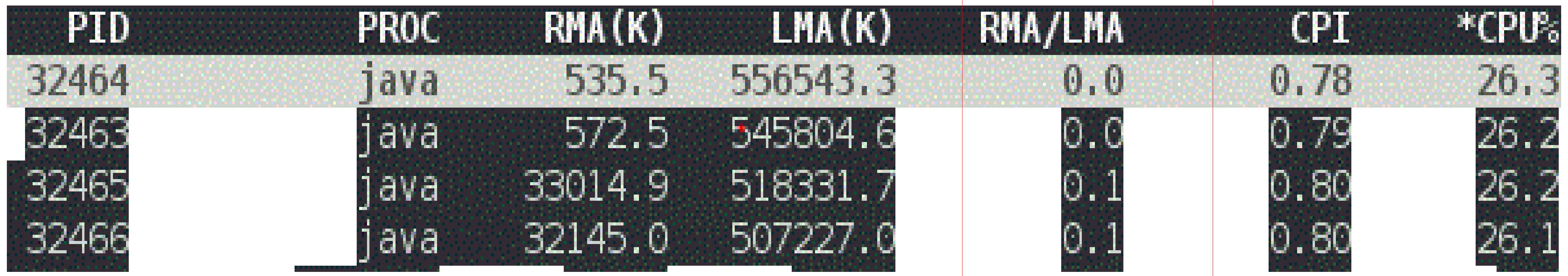
```
  PID          PROC     RMA(K)    LMA(K)   RMA/LMA     CPI    *CPU%
33142          java   375281.0  131458.3       2.9    0.96     25.9
33140          java   376322.6  124433.0       3.0    1.00     25.9
33141          java   377242.2  121182.1       3.1    1.02     25.8
33143          java   344362.5  157420.1       2.2    0.98     25.7
```

Pinned



```
  PID          PROC     RMA(K)    LMA(K)   RMA/LMA     CPI    *CPU%
31769          java      104.5  548278.1       0.0    0.78     26.0
31768          java      118.4  565379.1       0.0    0.77     25.9
31771          java       64.1  543975.8       0.0    0.77     25.9
31770          java      138.1  545564.5       0.0    0.78     25.9
```

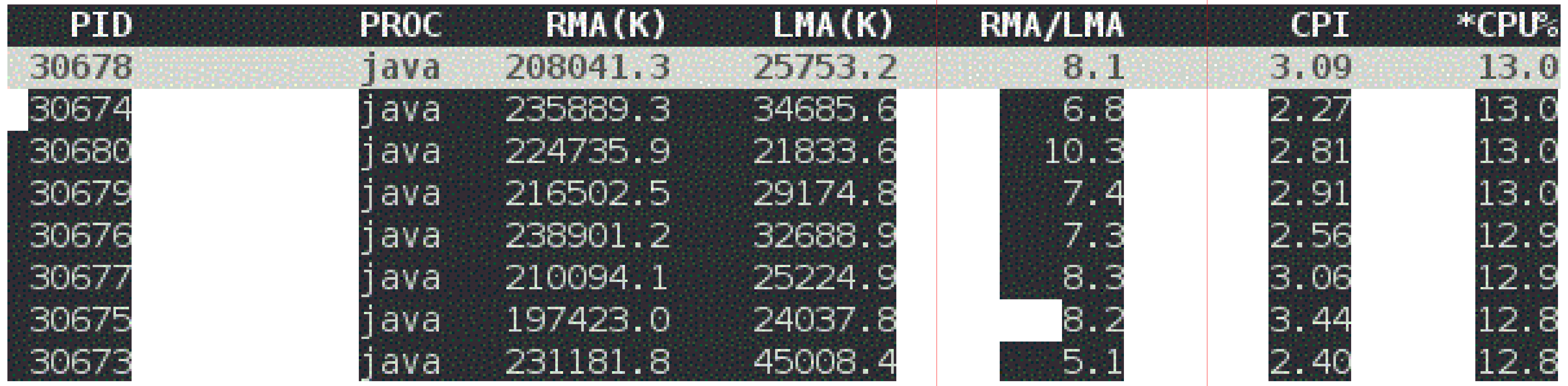
Automatic NUMA balancing



```
  PID          PROC     RMA(K)    LMA(K)   RMA/LMA     CPI    *CPU%
32464          java      535.5  556543.3       0.0    0.78     26.3
32463          java      572.5  545804.6       0.0    0.79     26.2
32465          java    33014.9  518331.7       0.1    0.80     26.2
32466          java    32145.0  507227.0       0.1    0.80     26.1
```

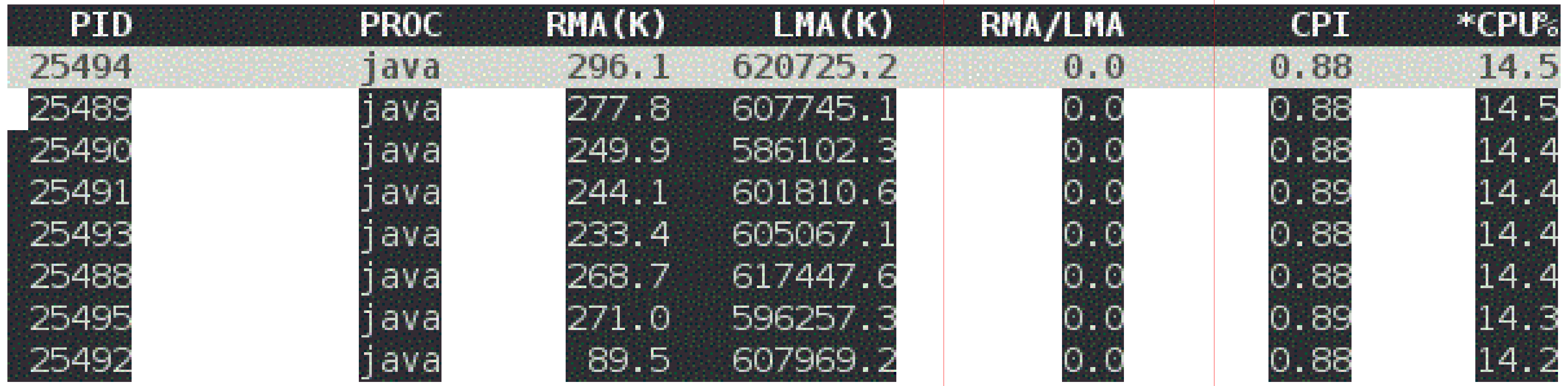
8-socket IVY-EX prototype server

Baseline



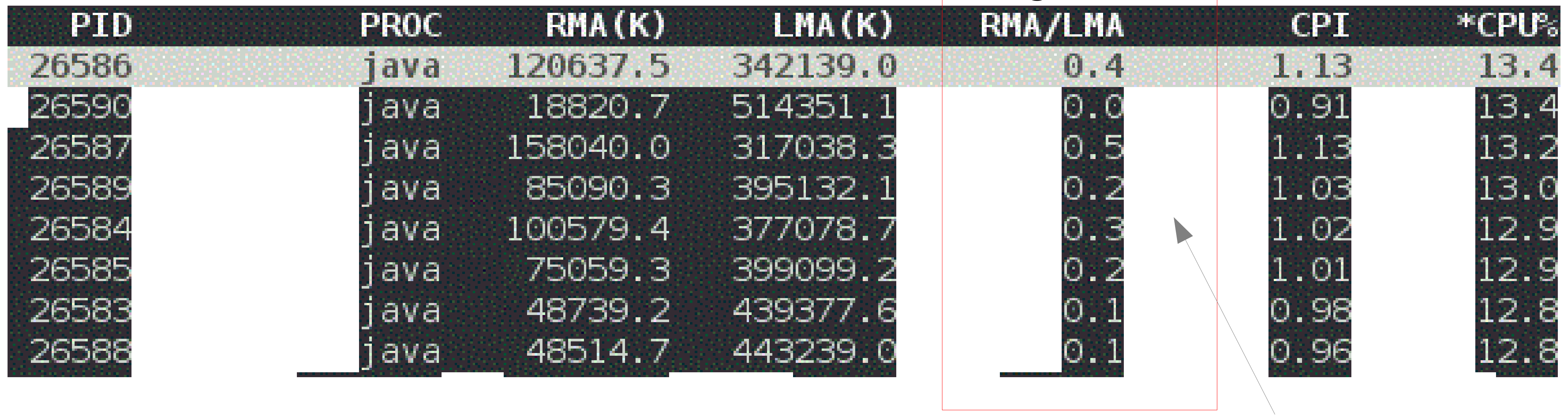
```
  PID          PROC     RMA(K)    LMA(K)   RMA/LMA     CPI    *CPU%
30678          java   208041.3   25753.2       8.1    3.09     13.0
30674          java   235889.3   34685.6       6.8    2.27     13.0
30680          java   224735.9   21833.6      10.3    2.81     13.0
30679          java   216502.5   29174.8       7.4    2.91     13.0
30676          java   238901.2   32688.9       7.3    2.56     12.9
30677          java   210094.1   25224.9       8.3    3.06     12.9
30675          java   197423.0   24037.8       8.2    3.44     12.8
30673          java   231181.8   45008.4       5.1    2.40     12.8
```

Pinned



```
  PID          PROC     RMA(K)    LMA(K)   RMA/LMA     CPI    *CPU%
25494          java      296.1  620725.2       0.0    0.88     14.5
25489          java      277.8  607745.1       0.0    0.88     14.5
25490          java      249.9  586102.3       0.0    0.88     14.4
25491          java      244.1  601810.6       0.0    0.89     14.4
25493          java      233.4  605067.1       0.0    0.88     14.4
25488          java      268.7  617447.6       0.0    0.88     14.4
25495          java      271.0  596257.3       0.0    0.89     14.3
25492          java       89.5  607969.2       0.0    0.88     14.2
```

Automatic NUMA balancing



```
  PID          PROC     RMA(K)    LMA(K)   RMA/LMA     CPI    *CPU%
26586          java   120637.5  342139.0       0.4    1.13     13.4
26590          java    18820.7  514351.1       0.0    0.91     13.4
26587          java   158040.0  317038.3       0.5    1.13     13.2
26589          java    85090.3  395132.1       0.2    1.03     13.0
26584          java   100579.4  377078.7       0.3    1.02     12.9
26585          java    75059.3  399099.2       0.2    1.01     12.9
26583          java    48739.2  439377.6       0.1    0.98     12.8
26588          java    48514.7  443239.0       0.1    0.96     12.8
```

Higher RMA/LMA

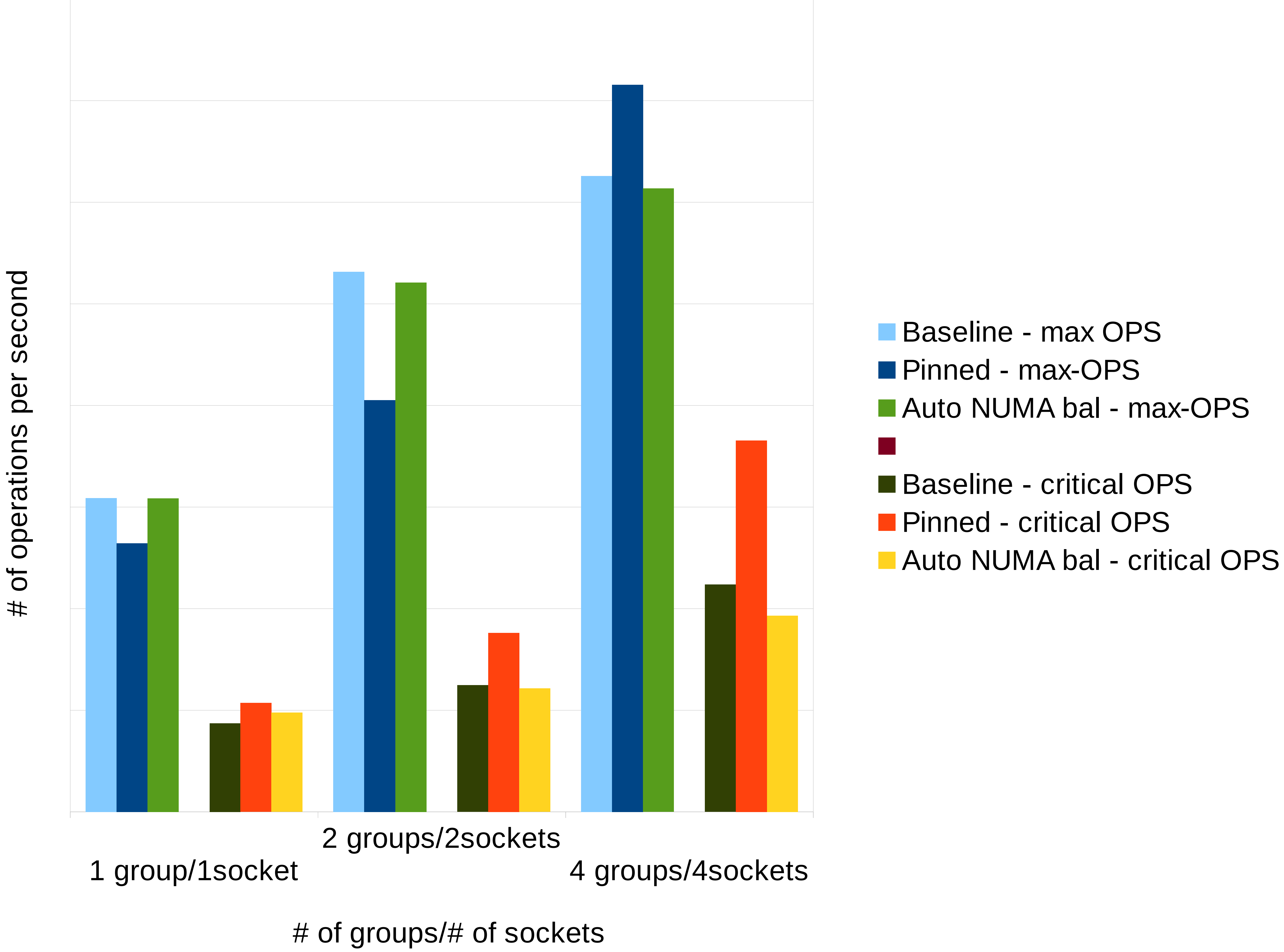
*** Courtesy numatop v1.0.2**

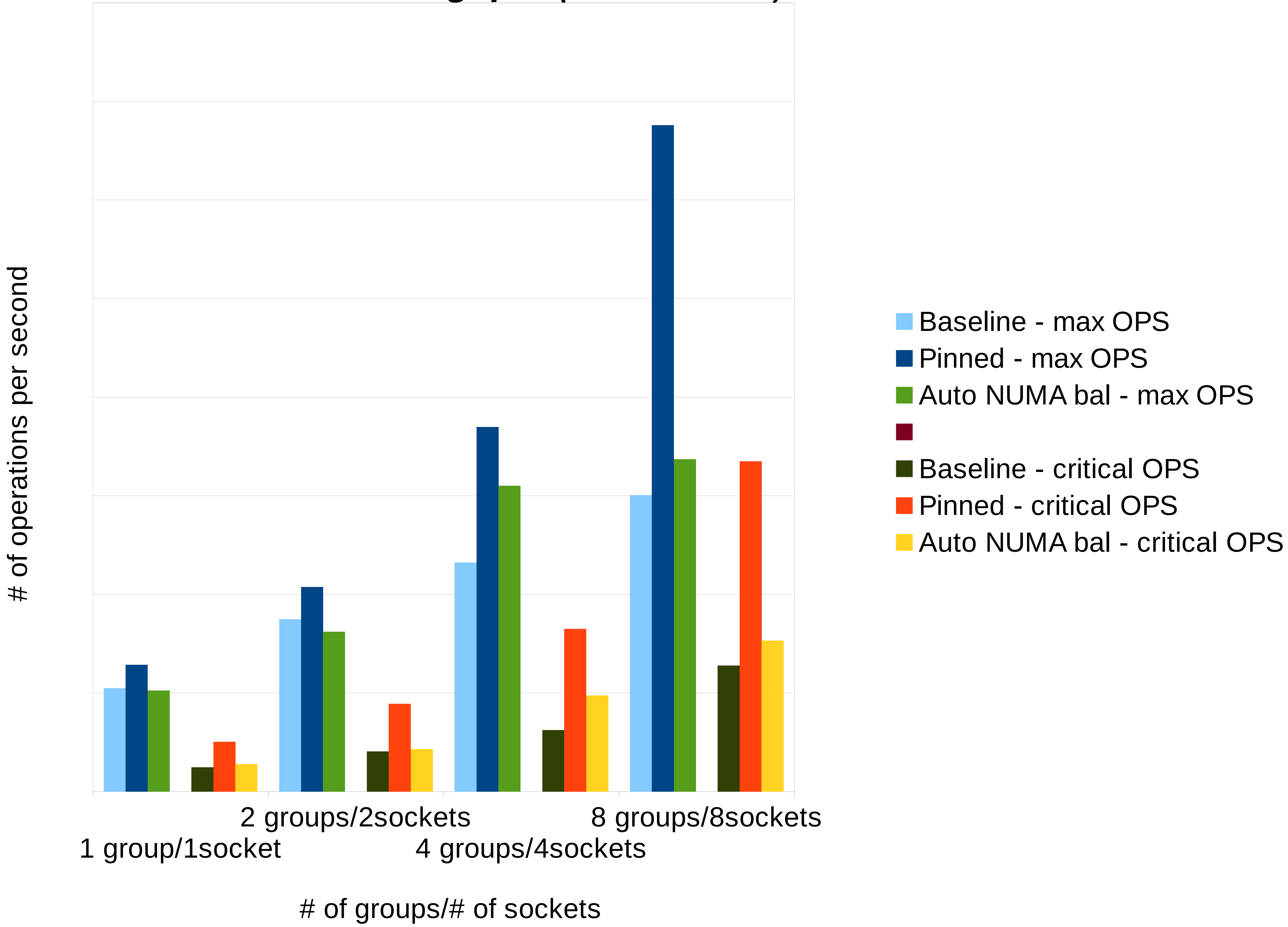
Multi-JVM server workload – bare-metal

(4-socket IVY-EX server vs. 8-socket IVY-EX *prototype* server)

Entities within each of the multiple Groups communicate with a Controller (using IPC) within the same host & the frequency of communication increases as the # of Groups increase

Two key metrics : Max throughput (max-OPS) with no constraints & Critical throughput (critical-OPS) under fixed SLA constraints

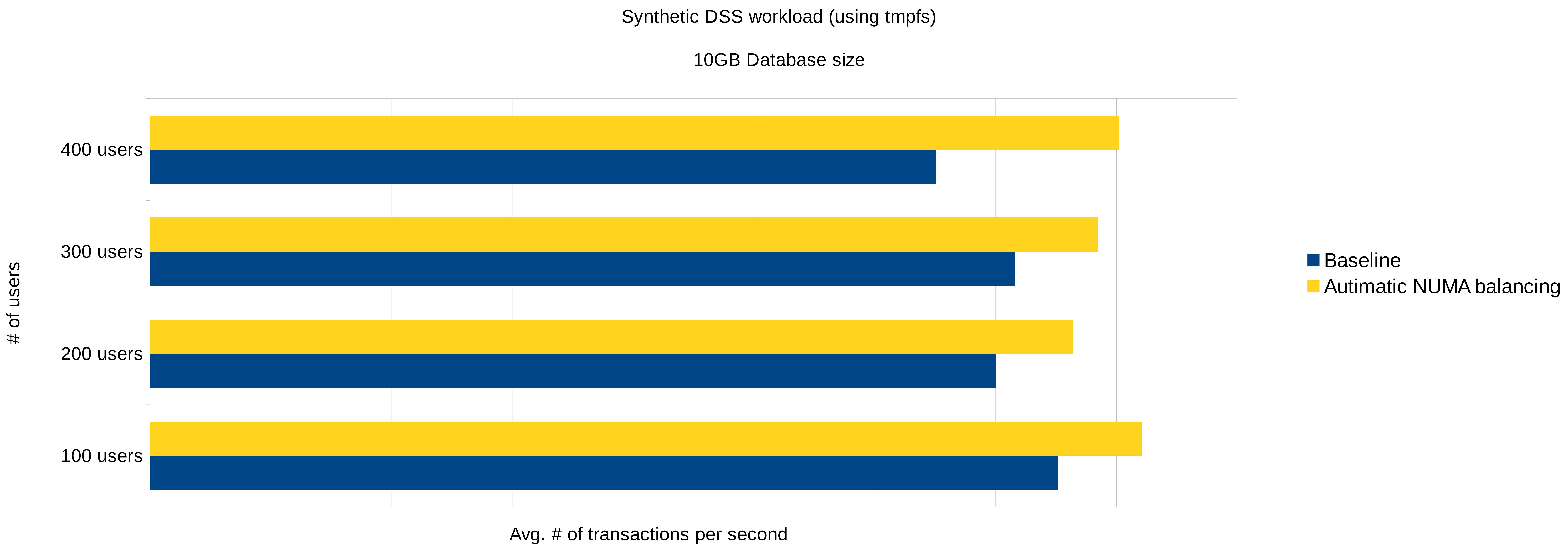




Some workloads will still need manual pinning !

Database workload - bare-metal

(4-socket IVY-EX server)



~9-18% improvement in Average transactions per second with Automatic NUMA balancing

KVM guests

- Virtual machines are getting larger and larger
  - Use case 1: classic enterprise scale-up VMs
  - Use case 2: VMs in private cloud environments
- Low overhead & predictable performance for VMs (of any size) => careful planning & provisioning
  - Host's NUMA topology & the current resource usage
  - Manual pinning/tuning using libvirt/virsh
- Problem: It's harder to live migrate a pinned VM
  - Resources not available on destination host
  - Destination host - different NUMA topology !

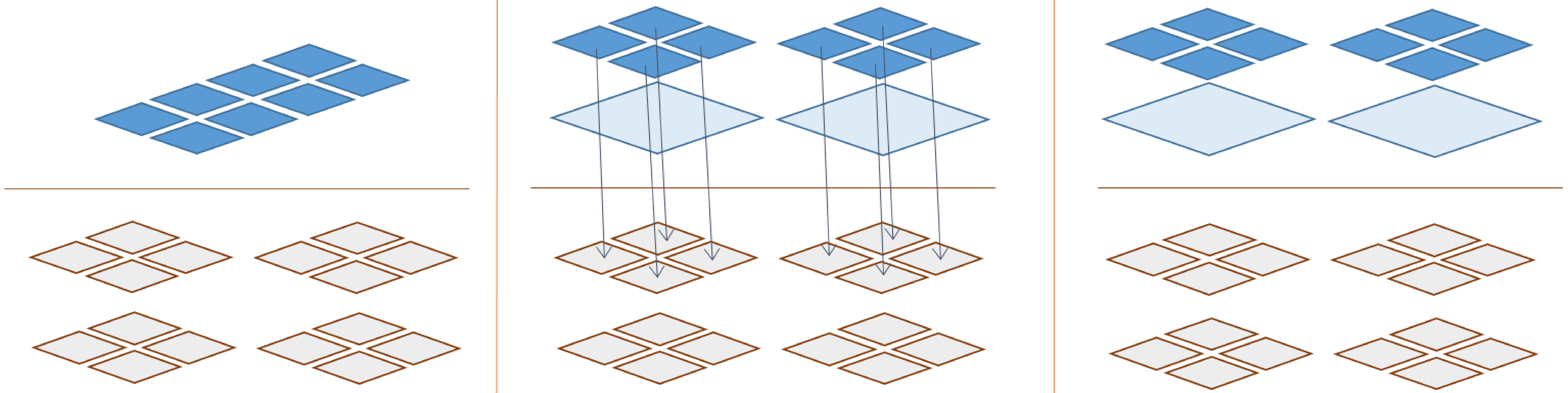
```
 <cputune>
   <vcpupin vcpu='0' cpuset='0'/>
   <vcpupin vcpu='1' cpuset='1'/>
   …
   <vcpupin vcpu='29' cpuset='29'/>
</cputune>

<numatune>
   <memory mode='preferred' nodeset='0-1'/>
</numatune>
```

KVM guests (cont.)

- Automatic NUMA balancing avoids need for manual pinning
  - Exceptions - where memory pages can't be migrated
    - hugetlbfs (instead of THPs)
    - Device assignment (get_user_pages())
    - Hosting applications with real time needs (mlock_all()).
- VM >1 socket - enable Virtual NUMA nodes
  - Helps guest OS to scale/perform
  - Automatic NUMA balancing is enabled
  - Can pin the workload to virtual NUMA nodes

```
 <cpu>
  <topology sockets='2' cores='15' threads='1'/>
  <numa>
    <cell cpus='0-14' memory='134217728'/>
    <cell cpus='15-29' memory='134217728'/>
  </numa>
</cpu>
```

KVM guests (cont.)



8VCPU & 32GB VM

4-socket host
4 cores & 32 GB / socket

Baseline VM

VCPUs could run anywhere

Backing memory could be remote

No virtual NUMA nodes in the guest

Pinned VM

Two 4VCPU/16GB virtual NUMA nodes

Each VCPU pinned to each Host CPU core
on specified sockets/nodes.

Backing memory from specified
Host nodes (strict/preferred/interleave)

Virtual NUMA nodes =>
Better scaling in the guest OS

Automatic NUMA bal. VM

Two 4VCPU/16GB virtual NUMA nodes

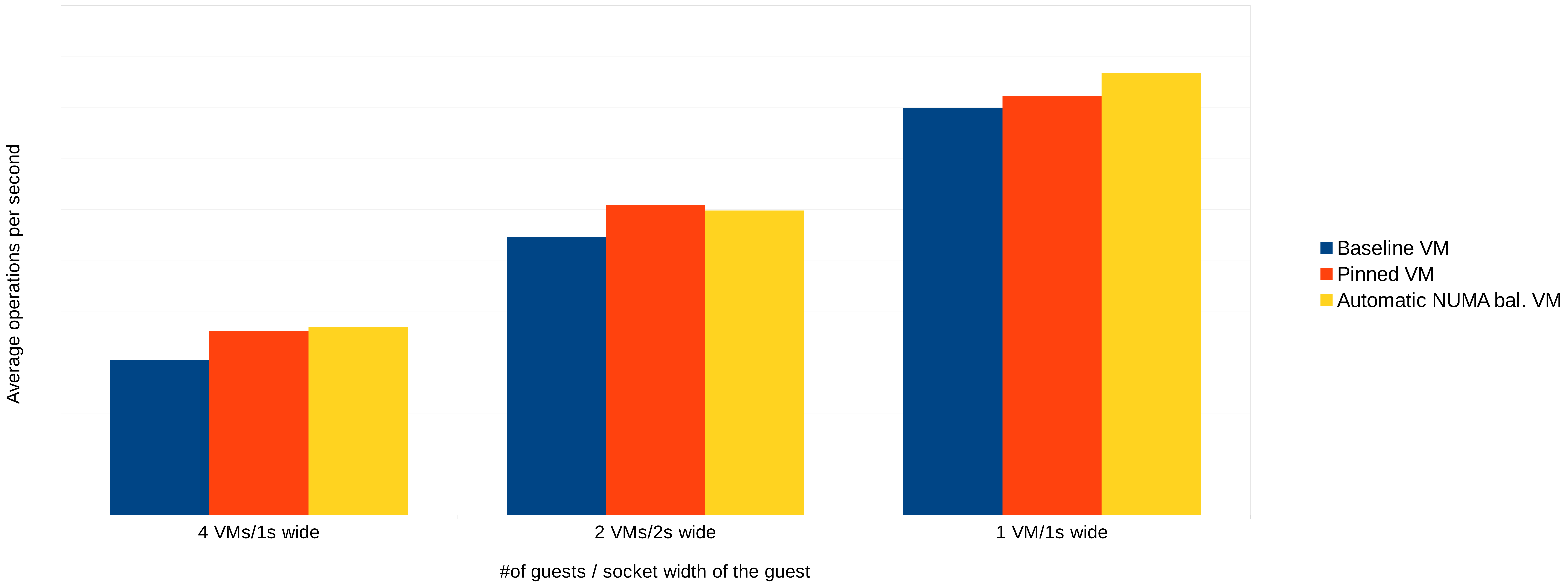
Host OS's Automatic NUMA bal. in control of
finding the best backing resources for the VM

Virtual NUMA nodes =>
Better scaling in the guest OS +
Automatic NUMA bal. in Guest OS.
Can also pin workload in the Guest OS

KVM guests - HP Proliant DL 580 Gen8 - 4-socket Ivy Bridge EX server

- Guest sizes
  - 1s-wide guest → 15VCPUs/128GB
  - 2s-wide guest → 30VCPUs/256GB (2 virtual NUMA nodes)
  - 4s-wide guest → 60VCPUs/512GB (4 virtual NUMA nodes)
- Configurations tested
  - **Baseline VM** => a typical public/private cloud VM today
    - No pinning, no virtual NUMA nodes, no Automatic NUMA balancing in host or guest
  - **Pinned VM** => a typical enterprise scale-up VM today
    - VCPUs and memory pinned, virtual NUMA nodes (for > 1s wide VM)
    - Workload pinned in the guest OS (to virtual NUMA nodes)
  - **Automatic NUMA balanced VM** => “out of box” for any type of VM
    - Automatic NUMA balancing in host and guest
    - Virtual NUMA nodes enabled in the VM (for > 1s wide VM)

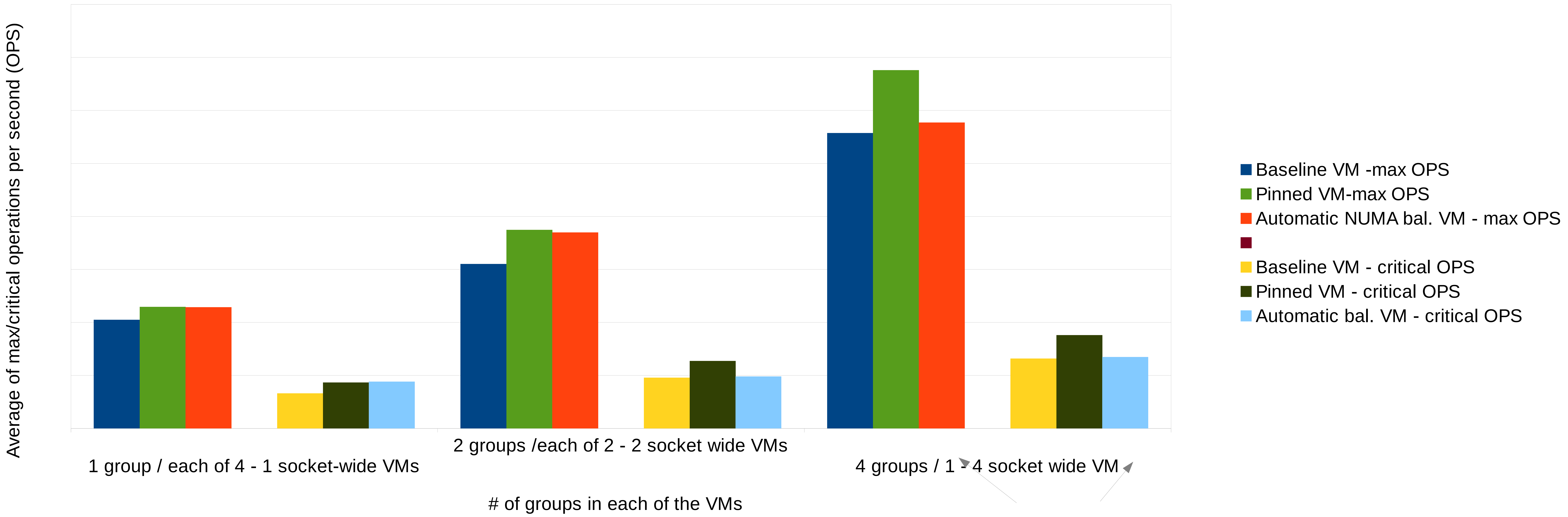
SPECjbb2005 in KVM
(4 socket IVY-EX server)



Pinned VM performed 5-16% better than Baseline VM
Automatic NUMA bal. VM & the Pinned VM were pretty close (+/- 3%).

Multi-JVM server workload in KVM

(4-socket IVY-EX server)



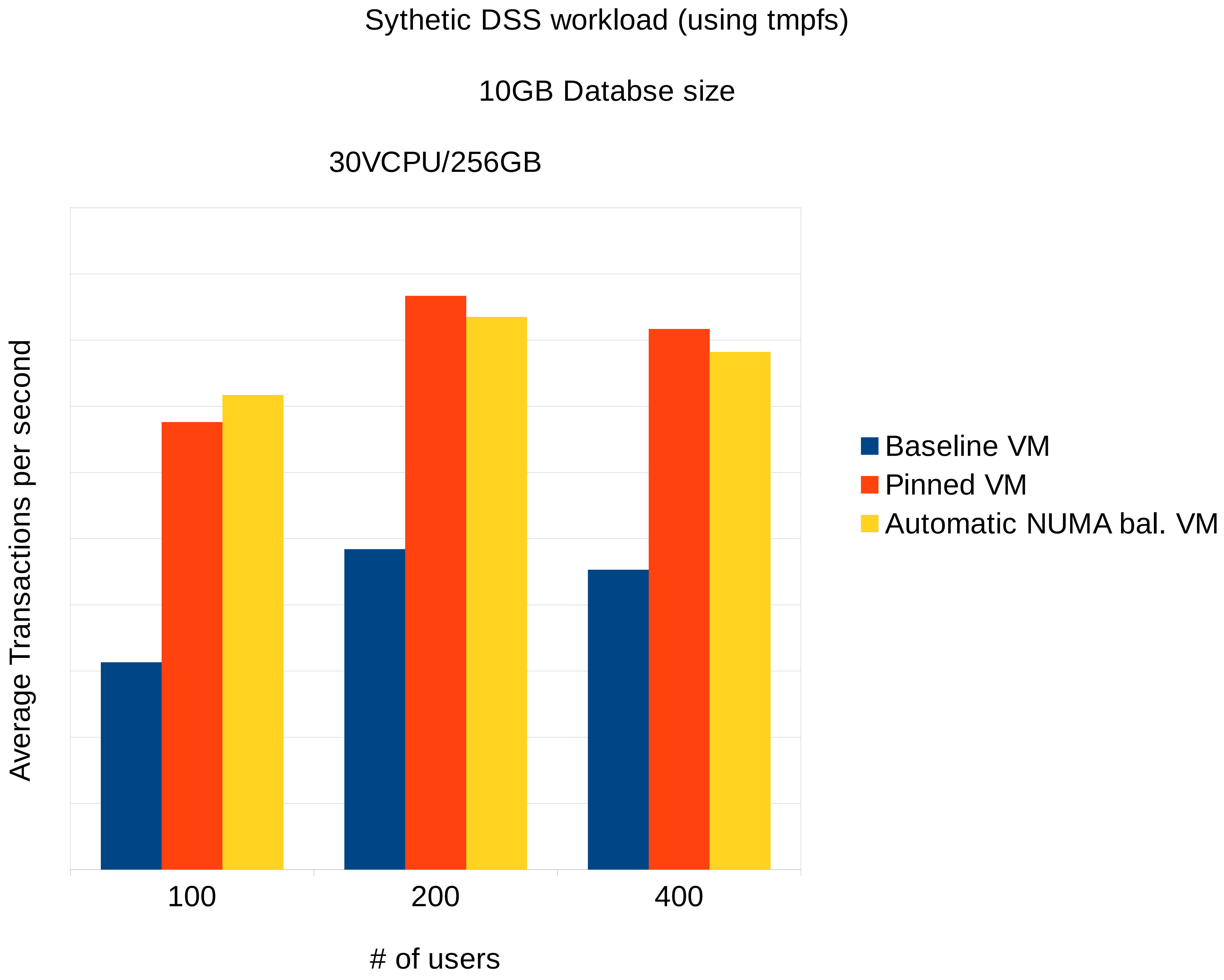
Pinned VM was 11-18% better for max-OPS and 24-26% better for critical-OPS relative to the Baseline VM.
For VMs up to 2 socket wide the Automatic NUMA bal. VM was closer to Pinned VM.

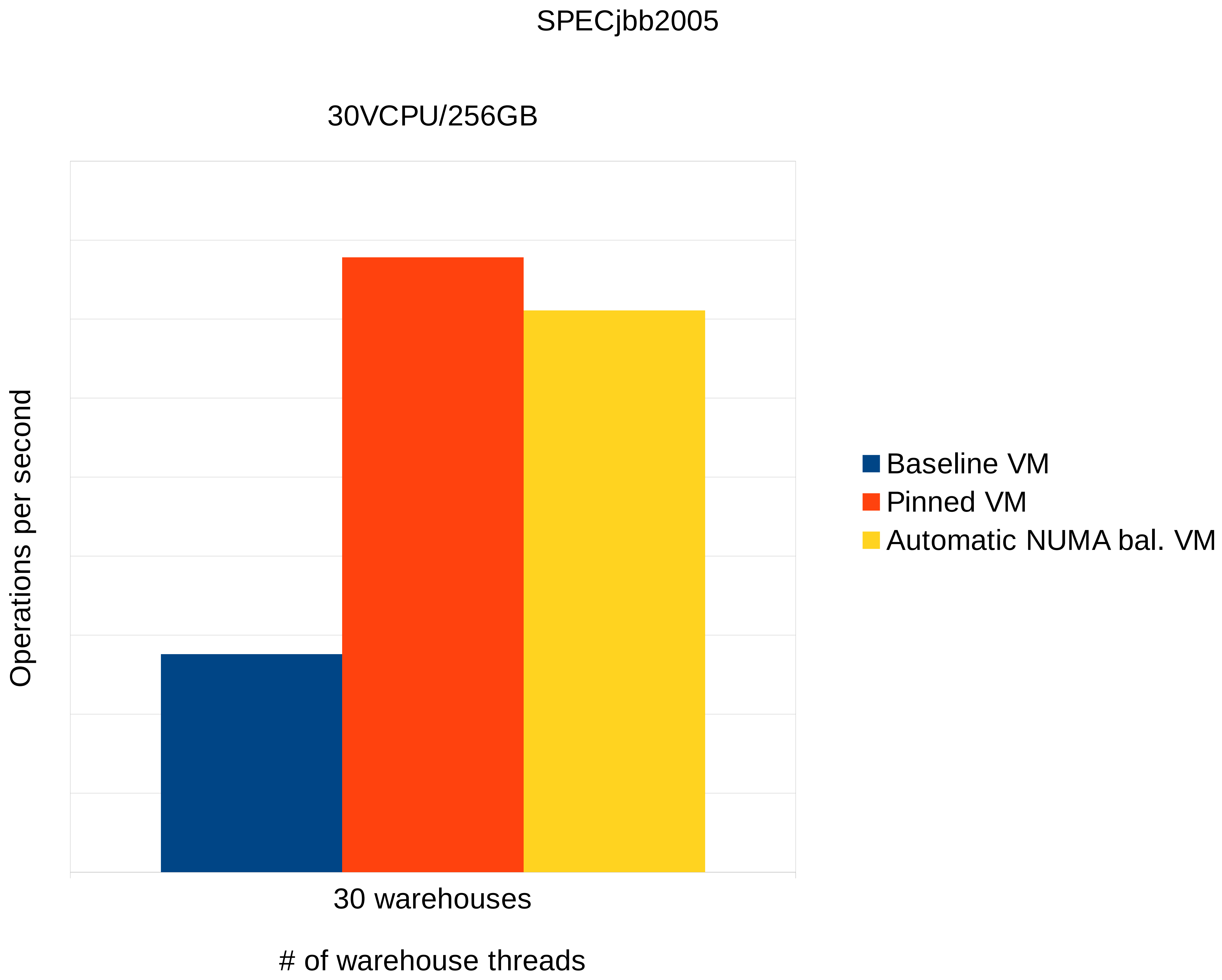
Delta between the Automatic NUMA bal. VM case & the Pinned VM case was much higher (~14% max-OPS and ~24% of critical-OPS)

Pinning the workload to the virtual NUMA nodes in the larger Automatic NUMA bal. Guest OS does bridge the gap.

KVM – server consolidation example 1

(Two VMs each running a different workload hosted on 4 Socket IVY-EX server)

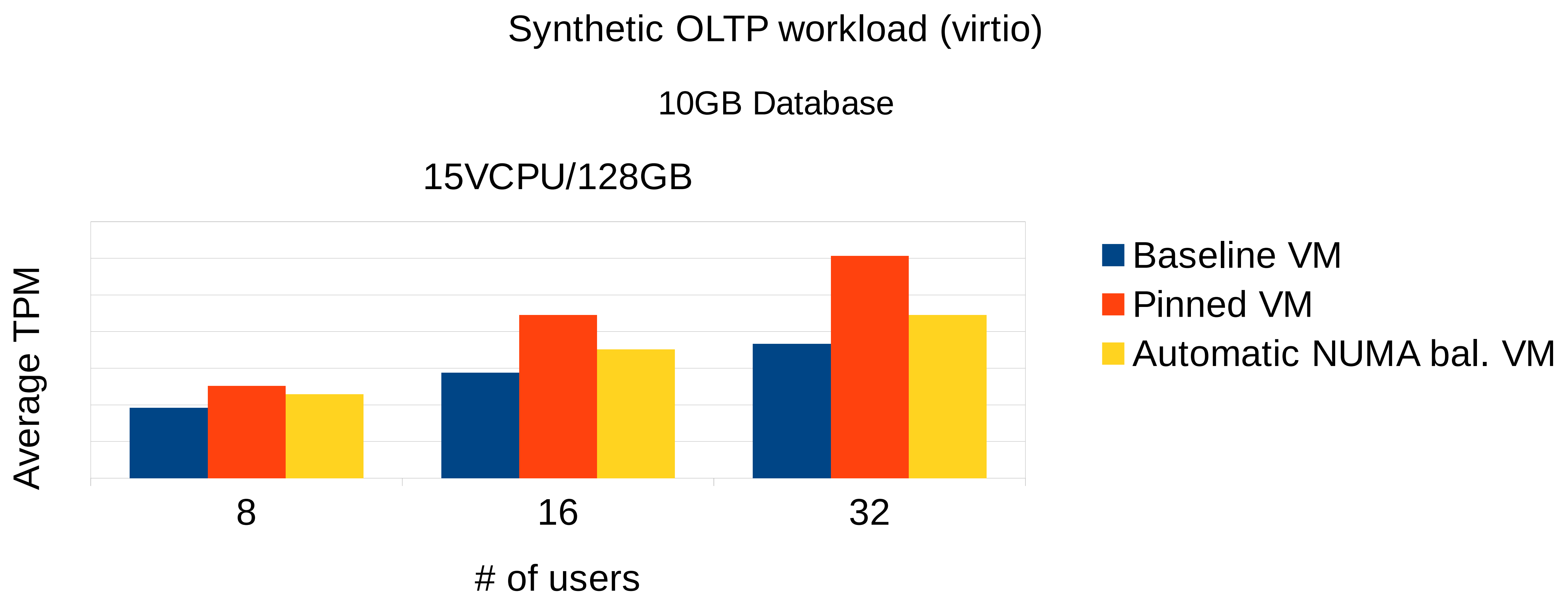


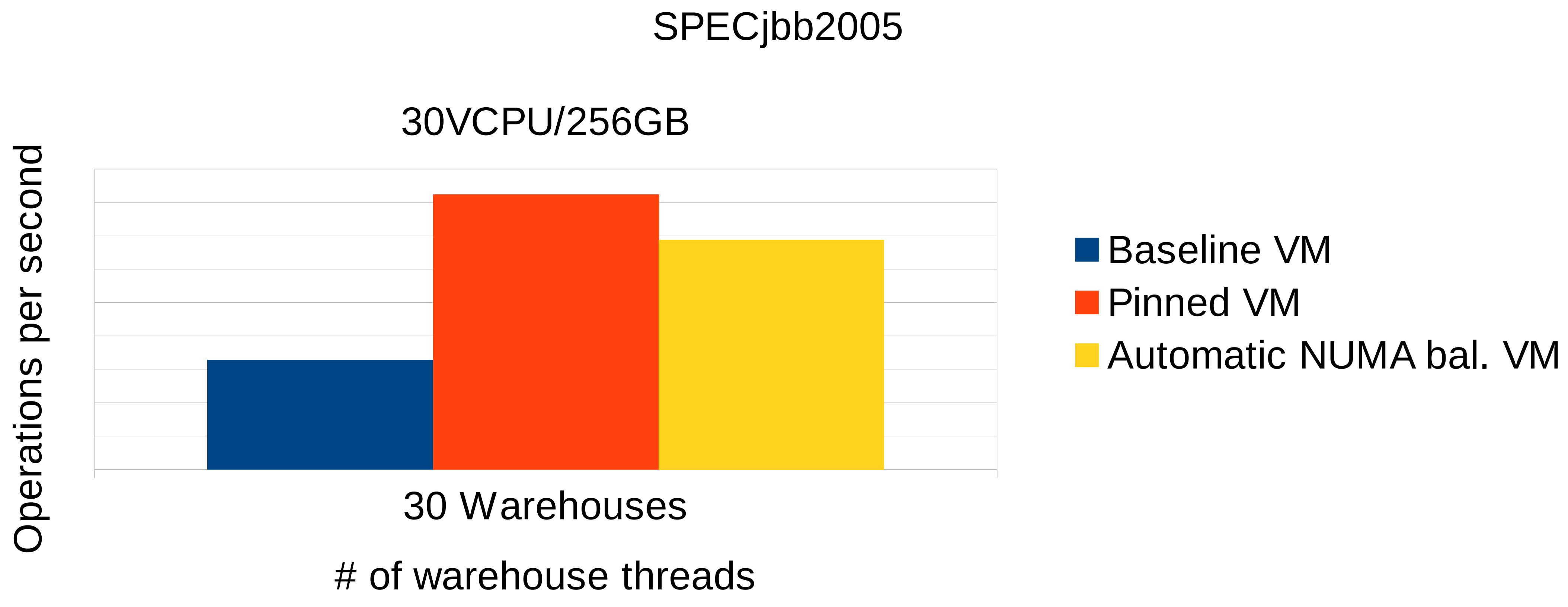


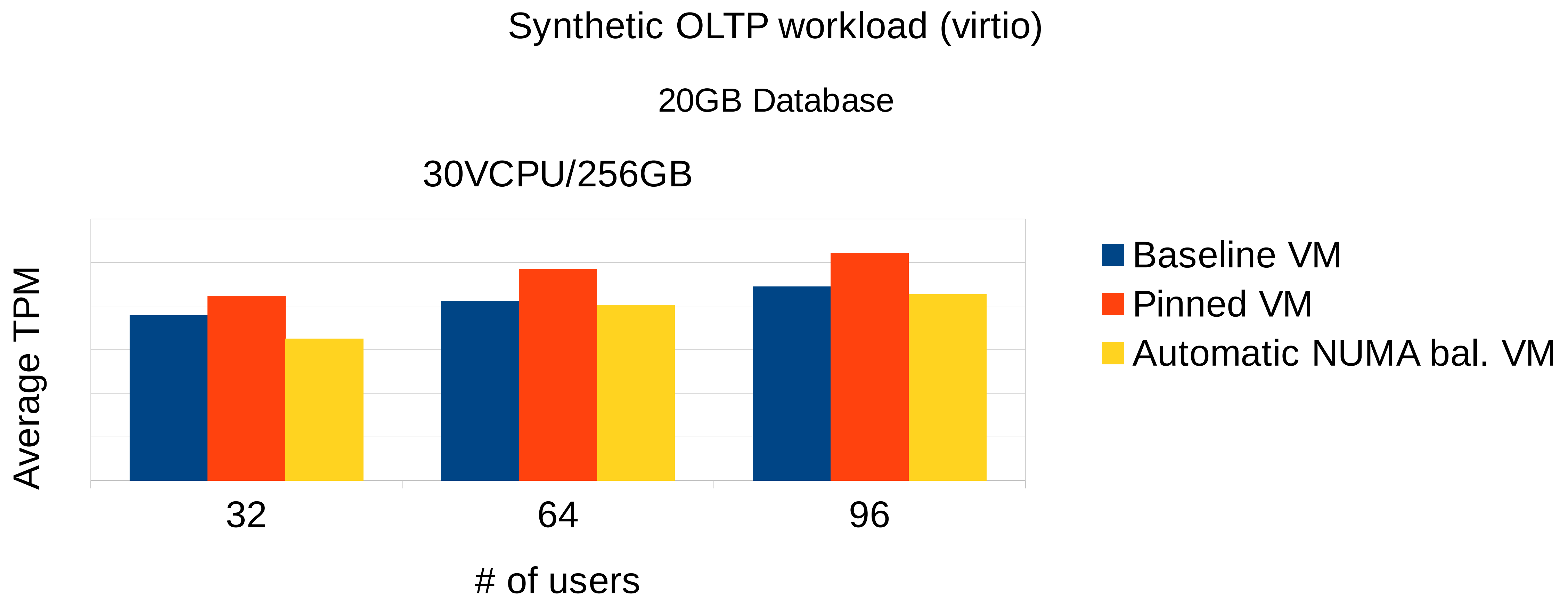
Pinned VM was at least 10% better than Baseline VM.
Automatic NUMA bal. VM & the Pinned VM were pretty close (~ +/- 1-2%).

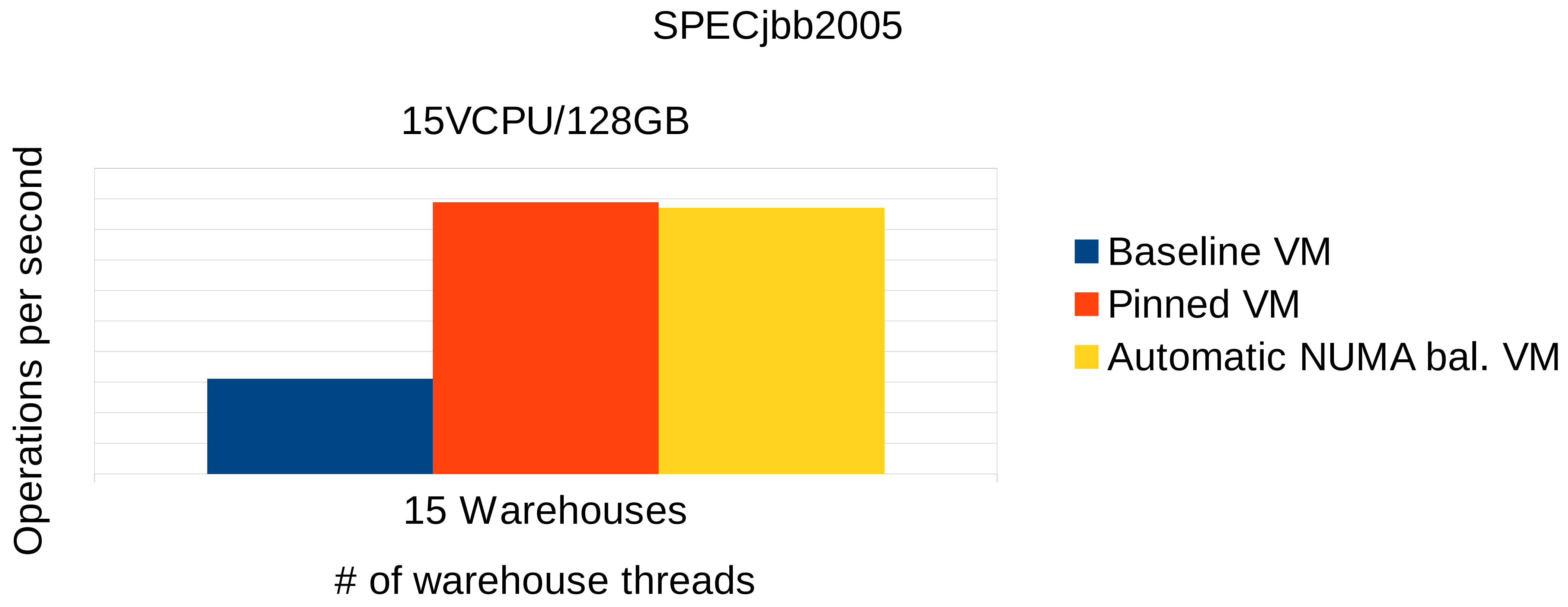
KVM – server consolidation example 2

(Two VMs each running a different workload hosted on 4 Socket IVY-EX server)









For smaller VM size Automatic NUMA balancing was ~10-25% lower than Pinned VM
For larger VM size Automatic NUMA balancing was 5-15% lower than Baseline case !

NUMA Tools

What can I do?

NUMA Tools

- Numactl
- Numad
- taskset
- NUMA statistics in /proc
- Red Hat Enterprise Virtualization

numactl

- Control NUMA policy for processes or shared memory
  - numactl --arguments <program> <program arguments>
  - Bind to a node, interleave, ...
  - numactl --shmid to change properties of shared memory segment
- Show NUMA properties of the system
  - numactl --hardware

numad

- Optional user level daemon to do NUMA balancing for workloads or KVM guests
- More static than in-kernel automatic NUMA balancing
  - Better for some workloads
  - Worse for others
- Available in RHEL 6 & RHEL 7
  - You can use it today

Taskset

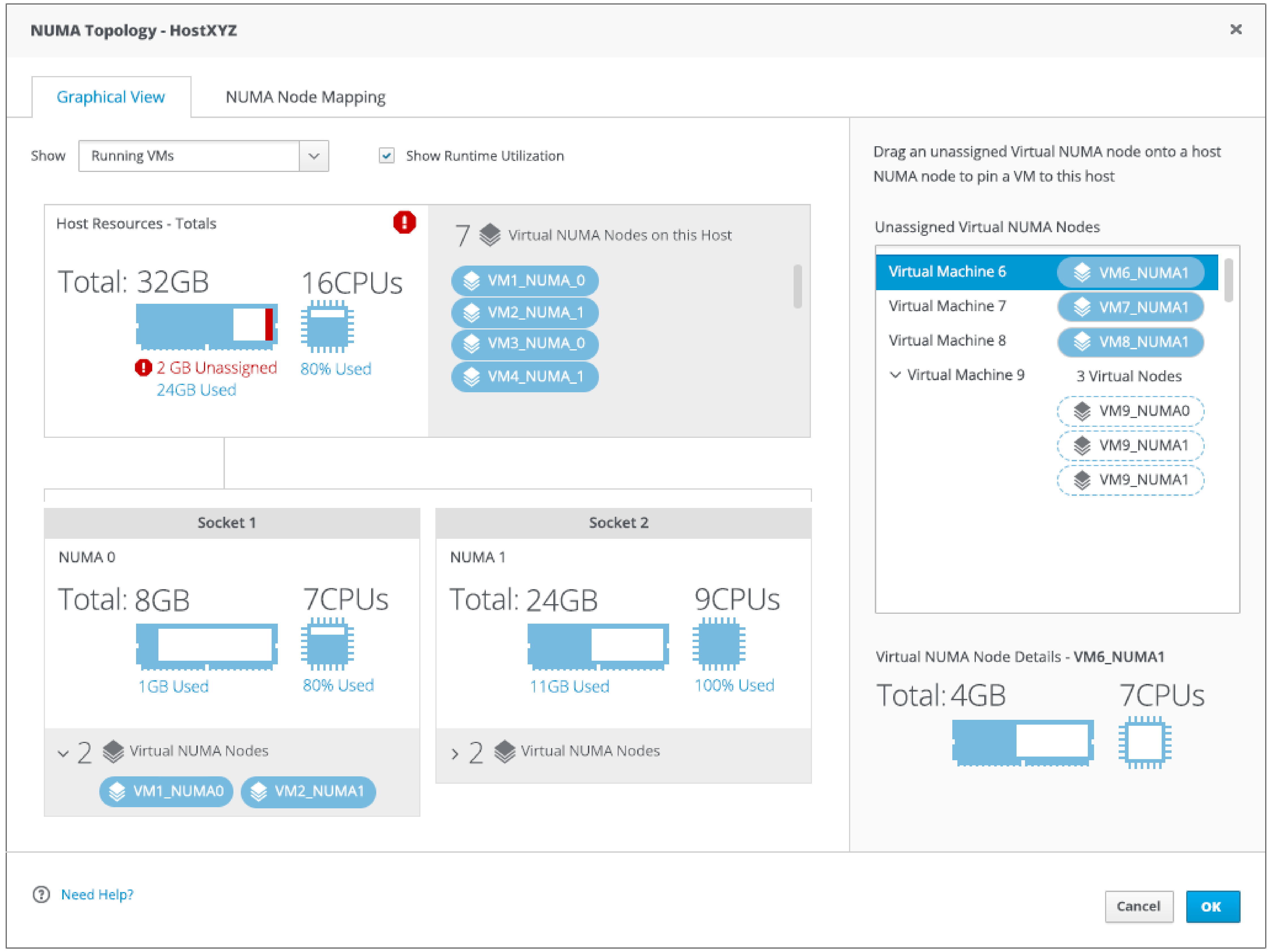
- Retrieve or set a process's CPU affinity
- Works on new commands, or PIDs of already running tasks
- Can bind tasks to the CPUs in a NUMA node
- Works for whole processes, or individual threads

NUMA statistics in /proc

- /proc/vmstat numa_* fields
  - NUMA locality (hit vs miss, local vs foreign)
  - Number of NUMA faults & page migrations
- /proc/<pid>/numa_maps
  - Location of process memory in NUMA system
- /proc/<pid>/sched
  - Numa scans, migrations & numa faults by node

Red Hat Enterprise Virtualization

(A preview of enhancements in the pipeline...)



NUMA Topology - HostXYZ

Show: Running VMs

☑ Show Runtime Utilization

Host Resources - Totals

Total: 32GB — 16CPUs

2 GB Unassigned
24GB Used
80% Used

7 Virtual NUMA Nodes on this Host

Socket 1

NUMA 0

Total: 8GB — 7CPUs

1GB Used — 80% Used

2 Virtual NUMA Nodes

Socket 2

NUMA 1

Total: 24GB — 9CPUs

11GB Used — 100% Used

2 Virtual NUMA Nodes

Drag an unassigned Virtual NUMA node onto a host NUMA node to pin a VM to this host

Unassigned Virtual NUMA Nodes

Virtual Machine 6	VM6_NUMA1
Virtual Machine 7	VM7_NUMA1
Virtual Machine 8	VM8_NUMA1
Virtual Machine 9	3 Virtual Nodes VM9_NUMA0 VM9_NUMA1 VM9_NUMA1

Virtual NUMA Node Details - **VM6_NUMA1**

Total: 4GB — 7CPUs

- Graphical tool for creating KVM guests with NUMA awareness
  - Visualize host NUMA topology and current resource usage.
  - Define Virtual NUMA nodes for a guest.
  - Bind to NUMA nodes on the host (optional)
- Will work with RHEL6 & RHEL7 based RHEV-H hypervisors

(final version of the GUI will differ)

Future Developments

What can't it do (yet)?

NUMA balancing future considerations

- Complex NUMA topologies & pseudo-interleaving
- Unmovable memory
- KSM
- Interrupt locality
- Inter Process Communication

Complex NUMA topologies & pseudo-interleaving

- Differing distances between NUMA nodes
  - Local node, nearby nodes, far away nodes
  - Eg. 20% & 100% performance penalty for nearby vs. far away
- Workloads that are spread across multiple nodes work better when those nodes are near each other
- Prototype implementation written last week
  - Improves some cases, still some issues remaining

NUMA balancing & unmovable memory

- Unmovable memory
  - Mlock
  - Hugetlbfs
  - Pinning for KVM device assignment & RDMA
- Memory is not movable ...
  - But the tasks are
  - NUMA faults would help move the task near the memory
  - Unclear if worthwhile, needs experimentation

KSM

- Kernel Samepage Merging
  - De-duplicates identical content between KVM guests
  - Also usable by other programs
- KSM has simple NUMA option
  - “Only merge between tasks on the same NUMA node”
  - Task can be moved after memory is merged
  - May need NUMA faults on KSM pages, and re-locate memory if needed
  - Unclear if worthwhile, needs experimentation

Interrupt locality

- Some tasks do a lot of IO
  - Performance benefit to placing those tasks near the IO device
  - Manual binding demonstrates that benefit
- Currently automatic NUMA balancing only does memory & CPU
- Would need to be enhanced to consider IRQ/device locality
- Unclear how to implement this
  - When should IRQ affinity outweigh CPU/memory affinity?

Inter Process Communication

- Some tasks communicate a LOT with other tasks
  - Benefit from NUMA placement near tasks they communicate with
  - Do not necessarily share any memory
- Loopback TCP/UDP, named socket, pipe, ...
- Unclear how to implement this
  - How to weigh against CPU/memory locality?

Conclusions

- NUMA systems are increasingly common
- Automatic NUMA balancing improves performance for many workloads, on many kinds of systems
  - Can often get within a few % of optimal manual tuning
  - Manual tuning can still help with certain workloads
- Future improvements may expand the range of workloads and systems where automatic NUMA placement works nearly optimal