A Hitchhikers Guide to Linux performance Issues

Christian Ehrhardt
IBM

4th August 2014
16244

#SHAREorg

SHARE in Pittsburgh 2014

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml .

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Red Hat, Red Hat Enterprise Linux (RHEL) are registered trademarks of Red Hat, Inc. in the United States and other countries.

SUSE and SLES are registered trademarks of SUSE LLC in the United States and other countries.

Java and Oracle are registered trademarks of Oracle and/or its affiliates.

Other product and service names might be trademarks of IBM or other companies.

Agenda

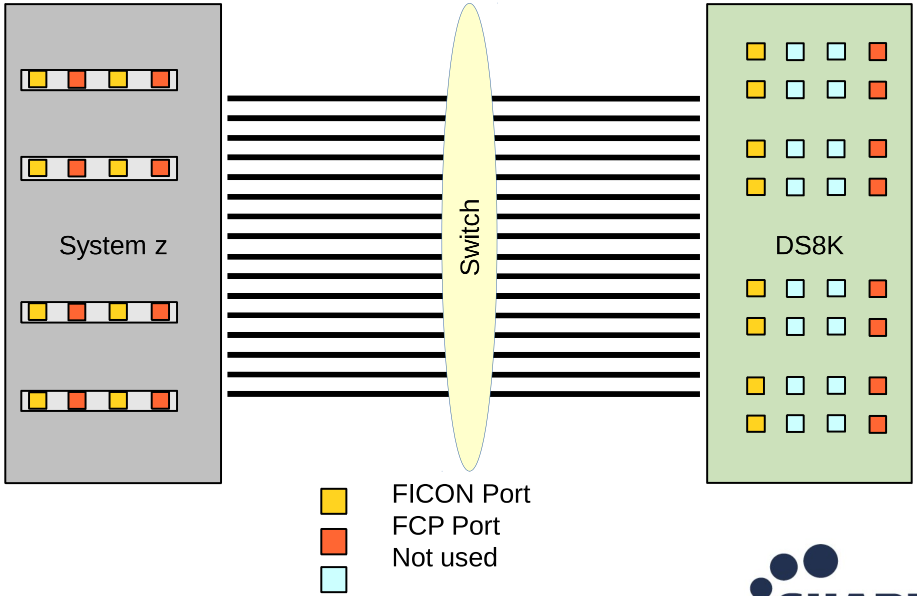
• **Disk performance**	approximately 55% of external support requests
• Network performance	approximately 25% of external support requests
• Compiler	two ISVs and one of the biggest logistic companies
• Huge pages	beneficial in almost every huge installation

- In any environment from which we got support requests at least one of these areas was set up sub-optimally wasting performance or efficiency
  - So lets derive optimistically:
    “maybe those people following this guide never have significant issues”
  - Let us work on making you one of those

Disk I/O - benchmark description and configuration

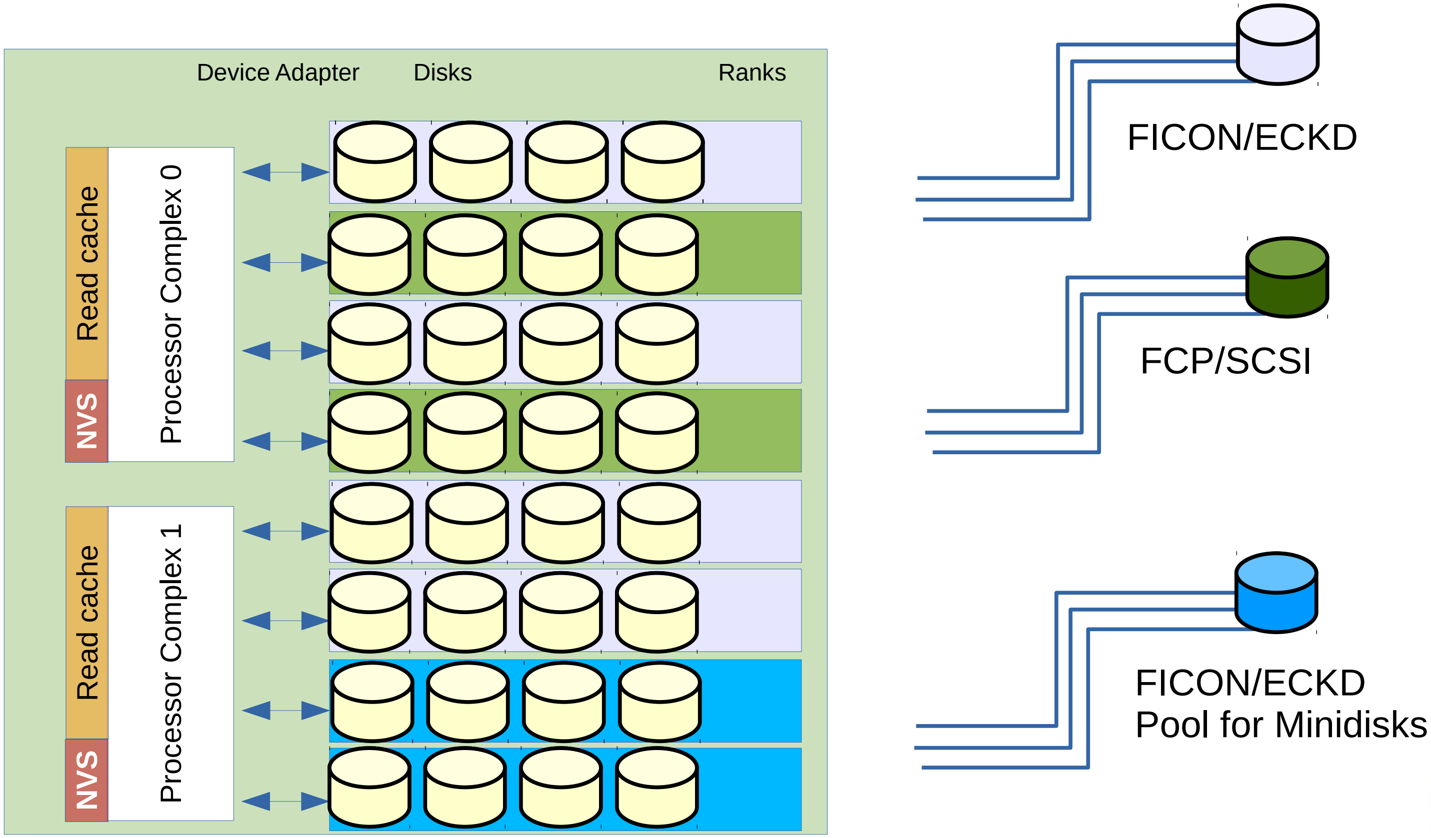
- Flexible I/O Testing Tool (FIO)
  - Benchmarking and hardware verification / stress I/O devices
  - Open Source (GPLv2)
  - Easy to customize to needs

- Configuration
  - 8 processors
  - 512 MB main memory
  - z196 connected to DS8800
  - FICON Express 8s
  - 64 single disks, each in FICON and SCSI



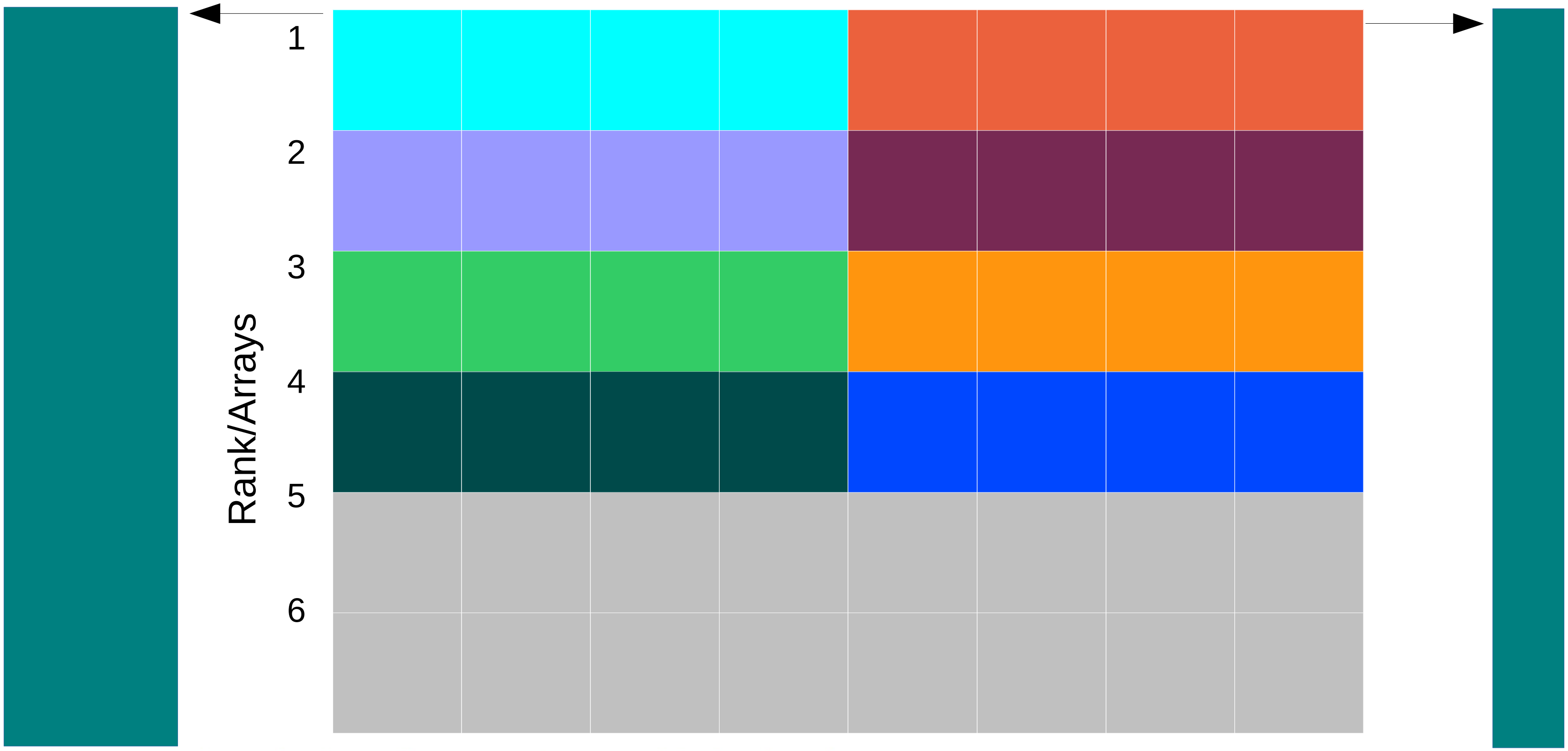
Disk I/O – Storage Server DS8x00

- Storage server basics – various configurations possible
  - Preferable many ranks into a extent pool with Storage Pool Striping (extents striped over all ranks within extent pool)



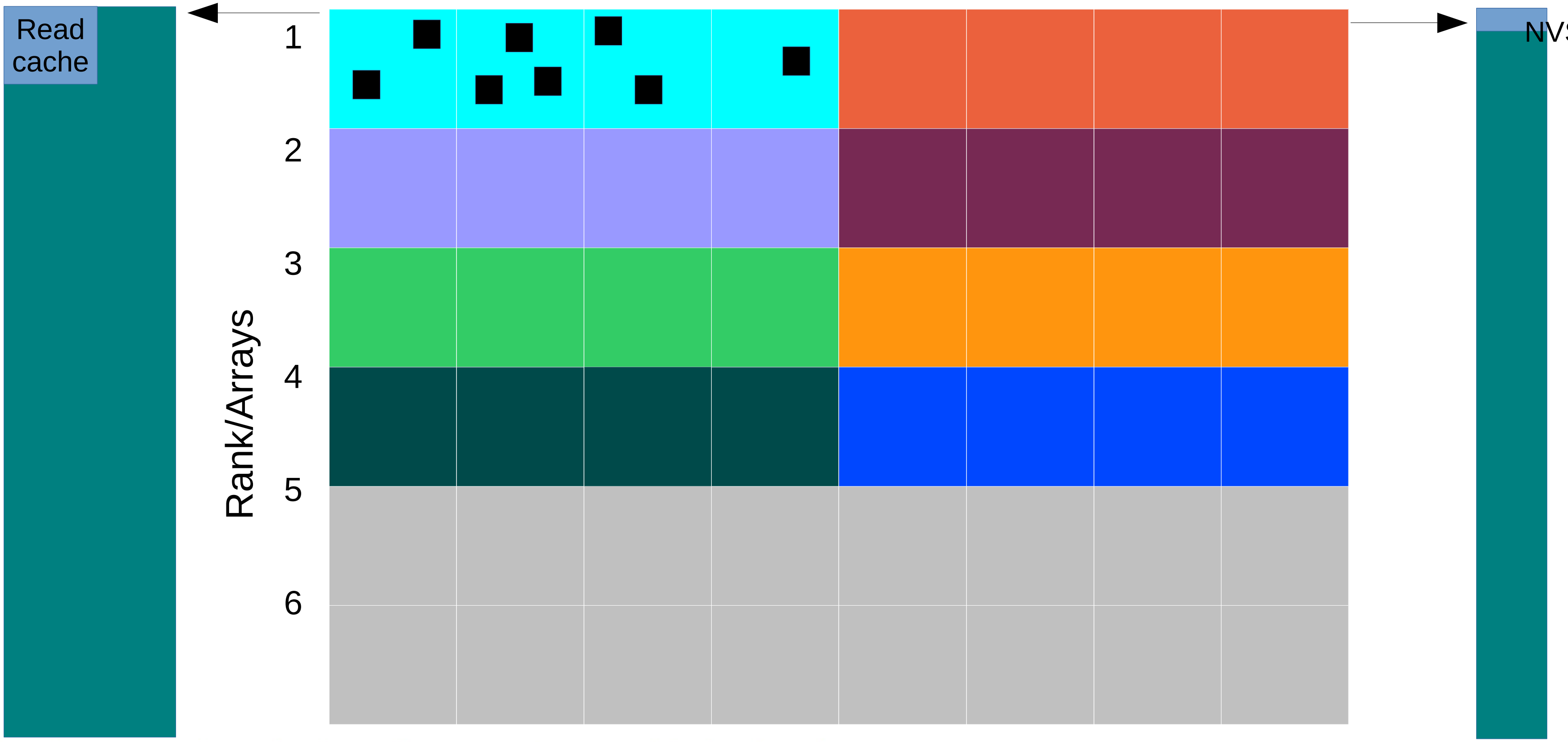
Disk I/O – Volumes

- Extent pool with 8 disks a 4 GB defined
  - Each rank has access to an adequate portion of the read cache and non-volatile storage (NVS – write cache)
- Example: random access to one volume
  - Usable portions of read cache and NVS very limited because just one rank is involved
  - Only one Device Adapter (DA) in use



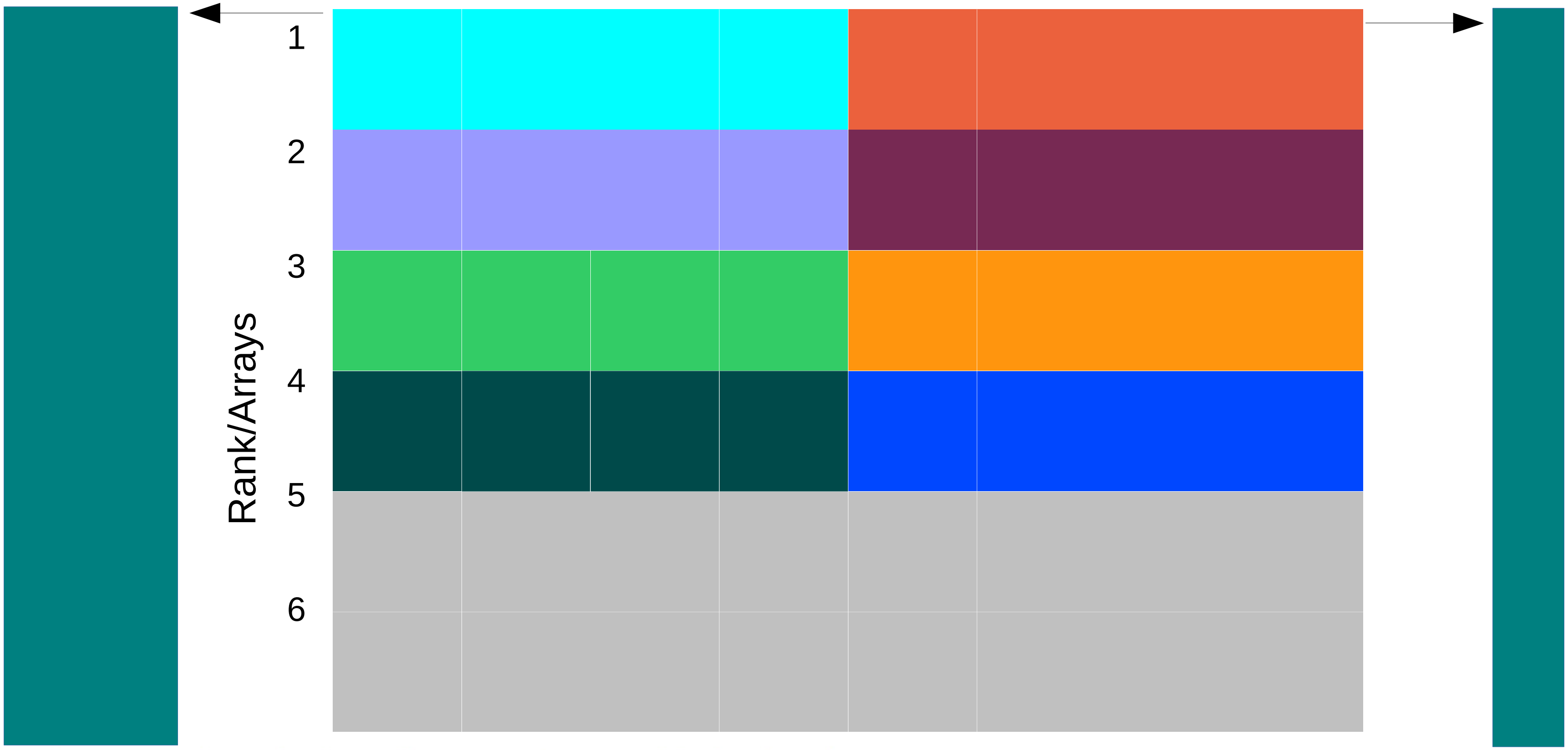
Disk I/O – Volumes

- Extent pool with 8 disks a 4 GB defined
  - Each rank has access to an adequate portion of the read cache and non-volatile storage (NVS – write cache)
- Example: random access to one volume
  - Usable portions of read cache and NVS very limited because just one rank is involved
  - Only one Device Adapter (DA) in use



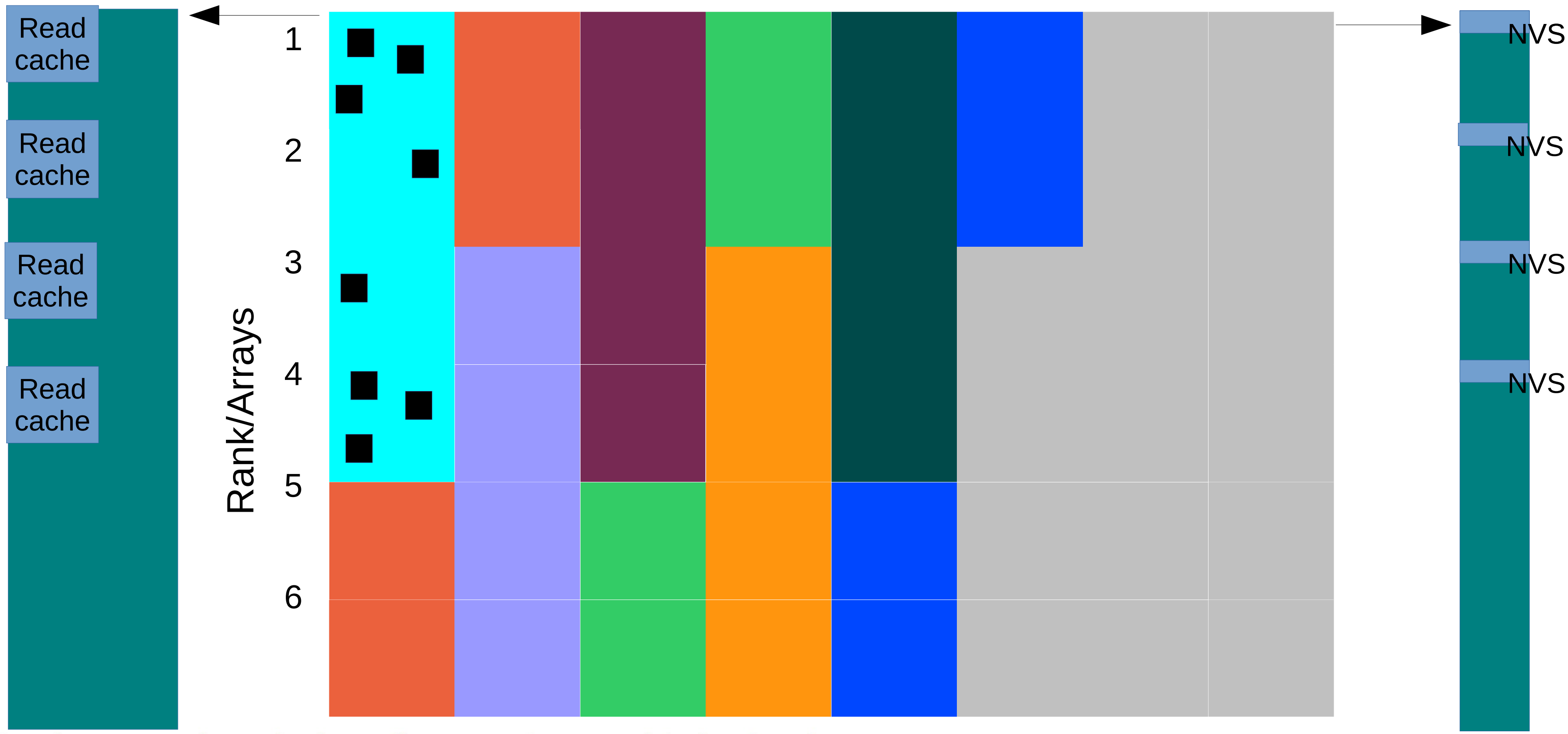
Disk I/O – Volumes with Storage Pool Striping (SPS)

- Extent pool example with 8 disks a 4 GB, with Storage Pool Striping (SPS)
  - Each rank has access to an adequate portion of the read cache and non-volatile storage (NVS – write cache)
- Example: random access to one SPS volume
  - Usable portions of read cache and NVS much bigger because four ranks are involved
  - Up to four Device Adapters (DA) are in use



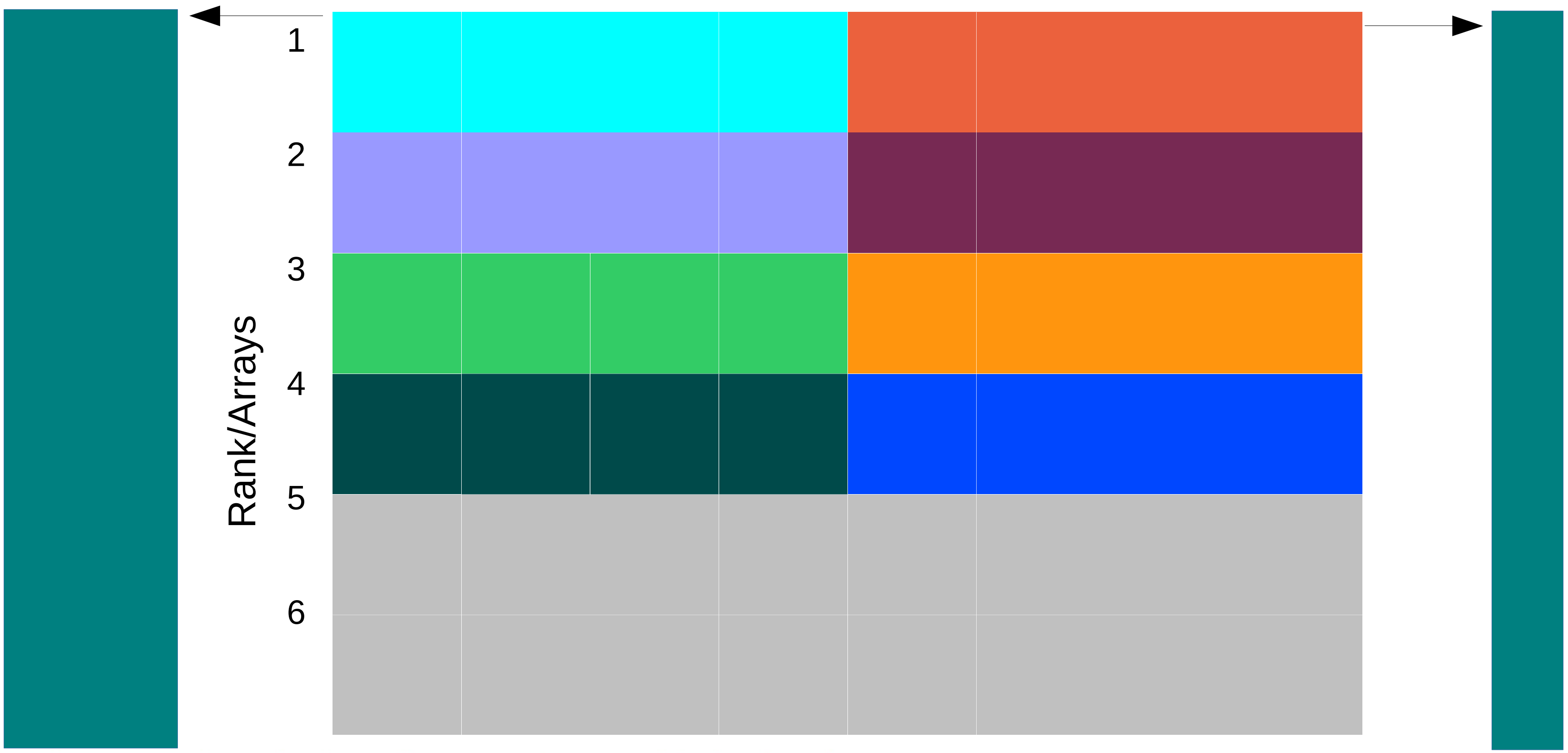
Disk I/O – Volumes with Storage Pool Striping (SPS)

- Extent pool example with 8 disks a 4 GB, with Storage Pool Striping (SPS)
  - Each rank has access to an adequate portion of the read cache and non-volatile storage (NVS – write cache)
- Example: random access to one SPS volume
  - Usable portions of read cache and NVS much bigger because four ranks are involved
  - Up to four Device Adapters (DA) are in use



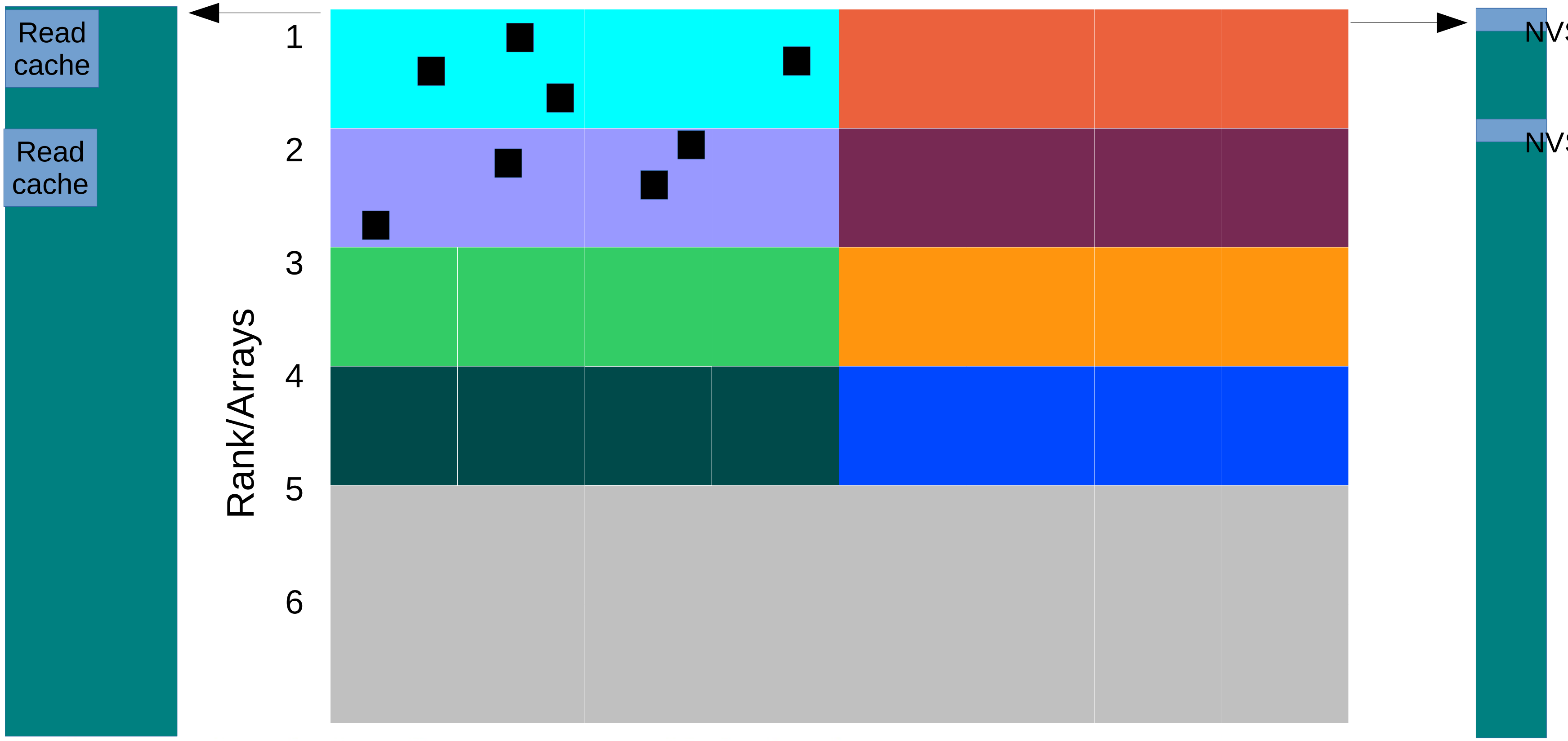
Disk I/O – two volumes in a striped LVM

- Extent pool example with 8 disks of 4 GB size
  - Each rank has access to an adequate portion of the read cache and non-volatile storage (NVS – write cache)
- Two volumes are used for the LVM
  - Usable read cache portions and NVS very limited because only two ranks are involved
  - Up to two Device Adapters (DA) are used for the connection to cache and NVS



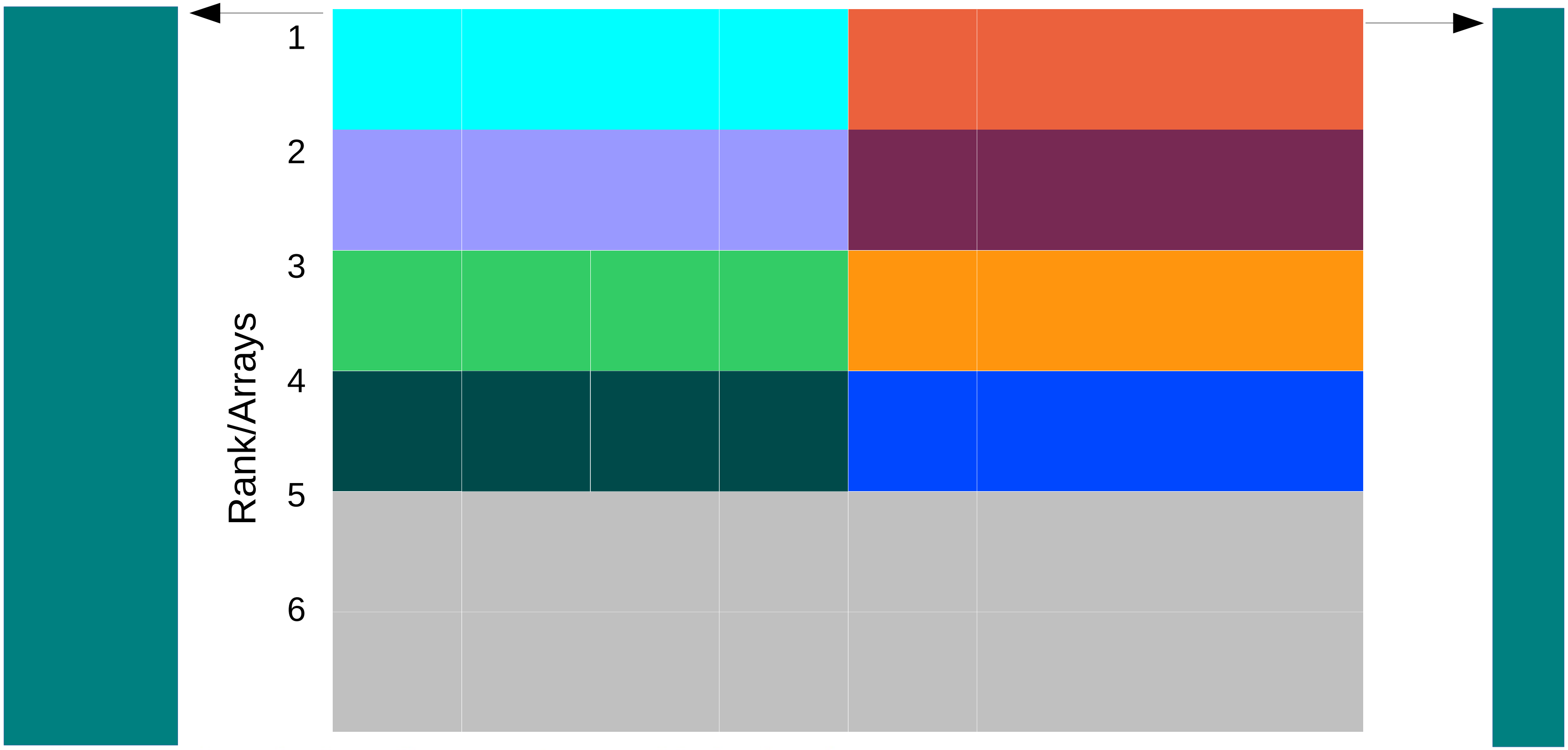
Disk I/O – two volumes in a striped LVM

- Extent pool example with 8 disks of 4 GB size
  - Each rank has access to an adequate portion of the read cache and non-volatile storage (NVS – write cache)
- Two volumes are used for the LVM
  - Usable read cache portions and NVS very limited because only two ranks are involved
  - Up to two Device Adapters (DA) are used for the connection to cache and NVS



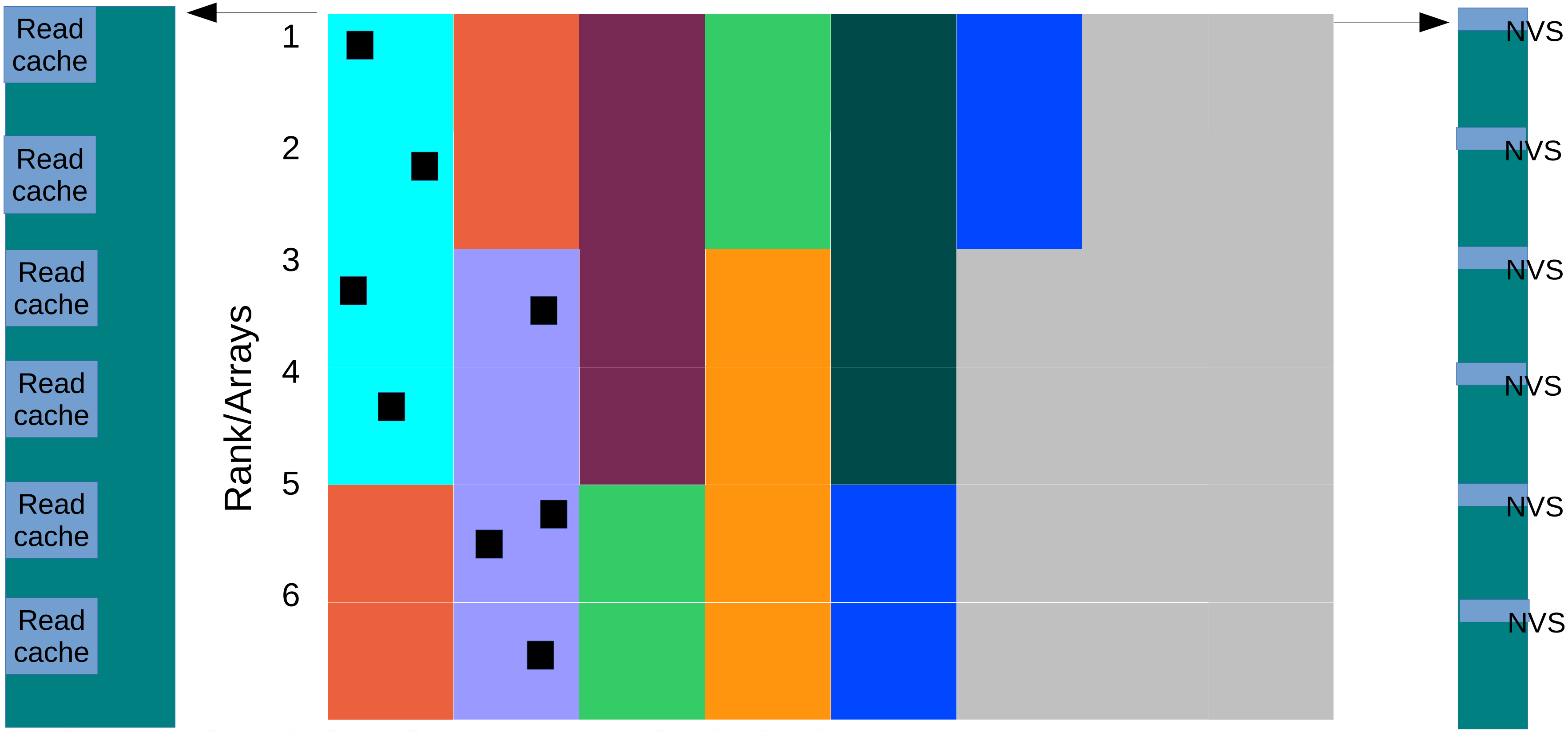
Disk I/O – two SPS volumes in a striped LVM

- Extent pool example with 8 disks a 4 GB, with Storage Pool Striping (SPS)
  - Each rank has access to an adequate portion of the overall amount of read cache and non-volatile storage (NVS – write cache)
- Two SPS volumes are used for the LVM
  - Usable portions of read cache and NVS much bigger because six ranks are involved
  - Up to six Device Adapters (DA) are used for the connection to cache and NVS



Disk I/O – two SPS volumes in a striped LVM

- Extent pool example with 8 disks a 4 GB, with Storage Pool Striping (SPS)
  - Each rank has access to an adequate portion of the overall amount of read cache and non-volatile storage (NVS – write cache)
- Two SPS volumes are used for the LVM
  - Usable portions of read cache and NVS much bigger because six ranks are involved
  - Up to six Device Adapters (DA) are used for the connection to cache and NVS



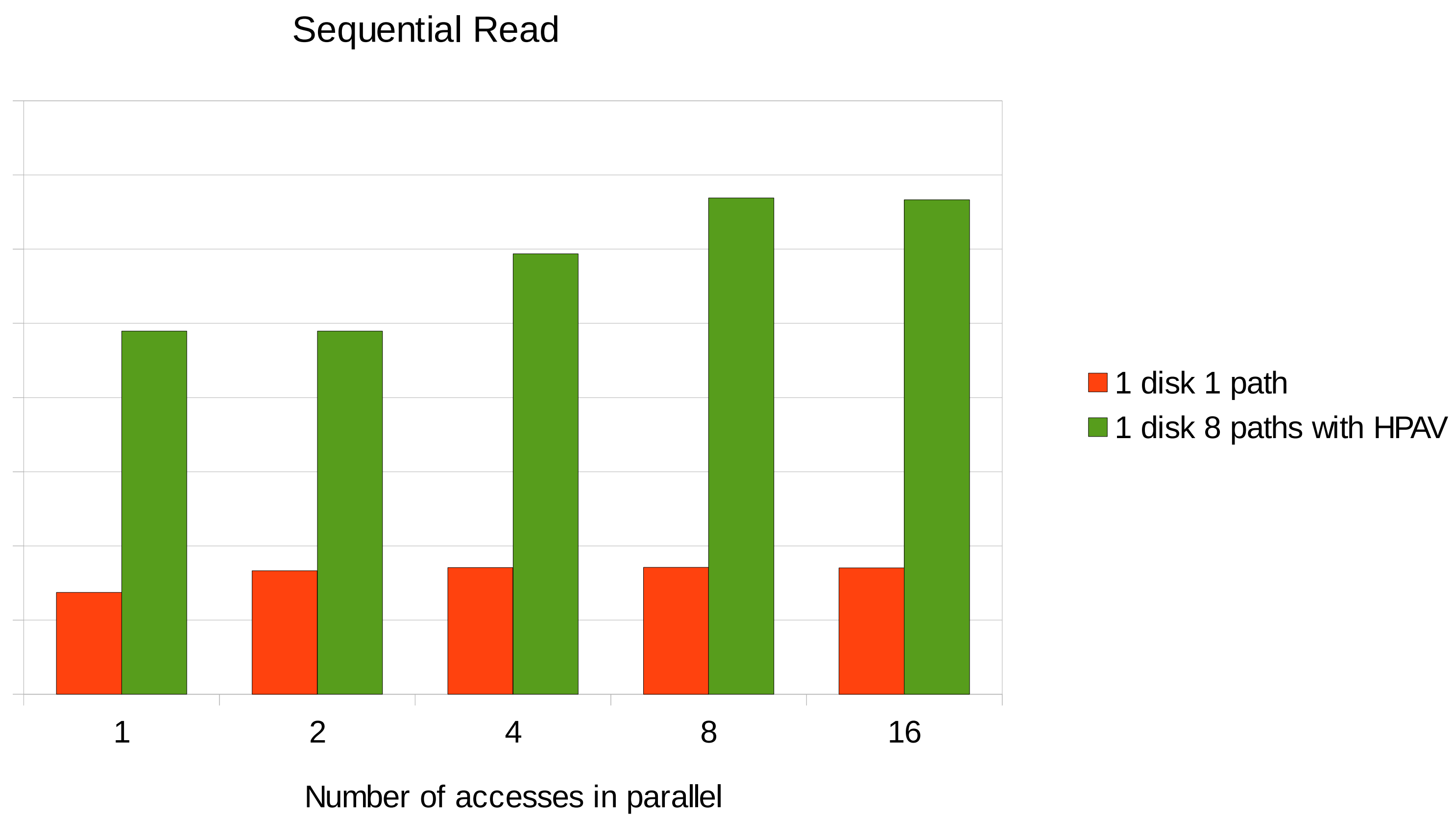
Disk I/O - striping options

- Striping is recommended and will result in higher throughput
  - Storage Pool Striped (SPS) disks with linear LV will perform better on many disk I/O processes
  - Device mapper striping on SPS disks will have good performance with few disk I/O processes

	Storage Pool Striping (SPS) or equivalent	Device mapper LV striping	No striping
Performance improvement	yes	yes	no
Processor consumption in Linux	no	yes	no
Complexity of administration	low	high	no

Disk I/O FICON / ECKD – number of paths in use

- Comparison of a single used subchannel to HyperPAV
  - Multiple (in example eight) paths perform much better
  - For reliable production systems you should use a multipath setup



Disk I/O FICON / ECKD – number of paths in use (cont.)

- iostat comparison (case 16 jobs in parallel)

```
...
04/10/14 23:52:20
Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
dasda             0.00     0.20    0.00    0.20     0.00     1.60    16.00     0.00    0.00    0.00    0.00   0.00   0.00
dasdb             0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
dasdc          2830.60     0.00  750.60    0.00 340915.20     0.00   908.38    36.06   48.03   48.03    0.00   1.33 100.00
…
```

```
...
04/11/14 01:15:31
Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
dasda             0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
dasdb             0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
dasdc         10243.20     0.00 2700.40    0.00 1229968.00     0.00   910.95    32.87   12.16   12.16    0.00   0.34  92.20
...
```

Disk I/O FICON / ECKD – number of paths in use (cont.)

- DASD statistics comparison (case 16 accesses in parallel)
- One CCW program must be finished before the next can executed in one path case
  - DASD driver queue size limited to maximal five entries
    - *First table shows the distribution in statistics of one to five requests queued*
- When more paths are used the requests gets distributed and parallel execution is possible
  - No more limitation to maximal five entries
    - *Second table shows a distribution in statistics with up to seventeen requests queued*
    - *Most of the time eight to twelve requests queued*

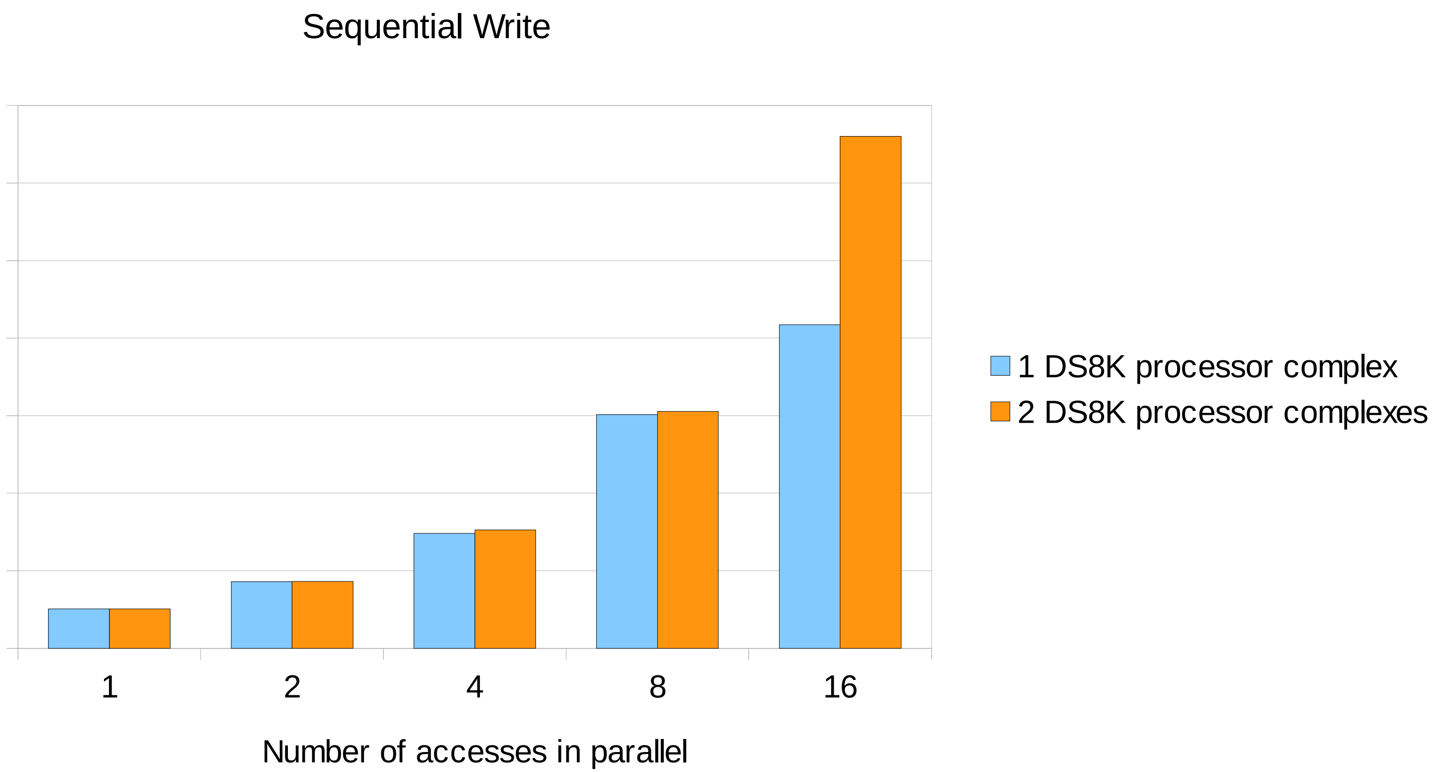
```
14513 dasd I/O requests
with 13108456 sectors(512B each)
Scale Factor is  1
  __<4    ___8   __16    __32   __64   _128   _256   _512   __1k   __2k   __4k   __8k   _16k   _32k   _64k   128k
  _256    _512   __1M    __2M   __4M   __8M   _16M   _32M   _64M   128M   256M   512M   __1G   __2G   __4G   _>4G


# of req in chanq at enqueuing (1..32)
     0      29   5396    7643   1445      0      0      0      0      0      0      0      0      0      0      0
     0       0      0       0      0      0      0      0      0      0      0      0      0      0      0      0
```

```
...
# of req in chanq at enqueuing (1..32)
     0      14      8      28     95     85    181   1265   2958   3329   3755   1796    620    126     28     18
     9       0      0       0      0      0      0      0      0      0      0      0      0      0      0      0
...
```

Disk I/O FICON / ECKD – usage of DS8K processor complexes

- Comparison one DS8K processor complex versus both processor complexes with LVM and HyperPAV
  - Recommendation if throughput matters: redistribute workload over both processor complexes
  - Write performance depends on available non-volatile write cache (NVS)



Disk I/O FICON / ECKD – usage of DS8K processor complexes

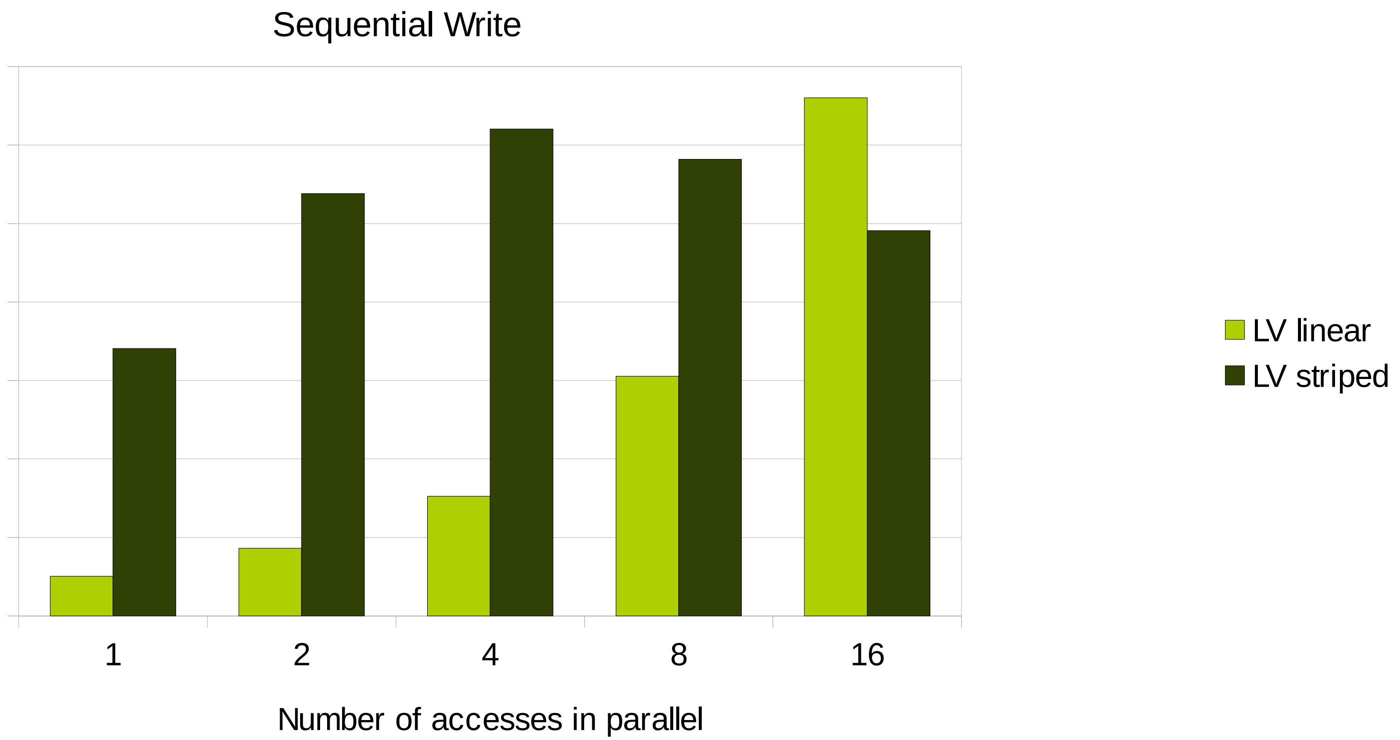
- Run iostat using command "`iostat -xtdk 10`"
- iostat results for sequential write using one DS8K processor complex compared to both processor complexes (16 streams write in parallel)
  - Much more throughput for both processor complexes with more NVS available
  - Less await and service time with both processor complexes

```
04/11/14 04:29:07
Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
dasda             0.00     0.20    0.00    0.20     0.00     1.60    16.00     0.00    0.00    0.00    0.00   0.00   0.00
...
...
dasddz            0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
dm-0              0.00     0.00    0.00 15577.60     0.00 1482777.60   190.37   139.00    9.41    0.00    9.41   0.06 100.00
...
```

```
04/11/14 20:58:22
Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
dasda             0.00     0.00    0.00    0.20     0.00     0.80     8.00     0.00    0.00    0.00    0.00   0.00   0.00
...
...
dm-0              0.00     0.00    0.00 33563.60     0.00 3194752.00   190.37   161.00    4.80    0.00    4.80   0.03  98.60
```

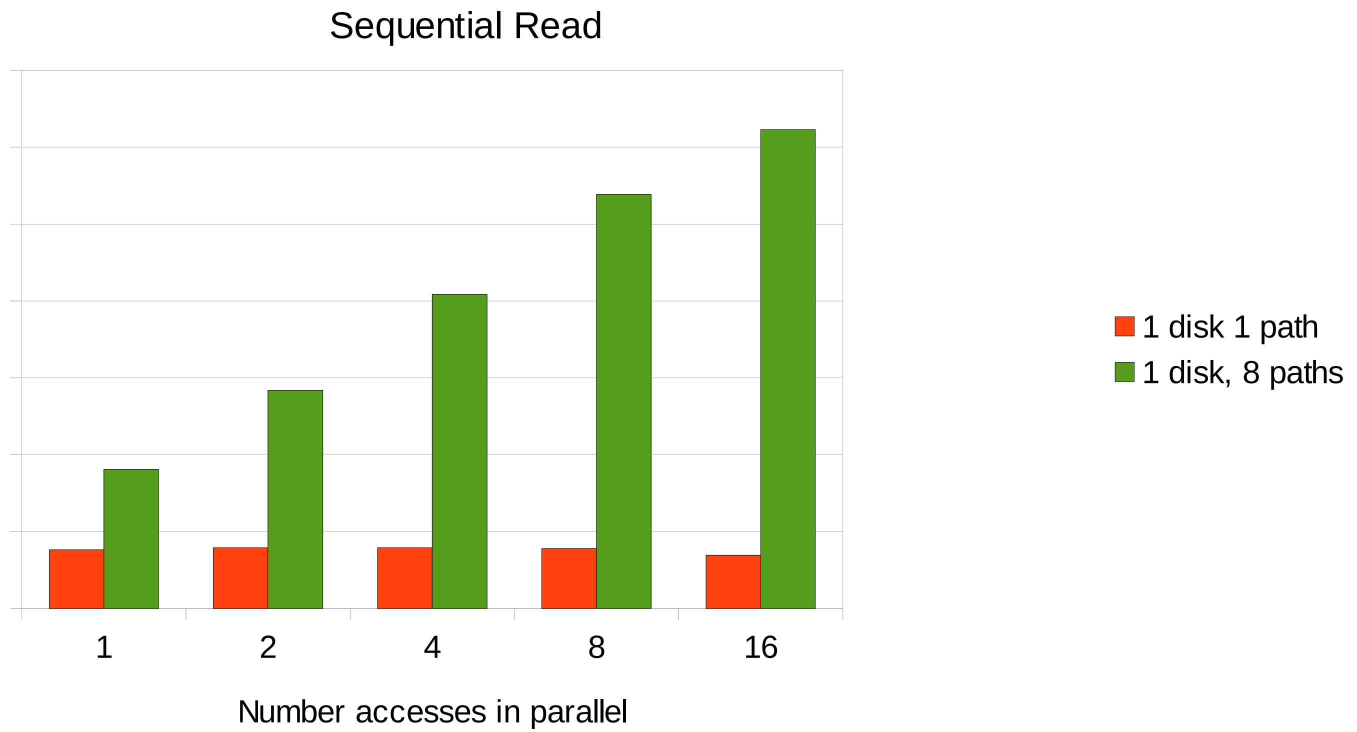
Disk I/O FICON / ECKD - LVM linear versus LVM striped

- Comparison Logical Volume linear versus Logical Volume striped
  - Much more parallelism when using striping with a few jobs running
  - Striping with sizes of 32kiB / 64 kiB may split up single big I/Os (bad)
    - *This applies especially to sequential workloads where read-ahead scaling take place*
  - Striping adds extra effort / processor consumption to the system
    - *Eventually can consume the benefits of striping by cpu induced latencies*



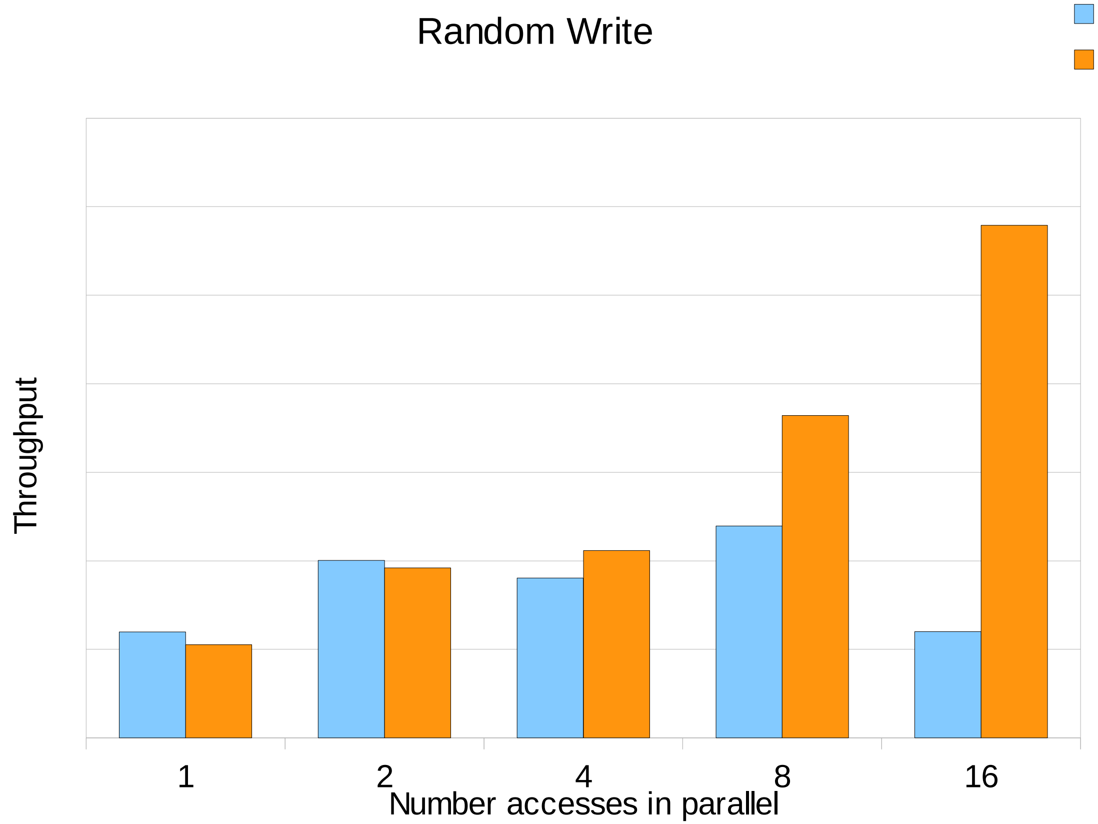
Disk I/O FCP / SCSI – number of paths in use

- Comparison single path setup to many paths
  - Multipath solution allows much more throughput
    - *Multipath requires some extra processor cycles*
  - Similar to comparison single subchannel versus HyperPAV
- For reliable production systems you should use a multipath setup anyway
  - Failover does not increase the capacity available to a path group, while multibus does



Disk I/O FCP / SCSI - usage of DS8K processor complexes

- Comparison usage of one processor complex versus both processor complexes with LVM
  - Usage of both processor complexes has an advantage if NVS became the limiting factor



Disk I/O – more tuning options

- Use latest hardware if throughput is important
  - Currently FICON Express 8S
- Use direct I/O and asynchronous I/O
  - Requires support by your used software products
  - More throughput at less processor consumption
  - In most cases advantageous if combined
- Use advanced FICON/ECKD techniques such as
  - High Performance FICON
  - Read Write Track Data
- Use the FCP/SCSI datarouter technique for further speedup (~5-15%)
  - Kernel parmline zfcp.datarouter=1, default on in more recent distribution releases
  - Requires 8S cards or newer
    - *Feature similar to the store-forward architecture of recent OSA Cards*
  - Allows the driver to avoid extra buffering in the card
    - *No in card buffering also means there can't be a stalling buffer shortage*

Disk I/O – performance considerations summary

- Use as much paths as possible
  - ECKD logical path groups combined with HyperPAV
  - SCSI Linux multipath multibus
- Use all advanced software, driver and Hardware features
- Storage Server
  - Use Storage Pool Striping (SPS) as a convenient tool
  - Define extent pools spanning over many ranks
  - Use both storage server complexes of the storage server (DS8x00)
- If you use Logical Volumes (LV)
  - Linear: with SPS and random access
  - Linear: with SPS and sequential access and many processes
  - Striped: for special setups that proved to be superior to SPS

- **So long story short: let nothing idle and use all you've got**

Agenda

• Disk performance	approximately 55% of external support requests
• **Network performance**	approximately 25% of external support requests
• Compiler	two ISVs and one of the biggest logistic companies
• Huge pages	beneficial in almost every huge installation

Network performance tuning

- It's not that hard actually...

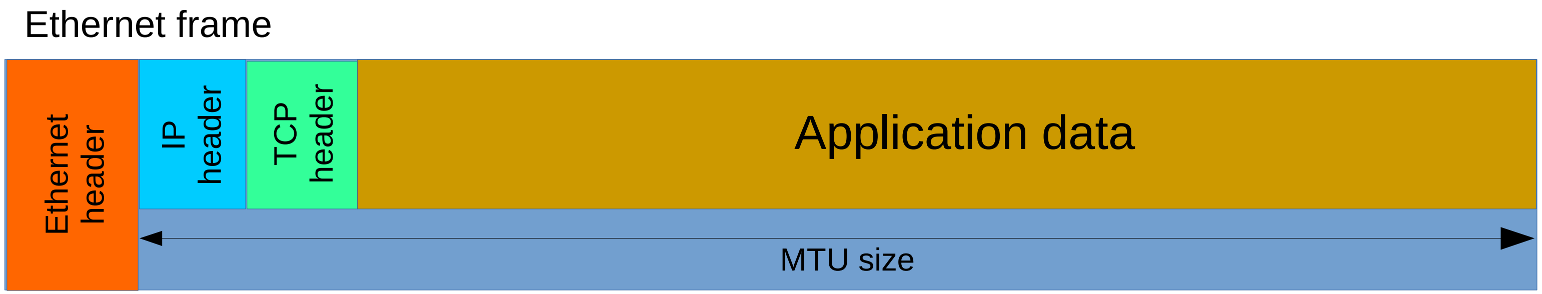
```
net.core.netdev_max_backlog = 25000
net.ipv4.tcp_dsack = 1
net.ipv4.tcp_sack = 1
net.core.somaxconn = 1024
net.ipv4.tcp_window_scaling = 1
net.ipv4.tcp_max_syn_backlog = 10000
net.ipv4.ip_local_port_range = 15000 65000
net.ipv4.tcp_timestamps = 1
net.ipv4.tcp_rmem = 4096 87380 524288
net.ipv4.tcp_fin_timeout = 1
net.core.rmem_max = 524288
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_tw_reuse = 1
net.core.wmem_max = 524288
net.ipv4.tcp_wmem = 4096 16384 524288
```

Network performance tuning

- But seriously...
- We won't go into all the gritty details here
  - Instead, we're going to introduce you to the concepts you can use to improve your network performance
  - If you really want to get into all the details (and especially *how* to do it), there are slides that go into that in the appendix of this presentation

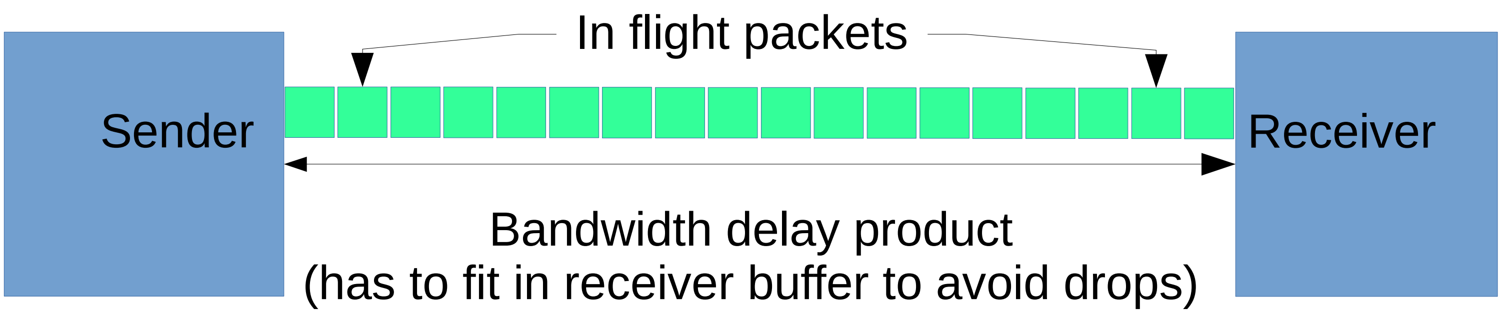
Tuning parameters - MTU size

- The maximum size usable for payload data in a single IP packet
  - Minus protocol headers
- The default for Ethernet is 1500
  - 1492 for OSA in layer 3 mode
- You can increase this to reduce segmentation overhead and thus CPU cycles
  - Those frames are called “jumbo frames”
  - Your infrastructure (switches, routers, …) must support those
  - Normally up to 9000, for OSA in layer 3 mode up to 8992
- Ideally, your MTU should not exceed the MTUs used on all the hops your packets pass through on their way to their target



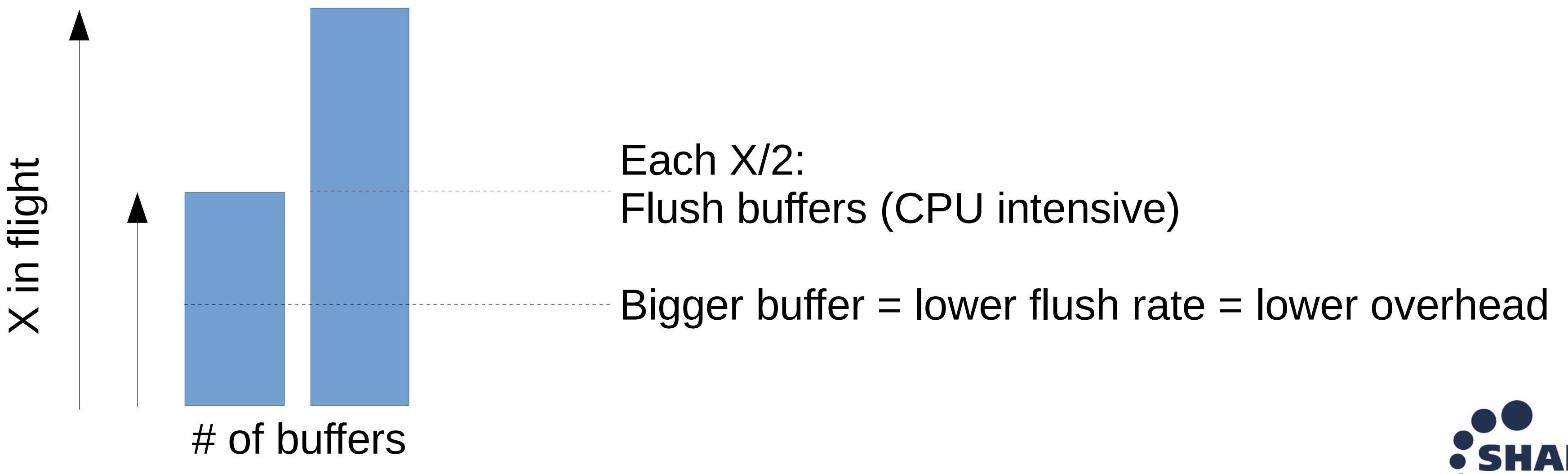
Tuning parameters - send / receive buffer size

- Buffer packets to accommodate for bandwidth mismatches between sender and receiver
  - Both could be a source of latencies if they are not drained fast enough (buffer bloat)
- Linux automatically manages the size of these buffers
  - You can set some bounds respected by the auto-tuning mechanism
- Depending on your scenario, bigger or smaller buffers work better
  - HiperSockets vs. OSA
    - *For HiperSockets with a MTU > 8000, the buffer size should not exceed 524288*
    - *For OSA, larger buffer sizes like 4194304 are preferred for optimal performance*
  - LAN vs. WAN
    - *Generally, if either your link speed or your round-trip latency (or both) increases, you'll need bigger buffers (based on the bandwidth delay product).*



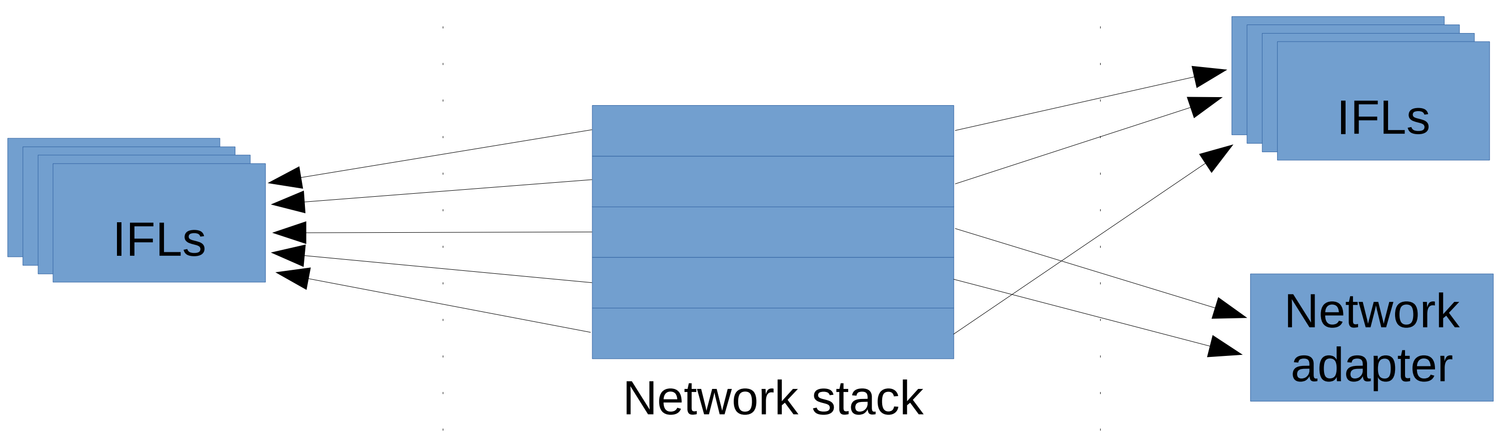
Tuning parameters - OSA inbound buffer count

- You can limit the number of buffers the OSA adapter uses for inbound connections
- The default here is 64
- For maximum performance, this should be increased to 128
- Caveat: this increases your memory consumption by 64 KiB per additional buffer



Tuning parameters - offloads

- Most network cards support some kind of hardware offloads
- Those shift work from the CPU to the network card itself
- The two most prominent here are TCP segmentation offload (TSO) and generic receive offload (GRO)
- It is advisable to enable those
  - Caveat: TSO only works for physical adapters in layer 3 mode
- Another relevant one would be TX and RX checksumming



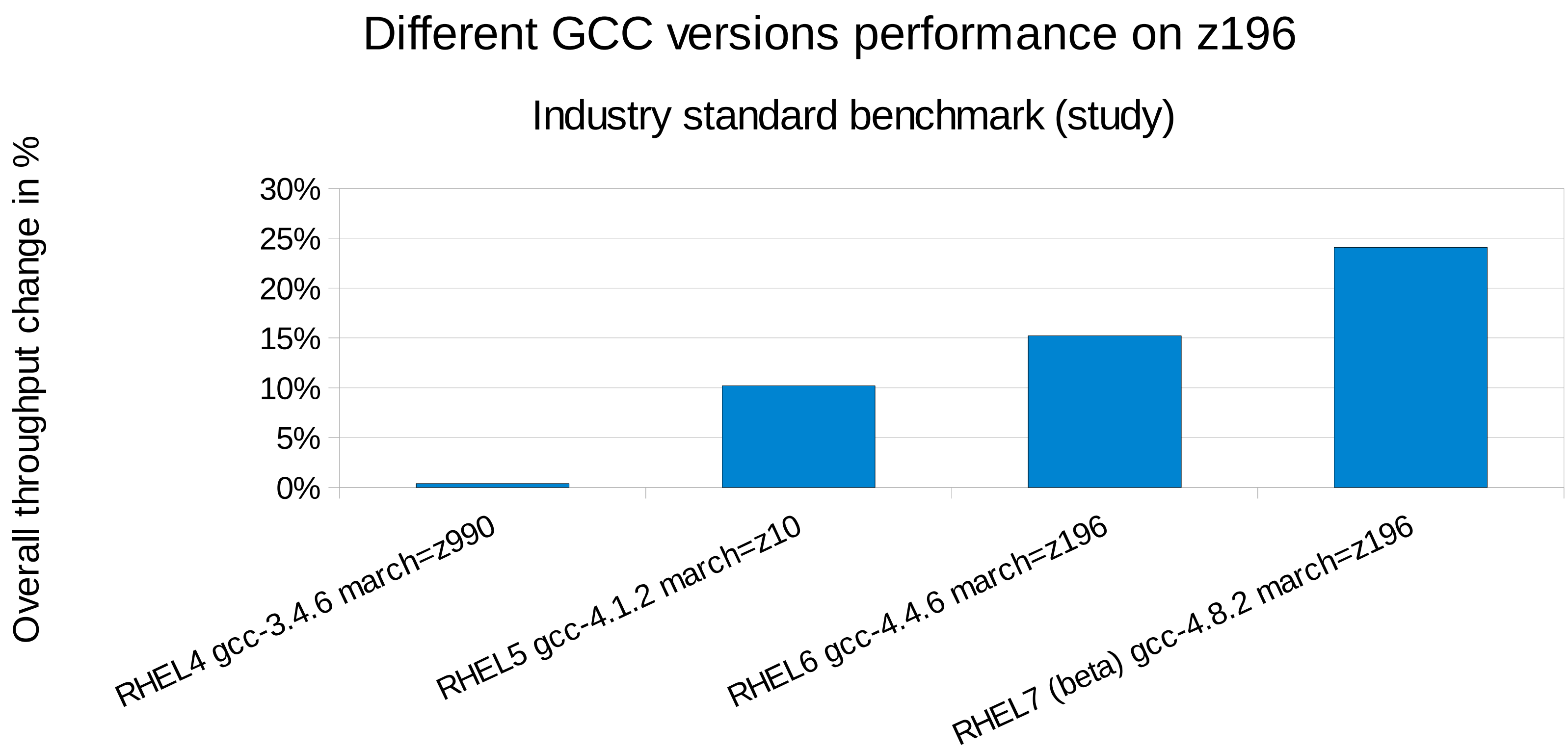
Without offloading

With offloading

Agenda

- Disk performance — approximately 55% of external support requests
- Network performance — approximately 25% of external support requests
- **Compiler** — two ISVs and one of the biggest logistic companies
- Huge pages — beneficial in almost every huge installation

GCC evolution



- Advantages of using current compilers are significant
  - Improved machine support is introduced with newer GCC versions
    - *Distributors often back-port patches*
  - Applications of different characteristics will show different throughput changes when using a newer compiler

GCC versions in Linux on System z supported distributions

GCC stream	x.y.0 release	Included in SUSE distribution	Included in Red Hat distribution
GCC-3.3	05/2003	SLES9 (z990 backport)	n/a
GCC-3.4	04/2004	n/a	RHEL4 (z990 support)
GCC-4.0	04/2005	n/a	n/a
GCC-4.1	02/2006	SLES10 (z9-109 support)	RHEL5 (z9-109 support)
GCC-4.2	05/2007	n/a	n/a
GCC-4.3	05/2008	SLES11 (z10 backport)	n/a
GCC-4.4	04/2009	n/a	RHEL6.1 / 5.6** (z196 backport)
GCC-4.5	04/2010	SLES11 SP1	n/a
GCC-4.6	03/2011	SLES11 SP2 (z196 support)*	n/a
GCC-4.7	03/2012	SLES11 SP3 (z196 support)*	n/a
GCC-4.8	03/2013	SLES12 (zEC12 support)****	RHEL7 (zEC12 support)***
GCC-4.9	04/2014	n/a	n/a

* included in SDK, optional, not fully supported
** fully supported add-on compiler
*** as announced for RHEL7 beta by Red Hat (Dec 2013)
**** as seen in SLES12 beta

Optimizing C and C++ code

- Produce optimized code
  - Options -O3 or -O2 (often found in delivered makefiles) are a good starting point and are used in most frequently in our performance measurements
  - Optimize GCC instruction scheduling with the performance critical target machine in mind using -mtune parameter
    - *-mtune=values <z900, z990 with all supported GCC versions>*
    - *<z9-109 with gcc-4.1>*
    - *<z10 with SLES11 gcc-4.3 or gcc-4.4>*
    - *<z196 with RHEL6 gcc-4.4, optional SLES11 SP1 gcc-4.5*, or GNU gcc-4.6>*
    - *<zEC12 with GNU gcc-4.8>*
  - Exploit also improved machine instruction set and new hardware capabilities using the -march parameter
    - *-march=values <z900, z990, z9-109, z10, z196, zEC12> available with the same compilers as mentioned above*
    - *Includes implicitly -mtune optimization if not otherwise specified*
    - *-march compiled code will only run on the target machine or newer machines*

* not fully supported version

GCC compile options

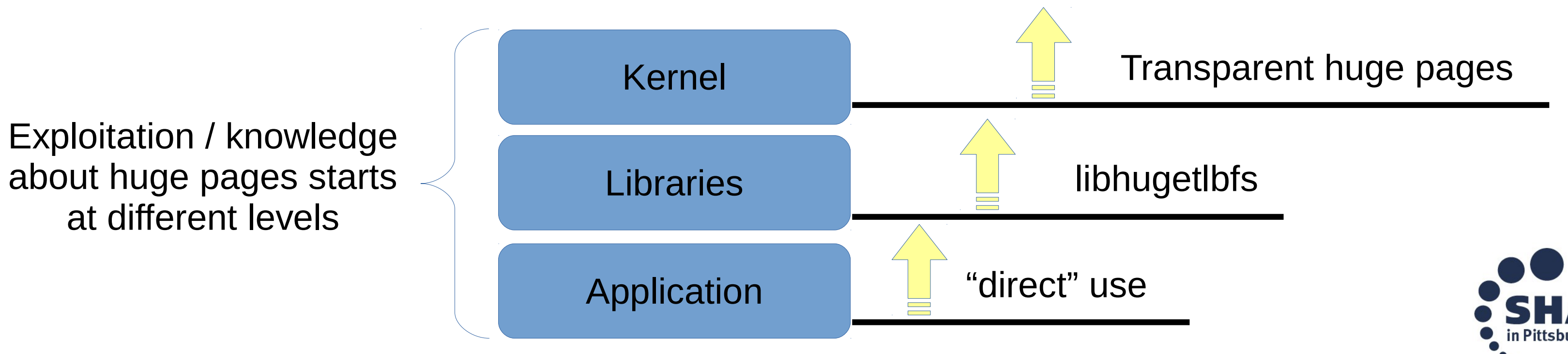
- Fine Tuning: additional general options on a file by file basis
  - -funroll-loops often has advantages on System z
    - *Unrolling is internal delimited to a reasonable value by default*
  - Use of inline assembler for performance critical functions may have advantages
  - -ffast-math speeds up calculations (if not exact implementation of IEEE or ISO rules/specifications for math functions is needed)
  - -fno-strict-aliasing helps to overcome code flaws detected with newer compiler versions

Agenda

• Disk performance	approximately 55% of external support requests
• Network performance	approximately 25% of external support requests
• Compiler	two ISVs and one of the biggest logistic companies
• **Huge pages**	beneficial in almost every huge installation

Huge pages – three kinds of exploitations

- Huge Pages exploited directly by applications
  - Common exploiters using this approach are Java, Databases and other common huge memory consumers
- Huge pages exploited via libhugetlbfs
  - Common exploiters using this approach are administrators who force an application to use huge pages without change to the application itself
- Huge Pages exploited via transparent huge pages
  - Common exploiters are full system environments starting with the given releases



Huge pages – three kinds of availability

- Huge Pages exploited directly by applications
  - hugetlbfs support available from kernel 2.6.26 on (SLES 11, RHEL 6)

- Huge pages exploited via libhugetlbfs
  - For libhugetlbfs System z support started with version 2.15 (SLES11-SP3, RHEL7*)

- Huge Pages exploited via transparent huge pages
  - Allows transparent access to huge pages for any application
  - Linux on System z support starting with kernel 3.7
    - *recommended usage starting with kernel 3.8*
    - *Expected to be available with RHEL 7* and SLES 12**
  - Check `/sys/kernel/mm/transparent_hugepage/*` in your live system

*part of current public beta program content

Huge pages – direct or via libtlbfs – 1. Preparation

- The kernel has to provide an amount of its memory as huge pages:
  - Configure nr_hugepages

    ```
    echo 2000 > /proc/sys/vm/nr_hugepages
    ```

  - To make this change boot-proof add entry in sysctl.conf

    ```
    sysctl -w vm.nr_hugepages=2000
    ```

    - *Could also be achieved via kernel parmline*

  - Mount hugetlbfs is only required by some applications, but never hurts

    ```
    mount -t hugetlbfs none /mnt/hugetlbfs
    ```

Huge pages – direct or via libtlbfs – 2. usage

- Applications that are coded to use huge pages need “their” parameters
  - e. g. Java enables huge pages via `-Xlp` for the Java Heap
    - *Starting with Java 7.1 huge pages are also used for classes*
  - e.g. for DB2 set ther vaiable `DB2_LARGE_PAGE_MEM *`

- libhugetlbfs enables other applications:
  - Is linked dynamically without requiring code changes and recompilation
  - Insert the library in the loading process exporting an LD_PRELOAD statement

    ```
    export LD_PRELOAD=libhugetlbfs.so
    ```
  - Check if the certification of software products covers the usage of libugetlbfs

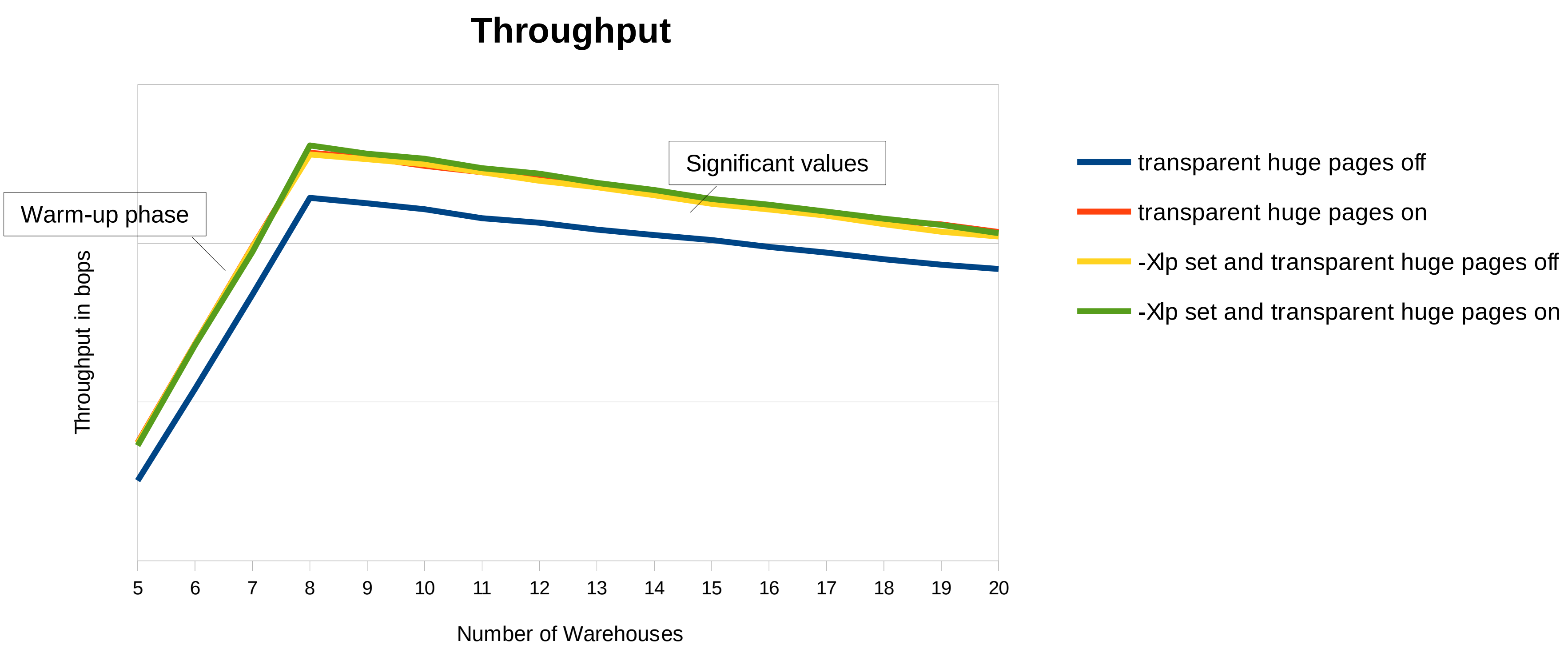
- For both the most important part is, check that they are really used
  - The memory is reserved for huge pages, if not used it is wasted
  - Often if an application “just doesn't fit” it falls back to normal for all its allocations

Huge pages – transparent usage

- Transparent huge pages is the striving of the kernel to back memory with huge pages
  - Can be swapped, although they have to be broken into 4k to do so
  - Are sensible to fragmentation, therefore there is a defrag daemon
  - The usage of huge pages is not guaranteed
  - All that management adds cpu overhead
    - *especially if fragmentation or swapping takes place*

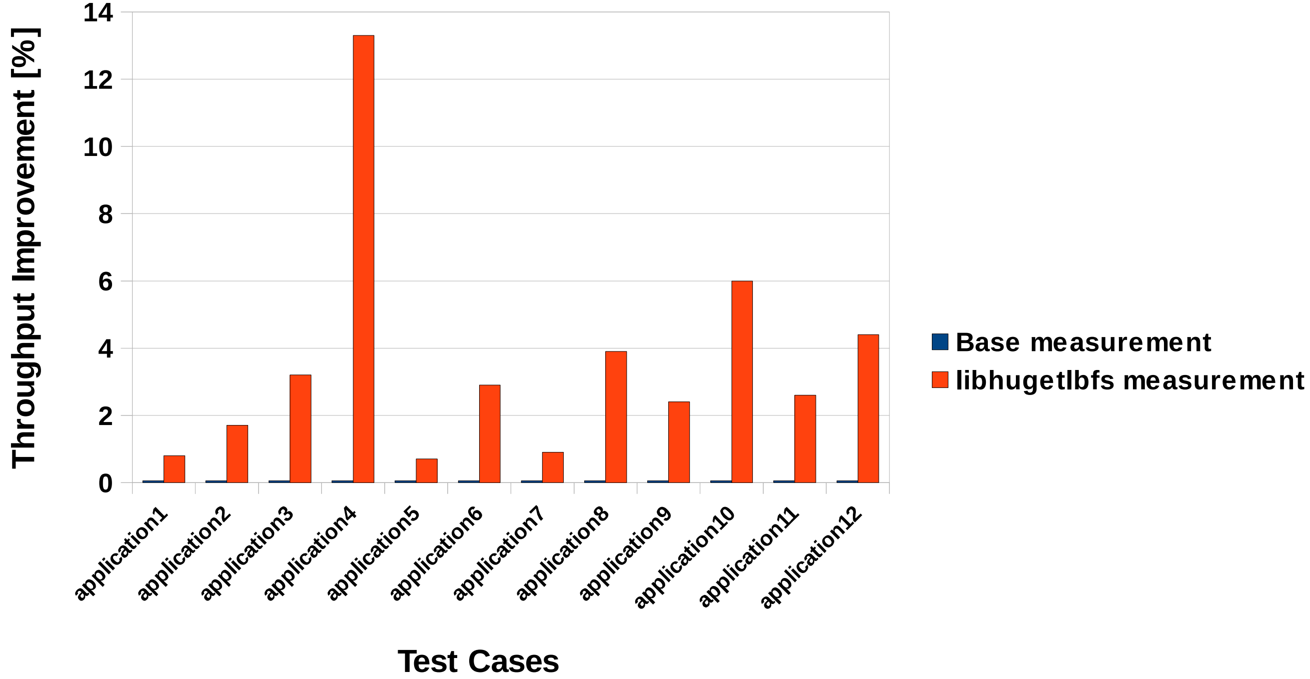
- Controlled via kernel parameter transparent_hugepage <never, **always**, madvise>
  - The default setting is "always"
  - Can be configured at runtime in `/sys/kernel/mm/transparent_hugepage/*`
  - Madvise affects only special regions where applications set `MADV_HUGEPAGE`
  - Comes most likely with RHEL 7 and SLES 12 (as seen in beta programs)

Huge pages for Java standard benchmark



- Usage of transparent huge pages doesn't conflict with direct usage of huge pages
  - Processor savings are comparable for all cases using huge pages (~ 5.5 %)
  - Usage of transparent huge pages yields ~ 5 % performance gain
  - direct usage of huge pages (-Xlp) results in approximately the same: ~ 5% performance gain

libhugetlbfs for compute intense integer benchmark



- Application had no native huge page code
- Usage of libhugetlbfs yields ~ 4 % overall performance gain
- All measured real life applications show a performance improvement
  - The degree of the performance improvement depends heavily on the characteristic and quantity of memory accesses
  - No tested application suffered from the usage of libhugetlbfs

Huge Pages for Oracle database memory

- Oracle Database uses many processes in parallel
- In general 10-15% can be gained by the reduction in processor usage as well as having a lot more memory for applications that would be consumed in Linux Page Tables
- The screen-shot shows that approximately 91GiB of memory were used for page tables without defined huge pages
  - At the same time system started slightly swapping
- Page tables were below 3G after switching to huge pages

```
procs -----------memory---------- ---swap-- -----io---- -system-- -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
338  8 1766820 1096980   1200 158901132    1  467 11419   721 2140 2724  1 93  0  0  7
125 13 1767088 1096700   1316 158896948    8  135  7199  1092 2227 4262  2 91  0  0  7
420  4 1767396 1073704   1416 158891792   17  137 18407 25048 5875 11215  6 80  4  5  |
302  5 1767588 1089200   1424 158876220    3  172  1256   329 1705 1483  0 93  0  0  6
227  7 1767652 1088700   1448 158870652    9   97  4889   361 1987 1926  1 92  0  0  7
165 16 1767796 1093696   1444 158858216    0  129  3617   605 2205 2874  2 91  0  0  7
452 16 1768980 1074352   1480 158858772   35  453 11801 14244 4667 8128  5 85  2  2  6
257 14 1769204 1096292   1276 158828368    5   84  1320   505 2066 2657  2 91  0  0  7
177  6 1769172 1098028   1320 158821092    0   20  1647   447 1761 1984  2 91  0  0  7
217 16 1769600 1095124   1364 158816144   19  224  2167  1055 2029 2703  2 91  0  0  7
144 17 1770068 1088160   1256 158814320   12  239  1760   659 1884 2295  2 91  0  0  7
122 11 1771576 1082412   1276 1[illegible]8810608   11  561  1817   868 1862 2049  2 92  0  0  7
219 10 1772768 1073684   1260 158807908   29  408  2385   863 2200 2916  2 91  0  0  7
315  3 2033292 1076748   1152 158561024  100 86901 21179 87940 45540 33283  0 93  0  0
```

```
SReclaimable:     386028 kB
SUnreclaim:       222484 kB
KernelStack:       16880 kB
PageTables:     91964268 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:   173377556 kB
Committed_AS:  214527304 kB
VmallocTotal:  134217728 kB
VmallocUsed:     2629972 kB
VmallocChunk:  131453796 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       1024 kB
```


SHARE in Pittsburgh 2014

Huge pages – usage considerations

- In Linux the terms “huge pages” and “large pages” are used synonymously
- Due to the fact that “normal” huge pages are not swappable they may increase pressure on memory management
  - If the system starts swapping frequently usage of huge pages may consume more processor cycles than saved by huge pages in the first place
- In LPAR
  - Decreased page table overhead by using hardware feature “Enhanced DAT”
- Under z/VM
  - z/VM does not support huge pages for its guests (EDAT)
  - Still Linux can “emulate” huge pages which still drops the page table sizes
    - *Can be useful for applications with a memory footprint > 10GB*
    - *Trade-off “cpu cycles for huge page emulation” for “page table size savings”*

Huge Pages - comparison and conclusion

	direct usage of huge pages (provided by application code)	**usage of huge pages via libhugetlbfs**	**Transparent huge pages**
Administration	Proper application configuration is administration effort	Properly setting LD_PRELOAD is administration effort	No extra effort
Certainty	Usage of huge pages guaranteed, once allocated	Usage of huge pages guaranteed, once allocated	Usage of huge pages if resources are available
Overhead	None	None	Defragmentation
Swap	Not swappable	Not swappable	Swappable

- Performance gains more or less equal, no matter which method is used
- Generally, transparent huge pages combine a lot of benefits: performance gain + low administration effort
  - Usage of transparent huge pages doesn't conflict with direct usage of huge pages
  - Usage of libhugetlbfs is also beneficial but can't compete with the advantages of transparent huge pages
  - Watch out for support statements of Software regarding libhugetlbfs and transparent huge pages
- Any of them is better than not using huge pages at all
  - One has to evaluate for his own benefit and conditions

Questions ?

- Further information is available at
  - Linux on System z – Tuning hints and tips
    http://www.ibm.com/developerworks/linux/linux390/perf/index.html
  - Live Virtual Classes for z/VM and Linux
    http://www.vm.ibm.com/education/lvc/

Christian Ehrhardt

Linux on System z
System Software
Performance Engineer

IBM Deutschland Research & Development
Schoenaicher Strasse 220
71032 Boeblingen, Germany

Phone +49 (0)7031–16–3385
Email ehrhardt@de.ibm.com

Backup

Network tuning details

- Here you'll find the detailed descriptions on how to adjust the network tuning parameters talked about earlier in this presentation

Tuning parameters - inbound buffer count

- For servers with high network traffic the OSA inbound buffer count should be increased to 128 to gain maximum performance
  - Default inbound buffer count is 64
  - Check actual buffer count with `lsqeth -p` command
  - We observed that the default of 64 limits the throughput of a HiperSockets connection with 10 parallel sessions and more
  - A buffer count of 128 leads to 8MiB memory consumption per device
    - One buffer consists of 16 x 4KiB pages which yields 64KiB => 128 x 64KiB = 8MiB
- Set the inbound buffer count in the appropriate config file
  - `SUSE SLES10: in /etc/sysconfig/hardware/hwcfg-qeth-bus-ccw-0.0.F200 add QETH_OPTIONS="buffer_count=128"`
  - `SUSE SLES11: in /etc/udev/rules.d/51-qeth-0.0.f200.rules add ACTION=="add", SUBSYSTEM=="ccwgroup", KERNEL=="0.0.f200", ATTR{buffer_count}="128"`
  - `Red Hat RHEL5/6/7: in /etc/sysconfig/network-scripts/ifcfg-<device> add OPTIONS="buffer_count=128"`

Tuning parameters - MTU size

- Choose your MTU size carefully. Set it to the maximum size supported by all hops on the path to the final destination to avoid fragmentation.
  - Use `tracepath <destination> command` to detect max MTU size
  - Example shows router at `192.168.111.1` with MTU 1200

```
[root@x5perf2 ~]# tracepath 192.168.112.2
 1:  192.168.111.2                                  0.096ms pmtu 1500
 1:  192.168.111.1                                  0.248ms
 1:  192.168.111.1                                  0.269ms
 2:  192.168.111.1                                  0.224ms pmtu 1200
 2:  192.168.112.2                                  0.310ms reached
```

If the application sends in chunks of <=1460 bytes, use MTU 1500
    - 1460 Bytes user data plus protocol overhead
  - If the application is able to send bigger chunks, use MTU 8992
    - Sending packets > 1460 bytes with MTU 8992 will increase throughput and save processor cycles

- For VSWITCH, MTU 8992 is recommended
  - Synchronous operation, SIGA required for every packet
  - No packing like normal OSA cards
  - No tso, tx-checksumming and rx-checksumming offloading

General tuning parameters (1/4)

- Command based tunings like ip and ethtool must be made persistent to survive a reboot
- System wide sysctl settings can be changed temporarily by the sysctl command or permanently in /etc/sysctl.conf
- System wide window size applies to all network devices
  - Applications can use setsockopt to adjust the window size for one device
    - Has no impact on other network devices
    - Disables window scaling which may have negative impact on throughput

General tuning parameters (2/4)

- Set the device transmission queue length from the default of 1000 to 3000
  - `ip link set <interface_name> txqueuelen 3000`

- Following settings do not necessarily fit to every environment and are just a starting point based on our experience
  - Increase the processor input packet queue length from the default of 1000
    - `net.core.netdev_max_backlog = 25000`
  - Increase the maximum number of requests queued to a listen socket, default is 128
    - `net.core.somaxconn = 1024`
    - Can be too much if the server cannot handle (watch CPU utilization in sadc)
  - Prevent SYN packet loss
    - `net.ipv4.tcp_max_syn_backlog = 10000`
    - Only meaningful if `net.core.somaxconn` is increased as well

General tuning parameters (3/4)

- Consider to increase port range for outgoing ports
  - If server software binds to ports > 14999 adjust minimum value accordingly
  - `net.ipv4.ip_local_port_range = 15000 65000`

- Settings to observe if you have a lot of sockets / connections sitting in TIME_WAIT state (specified in seconds)
  - Can be checked while an application is running by command

    ```
    netstat -tan | awk '{print $6}' | sort | uniq -c
    ```

    - `net.ipv4.tcp_fin_timeout = 1        < 1 for LAN|6 for WAN >`
  - Reuse active connection if application and protocol would allow it
    - `net.ipv4.tcp_tw_reuse = 1`
  - Make socket re-usable by switching on fast recycle
    - `net.ipv4.tcp_tw_recycle = 1`
  - Its worth trying settings `net.ipv4.tcp_tw_reuse` and `net.ipv4.tcp_tw_recycle` together

General tuning parameters (4/4)

- TCP Extensions for High Performance as described by RFC1323, RFC2018 and RFC2883
  - Following parameters are enabled by default and should only be changed for a reason
    - `net.ipv4.tcp_window_scaling = 1`
    - `net.ipv4.tcp_timestamps = 1`
    - `net.ipv4.tcp_sack = 1`
    - `net.ipv4.tcp_dsack = 1`

Note: Linux sysctl settings are system wide and apply to all network devices

HiperSockets recommendations (1/2)

- Frame size and MTU size are determined by chparm parameter of the IOCDS
  - Calculate MTU size = frame size – 8KiB
- Select the MTU size to suit the workload
  - If the application is mostly sending packets < 8KiB an MTU size of 8KiB is sufficient
- If the application is capable of sending big packets, a larger MTU size will increase throughput and save processor cycles
- MTU size 56KiB is recommended only for streaming workloads when application is able to send packets > 32KiB

- HiperSockets and OSA devices have contradictory demands regarding maximum send /receive size and autotuning buffer
  - For environments with OSA and HiperSockets trade-offs have to be made
  - Suggested values for OSA devices (on page 28) are also applicable for HiperSockets MTU 8KiB
  - HiperSockets MTU sizes > 8KiB require smaller settings
  - Maximum autotuning buffer size should not exceed 524288 bytes

HiperSockets recommendations (2/2)

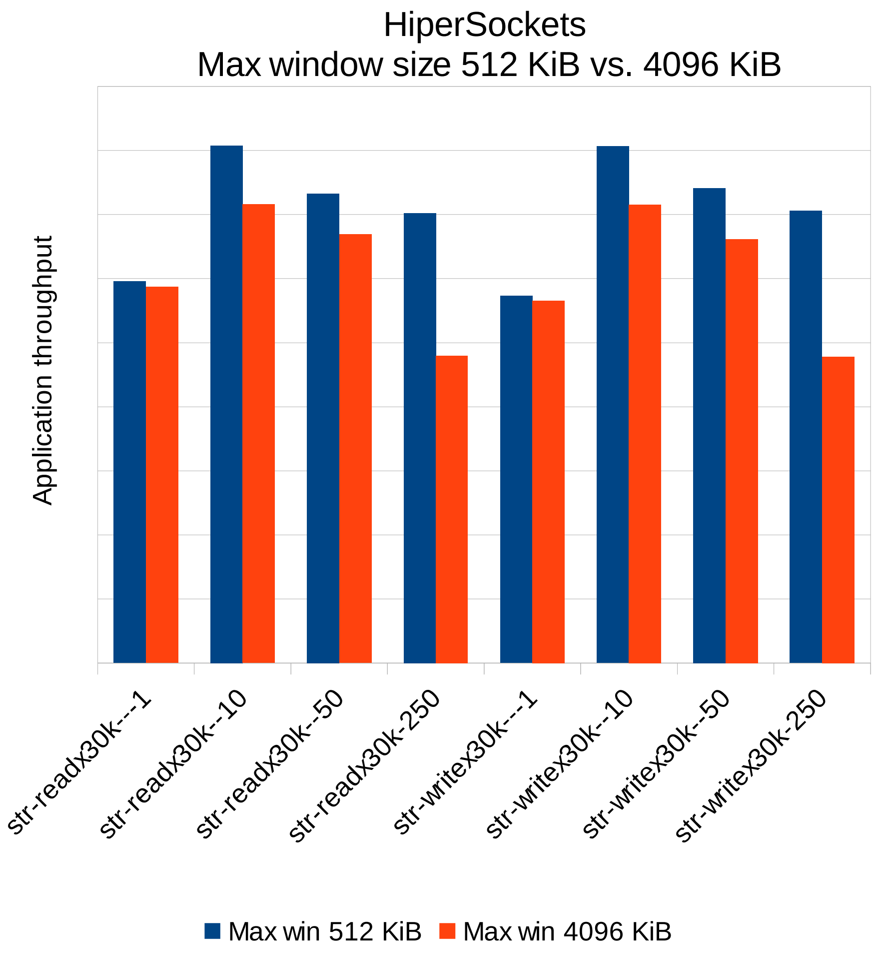
- Maximum socket send / receive buffer size which may be set by using the SO_SNDBUF / SO_RCVBUF socket option
  - `net.core.wmem_max = 524288`
  - `net.core.rmem_max = 524288`
- Set Linux maximum send / receive window size
  - Does not override `net.core.wmem_max` and `net.core.rmem_max`
  - Higher maximum window size leads to throughput degradation if MTU > 8 KiB
  - `net.ipv4.tcp_wmem = 4096 16384 524288`
  - `net.ipv4.tcp_rmem = 4096 87380 524288`
- Applications can use setsockopt to adjust the window size individually
  - Has no impact on other network devices
  - Disables window scaling which may have negative impact on throughput

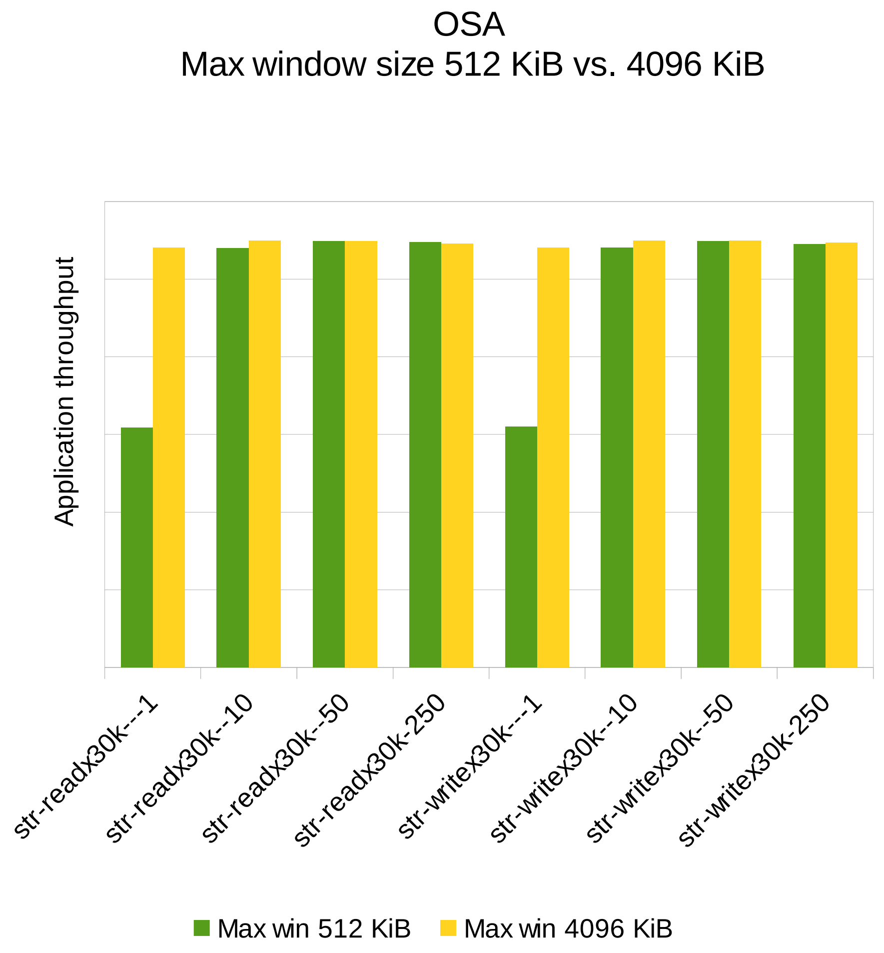
OSA recommendations

- Maximum socket send / receive buffer size which may be set by using the SO_SNDBUF / SO_RCVBUF socket option
  - `net.core.wmem_max = 4194304`
  - `net.core.rmem_max = 4194304`
- Set Linux maximum send / receive window size (default in current distributions)
  - A higher Bandwidth Delay Product BDP (data in flight) requires higher window size settings
  - In a low latency LAN (low BDP) with a massive amount of parallel sessions lower values might be an advantage
  - Does not override `net.core.wmem_max` and `net.core.rmem_max`
  - `net.ipv4.tcp_wmem = 4096 16384 4194304`
  - `net.ipv4.tcp_rmem = 4096 87380 4194304`

Performance implications of window size

- HiperSockets: Smaller window size improves throughput
- OSA: Bigger window size improves throughput





OSA - TCP segmentation offload (TSO)

- TCP Segmentation Offload (TSO) moves the effort of cutting application data in MTU sized packets from the TCP stack to the OSA hardware
  - Does not affect packets < MTU size
- Network device must support outbound (TX) checksumming and scatter gather (SG)
  - Only in Layer3 mode and physical adapters (OSA in LPAR or direct attached in z/VM)
  - Turn on scatter gather and outbound checksumming prior to configuring TSO
  - Turn on or off with a single ethtool command

```
# ethtool -K <interface_name> tx <on|off> sg <on|off> tso <on|off>
Example
# ethtool -K <interface_name> tx on sg on tso on
```

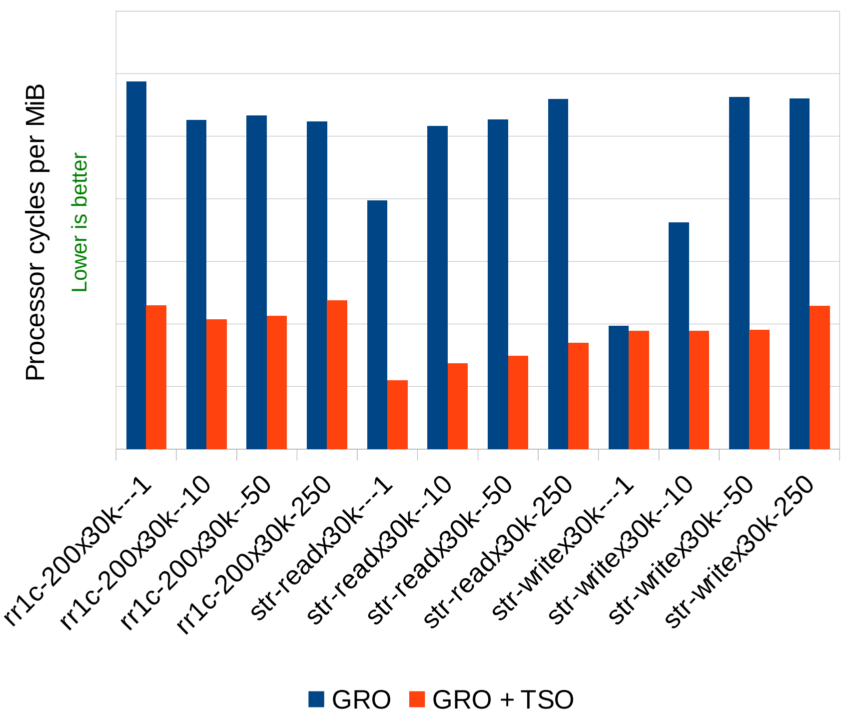
- When TCP segmentation is offloaded, the OSA feature performs the calculations
  - Applies only to packets that go out to the LAN
- When Linux instances are communicating via a shared OSA port the packages are forwarded by the OSA adapter but do not go out on the LAN
  - Exchange packages directly and no TCP segmentation calculation is performed
  - All TSO packets are dropped without warning because the qeth device driver cannot detect this

OSA – Generic Receive Offload (GRO)

- Generic Receive Offload (GRO) aggregates multiple incoming packets into a larger buffer before they are passed higher up the networking stack
  - Thus reducing the number of packets that have to be processed
  - Default since SLES11 and RHEL6
- Throughput improvement at combined usage of GRO and TSO
- Tremendously less processor cycles needed at combined usage of GRO and TSO

Throughput

Processor consumption



OSA recommendations – priority queueing

- Consider to switch on priority queueing if an OSA Express adapter in QDIO mode is shared amongst several LPARs
  - Queues 0 to 3 can be used whereby queue 2 is used as default
  - Queues are served in ascending order, queue 0 has highest priority
- How to activate
  - `SUSE SLES10: in /etc/sysconfig/hardware/hwcfg-qeth-bus-ccw-0.0.F200 add QETH_OPTIONS="priority_queueing=no_prio_queueing:0"`
  - `SUSE SLES11: in /etc/udev/rules.d/51-qeth-0.0.f200.rules add ACTION=="add", SUBSYSTEM=="ccwgroup", KERNEL=="0.0.f200", ATTR{priority_queueing}="no_prio_queueing:0"`
  - `Red Hat RHEL5/6: in /etc/sysconfig/network-scripts/ifcfg-eth0 add OPTIONS="priority_queueing=no_prio_queueing:0"`
- Select on the most important stack only
  - Priority queueing on one LPAR may impact the performance on other LPARs sharing the same OSA card

SAP Enqueue Server recommendations

- SAP networking is a transactional type of workload with a packet size < 8KB
- SAP Enqueue Server requires a proper set default send window size of 4 x MTU size
  - Required because of sub optimal return code checking
- HiperSockets
  - MTU 8192 is sufficient (default 4 x 8192 = 32768)

```
net.ipv4.tcp_wmem = 4096 32768 4194304
net.ipv4.tcp_rmem = 4096 87380 4194304
```

- OSA
  - Recommended MTU size is 8192 (default 4 x 8192 = 32768)
  - Alternatively if MTU size 8992 is used (default 4 x 8992 = 35968)