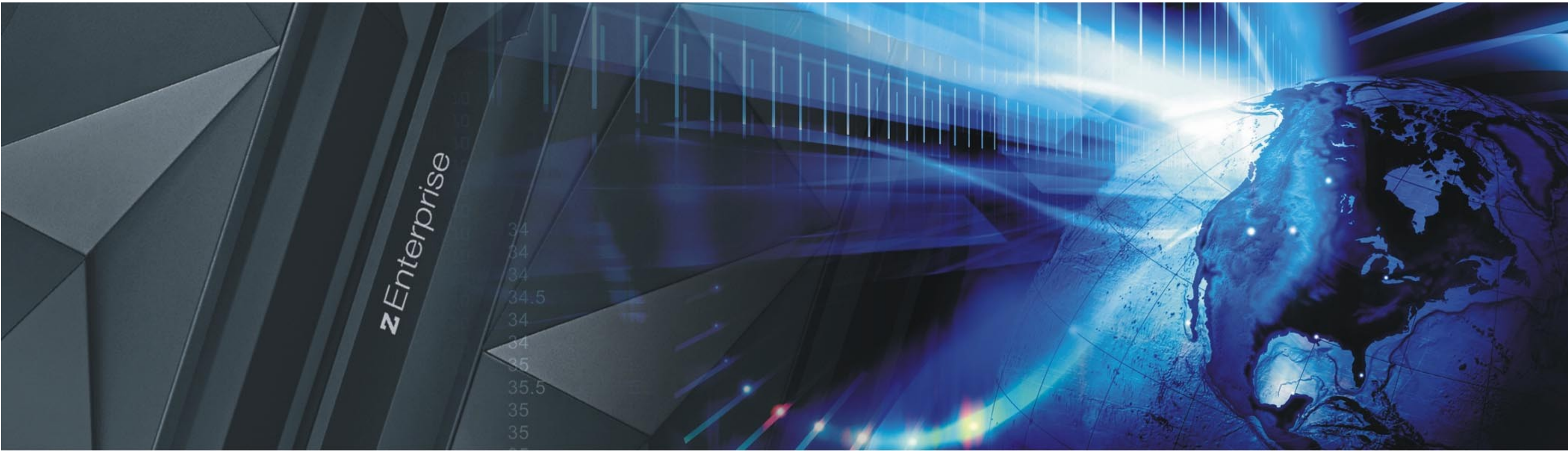
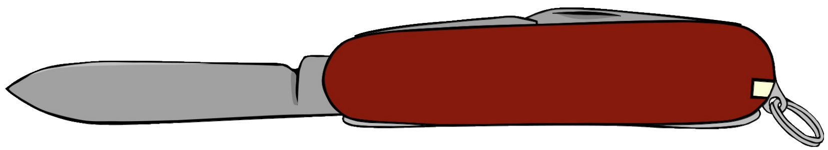
# How to Surprise by being a Linux Performance "know-it-all"

Christian Ehrhardt, IBM R&D Germany,
System Performance Analyst



# Agenda

- Tools are your swiss army knife
  - ps
  - top
  - sadc/sar
  - iostat
  - vmstat
  - netstat

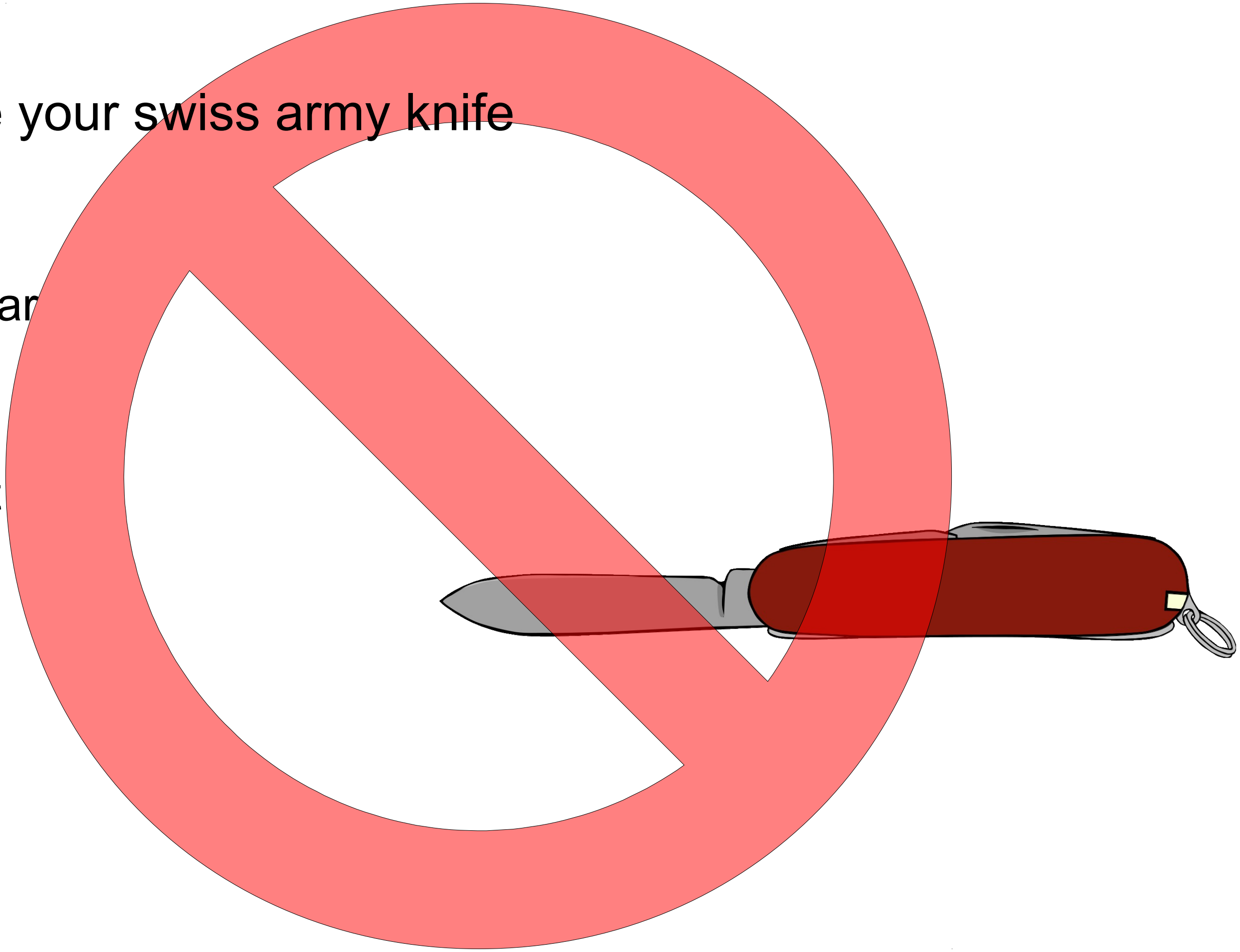




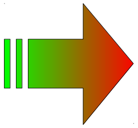
 

# Agenda

- Tools are your swiss army knife
  - ps
  - top
  - sadc/sar
  - iostat
  - vmstat
  - netstat



# Everything was nice and easy up to now, → be Ready for Take-Off



# Agenda

- Your swiss army knife for the complex cases
  - **Pidstat** – per process statistics
  - **Slabtop** – kernel memory pool consumption
  - **Lsof** – check file flags of open files
  - **Blktrace** – low level disk I/O analysis
  - **Hyptop** – cross guest cpu consumption monitor
  - **Iptraf** - network traffic monitor
  - **Dstat** – very configurable live system overview
  - **Irqstats** – check irq amount and cpu distribution
  - **Smem** – per process/per mapping memory overview
  - **Jinsight** – Java method call stack analysis
  - **Htop** – top on steroids
  - **Strace** – system call statistics
  - **Ltrace** – library call statistics
  - **Kernel tracepoints** – get in-depth timing inside the kernel
  - **Vmstat** – virtual memory statistics
  - **Sysstat** – full system overview
  - **Iostat** – I/O related statistics
  - **Dasdstat** – disk statistics
  - **scsi statistics** – disk statistics
  - **Perf** – hw counters, tracepoint based evaluations, profiling to find hotspots
  - **Valgrind** – in depth memory/cache analysis and leak detection
  - **Java Health Center** – high level java overview and monitoring
  - **Java Garbage Collection and Memory visualizer** – in depth gc analysis
  - **Netstat** – network statistics and overview
  - **Socket Statistics** – extended socket statistics
  - **top / ps** – process overview
  - **Icastats / lszcrypt** – check usage of crypto hw support
  - **Lsluns / multipath** – check multipath setup
  - **Lsqeth** – check hw checksumming and buffer count
  - **Ethtool** – check offloading functions
  - **Collectl** – full system monitoring
  - **Ftrace** – kernel function tracing
  - **Lttng** – complex latency tracing infrastructure ( no s390 support yet)
  - **Ziomon** – Analyze FCP setup and I/O
  - **Systemtap** – another kernel tracing infrastructure
  - **Wireshark / Tcpdump** – analyze network traffic in depth

# Agenda

- Your (little) swiss army knife for the complex cases
  - Pidstat
  - Slabtop
  - Smem
  - Valgrind
  - Lsof
  - Blktrace
  - Hyptop
  - Iptraf

60 min ?otto:
8 of 0x28

# Non-legal Disclaimer

- This is an introduction and cheat sheet
  - Know what is out there
  - What could be useful in which case
  - How could I debug even further
- These descriptions are not full explanations
  - Most tools could get at least 1-2 presentations on their own
  - Don't start using them without reading howtos / man pages
- This is not about monitoring
  - Some tools used to start performance analysis CAN be monitors, but thats not part of the presentation

# General thoughts on performance tools

- Things that are always to consider
  - Monitoring can impact the system
  - Most data gathering averages over a certain period of time → this flattens peaks
  - Start with defining the problem
    - which parameter(s) from the application/system indicates the problem
    - which range is considered as bad, what is considered as good
  - monitor the good case and save the results
    - comparisons when a problem occurs can save days and weeks

- Staged approach saves a lot of work
  - Try to use general tools to isolate the area of the issue
  - Create theories and try to quickly verify/falsify them
  - Use advanced tools to debug the identified area

# Orientation - where to go

| Tool | 1st overview | CPU cons. | latencies | Hot spots | Disk I/O | Memory | Network |
|---|---|---|---|---|---|---|---|
| top / ps | x | x | | | | | |
| sysstat | x | x | | | x | x | |
| vmstat | x | x | | | | x | |
| iostat | x | | | | x | | |
| dasdstat | | | | | x | | |
| scsistat | | | | | x | | |
| netstat / ss | x | | | | | | x |
| htop / dstat / pidstat | x | x | x | | x | | |
| irqstats | x | x | x | | | | |
| strace / ltrace | | | x | | | | |
| hyptop | | x | | | | | |
| perf | | x | x | x | x | x | x |
| jinsight | | x | x | | | | |
| Health Center | x | | | | | | |
| GMVC | | | x | | | x | |
| blktrace | | | | | x | | |
| lsof | | | | | x | | |
| valgrind | | | | | | x | |
| smem | | | | | | x | |
| slabtop | | | | | | x | |
| iptraf | x | | | | | | x |
| tracepoints | | | x | x | x | x | x |

# PIDSTAT

- Characteristics: Easy to use extended per process statistics
- Objective: Identify processes with peak activity
- Usage: `pidstat [-w|-r|-d]`
- Package: RHEL: sysstat SLES: sysstat

- Shows
  - `-w` context switching activity and if it was voluntary
  - `-r` memory statistics, especially minor/major faults per process
  - `-d` disk throughput per process

- Hints
  - Also useful if run as background log due to its low overhead
    - Good extension to sadc in systems running different applications/services
  - `-p <pid>` can be useful to track activity of a specific process

# Pidstat examples

```
12:46:18 PM       PID   cswch/s nvcswch/s  Command
12:46:18 PM         3      2.39      0.00  smbd
12:46:18 PM         4      0.04      0.00  sshd
12:46:18 PM      1073    123.42    180.18  Xorg
```

**Voluntarily / Involuntary**

```
12:47:51 PM       PID  minflt/s  majflt/s     VSZ    RSS   %MEM  Command
12:47:51 PM       985      0.06      0.00   15328   3948   0.10  smbd
12:47:51 PM       992      0.04      0.00    5592   2152   0.05  sshd
12:47:51 PM      1073    526.41      0.00 1044240 321512   7.89  Xorg
```

**Faults per process**

```
12:49:18 PM       PID   kB_rd/s   kB_wr/s kB_ccwr/s  Command
12:49:18 PM       330      0.00      1.15      0.00  sshd
12:49:18 PM      2899      4.35      0.09      0.04  notes2
12:49:18 PM      3045     23.43      0.01      0.00  audacious2
```

**How much KB disk I/O per process**

# Slabtop

- Characteristics: live profiling of kernel memory pools
- Objective: Analyze kernel memory consumption
- Usage: `slabtop`
- Package: RHEL: procps SLES: procps

- Shows
  - Active / Total object number/size
  - Objects per Slab
  - Object Name and Size
  - Objects per Slab

- Hints
  - -o is one time output e.g. to gather debug data
  - Despite slab/slob/slub in kernel its always slabtop

# Slabtop - example

```
 Active / Total Objects (% used)    : 2436408 / 2522983 (96.6%)
 Active / Total Slabs (% used)      : 57999 / 57999 (100.0%)
 Active / Total Caches (% used)     : 75 / 93 (80.6%)
 Active / Total Size (% used)       : 793128.19K / 806103.80K (98.4%)
 Minimum / Average / Maximum Object : 0.01K / 0.32K / 8.00K

  OBJS ACTIVE  USE OBJ SIZE  SLABS OBJ/SLAB CACHE SIZE NAME
578172 578172 100%    0.19K  13766       42    110128K dentry
458316 458316 100%    0.11K  12731       36     50924K sysfs_dir_cache
368784 368784 100%    0.61K   7092       52    226944K proc_inode_cache
113685 113685 100%    0.10K   2915       39     11660K buffer_head
113448 113448 100%    0.55K   1956       58     62592K inode_cache
111872  44251  39%    0.06K   1748       64      6992K kmalloc-64
 54688  50382  92%    0.25K   1709       32     13672K kmalloc-256
 40272  40239  99%    4.00K   5034        8    161088K kmalloc-4096
 39882  39882 100%    0.04K    391      102      1564K ksm_stable_node
 38505  36966  96%    0.62K    755       51     24160K shmem_inode_cache
 37674  37674 100%    0.41K    966       39     15456K dm_rq_target_io
```

- How is kernel memory managed by the sl[auo]b allocator used
  - Named memory pools or Generic kmalloc pools
  - Active/total objects and their size
  - growth/shrinks of caches due to workload adaption

# smem

- Characteristics: Memory usage details per process/mapping
- Objective: Where is userspace memory really used
- Usage: smem -tk -c "pid user command swap vss uss pss rss”
- smem -m -tk -c "map count pids swap vss uss rss pss avgrss avgpss"

- Package: RHEL: n/a SLES: n/a WWW http://www.selenic.com/smem/
- Shows
  - Pid, user, Command or Mapping, Count, Pid
  - Memory usage in categories vss, uss, rss, pss and swap

- Hints
  - Has visual output (pie charts) and filtering options as well
  - No support for huge pages or transparent huge pages (kernel interface missing)

# smem – process overview

```
smem -tk -c "pid user command swap vss uss pss rss"

  PID User     Command                         Swap      VSS      USS      PSS      RSS
 1860 root     /sbin/agetty -s sclp_line0         0     2.1M    92.0K   143.0K   656.0K
 1861 root     /sbin/agetty -s ttysclp0 11        0     2.1M    92.0K   143.0K   656.0K
  493 root     /usr/sbin/atd -f                   0     2.5M   172.0K   235.0K   912.0K
 1882 root     /sbin/udevd                        0     2.8M   128.0K   267.0K   764.0K
 1843 root     /usr/sbin/crond -n                 0     3.4M   628.0K   693.0K     1.4M
  514 root     /bin/dbus-daemon --system -        0     3.2M   700.0K   771.0K     1.5M
  524 root     /sbin/rsyslogd -n -c 5             0   219.7M   992.0K     1.1M     1.9M
 2171 root     ./hhhptest                         0     5.7G     1.0M     1.2M     3.2M
 1906 root     -bash                              0   103.8M     1.4M     1.5M     2.1M
 2196 root     ./hhhptest                         0     6.2G     2.0M     2.2M     3.9M
 1884 root     sshd: root@pts/0                   0    13.4M     1.4M     2.4M     4.2M
    1 root     /sbin/init                         0     5.8M     2.9M     3.0M     3.9M
 2203 root     /usr/bin/python /usr/bin/sm        0   109.5M     6.1M     6.2M     6.9M
```

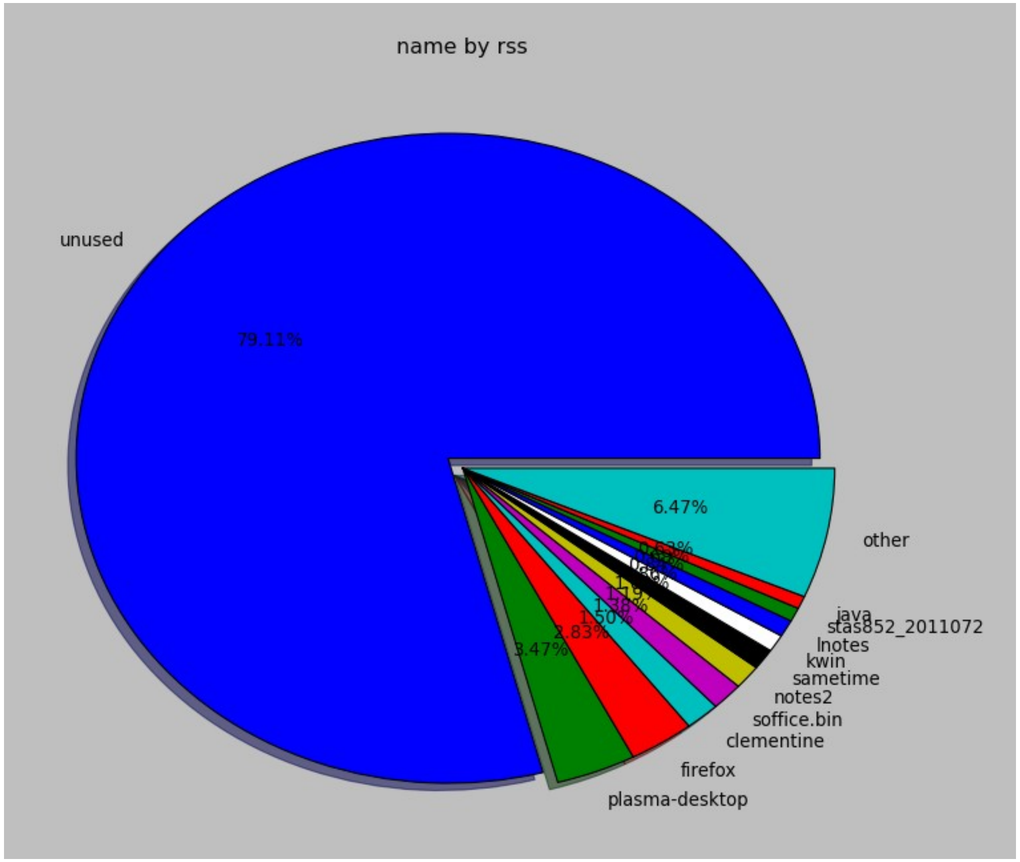
- How much of a process is:
  - Swap - Swapped out
  - VSS - Virtually allocated
  - USS - Really unique
  - RSS - Resident
  - PSS - Resident accounting a proportional part of shared memory

# smem – mappings overview

```
smem -m -tk -c "map count pids swap vss uss rss pss avgrss avgpss"

Map                                   Count  PIDs    Swap     VSS      USS      RSS      PSS   AVGRSS   AVGPSS
[stack:531]                               1     1       0    8.0M        0        0        0        0        0
[vdso]                                   25    25       0  200.0K        0   132.0K        0     5.0K        0
/dev/zero                                 2     1       0    2.5M     4.0K     4.0K     4.0K     4.0K     4.0K
/usr/lib64/sasl2/libsasldb.so.2.0.23      2     1       0   28.0K     4.0K     4.0K     4.0K     4.0K     4.0K
/bin/dbus-daemon                          3     1       0  404.0K   324.0K   324.0K   324.0K   324.0K   324.0K
/usr/sbin/sshd                            6     2       0    1.2M   248.0K   728.0K   488.0K   364.0K   244.0K
/bin/systemd                              2     1       0  768.0K   564.0K   564.0K   564.0K   564.0K   564.0K
/bin/bash                                 2     1       0    1.0M   792.0K   792.0K   792.0K   792.0K   792.0K
[stack]                                  25    25       0    4.1M   908.0K   976.0K   918.0K    39.0K    36.0K
/lib64/libc-2.14.1.so                    75    25       0   40.8M   440.0K     9.3M     1.2M   382.0K    48.0K
/lib64/libcrypto.so.1.0.0j                8     4       0    7.0M   572.0K     2.0M     1.3M   501.0K   321.0K
[heap]                                   16    16       0    8.3M     6.4M     6.9M     6.6M   444.0K   422.0K
<anonymous>                             241    25       0   55.7G    20.6M    36.2M    22.3M     1.4M   913.0K
```

- How much of a mapping is:
  - Swap - Swapped out
  - VSS - Virtually allocated
  - USS - Really unique
  - RSS - Resident
  - PSS - Resident accounting a proportional part of shared memory
  - Averages as there can be multiple mappers

# smem - visualizations




- Example of a memory distribution Visualization (many options)
- But before thinking of monitoring be aware that the proc/#pid/smaps interface is an expensive one

# Valgrind

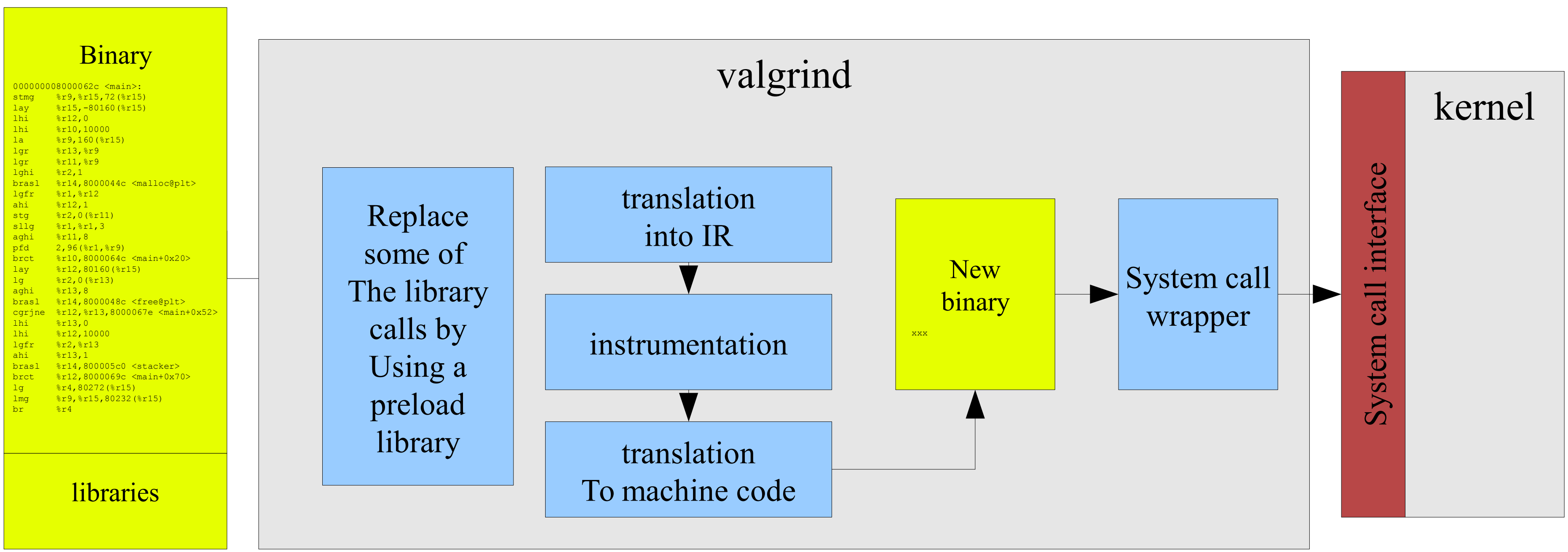
- Characteristics: in-depth memory analysis
- Objective: Find out where memory is leaked, sub-optimally cached, ...
- Usage: `valgrind [program]`
- Package: RHEL: valgrind SLES: valgrind

- Shows
  - Memory leaks
  - Cache profiling
  - Heap profiling

- Hints
  - Runs on binaries, therefore easy to use
  - Debug Info not required but makes output more useful

# Valgrind Overview

- Technology is based on a JIT (Just-in-Time Compiler)
- Intermediate language allows debugging instrumentation



# Valgrind – sample output of “memcheck”

```
# valgrind buggy_program
==2799== Memcheck, a memory error detector
==2799== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
==2799== Using Valgrind-3.6.1 and LibVEX; rerun with -h for copyright info
==2799== Command: buggy_program
==2799==
==2799== HEAP SUMMARY:
==2799==     in use at exit: 200 bytes in 2 blocks
==2799==   total heap usage: 2 allocs, 0 frees, 200 bytes allocated
==2799==
==2799== LEAK SUMMARY:
==2799==    definitely lost: 100 bytes in 1 blocks
==2799==    indirectly lost: 0 bytes in 0 blocks
==2799==      possibly lost: 0 bytes in 0 blocks
==2799==    still reachable: 100 bytes in 1 blocks
==2799==         suppressed: 0 bytes in 0 blocks
==2799== Rerun with --leak-check=full to see details of leaked memory
[...]
```

- Important parameters:
  - --leak-check=full
  - --track-origins=yes

# Valgrind - Tools

- Several tools
  - Memcheck (default): detects memory and data flow problems
  - Cachegrind: cache profiling
  - Massif: heap profiling
  - Helgrind: thread debugging
  - DRD: thread debugging
  - None: no debugging (for valgrind JIT testing)
  - Callgrind: codeflow and profiling

- Tool can be selected with –tool=xxx
- System z support since version 3.7 (SLES-11-SP2)
- Backports into 3.6 (SLES-10-SP4, RHEL6-U1)

# Valgrind - Good to know

- No need to recompile, but
  - Better results with debug info
  - Gcc option -O0 might result in more findings(the compiler might hide some errors)
  - Gcc option -fno-builtin might result in more findings

- --trace-children=yes will also debug child processes
- Setuid programs might cause trouble
  - Valgrind is the process container (→ no setuid)
  - Possible solution: remove setuid and start as the right user, check documentation for other ways

- The program will be slower
  - 5-30 times slower for memcheck

# lsof

- Characteristics: list of open files plus extra details
- Objective: which process accesses which file in which mode
- Usage: `lsof +fg`
- Package: RHEL: lsof SLES: lsof

- Shows
  - List of files including sockets, directories, pipes
  - User, Command, Pid, Size, Device
  - File Type and File Flags

- Hints
  - +fg reports file flags which can provide a good cross check opportunity

# lsof - example

```
COMMAND     PID   TID    USER   FD      TYPE          FILE-FLAG             DEVICE  SIZE/OFF      NODE NAME
crond     16129          root  mem       REG                                  94,1    165000    881893
/usr/lib64/ld-2.16.so
crond     16129          root    0r      CHR                 LG                1,3       0t0      2051 /dev/null
crond     16129          root    1u     unix                 RW 0x0000001f1ba02000       0t0    106645 socket
crond     16129          root    2u     unix                 RW 0x0000001f1ba02000       0t0    106645 socket
crond     16129          root    4r  a_inode            0x80000                0,9         0      6675 inotify
crond     16129          root    5u     unix         RW,0x80000 0x0000001f5d3ad000       0t0     68545 socket
dd        17617          root  cwd       DIR                                  94,1      4096     16321 /root
dd        17617          root  rtd       DIR                                  94,1      4096         2 /
dd        17617          root  txt       REG                                  94,1     70568   1053994 /usr/bin/dd
dd        17617          root  mem       REG                                  94,1    165000    881893
/usr/lib64/ld-2.16.so
dd        17617          root    0r      CHR                 LG                1,9       0t0      2055 /dev/urandom
dd        17617          root    1w      REG           W,DIR,LG               94,1   5103616     16423 /root/test
dd        17617          root    2u      CHR              RW,LG              136,2       0t0         5 /dev/pts/2
```

- You can filter that per application or per file
  - Fd holds fdnumber, type, characteristic and lock information
    - File descriptors can help to read strace/ltrace output
  - Flags can be good to confirm e.g. direct IO, async IO
  - Size (e.g. mem) or offset (fds), name, ...

# BLKTRACE

- Characteristics: High detail info of the block device layer actions
- Objective: Understand whats going with your I/O in the kernel and devices
- Usage: blktrace -d [device(s)]
- Then: blkparse -st [commontracefilepart]
- Package: RHEL: blktrace SLES: blktrace
- Shows
  - Events like merging, request creation, I/O submission, I/O completion, ...
  - Timestamps and disk offsets for each event
  - Associated task and executing CPU
  - Application and CPU summaries

- Hints
  - Filter masks allow lower overhead if only specific events are of interest
  - Has an integrated client/server mode to stream data away
    - Avoids extra disk I/O on a system with disk I/O issues

# Blktrace – when is it useful

- Often its easy to identify that I/O is slow, but
  → Where?
  → Because of what?

- Blocktrace allows to
  - Analyze Disk I/O characteristics like sizes and offsets
    - Maybe your I/O is split in a layer below
  - Analyze the timing with details about all involved Linux layers
    - Often useful to decide if HW or SW causes stalls
  - Summaries per CPU / application can identify imbalances

# Blktrace - events

Common:
A -- remap For stacked devices, incoming i/o is remapped to device below it in the i/o stack. The remap action details what exactly is being remapped to what.
Q -- queued This notes intent to queue i/o at the given location.  No real requests exists yet.
G -- get request To send any type of request to a block device, a struct request container must be allocated first.
I -- inserted A request is being sent to the i/o scheduler for addition to the internal queue and later service by the driver. The request is fully formed at this time.
D -- issued A request that previously resided on the block layer queue or in the i/o scheduler has been sent to the driver.
C -- complete A previously issued request has been completed.  The output will detail the sector and size of that request, as well as the success or failure of it.

Plugging & Merges:
P -- plug When i/o is queued to a previously empty block device queue, Linux will plug the queue in anticipation of future I/Os being added before this data is needed.
U -- unplug Some request data already queued in the device, start sending requests to the driver. This may happen automatically if a timeout period has passed (see next entry) or if a number of requests have been added to the queue.
Recent kernels associate the queue with the submitting task and unplug also on a context switch.
T -- unplug due to timer If nobody requests the i/o that was queued after plugging the queue, Linux will automatically unplug it after a defined period has passed.
M -- back merge A previously inserted request exists that ends on the boundary of where this i/o begins, so the i/o scheduler can merge them together.
F -- front merge Same as the back merge, except this i/o ends where a previously inserted requests starts.

Special:
B -- bounced The data pages attached to this bio are not reachable by the hardware and must be bounced to a lower memory location. This causes a big slowdown in i/o performance, since the data must be copied to/from kernel buffers. Usually this can be fixed with using better hardware -- either a better i/o controller, or a platform with an IOMMU.
S -- sleep No available request structures were available, so the issuer has to wait for one to be freed.
X  --  split  On raid or device mapper setups, an incoming i/o may straddle a device or internal zone and needs to be chopped up into smaller pieces for service. This may indicate a performance problem due to a bad setup of that raid/dm device, but may also just be part of normal boundary conditions. dm is notably bad at this and will clone lots of i/o.

# Blktrace - events

Common:
A -- remap For stacked devices, incoming i/o is remapped to device below it in the i/o stack. The remap action details what exactly is being remapped to what.
Q -- queued This notes intent to queue i/o at the given location. No real requests exists yet.
G -- get request To send any type of request to a block device, a struct request container must be allocated first.
I -- inserted A request is being sent to the i/o scheduler for addition to the internal queue and later service by the driver. The request is fully formed at this time.
D -- issued A request that previously resided on the block layer queue or in the i/o scheduler has been sent to the driver.
C -- complete A previously issued request has been completed. The output will detail the sector and size of that request, as well as the success or failure of it.

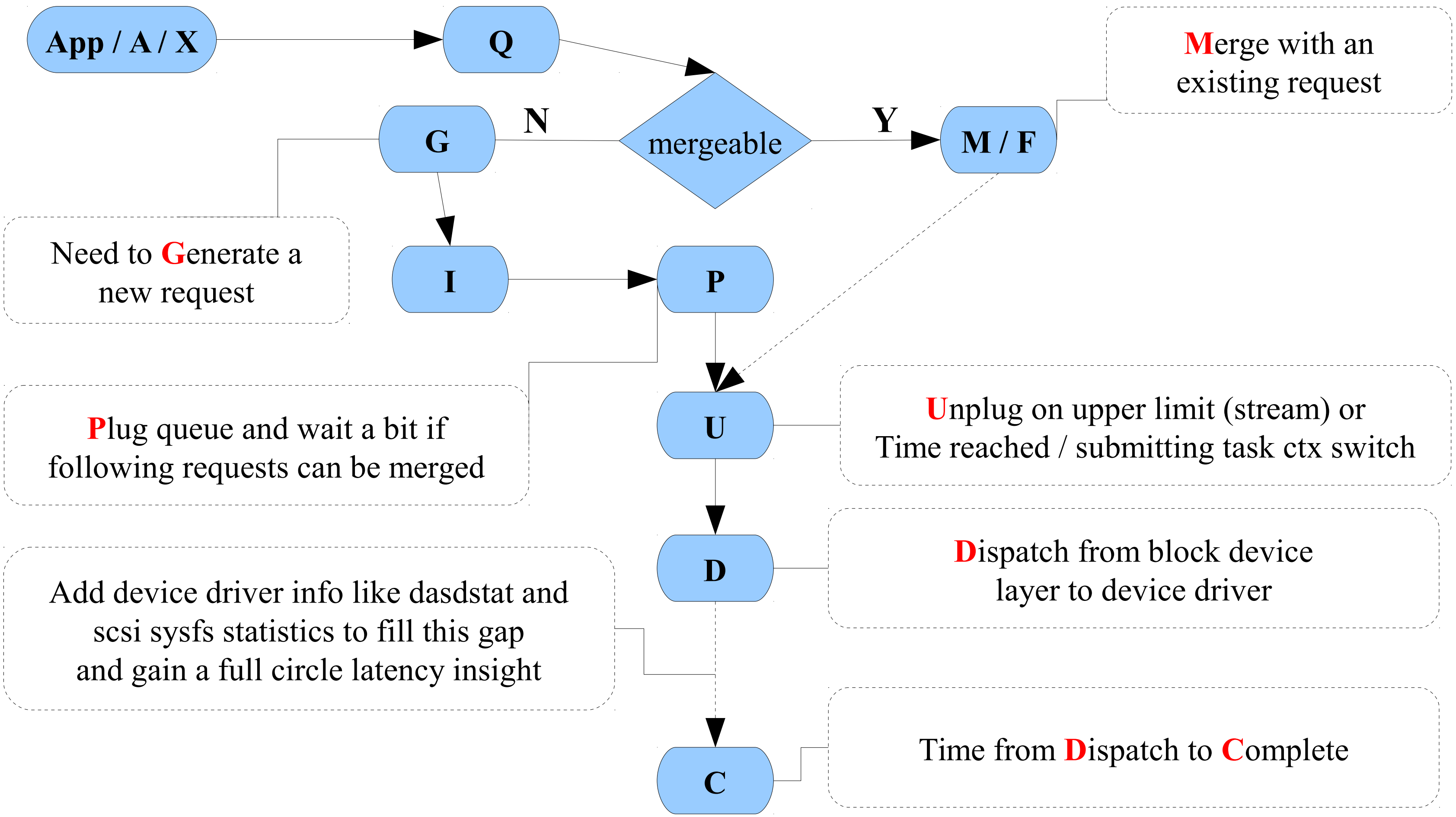
Plugging & Merges:
P -- plug When i/o is queued to a pre … queue in anticipation of future I/Os being added before this data is needed.
U -- unplug Some request data alrea … driver. This may happen automatically if a timeout period has passed (see next entry) or if a numb …
Recent kernels associate the queue … switch.
T -- unplug due to timer If nobody requests the i/o that was queued after plugging the queue, Linux will automatically unplug it after a defined period has passed.
M -- back merge A previously inserted request exists that ends on the boundary of where this i/o begins, so the i/o scheduler can merge them together.
F -- front merge Same as the back merge, except this i/o ends where a previously inserted requests starts.

Special:
B -- bounced The data pages attached to this bio are not reachable by the hardware and must be bounced to a lower memory location. This causes a big slowdown in i/o performance, since the data must be copied to/from kernel buffers. Usually this can be fixed with using better hardware -- either a better i/o controller, or a platform with an IOMMU.
S -- sleep No available request structures were available, so the issuer has to wait for one to be freed.
X -- split On raid or device mapper setups, an incoming i/o may straddle a device or internal zone and needs to be chopped up into smaller pieces for service. This may indicate a performance problem due to a bad setup of that raid/dm device, but may also just be part of normal boundary conditions. dm is notably bad at this and will clone lots of i/o.

Good as documentation, but hard to understand/remember

# Block device layer – events (simplified)



# blktrace

- Example Case
  - The snippet shows a lot of 4k requests (8x512 byte sectors)
    - We expected the I/O to be 32k
  - Each one is dispatched separately (no merges)
    - This caused unnecessary overhead and slow I/O

```
Maj/Min CPU    Seq-nr        sec.nsec   pid Action RWBS sect + size  map source / task
94,4      27       21     0.059363692 18994      A     R 20472832 + 8 <- (94,5) 20472640
94,4      27       22     0.059364630 18994      Q     R 20472832 + 8 [qemu-kvm]
94,4      27       23     0.059365286 18994      G     R 20472832 + 8 [qemu-kvm]
94,4      27       24     0.059365598 18994      I     R 20472832 + 8 (     312) [qemu-kvm]
94,4      27       25     0.059366255 18994      D     R 20472832 + 8 (     657) [qemu-kvm]
94,4      27       26     0.059370223 18994      A     R 20472840 + 8 <- (94,5) 20472648
94,4      27       27     0.059370442 18994      Q     R 20472840 + 8 [qemu-kvm]
94,4      27       28     0.059370880 18994      G     R 20472840 + 8 [qemu-kvm]
94,4      27       29     0.059371067 18994      I     R 20472840 + 8 (     187) [qemu-kvm]
94,4      27       30     0.059371473 18994      D     R 20472840 + 8 (     406) [qemu-kvm]
```

# blktrace

- Example Case
  - Analysis turned out that the I/O was from the swap code
    - Same offsets were written by kswapd
  - A recent code change there disabled the ability to merge I/O
  - The summary below shows the difference after a fix

```
Total initially
 Reads Queued:      560,888,     2,243MiB  Writes Queued:     226,242,   904,968KiB
 Read Dispatches:   544,701,     2,243MiB  Write Dispatches:  159,318,   904,968KiB
 Reads Requeued:          0                Writes Requeued:         0
 Reads Completed:   544,716,     2,243MiB  Writes Completed:  159,321,   904,980KiB
 Read Merges:        16,187,    64,748KiB  Write Merges:       61,744,   246,976KiB
 IO unplugs:        149,614                Timer unplugs:       2,940

Total after Fix
 Reads Queued:      734,315,     2,937MiB  Writes Queued:     300,188,     1,200MiB
 Read Dispatches:   214,972,     2,937MiB  Write Dispatches:  215,176,     1,200MiB
 Reads Requeued:          0                Writes Requeued:         0
 Reads Completed:   214,971,     2,937MiB  Writes Completed:  215,177,     1,200MiB
 Read Merges:       519,343,     2,077MiB  Write Merges:       73,325,   293,300KiB
 IO unplugs:        337,130                Timer unplugs:      11,184
```

# Hyptop

- Characteristics: Easy to use Guest/LPAR overview
- Objective: Check CPU and overhead statistics of your and sibling images
- Usage: `hyptop`
- Package: RHEL: s390utils-base SLES: s390-tools

- Shows
  - CPU load & Management overhead
  - Memory usage (only under zVM)
  - Can show image overview or single image details

- Hints
  - Good “first view” tool for linux admins that want to look “out of their linux”
  - Requirements:
    - For z/VM the Guest needs Class B
    - For LPAR “Global performance data control” checkbox in HMC

# Hyptop

memuse = resident

Why are exactly 4 CPUs used in all 6 CPU guests

All these do not fully utilize their 2 CPUs

No peaks in service guests

LPAR images would see other LPARs

service guest weights

11:12:56 CPU-T: UN(64)

| system (str) | #cpu (#) | cpu (%) | Cpu+ (hm) | online (dhm) | memuse (GiB) | memmax (GiB) | wcur (#) |
|---|---|---|---|---|---|---|---|
| R3729003 | 6 | 399.11 | 2:24 | 0:03:05 | 11.94 | 12.00 | 100 |
| R3729004 | 6 | 399.07 | 2:24 | 0:03:05 | 11.94 | 12.00 | 100 |
| R3729001 | 6 | 398.99 | 2:26 | 0:03:09 | 11.95 | 12.00 | 100 |
| R3729005 | 6 | 398.76 | 2:24 | 0:03:05 | 11.94 | 12.00 | 100 |
| R3729009 | 4 | 398.62 | 2:22 | 0:03:05 | 4.20 | 6.00 | 100 |
| R3729008 | 4 | 398.49 | 2:22 | 0:03:05 | 4.21 | 6.00 | 100 |
| R3729007 | 4 | 398.39 | 2:21 | 0:03:05 | 4.18 | 6.00 | 100 |
| R3729010 | 4 | 398.02 | 2:21 | 0:03:05 | 4.18 | 6.00 | 100 |
| R3729002 | 6 | 397.99 | 2:24 | 0:03:05 | 11.94 | 12.00 | 100 |
| R3729006 | 4 | 393.09 | 2:21 | 0:03:05 | 4.17 | 6.00 | 100 |
| R3729012 | 2 | 117.37 | 0:43 | 0:03:05 | 0.25 | 2.00 | 100 |
| R3729014 | 2 | 117.27 | 0:44 | 0:03:05 | 0.25 | 2.00 | 100 |
| R3729011 | 2 | 117.13 | 0:43 | 0:02:37 | 0.25 | 2.00 | 100 |
| R3729013 | 2 | 117.08 | 0:43 | 0:03:05 | 0.25 | 2.00 | 100 |
| R3729015 | 2 | 116.63 | 0:43 | 0:03:05 | 0.25 | 2.00 | 100 |
| VMSERVU | 1 | 0.00 | 0:00 | 0:03:10 | 0.01 | 0.03 | 1500 |
| VMSERVP | 1 | 0.00 | 0:00 | 0:03:10 | 0.01 | 0.06 | 1500 |
| VMSERVR | 1 | 0.00 | 0:00 | 0:03:10 | 0.01 | 0.03 | 1500 |
| RACFVM | 1 | 0.00 | 0:00 | 0:03:10 | 0.01 | 0.02 | 100 |
| OPERSYMP | 1 | 0.00 | 0:00 | 0:03:10 | 0.00 | 0.03 | 100 |
| TCPIP | 1 | 0.00 | 0:00 | 0:03:10 | 0.01 | 0.12 | 3000 |
| DTCVSW2 | 1 | 0.00 | 0:00 | 0:03:10 | 0.01 | 0.03 | 100 |
| OPERATOR | 1 | 0.00 | 0:00 | 0:03:10 | 0.00 | 0.03 | 100 |
| RSCS | 1 | 0.00 | 0:00 | 0:03:09 | 0.00 | 0.03 | 100 |
| RSCSDNS | 1 | 0.00 | 0:00 | 0:03:10 | 0.00 | 0.03 | 100 |
| AUTOVM | 1 | 0.00 | 0:00 | 0:03:10 | 0.00 | 0.03 | 100 |
| GCS | 1 | 0.00 | 0:00 | 0:03:10 | 0.00 | 0.02 | 100 |
| LGLOPR | 1 | 0.00 | 0:00 | 0:03:10 | 0.00 | 0.03 | 100 |
| DIRMAINT | 1 | 0.00 | 0:00 | 0:03:10 | 0.01 | 0.03 | 100 |

# IPTRAF

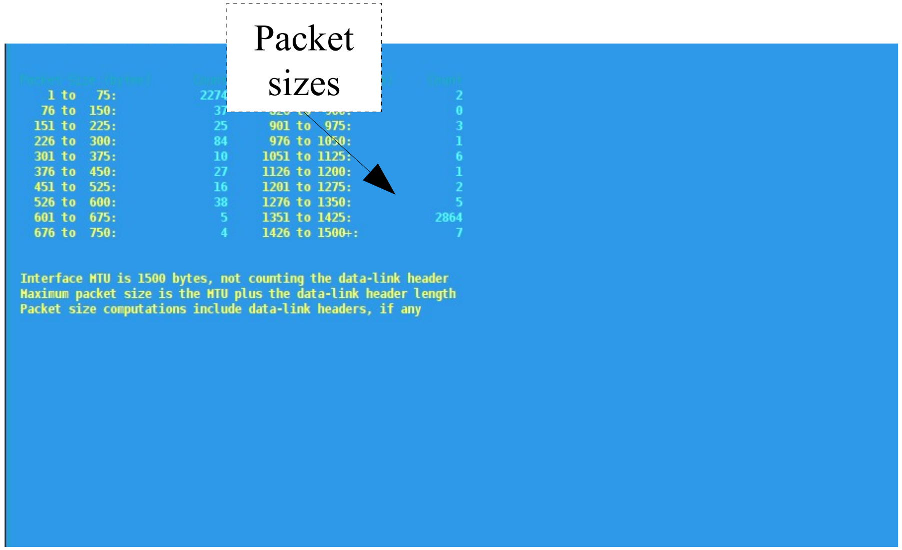
- Characteristics: Live information on network devices / connections
- Objective: Filter and format network statistics
- Usage: `iptraf`
- Package: RHEL: iptraf SLES: iptraf

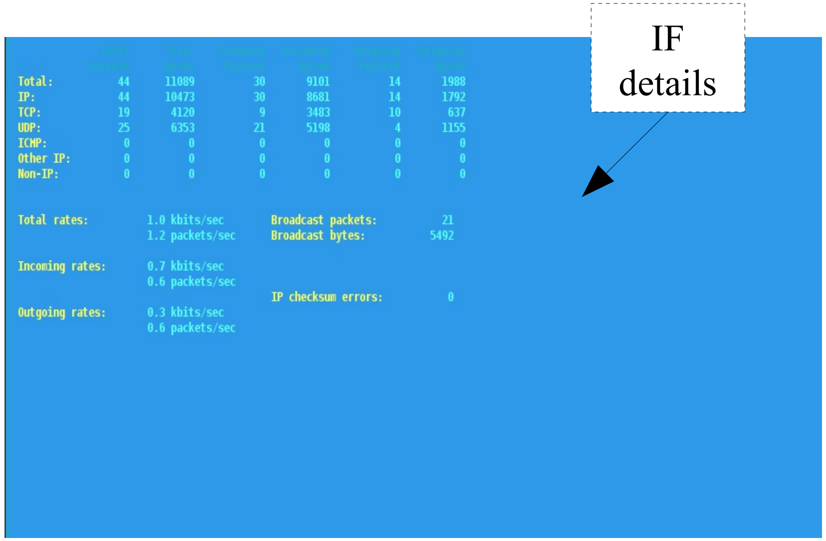
- Shows
  - Details per Connection / Interface
  - Statistical breakdown of ports / packet sizes
  - LAN station monitor

- Hints
  - Can be used for background logging as well
    - Use SIGUSR1 and logrotate to handle the growing amount of data
  - Knowledge of packet sizes important for the right tuning

# iptraf

- Questions that usually can be addressed
  - Connection behavior overview
  - Do you have peaks in your workload characteristic
  - Who does your host really communicate with
- Comparison to wireshark
  - Not as powerful, but much easier and faster to use
  - Lower overhead and no sniffing needed (often prohibited)



# Cachestat

- Characteristics: Simple per page views of caching
- Objective: Detect what parts of a file are in page cache
- Usage: Write – or search for example code
- Package: n/a (pure code around the mincore system call)

- Shows
  - How much of a file is in cache

- Hints
  - We are now going from unsupported to non existent packages
  - Still the insight can be so useful, it is good to know

# Cachestat usage

```
./cachestat -v ../Music/mysong.flac
pages in cache: 445/12626 (3.5%) [filesize=50501.0K, pagesize=4K]

cache map:
     0: |x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|
    32: |x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|
[...]
   384: |x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|
   416: |x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x| | | | |
   448: | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
[...]
 12576: | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
 12608: | | | | | | | | | | | | | | | | | |x|
```

- Here I show how much of a file is in cache while playing a song
  - You'll see readahead here
  - You'll also see the last block is almost always read in this case

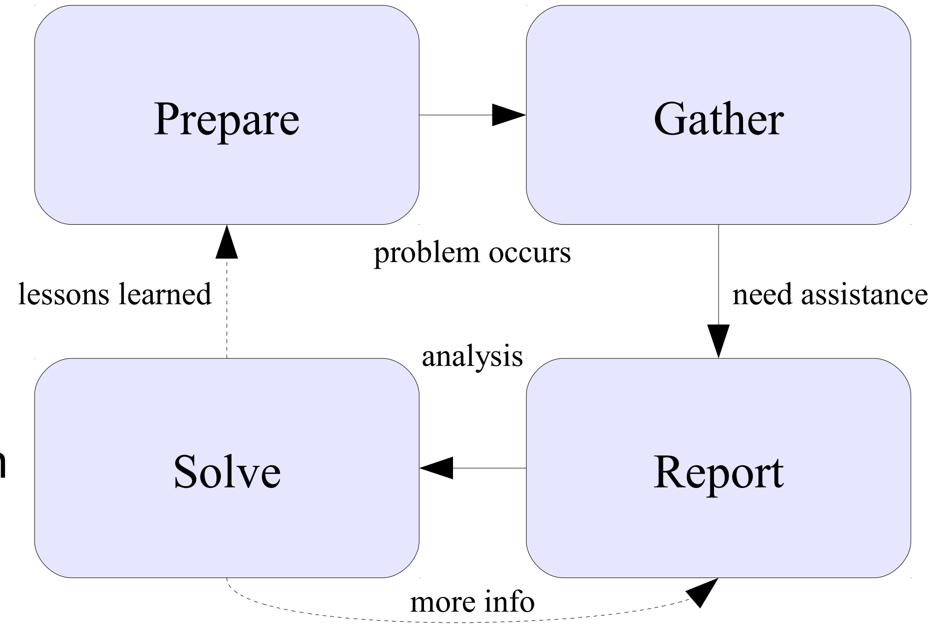
# Orientation - where to go

| Tool | 1st overview | CPU cons. | latencies | Hot spots | Disk I/O | Memory | Network |
|---|---|---|---|---|---|---|---|
| top / ps | x | x | | | | | |
| sysstat | x | x | | | x | x | |
| vmstat | x | x | | | | x | |
| iostat | x | | | | x | | |
| dasdstat | | | | | x | | |
| scsistat | | | | | x | | |
| netstat / ss | x | | | | | | x |
| htop / dstat / pidstat | x | x | x | | x | | |
| irqstats | x | x | x | | | | |
| strace / ltrace | | | x | | | | |
| hyptop | | x | | | | | |
| perf | | x | x | x | x | x | x |
| jinsight | | x | x | | | | |
| Health Center | x | | | | | | |
| GMVC | | | x | | | x | |
| blktrace | | | | | x | | |
| lsof | | | | | x | | |
| valgrind | | | | | | x | |
| smem | | | | | | x | |
| slabtop | | | | | | x | |
| iptraf | x | | | | | | x |
| tracepoints | | | x | x | x | x | x |

# Don't miss preparation

- Of all tools preparation is clearly
  - The most important
  - The most effective

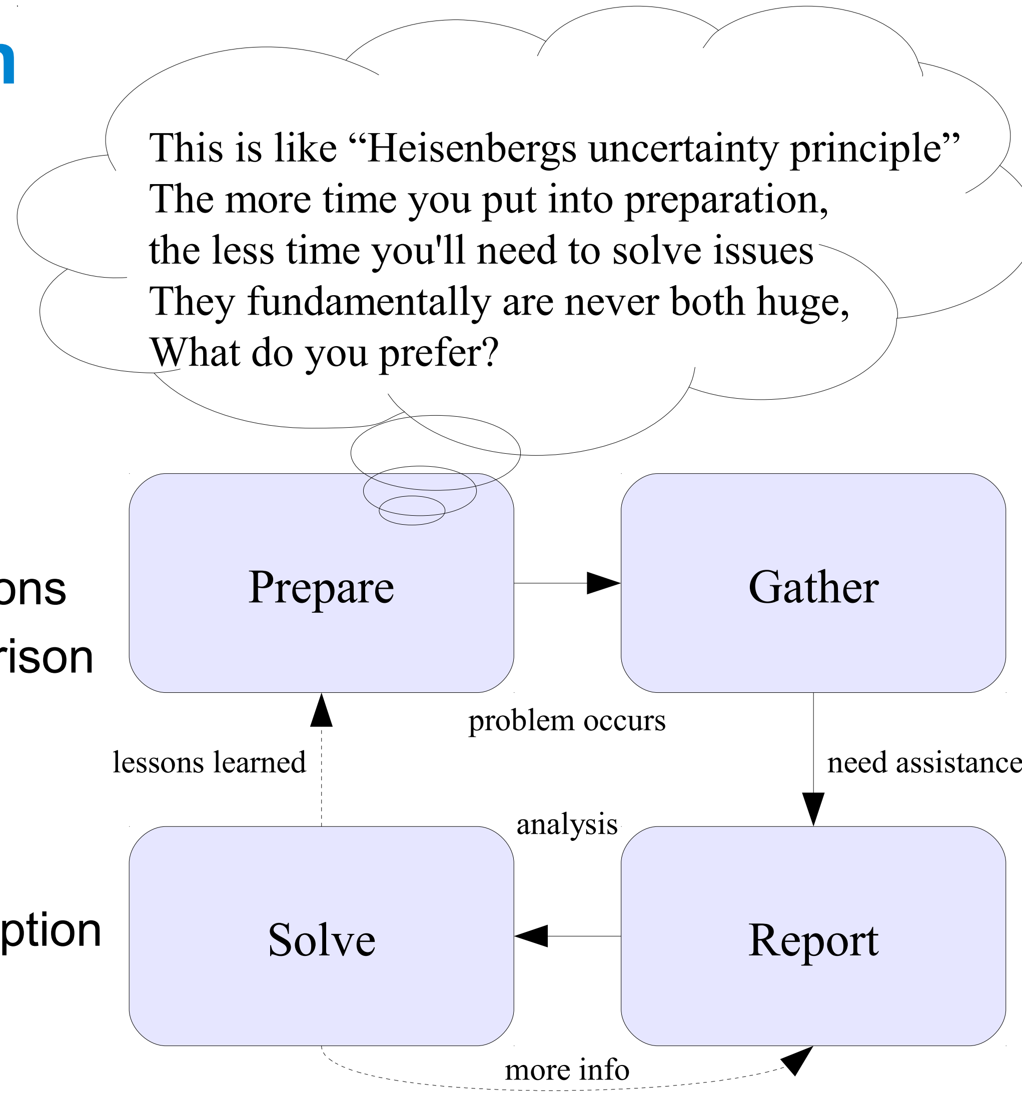
- Prepare
  - System and Workload descriptions
  - Healthy system data for comparison
- Gather
  - In case of emergency
- Report
  - How to report a Problem Description
- Solve
  - Tools to start an analysis



# Don't miss preparation

- Of all tools preparation is clearly
  - The most important
  - The most effective

- Prepare
  - System and Workload descriptions
  - Healthy system data for comparison
- Gather
  - In case of emergency
- Report
  - How to report a Problem Description
- Solve
  - Tools to start an analysis

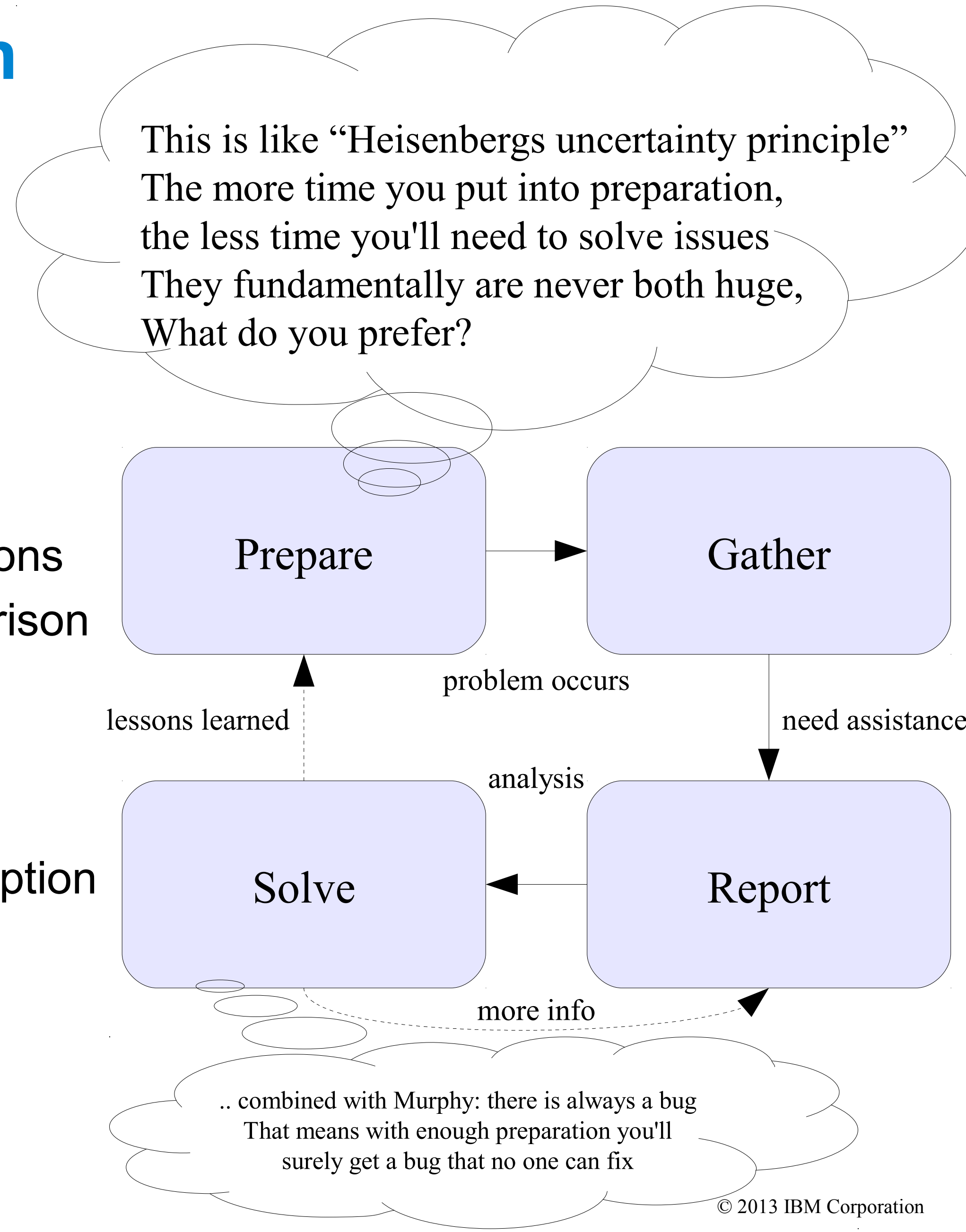


# Don't miss preparation

- Of all tools preparation is clearly
  - The most important
  - The most effective

- Prepare
  - System and Workload descriptions
  - Healthy system data for comparison
- Gather
  - In case of emergency
- Report
  - How to report a Problem Description
- Solve
  - Tools to start an analysis



# Ultimate Swiss Army knife

- The one you should always have → IBM System z Enterprise

# Appendix Preview covering even more tools

- Further complex tools
  - **Dstat** – very configurable live system overview
  - **Irqstats** – check irq amount and cpu distribution
  - **Java Health Center** – high level java overview and monitoring
  - **Java Garbage Collection and Memory visualizer** – in depth gc analysis
  - **Jinsight** – Java method call stack analysis
  - **Perf** – hw counters, tracepoint based evaluations, profiling to find hotspots
  - **Htop** – top on steroids
  - **Strace** – system call statistics
  - **Ltrace** – library call statistics
  - **Kernel tracepoints** – get in-depth timing inside the kernel
  - **Icastats / lszcrypt** – check usage of crypto hw support
  - **Lsluns / multipath** – check multipath setup
  - **Ethtool** – check offloading functions
  - **Ziomon** – analyze FCP setup and I/O
  - **Systemtap** – another kernel tracing infrastructure
  - **Wireshark / tcpdump** – analyze network traffic in depth

- Entry level Tools
  - **Vmstat** – virtual memory statistics
  - **Sysstat** – full system overview
  - **Iostat** – I/O related statistics
  - **Dasdstat** – disk statistics
  - **scsi statistics** – disk statistics
  - **Netstat** – network statistics and overview
  - **Socket Statistics** – extended socket statistics
  - **top / ps** – process overview
  - **Lsqeth** – check hw checksumming and buffer count

- Further tools - (no slides yet)
  - **Collectl** – full system monitoring
  - **Ftrace** – kernel function tracing
  - **Lttng** – complex latency tracing infrastructure ( no s390 support yet)

# 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*
*Performance Evaluation*

*Research & Development*
*Schönaicher Strasse 220*
*71032 Böblingen, Germany*

*ehrhardt@de.ibm.com*

# STRACE

- Characteristics: High overhead, high detail tool
- Objective: Get insights about the ongoing system calls of a program
- Usage: `strace -p [pid of target program]`
- Package: RHEL: strace SLES: strace

- Shows
  - Identify kernel entries called more often or taking too long
    - Can be useful if you search for increased system time
  - Time in call (`-T`)
  - Relative timestamp (`-r`)

- Hints
  - The option "`-c`" allows medium overhead by just tracking counters and durations

# strace - example

a lot, slow or failing calls?

shares to rate importance

name (see man pages)

```
strace -cf -p 26802
Process 26802 attached - interrupt to quit
^Process 26802 detached
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 58.43    0.007430          17       450           read
 24.33    0.003094           4       850       210 access
  5.53    0.000703           4       190        10 open
  4.16    0.000529           3       175           write
  2.97    0.000377           2       180           munmap
  1.95    0.000248           1       180           close
  1.01    0.000128           1       180           mmap
  0.69    0.000088          18         5           fdatasync
  0.61    0.000078           0       180           fstat
  0.13    0.000017           3         5           pause
------ ----------- ----------- --------- --------- ----------------
100.00    0.012715                  2415       225 total
```

# LTRACE

- Characteristics: High overhead, high detail tool
- Objective: Get insights about the ongoing library calls of a program
- Usage: `ltrace -p [pid of target program]`
- Package: RHEL: ltrace SLES: ltrace

- Shows
  - Identify library calls that are too often or take too long
    - Good if you search for additional user time
    - Good if things changed after upgrading libs
  - Time in call (`-T`)
  - Relative timestamp (`-r`)

- Hints
  - The option “`-c`” allows medium overhead by just tracking counters and durations
  - The option -S allows to combine ltrace and strace

# ltrace - example

shares to rate importance

a lot or slow calls?

name (see man pages)

```
ltrace -cf -p 26802
% time     seconds  usecs/call     calls      function
------ ----------- ----------- --------- --------------------
 98.33   46.765660     5845707         8 pause
  0.94    0.445621          10     42669 strncmp
  0.44    0.209839          25      8253 fgets
  0.08    0.037737          11      3168 __isoc99_sscanf
  0.07    0.031786          20      1530 access
  0.04    0.016757          10      1611 strchr
  0.03    0.016479          10      1530 snprintf
  0.02    0.010467        1163         9 fdatasync
  0.02    0.008899          27       324 fclose
  0.02    0.007218          21       342 fopen
  0.01    0.006239          19       315 write
  0.00    0.000565          10        54 strncpy
------ ----------- ----------- --------- --------------------
100.00   47.560161                 59948 total
```

# Strace / Ltrace – full trace

- Without -c both tools produce a full detail log
  - Via -f child processes can be traced as well
  - Extra options "`-Tr`" are useful to search for latencies follow time in call / relative timestamp
  - Useful to "read" what exactly goes on when

```
Example strace'ing a sadc data gatherer
0.000028 write(3, "\0\0\0\0\0\0\17\0\0\0\0\0\0\0"..., 680) = 680 <0.000007>
0.000027 write(3, "\0\0\0\0\0\0\17\0\0\0\0\0\0\0"..., 680) = 680 <0.000007>
0.000026 fdatasync(3)                    = 0 <0.002673>
0.002688 pause()                         = 0 <3.972935>
3.972957 --- SIGALRM (Alarm clock) @ 0 (0) ---
0.000051 rt_sigaction(SIGALRM, {0x8000314c, [ALRM], SA_RESTART}, 8) = 0 <0.000005>
0.000038 alarm(4)                        = 0 <0.000005>
0.000031 sigreturn()                     = ? (mask now []) <0.000005>
0.000024 stat("/etc/localtime", {st_mode=S_IFREG|0644, st_size=2309, ...}) = 0 <0.000007>
0.000034 open("/proc/uptime", O_RDONLY) = 4 <0.000009>
0.000024 fstat(4, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0 <0.000005>
0.000029 mmap(NULL, 4096, PROT_READ, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x3fffd20a000 <0.000006>
0.000028 read(4, "11687.70 24836.04\n", 1024) = 18 <0.000010>
0.000027 close(4)                        = 0 <0.000006>
0.000020 munmap(0x3fffd20a000, 4096) = 0 <0.000009>
```

# DSTAT

- Characteristics: Live easy to use full system information
- Objective: Flexible set of statistics
- Usage:  dstat -tv –aio –disk-util -n –net-packets -i –ipc
-             -D total,[diskname] –top-io [...] [interval]
- Short: `dstat -tinv`
- Package: RHEL: dstat SLES: n/a WWW: http://dag.wieers.com/home-made/dstat/
- Shows
  - Throughput
  - Utilization
  - Summarized and per Device queue information
  - Much more … it more or less combines several classic tools like iostat and vmstat
- Hints
  - Powerful plug-in concept
    - “`--top-io`” for example identifies the application causing the most I/Os
  - Colorization allows fast identification of deviations

# Dstat – the limit is your screen width

```
Report nice bugs to bug-coreutils@gnu.org
GNU coreutils home page: <http://www.gnu.org/software/coreutils/>
General help using GNU software: <http://www.gnu.org/gethelp/>
For complete documentation, run: info coreutils 'nice invocation'
[root@r3729001 ~]# nice -n 1 dstat -tv --aio --disk-util -n --net-packets -i --ipc -D total,sda --top-io --noupdate 5
----system---- ---procs--- ------memory-usage----- ---paging-- -dsk/total----dsk/sda-- ---system-- ----total-cpu-usage---- async so
     time     |run blk new| used  buff  cach  free|  in   out | read  writ: read  writ| int   csw |usr sys idl wai hiq siq| #aio|ut
17-07 17:41:18|0.0   0  38|1303M 13.5M 10.4G 57.4M|   0     0 |4137k   14M: 124k  337k|   0  4968 |  4   3  92   0   0   1|   0 |1.
17-07 17:41:24| 13   0   0|1307M 13.5M 10.4G 57.2M|   0     0 |1708k   30k:  45k    0 |   0   16k| 33   9  55   0   0   3|   0 |0.
17-07 17:41:28|9.4 0.2   0|1311M 13.5M 10.4G 59.0M|   0     0 |1626k   19k:  60k    0 |   0   15k| 63  15  17   0   0   5|   0 |0.
17-07 17:41:34| 13   0 0.2|1313M 13.5M 10.4G 59.5M|   0     0 |1325k   11k:  32k    0 |   0   11k| 71  17   6   0   0   6|   0 |
17-07 17:41:39|3.6   0   0|1317M 13.6M 10.4G 60.8M|   0     0 |1258k   23k:  26k    0 |   0   16k| 76  18   0   0   0   6|   0 |0.
17-07 17:41:44| 13   0   0|1318M 13.6M 10.4G 53.1M|   0     0 |1601k   16k:  37k    0 |   0   14k| 75  19   0   0   0   6|   0 |0.
17-07 17:41:49| 11   0 0.2|1322M 13.6M 10.4G 82.2M|   0     0 | 811k   16k:  31k    0 |   0   13k| 67  28   0   0   0   5|   0 |0.
```

similar to vmstat

similar to iostat
(also per device)

new in live tool

```
-i --ipc -D total,sda --top-io --noupdate 5
---dsk/sda-- ---system-- ----total-cpu-usage---- async sda- -net/total- -pkt/total- inter --sysv-ipc- ----most-expensive----
t: read  writ| int   csw |usr sys idl wai hiq siq| #aio|util| recv  send|#recv #send|  1  |msg sem shm|      i/o process
M: 124k  337k|   0  4968 |  4   3  92   0   0   1|   0 |1.59|   0     0 |   0     0 | 300 |  0  35   1|sshd          15M   25M
k:  45k    0 |   0   16k| 33   9  55   0   0   3|   0 |0.20| 21B  426B|0.40  0.40 |  81 |  0  35   1|postgres: p  78k     0
k:  60k    0 |   0   15k| 63  15  17   0   0   5|   0 |0.20| 10B  148B|0.20  0.20 |  74 |  0  35   1|postgres: p  75k     0
k:  32k    0 |   0   11k| 71  17   6   0   0   6|   0 |   0| 142B 148B|0.60  0.20 |  62 |  0  35   1|postgres: p 141k     0
k:  26k    0 |   0   16k| 76  18   0   0   0   6|   0 |0.40| 133B 151B|0.60  0.20 |  75 |  0  35   1|postgres: p  32k     0
k:  37k    0 |   0   14k| 75  19   0   0   0   6|   0 |0.40| 10B  151B|0.20  0.20 |  73 |  0  35   1|postgres: p 152k     0
k:  31k    0 |   0   13k| 67  28   0   0   0   5|   0 |0.20| 162B 161B|0.80  0.40 |  53 |  0  35   1|postgres: p  22k     0
```

# Perf

- Characteristics: Easy to use profiling and kernel tracing
- Objective: Get detailed information where & why CPU is consumed
- Usage: `perf` (to begin with)
- Package: RHEL: perf SLES: perf

- Shows
  - Sampling for CPU hotspots
    - Annotated source code along hotspots
  - CPU event counters
  - Further integrated non-sampling tools

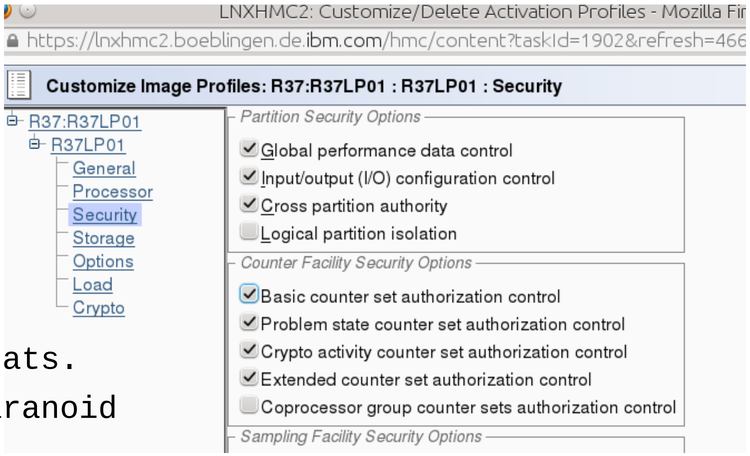
- Hints
  - Without HW support only userspace can be reasonably profiled
  - “successor” of oprofile that is available with HW support (SLES11-SP2)
  - Perf HW support partially upstream, wait for next distribution releases

# Perf

- What profiling can and what it can't
  - + Search hotspots of CPU consumption worth to optimize
  - + List functions according to their usage
  - - Search where time is lost (I/O, Stalls)

- Perf is not just a sampling tool
  - Integrated tools to evaluate tracepoints like
    "perf sched", "perf timechart", …
    - Other than real "sampling" this can help to search for stalls
  - Counters provide even lower overhead and report HW and Software events

# Perf stat - preparation




- Activate the cpu measurement facility
  - If not you'll encounter this

```
Error: You may not have permission to collect stats.
Consider tweaking /proc/sys/kernel/perf_event_paranoid
Fatal: Not all events could be opened.
```

  - Check if its activated

```
echo p > /proc/sysrq-trigger
dmesg
[...]
SysRq : Show Regs
perf.ee05c5: CPU[0] CPUM_CF: ver=1.2 A=000F E=0000 C=0000
[...]
```

  - A = authorized, E=enabled (ready for use), C=controlled (currently running)
  - F = last four bits for basic, problem, crypto and extended set

# Perf stat - usage

```
perf stat -B --event=cycles,instructions,r20,r21,r3,r5,sched:sched_wakeup  find / -iname "*foobar*"

 Performance counter stats for 'find / -iname *foobar*':
     3,623,031,935 cycles                    #    0.000 GHz
     1,515,404,340 instructions              #    0.42  insns per cycle
     1,446,545,776 r20
       757,589,098 r21
       705,740,759 r3
       576,226,424 r5
            40,675 sched:sched_wakeup
       6.156288957 seconds time elapsed
```

- Events
  - Cycles/Instructions globally
  - R20,R21 – Cycles/Instructions of Problem state
  - R3/R5 – Penalty cycles due for L1 instruction/data cache
  - Not only HW events, you can use any of the currently 163 tracepoints
- Further releases will make that readable and work with few arguments
  - Until then you can refer to this document to get the event numbers

    The Load-Program-Parameter and CPU-Measurement Facilities

# Java Performance in general

- "Too" many choices
  - There are many Java performance tools out there
- Be aware of common Java myths often clouding perception
- Differences
  - Profiling a JVM might hide the Java methods
  - Memory allocation of the JVM isn't the allocation of the Application

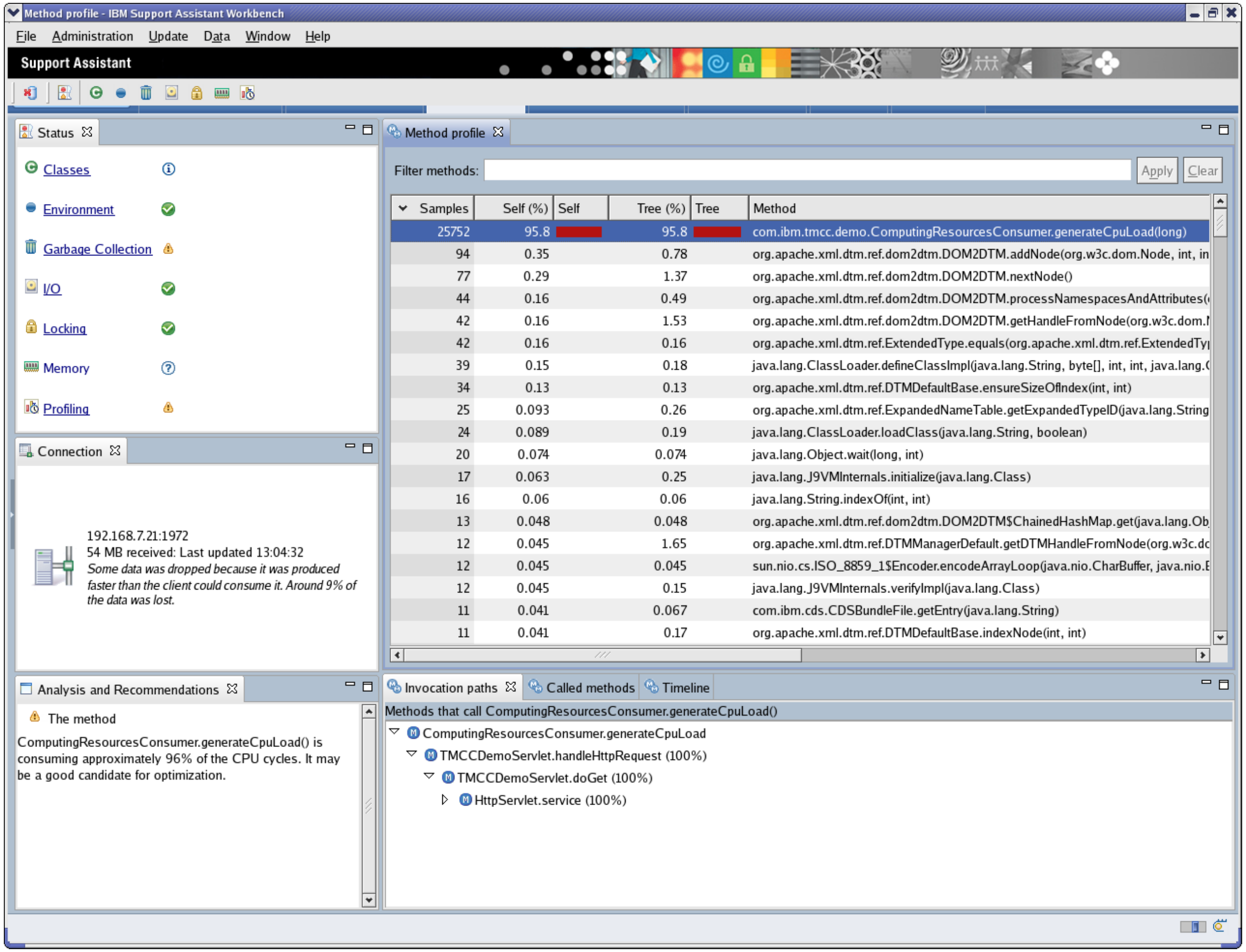
# Java - Health Center

- Characteristics: Lightweight Java Virtual Machine Overview
- Objective: Find out where memory is leaked, sub-optimally cached, ...
- Usage: `IBM Support Assistant (Eclipse)`
- Package: RHEL: n/a SLES: n/a WWW: ibm.com/developerworks/java/jdk/tools/healthcenter
  Java Agents integrated V5SR10+, V6SR3+, usually no target install required

- Shows
  - Memory usage
  - Method Profiling
  - I/O Statistics
  - Class loading
  - Locking

- Hints
  - Low overhead, therefore even suitable for monitoring
  - Agent activation -Xhealthcenter:port=12345
  - Can trigger dumps or verbosegc for in-depth memory analysis

# Health Center - example




- Example of method profiling

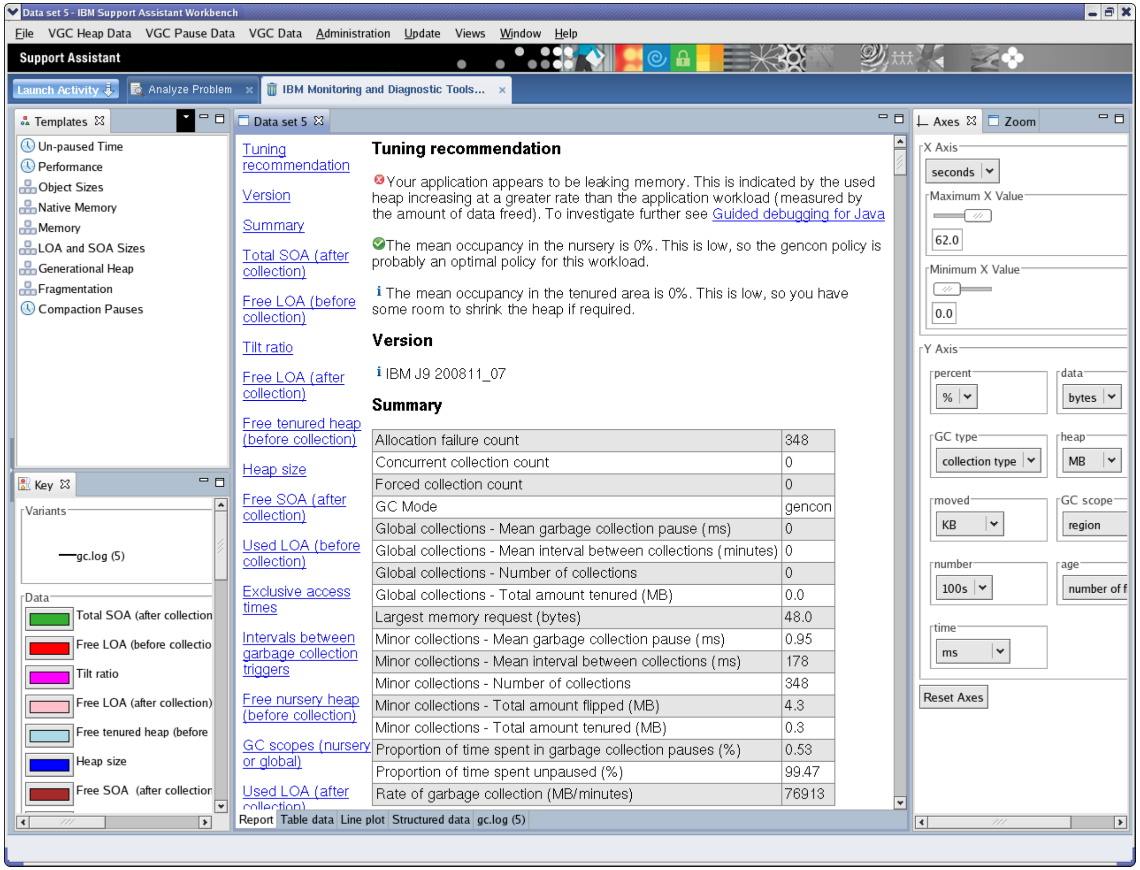
# Java - Garbage Collection and Memory Visualizer

- Characteristics: in-depth Garbage Collection analysis
- Objective: Analyze JVM memory management
- Usage: `IBM Support Assistant (Eclipse)`
- Package: RHEL: n/a SLES: n/a WWW: ibm.com/developerworks/java/jdk/tools/gcmv
  reads common verbosegc output, so usually no target install required

- Shows
  - Memory usage
  - Garbage Collection activities
  - Pauses
  - Memory Leaks by stale references

- Hints
  - GCMV can also compare output of two runs
  - Activate verbose logs `-verbose:gc -Xverbosegclog:<log_file>`

# Garbage Collection and Memory Visualizer




- Most important values / indicators are:
  - Proportion of time spent in gc pauses (should be less than 5%)
  - For gencon: global collections << minor collections

# IRQ Statistics

- Characteristics: Low overhead IRQ information
- Objective: Condensed overview of IRQ activity
- Usage: `cat /proc/interrupts` and `cat /proc/softirqs`
- Package: n/a (Kernel interface)

- Shows
  - Which interrupts happen on which cpu
  - Where softirqs and tasklets take place

- Hints
  - Recent Versions (SLES11-SP2) much more useful due to better naming
  - If interrupts are unintentionally unbalanced
  - If the amount of interrupts matches I/O
    - This can point to non-working IRQ avoidance

# IRQ Statistics

- Example
  - Network focused on CPU zero (in this case unwanted)
  - Scheduler covered most of that avoiding idle CPU 1-3
  - But caused a lot migrations, IPI's and cache misses

```
              CPU0        CPU1        CPU2        CPU3
EXT:         21179       24235       22217       22959
I/O:       1542959      340076      356381      325691
CLK:         15995       16718       15806       16531   [EXT] Clock Comparator
EXC:           255         325         332         227   [EXT] External Call
EMS:          4923        7129        6068        6201   [EXT] Emergency Signal
TMR:             0           0           0           0   [EXT] CPU Timer
TAL:             0           0           0           0   [EXT] Timing Alert
PFL:             0           0           0           0   [EXT] Pseudo Page Fault
DSD:             0           0           0           0   [EXT] DASD Diag
VRT:             0           0           0           0   [EXT] Virtio
SCP:             6          63          11           0   [EXT] Service Call
IUC:             0           0           0           0   [EXT] IUCV
CPM:             0           0           0           0   [EXT] CPU Measurement
CIO:           163         310         269         213   [I/O] Common I/O Layer Interrupt
QAI:     1 541 773     338 857     354 728     324 110   [I/O] QDIO Adapter Interrupt
DAS:          1023         909        1384        1368   [I/O] DASD
[…] 3215, 3270, Tape, Unit Record Devices, LCS, CLAW, CTC, AP Bus, Machine Check
```

# IRQ Statistics II

- Also softirqs can be tracked which can be useful to
  - check if tasklets execute as intended
  - See if network, scheduling and I/O behave as expected

```
                 CPU0       CPU1       CPU2       CPU3
         HI:      498       1522       1268       1339
      TIMER:     5640        914        664        643
     NET_TX:       15         16         52         32
     NET_RX:       18         34         87         45
      BLOCK:        0          0          0          0
BLOCK_IOPOLL:       0          0          0          0
    TASKLET:       13         10         44         20
      SCHED:     8055        702        403        445
    HRTIMER:        0          0          0          0
        RCU:     5028       2906       2794       2564
```

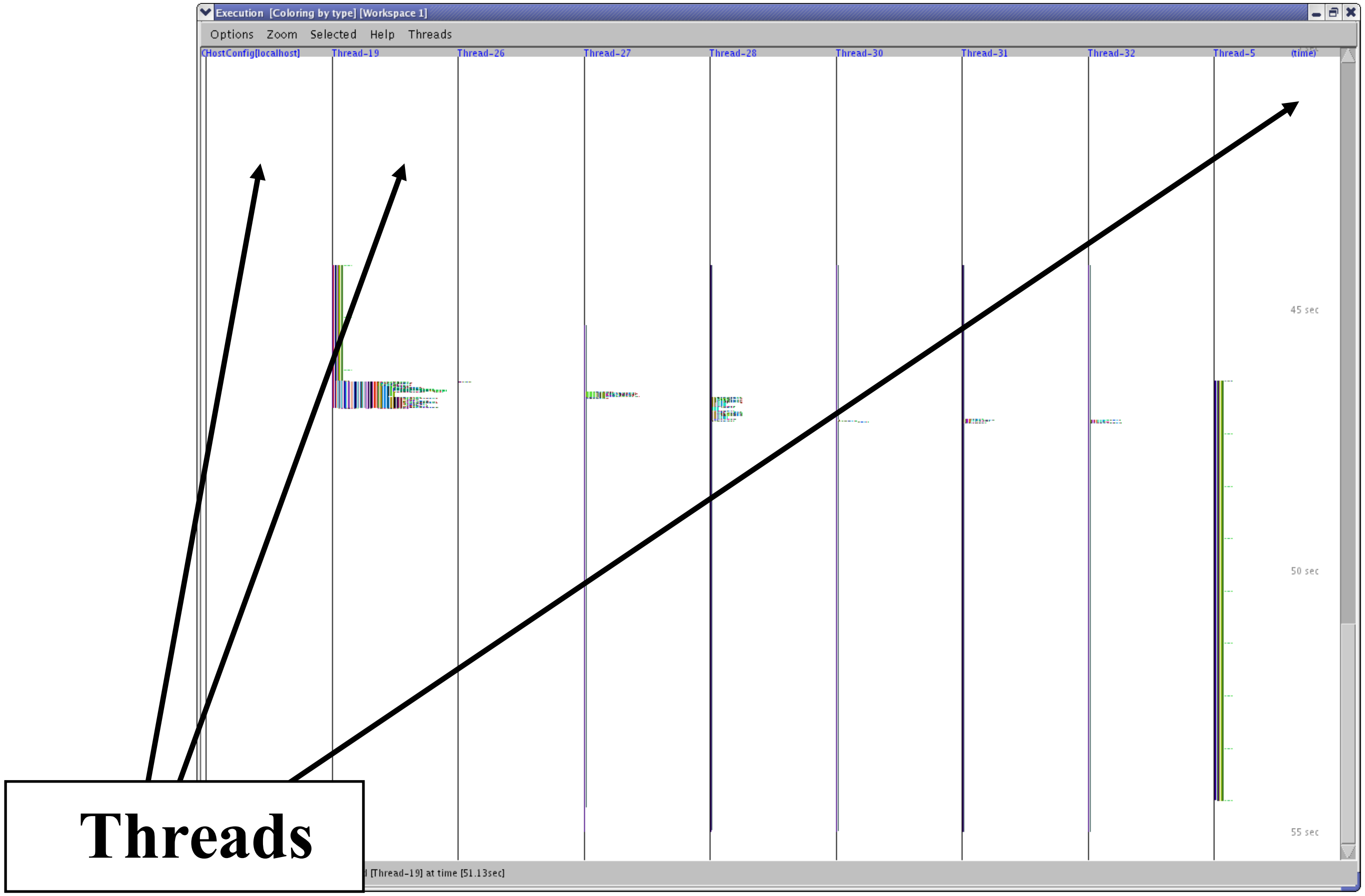
# Java - Jinsight

- Characteristics: zoomable call stack
- Objective: Analyze method call frequency and duration
- Usage: `jinsight_trace -tracemethods <yourProgram> <yourProgramArgs>`
- Package: RHEL: n/a SLES: n/a WWW: IBM alphaworks

- Shows
  - Call Stack and time

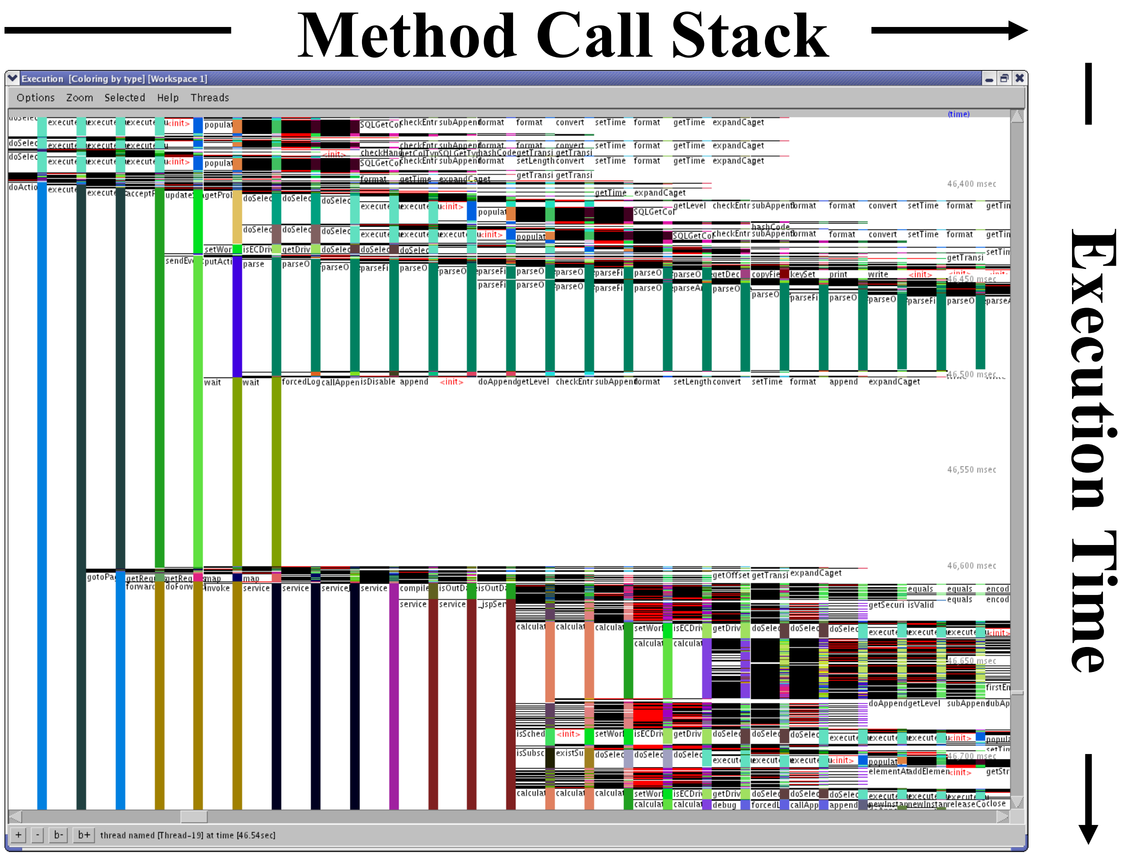
- Hints
  - Significant slowdown, not applicable to production systems
  - No more maintained, but so far still working

# Jinsight Execution View




- Threads in columns, select one to zoom in

# Jinsight Execution View, *continued*




- Many horizontal stages mean deep call stacks
- Long vertical areas mean long method execution
- Rectangles full of horizontal lines can be an issue

# Perf profiling

- Perf example how-to
  - We had a case where new code caused cpus to scale badly
  - `perf record “workload”`
    - Creates a file called perf.data that can be analyzes
  - We used “`perf diff`” on both data files to get a comparison

- “Myriad” of further options/modules
  - Live view with perf top
  - Perf sched for an integrated analysis of scheduler tracepoints
  - Perf annotate to see samples alongside code
  - Perf stat for a counter based analysis
  - [...]

# Perf profiling

- Perf example (perf diff)
  - found a locking issue causing increased cpu consumption

```
# Baseline  Delta                           Symbol
# ........ ..........  ................... ......
#
    12.14%     +8.07%  [kernel.kallsyms]   [k] lock_acquire
     8.96%     +5.50%  [kernel.kallsyms]   [k] lock_release
     4.83%     +0.38%  reaim               [.] add_long
     4.22%     +0.41%  reaim               [.] add_int
     4.10%     +2.49%  [kernel.kallsyms]   [k] lock_acquired
     3.17%     +0.38%  libc-2.11.3.so      [.] msort_with_tmp
     3.56%     -0.37%  reaim               [.] string_rtns_1
     3.04%     -0.38%  libc-2.11.3.so      [.] strncat
```

# HTOP

- Characteristics: Process overview with extra features
- Objective: Get a understanding about your running processes
- Usage: `htop`
- Package: RHEL: n/a SLES: n/a WWW: http://htop.sourceforge.net/
- Shows
  - Running processes
  - CPU and memory utilization
  - Accumulated times
  - I/O rates
  - System utilization visualization

- Hints
  - Htop can display more uncommon fields (in menu)
  - Able to send signals out of its UI for administration purposes
  - Processes can be sorted/filtered for a more condensed view

# htop

Configurable utilization visualization

```
0   [|||||||||||||||||||||||||||||||||||||||||||||13.1%]     Tasks: 101, 80 thr; 60 running
1   [|||||||||||||||||||||||||||||||||||||||||||||14.5%]     Load average: 42.03 16.67 6.24
2   [||||||||||||||||||||||||||||||||||||||||||||||1.9%]     Uptime: 00:17:11
3   [|||||||||||||||||||||||||||||||||||||||||||||13.6%]
4   [||||||||||||||||||||||||||||||||||||||||||||||0.0%]
5   [|||||||||||||||||||||||||||||||||||||||||||||14.1%]
Mem[||||||||||||||||||||||||||||||||||||||||945/12059MB]
Swp[                                              0/0MB]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM% - -  UTIME+   STIME+   IORR  IOWR   TIME+  Command
51931 postgres   20   0 3264M  142M  140M S  1.0  1.2 - -  0:00.47  0:00.21   627     0  0:00.68 postgres:
51962 postgres   20   0 3264M  157M  154M R  3.0  1.3 - -  0:00.56  0:00.24   483     0  0:00.80 postgres:
51981 postgres   20   0 3264M  170M  168M R  3.0  1.4 - -  0:00.61  0:00.26   424     0  0:00.87 postgres:
51921 postgres   20   0 3264M  164M  162M R  1.0  1.4 - -  0:00.57  0:00.25   398     0  0:00.83 postgres:
51953 postgres   20   0 3264M  169M  166M R  1.0  1.4 - -  0:00.62  0:00.27   280     0  0:00.89 postgres:
51934 postgres   20   0 3264M  174M  172M R  2.0  1.4 - -  0:00.64  0:00.27   269     0  0:00.91 postgres:
51923 postgres   20   0 3264M  156M  153M R  3.0  1.3 - -  0:00.55  0:00.26   269     0  0:00.81 postgres:
51933 postgres   20   0 3264M  154M  151M S  1.0  1.3 - -  0:00.55  0:00.26   251     0  0:00.81 postgres:
51942 postgres   20   0 3264M  178M  175M R  1.0  1.5 - -  0:00.68  0:00.31   205     0  0:00.99 postgres:
51946 postgres   20   0 3264M  139M  136M R  1.0  1.2 - -  0:00.47  0:00.22   200     0  0:00.69 postgres:
```

Common process info

Accumulated Usage and IO rates

Hierarchy

# Tracepoints (Events)

- Characteristics: Complex interface, but a vast source of information
- Objective: In kernel latency and activity insights
- Usage: Access debugfs mount point /tracing
- Package: n/a (Kernel interface)

- Shows
  - Timestamp and activity name
  - Tracepoints can provide event specific context data
  - Infrastructure adds extra common context data like cpu, preempts depth, ...

- Hints
  - Very powerful and customizable, there are hundreds of tracepoints
    - Some tracepoints have tools to be accessed “perf sched”, “blktrace” both base on them
    - Others need custom postprocessing
  - There are much more things you can handle with tracepoints check out
    Kernel Documentation/trace/tracepoint-analysis.txt (via perf stat)
    Kernel Documentation/trace/events.txt (custom access)

# Tracepoints – example I/III

- Here we use custom access since there was tool
  - We searched for 1.2ms extra latency
    - Target is it lost in HW, Userspace, Kernel or all of them
  - Workload was a simple 1 connection 1 byte←→1 byte load
  - Call "`perf list`" for a list of currently supported tracepoints
  - We used the following tracepoints

| Abbreviation | Tracepoint | Meaning |
|---|---|---|
| R | netif_receive_skb | low level receive |
| P | napi_poll | napi work related to receive |
| Q | net_dev_queue | enqueue in the stack |
| S | net_dev_xmit | low level send |

# Tracepoints – example II/III

## – (Simplified) Script

- # full versions tunes buffer sizes, checks files, ...

```
echo latency-format > /sys/kernel/debug/tracing/trace_options          # enable tracing type
echo net:* >> /sys/kernel/debug/tracing/set_event                      # select specific events
echo napi:* >> /sys/kernel/debug/tracing/set_event                     # “
echo "name == ${dev}" > /sys/kernel/debug/tracing/events/net/filter    # set filters
echo "dev_name == ${dev}" > /sys/kernel/debug/tracing/events/napi/filter # “
cat /sys/kernel/debug/tracing/trace >> ${output}                       # synchronous
echo !*:* > /sys/kernel/debug/tracing/set_event                        # disable tracing
```

## – Output

```
#                _------=> CPU#
#               / _-----=> irqs-off
#              | / _----=> need-resched
#              || / _---=> hardirq/softirq
#              ||| / _--=> preempt-depth
#              |||| /     delay
#  cmd     pid ||||| time  |   caller
#     \   /    |||||  \    |   /
   <...>-24116 0..s. 486183281us+: net_dev_xmit: dev=eth5 skbaddr=0000000075b7e3e8 len=67 rc=0
  <idle>-0     0..s. 486183303us+: netif_receive_skb: dev=eth5 skbaddr=000000007ecc6e00 len=53
  <idle>-0     0.Ns. 486183306us+: napi_poll: napi poll on napi struct 000000007d2479a8 fordevice eth
   <...>-24116 0..s. 486183311us+: net_dev_queue: dev=eth5 skbaddr=0000000075b7e3e8 len=67
   <...>-24116 0..s. 486183317us+: net_dev_xmit: dev=eth5 skbaddr=0000000075b7e3e8 len=67 rc=0
```

# Tracepoints – example III/III

- Example postprocessed

| | SUM | COUNT | AVERAGE | MIN | MAX | STD-DEV |
|---|---|---|---|---|---|---|
| P2Q: | 8478724 | 1572635 | 5.39 | 4 | 2140 | 7.41 |
| Q2S: | 12188675 | 1572638 | 7.65 | 3 | 71 | 4.89 |
| S2R: | 38562294 | 1572636 | 24.42 | 1 | 2158 | 9.08 |
| R2P: | 4197486 | 1572633 | 2.57 | 1 | 43 | 2.39 |
| SUM: | 63427179 | 1572635 | **40.03** | | | |

| | SUM | COUNT | AVERAGE | MIN | MAX | STD-DEV |
|---|---|---|---|---|---|---|
| P2Q: | 7191885 | 1300897 | 5.53 | 4 | 171 | 1.31 |
| Q2S: | 10622270 | 1300897 | 8.17 | 3 | 71 | 5.99 |
| S2R: | 32078550 | 1300898 | 24.66 | 2 | 286 | 5.88 |
| R2P: | 3707814 | 1300897 | 2.85 | 1 | 265 | 2.59 |
| SUM: | 53600519 | 1300897 | **41.20** | | | |

– Confirmed that ~all of the 1.2 ms were lost inside Linux (not in the fabric)
– Confirmed that it was not at/between specific function tracepoints
  - Eventually it was an interrupt locality issue causing bad caching

# vmstat

- Characteristics: Easy to use, high-level information
- Objective: First and fast impression of the current state
- Usage: vmstat [interval in sec]
- Package: RHEL: sysstat.s390x SLES: sysstat
- Output sample:

```
vmstat 1
procs -----------memory---------- ---swap-- -----io---- -system-- -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 2  2      0 4415152  64068 554100    0    0     4 63144  350   55 29 64  0  3  4
 3  0      0 4417632  64832 551272    0    0     0   988  125   60 32 67  0  0  1
 3  1      0 4415524  68100 550068    0    0     0  5484  212   66 31 64  0  4  1
 3  0      0 4411804  72188 549592    0    0     0  8984  230   42 32 67  0  0  1
 3  0      0 4405232  72896 555592    0    0     0    16  105   52 32 68  0  0  0
```

- Shows
  - Data per time interval
  - CPU utilization
  - Disk I/O
  - Memory usage/Swapping
- Hints
  - Shared memory usage is listed under 'cache'

# sadc/sar

- Characteristics: Very comprehensive, statistics data on device level
- Objective: Suitable for permanent system monitoring and detailed analysis
- Usage (recommended):
  - monitor `/usr/lib64/sa/sadc [-S XALL] [interval in sec] [outfile]`
  - View `sar -A -f [outfile]`
- Package: RHEL: sysstat.s390x SLES: sysstat
- Shows
  - CPU utilization
  - Disk I/O overview and on device level
  - Network I/O and errors on device level
  - Memory usage/Swapping
  - … and much more
  - Reports statistics data over time and creates average values for each item
- Hints
  - sadc parameter “-S XALL” enables the gathering of further optional data
  - Shared memory is listed under 'cache'
  - [outfile] is a binary file, which contains all values. It is formatted using sar
    - enables the creation of item specific reports, e.g. network only
    - enables the specification of a start and end time → time of interest

# SAR - Processes created

```
Linux 2.6.16.60-0.59.1-default (h42lp42)        23/02/10

14:14:55       proc/s
14:15:05         2.69
14:15:15         0.40
14:15:25         0.10
14:15:35         0.30
14:15:45         0.00
Average:         0.70
```

Processes created per second usually small except during startup.
If constantly at a high rate your application likely has an issue.
Be aware – the numbers scale with your system size and setup.

# SAR - Context Switch Rate

root@h42lp27:~

File Edit View Terminal Tabs Help

```
09:24:14 PM   cswch/s
09:24:24 PM    586.13
09:24:34 PM    548.35
09:24:44 PM     53.61
09:24:54 PM     74.10
09:25:04 PM    108.51
09:25:14 PM    601.49
09:25:24 PM    521.81
09:25:34 PM     92.06
09:25:44 PM     73.63
Average:       295.43
```

Context switches per second usually < 1000 per cpu
except during startup or while running a benchmark
if > 10000 your application might have an issue.

# SAR - CPU utilization

Per CPU values:
watch out for
- system time (kernel)
- user (applications)
- irq/soft (kernel, interrupt handling)
- idle (nothing to do)
- iowait time (runnable but waiting for I/O)
- steal time (runnable but utilized somewhere else)

root@h42lp42

File Edit View Terminal Help

```
14:14:55        CPU     %user     %nice   %system   %iowait    %steal     %idle
14:15:05        all     26.64      0.00     12.03     25.92      6.24     29.16
14:15:05          0     43.81      0.00      5.49     23.25      4.99     22.46
14:15:05          1      4.30      0.00     10.19     28.67      9.89     46.95
14:15:05          2     11.81      0.00     28.03     45.15      5.01     10.01
14:15:05          3     46.61      0.00      4.49      6.79      4.99     37.13
14:15:15        all     27.19      0.00     11.93     25.11      7.75     28.01
14:15:15          0     90.60      0.00      3.70      0.00      5.70      0.00
14:15:15          1      9.24      0.00     22.49     41.57      9.24     17.47
14:15:15          2      5.98      0.00     14.64     46.71      9.06     23.61
14:15:15          3      2.90      0.00      6.99     12.09      7.09     70.93
```

# SAR - Network traffic

root@h42lp42

File Edit View Terminal Help

| 14:14:55 | IFACE | rxpck/s | txpck/s | rxkB/s | txkB/s | rxcmp/s | txcmp/s | rxmcst/s |
|---|---|---|---|---|---|---|---|---|
| 14:15:05 | lo | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 14:15:05 | sit0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 14:15:05 | eth0 | 4587.92 | 5278.34 | 307.53 | 482.56 | 0.00 | 0.00 | 0.00 |
| 14:15:15 | lo | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 14:15:15 | sit0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 14:15:15 | eth0 | 4206.40 | 4827.10 | 281.43 | 441.17 | 0.00 | 0.00 | 0.00 |

Per interface statistic of packets/bytes
You can easily derive average packet sizes from that.
Sometimes people expect - and planned for – different sizes.

Has another panel for errors, drops and such events.

# SAR – Disk I/O I – overall

root@h42lp42

File Edit View Terminal Help

```
14:14:55          tps      rtps      wtps   bread/s   bwrtn/s
14:15:05       445.71     61.38    384.33   7715.77  55529.74
14:15:15       192.20     32.90    159.30   7308.80  68233.60
14:15:25       171.70      1.20    170.50      9.60  70798.40
14:15:35       327.25    174.95    152.30   1399.60  68261.88
14:15:45       444.74    310.51    134.23   2484.88  59704.50
Average:       316.35    116.15    200.20   3784.61  64504.50
```

Overview of
- operations per second
- transferred amount

# SAR – Disk I/O II – per device

root@h42lp42

File Edit View Terminal Help

| 14:18:14 | DEV | tps | rd_sec/s | wr_sec/s | avgrq-sz | avgqu-sz | await | svctm | %util |
|---|---|---|---|---|---|---|---|---|---|
| 14:18:24 | dev94-0 | 7.41 | 260.26 | 37.64 | 40.22 | 0.01 | 1.35 | 0.95 | 0.70 |
| 14:18:24 | dev94-4 | 403.20 | 46784.38 | 13756.96 | 150.15 | 5.06 | 12.56 | 2.03 | 81.88 |
| 14:18:24 | dev94-8 | 547.15 | 22830.83 | 21249.25 | 80.56 | 3.42 | 6.25 | 1.39 | 76.18 |
| 14:18:34 | dev94-0 | 8.30 | 557.31 | 10.28 | 68.38 | 0.01 | 1.31 | 0.71 | 0.59 |
| 14:18:34 | dev94-4 | 284.39 | 35453.75 | 35618.18 | 249.91 | 7.82 | 23.45 | 2.97 | 84.58 |
| 14:18:34 | dev94-8 | 549.51 | 16032.41 | 41554.94 | 104.80 | 25.23 | 40.35 | 1.42 | 78.06 |

Is your I/O balanced across devices?
Imbalances can indicate issues wit a LV setup.

tps and avgrq-sz combined can be important.
Do they match your sizing assumptions?

Await shows the time the application has to wait.

# SAR - Memory statistics - the false friend

root@h42lp42

File Edit View Terminal Help

| 14:18:14 | kbmemfree | kbmemused | %memused | kbbuffers | kbcached | kbswpfree | kbswpused | %swpused | kbswpcad |
|---|---|---|---|---|---|---|---|---|---|
| 14:18:24 | 9616 | 2045284 | 99.53 | 2772 | 90328 | 1621184 | 782792 | 32.56 | 616916 |
| 14:18:34 | 8624 | 2046276 | 99.58 | 2936 | 154636 | 1443732 | 960244 | 39.94 | 729948 |
| 14:18:44 | 7024 | 2047876 | 99.66 | 5400 | 240140 | 1132356 | 1271620 | 52.90 | 953644 |
| 14:18:54 | 7308 | 2047592 | 99.64 | 4556 | 348796 | 1201988 | 1201988 | 50.00 | 778752 |
| 14:19:04 | 7876 | 2047024 | 99.62 | 7800 | 333844 | 1201988 | 1201988 | 50.00 | 780656 |
| Average: | 8090 | 2046810 | 99.61 | 4693 | 233549 | 1320250 | 1083726 | 45.08 | 771983 |

Be aware that high %memused and low kbmemfree
is no indication of a memory shortage (common mistake).

Same for swap – to use swap is actually good,
but to access it (swapin/-out) all the time is bad.

# SAR - Memory pressure - Swap

root@h42lp42

File Edit View Terminal Help

```
14:18:14      pswpin/s pswpout/s
14:18:24       2853.95   2658.26
14:18:34       2003.26   5399.80
14:18:44         88.59   9921.92
14:18:54       3199.30     53.15
14:19:04       4057.46      0.00
Average:       2443.91   3598.50
```

The percentage seen before can be high,
But the swap rate shown here should be low.
Ideally it is near zero after a rampup time.
High rates can indicate memory shortages.

# SAR - Memory pressure – faults and reclaim

h37lp19 : root

File Edit View Scrollback Bookmarks Settings Help

| 10:12:15 AM | pgpgin/s | pgpgout/s | fault/s | majflt/s | pgfree/s | pgscank/s | pgscand/s | pgsteal/s | %vmeff |
|---|---|---|---|---|---|---|---|---|---|
| 10:12:17 AM | 109.45 | 336.32 | 634.83 | 1.99 | 4710.95 | 0.00 | 0.00 | 0.00 | 0.00 |
| 10:12:19 AM | 174.00 | 18.00 | 109.00 | 1.00 | 76.50 | 0.00 | 0.00 | 0.00 | 0.00 |
| 10:12:21 AM | 0.00 | 18.00 | 36.00 | 0.00 | 71.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 10:12:23 AM | 826.00 | 327910.00 | 1697.00 | 8.50 | 64659.00 | 66066.50 | 5424.50 | 64285.50 | 89.92 |
| 10:12:25 AM | 577.11 | 715393.03 | 43.28 | 1.49 | 178377.61 | 110505.47 | 96352.24 | 178305.97 | 86.20 |
| 10:12:27 AM | 588.12 | 679320.79 | 43.07 | 1.49 | 169312.87 | 101317.82 | 94495.54 | 169250.00 | 86.43 |
| 10:12:29 AM | 1040.00 | 688822.00 | 62.00 | 2.50 | 171417.50 | 99329.50 | 100065.50 | 171355.50 | 85.94 |
| 10:12:31 AM | 698.04 | 663082.35 | 45.59 | 2.45 | 165792.65 | 93984.80 | 95946.57 | 165715.69 | 87.25 |
| 10:12:33 AM | 1212.12 | 624048.48 | 84.34 | 4.55 | 155524.75 | 90932.32 | 87934.85 | 155378.28 | 86.87 |
| 10:12:35 AM | 595.07 | 215950.74 | 68.47 | 2.46 | 54027.09 | 27919.70 | 32992.61 | 53903.45 | 88.49 |
| 10:12:37 AM | 558.00 | 159790.00 | 43.50 | 1.50 | 38183.00 | 18968.50 | 21232.00 | 38122.50 | 94.83 |
| 10:12:39 AM | 1569.85 | 21949.75 | 102.51 | 4.02 | 5976.38 | 3144.72 | 2990.95 | 5868.84 | 95.65 |
| 10:12:41 AM | 1081.55 | 527207.77 | 213.59 | 1.46 | 134243.20 | 65822.33 | 90253.40 | 134170.87 | 85.97 |
| 10:12:43 AM | 1718.59 | 702936.68 | 62.31 | 2.51 | 176173.37 | 86268.34 | 118320.10 | 176107.54 | 86.08 |
| 10:12:45 AM | 1237.44 | 683623.65 | 42.86 | 1.48 | 171228.57 | 83624.14 | 114011.33 | 171166.01 | 86.61 |
| 10:12:47 AM | 1269.39 | 699144.90 | 44.39 | 1.53 | 173979.08 | 89181.63 | 112045.41 | 173909.69 | 86.42 |
| 10:12:49 AM | 1691.54 | 677327.36 | 62.19 | 2.49 | 171114.93 | 89499.00 | 104974.13 | 171048.26 | 87.95 |
| 10:12:51 AM | 1979.70 | 285857.87 | 143.15 | 4.57 | 69777.16 | 37740.61 | 44834.01 | 69590.86 | 84.28 |
| 10:12:53 AM | 458.00 | 20.00 | 57.00 | 1.00 | 156.50 | 98.00 | 0.00 | 96.00 | 97.96 |
| 10:12:55 AM | 433751.72 | 5944.83 | 1818.23 | 24.14 | 109168.97 | 3466.01 | 210573.89 | 108376.85 | 50.63 |
| 10:12:57 AM | 924042.00 | 336.00 | 248.50 | 26.50 | 231500.50 | 2384.50 | 461443.00 | 231027.50 | 49.81 |
| 10:12:59 AM | 906810.00 | 214.00 | 225.00 | 25.00 | 226950.50 | 2117.00 | 447010.00 | 226605.50 | 50.45 |
| 10:13:01 AM | 917504.00 | 180.00 | 206.00 | 38.00 | 230020.50 | 2225.00 | 460268.00 | 229486.50 | 49.62 |
| 10:13:03 AM | 865062.00 | 348.00 | 464.00 | 58.00 | 216892.50 | 7677.50 | 419680.50 | 215976.00 | 50.54 |
| 10:13:05 AM | 12.06 | 20.10 | 42.21 | 0.00 | 267.84 | 160.80 | 16.08 | 176.88 | 100.00 |
| 10:13:07 AM | 770.15 | 123.38 | 201.99 | 9.95 | 266.17 | 131.34 | 0.00 | 130.85 | 99.62 |
| 10:13:09 AM | 484.42 | 20.10 | 64.32 | 2.51 | 263.82 | 192.96 | 0.00 | 192.96 | 100.00 |
| 10:13:11 AM | 16.00 | 20.00 | 38.50 | 0.50 | 96.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 10:13:13 AM | 0.00 | 20.00 | 36.00 | 0.00 | 95.50 | 0.00 | 0.00 | 0.00 | 0.00 |
| 10:13:15 AM | 2700.00 | 318.00 | 698.00 | 9.00 | 22343.00 | 301.00 | 0.00 | 287.00 | 95.35 |
| 10:13:17 AM | 2.00 | 20.00 | 36.00 | 0.00 | 60.50 | 0.00 | 0.00 | 0.00 | 0.00 |
| 10:13:19 AM | 0.00 | 20.00 | 36.00 | 0.00 | 61.50 | 0.00 | 0.00 | 0.00 | 0.00 |

Don't trust pgpgin/-out absolute values
Faults populate memory
Major faults need I/O
Scank/s is background reclaim by kswap/flush (modern)
Scand/s is reclaim with a “waiting” allocation
Steal is the amount reclaimed by those scans

# SAR - System Load

| 14:14:55 | runq-sz | plist-sz | ldavg-1 | ldavg-5 | ldavg-15 |
|---|---|---|---|---|---|
| 14:15:05 | 3 | 87 | 3.76 | 3.69 | 3.70 |
| 14:15:15 | 4 | 87 | 4.10 | 3.76 | 3.72 |
| 14:15:25 | 3 | 88 | 4.54 | 3.87 | 3.76 |
| 14:15:35 | 2 | 89 | 4.45 | 3.87 | 3.76 |
| 14:15:45 | 2 | 87 | 4.70 | 3.94 | 3.78 |
| Average: | 3 | 88 | 4.31 | 3.83 | 3.74 |

Runqueue size are the currently runnable programs.
It's not bad to have many, but if they exceed the amount
of CPUs you could do more work in parallel.

Plist-sz is the overall number of programs, if that is always
growing you have likely a process starvation or connection issue.

Load average is a runqueue length average for 1/5/15 minutes.

# iostat

- Characteristics: Easy to use, information on disk device level
- Objective: Detailed input/output disk statistics
- Usage: iostat -xtdk [interval in sec]
- Package: RHEL: sysstat.s390x SLES: sysstat

- Shows
  - Throughput
  - Request merging
  - Device queue information
  - Service times

- Hints
  - Most critical parameter often is *await*
    - average time (in milliseconds) for I/O requests issued to the device to be served.
    - includes the time spent by the requests in queue and the time spent servicing them.
  - Also suitable for network file systems

# iostat

- Output sample:

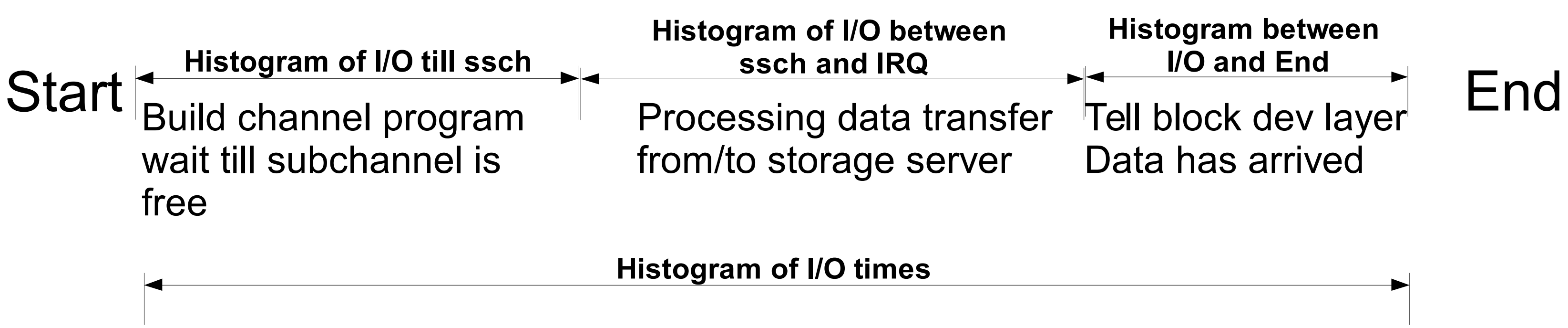
```
Time: 10:56:35 AM
Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await  svctm  %util
dasda             0.19     1.45    1.23    0.74    64.43     9.29    74.88     0.01    2.65   0.80   0.16
dasdb             0.02   232.93    0.03    9.83     0.18   975.17   197.84     0.98   99.80   1.34   1.33

Time: 10:56:36 AM
Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   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
dasdb             0.00  1981.55    0.00  339.81     0.00  9495.15    55.89     0.91    2.69   1.14  38.83

Time: 10:56:37 AM
Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   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
dasdb             0.00  2055.00    0.00  344.00     0.00  9628.00    55.98     1.01    2.88   1.19  41.00
```

# DASD statistics

- Characteristics: Easy to use, very detailed
- Objective: Collects statistics of I/O operations on DASD devices
- Usage:
  - enable: `echo on > /proc/dasd/statistics`
  - show:
    - Overall `cat /proc/dasd/statistics`
    - for individual DASDs `tunedasd -P /dev/dasda`

New Tool “dasdstat” available to handle that all-in-one

- Package: n/a for kernel interface, s390-tools for dasdstat
- Shows:
  - various processing times:



# DASD statistics - report

- Sample:

8*512b = 4KB <= request size < 1*512b =8KB

1ms <= response time < 2 ms

```
29432 dasd I/O requests
with 6227424 sectors(512B each)
   __<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

Histogram of sizes (512B secs)
      0       0    9925    3605    1866    4050    4102     933    2700    2251       0       0       0       0       0       0
      0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0

Histogram of I/O times (microseconds)
      0       0       0       0       0       0       0    1283    1249    6351    7496    3658    8583     805       7       0
      0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0

Histogram of I/O time till ssch
   2314     283      98      34      13       5      16     275     497    8917    5567    4232    7117      60       4       0
      0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
Histogram of I/O time between ssch and irq
      0       0       0       0       0       0       0   14018    7189    2402    1031    4758      27       4       3       0
      0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0

Histogram of I/O time between irq and end
   2733       6    5702    9376    5781     940    1113    3781       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    2740     628    1711    1328   23024       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
```

- Hints
  – Also shows data per sector which usually only confused

# FCP statistics

- Characteristics: Detailed latency information (SLES9 and SLES10)
- Objective: Collects statistics of I/O operations on FCP devices on request base, separate for read/write
- Package: n/a (Kernel interface)

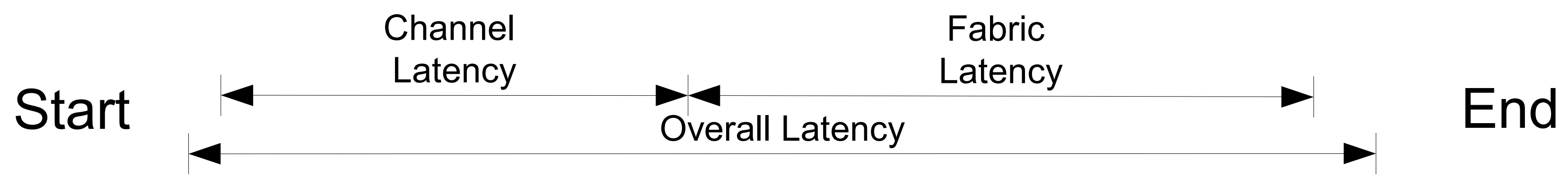
- Usage:
  - enable
    - CONFIG_STATISTICS=y must be set in the kernel config file
    - debugfs is mounted at /sys/kernel/debug/
    - For a certain LUN in directory `/sys/kernel/debug/statistics/zfcp-<device-bus-id>-<WWPN>-<LUN>` issue `echo on=1 > definition` (turn off with on=0, reset with data=reset)
  - view
    - `cat /sys/kernel/debug/statistics/zfcp-<device-bus-id>-<WWPN>-<LUN>/data`

- Hint
  - FCP and DASD statistics are not directly comparable, because in the FCP case many I/O requests can be sent to the same LUN before the first response is given. There is a queue at FCP driver entry and in the storage server

# FCP statistics

- Shows:
  - Request sizes in bytes (hexadecimal)
  - Channel latency Time spent in the FCP channel in nanoseconds
  - Fabric latency processing data transfer from/to storage server incl. SAN in nanoseconds
  - (Overall) latencies whole time spent in the FCP layer in milliseconds

  - Calculate the pass through time for the FCP layer as
    `pass through time = overall latency – (channel latency + fabric latency)`
    → Time spent between the Linux device driver and FCP channel adapter inclusive in Hypervisor



# FCP statistics example

```
cat /sys/kernel/debug/statistics/zfcp-0.0.1700-0x5005076303010482-0x4014400500000000/data
...
request_sizes_scsi_read 0x1000 1163
request_sizes_scsi_read 0x80000 805
request_sizes_scsi_read 0x54000 47
request_sizes_scsi_read 0x2d000 44
request_sizes_scsi_read 0x2a000 26
request_sizes_scsi_read 0x57000 25
request_sizes_scsi_read 0x1e000 25
...
latencies_scsi_read <=1 1076
latencies_scsi_read <=2 205
latencies_scsi_read <=4 575
latencies_scsi_read <=8 368
latencies_scsi_read <=16 0
...
channel_latency_read <=16000 0
channel_latency_read <=32000 983
channel_latency_read <=64000 99
channel_latency_read <=128000 115
channel_latency_read <=256000 753
channel_latency_read <=512000 106
channel_latency_read <=1024000 141
channel_latency_read <=2048000 27
channel_latency_read <=4096000 0
...
fabric_latency_read <=1000000 1238
fabric_latency_read <=2000000 328
fabric_latency_read <=4000000 522
fabric_latency_read <=8000000 136
fabric_latency_read <=16000000 0
...
```

request size 4KB, 1163 occurrences

response time <= 1ms

Channel response time <= 32µs
= all below driver

Fabric response time <= 1ms
= once leaving the card

# netstat

- Characteristics: Easy to use, connection information
- Objective: Lists connections
- Usage: `netstat -eeapn`
- Package: RHEL: net-tools SLES: net-tools

- Shows
  - Information about each connection
  - Various connection states

- Hints
  - Inodes and program names are useful to reverse-map ports to applications

# netstat -s

- Characteristics: Easy to use, very detailed information
- Objective: Display summary statistics for each protocol
- Usage: `netstat -s`

- Shows
  - Information to each protocol
  - Amount of incoming and outgoing packages
  - Various error states, for example TCP segments retransmitted!

- Hints
  - Shows accumulated values since system start, therefore mostly the differences between two snapshots are needed
  - There is always a low amount of packets in error or resets
  - Retransmits occurring only when the system is sending data
    When the system is not able to receive, then the sender shows retransmits
  - Use sadc/sar to identify the device

# netstat -s

- Output sample:

```
Tcp:
  15813 active connections openings
  35547 passive connection openings
  305 failed connection attempts
  0 connection resets received
  6117 connections established
  81606342 segments received
  127803327 segments send out
  288729 segments retransmitted
  0 bad segments received.
  6 resets sent
```

# Socket statistics

- Characteristics: Information on socket level
- Objective: Check socket options and weird connection states
- Usage: `ss -aempi`
- Package: RHEL: iproute-2 SLES: iproute2
- Shows
  - Socket options
  - Socket receive and send queues
  - Inode, socket identifiers

- Sample output

```
ss -aempi
 State        Recv-Q Send-Q    Local Address:Port       Peer Address:Port
 LISTEN       0      128                    :::ssh                    :::*
     users:(("sshd",959,4)) ino:7851 sk:ef858000  mem:(r0,w0,f0,t0)
```

- Hints
  - Inode numbers can assist reading strace logs
  - Check long outstanding queue elements

# Top

- Characteristics: Easy to use
- Objective: Shows resource usage on process level
- Usage: `top -b -d [interval in sec]  > [outfile]`
- Package: RHEL: procps SLES: procps

- Shows
  - CPU utilization
  - Detailed memory usage

- Hints
  - Parameter -b enables to write the output for each interval into a file
  - Use -p [pid1, pid2,...] to reduce the output to the processes of interest
  - Configure displayed columns using 'f' key on the running top program
  - Use the 'W' key to write current configuration to ~/.toprc
    → becomes the default

# top (cont.)

- See ~/.toprc file in backup

- Output sample:

```
top - 11:12:52 up  1:11,  3 users,  load average: 1.21, 1.61, 2.03
Tasks:  53 total,   5 running,  48 sleeping,   0 stopped,   0 zombie
Cpu(s):  3.0%us,  5.9%sy,  0.0%ni, 79.2%id,  9.9%wa,  0.0%hi,  1.0%si,  1.0%st
Mem:   5138052k total,   801100k used,  4336952k free,   447868k buffers
Swap:       88k total,        0k used,       88k free,   271436k cached

 PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  P SWAP DATA WCHAN     COMMAND
3224 root      18   0  1820  604  444 R  2.0  0.0   0:00.56 0 1216  252 -         dbench
3226 root      18   0  1820  604  444 R  2.0  0.0   0:00.56 0 1216  252 -         dbench
2737 root      16   0  9512 3228 2540 R  1.0  0.1   0:00.46 0 6284  868 -         sshd
3225 root      18   0  1820  604  444 R  1.0  0.0   0:00.56 0 1216  252 -         dbench
3230 root      16   0  2652 1264  980 R  1.0  0.0   0:00.01 0 1388  344 -         top
   1 root      16   0   848  304  256 S  0.0  0.0   0:00.54 0  544  232 select    init
   2 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 0    0    0 migration migration/0
   3 root      34  19     0    0    0 S  0.0  0.0   0:00.00 0    0    0 ksoftirqd ksoftirqd/0
   4 root      10  -5     0    0    0 S  0.0  0.0   0:00.13 0    0    0 worker_th events/0
   5 root      20  -5     0    0    0 S  0.0  0.0   0:00.00 0    0    0 worker_th khelper
```

- Hints
  - virtual memory: VIRT = SWAP + RES unit KB
  - physical memory used: RES = CODE + DATA unit KB
  - shared memory SHR unit KB

# Linux ps command

- Characteristics: very comprehensive, statistics data on process level
- Objective: reports a snapshot of the current processes
- Usage: “`ps axlf`”
- Package: RHEL: procps SLES: procps

```
  PID   TID NLWP POL USER     TTY        NI PRI PSR P STAT WCHAN         START     TIME %CPU %MEM    VSZ    SZ   RSS - COMMAND
  871   871    1 TS  root     ?          -5  29   0 * S<   kauditd_thre 10:01 00:00:00  0.0  0.0      0     0     0 - [kauditd]
 2835  2835    1 TS  root     pts/2       0  23   0 * Ss+  read_chan    10:38 00:00:00  0.0  0.0   5140   824  2644 - -bash
 3437  3437    1 TS  root     pts/1       0  23   0 * S+   wait4        11:39 00:00:00  0.0  0.0   1816   248   644 - dbench 3
 3438  3438    1 TS  root     pts/1       0  20   0 0 R+   -            11:39 00:00:24 33.1  0.0   1820   252   604 - dbench 3
 3439  3439    1 TS  root     pts/1       0  20   0 0 R+   -            11:39 00:00:23 32.8  0.0   1820   252   604 - dbench 3
 3440  3440    1 TS  root     pts/1       0  20   0 0 R+   -            11:39 00:00:23 31.8  0.0   1820   252   604 - dbench 3
 …
```

- Hints
  - Do not specify blanks inside the -o format string
  - Many more options available

# Lszcrypt / icastats

- Characteristics: overview of s390 crypto HW and libica usage
- Objective: am I really using my crypto hardware
- Usage: "`lszcrypt -VV[V]`" "`cat /proc/icastats`"
- Package: RHEL: s390utils-base SLES: s390-tools

```
lszcrypt -VV
card02: CEX3C       online  hwtype=9  depth=8
request_count=443
card03: CEX3A       offline hwtype=8  depth=8
request_count=0
```

```
Cat/proc/icastats
function | # hardware | # software
----------+------------+-----------
    SHA-1 |          0 |          0
  SHA-224 |          0 |          0
  SHA-256 |          0 |          0
  SHA-384 |          0 |          0
  SHA-512 |          0 |          0
   RANDOM |     187109 |          0
 MOD EXPO |          0 |          0
  RSA CRT |      93554 |          0
  DES ENC |          0 |          0
  DES DEC |          0 |          0
 3DES ENC |          0 |          0
 3DES DEC |          0 |          0
  AES ENC |    2574106 |          0
  AES DEC |    2075854 |          0
 CMAC GEN |          0 |          0
 CMAC VER |          0 |          0
```

- Never assume your HW correctly is used until you confirmed it
  - If not going via libica (e.g. Java pkcs#11 direct to HW you won't see it in icastat)

# lsluns

- Characteristics: overview of multipathing
- Objective: check your multipath setup hierarchy
- Usage: “`lsluns -a`”
- Package: RHEL: s390utils-base SLES: s390-tools

```
lsluns -a
adapter = 0.0.1700
        port = 0x500507630900c7c1
                lun = 0x4020402100000000        /dev/sg0        Disk    IBM:2107900
                lun = 0x4020402200000000        /dev/sg1        Disk    IBM:2107900
                lun = 0x4020402300000000        /dev/sg2        Disk    IBM:2107900
                lun = 0x4021402100000000        /dev/sg3        Disk    IBM:2107900
                lun = 0x4021402200000000        /dev/sg4        Disk    IBM:2107900
                lun = 0x4021402300000000        /dev/sg5        Disk    IBM:2107900
adapter = 0.0.1780
        port = 0x500507630903c7c1
                lun = 0x4020402100000000        /dev/sg17       Disk    IBM:2107900
                lun = 0x4020402200000000        /dev/sg23       Disk    IBM:2107900
                lun = 0x4020402300000000        /dev/sg32       Disk    IBM:2107900
                lun = 0x4021402100000000        /dev/sg39       Disk    IBM:2107900
                lun = 0x4021402200000000        /dev/sg43       Disk    IBM:2107900
                lun = 0x4021402300000000        /dev/sg46       Disk    IBM:2107900
[...]
```

- Lsluns provides a hierarchical view which often easily identifies missing paths, adapters or similar imbalances
- Adapter to WWPN associations can have concurring targets
  - Low overhead, max fallback capability, best performance, ...

# Multipath -ll

- Characteristics: overview of multipathing
- Objective: check your multipath setup configuration
- Usage: “`mutlipath -ll`”
- Package: RHEL: device-mapper-multipath SLES: mutlipath-tools

```
multipath -ll
swap-3of6 (36005076309ffc7c10000000000002022) dm-2 IBM     ,2107900
size=256G features='0' hwhandler='0' wp=rw
`-+- policy='service-time 0' prio=0 status=active
  |- 0:0:20:1075986464 sdb  8:16   active ready running
  |- 1:0:22:1075986464 sdx  65:112 active ready running
  |- 2:0:21:1075986464 sdh  8:112  active ready running
  |- 3:0:20:1075986464 sdn  8:208  active ready running
  |- 4:0:26:1075986464 sdz  65:144 active ready running
  |- 5:0:19:1075986464 sdy  65:128 active ready running
  |- 7:0:25:1075986464 sdac 65:192 active ready running
  `- 6:0:24:1075986464 sdad 65:208 active ready running
[...]
```

- This also reports multipath.conf inconsitencies
- Check all reported parameters are what you thought them to be
  - For example (in)famous rr_min_io renaming

# Systemtap

- Characteristics: tool to “tap” into the kernel for analysis
- Objective: analyze in kernel values or behavior that otherwise would be inaccessible or require a modification/recompile cycle
- Usage (mini example): “`stap -v -e 'probe vfs.read {printf(“read performed\n”); exit()}'`”
- Package: RHEL: systemtap + systemtap-runtime SLES: systemtap
- Also requires kernel debuginfo and source/devel packages

- Procedural and C-like language based on two main constructs
  - Probes – “catching events”
    - On functions, syscalls or single statements via file:linenumber
  - Functions – “what to do”
    - Supports local and global variables
    - Program flow statements if, loops, …
- Tapsets provide pre written probe libraries

- Fore more check out “Using SystemTap on Linux on System z” from Mike O'Reilly https://share.confex.com/share/118/webprogram/Handout/Session10452/atlanta.pdf

# lsqeth

- Characteristics: overview of network devices
- Objective: check your network devices basic setup
- Usage: "`lsqeth -p`"
- Package: RHEL: s390-utils-base SLES: s390-tools

```
lsqeth -p
devices                    CHPID interface  cardtype       port chksum prio-q'ing rtr4 rtr6 lay'2 cnt
-------------------------- ----- ---------- -------------- ---- ------ ---------- ---- ---- ----- -----
0.0.e000/0.0.e001/0.0.e002 x84   eth1       OSD_10GIG      0    sw     always_q_2 n/a  n/a  1     64
0.0.e100/0.0.e101/0.0.e102 x85   eth2       OSD_10GIG      0    sw     always_q_2 n/a  n/a  1     64
0.0.f200/0.0.f201/0.0.f202 x6B   eth0       OSD_1000       0    hw     always_q_2 no   no   0     64
```

- Check for layer, offload, and buffer counts
  - More buffers are usually better especially for massive amounts of concurrent connections

# Ethtool I

- Characteristics: overview of network device capabilities / offload settings
- Objective: check your network device (offload) settings
- Usage: “`ethtool <dev>`”, “`ethtool -k <dev>`”
- Package: RHEL: ethtool SLES: ethtool

```
ethtool eth1
Settings for eth1:
        Supported ports: [ FIBRE ]
        Supported link modes:   10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Half 1000baseT/Full
                                10000baseT/Full
        Supported pause frame use: No
        Supports auto-negotiation: Yes
        Advertised link modes:  10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Half 1000baseT/Full
                                10000baseT/Full
        Advertised pause frame use: No
        Advertised auto-negotiation: Yes
        Speed: 10000Mb/s
        Duplex: Full
        Port: FIBRE
        PHYAD: 0
        Transceiver: internal
        Auto-negotiation: on
        Link detected: yes
```

- Check e.g. announced speeds

# Ethtool II

- Offload Settings via "`ethtool -k <dev>`"
- Changes via upper case "-K"

```
ethtool -k eth1
Features for eth1:
rx-checksumming: off [fixed]
tx-checksumming: off
        tx-checksum-ipv4: off [fixed]
        tx-checksum-ip-generic: off [fixed]
        tx-checksum-ipv6: off [fixed]
        tx-checksum-fcoe-crc: off [fixed]
        tx-checksum-sctp: off [fixed]
scatter-gather: off
        tx-scatter-gather: off [fixed]
        tx-scatter-gather-fraglist: off [fixed]
tcp-segmentation-offload: off
        tx-tcp-segmentation: off [fixed]
        tx-tcp-ecn-segmentation: off [fixed]
        tx-tcp6-segmentation: off [fixed]
udp-fragmentation-offload: off [fixed]
generic-segmentation-offload: off [requested on]
generic-receive-offload: on
large-receive-offload: off [fixed]
rx-vlan-offload: off [fixed]
tx-vlan-offload: off [fixed]
[...]
```

```
[…]
ntuple-filters: off [fixed]
receive-hashing: off [fixed]
highdma: off [fixed]
rx-vlan-filter: on [fixed]
vlan-challenged: off [fixed]
tx-lockless: off [fixed]
netns-local: off [fixed]
tx-gso-robust: off [fixed]
tx-fcoe-segmentation: off [fixed]
tx-gre-segmentation: off [fixed]
tx-udp_tnl-segmentation: off [fixed]
fcoe-mtu: off [fixed]
tx-nocache-copy: off
loopback: off [fixed]
rx-fcs: off [fixed]
rx-all: off [fixed]
tx-vlan-stag-hw-insert: off [fixed]
rx-vlan-stag-hw-parse: off [fixed]
rx-vlan-stag-filter: off [fixed]
```

- In some cases external influences like layer2 prevent most offloads (the example here)

# TCPDump

- Characteristics: dumps network traffic to console/file
- Objective: analyze packets of applications manually
- Usage: "`tcpdump ...`"
- Package: RHEL: tcpdump SLES: tcpdump

```
tcpdump host pserver1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
13:30:00.326581 IP pserver1.boeblingen.de.ibm.com.38620 > p10lp35.boeblingen.de.ibm.com.ssh: Flags [.], ack 3142, win
102, options [nop,nop,TS val 972996696 ecr 346994], length 0

13:30:00.338239 IP p10lp35.boeblingen.de.ibm.com.ssh > pserver1.boeblingen.de.ibm.com.38620: Flags [P.], seq 3142:3222,
ack 2262, win 2790, options [nop,nop,TS val 346996 ecr 972996696], length 80
13:30:00.375491 IP pserver1.boeblingen.de.ibm.com.38620 > p10lp35.boeblingen.de.ibm.com.ssh: Flags [.], ack 3222, win
102, options [nop,nop,TS val 972996709 ecr 346996], length 0
[...]
^C
31 packets captured
31 packets received by filter
0 packets dropped by kernel
```

- Not all devices support dumping packets in older distribution releases
  - Also often no promiscuous mode
- Check flags or even content if your expectations are met
- -w flag exports captured unparsed data to a file for later analysis in libpcap format
  - Also supported by wireshark
- Usually you have to know what you want to look for

# Wireshark

- Characteristics: Analyzes captured network traffic
- Objective: In depth analysis of handshakes, missing replies, protocols, ...
- Usage: Dump in libpcap or pcap-ng format (`tcpdump, dumpcap`)
  then analyze on remote system via “`wireshark`”
- Package: RHEL: wireshark SLES: wireshark

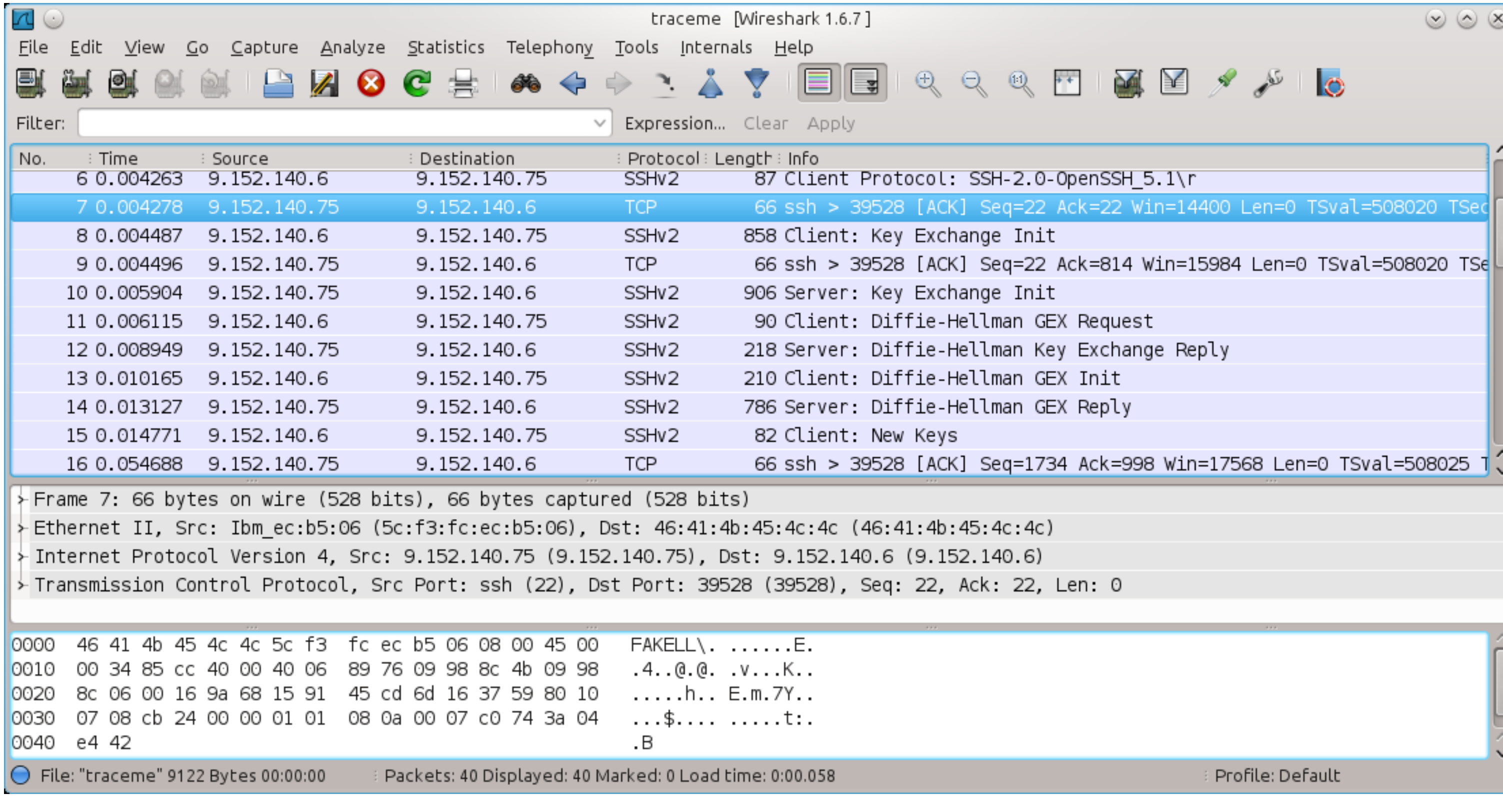
- No “direct” invocation on System z usually
  – e.g. on RH6 there is not even a wireshark binary
- Scrolling huge files on Remote X isn't fun anyway
  – Capturing tools are available

- Custom columns and profiles are important to visualize what you want to look for
- For more details you might start at
  – The share sessions of Mathias Burkhard
    https://share.confex.com/share/121/webprogram/Session13282.html
  – Official documentation http://www.wireshark.org/docs/wsug_html/

# Wireshark example

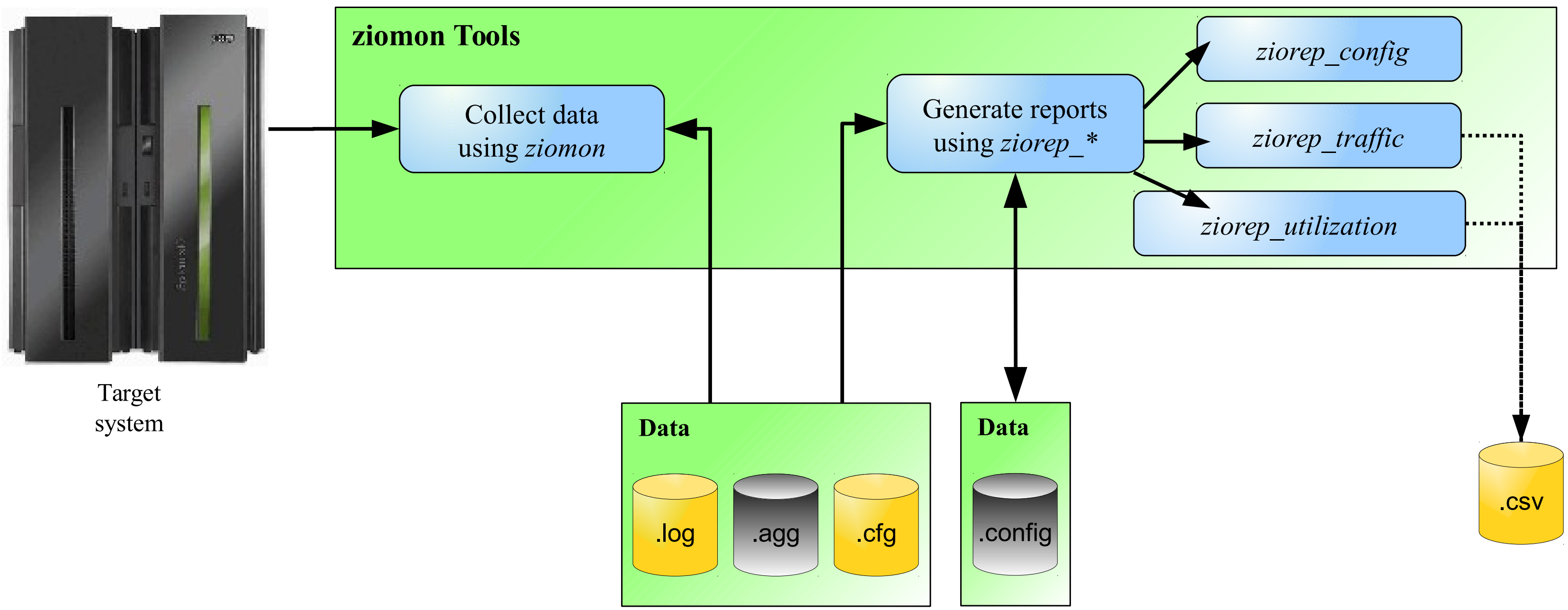
- 1. Dump via "`tcpdump -w`" or wiresharks "`dumpcap`"
- 2. analyze on remote system via "`wireshark`"

```
tcpdump host pserver1 -w traceme
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
^C40 packets captured
40 packets received by filter
0 packets dropped by kernel

[scp to my system]
wireshark traceme
```



# ziomon

- Characteristics: in depth zfcp based I/O analysis
- Objective: Analyze your FCP based I/O
- Usage: "`ziomon`" → "`ziorep*`"
- Package: RHEL: s390utils(-ziomon) SLES: s390-tools




- Be aware that ziomon can be memory greedy if you have very memory constrained systems
- The has many extra functions please check out the live virtual class of Stephan Raspl
  - PDF: http://www.vm.ibm.com/education/lvc/LVC0425.pdf
  - Replay: http://ibmstg.adobeconnect.com/p7zvdjz0yye/

# 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*
*Performance Evaluation*

*Research & Development*
*Schönaicher Strasse 220*
*71032 Böblingen, Germany*

*ehrhardt@de.ibm.com*