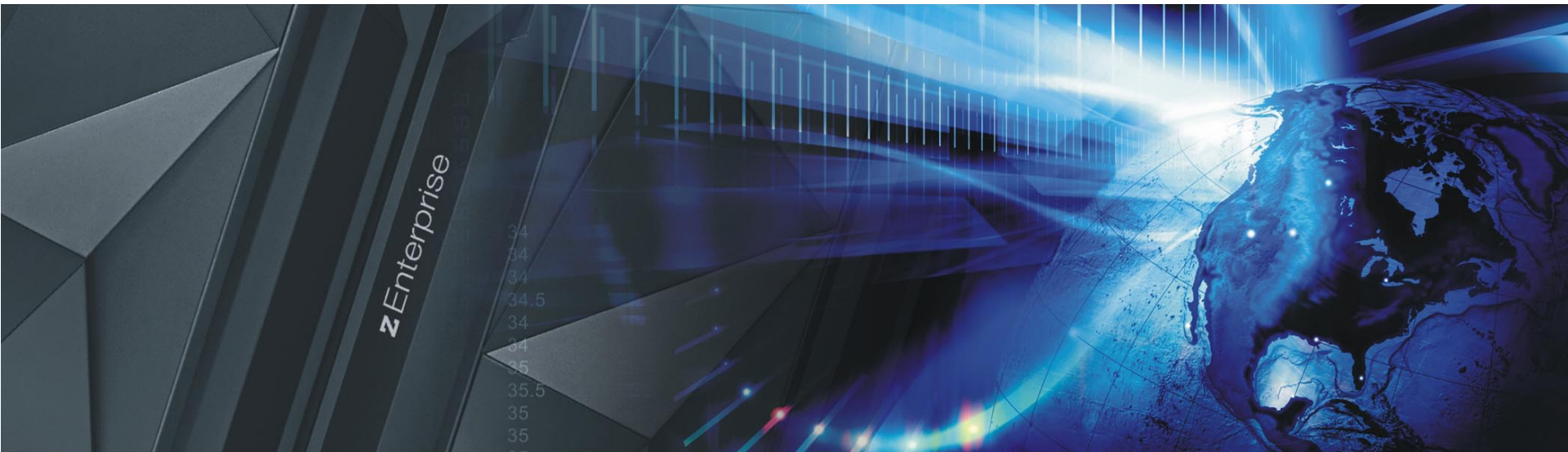


How to Surprise by being a Linux Performance "know-it-all"

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



Trademarks

IBM, the IBM logo, and `ibm.com` are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at www.ibm.com/legal/copytrade.shtml.

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
 - **Hypotop** – 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 / Iszcrypt** – 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
- **Ltng** – complex latency tracing infrastructure
- **Ziemon** – Analyze FCP setup and I/O
- **Systemtap** – another kernel tracing infrastructure
- **Wireshark / Tcpdump** – analyze network traffic in depth
- **Iotop** – order processes by disk I/O
- **Iftop** – per connection traffic overview
- ... ever growing

Structured Agenda

Basic	Intermediate	Advanced	Master	Elite
– Utilization	– General thoughts	– Strace	– Perf	– Cachestat
– Scheduling		– Ltrace	– slabtop	– Smem
– Page Cache	– Sysstat	– Lsof	– Blktrace	– Valgrind
– Swapping	– Dastat	– Lsluns	– Ziemon	– Iqstats
	– Scsi I/O statistics	– Multipath	– Tcpdump	– Wireshark
– top	– iotop	– hytop	– Java Health Center	– Kernel Tracepoints
– ps	– Lszcrpt	– Dstat	– Java Garbage Collection and Memory visualizer	– Systemtap
– vmstat	– icastats	– Htop		
	– Lsqeth	– Netstat	– Jinsight	
	– Ethtool	– Socket Statistics		
	– Preparation	– Iptraf		

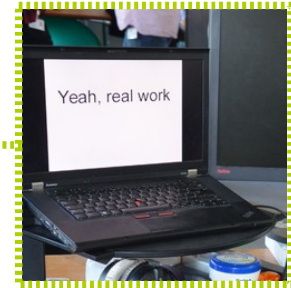
Utilization

- Utilization means that a cpu core is used
- Categories qualify what the core was used for
 - System, IRQ, SoftIRQ
 - Userspace, Guest
 - Idle, IOWait, Steal
 - Nice
- Accounting unit is Jiffy, reports usually as percentage
 - Percentage is better to express relative ratios
- The majority of those basics is usually known, but it still has a purpose
 - Clarify and synchronize the understanding between everybody
 - Provide metaphors that allow to explain it more easily next time

Utilization - Metaphor

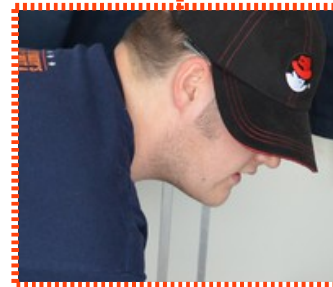
- For all the clarifications on basic terms I will use metaphors
 - Based on well known real world examples
 - At the beginning of a topic the matching metaphor is provided

- Imagine a laptop is a CPU core

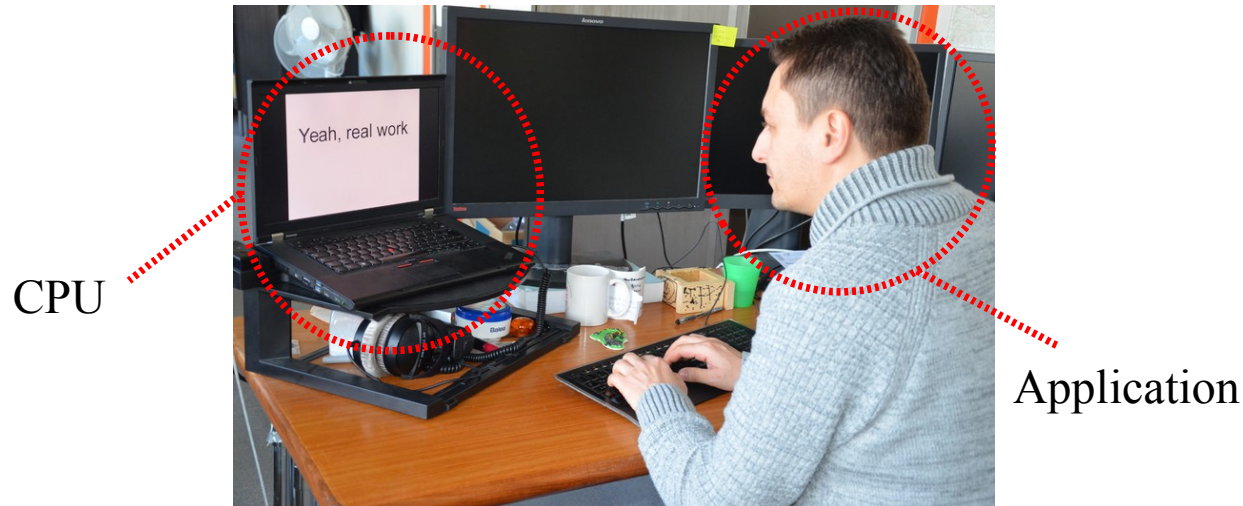


- People using that laptop are different Programs, that could be
 - Application(s)
 - Kernel
 - Hypervisor

**Roles are defined by clothes and equipment
not people (bad actors)*

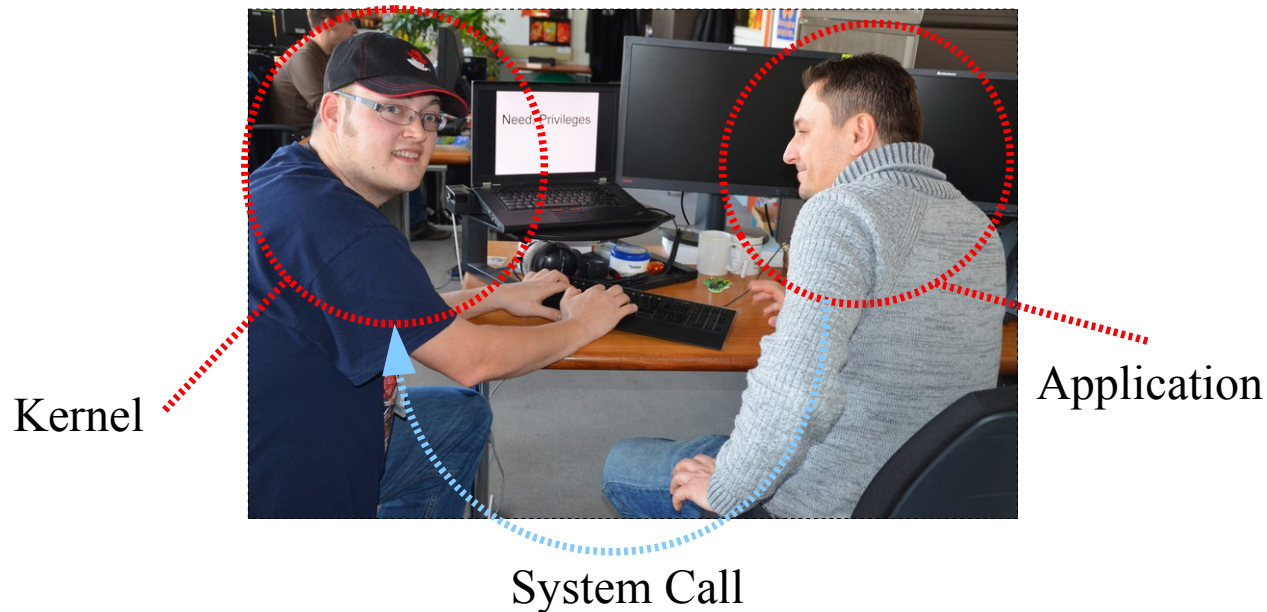


Utilization - USR



- If a userspace application is running it is accounted as **USR**
 - This is usually what you want
 - Also known as problem state, because there your problems are solved
 - If this “application” is actually a virtualized guest it is accounted to Guest instead

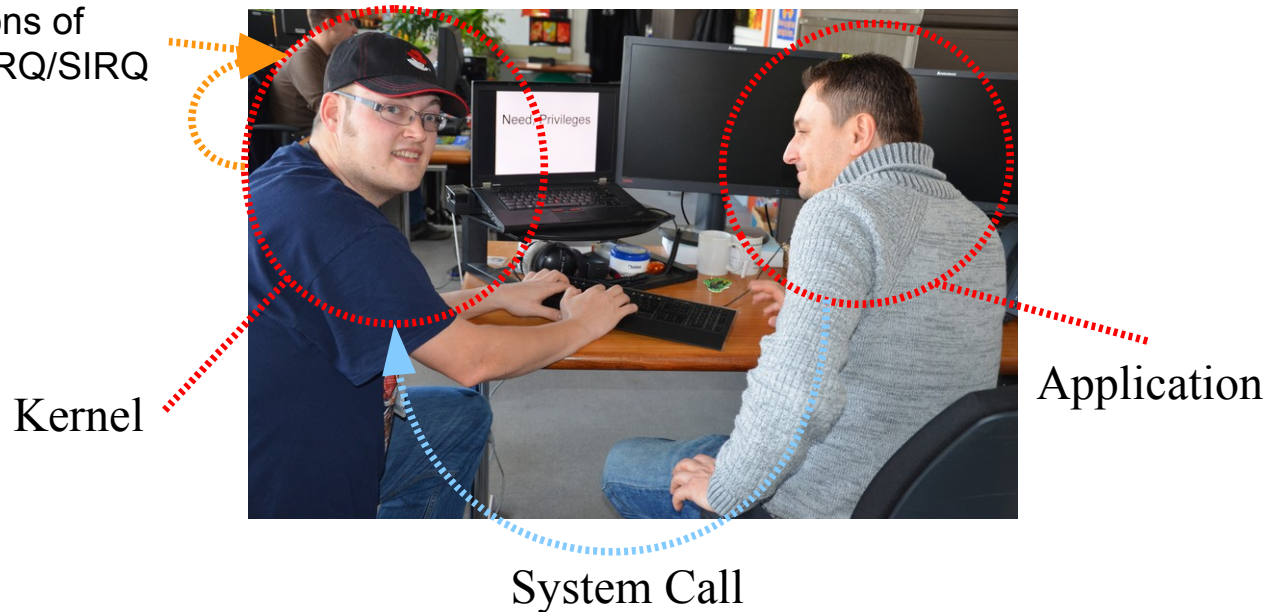
Utilization – SYS, (H)IRQ, SIRQ



- For some tasks you need certain privileges
 - so you call an administrator (System Call to the Kernel)
 - He executes the privileged stuff for you (accounted as **SY**Stem time)

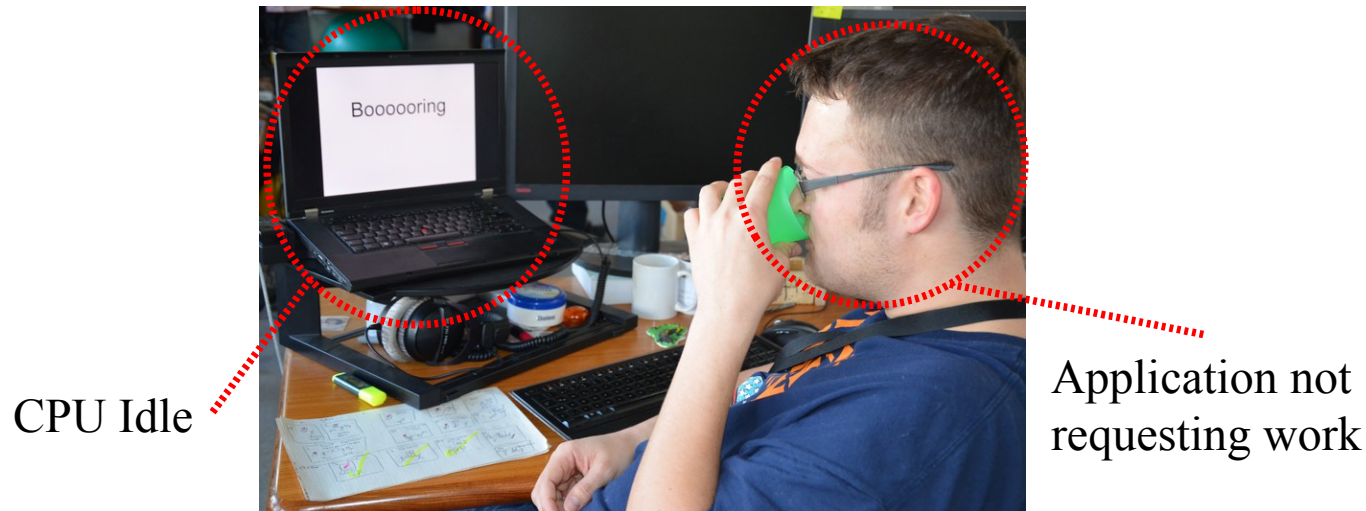
Utilization – SYS, (H)IRQ, SIRQ

Other invocations of
the Kernel → IRQ/SIRQ



- For some tasks you need certain privileges
 - so you call an administrator (System Call to the Kernel)
 - He executes the privileged stuff for you (accounted as **SY**Stem time)
 - There are subcategories
 - **(H)IRQ**: privileged work driven by interrupts (instead of the user)
 - **SIRQ**: privileged work driven by soft interrupts and tasklets (instead of the user)

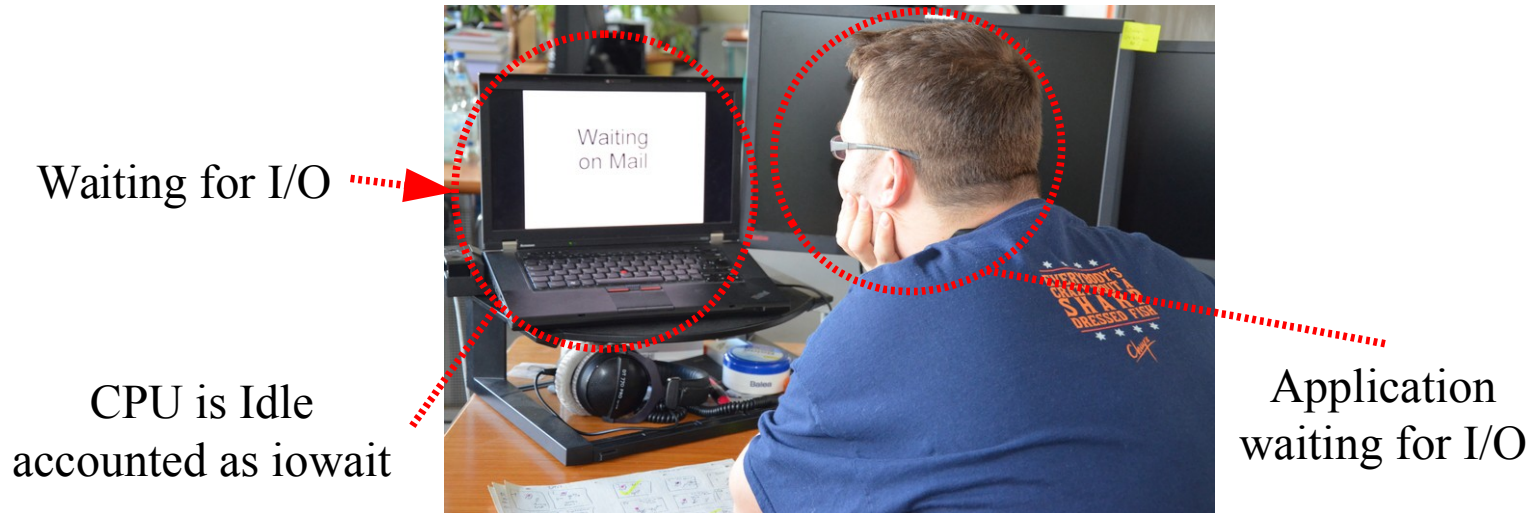
Utilization - Idle



This is the most simple case of “doing nothing”

- The CPU executes nothing because it is not requested to do so
- This is accounted as **idle**

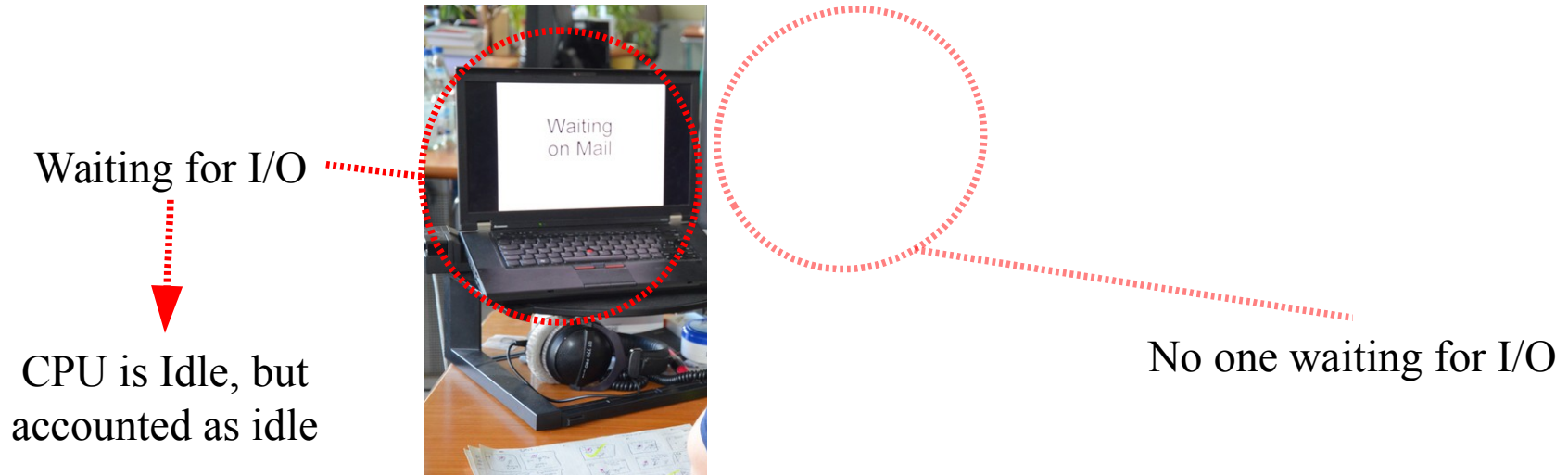
Utilization - IOWait



Again “doing nothing”, but no more that simple

- The System was requested to do some synchronous I/O
 - Still the CPU executes nothing because it is not requested to do so
 - But it knows it “could” do some work if that I/O would complete
- This is accounted as **iowait**

Utilization – IOWait



- If no one is waiting for I/O, (asynchronous)
 - Example 1: real Linux AIO
 - Example 2: writes via page cache
- Not accounted as **await**, but **idle**

Utilization - Steal

Application thought
it could work
(scheduled)

CPU seems to
be non existent

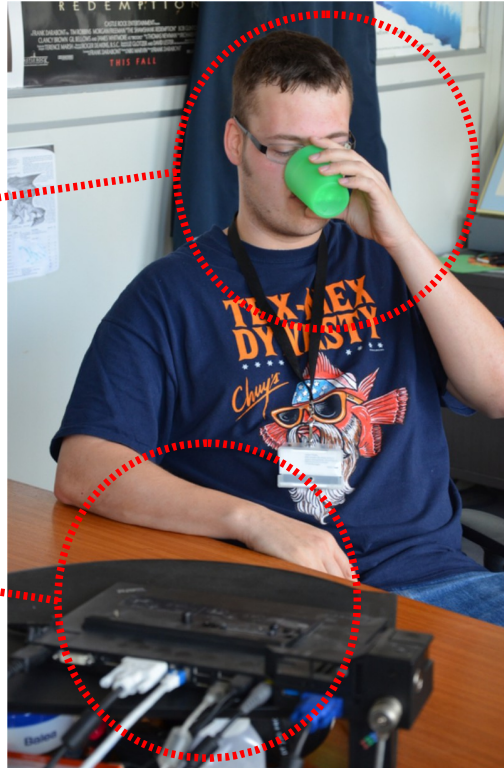
- The CPU is doing something, just not for you
- CPU doesn't exist for you, but you'd need a CPU to realize that
 - Accounted as **steal** time
 - Based on Virtual vs Real timers
- Imagine the laptop is used for multiple groups of people and switched between their docking stations
 - A group of people in front of one laptop causes Context switches (later)



Utilization - Steal

If the Application
doesn't work

Nobody cares about
the CPU being
“non existant”



- In case a CPU shouldn't run anyway the stealing isn't even recognized
- Still accounted as **idle** or **iowait**

Utilization - Steal

Thought it would
transmit network packets
It got from the App

CPU is still busy but
for other things
than Linux thought



Asked the kernel
for I/O handling

- A Linux does not know for which purpose the cpu was stolen
- Steal can indicate issues
 - Too high cpu or memory overcommitment
- But steal also isn't always bad
 - It could be work you requested, just like “USR->SYS in Linux alone”

Utilization – Steal Quiz

- I: Driving I/O synchronously e.g. causing a vswitch to work for your submission
=> Steal?

Utilization – Steal Quiz

- I: Driving I/O synchronously e.g. causing a vswitch to work for your submission
=> Steal?

Yes

Utilization – Steal Quiz

- II: Driving sync reads from a file, mdisk does work for you
=> Steal?

Utilization – Steal Quiz

- II: Driving sync reads from a file, mdisk does work for you
=> Steal?

No, IOWait

Utilization – Steal Quiz

- III: Driving async writes (AIO) by a DB, causing mdisk work in the HV
=> Steal?

Utilization – Steal Quiz

- III: Driving async writes (AIO) by a DB, causing mdisk work in the HV
=> Steal?

Kind of, Idle if idle without steal

Utilization – Steal Quiz

- IV: Paging in the HV takes place while you were running a Java based BI load utilizing all cores
=> Steal?

Utilization – Steal Quiz

- IV: Paging in the HV takes place while you were running a Java based BI load utilizing all cores
=> Steal?

Yes

Utilization – Steal Quiz

- V: Paging in the HV takes place while your system is an idling Dev testbed
=> Steal?

Utilization – Steal Quiz

- V: Paging in the HV takes place while your system is an idling Dev testbed
=> Steal?

No, Idle

Utilization – Steal Quiz summary

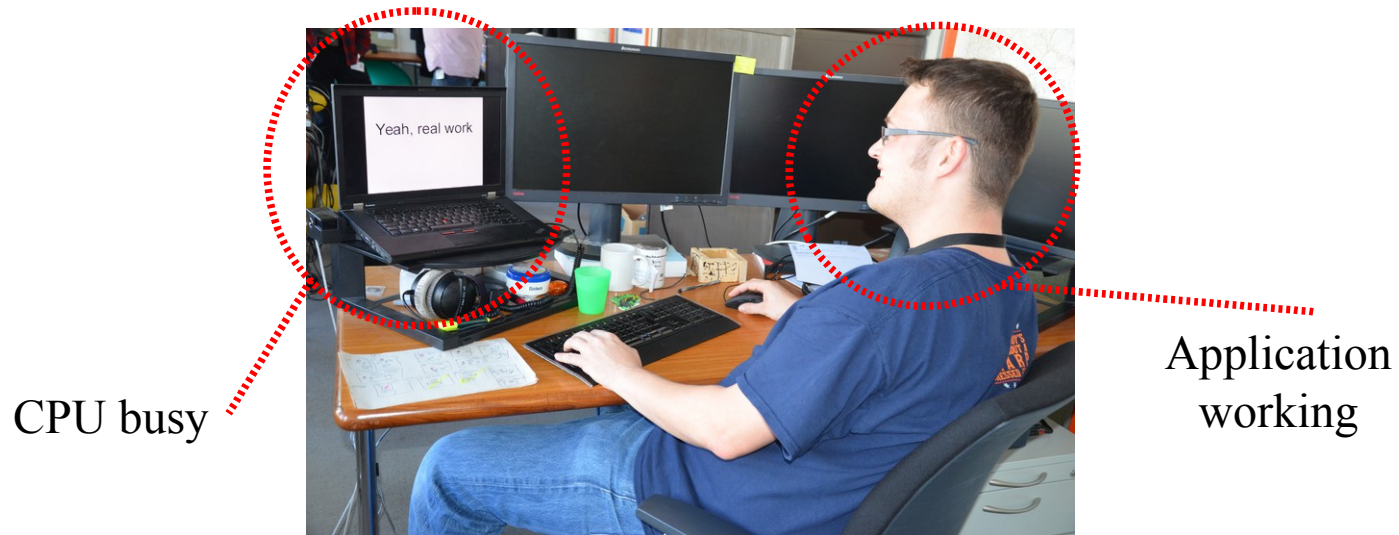
- I: Driving I/O synchronously e.g. causing a vswitch to work for your submission
=> Steal? - **Yes**
- II: Driving sync reads from a file, mdisk does work for you
=> Steal? - **No, IOWait**
- III: Driving async writes (AIO) by a DB, causing mdisk work in the HV
=> Steal? - **Kind of, Idle if idle without steal**
- IV: Paging in the HV takes place while you were running a Java based BI load utilizing all cores
=> Steal? - **Yes**
- V: Paging in the HV takes place while your system is an idling Dev testbed
=> Steal? - **No, Idle**

Scheduling on Cores

- Most systems have more programs than CPUs
- So the OS will have to schedule them (time multiplexing)
- Such scheduling is called Context switching

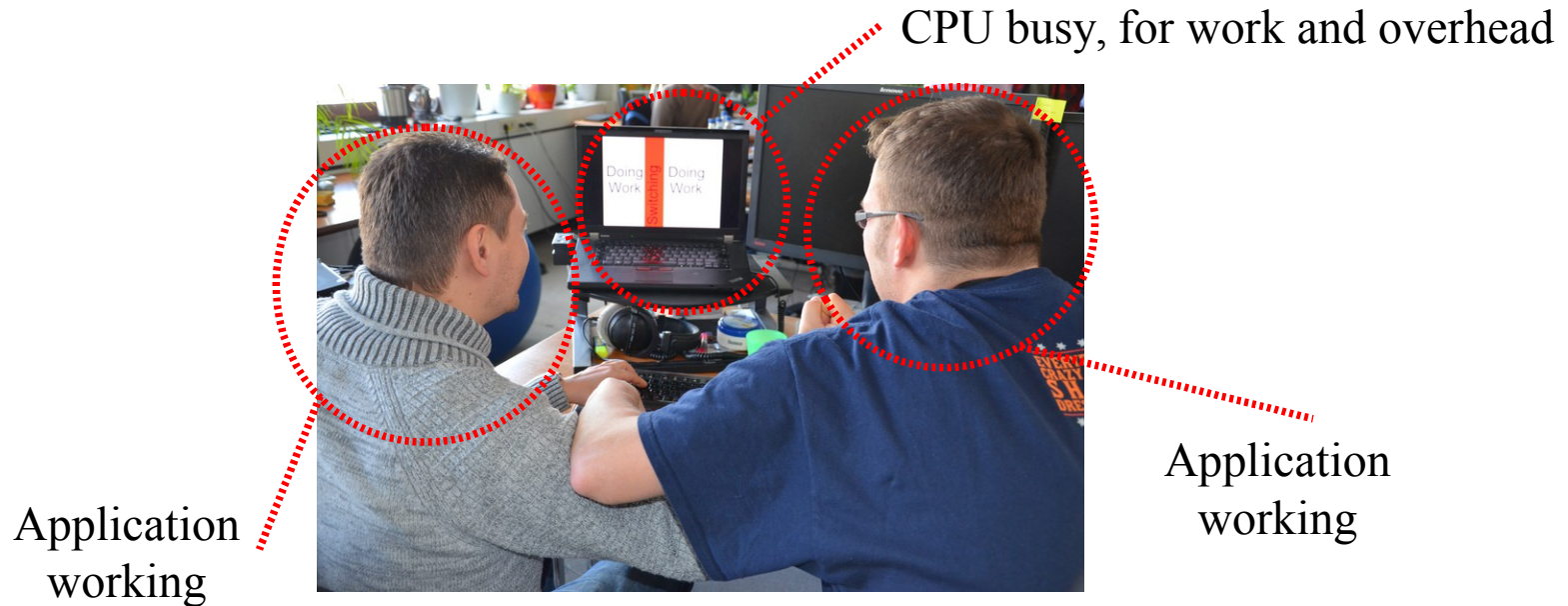
- So in our metaphor we have multiple people of the same privilege level using a laptop ...

Scheduling on Cores



- Well, that is easy a single program means no context switches

Scheduling on Cores



- With two programs the OS has to switch them every now and then
 - Every program shall get its fair share
 - Switching causes overhead
 - Some CPU time is no more used for the actual work done by the programs (red)

Scheduling on Cores

Applications trying
to work

CPU busy, but primarily for overhead

Add steal time if you want real trouble



- With too much runnable programs per CPU the OS gets in trouble
 - Actually this happens with any resource being shared a lot
- Eventually there are two bad options to choose from
 - Real throughput converging to zero (latency optimized)
 - Individual applications have to wait longer (throughput optimized)

Scheduling on Cores



cooperative
Applications

Voluntary
context switch

- There are ways to switch cooperatively
 - On all blocking system calls like reads, timer sleeps
 - On explicit generic or directed context switch
 - local (yield / yield_to)
 - virtual (diagnose X'44' / diagnose X'9C')

Scheduling on Cores



greedy
Applications

OS has to interrupt
and switch
→ non voluntary

- There are ways to switch cooperatively ... or not
 - OS can always interrupt and switch
 - Actually it looks more like one works and one has to wait, but we fight ...

Scheduling between Cores

- Most systems also have more than one CPU
- So the OS will have to schedule/dispatch programs on them
 - most “classic” System z Operating Systems use single queue dispatchers (z/OS, z/VM)
 - Linux has a multi queue scheduler (one queue per CPU)
 - Tasks are migrated to or pulled from cpus
- So in our metaphor we have
 - multiple people and a scheduler
 - in our office being a 4 laptop (CPU core)

Scheduling between Cores

4 Applications
in queue



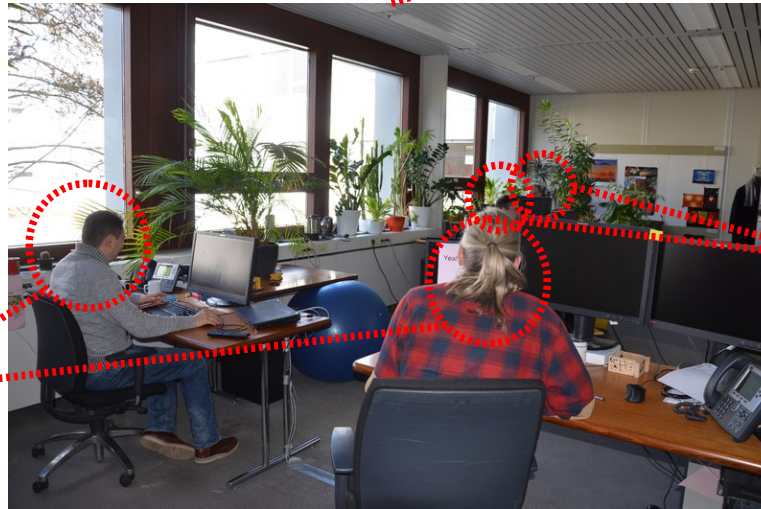
Dispatcher

- Single Queue scheduling
 - One scheduler instance directs programs to the CPU cores
 - Benefit of single control point and easy synchronization

Scheduling between Cores

No need for the scheduler(s) to do anything

Applications
in local runqueue



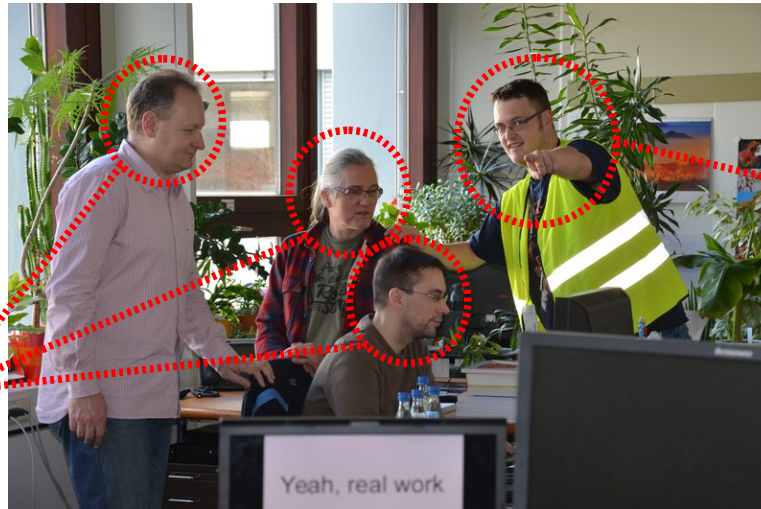
Applications
in local runqueue

■ Multi queue scheduler

- Here with the usual optimum of queues with 1 Program each
- There are cases where it is better to leave one CPU idle
 - When two task are bound by the speed of their communication
- You can also see topology here, as some cores are more “remote” than others

Scheduling between Cores

3 Applications
in local runqueue



Scheduler
migrates task

- Multi queue scheduler
 - Here one queue got rather full and a scheduler starts migrating tasks

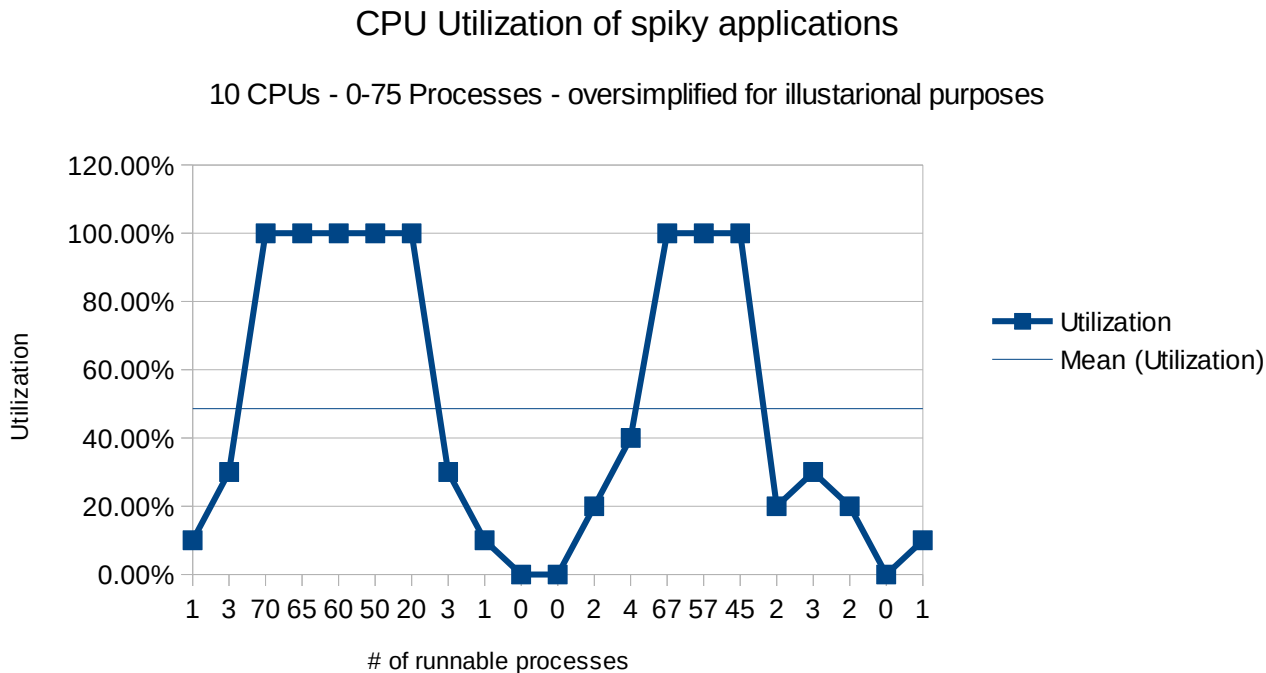
Scheduling / Utilization – Combo Quiz

- Based on a real customer case
 - Database with a lot of stored procedures that does parallelization
 - 0-75 runnable processes varying a lot in a “spiky” fashion
 - They tried various setups, initially 10, later 4 to 20 real CPU cores
 - They didn't achieve their expected target utilization of 85%+ USR

- Question 1: Why was the utilization with 10 CPUs so low at an average of ~45% despite up to 75 runnable processes?

Scheduling / Utilization – Combo Quiz

- Question 1: why was the utilization so low at ~45%?
- Eventually one can never get >100%, but easily less
 - Could the scheduler do anything about it? → No
 - Would it be different with a single queue scheduler? → No

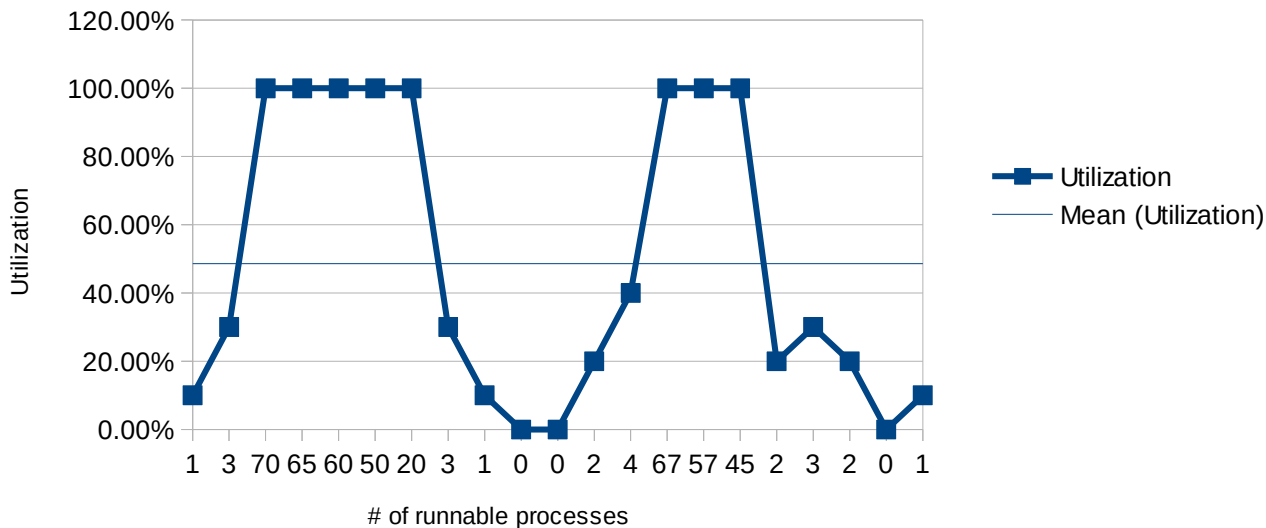


Scheduling / Utilization – Combo Quiz

- Question 2: Why did they achieve the following by changing cpu count?
 - low # of CPUs: fully utilized, but now with a lot of SYS overhead
 - high # of CPUs: even more underutilized but with almost no SYS overhead

CPU Utilization of spiky applications

10 CPUs - 0-75 Processes - oversimplified for illustrational purposes

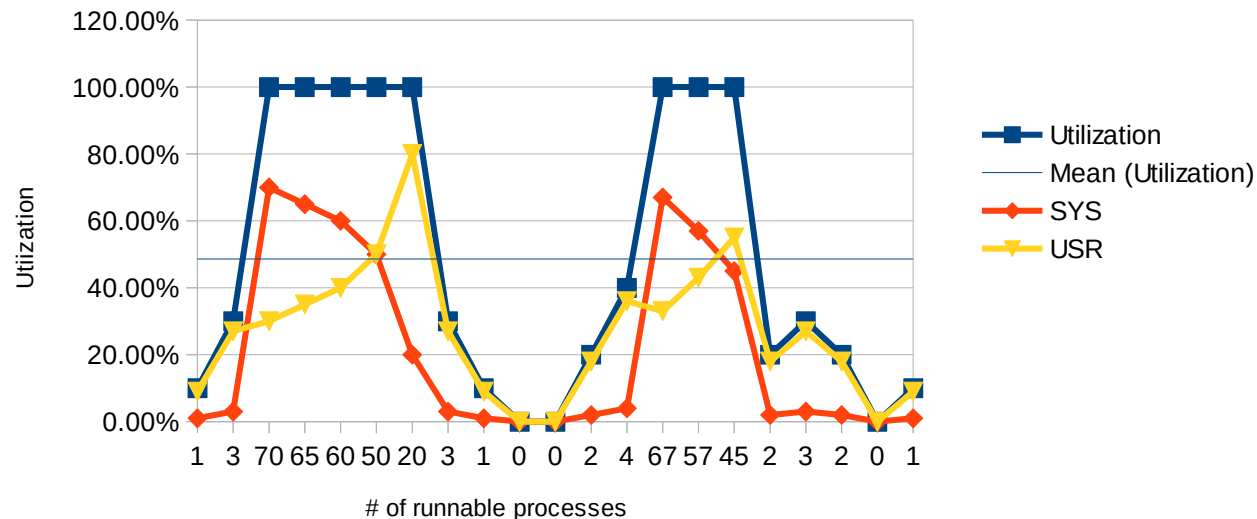


Scheduling / Utilization – Combo Quiz

- Question 2: Why did they achieve the following by changing cpu count?
 - low # of CPUs: fully utilized, but now with a lot of SYS overhead
 - Context switch overhead
 - high # of CPUs: even more underutilized but with almost no SYS overhead
 - Even more times of #runnable << #CPUs

CPU Utilization of spiky applications

10 CPUs - 0.75 Processes - oversimplified for illustrational purposes



Scheduling / Utilization – Combo Quiz

- Based on a real customer case
 - Database with a lot of stored procedures that does parallelization
 - Runnable processes varied a lot in a “spiky” fashion (range 0-75)
 - They tried various setups, initially 10, later 4 to 20 CPU cores
 - They didn't achieve their expected target utilization of 85%+ USR

- Eventually one can never get >100%, but easily less

- Question 3: Real fix approaches?

Scheduling / Utilization – Combo Quiz

- Based on a real customer case
 - Database with a lot of stored procedures that does parallelization
 - Runnable processes varied a lot in a “spiky” fashion (range 0-75)
 - They tried various setups, initially 10, later 4 to 20 CPU cores
 - They didn't achieve their expected target utilization of 85%+ USR
- Eventually one can never get >100%, but easily less
- Question 3: Real fix approaches?
 - **Application design (recommended)**
 - **Try to let the Hypervisor make underutilized resources otherwise usable (needs lower priority workload)**

Page Cache

- Keeping disk data available in memory is caching
 - Certain strategies take place
 - What should be cached
 - Read ahead of data that will likely be used
 - Coalesce writes to issue a single disk write for several memory writes
 - Proper management of caches can be complex (read cpu intensive)

- Imagine processes are kids
 - There is obviously some privileged person (kernel) watching
 - If I learned something they surely will make a mess over time (dirty pages)

 - As long as the kids just watch all their toys (read) things stay clean
 - But when they really play with things (write) the room gets messy (dirty)
 - How are things cleaned up?

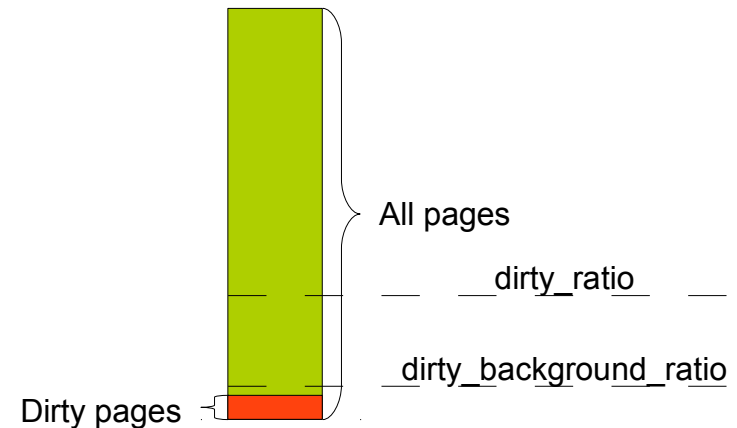
Page Cache – Cleaning I

- If things stay relatively clean nobody cleans anything

Linux tunable: (`% dirty pages < dirty_background_ratio`)

- Only long unused items are put back where they belong

Linux tunable: (`dirty_expire_centisecs`)

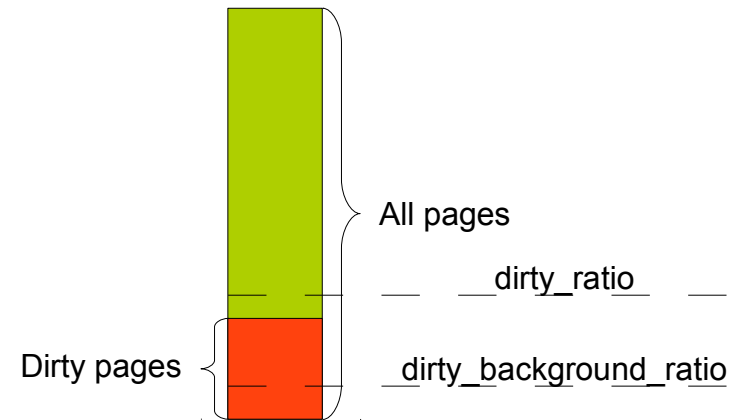


Page Cache – Cleaning II

- As long as the amount of dirtiness is in a sane range it is likely that the parents will clean a bit in background

Linux tunable: (`dirty` pages in % > `dirty_background_ratio`)

- Cleaning of dirty pages consumes CPU, done by the kernel
- Run by the kswap thread(s) and accounted as **SYS**
- Kernel tries to be nice and stay in background
(as you don't bother the kids too much the kernel tries not to take away cpu)

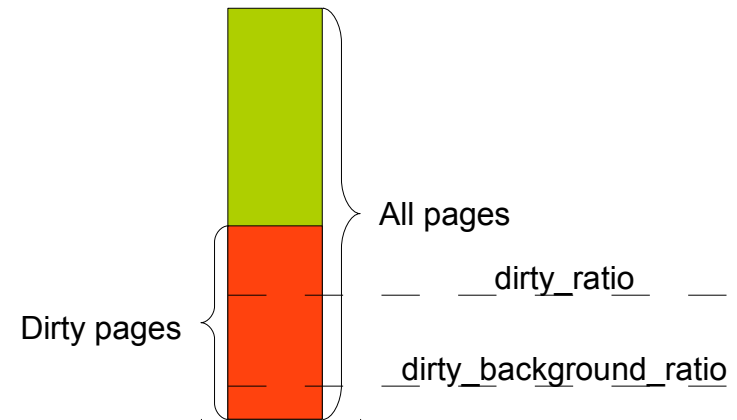


Page Cache – Cleaning III

- If the amount of dirtiness rises too a really high level the parents will force the kids to help cleaning up

Linux tunable: (`dirty pages in % > dirty_ratio`)

- Now writing processes have to contribute parts of their time slices to the kernel
- No more nice, but trying to stall those who make pages dirty



Page Cache – further details

- Another complex topic is the “proper” size of cache
 - Ever realized your flat/house is always too small except for cleaning it
 - If you use the kids room as office from 9am-6pm obviously toys have to be moved aside (cache shrink due to other workload)
 - On the other hand if you organize a party it is likely that they will consume more than just the kids room (cache grow due to more I/O)
- Even cleaning up before going to bed exists in IT
 - for actions like hibernate cache has to be cleaned up before going to sleep

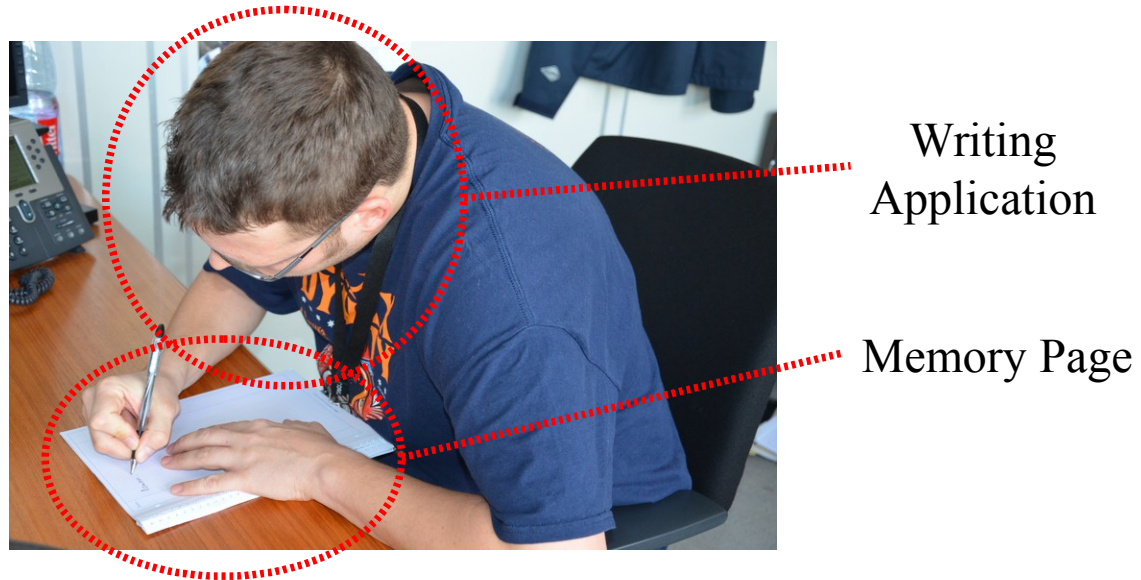


Your system, just like cleaning
Parents sometimes feels like
being in a wasteland

Paging/Swapping

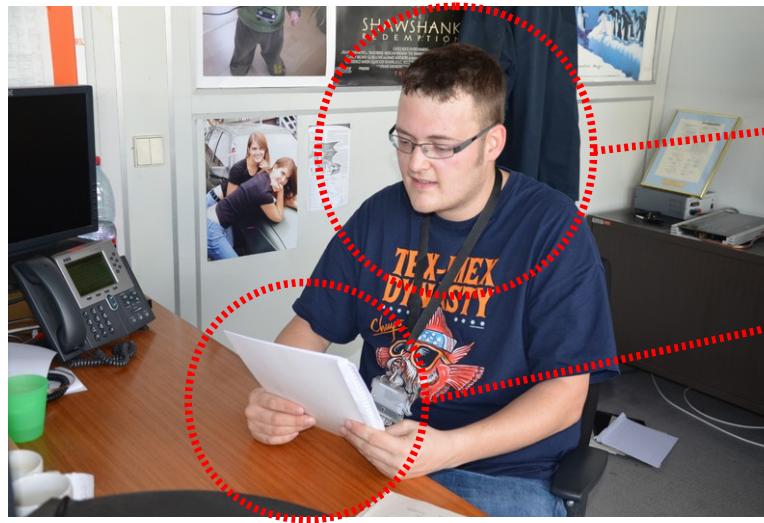
- Spending more memory than available is overcommitment
- In case the accessed memory exceeds the real memory paging has to take place
 - Paging (z/VM) is the same as swapping (Linux)
- As metaphor imagine a notebook page to be your memory ...

Paging/Swapping



- Initially a page is empty, so a process might write onto it

Paging/Swapping

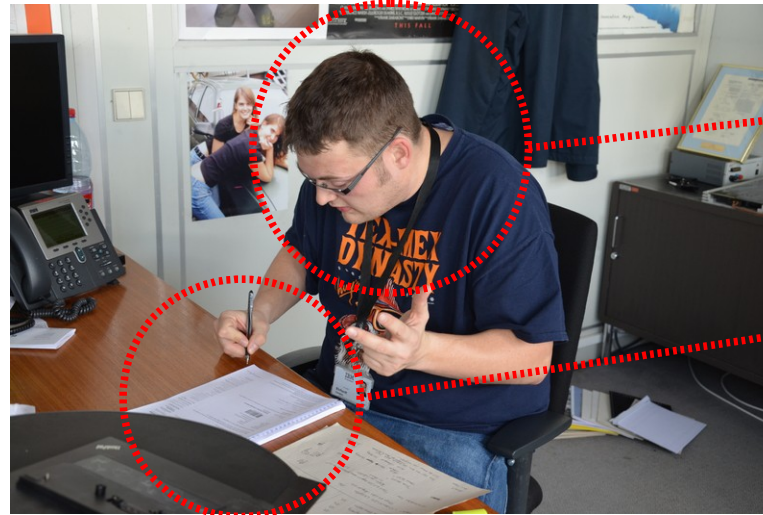


Reading
Application

Memory Page

- Later on the process (or someone else) might read from it

Paging/Swapping



Application wants
to write

Memory has no
free page left

- later the process might have something in mind that he wants to write to the next page
- But all pages the OS could provide are full
 - This now requires swapping (OS level)
 - Or paging (z/VM level)

Paging/Swapping



- The kernel takes over
 - Swaps out a page (based on least recently used plus some extras)
 - This makes room for a clean new page in real memory
 - Impact high, due to the orders of magnitude between disk and memory
- As you see this burden can literally break the back of your system :-)

Top

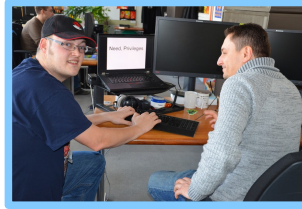
- Characteristics: Easy to use
- Objective: Shows resource usage on process level
- Usage: `top -b -d [interval in sec] > [outfile]`
- Package: RHEL: procps SLES: procps

- 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.)

■ Output



```
top - 11:12:52 up 1: 1, 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:
- physical memory used:
- shared memory

VIRT = SWAP + RES

unit KB

RES = CODE + DATA

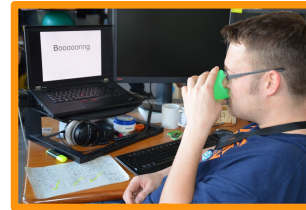
unit KB

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
- Shows
 - IDs: Pid, Tid, User, ...
 - Status: stat and wchan
 - Details: command, memory consumption and accumulated cpu time



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

...

Hints

- Do not specify blanks inside the -o format string
- Status is a one time shot, most interactive or I/O bound processes might sleep



Linux ps command

- cron that should not appear multiple times
 - often long jobs that stack up and make it worse
- Understand what is on there while healthy
- One should be able to name all processes and what they do

```
#> ps axlf
F  UID    PID  PPID  PRI  NI     VSZ   RSS  WCHAN  STAT  TTY          TIME COMMAND
1      0      2      0  20   0      0      0 kthrea  S    ?           0:00 [kthreadd]
1      0      3      2  20   0      0      0 smpboo  S    ?           0:00 \_ [ksoftirqd/0]
5      0      4      2  20   0      0      0 worker  S    ?           0:00 \_ [kworker/0:0]
1      0      5      2   0 -20     0      0 worker  S<   ?           0:00 \_ [kworker/0:0H]
...
4      0      1      0  20   0   9672   5096 Sys_ep  Ss   ?           0:01 /sbin/init
4      0   2023      1  20   0   3584   1400 hrtime  Ss   ?           0:00 /usr/sbin/crond -n
4      0   2024      1  20   0   2840    892 pause   Ss   ?           0:00 /usr/sbin/atd -f
...
4      0   2280      1  20   0  10892   3344 poll_s  Ss   ?           0:00 /usr/sbin/sshd -D
4      0   2316   2280  20   0  14472   4700 poll_s  Ss   ?           0:00 \_ sshd: root@pts/0
4      0   2318   2316  20   0 106392   2216 wait    Ss   pts/0       0:00 \_ -bash
0      0   2350   2318  20   0 105576   1056 -       R+   pts/0       0:00 \_ ps axlf
1      0   2351   2318  20   0 106392    796 sleep_  D+   pts/0       0:00 \_ -bash
...
0      0   2368   2318  20   0   2192    360 -       R    pts/0     110:04 /usr/bin/app
...
```

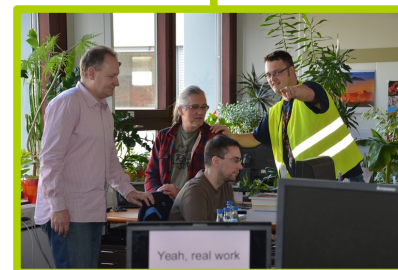
vmstat

- 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:

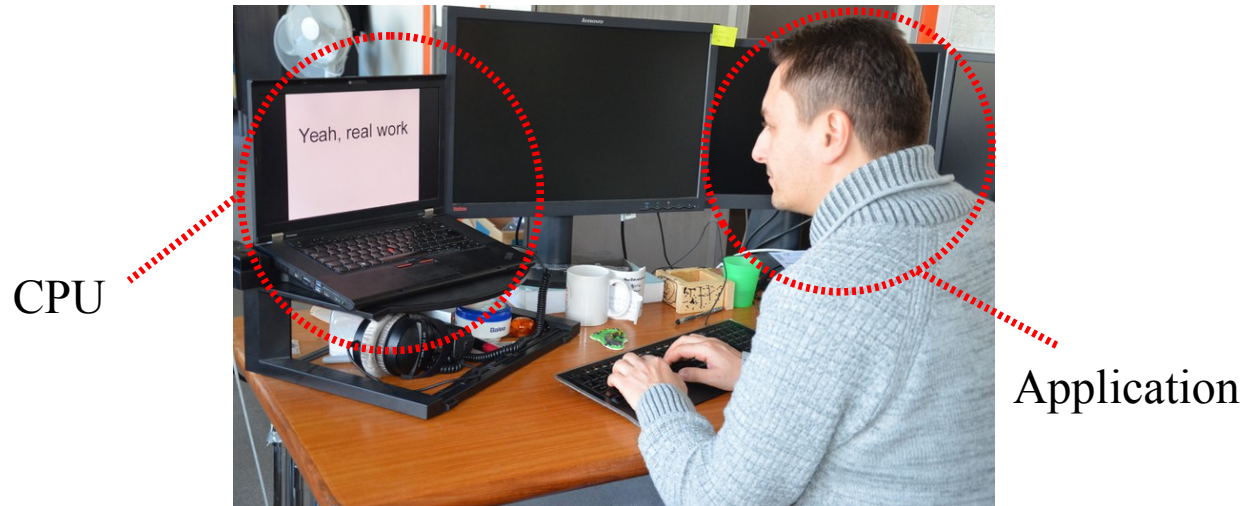
```
vmstat 1
```

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	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'



Utilization - USR



- If a userspace application is running it is accounted as **USR**
 - This is usually what you want
 - Also known as problem state, because there your problems are solved
 - If this “application” is actually a virtualized guest it is accounted to Guest instead

End of Part I

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



Agenda

Basic	Intermediate	Advanced	Master	Elite
– Utilization	– General thoughts	– Strace	– Perf	– Cachestat
– Scheduling		– Ltrace	– slabtop	– Smem
– Page Cache	– Sysstat	– Lsof	– Blktrace	– Valgrind
– Swapping	– Dastat	– Lsluns	– Ziemon	– Iqstats
	– Scsi I/O statistics	– Multipath	– Tcpdump	– Wireshark
– top	– iotop	– hytop	– Java Health Center	– Kernel Tracepoints
– ps	– Lszcrpt	– Dstat	– Java Garbage Collection and Memory visualizer	– Systemtap
– vmstat	– icastats	– Htop		
	– Lsqeth	– Netstat	– Jinsight	
	– Ethtool	– Socket Statistics		
	– Preparation	– Iptraf		

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

- Work with the tools before an issue occurs
 - Learn what should be normal on your system

Formulate a theory

- 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
 - Usually a lot of data is generated that needs to be analysed
 - Monitoring will impact your system

- Identify the most likely problem area
 - CPU related
 - Memory related
 - I/O related
 - Networking related
 - or a combination of several factors

Orientation - where to go if something is broken

Tool	Over-View	CPU cons.	lat.	Hot spots	Disk	Mem	Net
top / ps	X	X	n/a	n/a	n/a	n/a	n/a
sysstat	X	X	n/a	n/a	X	X	n/a
vmstat	X	X	n/a	n/a	n/a	X	n/a
iostat	X	n/a	n/a	n/a	X	n/a	n/a
dasdstat	n/a	n/a	n/a	n/a	X	n/a	n/a
scsistat	n/a	n/a	n/a	n/a	X	n/a	n/a
netstat / ss	X	n/a	n/a	n/a	n/a	n/a	X
htop / dstat / pidstat	X	X	X	n/a	X	n/a	n/a
irqstats	X	X	X	n/a	n/a	n/a	n/a
strace / ltrace	n/a	n/a	X	n/a	n/a	n/a	n/a
hyptop	n/a	X	n/a	n/a	n/a	n/a	n/a
perf	n/a	X	X	X	X	X	X
jinsight	n/a	X	X	n/a	n/a	n/a	n/a
Health Center	X	n/a	n/a	n/a	n/a	n/a	n/a
GMVC	n/a	n/a	X	n/a	n/a	X	n/a
blktrace / ziomon / lsof	n/a	n/a	n/a	n/a	X	n/a	n/a
Valgrind / smem / slabtop	n/a	n/a	n/a	n/a	n/a	X	n/a
iptraf	X	n/a	n/a	n/a	n/a	n/a	X
tracepoints	n/a	n/a	X	X	X	X	X
iotop	n/a	n/a	n/a	n/a	X	n/a	n/a
lszcrypt / icastat	n/a	X	n/a	n/a	n/a	n/a	n/a
lsqeth / ethtool / iftop / iptraf / tcpdump / wireshark	n/a	n/a	n/a	n/a	n/a	n/a	X
lsuns / multipath	n/a	n/a	n/a	n/a	X	n/a	n/a

Sysstat - 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

	proc/s	cswch/s
11:47:57		
11:48:05	7.98	53189.03
11:48:09	8.00	54030.25
11:48:13	8.75	53599.25
11:48:17	7.25	54208.75
11:48:21	1.25	54054.75
11:48:25	8.02	54390.48

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

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 - 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)

11:47:57	CPU	%usr	%sys	%iowait	%steal	%irq	%soft	%guest	%idle
11:48:05	all	70.66	23.47	0.00	0.00	0.12	5.74	0.00	0.00
11:48:05	0	70.32	23.44	0.00	0.00	0.25	5.74	0.00	0.25
11:48:05	1	70.75	23.50	0.00	0.00	0.25	5.50	0.00	0.00
11:48:09	all	71.84	22.53	0.00	0.00	0.00	5.51	0.00	0.13
11:48:09	0	72.00	22.25	0.00	0.25	0.00	5.50	0.00	0.00
11:48:09	1	71.75	22.50	0.00	0.00	0.00	5.50	0.00	0.25
11:48:13	all	71.38	23.12	0.00	0.00	0.00	5.50	0.00	0.00
11:48:13	0	70.75	23.75	0.00	0.00	0.00	5.50	0.00	0.00
11:48:13	1	71.93	22.56	0.00	0.00	0.00	5.51	0.00	0.00

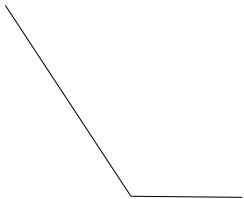
SAR - Network traffic

11:47:57	IFACE	rxpck/s	txpck/s	rxkB/s	txkB/s	rxcmp/s	txcmp/s	rxmcst/s
11:48:05	lo	52136.16	52136.16	45319.64	45319.64	0.00	0.00	0.00
11:48:05	eth1	0.00	0.00	0.00	0.00	0.00	0.00	0.00
11:48:05	eth0	0.00	0.00	0.00	0.00	0.00	0.00	0.00
11:48:09	lo	53254.25	53254.25	46308.95	46308.95	0.00	0.00	0.00
11:48:09	eth1	0.25	0.00	0.08	0.00	0.00	0.00	0.25
11:48:09	eth0	0.25	0.00	0.08	0.00	0.00	0.00	0.25

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

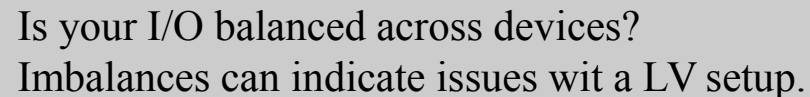
11:47:57	tps	rtps	wtps	bread/s	bwrtn/s
11:48:01	1023.06	1020.30	2.76	35689.22	92.23
11:48:05	2588.53	2587.03	1.50	89208.98	31.92
11:48:09	491.00	425.50	65.50	15352.00	880.00
11:48:13	278.50	276.00	2.50	9016.00	44.00
11:48:17	135.25	133.00	2.25	4560.00	42.00
11:48:21	698.75	152.00	546.75	5376.00	4566.00



Overview of
- operations per second
- transferred amount

SAR – Disk I/O II – per device

11:47:57	DEV	tps	rd_sec/s	wr_sec/s	avgrq-sz	avgqu-sz	await	svctm	%util
11:48:05	dev253-0	165.34	5564.09	0.00	33.65	0.06	0.38	0.21	3.49
11:48:05	dev253-1	170.32	5564.09	0.00	32.67	0.06	0.37	0.20	3.49
11:48:05	dev8-128	18.95	576.56	0.00	30.42	0.01	0.39	0.39	0.75
11:48:05	dev8-144	19.20	674.31	0.00	35.12	0.00	0.26	0.26	0.50
11:48:05	dev8-160	19.20	724.19	0.00	37.71	0.01	0.39	0.39	0.75
11:48:05	dev8-176	18.95	652.37	0.00	34.42	0.01	0.53	0.53	1.00



Is your I/O balanced across devices?
Imbalances can indicate issues with 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

11:47:57	kbmemfree	kbmemused	%memused	kbbuffers	kbcached	kbcommit	%commit
11:48:01	41324	8211980	99.50	16992	7776084	145764	0.94
11:48:05	40844	8212460	99.51	17008	7776388	145964	0.94
11:48:09	40816	8212488	99.51	17036	7776800	145964	0.94
11:48:13	41088	8212216	99.50	17052	7777028	145968	0.94
11:48:17	42592	8210712	99.48	17072	7775396	146100	0.9

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 (swpin/-out) all the time is bad.

SAR - Memory pressure - Swap

11:47:57	pswpin/s	pswpout/s
11:48:01	2853.95	2658.26
11:48:05	2003.26	5399.80
11:48:09	88.59	9921.92
11:48:13	3199.30	53.15

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

11:47:57	pgpgin/s	pgpgout/s	fault/s	majflt/s	pgfree/s	pgscank/s	pgscand/s	pgsteal/s	%vmeff
11:48:01	4799.00	46.12	55571.93	1.00	12821.05	1859.40	0.00	1859.40	100.00
11:48:05	11153.12	15.96	22226.68	0.00	13414.46	2767.08	0.00	2767.08	100.00
11:48:09	1919.00	125.00	825.00	0.00	11345.75	461.75	0.00	461.75	100.00
11:48:13	1127.00	22.00	598.50	0.00	11227.00	262.25	0.00	262.25	100.00
11:48:17	570.00	21.00	52.75	0.00	11184.50	254.25	0.00	254.25	100.00

Don't always 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

vmefficiency is the ratio of scans vs reused (stolen) pages

SAR - System Load

11:47:57	runq-sz	plist-sz	ldavg-1	ldavg-5	ldavg-15
11:48:01	17	172	0.72	0.35	0.14
11:48:05	15	172	1.95	0.61	0.23
11:48:09	15	172	2.91	0.84	0.30
11:48:13	5	172	3.88	1.07	0.38
11:48:17	15	172	4.21	1.19	0.42
11:48:21	17	171	4.21	1.19	0.42

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.

Sysstat - 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

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

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

Recent versions are improved by reporting reads/writes separately which is great as they have vastly different characteristics

Sysstat - PIDSTAT

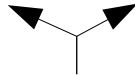
- Characteristics: Easy to use extended per process statistics
- Objective: Identify processes with peak activity
- Usage: `pidstat [-w | -r | -d]`
- Package: RHEL: `sysstat` SLES: `sysstat`

- 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

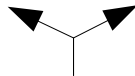
Pidstat examples

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



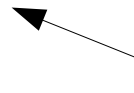
cswch/s means voluntary context switch; nvcswch non-voluntary

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



minflt & majflt indicate 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



kB_rd & kb_wr represent disk I/O per process

Sysstat - mpstat

- Characteristics: Show statistics per processor
- Objective: Identify imbalanced utilization or interrupt peaks
- Usage: `mpstat -A <interval>`
- Package: RHEL: sysstat SLES: sysstat

- Shows
 - `-u` utilization
 - `-I <CPU | SCPU | ALL>` Interrupts

- Hints
 - Can be restricted to selected processor(s) (`-P`)

Sysstat – mpstat example

- As one can see there are plenty of different (s)irq sources these days
 - Ordered horizontally per type and vertically per cpu

IRQs

10:40:12	CPU	EXT/s	I/O/s	AIO/s	CLK/s	EXC/s	EMS/s	TMR/s	TAL/s
PFL/s	DSD/s	VRT/s	SCP/s	IUC/s	CMS/s	CMC/s	CMR/s	CIO/s	QAI/s
DAS/s	C15/s	C70/s	TAP/s	VMR/s	LCS/s	CLW/s	CTC/s	APB/s	ADM/s
CSC/s	PCI/s	MSI/s	VIR/s	VAI/s	NMI/s	RST/s			
10:40:17	0	2.40	0.00	0.00	2.00	0.20	0.20	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10:40:17	1	1.40	0.00	0.00	1.40	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00									
[...]									

CPU: 0, 1
 IRQ type: EXT, I/O, AIO, CLK, EXC, EMS, TMR, TAL
 Rate per second: 2.40, 1.40

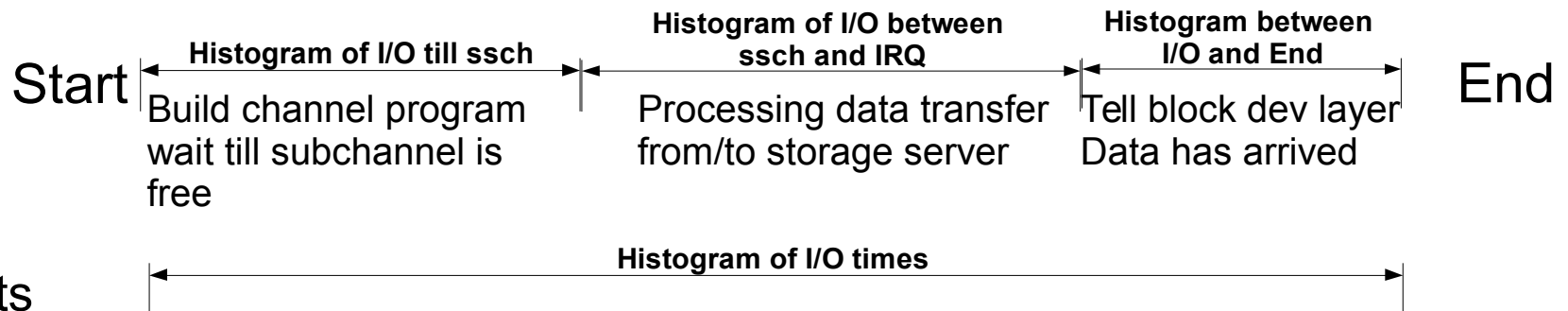
SoftIRQs

10:40:26	CPU	HI/s	TIMER/s	NET_TX/s	NET_RX/s	BLOCK/s	BLOCK_IOPOLL/s	TASKLET/s	SCHED/s
HRTIMER/s	RCU/s								
10:40:31	0	0.00	0.60	0.00	0.00	0.00	0.00	0.00	1.00
0.00	0.60								
10:40:31	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00								
[...]									

CPU: 0, 1
 Rate per second: 0.60, 0.00

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`
- Package: n/a for kernel interface, s390-tools for dasdstat
- Shows:



- Hints

- The tools dasdstat is available in s390-utils to handle all that in one place

DASD statistics – report

Sample:

8*512b = 4KB **<= request size < 16*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 ← look here for subchannel busy

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 ← look here for slow SAN

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 chang at enqueueing (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

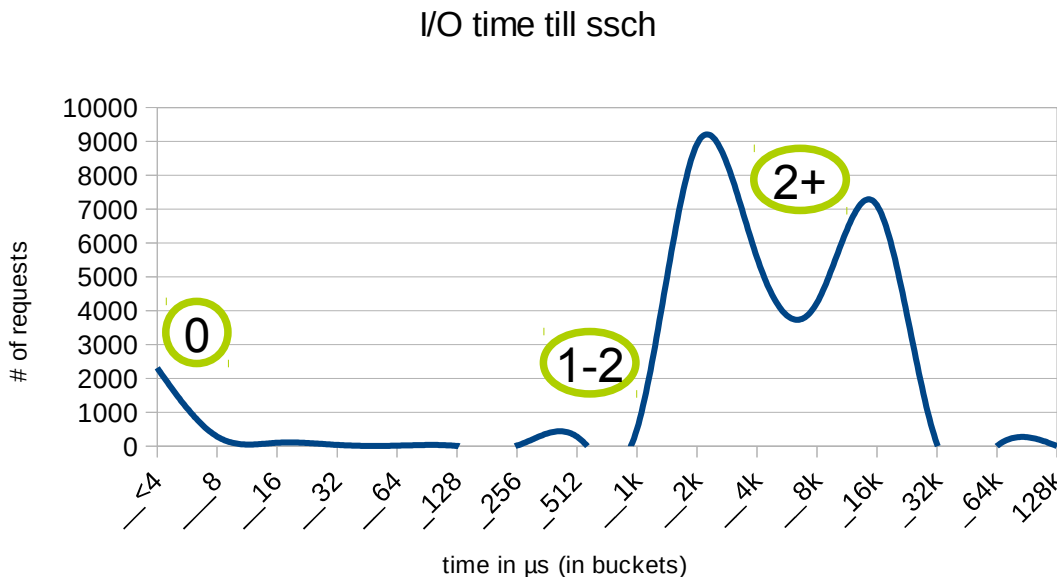
– Also shows data per sector which usually is only confusing

DASD statistics – look for subchannel busy issues

<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
[...]															
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

Time consists of

- “Build subchannel program (usually very fast)
- wait for a free subchannel (can be long without or too few HPAV)
- Please be aware that all columns in dasdstat are scaled by 2^n



Number of requests before the current one

FCP statistics

- Characteristics: Detailed latency information for FCP I/O
- Objective: Collect details of I/O operations on FCP devices
- Package: n/a (Kernel interface)

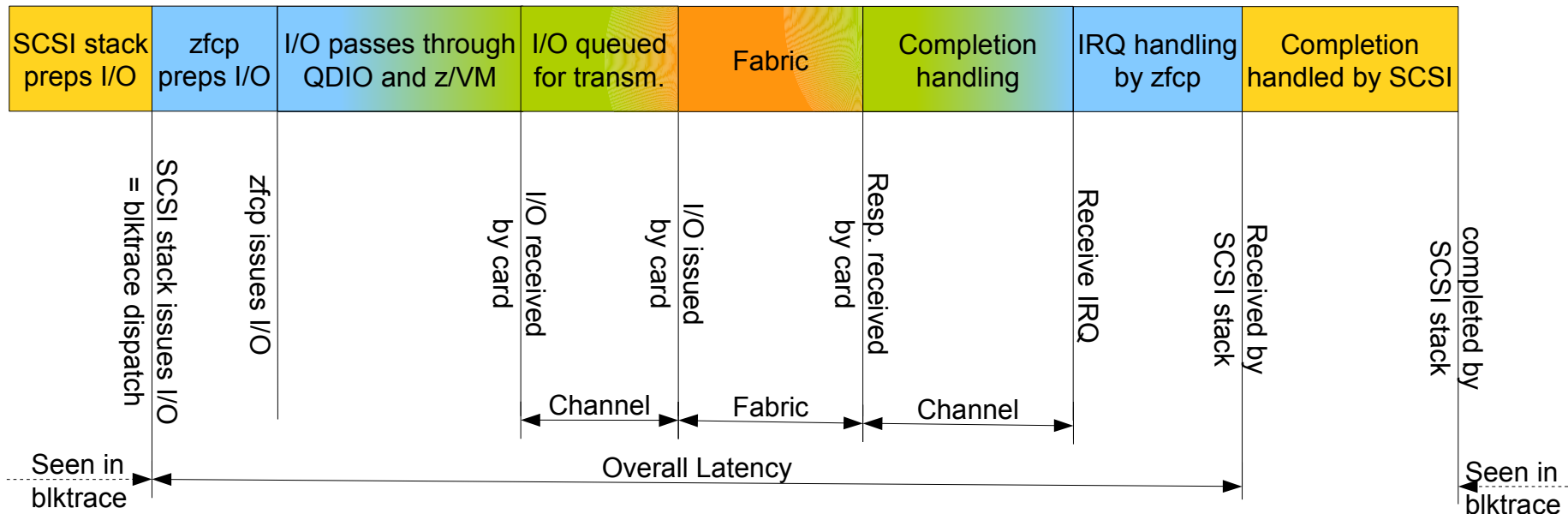
- Usage:
 - enable
 - CONFIG_STATISTICS=y must be set in the kernel config file
 - debugfs is mounted at /sys/kernel/debug/
 - For a certain LUN in directory
/sys/kernel/debug/statistics/zfcpx-<device-bus-id>-<WWPN>-<LUN>
issue `echo on=1 > definition` (turn off with `on=0`, reset with `data=reset`)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

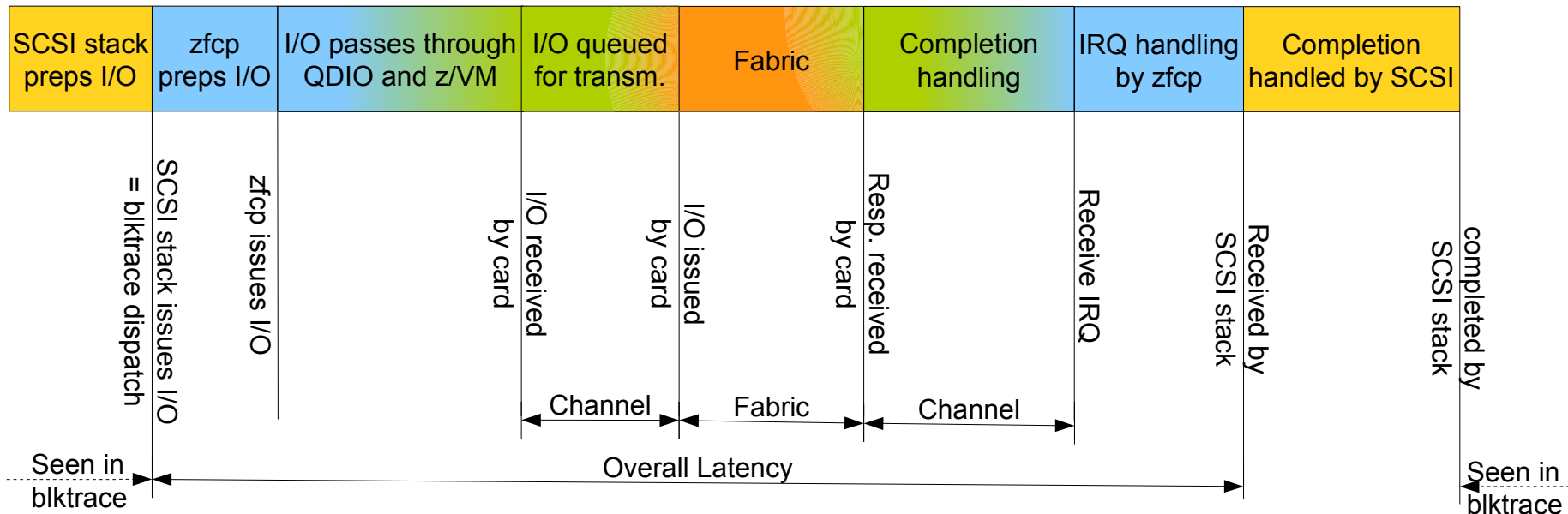
■ Shows:

- Request sizes in bytes (hexadecimal)
- Channel latency in ns Time spent on the FCP channel (internal transfer)
- Fabric latency in ns Time spent in the FCP fabric (outside transfer)
- (Overall) latencies whole time spent entry/exit of the zFCP layer in ms



FCP statistics

- On popular request – the “where to complain” guide
 - App to zfcpx driver: Linux developers / Distributor in general
 - zfcpx driver to hardware: zfcpx/qdio (optional also z/VM) developers
 - In system hardware to SAN: FCP card HW/FW stack
 - External: SAN administrator



FCP statistics example – rather unreadable

```
cat /sys/kernel/debug/statistics/zfc-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
...
```

← means: request size 4KB, 1163 occurrences

← means: response time <= 1ms

← means: Channel response time <= 32μs
= all below driver

← means: Fabric response time <= 1ms
= once leaving the card

FCP statistics example

- Some statistics are per device
- Some per adapter
 - Adapter statistics can not be reseted
- Example beautifying the same data with a little script (not public yet)

per device latency statistics (f - fabric; c - channel)

disk	rd-fmin	rd-fmax	rd-fsum	rd-favg	rd-cmin	rd-cmax	rd-csum	rd-cavg	rd-cnt	wr-* [..wr/cmd]
sda	92	130	1386	126.00	7	10	98	8.91	11	
sdb	127	131	2072	129.50	7	10	140	8.75	16	
sdc	126	140	2075	129.69	7	14	145	9.06	16	
sdd	126	132	1160	128.89	7	8	74	8.22	9	
[...]										
sdbd	n/a	n/a	0	0.00	n/a	n/a	0	0.00	0	
sdbe	n/a	n/a	0	0.00	n/a	n/a	0	0.00	0	

per adapter statistics

adapter (subch/dev)	rd-cnt	rd-mb	rd-avgsz	wr-cnt	wr-mb	wr-avgsz	cmd-cnt	sec-active
0.0.0004/0.0.1700	4899	18	3.76	11572	1249	110.52	240	17324
0.0.000c/0.0.1800	4901	16	3.34	11564	1265	112.02	236	17325
0.0.00d6/0.0.5100	4765	16	3.44	11595	1254	110.75	239	17318
0.0.00e2/0.0.5b00	1888	5	2.71	0	0	0.00	160	17309

iostat

- Characteristics: simple, top like I/O monitor
- Objective: Check which processes are doing I/O
- Usage: `iostat`
- Package: RHEL: `iostat` SLES: `iostat`

- Shows
 - Read/Write per thread
 - Can accumulate (-a) for updating summaries instead of live views
 - Useful for Disk I/O tests that don't account on their own
 - Separate accounting for swap

- Hints
 - Can be restricted to certain processes via (-p)
 - Has a batch mode like top

iotop - examples

■ Example I: disk I/O can be spread other than expected

System wide values:
total: app<->kernel
actual: kernel<->disk

Total DISK READ : 34.05 M/s | Total DISK WRITE : 45.20 B/s
Actual DISK READ: 24.02 B/s | Actual DISK WRITE: 31.19 M/s

TID	PRIO	USER	DISK READ	DISK WRITE	SWAPIN	IO>	COMMAND
7204	be/4	gemu	378.16 K/s	0.00 B/s	0.00 %	6.16 %	gemu-system-s390x
7231	be/4	gemu	350.87 K/s	0.00 B/s	0.00 %	6.08 %	gemu-system-s390x
7225	be/4	gemu	343.08 K/s	19.49 K/s	0.00 %	6.00 %	gemu-system-s390x
7174	be/4	gemu	346.57 K/s	0.00 B/s	0.00 %	6.00 %	gemu-system-s390x

Disk Read / Write
values are per process

■ Example II: even interesting for memory bound (overcommitted) loads

Total DISK READ : 971.07 M/s | Total DISK WRITE : 180.50 M/s
Actual DISK READ: 850.18 M/s | Actual DISK WRITE: 150.23 M/s

TID	PRIO	USER	DISK READ	DISK WRITE	SWAPIN	IO>	COMMAND
7204	be/4	gemu	7732.35 K/s	0.00 B/s	35.54 %	2.99 %	gemu-system-s390x
7231	be/4	gemu	4057.11 K/s	0.00 B/s	23.10 %	2.98 %	gemu-system-s390x
7225	be/4	gemu	6496.12 K/s	0.00 K/s	28.59 %	2.86 %	gemu-system-s390x
7174	be/4	gemu	6563.95 K/s	0.00 B/s	42.70 %	2.76 %	gemu-system-s390x

Swapin % of time
while swapping in

IO % means time
waiting for I/O

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 you won't see it in icastat)

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”

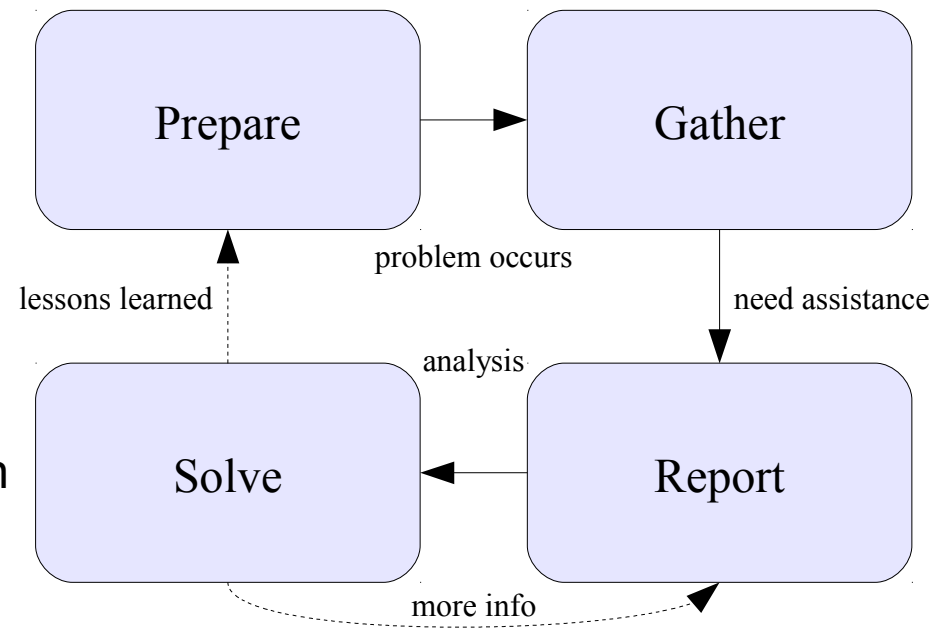
```

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]
[...]
[...]
```

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

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



End of Part II

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



Agenda

Basic	Intermediate	Advanced	Master	Elite
– Utilization	– General thoughts	– Strace	– Perf	– Cachestat
– Scheduling		– Ltrace	– slabtop	– Smem
– Page Cache	– Sysstat	– Lsof	– Blktrace	– Valgrind
– Swapping	– Dastat	– Lsluns	– Ziemon	– Iqstats
	– Scsi I/O statistics	– Multipath	– Tcpdump	– Wireshark
– top	– iotop	– hytop	– Java Health Center	– Kernel Tracepoints
– ps	– Lszcrpt	– Dstat	– Java Garbage Collection and Memory visualizer	– Systemtap
– vmstat	– icastats	– Htop		
	– Lsqeth	– Netstat	– Jinsight	
	– Ethtool	– Socket Statistics		
	– Preparation	– Iptraf		

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

% time reports shares, useful to rate the importance of a call

The columns “seconds”, “usecs/call”, “calls”, “errors” can reveal if one has a lot, slow or failing calls?

syscall references are available as “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

% time reports shares, useful to rate the importance of a call

The columns “seconds”, “usecs/call”, “calls”, “errors” can reveal if one has a lot, slow or failing calls?

Most function references are available as “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      strcmp
  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\0\0\17\0\0\0\0\0\0\0"..., 680) = 680 <0.000007>
0.000027 write(3, "\0\0\0\0\0\0\0\0\17\0\0\0\0\0\0\0"..., 680) = 680 <0.000007>
0.000026 fdatsync(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>
```

Isof

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

- 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

Isof - 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, ...

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: “mutliphath -ll”
- Package: RHEL: device-mapper-multipath SLES: mutliphath-tools

```
multipath -ll
swap-3of6 (36005076309ffc7c100000000000002022) 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 sda 65:208 active ready running
[...]
```

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

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 example

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

memuse = resident

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
VMSEVRU	1	0.00	0:00	0:03:10	0.01	0.03	1500
VMSEVRP	1	0.00	0:00	0:03:10	0.01	0.06	1500
VMSEVRV	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

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 -vtin`
- 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

```

-----system-----  ---procs---  -----memory-usage-----  ---paging--  -dsk/total-
      time      |run blk new|  used  buff  cach  free|  in  out  |  read  writ
04-02 10:52:50|  0    0 1.1| 291M 81.1M 160M 9382M|  0    0  |   93k 8591B
04-02 10:53:20|0.1    0 0.3|9771M 400k 60.3M 83.5M| 637M 565M| 303k   29k
04-02 10:53:50|  0    0   0|9785M 376k 60.2M 69.5M|1162M 166M| 229k 2458B
04-02 10:54:20|  0    0   0|9786M 528k 60.7M 68.0M| 48k   0  |   50k 1092B
04-02 10:54:50|0.1    0   0|9790M 428k 59.7M 64.5M|1102M 895k|   20k 1911B

```

similar to vmstat

new in live tool

similar to iostat
(also per device)

```

-----total-cpu-usage-----  inter  ---system--  -----virtual-memory-----
usr sys idl wai hiq siq|  1  |  int   csw  |majpf minpf alloc  free
  0   0 100   0   0   0|  9  | 145   279  |  0   307   195   689
  1   2  97   0   0   0|  4  | 12k 1119  | 22k 150k 308k 229k
  1   3  96   0   0   0|  3  | 892   960  | 69k 254k 304k 304k
  0   0 100   0   0   0|  2  | 309   619  |  2    8    20    1
  1   3  96   0   0   0|  1  | 594   875  | 79k 258k 282k 282k

```

many
many
more ...

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 example

Configurable utilization visualization

```

0  [||||| 13.1%]
1  [||||| 14.5%]
2  [|| 1.9%]
3  [||||| 13.6%]
4  [  0.0%]
5  [||||| 14.1%]
Mem[||||| 945/12059MB]
Swp[  0/0MB]
  
```

Tasks: 101, 80 thr; 60 running
 Load average: 42.03 16.67 6.24
 Uptime: 00:17:11

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:
51970	postgres	20	0	3264M	138M	136M	S	1.0	1.1	-	-	0:00.38	0:00.21	187	0	0:00.50	postgres:

Common process info

Accumulated Usage
and IO rates

Hierarchy

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

IPTRAF

- Characteristics: Live information on network devices / connections
- Objective: Filter and format network statistics
- Usage: `iptraf`
- Package: RHEL: `iptraf` / `iptraf-ng` 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
 - There are various other tools: `iftop`, `bmon`, ...
 - like with net benchmarks no one seem to fit all

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)

Packet
sizes

IF
details

```
IPtraf
Interface: eth0
Packet size brackets for interface eth0

+-----+-----+-----+-----+
| Range (bytes) | Count | Range (bytes) | Count |
+-----+-----+-----+-----+
| 1 to 75:      | 2274  | 751 to 825:   | 2     |
| 76 to 150:    | 37    | 826 to 900:   | 0     |
| 151 to 225:   | 25    | 901 to 975:   | 3     |
| 226 to 300:   | 84    | 976 to 1050:  | 1     |
| 301 to 375:   | 10    | 1051 to 1125: | 6     |
| 376 to 450:   | 27    | 1126 to 1200: | 1     |
| 451 to 525:   | 16    | 1201 to 1275: | 2     |
| 526 to 600:   | 38    | 1276 to 1350: | 5     |
| 601 to 675:   | 5     | 1351 to 1425: | 2864  |
| 676 to 750:   | 4     | 1426 to 1500+:| 7     |
+-----+-----+-----+-----+

Interface MTU is 1500 bytes, not counting the data-link header
Maximum packet size is the MTU plus the data-link header length
Packet size computations include data-link headers, if any
```

```
IPtraf
Interface: eth0

+-----+-----+-----+-----+-----+-----+
| Total:  | 44    | 11808 | 30    | 9101  | 14    | 1988 |
| IP:     | 44    | 10472 | 30    | 8681  | 14    | 1792 |
| TCP:    | 19    | 4120  | 9     | 3483  | 10    | 637  |
| UDP:    | 25    | 6353  | 21    | 5198  | 4     | 1155 |
| ICMP:   | 0     | 0     | 0     | 0     | 0     | 0     |
| Other IP:| 0     | 0     | 0     | 0     | 0     | 0     |
| Non-IP: | 0     | 0     | 0     | 0     | 0     | 0     |
+-----+-----+-----+-----+-----+-----+

Total rates:      1.0 kbits/sec      Broadcast packets: 21
                  1.2 packets/sec    Broadcast bytes: 5492

Incoming rates:   0.7 kbits/sec
                  0.6 packets/sec

Outgoing rates:   0.3 kbits/sec
                  0.6 packets/sec

IP checksum errors: 0
```

End of Part III

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



Agenda

Basic	Intermediate	Advanced	Master	Elite
– Utilization	– General thoughts	– Strace	– Perf	– Cachestat
– Scheduling		– Ltrace	– slabtop	– Smem
– Page Cache	– Sysstat	– Lsof	– Blktrace	– Valgrind
– Swapping	– Dastat	– Lsluns	– Ziemon	– Iqstats
	– Scsi I/O statistics	– Multipath	– Tcpdump	– Wireshark
– top	– iotop	– hytop	– Java Health Center	– Kernel Tracepoints
– ps	– Lszcrpt	– Dstat	– Java Garbage Collection and Memory visualizer	– Systemtap
– vmstat	– icastats	– Htop		
	– Lsqeth	– Netstat	– Jinsight	
	– Ethtool	– Socket Statistics		
	– Preparation	– Iptraf		

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
 - list functions according to their usage
 - 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 upstream but still experimental

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, Latency caused 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 profiling

■ Perf example how-to

- Needs proper HW support to work well for the kernel (not yet in the field)
 - Ignore and kernel profiling data until this is available!
- We had a case where new code caused cpus to scale badly
- `perf record "workload"`
 - Creates a file called `perf.data` that can be analyzed
- 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
```


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

- separate for counter and sampling
- Basic and/or Diagnostic mode

```
lscpumf -i
```

```
CPU-measurement counter facility
```

```
[...]
```

```
Sampling facility information for cpum_sf
```

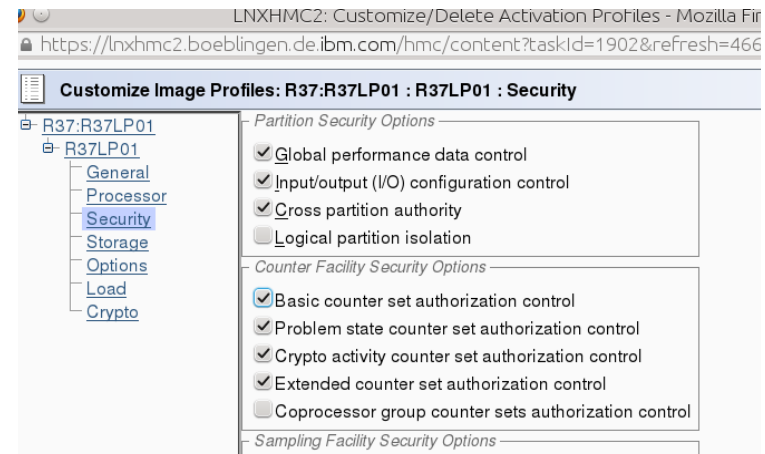
```
[...]
```

```
Authorized sampling modes:
```

```
    basic          (sample size:  32 bytes)
```

```
    diagnostic     (sample size:  85 bytes)
```

```
[...]
```



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 PROBLEM_STATE_CPU_CYCLES
757,589,098 PROBLEM_STATE_INSTRUCTIONS
705,740,759 L1I_PENALTY_CYCLES
576,226,424 L1D_PENALTY_CYCLES
40,675 sched:sched_wakeup
6.156288957 seconds time elapsed
```

- Still experimental
- Events
 - Cycles/Instructions globally
 - Note: counters are now readable, but aliases can still be used
 - e.g. r20 = PROBLEM_STATE_CPU_CYCLES
 - list of all existing events `lscpumf -C`
 - counters available to you `lscpumf -c`
 - Not only HW events, you can use any of the currently 163 tracepoints

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 slab allocator used
 - Named memory pools or Generic kmalloc pools
 - Active/total objects and their size
 - growth/shrinks of caches due to workload adaption

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.

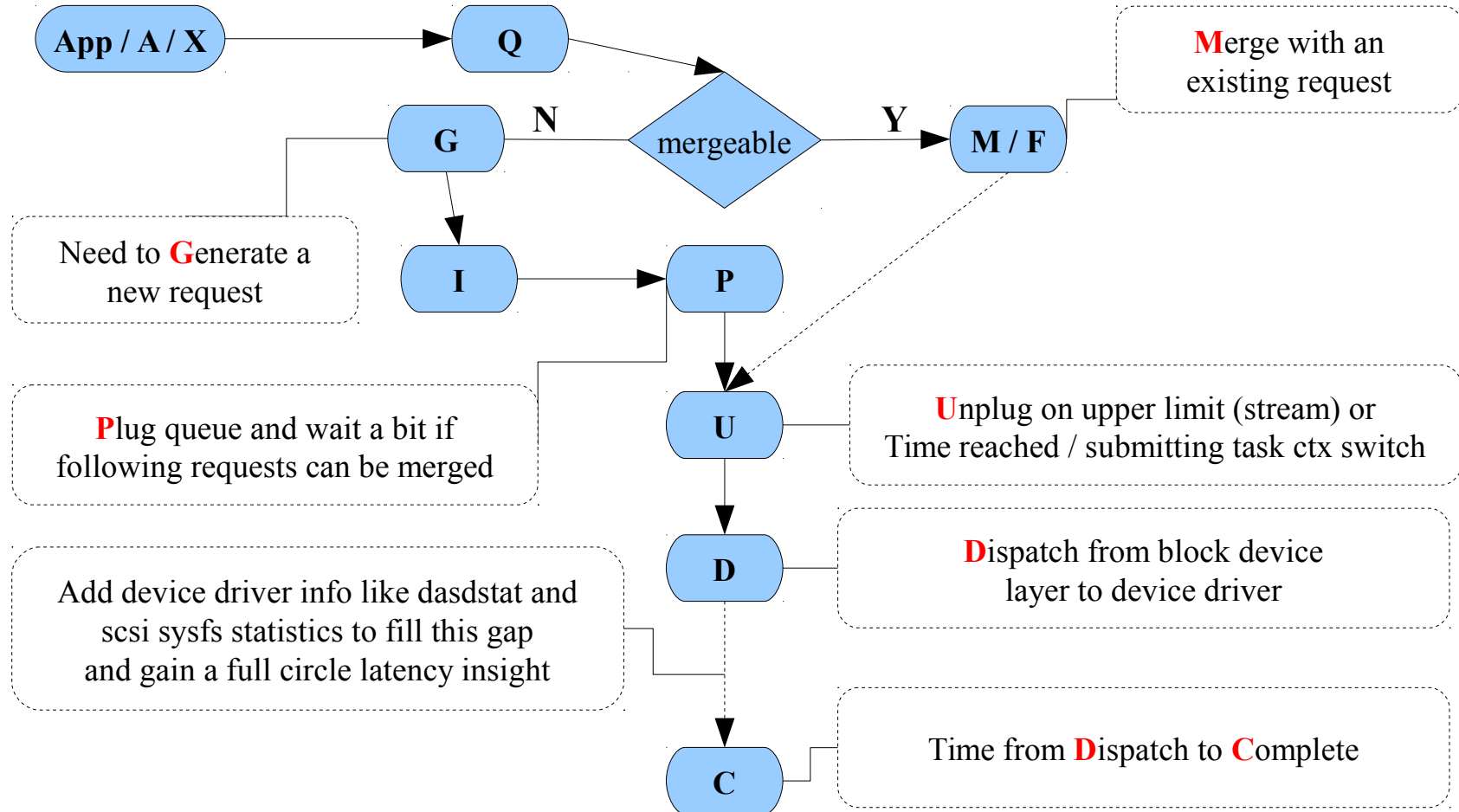
Plugging & Merges:

- 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.

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

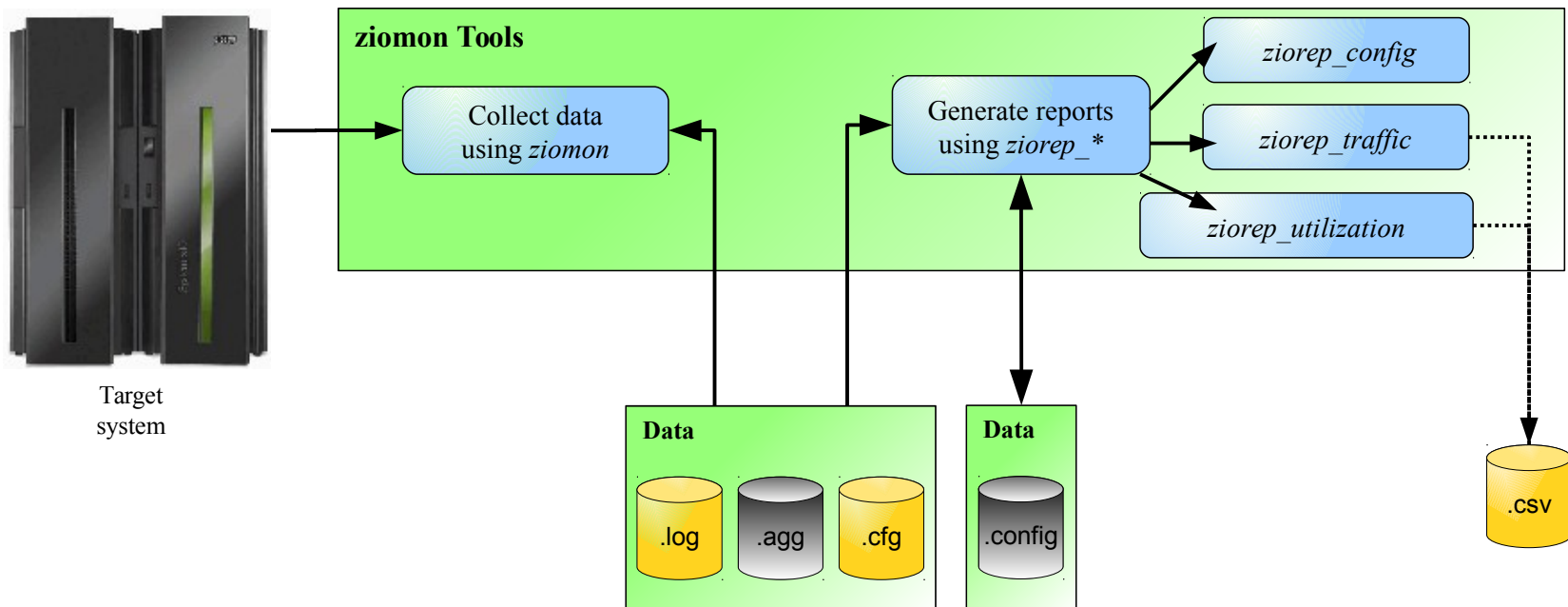
Reads Queued:	560,888,	2,243MiB	Writes Queued:	226,242,	904,968KiB
Read Dispatches:	544,701,	2,243MiB	Write Dispatches:	159,318,	904,968KiB
Reads Requeued:	0		Writes Requeued:	0	
Reads Completed:	544,716,	2,243MiB	Writes Completed:	159,321,	904,980KiB
Read Merges:	16,187,	64,748KiB	Write Merges:	61,744,	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	

ziomon

- Characteristics: in depth zfcps 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 Stefan Raspl
 - PDF: <http://www.vm.ibm.com/education/lvc/LVC0425.pdf>
 - Replay: <http://ibmstg.adobeconnect.com/p7zvdjz0yye/>

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

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

Method profile - IBM Support Assistant Workbench

File Administration Update Data Window Help

Support Assistant

Status

- Classes
- Environment
- Garbage Collection
- I/O
- Locking
- Memory
- Profiling

Method profile

Filter methods: [] [Apply] [Clear]

Samples	Self (%)	Self	Tree (%)	Tree	Method
25752	95.8		95.8		corn.ibm.tmc.demo.ComputingResourcesConsumer.generateCpuLoad(long)
94	0.35		0.78		org.apache.xml.dtm.ref.dom2dtm.DOM2DTM.addNode(org.w3c.dom.Node, int, in
77	0.29		1.37		org.apache.xml.dtm.ref.dom2dtm.DOM2DTM.nextNode()
44	0.16		0.49		org.apache.xml.dtm.ref.dom2dtm.DOM2DTM.processNamespacesAndAttributes(
42	0.16		1.53		org.apache.xml.dtm.ref.dom2dtm.DOM2DTM.getHandleFromNode(org.w3c.dom.N
42	0.16		0.16		org.apache.xml.dtm.ref.ExtendedType.equals(org.apache.xml.dtm.ref.ExtendedTy
39	0.15		0.18		java.lang.ClassLoader.defineClassImpl(java.lang.String, byte[], int, java.lang.C
34	0.13		0.13		org.apache.xml.dtm.ref.DTMDefaultBase.ensureSizeOfIndex(int, int)
25	0.093		0.26		org.apache.xml.dtm.ref.ExpandedNameTable.getExpandedTypeID(java.lang.String
24	0.089		0.19		java.lang.ClassLoader.loadClass(java.lang.String, boolean)
20	0.074		0.074		java.lang.Object.wait(long, int)
17	0.063		0.25		java.lang.J9VMInternals.initialize(java.lang.Class)
16	0.06		0.06		java.lang.String.indexOf(int, int)
13	0.048		0.048		org.apache.xml.dtm.ref.dom2dtm.DOM2DTMSChainedHashMap.get(java.lang.Ob
12	0.045		1.65		org.apache.xml.dtm.ref.DTMManagerDefault.getDTMHandleFromNode(org.w3c.dc
12	0.045		0.045		sun.nio.cs.ISO_8859_1\$Encoder.encodeArrayLoop(java.nio.CharBuffer, java.nio.F
12	0.045		0.15		java.lang.J9VMInternals.verifyImpl(java.lang.Class)
11	0.041		0.067		com.ibm.cds.CDSBundleFile.getEntry(java.lang.String)
11	0.041		0.17		org.apache.xml.dtm.ref.DTMDefaultBase.indexNode(int, int)

Connection

192.168.7.21:1972
54 MB received: Last updated 13:04:32
Some data was dropped because it was produced faster than the client could consume it. Around 9% of the data was lost.

Analysis and Recommendations

The method
ComputingResourcesConsumer.generateCpuLoad() is consuming approximately 96% of the CPU cycles. It may be a good candidate for optimization.

Invocation paths

Methods that call ComputingResourcesConsumer.generateCpuLoad()

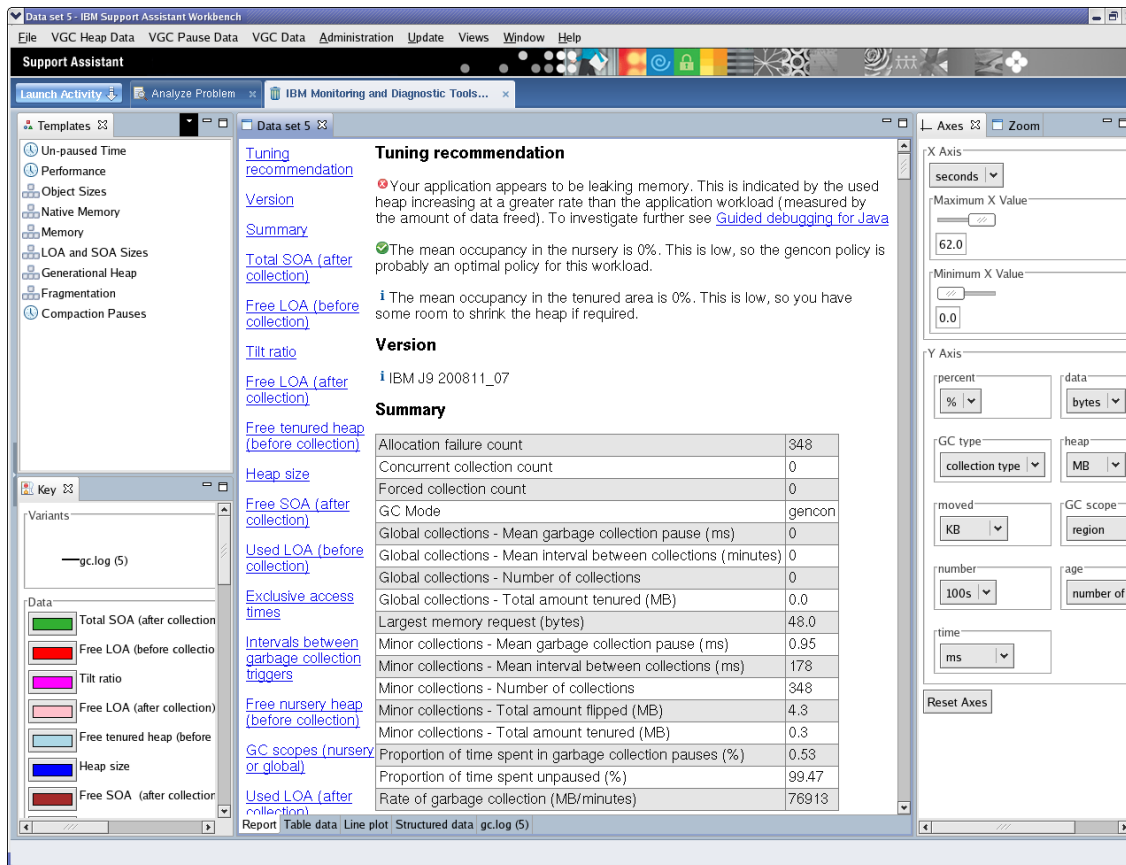
- ComputingResourcesConsumer.generateCpuLoad
 - TMCCDemoServlet.handleHttpRequest (100%)
 - TMCCDemoServlet.doGet (100%)
 - HttpServlet.service (100%)

- The method profiling view will show the name of the methods and their time consumption as percentage

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



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

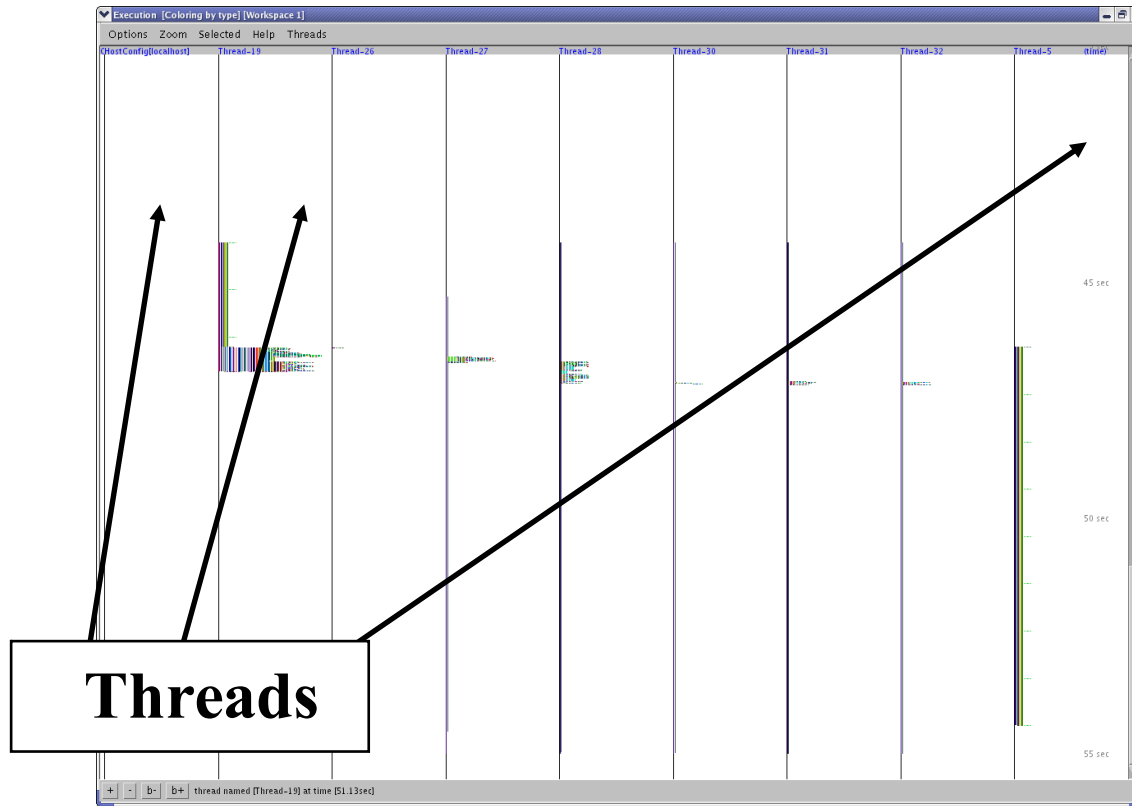
Java - Jinsight

- 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



- The execution view groups threads in columns, select one to zoom in
- function calls can be seen as vertical bar (colors represent the call depth)
- Time passes from top to bottom

End of Part IV

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



Agenda

Basic	Intermediate	Advanced	Master	Elite
– Utilization	– General thoughts	– Strace	– Perf	– Cachestat
– Scheduling		– Ltrace	– slabtop	– Smem
– Page Cache	– Sysstat	– Lsof	– Blktrace	– Valgrind
– Swapping	– Dastat	– Lsluns	– Ziemon	– Iqstats
	– Scsi I/O statistics	– Multipath	– Tcpdump	– Wireshark
– top	– iotop	– hytop	– Java Health Center	– Kernel Tracepoints
– ps	– Lszcrpt	– Dstat	– Java Garbage Collection and Memory visualizer	– Systemtap
– vmstat	– icastats	– Htop		
	– Lsqeth	– Netstat	– Jinsight	
	– Ethtool	– Socket Statistics		
	– Preparation	– Iptraf		

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

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/udev	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

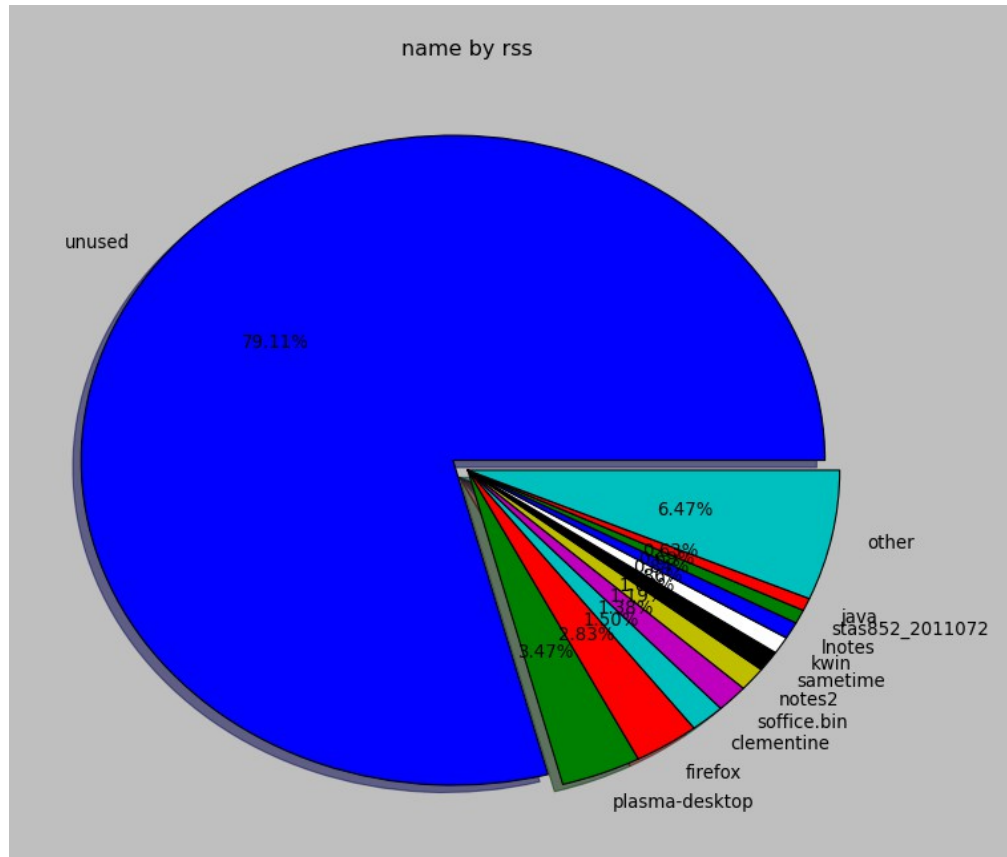
```
smem -m -tk -c "map count pids swap vss uss rss pss avgrss avgpss"
```

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



- There are many options to visualize memory distribution
- But avoid using that for monitoring as the `proc/<pid>/smaps` interface is an expensive one

Valgrind

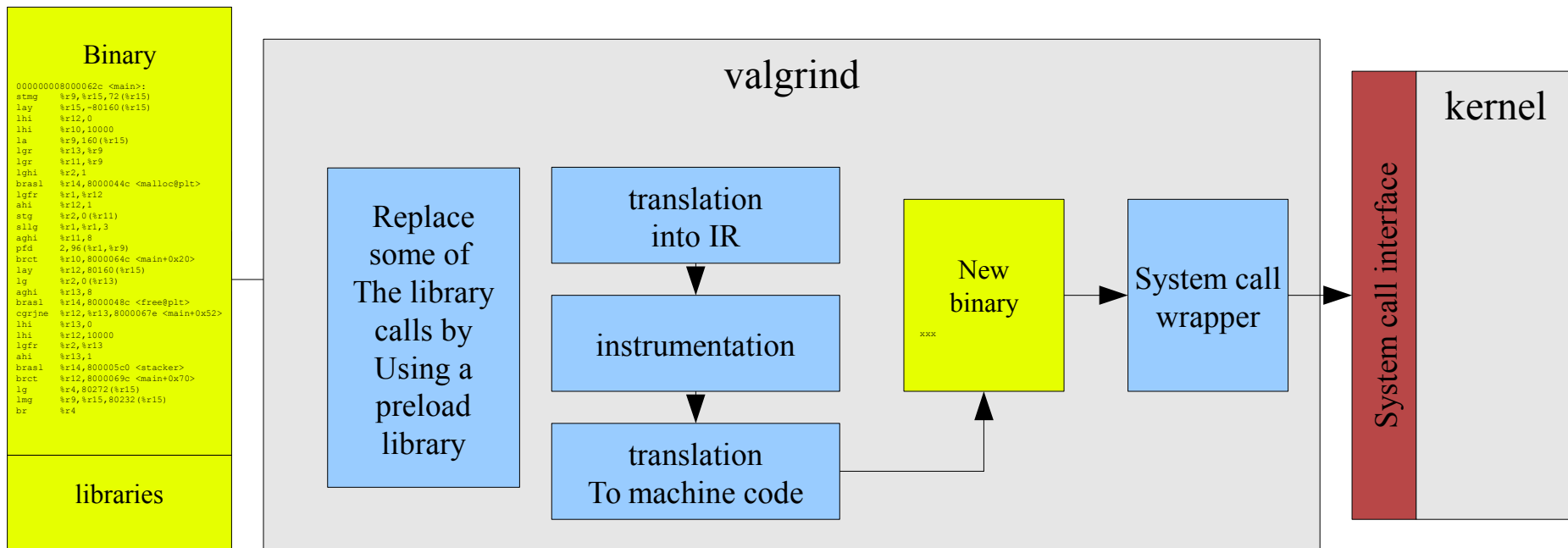
- Characteristics: in-depth memory analysis
- Objective: Find out where memory is leaked, sub-optimally cached, ...
- Usage: `valgrind [program]`
- Package: RHEL: `valgrind` SLES: `valgrind`

- 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
 - Binary is translated and augmented by valgrind instrumentation
 - New binary is then executed



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

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
 - Essentially this is the data source for “`mpstat -l`” which we had before

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

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
```

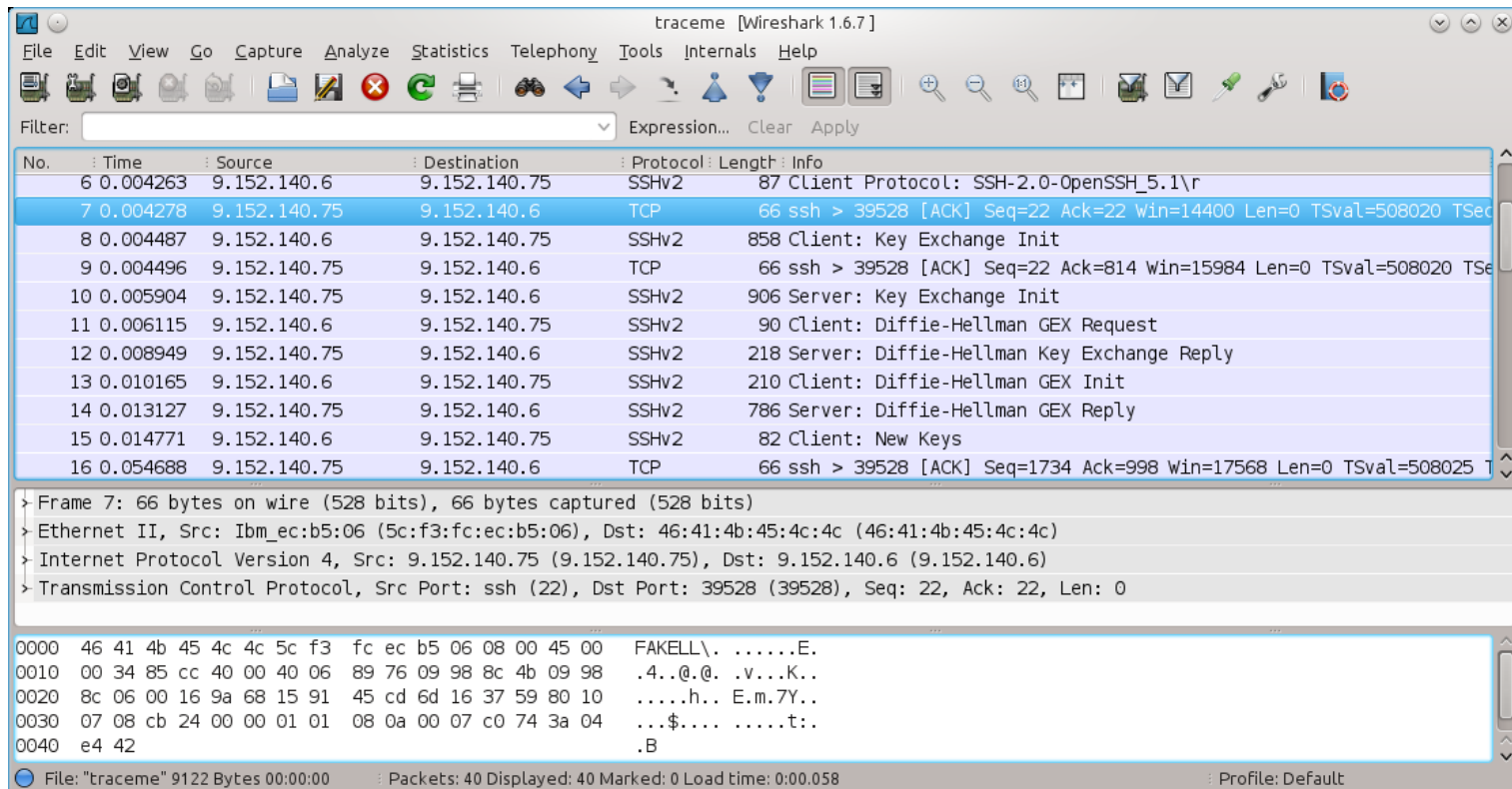
```
^C40 packets captured
```

```
40 packets received by filter
```

```
0 packets dropped by kernel
```

```
[scp to my system]
```

```
wireshark traceme
```



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 \longleftrightarrow 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

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>

There would be even more tools to cover ...

- Further tools - (no slides yet)
 - **Collectl** – full system monitoring
 - **Ftrace** – kernel function tracing
 - **Lttng** – complex latency tracing infrastructure (packages start to appear in Fedora 19)
 - **Nicstat, ktap, stap, ...**

End of Part V

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



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*

ehrhadt@de.ibm.com