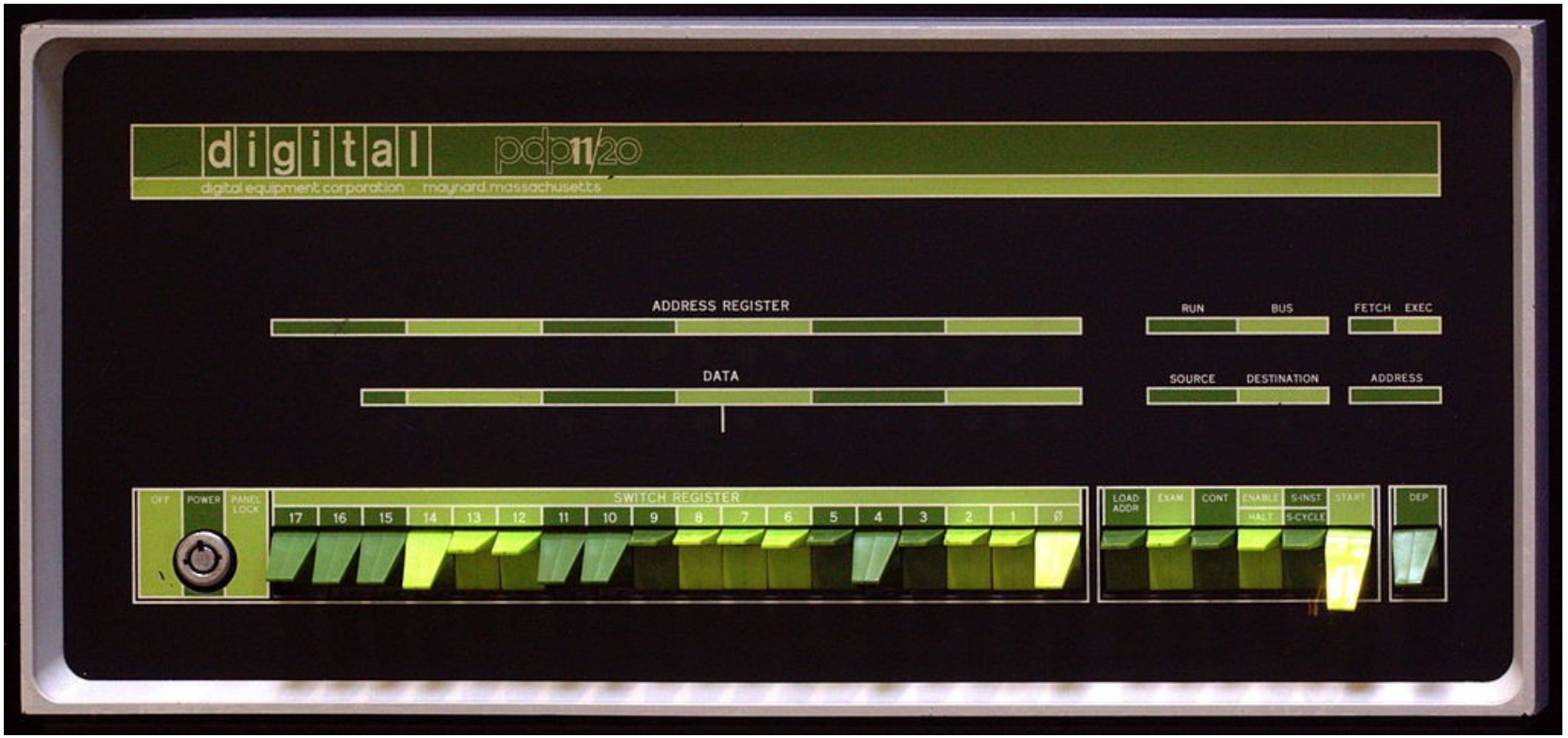
DTrace
Ein Blick unter die Haube von Solaris

Thomas Nau, kiz (Thomas.Nau@uni-ulm.de)

Kommunikations- und Informationszentrum (kiz)

- Die Aufgaben der "Abteilung Infrastruktur" umfassen
  - Cluster basierende universitätsweite Mail-, LDAP-, Portal-, Datenbank- und File-Services, ...
  - Betreuung von ca. 600 Desktop und Laptop Arbeitsplätzen
    - 25% Linux, 75% Windows
  - Backup Service für mehrere Universitäten in Baden-Württemberg
  - Landes HPC Cluster mit Schwerpunkt "Theoretische Chemie"
  - 4 lokale Netzwerke plus flächendeckendes Campus WLAN und MAN im Ulmer Stadtbereich
  - Telefonanlage mit ca. 14.000 Anschlüssen unter Einsatz von VoIP und 2-Draht Technik
  - Azubi Ausbildung

Back in the "Old Days"



Source: Wikipedia, Autor: Rama & Musée Bolo, Derivative Work: Morn

vmstat(1m)

- liefert statistische Daten über Kernel Threads, Platten IO, CPU Last, virtuellen Speicher und traps
- liefert nur wenige Anhaltspunkte wo Engpässe zu finden sind
  - hohe Anzahl von system-calls oder context-switches
  - viele page-in oder page-out Ereignisse

```
jedi# vmstat 5
 kthr      memory            page            disk          faults      cpu
 r b w   swap  free  re  mf pi po fr de sr m0 m1 m3 m1   in   sy   cs us sy id
 0 0 0 8620144 4617688 67 69 1  1  1  0  0  0  1  0  0 6147 1254 5529  0  1 99
 0 0 0 8702320 4730536 6  5  0  0  0  0  0  0  0  0  0 8010  532 7370  0  2 98
 0 0 0 8788856 4816776 6  1  0  0  0  0  0  0  0  0  0 7990  534 7428  0  2 98
^C
```

prstat(1m)

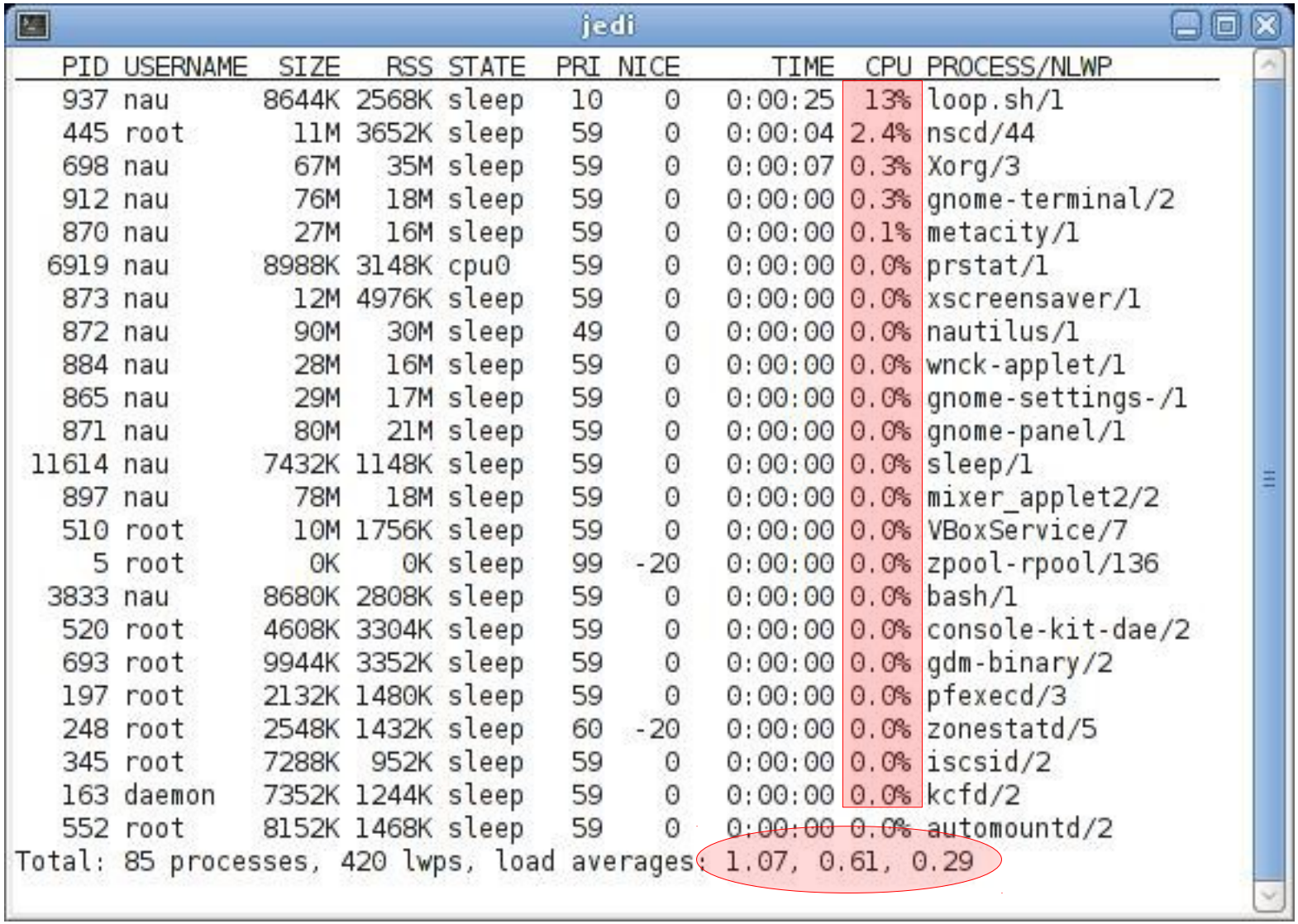
- liefert viele Informationen über laufende Prozesse und deren Threads
  - guter Ausgangspunkt falls Anwendungsprobleme vermutet werden

```
jedi# prstat -Lm
   PID USERNAME USR SYS TRP TFL DFL LCK SLP LAT VCX ICX SCL SIG PROCESS/LWPID
 16480 xvm      6.9 9.8 0.0 0.0 0.0  27  54 1.8 30K 114 .2M   0 qemu-dm/3
   363 xvm      0.1 0.2 0.0 0.0 0.0 0.0 100 0.0   4   1  2K   0 xenstored/1
 16374 root     0.0 0.1 0.0 0.0 0.0 100 0.0 0.0  10   0  1K   0 dtrace/1
  1644 xvm      0.1 0.1 0.0 0.0 0.0  33  66 0.0 569   7 835   0 qemu-dm/3
  2399 root     0.0 0.1 0.0 0.0 0.0 0.0 100 0.0  49   0 388   0 sshd/1
 16376 root     0.0 0.1 0.0 0.0 0.0 0.0 100 0.0  38   0 297   0 prstat/1
 11705 xvm      0.0 0.1 0.0 0.0 0.0  50  50 0.0 576  15 858   0 qemu-dm/4
 16536 root     0.0 0.1 0.0 0.0 0.0 0.0 100 0.0  48   0 286   0 vncviewer/1
```

prstat(1m)

- Anhaltpunkte:
  - USR+SYS — Effizienz?
  - LAT — Zu wenig CPUs?
  - DFL — Zu wenig Speicher?
  - SLP — Auf was wird gewartet?
  - LCK — Wer sperrt was und warum?

prstat(1m) Erklärungsnot



truss(1)

- *truss(1)* ist nicht dynamisch
  - Analyse vorübergehender oder kurzlebiger Probleme ist schwierig oder gar unmöglich
  - stoppt die Anwendung um Daten zu sammeln
    - die Anwendung, besonders das Timing, wird beeinflusst
- Auswertung zusammenhängender Prozesse schwierig
- die Kernel Grenze kann nicht überschritten werden

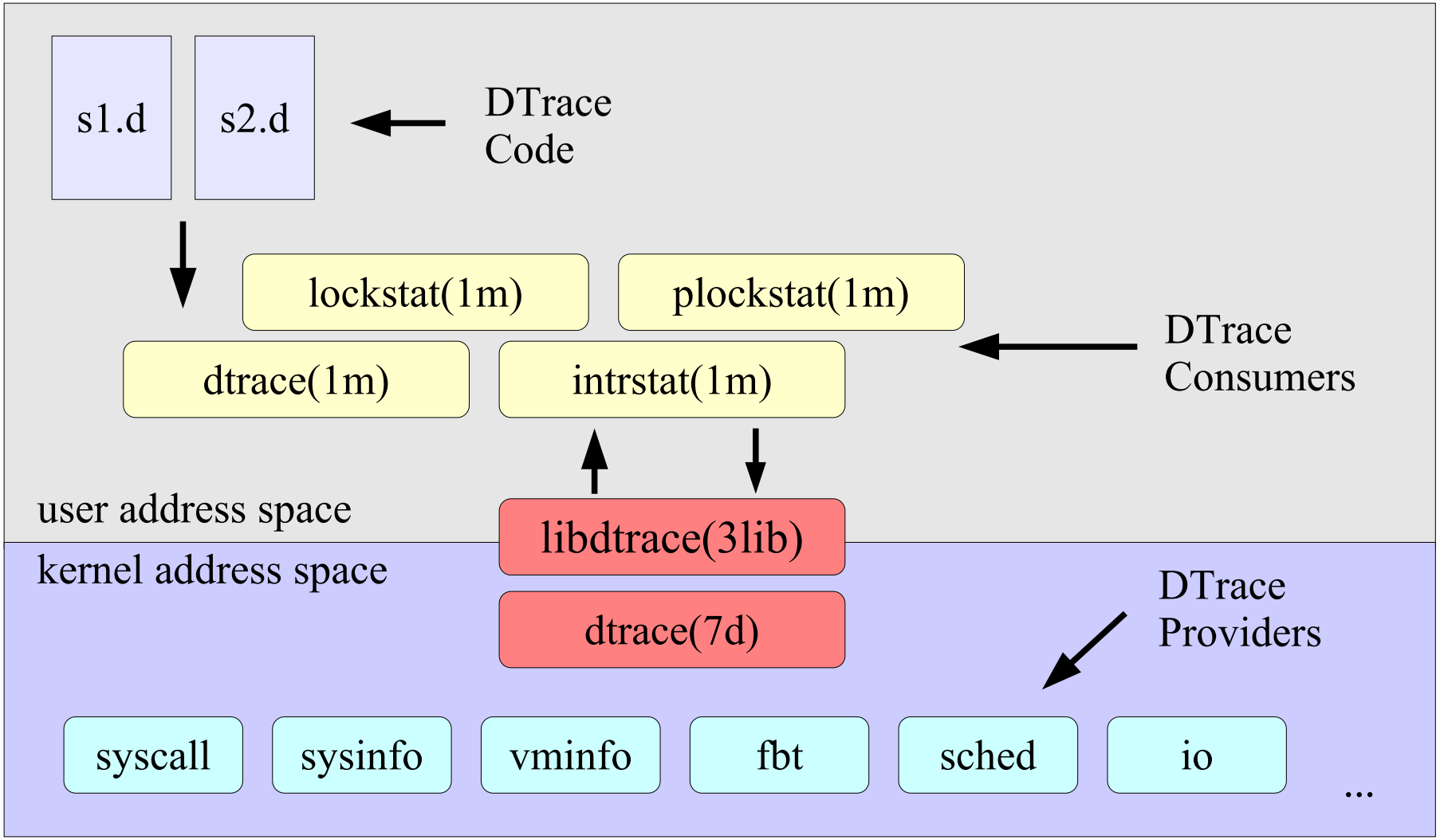
Was würde MacGyver tun um weiter zu kommen?

Hinweis: Beispiele stammen von Solaris 11 Hosts

DTrace, die "Dynamic Tracing Facility"

- instrumentiert dynamisch und effizient Kernel- aber auch Bibliotheks- und Anwendungs-Code
- derzeit 100.000+ „Probes“ verfügbar
  - Zahl ist von geladenen Kernel-Modulen abhängig
  - lassen sich beliebig und effizient ein- bzw. ausschalten
- skriptfähig durch "C" ähnliche Sprache "D"
- wird im Kernel Kontext ausgeführt
  - keine Schleifen (for, while, …) aus Sicherheitsgründen möglich
- viele Solaris Tools verwenden DTrace
  - more to come

DTrace Block Schaubild



Provider

- stellen sogenannte "Probes" zur Verfügung
- decken spezifische Bereiche ab und bereiten Daten auf
  - proc:
    Liefert Informationen über Prozesse und Threads
  - syscall:
    Probes für alle Ein- und Austrittspunkte der Solaris system-calls
  - io:
    Disk-IO bezogene Probes
  - fbt (function boundary tracing):
    Probes für die meisten Solaris Kernel Funktionen
- Updates bringen neue Provider
  - Solaris 11: ip, tcp, NFSv3, NFSv4, iSCSI, ...

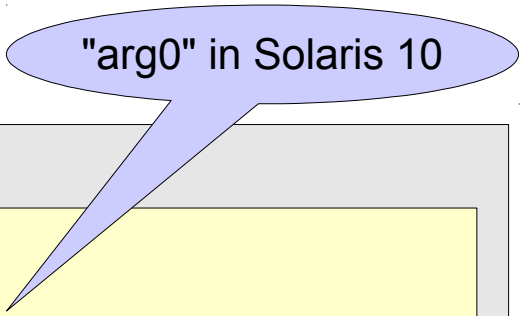
Probes

- Triggerpunkte die vielfältige Aktionen auslösen können
  - Aufzeichnung von Kernel- oder User-Stacks
  - Aufzeichnung von Datenstrukturen oder Speicherinhalten
  - Manipulation von Daten oder Speicherinhalten
  - ...
- Namens-Syntax für Probes

  provider:module:function:name

  sched:unix:resume:off-cpu

Beispiel: wer öffnet welche Datei?




```
obi-wan# dtrace -q -n \
  'syscall::open*:entry {
     printf("%-20s %s\n", execname, copyinstr(arg1));
}'
```

```
httpd                /www/htdocs/guc/bilder/grafik/m_kontakt.gif
httpd                /www/cms/uploads/pics/rw_logo2_03.png
mysqld               /www/mysqldata/cms/fe_users.MYI
mysqld               ./cms/fe_users.MYD
mysqld               /www/mysqldata/cms/fe_groups.MYI
...
```

“D” Sprachstruktur

- "D" definiert eine Art Unterprogramm-Sammlung
- einfache Struktur die sequentiell von oben nach unten ausgewertet wird; "Bedingungen" sind optional

```
Proben-Namen
/ Bedingung /
{
        Aktionen
}
```

- aus Sicherheitsgründen kennt "D" keine Schleifen, …
- einfach auch in eigene Anwendungen integrierbar
  - perl, MySQL, ...

Variable, Typen und Operatoren

- "C look and feel" mit geringen Unterschieden

  int8_t, uintptr_t, ...

- Skalare, Strings, Zeiger, *structs* und *unions* sind verfügbar
  - provider und consumer laufen in unterschiedlichen Adressräumen
- Variable müssen nicht vor Verwendung definiert werden
  - viele schon vordefiniert: *pid, uid, execname, curcpu, …*
  - lesender Zugriff auf kernel variable möglich: *`kmem_flags*
- grundlegende Arithmetik-, Logik- und Vergleichs-Operatoren stehen zur Verfügung
- Vergleichs-Operatoren können auf Strings angewendet werden; liefern 1 im Falle von Gleichheit sonst 0

Ausgabe bzw. Sammeln von Daten

- printf() arbeitet wie in "C" mit Format-Strings
  - *printa()* arbeitet analog zu *prinf()* mit "aggregations"
  - nützliche Erweiterungen verfügbar
    - 'a' Ausgabe eines Zeigers als Kernel Symbol
    - 'p' Hexadezimal Ausgabe eines Zeigers
    - 'S' Ausgabe von Strings wobei nicht druckbare Zeichen mit "\" dargestellt werden
    - 'Y' formatiert ein Argument "Nanosekunden seit 1.1.1970" als für Menschen verständlichen String
- *stack(), ustack()*
- *tracemem()*

DTrace's Warp Antrieb: Aggregations

- Aggregations nutzen eine besondere mathematische Eigenschaft bestimmter Funktionen aus
  - f(f(x0) U f(x1) U ... U f(xn)) = f(x0 U x1 U ... U xn)

???

Beispiel: SUM Aggregation

$$\Sigma\ 1, 2, 3, \ldots 99, 100 = 5050$$

$\Sigma\ 1, \ldots 10 = 55$

$\Sigma\ 11, \ldots 20 = 155$

...

$\Sigma\ 91, \ldots 100 = 955$

$\Big\}\ \Sigma = 5050$

DTrace's Warp Antrieb: Aggregations

- helfen den Speicherbedarf gering zu halten
  - keine Notwendigkeit alle Daten zwischenzuspeichern
  - Probleme mit der Skalierung werden vermieden
- derzeit
  - count(), sum()
  - min(), max(), avg(), stddev()
  - quantize(), lquantize()
- der Index von Aggregations ist nahezu beliebig, etwa *ustack()* und *stack()* oder *execname, pid, ...*

Beispiel: Solaris 11 IP Provider Probes

- send — Solaris Netzwerk Stack verschickt ein IP Paket
- receive — Solaris Netzwerk Stack empfängt ein IP Paket

- der IP Provider übergibt Daten über 6 Pointer auf:

pktinfo_t *	csinfo_t *	ipinfo_t *
ifinfo_t *	ipv4info_t *	ipv6info_t *

ipinfo_t und *ipv4info_t* Definitionen

```
typedef struct ipinfo {
        uint8_t ip_ver;         /* IP version (4, 6)     */
        uint16_t ip_plength;    /* payload length        */
        string ip_saddr;        /* source address        */
        string ip_daddr;        /* destination address   */
} ipinfo_t;

typedef struct ipv4info {
    ...
    uint8_t ipv4_protocol;      /* next level protocol   */
    string ipv4_protostr;       /* same as a string      */
    ...
    ipaddr_t ipv4_src;          /* source address        */
    ipaddr_t ipv4_dst;          /* destination address   */
    string ipv4_saddr;          /* src address, string   */
    string ipv4_daddr;          /* dest address, string  */
    ...
} ipv4info_t;
```

Beispiel: Verteilung der IP Paket Größe

- basiert auf Beispiel von

  https://wikis.oracle.com/display/DTrace/ip+Provider

- gut geeignet für IP basierte File- oder Storage Server

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

ip:::send
{
    @c[args[2]->ip_daddr, execname] =
        quantize(args[2]->ip_plength);
}
```

2^n Verteilung

Beispiel: Verteilung der IP Paket Größe

```
...
192.168.23.23                                                  nfsd

    value  ------------- Distribution -------------  count
       16 |                                         0
       32 |                                         1
       64 |                                         0
      128 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  244
      256 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  88
      512 |                                         0
     1024 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  32
     2048 |                                         0
...
```

Hilfreiche Funktionen für Aggregations

- *trunc(@aggr [, n])*
  löscht eine Aggregation oder Teile davon

  n > 0 die obersten n Einträge bleiben erhalten

  n < 0 die untersten n Einträge bleiben erhalten

- *clear(@aggr)*
  setzt die Werte einer Aggregation auf 0

- *normalize(@aggr, val)*
  dividiert alle Werte durch *val*

- *denormalize()*
  macht die Normierung rückgängig

- Tipp: im Zusammenspiel mit der *tick* probe lassen sich einfach top-ten artige Ausgaben realisieren

Sortierung von Aggregations

- die Sortierung lässt sich über Optionen steuern

```
setopt("aggsortkey", true);
```

- Möglichkeiten

aggsortkey	Sortierung nach Index, nicht Wert
aggsortkeypos	Nummer des Index nach dem sortiert wird
aggsortrev	Umkehrung der Reihenfolge

Beispiel: Netzverkehr einzelner Clients

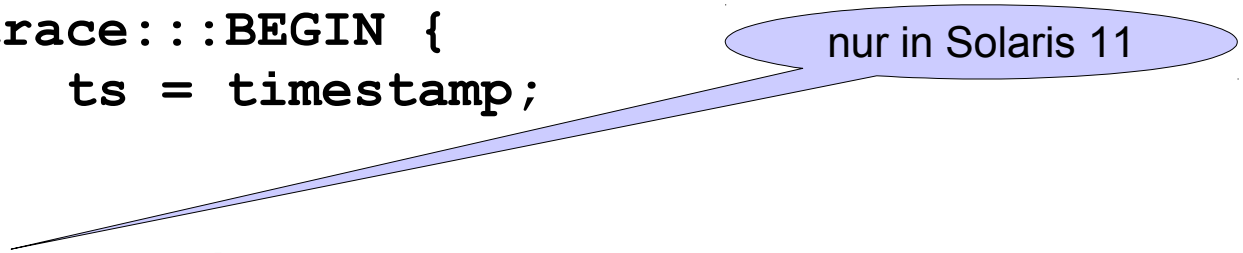
```
#!/usr/sbin/dtrace -s

#pragma D option quiet

dtrace:::BEGIN {
    ts = timestamp;
}

ip:::send {
    @bytes[args[2]->ip_daddr, probename] =
        sum(args[2]->ip_plength);
}

ip:::receive {
    @bytes[args[2]->ip_saddr, probename] =
        sum(args[2]->ip_plength);
}
```



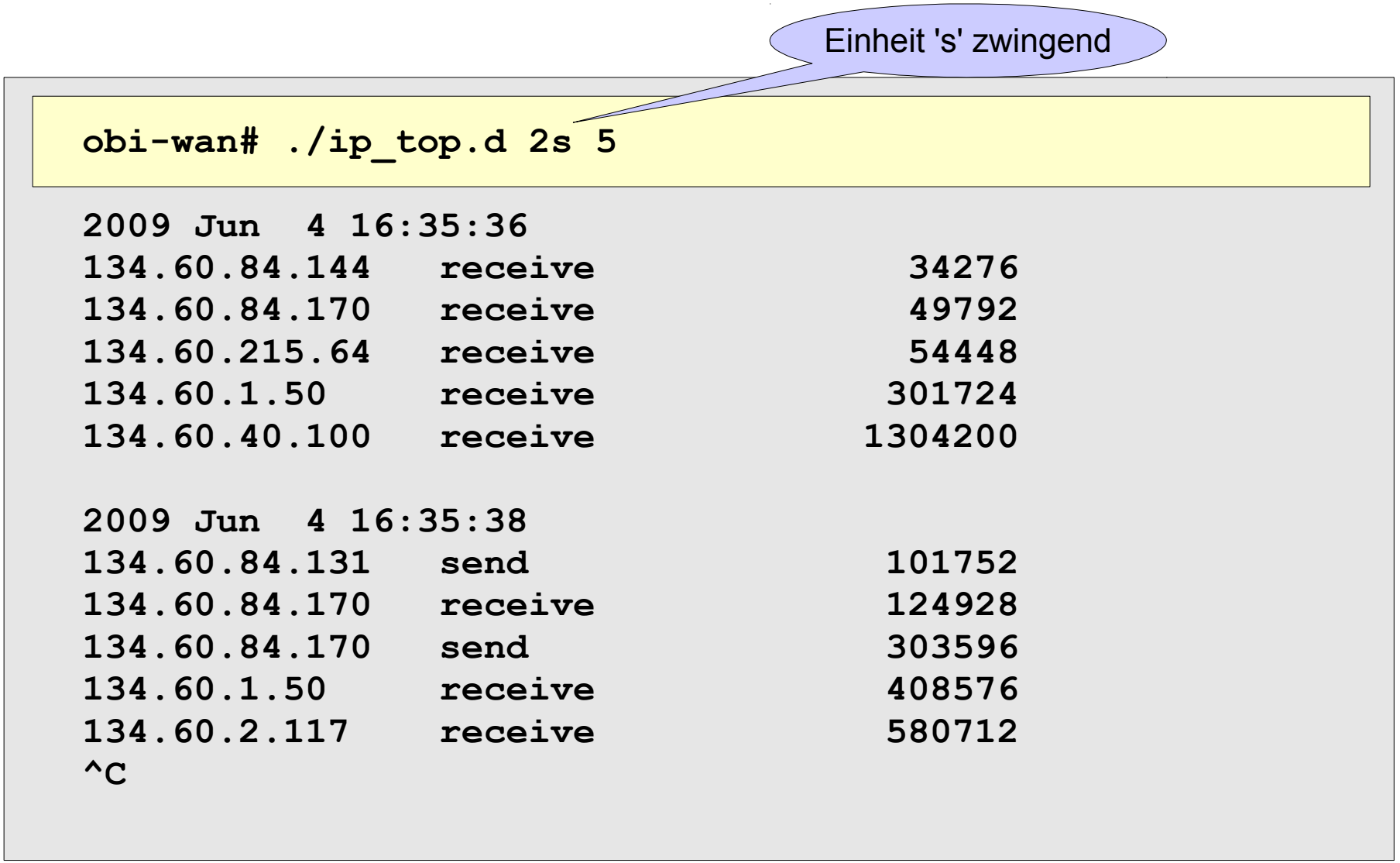
Beispiel: Netzverkehr einzelner Clients

```
/* output collected data
 */
tick-$1 {
        /* top-n only */
    trunc(bytes@, $2);

        /* print per second rate */
    normalize(@bytes,(timestamp-ts) / 1000000000);
    printf("\n%Y\n", walltimestamp);
    printa("%-15s %-10s %@15d\n", @bytes);

        /* start next interval from scratch */
    ts = timestamp;
    trunc(@bytes);
}
```

Beispiel: Netzverkehr einzelner Clients

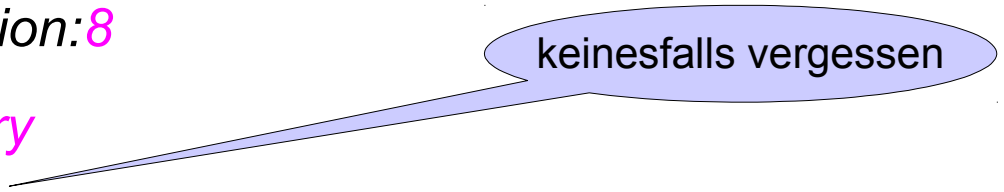


Der *pid* Provider

- *der pid* Provider erlaubt die Verfolgung von Funktionen oder Instruktionen einer Anwendung
  - Einschränkung: "inlining" durch den Compiler
- Offset Adressierung bezüglich Start der Routine

  pid54321:my-object:my-function:8

  pid54321:libc.so.1:strcpy:entry
  pid54321:libc.so:strcpy:entry
  pid54321:libc:strcpy:entry



- die Probes werden bei Bedarf generiert
  - nicht Bestandteil der Ausgabe von "*dtrace -l*"
- Tipp: Kommandozeilen Optionen "-p" und "-c"

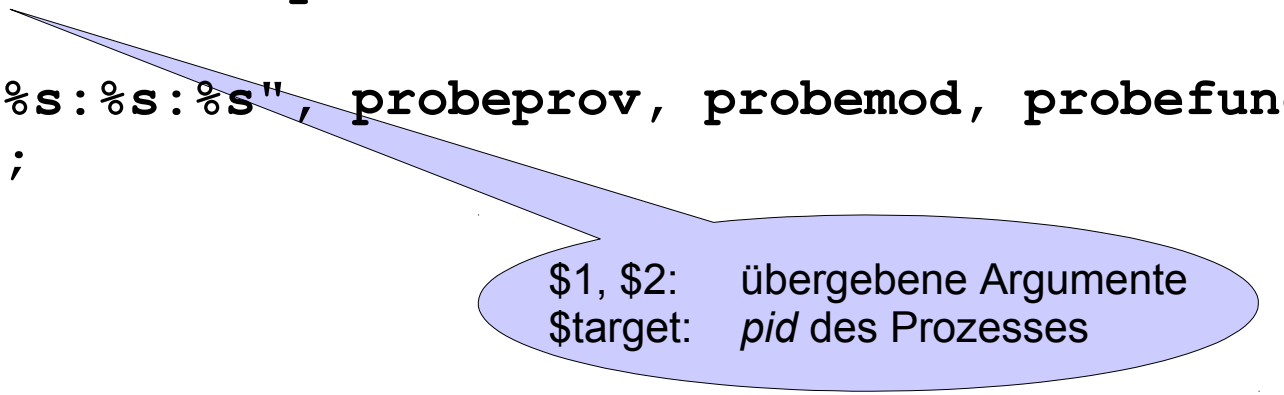
Beispiel: Aufruf-Stacks (lib_call_stack.d)

```
#!/usr/sbin/dtrace -s

/* uses the 'pid' Provider to print call stacks */

#pragma D option quiet

pid$target:$1:$2:entry
{
    printf("%s:%s:%s", probeprov, probemod, probefunc);
    ustack();
}
```



Beispiel: Aufruf-Stacks

```
obi-wan# ./lib_call_stack.d -c ls 'libc' 'str*'
```

```
lib_call_stack.d
```

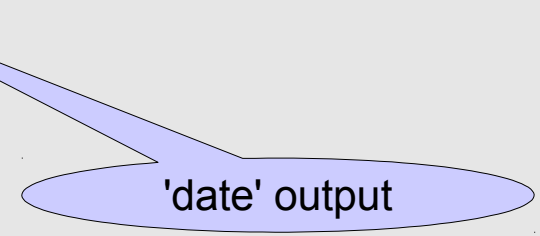
'ls' output

```
pid1002:libc.so.1:strcmp
              libc.so.1`strcmp
              libc.so.1`setlocale+0x1378
              ls`main+0x22
              ls`_start+0x7a
pid1002:libc.so.1:strlen
              libc.so.1`strlen
              ls`xstrdup+0x12
              ls`main+0x486
              ls`_start+0x7a
pid1002:libc.so.1:strlen
              libc.so.1`strlen
              ls`xstrdup+0x12
```

Weiteres Beispiel: libc Trace

```
obi-wan# ./libc_trace.d -F -c date

dtrace: script './libc_trace.d' matched 5789 Probes
Sunday, November 11, 2012 11:50:04 AM CET
dtrace: pid 16648 has exited
CPU FUNCTION
  5  -> __tls_static_mods
  5    -> lmalloc
  5      -> getbucketnum
  5      <- getbucketnum
  5      -> initial_allocation
  5        -> __systemcall
  5        <- __systemcall
  5      <- initial_allocation
  5    <- lmalloc
...
```



A blast from the past ...

Beispiel: Active Directory Controller

- wir verwendeten OpenSolaris xVM (Xen) um Windows Server 2008 zu virtualisieren
- Platten IO war auf Grund der para-virtualisierten Treiber sehr gut
- Problem: massiver Leistungseinbruch nachdem ein System die Funktion eines AD Controllers übernahm

Beispiel: Active Directory Controller

- "vmstat 5" liefert Hinweise auf eine hohe system-call Rate
- "prstat -Lm" zeigt welcher Prozess es ist

```
PID   NAME USR SYS TRP TFL DFL LCK SLP LAT VCX ICX SCL SIG PROC/NLWP
-----------------------------------------------------------------
16480 xvm  6.9 9.8 0.0 0.0 0.0  27  54 1.8 30K 114 .2M 0 qemu-dm/3
  363 xvm  0.1 0.2 0.0 0.0 0.0 0.0 100 0.0   4   1  2K 0 xenstored/1
16374 root 0.0 0.1 0.0 0.0 0.0 100 0.0 0.0  10   0  1K 0 dtrace/1
 1644 xvm  0.1 0.1 0.0 0.0 0.0  33  66 0.0 569   7 835 0 qemu-dm/3
 2399 root 0.0 0.1 0.0 0.0 0.0 0.0 100 0.0  49   0 388 0 sshd/1
16376 root 0.0 0.1 0.0 0.0 0.0 0.0 100 0.0  38   0 297 0 prstat/1
11705 xvm  0.0 0.1 0.0 0.0 0.0  50  50 0.0 576  15 858 0 qemu-dm/4
16536 root 0.0 0.1 0.0 0.0 0.0 0.0 100 0.0  48   0 286 0 vncviewer/1
...
```

Beispiel: Active Directory Controller

```
#!/usr/sbin/dtrace -s

/* get some statistic about system call rates */

BEGIN {
        timer = timestamp;
}

syscall:::entry {
        @c[pid, execname, probefunc] = count();
}

tick-5s {
        trunc(@c, 10);
        normalize(@c, (timestamp-timer) /1000000000);
        printa("%5d %-20s %6@d %s\n", @c);
        clear(@c);
        printf("\n");
        timer = timestamp;
}
```

Beispiel: Active Directory Controller

```
leto# ./count_syscalls.d


  209 nscd                                   27 xstat
16376 prstat                                 35 pread
16480 qemu-dm                               117 pollsys
16480 qemu-dm                               123 read
16480 qemu-dm                               136 ioctl
11705 qemu-dm                               145 pollsys
 1644 qemu-dm                               151 pollsys
16374 dtrace                                331 ioctl
16480 qemu-dm                             35512 lseek
16480 qemu-dm                             35607 write
```

Beispiel: Active Directory Controller

```
leto# ./seek-write-stat.d

lseek fdesc       delta       count
            5          26          28
            5          29          28
            5           0          42
            5          21          42
            5           1      134540
write fdesc        size       count
            5          21          42
           15           4          54
           16           4          63
           14           4         441
            5           1      134554
```

Beispiel: Active Directory Controller

- Überprüfung von file-descriptor #5 mit Hilfe von *pfiles(1)* liefert Hinweise darauf, das es sich um Zugriffe auf die virtuelle Platte handeln muss

```
leto# pfiles 16480
```

```
...
   5: S_IFREG mode:0600 dev:182,65543 ino:26 uid:60
gid:0 size:11623923712
      O_RDWR|O_LARGEFILE
      /xvm/hermia/disk_c/vdisk.vmdk
...
```

Beispiel: Active Directory Controller

```
#!/usr/sbin/dtrace -s

/* print call stack statistics for "qemu-dm"
 * for the lseek() and write() system calls
 */

#pragma D option quiet

syscall::lseek:entry, syscall::write:entry
/ execname == "qemu-dm" /
{
        @c[probefunc, ustack()] = count();
}

tick-5s {
        trunc(@c, 3);
        printa(@c);
        clear(@c);
}
```

Beispiel: Active Directory Controller

```
...
  write
                libc.so.1`__write+0xa
                qemu-dm`RTFileWrite+0x37
                qemu-dm`RTFileWriteAt+0x48
                qemu-dm`vmdkWriteDescriptor+0x1d5
                qemu-dm`vmdkFlushImage+0x23
                qemu-dm`vmdkFlush+0x9
                qemu-dm`VDFlush+0x91
                qemu-dm`vdisk_flush+0x1c
                qemu-dm`bdrv_flush+0x2e
                qemu-dm`ide_write_dma_cb+0x187
                qemu-dm`bdrv_aio_bh_cb+0x16
                qemu-dm`qemu_bh_poll+0x2d
                qemu-dm`main_loop_wait+0x22c
                qemu-dm`main_loop+0x7a
                qemu-dm`main+0x1886
                qemu-dm`_start+0x6c
               28758
```

Beispiel: Active Directory Controller

- auf Grund des call-stacks liegt die Vermutung nahe, dass
  - Windows den Disk Cache "flushed"
  - dieser ggf. deaktiviert ist
  - letzteres war der Fall
- Aktivierung des Disk Cache in einem Windows Start-Skript beseitigt die IO Probleme
  - Verifizierung über eine DTrace Quantisierung der IO Größe

Beispiel: Active Directory Controller

```
leto# ./qemu-stat-quant.d 5
```

```
  write
  value  ------------- Distribution ------------- count
    256 |                                         0
    512 |@@@@@@                                   14
   1024 |                                         0
   2048 |@@@@                                     10
   4096 |@@@@@@@@@@@@@@@@@                        39
   8192 |@@@@@@@@@@@@@@@@                         36
  16384 |                                         0
```

Literatur

- hervorragende Informationsquelle zumal auch alle neuen und nur in Solaris 11 verfügbaren Provider dokumentiert sind

  http://wikis.oracle.com/display/DTrace

- das DTrace Manual (PDF)

  http://docs.oracle.com/cd/E26502_01/pdf/E28556.pdf

- Beispiele finden sich unter

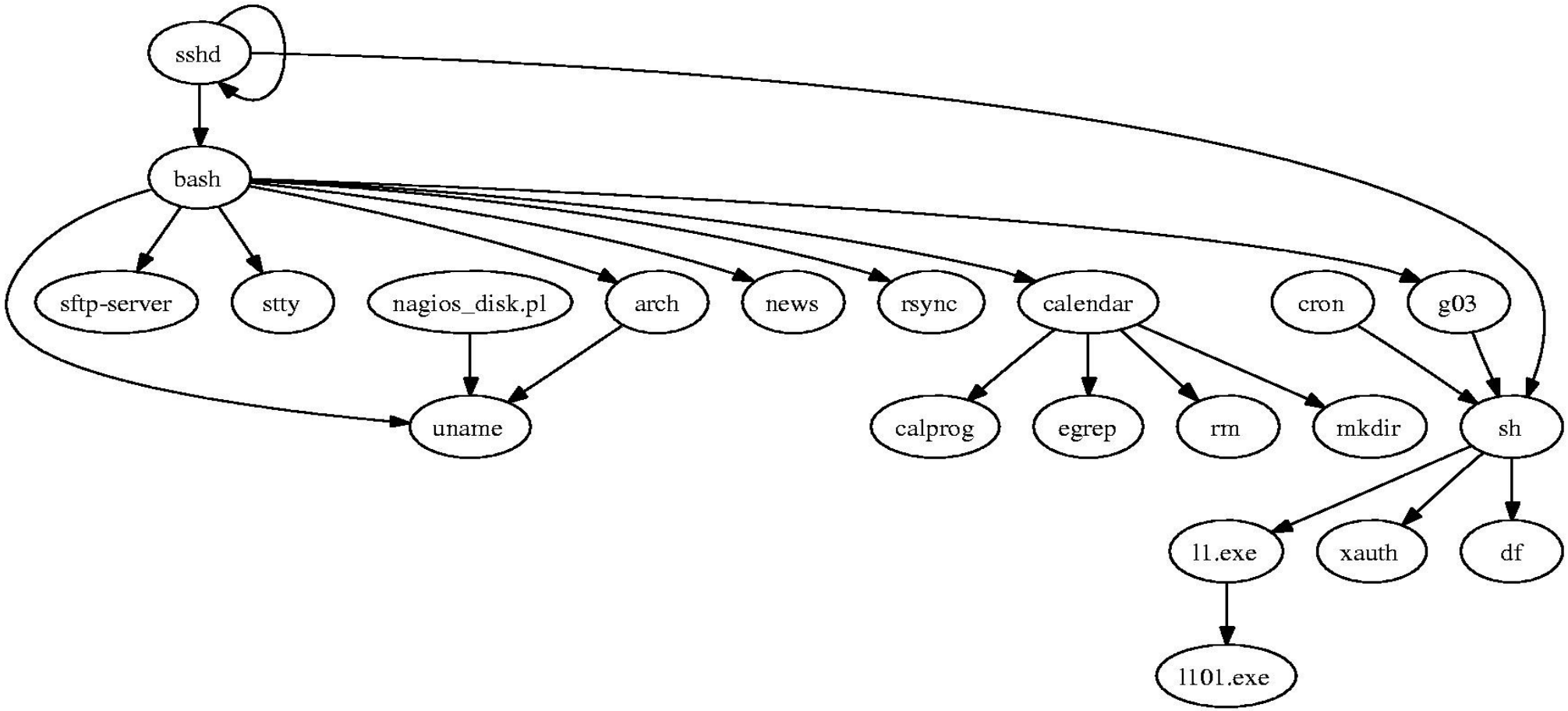
  https://wikis.oracle.com/display/DTrace/DTrace

  http://dtracebook.com

  http://www.brendangregg.com/dtrace.html

  /usr/demo/dtrace

Pimp my DTrace: *proc* provider plus Graphviz



Danke für's Zuhören!