Joyent

ZFS Performance Analysis and Tools

Brendan Gregg
Lead Performance Engineer

brendan@joyent.com
@brendangregg

October, 2012

whoami

Joyent

- G'Day, I'm Brendan
- These days I do systems performance analysis of the cloud
- Regarding ZFS:
    - Perf analysis of ZFS (mostly using DTrace) for 5+ years, both enterprise and cloud usage
        - Wrote many DTrace-based ZFS perf analysis tools including those in the DTrace book
    - Developed ZFS L2ARC while at Sun

Who is Joyent

- Cloud computing provider (public cloud + software)
- Use ZFS as much as possible:
  - Host storage
  - Guest storage: OS virtualization (SmartOS), and KVM guests (Linux, Windows)
- We use ZFS because
  - Reliability: checksums, COW
  - Features: snapshots, clones, compression, ...
  - Performance: large ARCs
  - It can boil oceans

Joyent, cont.

Joyent

- We build tools for ZFS automation and observability.
- Performance is a key company feature.
  - Need to solve FS/disk issues fast.

Agenda

Joyent

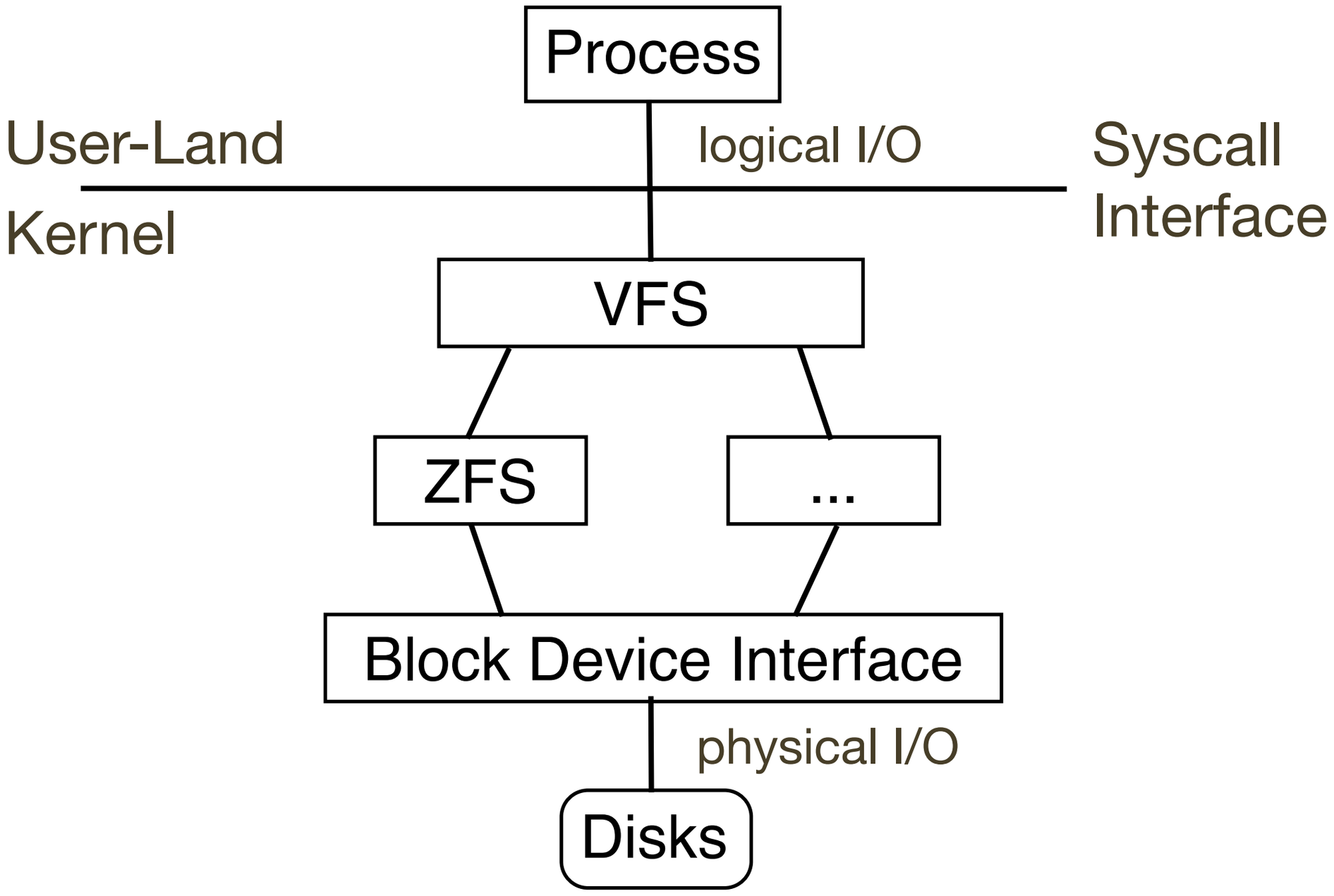
- My top 12 tools for ZFS performance analysis (unsorted):
  - iostat
  - vfsstat
  - zfsslower.d
  - iosnoop
  - iostacks
  - metaslab_free.d
  - spasync.d
  - arcstat
  - arcaccess.d
  - latency counters
  - scatter plots
  - heat maps (CA)

For cloud computing
from within a Zone, add:

- mysqld_pid_fslatency.d
- syscall with fi_fs == zfs

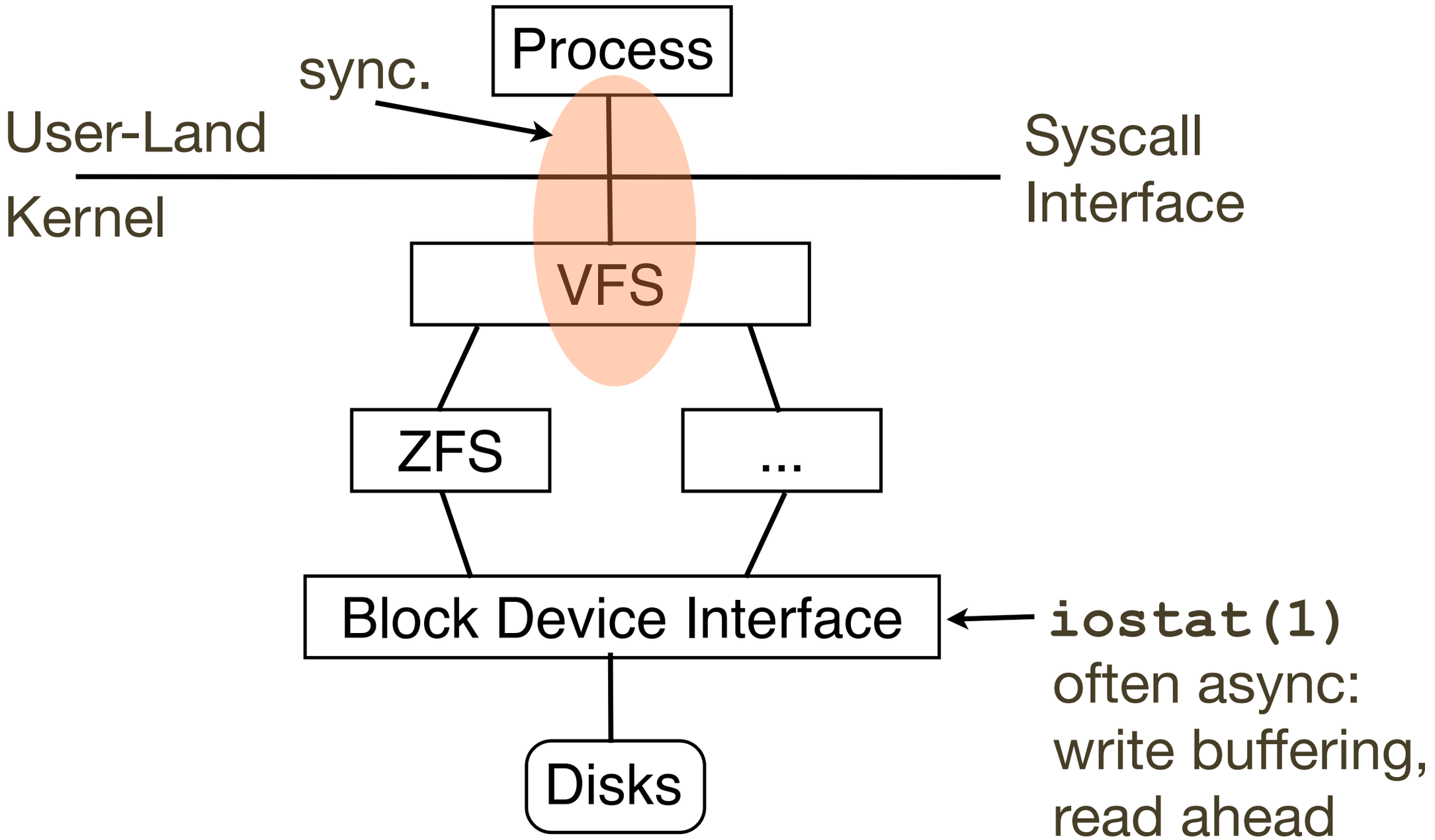
Functional diagram: full stack

- Unix 101



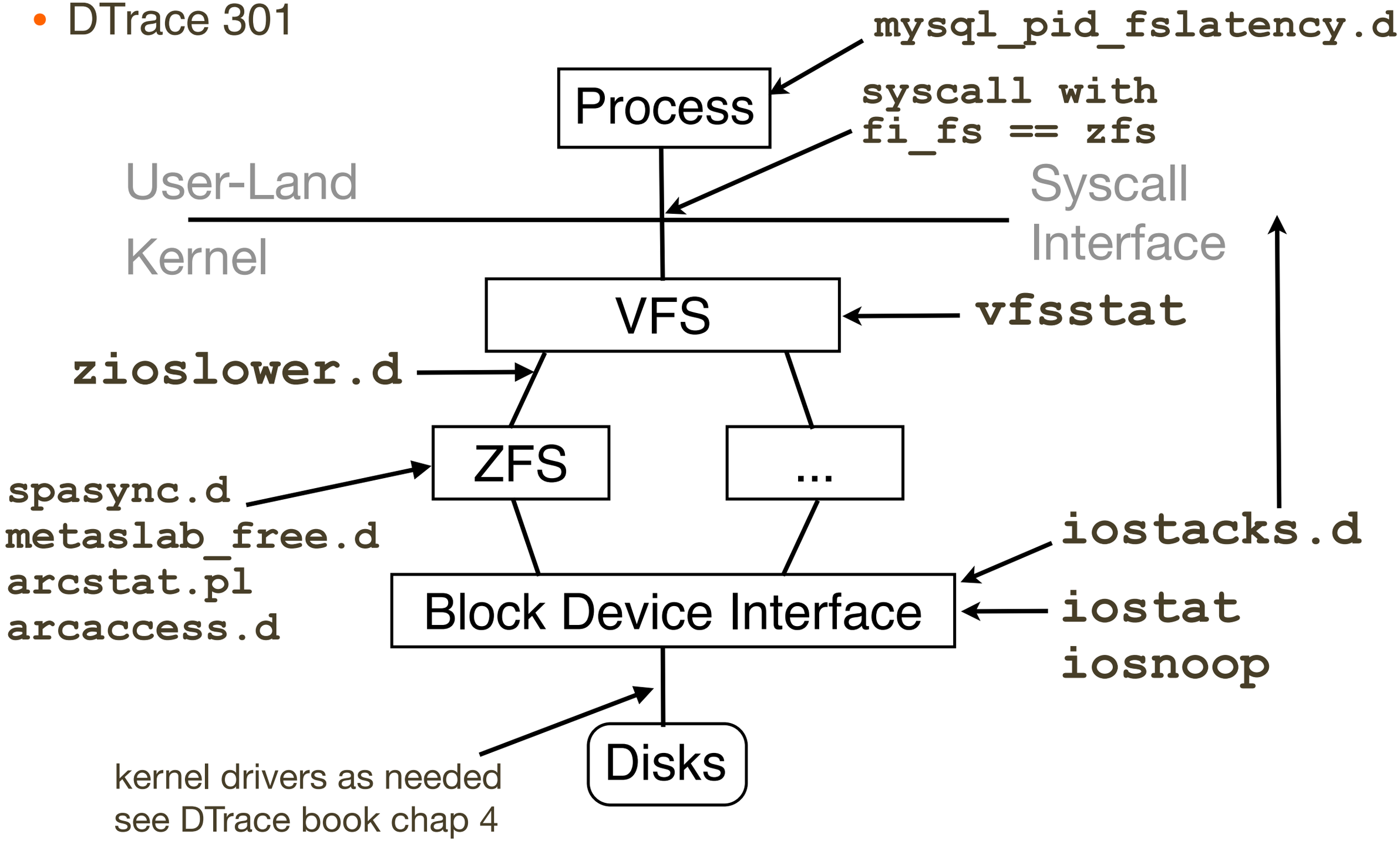
Functional diagram: full stack, cont.

- Unix 102



Functional diagram: full stack, cont.

- DTrace 301



ZFS Internals

Joyent

- That’s just my top 12
- Use more as needed

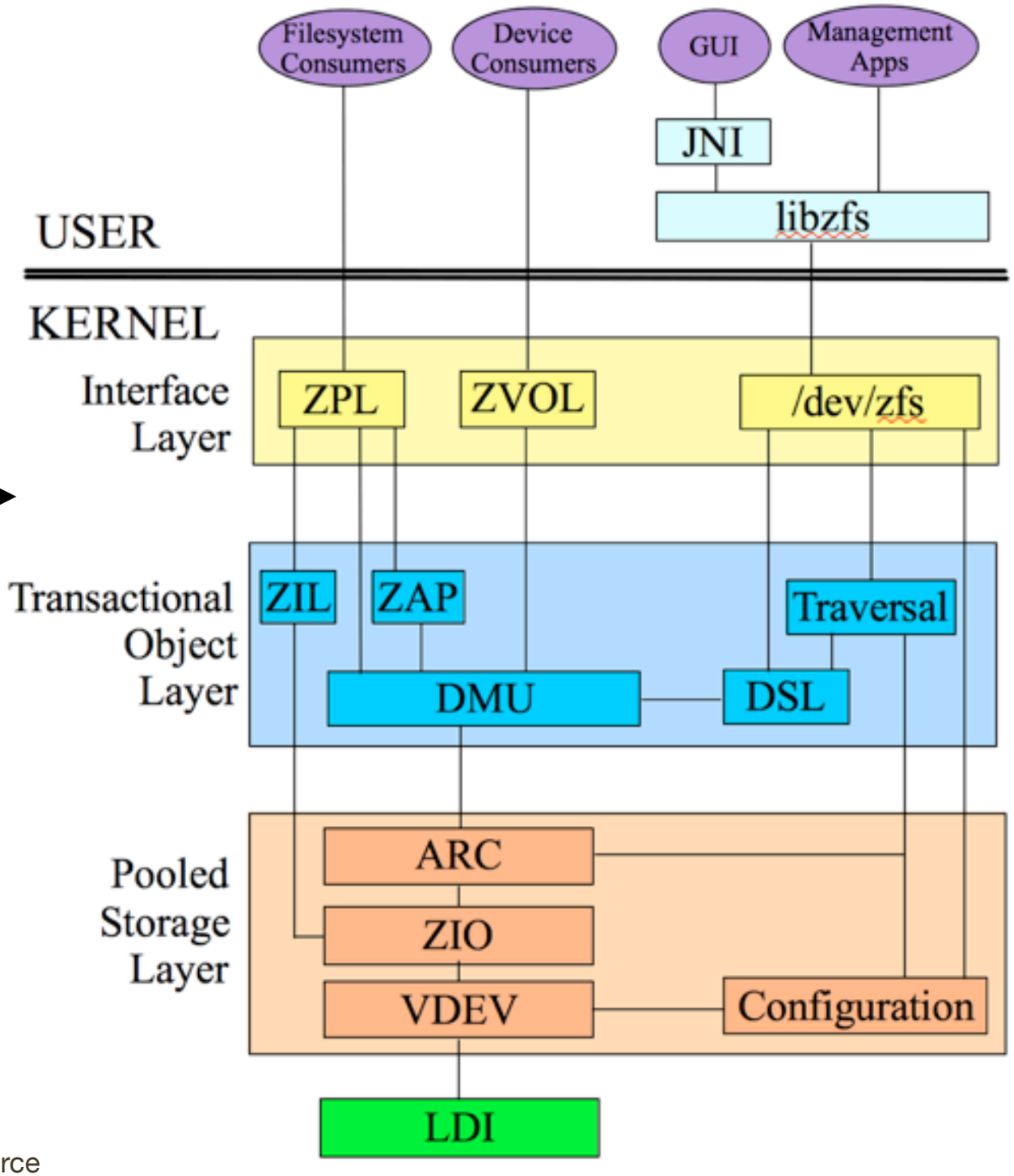
ZFS Internals

Joyent

- That's just my top 12
- Use more as needed

DTRACE ALL

THE THINGS!



http://hub.opensolaris.org/bin/view/Community+Group+zfs/source
http://hyperboleandahalf.blogspot.com/2010/06/this-is-why-ill-never-be-adult.html

iostat

Joyent

- Block-device level (almost disk-level) I/O statistics:

```
$ iostat -xnz 1
[...]
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
    0.0   11.0    0.0   52.0  0.0  0.0    0.0    1.0   0   1 c0t0d0
    1.0  381.0   16.0 43325.5  0.0  4.0    0.0   10.4   1  12 c0t1d0
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
    0.0    9.0    0.0   34.0  0.0  0.0    0.0    0.1   0   0 c0t0d0
    1.0  154.9   16.0 1440.5  0.0  2.0    0.0   12.6   0  10 c0t1d0
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
    0.0    8.0    0.0   36.0  0.0  0.0    0.0    0.0   0   0 c0t0d0
    6.0    0.0   96.0    0.0  0.0  0.0    0.0    7.9   0   4 c0t1d0
```

ZFS->Disk Workload | Disk Resulting Performance

iostat, cont.

- Effective tool for a class of disk issues, especially:

```
$ iostat -xnz 1
[...]
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
    0.0    0.0    0.0    0.0  0.0  4.0    0.0    0.0   0 100 c0t0d0
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
    0.0    0.0    0.0    0.0  0.0  4.0    0.0    0.0   0 100 c0t0d0
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
    0.0    0.0    0.0    0.0  0.0  4.0    0.0    0.0   0 100 c0t0d0
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
    0.0    0.0    0.0    0.0  0.0  4.0    0.0    0.0   0 100 c0t0d0
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
    0.0    0.0    0.0    0.0  0.0  4.0    0.0    0.0   0 100 c0t0d0
```

- Disks “out to lunch” (PERC ECC error)

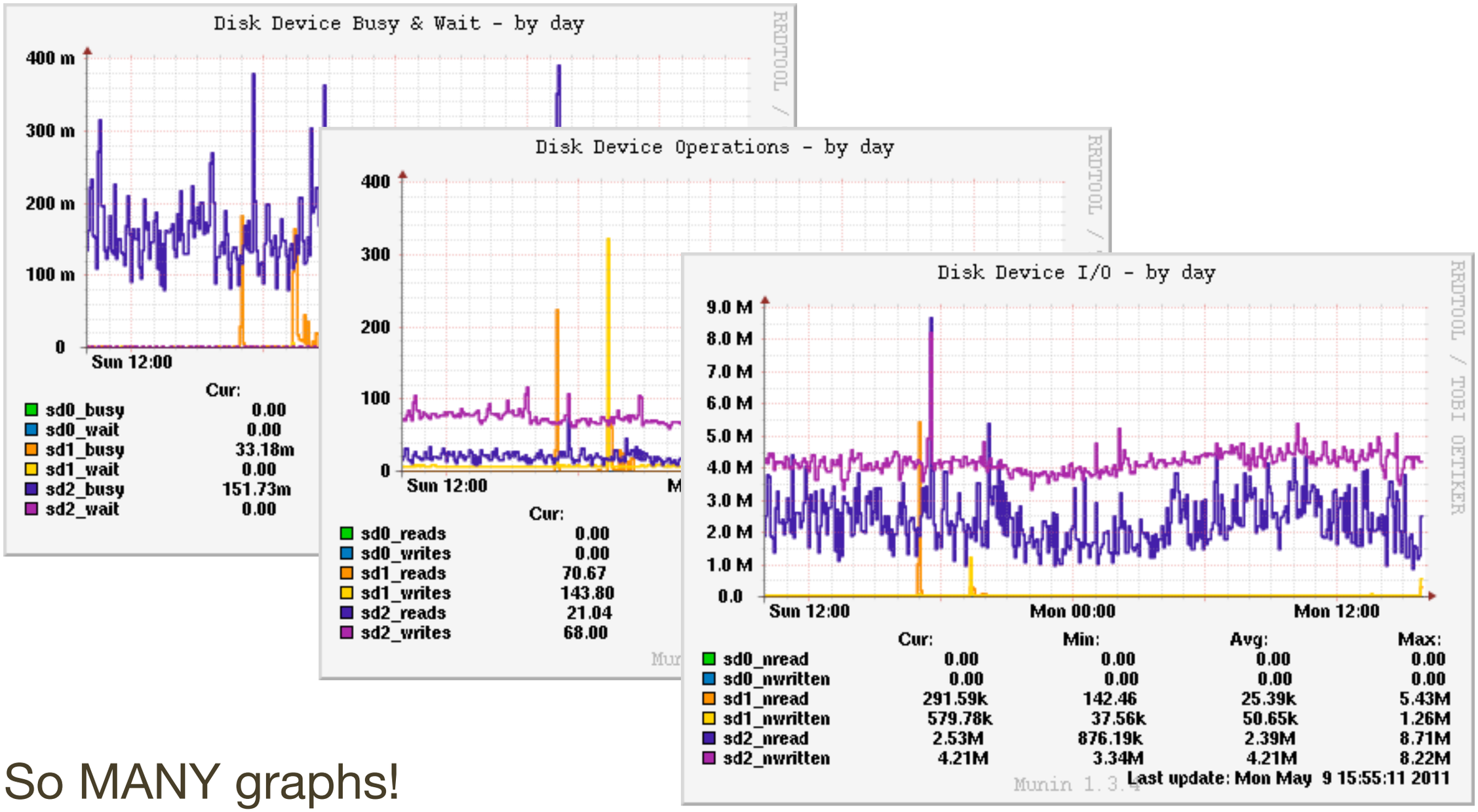
iostat, cont.

Joyent

- Minor nits:
  - does not show read/write latency separately. ZFS TXG flushes drag up the latency, which looks alarming, but are asynchronous. Can use DTrace for the split.
  - no higher level context: PID, ZFS dataset, file pathname, ... (not its role)
- Major problem (although, not iostat's fault): commonly confused with application-level (logical) I/O.
  - The I/O rates, sizes, and latency, can dramatically differ between logical file system I/O and physical disk I/O.
  - Users commonly draw the wrong conclusions when only provided with iostat statistics to understand a system.

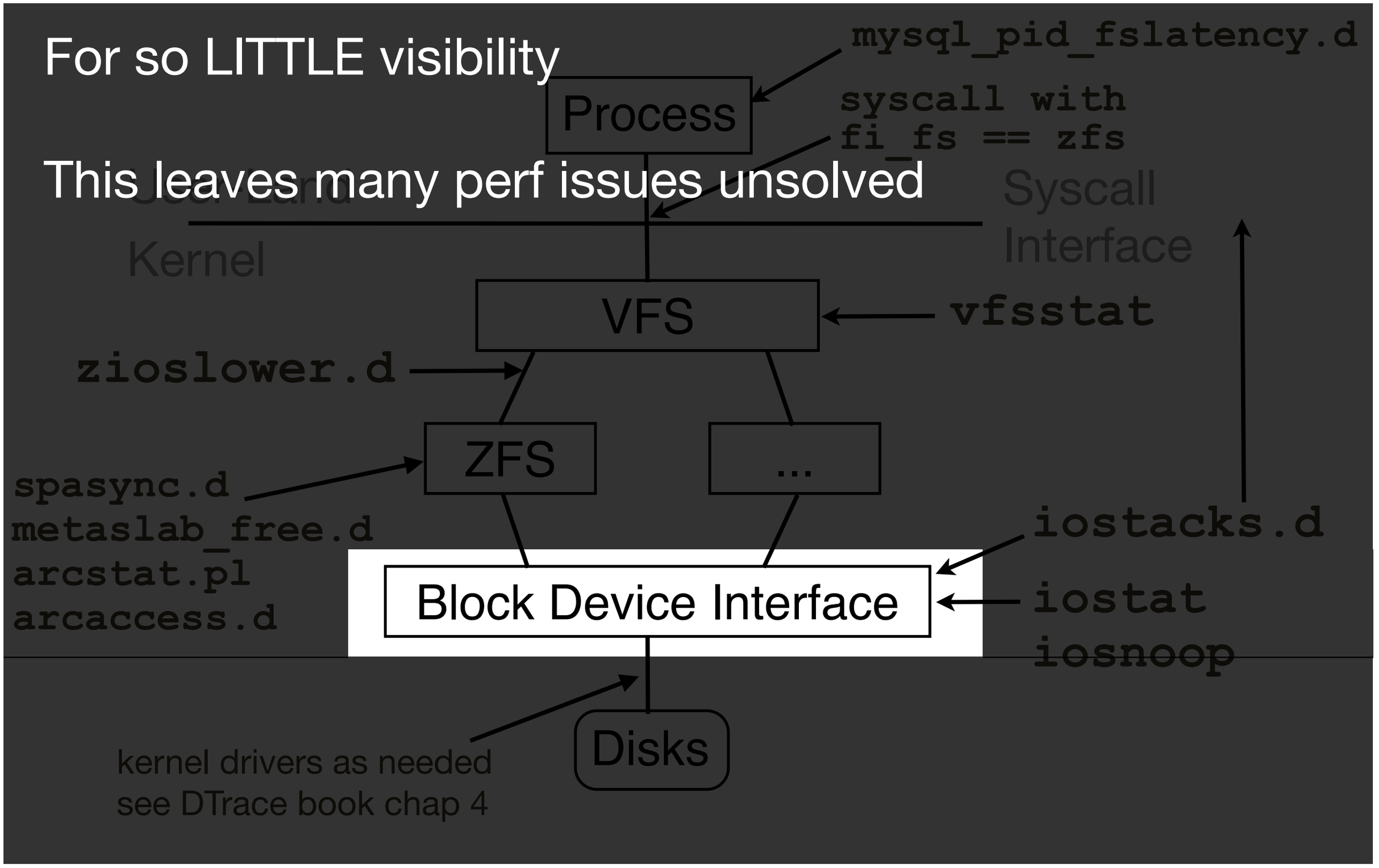
iostat, cont.

- iostat output (or disk kstats) graphed by various monitoring software



So MANY graphs!

iostat, cont.


For so LITTLE visibility
This leaves many perf issues unsolved
mysql_pid_fslatency.d
Process
syscall with
fi_fs == zfs
User Land
Kernel
Syscall
Interface
VFS
vfsstat
zioslower.d
ZFS
...
spasync.d
metaslab_free.d
arcstat.pl
arcaccess.d
iostacks.d
Block Device Interface
iostat
iosnoop
Disks
kernel drivers as needed
see DTrace book chap 4

vfsstat

Joyent

- VFS-level I/O statistics (VFS-iostat):

```
# vfsstat -Z 1
  r/s   w/s  kr/s  kw/s ractv wactv read_t writ_t  %r  %w   d/s  del_t zone
  1.2   2.8   0.6   0.2   0.0   0.0    0.0    0.0   0   0   0.0    0.0 global (0)
  0.1   0.0   0.1   0.0   0.0   0.0    0.0    0.0   0   0   0.0   34.9 9cc2d0d3 (2)
  0.1   0.0   0.1   0.0   0.0   0.0    0.0    0.0   0   0   0.0   46.5 72188ca0 (3)
  0.0   0.0   0.0   0.0   0.0   0.0    0.0    0.0   0   0   0.0   16.5 4d2a62bb (4)
  0.3   0.1   0.1   0.3   0.0   0.0    0.0    0.0   0   0   0.0   27.6 8bbc4000 (5)
  5.9   0.2   0.5   0.1   0.0   0.0    0.0    0.0   0   0   5.0   11.3 d305ee44 (6)
  0.1   0.0   0.1   0.0   0.0   0.0    0.0    0.0   0   0   0.0  132.0 9897c8f5 (7)
  0.1   0.0   0.1   0.0   0.0   0.0    0.0    0.1   0   0   0.0   40.7 5f3c7d9e (9)
  0.2   0.8   0.5   0.6   0.0   0.0    0.0    0.0   0   0   0.0   31.9 22ef87fc (10)
```

App->ZFS Workload | ZFS Resulting Performance | ZFS I/O Throttling

vfsstat, cont.

Joyent

- Good high-level summary of logical I/O: application FS workload
- Summarizes by zone
  - Impetus was observability for cloud “noisy neighbors”
  - Shows affect of ZFS I/O throttling (performance isolation)
- Summarizes performance applications actually experience!
  - Usually a lot better than disk-level, due to ZFS caching (ARC, L2ARC) and buffering
- Required kernel changes, new kstats (thanks Bill Pijewski)

zfsslower.d

Joyent

- ZFS reads/writes slower than 10 ms:

```
# ./zfsslower.d 10
TIME                  PROCESS  D    KB   ms FILE
2012 Sep 30 04:56:22 beam.smp R     2   12 /var/db/riak/leveldb/.../205788.sst
2012 Sep 30 04:56:23 beam.smp R     4   15 /var/db/riak/leveldb/.../152831.sst
2012 Sep 30 04:56:24 beam.smp R     3   11 /var/db/riak/leveldb/.../220432.sst
2012 Sep 30 04:56:24 beam.smp R     2   12 /var/db/riak/leveldb/.../208619.sst
2012 Sep 30 04:56:25 beam.smp R     0   21 /var/db/riak/leveldb/.../210710.sst
2012 Sep 30 04:56:25 beam.smp R     2   18 /var/db/riak/leveldb/.../217757.sst
2012 Sep 30 04:56:25 beam.smp R     2   13 /var/db/riak/leveldb/.../191942.sst
2012 Sep 30 04:56:26 cat       R    5   10 /db/run/beam.smp.pid
2012 Sep 30 04:56:26 beam.smp R     2   11 /var/db/riak/leveldb/.../220389.sst
2012 Sep 30 04:56:27 beam.smp R     2   12 /var/db/riak/leveldb/.../186749.sst
[...]
```

- Traces at VFS level to show the true application suffered I/O time
  - allows immediate confirm/deny of FS (incl. disk) based issue

zfsslower.d

Joyent

- ZFS reads/writes slower than 100 ms:

```
# ./zfsslower.d 100
TIME                  PROCESS  D    KB   ms FILE
2012 Sep 30 05:01:17 beam.smp R     2  144 /var/db/riak/leveldb/.../238108.sst
2012 Sep 30 05:01:54 beam.smp R     1  149 /var/db/riak/leveldb/.../186222.sst
2012 Sep 30 05:02:35 beam.smp R     2  188 /var/db/riak/leveldb/.../200051.sst
2012 Sep 30 05:02:35 beam.smp R     2  159 /var/db/riak/leveldb/.../209376.sst
2012 Sep 30 05:02:35 beam.smp R     1  178 /var/db/riak/leveldb/.../203436.sst
2012 Sep 30 05:02:40 beam.smp R     1  172 /var/db/riak/leveldb/.../204688.sst
2012 Sep 30 05:03:11 beam.smp R     0  200 /var/db/riak/leveldb/.../219837.sst
2012 Sep 30 05:03:38 beam.smp R     1  142 /var/db/riak/leveldb/.../222443.sst
[...]
```

less frequent

zfsslower.d, cont.

Joyent

- All ZFS read/writes:

```
# ./zfsslower.d
TIME                     PROCESS  D    KB    ms FILE
2012 Sep 30 04:46:09 beam.smp R     2     0 /var/db/riak/leveldb/.../221844.sst
2012 Sep 30 04:46:09 beam.smp R     2     0 /var/db/riak/leveldb/.../221155.sst
2012 Sep 30 04:46:09 beam.smp R     2     0 /var/db/riak/leveldb/.../215917.sst
2012 Sep 30 04:46:09 beam.smp R     1     0 /var/db/riak/leveldb/.../190134.sst
2012 Sep 30 04:46:09 beam.smp R     3     0 /var/db/riak/leveldb/.../234539.sst
2012 Sep 30 04:46:09 nginx    W     0     0 /db/log/apps/prod17_nginx_access.log
2012 Sep 30 04:46:09 beam.smp R     4     0 /var/db/riak/leveldb/.../205368.sst
2012 Sep 30 04:46:09 beam.smp R     2     1 /var/db/riak/leveldb/.../199665.sst
2012 Sep 30 04:46:09 beam.smp R     0     0 /var/db/riak/leveldb/.../177866.sst
2012 Sep 30 04:46:09 beam.smp R    56     0 /var/db/riak/leveldb/.../177866.sst
2012 Sep 30 04:46:09 beam.smp R     0     0 /var/db/riak/leveldb/.../177866.sst
2012 Sep 30 04:46:09 beam.smp R     2     0 /var/db/riak/leveldb/.../177866.sst
2012 Sep 30 04:46:09 beam.smp R     5     0 /var/db/riak/leveldb/.../218473.sst
2012 Sep 30 04:46:09 beam.smp R     2     0 /var/db/riak/leveldb/.../210712.sst
2012 Sep 30 04:46:09 beam.smp R     3     0 /var/db/riak/leveldb/.../234194.sst
2012 Sep 30 04:46:09 beam.smp R     0     0 /var/db/riak/leveldb/.../233302.sst
2012 Sep 30 04:46:09 beam.smp R    41     0 /var/db/riak/leveldb/.../233302.sst
2012 Sep 30 04:46:09 beam.smp R     0     0 /var/db/riak/leveldb/.../233302.sst
[...]
```

zfsslower.d, cont.

Joyent

- Written in DTrace

```
[...]
fbt::zfs_read:entry,
fbt::zfs_write:entry
{
        self->path = args[0]->v_path;
        self->kb = args[1]->uio_resid / 1024;
        self->start = timestamp;
}

fbt::zfs_read:return,
fbt::zfs_write:return
/self->start && (timestamp - self->start) >= min_ns/
{
        this->iotime = (timestamp - self->start) / 1000000;
        this->dir = probefunc == "zfs_read" ? "R" : "W";
        printf("%-20Y %-16s %1s %4d %6d %s\n", walltimestamp,
            execname, this->dir, self->kb, this->iotime,
            self->path != NULL ? stringof(self->path) : "<null>");
}
[...]
```

- zfsslower.d, also on github, originated from the DTrace book

zfsslower.d, cont.

Joyent

- Traces VFS/ZFS interface (kernel) from usr/src/uts/common/fs/zfs/zfs_vnops.c:

```
/*
 * Regular file vnode operations template
 */
vnodeops_t *zfs_fvnodeops;
const fs_operation_def_t zfs_fvnodeops_template[] = {
        VOPNAME_OPEN,           { .vop_open = zfs_open },
        VOPNAME_CLOSE,          { .vop_close = zfs_close },
        VOPNAME_READ,           { .vop_read = zfs_read },
        VOPNAME_WRITE,          { .vop_write = zfs_write },
        VOPNAME_IOCTL,          { .vop_ioctl = zfs_ioctl },
        VOPNAME_GETATTR,        { .vop_getattr = zfs_getattr },
    [...]
```

iosnoop

Joyent

- Traces block-device-level I/O:

```
# ./iosnoop
  UID    PID D      BLOCK   SIZE       COMM PATHNAME
  103   5891 R 238517878 131072   beam.smp <none>
  103   5891 R 544800120 131072   beam.smp <none>
  103   5891 R 286317766 131072   beam.smp <none>
  103   5891 R  74515841 131072   beam.smp <none>
  103   5891 R  98341339 131072   beam.smp <none>
[...]
```

- Has many options:

```
# ./iosnoop -Dots
STIME(us)      TIME(us)       DELTA(us)  DTIME(us)  UID   PID D     BLOCK   SIZE       COMM PATHNAME
787809430864   787809435385   4520       4539       103   5891 R 128302372 131072   beam.smp <none>
787809472876   787809478812   5935       5955       103   5891 R 143783200 131072   beam.smp <none>
787809479681   787809483470   3788       3809       103   5891 R  84913822 131072   beam.smp <none>
787809484451   787809490703   6251       6266       103   5891 R  14964144 131072   beam.smp <none>
787809489555   787809497167   7611       6463       103   5891 R 283398320 131072   beam.smp <none>
787809491122   787809498010   6888       842        103   5891 R 288468148 131072   beam.smp <none>
[...]
```

iosnoop, cont.

Joyent

- Written in DTrace

```
[...]
 io:genunix::done
 /start_time[args[0]->b_edev, args[0]->b_blkno]/
 {
[...]

        /* fetch entry values */
        this->dev = args[0]->b_edev;
        this->blk = args[0]->b_blkno;
        this->suid = start_uid[this->dev, this->blk];
        this->spid = start_pid[this->dev, this->blk];
        this->sppid = start_ppid[this->dev, this->blk];
        self->sargs = (int)start_args[this->dev, this->blk] == 0 ?
            "" : start_args[this->dev, this->blk];
        self->scomm = start_comm[this->dev, this->blk];
[...]
            printf("%5d %5d %1s %8d %6d ",
            this->suid, this->spid, args[0]->b_flags & B_READ ? "R" : "W",
            args[0]->b_blkno, args[0]->b_bcount);

[...]
```

- From the DTraceToolkit

iosnoop, cont.

Joyent

- 370 lines of code, mostly to process options:

```
USAGE: iosnoop [-a|-A|-DeghiNostv] [-d device] [-f filename]
               [-m mount_point] [-n name] [-p PID]
       iosnoop                 # default output
                -a             # print all data (mostly)
                -A             # dump all data, space delimited
                -D             # print time delta, us (elapsed)
                -e             # print device name
                -g             # print command arguments
                -i             # print device instance
                -N             # print major and minor numbers
                -o             # print disk delta time, us
                -s             # print start time, us
                -t             # print completion time, us
                -v             # print completion time, string
                -d device      # instance name to snoop
                -f filename    # snoop this file only
                -m mount_point # this FS only
                -n name        # this process name only
                -p PID         # this PID only
  eg,
       iosnoop -v              # human readable timestamps
       iosnoop -N              # print major and minor numbers
       iosnoop -m /            # snoop events on filesystem / only
```

iosnoop, cont.

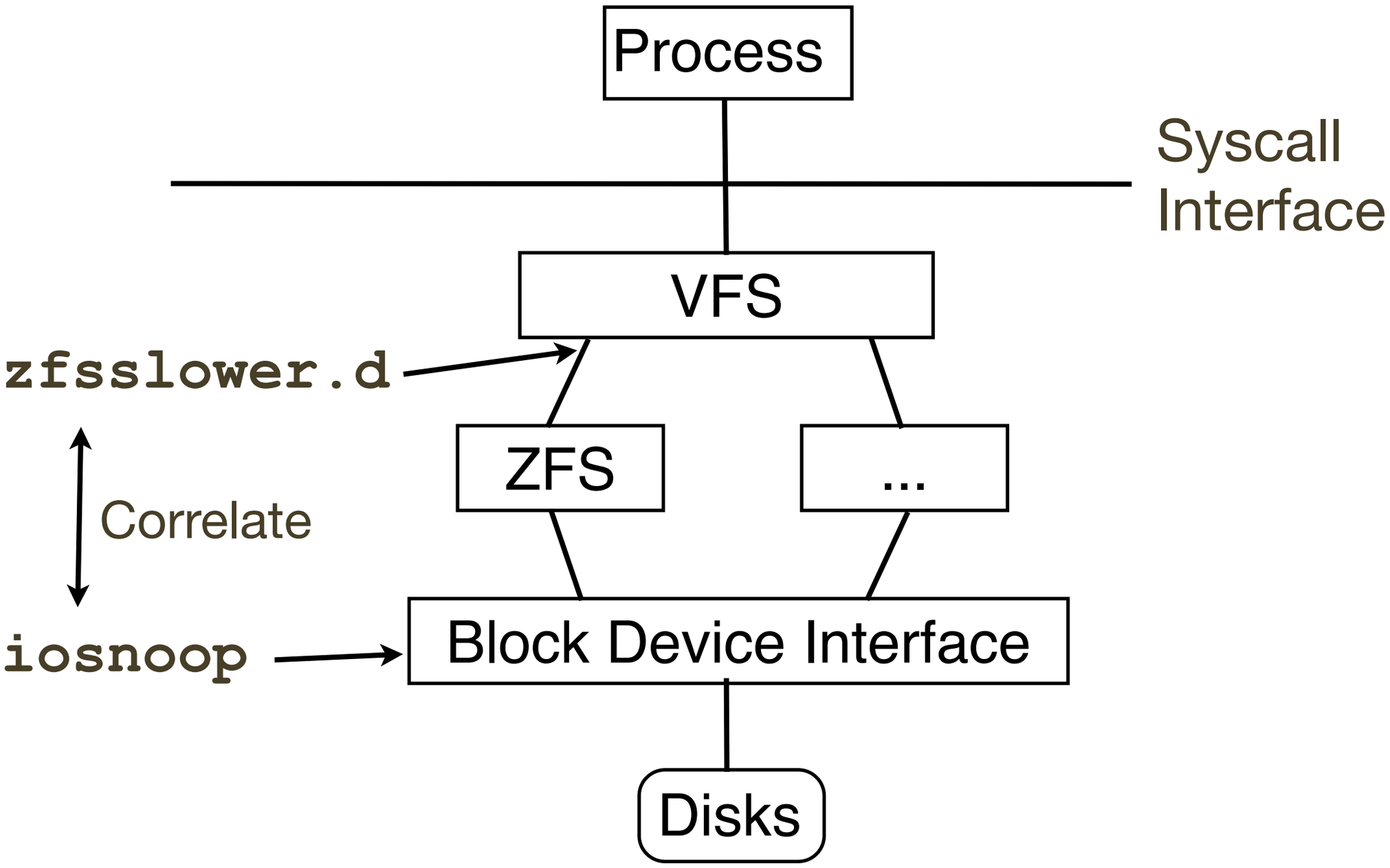
Joyent

- I originally wrote it in 2004 to solve disk-I/O-by-process
  - Shipped by default on Mac OS X (/usr/bin), Oracle Solaris 11
  - I wrote a companion called iotop. A similar iotop tool later appeared for Linux (via the blk tracing framework)
- Dosen't do ZFS pathnames yet (known issue; hackathon?)
  - Pathnames from the block-device layer was always a party trick: relied on a cached vnode->v_path
  - ZFS aggregates I/O. At the block-device layer, no one vnode responsible (or vnode pointer)

iosnoop, cont.

Joyent

- Locate latency origin:



iostacks.d

Joyent

- Just a one-liner really (could make an “iostacks.d”):

```
# dtrace -n 'io:::start { @[stack()] = count(); }'
dtrace: description 'io:::start ' matched 6 probes
^C
              genunix`ldi_strategy+0x53
              zfs`vdev_disk_io_start+0xcc
              zfs`zio_vdev_io_start+0xab
              zfs`zio_execute+0x88
              zfs`zio_nowait+0x21
              zfs`vdev_mirror_io_start+0xcd
              zfs`zio_vdev_io_start+0x250
              zfs`zio_execute+0x88
              zfs`zio_nowait+0x21
              zfs`arc_read_nolock+0x4f9
              zfs`arc_read+0x96
              zfs`dsl_read+0x44
              zfs`dbuf_read_impl+0x166
              zfs`dbuf_read+0xab
              zfs`dmu_buf_hold_array_by_dnode+0x189
              zfs`dmu_buf_hold_array+0x78
              zfs`dmu_read_uio+0x5c
              zfs`zfs_read+0x1a3
              genunix`fop_read+0x8b
              genunix`read+0x2a7
              143
```

iostacks.d, cont.

Joyent

- Stack recognition chart:

```
# dtrace -n 'io:::start { @[stack()] = count(); }'
dtrace: description 'io:::start ' matched 6 probes
^C
              genunix`ldi_strategy+0x53
              zfs`vdev_disk_io_start+0xcc
              zfs`zio_vdev_io_start+0xab
              zfs`zio_execute+0x88
              zfs`zio_nowait+0x21
              zfs`vdev_mirror_io_start+0xcd
              zfs`zio_vdev_io_start+0x250
              zfs`zio_execute+0x88
              zfs`zio_nowait+0x21
              zfs`arc_read_nolock+0x4f9
              zfs`arc_read+0x96
              zfs`dsl_read+0x44
              zfs`dbuf_read_impl+0x166
              zfs`dbuf_read+0xab
              zfs`dmu_buf_hold_array_by_dnode+0x189
              zfs`dmu_buf_hold_array+0x78
              zfs`dmu_read_uio+0x5c
              zfs`zfs_read+0x1a3
              genunix`fop_read+0x8b
              genunix`read+0x2a7
              143
```

syscall ← read() arc-miss

iostacks.d, cont.

Joyent

- Stack recognition chart:

```
# dtrace -n 'io:::start { @[stack()] = count(); }'
dtrace: description 'io:::start ' matched 6 probes
^C
              genunix`ldi_strategy+0x53
              zfs`vdev_disk_io_start+0xcc
              zfs`zio_vdev_io_start+0xab
              zfs`zio_execute+0x88
              zfs`vdev_queue_io_done+0x70
              zfs`zio_vdev_io_done+0x80
              zfs`zio_execute+0x88
              genunix`taskq_thread+0x2d0
              unix`thread_start+0x8
              395
```

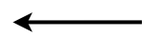
← ZIO pipeline

iostacks.d, cont.

Joyent

- From zio, when parent == NULL (first zio):

```
# dtrace -n 'zio_create:entry /arg0 == NULL/ { @[stack()] = count(); }'
[...]
              zfs`zio_null+0x77
              zfs`zio_root+0x2d
              zfs`dmu_buf_hold_array_by_dnode+0x113
              zfs`dmu_buf_hold_array+0x78
              zfs`dmu_write+0x80
              zfs`space_map_sync+0x288
              zfs`metaslab_sync+0x135
              zfs`vdev_sync+0x7f
              zfs`spa_sync+0x38b
              zfs`txg_sync_thread+0x204
              unix`thread_start+0x8
                7
```



SPA sync

iostacks.d, cont.

Joyent

- Can identify the reason for disk I/O
  - including unexplained additional I/O from ZFS
- Times it doesn't work (ZIO pipeline task pool), can try a similar one-liner from fbt::zio_create:entry, for the stack trace from the creation of all ZIO.

metaslab_free.d

Joyent

- Traces ZFS metaslab details:

```
# ./metaslab_free.d
Tracing ZFS metaslab alloc.  metaslab_df_free_pct = 4 %

2012 Sep 30 05:32:47 free %pct by allocations:
                13                 12
                89                 34
                81                287
                12                389

2012 Sep 30 05:32:49 free %pct by allocations:
                89                 15
                81                265
                12                378
```

- ZFS pools are split into metaslabs (eg, 10 GB each)
- Metaslabs have two allocation algorithms:

  metaslab %free >= metaslab_df_free_pct == first fit (fast)
  metaslab %free < metaslab_df_free_pct == best fit (slow)

metaslab_free.d, cont.

Joyent

- Written in DTrace

```
dtrace:::BEGIN
{
        printf("Tracing ZFS metaslab alloc.  metaslab_df_free_pct = %d %%\n",
            `metaslab_df_free_pct);
}

fbt::metaslab_df_alloc:entry
{
        this->pct = args[0]->sm_space * 100 / args[0]->sm_size;
        @[this->pct] = count();
}

profile:::tick-1s
{
        printf("\n%Y free %%pct by allocations:", walltimestamp);
        printa(@);
        trunc(@);
}
```

- metaslab_free.d is also on github

metaslab_free.d, cont.

Joyent

- Shows if allocations are currently fast fit or best fit
- Correlate to performance changes
- Use while tuning
    - Can’t just try out a new setting and watch performance: perf can improve if ZFS switched to a new, emptier, metaslab at the same time. Script identifies if that occurred.

spasync.d

Joyent

- Traces ZFS SPA syncs:

```
# ./spasync.d
Tracing ZFS spa_sync() slower than 1 ms...
2012 Sep 30 05:56:07 zones          30 ms, 25 MB 464 I/O
2012 Sep 30 05:56:12 zones          22 ms, 15 MB 366 I/O
2012 Sep 30 05:56:17 zones          39 ms, 19 MB 361 I/O
2012 Sep 30 05:56:22 zones         143 ms, 31 MB 536 I/O
2012 Sep 30 05:56:27 zones          32 ms, 27 MB 465 I/O
2012 Sep 30 05:56:32 zones          27 ms, 24 MB 468 I/O
2012 Sep 30 05:56:37 zones          31 ms, 23 MB 489 I/O
2012 Sep 30 05:56:42 zones         200 ms, 85 MB 956 I/O
2012 Sep 30 05:56:47 zones         122 ms, 46 MB 593 I/O
2012 Sep 30 05:56:52 zones          48 ms, 18 MB 394 I/O
2012 Sep 30 05:56:57 zones          43 ms, 16 MB 409 I/O
2012 Sep 30 05:57:02 zones          33 ms, 27 MB 526 I/O
[...]
```

- Check for correlations with I/O latency

spasync.d, cont.

Joyent

- Written in DTrace

```
[...]
fbt::spa_sync:entry
/!self->start/
{
        in_spa_sync = 1;
        self->start = timestamp;
        self->spa = args[0];
}

io:::start
/in_spa_sync/
{
        @io = count();
        @bytes = sum(args[0]->b_bcount);
}

fbt::spa_sync:return
/self->start && (this->ms = (timestamp - self->start) / 1000000) > MIN_MS/
{
        normalize(@bytes, 1048576);
        printf("%-20Y %-10s %6d ms, ", walltimestamp,
            stringof(self->spa->spa_name), this->ms);
        printa("%@d MB %@d I/O\n", @bytes, @io);
}
[...]
```

- spasync.d is also on github

spasync.d, cont.

Joyent

- Simple and effective: helps diagnose TXG sync issues, providing information to help guide tuning.
- Long heritage: Matt Ahrens -> Roch Bourbannias -> Ben Rockwood -> Brendan Gregg

%iowait

Joyent

- I/O wait percent

```
# mpstat 1
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  0  483   0  427  2216  913 4415    6   79   45    0  9275    7  11   0  83
  1  625   0  153  4171 1748 2887    4   42   37    0  5909    7   8   0  85
  2  269   0  702  1720  636 2943    5   71   39    0  6422    5   8   0  87
  3  281   0  161  1433  529 2255    3   39   35    0  5269    5   5   0  90
  4  173   0  577  3361 1465 2565    3   50   38    0  4645    4   6   0  90
  5  172   0   98  1108  418 1640    2   27   30    0  3699    3   3   0  94
  6  139   0  678  2406  802 2509    4   60   34    0  5135    4   6   0  90
  7  140   0   91  1343  527 2196    2   36   31    0  4879    4   4   0  92
  8  124   0  527  1145  419 1676    2   43   29    0  3335    4   4   0  92
[...]
```

%iowait

Joyent

- I/O wait percent ... still zero in illumos!

```
# mpstat 1
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  0  483   0  427  2216  913 4415    6   79   45    0  9275    7  11   0  83
  1  625   0  153  4171 1748 2887    4   42   37    0  5909    7   8   0  85
  2  269   0  702  1720  636 2943    5   71   39    0  6422    5   8   0  87
  3  281   0  161  1433  529 2255    3   39   35    0  5269    5   5   0  90
  4  173   0  577  3361 1465 2565    3   50   38    0  4645    4   6   0  90
  5  172   0   98  1108  418 1640    2   27   30    0  3699    3   3   0  94
  6  139   0  678  2406  802 2509    4   60   34    0  5135    4   6   0  90
  7  140   0   91  1343  527 2196    2   36   31    0  4879    4   4   0  92
  8  124   0  527  1145  419 1676    2   43   29    0  3335    4   4   0  92
[...]
```

arcstat

Joyent

- ZFS ARC and L2ARC statistics:

```
$ ./arcstat 1
    time  read  miss  miss%  dmis  dm%  pmis  pm%  mmis  mm%  arcsz     c
04:45:47     0     0      0     0    0     0    0     0    0    14G   14G
04:45:49   15K    10      0    10    0     0    0     1    0    14G   14G
04:45:50   23K    81      0    81    0     0    0     1    0    14G   14G
04:45:51   65K    25      0    25    0     0    0     4    0    14G   14G
04:45:52   30K    11      0    11    0     0    0     3    0    14G   14G
[...]
```

- Statistics are per-interval:
  - read, miss: total ARC accesses, misses
  - miss%, dm%, pm%, mm%: ARC miss percentage total, demand, prefetch, metadata
  - dmis, pmis, mmis: misses for demand, prefetch, metadata
  - arcsz, c: ARC size, ARC target size

arcstat, cont.

- Written in Perl, uses kstats (zfs::arcstats:):

```
[...]
sub snap_stats {
        my %prev = %cur;
        if ($kstat->update()) {
                printf("<State Changed>\n");
        }
        my $hashref_cur = $kstat->{"zfs"}{0}{"arcstats"};
        %cur = %$hashref_cur;
        foreach my $key (keys %cur) {
                next if $key =~ /class/;
                if (defined $prev{$key}) {
                        $d{$key} = $cur{$key} - $prev{$key};
[...]
sub calculate {
        %v = ();
        $v{"time"} = strftime("%H:%M:%S", localtime);
        $v{"hits"} = $d{"hits"}/$int;
        $v{"miss"} = $d{"misses"}/$int;
        $v{"read"} = $v{"hits"} + $v{"miss"};
        $v{"hit%"} = 100*$v{"hits"}/$v{"read"} if $v{"read"} > 0;
        $v{"miss%"} = 100 - $v{"hit%"} if $v{"read"} > 0;
[...]
```

- github.com/mharsch/arcstat

arcstat, cont.

Joyent

- Options to configure output, including L2ARC stats
- Crucial data when analyzing cache performance
  (and easier to read than the raw form: kstat -pn arcstats)
- Originally by Neelakanth Nadgir, then Jason Hellenthal (FreeBSD port) and Mike Harsh

arcaccess.d

Joyent

- ARC population age:

```
# ./arcaccess.d -n 'tick-10s { exit(0); }'
lbolt rate is 100 Hertz.
Tracing lbolts between ARC accesses...
```

Age:

```
           value  ------------- Distribution ------------- count
              -1 |                                         0
               0 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  729988
               1 |                                         3805
               2 |                                         3038
               4 |                                         2028
               8 |                                         1428
              16 |                                         1398
              32 |                                         1618
              64 |                                         2883
             128 |                                         738
             256 |                                         681
             512 |                                         338
            1024 |                                         569
            2048 |                                         166
            4096 |                                         607
            8192 |                                         632
           16384 |                                         808
           32768 |                                         373
           65536 |                                         110
          131072 |                                         142
          262144 |                                         39
          524288 |                                         5
         1048576 |                                         97
         2097152 |                                         10
         4194304 |                                         44
         8388608 |                                         617
        16777216 |                                         1
        33554432 |                                         0
```

10 ms

1 second

1 minute

1 hour

1 day

arcaccess.d, cont.

Joyent

- Written in DTrace

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

#pragma D option quiet

dtrace:::BEGIN
{
     printf("lbolt rate is %d Hertz.\n", `hz);
     printf("Tracing lbolts between ARC accesses...");
}

fbt::arc_access:entry
{
     self->ab = args[0];
     self->lbolt = args[0]->b_arc_access;
}

fbt::arc_access:return
/self->lbolt/
{
     @ = quantize(self->ab->b_arc_access - self->lbolt);
     self->ab = 0;
     self->lbolt = 0;
}
```

- http://dtrace.org/blogs/brendan/2012/01/09/activity-of-the-zfs-arc/

arcaccess.d, cont.

Joyent

- Shows population age of the ARC based on access time
  - Helps determine working set size and turn over rate
- Previous example showed ARC was so large, it was able to keep buffers that had not been accessed in over a day.
  - Turn over rate == low, working set size == entirely fits, likely much smaller.

Latency counters

Joyent

- kstat -p zone_vfs:::*_ops, show latency outlier counts since boot:

```
$ kstat -p zone_vfs:::*_ops
zone_vfs:0:global:100ms_ops     13
zone_vfs:0:global:10ms_ops      2220
zone_vfs:0:global:10s_ops       0
zone_vfs:0:global:1s_ops        0
zone_vfs:1:5cedb79e:100ms_ops     173367
zone_vfs:1:5cedb79e:10ms_ops      64071247
zone_vfs:1:5cedb79e:10s_ops       0
zone_vfs:1:5cedb79e:1s_ops        430
```

- reads/writes:

```
$ kstat -p zone_vfs:::reads zone_vfs:::writes
zone_vfs:0:global:reads 666148884
zone_vfs:1:5cedb79e-51c6-41b3-b99d-358b33:reads 1791879457
zone_vfs:0:global:writes        46861078
zone_vfs:1:5cedb79e-51c6-41b3-b99d-358b33:writes        356075500
```

Latency counters

Joyent

- kstat -p zone_vfs:::*_ops, show latency outlier counts since boot:

```
$ kstat -p zone_vfs:::*_ops
zone_vfs:0:global:100ms_ops     13
zone_vfs:0:global:10ms_ops      2220
zone_vfs:0:global:10s_ops       0
zone_vfs:0:global:1s_ops        0
zone_vfs:1:5cedb79e:100ms_ops     173367
zone_vfs:1:5cedb79e:10ms_ops      64071247
zone_vfs:1:5cedb79e:10s_ops       0
zone_vfs:1:5cedb79e:1s_ops        430
```

yikes, 430 ops > 1s!
however, this is a small fraction

- reads/writes:

```
$ kstat -p zone_vfs:::reads zone_vfs:::writes
zone_vfs:0:global:reads 666148884
zone_vfs:1:5cedb79e-51c6-41b3-b99d-358b33:reads 1791879457
zone_vfs:0:global:writes        46861078
zone_vfs:1:5cedb79e-51c6-41b3-b99d-358b33:writes        356075500
```

>1s = 0.00002%

Latency counters, cont.

Joyent

- Added by Bill Pijewski of Joyent
- Proven invaluable, usually to quickly eliminate ZFS (and disk) as a source of issues, *after the fact* (when counters are near-zero)
    - Has had identified real ZFS/disk-level issues as well

Scatter plots

Joyent

```
iosnoop -Dots > outfile
```

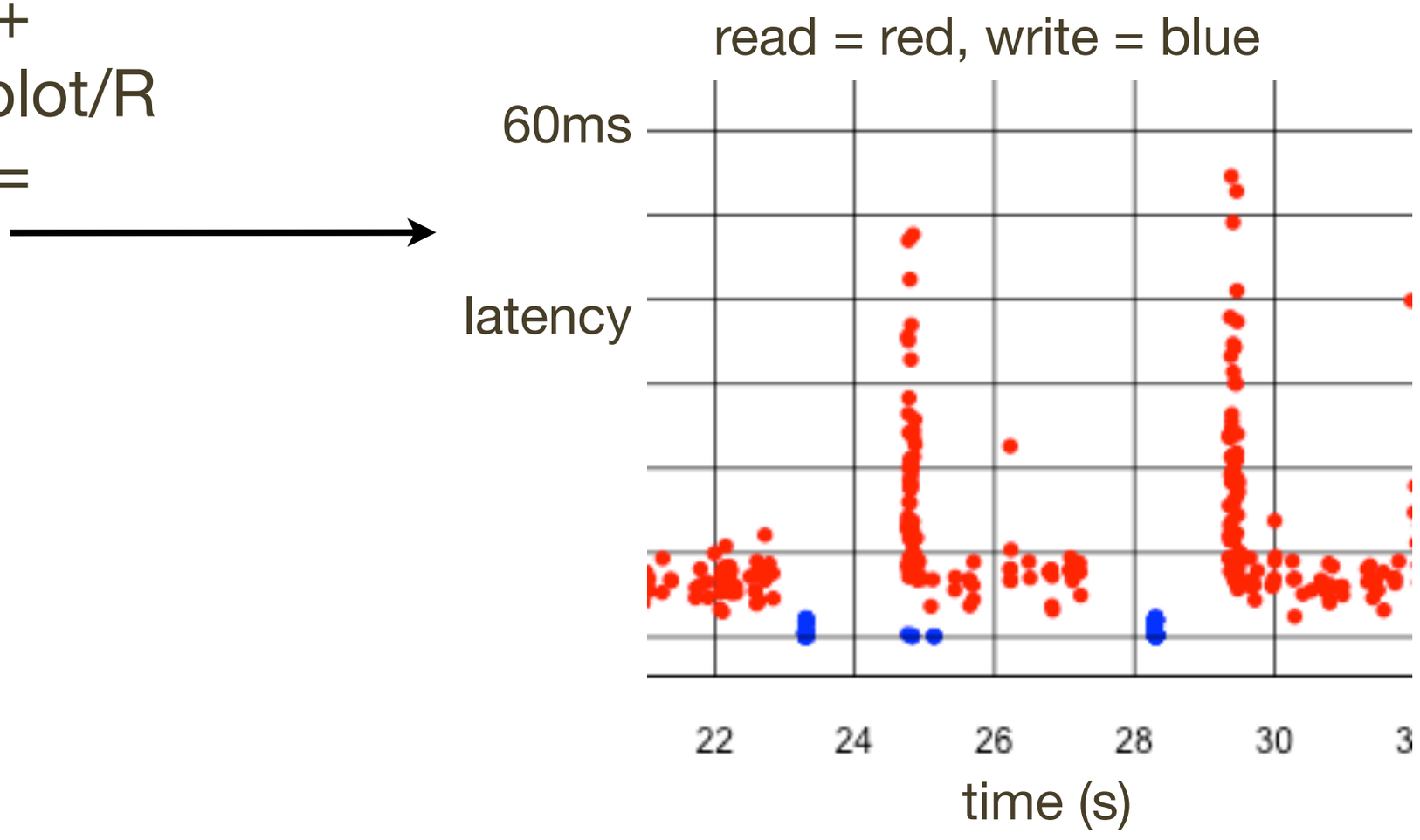
+

awk/perl

+

gnuplot/R

=



Scatter plots, cont.

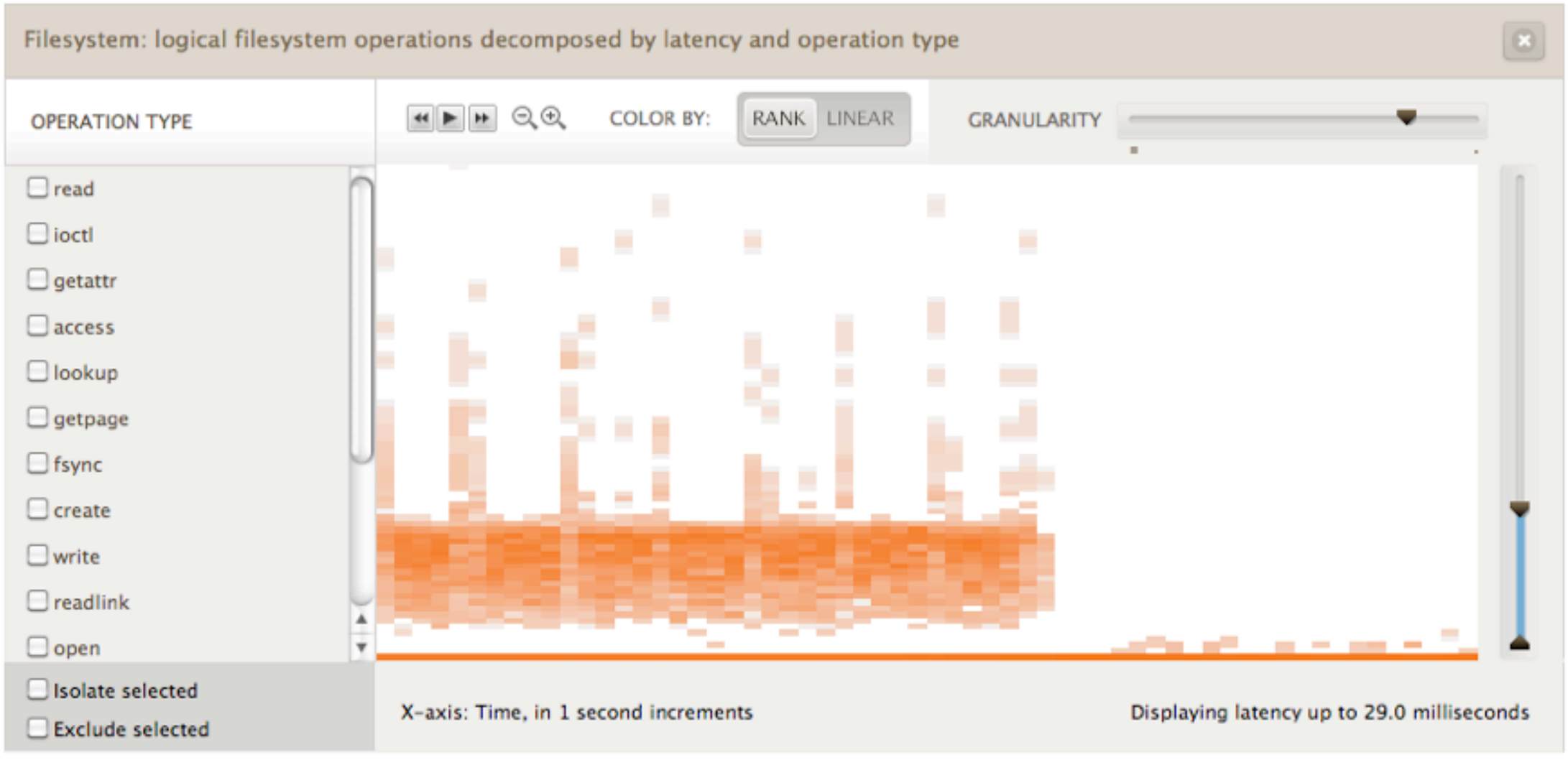
Joyent

- Has been an effective last-resort to investigate nasty PERC issues: reads queueing behind writes
- Has a big data problem: many points, x-y coordinates, takes time & space to capture and render.
    - Heat maps solve this, and can be real-time

Heat maps

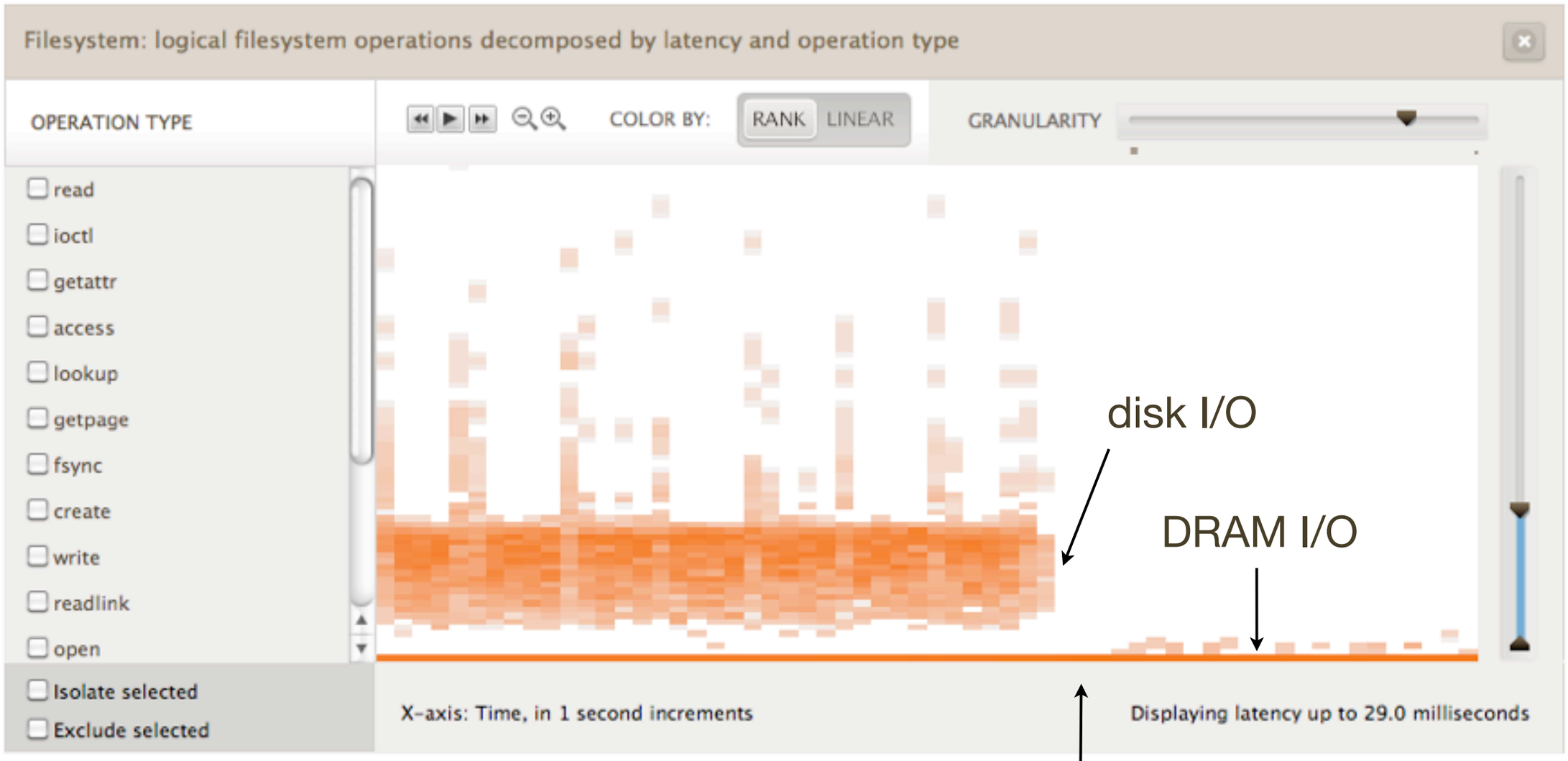
Joyent

- WHAT DOES IT MEAN?



Heat maps

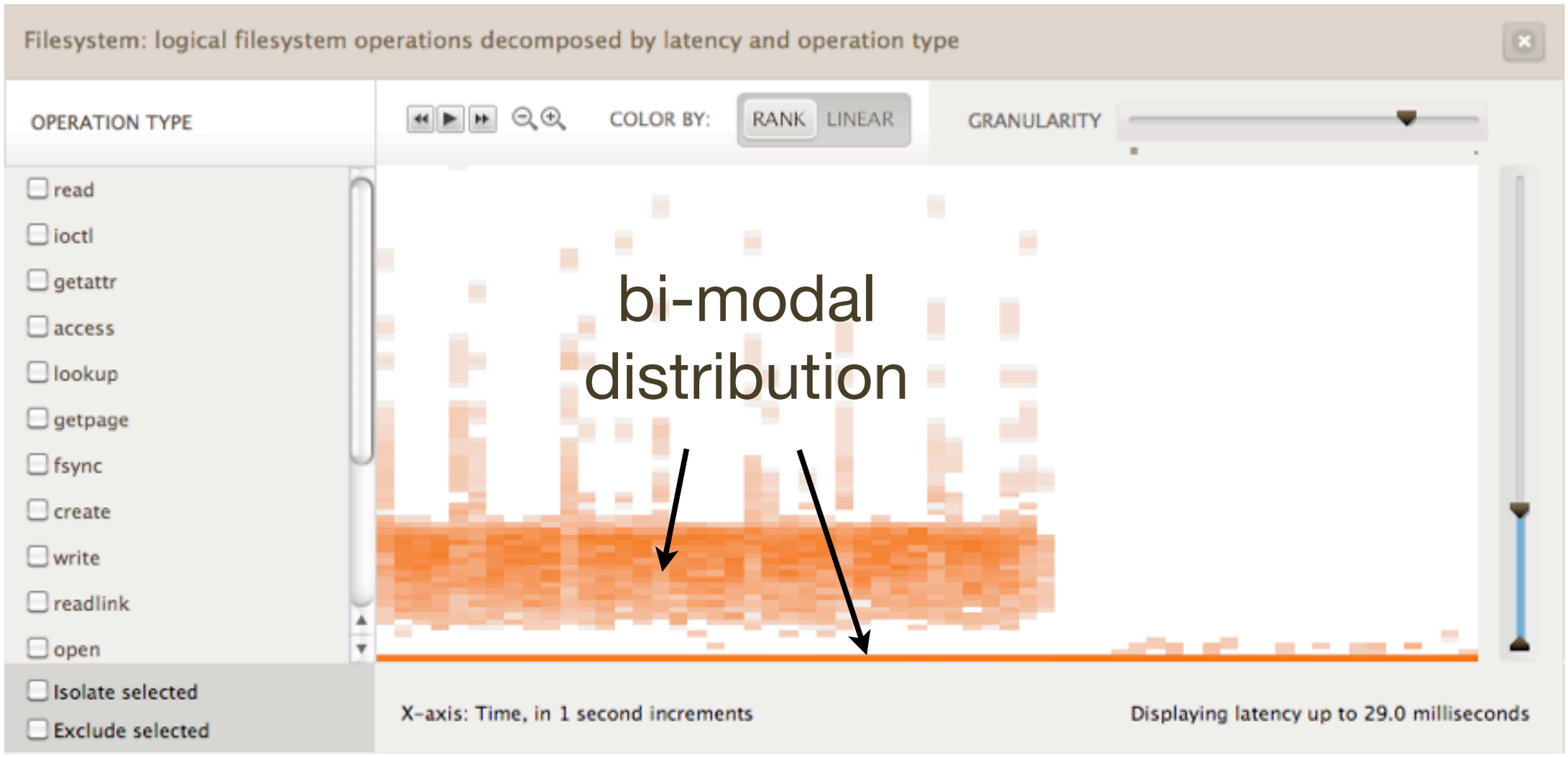
- WHAT DOES IT MEAN?



Heat maps

Joyent

- WHAT DOES IT MEAN?



Heat maps

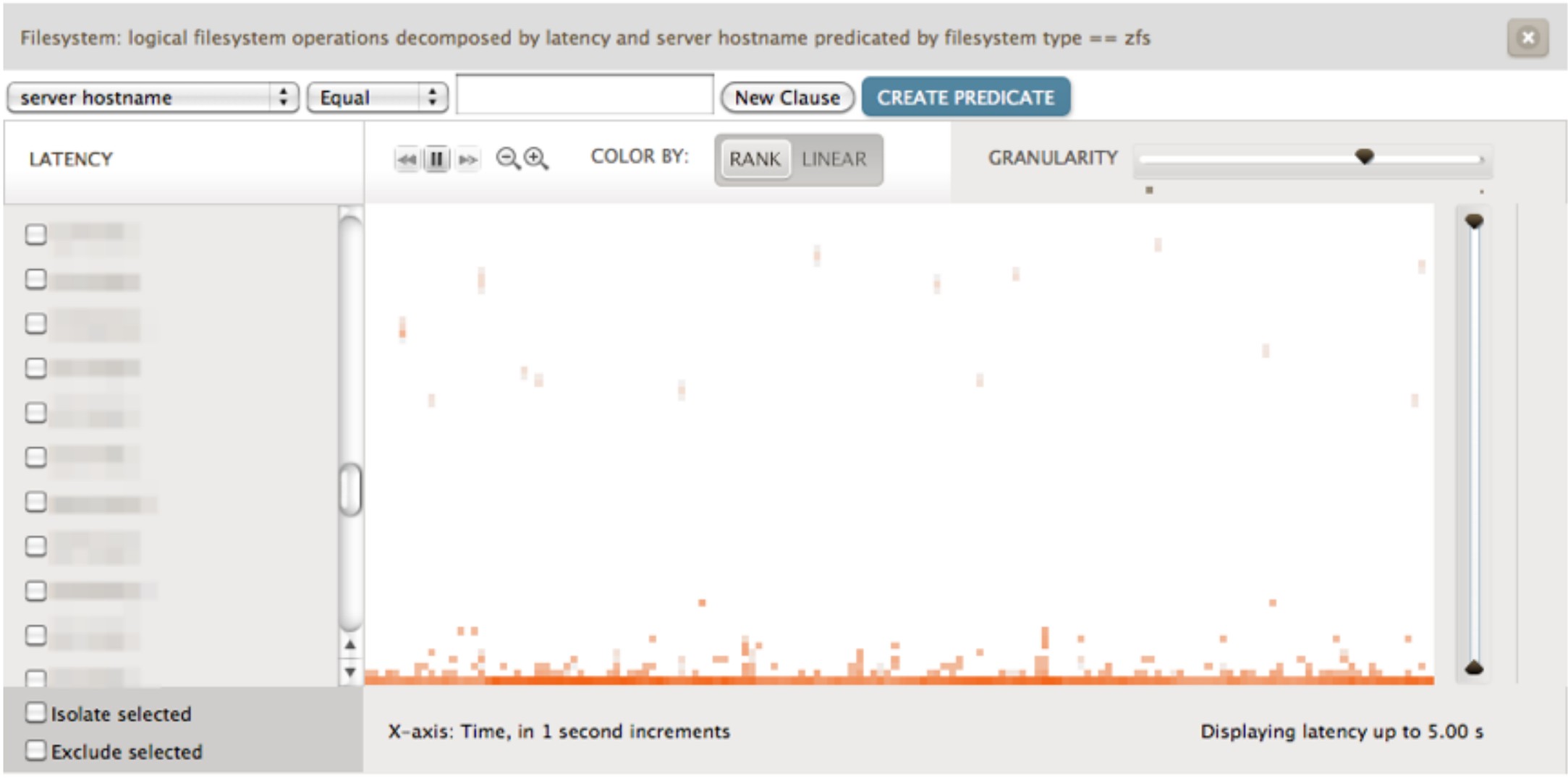
Joyent

- File system latency is commonly bi-modal: cache hits vs misses
- Average latency, stddev, 99th percentile, don't make a lot of sense
    - Given line graphs of these, we would not be able to solve many problems
- Need to see the full distribution: heat maps
    - Also good at identifying outliers

Heat maps, cont.

Joyent

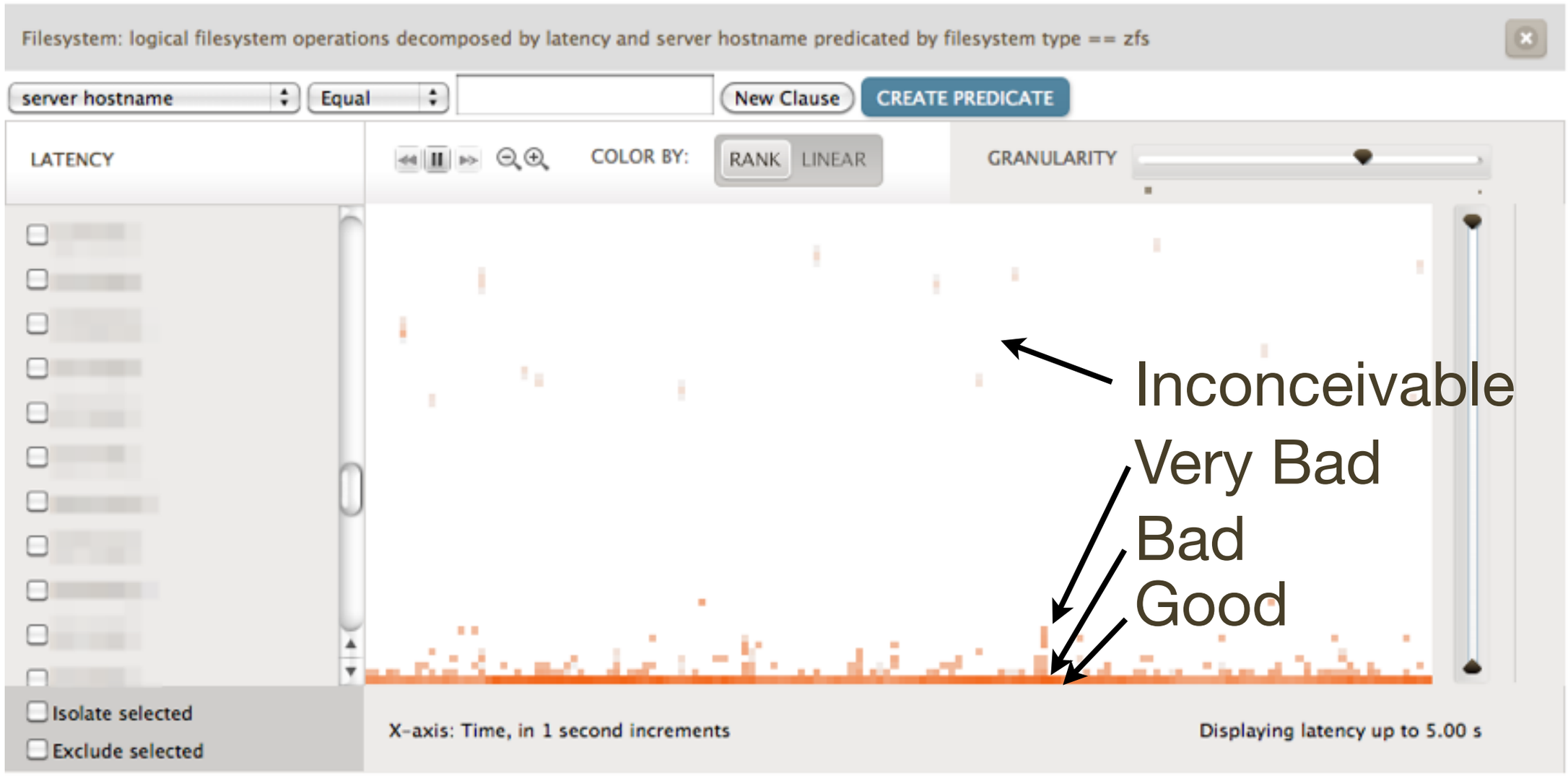
- Latency outliers, cloud wide (entire datacenter):



Heat maps, cont.

Joyent

- Latency outliers, cloud wide (entire datacenter):



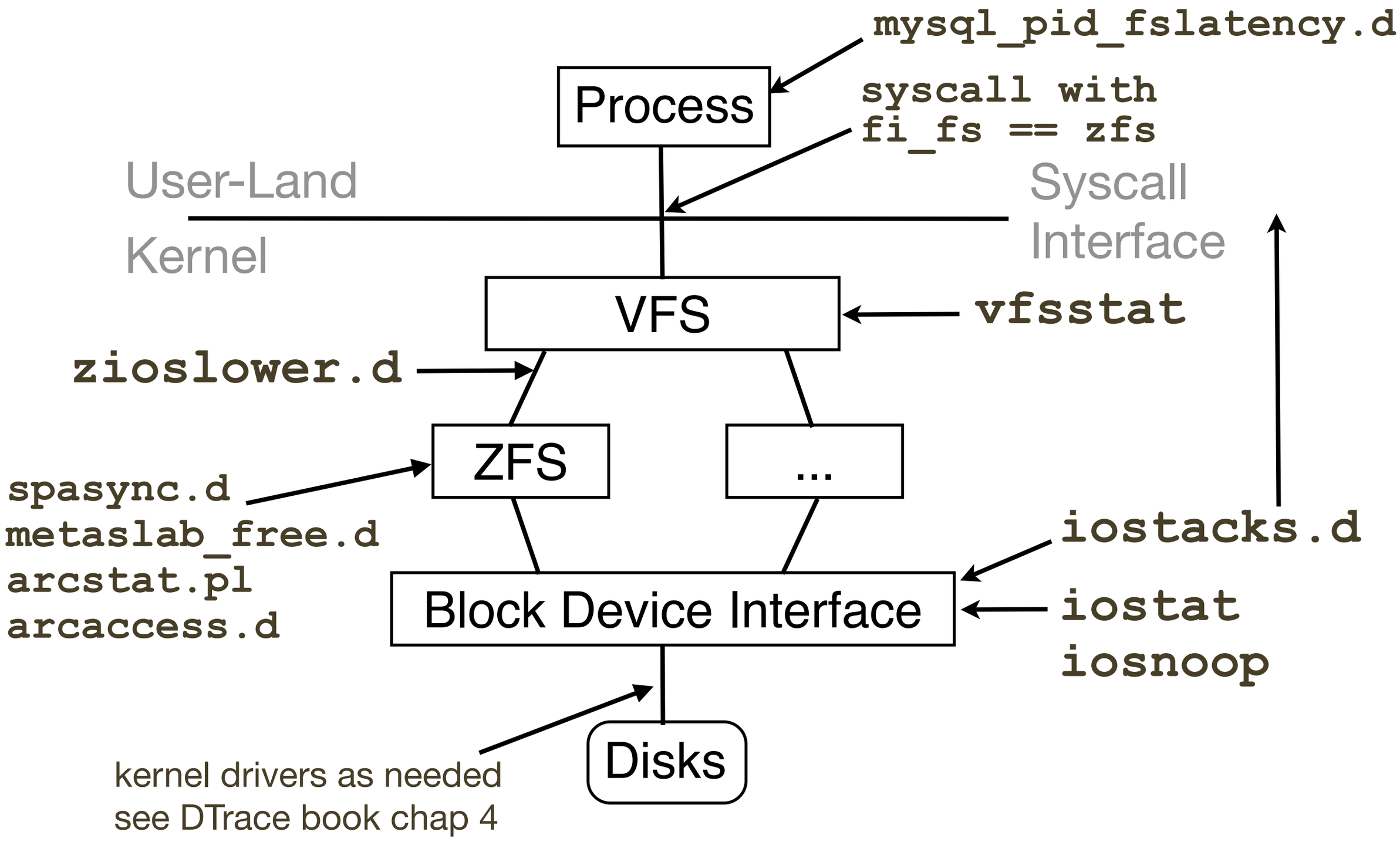
Cloud computing: within the zone

Joyent

- Suspected “noisy neighbors” hurting ZFS performance
  - ZFS I/O throttling was added to address this
- Need ZFS observability to confirm/deny issues from *within a zone* (eg, Joyent SmartMachine), however, kernel access (including DTrace io and fbt providers) is not available.
  - vfsstat (already covered) does work, as it is zone-aware
- Cloud safe DTrace tools include:
  - mysql_pid_latency.d: tracing FS calls from within the app
  - syscall tracing when fds[].fi_fs == “zfs”
- Scripts are in:
  http://dtrace.org/blogs/brendan/2011/05/18/file-system-latency-part-3/
  http://dtrace.org/blogs/brendan/2011/05/24/file-system-latency-part-4/

Recap

Joyent



There's more

Joyent

- There are more tools
  - zpool iostat
  - zpool status
    - degraded? resilvering?
  - zfs get all
    - recsize match workload? compression? dedup?
  - ziostat

There's more, cont.

- There are more tools
  - And much, much, more DTrace

DTrace book, Chapter 4 Disks

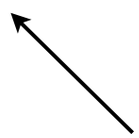
Joyent

Table 4-5 Script Summary

Script	Target	Description	Provider
iolatency.d	I/O	Systemwide I/O latency as a distribution plot	io
disklatency.d	I/O	Measures I/O latency and shows as a distribution plot by device	io
iotypes.d	I/O	Measures I/O latency by type of I/O	io
rwtime.d	I/O	Shows read and write I/O times	io
bitesize.d	I/O	Shows disk I/O sizes as a distribution plot	io
seeksize.d	I/O	Shows disk I/O seek distances as a distribution plot	io
iosnoop	I/O	Traces disk I/O live with various details	io
iotop	I/O	Summarizes disk I/O and refresh screen	io
iopattern	I/O	Shows disk I/O statistics including %random	io
geomiosnoop.d	I/O	Traces GEOM I/O requests (FreeBSD)	fbt
sdqueue.d	SCSI	Shows I/O wait queue times as a distribution plot by device	fbt, sdt
sdretry.d	SCSI	A status tool for SCSI retries	fbt
scsicmds.d	SCSI	Frequency count SCSI commands, with descriptions	fbt
scsilatency.d	SCSI	Summarizes SCSI command latency by type and result	fbt
scsirw.d	SCSI	Shows various SCSI read/write/sync statistics, including bytes	fbt
scsireasons.d	SCSI	Shows SCSI I/O completion reasons and device names	fbt
scsi.d	SCSI	Traces SCSI I/O live with various details or generates reports	fbt
satacmds.d	SATA	Frequency count SATA commands, with descriptions	fbt
satarw.d	SATA	Shows various SATA read/write/sync statistics, including bytes	fbt

Table 4-5 Script Summary (*Continued*)

Script	Target	Description	Provider
satareasons.d	SATA	Shows SATA I/O completion reasons and device names	fbt
satalatency.d	SATA	Summarizes SATA command latency by type and result	fbt
idelatency.d	IDE	Summarizes IDE command latency by type and result	fbt
iderw.d	IDE	Shows IDE read/write/sync statistics, including bytes	fbt
ideerr.d	IDE	Shows IDE command completion reasons with errors	fbt
mptsasscsi.d	SAS	Shows SAS SCSI commands with SCSI and mpt details	fbt
mptevents.d	SAS	Traces special mpt SAS events with details	sdt, fbt
mptlatency.d	SAS	Shows mpt SCSI command times as a distribution plot	sdt



Disk I/O subsystem scripts

Chapter 5 is FS includes ZFS scripts

Future

Joyent

- More tools
  - as needed, working on customer issues

Future

Joyent

- More tools
  - as needed, working on customer issues
- But not too many...

Wenger giant

Thank you!

Joyent

- email: brendan@joyent.com
- twitter: @brendangregg
- Resources:
  - http://dtrace.org/blogs/brendan
    - http://dtrace.org/blogs/brendan/2012/01/09/activity-of-the-zfs-arc/
    - http://dtrace.org/blogs/brendan/2011/06/03/file-system-latency-part-5/
  - https://github.com/brendangregg/dtrace-cloud-tools/tree/master/fs
  - http://www.dtracebook.com
  - More on latency heat maps: http://queue.acm.org/detail.cfm?id=1809426