ZFS: NEW FEATURES IN REPLICATION

WHO AM I?

Dan Kimmel

- ZFS Committer
- Filesystem Team Manager

dan@delphix.com

@dankimmel on GitHub

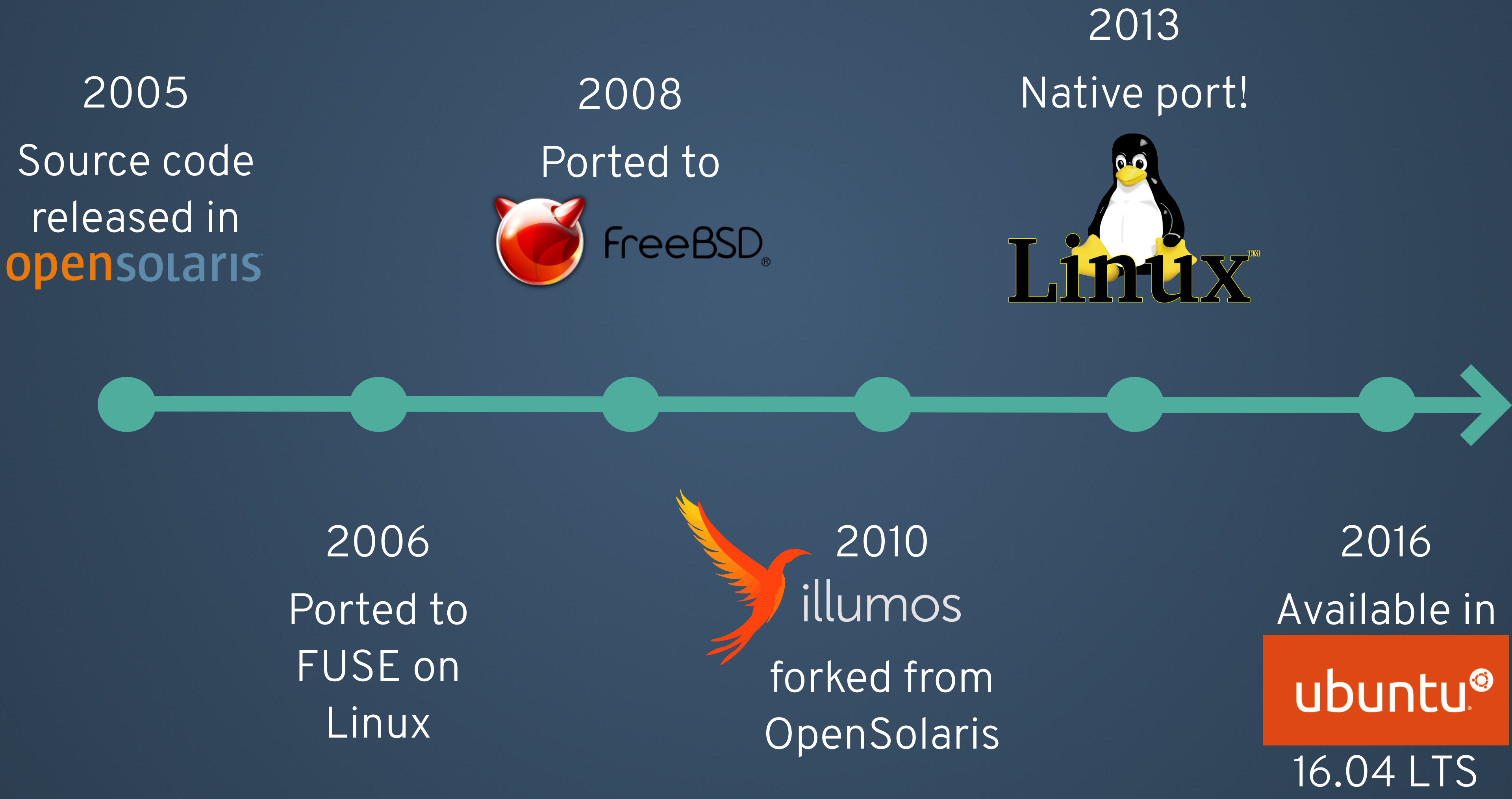
DELPHIX

the leader in database virtualization, and a leading contributor to OpenZFS

SHOW OF HANDS!

HOW MANY PEOPLE HAVE USED ZFS?

HISTORY LESSON



2005

Source code released in opensolaris

2006

Ported to FUSE on Linux

2008

Ported to FreeBSD®

2010

illumos forked from OpenSolaris

2013

Native port!

Linux™

2016

Available in ubuntu® 16.04 LTS

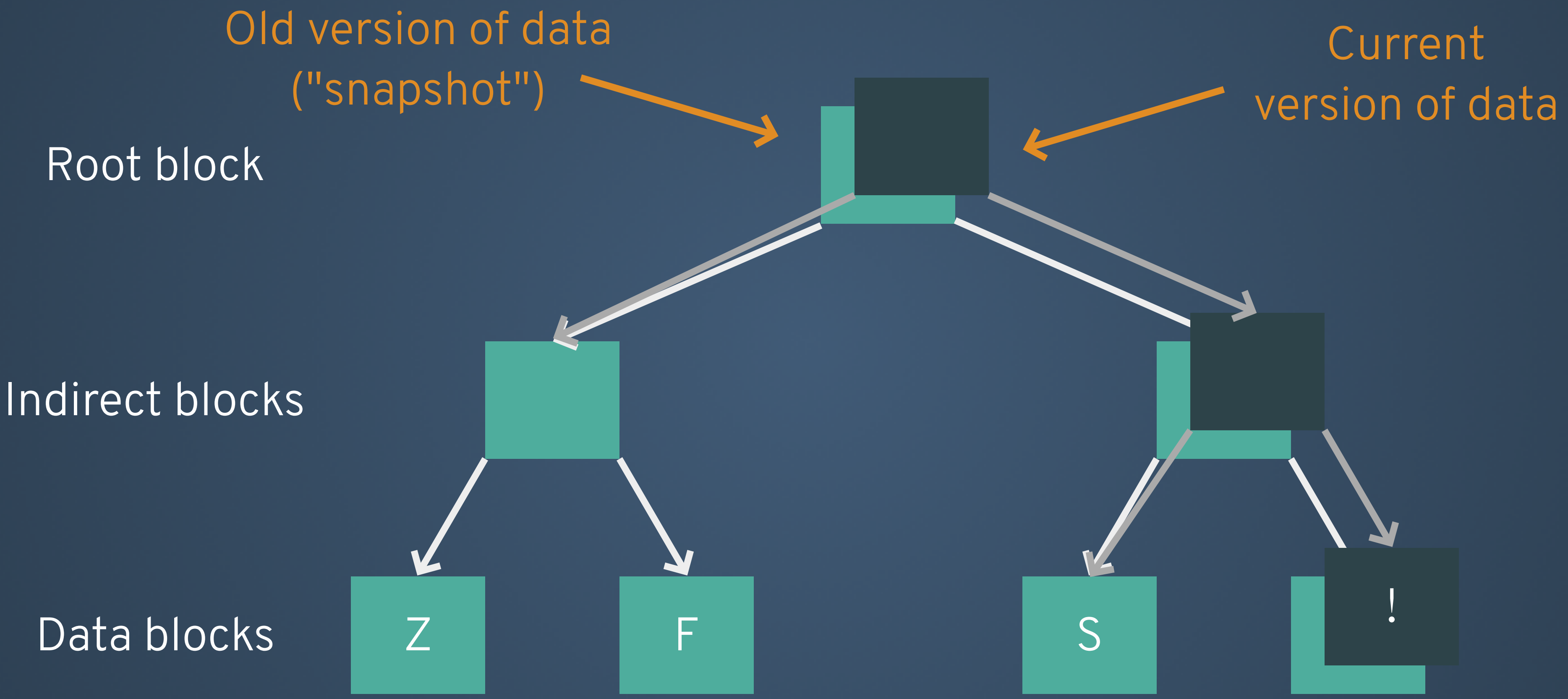
ZFS...

- Is a local filesystem
- Includes logical volume management
- Does snapshots and clones
- Can compress data on disk
- Checksums data end-to-end, ensuring integrity
- Has many other awesome features
  - ... which are not relevant to this talk :-)

CLI CRASH COURSE

```
# Create a pool named "tank", a mirror of two disks.
zpool create tank mirror disk1 disk2
# tank
#   mirror-0
#     disk1
#     disk2

# Create an LZ4-compressed filesystem on the pool.
zfs create -o compress=lz4 tank/my-fs

# Write some data into it.
cp hamlet.txt /tank/my-fs

# Take a snapshot of that filesystem.
zfs snapshot tank/my-fs@monday

# Make a clone based on that snapshot.
zfs clone tank/my-fs@monday tank/my-new-fs
```

HOW SNAPSHOTS WORK



ZFS REPLICATION

A.K.A. SEND AND RECEIVE

- Take a snapshot of the filesystem you want to send
- Serialize the snapshot using "zfs send"
- Recreate filesystem elsewhere using "zfs receive"

EXAMPLE

```
# Take a snapshot of your filesystem.
zfs snapshot tank/my-fs@monday

# Serialize that snapshot to a file.
zfs send tank/my-fs@monday >monday.zstream

# Recreate that snapshot.
zfs receive tank/new-fs <monday.zstream

# Now look at what you've done.
zfs list -t all -r tank
# NAME                 USED  AVAIL  REFER  MOUNTPOINT
# tank                2.00G  21.1G    23K  /tank
# tank/mds             111M  23.0G   111M  /mds
# tank/my-fs            23K  21.1G    23K  /tank/my-fs
# tank/my-fs@6pm          0      -    23K  -
# tank/new-fs           23K  21.1G    23K  /tank/new-fs
# tank/new-fs@6pm         0      -    23K  -
```

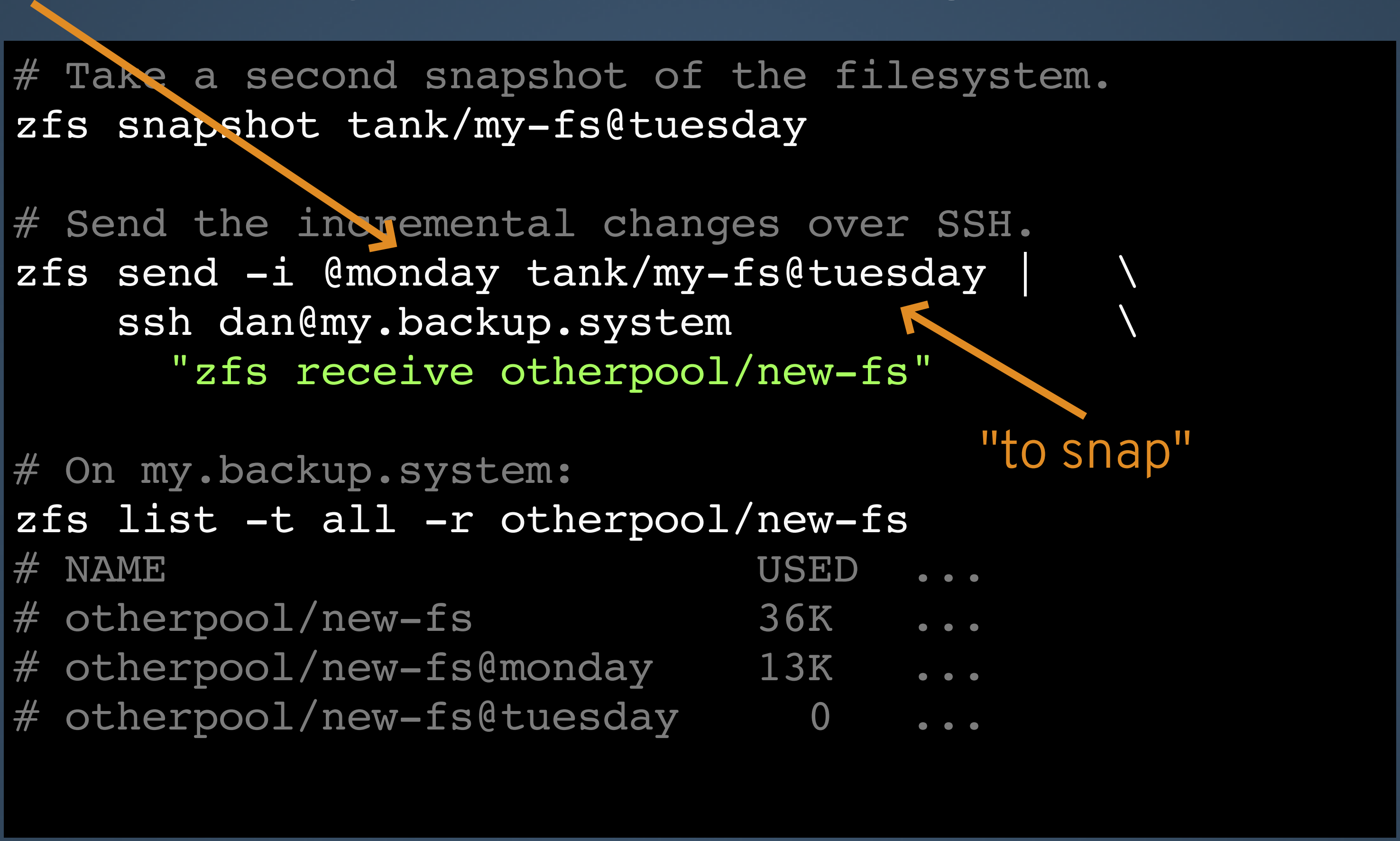
(same as piping "send | recv")

OVER THE NETWORK

```
# Take a snapshot of your filesystem.
zfs snapshot tank/my-fs@monday

# Send the snapshot over SSH and receive
# it on the other side.
zfs send tank/my-fs@monday |          \
    ssh dan@my.backup.system          \
      "zfs receive otherpool/new-fs"

# On my.backup.system:
zfs list -t all -r otherpool/new-fs
# NAME                       USED  ...
# otherpool/new-fs           36K   ...
# otherpool/new-fs@monday    13K   ...
```

INCREMENTAL SEND

"from snap"



```
# Take a second snapshot of the filesystem.
zfs snapshot tank/my-fs@tuesday

# Send the incremental changes over SSH.
zfs send -i @monday tank/my-fs@tuesday |    \
    ssh dan@my.backup.system                \
      "zfs receive otherpool/new-fs"

# On my.backup.system:
zfs list -t all -r otherpool/new-fs
# NAME                        USED   ...
# otherpool/new-fs            36K    ...
# otherpool/new-fs@monday     13K    ...
# otherpool/new-fs@tuesday      0    ...
```

"to snap"

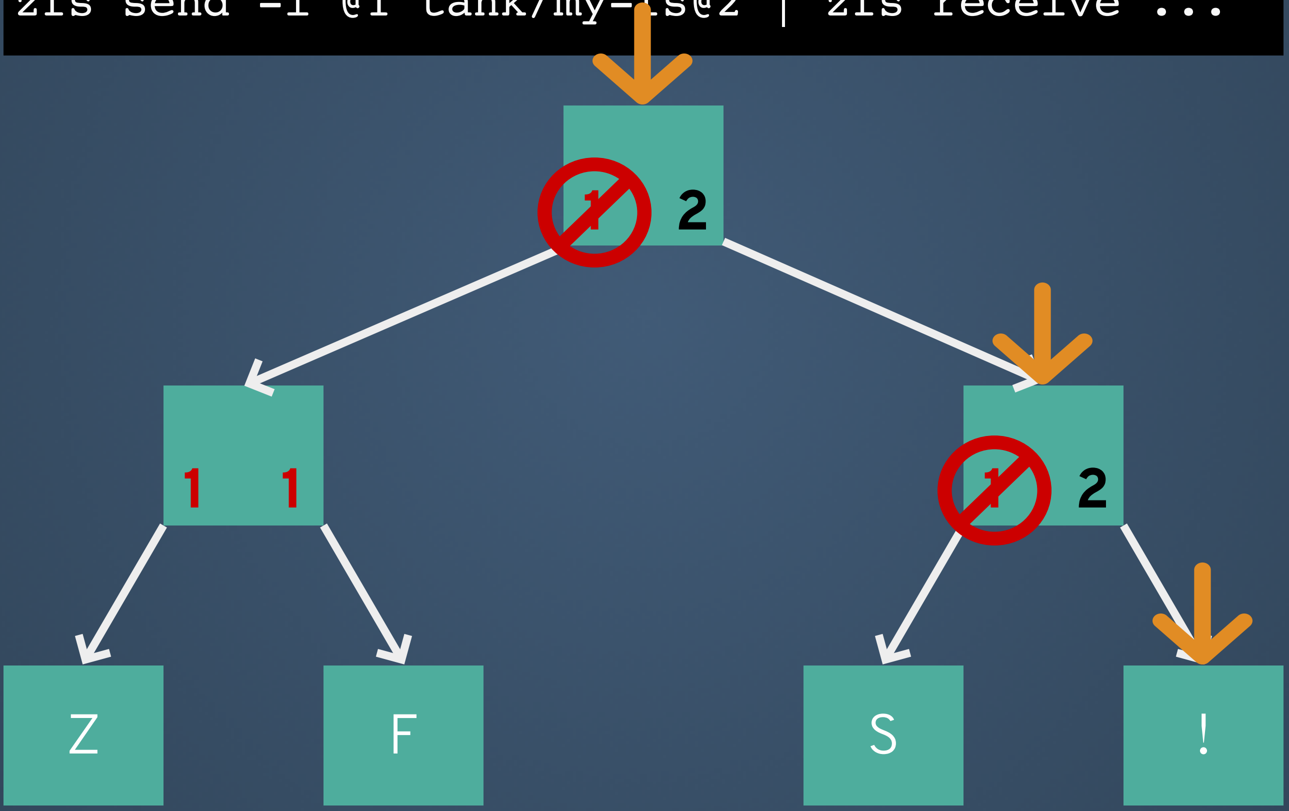
COMPARISON TO OTHER TOOLS

- Communicates in **only one** direction (send ➡ receive)
  - Not latency sensitive, can use full net throughput
- Uses prefetching, can use full disk throughput
- Read / send minimal amount of data, even for incremental changes to the data
  - Only changed blocks are read / sent (using birth times)
  - Maintain block-sharing relationships between snapshots
- Completeness of data sent
  - Preserves all POSIX layer state
  - No special-purpose code for permissions

ONLY TRAVERSE CHANGED DATA

EVEN FOR INCREMENTAL DATA UPDATES

```
zfs send -i @1 tank/my-fs@2 | zfs receive ...
```



* I'm fibbing slightly for explanatory purposes. ZFS actually uses transaction group number (rather than snapshot name) to track birth times.

COMPLETENESS OF DATA SENT

- ZFS send operates exclusively on DMU objects
- Doesn't try to interpret data being sent
- All esoteric POSIX-layer features preserved by design
  - Files, directories, permissions metadata
  - SID (Windows) users
  - Full NFSv4 ACLs
  - Sparse files
  - Extended attributes

NEW ZFS SEND FEATURES

1. RESUMABLE REPLICATION
2. COMPRESSED SEND STREAMS

1. RESUMABLE REPLICATION

PROBLEM STATEMENT

- Your replication will take ~10 days
- There's a network outage ~once a week
  - (or sender / receiver reboot)
- Partial progress is destroyed because there's no way to pick up a partial send or receive
- **Your replication may never complete!**

☹

SOLUTION

Remember where you left off.

SENDING SIDE

- Always send stuff in order of increasing <DMU object #, offset>
- Allow someone to start a send from a particular <DMU object #, offset>

RECEIVING SIDE

- Record the <DMU object #, offset> you're at as you receive the stream
- Allow user to pull that information out after a failure with new property **receive_resume_token**

Repeat for each failure during a send

WHAT'S IN THE TOKEN?

- "From snap" snapshot GUID
- "To snap" snapshot name
- List of stream features used during the original send
- Last <DMU object #, offset> successfully received

SHOW ME HOW!

```
zfs send ... | <network> | zfs receive -s otherpool/new-fs
```

First fix the cord, then...

On the receiving side, get the opaque token with the <DMU object #, offset> stored in it

```
zfs get receive_resume_token otherpool/new-fs
# 1-e604ea4bf-e0-789c63a2...
```

Re-start sending from the <DMU object #, offset> stored in the token

```
zfs send -t 1-e604ea4bf-e0-789c63a2... | \
    <network> | zfs receive -s otherpool/new-fs
```

Does this violate the "only communicate in one direction" rule?

Kind of — but presumably you'd hide the scissors after the first time.

ANOTHER PROBLEM EXPOSED

- To ensure data integrity, sends add a checksum as the **last** thing in the stream
- If the stream is corrupted early, we waste a lot of effort and have to retry from scratch
    - The token doesn't help us figure out when the corruption occurred, just if it ended prematurely

SOLUTION: CHECKSUM AT THE END OF EVERY RECORD

- Now we know as soon as a record is corrupted, and fail receive
- We can resume sending right where the corruption happened

FINAL DETAILS

- If you don't want to resume the send, abort to remove the partial state on the receiving system:

```
zfs receive -A otherpool/new-fs
```

- All ZFS CLI operations, including these new ones, can be called programmatically as well
  - libzfs, libzfs_core

2. COMPRESSED SEND STREAMS

PROBLEM STATEMENT

- You're replicating between data centers
- You have 200GB to transfer
- And a 2Mbps network connection
- **That's ~10 days of waiting for data!**

☹

SOLUTION

Send the data compressed.

FINE, COMPRESSION

WHAT'S THE BIG DEAL?

```
zfs send ... | gzip | <network> |       \
    gunzip | zfs recv otherpool/new-fs
```

- Read the data from disk
- Compress it
- **Send less data!**
- Decompress it
- Write the stream to disk

MORE PROBLEMS...

- gzip is slow (for the compression ratio)
  - OK, let's use LZ4
- gzip is single threaded
  - OK, let's split up the stream, compress, reconstitute
- Now all the CPUs are pegged! It would be nice if we didn't have to do all this computation...
  - Use the filesystem's on-disk compression?

A BETTER SOLUTION

SENDING SIDE

- Read the data as it's compressed on disk
- Put it directly into the send stream with no additional processing

RECEIVING SIDE

- Bypass any compression settings the system has set
- Write the compressed data directly to disk

No extra CPU time needed!

HOW CAN I USE IT?

On the sending system

```
zfs send --compressed tank/my-fs@today | ...
```

That's it!

RESULTS

Send of 200GB logical / 75GB physical snapshot:

- Compression ratio of 2.67x
  - **Logical send speedup of ~2.5x** over constrained network!
- When sending data from cache with no network, **2.7x reduction in CPU cost** compared to old sending code*

* 2.7 looks related to the compression ratio 2.67, but it actually isn't.

It's the ratio: (CPU cost of decompressing plus sending) / (CPU cost of sending)

WRAPPING UP

- Resumable sends are available in ZFS on Linux 0.7.0-rc1
- Compressed send streams are in ZFS on Linux 0.7.0-rc2
- **0.7.0 is shaping up to be a huge release!**
    - Compressed ARC (RAM cache) can store 3x larger data
    - New cryptographic checksums: SHA-512, Skein, Edon-R
    - Hardware-accelerated RAID-Z parity, checksums
    - Big performance wins in block allocation on near-full pools
    - Greatly improved interaction with Linux memory mgmt
    - Automated (and scriptable) fault management
    - And much more...

THANK YOU!

ANY QUESTIONS?

For more information:

- OpenZFS homepage / GitHub
- OpenZFS talks (including yearly Developer Summits)
- ZFS on Linux homepage / GitHub / release notes
- ZFS User Conference (3/16-3/17 — tickets still available!)