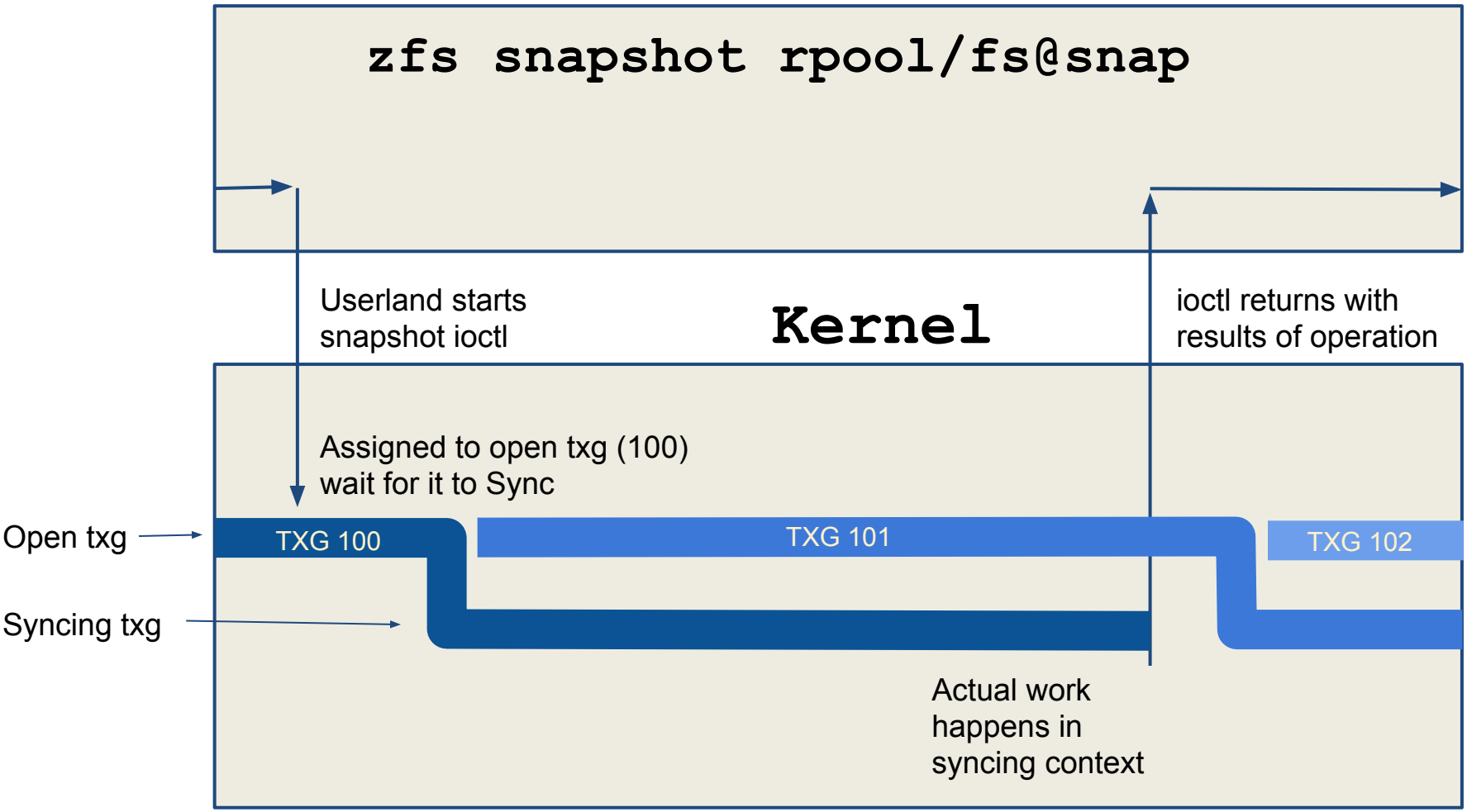
Improving the ZFS Userland-Kernel API: Channel Programs

Matt Ahrens
Principal Engineer
Delphix

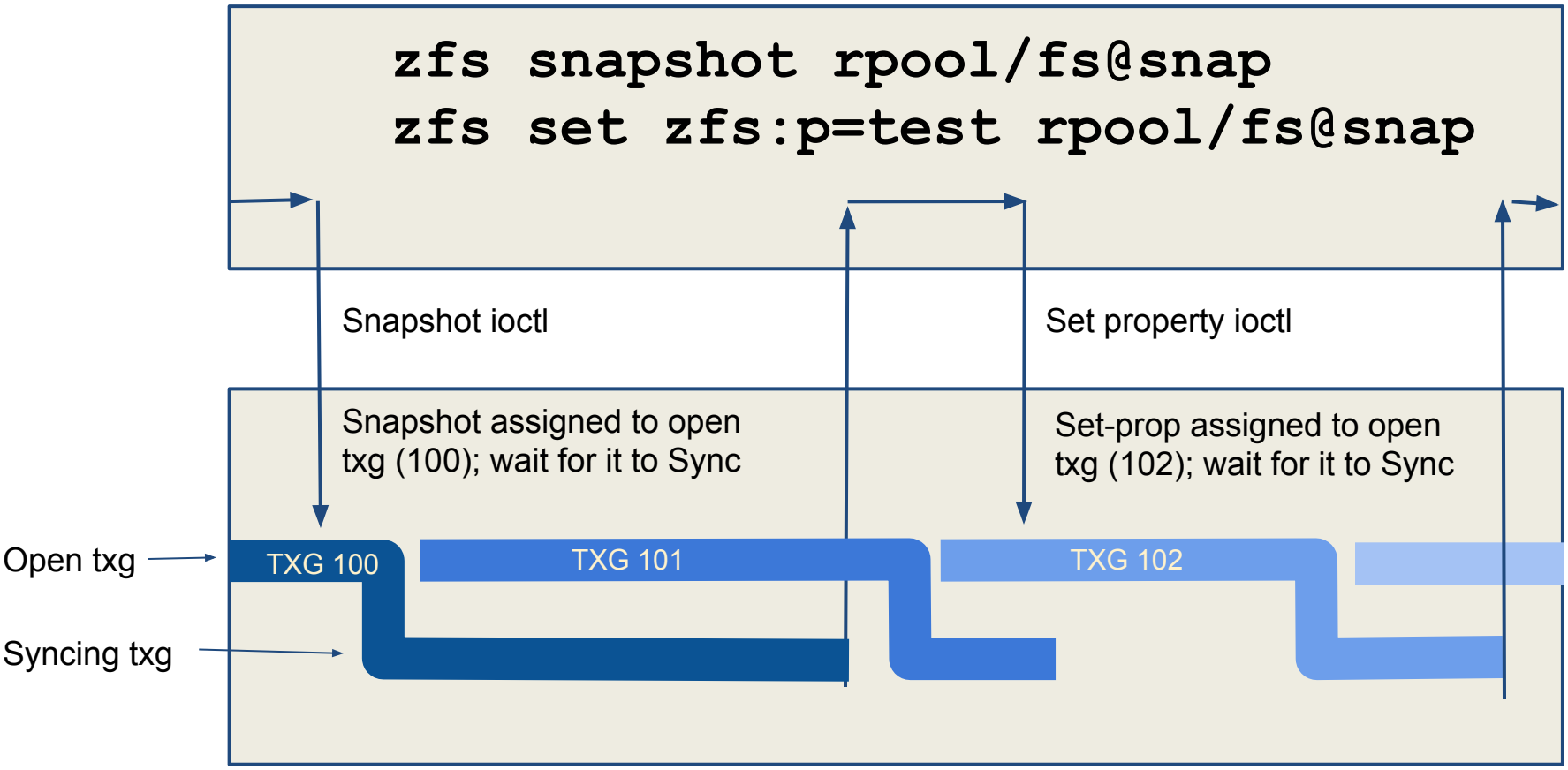
- Problems
  - Performance
  - Atomicity
  - API sprawl
- Use case - Delphix database virtualization
- Solution: Channel Programs
- Real-world use cases
- Future work

- Exciting OpenZFS announcements!

Background: ZFS Administrative Operations

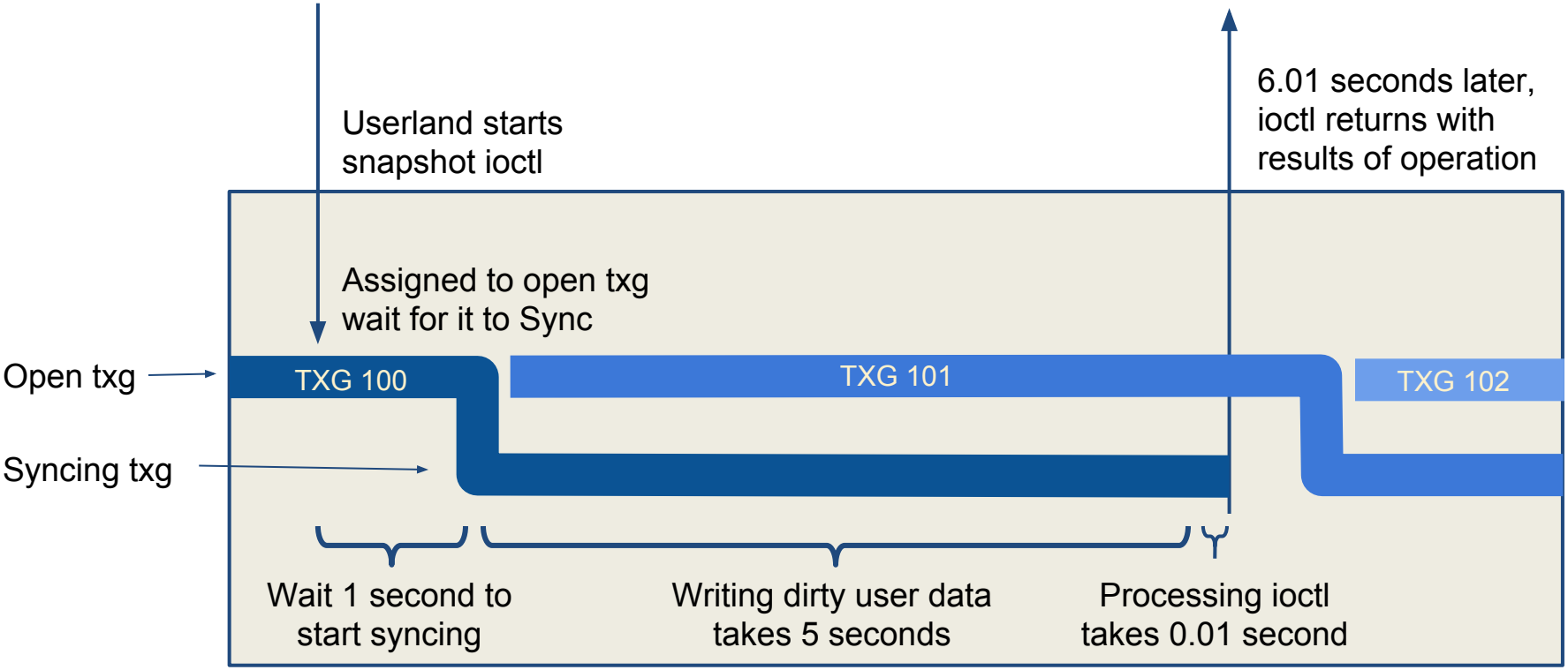


Background: Dependent Operations

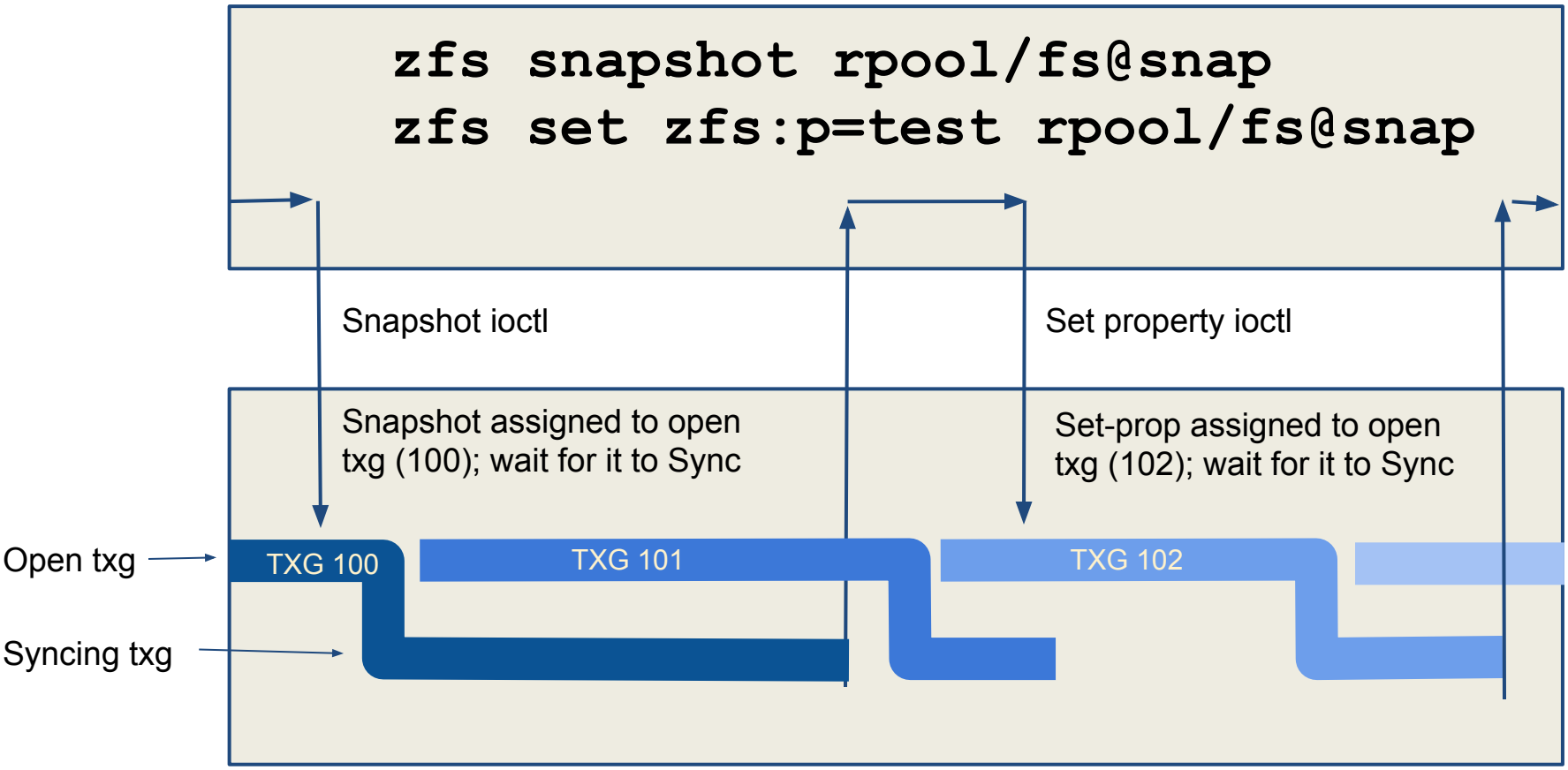


Background: spa_sync Time

- Syncing TXG takes longer when pool processing lots of writes
- Each one can take seconds
- Userland sees massive delay for each operation

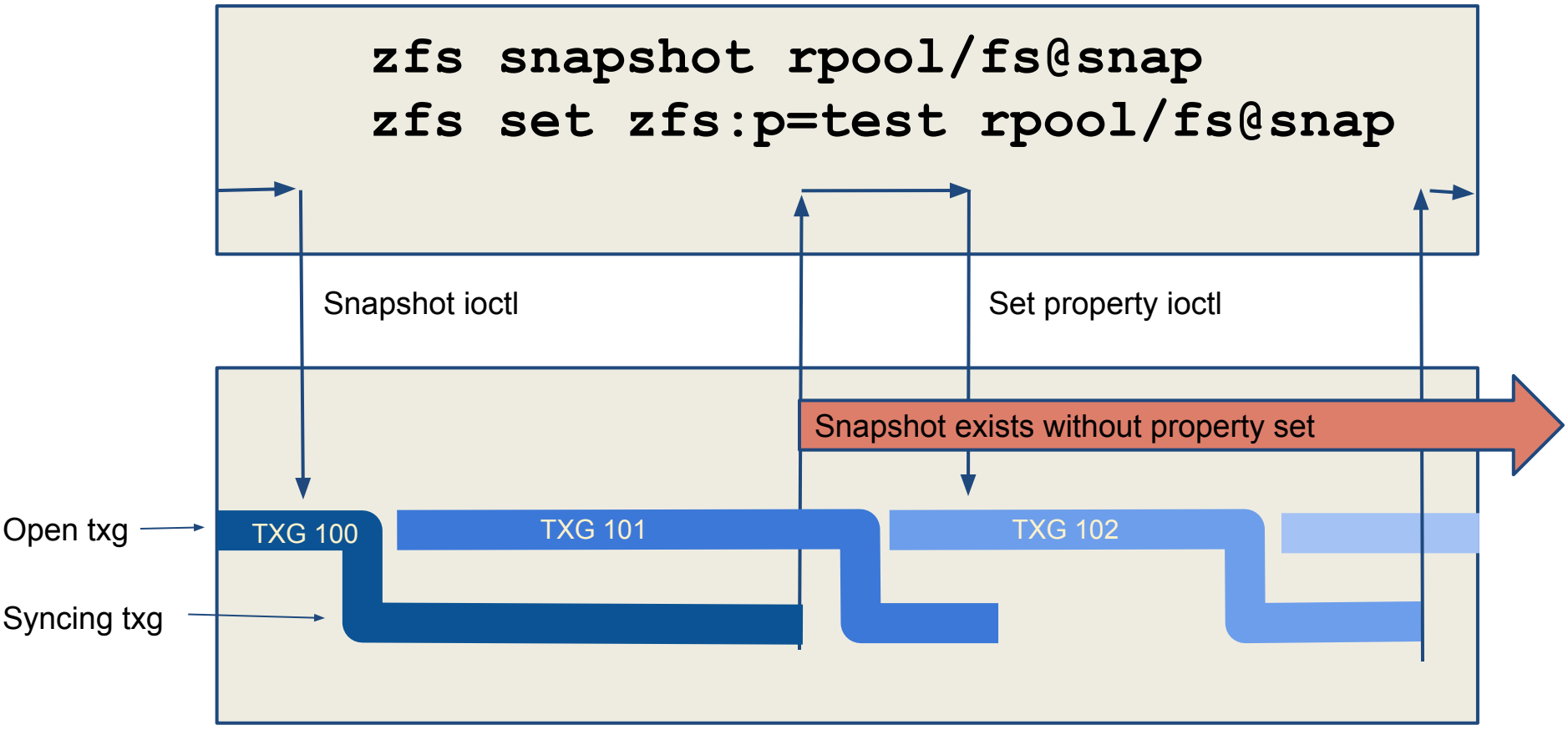


Background: Dependent Operations

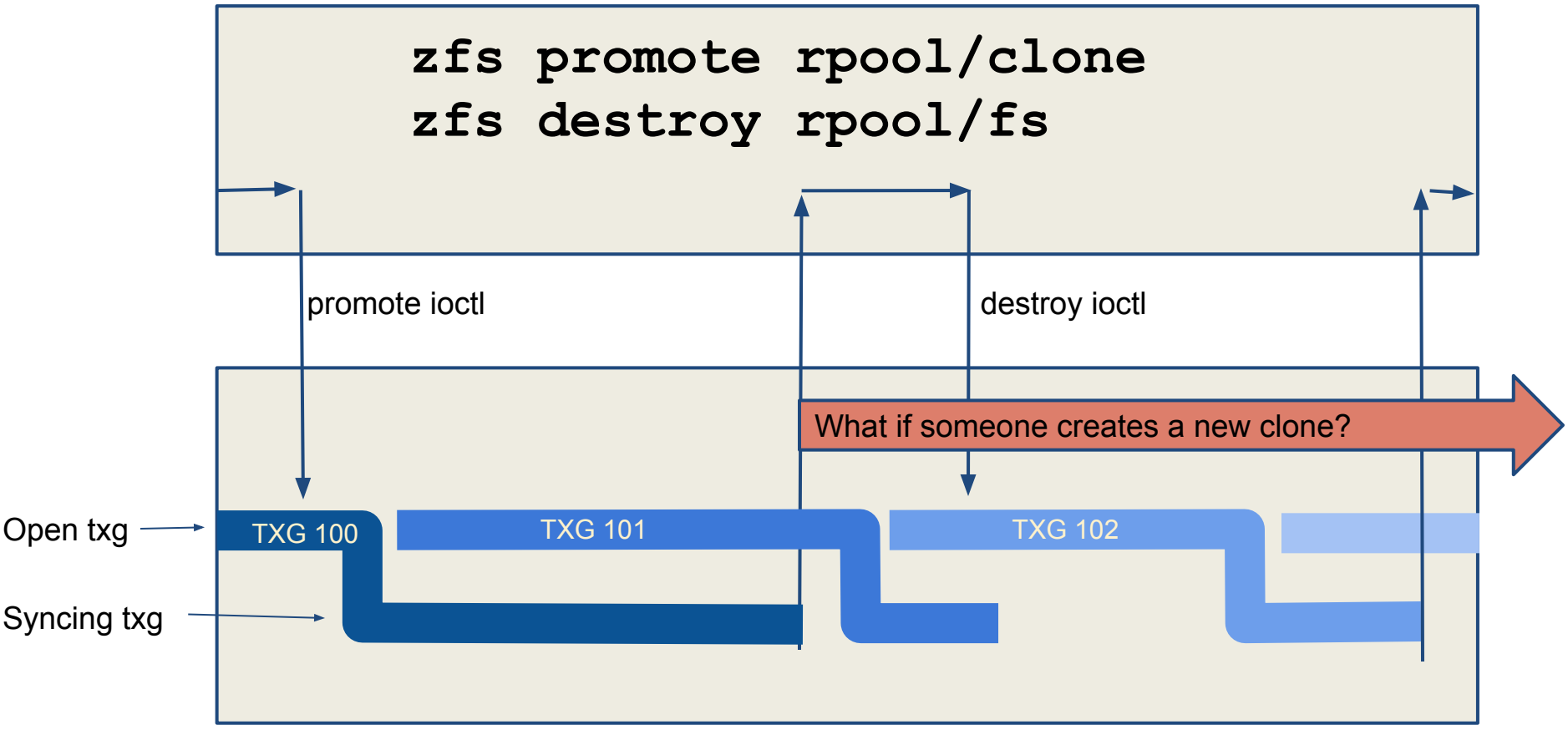


For 2 dependent operations: 10 seconds

Background: Atomicity



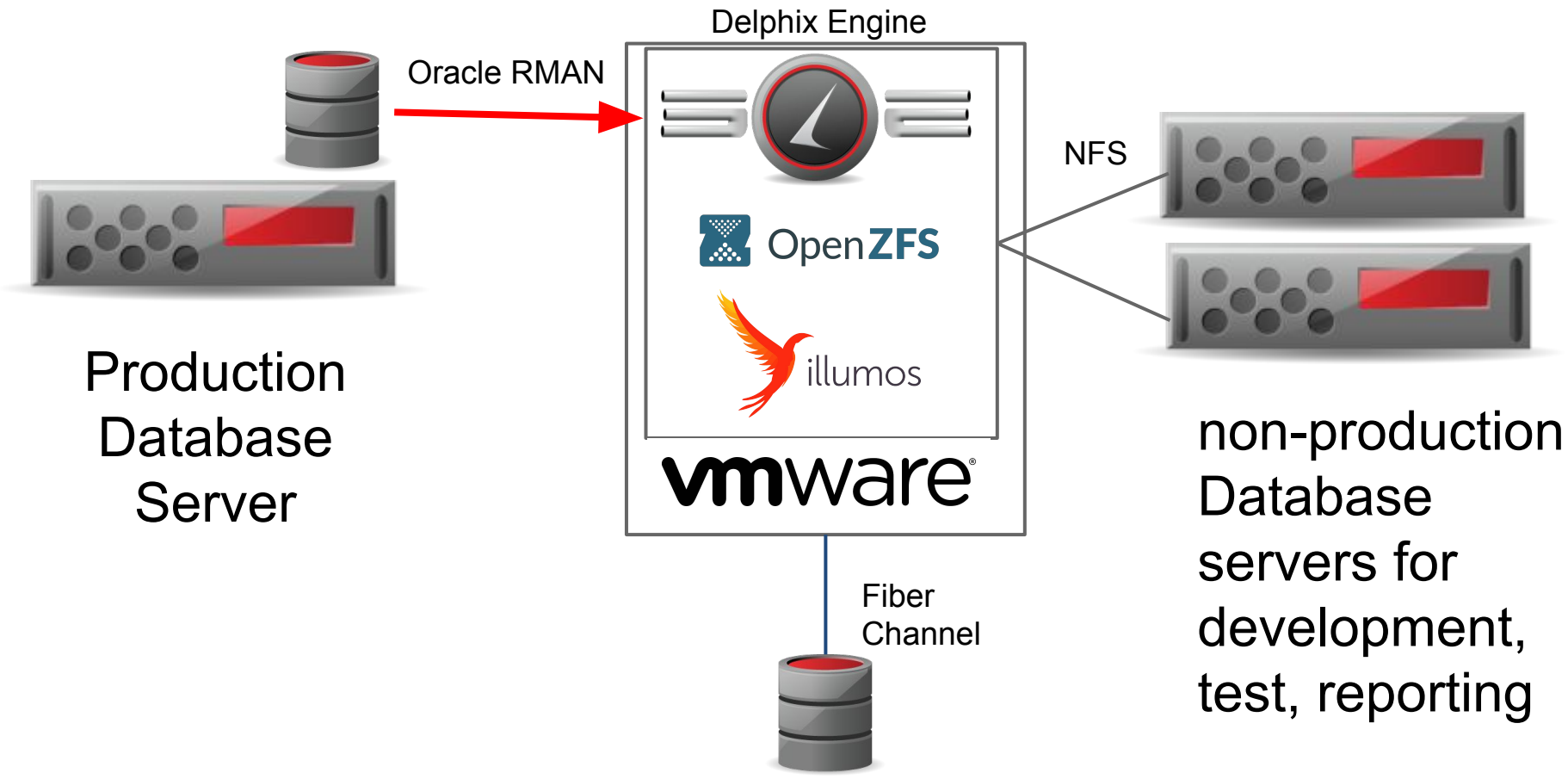
Background: Atomicity



How an ioctl Evolves: Snapshots

1. Start simple:
   - snapshot(“rpool/fs@snap”)
2. Need atomicity/speed for multiple snapshots:
   - snapshot(“rpool/fs@snap”, “rpool/fs2@snap”, …)
   - All or nothing: if any snapshot fails none are created
3. ‘zfs snapshot -r’ doesn’t work with “all or nothing”:
   - If any snapshot fails with something other than ENOENT none are created
4. Want to set properties while creating snapshots:
   - snapshot(“rpool/fs@snap”, “rpool/fs2@snap”, …, props={map})

Why not just have an ioctl for ‘zfs snapshot -r’?


Delphix Engine
Oracle RMAN
NFS
OpenZFS
illumos
vmware
Fiber
Channel
Production
Database
Server
non-production
Database
servers for
development,
test, reporting

DELPHIX

Create Virtual Database

```
zfs create pool/vdb1
zfs create pool/vdb1/version1
zfs create pool/vdb1/version1/temp
zfs clone pool/prod_db/datafile@today pool/vdb/version1/datafile
zfs snapshot pool/prod_db/logs pool/prod_db/logs@tempsnap
zfs clone pool/prod_db/logs@tempsnap pool/vdb/version1/logs
zfs destroy -d pool/prod_db/logs@tempsnap
zfs set com.delphix:user_prop=bla_bla pool/vdb/version1
zfs set sharenfs=rw=10.0.4.123 pool/vdb/version1/datafile
zfs set sharenfs=rw=10.0.4.123 pool/vdb/version1/temp
zfs set sharenfs=rw=10.0.4.123 pool/vdb/version1/logs
```

Refresh Virtual Database

```
zfs create pool/vdb1/version2
zfs create pool/vdb1/version2/temp
zfs clone pool/prod_db/datafile@today pool/vdb/version2/datafile
zfs snapshot pool/prod_db/logs pool/prod_db/logs@tempsnap
zfs clone pool/prod_db/logs@tempsnap pool/vdb/version2/logs
zfs destroy -d pool/prod_db/logs@tempsnap
zfs set com.delphix:user_prop=bla_bla pool/vdb/version2
zfs set sharenfs=rw=10.0.4.123 pool/vdb/version2/datafile
zfs set sharenfs=rw=10.0.4.123 pool/vdb/version2/temp
zfs set sharenfs=rw=10.0.4.123 pool/vdb/version2/logs
```

10+ TXG's (>2 minutes)

```
. . .
zfs destroy -r pool/vdb1/version1
```

5+ TXG's (>1 minute)

Streamline the user/kernel API: Channel Programs

- Core operations are not changing frequently:
  - snapshot(“rpool/fs@onesnap”)
  - create(“rpool/onefs”)
  - destroy(“rpool/onefs”, defer=true/false)

- Stop creating a new ioctl for every possible combination of core operations

- Have syncing context interpret “channel programs” that describe what combination of operations to perform, how to do iteration, and how to deal with errors

Channel Programs: use in Delphix product

- Create snap + clone
- Destroy multiple fs & snaps
- Snapshot MDS + list datasets & props

Refresh Virtual Database

```
zfs create pool/vdb1/version2
zfs create pool/vdb1/version2/temp
zfs clone pool/prod_db/datafile@today pool/vdb/version2/datafile
zfs snapshot pool/prod_db/logs pool/prod_db/logs@tempsnap
zfs clone pool/prod_db/logs@tempsnap pool/vdb/version2/logs
zfs destroy -d pool/prod_db/logs@tempsnap
zfs set com.delphix:user_prop=bla_bla pool/vdb/version2
zfs set sharenfs=rw=10.0.4.123 pool/vdb/version2/datafile
zfs set sharenfs=rw=10.0.4.123 pool/vdb/version2/temp
zfs set sharenfs=rw=10.0.4.123 pool/vdb/version2/logs
```

10+ TXG's (>2 minutes)

```
. . .
zfs destroy -r pool/vdb1/version1
```

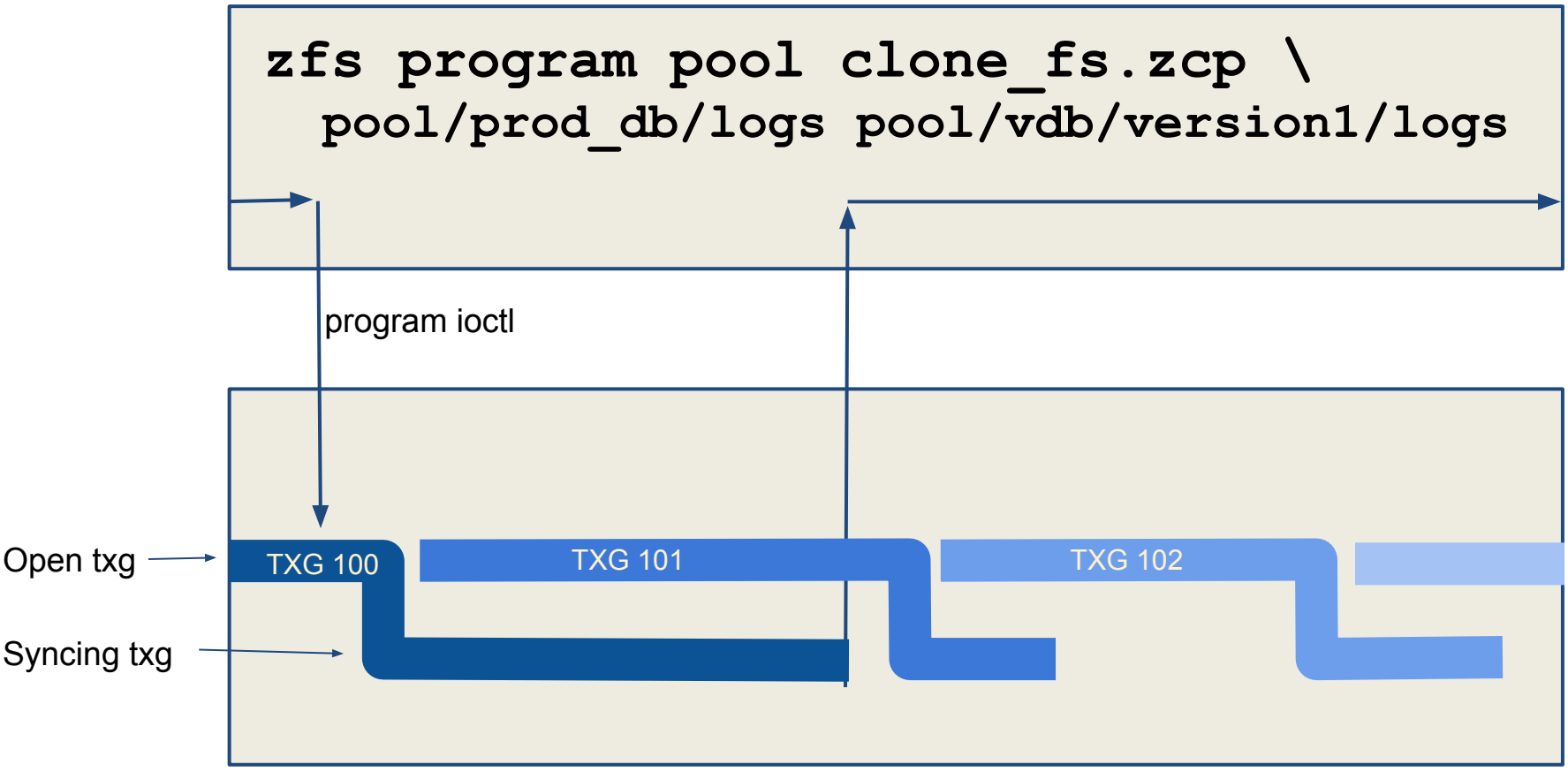
5+ TXG's (>1 minute)

Channel Programs: Clone filesystem

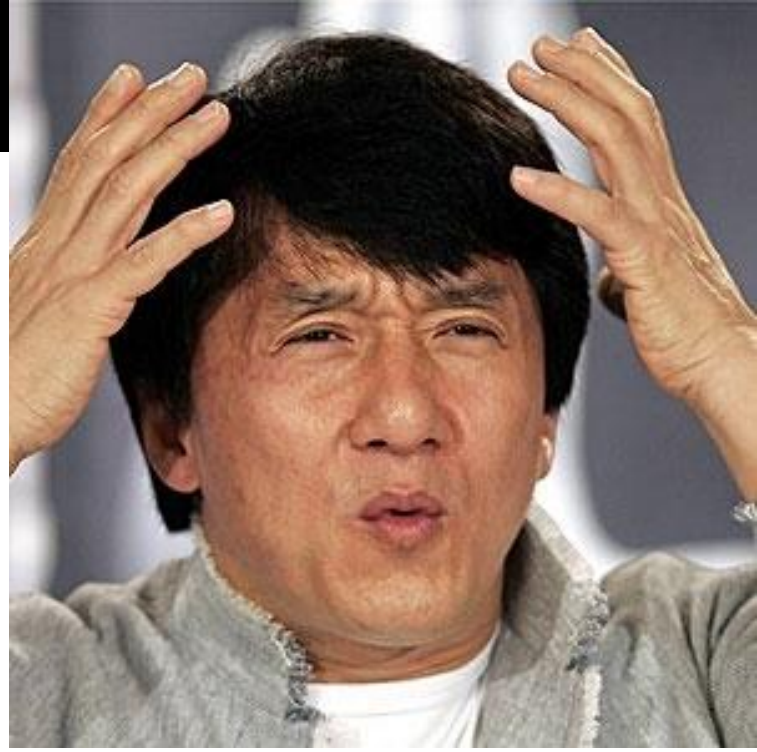
```
snap = input.fsname .. "@tempsnap"
err = zfs.sync.snapshot(snap)
if err ~= 0 then
  return err
done
err = zfs.sync.clone(snap, input.clonename)
zfs.sync.destroy(snap, defer=true)
return err
```

- Clones the current state of a filesystem, creating a new snapshot that is deferred-destroyed in the same transaction

Channel Programs: Clone filesystem



Create snapshot, clone, defer destroy in one TXG



- Safety
  - Must be root (for now)
  - Memory limit
  - Instruction count limit
- Works best for programmatic consumers
- Not every ZFS operation fits this model
  - e.g. zpool add, zfs send/receive

Refresh Virtual Database

```
zfs create pool/vdb1/version2
zfs create pool/vdb1/version2/temp
zfs clone pool/prod_db/datafile@today pool/vdb/version2/datafile
zfs snapshot pool/prod_db/logs pool/prod_db/logs@tempsnap
zfs clone pool/prod_db/logs@tempsnap pool/vdb/version2/logs
zfs destroy -d pool/prod_db/logs@tempsnap
zfs set com.delphix:user_prop=bla_bla pool/vdb/version2
zfs set sharenfs=rw=10.0.4.123 pool/vdb/version2/datafile
zfs set sharenfs=rw=10.0.4.123 pool/vdb/version2/temp
zfs set sharenfs=rw=10.0.4.123 pool/vdb/version2/logs
```

10+ TXG's (>2 minutes)

```
. . .
zfs destroy -r pool/vdb1/version1
```

5+ TXG's (>1 minute)

Channel Programs: destroy -r

```
function destroy_recursive(root)
    for child in zfs.list.children(root) do
        destroy_recursive(child)
    end
    for snap in zfs.list.snapshots(root) do
        zfs.sync.destroy(snap)
    end
    zfs.sync.destroy(root)
end

destroy_recursive(args['fs'])
```

Channel Programs: destroy -r (w/error handling)

```
function destroy_recursive(root)
    for child in zfs.list.children(root) do
        destroy_recursive(child)
    end
    for snap in zfs.list.snapshots(root) do
        err = zfs.sync.destroy(snap)
        if (err ~= 0) then
            zfs.debug('snap destroy failed: ' .. snap .. ' with error ' .. err)
            assert(false)
        end
    end
    err = zfs.sync.destroy(root)
    if (err ~= 0) then
        zfs.debug('fs destroy failed: ' .. root .. ' with error ' .. err)
        assert(false)
    end
end

args = ...
destroy_recursive(args['fs'])
```

Upgrade validation

filesystems/snaps in pool must match internal database

```
zfs list -t all -o name,used,origin,...
zfs snapshot pool/mds@upgrade_test
```

But there are concurrent `zfs snapshot`, `zfs destroy`...

Problems:

- List of filesystems/snapshots is not self-consistent
- Snapshot of MDS is not consistent with list of fs/snap
- Check spuriously fails

Solution:

- Add locking to application to prohibit concurrent zfs ops?

Channel Programs: snapshot, list fs, get props

```
args = ...
pool = args['pool']
snapName = args['snapName']
properties = args['properties']
datasets = {}

function get_all(root)
    local snapshots = {}
    local child_datasets = {}
    datasets[root] = {}
    datasets[root].properties = get_props(root)
    for snap in zfs.list.snapshots(root) do
        table.insert(snapshots, snap)
        datasets[snap] = {}
        datasets[snap].properties = get_props(snap)
    end
    for child in zfs.list.children(root) do
        table.insert(child_datasets, child)
        get_all(child)
    end
    datasets[root].snapshots = snapshots
    datasets[root].children = child_datasets
end
```

```
function get_props(ds)
    local property_table={}
    for _, property in ipairs(properties) do
        property_val = zfs.get_prop(ds, property)
        property_table[property] = property_val
    end
    return property_table
end

get_all(pool)

if snapName ~= nil and snapName ~= '' then
    err = zfs.sync.snapshot(snapName)
    if (err ~= 0) then
        zfs.debug('fs snapshot failed: ' .. snapName .. ' w
        assert(false)
    end
end

return datasets
```

```
zfs.list.snapshots(dataset)
zfs.list.clones(snapshot)
zfs.list.children(dataset)
zfs.sync.delete(<snap | fs>)
zfs.sync.promote(dataset)
zfs.sync.rollback(dataset)
zfs.sync.snapshot(snapshot)
zfs.get_prop(dataset, property)
```

Next up:

Integrate Pull Request!

Create filesystem / volume / clone

Set properties

News

- OpenZFS Pull Requests now tested on:

- Donations accepted via  &

October 24-25th (Tues-Wed)

San Francisco

Talks; Hackathon

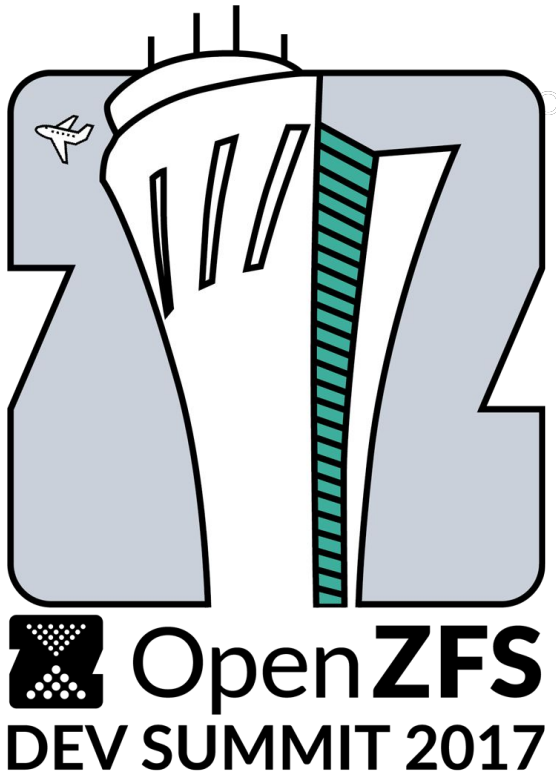
http://open-zfs.org/

Submit talks by September 4th

Registration now open!

Sponsorship opportunities

Thanks to early-bird sponsors:



DELPHIX

datto

syneto
YOUR DATA • OUR BUSINESS

intel Software

FreeBSD FOUNDATION

OVH.com
Innovation is Freedom

iX systems