ZFS and other things

Itinerary

- Part 1: ZFS
- Part 2: ^ZFS

ZFS

Brief History

Reiser3 devel starts : 2001
Rieser4 devel starts : 2004

- 2005: Development starts at Sun Microsystems (for Solaris)
- 2006: In OpenSolaris (open-sourced implementation via CDDL)
- 2006: ZFS on FUSE [to avoid CDDL/GPL incompatibility]
- 2007: Ported to FreeBSD

btrfs devel starts (at Oracle!) : 2007

- 2008: ZFS For Linux project begins at LLNL

Reiser4 devel stalls as Hans : 2008
Reiser convicted of murder.

- 2010: Oracle acquires Sun, closes original ZFS source (killing their fork of ZFS by 2017)
- 2010: illumos fork the last CDDL release of ZFS to continue project.
- 2013: Initial public launch of ZFS for Linux, start of OpenZFS project.
- 2016: ZFS for Linux bundled in Ubuntu distribution repos.

btrfs RAID still unstable : 2017

ZFS on Linux

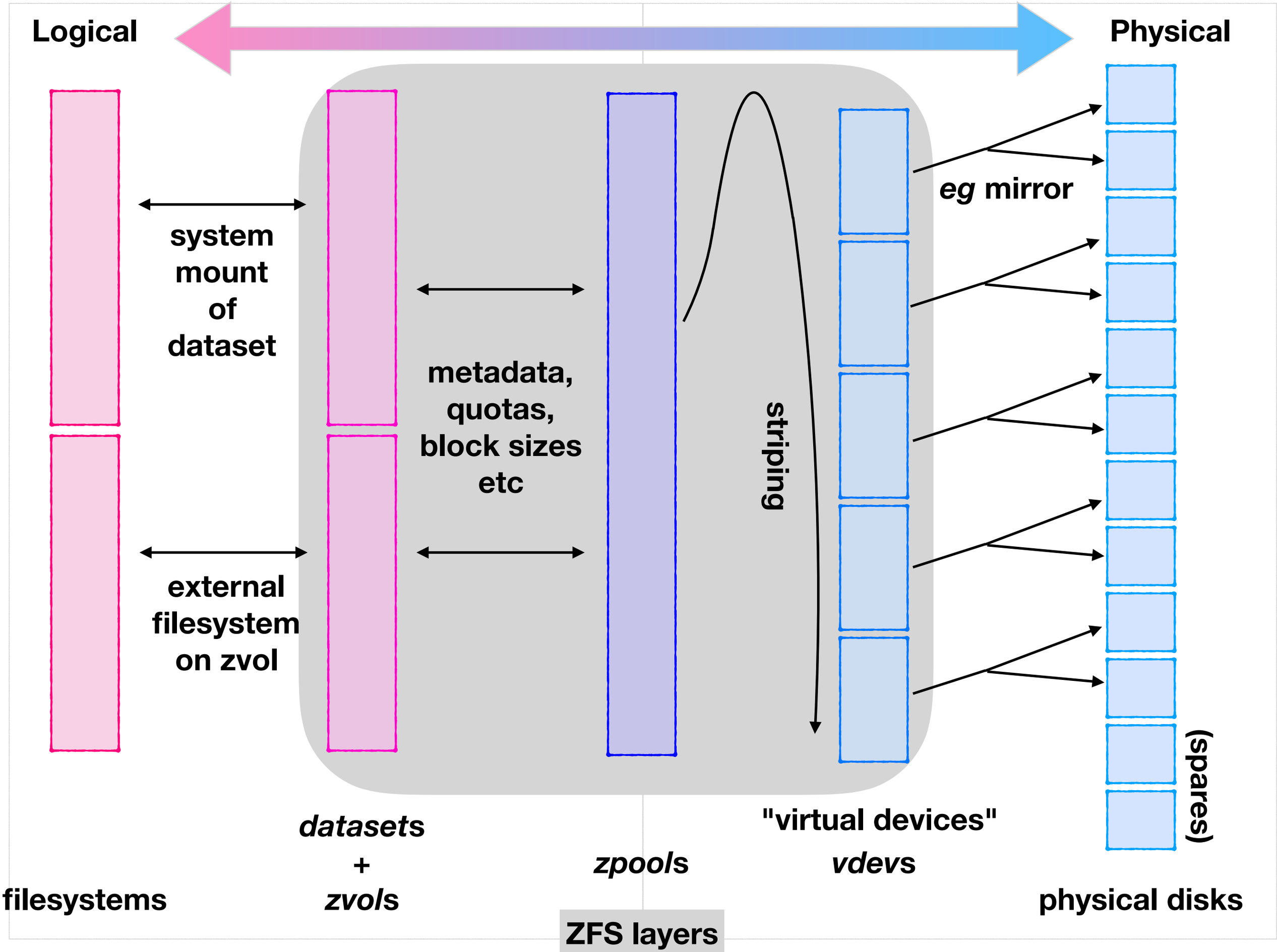
- Part of OpenZFS collaboration
- Development separate but parallel to BSD/illumos ZFS
    - (feature sets are overlapping, but neither is a subset of the other)
- Current release is 0.7.1 [Aug 2017], a bug fix of 0.7.0 [Jul 2017].

Why ZFS?

- Features:
  - file checksumming
  - file compression (w/ almost zero CPU overhead)
  - transparently resizable filesystems
  - software "data-aware RAID" (actually filesystem level erasure coding)
    - automated self-repair
  - snapshot (and replication) support

Why not ZFS?

- Needs control of disks (ideally)
    - Issues with RAID controller "poor-man's JBOD"
- Historically tricky to configure perfectly.
    - Needs more RAM than thin storage server
    - Benefits from SSD caches in some circumstances.
- Can't shrink "arrays" (zpools or vdevs) after the fact.
- That GPL/CDDL licensing issue


Logical
Physical
system
mount
of
dataset
metadata,
quotas,
block sizes
etc
striping
eg mirror
external
filesystem
on zvol
(spares)
datasets
+
zvols
zpools
"virtual devices"
vdevs
filesystems
physical disks
ZFS layers

Recent improvements (zfs-0.7.x)

- Hot Spares are actually Hot.
    - Disks marked as global hot spares will *automatically* replace any "failed" disk, in any zpool.
- Metadata performance actually is.
    - A host of improvements, including multithreading and batching of updates, extended attrs in dnodes etc
- I/O performance actually is.
    - Improvements to ARC ("page cache"), including in-memory compression, improved efficiency

Experiences - ECDF

- SE / Grid Storage:
  - Mostly RAID "fake-JBOD" servers (entire pool storage migrated)
    - compression on
- Hypervisors backed by ZFS for *all* services in VMs
- Debugging problems much faster with ZFS monitoring tools.

Experiences - ECDF

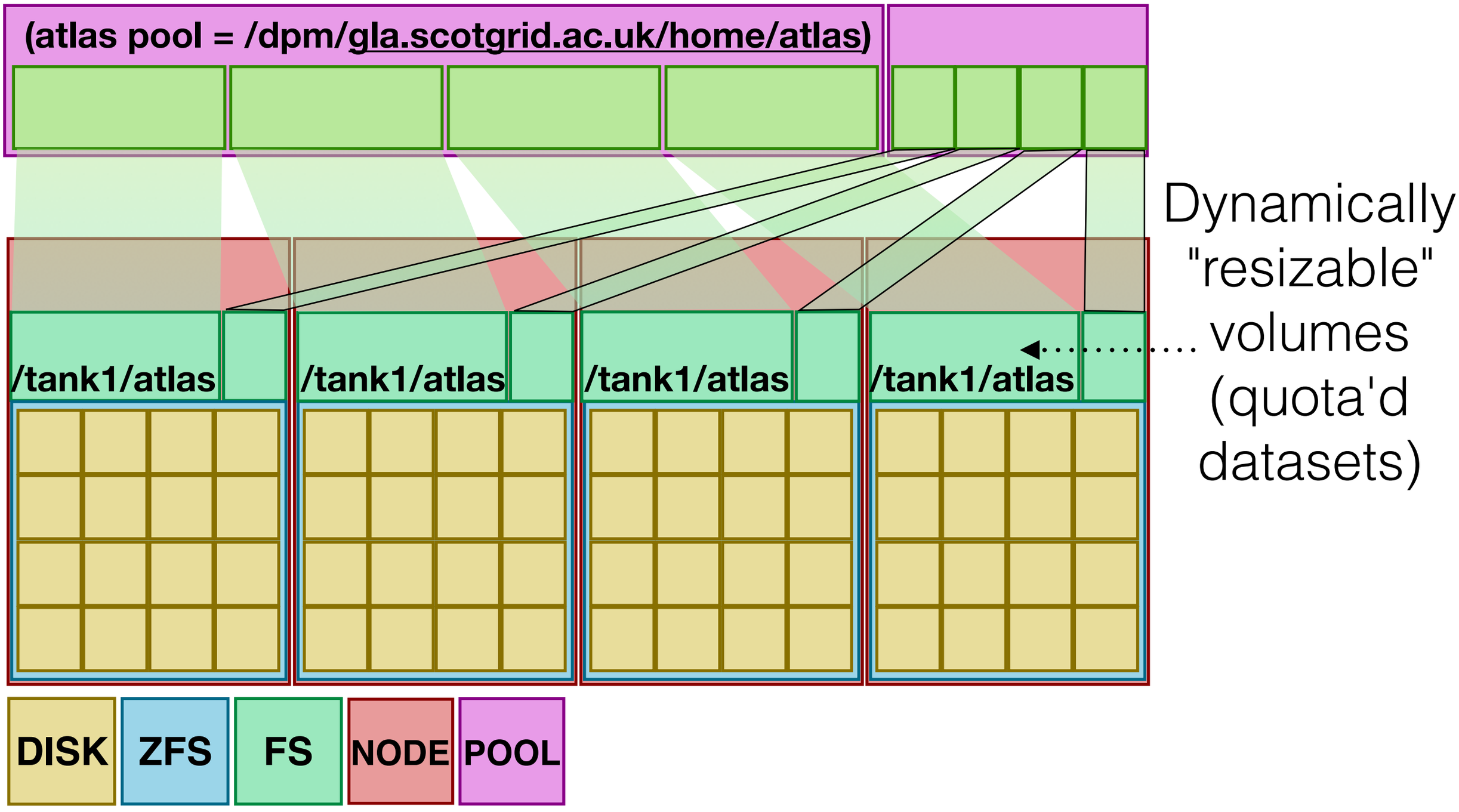
- Negatives:
  - ZFS does not like transparent_hugetables (on in RH7)
    - breaks the ARC caching layer
  - ZFS does not always report the I/O errors which cause it to drop a disk (but it does drop disks correctly)
    - known issue - see, e.g. https://github.com/zfsonlinux/zfs/issues/4149
  - ZFS may not also track all errors over reboots?
- Future: ZFS/Docker integration is very interesting.

Experiences - Glasgow

- Multiple classes of pool nodes:
  - new HBA-backed storage
    - very straightforward
    - 2 x zpools containing single RAIDZ2 vdev, 2 hot spares.
    - compression on (lz4)
  - older RAID-controller backed (with no JBOD support)
    - controller specific workarounds needed, various limits
    - Some controllers can't export all their disks as RAID0, as more disks than device ids!

DPM with ZFS Datasets

DPM does not directly control pool space



Example config

```
tank1  used                  75.8T                  -
tank1  available             146T                   -
tank1  referenced            288K                   -
tank1  compressratio         1.01x                  -
tank1  mounted               yes                    -
tank1  quota                 none                   default
tank1  reservation           none                   default
```

```
tank1/atlas  used                  75.2T                  -
tank1/atlas  available             125T                   -
tank1/atlas  referenced            75.2T                  -
tank1/atlas  compressratio         1.01x                  -
tank1/atlas  quota                 200T                   local
tank1/atlas  reservation           none                   default
```

```
tank1/others  used                  680G                   -
tank1/others  available             21.3T                  -
tank1/others  referenced            680G                   -
tank1/others  compressratio         1.09x                  -
tank1/others  mounted               yes                    -
tank1/others  quota                 22T                    local
```

Interesting fact = "other vo" storage *slightly* more compressible

Experiences - NDGF

- Some NDGF sites provided Tier 1 distributed storage on ZFS in 2015/6

- Especially poor performance for ALICE workflows

    - ALICE I/Os contain many v small (*20 byte*!) reads
        - ZFS calculates checksums on reads - large I/O overhead compared to read size.

- (Arguably, this is an example of a poor workflow design, as much as a poorly chosen filesystem.)

- This is *also* a problem for conventional RAID systems, and other systems (see Alastair re Object Stores v Posix etc)

^ZFS

Topics

- Storage Evolution / Small Tier-2s

- Other transitions: GFAL2 (historical), Globus (right now)

Storage Evolution

- As Andrew noted yesterday, significant mismatch between
  - WLCG Experiment movement (less/no storage at many T2s)
    - see, eg Chris' CMS diskless sites talk yesterday
      - ATLAS a bit behind (and our testing here contingent on ATLAS development too)
  - Other VOs/Communities?
    - "modern" object stores at ?every? site -> workernode local data distribution?
    - do these bids look like "reality", or "what they think looks good"?

"Industry" Big Data

- "Big Data" in Industry
  - Active Storage:
    - (transient) *distributed* data stores (Spark RRDs etc)
    - *coupled* to the *parallel* workflows performed on them
    - data locality is *important* (and optimised for)
  - Archive Storage:
    - distributed, resilient object storage
    - **HTTP** transport, **capability**-token security, (block replicated+erasure coded across servers)

HDFS/CEPH/ other distributed block storage "on" worker nodes.

S3/Swift interface to object stores.

"Industry" Big Data

- "Big Data" in Industry
  - Active Storage:
    - (transient) *distributed* data stores (Spark RRDs etc)
    - *coupled* to the *parallel* workflows performed on them
    - data locality is *important* (and optimised for)
  - Archive Storage:
    - distributed, resilient object storage
    - HTTP transport, capability-token security, (block replicated+erasure coded across servers)

~ WLCG event parallelism see presentations at GDB and previous CHEPs

further from traditional WLCG/Grid SE: Auth* is very different Resiliency is smarter.

Every Site is Different

- Some Tier-2 sites share resources with non-GridPP entities
  - These sites can and should move with their other stakeholders. [See, eg, Durham]
  - (Shared service sites like ECDF don't have a problem, as they're a level up in abstraction)
  - WLCG VOs can and will adapt to trends in cpu + storage provisioning.
- Some Tier-2 sites have wholly-GridPP-owned resources.
  - This is the larger problem…

Some simple steps..

- Smallest sites need most critical engagement
    - Most of these are majority ATLAS in the UK.
- Difficult to get things to move, because of limited time at those sites.
    - (Even with CMS Diskless work, the majority of work has been at the big sites, with no effort needed at Oxford, RHUL [or even Glasgow, etc])
    - Is it possible to achieve a "zero effort" test for small sites which isn't just "turn things off"?
        - Even requiring a physical host can be too much investment.
- We would like to work with the smallest sites with ATLAS provision to test diskless operations…

Some quick notes from the preGBD + GDB

- Object stores covered by Alastair
- Much development in this, also driven by other "future stuff" work (eg work with Apache Spark etc is somewhat synergistic)
- Dynafed becoming the favoured "glue"/"translator" layer.
- I'll leave tapes to the Tier 1
    - Still not being beaten by disks [but there was some discussion around the usual backups v archives thing]
    - "What is the (practical) intent of Tape1 storage class?"

WLCG Workload Management @ GDB

- GDB on Wednesday:
  - recommended CE/batch pairs are:
    - HTCondorCE / HTCondor
      - no obvious synergy with storage migration
    - ARC-CE / [SLURM or HTCondor]
      - obvious synergy with ARC-cache for smaller sites (mentioned at GDB too), which *is* well-tested for ATLAS (even in UK).

A note on Globus

- Globus "central" support going away Jan 2018
- Agreed that OSG + EGI SW providers will take up necessary support in short term.
- Long term:
  - Migration from GridFTP -> HTTP(S), xrootd
    - [Supported by DPM, dCache, StoRM already…]
    - **Opportunity** here to pivot to more "standard" interface.
  - Migration from GSI libs -> Oauth2 (?)
    - [Needs more work? Via DynaFed?]
    - **Opportunity** here to pivot to more sensible capability models.

Storage Accounting + Reporting

- Storage Accounting:
  - Works - needs deployment more widely.
    - UK has one of the highest buy-ins of any region ✔
- Storage Reporting (SRM replacement):
  - final draft spec circulated [see preGDB summary]
  - Gets lighter with each iteration…

Finis

- Discussion:
    - Where do experiments need to be for us to test small site things?
    - What's the minimum we can require a small site to do?
        - Nothing?
        - Provide a VM?
        - Provide a physical host?

Backup Slides

Aside: EC > replication

Imagine N sites.

You replicate data at 2 of them

If one site fails, you always have at least one copy

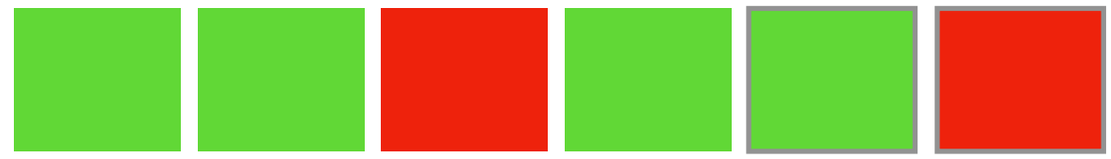
If two sites fail, then you have probability 2/(N(N-1)) of having no copies

At a cost of twice the storage for 1 copy, you gain perfect resilience for 1 failure. Lose 2/(N(N-1)) of data on 2 fails. Twice the single copy read throughput.

You *stripe* data + 2 parity across them all

If one site fails, you can always reconstruct

If two sites fail… you can always reconstruct



At a cost of 1+2/(N-2) times the storage for 1 copy, you gain perfect resilience for 2 failures. *N* times the single copy read throughput.