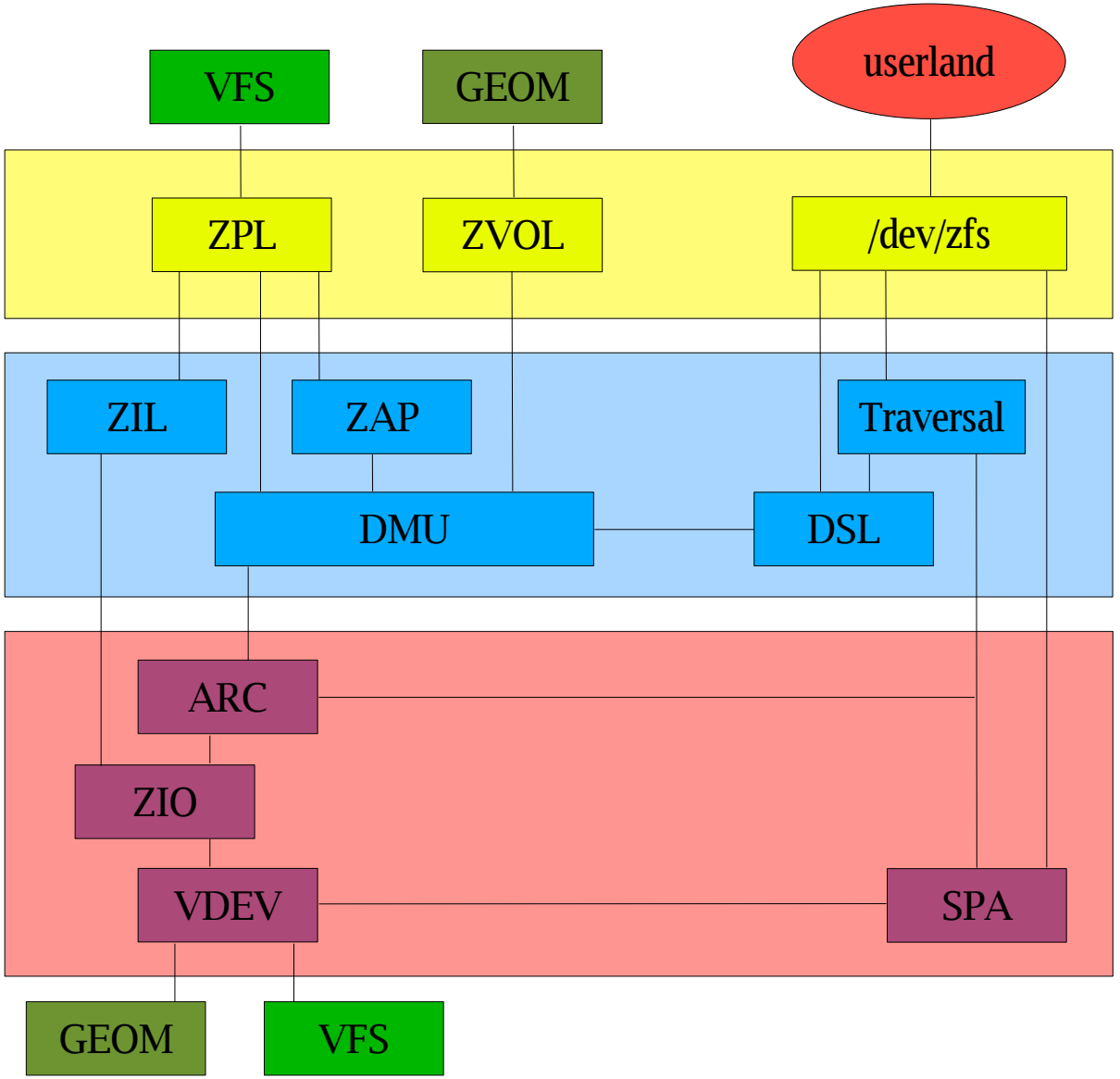
ZFS
the internals

Paweł Jakub Dawidek
<pjd@FreeBSD.org>

BSDCan
Ottawa, 2008

Layers

ZFS Layers


VFS
GEOM
userland
ZPL
ZVOL
/dev/zfs
ZIL
ZAP
Traversal
DMU
DSL
ARC
ZIO
VDEV
SPA
GEOM
VFS

SPA – Storage Pool Allocator

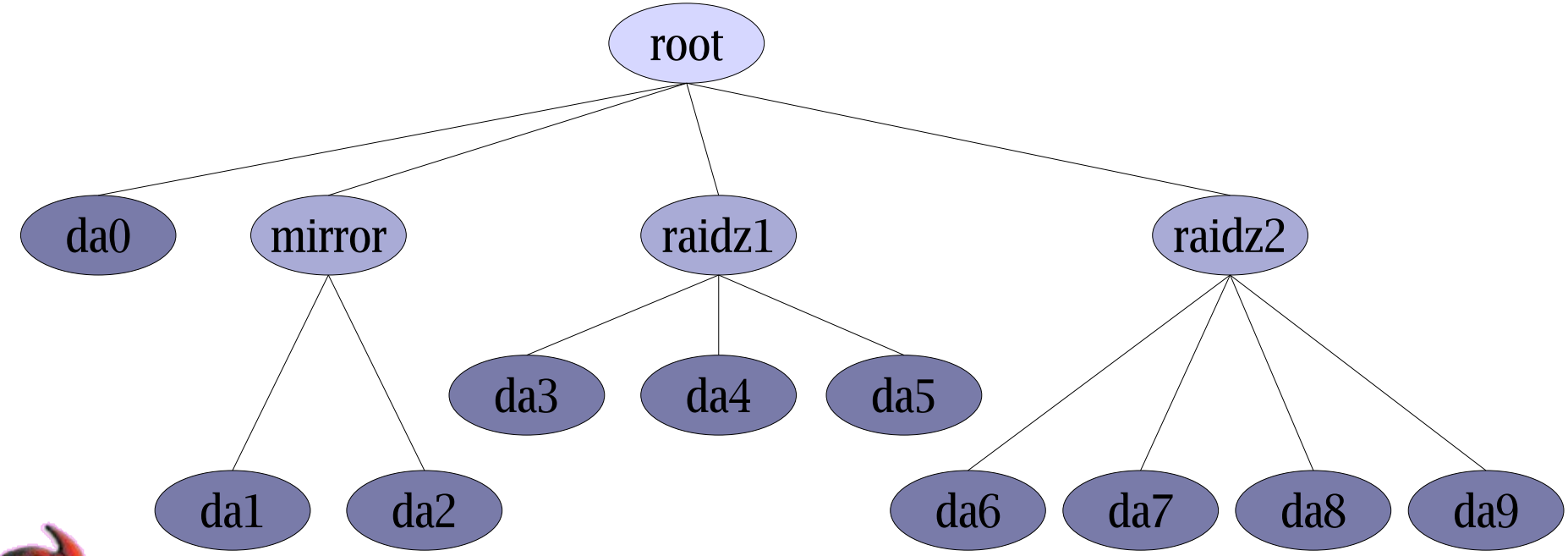
- responsible for managing pools configuration
  # zpool create/destroy/add/remove/attach/detach/...
- keeps pools' history
  # zpool history
- logs persistent pool-wide data errors
  # zpool status -v

VDEV – Virtual Devices

- provides a unified method of arranging and accessing devices
- vdevs form a tree:
  - one root vdev
  - multiple interior vdevs (mirror, RAID-Z)
  - multiple leaf vdevs (disks, files)
- VDEV is responsible for handling I/O requests and laying out the blocks

VDEV – Virtual Devices

```
# zpool create -f tank da0 mirror da1 da2 raidz1 da3 da4 da5 raidz2 da6 da7 da8 da9
# zpool status tank
```



ZIO – ZFS I/O Pipeline

- all I/O requests goes through this pipeline
- compression, checksumming (and soon encryption) happens here
- I/O requests have priorities

ARC – Adjustable Replacement Cache

- implemented based on FAST03 paper by Megiddo and Modha
- dynamically, adaptively and continually balances between recency and frequency components in an online and self-tuning fashion
- is scan-resistant (allows one-time sequential requests to pass through without polluting the cache)

DMU – Data Management Unit

- implements transactional object model on top of the flat address space presented by the SPA
- consumers interact with DMU via objsets, objects and transactions
- object represents an arbitrary piece of storage from the SPA
- objset is a collection of objects
- transaction is a series of operations that are guaranteed to be committed to the disk(s) together

DSL – Dataset and Snapshot Layer

- aggregates DMU objsets into a hierarchical namespace with inherited properties
- enforces quota and reservations
- responsible for managing snapshots and clones

ZIL – ZFS Intent Log

- makes fsync(2) and O_FSYNC work as expected

ZAP – ZFS Attribute Processor

- implemented atop of DMU
- uses scalable hash algorithms to create arbitrary (name, object) associations within an objset
- mostly used by ZPL to implement file system directories, but not only
- two types of ZAP:
  - micro ZAP (small entries, number of entires relatively small)
  - fat ZAP (big entires, huge number of entires)

Traversal

- provides safe way of traversing all data within a live pool
- used for scrub/resilver

ZVOL – ZFS Emulated Volumes

- allows to access storage pool data through GEOM providers
- /dev/zvol/<dataset>

ZPL – ZFS POSIX Layer

- presents file system abstraction of files and directories to VFS
- implements all the VOPs and VFSOPs
- allows to access ZFS managed storage via file system operations

/dev/zfs

- communication gate between userland tools (zfs(8) and zpool(8)) and the kernel

Features

RAID5, RAID6

- Redundant Arrays of **Inexpensive** Disks?
- writting to multiple disks is not atomic!
- the write hole problem – synchronize all disks just in case when an outage occurs or use expensive RAID controllers

- doing partial-stripe writes is slow (read-modify-write cycle)

RAID-Z, RAID-Z2

- similar to RAID5/RAID6 and yet so much different
- only full-stripe writes
- self-healing
- integrated with file system:
  - atomic writes (file system handles that)
  - intelligent reconstruction:
    - most important data first
    - synchronize only changes (when disk was missing for a moment only)
    - validate against checksum

RAID-Z layout

P(0,2,3,4)	D0	D2	D3	D4
P(1)	D1	P(0,1)	D0	D1
P(0,3,6,8)	D0	D3	D6	D8
P(1,4,7,9)	D1	D4	D7	D9
P(2,5)	D2	D5	P(0)	D0
P(1)	D1	P(0,1)	D0	D1

End-to-end data integrity 1/4

A regular file system: „Here is a package. It may be broken, it may not be yours, we don't care.”

End-to-end data integrity 2/4

Checksumming in hardware: „Here is a package. We are not sure if it is yours, but we know it wasn't broken when we pick it up.”

End-to-end data integrity 3/4

File system with block consistency verification: „Here is a package. We are sure it is not broken, but not so sure if this is actually your package.”

End–to–end data integrity 4/4

- block is verified against independent checksum
- for redundant configurations ZFS looks for correct block, returns that to the application and repairs corrupted copy
- ZFS blocks form Merkle tree – each block validates all its children, so the checksum in uberblock provides cryptographically-strong (in case of SHA-256) signature of the entire pool

Snapshots

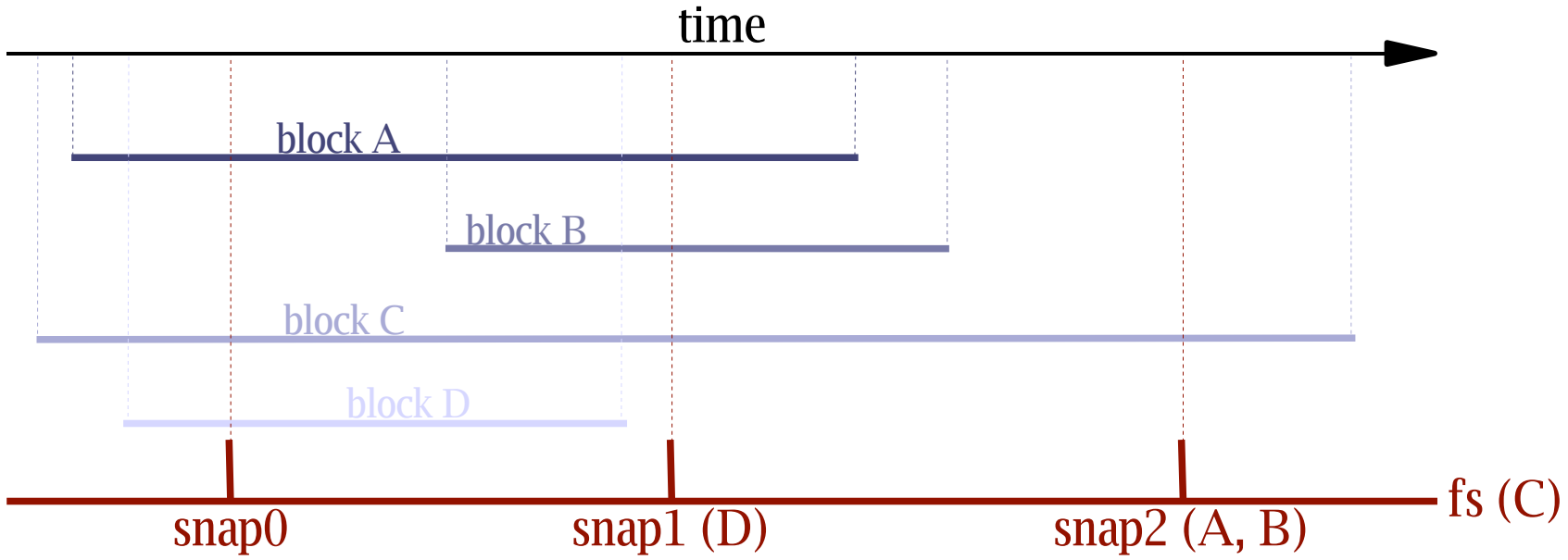
- no limit for number of snapshots
- taking a snapshot is constant-time operation
- snapshots don't slow down other file system operations
- removing a snapshot is takes time proportional to the number of blocks to free
- file system snapshots location:
  /<dataset_mount_point>/.zfs/snapshot/<snapshot_name>
- ZVOL snapshots location:
  /dev/zvol/<dataset_name>@<snapshot_name>

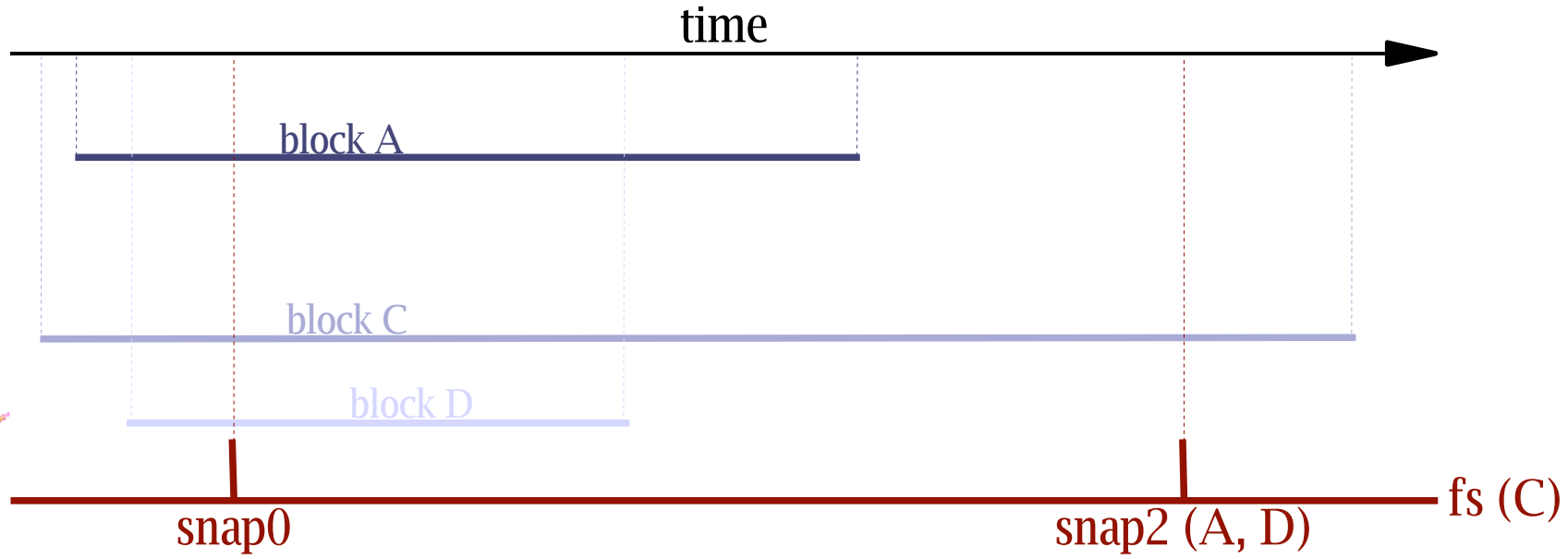
Snapshots

- to maintain snapshots ZFS:
  - keeps block birthtime with the block pointer
  - maintains list of dead blocks (blocks that were removed from the file system, but are still visible in one of the snapshots)
- when snapshot is destroyed we can free blocks that meet the following conditions:

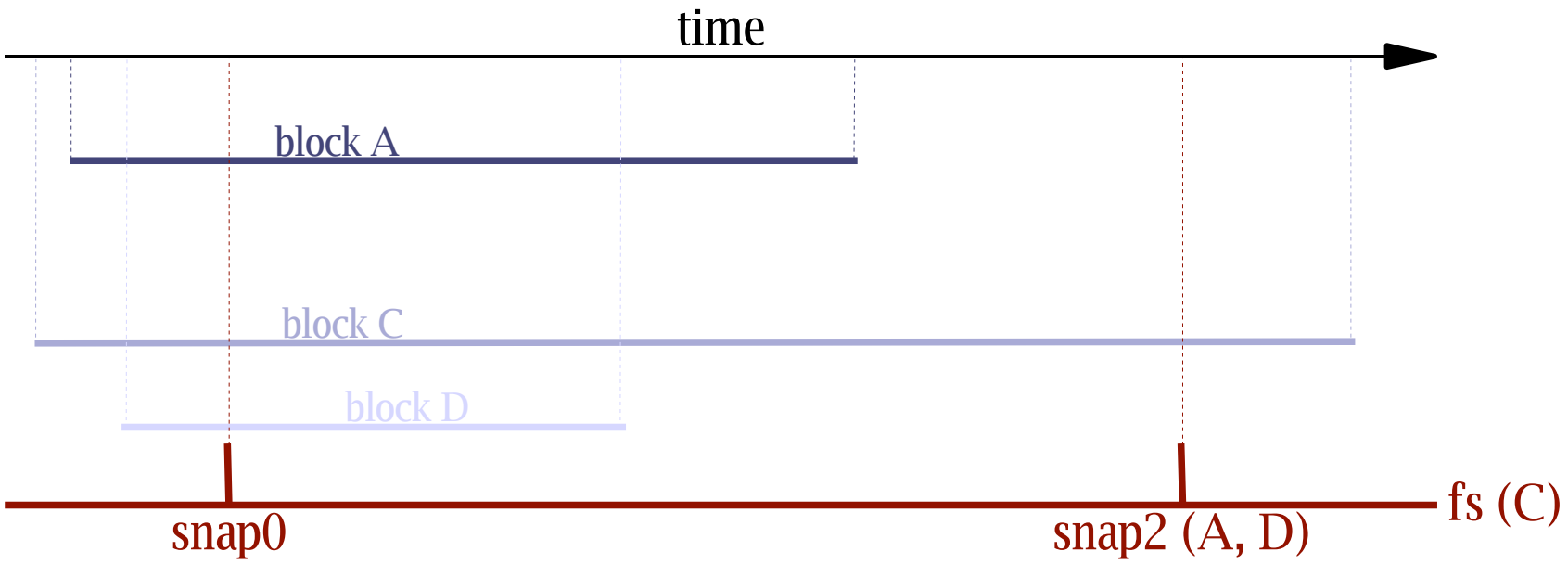
1) they were born after the previous snapshot
2) they were born before this snapshot
3) they died after this snapshot
4) they died before the next snapshot

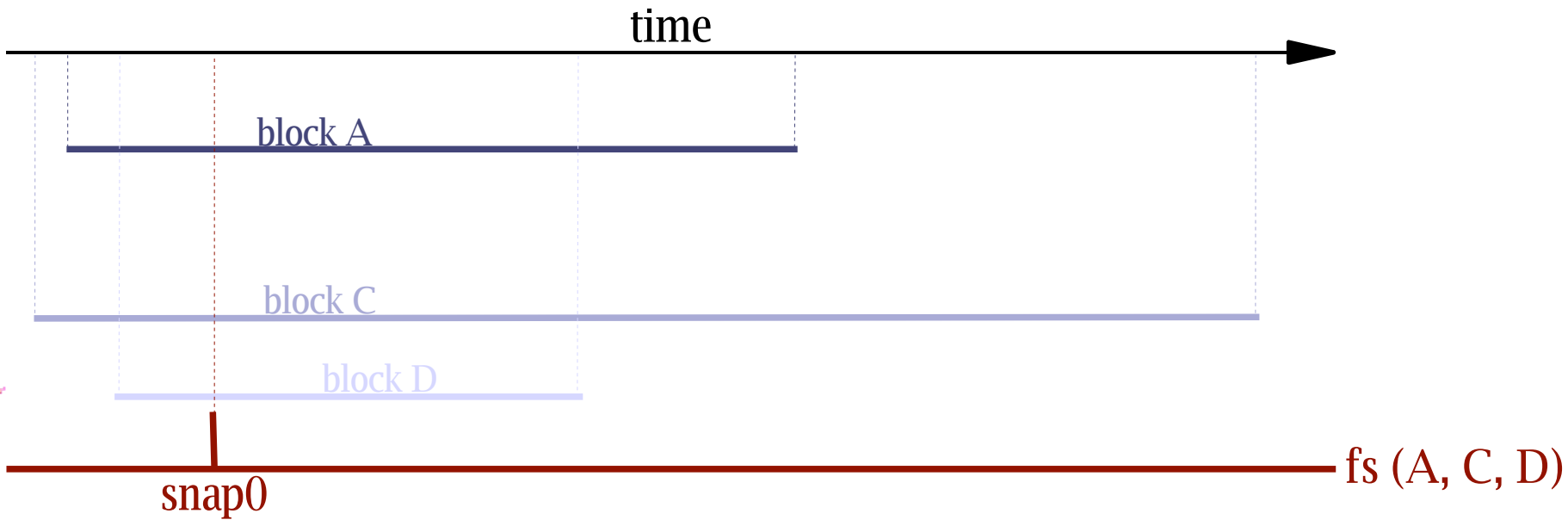
Destroying a snapshot 1/3



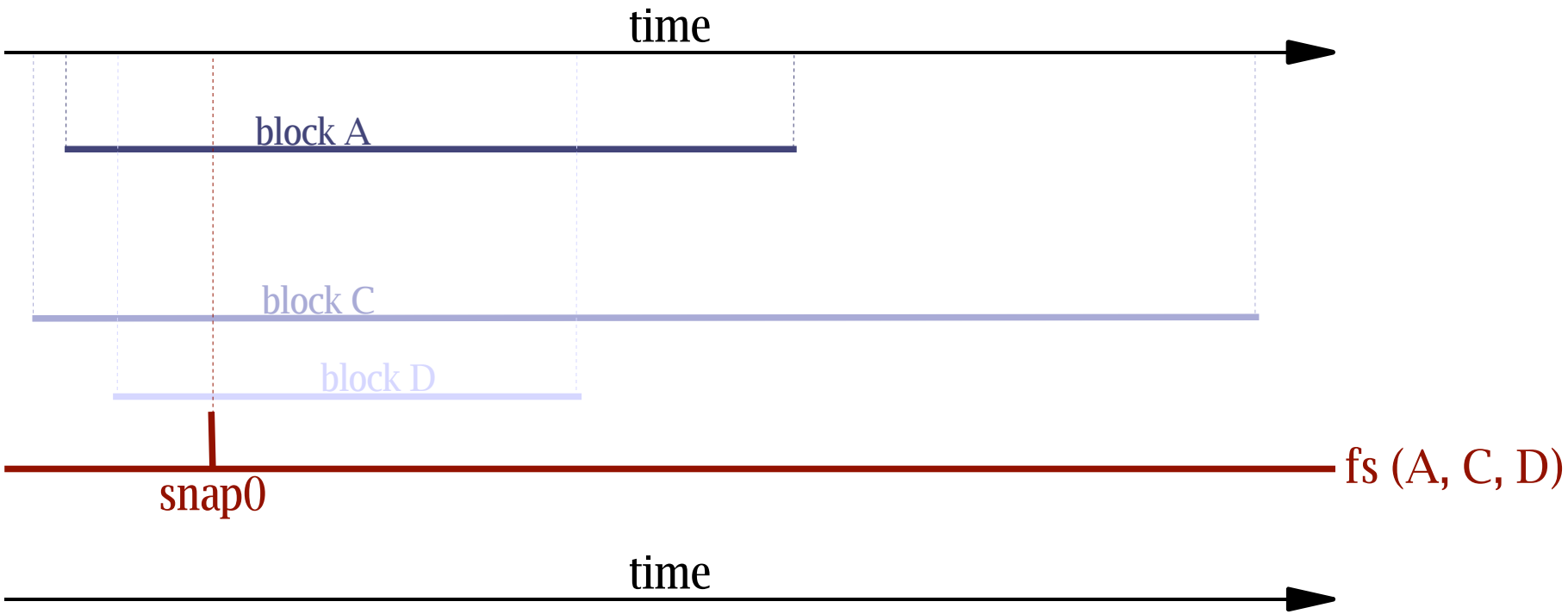


Destroying a snapshot 2/3





Destroying a snapshot 3/3



fs

Resilvering

- ZFS traverse metadata when synchronizing disks, which gives us the following benefits:
  - data integrity verification is done before the copy
  - copy only the live blocks
  - copy only the difference (in case of transient outage)

Status

ZFS/FreeBSD Status

- fresh and even more cool ZFS version waiting in perforce to be committed
- code-wise complete, but still needs testing
- some new features:
  - delegated administration
  - L2ARC
  - dedicated log vdevs
  - corrupted files list
  - stability improvements (no more kmem_map too small panics?)
  - ZFSboot
  - zpool properties
  - failure modes (wait, continue, panic)
  - when ZFS will be ready for production use?

Questions?