PAWEL DAWIDEK

BSDCAN 2023

OPENZFS BLOCK CLONING

FUDO SECURITY

HTTPS://FUDOSECURITY.COM

FUDO SECURITY

WHAT IS IT?

- on-demand deduplication
- accessible through copy_file_range(2) for file systems
- accessible through SCSI EXTENDED COPY for ZVOLs (not implemented)

```
ssize_t
copy_file_range(int infd, off_t *inoffp, int outfd, off_t *outoffp,
                size_t len, unsigned int flags);
```

BIG PICTURE

- we cannot modify BP (still!), so we need an additional table
- Block Reference Table (BRT, similar to DDT) per top-level vdev
- when we clone, we don’t read data blocks
- we don’t write any data, just add entries to the BRT or bump reference counters
- moving files between datasets doesn’t grow BRT (refcnt++, refcnt–)
- always on* (no additional cost when unused)
- need to consult BRT on every free (optimized)

BLOCK REFERENCE TABLE

- BRT entry: vdev id, offset, refcnt
- BRT per top-level vdev, so BRT entry: offset (the key), refcnt (the value)
- using micro ZAP, just like dedup

BLOCK CLONING VS DEDUPLICATION

- no cost on write
- requires no data
- works with any checksum algorithm (or without checksum)
- small, localized entries (based on offset, not random hash)
- no entries with a single reference
- on-demand
- always available (no dataset property)
- prefetchable

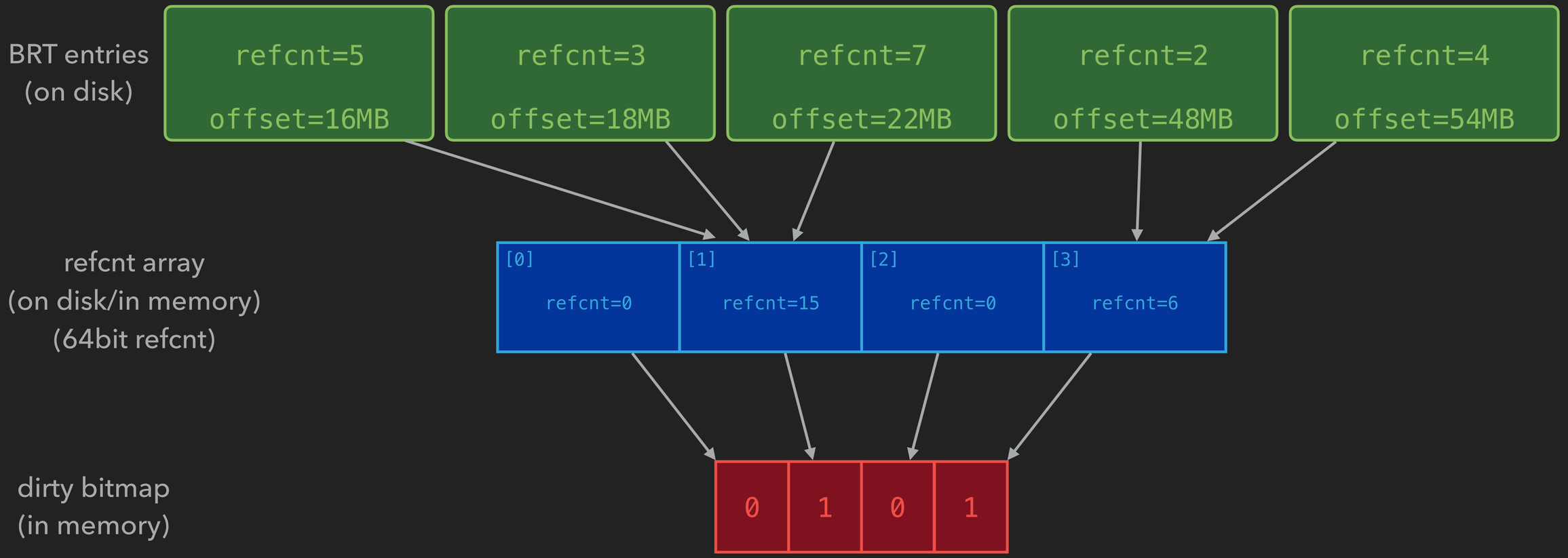
USE CASES

- space savings when cloning files
- space savings when recovering files from snapshots
- super-fast copy
- moving files between datasets

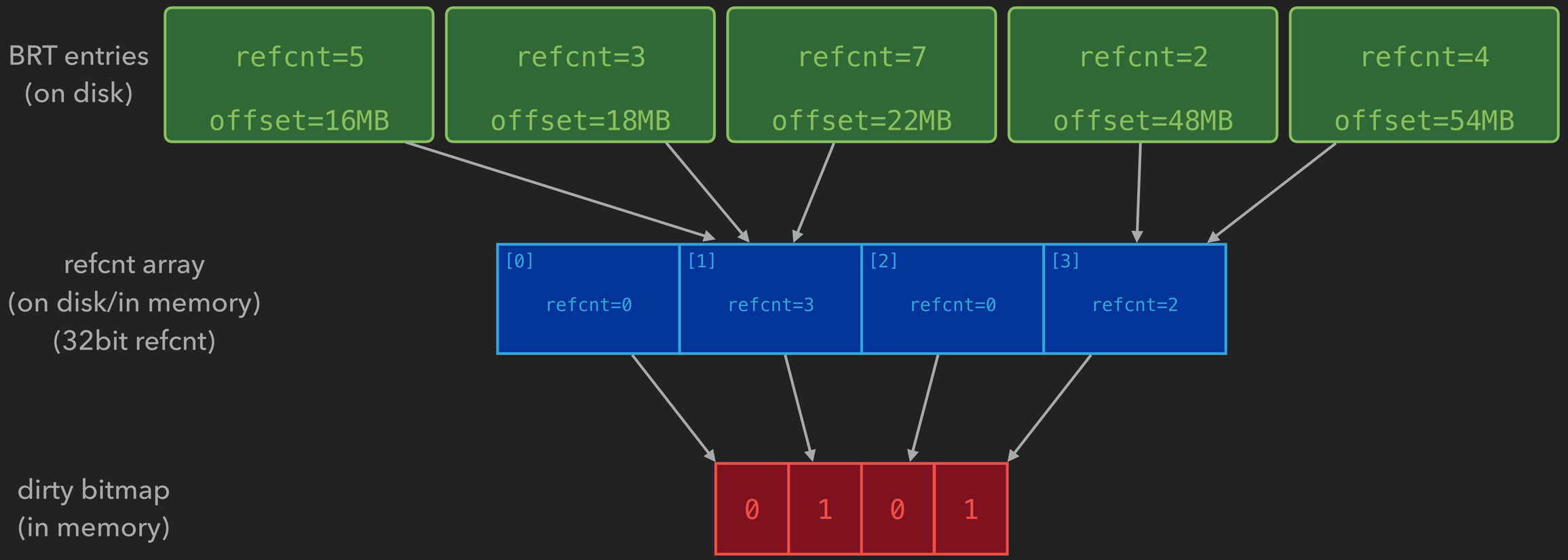
OPTIMIZING FREE

- we don’t want to perform BRT lookup on every free
- keep in-memory per top-level vdev array of refcnts per dirty region
- 128kB of memory per 1TB of storage for 16MB regions
- allows for: bool brt_maybe_exists(spa_t *spa, const blkptr_t *bp)
- sync to disk on every brt_sync()
- additional bitmap to not sync entire array (useful with smaller regions)

OPTIMIZING FREE



OPTIMIZING FREE



BLOCK CLONING / DEDUPLICATION INTERACTION

- imagine a block that exists in both DDT and BRT
- on free, which refcnt should we decrement first?

solution:

- when D flag is set, don’t create BRT entry, just increase DDT entry’s refcnt (@allanjude)

CROSSING DATASET BOUNDARY

- perfectly doable in general case
- not supported when datasets use different encryption keys
- not supported when one dataset is encrypted and the other is not
- makes ZIL problematic

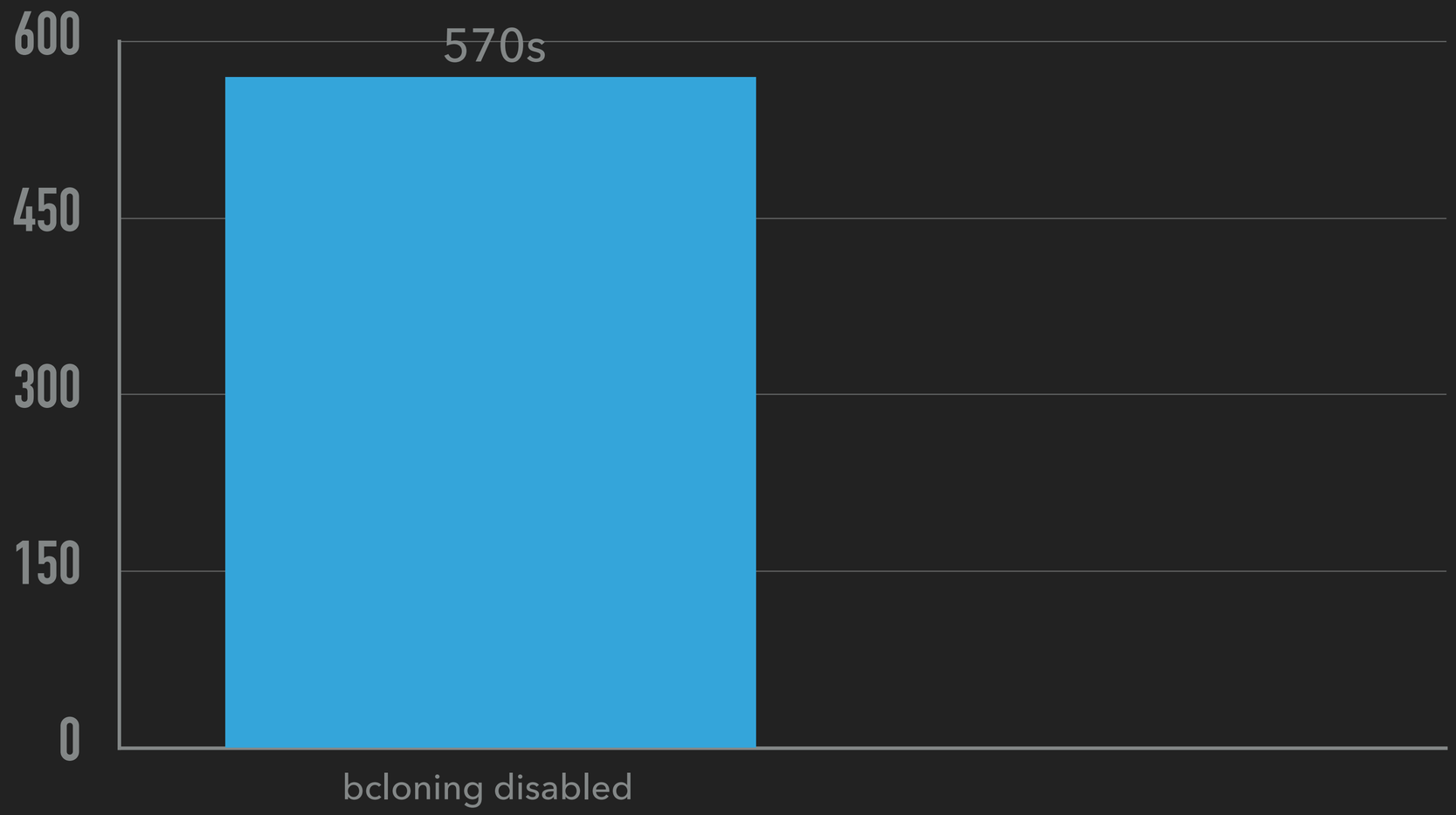
ZIL AND BLOCK CLONING

- need separate functions to do the cloning and to replay the log
- source and destination files may reside on separate datasets
- we store BPs in the log, but they may disappear

solutions:

- zil_claim()

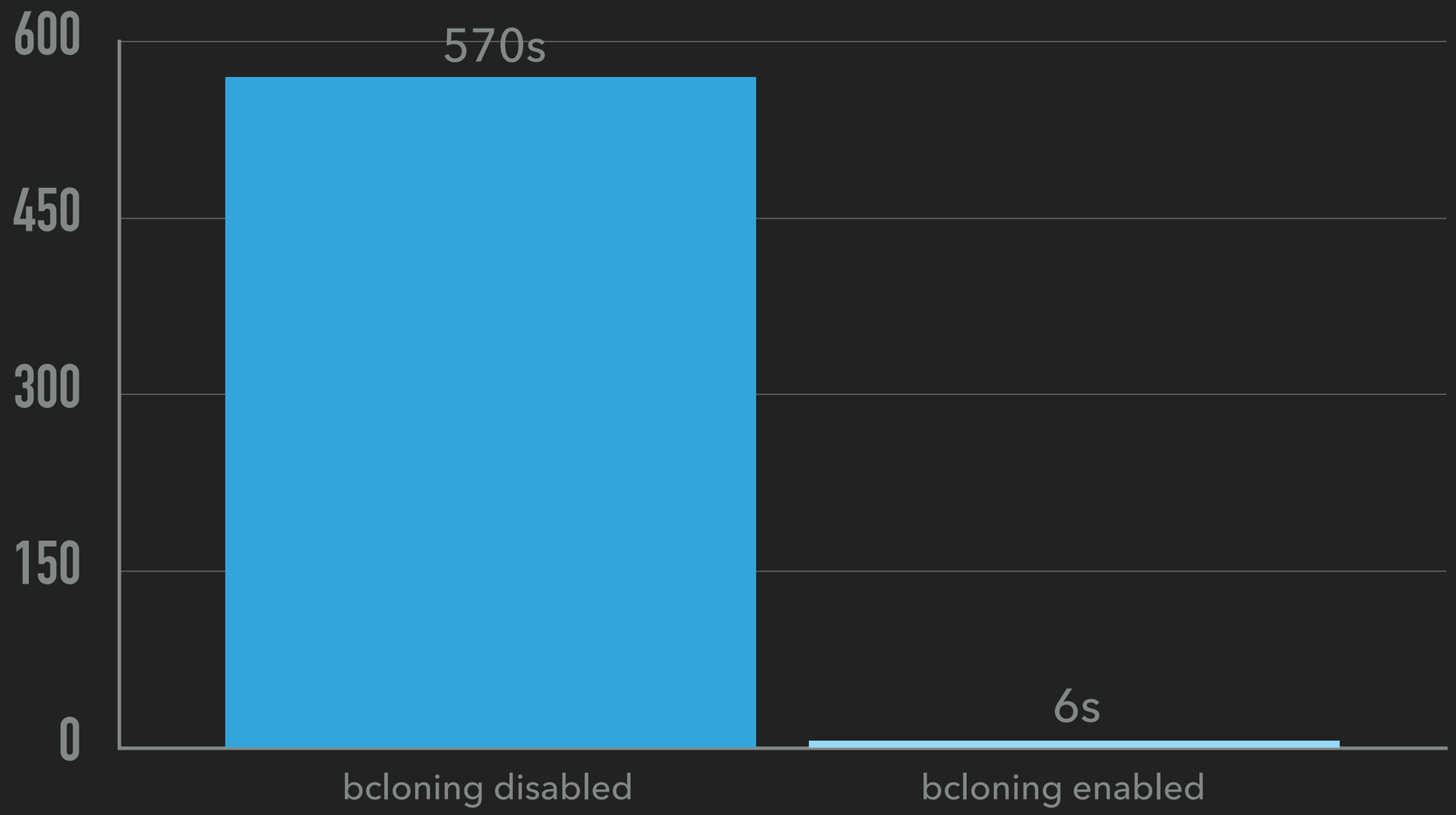
PERFORMANCE (ZPOOL IMPORT && 32GB FILE COPY && ZPOOL SYNC)



single SSD pool (raw read: 405MB/s, raw write: 492MB/s)

FUDO SECURITY

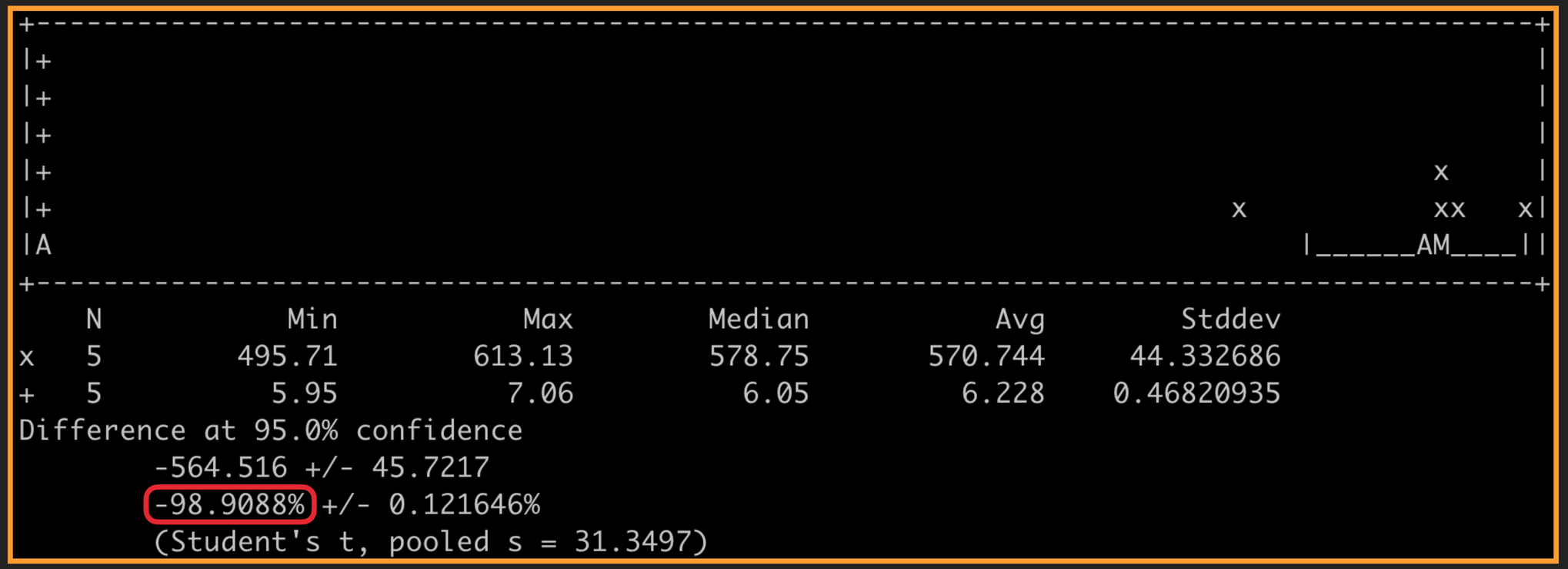
PERFORMANCE (ZPOOL IMPORT && 32GB FILE COPY && ZPOOL SYNC)



single SSD pool (raw read: 405MB/s, raw write: 492MB/s)

FUDO SECURITY

PERFORMANCE (32GB FILE COPY)



FUDO SECURITY

STATISTICS

- three new zpool properties

```
openzfs:root:~# zpool get bcloneused,bclonesaved,bcloneratio tank
NAME  PROPERTY           VALUE              SOURCE
tank  bcloneused         4G                 -
tank  bclonesaved        8G                 -
tank  bcloneratio        3.00x              -
```

SPECIAL CASES

- the block we want to clone have been created in the same txg
- the block we want to clone have been modified in the same txg
- the cloned block may be fully overwritten in the same txg
- the cloned block may be partially overwritten in the same txg
- dbuf size may not match even if the file is empty
- the block may be cloned multiple times in the same txg

SPECIAL CASES

- block size does not match
- the block may be cloned and freed in the same txg
- the block may be cloned and the clone may be cloned again in the same txg
- some of the blocks might have embedded data or be holes
- the file might have been deleted, but the caller still has a file descriptor open to the file and clones it
- … to be discovered

RANDOM NOTES

- vdev growing is supported, shrinking is not
- no entries in MOS when unused or no longer used
- offsets and length in copy_file_range(2) has to be recordsize-aligned (for now)
- cloning into non-empty file with different recordsize is a problem
- copy_file_range(2) operates within a single mount point (for now)
- we lose all the savings on zfs send/recv (turn into dedup?)

BLOCK CLONING - STATUS

- merged to main
- experimental for the time being (enable using sysctl)
- cp(1) is already using copy_file_range(2)
- patches for mv(1) and install(1)
- patches for cross-mount-point copy_file_range(2)

BLOCK CLONING – Q&A