Squishing Squids

A Ceph Compression Guide

Bryan Stillwell (Clyso)
Ceph Solutions Architect

March 25, 2026

CL'SO

Outline

- Why compression matters
- Where compression can be used in Ceph
- Hitting the algorithmic sweet spot
- Monitoring compression efficiency
- Common pitfalls

Why Compression Matters

Economic Impact (TCO)

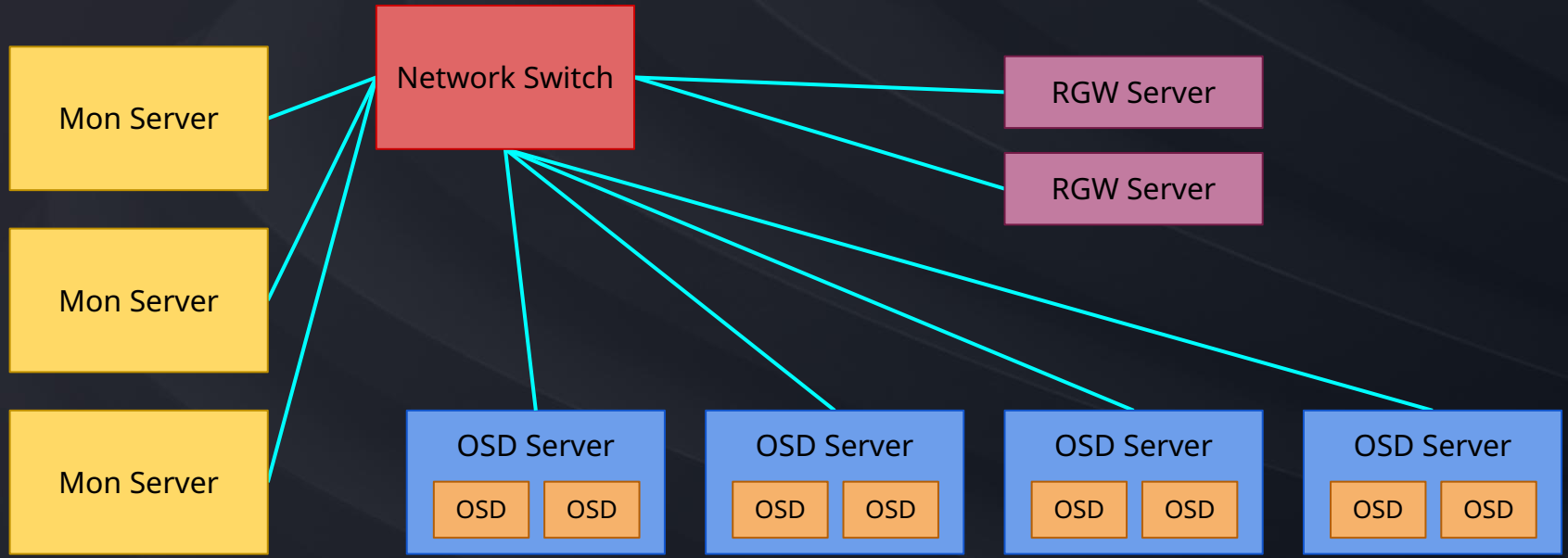
- Reduced CapEx
  - Store more on the same hardware
- Lower OpEx
  - Fewer drives need less power and cooling

Why Compression Matters

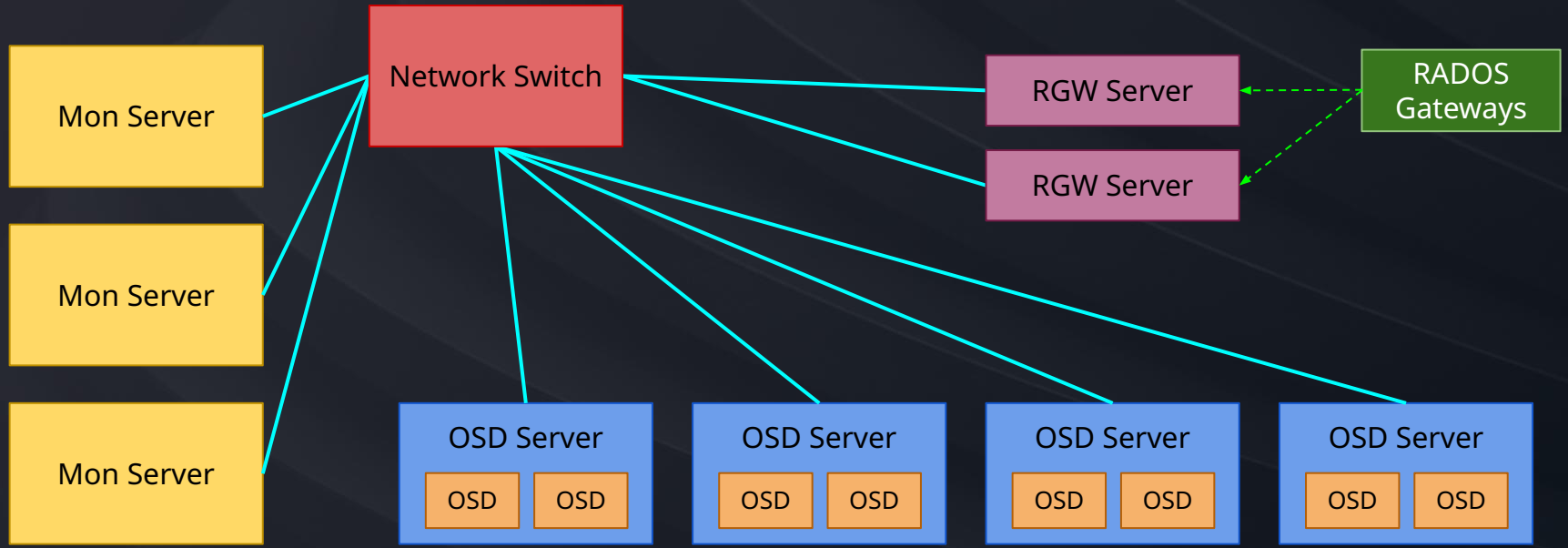
Performance & Efficiency

- Reduced flash wear and tear
  - Writing fewer bytes reduces total bytes written
- Network bandwidth optimization
  - Less data needs to be replicated between OSDs speeding up recovery and balancing
- Improved throughput
  - HDDs can achieve higher throughput than physical limits
    Example: 3x compression can write 300 MB/s to a drive at 100 MB/s

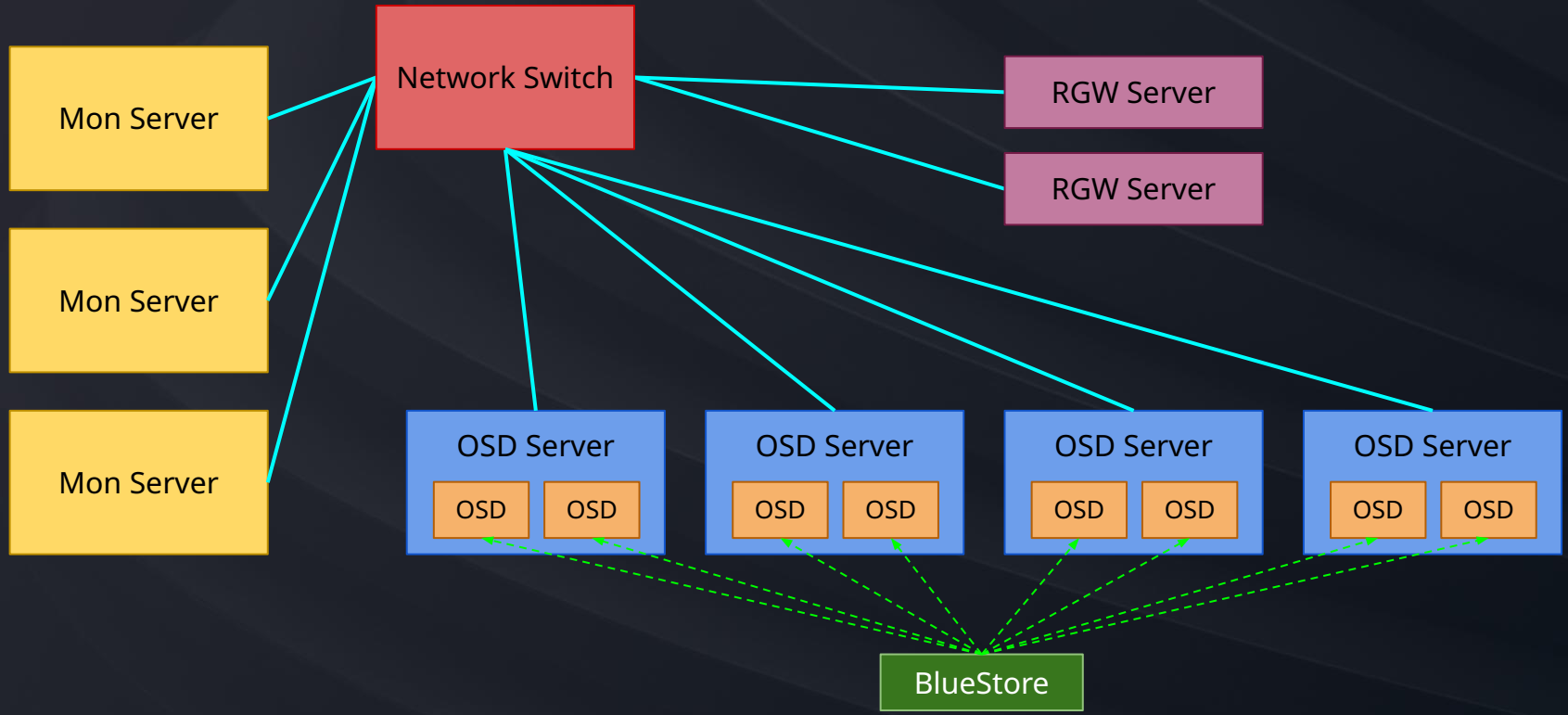
Where compression can be used in Ceph



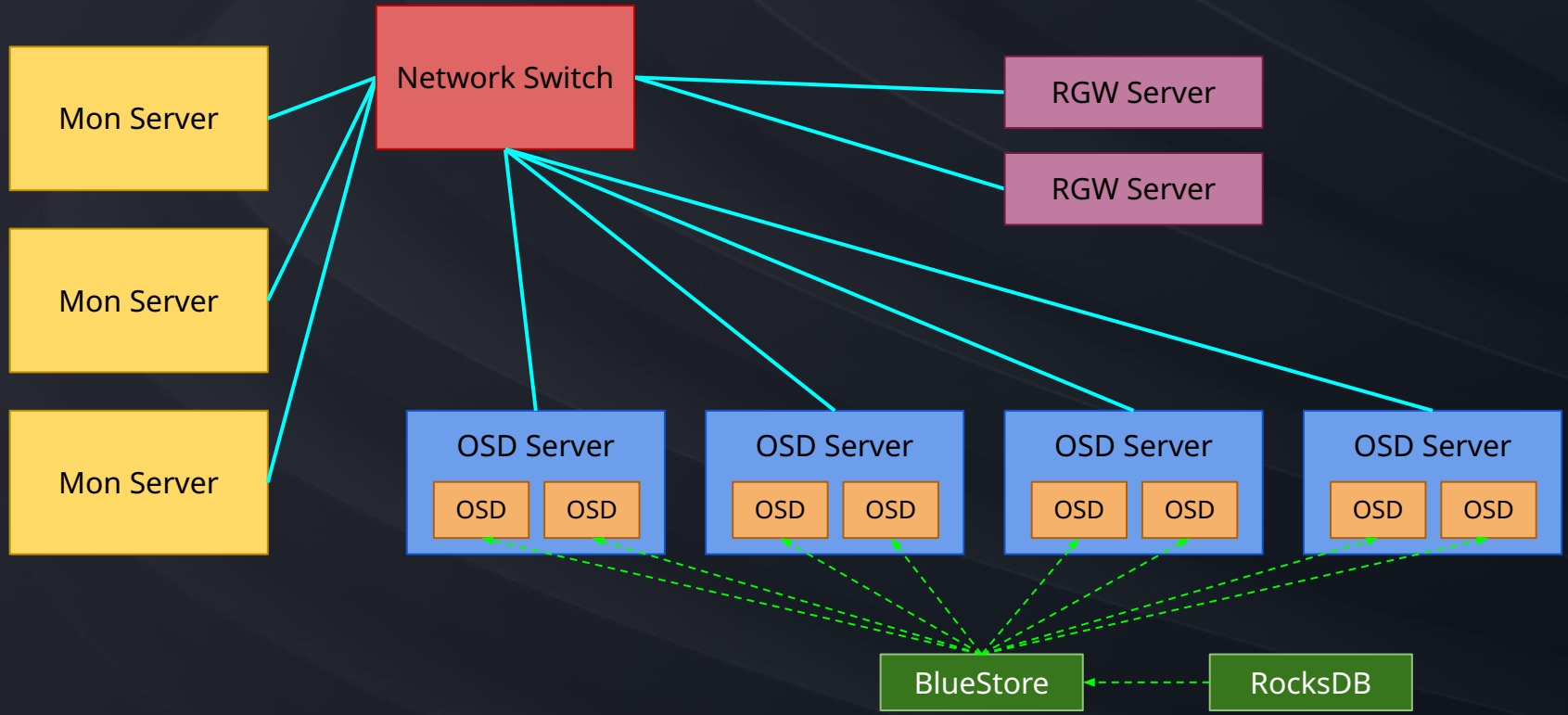
Where compression can be used in Ceph



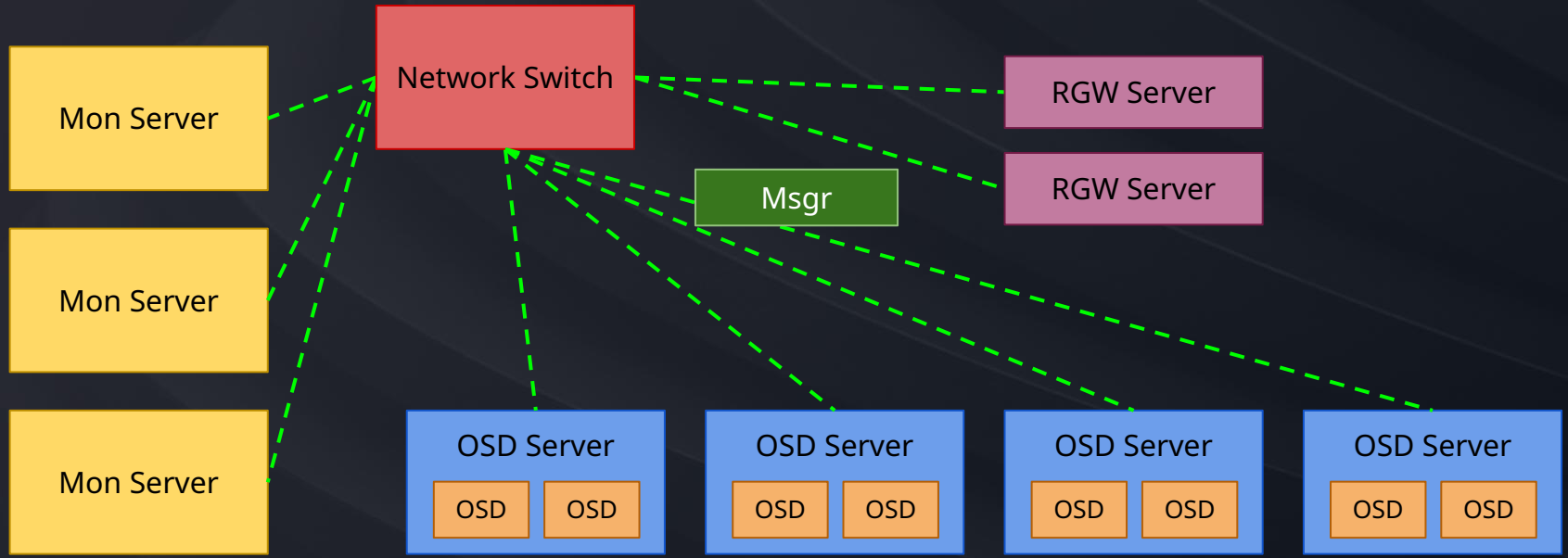
Where compression can be used in Ceph



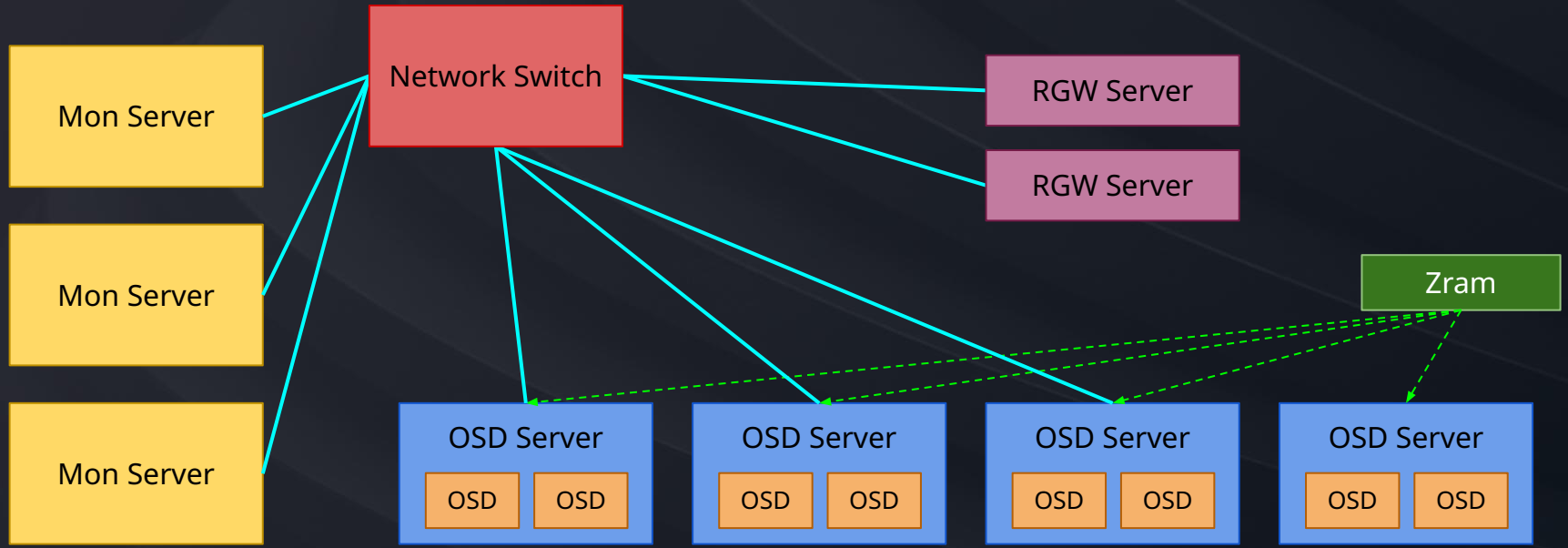
Where compression can be used in Ceph



Where compression can be used in Ceph



Where compression can be used in Ceph



Checking compressibility

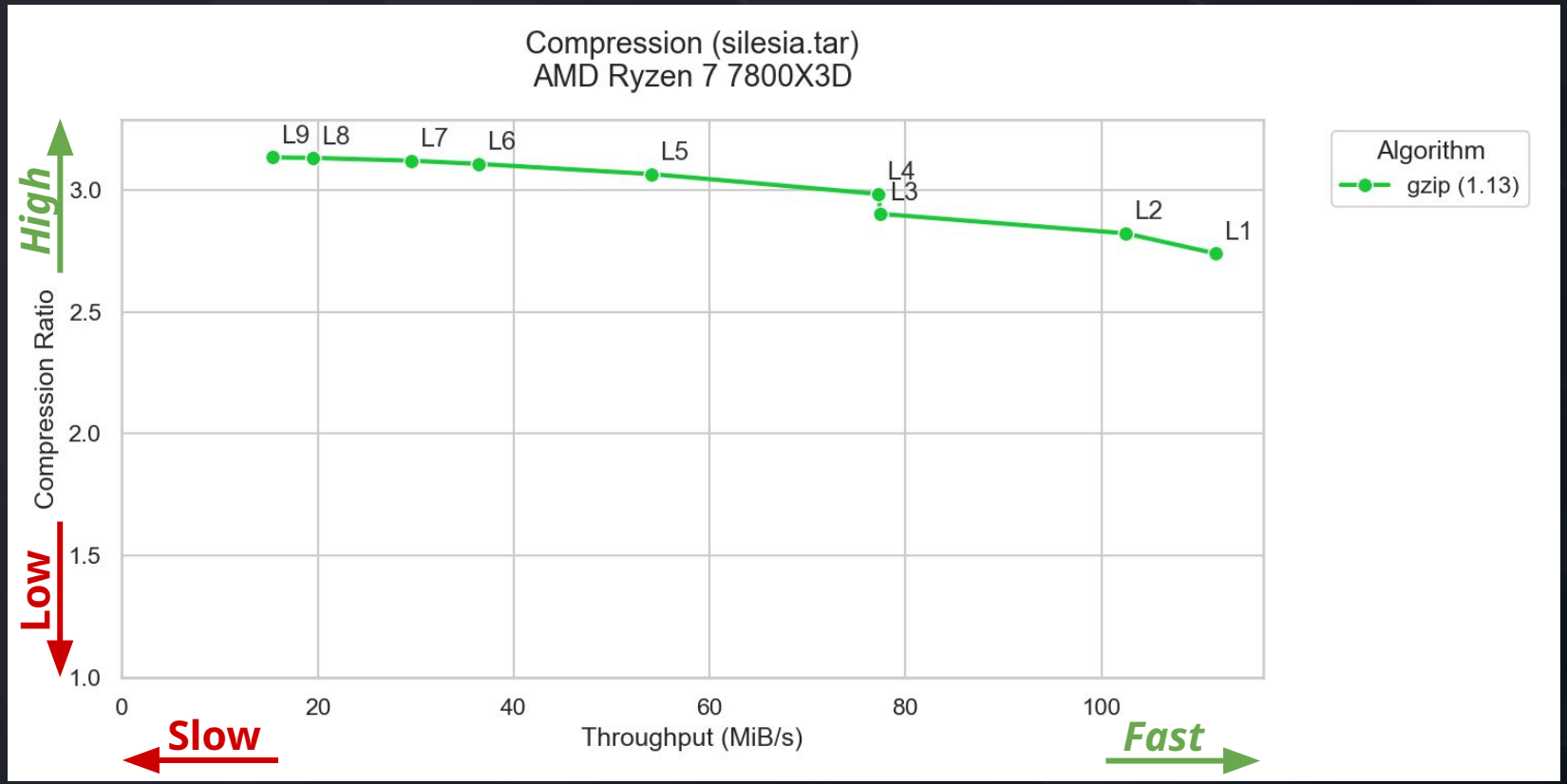
Before getting started, can my data be compressed?

```
$ ./pool-compressibility.py --pool default.rgw.buckets.data
[...snip...]
Elapsed    | Objects    | Data Scanned    | Avg Ratio    | Factor
------------------------------------------------------------------------
1s         | 1          | 4194304         | 35.91%       | 2.79x
[...snip...]
FINAL SUMMARY
=============================
Time elapsed: 30.69s
Objects checked: 38
Total Raw Bytes: 155189248
Global Compression Ratio: 33.65%
Compression Factor: 2.97x
Potential Space Savings: 66.35%
```

https://github.com/bstillwell/compression-testing/blob/main/pool-compressibility.py

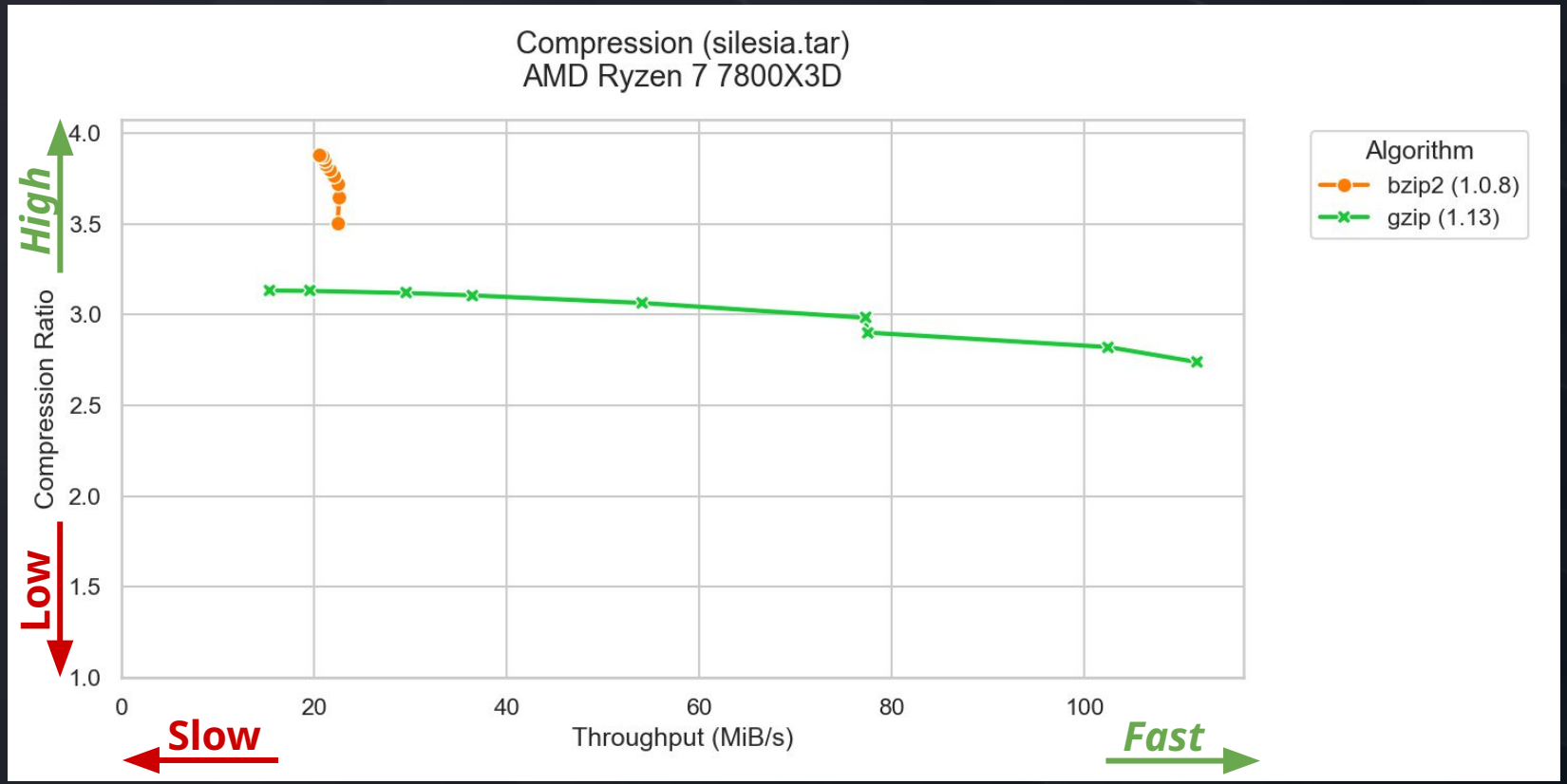
CLISO

The algorithms (gzip)

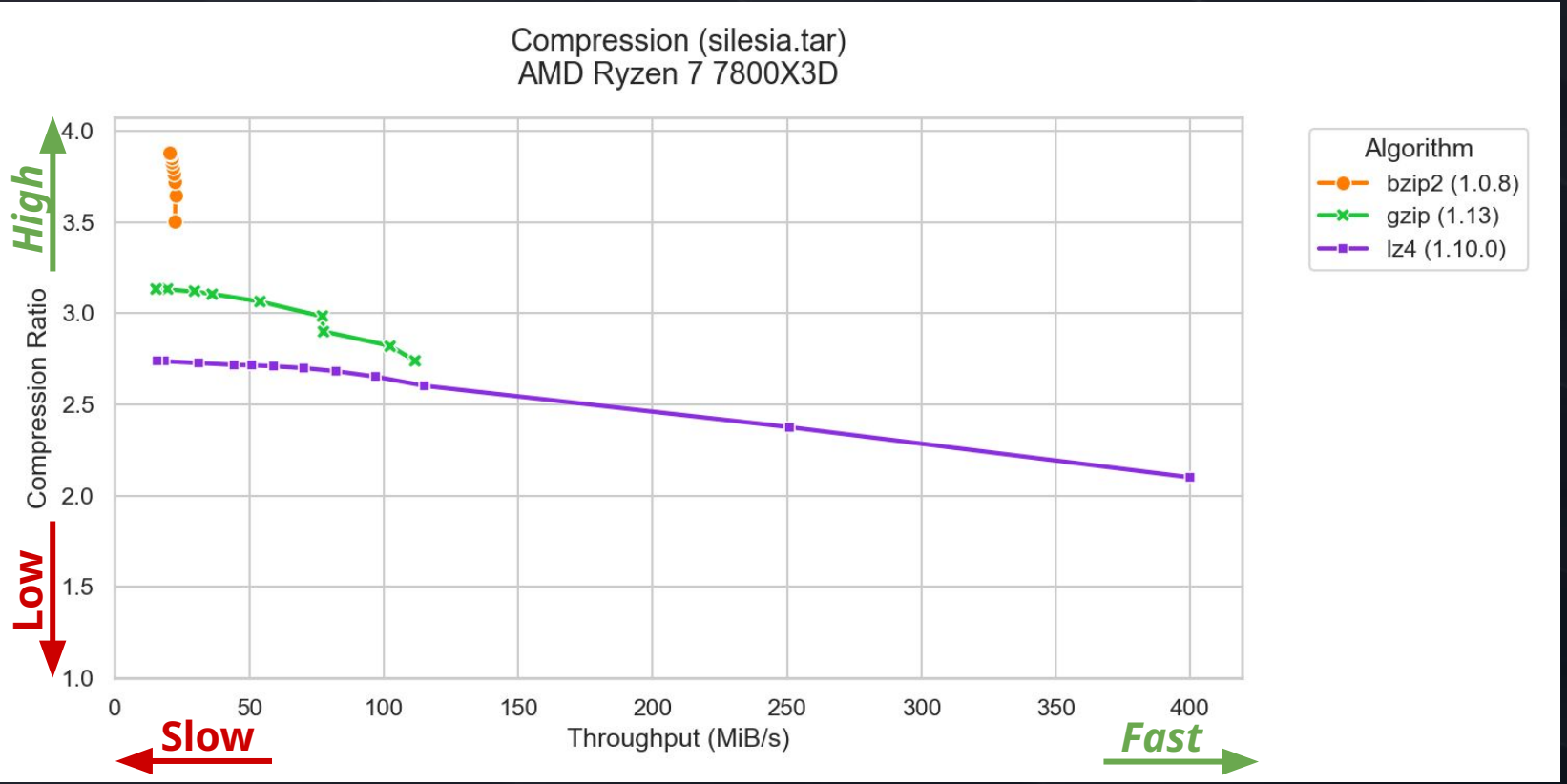


CL'SO

The algorithms (bzip2)

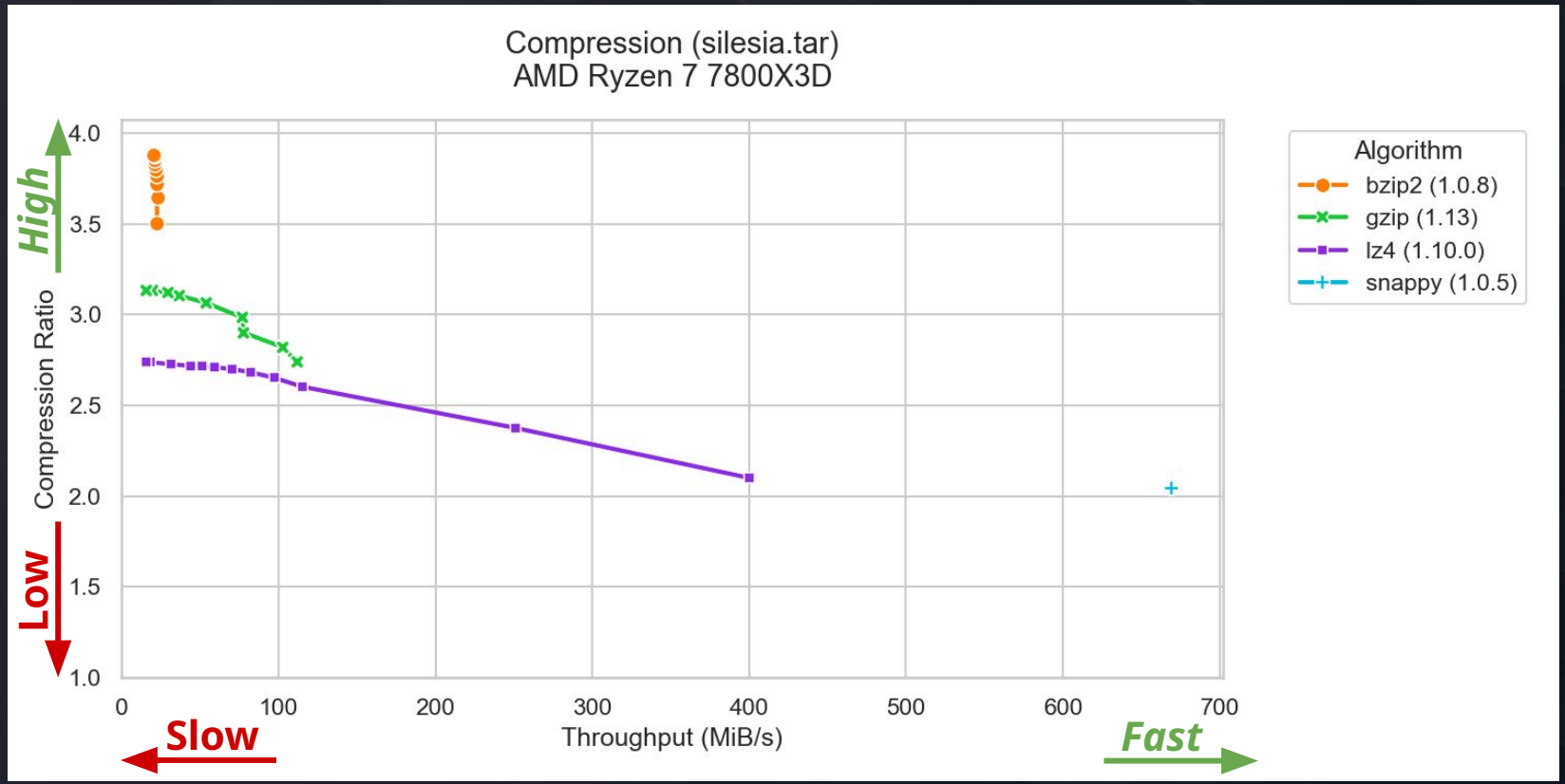


The algorithms (lz4)

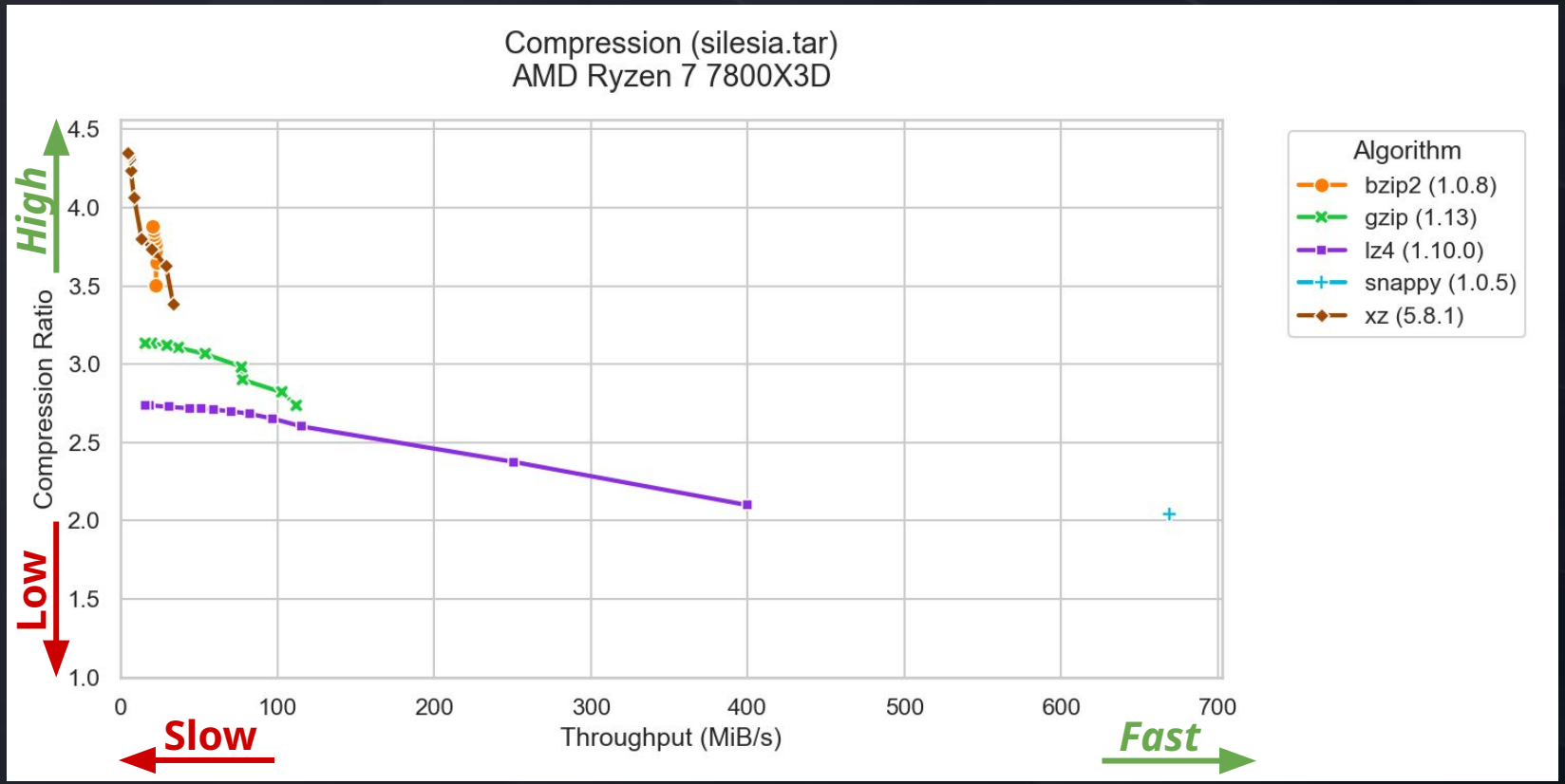


CL'SO

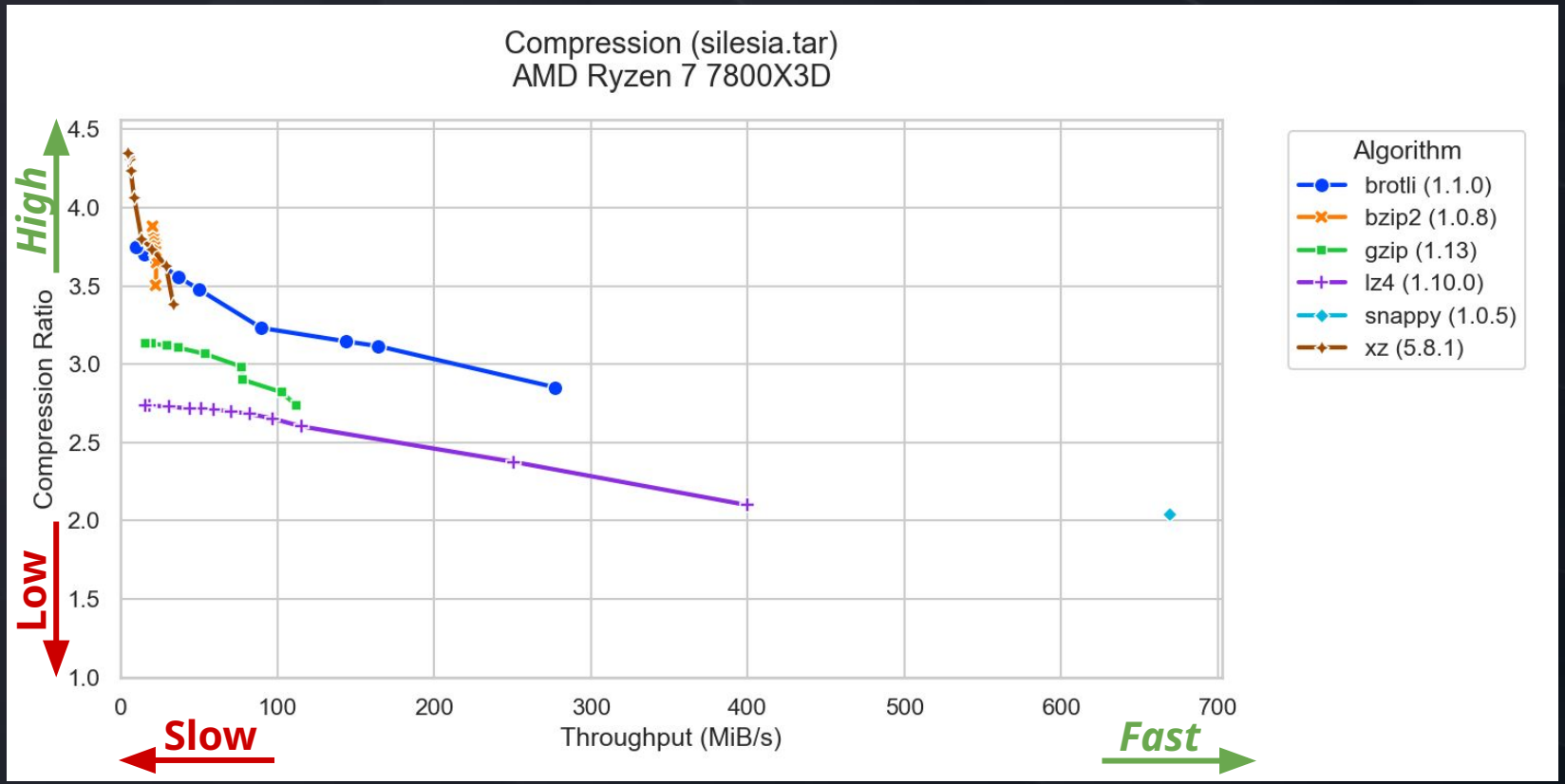
The algorithms (snappy)



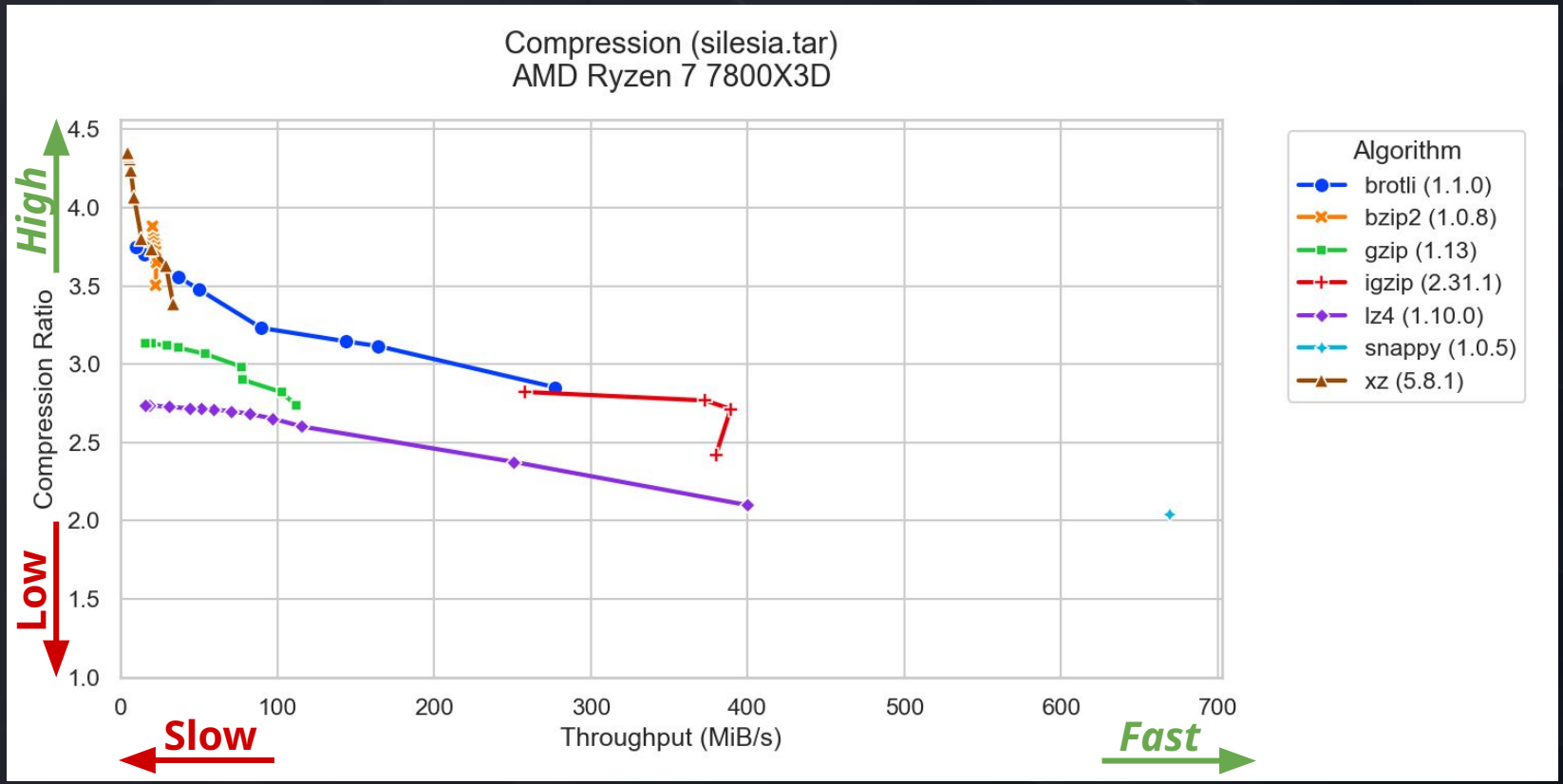
The algorithms (xz)



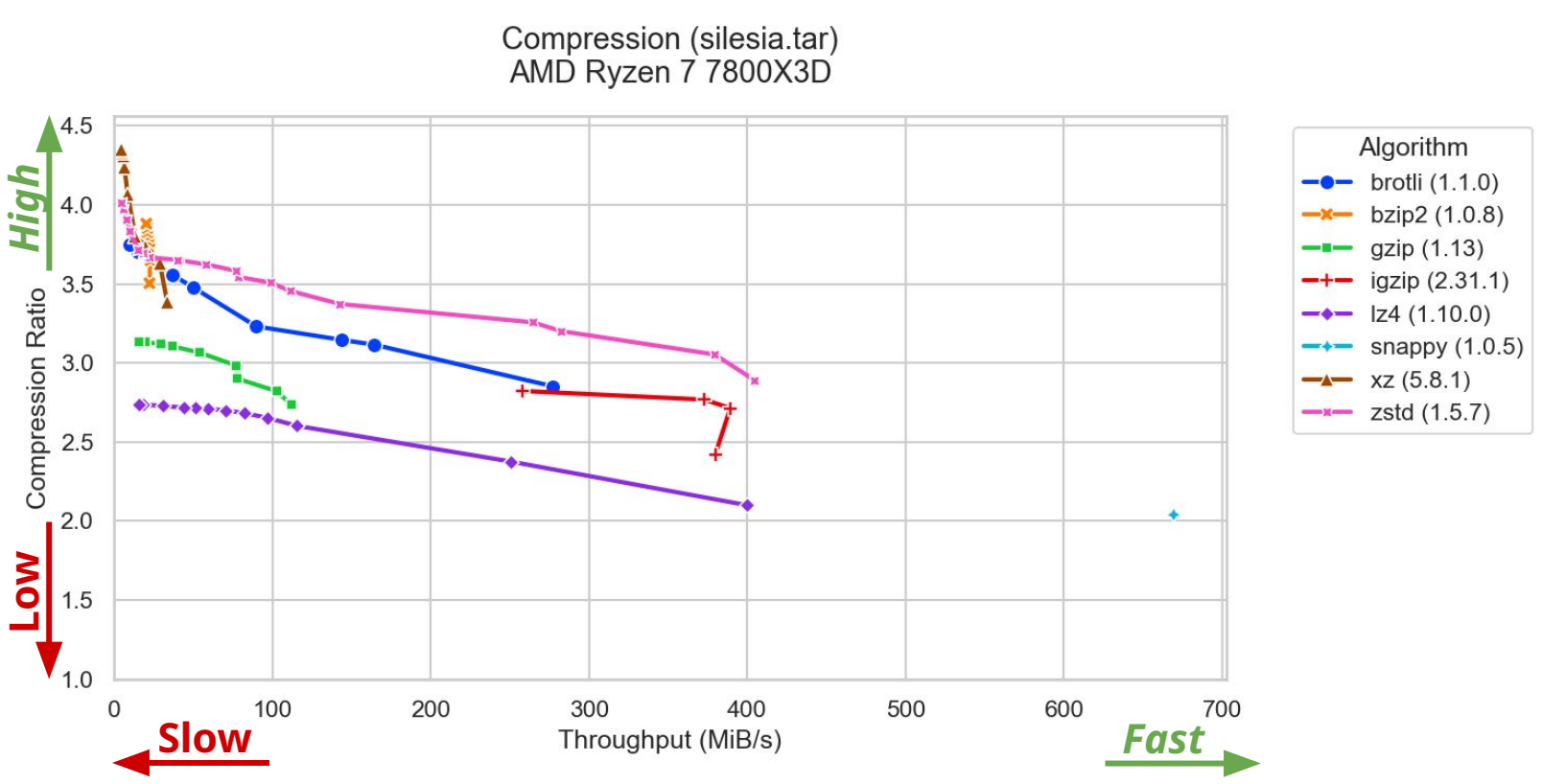
The algorithms (brotli)



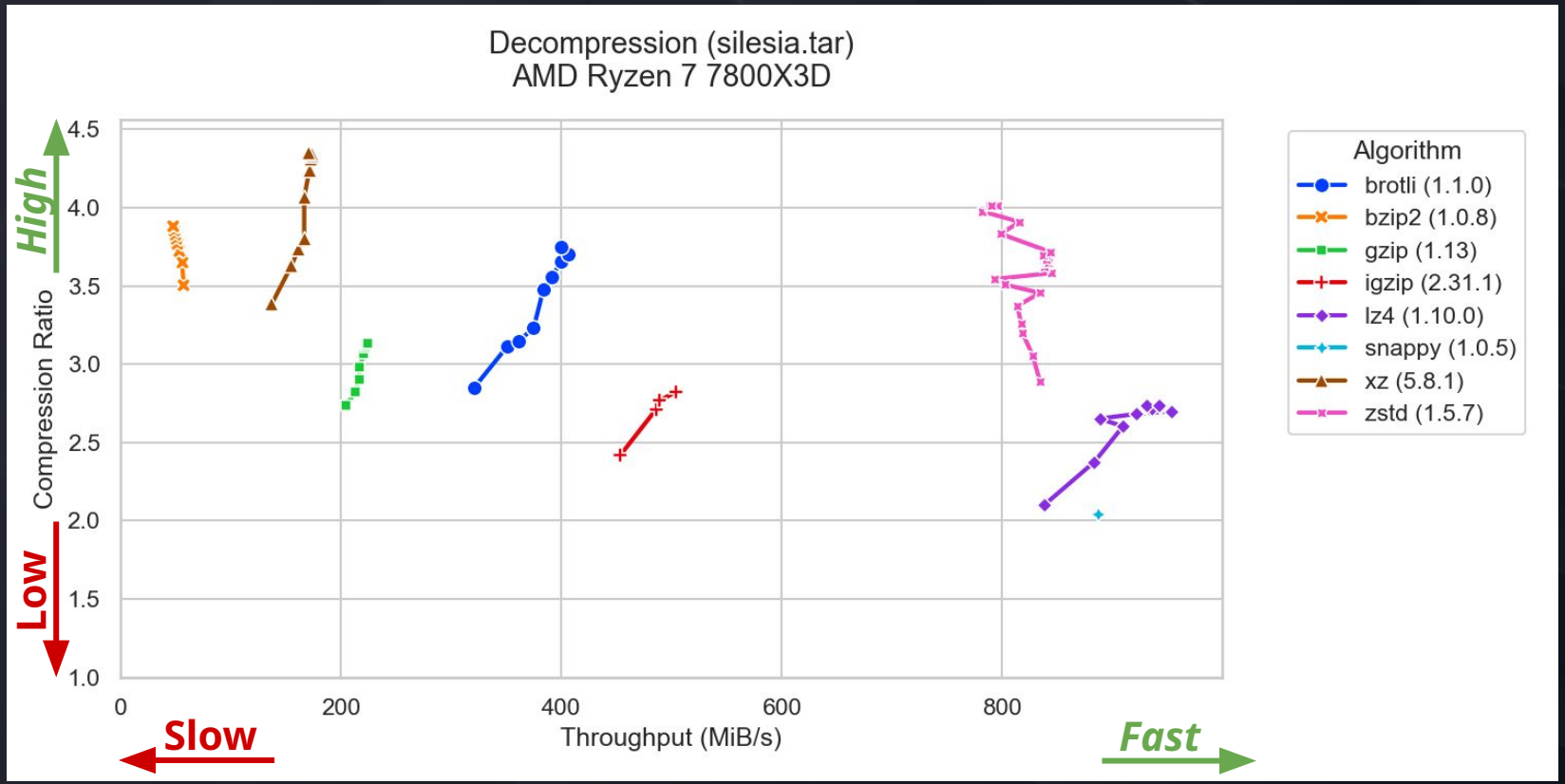
The algorithms (igzip)



The algorithm sweet spot (zstd)



The algorithm sweet spot (zstd)



Configuring compression

BlueStore (OSDs)

```
# Enable compression
ceph osd pool set <pool_name> compression_mode aggressive
ceph osd pool set <pool_name> compression_algorithm zstd

# Verify it's working
ceph df detail

# Disable compression
ceph osd pool set <pool_name> compression_mode none
ceph osd pool set <pool_name> compression_algorithm none
```

Configuring compression

RADOS Gateways (RGWs)

```
# Enable compression
radosgw-admin zone placement modify  \
    --rgw-zone default               \
    --placement-id default-placement \
    --compression zstd

# Restart RGWs

# Disable compression
radosgw-admin zone placement modify  \
    --rgw-zone default               \
    --placement-id default-placement \
    --compression none
```

Adjusting compression levels

Compression levels

```
# Set zlib compression level (default: 5)
ceph config set global compressor_zlib_level 1

# Set zstd compression level (default: 1)
ceph config set global compressor_zstd_level 1

# lz4 compression level (default: 1)
Not configurable

# snappy compression level (default: N/A)
Not configurable
```

CL'SO

Manual RGW benchmarking

No compression

```
# Use s5cmd to do multipart uploads in parallel
$ s5cmd --endpoint-url http://172.24.88.46 cp --show-progress enwik9 s3://benchmark
100.00%  ═══════════════════════════════════════  1.00 GB / 1.00 GB (84.97 MB/s) 12s (1/1)

# View pool stats
$ ceph df | grep -e STORED -e default.rgw.buckets.data
POOL                        ID  PGS   STORED  OBJECTS     USED  %USED  MAX AVAIL
default.rgw.buckets.data    21   32  954 MiB      249  2.8 GiB   0.02    4.6 TiB

# Remove object
$ s5cmd --endpoint-url http://172.24.88.46 rm s3://benchmark/enwik9

# Perform garbage collection
radosgw-admin gc process --include-all
```

Test data: https://mattmahoney.net/dc/enwik9.zip

CL'SO

Manual RGW benchmarking

BlueStore compression (zstd)

```
# Use s5cmd to do a parallel upload
$ s5cmd --endpoint-url http://172.24.88.46 cp --show-progress enwik9 s3://benchmark
100.00%  ═══════════════════════════════════════  1.00 GB / 1.00 GB (48.32 MB/s) 21s (1/1)

# View pool stats
$ ceph df | grep -e STORED -e default.rgw.buckets.data
POOL                        ID  PGS   STORED  OBJECTS     USED  %USED  MAX AVAIL
default.rgw.buckets.data    21   32  954 MiB      249  1.5 GiB   0.01    4.6 TiB

# Remove object
$ s5cmd --endpoint-url http://172.24.88.46 rm s3://benchmark/enwik9

# Perform garbage collection
radosgw-admin gc process --include-all
```

Test data: https://mattmahoney.net/dc/enwik9.zip

Manual RGW benchmarking

RGW compression (zstd)

```
# Use s5cmd to do a parallel upload
$ s5cmd --endpoint-url http://172.24.88.46 cp --show-progress enwik9 s3://benchmark
100.00%  ═══════════════════════════════════════  1.00 GB / 1.00 GB (152.01 MB/s) 6.8s (1/1)

# View pool stats
$ ceph df | grep -e STORED -e default.rgw.buckets.data
POOL                        ID  PGS   STORED  OBJECTS     USED  %USED  MAX AVAIL
default.rgw.buckets.data    21   32  341 MiB       97 1022 MiB      0    4.6 TiB

# Remove object
$ s5cmd --endpoint-url http://172.24.88.46 rm s3://benchmark/enwik9

# Perform garbage collection
radosgw-admin gc process --include-all
```

Test data: https://mattmahoney.net/dc/enwik9.zip

CL'SO

Manual benchmark results

	Throughput	Stored	Used	Objects
No compression	84.97 MB/s	954 MiB	2.8 GiB	249
BlueStore (zstd)	48.32 MB/s	954 MiB	1.5 GiB	249
RGW (zstd)	152.01 MB/s	341 MiB	1,022 MiB	97

- BlueStore compression (3x replica == triple compression)
- RGW compression best
  - Highest throughput (faster uploads)
  - Highest compression (less space used)
  - Fewest objects (speeds up recovery/balancing)

CL'SO

Multi-part upload failure cleanup

Multi-part failures

```
# List incomplete multipart uploads (needs ~/.s3cfg)
$ s3cmd multipart s3://benchmark
s3://benchmark/
Initiated Path Id
2026-02-23T14:19:35.851Z s3://benchmark/enwik9    2~zmdk7JaZ-huRL_1GNi04wehYsvdBxwE

# Remove the failed upload
$ s3cmd abortmp s3://benchmark/enwik9 2~zmdk7JaZ-huRL_1GNi04wehYsvdBxwE

# Perform garbage collection
$ radosgw-admin gc process --include-all
```

CL'SO

AWS S3 Storage Classes

Storage Class	Use Case	Access Pattern	Retrieval Time	Pricing/Month (early 2026)*
Express One Zone	Latency-sensitive apps	Very frequent	Single-digit ms	$0.11/GB
Standard	General purpose	Frequent	Milliseconds	$0.021-$0.023/GB
Intelligent-Tiering	Good starting point	Hot initially, cools down over 30 days	Milliseconds	Mix of Standard + Standard-IA
Standard-IA	Long-lived, backups	Infrequent	Milliseconds	$0.0125/GB
One Zone-IA	Recreatable, less critical data	Infrequent	Milliseconds	$0.01/GB

**** Does not include requests & data retrieval costs***

Ceph S3 Storage Classes

Express One Zone (3x replicas on NVMe)

```
# Add the storage class
radosgw-admin zonegroup placement add \
    --rgw-zonegroup default           \
    --placement-id default-placement  \
    --storage-class EXPRESS_ONEZONE

# Define the zone-specific placement for the storage class
radosgw-admin zone placement add                \
    --rgw-zone default                          \
    --placement-id default-placement            \
    --storage-class EXPRESS_ONEZONE             \
    --data-pool default.rgw.buckets.data_nvme
```

CL'SO

Ceph S3 Storage Classes

Standard (3x replicas on HDDs)

```
# Add the storage class
radosgw-admin zonegroup placement add \
    --rgw-zonegroup default           \
    --placement-id default-placement  \
    --storage-class STANDARD

# Define the zone-specific placement for the storage class
radosgw-admin zone placement add                \
    --rgw-zone default                          \
    --placement-id default-placement            \
    --storage-class STANDARD                    \
    --data-pool default.rgw.buckets.data        \
    --compression snappy
```

Ceph S3 Storage Classes

Standard-IA (8+3 erasure coding on HDDs)

```
# Add the storage class
radosgw-admin zonegroup placement add \
    --rgw-zonegroup default           \
    --placement-id default-placement  \
    --storage-class STANDARD_IA

# Define the zone-specific placement for the storage class
radosgw-admin zone placement add                \
    --rgw-zone default                          \
    --placement-id default-placement            \
    --storage-class STANDARD_IA                 \
    --data-pool default.rgw.buckets.data_ec83 \
    --compression zstd
```

CL'SO

Ceph S3 Storage Classes

Intelligent-Tiering (Standard initially, then Standard-IA after 30 days)

```
# Create life-cycle policy (lc-intelligent-tiering.xml)
<LifecycleConfiguration>
    <Rule>
        <ID>Migrate data to STANDARD_IA after 30 days</ID>
        <Prefix></Prefix>
        <Status>Enabled</Status>
        <Transition>
            <Days>30</Days>
            <StorageClass>STANDARD_IA</StorageClass>
        </Transition>
    </Rule>
</LifecycleConfiguration>
```

CL'SO

Ceph S3 Storage Classes

Intelligent-Tiering (Standard initially, then Standard-IA after 30 days)

```
# Set the lifecycle policy for a bucket
s3cmd setlifecycle lc-intelligent-tiering.xml s3://benchmark
```

**** Only partially working currently***

Data is moved between pools, but not compressed/re-compressed

Bug Filed: https://tracker.ceph.com/issues/75411

Pending fix: https://github.com/ceph/ceph/pull/67725

CL'SO

Benchmarking scripts

```
[2026-02-20 16:52:59] ✔ [bryan@venetian:/home/bryan/src/compression-testing]$ ./benchmark-compression.py

--- Zstandard (Binary: zstd) ---
[Finished] zstd [1.5.7] Lvl 1                | Comp:   428.46 MiB/s | Decomp:   925.45 MiB/s | Ratio:  2.89x
[Finished] zstd [1.5.7] Lvl 2                | Comp:   380.09 MiB/s | Decomp:   871.22 MiB/s | Ratio:  3.05x
[Finished] zstd [1.5.7] Lvl 3                | Comp:   291.91 MiB/s | Decomp:   854.08 MiB/s | Ratio:  3.20x
[Finished] zstd [1.5.7] Lvl 4                | Comp:   269.59 MiB/s | Decomp:   819.29 MiB/s | Ratio:  3.26x
[Finished] zstd [1.5.7] Lvl 5                | Comp:   145.16 MiB/s | Decomp:   778.73 MiB/s | Ratio:  3.37x
[Finished] zstd [1.5.7] Lvl 6                | Comp:   109.49 MiB/s | Decomp:   859.93 MiB/s | Ratio:  3.45x
[Finished] zstd [1.5.7] Lvl 7                | Comp:    99.32 MiB/s | Decomp:   847.01 MiB/s | Ratio:  3.51x
[Finished] zstd [1.5.7] Lvl 8                | Comp:    80.82 MiB/s | Decomp:   843.26 MiB/s | Ratio:  3.54x
[Finished] zstd [1.5.7] Lvl 9                | Comp:    78.99 MiB/s | Decomp:   835.90 MiB/s | Ratio:  3.58x
[Finished] zstd [1.5.7] Lvl 10               | Comp:    62.15 MiB/s | Decomp:   863.19 MiB/s | Ratio:  3.62x
[Finished] zstd [1.5.7] Lvl 11               | Comp:    43.56 MiB/s | Decomp:   852.81 MiB/s | Ratio:  3.65x
[Finished] zstd [1.5.7] Lvl 12               | Comp:    42.23 MiB/s | Decomp:   854.17 MiB/s | Ratio:  3.65x
--> Testing zstd [1.5.7] Lvl 13...
```

https://github.com/bstillwell/compression-testing/blob/main/benchmark-compression.py

CL'SO

Benchmarking scripts

```
[2026-02-20 16:58:36] ✓ [bryan@venetian:/home/bryan/src/compression-testing]$ ./graph-results.py
Finished processing: amd-ryzen-7-7800x3d-zstd
Finished processing: apple-m4-pro-zstd
Finished processing: odroid-hc4-all
Finished processing: odroid-hc4-zstd
Finished processing: raspberry-pi-5-zstd
Finished processing: apple-m4-pro-all
Finished processing: raspberry-pi-5-all
Finished processing: amd-ryzen-7-7800x3d-all

All graphs generated in the 'graphs/' directory.
```

https://github.com/bstillwell/compression-testing/blob/main/graph-results.py

CL'SO

Monitoring compression

RGW compression (zstd)

```
# View bucket stats
$ radosgw-admin bucket stats --bucket=benchmark | jq .usage | grep -v size_kb
{
  "rgw.main": {
    "size": 1000000000,
    "size_actual": 1000001536,
    "size_utilized": 357201295,
    "num_objects": 1
  },
  "rgw.multimeta": {
    "size": 0,
    "size_actual": 0,
    "size_utilized": 0,
    "num_objects": 0
  }
}
```

Monitoring compression

BlueStore compression (zstd)

```
# View compression by pool
$ ceph df detail

# View compression by osd
$ ceph tell osd.0 perf dump | grep -i compressed
        "compressed": 306389118,
        "compressed_allocated": 361349120,
        "compressed_original": 722698240,
```

Common pitfalls

- The "Incompressible Data" penalty
  - Already compressed data (gz/xz/zip/png/jpg/mp4/mkv)
  - Already encrypted data
- Not enough CPU cycles (your RGWs are already at 90% CPU util)
- Enabling compression doesn't re-compress data already in the pool
- Not useful for small objects

Future testing

- AMD's AOCL-Compression
  - Vendor optimized CPU library like Intel's ISA-L
- Hardware offloading
  - Intel QuickAssist
  - NVIDIA nvCOMP
  - UADK (often used with ARM64 - Hisilicon Kunpeng 920)
- Neural Network compression
  - NNCP by Fabrice Bellard (QEMU/FFMPEG)

CL'SO

Conclusions

- Validate your data is compressible before enabling compression
- Zstd is usually the best choice (fast & good compression)
- Compressing on the RGWs is better than on BlueStore
- Don't compress data that doesn't compress (it wastes CPU cycles)

CLYSO

Thank you!

Bryan Stillwell – bryan.stillwell@clyso.com