Blosc2: Debunking Compression Myths

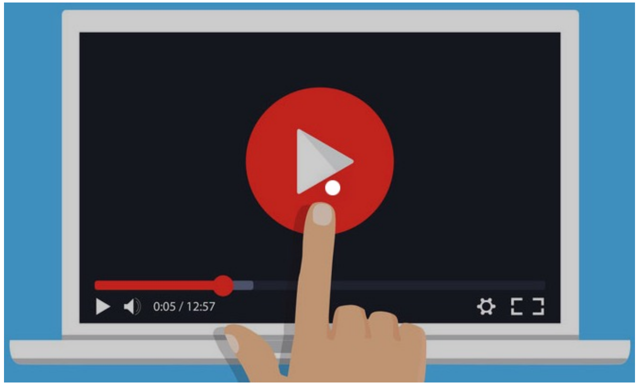
@Blosc2

https://blosc.org/

Francesc Alted - @FrancescAlted
The Blosc Development Team
CEO ironArray.io

Python Castelló, November 4th 2022

Compression Is Everywhere



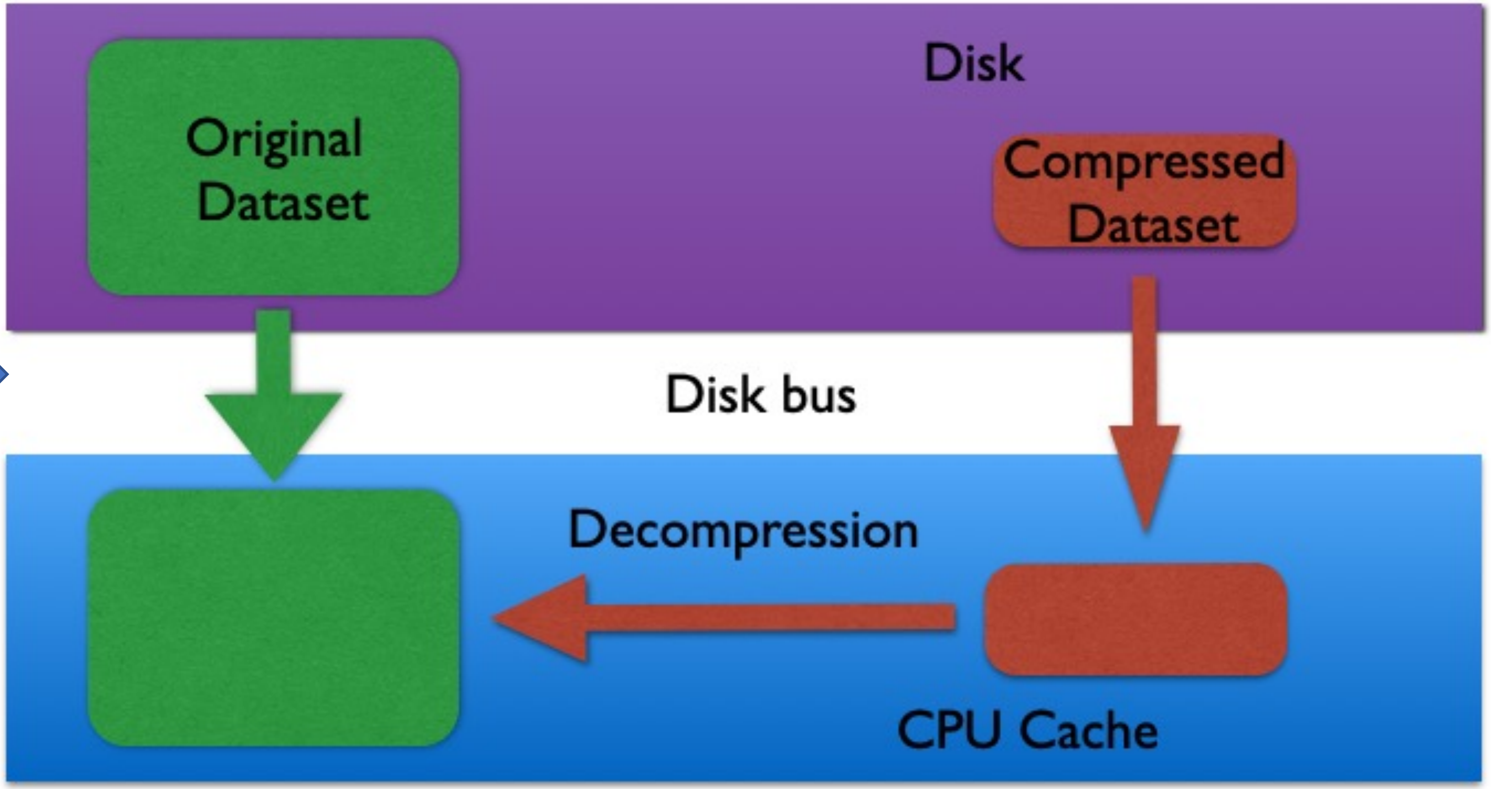
Blosc is Compression for Binary Data (and a little bit more)

Goals

- Compress fast
- Compress efficiently
- Accelerate computation with numbers (binary data)
- Persistence of (compressed) data (**Blosc2**)

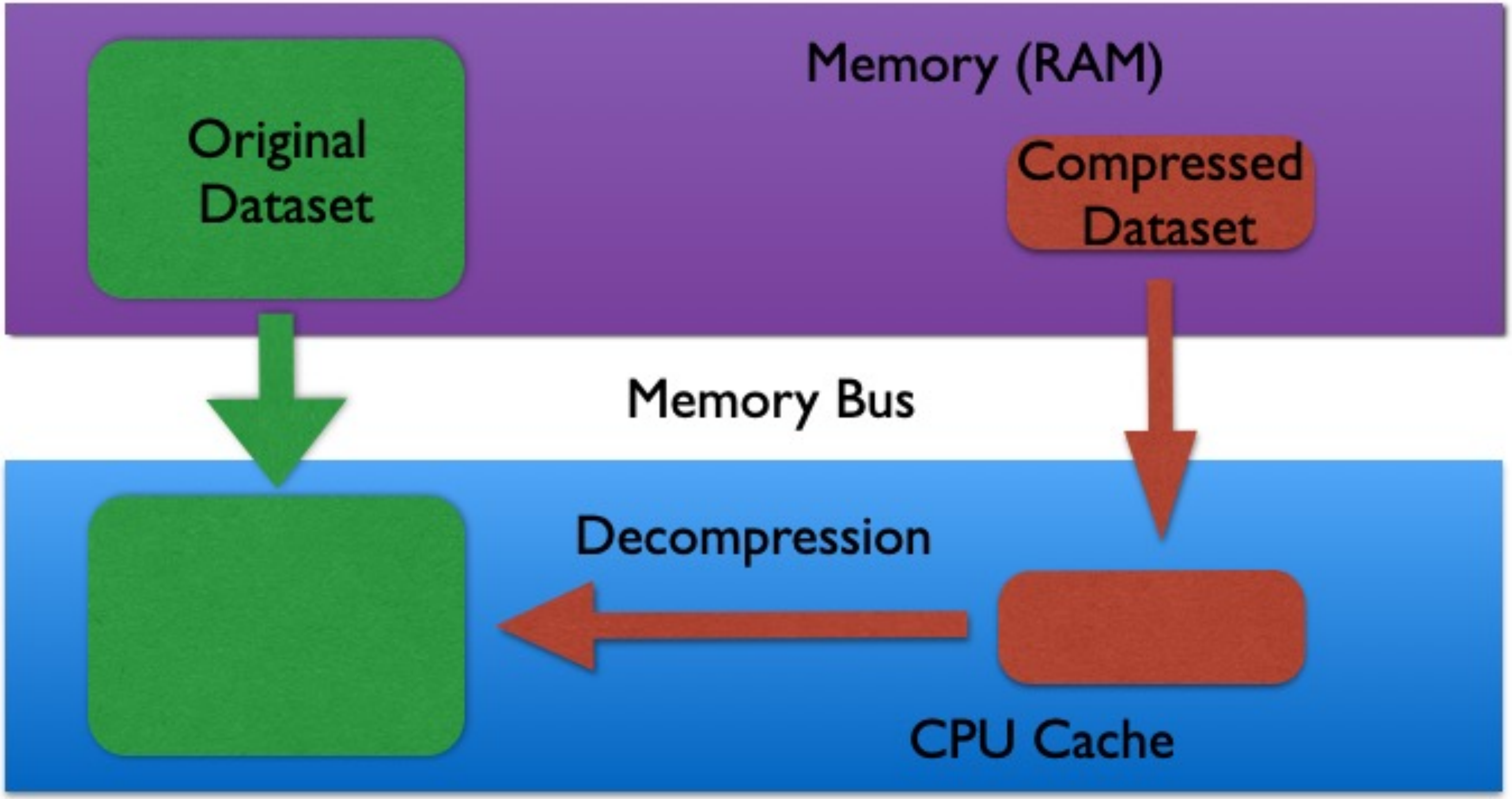
Leveraging Compression Straight to CPU (I)

Where game industry is now →

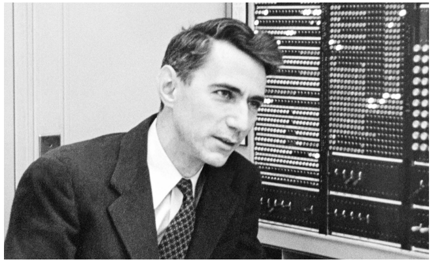


Leveraging Compression Straight to CPU (II)

Where Blosc is headed ➡



Breaking entropy (I)



Back in the 1940’s, Claude Shannon invented a way to measure the information content of a message and called it **information entropy**:

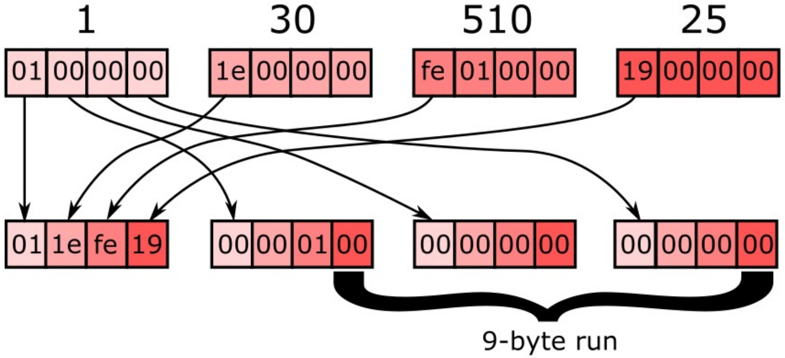
$$H(s) = -\sum_{i=1}^{n} p_i log_2(p_i)$$

In theory, you *cannot compress a dataset beyond that entropy*.

However, Shannon did not take into account that **symbol ordering** (and not only *probability of occurrence*) is important when finding ways to express messages in less space than such information entropy.

Breaking entropy (II)

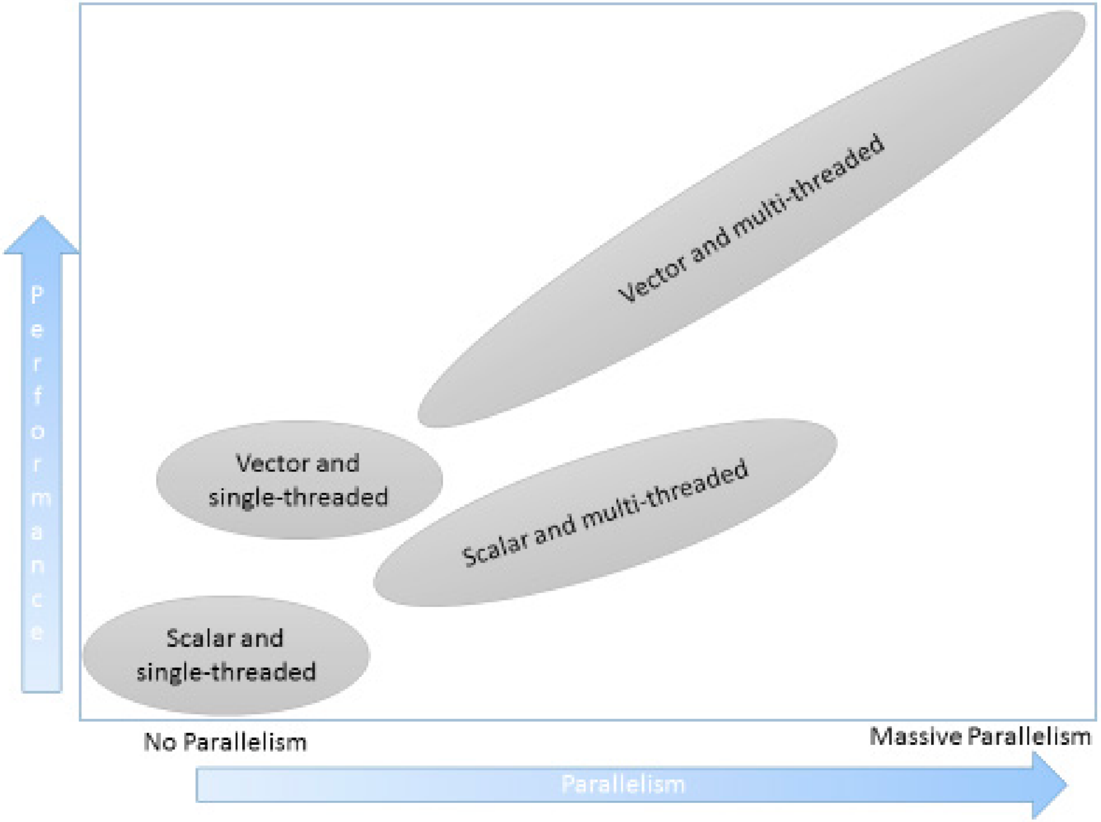
Blosc comes with so-called filters that are about re-ordering data before the encoding stage. One example is the **shuffle filter**:



This typically allows codecs to **go beyond information entropy limits**.

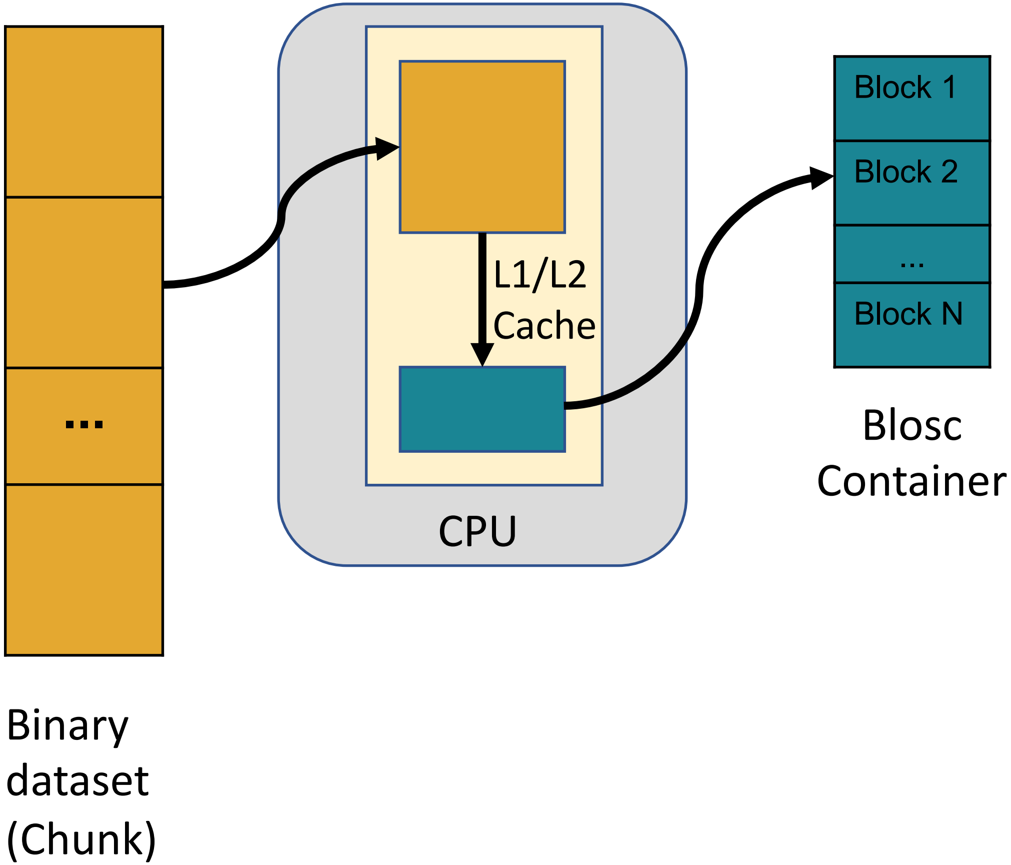
[BTW, Blosc2 has optimized versions of the shuffle filter for Intel (SSE2, AVX2), ARM (NEON) and PowerPC (ALTIVEC)]

Performance of Vector and Parallel Hardware


Vector and multi-threaded
Vector and single-threaded
Scalar and multi-threaded
Scalar and single-threaded
Performance
No Parallelism
Massive Parallelism
Parallelism

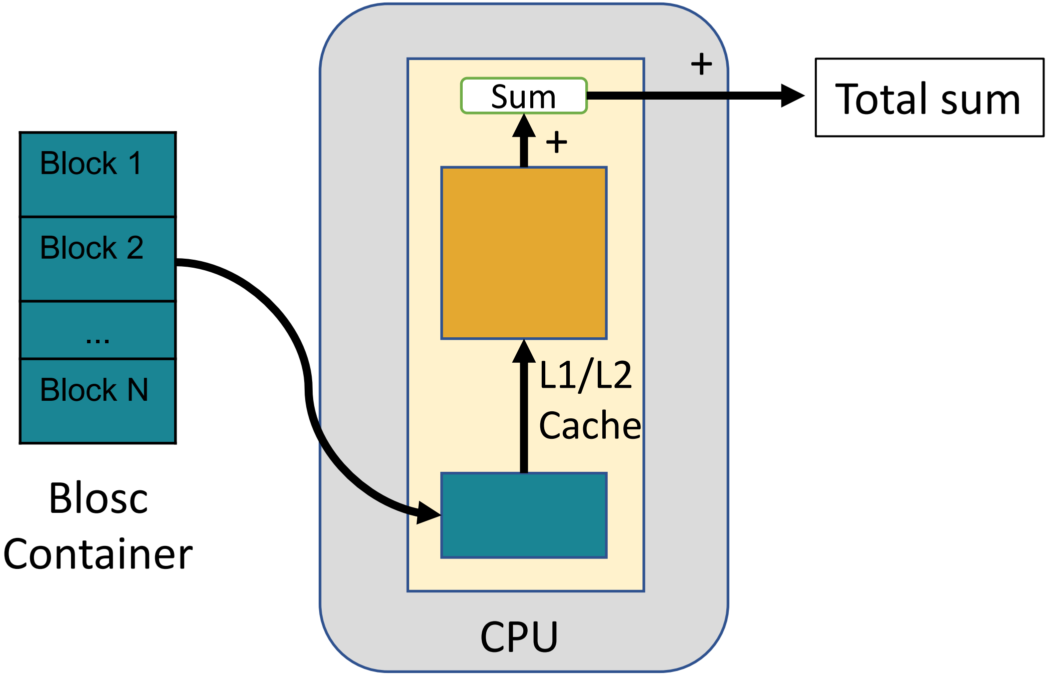
What is Blosc?

- ✓ Sending data from CPU to memory (and back) faster than *memcpy().*
- ✓ Split in blocks for better cache use: divide and conquer.
- ✓ It can use different filters (e.g. shuffle, bitsuffle) and codecs (e.g. LZ4, Zlib, Zstd, BloscLZ).

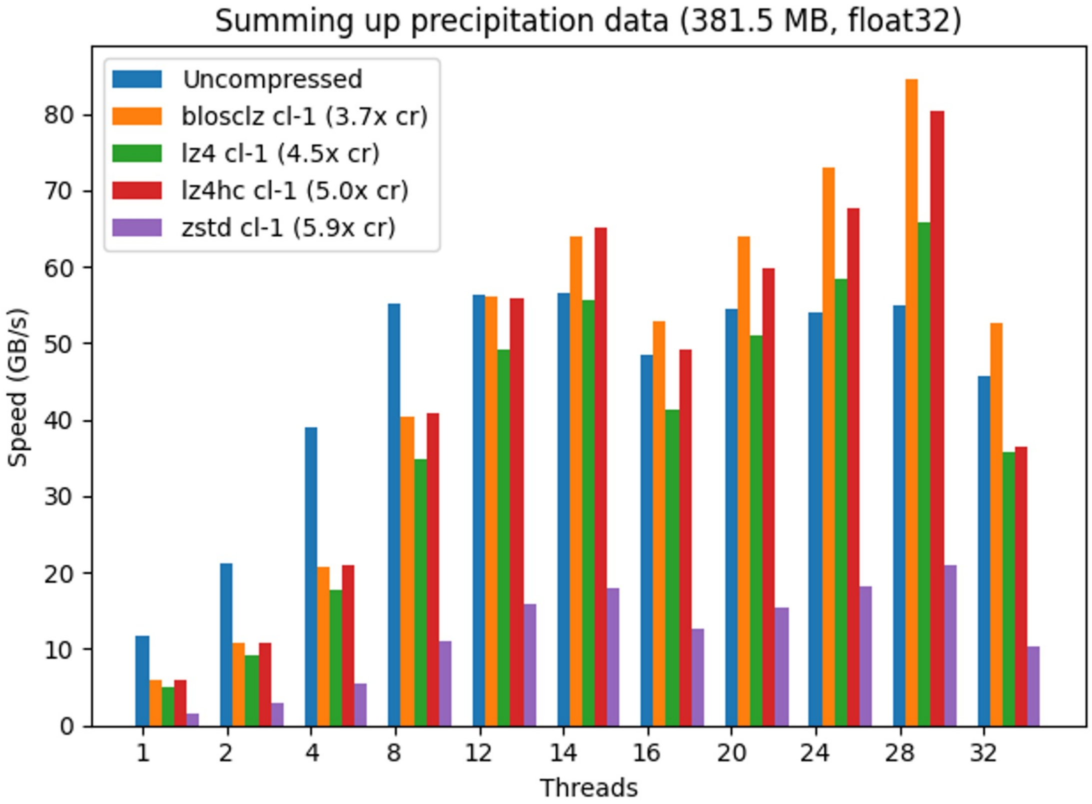


Leveraging Blosc the ‘right way’

- ✓ Blocks should be decompressed and operated in private caches for best performance.
- ✓ The need for data to fit in private caches is to avoid contention in Blosc multithreading.
- ✓ If possible, use all the data before it leaves caches.

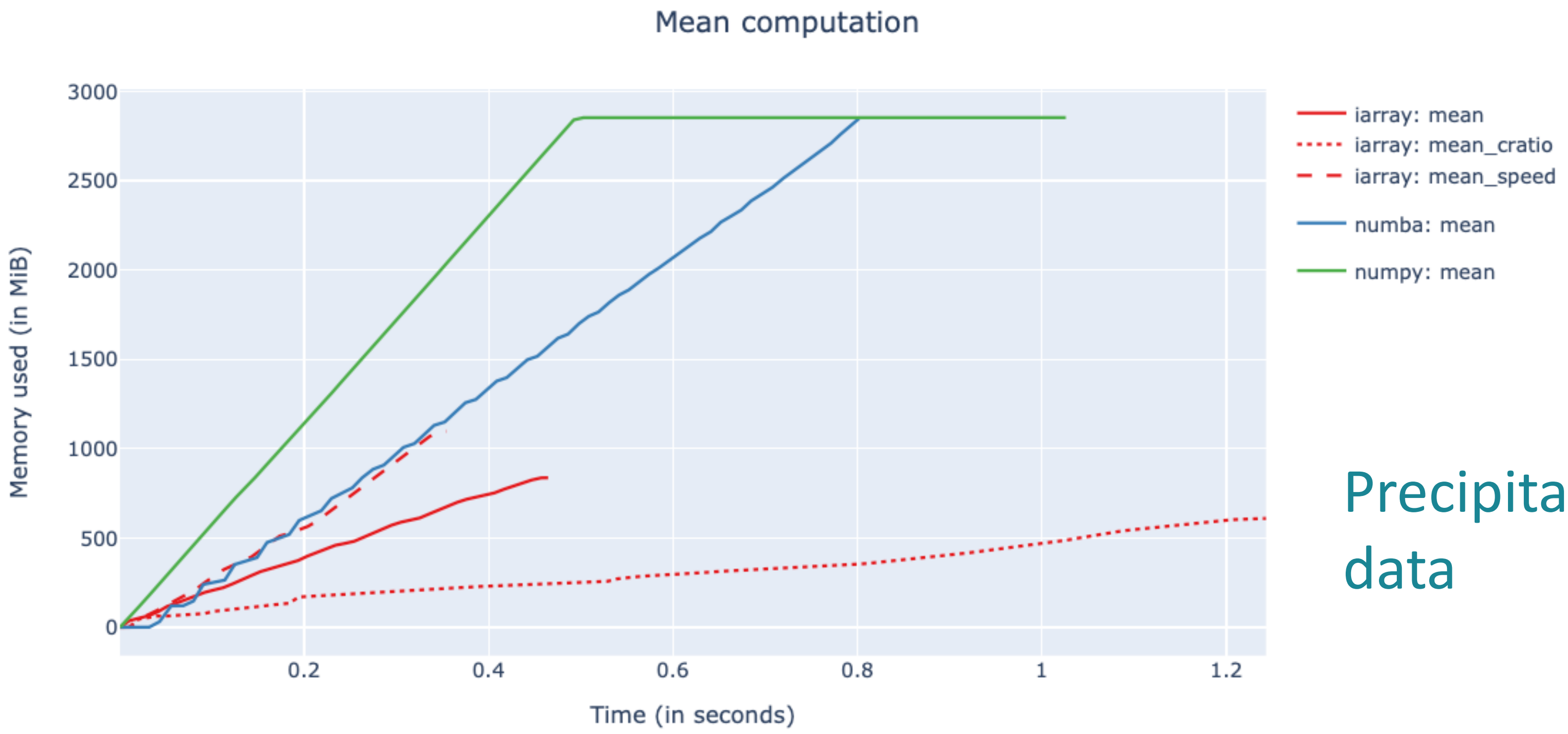


Breaking memory walls (I)
Computing a reduction



https://www.blosc.org/posts/breaking-memory-walls/

Breaking memory walls (II)
Computing a mean `(a + b + c) / 3`



Precipitation data

ironArray (leveraging Blosc2) can compute faster than NumPy, and also (parallel) Numba.

Where is Blosc used?

Blosc is used in many places in the PyData ecosystem:

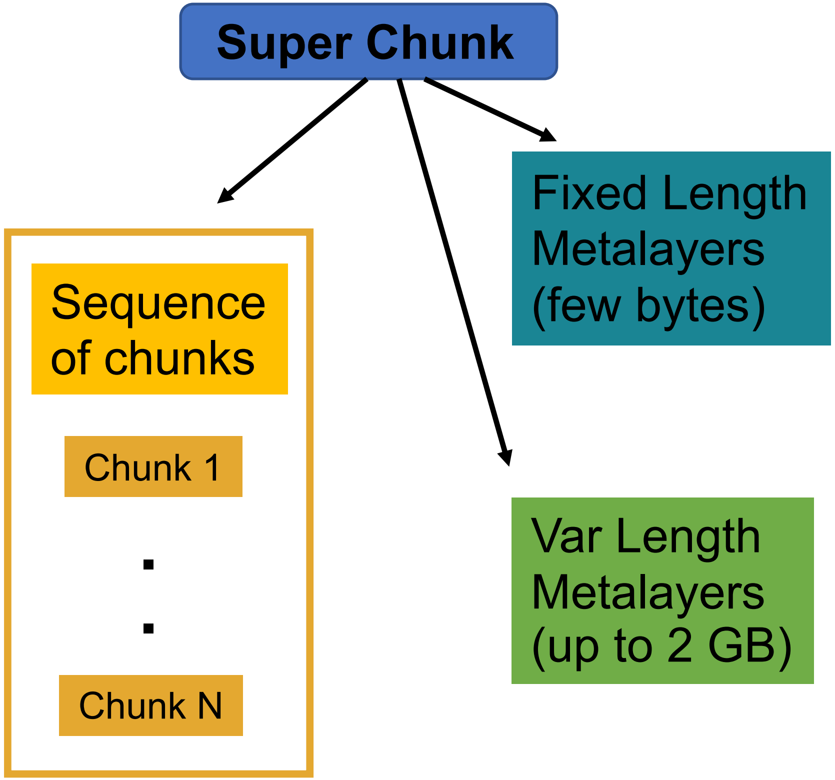
- HDF5 / h5py (via hdf5plugin)

- HDF5 / PyTables (native)
- Zarr (via numcodecs)
- ironArray (Blosc2)

Lots of terrabytes compressed (and decompressed) on a daily basis!

What is Blosc2?

- ✓ Blosc2 is the next generation of Blosc(1).
- ✓ Blosc2 adds 63-bit containers (super-chunks) that expand over the existing 31-bit containers (chunks) in Blosc1.
- ✓ Metalayers for adding info for apps and users.


Super Chunk
Sequence of chunks
Chunk 1
.
.
Chunk N
Fixed Length Metalayers (few bytes)
Var Length Metalayers (up to 2 GB)

Blosc2: New features

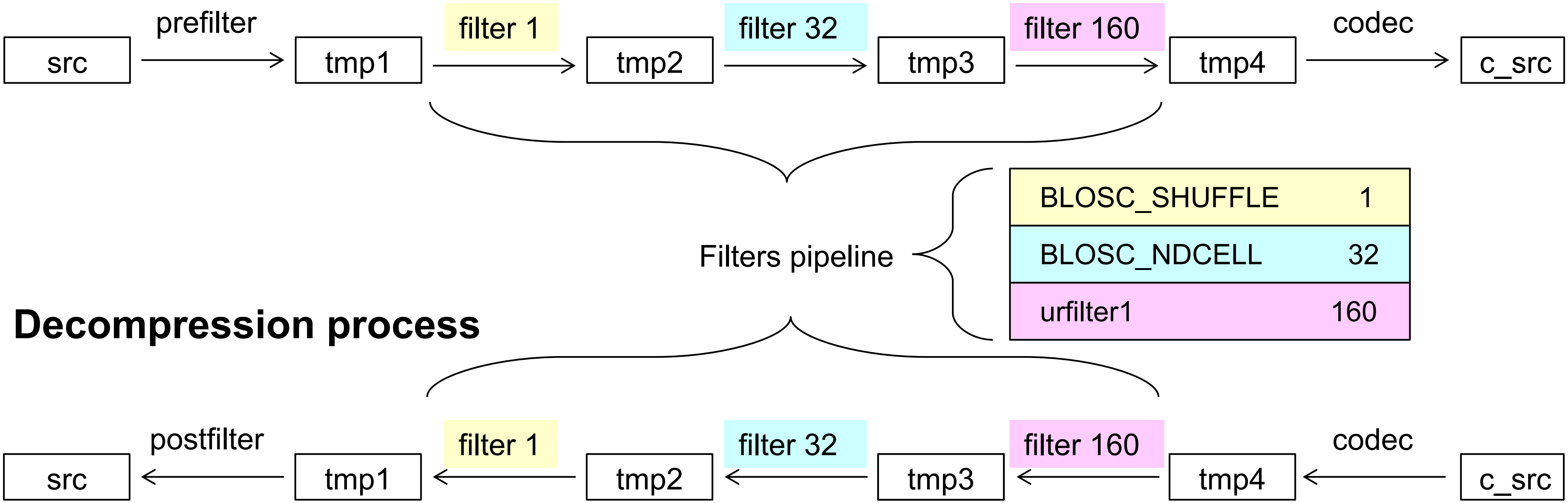
Filter Pipeline	Serialization Format
Parallel I/O	Pluggable Codecs & Filters

Blosc2: New features

Filter Pipeline	Serialization Format
Parallel I/O	Pluggable Codecs & Filters

Filter pipeline: composing filters + codecs

Compression process



src → prefilter → tmp1 → filter 1 → tmp2 → filter 32 → tmp3 → filter 160 → tmp4 → codec → c_src

Filters pipeline

BLOSC_SHUFFLE	1
BLOSC_NDCELL	32
urfilter1	160

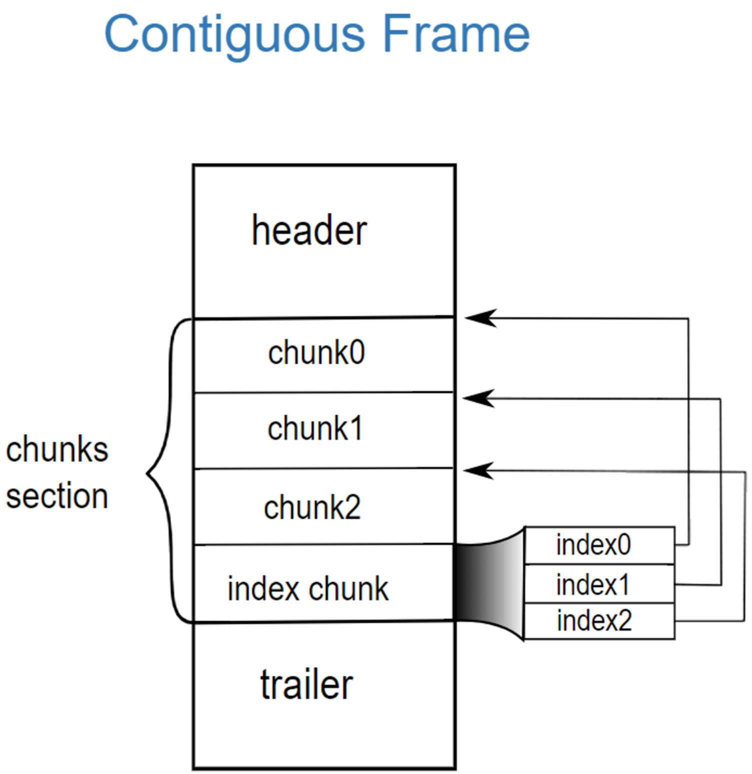
Decompression process

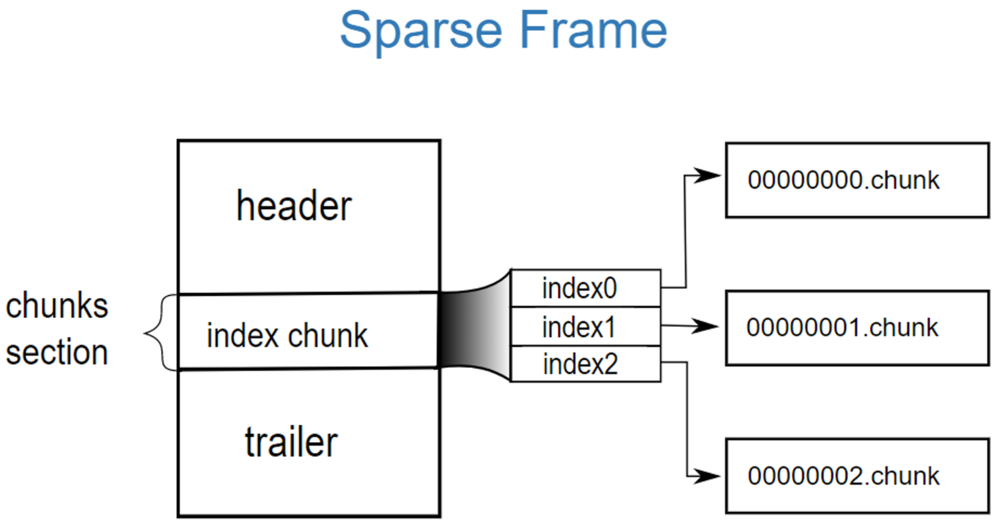
src ← postfilter ← tmp1 ← filter 1 ← tmp2 ← filter 32 ← tmp3 ← filter 160 ← tmp4 ← codec ← c_src

Blosc2: New features

Filter Pipeline	**Serialization Format**
Parallel I/O	Pluggable Codecs & Filters

Frames: Serializing super-chunks





Frames can live either **on disk** or **in memory**

Frame specification is very simple

- Fully documented in **less than 700 lines of text**:

```
> wc -l README_*_FORMAT.rst
     278 README_CFRAME_FORMAT.rst
     283 README_CHUNK_FORMAT.rst
      76 README_SFRAME_FORMAT.rst
     637 total
```

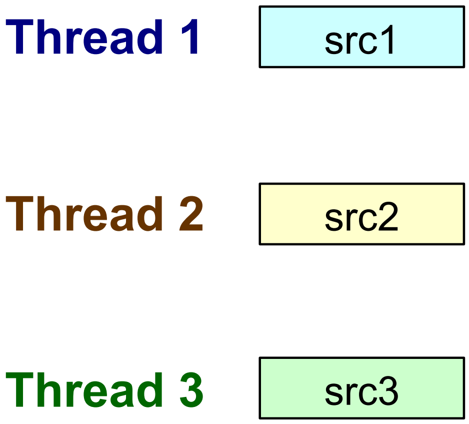
- One of the reasons is that it rests on the shoulders of MessagePack (https://msgpack.org), an efficient binary serialization format.
- Simplicity is important in terms of portability, and specially, safety.

Blosc2: New features

Filter Pipeline	Serialization Format
Parallel I/O	Pluggable Codecs & Filters

Filters and codecs work in parallel

Compression process



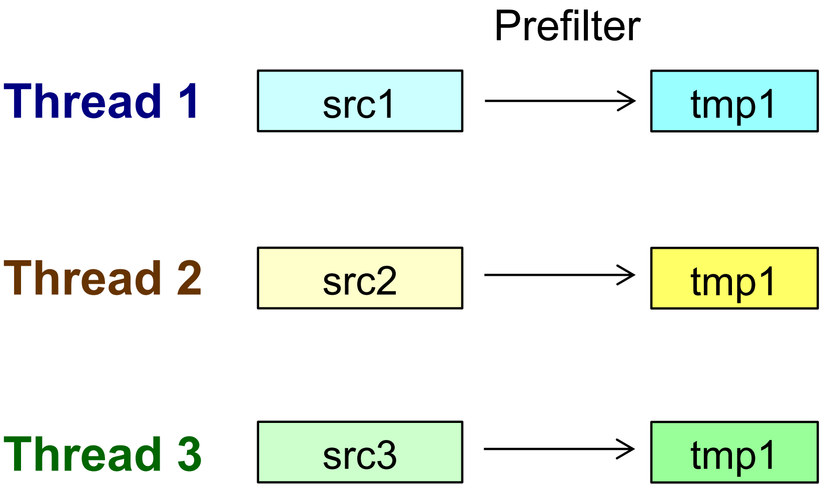
Thread 1 src1

Thread 2 src2

Thread 3 src3

Filters and codecs work in parallel

Compression process



Prefilter

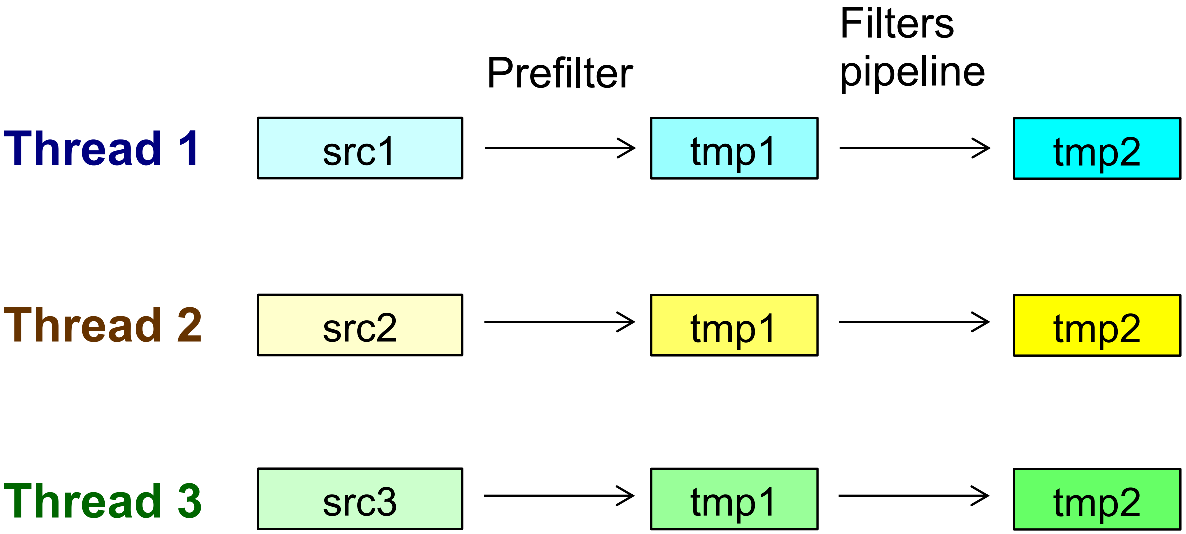
Thread 1 src1 → tmp1

Thread 2 src2 → tmp1

Thread 3 src3 → tmp1

Filters and codecs work in parallel

Compression process



Prefilter

Filters pipeline

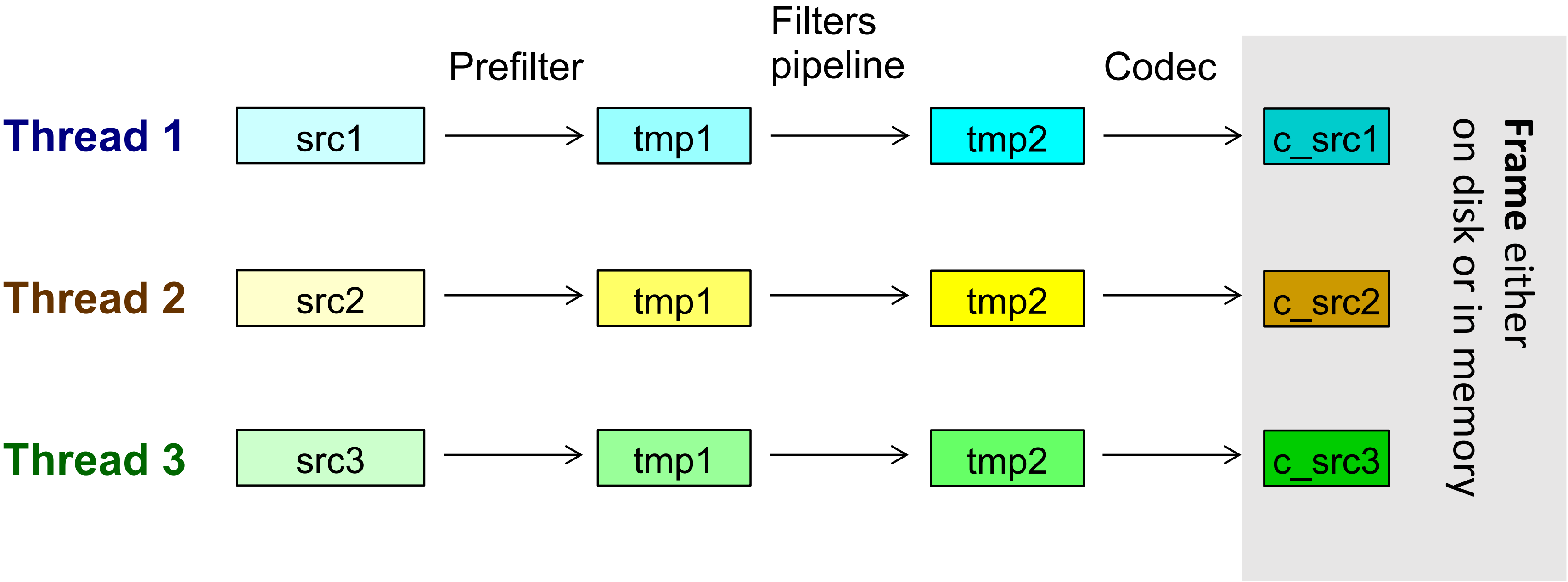
Thread 1 src1 → tmp1 → tmp2

Thread 2 src2 → tmp1 → tmp2

Thread 3 src3 → tmp1 → tmp2

Filters and codecs work in parallel

Compression process



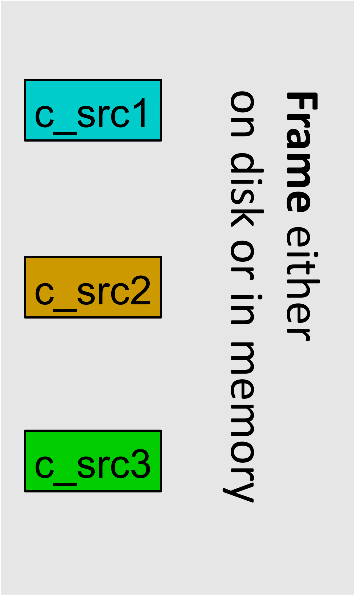
Filters and codecs work in parallel

Decompression process

Thread 1

Thread 2

Thread 3



c_src1

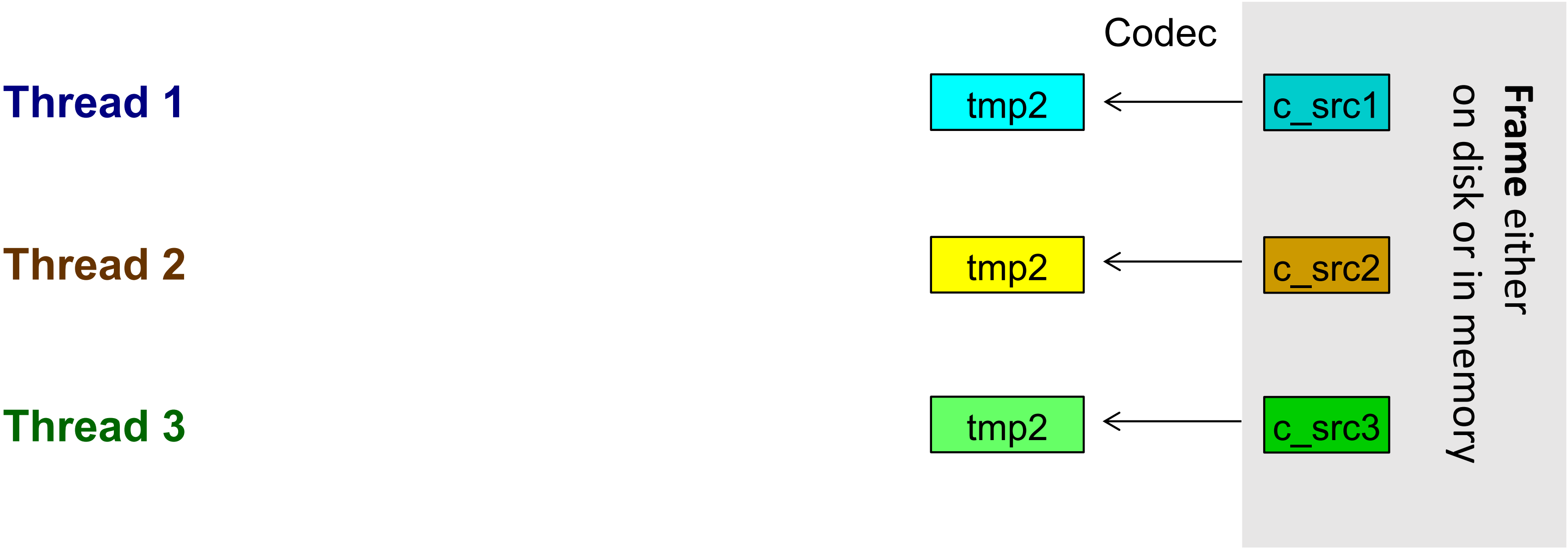
c_src2

c_src3

Frame either
on disk or in memory

Filters and codecs work in parallel

Decompression process



Codec

Thread 1

tmp2 ← c_src1

Thread 2

tmp2 ← c_src2

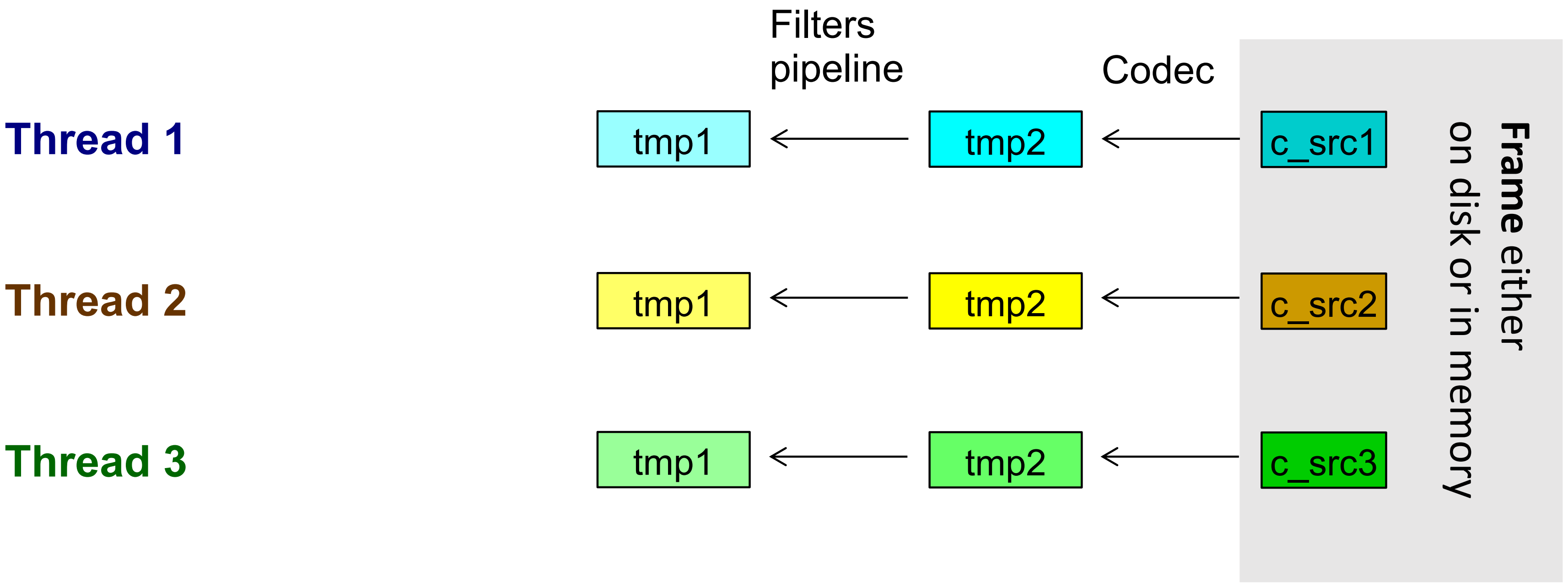
Thread 3

tmp2 ← c_src3

Frame either on disk or in memory

Filters and codecs work in parallel

Decompression process



Filters pipeline

Codec

Thread 1

tmp1 ← tmp2 ← c_src1

Thread 2

tmp1 ← tmp2 ← c_src2

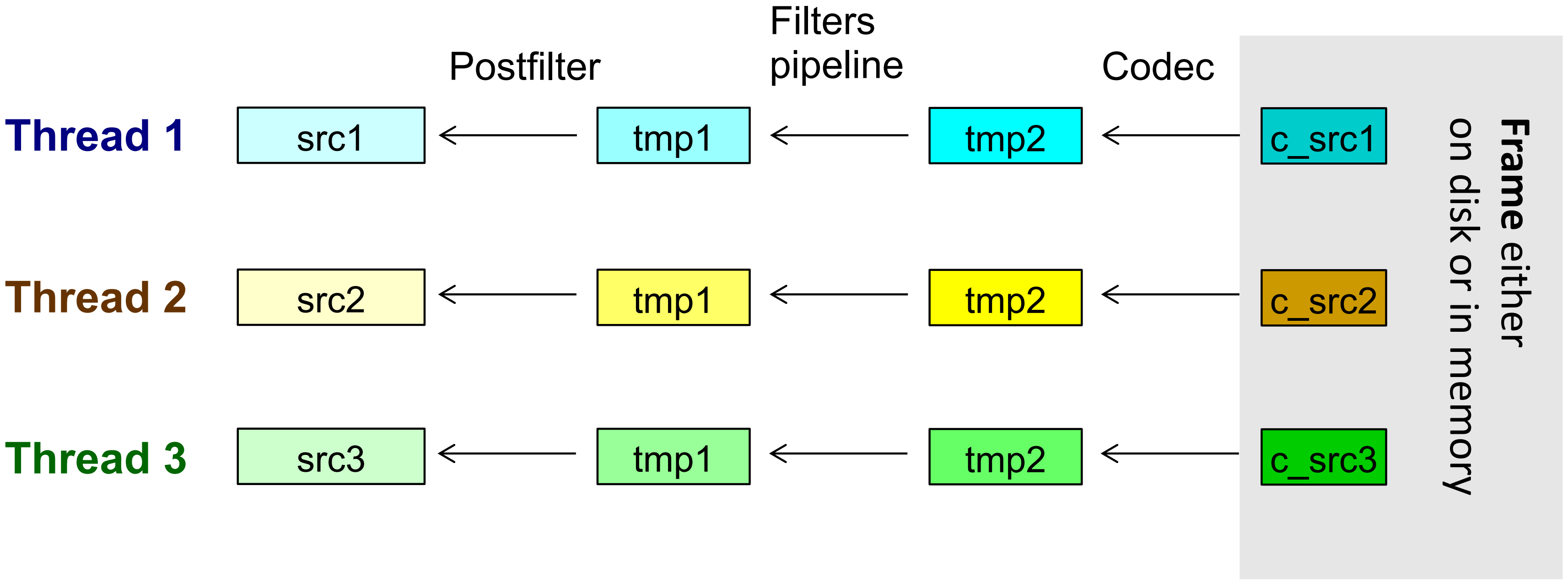
Thread 3

tmp1 ← tmp2 ← c_src3

Frame either
on disk or in memory

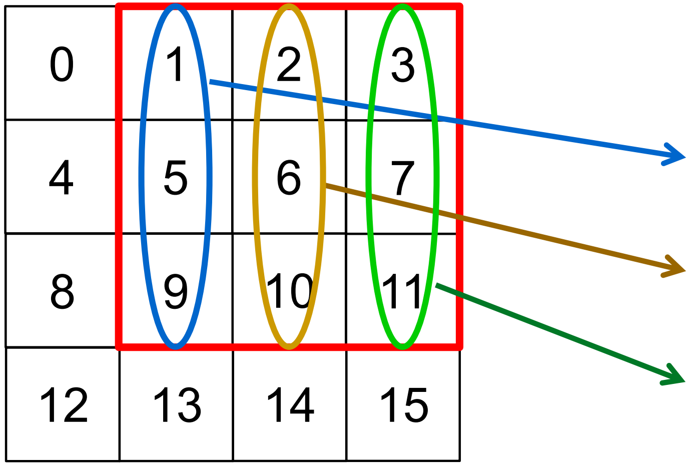
Filters and codecs work in parallel

Decompression process



Block masks and parallel I/O

Block maskout	F	T	T	T	F	T	T	T	F	T	T	T	F	F	F	F
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

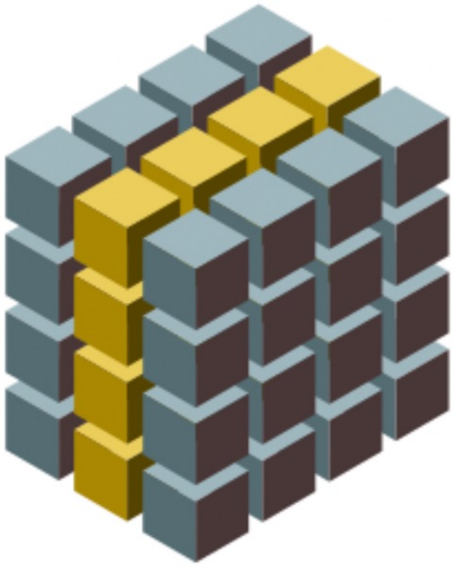
Thread 1: 1, 5, 9

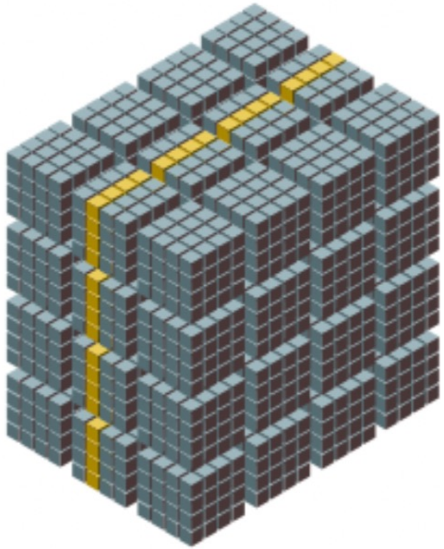
Thread 2: 2, 6, 10

Thread 3: 3, 7, 11

Specially effective when retrieving slices of multidim datasets.

Masked & paralel I/O in multidim datasets

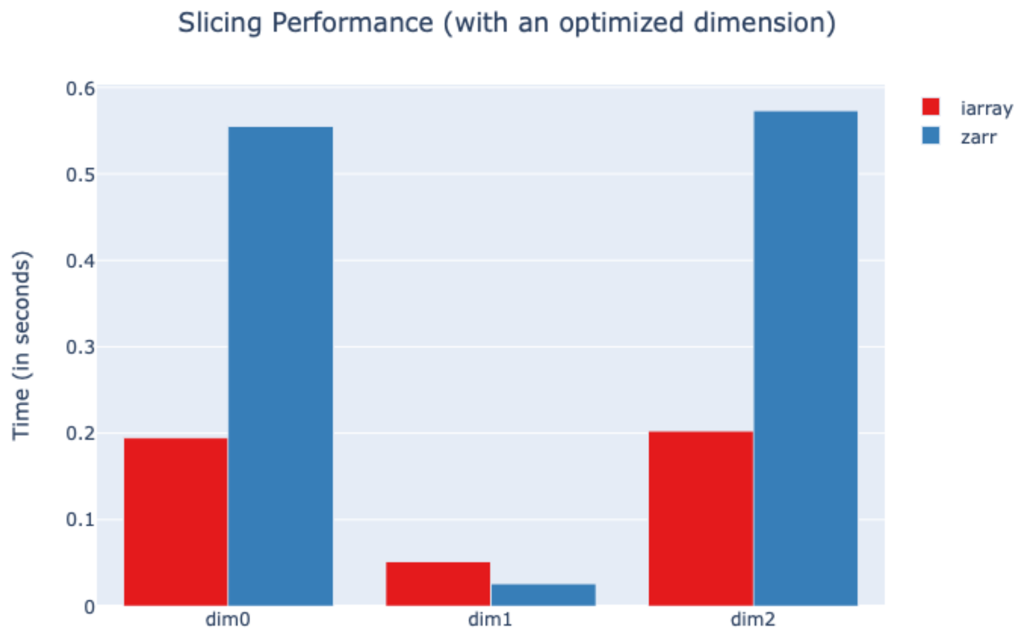




Much more selective and faster queries!

Caterva (https://github.com/Blosc/caterva) and ironArray (https://ironarray.io)

Masked & paralel I/O in multidim datasets



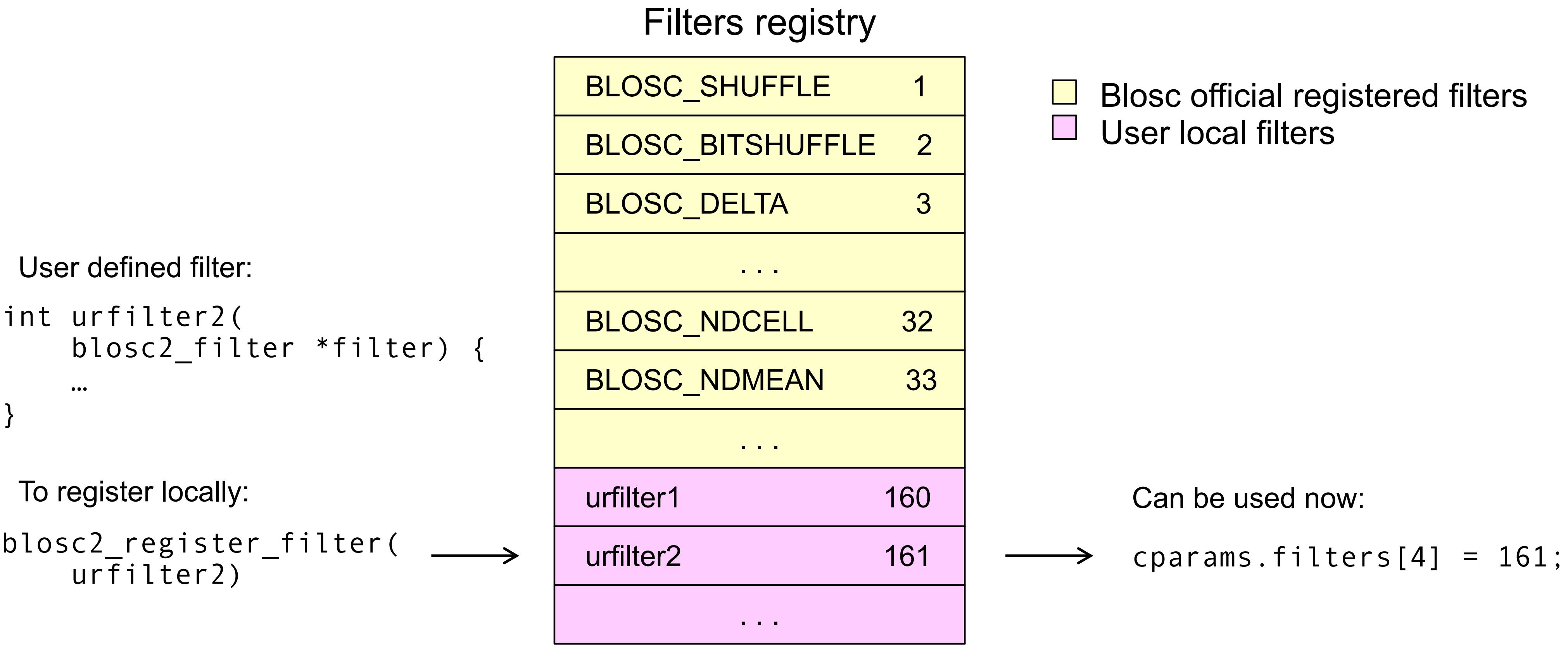
Better performance in general
(except for dimension where retrieving a chunk is already optimal)

https://ironarray.io/docs/html/tutorials/03.Slicing_Datasets_and_Creating_Views.html

Blosc2: New features

Filter Pipeline	Serialization Format
Parallel I/O	**Pluggable Codecs & Filters**

Adaptability: plugins in local registry



Filters registry

BLOSC_SHUFFLE	1
BLOSC_BITSHUFFLE	2
BLOSC_DELTA	3
. . .	
BLOSC_NDCELL	32
BLOSC_NDMEAN	33
. . .	
urfilter1	160
urfilter2	161
. . .	

🟨 Blosc official registered filters
🟪 User local filters

User defined filter:

```
int urfilter2(
    blosc2_filter *filter) {
    …
}
```

To register locally:

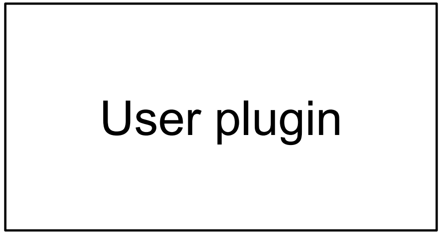
```
blosc2_register_filter(
    urfilter2)
```

Can be used now:

```
cparams.filters[4] = 161;
```

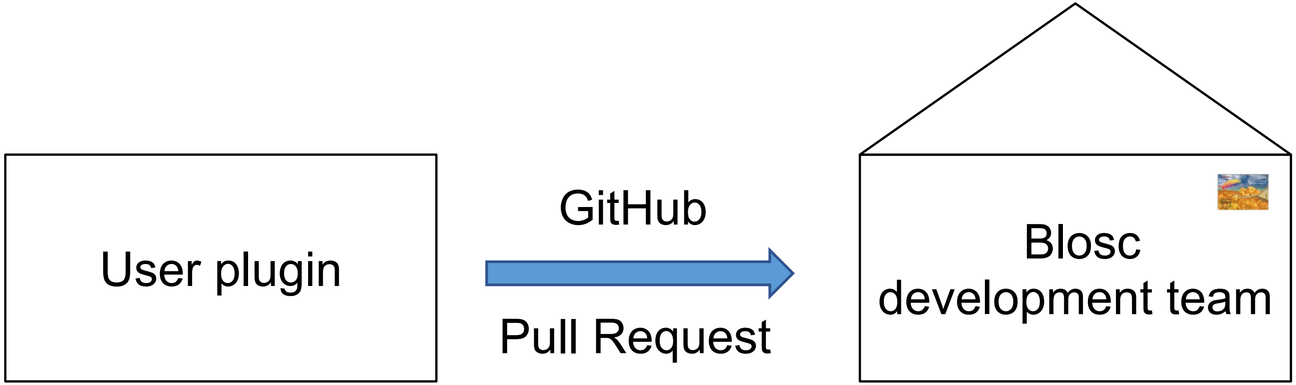
And a similar procedure goes for codecs too!

Registering plugins in central registry



User plugin

Registering plugins in central registry



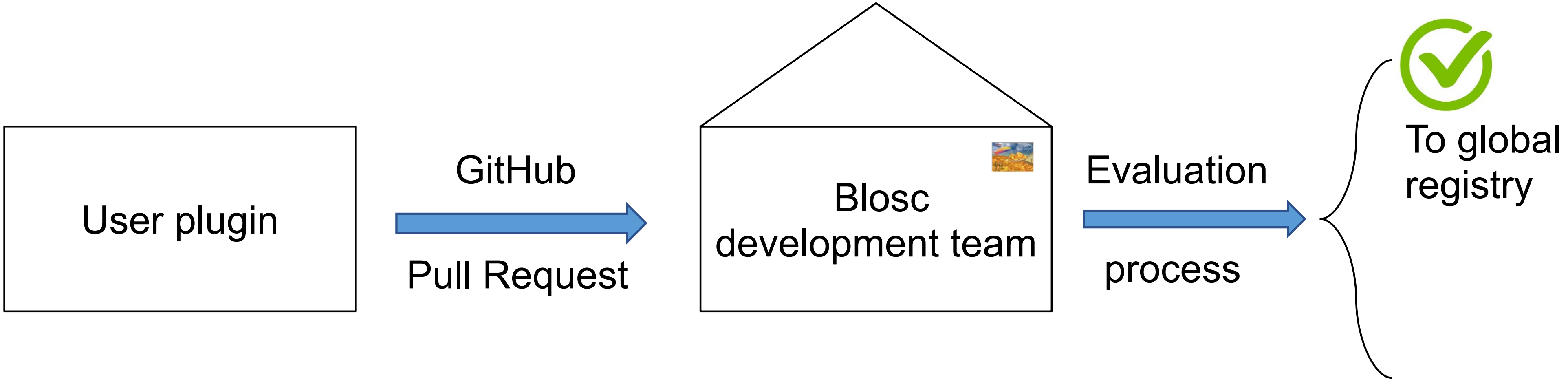
User plugin

GitHub

Pull Request

Blosc development team

Registering plugins in central registry



User plugin

GitHub

Pull Request

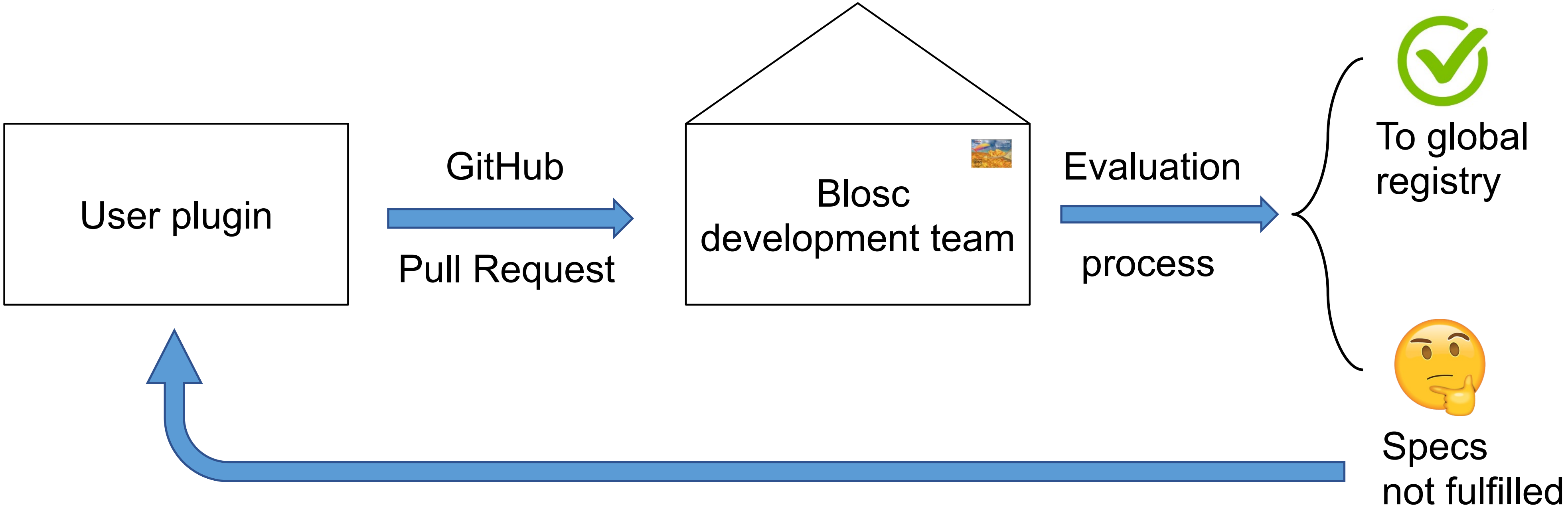
Blosc development team

Evaluation

process

To global registry

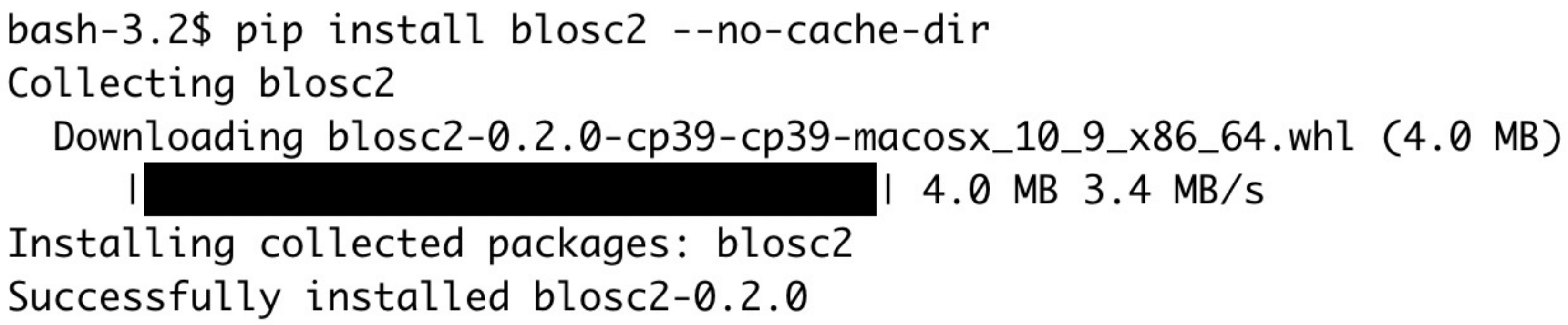
Registering plugins in central registry



Specs: https://github.com/Blosc/c-blosc2/tree/main/plugins

Plugins in central registry are easy to deploy

Central registered plugins are **included** and **distributed** within the Blosc2 library, which can be installed using the Python wheels:



```
bash-3.2$ pip install blosc2 --no-cache-dir
Collecting blosc2
  Downloading blosc2-0.2.0-cp39-cp39-macosx_10_9_x86_64.whl (4.0 MB)
     |████████████████████████████████| 4.0 MB 3.4 MB/s
Installing collected packages: blosc2
Successfully installed blosc2-0.2.0
```

Very convenient in making your filter/codec accessible for everybody

Other features for Blosc2

- **Safety/Security**: we are actively using the OSS-Fuzz service for uncovering programming errors in C-Blosc2.
- **Nice markup for documentation**: See https://c-blosc2.readthedocs.io
- **Efficient support for special values**: repeated values can be represented with an efficient, simple and fast run-length encoding. This is really useful for storing sparse data.
- **Python wrapper for Blosc2 (new 0.2.0 released):** https://python-blosc2.readthedocs.io

Python-Blosc2: A Python Wrapper

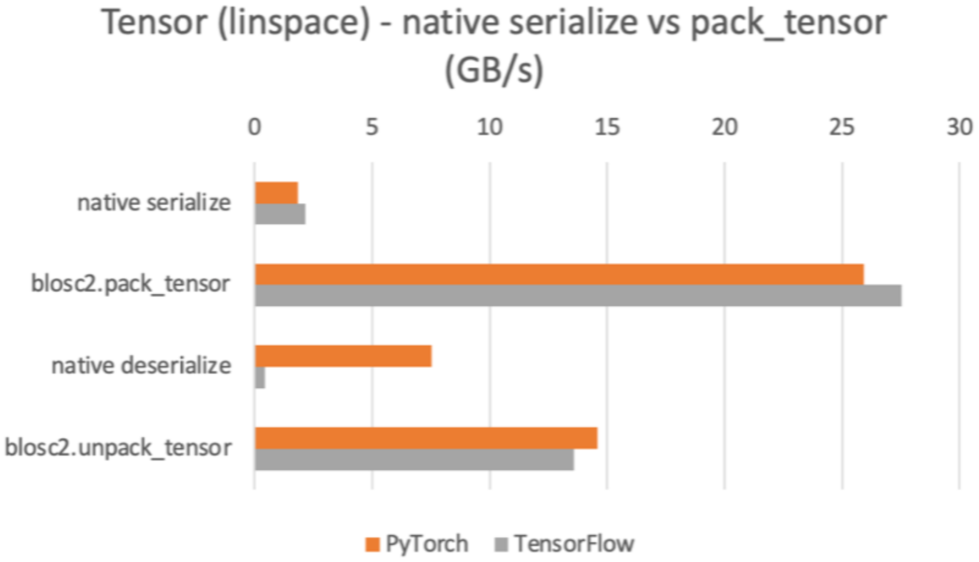
Python-Blosc2 is the official wrapper for the C-Blosc2 library:

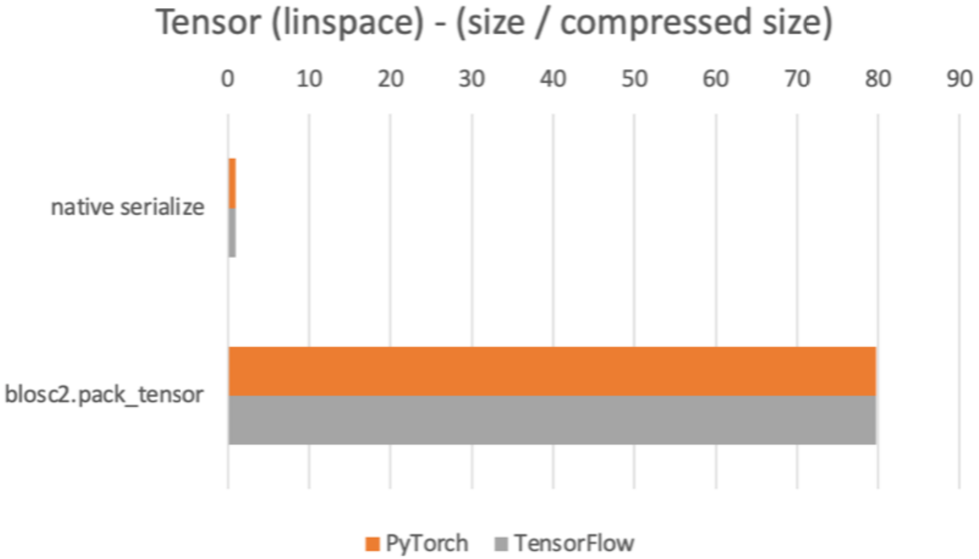
```
import numpy as np
import blosc2

a = np.arange(1_000_000)

file_size = blosc2.save_tensor(a, "save_tensor.bl2", mode="w")
print("Length of saved tensor in file (bytes):", file_size)

a2 = blosc2.load_tensor("save_tensor.bl2")
assert np.alltrue(a == a2)
```

Supports NumPy, PyTorch and TensorFlow
(PyDTNN would be nice; volunteers? ;-)

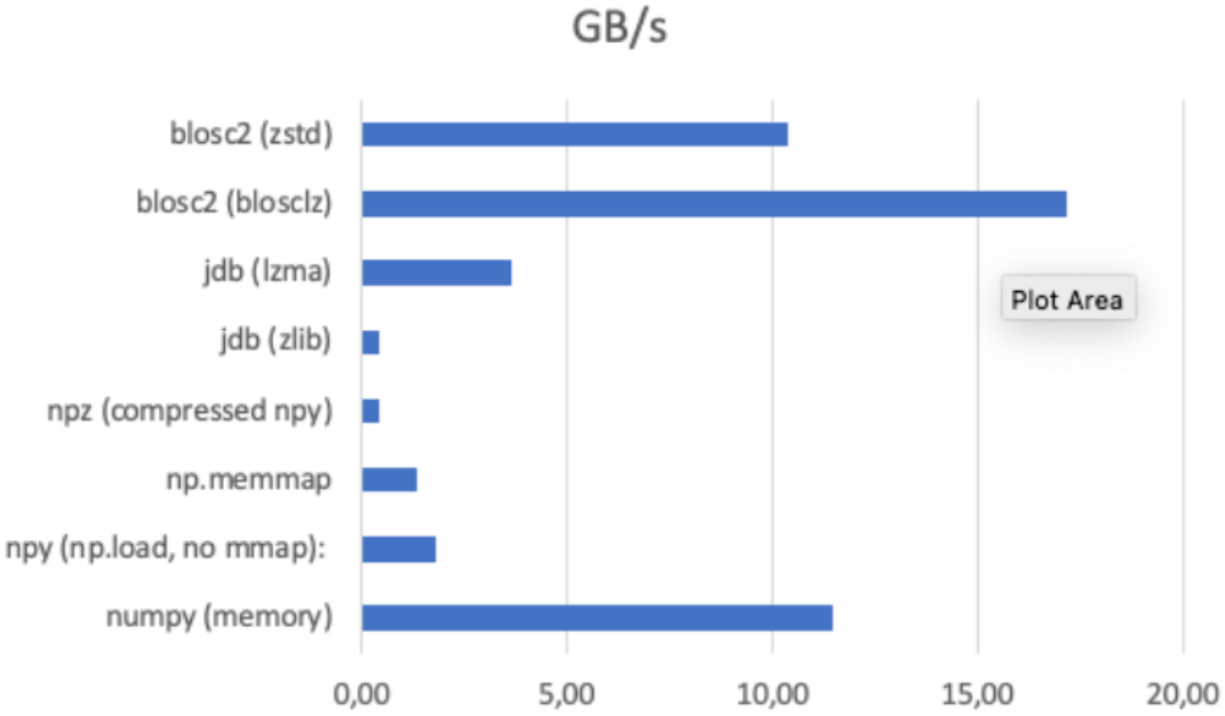
Speed vs Native Serialization

- Blosc2 is typically faster (and sometimes much faster) than PyTorch / TensorFlow.
- Compression allows for using less storage too





Comparison with other libs



Blosc2 can be faster than memcpy

[No news, but still good to know!]

https://mail.python.org/archives/list/numpy-discussion@python.org/message/GDPQSAMXCWQDXVTQVJRDXKAQYGHM6JQ4/

Conclusion

Adapting compression to your needs

- ➔Tackling compression includes a gazillion ways to do it, but basically:
  - ➔Get the maximum compression ratio
  - ➔Reduce the compression/decompression time to a maximum
- ➔Blosc2 comes with a **rich set of codecs and filters** that users can easily try to find the one that better fits to their needs
- ➔Blosc2 orchestrates these codecs and filters for:
  - ➔**Parallelization** via multithreading
  - ➔Reuse and sharing internal buffers for **optimal memory consumption**

The result is a highly efficient tool for **compressing your way**

Data is the most important part of your system

The Blosc development team is committed to the future of your data:

- Blosc2 has a very simple format, and hence is very portable and maintenable
- We have spent quite a lot of energy keeping it orderly and clean
- Last but not least, safety/security is paramount for us

Proactivity should be the primary mechanism of **data integrity**

Myths To Be Debunked

- Compression must introduce a lot of overhead
  - **FALSE**: if correctly done, compression can actually accelerate most of the processes, including heavy in-memory computations!
- Compression is for experts
  - **FALSE**: Blosc allows for easily trying different combinations of codecs for you to experiment! If not satisfied, come with your own filter (easy) or codec (a bit more complex, yes ☺)

The Blosc Development Team

Aleix Alcacer

Oscar Guiñón

Marta Iborra

Alberto Sabater

Nathan Moinvaziri

Francesc Alted

Thanks to donors!

HUAWEI

[[ ]] ironArray

NUMFOCUS
OPEN CODE = BETTER SCIENCE

python SOFTWARE FOUNDATION

ESRF
The European Synchrotron

Google

Jeff Hammerbacher

Without them, we could not have possibly put Blosc2 into production status: Blosc2 2.0.0 came out in June 2021; now at 2.4.3.

Enjoy data!

https://blosc.org/