Scalable array processing with bounded memory in Python

Worthy
GORDON AND BETTY MOORE FOUNDATION
Lamont-Doherty Earth Observatory
COLUMBIA UNIVERSITY | EARTH INSTITUTE

*Tom Nicholas
Tom White

earthmover

*tom@earthmover.io
@TomNicholas
@tegnicholas.bsky.social

What I will talk about:

- Vision for Science at Scale
- What is Cubed?
- Xarray integration
- Initial results
- Pros and Cons
- Next steps

Tale of two Toms:

Tom Nicholas

- Xarray core dev

xarray Zarr

- Background in Plasma Physics + Oceanography

PANGEO

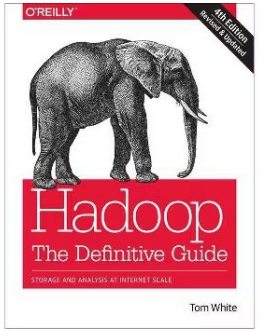
- Now engineer at Earthmover

earthmover

Tom White

- Sgkit developer

sgkit



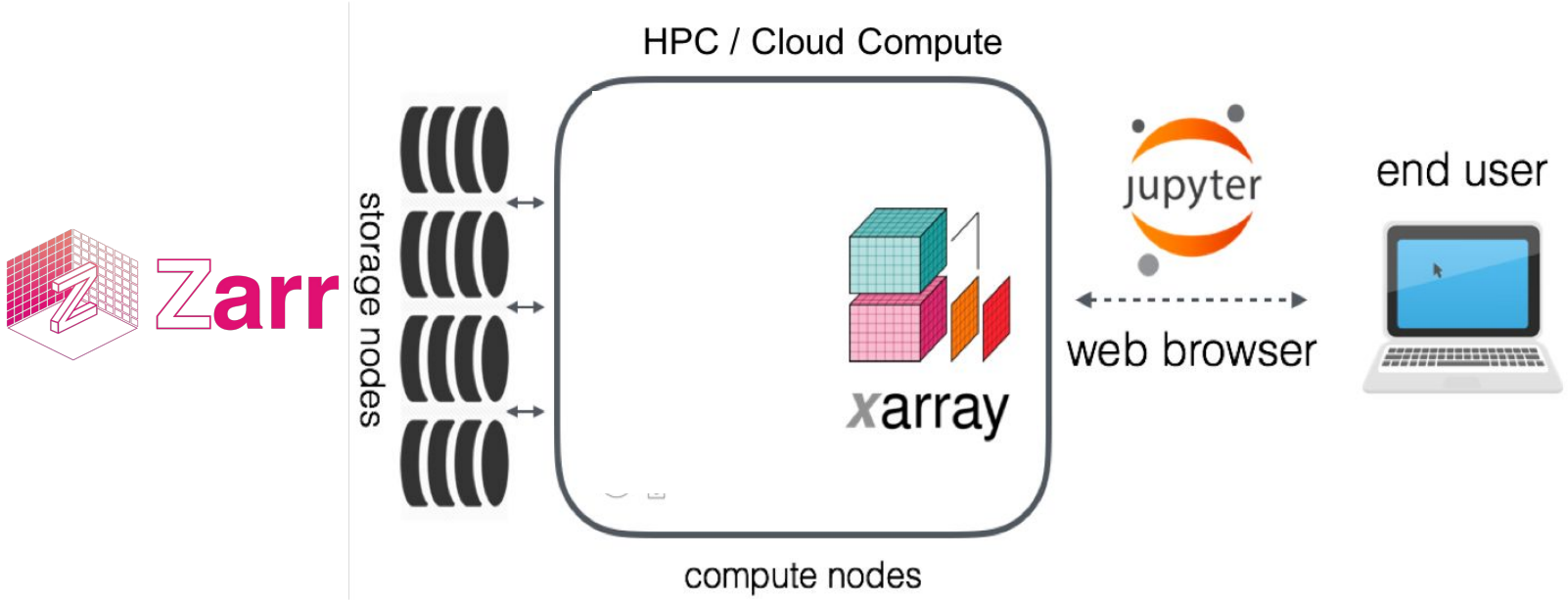
- Hadoop maintainer

- Cubed main developer

Cubed

Vision for Science at Scale (Tom's 🎄 list 🎁)

- My perfect parallel executor…



Vision for Science at Scale (Tom's  list ) - (1)

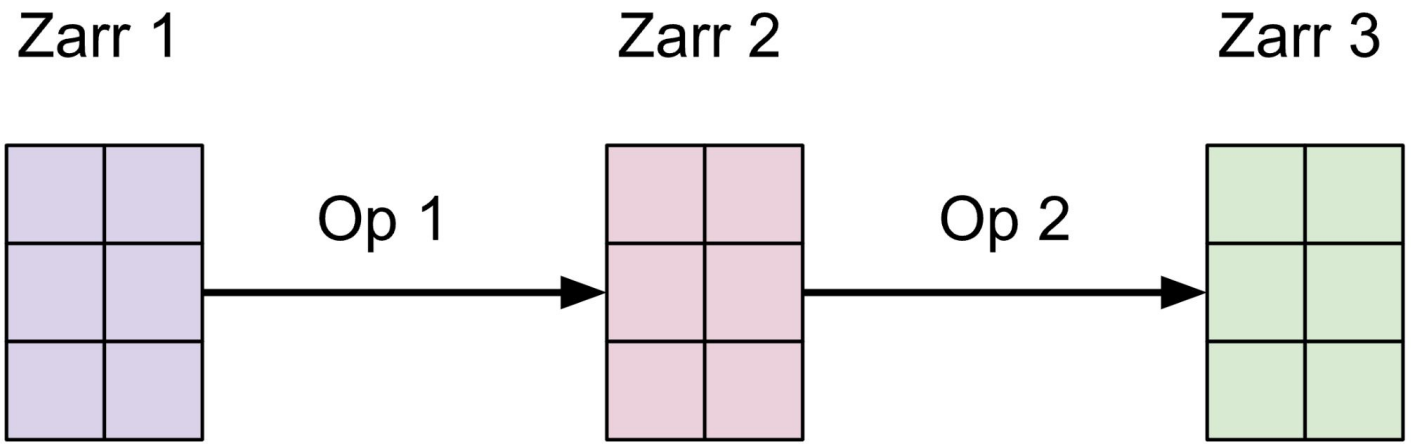
- Expressive
  - Scale without rewriting
- Perfect weak horizontal scaling
  - (1000x problem in 1x time with 1000x CPUs)
- Predictable (no nasty RAM surprises)
- Robust to small failures

Vision for Science at Scale (Tom's  list ) - (2)

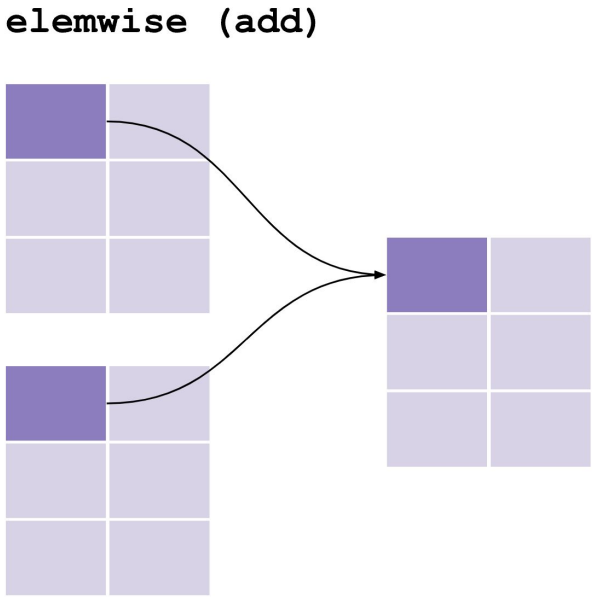
- …
- Resumable
- Forget about the Cluster
- Fully open
  - Not locked in to any one service or platform

What is Cubed?

- Idea: All array operations take Zarr -> Zarr

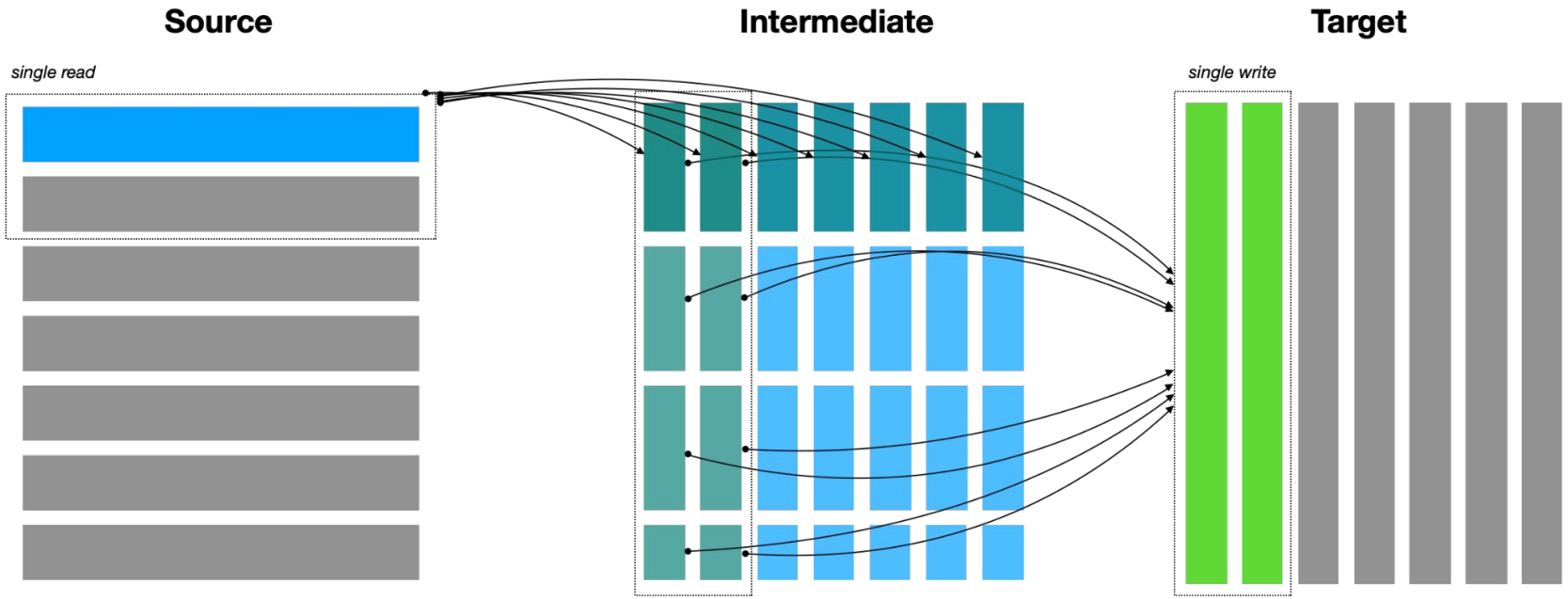


- Many simple array operations are “blockwise”



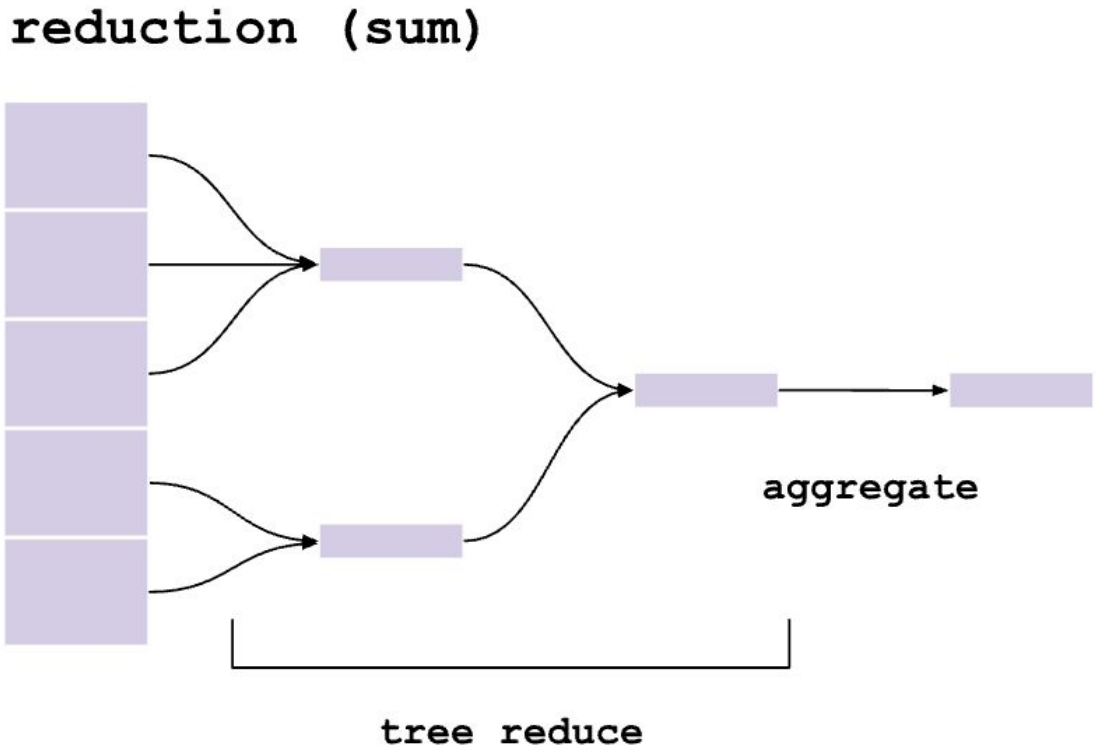
  - Read: embarrassingly parallel across chunks

- But some array operations require a “rechunk”

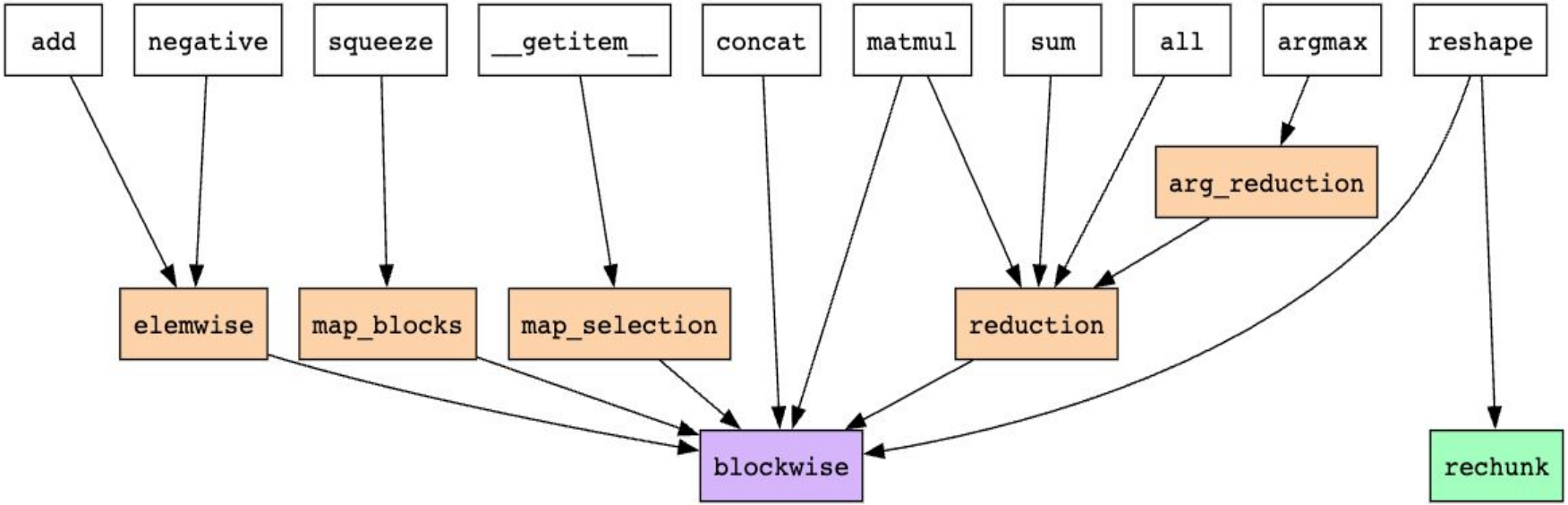


(diagram from pangeo-data/rechunker package)

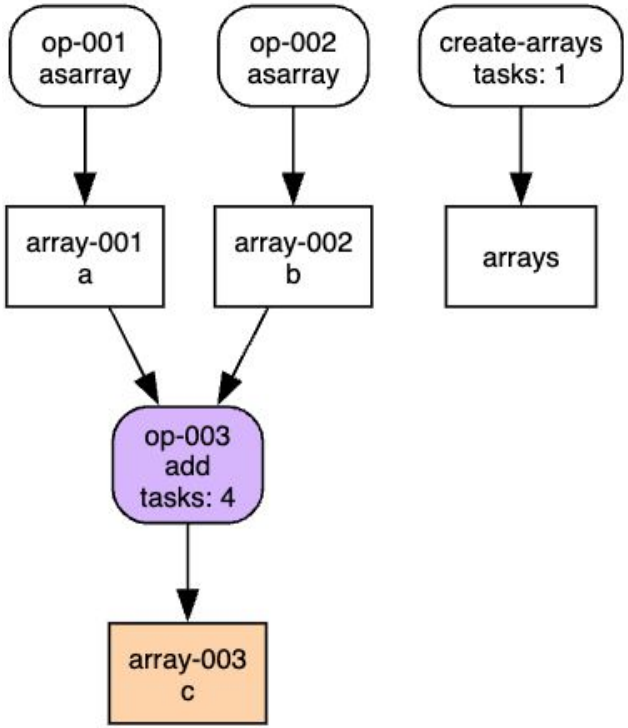
- Some operations are more complex



- Turns out blockwise + rechunk cover all numpy-like array operations!



- Chain operations into a high-level “Plan”
  - Represented at array-level, not chunk-level

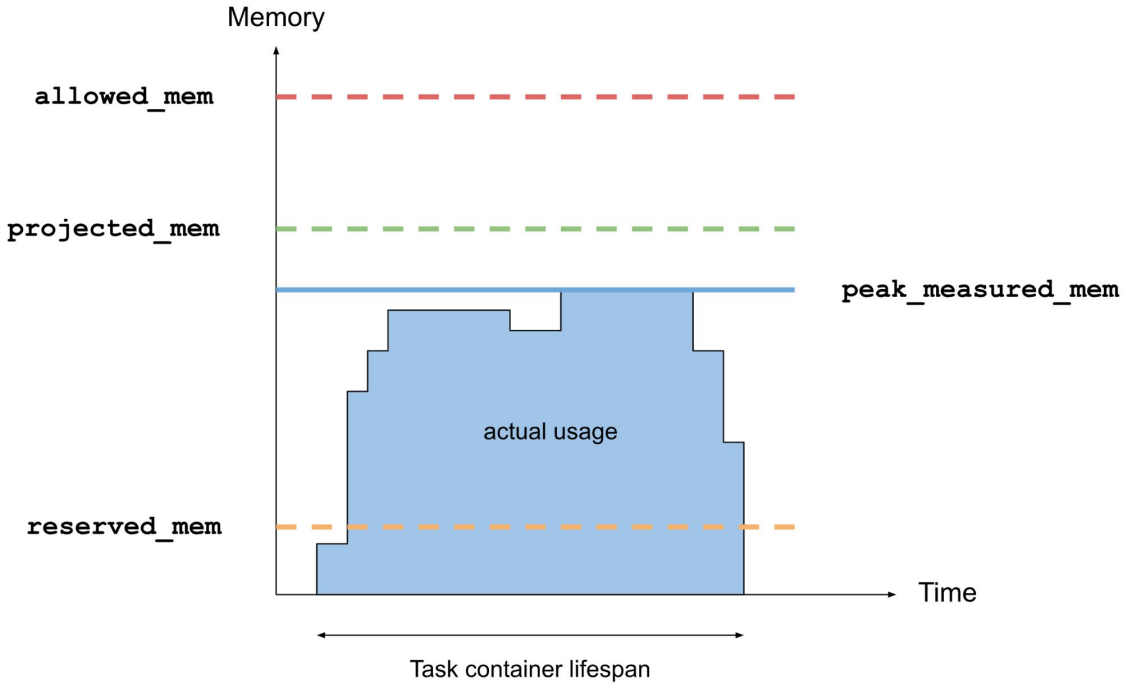


Bounded-memory operations

- Blockwise processes one chunk at a time
- Rechunk can be constant memory if spilling to intermediate Zarr store
  - (see pangeo Rechunker package)

Bounded-memory operations

- Can therefore predict memory usage before launching!

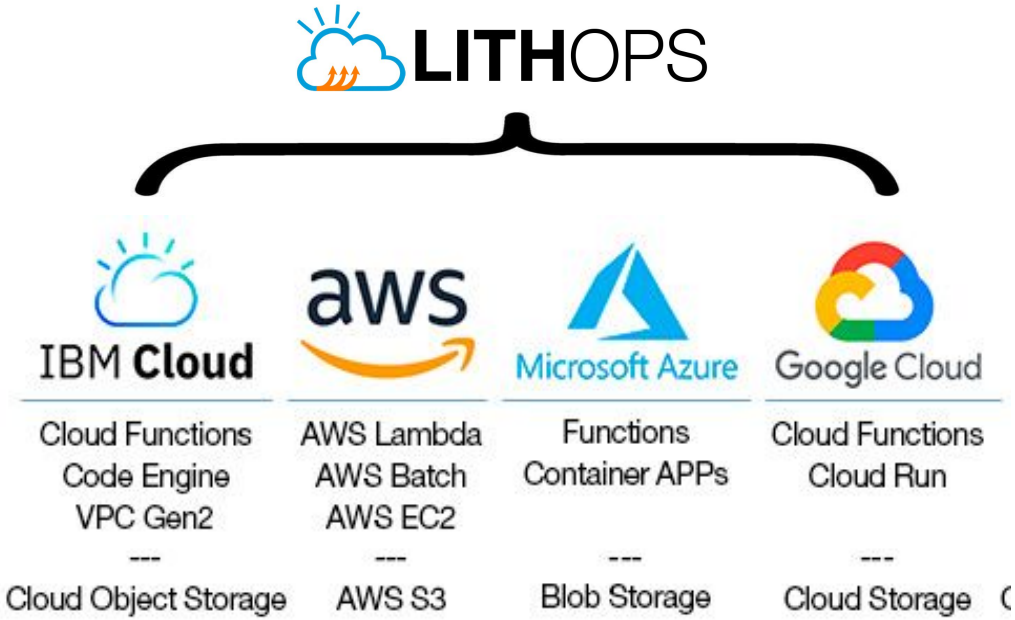


Serverless execution

- Every op is (a series of) embarrassingly parallel tasks
- Just launch them all simultaneously
- Ideal fit for ✨Serverless✨ cloud services
  - e.g. AWS Lambda, Google Cloud Functions
- (Means no cluster to manage!)

Range of Executors

- Serverless



Modal

Modal Apps

Coiled
A Dask Company

Coiled Functions

Range of Executors

- Cluster / HPC

Apache Spark, RAY, dask, Apache Beam, globus, Dragon?

- Single multicore machine

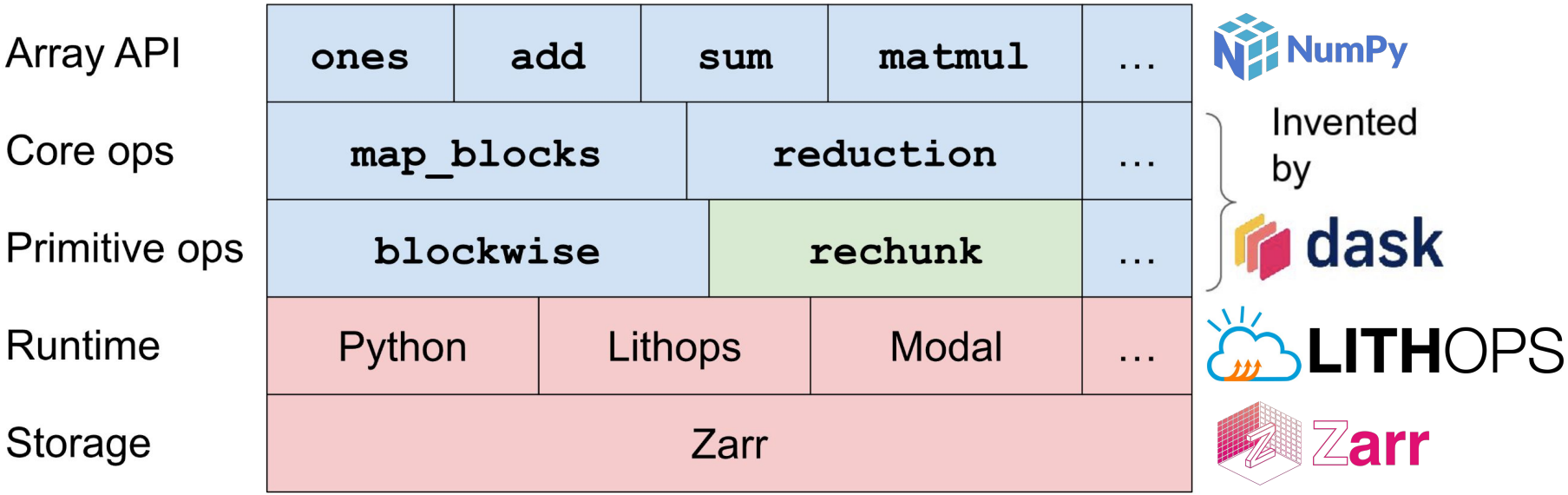
Start on your Laptop

- Process hundreds of GB on your laptop using all available cores

```
  0[||||||||||||||||||||||||||||||||90.8%]   4[||||||||||||||||||||||||||||||||95.4%]
  1[||||||||||||||||||||||||||||||||92.1%]   5[||||||||||||||||||||||||||||||||94.1%]
  2[||||||||||||||||||||||||||||||||92.8%]   6[||||||||||||||||||||||||||||||||91.4%]
  3[||||||||||||||||||||||||||||||||92.8%]   7[||||||||||||||||||||||||||||||||90.2%]
Mem[||||||||||||||||||||| 5.63G/16.0G] Tasks: 491, 2687 thr, 0 kthr; 8 runnin
Swp[|||||||||||||||||||||6.96G/8.00G] Load average: 4.73 2.03 1.55
                                      Uptime: 9 days, 21:30:10
```

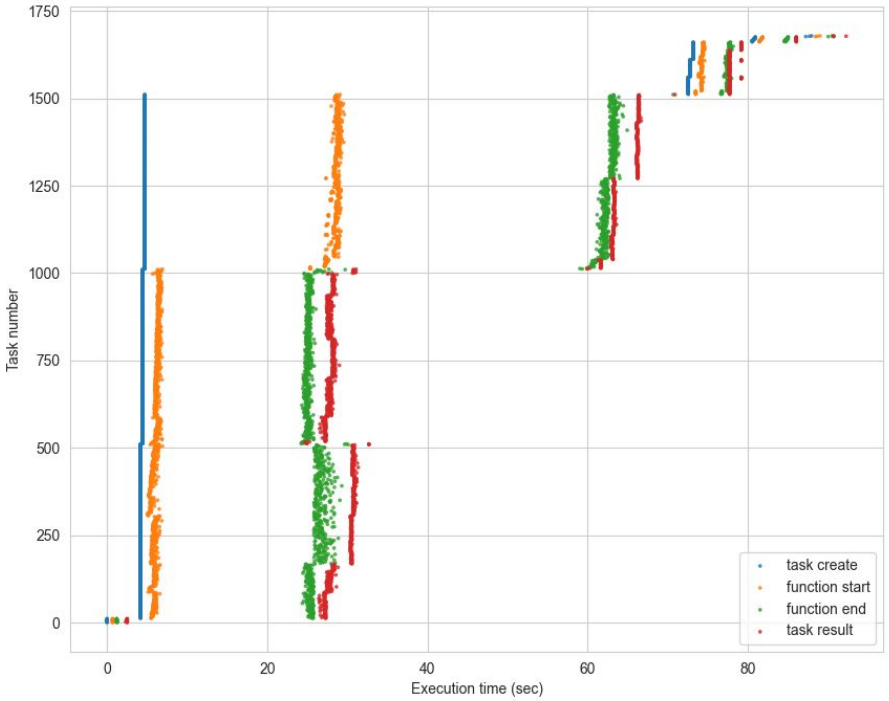
Overall design

- Bounded-Memory Serverless Array Processing



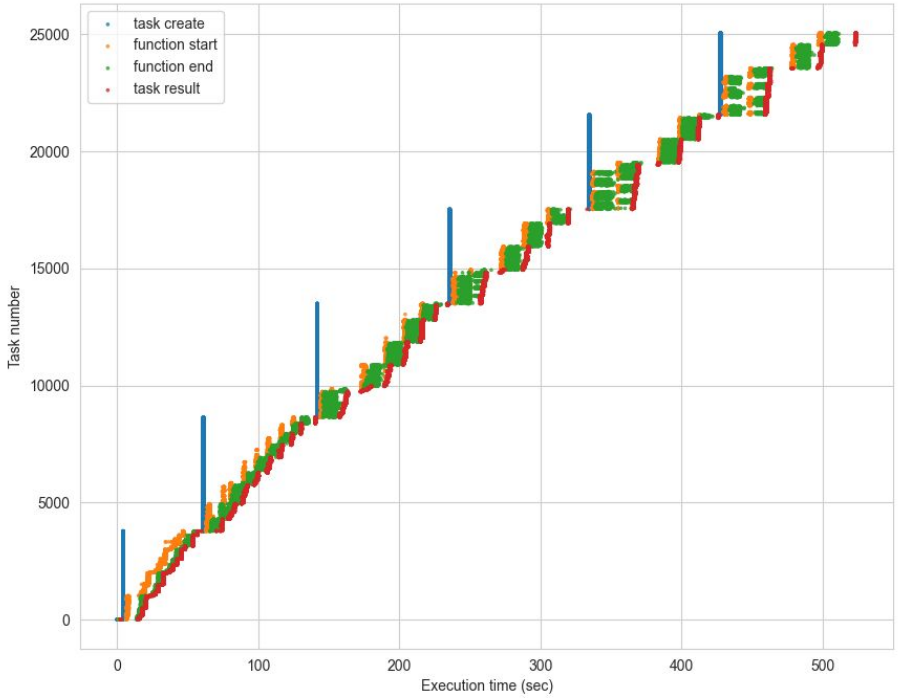
Benchmark: “Quadratic Means” problem

- Reduction computation
- Input: 1.5TB
- Time: 1m 40s
- Lithops on AWS Lambda with 1000 workers



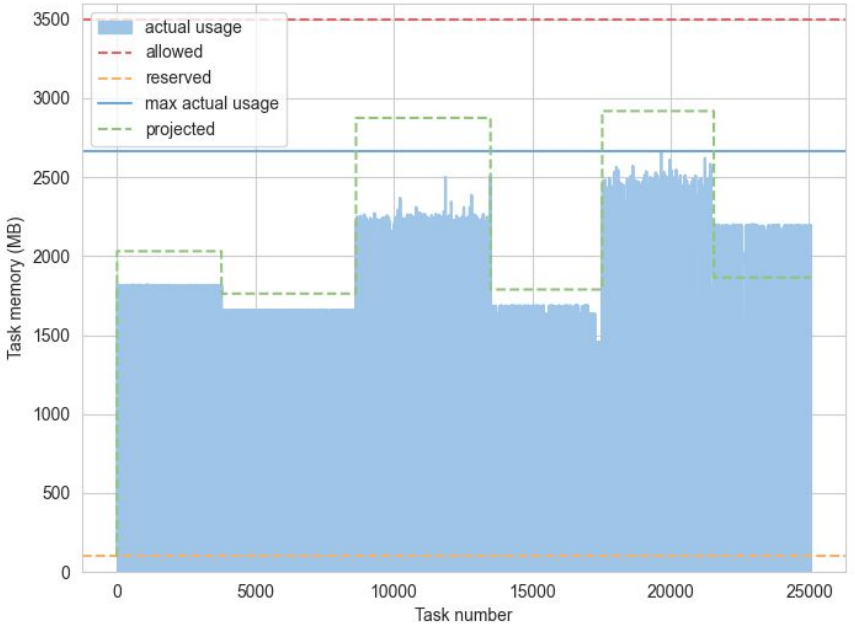
Benchmark: Rechunking ERA5 dataset variable

- All-to-all rechunk
- Multi-stage algorithm
- Input: 1.5TB
- Time: 8m 48s
- Lithops on AWS Lambda with 1000 workers



Benchmark: Rechunking ERA5 dataset variable

- Actual memory usage is always less than allowed maximum
- Room to optimize further



Xarray Integration

- Xarray has been generalized to wrap any chunked array type
- Install cubed & cubed-xarray
- Then specify the allowed memory
  - (And the location for intermediate Zarr stores)

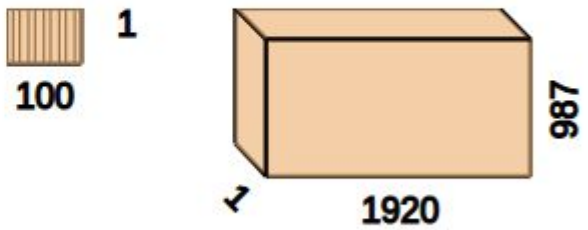
```
from cubed import Spec

spec = Spec(work_dir='tmp', allowed_mem='1GB')
```

Xarray Integration

- Now you can directly open from disk as `cubed.Array` objects

```
ds = open_dataset(
    'data.zarr',
    chunked_array_type='cubed',
    from_array_kwargs={'spec': spec})
    chunks={},
)
```

	Array	Chunk
Bytes	1.41 GiB	144.58 MiB
Shape	(100, 1, 987, 1920)	(10, 1, 987, 1920)
Count	4 arrays in Plan	10 Chunks
Type	float64	np.ndarray



Xarray Integration

- Now just .compute, with your chosen serverless Executor!

```
from cubed.runtime.executors.lithops import LithopsExecutor


ds.compute(executor=LithopsExecutor())
```

Xarray Integration

- Xarray now wraps Cubed OR Dask OR [new things??]

Tabular data: 

Executes via 

Array data: 

Executes via 

Other??

Vision for Science at Scale (Tom's  list )

- Expressive ✅
- Horizontal scaling ✅
- No Cluster ✅
- Predictable RAM usage ✅
- Retry failures ✅
- Resumable ✅
- Fully open ✅

Disadvantages

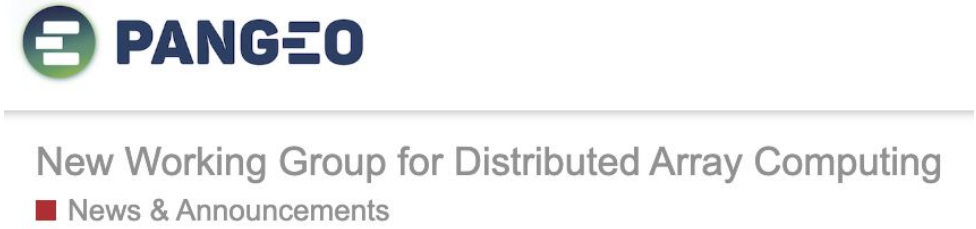
- I/O to Zarr is slow compared to ideal dask case of staying in RAM
- Serverless more expensive per CPU-hour
- Only array operations

Next steps

- We want your use cases to test on!
- Optimizations
- Other array types (JAX on GPU)
- Other storage layers (obstore, zarrs-python)

Read the blog post! xarray.dev/blog/cubed-xarray

- Join the discussion!



Thanks to Tom White for writing Cubed!