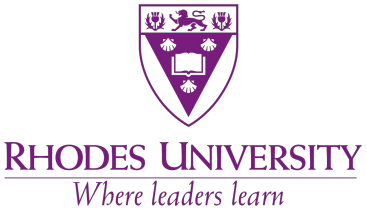
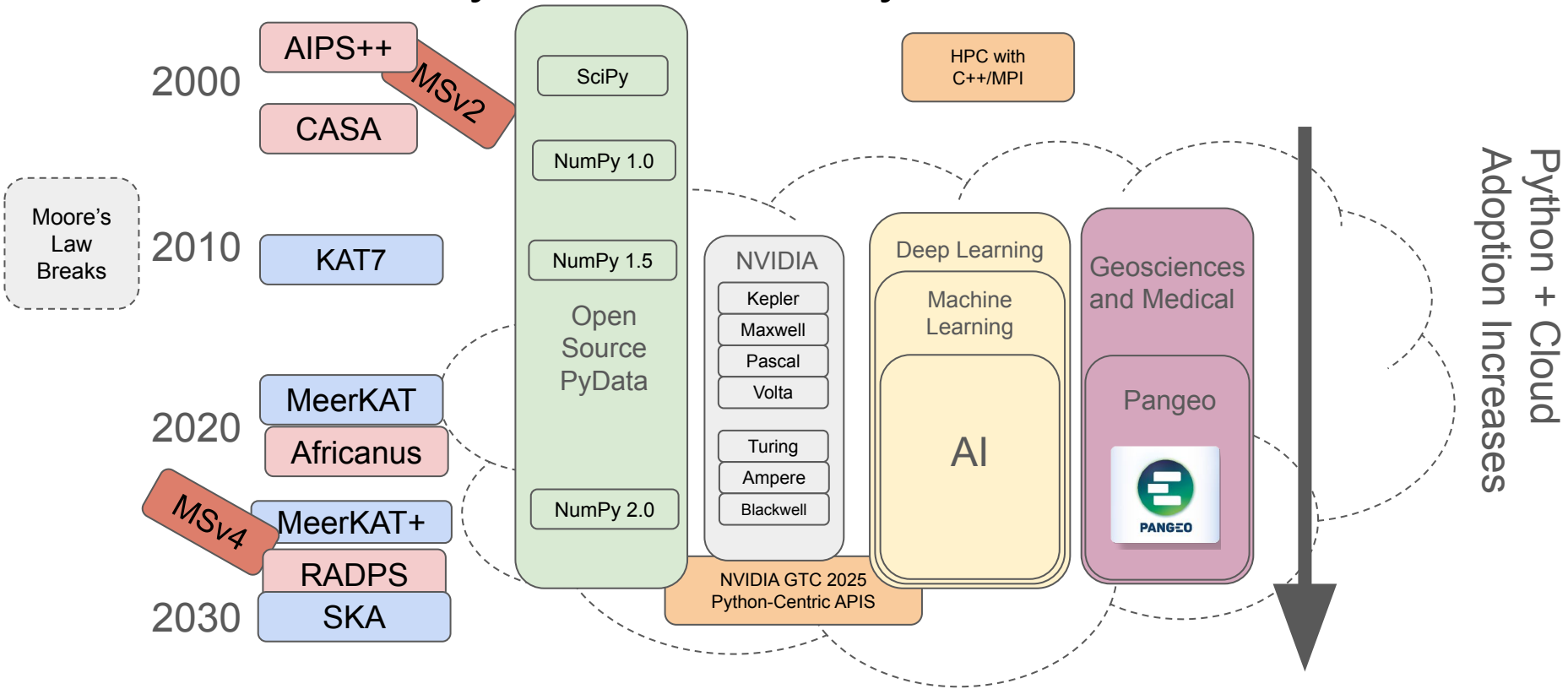
The PyData and Radio Astronomy Ecosystems

Simon Perkins, Jon Kenyon, Landman Bester, Oleg Smirnov and Ben Hugo



Radio Astronomy Software Ecosystems in Context



“Big Data may be a solved problem”

PyData and AI ecosystems offer high quality, well-maintained formats and libraries

Approach pioneered by Geosciences: Pangeo Project

NumPy

Numba

CuPy

dask

xarray

SciPy

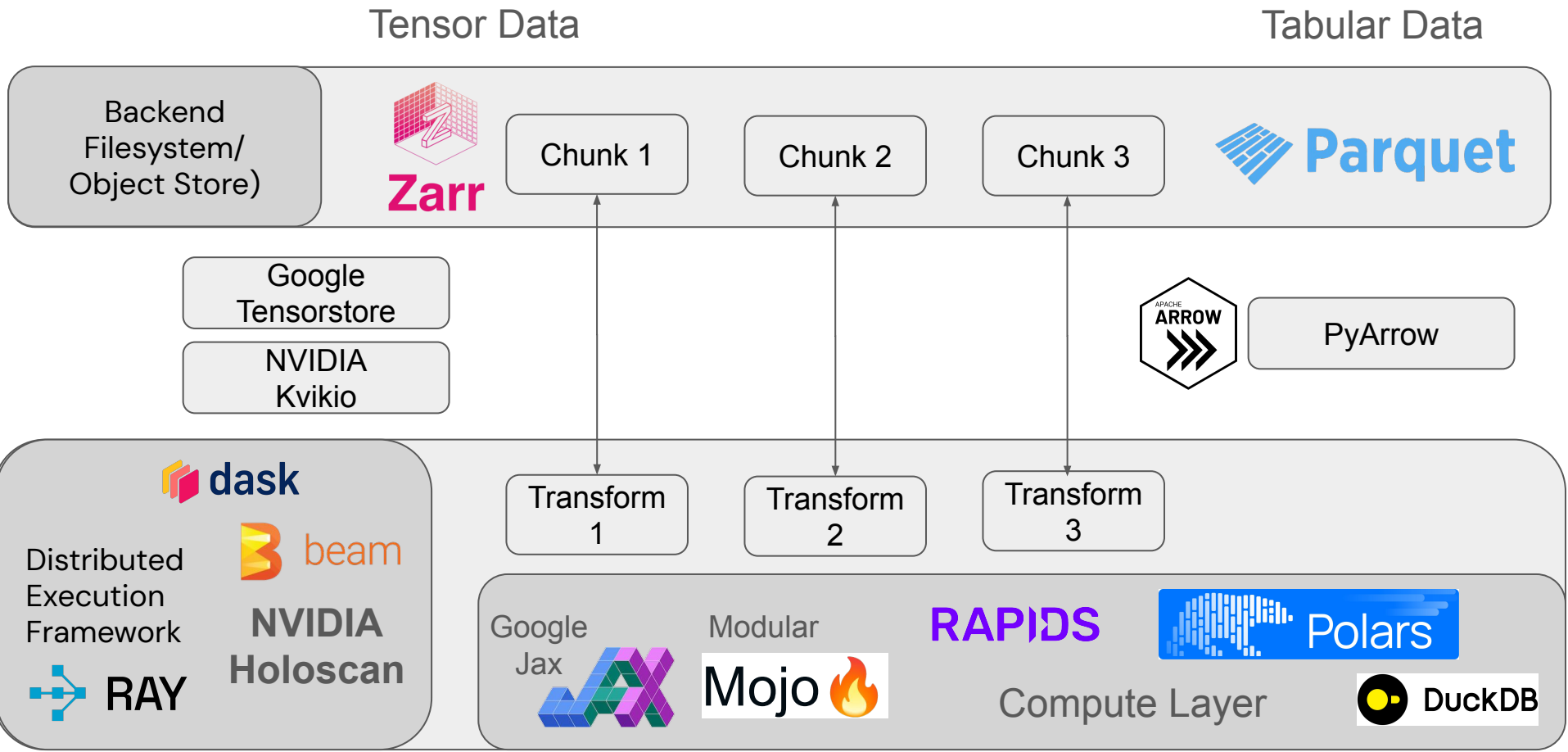
Zarr

APACHE ARROW

astropy
A Community Python Library for Astronomy

PyTorch

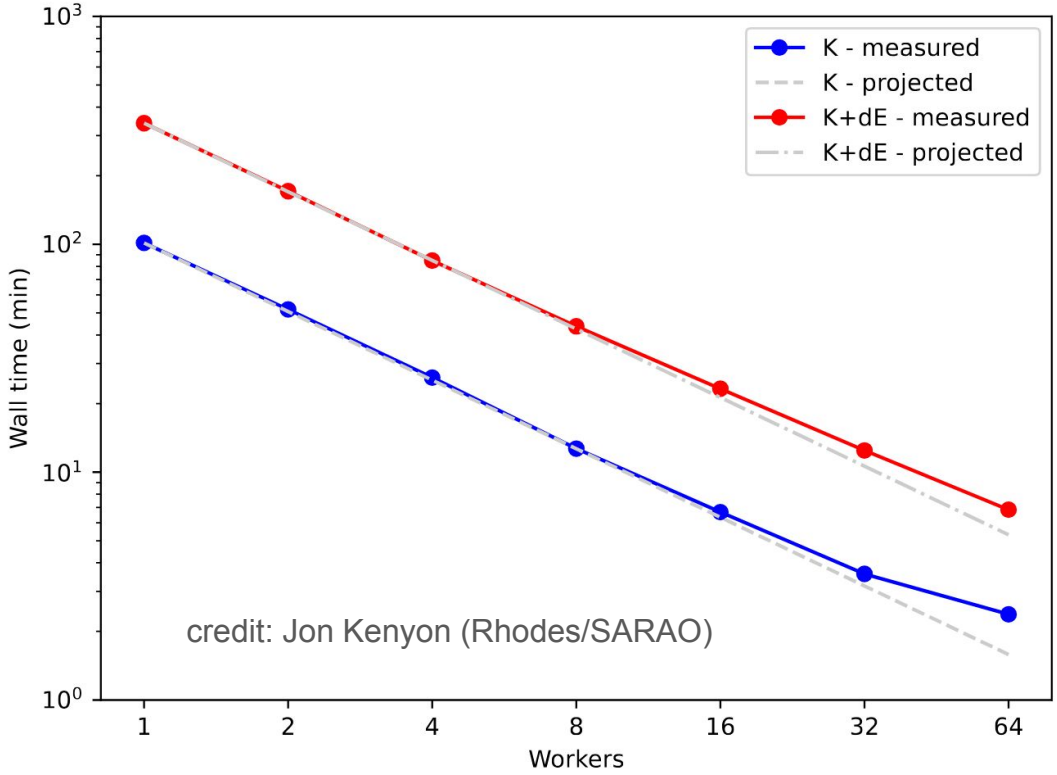
An Opinionated Distributed Compute Architecture



Why Opinionated? Good scaling properties!

Quartical Strong Scaling Experiment on AWS: c6in.8xlarge EC2 instances

- Shard by time/frequency solution intervals

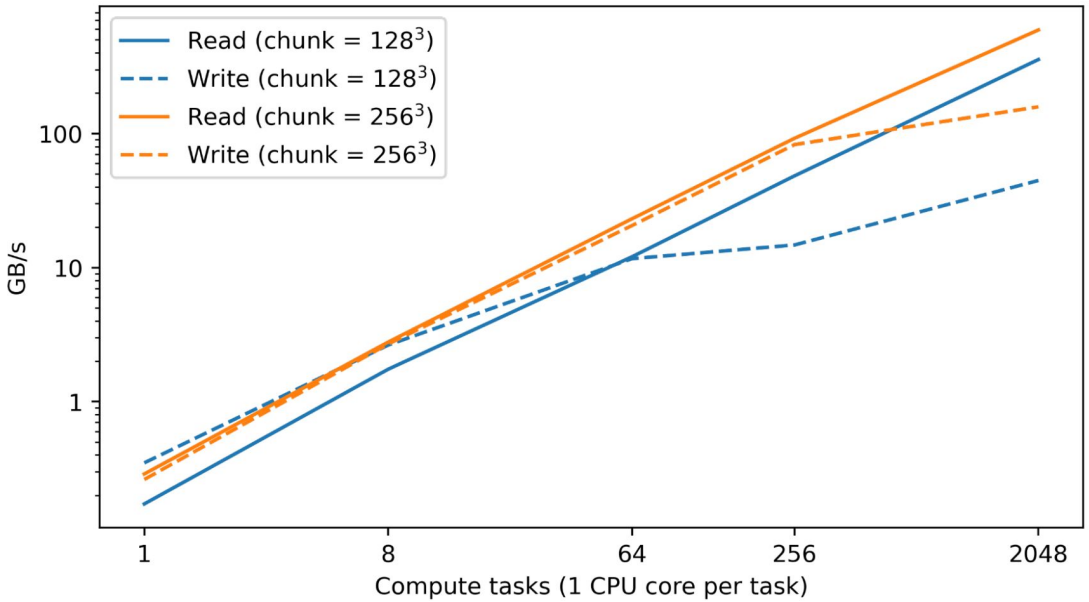


Numba

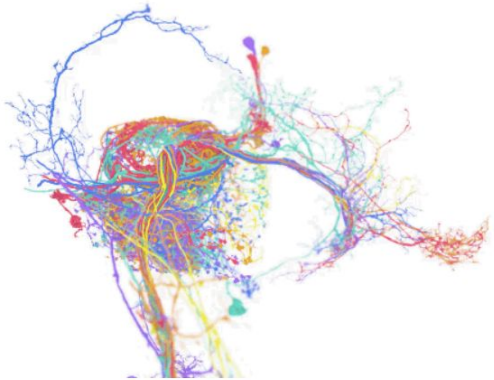
dask

Zarr

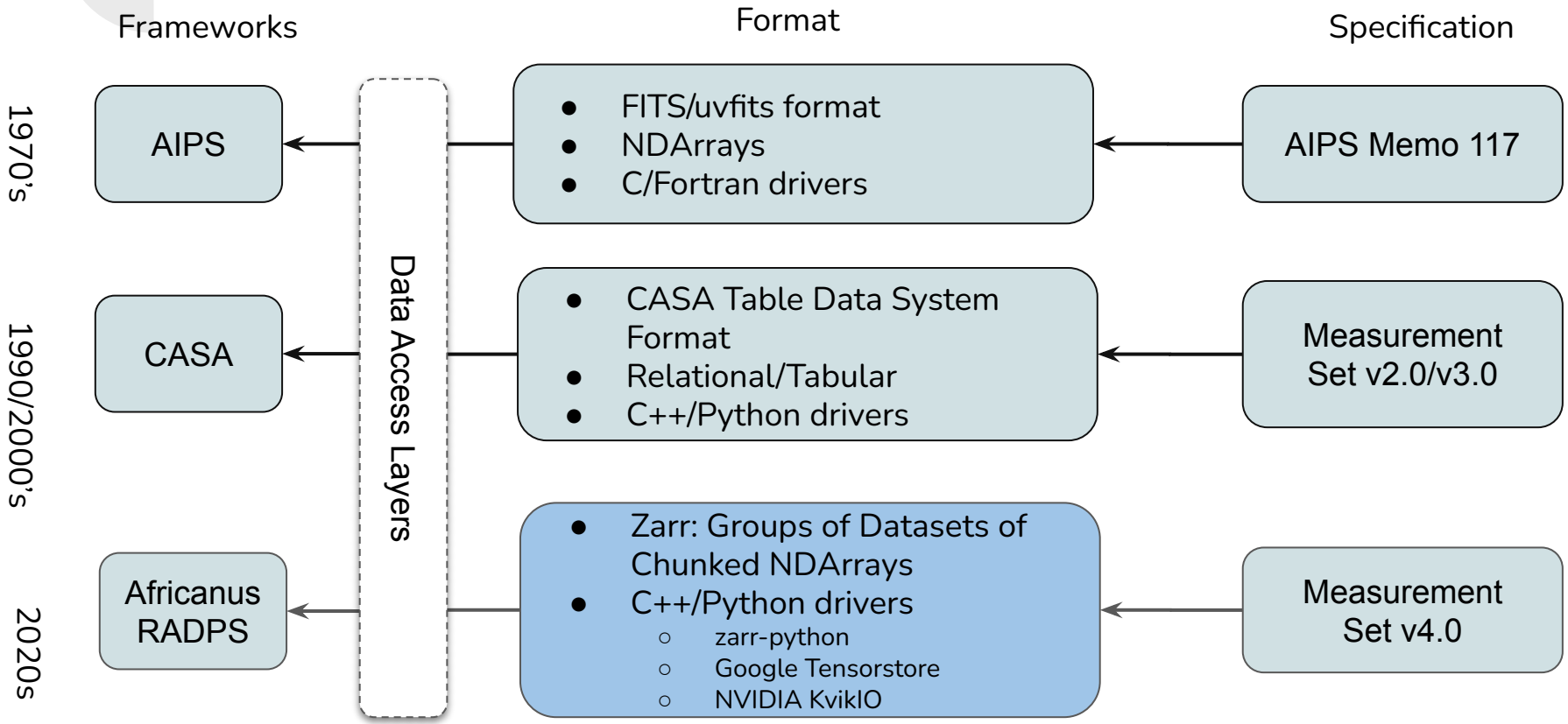
Google Tensorstore raw Zarr scaling (Google Cloud Storage)



- Used for 3D electron microscopy at petascale
- Climate Science
- https://research.google/blog/a-browsable-petascale-reconstruction-of-the-human-cortex/



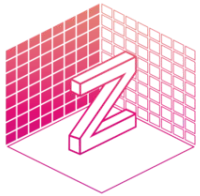
Measurement Set v4 (NRAO/SKAO/SARAO)



“Zarr is motivated by the need for a simple, transparent, open, and community-driven format that supports high-throughput distributed I/O on different storage systems.”

- Conceptually **similar to HDF5**
- **Designed for distributed systems**
  - Distributed **Object Stores** (S3, Google Cloud Storage, Ceph)
  - **Posix** File Systems
- NDArrays **Chunked** into many objects/files
  - Read/Write chunks **independently**
  - **Horizontal Scaling**
  - **Compression**
- **Petascale Datasets**

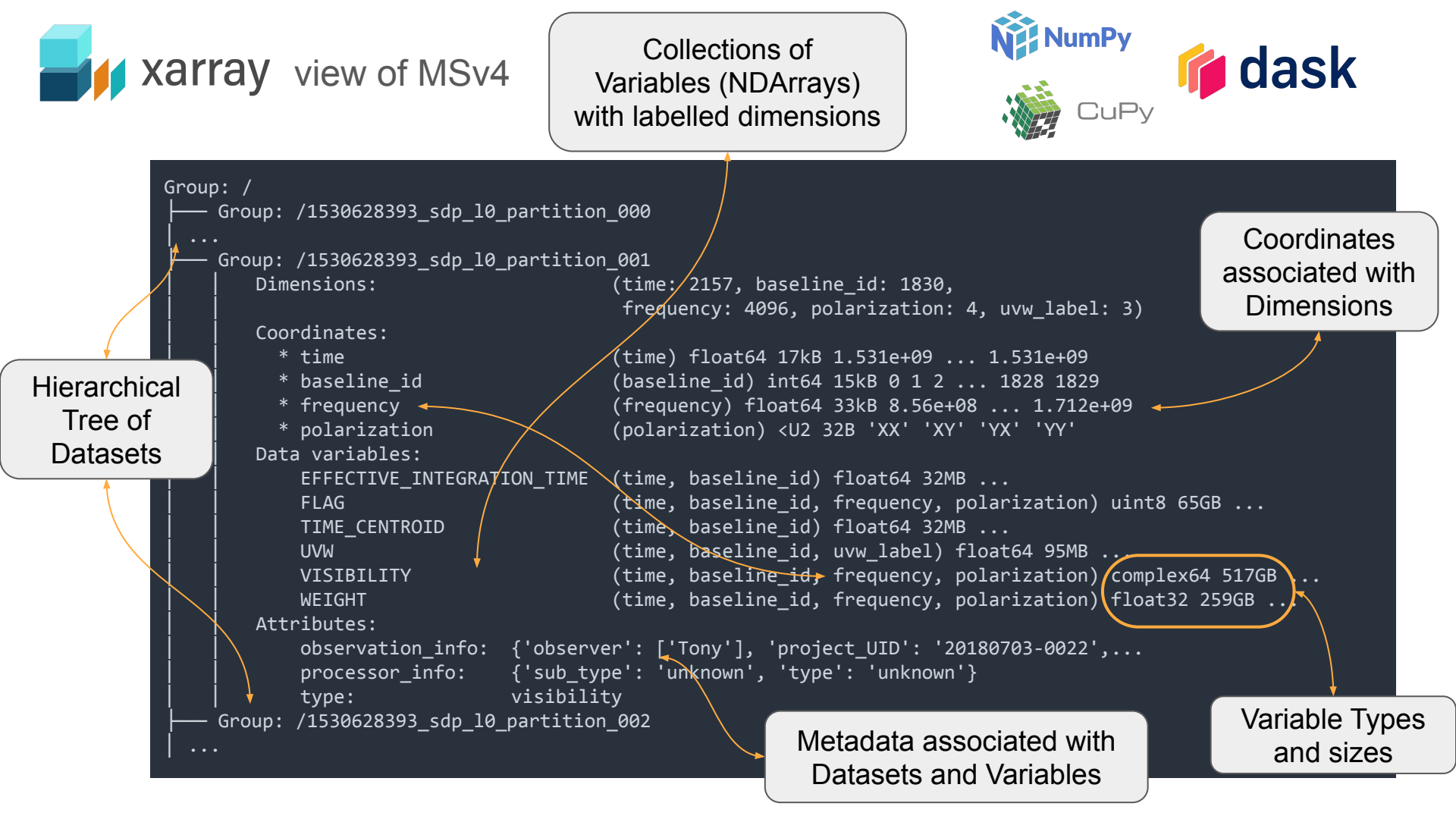
```
>>> ms = zarr.open_group("/tmp/ds.zarr")
>>> uvw = ms.require_dataset("uvw",
                             shape=(10000, 3),
                             chunks=(2500, 3),
                             dtype=np.float64)
>>> data = ms.require_dataset("data",
                             shape=(10000, 64, 4),
                             chunks=(2500, 16, 4),
                             dtype=np.complex64)
>>> data.info
Name               : /data
Type               : zarr.core.Array
Data type          : complex64
Shape              : (10000, 64, 4)
Chunk shape        : (2500, 16, 4)
Order              : C
Read-only          : False
Compressor         : Blosc(cname='lz4', clevel=5,
shuffle=SHUFFLE, blocksize=0)
Store type         : zarr.storage.DirectoryStore
No. bytes          : 20480000 (19.5M)
No. bytes stored   : 83113 (81.2K)
Storage ratio      : 246.4
Chunks initialized : 0/16
```



Zarr

- NumPy interface
- Compression
- In-memory data stores

```
>>> uvw[:] = np.random.random((10000, 3))
>>> data[:] = np.random.random((10000, 64, 4))
>>> data.info
No. bytes          : 20480000 (19.5M)
No. bytes stored   : 9108938 (8.7M)
Storage ratio      : 2.2
Chunks initialized : 16/16
```


xarray view of MSv4
Collections of Variables (NDArrays) with labelled dimensions
NumPy
CuPy
dask
Group: /
├── Group: /1530628393_sdp_l0_partition_000
│ ...
└── Group: /1530628393_sdp_l0_partition_001
Dimensions: (time: 2157, baseline_id: 1830, frequency: 4096, polarization: 4, uvw_label: 3)
Coordinates:
* time (time) float64 17kB 1.531e+09 ... 1.531e+09
* baseline_id (baseline_id) int64 15kB 0 1 2 ... 1828 1829
* frequency (frequency) float64 33kB 8.56e+08 ... 1.712e+09
* polarization (polarization) <U2 32B 'XX' 'XY' 'YX' 'YY'
Data variables:
EFFECTIVE_INTEGRATION_TIME (time, baseline_id) float64 32MB ...
FLAG (time, baseline_id, frequency, polarization) uint8 65GB ...
TIME_CENTROID (time, baseline_id) float64 32MB ...
UVW (time, baseline_id, uvw_label) float64 95MB ...
VISIBILITY (time, baseline_id, frequency, polarization) complex64 517GB ...
WEIGHT (time, baseline_id, frequency, polarization) float32 259GB ...
Attributes:
observation_info: {'observer': ['Tony'], 'project_UID': '20180703-0022',...
processor_info: {'sub_type': 'unknown', 'type': 'unknown'}
type: visibility
├── Group: /1530628393_sdp_l0_partition_002
│ ...
Coordinates associated with Dimensions
Hierarchical Tree of Datasets
Metadata associated with Datasets and Variables
Variable Types and sizes

xarray

is a **lazy** interface/presentation layer for data
It abstracts the underlying data store away from the user

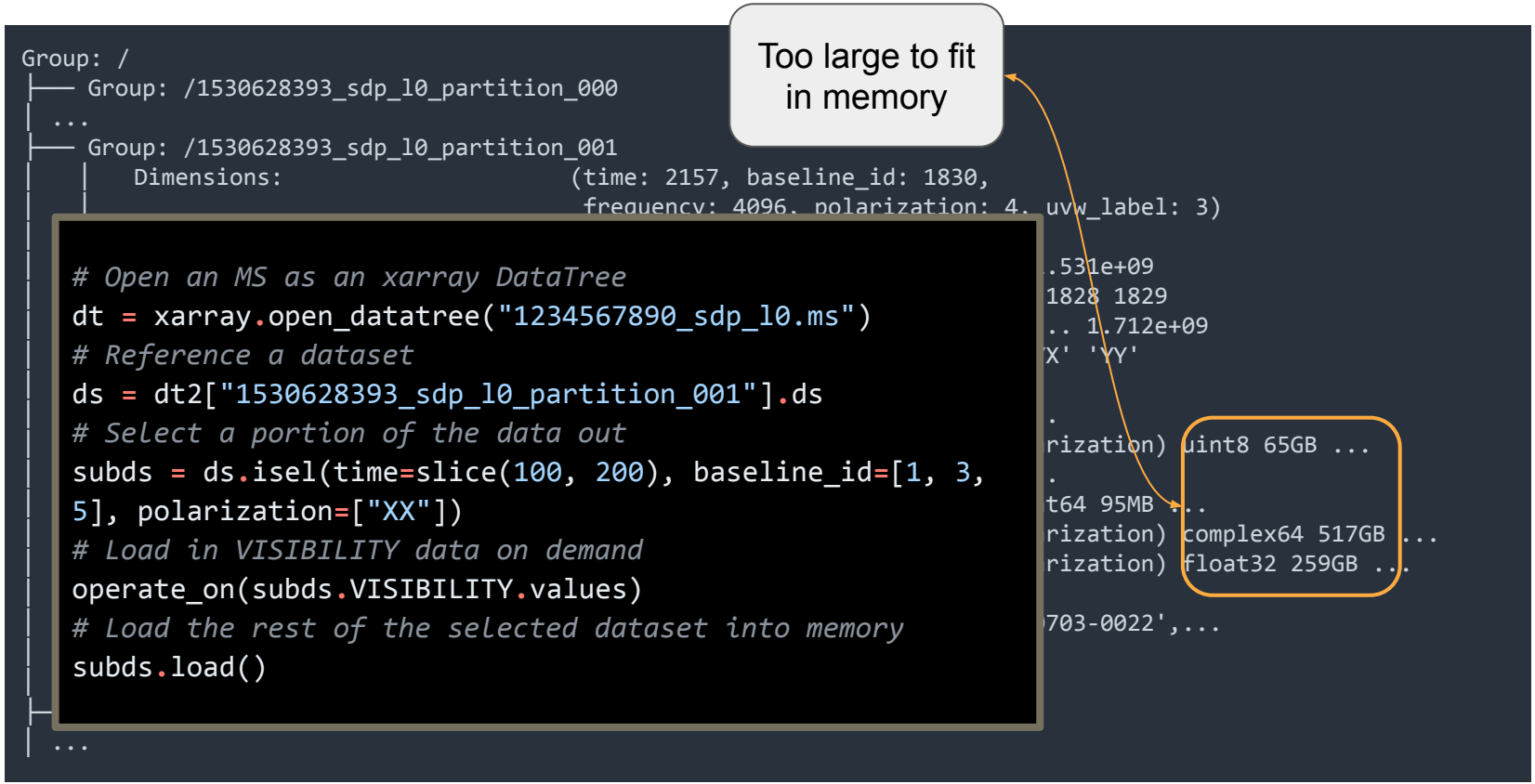
```
Group: /
├── Group: /1530628393_sdp_l0_partition_000
│   ...
├── Group: /1530628393_sdp_l0_partition_001
│   │   Dimensions:                    (time: 2157, baseline_id: 1830,
│   │                                   frequency: 4096, polarization: 4, uv
│   │   Coordinates:
│   │     * time                       (time) float64 17kB 1.531e                e+09
│   │     * baseline_id                (baseline_id) int64 1                ... 1828 1829
│   │     * frequency                  (frequency) flo                B 8.56e+08 ... 1.712e+09
│   │     * polarization               (polarizati                32B 'XX' 'XY' 'YX' 'YY'
│   │   Data variables:
│   │       EFFECTIVE_INTEGRATION_TIME (time, baseline_id) float64 32MB ...
│   │       FLAG                       (time, baseline_id, frequency, polarization) uint8 65GB ...
│   │       TIME_CENTROID              (time, baseline_id) float64 32MB ...
│   │       UVW                        (time, baseline_id, uvw_label) float64 95MB ...
│   │       VISIBILITY                 (time, baseline_id, frequency, polarization) complex64 517GB ...
│   │       WEIGHT                     (time, baseline_id, frequency, polarization) float32 259GB ...
│   │   Attributes:
│   │       observation_info:  {'observer': ['Tony'], 'project_UID': '20180703-0022',...
│   │       processor_info:    {'sub_type': 'unknown', 'type': 'unknown'}
│   │       type:              visibility
├── Group: /1530628393_sdp_l0_partition_002
│   ...
```

800GB too large to fit in memory

FORMAT BOF BOF BOF!!!

xarray

is a **lazy** interface/presentation layer for data
Can load variables on demand and select out regions



Different backends, same interface

```python
import xarray_ms
import xarray

ms = "1234567890_sdp_l0.ms"
dt = xarray.open_datatree(ms)
# Export to Zarr
dt.to_zarr("1234567890_sdp_l0.zarr”)
```

MSv4 view over MSv2 data view xarray-ms

```python
import xarray

zms = "1234567890_sdp_l0.zarr"
dt = xarray.open_datatree(zms)
```

Zarr via xarray

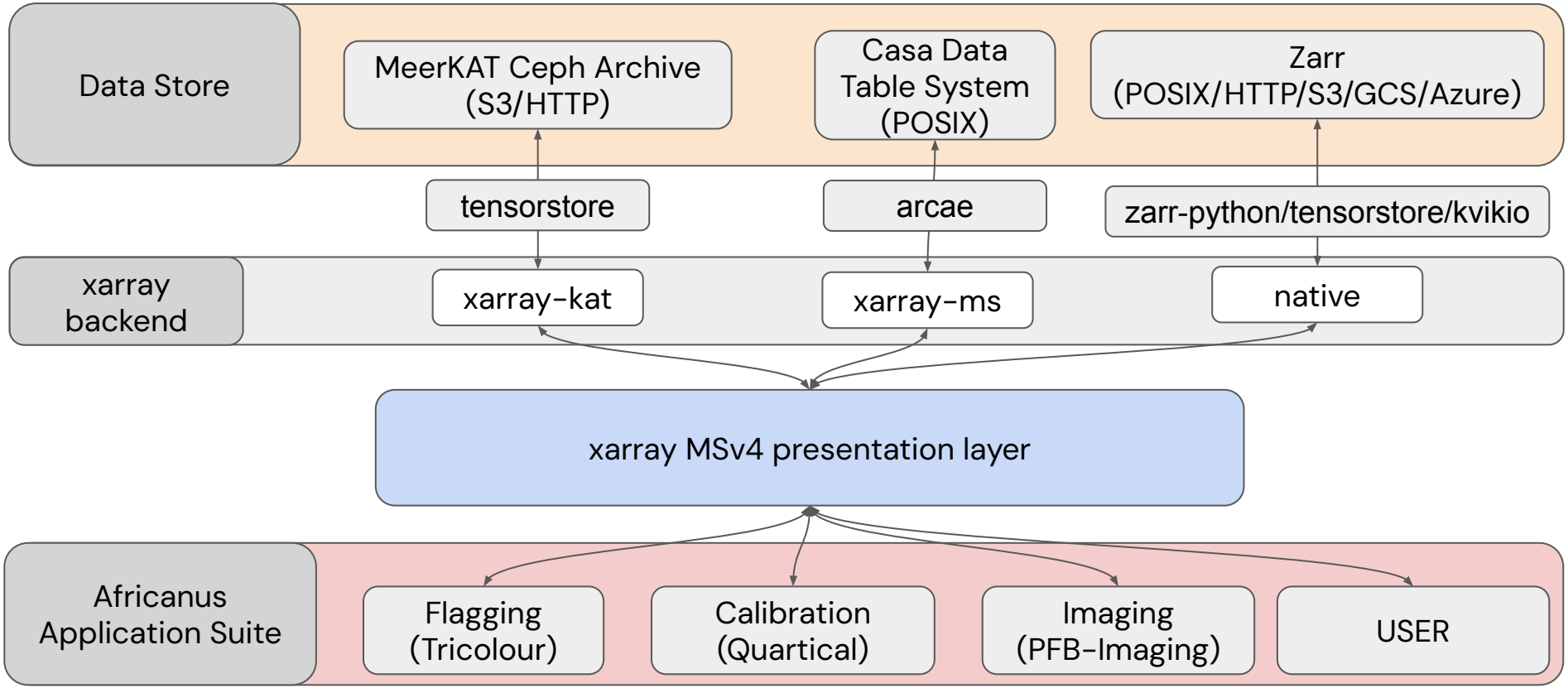
```python
import xarray_kat
import xarray

url = "https://archive-gw-1.kat.ac.za/..."
dt = xarray.open_datatree(url)
# Export to Zarr
dt.to_zarr("1234567890_sdp_l0.zarr")
```

MSv4 view over the MeerKAT Archive via xarray-kat

```python
# Select out a dataset
ds = dt["1234567890_sdp_l0_000"].ds
# Select a subset of the dataset
sub_sel_ds = ds.isel(
  time=slice(100, 200),
  baseline_id=slice(20, 60),
  frequency=slice(256, 768))
# Load in the subselection
sub_sel_ds.load()
# Pass data into imaging function...
image(sub_sel_ds.VISIBILITY.values,
      sub_sel_ds.WEIGHT.values,
      sub_sel_ds.FLAG.values)
```

“Imaging Workflow”

Africanus Ecosystem adapted to MSv4



Time to sum 517GB visibilities and 259GB weights? CASA left vs Zarr right

```
In [42]: import xarray

In [43]: import xarray_ms

In [44]: dt = xarray.open_datatree("1530628393_sdp_l0.ms",
          chunks={"time": 128, "baseline_id": 128},
          ninstances=16)

In [45]: ds = dt["1530628393_sdp_l0_partition_001"].ds

In [46]: %time dask.compute(
          ds.VISIBILITY.data.sum(),
          ds.WEIGHT.data.sum())
CPU times: user yymin zzs, sys: xh ymin zs, total: xh yymin zzs
Wall time: ymin zzs

Out[46]: (np.complex64(2.198126e+07-5.5812296e+07j),
np.float32(1.3533127e+14))
```

```
In [48]: import xarray

In [49]: dt2 = xarray.open_datatree(
    "1530628393_sdp_l0.zarr", chunks={})

In [50]: ds = dt2["1530628393_sdp_l0_partition_001"].ds



In [51]: %time dask.compute(
          ds.VISIBILITY.data.sum(),
          ds.WEIGHT.data.sum())
CPU times: user yymin zzs, sys: xh yymin zzs, total: yymin zzs
Wall time: ymin zzs

Out[51]: (np.complex64(2.198126e+07-5.5812296e+07j),
np.float32(1.3533127e+14))
```

Time to sum 517GB visibilities and 259GB weights? CASA left vs Zarr right

xarray-ms powered by **arcae** a python-casacore clone supporting multi-threaded access to CASA tables.

```
In [44]: dt = xarray.open_datatree("1530628393_sdp_l0.ms",
             chunks={"time": 128, "baseline_id": 128},
             ninstances=16)

In [45]: ds = dt["1530628393_sdp_l0_partition_001"].ds

In [46]: %time dask.compute(
             ds.VISIBILITY.data.sum(),
             ds.WEIGHT.data.sum())
CPU times: user 21min 19s, sys: 1h 7min 3s, total: 1h 28min 23s
Wall time: 4min 32s

Out[46]: (np.complex64(2.198126e+07-5.5812296e+07j),
np.float32(1.3533127e+14))
```

- AMD EPYC 7702P 64-Core Processor, 528GB RAM
- 4 Seagate FireCuda 530 ZP4000GM30013 nvme in RAID0
- 5,645MB/s theoretical sequential read. ~22,580MB/s seq read in RAID0
- 16,600 MB/s in fio sequential read benchmark

```
In [48]: import xarray

             ds.VISIBILITY.data.sum(),
             ds.WEIGHT.data.sum())
CPU times: user 23min 56s, sys: 28min 5s, total: 52min 2s
Wall time: 6min 55s

Out[51]: (np.complex64(2.198126e+07-5.5812296e+07j),
np.float32(1.3533127e+14))
```

Zarr may spend more time decompressing

Conclusion: There's a plethora of software that makes working with large data easier and faster.

- PFB-Imaging
- Galactic Centre has a complex structure
- Classic CLEAN algorithm => un-physical result
- Variant of SARA algorithm for deconvolution
- 2X longer, 2X more memory than CLEAN, but far less manual effort, more physically correct result

Credit: Bester & Heywood