Getting started

If you haven't done so already, please clone the tutorial repository and start the Jupyter notebook server:

```
git clone https://github.com/tomkooij/scipy2017

cd scipy2017/notebooks

jupyter notebook
```

HDF5 take 2: h5py and PyTables

July 10, 2017.

Tom Kooij

The HDF Group

Tutorial outline 1.30pm - 5.30pm

- Introduction
- Part 1: HDF5 Basics: Using the hierarchy, datatypes and attributes [1 hour?]
- Part 2: Chunking and compression [45 min]
- *Break for afternoon snacks around 3pm*
- Part 3: Queries, Indexing and out-of-core calculations [45 min]
- Part 4: Pandas and HDF5 [30 min]
- Part 5: Advanced HDF5: Low level API, Parallel HDF5 [45 min]

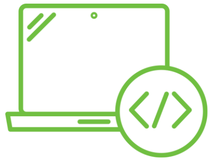
Tom Kooij, M.Sc.

- Nikhef: Dutch National Institute for Subatomic Physics. HiSPARC Cosmic Ray outreach.
- Physics Teacher: Coornhert Gymnasium Gouda, Netherlands.
- PyTables maintainer

Twitter: @tomkooij
Github: tomkooij

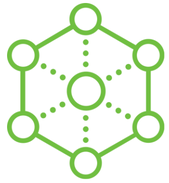
Who is the HDF Group?

HDF Group has developed open source solutions for Big Data challenges for nearly 30 years



Small company (~ 40 employees) with focus on High Performance Computing and Scientific Data

Offices in Champaign, IL + Boulder, CO



Our flagship platform – HDF5 – is at the heart of our open source ecosystem.

Tens of thousands use HDF5 every day, as well as build their own solutions (~~600~~ ~~700~~ 800+ projects on Github)

“De-facto standard for scientific computing” and integrated into every major analytics + visualization tool

What does the HDF Group do?

Products	• HDF5 Community (Open Source) + Enterprise (Future) • Connectors: ODBC + Cloud (Beta) • Add-Ons: compression + encryption
Support	• HDF Support Packages (Basic + Pro + Premier) • Support for h5py + PyTables + pandas (NEW) • Training
Consulting	• HDF: new functionality + performance tuning for specific platforms • General HPC software engineering with scientific expertise • Metadata science and expert services

Our Industries

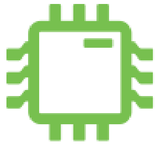
Financial Services

Oil and Gas

Aerospace

Automotive

Medical & Biotech



Silicon Manufacturing

Electronics Instrument

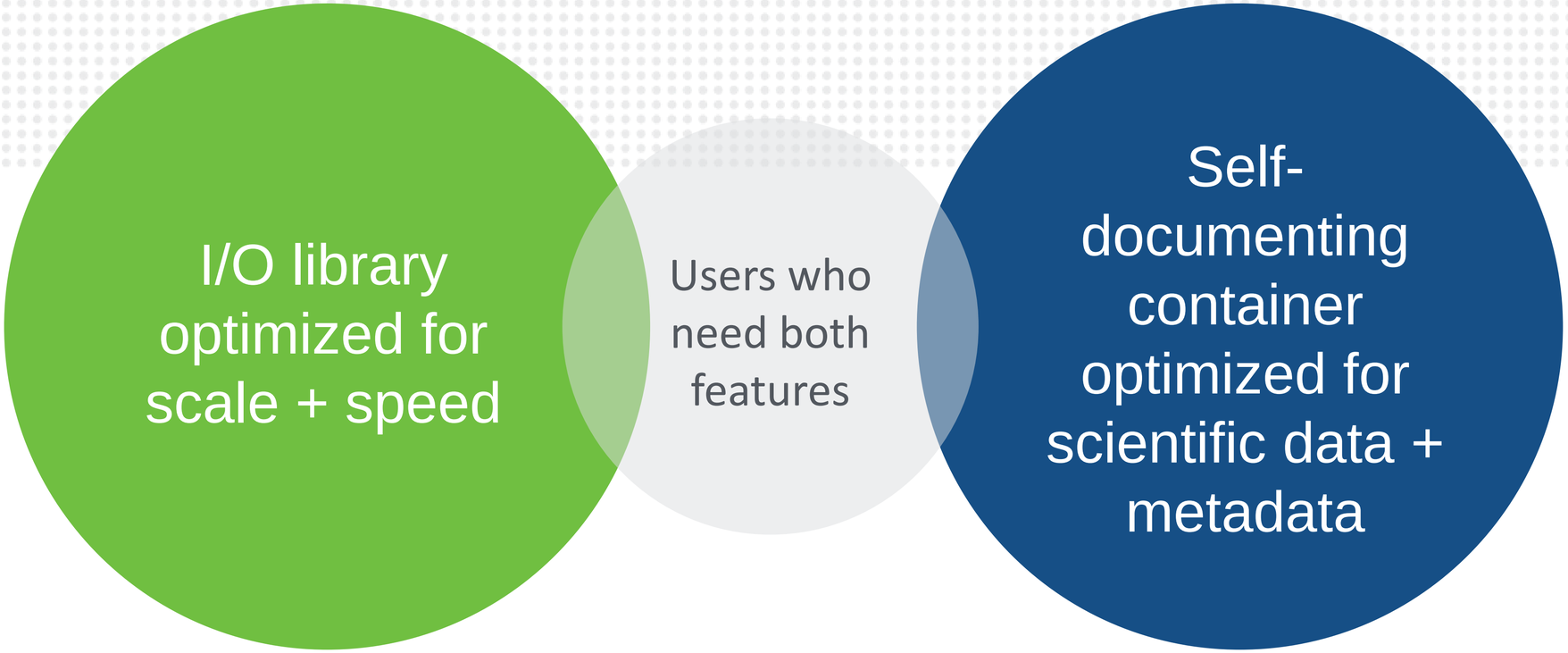
Government

Defense & National Security

Academic Research

A few of our users

Why Use HDF5?

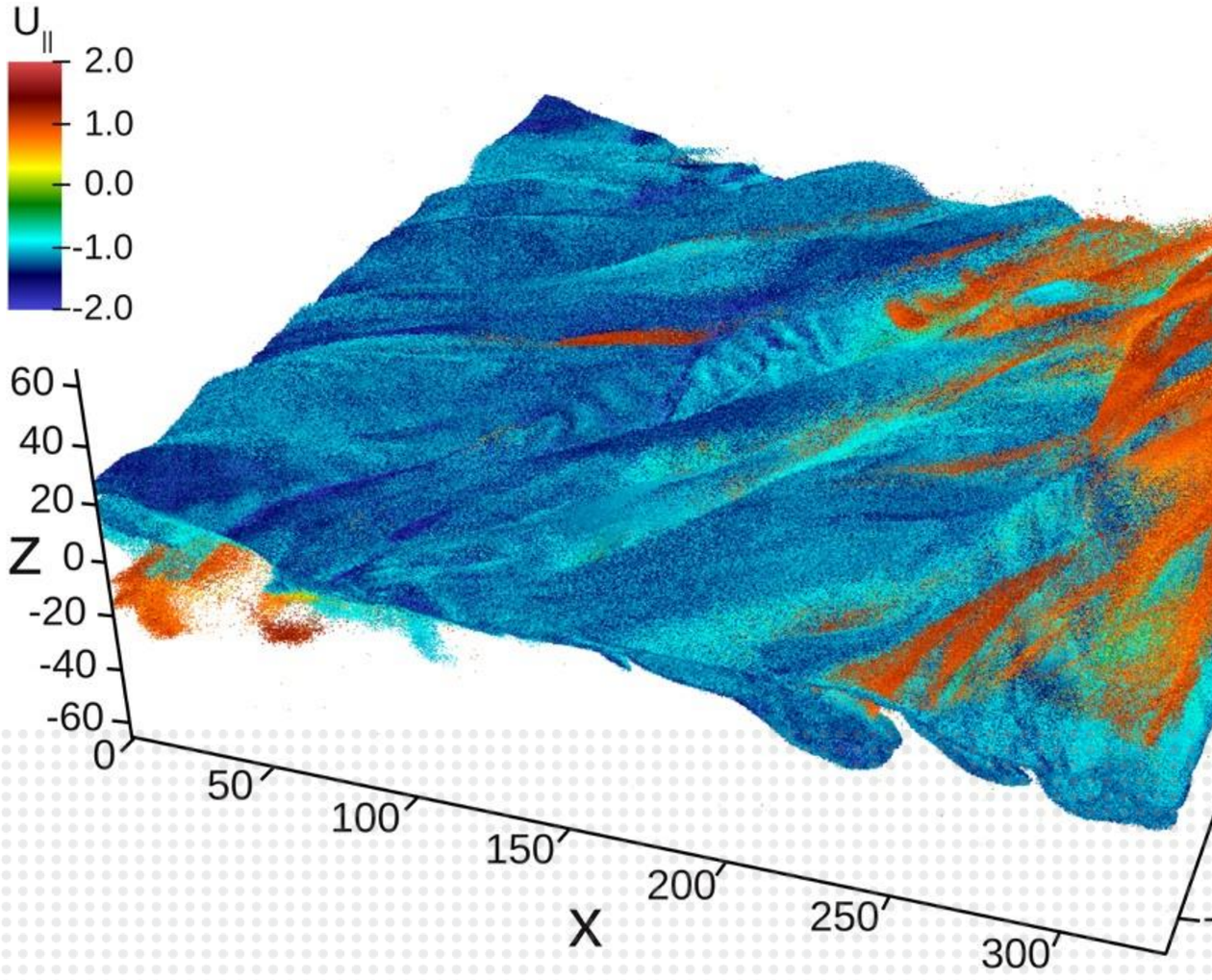

I/O library optimized for scale + speed
Users who need both features
Self-documenting container optimized for scientific data + metadata

TRILLION-PARTICLE SIMULATION

Lawrence Berkeley National Laboratory (LBNL)

- Complex collisions of particle that light up the aurora borealis can fracture Earth's magnetic shield and wreak havoc on electronics, power grids, and space satellites

- Visualization of trillion-particle datasets made possible with HDF5 are helping scientists decipher how.



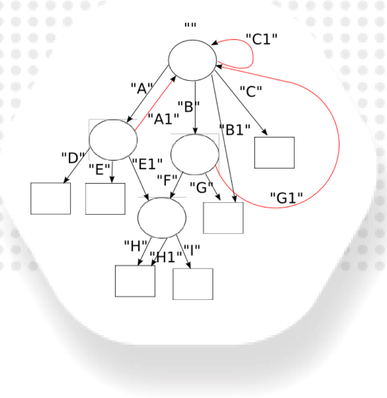
EARTH OBSERVING SYSTEM

NASA

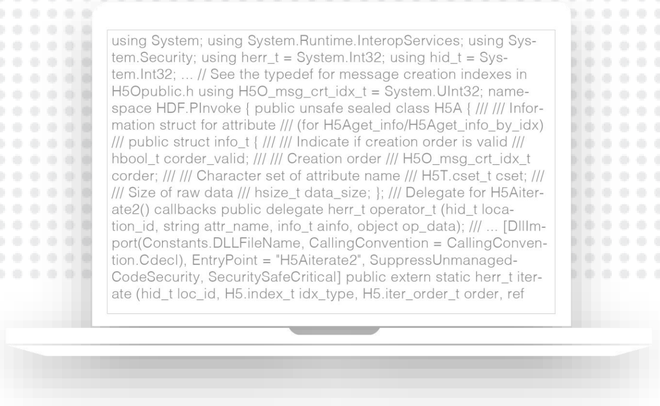
- Deliver 6,700 Different Data Products to 12 Data Archive Centers
- Nearly 16 terabytes per day are redistributed to more than 1.7 million end users worldwide

The HDF5 Platform

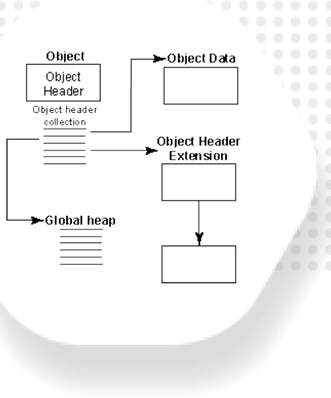
Marriage of data model + I/O software + binary container



HDF5 **abstract data model**



HDF5 **library**

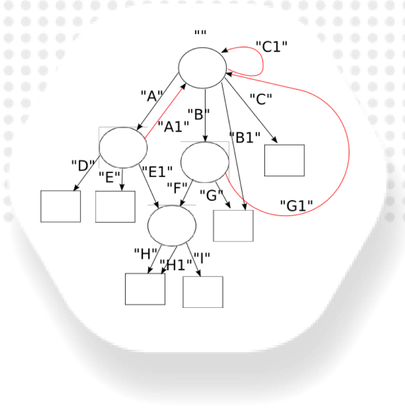


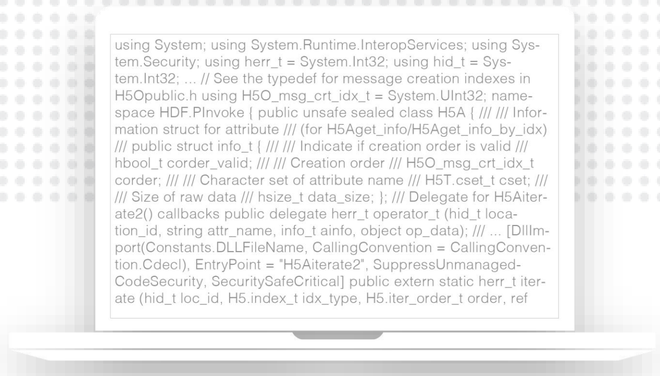
HDF5 **file format**

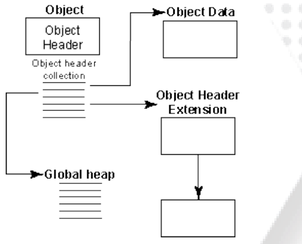
When People say ‘HDF5’…

…They usually mean one of the following:

1. A *Data Model* that organizes array variables in a property graph
2. A *Library* that maps/manages model instances in storage contexts (core, FS, net, obj. store)
3. A self-describing *Format* for serializing model instances into single or multi-file layouts
4. The *technology stack* that includes 1. - 3.
5. A domain- specific format implemented on top of 4. (HDF5 as a *Universal File Format*)
6. An *Ecosystem* (language bindings, 3rd party apps., standards





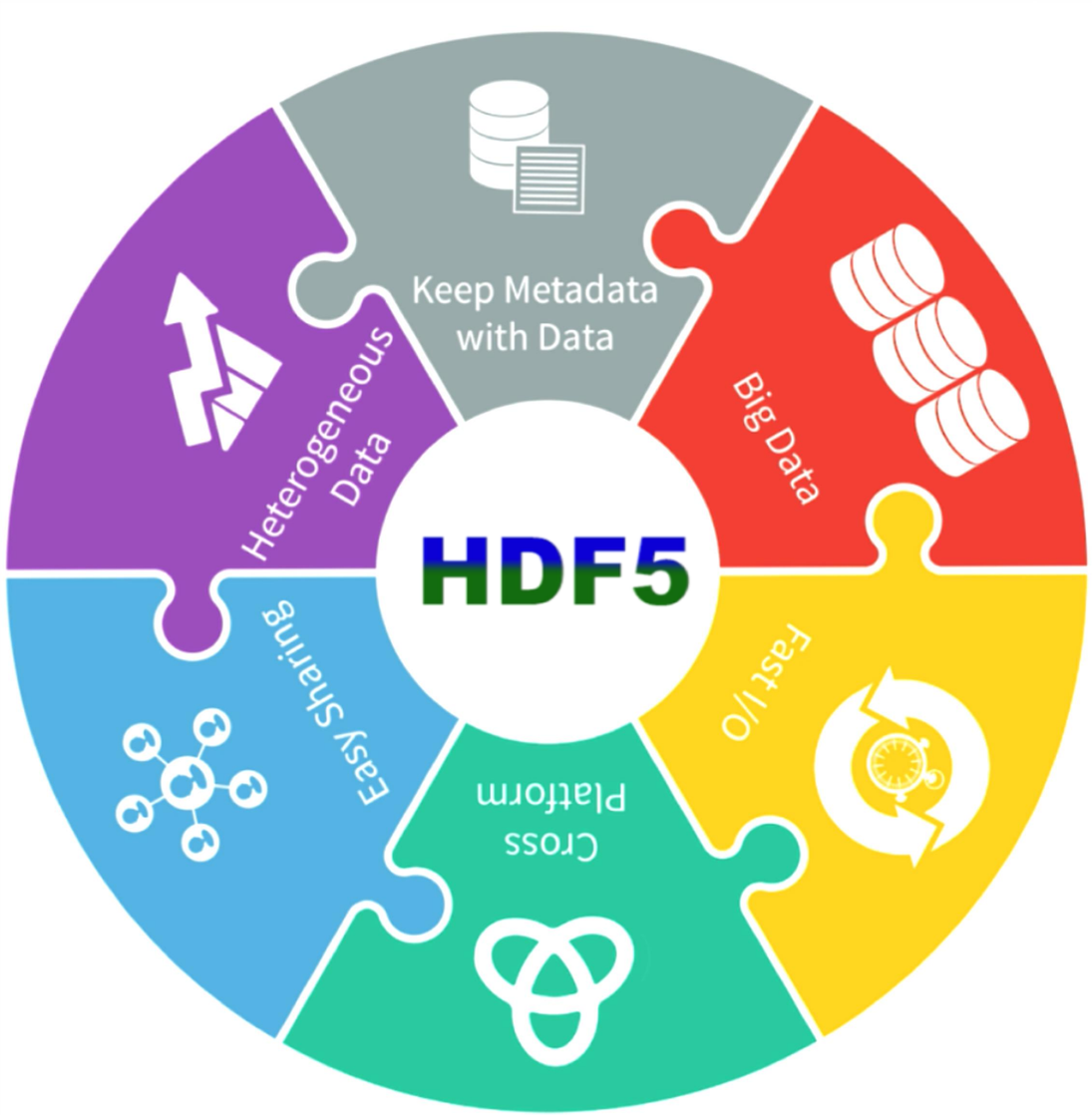


HDF5 **data model**

HDF5 **library**

HDF5 **file format**

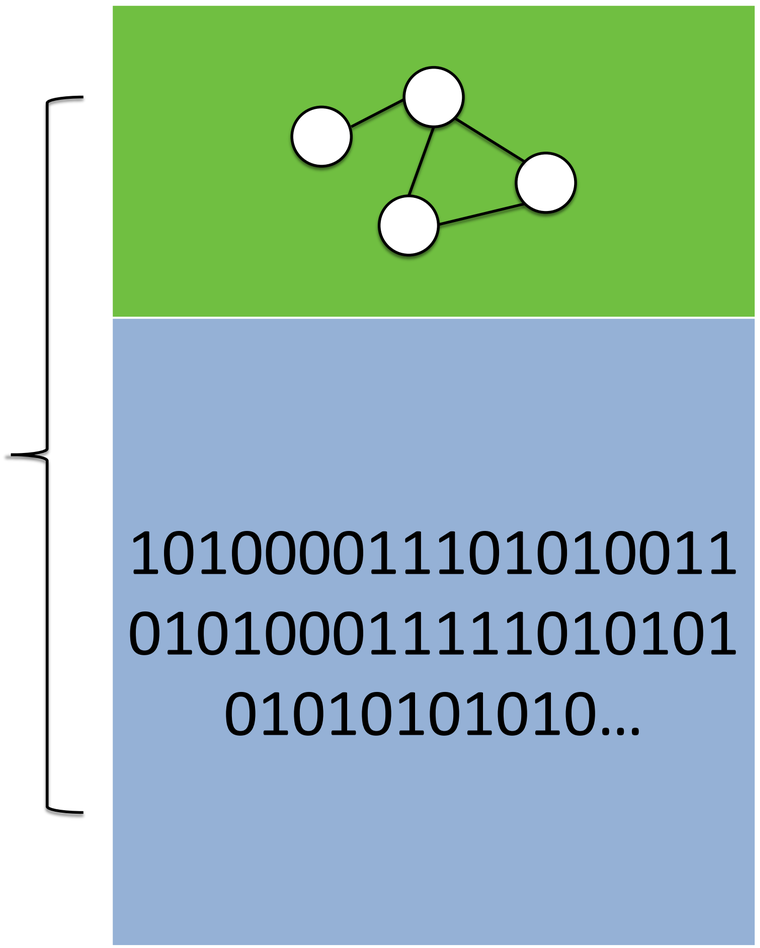
Why is this concept so different + useful?



- **Native support for multidimensional data**
- **Data and metadata in one place => streamlines data lifecycle & pipelines**
- Portable, no vendor lock-in
- Maintains logical view while adapting to storage context
- In-memory, over-the-wire, on-disk, parallel FS, object store
- Pluggable filter pipeline for compression, checksum, encryption, etc.
- High-performance I/O
- Large ecosystem (700+ Github projects)

HDF5 Container

NOT an accurate depiction of the physical layout!



Metadata (“data about data”, documentation)

- Structure + organization
- Types and encodings
- Names
- Array shapes
- Conventions
- Annotations

Data (payload, variable, measurement, “raw”)

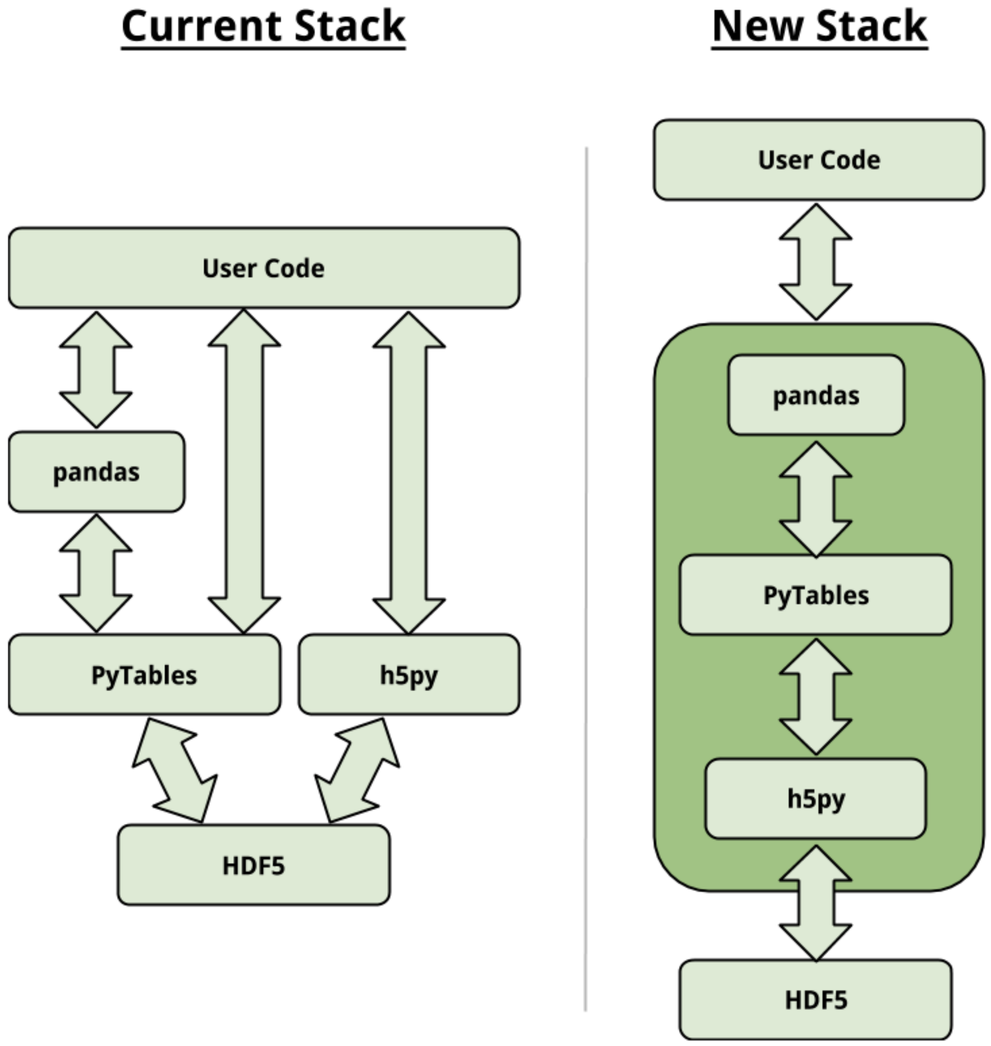
- Problem-sized (small to large)
- Binary
- Optionally
  - compressed
  - encrypted
  - checksum’d
  - “filtered”

HDF5 in Python

- **H5py**
  - Pythonic (numpy-like) interface to HDF5
  - Low level interface to HDF5 library
  - Exposes the complete HDF5 API in Python.

- **PyTables**
  - High level API
  - Advanced features on-top of HDF5 library
    - Indexing (OPSI)
    - Out-of-core calculations (numexpr)
    - Advanced compression (Blosc)

Packages with overlapping capabilities…



https://www.hdfgroup.org/2015/09/python-hdf5-a-vision/

Part 1

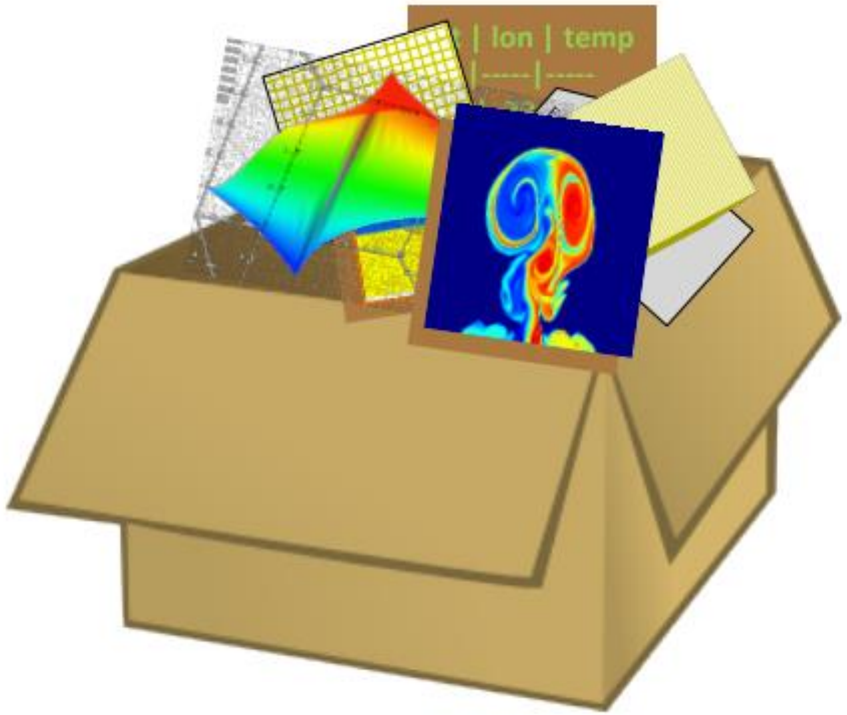
HDF5 Basics

Using the hierarchy, datatypes, attributes and more

Warmup. h5py vs PyTables.

HDF5 Container

An HDF5 file is a **container** that holds data objects.



HDF5 Elements

Dataset

- Array

Group

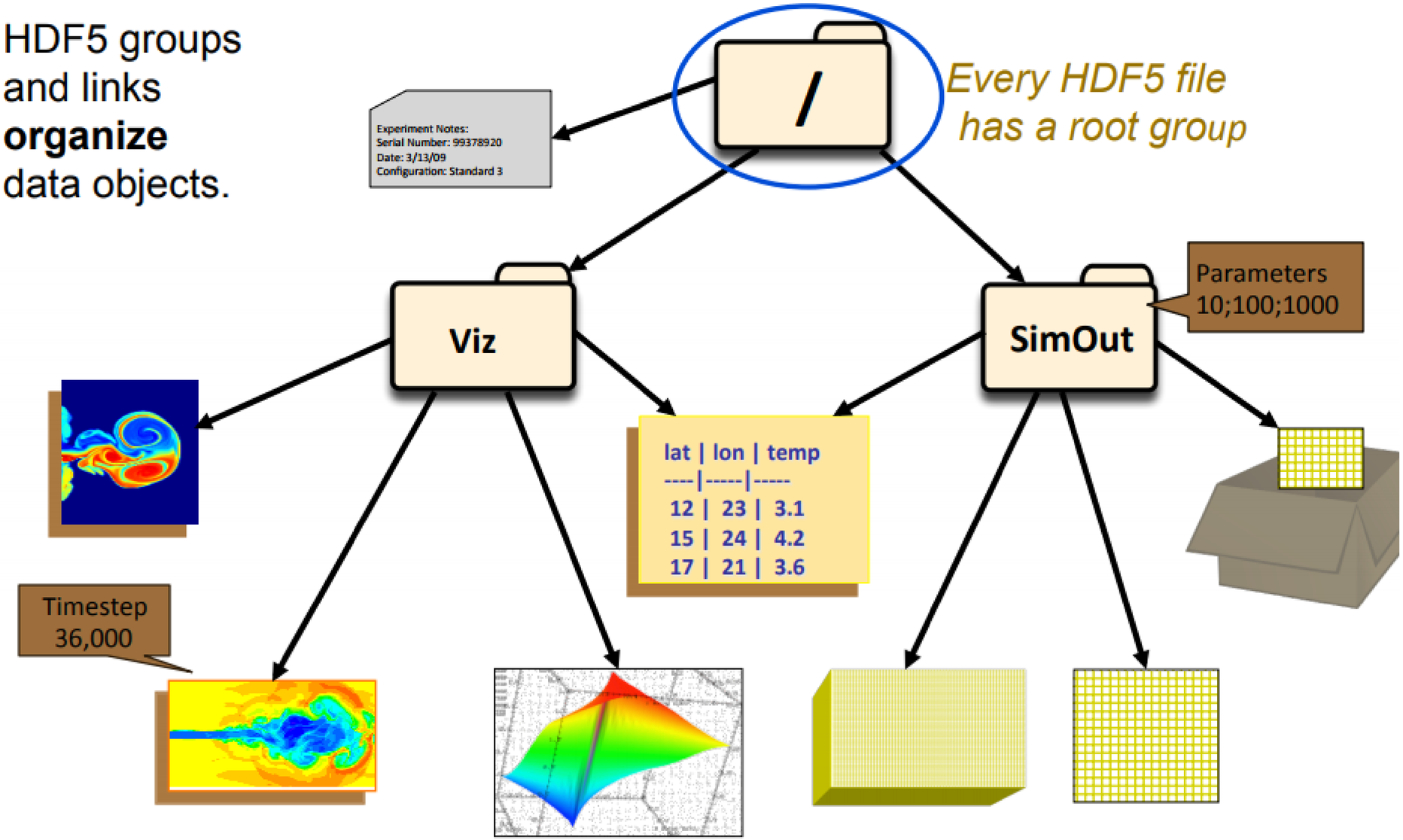
- folder
- POSIX path

Attributes

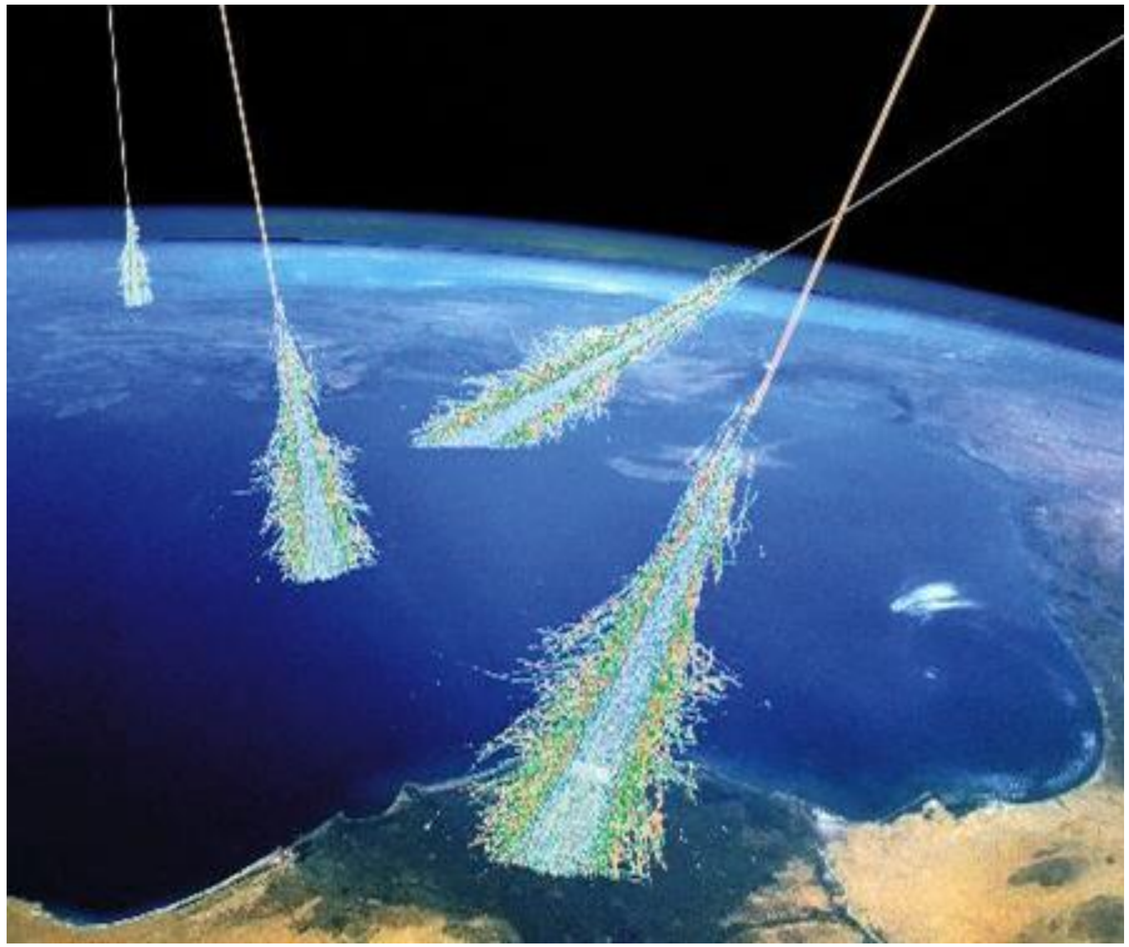
- Name
- Value

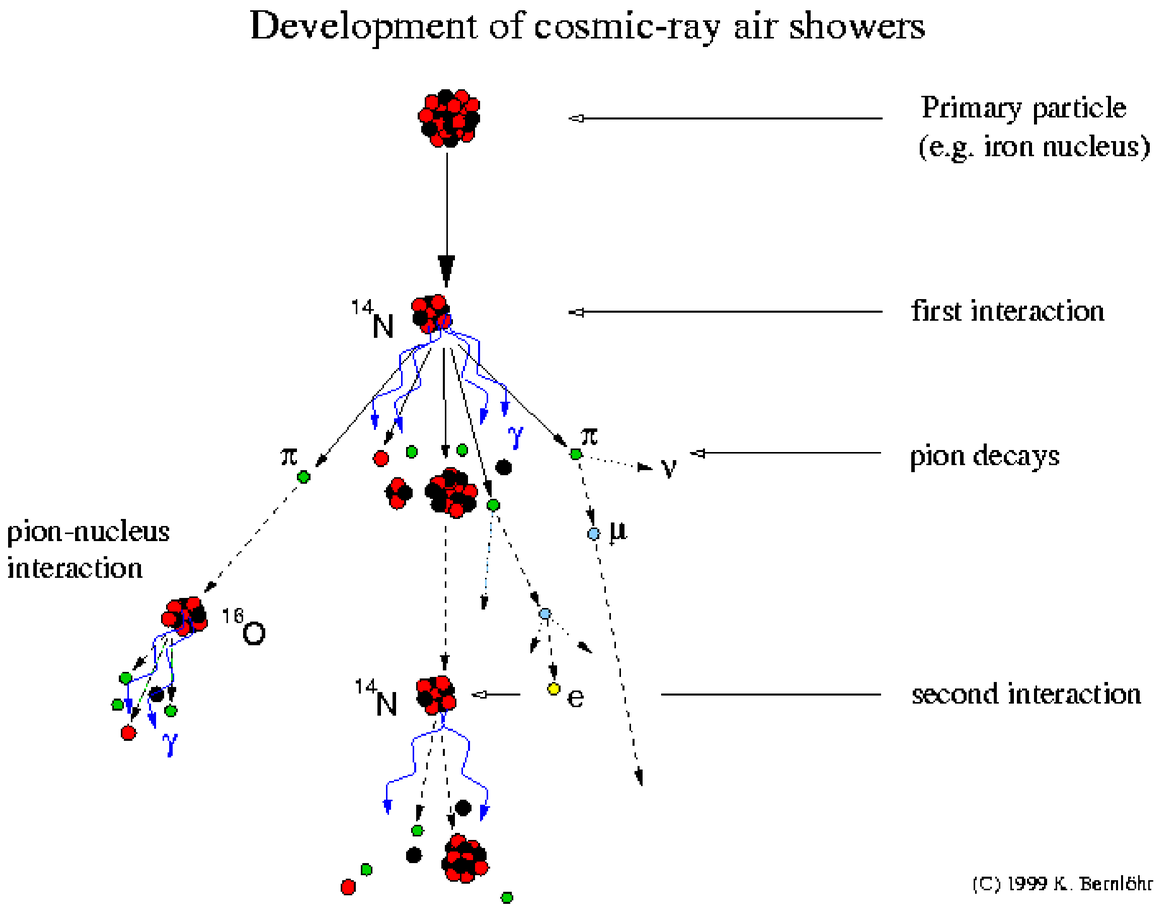
HDF5 hierarchy

HDF5 groups and links **organize** data objects.



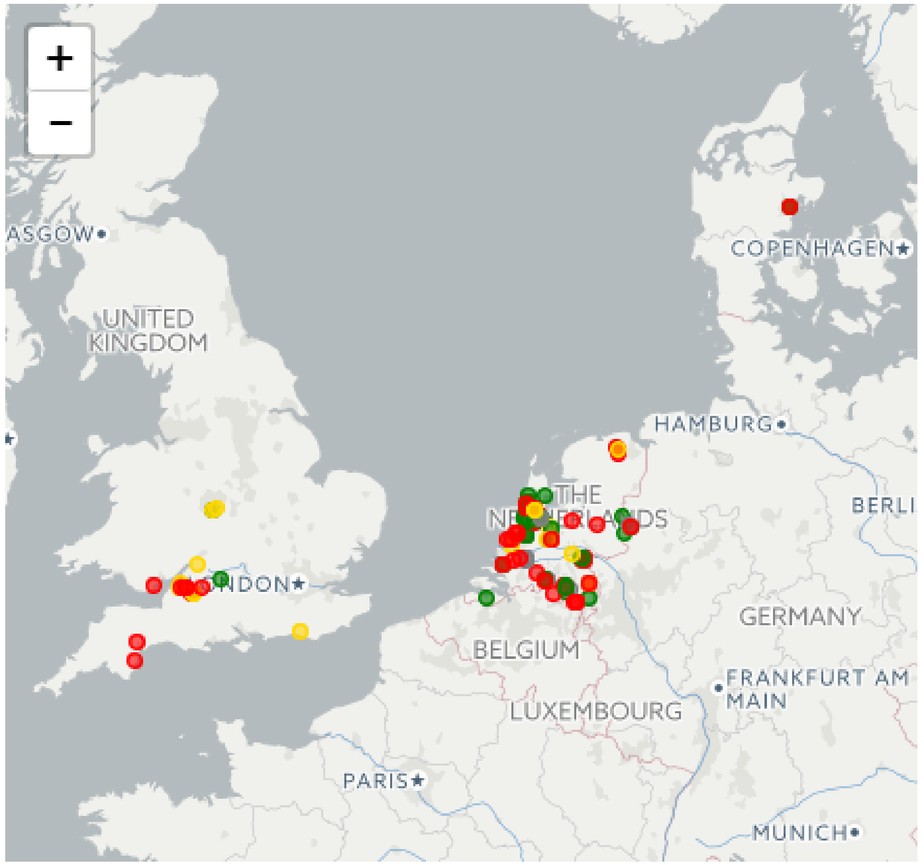
Demo HiSPARC daily raw dataset



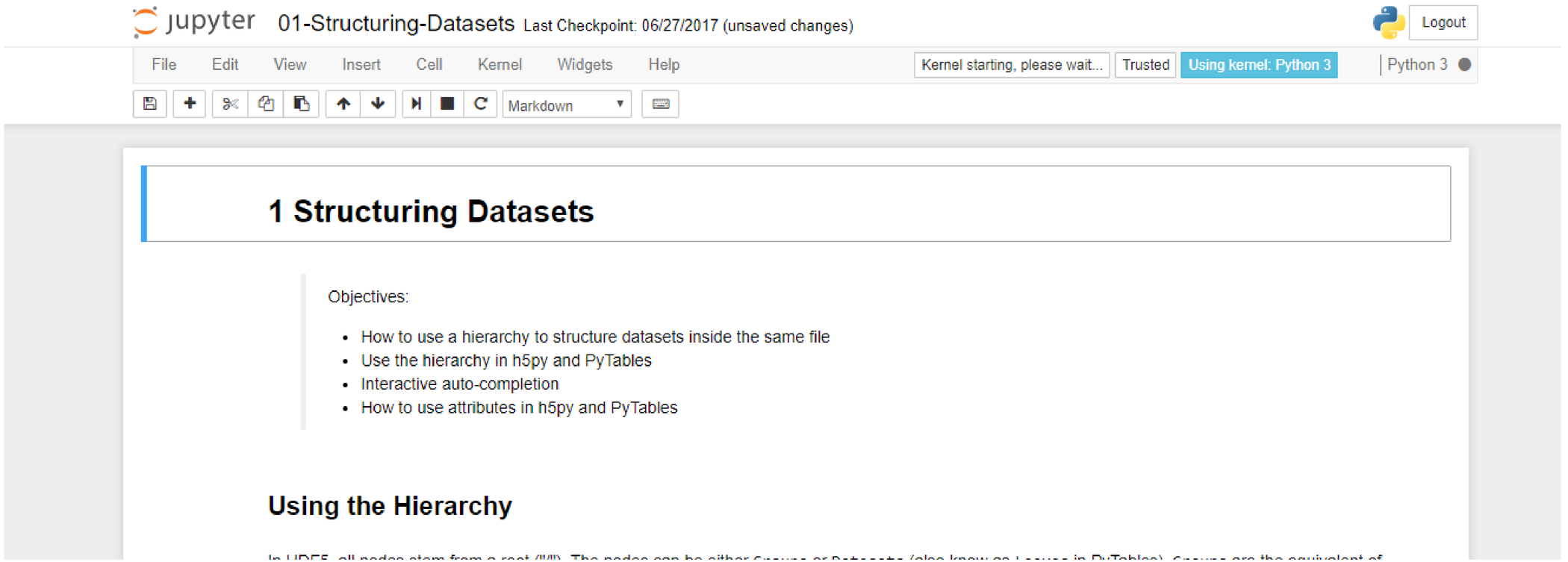


Demo HiSPARC daily raw dataset

- **Cosmic Ray array on rooftops of high schools:**



Notebooks 1 and 2



Part 2

Chunking and compression

HDF5 is C-library

- **HDF5 and numpy are C-libraries.**
- **row major order: A[row][column]**
- Data is stored in row major order:
  rows are fast, columns are slow.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

Row-major order, e.g., C (0-indexed)

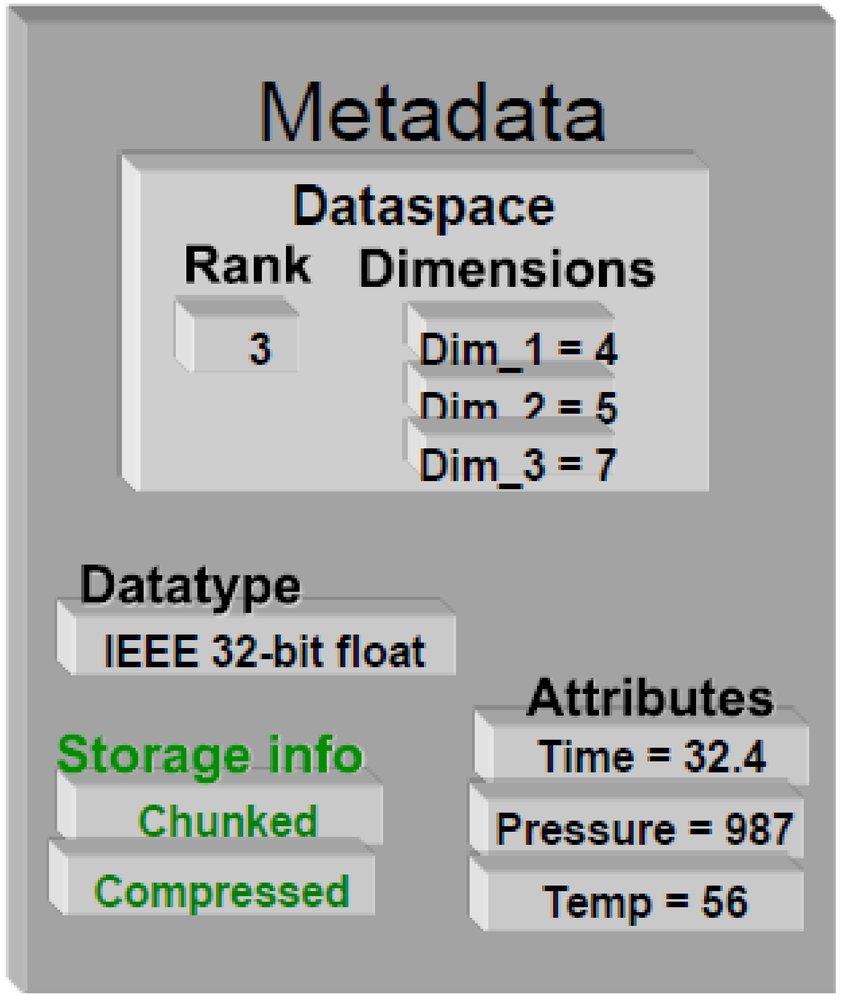
Address	Access	Value
0	`A[0][0]`	a_{11}
1	`A[0][1]`	a_{12}
2	`A[0][2]`	a_{13}
3	`A[1][0]`	a_{21}
4	`A[1][1]`	a_{22}
5	`A[1][2]`	a_{23}

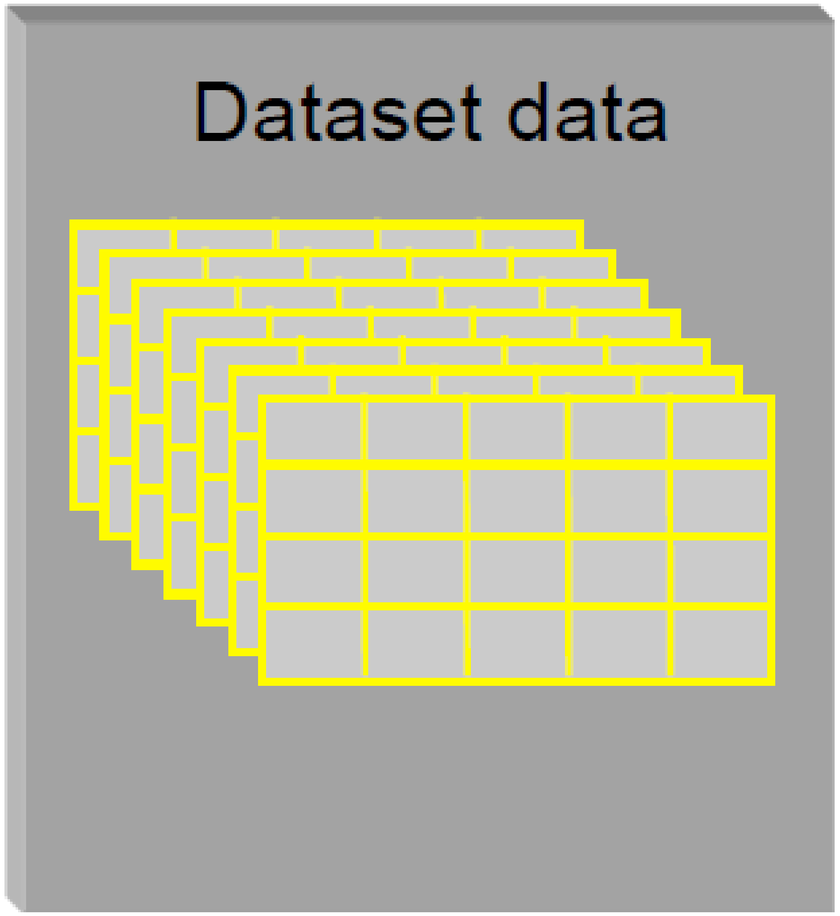
Column-major order, e.g., Fortran (1-indexed)

Address	Access	Value
1	`A(1,1)`	a_{11}
2	`A(2,1)`	a_{21}
3	`A(1,2)`	a_{12}
4	`A(2,2)`	a_{22}
5	`A(1,3)`	a_{13}
6	`A(2,3)`	a_{23}

https://en.wikipedia.org/wiki/Row-_and_column-major_order

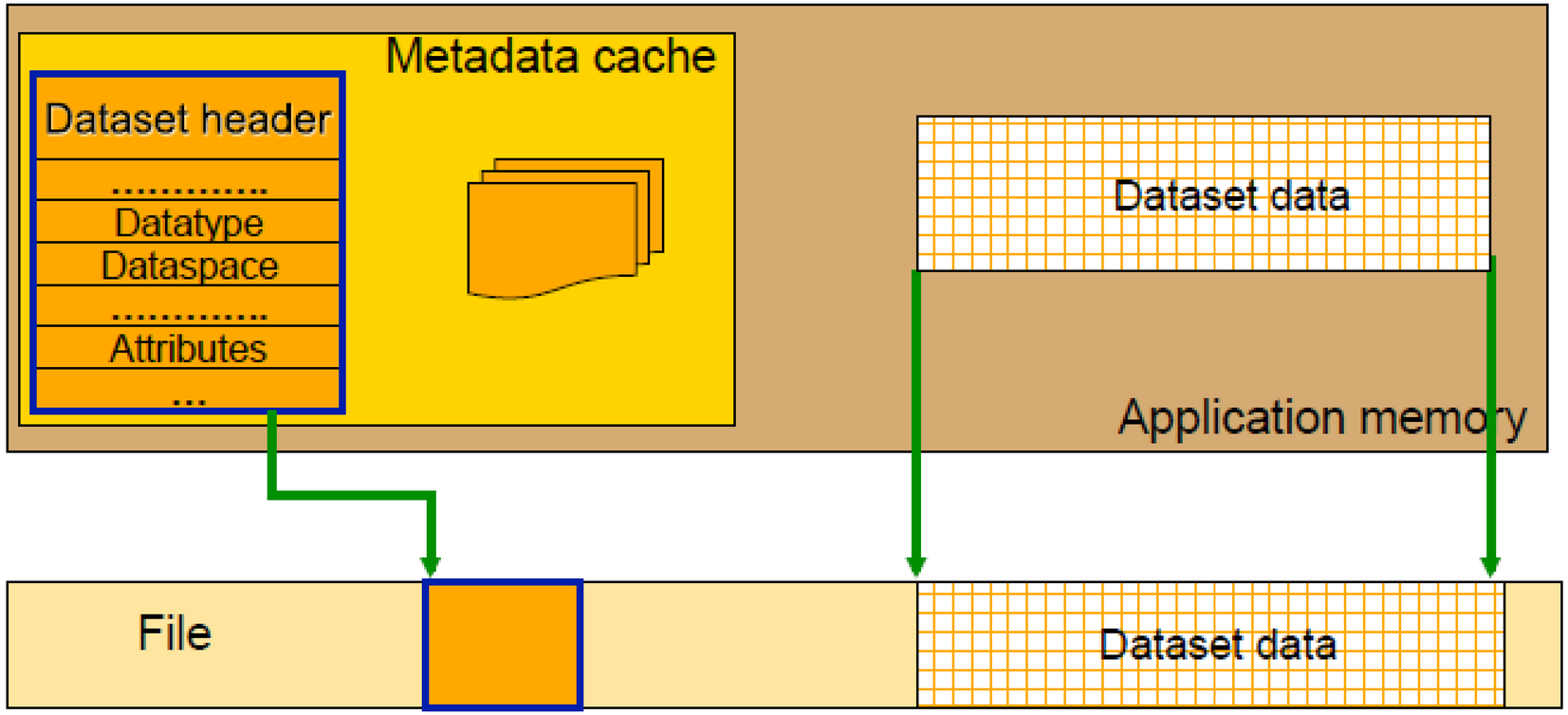
HDF5 Dataset





HDF5 File layout

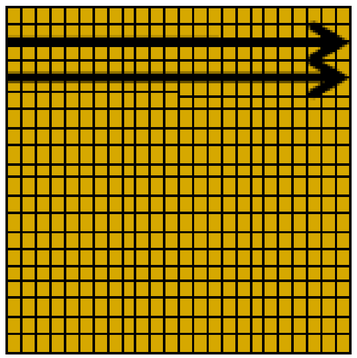
- Metadata header separate from dataset data
- Data stored in one contiguous block in HDF5



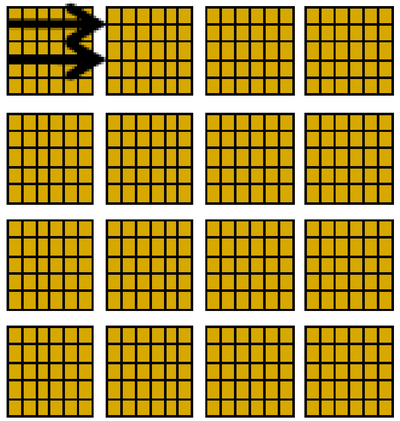
What is chunking?

- Data is stored in chunks of predefined size
- Two-dimensional instance may be referred to as data tiling
- HDF5 library always writes/reads the whole chunk

Contiguous

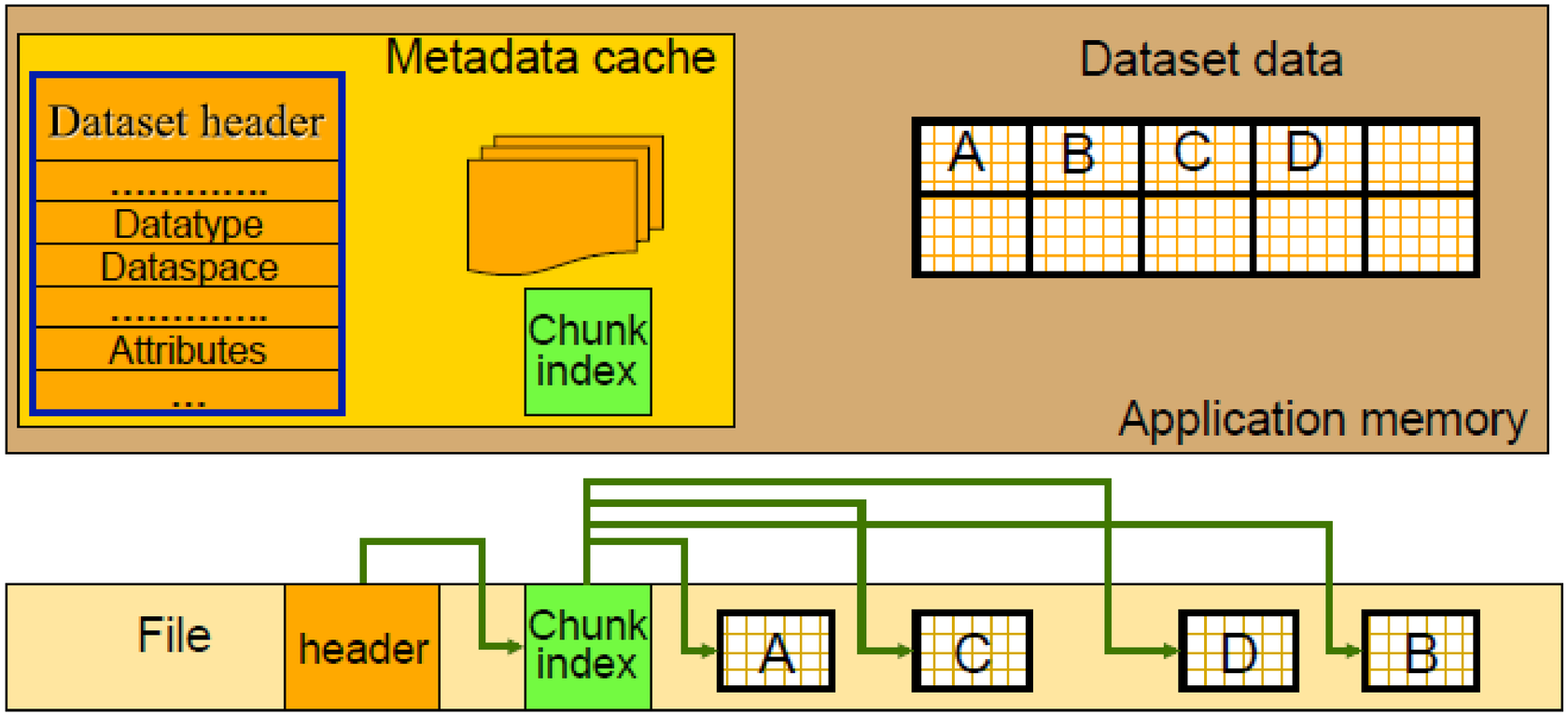


Chunked



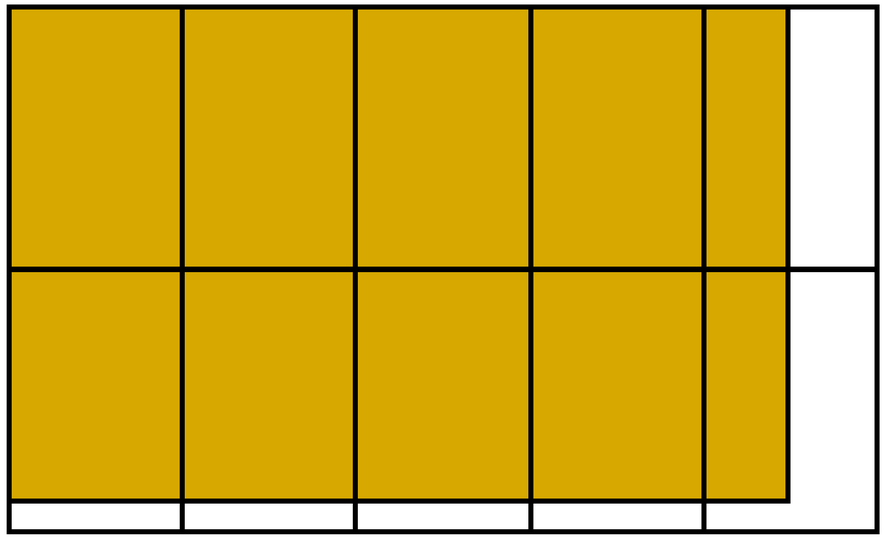
HDF5 File layout: Chunked.

- Dataset data is divided into equally sized blocks (chunks).
- Each chunk is stored separately as a contiguous block in HDF5 file.



Things to remember:

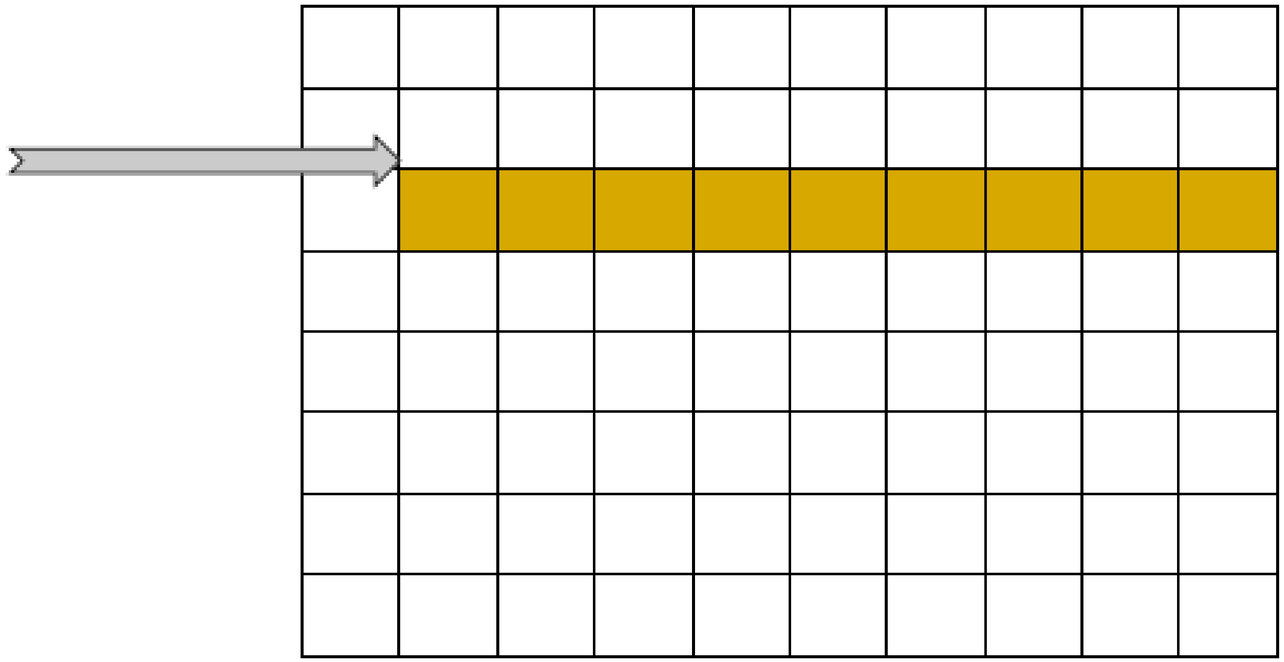
- Chunk always has the same rank as a dataset
- Chunk's dimensions do not need to be factors of dataset's dimensions
- *Caution: May cause **more** I/O than desired (see white portions of the chunks below)*



Quiz

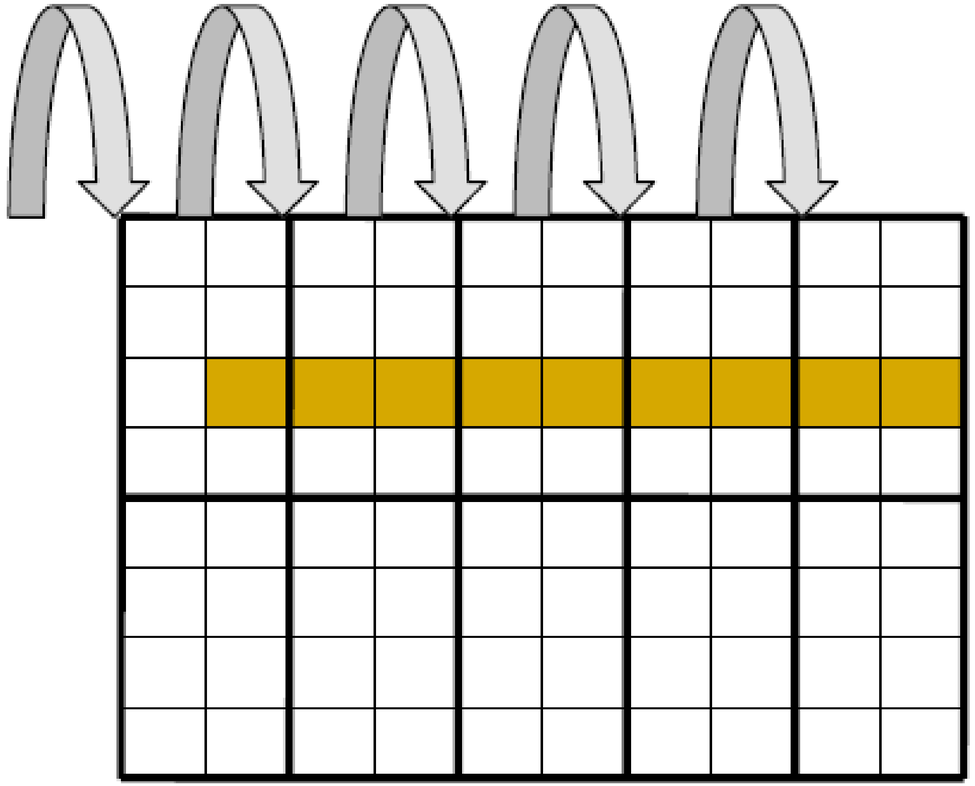
- Why shouldn't I make a chunk with dimension sizes equal to one?
- Can I change chunk size after dataset was created?

Accessing a row in a contiguous dataset



One seek is needed to find the starting location of row of data.
Data is read/written using one disk access.

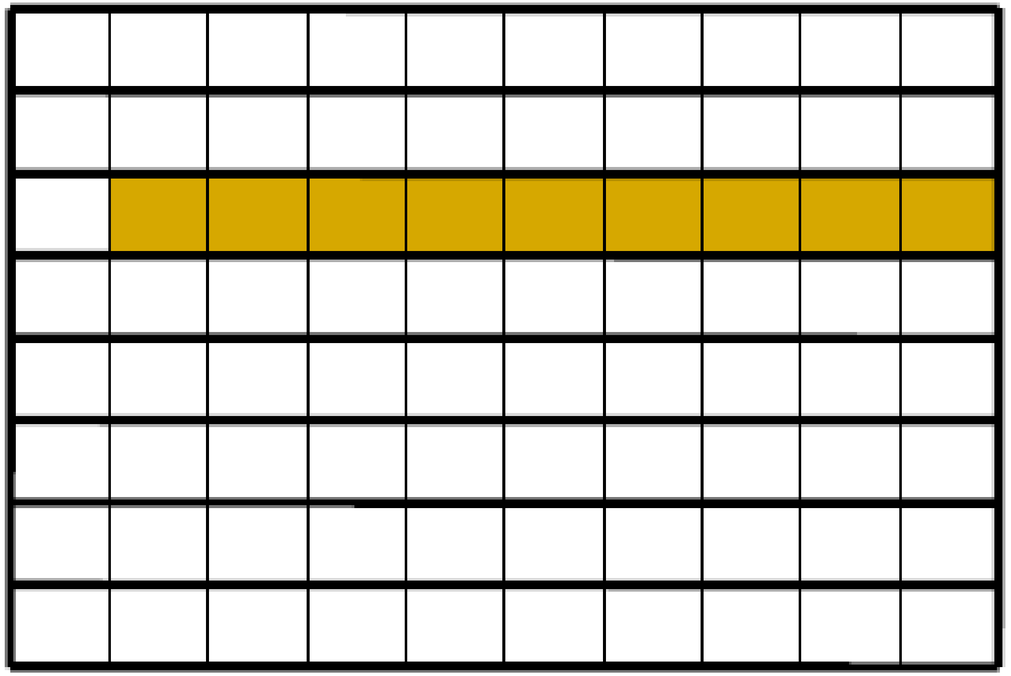
Accessing a row in a chunked dataset



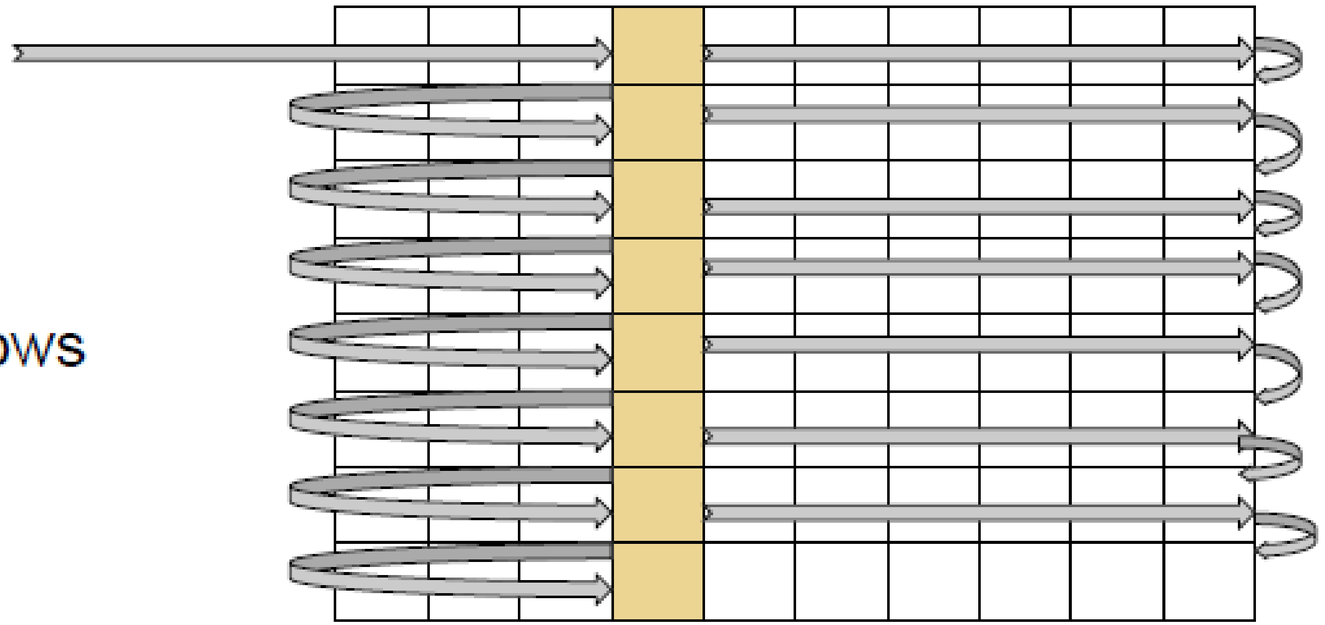
Five seeks is needed to find each chunk. Data is read/written using five disk accesses. Chunking storage is less efficient than contiguous storage.

Quiz

- How might I improve this situation, if it is common to access my data in this way?

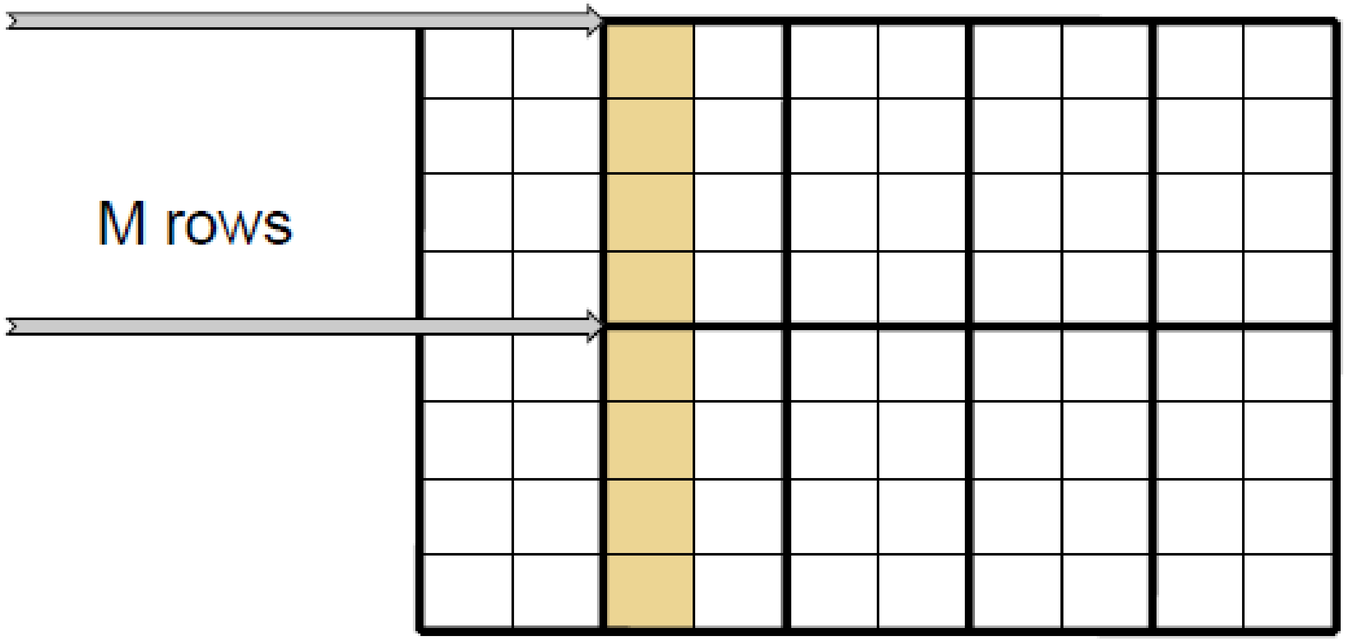


Columnar access of a contiguous dataset



M seeks are needed to find the starting location of the element. Data is read/written using M disk accesses. Performance may be very bad.

Motivation for using chunked storage



Two seeks are needed to find two chunks. Data is read/ written using two disk accesses. For this pattern chunking helps with I/O performance.

Notebook 3: Chunking in HDF5

jupyter 03-Chunking Last Checkpoint: 06/27/2017 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3

Markdown

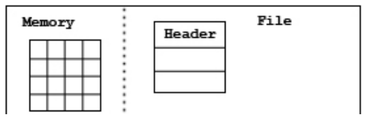
3-Chunking in HDF5

Objectives:

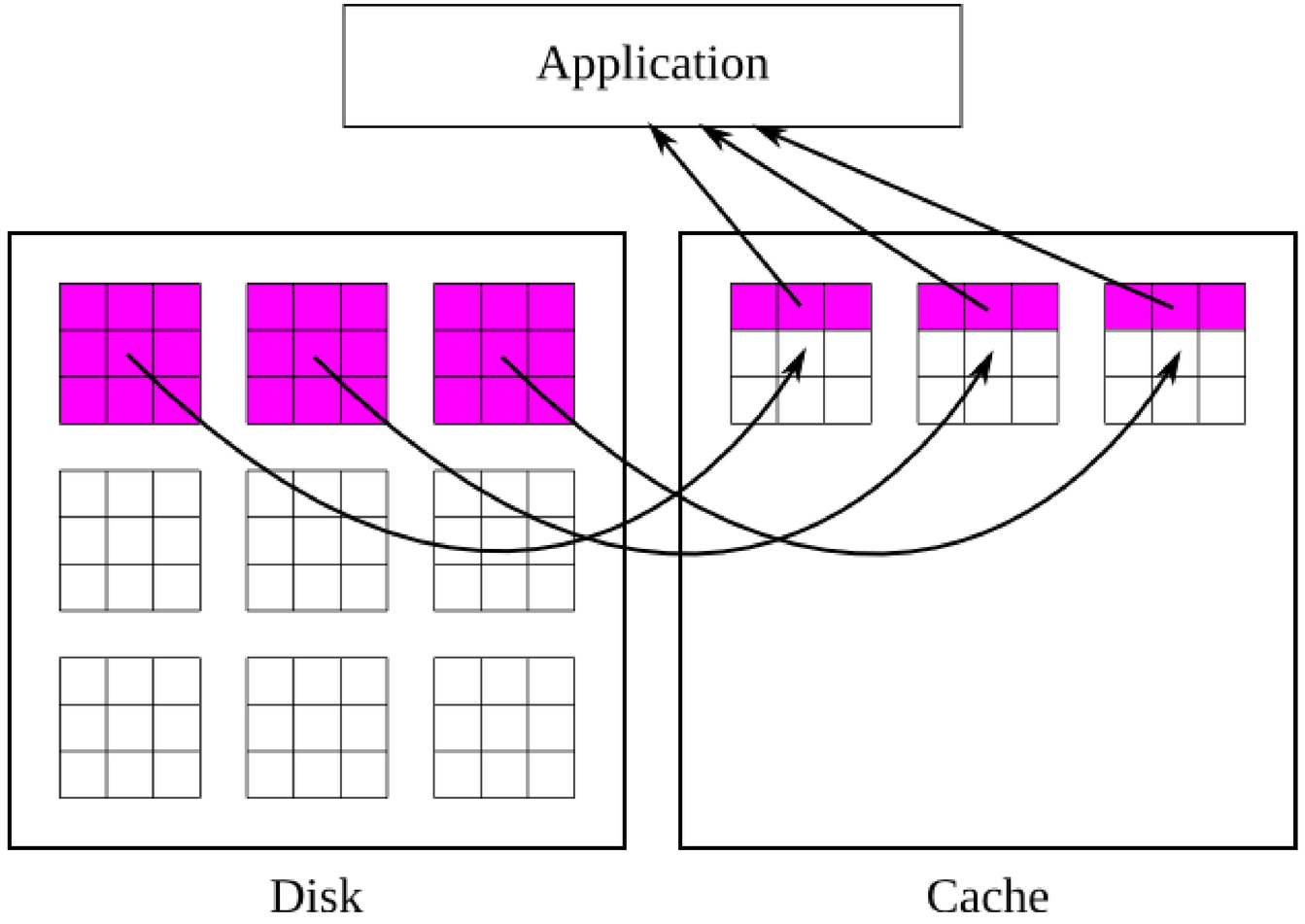
- Explain the concept of data chunking
- Show how to create and read datasets that are chunked
- Learn how to choose reasonable chunk sizes for your datasets

The HDF5 library supports several layouts so as to store datasets.

- Continuous layout:

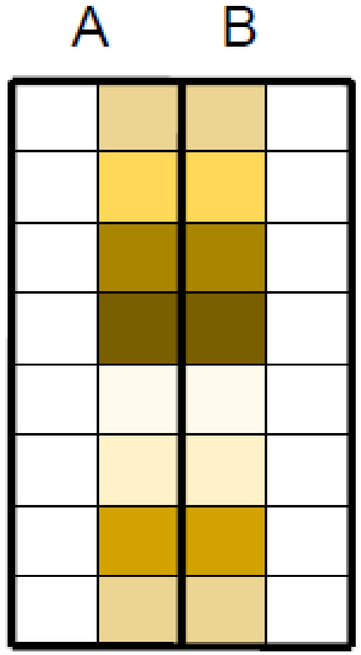


Mind the chunk cache!



Mind the chunk cache!

M rows
Each row is read by a separate call to H5Dread



The Good: If both chunks fit into cache, 2 disks accesses are needed to read the data.
The Ugly: If one chunk fits into cache, 2M disks accesses are needed to read the data (compare with M accesses for contiguous storage).

Pitfalls

- ***Chunks are too small***: Large overhead for looking up chunks (B-tree)
- ***Chunks are too large***: Large performance penalty. Keep in mind that chunks are atomic: To read (write) a single item the entire chunk needs to be read (written).
- ***Chunk cache too small***: All chunks in an I/O operation (selection) should fit in the chunkcache.

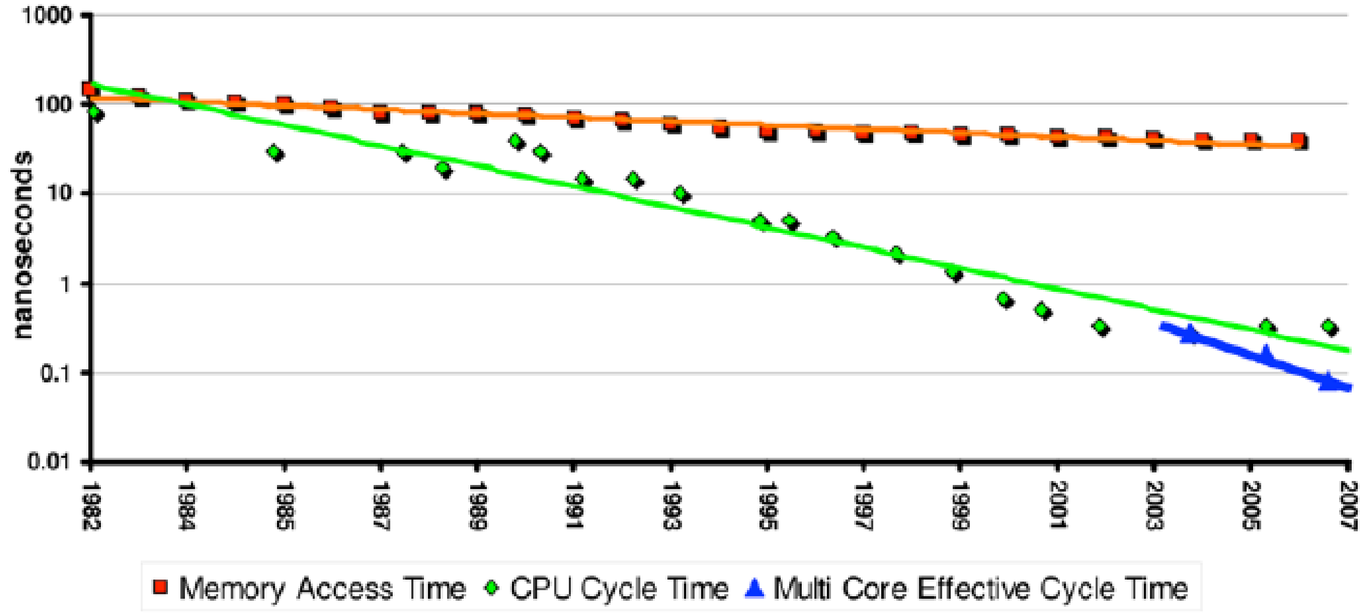
Part 2b

Compression

The starving CPU problem

The starving CPU problem

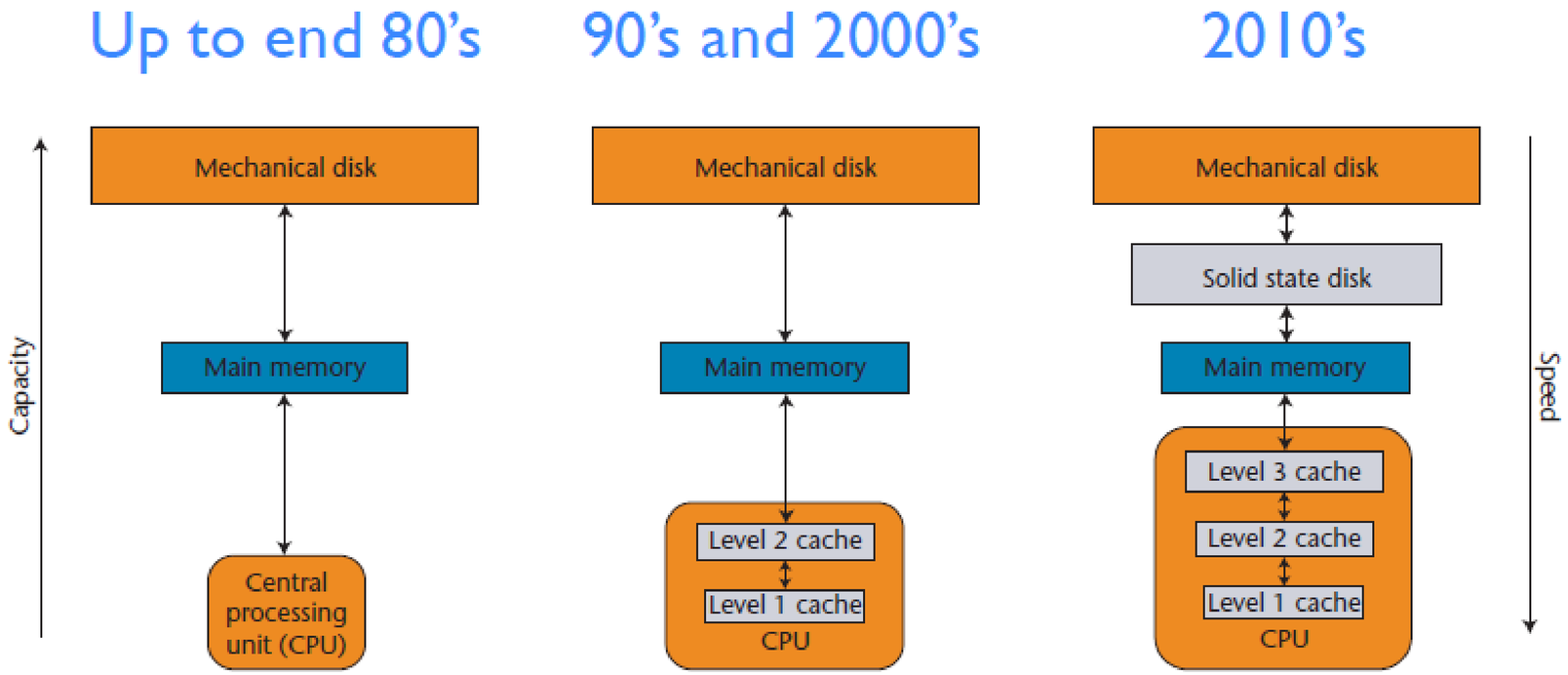
- **Memory access time vs CPU cycle time (1992-2007):**



- **Current CPU are bored: Waiting for data.**

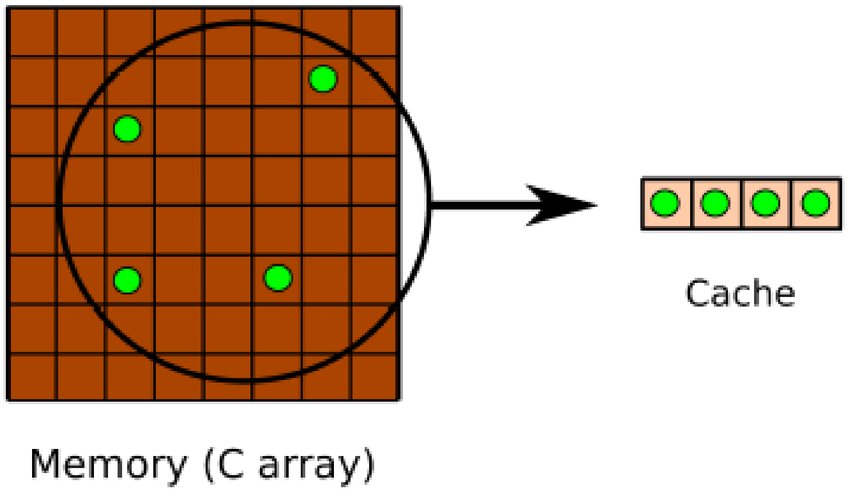
CPU Starvation in 2017

- Memory latency is much slower (between 100x and 250x) than processors.
- Memory bandwidth is improving at a better rate than memory latency, but it is also lagging behind processors (between30x and 100x).

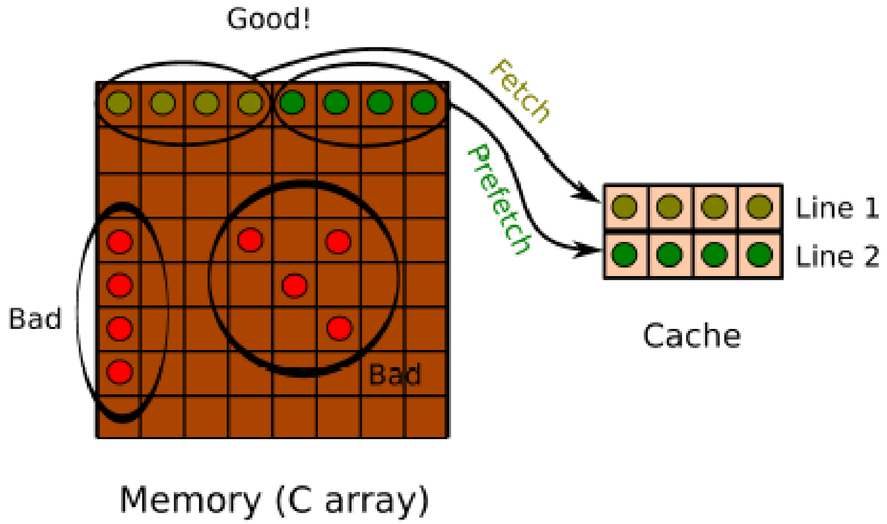


When are caches effective?

- Time locatily: part of dataset is reused

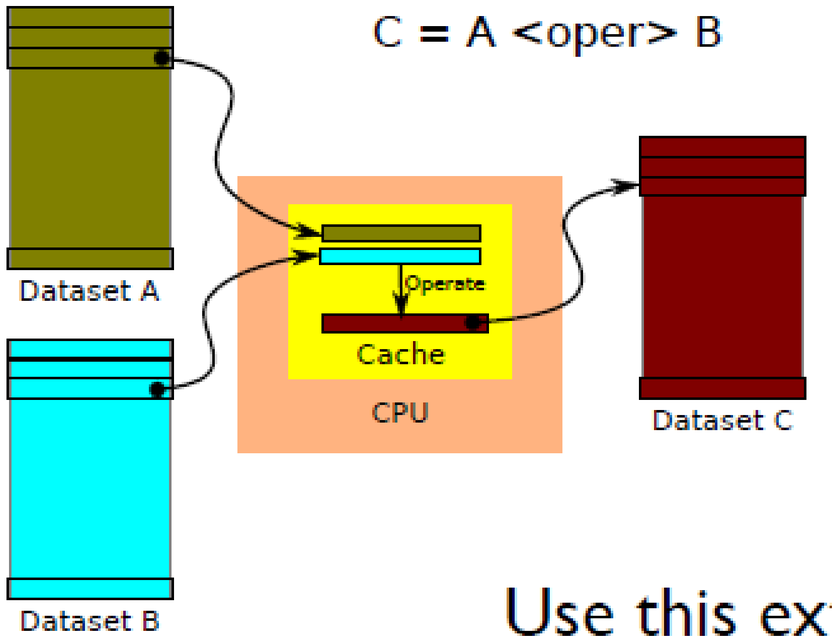


- Spatial locatily: dataset is access sequentially.



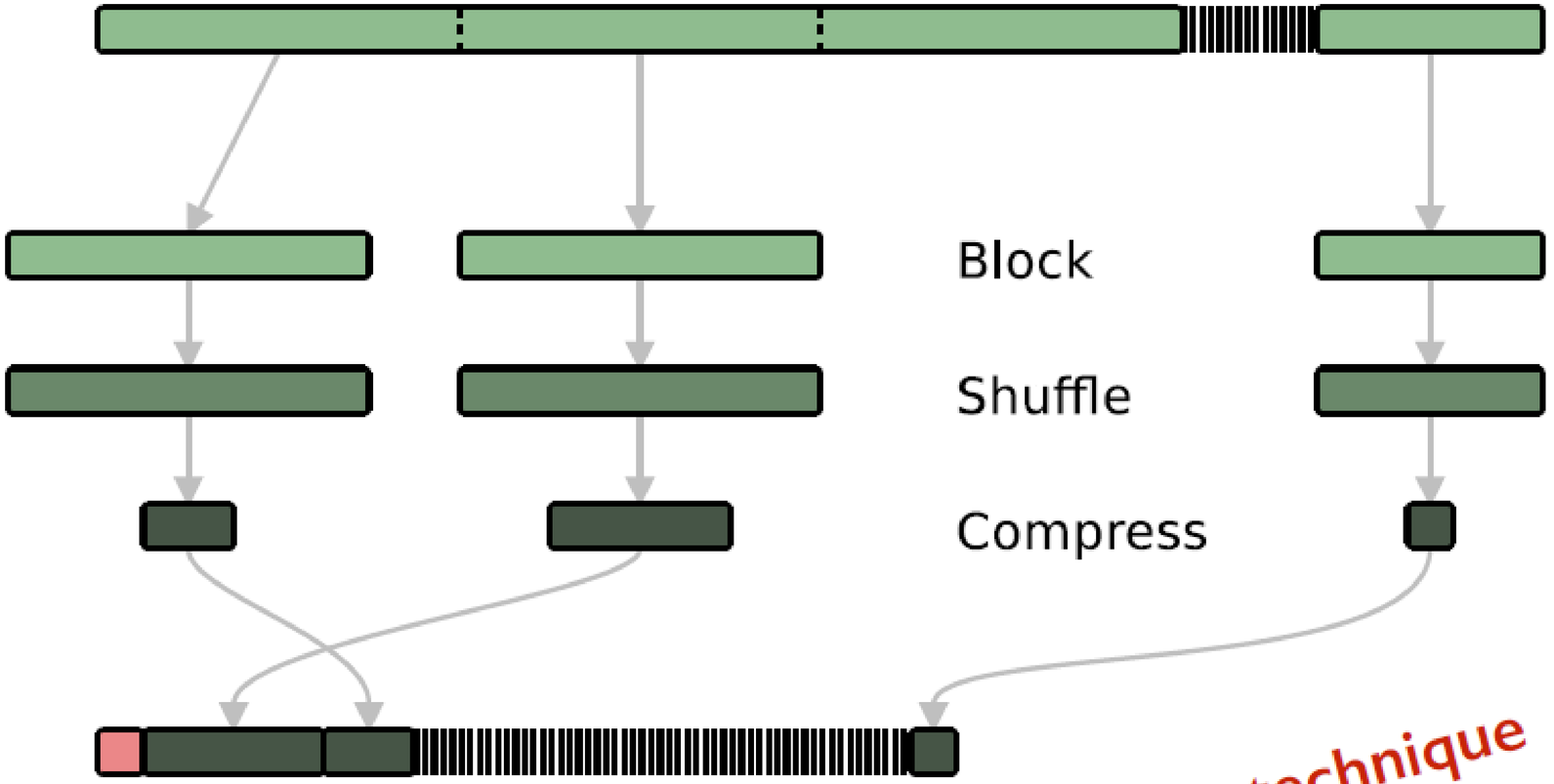
Blocking

When accessing disk or memory, get a contiguous block that fits in CPU cache, operate upon it and reuse it as much as possible.



Use this extensively to leverage spatial and temporal localities

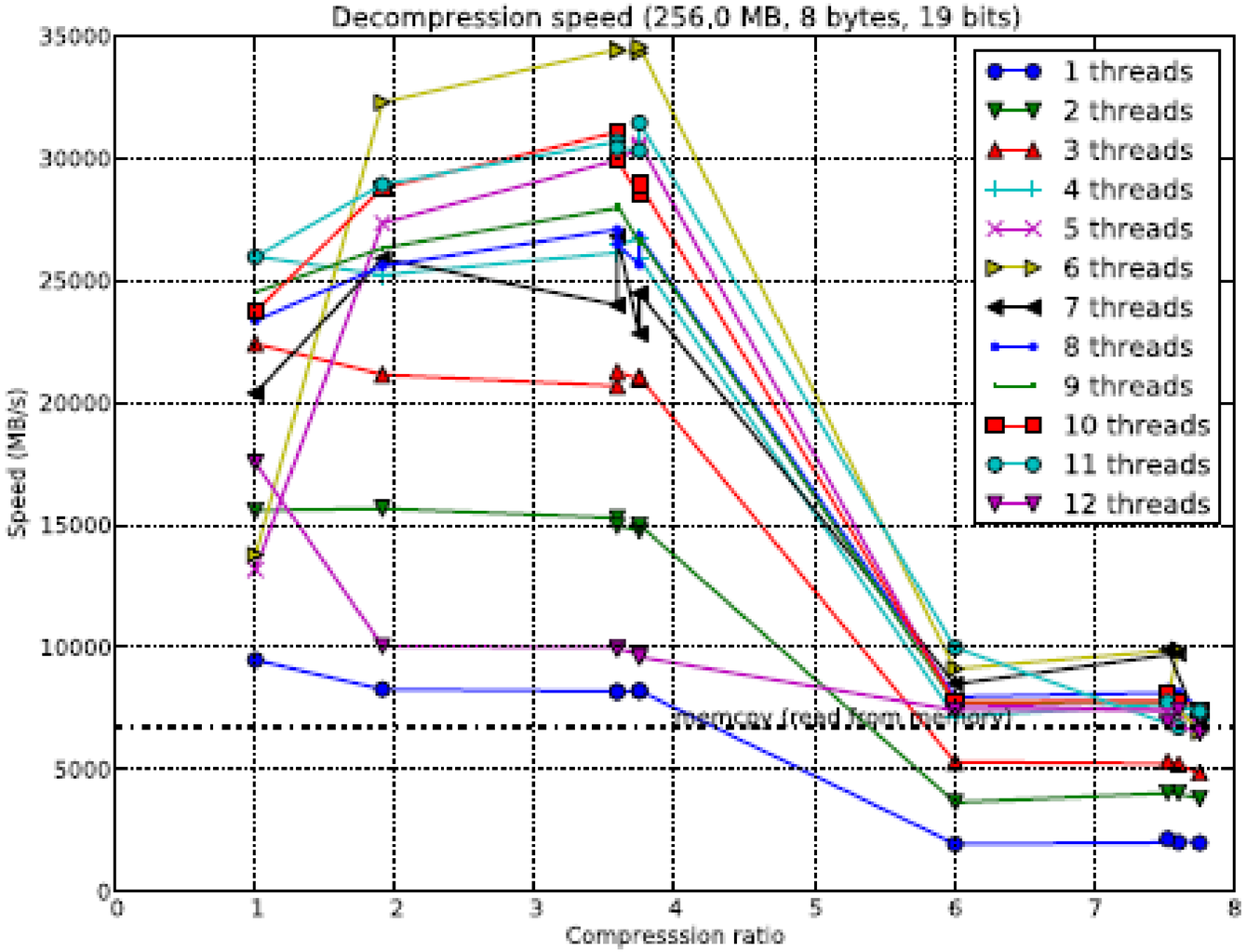
How blosc works



Attr: Valentin Haenel

Blocking technique at work!

Faster than memcpy(): low overhead



Compression in HDF5: filters.

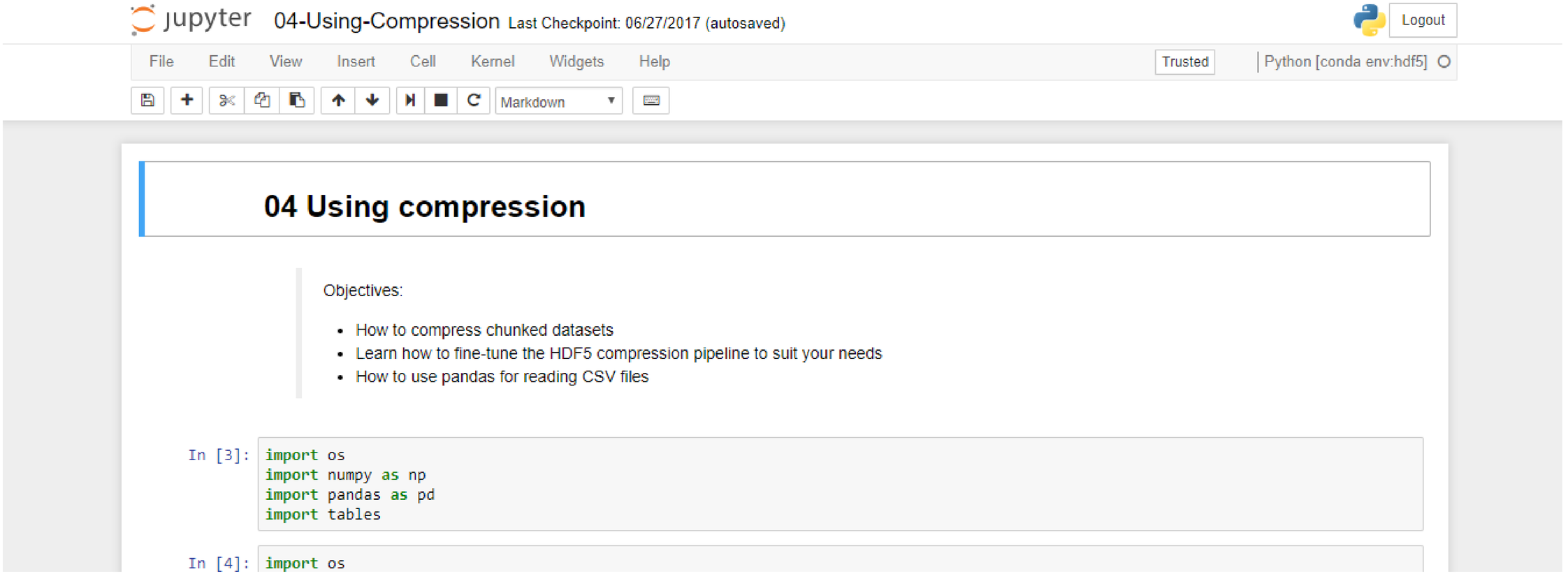
- **Compression in HDF5 is handled by the filterpipeline.**
- **In PyTables:**

```
import tables

filters = tables.Filters(complevel=..., complib=...)

dset = tables.create_table(..., filters=filters, ...)
```

Available compressors in PyTables

- **zlib** `complib=zlib`
- **bzip2** `complib=bzip2`
- **(lzo)**
- **BLOSC:**
  - blosclz `complib=blosc`
  - zlib `complib=blosc:zlib`
  - lz4 `blosc:lz4`
  - lz4hc `blosc:lz4hc`
  - zstd `blosc:zstd`
  - snappy `blosc:snappy`

Notebook 4: Using Compression



Part 3

Queries

Part 4

Integration with Pandas: HDFStore

Part 5

Low level API and Parallel HDF5