Working with on-disk data formats

- Advanced topics in single-cell transcriptomics

Mike Smith

@grimbough

Why care about how data are stored on disk?

- Understanding software options
- Exploring files of uncertain origin
- Sometimes you don’t necessarily know you’re using on-disk data
- If you develop methods how data are stored can hugely impact performance

On-disk data can be use transparently in Bioconductor

Rhdf5lib

rhdf5

HDF5Array

SingleCellExperiment

zinbwave

scater

scran

scone

scPipe

DropletUtils

...

C / C++ Library → R Interface → Counts Matrix → Complete SC Dataset → Analysis Tools

Representing count matrices

It’s all about the count matrix

- The sparse nature affects almost everything
  - Biological interpretation - biologically-true zero expression vs technical artefact
  - Statistical analysis
  - Software we manipulate the data with
  - How we can store and transfer

Dense matrices

Matrix

10	3	0	3	0
0	9	7	0	8
0	0	8	8	0
0	0	7	7	9
2	0	0	5	13

Dense matrices

Matrix

10	3	0	3	0
0	9	7	0	8
0	0	8	8	0
0	0	7	7	9
2	0	0	5	13

Memory requirements scale linearly

- 1 Column ~ 120KB (4 bytes * 30,000 rows)
- 100 Columns ~ 12MB
- 10,000 Columns ~ 1.2GB
- 1,000,000 Columns ~ 120GB

Sparse matrices

Matrix

10	3	0	3	0
0	9	7	0	8
0	0	8	8	0
0	0	7	7	9
2	0	0	5	13

Values

10	2	3	9	7	8	7	3	8	7	5	8	9	13

Row indices

0	4	0	1	1	2	3	0	2	3	4	1	3	4

Column indices

0	0	1	1	2	2	2	3	3	3	3	4	4	4

Sparse matrices

Matrix

10	3	0	3	0
0	9	7	0	8
0	0	8	8	0
0	0	7	7	9
2	0	0	5	13

Values

10	2	3	9	7	8	7	3	8	7	5	8	9	13

Row indices

0	4	0	1	1	2	3	0	2	3	4	1	3	4

Column pointers

0	2	4	7	11	14

Space benefit proportional to level of sparsity

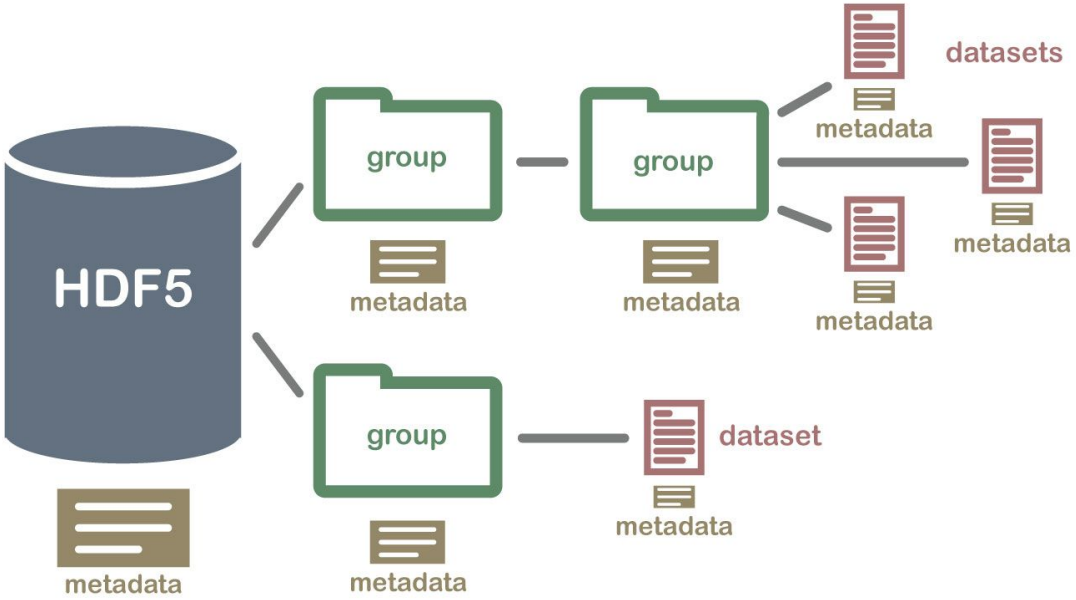
- Worst case - 3 times the storage
- Best case - equivalent to 1 row

What about on-disk?

- At some point you want to save or share these counts
- Some data are too big to ever be read into memory
- These choices of representations also apply on-disk and there’s no consensus!
  - MEX files from 10X use the sparse format in a text file
  - H5 files from 10X use the sparse format in a HDF5 file
  - EDS (Efficient Data Storage) produced by Alevin is a binary sparse format
  - Loom files are HDF5 and mandate a dense matrix
  - SingleCellExperiment in Bioconductor can use an HDF5 file contain dense or sparse matrices

Introducing HDF5

‘Self describing’ hierarchical data format



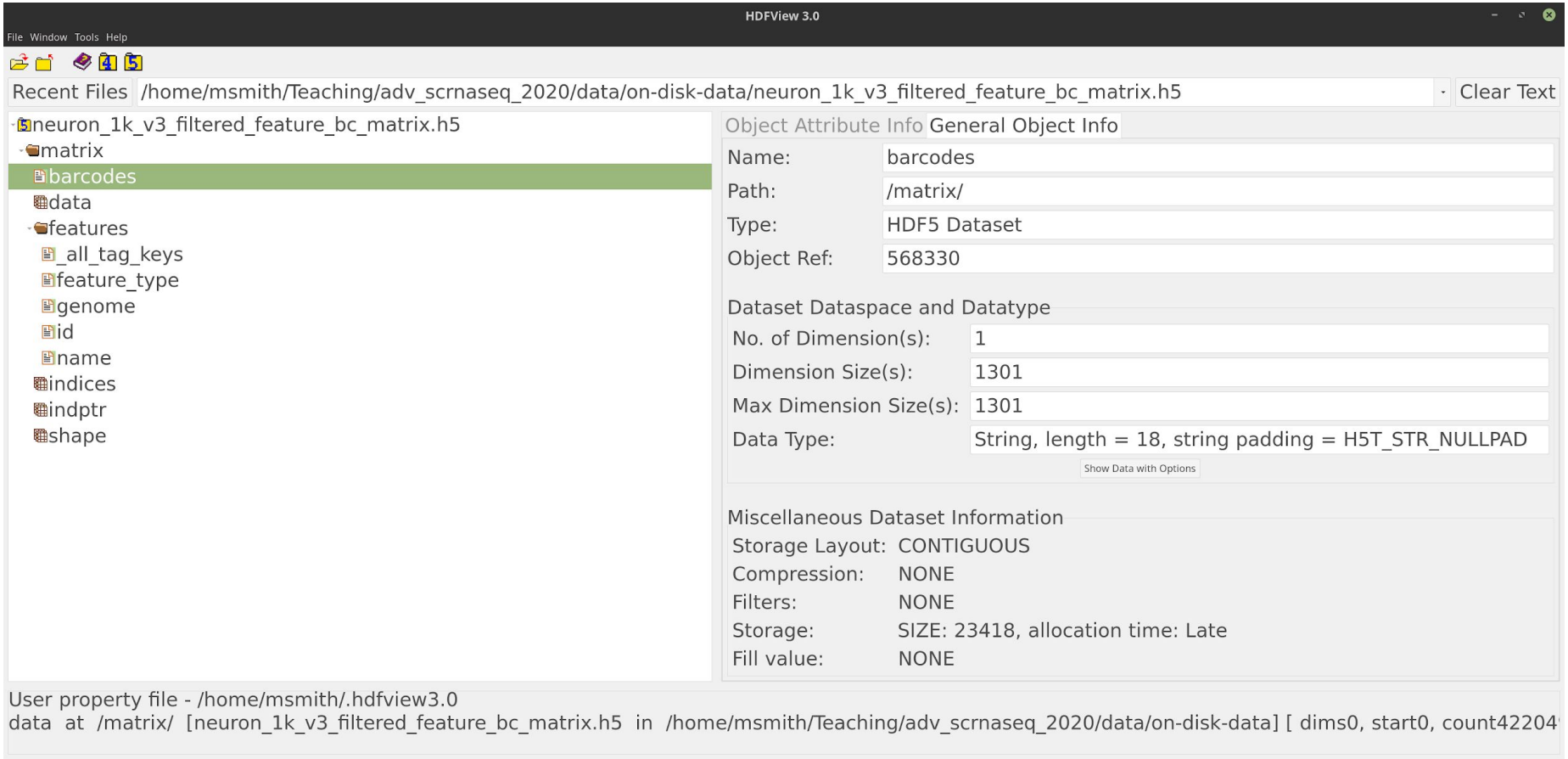
http://neondataskills.org/HDF5/About

10X Cell Ranger comparison

```
filtered_gene_bc_matrices
└── hg19
    ├── barcodes.tsv
    ├── genes.tsv
    └── matrix.mtx
```

```
%%MatrixMarket matrix coordinate integer general
%
33694 3471 4028148
46 1 1
154 1 21
198 1 1
```

```
└── matrix [HDF5 group]
    ├── barcodes
    ├── data
    ├── indices
    ├── indptr
    ├── shape
    └── features [HDF5 group]
        ├── _all_tag_keys
        ├── feature_type
        ├── genome
        ├── id
        ├── name
        ├── pattern [Feature Barcoding only]
        ├── read [Feature Barcoding only]
        └── sequence [Feature Barcoding only]
```


HDFView 3.0
File Window Tools Help
Recent Files
/home/msmith/Teaching/adv_scrnaseq_2020/data/on-disk-data/neuron_1k_v3_filtered_feature_bc_matrix.h5
Clear Text
neuron_1k_v3_filtered_feature_bc_matrix.h5
matrix
barcodes
data
features
_all_tag_keys
feature_type
genome
id
name
indices
indptr
shape
Object Attribute Info
General Object Info
Name: barcodes
Path: /matrix/
Type: HDF5 Dataset
Object Ref: 568330
Dataset Dataspace and Datatype
No. of Dimension(s): 1
Dimension Size(s): 1301
Max Dimension Size(s): 1301
Data Type: String, length = 18, string padding = H5T_STR_NULLPAD
Show Data with Options
Miscellaneous Dataset Information
Storage Layout: CONTIGUOUS
Compression: NONE
Filters: NONE
Storage: SIZE: 23418, allocation time: Late
Fill value: NONE
User property file - /home/msmith/.hdfview3.0
data at /matrix/ [neuron_1k_v3_filtered_feature_bc_matrix.h5 in /home/msmith/Teaching/adv_scrnaseq_2020/data/on-disk-data] [dims0, start0, count42204

Flexibility of HDF5 (and similar formats)

Other useful features

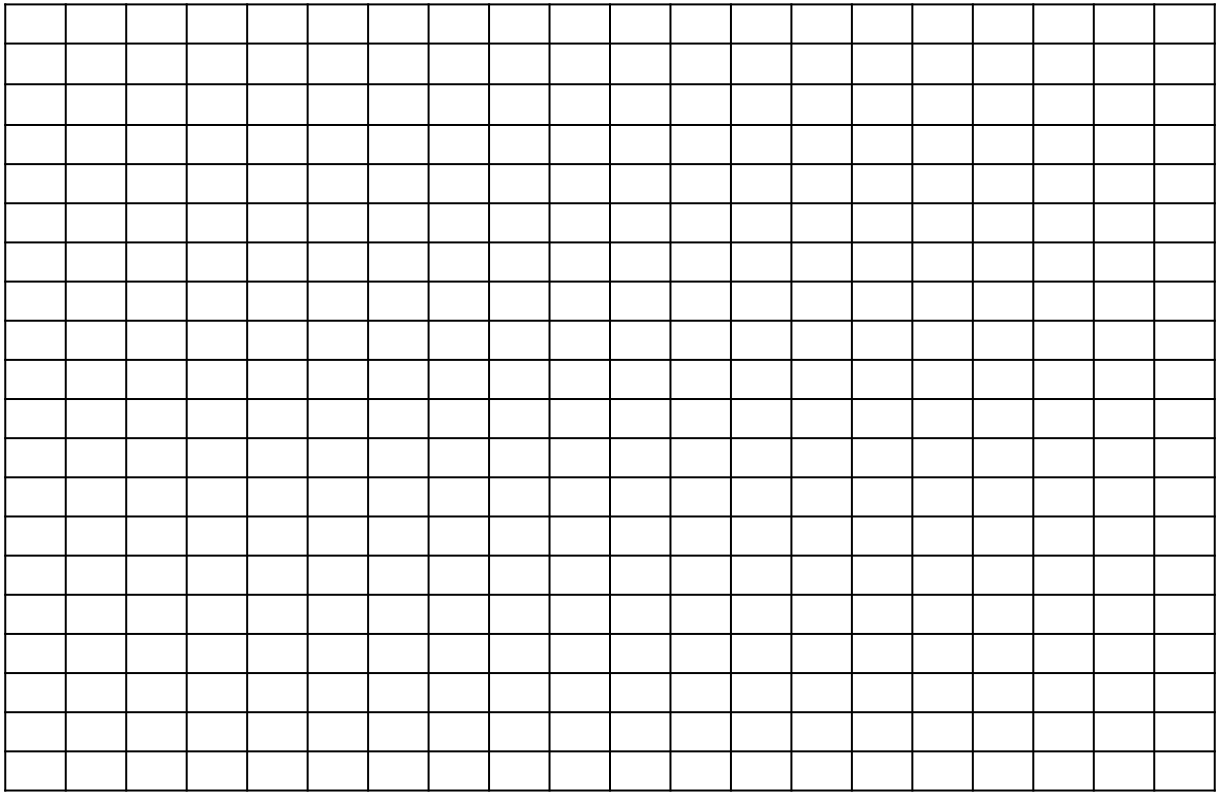
- Efficient access & subsetting
- Data compression
- Storage of heterogeneous data
- Platform independent
- Interfaces in many languages: C, C++, Java, Python, R, ...

Major drawback - these are complex libraries and almost always require installation of additional system software

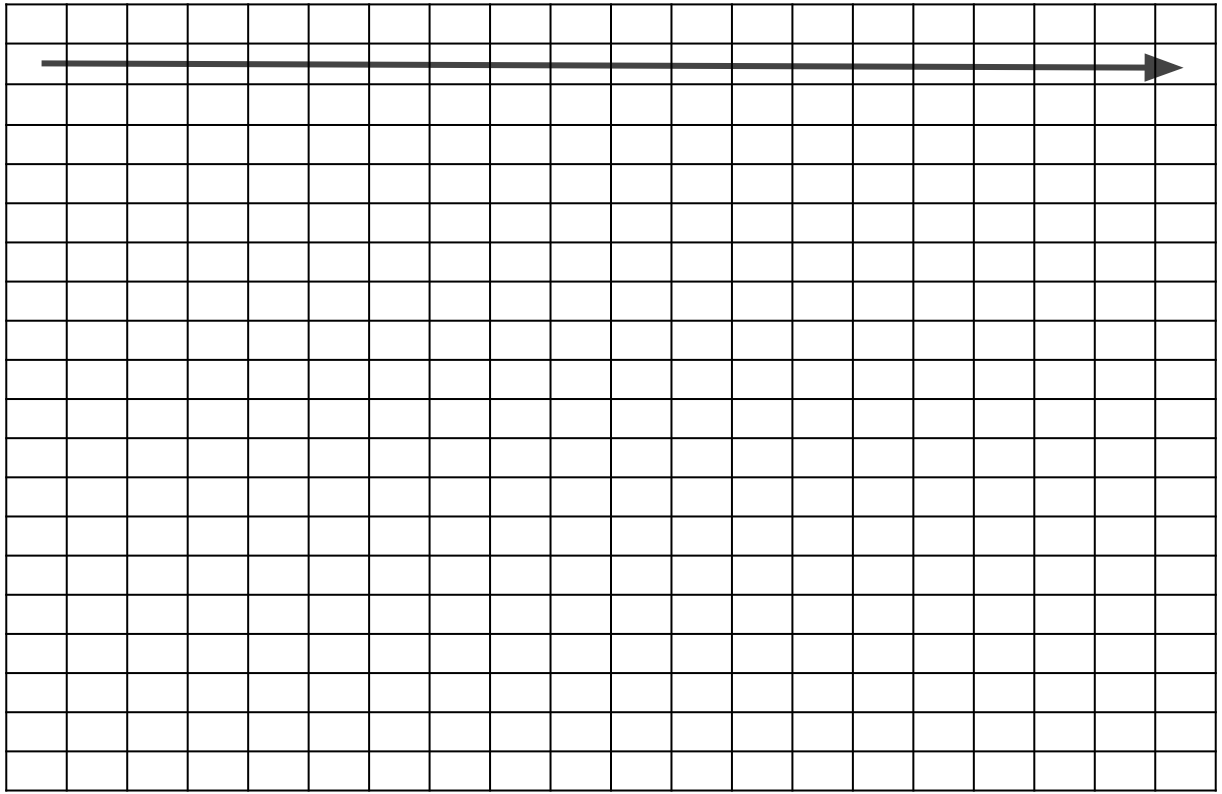
Consider a single dataset

2D dense count matrix

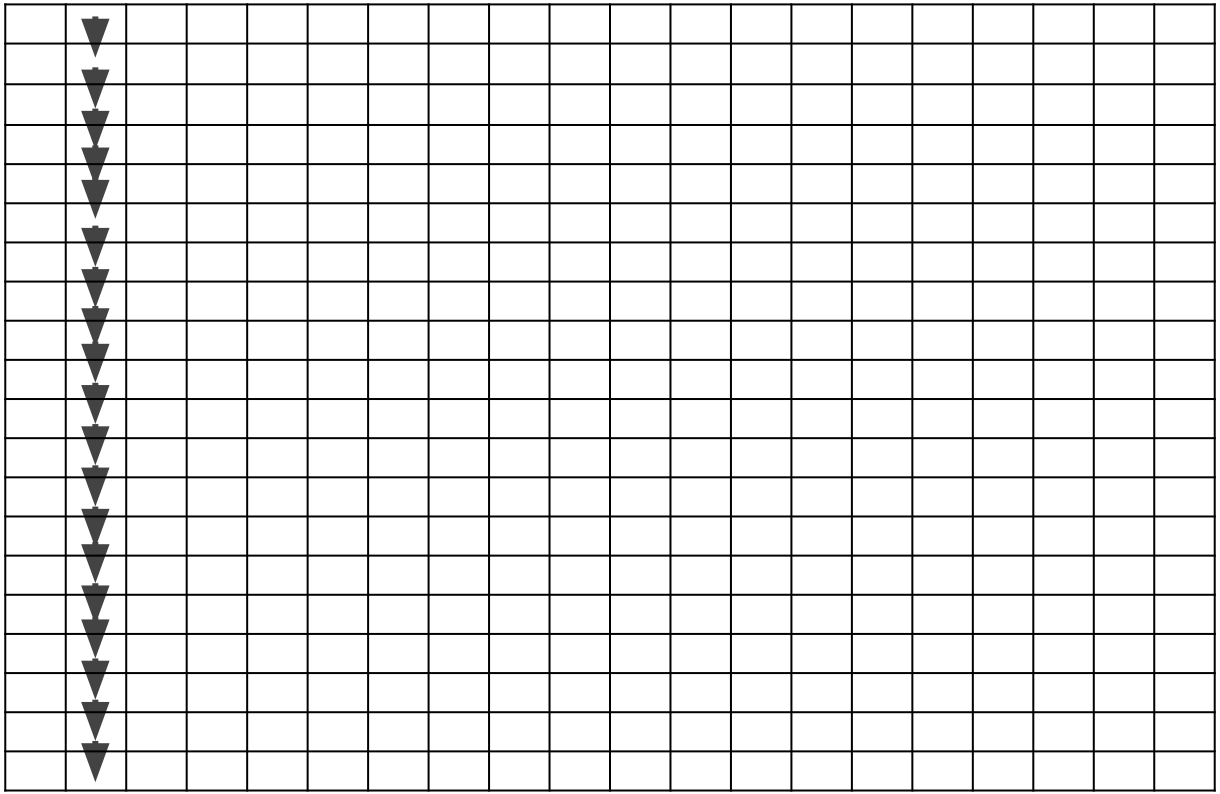
Contiguous layout



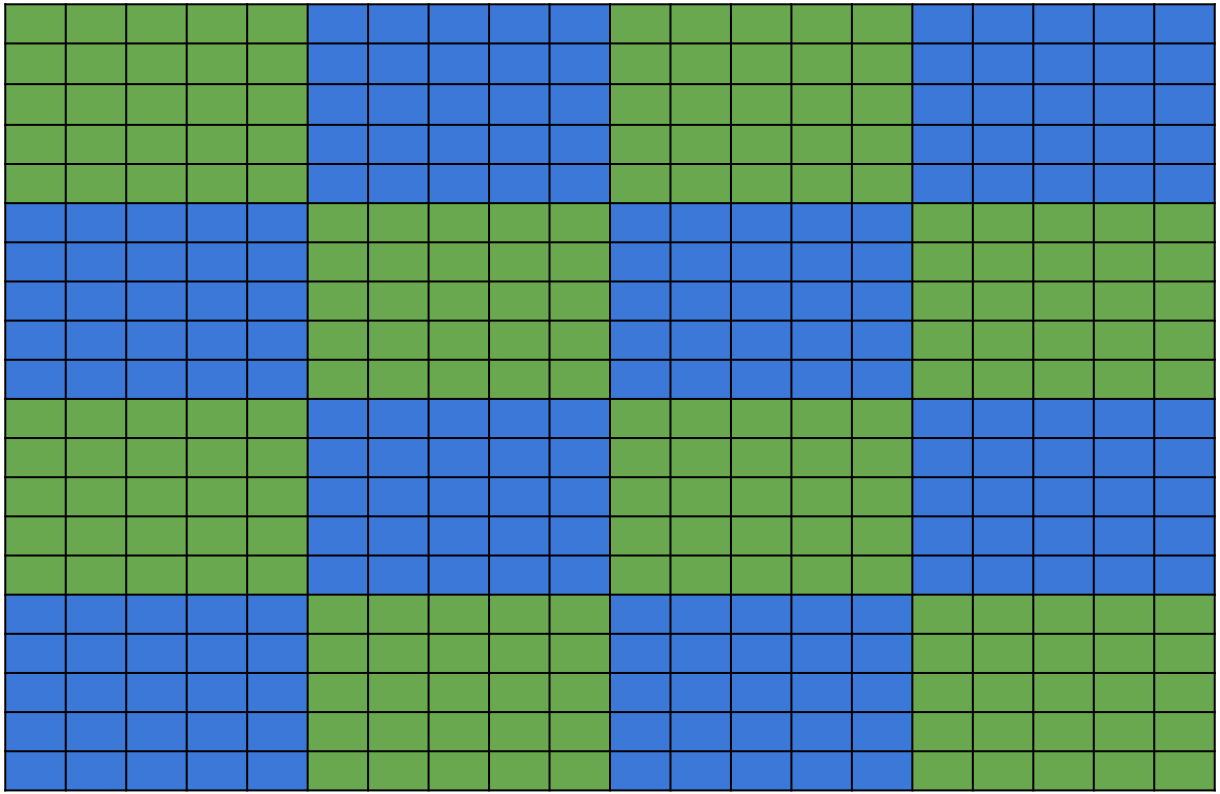
Contiguous layout - Row reading fast



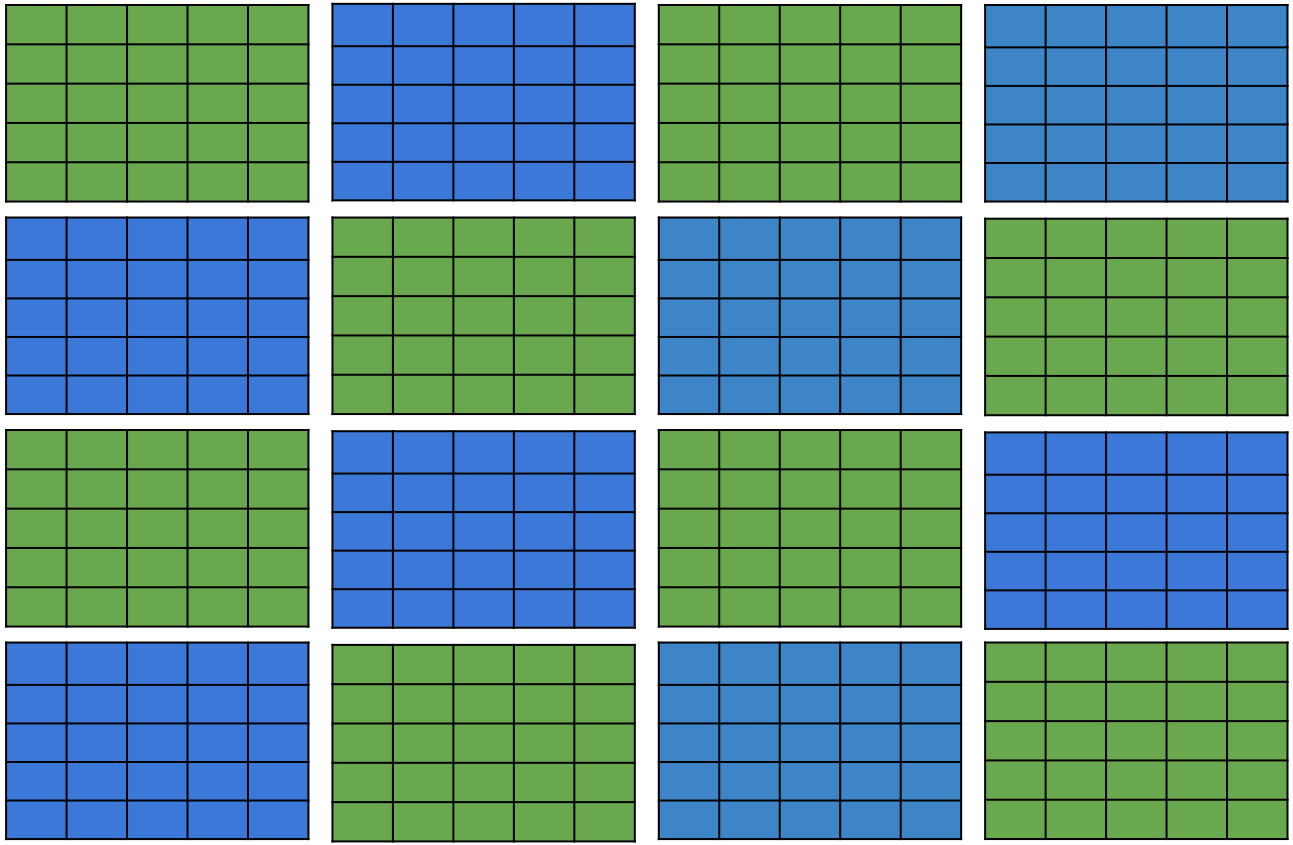
Contiguous layout - Column reading slow



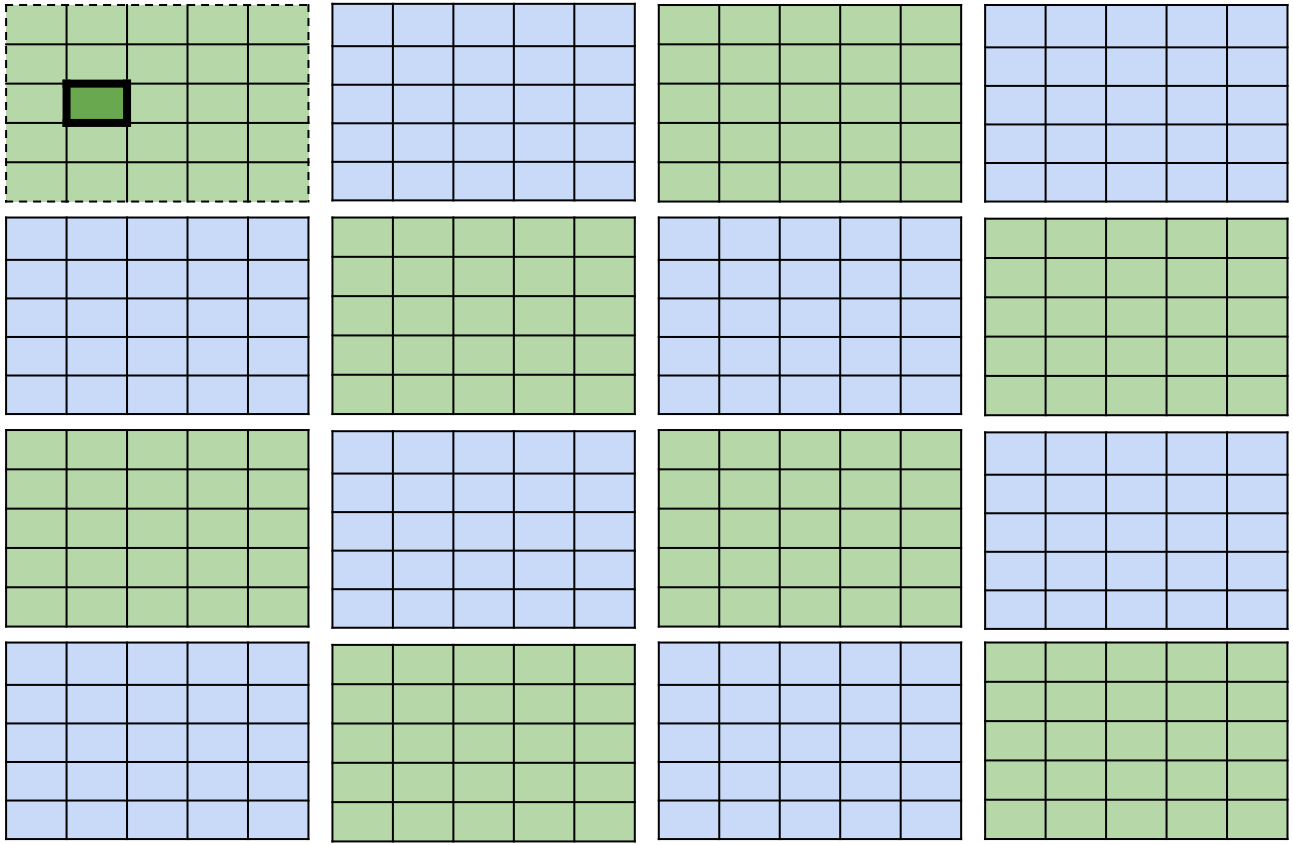
HDF5 datasets are not contiguous, but stored in chunks



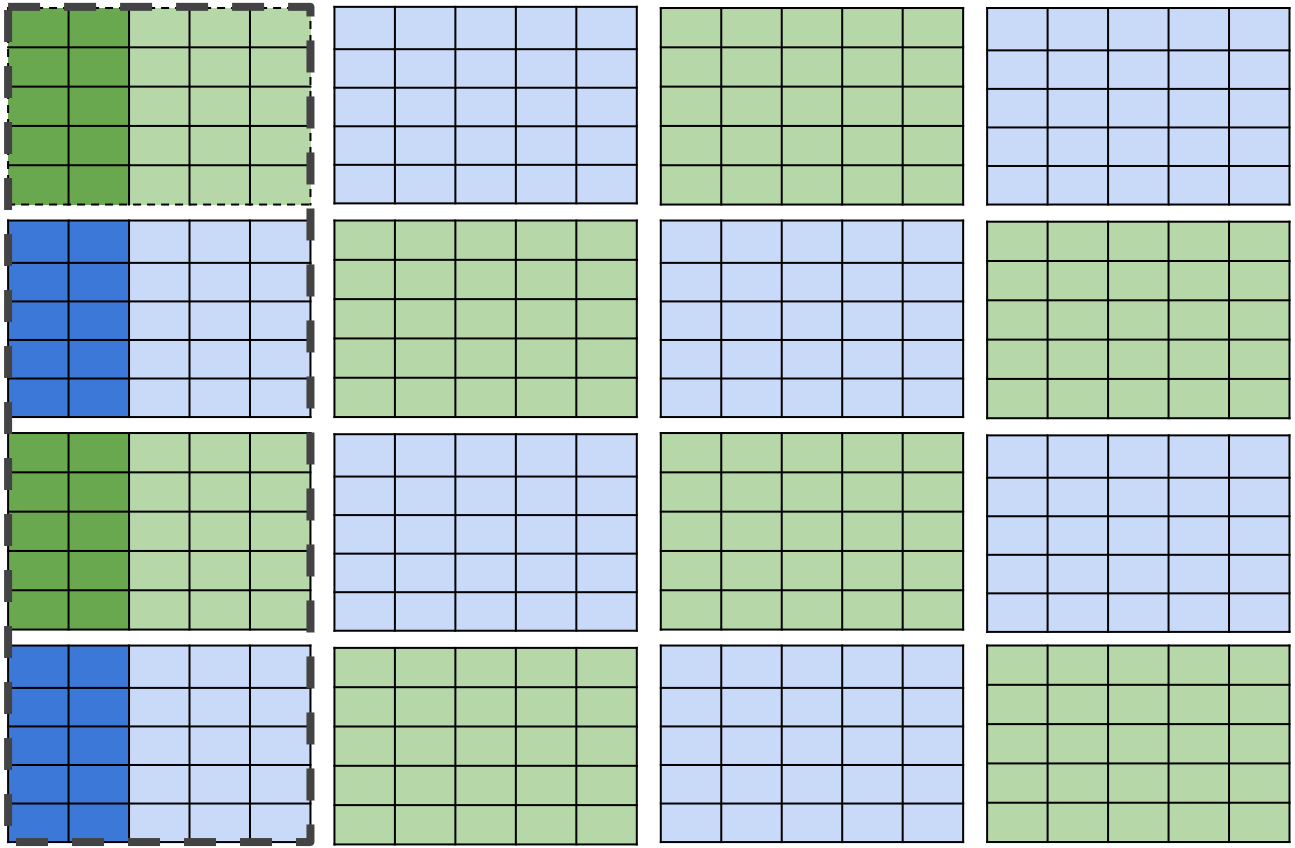
Chunks are stored separately on disk



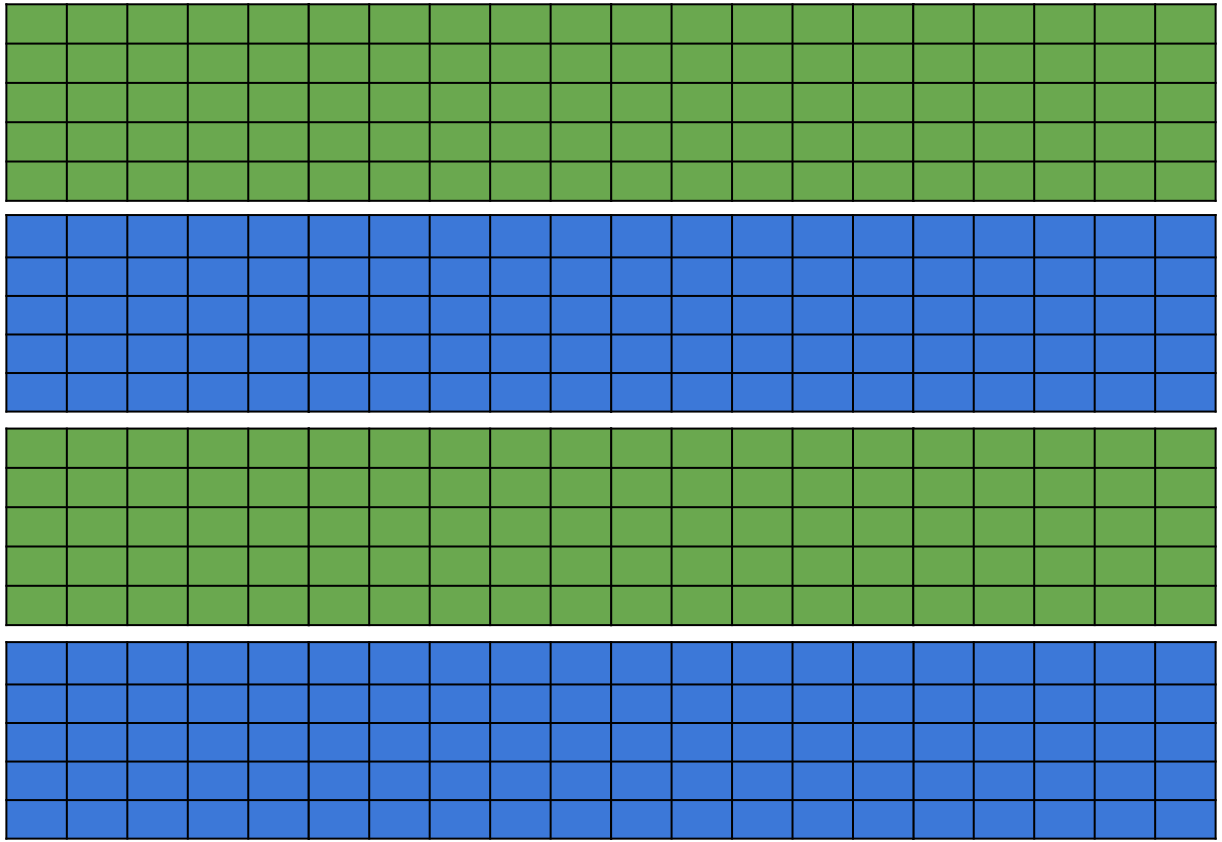
Only read the chunks needed for a subset



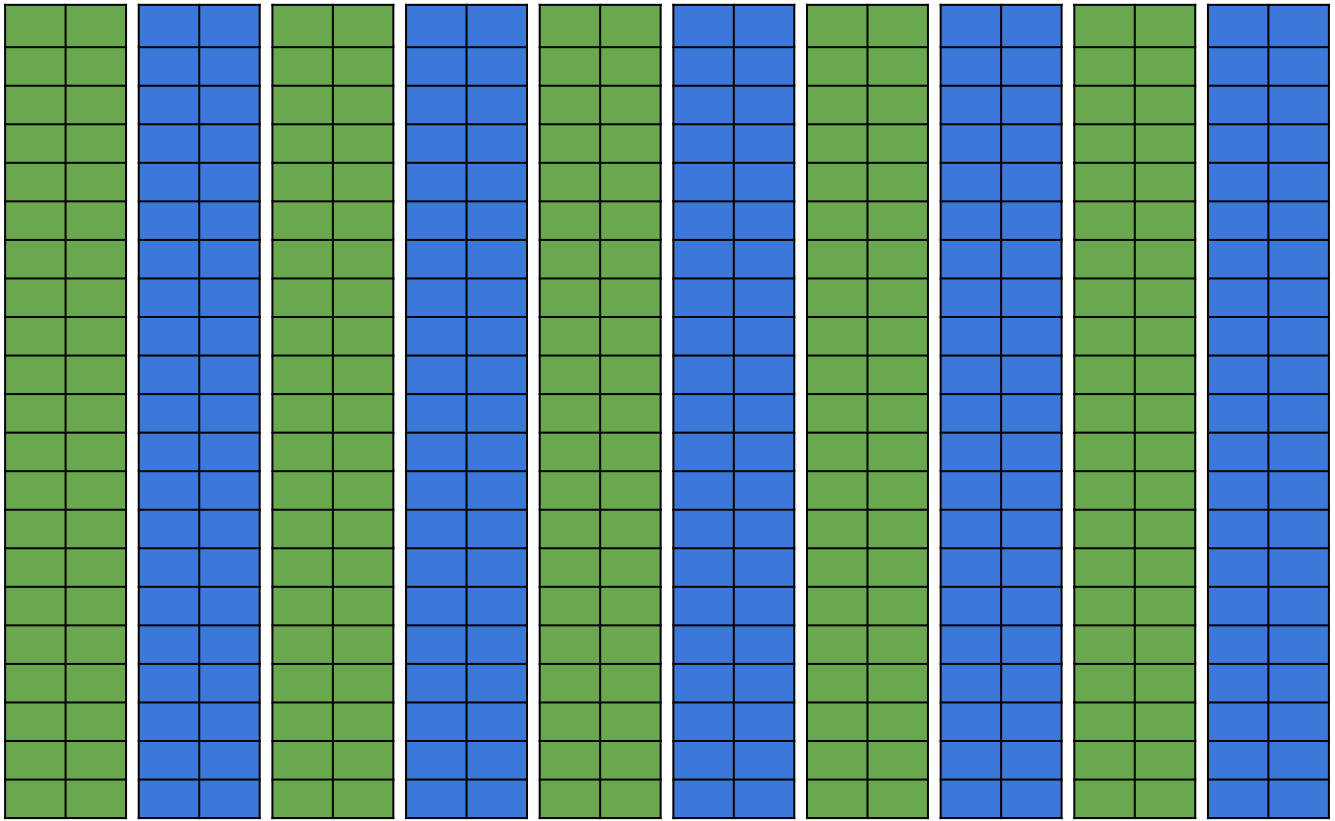
Only read the chunks needed for a subset



Chunks don’t need to be ‘square’



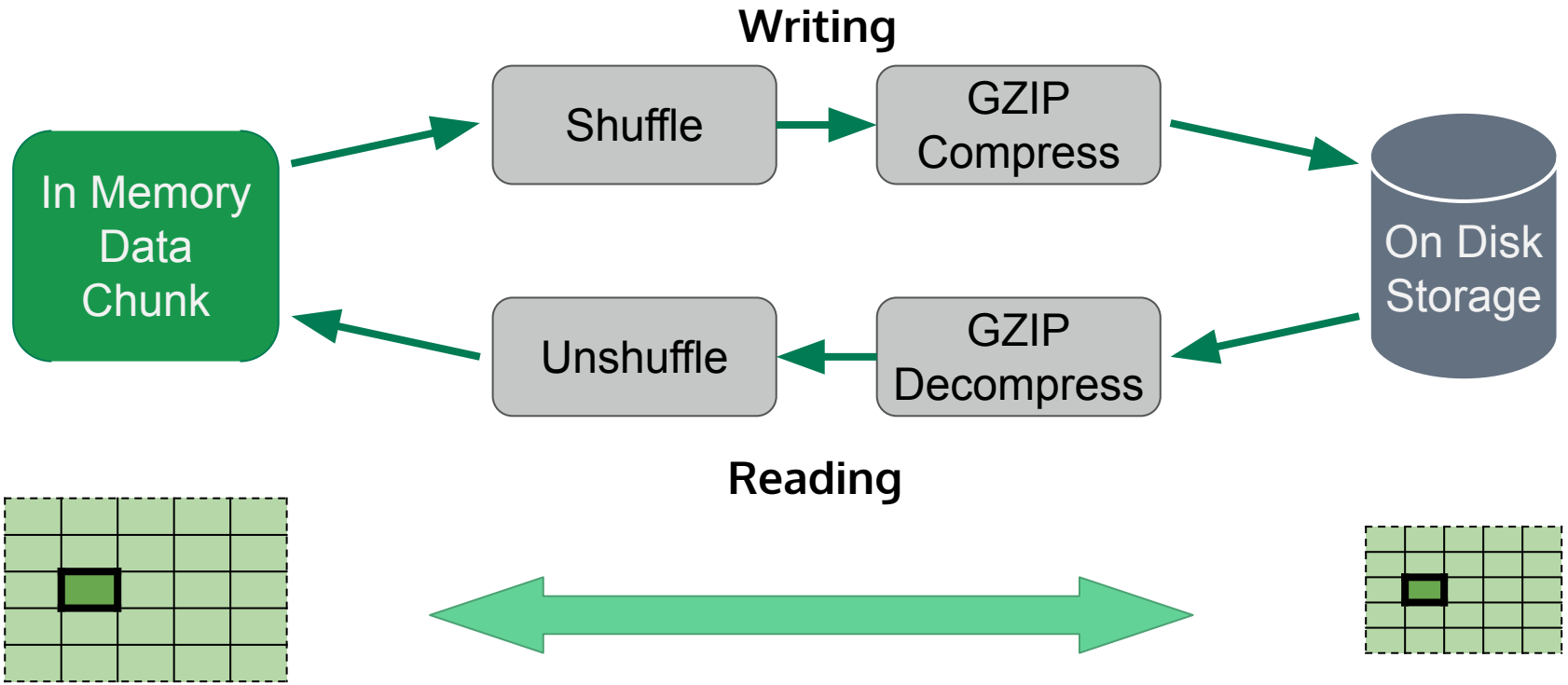
Chunks don’t need to be ‘square’



Picking chunk dimensions

- The size and shape of chunks can greatly impact the efficiency with which data can be retrieved
- If you know the access patterns you’re most likely to use, you can tailor the chunk shape to match
- If you don’t know, squares or shapes proportional to the overall dimensions are a good compromise
- Entire chunks need to be read - don’t make them too big!
- There’s an overhead to finding chunks - don’t make too many of them!

Chunks can be processed by filters - usually for compression



- This compression is why it’s OK to store the dense matrix
  - Lots of 0’s compress fairly well
- The effectiveness of any compression is also related to the chunk size.
  - Chunks of size 1 won’t compress at all
  - A single chunk compresses best, but throws away our subset options
-

These benefits apply to sparse representations too

- Each of our vectors can be chunked and compressed independently

Values

Row indices

Column pointers

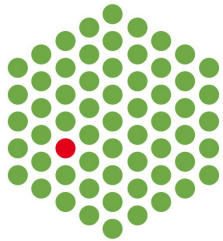
Summary

- There is no clear consensus on how to store single-cell datasets!
  - Both for file formats and data representations
- HDF5 probably has the biggest market share
  - Even here there are many varieties of layout
- Optimising is a balancing act between many competing factors
  - Knowing a specific use case can help, but often the middle ground is the best compromise
- Accessing data from disk is orders of magnitude slower than memory, these choices potentially have a huge impact on processing time

Thanks to EMBL Huber Lab & BioC community!

CHAN
ZUCKERBERG
INITIATIVE

EMBL



@grimbough