FLEXIBLE REPRESENTATIONS FOR COMPLEX DATA ON DISK

Kylie A. Bemis

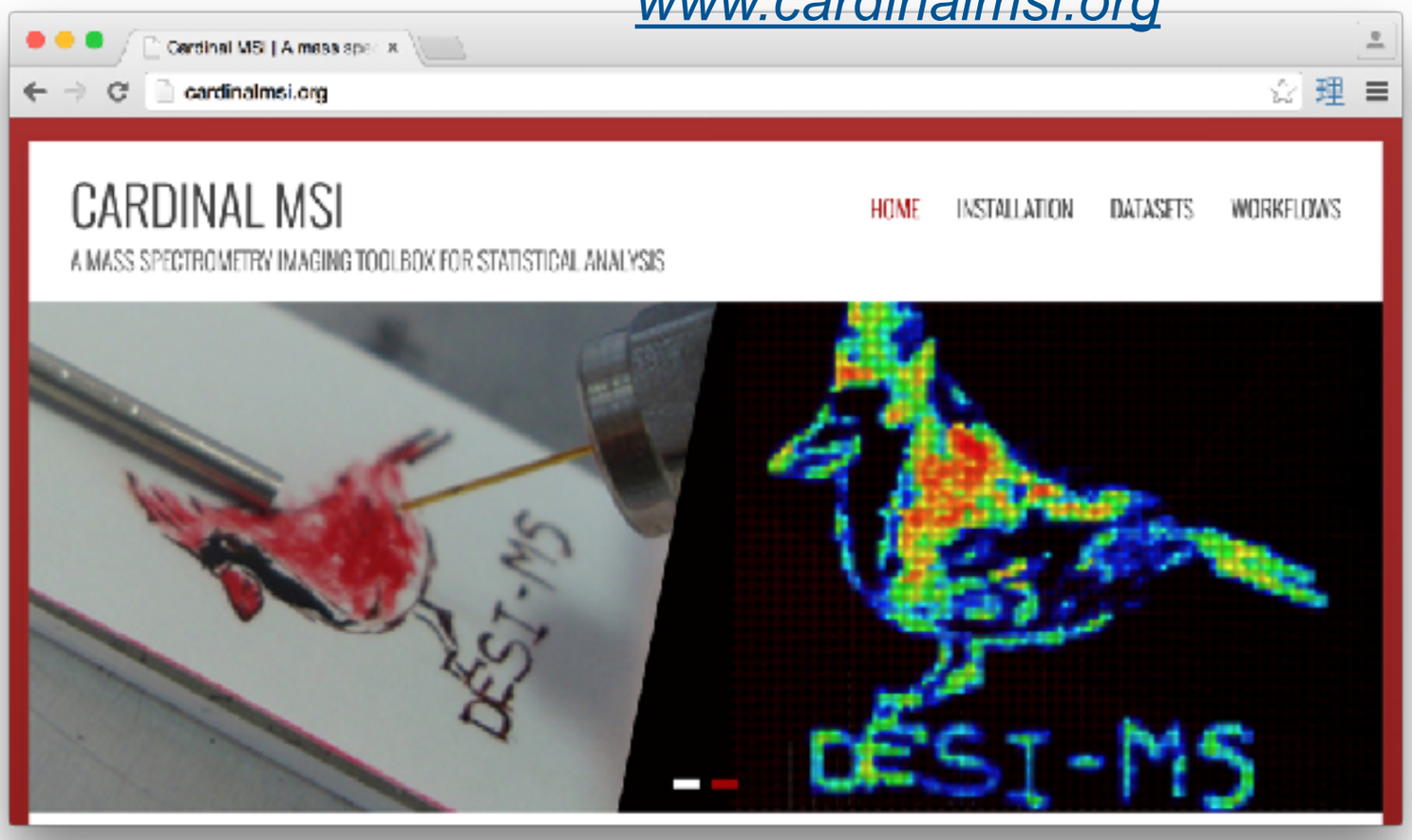
Northeastern University
College of Computer & Information Science

Northeastern University

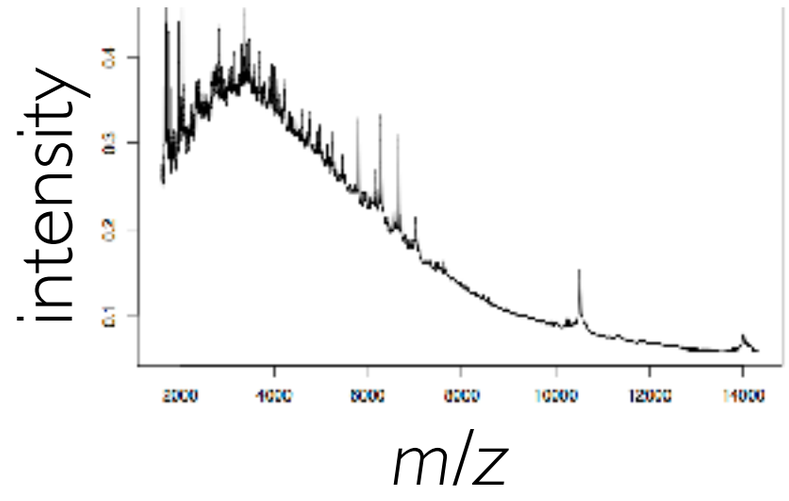
CARDINAL

open-source statistical software for MS imaging

www.cardinalmsi.org



- Free, open-source
- R-based
- Available on Bioconductor
- Source code on Github

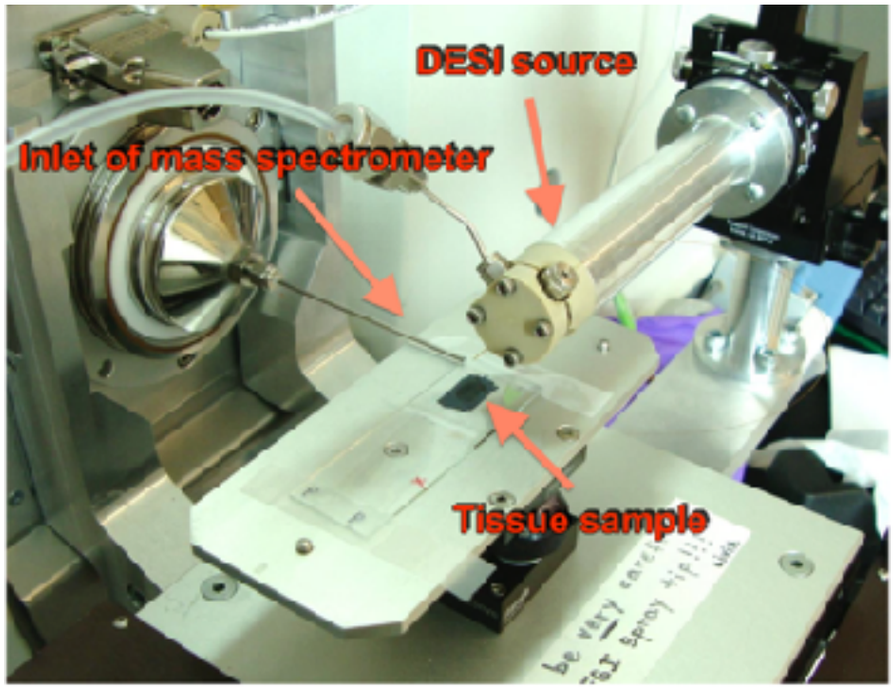


- >3,000 unique downloads since public release on April 17, 2015
- Winner of the 2015 John M. Chambers Statistical Software Award
- Latest release on May 1, 2018 with Bioconductor 3.7

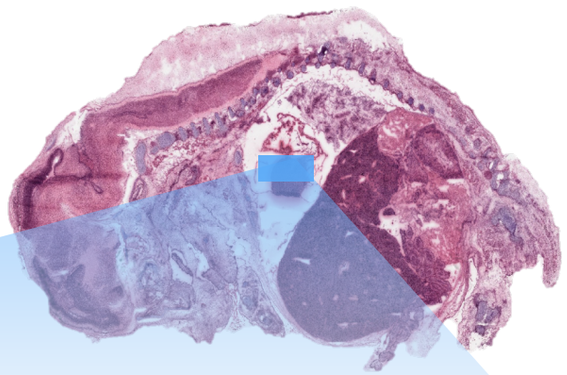
- K. D. Bemis, A. Harry, L. S. Eberlin, C. Ferreira, S. M. van de Ven, P. Mallick, M. Stolowitz, O. Vitek. "Cardinal: an R package for statistical analysis of mass spectrometry-based imaging experiments". Bioinformatics, 31:2418, 2015

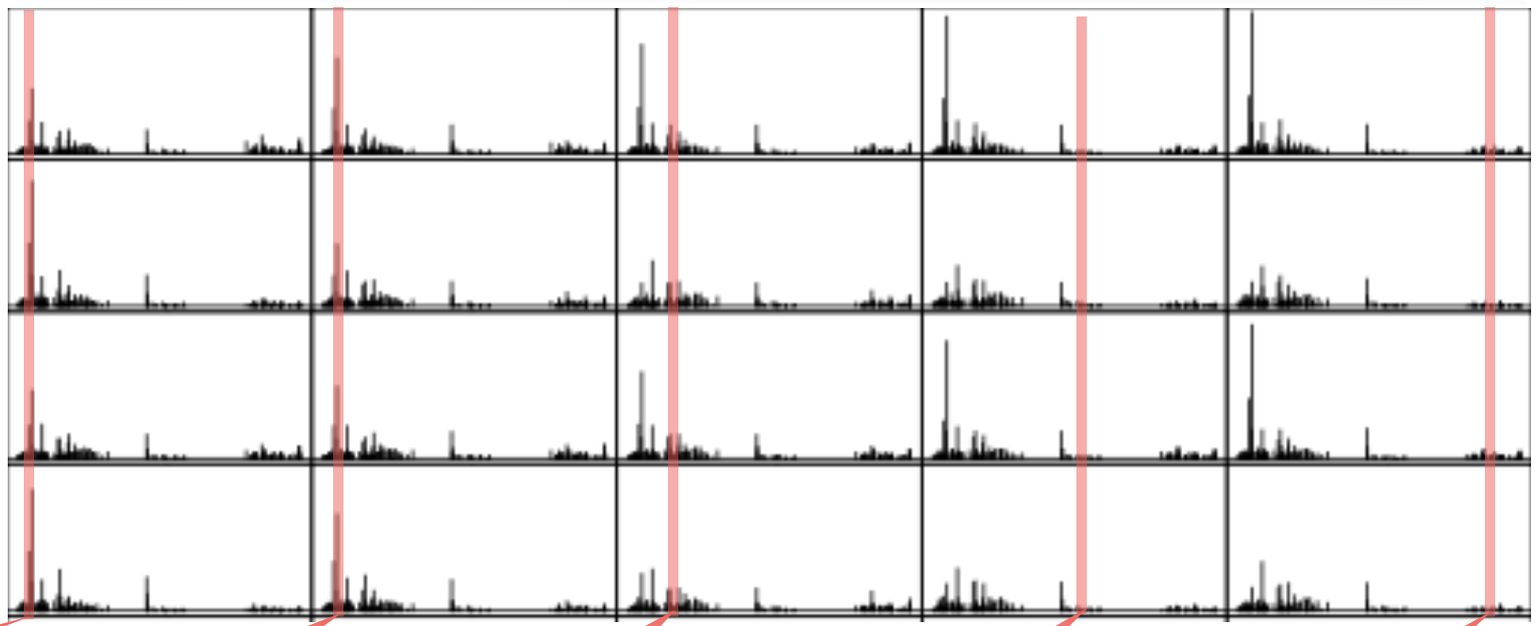
MASS SPECTROMETRY IMAGING

Investigate spatial distribution of analytes

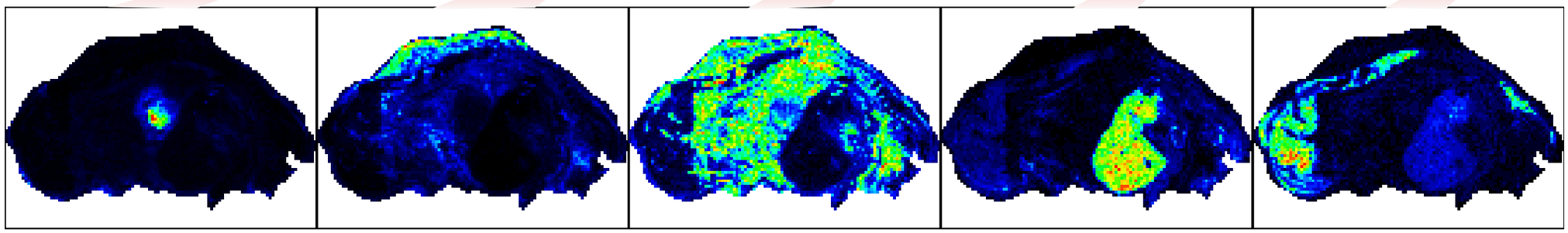


- Scan with laser/spray
- Collect mass spectra
- Reconstruct ion images
- Date "cube"





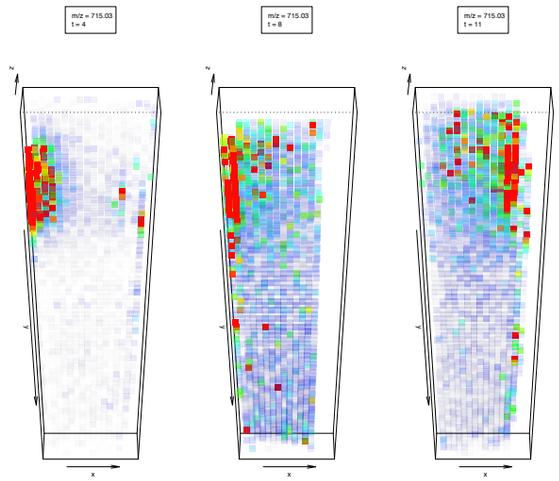
R. Graham Cooks and lab



PROBLEM: LARGER-THAN-MEMORY DATA

challenges statistical method development

- MS imaging experiments rapidly advancing
  - Increasing mass and spatial resolutions
  - Larger sample sizes, multiple files
- Growing data size poses difficulty for statistics
  - Need to test methods on larger-than-memory data
  - Need to work with domain-specific formats
  - Current R solutions are inflexible



120GB on disk

130GB RAM does not make it

I have a RAM with 32 GB but this will be insufficient.

20 GB, 2 slices

Greg Drazek

Hi Kyle,

I have data from flexImaging 4.1 (20 GB, 2 slices)

fmcp1979@gmail.com

Hi Kyle,
I would like to make a supervised analysis of a "big" set of samples. I have a RAM with 32 GB but this will be insufficient.

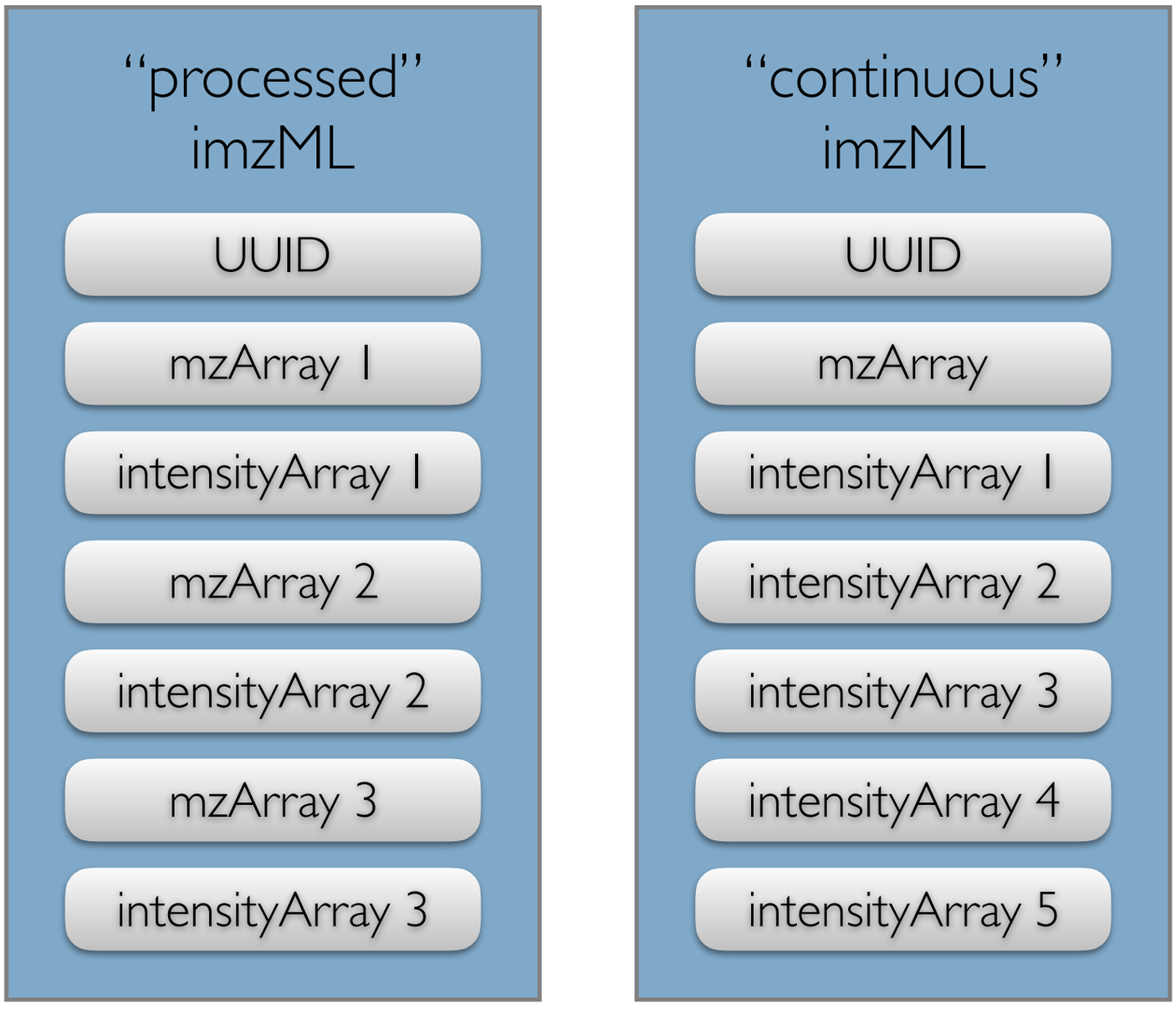
Uli Wellner

Dear all,

we are trying to load large processed imzml files (120GB on disk)
our server with 130GB RAM does not make it

Cardinal help Google group

NEED TO WORK WITH MS IMAGING FILES
e.g., "processed" and "continuous" imzML



- Open-source format for MS imaging experiments
- XML metadata file defines binary data file structure
  - Binary data schema is incompatible with *bigmemory* and *ff*
  - Prefer to avoid additional file conversion
    - Need random access into different parts of the file
- Often one-sample-per-file
  - Need to seamlessly work with multiple files in an experiment
  - Each file can be very large
- Need a flexible backend to solve these problems

PRE-EXISTING SOLUTIONS IN R

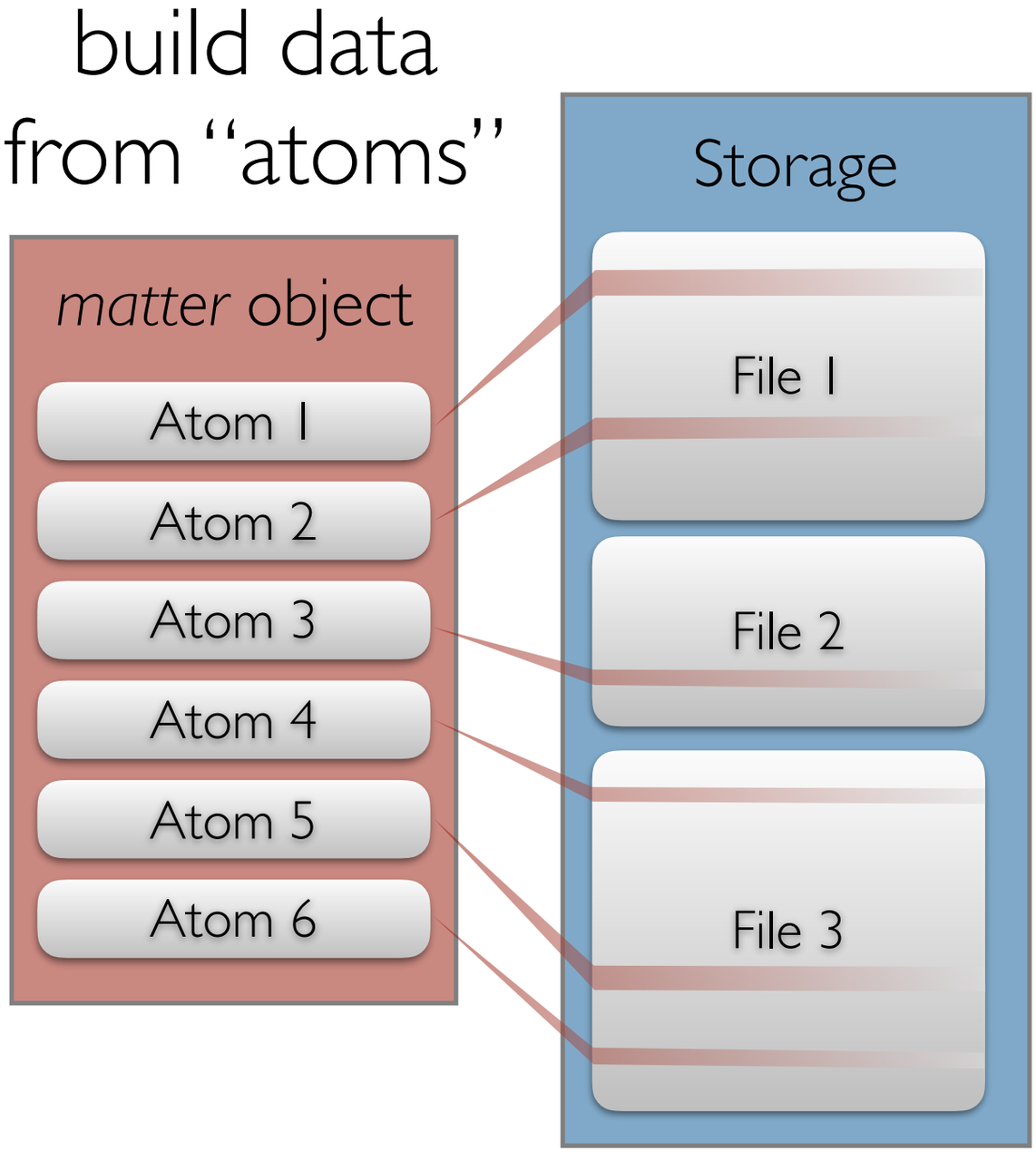
require file conversion

- *bigmemory*
  - Homogenous 2D column-major matrices only
  - Uses shared memory for in-memory matrices
  - Uses *mmap* for file-backed matrices
  - **File-backed matrices are flat files (1-file-per-matrix or 1-file-per-column)**
  - *Use of non-R memory made memory use difficult to track from R
- *ff*
  - Vectors, matrices, arrays, data frames, etc.
  - **Uses flat files for all objects (except data frames)**
  - **Limited to 2^31 elements** due to length being stored as a 32-bit integer
  - Used as a backend for **rMSI** package (Ràfols, et al., 2017)
- *DelayedArray + rhdf5*
  - **Conversion to HDF5 required**
  - **HDF5Array** was still early in development when *matter* development began

MATTER

open-source statistical computing with data on disk

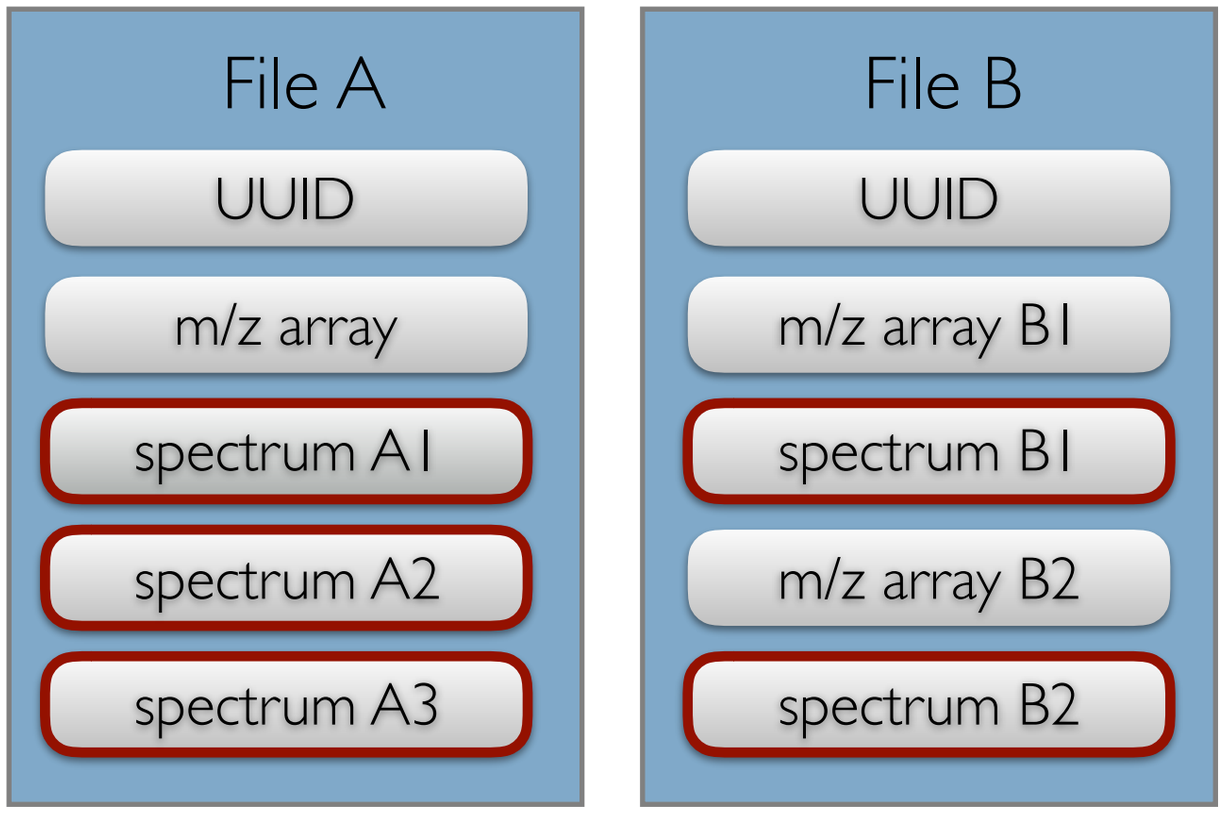
- Work with larger-than-memory datasets on disk in R
- Emphasizes flexibility with a minimal memory footprint
  - Adaptable to more file formats than *bigmemory* and *ff*
  - Potentially slower computation
- Designed for statistical method development in R
  - Rapid prototyping with minimal additional effort
  - Efficient calculation of summary statistics
  - Infrastructure for statistical computing on large data

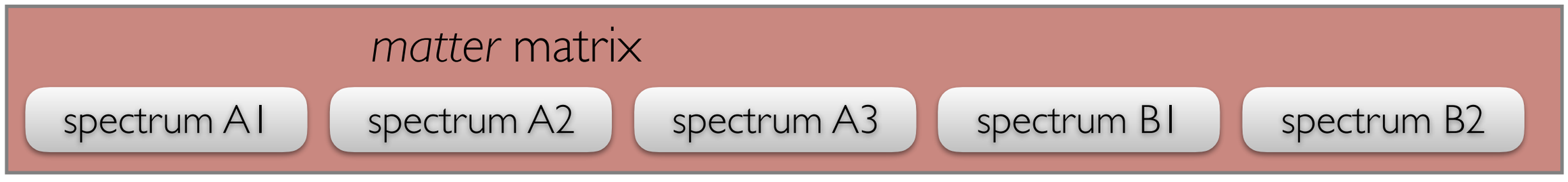


- K. A. Bemis, O. Vitek. "matter: an R package for rapid prototyping with larger-than-memory datasets on disk". Bioinformatics, doi:10.1093/bioinformatics/btx392

FLEXIBLE ACCESS TO DATA ON DISK
any binary format, any file structure

- User-defined file structure
- Data can come from anywhere
  - Any part of a file
  - Any combination of files
- Representation in R can be different from on disk
  - Access as ordinary R vector/matrix
  - No need to worry about data size or memory management





DATA REPRESENTATION

- Data structures in *matter* are built on the idea of "atoms"
- S4 class: *atoms*
  - An atom is sequence of data elements stored in a contiguous location on disk
  - Each atom can be loaded into memory in a single disk read operation
  - An atom is defined by:
    - a ***path*** to the data source (i.e., a file) containing the data
    - a byte ***offset*** from the beginning of the file
    - an ***extent*** giving the number of data elements
    - a ***group*** which may describe relationship with other atoms
- S4 class: *matter*
  - A collection of atoms with additional slots (`length`, `dim`, etc.)
  - Subclass methods define how atoms are converted to user representation

Create an on-disk *matter* vector *x*

```
x <- matter(1:10)
x
```

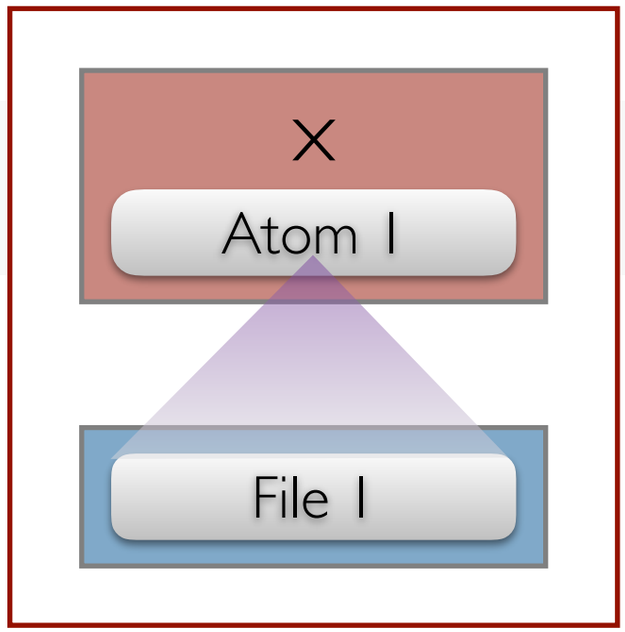
```
## An object of class 'matter_vec'
##   <10 length> on-disk vector
##     sources: 1
##     datamode: integer
##     5.8 KB in-memory
##     40 bytes on-disk
```

```
x[]
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
atomdata(x)
```

```
##   group_id source_id datamode offset extent index_offset index_extent
## 1        1         1      int      0     10            0           10
```



Create another on-disk *matter* vector *y*

```
y <- matter(11:20)
y
```

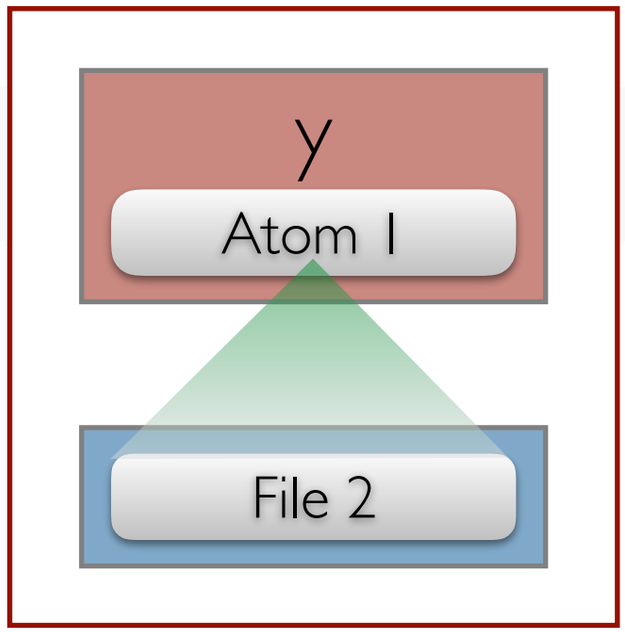
```
## An object of class 'matter_vec'
##   <10 length> on-disk vector
##     sources: 1
##     datamode: integer
##     5.8 KB in-memory
##     40 bytes on-disk
```

```
y[]
```

```
##  [1] 11 12 13 14 15 16 17 18 19 20
```

```
atomdata(y)
```

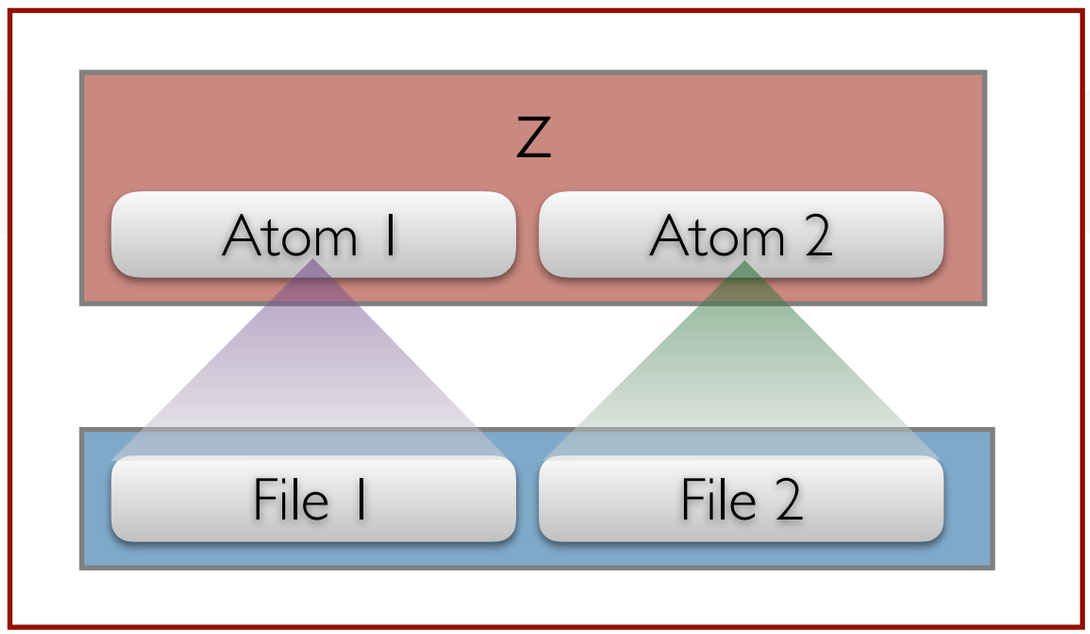
```
##   group_id source_id datamode offset extent index_offset index_extent
## 1        1         1      int      0     10            0           10
```



Combine on-disk vectors

```
z <- c(x, y)
z
```

```
## An object of class 'matter_vec'
##   <20 length> on-disk vector
##     sources: 2
##     datamode: integer
##     6 KB in-memory
##     80 bytes on-disk
```




```
z[]
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
atomdata(z)
```

```
##   group_id source_id datamode offset extent index_offset index_extent
## 1        1         1      int      0     10            0           10
## 2        1         2      int      0     10           10           20
```

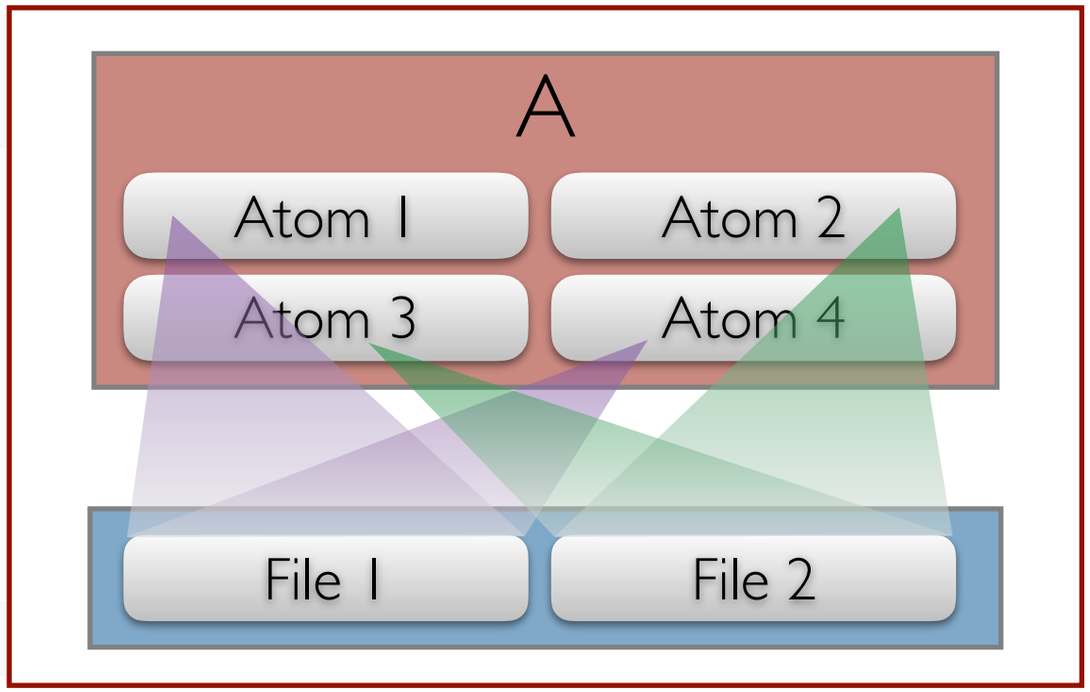
Bind on-disk vectors into matrices

```
A <- rbind(z, c(y, x))
A
```

```
## An object of class 'matter_matr'
##   <2 row, 20 column> on-disk matrix
##     sources: 2
##     datamode: integer
##     6 KB in-memory
##     160 bytes on-disk
```

```
A[]
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,]    1    2    3    4    5    6    7    8    9    10    11    12    13
## [2,]   11   12   13   14   15   16   17   18   19    20     1     2     3
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20]
## [1,]    14    15    16    17    18    19    20
## [2,]     4     5     6     7     8     9    10
```



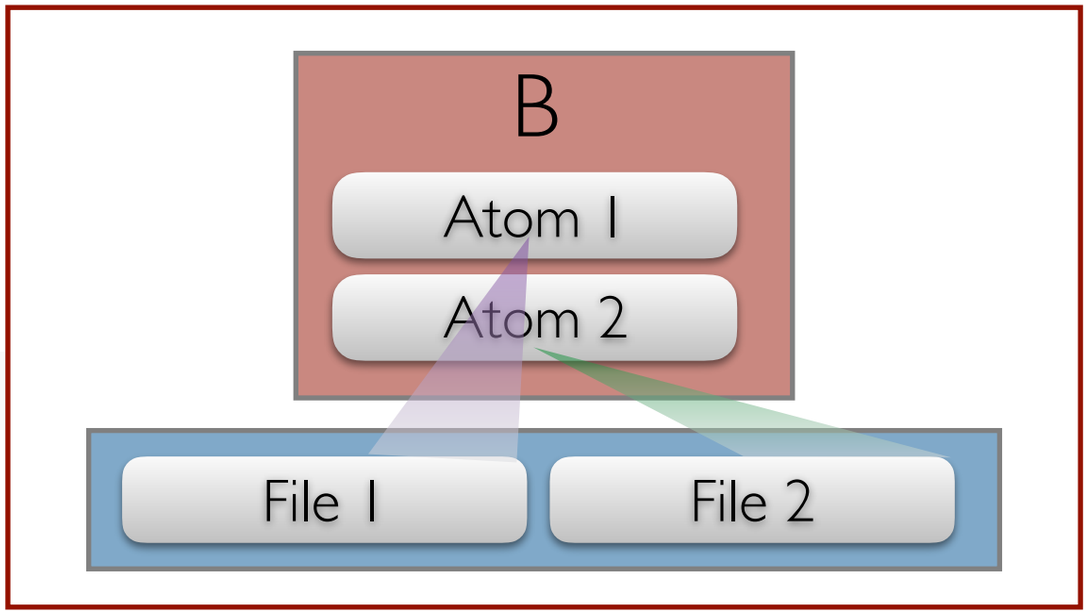

```
atomdata(A)
```

```
##   group_id source_id datamode offset extent index_offset index_extent
## 1        1         1      int      0     10            0           10
## 2        1         2      int      0     10           10           20
## 3        2         2      int      0     10            0           10
## 4        2         1      int      0     10           10           20
```

Create on-disk *matter* matrix from metadata

```
sizeof_int <- 4
B <- matter_mat(datamode="int",
                nrow=2, ncol=5,
                paths=c(paths(x)[1], paths(y)[1]),
                offset=c(5 * sizeof_int, 5 * sizeof_int),
                extent=c(5,5),
                rowMaj=TRUE)
B
```

```
## An object of class 'matter_matr'
##   <2 row, 5 column> on-disk matrix
##     sources: 2
##     datamode: integer
##     6 KB in-memory
##     40 bytes on-disk
```




```
B[]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    6    7    8    9   10
## [2,]   16   17   18   19   20
```

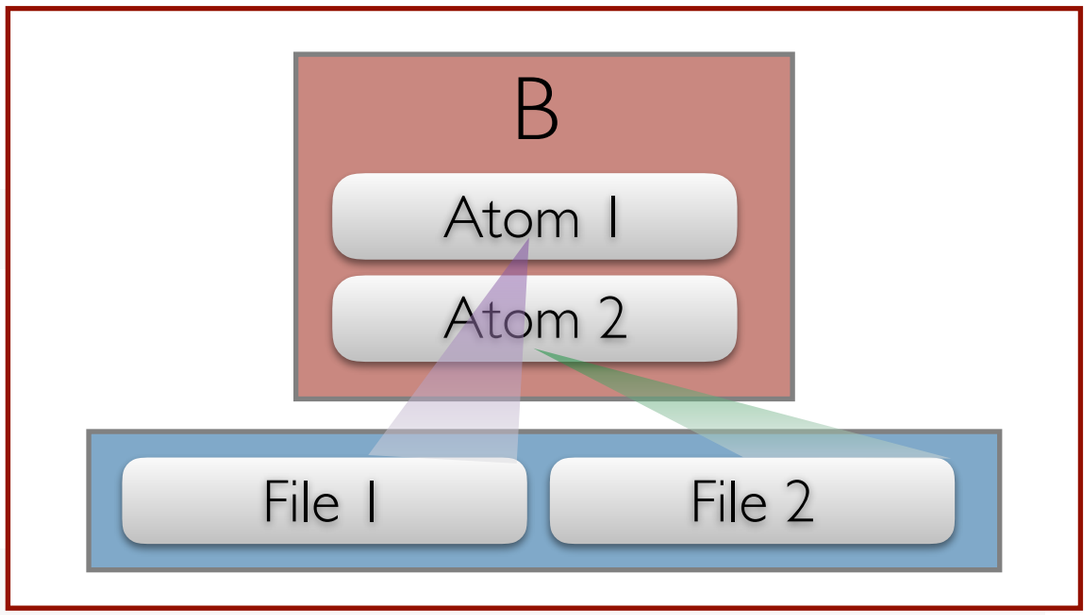
```
atomdata(B)
```

```
##   group_id source_id datamode offset extent index_offset index_extent
## 1        1         1      int     20      5            0            5
## 2        2         2      int     20      5            0            5
```

Endomorphic subsetting with drop=NULL

```
B2 <- rbind(x, y)
B2 <- B2[, 6:10, drop=NULL]
B2
```

```
## An object of class 'matter_matr'
##   <2 row, 5 column> matrix
##     sources: 2
##     datamode: integer
##     6.3 KB real memory
##     40 bytes virtual memory
```




```
B2[]
```

```
##       [,1] [,2] [,3] [,4] [,5]
## [1,]    6    7    8    9   10
## [2,]   16   17   18   19   20
```

```
atomdata(B2)
```

```
##   group_id source_id datamode offset extent index_offset index_extent
## 1        1         1      int     20      5            0            5
## 2        2         2      int     20      5            0            5
```

PACKAGE FEATURES

- Data structures
  - `matter_vec` (integer & numeric vectors)
  - `matter_mat` (integer & numeric matrices)
  - `matter_arr` (integer & numeric n-D arrays)
  - `matter_list` (lists of integer & numeric vectors)
  - `matter_str` (character vectors)
  - `matter_df` (data frames)
- Delayed operations
  - `Arith` (`+`, `-`, `*`, `^`, `%%`, `%/%`, `/`)
  - `Compare` (`==`, `>`, `<`, `!=`, `<=`, `>=`)
  - `Math` (`exp`, `log`, `log2`, `log10`)
- Efficient statistical summaries
  - `sum()`, `mean()`, `sd()`, `var()`, `etc.`
  - `colSums()`, `colMeans()`, `colSds()`, `colVars()`, `etc.`

DELAYED OPERATIONS

- Similar to the *DelayedArray* class, certain operations applied to *matter* matrices and vectors will have their execution delayed until data is accessed
- The delayed operation will be applied only to the portion of data that is accessed and no other part of the data
- Attempting to combine a *matter* objects with delayed operations drops the delayed operations with a warning
- All delayed operations are applied in efficient C++ code as soon as the data is read from disk so `mean()` etc. work correctly

```
logz <- log2(z + 1)
head(z)
```

```
## [1] 1 2 3 4 5 6
```

```
head(logz)
```

```
## [1] 1.000000 1.584963 2.000000 2.321928 2.584963 2.807355
```

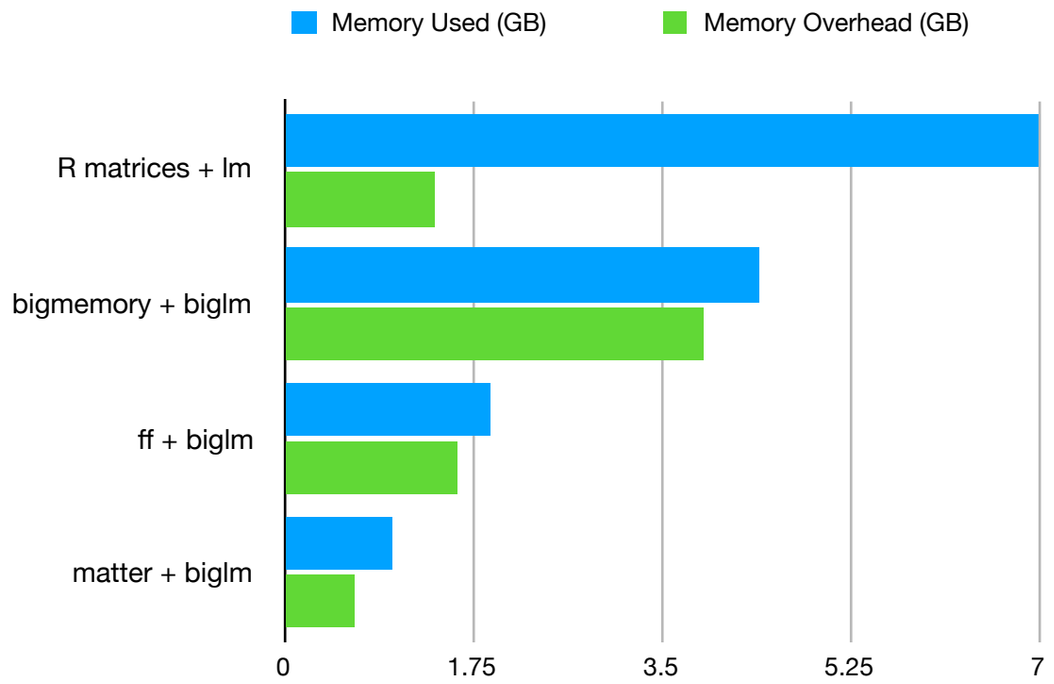
```
z3 <- c(z, logz)
```

```
## Warning in combine(x, ...): dropping delayed operations
```

EXAMPLES

EXAMPLE: LINEAR REGRESSION
with a 1.2 GB simulated dataset

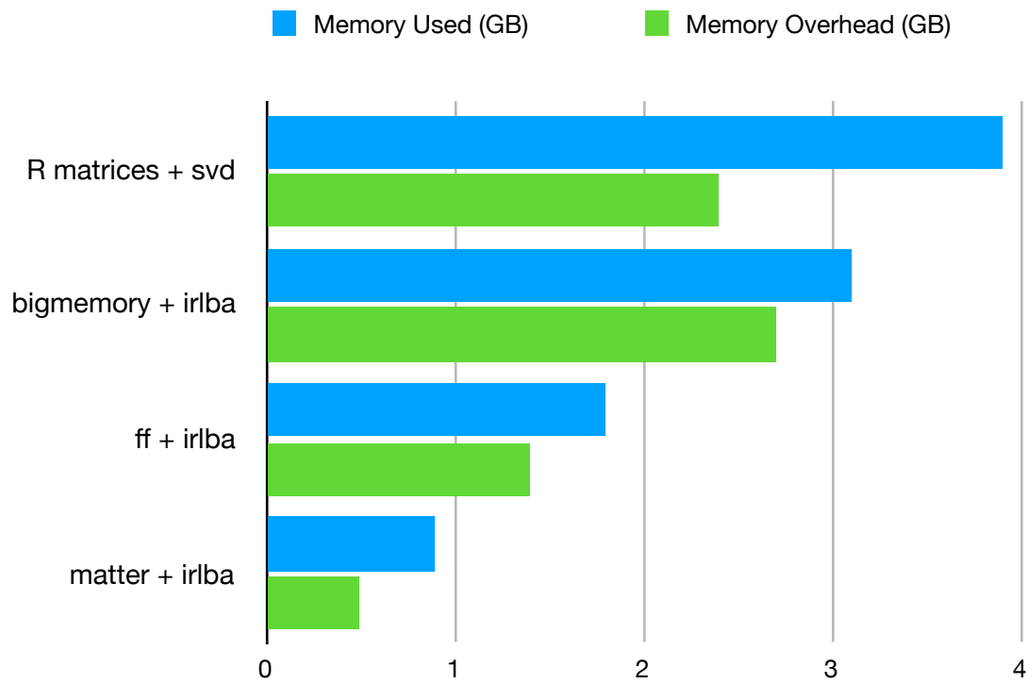
- 1.2 GB dataset
  - N = 15,000,000 observations
  - P = 9 variables
- Linear regression
  - Using *biglm* package
  - Specifically for large datasets



	Memory Used	Memory Overhead	Time
R matrices + lm	7 GB	1.4 GB	33 sec
bigmemory + biglm	4.4 GB	3.9 GB	21 sec
ff + biglm	1.9 GB	1.6 GB	57 sec
matter + biglm	1 GB	660 MB	47 sec

EXAMPLE: PRINCIPAL COMPONENTS ANALYSIS
with a 1.2 GB simulated dataset

- 1.2 GB dataset
  - N = 15,000,000 observations
  - P = 10 variables
- PCA
  - Using *irlba* package
  - Not specifically for large datasets



	Memory Used	Memory Overhead	Time
R matrices + svd	3.9 GB	2.4 GB	66 sec
bigmemory + irlba	3.1 GB	2.7 GB	15 sec
ff + irlba	1.8 GB	1.4 GB	130 sec
matter + irlba	890 MB	490 MB	110 sec

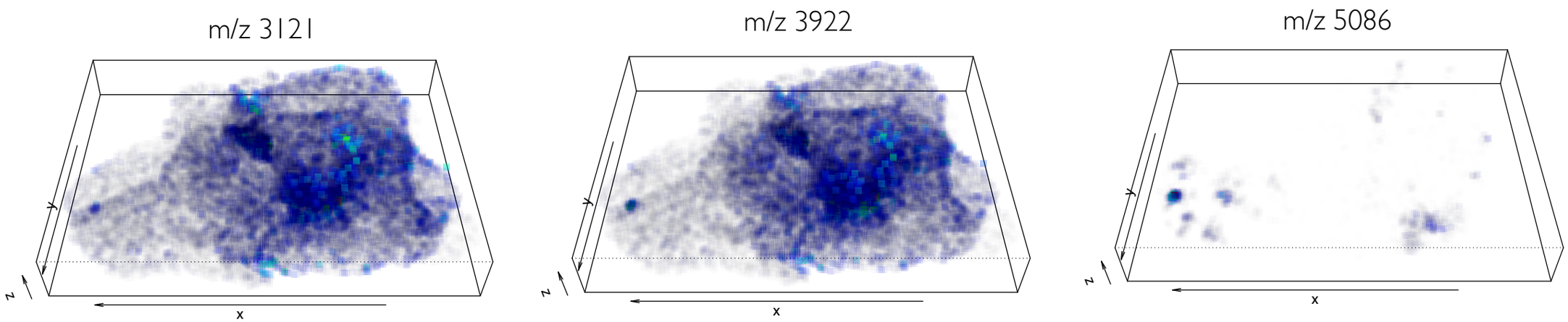
EXAMPLE: VISUALIZATION
of a 26.45 GB mouse pancreas experiment

- 3D mouse pancreas
- 26.45 GB on disk
- 497,225 pixels
- 13,312 features

Work with arbitrarily large datasets from any number of files

Example: 26.45 GB dataset on a personal laptop w/ 16 GB RAM

Oetjen et al, Gigascience, 2015




```
mouse <- readMSIData("3D_Mouse_Pancreas.imzML", attach.only=TRUE)

image3D(mouse, mz=c(3121, 3922, 5086), layout=c(1,3))
```

EXAMPLE: VISUALIZATION
of a 26.45 GB mouse pancreas experiment

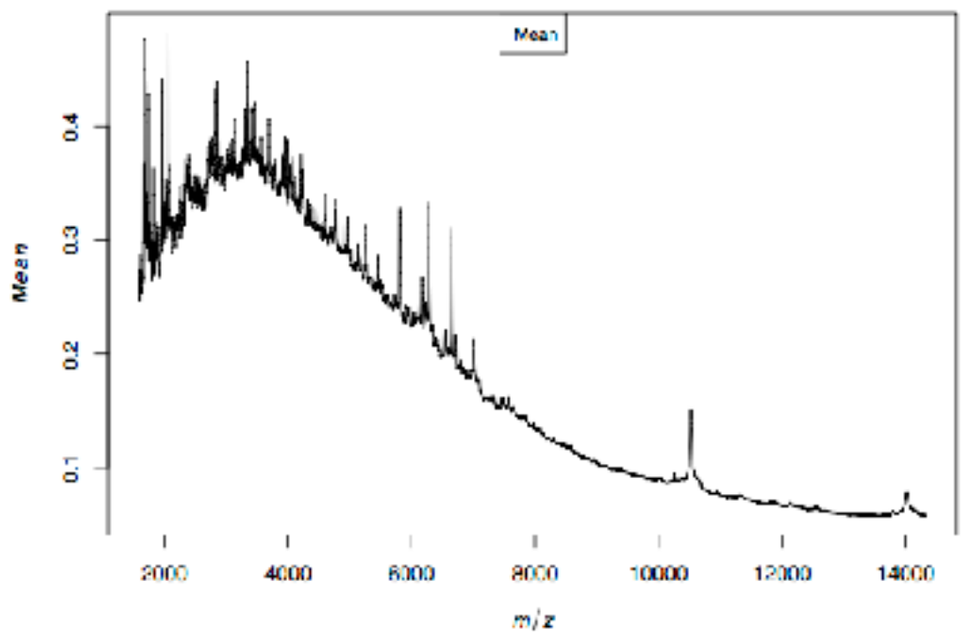
- 3D mouse pancreas
- 26.45 GB on disk
- 497,225 pixels
- 13,312 features

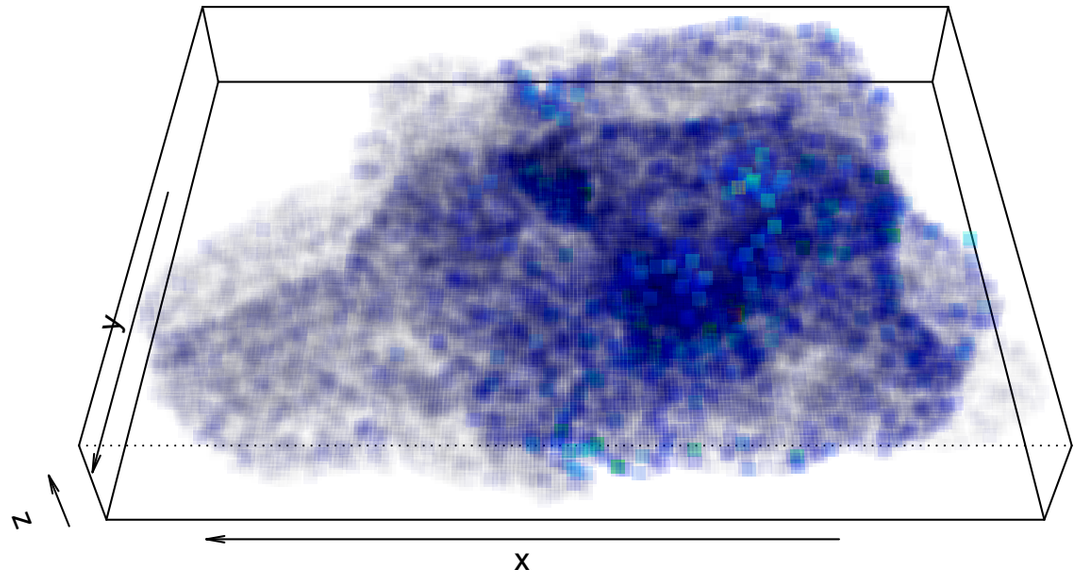
Work with arbitrarily large datasets from any number of files

Example: 26.45 GB dataset on a personal laptop w/ 16 GB RAM

Oetjen et al, Gigascience, 2015

```
fData(mouse)$Mean <- featureApply(mouse, mean)
plot(mouse, Mean ~ mz)
```



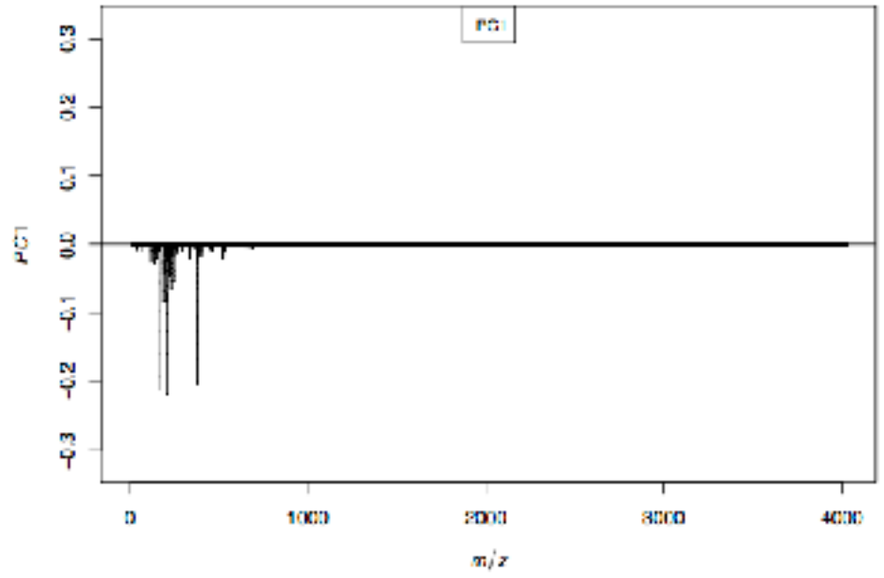



```
pData(mouse)$TIC <- pixelApply(mouse, sum)
image3D(mouse, TIC ~ x * y * z)
```

EXAMPLE: PRINCIPAL COMPONENTS ANALYSIS
with a 2.85 GB microbial time-course experiment

- 3D microbial time-course
- 2.85 GB on disk
- 17,672 pixels
- 40,299 features

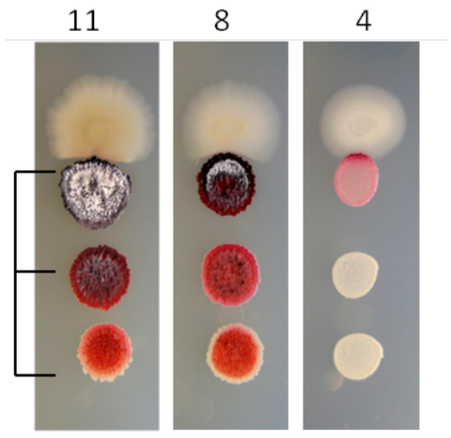
13 min,
6 sec
for 2 PC

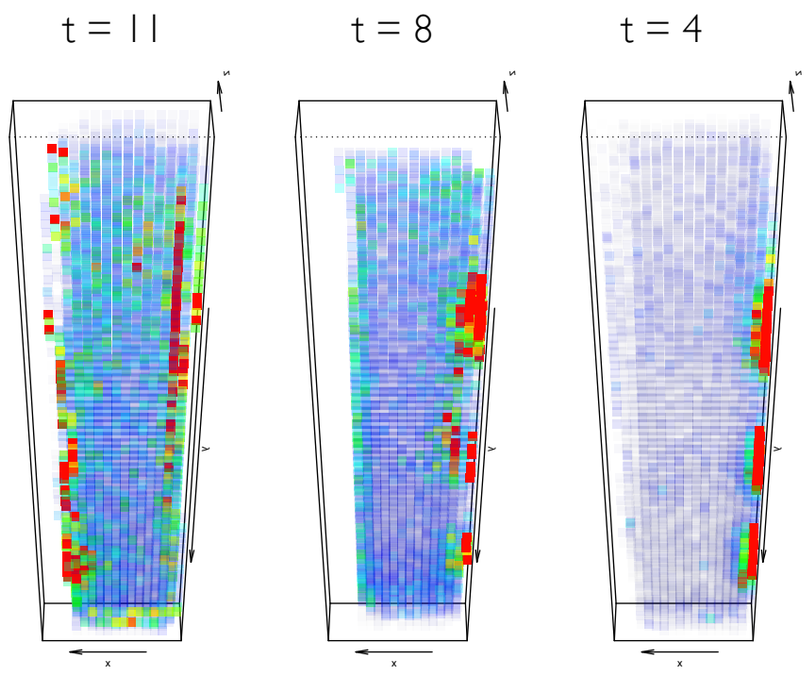


PC1 loadings

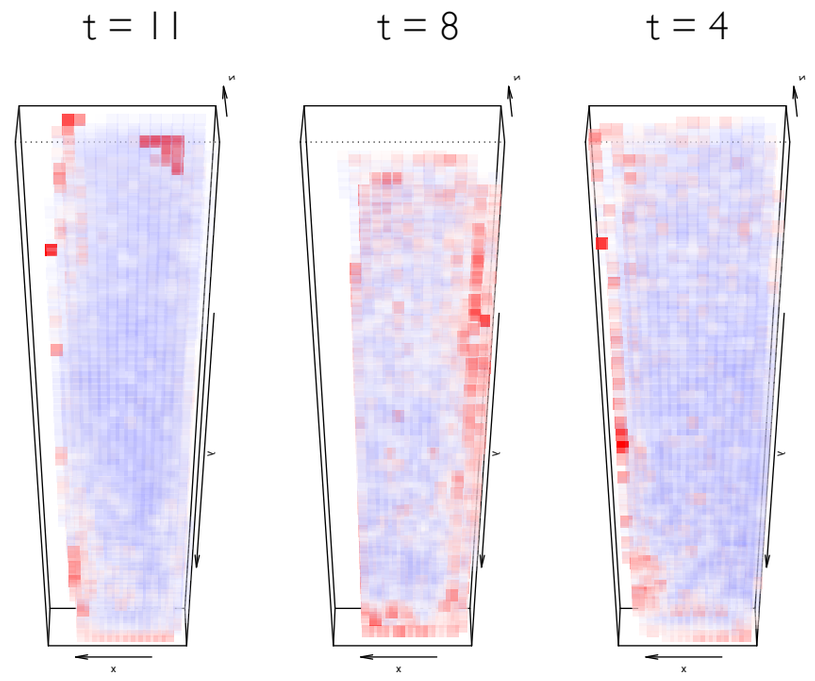
228 MB total memory used
50 MB memory overhead

Oetjen et al, Gigascience, 2015





m/z 262



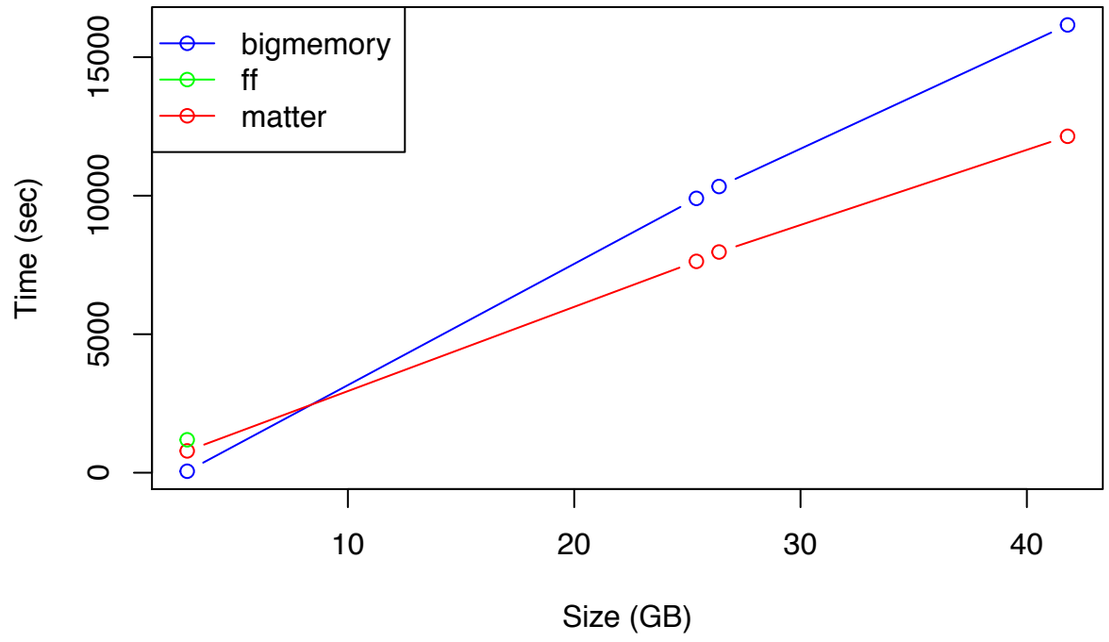
PC1 scores

EXAMPLE: PRINCIPAL COMPONENTS ANALYSIS
with all 'continuous' imzML datasets in an open repository

Oetjen et al, Gigascience, 2015

Up to:

- 3D mouse kidney
- 41.8 GB on disk
- 1,362,830 pixels
- 7,680 features



Principal components analysis					
Dataset	Size	Method	Mem. used	Mem. overhead	Time
3D Microbial Time Course	2.9 GB	matter	228 MB	50 MB	13 min, 6 sec
		bigmemory	330 MB	141 MB	53 sec
		ff	278 MB	85 MB	19 min, 48 sec
3D Oral Squamous Cell Carcinoma	25.4 GB	matter	977 MB	668 MB	2 hr, 7 min, 9 sec
		bigmemory	408 MB	266 MB	2 hr, 28 min, 2 sec
		ff	–	–	–
3D Mouse Pancreas	26.4 GB	matter	628 MB	370 MB	2 hr, 12 min, 46 sec
		bigmemory	303 MB	164 MB	2 hr, 52 min, 10 sec
		ff	–	–	–
3D Mouse Kidney	41.8 GB	matter	1.5 GB	1074 MB	3 hr, 22 min, 23 sec
		bigmemory	617 MB	431 MB	4 hr, 29 min, 23 sec
		ff	–	–	–

**bigmemory* memory use tracking not reliable within R due to use of shared memory

EXAMPLE: 'PROCESSED' IMZML
sparse mass spectrometry data-on-disk

Römpp et al, Angew. Chem. Int. Ed., 2016

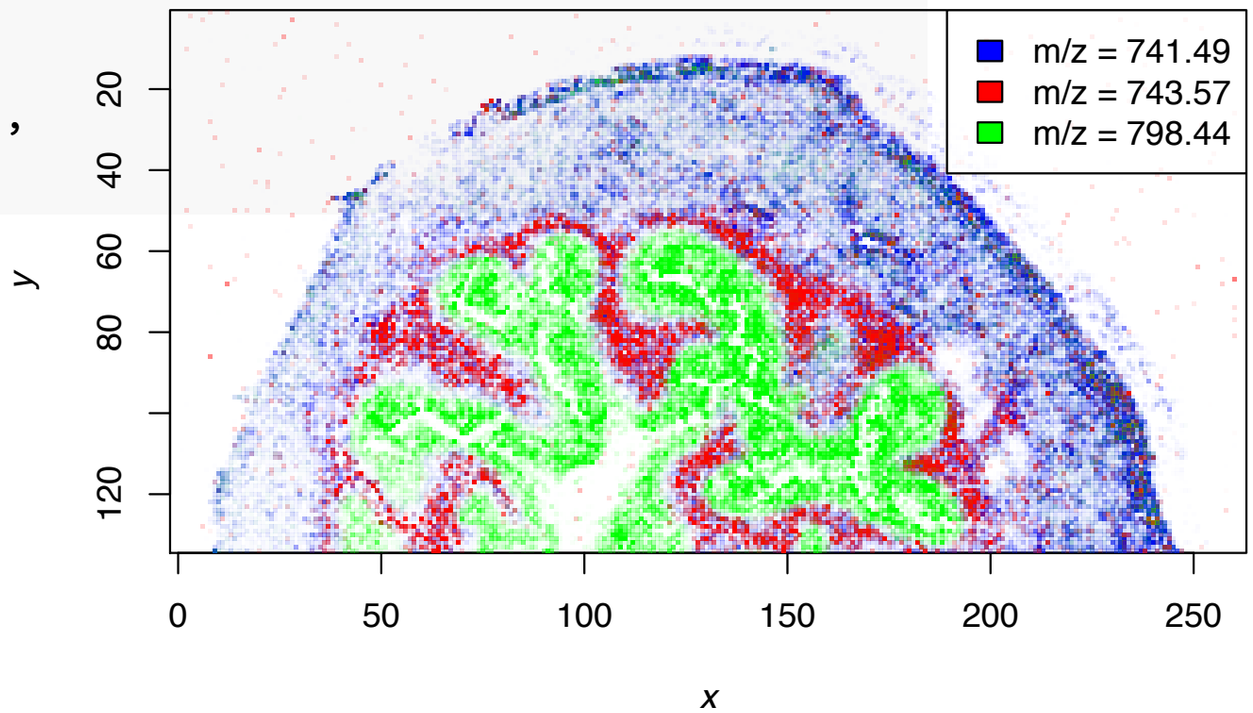
```
library(Cardinal)

filename <- paste0("~/Documents/Datasets/PRIDE/",
                   "HR2MSI mouse urinary bladder S096/",
                   "HR2MSI mouse urinary bladder S096.imzML")

mouse <- readMSIData(filename, attach.only=TRUE)

image(mouse,
      mz=c(741.5307, 743.5482, 798.541),
      plusminus=0.1,
      col=c("blue", "red", "green"),
      normalize.image="linear",
      contrast.enhance="suppression",
      key=TRUE, superpose=TRUE)
```

- Mouse bladder
- 815 MB on disk
- 'Processed' imzML
- Sparse mass spectra



EXAMPLE: 'PROCESSED' IMZML
sparse mass spectrometry data-on-disk

Römpp et al, Angew. Chem. Int. Ed., 2016

```
spectra(mouse)
```

```
## An object of class 'sparse_matc'
##   <2291 row, 34840 column> sparse matrix
##     datamode: numeric
##     2.3 MB real memory:
##     815 MB virtual memory: keys, values
##     67916471 non-zero elements
##     85% density
```

'Processed' spectra are stored in sparse format

sparse_mat() allows on-disk sparse matrices

```
atomdata(spectra(mouse))
```

```
## $keys
## An object of class 'matter_list'
##   <34840 length> list
##     sources: 1
##     datamode: numeric numeric numeric numeric numeric numeric [and 34834 more]
##     1.1 MB real memory
##     543.3 MB virtual memory
##
## $values
## An object of class 'matter_list'
##   <34840 length> list
##     sources: 1
##     datamode: numeric numeric numeric numeric numeric numeric [and 34834 more]
##     1.1 MB real memory
##     271.7 MB virtual memory
```

EXAMPLE: 'PROCESSED' IMZML
sparse mass spectrometry data-on-disk

Römpp et al, Angew. Chem. Int. Ed., 2016

Different spectra may have different m/z measurements

First mass spectrum:

```
head(atomdata(spectra(mouse))$keys[[1]]) # m/z values
```

```
## [1] 404.0927 404.0958 404.0989 404.1019 404.1050 404.1081
```

```
head(atomdata(spectra(mouse))$values[[1]]) # intensities
```

```
## [1]   0.0000   0.0000   0.0000   0.0000 196.4097 272.0302
```

Data are binned on-the-fly when read into memory

Second mass spectrum:

```
head(atomdata(spectra(mouse))$keys[[2]]) # m/z values
```

```
## [1] 409.9243 409.9274 409.9306 409.9337 409.9368 409.9400
```

```
head(atomdata(spectra(mouse))$values[[2]]) # intensities
```

```
## [1]   0.0000   0.0000   0.0000   0.0000 179.3817 240.8613
```

EXAMPLE: FASTQ EXTENSION
experimental proof-of-concept for text data

- *matter* supports on-disk character vectors via *matter_str()*
- Can we use this functionality to read genomic data?
- Let's create an S4 class for on-disk Fastq data

```
setClass("Fastq", slots=c(
    id = "matter_str",
    sread = "matter_str",
    quality = "matter_str"))

setGeneric("id", function(x) standardGeneric("id"))
setGeneric("sread", function(object, ...) standardGeneric("sread"))
setGeneric("quality", function(object, ...) standardGeneric("quality"))

setMethod("id", "Fastq", function(x) x@id)
setMethod("sread", "Fastq", function(object) object@sread)
setMethod("quality", "Fastq", function(object) object@quality)
```

Need to parse the full data first to identify line breaks

```
parseFastq <- function(file) {
    length <- file.info(file)$size
    raw <- matter_vec(paths=file, length=length, datamode="raw")
    newlines <- which(raw == charToRaw('\n')) # parses the file in chunks
    if ( newlines[length(newlines)] == length )
        newlines <- newlines[-length(newlines)]
    byte_start <- c(0L, newlines)
    byte_end <- c(newlines, length) - 1L # don't include the '\n'
    line_offset <- byte_start
    line_extent <- byte_end - byte_start
    id <- matter_str(paths=file,
        offset=1L + line_offset[c(TRUE,FALSE,FALSE,FALSE)], # skip the '@'
        extent=line_extent[c(TRUE,FALSE,FALSE,FALSE)] - 1L) # adjust for '@'
    sread <- matter_str(paths=file,
        offset=line_offset[c(FALSE,TRUE,FALSE,FALSE)],
        extent=line_extent[c(FALSE,TRUE,FALSE,FALSE)])
    quality <- matter_str(paths=file,
        offset=line_offset[c(FALSE,FALSE,FALSE,TRUE)],
        extent=line_extent[c(FALSE,FALSE,FALSE,TRUE)])
    new("Fastq", id=id, sread=sread, quality=quality)
}
```

*the version of which() that will be executed here is implemented by *matter* and processes the file in chunks

Parse and attach the on-disk Fastq data

```
path <- "~/Documents/Datasets/Test-Data/fastq-example/ERR127302_1_subset.fastq"
fq1 <- parseFastq(path)
sread(fq1)
```

```
## An object of class 'matter_str'
##   <20000 length> string
##     sources: 1
##     datamode: raw
##     252 KB real memory
##     1.4 MB virtual memory
##     encoding: unknown
```

```
head(sread(fq1))
```

```
## [1] "GTCTGCTGTATCTGTGTCGGCTGTCTCGCGGGACATGAAGTCAATGAAGGCCTGGAATGTCACTACCCCCAG"
## [2] "CTAGGGCAATCTTTGCAGCAATGAATGCCAATGGGTAGCCAGTGGCTTTTGAGGCCAGAGCAGACCTTCGGG"
## [3] "TGGGCTGTTCCTTCTCACTGTGGCCTGACTAAAACCCAGTGGCATTAAGAAAGAGTCACGTTTCCCAAGTCT"
## [4] "CTCATCCACACCTTTGGTCTTGATGGCTGTTTCAATGTTCAAAGCATCCCGCTCAGCATCAAAGTTAGTATA"
## [5] "GTTTGGATATATGGAGGATGGGGATTATTGCTAGGATGAGGATGGATAGTAATAGGGCAAGGACGCCTCCTA"
## [6] "CCACGCTGGGGGCAGCTGGCCTCCGCCTCCGCCCACGCACCAGCAGCCGCCTCCATGCCCTAGGTTGGGCCC"
```

PROBLEMS

with using *matter* for genomics and text data

- Takes a long time to parse text file
  - Try larger chunk sizes via `chunksize()` accessor?
  - Worth it to save memory?
- Requires many lines to be the same length
  - Not a strict requirement, but memory blows up otherwise
  - Atoms are compressed via delta-run-length-encoding
  - Varied line lengths can yield incompressible metadata
- Longer line lengths are better
  - Each line is a separate atom, so short lines means too many atoms
  - Unless easily compressed, metadata for each atom takes up memory
  - No guaranteed that on-disk data size is larger than in-memory metadata
- Similar problems for most types of text data
- Can these issues be solved?

FUTURE DIRECTIONS

for *matter*

- Better support for parallelization
  - *matter* objects can be passed easily to other R sessions for parallel read/write
  - Currently up to the user how to do this and avoid read/write problems
  - Add parallelized `lapply(), sapply(), apply()`, etc.
  - Should `mean()`, etc. be parallelized? (Difficult to say…)
- Better support for on-disk data frames
  - Current support is somewhat minimal
  - Not many ways to manipulate *matter_df* objects (`rbind(), cbind()`, etc.)
  - How do people use on-disk data frames?
  - How to integrate existing syntax? What does [[return?
- Expose C++ backend via Rcpp
- Add more support for logical operations
  - All `Compare` operators currently supported
  - Need to `Logic` operators `&` and `|`
  - Add more functions like `which()`, `%in%`, etc.
- Add support for remote data sources?

THANK YOU

Northeastern

Olga Vitek

Purdue

April Harry

Graham Cooks and lab

Livia Eberlin

Stanford

Mark Stolowitz

Parag Mallick

- *Cardinal* : A mass spectrometry imaging toolbox for statistical analysis
  - Available on Bioconductor
  - Website: *www.cardinalmsi.org*
- *matter* : Rapid prototyping with larger-than-memory data on disk
  - Available Bioconductor
  - Also available at *github.com/kuwisdelu/matter*

This work was supported by the NSF Graduate Research Fellowship Program and by the Northeastern University Future Faculty Fellowship