DelayedArray: a tibble for arrays

Peter Hickey


@PeteHaitch

~~Department of Biostatistics, Johns Hopkins Bloomberg School of Public Health~~

Walter and Eliza Hall Institute of Medical Research

Slides: www.bit.ly/useR2018

Why I’m here

Most of what I’m presenting is the work of **Hervé Pagès** (@hpages)

Why I'm here

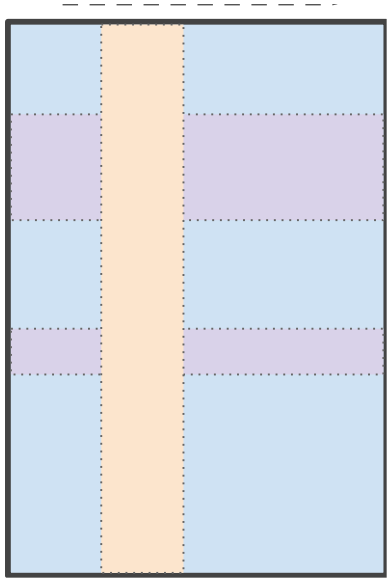
Most of what I'm presenting is the work of **Hervé Pagès** (@hpages)

I'm an early adopter of the DelayedArray framework, using it to analyse large datasets at the cutting edge of high-throughput biology.

I'm a developer of packages (**bsseq, minfi, DelayedMatrixStats**) that use and extend the DelayedArray framework.

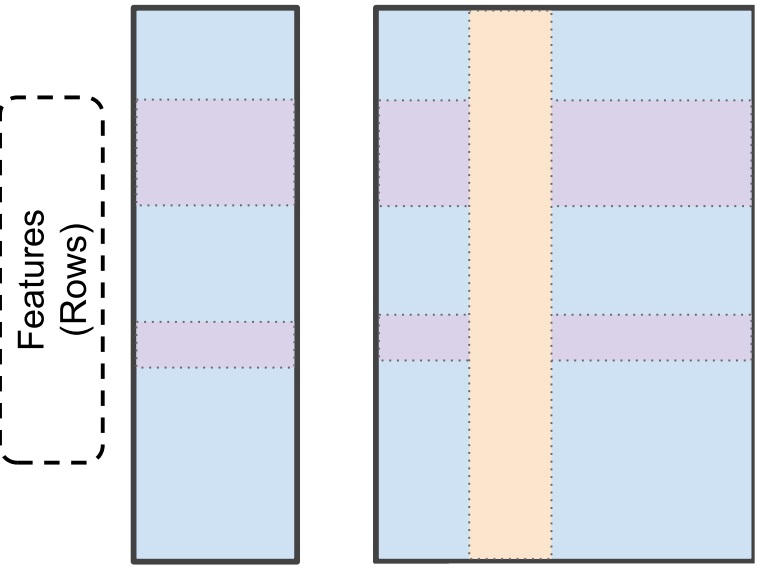
SummarizedExperiment

A core Bioconductor data structure used to store rectangular matrices of experimental results



SummarizedExperiment

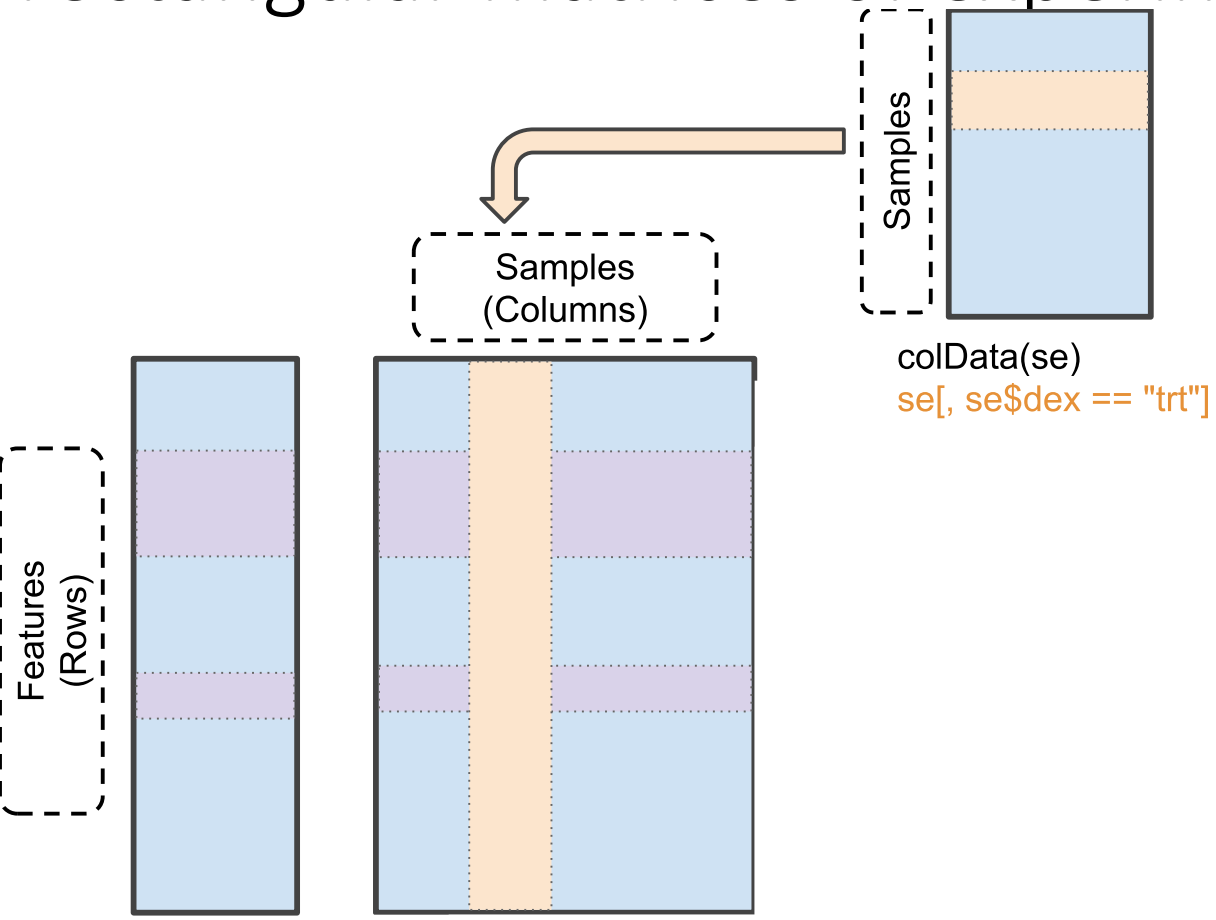
A core Bioconductor data structure used to store rectangular matrices of experimental results



rowRanges(se)
rowData(se)
subsetByOverlaps(se, roi)

SummarizedExperiment

A core Bioconductor data structure used to store rectangular matrices of experimental results

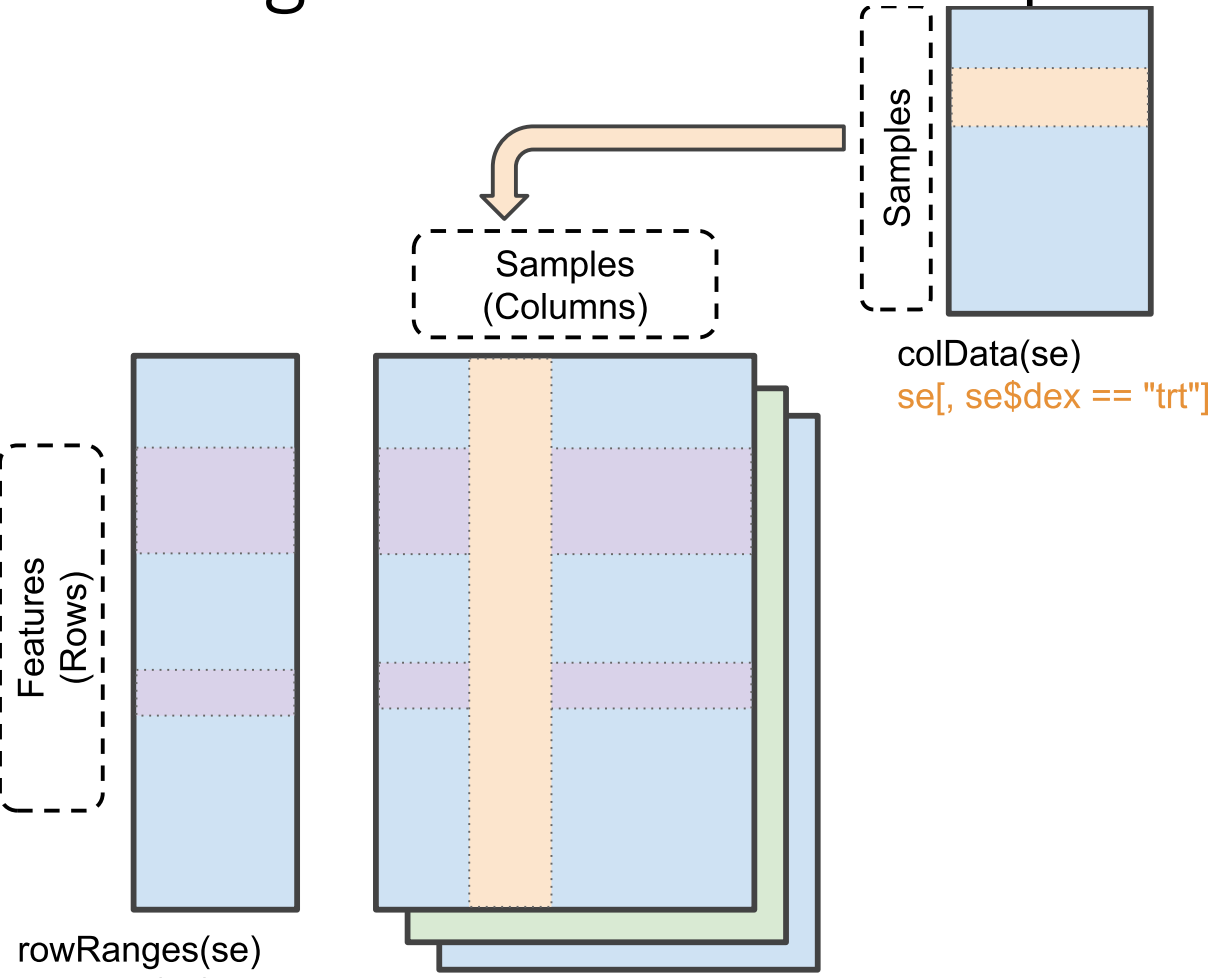


colData(se)
se[, se$dex == "trt"]

rowRanges(se)
rowData(se)
subsetByOverlaps(se, roi)

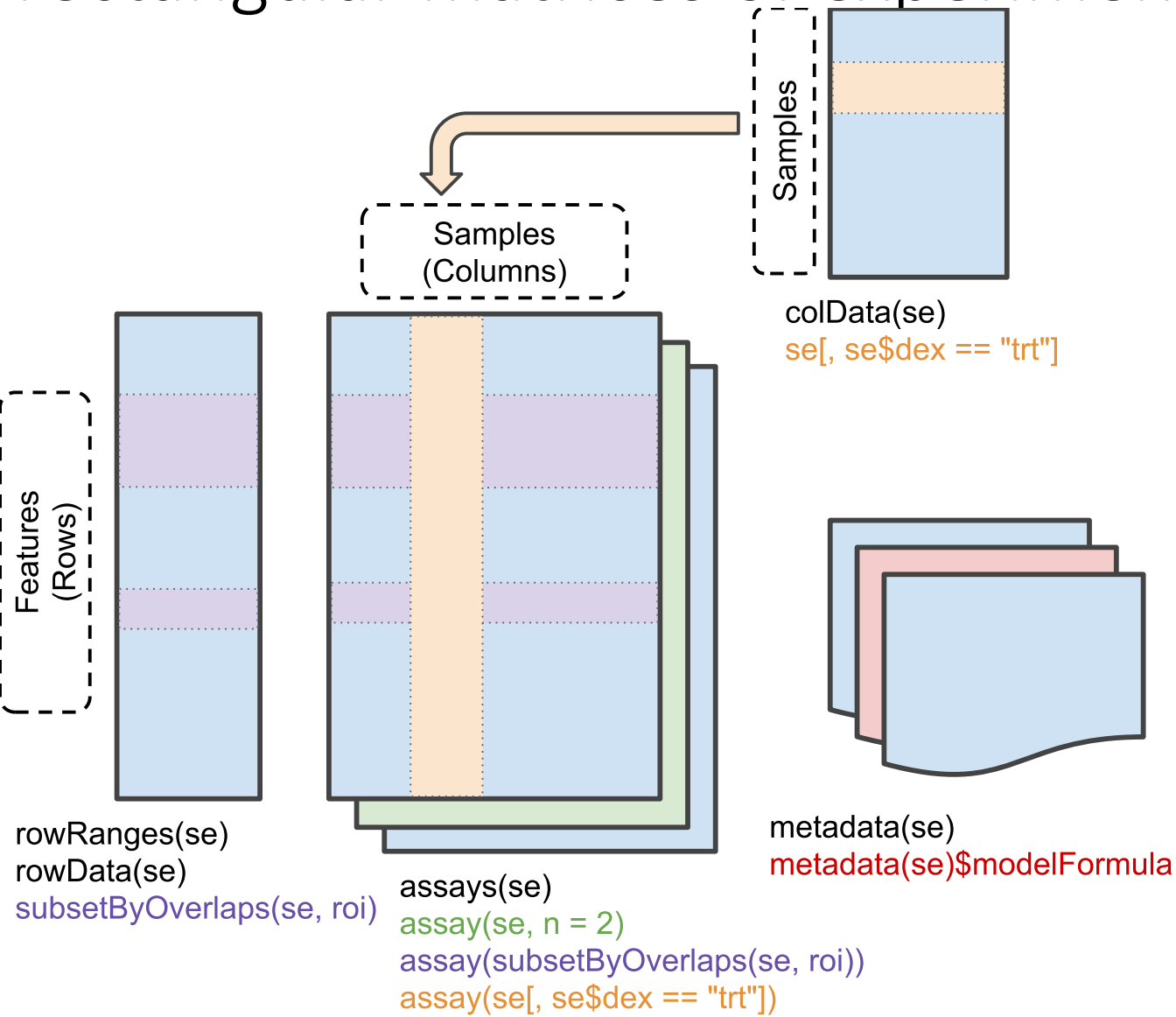
SummarizedExperiment

A core Bioconductor data structure used to store rectangular matrices of experimental results



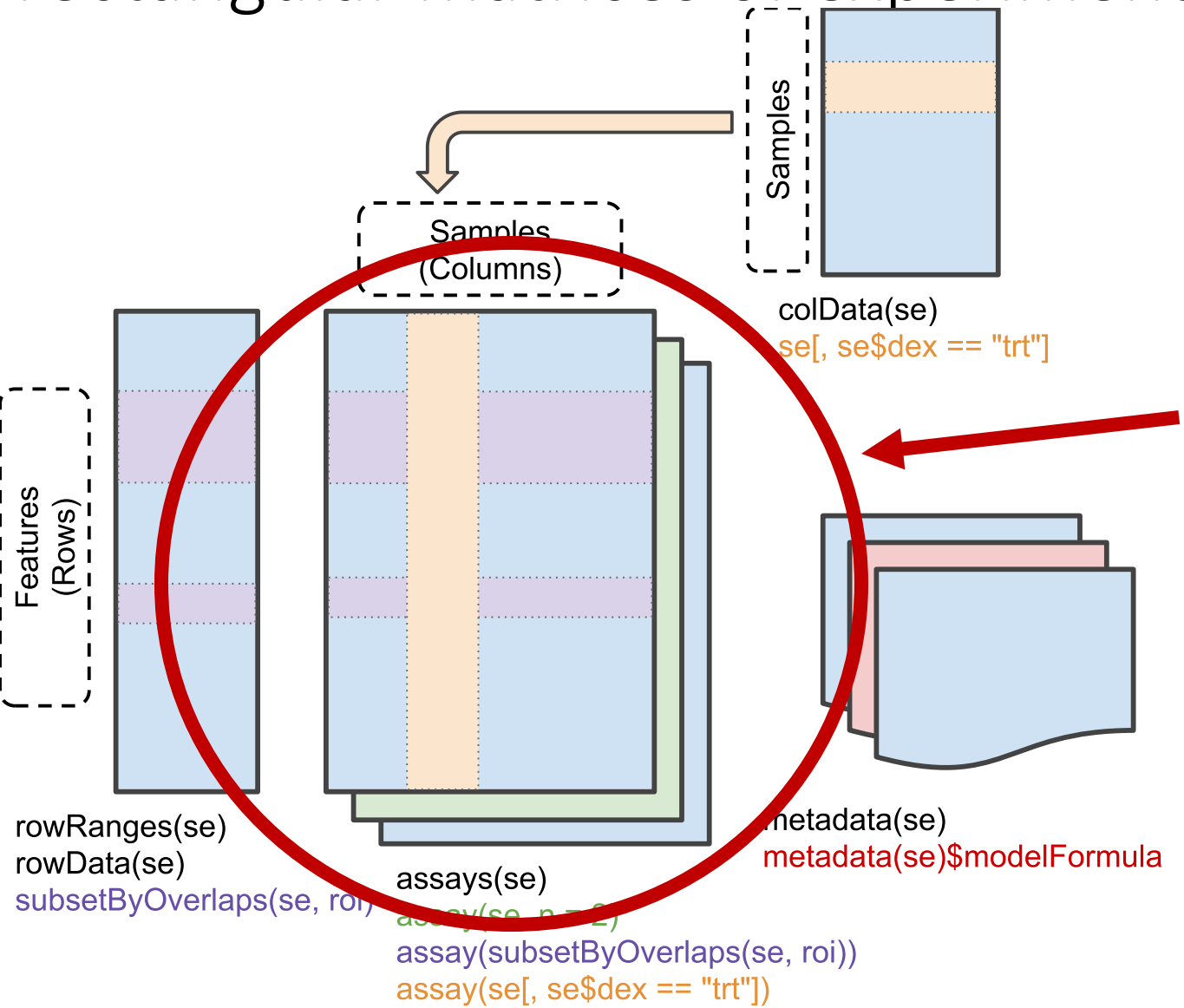
SummarizedExperiment

A core Bioconductor data structure used to store rectangular matrices of experimental results



SummarizedExperiment

A core Bioconductor data structure used to store rectangular matrices of experimental results



Assay data (the measurements)

What I'll be talking about today

Typically, an ordinary R *array*

Why ordinary R arrays?

- ✓Structured (but not tidy™)
- ✓Familiar base R API
- ✓Powerful **matrixStats** API
- ✓Matrix algebra and BLAS/LAPACK-ready
- ✓C/C++-ready
- ✓Conducive to interactive data analysis

But data are getting too big for ordinary R arrays

- **TENxBrainData**
  - Single-cell RNA-seq data for 1.3 million brain cells from mice
  - 1 matrix
  - 27,998 genes (rows)
  - 1,306,127 samples (columns)
  - 146 GB as an ordinary array
- GTEx DNA methylation data
  - Whole genome bisulfite-sequencing (CpG and non-CpG)
  - 3 matrices
  - 31,000,000 – 222,000,000 loci (rows)
  - 183 samples (columns)
  - 91 – 650 GB as ordinary arrays

DelayedArray to the rescue!

- **TENxBrainData**
  - *SummarizedExperiment* is 184 Mb in memory (most of that the `colData`)
- GTEx DNA methylation data
  - *SummarizedExperiment* is 235 Mb in memory (most of that the `rowRanges`)
- How is this done?
  - Assay data live on disk in an HDF5 file that is wrapped in a *DelayedArray*
- Assay data still "look" and "feel" like an ordinary R *array*
  - ✓Structured (but not tidy™)
  - ✓Familiar base R API
  - ✓Powerful matrixStats API (via **DelayedMatrixStats**)
  - ✓Matrix algebra and BLAS/LAPACK-ready (via *block-processing*)
  - ✓C/C++-ready (via **beachmat**)
  - ✓Conducive to interactive data analysis

But what exactly is “DelayedArray”?

- DelayedArray refers to a class, a package, and an extensible framework
- Available as part of Bioconductor
- Developed by **Hervé Pagès**, member of Bioconductor Core Team
- Developed using S4 object oriented system (like most of Bioconductor)

```
install.packages(”BiocManager")
BiocManager::install(”DelayedArray")
```

DelayedArray has analogies to **tibble** and **dplyr**

DelayedArray DESCRIPTION

"Wrapping an array-like object (typically an on-disk object) in a *DelayedArray* object allows one to perform common array operations on it without loading the object in memory.

In order to reduce memory usage and optimize performance, operations on the object are either delayed or executed using a block processing mechanism."

"Note that this also works on in-memory array-like objects like *DataFrame* objects (typically with *Rle* columns), *Matrix* objects, and ordinary arrays and data frames."

tibble and **dplyr** READMEs

"A *tibble*, or *tbl_df*, is a modern reimagining of the *data.frame*, keeping what time has proven to be effective, and throwing out what is not. Tibbles are data.frames that are lazy and surly."

"**dplyr** is designed to abstract over how the data is stored. That means as well as working with local data frames, you can also work with remote database tables, using exactly the same R code."

Why DelayedArray? Why not **rhdf5, hdf5r, matter**, ff, **bigmemory**, **fst**, a database, ...?

- You can still use these!
  - Create new DelayedArray “backends”
- The DelayedArray framework is a powerful abstraction
- **DelayedArray** is developed by the Bioconductor core team
  - Strong integration with core Bioconductor infrastructure

Seeds and backends

- Every *DelayedArray* must have a *seed.*
  - The *seed* stores the actual data.
  - Can be in-memory, locally on-disk, or remotely served.
  - The “seed contract”: `dim()`, `dimnames()`, `extract_array()`.

Seeds and backends

```
library(DelayedArray)
mat <- matrix(rep(1:20, 1:20), ncol = 2)
da_mat <- DelayedArray(seed = mat)
da_mat
#> <105 x 2> DelayedMatrix object of type "integer":
#>          [,1]  [,2]
#>     [1,]    1   15
#>     [2,]    2   15
#>     [3,]    2   15
#>     [4,]    3   15
#>     [5,]    3   15
#> ...           .     .
#> [101,]   14   20
#> [102,]   14   20
#> [103,]   14   20
#> [104,]   14   20
#> [105,]   14   20
```

```
# We can use in-memory seeds.
DelayedArray(seed = Matrix::Matrix(mat))
DelayedArray(seed = as.data.frame(mat))
DelayedArray(seed = tibble::as_tibble(mat))
DelayedArray(seed = S4Vectors::DataFrame(mat))
# A slightly more complex in-memory seed.
RleArray(rle = S4Vectors::Rle(mat), dim = dim(mat))


# We can use on-disk seeds.
library(HDF5Array)
rhdf5::h5ls(hdf5_file)
#>   group        name          otype     dclass       dim
#> 0     /    hdf5_mat   H5I_DATASET    INTEGER 105 x 2
HDF5Array(filepath = hdf5_file, name = "hdf5_mat")


# We can use remotely served seeds.
 library(rhdf5client)
H5S_Array(filepath = “http://host.org", host = hdf5_file)
```

Seeds and backends

- Every *DelayedArray* must have a *seed.*
  - The *seed* stores the actual data.
  - Can be in-memory, locally on-disk, or remotely served.
  - The “seed contract”: `dim()`, `dimnames()`, `extract_array()`.
- A seed is closely related to and tied to a *backend*.
  - **RleArray**
  - **HDF5Array**
  - **rhdf5client**
- What backend should I use?
  - Right now, if you need on-disk data then I’d recommend **HDF5Array**.

Delayed operations

```
# x_h5 is a DelayedArray with an HDF5 seed.
dim(x_h5)
#> [1] 6 2 90354753
# Delayed operations are fast!
system.time(x_h5 + 1L)
#> user system elapsed
#> 0.005 0.000 0.005
x <- as.array(x_h5)
system.time(x + 1L)
#> user system elapsed
#> 4.872 1.761 6.931
```

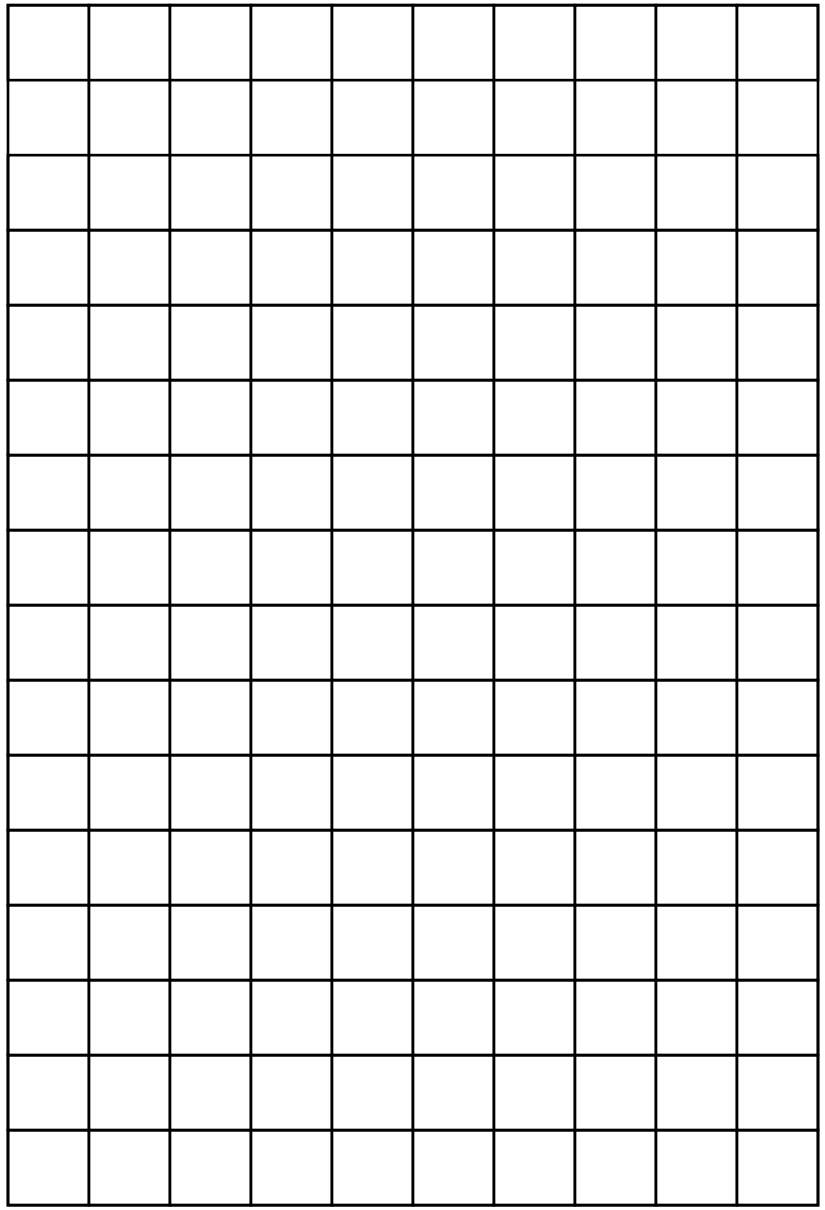
```
showtree(x_h5) # showtree() is kind of like str()
#> 6x2x90354753 integer: HDF5Array object
#> └─ 6x2x90354753 integer: [seed] HDF5ArraySeed object

# They're fast because they don't yet compute anything.
showtree(x_h5 + 1L)
#> 6x2x90354753 integer: DelayedArray object
#> └─ 6x2x90354753 integer: Unary iso op
#>    └─ 6x2x90354753 integer: [seed] HDF5ArraySeed object

showtree(x_h5[1:2, , ])
#> 2x2x90354753 integer: DelayedArray object
#> └─ 2x2x90354753 integer: Subset
#>    └─ 6x2x90354753 integer: [seed] HDF5ArraySeed object

showtree(t(x_h5[1, , ]))
#> 90354753x2 integer: DelayedMatrix object
#> └─ 90354753x2 integer: Aperm (perm=c(3,2))
#>    └─ 1x2x90354753 integer: Subset
#>       └─ 6x2x90354753 integer: [seed] HDF5ArraySeed object
```

Realization

```
# Realize the result to an autogenerated HDF5 file, return as a DelayedArray.
y_h5 <- realize(x_h5 + 1L, BACKEND = "HDF5Array")
# path() tells you the location of the HDF5 seed
path(seed(x_h5))
#> [1]
"/Library/Frameworks/R.framework/Versions/3.5/Resources/library/h5vcData/extdata/example.tally.hfs5"
path(seed(y_h5))
#> [1]
"/private/var/folders/f1/6pjy5xbn0_9_7xwq6l7fj2yc0000gn/T/RtmpRC1xlB/HDF5Array_dump/auto00001.h5"
# Realize the result in memory as an array, return as a DelayedArray.
y <- realize(x_h5 + 1L, BACKEND = NULL)
```

Block-processing

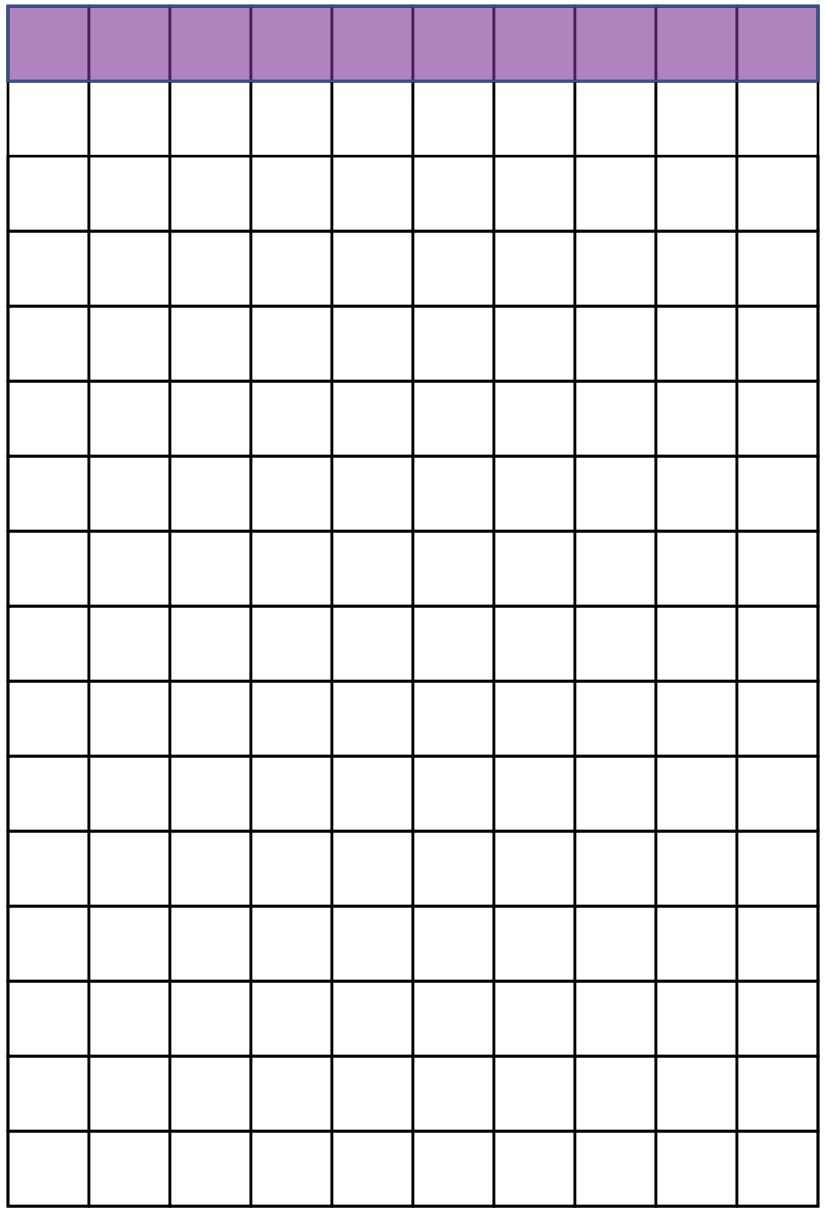
Problem: I need to traverse the array and performing some operation(s) but can only load **n** elements into memory.
The operation(s) could be element-wise or block-wise.
Side note: at the heart of realization.
Side note: **n** is controlled by `getOption("DelayedArray.block.size")`



Each block is a row

E.g., `rowSums()`

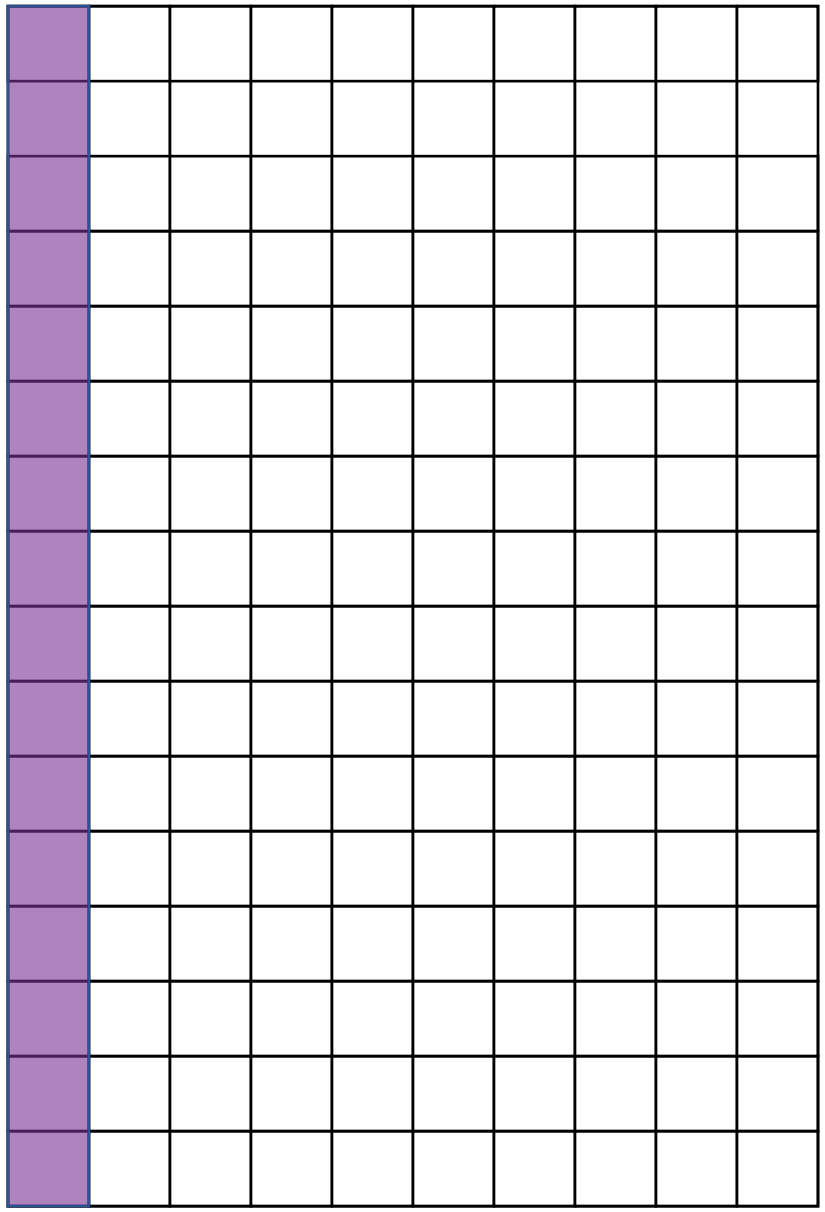
```
RegularArrayGrid(
   refdim = dim(x),
   spacings = c(1L, ncol(x)))
```



Each block is a column

E.g., `colSums()`

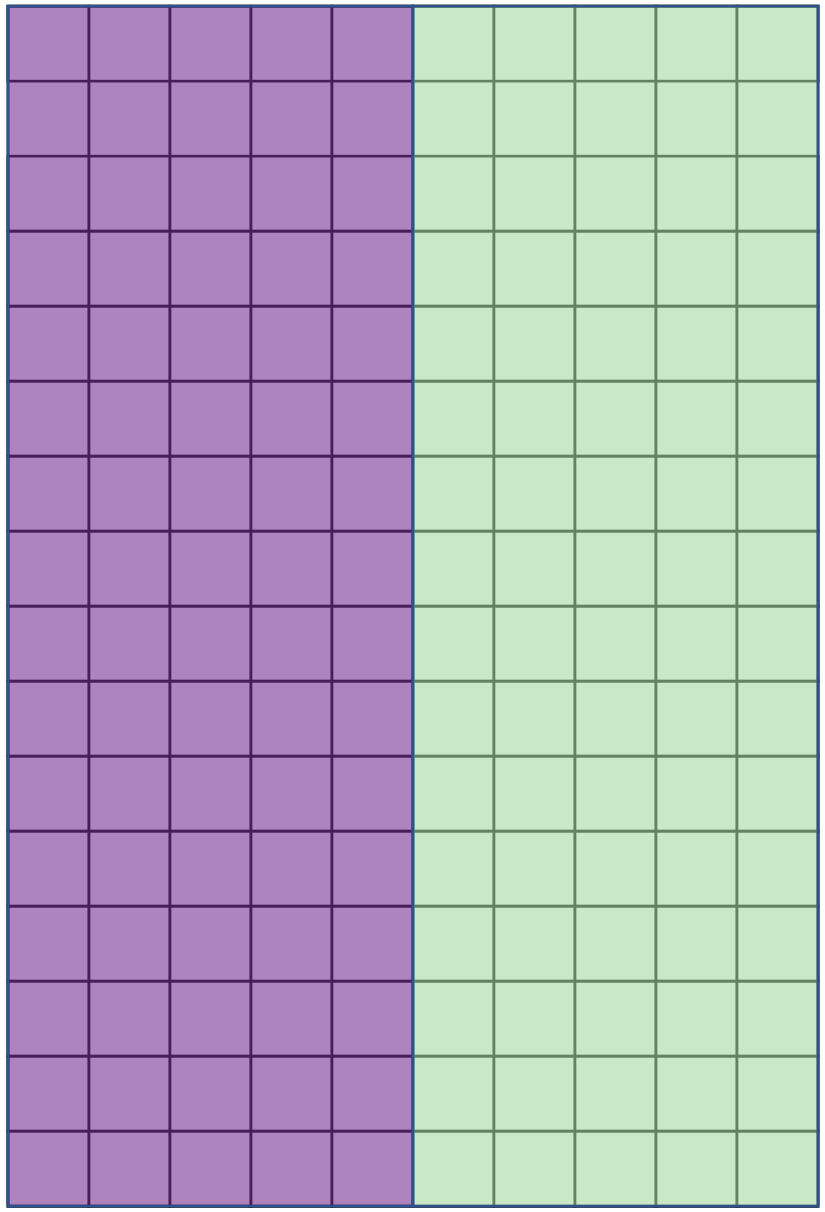
```
RegularArrayGrid(
   refdim = dim(x),
   spacings = c(nrow(x), 1L))
```



Each block is a fixed number of columns

E.g., `colSums()`. More efficient if you can load > 1 columns' worth of data into memory.

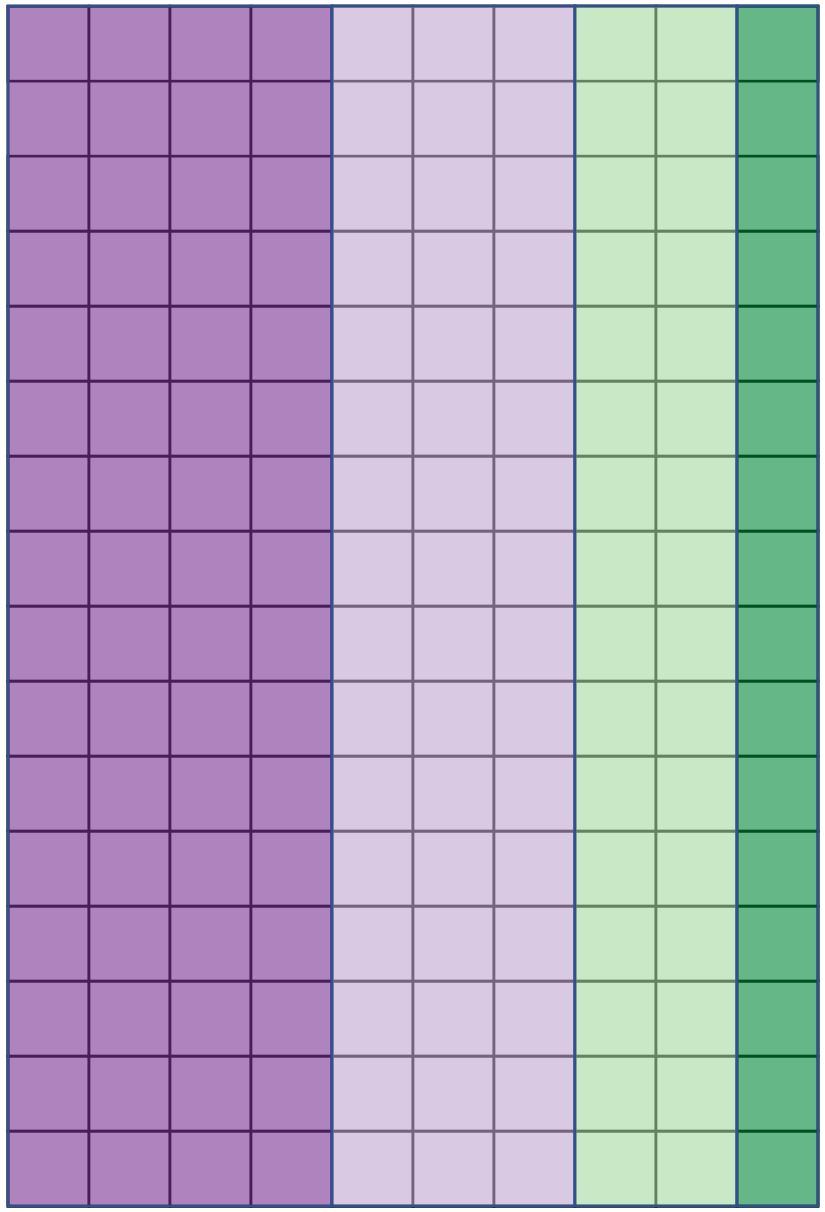
```
RegularArrayGrid(
   refdim = dim(x),
   spacings = c(nrow(x), 5L))
```



Each block is a variable number of columns

E.g., `rowsum()`

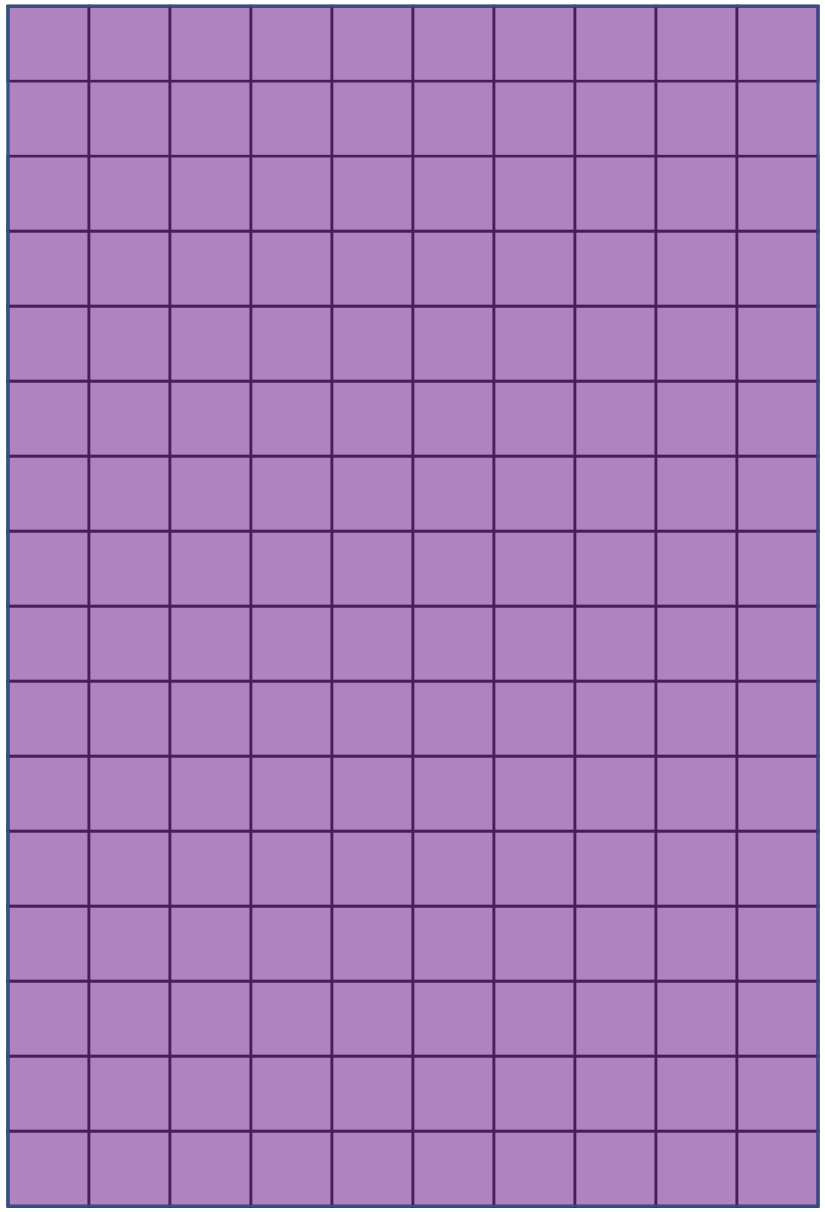
```
ArbitraryArrayGrid(
  tickmarks = list(
    nrow(x),
    c(4L, 7L, 9L, 10L)))
```



Each block is the matrix

You probably don’t want to do this!

```
RegularArrayGrid(
   refdim = dim(x),
   spacings = c(nrow(x), ncol(x))
```



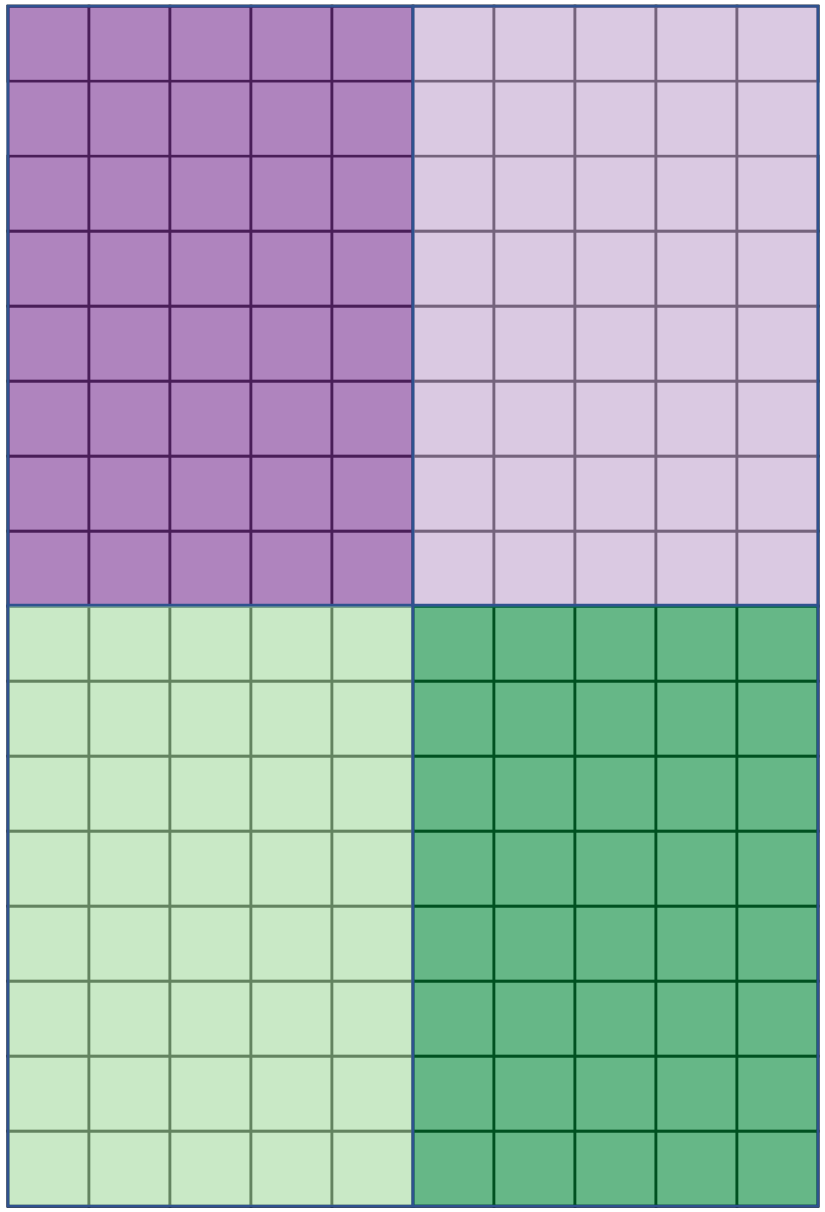
Each block is “optimal”

E.g., when the data are chunked on disk in an HDF5 file.

```
blockGrid(
   x = x,
   block.shape = “hypercube”)
```

`block.shape` can be one of:

- “hypercube”
- “scale”
- “first-dim-grows-first”
- “last-dim-grows-first”



Block-processing pseudocode

```
grid <- blockGrid(x)
for (b in seq_along(grid)) {
  viewport <- grid[[b]]
  block <- read_block(x, viewport)
  FUN(block)
}
```

Block-processing pseudocode

```
library(BiocParallel)
grid <- blockGrid(x)
bplapply(seq_along(grid), function(b) {
  viewport <- grid[[b]]
  block <- read_block(x, viewport)
  FUN(block)
})
```

Block-processing in practice

```
DelayedArray::blockApply(
  x,
  FUN,
  ...,
  grid=NULL,
  BPREDO=list(),
  BPPARAM=bpparam())
```

```
DelayedArray::blockReduce(
  FUN,
  x,
  init,
  BREAKIF=NULL,
  grid=NULL)
```

- These can be used to implement most block-processing algorithms.
- Abstractions for some problems still being worked out. E.g.,
  - Iterating over multiple *DelayedArray* instances.
  - What to do when `FUN()` returns an equally large (or larger) object?
- Can also write to a block with `DelayedArray::write_block()`.

DelayedMatrixStats

- The one slide of this talk about something I done made
- A port of the **matrixStats** API for use with *DelayedMatrix* objects
- Complete coverage of **matrixStats** API (74 methods) via block-processing
- Continual development on seed-aware, optimized methods

API coverage

- ✔ = Implemented in **DelayedMatrixStats**
- ☑ = Implemented in **DelayedArray**
- ❌ : = Not yet impelemented

Method	Block processing	*base::matrix* optimized	*Matrix::Matrix* optimized	*DelayedArray::RleArray* (*SolidRleArraySeed*) optimized	*DelayedArray::RleArray* (*ChunkedRleArraySeed*) optimized	*HDF5Array::HDF5Matrix* optimized	*base::data.frame* optimized	*S4Vectors::DataFrame* optimized
`colAlls()`	✔	✔	❌	❌	❌	❌	❌	❌
`colAnyMissings()`	✔	✔	❌	❌	❌	❌	❌	❌
`colAnyNAs()`	✔	✔	❌	❌	❌	❌	❌	❌
`colAnys()`	✔	✔	❌	❌	❌	❌	❌	❌
`colAvgsPerRowSet()`	✔	✔	❌	❌	❌	❌	❌	❌

beachmat

- Developed by Aaron Lun with Hervé Pagès and Mike Smith.
- Provides a consistent C++ class interface for a variety of commonly used matrix types, including sparse and HDF5-backed matrices.
- Uses Rcpp with `SystemRequirements: C++11`
- Extracting data
  - `get_row()` and get back a *Rcpp::Vector::iterator*
  - `get_col()` and get back a *Rcpp::Vector::iterator*
  - `get()` and get back a *integer*/*double* value
- Writing data
  - `set_row()`
  - `set_col()`
  - `set()`

Summary

- *DelayedArray* is to an *array* as a *tibble* is to a *data.frame*.
- A powerful abstraction for array-like data that may be in-memory, locally stored on-disk, or remotely served.
- The DelayedArray framework may be used directly or extended.
- Combined with an on-disk backend (e.g., HDF5), the DelayedArray framework is enabling the analysis of large genomics data sets.

Acknowledgements

- Hervé Pagès (**DelayedArray, HDF5Array**, the DelayedArray framework)
- Mike Smith, Bernd Fischer, Gregoire Pau, Martin Morgan, Daniel van Twisk (**rhdf5**)
- Aaron Lun, Hervé Pagès, Mike Smith (**beachmat**)
- Samuela Pollack, Shweta Gopaulakrishnan, Vince Carey (**rhdf5client**)
- Qian Liu, Martin Morgan, Hervé Pagès (**GDSArray**)
- Mike Jiang (fellow canary down the coal mine, backend hacker)
- Martin Morgan (Bioconductor Project Lead)
- Kasper Hansen (creator of **bsseq** and **minfi**, postdoc advisor, and tolerator of scenic detours and occasional car crashes)

Links

Papers

Neuronal brain region-specific DNA methylation and chromatin accessibility are associated with neuropsychiatric disease heritability *Rizzardi*, Hickey*, et al.*: https://doi.org/10.1101/120386

beachmat: https://doi.org/10.1371/journal.pcbi.1006135

Workshop: https://bioconductor.github.io/BiocWorkshops/

Presenting at BioC2018 in Toronto on July 25

Material available in 1-2 weeks (well, it had better be …)

Slides: www.bit.ly/useR2018

```
install.packages("BiocManager")
BiocManager::install("DelayedArray")
```

@PeteHaitch