The `ff` package: Handling Large Data Sets in R with Memory Mapped Pages of Binary Flat Files

D. Adler, O. Nenadić, W. Zucchini, C. Gläser
*Institut für Statistik und Ökonometrie,
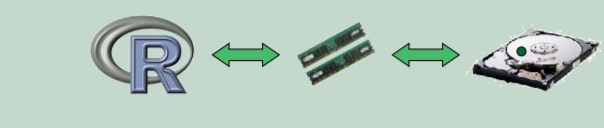Georg-August-Universität Göttingen,
Germany*

Introduction

Limitations on 32-bit platforms:

memory limitations:
- only 2-4 GB memory available

```
> x <- rep(0, 2^31-1)
Error: cannot allocate vector of length 2147483647
```
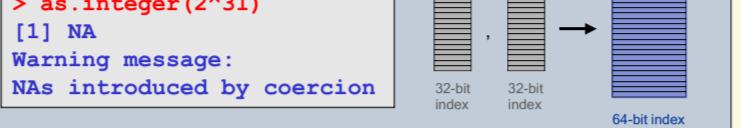
chunk-based data processing

addressing limitations:
- maximum addressable range: $2^{31}-1$

```
> as.integer(2^31-1)
[1] 2147483647
> as.integer(2^31)
[1] NA
Warning message:
NAs introduced by coercion
```

multi-indexing

- The `ff` package is designed to overcome these limitations by implementing a new container type (binary flat files)

- 64-bit platforms also benefit from the `ff` package (i.e. 8 GB practical RAM limit, tweakable address space)

- The package has been ported to Windows, Linux, Mac OS X and FreeBSD; it is available from http://wsopuppenkiste.wiso.uni-goettingen.de/ff

+ Windows™ (logo omitted due to trademark guidelines)

Package overview

The `ff` package comprises two parts, an **R part** and a **C++ part**.

Opening / creating flat files
Controlled by the two core functions `ff` and `ffm`

I/O operations
Controlled by the "`[]`" and the "`[]<-`" operators

Generic functions / methods
Methods for `dim` and `length`; `sample` converted to a generic function
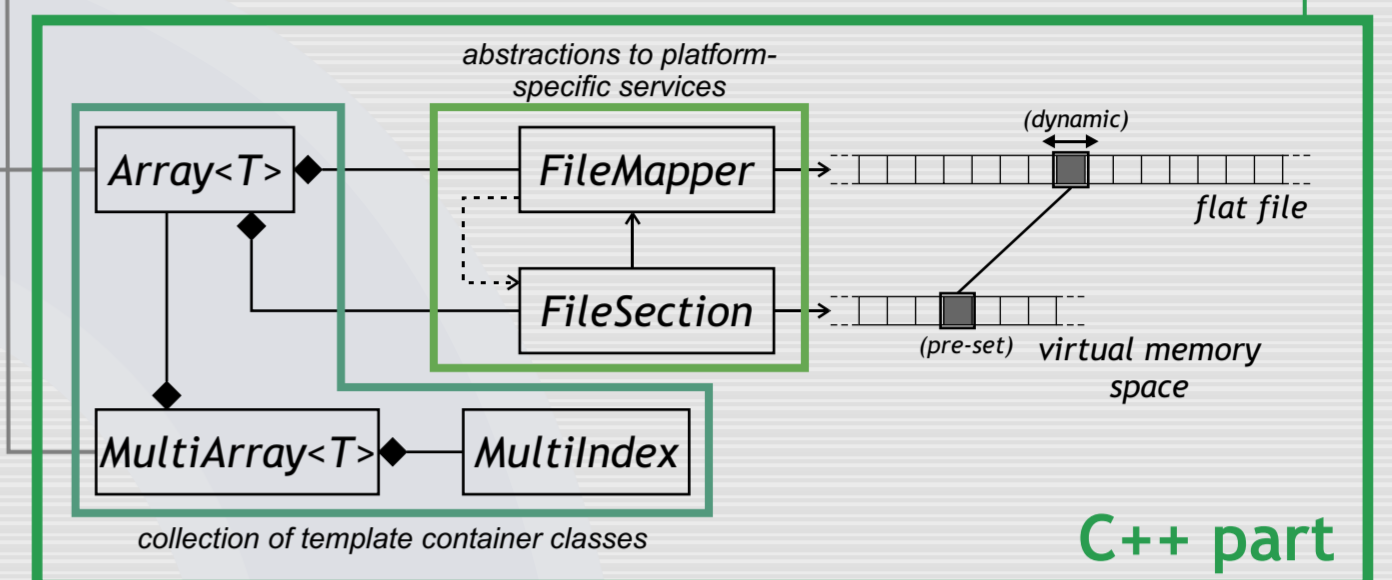
Auxillary functions
E.g. `seqpack` for sequence analysis and `ffm.data.frame` for wrapping `ffm` objects into data frames

R part

The main functions are `ff` and `ffm`, which are used for opening and creating flat files. The syntax for I/O operations is equivalent to native R objects, namely "`[]`" and "`[]<-`" operators. Methods and generic functions are provided.

The C++ part contains two parts, namely *abstractions to platform-specific services* (*FileMapper* and *FileSection* class) and a *collection of template container classes* (*Array*, *MultiArray* and *MultiIndex* for doubles)
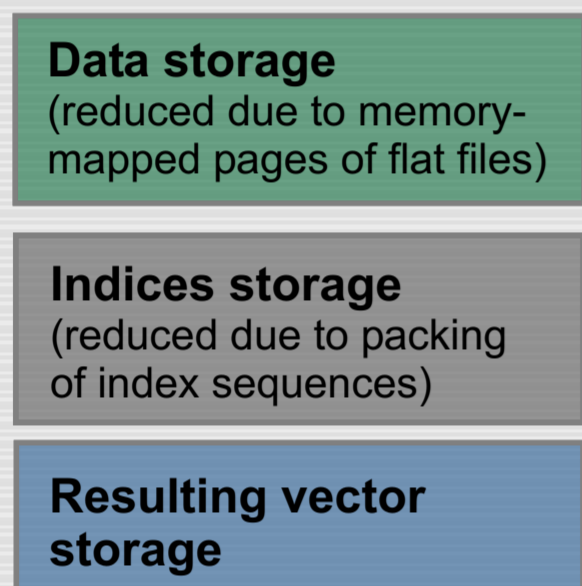
abstractions to platform-specific services

Array<T> — *FileMapper* → (dynamic) flat file
FileSection → (pre-set) virtual memory space

MultiArray<T> — *MultiIndex*

collection of template container classes

C++ part

Comparison between `ff` and standard R objects

How the creation of n values effects the run-time virtual memory address space:
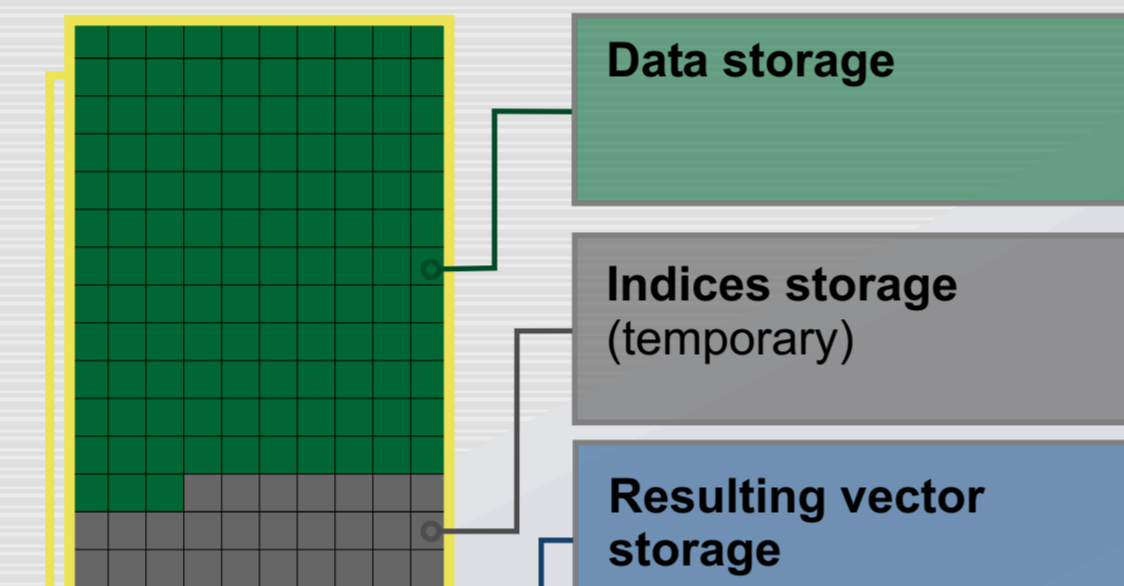
ff object:

```
> ffOb <- ff("foo",8000000)
> aVal <- ffOb[1:2000000]
```

Data storage
(reduced due to memory-mapped pages of flat files)

Indices storage
(reduced due to packing of index sequences)

Resulting vector storage

The amout of memory required by an `ff` object.

native R vector:

```
> rObj <- numeric(8000000)
> aVal <- rObj[1:2000000]
```

Data storage

Indices storage
(temporary)

Resulting vector storage

The amout of memory required by a native R vector object.

Virtual Memory Address Space

☐ 512 kilobytes

Chunk based data processing

The `biglm` package (T. Lumley) provides a facility for fitting generalized linear models to large data sets (i.e. data sets that are larger than memory).
In the examples below we demonstrate on a small data set how `ffm` objects can be used as input for the `bigglm` function.

```
> library("ff")
> m <- ffm("/tmp/foom", dim = c(31,3))
> data("trees")
> m[1:31,1:3] <- trees[1:31,1:3]
```

After creating the `ffm` object `m` the "`trees`" data set (31x3 matrix) is stored in `m`.

```
> library("biglm")
> ffmdf <- ffm.data.frame(m,
+          c("Girth", "Height", "Volume"))
> fg <- log(Volume) ~ log(Girth) + log(Height)
```

The trees data stored in `m`, a `ffm` object, are wrapped into a `ffm` data frame. The model formula is defined in the usual way (c.f. examples for `bigglm`).

```
> mod <- bigglm(fg, data = ffmdf, chunksize = 10,
+          sandwich = TRUE)
> summary(mod)
Large data regression model:
  bigglm(formula = formula, data = datafun, ...)
Sample size = 31
            Coef    (95%    CI)    SE p
(Intercept) -6.632 -8.087 -5.176 0.728 0
log(Girth)   1.983  1.871  2.094 0.056 0
log(Height)  1.117  0.733  1.501 0.192 0
Sandwich (model-robust) standard errors
```
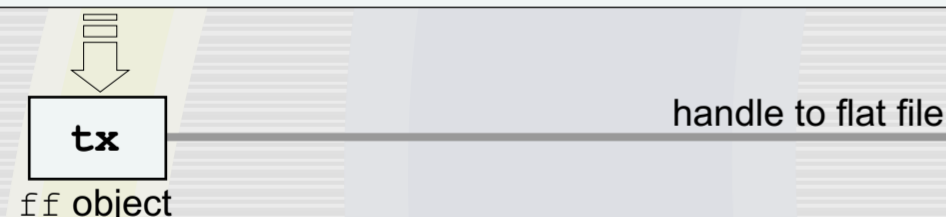
The syntax is the same as in the `bigglm` example. However, the fitting is carried out on a `ffm` object wrapped into a `ffm.data.frame` object.

How `ff` works

An example of what happens behind the scenes:
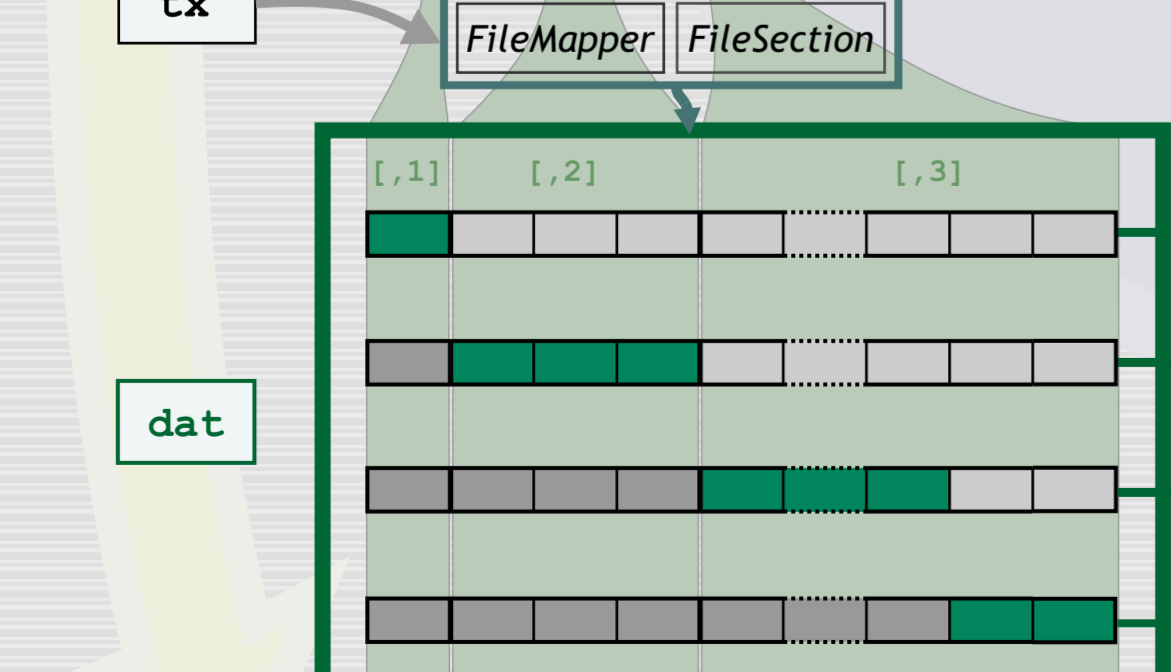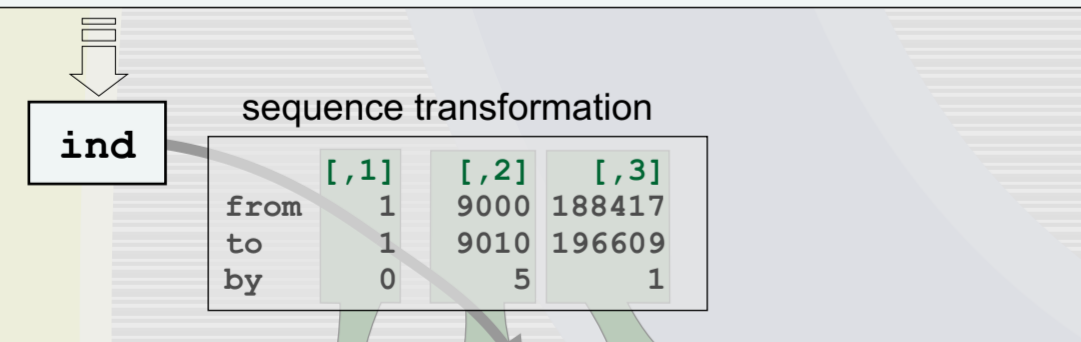
① The first part of the code shows how an existing flat file is "opened": by calling `ff` a handle to a flat file is established

```
> tx <- ff("/tmp/texas_p", pagesize = 64*1024)
```

tx
`ff` object

handle to flat file

② When extracting a subset of the data the commands do not differ from the "standard R procedure". However, the extraction procedure differs substantially

```
> ind <- c(1,9000,9005,9010,188417:196609)
> dat <- tx[ind]
```

ind

sequence transformation

	[,1]	[,2]	[,3]
from	1	9000	188417
to	1	9010	196609
by	0	5	1

tx

FileMapper *FileSection*

	[,1]	[,2]	[,3]

dat

① ② ③ ④

③ The object returned is a standard R object

```
> dat
  [1] 32  0  0 27  0 38 40  0  0  0  0  0 43 49 19
 [16] 30 37 30 47 43 33 44 45 37 53 50 57 38 70 44
 [31]  0  0  0  0  0 30 67 37 37 60 47 70 44 29 50
...
```

page 1
① The requested data item (at index 1) is in the currently active page.

page 2
② The next three requested items are in the next page. The corresponding page is loaded and the items are extracted

page 24
③ The third block of requested items is split over two pages. The first page is mapped into main memory and the items are extracted

page 25
④ Then the next page containing the remaining entries is mapped and the corresponding items are extracted.

a flat file in native binary format

☐ data entry (8 Bytes)   ☐ disk space not used

☐ requested data   ☐ page (here: 64 KB)

Future Work

- Support for additional data types besides doubles

- Incorporating additional storage and caching policies (multi-threaded, non-blocking I/O, prefetching)

- I/O optimization (effect of the page size on performance)

- Implementing chunk-based algorithms and methods (`sum`, `mean`, `var`, `min`, `max`, ...)

- Feedback and suggestions are welcome:
  <dadler,onenadi>@uni-goettingen.de