

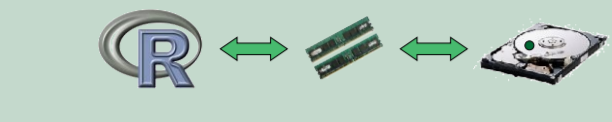
Introduction

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
```

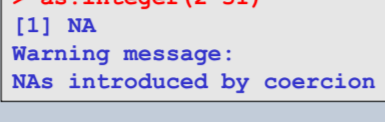


chunk-based data processing

addressing limitations:

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

```
> as.integer(2^31-1)
[1] 2147483647
> as.integer(2^31)
[1] NA
Warning message:
NA& 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>



Package overview

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

Auxiliary functions

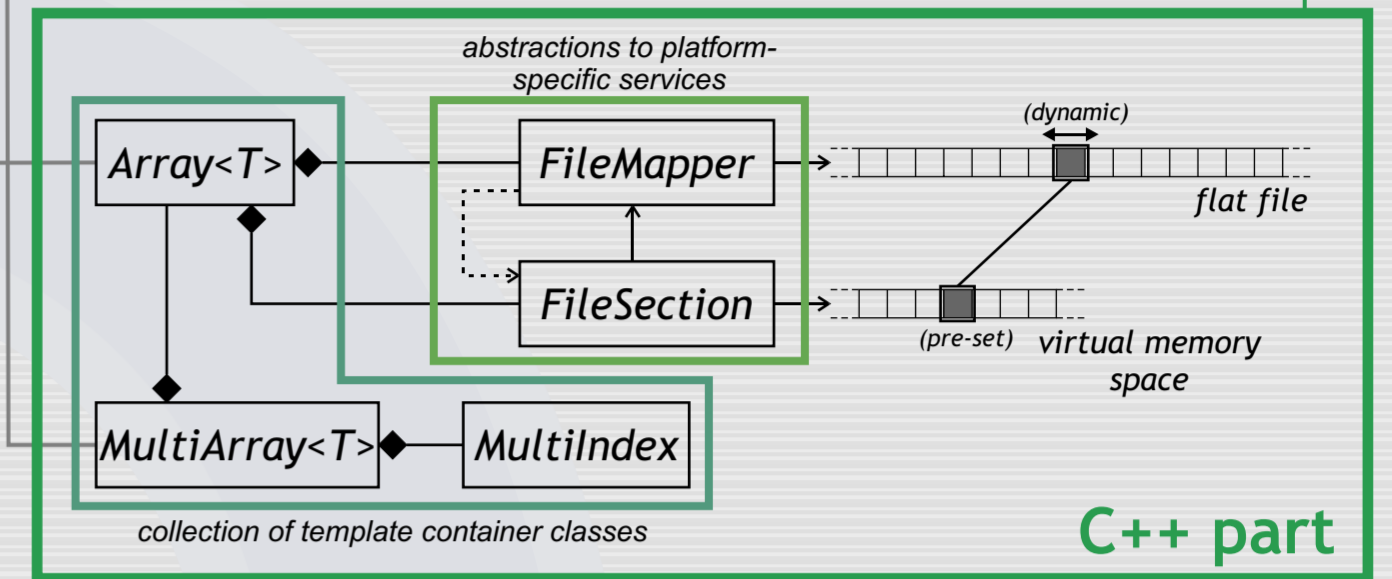
E.g. **seqpack** for sequence analysis and **ffm.data.frame** for wrapping **ffm** objects into data frames

R part

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

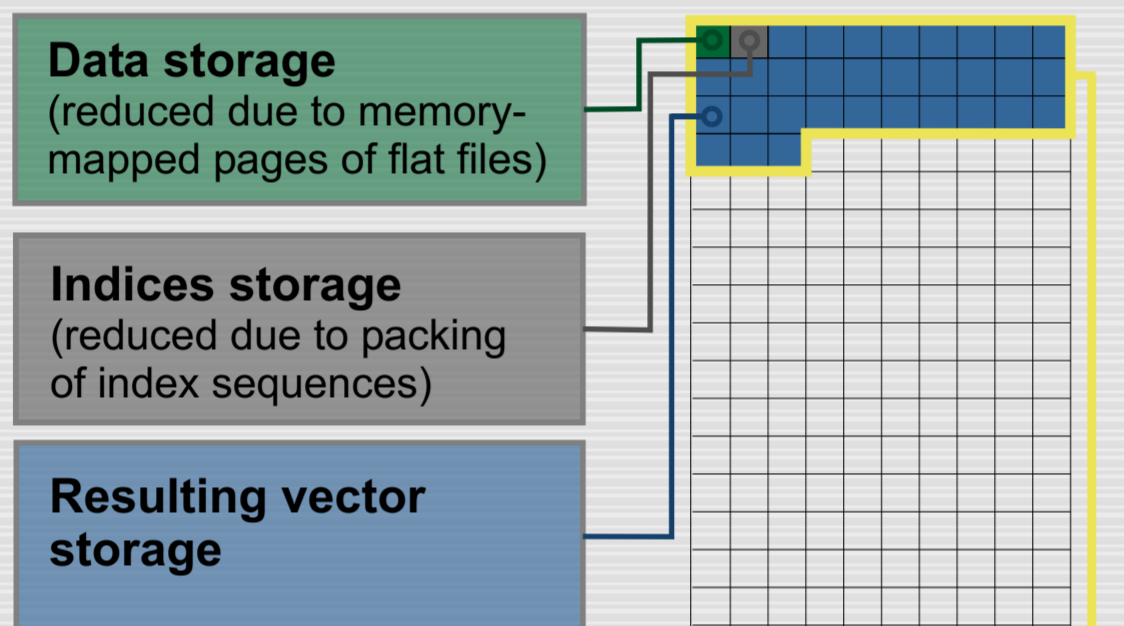


Comparison between ff and standard R objects

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

ff object:

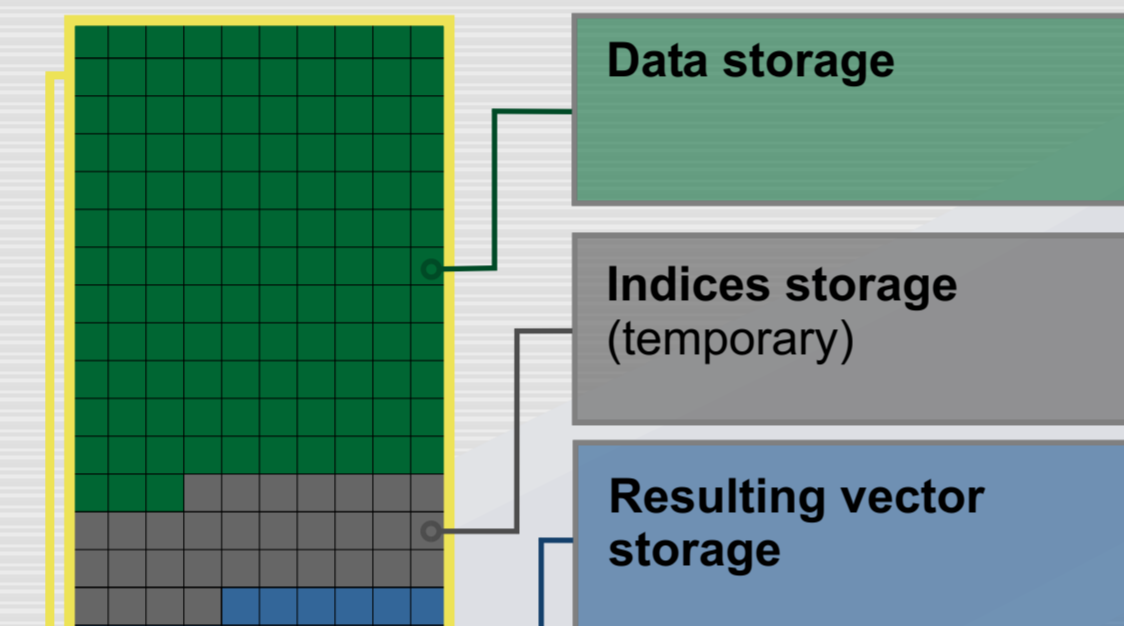
```
> ffObj <- ff("foo", 8000000)
> aVal <- ffObj[1:2000000]
```



The amount of memory required by an **ff** object.

native R vector:

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



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

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:

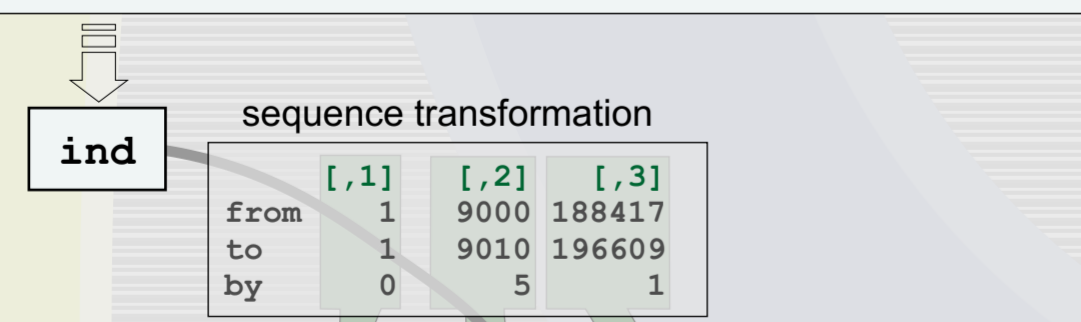
- The first part of the code shows how an existing flat file is "opened": by calling **ff** a handle to a flat file is established

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



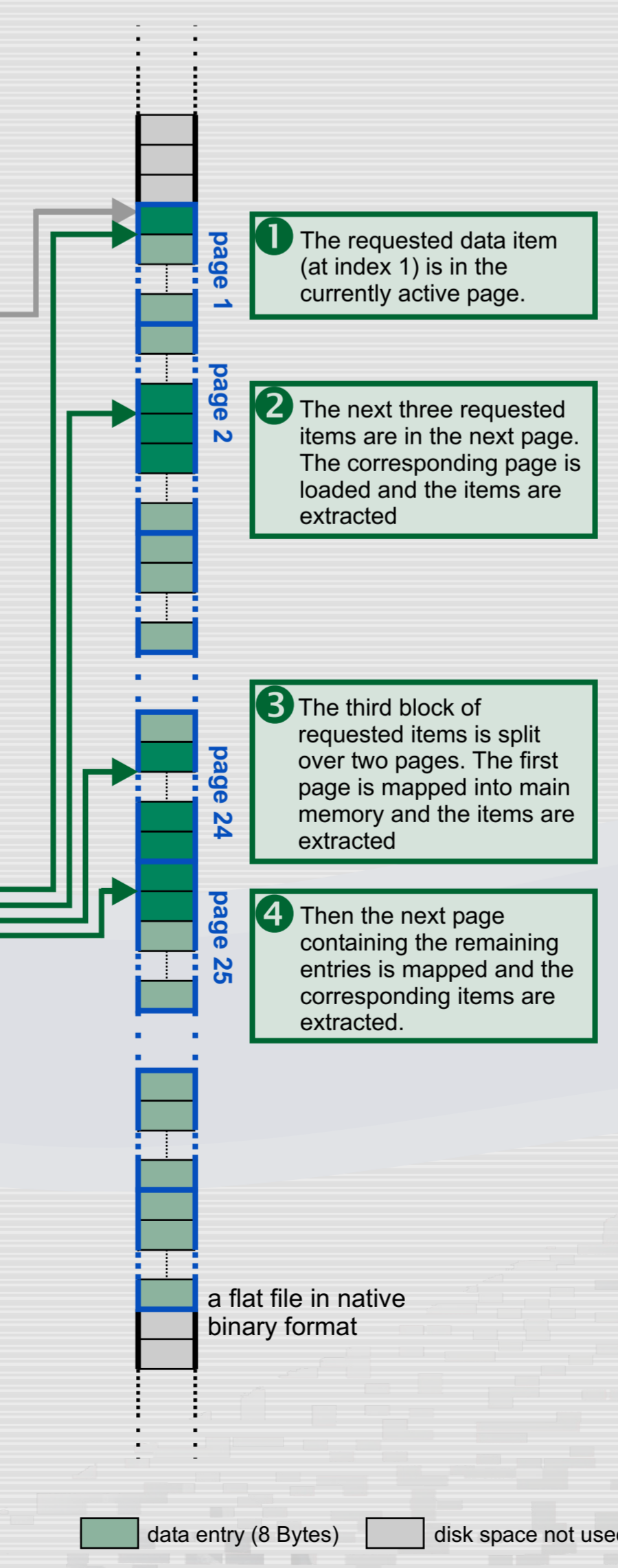
- When extracting a subset of the data the commands do not differ from the "standard R procedure". However, the extraction procedure differs substantially

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



- 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
...
```



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>](mailto:dadler,onenadi@uni-goettingen.de)